CS Space

Клуб технологий и науки

Видеокарты: что они могут? Могут ли они хоть что-то? Давайте выясним!

@UnicornGlade

@PolarNick239

polarnick239@gmail.com

Николай Полярный

План лекции

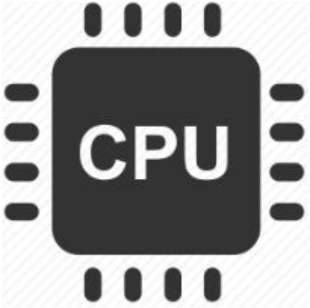
1) **Вычисления**:
   shared instruction pointer, code divergence
2) **Работа с памятью**:
   hyper-threading, latency hiding, occupancy, registers pressure/spilling, coalesced memory access pattern, cache lines, local/shared memory
3) **Общая картина ЭВМ-архитектуры**: CPU - RAM - PCI-E - VRAM - GPU
4) **Модель вычислений массового параллелизма**
5) **Профилирование и оптимизация** GPGPU-алгоритма
6) **Примеры:** A+B, максимум по массиву, merge-sort
7) **Матрицы:** транспонирование, умножение, **tensor cores**, **DeepSeek**
8) **Ray Tracing:** real-time BVH, **ray tracing cores**
9) **Выводы:** какие алгоритмы ускоряются на GPU? OpenCL, CUDA или Vulkan?

Глава 1: Вычисления

shared instruction pointer, code divergence

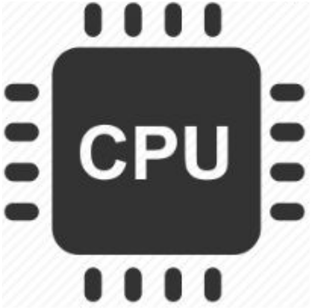
Архитектура

$100 \cdot 10^{\mathbf{9}}$ FLOPS (**F**loating-point operations **p**er **s**econd)

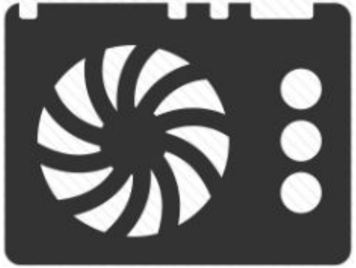


Архитектура

$100 \cdot 10^{9}$ FLOPS (**F**loating-point operations **p**er **s**econd)



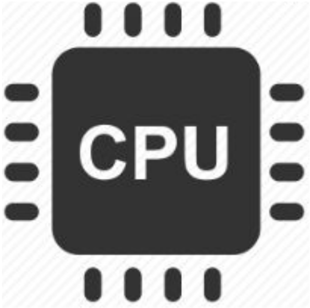
$100 \cdot 10^{12}$ FLOPS (**x1000 раз больше**)



GPU

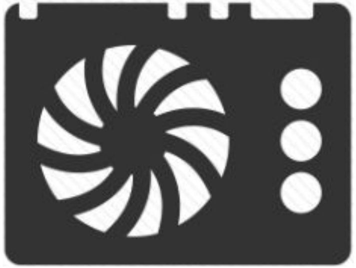
Архитектура

$100 \cdot 10^{9}$ FLOPS



40 GB/s memory bandwidth

$100 \cdot 10^{12}$ FLOPS (**x1000 раз больше**)

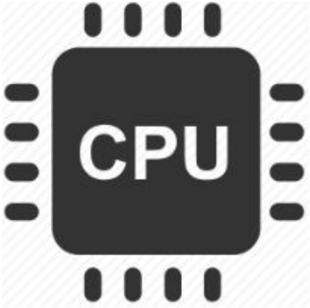


1000 GB/s (**x25 раз больше**)

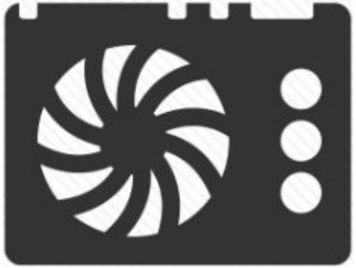
GPU

Архитектура

$100 \cdot 10^{9}$ FLOPS



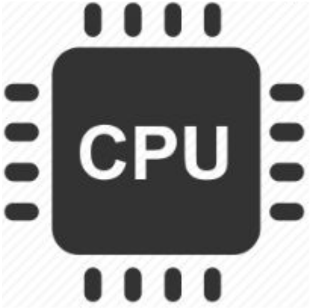
$100 \cdot 10^{12}$ FLOPS



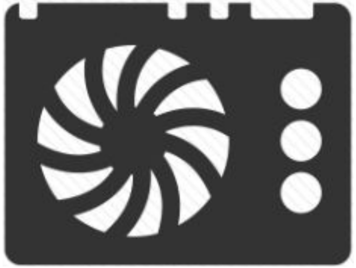
GPU

Архитектура

$100 \cdot 10^9$ FLOPS



$100 \cdot 10^{12}$ FLOPS

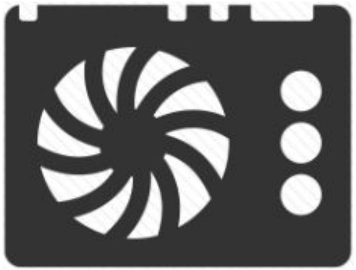


GPU

Архитектура

Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

$100 \cdot 10^{12}$ FLOPS



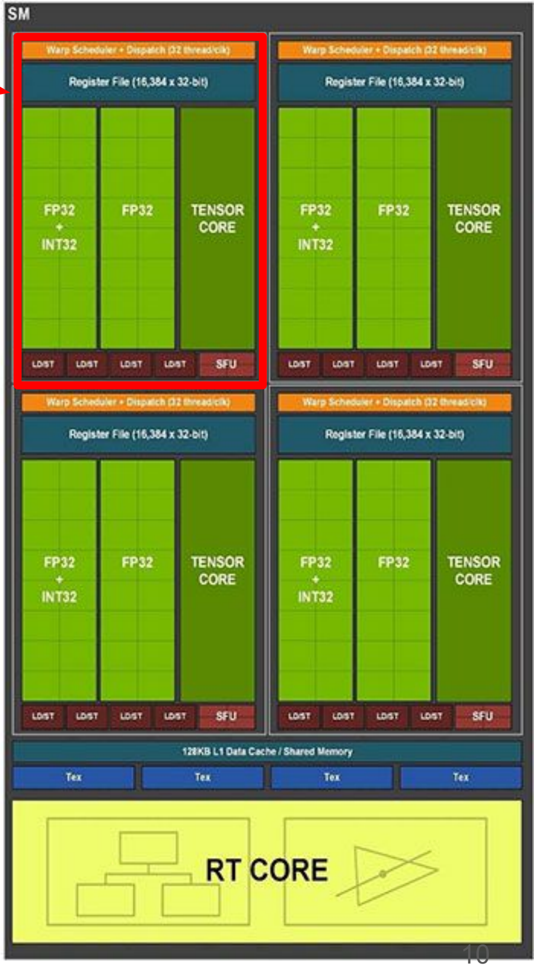
GPU

Архитектура

warp

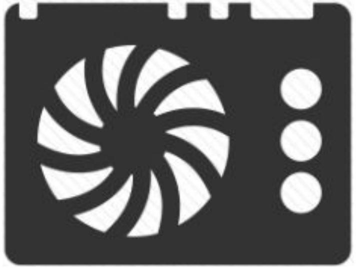
32 слабых медленных
CUDA ядер

SM - Streaming Multiprocessor



Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

$100 \cdot 10^{12}$ FLOPS



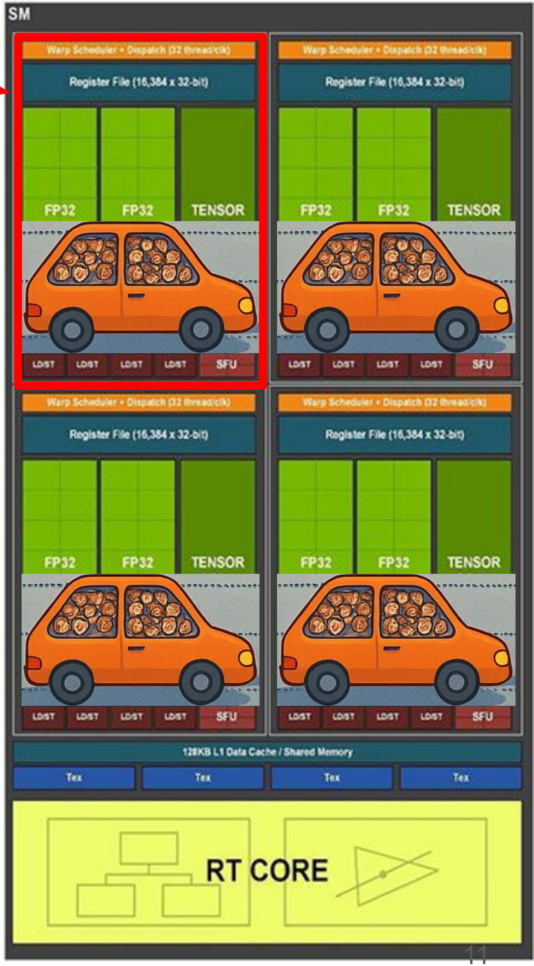
GPU

Архитектура

warp

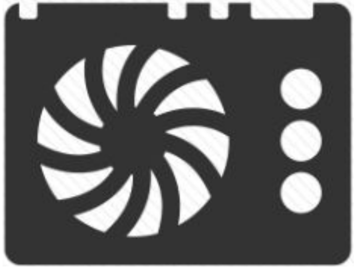
32 слабых медленных
CUDA ядер

SM - Streaming Multiprocessor



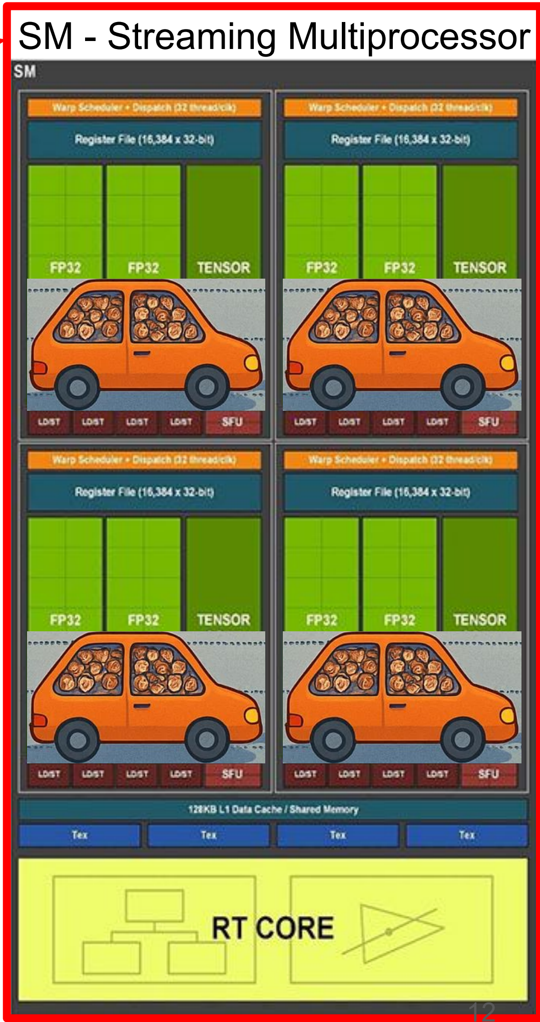
Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

$100 \cdot 10^{12}$ FLOPS



GPU

RTX 3090: **10496 CUDA cores** = 82 SM · 4 warps · 32 ALUs



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

CPU ядро

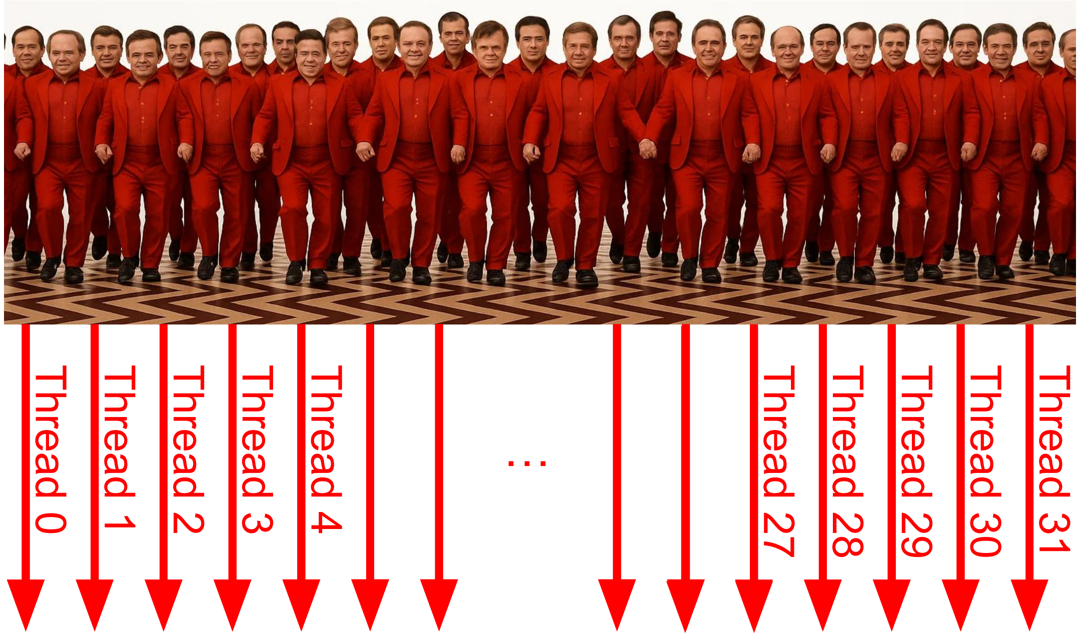
GPU warp

Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
…
Thread 27
Thread 28
Thread 29
Thread 30
Thread 31

Как уместить ТЫСЯЧИ ядер?
На CPU ведь это почему-то невозможно!

Архитектура

GPU warp

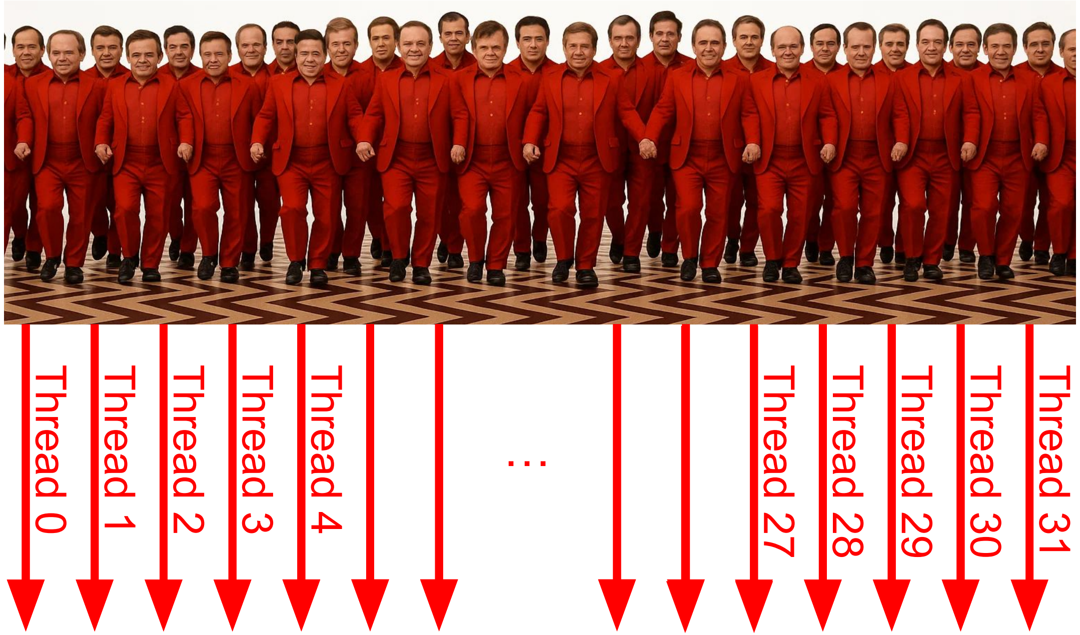



```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID];
} else {
    result = dataB[warpThreadID];
}
```

Один Instruction pointer
на все потоки warp-a!

Архитектура

GPU warp

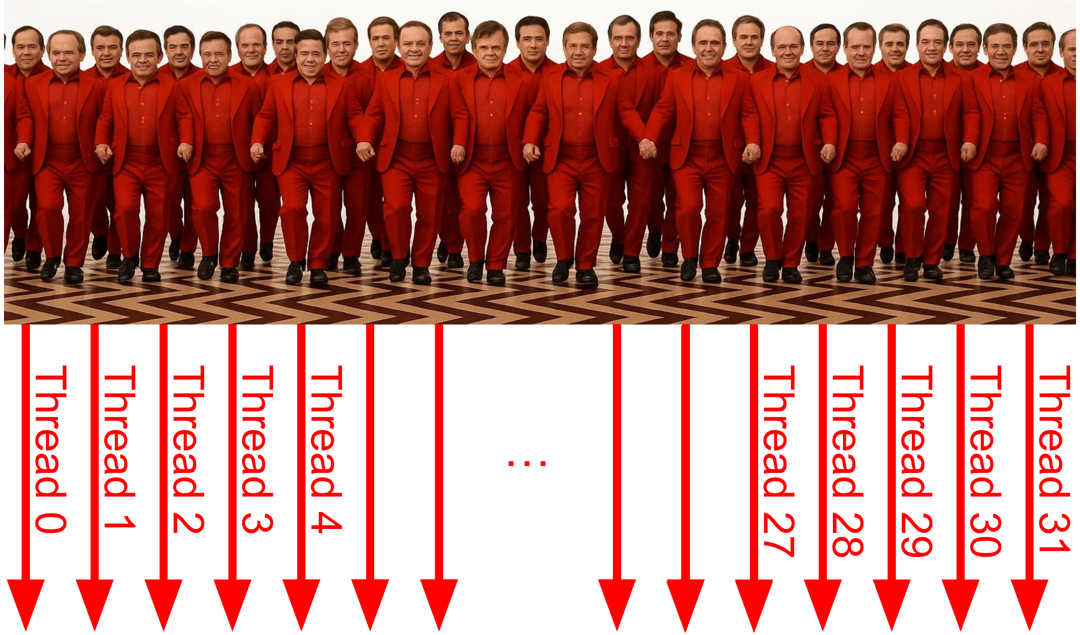



```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID];
} else {
    result = dataB[warpThreadID];
}
```

Один Instruction pointer
на все потоки warp-a!

Архитектура

GPU warp




```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID];
} else {
    result = dataB[warpThreadID];
}
```

Один Instruction pointer
на все потоки warp-a!

Архитектура

GPU warp

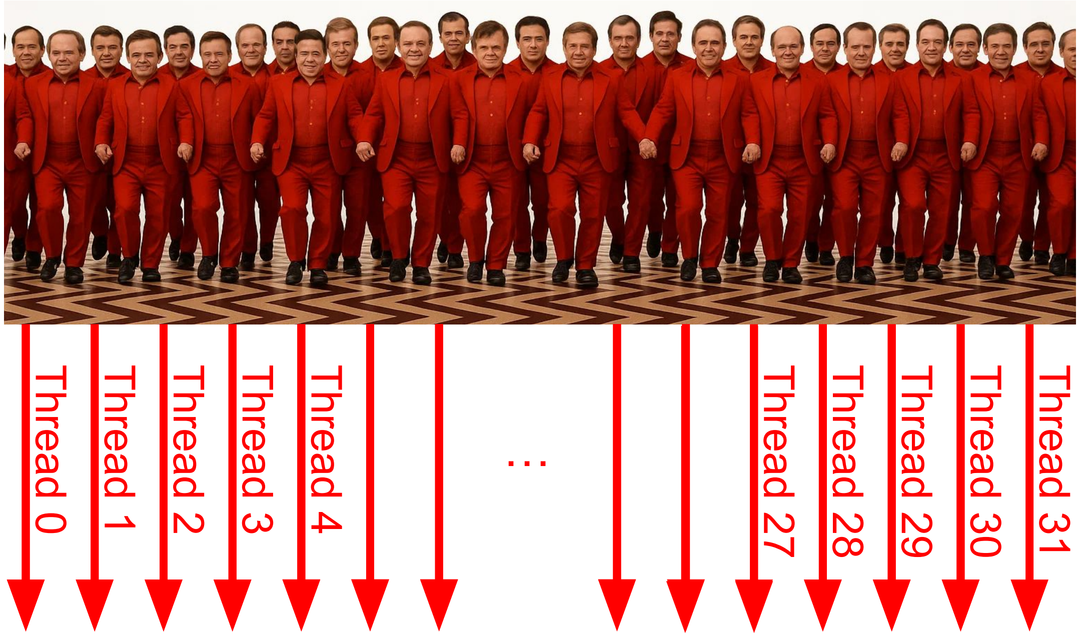
Thread 0 Thread 1 Thread 2 Thread 3 Thread 4 … Thread 27 Thread 28 Thread 29 Thread 30 Thread 31

```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID];
} else {
    result = dataB[warpThreadID];
}
```

Один Instruction pointer
на все потоки warp-а!

Архитектура

GPU warp

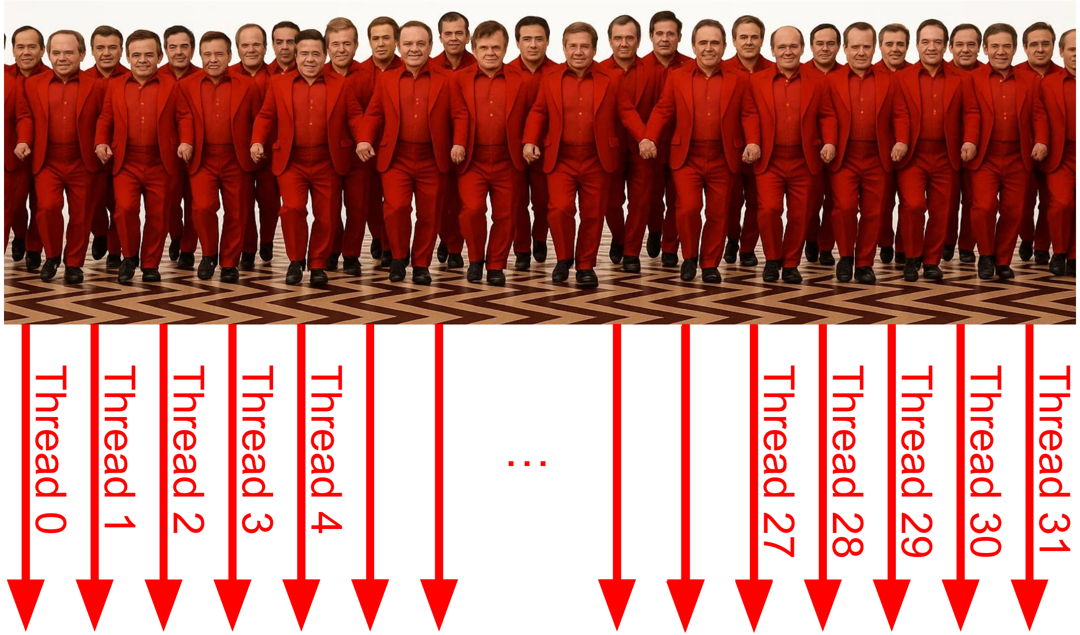


Один Instruction pointer
на все потоки warp-a!

```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID];
} else {
    result = dataB[warpThreadID];
}
```

Архитектура

GPU warp

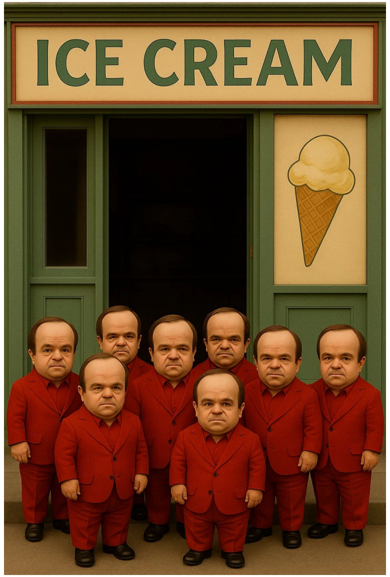


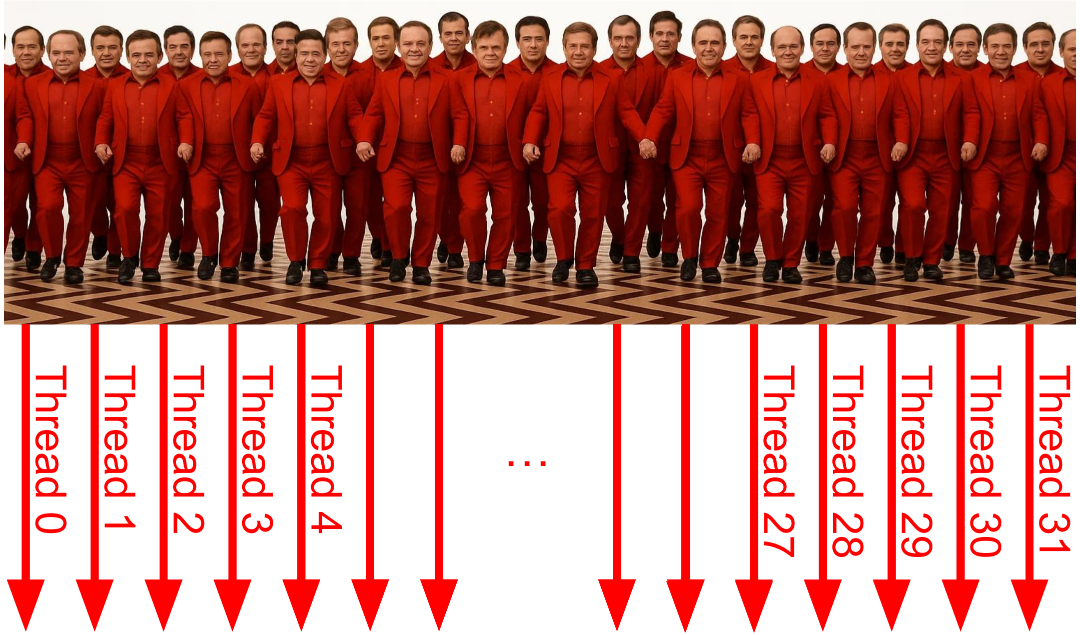
Один Instruction pointer
на все потоки warp-а!

```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID];
} else {
    result = dataB[warpThreadID];
}
```

Архитектура

GPU warp





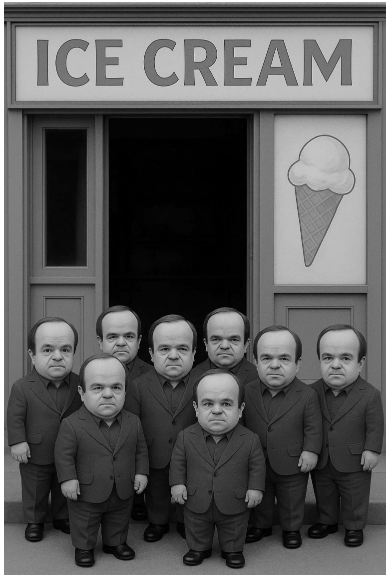

```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID];
} else {
    result = dataB[warpThreadID];
}
```

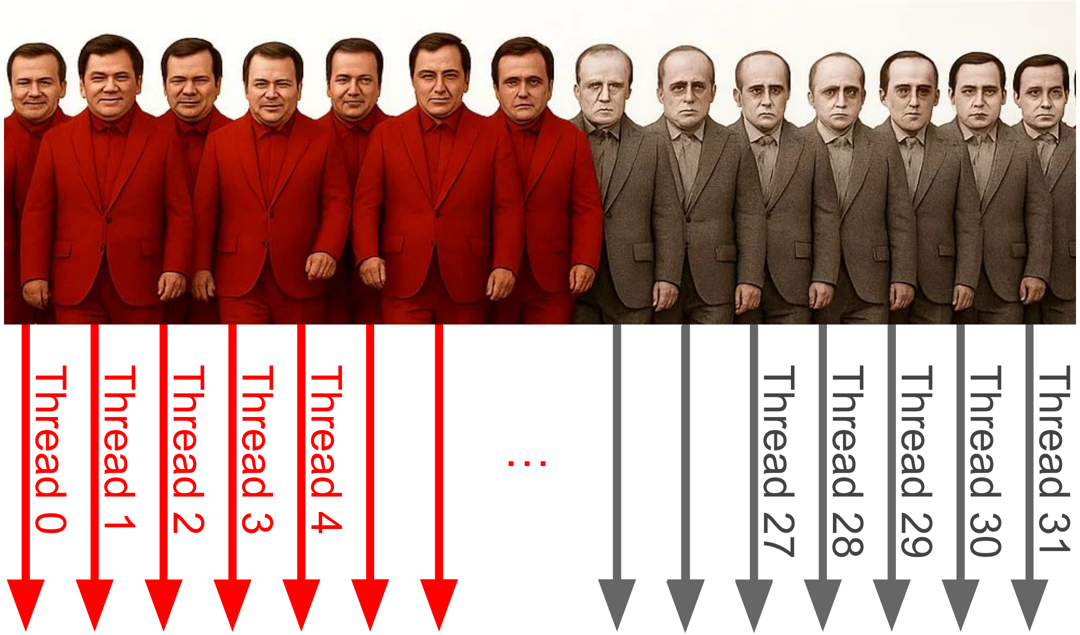
Один Instruction pointer
на все потоки warp-а!

Выходит зайдем в **обе ветки if**-а?

Архитектура

GPU warp



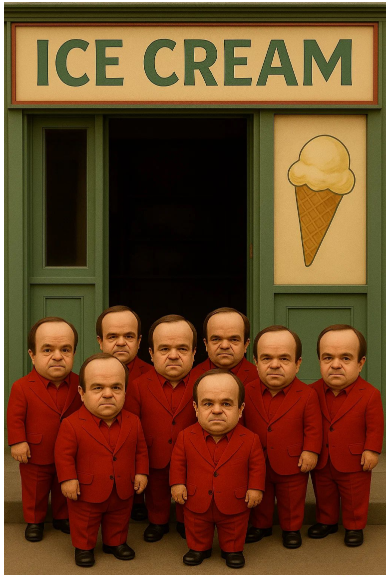


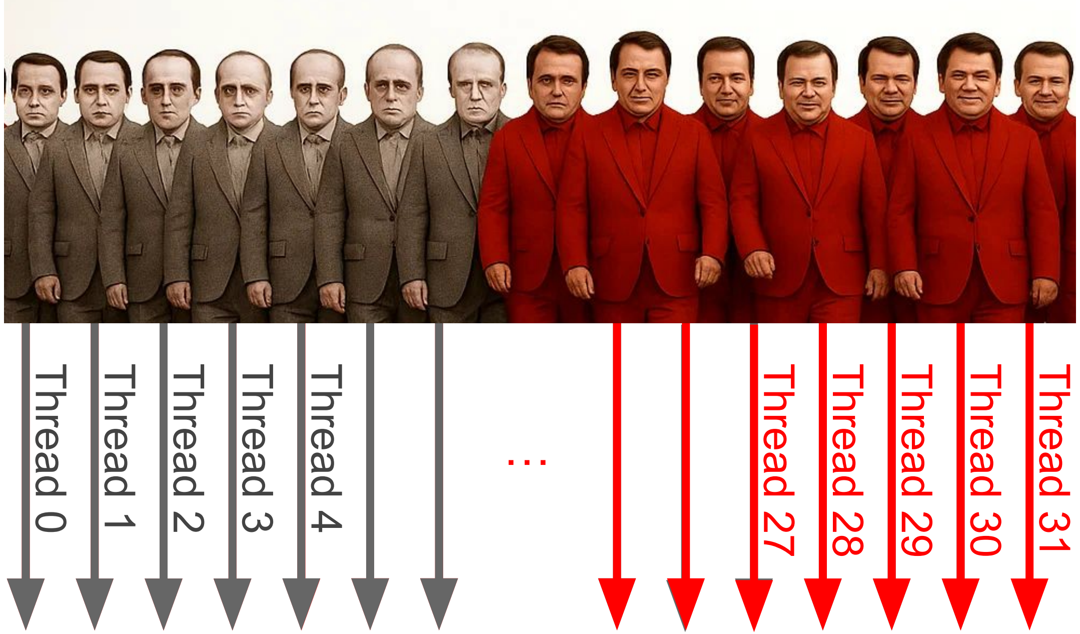

```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID]; // exec mask: [++++ ++++ ++++ ++++ ---- ---- ---- ----]
} else {
    result = dataB[warpThreadID]; // exec mask: [---- ---- ---- ---- ++++ ++++ ++++ ++++]
}
```

Один Instruction pointer
на все потоки warp-а!

Архитектура

GPU warp



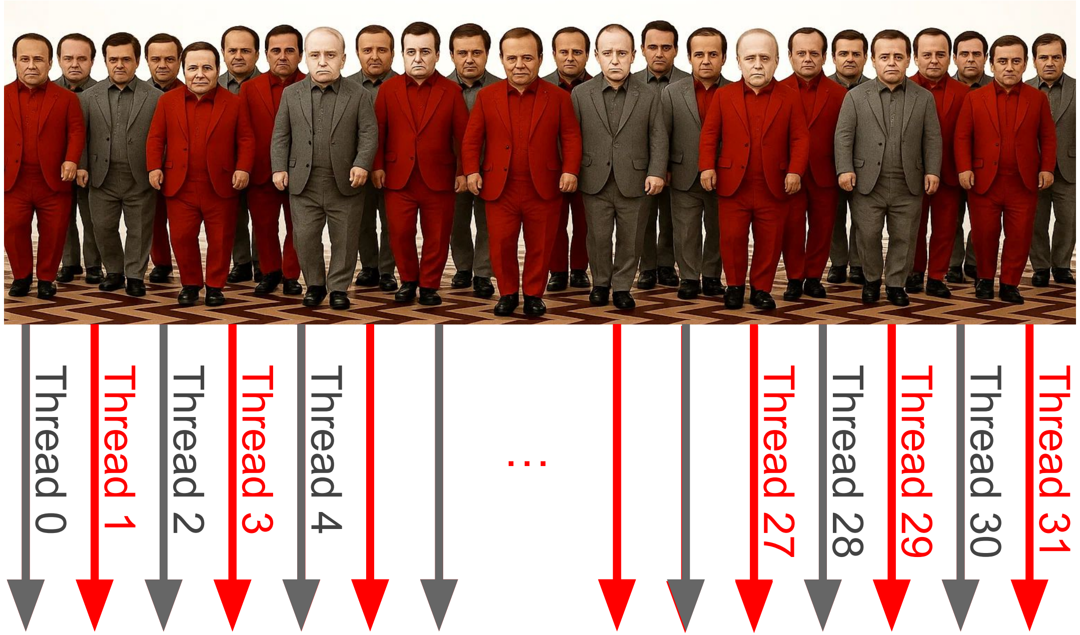



```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID < 16) {
    result = dataA[warpThreadID]; // exec mask: [++++ ++++ ++++ ++++ ---- ---- ---- ----]
} else {
    result = dataB[warpThreadID]; // exec mask: [---- ---- ---- ---- ++++ ++++ ++++ ++++]
}
```

Один Instruction pointer
на все потоки warp-а!

Архитектура

GPU warp

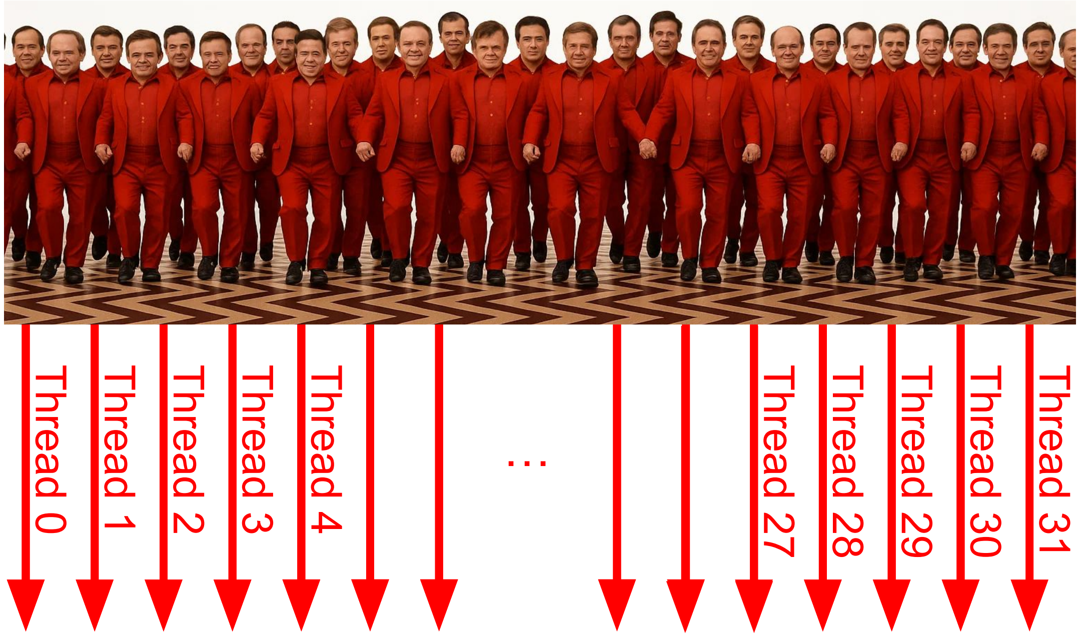


Один Instruction pointer
на все потоки warp-а!

```
int warpThreadID = threadIdx.x % 32;
int result = 0;
if (warpThreadID % 2 == 0) {
    result = dataA[warpThreadID]; // exec mask: [+-+- +-+- +-+- +-+- +-+- +-+- +-+- +-+-]
} else {
    result = dataB[warpThreadID]; // exec mask: [-+-+ -+-+ -+-+ -+-+ -+-+ -+-+ -+-+ -+-+]
}
```

Архитектура

GPU warp



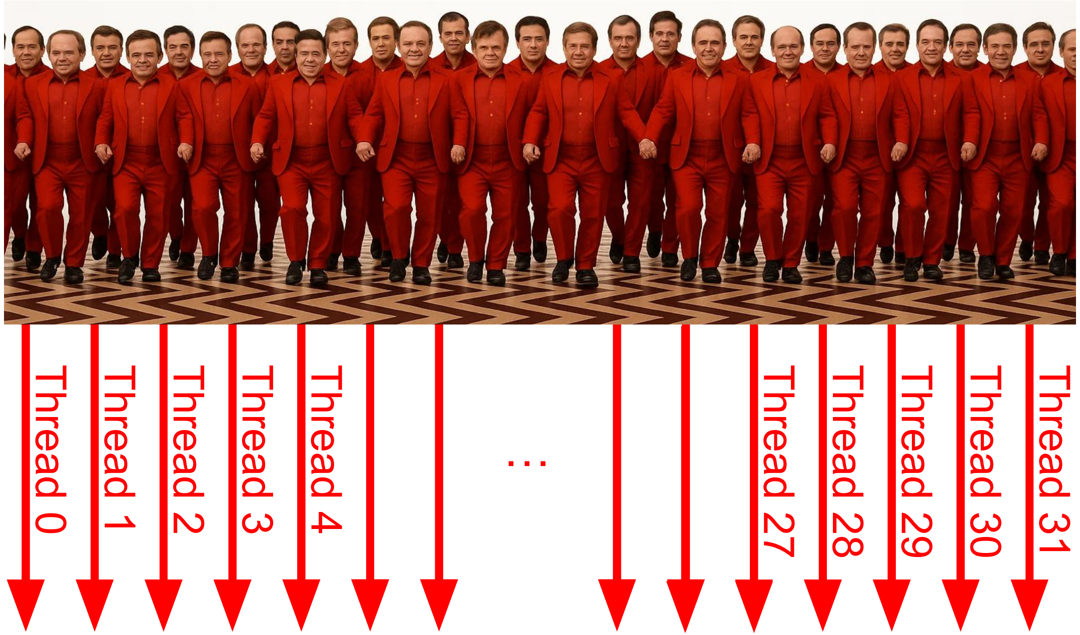
Один Instruction pointer
на все потоки warp-а!

```
if (predicate1) {
    if (predicate2) {
        if (predicate3) {
            // A1
        } else {
            // A2
        }
    } elif (predicate4) {
        // A3
    }
} else {
    // A4
}
```

Сколько “времени” будет работать warp?

Архитектура

GPU warp 



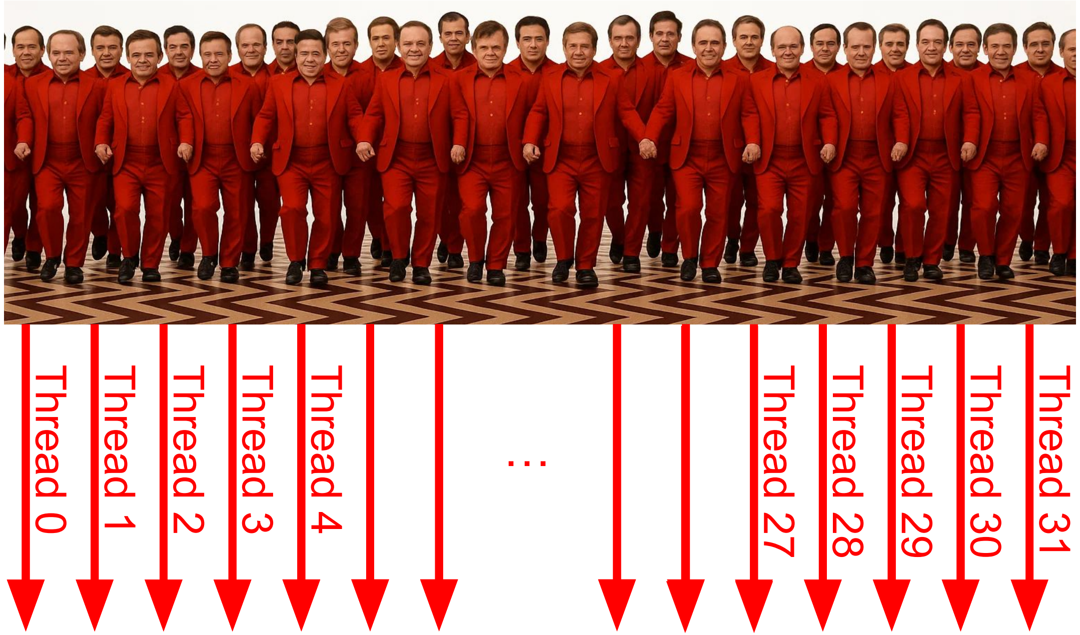
Один Instruction pointer
на все потоки warp-а!

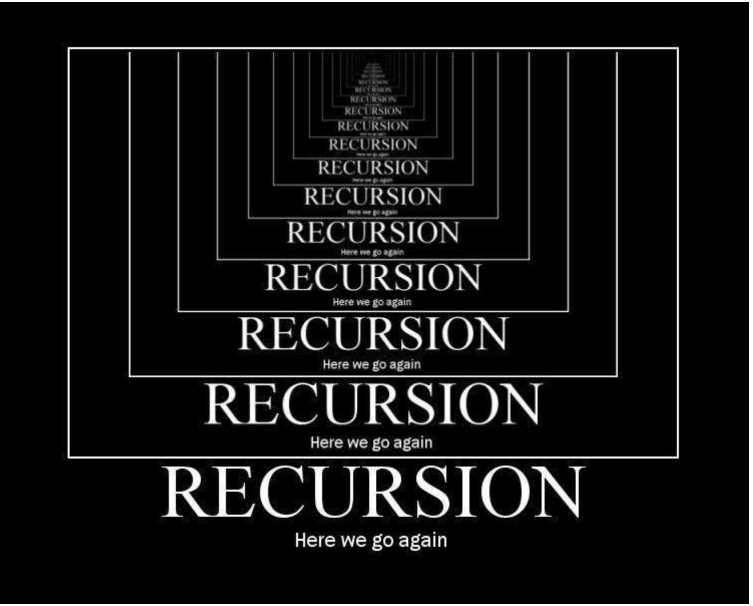
```
if (predicate1) {
    if (predicate2) {
        if (predicate3) {
            // A1
        } else {
            // A2
        }
    } elif (predicate4) {
        // A3
    }
} else {
    // A4
}
```

Сколько “времени” будет работать warp?
A1 + A2 + A3 + A4 при **code divergence**

Архитектура

GPU warp



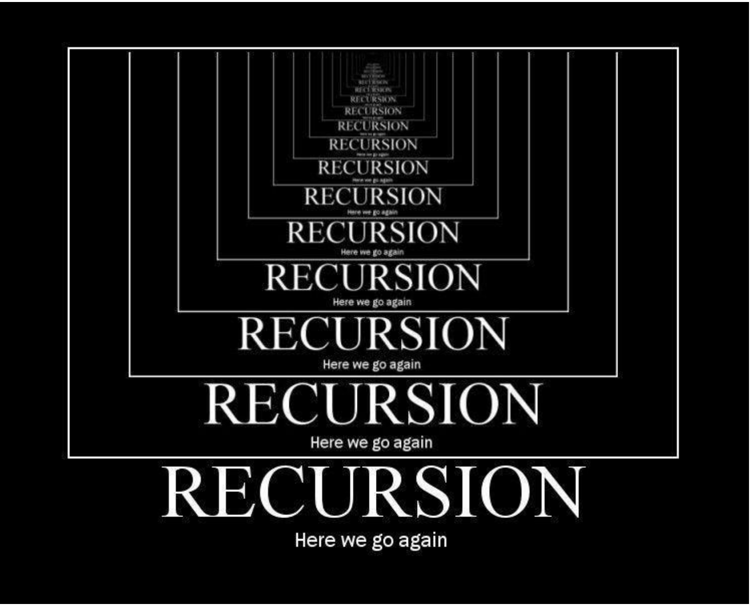


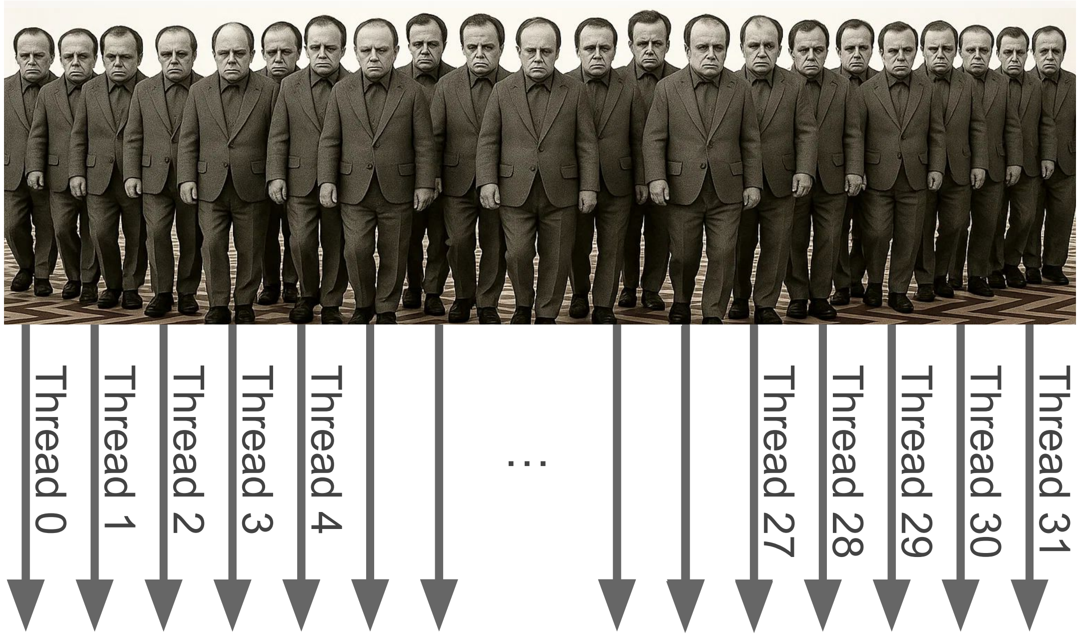
Один Instruction pointer
на все потоки warp-а!

Сколько “времени” будет работать достаточно глубокая рекурсия?

Архитектура

GPU warp



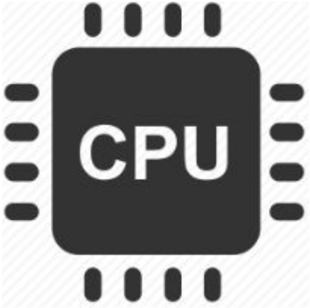


Один Instruction pointer
на все потоки warp-a!

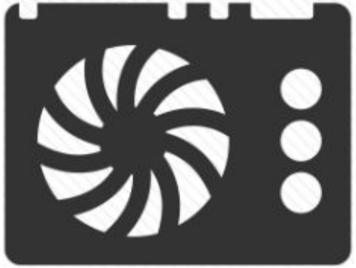
Глава 2: Работа с памятью

hyper-threading, latency hiding, occupancy, registers pressure/spilling, coalesced memory access pattern, cache lines, local/shared memory

Архитектура



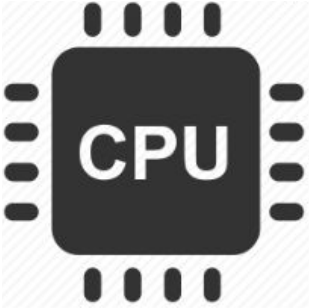
40
GB/s



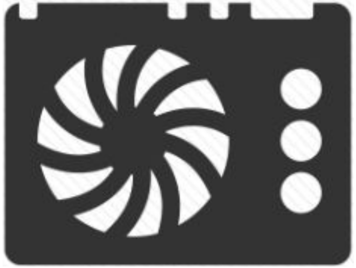
1000
GB/s

GPU

Архитектура



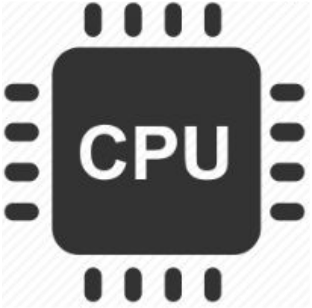
40
GB/s



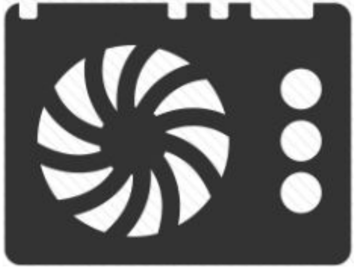
1000
GB/s

GPU

Архитектура



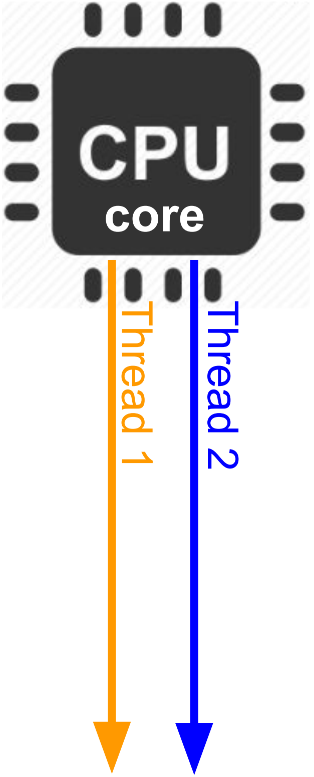
40
GB/s

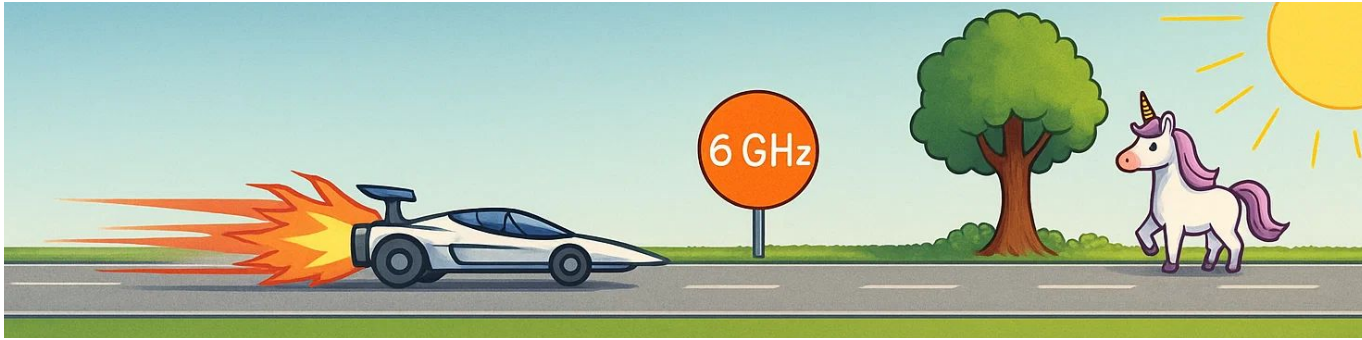


1000
GB/s

GPU

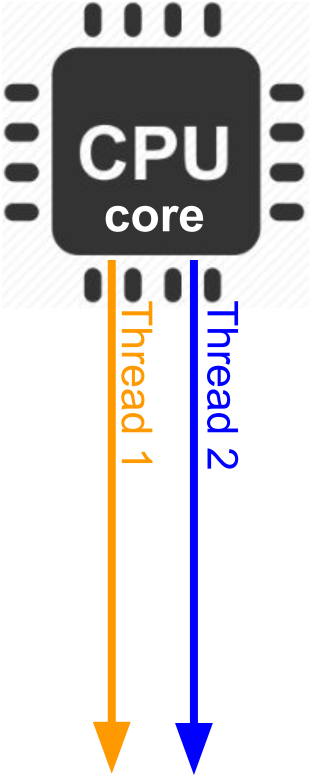
Архитектура CPU: многопоточность

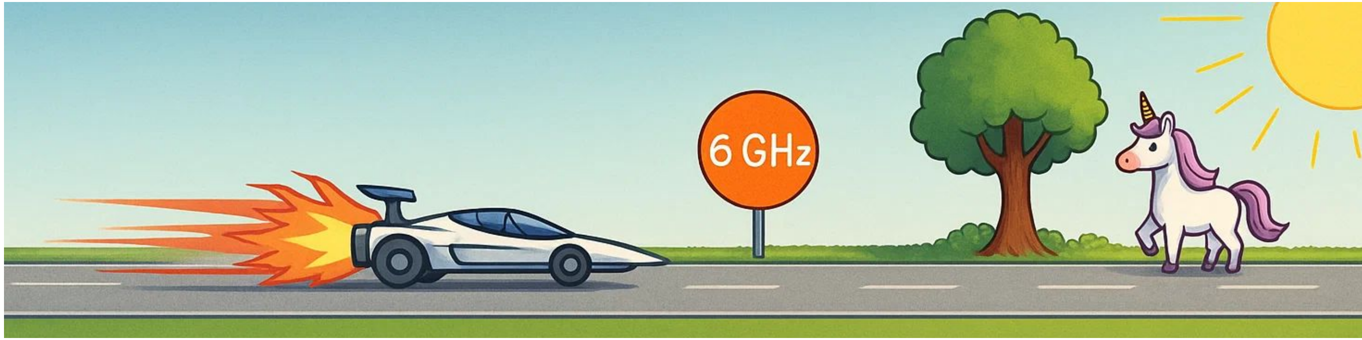




Многопоточность благодаря переключению контекста!

Архитектура CPU: многопоточность

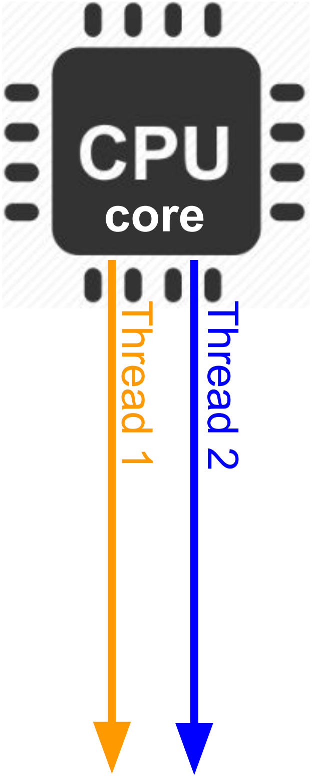


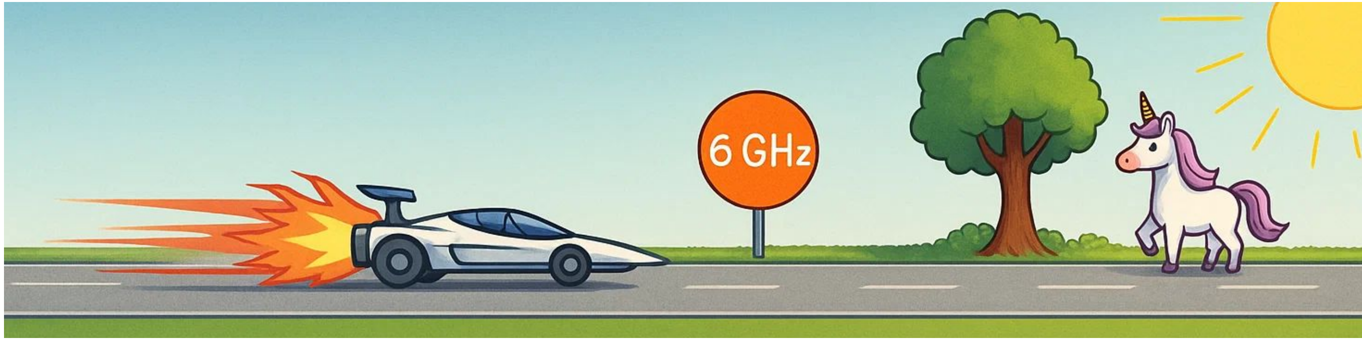


Многопоточность благодаря переключению контекста!

Что нужно перещелкнуть в состоянии ЦПУ ядра для другого потока?

Архитектура CPU: многопоточность

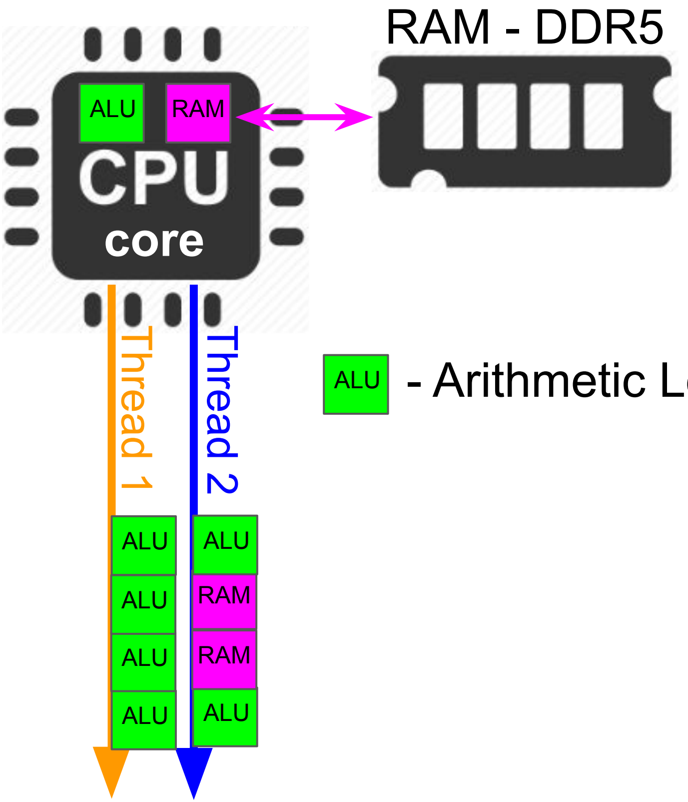


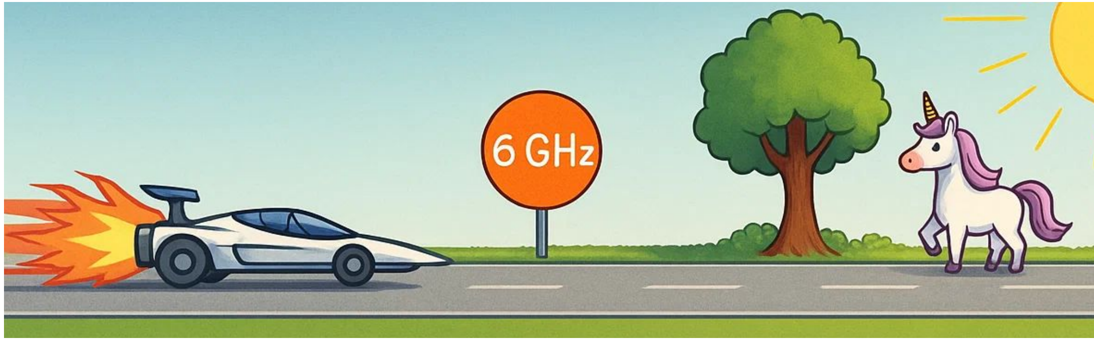


Многопоточность благодаря переключению контекста!

Context switch:
- Обновляет Instruction pointer (указатель на строку кода)
- Подгружает значения регистров процессора (откуда?)

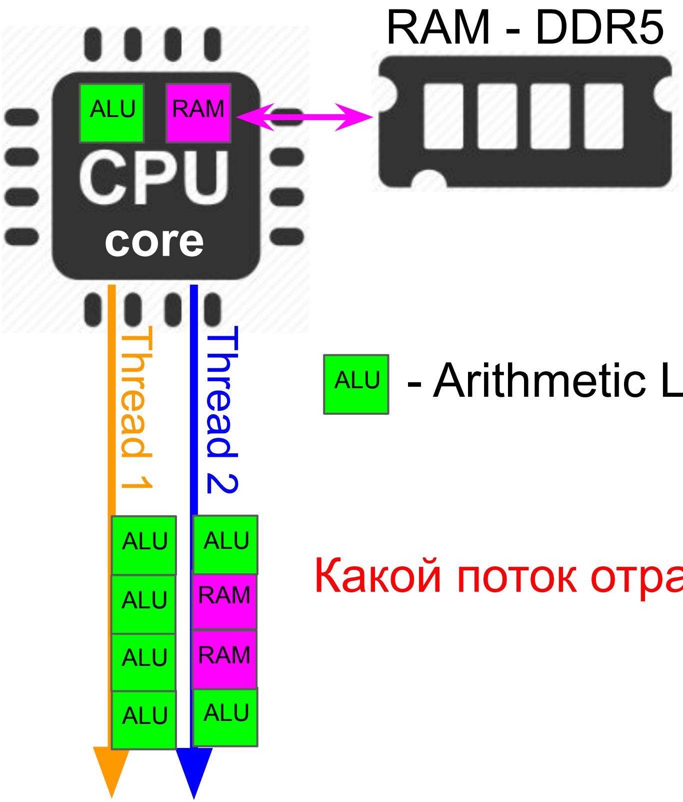
Архитектура CPU: Hyper-Threading, SMT

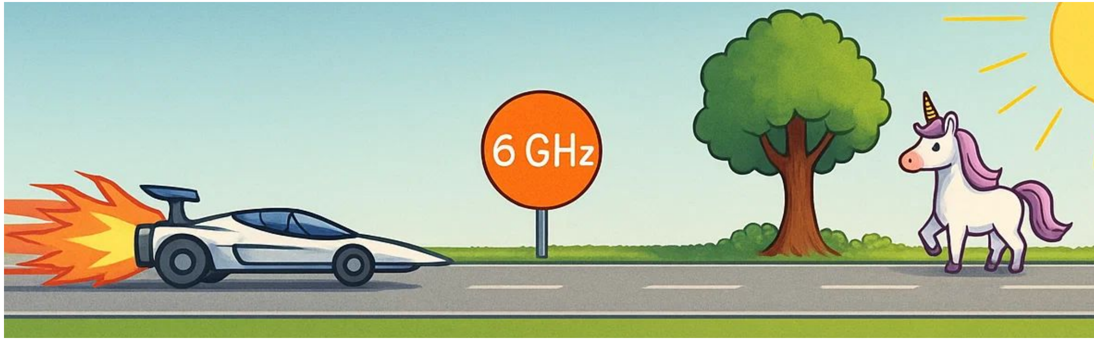




ALU - Arithmetic Logical Unit

Архитектура CPU: Hyper-Threading, SMT

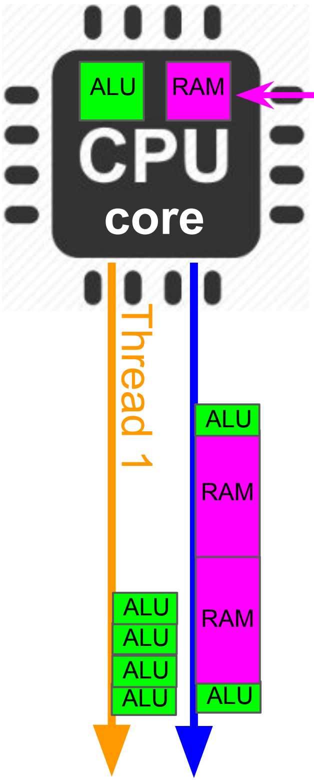


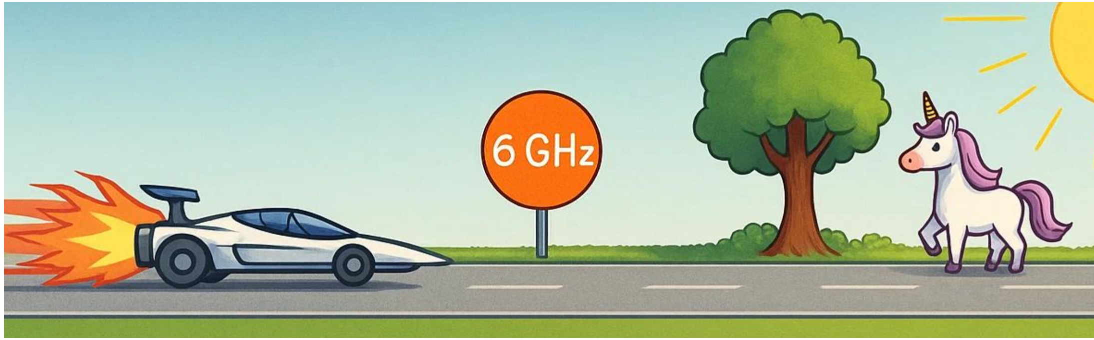


ALU - Arithmetic Logical Unit

Какой поток отработает быстрее?

Архитектура CPU: Hyper-Threading, SMT

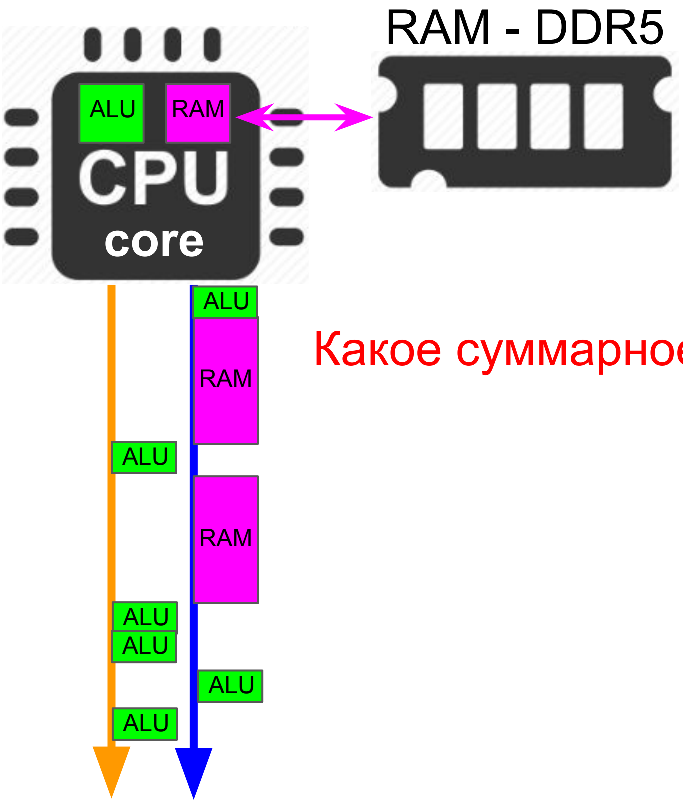


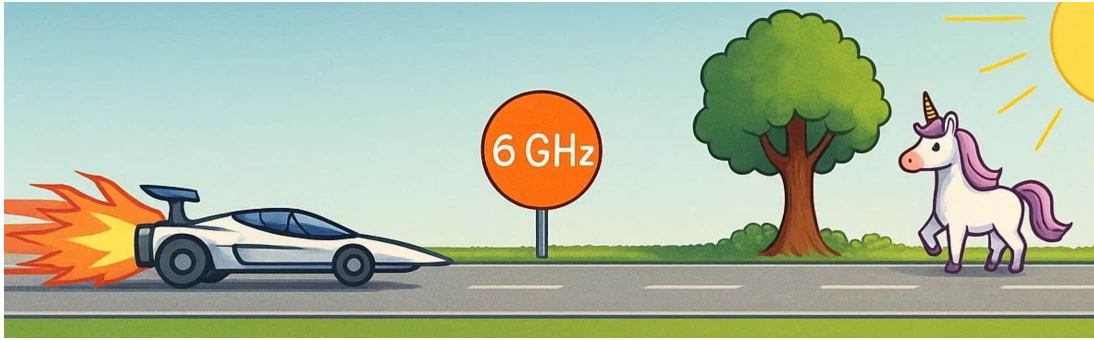


Thread 1 отработает быстрее!

Latency RAM много больше чем у ALU

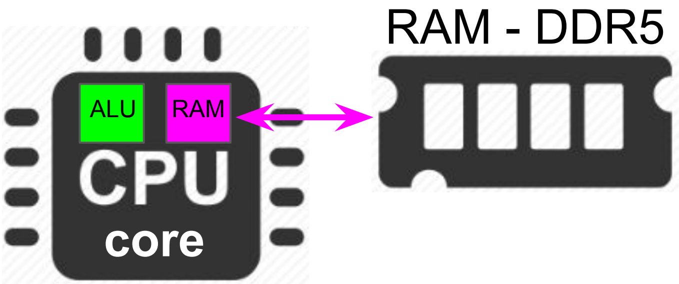
Архитектура CPU: Hyper-Threading, SMT

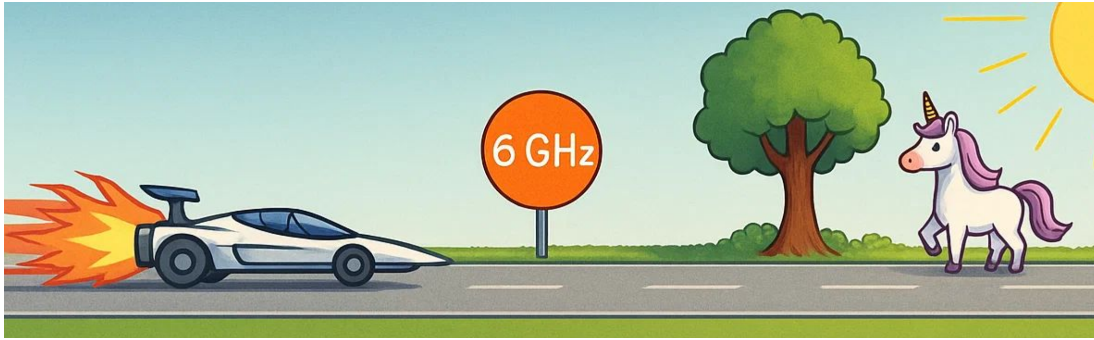


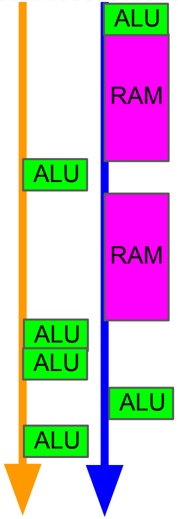


Какое суммарное время работы двух таких потоков?

Архитектура CPU: Hyper-Threading, SMT



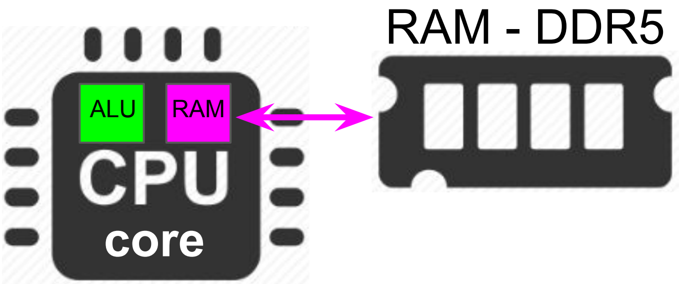


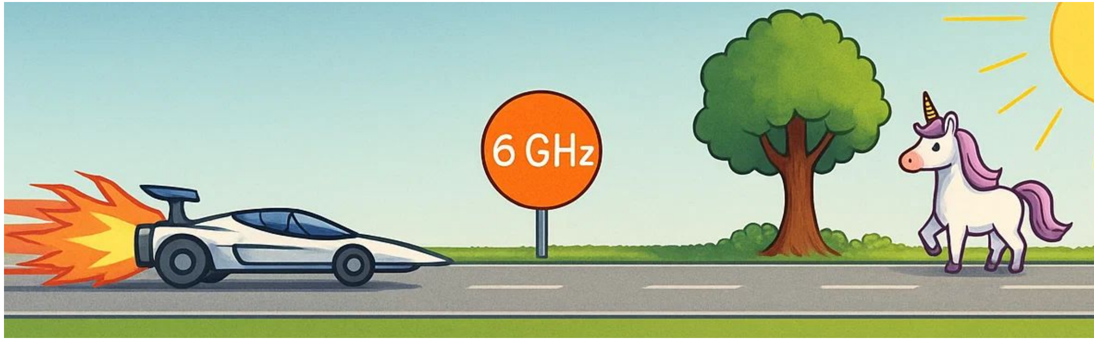


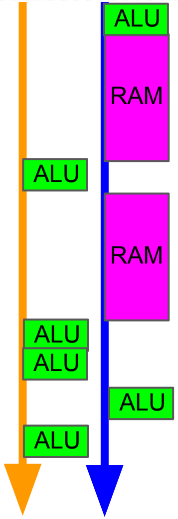
Какое суммарное время работы двух таких потоков?

6x ALU +2x RAM

Архитектура CPU: Hyper-Threading, SMT





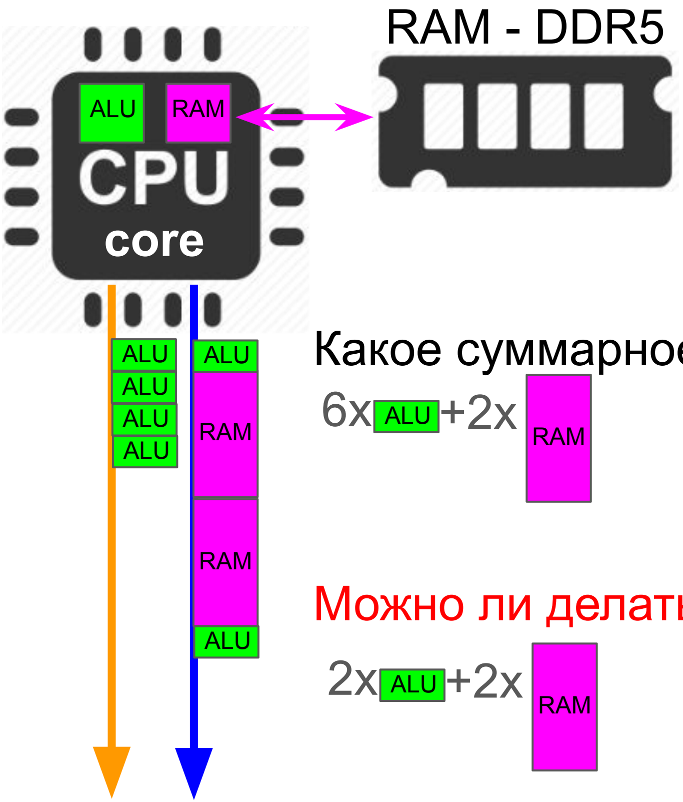


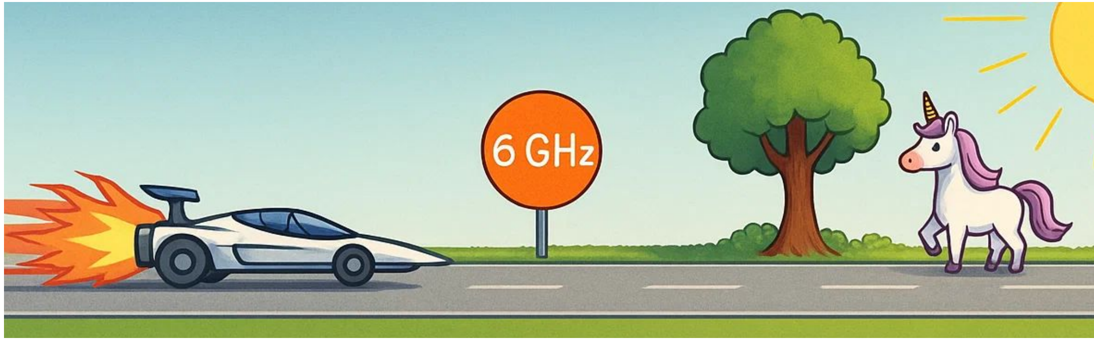
Какое суммарное время работы двух таких потоков?

6x ALU +2x RAM

Можно ли быстрее?

Архитектура CPU: Hyper-Threading, SMT





Какое суммарное время работы двух таких потоков?

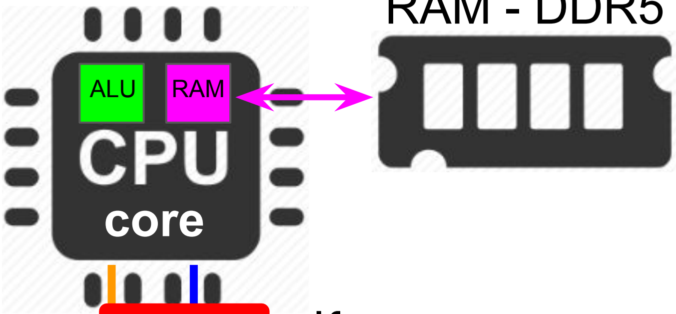
6x ALU +2x RAM

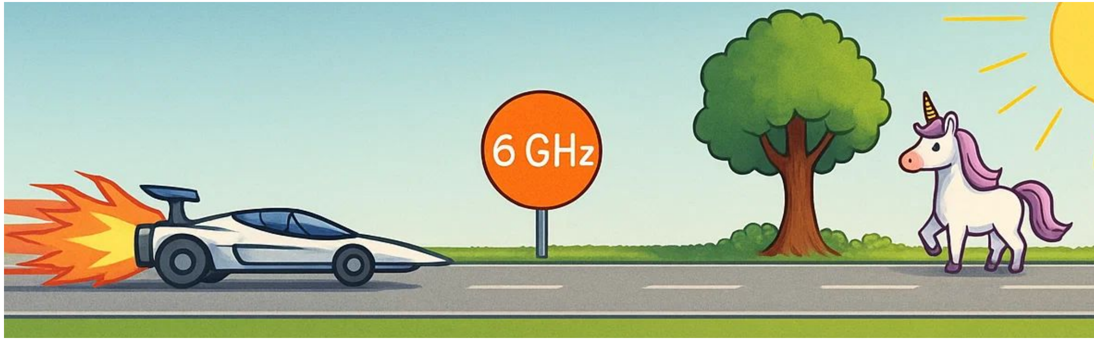
Можно ли делать ALU пока ждем RAM ?

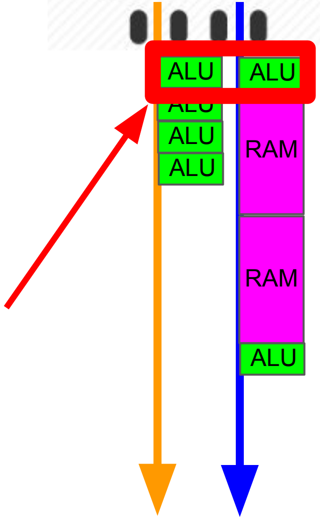
2x ALU +2x RAM

Архитектура CPU: Hyper-Threading, SMT

RAM - DDR5







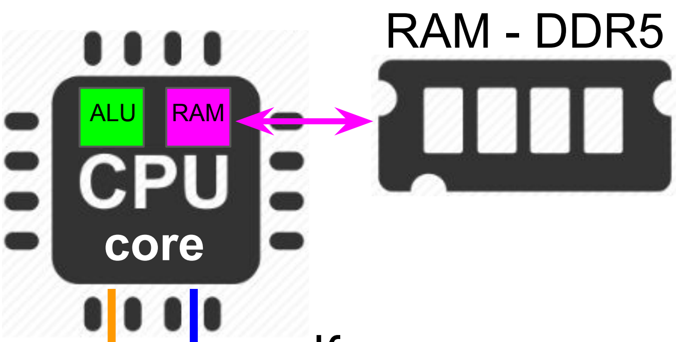
Какое суммарное время работы двух таких потоков?

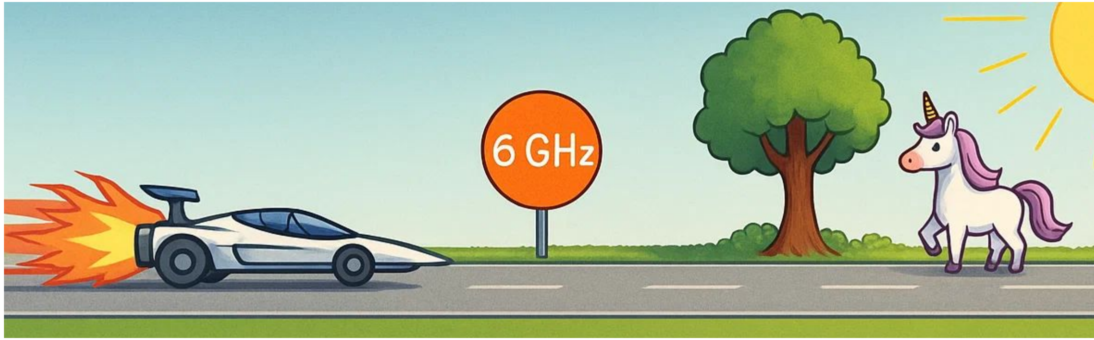
6x ALU +2x RAM

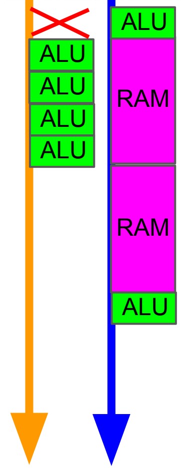
Можно ли делать ALU пока ждем RAM ?

2x ALU +2x RAM

Архитектура CPU: Hyper-Threading, SMT







Какое суммарное время работы двух таких потоков?

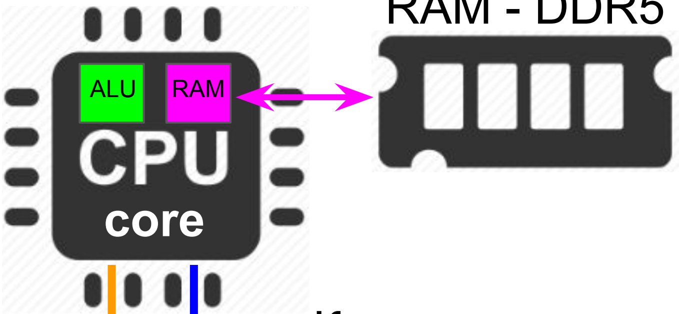
6x ALU +2x RAM

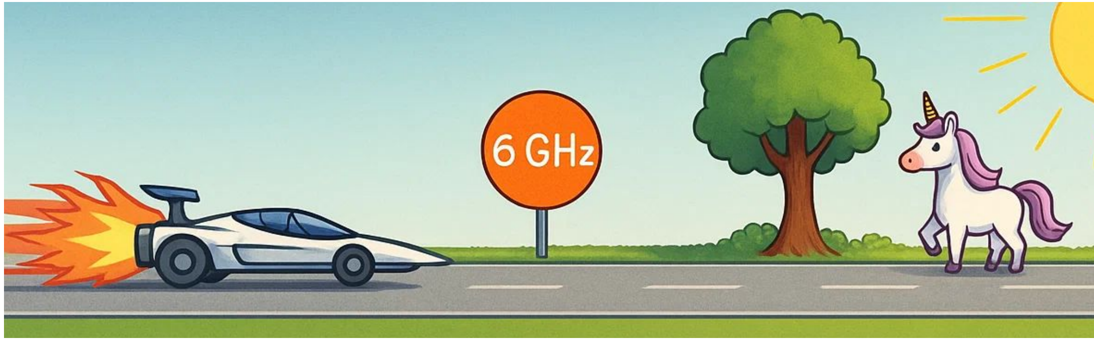
Можно ли делать ALU пока ждем RAM ?

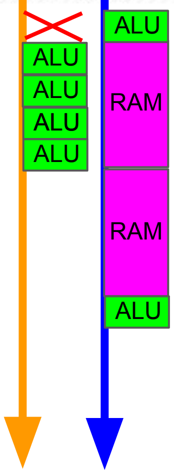
2x ALU +2x RAM

Архитектура CPU: Hyper-Threading, SMT

RAM - DDR5







Какое суммарное время работы двух таких потоков?

6x ALU +2x RAM

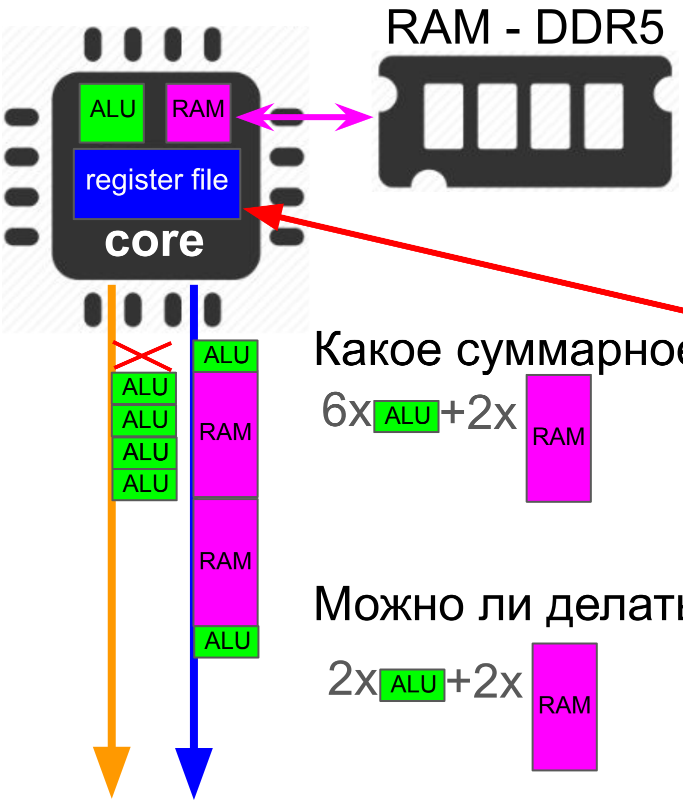
~~Можно ли делать ALU пока ждем RAM ?~~

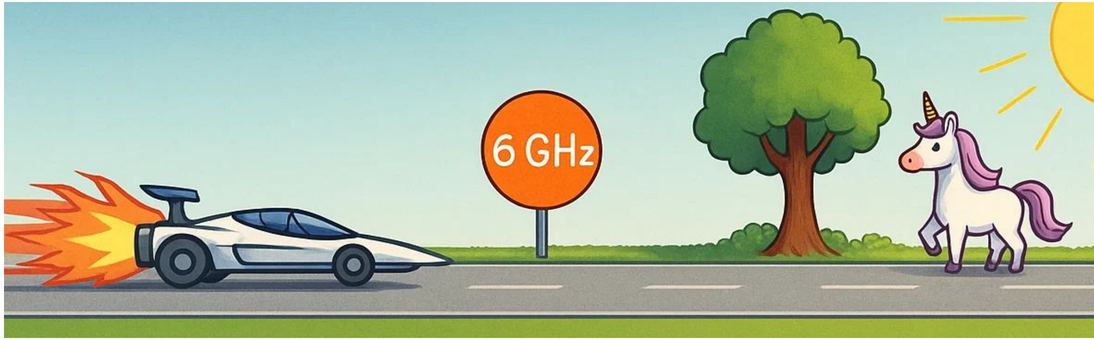
~~2x ALU +2x RAM~~

Context switch - дорого!

Долго подгружает регистры!

Архитектура CPU: Hyper-Threading, SMT





Какое суммарное время работы двух таких потоков?

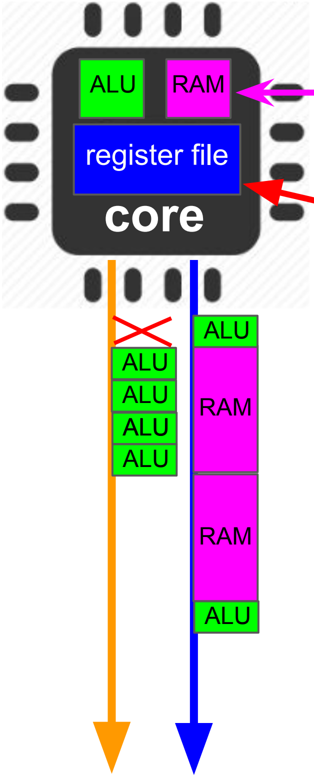
6x ALU +2x RAM

Давайте держать два множества регистров!

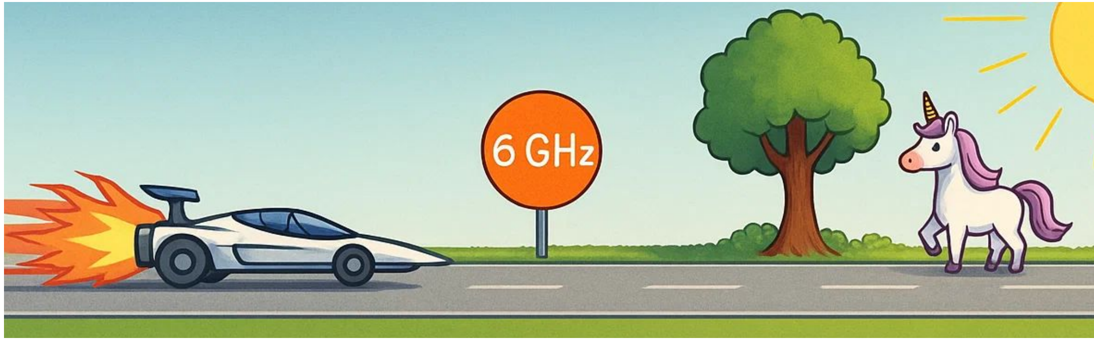
Можно ли делать ALU пока ждем RAM ?

2x ALU +2x RAM

Архитектура CPU: Hyper-Threading, SMT



RAM - DDR5



Какое суммарное время работы двух таких потоков?

6x ALU +2x RAM

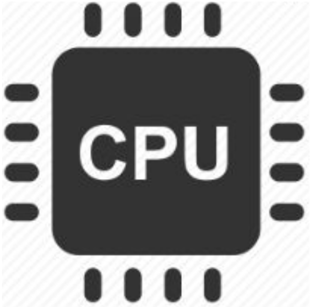
Давайте держать два множества регистров!

Можно ли делать ALU пока ждем RAM ?

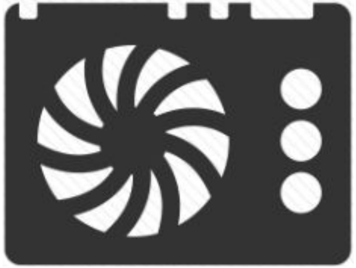
2x ALU +2x RAM

Это и есть SMT и HT!

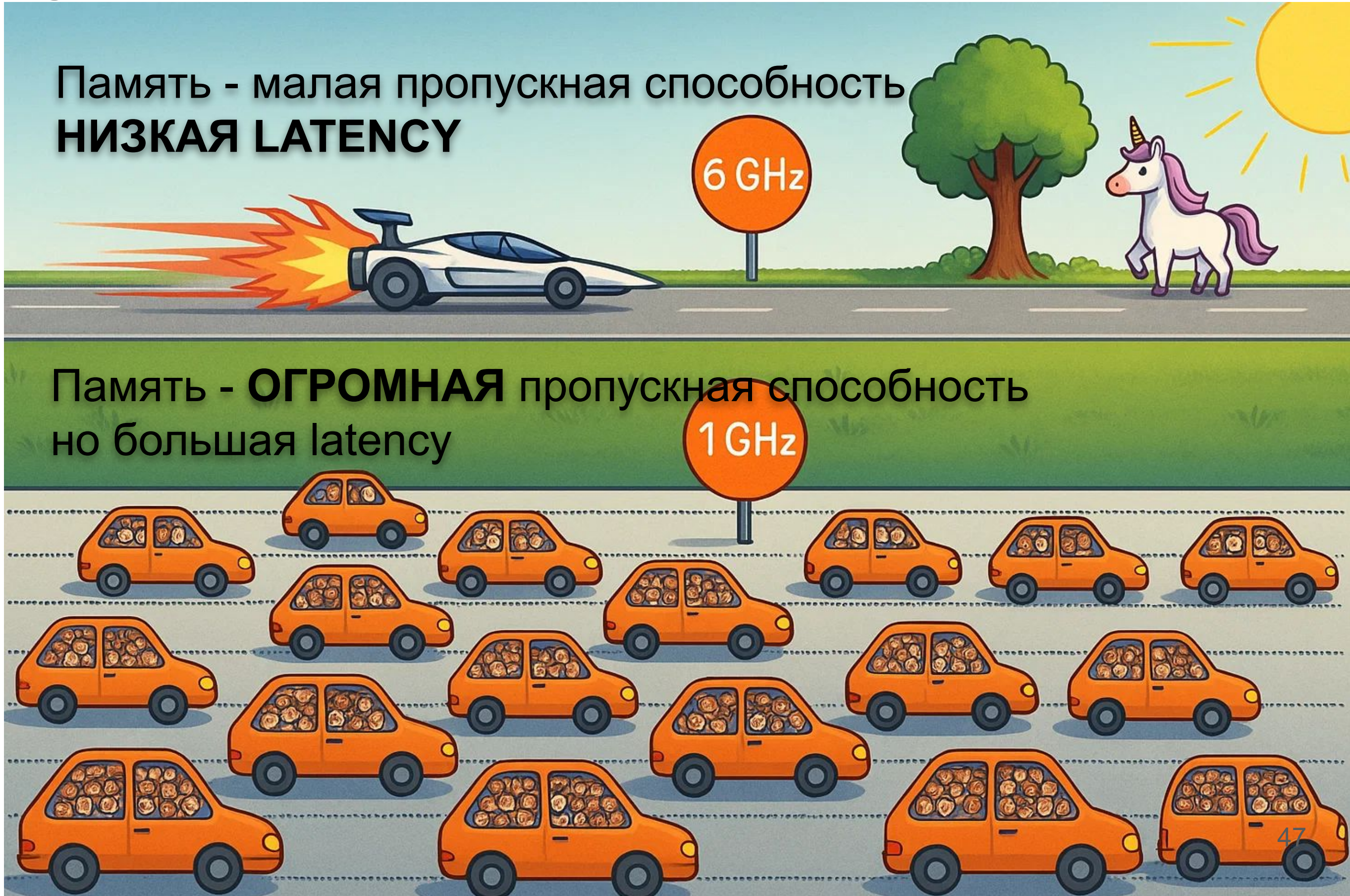
Архитектура GPU



40
GB/s



1000
GB/s

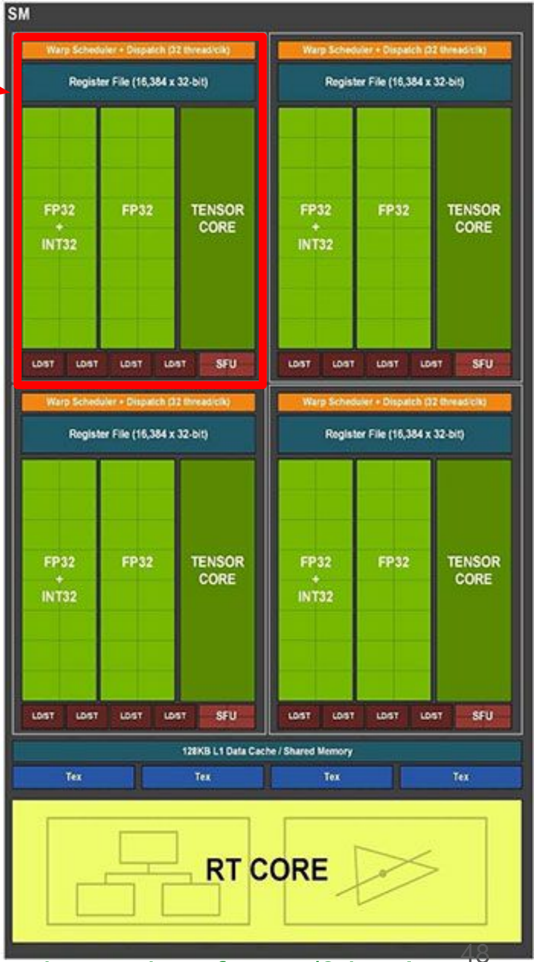


Архитектура GPU

warp

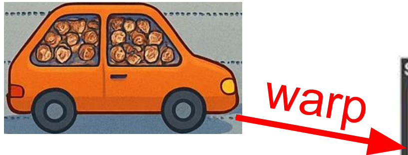
32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor



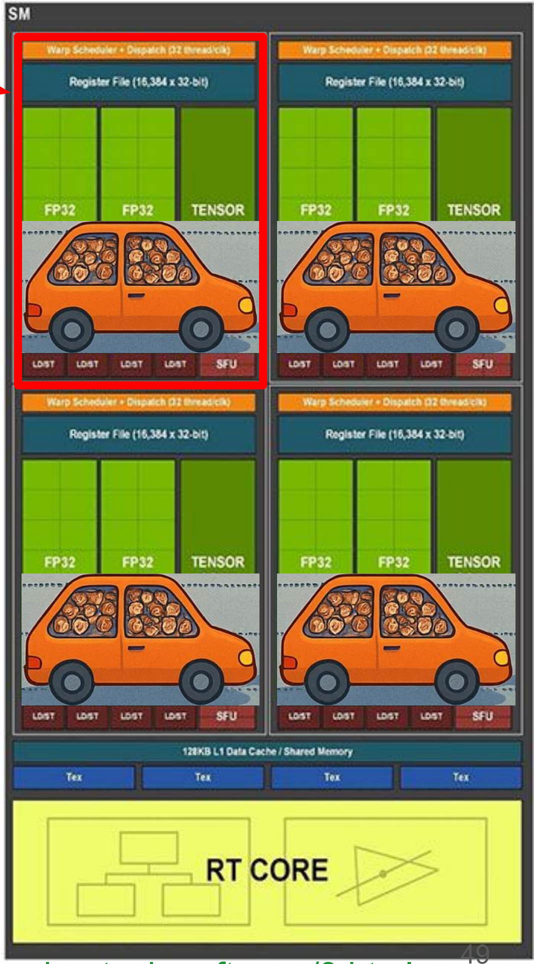
https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU



32 слабых медленных CUDA ядер

SM - Streaming Multiprocessor



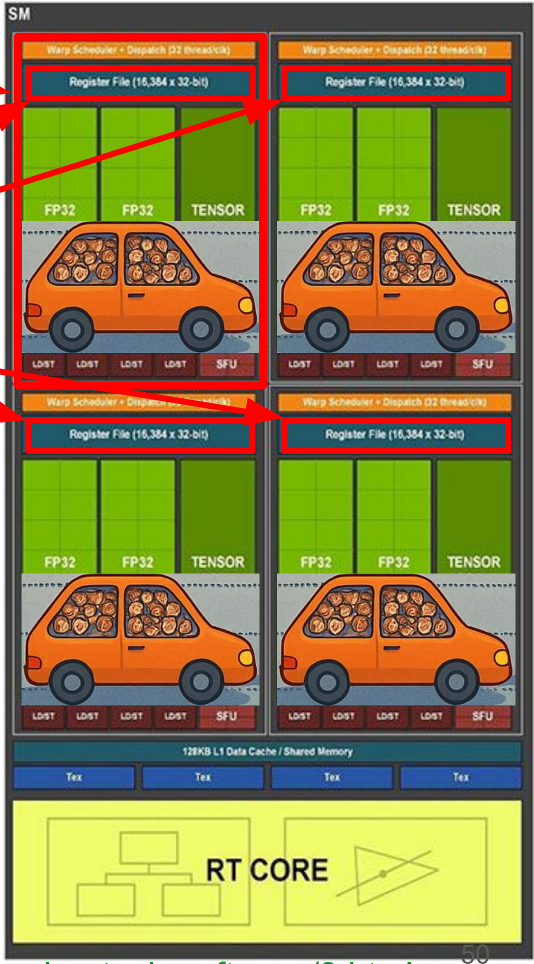
https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

32 слабых медленных CUDA ядер

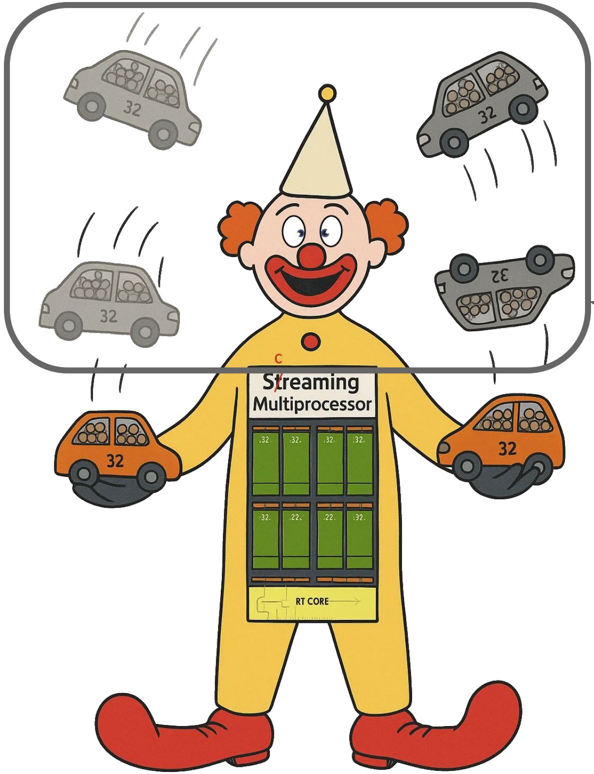
SM - Streaming Multiprocessor



Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU



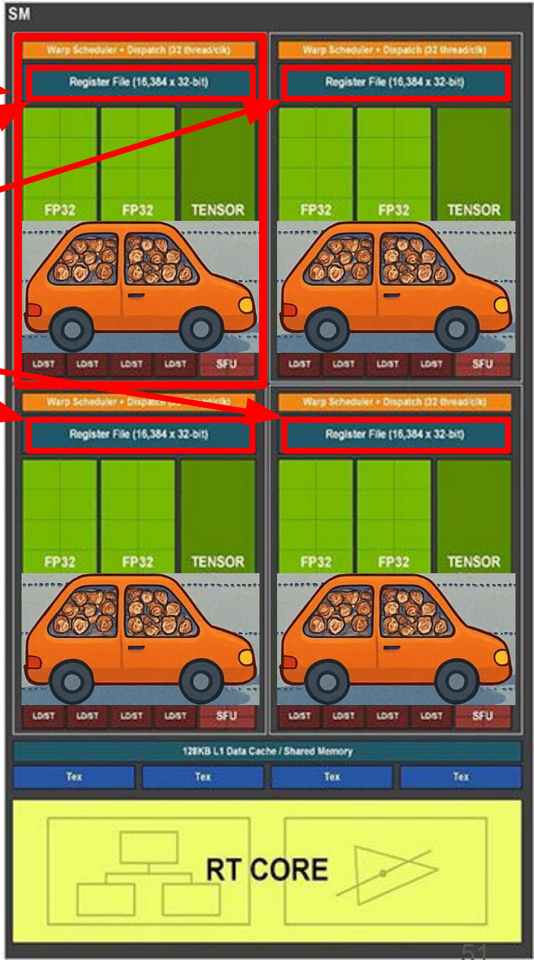
warp

32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Inactive warps: ждут когда придут данные из VRAM

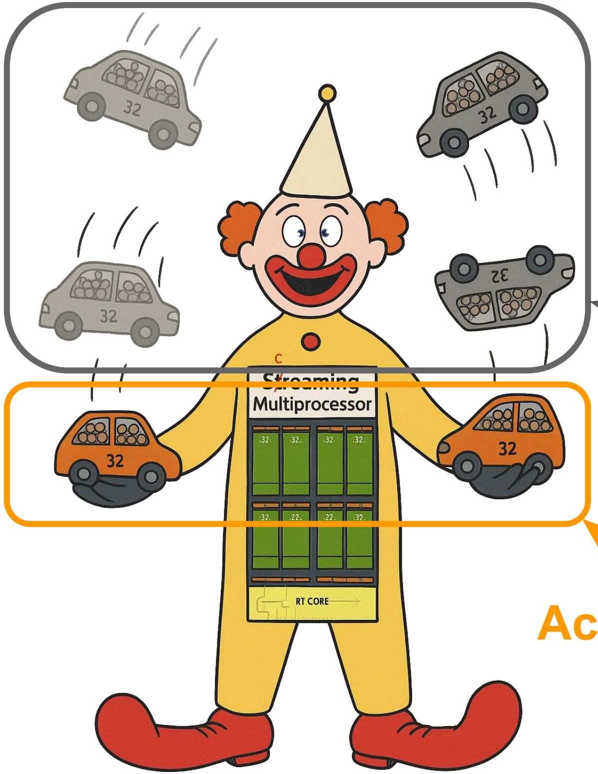
SM - Streaming Multiprocessor



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp



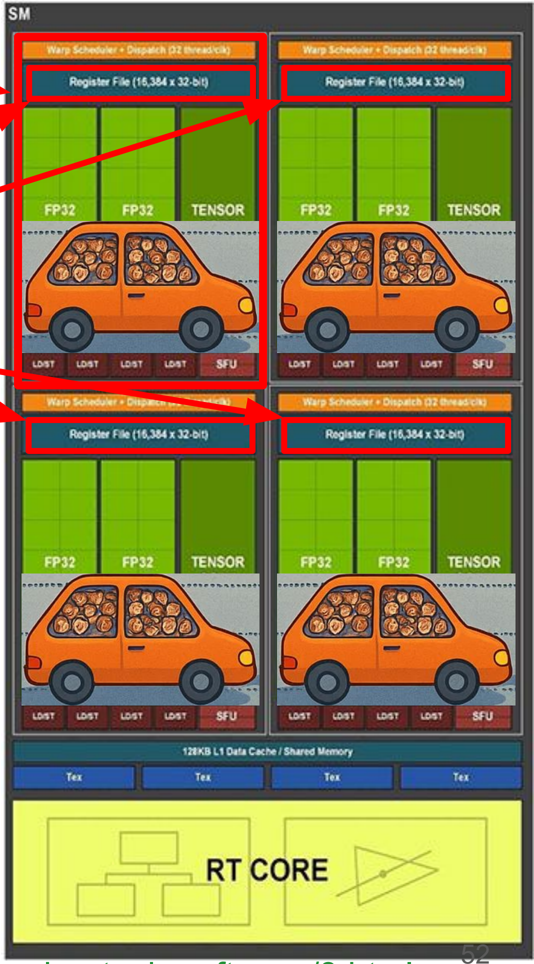
32 слабых медленных
CUDA ядер

Register File для быстрого
жонглирования warp-ами!
(быстрый “context switch”)

Inactive warps: ждут когда
придут данные из VRAM

Active warps: данные в регистрах
Работаем-работаем! 😎

SM - Streaming Multiprocessor



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

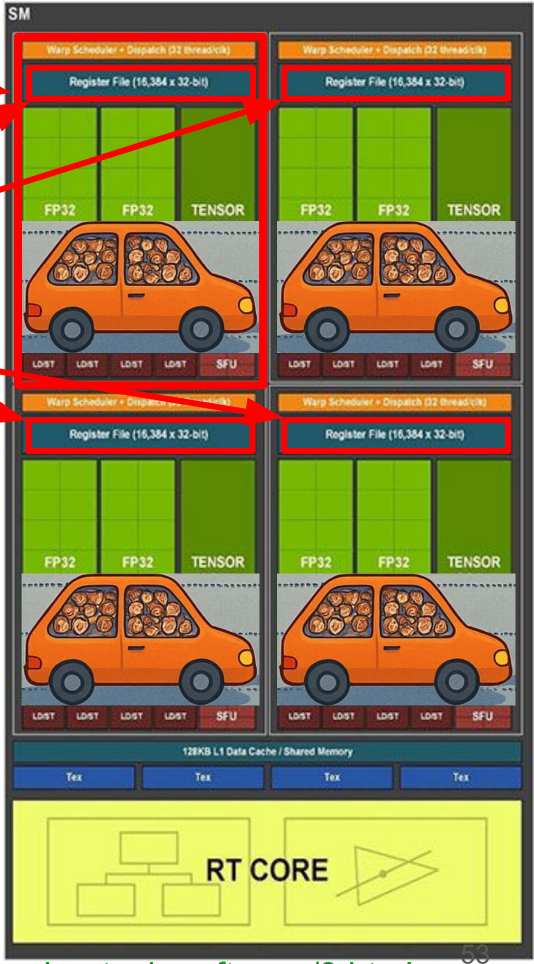
warp

32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Occupancy = насколько много доступно warp-ов для жонглирования.

SM - Streaming Multiprocessor



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

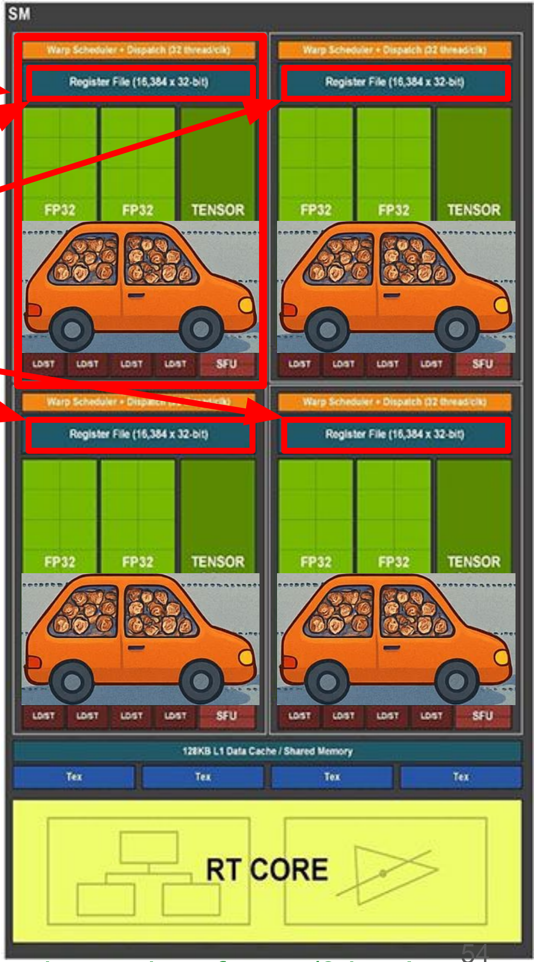
32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то что?

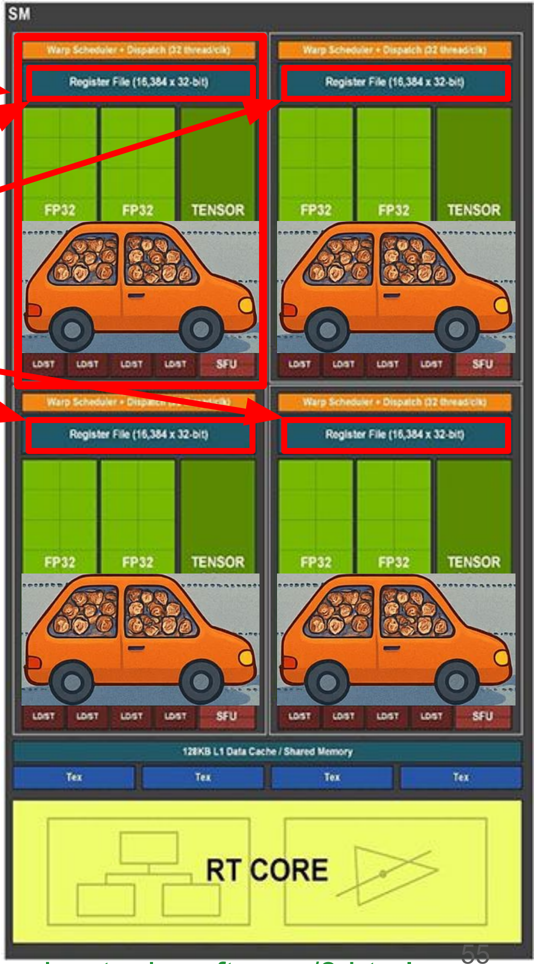
SM - Streaming Multiprocessor



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

warp

SM - Streaming Multiprocessor



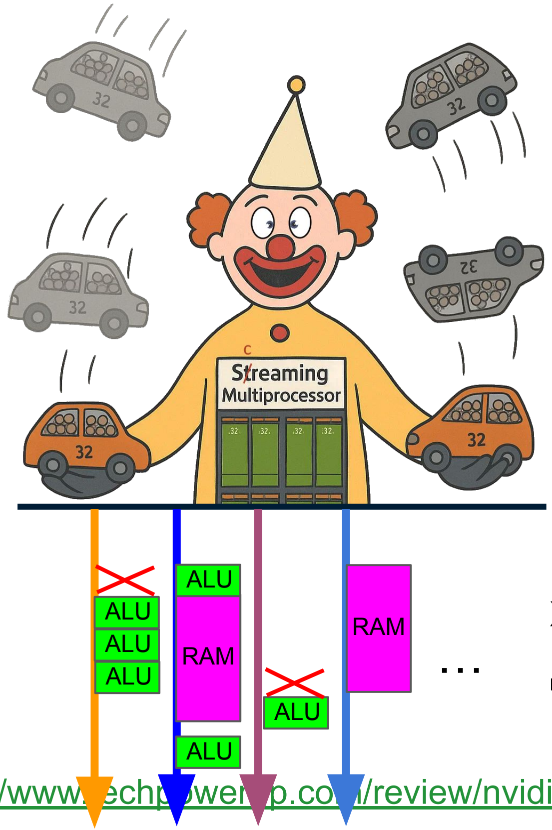
Архитектура GPU

32 слабых медленных CUDA ядер

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

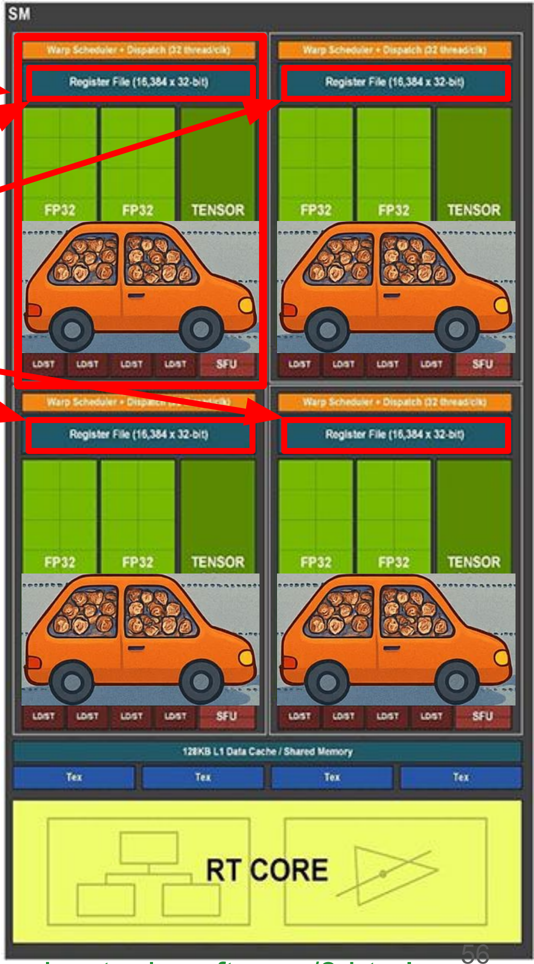


https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

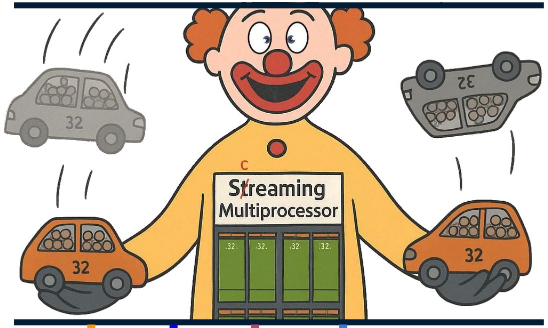
warp

SM - Streaming Multiprocessor



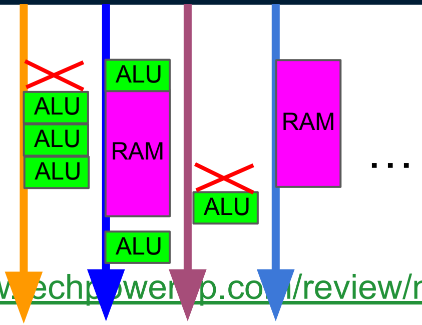
32 слабых медленных CUDA ядер

А почему у SM может быть разное число warp-ов?



Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Occupancy = насколько много доступно warp-ов для жонглирования.



Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

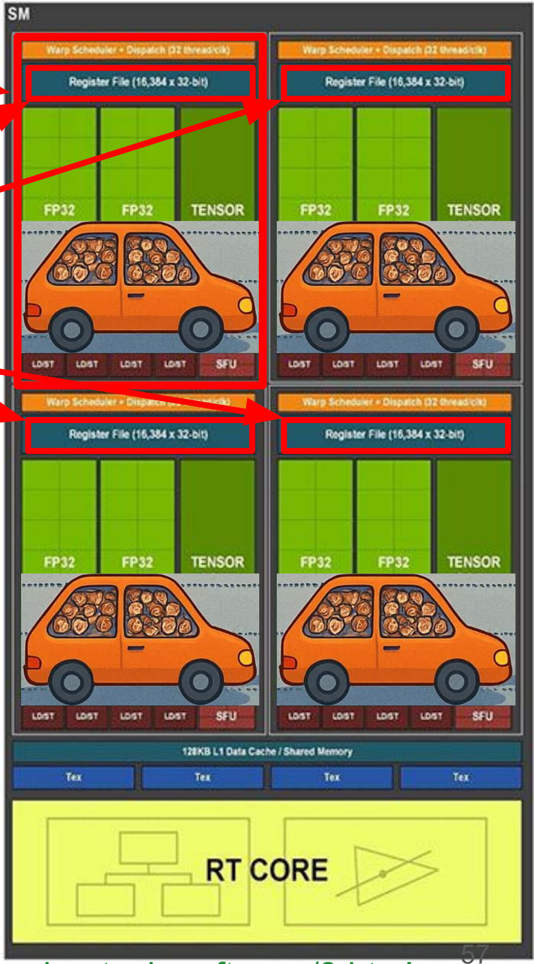
https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

warp

32 слабых медленных CUDA ядер

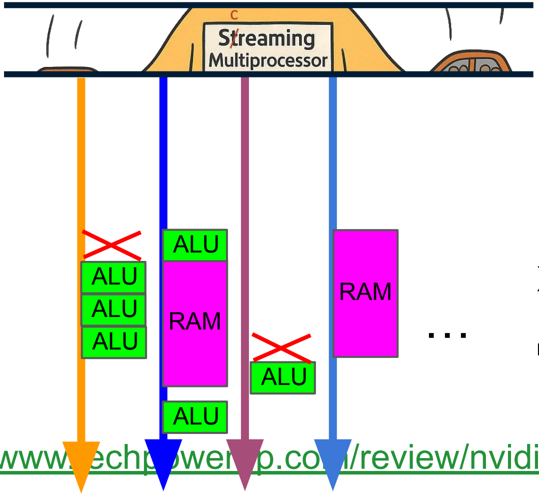
SM - Streaming Multiprocessor



А почему у SM может быть разное число warp-ов?

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Registers Pressure!



Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

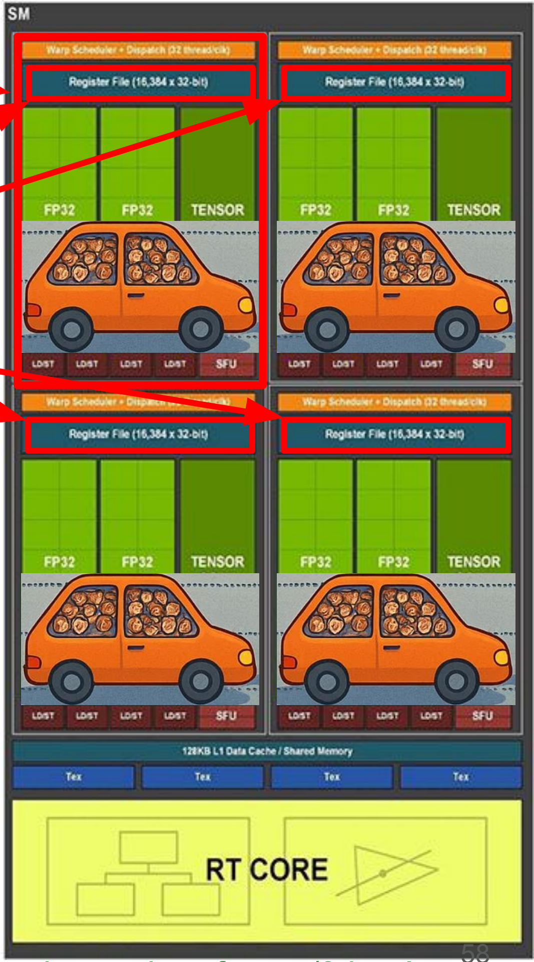
https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура GPU

SM - Streaming Multiprocessor

warp

32 слабых медленных CUDA ядер

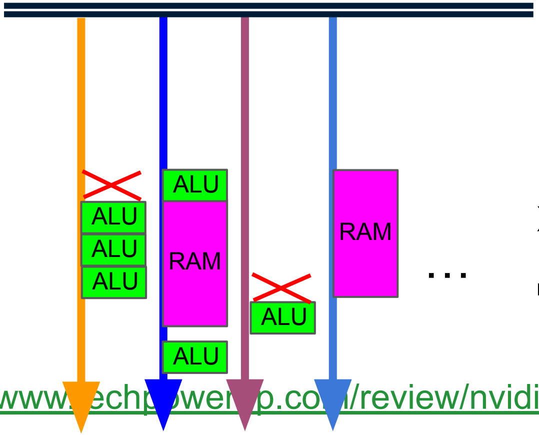


А почему у SM может быть разное число warp-ов?

Register File для быстрого жонглирования warp-ами! (быстрый “context switch”)

Registers Pressure!

Вплоть до **Registers Spilling**!

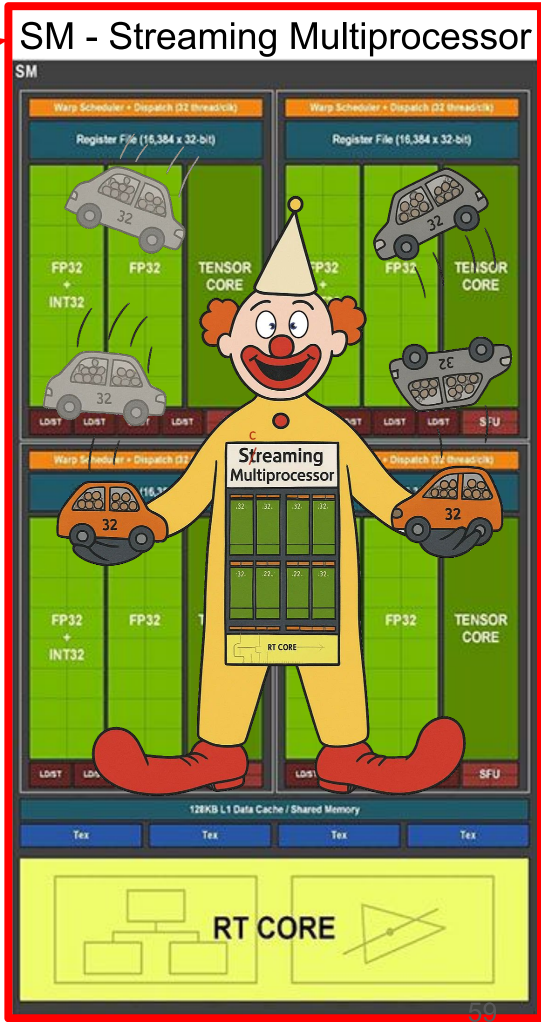


Occupancy = насколько много доступно warp-ов для жонглирования.

Если occupancy высокая, то **хорошо скрывается latency доступа к памяти** т.к. всегда найдется готовый warp!

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

RTX 3090: **10496 CUDA cores** = 82 SM · 4 warps · 32 ALUs



https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

RTX 3090: **10496 CUDA cores** = 82 SM · 4 warps · 32 ALUs

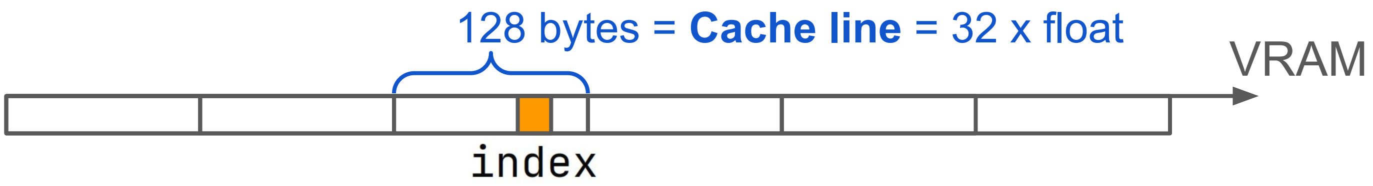
Куда сбежали два клоуна?

https://www.techpowerup.com/review/nvidia-geforce-ampere-architecture-board-design-gaming-tech-software/3.html

Архитектура VRAM (**coalesced** memory access pattern)

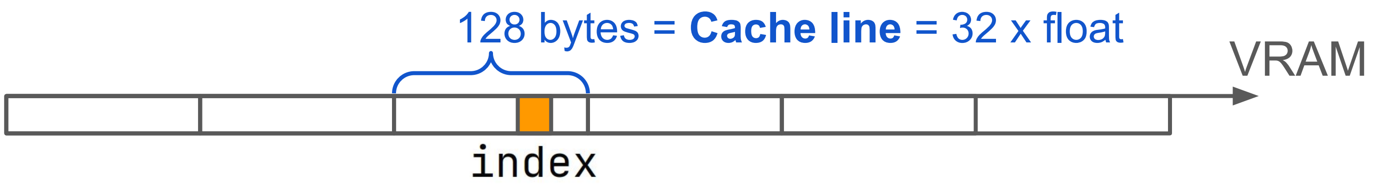

```
float value = data[index];
```

Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

Архитектура VRAM (**coalesced** memory access pattern)

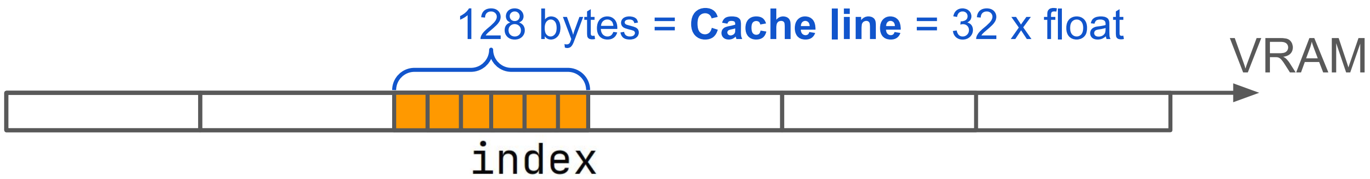
128 bytes = **Cache line** = 32 x float


```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	???

Сколько cache line-ов чтобы покрыть заказ?
Сколько потребуется **транзакций** из VRAM?

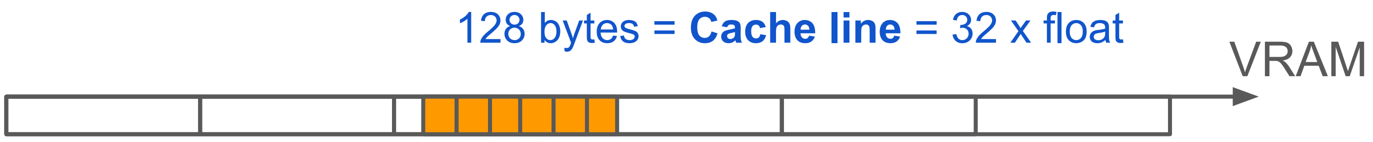
Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)

Архитектура VRAM (**coalesced** memory access pattern)

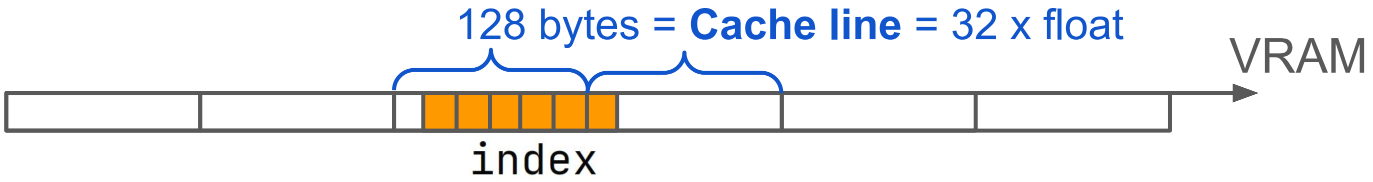



```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	???

Сколько cache line-ов чтобы покрыть заказ?
Сколько потребуется **транзакций** из VRAM?

Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)

Насколько просела достигнутая полезная пропускная способность VRAM?

Архитектура VRAM (**coalesced** memory access pattern)

128 bytes = **Cache line** = 32 x float


```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	???

Сколько cache line-ов чтобы покрыть заказ?

Сколько потребуется **транзакций** из VRAM?

Архитектура VRAM (**coalesced** memory access pattern)

128 bytes = **Cache line** = 32 x float

VRAM

index

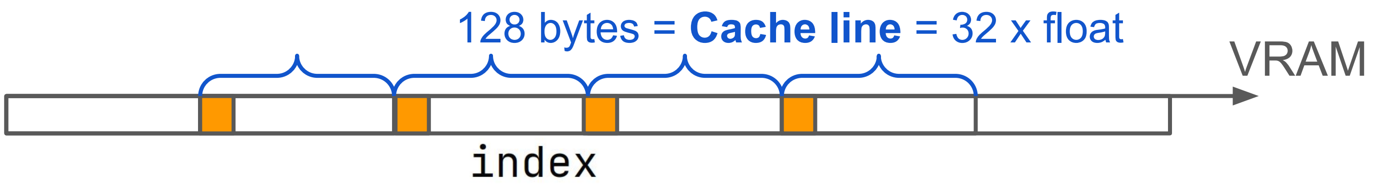
```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	???

Сколько cache line-ов чтобы покрыть заказ?

Сколько потребуется **транзакций** из VRAM?

Архитектура VRAM (**coalesced** memory access pattern)




```
float value = data[index];
```

index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	32 транзакции (**uncoalesced**)

Насколько просела достигнутая полезная пропускная способность VRAM?

Архитектура VRAM (**coalesced** memory access pattern)

$\overbrace{}^{\text{128 bytes = Cache line = 32 x float}}$

VRAM

index

Почему именно 128 байт?

```
float value = data[index];
```

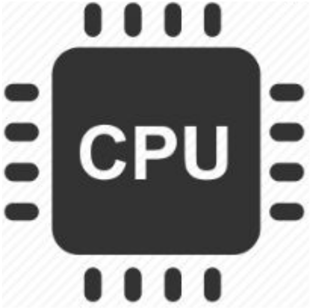
index	cache lines count
[1024+**0**; 1024+**32**)	1 транзакция (**coalesced**)
[1024+**1**; 1024+**33**)	2 транзакции (**coalesced**)
{1024 + **i*32**}	32 транзакции (**uncoalesced**)

Глава 3: Общая картина ЭВМ-архитектуры

CPU - RAM - PCI-E - VRAM - GPU

Архитектура

$100 \cdot 10^{9}$ FLOPS

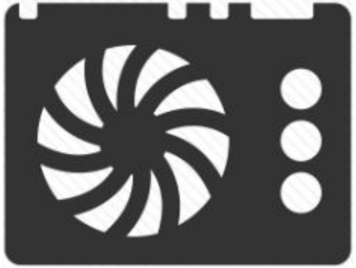


40 GB/s

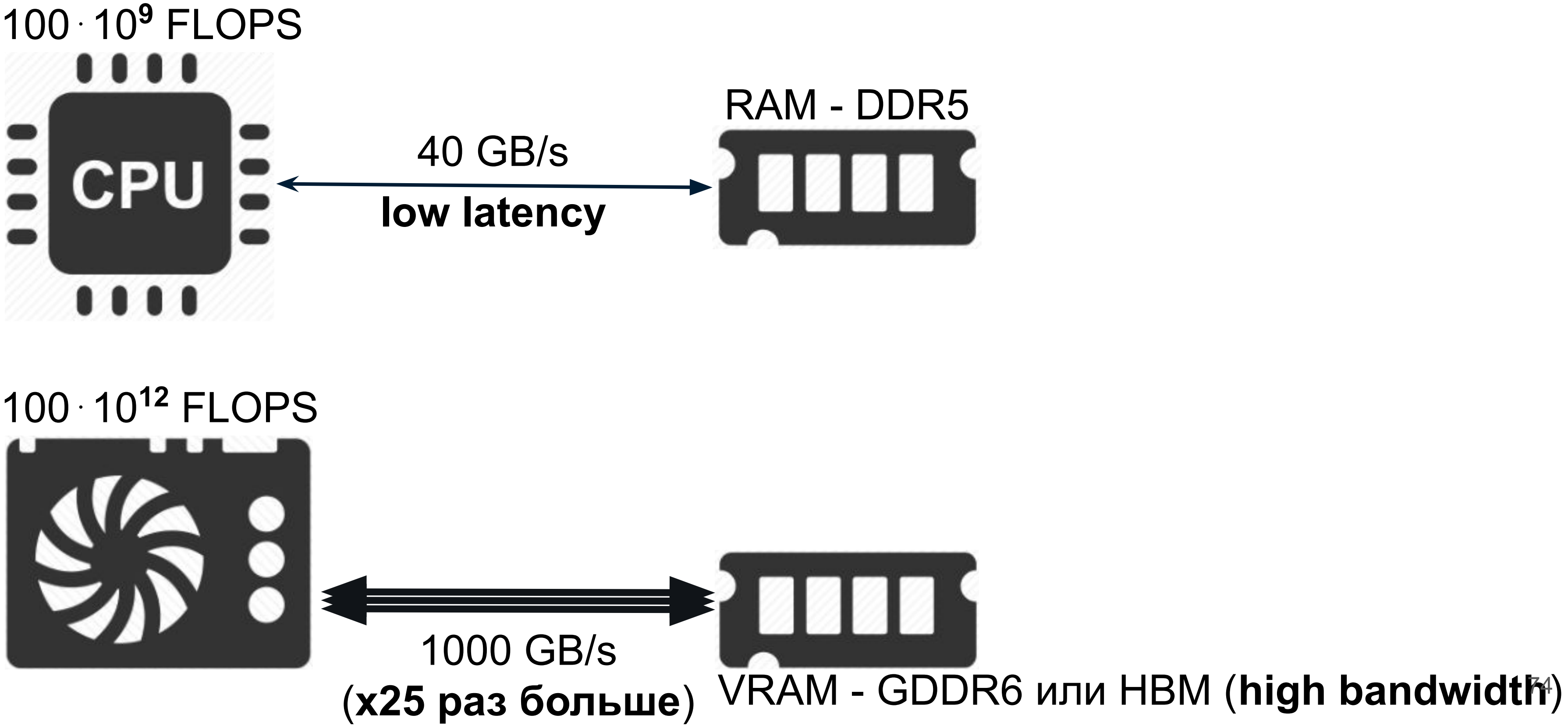
low latency

RAM - DDR5

$100 \cdot 10^{12}$ FLOPS



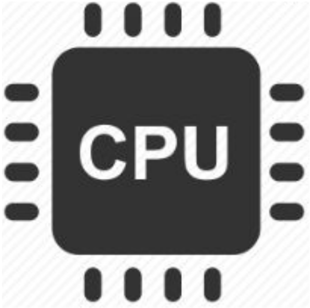
Архитектура



Архитектура

Где лучше исполнять OS?
(например реагировать на клики пользователя)
Какие есть метрики качества?

$100 \cdot 10^9$ FLOPS

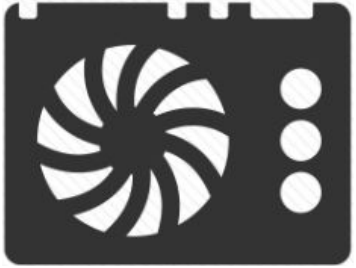


40 GB/s

low latency

RAM - DDR5

$100 \cdot 10^{12}$ FLOPS

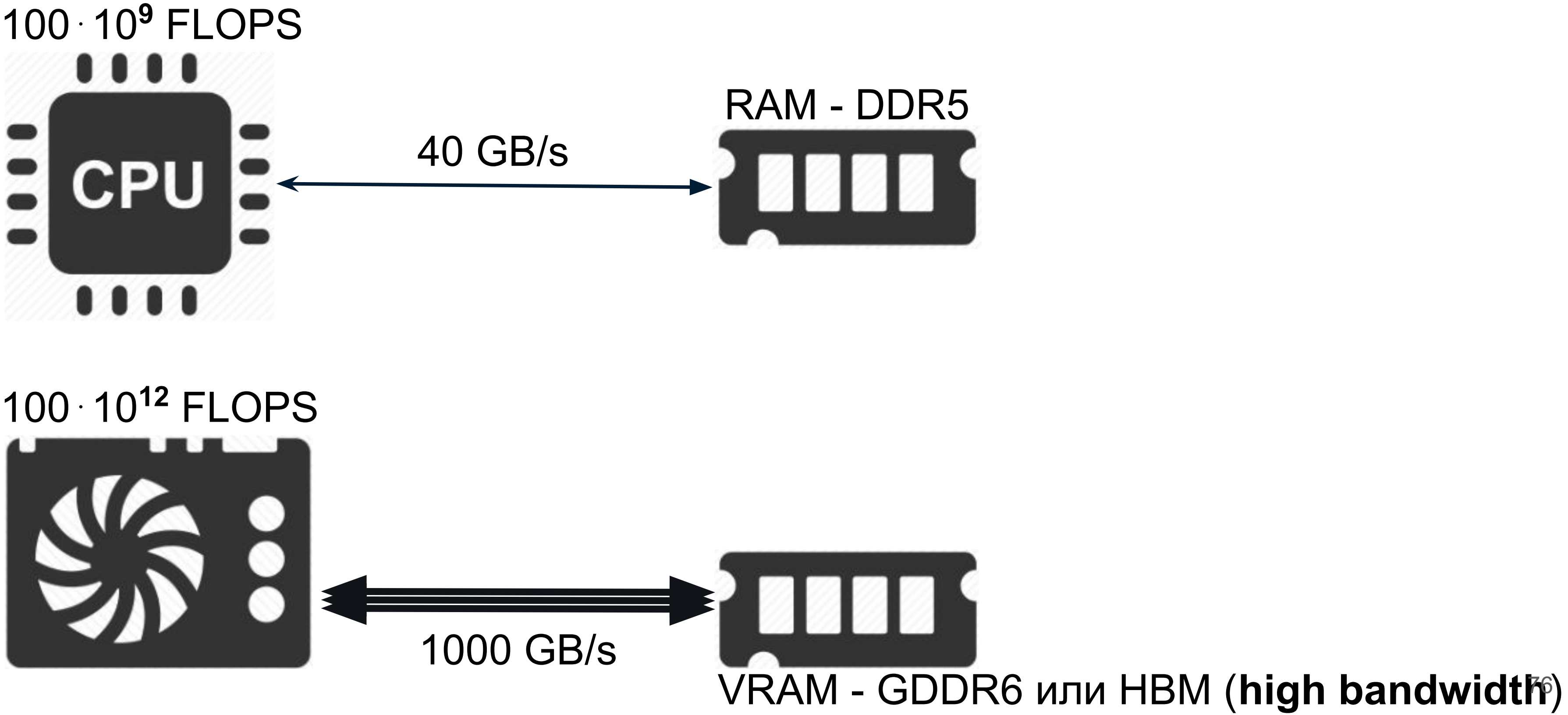


1000 GB/s
(**x25 раз больше**)

VRAM - GDDR6 или HBM (**high bandwidth**)

Архитектура

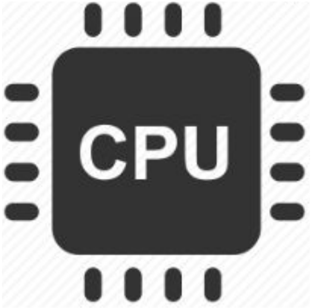
Где быстрее сложить два массива чисел?



Архитектура

Где быстрее сложить два массива чисел?
И во что мы упираемся - в память или в ALUs?
(ALUs = Arithmetic Logic Units)

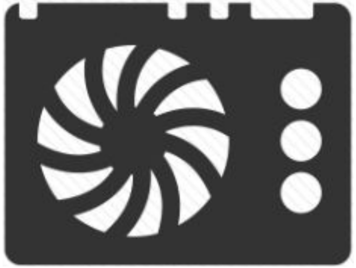
$100 \cdot 10^9$ FLOPS



40 GB/s

RAM - DDR5

$100 \cdot 10^{12}$ FLOPS

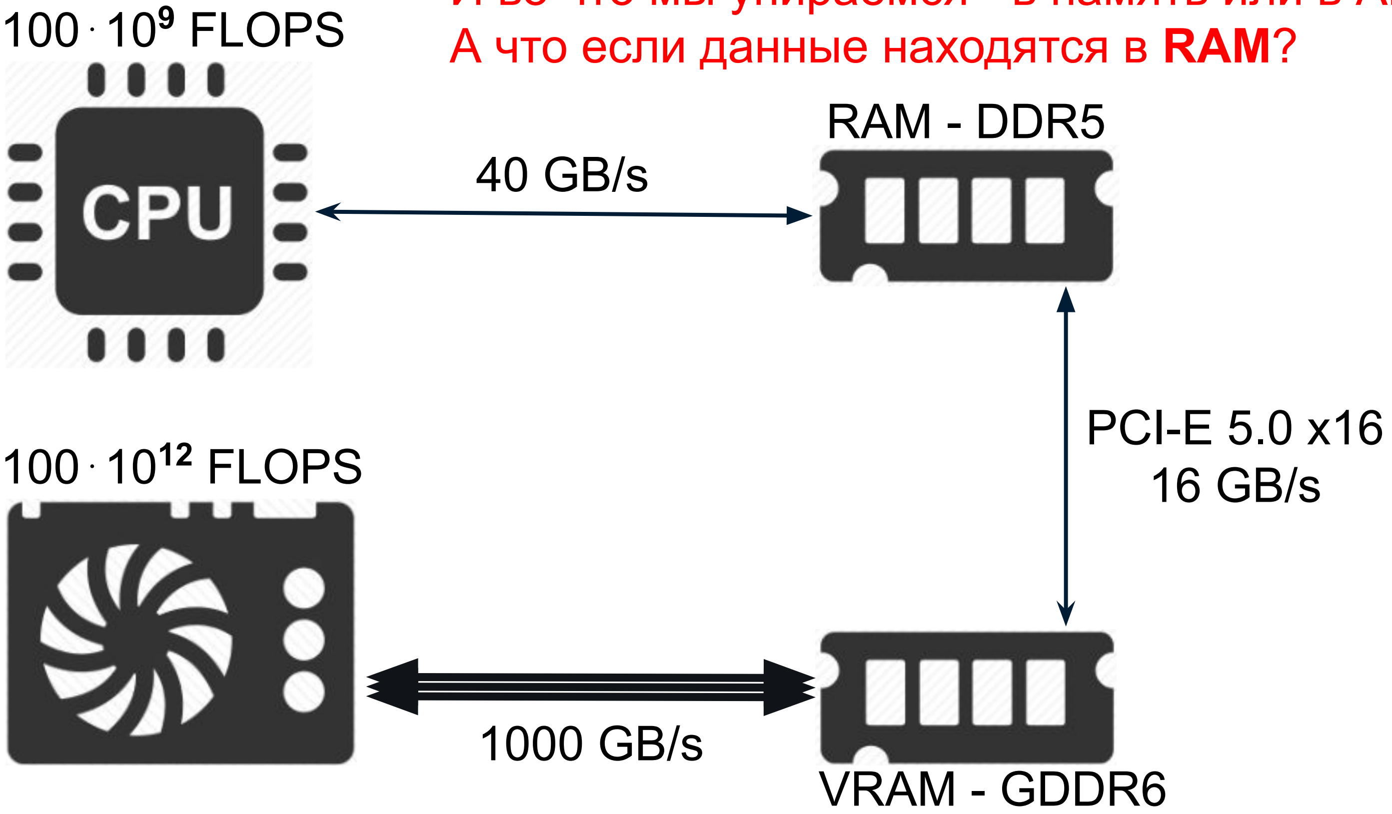


1000 GB/s

VRAM - GDDR6 или HBM (**high bandwidth**)

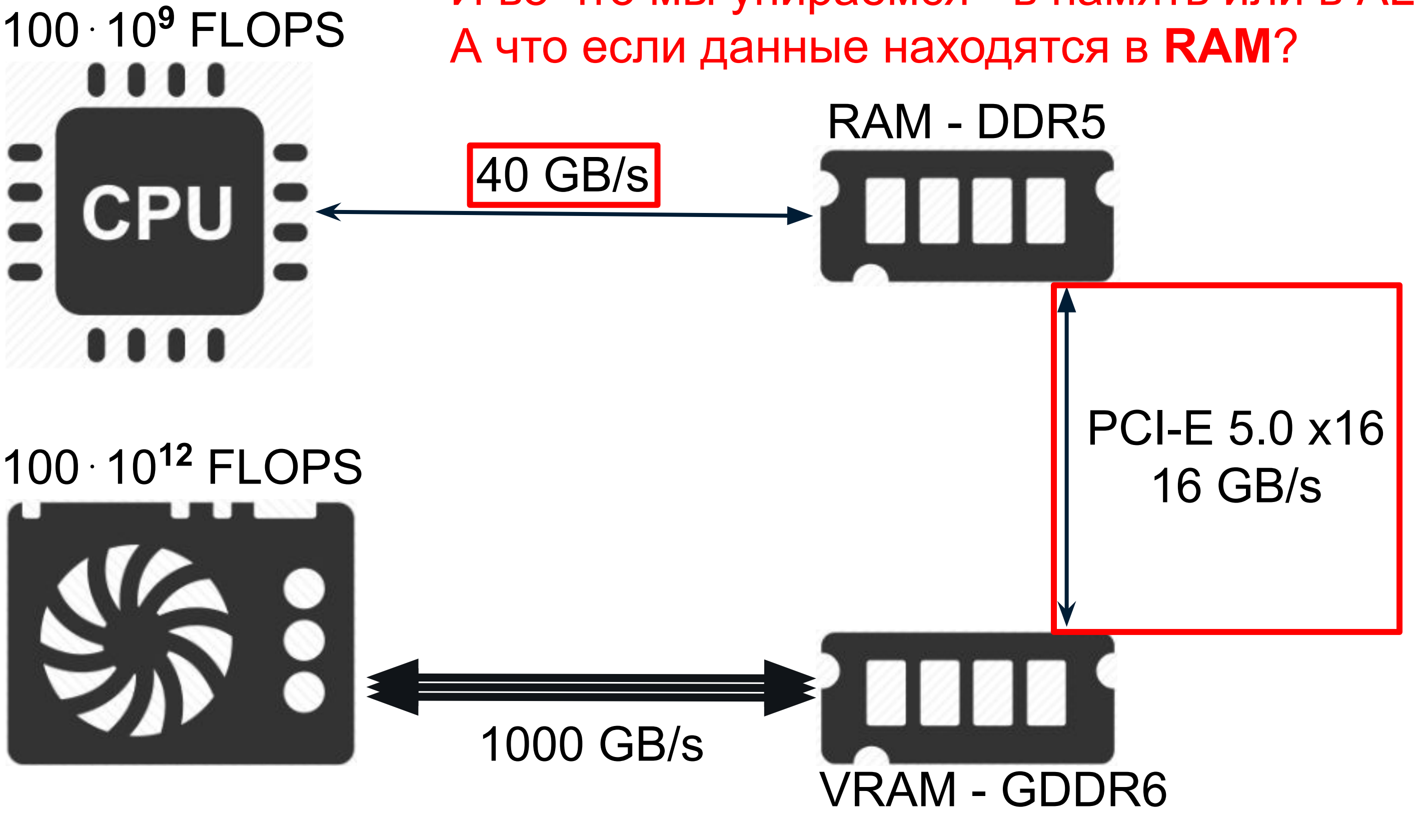
Архитектура

Где быстрее сложить два массива чисел?
И во что мы упираемся - в память или в ALUs?
А что если данные находятся в **RAM**?

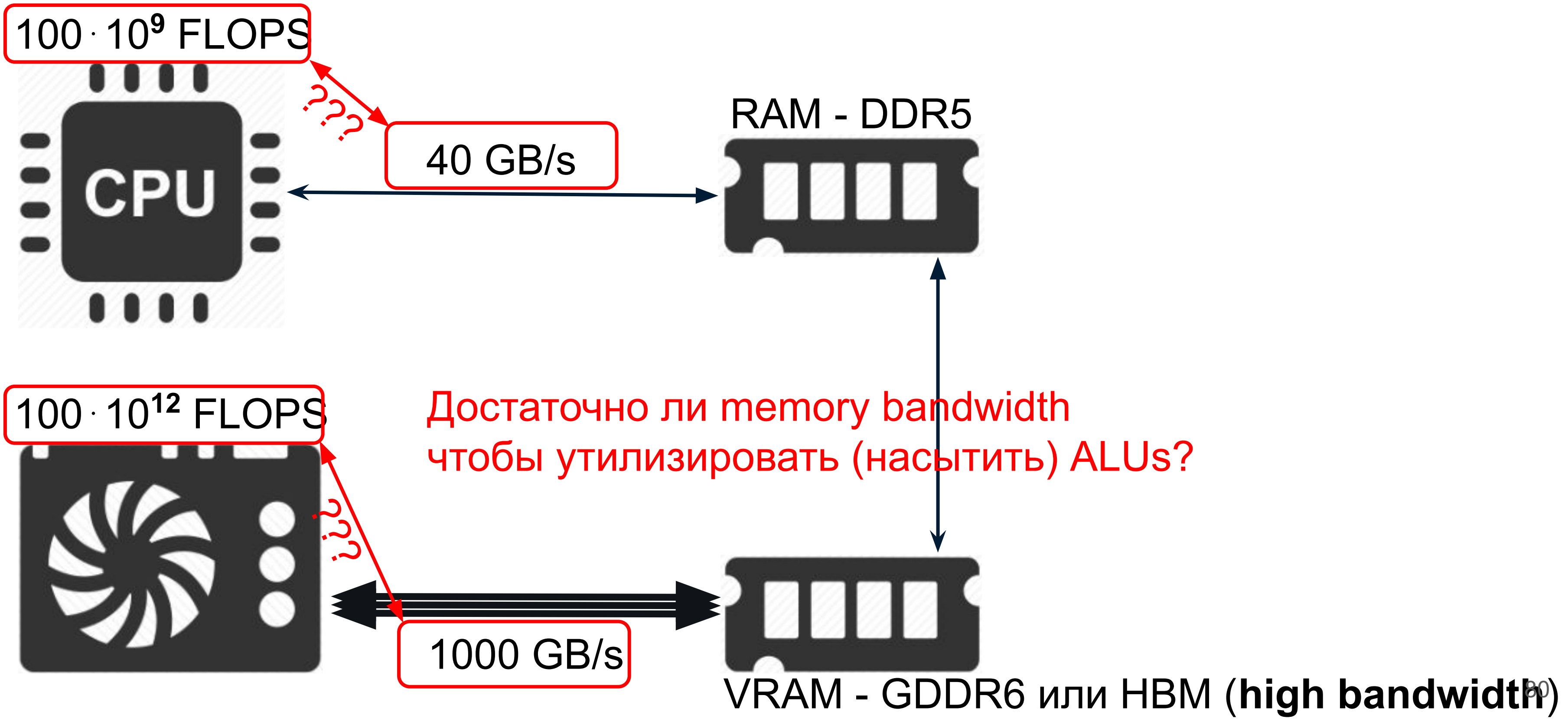


Архитектура

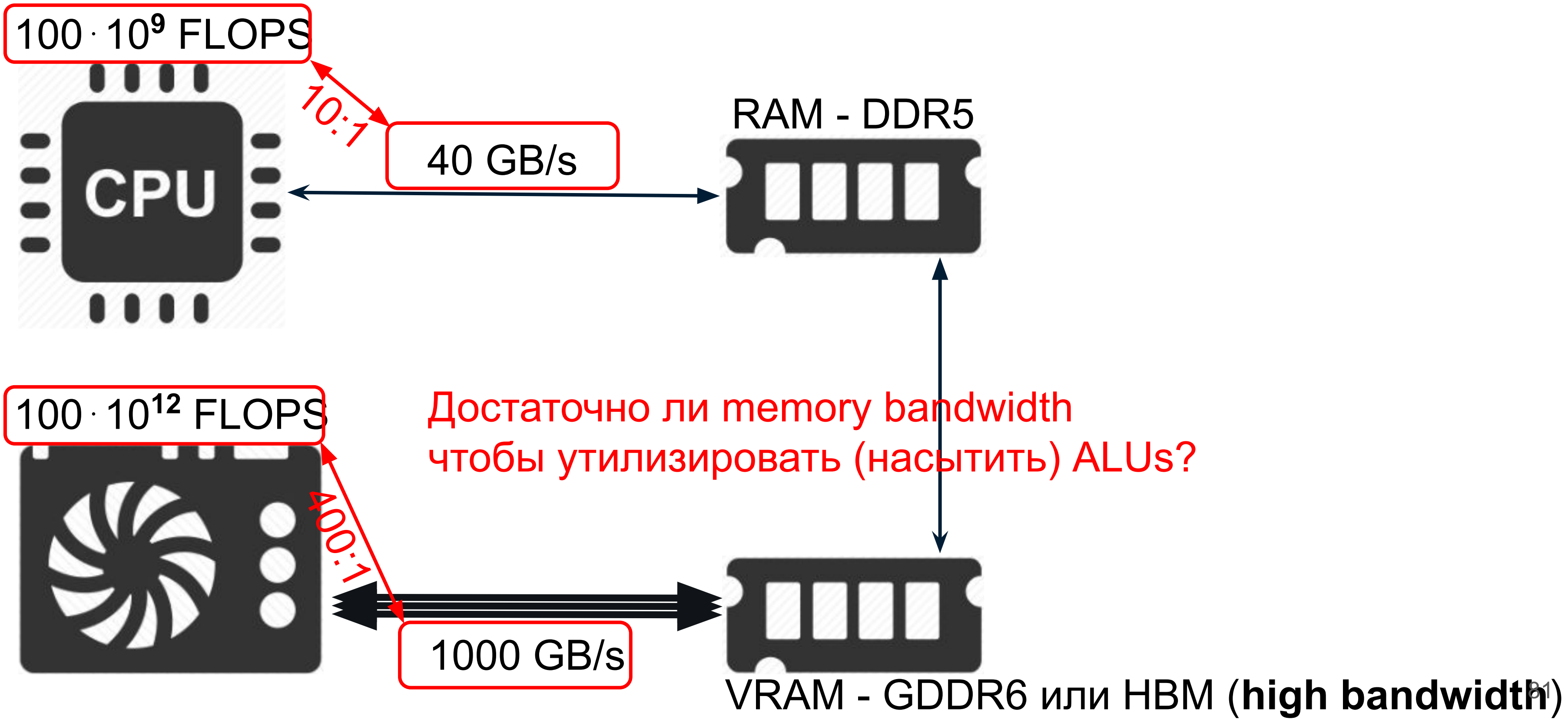
Где быстрее сложить два массива чисел?
И во что мы упираемся - в память или в ALUs?
А что если данные находятся в **RAM**?



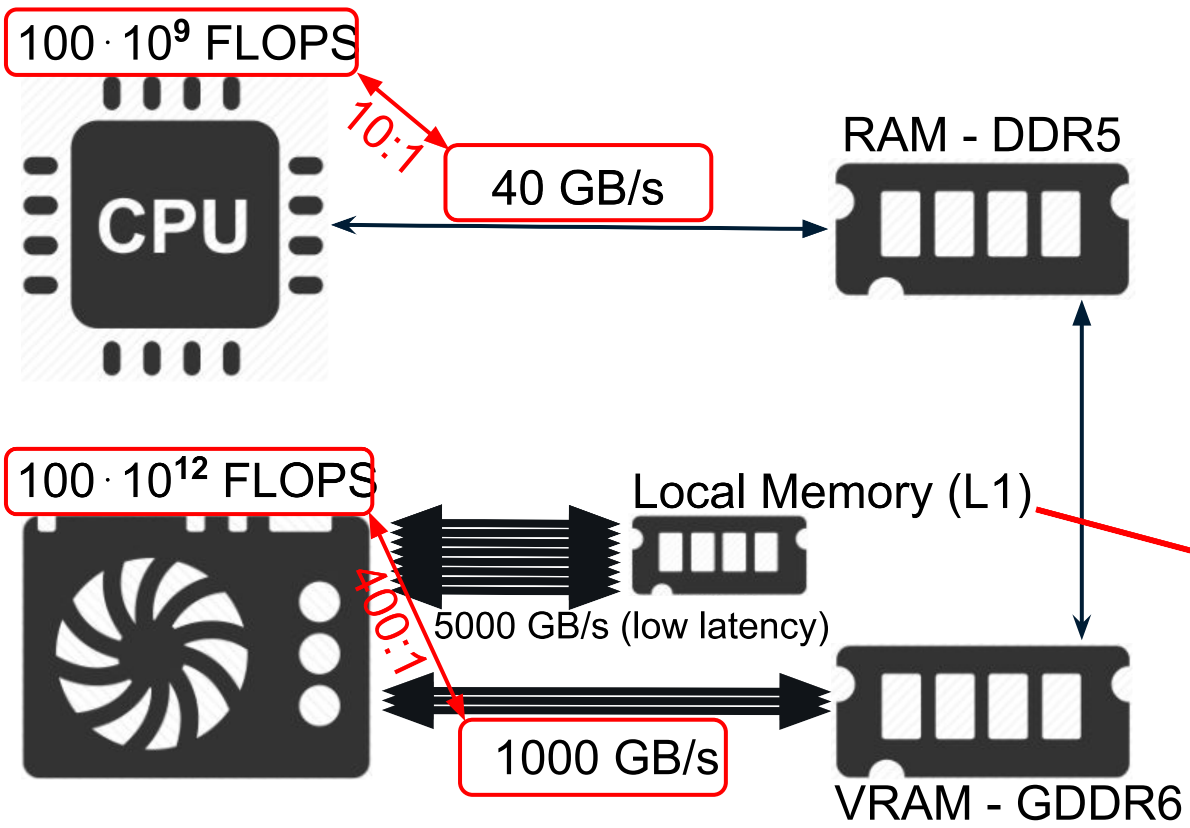
Архитектура



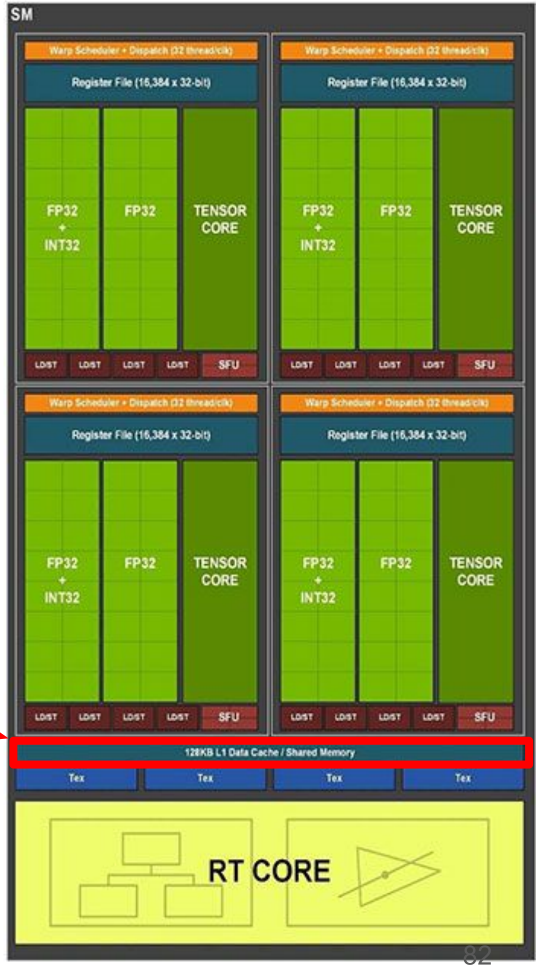
Архитектура



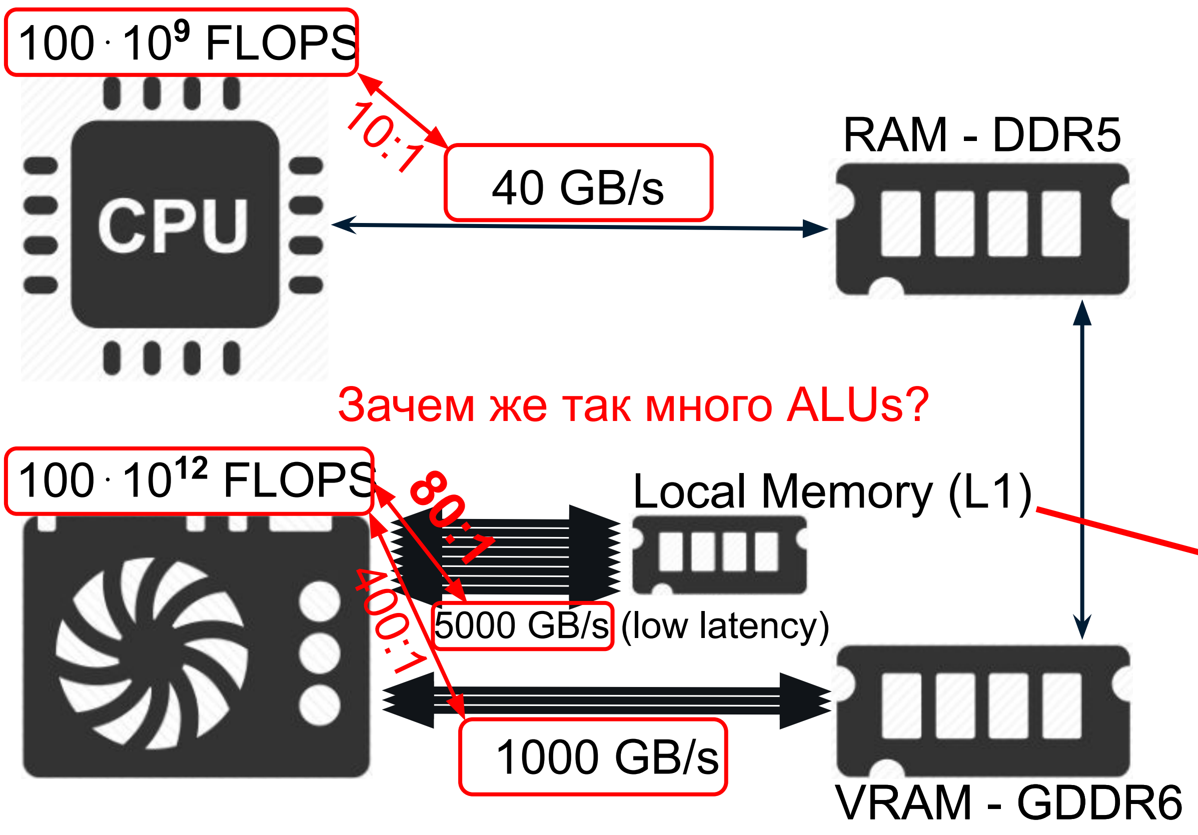
Архитектура



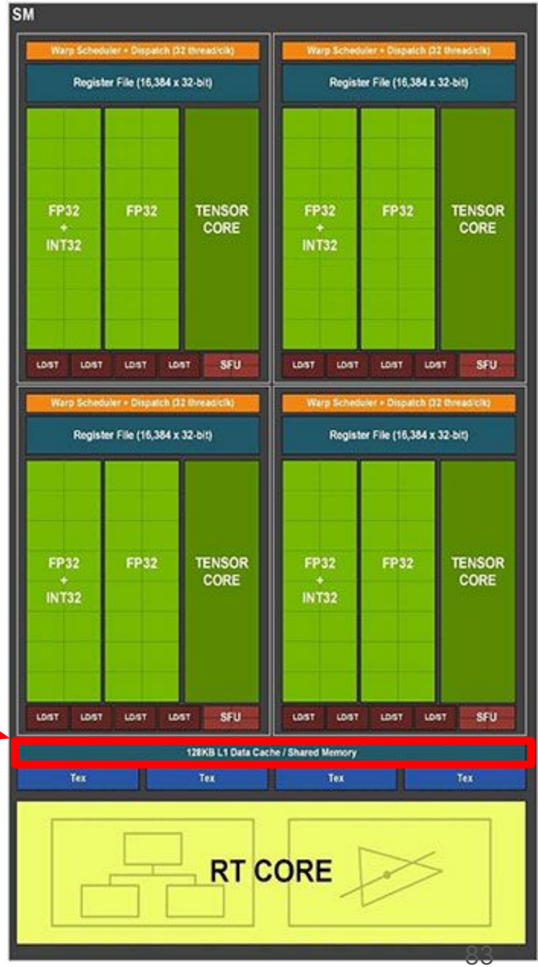
SM - Streaming Multiprocessor



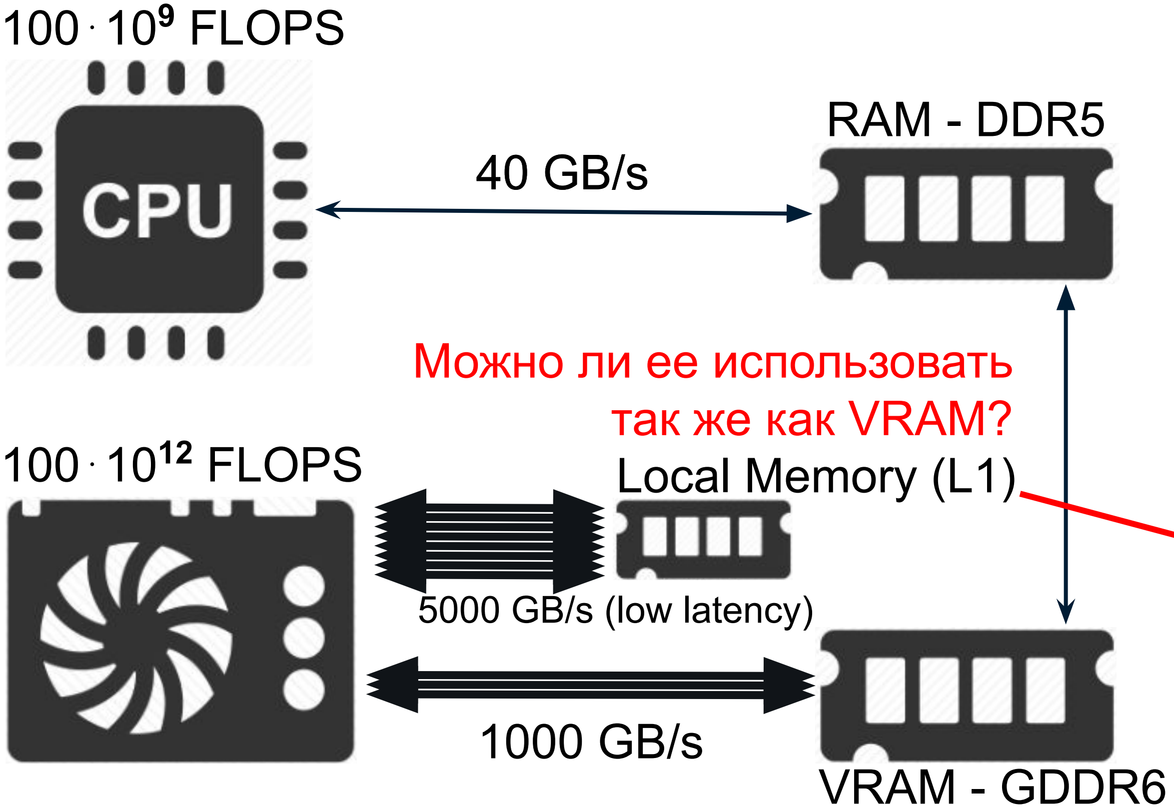
Архитектура



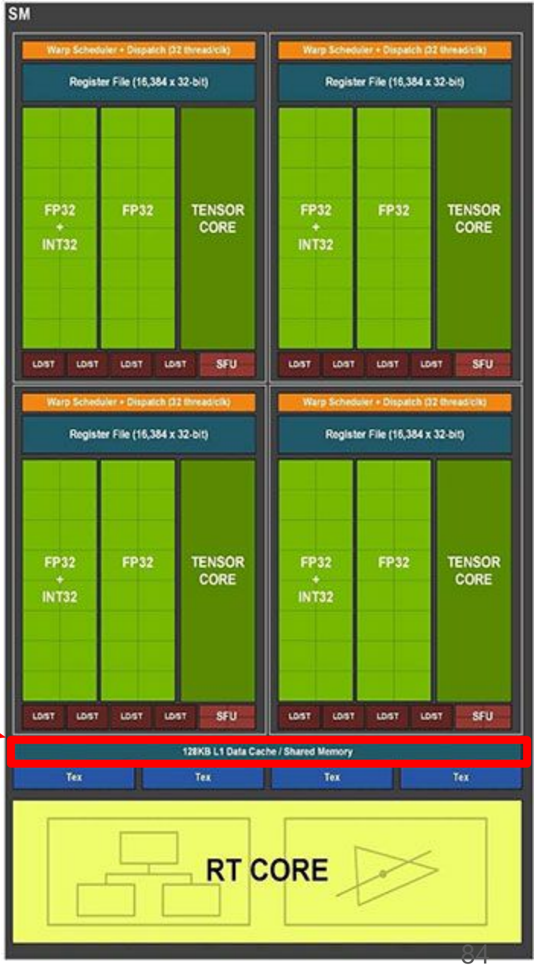
SM - Streaming Multiprocessor



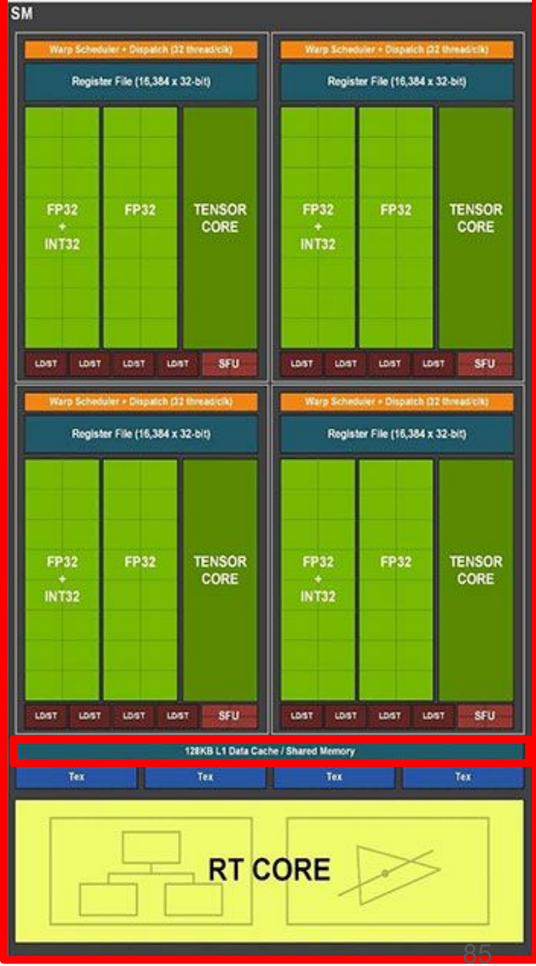
Архитектура



SM - Streaming Multiprocessor

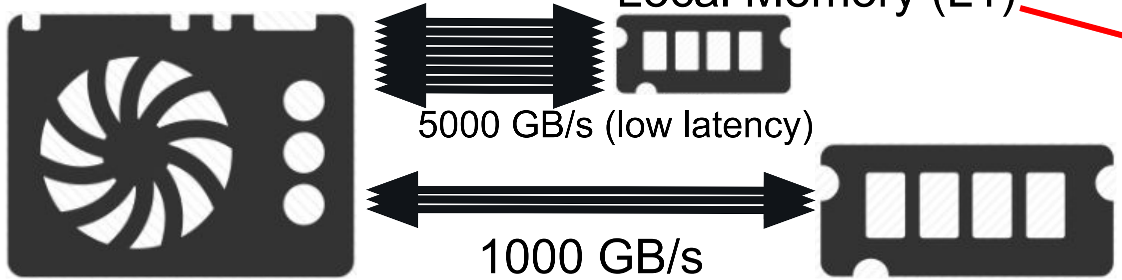


SM - Streaming Multiprocessor



$100 \cdot 10^{12}$ FLOPS

Local Memory (L1)

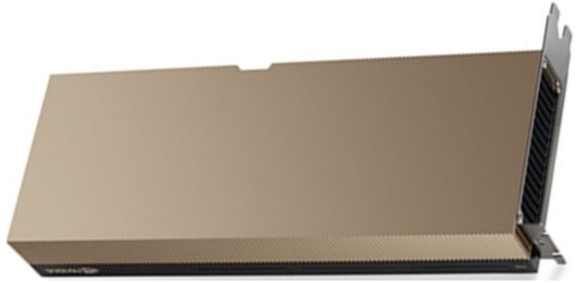


5000 GB/s (low latency)

1000 GB/s

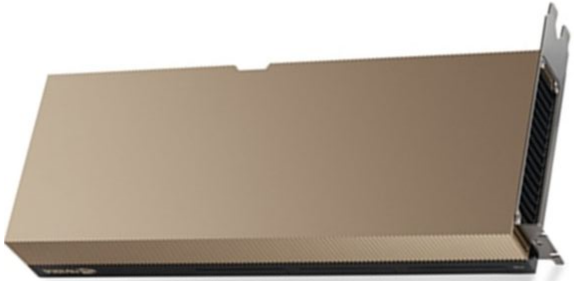
VRAM - GDDR6

Архитектура GPU



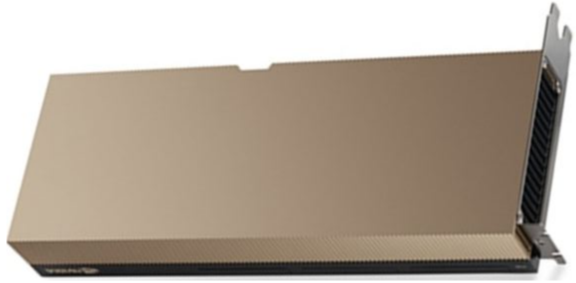
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)

Архитектура GPU



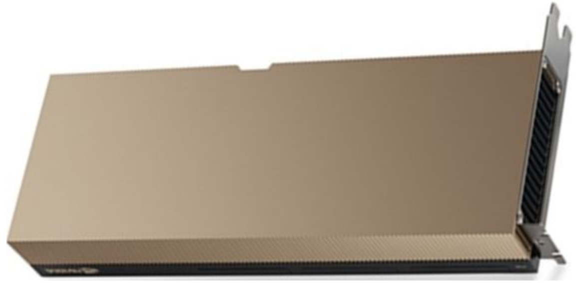
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра

Архитектура GPU



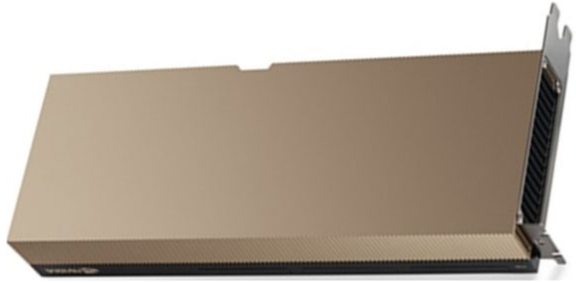
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)

Архитектура GPU



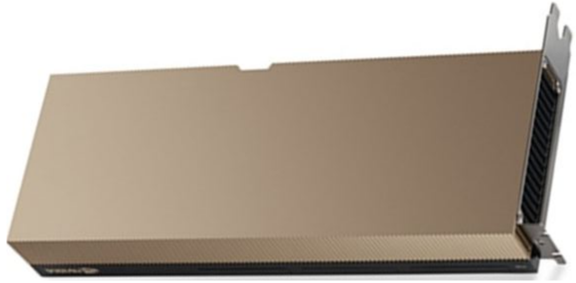
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)

Архитектура GPU



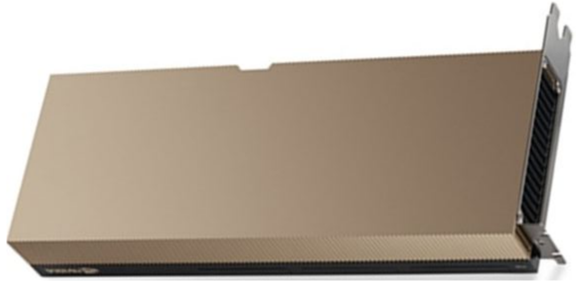
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**

Архитектура GPU



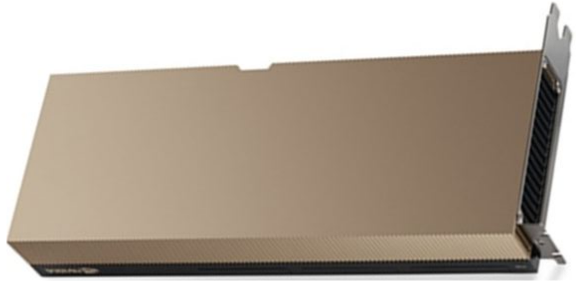
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)

Архитектура GPU



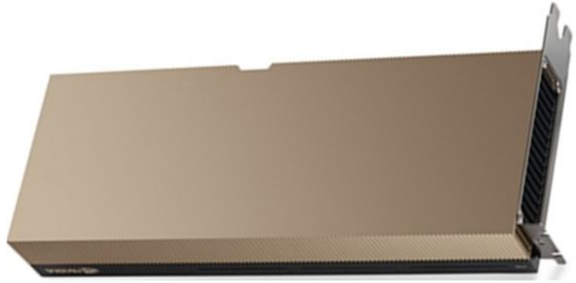
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)

Архитектура GPU

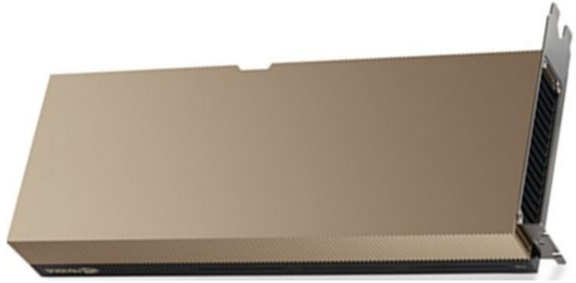


1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**

Архитектура GPU



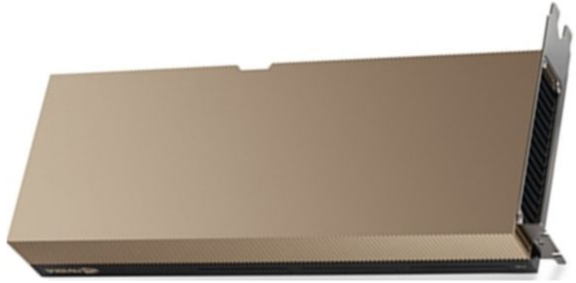
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность



Архитектура GPU

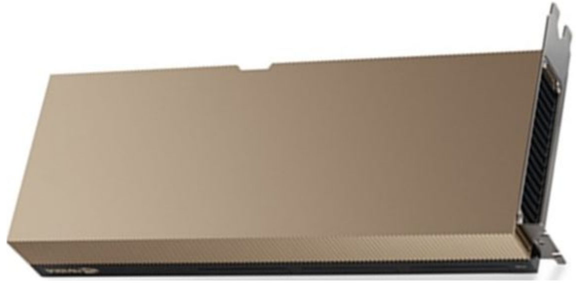
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):

Архитектура GPU



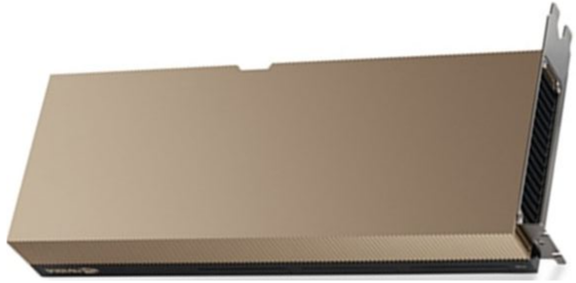
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка

Архитектура GPU



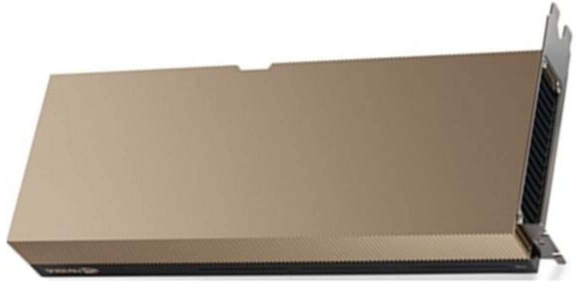
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном

Архитектура GPU



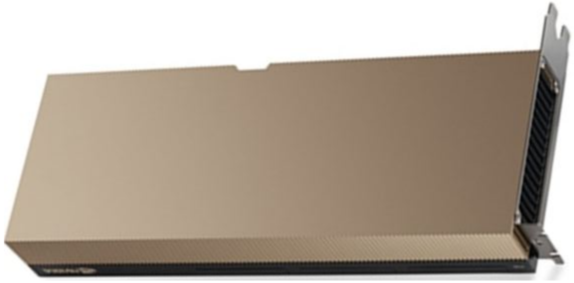
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном
   - **но?**

Архитектура GPU



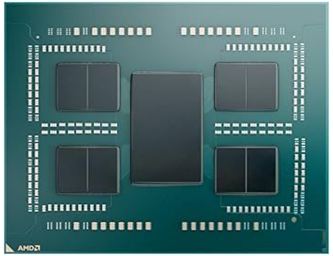
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном
   - но **небольшая** + **локальная** для каждого ~~**warp**~~ **WorkGroup=Block**

Архитектура GPU



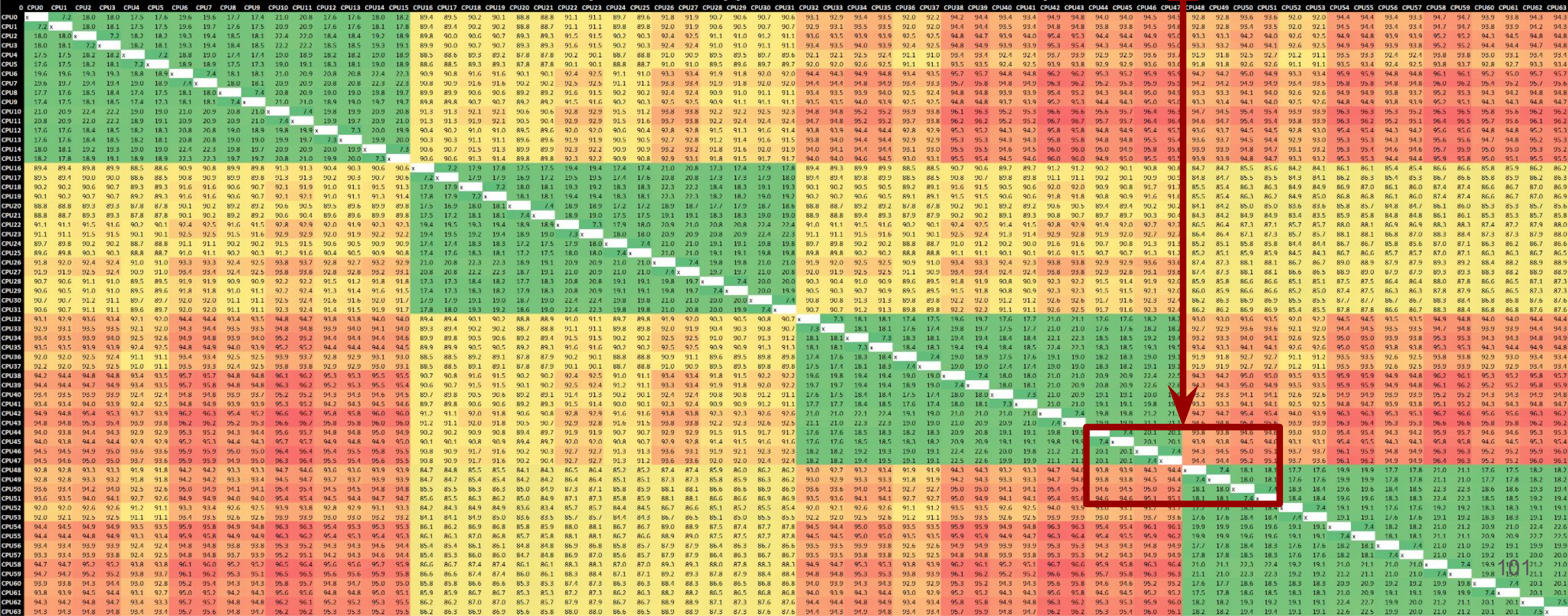
1) Десятки тысяч ядер (много **FLOPs**):
   - сгруппированы по 32 в **warp**-ы (по 64 в **wavefront**-ы у AMD)
   - но слабые ядра
   - но у warp-а общий **instruction pointer** (опасность **code divergence**)
   - активен **warp** который не ждет данных из **VRAM** (при высокой **occupancy**)
2) Огромная пропускная способность **VRAM:**
   - но большая задержка (**latency**)
   - скрывается за счет **сверх-SMT**/Hyper-Threading (при высокой **occupancy**)
   - транзакции разбиты на 128-байтные **cache line**
   - но при **un-coalesced** memory access pattern - малая проп. способность
3) Локальная память (**local/shared memory**):
   - **титаническая** пропускная способность + низкая задержка
   - нет проблемы с coalesced-паттерном
   - но **небольшая** + **локальная** для каждого ~~**warp**~~ **WorkGroup=Block**

И в этом секрет успеха! Рецепт к **масштабируемости**!

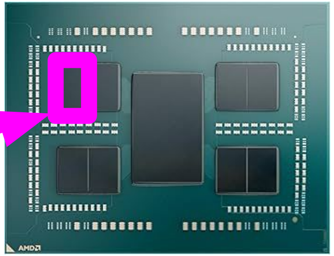


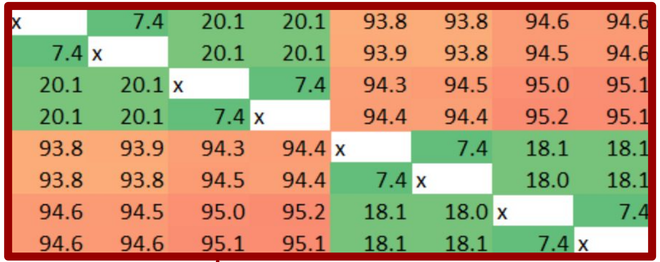
x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x

AMD Ryzen Threadripper 7980X Core-to-Core Latency

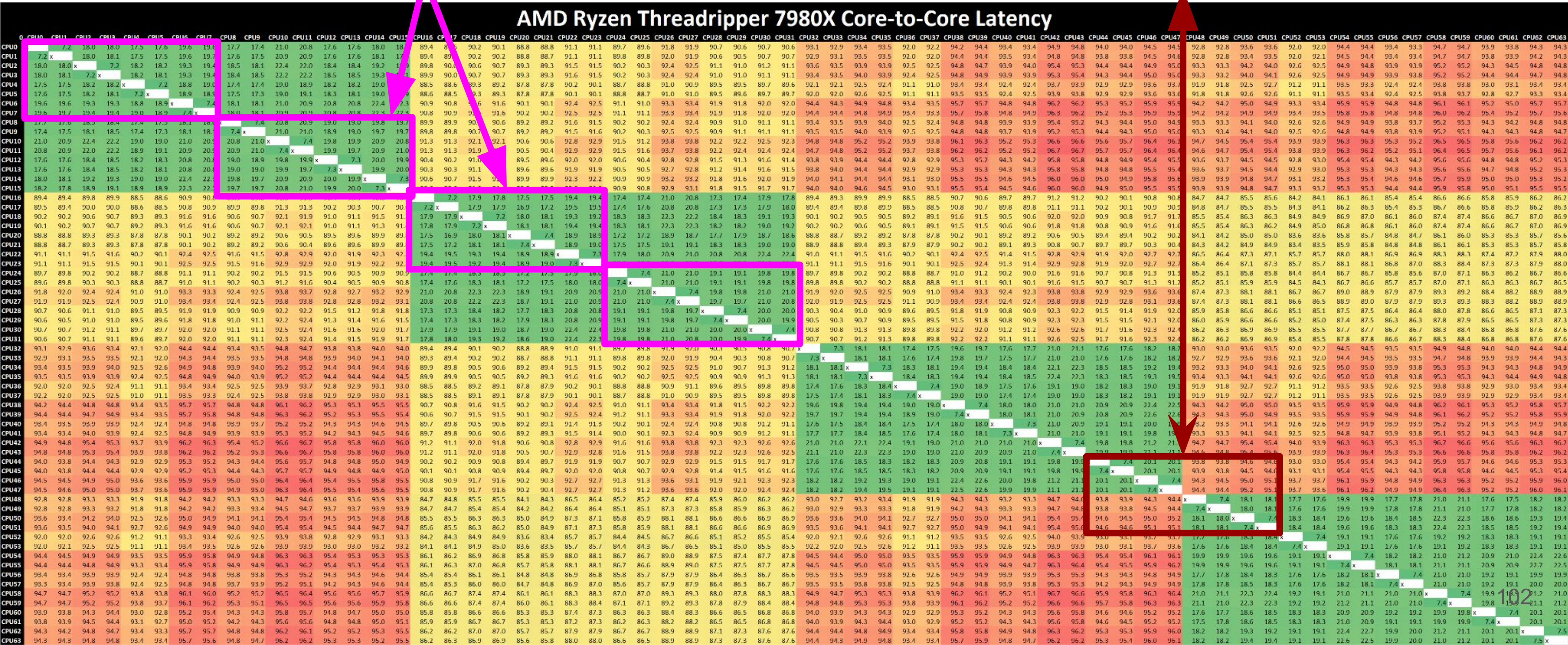


64 ядра = 8 x Чиплетов (по 8 ядер)



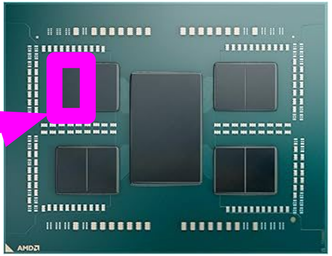


x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x

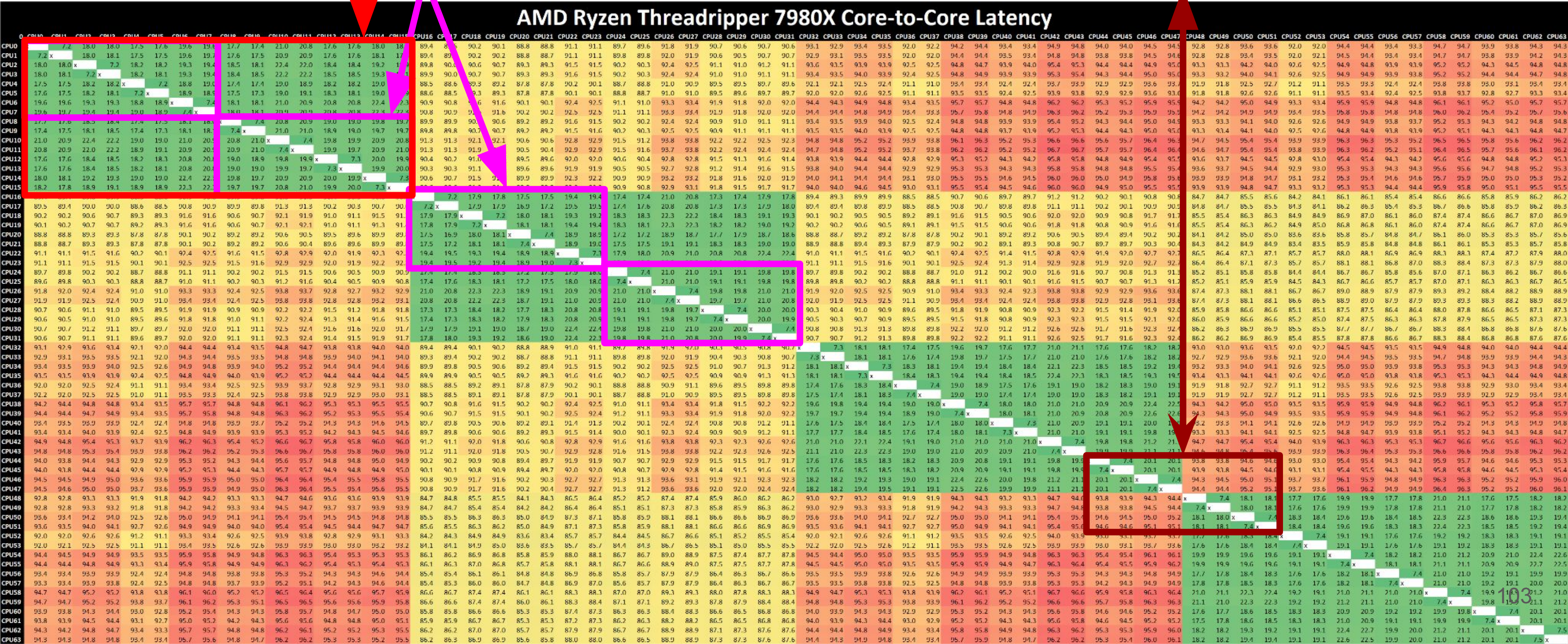


Но почему задержка маленькая группами по 16 ядер?

64 ядра = 8 x Чиплетов (по 8 ядер)

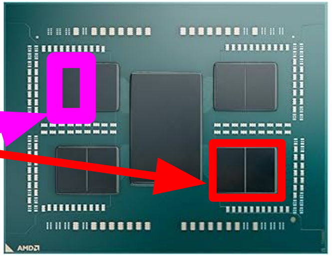


x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x

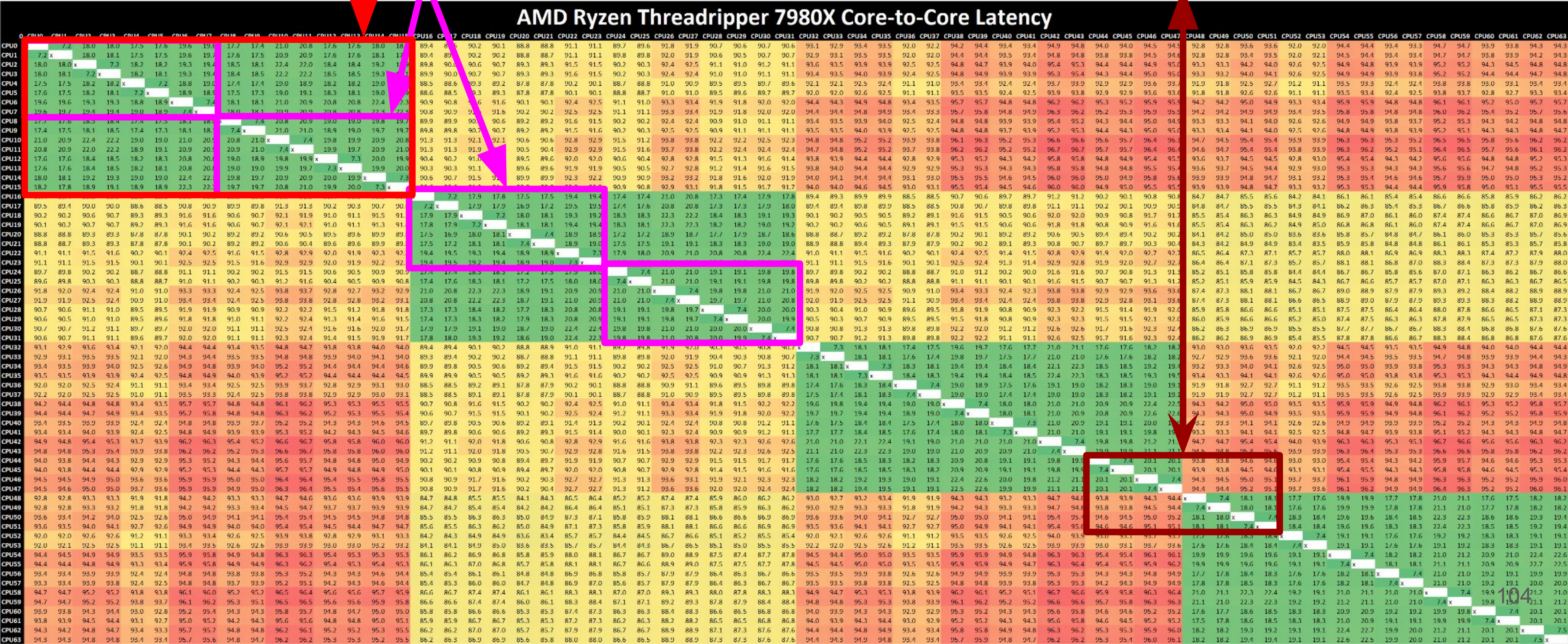


Но почему задержка маленькая группами по 16 ядер?

64 ядра = 8 x Чиплетов (по 8 ядер)

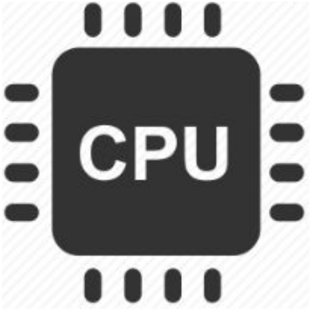


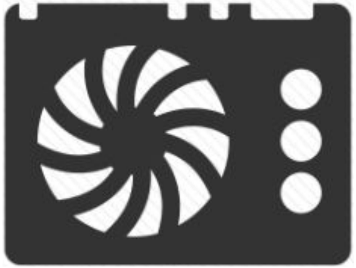
x	7.4	20.1	20.1	93.8	93.8	94.6	94.6
7.4	x	20.1	20.1	93.9	93.8	94.5	94.6
20.1	20.1	x	7.4	94.3	94.5	95.0	95.1
20.1	20.1	7.4	x	94.4	94.4	95.2	95.1
93.8	93.9	94.3	94.4	x	7.4	18.1	18.1
93.8	93.8	94.5	94.4	7.4	x	18.0	18.1
94.6	94.5	95.0	95.2	18.1	18.0	x	7.4
94.6	94.6	95.1	95.1	18.1	18.1	7.4	x

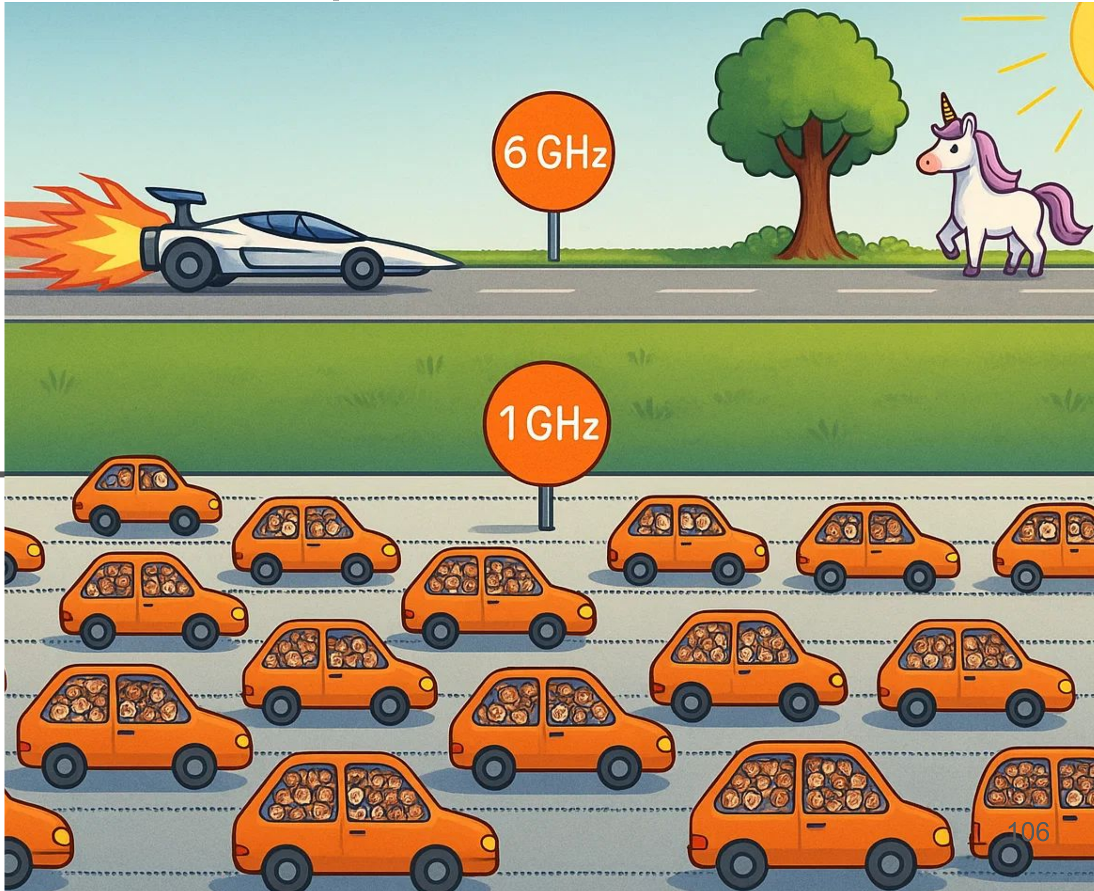


Глава 4: Модель вычислений массового параллелизма

Модель вычислений массового параллелизма

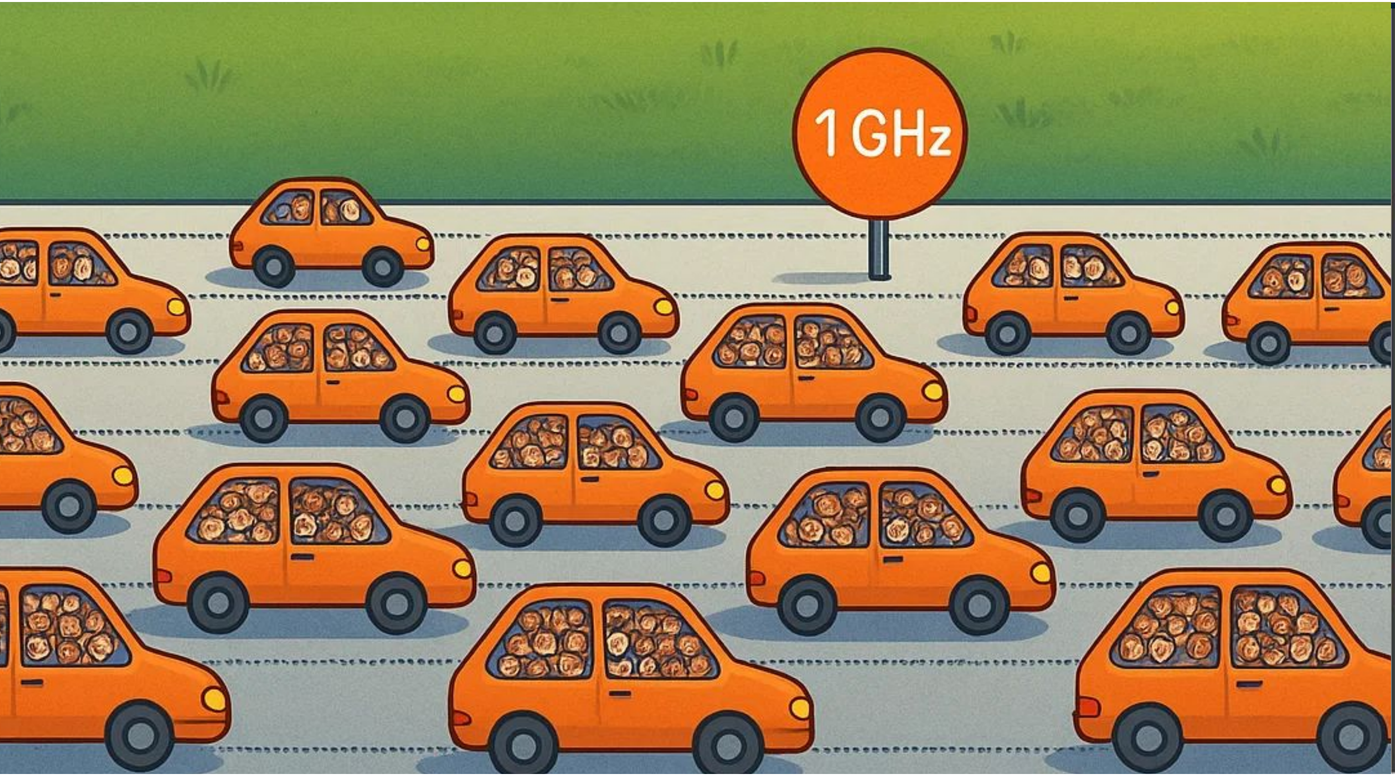






Модель вычислений массового параллелизма

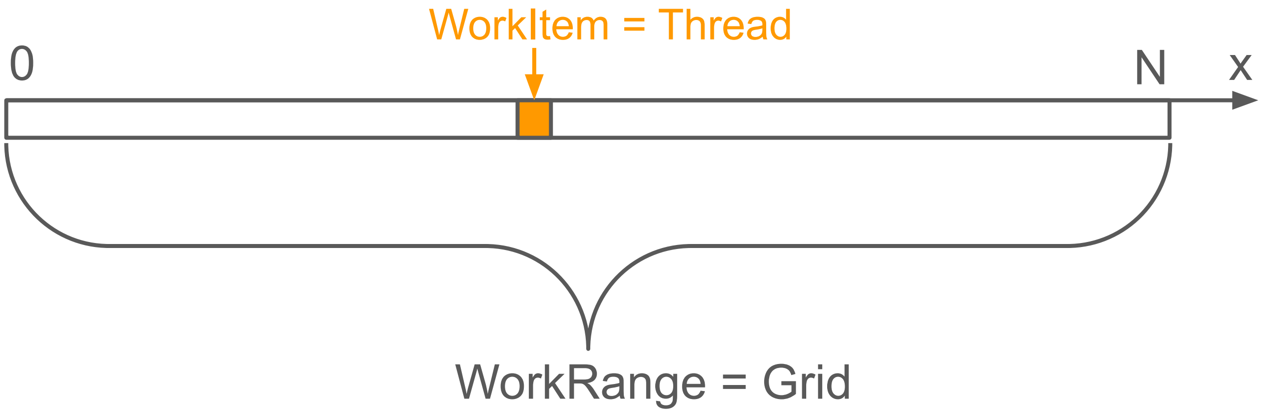
SM - Streaming Multiprocessor



Модель вычислений массового параллелизма

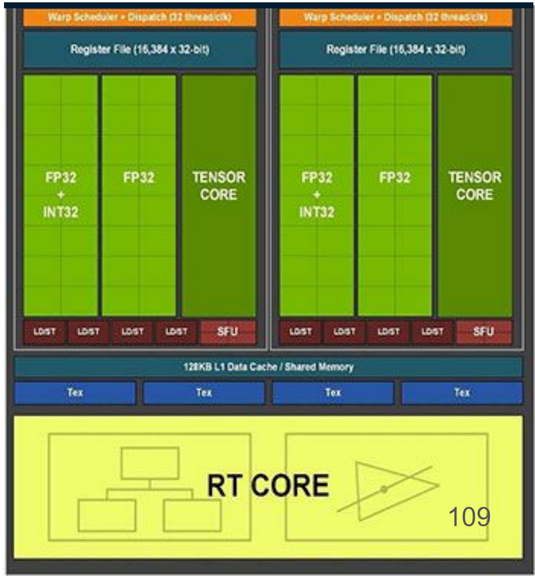
SM - Streaming Multiprocessor

Модель вычислений массового параллелизма

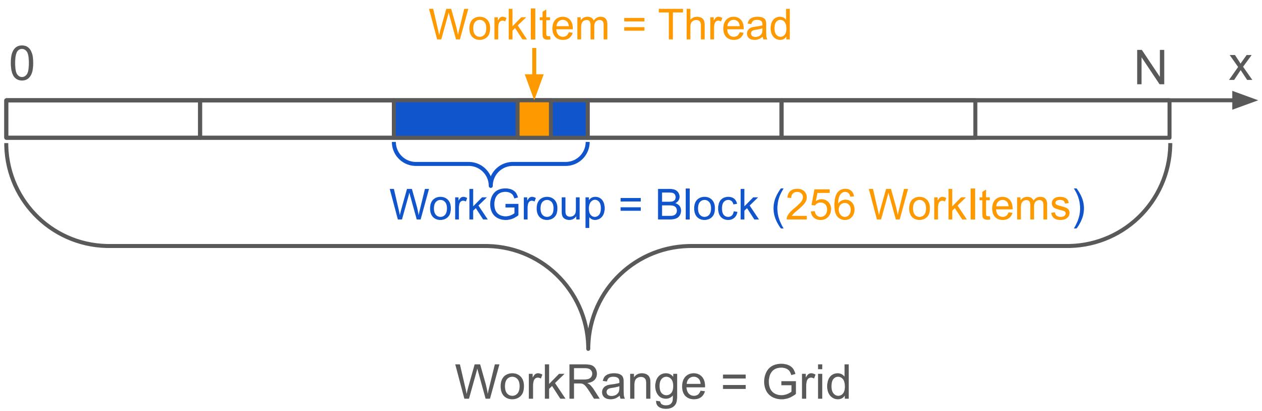


Много SM

SM - Streaming Multiprocessor

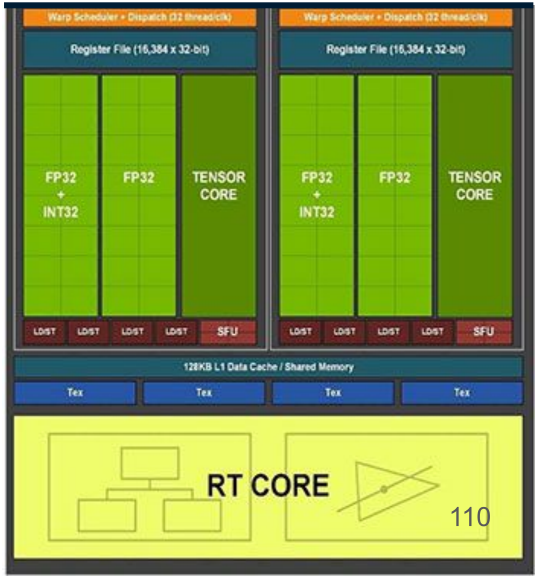


Модель вычислений массового параллелизма

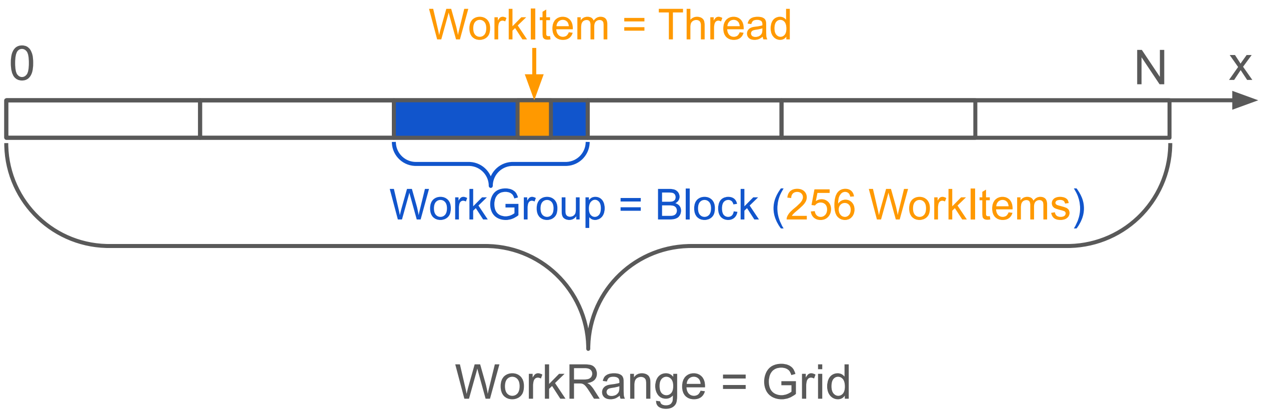


Много SM

SM - Streaming Multiprocessor

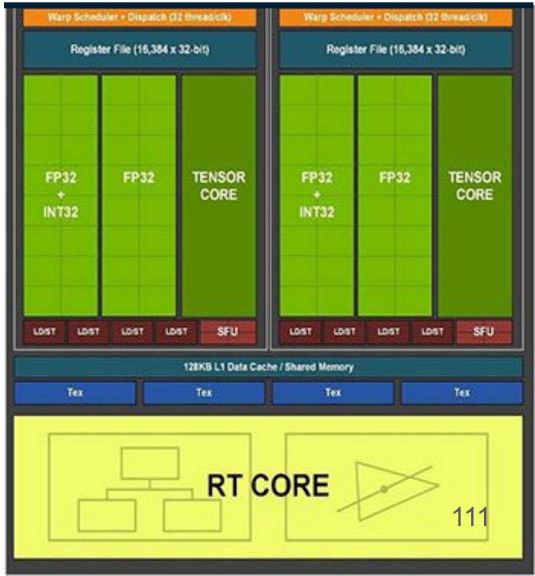


Модель вычислений массового параллелизма



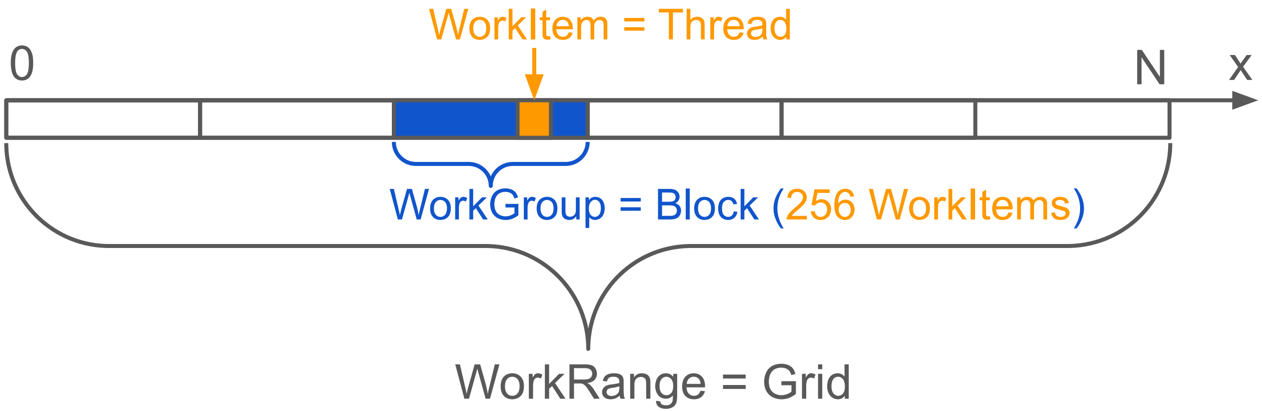
Много SM

SM - Streaming Multiprocessor



Но ведь в одном warp 32 потока!
(у **AMD**: 64 потока в wavefront)
Как так?

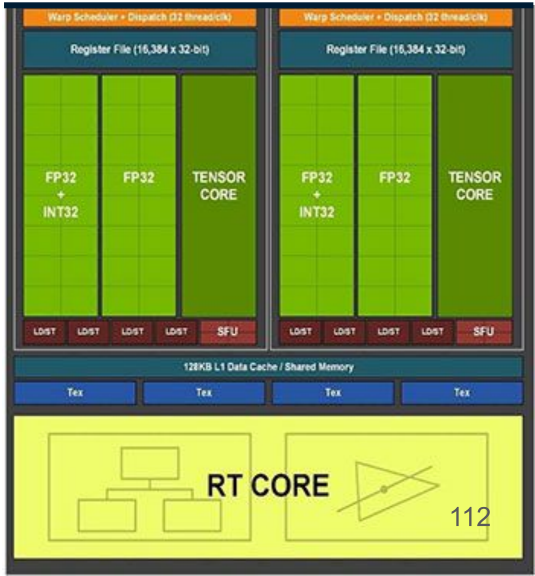
Модель вычислений массового параллелизма



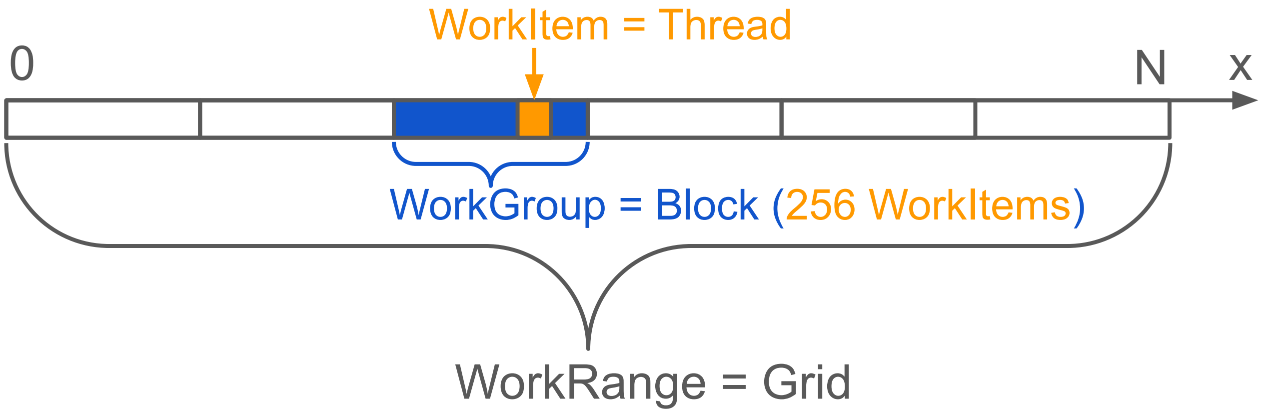
WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

Много SM

SM - Streaming Multiprocessor

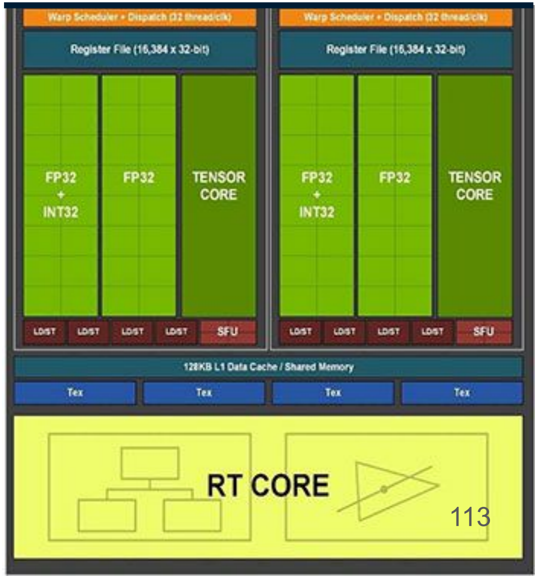


Модель вычислений массового параллелизма



Много SM

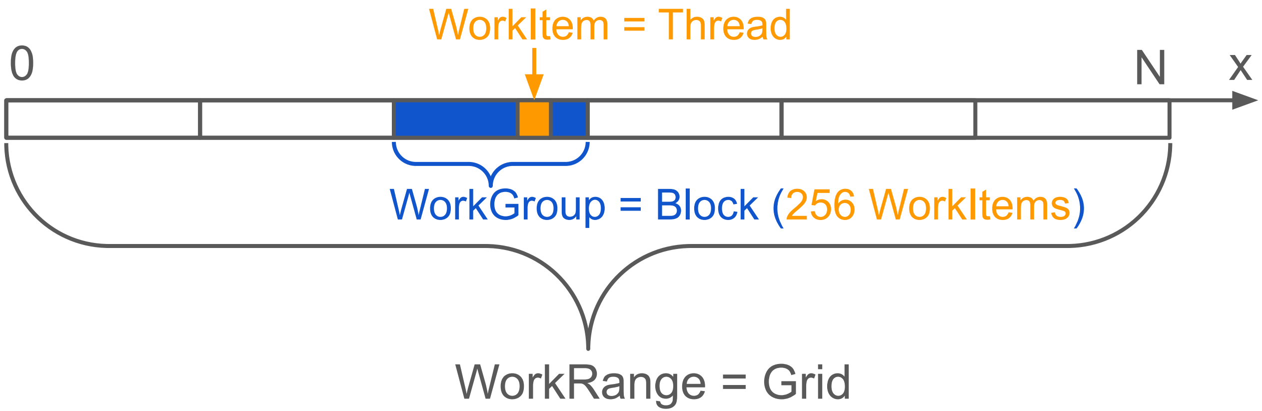
SM - Streaming Multiprocessor



WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

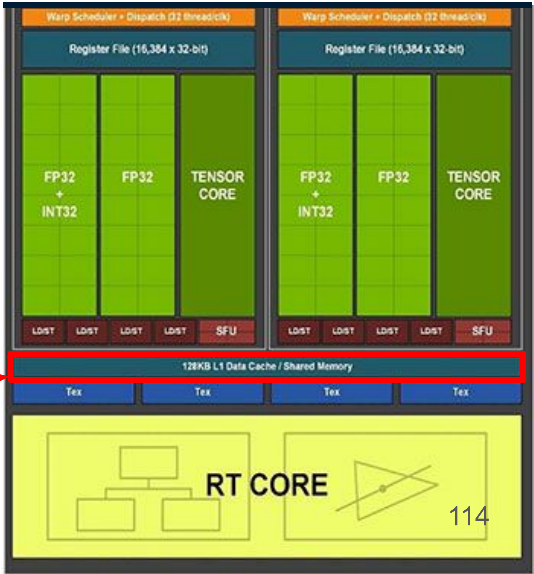
Зачем нам контроль над **WorkGroups**?
Зачем на уровне API знать про **warps**?

Модель вычислений массового параллелизма



Много SM

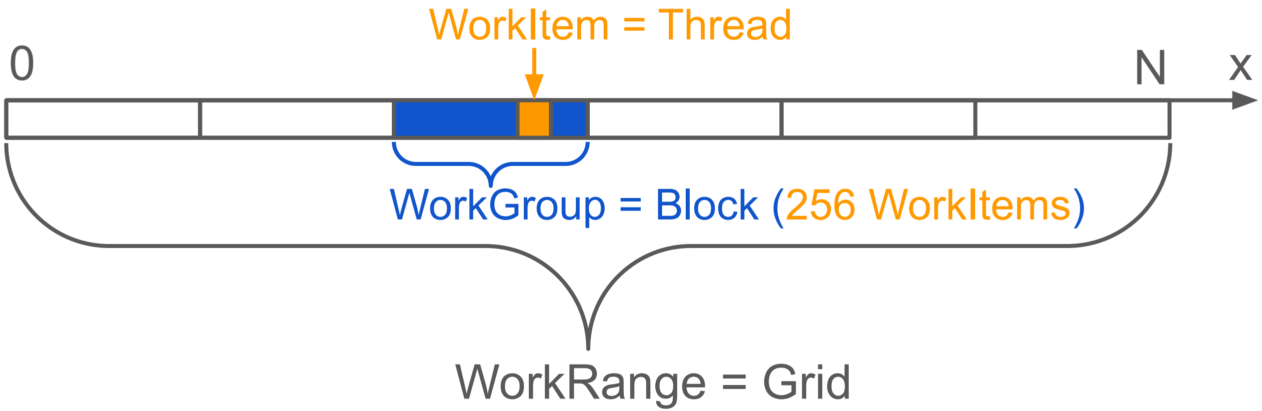
SM - Streaming Multiprocessor



WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

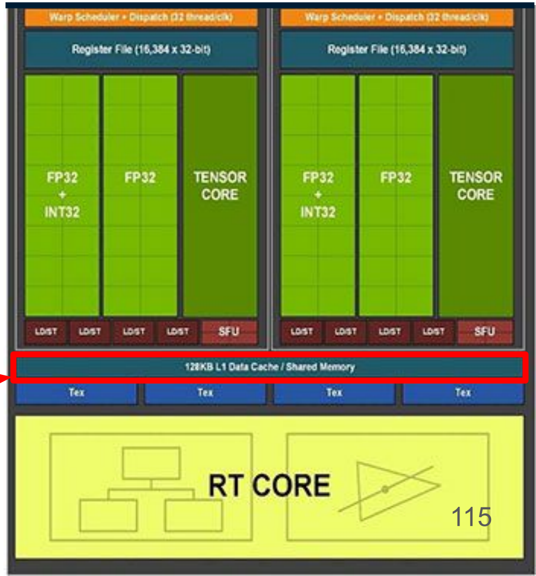
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

Модель вычислений массового параллелизма



Много SM

SM - Streaming Multiprocessor

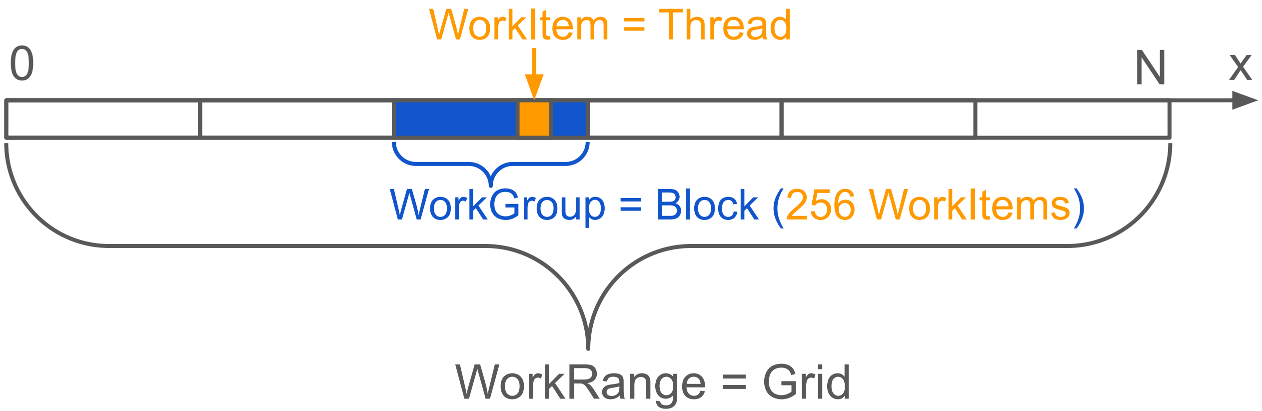


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

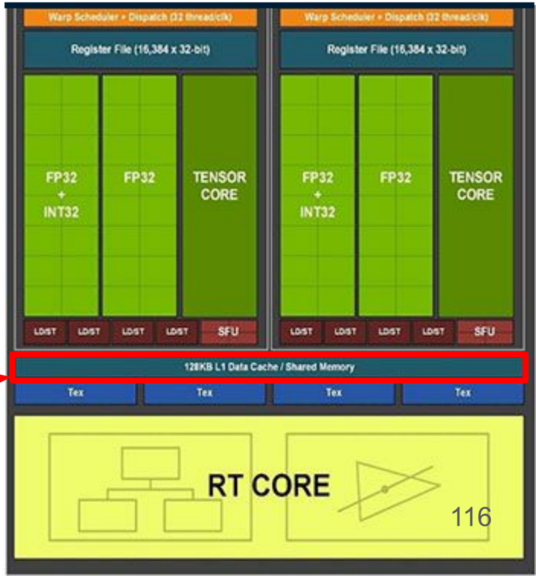
Могут ли **warp**-ы одной **WorkGroup** исполняться на разных SM?

Модель вычислений массового параллелизма



Много SM

SM - Streaming Multiprocessor

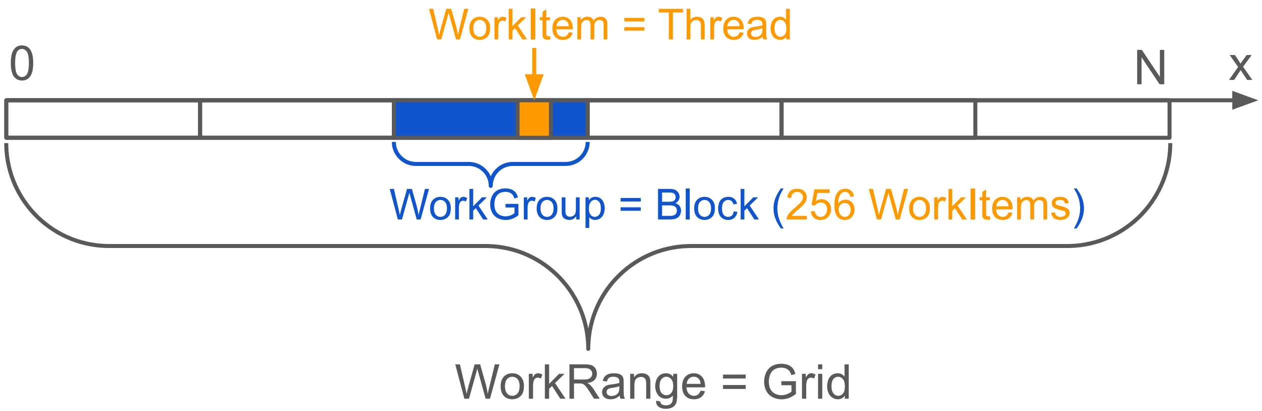


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

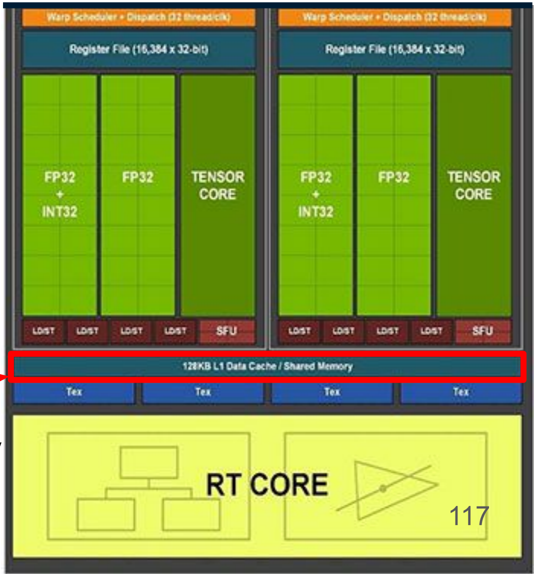
А если один такой поток записал в **Local Memory** число - увидит ли другой поток этой **WorkGroup**?

Модель вычислений массового параллелизма



Много SM

SM - Streaming Multiprocessor



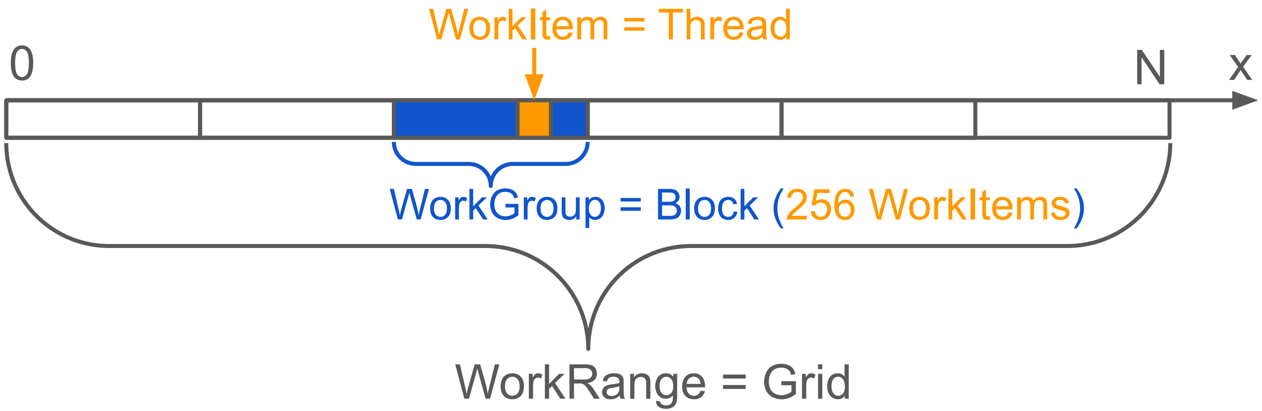
WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

А если один такой поток записал в **Local Memory** число - увидит ли другой поток этой **WorkGroup**?

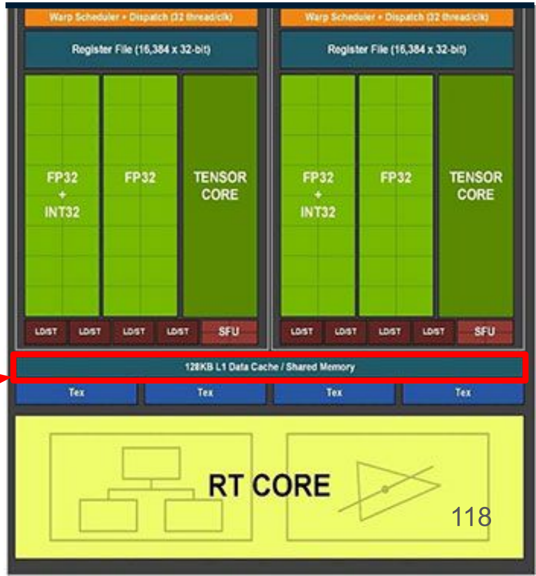
memory barrier!

Модель вычислений массового параллелизма



Много SM

SM - Streaming Multiprocessor

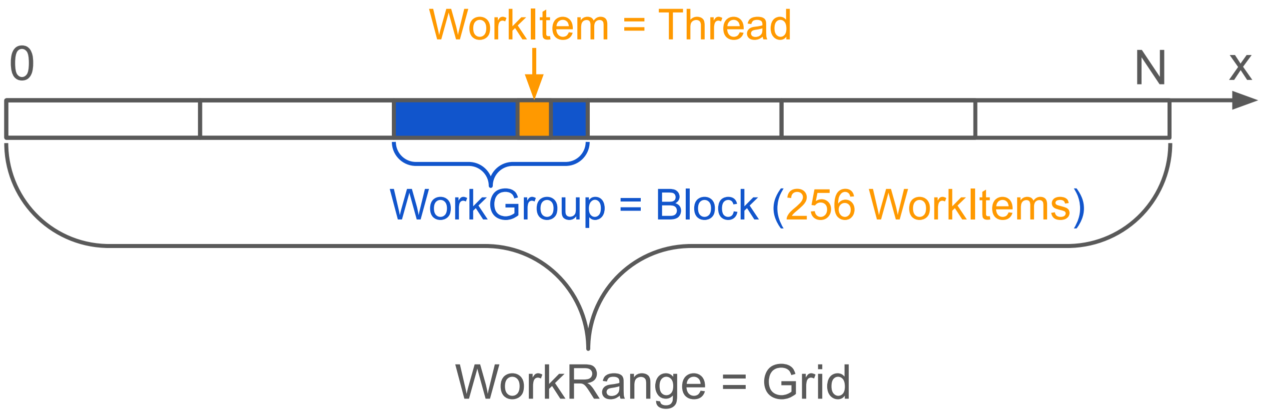


WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**

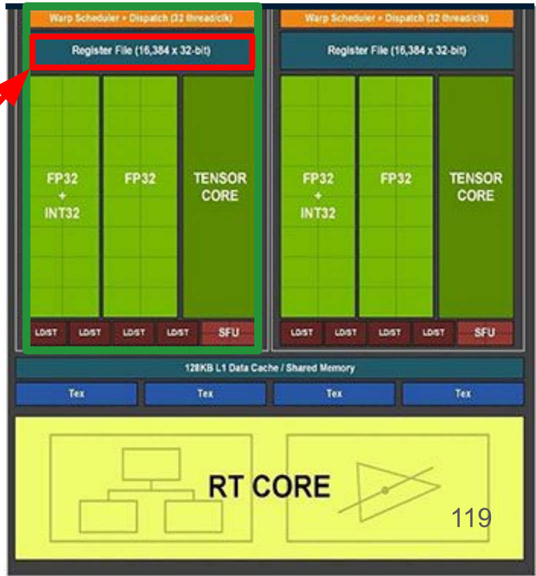
Могут ли потоки одного warp-а общаться еще эффективнее?

Модель вычислений массового параллелизма



Много SM

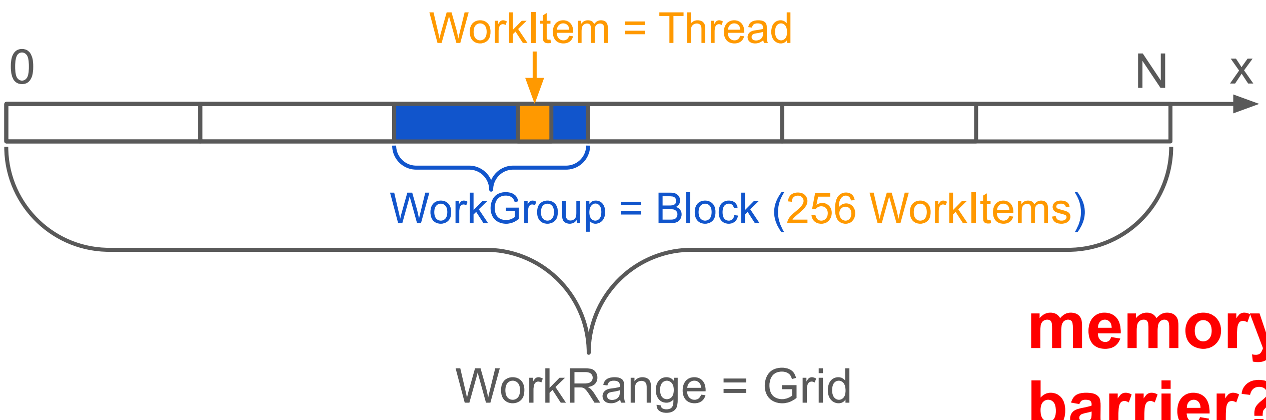
SM - Streaming Multiprocessor



WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

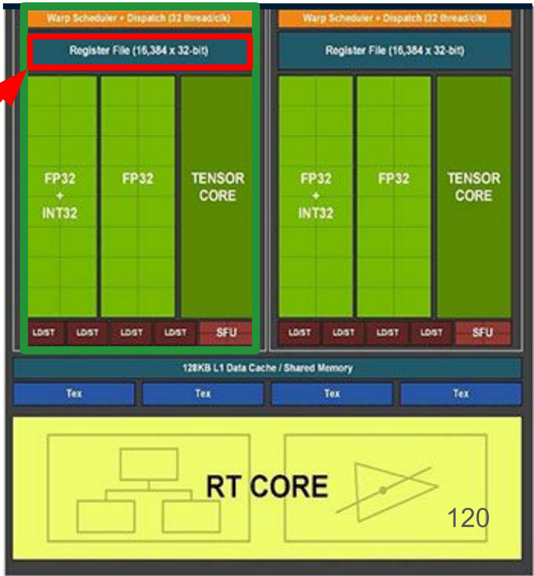
Модель вычислений массового параллелизма



Много SM

SM - Streaming Multiprocessor

memory barrier?



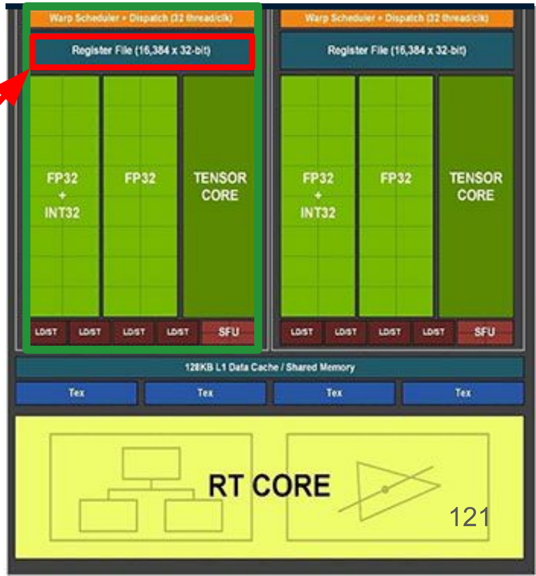
WorkGroup = Block (256 WorkItems) = 8 x Warps (32 threads)

- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

GLSL (Graphics Library Shading Language)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y

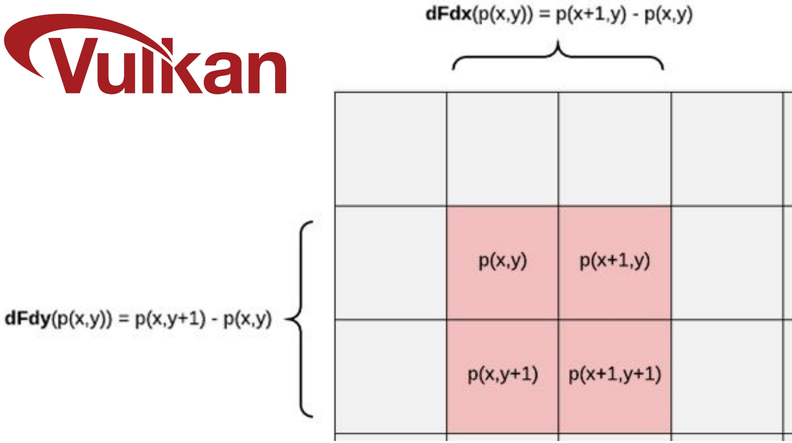
SM - Streaming Multiprocessor



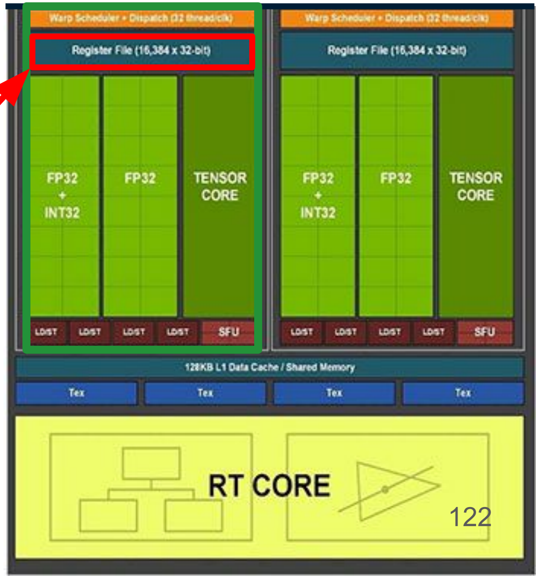
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

GLSL (Graphics Library Shading Language) OpenGL Vulkan

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y



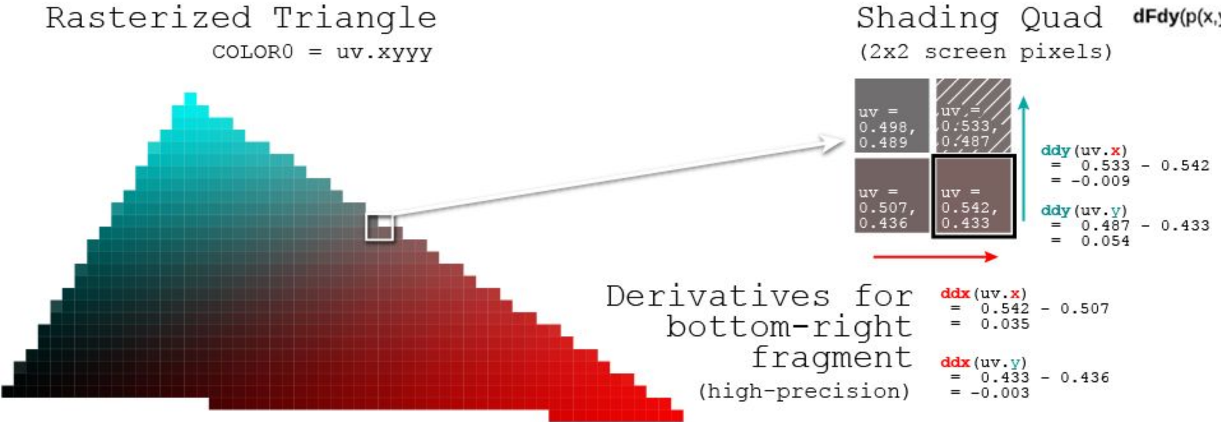
SM - Streaming Multiprocessor

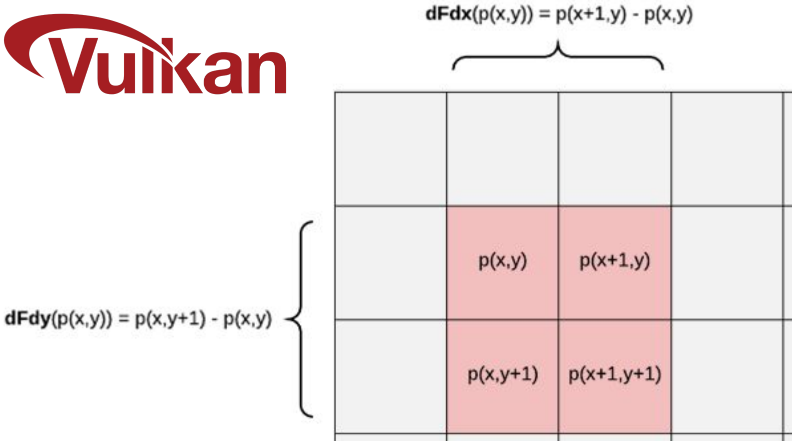


- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

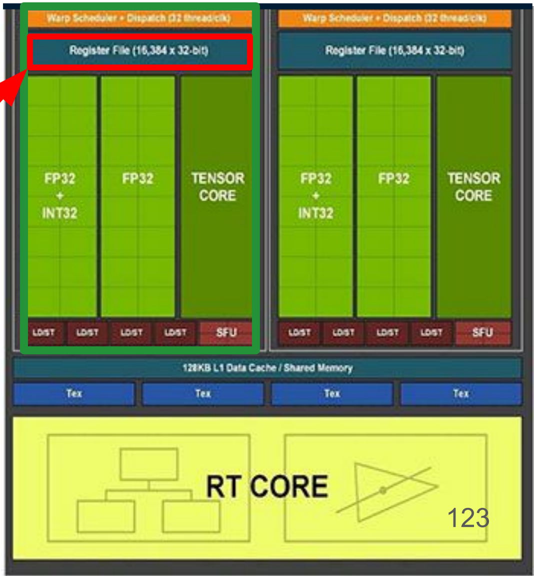
GLSL (Graphics Library Shading Language)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y



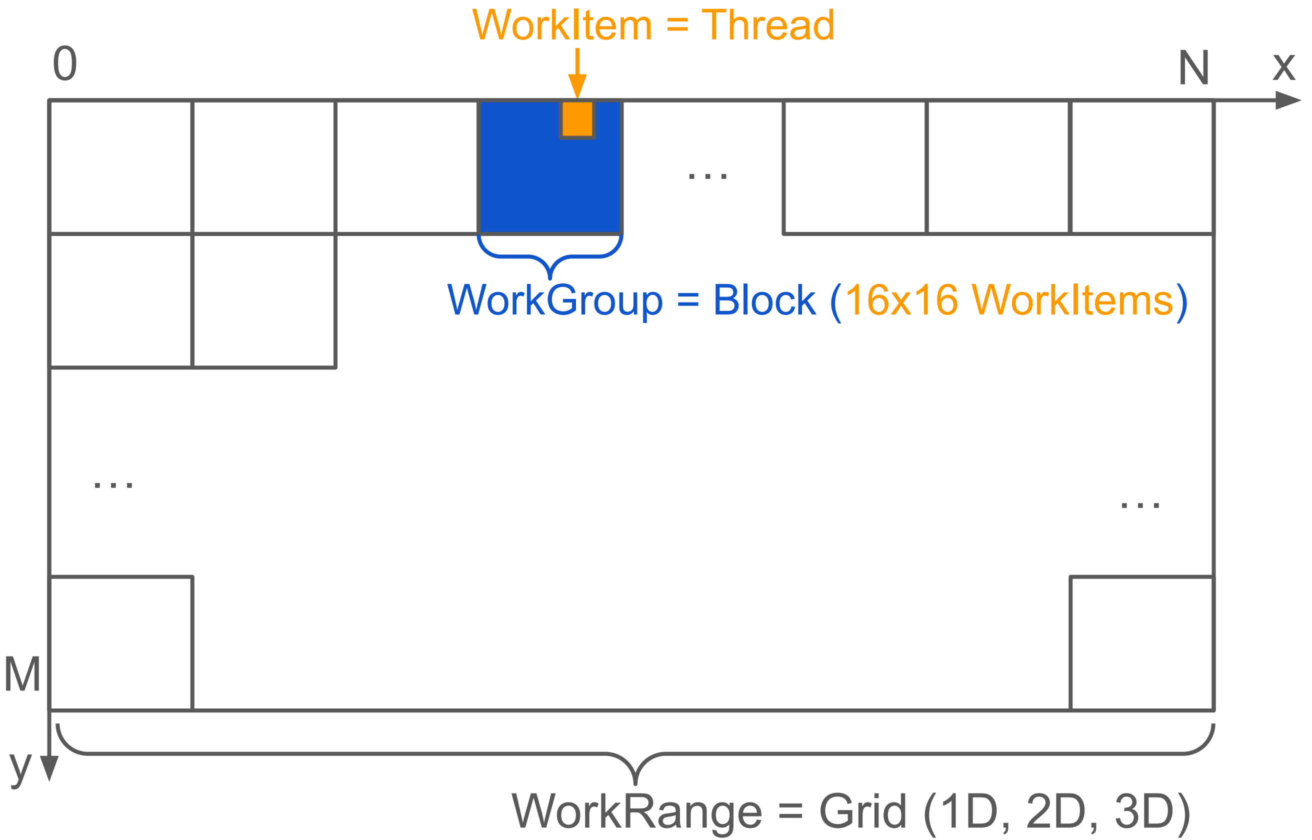


SM - Streaming Multiprocessor



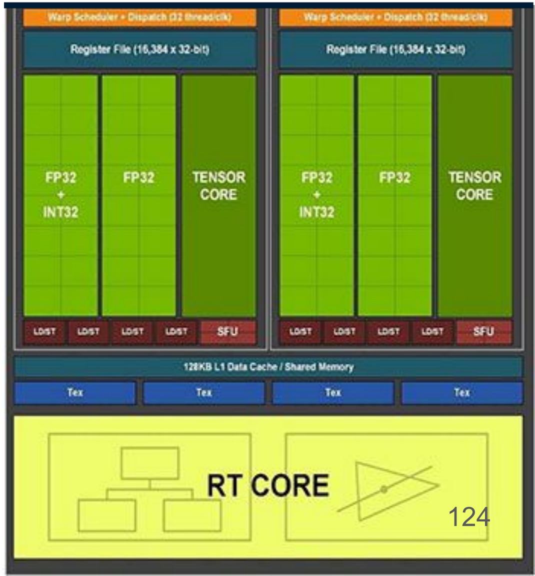
- WorkItems коммуницируют через VRAM
- WorkItems в рамках одной WorkGroup общаются эффективнее - через **Shared/Local Memory (L1)**
- WorkItems в рамках одного warp-а могут подглядывать друг другу в **регистры** (***shuffle instr.***)

Модель вычислений массового параллелизма

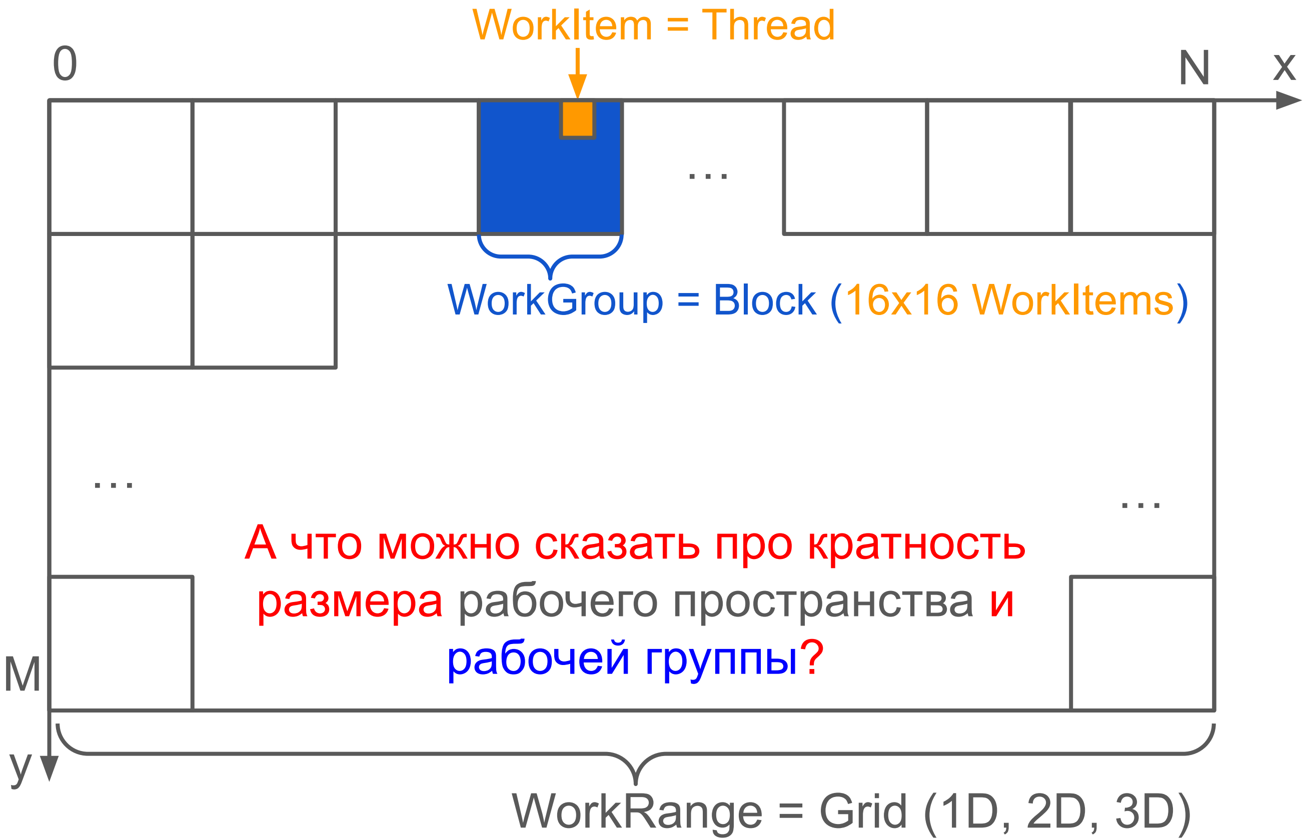


Много SM

SM - Streaming Multiprocessor

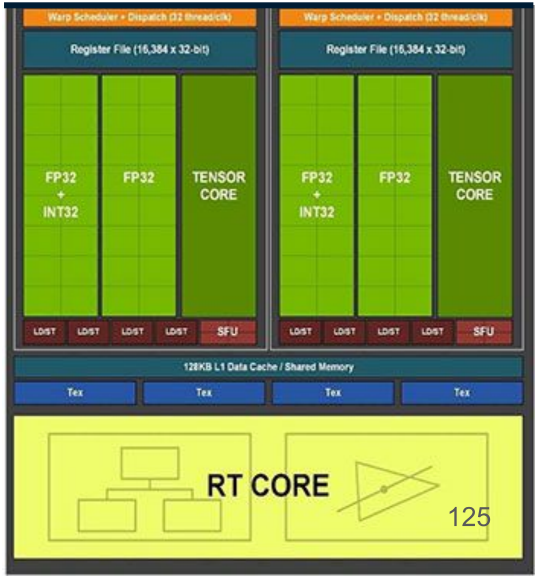


Модель вычислений массового параллелизма



Много SM

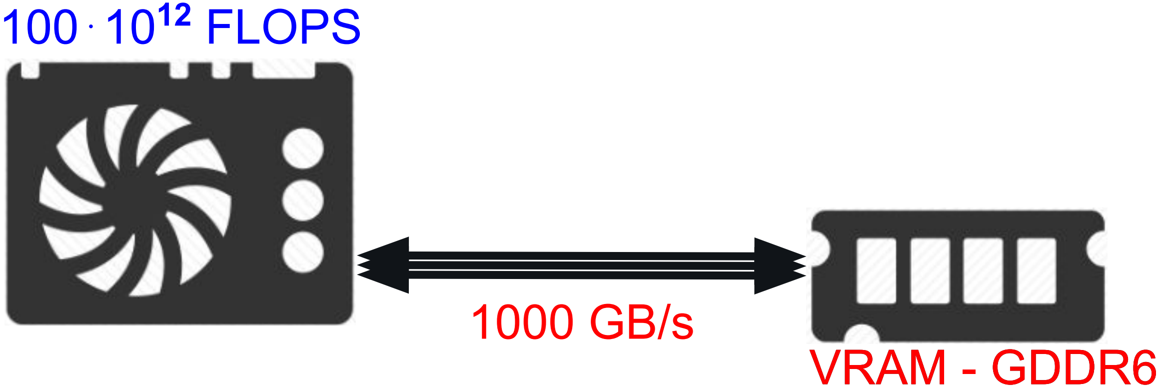
SM - Streaming Multiprocessor



Глава 5: Профилирование и оптимизация

МНОГО вычислений? (**ALU**)

МНОГО читать+писать? (**VRAM**)



Чего больше?
Как сравнить яблоки и апельсины?

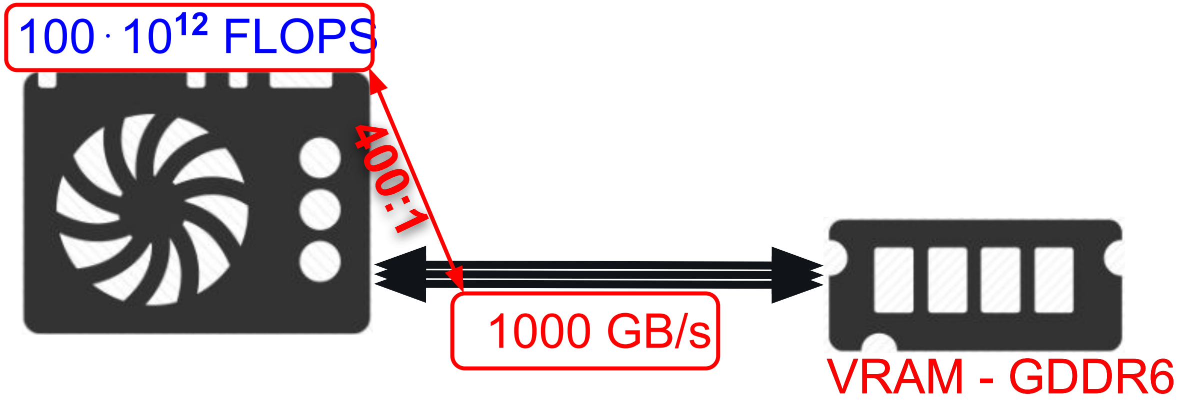
МНОГО вычислений? (**ALU**)

МНОГО читать+писать? (**VRAM**)

Чего больше?

МНОГО вычислений? (**ALU**)

МНОГО читать+писать? (**VRAM**)



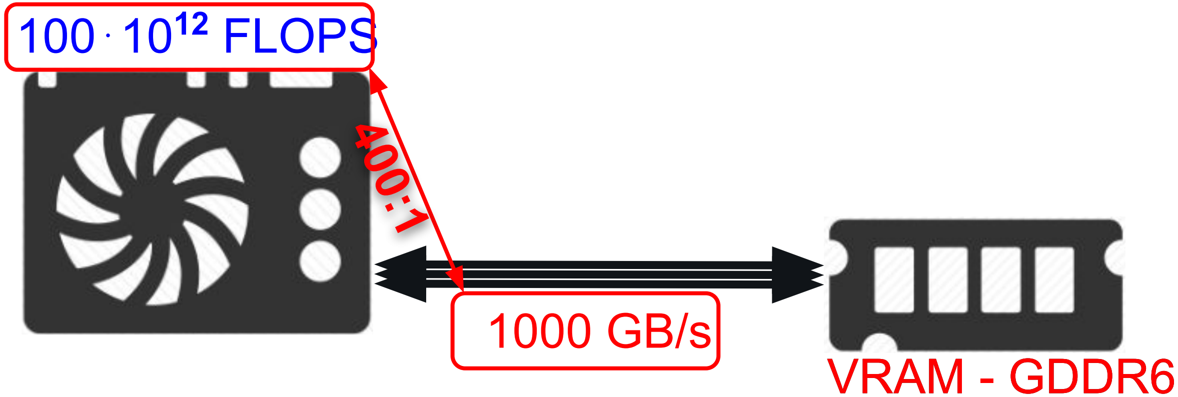
Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

МНОГО вычислений? (**ALU**)

Если **ALU:VRAM** < 400:1

МНОГО читать+писать? (**VRAM**)



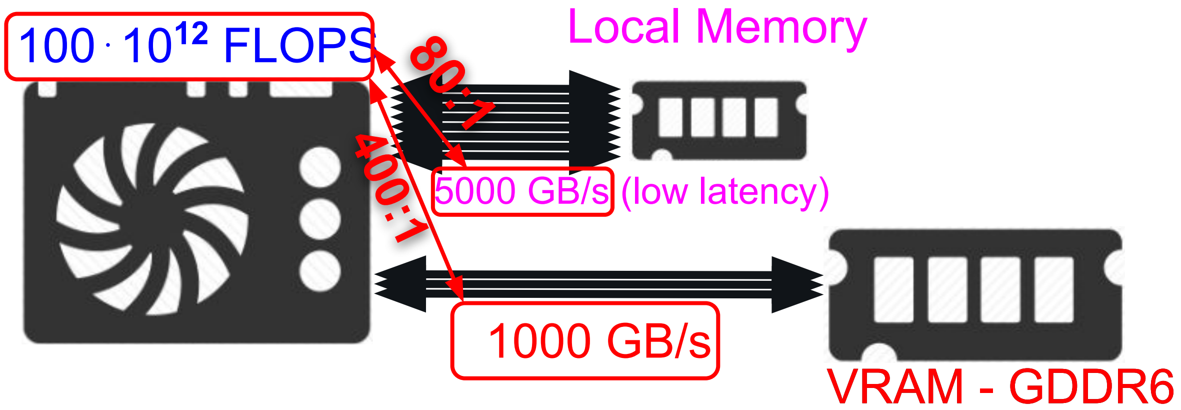
Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

МНОГО вычислений? (**ALU**)

Если **ALU:VRAM** < 400:1

МНОГО читать+писать? (**VRAM**)



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

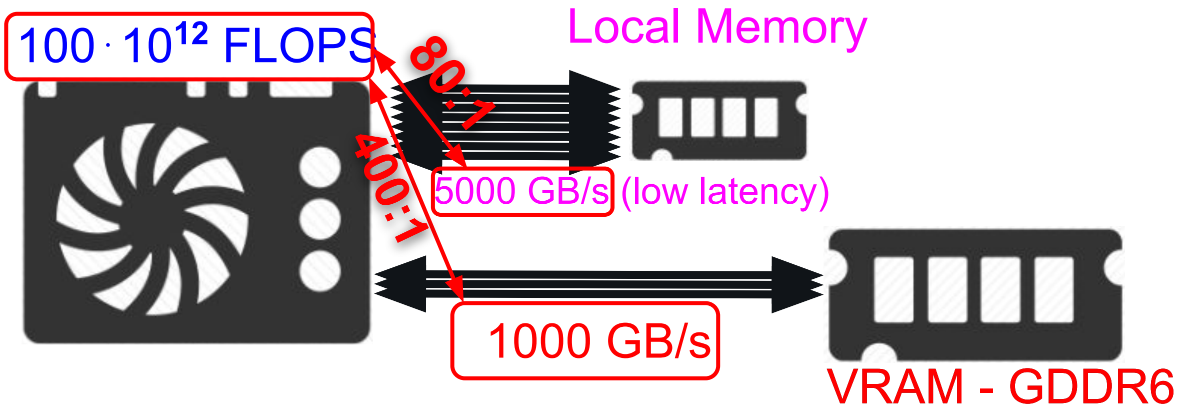
МНОГО вычислений? (ALU)

Если утилизируется >70% ALU FLOPs

compute bound

Если **ALU:VRAM** < 400:1

МНОГО читать+писать? (VRAM)



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

МНОГО вычислений? (**ALU**)

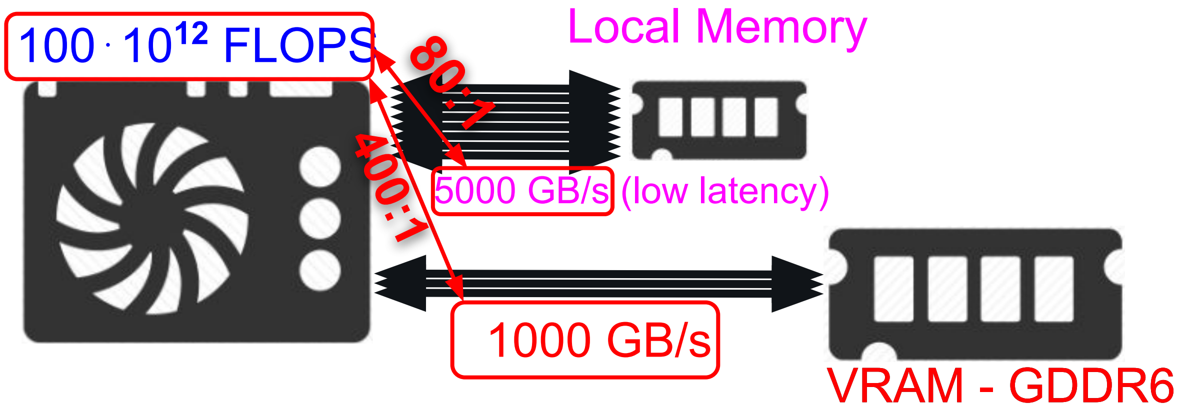
Если утилизируется
>70% ALU FLOPs

compute bound

Устранить **code divergence**

Если **ALU:VRAM** < 400:1

МНОГО читать+писать? (**VRAM**)



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

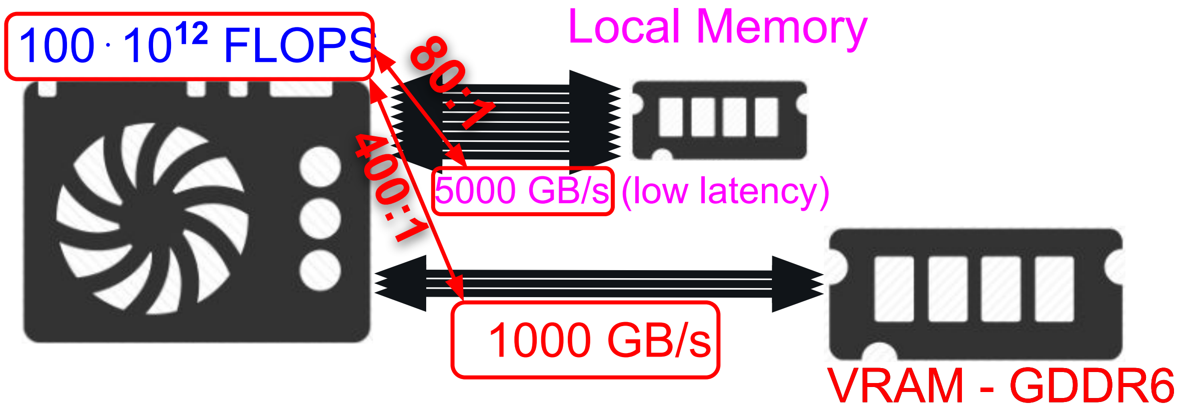
Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется >70% ALU FLOPs

compute bound

Устранить **code divergence**



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется >70% ALU FLOPs

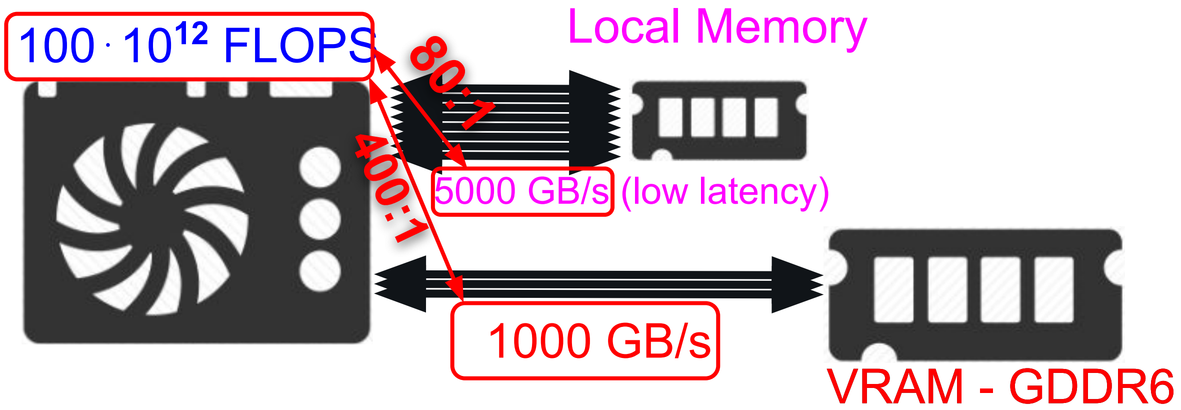
compute bound

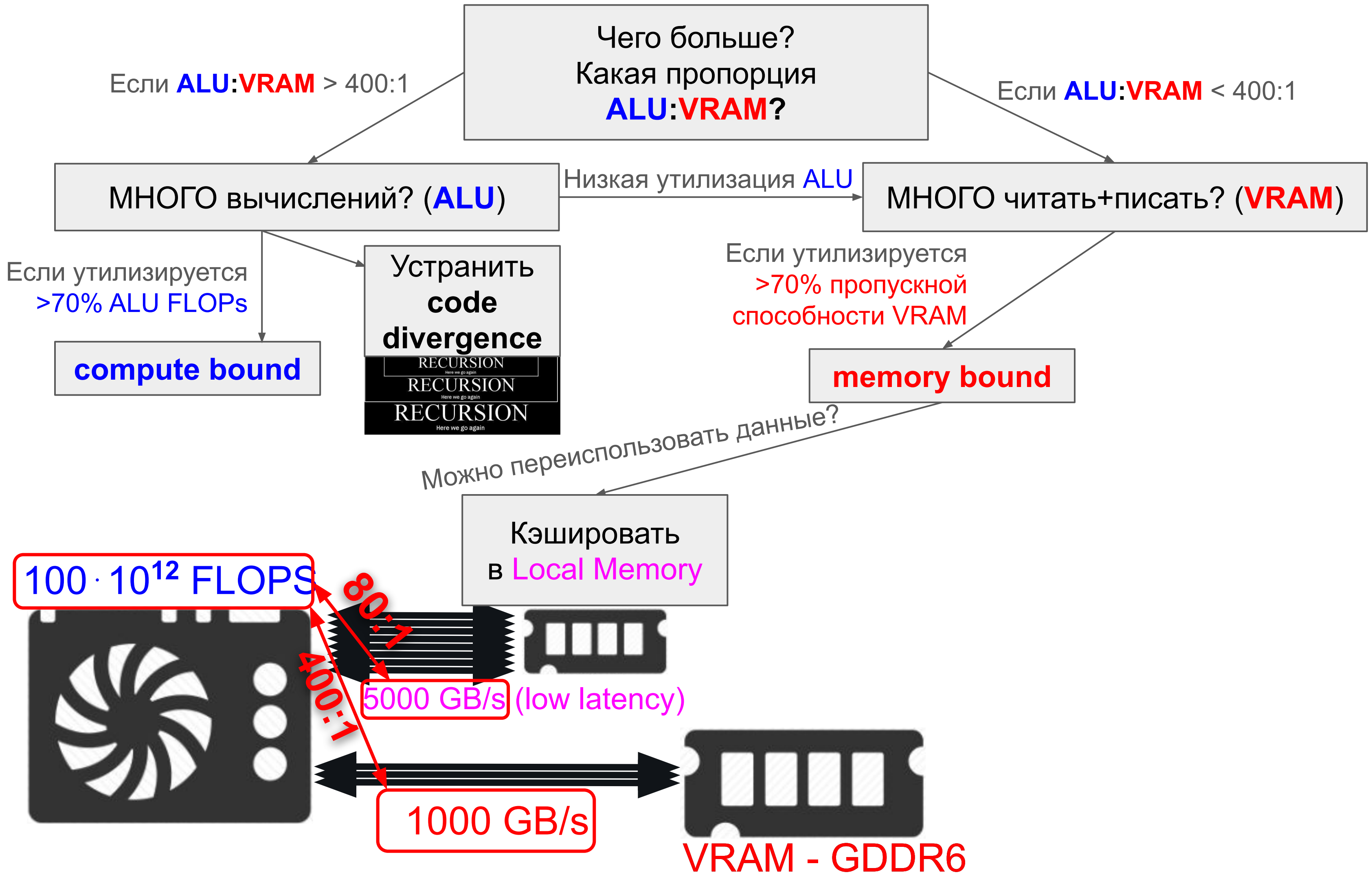
Устранить **code divergence**

Если утилизируется >70% пропускной способности VRAM

memory bound

Можно ли что-то сделать?





Чего больше?
Какая пропорция
ALU:VRAM?
Если ALU:VRAM > 400:1
Если ALU:VRAM < 400:1
МНОГО вычислений? (ALU)
Низкая утилизация ALU
МНОГО читать+писать? (VRAM)
Если утилизируется
>70% ALU FLOPs
compute bound
Устранить
code
divergence
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
Если утилизируется
>70% пропускной
способности VRAM
memory bound
Можно переиспользовать данные?
Кэшировать
в Local Memory
100 · 10^12 FLOPS
80:1
400:1
5000 GB/s
(low latency)
1000 GB/s
VRAM - GDDR6

Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

МНОГО вычислений? (ALU)

Если утилизируется >70% ALU FLOPs → **compute bound**

Устранить **code divergence**

Низкая утилизация ALU →

Если **ALU:VRAM** < 400:1

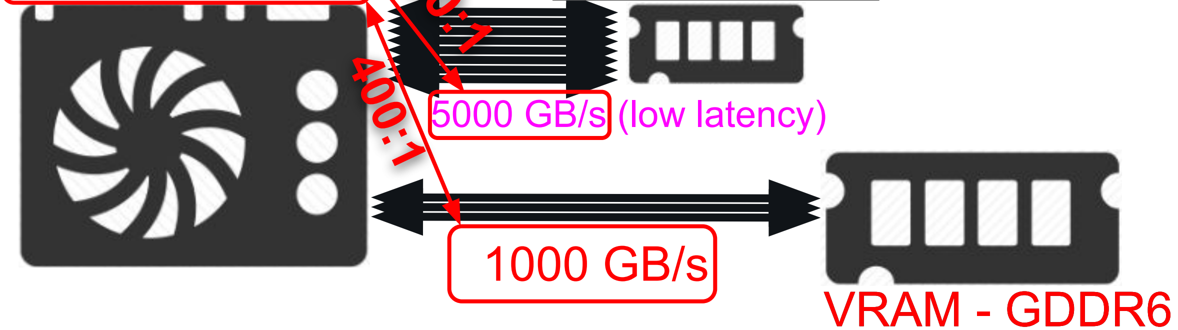
МНОГО читать+писать? (VRAM)

Если утилизируется >70% пропускной способности VRAM → **memory bound**

Низкая утилизация проп. способоности VRAM → **Почему?**

Можно переиспользовать данные?

Кэшировать в Local Memory



Чего больше?
Какая пропорция
ALU:**VRAM**?

Если **ALU**:**VRAM** > 400:1

Если **ALU**:**VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется >70% ALU FLOPs

compute bound

Устранить **code divergence**

Если утилизируется >70% пропускной способности VRAM

memory bound

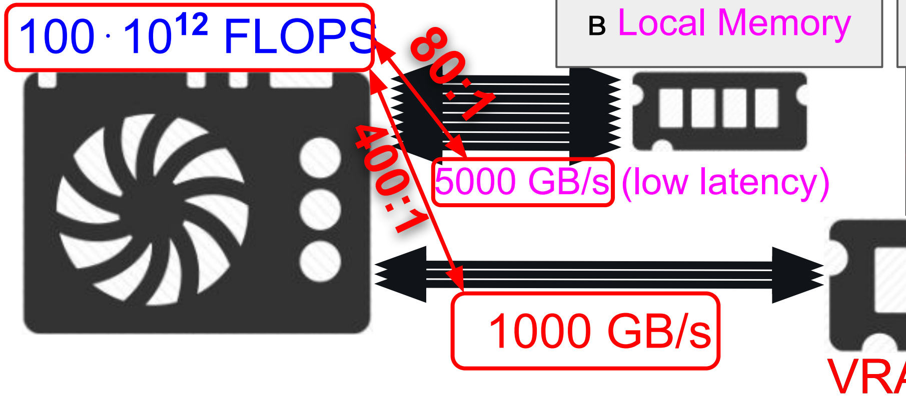
Низкая утилизация проп. способоности VRAM

Почему?

Можно переиспользовать данные?

Кэшировать в Local Memory

Сделать паттерн доступа **coalesced**



Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

МНОГО вычислений? (ALU)

Если утилизируется
>70% ALU FLOPs

compute bound

Устранить **code divergence**

Низкая утилизация ALU

Если **ALU:VRAM** < 400:1

МНОГО читать+писать? (VRAM)

Если утилизируется
>70% пропускной
способности VRAM

memory bound

Можно переиспользовать данные?

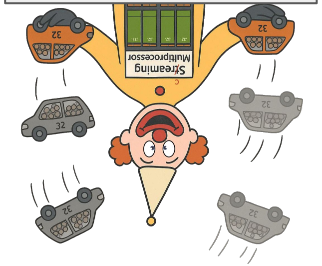
Кэшировать
в Local Memory

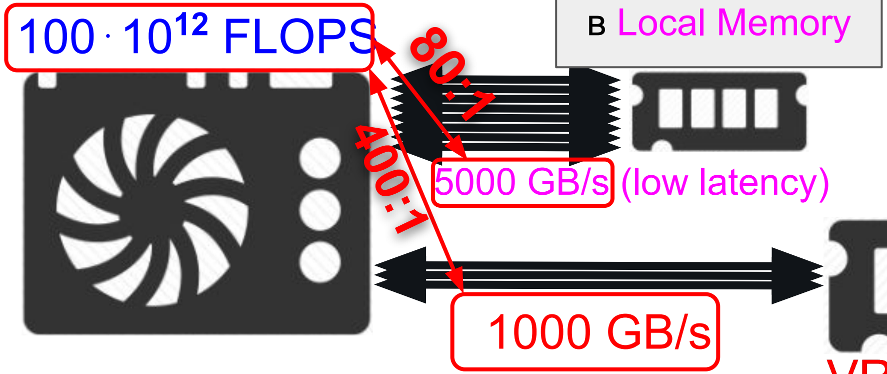
Низкая утилизация
проп. способоности VRAM

Почему?

Сделать паттерн
доступа **coalesced**

Увеличить
occupancy





Чего больше?
Какая пропорция
ALU:VRAM?

Если **ALU:VRAM** > 400:1

Если **ALU:VRAM** < 400:1

МНОГО вычислений? (**ALU**)

Низкая утилизация ALU

МНОГО читать+писать? (**VRAM**)

Если утилизируется >70% ALU FLOPs

compute bound

Устранить **code divergence**

Если утилизируется >70% пропускной способности VRAM

memory bound

Низкая утилизация проп. способоности VRAM

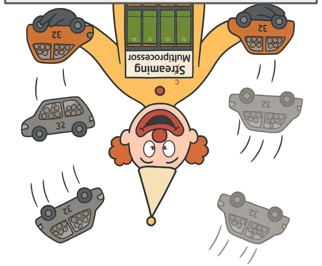
Почему?

Можно переиспользовать данные?

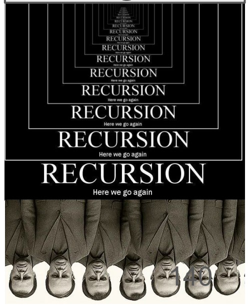
Кэшировать в Local Memory

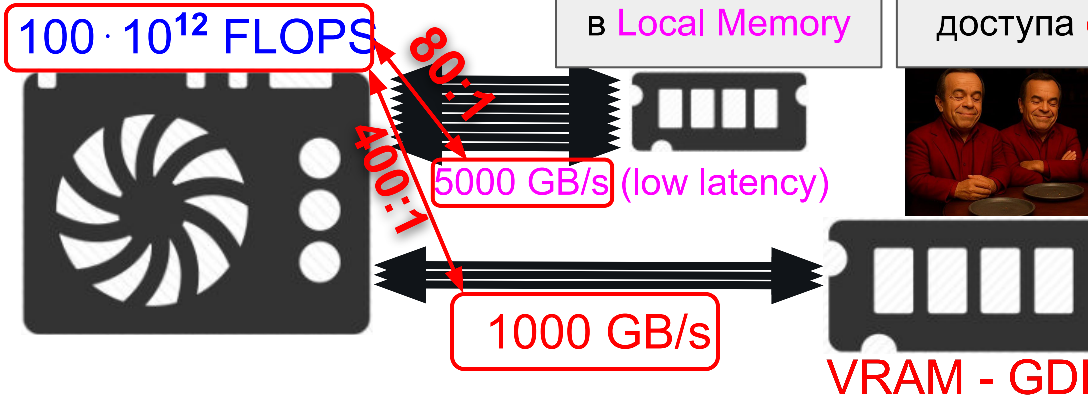
Сделать паттерн доступа **coalesced**

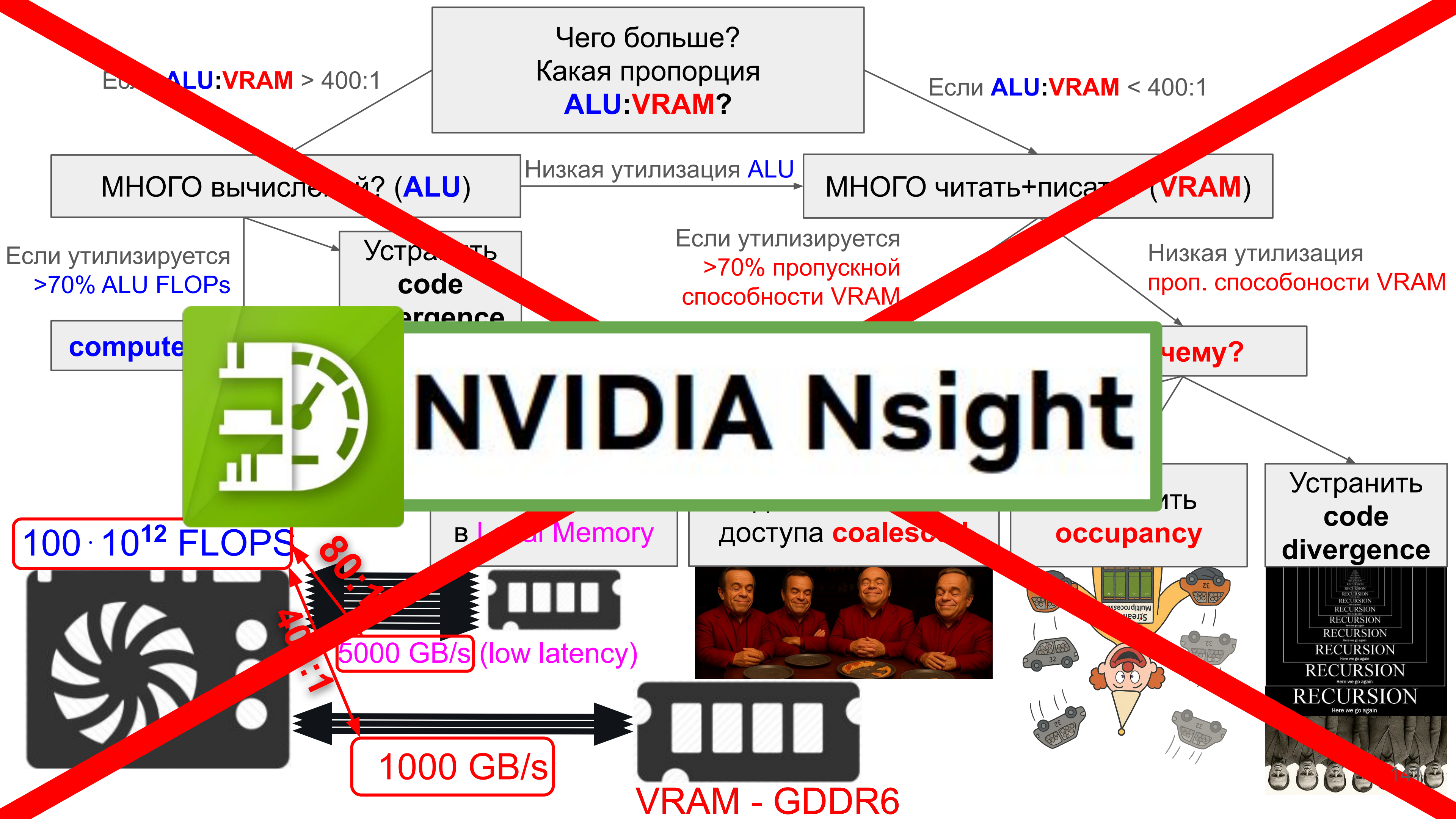
Увеличить **occupancy**



Устранить **code divergence**

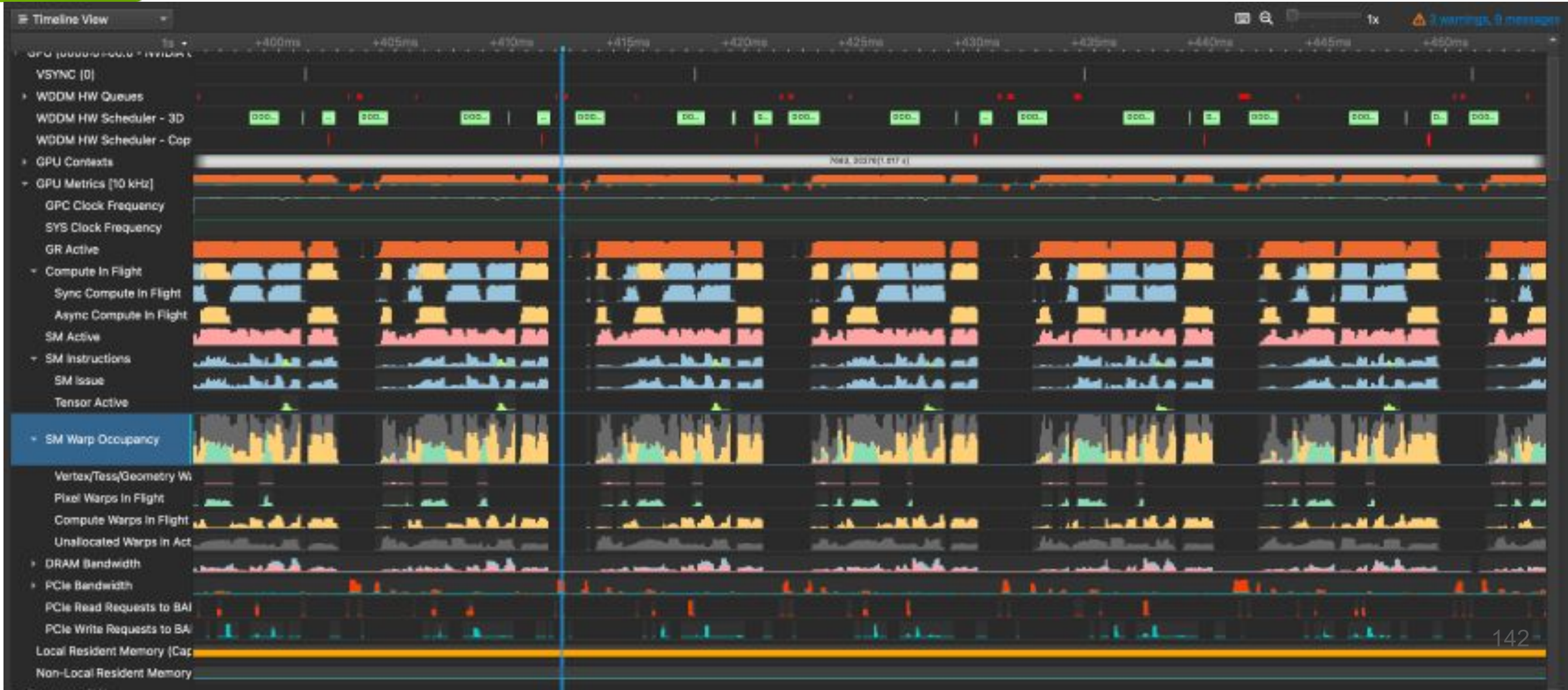


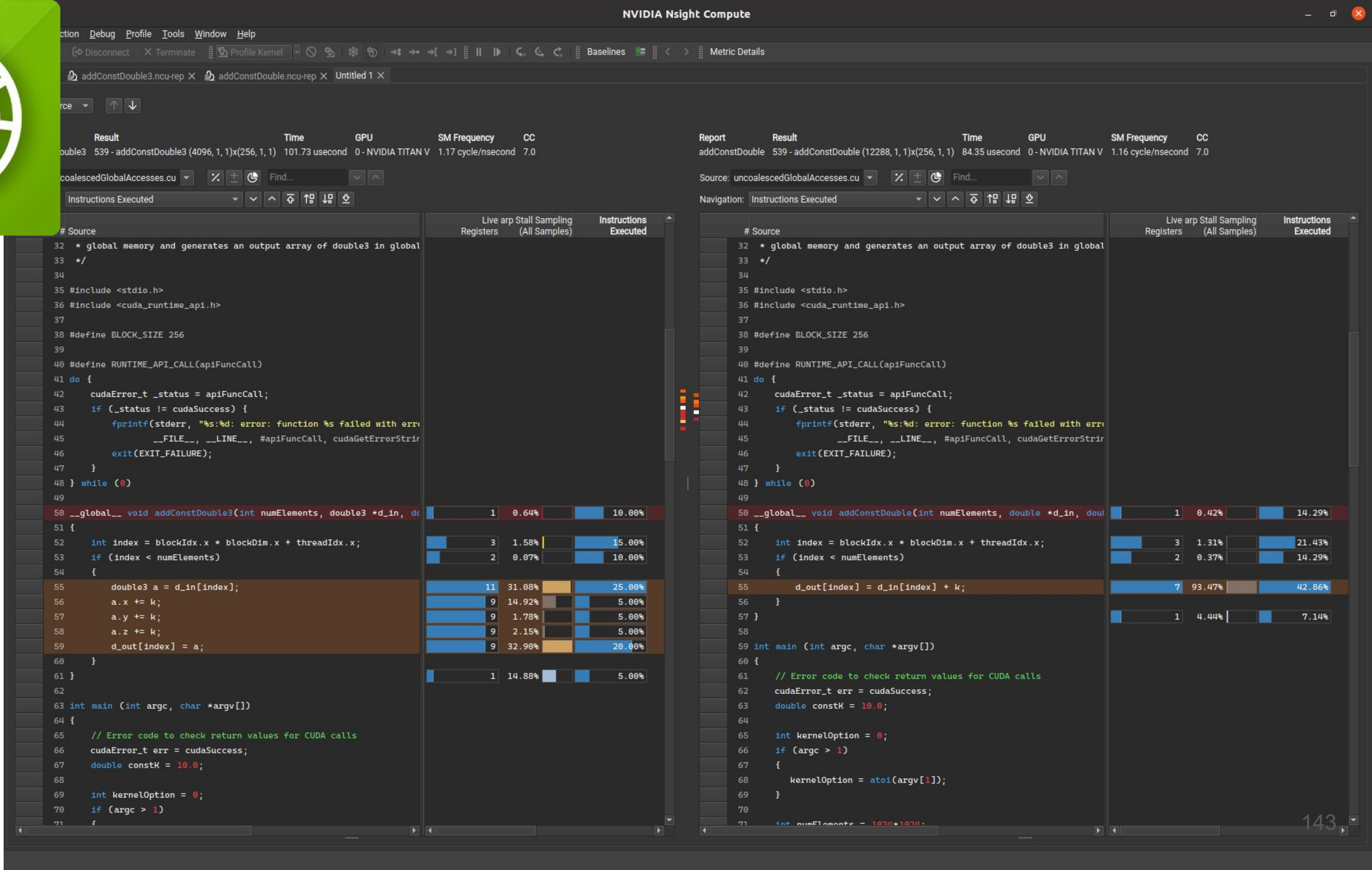


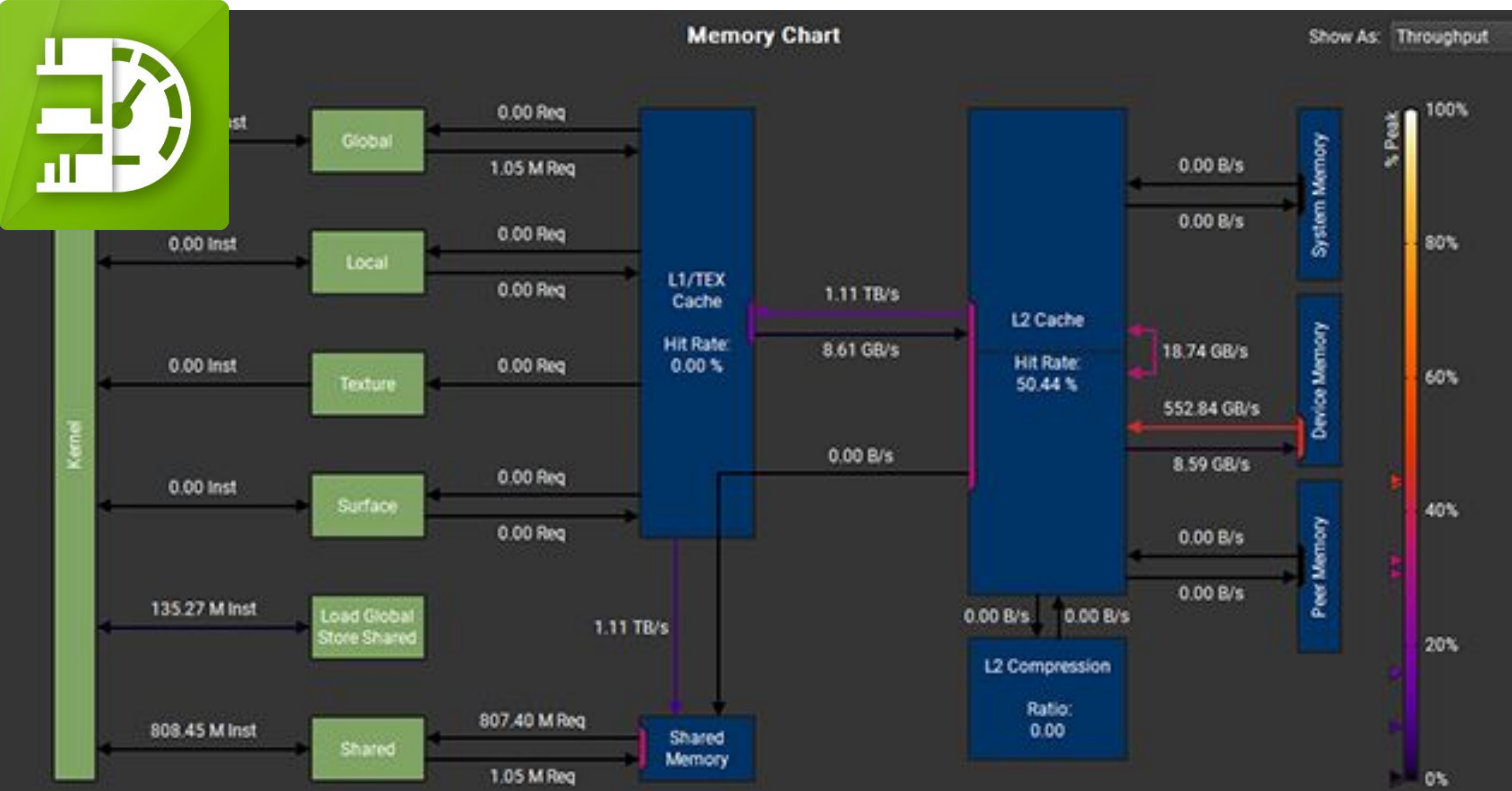


Чего больше?
Какая пропорция
ALU:VRAM?
ALU:VRAM > 400:1
Если ALU:VRAM < 400:1
МНОГО вычисле й? (ALU)
Низкая утилизация ALU
МНОГО читать+писат (VRAM)
Если утилизируется
>70% ALU FLOPs
Устра ть
code
ergence
Если утилизируется
>70% пропускной
способности VRAM
Низкая утилизация
проп. способоности VRAM
compute
чему?
NVIDIA Nsight
ить
Устранить
code
divergence
в Memory
доступа coalesc
occupancy
$100 \cdot 10^{12}$ FLOPS
80:
40:1
5000 GB/s (low latency)
1000 GB/s
VRAM - GDDR6
RECURSION
Here we go again

NVIDIA Nsight




NVIDIA Nsight Compute
Debug
Profile
Tools
Window
Help
Disconnect
Terminate
Profile Kernel
Baselines
Metric Details
addConstDouble3.ncu-rep
addConstDouble.ncu-rep
Untitled 1
Result
Time
GPU
SM Frequency
CC
539 - addConstDouble3 (4096, 1, 1)x(256, 1, 1)
101.73 usecond
0 - NVIDIA TITAN V
1.17 cycle/nsecond
7.0
Report
addConstDouble
539 - addConstDouble (12288, 1, 1)x(256, 1, 1)
84.35 usecond
1.16 cycle/nsecond
Source:
uncoalescedGlobalAccesses.cu
Navigation:
Instructions Executed
Find...
Source
Live Registers
Warp Stall Sampling (All Samples)
Instructions Executed
* global memory and generates an output array of double3 in global
*/
#include <stdio.h>
#include <cuda_runtime_api.h>
#define BLOCK_SIZE 256
#define RUNTIME_API_CALL(apiFuncCall)
do {
cudaError_t _status = apiFuncCall;
if (_status != cudaSuccess) {
fprintf(stderr, "%s:%d: error: function %s failed with err
__FILE__, __LINE__, #apiFuncCall, cudaGetErrorStrin
exit(EXIT_FAILURE);
}
} while (0)
__global__ void addConstDouble3(int numElements, double3 *d_in, d
int index = blockIdx.x * blockDim.x + threadIdx.x;
if (index < numElements)
double3 a = d_in[index];
a.x += k;
a.y += k;
a.z += k;
d_out[index] = a;
int main (int argc, char *argv[])
// Error code to check return values for CUDA calls
cudaError_t err = cudaSuccess;
double constK = 10.0;
int kernelOption = 0;
if (argc > 1)
1 0.64% 10.00%
3 1.58% 15.00%
2 0.07% 10.00%
11 31.08% 25.00%
9 14.92% 5.00%
9 1.78% 5.00%
9 2.15% 5.00%
9 32.90% 20.00%
1 14.88% 5.00%
__global__ void addConstDouble(int numElements, double *d_in, doul
d_out[index] = d_in[index] + k;
kernelOption = atoi(argv[1]);
1 0.42% 14.29%
3 1.31% 21.43%
2 0.37% 14.29%
7 93.47% 42.86%
1 4.44% 7.14%



Memory Chart
Show As: Throughput
Kernel
Global
Local
Texture
Surface
Load Global Store Shared
Shared
0.00 Inst
0.00 Inst
0.00 Inst
135.27 M Inst
808.45 M Inst
0.00 Req
1.05 M Req
0.00 Req
0.00 Req
0.00 Req
0.00 Req
0.00 Req
807.40 M Req
1.05 M Req
L1/TEX Cache
Hit Rate: 0.00 %
1.11 TB/s
8.61 GB/s
0.00 B/s
1.11 TB/s
Shared Memory
L2 Cache
Hit Rate: 50.44 %
18.74 GB/s
552.84 GB/s
8.59 GB/s
0.00 B/s
0.00 B/s
0.00 B/s
0.00 B/s
0.00 B/s
0.00 B/s
L2 Compression
Ratio: 0.00
System Memory
Device Memory
Peer Memory
% Peak
100%
80%
60%
40%
20%
0%

: 2 - 566 - mergeRanksAndIndicesKer | Clear Baselines | Apply Rules | Occupancy Calculator | Source Comparison | Copy as Image

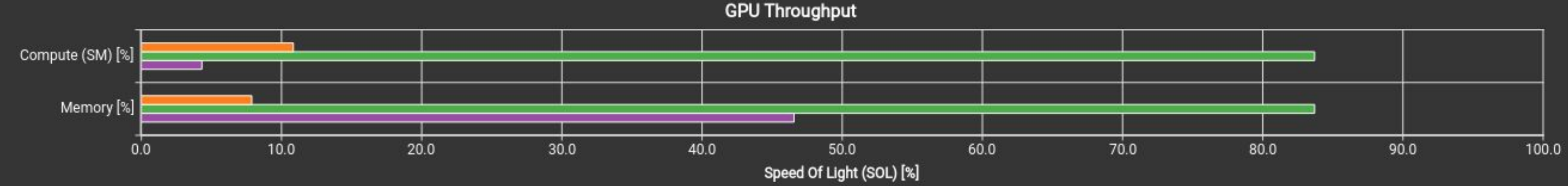
	Time	Cycles	GPU	SM Frequency	Process	Attributes
geRanksAndIndicesKernel (64, 1, 1)x(256, 1, 1)	4.19 us	4,326	0 - NVIDIA RTX A4500	1.03 cycle/ns	[762349] CuMergeSort	
geSortSharedKernel (4096, 1, 1)x(512, 1, 1)	555.10 us	5,79,537	0 - NVIDIA RTX A4500	1.04 cycle/ns	[762349] CuMergeSort	
erateSampleRanksKernel (64, 1, 1)x(256, 1, 1)	26.72 us	27,984	0 - NVIDIA RTX A4500	1.05 cycle/ns	[762349] CuMergeSort	

GPU Speed Of Light Throughput

GPU Throughput Chart

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

Compute (SM) Throughput [%]	10.84 (-75.36%, z=-0.61)	Duration [us]	4.19 (-98.56%, z=-0.75)
Memory Throughput [%]	7.88 (-87.90%, z=-1.23)	Elapsed Cycles [cycle]	4,326 (-98.58%, z=-0.75)
L1/TEX Cache Throughput [%]	10.84 (-79.01%, z=-0.82)	SM Active Cycles [cycle]	2,192.52 (-99.26%, z=-0.74)
L2 Cache Throughput [%]	7.88 (-51.10%, z=-0.75)	SM Frequency [cycle/ns]	1.03 (-1.32%, z=-1.38)
DRAM Throughput [%]	5.53 (-83.26%, z=-1.08)	DRAM Frequency [cycle/ns]	7.55 (-0.53%, z=-1.39)



PM Sampling

Timeline view of PM metrics sampled periodically over the workload duration. Data is collected across multiple passes. Use this section to understand how workload behavior changes over its runtime.

Baselines

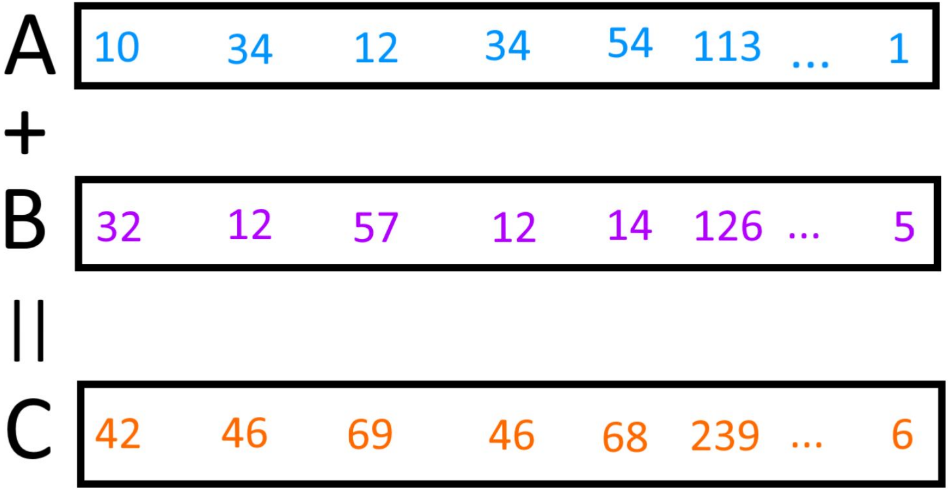
Difference Bars Green/Red

	Name	Launch	Report	Time	Cycles	Registers	GPU	SM Frequency	CC	Process
✓	mergeSort	560 - mergeSortSharedKernel (4096, ...	/home/prabhanshuc/Documents/me...	555.10 us	5,79,537	17	0 - NVIDIA RTX ...	1.04 cycle/ns	8.6	[762349] CuMergeSort
✓	generateSample	563 - generateSampleRanksKernel (6...	/home/prabhanshuc/Documents/me...	26.72 us	27,984	18	0 - NVIDIA RTX ...	1.05 cycle/ns	8.6	[762349] CuMergeSort
✓	mergeRanks	566 - mergeRanksAndIndicesKernel (...	/home/prabhanshuc/Documents/me...	4.19 us	4,326	16	0 - NVIDIA RTX ...	1.03 cycle/ns	8.6	[762349] CuMergeSort

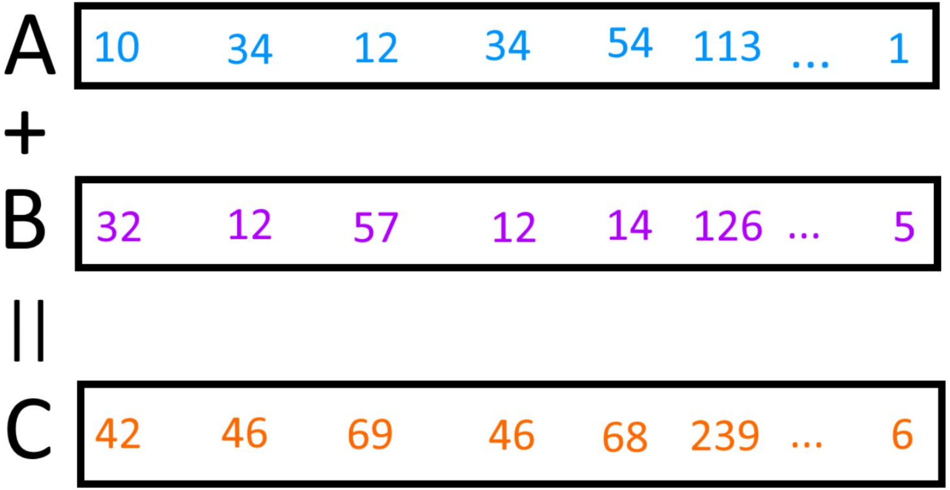
Глава 6: Примеры

A+B, максимум по массиву, merge-sort

Пример: A + B

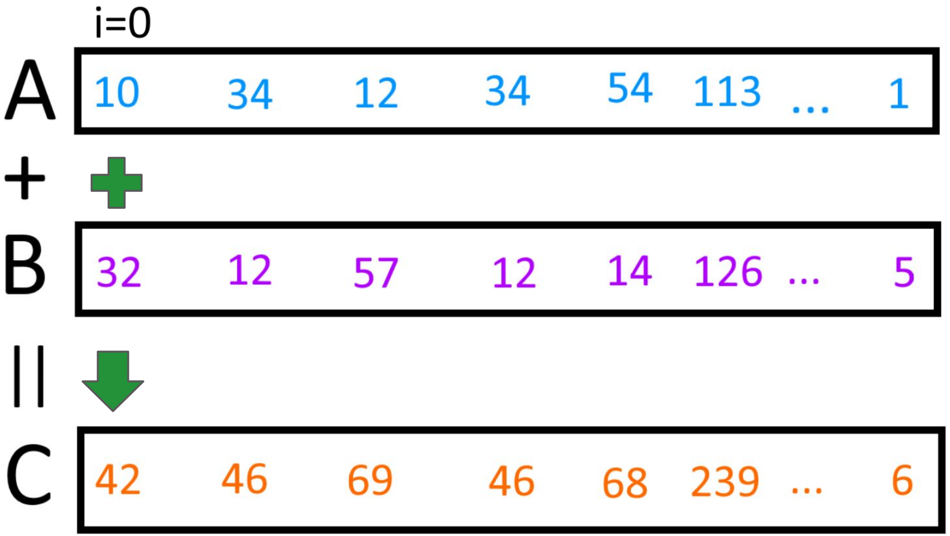


Пример: A + B



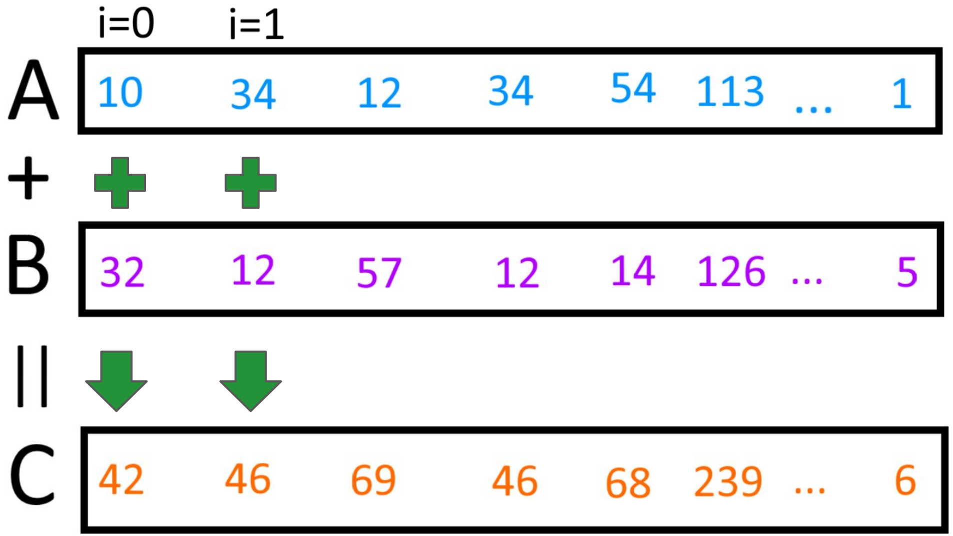

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Пример: A + B



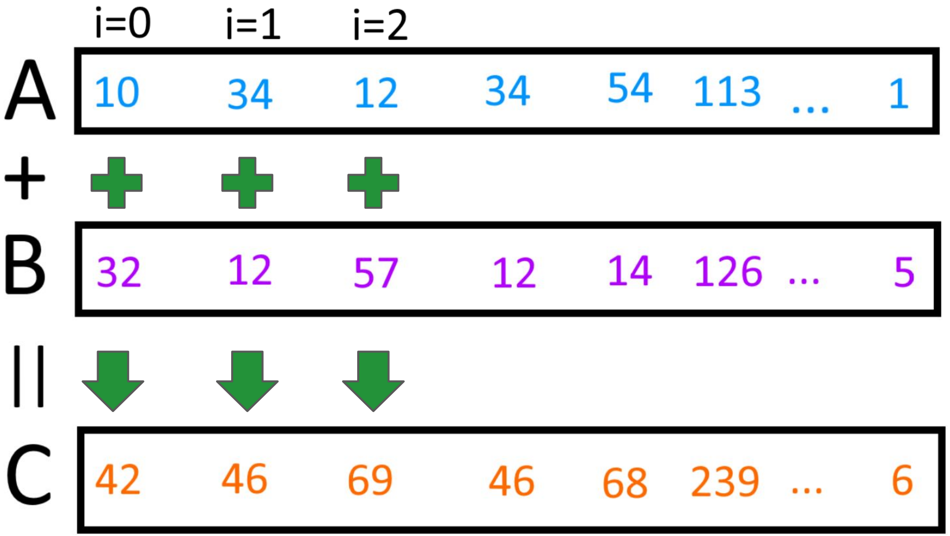

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Пример: A + B



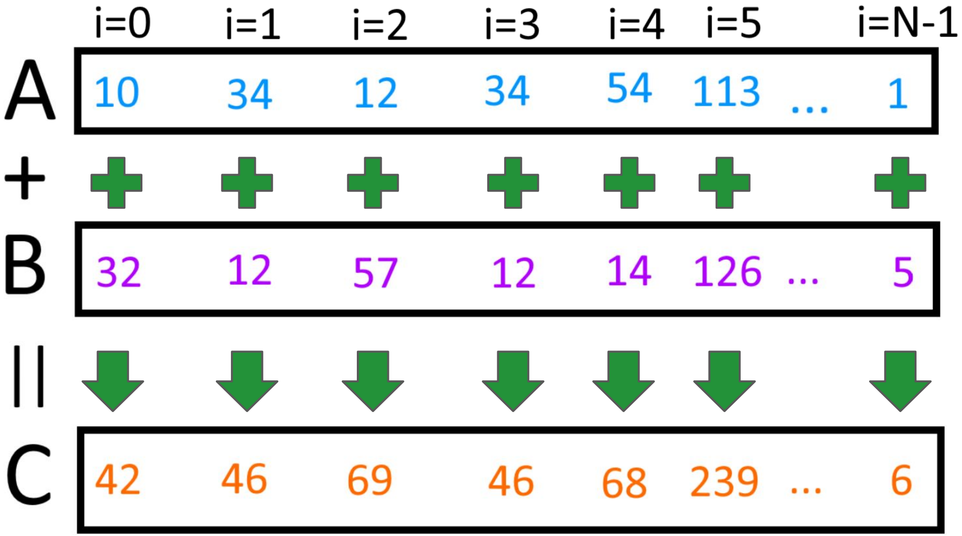

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Пример: A + B




```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

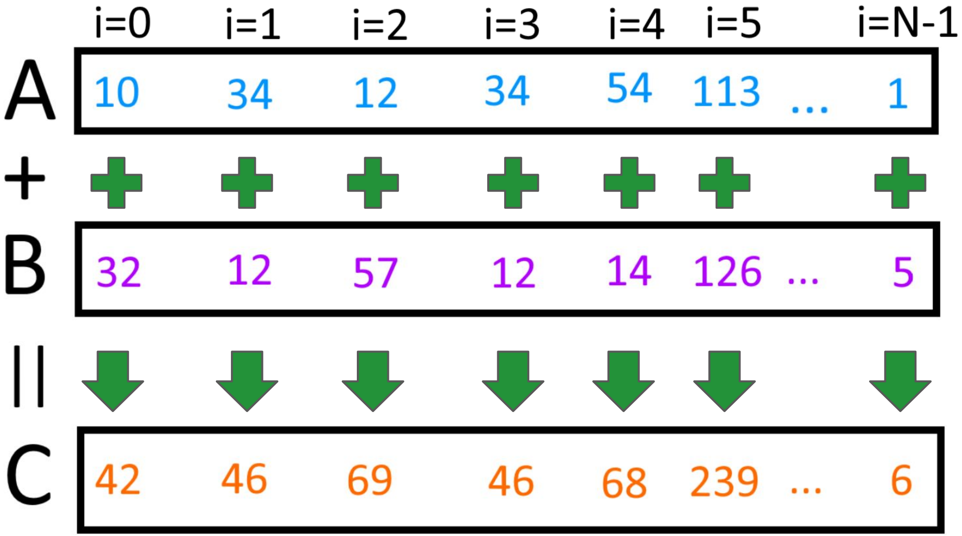
Пример: A + B




```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

Какая асимптотика?

Пример: A + B

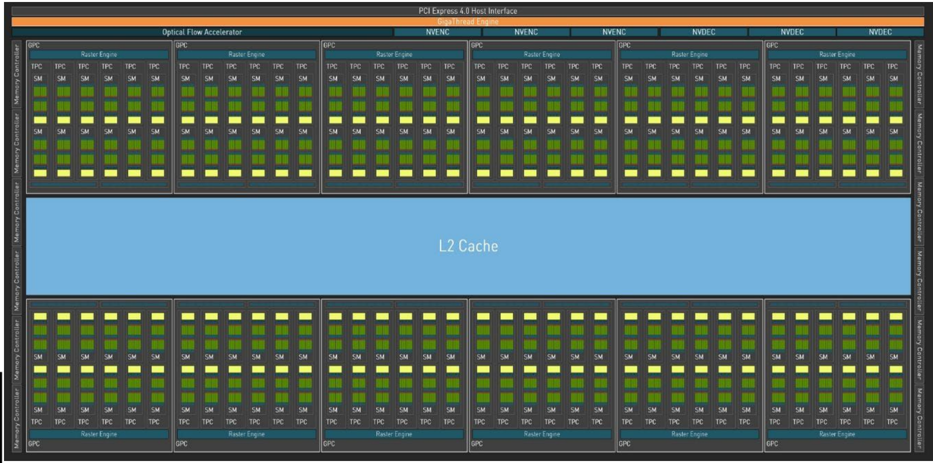



```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

Пример: A + B

A	10	34	12	34	54	113	...	1
+								
B	32	12	57	12	14	126	...	5
=								
C	42	46	69	46	68	239	...	6



NVIDIA RTX 4090 - 16 тысяч ядер!

OpenCL™ NVIDIA CUDA Vulkan

Пример: A + B

A	10	34	12	34	54	113	...	1
+								
B	32	12	57	12	14	126	...	5
||								
C	42	46	69	46	68	239	...	6

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

```
void solveGPU(int[] a, int[] b, int c[], int n) {
    const int i = get_global_id(0);
    c[i] = b[i] + a[i];
}
```

Супер многопоточно!

NVIDIA RTX 4090 - 16 тысяч ядер!

Пример: A + B

A	10	34	12	34	54	113	…	1
+								
B	32	12	57	12	14	126	…	5
||								
C	42	46	69	46	68	239	…	6


```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

```
void solveGPU(int[] a, int[] b, int c[], int n) {
    const int i = get_global_id(0);
    c[i] = b[i] + a[i];
}
```

Какая асимптотика?

NVIDIA RTX 4090 - 16 тысяч ядер!

OpenCL™ NVIDIA CUDA Vulkan

Пример: A + B

	i=0	i=1	i=2	i=3	i=4	i=5		i=N-1
A	10	34	12	34	54	113	...	1
+	+	+	+	+	+	+		+
B	32	12	57	12	14	126	...	5
||	↓	↓	↓	↓	↓	↓		↓
C	42	46	69	46	68	239	...	6

```
void solveCPU(int[] a, int[] b, int c[], int n) {
    for (int i = 0; i < n; ++i) {
        int sum = a[i] + b[i];
        c[i] = sum;
    }
}
```

O(N)

```
void solveGPU(int[] a, int[] b, int c[], int n) {
    const int i = get_global_id(0);
    c[i] = b[i] + a[i];
}
```

O(N / 16384)

NVIDIA RTX 4090 - 16 тысяч ядер!

OpenCL™ NVIDIA CUDA Vulkan

Пример: A + B (N=100.000.000)

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

Пример: A + B (N=100.000.000)

CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```

Пример: A + B (N=100.000.000)

CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```

CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```

Пример: A + B (N=100.000.000)

CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```

CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```

Почему разница такая незначительная?

Как проверить гипотезу? Как спровоцировать значительную разницу?

Пример: $\boxed{100*cos}$(A + B) (N=100.000.000)

CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = 100.0*cos( Left: as[i] + bs[i]);
}
```

```
median time: 1.178 sec (+-0.0235009)
median RAM bandwidth: 0.948716 GB/s
```

CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = 100*cos( Left: as[i] + bs[i]);
}
```

```
median time: 0.109 sec (+-0.0121988)
median RAM bandwidth: 10.2531 GB/s
```

x10.9

Пример: A + B (N=100.000.000)

CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```

CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```

GPU - NVIDIA RTX 4090

```
__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

```
a + b median time: 0.002 sec (+-0.000632456)
a + b median VRAM bandwidth: 558.794 GB/s
```

Пример: A + B (N=100.000.000)

CPU - Intel 13700K

```
for (size_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.068 sec (+-0.00481664)
a + b median RAM bandwidth: 16.4351 GB/s
```

CPU - Intel 13700K

```
#pragma omp parallel for
for (ptrdiff_t i = 0; i < n; ++i) {
    cs[i] = as[i] + bs[i];
}
```

```
a + b median time: 0.047 sec (+-0.00153623)
a + b median RAM bandwidth: 23.7784 GB/s
```

GPU - NVIDIA RTX 4090

```
__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

x23.5

```
a + b median time: 0.002 sec (+-0.000632456)
a + b median VRAM bandwidth: 558.794 GB/s
```

Пример: A + B (N=100.000.000)

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5




__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id(dimindx: 0);
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

Пример: A + B (N=100.000.000)

OpenCL™

```
#ifdef __CLION_IDE__
#include <libgpu/opencl/cl/clion_defines.cl>
#endif

#line 5



__attribute__((reqd_work_group_size(256, 1, 1)))

__kernel void aplusb(__global const float* a,
                     __global const float* b,
                     __global       float* c,
                     unsigned int n)
{
    const unsigned int index = get_global_id( dimindx: 0)
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

NVIDIA CUDA

```
blockIdx.x * blockDim.x + threadIdx.x;
```

```
__global__ void aplusb(const float* a,
                       const float* b,
                       float* c,
                       unsigned int n)
{
    const unsigned int index = blockId
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

Пример: A + B (N=100.000.000)

NVIDIA CUDA

```
__global__ void aplusb(const float* a,
                       const float* b,
                       float* c,
                       unsigned int n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

Пример: A + B (N=100.000.000)

```
void cuda_aplusb(const gpu::WorkSize &workSize,
                 const float* a, const float* b, float* c, unsigned int n,
                 cudaStream_t stream)
{
    const unsigned int blockSize = 256;
    const unsigned int gridSize = (n + blockSize - 1) / blockSize;
    aplusb<<<gridSize, blockSize, 0, stream>>>(a, b, c, n);
    CUDA_CHECK_KERNEL(stream);
}
```

```
__global__ void aplusb(const float* a,
                       const float* b,
                       float* c,
                       unsigned int n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index >= n)
        return;

    c[index] = a[index] + b[index];
}
```

NVIDIA
CUDA

Пример: A + B (N=100.000.000)

GLSL (Graphics Library Shading Language)

```
#version 450

#include <libgpu/vulkan/vk/common.vk>

layout (std430, binding = 0) readonly   buffer AsIn     {   uint as[];  };
layout (std430, binding = 1) readonly   buffer BsIn     {   uint bs[];  };
layout (std430, binding = 2) writeonly  buffer CsOut    {   uint cs[];  };

layout (push_constant) uniform PushConstants {
    uint n;
} params;

layout (local_size_x = 256) in;

void main()
{
    const uint index = gl_GlobalInvocationID.x;
    if (index >= params.n)
        return;

    cs[index] = as[index] + bs[index];
}
```

Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090

Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090

Как это реализовать в модели массового параллелизма?

Какой размер рабочего пространства?

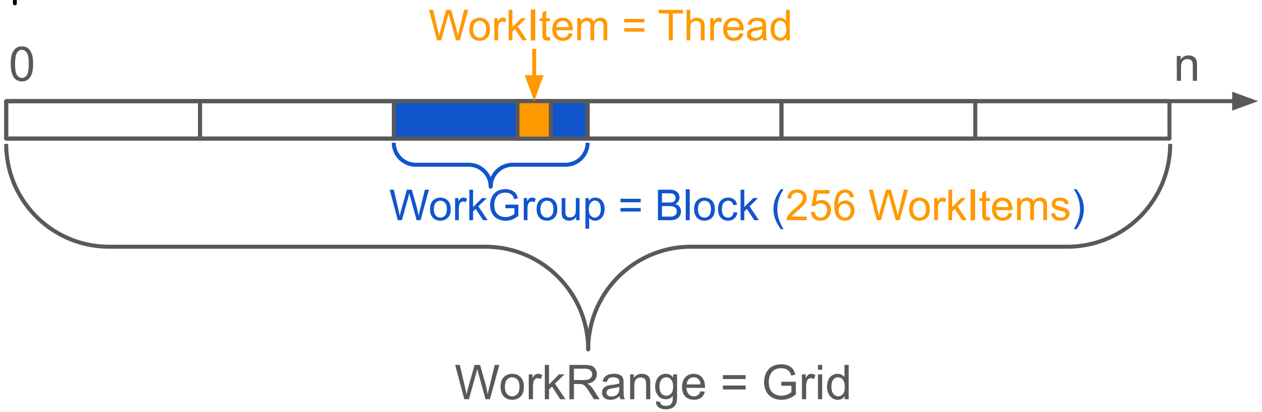
Что делает каждый поток (work item)?

Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090

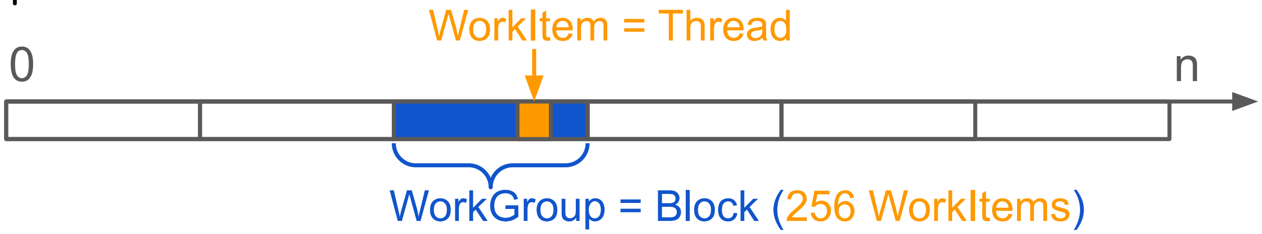


Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090



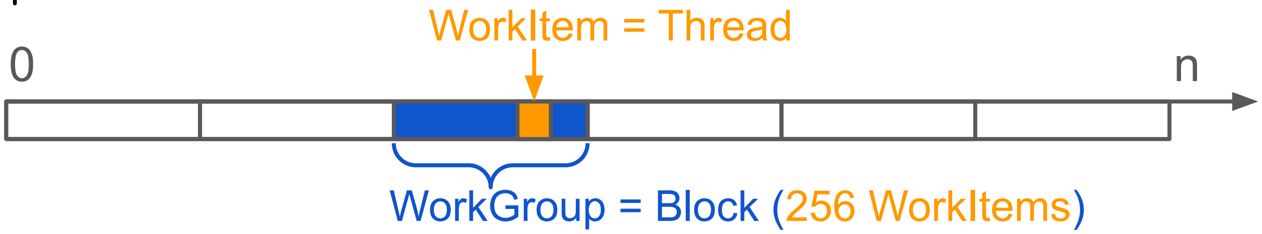
WorkGroup **обречен** считывать как минимум 32 элемента подряд. **Почему?**

Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090



WorkGroup **обречен** считывать как минимум 32 элемента подряд ради ***coalesced*** access pattern.

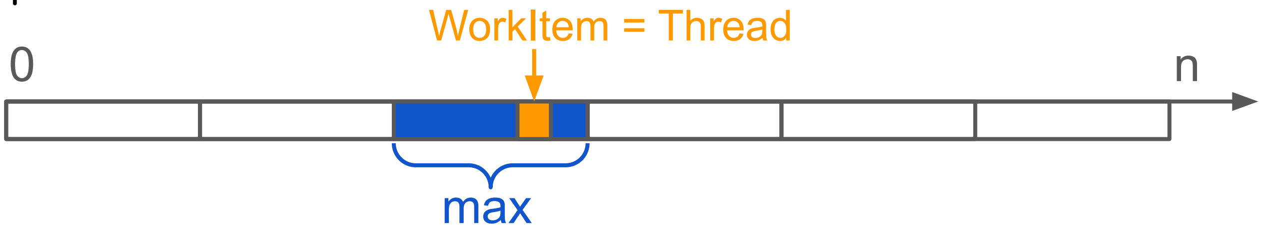
Что будем делать в WorkGroup?

Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090

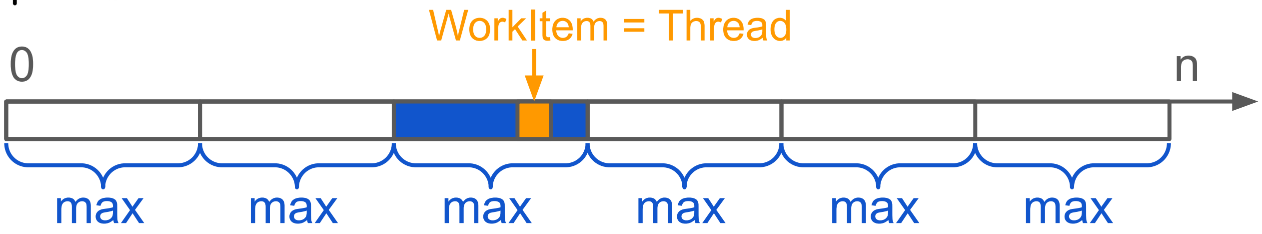


Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090



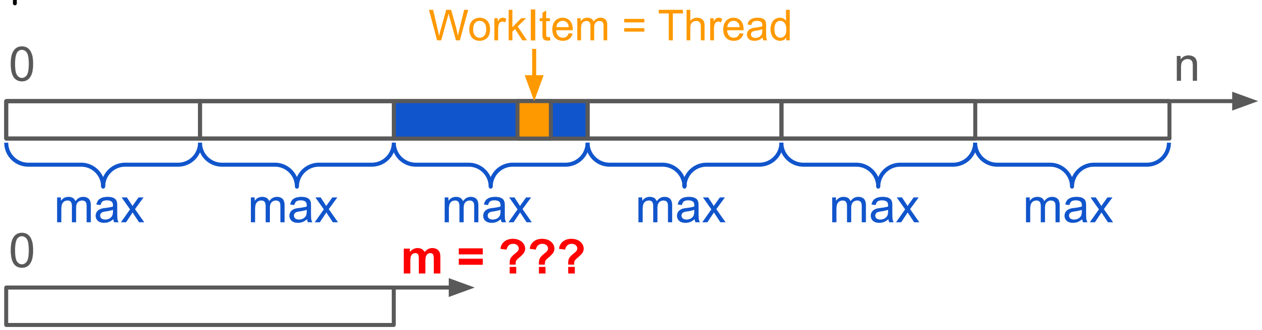
редукция (reduce, reduction)

Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090

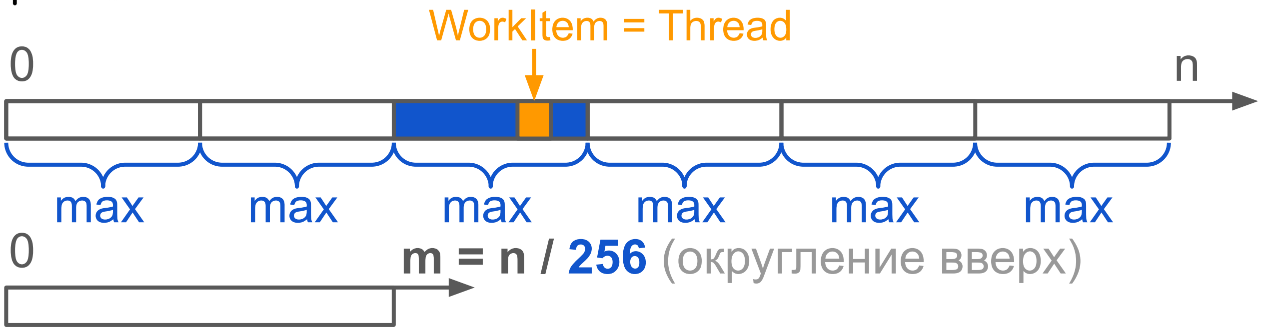


Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090

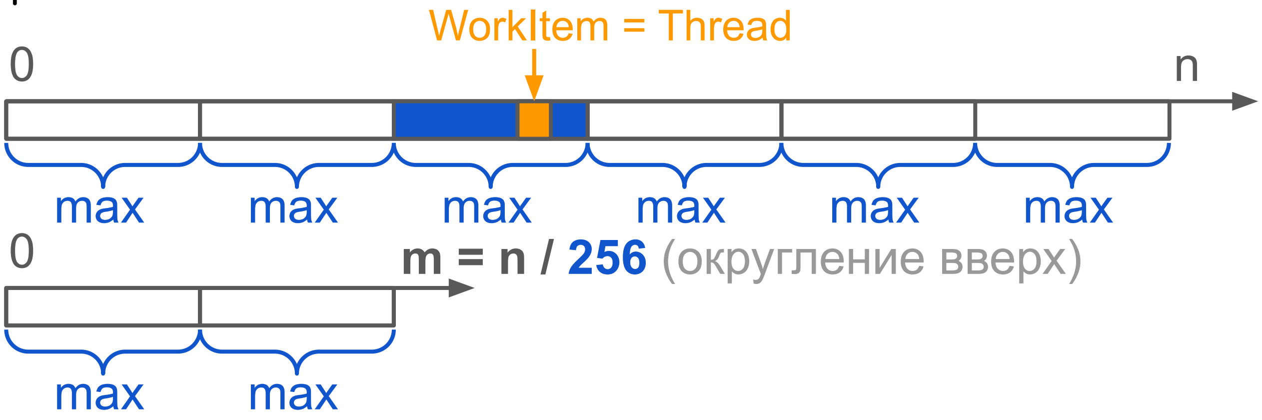


Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090

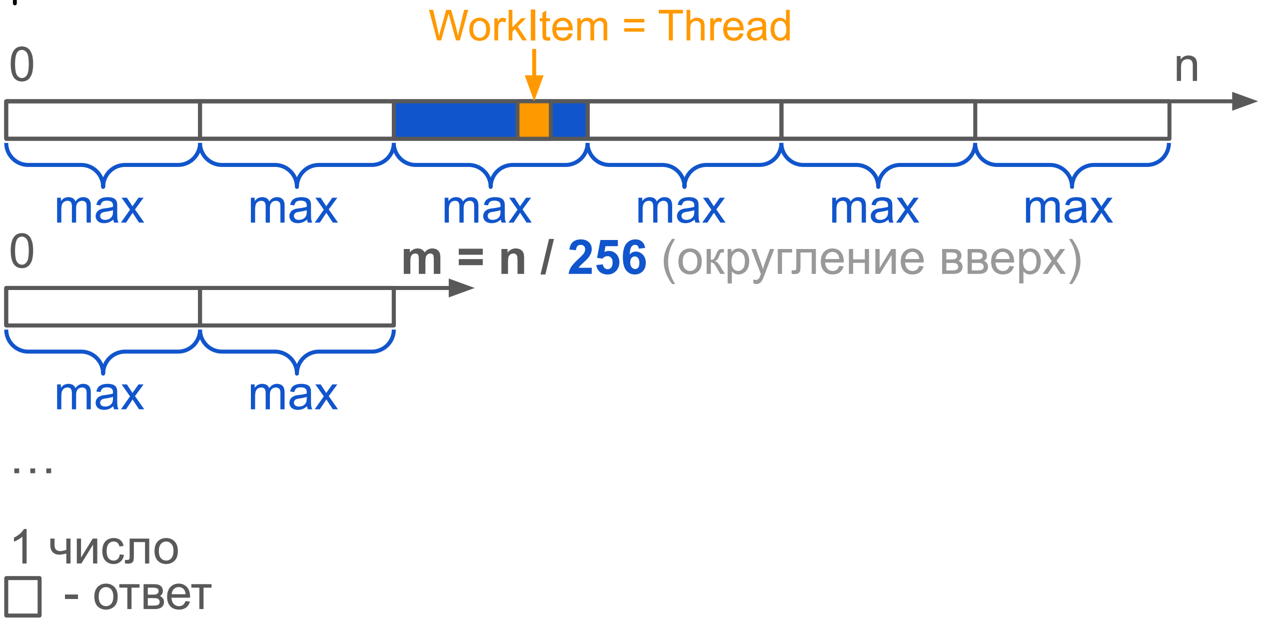


Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

GPU NVIDIA RTX 4090



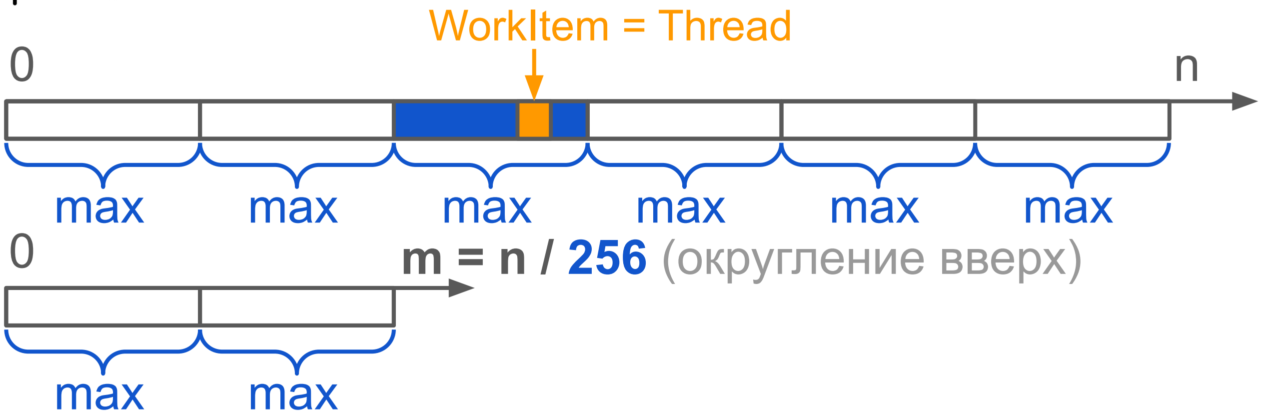
Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

```
max median time: 0.195 sec (+-0.00200998)
max median RAM bandwidth: 1.91041 GB/s
```

GPU NVIDIA RTX 4090



...

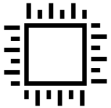
1 число

☐ - ответ

Все ли особенности мы обсудили?

```
max median time: 0.003 sec (+-0.00067082)
max median VRAM bandwidth: 124.176 GB/s
```

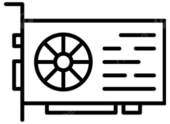
Пример: максимум по массиву



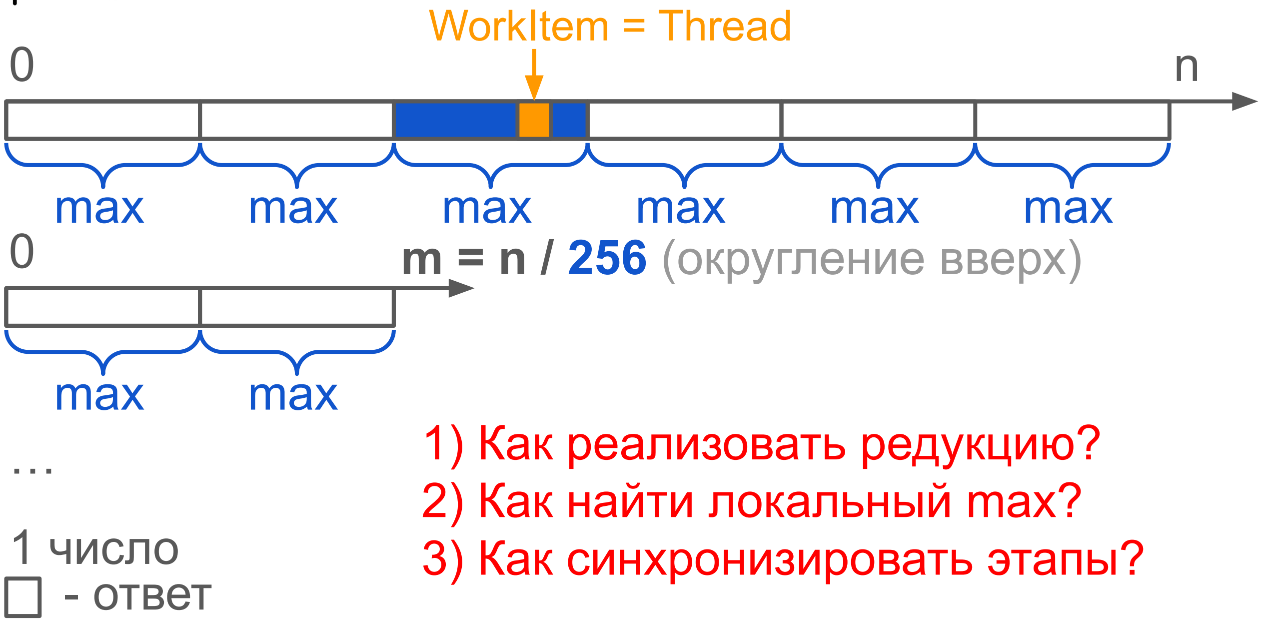
CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

```
max median time: 0.195 sec (+-0.00200998)
max median RAM bandwidth: 1.91041 GB/s
```



GPU NVIDIA RTX 4090



1) Как реализовать редукцию?
2) Как найти локальный max?
3) Как синхронизировать этапы?

```
max median time: 0.003 sec (+-0.00067082)
max median VRAM bandwidth: 124.176 GB/s
```

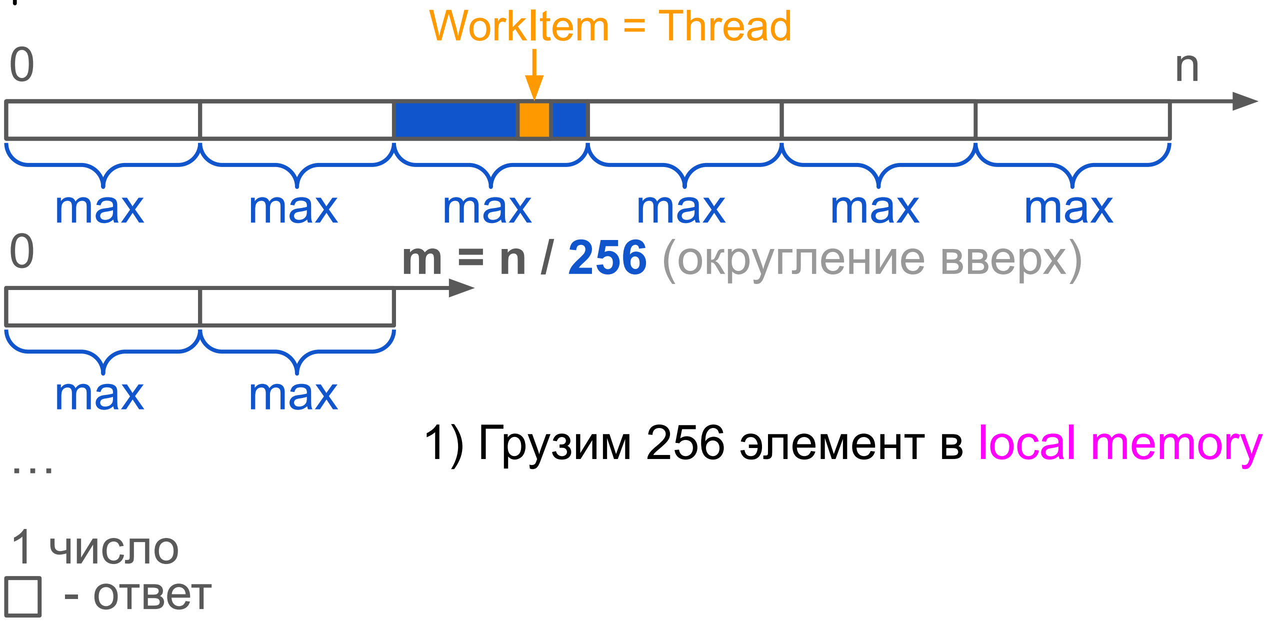
Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

```
max median time: 0.195 sec (+-0.00200998)
max median RAM bandwidth: 1.91041 GB/s
```

GPU NVIDIA RTX 4090



m = n / **256** (округление вверх)

1) Грузим 256 элемент в local memory

```
max median time: 0.003 sec (+-0.00067082)
max median VRAM bandwidth: 124.176 GB/s
```

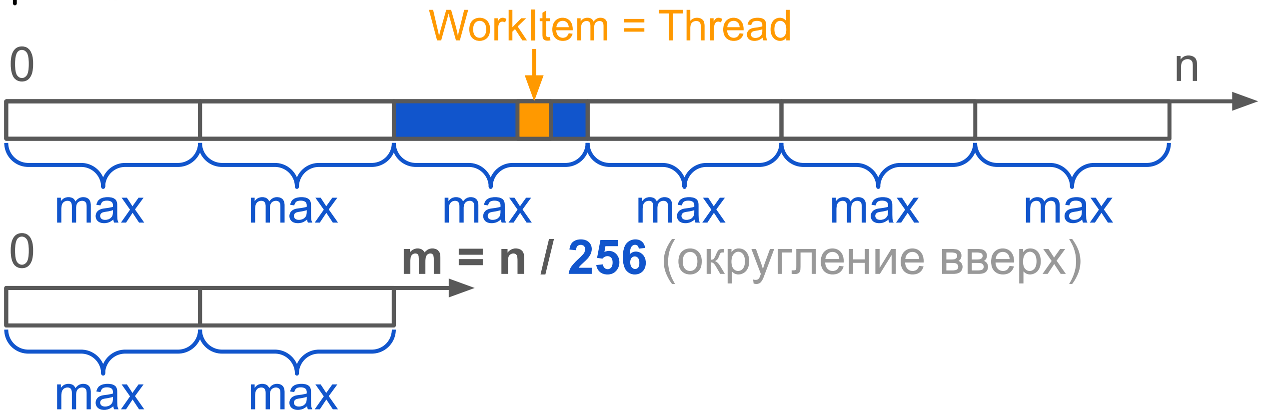
Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

```
max median time: 0.195 sec (+-0.00200998)
max median RAM bandwidth: 1.91041 GB/s
```

GPU NVIDIA RTX 4090



…

1 число

☐ - ответ

1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**

```
max median time: 0.003 sec (+-0.00067082)
max median VRAM bandwidth: 124.176 GB/s
```

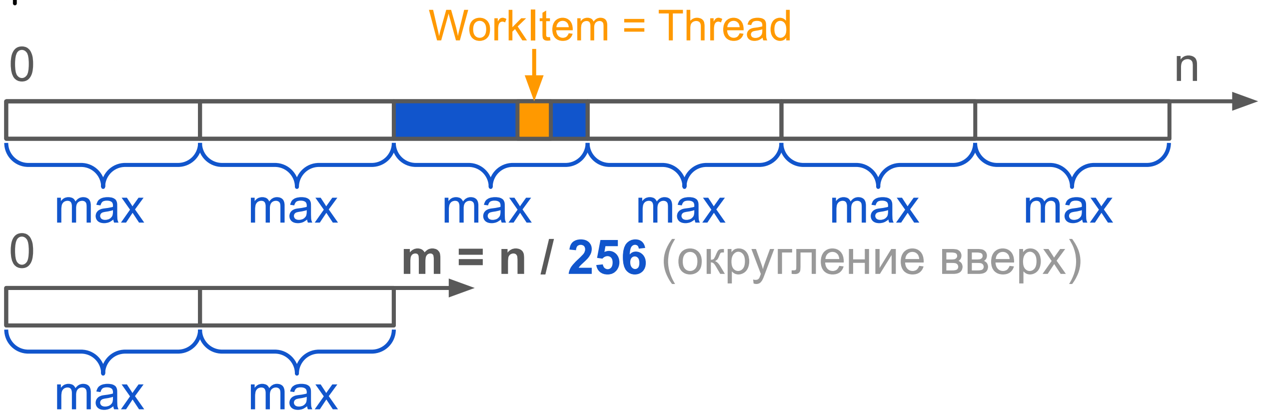
Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

```
max median time: 0.195 sec (+-0.00200998)
max median RAM bandwidth: 1.91041 GB/s
```

GPU NVIDIA RTX 4090



…

1 число

□ - ответ

1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**
3) Мастер-поток группы пишет **max**

```
max median time: 0.003 sec (+-0.00067082)
max median VRAM bandwidth: 124.176 GB/s
```

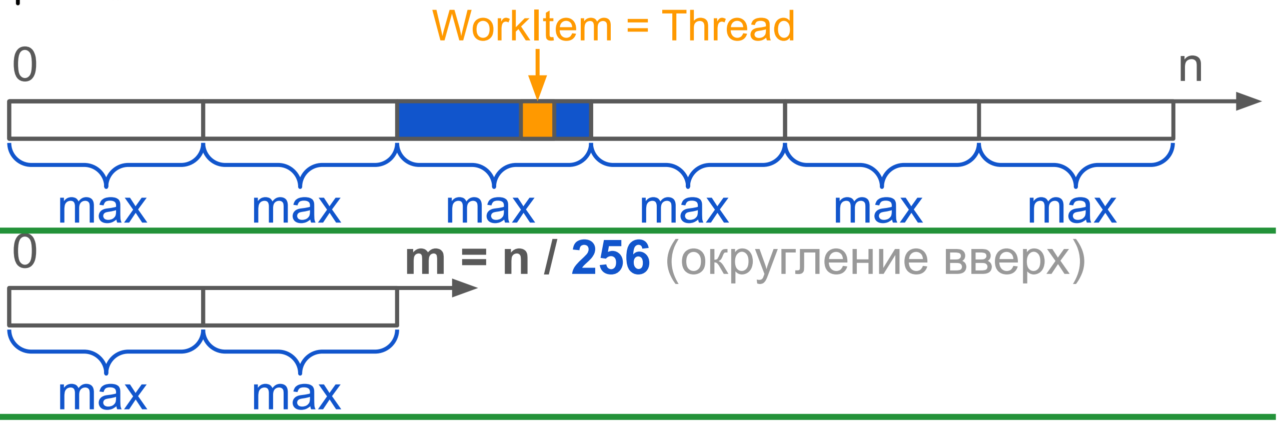
Пример: максимум по массиву

CPU Intel 13700K

```
max = -FLT_MAX;
for (size_t i = 0; i < n; ++i) {
    max = std::max(max, as[i]);
}
```

```
max median time: 0.195 sec (+-0.00200998)
max median RAM bandwidth: 1.91041 GB/s
```

GPU NVIDIA RTX 4090



…

1 число

☐ - ответ

1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**
3) Мастер-поток группы пишет **max**
4) Синхронизация запусками kernel-ов

```
max median time: 0.003 sec (+-0.00067082)
max median VRAM bandwidth: 124.176 GB/s
```

Пример: максимум по массиву

```
// Чтение данных
if (global_id < params.n) {
    local_data[local_id] = a[global_id];
} else {
    // Если размер входного массива не кратен рабочей группе
    // то проще потокам за пределами массива подсунуть
    // нейтральный элемент (с точки зрения поиска максимума)
    local_data[local_id] = -FLT_MAX;
}
```

1) Грузим 256 элемент в local memory

Пример: максимум по массиву

```
// Чтение данных
if (global_id < params.n) {
    local_data[local_id] = a[global_id];
} else {
    // Если размер входного массива не кратен рабочей группе
    // то проще потокам за пределами массива подсунуть
    // нейтральный элемент (с точки зрения поиска максимума)
    local_data[local_id] = -FLT_MAX;
}
```

```
// Мастер поток ищет максимум и записывает в выход
if (local_id == 0) {
    float max_val = local_data[0];
    for (uint i = 1; i < group_size; ++i) {
        if (local_data[i] > max_val) {
            max_val = local_data[i];
        }
    }
```

1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**

Пример: максимум по массиву

```
// Чтение данных
if (global_id < params.n) {
    local_data[local_id] = a[global_id];
} else {
    // Если размер входного массива не кратен рабочей группе
    // то проще потокам за пределами массива подсунуть
    // нейтральный элемент (с точки зрения поиска максимума)
    local_data[local_id] = -FLT_MAX;
}
```

```
// Мастер поток ищет максимум и записывает в выход
if (local_id == 0) {
    float max_val = local_data[0];
    for (uint i = 1; i < group_size; ++i) {
        if (local_data[i] > max_val) {
            max_val = local_data[i];
        }
    }
    a_reduced[group_id] = max_val;
}
```

1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**
3) Мастер-поток группы пишет **max**

Пример: максимум по массиву

```
    // Чтение данных
    if (global_id < params.n) {
        local_data[local_id] = a[global_id];
    } else {
        // Если размер входного массива не кратен рабочей группе
        // то проще потокам за пределами массива подсунуть
        // нейтральный элемент (с точки зрения поиска максимума)
        local_data[local_id] = -FLT_MAX;
    }
```

```
    // Мастер поток ищет максимум и записывает в выход
    if (local_id == 0) {
        float max_val = local_data[0];
        for (uint i = 1; i < group_size; ++i) {
            if (local_data[i] > max_val) {
                max_val = local_data[i];
            }
        }
        a_reduced[group_id] = max_val;
    }
```

1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**
3) Мастер-поток группы пишет **max**

Нет ли баги?

Пример: максимум по массиву

```
    // Чтение данных
    if (global_id < params.n) {
        local_data[local_id] = a[global_id];
    } else {
        // Если размер входного массива не кратен рабочей группе
        // то проще потокам за пределами массива подсунуть
        // нейтральный элемент (с точки зрения поиска максимума)
        local_data[local_id] = -FLT_MAX;
    }

    // Барьер для синхронизации всей рабочей группы
    barrier();

    // Мастер поток ищет максимум и записывает в выход
    if (local_id == 0) {
        float max_val = local_data[0];
        for (uint i = 1; i < group_size; ++i) {
            if (local_data[i] > max_val) {
                max_val = local_data[i];
            }
        }
        a_reduced[group_id] = max_val;
    }
```

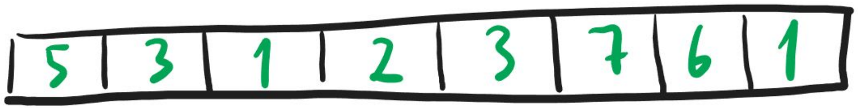
1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**
3) Мастер-поток группы пишет **max**

Пример: максимум по массиву

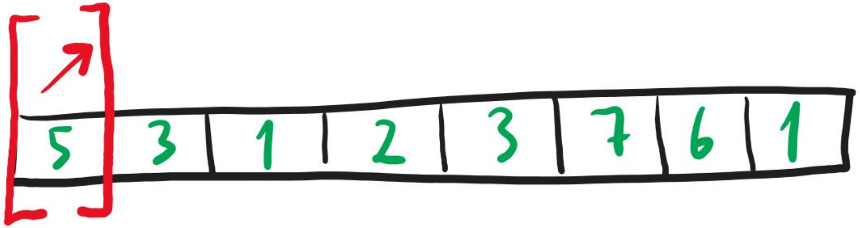
```
    // Чтение данных
    if (global_id < params.n) {
        local_data[local_id] = a[global_id];
    } else {
        // Если размер входного массива не кратен рабочей группе
        // то проще потокам за пределами массива подсунуть
        // нейтральный элемент (с точки зрения поиска максимума)
        local_data[local_id] = -FLT_MAX;
    }

    // Барьер для синхронизации всей рабочей группы
    barrier();

    // Мастер поток ищет максимум и записывает в выход
    if (local_id == 0) {
        float max_val = local_data[0];
        for (uint i = 1; i < group_size; ++i) {
            if (local_data[i] > max_val) {
                max_val = local_data[i];
            }
        }
        a_reduced[group_id] = max_val;
    }
```

1) Грузим 256 элемент в local memory
2) Мастер-поток группы ищет **max**
3) Мастер-поток группы пишет **max**
4) Синхронизация запусками kernel-ов

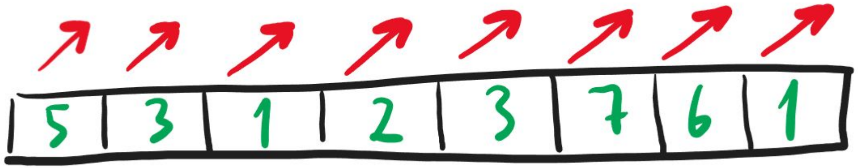
Merge-sort



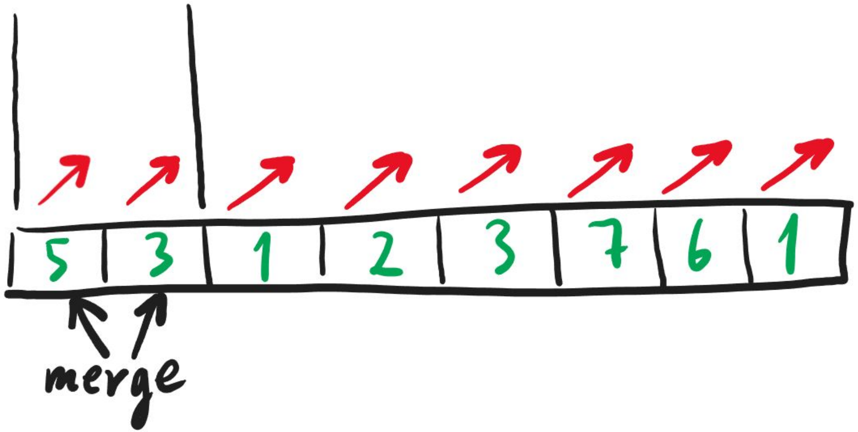
Merge-sort



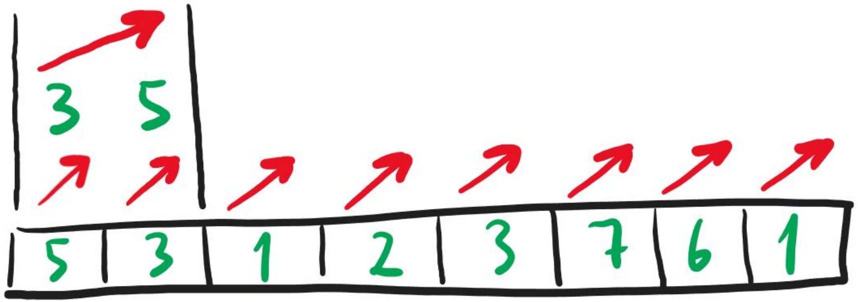
Merge-sort



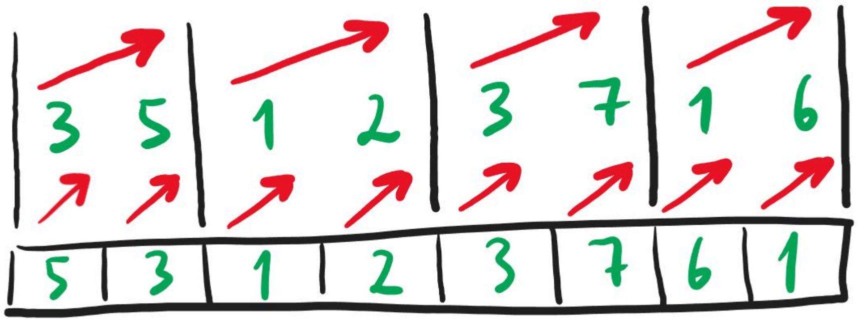
Merge-sort



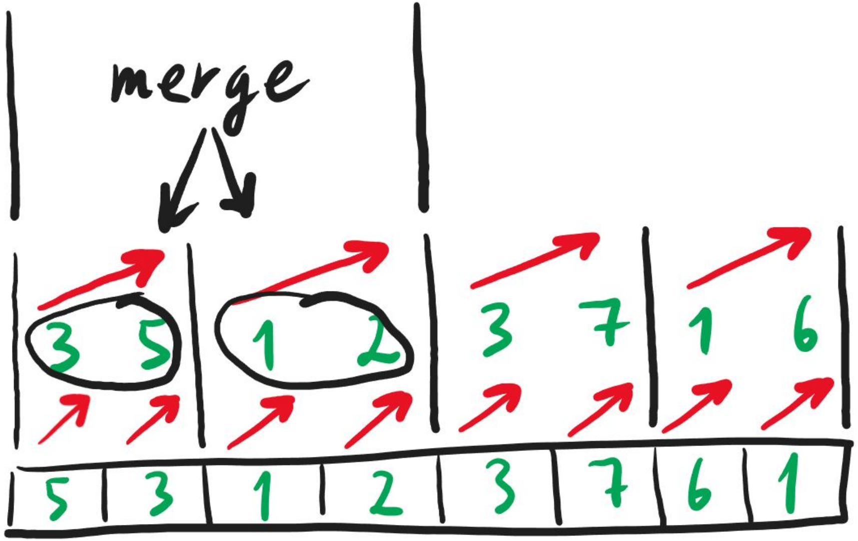
Merge-sort



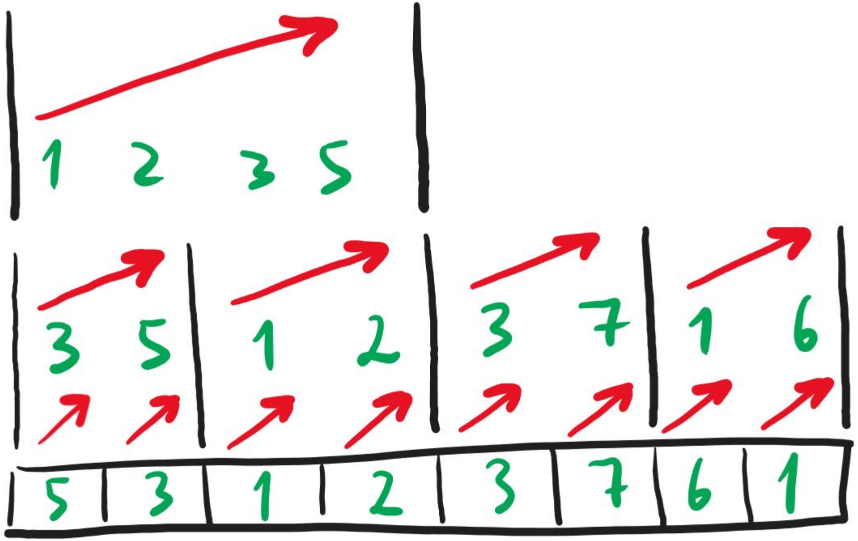
Merge-sort



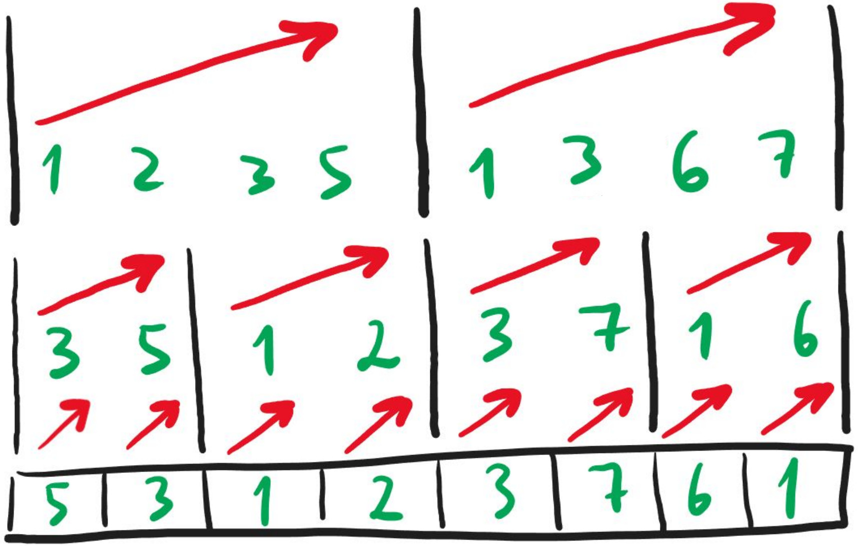
Merge-sort



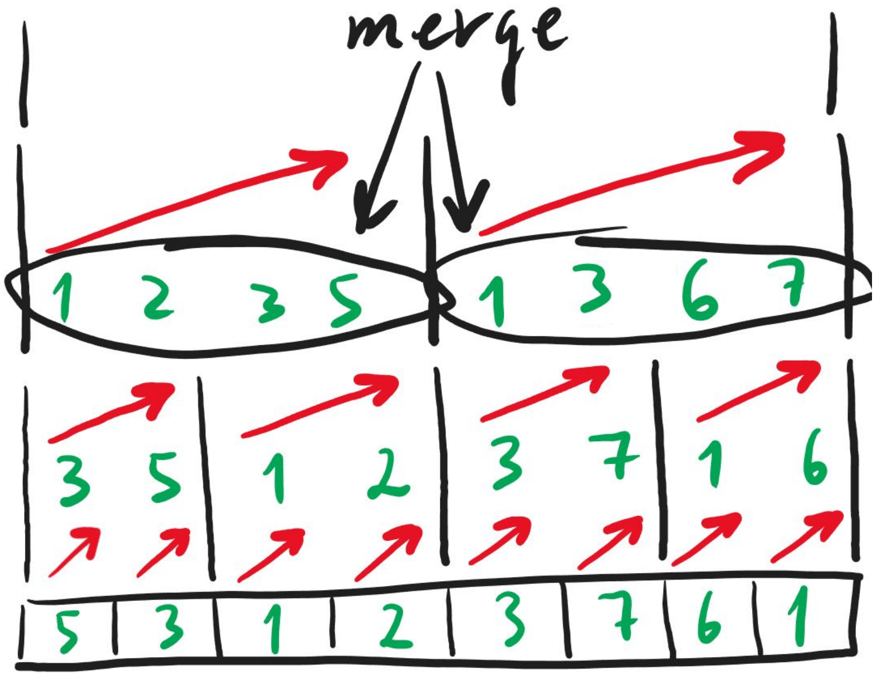
Merge-sort



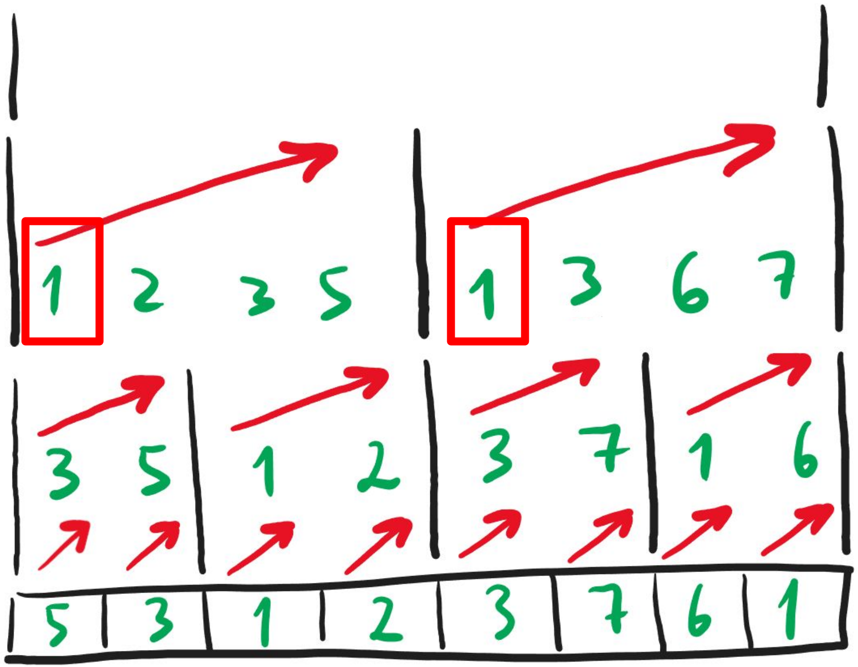
Merge-sort



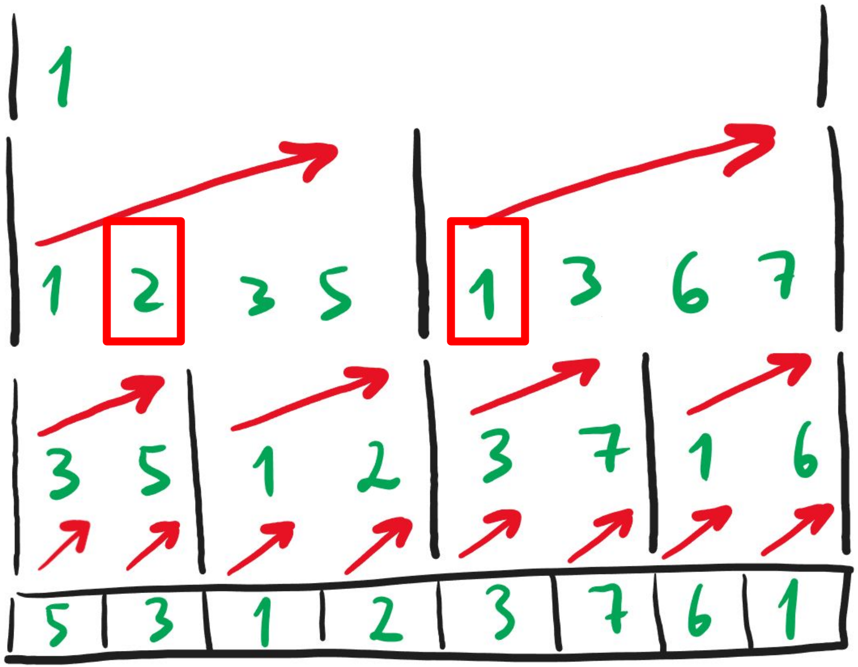
Merge-sort



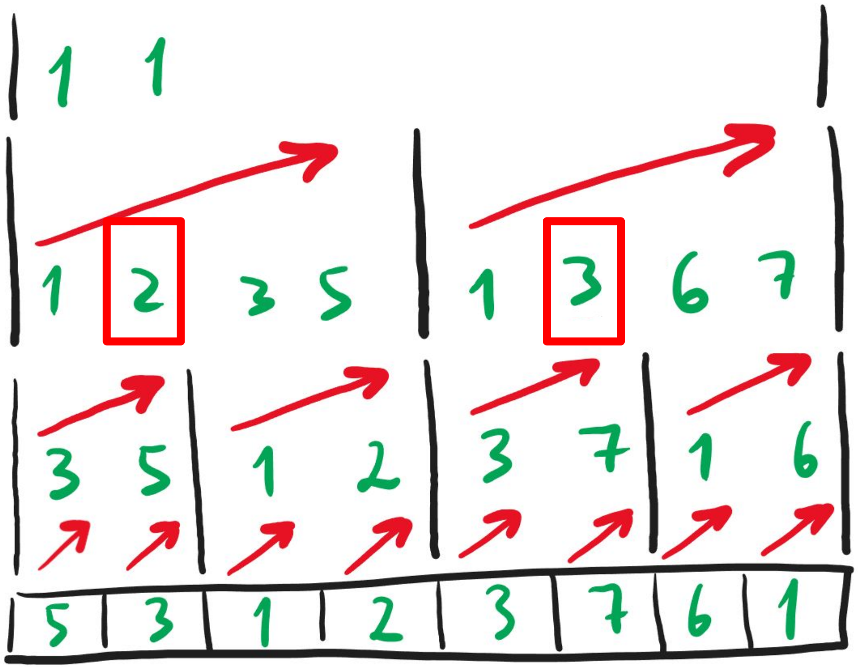
Merge-sort



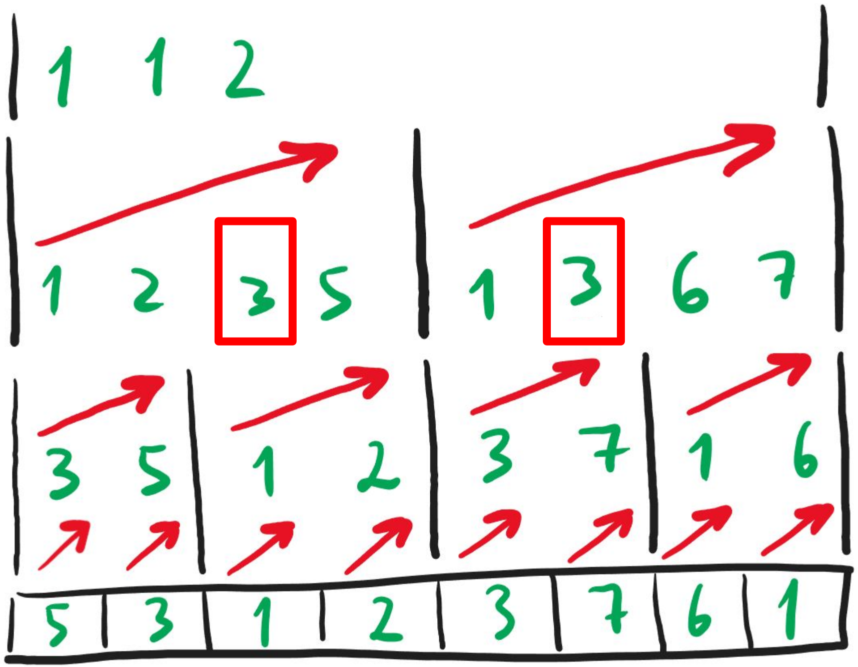
Merge-sort



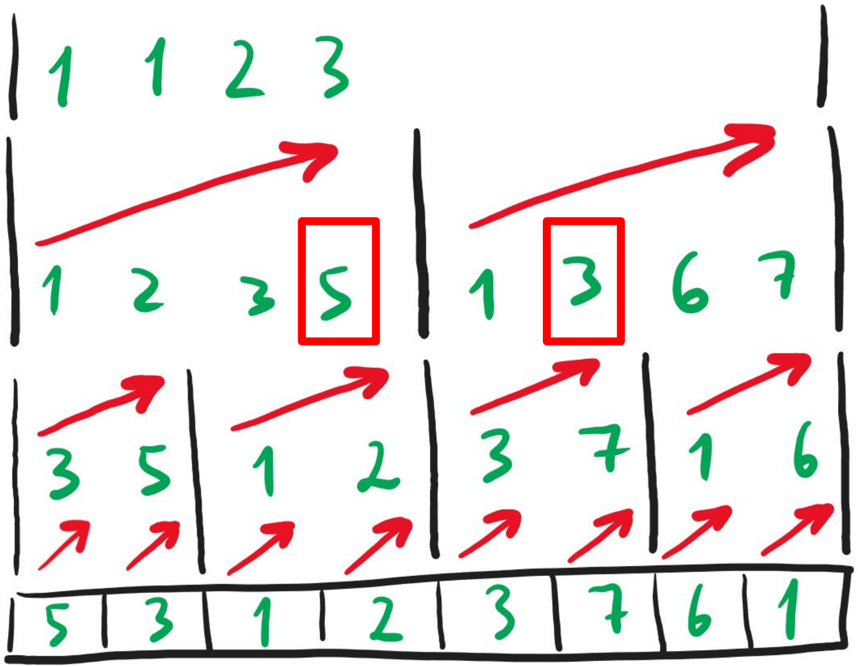
Merge-sort



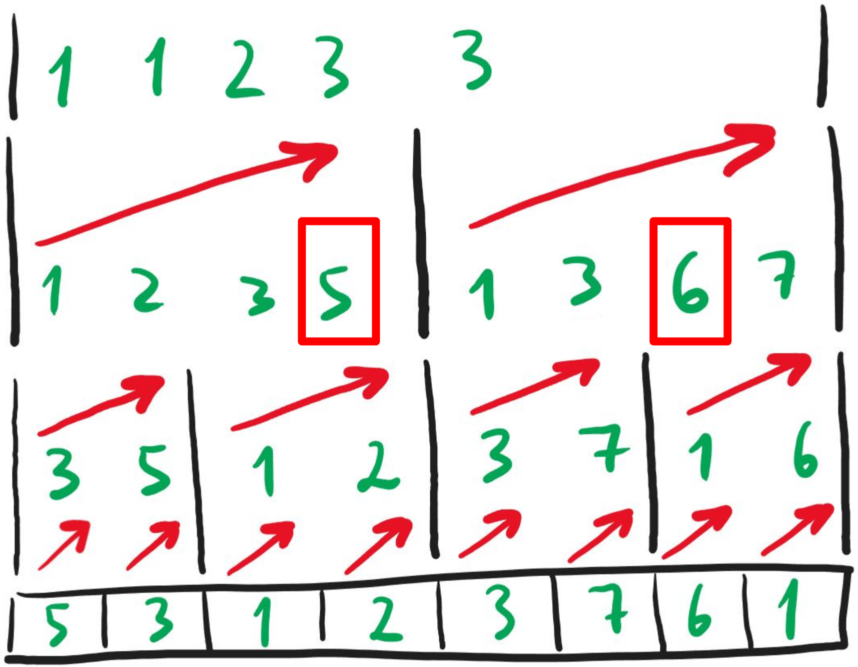
Merge-sort



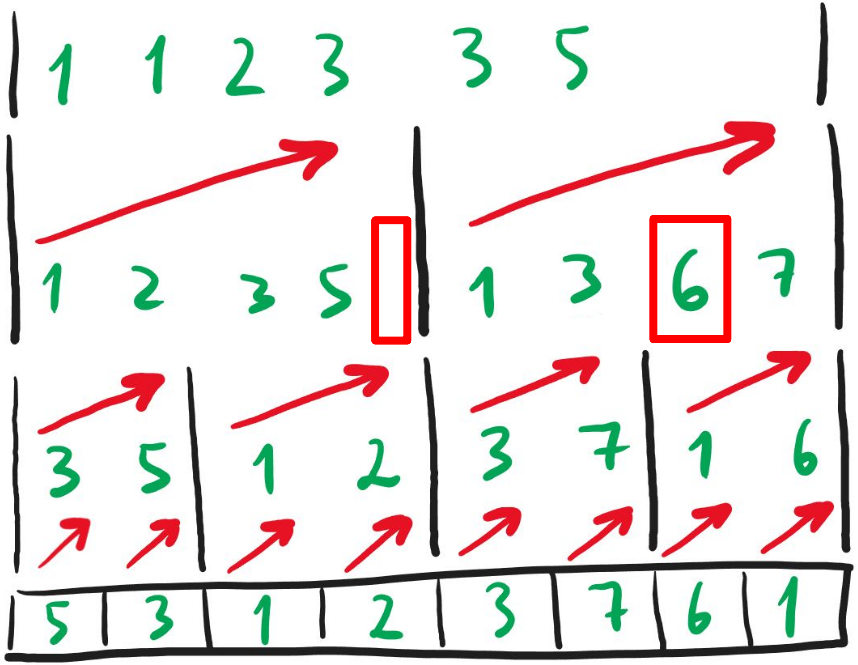
Merge-sort



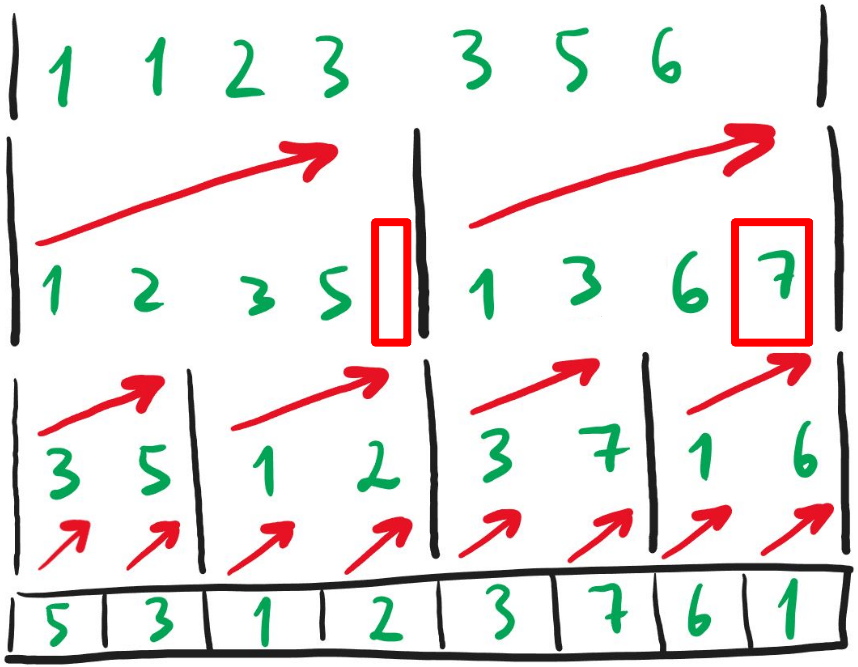
Merge-sort



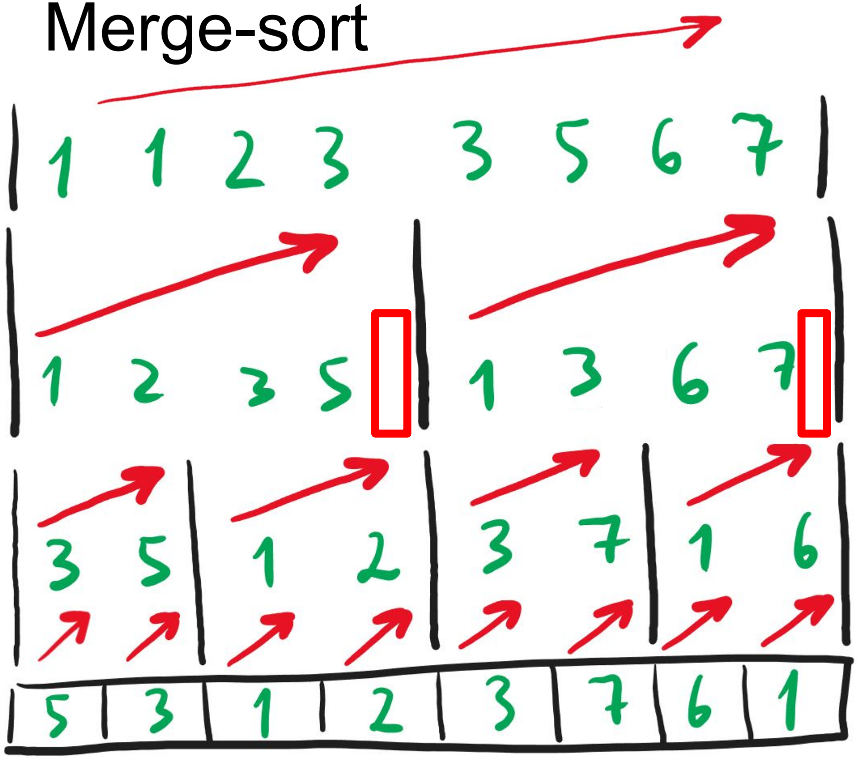
Merge-sort



Merge-sort

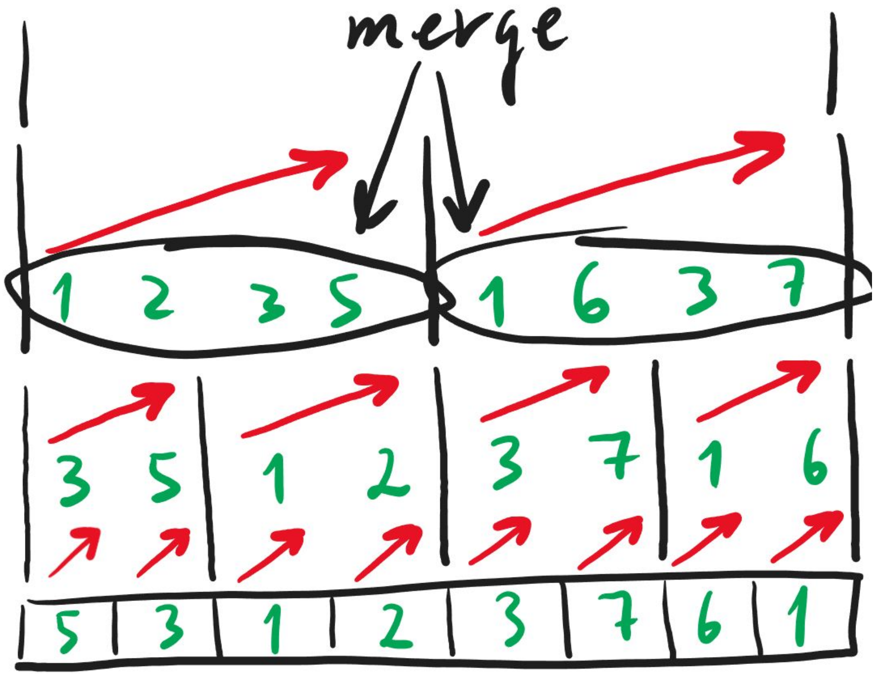


Merge-sort



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

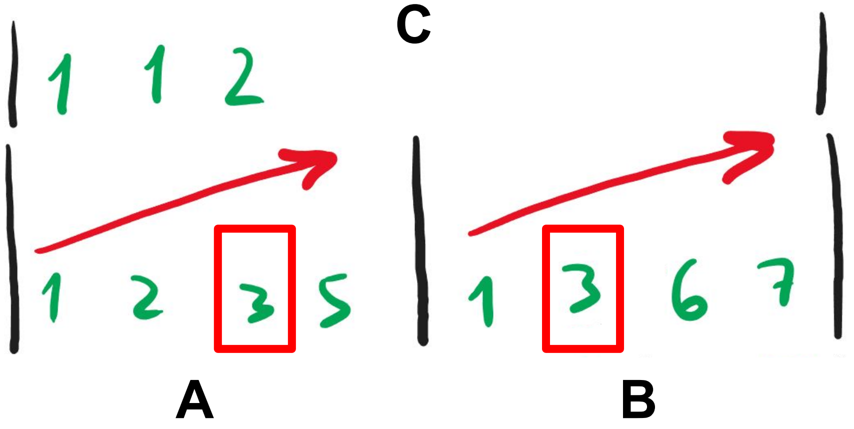
Merge-sort



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять операцию merge супер многопоточно!

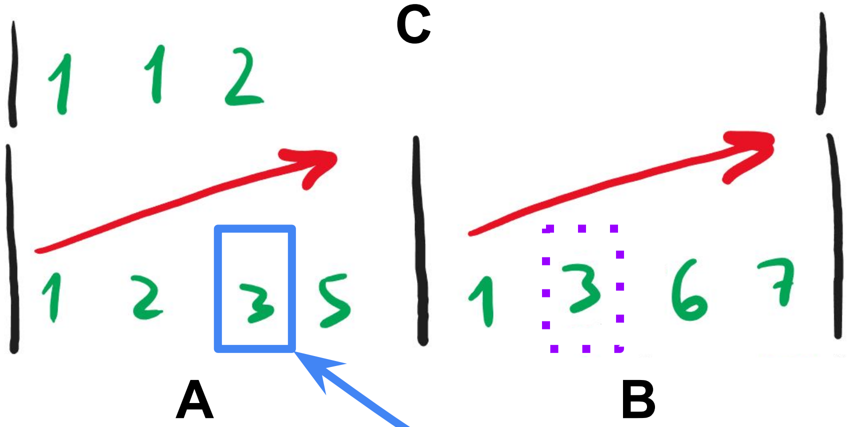
Merge-sort



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять операцию merge супер многопоточно!

Merge-sort

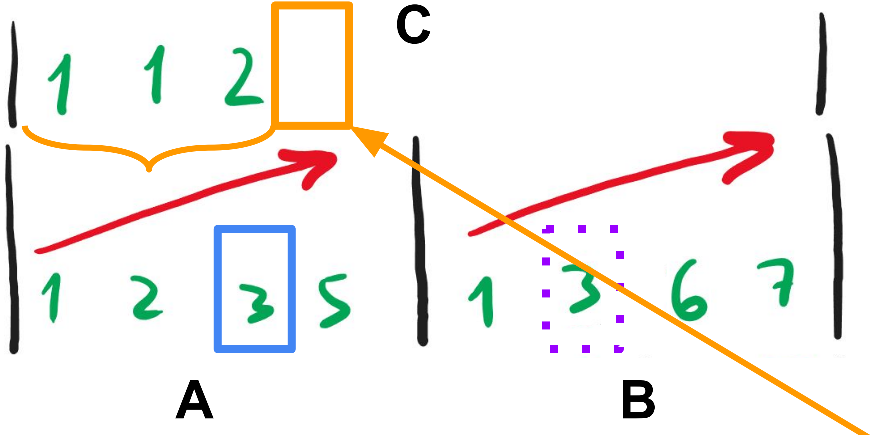


Как отсортировать на видеокарте? Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять операцию merge супер многопоточно!

Например один поток - **одно число из A**

Merge-sort



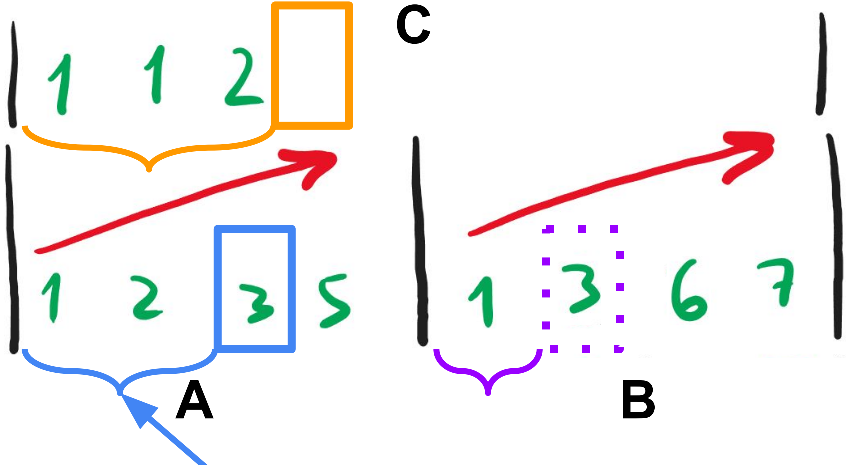
Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять операцию merge супер многопоточно!

Например один поток - **одно число из А**

На какую позицию в С записать его?

Merge-sort



Знаем ли мы сколько чисел до нас в A?

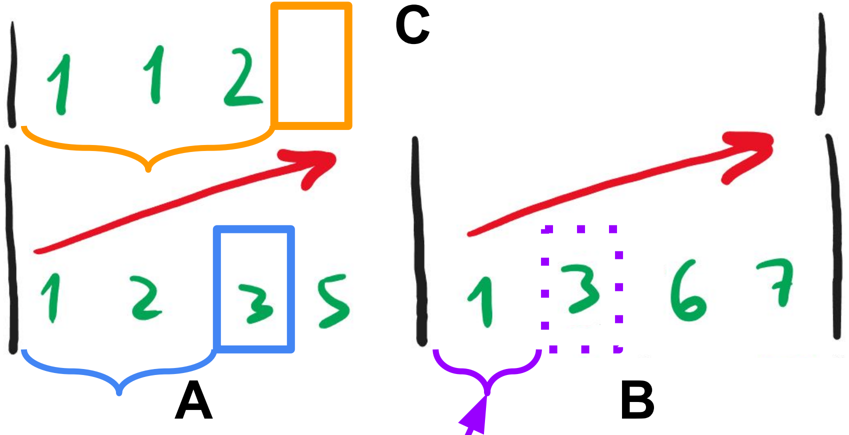
Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять операцию merge супер многопоточно!

Например один поток - **одно число из A**

На какую позицию в C записать его?

Merge-sort



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять операцию merge супер многопоточно!

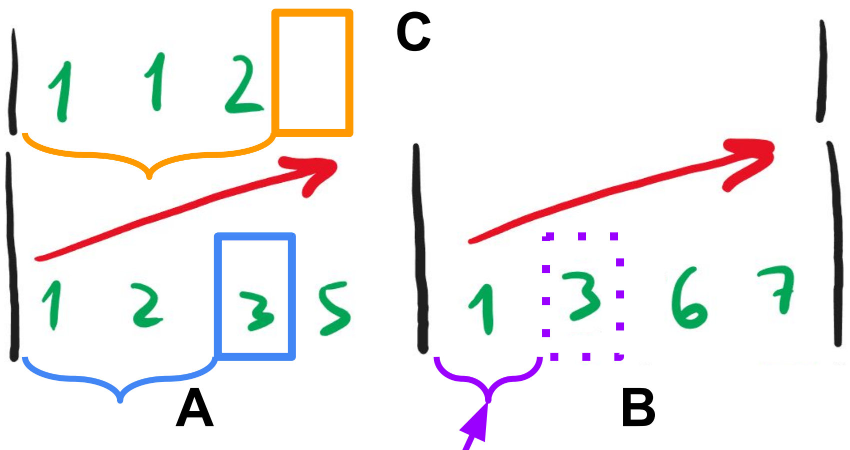
Например один поток - **одно число из A**

Знаем ли мы сколько чисел до нас в A?

На какую позицию в C записать его?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Merge-sort



Как отсортировать на видеокарте? Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять операцию merge супер многопоточно!

Например один поток - **одно число из A**

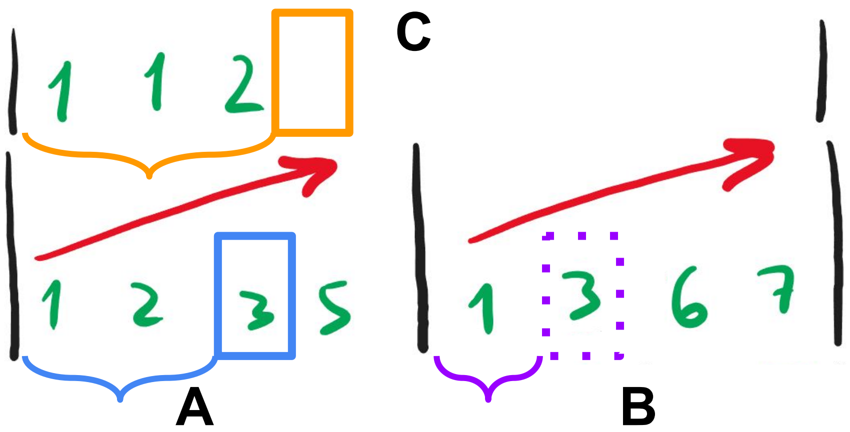
На какую позицию в C записать его?

Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! O(log N)

Merge-sort



Пусть **размер** **A** **и** **B** **- N** элементов.
Пусть у нас **K ядер** в видеокарте.

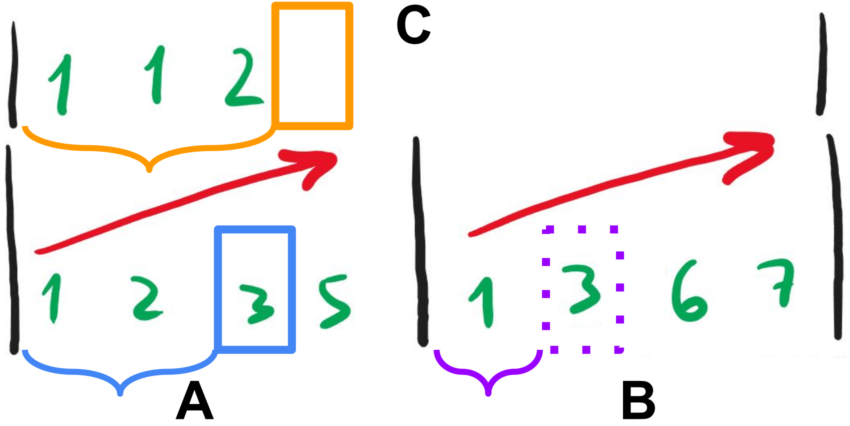
Какая асимптотика **merge** операции?

Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! O(log N)

Merge-sort



Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

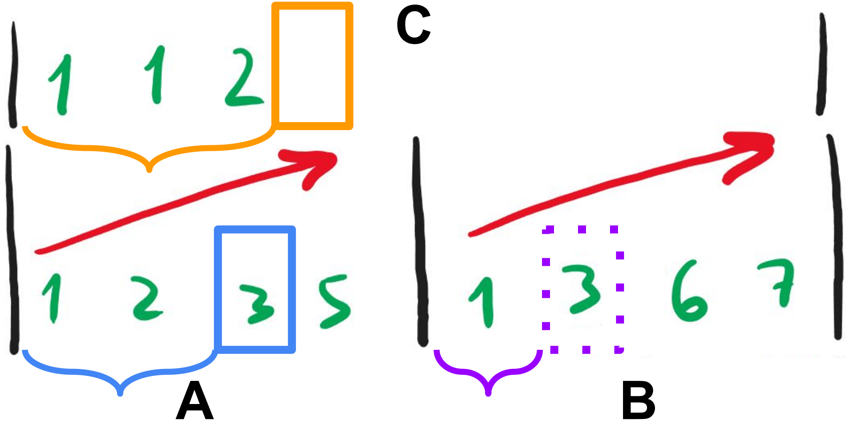
O(? * $\boxed{\text{O(log N)}}$ / ?)

Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! $\boxed{\text{O(log N)}}$

Merge-sort



Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

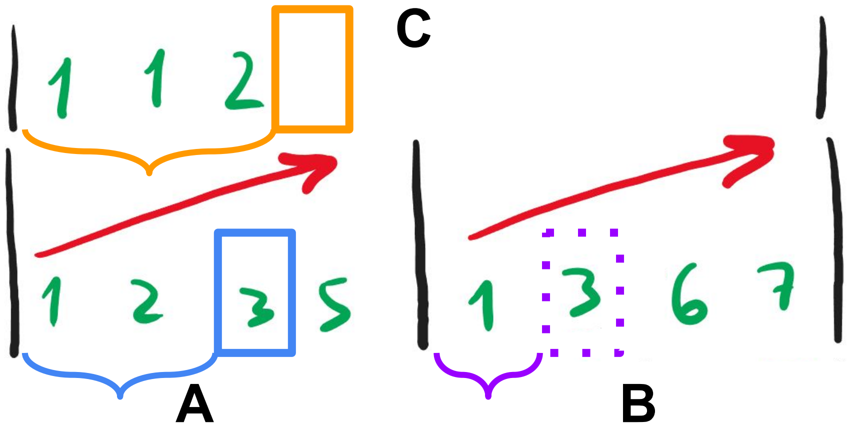
O(N * $\boxed{\text{O(log N)}}$ / ?)

Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! $\boxed{\text{O(log N)}}$

Merge-sort



Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

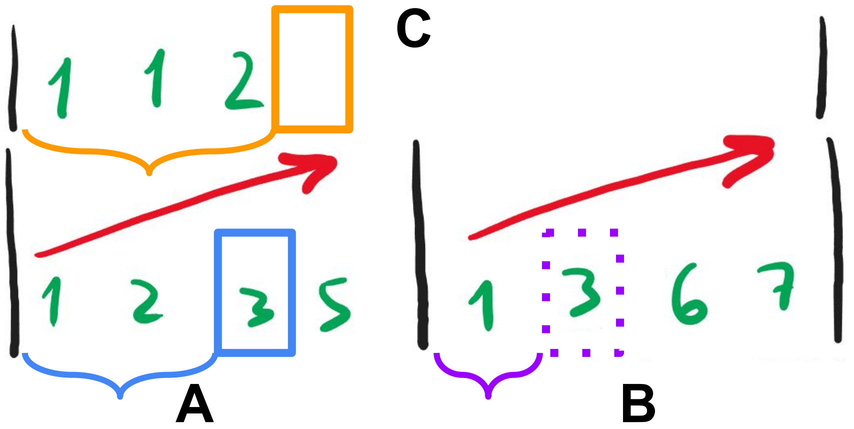
O(N * O(log N) / K)

Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! O(log N)

Merge-sort



Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

O(N * O(log N) / K)

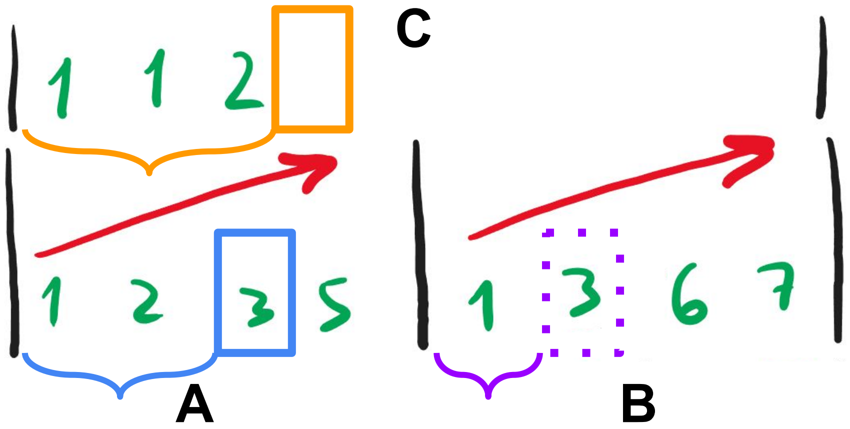
Всего у нас **O(log N)** операций слияния

Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! O(log N)

Merge-sort



Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

O(N * O(log N) / K)

Всего у нас **O(log N)** операций слияния

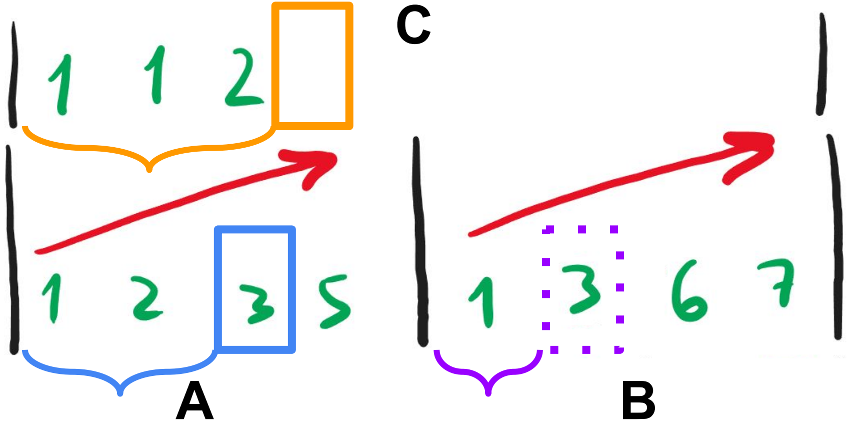
Какая асимптотика на CPU?

Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! O(log N)

Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! $\boxed{\mathbf{O(\log N)}}$

Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

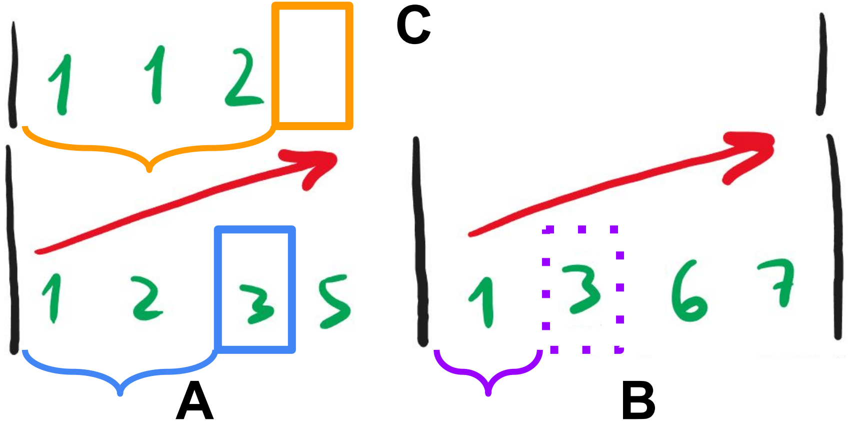
Какая асимптотика **merge** операции?

$\mathbf{O(N * \boxed{O(\log N)} / K)}$

Всего у нас $\boxed{\mathbf{O(\log N)}}$ операций слияния

Какая асимптотика на CPU? $\mathbf{O(\boxed{\log N} * N)}$

Merge-sort



Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

$O(N * \boxed{O(\log N)} / K)$

Всего у нас $\boxed{O(\log N)}$ операций слияния

Какая асимптотика на CPU? $O(\boxed{\log N} * N)$

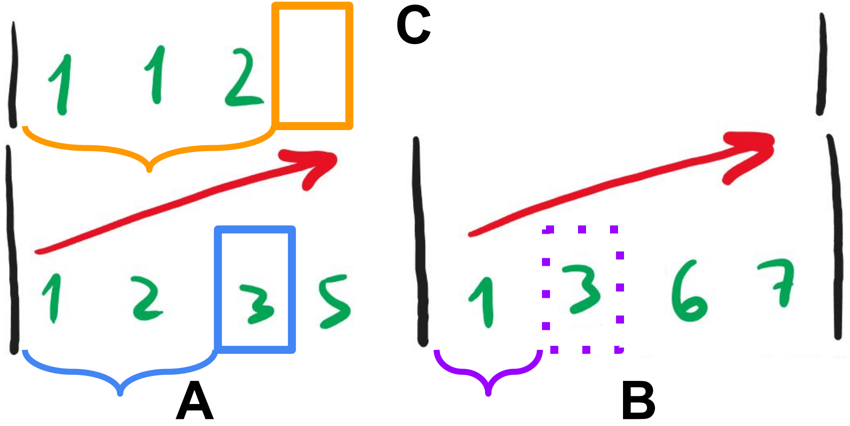
А на GPU? $O(\boxed{\log N} * ?)$

Знаем ли мы сколько чисел до нас в А?

А как найти сколько элементов в В **меньше** чем **наше число из А**?

Бинарный поиск! $\boxed{O(\log N)}$

Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B **меньше** чем **наше число из A**?

Бинарный поиск! $\boxed{\textbf{O(log N)}}$

Пусть **размер A и B - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

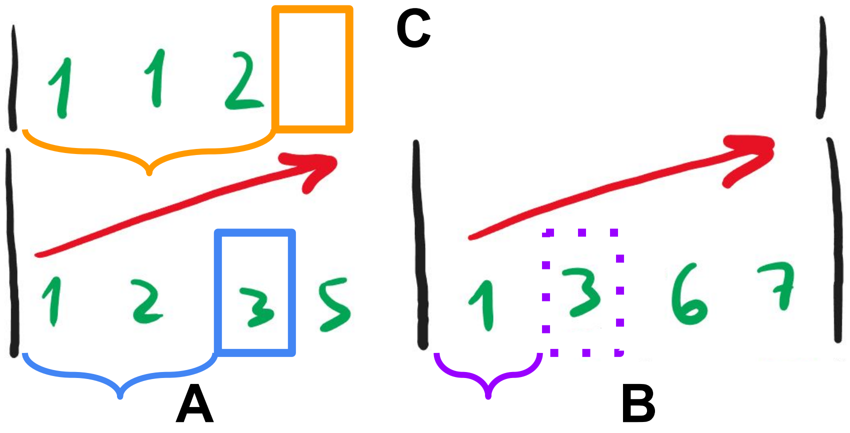
O(N * $\boxed{\textbf{O(log N)}}$ **/ K)**

Всего у нас $\boxed{\textbf{O(log N)}}$ операций слияния

Какая асимптотика на CPU? **O(**$\boxed{\textbf{log N}}$ *** N)**

А на GPU? O($\boxed{\textbf{log N}}$ *** N *** $\boxed{\textbf{log N}}$ **/ K)**

Merge-sort



Пусть **размер А и В - N** элементов.
Пусть у нас **K ядер** в видеокарте.

Какая асимптотика **merge** операции?

O(N * $\boxed{\text{O(log N)}}$ / K)

Всего у нас $\boxed{\text{O(log N)}}$ операций слияния

Какая асимптотика на CPU? **O($\boxed{\text{log N}}$ * N)**

А на GPU? **O($\boxed{\text{log N}}$ * N * $\boxed{\text{log N}}$ / K)**

Отличается в (**log N / K**) раз, **K - большое!**

Знаем ли мы сколько чисел до нас в А?

А как найти сколько элементов в В **меньше** чем **наше число из А**?

Бинарный поиск! $\boxed{\text{O(log N)}}$

Merge-sort	**N** = 33.554.432 ~ 3*10^7
CPU - Intel 13700K O(log N * N)	6.462 секунд
GPU - NVIDIA RTX 4090 O(log N * N * log N / K) Отличается в (**log N / K)** раз, где **K** = 16 тысяч ядер	0.011 секунд **Быстрее в 582 раза!**

Gipuma

Massively Parallel Multiview Stereopsis by Surface Normal Diffusion

Silvano Galliani Katrin Lasinger Konrad Schindler

Photogrammetry and Remote Sensing, ETH Zurich

Abstract

We present a new, massively parallel method for high-quality multiview matching. Our work builds on the Patchmatch *idea: starting from randomly generated 3D planes in scene space, the best-fitting planes are iteratively propagated and refined to obtain a 3D depth and normal field per view, such that a robust photo-consistency measure over all images is maximized. Our main novelties are on the one hand to formulate Patchmatch in scene space, which makes it possible to aggregate image similarity across multiple views and obtain more accurate depth maps. And on the other hand a modified, diffusion-like propagation scheme that can be massively parallelized and delivers dense multiview correspondence over ten 1.9-Megapixel images in 3 seconds, on a consumer-grade GPU. Our method uses a slanted support window and thus has no fronto-parallel bias; it is completely local and parallel, such that computation time scales linearly with image size, and inversely proportional to the number of parallel threads. Further-*

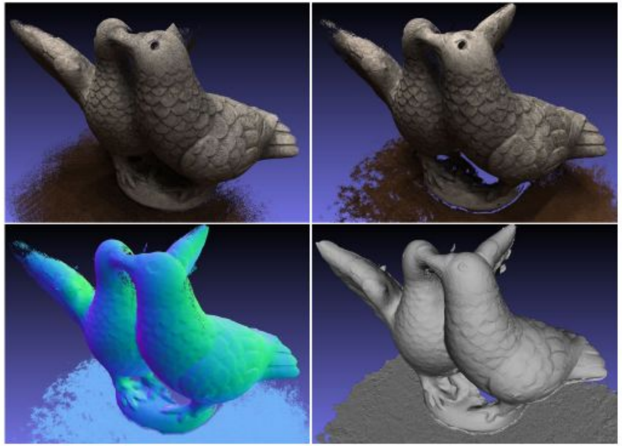
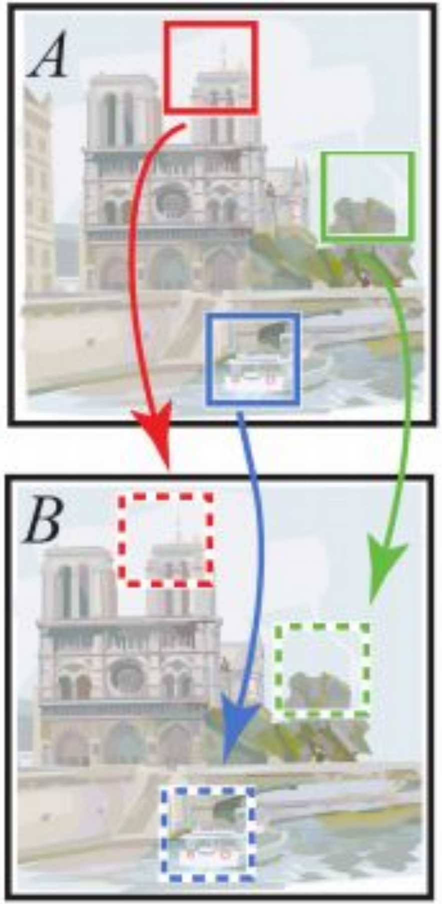
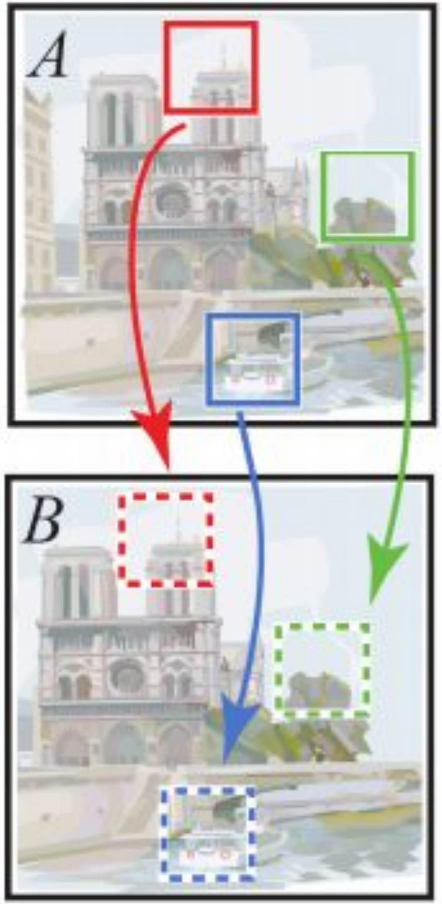


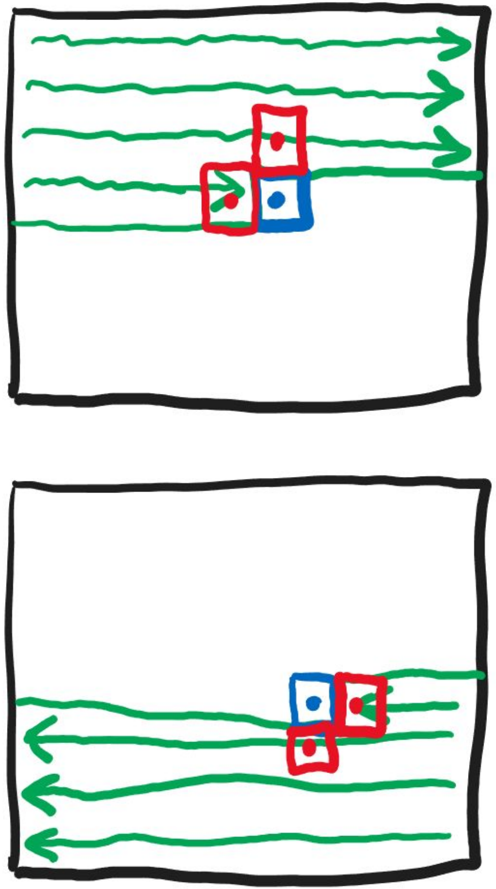
Figure 1: Results on one of the 80 evaluated objects on the DTU benchmark [22]. *Top left*: Ground truth point cloud; *top right*: reconstructed point cloud with texture; *bottom left*: color-coded surface normals; *bottom right*: reconstructed surface.

Gipuma



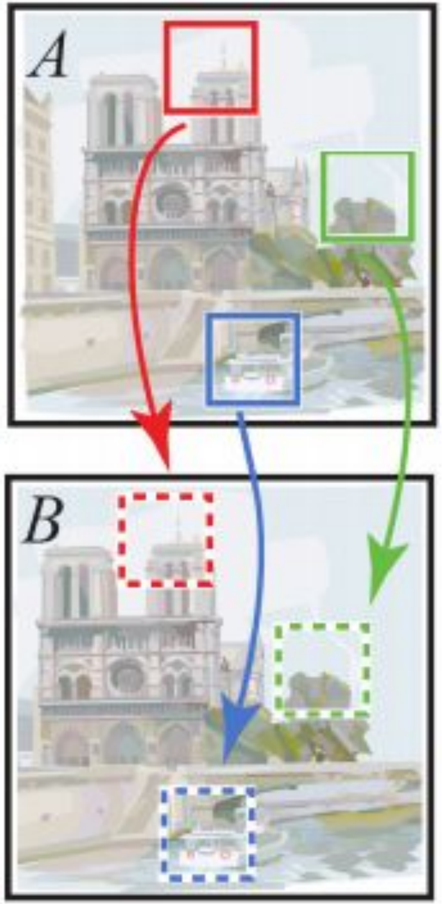
Gipuma

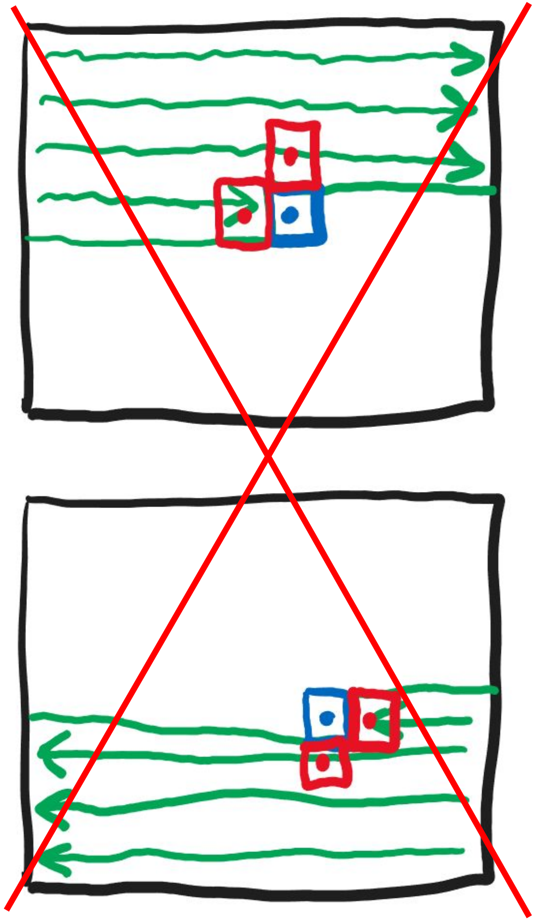




Gipuma

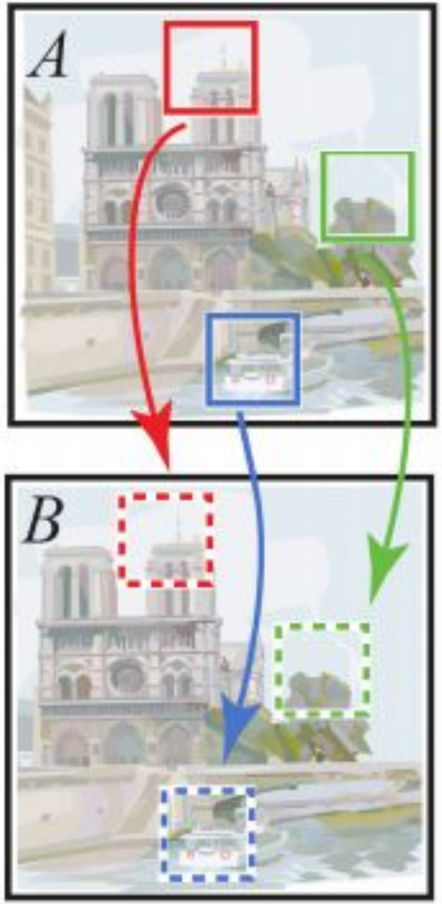
Как адаптировать для массового параллелизма?

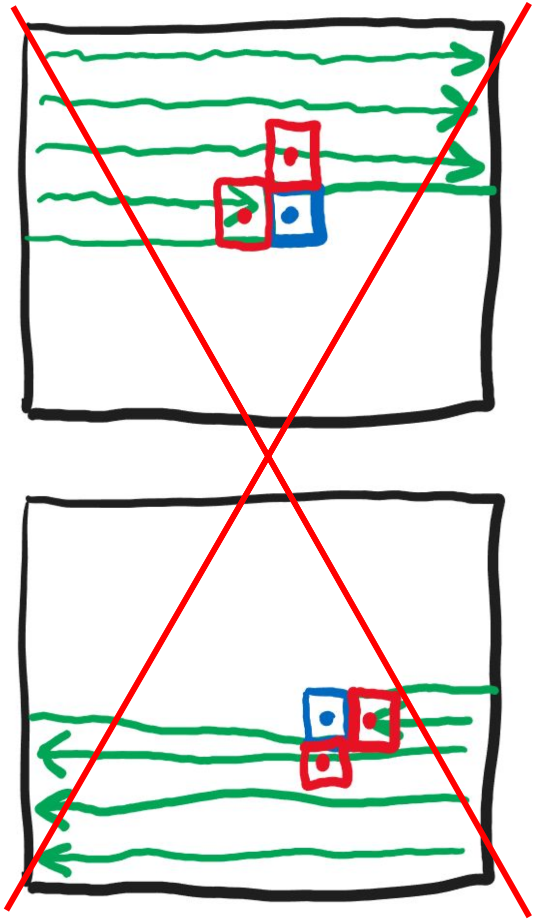


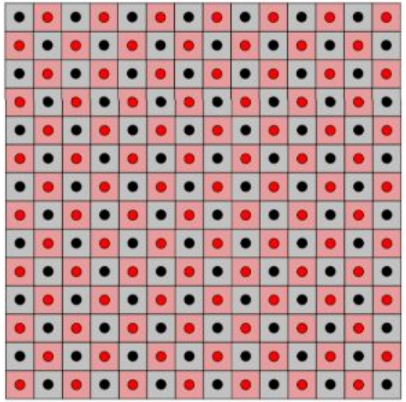


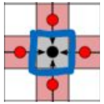
Gipuma

Как адаптировать для массового параллелизма?

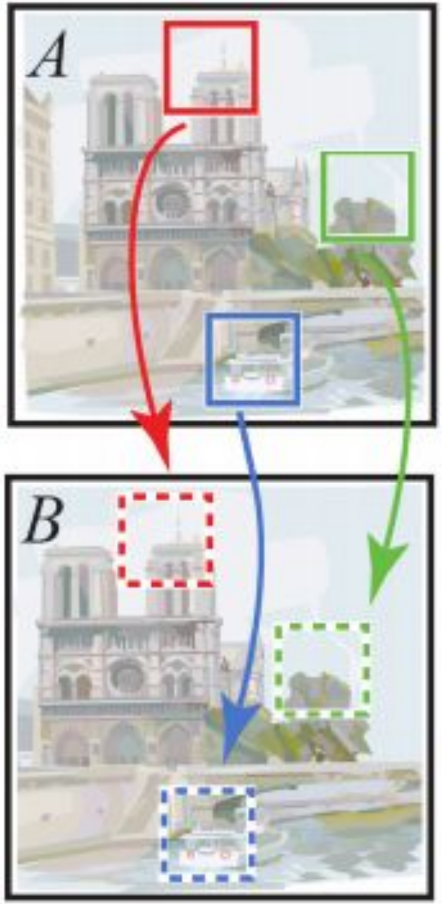




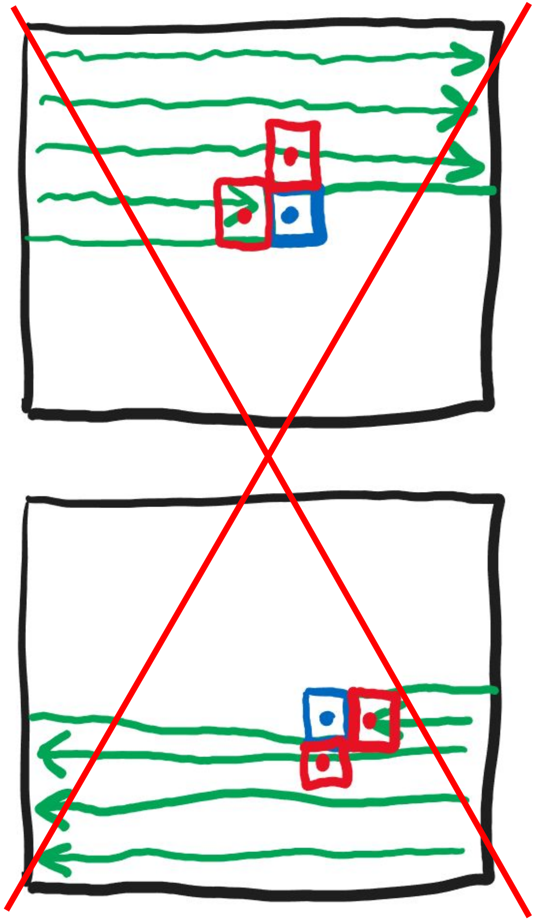


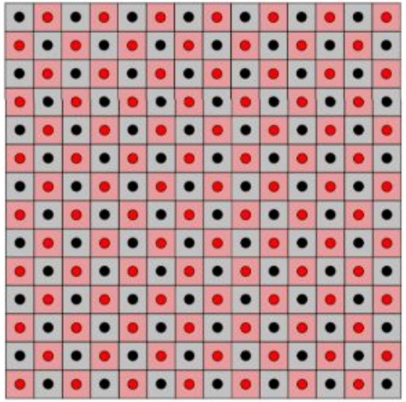


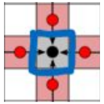
Gipuma



Как адаптировать для массового параллелизма?



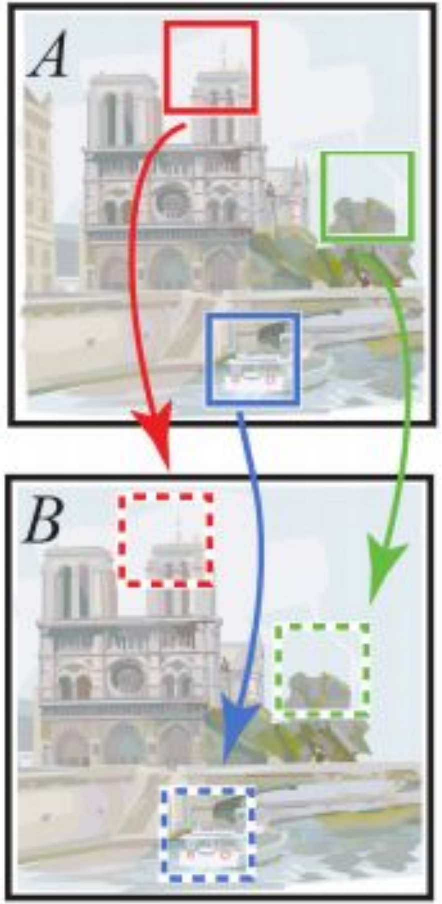


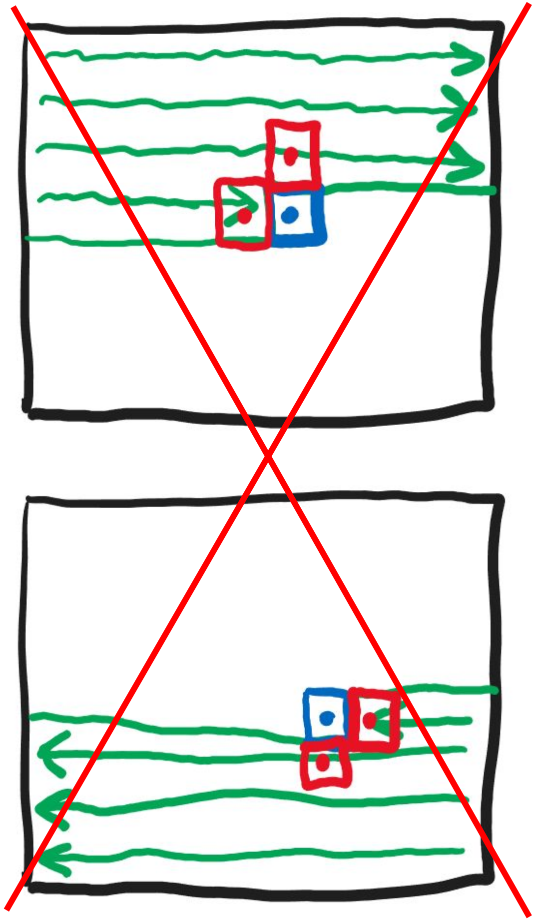


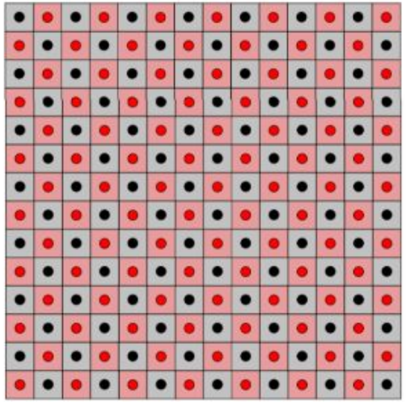
Как ускорить распространение информации? (правильного ответа)

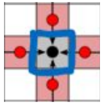
Gipuma

Как адаптировать для массового параллелизма?

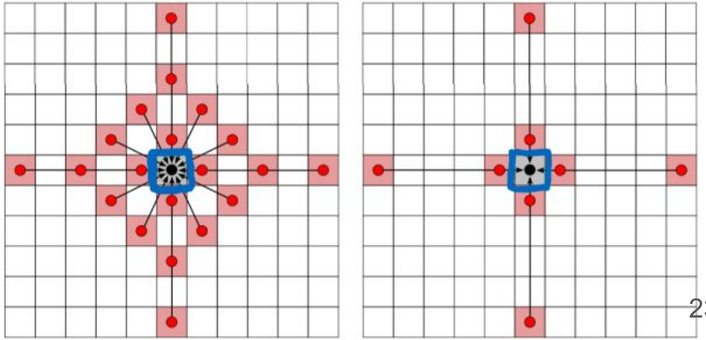








Как ускорить распространение информации? (правильного ответа)



Глава 7: Матрицы

транспонирование, умножение, tensor cores, DeepSeek

Транспонирование матрицы

2	4	-1
-10	5	11
18	-7	6

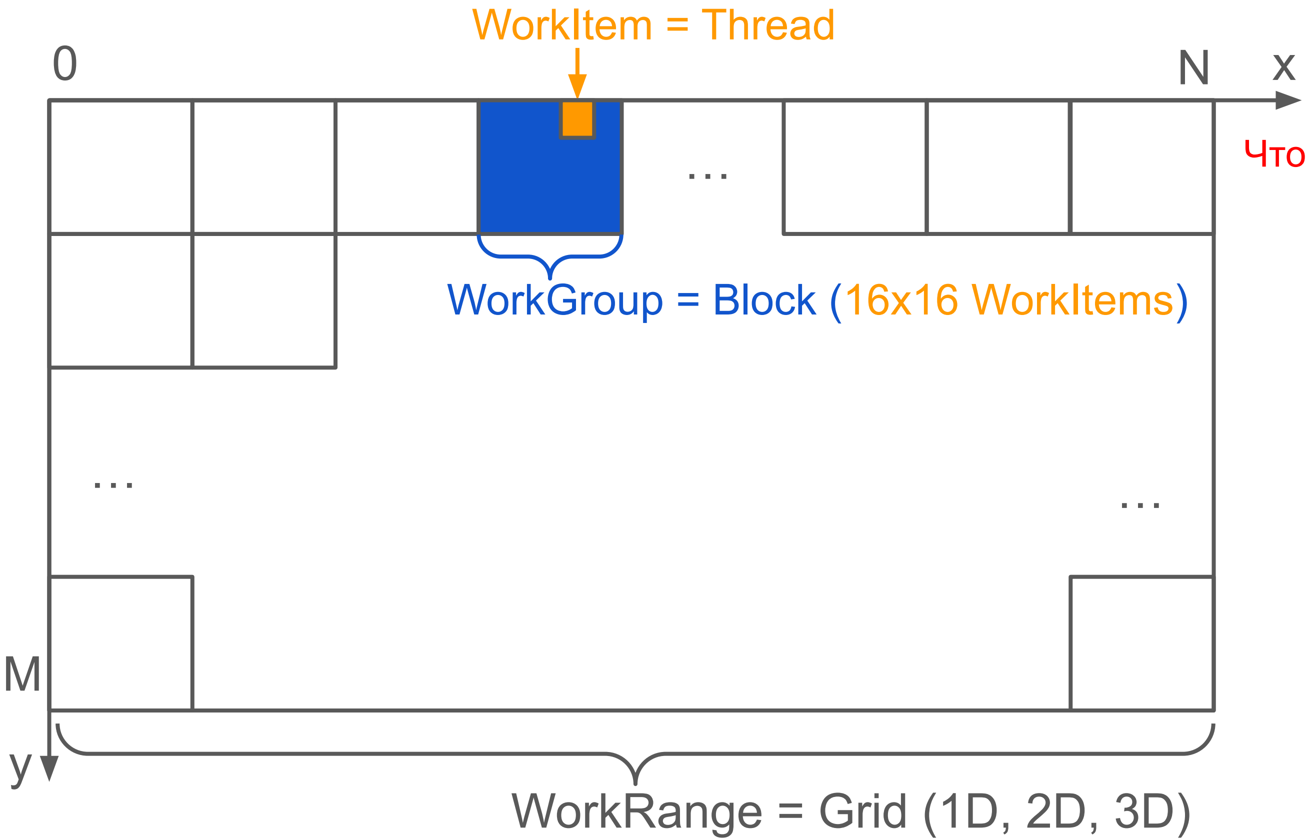
2	-10	18
4	5	-7
-1	11	6

Как это реализовать в модели массового параллелизма?

Какой размер рабочего пространства?

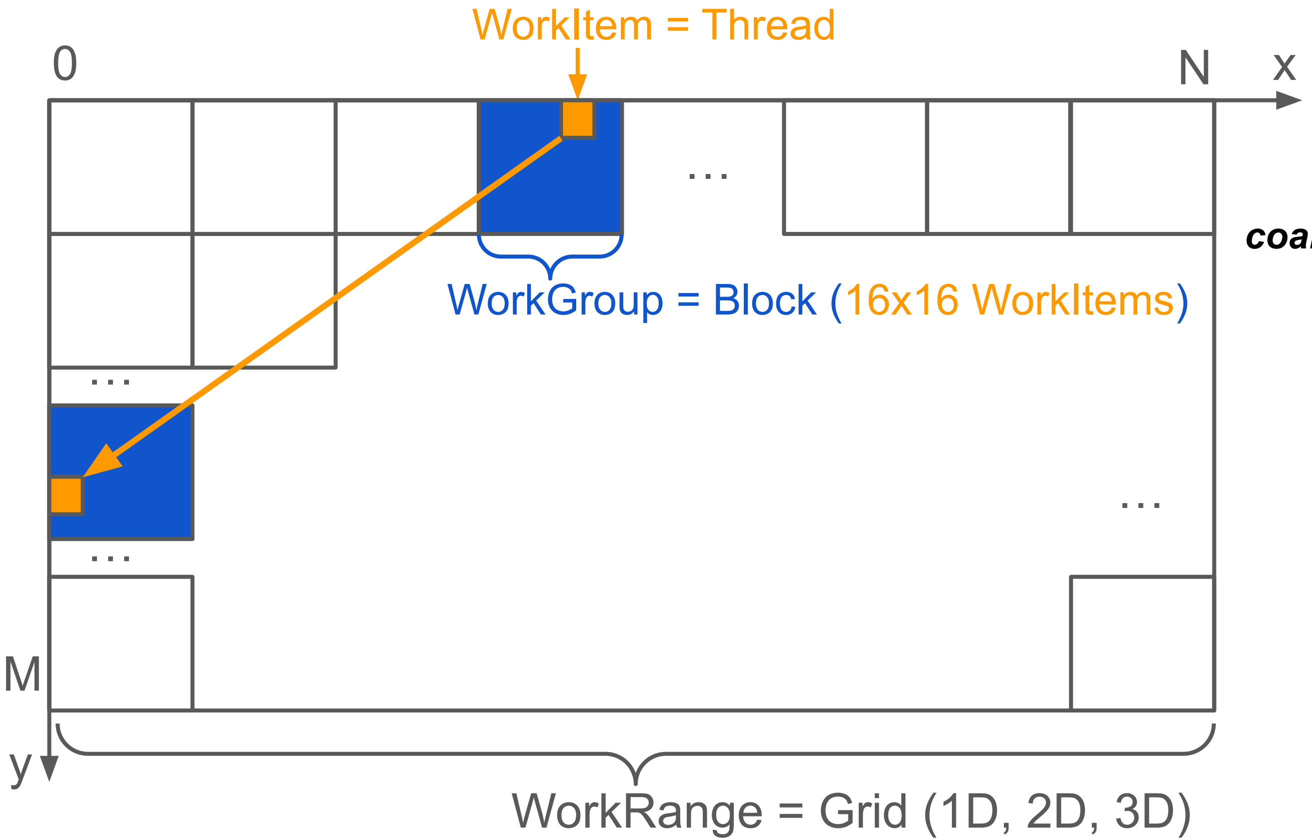
Что делает каждый поток (work item)?

Транспонирование матрицы



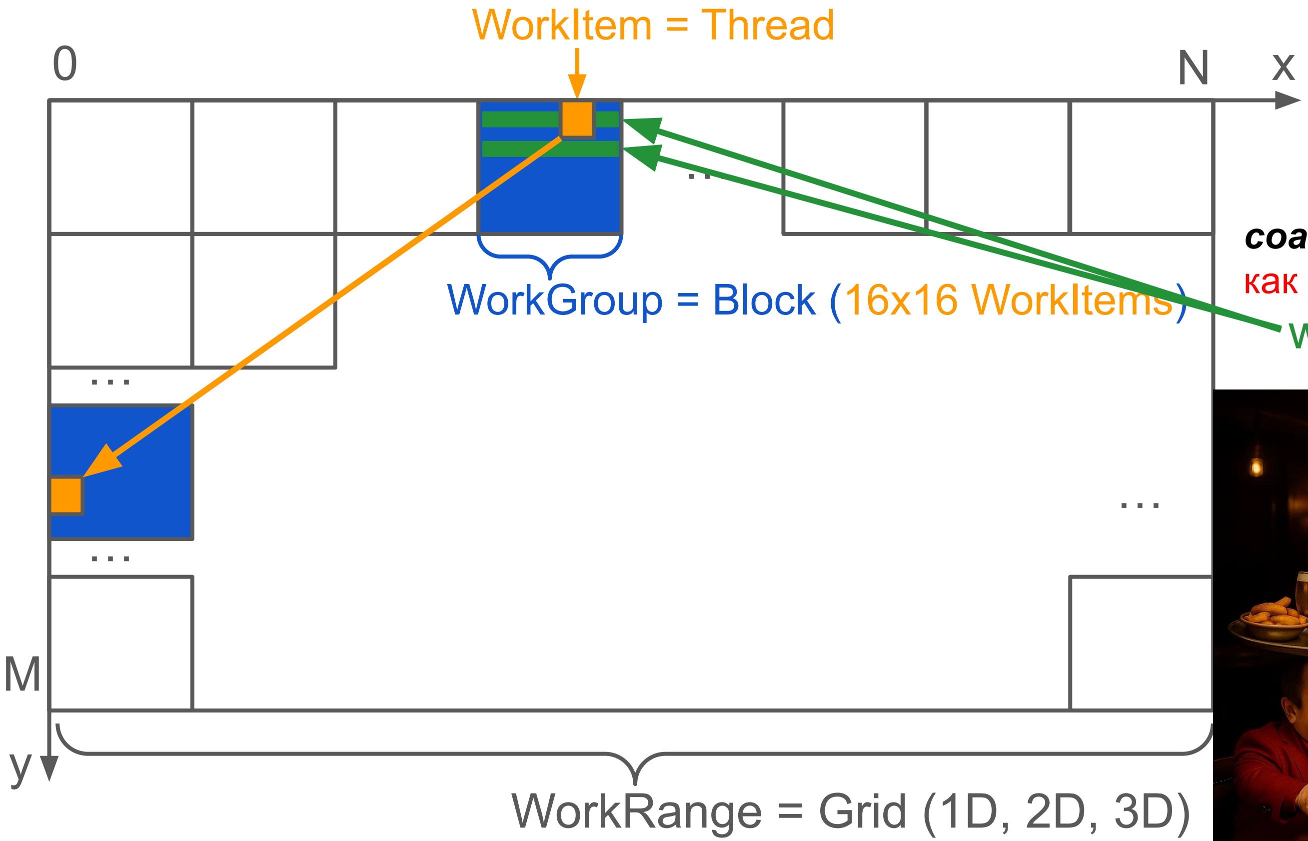
Что делает каждый поток (WorkItem)?

Транспонирование матрицы (*coalesced memory access*)



coalesced ли паттерн чтения?

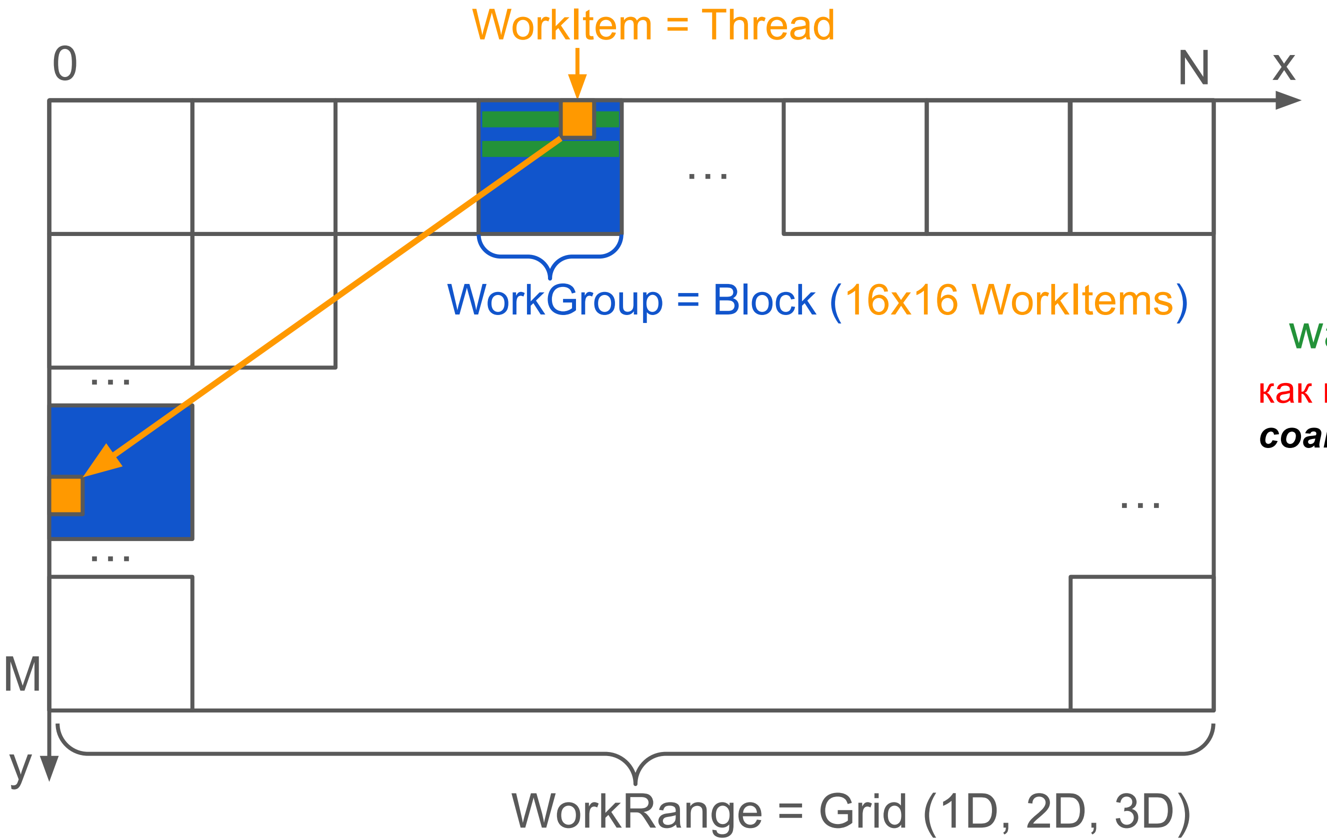
Транспонирование матрицы (*coalesced memory access*)



coalesced ли паттерн чтения?
как выглядит выкладка **warp**?

warp = 32 = 16+16

Транспонирование матрицы (*coalesced memory access*)

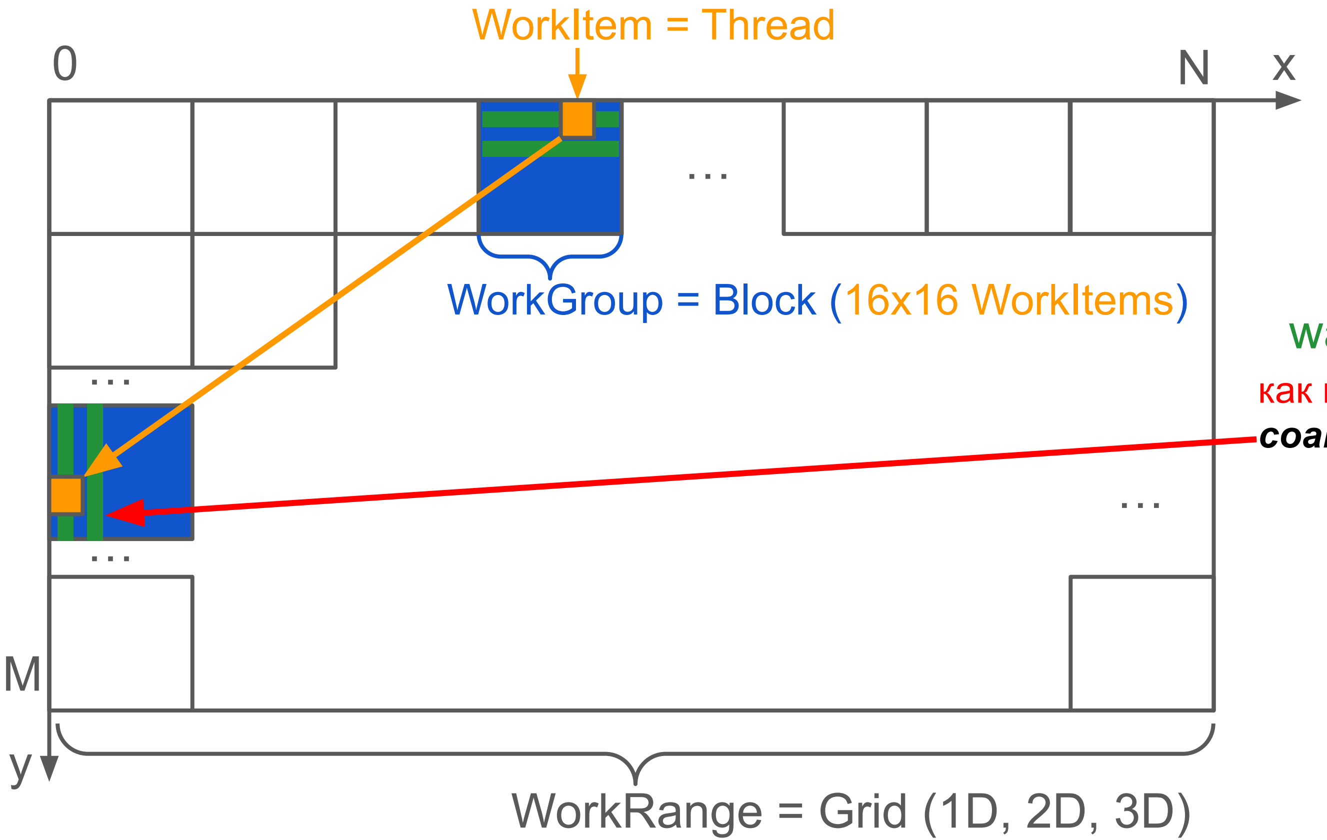


warp = 32 = 16+16

как выглядит паттерн **записи**?

coalesced ли паттерн **записи**?

Транспонирование матрицы (*coalesced memory access*)

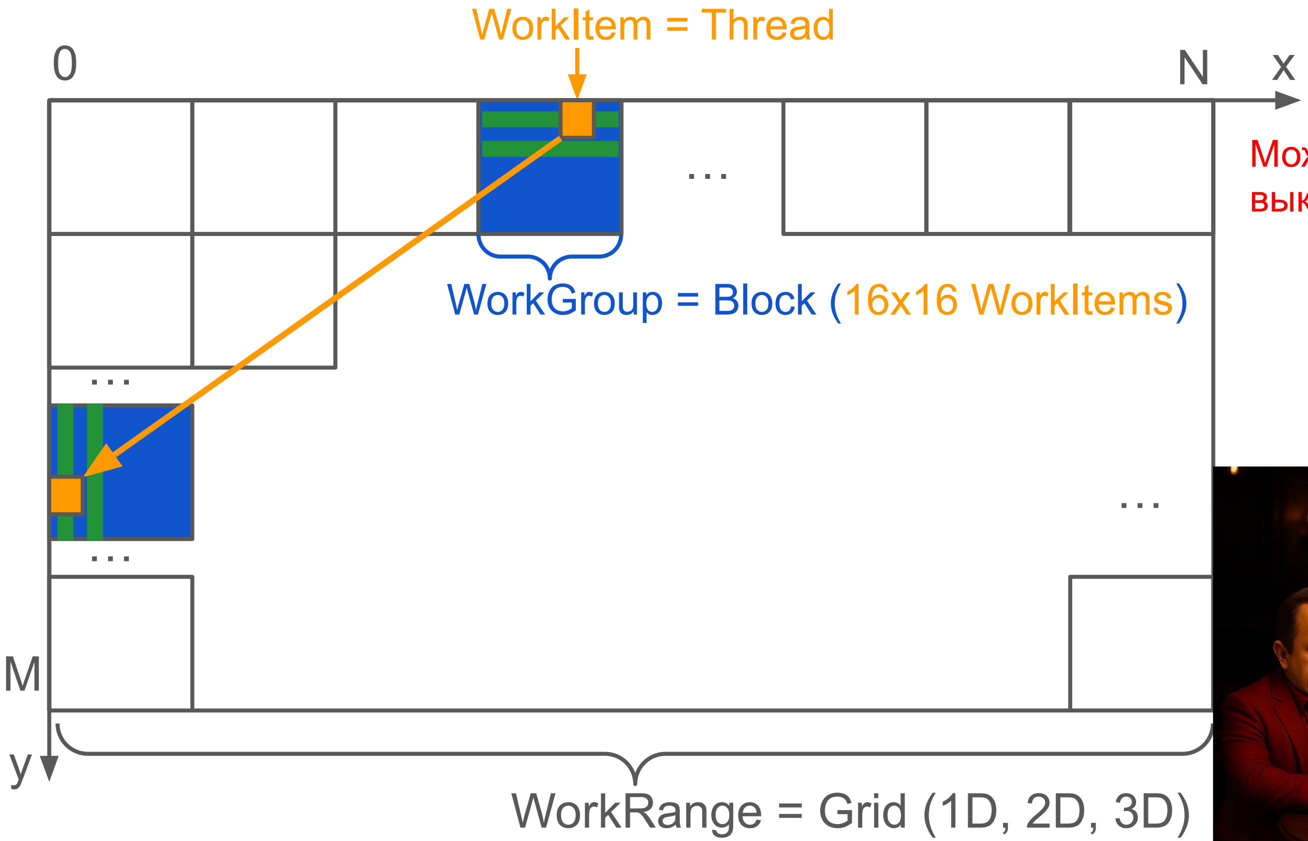


warp = 32 = 16+16

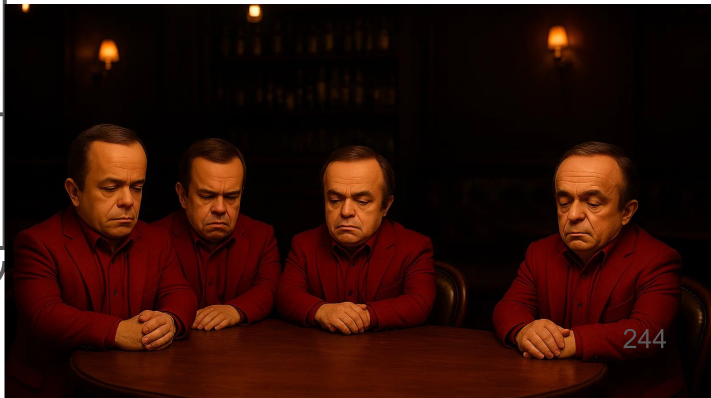
как выглядит паттерн **записи**?

coalesced ли паттерн **записи**?

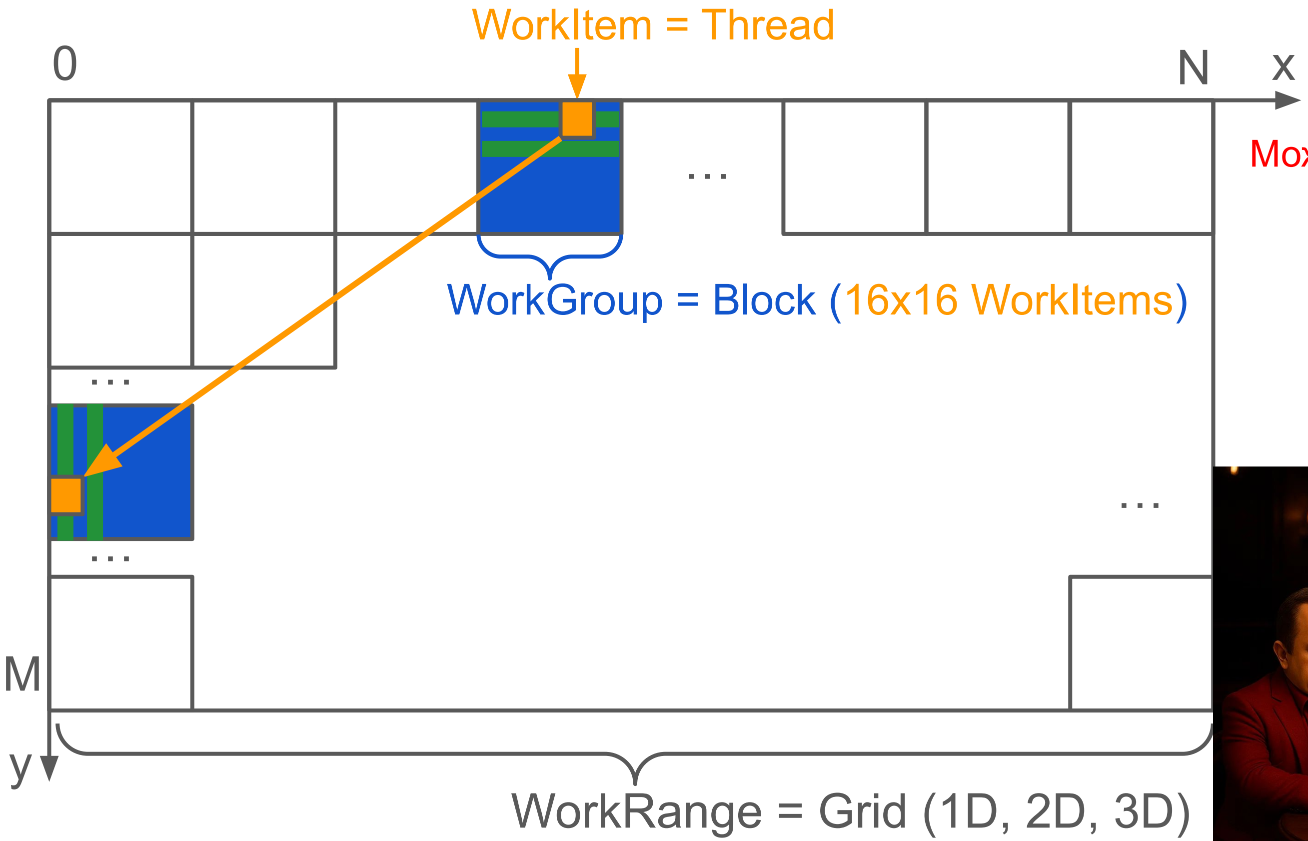
Транспонирование матрицы (*coalesced memory access*)



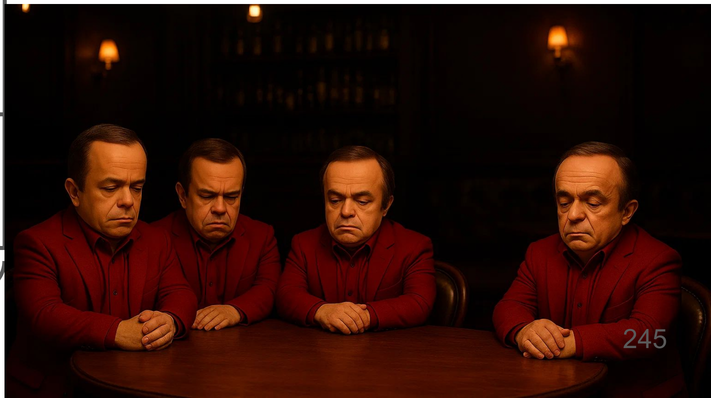
Можем ли мы исправить это поменяв выкладку **warp**-а?



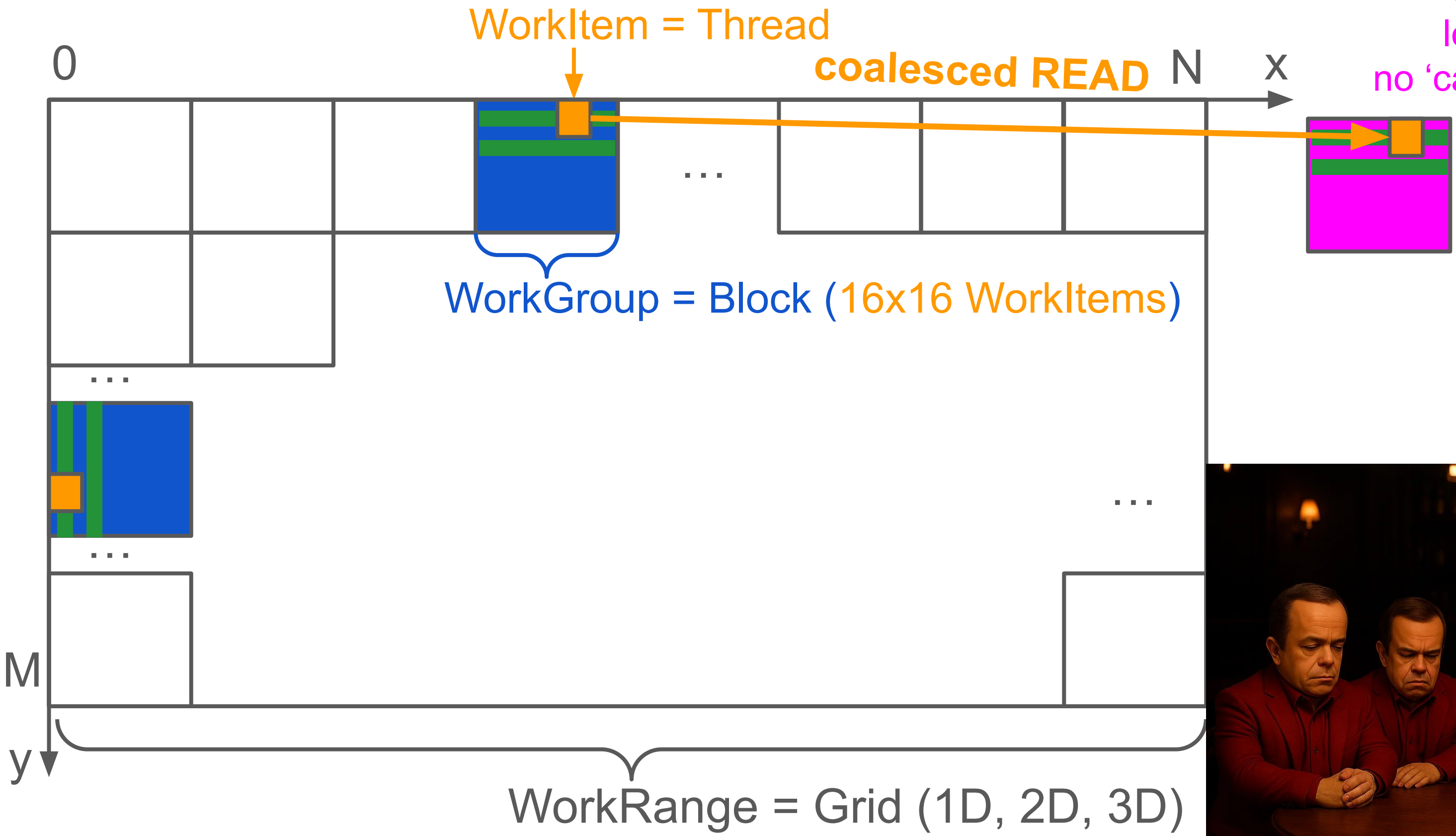
Транспонирование матрицы (*coalesced memory access*)

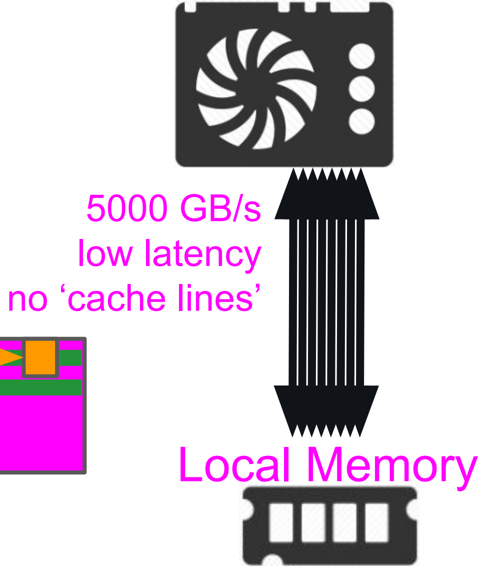


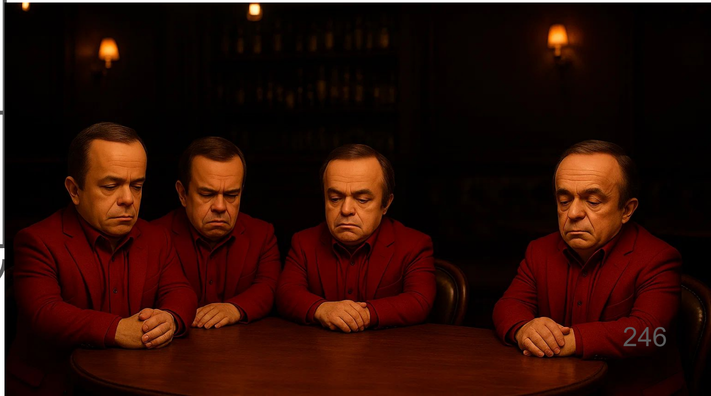
Можем ли мы исправить это?



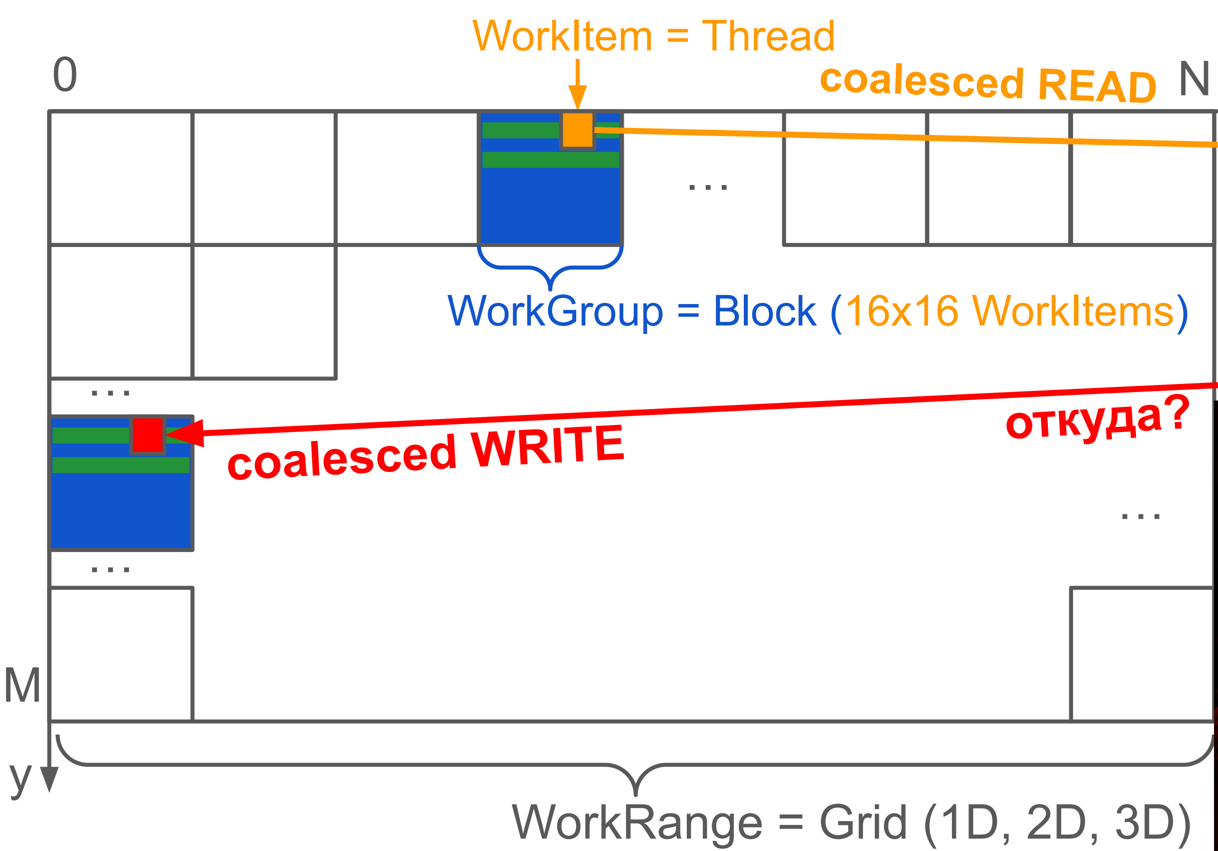
Транспонирование матрицы (*coalesced*)



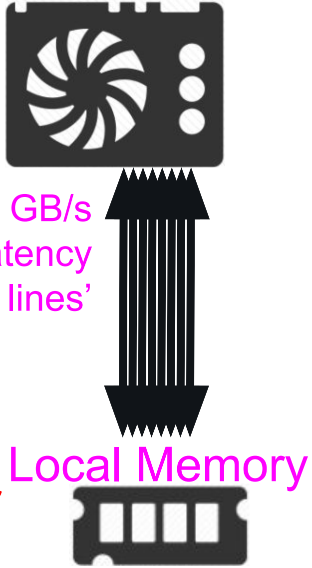


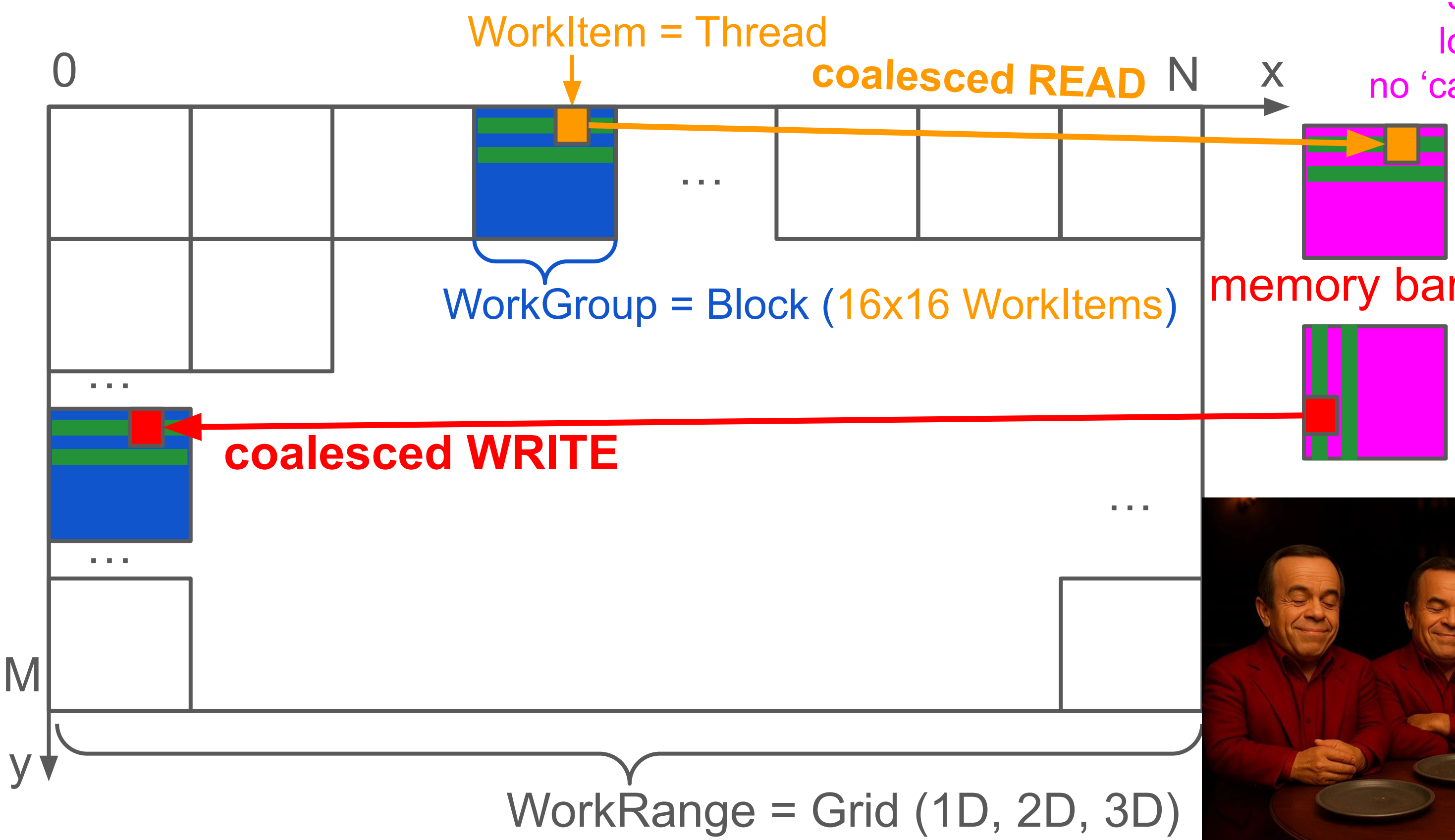


Транспонирование матрицы (*coalesced*)

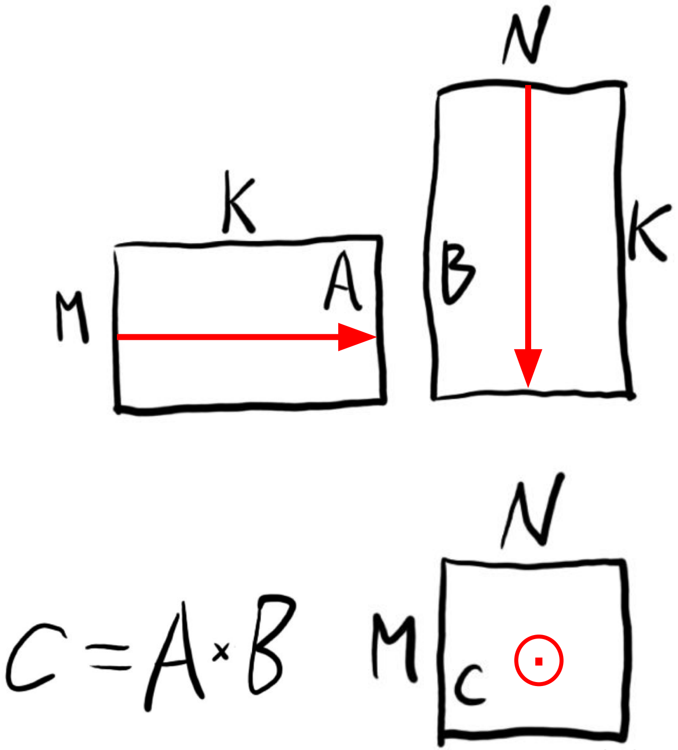


Транспонирование матрицы (*coalesced*)





Умножение матриц

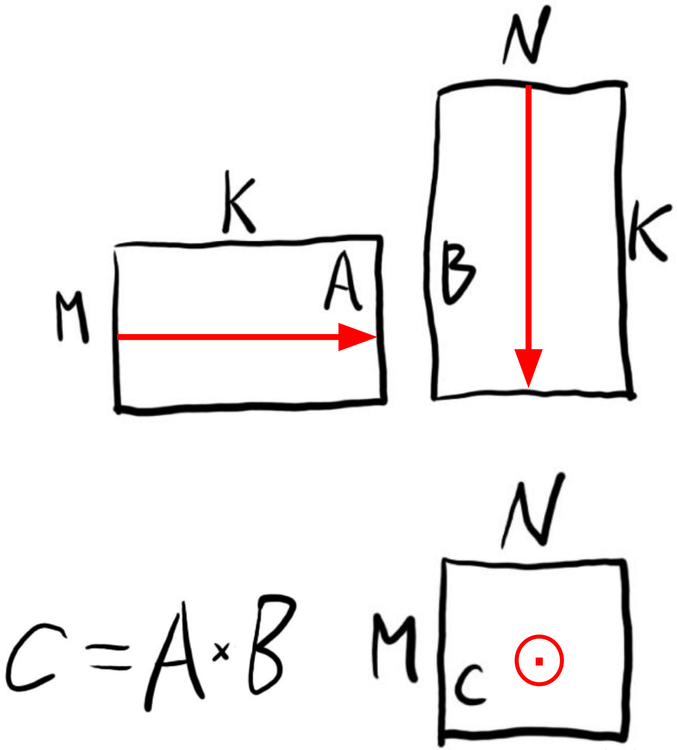


Умножение матриц

Сколько у нас вычислений?

Сколько у нас чтений/записей данных?

Какая пропорция?



Умножение матриц

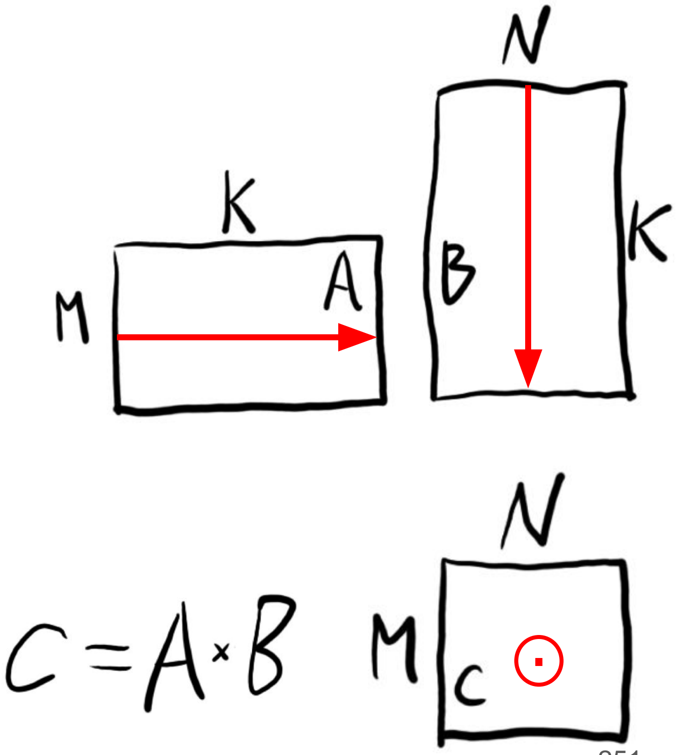
Сколько у нас вычислений?

O(N□M□K)

Сколько у нас чтений/записей данных?

O(N□M□K)

Какая пропорция?



Умножение матриц

Сколько у нас вычислений?

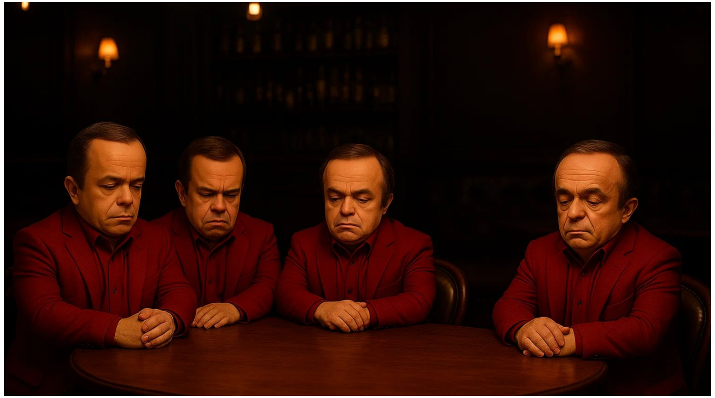
O(N□M□K)

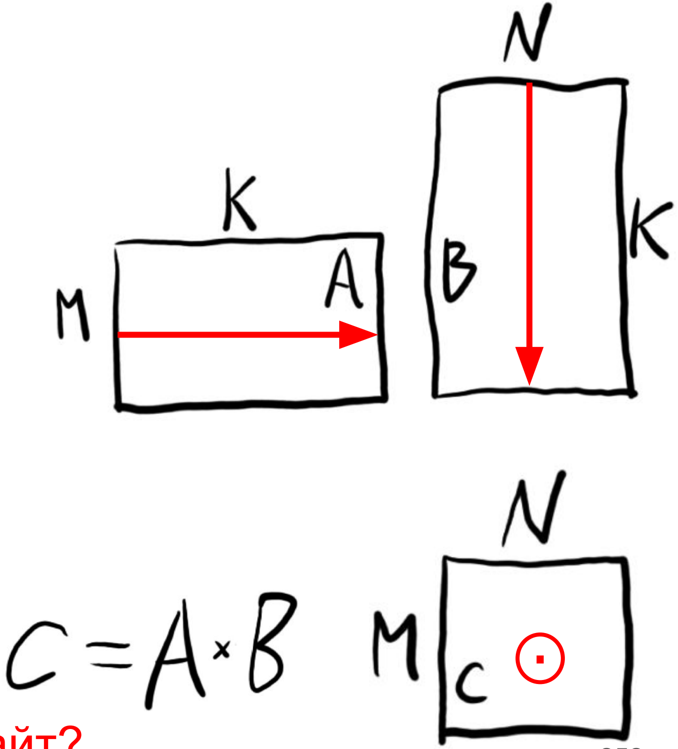
Сколько у нас чтений/записей данных?

O(N□M□K)

Какая пропорция?

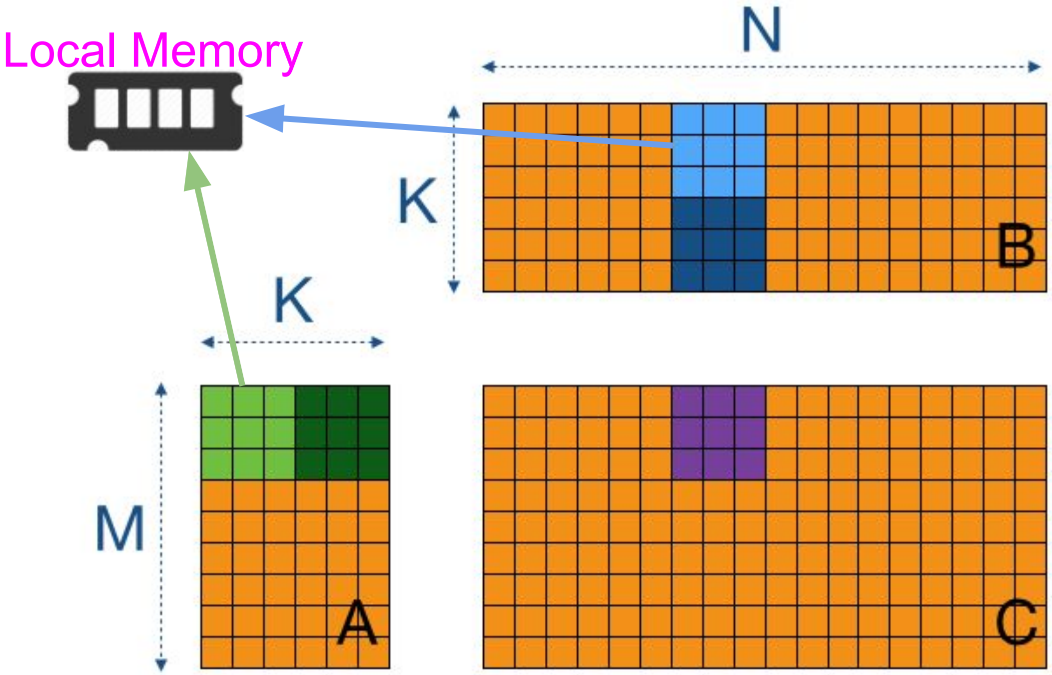
1:1



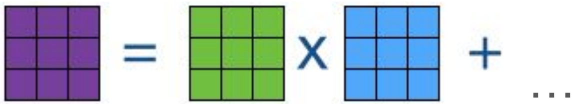


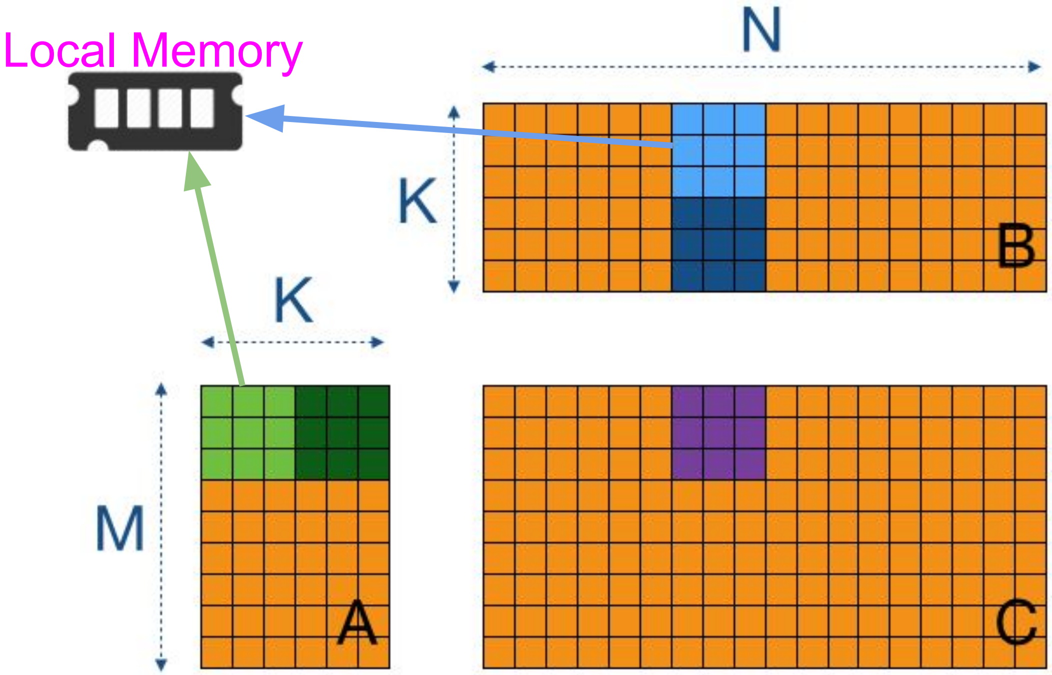
Как увеличить объем вычислений на считанный байт?

Умножение матриц

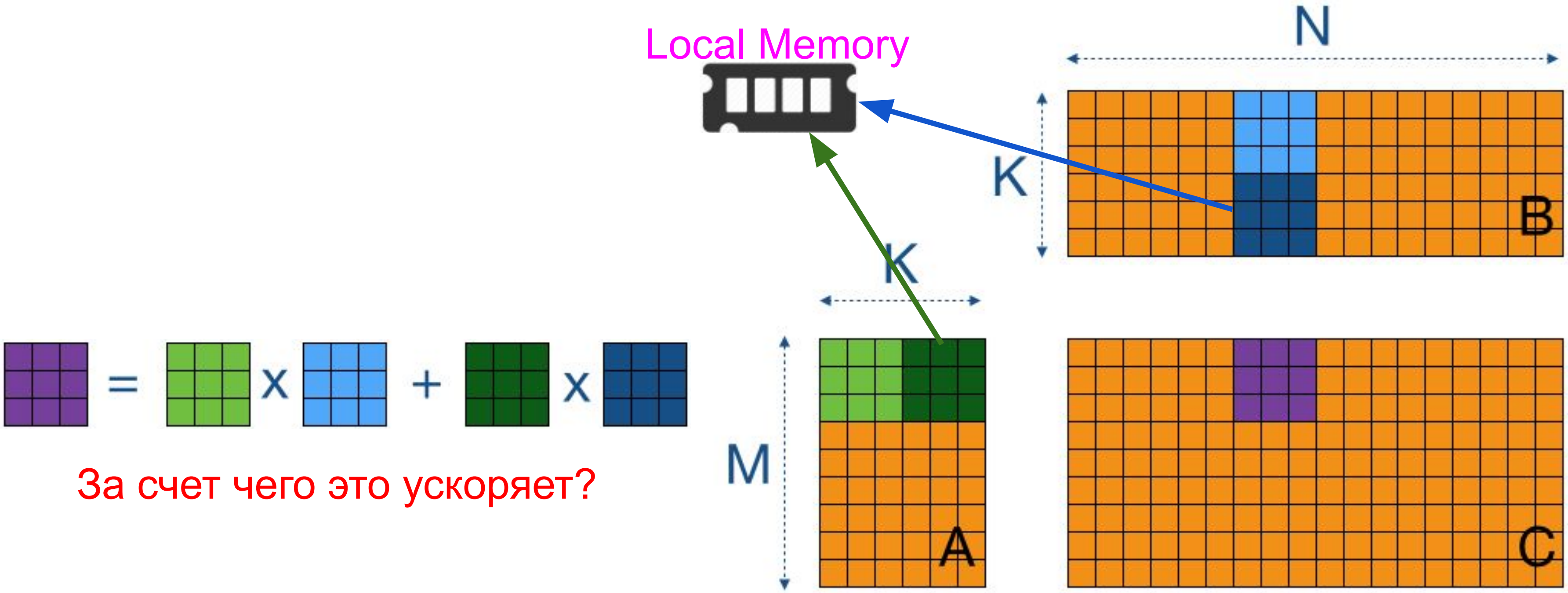


Умножение матриц





Умножение матриц



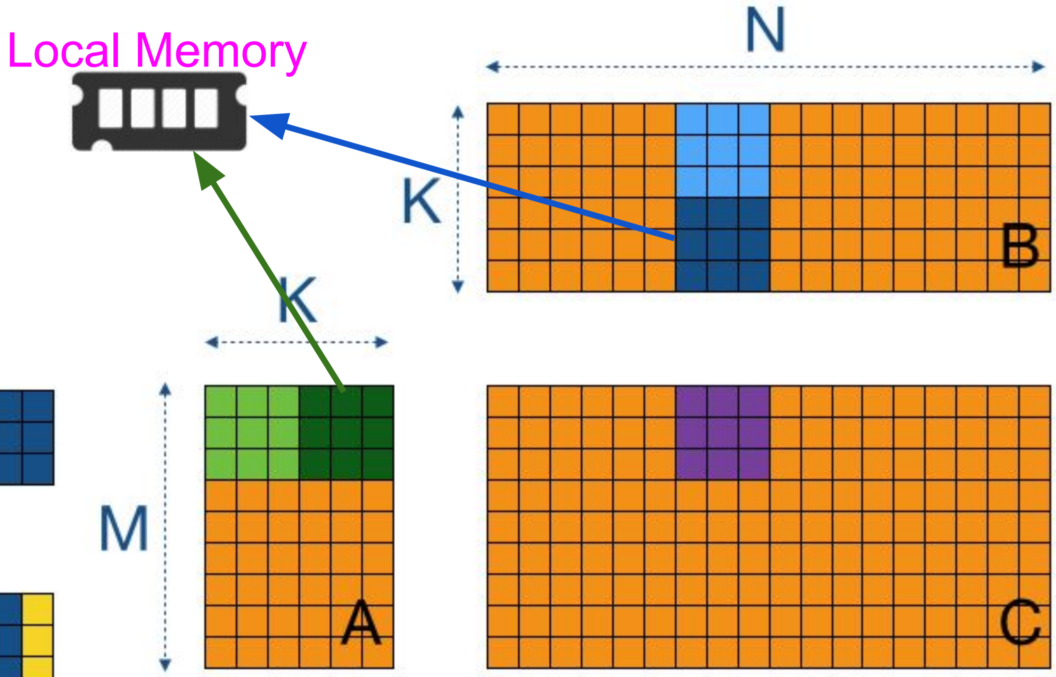
За счет чего это ускоряет?

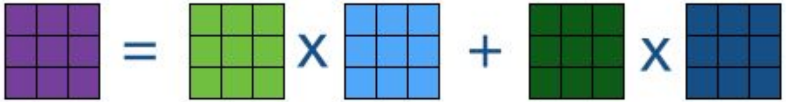
Умножение матриц

Сколько у нас вычислений?

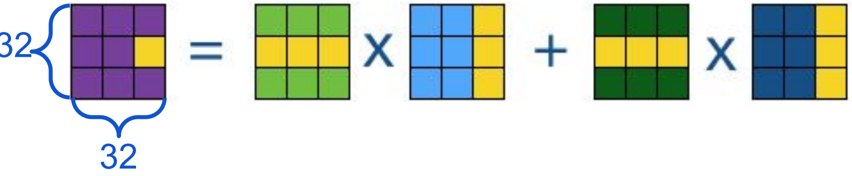
Сколько у нас чтений/записей?

Какая пропорция?





Переиспользование данных:



Умножение матриц

Сколько у нас вычислений?

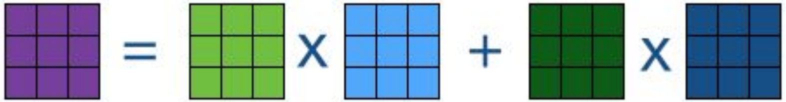
O(N□M□K)

Сколько у нас чтений/записей?

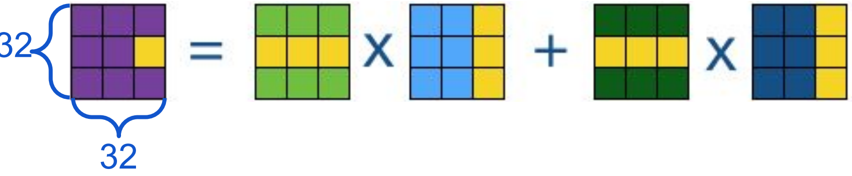
O(N□M□K / **32**)

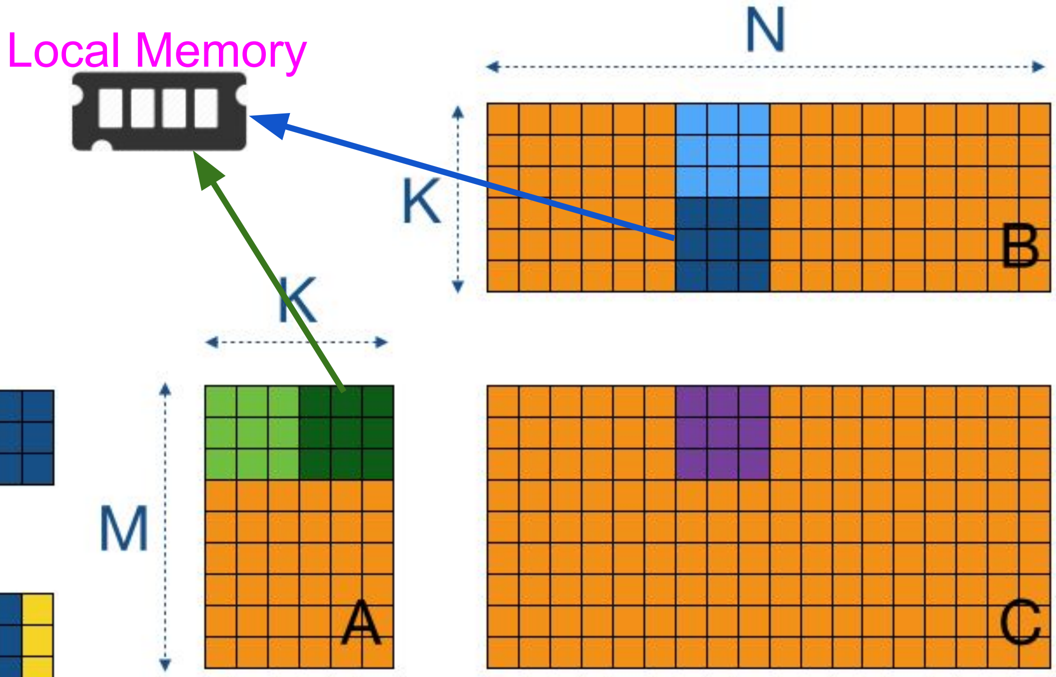
Какая пропорция?

32:1



Переиспользование данных:





Умножение матриц

Неравная битва за гигафлопсы при умножении матриц
(хорошо описанная аналитика, профилирование, оптимизация):

- AMD RDNA3 - https://seb-v.github.io/optimization/update/2025/01/20/Fast-GPU-Matrix-multiplication.html
- NVIDIA Kepler - https://cnugteren.github.io/tutorial/pages/page15.html
- https://siboehm.com/articles/22/CUDA-MMM

Умножение матриц

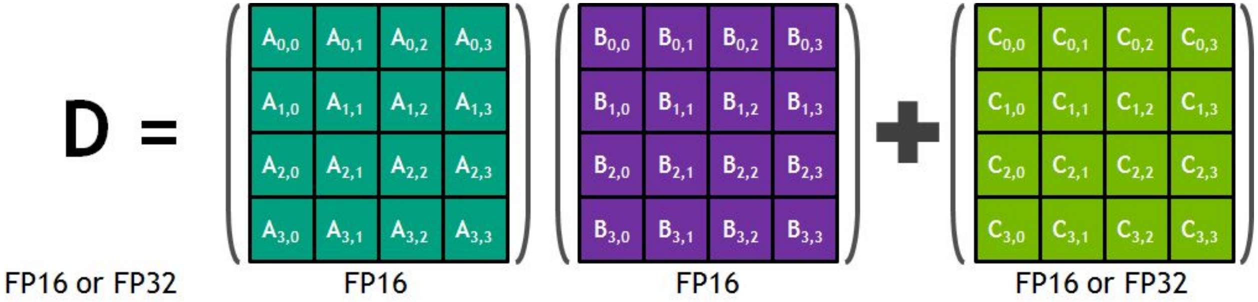
Tensor Cores

	VRAM *TB/s*	**FP 32** *TFlops*	**FP 16** *TFlops*	**FP 16 (tensor)** *TFlops*
RTX 3090	0.93	29	29	142
RTX 4090	1.00	73	73	330
RTX 5090	1.79	105	105	419
Tesla H100	3.35	67	268 (4:1)	990 (16:1)

Умножение матриц

Tensor Cores

4x4



Умножение матриц

Tensor Cores

4x4

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 FP16 FP16 FP16 or FP32

Tensor Cores (CUDA kernel)

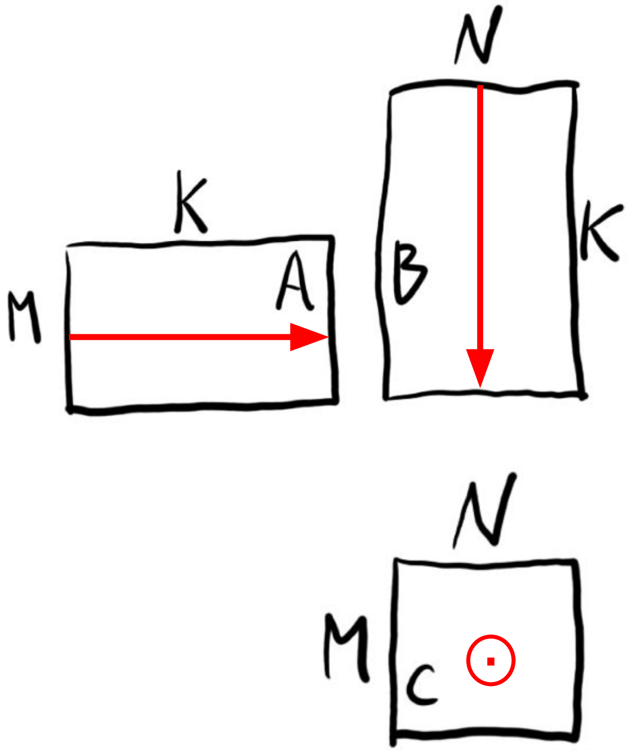
16x16

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,\ldots} & A_{0,15} \\ A_{1,0} & A_{1,1} & A_{1,\ldots} & A_{1,15} \\ A_{\ldots,0} & A_{\ldots,1} & A_{\ldots,\ldots} & A_{\ldots,15} \\ A_{15,0} & A_{15,1} & A_{15,\ldots} & A_{15,15} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,\ldots} & B_{0,15} \\ B_{1,0} & B_{1,1} & B_{1,\ldots} & B_{1,15} \\ B_{\ldots,0} & B_{\ldots,1} & B_{\ldots,\ldots} & B_{\ldots,15} \\ B_{15,0} & B_{15,1} & B_{15,\ldots} & B_{15,15} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,\ldots} & C_{0,15} \\ C_{1,0} & C_{1,1} & C_{1,\ldots} & C_{1,15} \\ C_{\ldots,0} & C_{\ldots,1} & C_{\ldots,\ldots} & C_{\ldots,15} \\ C_{15,0} & C_{15,1} & C_{15,\ldots} & C_{15,15} \end{pmatrix}$$

FP16 or FP32 FP16 FP16 FP16 or FP32

```
// Performs an MxNxK GEMM (C=alpha*A*B + beta*C)

__global__ void wmma_example(half *a, half *b, float *c, int M, int N, int K, float alpha, float beta) {
```

$$C = \alpha A \cdot B + \beta C$$

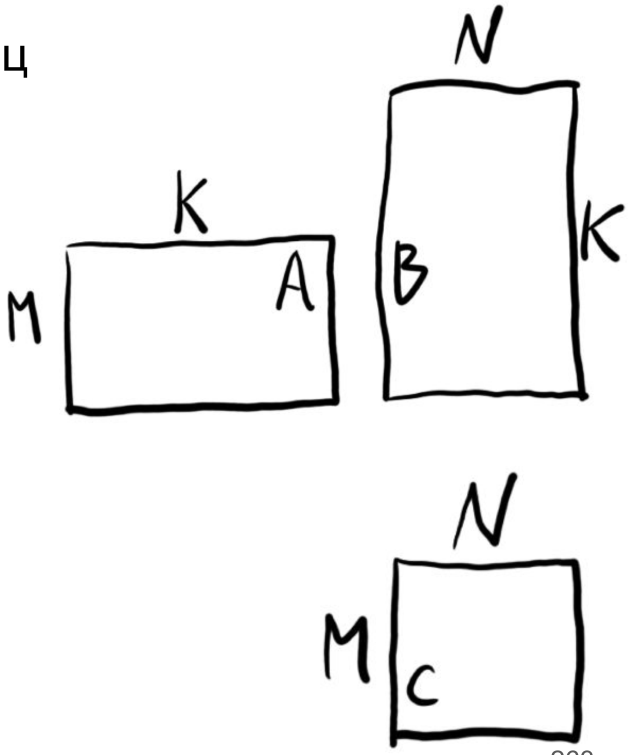


https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

```
// Performs an MxNxK GEMM (C=alpha*A*B + beta*C)
__global__ void wmma_example(half *a, half *b, float *c, int M, int N, int K, float alpha, float beta) {
   // Declare the fragments
   wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> a_frag;
   wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> b_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> acc_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> c_frag;
```

$$C = \alpha A \cdot B + \beta C$$

Общие на весь **warp** 16x16 фрагменты матриц
WMMA = **Warp** Matrix Multiply-Accumulate



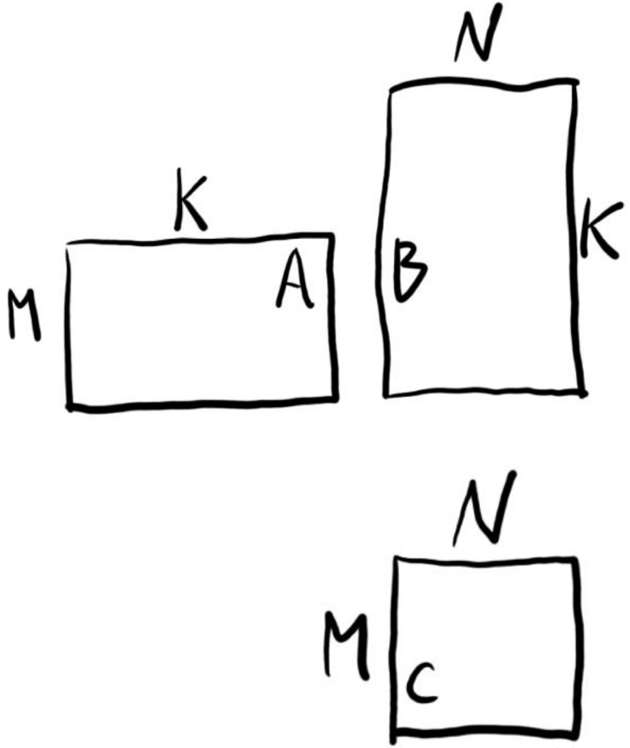
https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

```
// Performs an MxNxK GEMM (C=alpha*A*B + beta*C)

__global__ void wmma_example(half *a, half *b, float *c, int M, int N, int K, float alpha, float beta) {

    // Declare the fragments
    wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> a_frag;
    wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> b_frag;
    wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> acc_frag;
    wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> c_frag;

    wmma::fill_fragment(acc_frag, 0.0f);
```

16x16

acc_frag

$$C = \alpha A \cdot B + \beta C$$



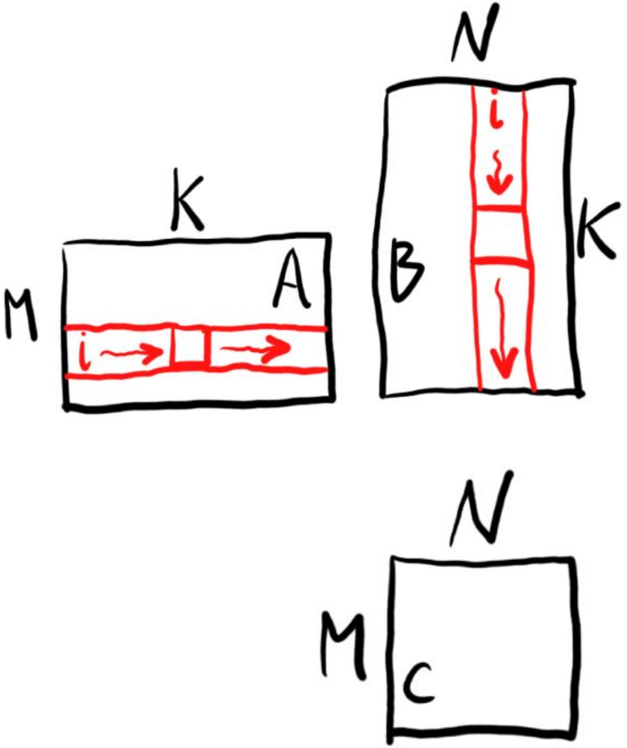
https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

```
// Performs an MxNxK GEMM (C=alpha*A*B + beta*C)
__global__ void wmma_example(half *a, half *b, float *c, int M, int N, int K, float alpha, float beta) {
   // Declare the fragments
   wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> a_frag;
   wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> b_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> acc_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> c_frag;

   wmma::fill_fragment(acc_frag, 0.0f);

   for (int i = 0; i < K; i += WMMA_K) {
      int aRow = warpM * WMMA_M;
      int aCol = i;

      int bRow = i;
      int bCol = warpN * WMMA_N;

      // Bounds checking
      if (aRow < M && aCol < K && bRow < K && bCol < N) {
         // Load the inputs
         wmma::load_matrix_sync(a_frag, a + aRow + aCol * lda, lda);
         wmma::load_matrix_sync(b_frag, b + bRow + bCol * ldb, ldb);
```

16x16

acc_frag

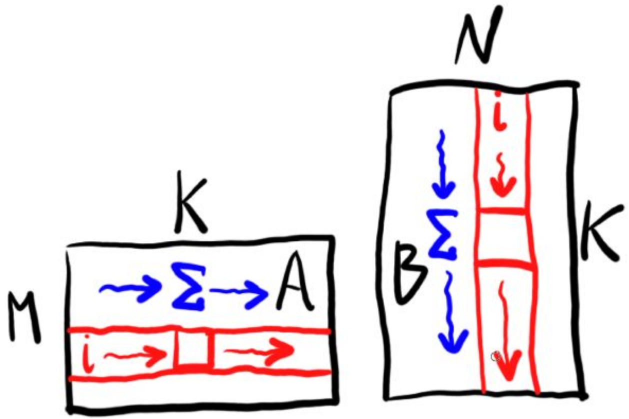
$$C = \alpha A \cdot B + \beta C$$

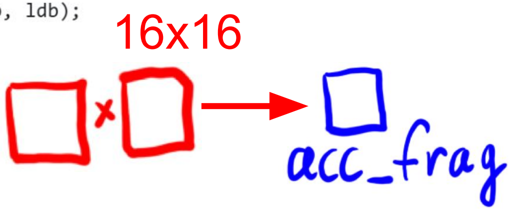


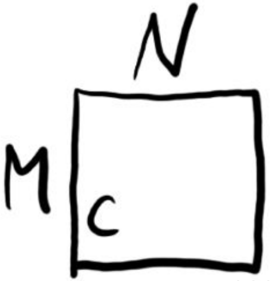
https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

```
// Performs an MxNxK GEMM (C=alpha*A*B + beta*C)

__global__ void wmma_example(half *a, half *b, float *c, int M, int N, int K, float alpha, float beta) {

   // Declare the fragments
   wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> a_frag;
   wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> b_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> acc_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> c_frag;

   wmma::fill_fragment(acc_frag, 0.0f);

   for (int i = 0; i < K; i += WMMA_K) {
      int aRow = warpM * WMMA_M;
      int aCol = i;

      int bRow = i;
      int bCol = warpN * WMMA_N;

      // Bounds checking
      if (aRow < M && aCol < K && bRow < K && bCol < N) {
         // Load the inputs
         wmma::load_matrix_sync(a_frag, a + aRow + aCol * lda, lda);
         wmma::load_matrix_sync(b_frag, b + bRow + bCol * ldb, ldb);

         // Perform the matrix multiplication
         wmma::mma_sync(acc_frag, a_frag, b_frag, acc_frag);

      }
   }
```

$$C = \alpha A \cdot B + \beta C$$



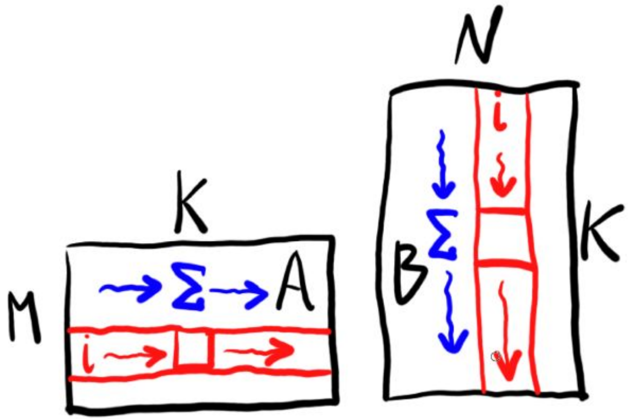


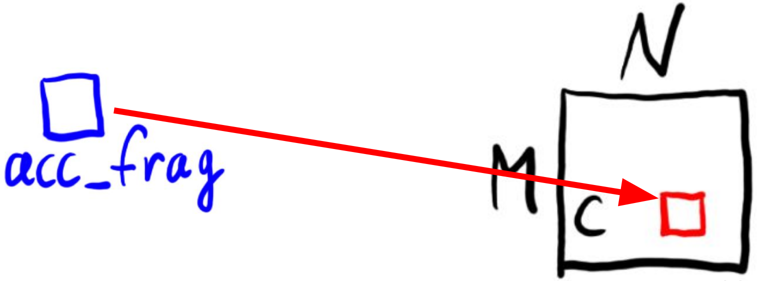


https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

```
// Performs an MxNxK GEMM (C=alpha*A*B + beta*C)
__global__ void wmma_example(half *a, half *b, float *c, int M, int N, int K, float alpha, float beta) {
   // Declare the fragments
   wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> a_frag;
   wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> b_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> acc_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> c_frag;
   .........
```

$$C = \alpha A \cdot B + \beta C$$

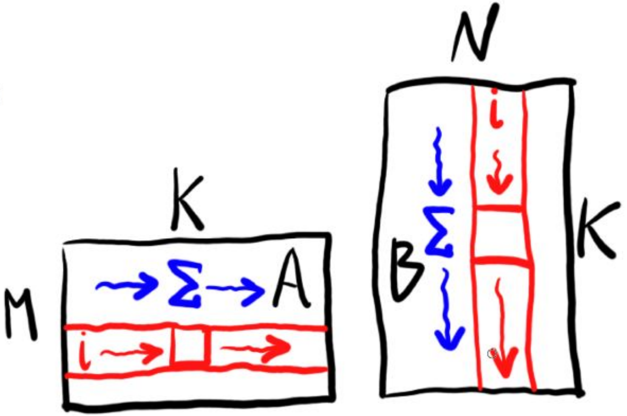


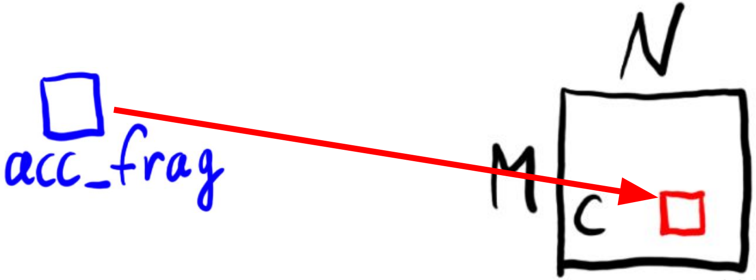


https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

```
// Performs an MxNxK GEMM (C=alpha*A*B + beta*C)
__global__ void wmma_example(half *a, half *b, float *c, int M, int N, int K, float alpha, float beta) {
   // Declare the fragments
   wmma::fragment<wmma::matrix_a, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> a_frag;
   wmma::fragment<wmma::matrix_b, WMMA_M, WMMA_N, WMMA_K, half, wmma::col_major> b_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> acc_frag;
   wmma::fragment<wmma::accumulator, WMMA_M, WMMA_N, WMMA_K, float> c_frag;

   .........

    // Load in the current value of c, scale it by beta, and add this our result scaled
    int cRow = warpM * WMMA_M;
    int cCol = warpN * WMMA_N;

    if (cRow < M && cCol < N) {
       wmma::load_matrix_sync(c_frag, c + cRow + cCol * ldc, ldc, wmma::mem_col_major);

 #pragma unroll
       for(int i=0; i < c_frag.num_elements; i++) {
          c_frag.x[i] = alpha * acc_frag.x[i] + beta * c_frag.x[i];
       }

       // Store the output
       wmma::store_matrix_sync(c + cRow + cCol * ldc, c_frag, ldc, wmma::m
    }
 }
```

$$C = \boxed{\alpha A \cdot B + \beta C}$$





https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

Умножение матриц

Tensor Cores

https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/

https://github.com/NVIDIA-developer-blog/code-samples/blob/master/posts/tensor-cores/simpleTensorCoreGEMM.cu

https://github.com/NVIDIA/cutlass/blob/main/examples/00_basic_gemm/basic_gemm.cu

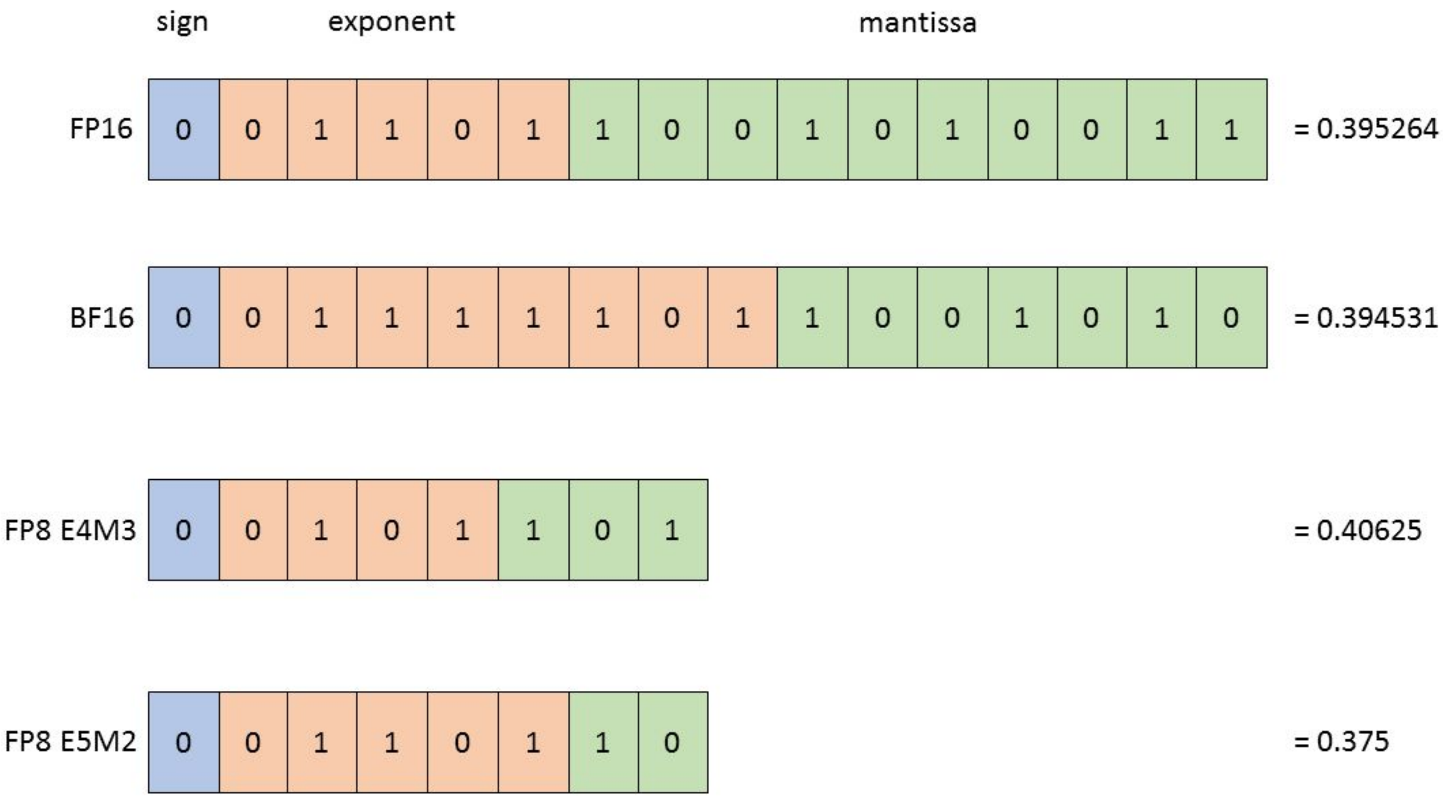
https://developer.nvidia.com/blog/cutlass-linear-algebra-cuda

https://docs.nvidia.com/cuda/cuda-c-programming-guide/

Неравная битва за гигафлопсы при умножении матриц (хорошо описанная аналитика, профилирование, оптимизация):

- AMD RDNA3 - https://seb-v.github.io/optimization/update/2025/01/20/Fast-GPU-Matrix-multiplication.html
- NVIDIA Kepler - https://cnugteren.github.io/tutorial/pages/page15.html
- https://siboehm.com/articles/22/CUDA-MMM

DeepSeek: x2 ускорение обучения (fp16 → fp8)



https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/fp8_primer.html

DeepSeek: x2 ускорение обучения (fp16 → fp8)

NVIDIA H100 (почти то же что и H800*):

- Memory bandwidth: **3.35 TB/sec**
- FP 32: **67 TFlops**
- FP 16: **268 TFlops**
- FP 32 (tensor): **495 TFlops**
- FP 16 (tensor): **990 TFlops**
- FP 8 (tensor): **1979 TFlops**

*H800 - почти H100, но соответствует регуляциям экспорта из США в Китай (400 GBs NVlink вместо 600 GB/s + 10% медленнее + нет FP64)

DeepSeek: x2 ускорение обучения (fp16 → fp8)

NVIDIA H100 (почти то же что и H800*):

- Memory bandwidth: **3.35 TB/sec**
- FP 32: **67 TFlops**
- FP 16: **268 TFlops**
- FP 32 (tensor): **495 TFlops**
- FP 16 (tensor): **990 TFlops**
- FP 8 (tensor): **1979 TFlops**

x2

*H800 - почти H100, но соответствует регуляциям экспорта из США в Китай (400 GBs NVlink вместо 600 GB/s + 10% медленнее + нет FP64)

DeepSeek: x2 ускорение обучения (fp16 → fp8)

NVIDIA H100 (почти то же что и H800*):

- Memory bandwidth: **3.35 TB/sec**
- FP 32: **67 TFlops**
- FP 16: **268 TFlops**
- FP 32 (tensor): **495 TFlops**
- FP 16 (tensor): **990 TFlops**
- FP 8 (tensor): **1979 TFlops**

x2

Хватит ли пропускной способности памяти чтобы насытить ALU (tensor cores)?

*H800 - почти H100, но соответствует регуляциям экспорта из США в Китай (400 GBs NVlink вместо 600 GB/s + 10% медленнее + нет FP64)

DeepSeek: x2 ускорение обучения (fp16 → fp8)

NVIDIA H100 (почти то же что и H800*):

- Memory bandwidth: **3.35 TB/sec**
- FP 32: **67 TFlops**
- FP 16: **268 TFlops**
- FP 32 (tensor): **495 TFlops**
- FP 16 (tensor): $\boxed{\textbf{990 TFlops}}$
- FP 8 (tensor): $\boxed{\textbf{1979 TFlops}}$

x2

Какие риски?
Почему так не делают все?

*H800 - почти H100, но соответствует регуляциям экспорта из США в Китай (400 GBs NVlink вместо 600 GB/s + 10% медленнее + нет FP64)

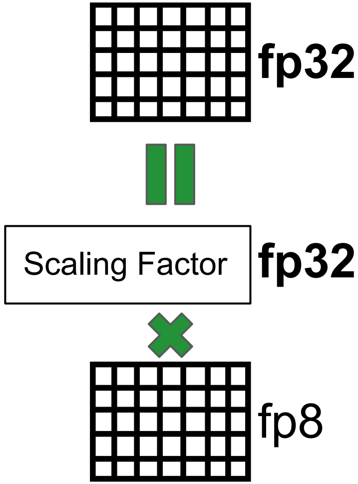
DeepSeek: fine-grained fp8 quantization

NVIDIA H100 (почти то же что и H800*):

- Memory bandwidth: **3.35 TB/sec**
- FP 32: **67 TFlops**
- FP 16: **268 TFlops**
- FP 32 (tensor): **495 TFlops**
- FP 16 (tensor): **990 TFlops**
- FP 8 (tensor): **1979 TFlops**
- DeepGEMM достиг **1550 TFlops** на H800!

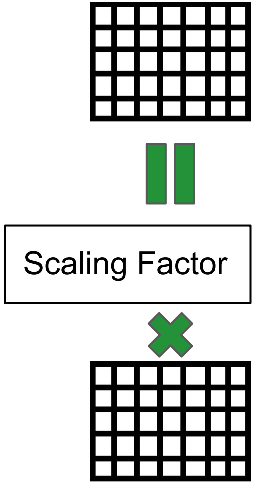
DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437
Репозиторий - https://github.com/deepseek-ai/DeepGEMM (GEMM - General Matrix Multiplications)

DeepSeek: fine-grained fp8 quantization



DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization



fp32 Input Values

fp32 absmax(Input Values) / 448

fp8 Input Values / Scaling Factor

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization

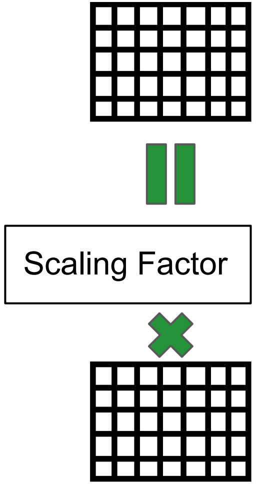
fp32 Input Values

Scaling Factor **fp32** absmax(Input Values) / $\boxed{448}$ Что это за волшебное число? 🦄

fp8 Input Values / Scaling Factor

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization



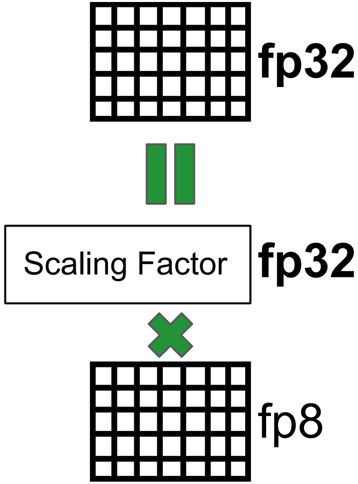
fp32 Input Values

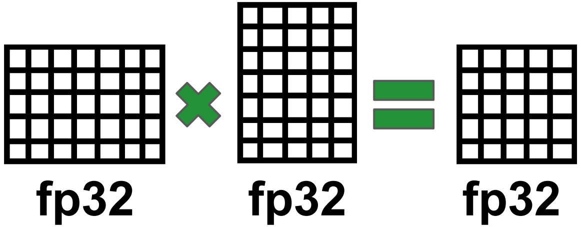
fp32 absmax(Input Values) / $\boxed{448}$ = FP8_MAX

fp8 $\boxed{\text{Input Values / Scaling Factor}}$ ∈ [-448; +448]

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

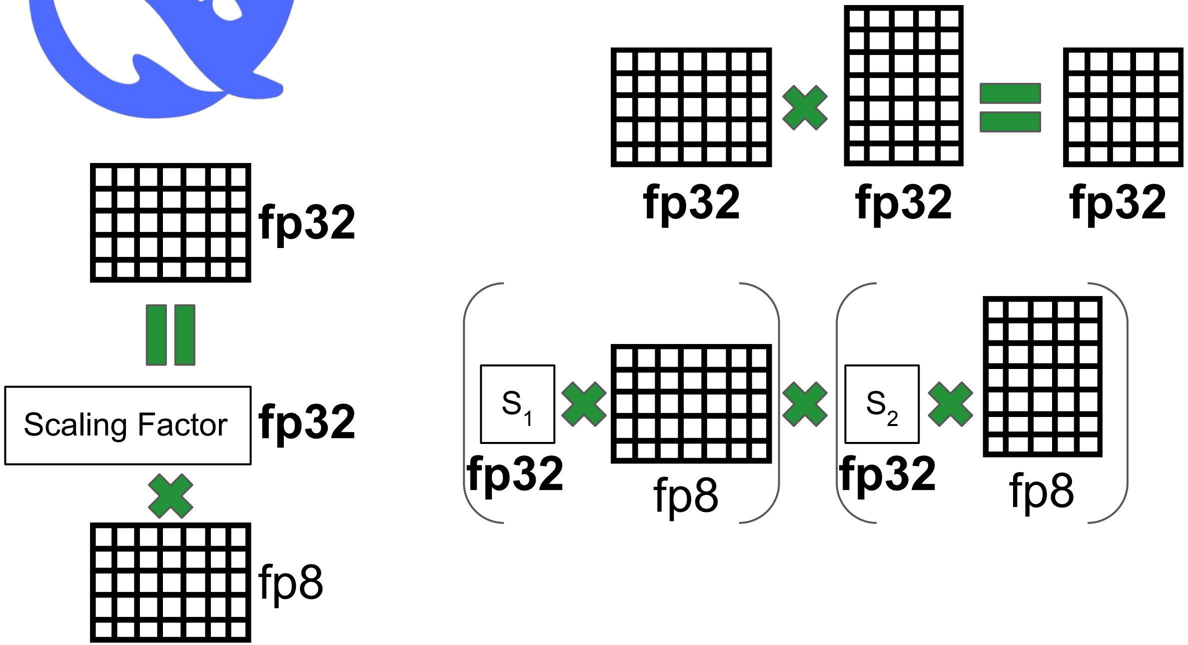
DeepSeek: fine-grained fp8 quantization





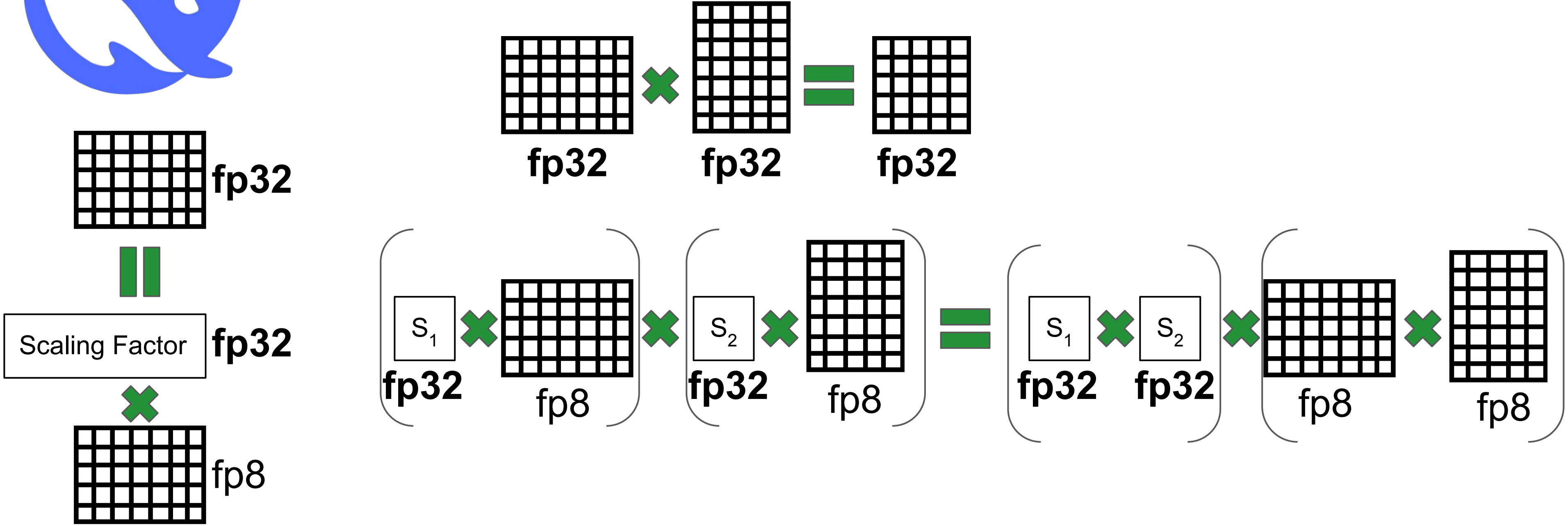
DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization



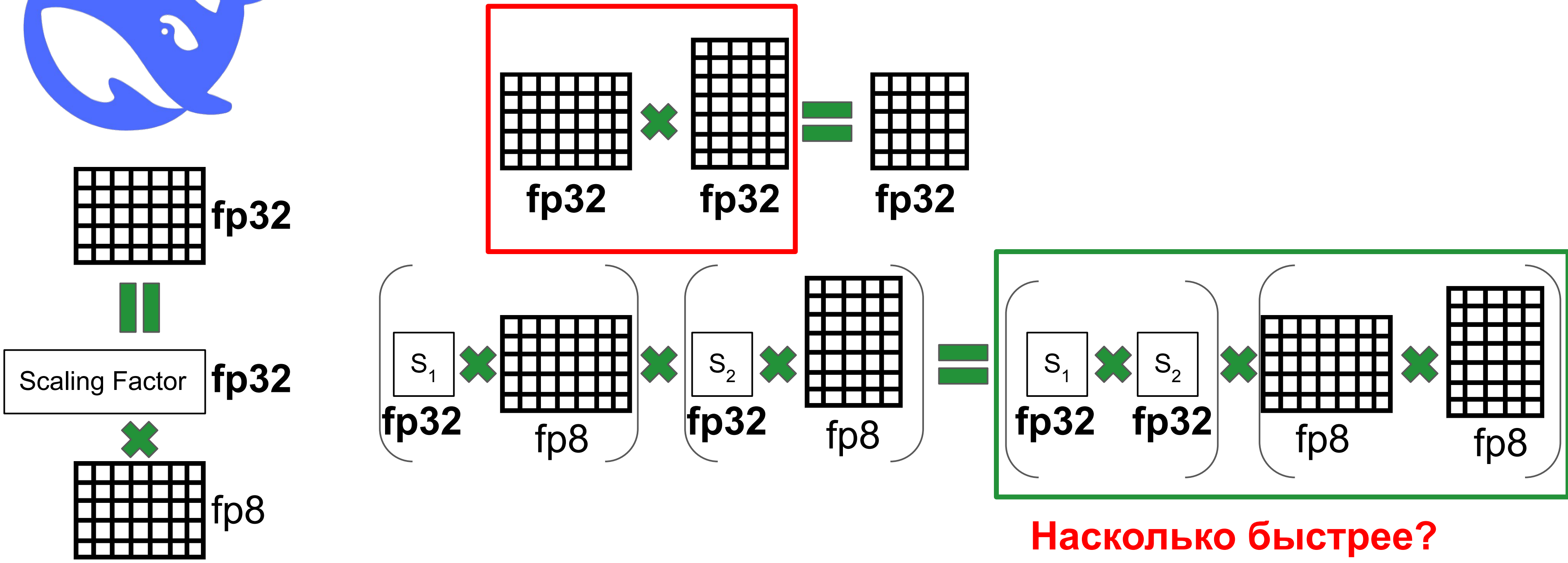
DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization



DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

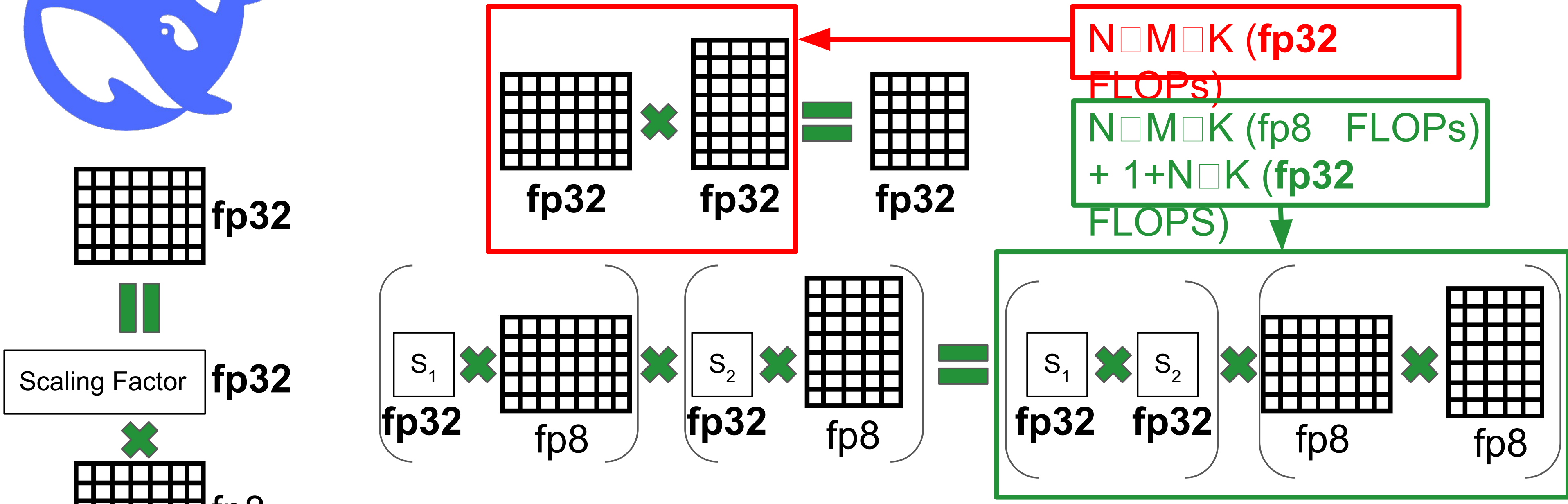
DeepSeek: fine-grained fp8 quantization



Насколько быстрее?

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

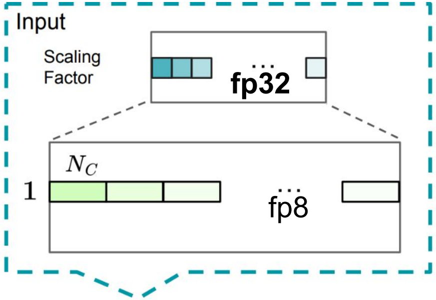
DeepSeek: fine-grained fp8 quantization

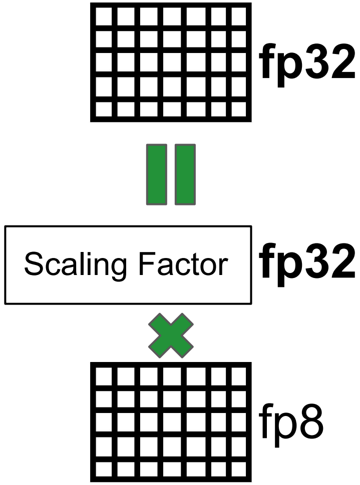


Насколько быстрее?

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

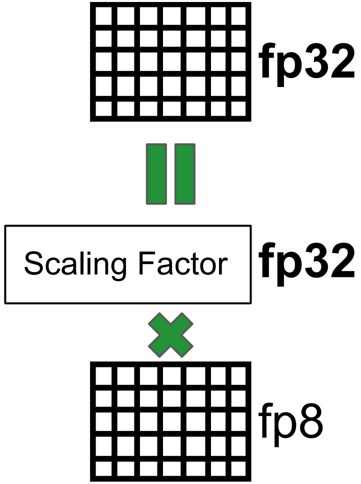
DeepSeek: fine-grained fp8 quantization

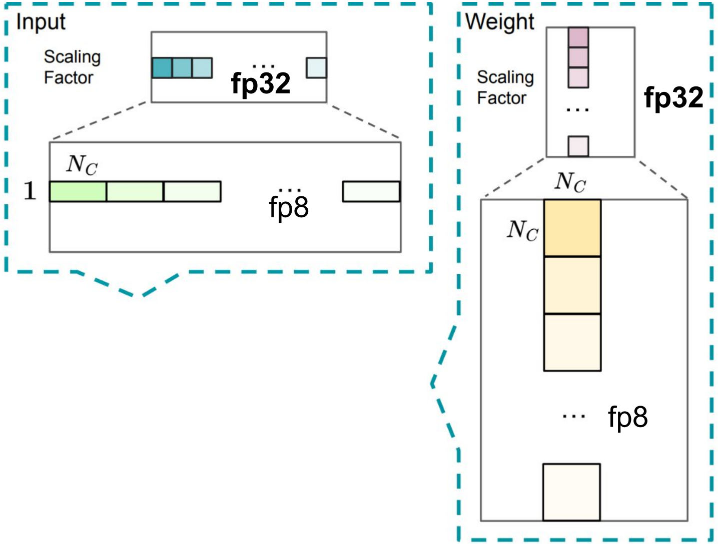




DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

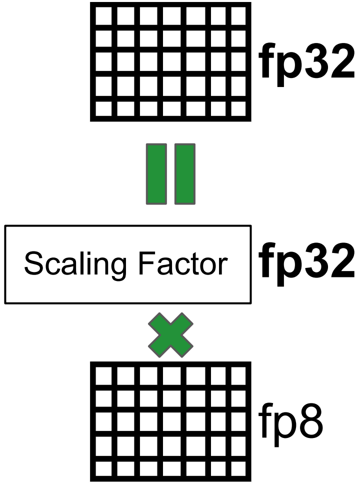
DeepSeek: fine-grained fp8 quantization

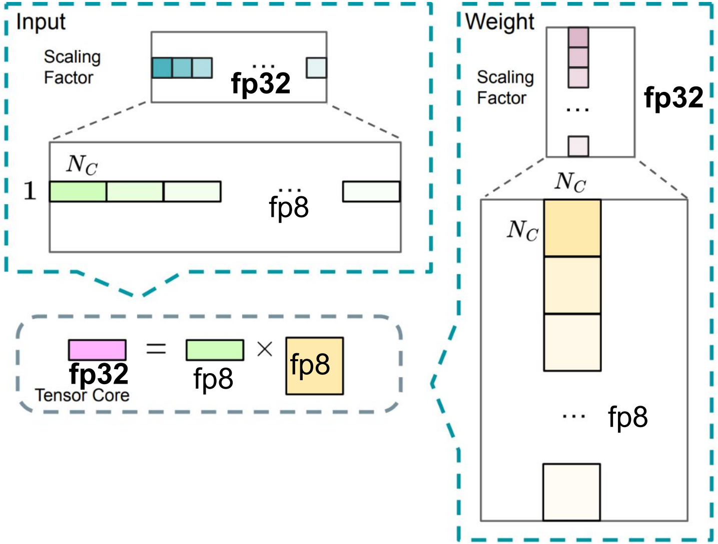




DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

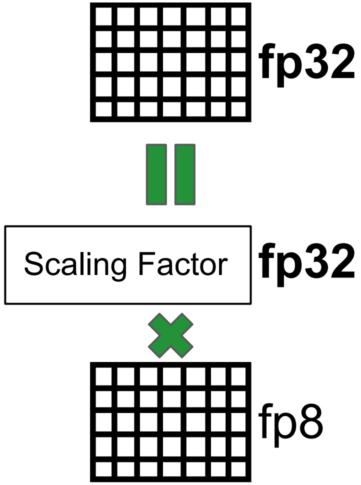
DeepSeek: fine-grained fp8 quantization

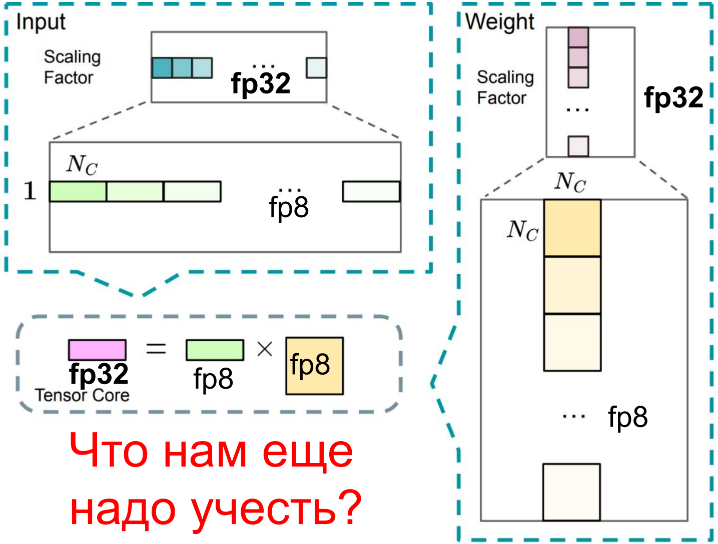




DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization

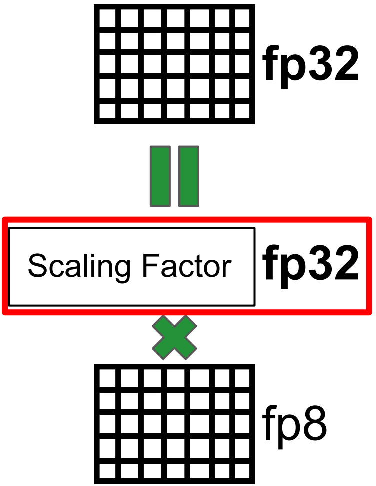


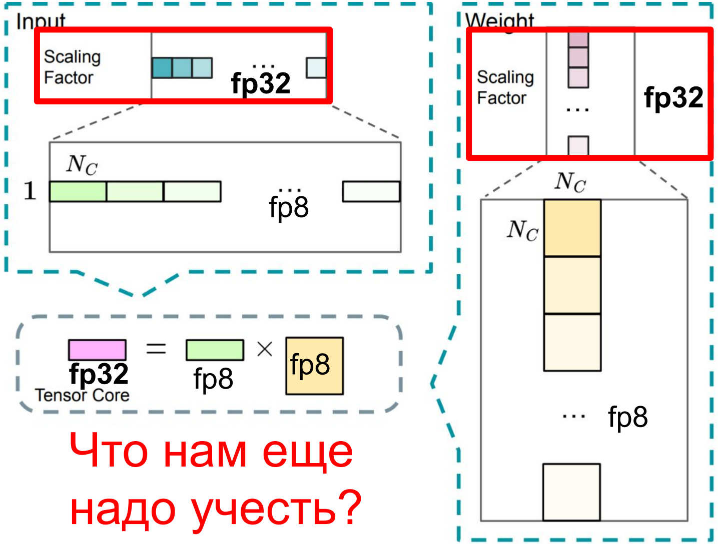


Что нам еще надо учесть?

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization

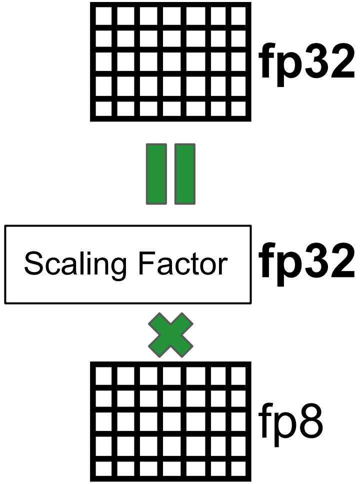


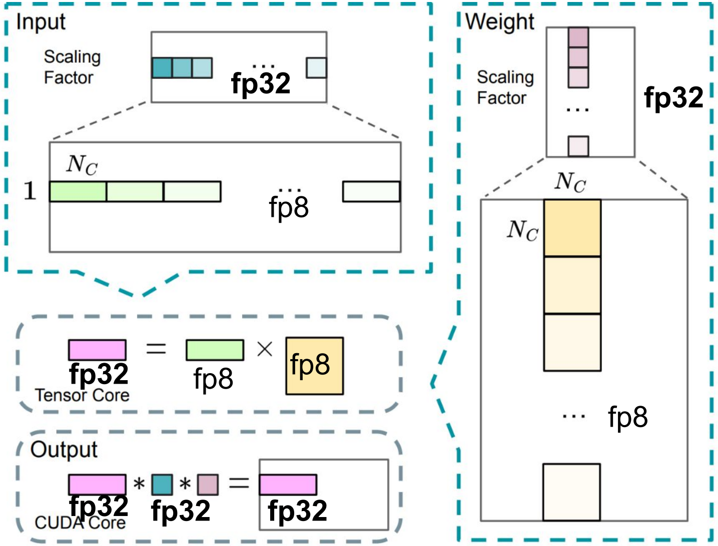


Что нам еще надо учесть?

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

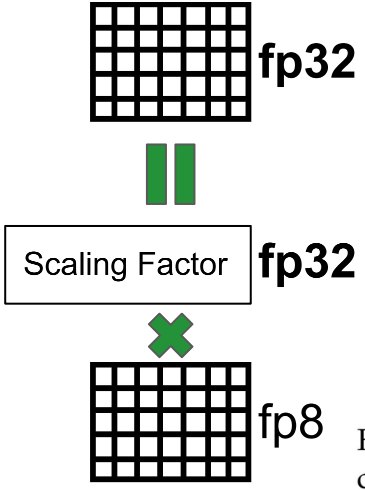
DeepSeek: fine-grained fp8 quantization

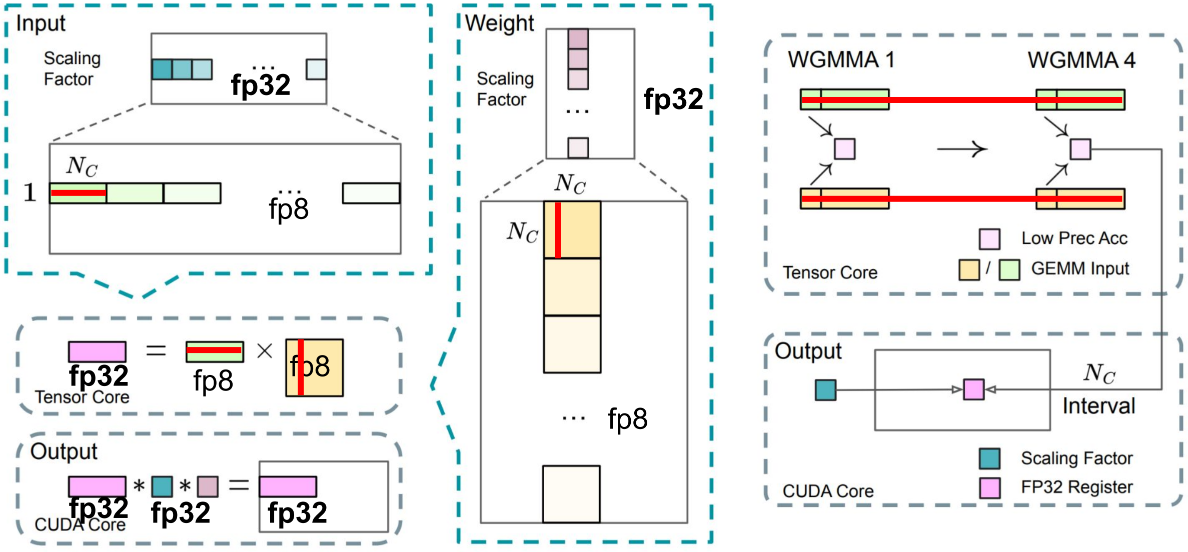




DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization





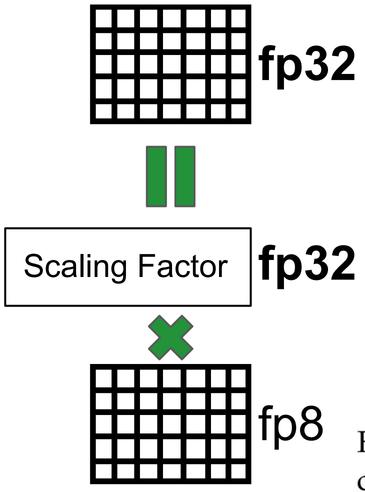
(a) Fine-grained quantization

(b) Increasing accumulation precision

Figure 7 | (a) We propose a fine-grained quantization method to mitigate quantization errors caused by feature outliers; for illustration simplicity, only `Fprop` is illustrated. (b) In conjunction with our quantization strategy, we improve the FP8 GEMM precision by promoting to CUDA Cores at an interval of $N_C = 128$ elements MMA for the high-precision accumulation.

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization



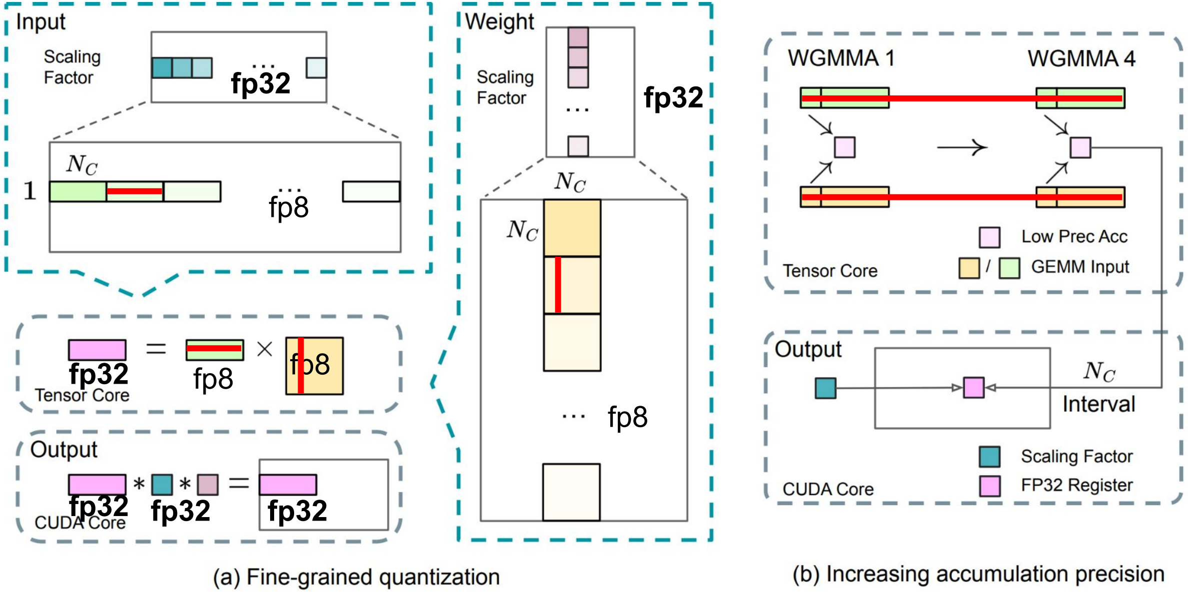
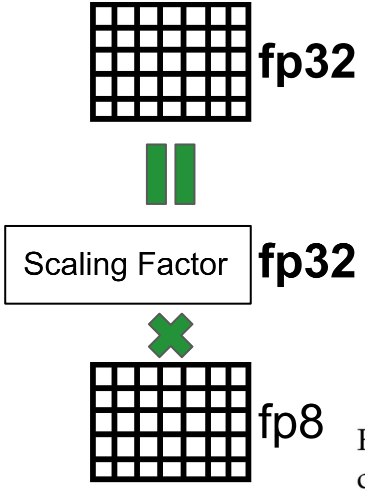
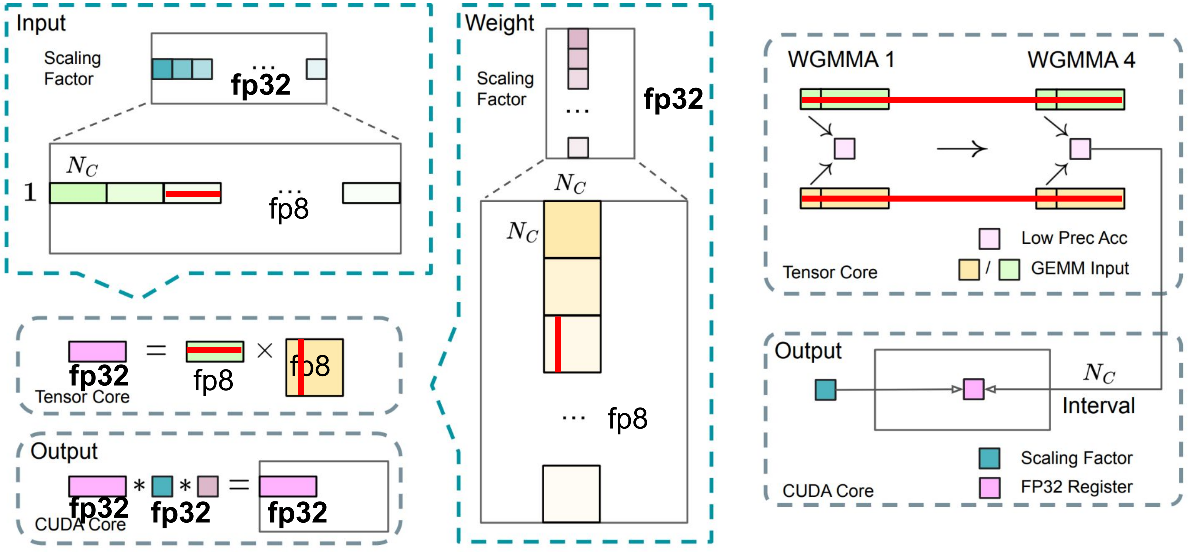


Figure 7 | (a) We propose a fine-grained quantization method to mitigate quantization errors caused by feature outliers; for illustration simplicity, only Fprop is illustrated. (b) In conjunction with our quantization strategy, we improve the FP8 GEMM precision by promoting to CUDA Cores at an interval of $N_C = 128$ elements MMA for the high-precision accumulation.

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization





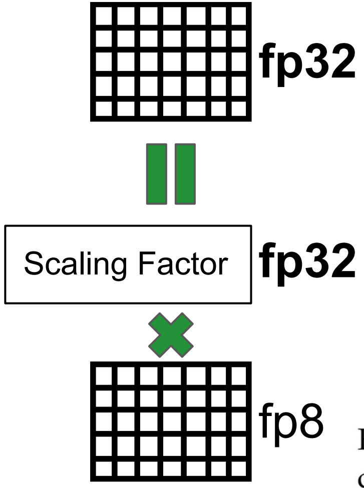
(a) Fine-grained quantization

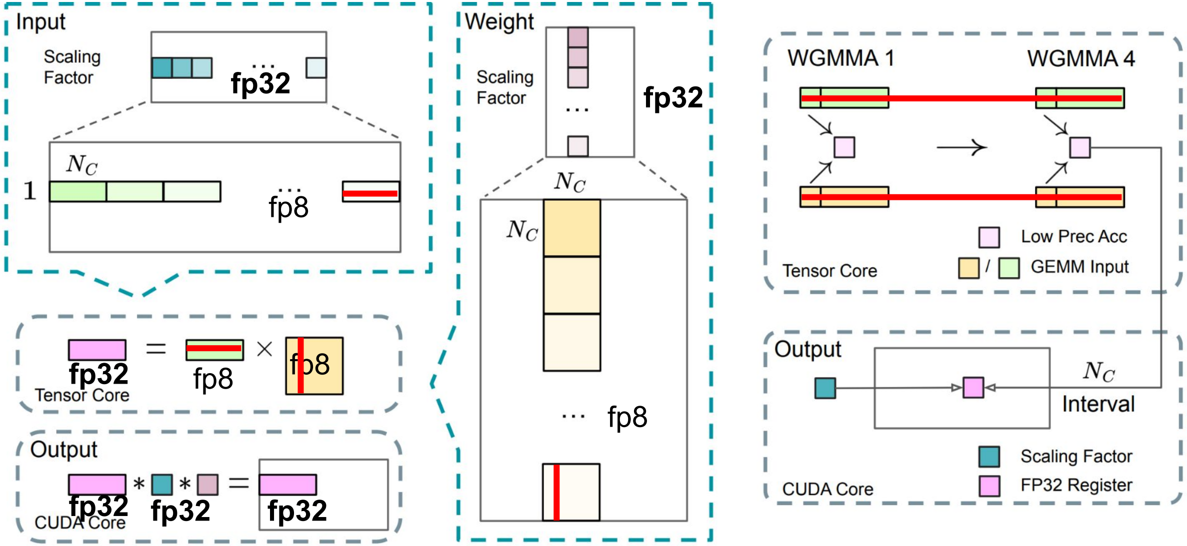
(b) Increasing accumulation precision

Figure 7 | (a) We propose a fine-grained quantization method to mitigate quantization errors caused by feature outliers; for illustration simplicity, only Fprop is illustrated. (b) In conjunction with our quantization strategy, we improve the FP8 GEMM precision by promoting to CUDA Cores at an interval of $N_C = 128$ elements MMA for the high-precision accumulation.

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization





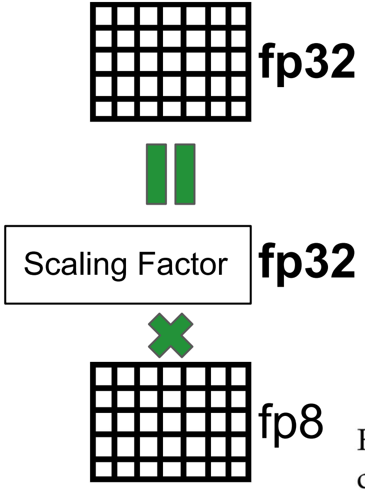
(a) Fine-grained quantization

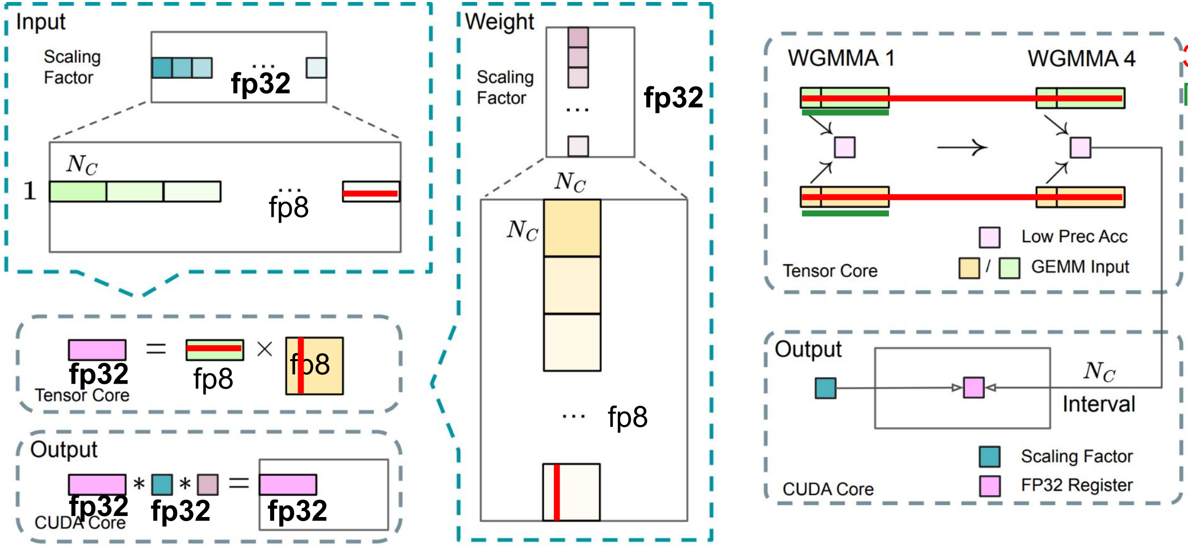
(b) Increasing accumulation precision

Figure 7 | (a) We propose a fine-grained quantization method to mitigate quantization errors caused by feature outliers; for illustration simplicity, only `Fprop` is illustrated. (b) In conjunction with our quantization strategy, we improve the FP8 GEMM precision by promoting to CUDA Cores at an interval of $N_C = 128$ elements MMA for the high-precision accumulation.

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization





(a) Fine-grained quantization

(b) Increasing accumulation precision

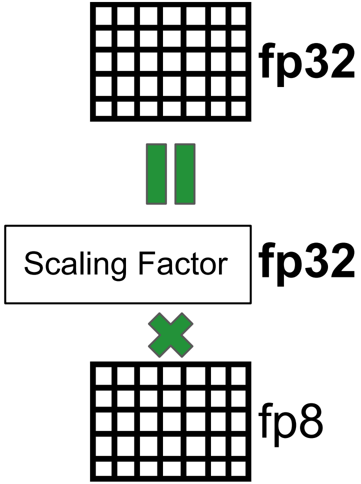
Зачем 4xWGMMA?

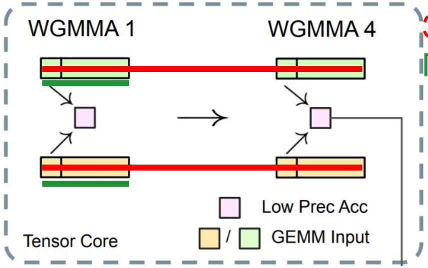
Почему не 1xWGMMA?

Figure 7 | (a) We propose a fine-grained quantization method to mitigate quantization errors caused by feature outliers; for illustration simplicity, only Fprop is illustrated. (b) In conjunction with our quantization strategy, we improve the FP8 GEMM precision by promoting to CUDA Cores at an interval of $N_C = 128$ elements MMA for the high-precision accumulation.

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization



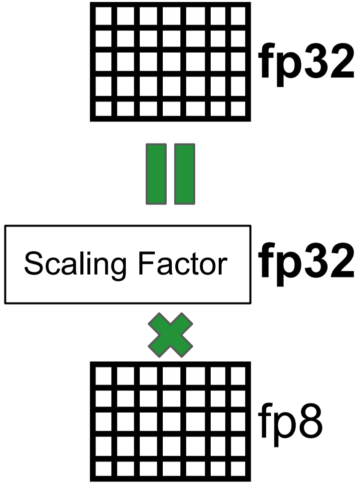


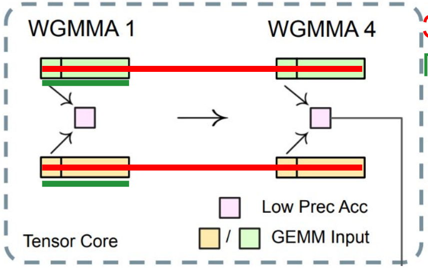
Зачем 4xWGMMA?
Почему не 1xWGMMA?

It is worth noting that this modification reduces the WGMMA (Warpgroup-level Matrix Multiply-Accumulate) instruction issue rate for a single warpgroup. However, on the H800 architecture, it is typical for two WGMMA to persist concurrently: while one warpgroup performs the promotion operation, the other is able to execute the MMA operation. This design enables overlapping of the two operations, maintaining high utilization of Tensor Cores. Based on our experiments, setting $N_C = 128$ elements, equivalent to 4 WGMMAs, represents the minimal accumulation interval that can significantly improve precision without introducing substantial overhead.

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: fine-grained fp8 quantization





Зачем 4xWGMMA?
Почему не 1xWGMMA?

It is worth noting that this modification reduces the WGMMA (Warpgroup-level Matrix Multiply-Accumulate) instruction issue rate for a single warpgroup. However, on the H800 architecture, it is typical for two WGMMA to persist concurrently: while one warpgroup performs the promotion operation, the other is able to execute the MMA operation. This design enables overlapping of the two operations, maintaining high utilization of Tensor Cores. Based on our experiments, setting $N_C = 128$ elements, equivalent to 4 WGMMAs, represents the minimal accumulation interval that can significantly improve precision without introducing substantial overhead.

Не знаю почему нужно делать 4xWGMMA + PROMOTION вместо 4x(WGMMA + PROMOTION)

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

DeepSeek: x2 ускорение обучения (fp16 → fp8)

DeepSeek-V3 Technical Report - https://arxiv.org/abs/2412.19437

Репозиторий - https://github.com/deepseek-ai/DeepGEMM (GEMM - General Matrix Multiplications)

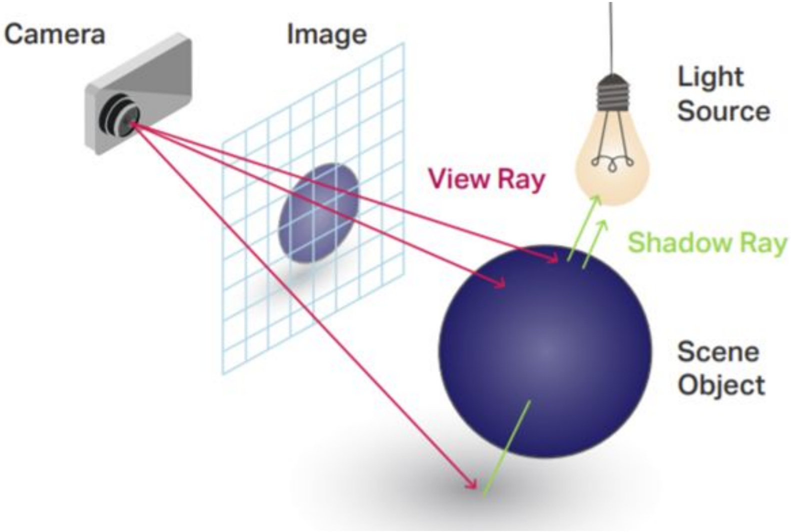
Fast Matrix-Multiplication with WGMMA on NVIDIA® Hopper™ GPUs

https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/fp8_primer.html

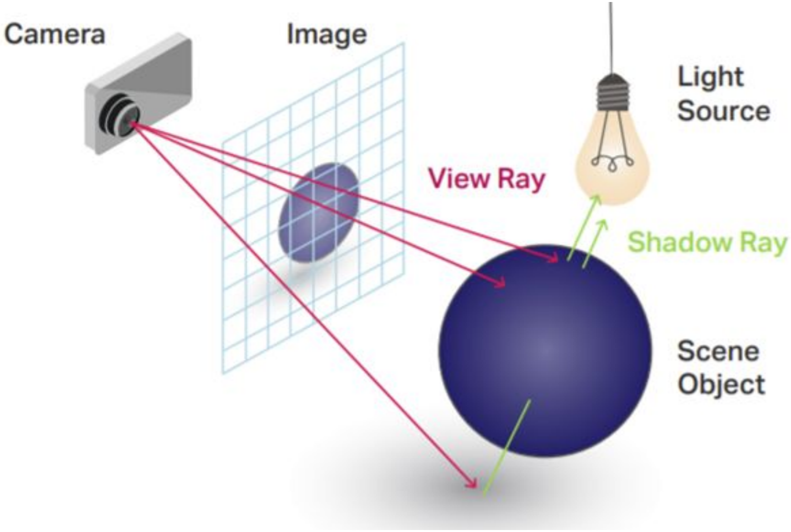
Глава 8: Ray Tracing

real-time BVH, ray tracing cores

Ray Tracing

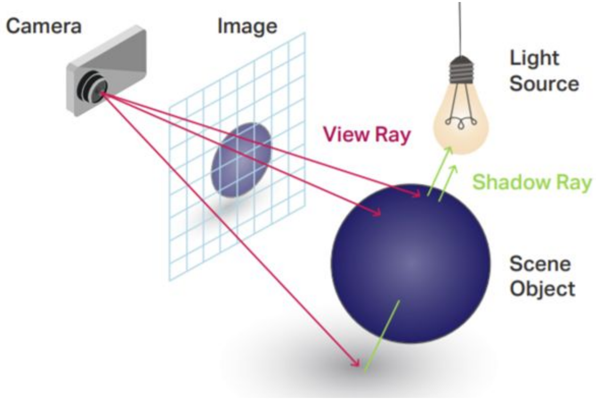


Ray Tracing

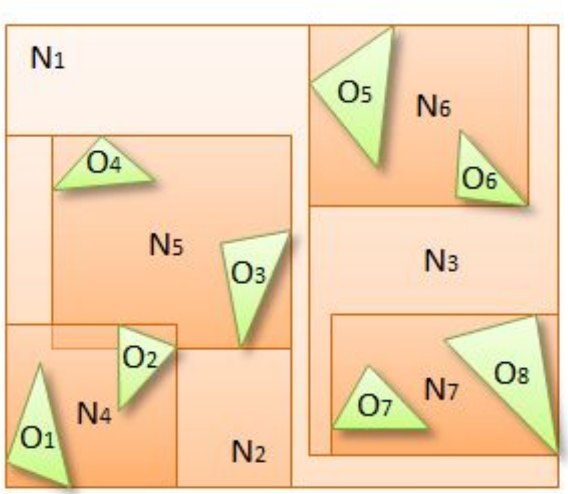


Как найти пересечение с треугольником?
Перебором для каждого луча всех треугольников?

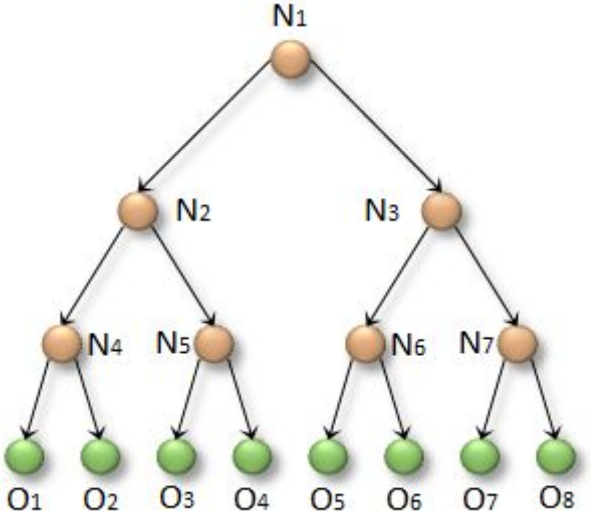
Ray Tracing



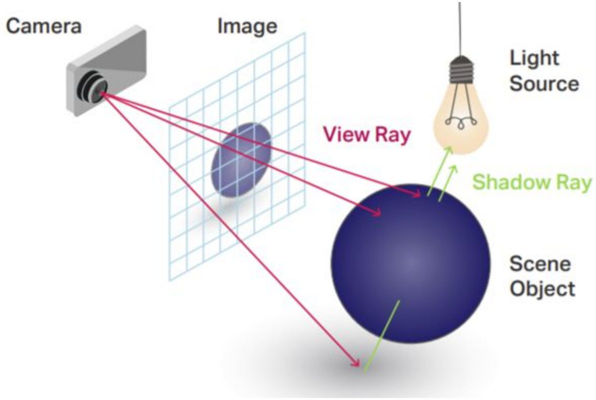
BVH - Bounding Volume Hierarchy

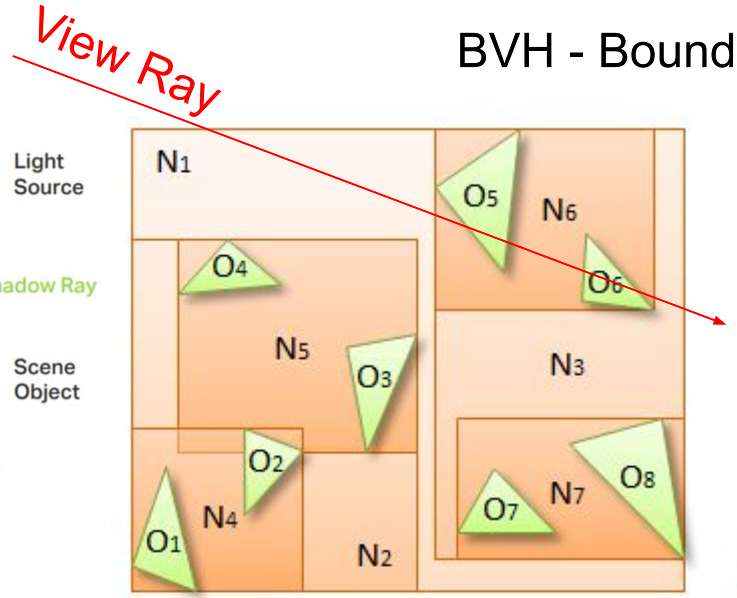


Axis-Aligned Bounding Boxes



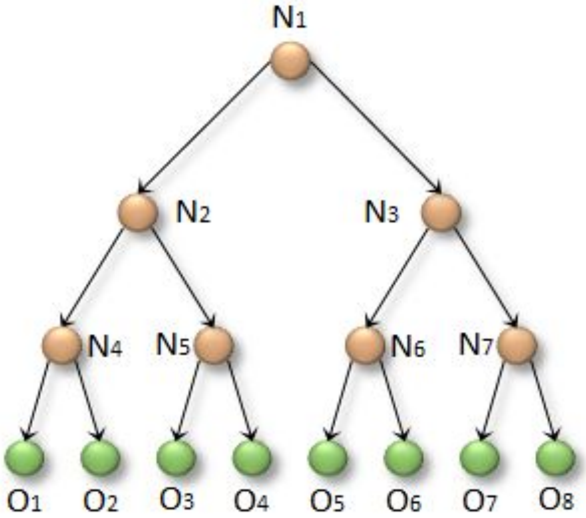
Ray Tracing



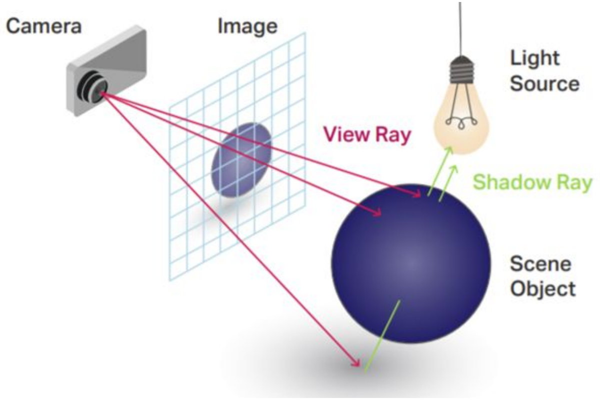


Axis-Aligned Bounding Boxes

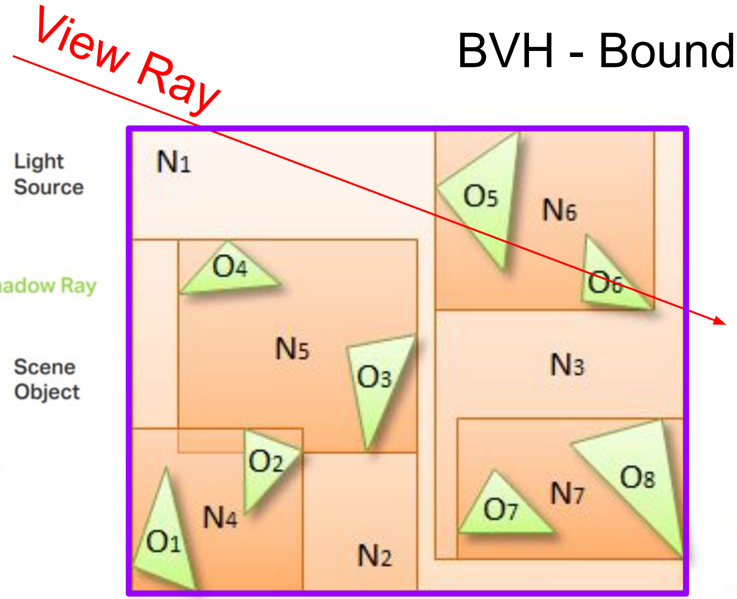
BVH - Bounding Volume Hierarchy



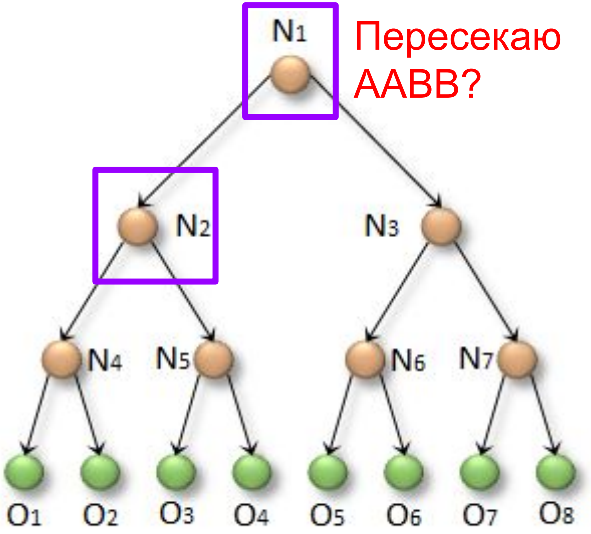
Ray Tracing



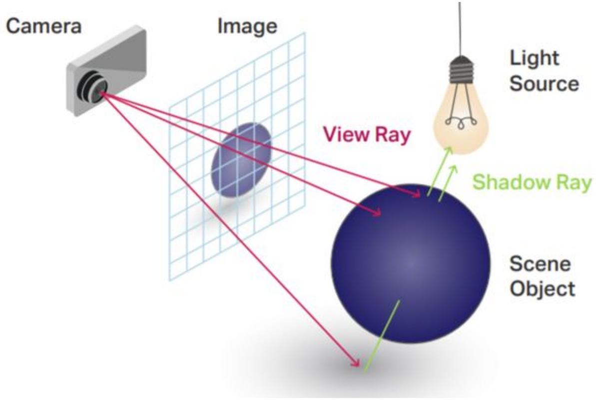
BVH - Bounding Volume Hierarchy

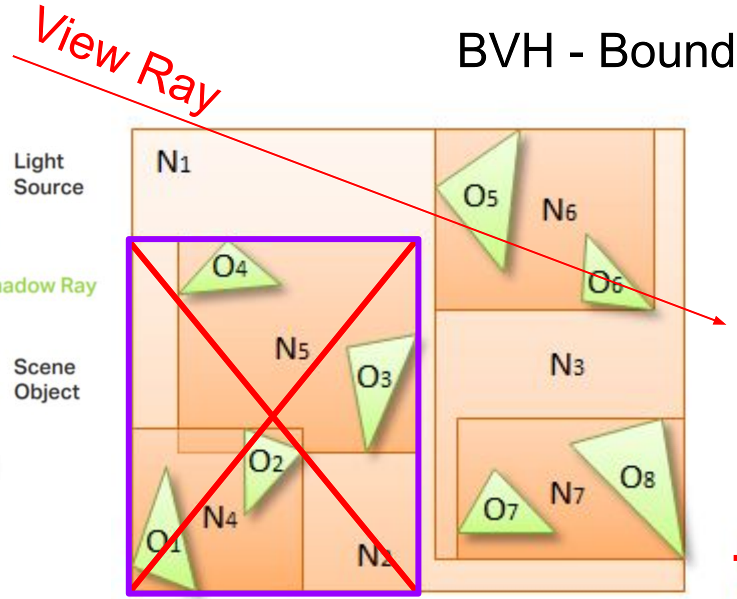


Axis-Aligned Bounding Boxes



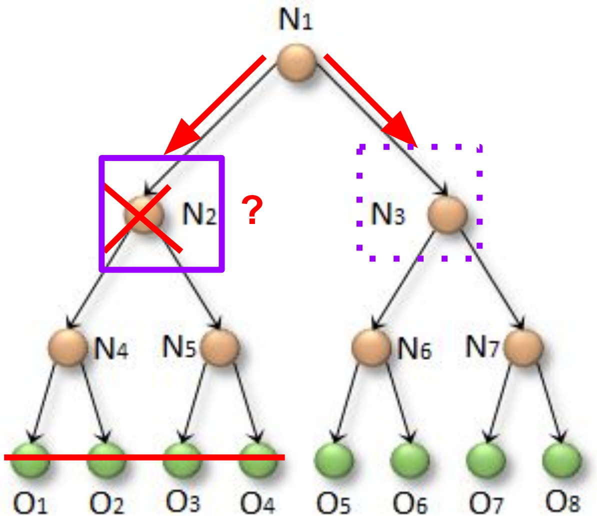
Ray Tracing



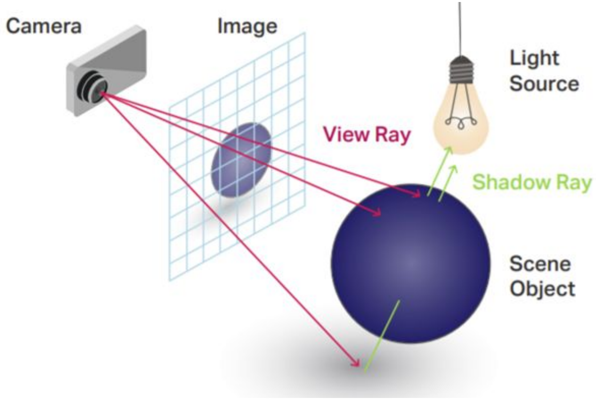


Axis-Aligned Bounding Boxes

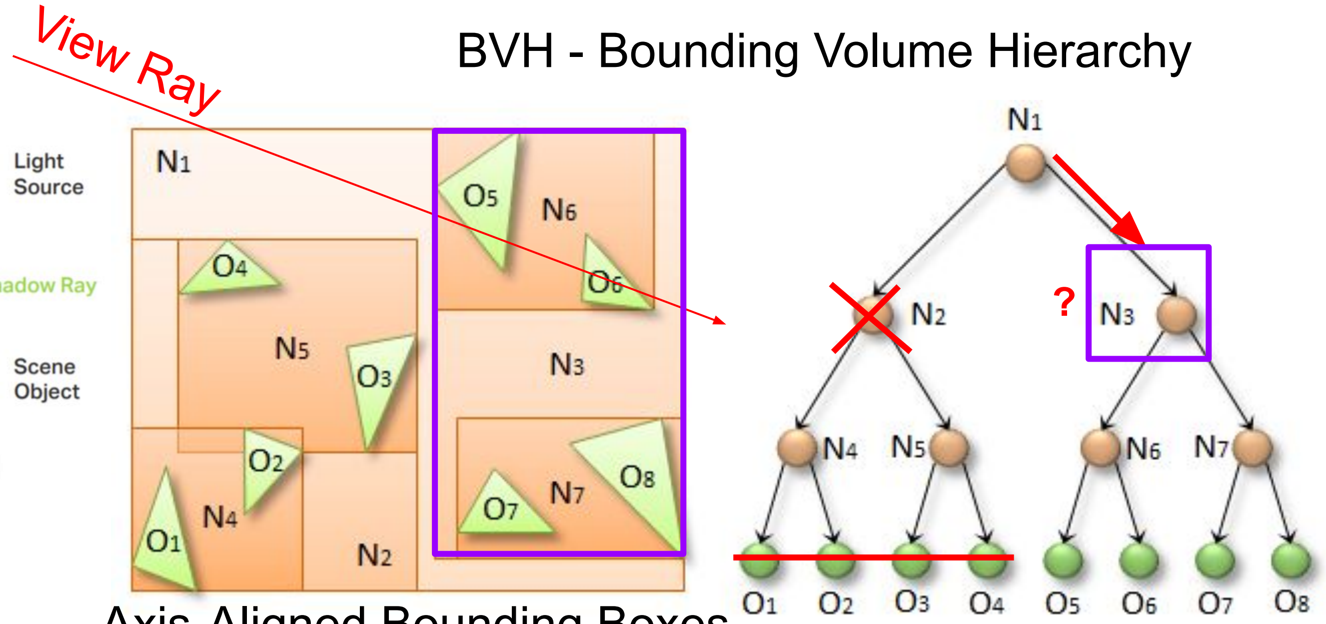
BVH - Bounding Volume Hierarchy



Ray Tracing

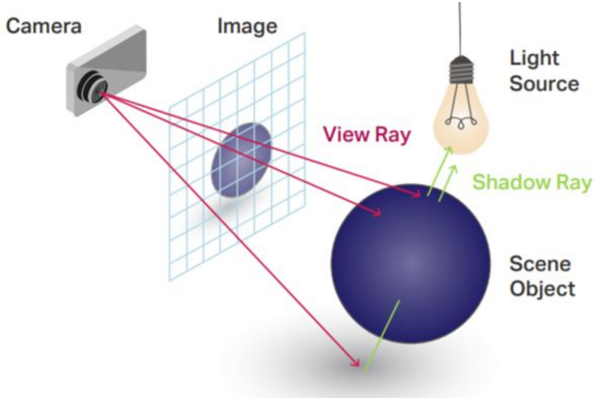


BVH - Bounding Volume Hierarchy

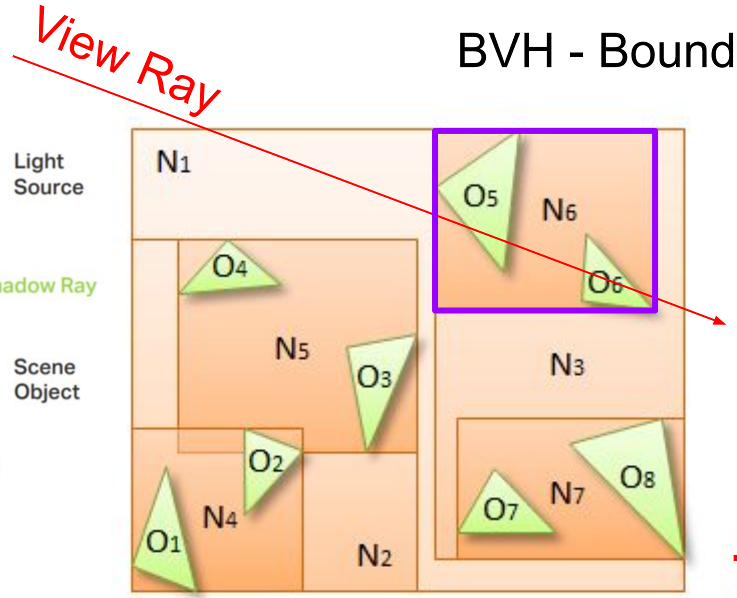


Axis-Aligned Bounding Boxes

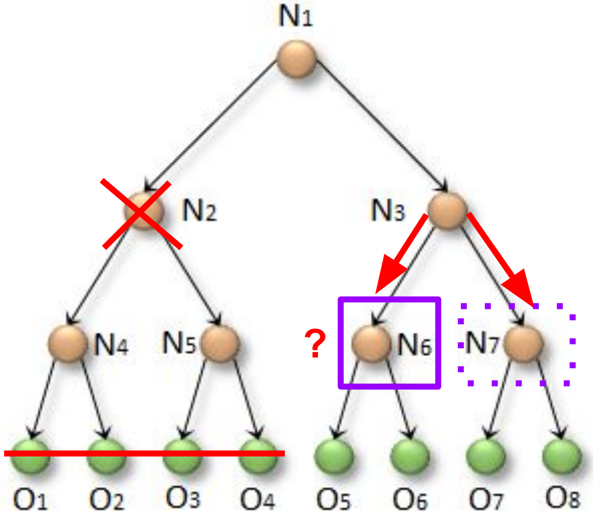
Ray Tracing



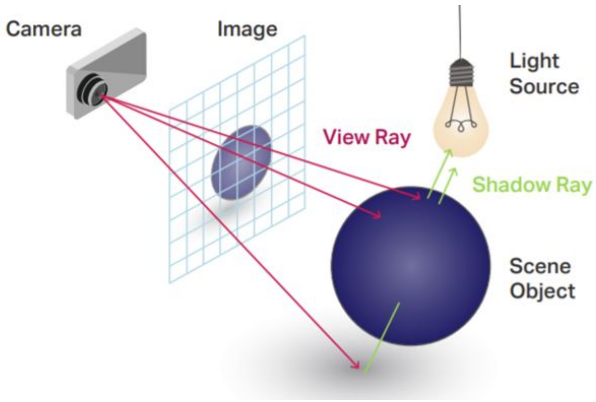
BVH - Bounding Volume Hierarchy

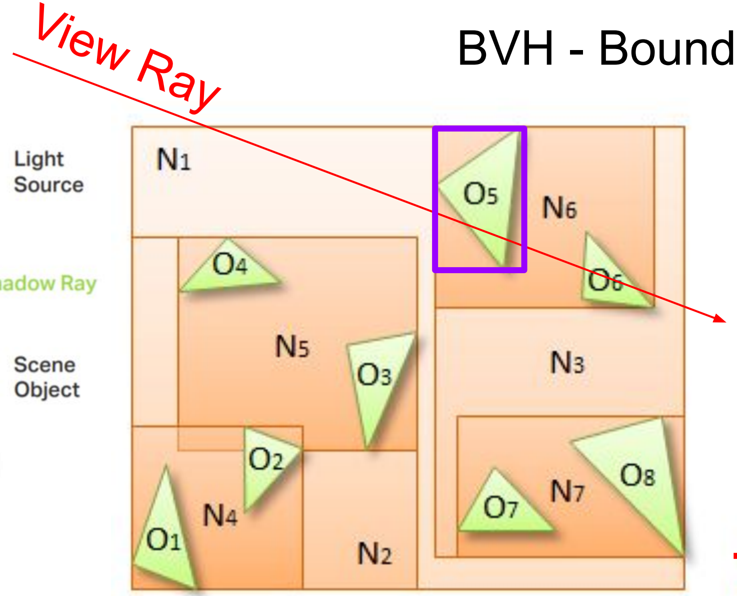


Axis-Aligned Bounding Boxes



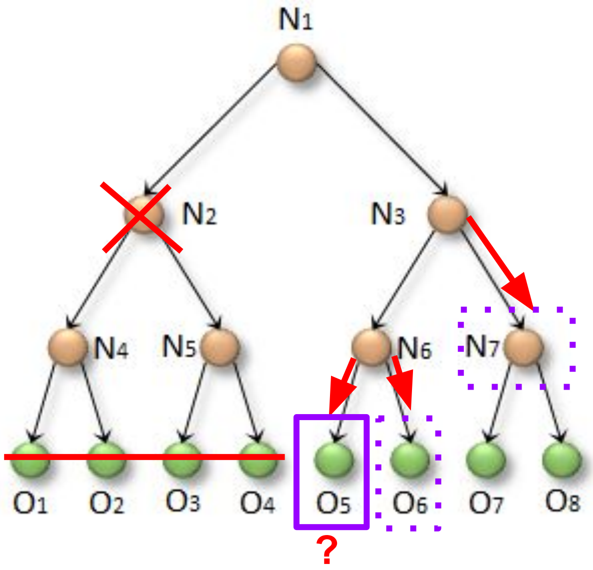
Ray Tracing



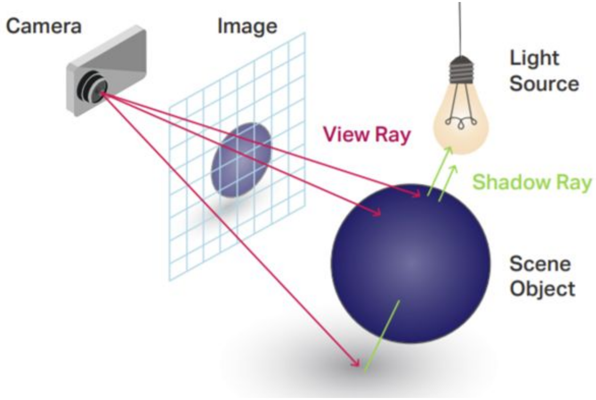


Axis-Aligned Bounding Boxes

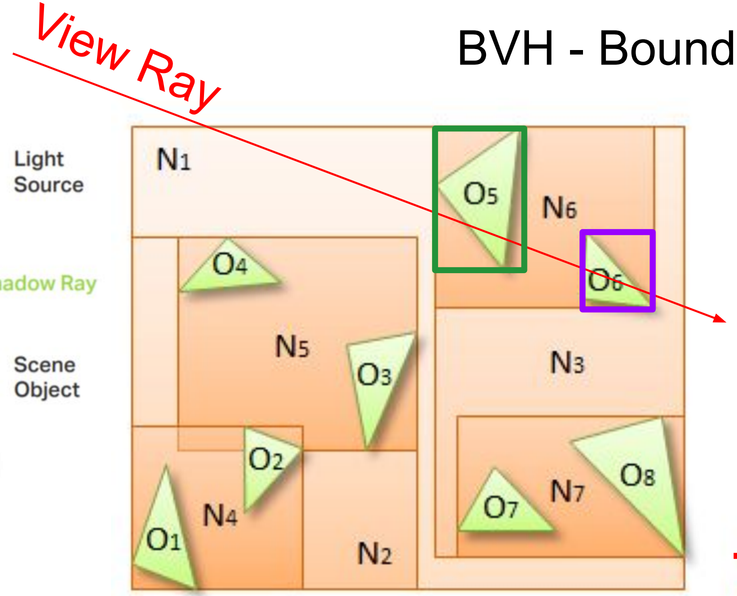
BVH - Bounding Volume Hierarchy



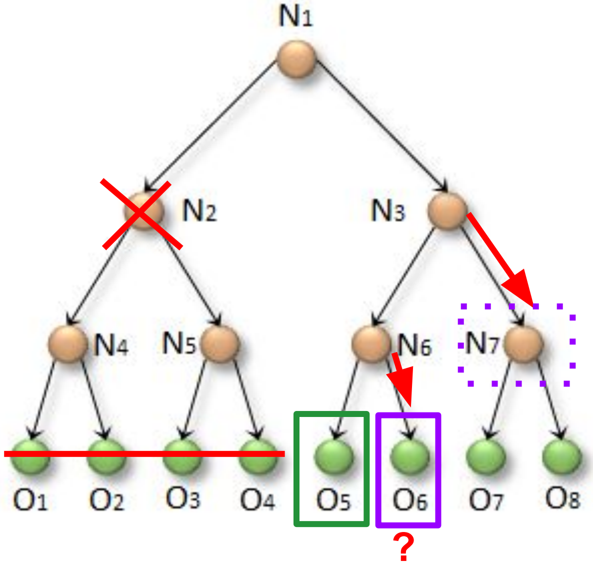
Ray Tracing



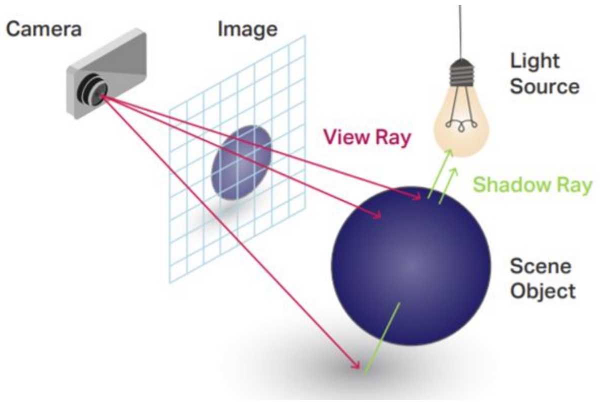
BVH - Bounding Volume Hierarchy

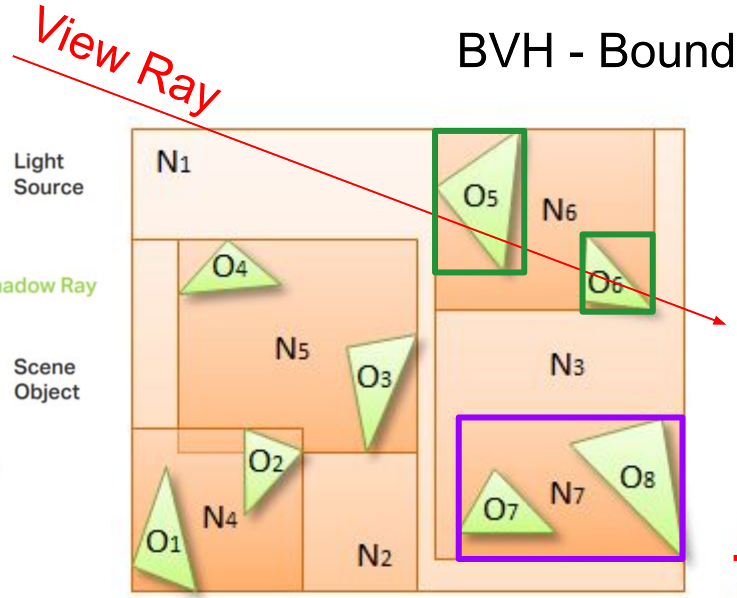


Axis-Aligned Bounding Boxes



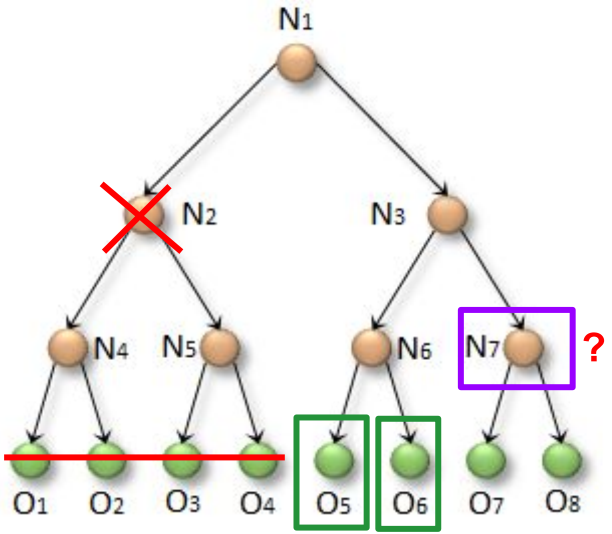
Ray Tracing



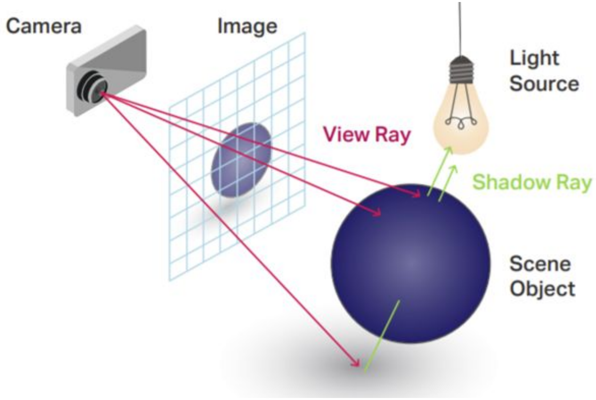


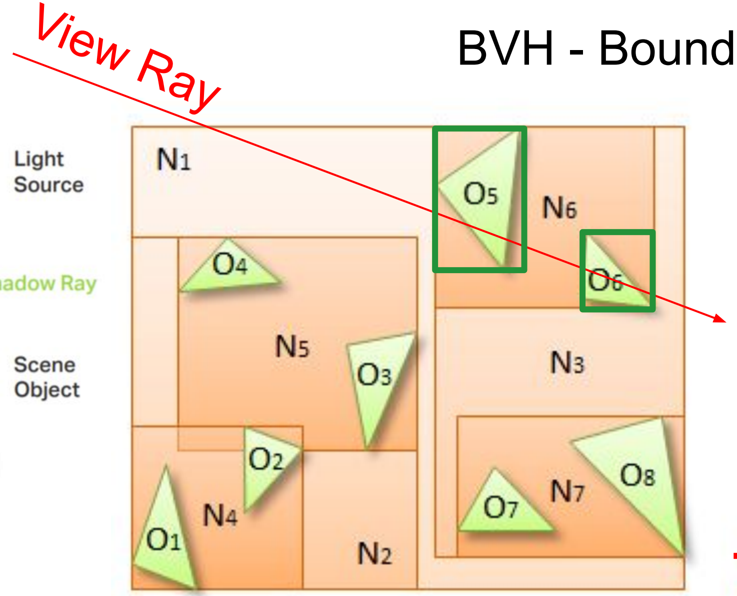
Axis-Aligned Bounding Boxes

BVH - Bounding Volume Hierarchy



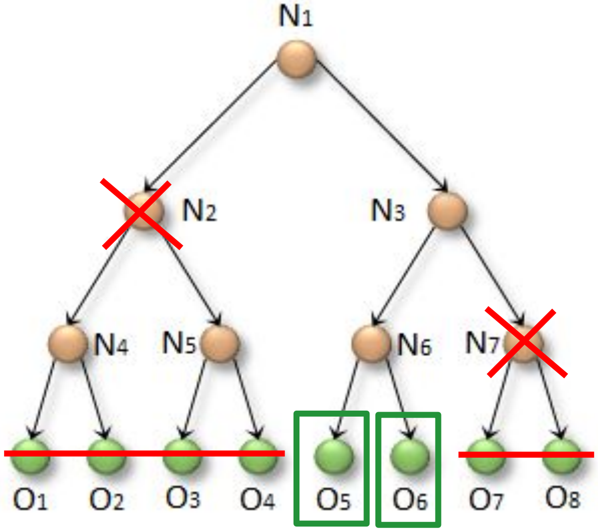
Ray Tracing



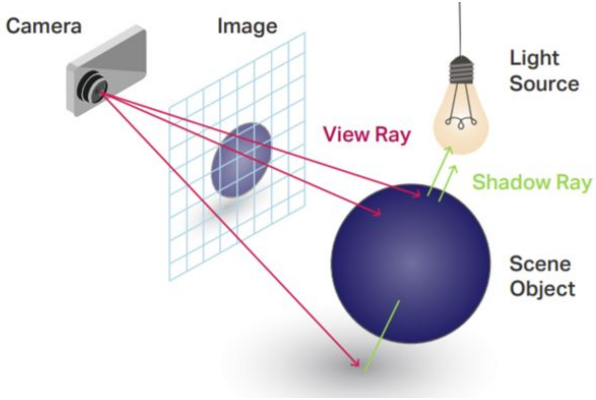


Axis-Aligned Bounding Boxes

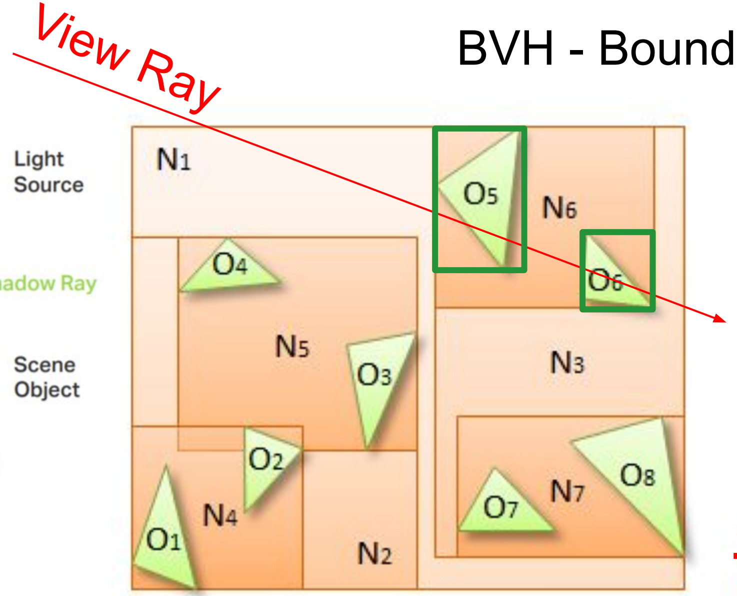
BVH - Bounding Volume Hierarchy



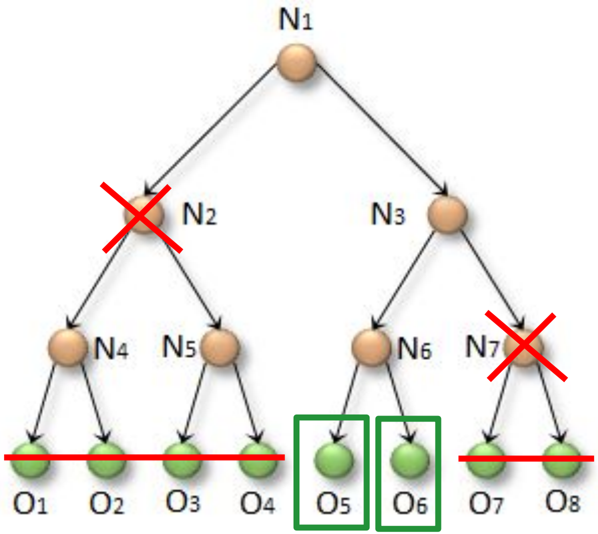
Ray Tracing



BVH - Bounding Volume Hierarchy

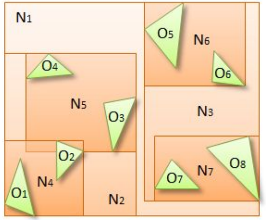


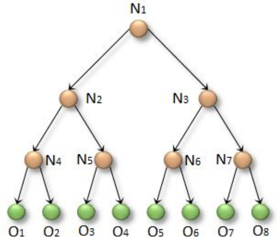
Axis-Aligned Bounding Boxes



Как это будет выглядеть в коде?

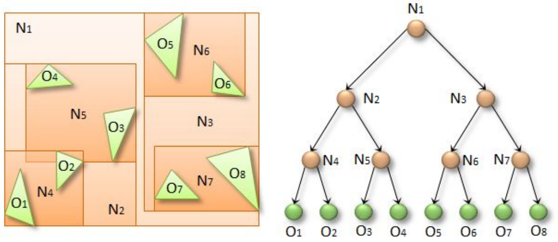
BVH обход (рекурсивный)



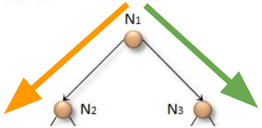



```
void traverseRecursive(AABB                     queryAABB,
                       int                      queryObjectIdx,
                       const __global BVHNode*  node,
                       __local CollisionList*   list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                              childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                              childR, list);
        }
    }
}
```

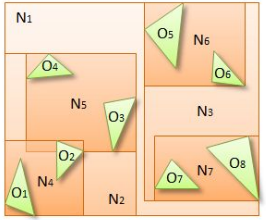
BVH обход (рекурсивный)

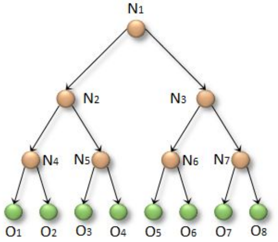



```
void traverseRecursive(AABB                         queryAABB,
                       int                          queryObjectIdx,
                       const __global BVHNode*      node,
                       __local CollisionList*       list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                              childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                              childR, list);
        }
    }
}
```

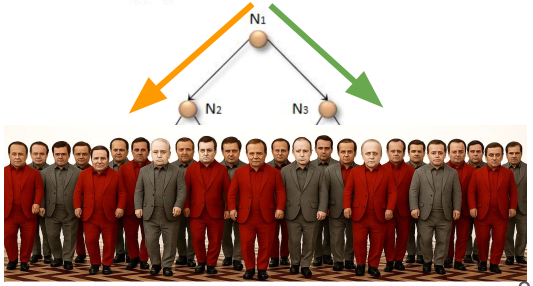


BVH обход (рекурсивный)



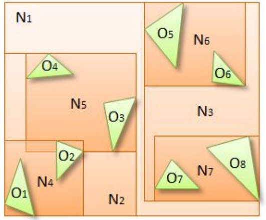


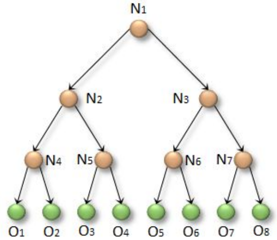

```
void traverseRecursive(AABB                     queryAABB,
                       int                      queryObjectIdx,
                       const __global BVHNode*  node,
                       __local CollisionList*   list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                              childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                              childR, list);
        }
    }
}
```



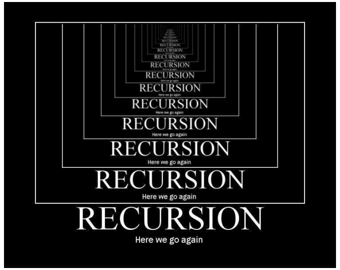
Code divergence!

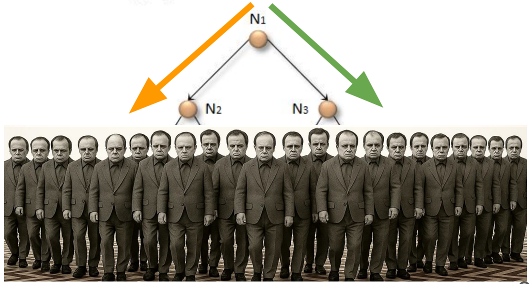
BVH обход (рекурсивный)






```
void traverseRecursive(AABB                        queryAABB,
                       int                         queryObjectIdx,
                       const __global BVHNode*     node,
                       __local CollisionList*      list)
{
    if (checkOverlap(node->getAABB(), queryAABB)) {
        if (node->isLeaf()) {
            list.add(queryObjectIdx, node->getObjectIdx());
        } else {
            const __global BVHNode childL = node->getLeftChild();
            const __global BVHNode childR = node->getRightChild();
            traverseRecursive(queryAABB, queryObjectIdx,
                              childL, list);
            traverseRecursive(queryAABB, queryObjectIdx,
                              childR, list);
        }
    }
}
```

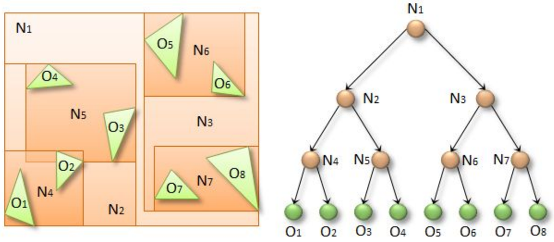




Что делать?

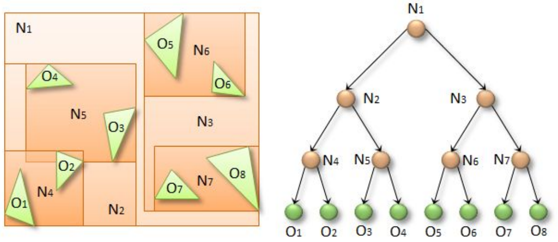
Code divergence!

BVH обход (на стеке)

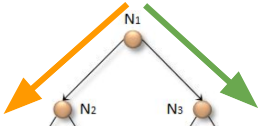



```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;
do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```

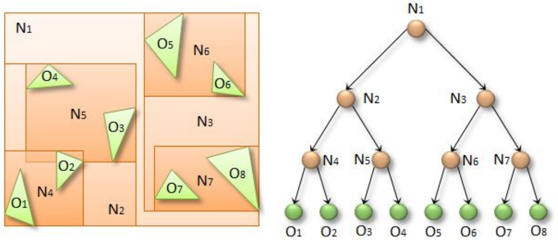
BVH обход (на стеке)



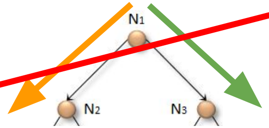

```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;
do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```



BVH обход (на стеке)

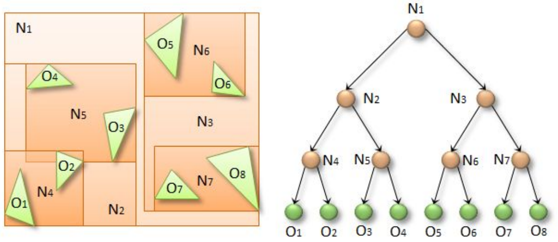



```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;
do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```

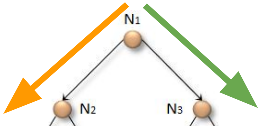


Code divergence остался только в виде числа итераций цикла.

BVH обход (на стеке)



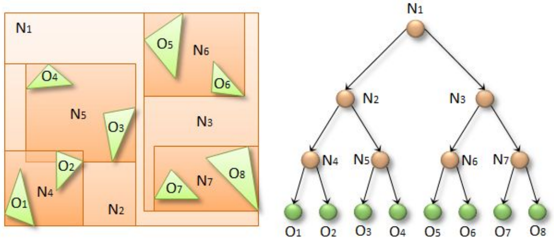

```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;
do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```



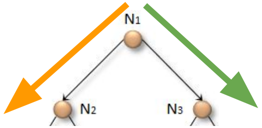
Code divergence остался только в виде числа итераций цикла.

Data divergence увеличивает количество запрашиваемых кеш-линий.

BVH обход (на стеке)




```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;
do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```



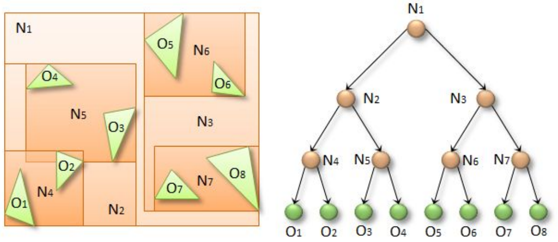
Как увеличить когерентность по данным?

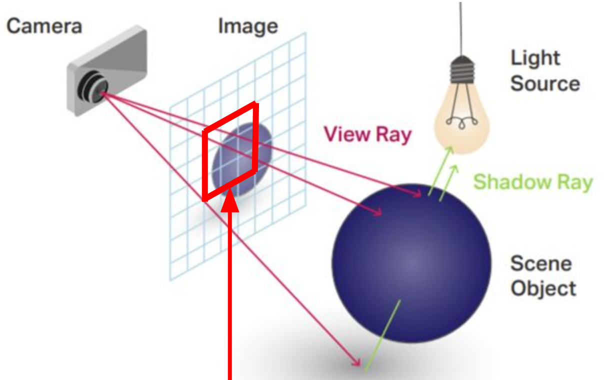
Code divergence остался только в виде числа итераций цикла.

Data divergence увеличивает количество запрашиваемых кеш-линий.

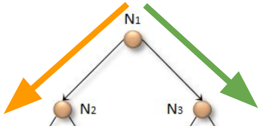
BVH обход (на стеке)

```
BVHNode* stack[MAX_STACK_SIZE];
BVHNode* node = bvhRoot;
do {
    BVHNode* childL = node->getLeftChild();
    BVHNode* childR = node->getRightChild();
    bool overlapL = checkOverlap(queryAABB, childL->getAABB());
    bool overlapR = checkOverlap(queryAABB, childR->getAABB());

    if (overlapL && childL->isLeaf())
        list.add(queryObjectIdx, childL->getObjectIdx());
    if (overlapR && childR->isLeaf())
        list.add(queryObjectIdx, childR->getObjectIdx());

    if (!traverseL && !traverseR) {
        node = stack.pop();
    } else {
        node = traverseL ? childL : childR;
        if (traverseL && traverseR)
            stack.push(childR);
    }
} while (node != NULL);
```





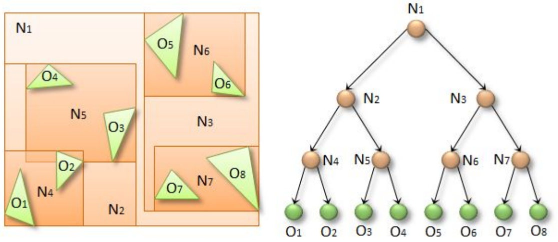
Данные когерентнее если в одном **warp - пучок лучей**!



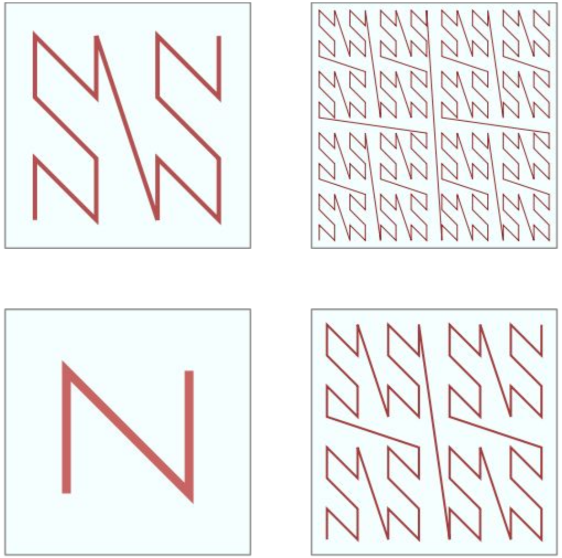
Code divergence остался только в виде числа итераций цикла.

Data divergence увеличивает количество запрашиваемых кеш-линий.

Real-time BVH construction

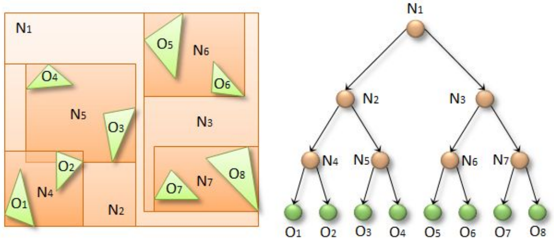


Z curve, Morton Code

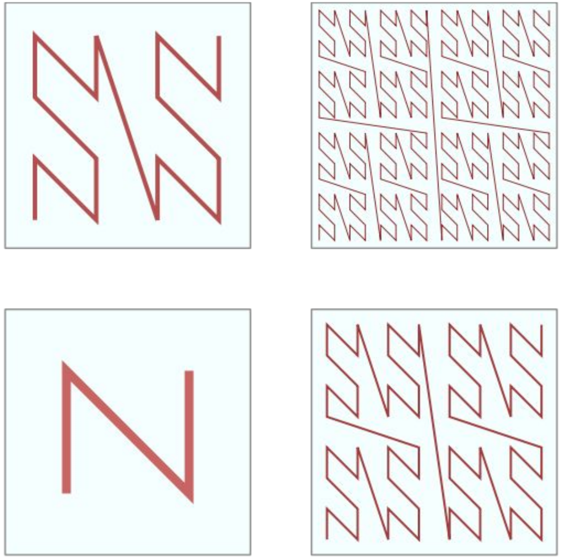


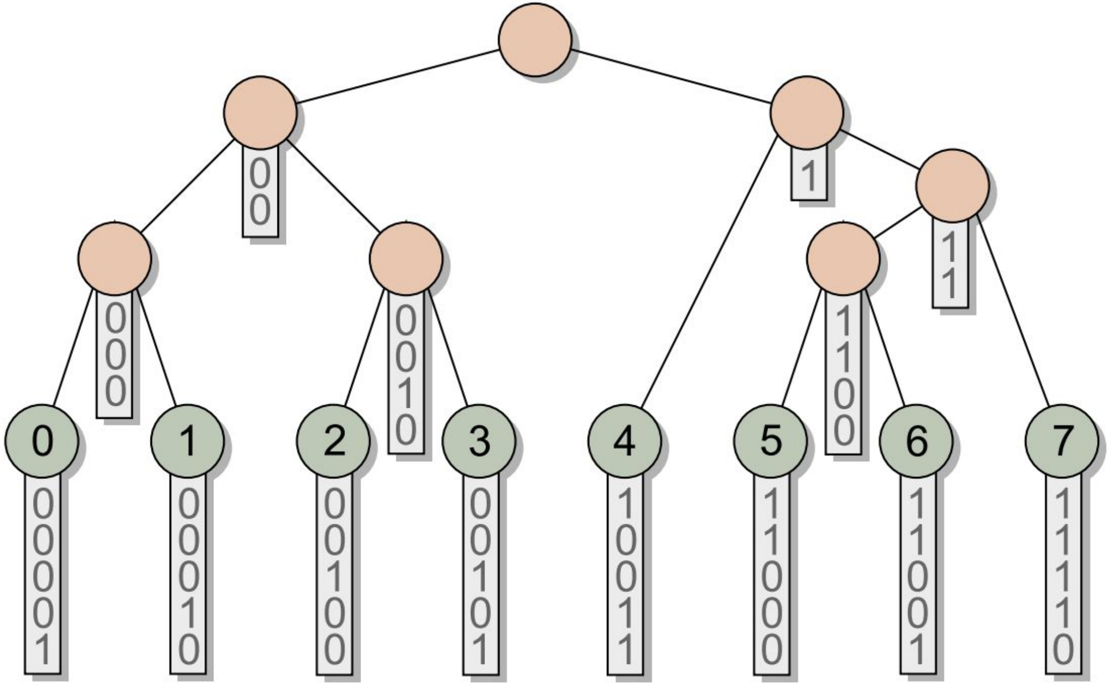
https://youtu.be/aDINUfYfaAY?list=PLlb7e2G7aSpSkDWlyJQzT9Qx9rrgKSgAp&t=3528

Real-time BVH construction

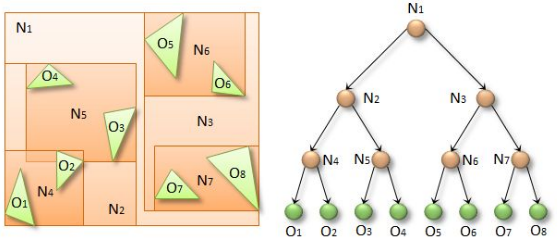


Z curve, Morton Code



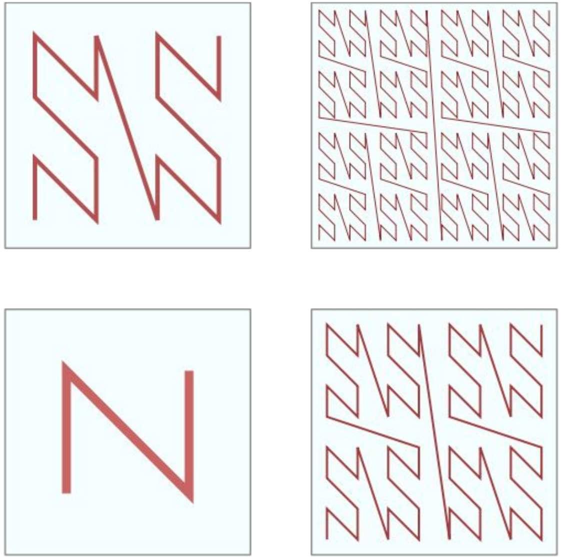


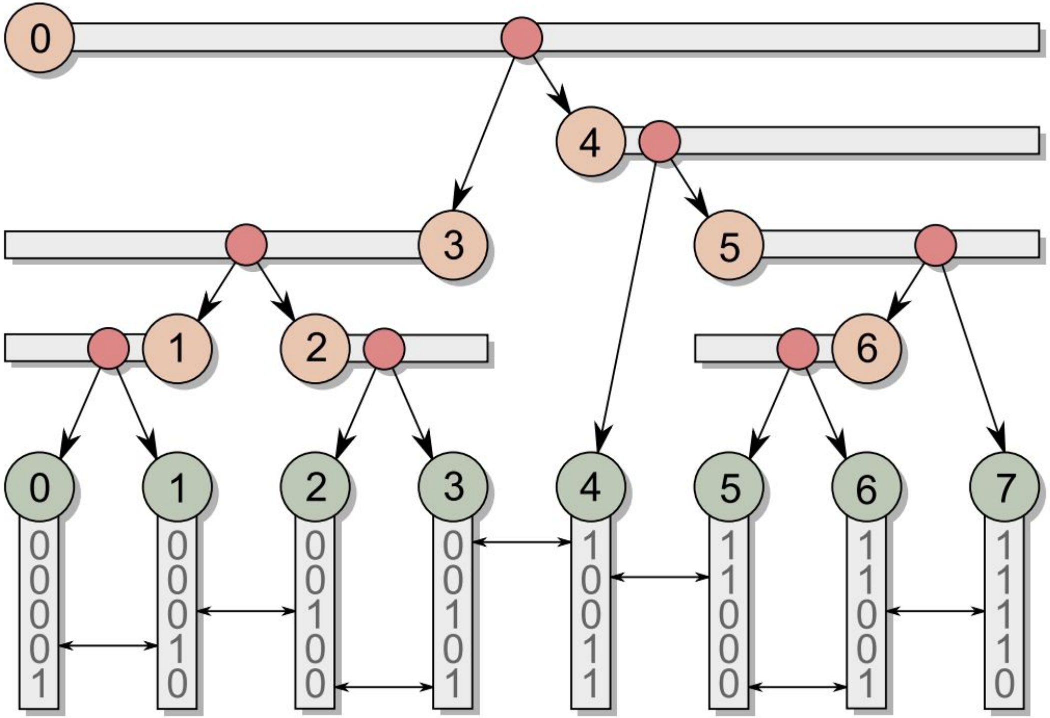
https://youtu.be/aDINUfYfaAY?list=PLlb7e2G7aSpSkDWlyJQzT9Qx9rrgKSgAp&t=3528



Real-time BVH construction

Z curve, Morton Code





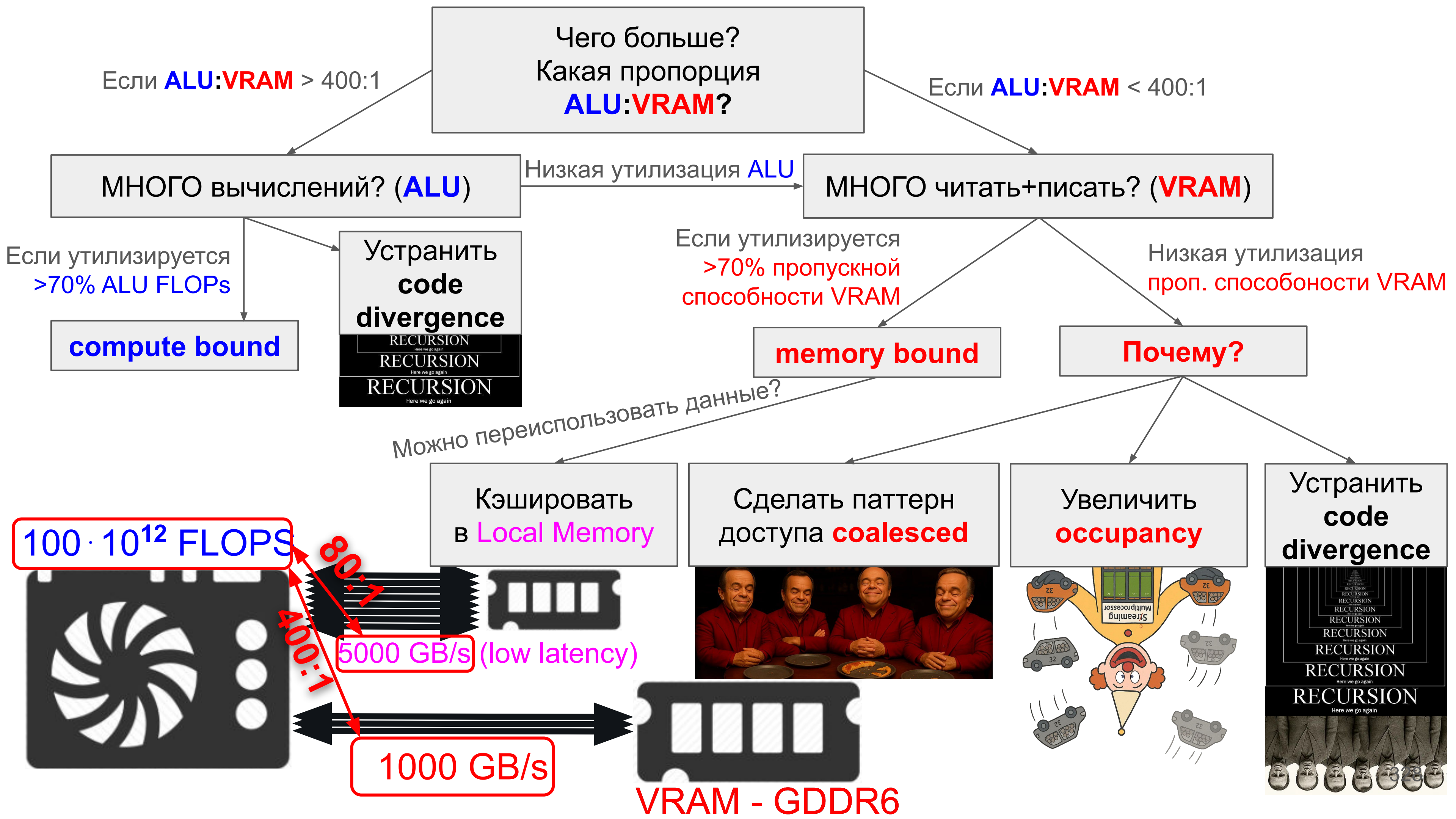
https://youtu.be/aDINUfYfaAY?list=PLlb7e2G7aSpSkDWlyJQzT9Qx9rrgKSgAp&t=3528

Ray Tracing Cores

1) Ускоряют обход BVH иерархии
2) Ускоряют пересечение луча с треугольником

Глава 9: Выводы

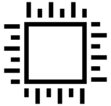
Какие алгоритмы ускоряются на GPU?
OpenCL, CUDA или Vulkan?



Чего больше?
Какая пропорция
ALU:VRAM?
Если ALU:VRAM > 400:1
Если ALU:VRAM < 400:1
МНОГО вычислений? (ALU)
Низкая утилизация ALU
МНОГО читать+писать? (VRAM)
Если утилизируется
>70% ALU FLOPs
compute bound
Устранить
code
divergence
RECURSION
RECURSION
Here we go again
RECURSION
Here we go again
Если утилизируется
>70% пропускной
способности VRAM
memory bound
Низкая утилизация
проп. способоности VRAM
Почему?
Можно переиспользовать данные?
Кэшировать
в Local Memory
Сделать паттерн
доступа coalesced
Увеличить
occupancy
Устранить
code
divergence
RECURSION
RECURSION
RECURSION
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
100 · 10^12 FLOPS
80:1
400:1
5000 GB/s (low latency)
1000 GB/s
VRAM - GDDR6

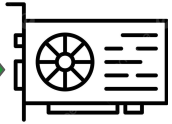
Какие алгоритмы ускоряются на GPU?

Нет

CPU 

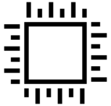
Задача большая?
Вычисления можарируют трансфер данных по PCI-E шине?
Или данные уже на GPU?

Да

 **GPU**

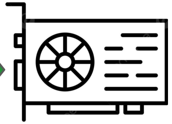
Какие алгоритмы ускоряются на GPU?

Нет

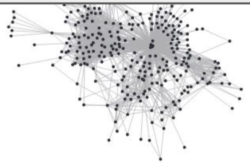
CPU 

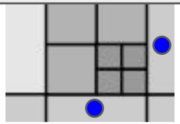
Задача большая?
Вычисления можарируют трансфер данных по PCI-E шине?
Или данные уже на GPU?

Да

 **GPU**

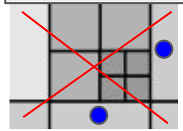
Произвольные графы, несбалансированное октодерево, min-cut/max flow (BK, IBFS), disjoint sets (CHM)

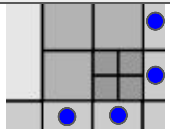


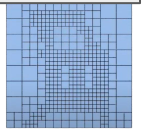


Есть ли **массовый единообразный параллелизм**?
Структура данных - **регулярная?**
Не будет ли **uncoalesced** memory access pattern? Не будет ли **гонки**?
Не будет ли **code divergence**?

FEMs (Finite Element Methods) - **численные методы** на **регулярной решетке** и 2:1 сбалансированном октодереве







Какие алгоритмы ускоряются на GPU?

Нет

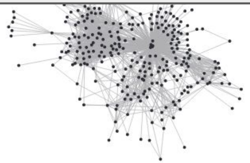
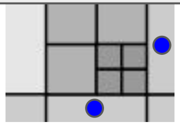
CPU

Задача большая?
Вычисления можарируют трансфер данных по PCI-E шине?
Или данные уже на GPU?

Да

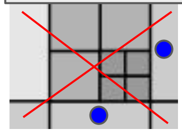
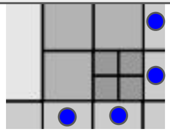
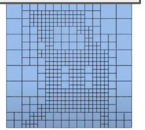
GPU

Произвольные графы, несбалансированное октодерево, min-cut/max flow (BK, IBFS), disjoint sets (CHM)

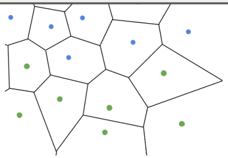
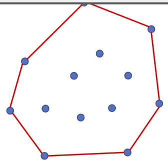



Есть ли **массовый единообразный параллелизм**?
Структура данных - **регулярная?**
Не будет ли **uncoalesced** memory access pattern? Не будет ли **гонки**?
Не будет ли **code divergence**?

FEMs (Finite Element Methods) - **численные методы** на **регулярной решетке** и 2:1 сбалансированном октодереве

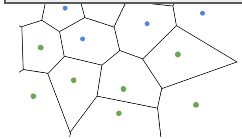
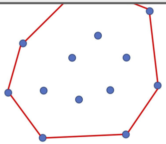




GIS, Триангуляция Делоне

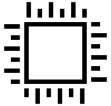
Не нужна ли **интервальная, рациональная, длинная** арифметика для абсолютной точности?

Иногда можно посчитать простые случаи на GPU, затем **на CPU досчитать** пограничные (**Триангуляция Делоне**)

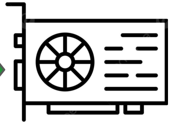

Какие алгоритмы ускоряются на GPU?

Нет

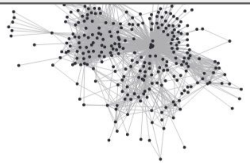
CPU 

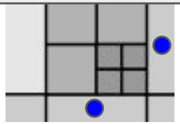
Задача большая?
Вычисления можарируют трансфер данных по PCI-E шине?
Или данные уже на GPU?

Да

 **GPU**

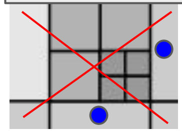
Произвольные графы, несбалансированное октодерево, min-cut/max flow (BK, IBFS), disjoint sets (CHM)

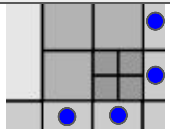


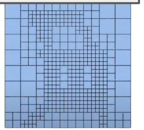


Есть ли **массовый единообразный параллелизм**?
Структура данных - **регулярная?**
Не будет ли **uncoalesced** memory access pattern? Не будет ли **гонки**?
Не будет ли **code divergence**?

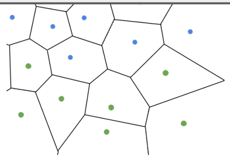
FEMs (Finite Element Methods) - **численные методы** на **регулярной решетке** и 2:1 сбалансированном октодереве

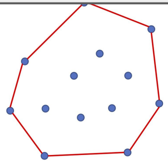






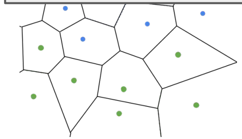
GIS, Триангуляция Делоне

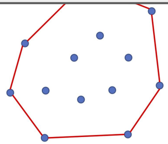




Не нужна ли **интервальная, рациональная, длинная** арифметика для абсолютной точности?

Иногда можно посчитать простые случаи на GPU, затем **на CPU досчитать** пограничные (**Триангуляция Делоне**)

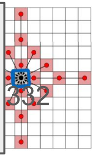




Приоритетные очереди (жадины), min-cut/max flow (BK, IBFS), disjoint sets (CHM)

Нет ли **ГОРАЗДО** более эффективного асимптотически алгоритма, но при этом очень **линейного**?

Иногда можно выкрутиться:
- merge-sort
- BVH construction
- Gipuma



CUDA, Vulkan или OpenCL?

NVIDIA CUDA

Vulkan

OpenCL™

CUDA, Vulkan или OpenCL?

Запуск везде:
AMD, NVidia, Intel, Snapdragon, …
Windows, Linux, Mac, Android

MoltenVK

NVIDIA CUDA

Vulkan

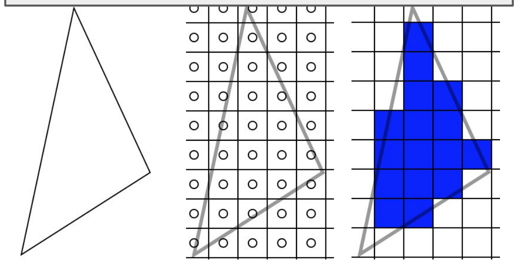
OpenCL™

CUDA, Vulkan или OpenCL?

Запуск везде:
AMD, NVidia, Intel, Snapdragon, …
Windows, Linux, Mac, Android

MoltenVK

Растеризация, depth-test,
color blending



NVIDIA CUDA

Vulkan

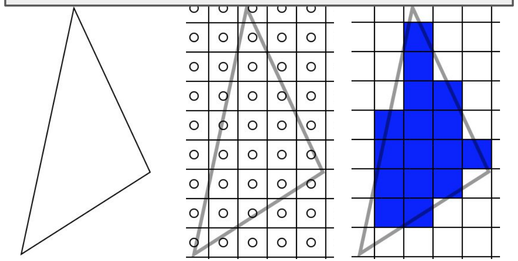
OpenCL™

CUDA, Vulkan или OpenCL?

Запуск везде:
AMD, NVidia, Intel, Snapdragon, …
Windows, Linux, Mac, Android

MoltenVK

Растеризация, depth-test,
color blending



GPGPU: API и синтаксис
кернела проще чем в **Vulkan**

NVIDIA CUDA

Vulkan

OpenCL™

CUDA, Vulkan или OpenCL?

Запуск везде:
AMD, NVidia, Intel, Snapdragon, …
Windows, Linux, Mac, Android

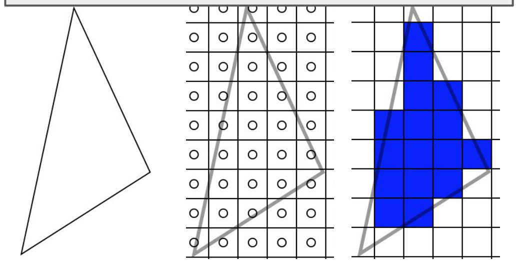
MoltenVK

Тулинг: профилировщики, санитайзеры (cuda-memcheck/racecheck)

NVIDIA Nsight

NVIDIA CUDA

Растеризация, depth-test, color blending



Vulkan

GPGPU: API и синтаксис кернела проще чем в **Vulkan**

OpenCL™

CUDA, Vulkan или OpenCL?

Тулинг: профилировщики, санитайзеры (cuda-memcheck/racecheck)

NVIDIA Nsight

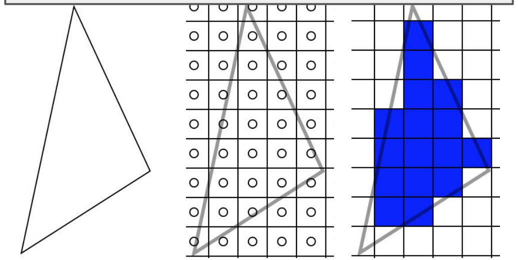
NVIDIA CUDA

Тулинг:
Validation Layers,
SPIRV-Reflect,
RenderDoc,
профайлеры

Запуск везде:
AMD, NVidia, Intel, Snapdragon, …
Windows, Linux, Mac, Android

MoltenVK

Растеризация, depth-test, color blending



Vulkan

GPGPU: API и синтаксис кернела проще чем в **Vulkan**

OpenCL™

CUDA, Vulkan или OpenCL?

AI/ML: **очень** хорошо оптимизированные **библиотеки**, **профилировщики**

cuDNN

Тулинг: профилировщики, санитайзеры (cuda-memcheck/racecheck)

NVIDIA Nsight

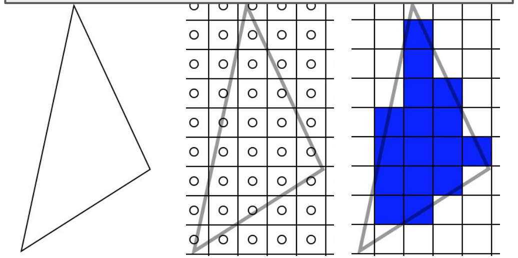
NVIDIA CUDA

Тулинг:
Validation Layers,
SPIRV-Reflect,
RenderDoc,
профайлеры

Запуск везде:
AMD, NVidia, Intel, Snapdragon, …
Windows, Linux, Mac, Android

MoltenVK

Растеризация, depth-test, color blending



Vulkan

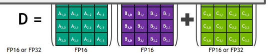
GPGPU: API и синтаксис кернела проще чем в **Vulkan**

OpenCL™

CUDA, Vulkan или OpenCL?

Low-Level NVIDIA HW:
Tensor Cores
Ray Tracing Cores

NVIDIA. OPTIX



AI/ML: **очень** хорошо оптимизированные **библиотеки**, **профилировщики**

cuDNN

Тулинг: профилировщики, санитайзеры (cuda-memcheck/racecheck)

NVIDIA Nsight

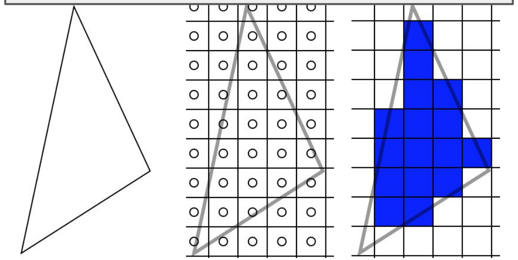
NVIDIA CUDA

Запуск везде:
AMD, NVidia, Intel, Snapdragon, …
Windows, Linux, Mac, Android

MoltenVK

Тулинг:
Validation Layers,
SPIRV-Reflect,
RenderDoc,
профайлеры

Растеризация, depth-test, color blending



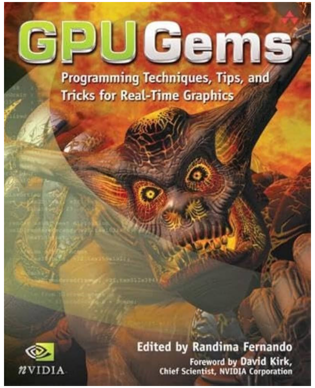
Vulkan

GPGPU: API и синтаксис кернела проще чем в **Vulkan**

OpenCL™

Дополнительные материалы

1) **Source Code** - https://github.com/GPGPUCourse/GPGPUVulkan
2) **Книги/циклы статей - GPU Gems**
3) **Конференции - SIGGRAPH**

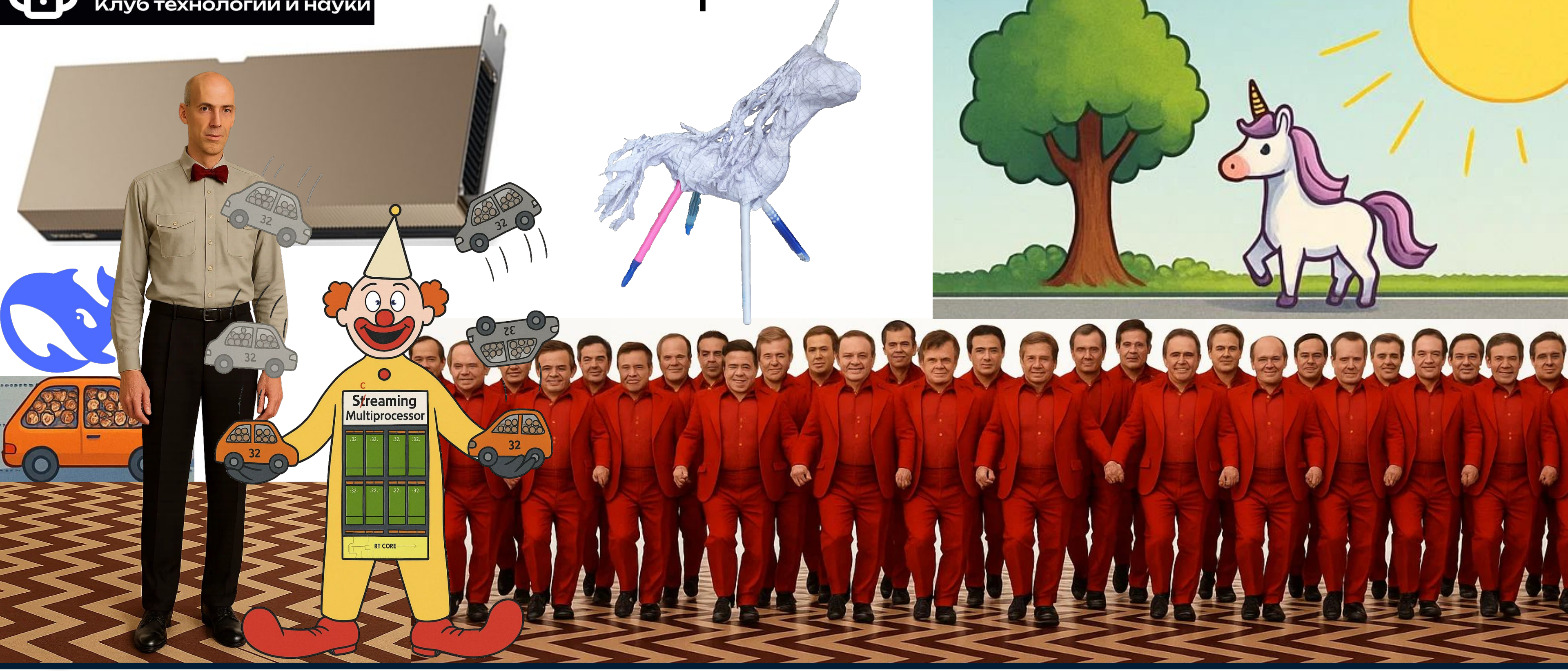


4) **Как работает Nanite в Unreal Engine 5** - https://youtu.be/ltUzX1IR9JI
5) **Real time BVH construction**
6) **Курс по видеокартам** - https://youtu.be/LDt4KQEdImY
7) **Курс фотограмметрии** - https://youtu.be/rEF0zkv2cn8


Вопросы?

CS Space
Клуб технологий и науки

Вопросы?



@PolarNick239 ⟵ он рад обсудить интересное!
polarnick239@gmail.com
Николай Полярный

Agisoft Metashape

Vulkan OpenCL™ NVIDIA CUDA

Вопросы?

Вопросы?

Вопросы?


Вопросы?