



# Модель вычислений CUDA

# О чём сегодня пойдет речь?



Затрагиваемая тема очень сложная  
Вопрос, приготовленный вам на экзамен,  
тоже сложный  
Однако вся необходимая для Вас  
информация располагается по этому QR коду  
**Обязательно сканируйте QR-код, особенно,  
если вы заинтересуетесь данной темой.**

Где вы можете сегодня встретиться с CUDA?

# Где вы можете сегодня встретиться с CUDA?

Здесь



**WOLFRAM MATHEMATICA**  
The world's definitive system for modern technical computing

**Math. Graphics. Programming.**

MATLAB is a programming and numeric computing platform used by millions of engineers and scientists to analyze data, develop algorithms, and create models.

[Get MATLAB](#)   [Request a quote](#)

WolframAlpha.com | WolframCloud.com | All Sites & Public Resources...  
 Products & Services ▾ Technologies ▾ Solutions ▾ Learning & Support ▾ Company ▾ Q Search

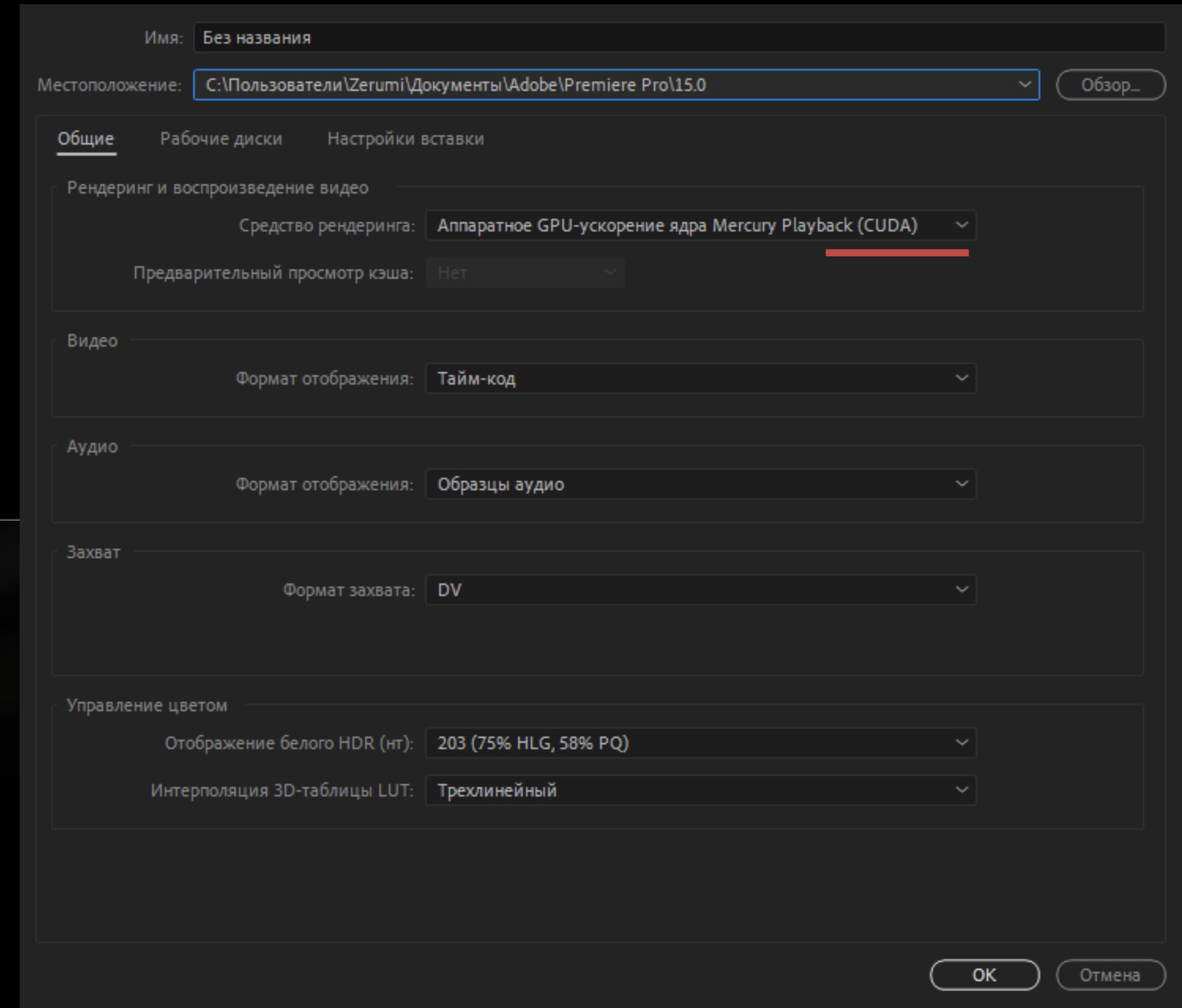
Available on desktop, cloud & mobile

Get MATLAB

What Is MATLAB?

# Где вы можете сегодня встретиться с CUDA?

Здесь



# Где вы можете сегодня встретиться с CUDA?

Здесь



The screenshot shows the Microsoft Learn Windows documentation page. At the top, there's a navigation bar with links for Learn, Discover, Product documentation, Development languages, and Topics. Below that is a secondary navigation bar with links for Windows, Release health, Windows client, Application developers, Hardware developers, Windows Server, Windows for IoT, Windows Insider Program, and Windows 365. A search bar and sign-in link are also present.

The main content area features a large title "Get started with GPU acceleration for ML in WSL". Below it is a sub-headline "In this article". A sidebar on the left lists various topics under "WSL Documentation", including Overview, Install, Tutorials (with "Set up GPU acceleration (NVIDIA CUDA/DirectML)" highlighted), Concepts, How-to, Enterprise security, Frequently Asked Questions, Troubleshooting, and Release Notes. The central content area contains the article text, which discusses the benefits of using GPU acceleration in WSL for machine learning and provides instructions on how to set it up.

# Где вы можете сегодня встретиться с CUDA?

И даже здесь



The screenshot displays the Rhino Health website. At the top, there's a navigation bar with links to Home, Platform, Solutions, Partners, Media, and About. Below the navigation, a section titled "Key features of the Rhino FCP include" lists six features with corresponding icons:

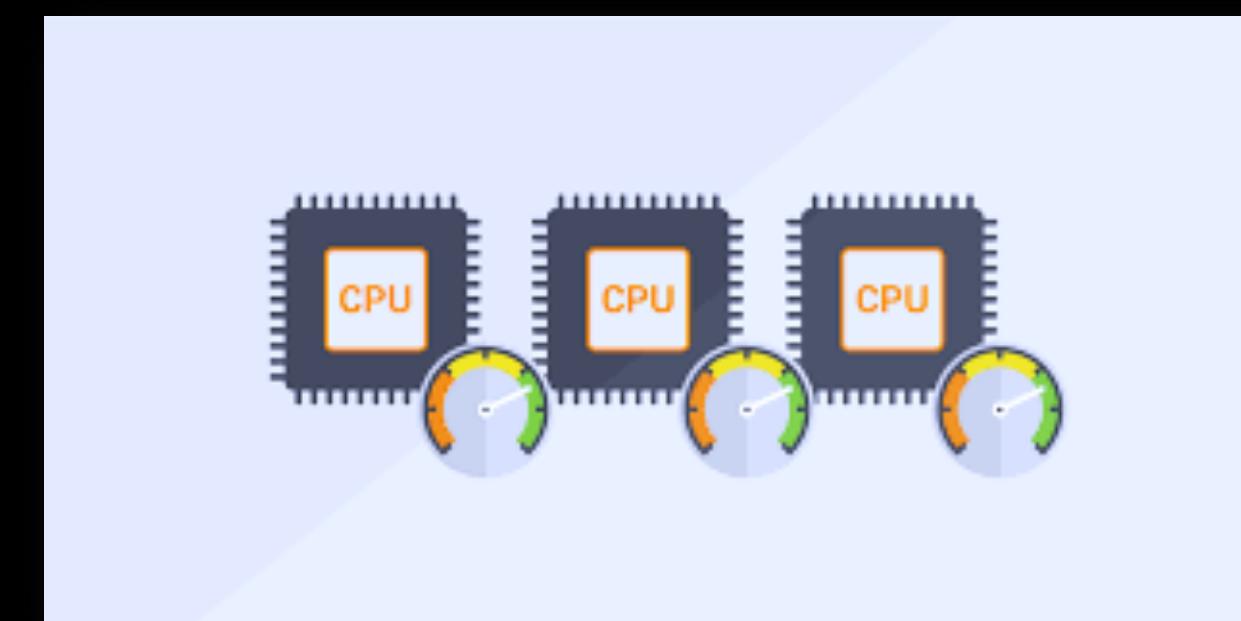
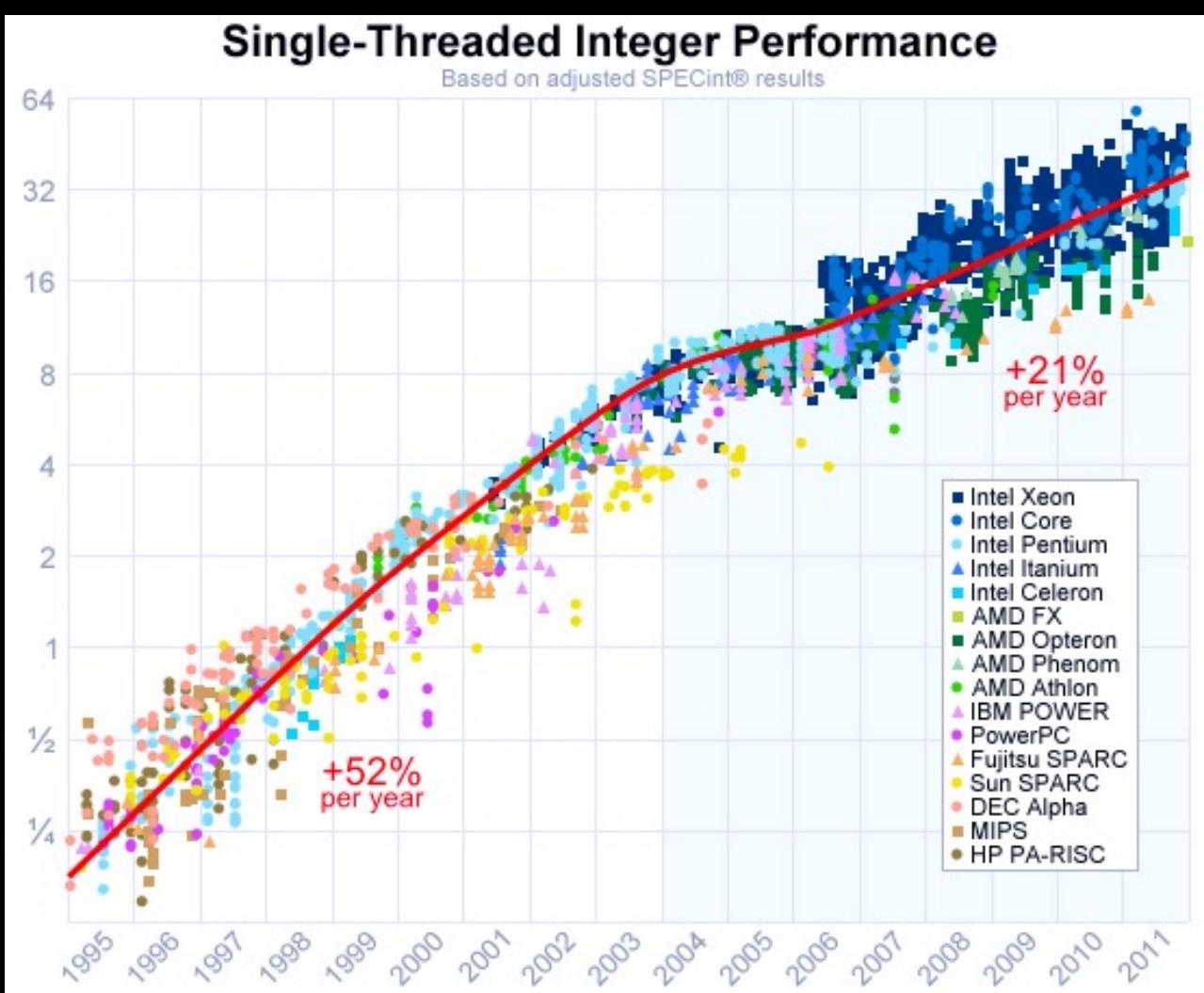
- Data validation - Auto-identification of data, generating reusable templates ('schemas') that can be used for validation and versioning
- Generalized Compute (GC) - Run any custom code on data, 'on the edge'
- iGC - Interactive with a broad range of softwares (including GUI), without the need for lengthy integrations
- Secure Access - Integrated zero-footprint data and image viewer / annotation tool
- Analytics Toolbox - exploratory data analytics, and a range of epidemiological and statistical analyses, including results visualization
- GUI / SDK - Access via a web-interface and a programmatic interface

On the right side of the page, there are two medical images: a 3D brain scan with a yellow circle highlighting a specific area, and a 2D brain scan with a yellow circle and a green outline. Below these images, a large call-to-action button says "Build Safe and Effective Medical Devices and Healthcare Software". A subtext below it reads: "Tame the complexity of your medical digital device development with comprehensive cross-platform tools targeted to each stage of the software delivery and maintenance lifecycle." At the bottom of the page, there's a "Explore Qt Development Framework and Tools" button.

# Что не так с CPU? Зачем нам понадобилась еще одна платформа?

Всему виной  
производительность

Вычисления должны  
выполняться еще  
быстрее...

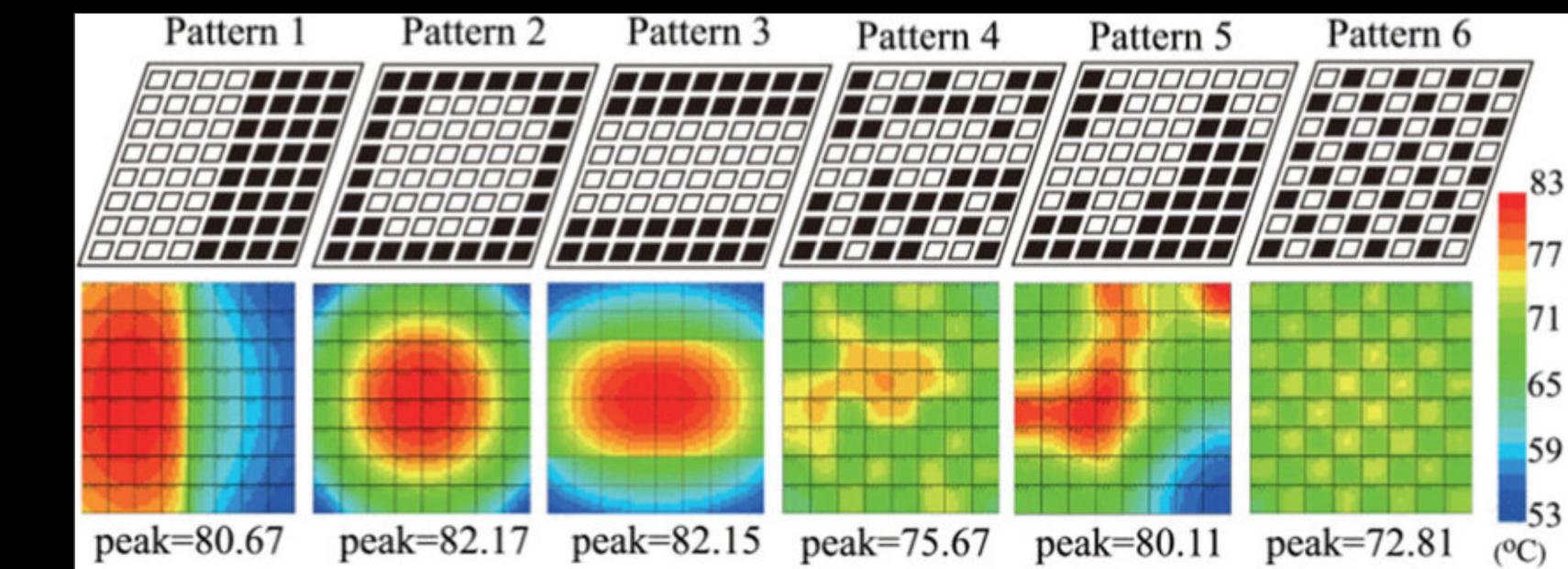
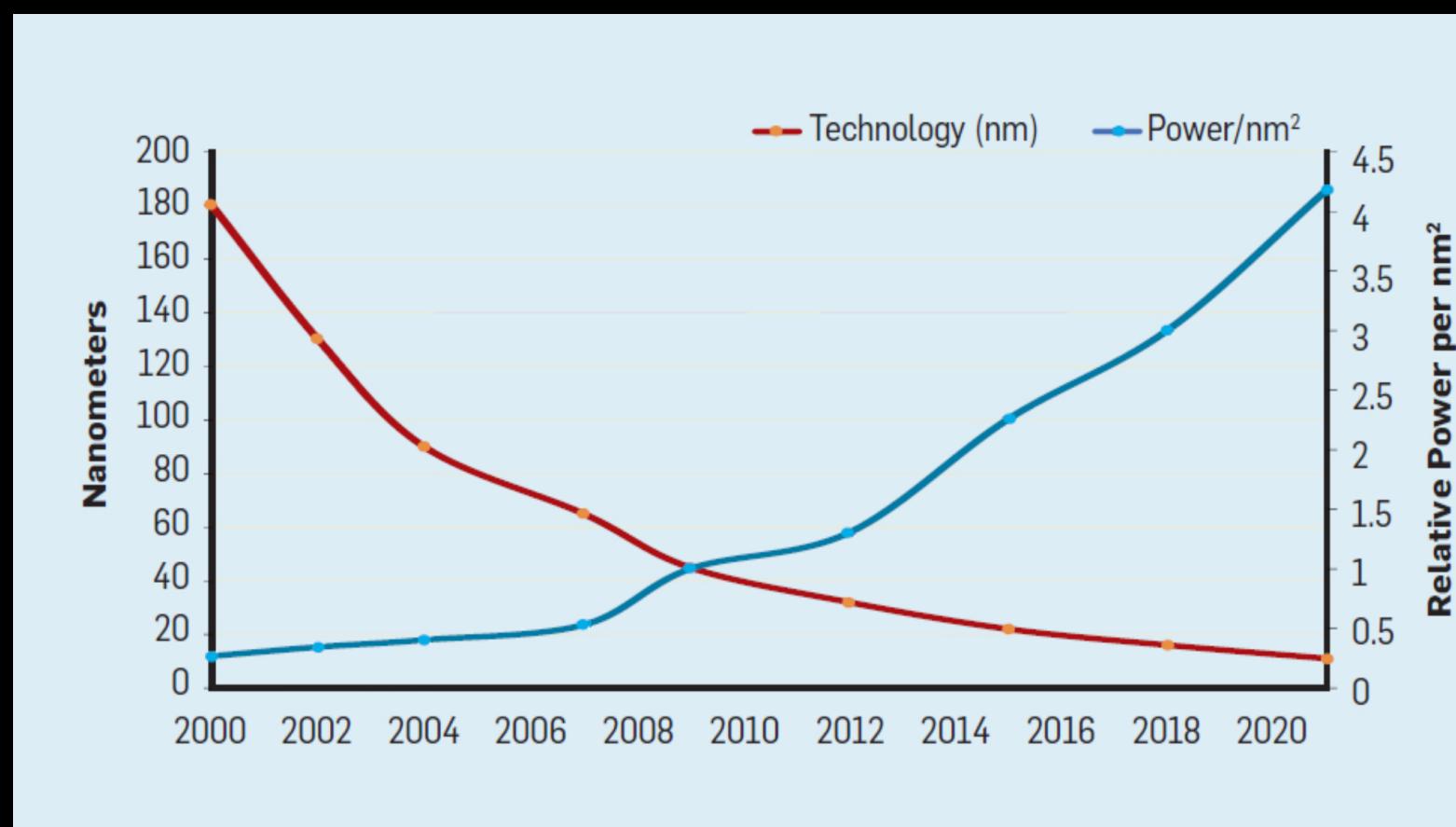


# Что не так с CPU? Зачем нам понадобилась еще одна платформа?

Как же нам быть?

Поднять тактовую  
частоту не вариант

Бесконечный рост  
транзисторов тоже  
невозможен

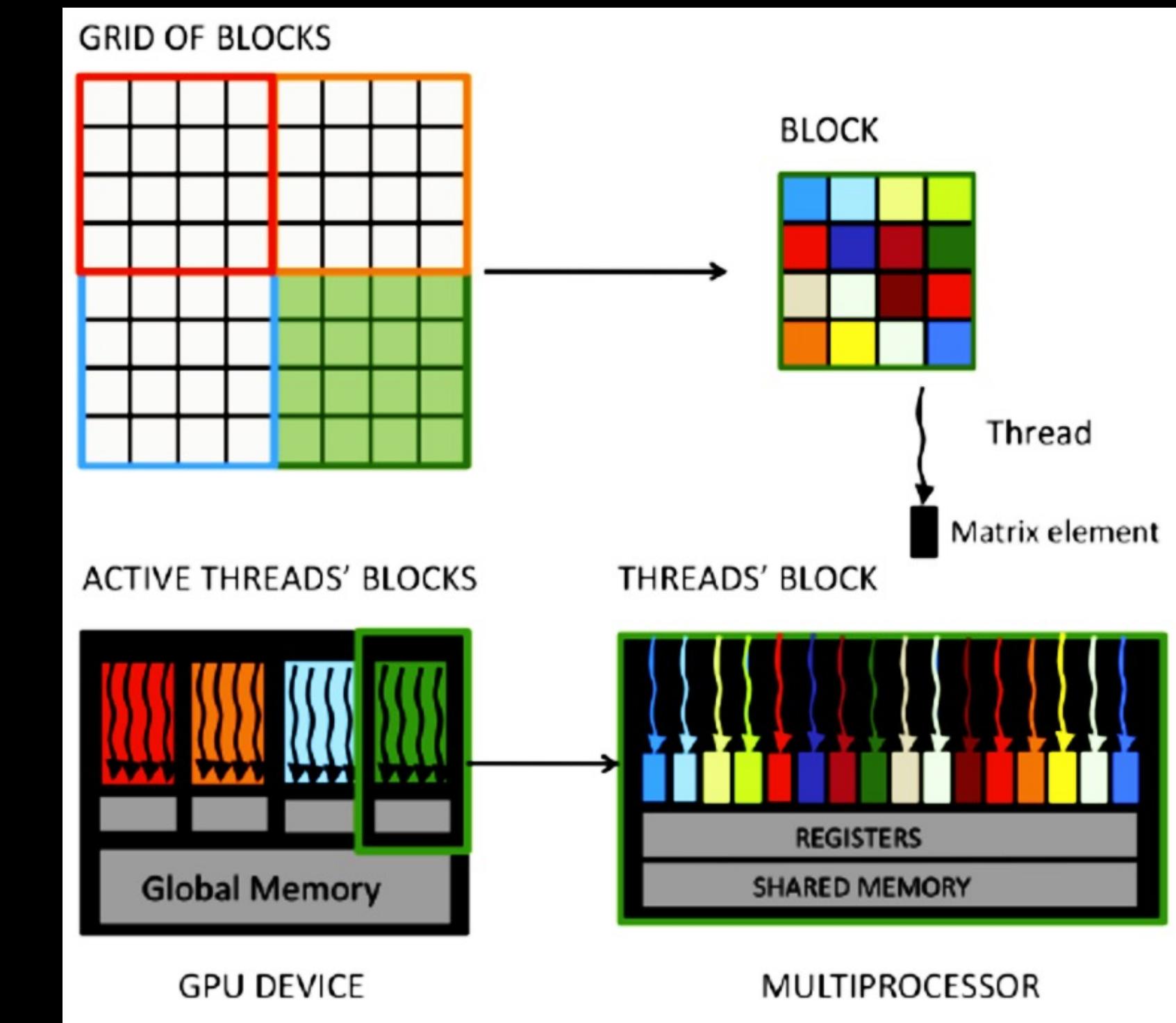


8\*8 Alpha 21264 core array; 22 nm technique; @2.6Ghz, 64GB Memory  
Dark Silicon on the chip is 50%; McPAT [34]; Hotspot v5.02 [35]; Tsafe = 81.8 °C [26]

# Что не так с CPU? Зачем нам понадобилась еще одна платформа?

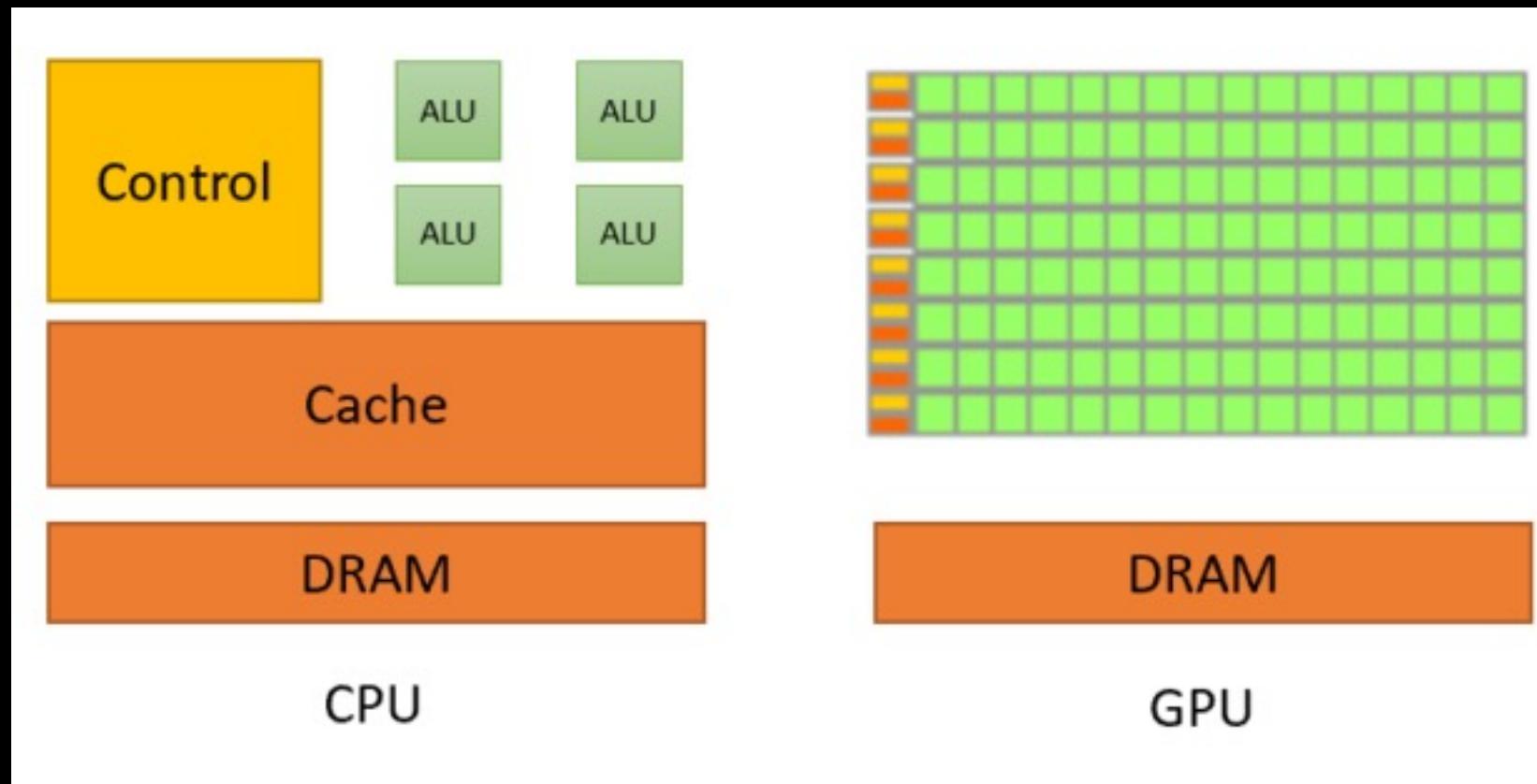
На помощь приходит GPU

Их архитектура изначально подразумевает независимую параллельную обработку множества данных

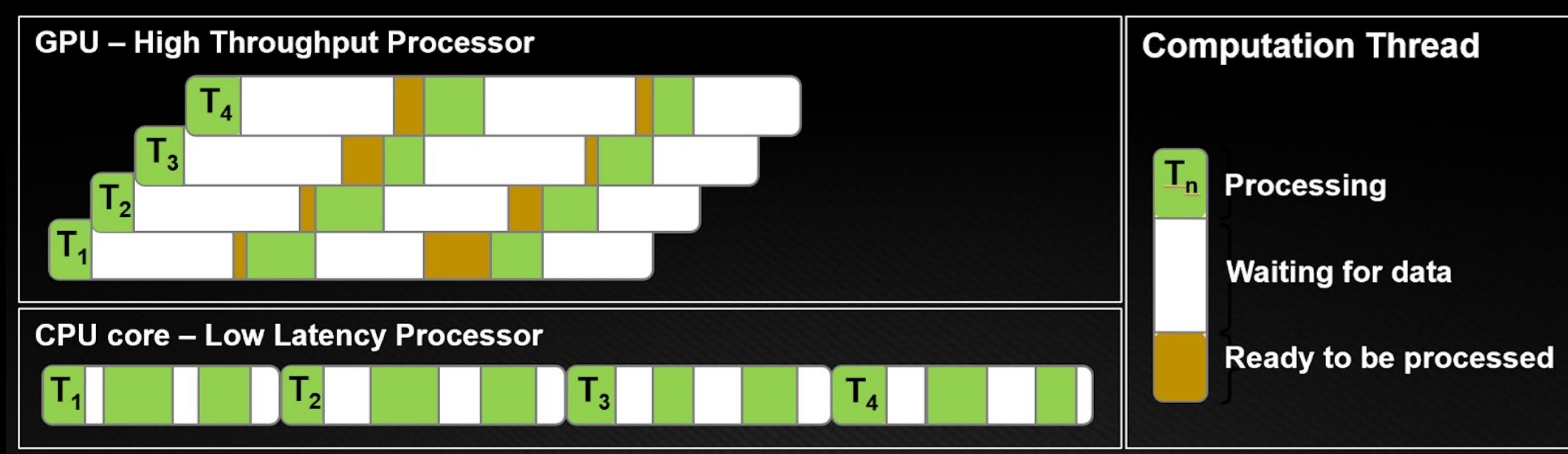
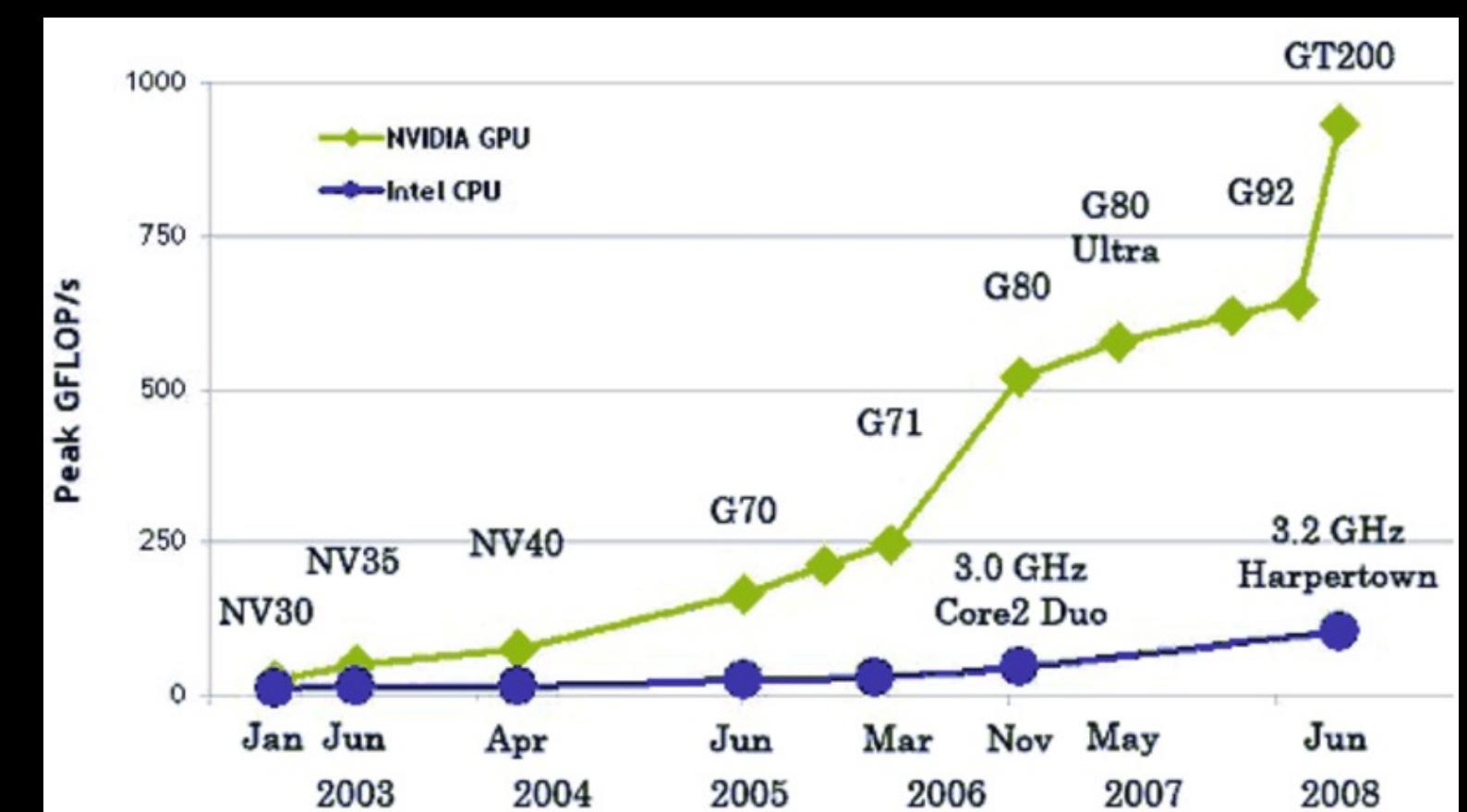


# Что не так с CPU? Зачем нам понадобилась еще одна платформа?

**CPU - медленно**

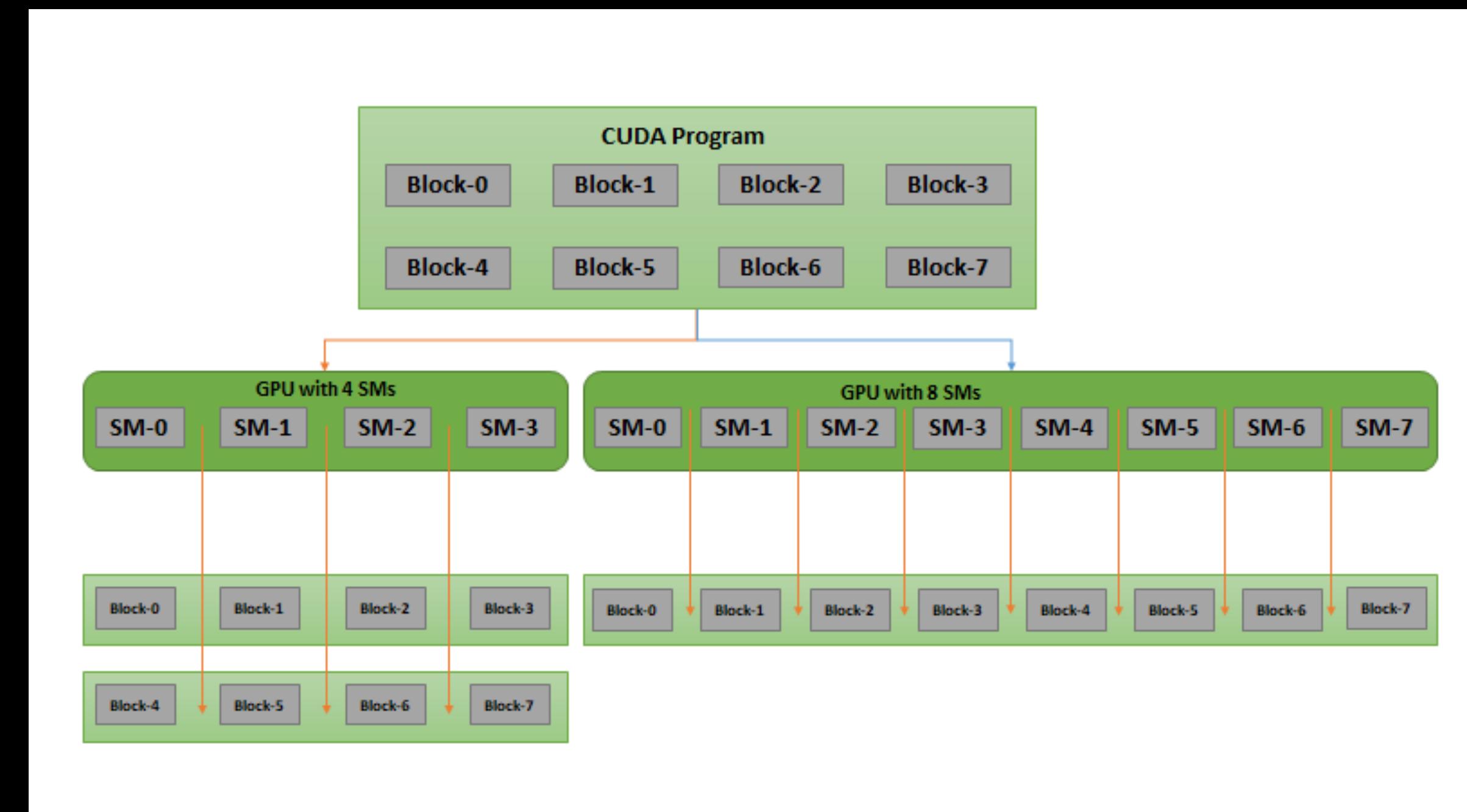


**GPU – быстро\***



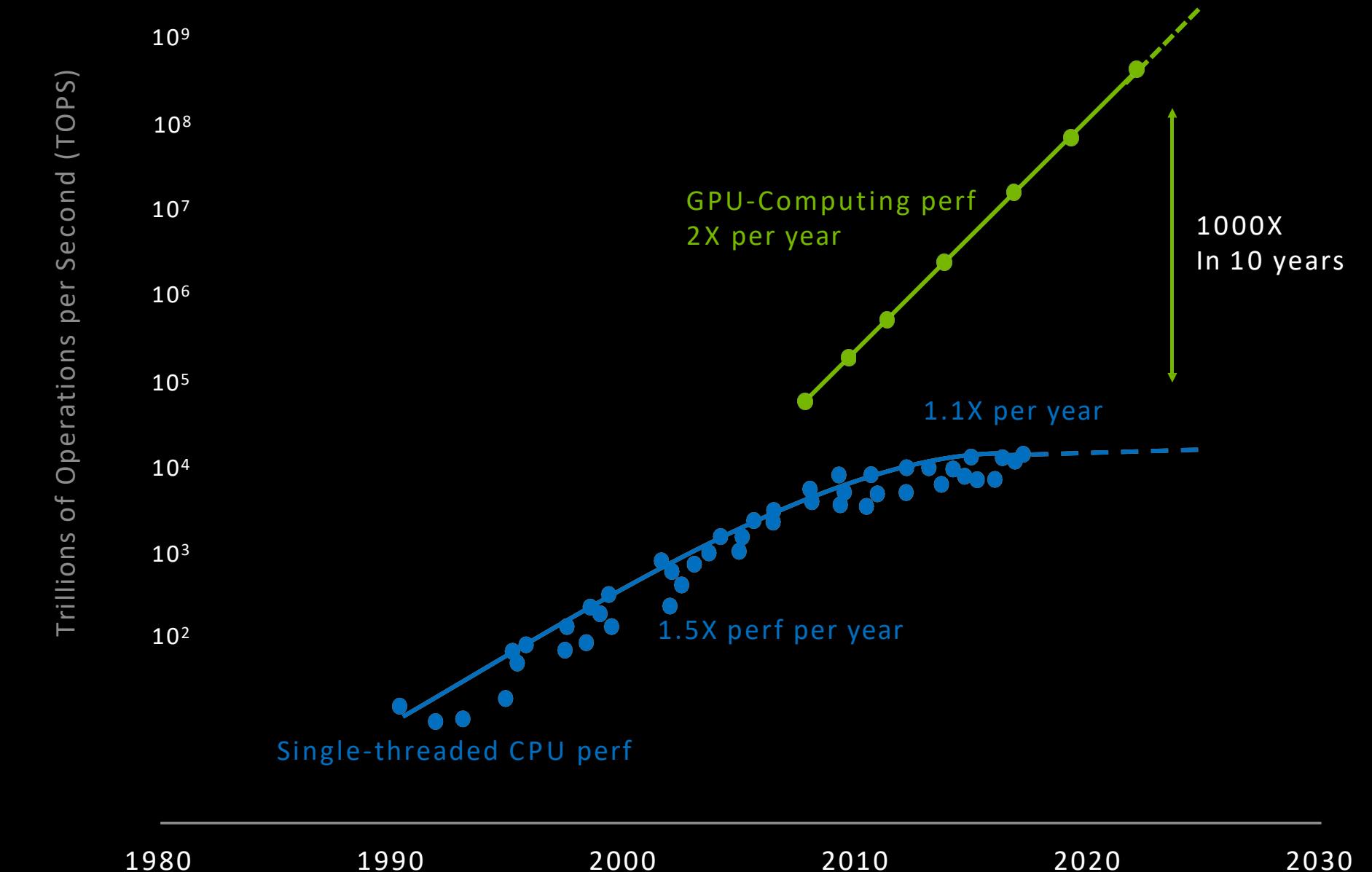
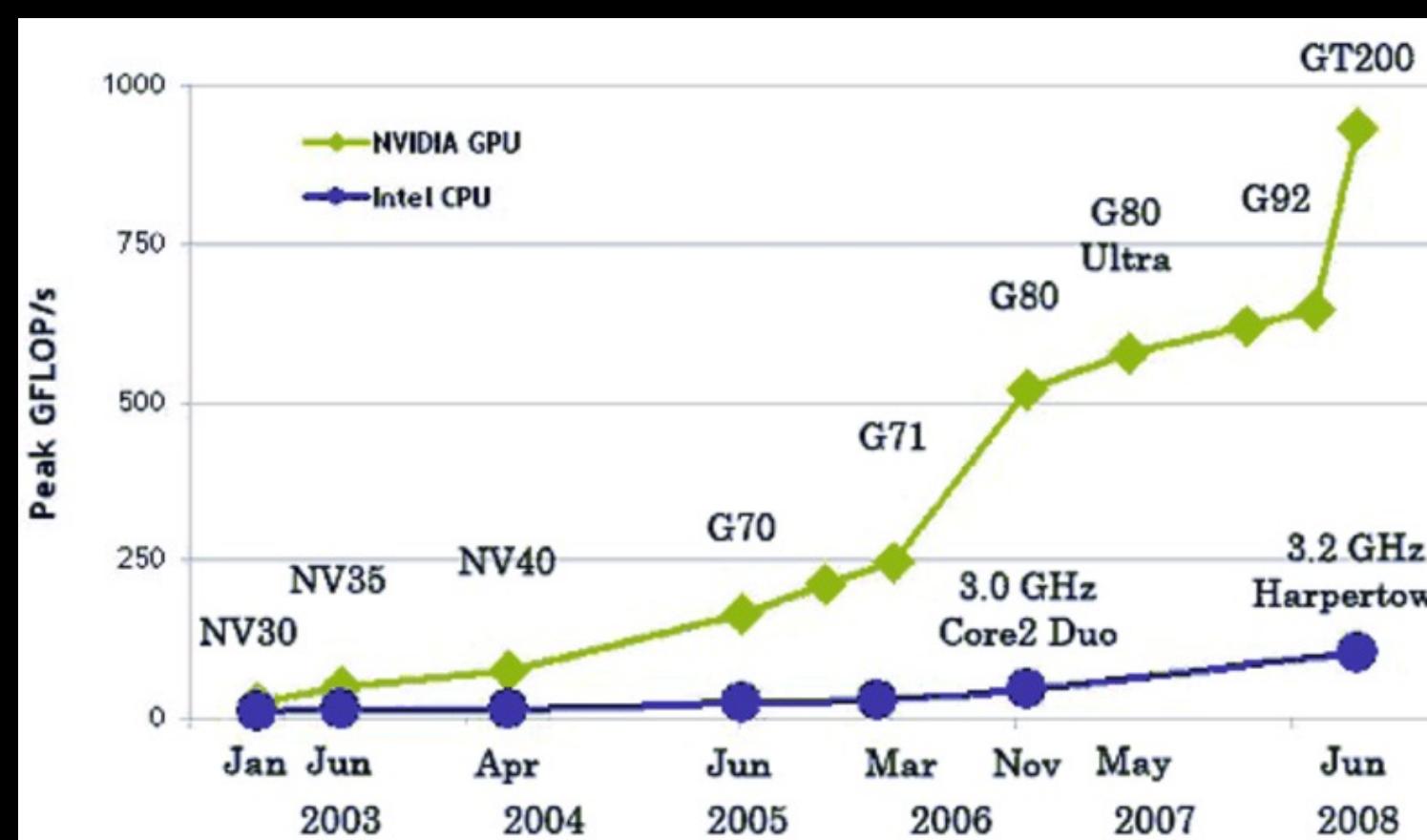
# Что не так с CPU? Зачем нам понадобилась еще одна платформа?

GPU не имеют проблем с масштабируемостью



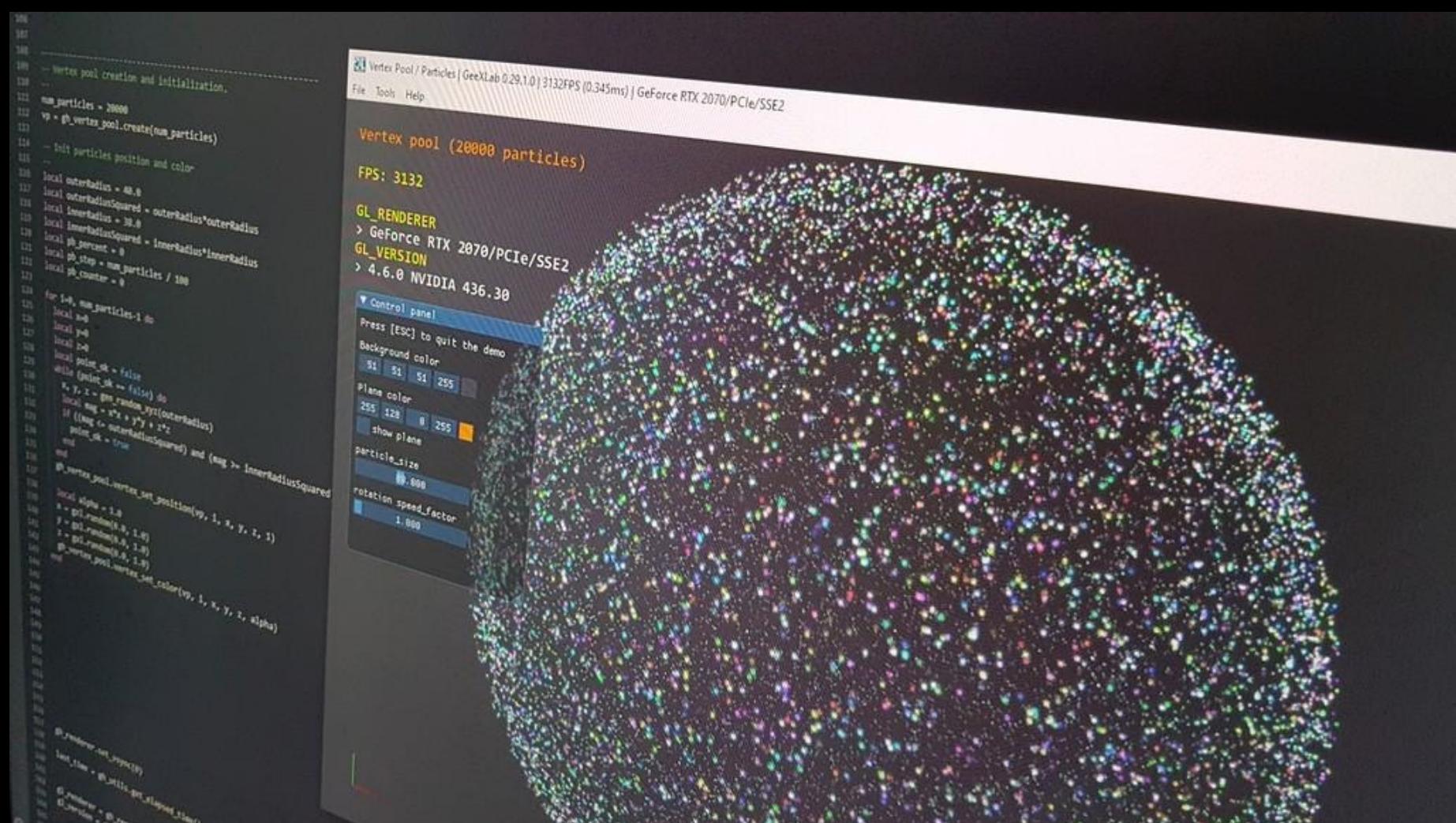
# Что не так с CPU? Зачем нам понадобилась еще одна платформа?

**GPU обгоняет по скорости вычислений CPU**



Но если бы все было так радужно...

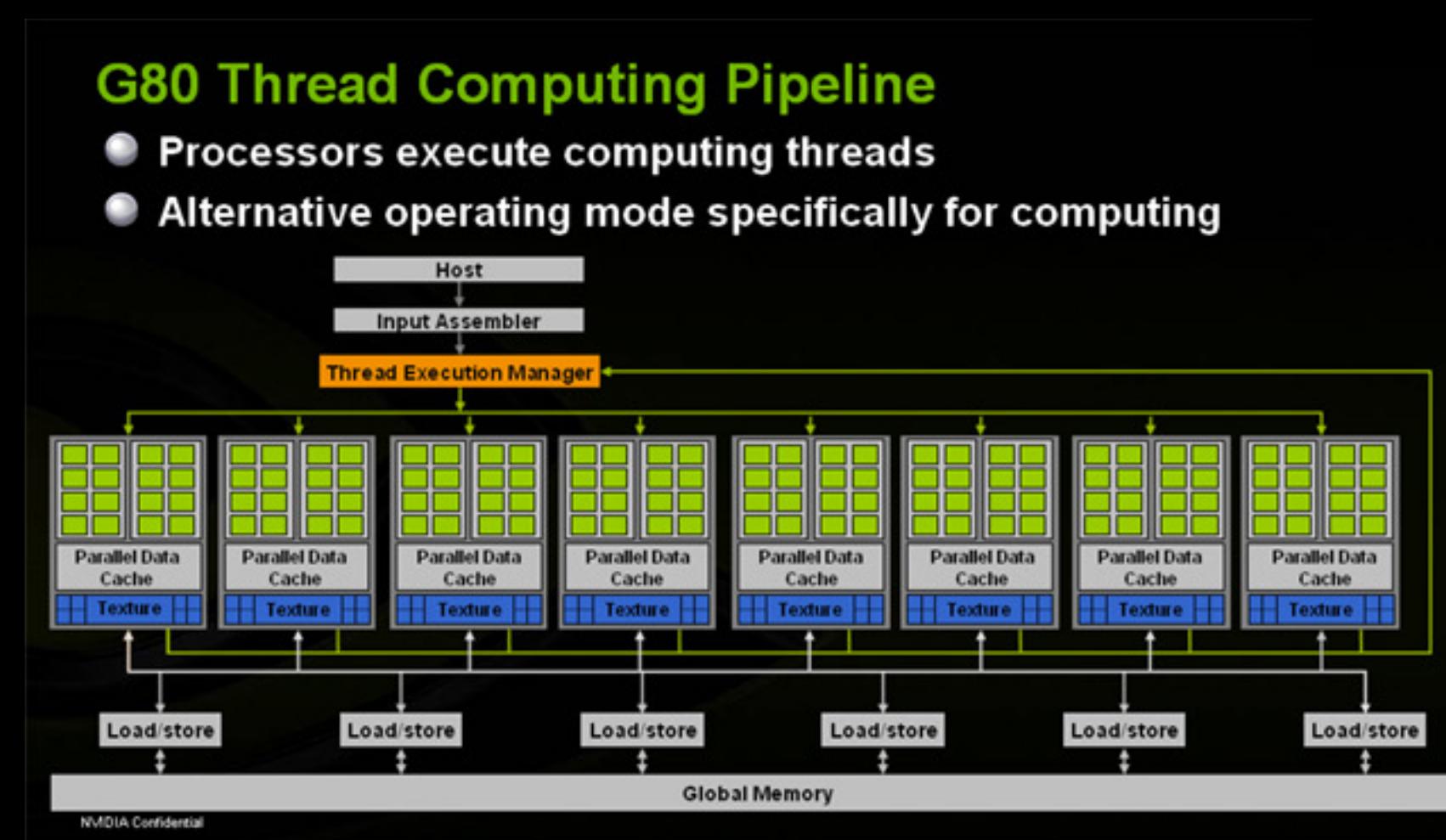
# Краткая историческая справка



До этого GPU не использовался для общих параллельных вычислений

- Однако из-за преимущества архитектуры, подобные вычисления намного эффективнее производить именно на GPU, а не на CPU
- Существовали обходные пути, GPU производили «графические» вычисления -> издержки на «ненужные» операции
- Пример - BrookGPU

# Краткая историческая справка



Все началось в 2006 году с появлением NVIDIA Tesla

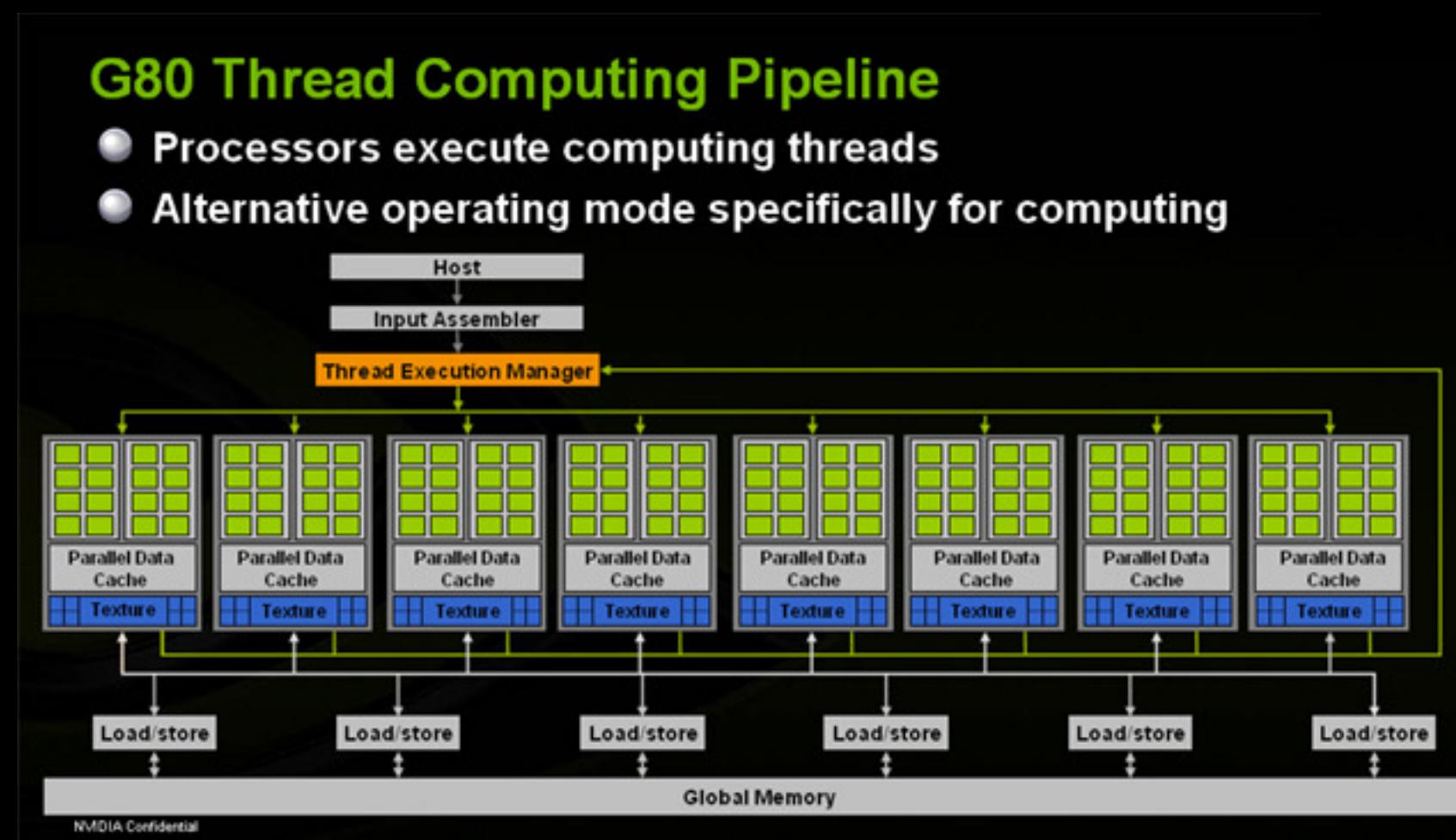
До этого GPU не использовался для общих параллельных вычислений

- После появления BrookGPU, на это обратили внимание в NVIDIA
- Часть разработчиков BrookGPU перешла в NVIDIA

Открытая и унифицированная модель вычислений:

- Позволяла выполнять на видеокарте не только графические вычисления
- Разработчикам был предоставлен компилятор, впоследствии Runtime был включен в драйверы, и написанные программы можно было запускать на любой совместимой видеокарте
- Vendor-lock solution: работает только на GPU NVIDIA.
- Позднее в гонку включится ATI (AMD)

# Краткая историческая справка

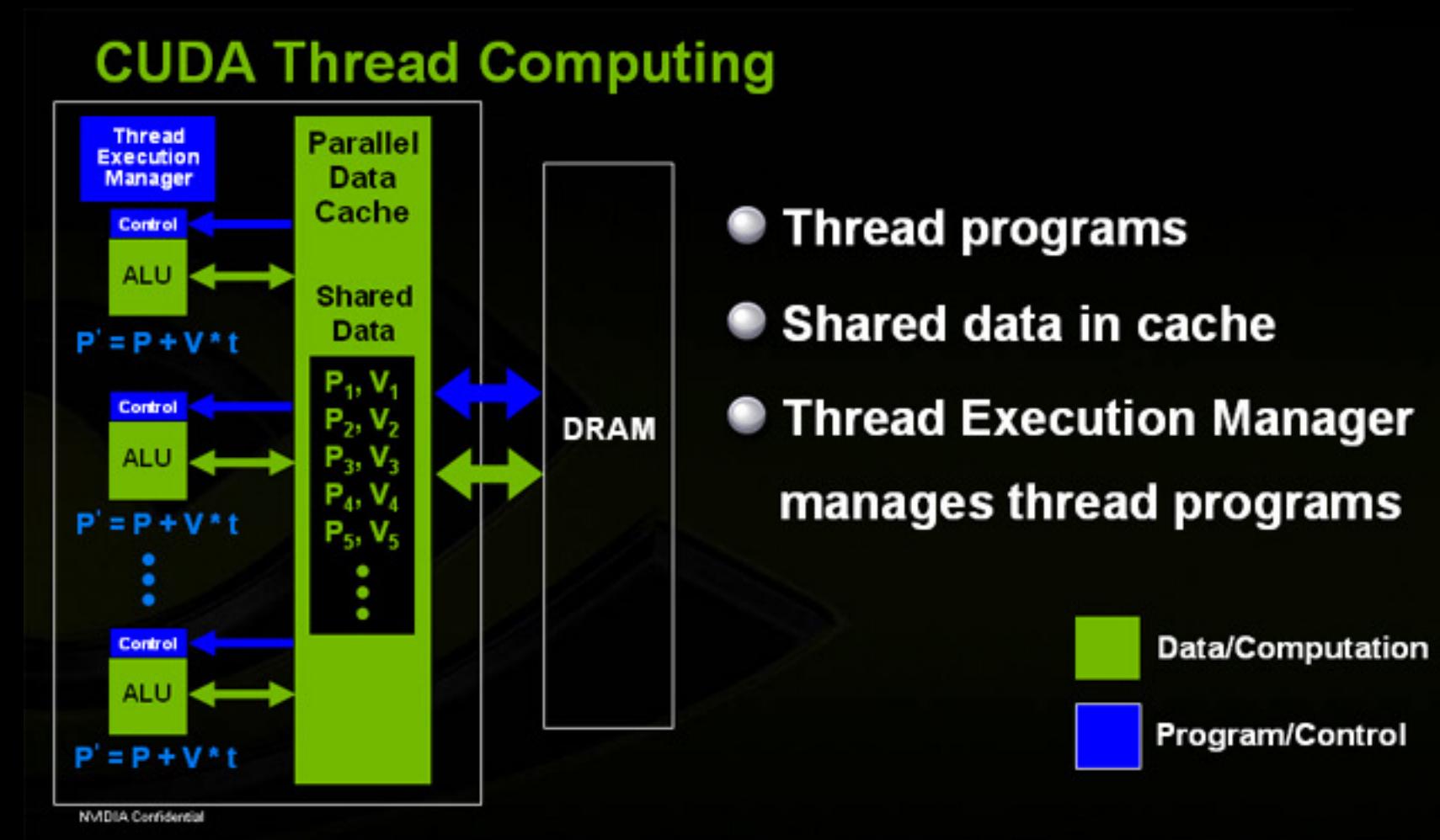


Все началось в 2006 году с появлением NVIDIA Tesla

Стриминговый процессор в GeForce 8 Series:

- Является изолированным устройством, поддерживающим вычисления общего назначения с 32-разрядными числами с плавающей точкой
- Поддерживал эффективную работу с входными и выходными данными
- Объединялся в группу процессоров для поддержки массивных параллельных вычислений
- Управление вычислительным процессором берет на себя планировщик – NVIDIA GigaThread

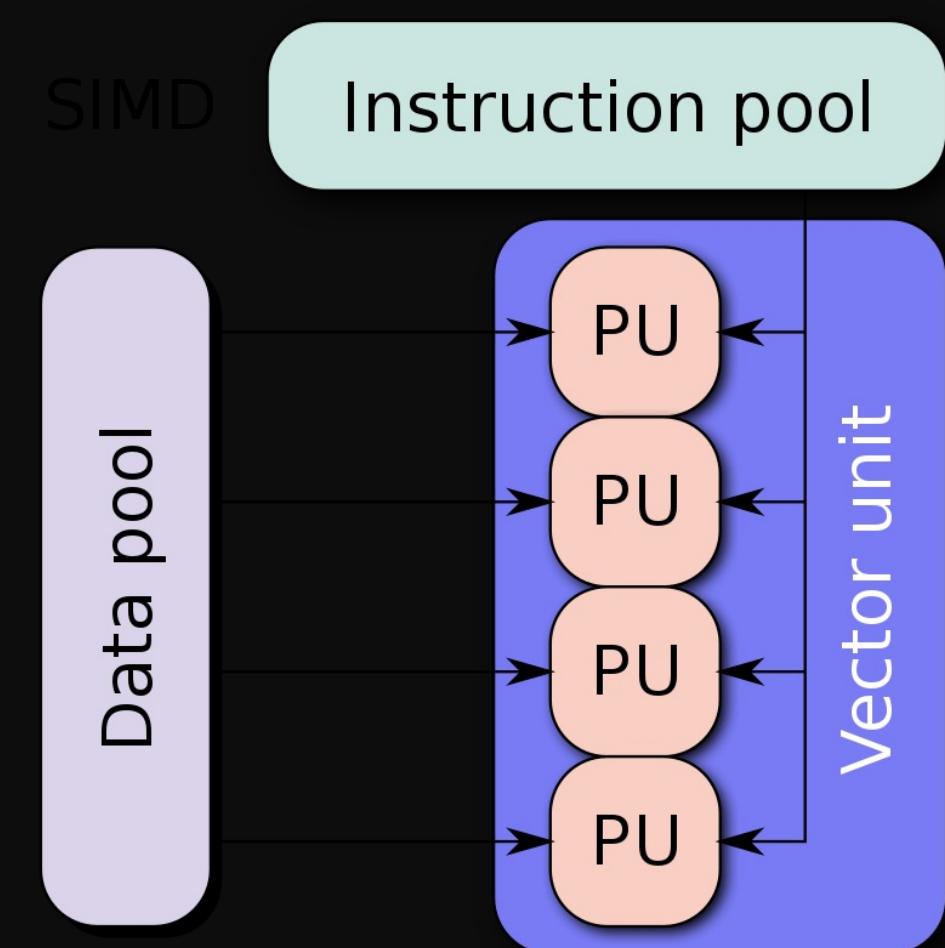
# Краткая историческая справка



Единый набор инструкций – множество потоков данных

Революционная архитектура CUDA:

- Compute Unified Device Architecture – сокращенно CUDA
- Эффективные параллельные вычисления общего назначения
- Начало эры General Purpose GPU
- За основу взята архитектура параллельных вычислений SIMD, слегка доработанная

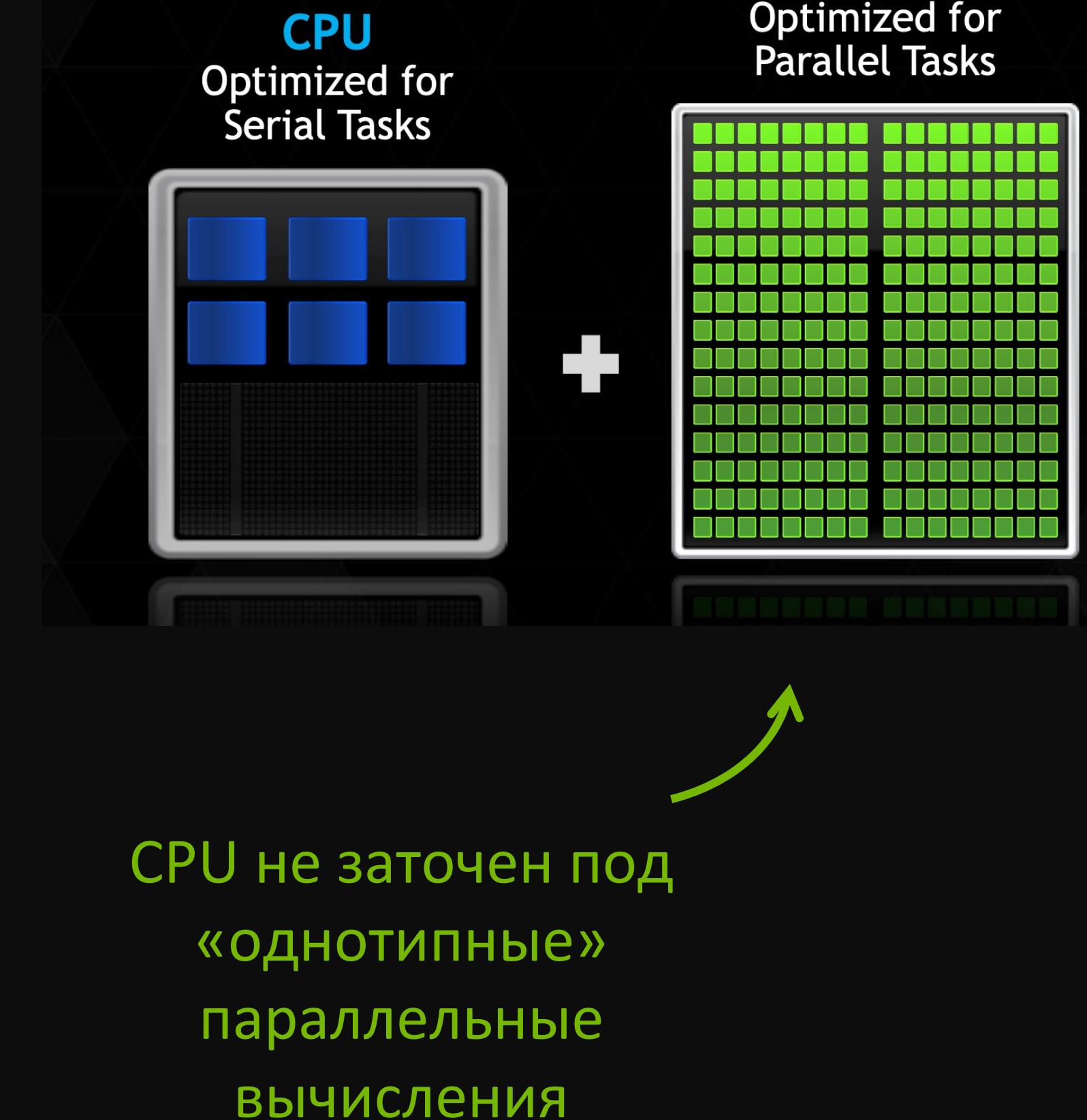


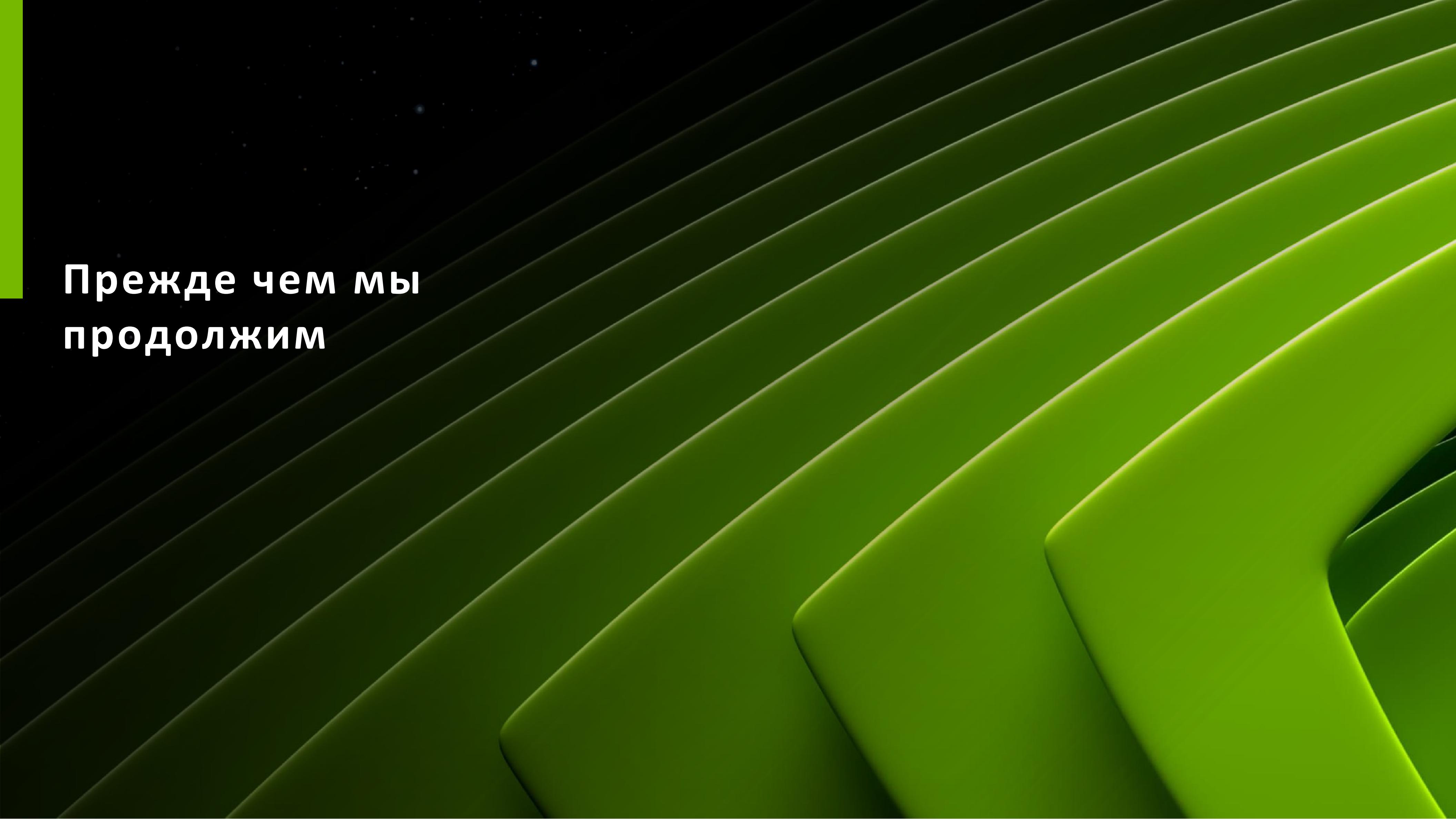
## Почему это помогло?

Мы получили возможность эффективно исполнять вычисления, которые можно распараллелить

Центральный процессор (CPU) изначально не предполагался для параллельных вычислений

Решения не конфликтуют друг с другом: GPU не может заменить CPU и наоборот. Они создавались для разных задач

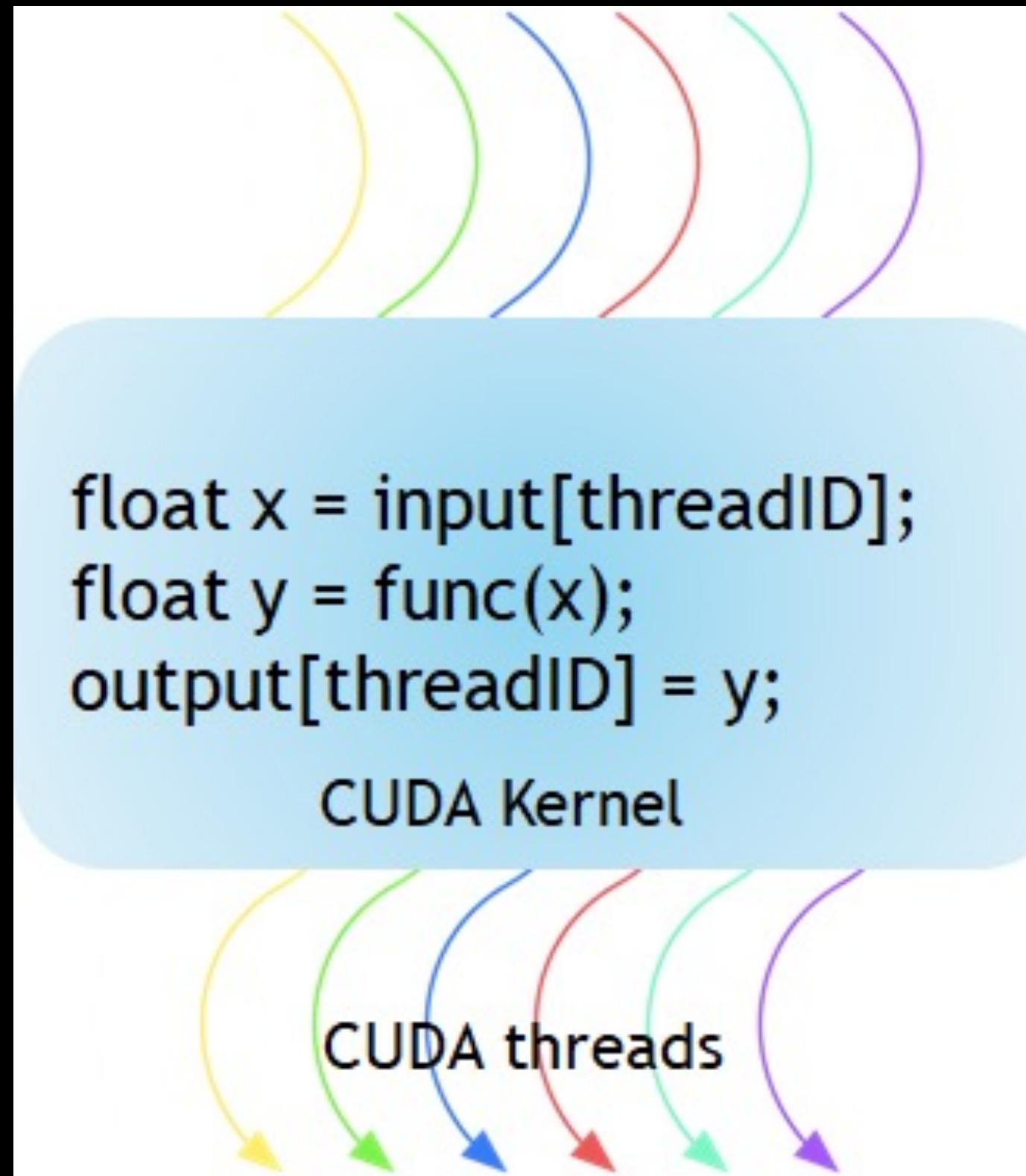




Прежде чем мы  
продолжим

# Разберемся в терминологии

## CUDA Kernel



Функция, выполняющая вычисления, нуждающиеся в распараллеливании

Далее – CUDA Kernel

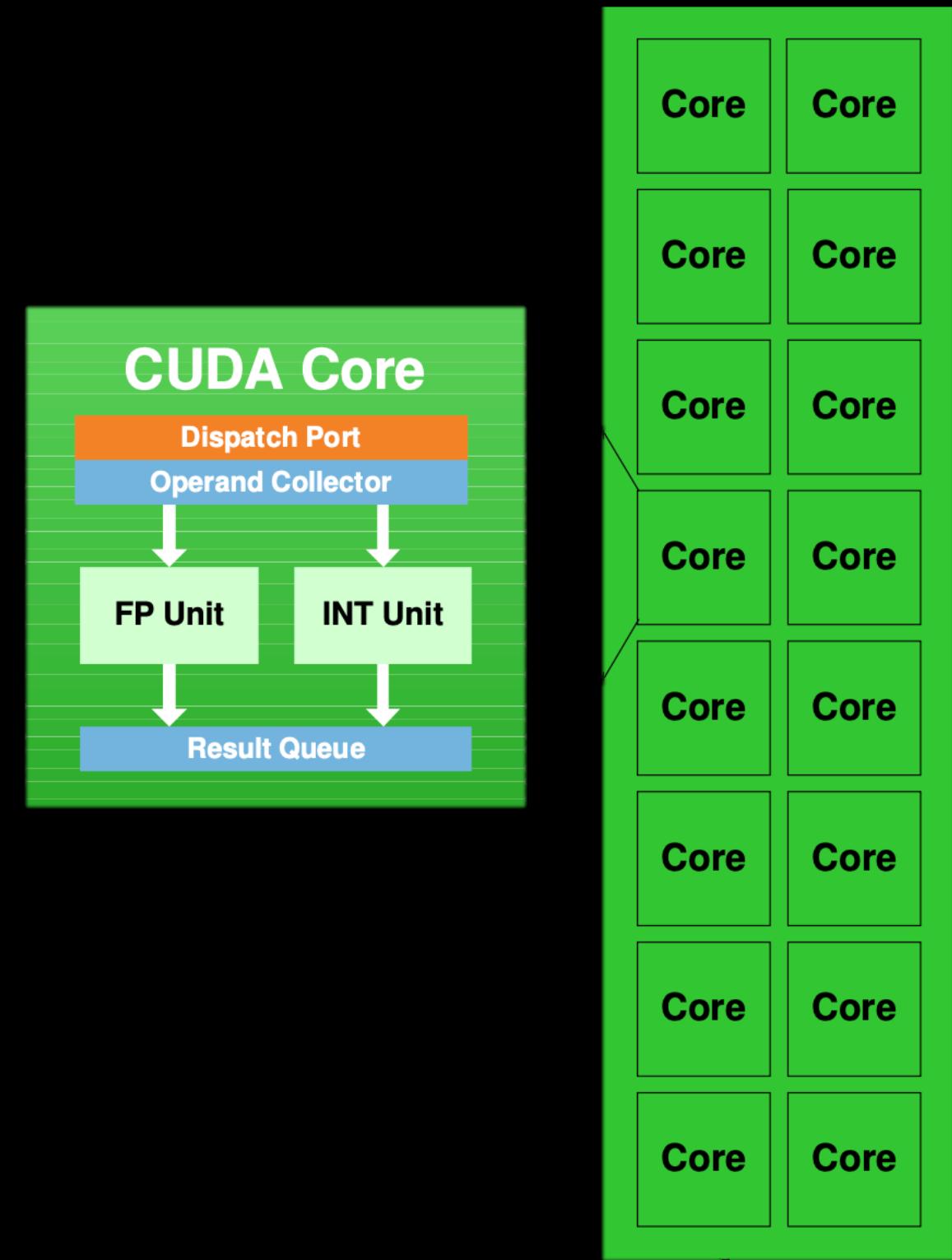
# Разберемся в терминологии

## CUDA Core

Основная боевая единица.  
Исполняет потоки, в которых  
содержится CUDA Kernel

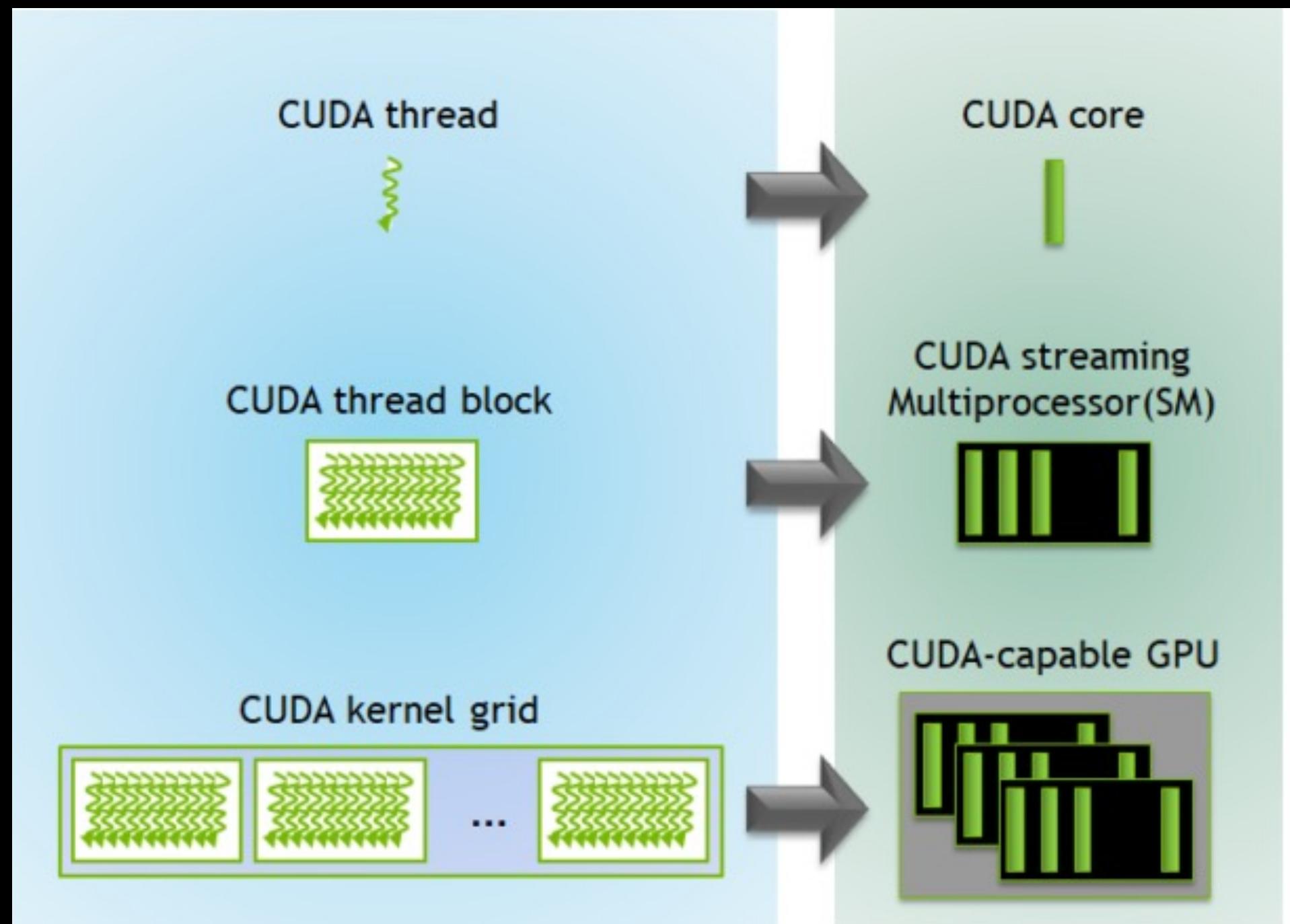
Синоним – CUDA Streaming  
Processor

Ядра формируют архитектуру  
CUDA: Single Instruction,  
Multiple Threads (SIMT). Это и  
есть модификация SIMD, о  
которой упоминалось ранее



# Разберемся в терминологии

## CUDA Thread Block

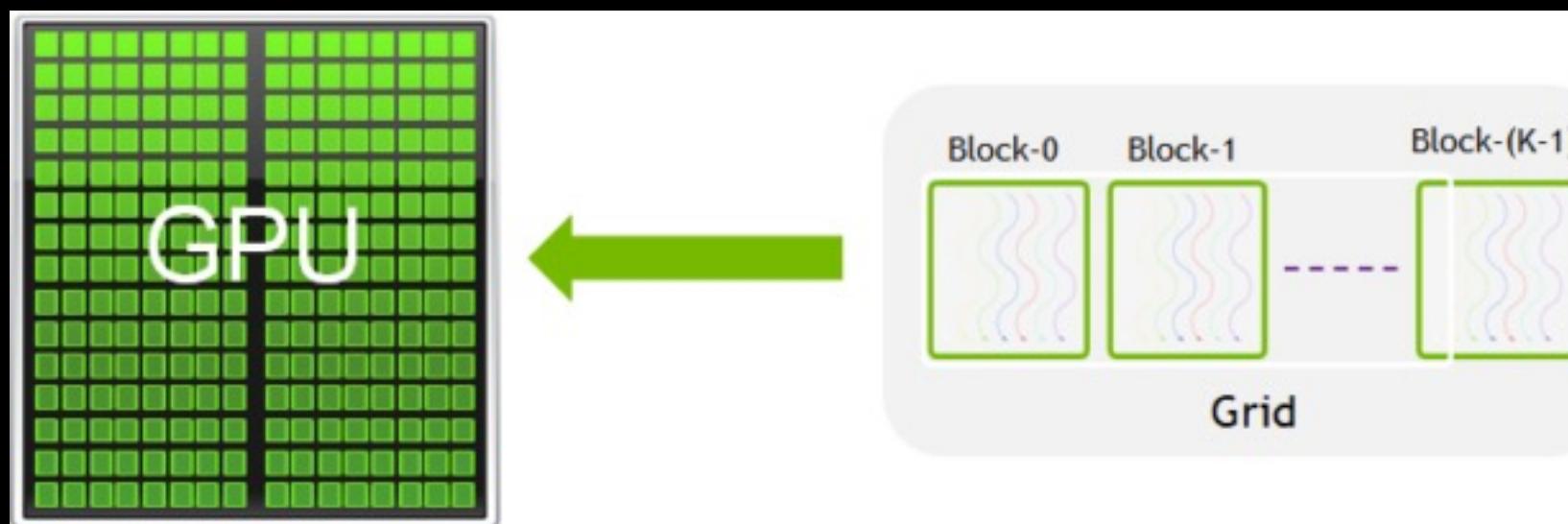


CUDA Kernel'ы объединяются в блоки.

Каждый блок будет выполнен на одном из стриминговых мультипроцессоров (SM)

# Разберемся в терминологии

## CUDA Thread Grid



CUDA блоки загоняются в сетку, которая уже далее может быть исполнена на GPU.

# Что доступно Вам как разработчику?

## CUDA Toolkit

В нем есть **все** что вам нужно:

- Компилятор и CUDA Runtime
- Высокоуровневые библиотеки
- Средства разработки
- Прикладная документация
- Множество учебных материалов
- И все это бесплатно

## CUDA Toolkit 12.4 Downloads

**Select Target Platform**

Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the [CUDA EULA](#).

<b>Operating System</b>	Linux	Windows	
<b>Architecture</b>	x86_64		
<b>Version</b>	10	11	Server 2022

## Resources

- [CUDA Documentation/Release Notes](#)
- [MacOS Tools](#)
- [Training](#)
- [Sample Code](#)
- [Forums](#)
- [Archive of Previous CUDA Releases](#)
- [FAQ](#)
- [Open Source Packages](#)
- [Submit a Bug](#)
- [Tarball and Zip Archive Deliverables](#)



Как происходят вычисления

# Модель организации памяти в CUDA-совместимых GPU

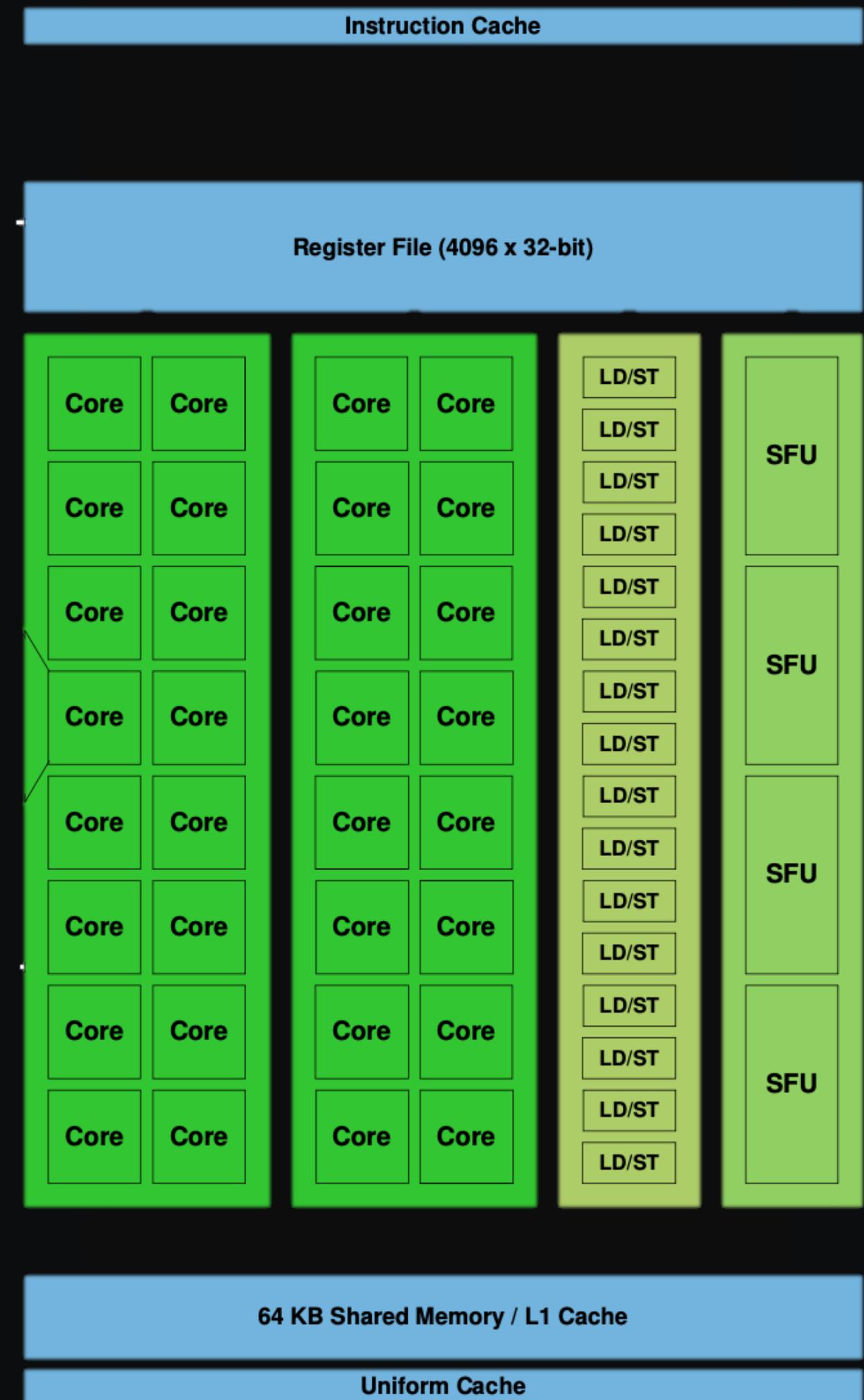
Обратите внимание!

Это может являться вашим вопросом на экзамене

# Модель организации памяти

SM включает в себя следующие элементы памяти:

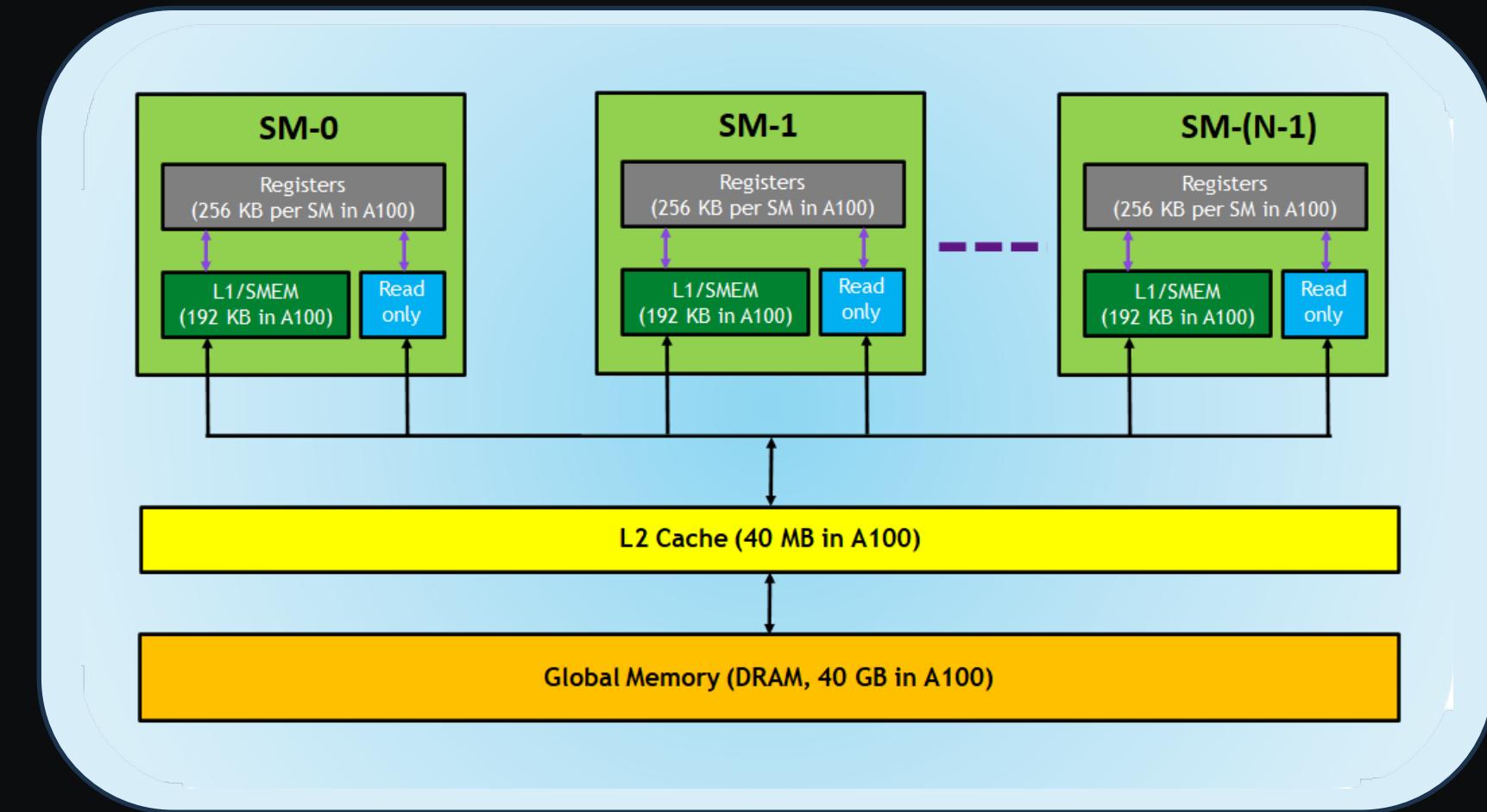
- Блок регистров
- Разделяемая память / кэш первого уровня
- Кэш инструкции, универсальный кэш, и пр.



# Модель организации памяти

За пределами SM располагается:

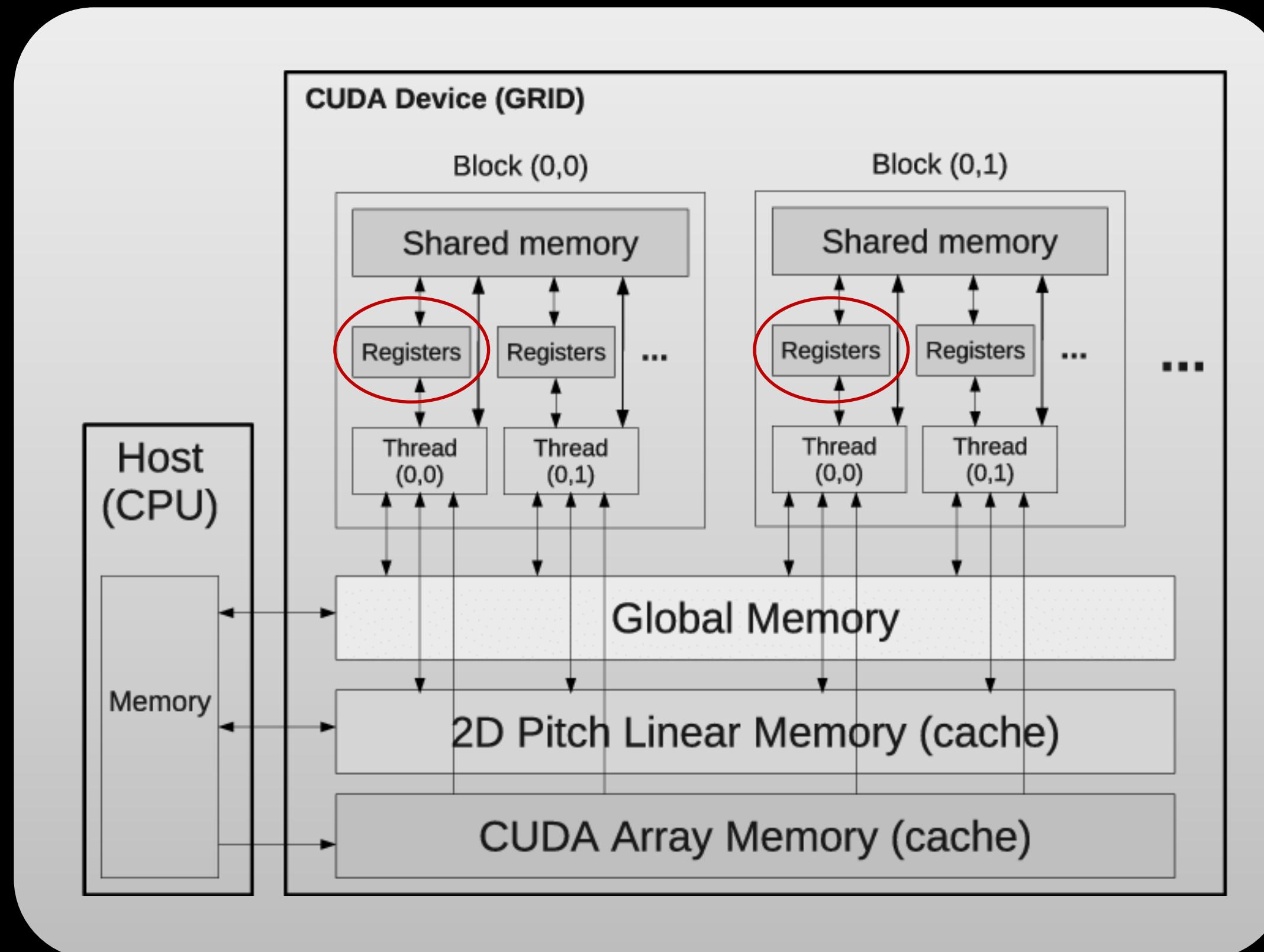
- Кэш второго уровня
- Глобальная память (DRAM)



Остановимся на каждом типе памяти  
подробнее

# Модель организации памяти

## Регистры



Самая быстрая память,  
доступная CUDA ядру

Резервируются и закрепляются  
за каждым потоком

На SM не может разместиться  
блок, если хотя бы одному  
потоку не хватило регистров

# Модель организации памяти

## Специальные регистры

threadIdx, blockIdx,  
gridDim, blockDim...



```
__global__ void incKernel(int *g_out, int *g_in, int N, int inner_reps) {  
    int idx = blockIdx.x * blockDim.x + threadIdx.x;  
  
    if (idx < N) {  
        for (int i = 0; i < inner_reps; ++i) {  
            g_out[idx] = g_in[idx] + 1;  
        }  
    }  
}
```

Полный список  
регистров см. PTX ISA  
(о ней немного позже)

# Модель организации памяти

Разделяемая память

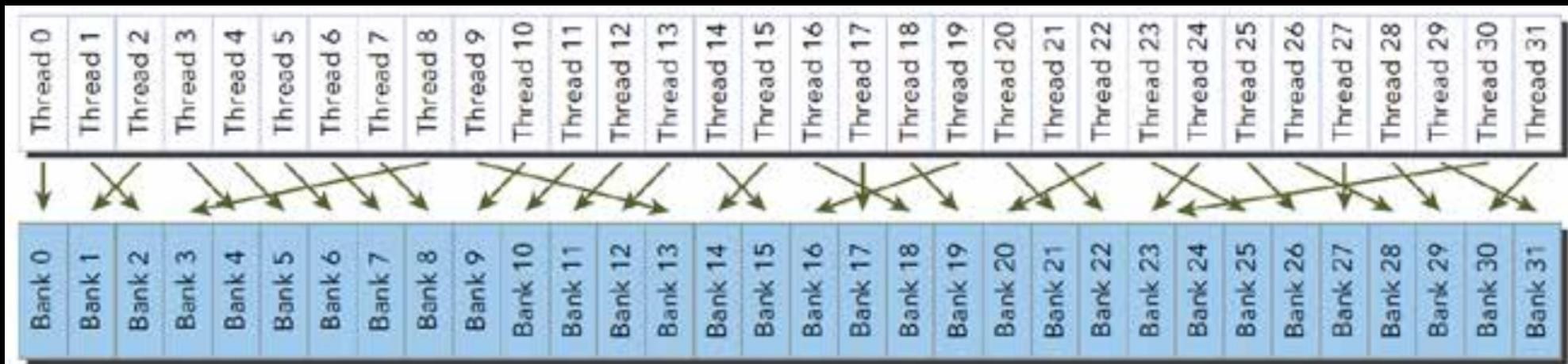


Рис. 6б. Бесконфликтный доступ к банкам данных

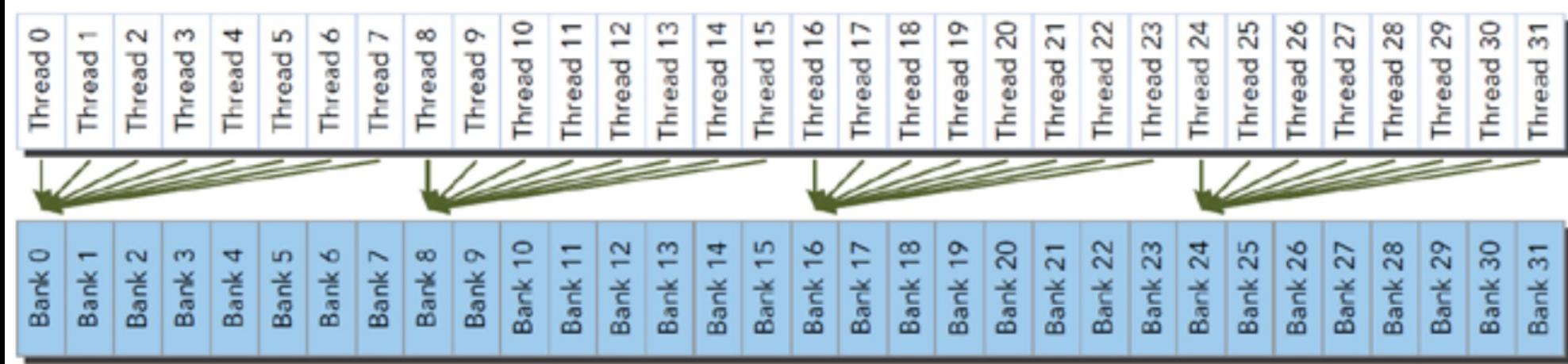


Рис. 6в. Конфликтный доступ к банкам данных

Разделяется на банки

Используется для достижения  
высокой пропускной  
способности

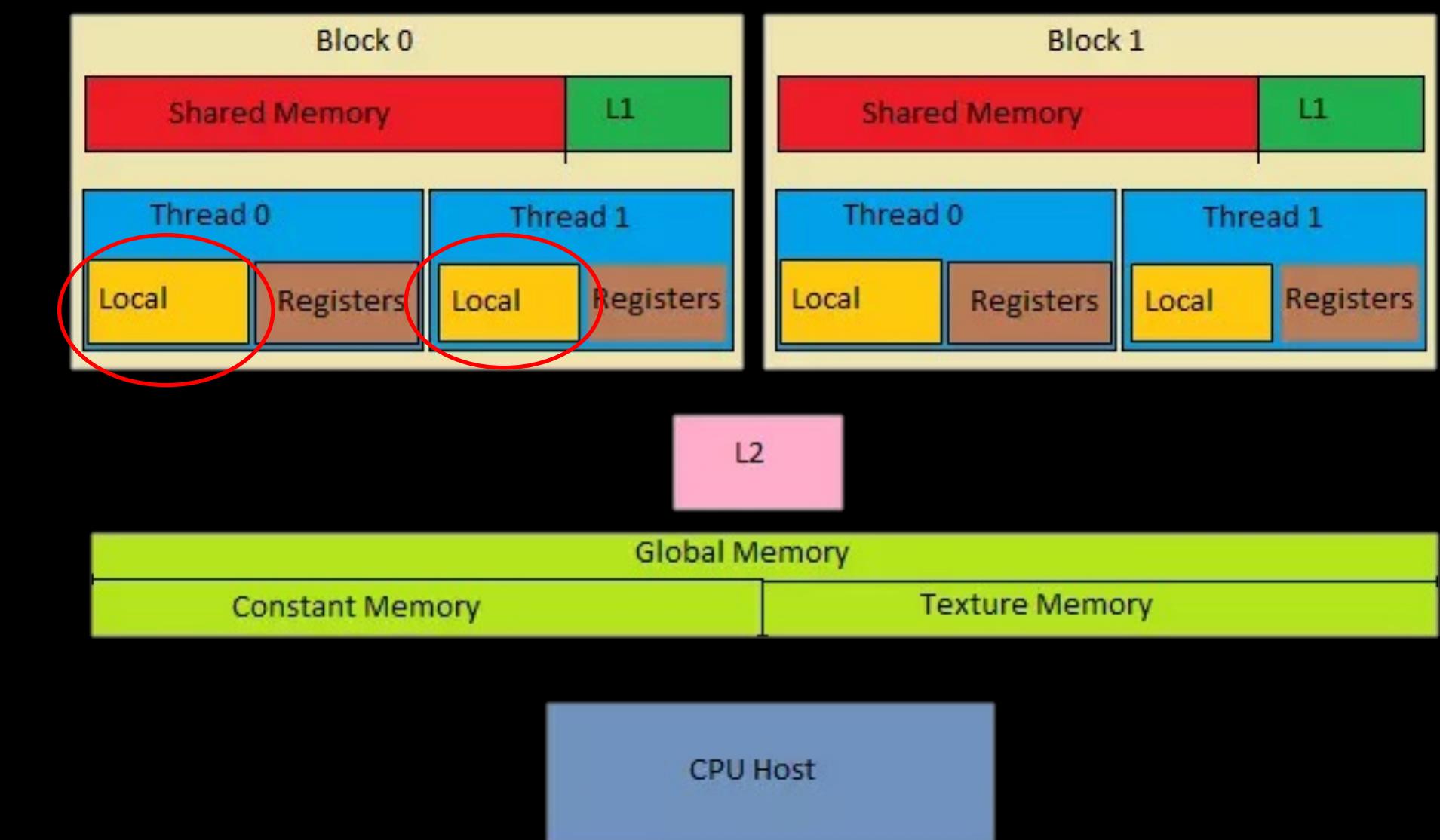
Криворукость -> конфликты

# Модель организации памяти

## Локальная память

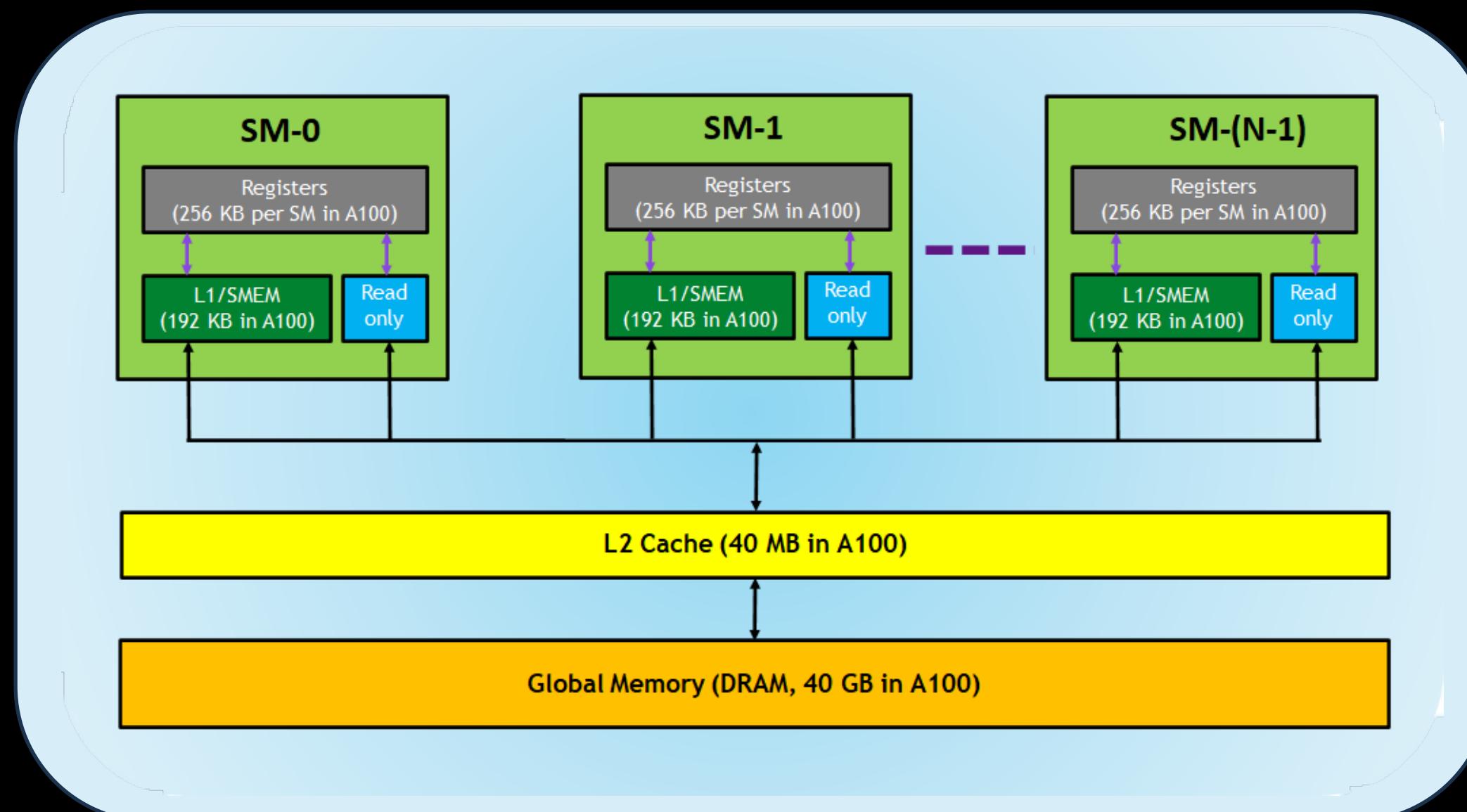
Размещается в DRAM/кэшах, но недоступна другим потокам

Туда кладется все, что плохо лежит в регистрах (не помещается, или даже не задумывалось, что оно может помещаться...)



# Модель организации памяти

## Глобальная память



Доступна всем потокам со всех SM  
в любое время

Ее больше всего на видеочипе

Медленная, не синхронизирована,  
кэшируется, является  
потенциальным источником  
головной боли

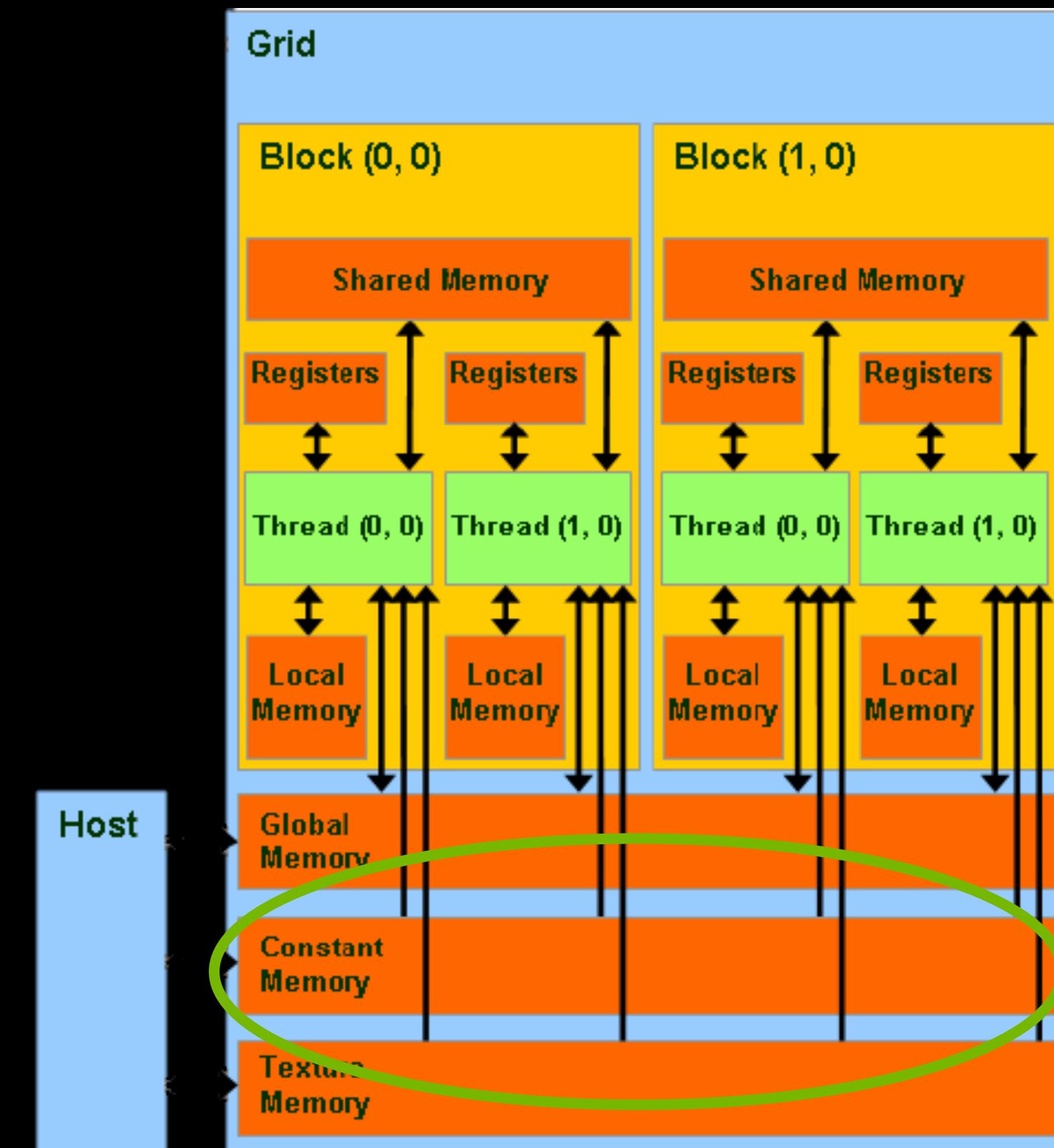
# Модель организации памяти

## Константная память

От части легаси – кэши для глобальной памяти как правило будут быстрее

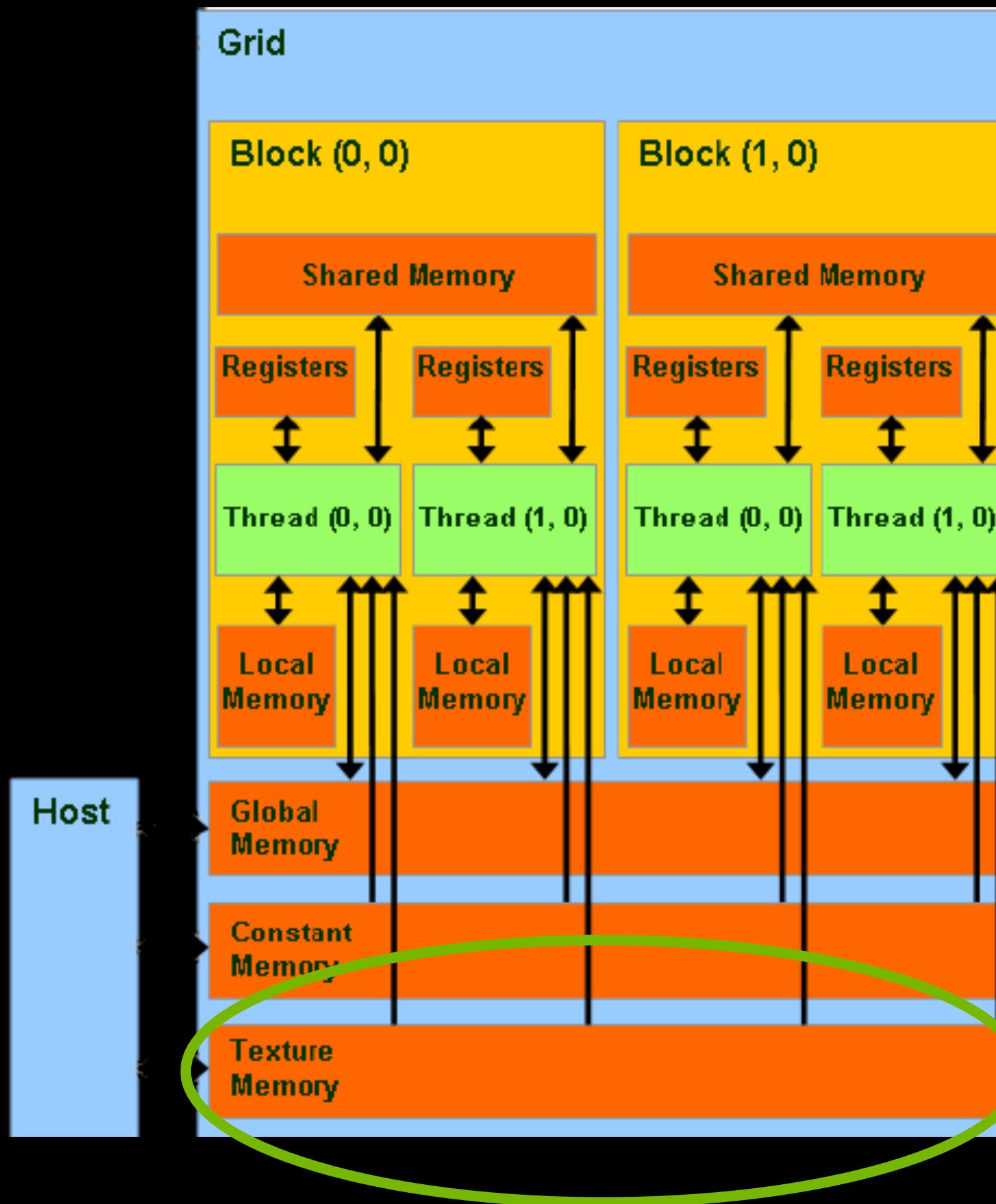
Тем не менее, существуют маневры для оптимизации с использованием константной памяти

Делится на банки – 0й статический, остальное -- динамика



# Модель организации памяти

## Текстурная память



Пришла к нам из графической природы GPU

Однако даже вычисления общего назначения могут получить преимущество, связанное с организацией текстурной памяти

Out of our scope

# Сводная таблица доступной памяти

Тип	Доступ	Видимость	Расположение	Кэшируется
Регистры	Чтение/запись	Поток	SM	Нет
Специальные регистры	Только чтение	Поток	SM	Нет
Разделяемая память	Чтение/запись	CUDA Блок	SM	Нет
Локальная память	Чтение/запись	Поток	DRAM	Да
Глобальная память	Чтение/запись	Для всех	DRAM	Да
Константная память	Только чтение	Для всех	DRAM	Да
Текстурная память	Только чтение	Для всех	DRAM	Да

# Организация вычислений

# Модель организации вычислений

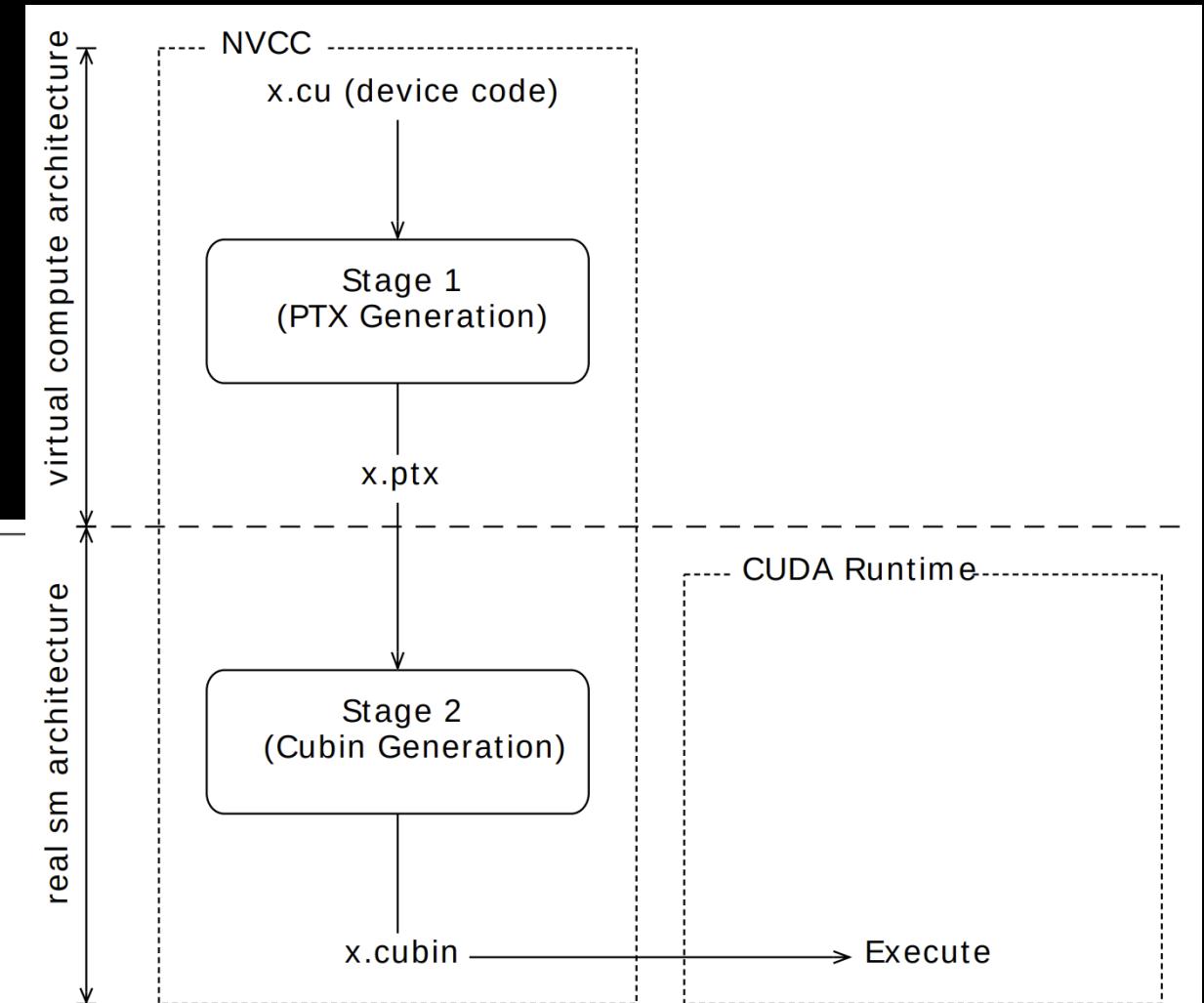
## Уровень системной платформы

Parallel Thread Execution (PTX) –  
уровень системной платформы  
для GPU NVIDIA

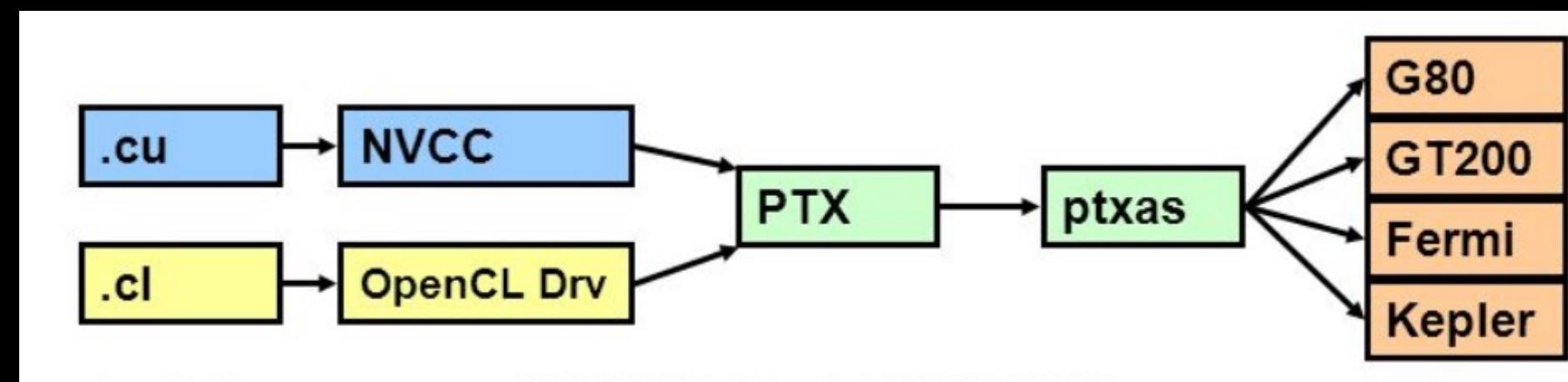
Позволяет абстрагироваться  
от конкретной программе от  
конкретной карточки и наоборот

```
1 .version 2.1
2 .target sm_20
3
4 .entry squareArray (.param .u32 a)
5 {
6     .reg .u32 arrayIndex;
7     .reg .u32 address;
8     .reg .f32 value;
9
10    ld .param .u32 address, [a];
11    mul .u32 arrayIndex, %tid.x, 4;
12    add .u32 address, address, arrayIndex;
13    ld .global .f32 value, [address];
14    mul .f32 value, value, value;
15    st .global .f32 [address], value;
16    exit;
17 }
```

Listing 2.2: PTX version of program 2.1's squareArray kernel



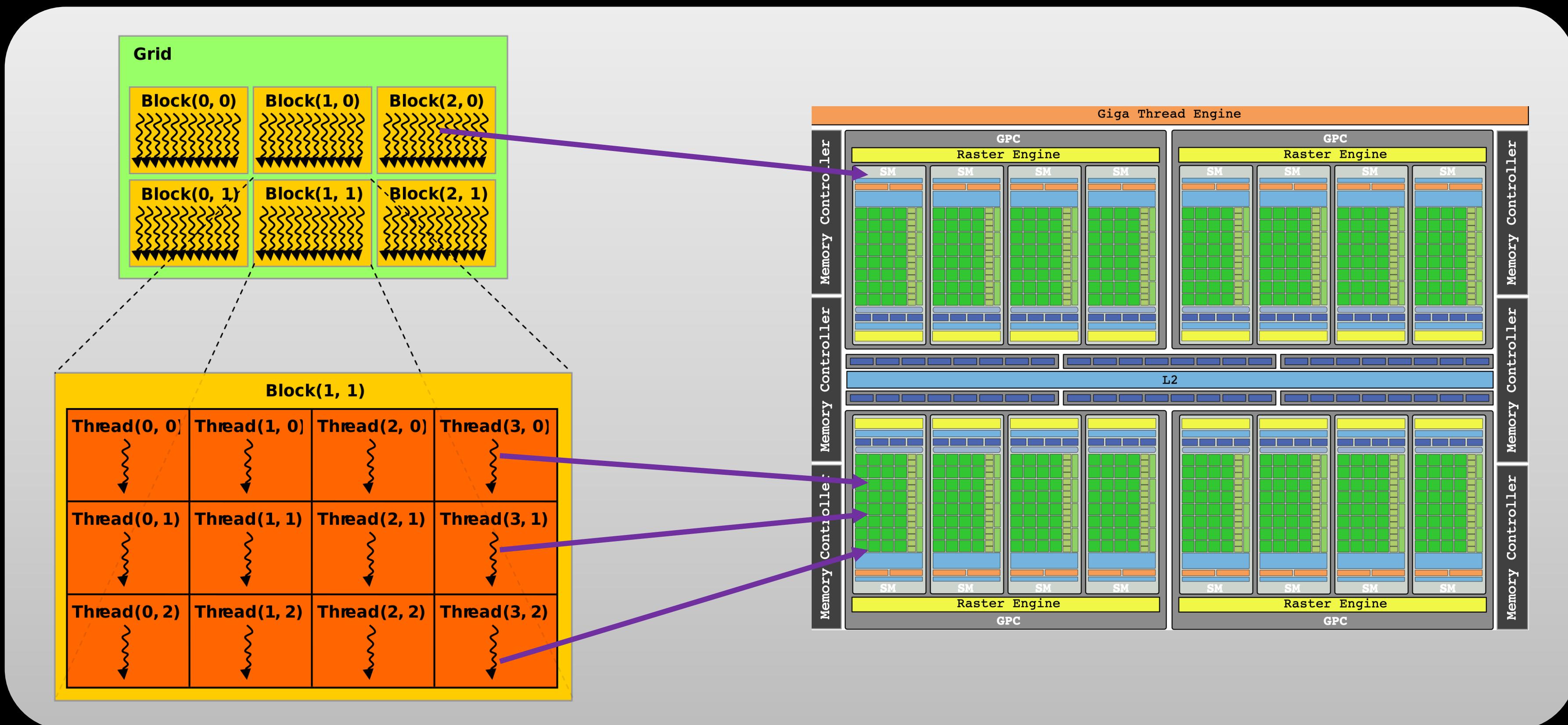
# Основные цели PTX



- Предоставлять стабильную систему команд, расширяемую на разные GPU
- Компилируемое приложение будет иметь такую же производительность, как и если бы мы напрямую программировали вычисления на карте
- Помощь всем сторонним разработчикам (компиляторов, middleware, etc.) путем предоставления унифицированной системы команд для взаимодействия с разными GPU
- Все преимущества, предоставляемые любой системной платформой любой железки относятся и к этому случаю тоже

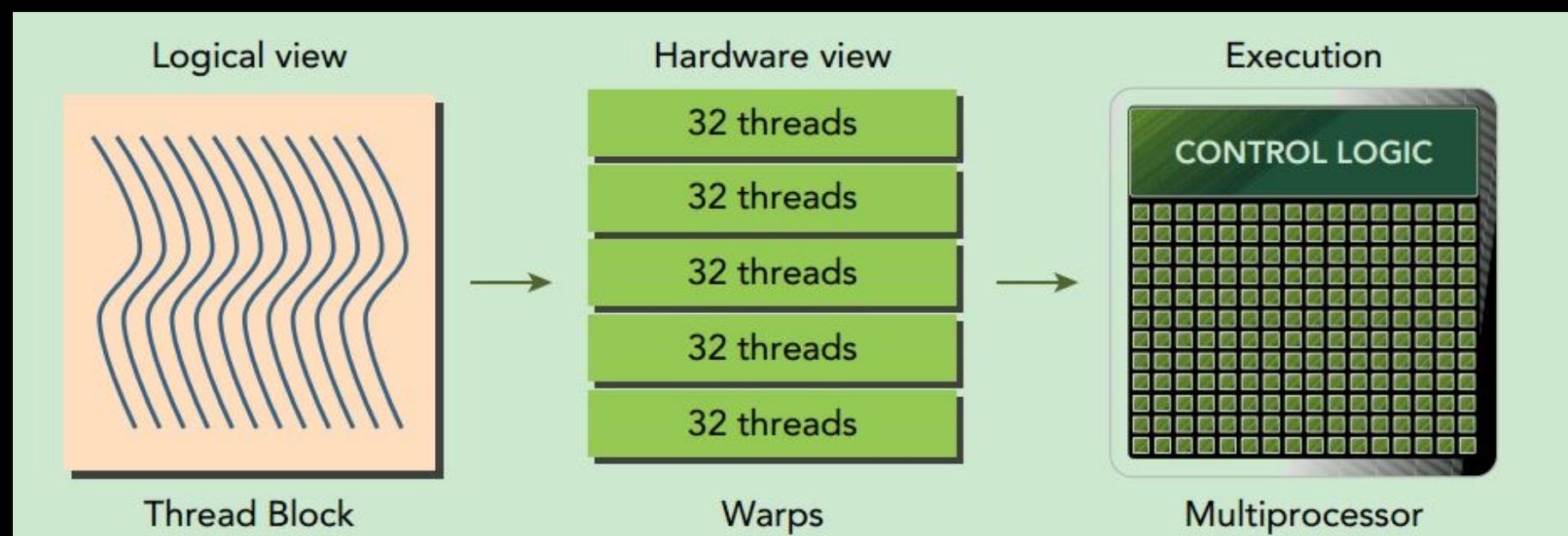
# Модель организации вычислений

## Организация группировки



# Модель организации вычислений

## Варпы



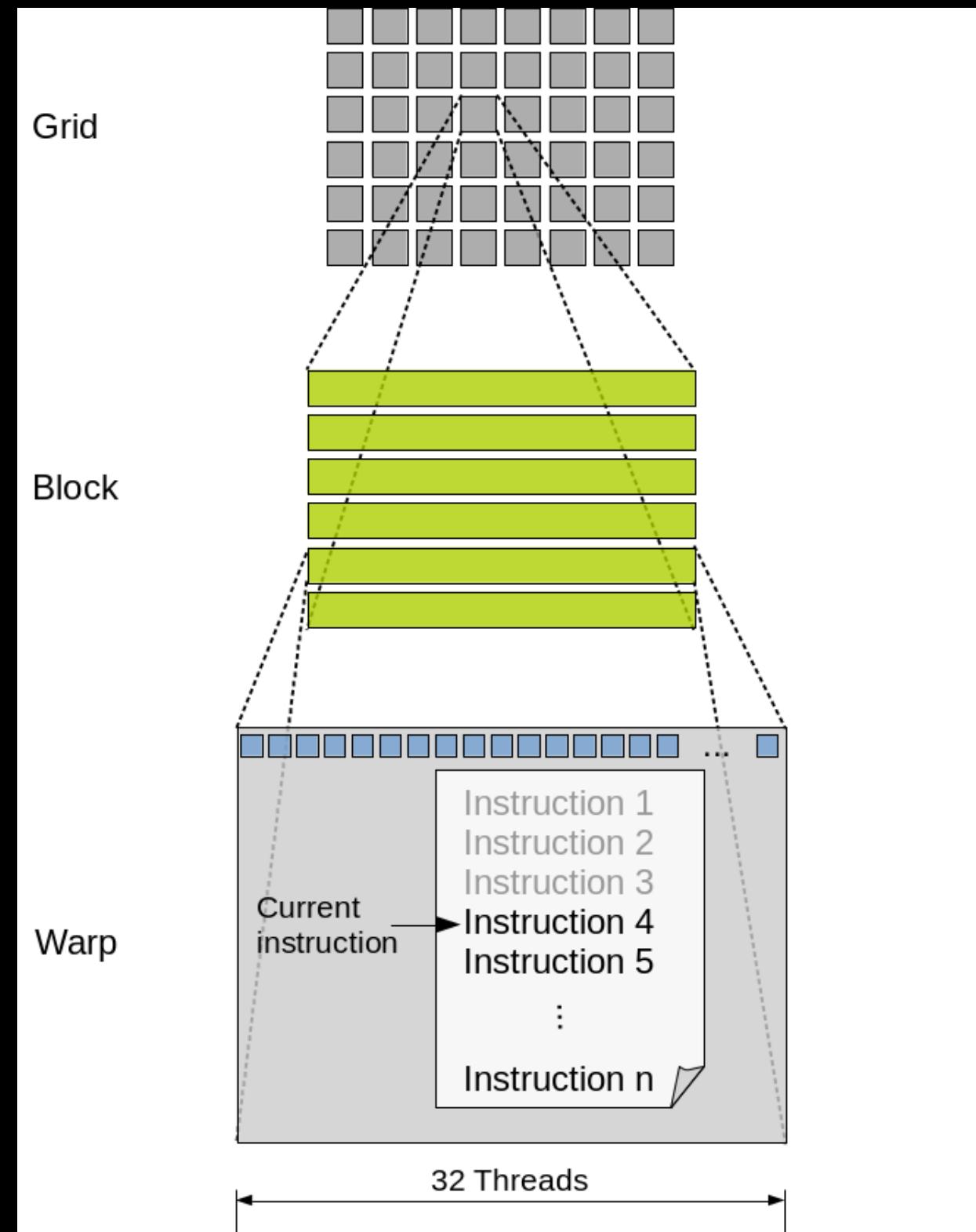
Для того, чтобы исполнить блок на SM:

- Мы имеем лишь немного CUDA ядер, сильно меньше чем размер блока
- Давайте разобьём блок на равные кусочки (32), чтобы SM выполнял блок этими кусочками – кусочек называется варпом
- Последовательная нумерация потоков в варпе – 0..31, 32..63, ...
- Проблема решена, теперь блок исполняется варпами

# Модель организации вычислений

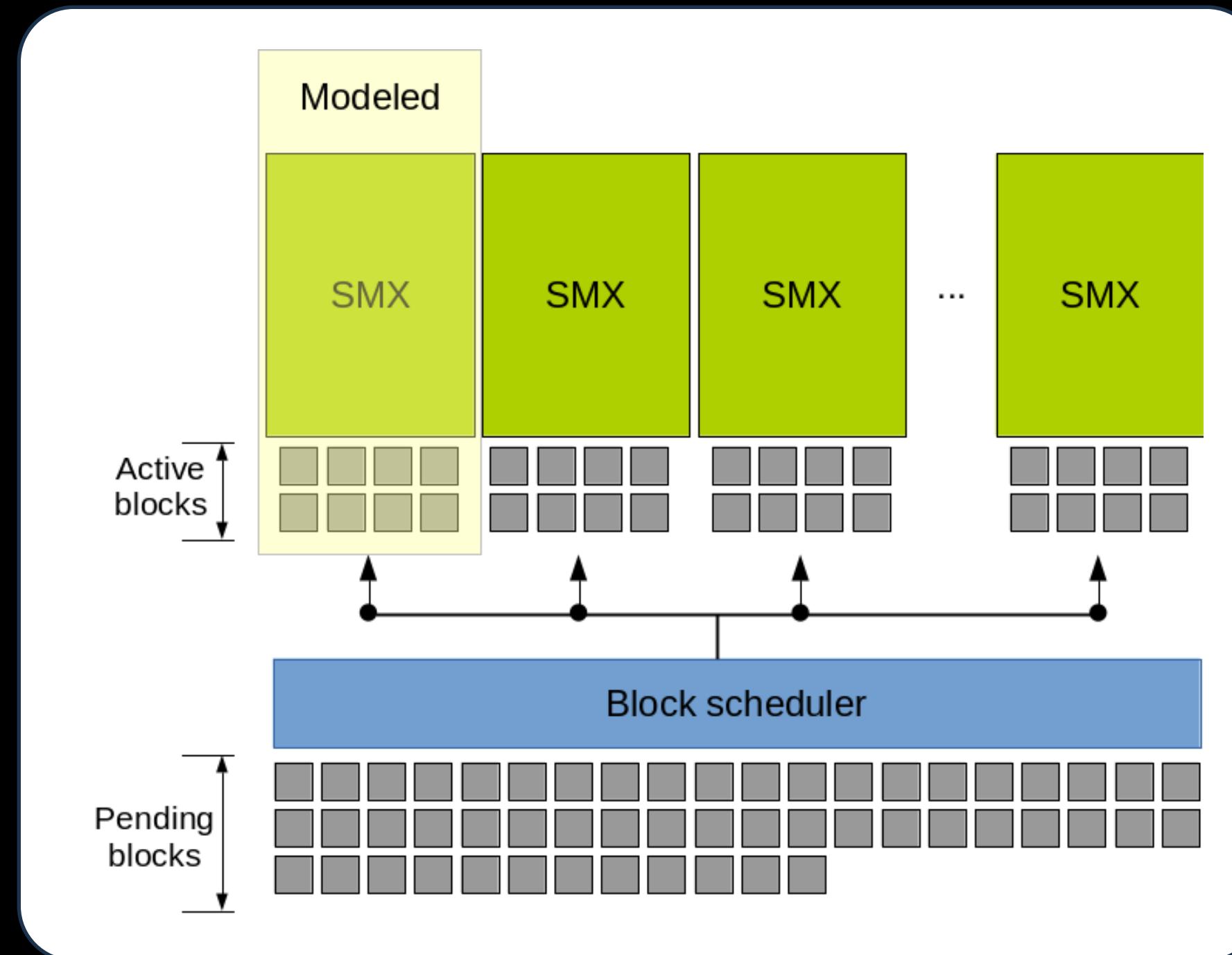
## Исполнение варпов

- Варп полностью загружает SM – в один момент времени SM занимается исполнением одного варпа
- Другие варпы исполняться в этот момент не могут, так как весь блок размещен на одном SM
- Для всех потоков в варпе в единый момент времени выполняется одна и та же инструкция
- Для максимальной эффективности не рекомендуется использовать ветвления
- Если последние все же присутствуют, потоки, не прошедшие условия, заморозятся, пока для всех вновь не будет исполняться общая инструкция



# Модель организации вычислений

Разница в переключении контекста потоков



- GPU умеет очень эффективно переключать контекст исполнения
- Информация о текущей исполняемой инструкции для каждого варпа хранится в SM

# Модель организации вычислений

На что еще способен планировщик?

- Планировщик умеет загрузить SM полностью
- Все вычислительные мощности на SM обязаны выполнять полезную работу
- Они могут это делать довольно долго
- Планировщик умеет с этим жить и работать
- Как он работает – магия

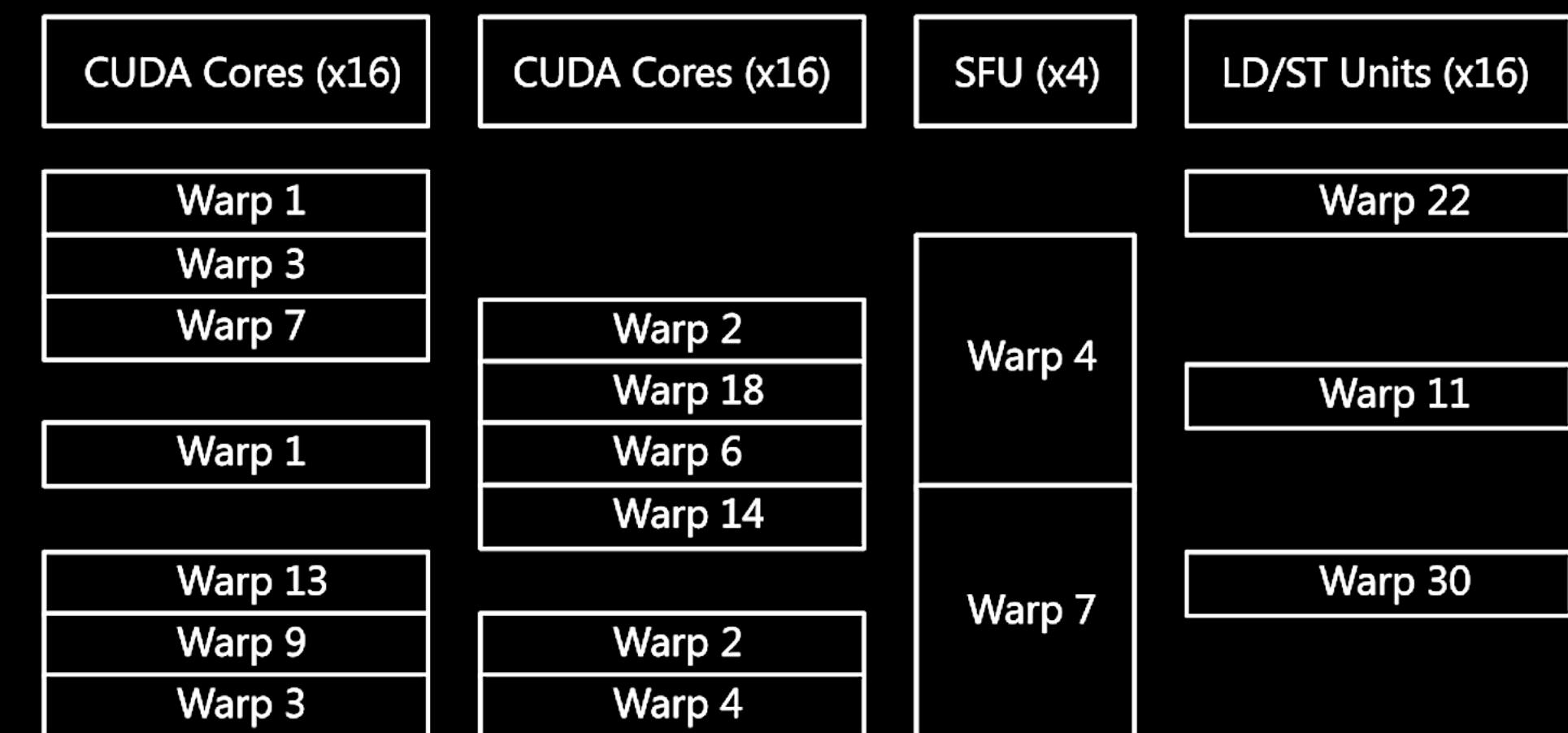
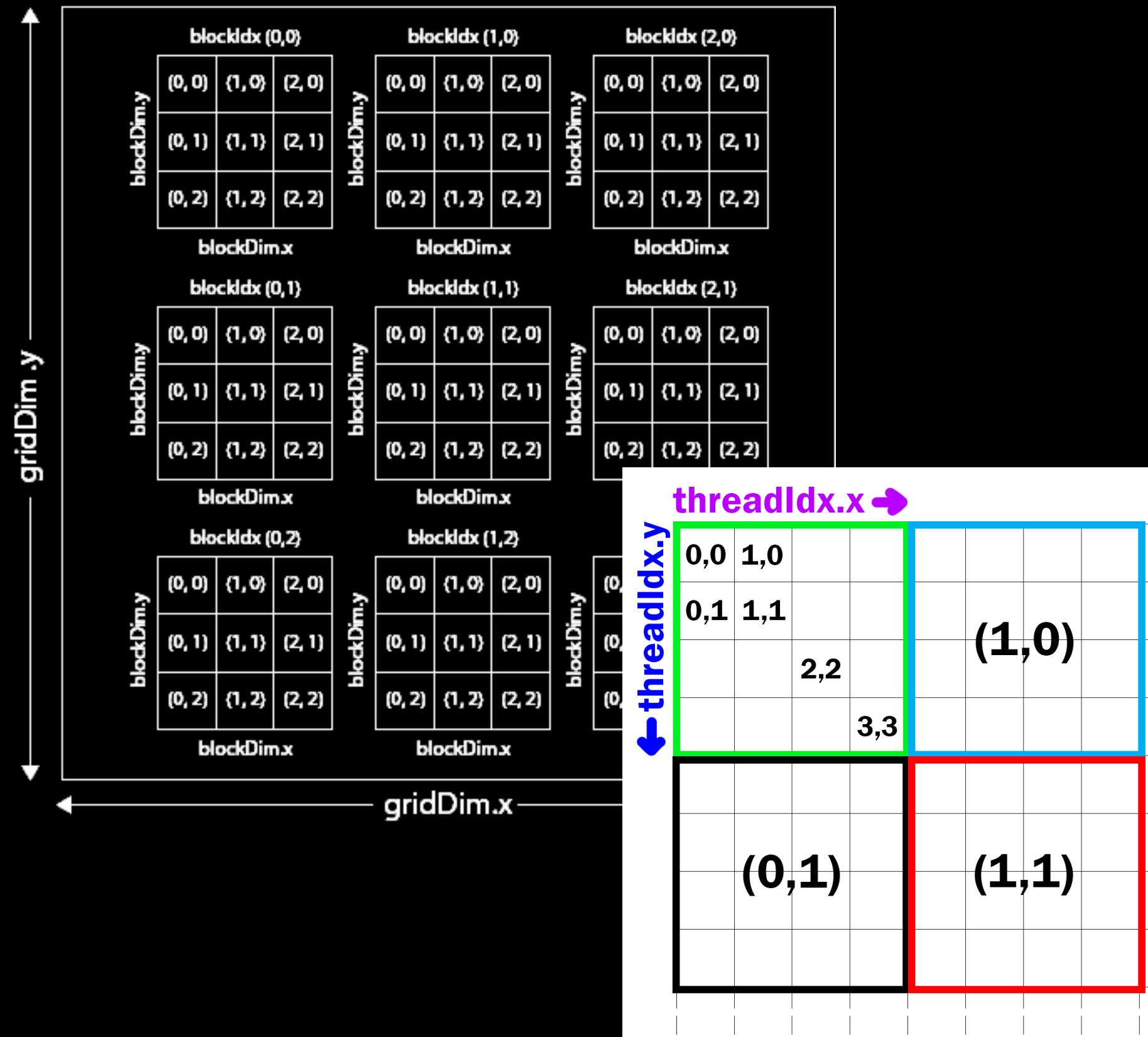


Figure 2.3: Utilization of Execution Blocks

# Модель организации вычислений

У всего есть свой ID...

## CUDA Grid

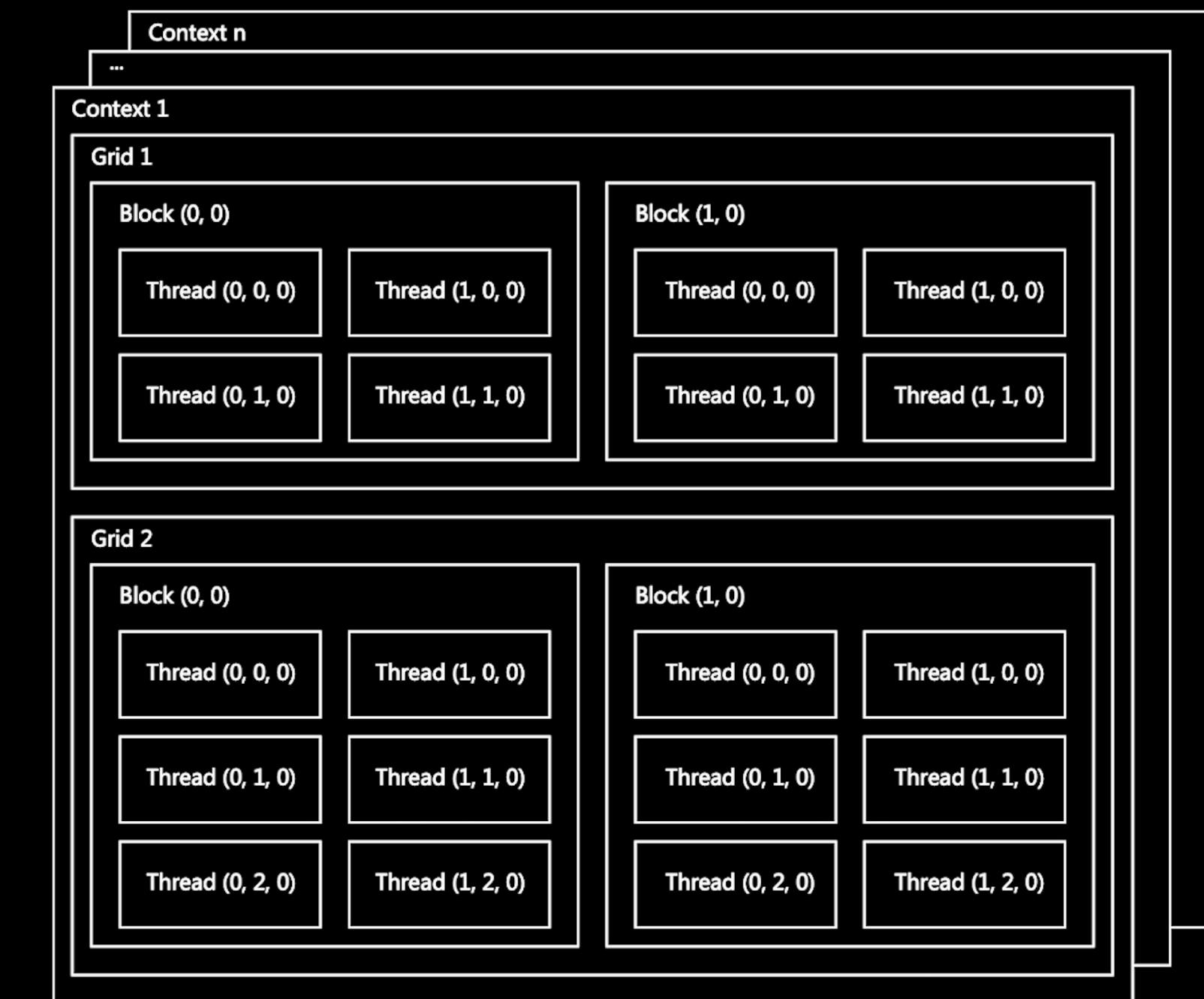


- Блокам в сетке присвоен уникальный ID
- Потокам в блоке тоже присвоен уникальный ID
- На картинке показана двумерная сетка с двумерными блоками, но они также могут быть и трехмерными!

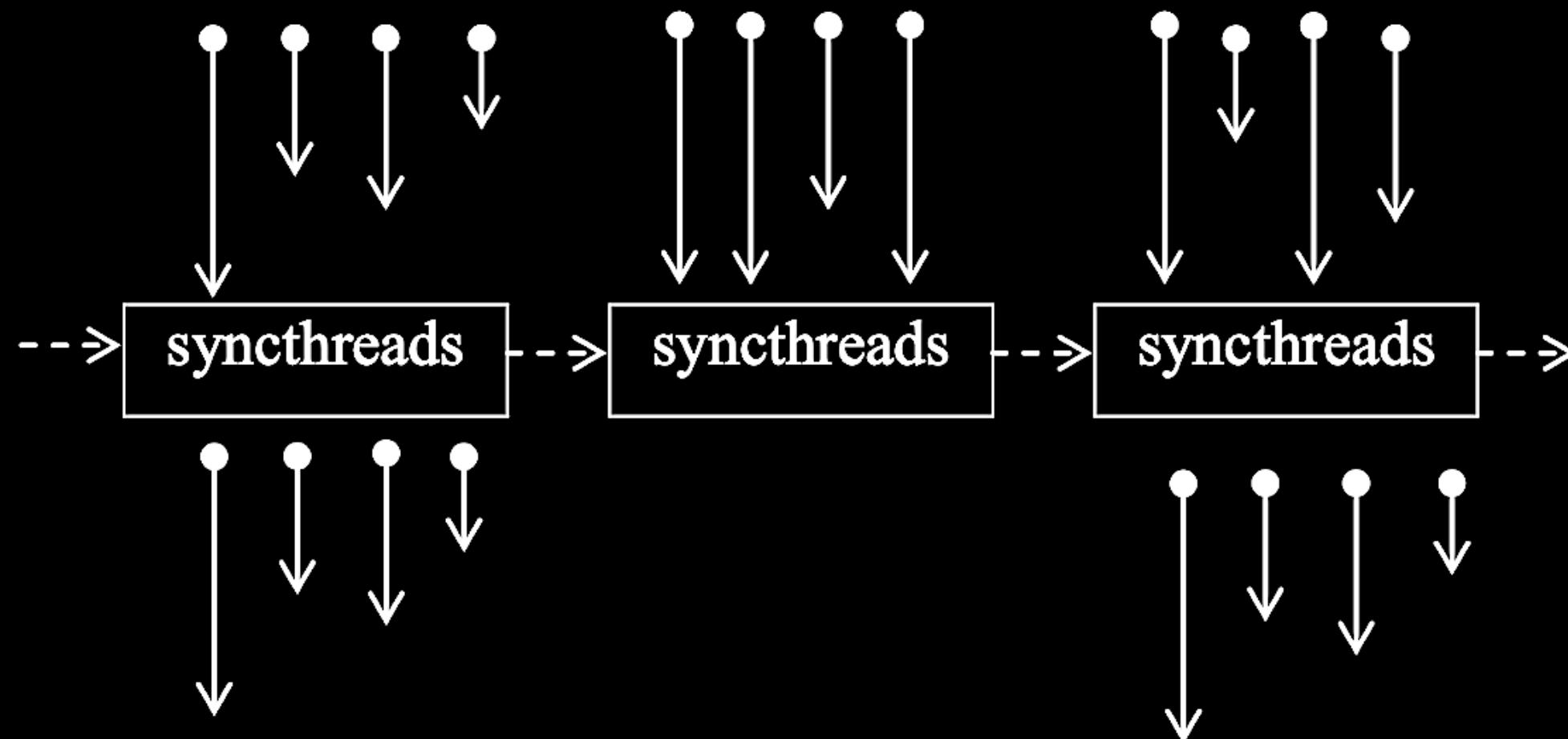
# Понятие контекста

Как можно объединить сетки и заставить их последовательно исполняться?

- Иногда нам нужно исполнять несколько разных kernel функций
- Сетки выстраиваются в контекст, и исполняются по очереди
- Напоминание: GPU исполняет за раз только потоки из одного контекста
- В контексте может быть до 16 сеток (для архитектуры Fermi)



# Механизмы синхронизации



- Потоки из разных варпов почти всегда рассинхронизированы
- Тем не менее, мы должны убедиться, что все потоки из блока выполнили какую-то операцию с памятью, и мы можем посмотреть на ее результат
- Решение – барьерная синхронизация
- Лишь только когда все потоки из блока войдут в барьер, потоки из блока смогут продолжить выполнение Kernel-функции

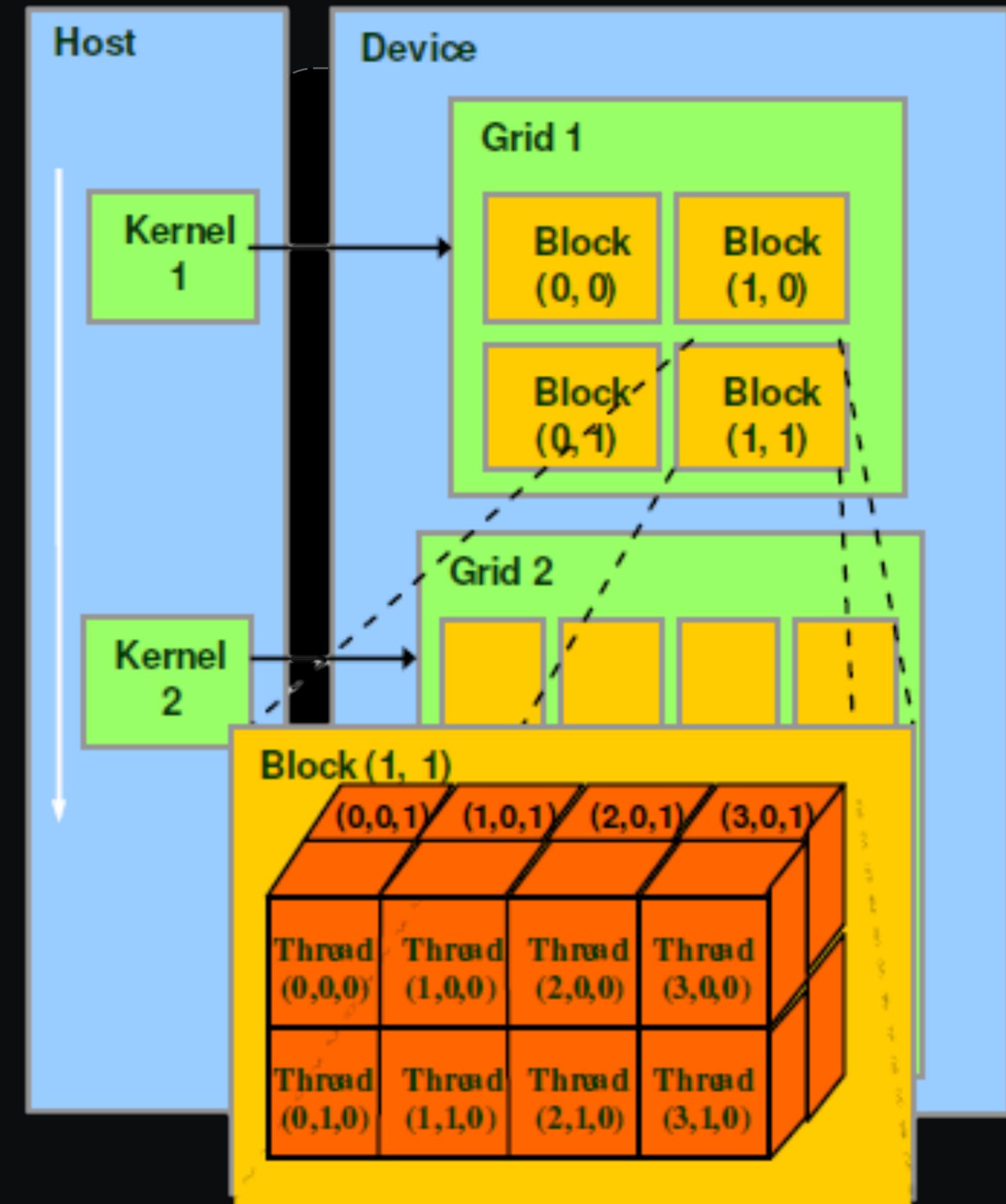
# Возможности синхронизации заканчиваются

А проблемы никуда не исчезли

- Два и более потока могут записывать данные в одну ячейку глобальной памяти
- -> Доступ к глобальной памяти на запись с целью дальнейшего чтения другим потоком как правило небезопасен
- Данные из глобальной памяти могут быть закэшированы.
- -> Обновление данных в DRAM не будет означать обновление данных в кэше

## На помощь приходят контексты

- Разные Kernel-функции можно определить в разные контексты
- Гарантируется, что к концу исполнения вычислений в глобальной памяти будет лежать их результат
- Соответственно, мы можем безопасно читать данные из глобальной памяти
- Очень дорогая операция



# Подведем итоги

Вопрос на экзамен:

Модель организации памяти в CUDA-совместимых GPU. Организация доступа к памяти между разными потоками.

Ручная и автоматическая оптимизация операций с памятью

Сегодня мы узнали:

- Что такое NVIDIA CUDA
- Какие проблемы породили данную платформу
- Как эти проблемы удалось решить
- Устройство платформы NVIDIA CUDA и модель ее вычисления
- Базовые понятия о CUDA-совместимой GPU
- Механизмы синхронизации потоков



<- Issue

# Спасибо за внимание!

Пожалуйста, оставьте отзыв /lab1feedback  
...или по QR-коду ниже

