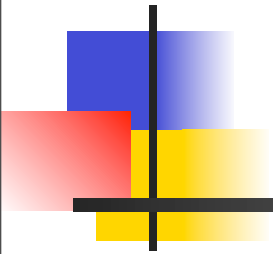


# intro

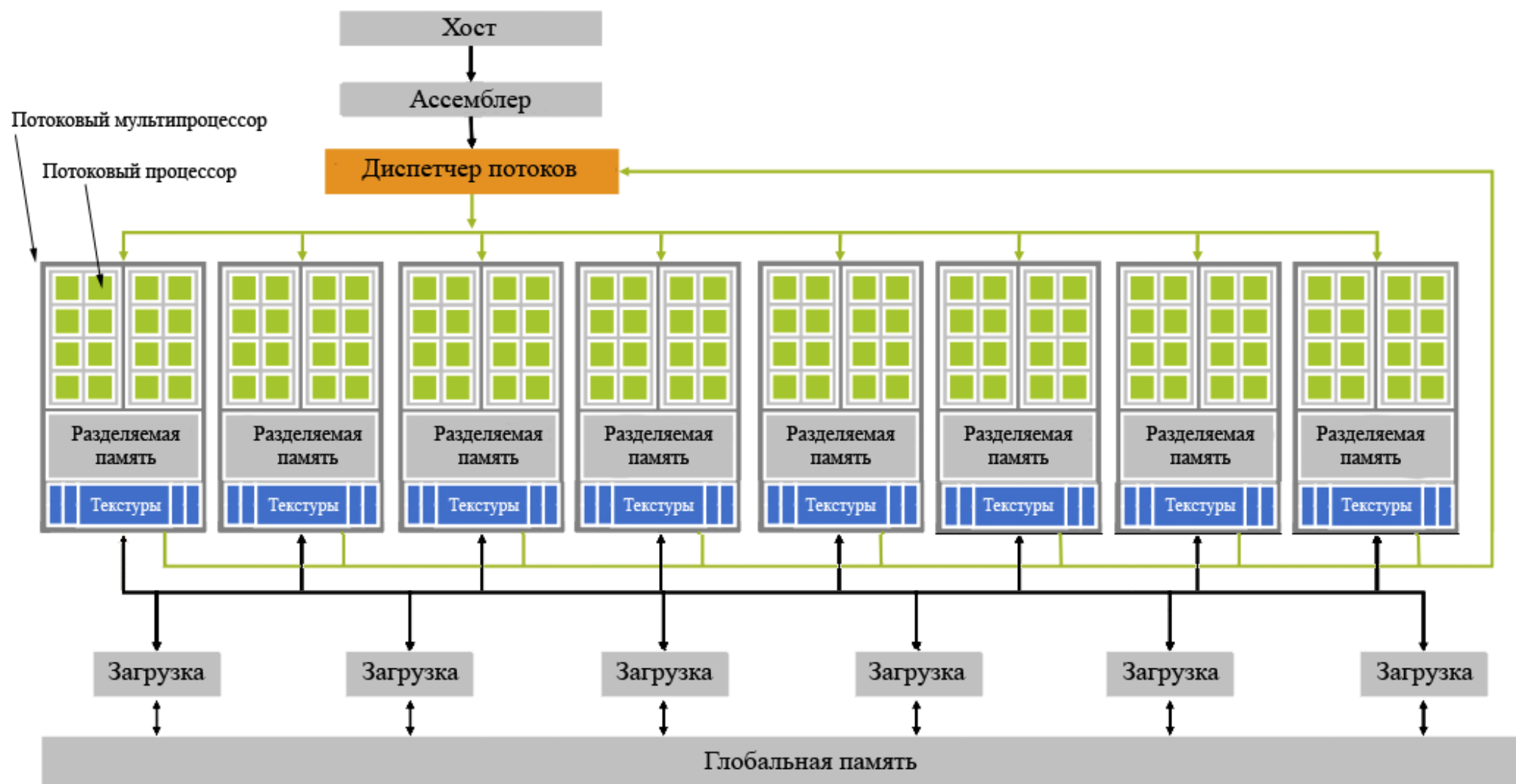


# **GPGPU. NVidia CUDA**

## **Лекция 1**

**Лектор: Селиверстов  
Евгений Юрьевич**

# Принципиальная схема GPU



# Архитектуры и СС

---

CUDA вводит понятие Compute Capability (вычислительные характеристики).

Поддержка программой различных поколений устройств.

Принципиальные архитектурные различия.

# Compute Capability 1.x

---

- 1.0, 1.1 - G80

8 SP на MP

2 MP на TPC

2 SFU

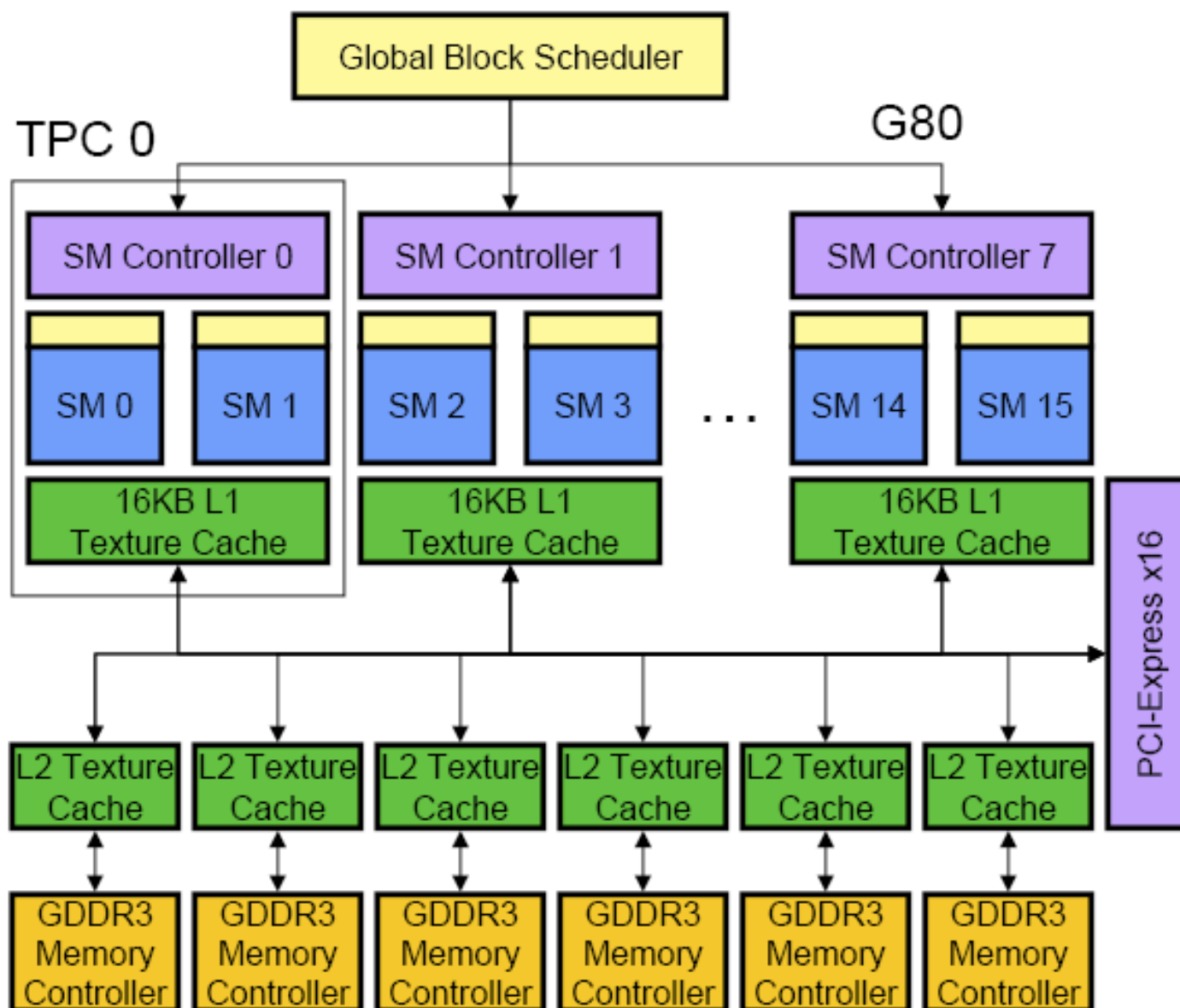
8K регистров

16K общей памяти

24-битная точность для int

32-битная точность (single) для float

# Cxema G80



# Compute Capability 1.x

---

1.2 - GT200

2 MP на TPC

16K регистров

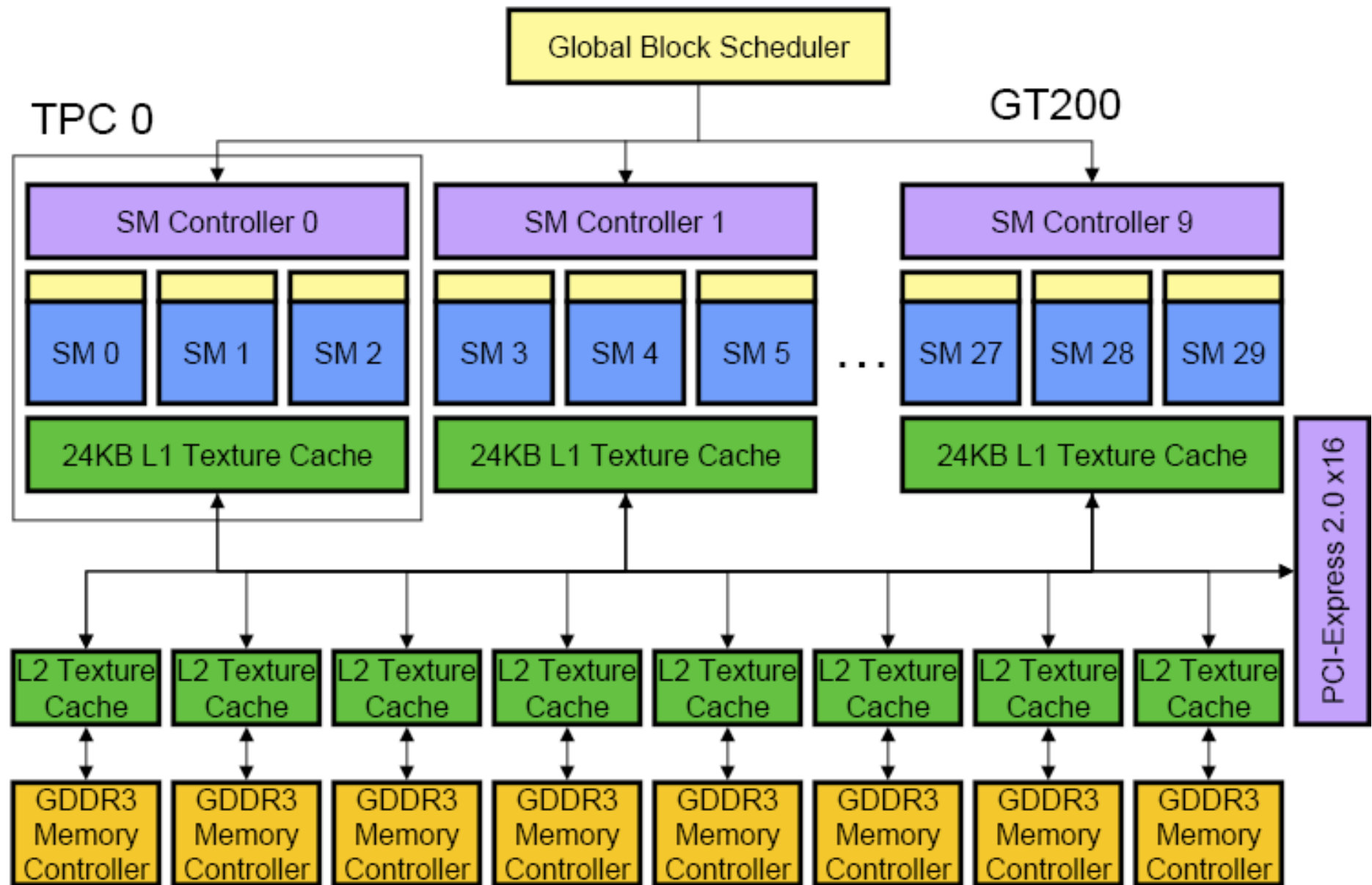
1.3 - GT200

блок вычислений double

64-битная точность (double) для float

производительность double  $\approx 1/8$

# Схема GT200





# Compute Capability 2.x

---

- 2.0

32 SP на MP

TPC -> GPC (4 MP)

4 SFU

32K регистров

48K общей памяти

32-битная точность для int

64-битная точность (double) для float

производительность double  $\approx 1/2$

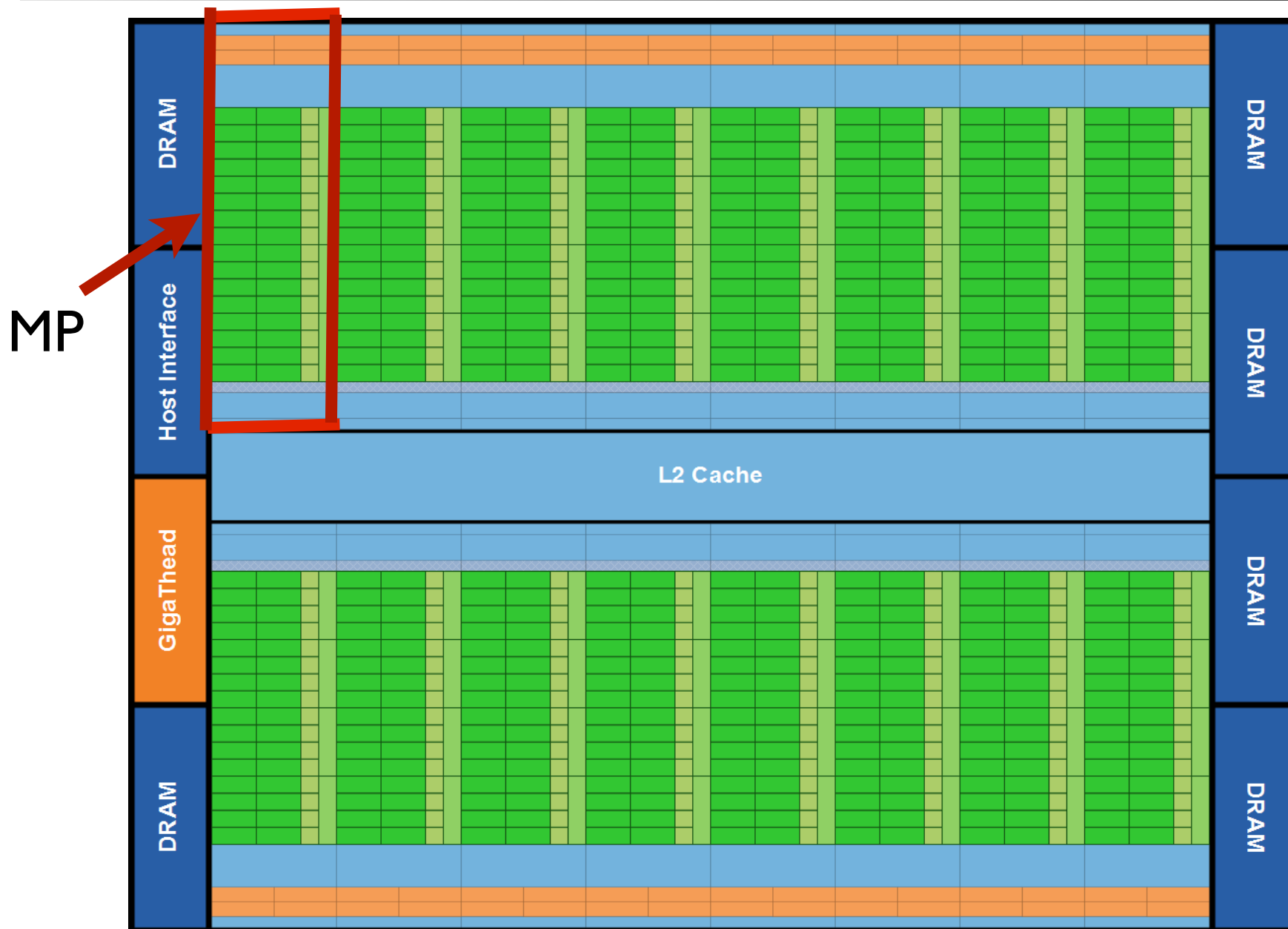
- 2.1

48 SP на MP

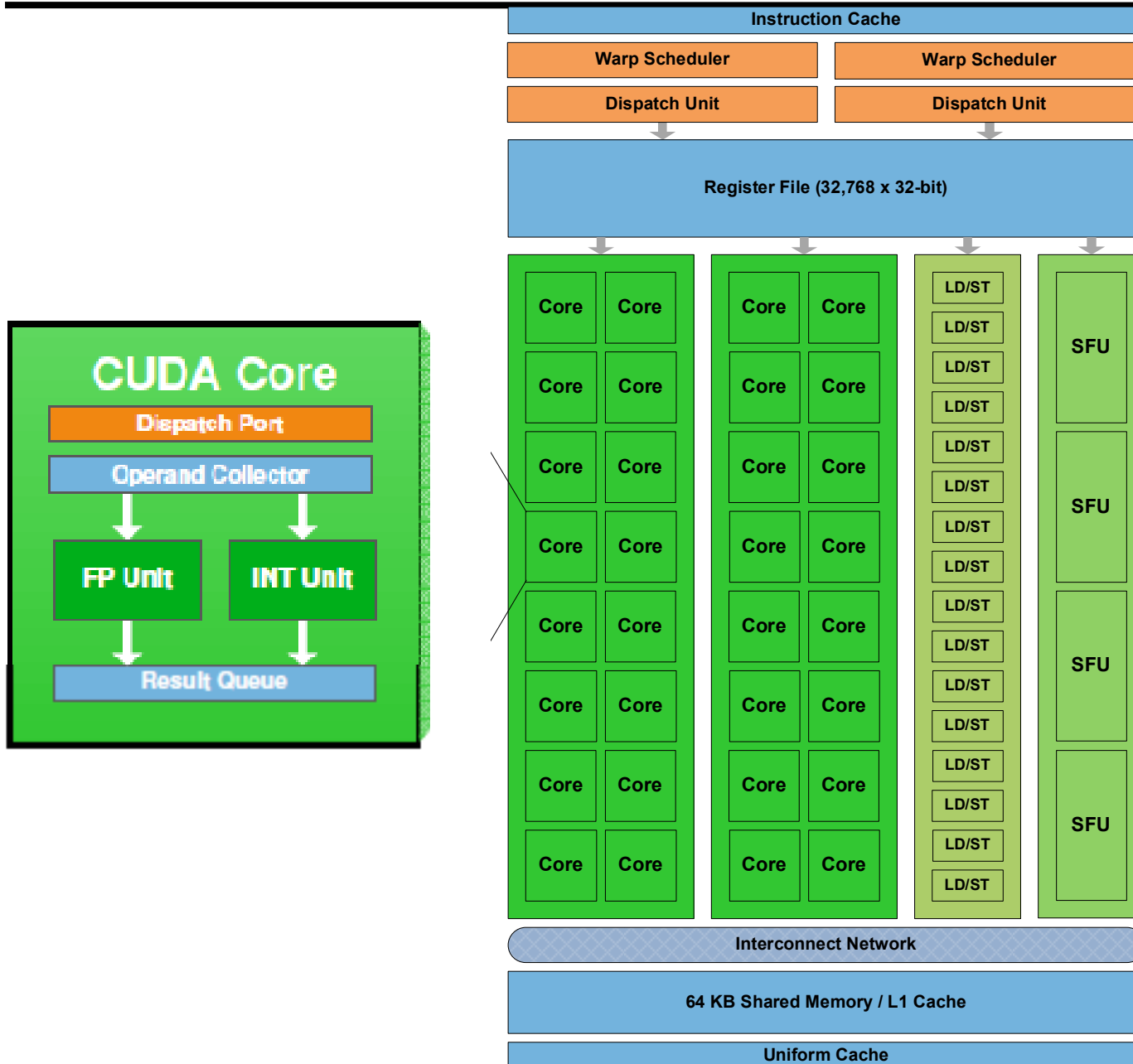
8 SFU

выполнение двух инструкций за такт

# Схема GF100



# Cxema GF100



# GeForce

---

- 8800 Ultra

G80 - CC 1.0, 1.1

128 SP, 16 MP, 0.5Gb, 518 GFLOPs

- GTX295

GT200 - CC 1.2, 1.3

240 SP, 30 MP, 2GB, 1063 GFLOPs

- GTX480

GF100 - CC2.0

480 SP, 15 MP, 1.5Gb, 1344 GFLOPs

# Tesla

---

Процессоры:

- Tesla C860
- Tesla C1060
- Tesla C2050
- Tesla C2070
- Tesla C2075

Сервера:

- Tesla Sxxx

Из 2/4 процессоров.

# Tesla 8

---

## **Tesla C870** (на основе G80)

Ядра - 128 SP, 16 MP, 1.35 GHz  
пиковая производительность 518/0 GFLOPs

Память - 1.5 Gb, 384 bit, 1.6GHz  
пропускная способность 76 GB/s

СС 1.0

# Tesla 10

---

## **Tesla C1060** (на основе GTX285)

Ядра - 240 SP, 30 MP, 1.35 GHz  
пиковая производительность 933/76 GFLOPs

Память - 4 Gb, 512 bit, 1.6GHz  
пропускная способность 102 GB/s

СС 1.3

# Tesla 20

---

## **Tesla C20xx** (на основе GF100)

Ядра - 448 SP, 14 MP, 1.15GHz  
пиковая производительность 1288/515 GFLOPs

Память - 3/6 Gb, 384 bit, 1.5GHz  
пропускная способность 144 GB/s

- C2050
- C2070 + 3Gb
- C2075 + DVI

СС 2.0

Высокая скорость вычислений DP (4x)



# Fermi

---

Устройство GF100:

- 16 потоковых мультипроцессоров (SM)
- Структура SM
  - 32/48 ядра (против 8)
  - int/float блоки в каждом ядре
  - 4/8 SFU (против 2)
  - 64Kb разделяемой памяти (против 16)
  - 32K регистров
- кэш L1
- кэш L2

# Fermi

---

## Особенности Fermi с точки зрения CUDA

- Параллельное выполнение различных ядер
- Высокая скорость double precision
- Объединенное адресное пространство
- Конфигурируемые кэши L1/L2
- Параллельная выборка двух варпов

# GPGPU

---

GPGPU - применение GPU для вычислений, не связанных с выводом графики

Мотивация:

- высокая пиковая производительность
- большая степень аппаратного параллелизма
- хорошая приспособленность к задачам с высокой арифметической интенсивностью

# Реализации GPGPU

---

Основные подходы к GPGPU:

- шейдеры (HLSL, Cg)
- BrookGPU
- CUDA (NVidia)
- Close to Metal, Stream (ATI)
- DirectCompute (Microsoft)
- OpenCL (Khronos)

# Реализации GPGPU

## GPU Computing Applications

### C

- With CUDA Extensions
- Over 60,000 developers
- Running in Production since July 2007
- SDK + Lib's + Visual Profiler and debugger

### OpenCL™

- OpenCL Conformant Driver for GPU
- Shipped 1st OpenCL GPU Driver

### DirectCompute

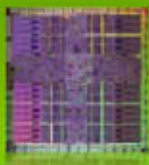
- 1st GPU demo
- Shipped 1st driver to Microsoft's Win7 developers
- Supports all CUDA-Architecture GPU's

### FORTRAN

- SW supplied by:
- The Portland Group
  - NOAA release

### Java and Python

- Compute Kernels
- Driver API Bindings



## NVIDIA GPU

with the CUDA Parallel Computing Architecture

# GPGPU: CUDA

---

Программно-аппаратная архитектура для высокопроизводительных вычислений на графических процессорах.

# GPGPU/CUDA по Флинну

---

Классификация по Флинну:

- SISD
- MISD
- SIMD
- MIMD

Другие модели:

- SIMT
- SPMD

# SIMD

---

Почему не SIMD?

- потоки, а не инструкции
- сложный поток выполнения
- иерархическая структура

SIMD:

- Векторные процессоры
- MMX
- SSEx
- AltiVec
- AVX



# GPGPU: CUDA

---

Программа для CUDA состоит из:

- функций хоста - обычный C/C++
- функций устройства (kernel, ядро) - диалект C

GPU можно рассматривать как сопроцессор, на котором выполняются ядра.

# Минимальный пример

---

Не делает ничего.

**main.cu:**

```
#include <cuda.h>

__global__ void helloCuda()
{
}

int main(int argc, char** argv)
{
    helloCuda<<<
        dim3(1,1,1), dim3(8,1,1), 0>>>();
}
```

**КОМПИЛЯЦИЯ:**

`nvcc main.cu`

# Состав CUDA

---

## 1. Driver

драйвер ядра ОС

## 2. Toolkit

компилятор, профилировщик, среда выполнения, библиотеки

## 3. SDK

примеры, сборочные скрипты

# Поддерживаемые платформы

---

## **ОС:**

- Windows
- Linux
- Mac OS X

Драйвера 32/64 битные.

Программы 32 битные (недавно 64-битные)

## **Компиляторы**

Windows: MSVC

Linux/OS X: gcc

# Уровни абстракции

---

CUBIN - машинный код, исполняемый GPU

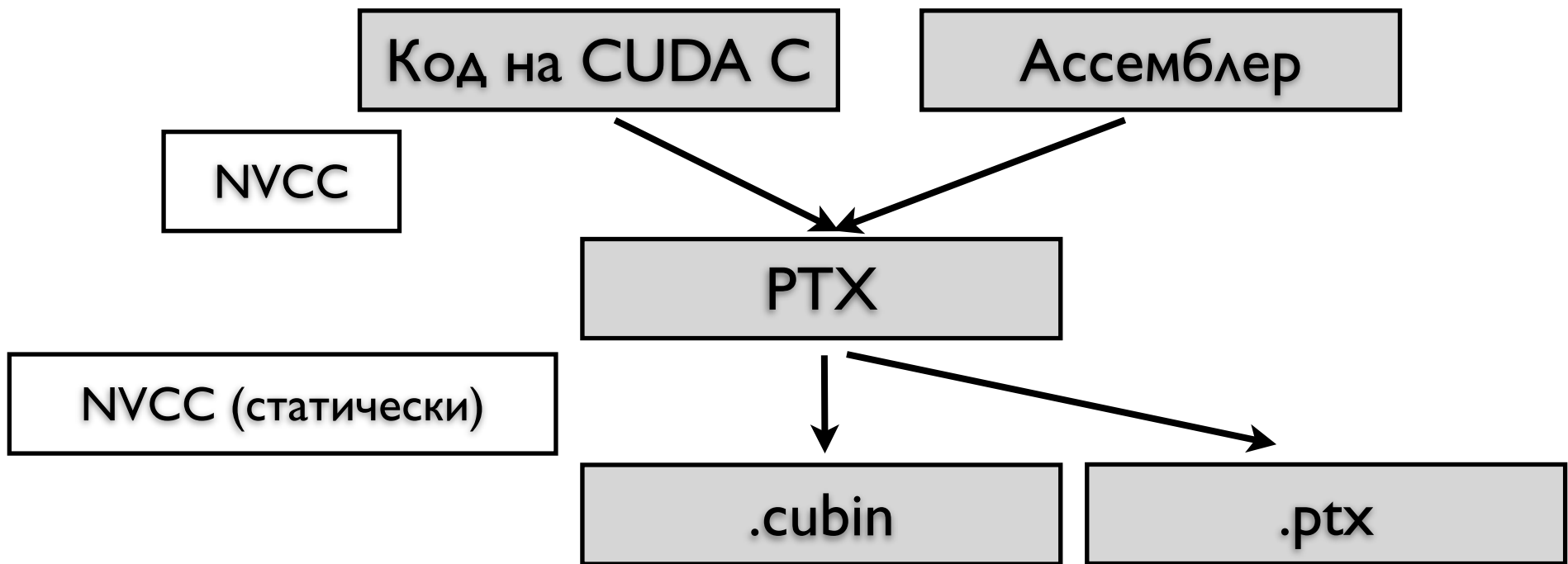
PTX - ассемблерный код

CUDA C/C++

Компиляция и выполнение возможны в два вида представлений – CUBIN или PTX.

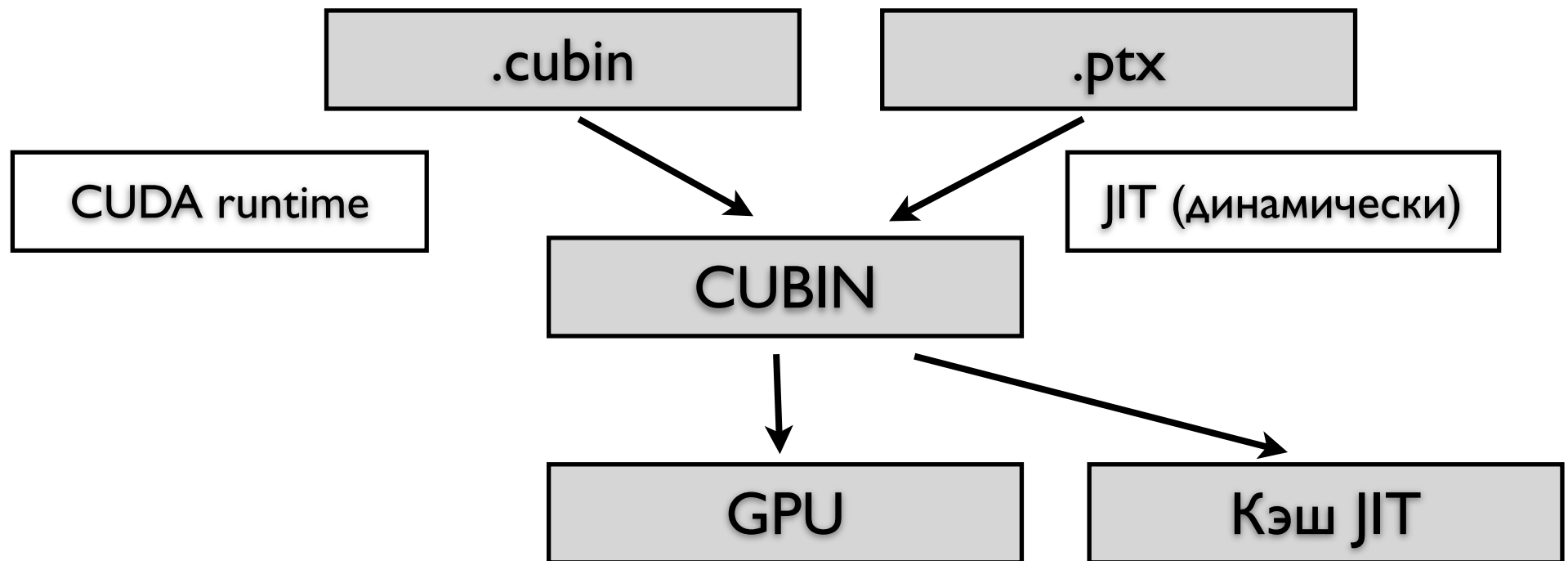
# Компиляция

---



# Выполнение

---



# Компилятор NVCC

---

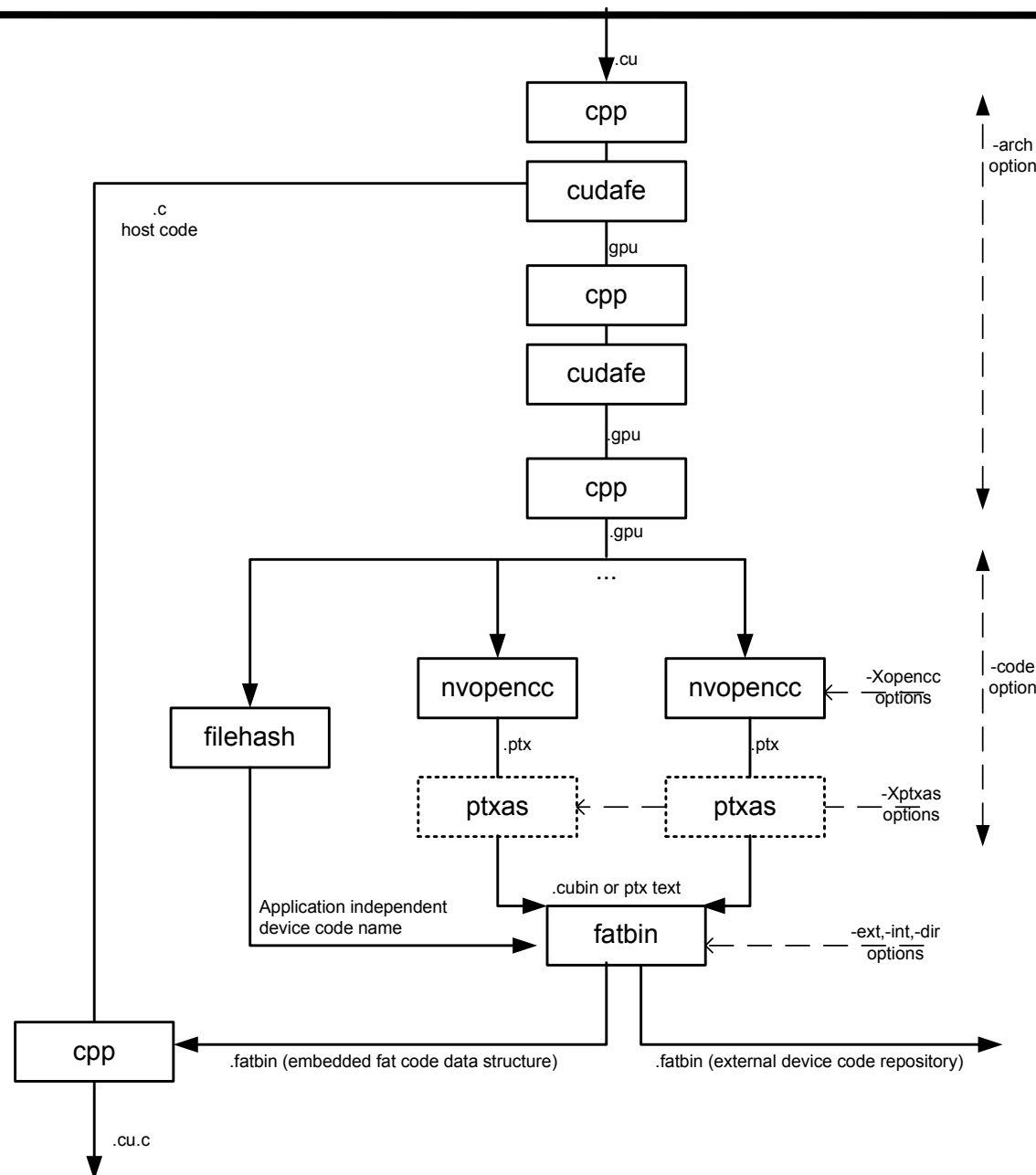
Обеспечивает процесс сборки CUDA-программ

- фронтенд над обычным компилятором (gcc, msvc) - для компиляции и линковки
- трансляция CUDA C в ассемблерный PTX-код
- трансляция PTX в CUBIN
- поддержка различных вычислительных характеристик

Удобна сборка с помощью CUDA SDK (makefile, vsproj)



# Стадии компиляции NVCC



# Совместимость версий

---

Compute Capability - 1.0, 1.1, 1.2, 1.3, 2.0, 2.1

CUBIN специфичен для конкретной архитектуры (значение CC).

PTX переносимый.

CUBIN обладает частичной прямой совместимостью – только для младших версий CC.

PTX обладает полной прямой совместимостью – для всех версий CC.

# Совместимость версий

---

Флаги компиляции:

- `-gencode arch=compute_xx` (версия PTX)
- `-gencode code=compute_xx` (версия CUBIN)
- `-arch=compute_xx` – упрощенное указание

```
nvcc -gencode=arch=compute_20,code=sm_20
```

Допустимо собирать несколько сочетаний CC.

Макрос `__CUDA_ARCH__` – минимальная версия.  
Версию CC устройства можно узнать при выполнении программы.

# Программная модель

---

Многопоточная модель

Несколько фаз исполнения:

- код хост-программы
- код функций ядер (ядер, kernels)

Ядро – функция на CUDA C, выполняемая на устройстве одним из потоков

# Программная модель

---

Ядра группируются в блоки потоков  
(размерности 1, 2, 3)

Блоки группируются в двумерную сеть

Ядро может однозначно определить:

- индекс ядра в блоке – `threadIdx`
- индекс блока в сети – `blockIdx`
- размер блока – `blockDim`
- размер сети – `gridDim`

*сравните: MPI*

# Программная модель

---

Ядро запускается из хост-программы с помощью специального синтаксиса.

```
const int N = 64;  
dim3 dimBlock(N,N);  
dim3 dimGrid(1,1);
```

```
MatrixMul<<<dimGrid, dimBlock>>>(A,B,R,N);
```

<<<...>>> – конфигурация запуска

Данные предварительно загружены в память GPU

# Программная модель

---

Данные между хостом и GPU передаются через глобальную память (шина PCI-E, небыстро)

Вызов ядра из хост-кода блокирующий

Процесс вызова ядра:

- Среда выполнения и драйвер загружают на GPU код ядра из CUBIN или PTX
- [Опционально] Асинхронная передача данных
- Ожидается окончание выполнения ядра на GPU
- [fermi] Конкурентное выполнение ядер

# Программная модель

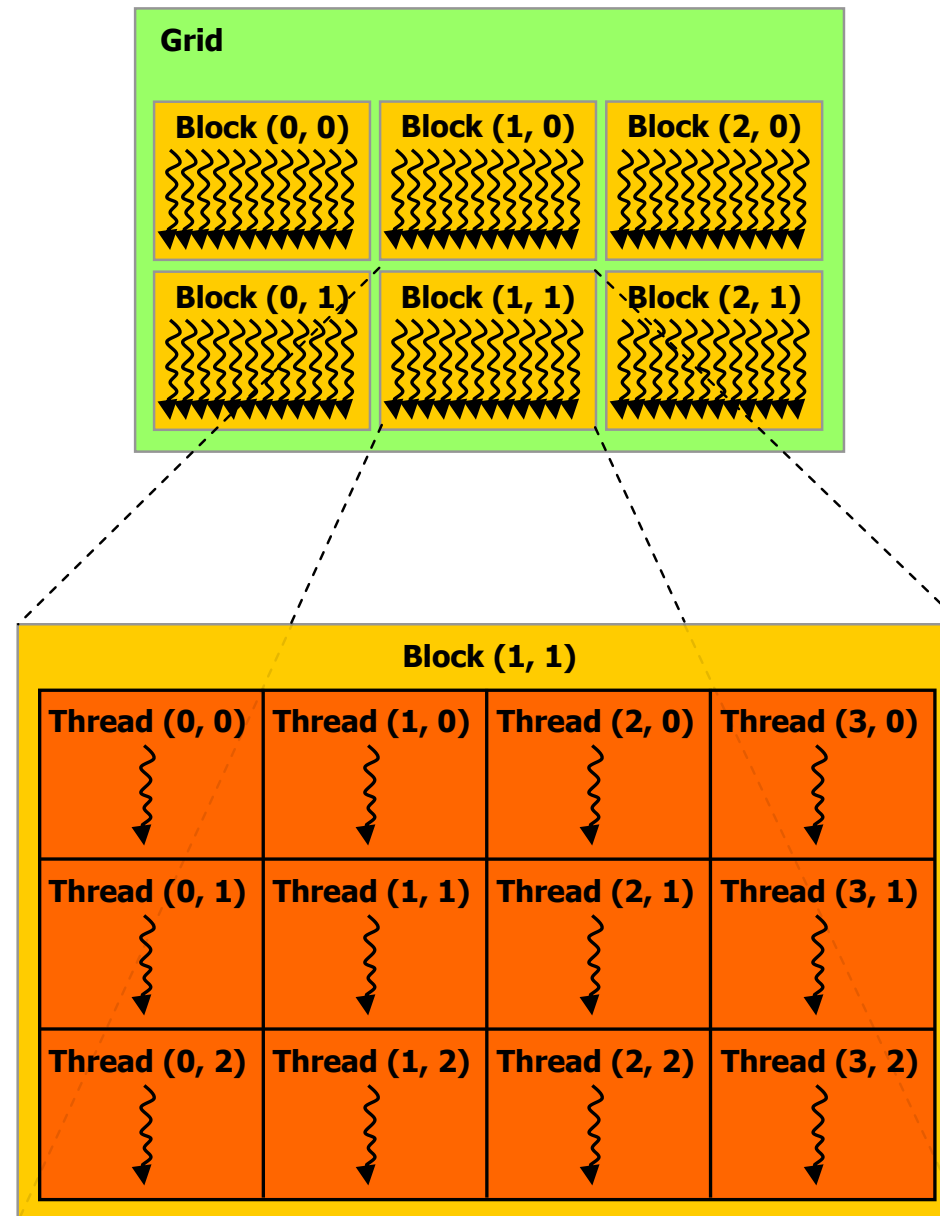
---

Примерные ограничения:

- размеры сети:  $65535 \times 65535$
- размеры блока:  $512 \times 512 \times 64$
- число блоков на МР: 8
- число потоков на МР: 768/1536
- число регистров на МР: 8000-32000



# Программная модель



# Планирование выполнения

---

Видеопроцессор: одна сеть

Мультипроцессор (MP): один или несколько блоков

Потоковый процессор (SP): один поток

Потоки группируются в варпы (32 потока) и планируются на выполнение мультипроцессором.

Мультипроцессор в один момент времени выполняет один или два (fermi) варпа.

Все потоки в пределах варпа параллельны

Варпы могут сменяться при ожидании памяти

# Планирование выполнения

---

Потоки легковесные

Обращение к «ближней» (локальная, общая, регистровая) памяти быстрое

Обращение к «дальней» (константная, текстурная, глобальная) памяти медленное

Планирование выполнения варпов снижает латентность памяти

# Планирование выполнения

---

## Взаимодействие потоков

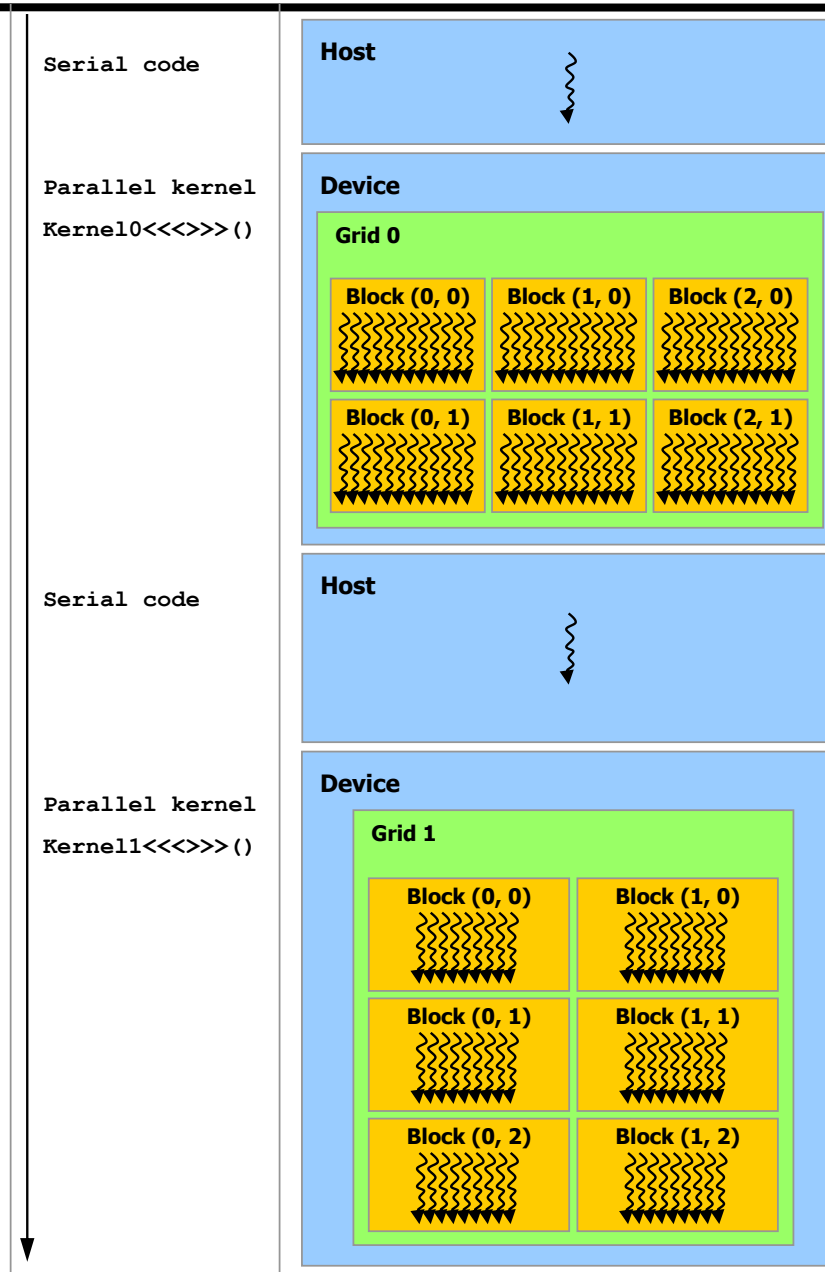
Потоки взаимодействуют только в пределах блока

Блоки в пределах сети не взаимодействуют

Передача данных через общую память на мультипроцессоре

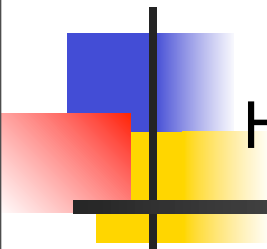
Барьерная синхронизация потоков

# Планирование выполнения



# Ресурсы этого курса

**<http://omniverse.ru/bmstu/cuda-rk6>**



Наш FTP: <ftp://cad.bmstu.ru/manichev/ZHUK-CUDA-6-kurs/>

---

NVidia: <http://nvidia.com/cuda>

CUDA C Programming Guide

CUDA C Best Practices Guide

CUDA Toolkit Reference Manual

Programming Massively Parallel Processors: <http://amzn.to/99ulEZ>

CUDA By Example: <http://amzn.to/9lbtEw>

Курс МГУ: <http://groups.google.com/group/cudacsmsusu>