Лекция
1. Многопоточные архитектуры
2. Введение в организацию GPU

ФИТ НГУ, 2 курс

ЭВМ и периферийные устройства

4.12.2013

Многопоточные архитектруры: Программная многопоточность

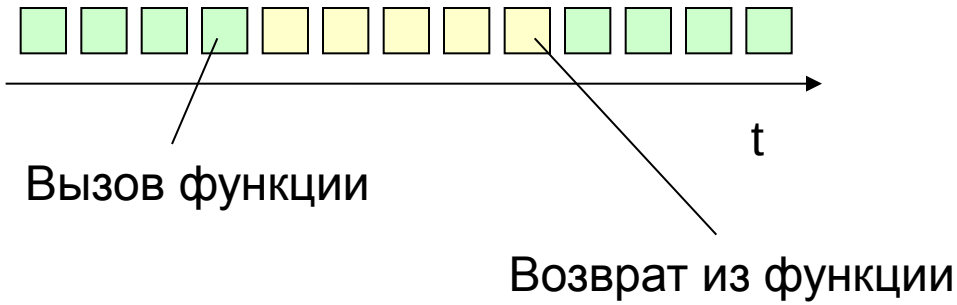
Поток исполнения (так же нить, thread)

Пример программы:

```
void func()
{
    ...
    return;
}
```

```
int main()
{
    ...
    func();
    ...
}
```

Ход исполнения программы:



- Существует единственный поток исполнения программы
- Последовательность потока нарушается командами условных и безусловных переходов

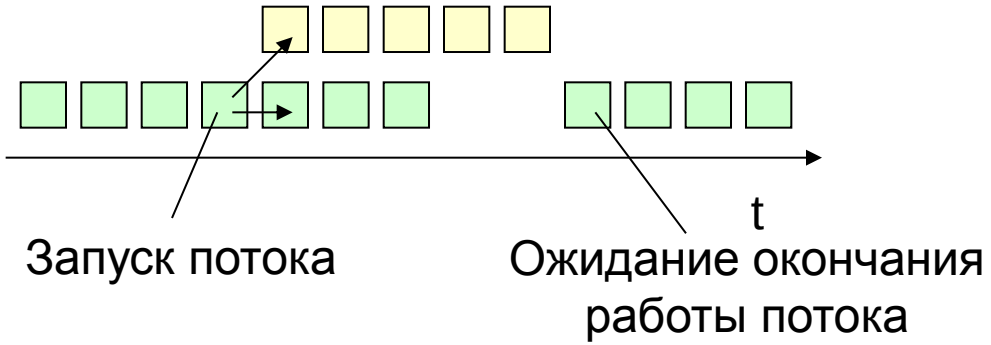
Многопоточное исполнение

Пример программы:

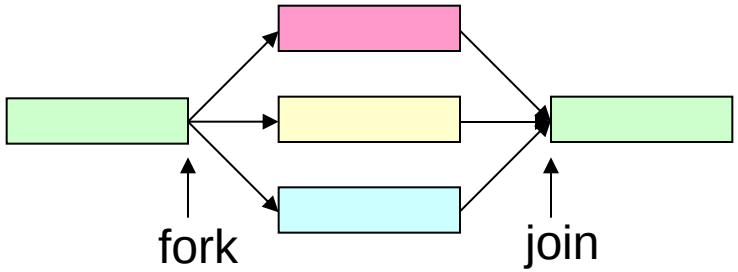
```
void func()
{
    ...
    return;
}
```

```
int main()
{
    ...
    start func();
    ...
    wait func();
    …
}
```

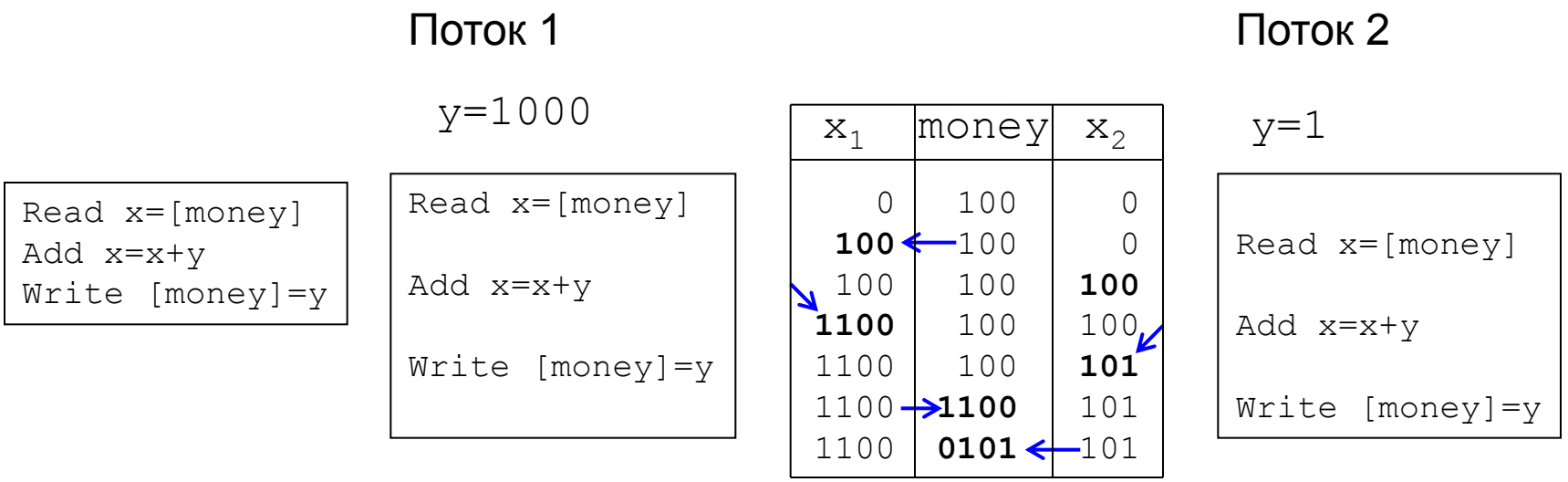
Ход исполнения программы:



- Новый поток исполнения может быть создан (шаблон параллельного исполнения **fork-join**)
- Порядок выполнения команд в потоках недетерминирован



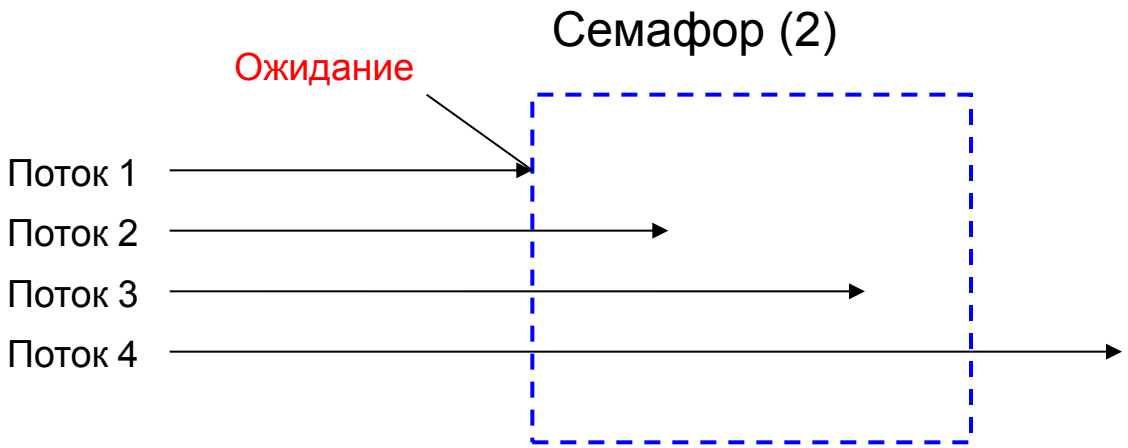
Проблема синхронизации потоков



- При одновременном доступе нескольких потоков к общим ресурсам могут возникать ошибки
- Требуются средства синхронизации работы потоков

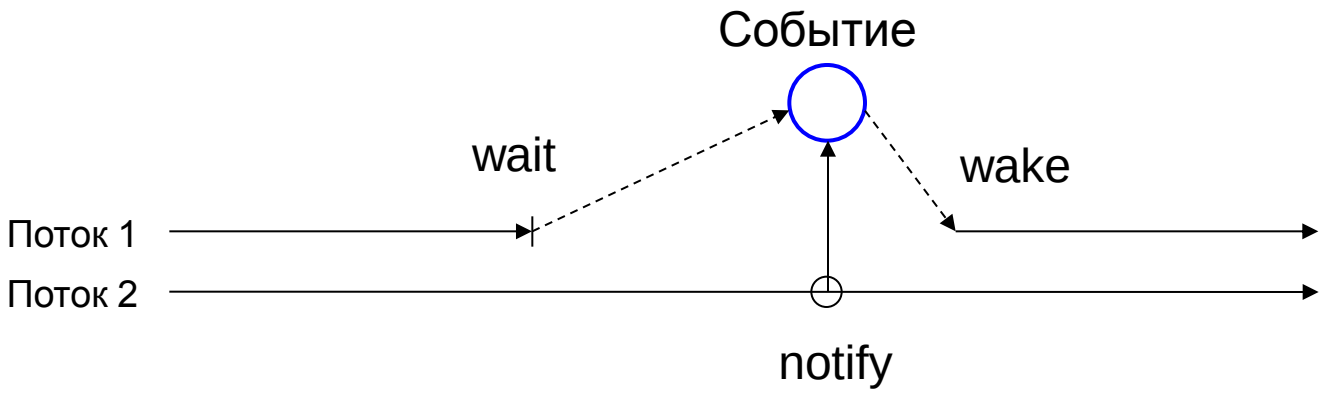
Примитивы для синхронизации работы потоков: **семафор**

- Семафор (Semaphore)
  - Для некоторой области кода ограничивается максимальное число потоков, которые могут в него одновременно зайти



Примитивы для синхронизации работы потоков: **событие**

- Событие (Event)
  - Объект, на котором могут **ожидать** один или более потоков
  - Другой поток может **активировать** событие
  - После активации ожидающие потоки продолжают исполнение



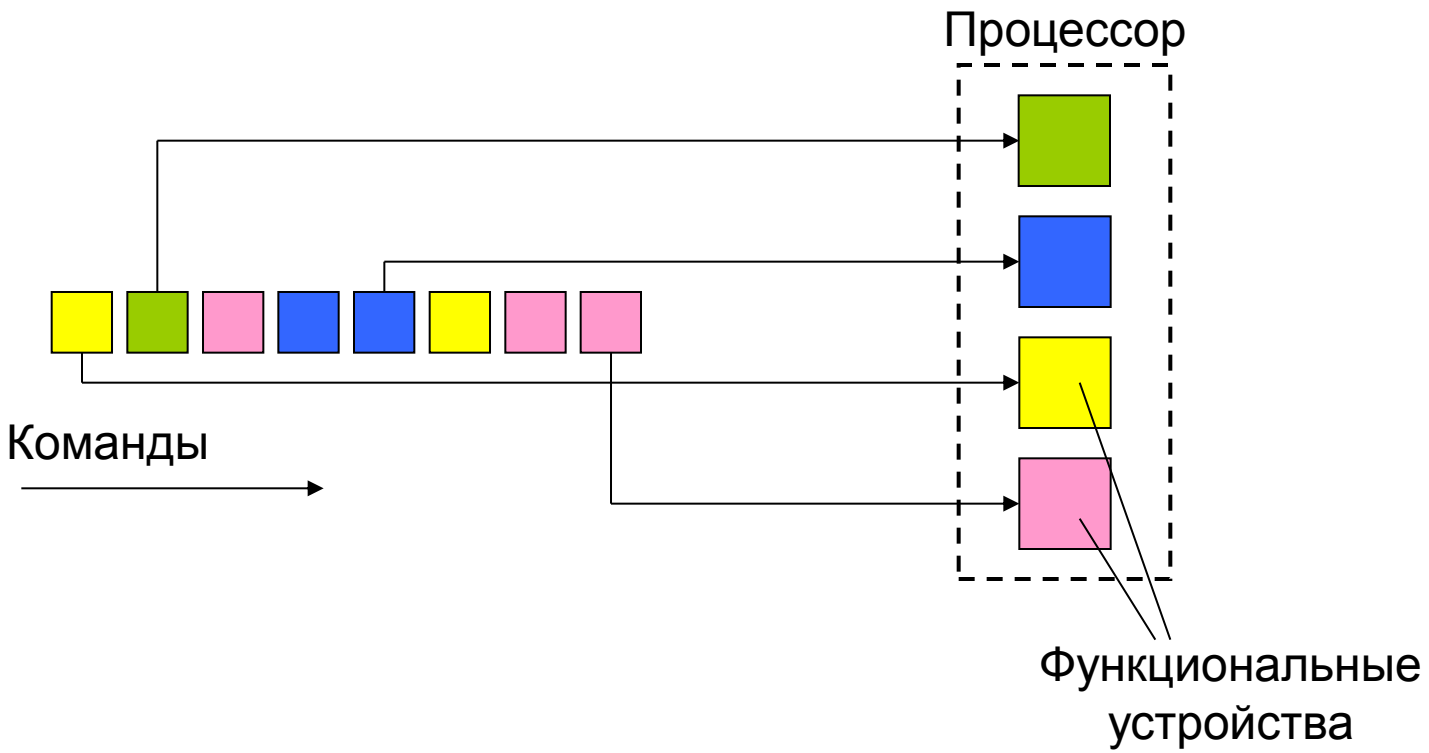
Многопоточные архитектуры: Аппаратная многопоточность

Пути увеличения производительности

- Наращивание тактовой частоты
- Реализация параллелизма на уровне команд (**ILP**: **I**nstruction **L**evel **P**arallelism)
- Реализация параллелизма на уровне нитей (**TLP**: **T**hread **L**evel **P**arallelism)

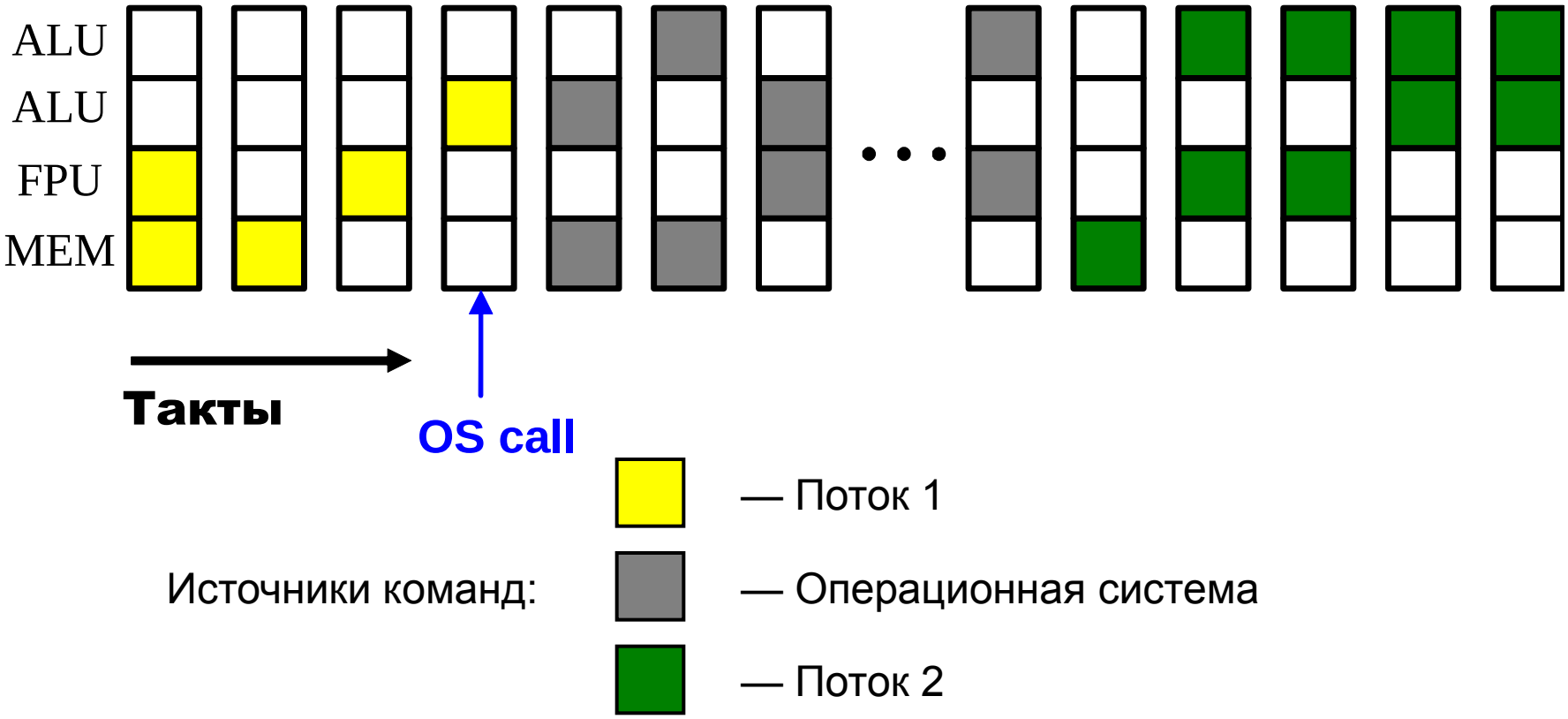
Идея TLP: Если нельзя полностью загрузить процессор одной задачей (нитью), то можно увеличить загрузку выполнением других задач (нитей)

Параллелизм на уровне команд



Программная многопоточность

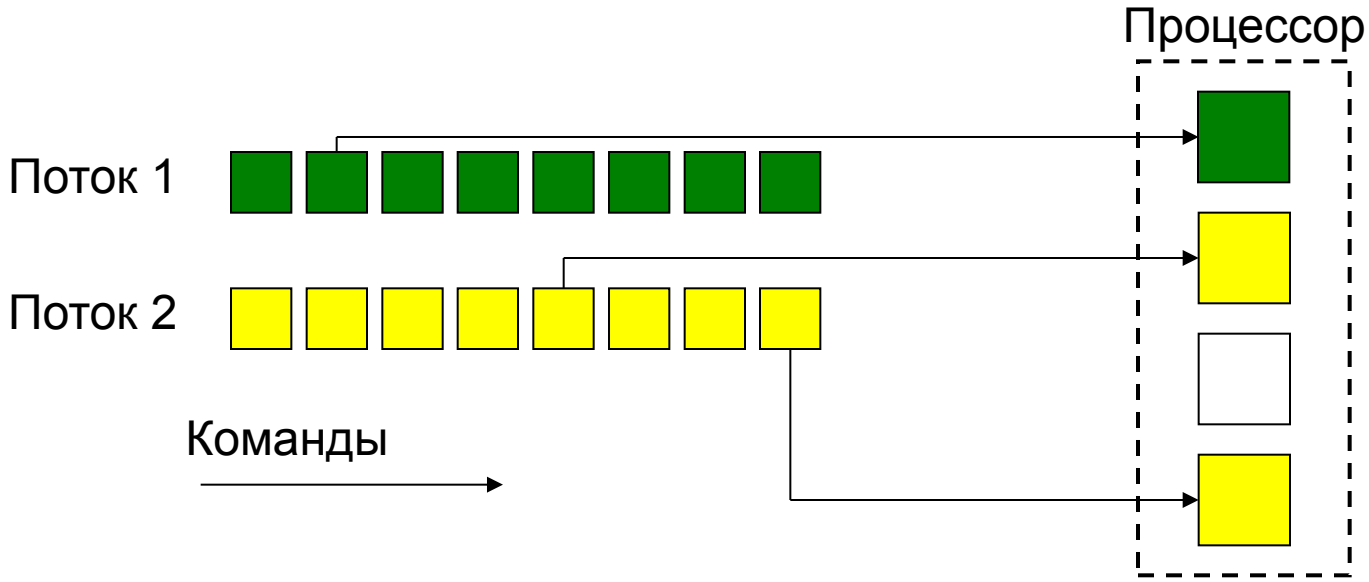
- Переключение контекстов потоков средствами операционной системы



Достоинства и недостатки программной многопоточности

- Достоинство:
  - Эффект параллельной работы нескольких потоков
- Недостатки:
  - Отсутствие реального параллелизма
  - Относительно большие накладные расходы на переключение потоков

Идея аппаратной многопоточности



- Не переключать потоки средствами операционной системы, а средствами процессора выбирать на исполнение команды из различных потоков

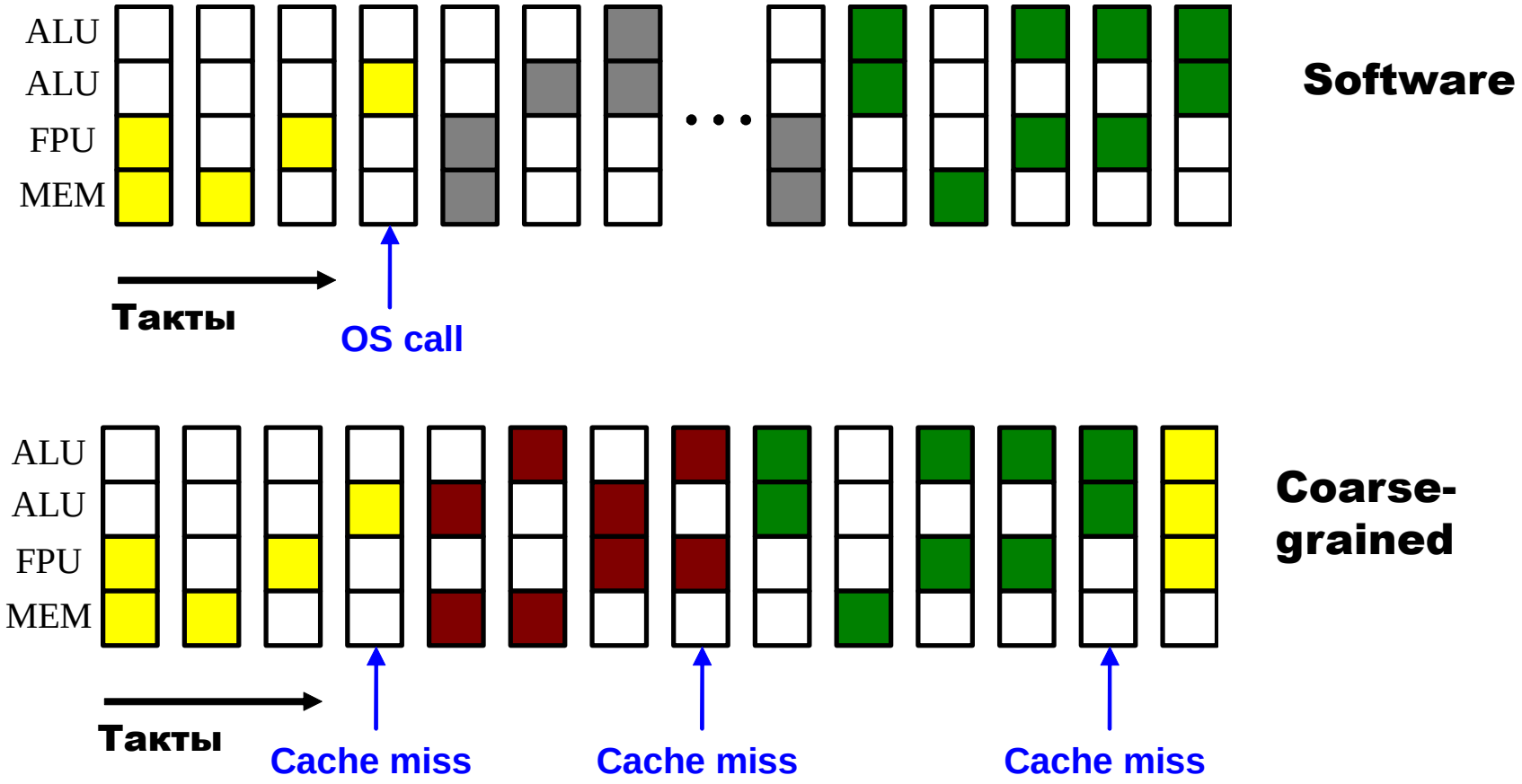
Типы аппаратной многопоточности для одноядерных архитектур

- Крупнозернистая многопоточность (Coarse-grained multithreading)
- Мелкозернистая многопоточность (Fine-grained multithreading)
- Одновременная многопоточность (Simultaneous multithreading)

Крупнозернистая многопоточность (Coarse-grained MT)

- 2 и более аппаратных контекстов
  - Регистры общего назначения
  - Счетчик команд
  - Буфер выборки инструкций
- Одновременно **не более 1 нити**
- Аппаратное переключение контекстов при **прерывании**

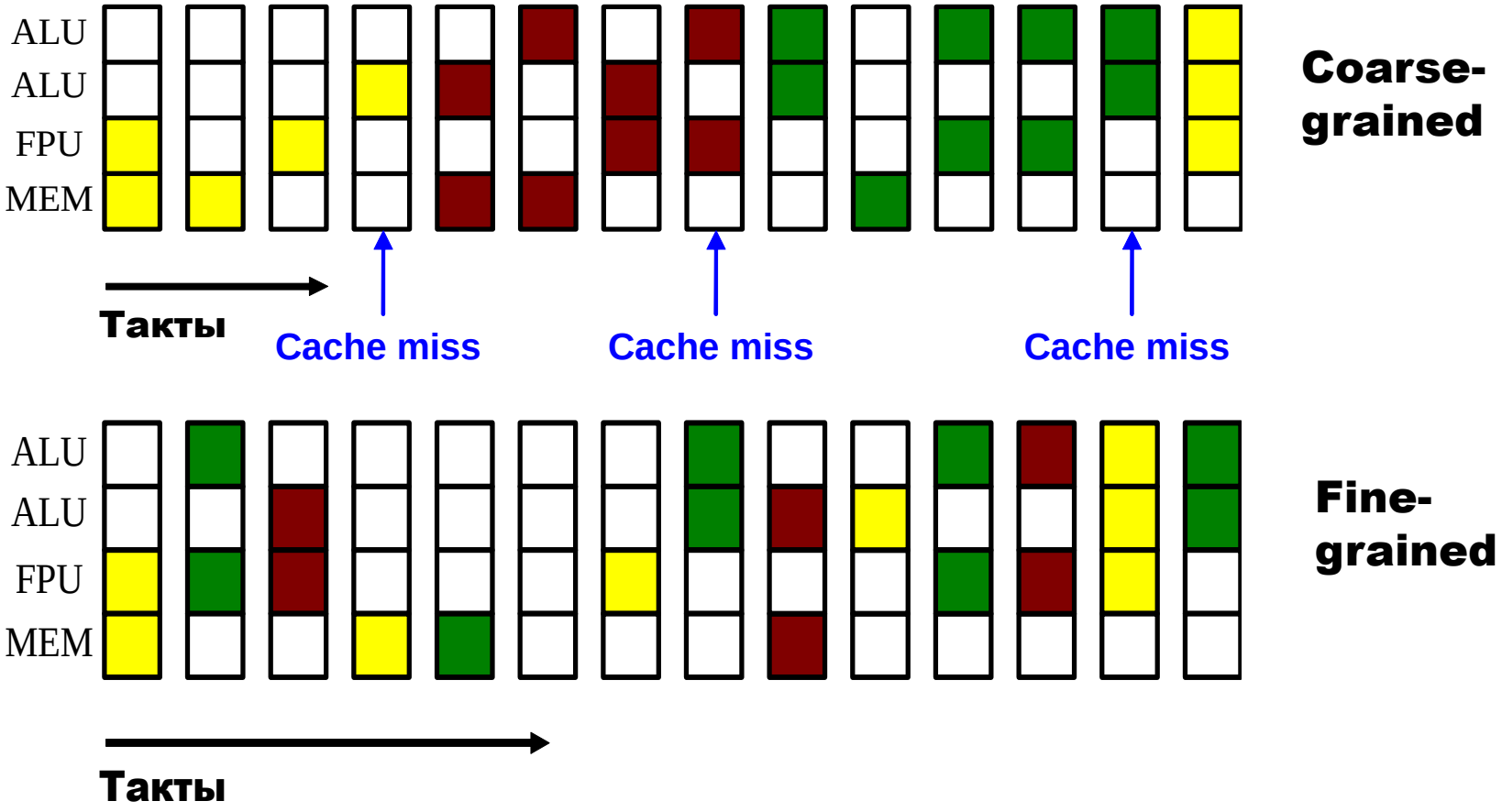
Крупнозернистая многопоточность



Мелкозернистая многопоточность (Fine-grained MT)

- 2 и более аппаратных контекстов
  - Регистры общего назначения
  - Счетчик команд
- Одновременно **не более 1 нити**
- Аппаратное переключение **на каждом такте**

Мелкозернистая многопоточность



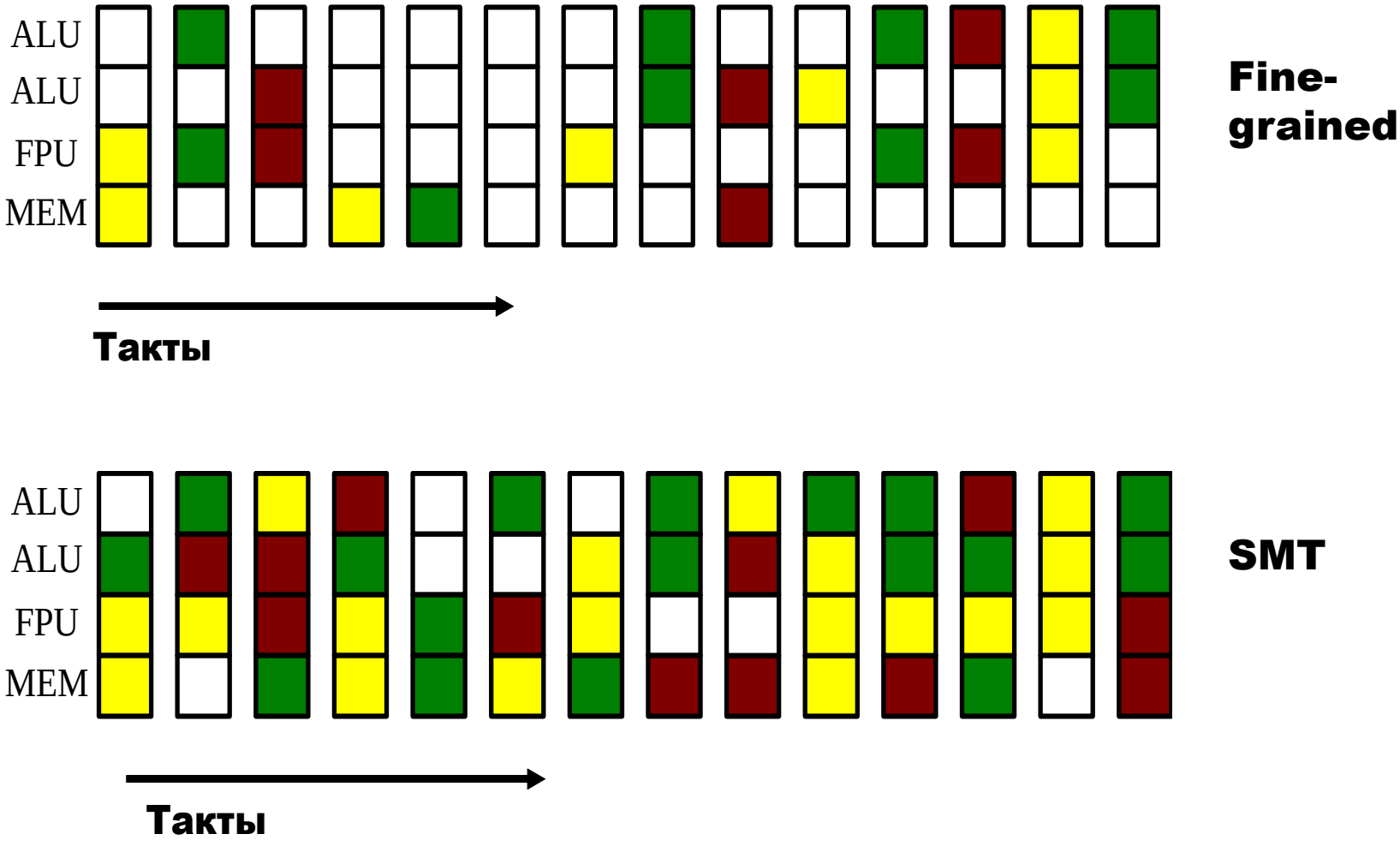
Достоинства и недостатки крупнозернистой и мелкозернистой многопоточности

- Достоинства:
  - Моментальное переключение потоков
  - Сокрытие задержек длинных операций
- Недостатки:
  - Ограниченное число аппаратных контекстов
  - Нет параллельного исполнения команд

Одновременная многопоточность (Simultaneous MT)

- 2 и более аппаратных контекстов
  - Регистры общего назначения
  - Буфер выборки инструкций
  - Буфер переупорядочивания
  - Стек возврата
- Привязка команд и нитей
- **До нескольких нитей одновременно**
- Контексты активны, **переключения не происходит**

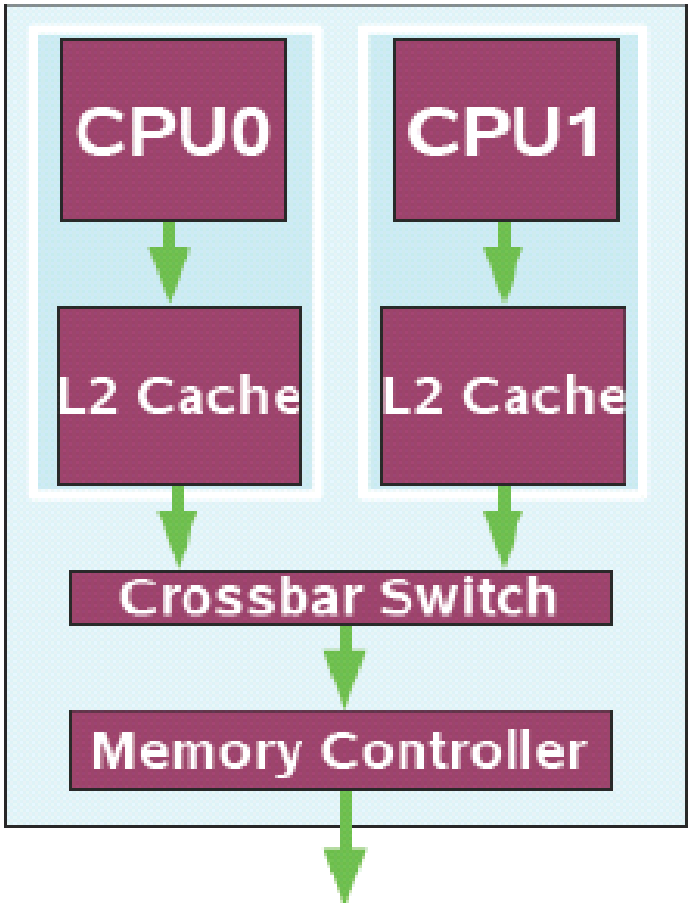
Одновременная многопоточность

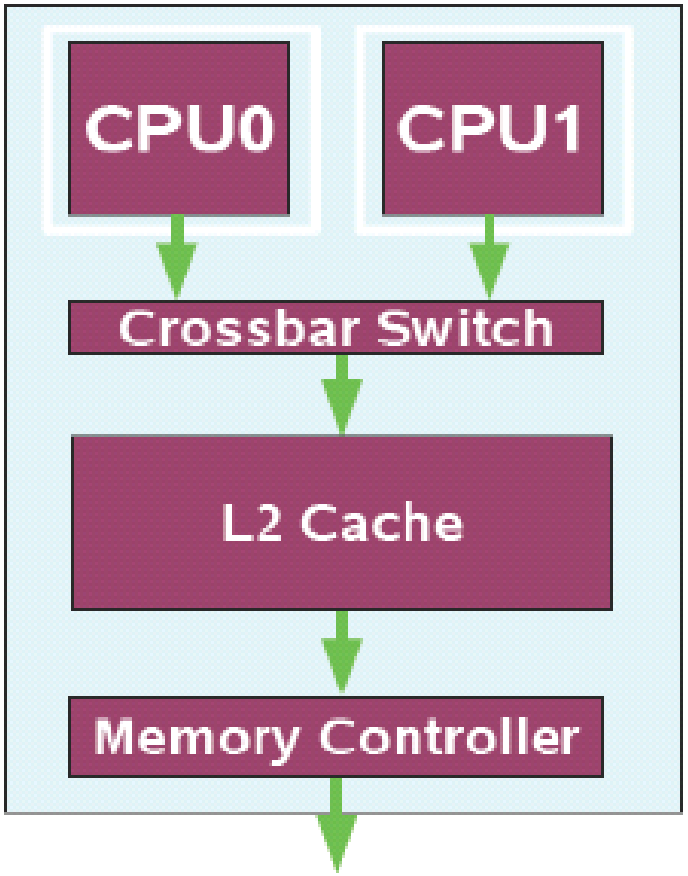


Многоядерность vs многопроцессорность (CMP – Chip Multiprocessors)

- Меньше стоимость при той же производительности
- Выше скорость обмена между ядрами
- Меньше места, меньше выделяемого тепла, меньше потребляемая мощность

Структура CMP-процессора





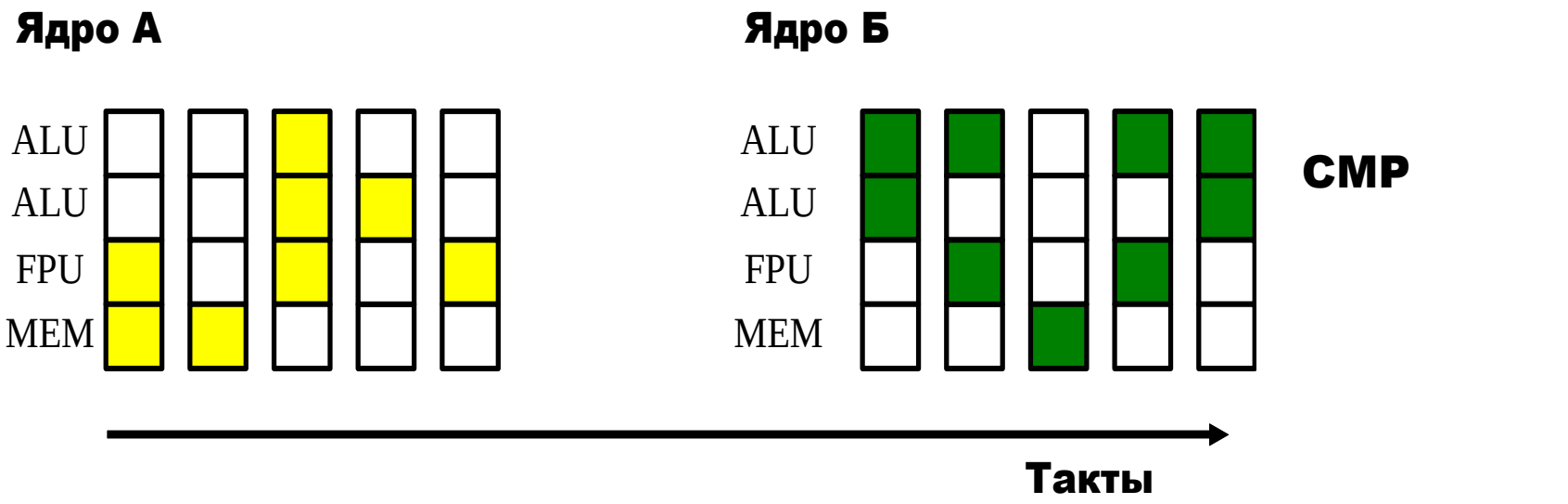
Многопоточность на основе CMP

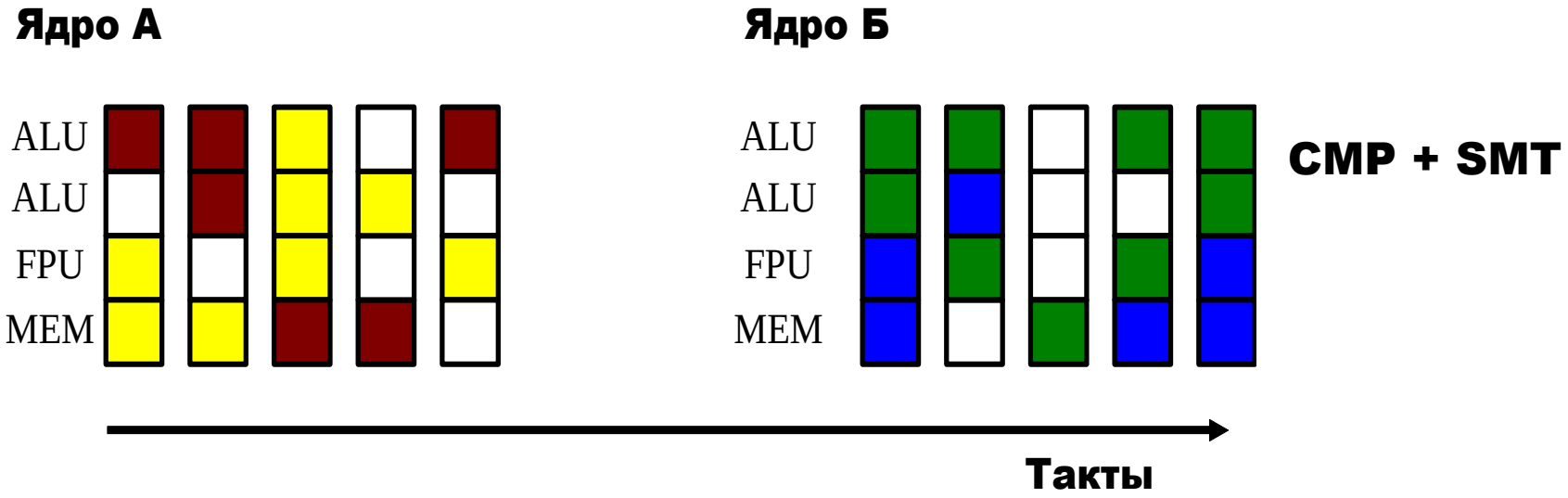
- Простая логика, один поток на ядро
- Масштабируемость за счет локальности
- Условная многопоточность (Speculative MultiThreading) для ускорения последовательного кода

Распределение ресурсов

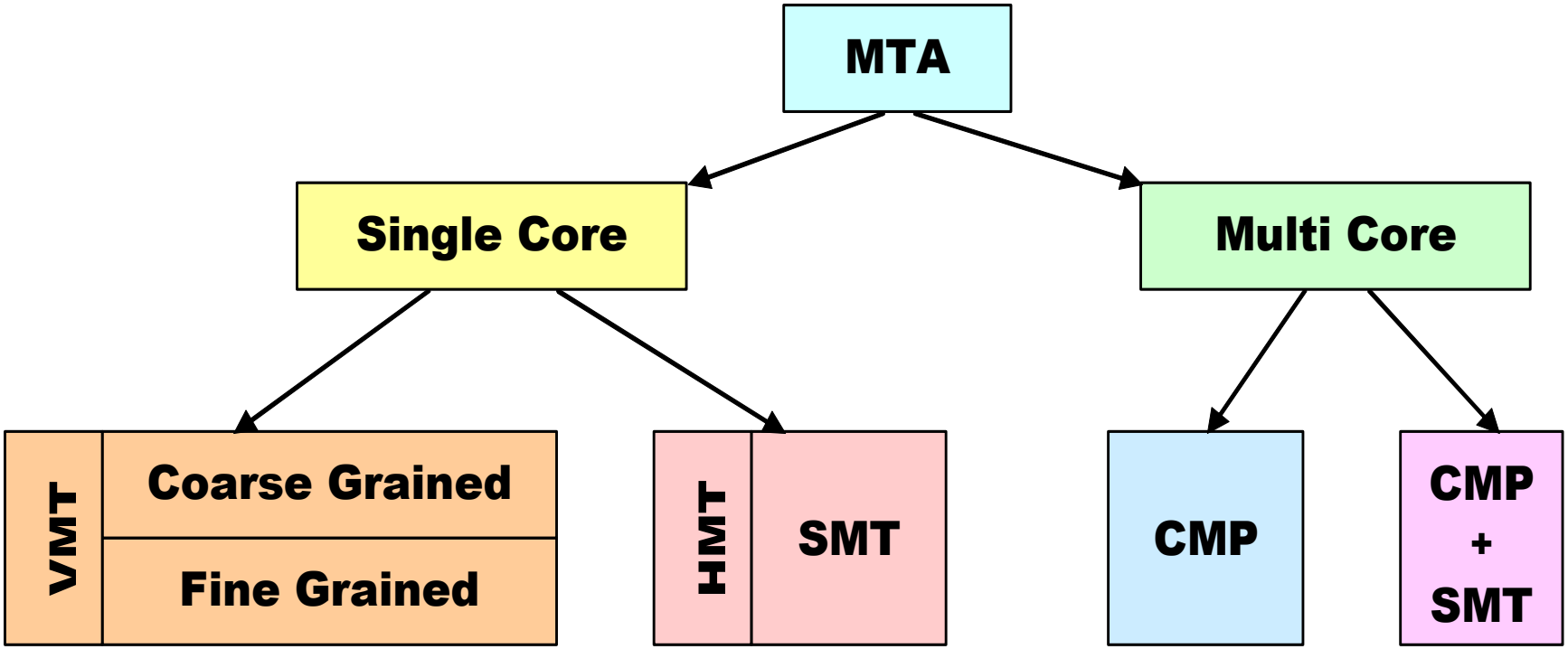
MT	**Общие ресурсы**	**Переключение контекстов**
нет	Все	Явное переключение ОС
Fine-grained	Все кроме регистров и управляющей логики	На каждом такте
Coarse-grained	Все кроме регистров, управляющей логики и буферов выборки инструкций	При остановке конвейера
SMT	Все кроме регистров, управляющей логики, буферов выборки инструкций, стека адресов возврата, буфера переупорядочивания, очереди записи и т.п.	Все контексты активны, переключения нет
CMP	Кэш второго уровня, системное межсоединение ядер	Все контексты активны, переключения нет

CMP+SMT


Ядро А
Ядро Б
ALU
ALU
FPU
MEM
ALU
ALU
FPU
MEM
CMP
Такты


Ядро А
Ядро Б
ALU
ALU
FPU
MEM
ALU
ALU
FPU
MEM
CMP + SMT
Такты

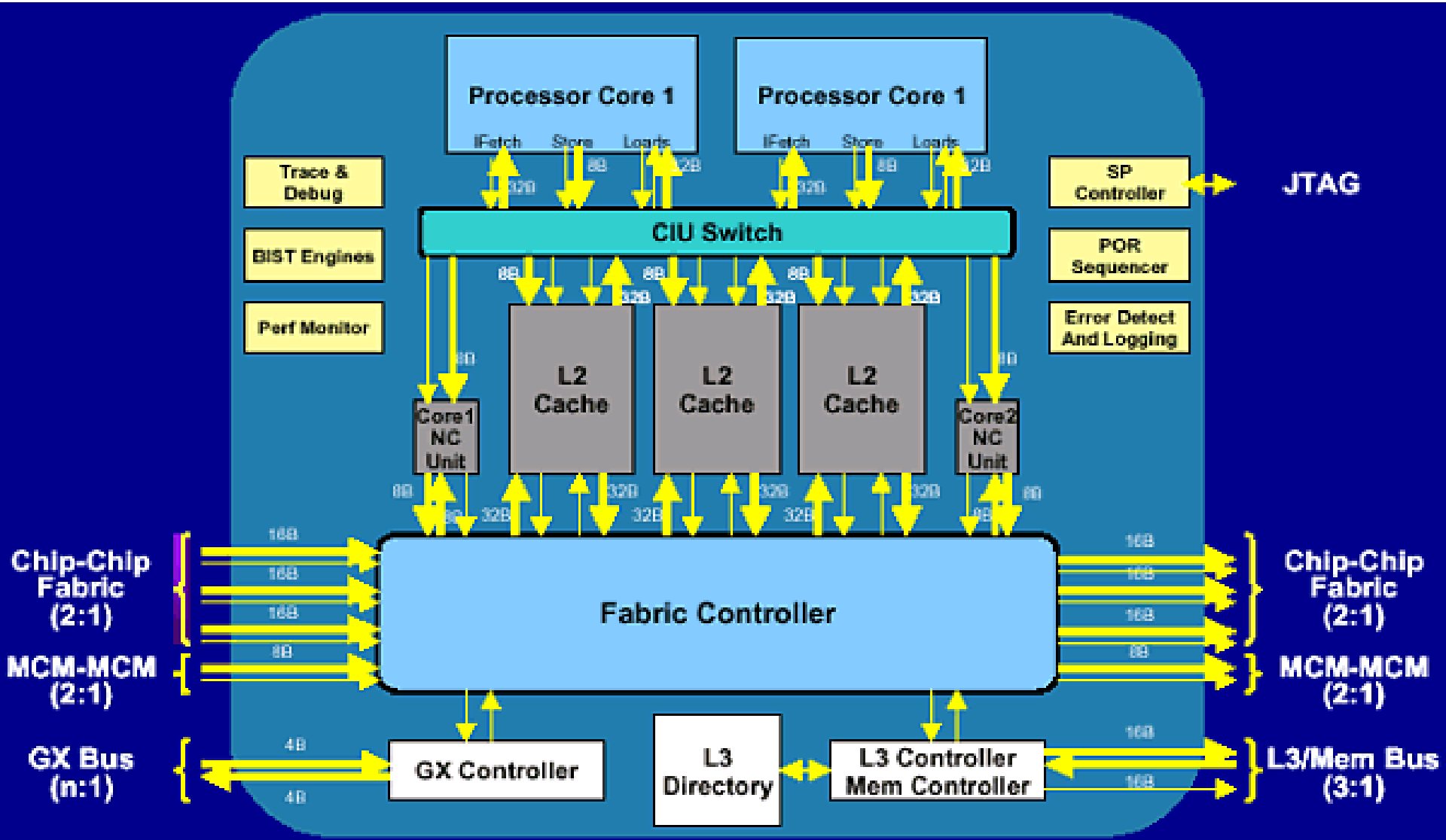
Мультитредовые архитектуры



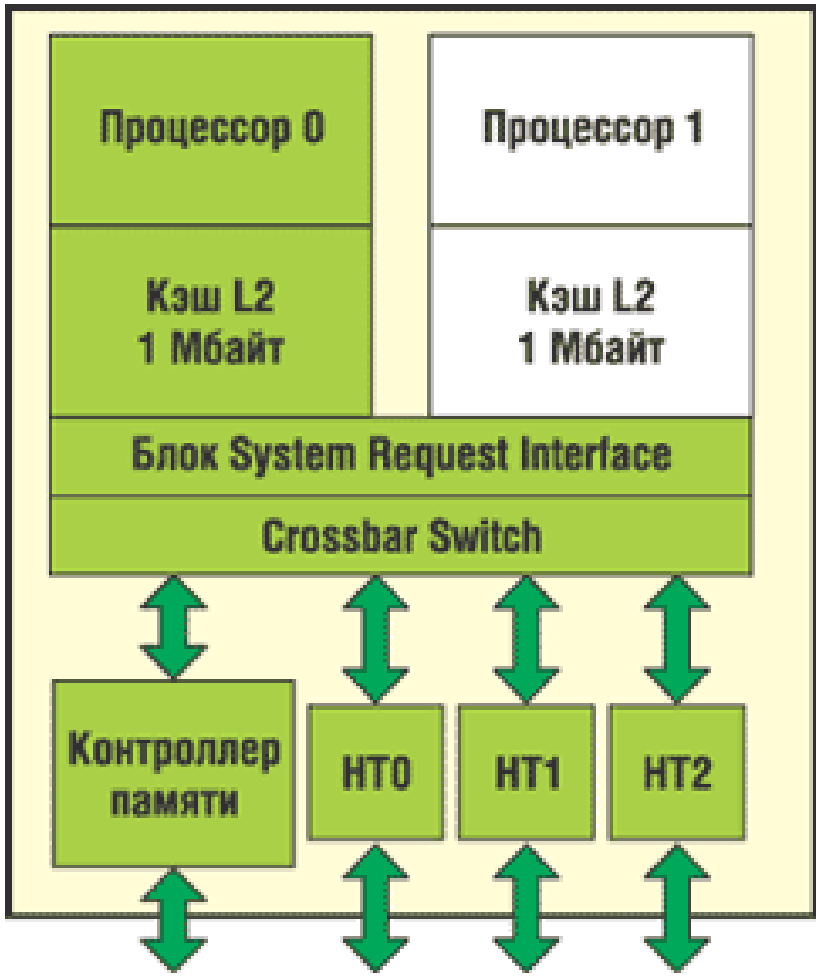
Примеры мультитредовых архитектур

- **CMP**
  - POWER 4
  - Barcelona
  - Clovertown
  - Opteron
- **SMT**
  - Alpha 21464
  - Pentium 4
  - Itanium 2
- **CMP + SMT**
  - POWER 5
  - UltraSPARC T2
  - Keifer

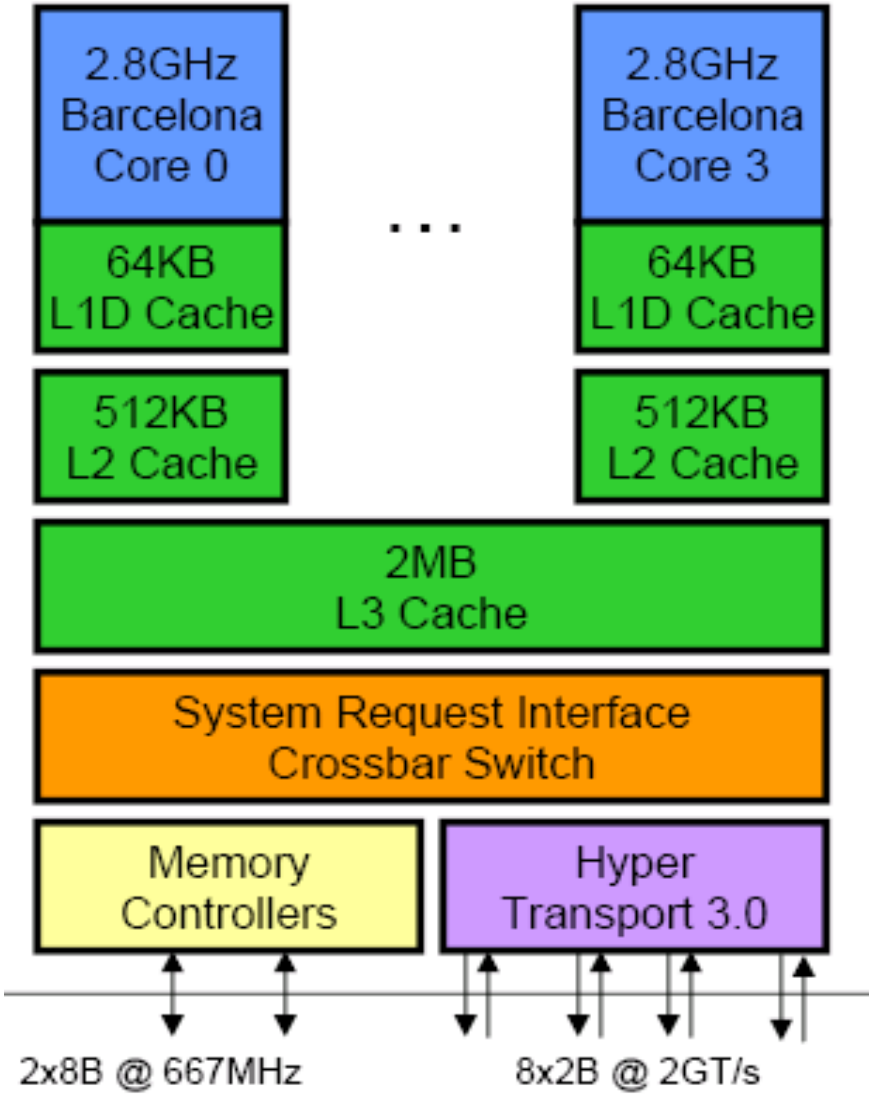
POWER 4
(2 ядра)



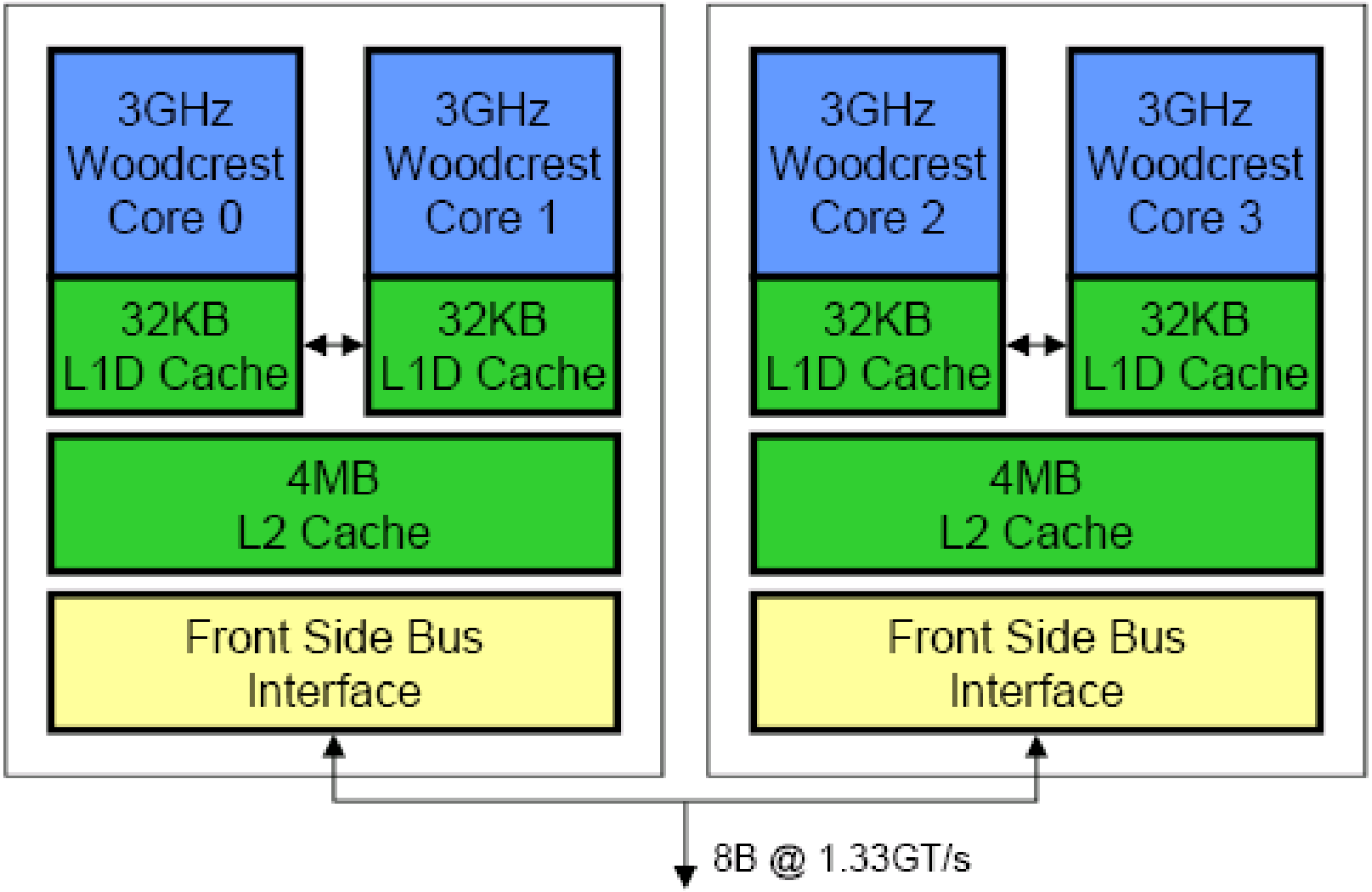
Dual core Opteron (2 ядра)



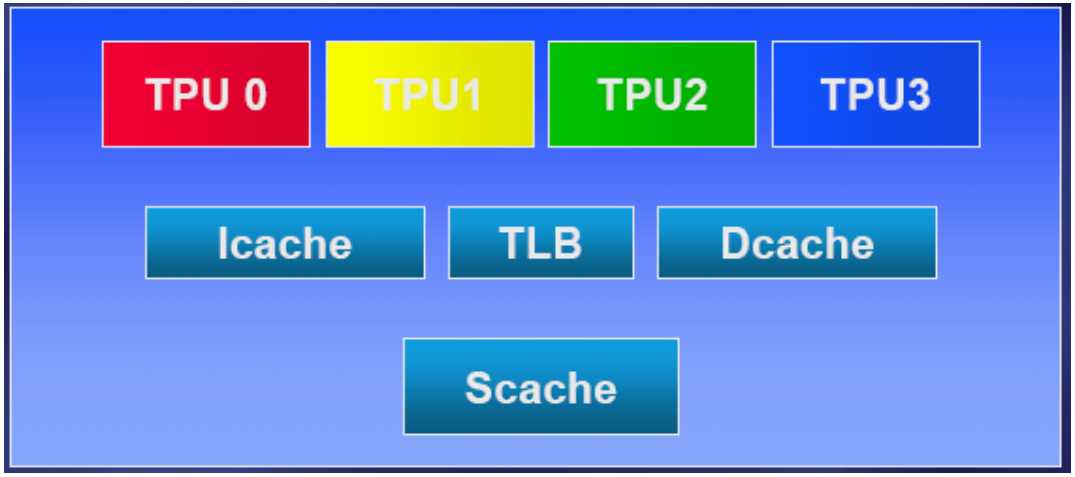
Barcelona
(4 ядра)

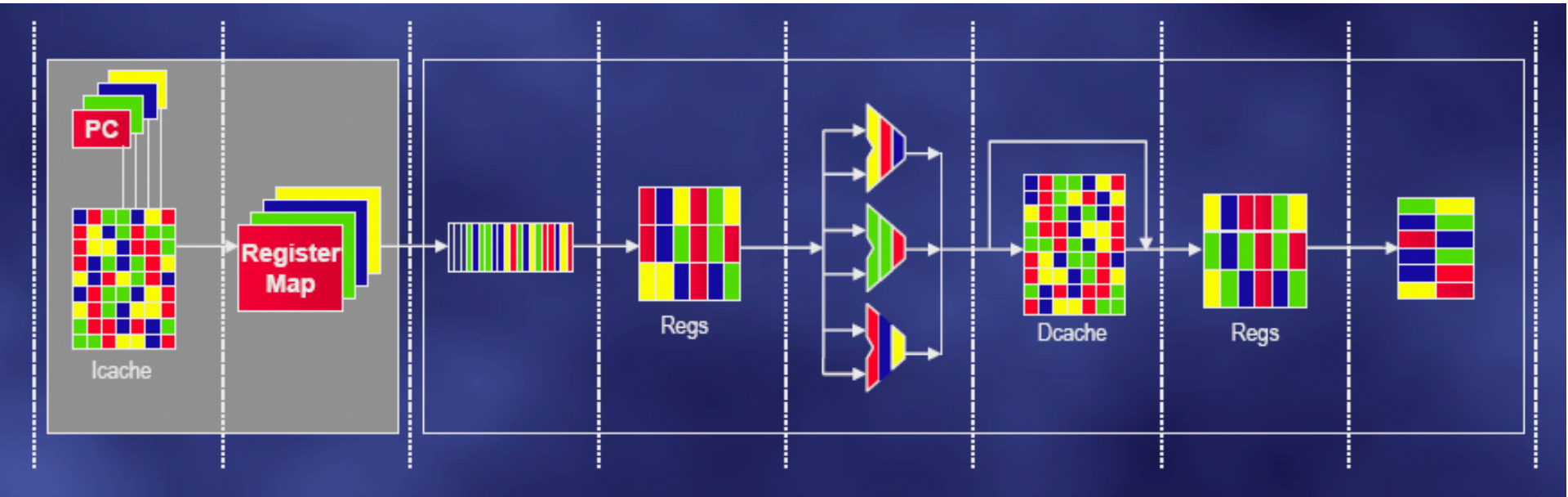


Clovertown
(4 ядра)

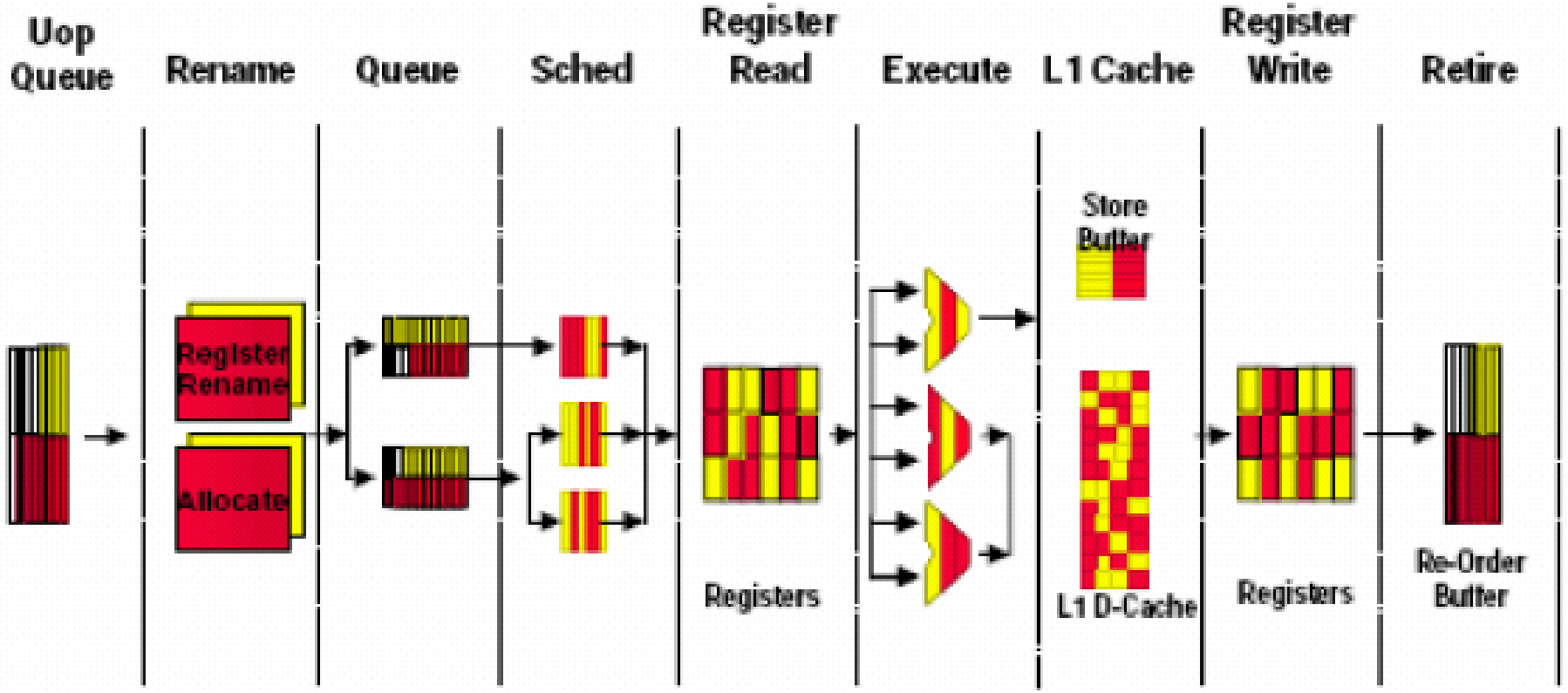


Alpha 21464
(4 потока)

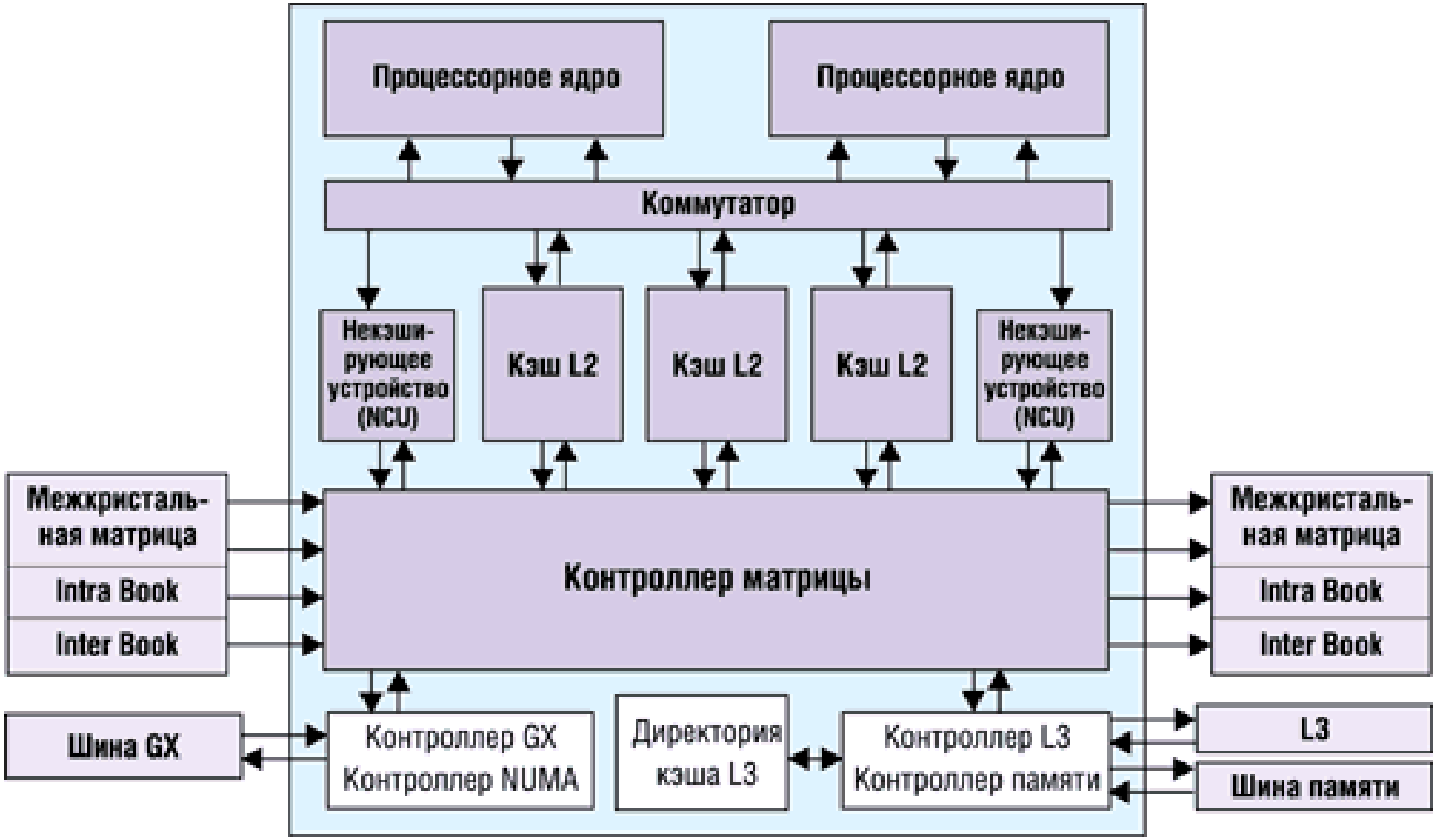




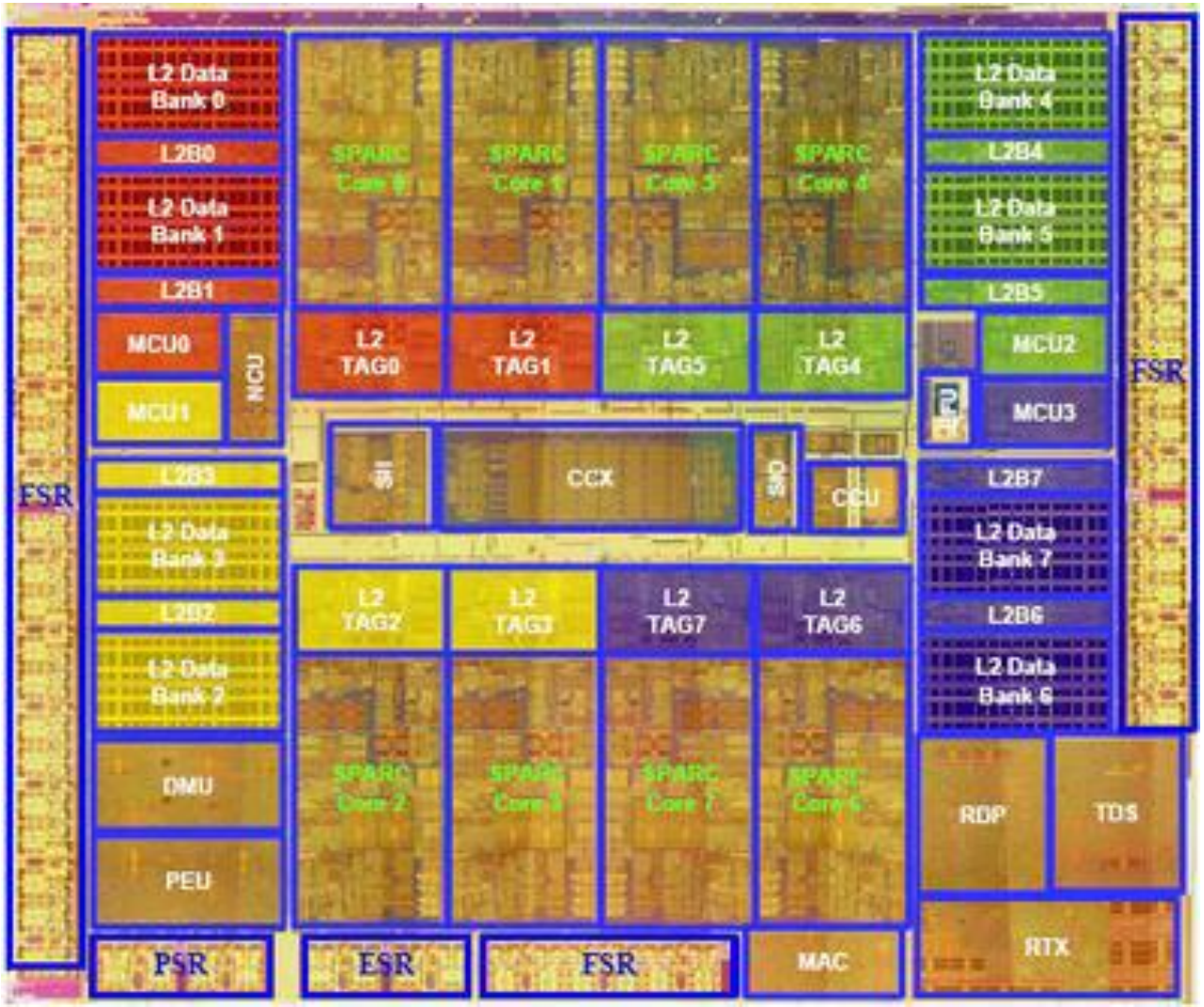
Pentium 4
(2 потока)



POWER 5
2 ядра 2 нити



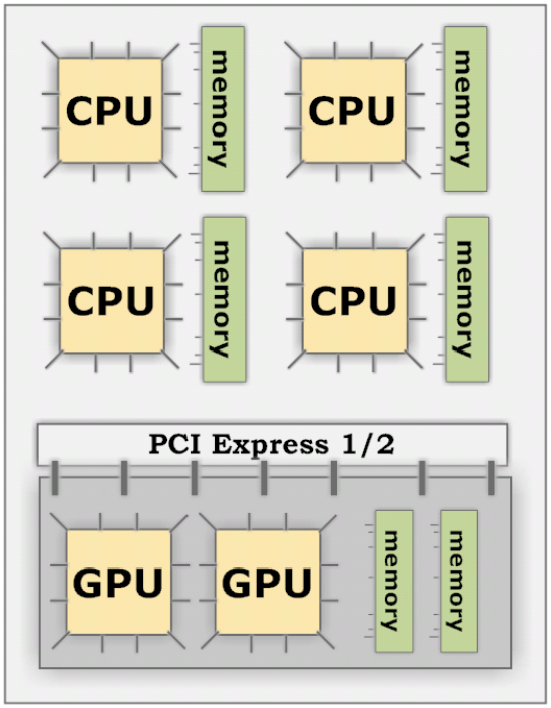
UltraSPARC T2
(8 ядер x 8 потоков)



Введение в организацию GPU

GPU: **G**raphical **P**rocessing **U**nit

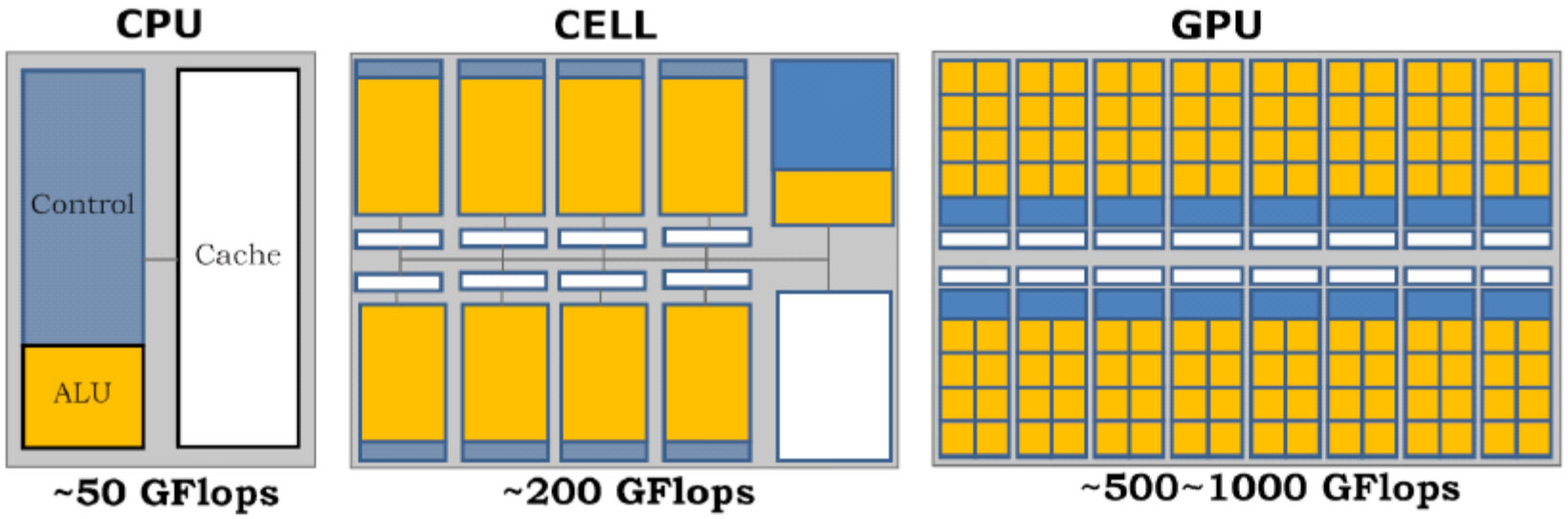
- **GPU** – отдельное устройство
  - Процессор
  - Набор команд
  - Память
  - Единое адресное пространство
  - Шина процессоры-память



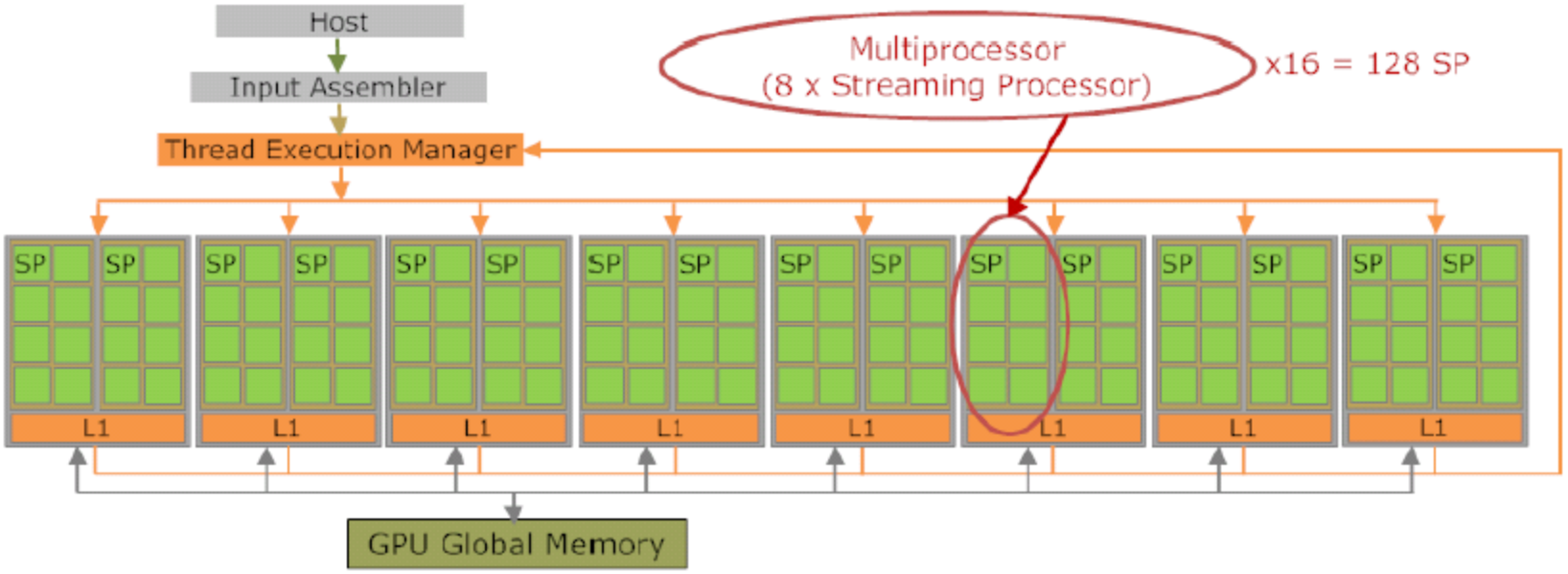
GPU: Graphical Processing Unit (продолжение)

- Процессор GPU имеет свою архитектуру и набор команд
- Программа, исполняющаяся на GPU, не может связаться с программой на хосте
- Программа, исполняющаяся на GPU, не имеет доступа в память хоста
- Хост имеет доступ в память GPU через шину PCI Express
- Видеокарта используется как сопроцессор
- На хосте может быть столько видеокарт, сколько позволяет материнская плата

CPU vs CELL vs GPU



Микроархитектура GPU

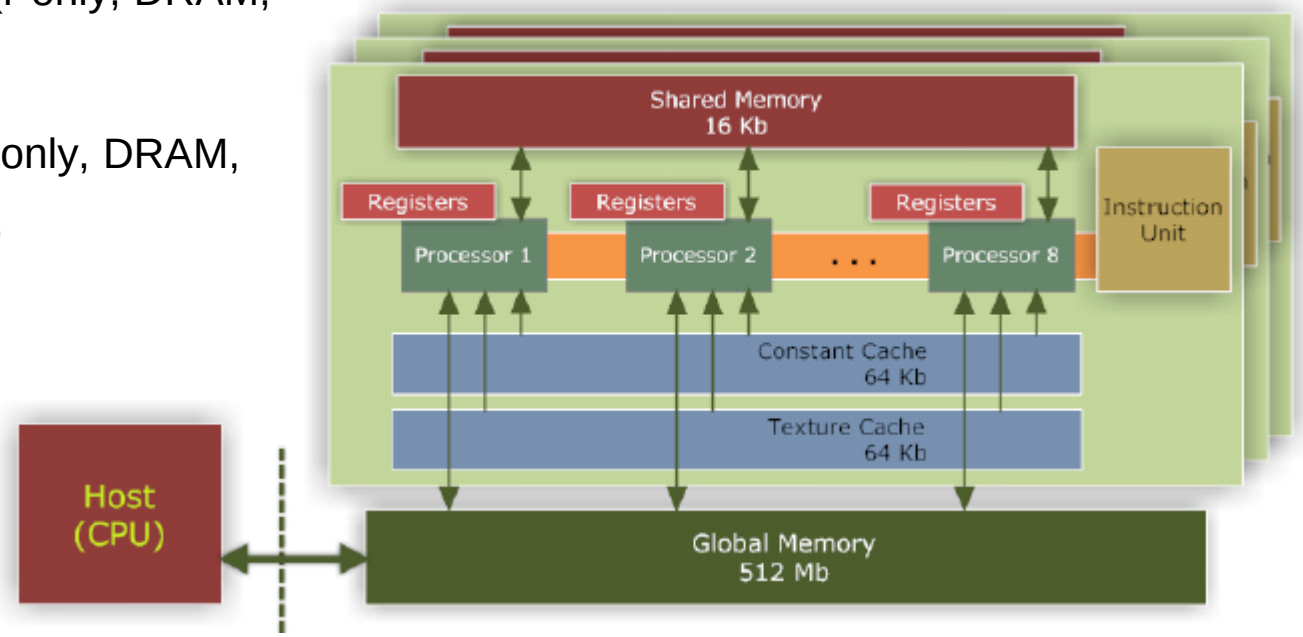


Некоторые сокращения

- **GPGPU**
  - **G**eneral **P**urpose computations on **GPU**
- **CUDA**
  - **C**ompute **U**nified **D**evice **A**rchitecture (программно-аппаратная архитектура для вычислений на GPU Nvidia GeForce 8400 и более поздних

Иерархия памяти GPU

- **Registers** (r/w, per-thread)
- **Local memory** (r/w, DRAM, uncached, per-thread)
- **Shared memory** (r/w per-block)
- **Global memory** (r/w, DRAM, uncached, per-context)
- **Constant memory** (r-only, DRAM, cached, per context)
- **Texture memory** (r-only, DRAM, cached, per-context)



Иерархия памяти GPU (продолжение)

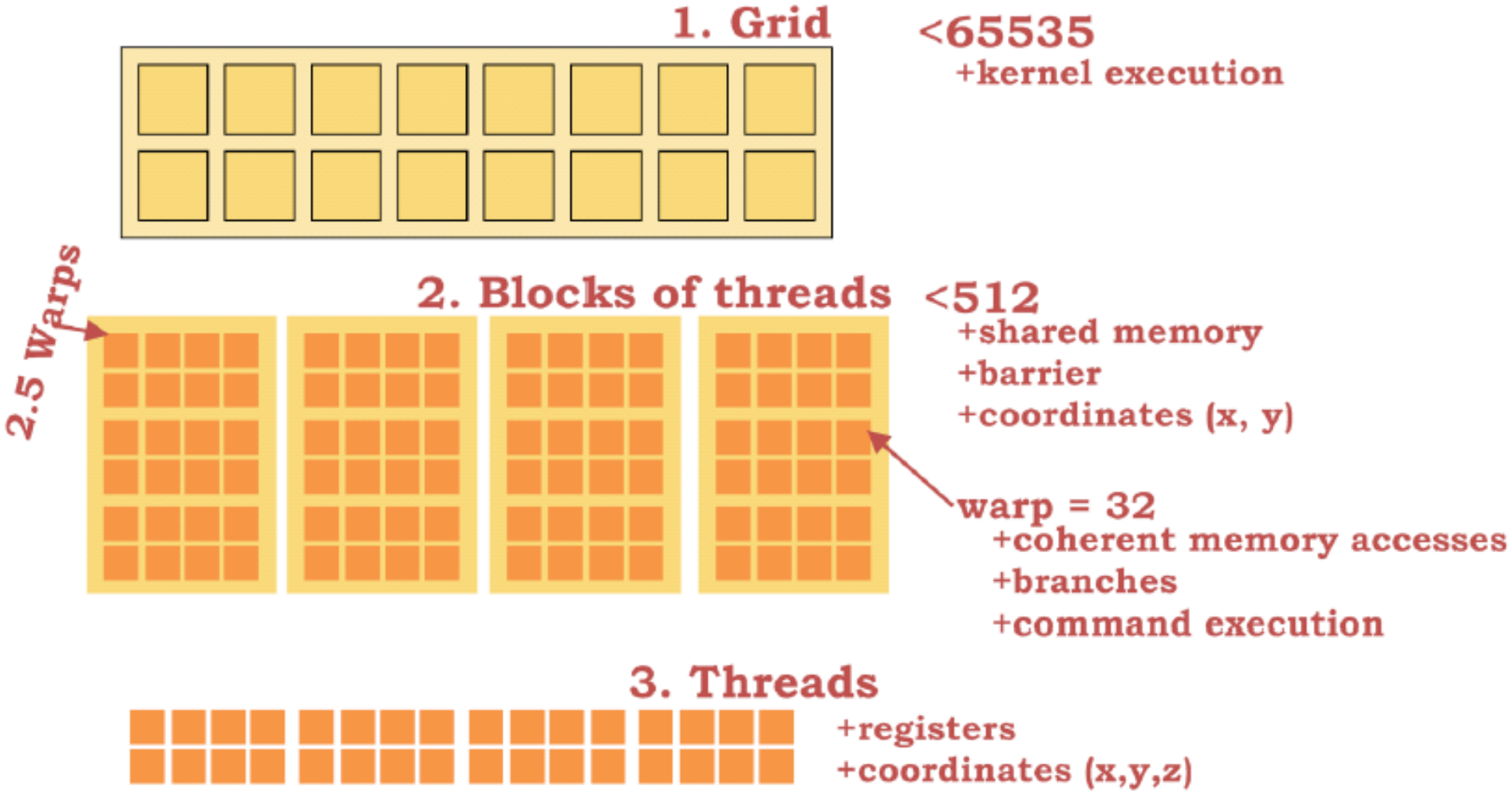
- Файл 32-битных регистров делится между процессорами
- Размер файлового регистра – 8192 или 16384 регистра
- Резделяемая память мультипроцессора доступна всем его процессорам (и только им)
- Разделяемую память мультипроцессора можно назвать «управляемым кэшем»
- Размер разделяемой памяти – 16 Кбайт
- Глобальная память – одна на всю видеокарту
- В локальной памяти размещаются переменные, которым не хватило физических регистров
- С хоста можно записывать в глобальную, константную и текстурную памяти

CUDA

Программно-аппаратная архитектура

- Для вычислений общего назначения
- Язык разработки является диалектом языка C
- Compiler, profiler, emulator

Программная модель: иерархия потоков

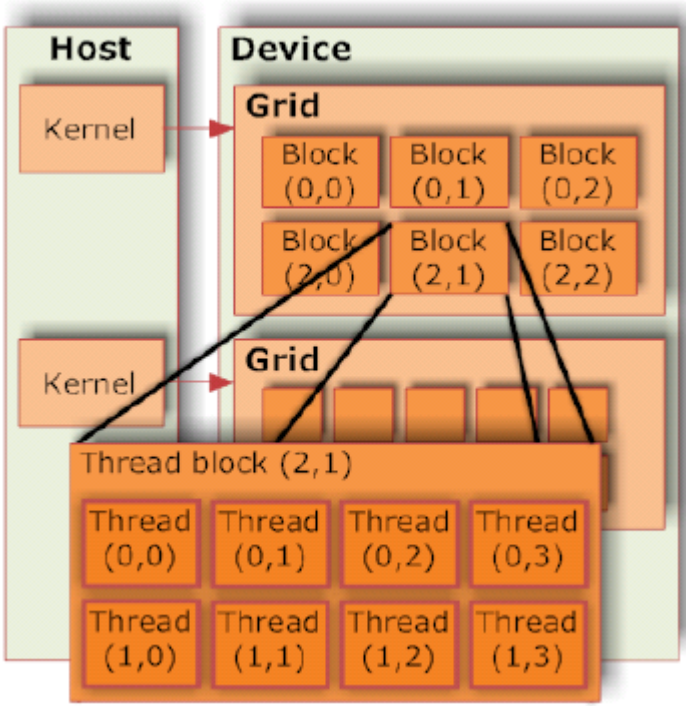


Программная модель: иерархия потоков

- Множество потоков организовано в иерархию, каждый уровень имеет свои особенности
- На самом нижнем уровне – поток, у него есть только регистры
- Каждый поток имеет свои координаты в блоке потоков
- Потоки одного блока могут обмениваться данными через разделяемую память и синхронизоваться с помощью барьеров
- Все блоки потоков в одной сетке имеют одни и те же размеры
- Подуровень warp'ов появляется в момент исполнения: одна инструкция исполняется над целым warp'ом на 8 процессорах одного мультипроцессора за 4 такта
- Единицей планирования в мультипроцессоре является warp

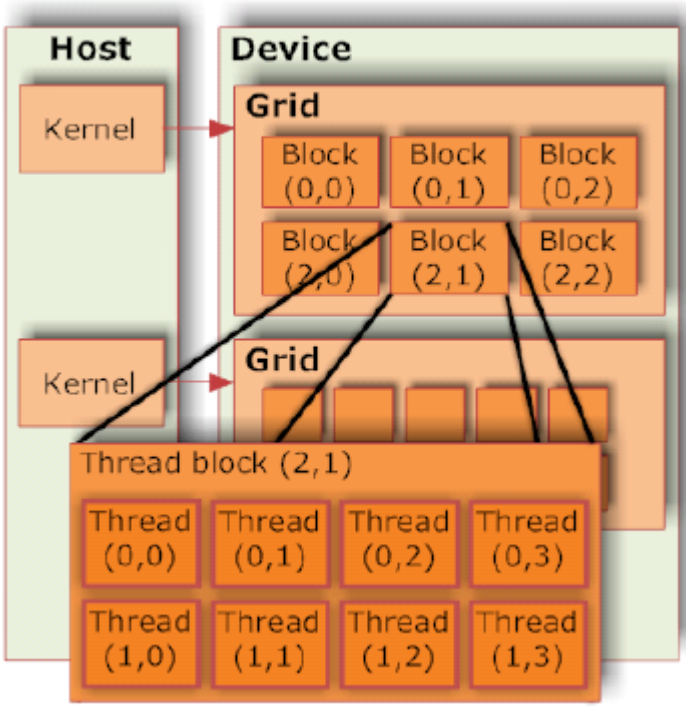
Программная модель

- **Ядро** (kernel) – функция, применяемая к сетке блоков потоков
- **Блок потоков** (threads block) – набор потоков, способный общаться между собой через
  - Разделяемую память (shared memory)
  - Точки синхронизации (барьеры)



Модель исполнения

- Блок потоков всегда исполняется только на одном мультипроцессоре
  - Разделяемые переменные хранятся в общей памяти
  - Файл локальных регистров делится между обрабатываемыми мультипроцессором потоками
- На одном мультипроцессоре может обрабатываться несколько блоков потоков
  - Разделяемая память и файл регистров делятся между обрабатываемыми блоками



Итого

- **GPU** – массив мультипроцессоров
- **Мультипроцессор** – набор потоковых процессоров и общая память
- **Ядро** – процедура для GPU
- **Сетка** – массив блоков потоков
- **Блок потоков** – массив SIMD-потоков

На следующей лекции:

Представление кафедры
Параллельных вычислений