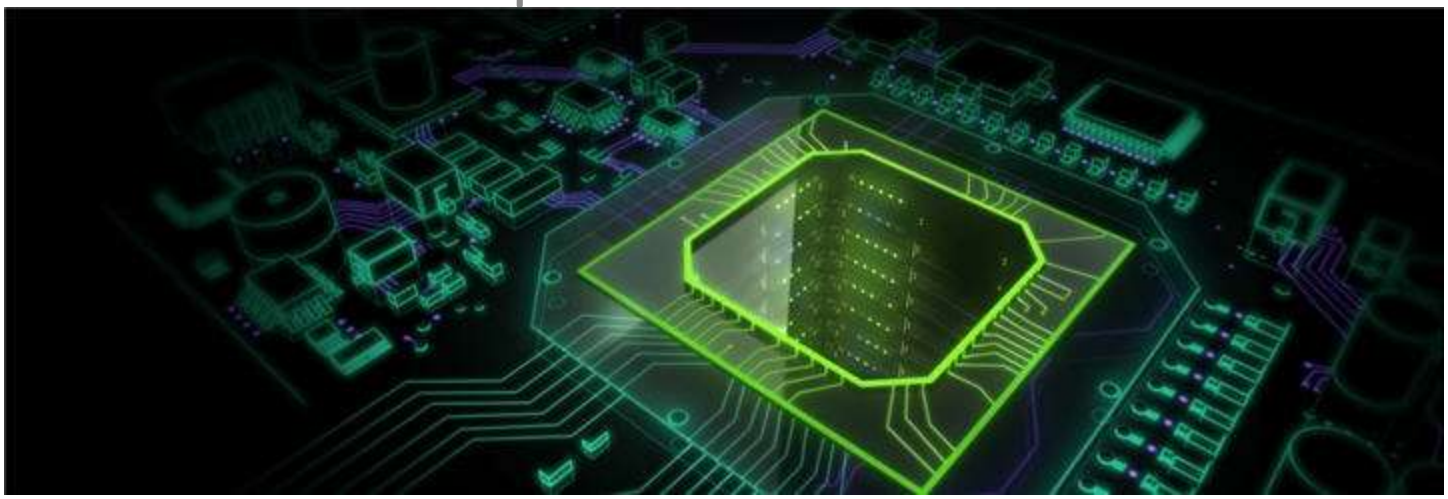
NYU
New York University

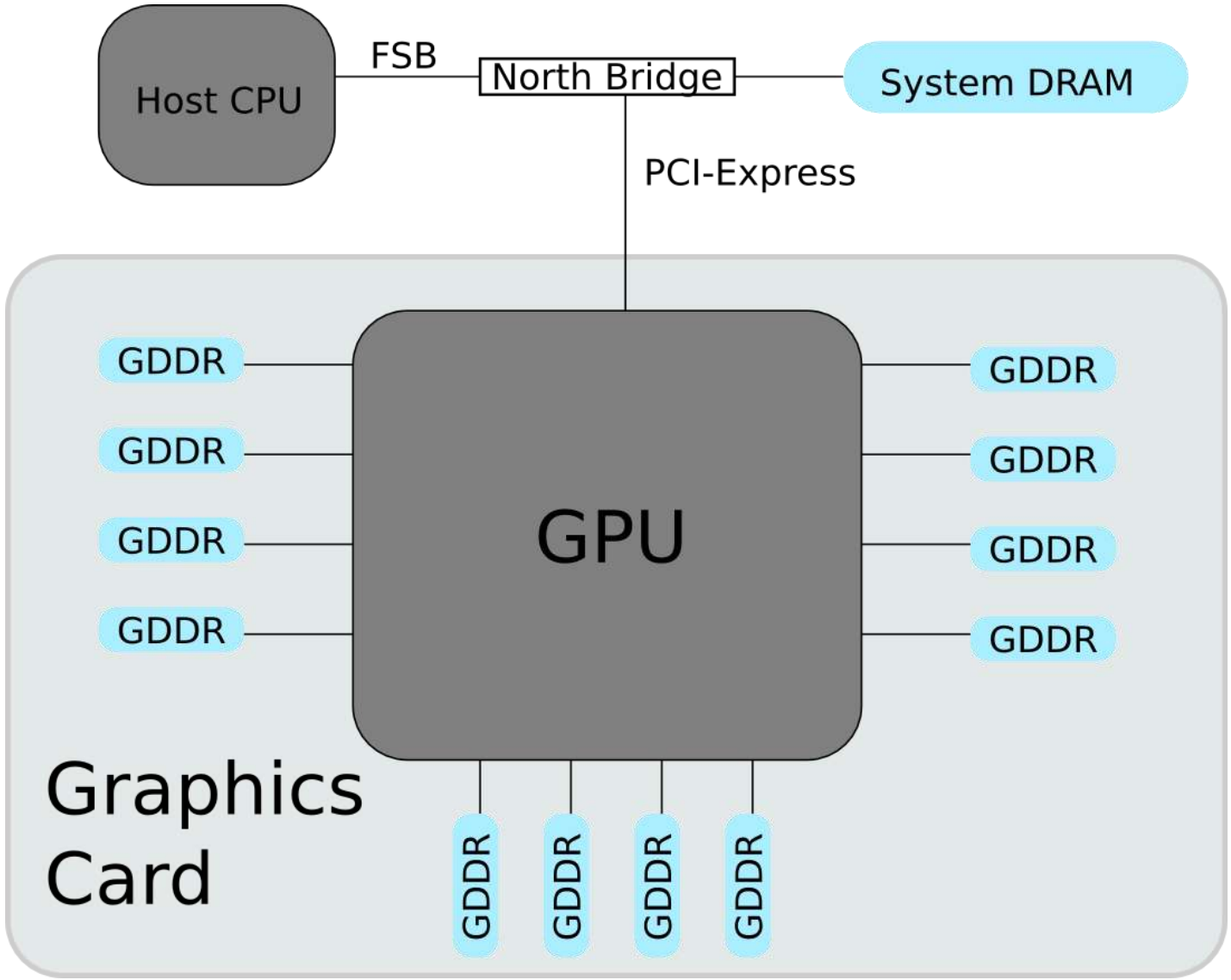
CSCI-GA.3033-012

Graphics Processing Units (GPUs): Architecture and Programming

Lecture 3: Modern GPUs A Hardware Perspective

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com

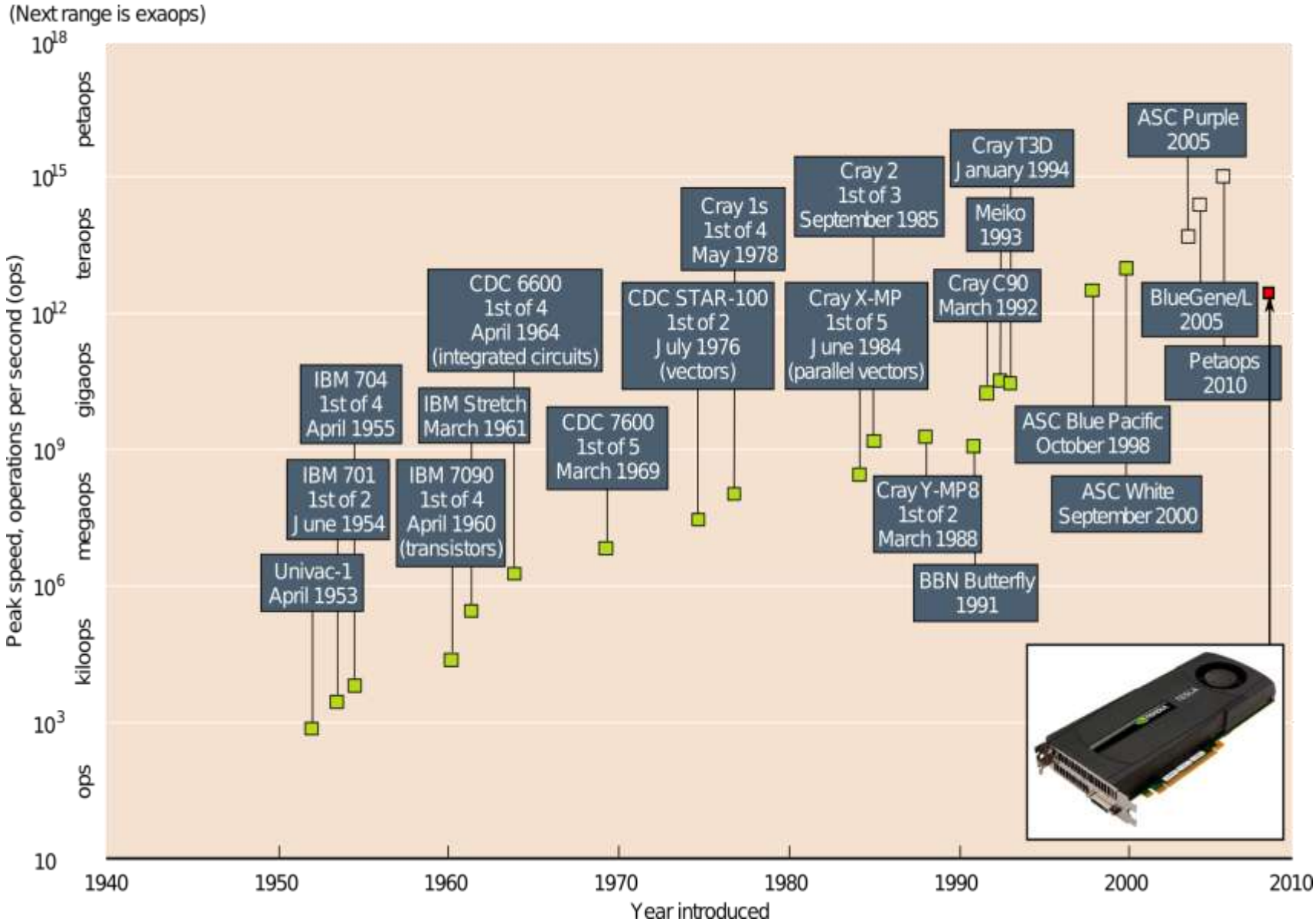



Host CPU
FSB
North Bridge
System DRAM
PCI-Express
GDDR
GDDR
GDDR
GDDR
GPU
GDDR
GDDR
GDDR
GDDR
Graphics Card
GDDR
GDDR
GDDR
GDDR

Modern GPU Hardware

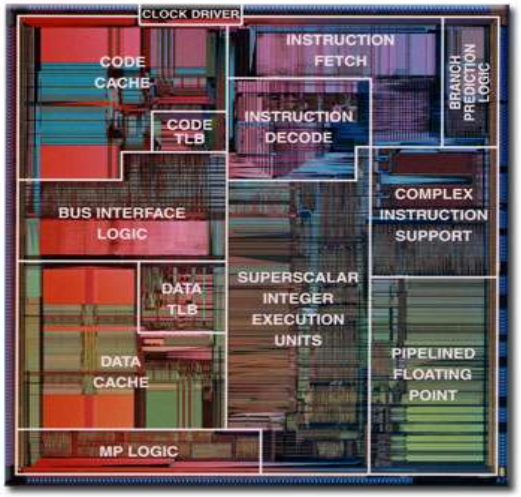
- GPUs have many parallel execution units and higher transistor counts, while CPUs have few execution units and higher clock speeds
- A GPU is for the most part deterministic in its operation (quickly changing).
- GPUs have much deeper pipelines (several thousand stages vs 10-20 for CPUs)
- GPUs have significantly faster and more advanced memory interfaces as they need to shift around a lot more data than CPUs

Single-Chip GPU vs Supercomputers

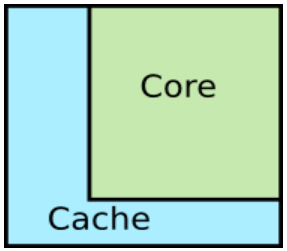


Evolution of Intel Pentium

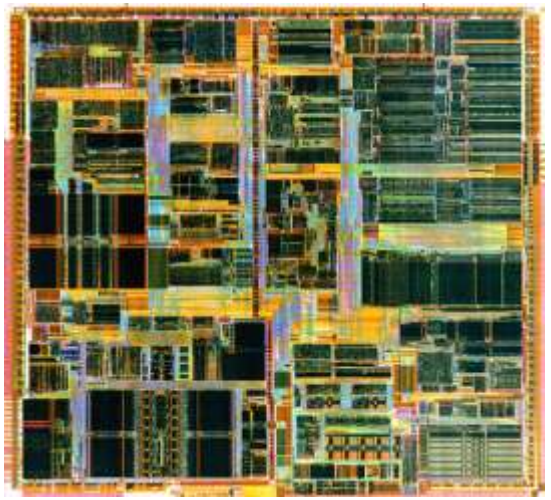
Pentium I

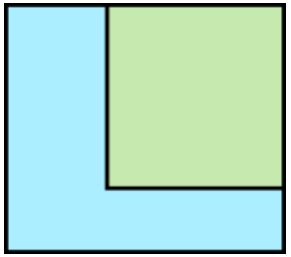


Chip area breakdown

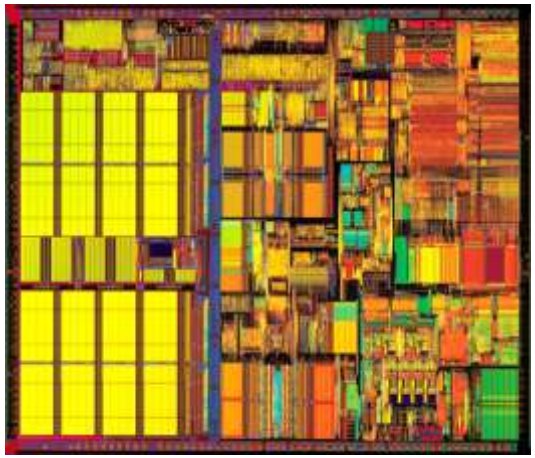


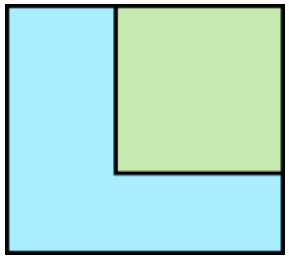
Pentium II



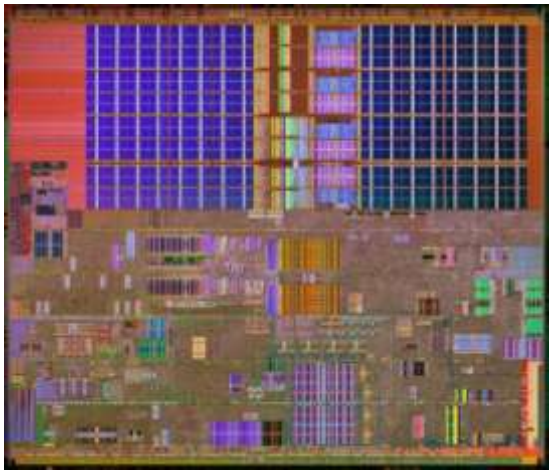


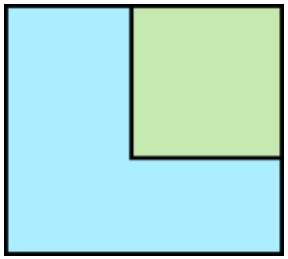
Pentium III





Pentium IV

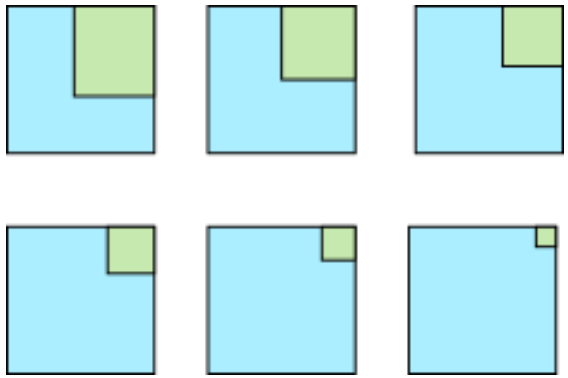




source: http://www.es.ele.tue.nl/~heco/courses/EmbeddedComputerArchitecture/GPU.ppt

Extrapolation of Single Core CPU

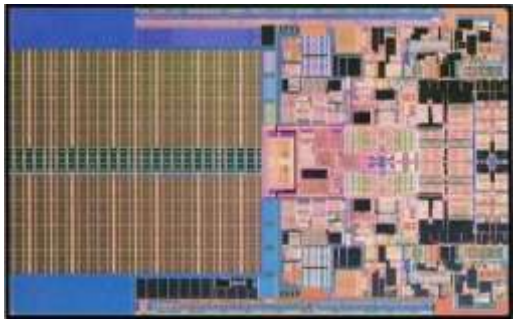
If we extrapolate the trend, in a few generations, Pentium will look like:



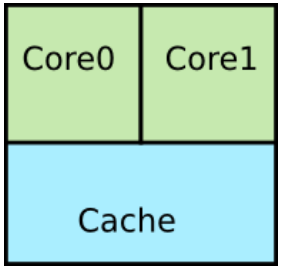
source: http://www.es.ele.tue.nl/~heco/courses/EmbeddedComputerArchitecture/GPU.ppt

Evolution of Multi-core CPUs

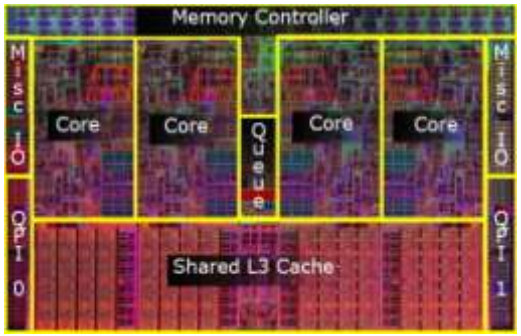
Penryn

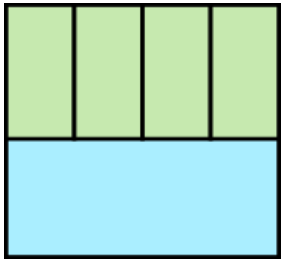


Chip area breakdown

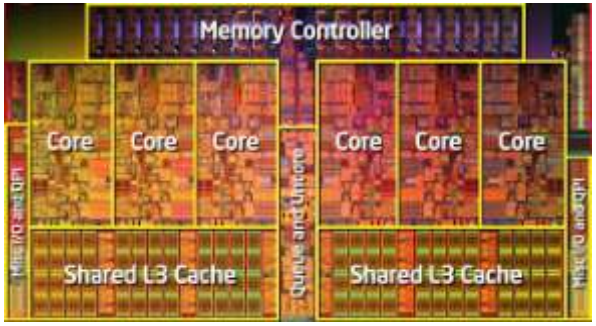


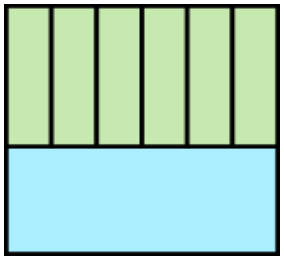
Bloomfield



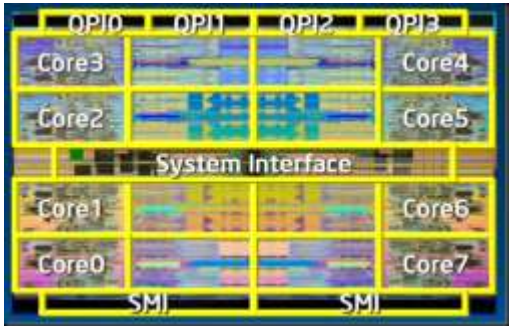


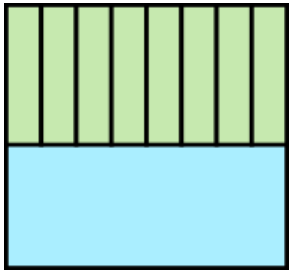
Gulftown





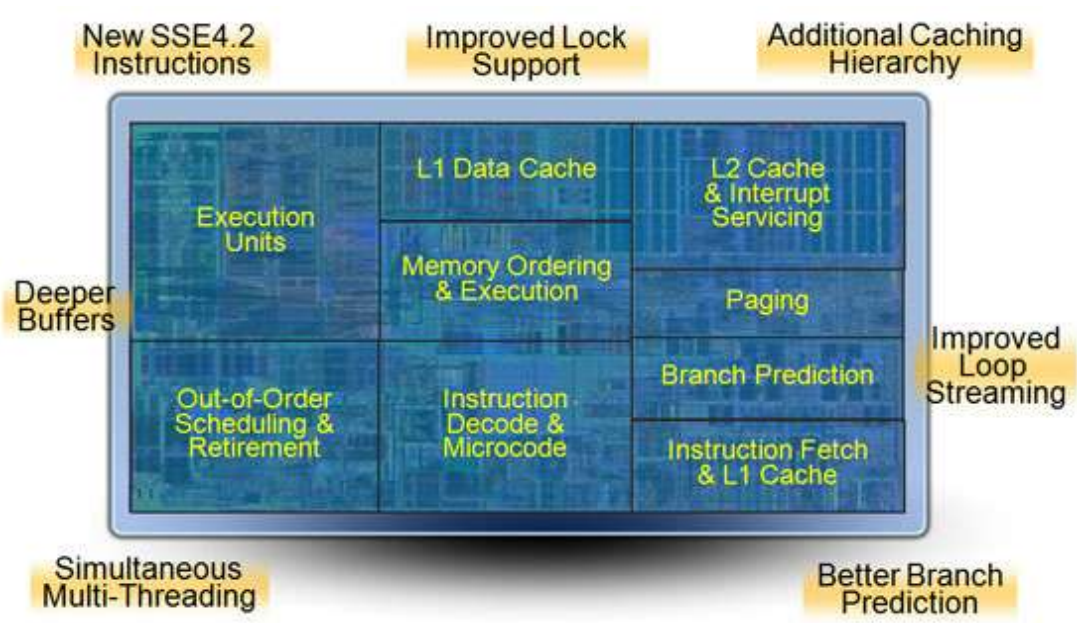
Beckton

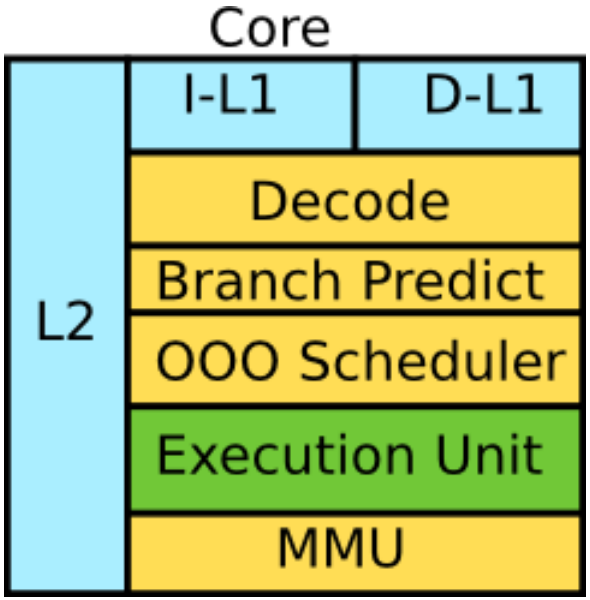




source: http://www.es.ele.tue.nl/~heco/courses/EmbeddedComputerArchitecture/GPU.ppt

Let's Take a Closer Look

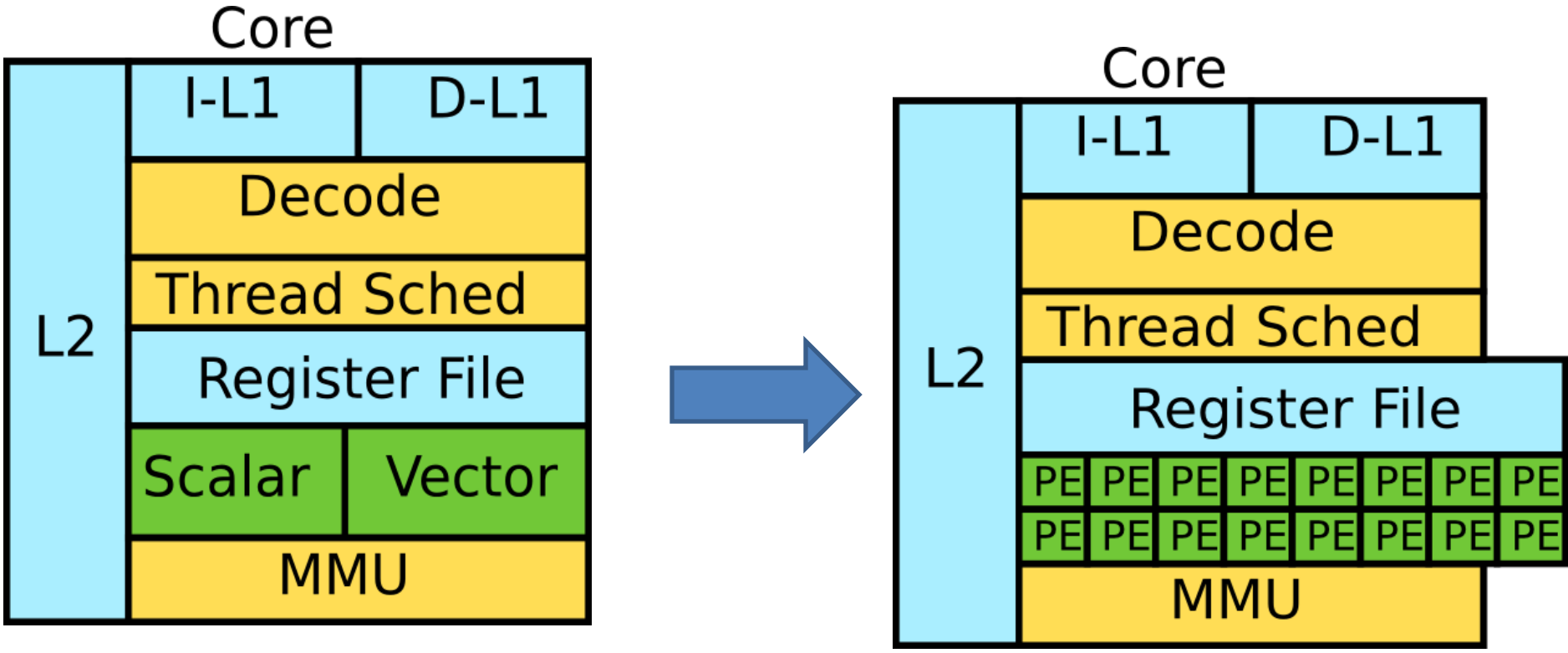


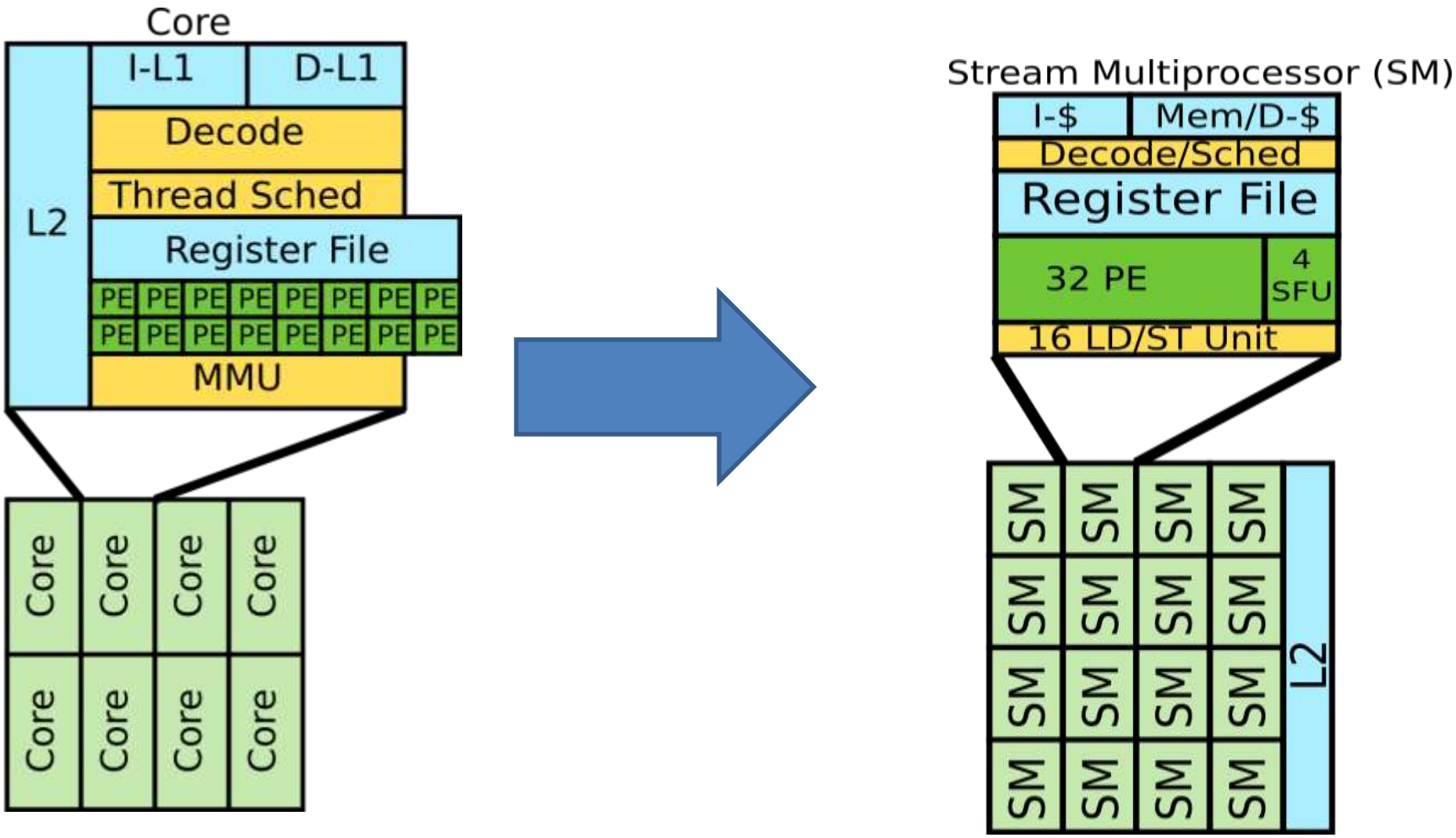


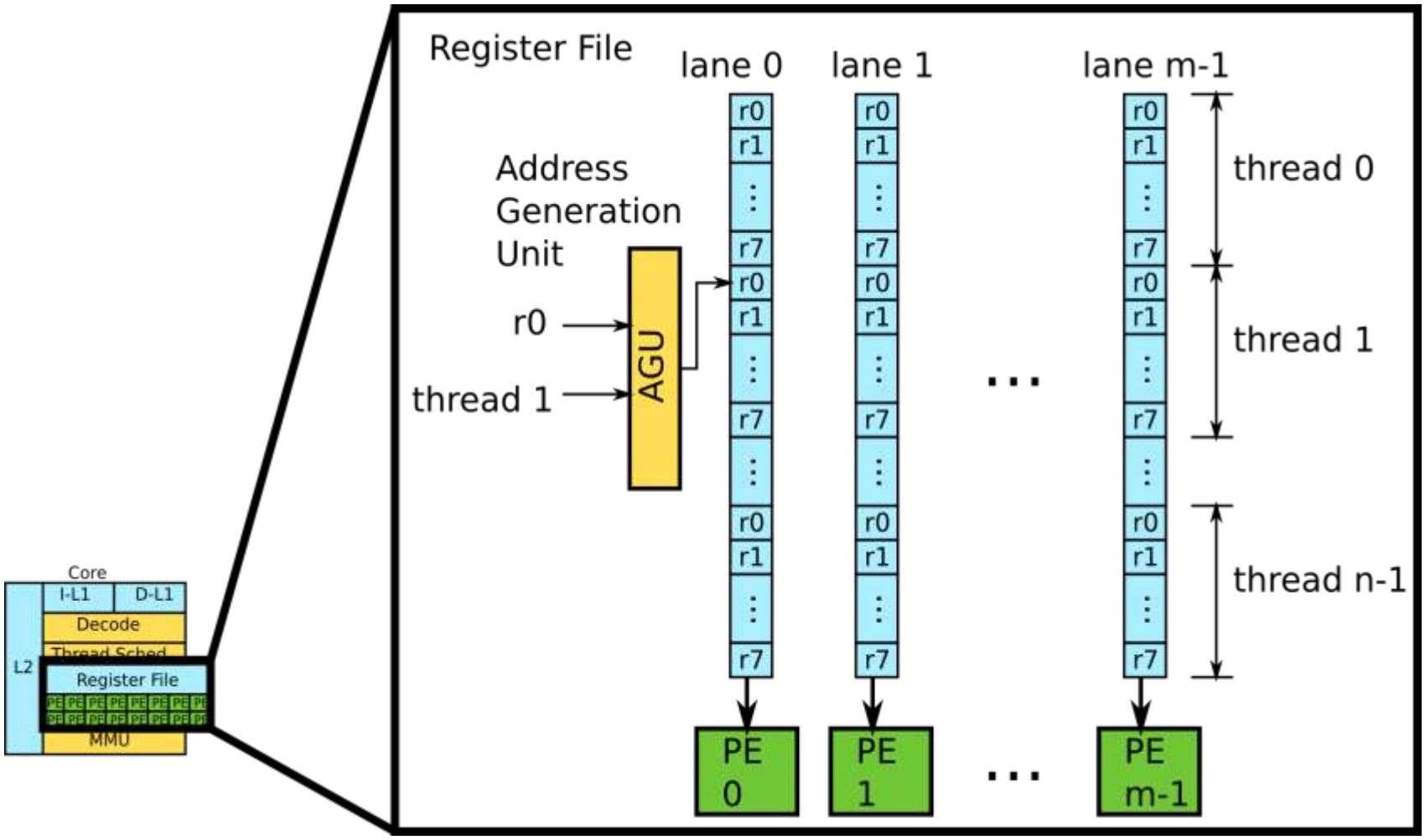
Less than 10% of total chip area is used for the real execution.

source: http://www.es.ele.tue.nl/~heco/courses/EmbeddedComputerArchitecture/GPU.ppt

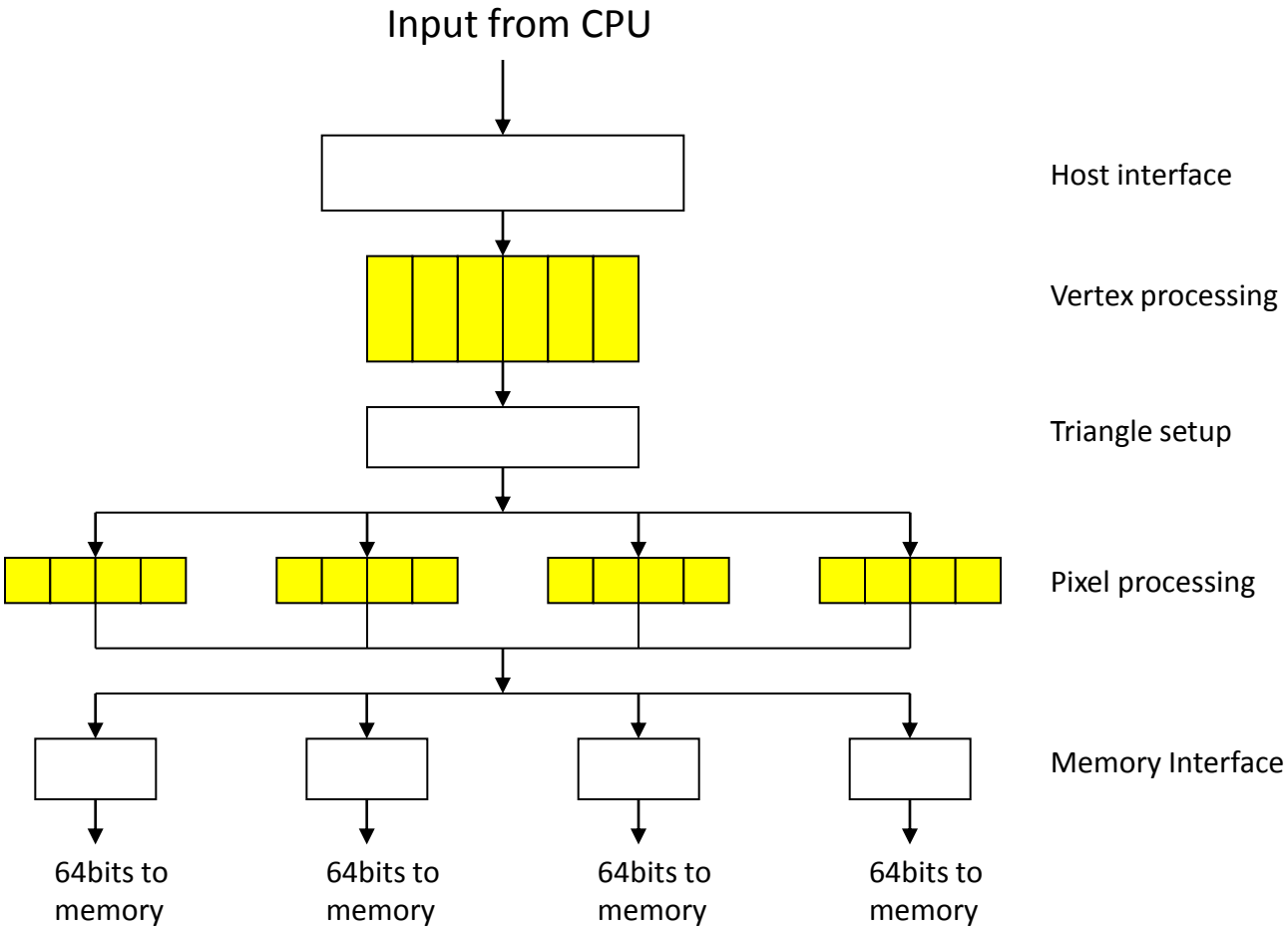
How About …


Core
L2
I-L1
D-L1
Decode
Thread Sched
Register File
Scalar
Vector
MMU
Core
L2
I-L1
D-L1
Decode
Thread Sched
Register File
PE PE PE PE PE PE PE PE
PE PE PE PE PE PE PE PE
MMU


Core
I-L1
D-L1
Decode
Thread Sched
L2
Register File
PE PE PE PE PE PE PE PE
PE PE PE PE PE PE PE PE
MMU
Core Core Core Core
Core Core Core Core
Stream Multiprocessor (SM)
I-$
Mem/D-$
Decode/Sched
Register File
32 PE
4 SFU
16 LD/ST Unit
SM SM SM SM
SM SM SM SM
SM SM SM SM
SM SM SM SM
L2

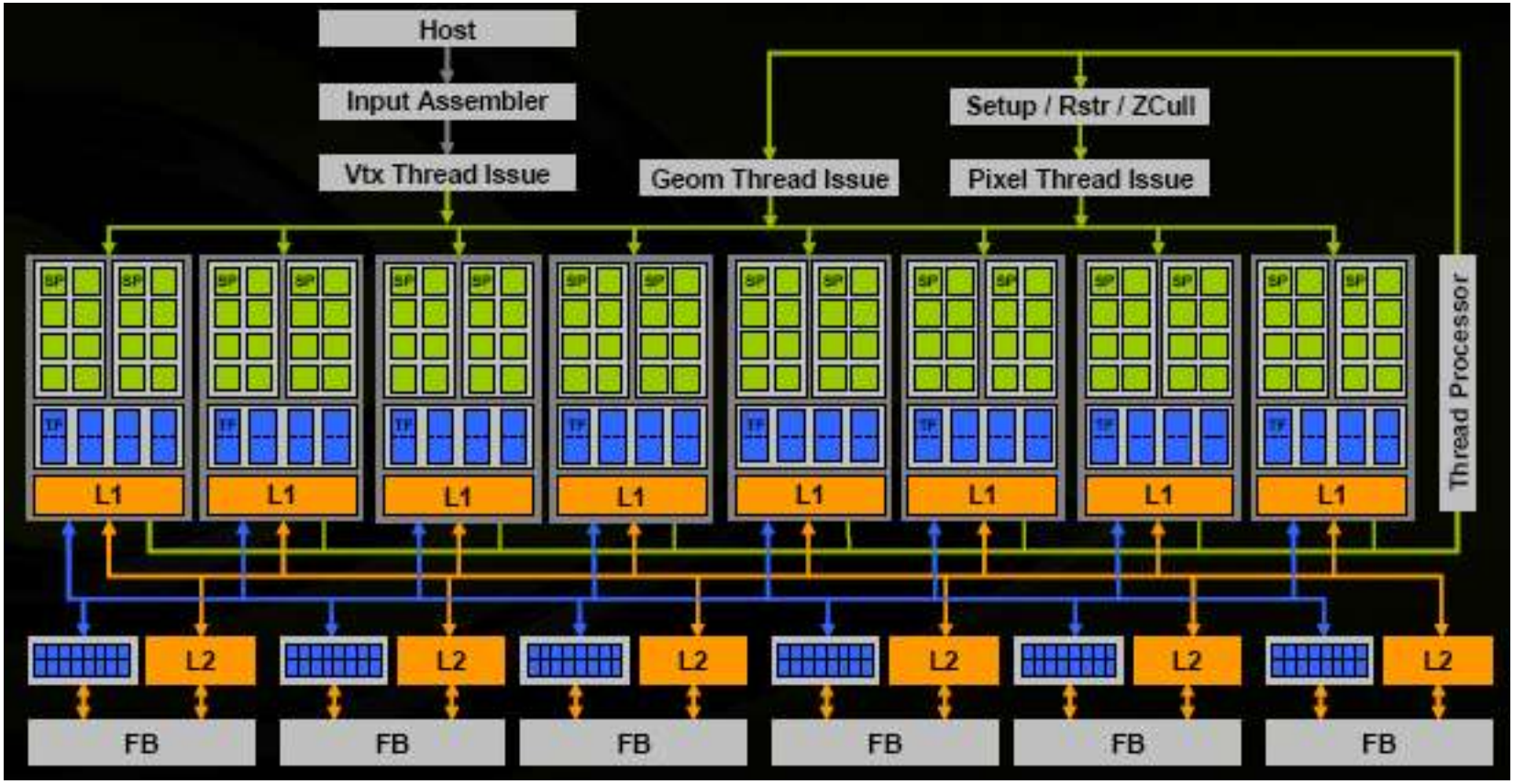

Core
I-L1
D-L1
Decode
Thread Sched
L2
Register File
MMU
Register File
lane 0
lane 1
lane m-1
Address
Generation
Unit
r0
thread 1
AGU
r0
r1
r7
thread 0
thread 1
thread n-1
PE
0
PE
1
PE
m-1

Simplified Diagram

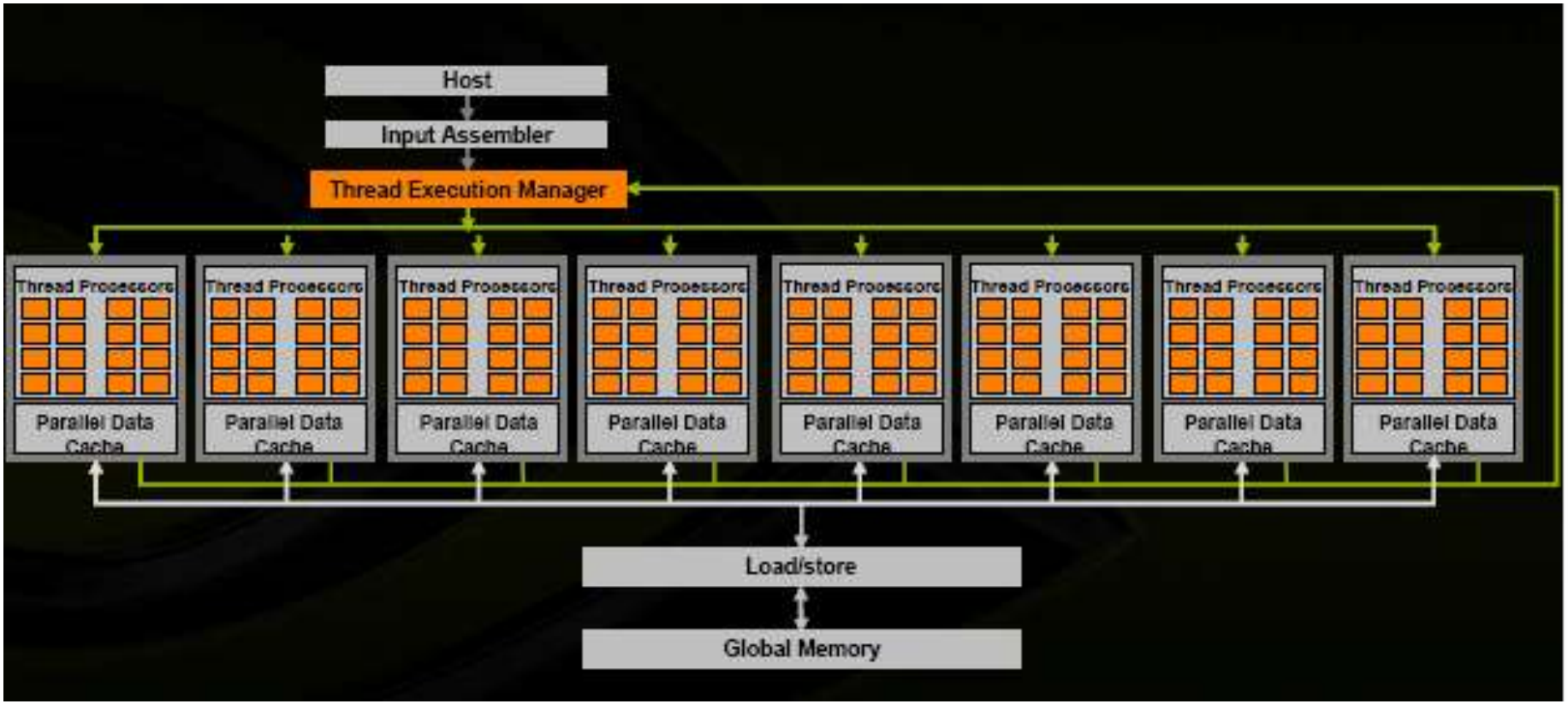


Yellow parts are programmable

This is What We Saw



This is how we expose GPU as parallel processor.



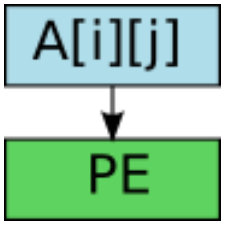
Scalar vs Vector vs Threaded

- Scalar program
-
- float A[4][8];
- do-all(i=0;i<4;i++){
- do-all(j=0;j<8;j++){
- A[i][j]++;
- }
- }

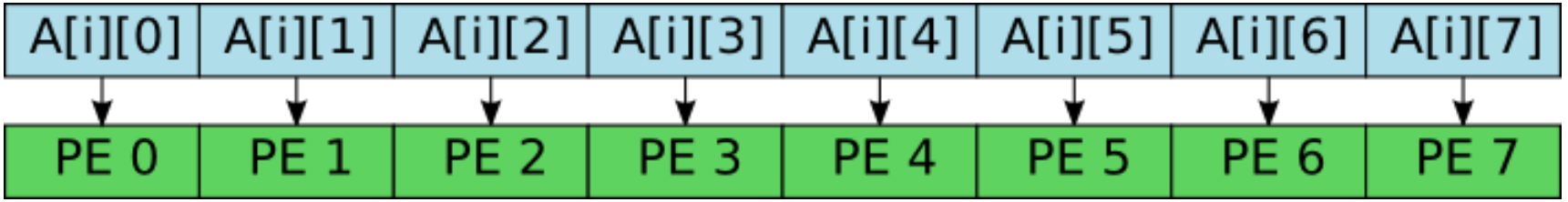
Vector Program

Vector width is exposed to programmers.

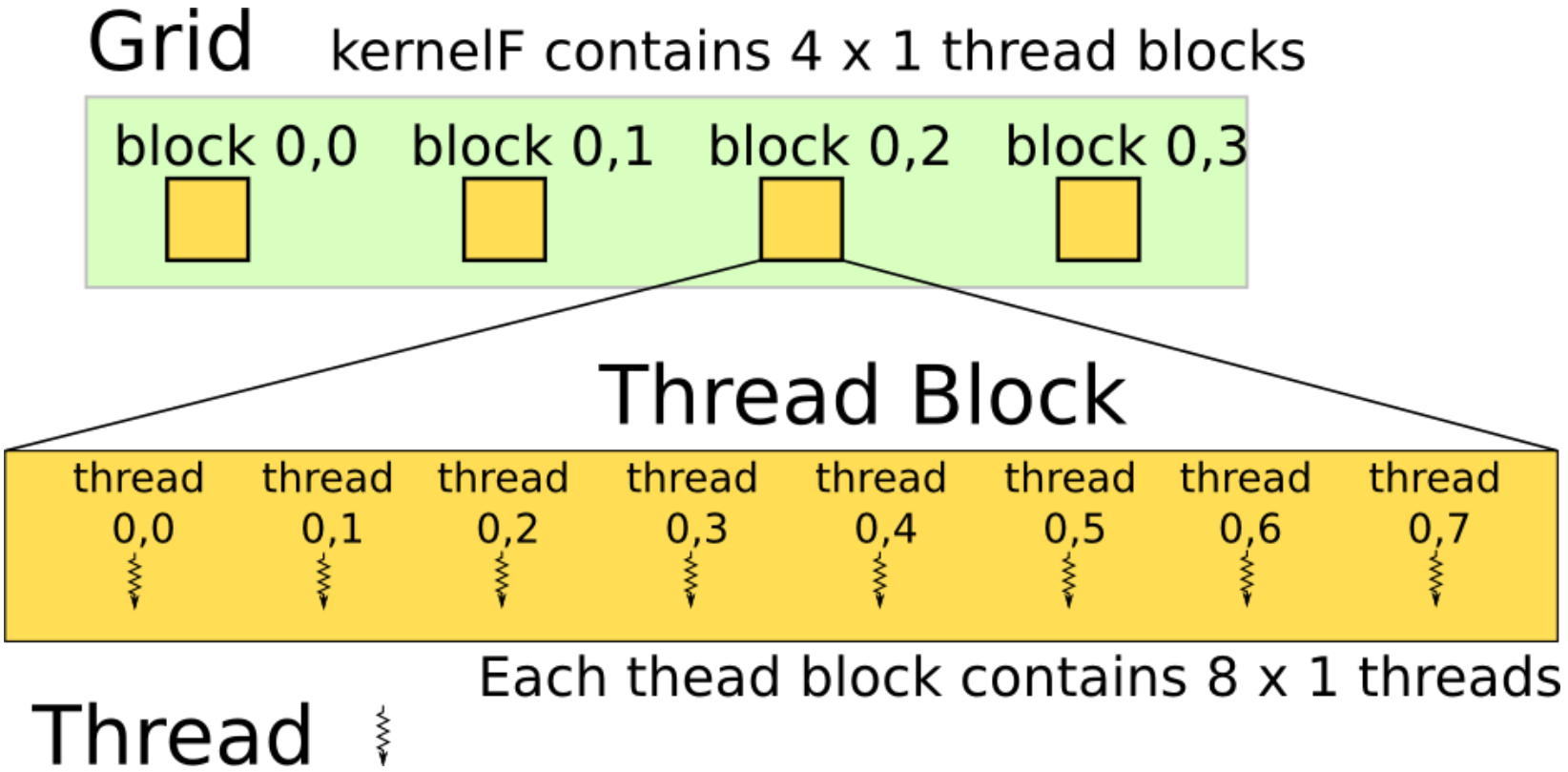
Scalar



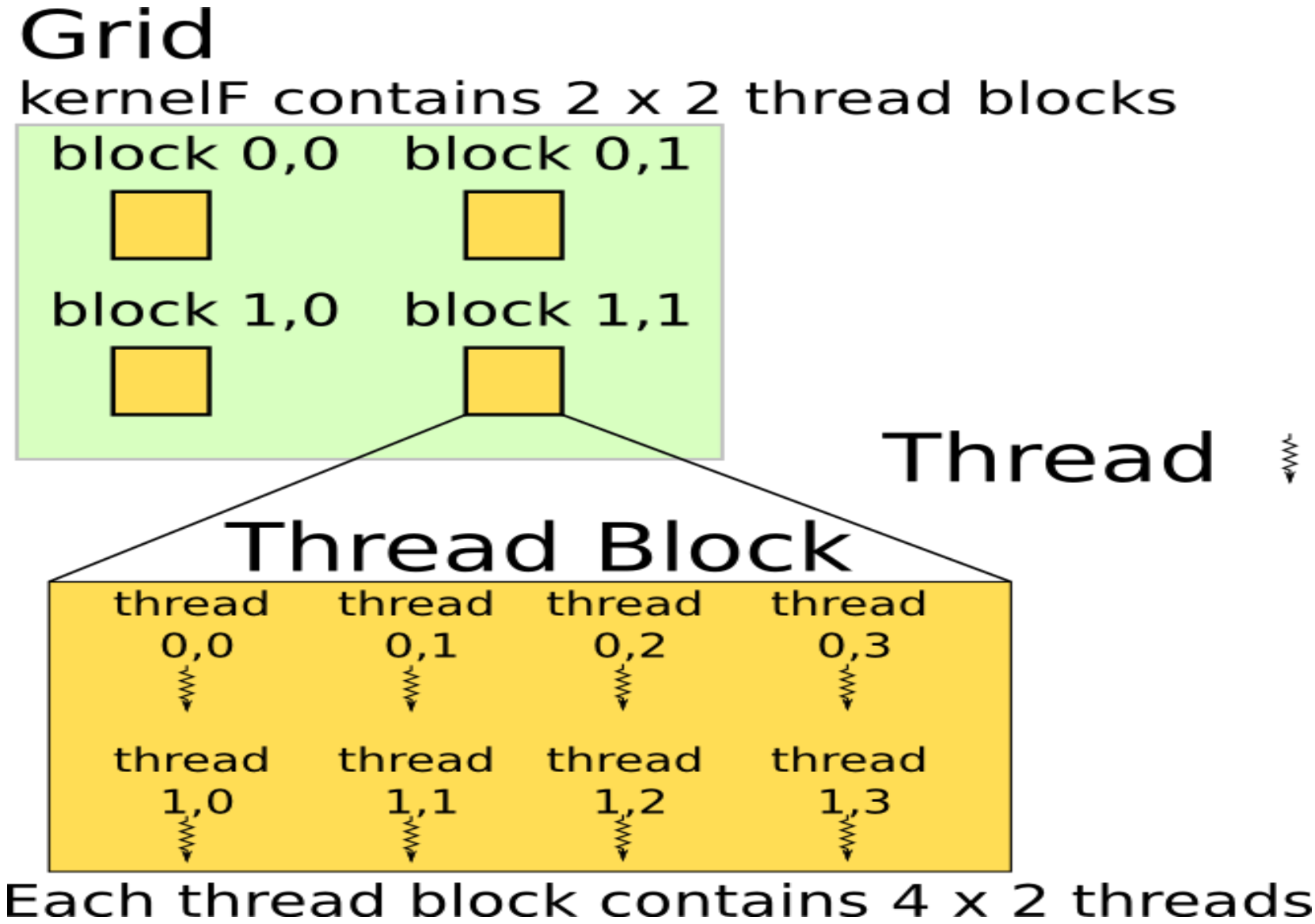
Vector if width 8



Multithreaded: (4x1)blocks – (8x1) threads



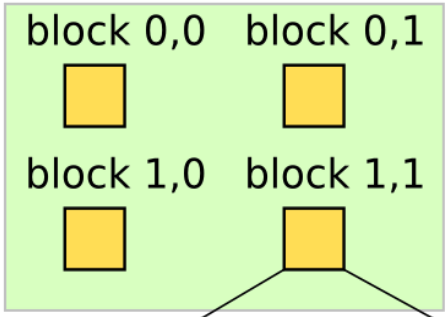
Multithreaded: (2x2)blocks – (4x2) threads

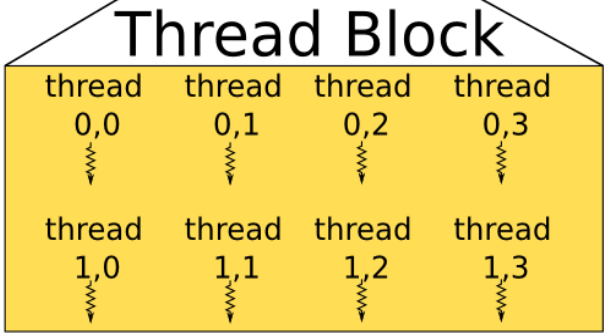


Scheduling Thread Blocks on SM

Grid

kernelF contains 2 x 2 thread blocks

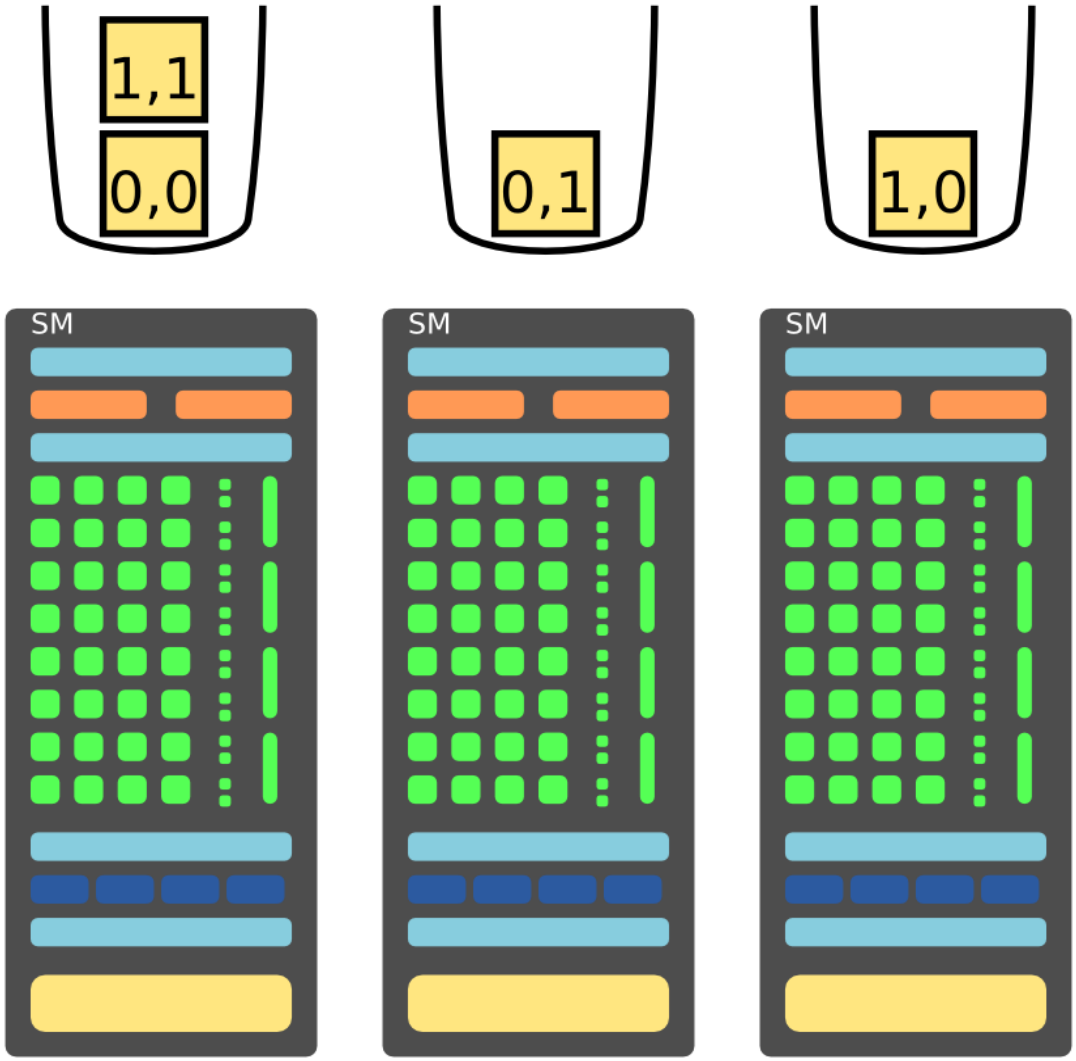




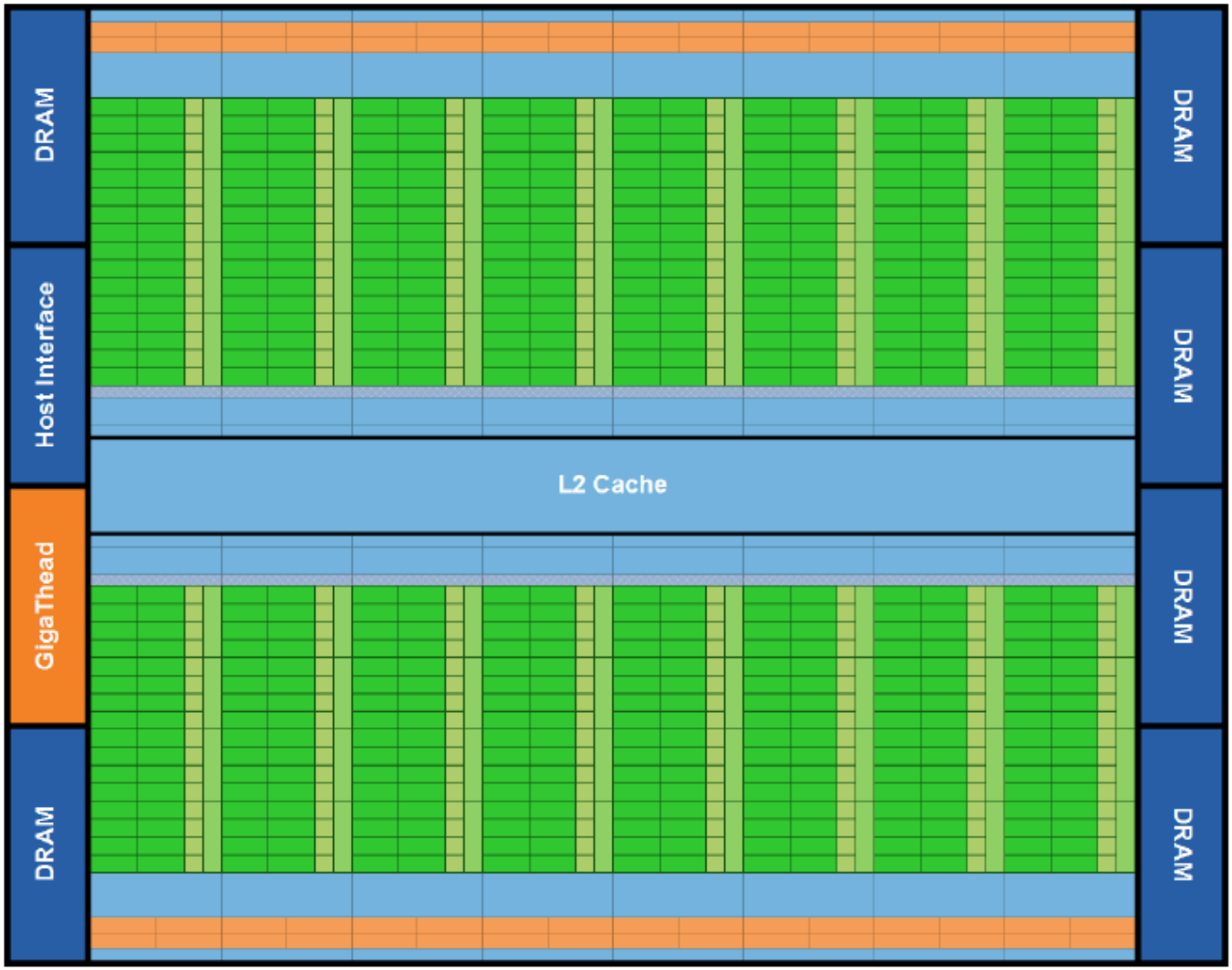
Each thread block contains 4 x 2 threads

Example:

Scheduling 4 thread blocks on 3 SMs.



State-of-the-art GPU: FERMI



32 cores/SM

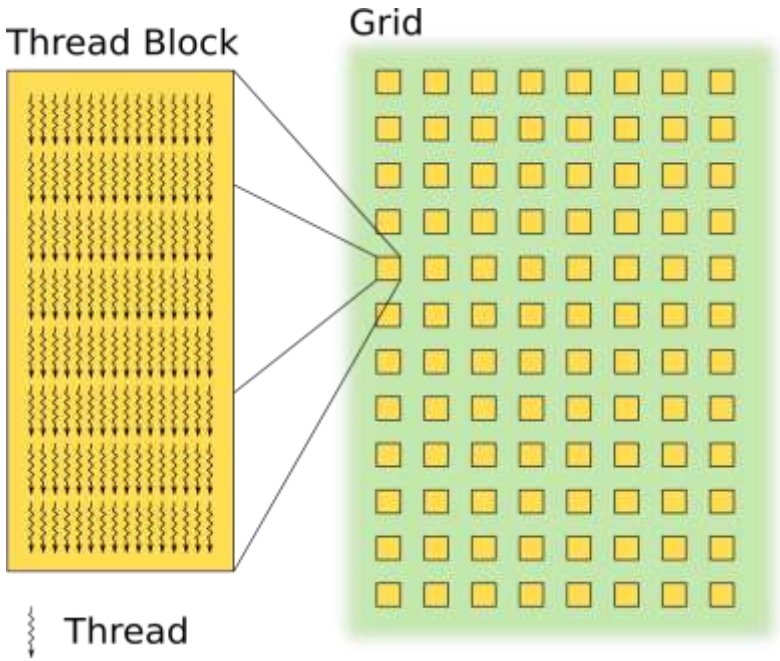
~3B Transistors

Main Goals of Fermi

- Increasing floating-point throughput
- Allowing software developers to focus on algorithm design rather than the details of how to map the algorithm to the hardware

Quick Glimpse At Programming Models

Application → **Kernels** → **Threads** → **Blocks**



↓

Grid

Quick Glimpse At Programming Models

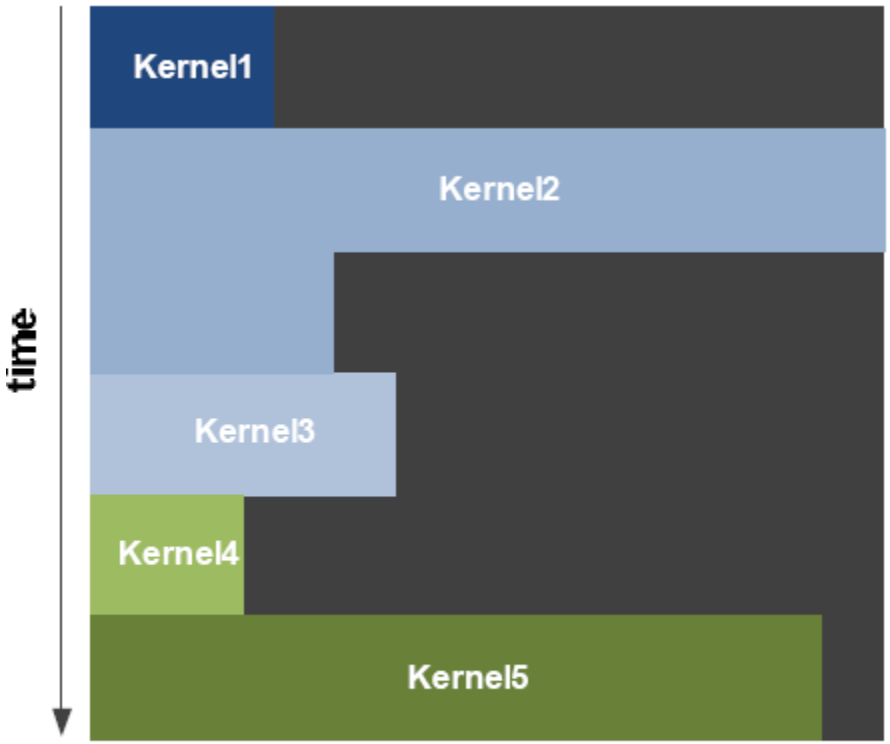
- Application can include multiple kernels
- Threads of the same block run on the same SM
  - So threads in SM can operate and share memory
  - Block in an SM is divided into warps of 32 threads each
  - A warp is the fundamental unit of dispatch in an SM
- Blocks in a grid can coordinate using global shared memory
- Each grid executes a kernel

Scheduling In Fermi

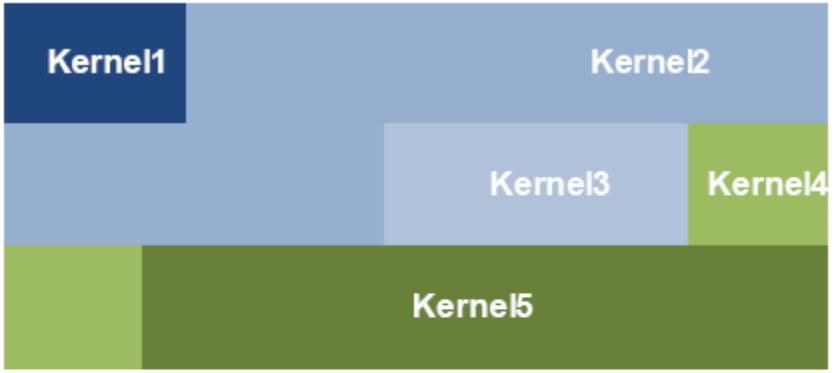
- At any point of time the entire Fermi device is dedicated to a single application
  - Switch from an application to another takes 25 microseconds
- Fermi can simultaneously execute multiple kernels of the same application
- Two warps from different blocks (or even different kernels) can be issued and executed simultaneously

Scheduling In Fermi

- two-level, distributed thread scheduler
  - At the chip level: a global work distribution engine schedules thread blocks to various SMs
  - At the SM level, each warp scheduler distributes warps of 32 threads to its execution units.



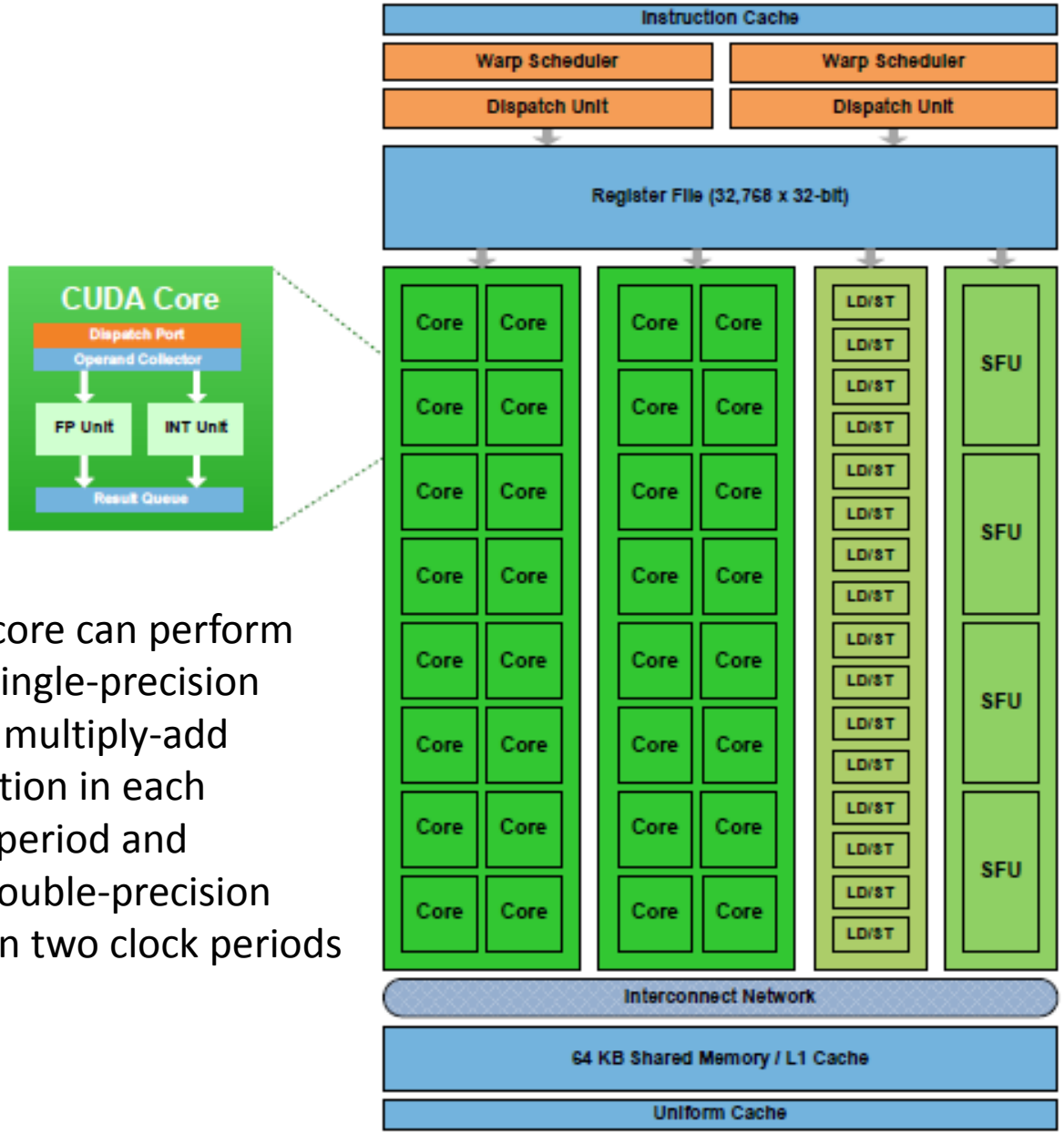
Serial Kernel Execution



Concurrent Kernel Execution

Source: Nvidia

An SM in Fermi



- 32 cores
- SFU = Special Function Unit
- 64KB of SRAM split between cache and local mem

Each core can perform one single-precision fused multiply-add operation in each clock period and one double-precision FMA in two clock periods

The Memory Hierarchy

- All addresses in the GPU are allocated from a continuous 40-bit (one terabyte) address space.
- Global, shared, and local addresses are defined as ranges within this address space and can be accessed by common load/store instructions.
- The load/store instructions support 64-bit addresses to allow for future growth.

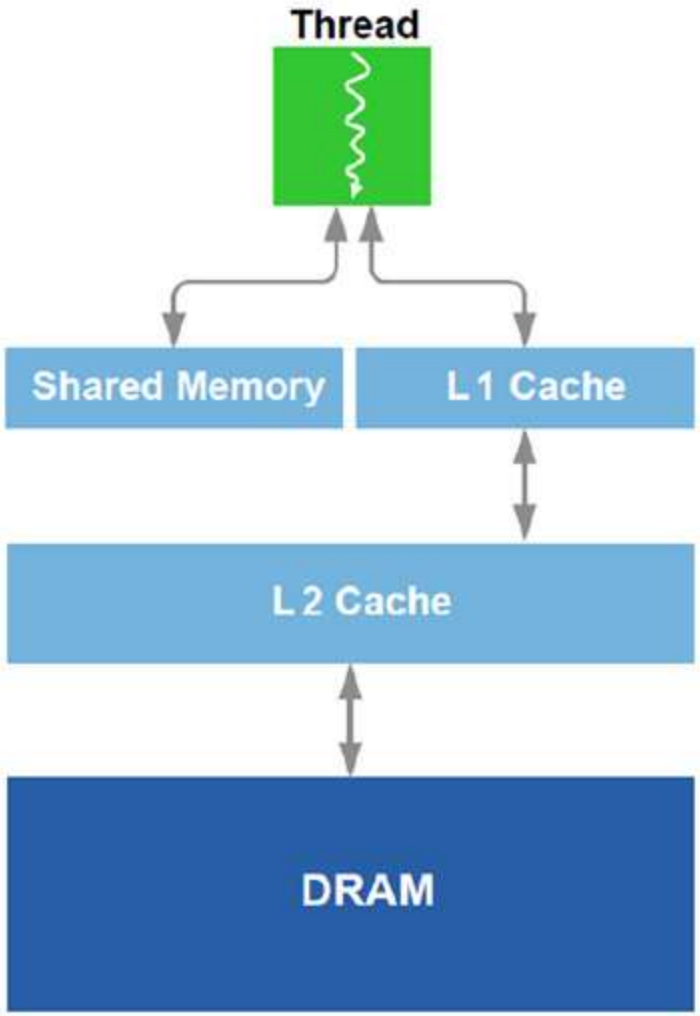
The Memory Hierarchy

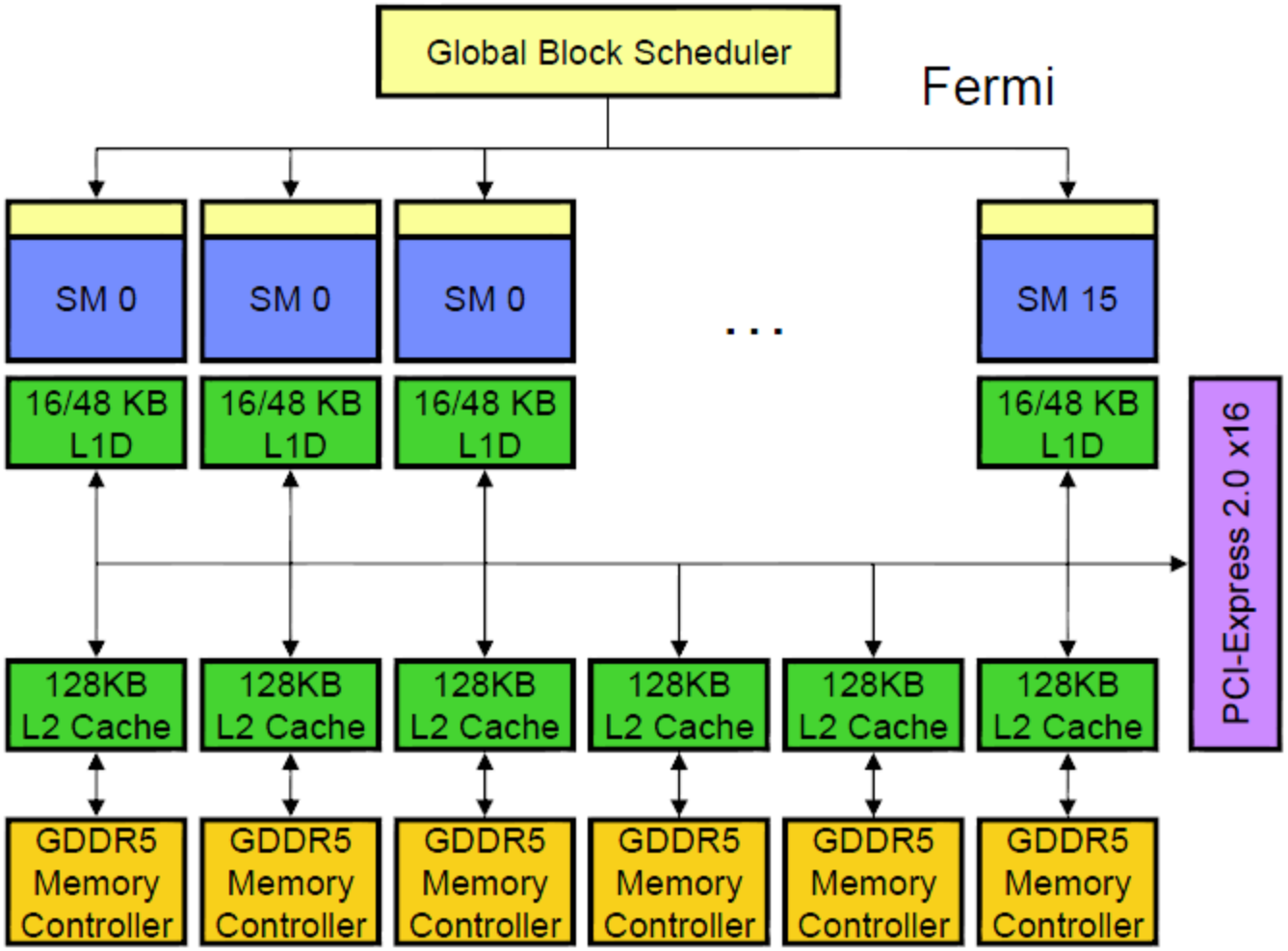
- Local memory in each SM
- The ability to use some of this local memory as a first-level (L1) cache for global memory references.
- The local memory is 64K in size, and can be split 16K/48K or 48K/16K between L1 cache and shared memory.
- Because the access latency to this memory is also completely predictable, algorithms can be written to interleave loads, calculations, and stores with maximum efficiency.

The Memory Hierarchy

- Fermi GPU is also equipped with an L2.
- The L2 cache covers GPU local DRAM as well as system memory.
- The L2 cache subsystem also implements a set of memory read-modify-write operations that are atomic

Fermi Memory Hierarchy




Global Block Scheduler
Fermi
SM 0
SM 0
SM 0
SM 15
16/48 KB L1D
16/48 KB L1D
16/48 KB L1D
16/48 KB L1D
PCI-Express 2.0 x16
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
128KB L2 Cache
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller
GDDR5 Memory Controller

Conclusions

- By looking at the hardware features, can you see how you can write more efficient programs for GPUs?
- Start forming groups of up to 5 students for the project
- Two papers to read (links on the web page)