Lecture 09
캐시 코히어런스, GPU 아키텍처

캐시 코히어런스

Cache Coherence Problem

- Caching is vital to reducing memory latency in multiprocessor systems
- Private caches in a multiprocessor system may create a coherence problem
  - Copies of a variable can be present in multiple caches
- We expect that a read by any processor to return the most up-to-date value
- A write by one processor may not become visible to others
  - The result of the write are not observed by others
  - Stale values

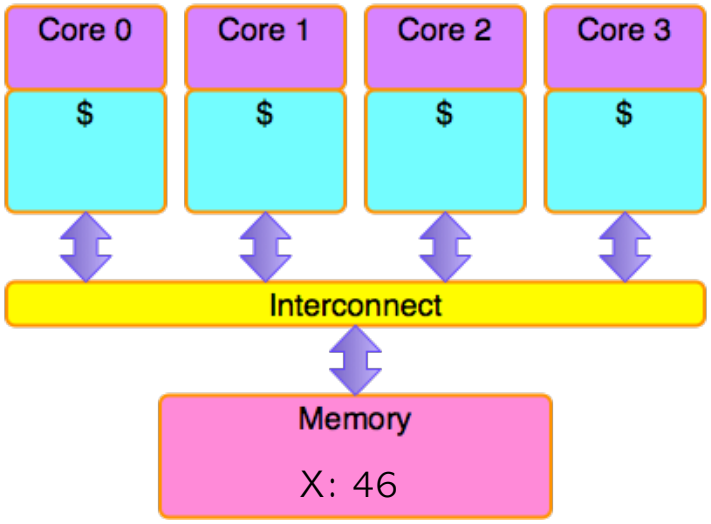
Cache Coherence Problem (contd.)

- Easy in uniprocessors except I/O

- Coherence problems between I/O devices and the processor caches
  - Uncacheable memory region, uncacheable operations, flushing pages, passing I/O data through caches, etc.

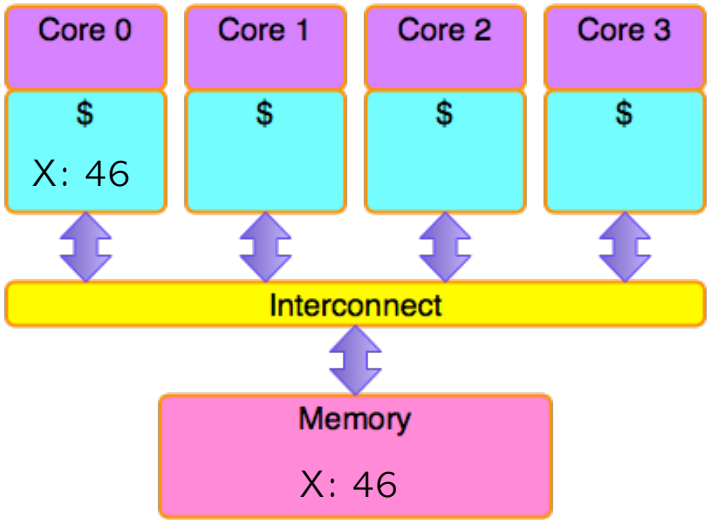
Cache Coherence Problem

- Assume write back, write allocate caches



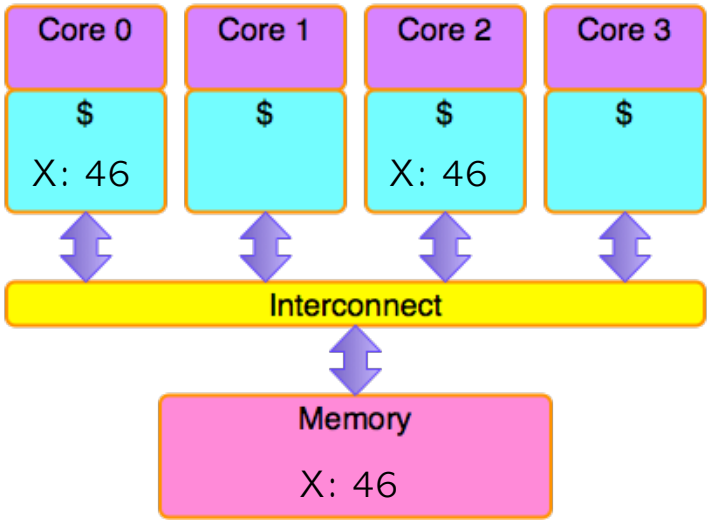
Cache Coherence Problem

- Assume write back, write allocate caches
- Core 0 reads location X



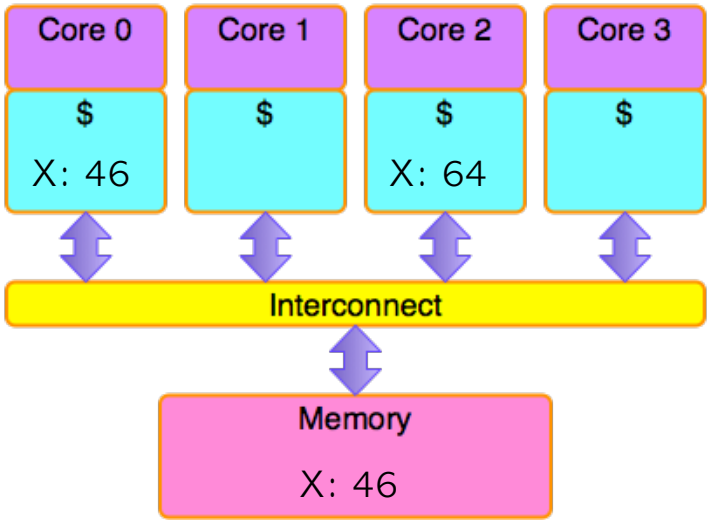
Cache Coherence Problem

- Assume write back, write allocate caches
- Core 2 reads location X



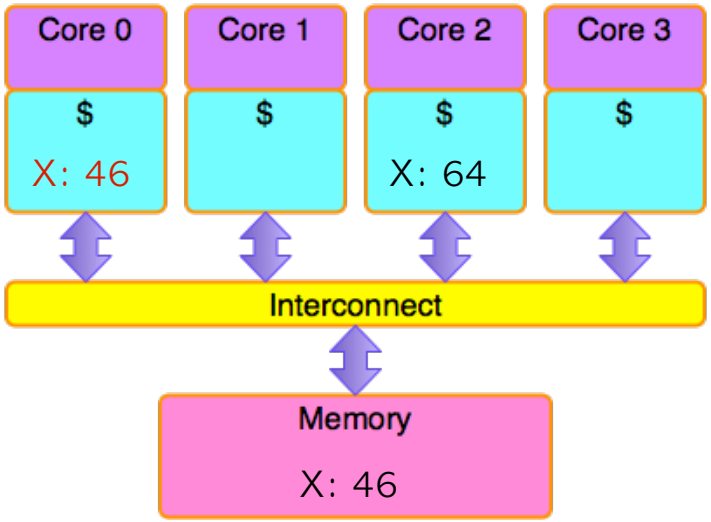
Cache Coherence Problem (contd.)

- Core 2 writes location X



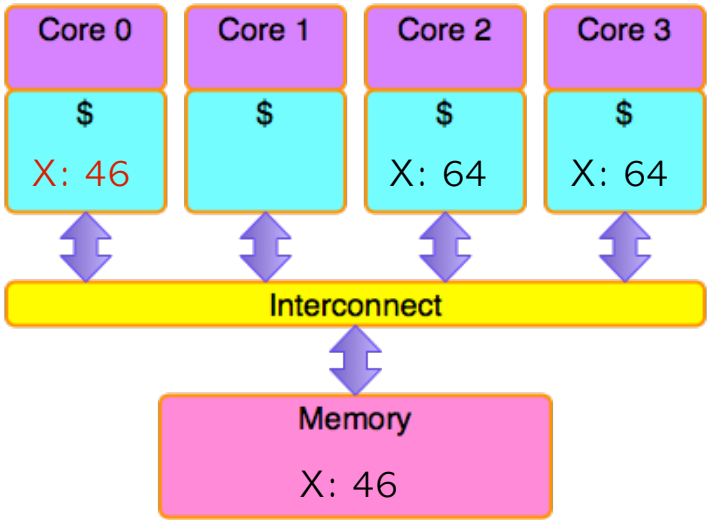
Cache Coherence Problem (contd.)

- Core 0 has a stale copy of X
- Some action must be taken
  - Update or invalidate (more common)



Cache Coherence Problem (contd.)

- Core 3 reads location X
- Core 2 supplies the updated copy of X to Core 3



Solutions to the Cache Coherence Problem

- Software based vs. hardware based
- Software-based:
  - Compiler or/and runtime
  - With or without hardware support
  - Perfect memory access information is needed
    - Aliasing and parallelism
- Hardware-based solutions are more common

Solutions to the Cache Coherence Problem (contd.)

- Hardware cache coherence protocol ensures that requests for a certain data item always return the most recent value
  - Snoopy and directory-based protocols
  - MSI, MESI, MOESI, etc.
- Coherence misses
  - Misses due to invalidations when using an invalidation-based cache coherence protocol
- Shared caches do not have a coherence problem

Snooping

- Each processor snoops every address placed on the bus when a processor requests a read from memory

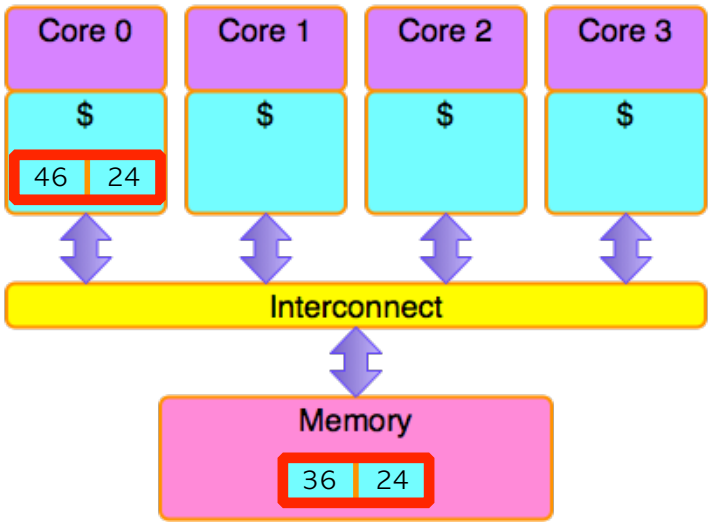
- If a processor has a dirty copy of the requested cache block, provide that cache block to the requestor and abort the memory access

Coherence and Write-through Caches

- All processor writes result in update of the local cache and a global bus write
  - Updates main memory
  - Needs to invalidate/update all other caches
- Simple
- Memory contains the most up-to-date value

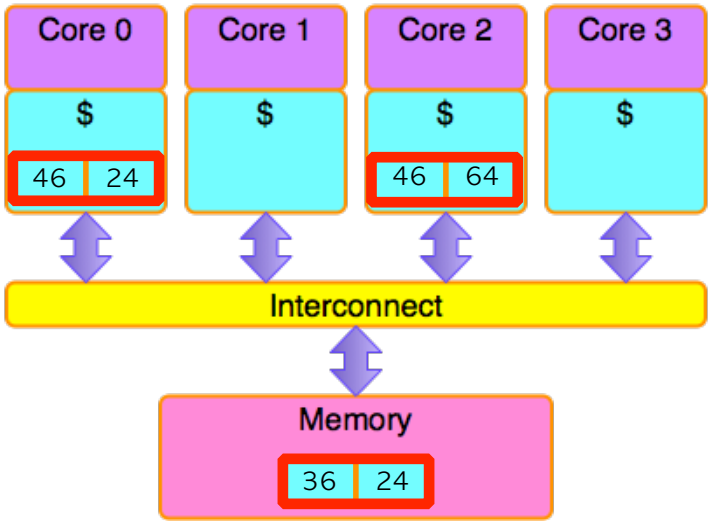
False Sharing

- Core 0 writes to the left half of a cache block



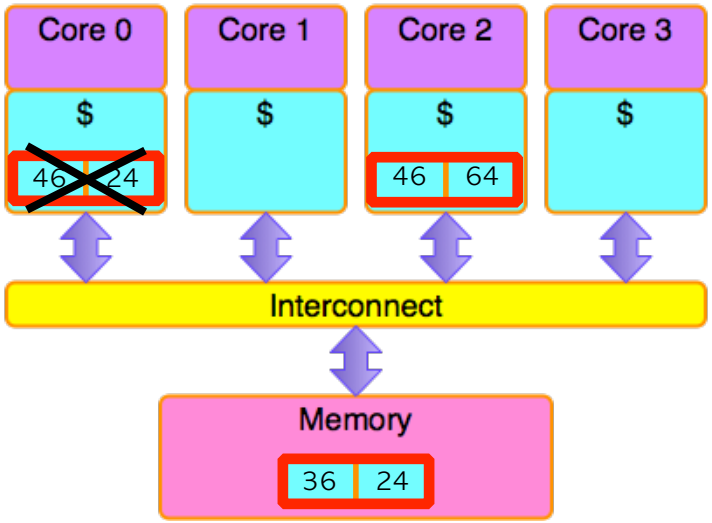
False Sharing (contd.)

- Core 2 writes to the right half of the cache block



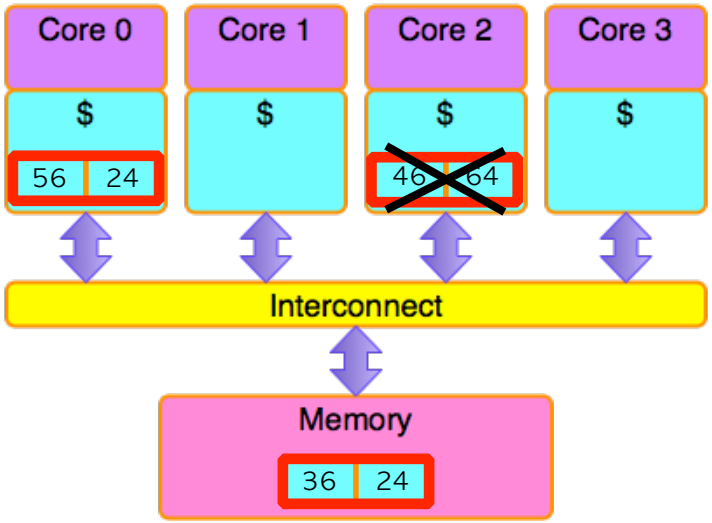
False Sharing (contd.)

- The cache block of Core 0 is invalidated



False Sharing (contd.)

- Core 0 writes to the left half of the cache block again



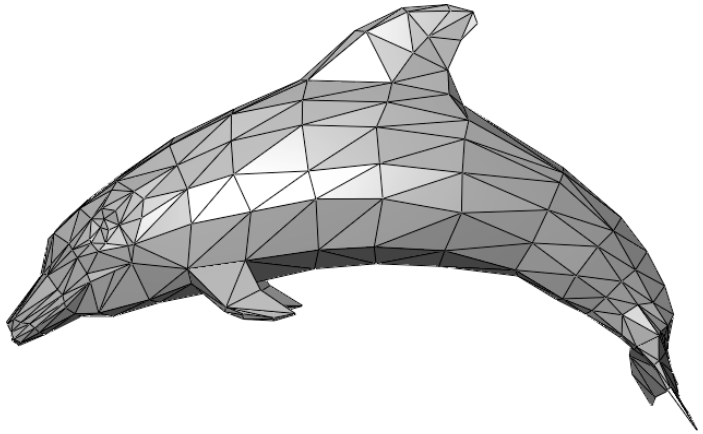
False Sharing (contd.)

- Invalidations can lead to problems with false sharing
  - Different cores write to different locations in the same cache block repeatedly
  - Force the cache line to ping-pong back and forth between the two cores
- Not occur in update-based protocols

GPU 아키텍처

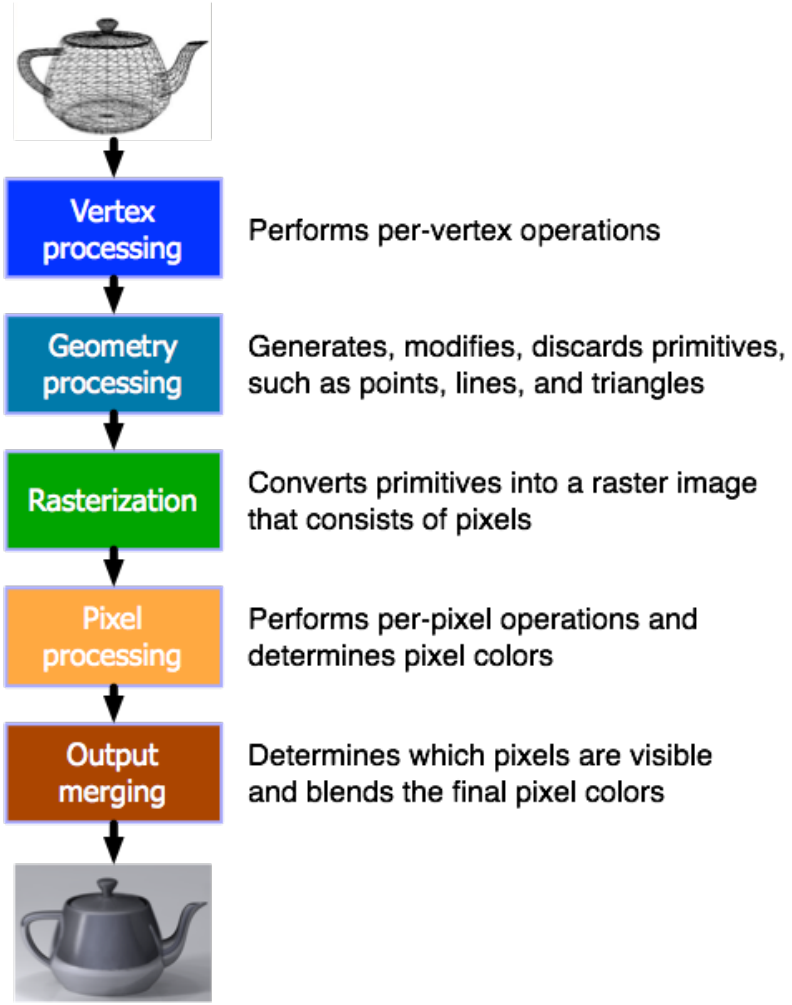
Rendering

- The process of generating an image from a 3D model
- 3D modeling
  - The process of developing a mathematical representation (i.e., model) of any three-dimensional surface of an object
  - Polygon or triangle meshes



Rendering (Graphics) Pipeline

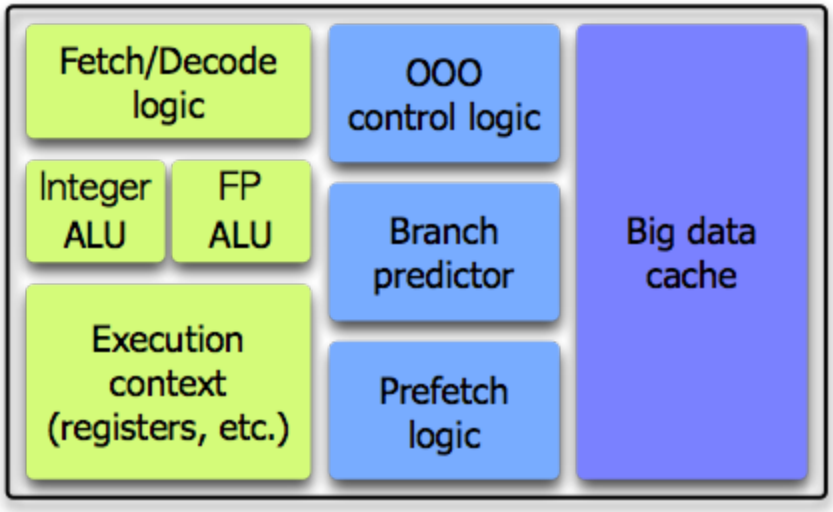
- Generates 2D images from 3D models
- Supported by commodity hardware
- Some stages are programmable
  - Vertex
  - Geometry
  - Pixel
  - ...



Shaders

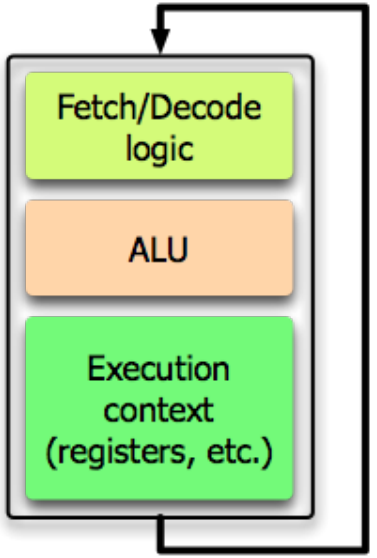
- Computer program that runs on the GPU and its purpose is to execute one of the programmable stages of the rendering pipeline

General-purpose CPU



Shader Cores

- Very simple, programmable
- No architecture components that make a single instruction stream run fast
- Logical graphics pipeline
  - Vertex shader, Geometry shader, Pixel shader, etc.
  - Graphics pipeline stages are implemented by visiting the programmable shader core several times

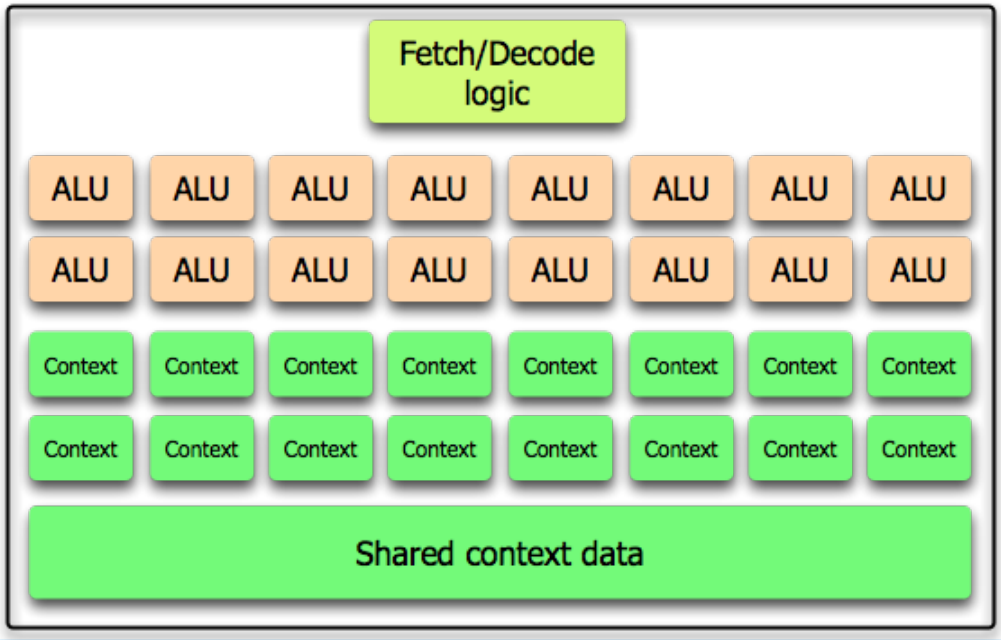


Characteristics of GPU Applications

- Data independence between triangles and pixels
- Millions of triangles and pixels
- Can perform massively parallel processing

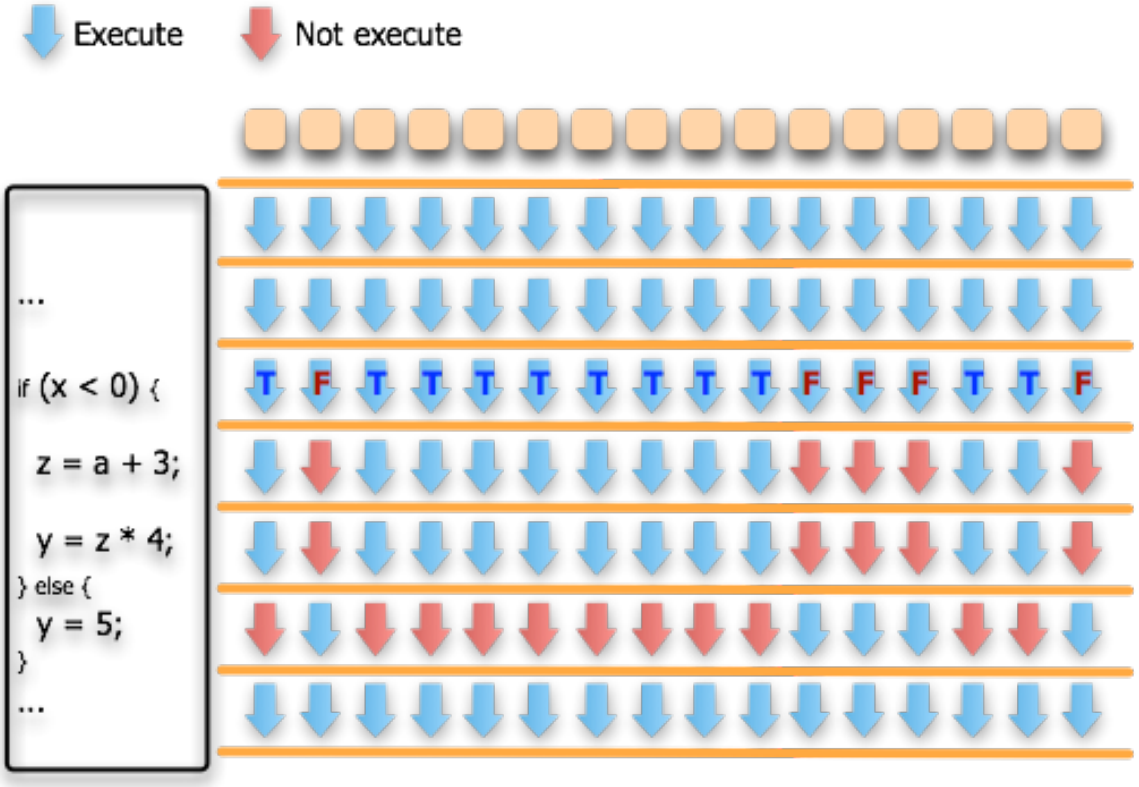
Exploiting Massive Parallelism

- Streaming multiprocessors
  - SIMD processing across many ALUs
  - A single program counter across multiple ALUs
- SIMT
  - Single Instruction, Multiple Thread



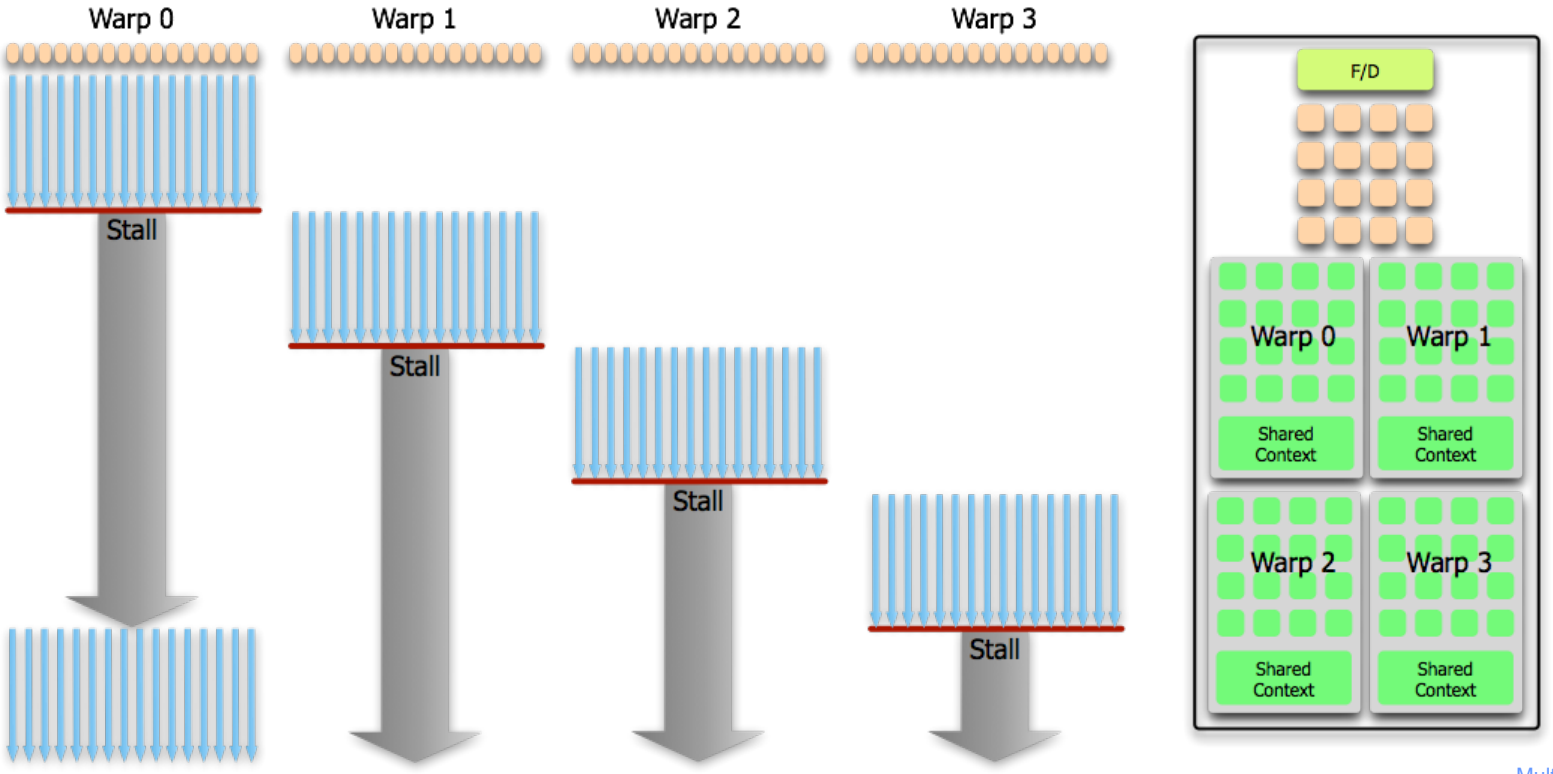
Executing Branches

- Conditional execution in a warp
- Hardware automatically handles divergence



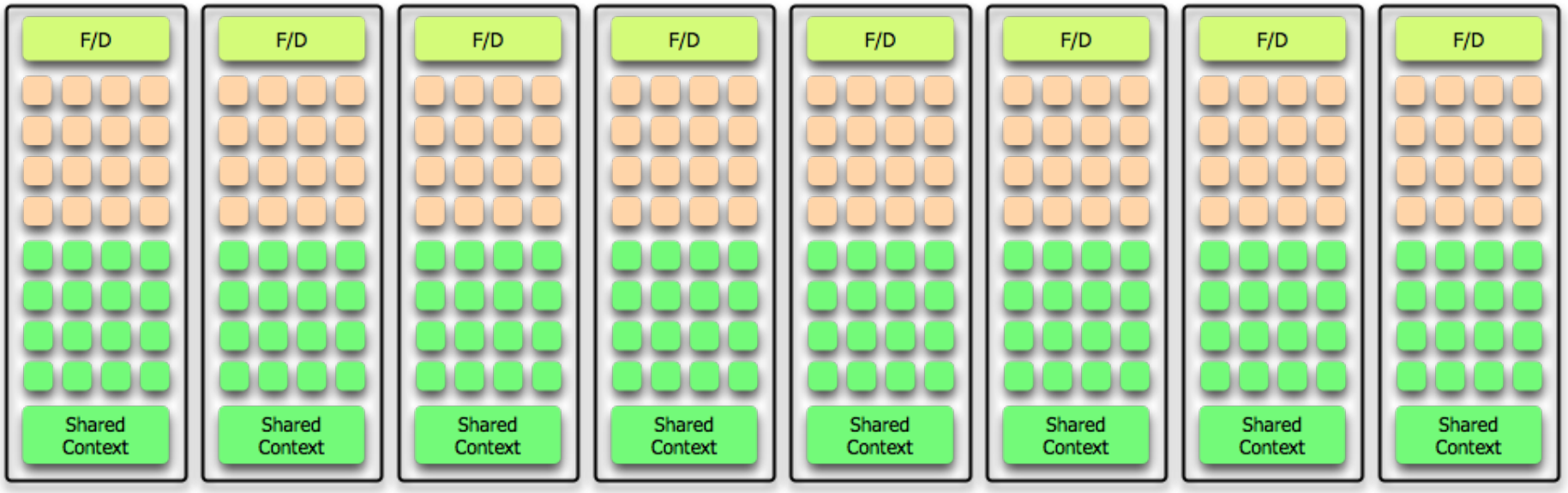
Hardware Context Switch

- To avoid stalls caused by high latency operations
- A single SM can run more than one warp
  - A warp is a group of instruction execution instances and is the unit of context switch
  - All threads in a warp execute the same instruction at a time

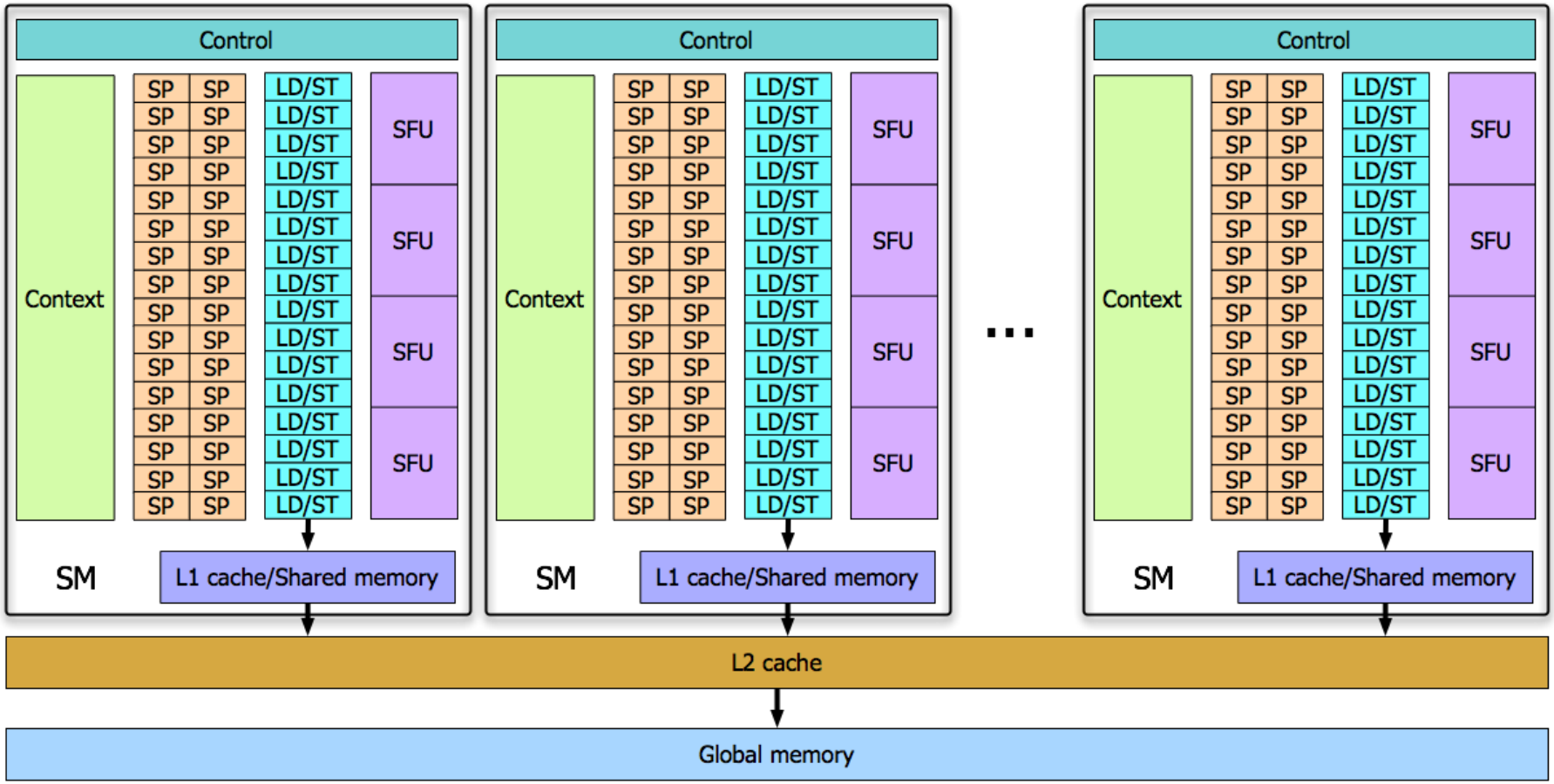


Multiple Streaming Multiprocessors

- Different streaming multiprocessors (SMs) execute different shaders
- N SMs
  - N simultaneous instruction streams



The Overall Picture



GPU Summary

- Exploits massive data parallelism
  - Many simple compute units
  - SIMD (a single program counter)
  - Many threads and no synchronization between threads (for graphics applications)

- Hardware context switch
  - To tolerate high latencies