EE 457 Unit 9c

Thread Level Parallelism

Credits

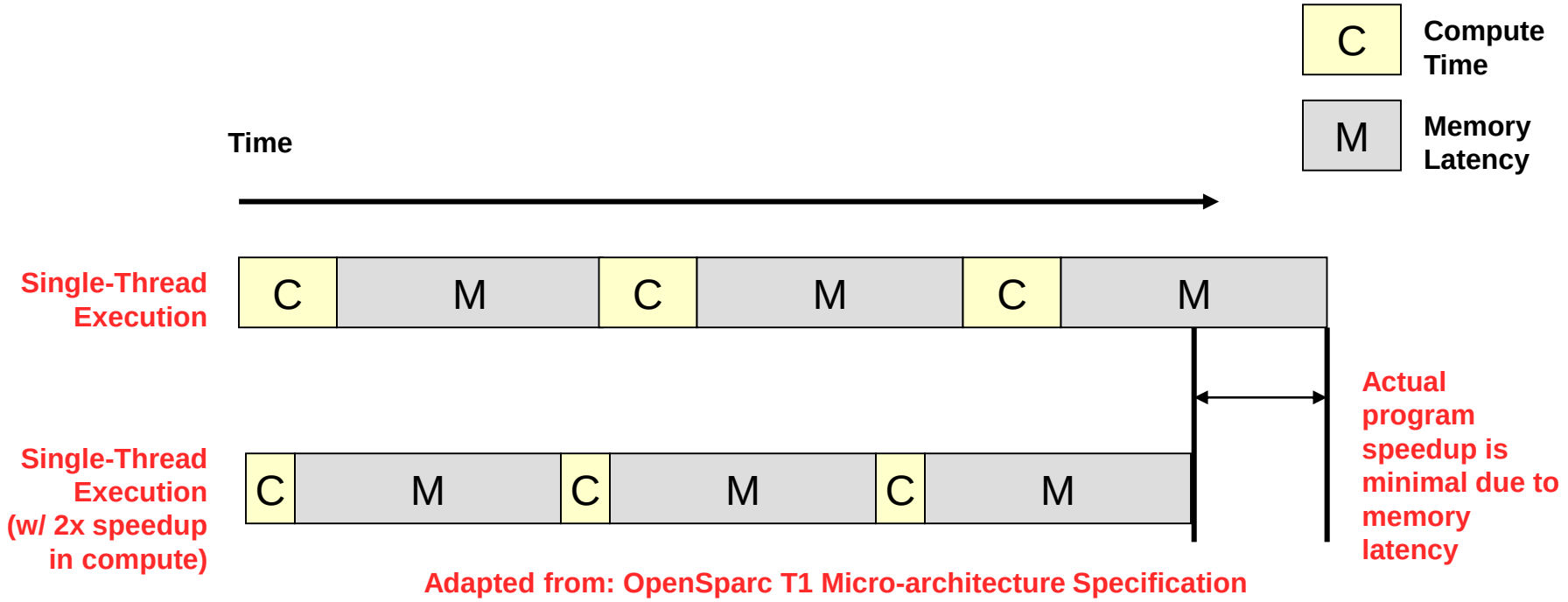
- Some of the material in this presentation is taken from:
  - Computer Architecture: A Quantitative Approach
    - John Hennessy & David Patterson
- Some of the material in this presentation is derived from course notes and slides from
  - Prof. Michel Dubois (USC)
  - Prof. Murali Annavaram (USC)
  - Prof. David Patterson (UC Berkeley)

A Case for Thread-Level Parallelism

CHIP MULTITHREADING AND MULTIPROCESSORS

The Problem with 5-Stage Pipeline

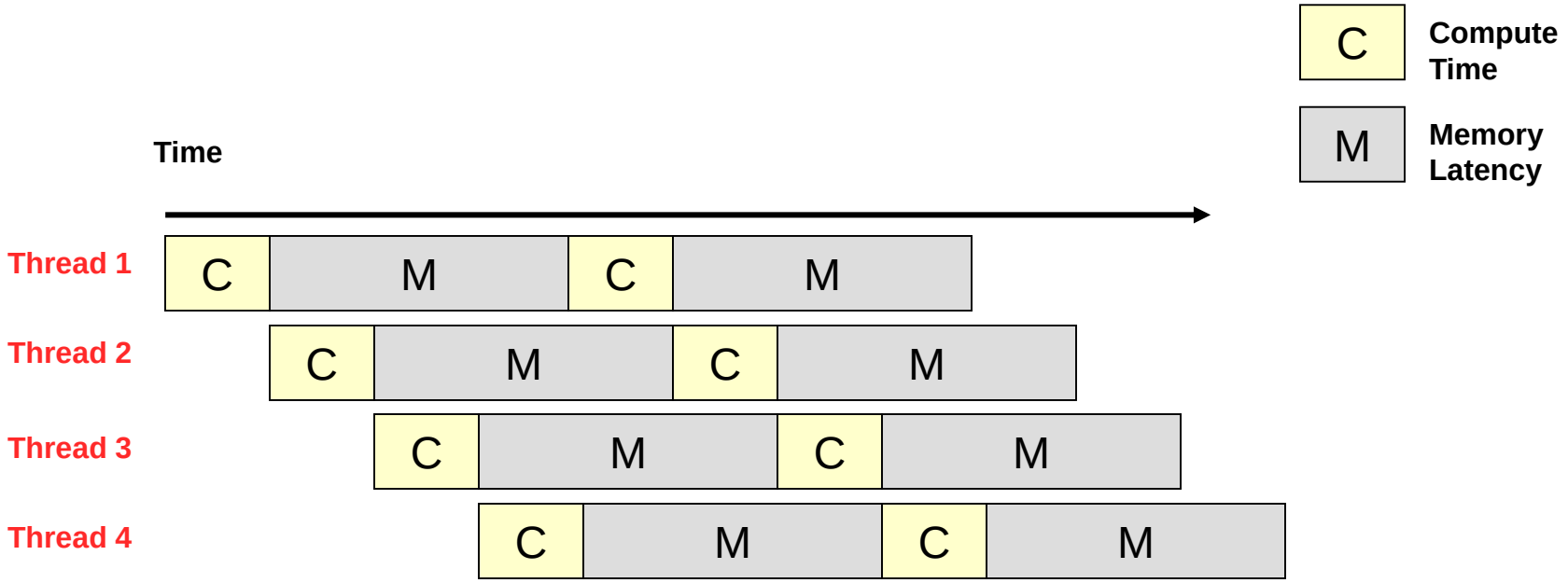
- A cache miss (memory induced stall) causes computation to stall
- A 2x speedup in compute time yields only minimal overall speedup due to memory latency dominating compute



Adapted from: OpenSparc T1 Micro-architecture Specification

Case for Multithreading

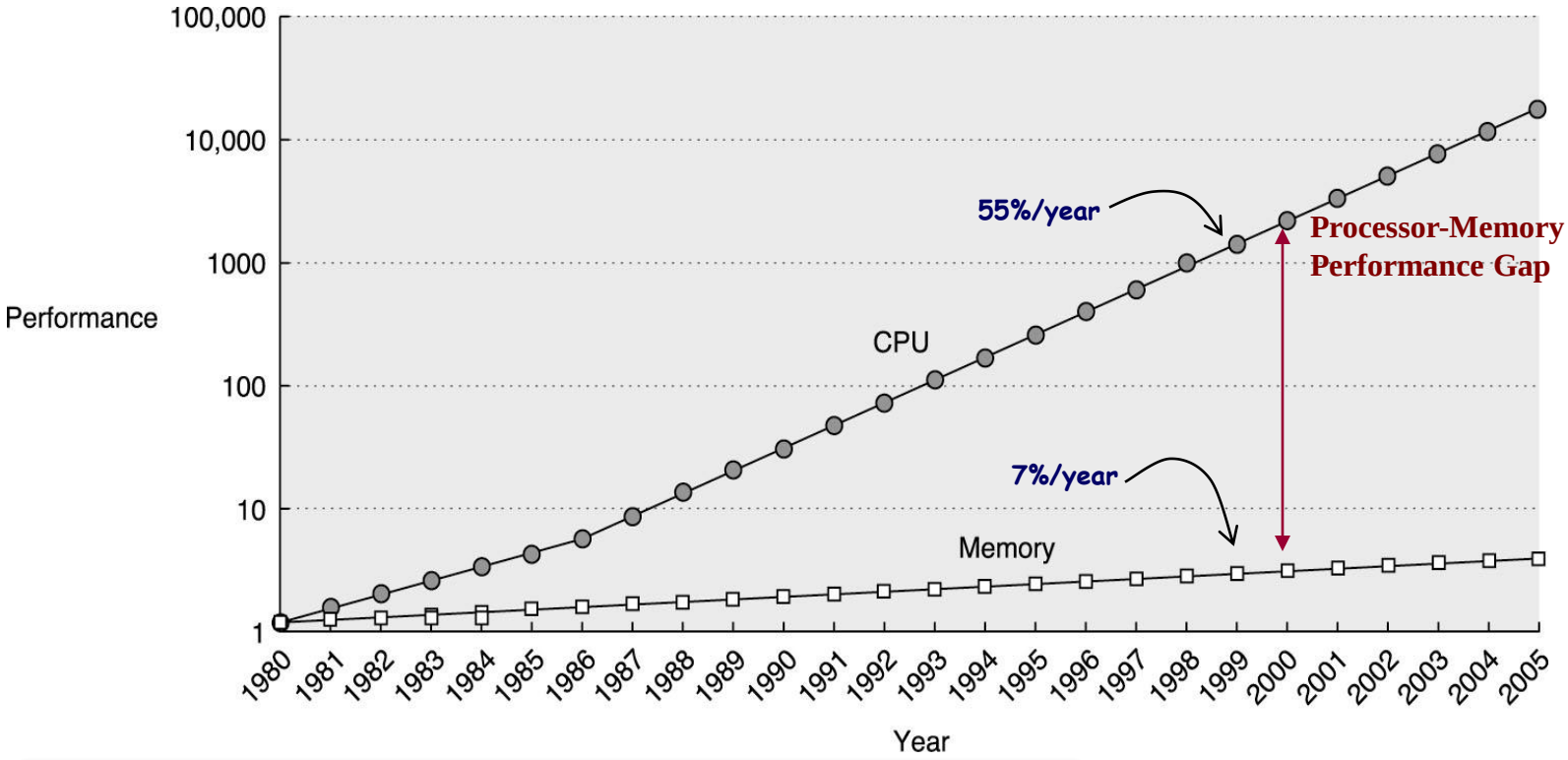
- By executing multiple threads we can keep the processor busy with useful work
- Swap to the next thread when the current thread hits a long-latency even (i.e. cache miss)



Adapted from: OpenSparc T1 Micro-architecture Specification

Memory Wall Problem

- Processor performance is increasing much faster than memory performance



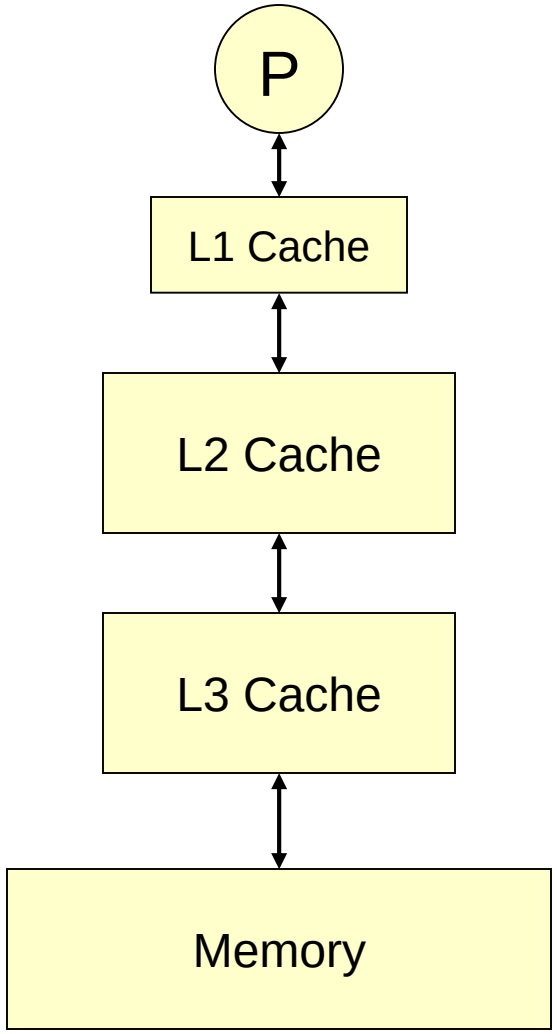
There is a limit to ILP!

If a cache miss requires several hundred clock cyles even OoO pipelines with 10's or 100's of in-flight instructions may stall.

Hennessy and Patterson, *Computer Architecture – A Quantitative Approach* (2003)

Cache Hierarchy

- A hierarchy of cache can help mitigate the cache miss penalty
- L1 Cache
  - 64 KB
  - 2 cycle access time
  - Common Miss Rate ~ 5%
- L2 Cache
  - 1 MB
  - 20 cycle access time
  - Common Miss Rate ~ 1%
- Main Memory
  - 300 cycle access time

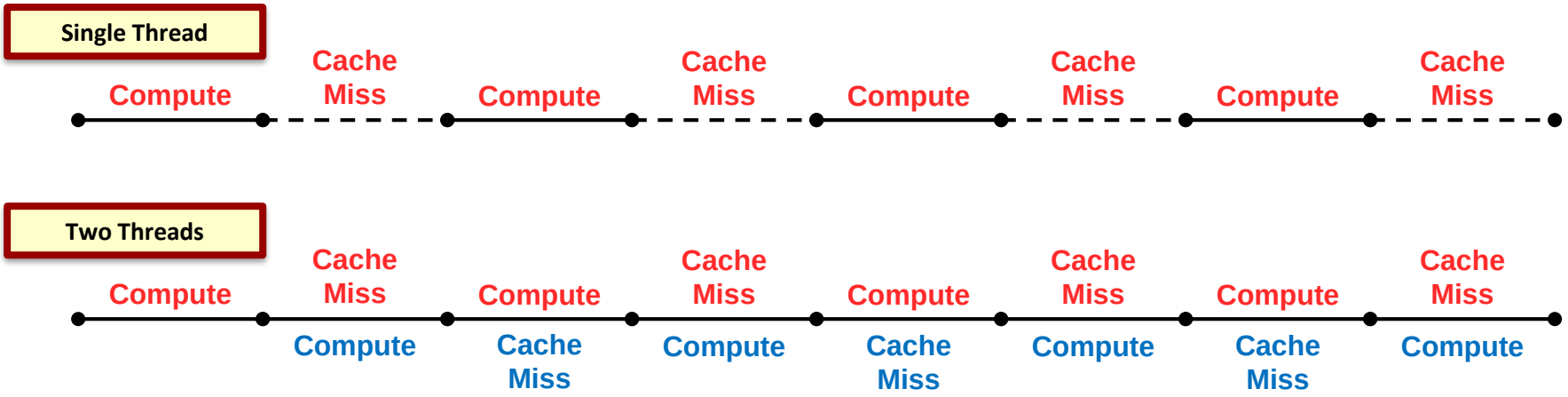


Cache Penalty Example

- Assume an L1 hit rate of 95% and miss penalty of 20 clock cycles (assuming these misses hit in L2). What is the CPI for our typical 5 stage pipeline?
  - 95 instructions take 95 cycles to execute
  - 5 instructions take 105=5+5*20 cycles to execute
  - Total 200 cycles for 100 instructions = CPI of 2

Multithreading

- Long latency events
  - Cache Miss, Exceptions, Lock (Synchronization), Long instructions such as MUL/DIV
- Long latency events cause Io and even OoO pipelines to be underutilized
- Idea: Share the processor among two executing threads, switching when one hits a long latency event
  - Only penalty is flushing the pipeline.



Non-Blocking Caches

- Cache can service hits while fetching one or more miss requests
  - Example: Pentium Pro has a non-blocking cache capable of handling 4 outstanding misses

Software Multithreading

- Used since 1960's to hide I/O latency
  - Multiple processes with different virtual address spaces and process control blocks
  - On an I/O operation, state is saved and another process is given to the CPU
  - When I/O operation completes the process is rescheduled
- On a context switch
  - Trap processor and flush pipeline
  - Save state in process control block (PC, register file, Interrupt vector, page table base register)
  - Restore state of another process
  - Start execution and fill pipeline
- Very high overhead!
- Context switch is also triggered by timer for fairness

Hardware Multithreading

- Run multiple threads in turn on the same core
- Requires additional hardware for fast context switching
  - Multiple register files
  - Multiple state registers (condition codes, interrupt vector, etc.)
  - Avoids saving context manually (via software)

Power

- Power consumption decomposed into:
  - Static: Power constantly being dissipated (grows with # of transistors)
  - Dynamic: Power consumed for switching a bit (1 to 0)
- $P_{DYN} = I_{DYN} * V_{DD} \approx \frac{1}{2} C_{TOT} V_{DD}^2 f$
  - Recall, I = C dV/dt
  - V_{DD} is the logic '1' voltage, f = clock frequency
- Dynamic power favors parallel processing vs. higher clock rates
  - V_{DD} value is tied to f, so a reduction/increase in f leads to similar change in Vdd
  - Implies power is proportional to f^3 (a cubic savings in power if we can reduce f)
  - Take a core and replicate it 4x => 4x performance and 4x power
  - Take a core and increase clock rate 4x => 4x performance and 64x power
- Static power
  - Leakage occurs no matter what the frequency is

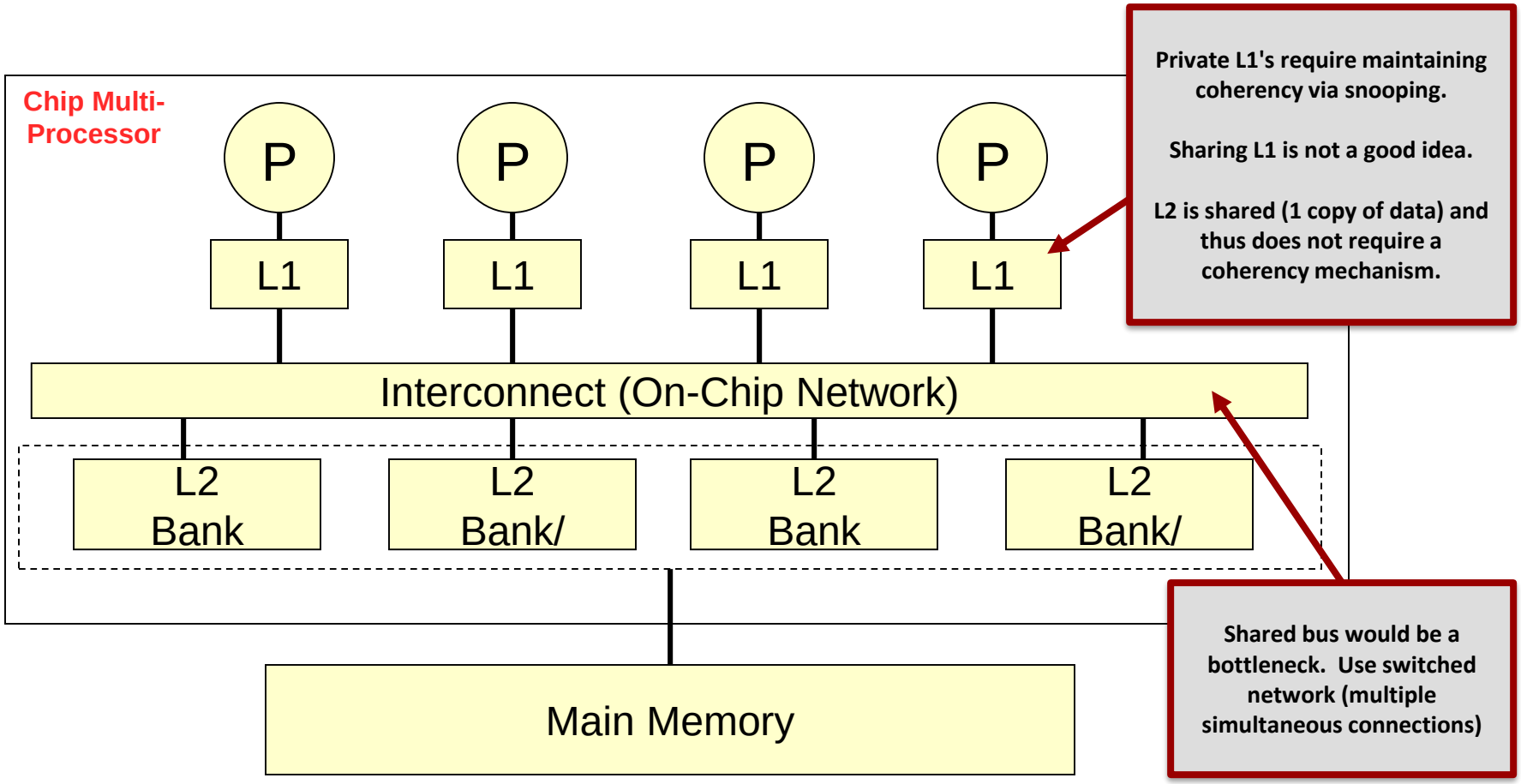
Temperature

- Temperature is related to power consumption
  - Locations on the chip that burn more power will usually run hotter
    - Locations where bits toggle (register file, etc.) often will become quite hot especially if toggling continues for a long period of time
  - Too much heat can destroy a chip
  - Can use sensors to dynamically sense temperature
- Techniques for controlling temperature
  - External measures: Remove and spread the heat
    - Heat sinks, fans, even liquid cooled machines
  - Architectural measures
    - Throttle performance (run at slower frequencies / lower voltages)
    - Global clock gating (pause..turn off the clock)
    - None…results can be catastrophic
- http://www.tomshardware.com/2001/09/17/hot_spot/

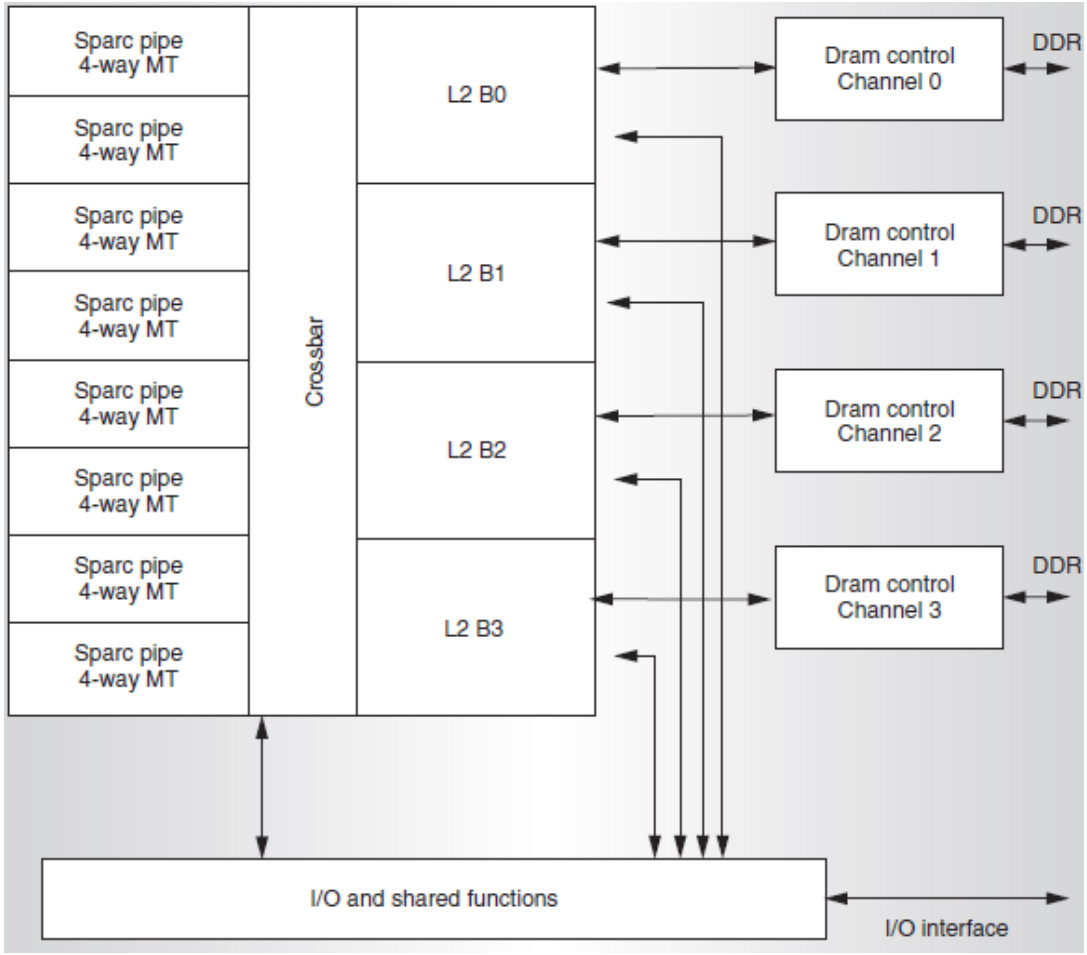
Wire Delay

- In modern circuits wire delay (transmitting the signal) begins to dominate logic delay (time for gate to switch)
- As wires get longer
  - R goes up
  - C goes up
- Dynamically scheduled, OoO processors require longer wire paths for buses, forwarding, etc.
- Simpler pipelines often lead to local, shorter signal connections (wires)
- CMP is really the only viable choice

Typical CMP Organization



Sun T1 "Niagara" Block Diagram



2005
Ex. of Fine-grained Multithreading

http://ogun.stanford.edu/~kunle/publications/niagra_micro.pdf

Sparc T1 Niagara

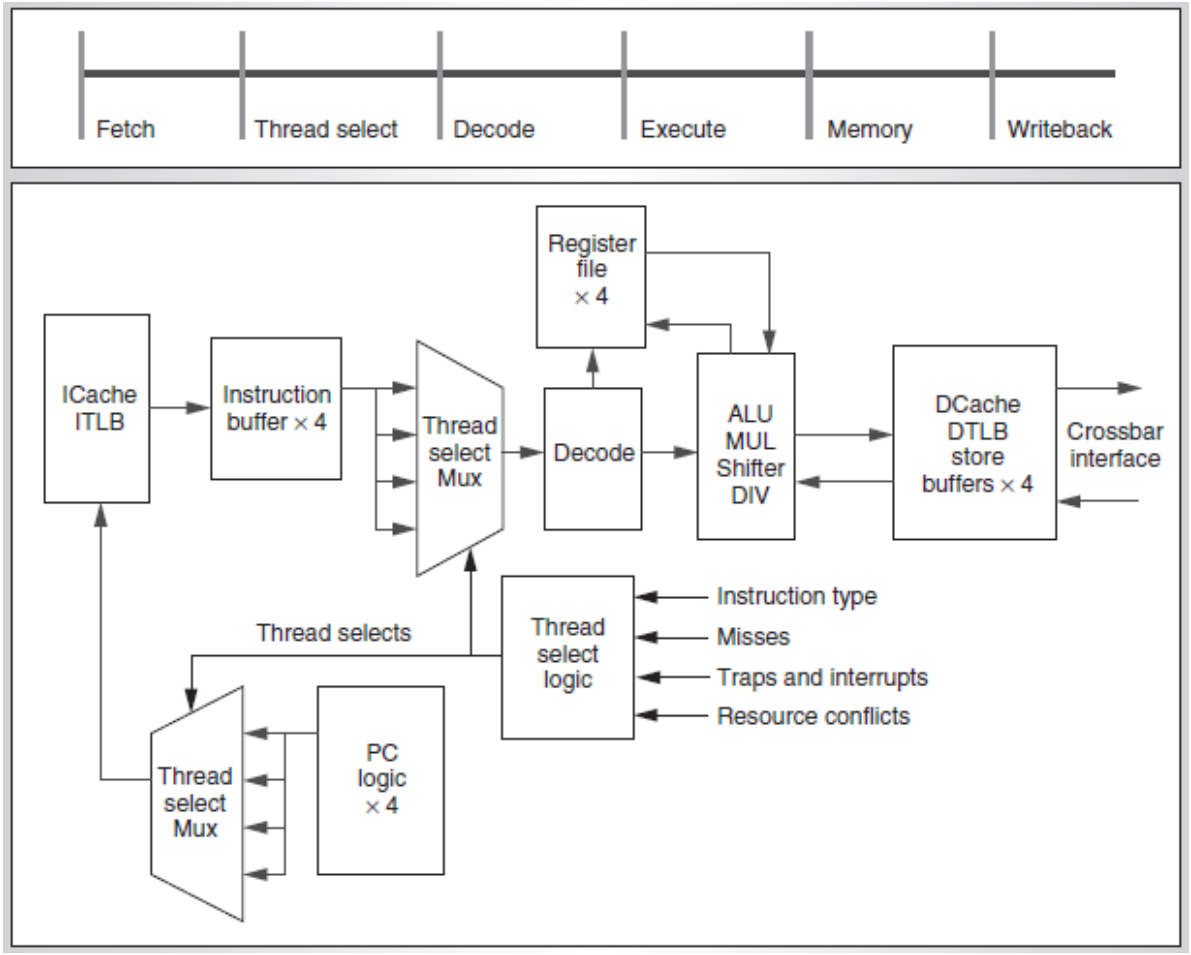
- 8 pipelines each executing 4 threads called a thread group
  - Zero cycle thread switching penalty (round-robin)
  - 6 stage pipeline
- Each pipe has its own L1 cache
- 3 MB shared L2 Cache, 4-banks, 12-way set-associative
  - Is it a problem that it's not a power of 2? No!

Fetch	(Thread) Select	Decode	Exec.	Mem.	WB

T1 Pipeline

- 4-way multithreaded
- Each thread has its own
  - Register file, instruction and store buffers
- Threads share…
  - L1 cache, TLB, and execution units

Sun T1 "Niagara" Pipeline



http://ogun.stanford.edu/~kunle/publications/niagra_micro.pdf

T1 Pipeline

- Fetch stage
  - Thread select mux chooses PC
  - Access I-TLB and I-Cache
  - 2 instructions fetched per cycle
- Thread select stage
  - Choose instructions to issue from ready threads
  - Issues based on
    - Instruction type
    - Misses
    - Resource conflicts
    - Traps and interrupts

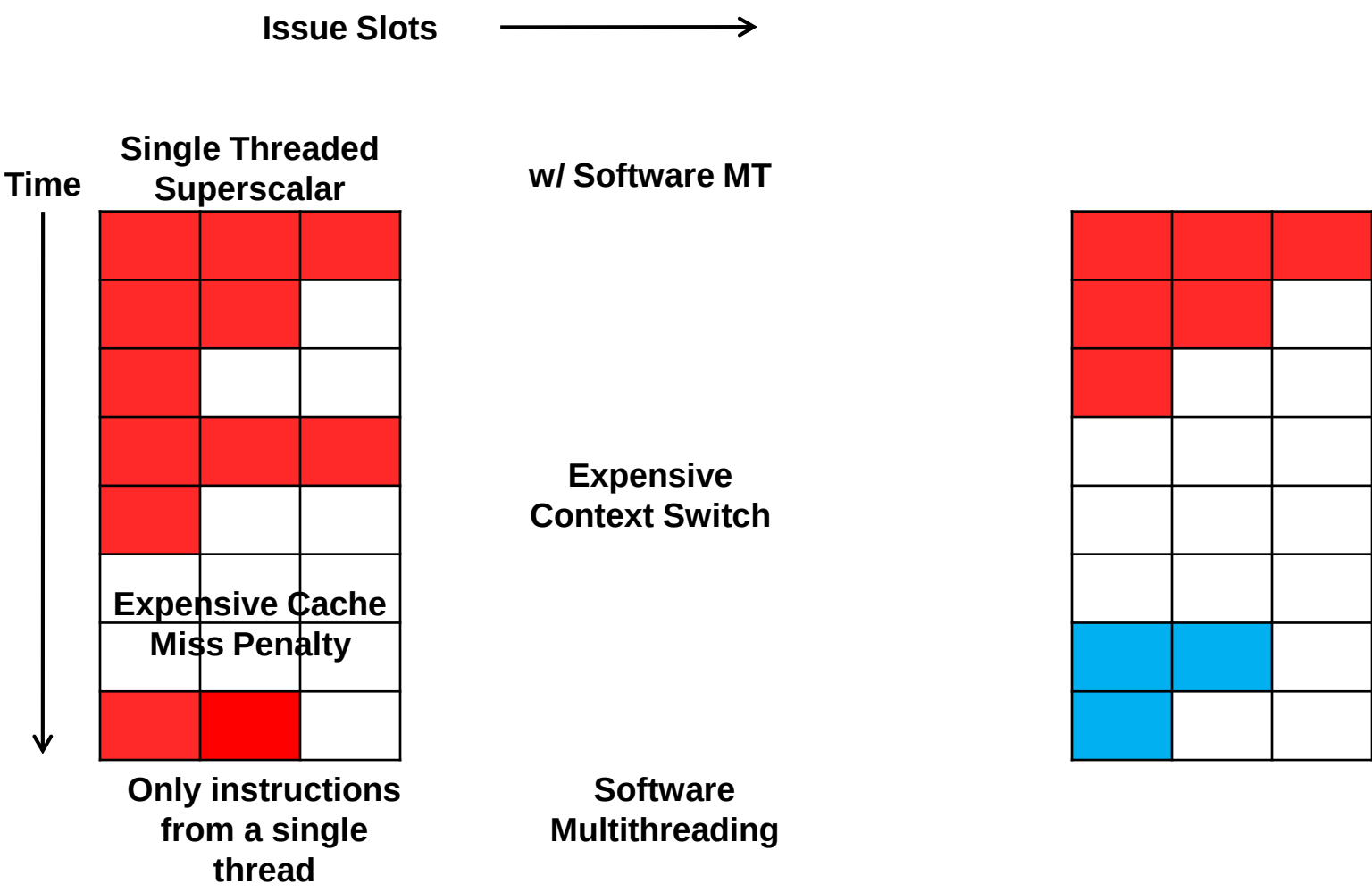
T1 Pipeline

- Decode stage
  - Accesses register file
- Execute Stage
  - Includes ALU, shifter, MUL and DIV units
  - Forwarding Unit
- Memory stage
  - DTLB, Data Cache, and 4 store buffers (1 per thread)
- WB
  - Write to register file

Pipeline Scheduling

- No pipeline flush on context switch (except on cache miss)
- Full forwarding/bypassing to younger instructions of same thread
- In case of load, wait 2 cycles before an instruction from the same thread is issued
  - Solved forwarding latency issue
- Scheduler guarantees fairness between threads by prioritizing the least recently scheduled thread

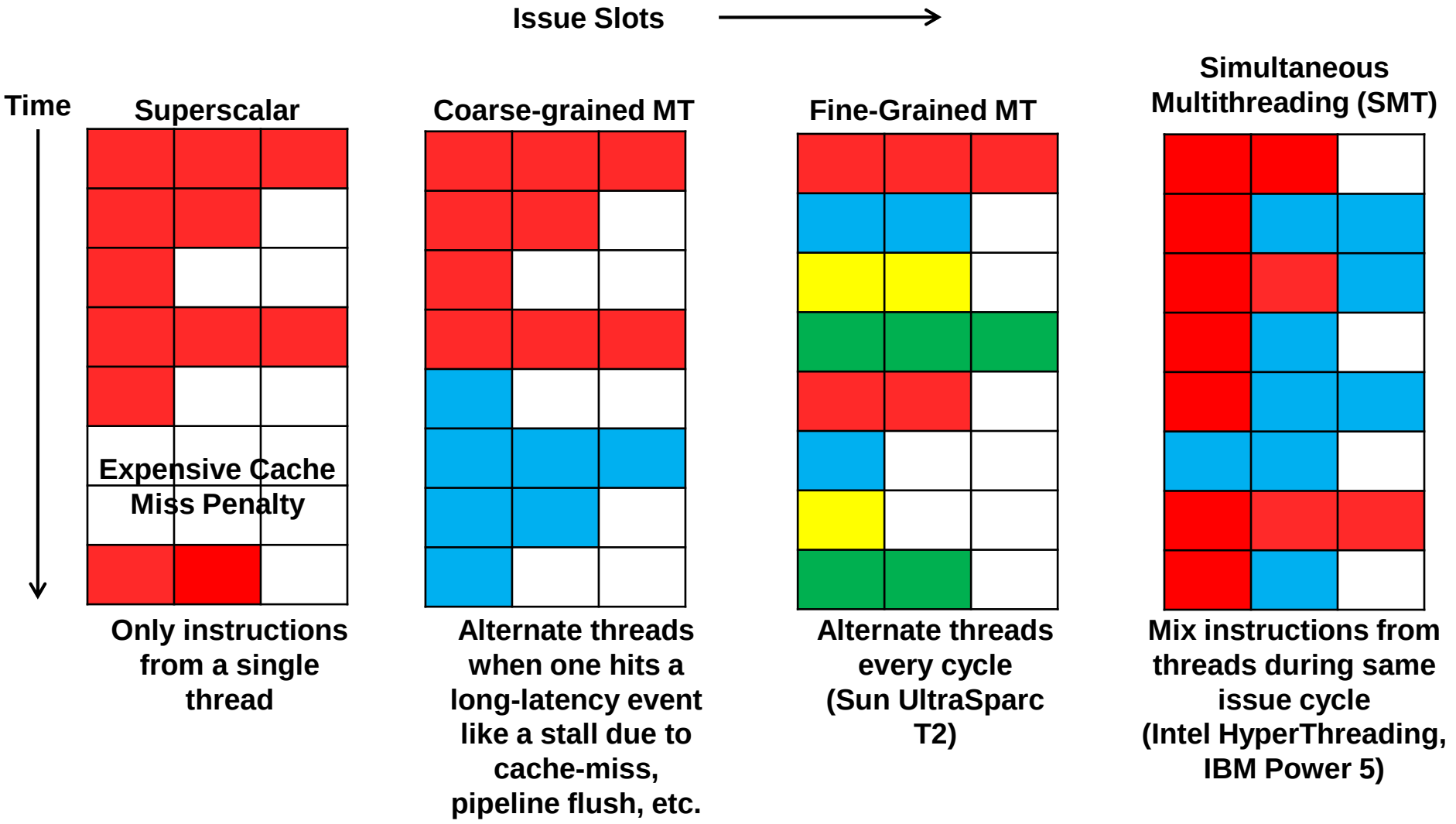
A View Without HW Multithreading



Types/Levels of Multithreading

- How should we overlap and share the HW between instructions from different threads
  - Coarse-grained Multithreading: Execute one thread with all HW resource until a cache-miss or misprediction will incur a stall or pipeline flush, then switch to another thread
  - Fine-grained Multithreading: Alternate fetching instructions from a different thread each clock
  - Simultaneous Multithreading: Fetch and execute instructions from different threads at the same time

Levels of TLP



Fine Grained Multithreading

- Like Sun Niagara
- Alternates issuing instructions from different threads each cycle provided a thread has instructions ready to execute (i.e. not stalled)
- With enough threads, long latency events can be hidden
- Degrades single thread performance since it only gets 1 out of every N cycles if all N threads are ready

Coarse Grained Multithreading

- Swaps threads on long-latency event
- Hardware does not have to swap threads in a single cycle (as in fine-grained multithreading) but can take a few cycles since the current thread has hit a long latency event
- Requires flushing pipeline of current thread's instructions and filling pipeline with new thread's
- Better single-thread performance

ILP and TLP

- TLP can also help ILP by providing another source of independent instructions
- In a 3- or 4-way issue processor, better utilization can be achieved when instructions from 2 or more threads are executed simultaneously

Simultaneous Multithreading

- Uses multiple-issue, dynamic scheduling mechanisms to execute instructions from multiple threads at the same time by filling issue slots with as many available instructions from either thread
  - Overcome poor utilization due to cache misses or lack of independent instructions
  - Requires HW to tag instructions based on their thread
- Requires greater level of hardware resources (separate register renamer, status, and multiple register files, etc.)

Example

- Intel HyperThreading Technology (HTT) is essentially SMT
- Recent processors including Core i7 are multi-core, multi-threaded, multi-issue, OoO (dynamically scheduled) superscalar processors

Future of Multicore/Multithreaded

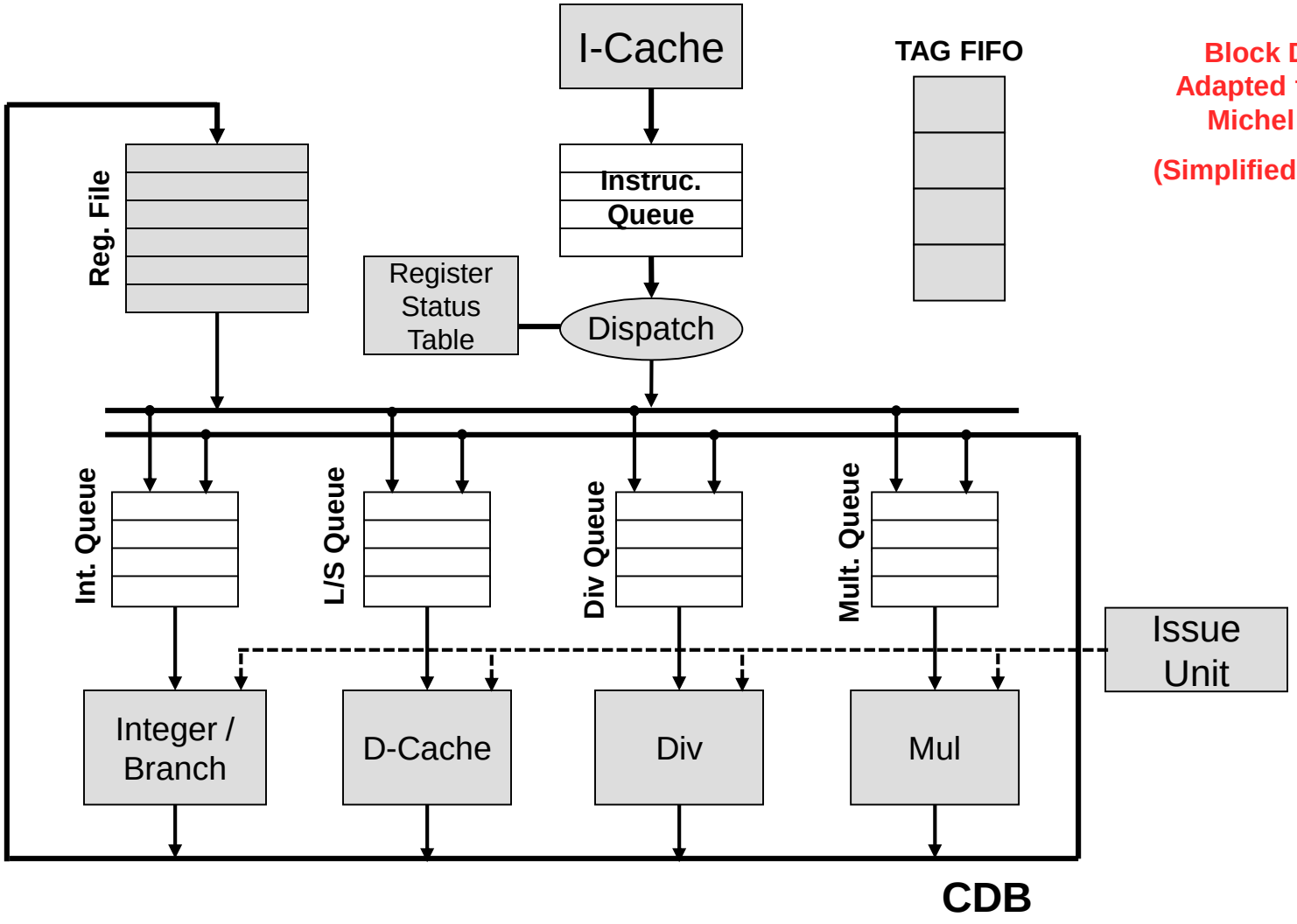
- Multiple cores in shared memory configuration
- Per-core L1 or even L2
- Large on-chip shared cache
- Multiple threads on each core to fight memory wall
- Ever increasing on-chip threads
  - To continue to meet Moore's Law
  - CMP's with 1000's of threads envisioned
  - Only sane option from technology perspective (i.e. out of necessity)
  - The big road block is parallel programming

Parallel Programming

- Implicit parallelism via…
  - Parallelizing compilers
  - Programming frameworks (e.g. MapReduce)
- Explicit parallelism
  - OpenMP
  - Task Libraries
    - Intel Thread Building Blocks, Java Task Library
  - Native threading (Windows threads, POSIX threads)
  - MPI

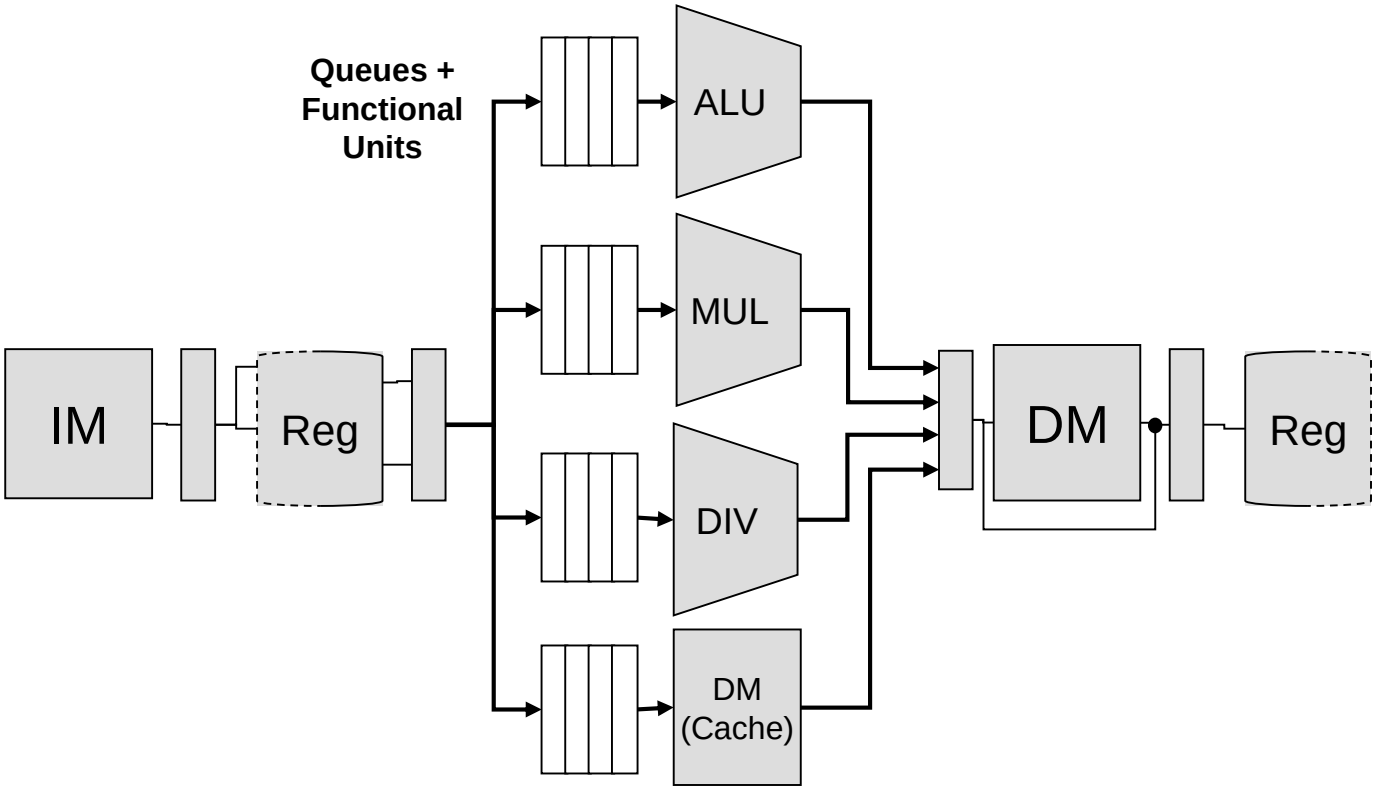
BACKUP

Organization for OoO Execution



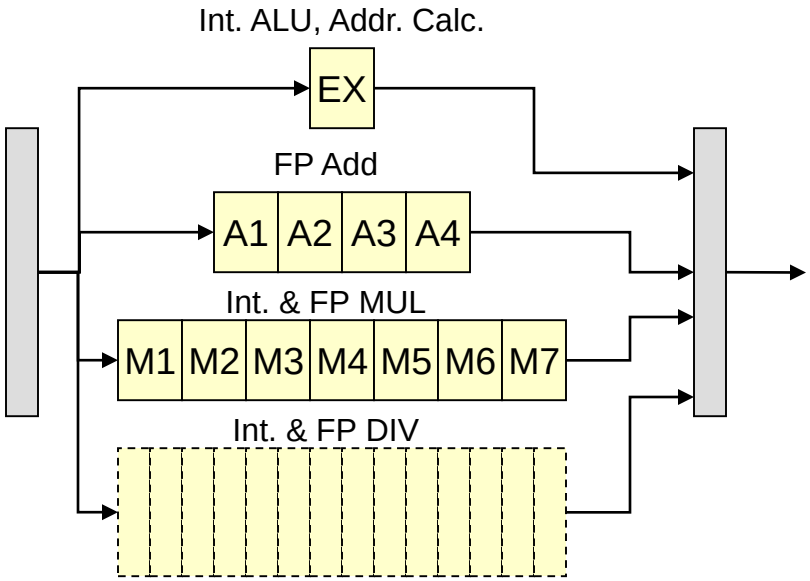
Multiple Functional Units

- We now provide multiple functional units
- After decode, issue to a queue, stalling if the unit is busy or waiting for data dependency to resolve



Functional Unit Latencies

An added complication of out-of-order execution & completion: *WAW* & *WAR* hazards

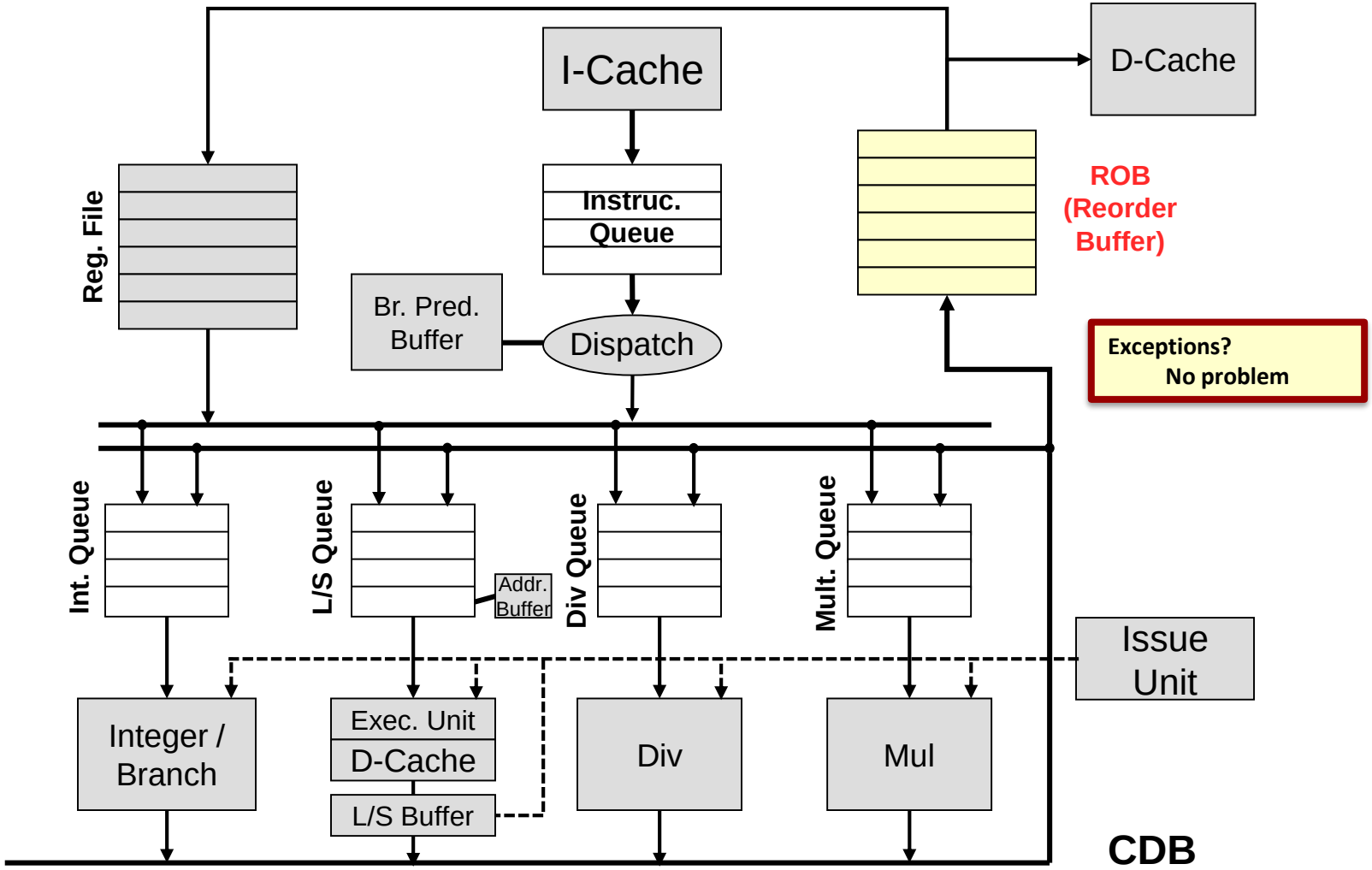


Look Ahead: Tomasulo Algorithm will help absorb latency of different functional units and cache miss latency by allowing other ready instruction proceed out-of-order

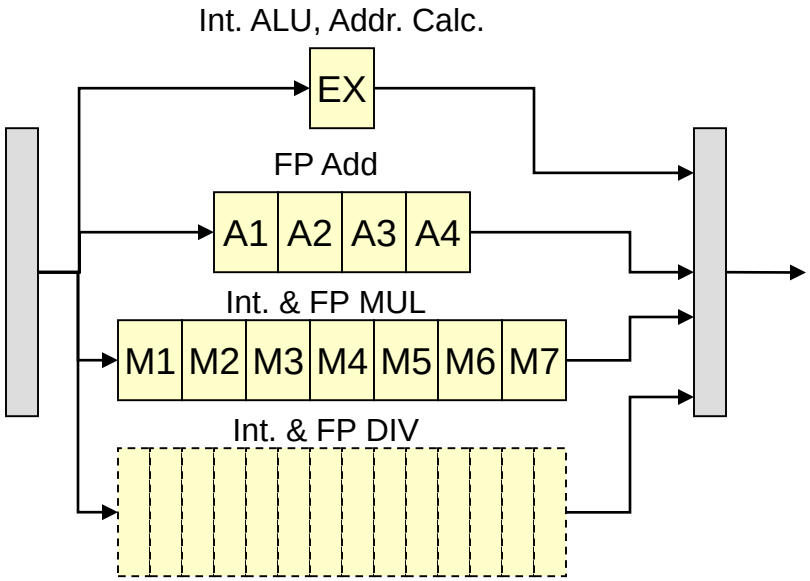
Functional Unit	Latency (Required stalls cycles between dependent [RAW] instrucs.)	Initiation Interval (Distance between 2 independent instructions requiring the same FU)
Integer ALU	0	1
FP Add	3	1
FP Mul.	6	1
FP Div.	24	25

OoO Execution w/ ROB

- ROB allows for OoO execution but in-order completion

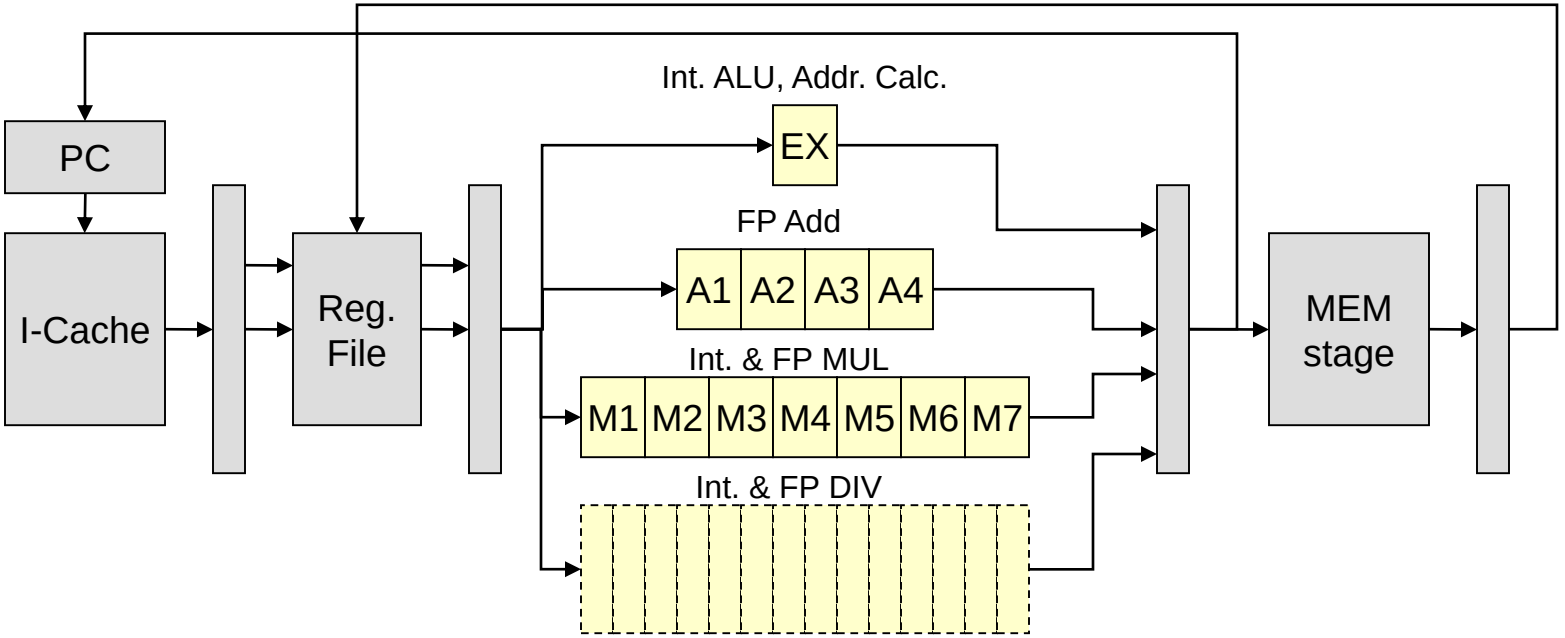


Updated Pipeline



Functional Unit	Latency (Required stalls cycles between dependent [RAW] instrucs.)	Initiation Interval (Distance between 2 independent instructions requiring the same FU)
Integer ALU	0	1
FP Add	3	1
FP Mul.	6	1
FP Div.	24	25

Updated Pipeline



Functional Unit	Latency (Required stalls cycles between dependent [RAW] instrucs.)	Initiation Interval (Distance between 2 independent instructions requiring the same FU)
Integer ALU	0	1
FP Add	3	1
FP Mul.	6	1
FP Div.	24	25