Computer Architecture
ELE 475 / COS 475
Slide Deck 12: Multithreading

David Wentzlaff

Department of Electrical Engineering

Princeton University

PRINCETON UNIVERSITY

PRINCETON
School of Engineering and Applied Science

Agenda

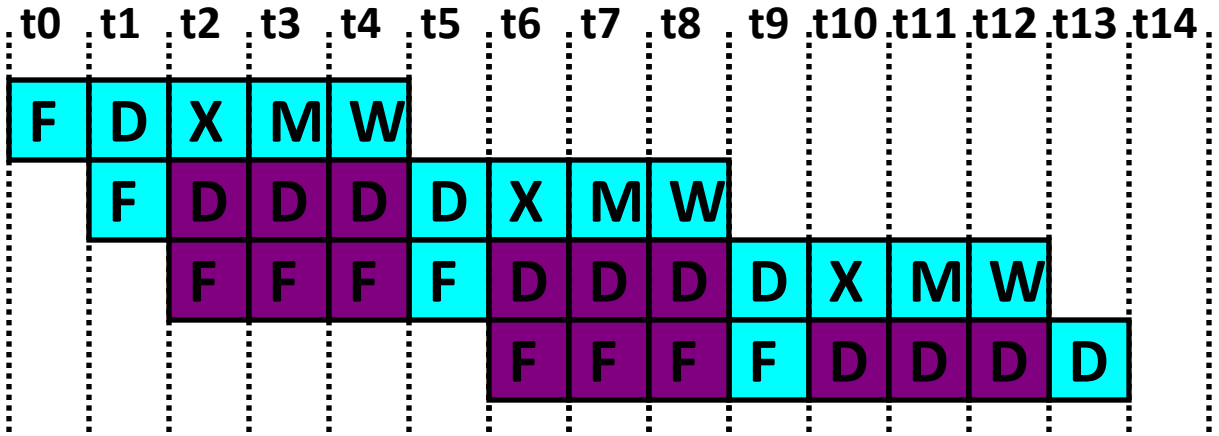
- Multithreading Motivation
- Course Grain Multithreading
- Simultaneous Multithreading

Multithreading

- Difficult to continue to extract instruction-level parallelism (ILP) or data level parallelism (DLP) from a single sequential thread of control
- Many workloads can make use of thread-level parallelism (TLP)
  - TLP from multiprogramming (run independent sequential jobs)
  - TLP from multithreaded applications (run one job faster using parallel threads)
- Multithreading uses TLP to improve utilization of a single processor

Pipeline Hazards

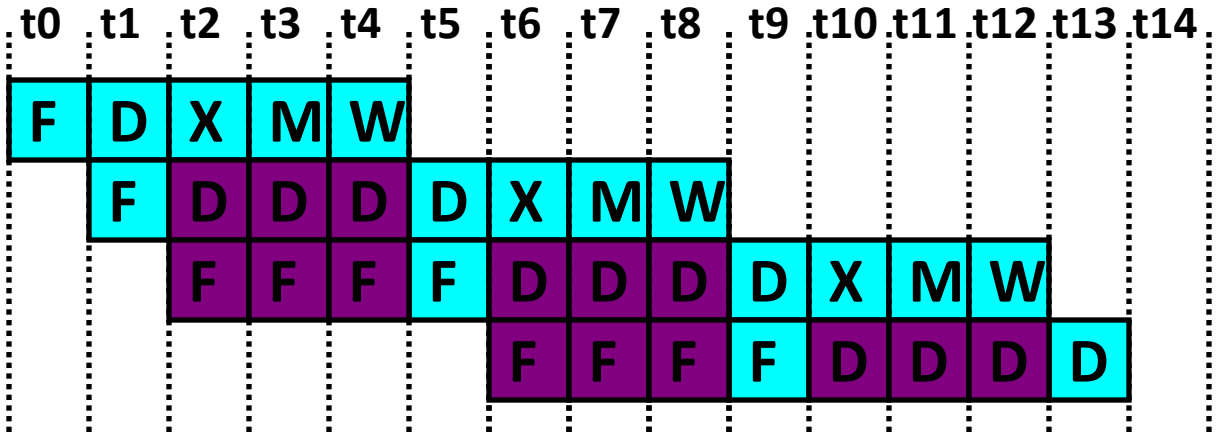
LW r1, 0(r2)
LW r5, 12(r1)
ADDI r5, r5, #12
SW 12(r1), r5



- Each instruction may depend on the next

Pipeline Hazards

LW r1, 0(r2)
LW r5, 12(r1)
ADDI r5, r5, #12
SW 12(r1), r5

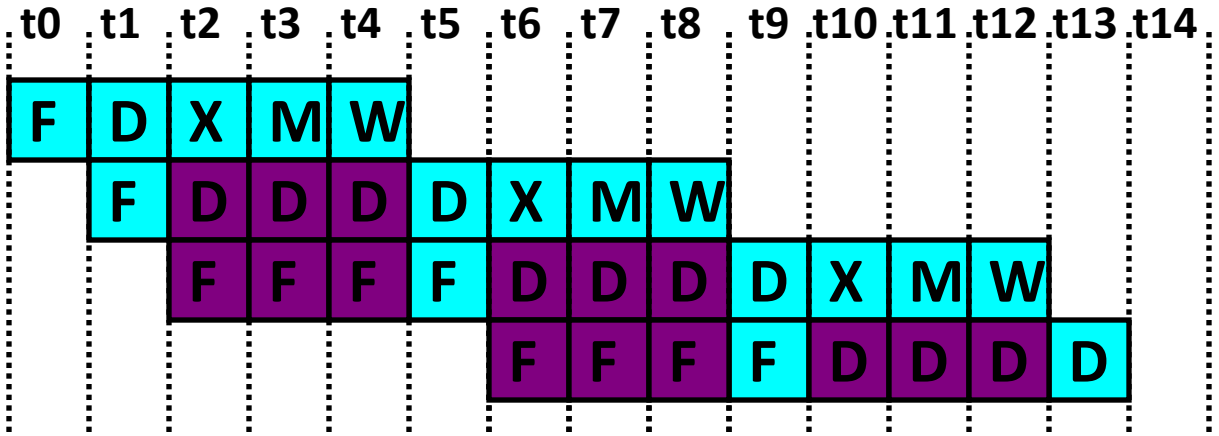


- Each instruction may depend on the next

What is usually done to cope with this?

Pipeline Hazards

LW r1, 0(r2)
LW r5, 12(r1)
ADDI r5, r5, #12
SW 12(r1), r5



- Each instruction may depend on the next

What is usually done to cope with this?

- *interlocks (slow)*
- *or bypassing (needs hardware, doesn't help all hazards)*

Multithreading

How can we guarantee no dependencies between instructions in a pipeline?

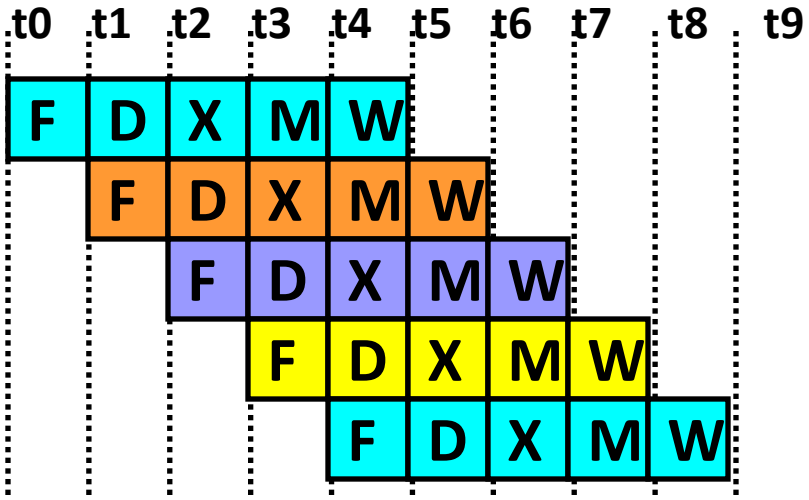
Multithreading

How can we guarantee no dependencies between instructions in a pipeline?

-- One way is to interleave execution of instructions from different program threads on same pipeline

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

T1: LW r1, 0(r2)
T2: ADD r7, r1, r4
T3: XORI r5, r4, #12
T4: SW 0(r7), r5
T1: LW r5, 12(r1)



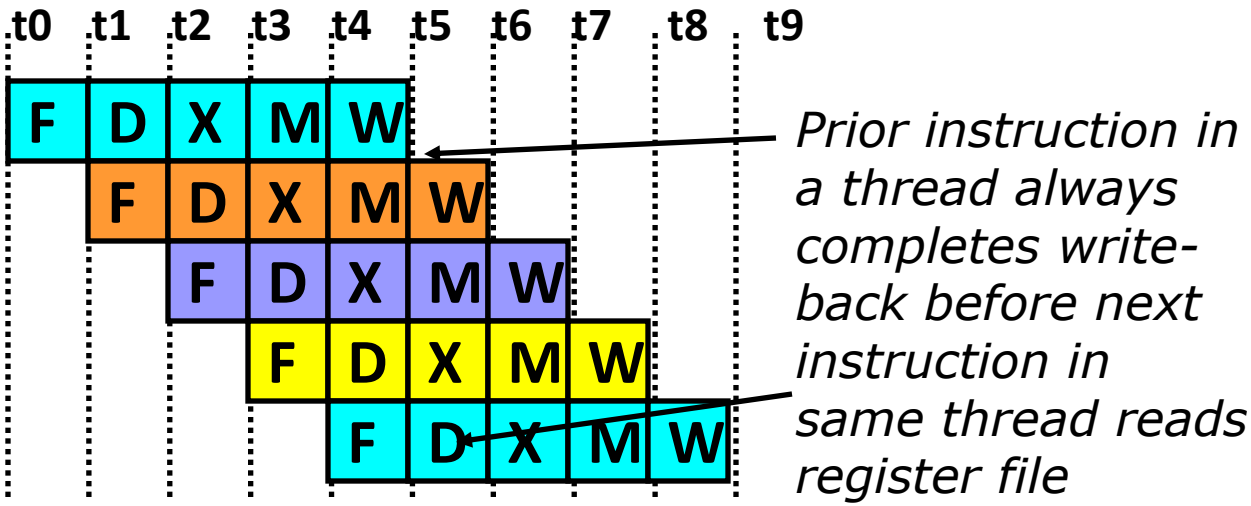
Multithreading

How can we guarantee no dependencies between instructions in a pipeline?

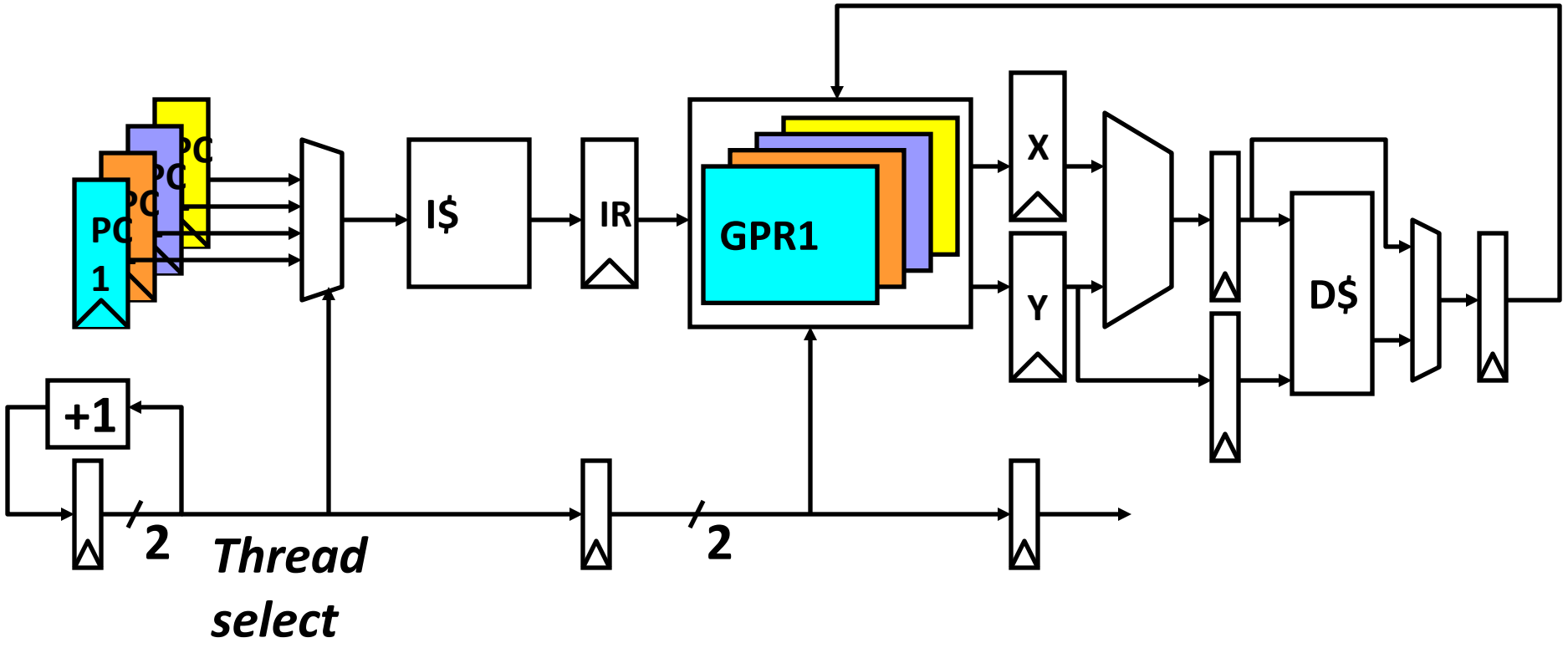
-- One way is to interleave execution of instructions from different program threads on same pipeline

Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

T1: LW r1, 0(r2)
T2: ADD r7, r1, r4
T3: XORI r5, r4, #12
T4: SW 0(r7), r5
T1: LW r5, 12(r1)



Simple Multithreaded Pipeline



- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

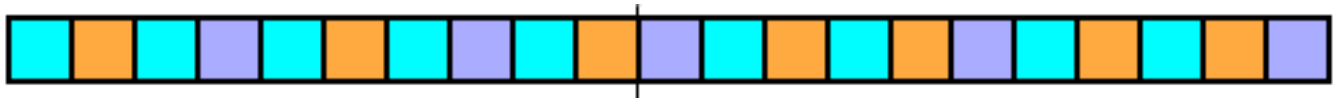
Multithreading Costs

- Each thread requires its own user state
  - PC
  - GPRs
- Also, needs its own system state
  - virtual memory page table base register
  - exception handling registers
  - Other system state
- *Other overheads:*
  - Additional cache/TLB conflicts from competing threads
  - (or add larger cache/TLB capacity)
  - More OS overhead to schedule more threads (where do all these threads come from?)

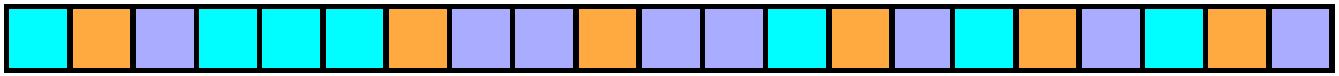
Thread Scheduling Policies

- Fixed interleave *(CDC 6600 PPUs, 1964)*
  - Each of N threads executes one instruction every N cycles
  - If thread not ready to go in its slot, insert pipeline bubble
  - Can potentially remove bypassing and interlocking logic

- Software-controlled interleave *(TI ASC PPUs, 1971)*
  - OS allocates S pipeline slots amongst N threads
  - Hardware performs fixed interleave over S slots, executing whichever thread is in that slot

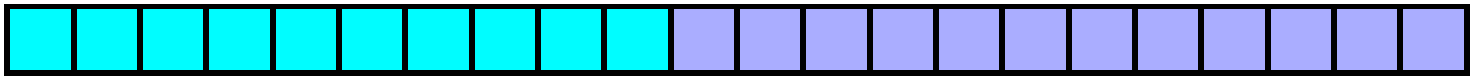


- Hardware-controlled thread scheduling *(HEP, 1982)*
  - Hardware keeps track of which threads are ready to go
  - Picks next thread to execute based on hardware priority scheme



Coarse-Grain Hardware Multithreading

- Some architectures do not have many low-latency bubbles
- Add support for a few threads to hide occasional cache miss latency
- Swap threads in hardware on cache miss



Denelcor HEP

(Burton Smith, 1982)

BRL HEP Machine
Image Credit:
Denelcor

http://ftp.arl.army.mil/ftp/historic-computers/png/hep2.png

First commercial machine to use hardware threading in main CPU

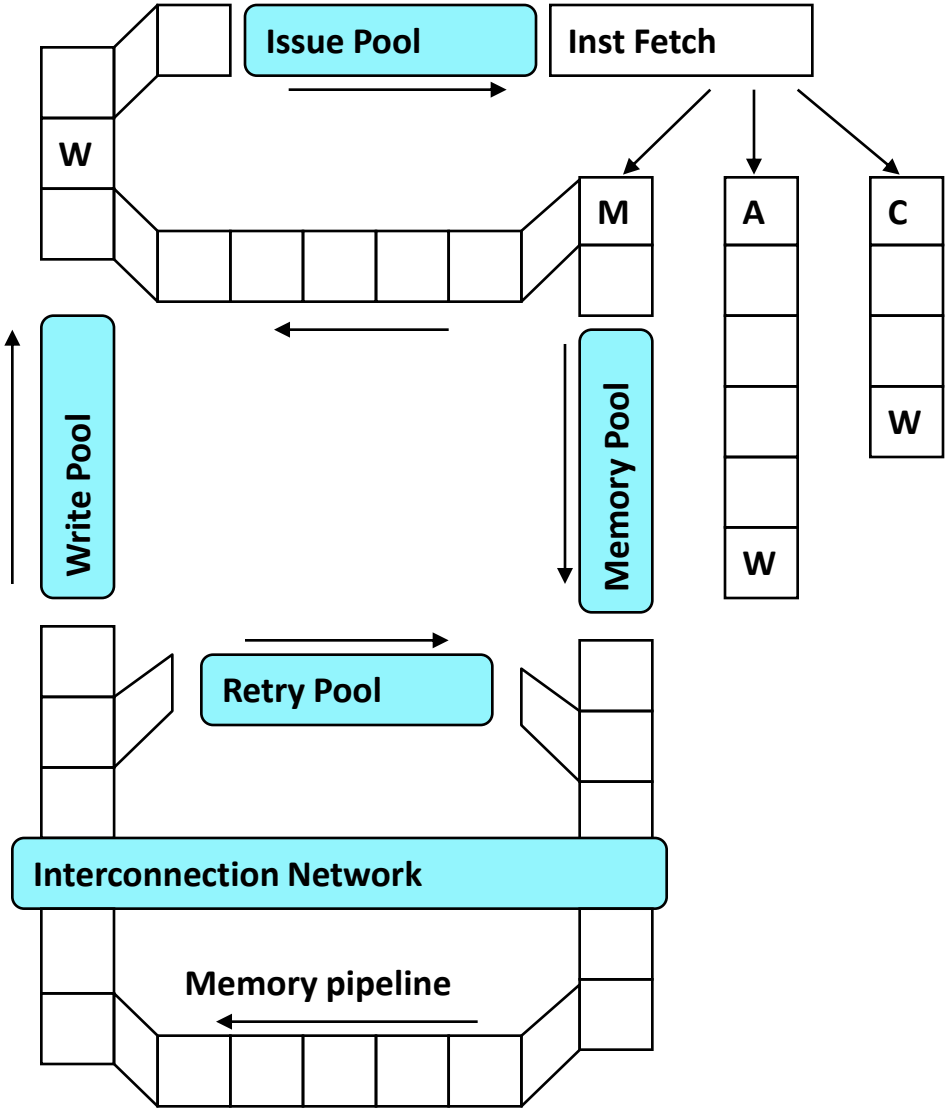
- 120 threads per processor
- 10 MHz clock rate
- Up to 8 processors
- precursor to Tera MTA / Cray XMT (Multithreaded Architecture)

Tera (Cray) MTA (1990)

- Up to 256 processors
- Up to 128 active threads per processor
- Processors and memory modules populate a sparse 3D torus interconnection fabric
- Flat, shared main memory
  - No data cache
  - Sustains one main memory access per cycle per processor
- GaAs logic in prototype, 1KW/processor @ 260MHz
  - Second version CMOS, MTA-2, 50W/processor
  - New version, XMT, fits into AMD Opteron socket, runs at 500MHz

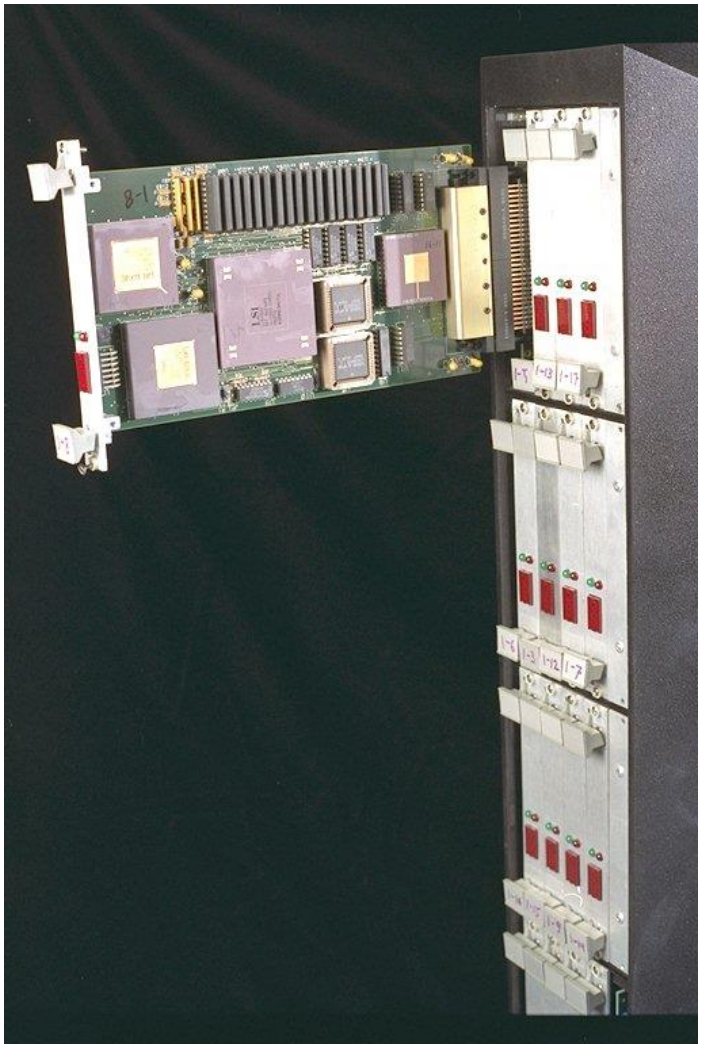
Image Credit:
Tera Computer Company

MTA Pipeline



- Every cycle, one VLIW instruction from one active thread is launched into pipeline
- Instruction pipeline is 21 cycles long
- Memory operations incur ~150 cycles of latency

MIT Alewife (1990)



- Modified SPARC chips
  - register windows hold different thread contexts
- Up to four threads per node
- Thread switch on local cache miss

Image Credit: MIT

Oracle/Sun Niagara processors

- Target is datacenters running web servers and databases, with many concurrent requests
- Provide multiple simple cores each with multiple hardware threads, reduced energy/operation though much lower single thread performance

- Niagara-1 [2004], 8 cores, 4 threads/core
- Niagara-2 [2007], 8 cores, 8 threads/core
- Niagara-3 [2009], 16 cores, 8 threads/core

Oracle/Sun Niagara-3, "Rainbow Falls" 2009

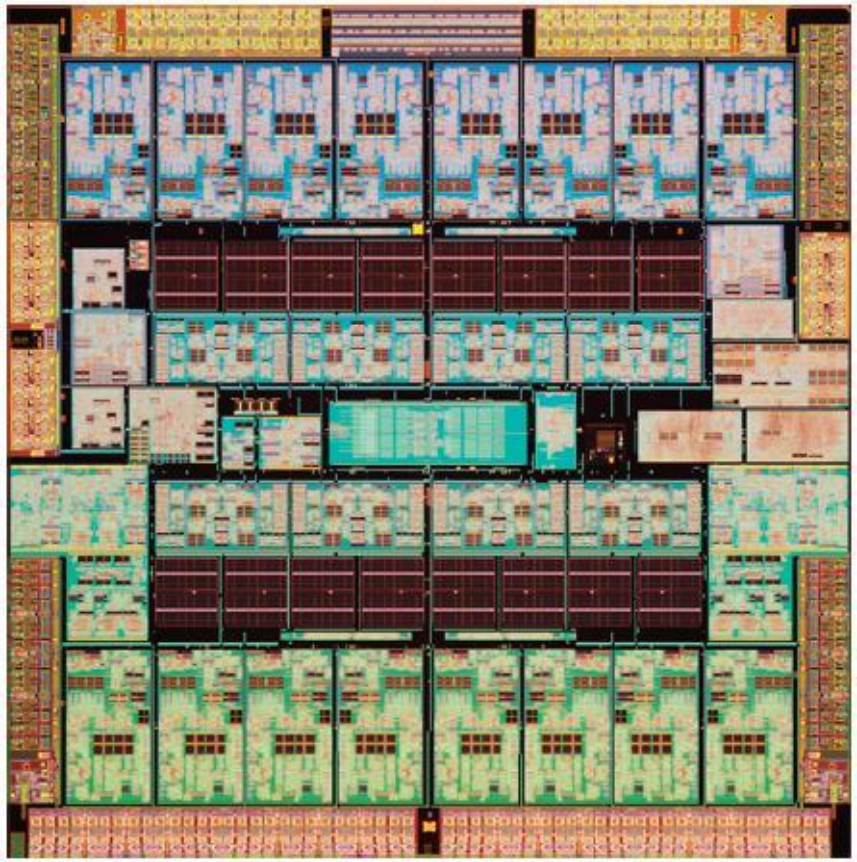


Image Credit: Oracle/Sun

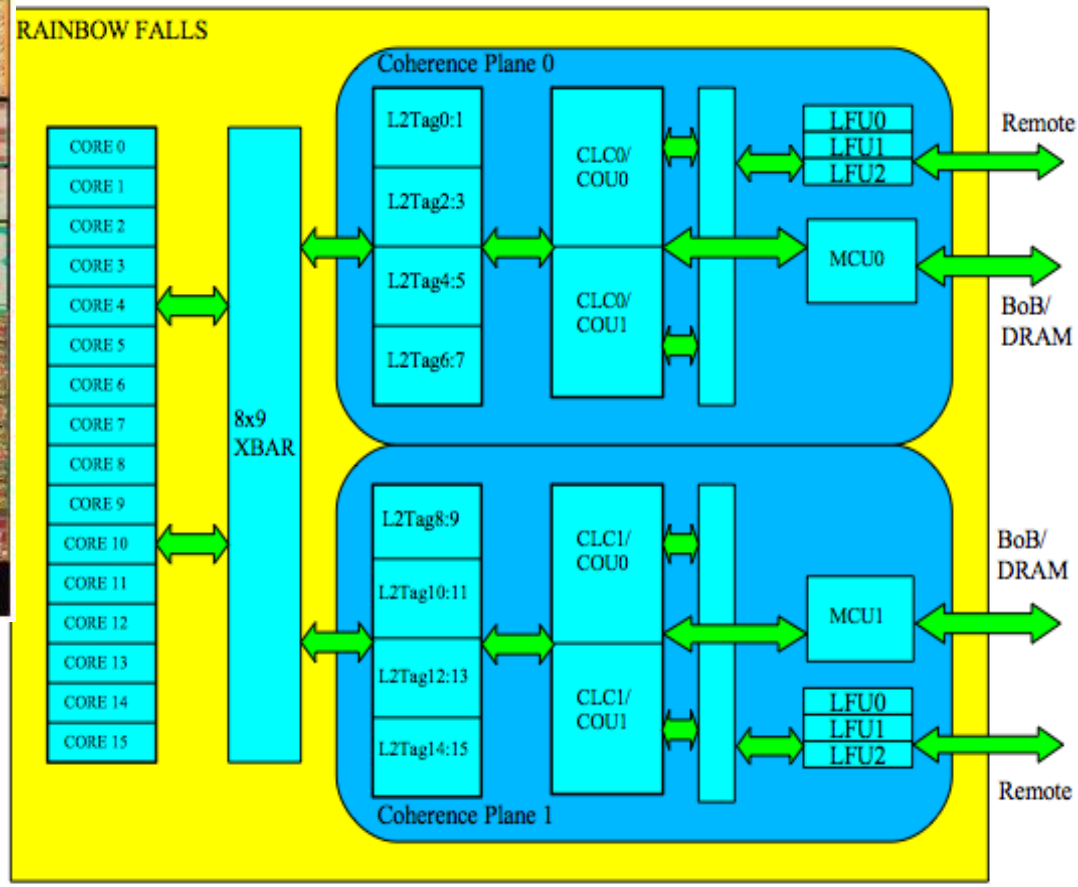


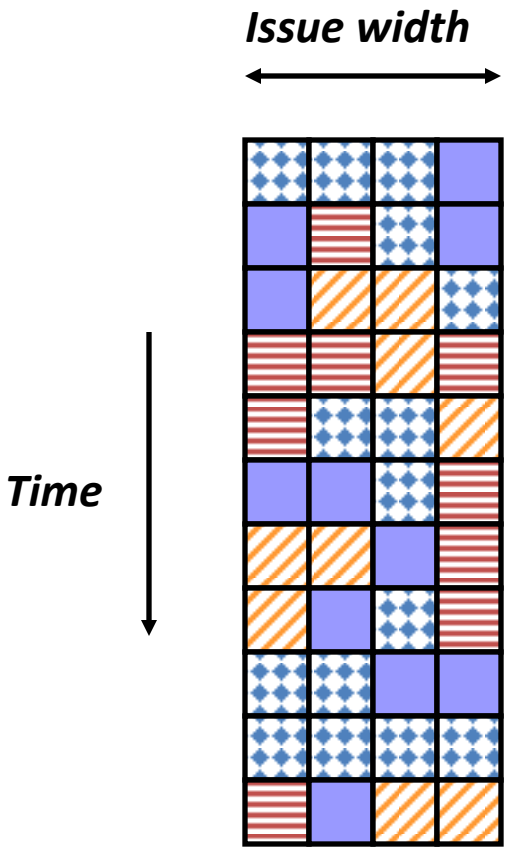
Image Credit: Oracle/Sun
From Hot Chips 2009 Presentation by Sanjay Patel

Simultaneous Multithreading (SMT) for OOO Superscalars

- Techniques presented so far have all been "vertical" multithreading where each pipeline stage works on one thread at a time
- SMT uses fine-grain control already present inside an OOO superscalar to allow instructions from multiple threads to enter execution on same clock cycle. Gives better utilization of machine resources.

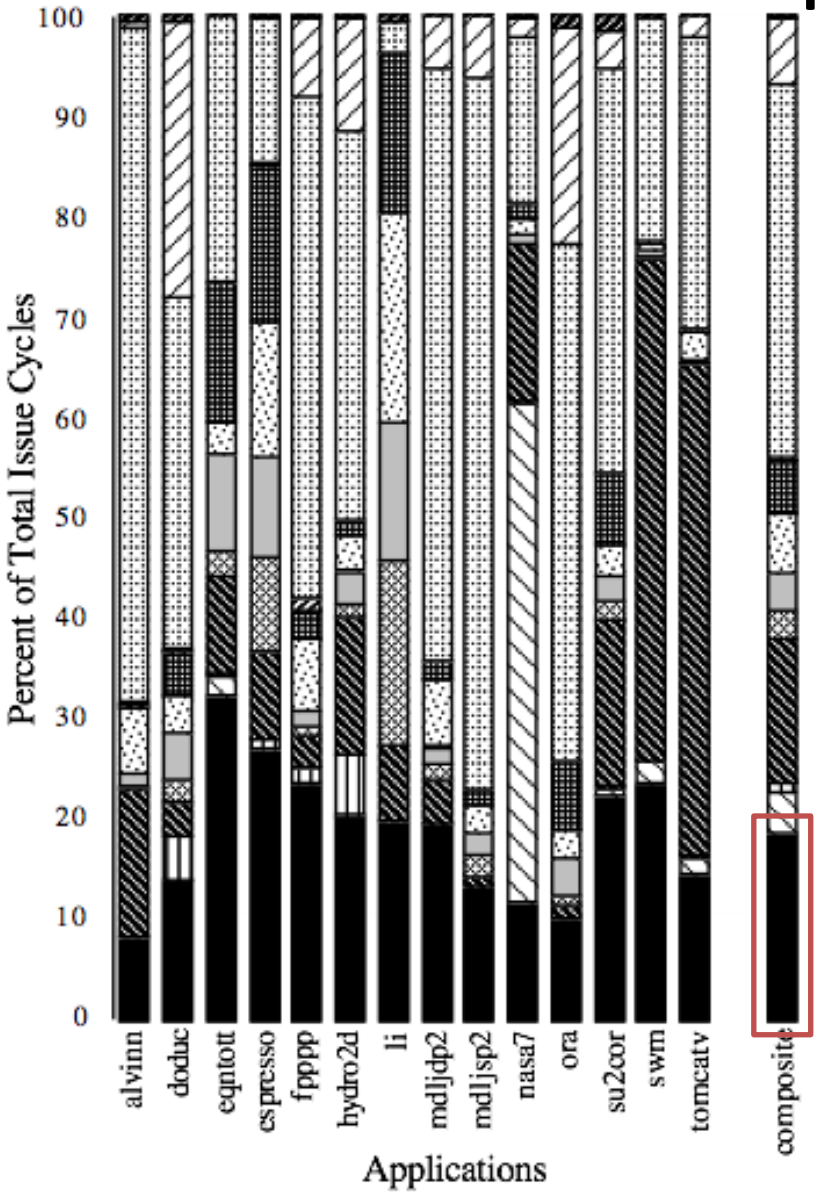
Ideal Superscalar Multithreading

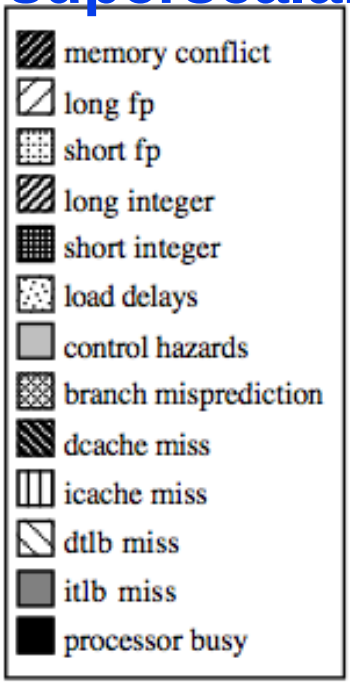
[Tullsen, Eggers, Levy, UW, 1995]



- Interleave multiple threads to multiple issue slots with no restrictions

For most apps, most execution units lie idle in an OOO superscalar

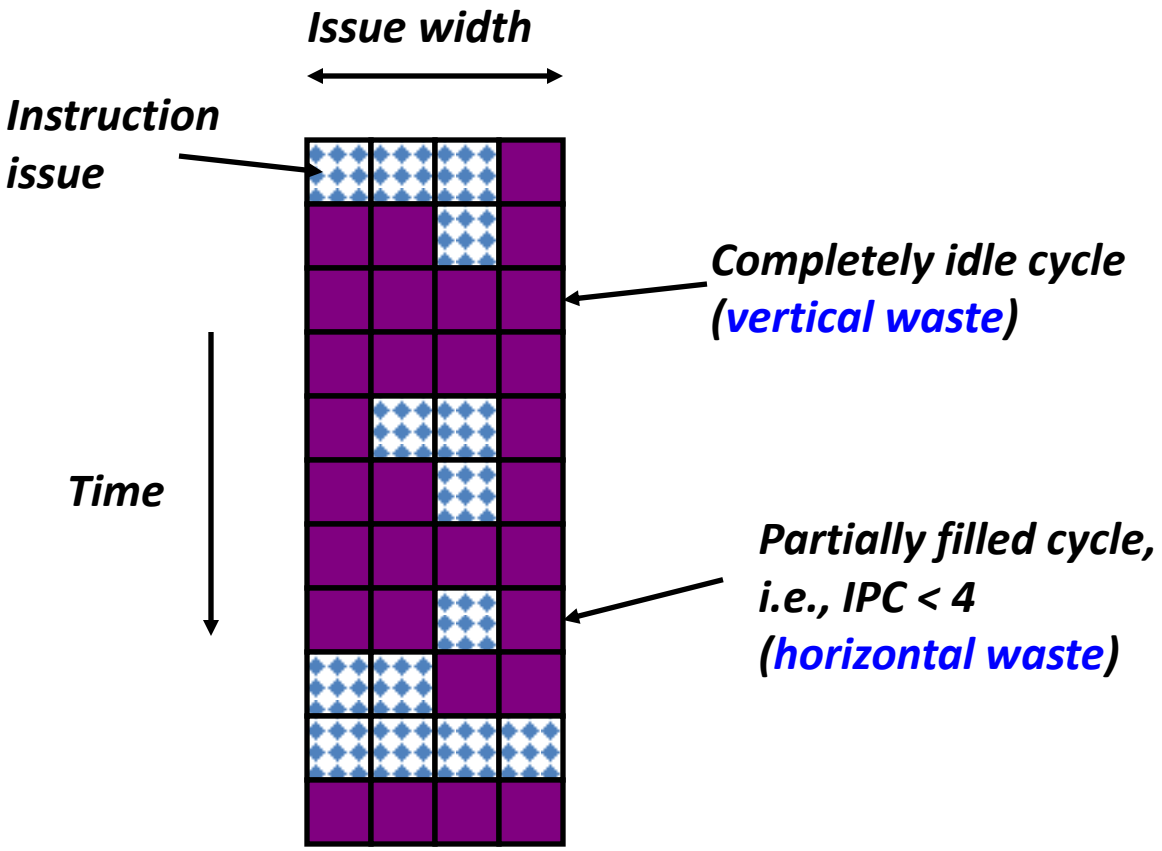




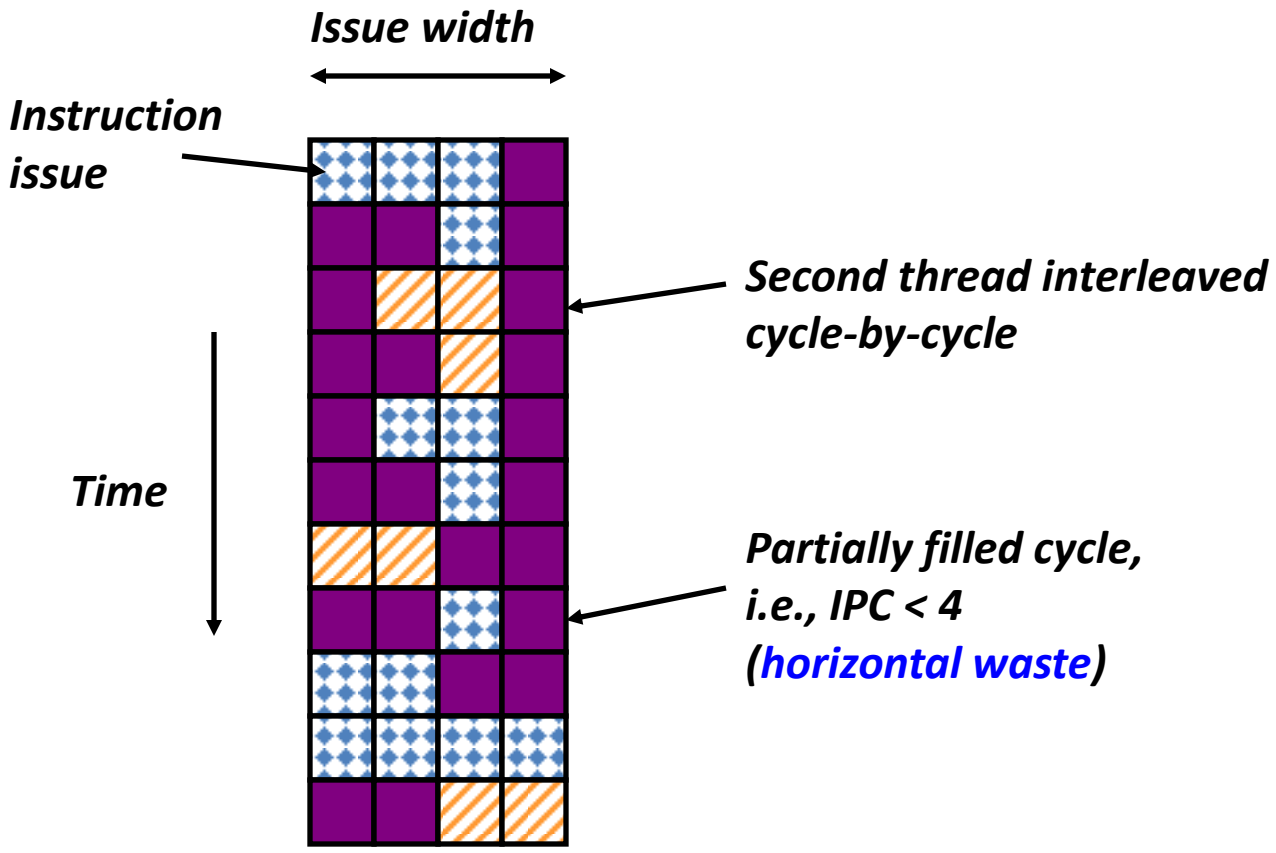
For an 8-way superscalar.

Image From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism", ISCA 1995.

Superscalar Machine Efficiency

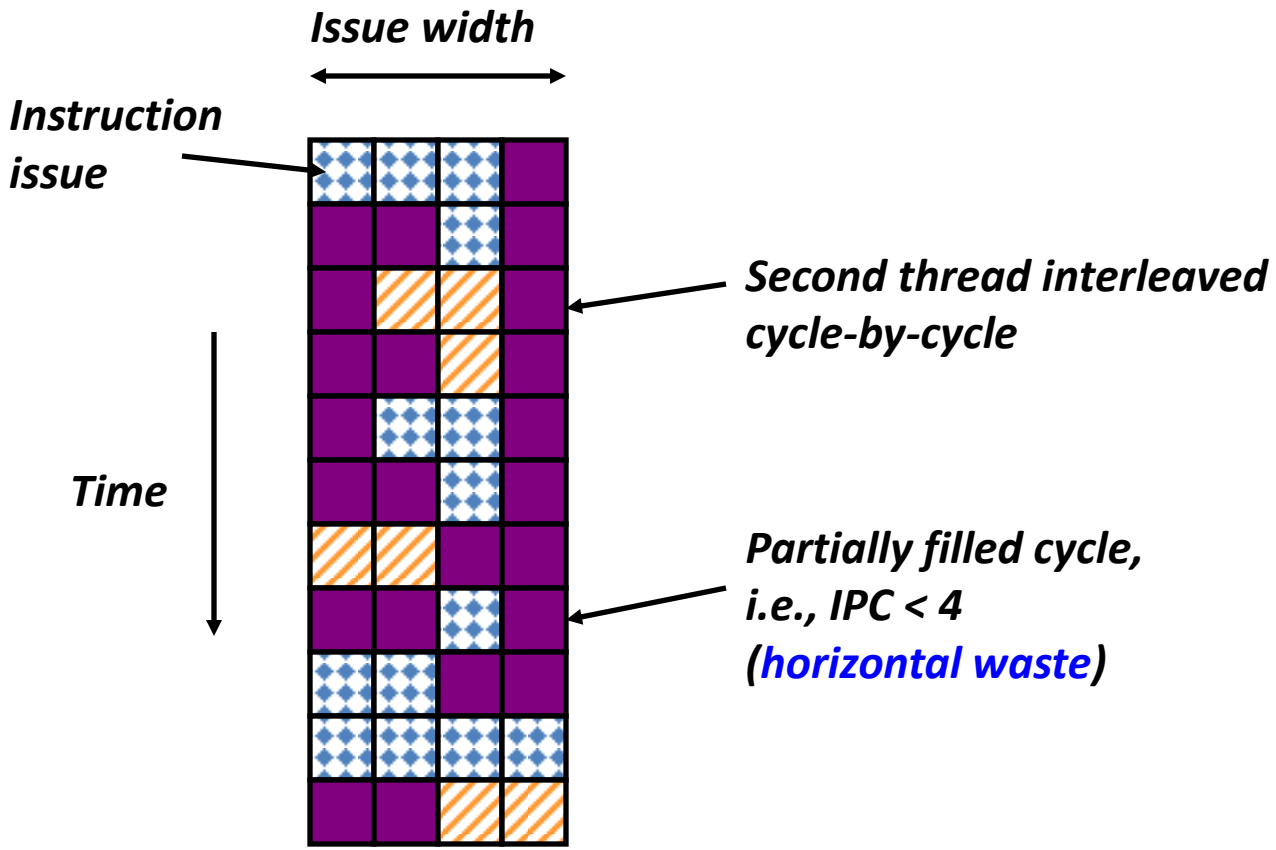


Vertical Multithreading



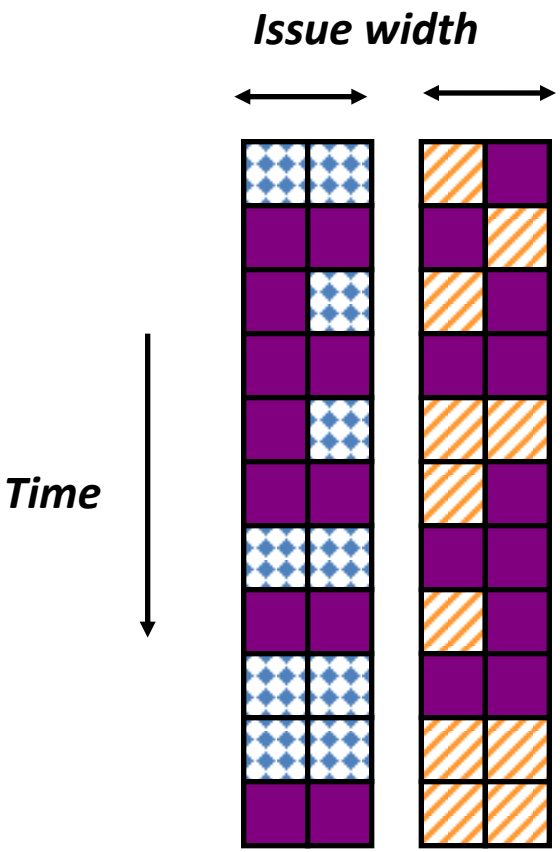
- What is the effect of cycle-by-cycle interleaving?

Vertical Multithreading



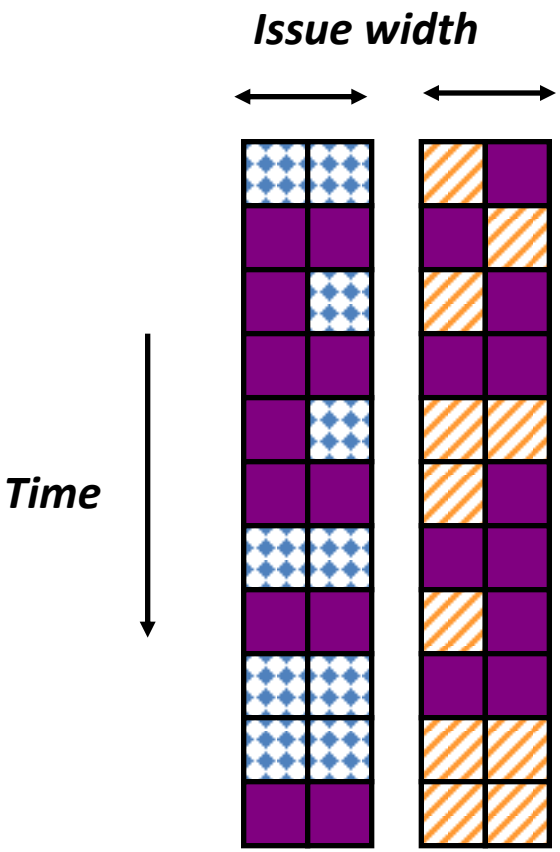
- What is the effect of cycle-by-cycle interleaving?
  - removes vertical waste, but leaves some horizontal waste

Chip Multiprocessing (CMP)



- What is the effect of splitting into multiple processors?

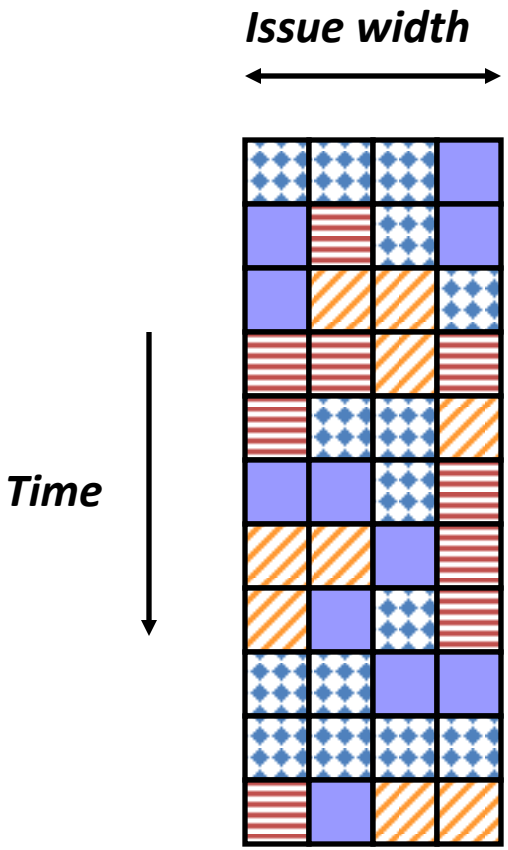
Chip Multiprocessing (CMP)



- What is the effect of splitting into multiple processors?
  - reduces horizontal waste,
  - leaves some vertical waste, and
  - puts upper limit on peak throughput of each thread.

Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]



- Interleave multiple threads to multiple issue slots with no restrictions

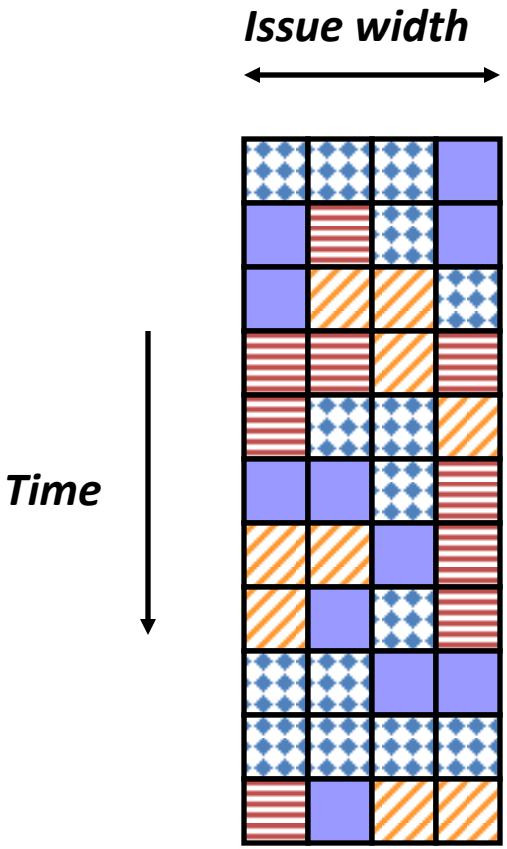
OOO Simultaneous Multithreading

[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

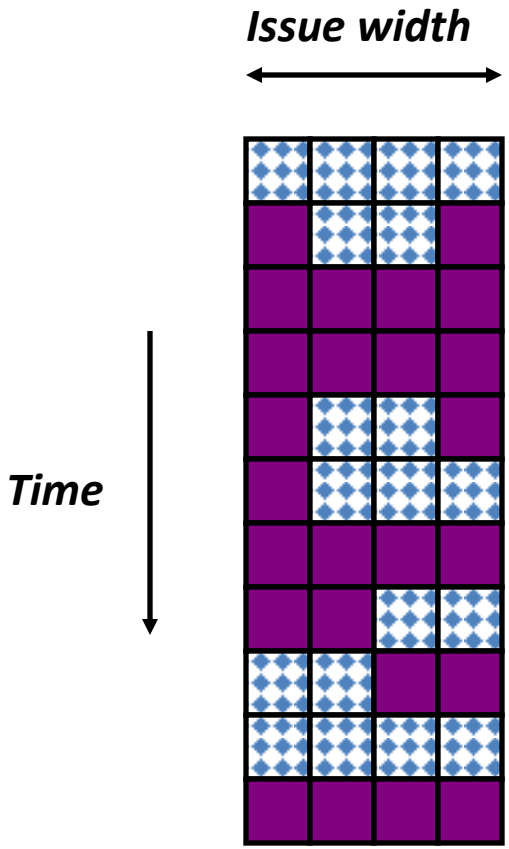
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required to schedule from multiple threads
- Any single thread can utilize whole machine

SMT adaptation to parallelism type

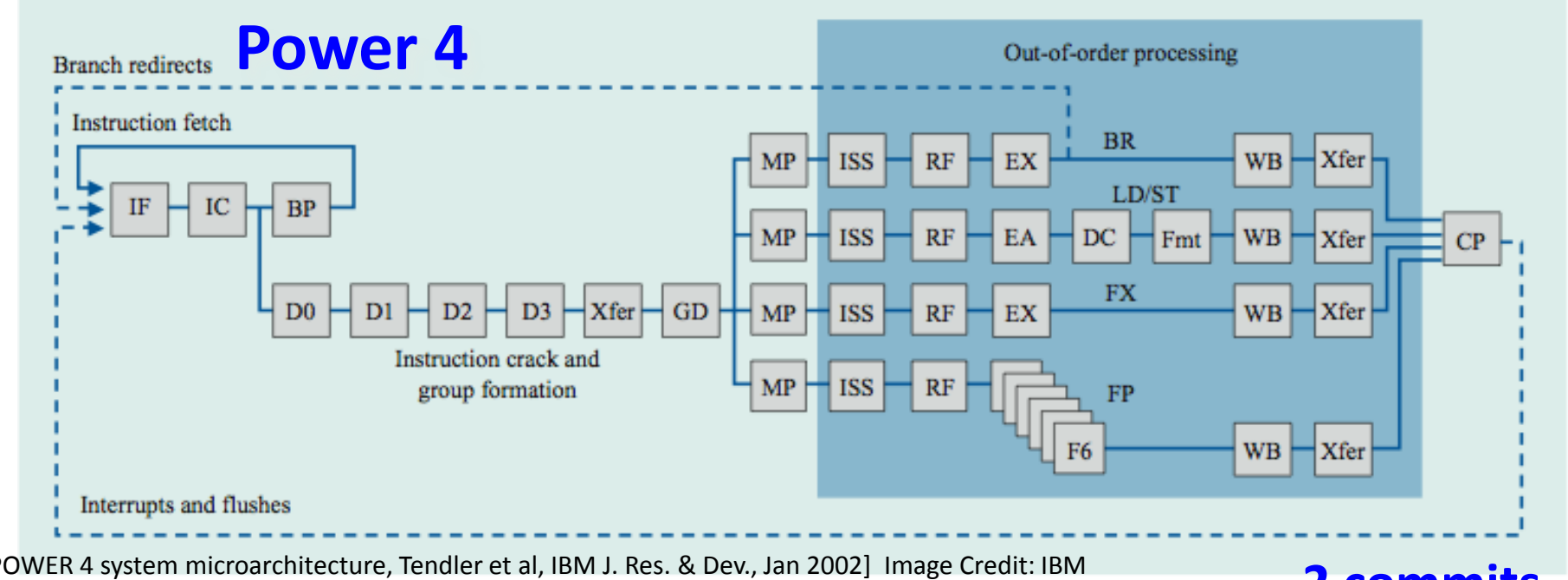
For regions with high thread level parallelism (TLP) entire machine width is shared by all threads



For regions with low thread level parallelism (TLP) entire machine width is available for instruction level parallelism (ILP)



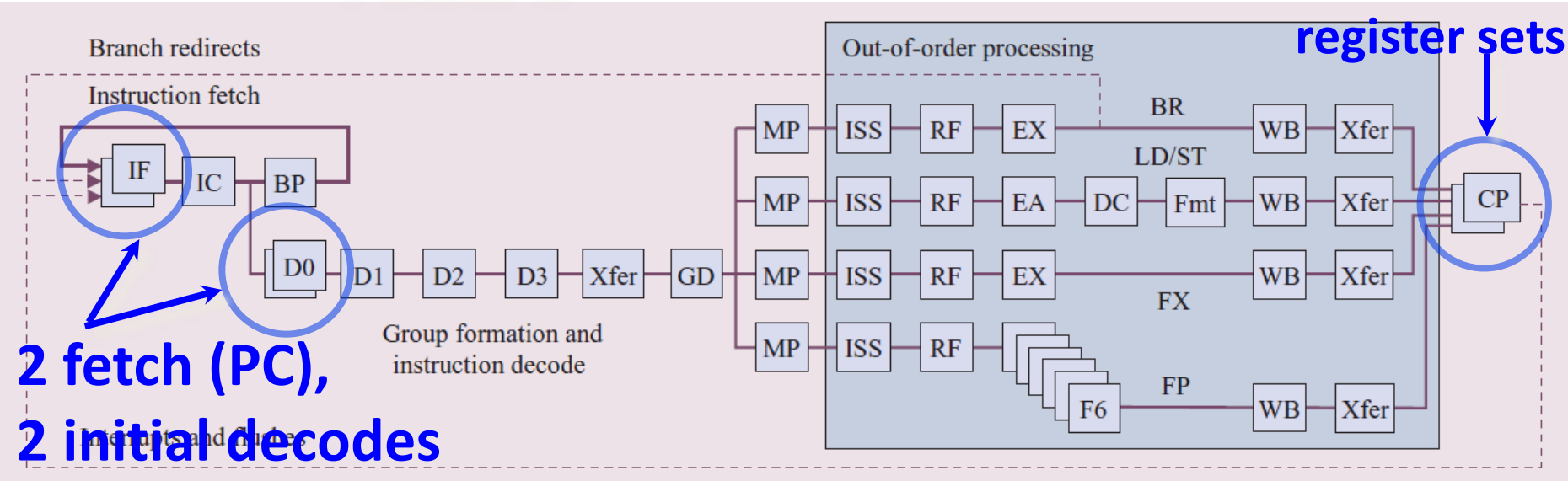
Power 4



[POWER 4 system microarchitecture, Tendler et al, IBM J. Res. & Dev., Jan 2002] Image Credit: IBM
Courtesy of International Business Machines, © International Business Machines.

Power 5

2 commits (architected register sets)



2 fetch (PC),
2 initial decodes

[POWER 5 system microarchitecture, Sinharoy et al, IBM J. Res. & Dev., Jul/Sept 2005] Image Credit: IBM
Courtesy of International Business Machines, © International Business Machines.

Power 5 data flow ..

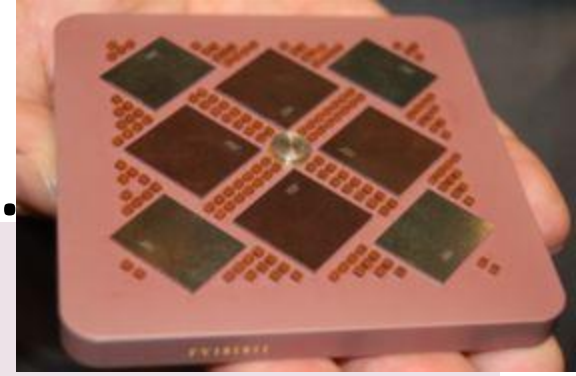
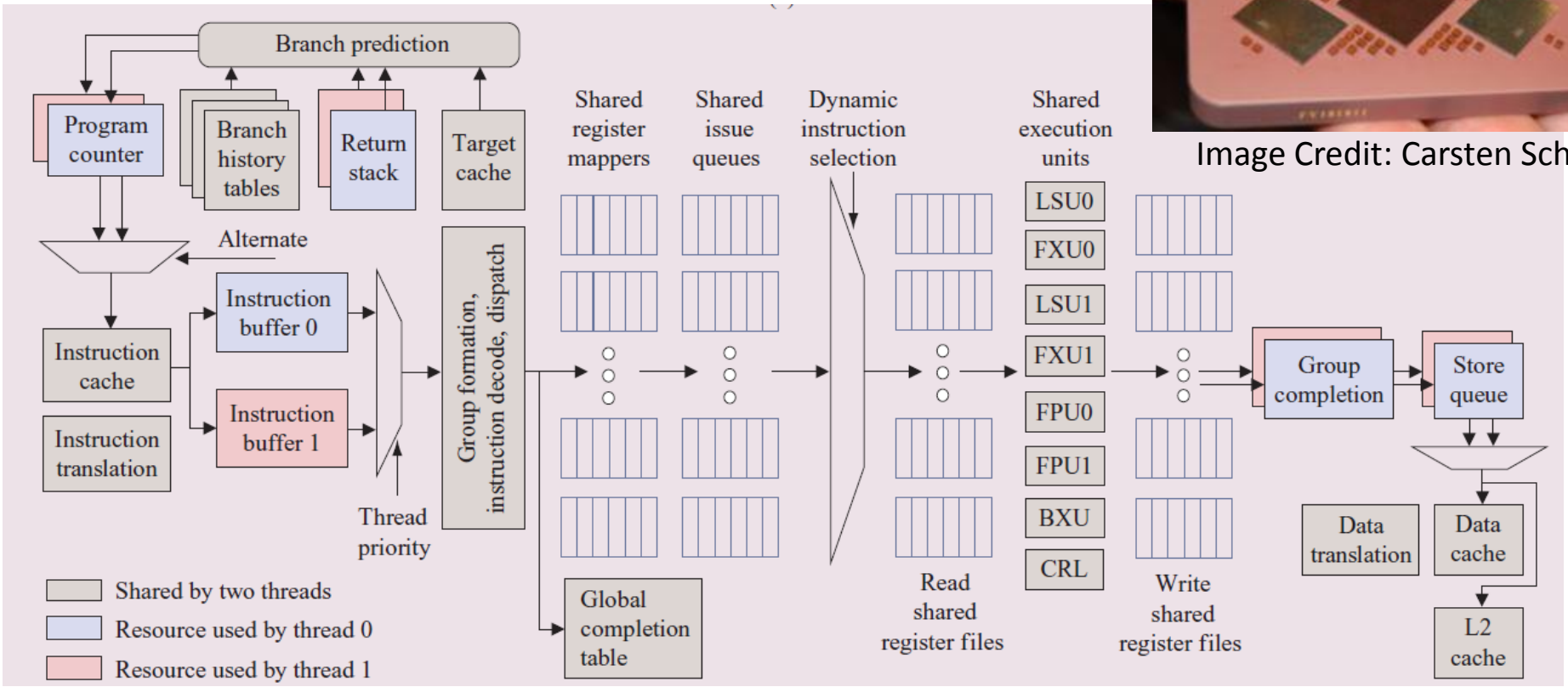


Image Credit: Carsten Schulz



[POWER 5 system microarchitecture, Sinharoy et al, IBM J. Res. & Dev., Jul/Sept 2005] Image Credit: IBM
Courtesy of International Business Machines, © International Business Machines.

Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

Changes in Power 5 to support SMT

- Increased associativity of L1 instruction cache and the instruction address translation buffers
- Added per thread load and store queues
- Increased size of the L2 (1.92 vs. 1.44 MB) and L3 caches
- Added separate instruction prefetch and buffering per thread
- Increased the number of virtual registers from 152 to 240
- Increased the size of several issue queues
- The Power5 core is about 24% larger than the Power4 core because of the addition of SMT support

Pentium-4 Hyperthreading (2002)

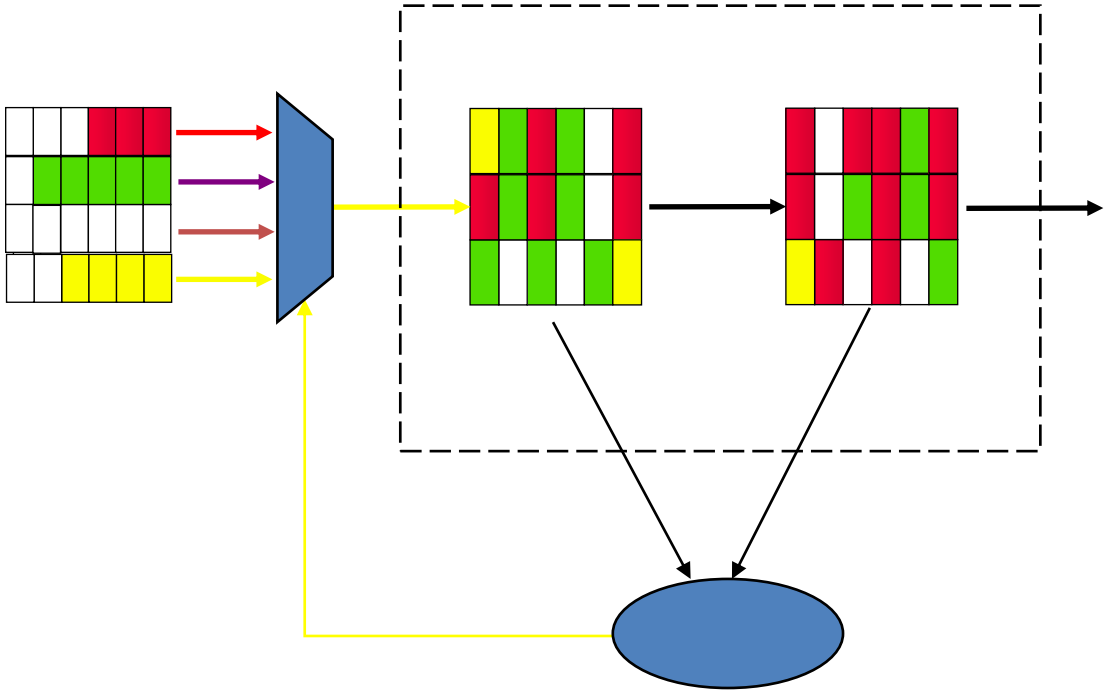
- First commercial SMT design (2-way SMT)
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can use all entries in queues when two threads are active
- Processor running only one active software thread runs at approximately same speed with or without hyperthreading
- Hyperthreading dropped on OOO P6 based follow-ons to Pentium-4 (Pentium-M, Core Duo, Core 2 Duo), until revived with Nehalem generation machines in 2008.
- Intel Atom (in-order x86 core) has two-way vertical multithreading

Initial Performance of SMT

- Pentium 4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on Pentium 4 each of 26 SPEC benchmarks paired with every other (26^2 runs) speed-ups from 0.90 to 1.58; average was 1.20
- Power 5, 8-processor server 1.23 faster for SPECint_rate with SMT, 1.16 faster for SPECfp_rate
- Power 5 running 2 copies of each app speedup between 0.89 and 1.41
  - Most gained some
  - Floating Point apps had most cache conflicts and least gains

Icount Choosing Policy

Fetch from thread with the least instructions in flight.



Why does this enhance throughput?

Summary: Multithreaded Categories

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Christopher Batten (Cornell)

- MIT material derived from course 6.823
- UCB material derived from course CS252 & CS152
- Cornell material derived from course ECE 4750