CS 152/252A Computer Architecture and Engineering

Sophia Shao

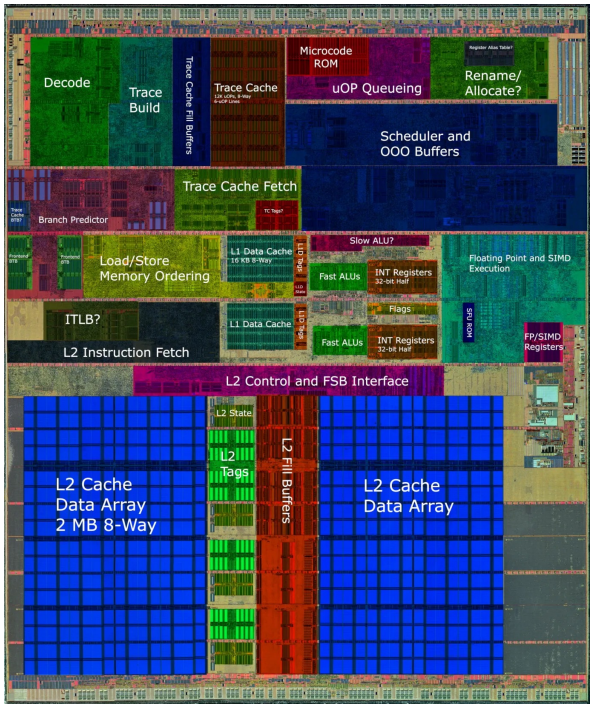
Lecture 16: Multithreading

Intel's Netburst: Failure is a Foundation for Success

In the world of today's high performance CPUs, major architectural changes don't happen often. Iterating off a proven base is safer, cheaper, and faster than attempting to massively rework the basics of how a CPU fetches and executes instructions. But more than 20 years ago, things hadn't settled down yet. Intel made two attempts to replace its solid but aging P6 microarchitecture with something completely different.

One was Itanium, which avoided the complexity associated with out-of-order execution and variable length decode to deliver very wide in-order execution.

Pentium 4 was the other, and …. Its microarchitecture, called Netburst, targeted very high clock speeds using a long pipeline. Alongside this key feature, it brought a wide range of innovative architectural features. As we all know, it didn't quite pan out the way Intel would have liked.



https://chipsandcheese.com/2022/06/17/intels-netburst-failure-is-a-foundation-for-success/

Last Time Lecture: VLIW

- In a classic VLIW, compiler is responsible for avoiding all hazards -> simple hardware, complex compiler.
- Later VLIWs added more dynamic hardware interlocks, which reduce relative hardware benefits
- Use loop unrolling and software pipelining for loops, trace scheduling for more irregular code
- Static scheduling difficult in presence of unpredictable branches and variable latency memory
- VLIW has failed in general-purpose computing, but still used in deeply embedded processors and DSPs

Last Time in Lecture: Branch Prediction

- Branch prediction
  - temporal, history of a single branch
  - spatial, based on path through multiple branches
- Branch History Table (BHT) vs. Branch Target Buffer (BTB)
  - tradeoff in capacity versus latency
- Advanced branch prediction structures
  - Perceptron
  - TAGE

Thread-Level Parallelism (TLP)

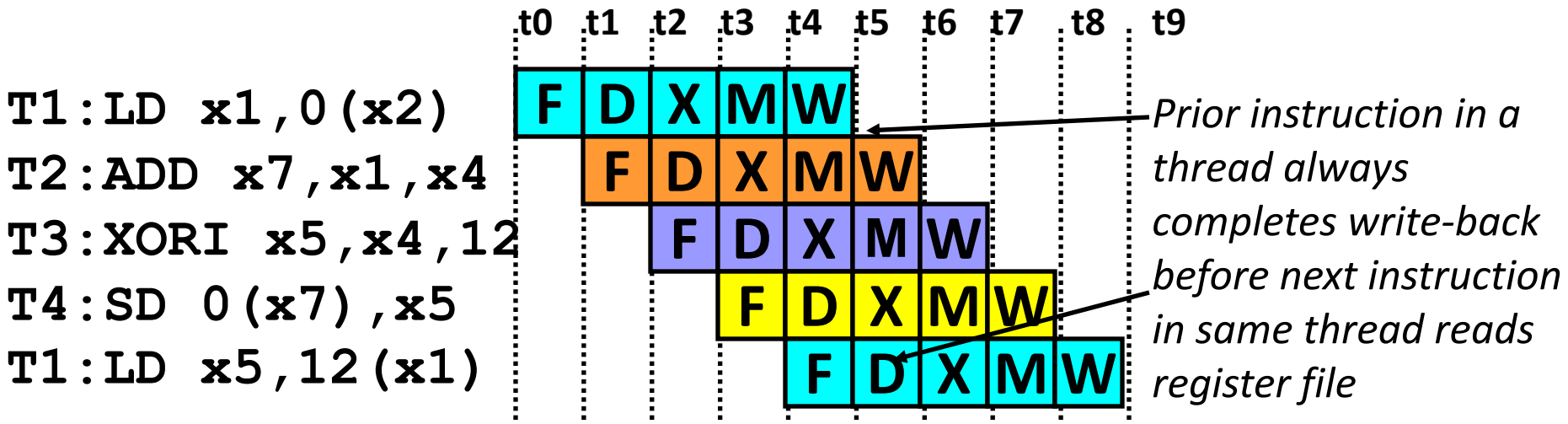
- Difficult to continue to extract instruction-level parallelism (ILP) from a single sequential thread of control
- Many workloads can make use of thread-level parallelism:
  - TLP from ***multiprogramming*** (run independent sequential jobs)
  - TLP from ***multithreaded*** applications (run one job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

Multithreading

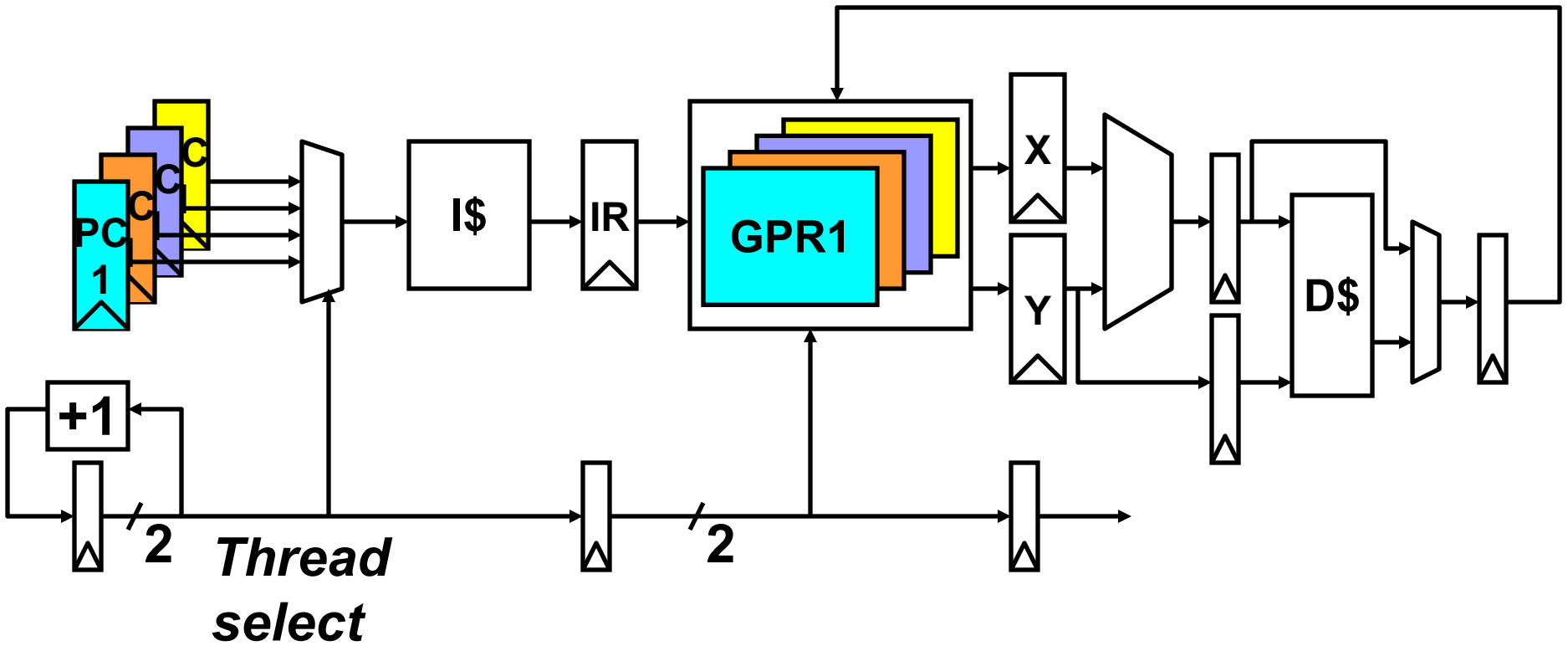
How can we guarantee no dependencies between instructions in a pipeline?

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*



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

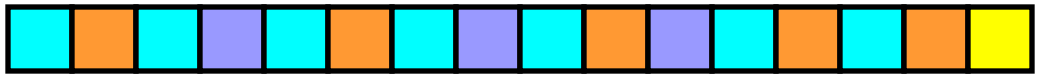
Multithreading Costs

- 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 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

- 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

CDC 6600 Peripheral Processors
(Cray, 1964)

- First multithreaded hardware
- 10 “virtual” I/O processors
- Fixed interleave on simple pipeline
- Pipeline has 100ns cycle time
- Each virtual processor executes one instruction every 1000ns
- Accumulator-based instruction set to reduce processor state

Denelcor Heterogeneous Element Processor (HEP)

(Burton Smith, 1982)

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 (Multithreaded Architecture)
- Acquired Cray

Coarse-Grain Multithreading

- Tera MTA designed for supercomputing applications with large data sets and low locality
  - No data cache
  - Many parallel threads needed to hide large memory latency
- Other applications are more cache friendly
  - Few pipeline bubbles if cache mostly has hits
  - Just add a few threads to hide occasional cache miss latencies
  - Swap threads on cache misses

IBM PowerPC RS64-IV (2000)

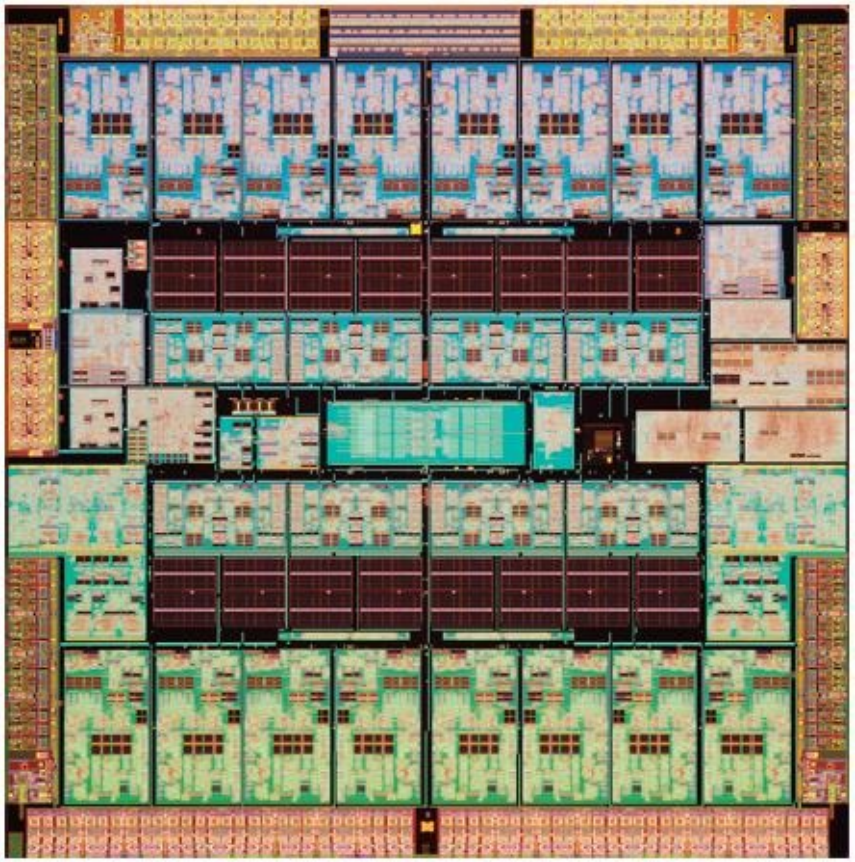
- Commercial coarse-grain multithreading CPU
- Based on PowerPC with quad-issue in-order five-stage pipeline
- Each physical CPU supports two virtual CPUs
- On L2 cache miss, pipeline is flushed and execution switches to second thread
  - short pipeline minimizes flush penalty (4 cycles), small compared to memory access latency
  - flush pipeline to simplify exception handling

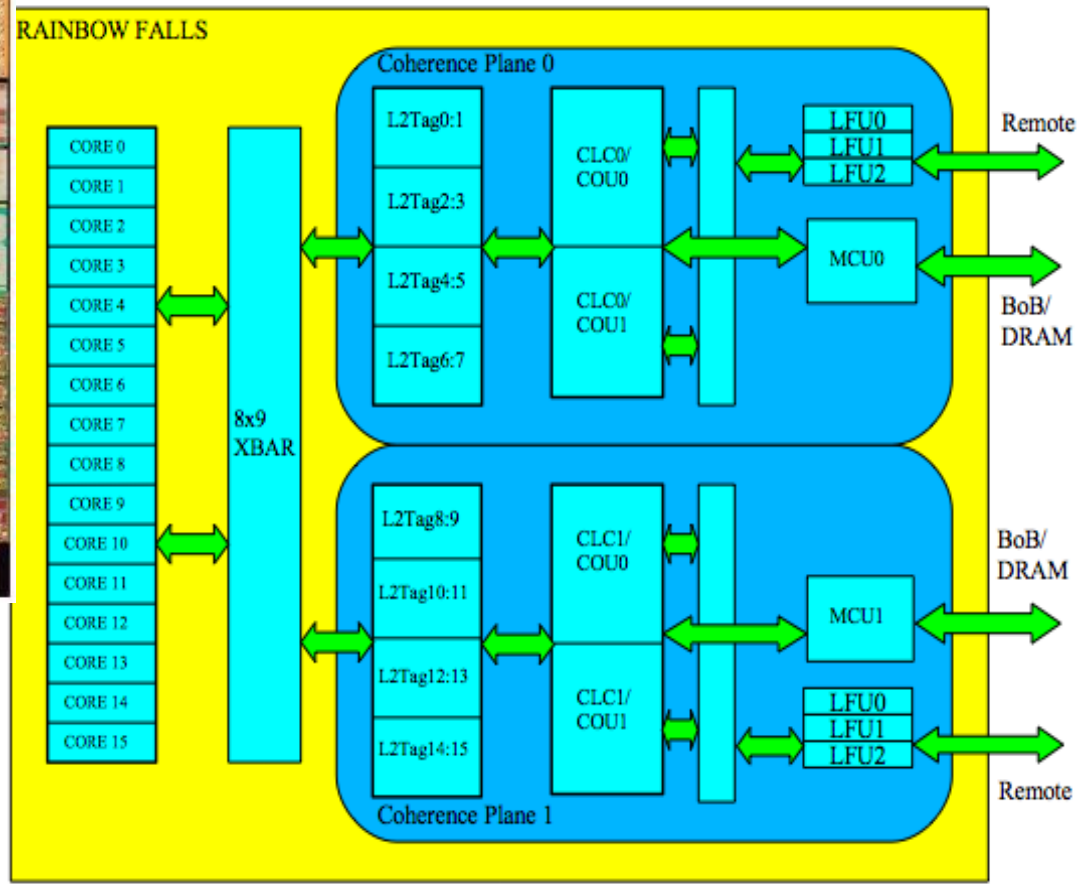
Oracle/Sun Niagara processors

- 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
- T4 [2011], 8 cores, 8 threads/core
- T5 [2012], 16 cores, 8 threads/core
- M5 [2012], 6 cores, 8 threads/core
- M6 [2013], 12 cores, 8 threads/core

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

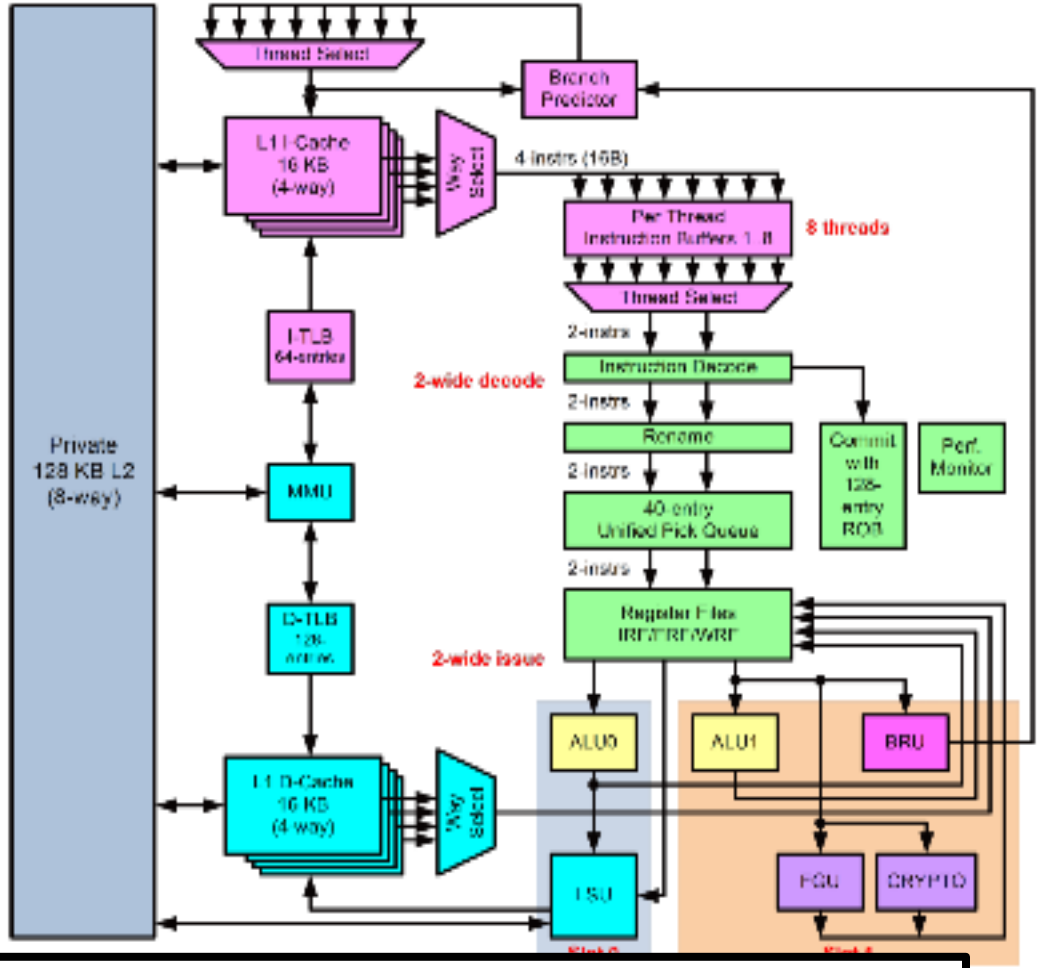




Oracle SPARC M6 Core (2013)

SPARC S3 Core

- Dual-issue, out-of-order
- Integrated encryption acceleration instructions
- Enhanced instruction set to accelerate Oracle SW stack
- 1-8 strands, dynamically threaded pipeline



Oracle ended SPARC programs after M8 in 2017

CS152 Administrivia

- HW 3 due 03/14
- Lab 3 due 03/23
- Join our really fun discussions:
  - Prashanth, Wednesday 3-5
  - Abe/Edwin, Thursday 2-4
  - Allison/Divija, Thursday 5-7
  - Animesh/Jamie, Friday 12-2

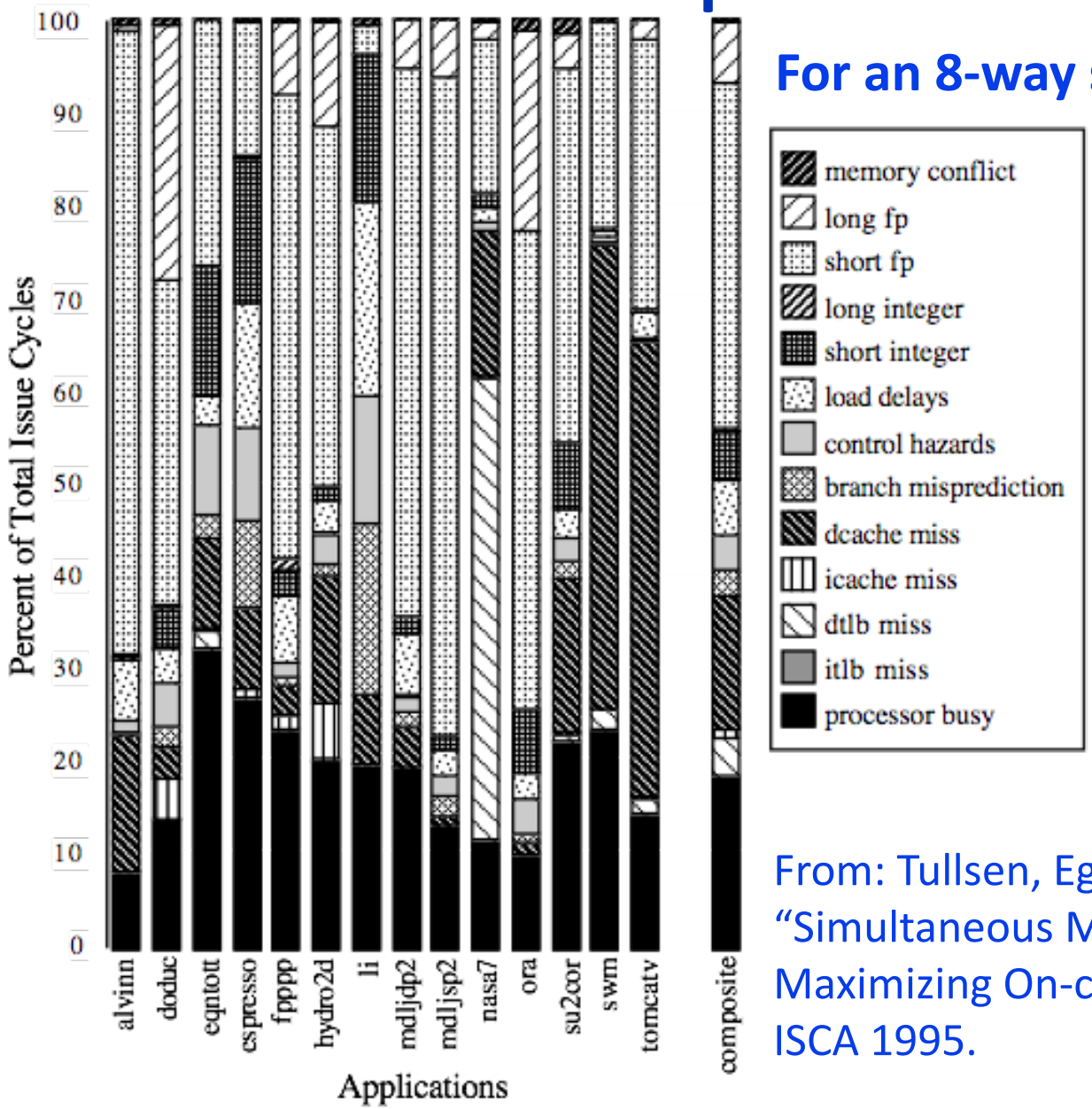
CS252 Administrivia

- Readings on OoO architectures next week

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.

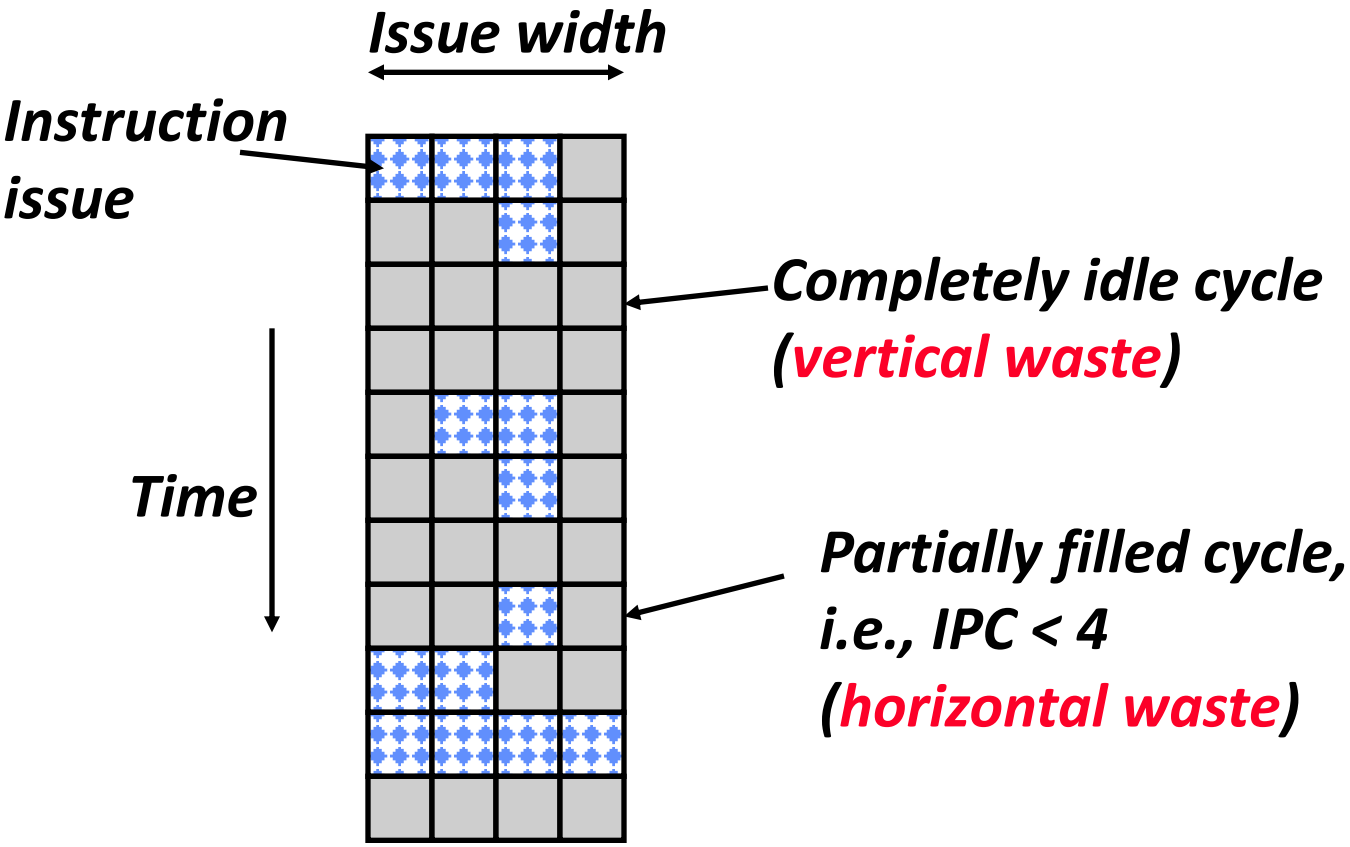
For most apps, most execution units lie idle in an OoO superscalar



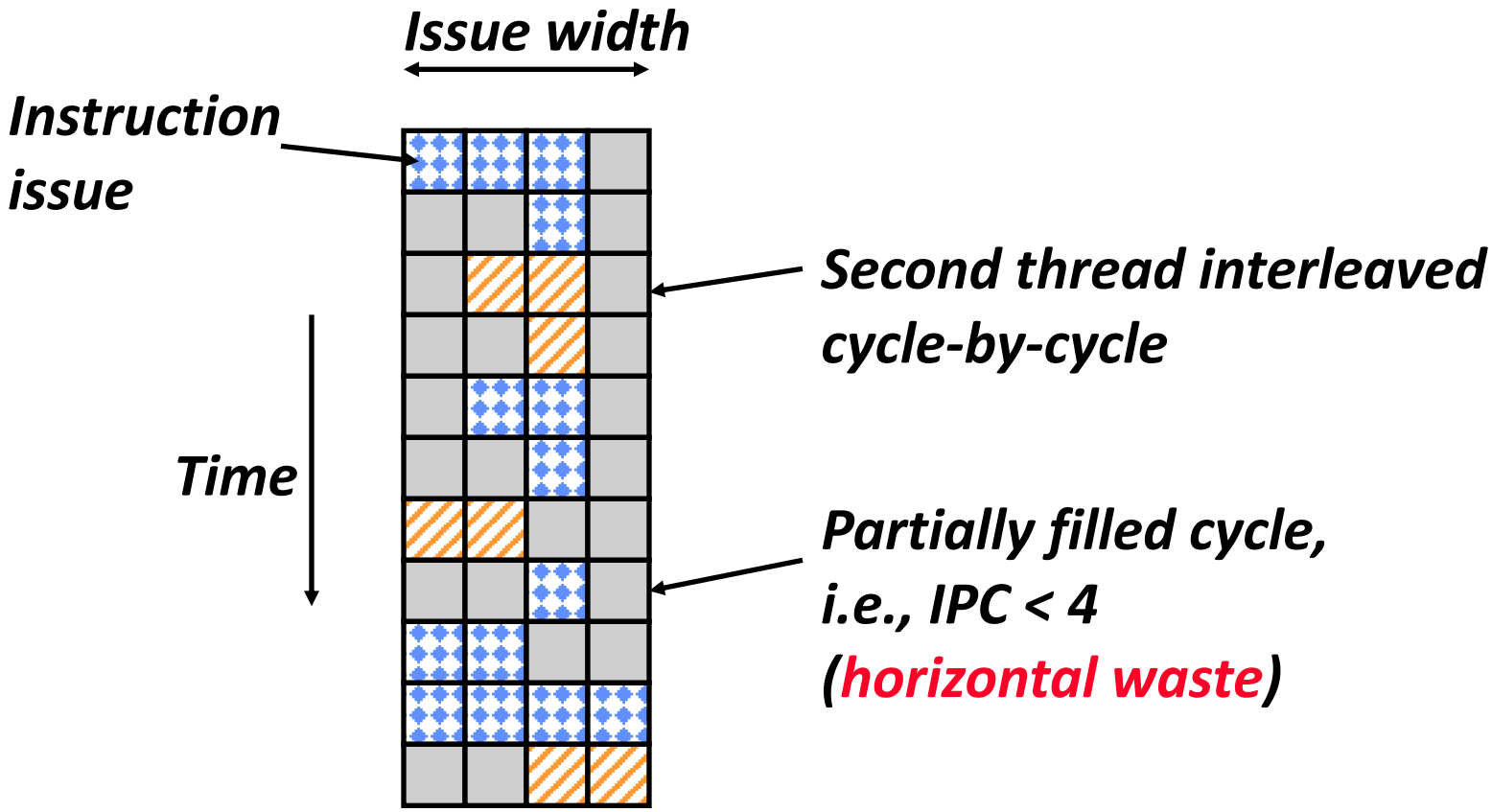
For an 8-way superscalar.

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

Superscalar Machine Efficiency

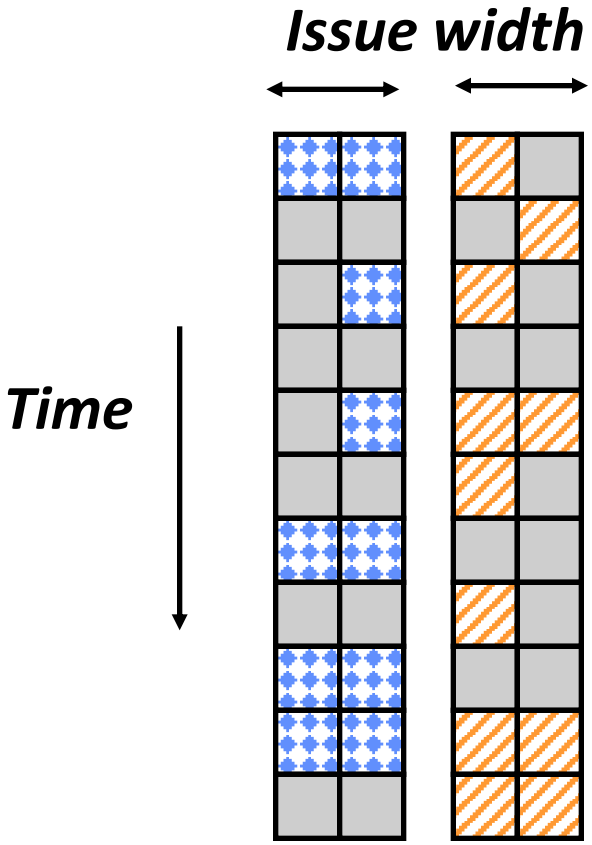


Vertical Multithreading



- Cycle-by-cycle interleaving removes vertical waste, but leaves some horizontal waste

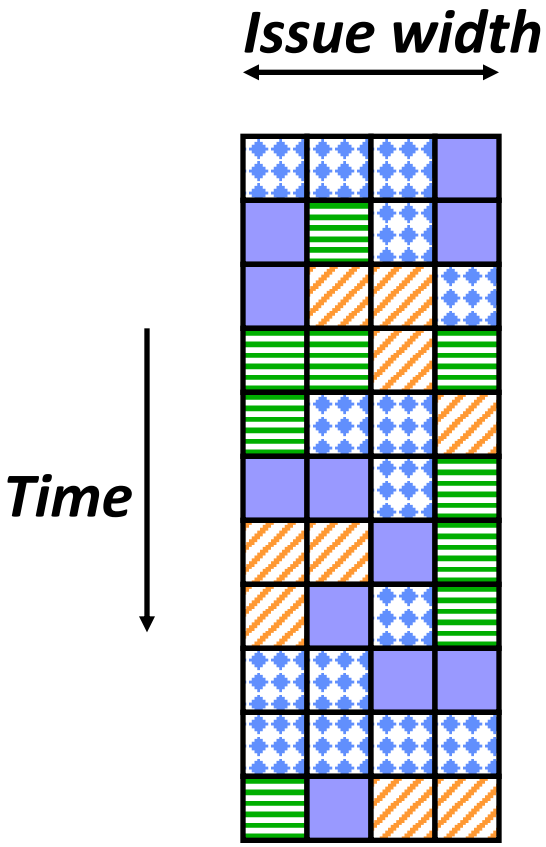
Chip Multiprocessing (CMP)



- 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

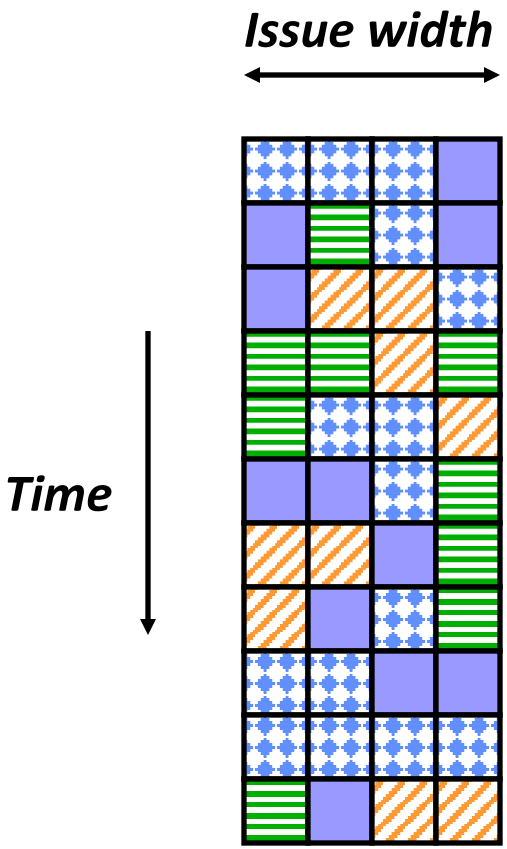
O-o-O Simultaneous Multithreading

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

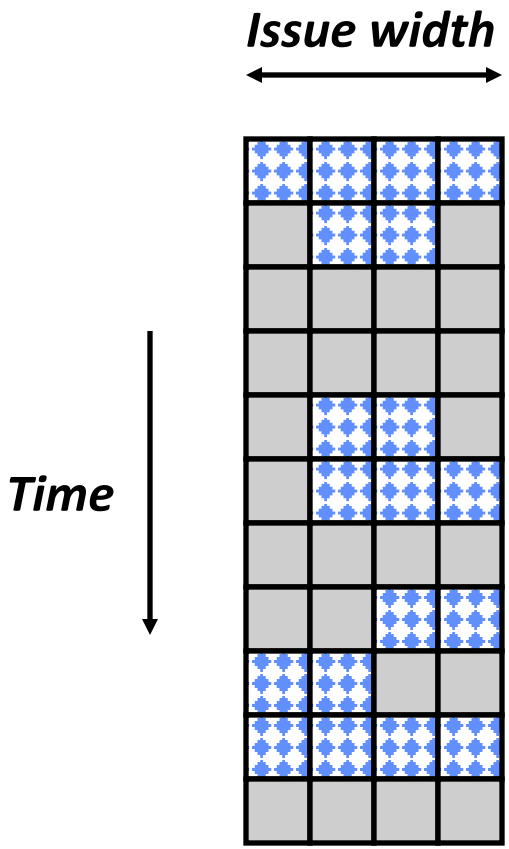
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required 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)

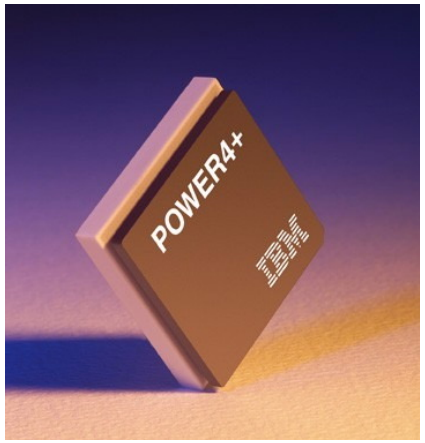


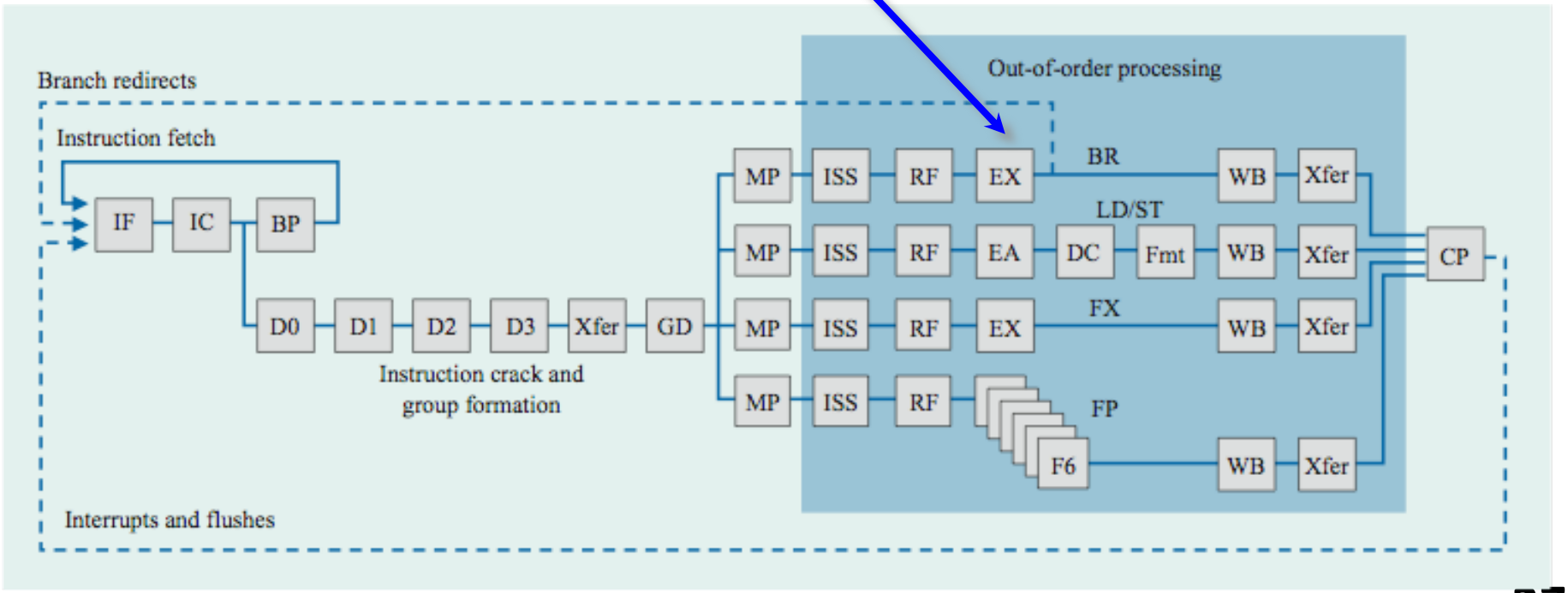
Pentium-4 Hyperthreading (2002)

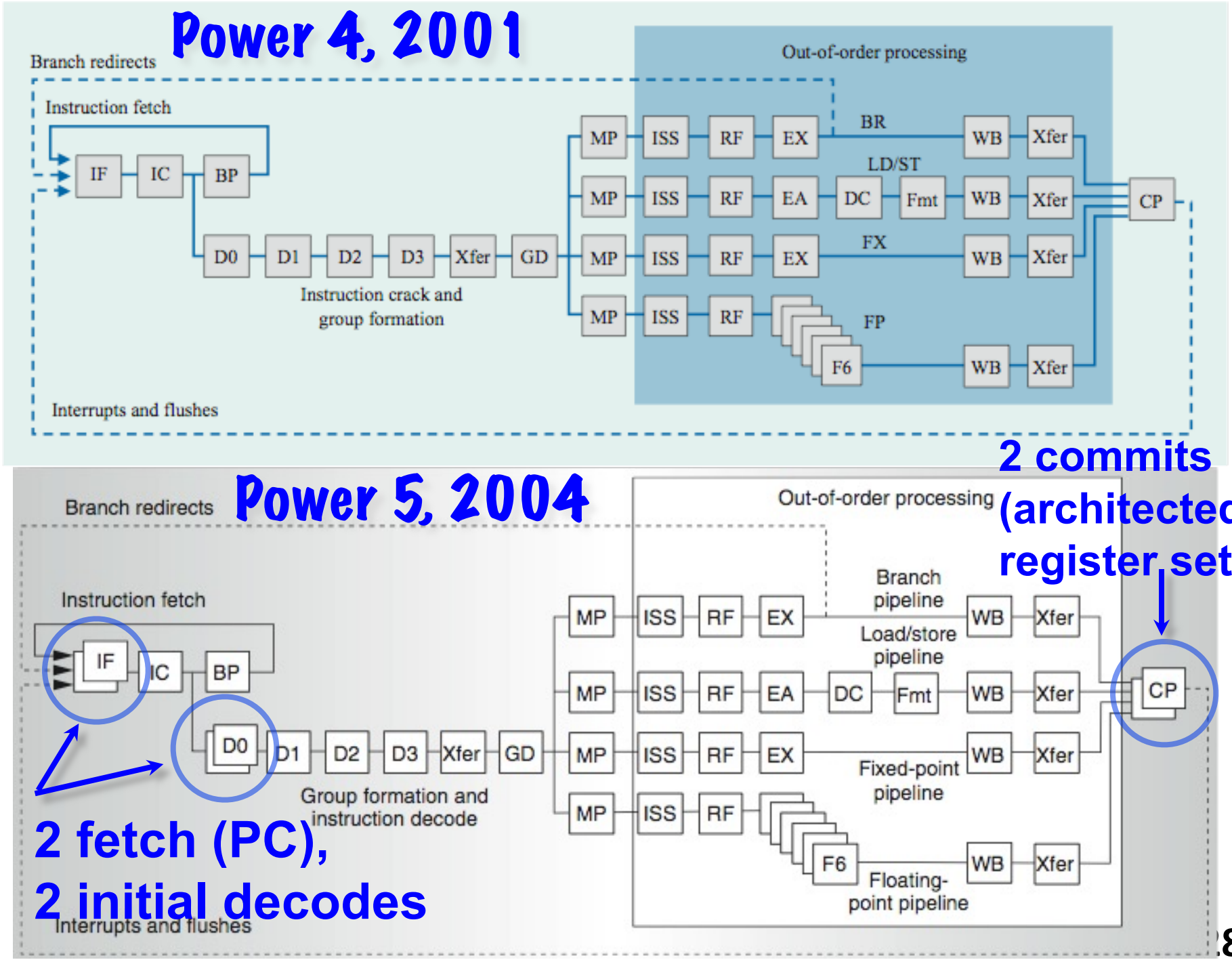
- First commercial SMT design (2-way SMT)
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can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

IBM Power 4, 2001

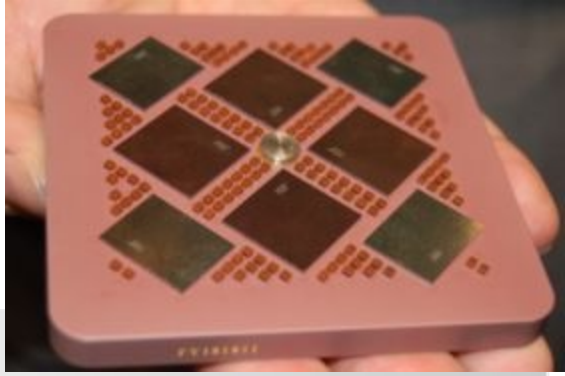
Single-threaded predecessor to Power 5.
8 execution units in out-of-order engine,
each may issue an instruction each cycle.

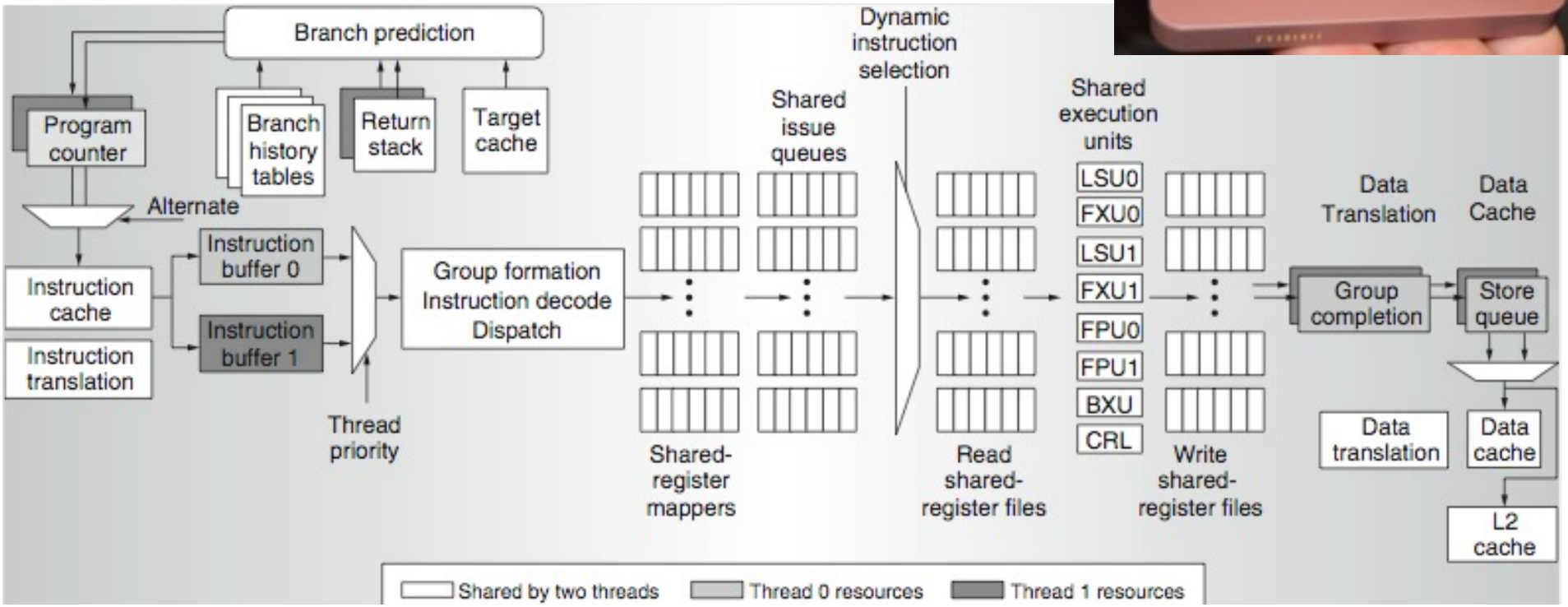





Power 4, 2001
Branch redirects
Out-of-order processing
Instruction fetch
IF
IC
BP
D0
D1
D2
D3
Xfer
GD
Instruction crack and group formation
MP
ISS
RF
EX
BR
WB
Xfer
LD/ST
EA
DC
Fmt
CP
FX
FP
F6
Interrupts and flushes
Power 5, 2004
Branch redirects
Out-of-order processing
Instruction fetch
Branch pipeline
Load/store pipeline
Fixed-point pipeline
Floating-point pipeline
Group formation and instruction decode
2 commits (architected register sets)
2 fetch (PC), 2 initial decodes
Interrupts and flushes

Power 5 data flow ...



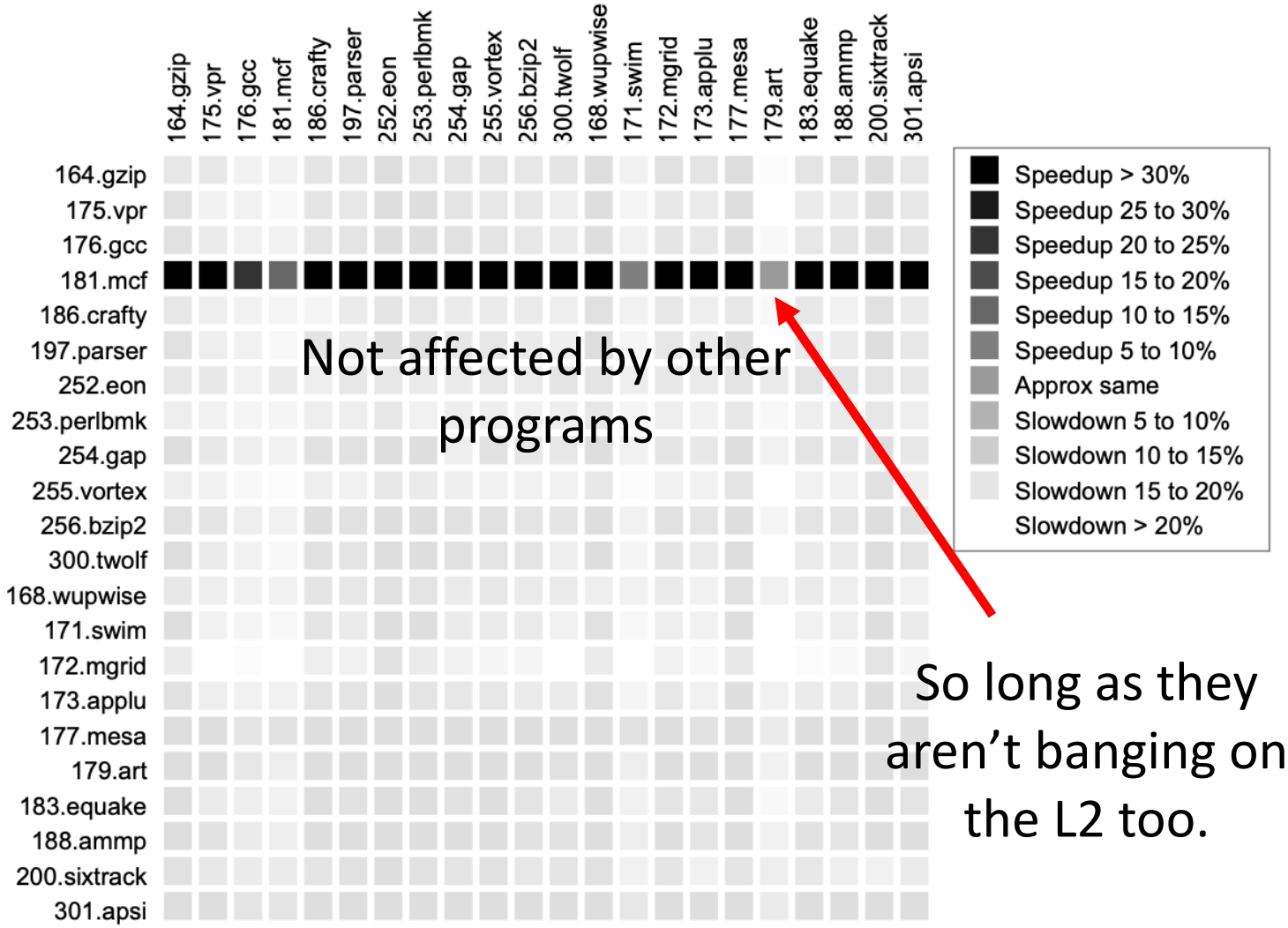


Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

Initial Performance of SMT

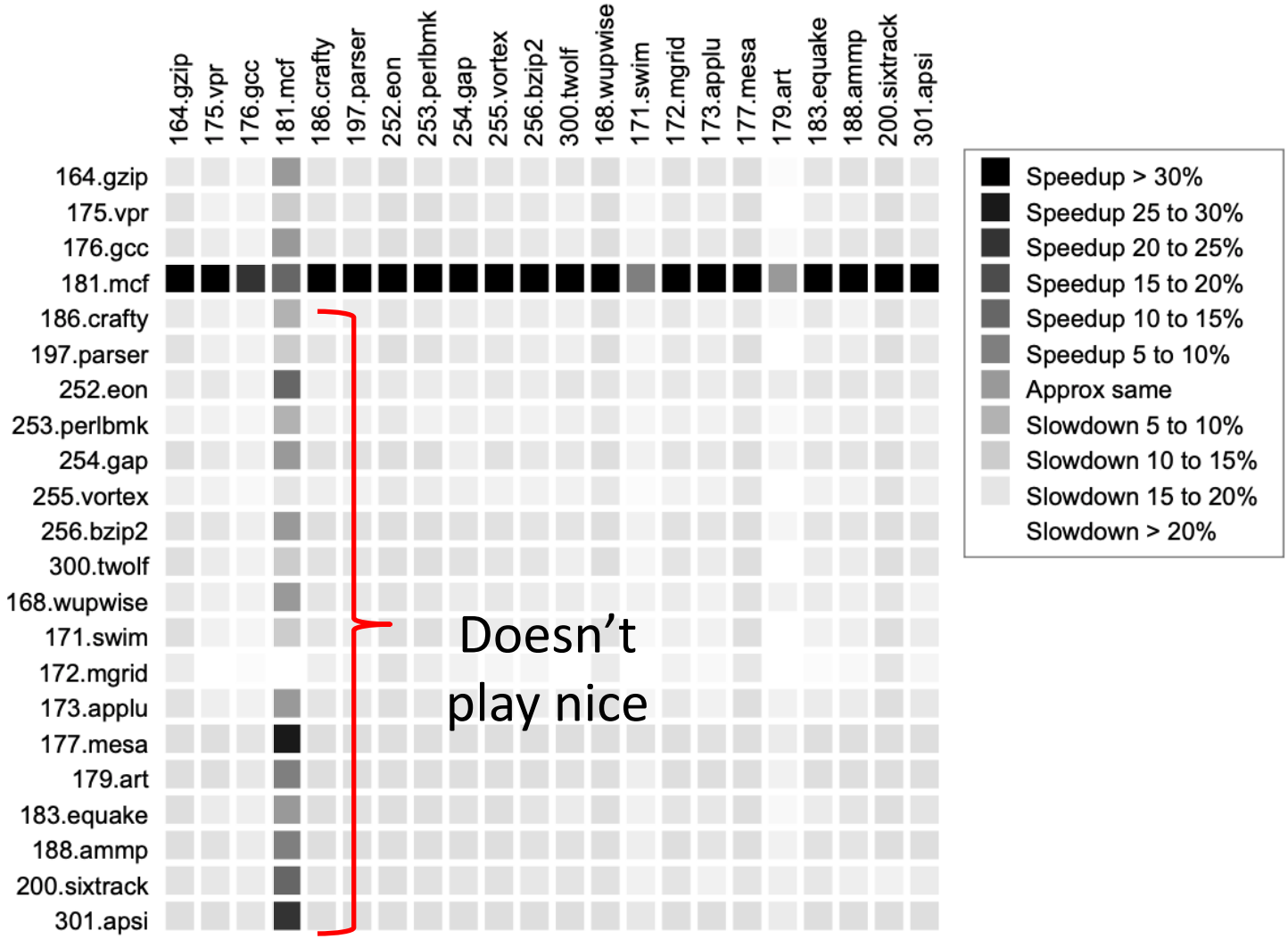
- Pentium-4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium-4 is dual-threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on Pentium-4 each of 26 SPEC benchmarks paired 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
  - Fl.Pt. apps had most cache conflicts and least gains

SMT Performance: Application Interaction



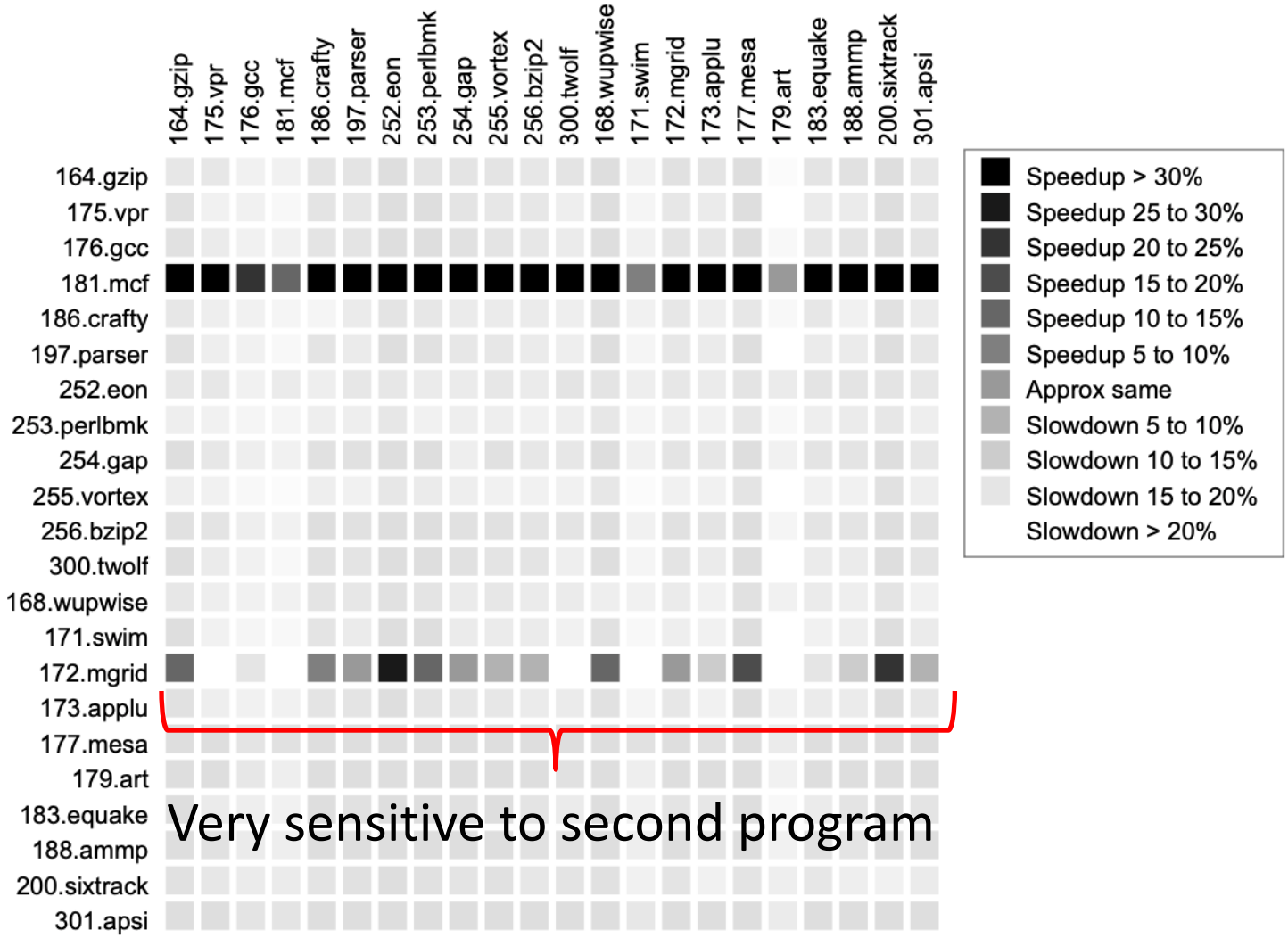
Bulpin et al, "*Multiprogramming Performance of Pentium 4 with Hyper-Threading*"

SMT Performance: Application Interaction



Bulpin et al, "*Multiprogramming Performance of Pentium 4 with Hyper-Threading*"

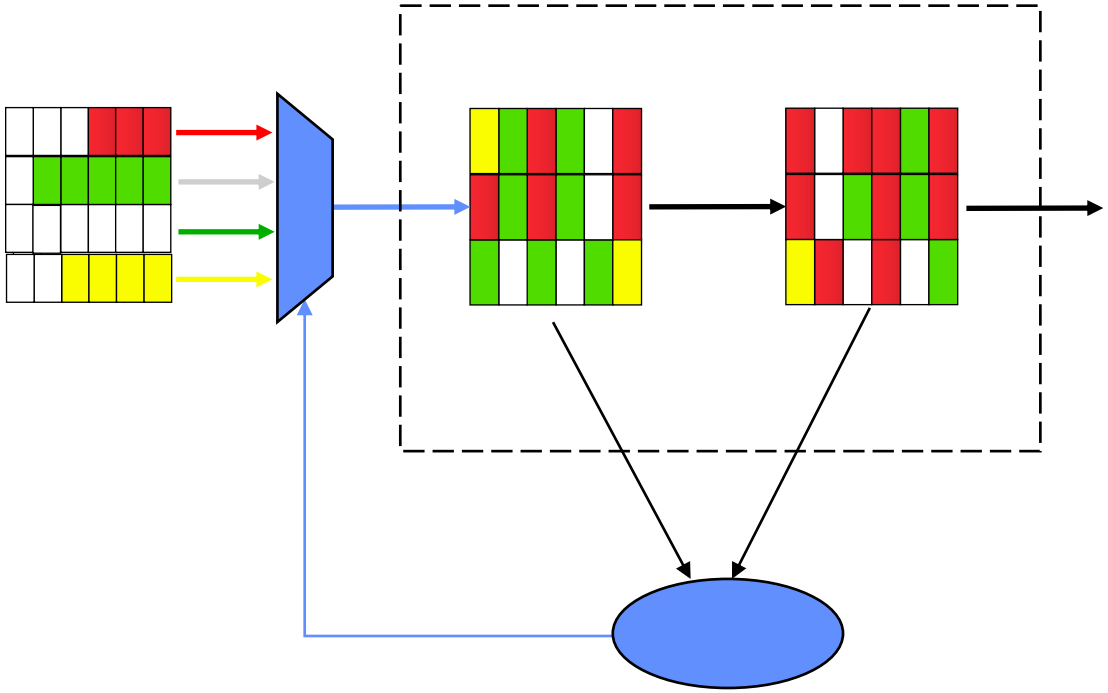
SMT Performance: Application Interaction



Bulpin et al, "*Multiprogramming Performance of Pentium 4 with Hyper-Threading*"

Icount Choosing Policy

Fetch from thread with the least instructions in flight.



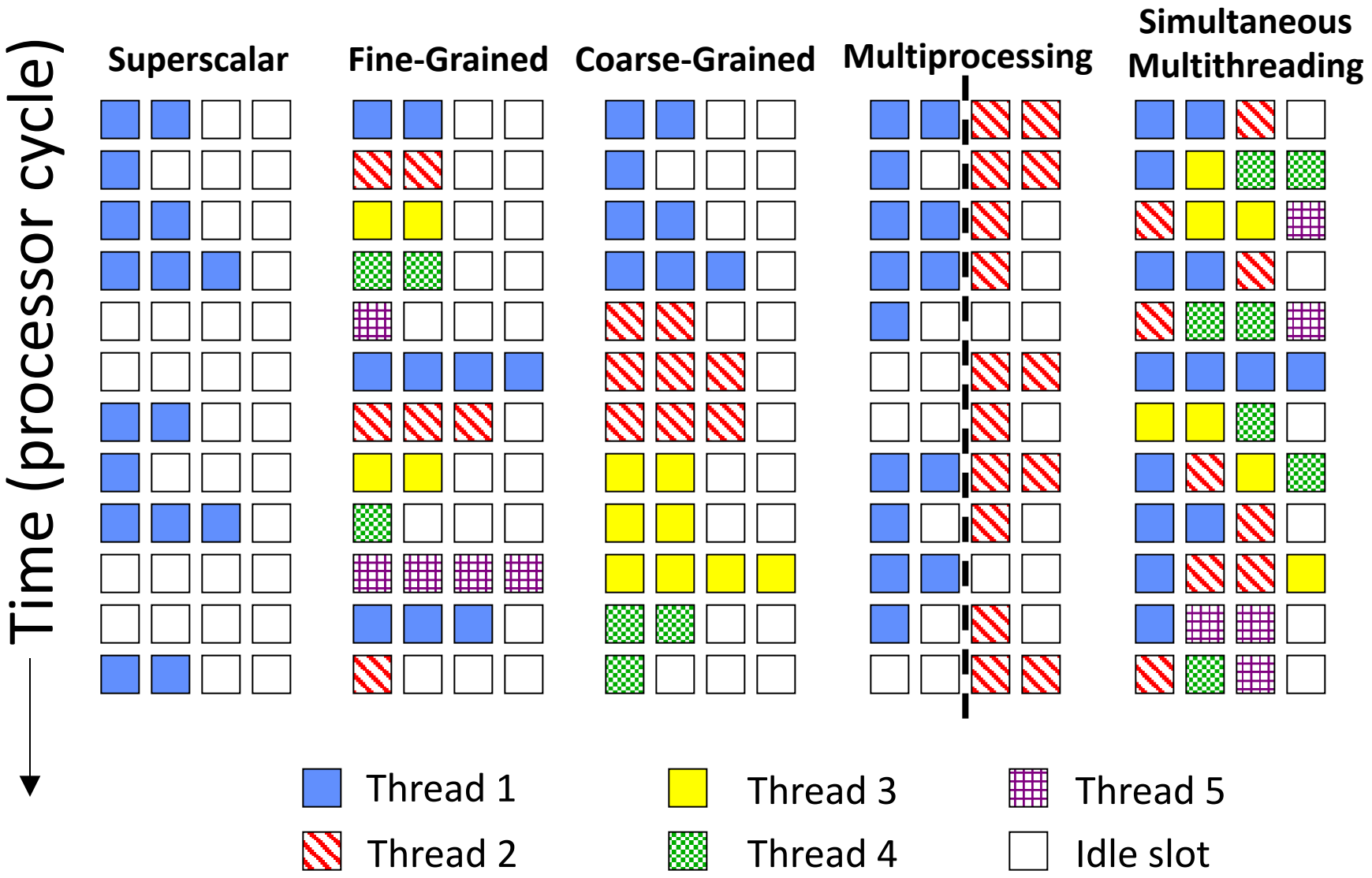
Why does this enhance throughput?

SMT & Security

- Most hardware attacks rely on shared hardware resources to establish a side-channel
  - Eg. Shared outer caches, DRAM row buffers
- SMT gives attackers high-BW access to previously private hardware resources that are shared by co-resident threads:
- TLBs: TLBleed (June, ‘18)
- L1 caches: CacheBleed (2016)
- Functional unit ports: PortSmash (Nov, ’18)

OpenBSD 6.4 → Disabled HT in BIOS, AMD SMT to follow

Summary: Multithreaded Categories



Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)