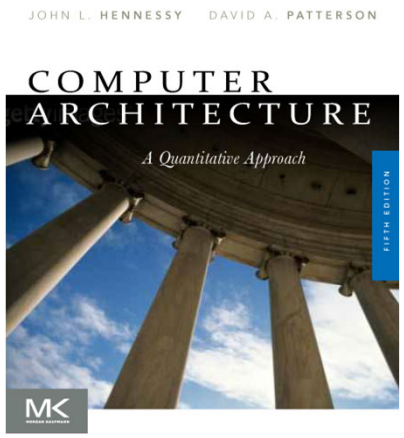


ESE 545 Computer Architecture

Thread-level Parallelism (TLP) and Multithreaded Processors

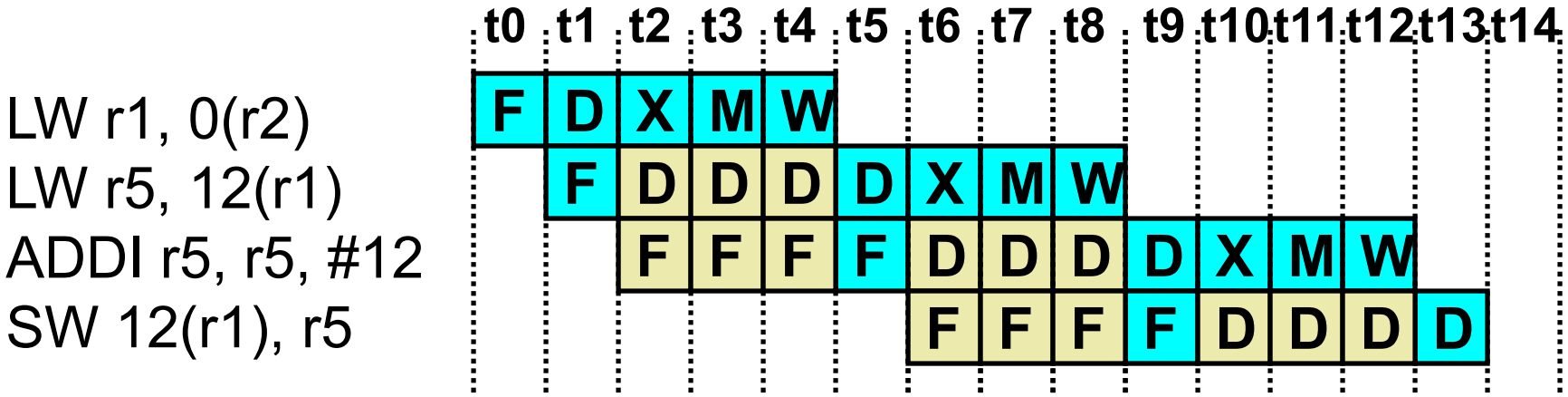
Thread Level Parallelism

- 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)
  - 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
  - Goal: Increase throughput, not latency

Parallel Threads

- **Thread**: process with own instructions and data
  - thread may be a process part of a parallel program of multiple processes, or it may be an independent program
  - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute

Pipeline Hazards



- Each instruction may depend on the next

What is usually done to cope with this?

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

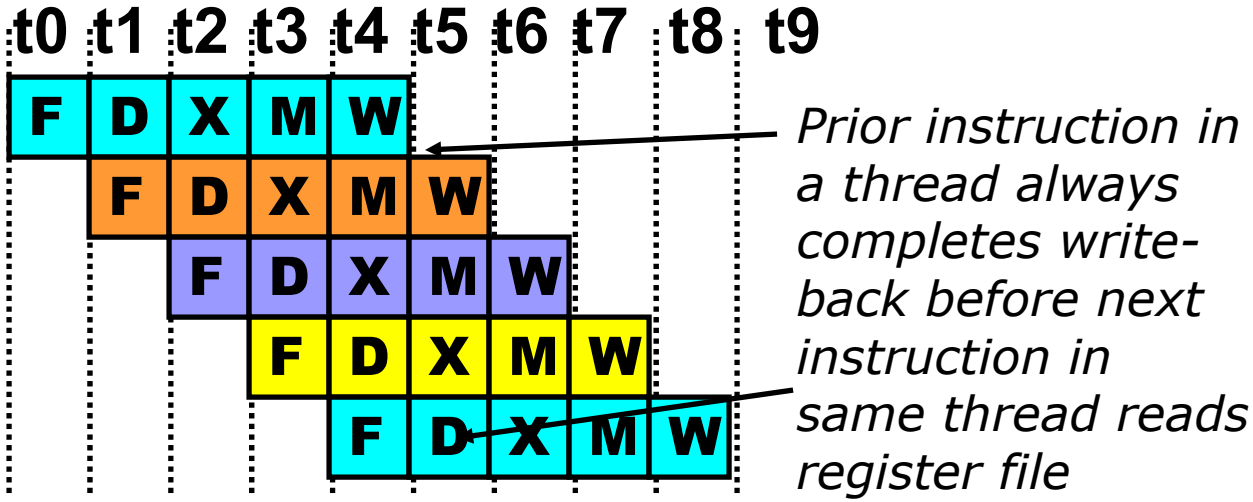
Multithreading

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)

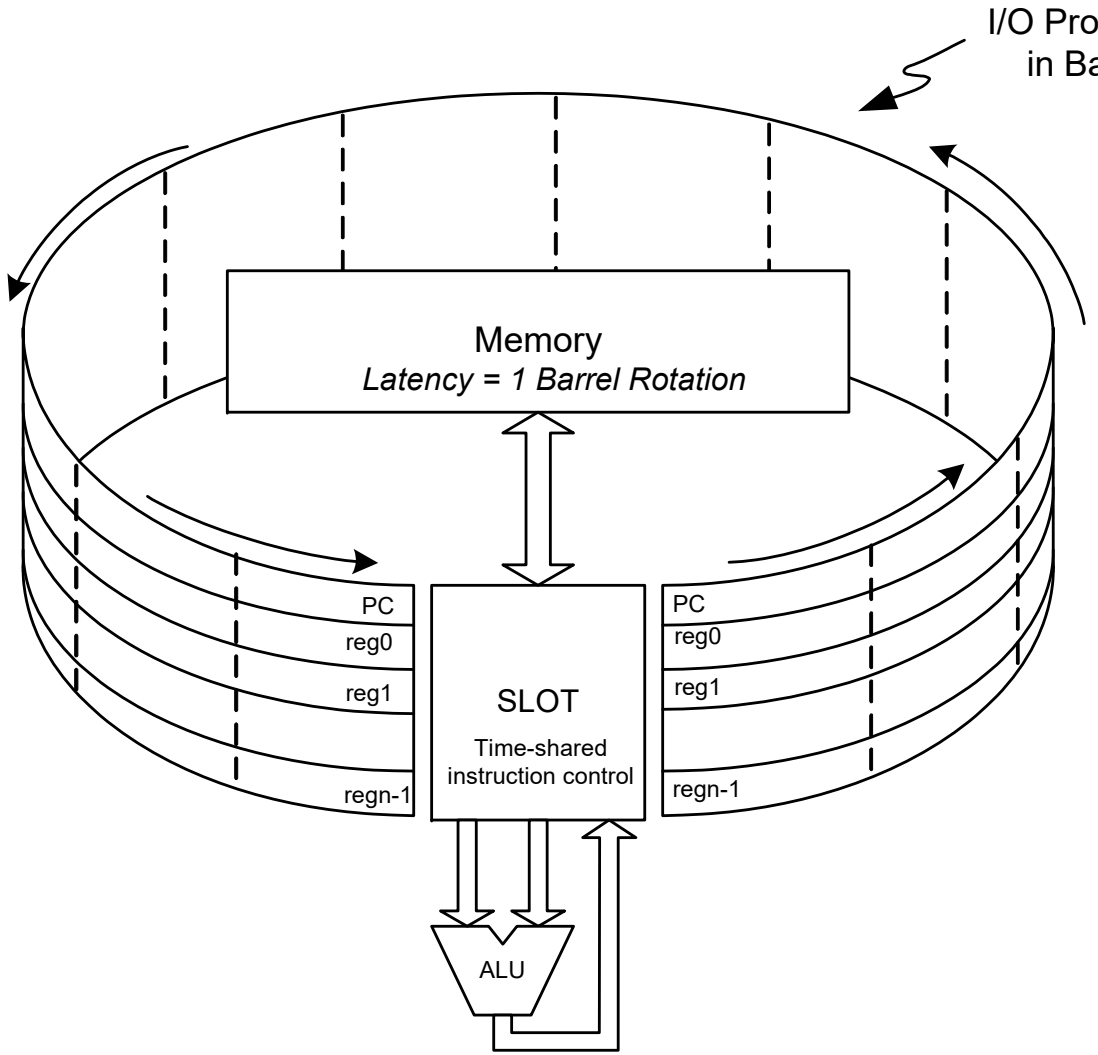


CDC 6600 Peripheral Processors

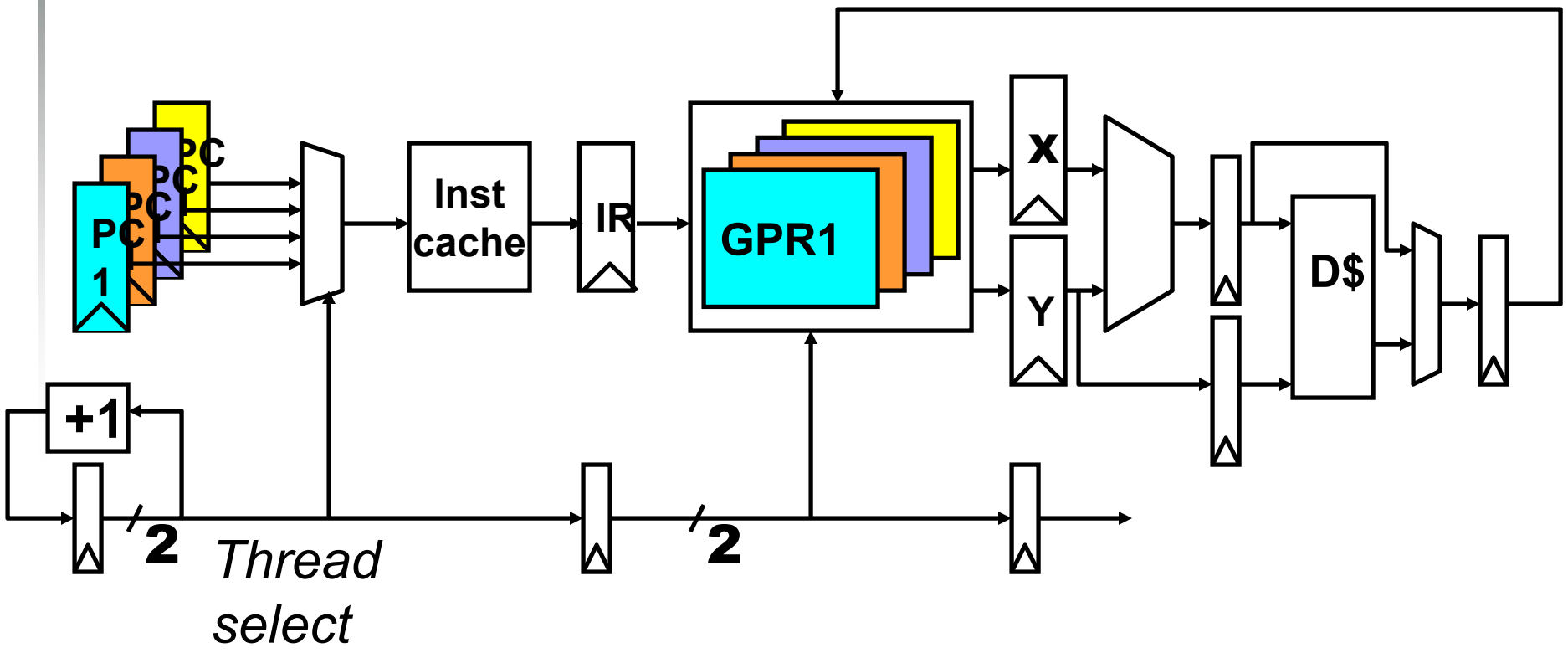
(Cray, 1964)

- 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

CDC 6600 Peripheral Processors



Simple Multithreaded Pipeline



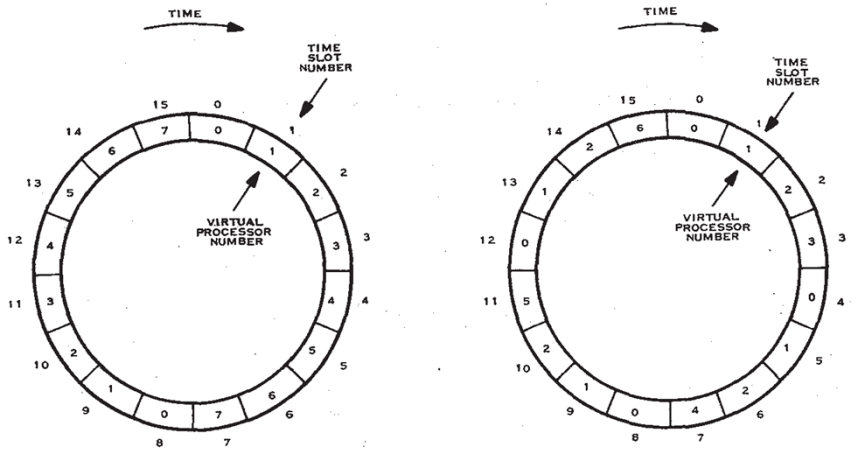
- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

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

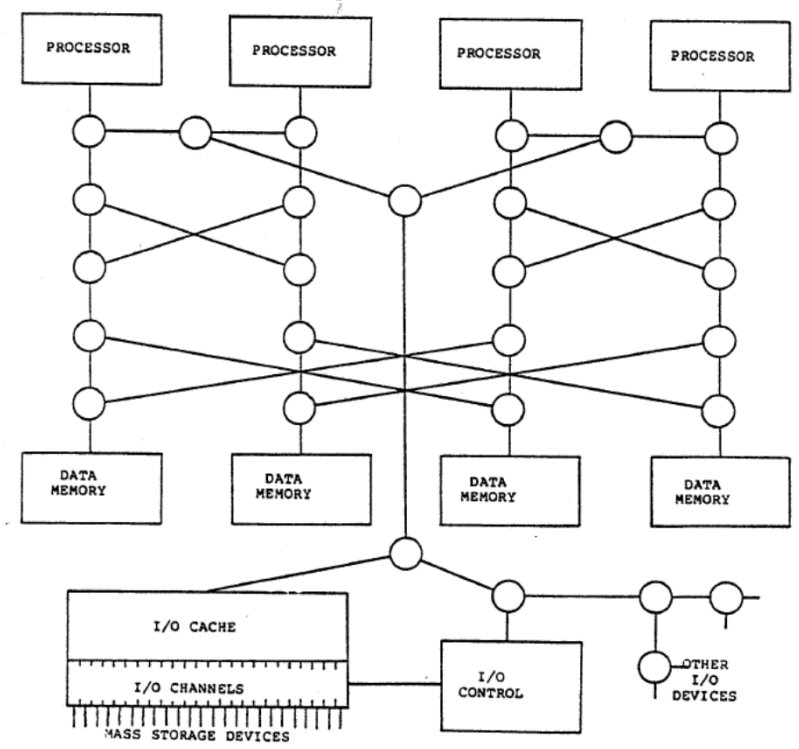
Thread Scheduling Policies

- **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** in *TI ASC* with 8 Virtual Processors in the Peripheral Processor with 16 pipeline slots, 1971
  - OS allocates 16 pipeline slots amongst 8 threads
  - Hardware performs fixed interleave over 16 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

Denelcor HEP (Burton Smith, 1982)



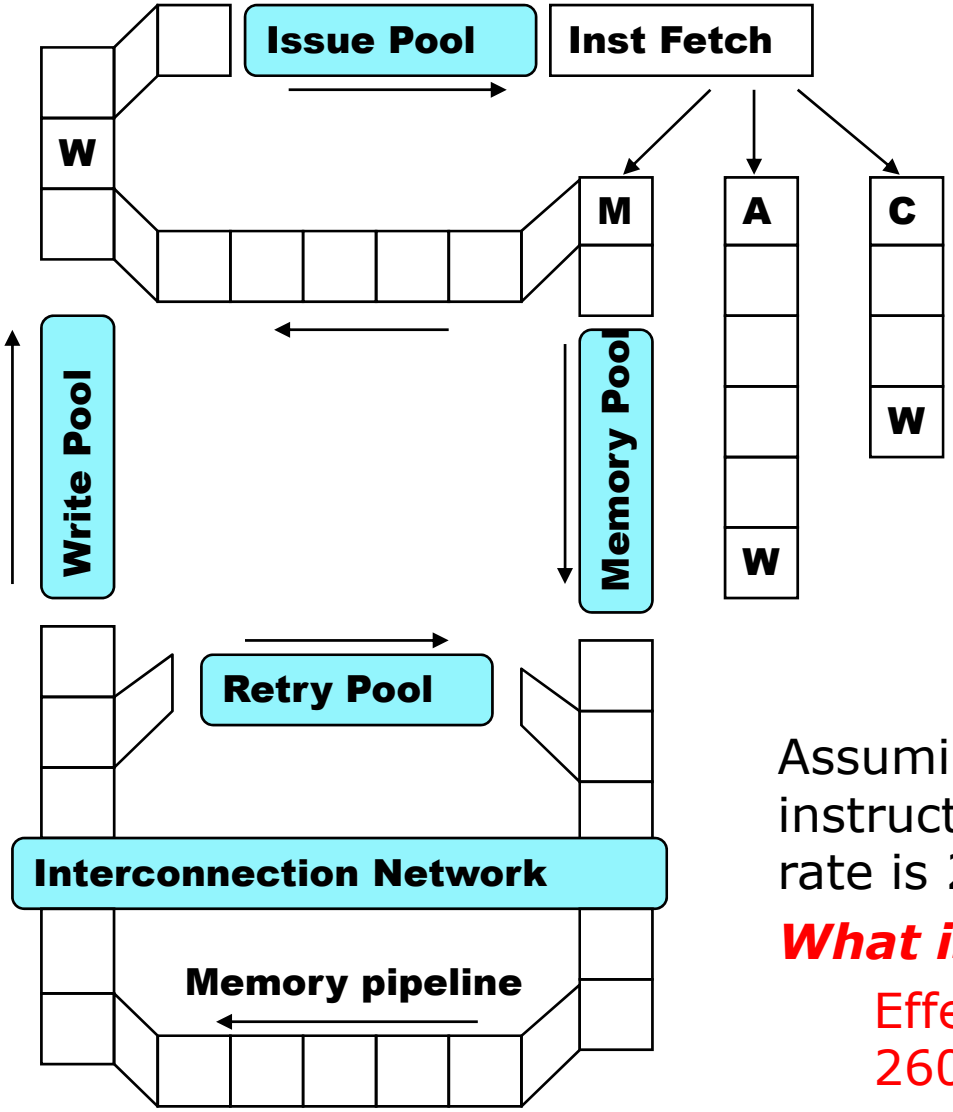
First commercial machine to use hardware threading in main CPU

- 120 threads per processor
- 10 MHz clock rate
- Up to 16 processors and up to 128 memory modules
- Three-input switches and chaotic routing
- precursor to Tera MTA (Multithreaded Architecture)

Tera 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

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

Assuming a single thread issues one instruction every 21 cycles, and clock rate is 260 MHz…

What is single-thread performance?

Effective single-thread issue rate is 260/21 = 12.4 MIPS

Coarse-Grain Multithreading

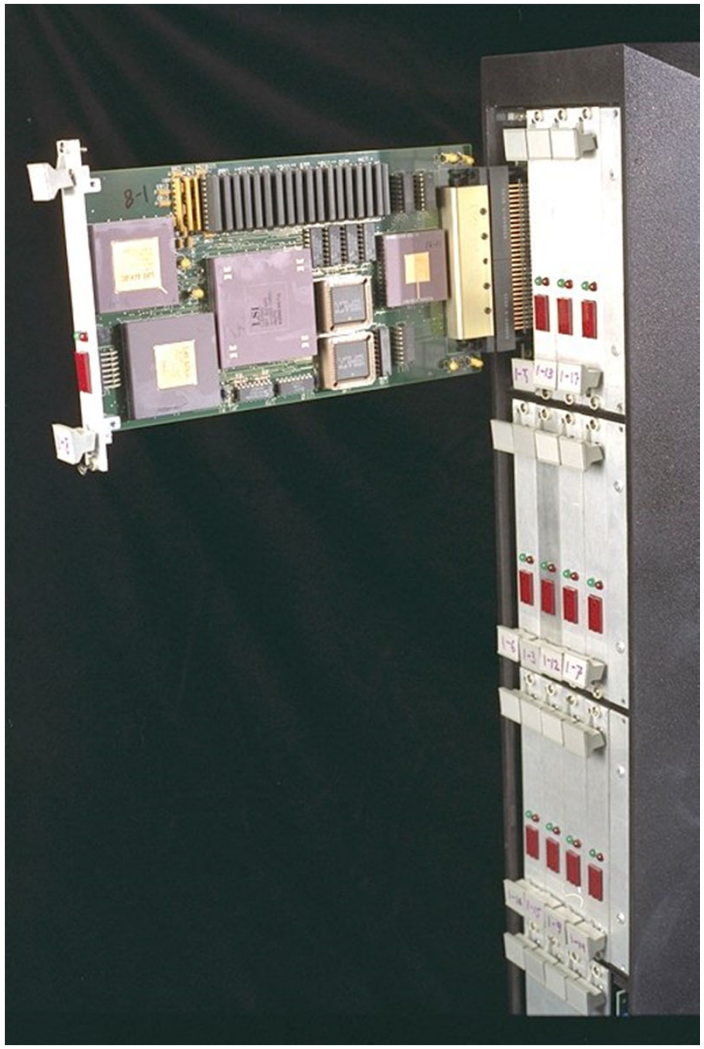
Tera MTA designed for supercomputing applications with large data sets and low locality

- 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

MIT Alewife (1990)



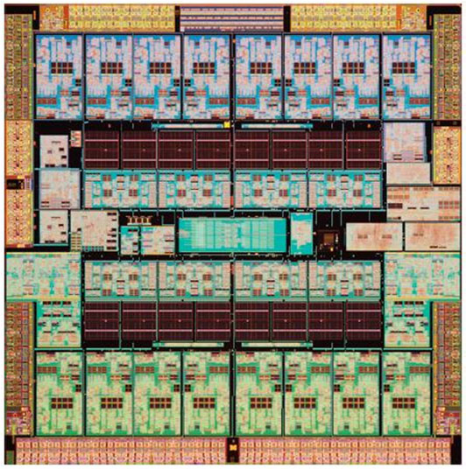
- Modified SPARC chips
  - register windows hold different thread contexts
- Up to four threads per node
- Thread switch on local cache miss

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

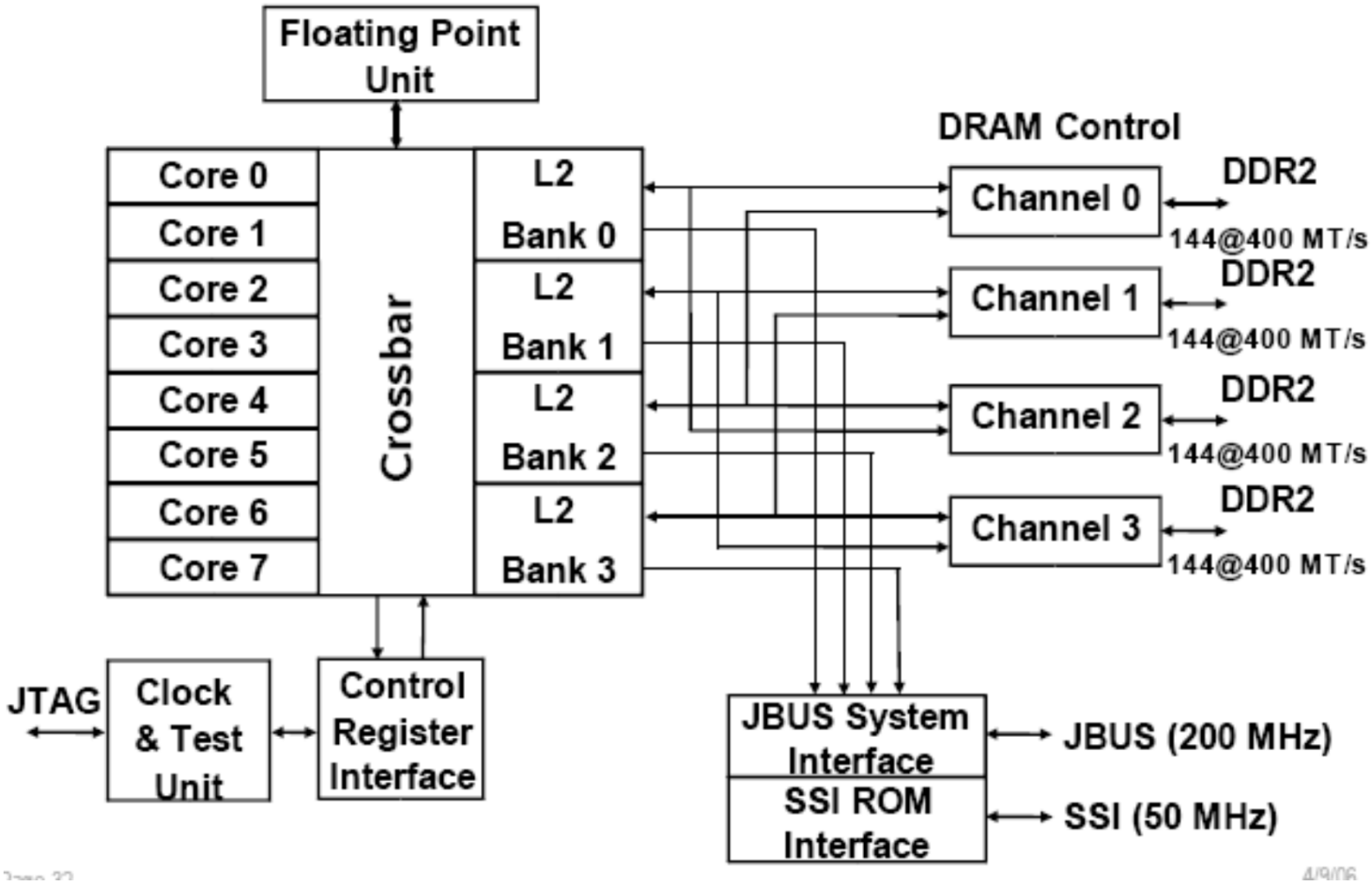
Oracle/Sun Niagara processors

- 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



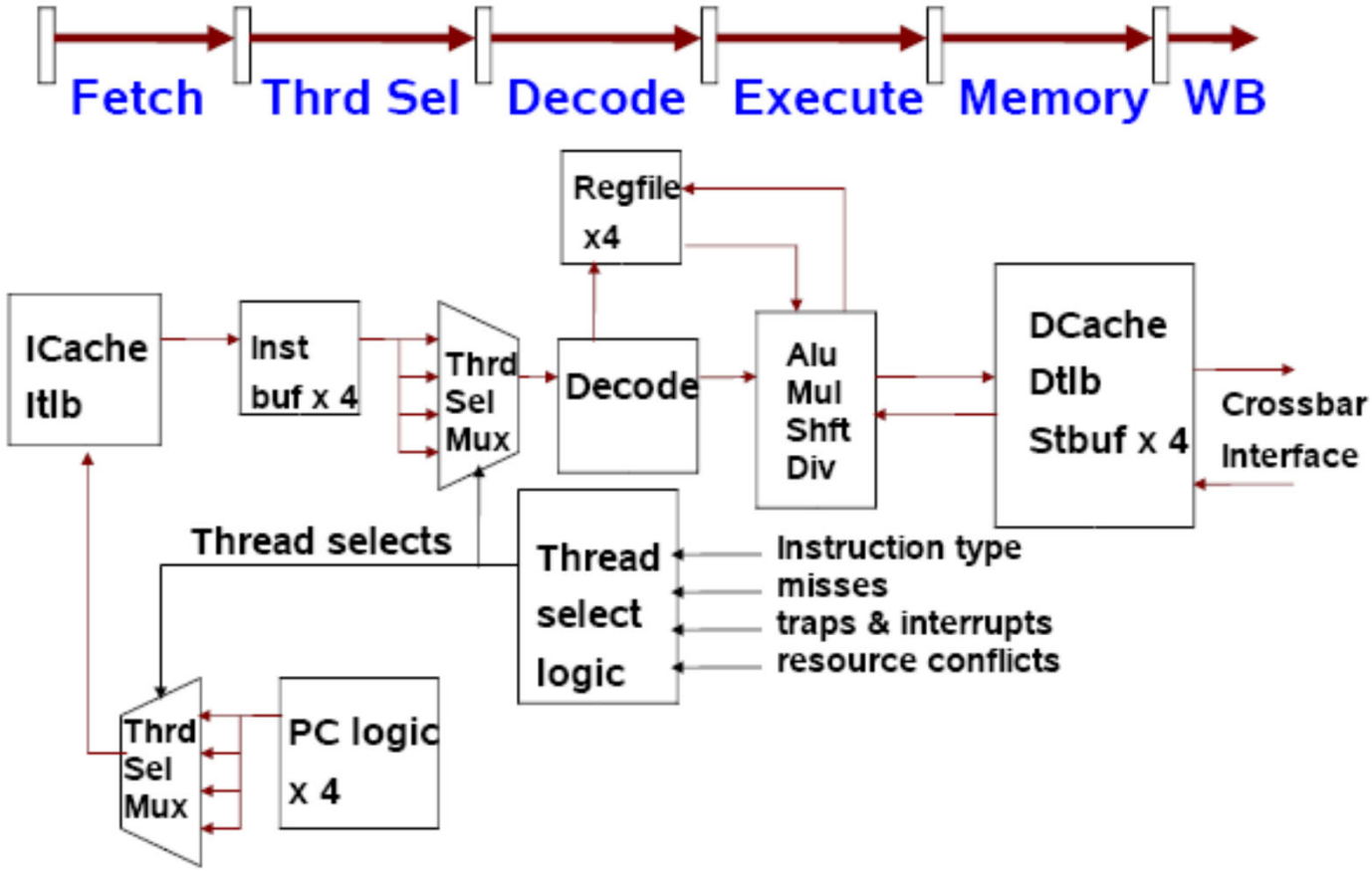
- Niagara-1 [2004], 8 cores, 4 threads/core
- Niagara-2 [2007], 8 cores, 8 threads/core
- Niagara-3 [2009], 16 cores, 8 threads/core

Niagara-1 Block Diagram



- 8 in-order cores, 4 threads each
- 4 L2 banks, 4 DDR2 memory controllers

Niagara-1 Pipeline



- 6-stage pipeline
- Fine-grained multithreading

Modern Day Simultaneous Multithreading

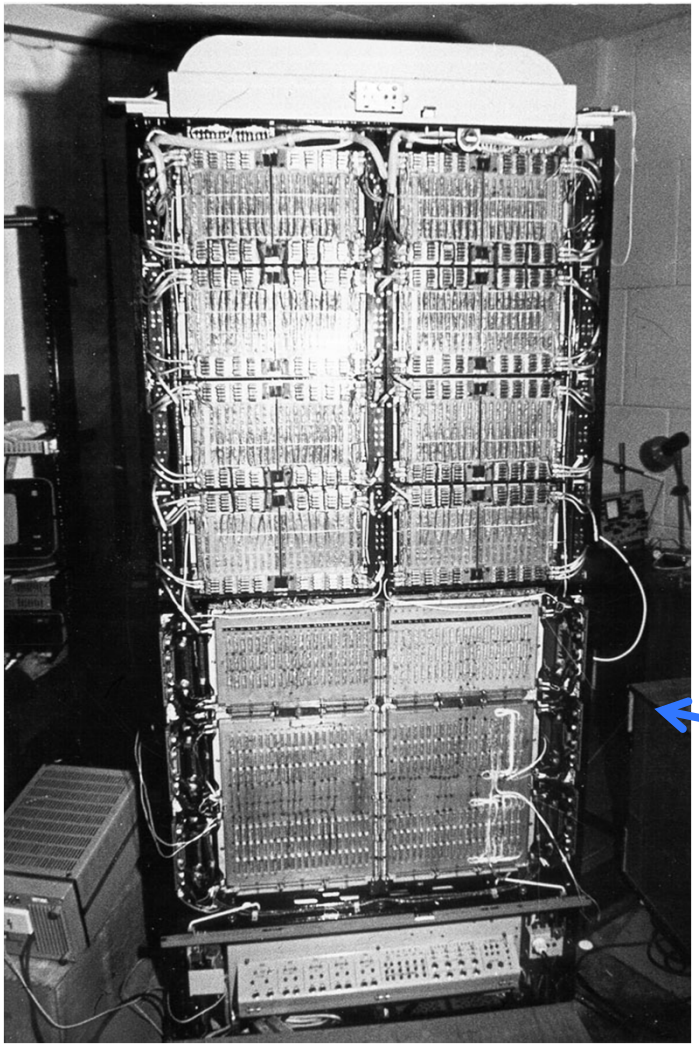
- Apply to superscalar pipelines
  - More resources to share
- Also one-wide in-order processors
  - Provide high efficiency for throughput-oriented servers
- Examples: Intel P4, Intel Nehalem, IBM Power 5/6/7/8, Alpha EV8/21464, AMD Zen

Simultaneous Multithreading (SMT) for OoO Superscalars

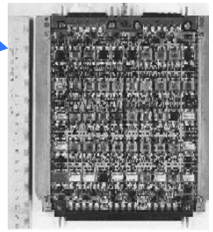
- Techniques presented so far have all been "**vertical**" multithreading where each pipeline stage works on **one thread at a time**
- **SMT** uses fine-grain control already present inside an OoO superscalar to allow **instructions from multiple threads to enter execution on same clock cycle.** Gives better utilization of machine resources.
- Benefits:
  - Natural fit for OOO superscalar
  - Improved throughput
  - Low incremental cost
- Drawbacks:
  - Additional complexity over OOO superscalar
  - Cache conflicts

MARS-M: Truly First SMT System (M. Dorojevets, 1988)

built & demonstrated well before Tulsen's SMT paper, IBM, and Intel!



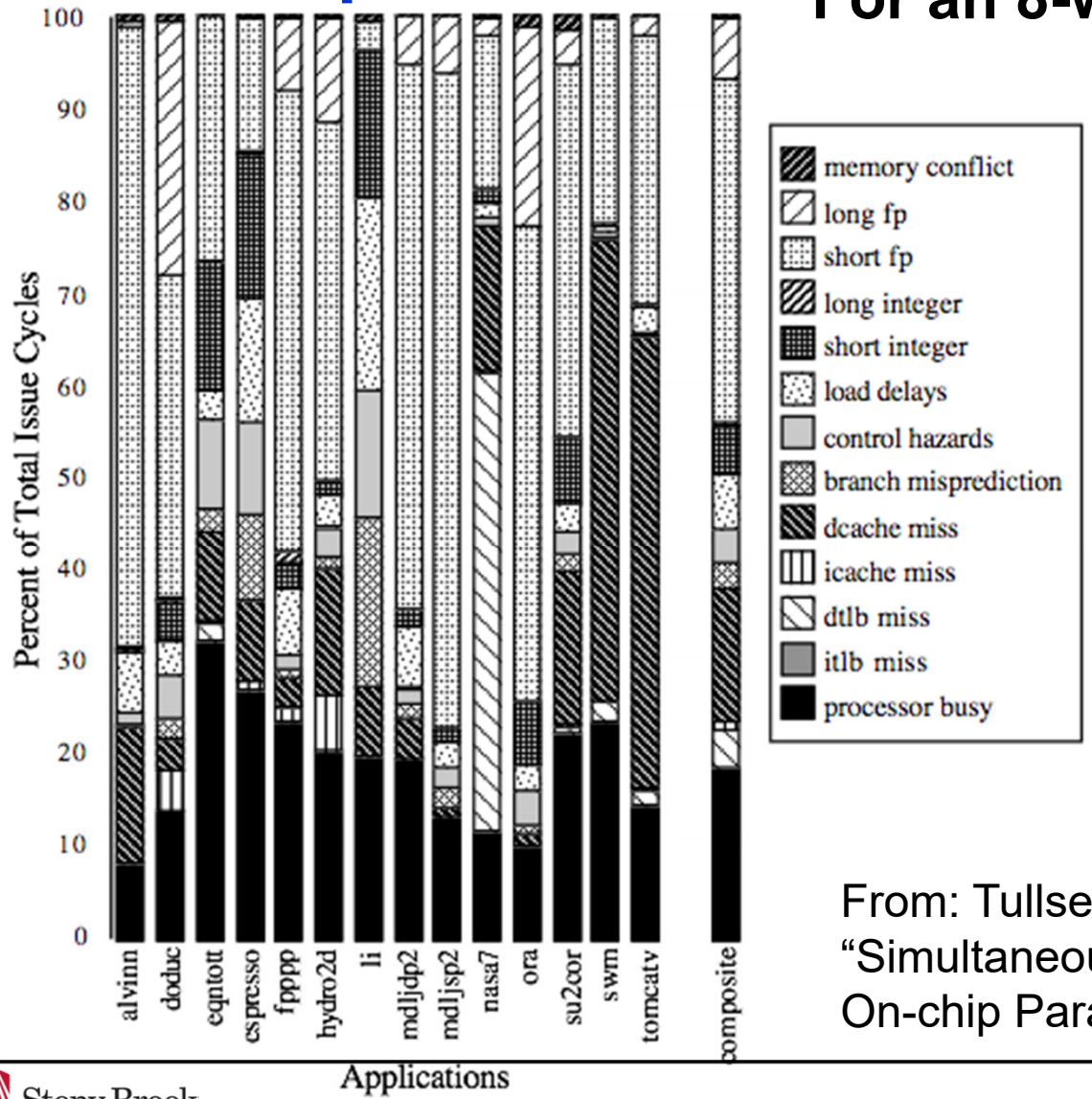
- **Decoupled multithreaded architecture** with Control, Address, and Execution multithreaded processors and multi-ported multi-bank memory sub-system
- **Simultaneous multithreading** in the Address and Execution VLIW processors
  - Up to 4 threads can issue VLIW instructions each cycle in each Address/Execution processor
- **Coarse-grain multithreading** in Control VLIW processor with 8 thread contexts
  - thread switch on issuing load instructions (no cache)
- Three water-cooled cabinets
  - memory-subsystem cabinet shown on the left photo
- 739 boards with up 100 MSI and LSI ECL ICs mounted on both sides



Dorozhevets M.N. and Wolcott P. The El'brus-3 and MARS-M: Recent Advances in Russian High-performance Computing, The Journal of Supercomputing, USA, 6, pp. 5-48, 1992

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

For an 8-way superscalar



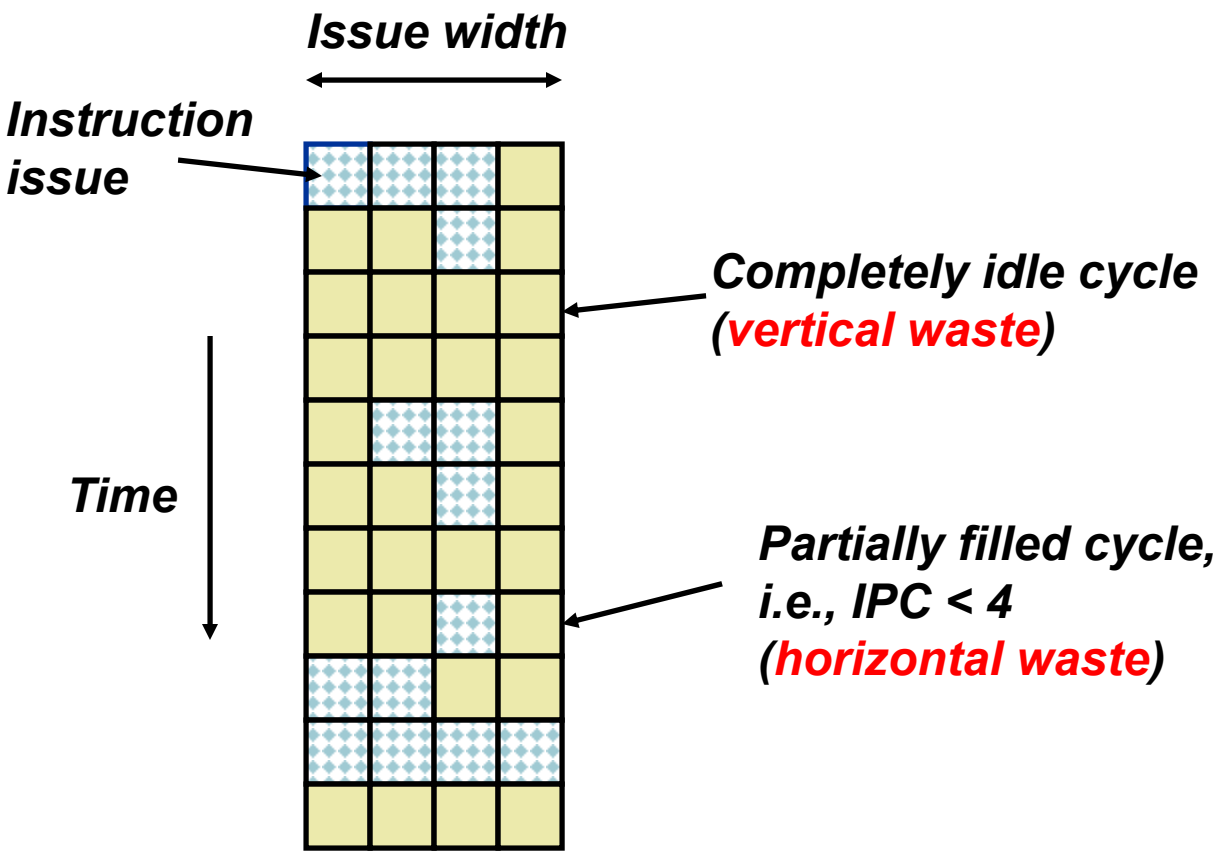
From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism", ISCA 1995.

O-o-O Simultaneous Multithreading

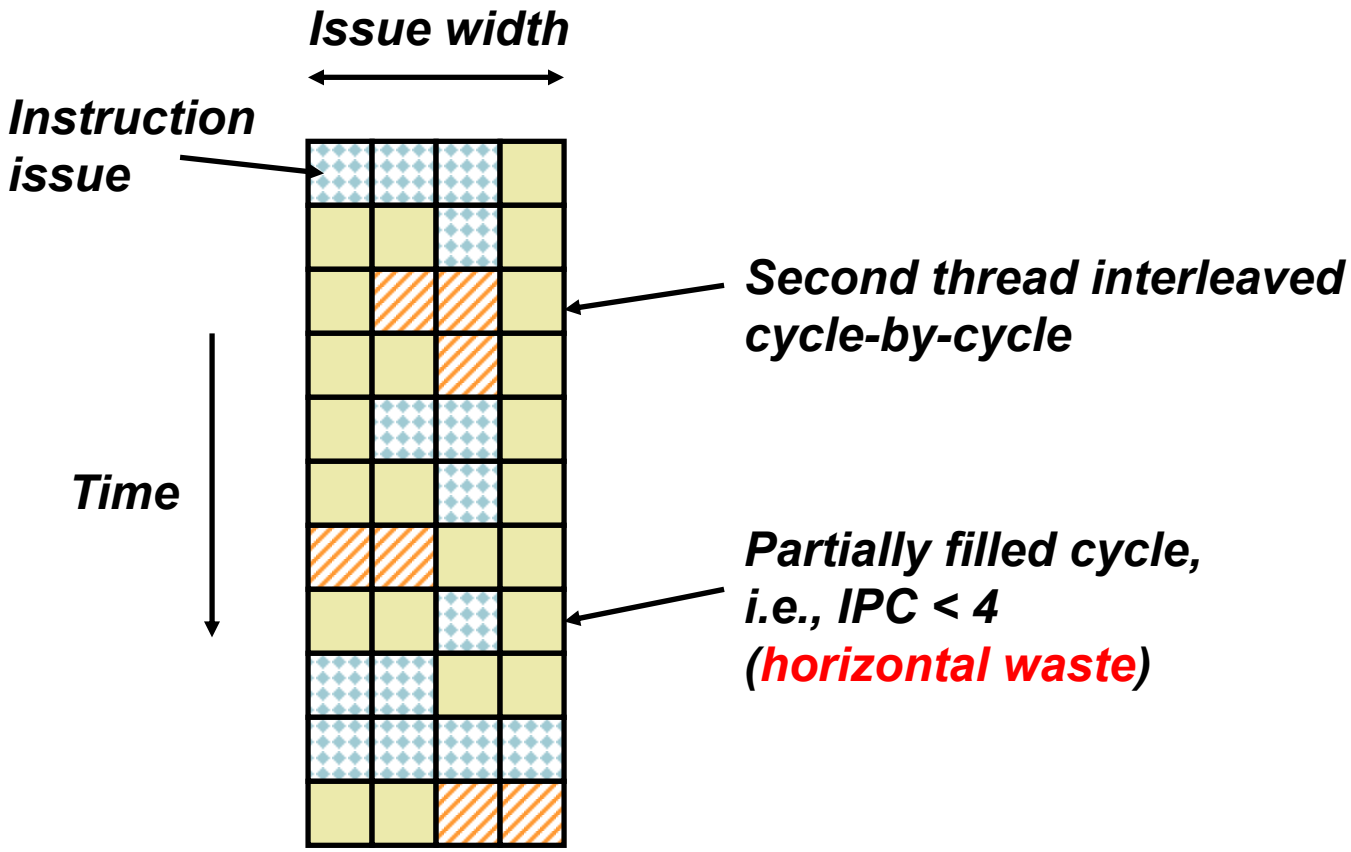
[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

- 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

Superscalar Machine Efficiency



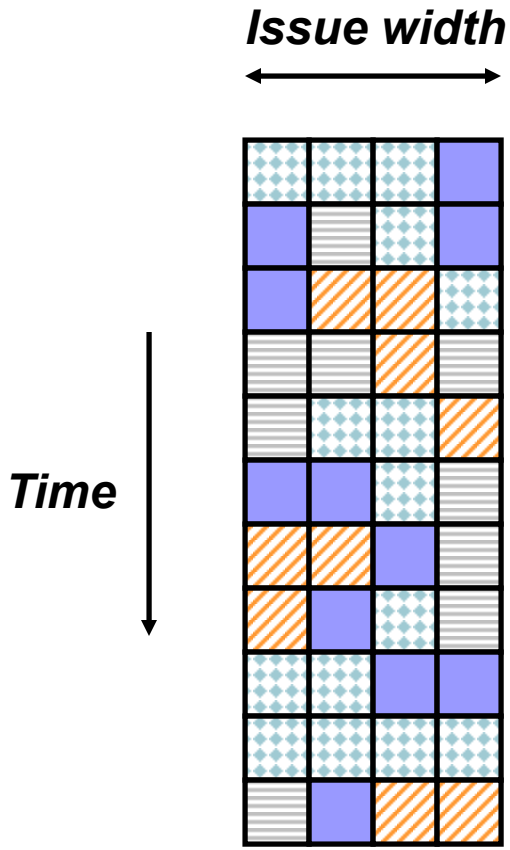
Vertical Multithreading



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

Ideal Superscalar Multithreading

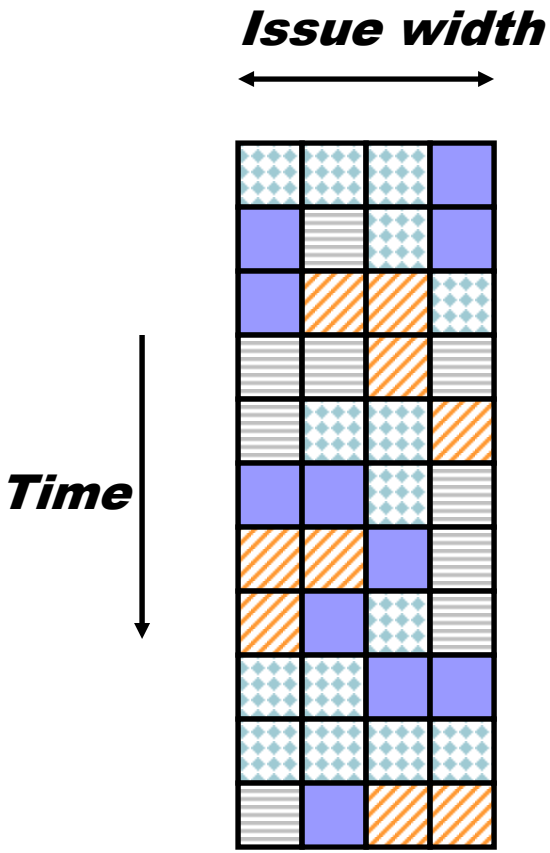
[Tullsen, Eggers, Levy, UW, 1995]



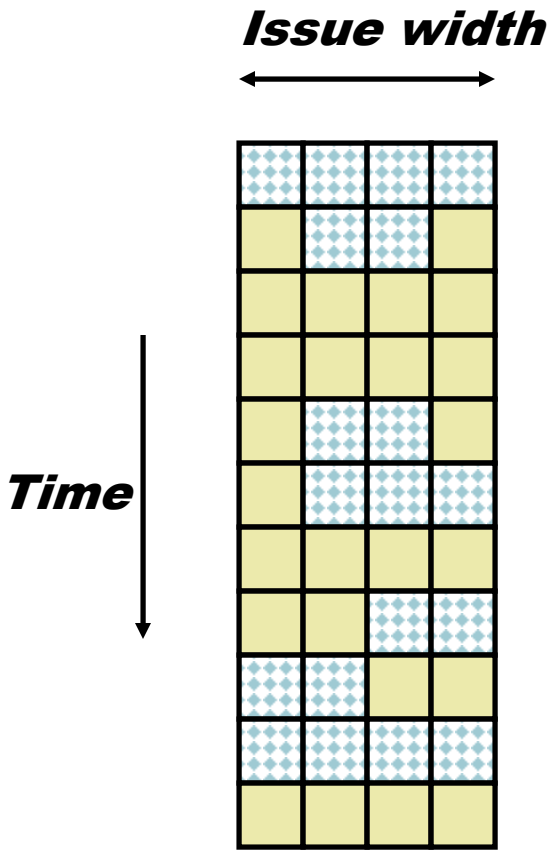
- Interleave multiple threads to multiple issue slots with no restrictions

SMT Adaptation to Parallelism Type

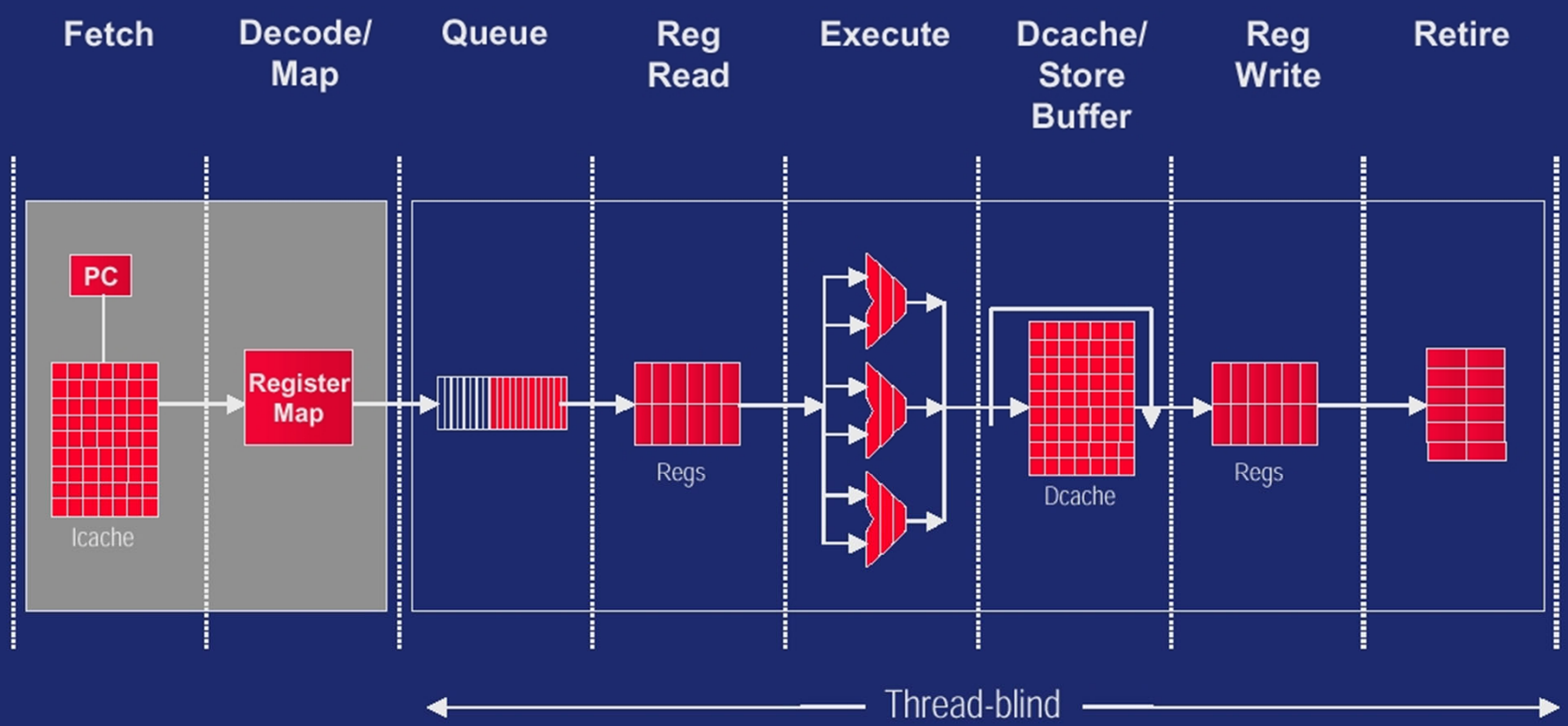
For regions with high thread level parallelism (TLP) entire machine width is shared by all threads



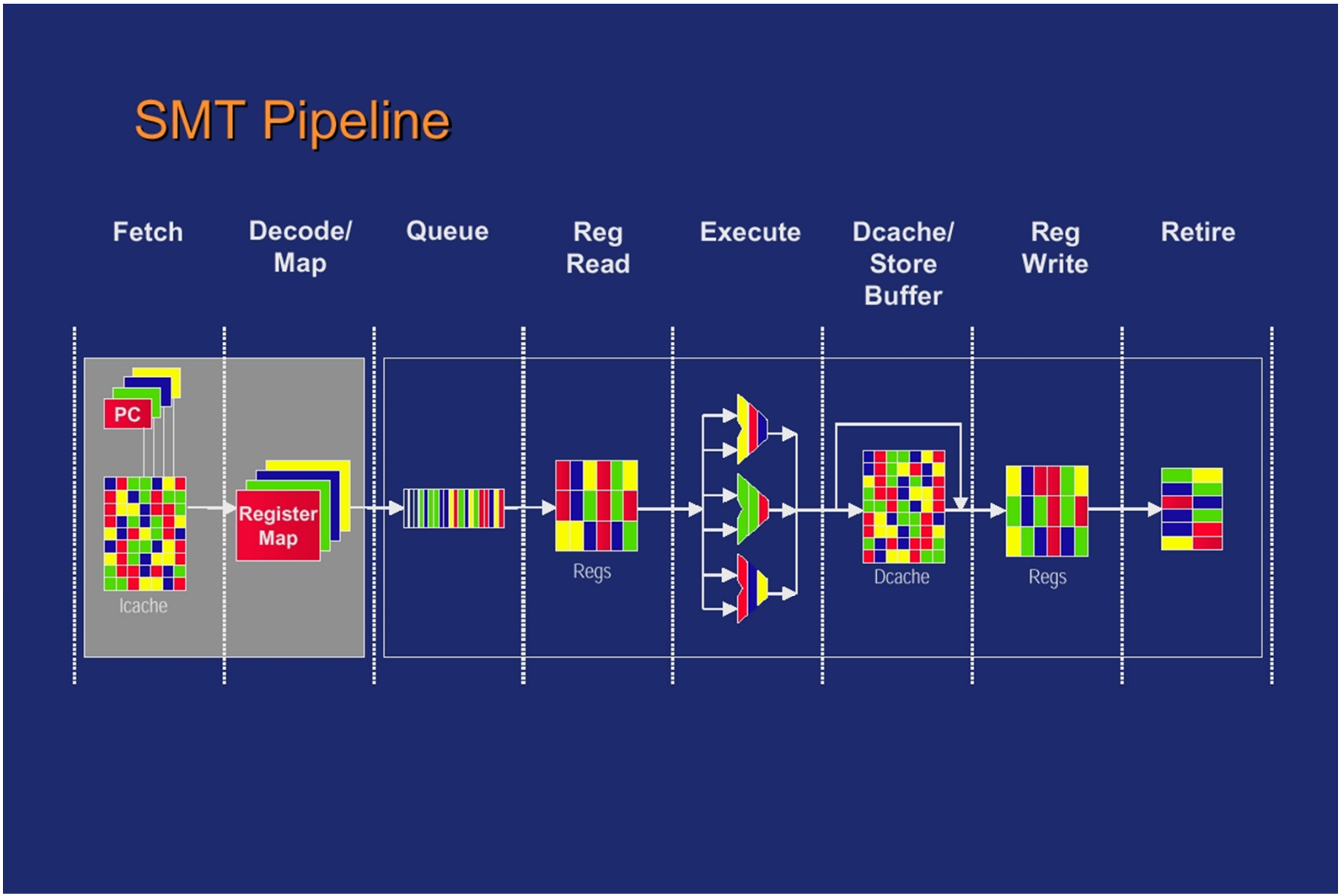
For regions with low thread level parallelism (TLP) entire machine width is available for instruction level parallelism (ILP)



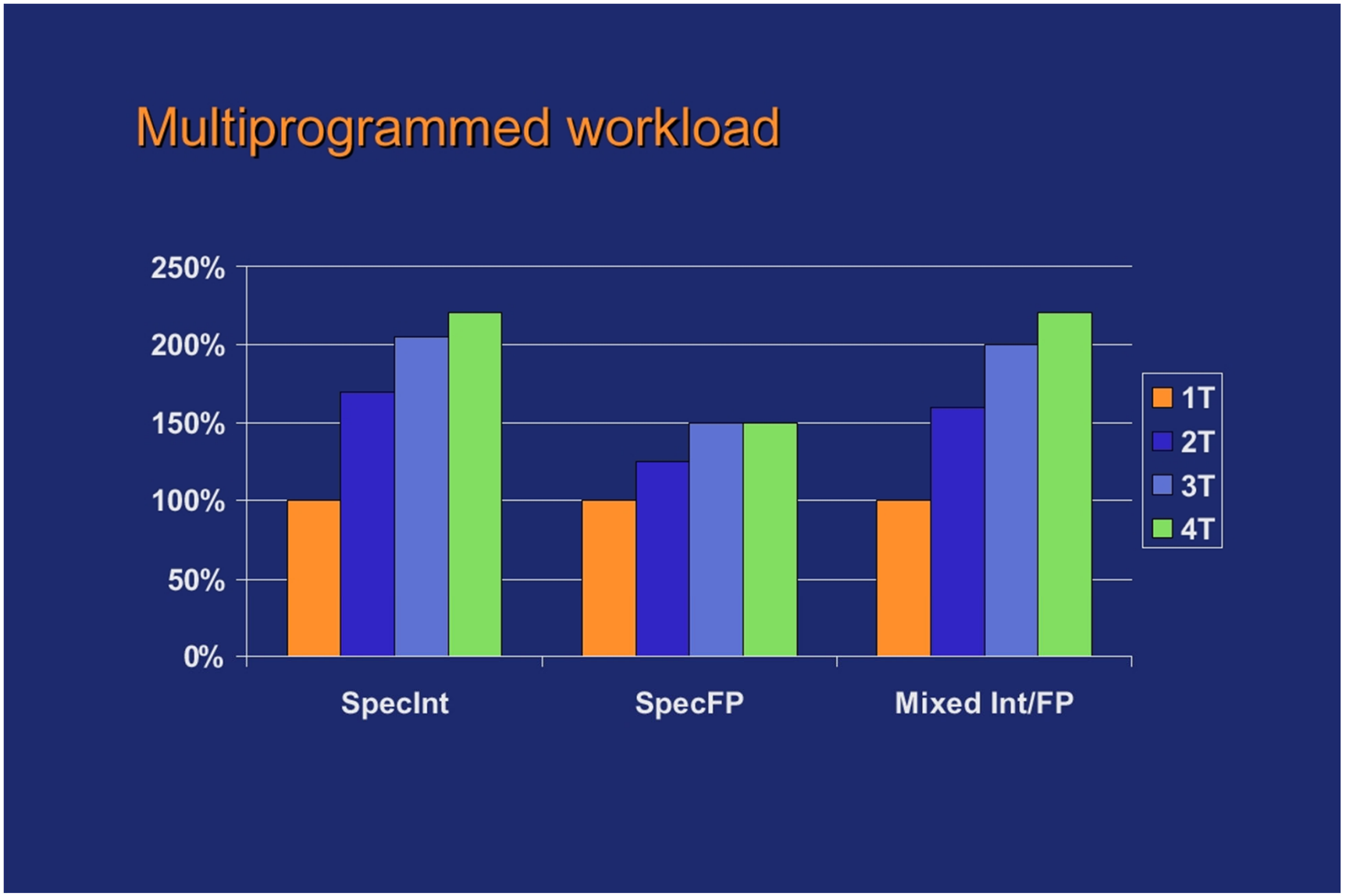
Basic Out-of-order Pipeline



SMT Microarchitecture



SMT Performance



Pentium-4 Hyperthreading (2002)

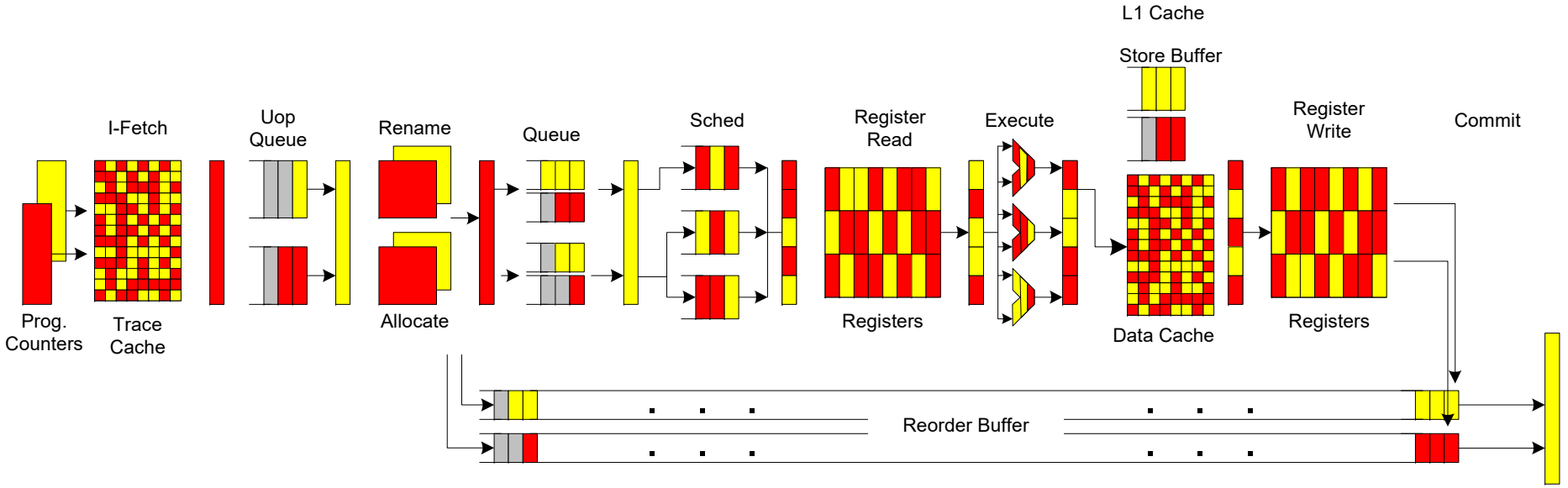
- First commercial SMT design (2-way SMT)
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - 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

Intel Hyperthreading

- Part of Pentium 4 design (Xeon)
- Two threads per processor
- Goals
  - Low cost – less than 5% overhead for replicated state
  - Assure forward progress of both threads
    - Make sure both threads get some buffer resources
    - through partitioning or budgeting
  - Single thread running alone does not suffer slowdown

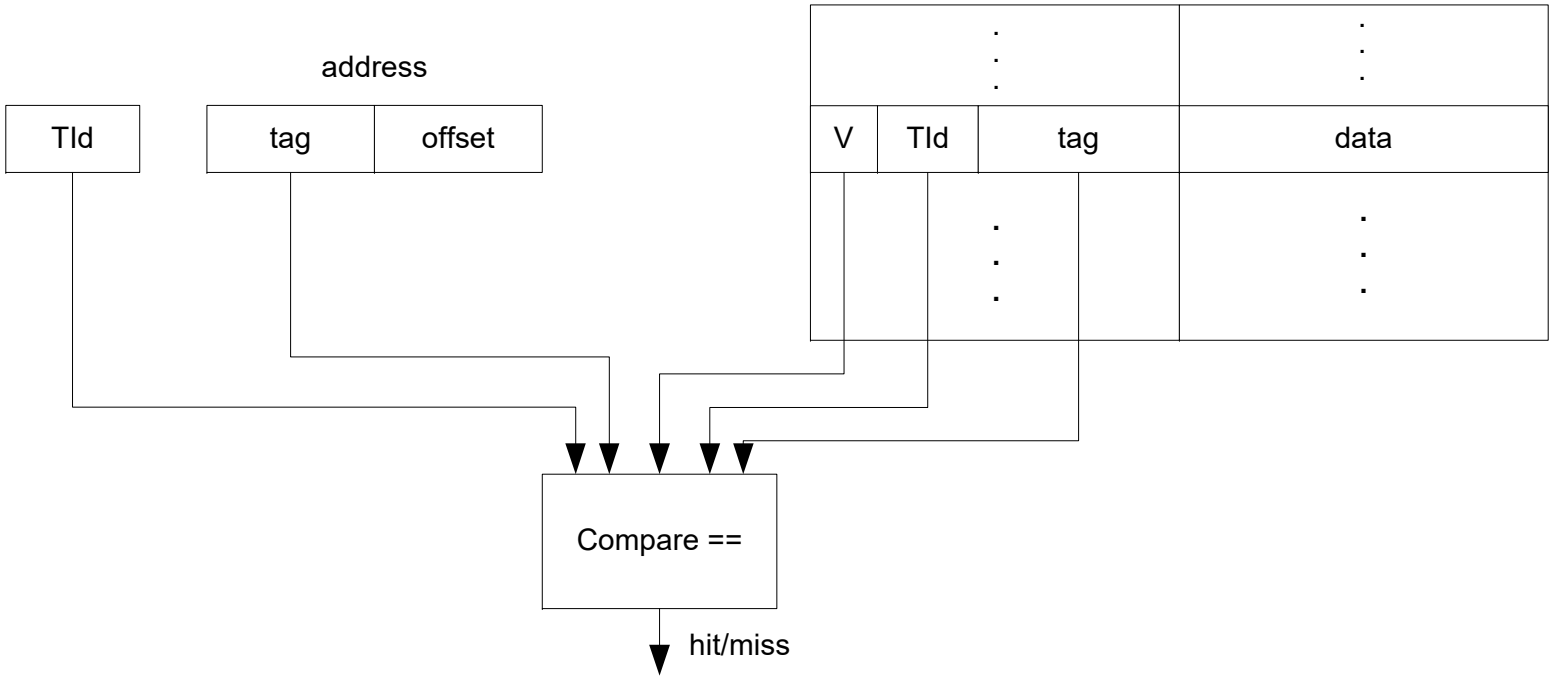
Intel Hyperthreading

- Main pipeline
  - Pipeline prior to trace cache not shown
- Round-Robin instruction fetching
  - Alternates between threads
  - Avoids dual-ported trace cache
  - BUT trace cache is a shared resource



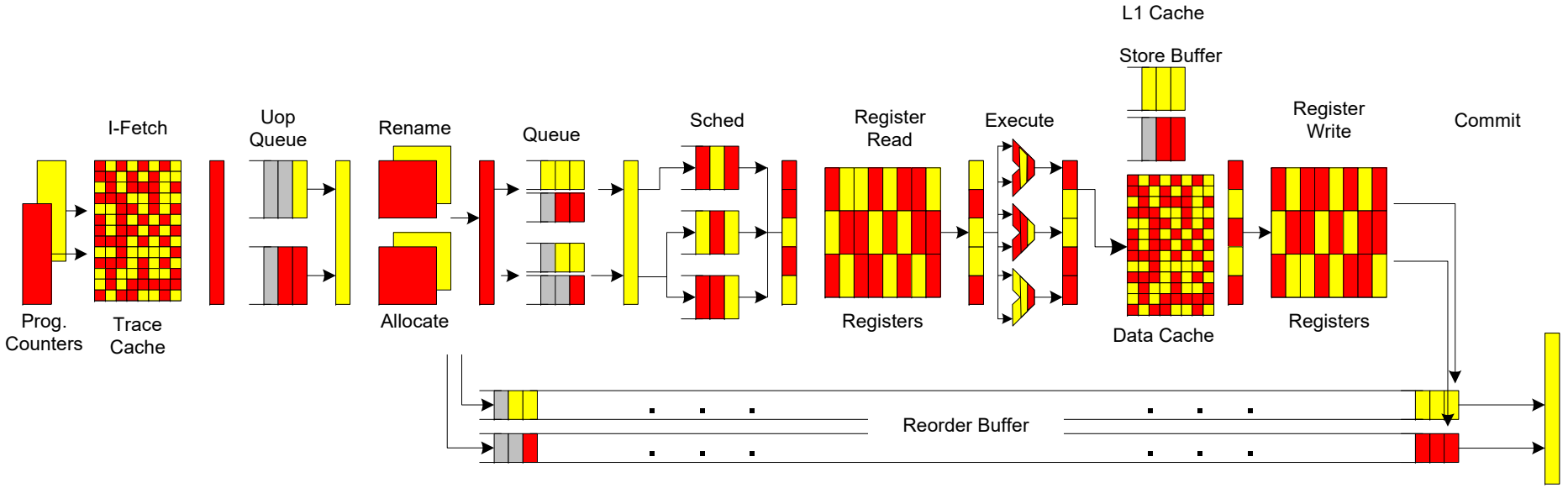
Capacity Resource Sharing

- Append thread identifier (TId) to threads in shared capacity (storage) resource
- Example: cache memory



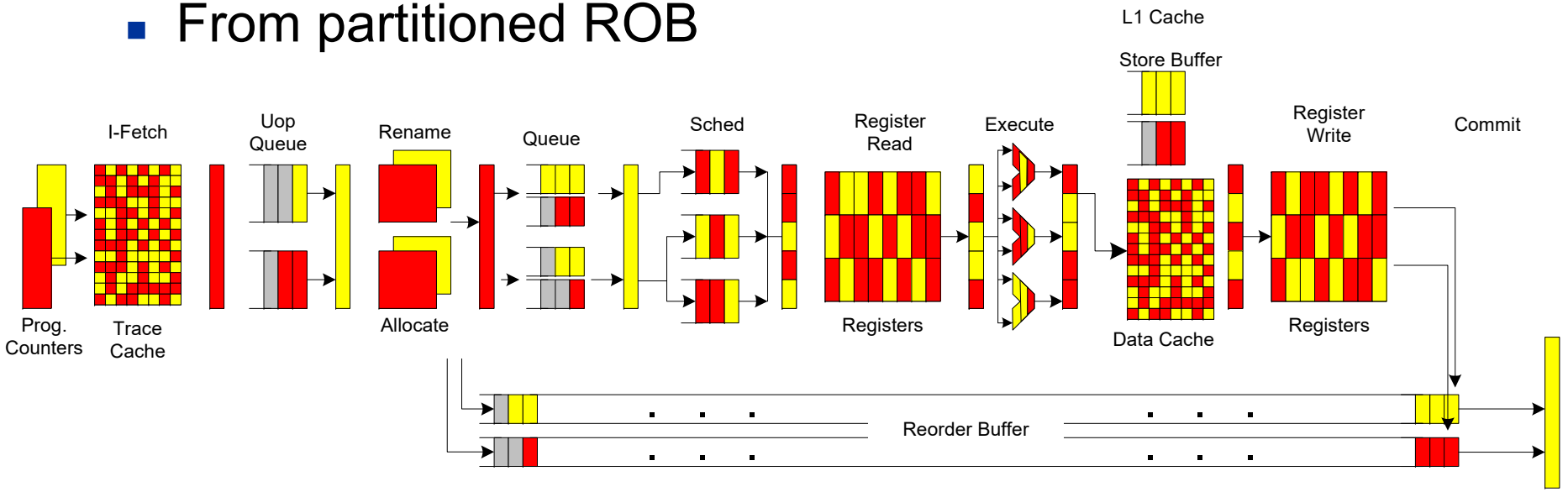
Frontend Implementation

- Partitioned front-end resources
  - Fetch queue (holds uops)
  - Rename and allocate tables
  - Post-rename queues
- Partitioning assures forward progress if other thread is blocked



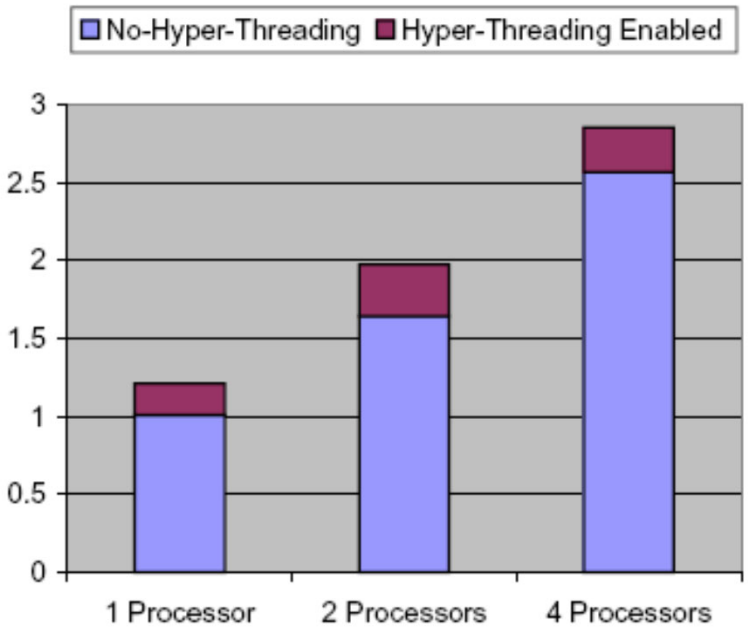
Backend Implementation

- Physical registers are pooled (shared)
- Five instruction buffers (schedulers)
  - Shared
  - With an upper limit
- Instruction issue is irrespective of thread ID
- Instruction commit is round-robin
  - From partitioned ROB



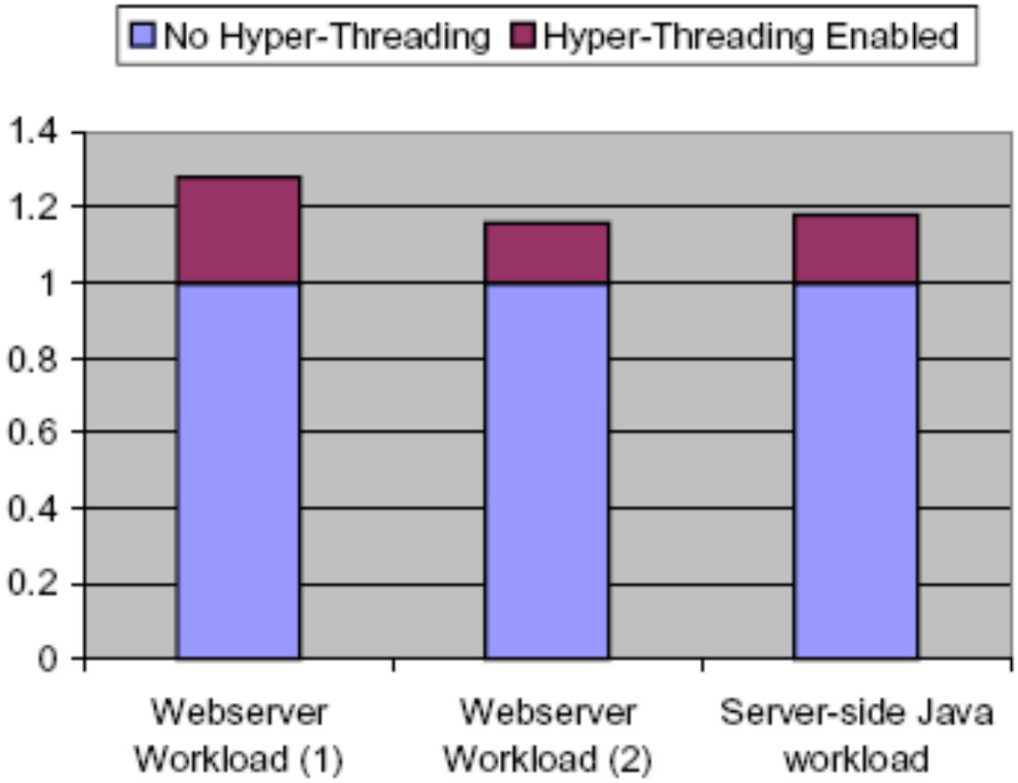
Performance

- Online Transactional Processing (OLTP) workload
  - 21% gain in single and dual systems
  - Likely external bottleneck in 4 processor systems
    - Most likely front-side bus (FSB), i.e. memory bandwidth



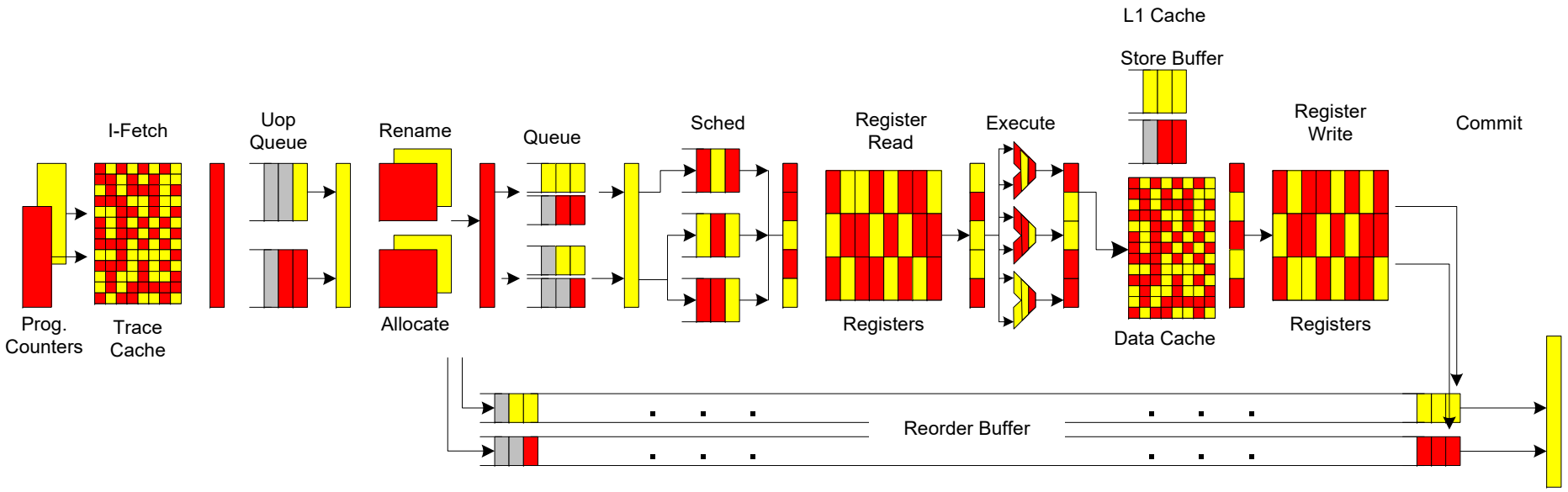
Performance

- Web server apps

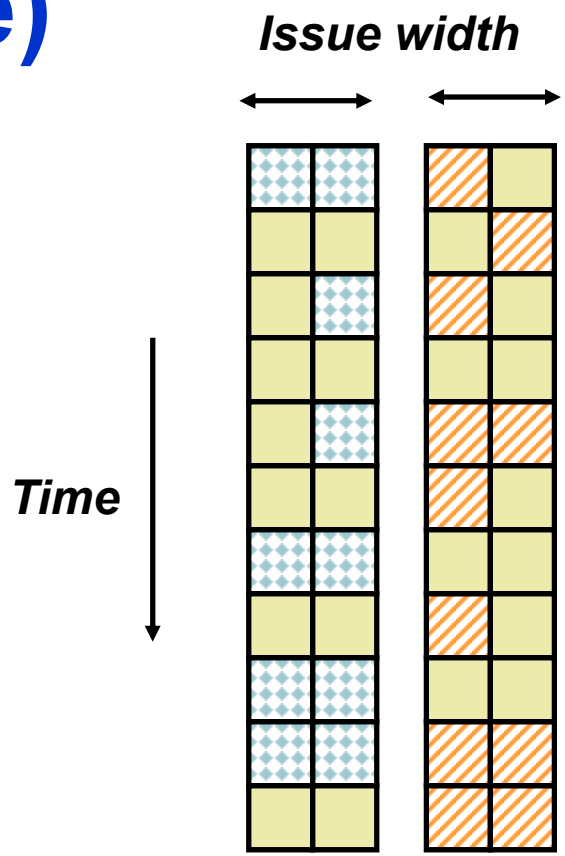


Intel Hyperthreading Summary

- Mix of partitioned and shared resources
- Mostly round-robin scheduling
- Primary objective: performance
- Secondary objective: fairness



Chip Multiprocessing (CMP) (multicore)



- What is the effect of splitting into multiple processor cores?
  - reduces horizontal waste,
  - leaves some vertical waste, and
  - puts upper limit on peak throughput of each thread.
  - allows have different cores designed/optimized for different tasks

13th Gen Intel® Core™ Desktop Processors

Processor Number	Processor Cores (P+E)	Processor Threads	Intel® Smart Cache (L3)	Total L2 Cache	P-core Max Turbo Frequency (GHz)	E-core Max Turbo Frequency (GHz)	P-core Base Frequency (GHz)	E-core Base Frequency (GHz)	Processor Graphics	Total CPU PCIe Lanes	Max Memory Speed (MT/S)	Memory Capacity	Processor Base Power (W)	Max Turbo Power (W)	RCP (USD$)
i9-13900	24 (8+16)	32	36MB	32MB	Up to 5.6	Up to 4.2	2.0	1.5	Intel® UHD Graphics 770	20	DDR5 5600 DDR4 3200	128GB	65	219	$549
i9-13900F	24 (8+16)	32	36MB	32MB	Up to 5.6	Up to 4.2	2.0	1.5	n/a	20	DDR5 5600 DDR4 3200	128GB	65	219	$524
i7-13700	16 (8+8)	24	30MB	24MB	Up to 5.2	Up to 4.1	2.1	1.5	Intel® UHD Graphics 770	20	DDR5 5600 DDR4 3200	128GB	65	219	$384
i7-13700F	16 (8+8)	24	30MB	24MB	Up to 5.2	Up to 4.1	2.1	1.5	n/a	20	DDR5 5600 DDR4 3200	128GB	65	219	$359
i5-13600	14 (6+8)	20	24MB	11.5MB	Up to 5.0	Up to 3.7	2.7	2.0	Intel® UHD Graphics 770	20	DDR5 4800 DDR4 3200	128GB	65	154	$255
i5-13500	14 (6+8)	20	24MB	11.5MB	Up to 4.8	Up to 3.5	2.5	1.8	Intel® UHD Graphics 770	20	DDR5 4800 DDR4 3200	128GB	65	154	$232
I5-13400	10 (6+4)	16	20MB	9.5MB	Up to 4.6	Up to 3.3	2.5	1.8	Intel® UHD Graphics 730	20	DDR5 4800 DDR4 3200	128GB	65	148	$221
i5-13400F	10 (6+4)	16	20MB	9.5MB	Up to 4.6	Up to 3.3	2.5	1.8	n/a	20	DDR5 4800 DDR4 3200	128GB	65	148	$196
i3-13100	4 (4+0)	8	12MB	5MB	Up to 4.5	n/a	3.4	n/a	Intel® UHD Graphics 730	20	DDR5 4800 DDR4 3200	128GB	60	89	$134
i3-13100F	4 (4+0)	8	12MB	5MB	Up to 4.5	N/a	3.4	n/a	n/a	20	DDR5 4800 DDR4 3200	128GB	58	89	$109

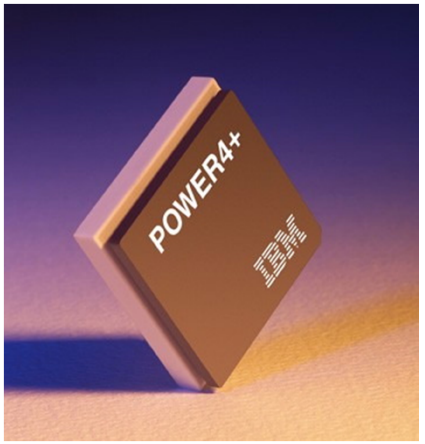
Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families.
The frequency of cores and core types varies by workload, power consumption and other factors. Visit https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html for more information. Max Turbo Frequency for P-cores may include Intel® Thermal Velocity Boost and/or Intel Turbo Boost Max 3.0.
All SKUs listed above support up to DDR5 (5600 MT/S)/DDR4 (3200 MT/S) memory.See ark.intel.com for more specification details

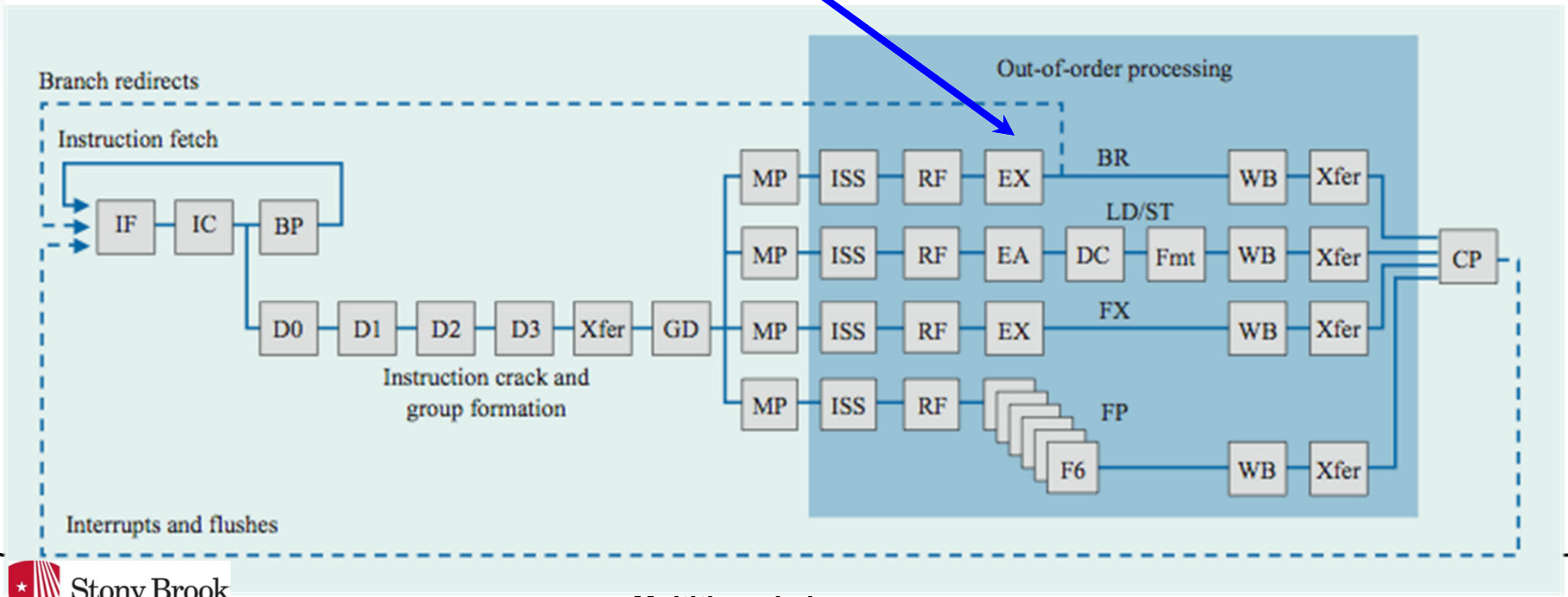
intel.

- P cores: high-performance SMT2 cores with hyperthreading
- E cores: Energy-efficient non-SMT (single-thread) cores for background tasks

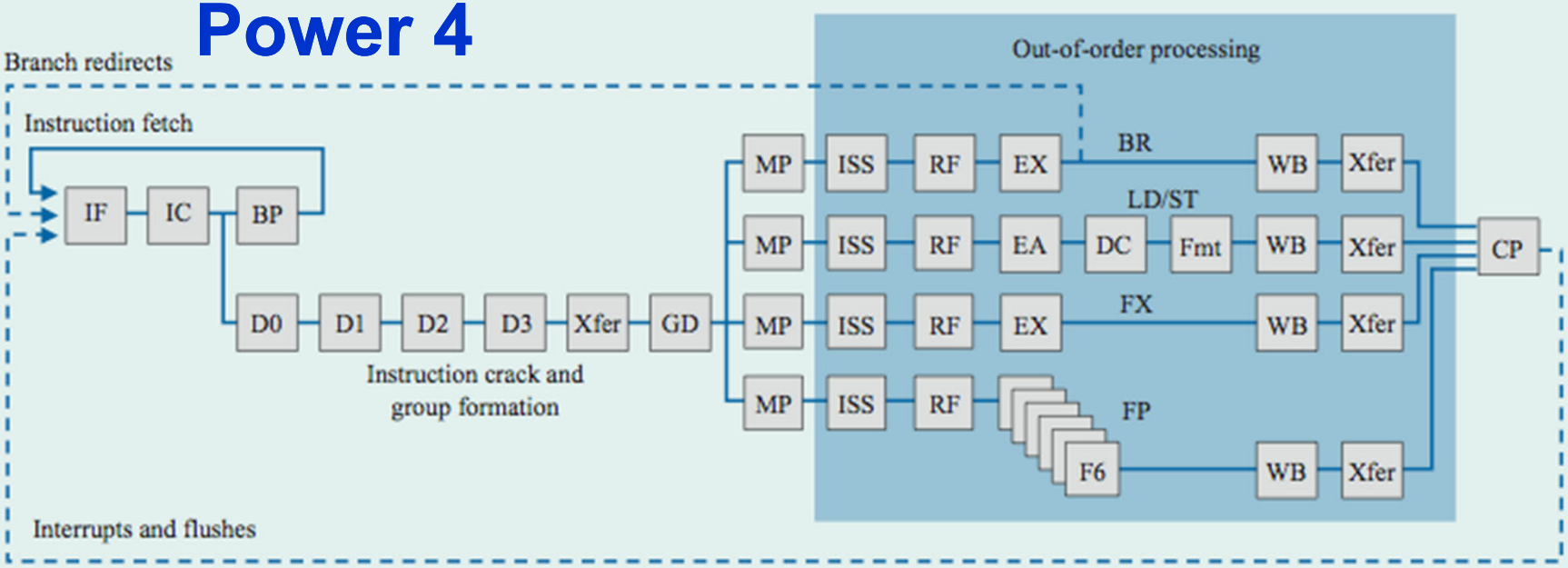
IBM Power 4

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

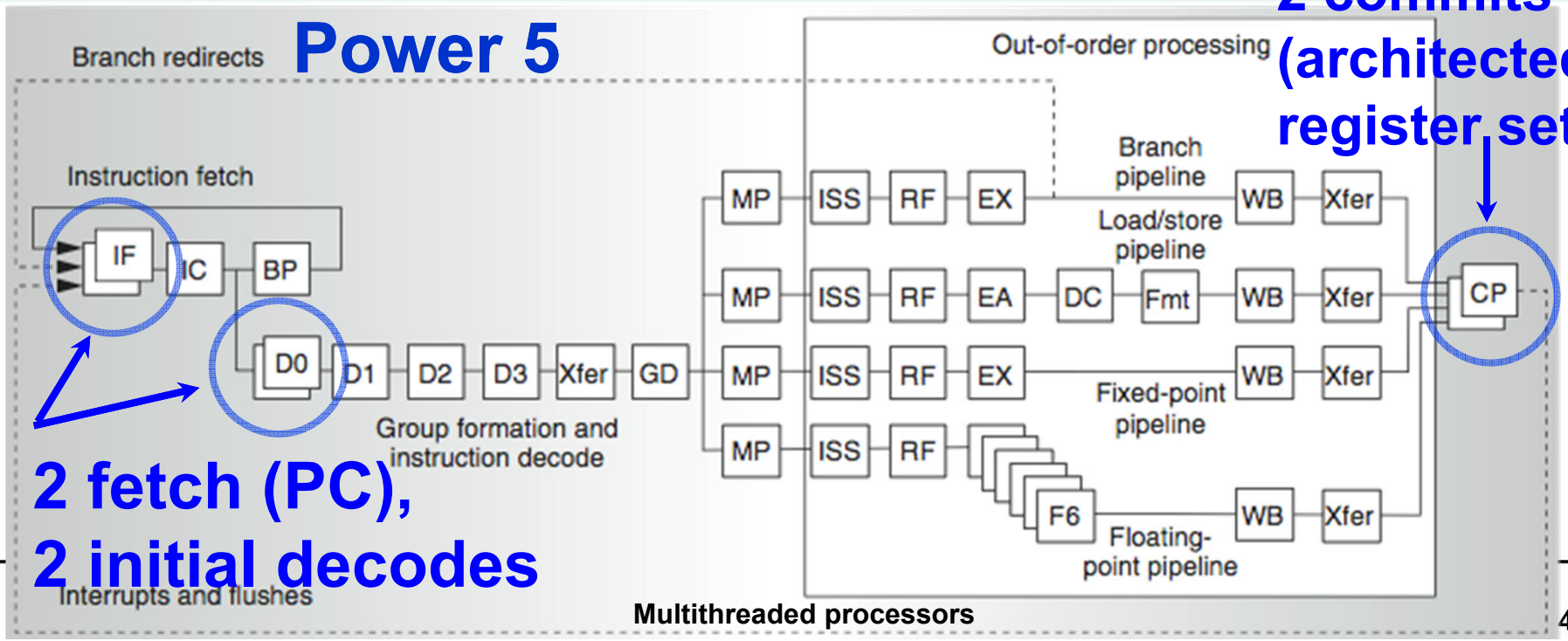




Power 4



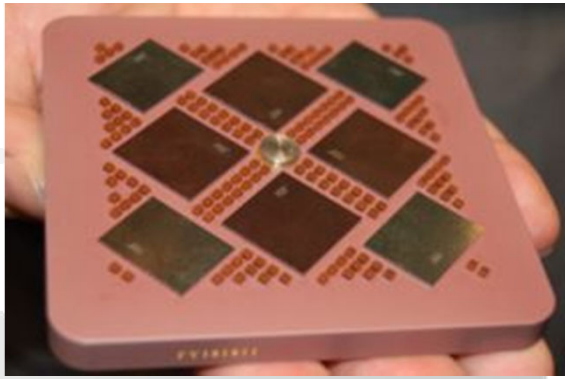
Power 5

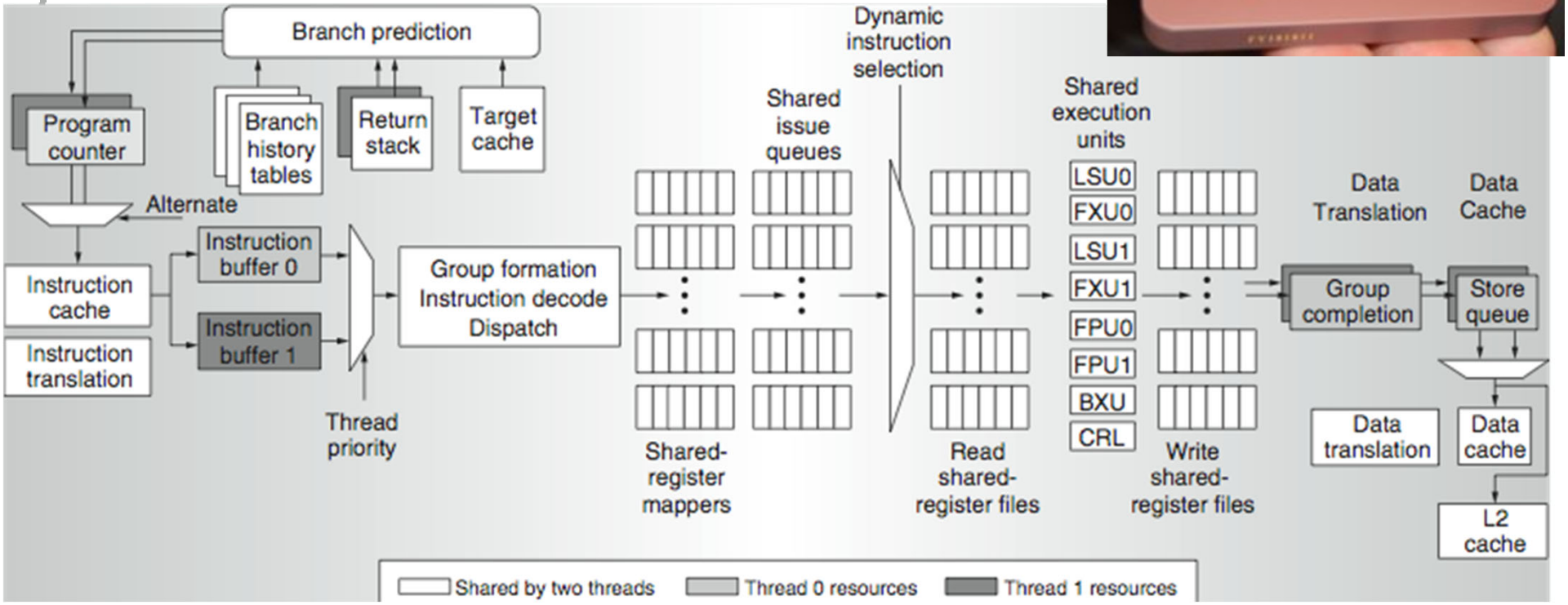


2 commits (architected register sets)

2 fetch (PC),
2 initial decodes

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

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

SMT Register File Capacity

- IBM Power8: 12 cores, 8T each, (32 FX+32FP) registers per thread. Consider FX only:
  - 8x32 = 256 physical registers minimum
  - Also need registers for inflight instructions
    - GCT (ROB) is 28x8=224 instructions x 75% = 168 registers
  - Total demand: 168+256 = 424 !!!
  - Very large, slow, power hungry
  - Instead: two-level RF for SMT8 mode:
    - 1st level (GPR) is two 124-entry register files
    - 2nd level (SAR) Software Architected Registers w/ 72-entries/thread
- More: B. Sinharoy et al., “IBM POWER8 processor core microarchitecture, ” IBM J R&D 59(1), Jan 2015.

IBM Power 10 E1080 (2021)

POWER10 Processor Chip

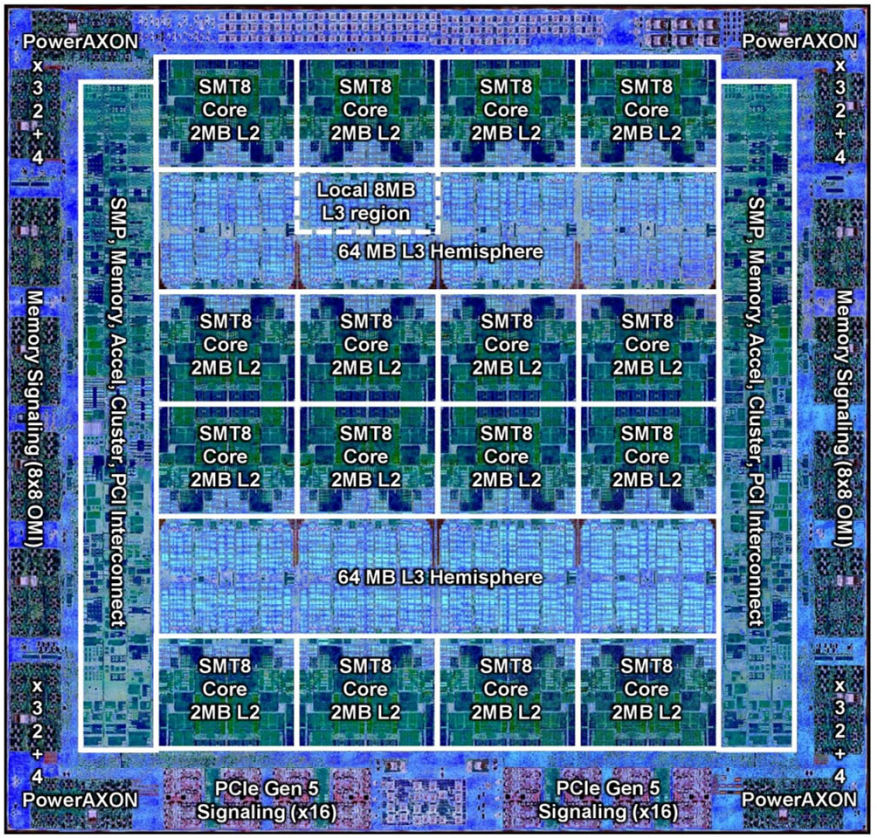
Technology and Packaging:
- 602mm² 7nm Samsung (18B devices)
- 18 layer metal stack, enhanced device
- Single-chip or Dual-chip sockets

Computational Capabilities:
- Up to 15 SMT8 Cores (2 MB L2 Cache / core)
  (Up to 120 simultaneous hardware threads)
- Up to 120 MB L3 cache (low latency NUCA mgmt)
- 3x energy efficiency relative to POWER9
- Enterprise thread strength optimizations
- AI and security focused ISA additions
- 2x general, 4x matrix SIMD relative to POWER9
- EA-tagged L1 cache, 4x MMU relative to POWER9

Open Memory Interface:
- 16 x8 at up to 32 GT/s (1 TB/s)
- Technology agnostic support: near/main/storage tiers
- Minimal (< 10ns latency) add vs DDR direct attach

PowerAXON Interface:
- 16 x8 at up to 32 GT/s (1 TB/s)
- SMP interconnect for up to 16 sockets
- OpenCAPI attach for memory, accelerators, I/O
- Integrated clustering (memory semantics)

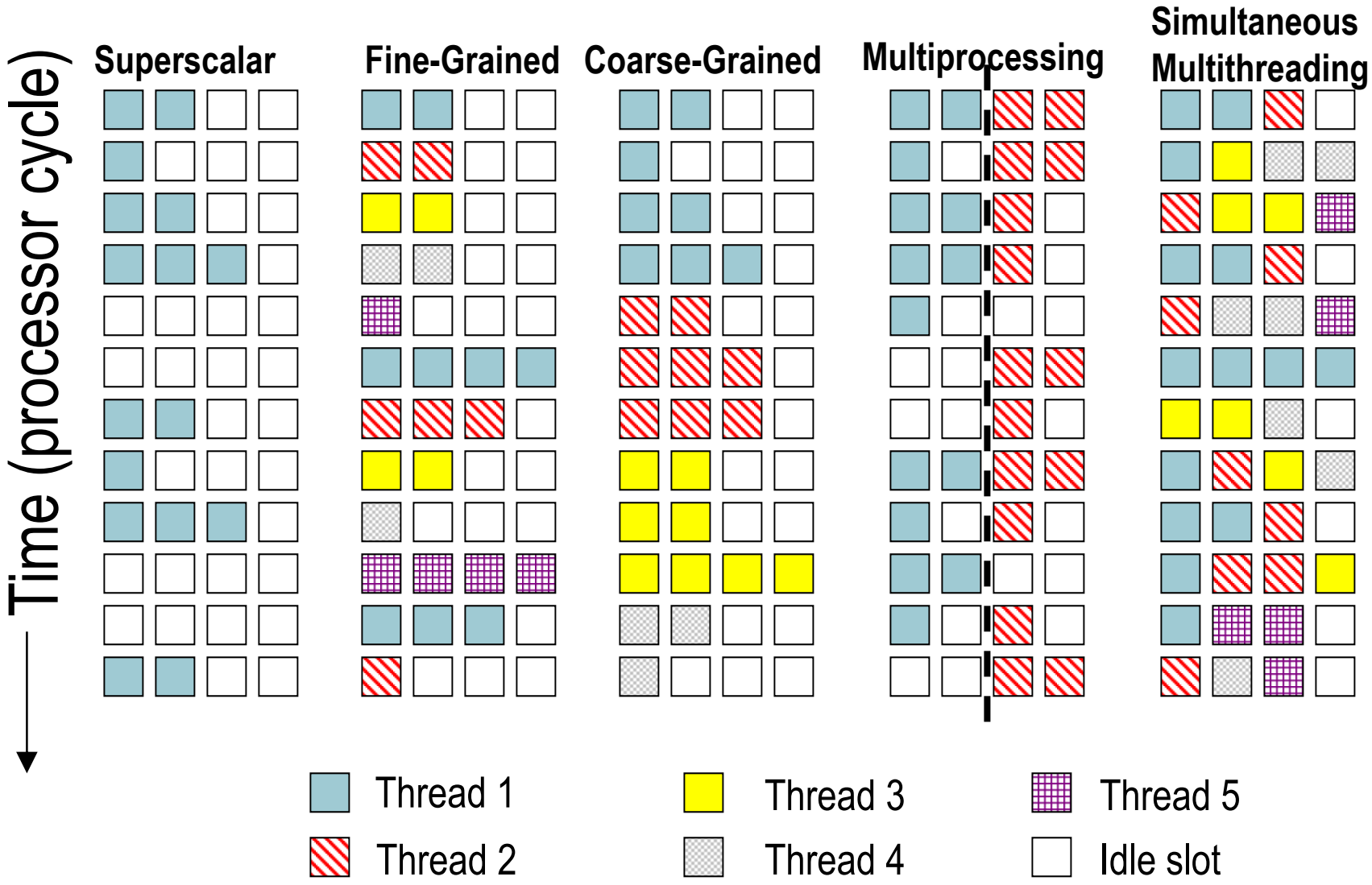
PCIe Gen 5 Interface:
- x64 / DCM at up to 32 GT/s



Die Photo courtesy of Samsung Foundry

- Can be configured by firmware as 30 SMT4 cores
  - but cannot be changed after manufacturing

Summary: Multithreaded Categories



Multithreading Summary

- Goal: increase throughput
  - Not latency
- Utilize execution resources by sharing among multiple threads:
  - Fine-grained, coarse-grained, simultaneous
- Usually some hybrid of fine-grained and SMT
  - Front-end is FG, core is SMT, back-end is FG
- Resource sharing
  - I cache, D cache, ALU – shared
  - Instruction Queue, ROB, Load queue, Store queue – partitioned vs. shared

What's Next to Learn in ESE 565

- How to be a successful computer designer and programmer of parallel processing systems in the 21st century
  - How to write a (multithreaded) program that can efficiently exploit all computing resources available in parallel systems such as
    - Multiple workstations connected by Internet
    - Multiple cores per Intel/ AMD host CPU in each workstation
    - Multiple Cell-like SIMD units per core
    - Multiple CUDA cores and Tensor units per GPU
    - Multiple Nvidia GPUs per workstation
  - How to design processors and interconnect buses to support parallel processing in multicore CPUs?
  - How are GPUs designed?
- Programming assignments with ISPC, Pthreads, OpenMP, MPI, and CUDA - 45%
- A system design/analysis/parallel programming project either of student's choice (to be approved by Instructor) or an HDL design project suggested by Instructor – 30%
- Midterm exam – 25%

Acknowledgements

- These slides contain material developed and copyright by:
  - Morgan Kauffmann (Elsevier, Inc.)
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - J. E. Smith (UWM)
  - Mikhail Dorojevets (SBU)