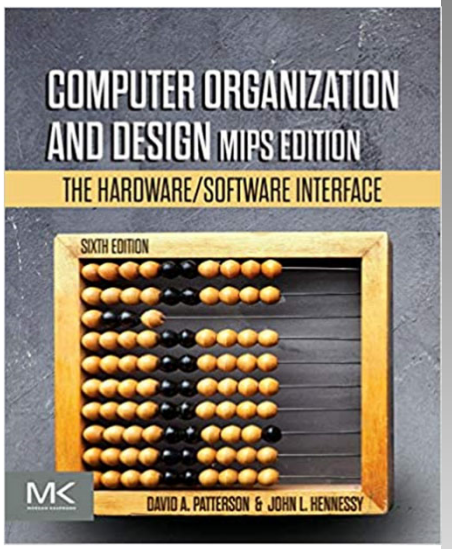
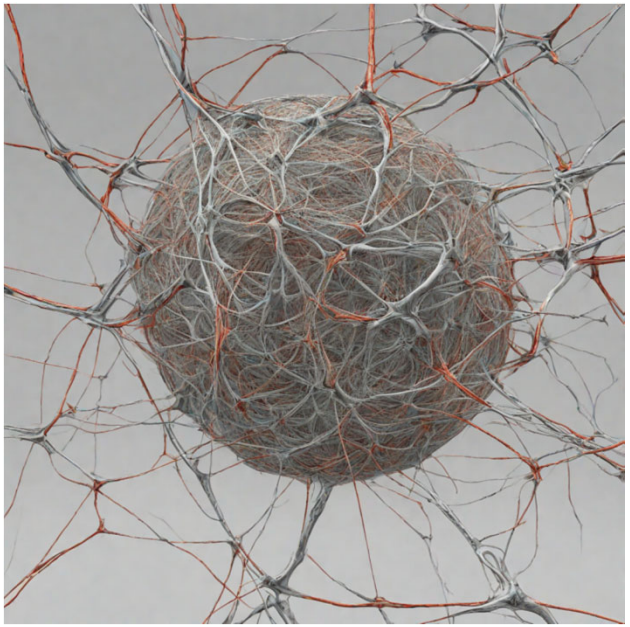


ESE 345 Computer Architecture

Thread-level Parallelism (TLP) : Multithreading



"multithreading" image created by SDXL text-to-image AI generative model 2023

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

Example: Thread creation in OpenMP parallel regions

- You create threads in OpenMP* with the parallel construct.
- For example, to create a region:

Each thread executes a copy of the code within the structured block

```
double A[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
        int ID = omp_get_thread_num();
        foo(ID,A);
}
```

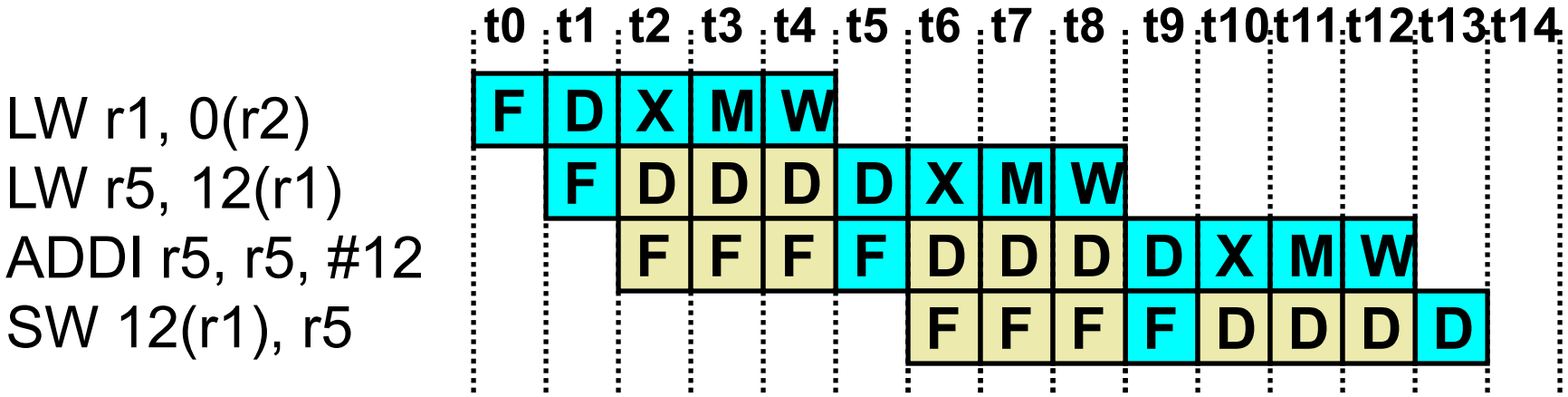
Globally declared var (outside the scope of #pragma omp parallel)

Runtime function to request a certain number of threads

Runtime function returning a thread ID

- Each thread calls foo(ID,A) for ID = 0 to 3

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)***

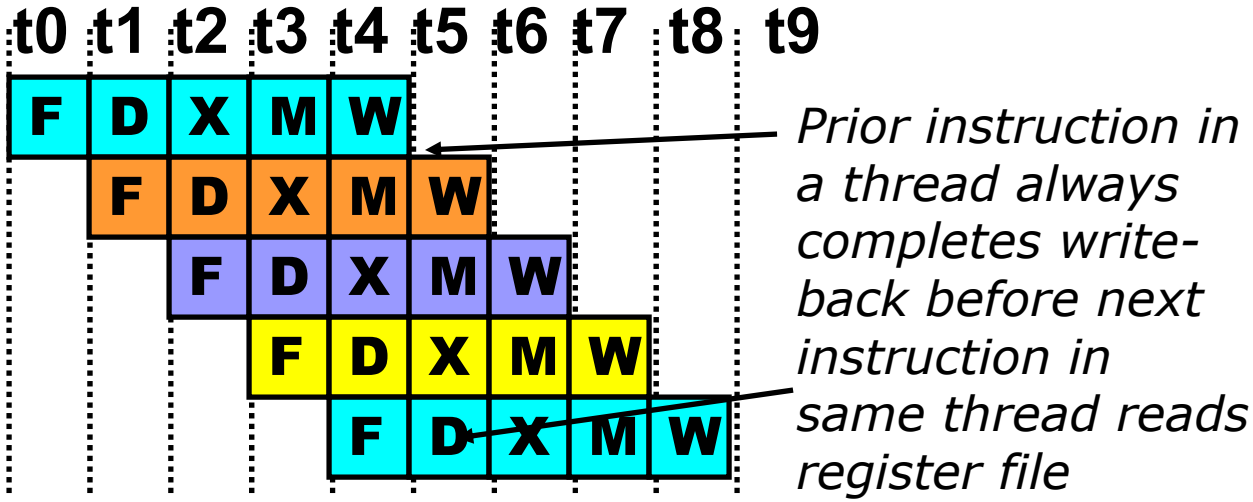
Multithreading

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)

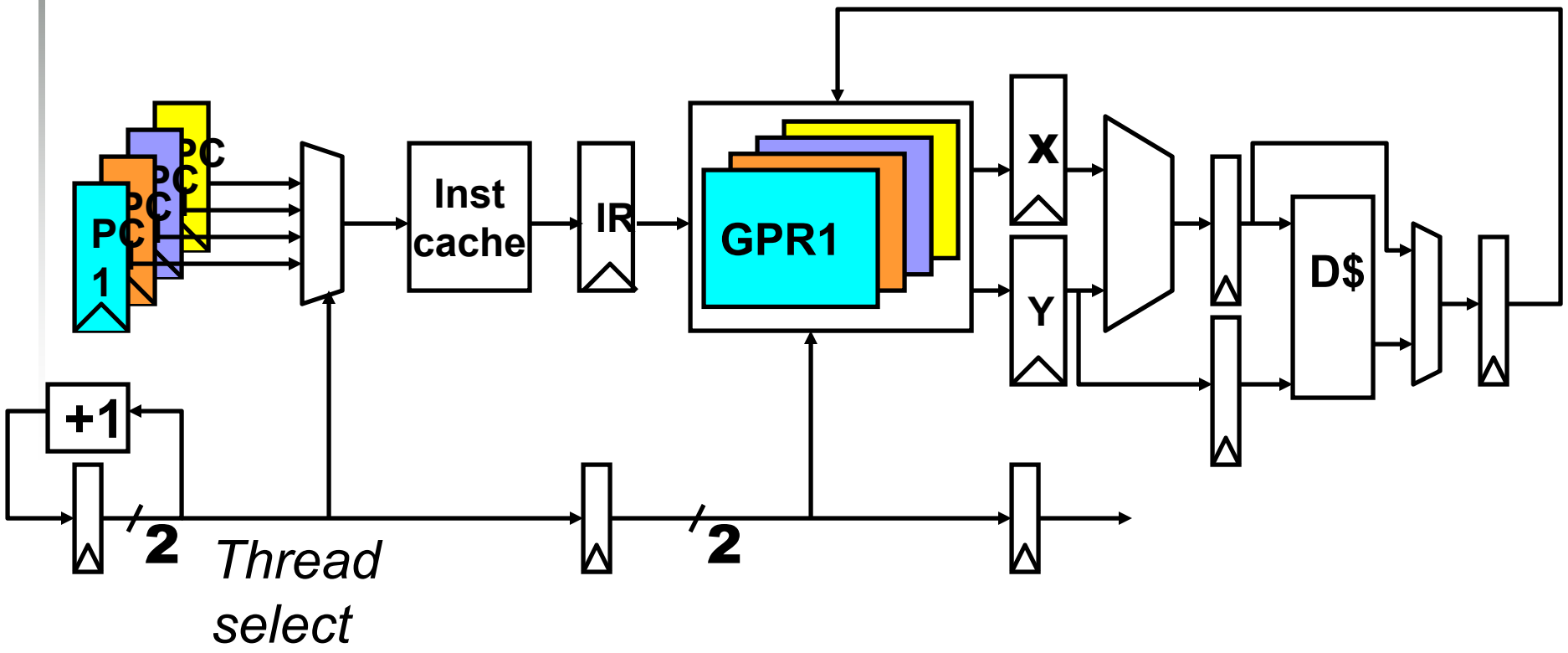


CDC 6600 Peripheral Processors

(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

Simple Multithreaded Pipeline



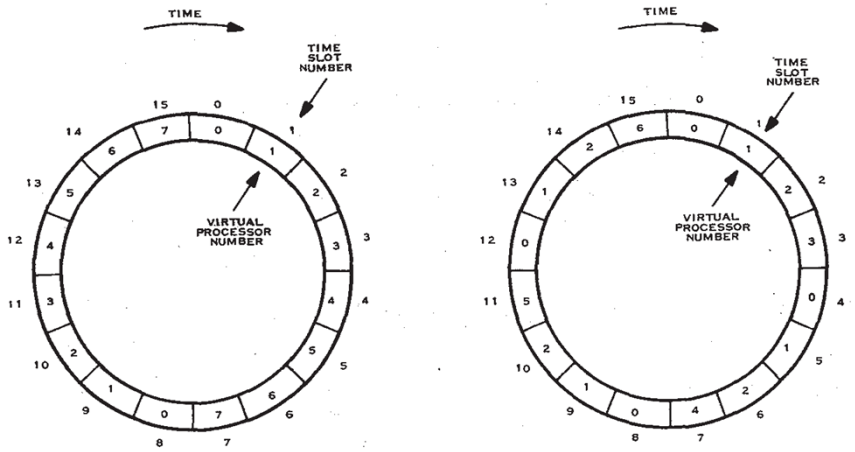
- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

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)

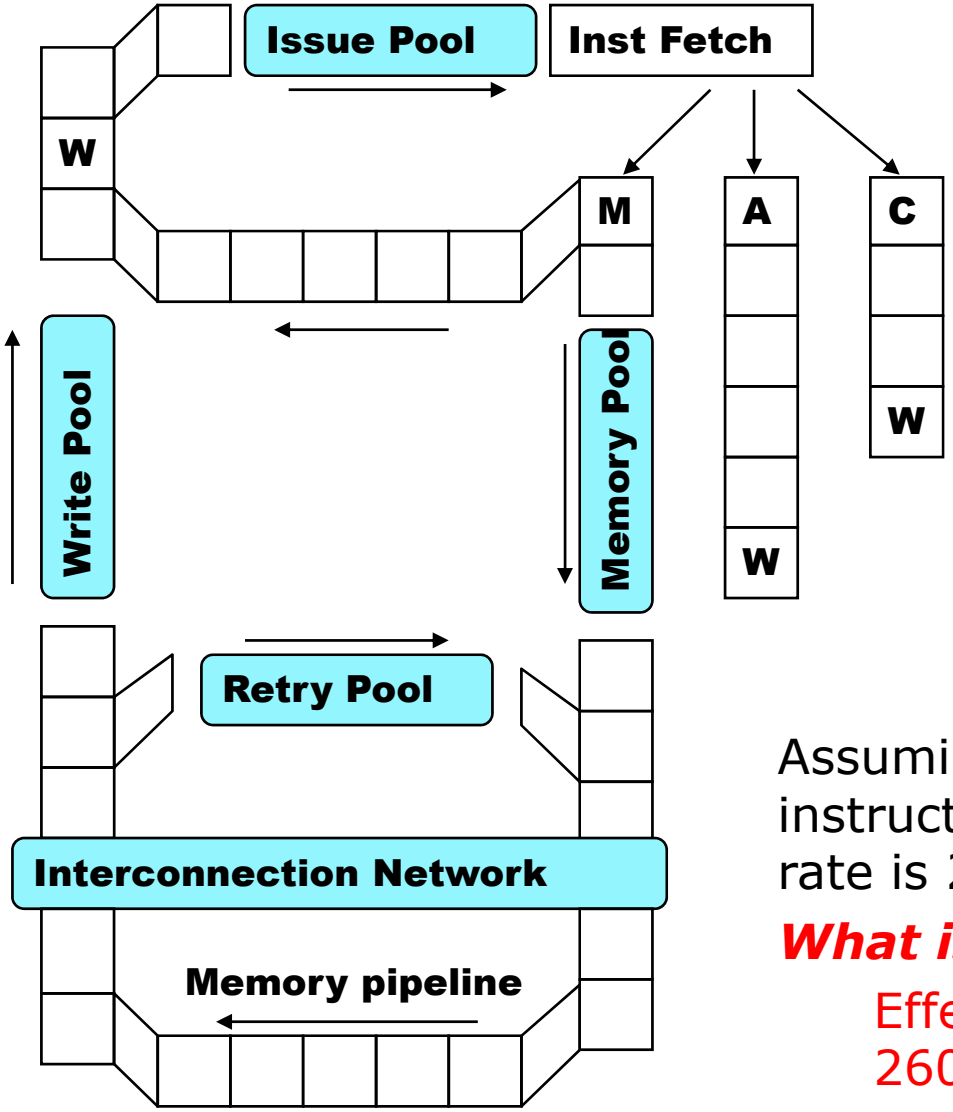
First commercial machine to use hardware threading in main CPU

- 120 threads per processor
- 10 MHz clock rate
- Up to 8 processors
- 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

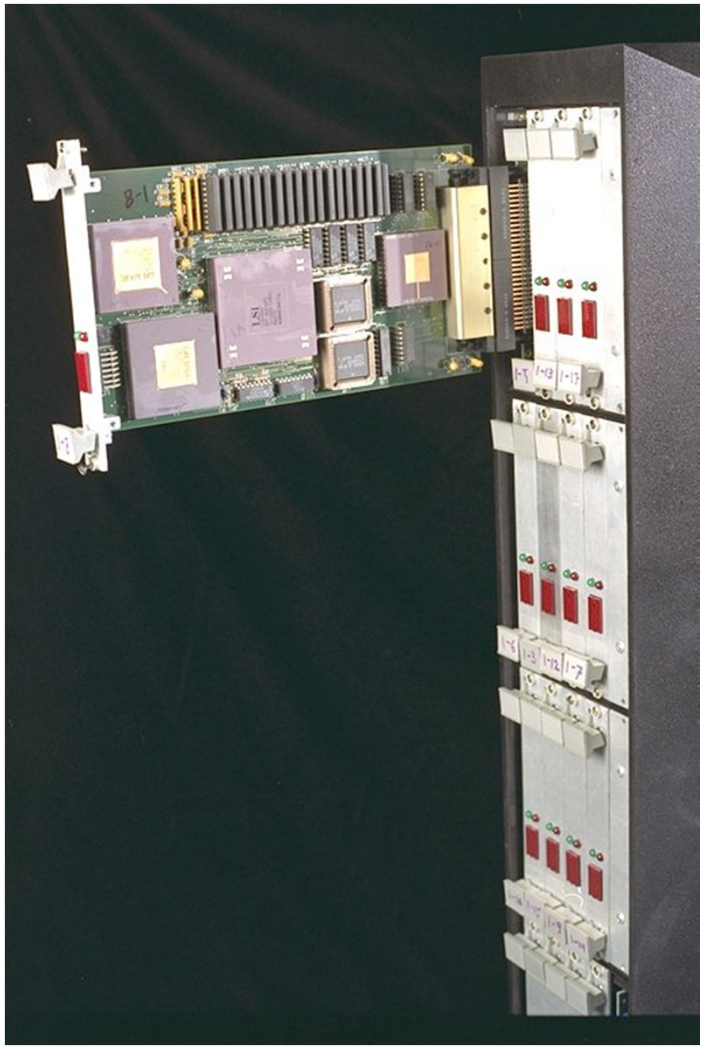
Tera MTA designed for supercomputing applications with large data sets and low locality

- 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)



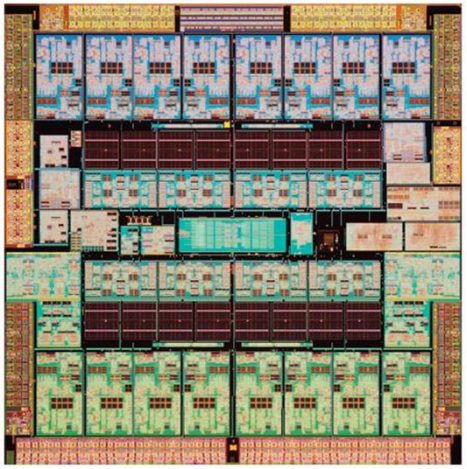
- Modified SPARC chips
  - register windows hold different thread contexts
- Up to four threads per node
- Thread switch on local cache miss

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

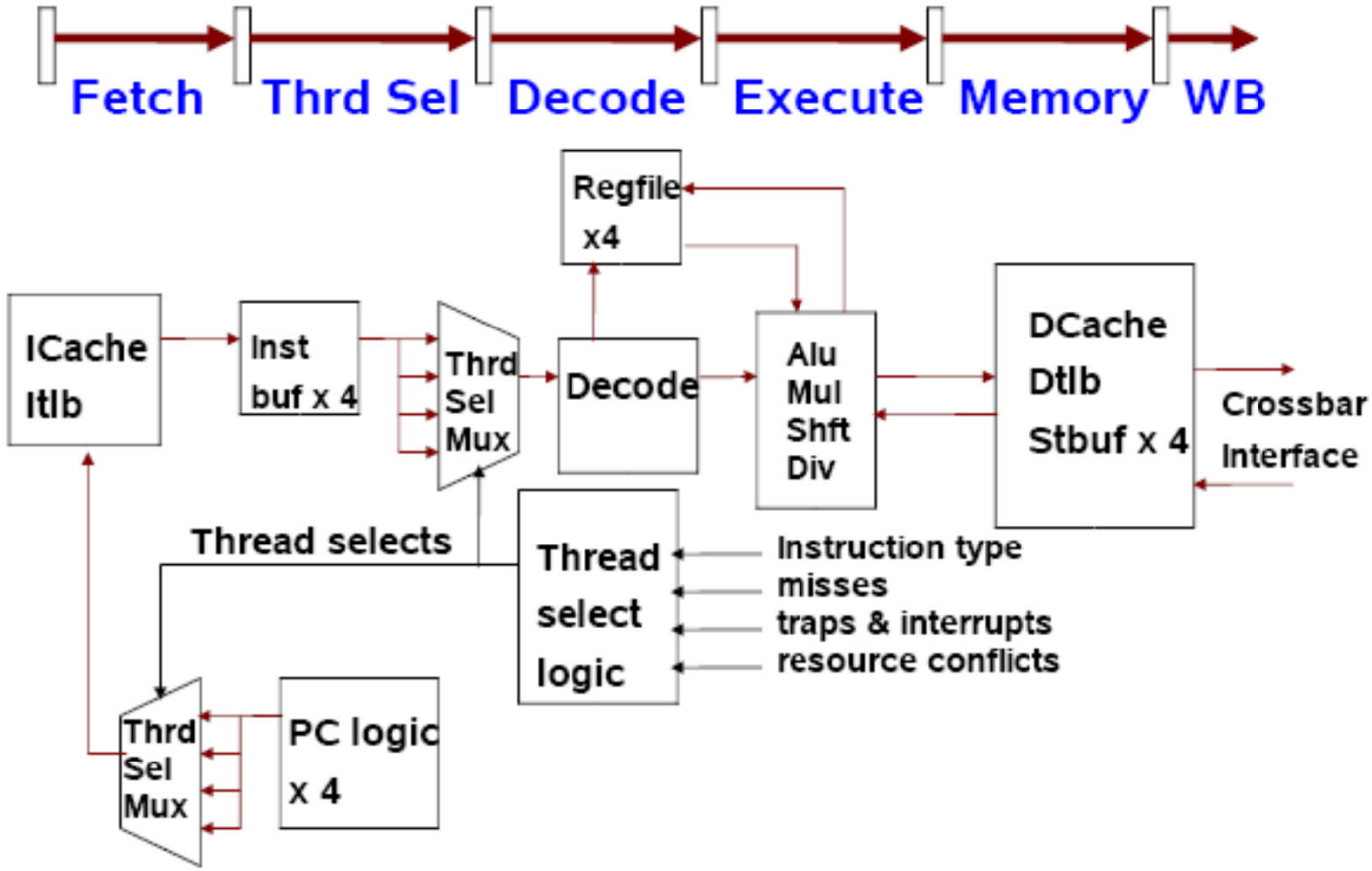
Oracle/Sun Niagara processors

- 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 Pipeline

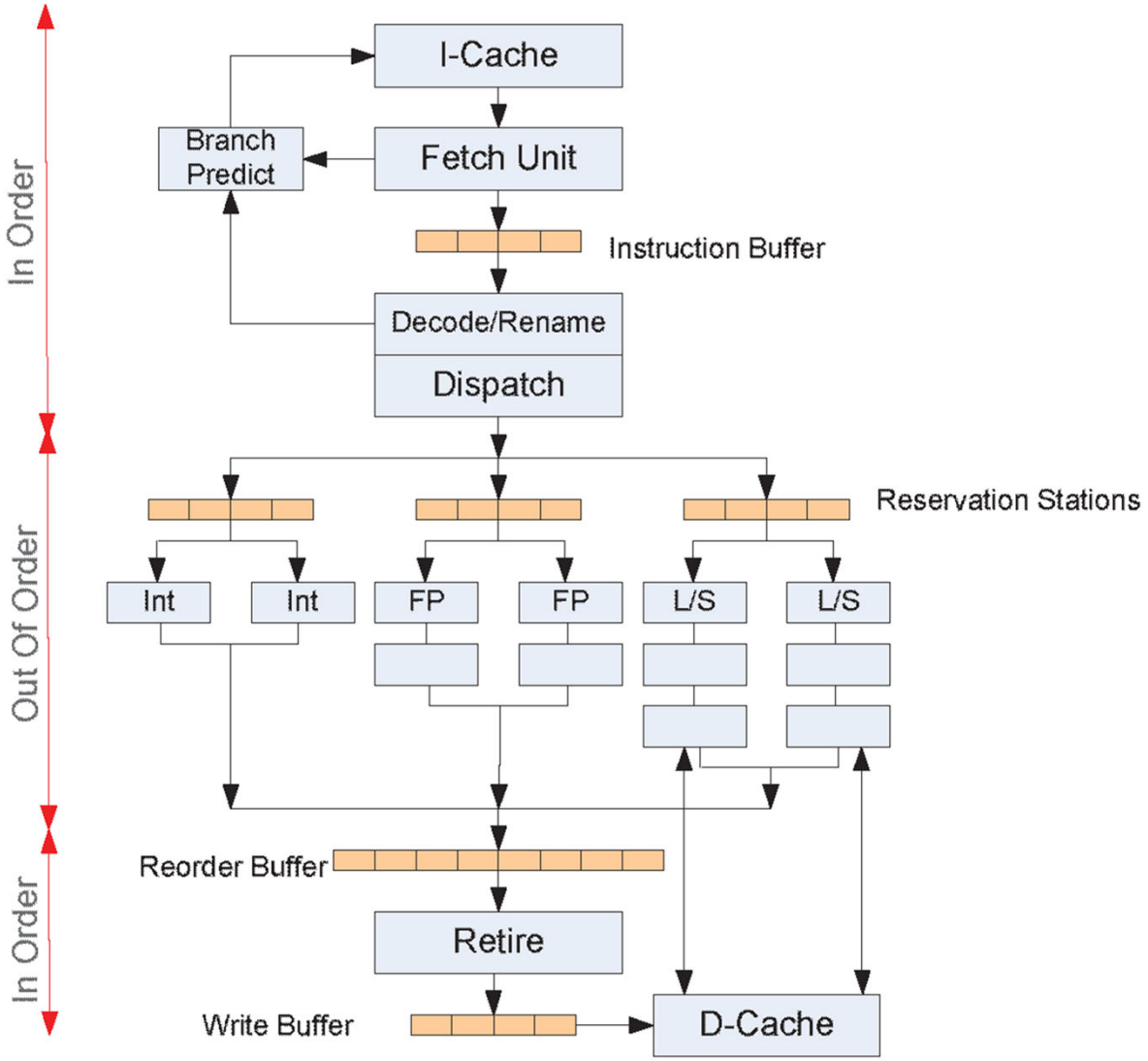


- 6-stage pipeline
- Fine-grained multithreading

Simultaneous Multithreading (SMT) for OoO Superscalars

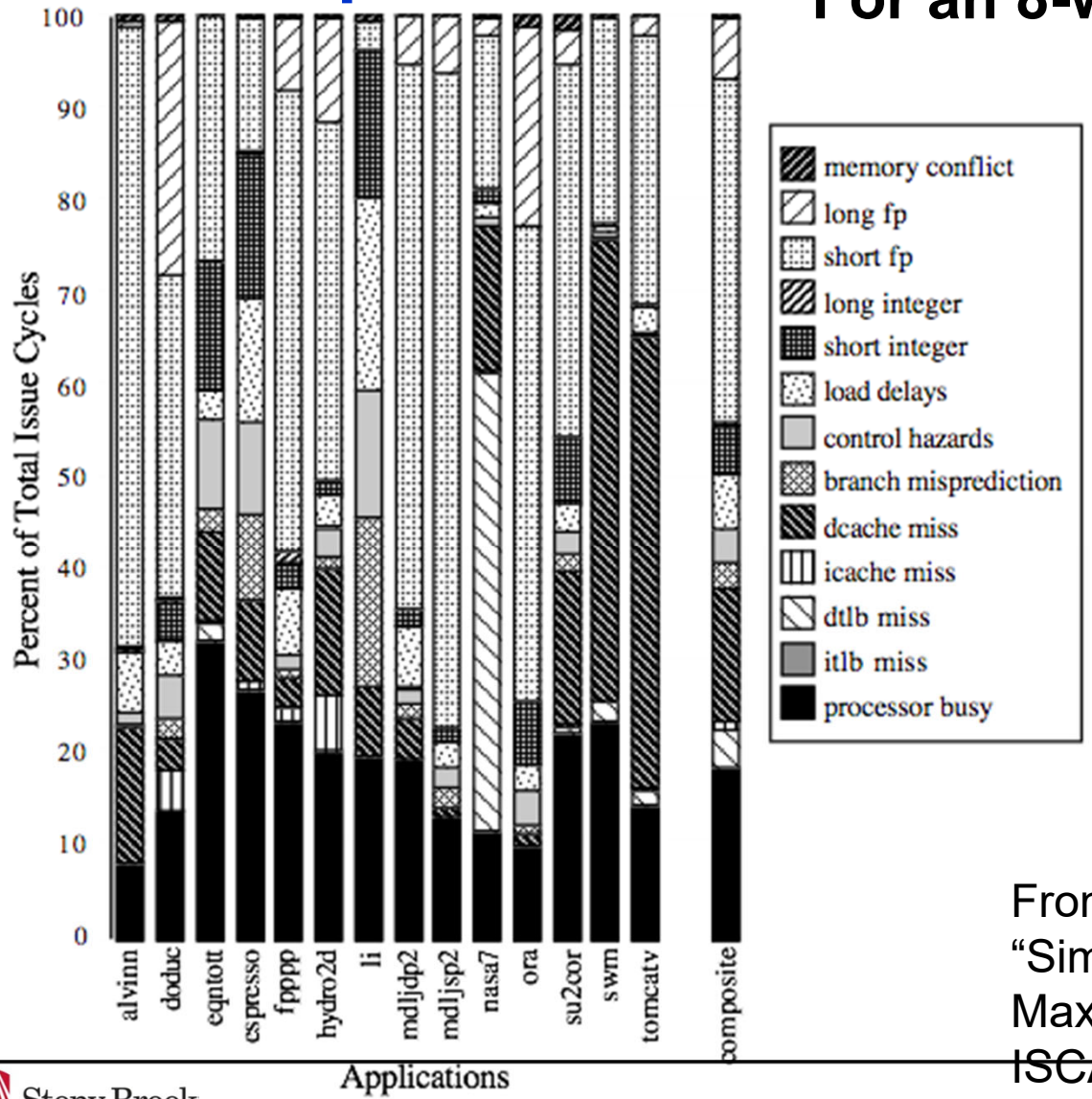
- Techniques presented so far have all been "**vertical**" multithreading where each pipeline stage works on **one thread at a time**
- **SMT** uses fine-grain control already present inside an OoO superscalar to allow **instructions from multiple threads to enter execution on same clock cycle.** Gives better utilization of machine resources.
- Examples: Intel P4, Intel Nehalem, IBM Power 5/6/7/8, Alpha EV8/21464, AMD Zen

A Modern Superscalar Out-of-Order Processor



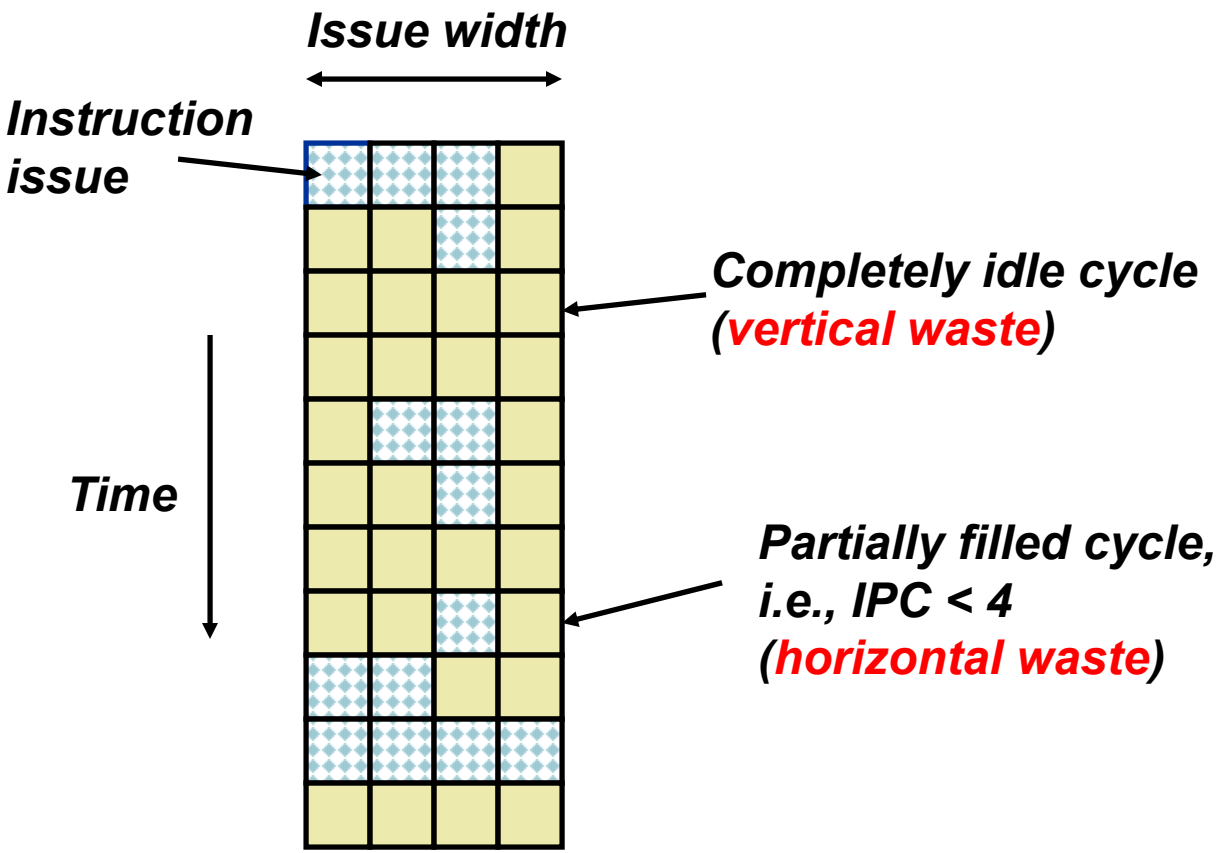
For most apps, most execution units lie idle in an OoO superscalar

For an 8-way superscalar.

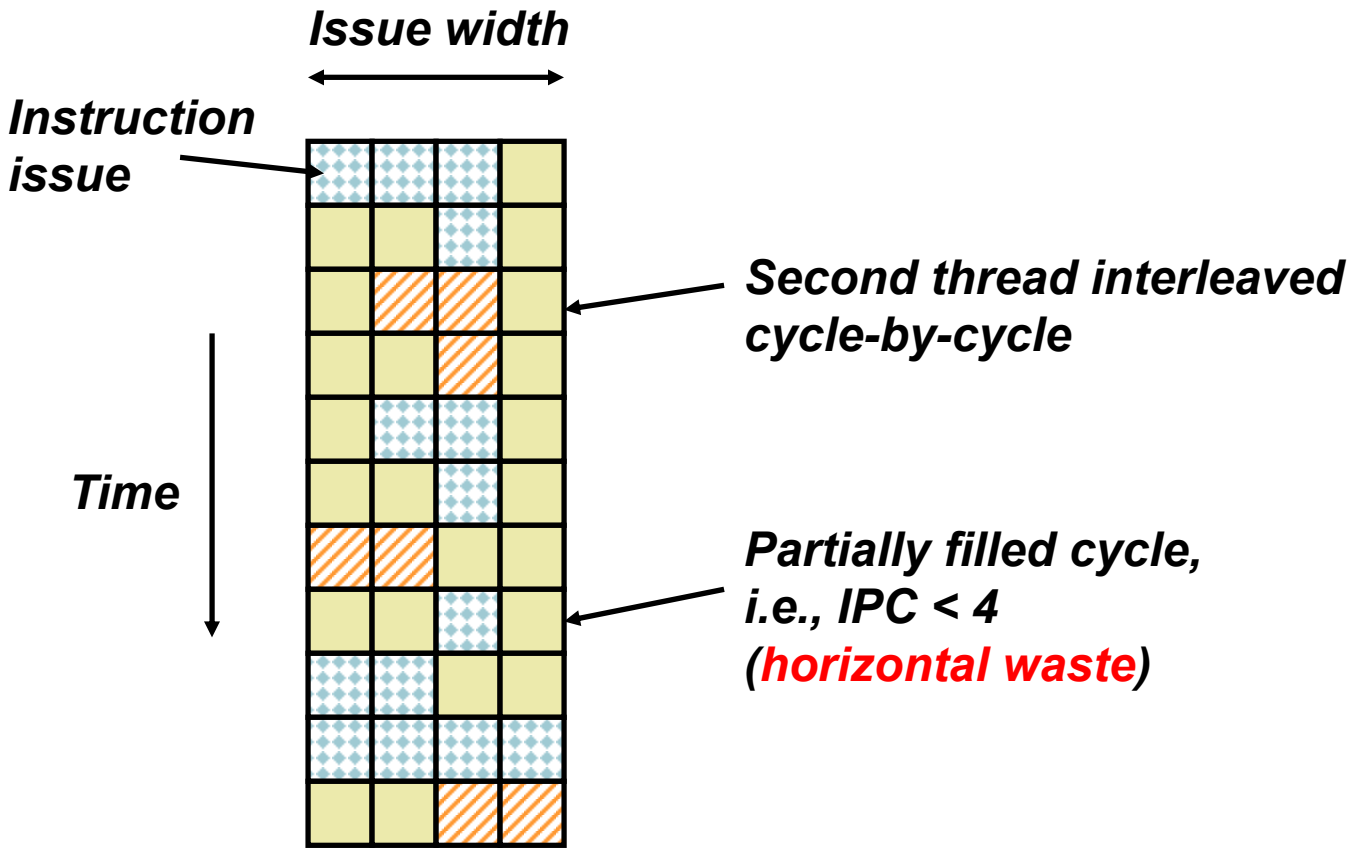


From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism", ISCA 1995.

Superscalar Machine Efficiency

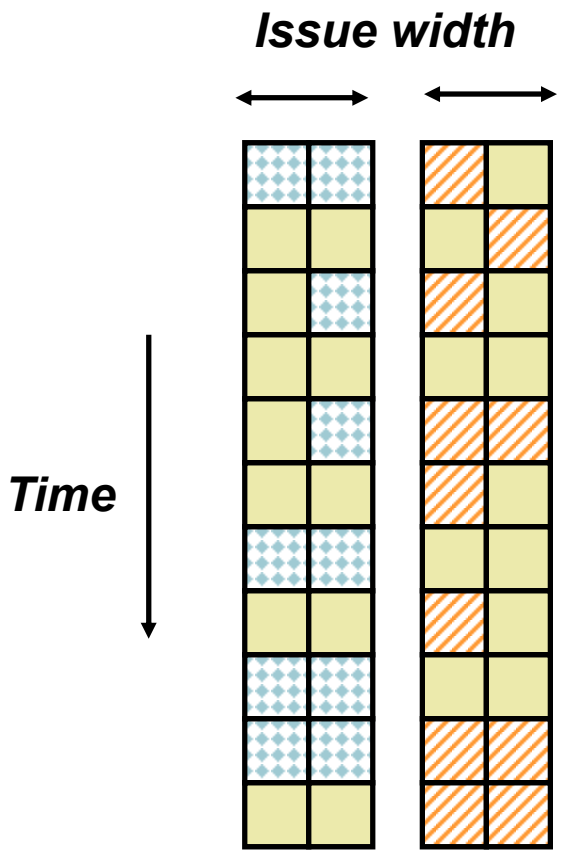


Vertical Multithreading



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

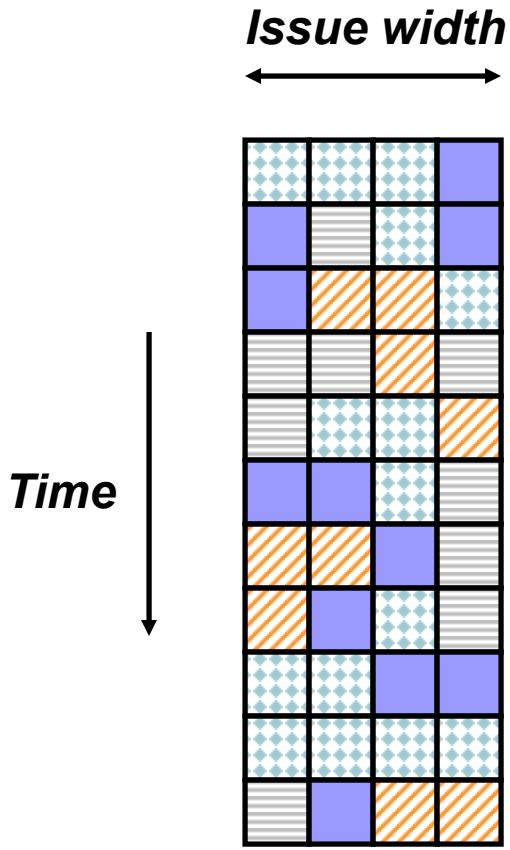
Chip Multiprocessing (CMP)



- What is the effect of splitting into multiple processors?
  - 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

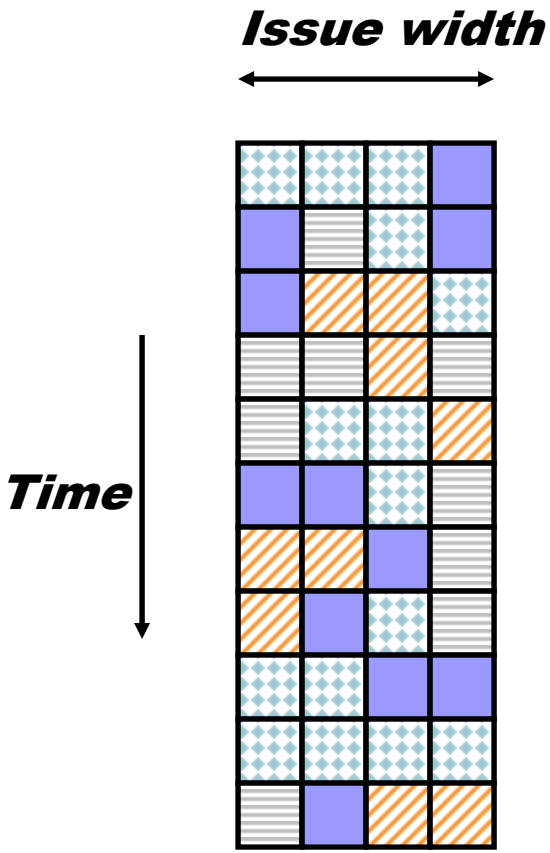
O-o-O Simultaneous Multithreading

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

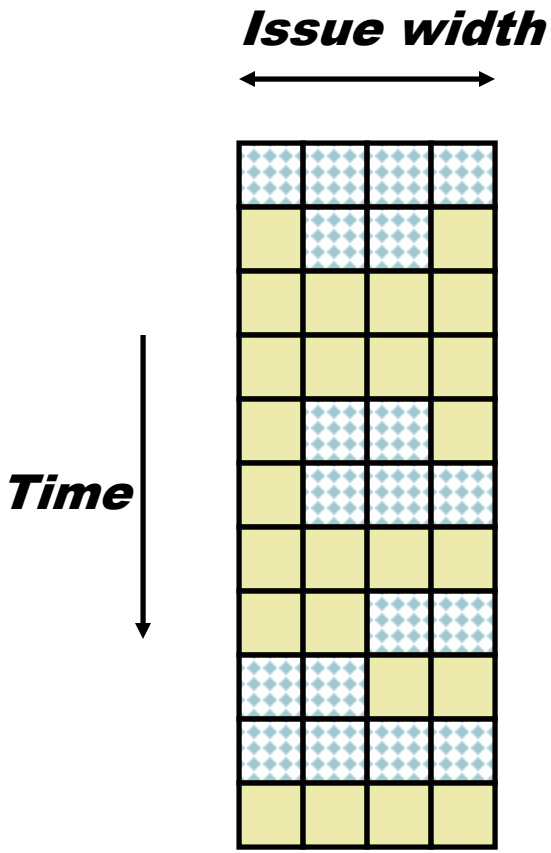
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required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

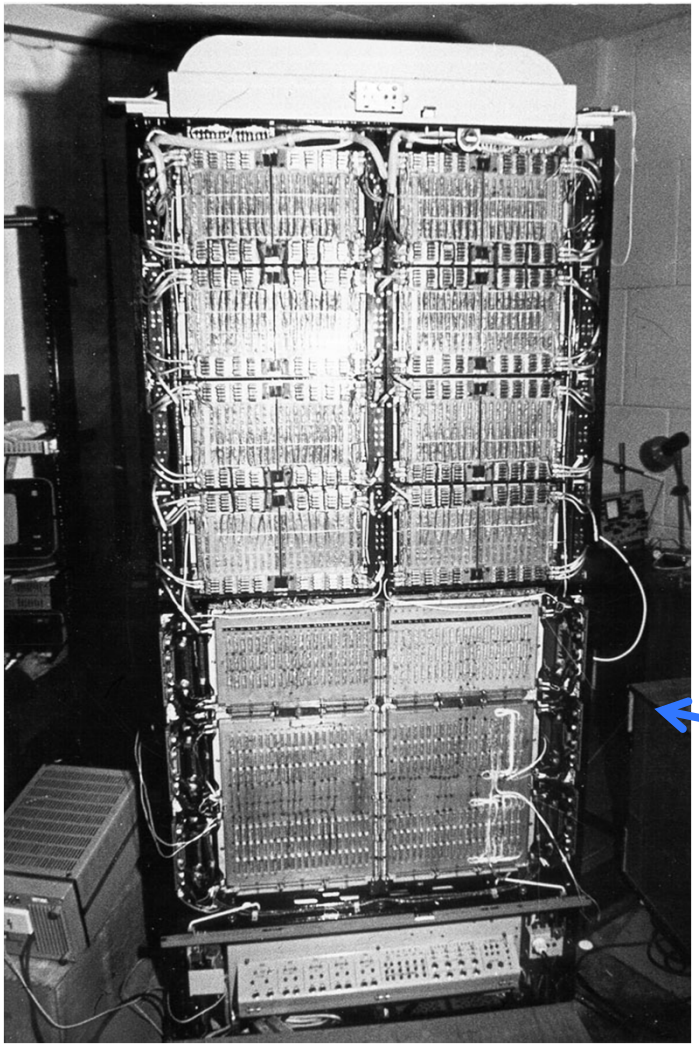


For regions with low thread level parallelism (TLP) entire machine width is available for instruction level parallelism (ILP)

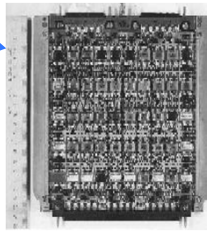


MARS-M: Truly First SMT System – Russia 1988

my first computer built well before Tulsen's SMT paper, IBM, and Intel computers!



- **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



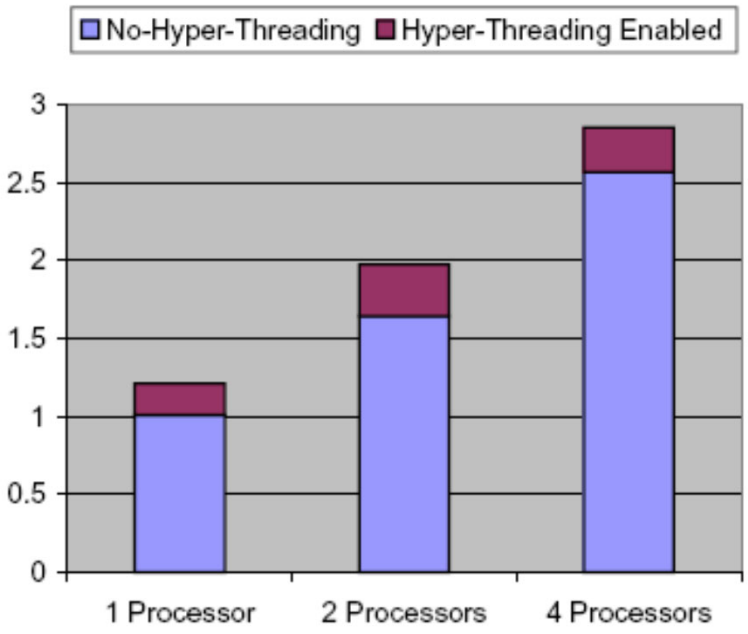
Dorozhevets M.N. and Wolcott P. The El'brus-3 and MARS-M: Recent Advances in Russian High-performance Computing, The Journal of Supercomputing, USA, 6, pp. 5-48, 1992

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

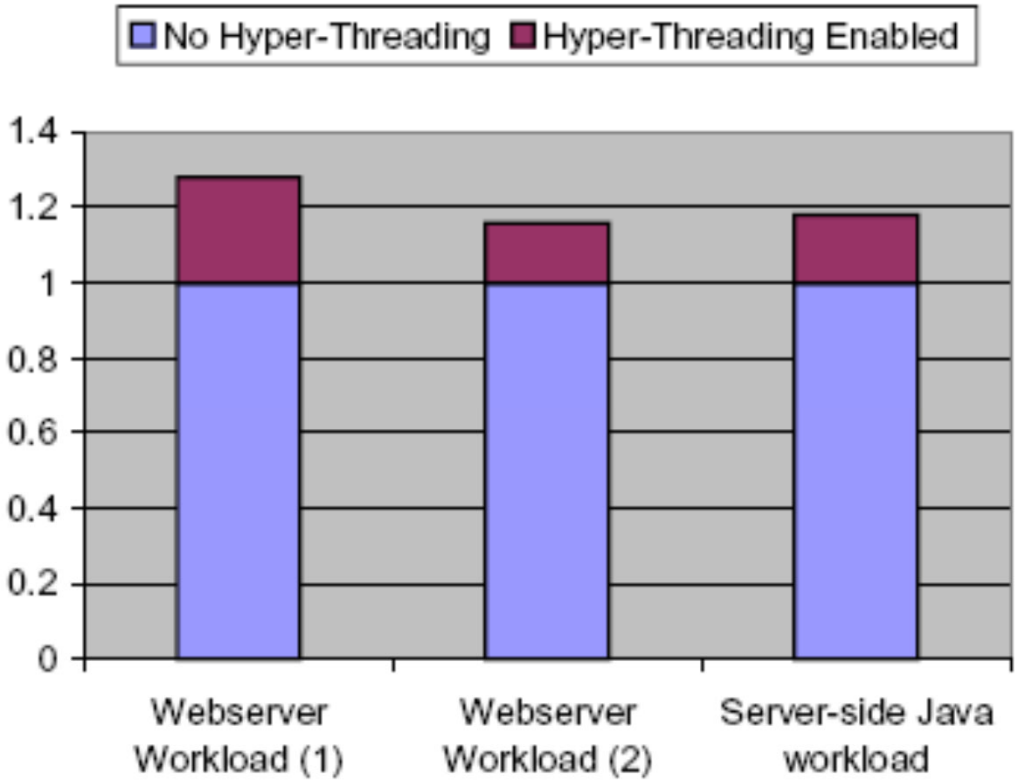
Performance

- 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



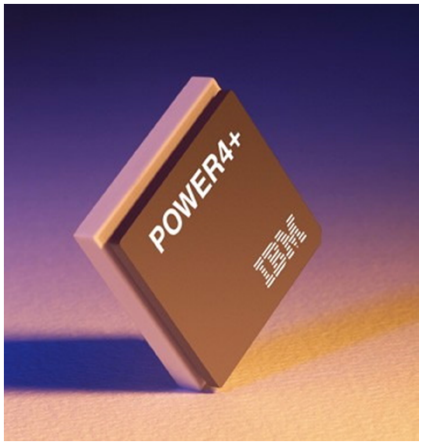
Performance

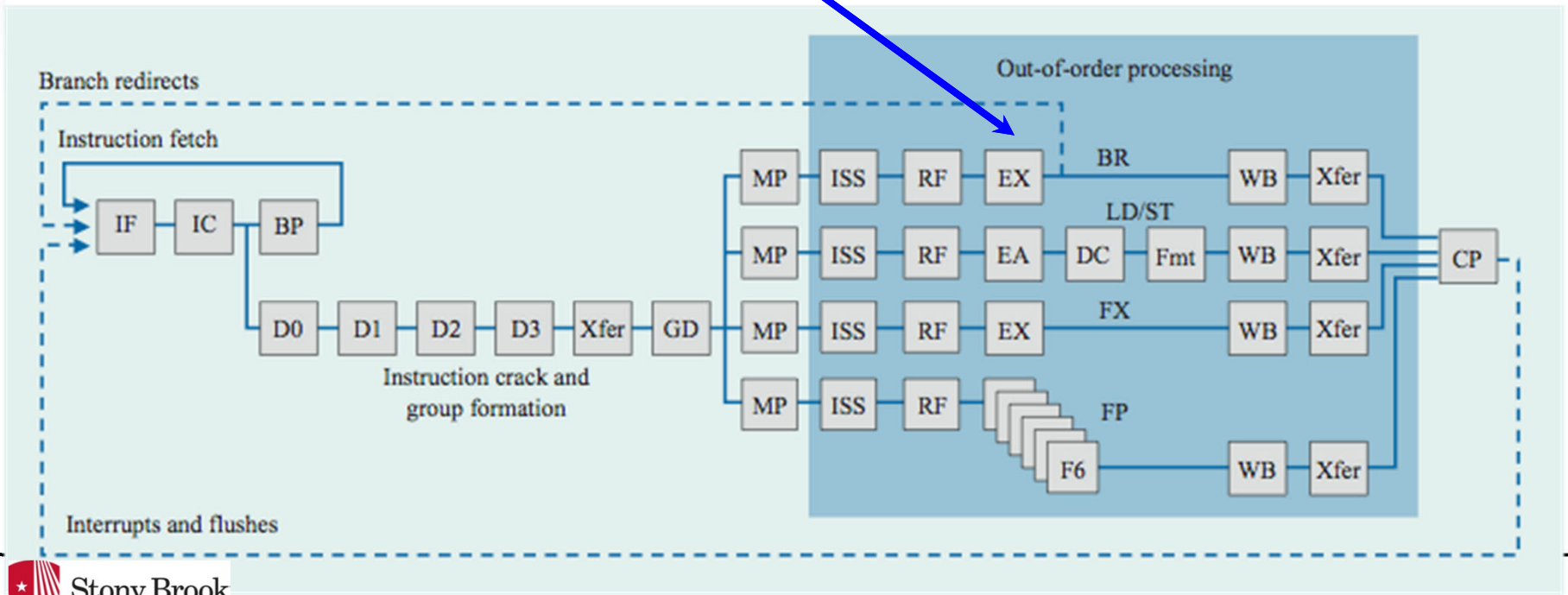
- Web server apps



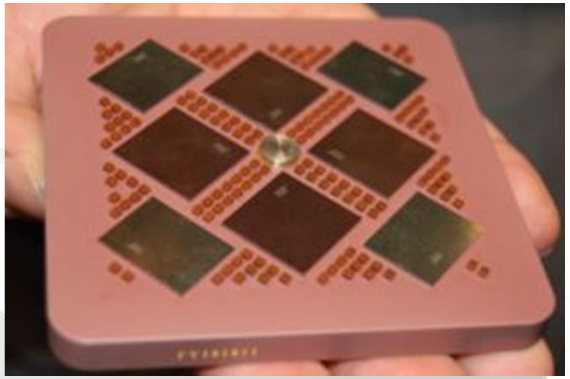
IBM Power 4

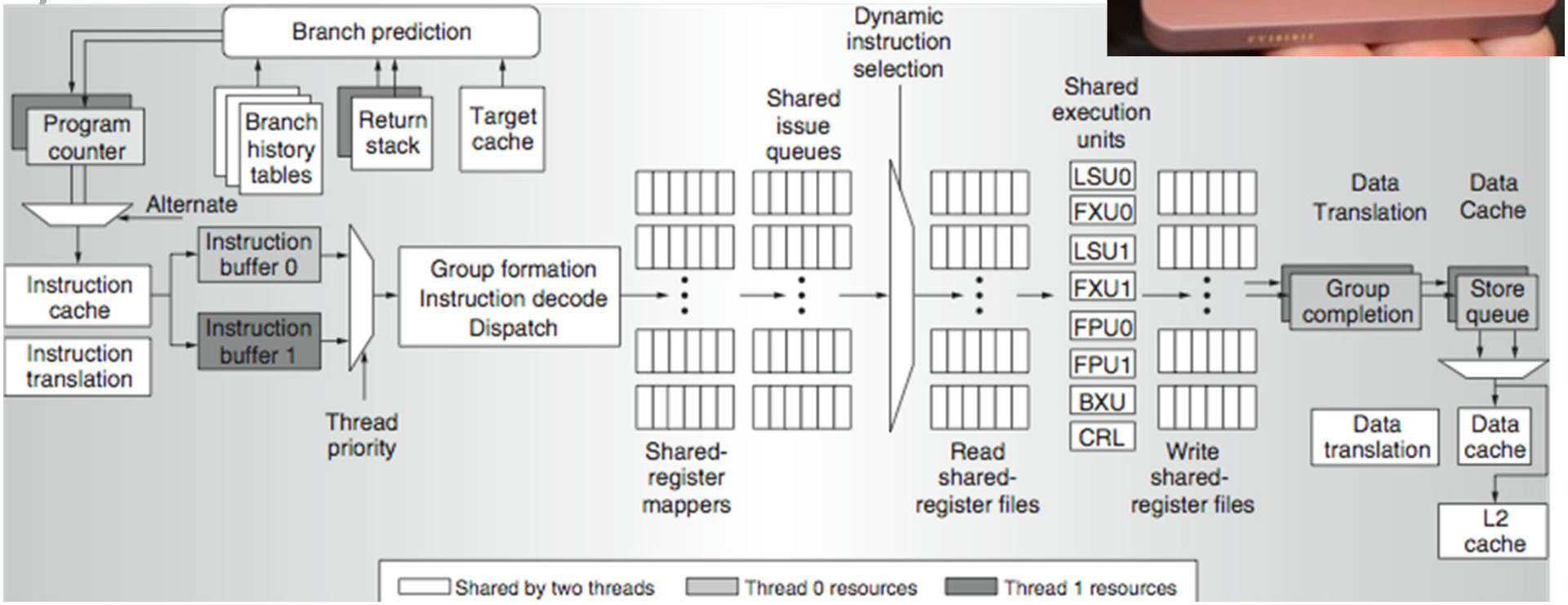
Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.





Dual-threaded IBM Power 5



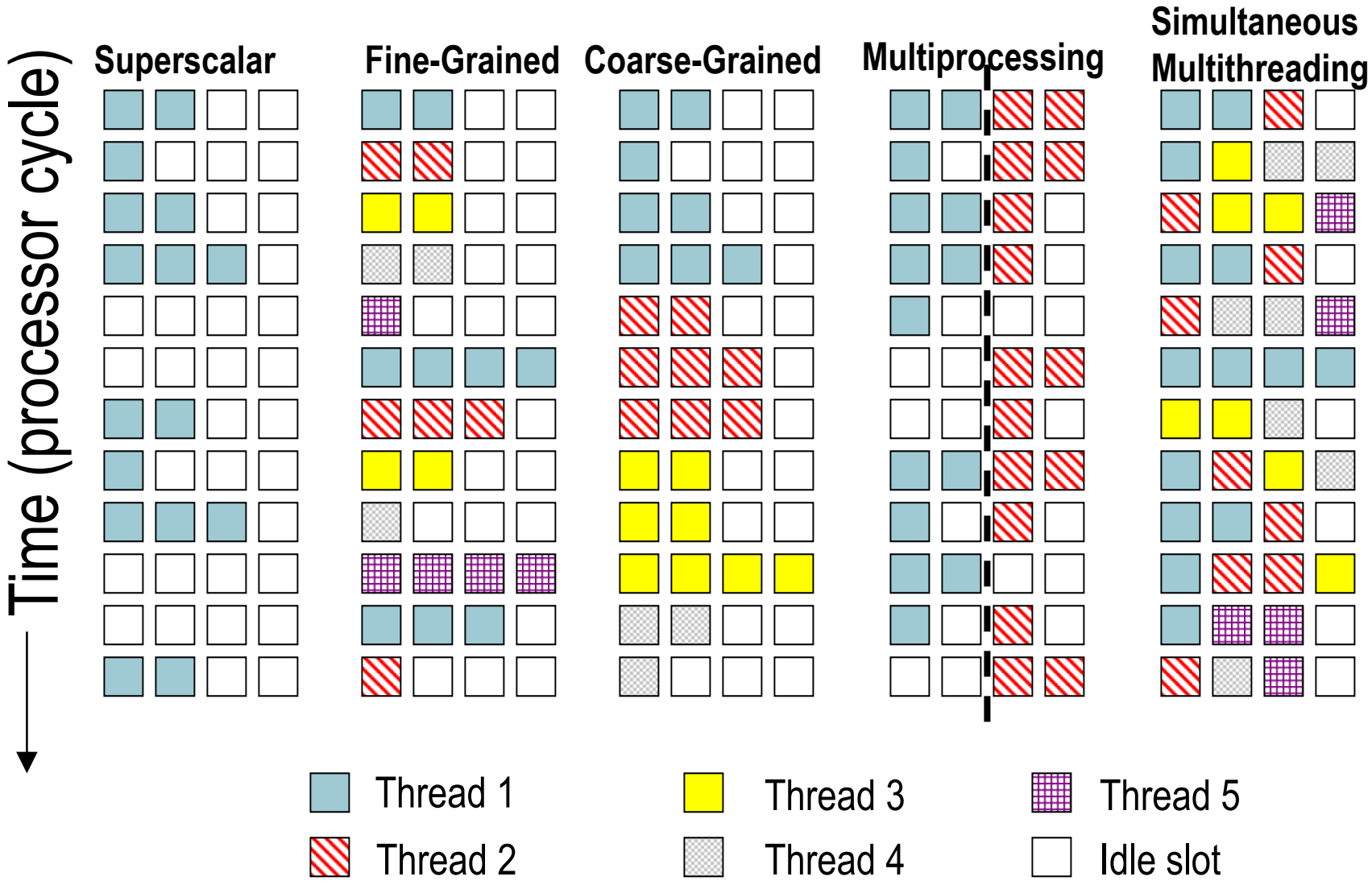


Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

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

Summary: Multithreaded Categories



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)
  - Mikhail Dorojevets (SBU)