ECE 563
Advanced Computer Architecture

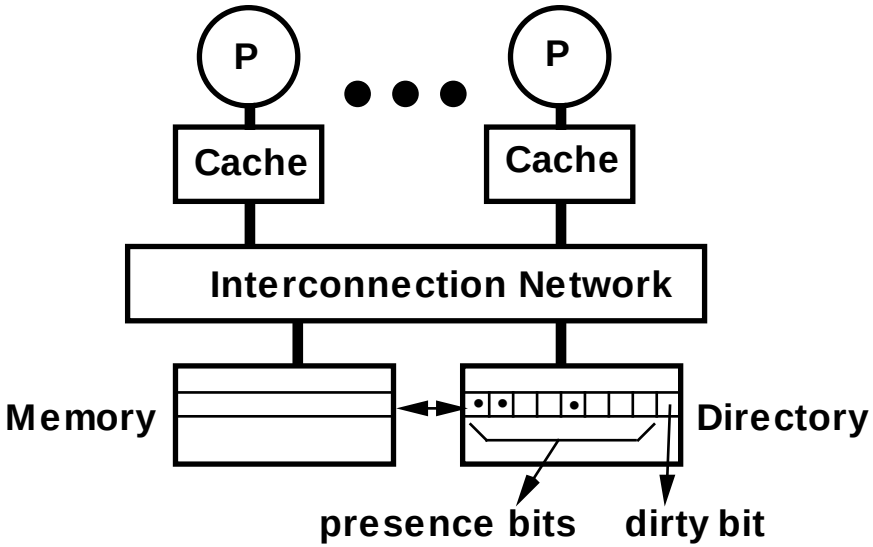
Fall 2009

Lecture 12: Multithreading

Reading list

- Simultaneous Multithreading: Maximizing On-Chip Parallelism by Tullsen, Eggers, and Levy
- Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor by Tullsen, Eggers, Emer, Levy, Lo, and Stamm
- POWR5 System Microarchitecture by Sinharoy et. Al. (IBM Journals on Research & Development)
- Hyper-Threading Technology Architecture and Microarchitecture by Marr et. Al. (Intel)

Recap: Directory Coherence Protocols



- k processors.
- With each cache-block in memory:
  k presence-bits, 1 dirty-bit
- With each cache-block in cache:
  1 valid bit, and 1 dirty (owner) bit

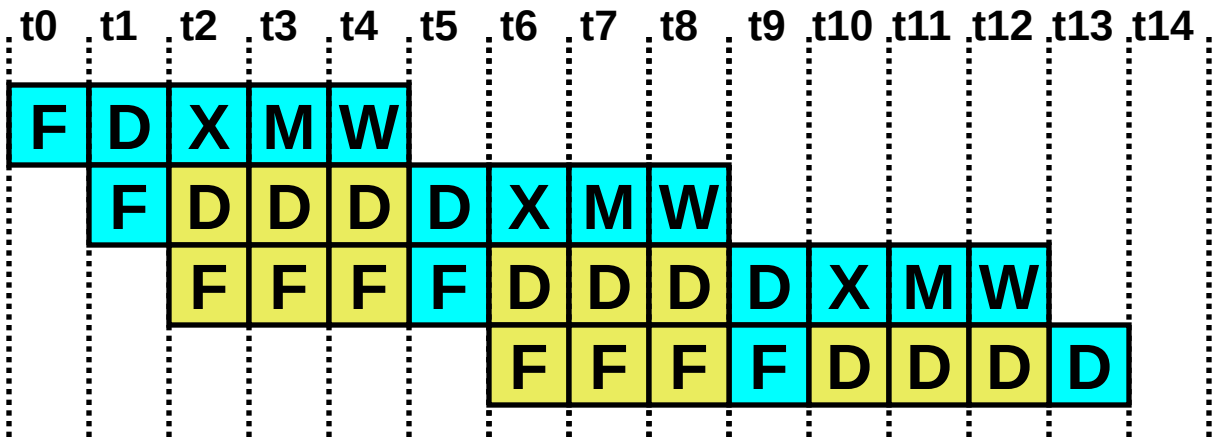
- **Scale to larger numbers of processors by replacing snoopy broadcast with point-point messages**
- **Requires additional directory storage**
- **Usually longer latency than snoopy protocols**
- **Often combined with snooping**
  - **Snoop within small cluster of processors, use directory between clusters**

Multithreading

- Difficult to continue to extract ILP from a single thread
- 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 previous**

What can be done to cope with this?

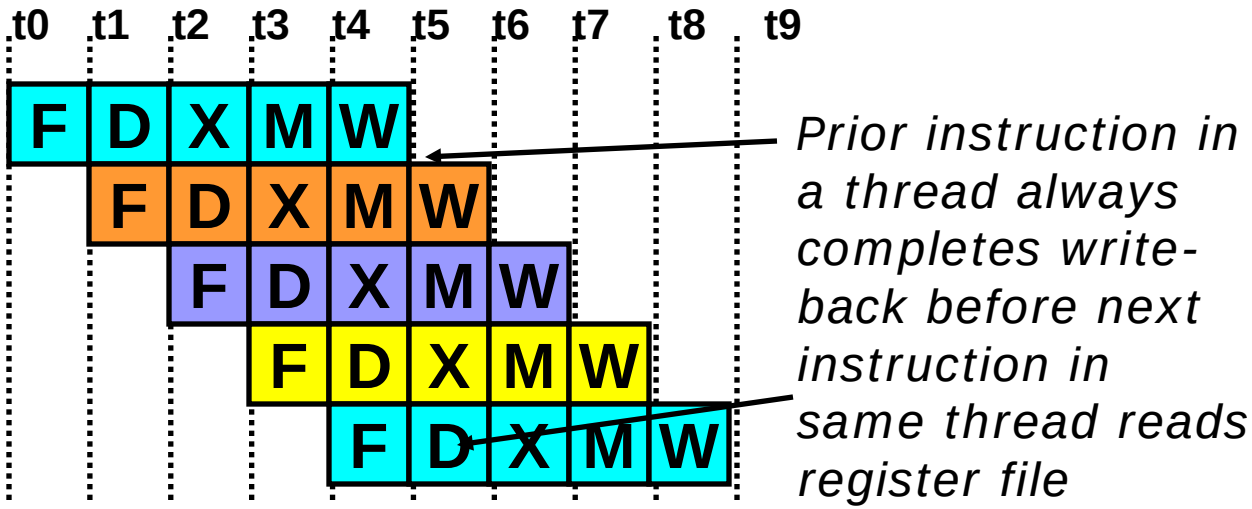
Multithreading

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)



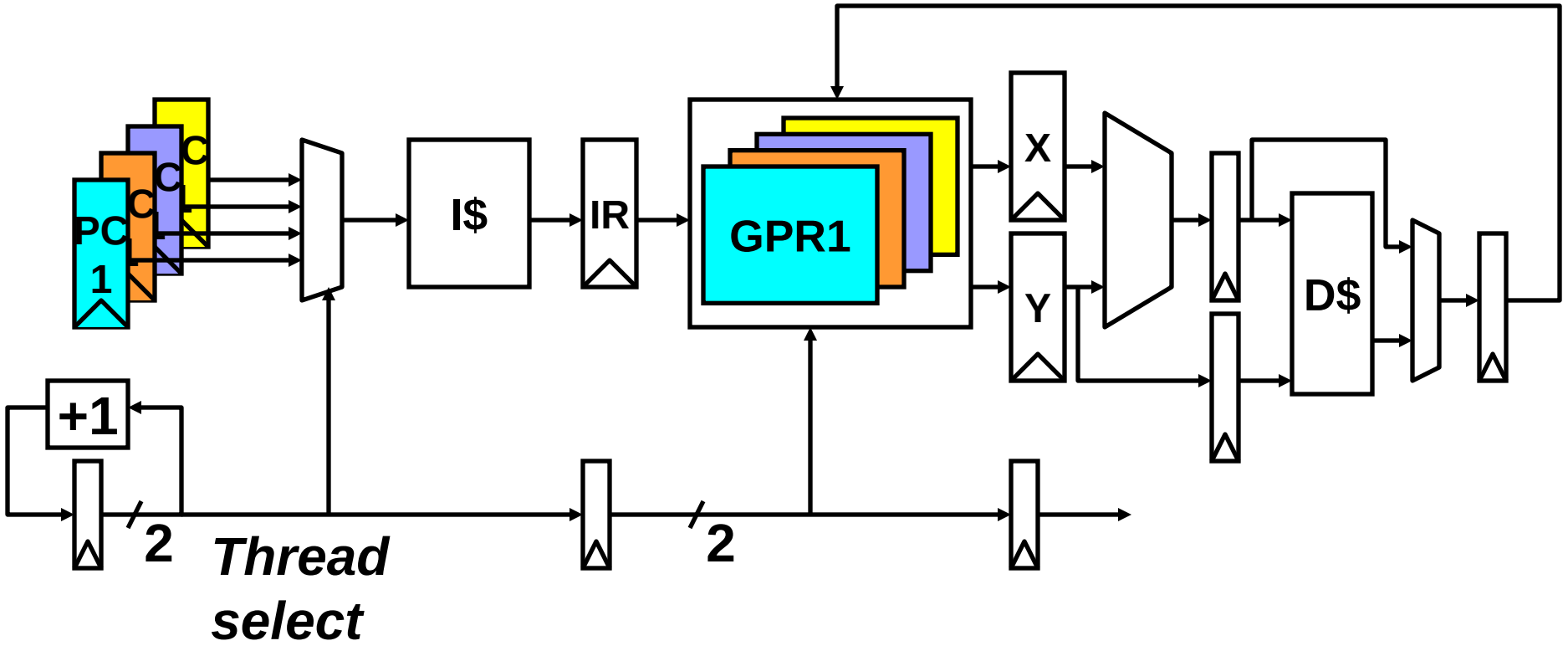
Multithreading

- **Advantages**
  - To overlap cache misses
  - To better utilize functional units
  - Sometimes threads share data, leading to a better cache performance
- **Disadvantages**
  - Multiple threads can interfere with each other when sharing hardware resources such as caches or translation lookaside buffers(TLBs).
  - Execution times of a single-thread are not improved but can be degraded, even when only one thread is executing. This is due to slower frequencies and/or additional pipeline stages that are necessary to accommodate thread-switching hardware.
  - Hardware support for Multithreading is more visible to software

CDC 6600 Peripheral Processors
(Cray, 1964)

- First multithreaded hardware
- 10 “virtual” I/O processors sharing the same execution units
- Fixed interleave on simple pipeline
- Pipeline has 100ns cycle time
- Each virtual processor executes one instruction every 1000ns
- Why? ➔ memory access requires 10 cycles

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

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

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
  - The eight-stage instruction pipeline allowed instructions from eight different processes to proceed at once. In fact, only one instruction from a given process was allowed to be present in the pipeline at any point in time.
- precursor to Tera MTA (Multithreaded Architecture)

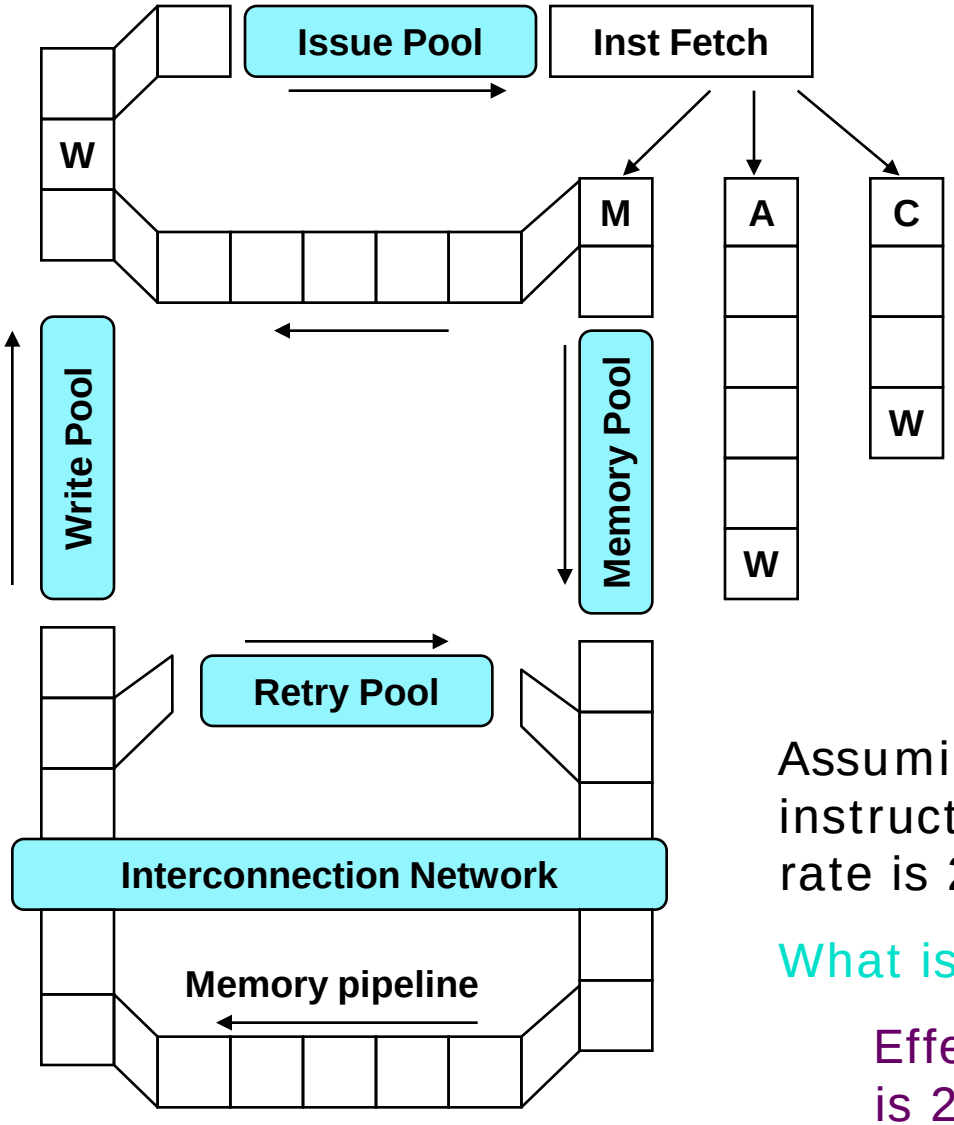
Tera MTA (1990-97)

- 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
  - CMOS version, MTA-2, 50W/processor

MTA Architecture

- Each processor supports 128 active hardware threads
  - 1 x 128 = 128 stream status word (SSW) registers,
  - 8 x 128 = 1024 branch-target registers,
  - 32 x 128 = 4096 general-purpose registers
- Three operations packed into 64-bit instruction (short VLIW)
  - One memory operation,
  - One arithmetic operation, plus
  - One arithmetic or branch operation
- Thread creation and termination instructions
- Explicit 3-bit “lookahead” field in instruction gives number of subsequent instructions (0-7) that are independent of this one
  - c.f. instruction grouping in VLIW
  - allows fewer threads to fill machine pipeline
  - used for variable-sized branch delay slots

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

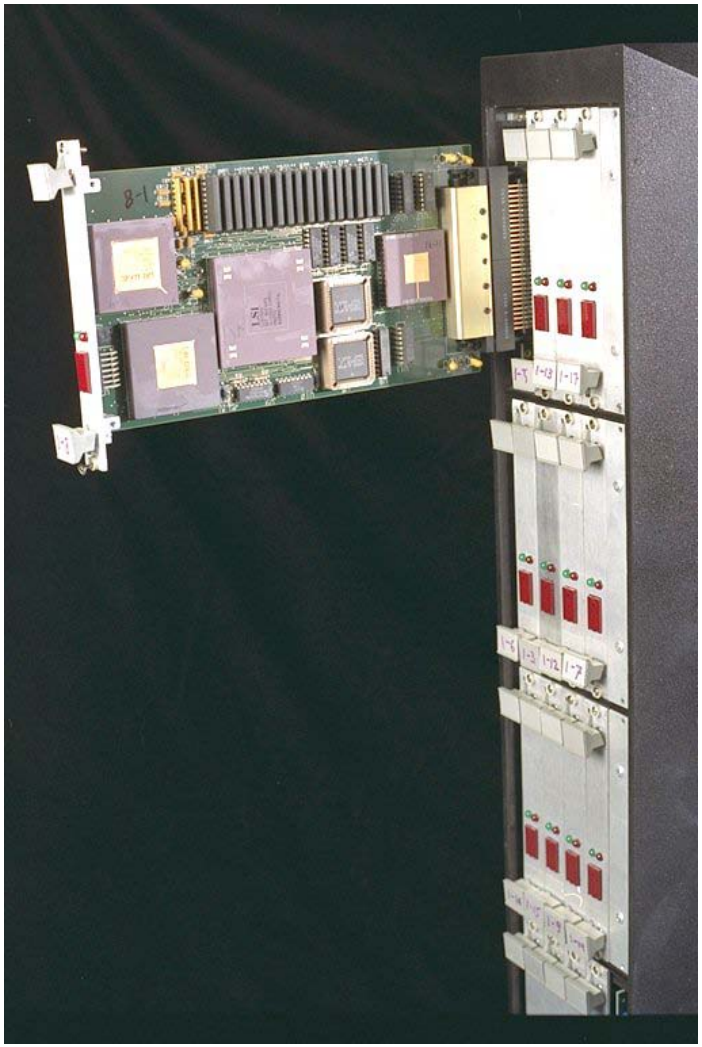
Tera MTA designed for supercomputing applications with large data sets and low locality

- **No data cache**
- **Many parallel threads needed to hide large memory latency**
  - **Tera --- 256 mem-ops/cycle * 150 cycles/mem-op = 38K instructions-in-flight…**
  - **Tera not very successful, 2 machines sold.**
  - **Changed their name back to Cray!**

Other applications are more cache friendly

- **Few pipeline bubbles when cache getting 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

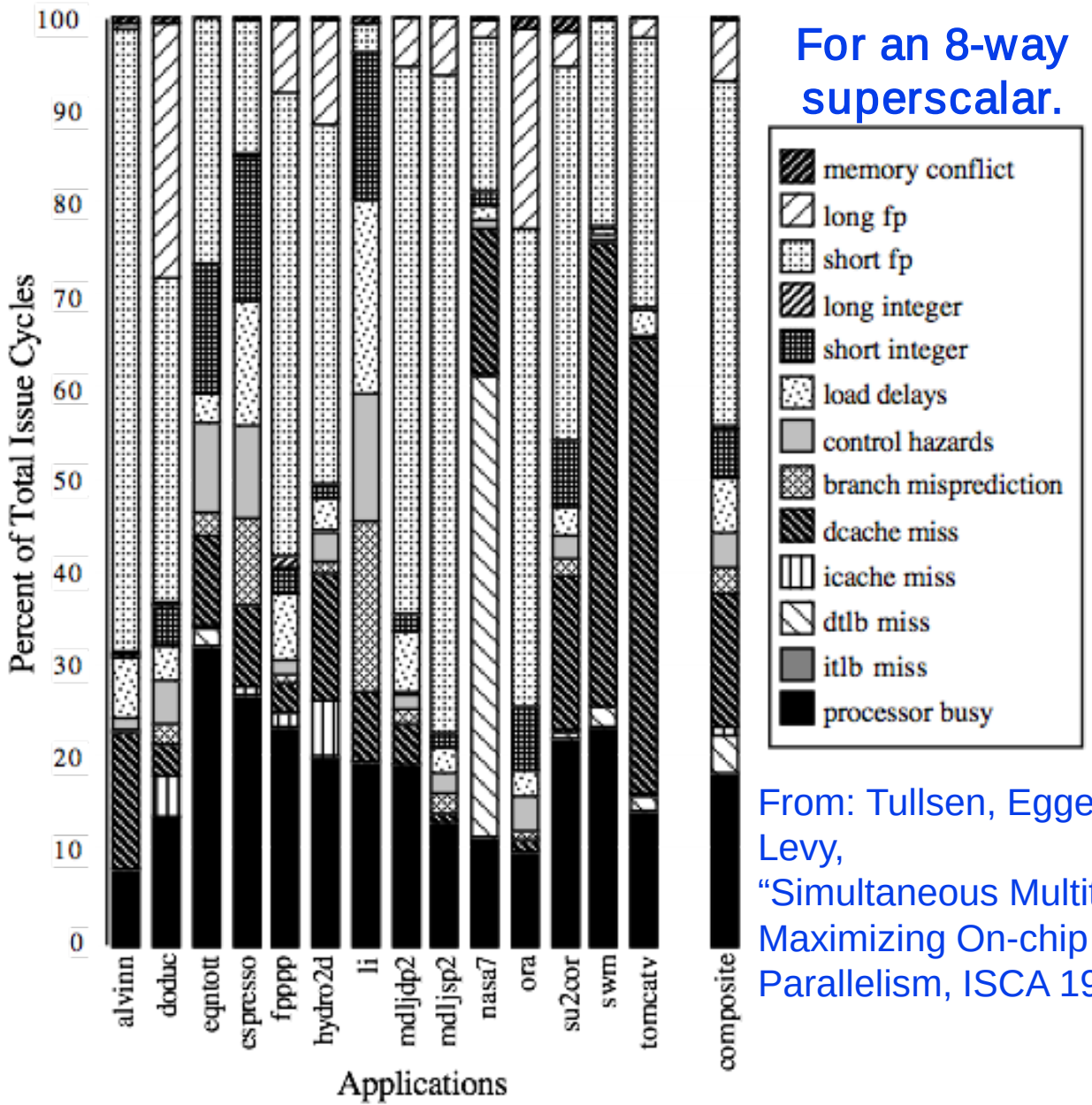
IBM PowerPC RS64-IV (2000)

- 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

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.

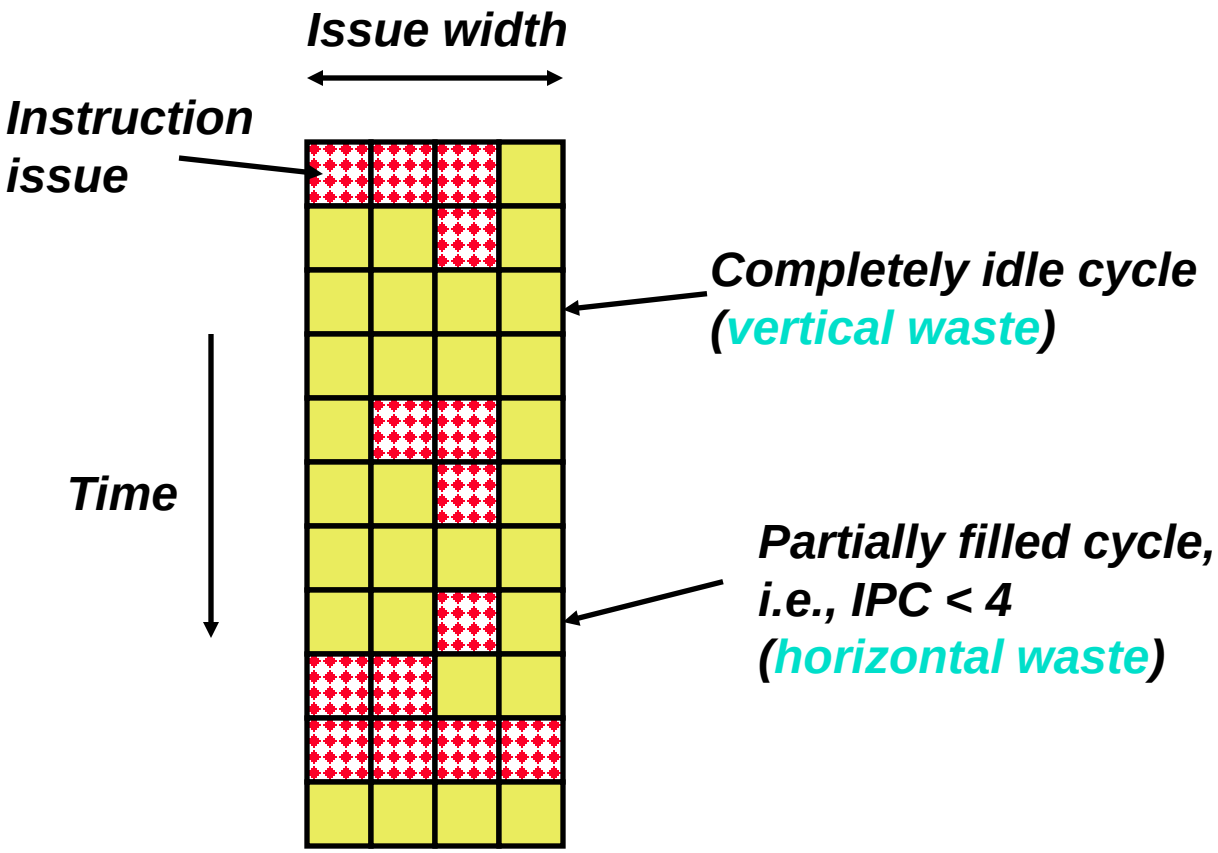
For most apps, most execution units lie idle in an OoO superscalar



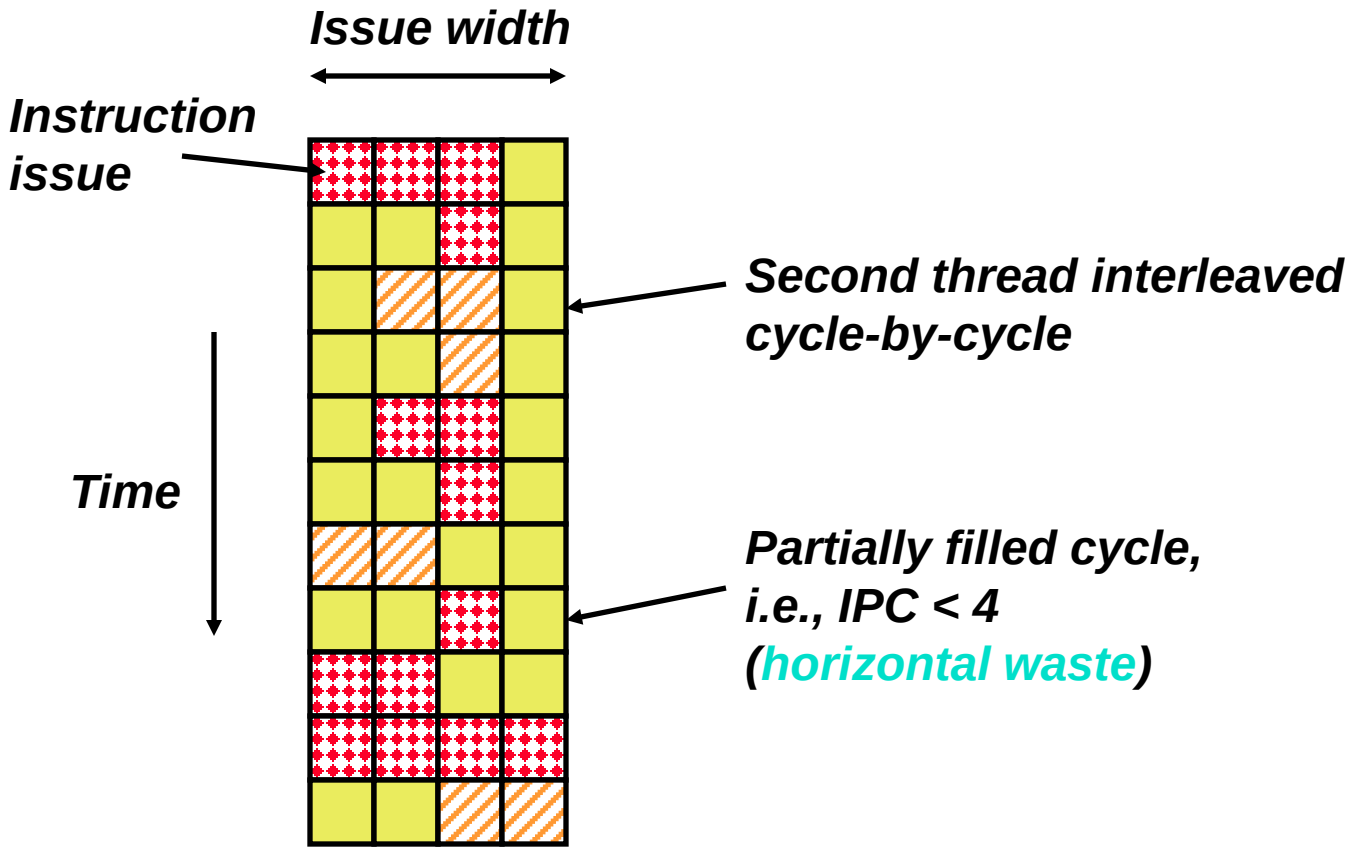
For an 8-way superscalar.

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

Superscalar Machine Efficiency

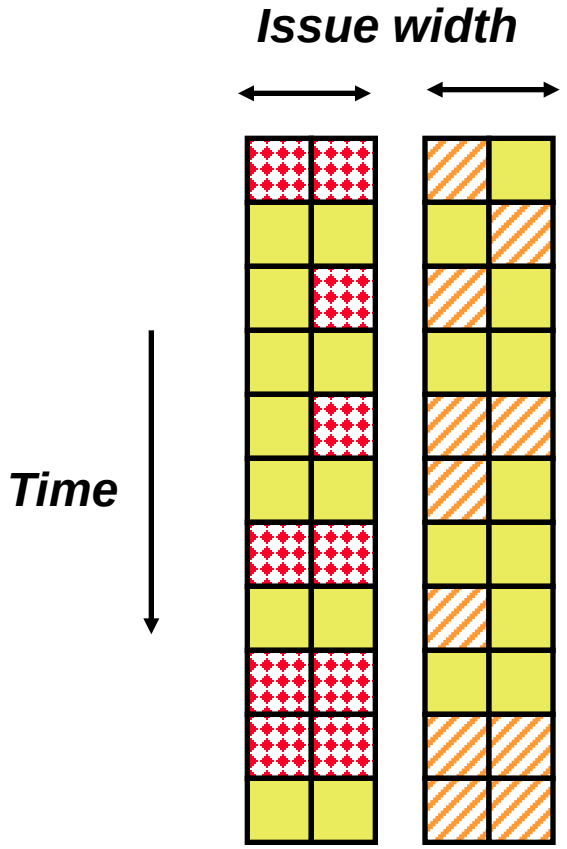


Vertical Multithreading



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

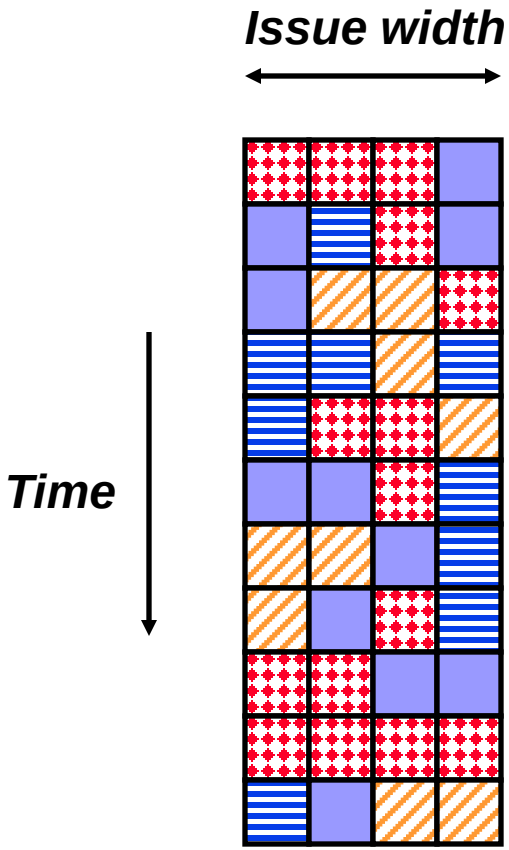
Chip Multiprocessing (CMP)



- 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

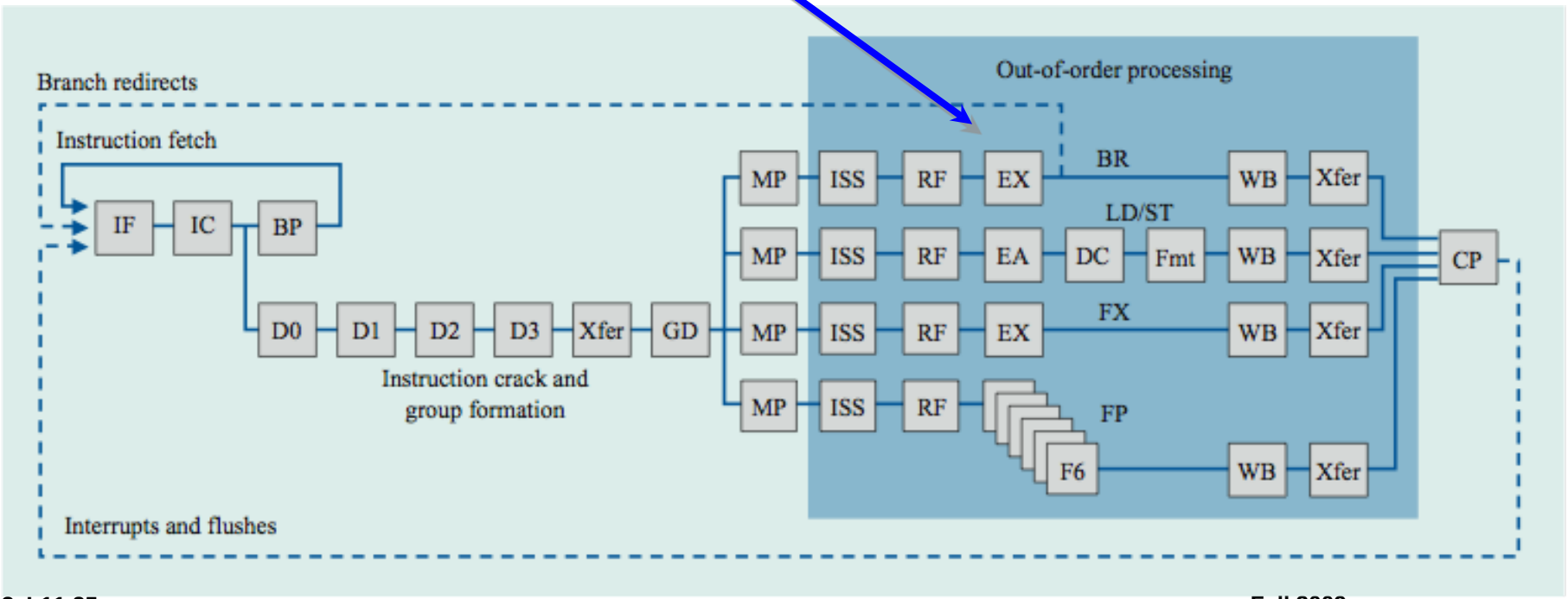
O-o-O Simultaneous Multithreading

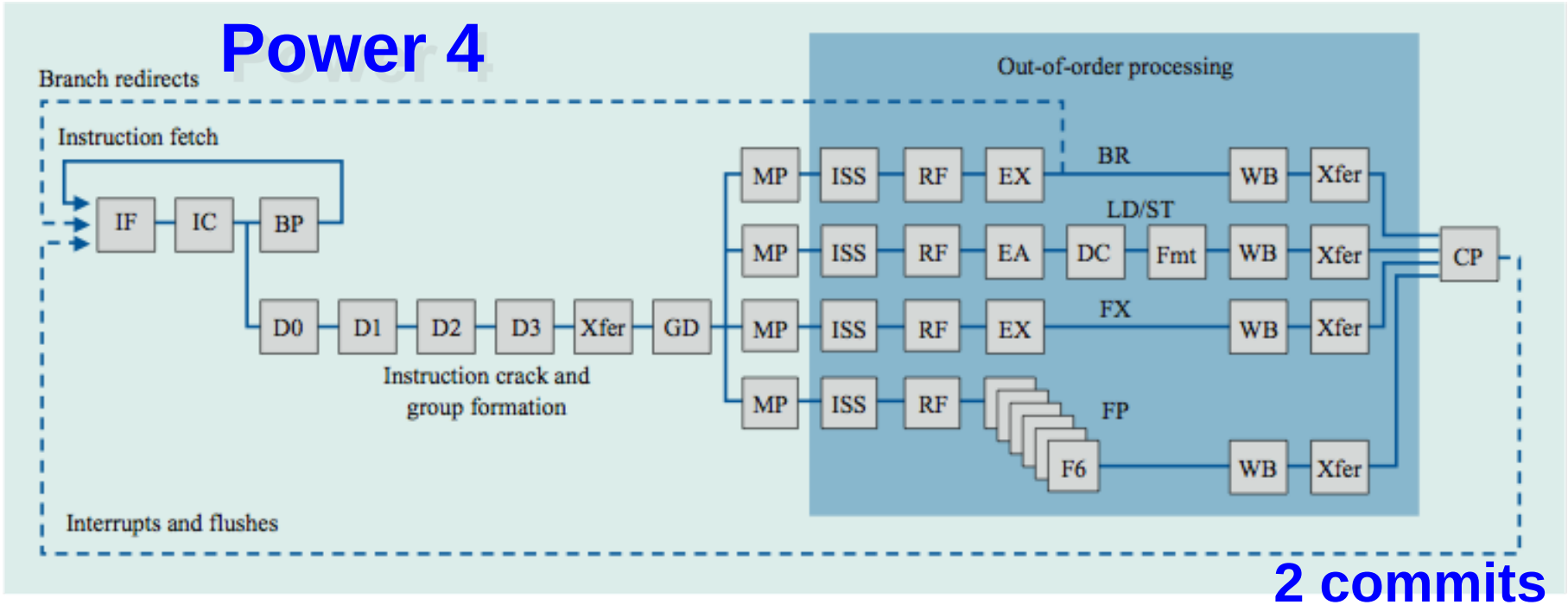
[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

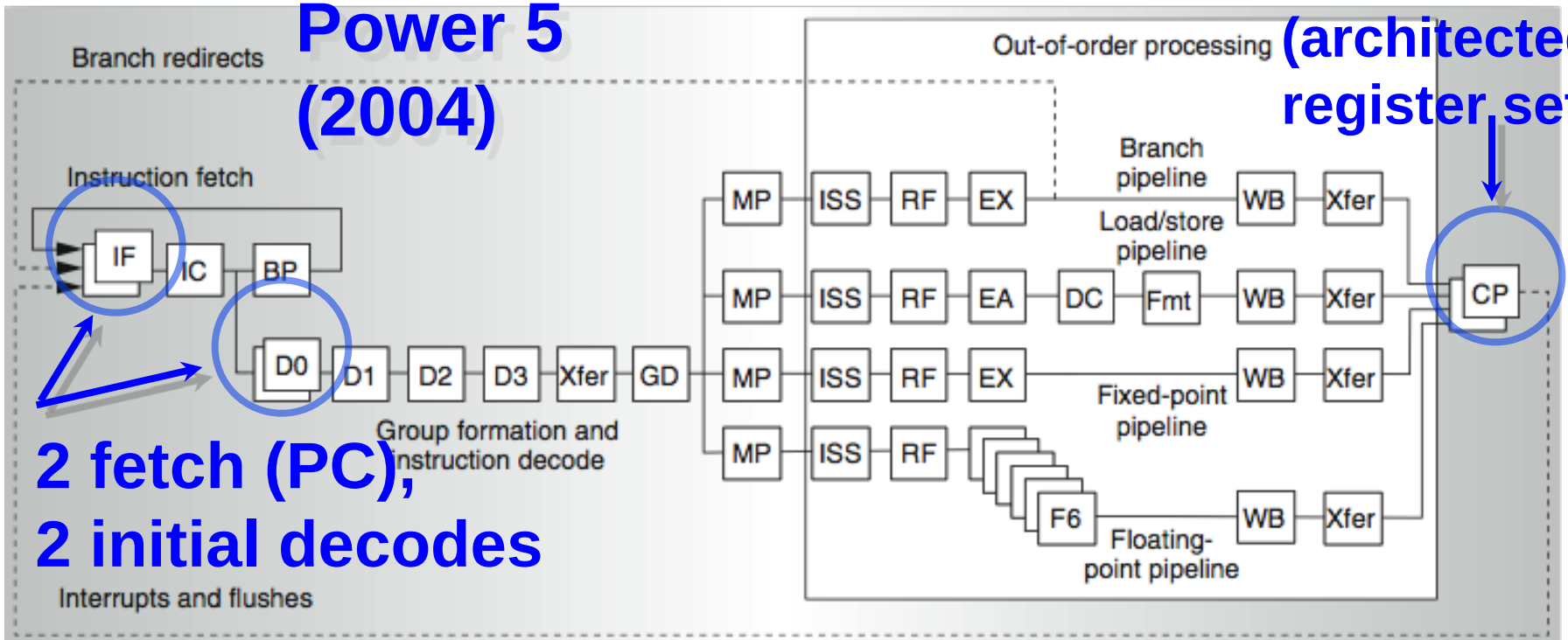
- **Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously**
- **Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads**
- **OOO instruction window already has most of the circuitry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

Power 4

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



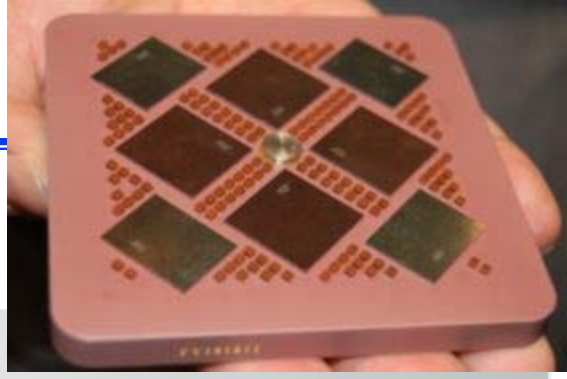


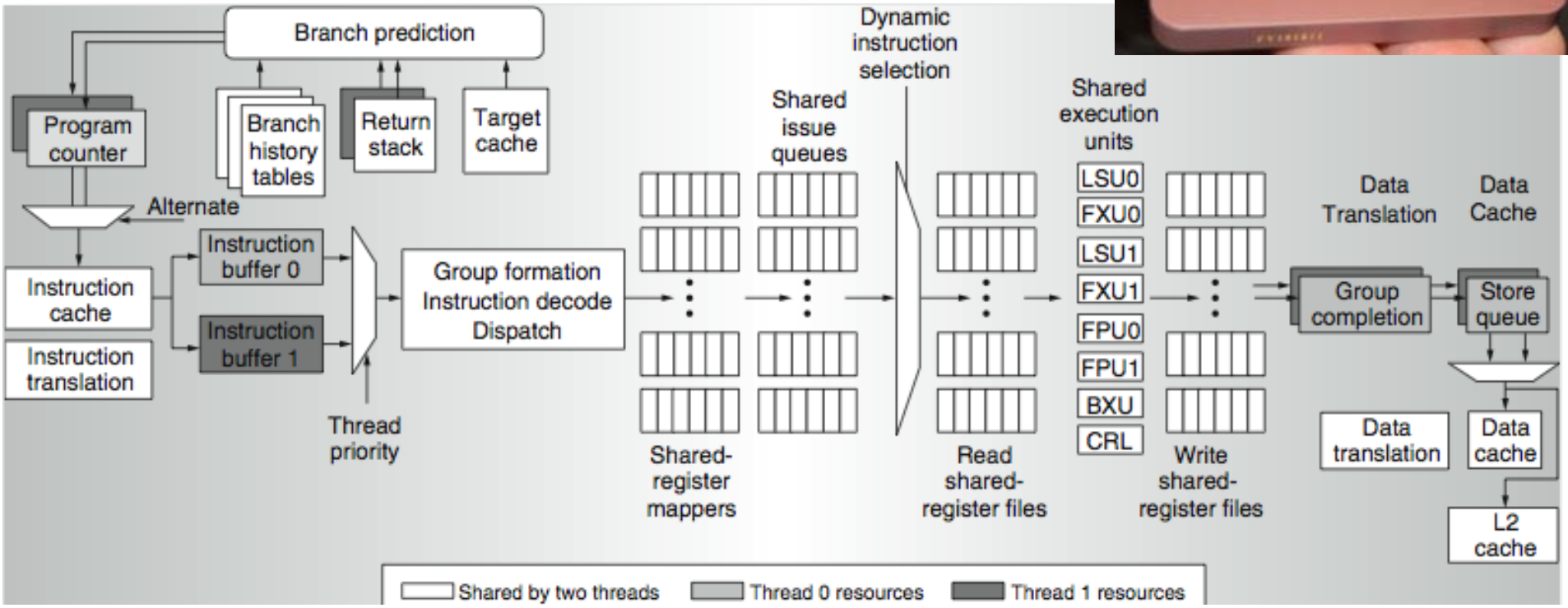


2 commits (architected register sets)

2 fetch (PC), 2 initial decodes

Power 5 data flow ...



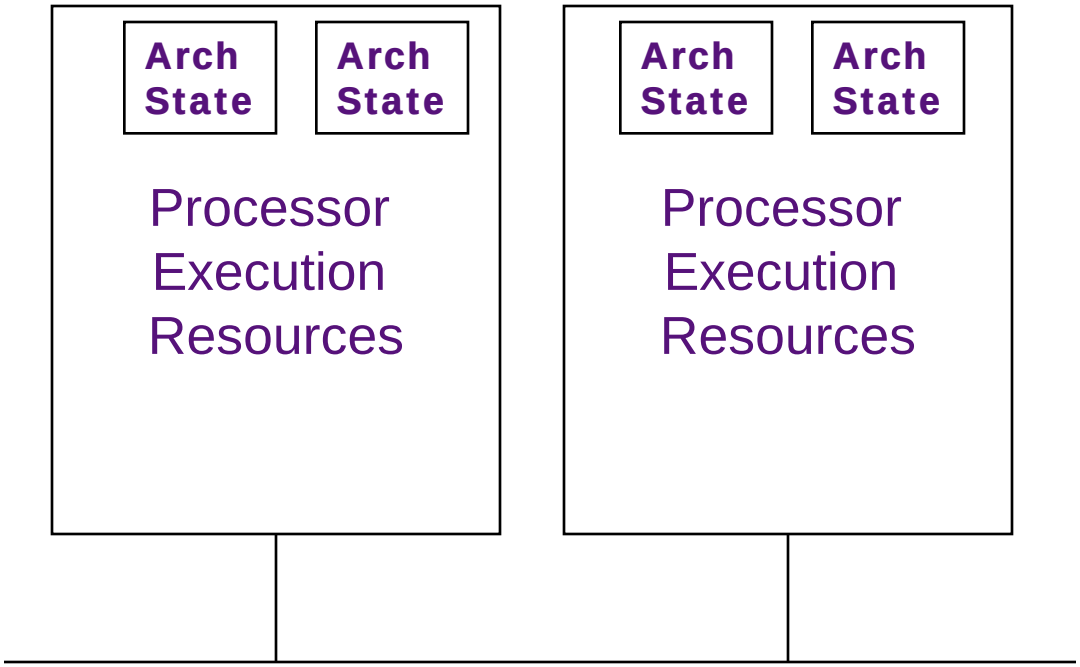


Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

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 (physical) 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

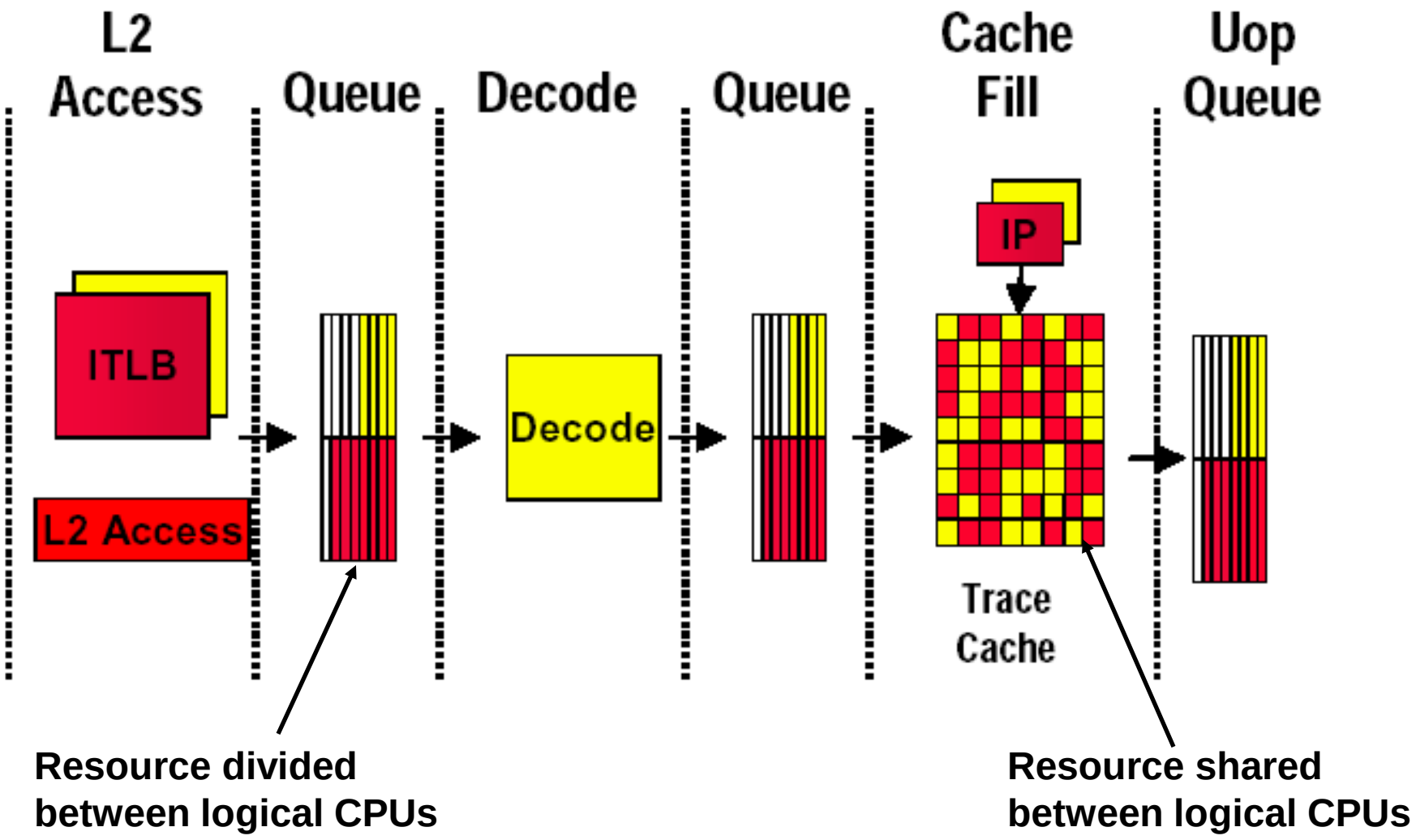
Pentium-4 Hyperthreading (2002)



Pentium-4 Hyperthreading (2002)

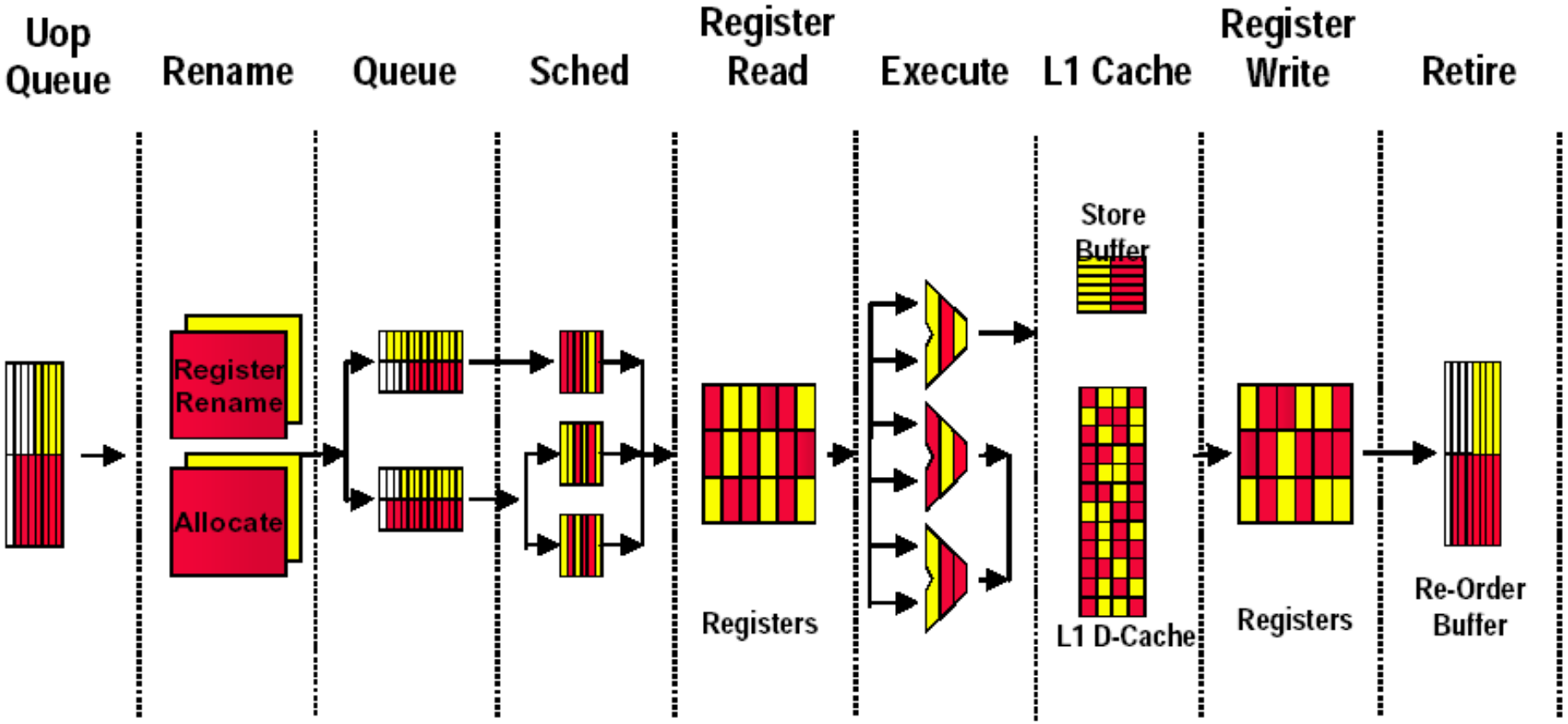
- First commercial SMT design (2-way SMT)
  - Hyperthreading == SMT
- Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress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

Pentium-4 Hyperthreading
Front End



[Intel Technology Journal, Q1 2002]

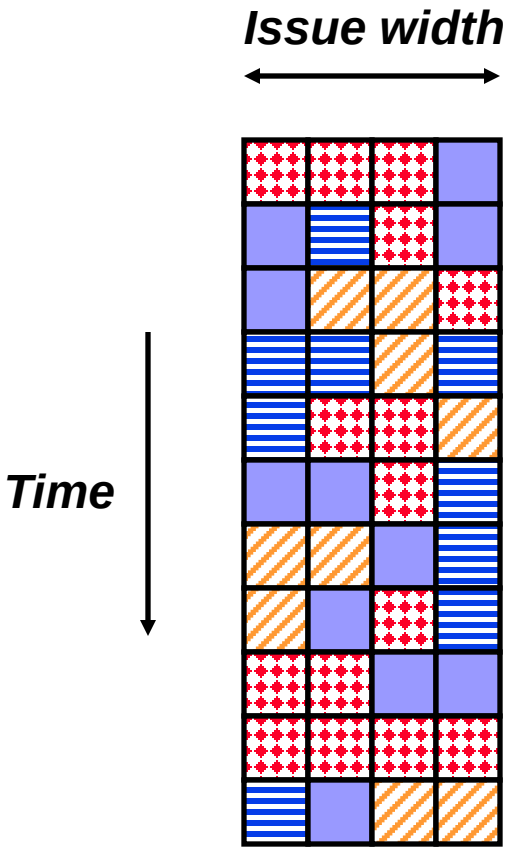
Pentium-4 Hyperthreading
Execution Pipeline



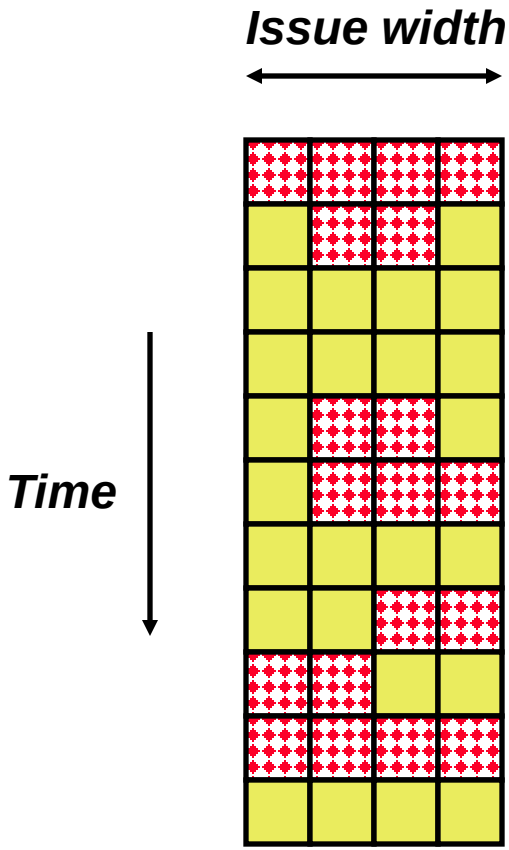
[Intel Technology Journal, Q1 2002]

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)



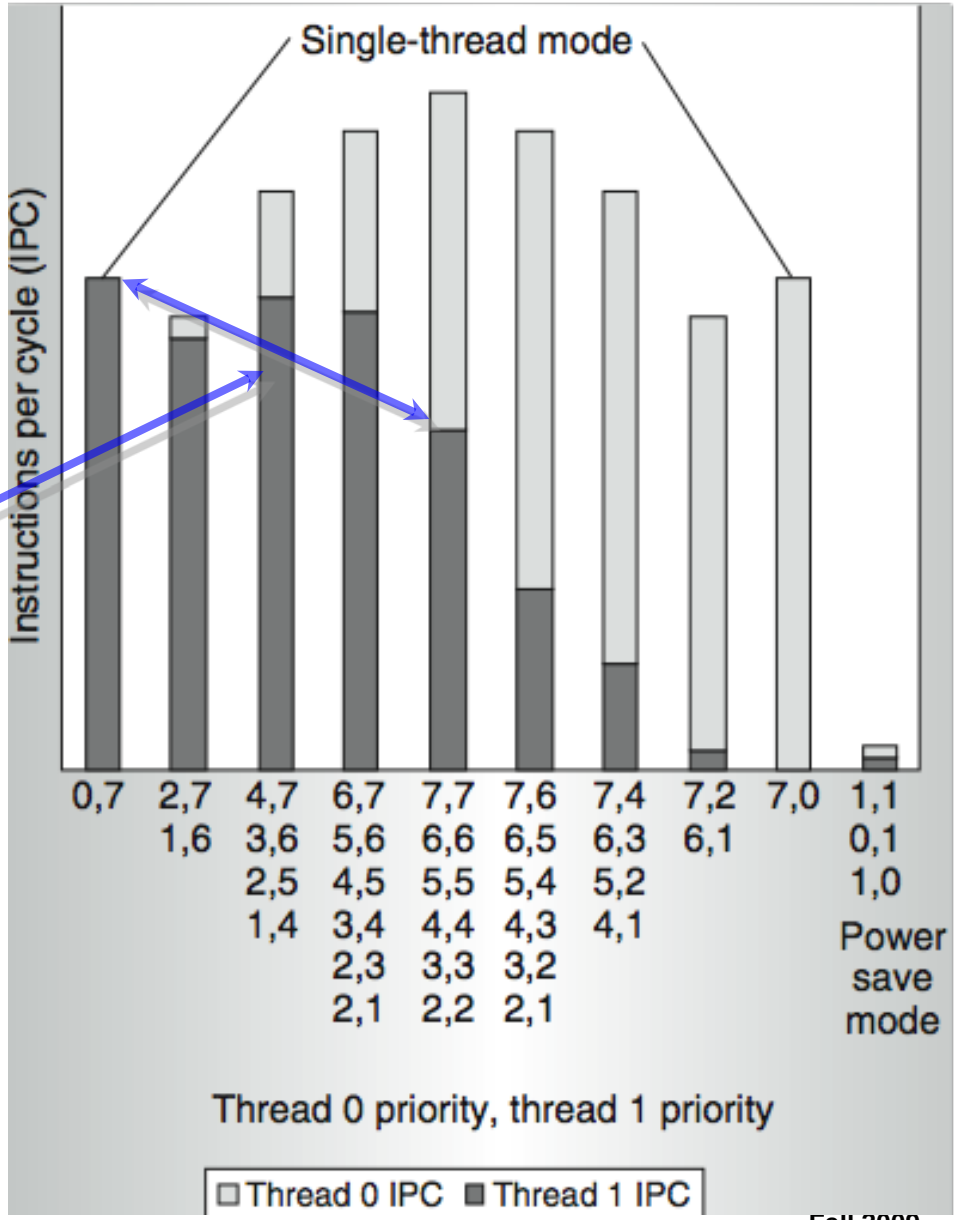
Initial Performance of SMT

- 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

Power 5 thread performance ...

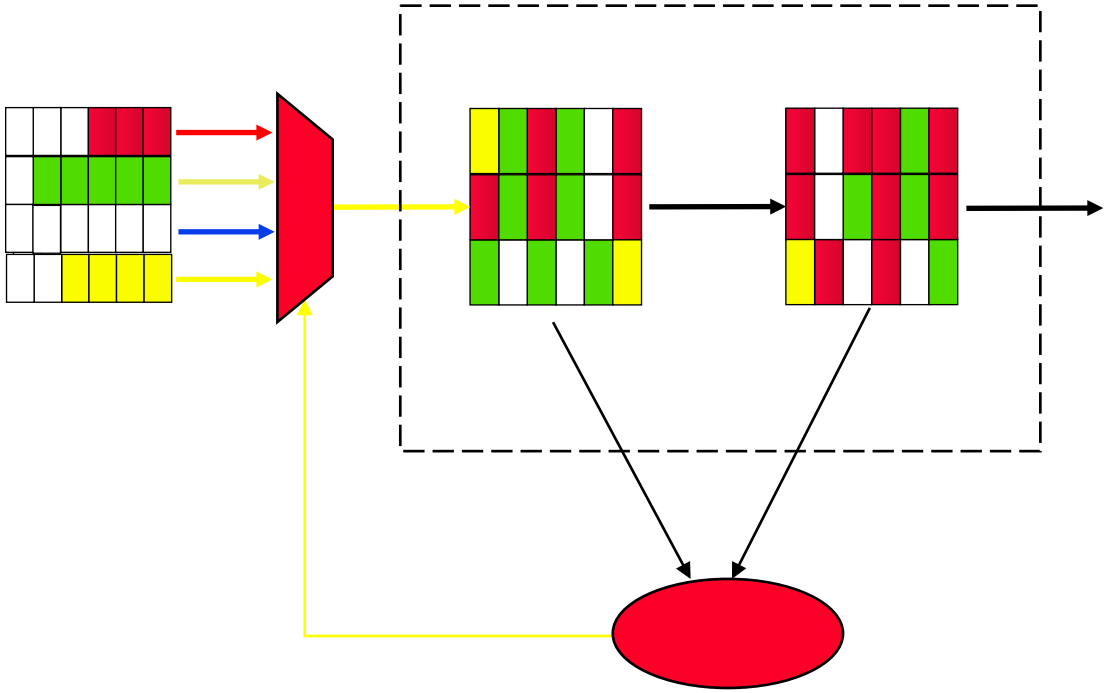
Relative priority of each thread controllable in hardware.

For balanced operation, both threads run slower than if they "owned" the machine.



Icount Choosing Policy

Fetch from thread with the least instructions in flight.

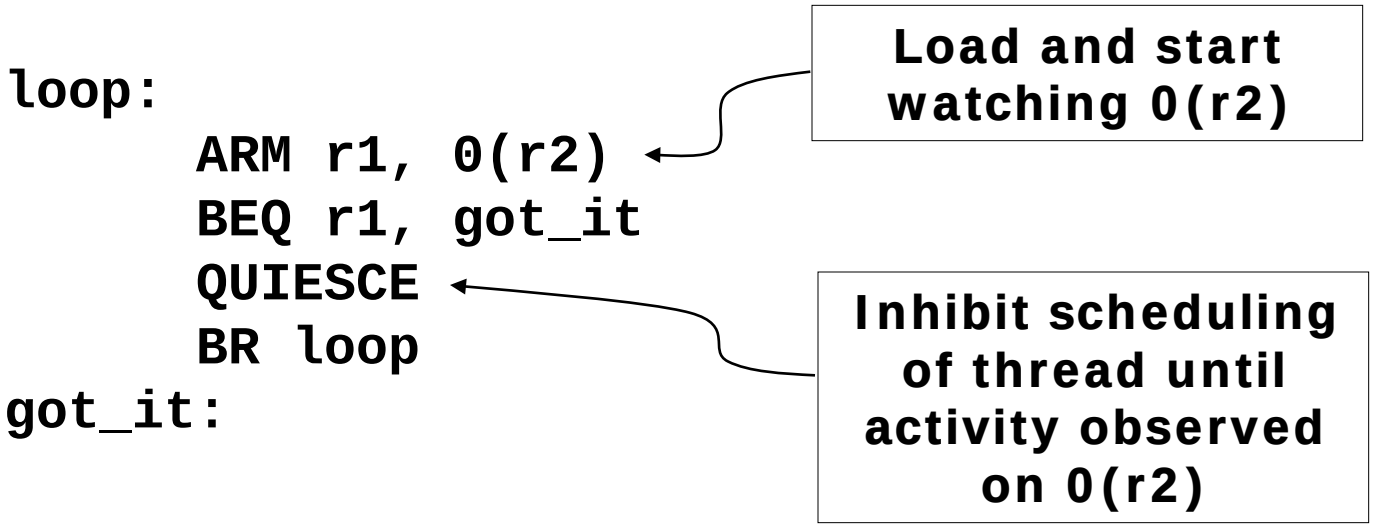


Why does this enhance throughput?

SMT Fetch Policies (Locks)

- **Problem:**
  Spin looping thread consumes resources

- **Solution:**
  Provide quiescing operation that allows a thread to sleep until a memory location changes



Summary: Multithreaded Categories