Lecture 10:
Thread Level Parallelism (TLP)

Iakovos Mavroidis

Computer Science Department
University of Crete

Multiple Issue

$$CPI = CPI_{ideal} + Stalls_{structural} + Stalls_{RAW} + Stalls_{WAR} + Stalls_{WAW} + Stalls_{control}$$

Προσοχή να διατηρουνται

1. **Data flow**
2. **Exception Behavior**

Έχουμε μελετήσει
θα μελετήσουμε σημερα

Δυναμικές δρομολόγηση εντολών (hardware)

- **Scoreboard (ελάττωση RAW stalls)**
- **Register Renaming**
  - **α)Tomasulo**
  - **(ελάττωση WAR και WAW stalls)**
  - **β)Reorder Buffer**
- **Branch prediction**
  - **(ελάττωση Control stalls)**
- **Multiple Issue (CPI < 1)**
- **Multithreading (CPI < 1)**

Στατικές (shoftware/compiler)

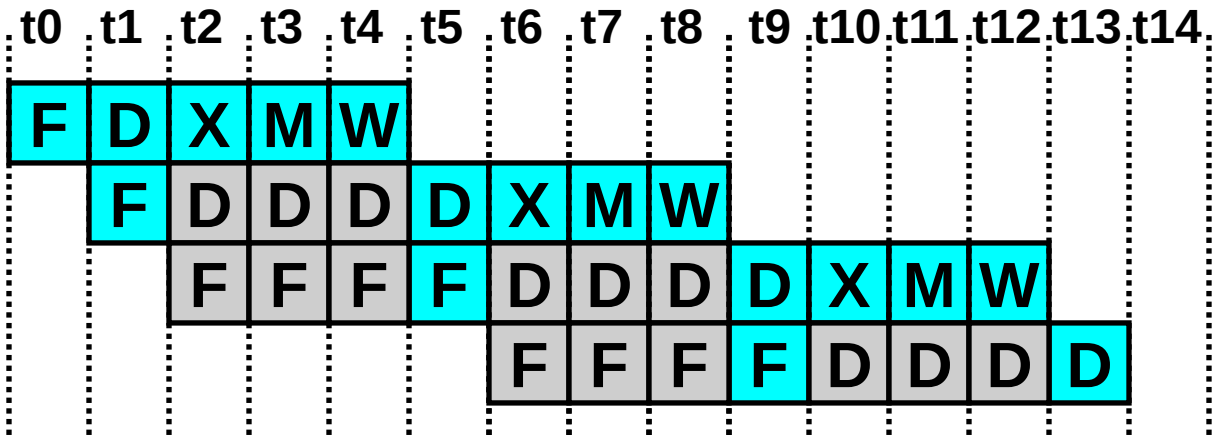
- **Loop Unrolling**
- **Software Pipelining**
- **Trace Scheduling**

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 next

What can be done to cope with this?

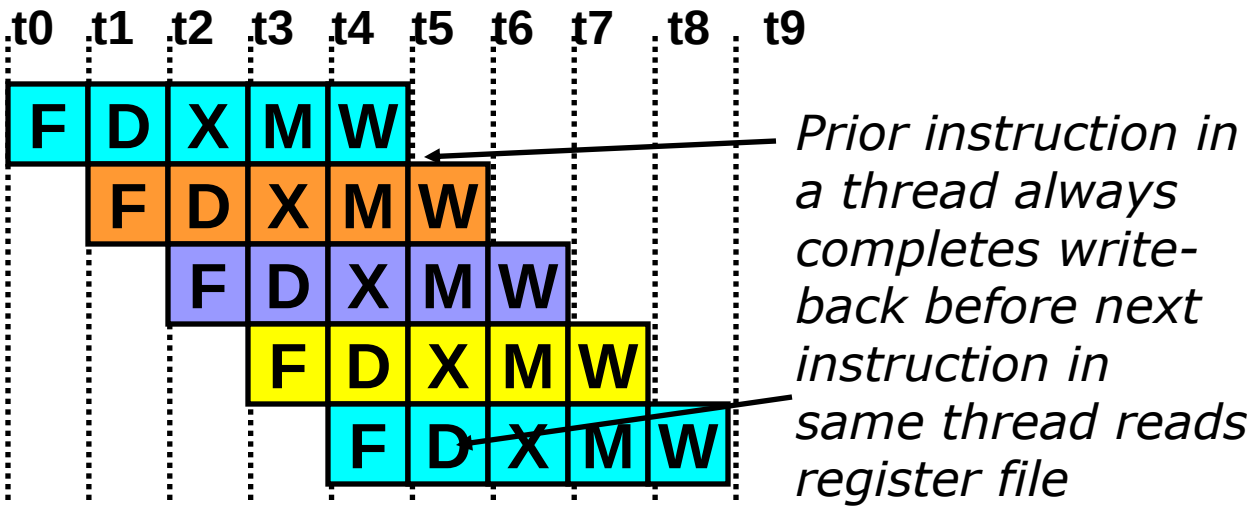
Solution with Multithreading

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

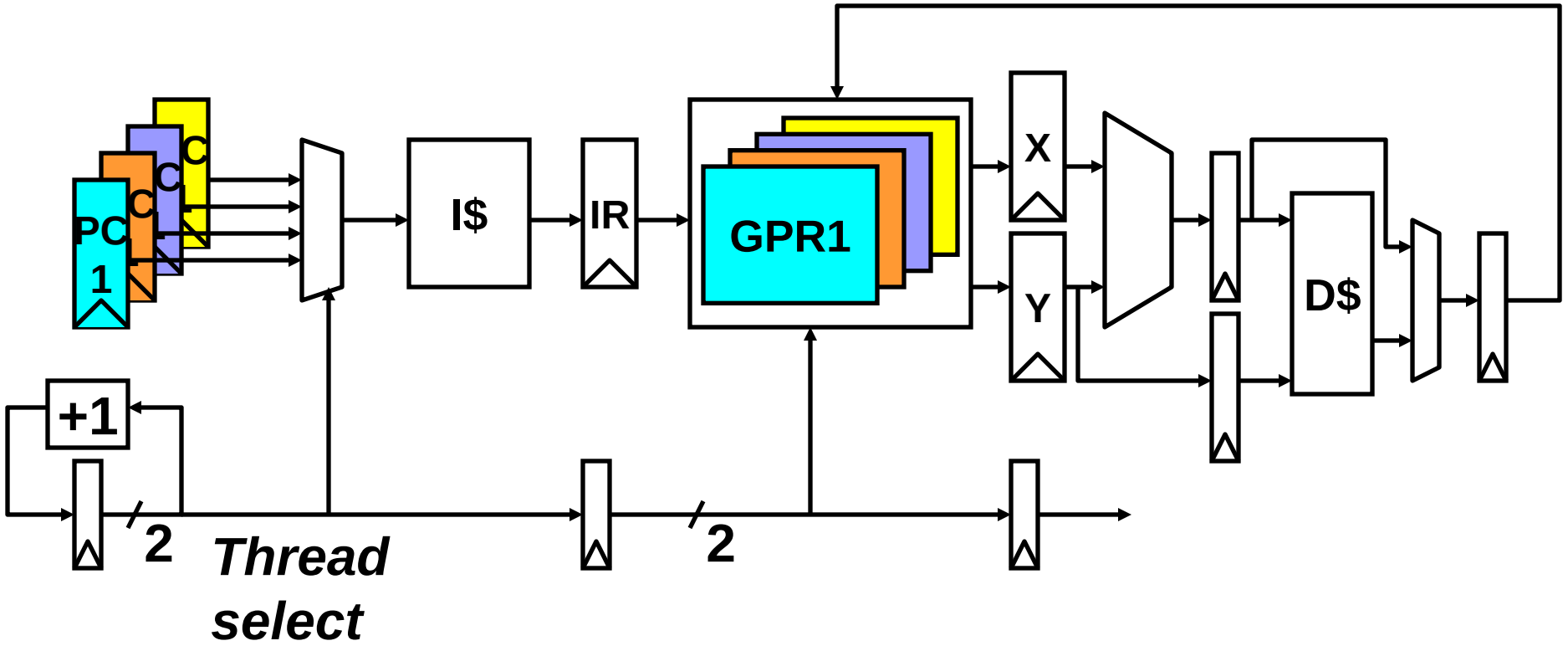
-- One way is to interleave execution of instructions from different program threads on same pipeline

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)



Multithreaded DLX



- 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 Cost

- 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 costs?*

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

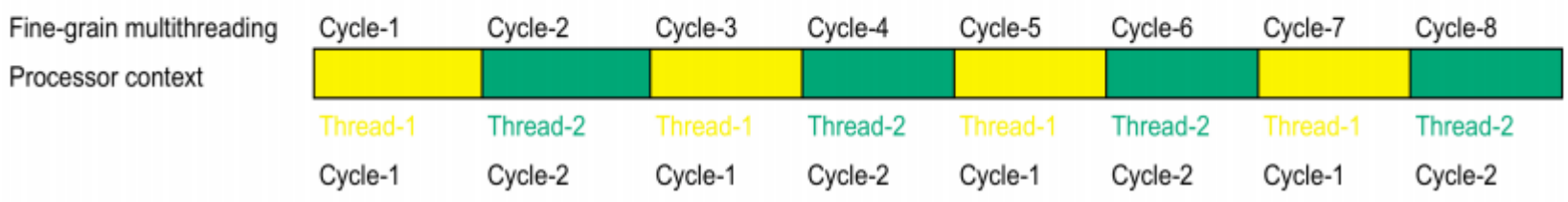
- Software-controlled interleave *(TI ASC PPUs, 1971)*
  - OS allocates S pipeline slots amongst N threads
  - hardware performs fixed interleave over S slots, executing whichever thread is in that slot

- 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

HW Multithreading alternatives

Fine-Grain Multithreading

- Fine-grain multithreading switches processor context every thread cycle
- Context belongs to same address space



HW Multithreading alternatives: Fine-Grain Multithreading

Fine-Grain Multithreading

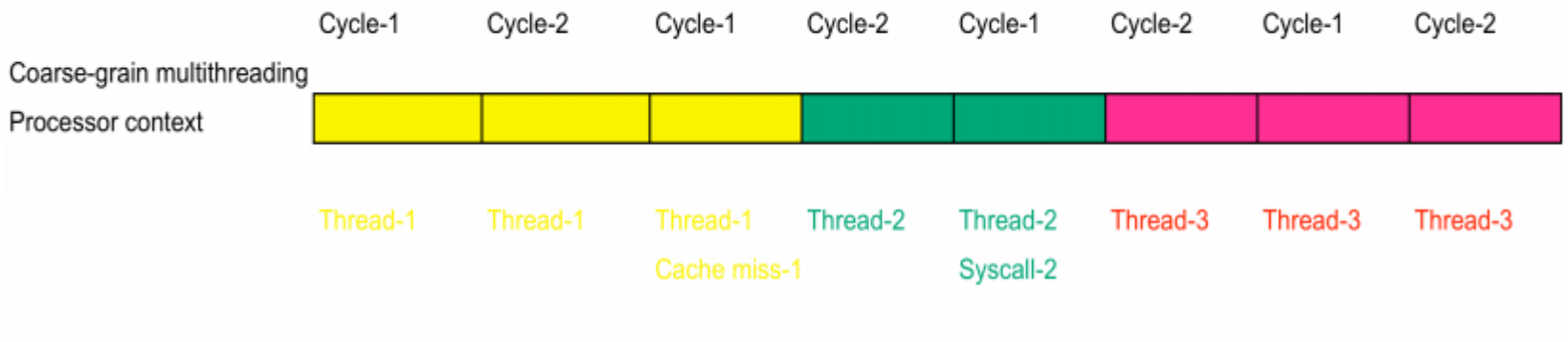
Switch every clock cycle

- Need fast HW switch between contexts
  - Multiple PCs and register files
  - Alternatively, thread ID attached to each GP register
- Implemented with round-robin scheduling, skipping stalled threads
- Hides both short and long stalls
- Delays all threads, even if they have no stalls

HW Multithreading alternatives: Fine-Grain Multithreading

Coarse-Grain Multithreading

- Coarse-grain multithreading switches processor context upon long-latency event
- Context may belong to different address space



HW Multithreading alternatives: Coarse-Grain Multithreading

Coarse-Grain Multithreading

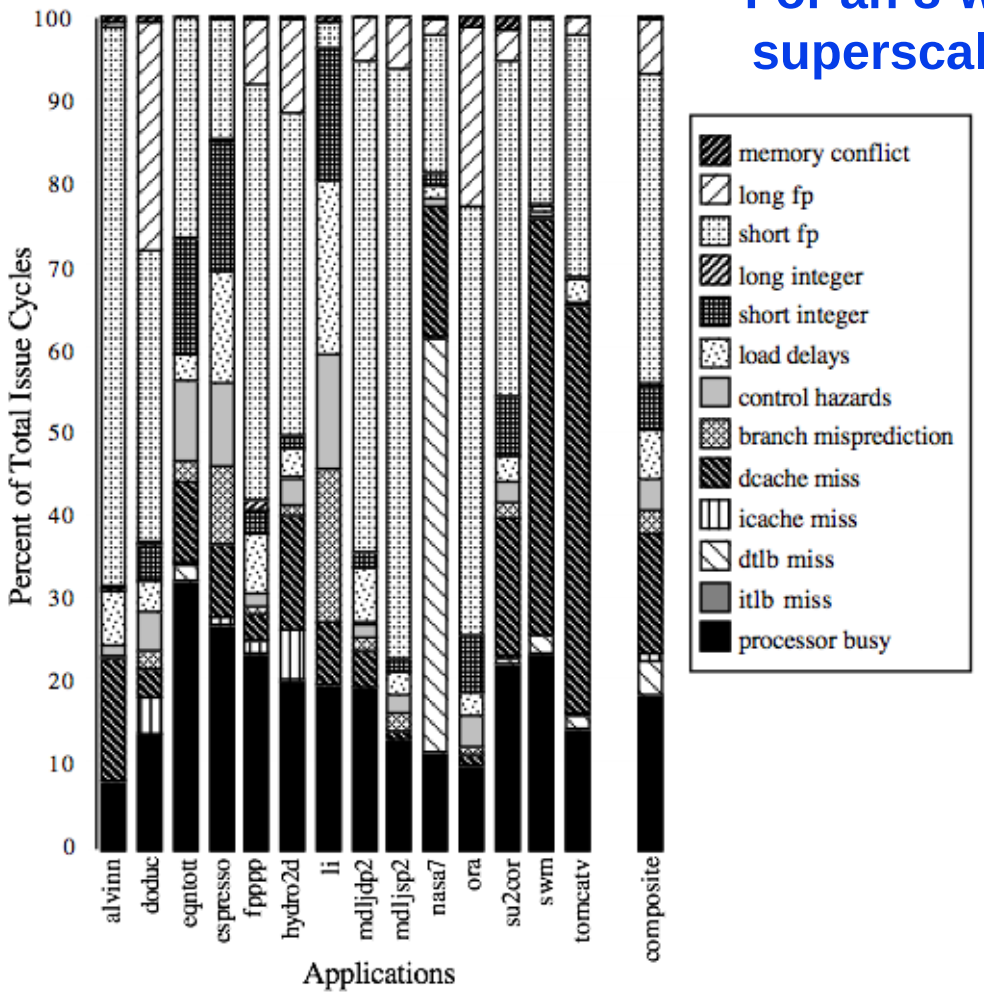
Switch upon long upon long-latency events

- Can afford slower context switch than fine-grain multithreading
- Threads are not slowed down
  - Other thread runs when current thread stalls
- Pipeline startup cost upon thread switching
  - Processor issues instructions from one thread (address space)

HW Multithreading alternatives: Simultaneous Multithreading

- 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.

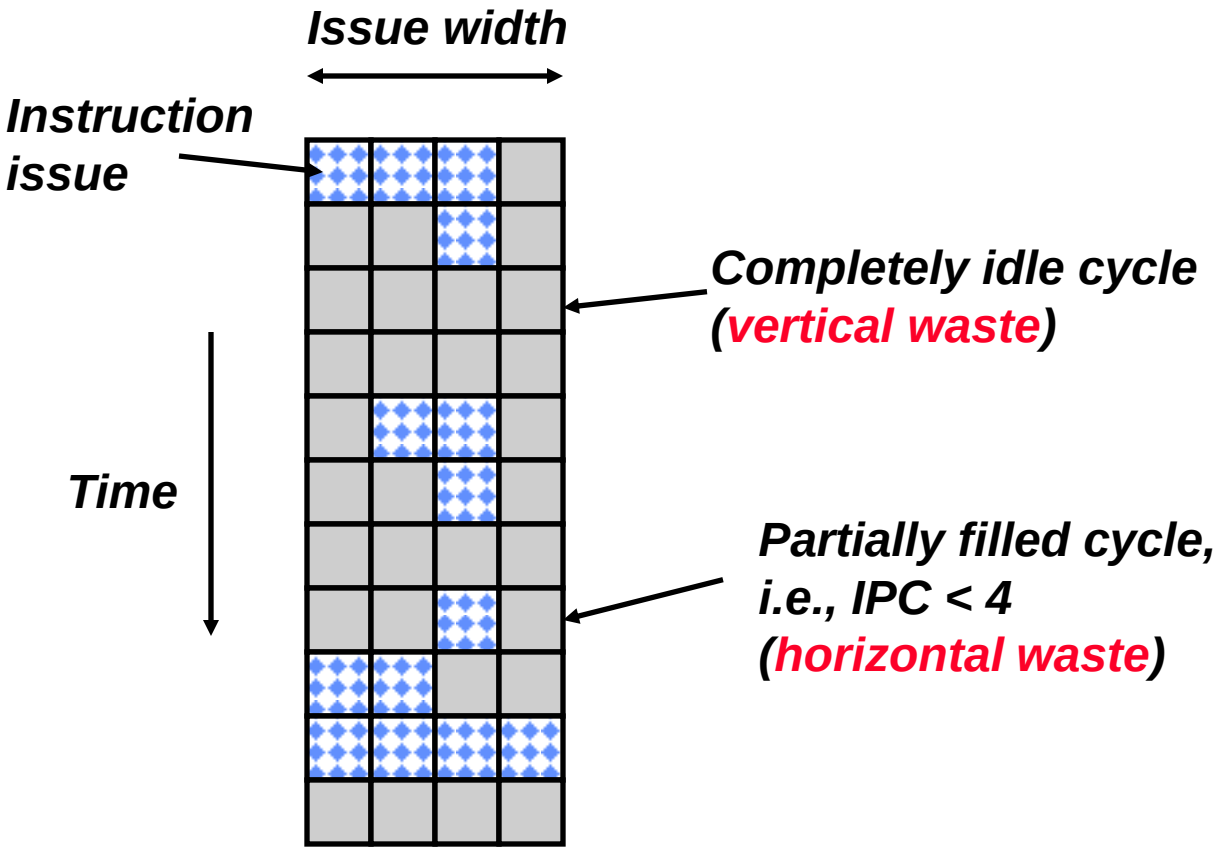
For most apps, most execution units lie idle in an OoO superscalar



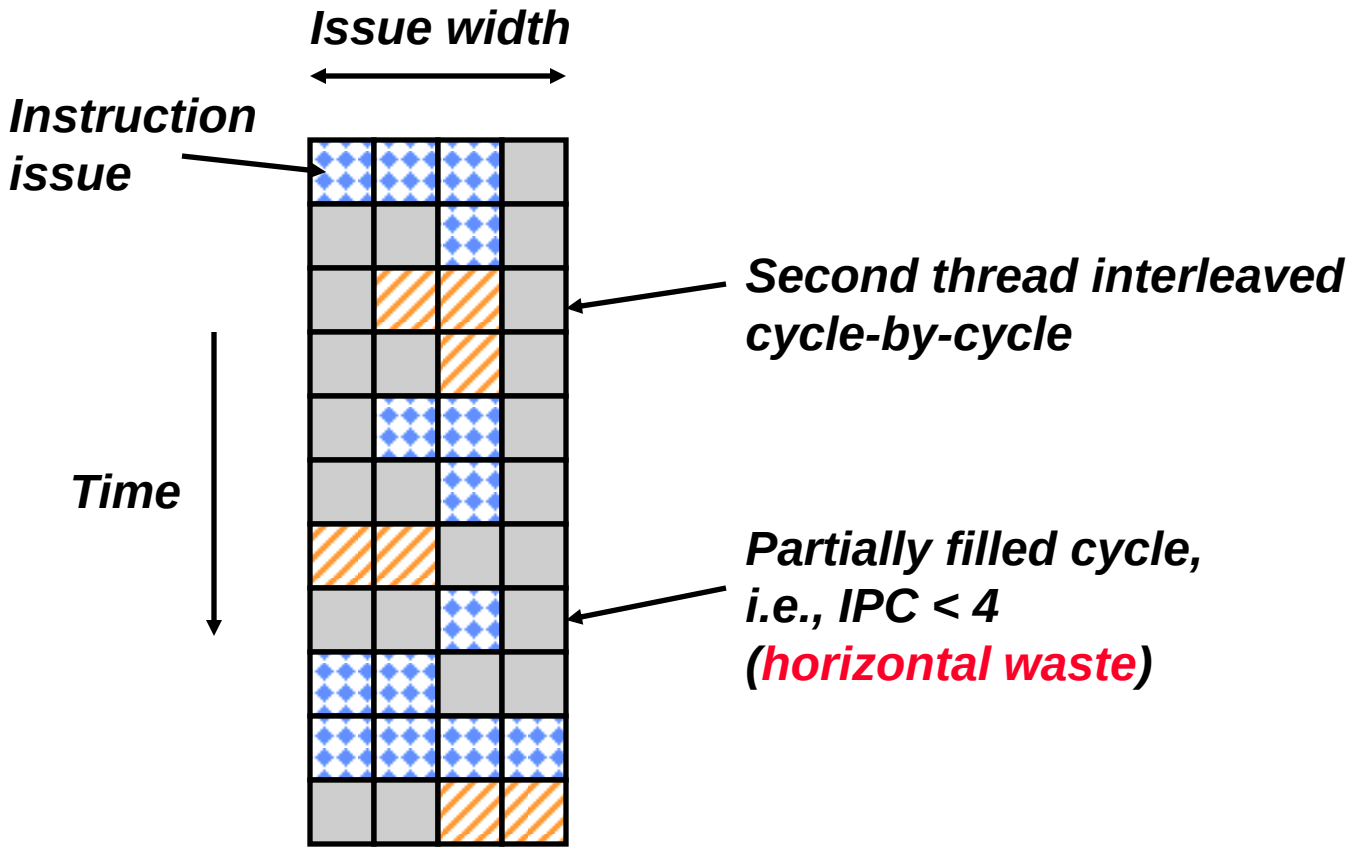
For an 8-way superscalar.

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

Superscalar Machine Efficiency

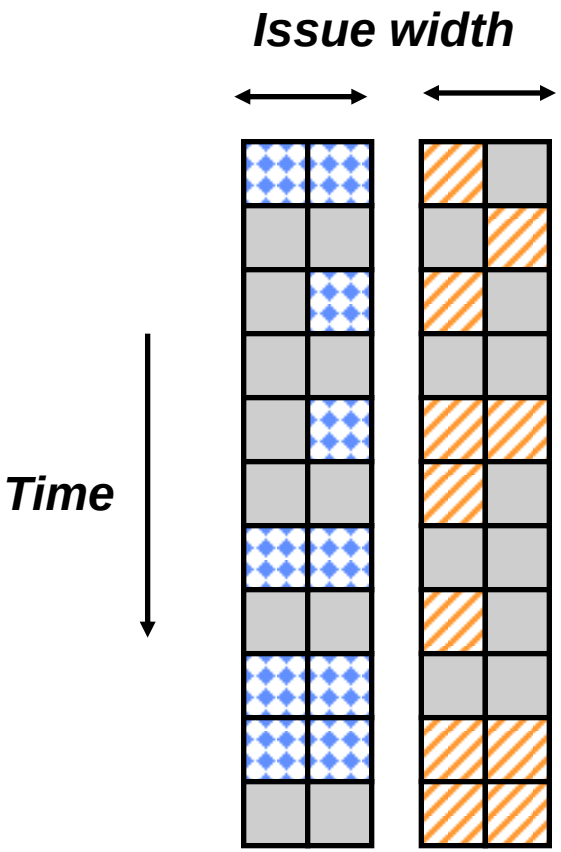


Vertical Multithreading



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

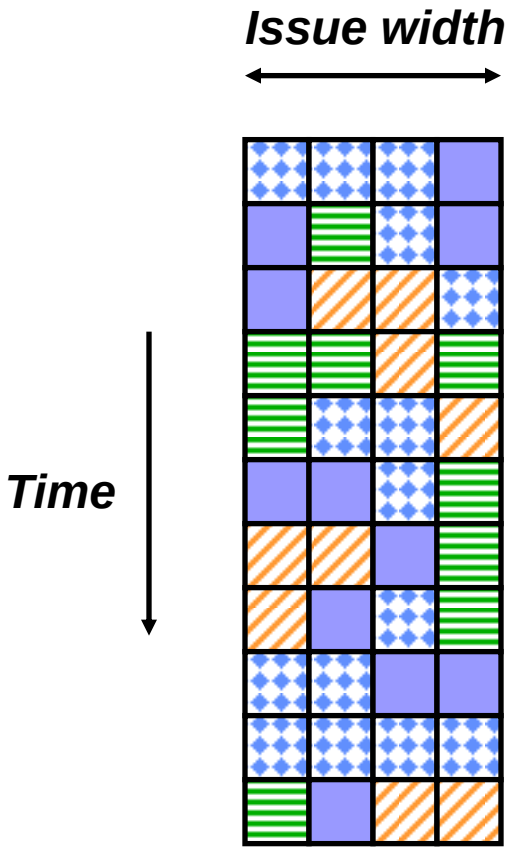
Chip Multiprocessing (CMP)



- 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: SMT

[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

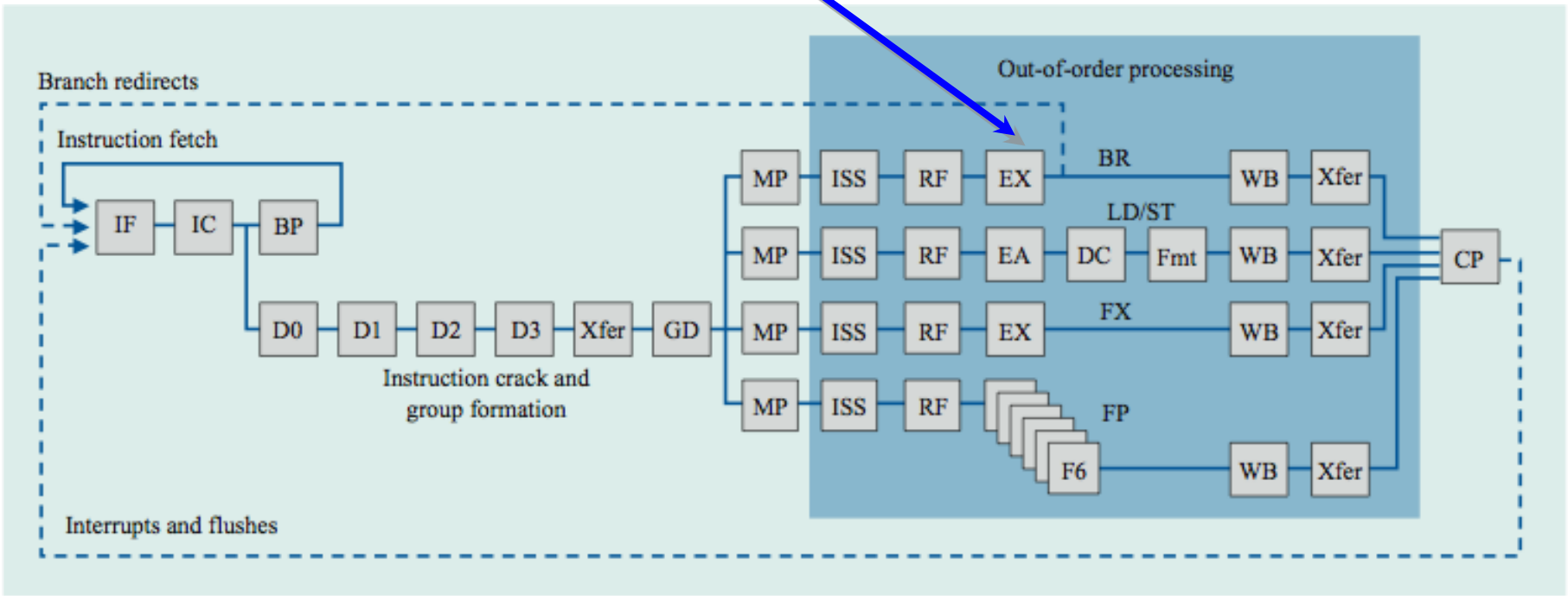
Shared HW mechanisms

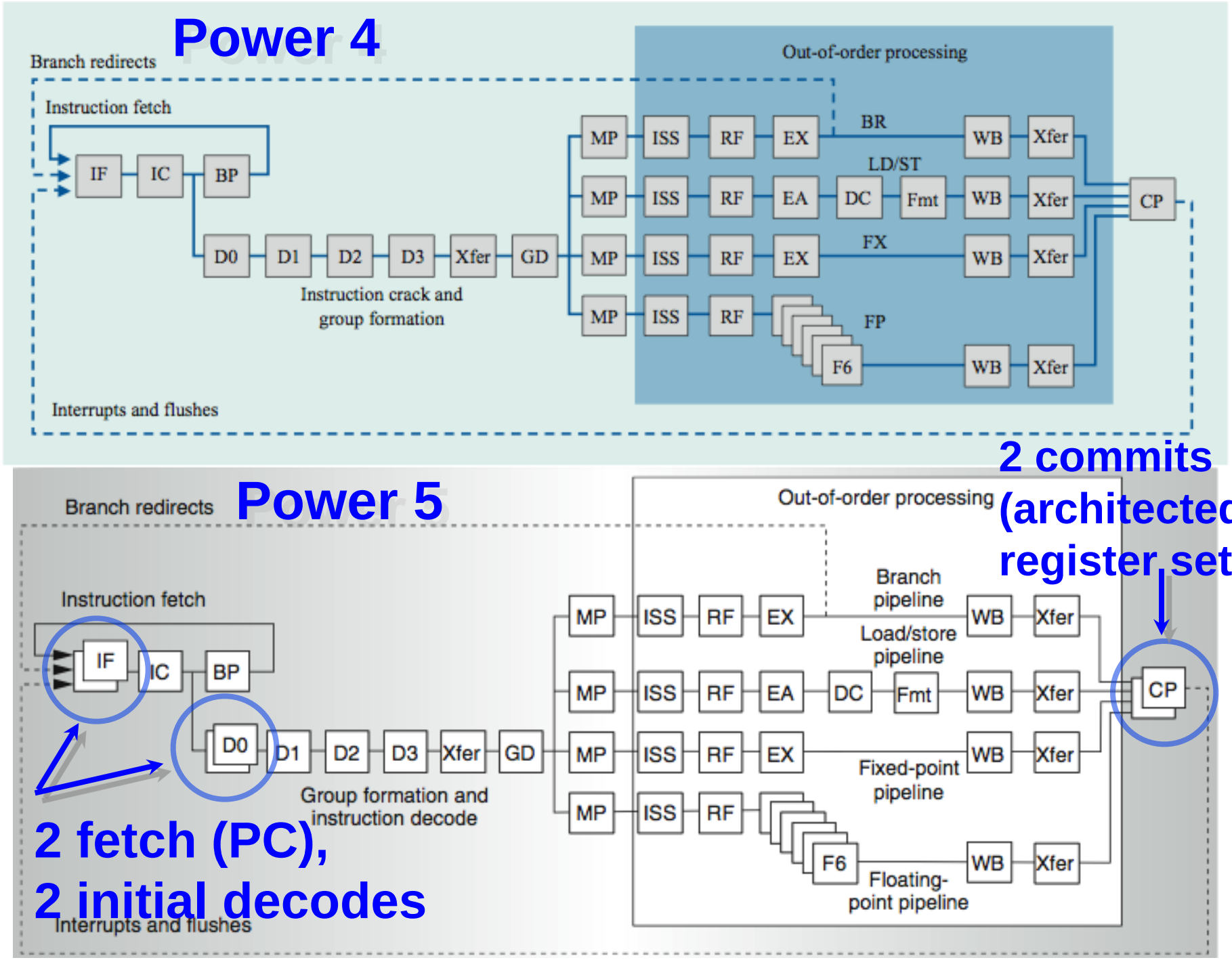
- Large set of virtual registers can hold register sets of independent threads
- Renaming provides unique register identifiers to different threads
- Out-of-order completion of instructions from different threads allowed
  - No cross-thread RAW, WAW, WAR hazards
- Separate reorder buffer per thread

Summary: Multithreaded Categories

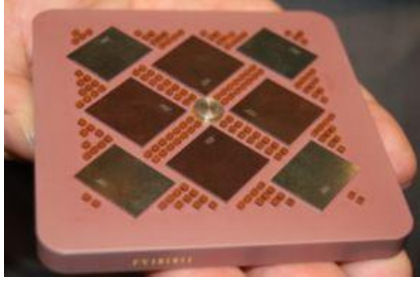
Power 4

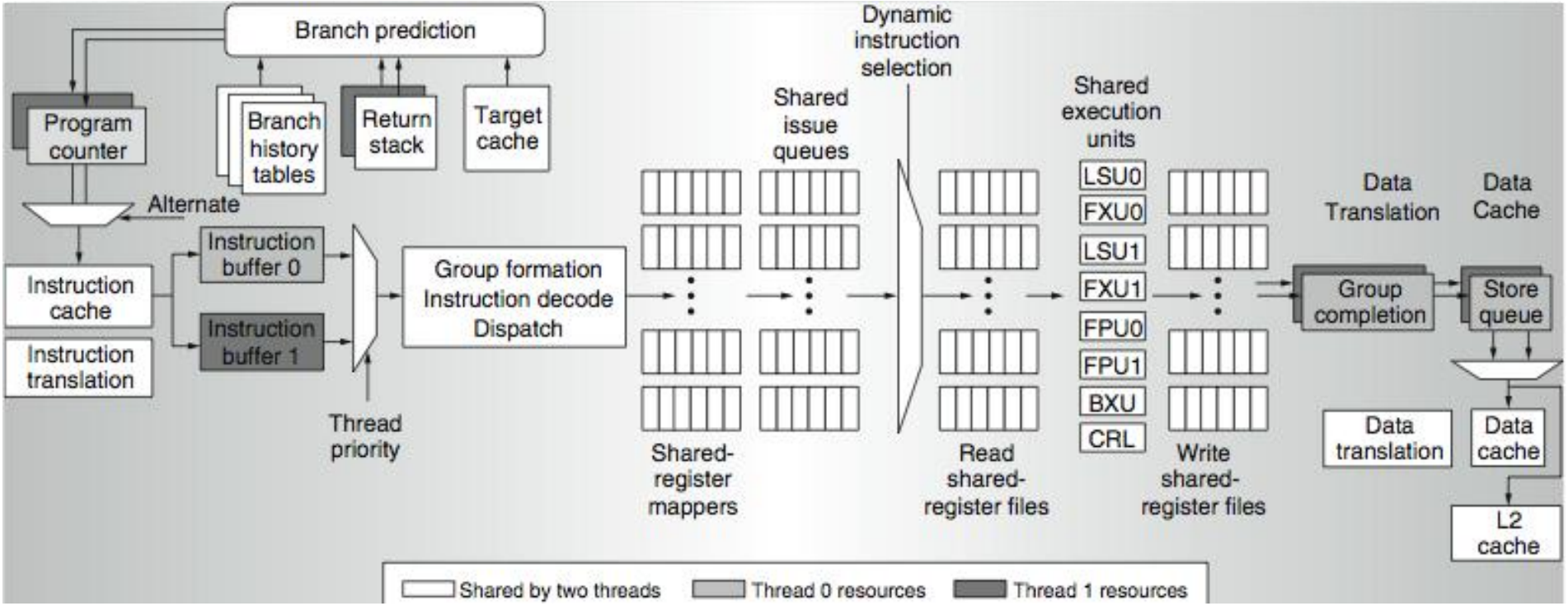
Single-threaded predecessor to Power 5. 8 execution units in out-of-order engine, each may issue an instruction each cycle.




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

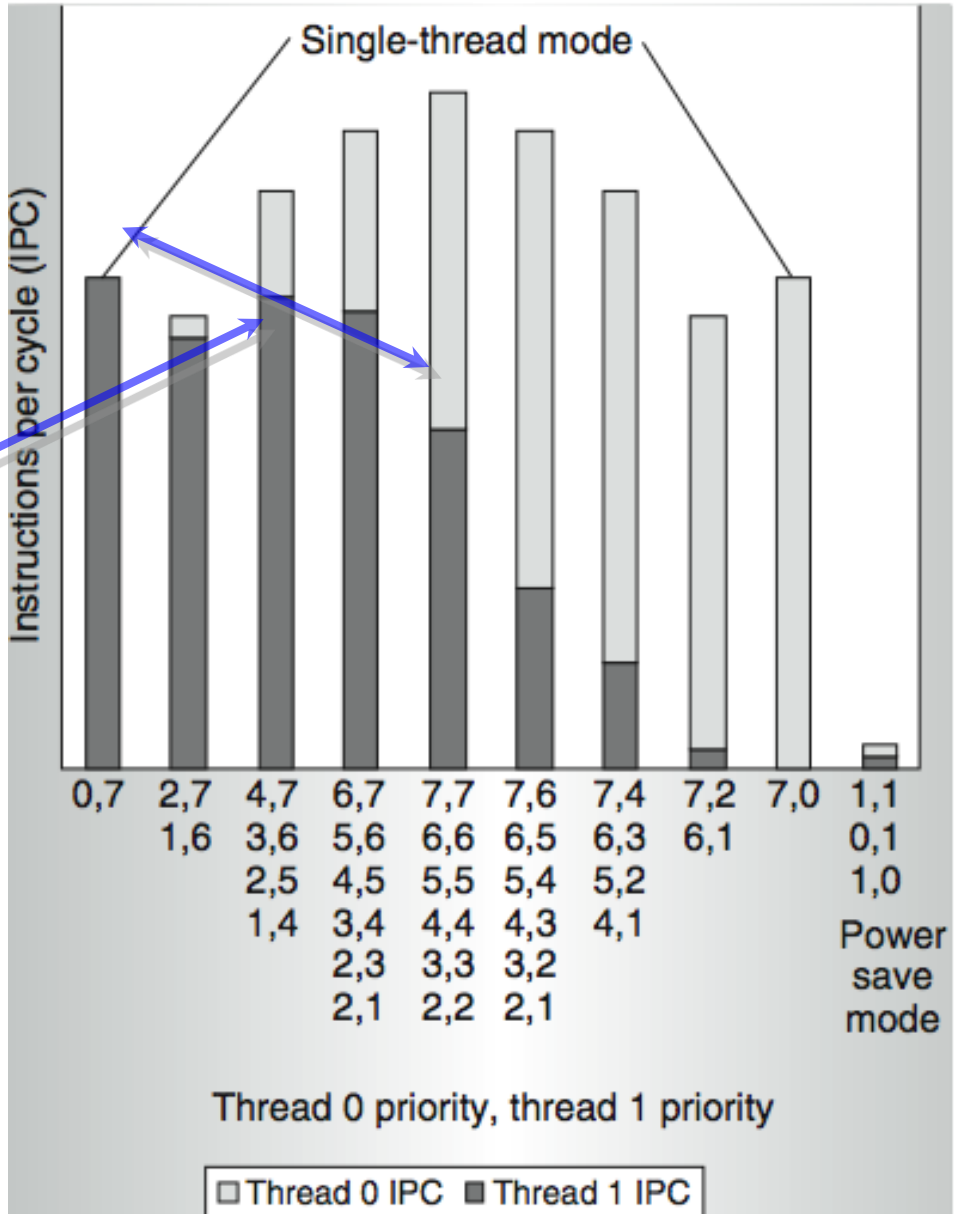
Changes in Power 5 to support SMT

- 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

Power 5 thread performance ...

Relative priority of each thread controllable in hardware.

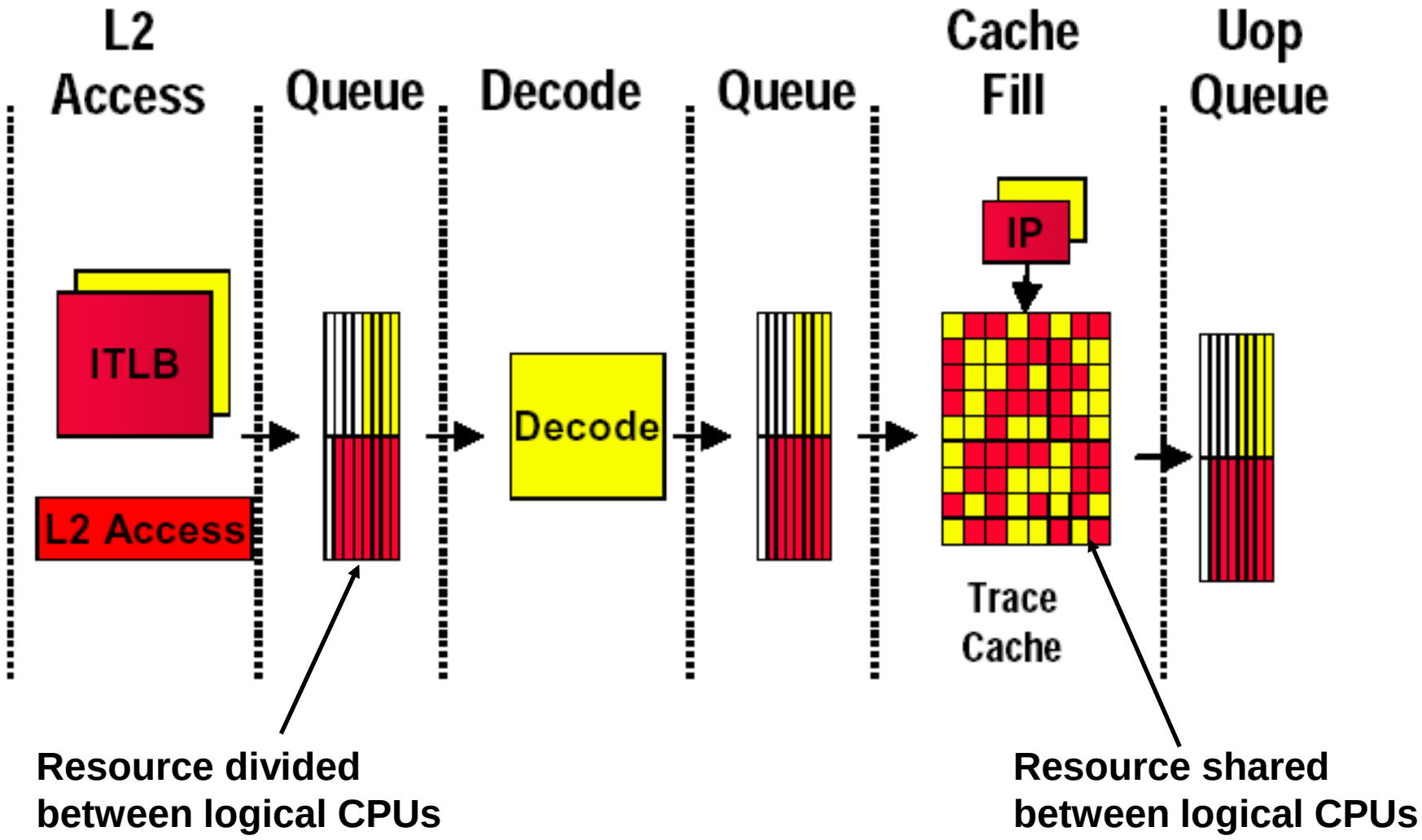
For balanced operation, both threads run slower than if they "owned" the machine.



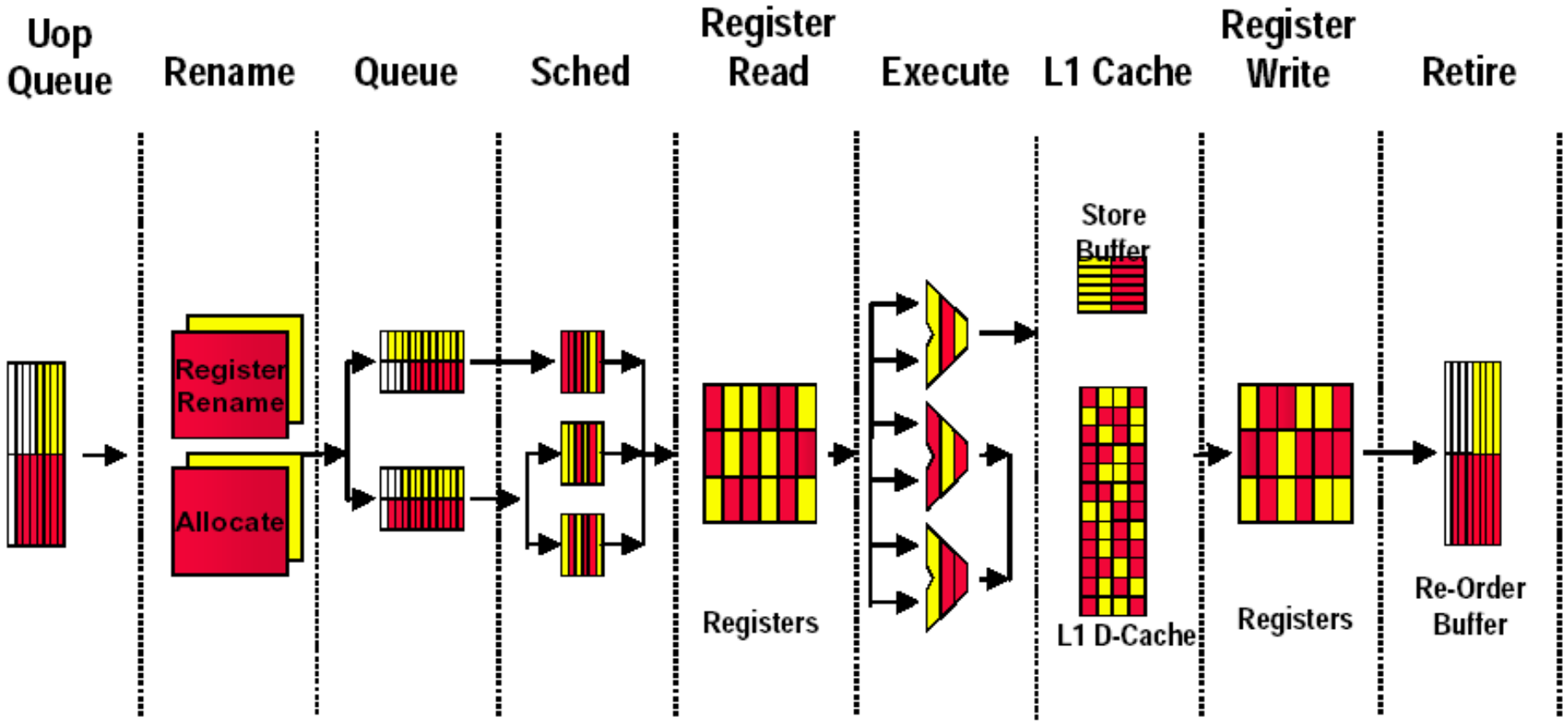
Pentium-4 Hyperthreading (2002)

- 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



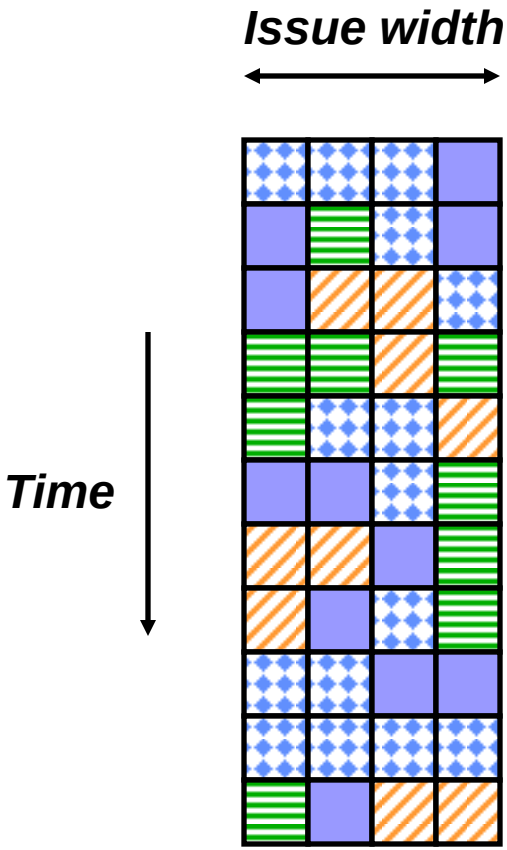
Pentium-4 Hyperthreading
Execution Pipeline



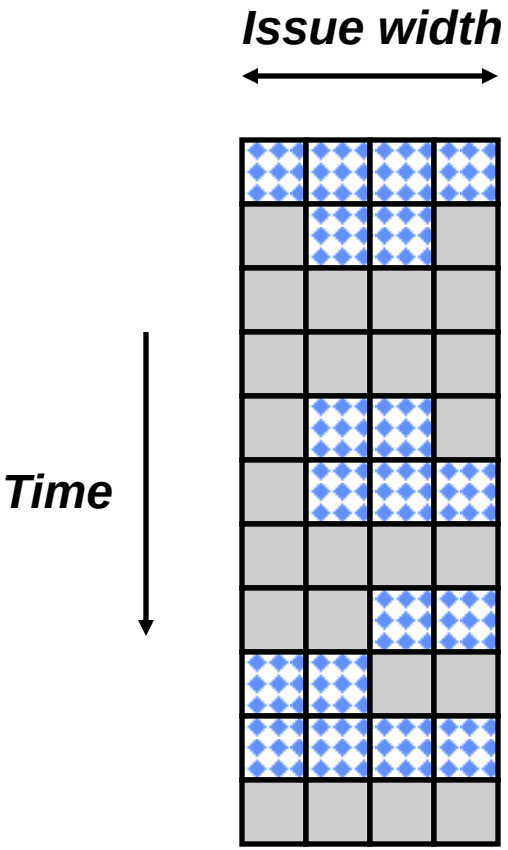
[Intel Technology Journal, Q1 2002]

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

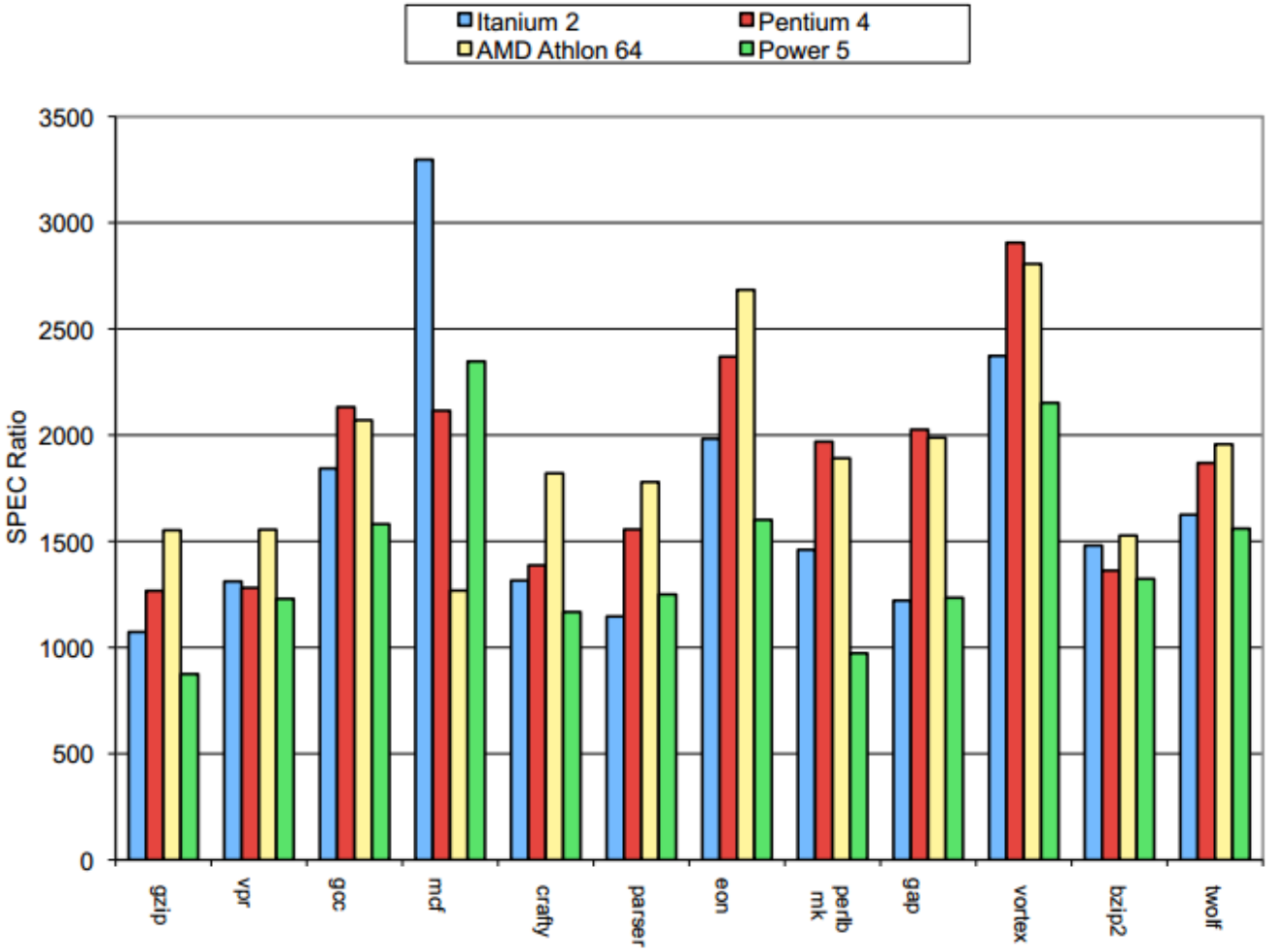
- Pentium 4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on Pentium 4 each of 26 SPEC benchmarks paired 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

Comparison between ILP processors

Processor	Micro architecture	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transis-tors Die size	Power
Intel Pentium 4 Extreme	Speculative dynamically scheduled; deeply pipelined; SMT	3/3/4	7 int. 1 FP	3.8	125 M 122 mm^2	115 W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3 FP	2.8	114 M 115 mm^2	104 W
IBM Power5 (1 CPU only)	Speculative dynamically scheduled; SMT; 2 CPU cores/chip	8/4/8	6 int. 2 FP	1.9	200 M 300 mm^2 (est.)	80W (est.)
Intel Itanium 2	Statically scheduled VLIW-style	6/5/11	9 int. 2 FP	1.6	592 M 423 mm^2	130 W

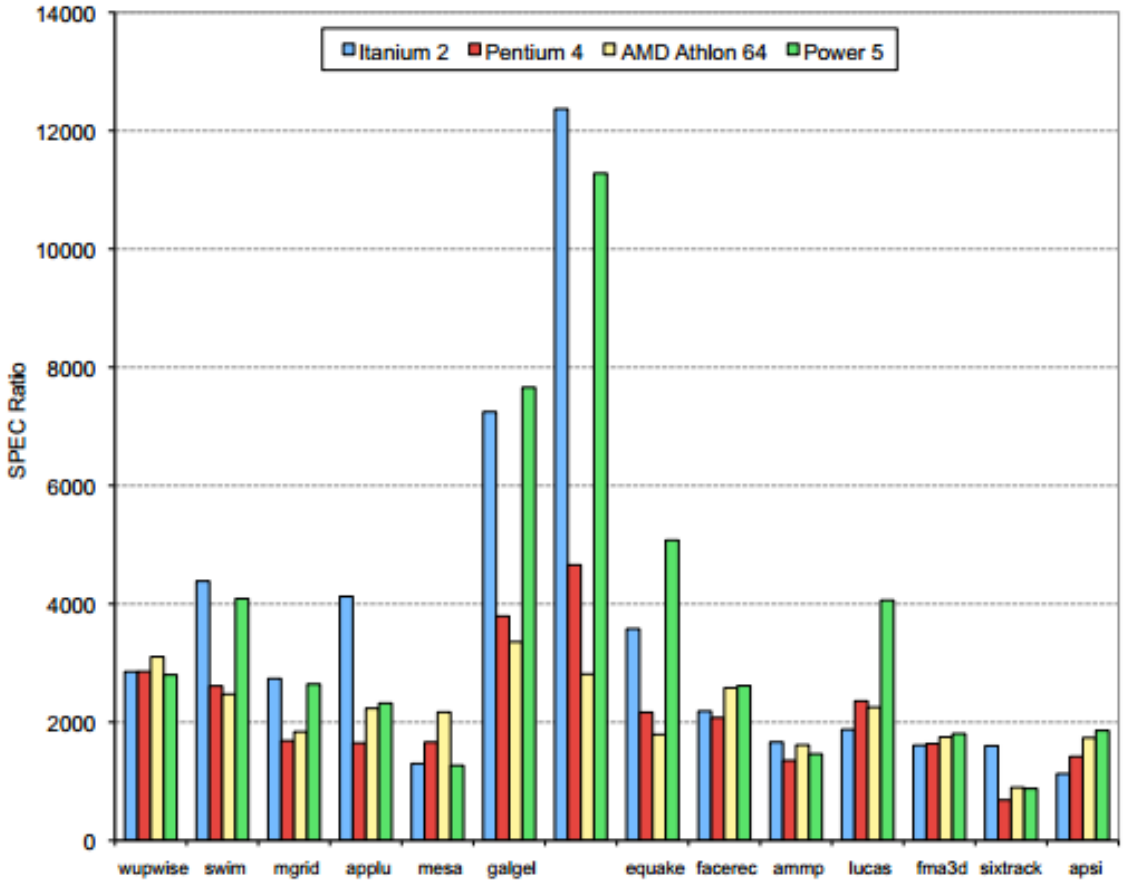
Comparison between ILP processors

SPEC INT rate



Comparison between ILP processors

SPEC FP rate



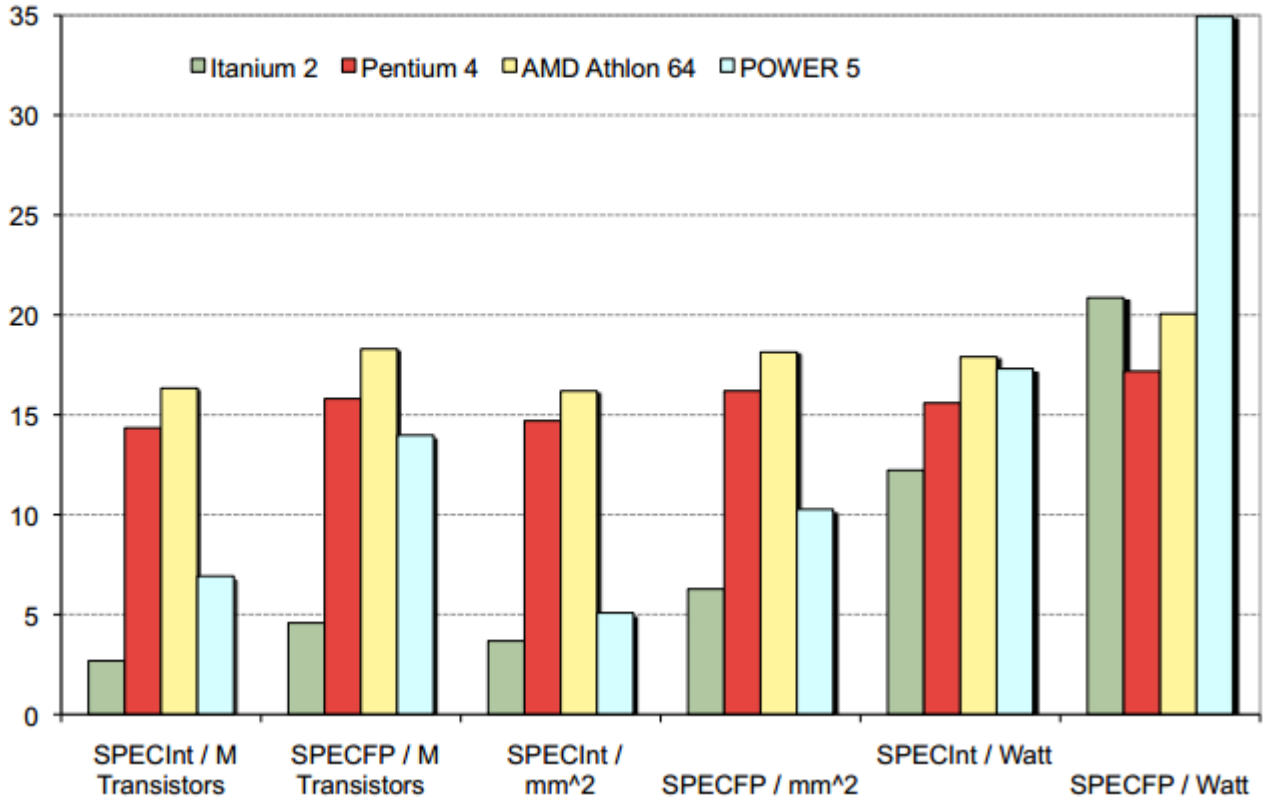
Measuring processor efficiency

Area- and power-efficiency

- Processor performance gain comes at an area/power budget cost
  - Weigh performance again against power and area increase
- Area-efficiency
  - Performance / transistor (e.g. SPECrate/million transistors)
- Power-efficiency
  - Performance / watt (e.g. SPECrate/watt)

Comparison between ILP processors

Power and area efficiency



Best ILP approach?

Results with commercial processors

- AMD Athlon most performance-efficient in INT programs
- Power5 most performance-efficient in FP programs
- Power5 most power-efficient overall
- Itanium VLIW least power-efficient and area-efficient overall