CS425
Computer Systems Architecture

Fall 2017

Thread Level Parallelism (TLP)

Multiple Issue

$CPI = CPI_{IDEAL} + Stalls_{STRUC} + Stalls_{RAW} + Stalls_{WAR} + Stalls_{WAW} + Stalls_{CONTROL}$

- Have to maintain:
  - **Data Flow**
  - **Exception Behavior**

Dynamic instruction scheduling (HW)	Static instruction scheduling (SW/compiler)
Scoreboard (reduce RAW stalls)	Loop Unrolling
Register Renaming (reduce WAR & WAW stalls) •**Tomasulo** • **Reorder buffer**	SW pipelining
Branch Prediction (reduce control stalls)	Trace Scheduling
Multiple Issue (CPI < 1) **Multithreading** (CPI < 1)	

Common Way of Designing Architectures

- Networking, single/multi-core processor, virtually any design:
  - **Broadcasting:** Use Common Data Bus or Point to point
  - **Asynchronous communication** between "processing stages" with different throughputs (a processing stage can be a whole system, for example router, switch, processor, or a simple block, for example IF, ID stages). Use Elastic Buffer & Flow Control. For example instruction buffer, reservation stations and reorder buffer
  - **Faster clock:** Pipelining. Split a stage in multiple stages. For example split Issue stage (super-pipelining)
  - **Higher Throughput:** Parallel processing. For example superscalar.
  - **Lower Latency:** Forwarding/Bypassing
- A processor is a sophisticated design that follows the "unwritten" design rules every architect should follow.

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

	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
LW r1, 0(r2)	F	D	X	M	W										
LW r5, 12(r1)		F	D	D	D	D	X	M	W						
ADDI r5, r5, #12			F	F	F	F	D	D	D	D	X	M	W		
SW 12(r1), r5							F	F	F	F	D	D	D	D	

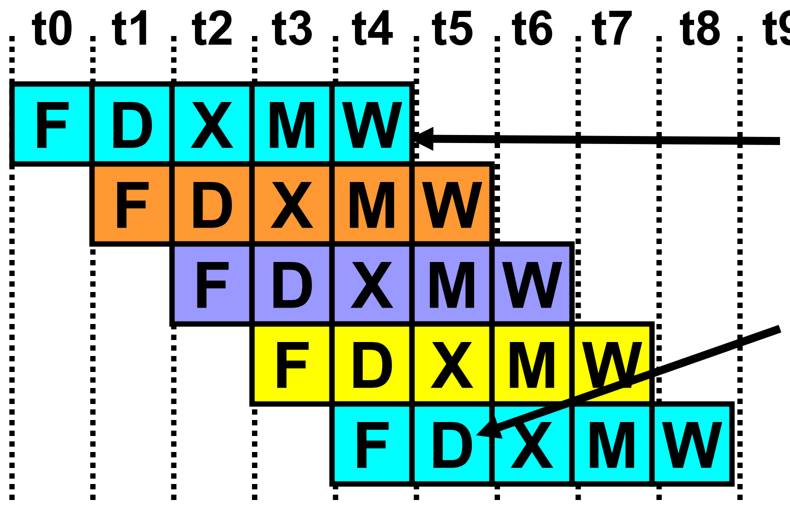
- Each instruction may depend on the next
- What can be done to cope with this?

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

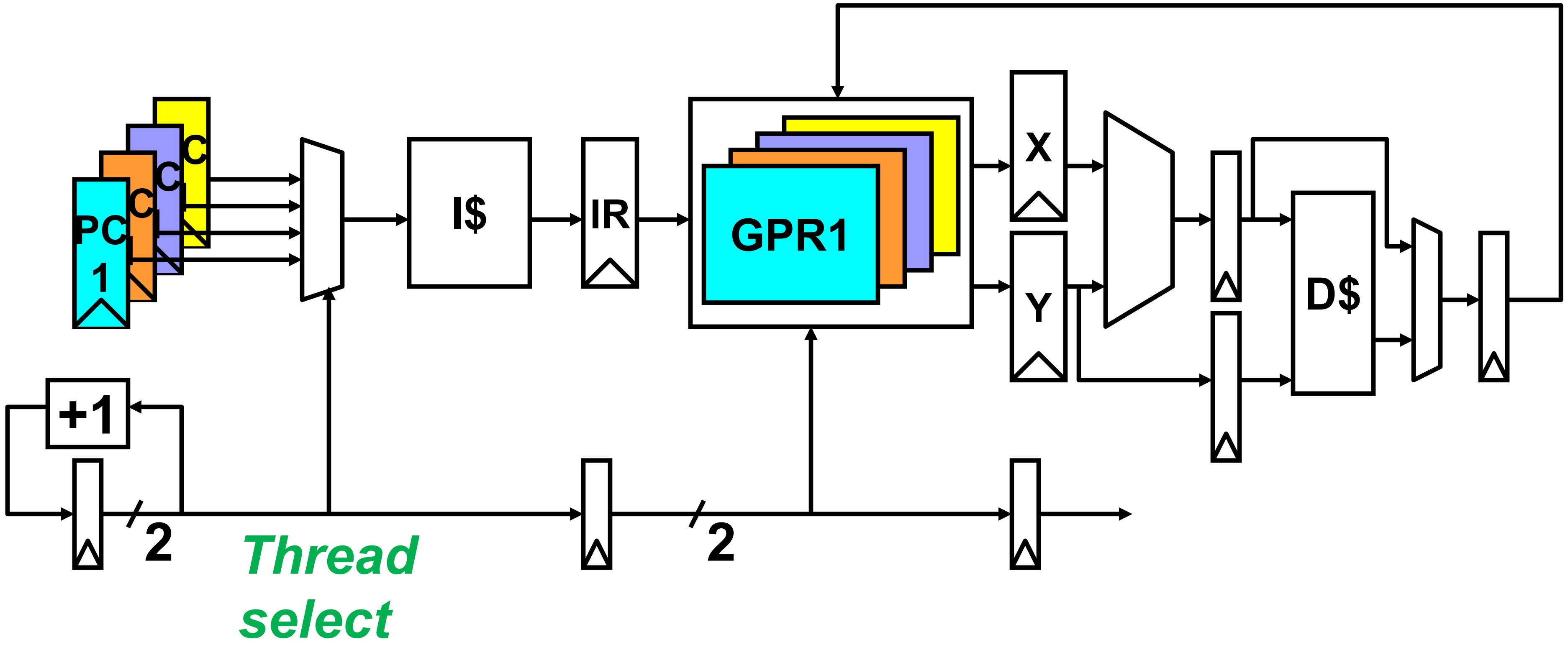
Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe

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)



Prior instruction in a thread always completes write-back before next instruction in same thread reads register file

Multithreaded DLX



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

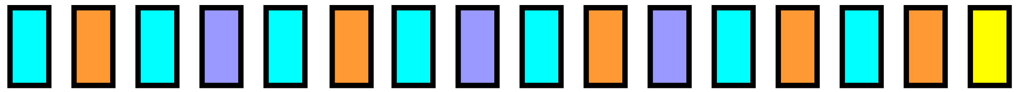
Multithreading Cost

- Each thread requires its own user state. **Many CPU resources are split or shared!**
  - PC
  - GPRs & Physical/HW registers
  - Prefetch & Instruction buffers
  - Reorder buffer
  - Load/Store buffer
  - Issue buffers
- Also, needs its own system state
  - virtual memory page table base register
  - exception handling registers

- *Other costs?*
- *Take care of performance when executing in Single Thread (ST) mode!*

Thread Scheduling Policies

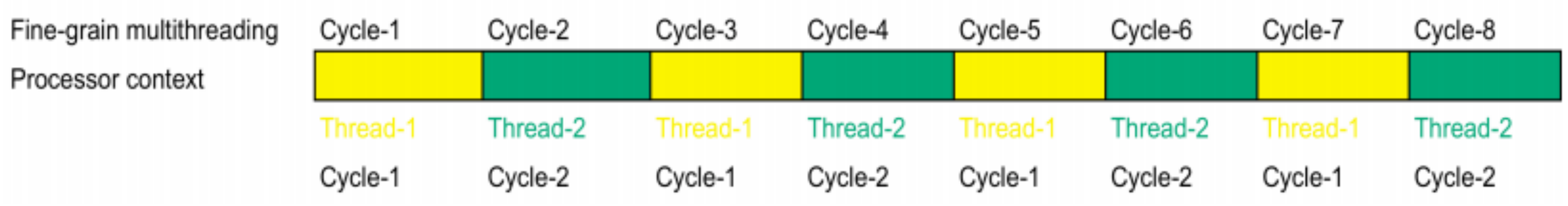
- Fixed interleaving *(CDC 6600 PPUs, 1964)*
  - each of N threads executes one instruction every N cycles
  - if thread not ready to go in its slot, insert pipeline bubble
- Software-controlled interleave *(TI ASC PPUs, 1971)*
  - OS allocates S pipeline slots amongst N threads
  - hardware performs fixed interleave over S slots, executing whichever thread is in that slot



- Hardware-controlled thread scheduling *(HEP, 1982) (Power 5)*
  - 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

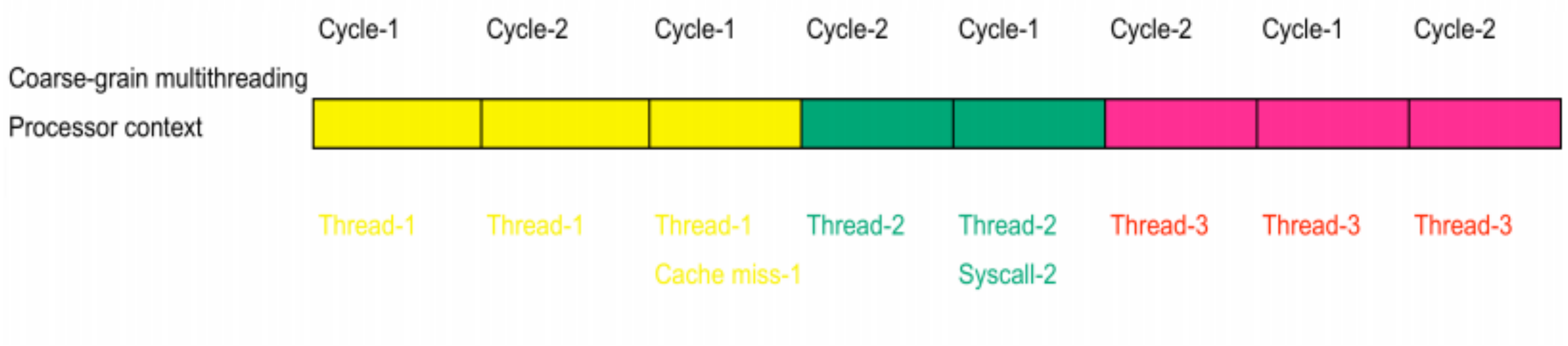
- **Fine-Grain Multithreading**

  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

- **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**

  Switch upon long upon long-latency events

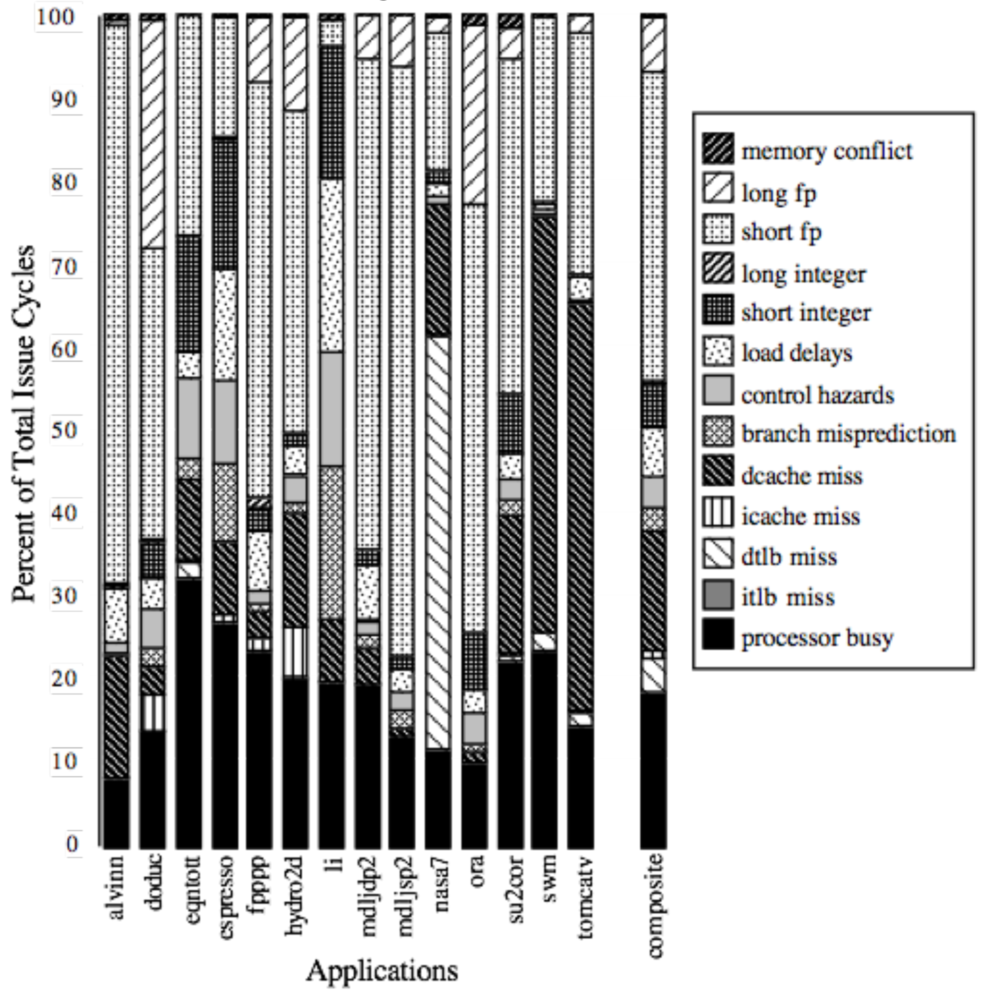
  - Can afford slower context switch than fine-grain multithreading
  - Threads are not slowed down
    - Other thread runs when current thread stalls
  - Pipeline startup cost upon thread switching
    - Processor issues instructions from one thread (address space)

HW Multithreading alternatives

- **Simultaneous Multithreading (SMT)**
- 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.

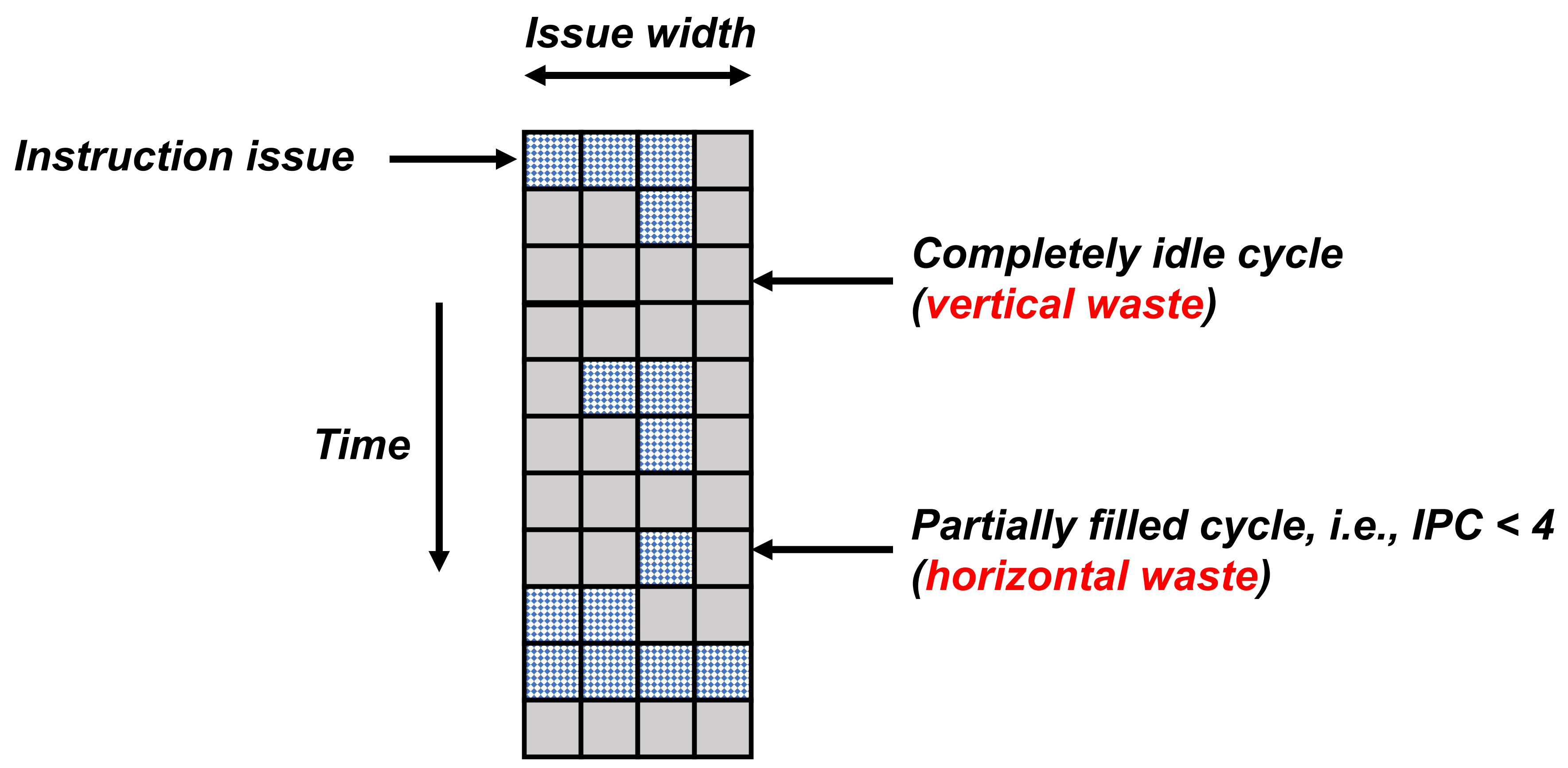
For most apps, most execution units are idle in an OoO superscalar

For an 8-way superscalar

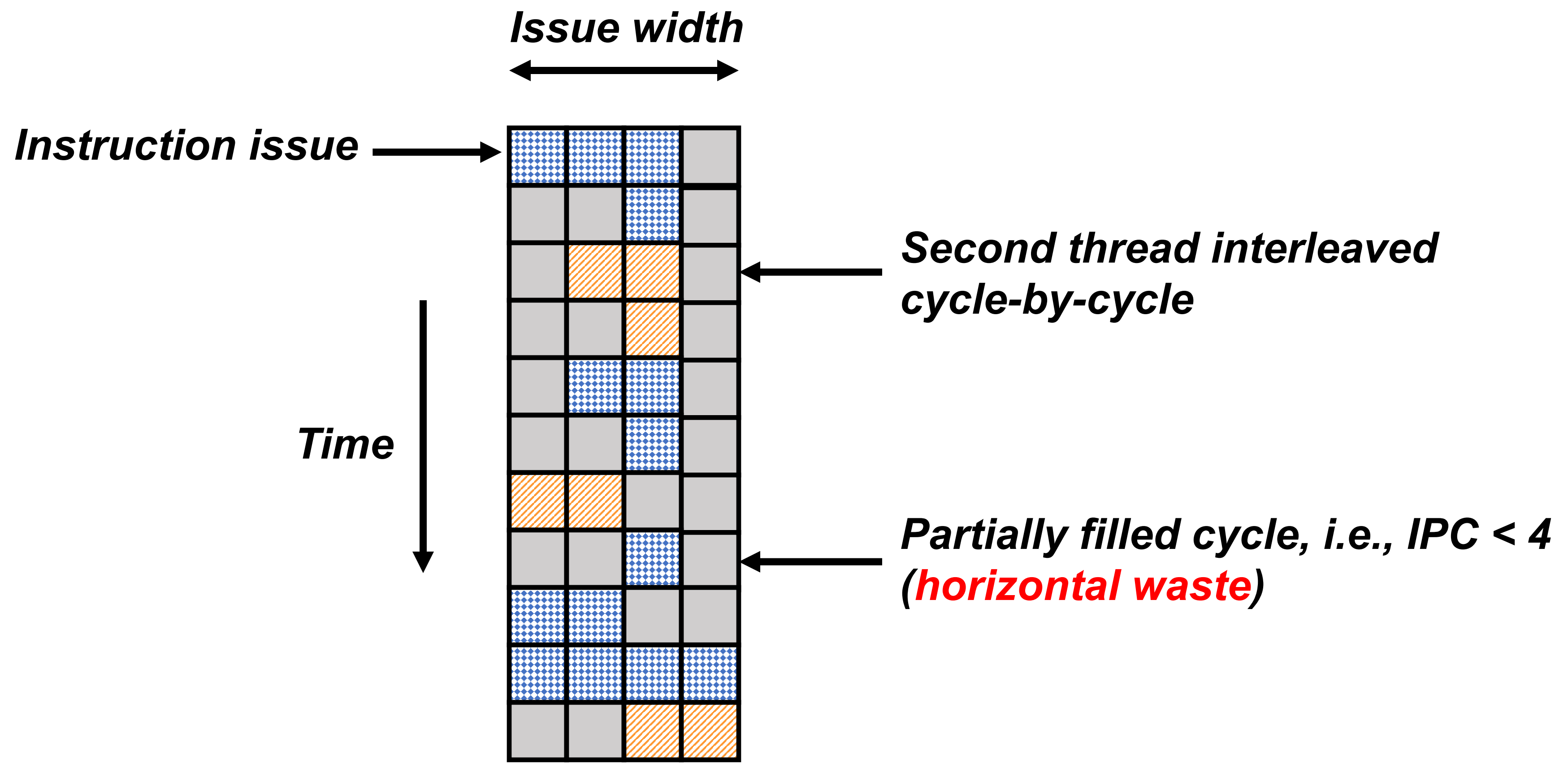


From: Tullsen, Eggers, and Levy, "Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.

Superscalar Machine Efficiency



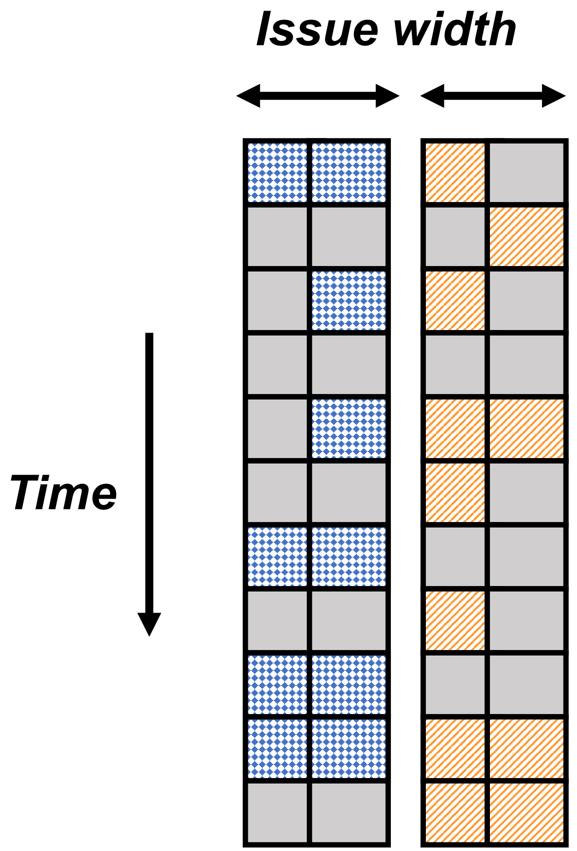
Vertical Multithreading



Issue width
Instruction issue
Time
Second thread interleaved cycle-by-cycle
Partially filled cycle, i.e., IPC < 4 (horizontal waste)

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

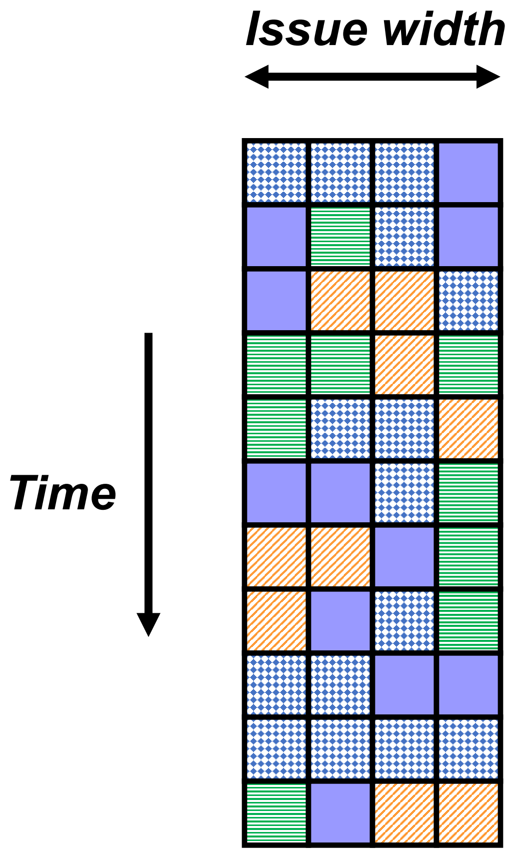
Chip Multiprocessing (CMP)



- 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** → **single thread execution is slower**

Ideal Superscalar Multithreading: SMT

[Tullsen, Eggers, Levy, UW, 1995]

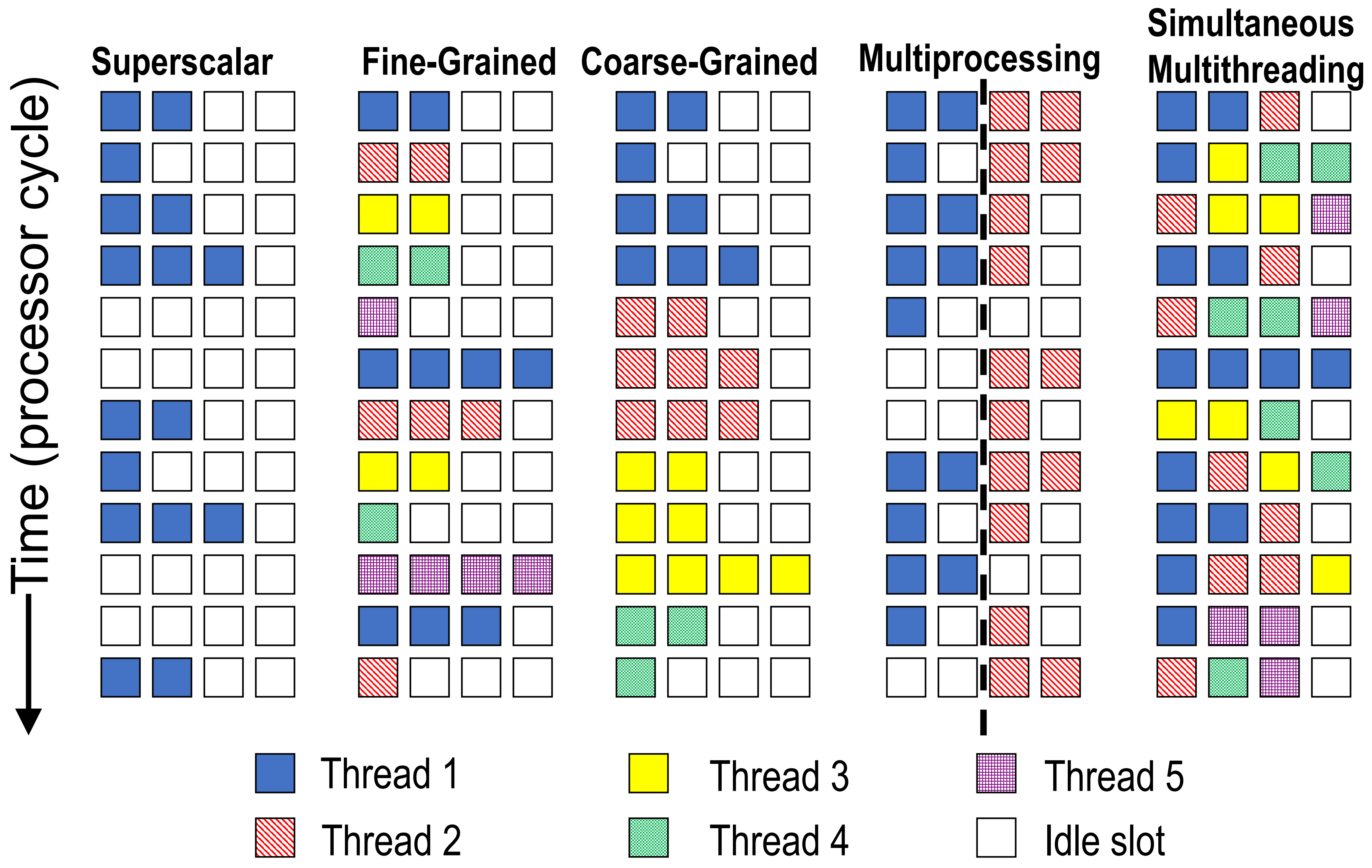


- Interleave multiple threads to multiple issue slots with no restrictions

O-o-O Simultaneous Multithreading
[Tullsen, Eggers, Emer, Levy, Stamm, Lo, DEC/UW, 1996]

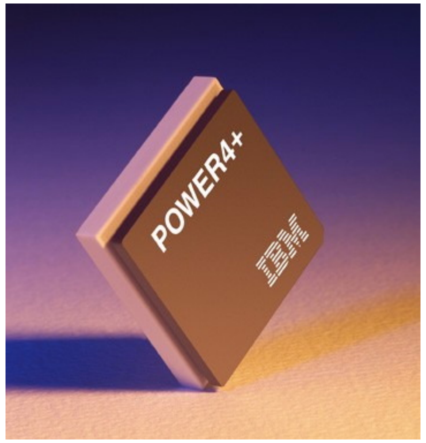
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OOO instruction window already has most of the circuitry required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

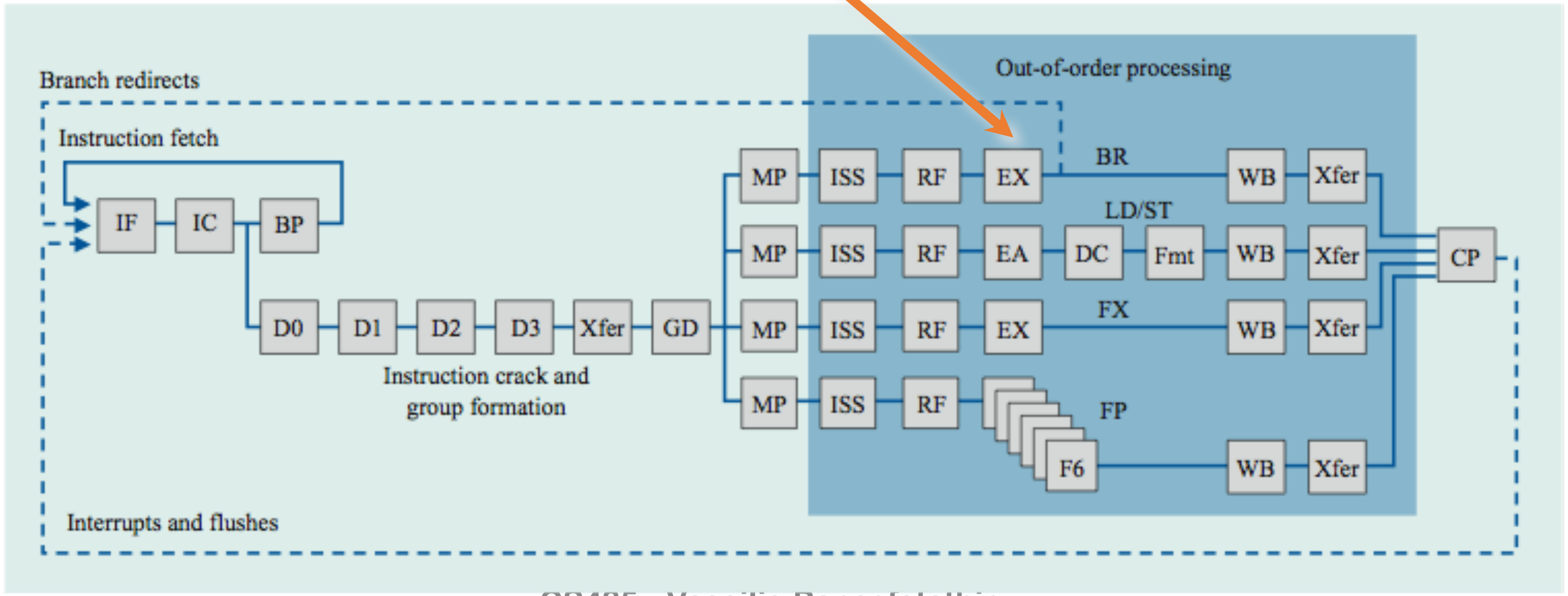
Summary: Multithreaded Categories

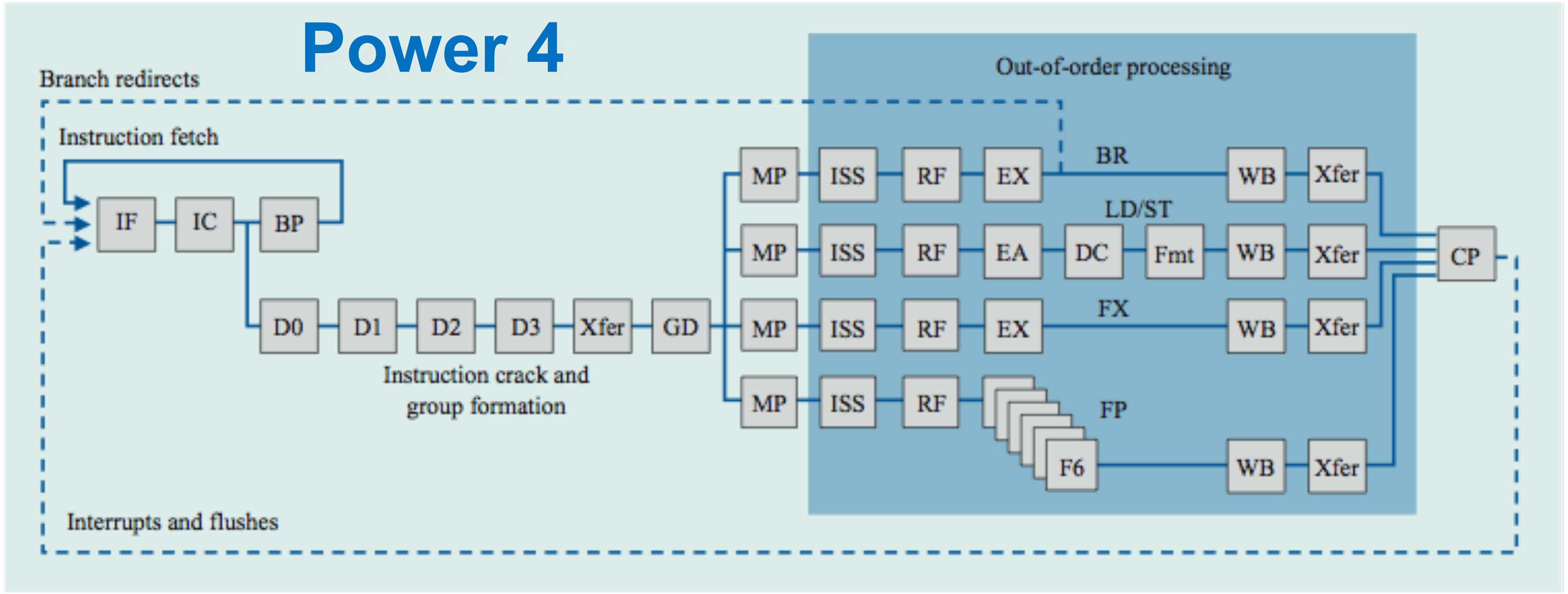


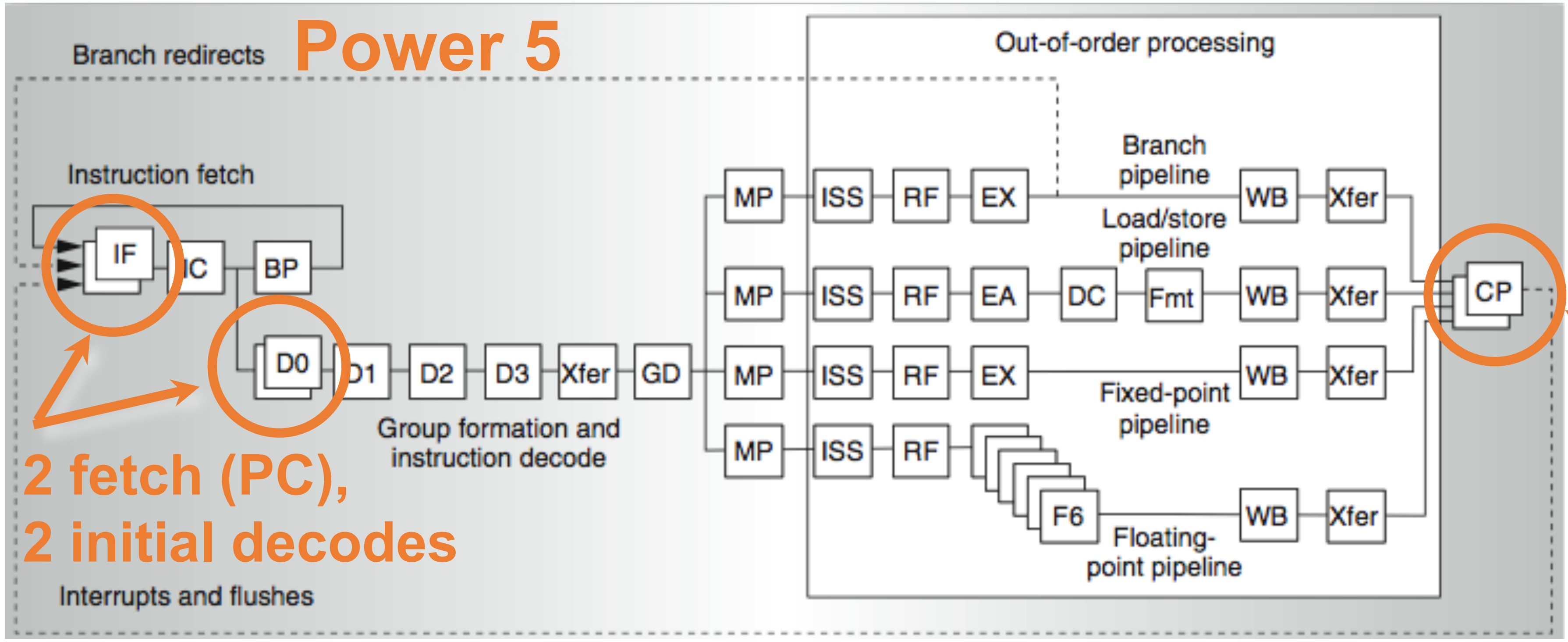
IBM Power 4

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





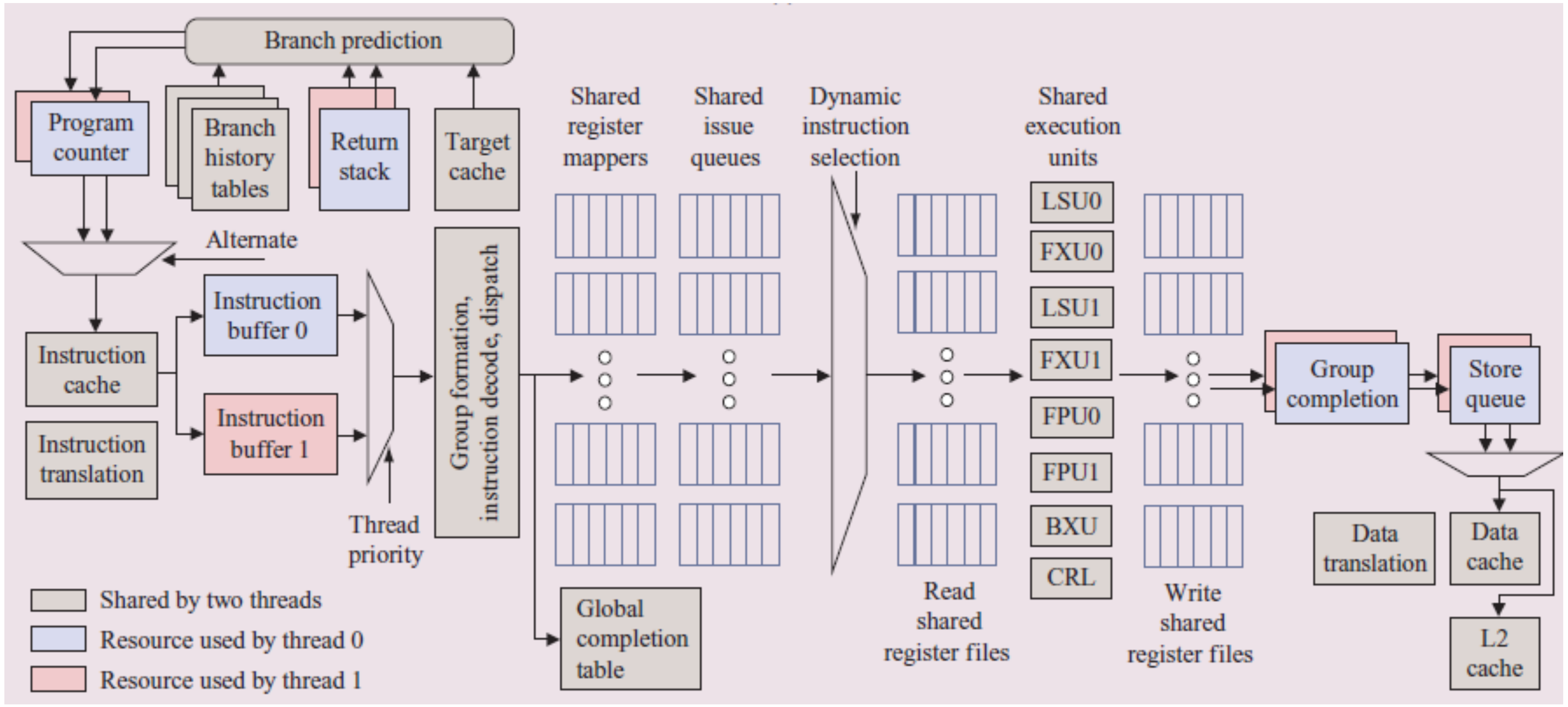


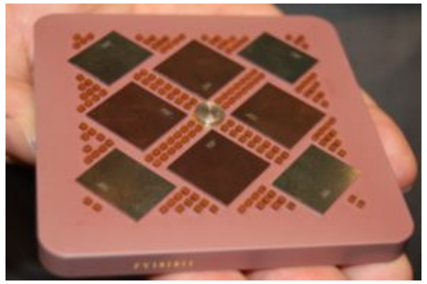


2 fetch (PC),
2 initial decodes

2 commits
(architected
register sets

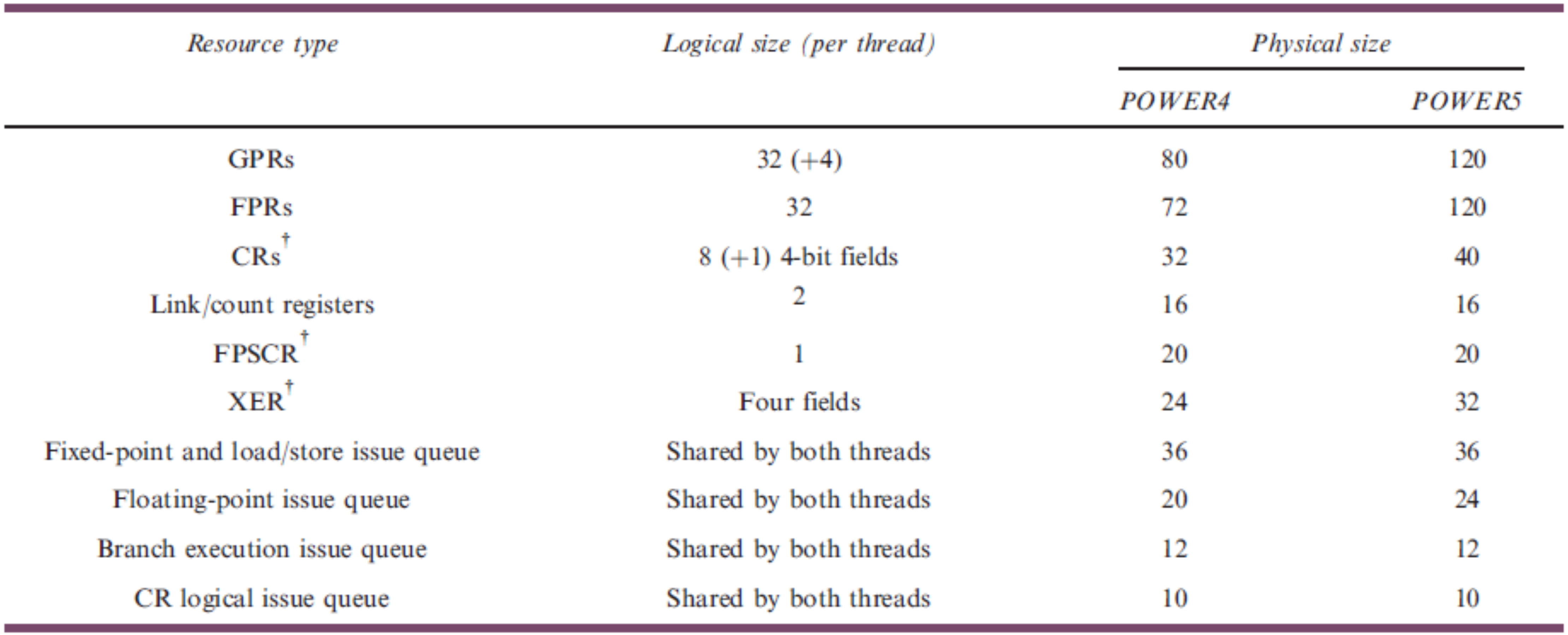
Power 5 data flow ...





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

Rename Registers and issue queue sizes



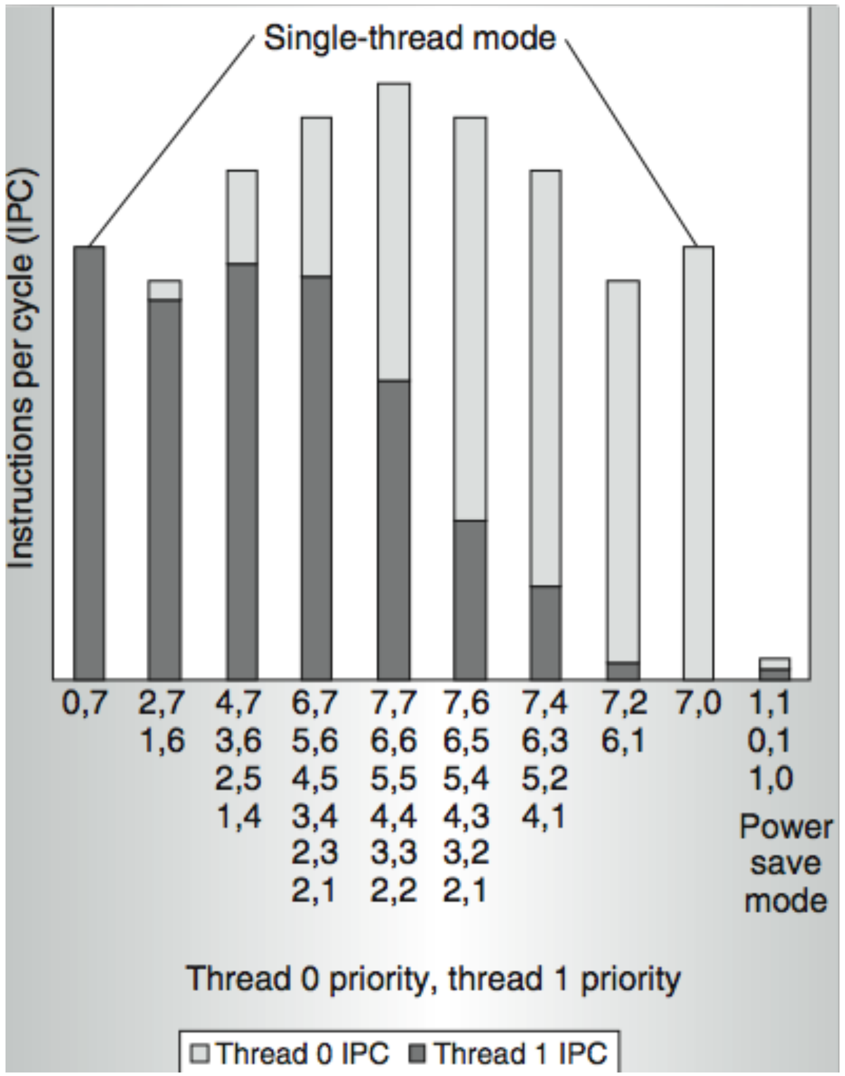
Resource type	*Logical size (per thread)*	*Physical size*	
		POWER4	*POWER5*
GPRs	32 (+4)	80	120
FPRs	32	72	120
CRs[†]	8 (+1) 4-bit fields	32	40
Link/count registers	2	16	16
FPSCR[†]	1	20	20
XER[†]	Four fields	24	32
Fixed-point and load/store issue queue	Shared by both threads	36	36
Floating-point issue queue	Shared by both threads	20	24
Branch execution issue queue	Shared by both threads	12	12
CR logical issue queue	Shared by both threads	10	10

Changes in Power 5 to support SMT

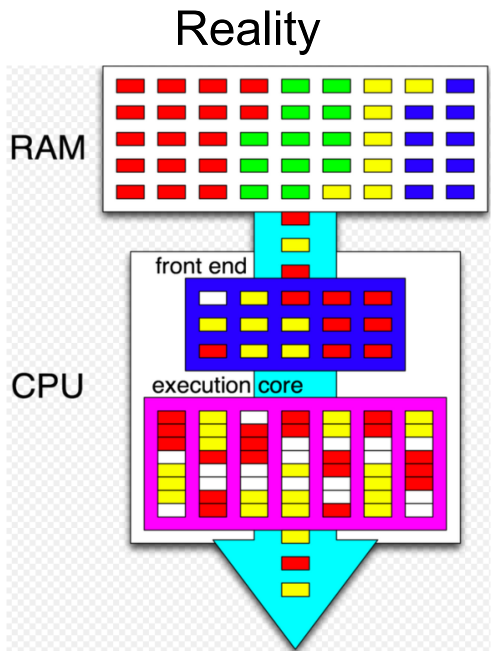
- Two separate program counters are used, one for each thread
- Added per thread load and store queues. Added virtual entries.
- The size of the BIQ (Branch Information Queue) remains at 16 entries but split in two, with eight entries per thread.
- Added separate instruction prefetch and buffering per thread.
- Each logical register number has a thread bit appended and mapped as usual. Increased the number of physical registers from 152 to 240
- Increased the size of FP issue queue.
- Shared global completion table (GCT). Two linked lists to implement in order commit from the two threads.
- The Power5 core is about 24% larger than the Power4 core because of the addition of SMT support

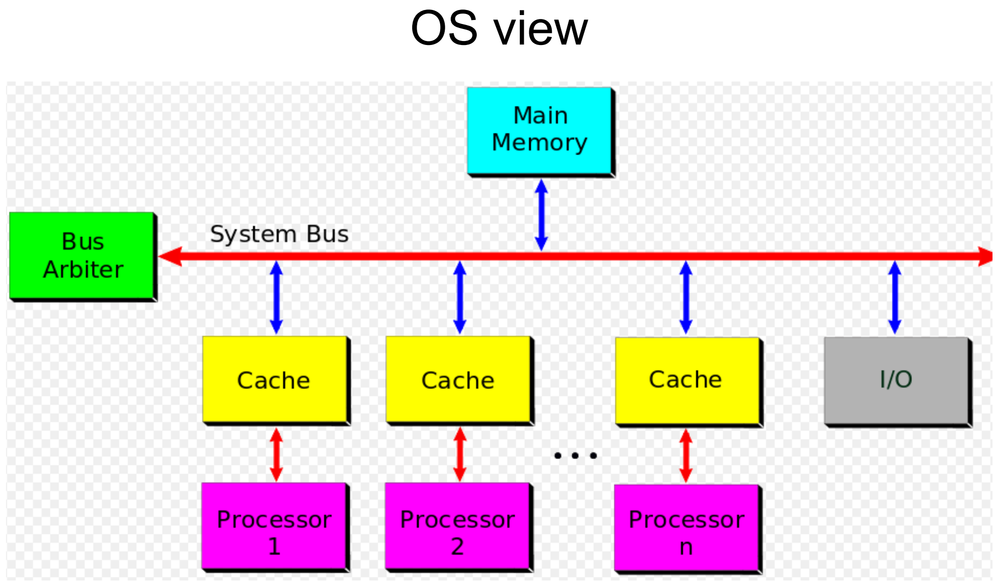
Power 5 thread performance

- Priority is set by SW and enforced by HW.
- Relative priority of each thread controllable in hardware.
- For balanced operation, both threads run slower than if they “owned” the machine.



Intel Hyper-Threading Technology



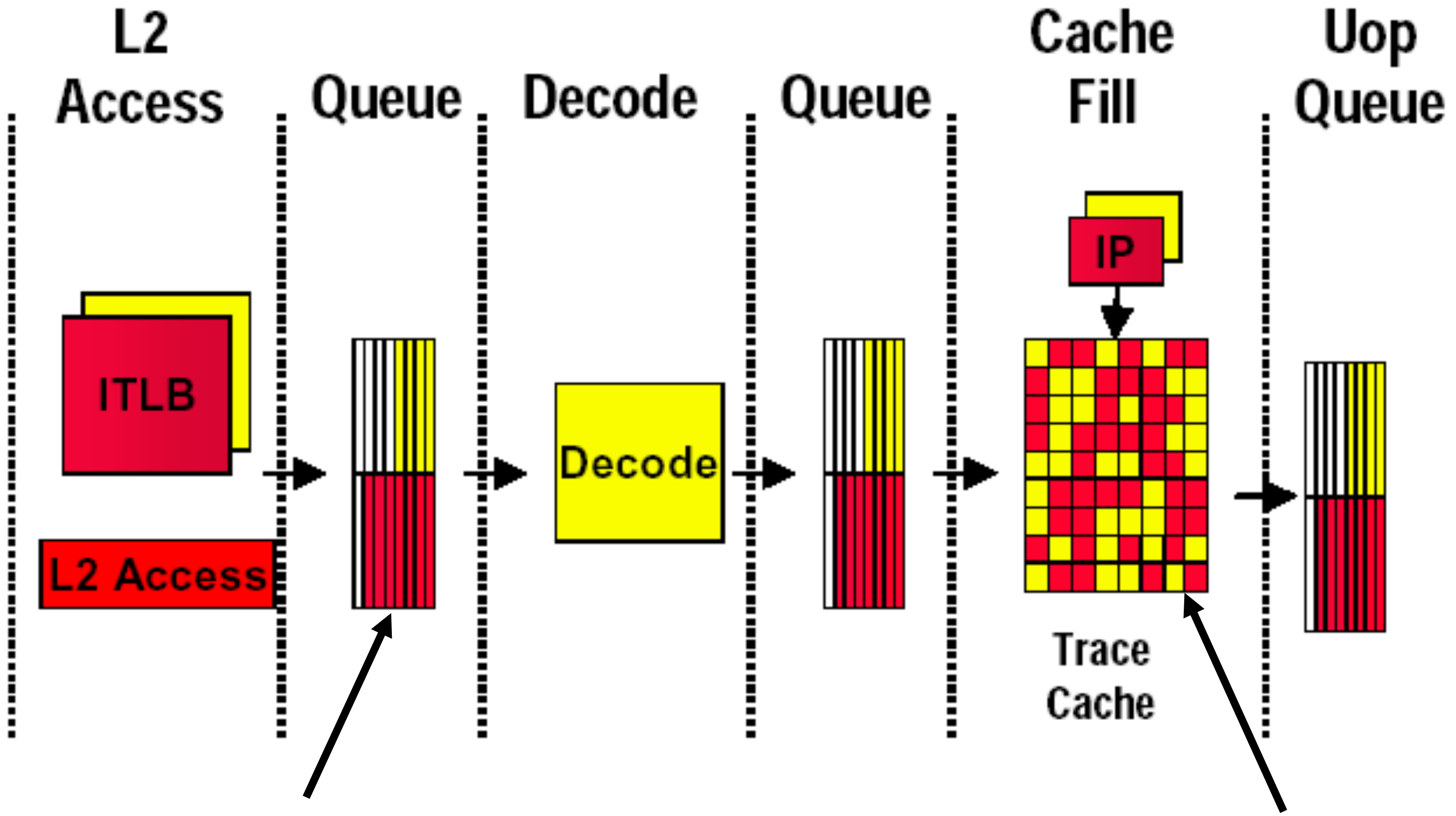


- Hyper-Threading Technology is SMT introduced by Intel. HTT has two logical processors, with its own processor architectural state
- HTT duplicates the architectural state but not the main execution resources
- Transparent to OS: minimum requirement is symmetric multiprocessing (SMP) support
- SMP involves two or more identical processors connect to a single, shared main memory, all I/O devices, controlled by single OS

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 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

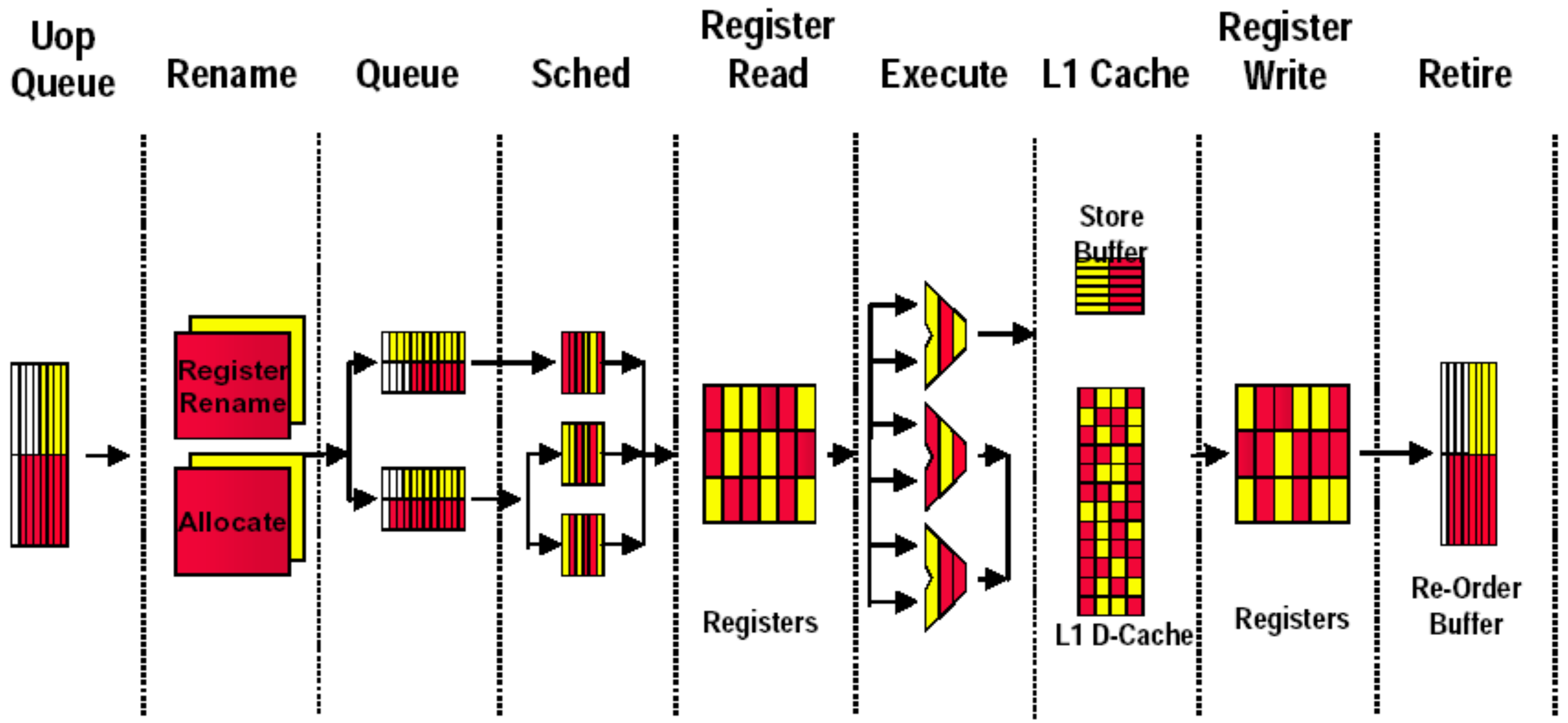
Pentium-4 Hyperthreading: Front End



Resource divided between logical CPUs

Resource shared between logical CPUs

Pentium-4 Hyperthreading: Execution Pipe



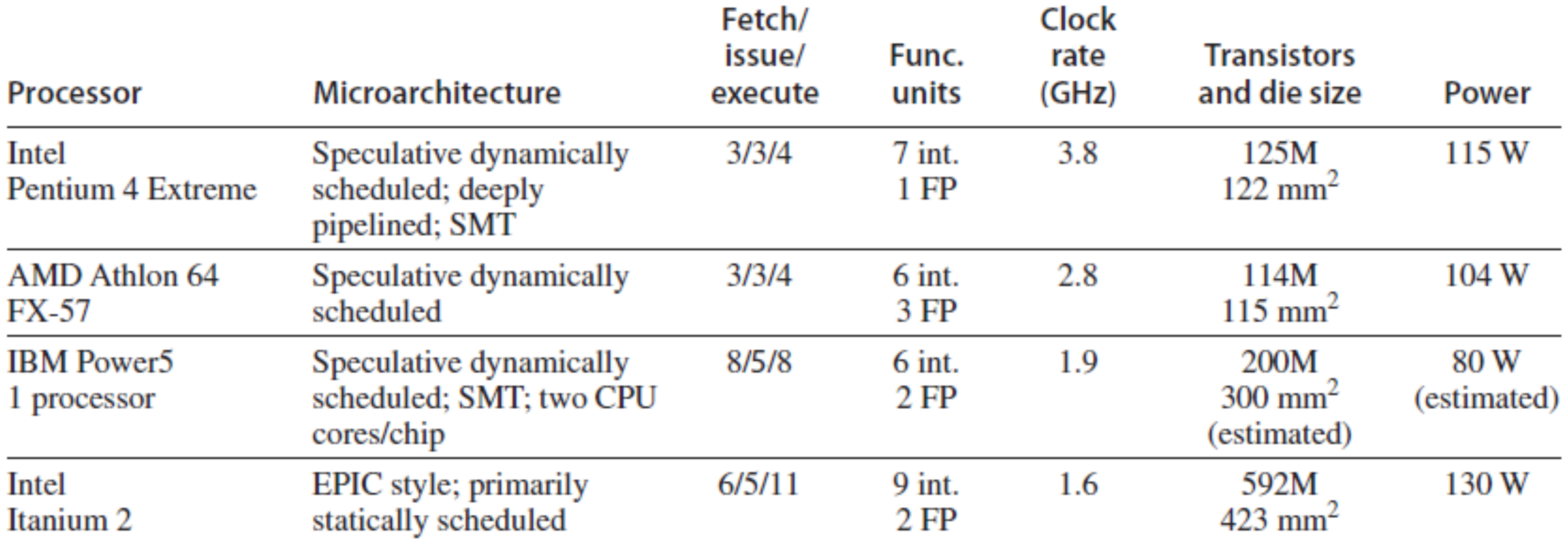
[Intel Technology Journal, Q1 2002]

Initial Performance of SMT

Multi-program workloads

- Pentium 4 Extreme SMT achieves 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 is 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 FP apps had cache conflicts and least gains

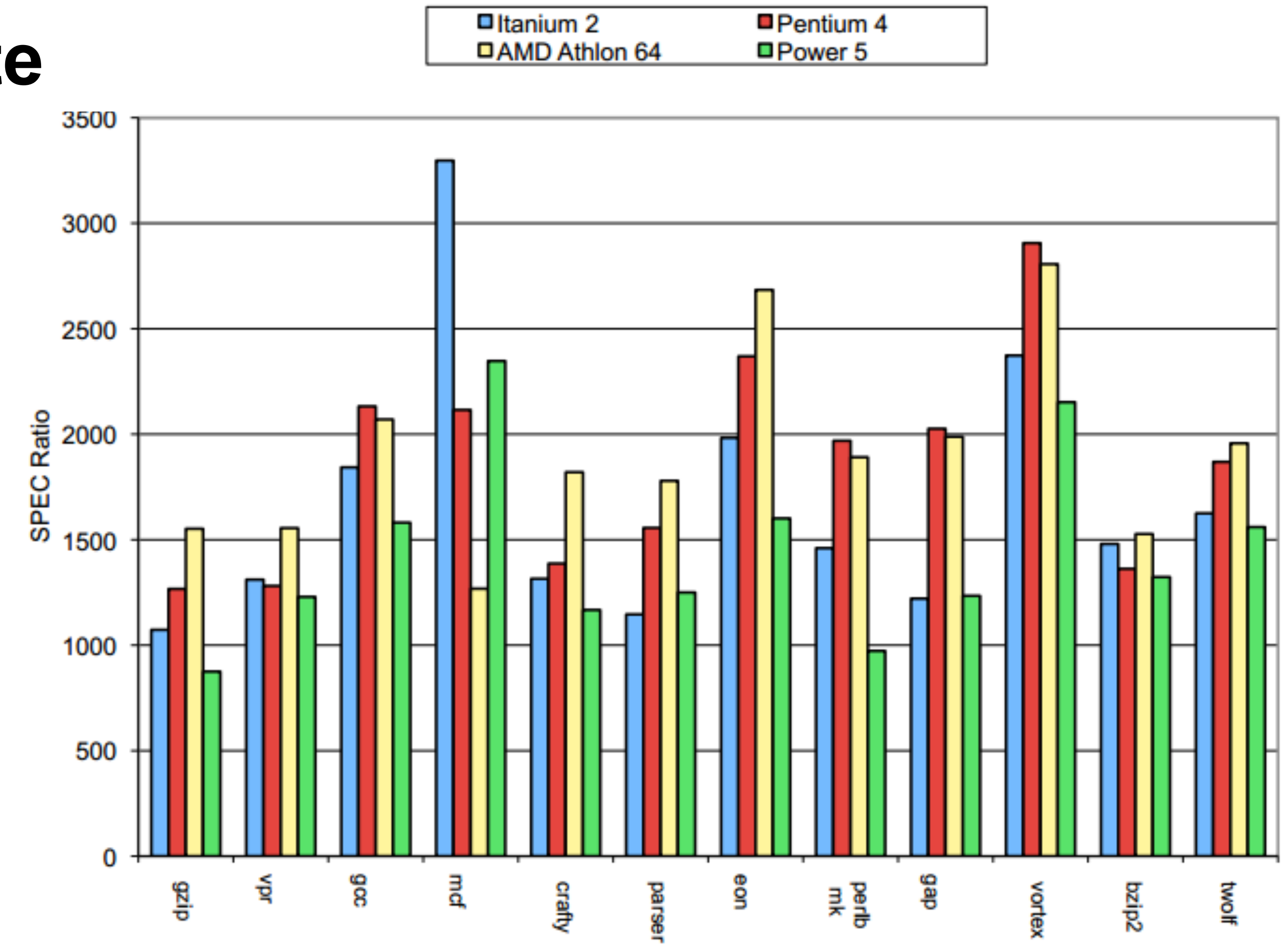
Comparison of multiple-issue processors



Processor	Microarchitecture	Fetch/ issue/ execute	Func. units	Clock rate (GHz)	Transistors and die size	Power
Intel Pentium 4 Extreme	Speculative dynamically scheduled; deeply pipelined; SMT	3/3/4	7 int. 1 FP	3.8	125M 122 mm^2	115 W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3 FP	2.8	114M 115 mm^2	104 W
IBM Power5 1 processor	Speculative dynamically scheduled; SMT; two CPU cores/chip	8/5/8	6 int. 2 FP	1.9	200M 300 mm^2 (estimated)	80 W (estimated)
Intel Itanium 2	EPIC style; primarily statically scheduled	6/5/11	9 int. 2 FP	1.6	592M 423 mm^2	130 W

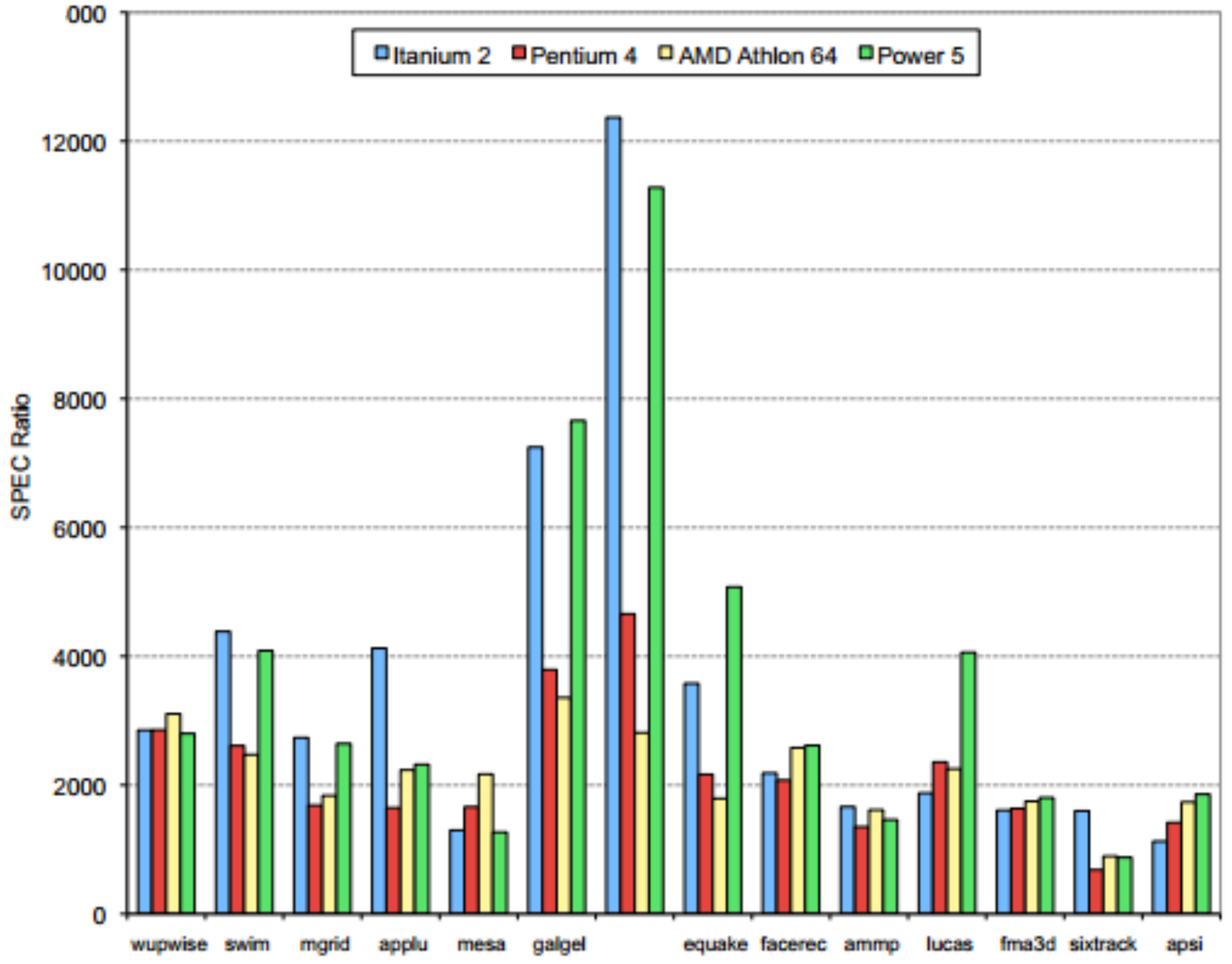
ILP Comparison of multiple-issue processors

SPEC INT rate



ILP Comparison of multiple-issue processors

SPEC FP rate



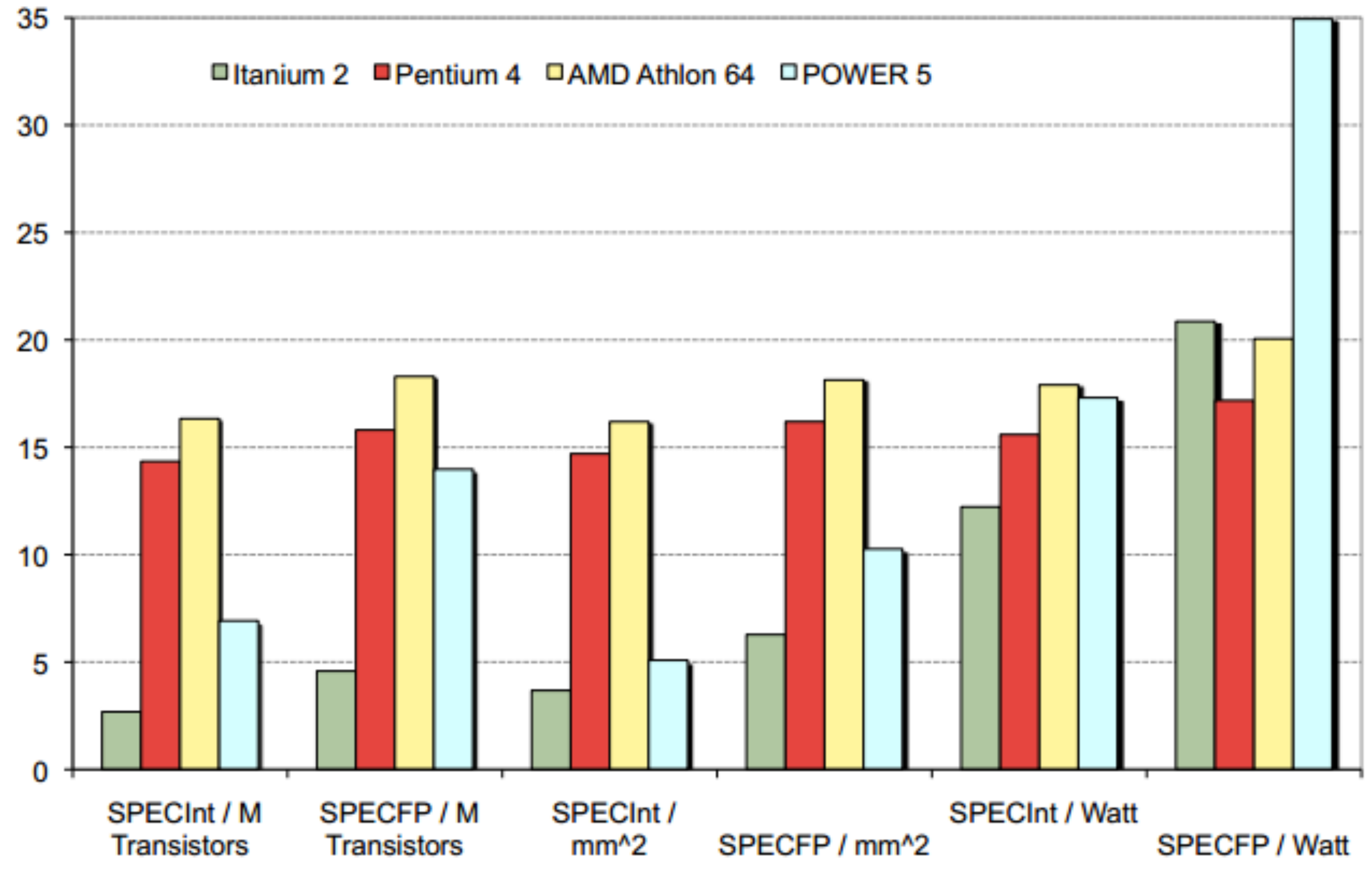
Measuring processor efficiency

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)

ILP Comparison of multiple-issue 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