



西安交通大学
XIAN JIAOTONG UNIVERSITY

Modern Computer Architecture

Lecture6 Multithreading

Hongbin Sun

国家集成电路人才培养基地

Xi'an Jiaotong University

How to Exceed ILP Limits of this study?

- **These are not laws of physics; just practical limits for today, and perhaps overcome via research**
- **Compiler and ISA advances could change results**
- **WAR and WAW hazards through memory: eliminated WAW and WAR hazards through register renaming, but not in memory usage**
 - **Can get conflicts via allocation of stack frames as a called procedure reuses the memory addresses of a previous frame on the stack**

HW v. SW to increase ILP

- **Memory disambiguation: HW best**
- **Speculation:**
 - HW best when dynamic branch prediction better than compile time prediction
 - Exceptions easier for HW
 - HW doesn't need bookkeeping code or compensation code
 - Very complicated to get right
- **Scheduling: SW can look ahead to schedule better**
- **Compiler independence: HW does not require new compiler, recompilation to run well**

Performance beyond single thread ILP

- There can be much higher natural parallelism in some applications (e.g., Database or Scientific codes)
- Explicit **Thread Level Parallelism** or **Data Level Parallelism**
- **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
- **Data Level Parallelism**: Perform identical operations on data, and lots of data

Thread Level Parallelism (TLP)

- **ILP exploits implicit parallel operations within a loop or straight-line code segment**
- **TLP explicitly represented by the use of multiple threads of execution that are inherently parallel**
- **Goal: Use multiple instruction streams to improve**
 1. **Throughput of computers that run many programs**
 2. **Execution time of multi-threaded programs**
- **TLP could be more cost-effective to exploit than ILP**

Another Approach: Multithreaded Execution

- **Multithreading: multiple threads to share the functional units of 1 processor via overlapping**
 - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
 - memory shared through the virtual memory mechanisms, which already support multiple processes
 - HW for fast thread switch; much faster than full process switch \approx 100s to 1000s of clocks
- **When switch?**
 - Alternate instruction per thread (fine grain)
 - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)

Fine-Grained Multithreading

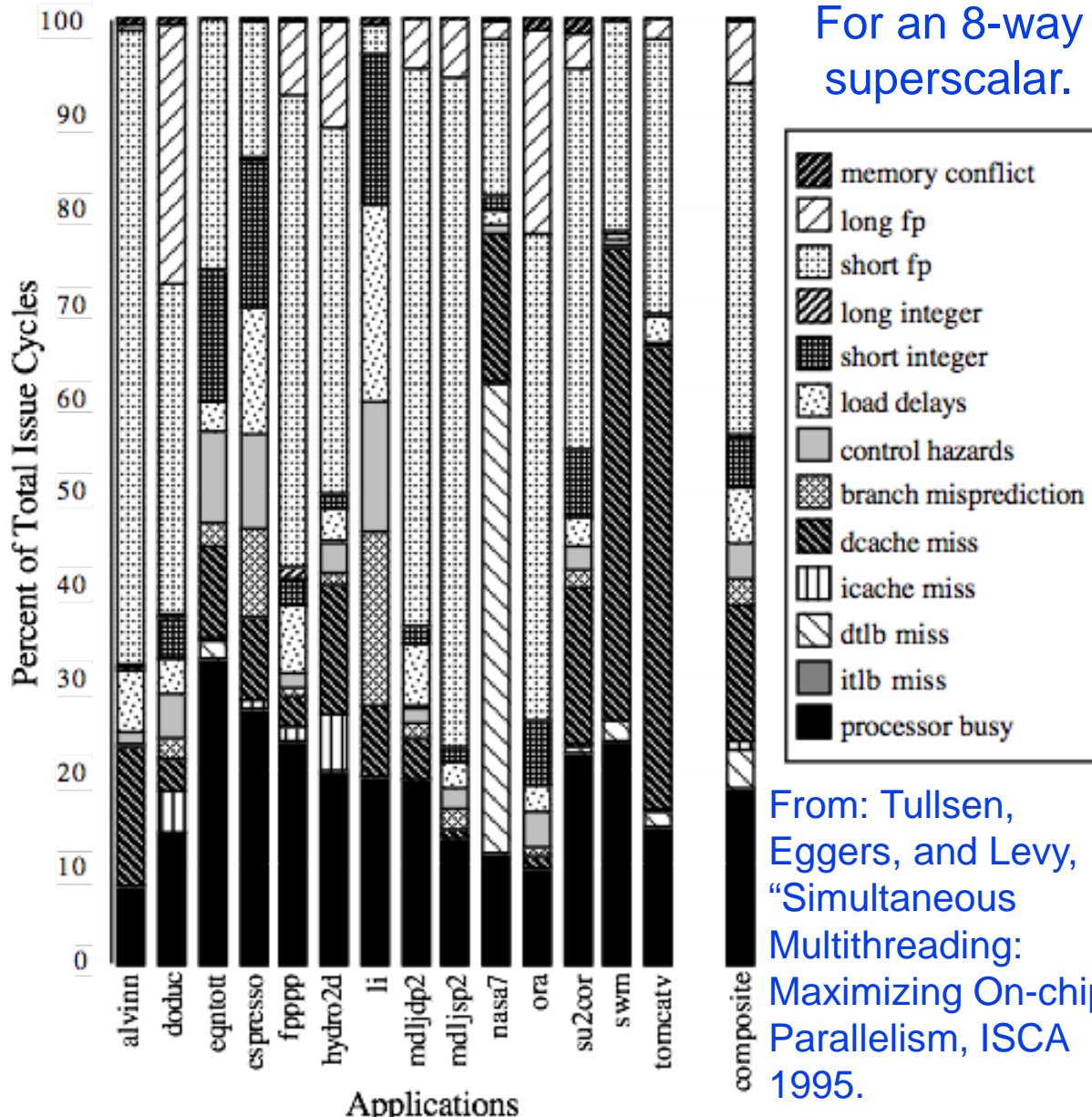
- **Switches between threads on each instruction, causing the execution of multiples threads to be interleaved**
- **Usually done in a round-robin fashion, skipping any stalled threads**
- **CPU must be able to switch threads every clock**
- **Advantage is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls**
- **Disadvantage is it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads**
- **Used on Sun's Niagara (will see later)**

Course-Grained Multithreading

- **Switches threads only on costly stalls, such as L2 cache misses**
- **Advantages**
 - Relieves need to have very fast thread-switching
 - Doesn't slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
- **Disadvantage is hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs**
 - Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen
 - New thread must fill pipeline before instructions can complete
- **Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill \ll stall time**
- **Used in IBM AS/400**

For most apps: most execution units lie idle

For an 8-way
superscalar.



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

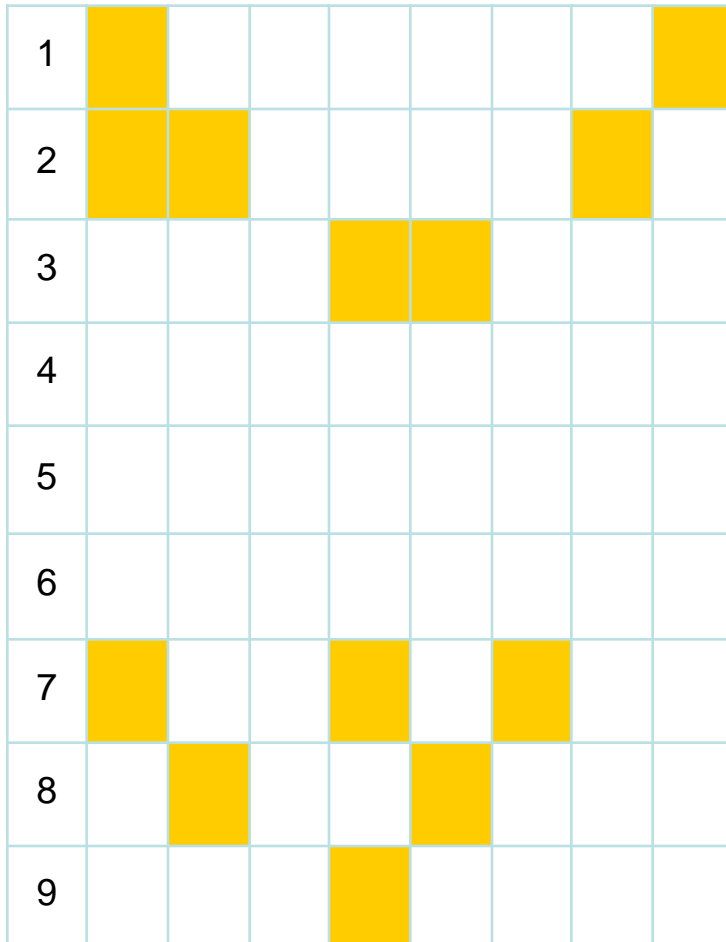
Do both ILP and TLP?

- **TLP and ILP exploit two different kinds of parallel structure in a program**
- **Could a processor oriented at ILP to exploit TLP?**
 - functional units are often idle in data path designed for ILP because of either stalls or dependences in the code
- **Could the TLP be used as a source of independent instructions that might keep the processor busy during stalls?**
- **Could TLP be used to employ the functional units that would otherwise lie idle when insufficient ILP exists?**

Simultaneous Multi-threading

One thread, 8 units

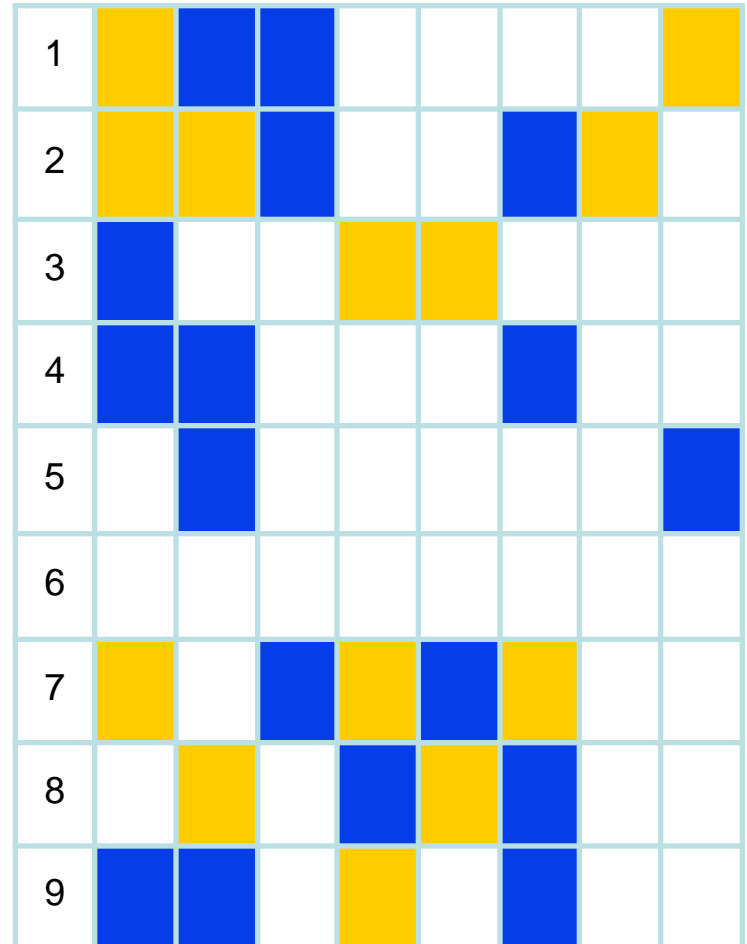
Cycle M M FX FX FP FP BR CC



...

Two threads, 8 units

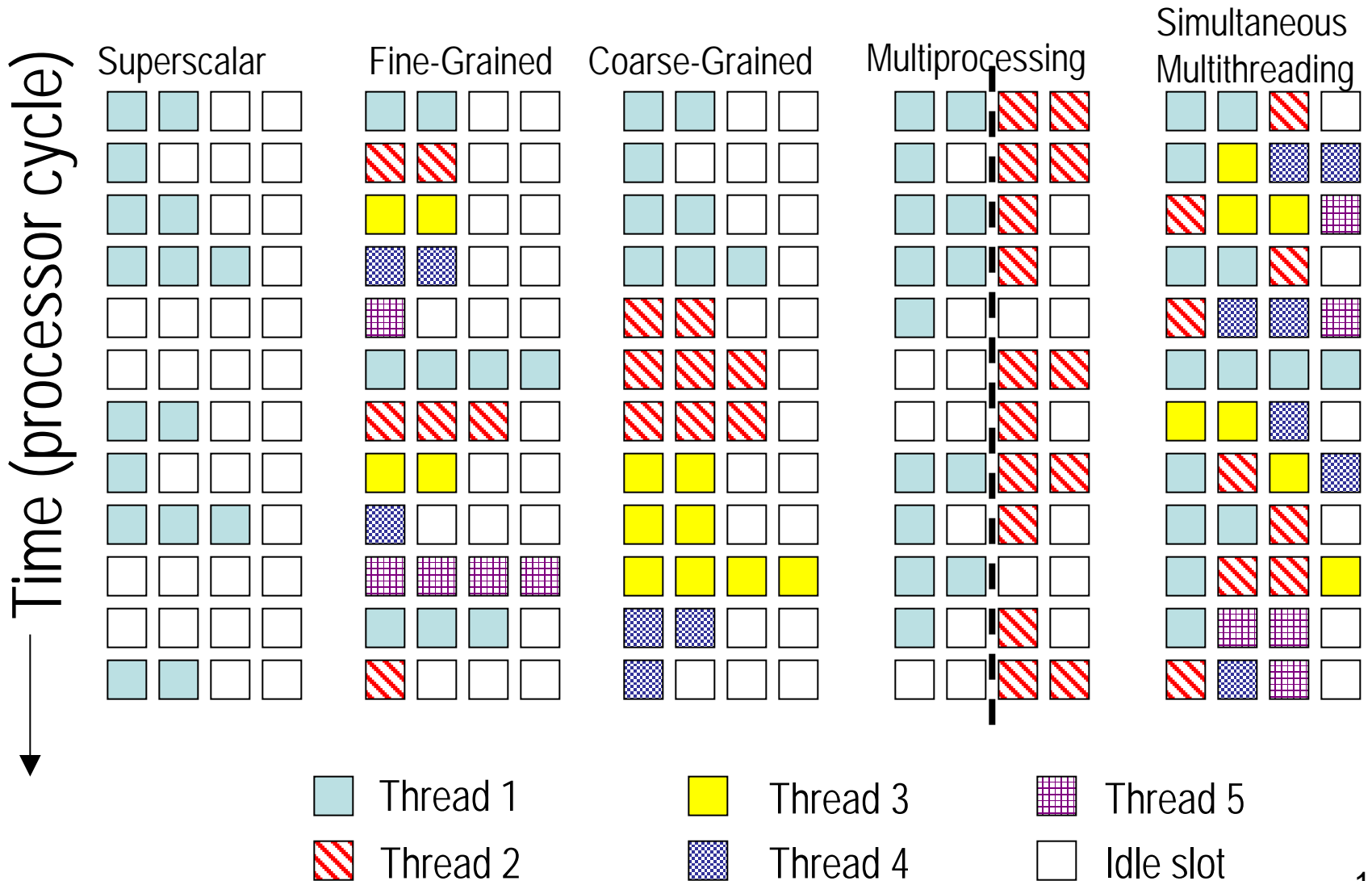
Cycle M M FX FX FP FP BR CC



Simultaneous Multithreading (SMT)

- **Simultaneous multithreading (SMT): insight that dynamically scheduled processor already has many HW mechanisms to support multithreading**
 - Large set of virtual registers that can be used to hold the register sets of independent threads
 - Register renaming provides unique register identifiers, so instructions from multiple threads can be mixed in datapath without confusing sources and destinations across threads
 - Out-of-order completion allows the threads to execute out of order, and get better utilization of the HW
- **Just adding a per thread renaming table and keeping separate PCs**
 - Independent commitment can be supported by logically keeping a separate reorder buffer for each thread

Multithreaded Categories

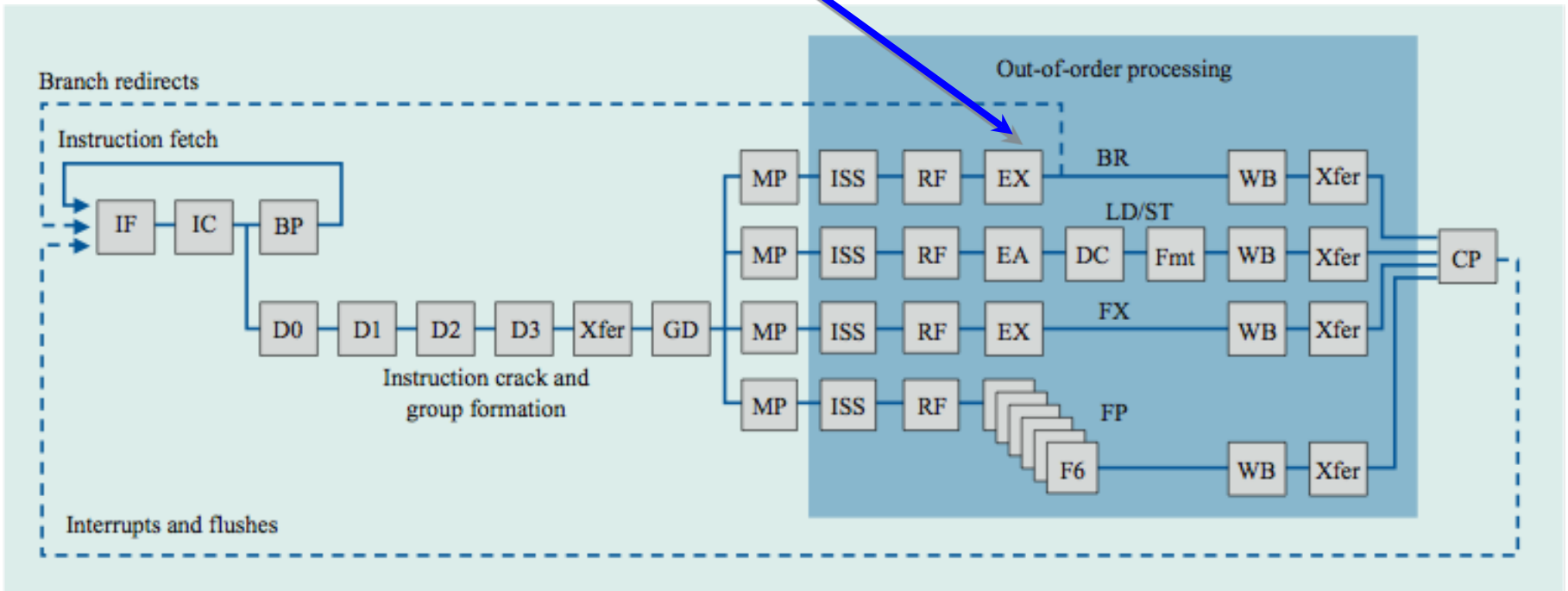
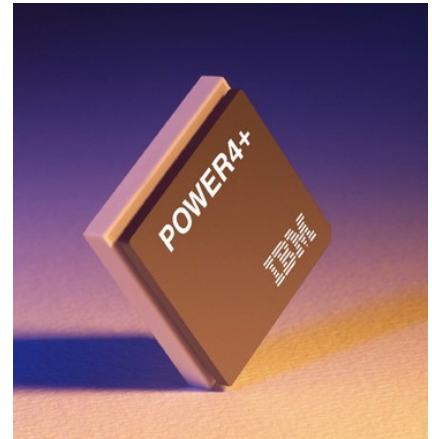


Design Challenges in SMT

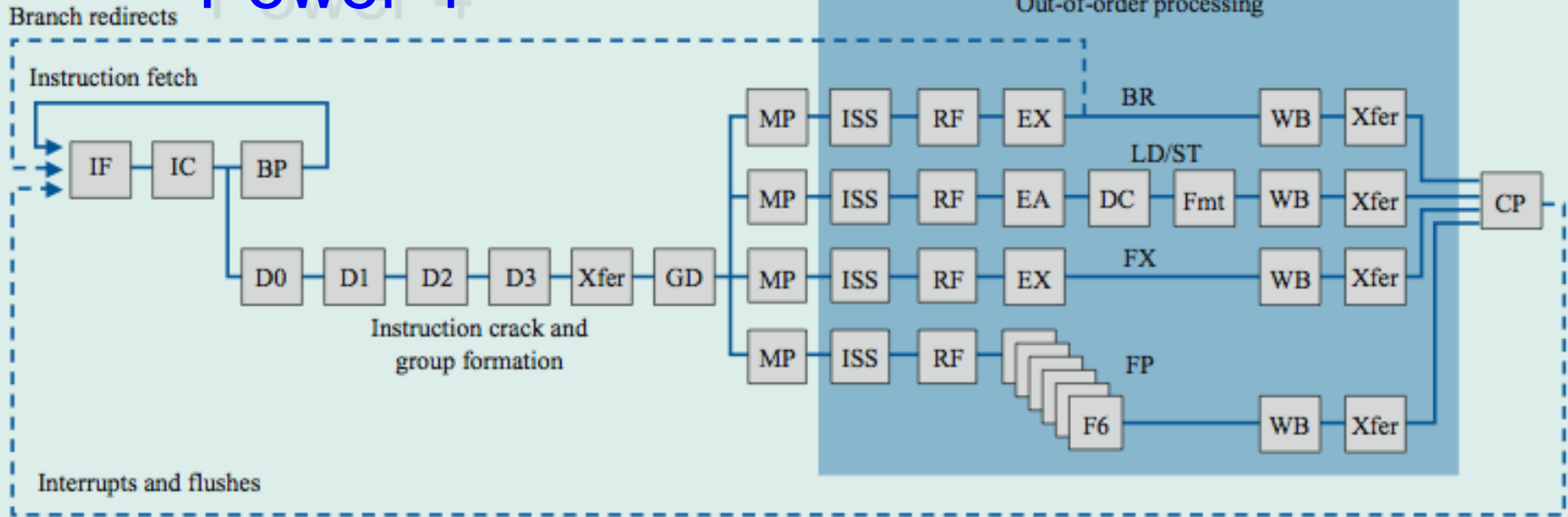
- **Since SMT makes sense only with fine-grained implementation, impact of fine-grained scheduling on single thread performance?**
 - A preferred thread approach sacrifices neither throughput nor single-thread performance?
 - Unfortunately, with a preferred thread, the processor is likely to sacrifice some throughput, when preferred thread stalls
- **Larger register file needed to hold multiple contexts**
- **Clock cycle time, especially in:**
 - Instruction issue - more candidate instructions need to be considered
 - Instruction completion - choosing which instructions to commit may be challenging
- **Ensuring that cache and TLB conflicts generated by SMT do not degrade performance**

Power 4

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

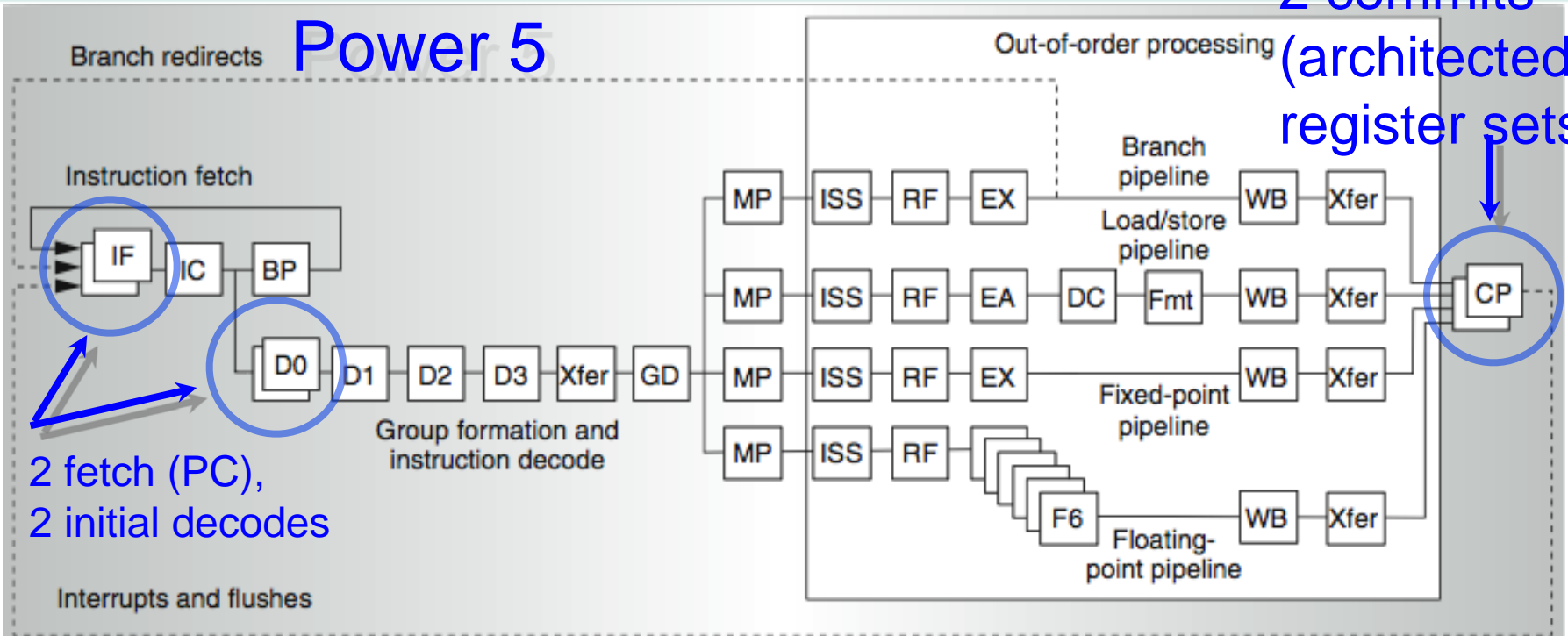


Power 4



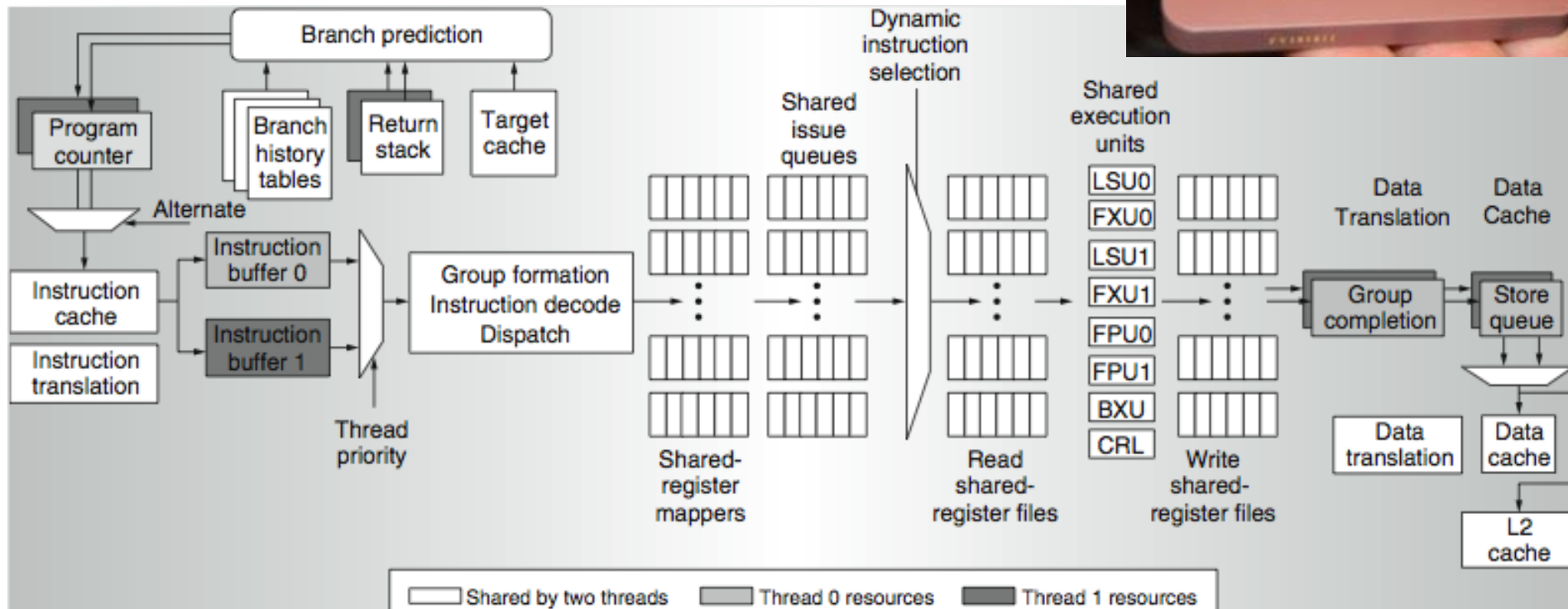
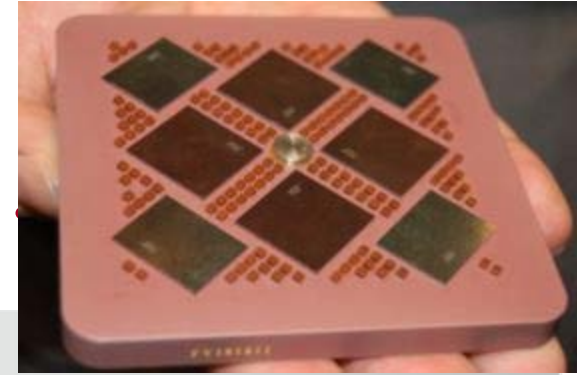
2 commits
(architected
register sets)

Power 5



2 fetch (PC),
2 initial decodes

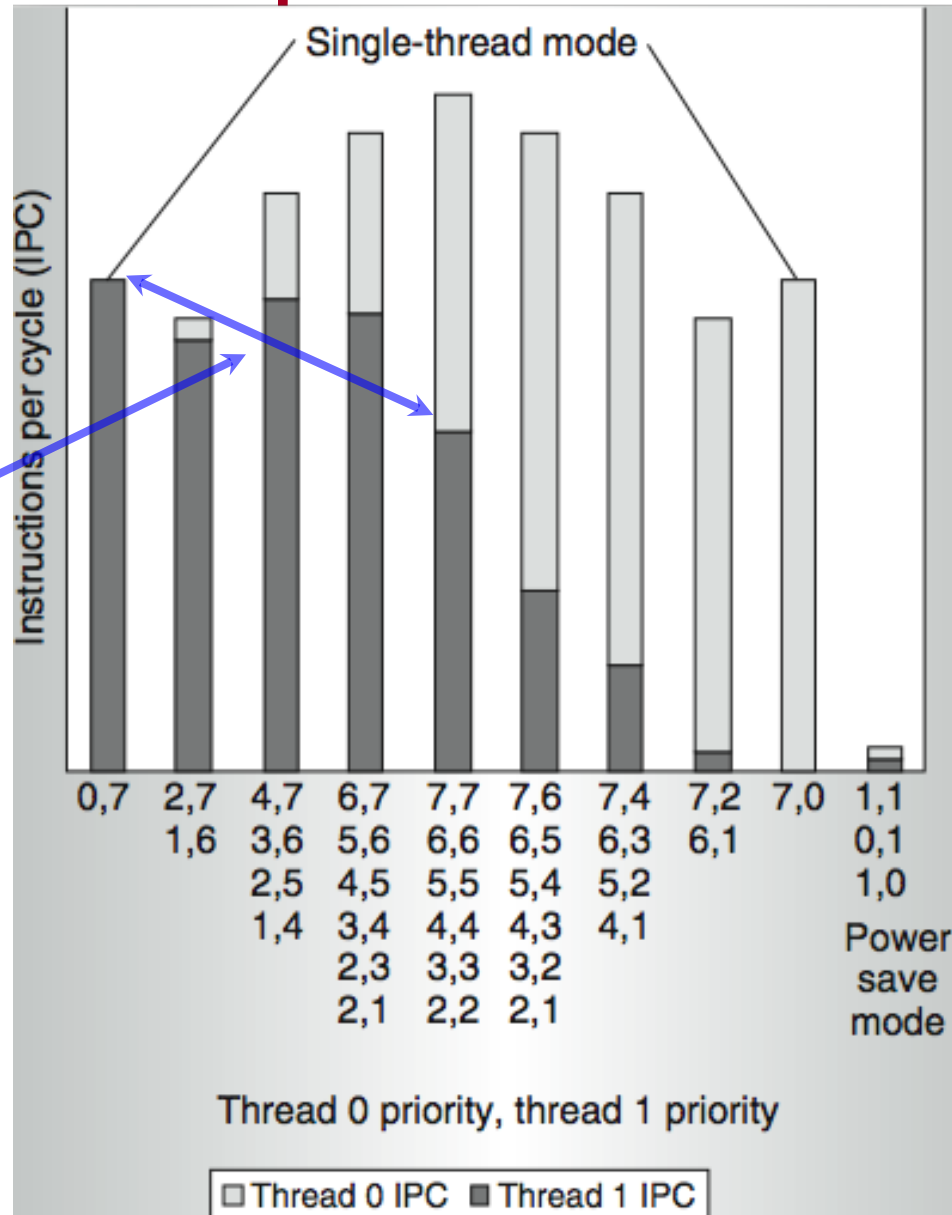
Power 5 data flow



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

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.

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**

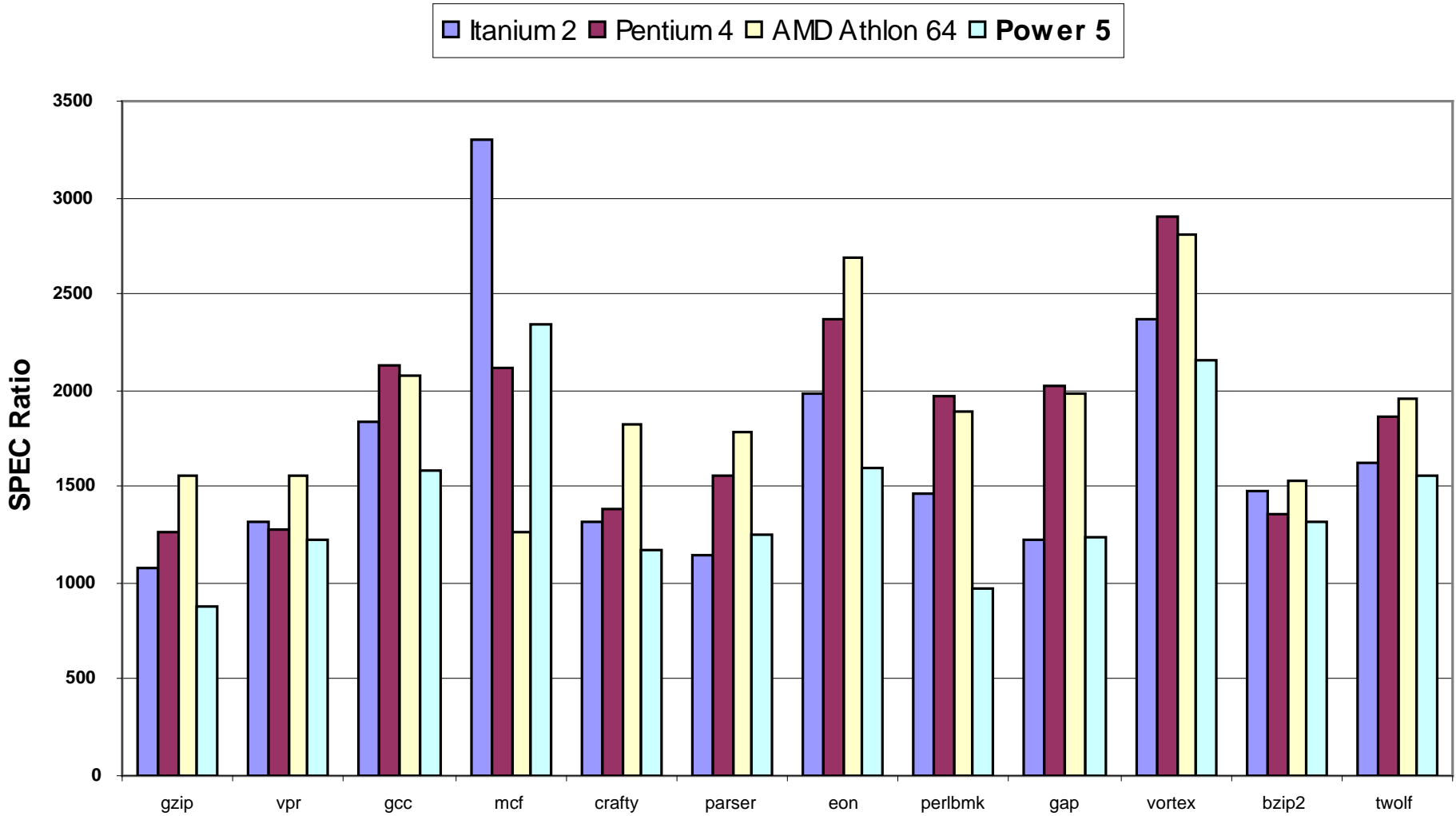
Initial Performance of SMT

- **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

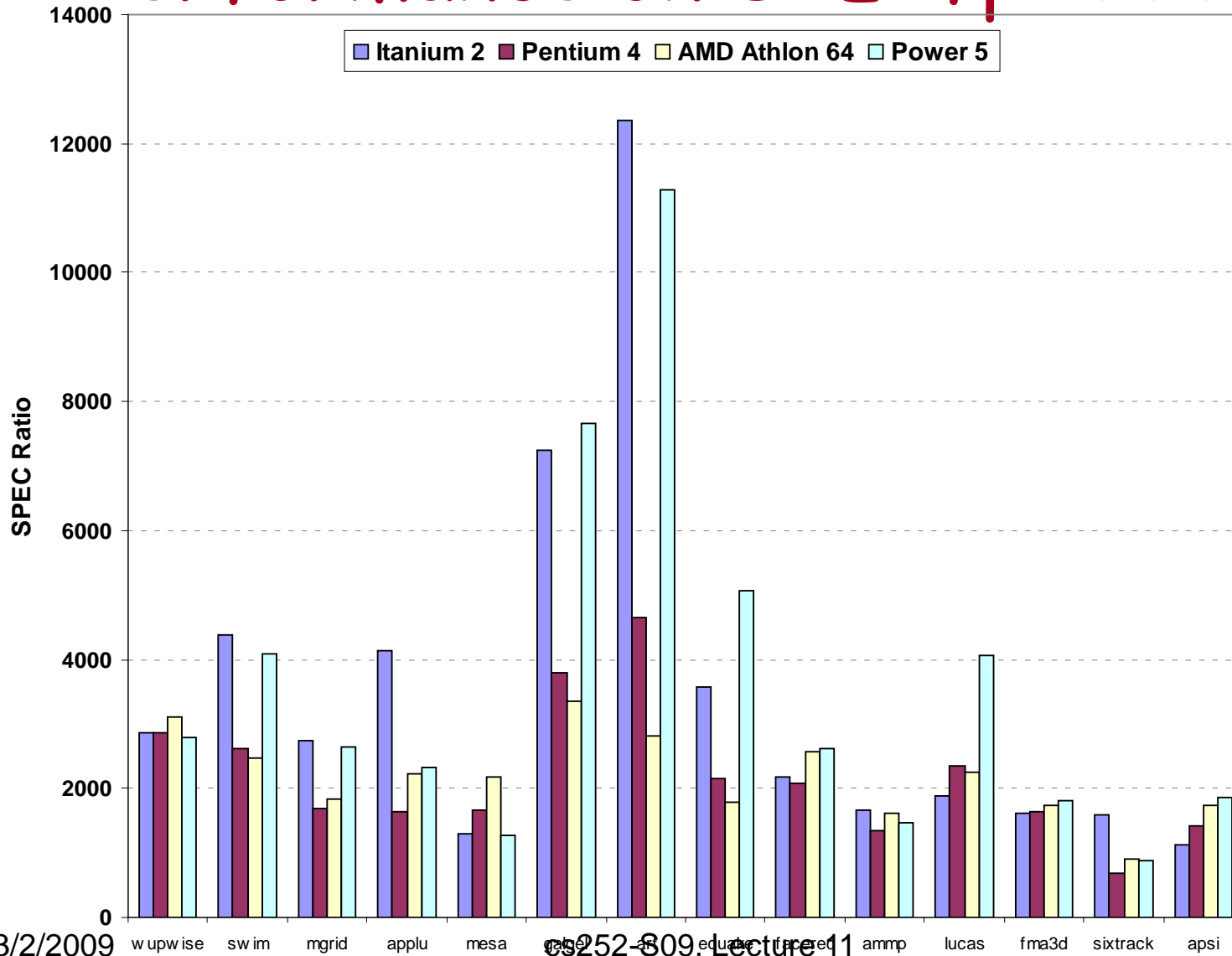
Head to Head ILP competition

Processor	Micro architecture	Fetch / Issue / Execute	FU	Clock Rate (GHz)	Transistors 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

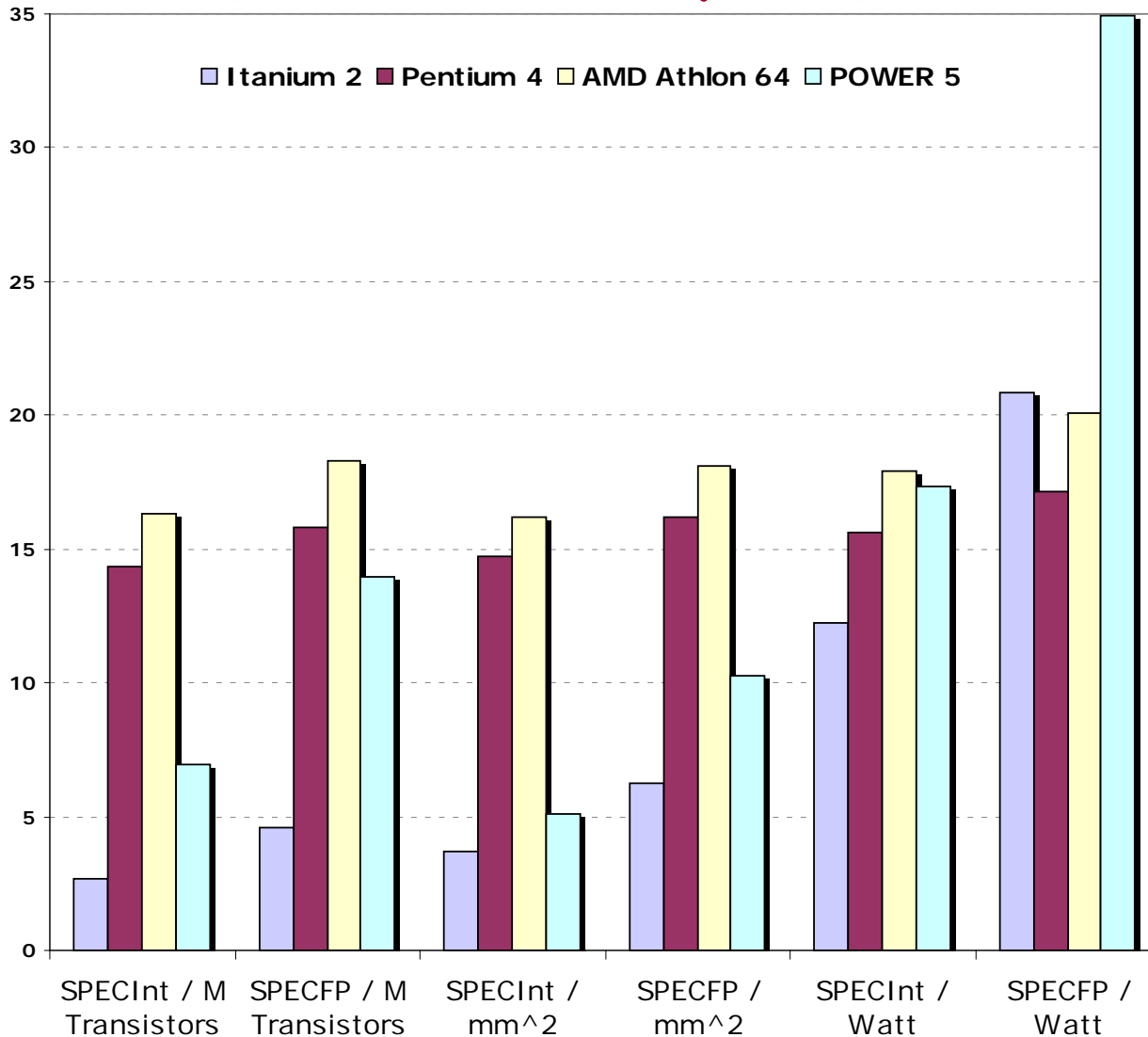
Performance on SPECint2000



Performance on SPECfp2000



Normalized Performance: Efficiency



Rank	Itanium 2	Pentium 4	Athlon	Power 5
Int/Trans	4	2	1	3
FP/Trans	4	2	1	3
Int/area	4	2	1	3
FP/area	4	2	1	3
Int/Watt	4	3	1	2
FP/Watt	2	4	3	1

No Silver Bullet for ILP

- No obvious over all leader in performance
- The AMD Athlon leads on SPECInt performance followed by the Pentium 4, Itanium 2, and Power5
- Itanium 2 and Power5, which perform similarly on SPECFP, clearly dominate the Athlon and Pentium 4 on SPECFP
- Itanium 2 is the most **inefficient** processor both for Fl. Pt. and integer code for all but one efficiency measure (SPECFP/Watt)
- Athlon and Pentium 4 both make good use of transistors and area in terms of efficiency,
- IBM Power5 is the most effective user of energy on SPECFP and essentially tied on SPECINT

Limits to ILP

- **Doubling issue rates above today's 3-6 instructions per clock, say to 6 to 12 instructions, probably requires a processor to**
 - issue 3 or 4 data memory accesses per cycle,
 - resolve 2 or 3 branches per cycle,
 - rename and access more than 20 registers per cycle, and
 - fetch 12 to 24 instructions per cycle.
- **The complexities of implementing these capabilities is likely to mean sacrifices in the maximum clock rate**
 - E.g, widest issue processor is the Itanium 2, but it also has the slowest clock rate, despite the fact that it consumes the most power!

Limits to ILP

- Most techniques for increasing performance increase power consumption
- The key question is whether a technique is *energy efficient*: does it increase power consumption faster than it increases performance?
- Multiple issue processors techniques all are energy inefficient:
 1. Issuing multiple instructions incurs some overhead in logic that grows faster than the issue rate grows
 2. Growing gap between peak issue rates and sustained performance
- Number of transistors switching = $f(\text{peak issue rate})$, and performance = $f(\text{sustained rate})$, growing gap between peak and sustained performance
⇒ increasing energy per unit of performance

Discussion of papers:

Complexity-effective superscalar processors

- “Complexity-effective superscalar processors”, Subbarao Palacharla, Norman P. Jouppi and J. E. Smith.
 - Several data structures analyzed for complexity WRT issue width
 - Rename: Roughly Linear in IW, steeper slope for smaller feature size
 - Wakeup: Roughly Linear in IW, but quadratic in window size
 - Bypass: Strongly quadratic in IW

Issue width	Window size	Rename delay (ps)	Wakeup+Select delay (ps)	Bypass delay (ps)
0.8 μ m technology				
4	32	1577.9	2903.7	184.9
8	64	1710.5	3369.4	1056.4
0.35 μ m technology				
4	32	627.2	1248.4	184.9
8	64	726.6	1484.8	1056.4
0.18 μ m technology				
4	32	351.0	578.0	184.9
8	64	427.9	724.0	1056.4

- Overall results: Table 2: Overall delay results.
 - Bypass significant at high window size/issue width
 - Wakeup+Select delay dominates otherwise
- Proposed Complexity-effective design:
 - Replace issue window with FIFOs/steer dependent Insts to same FIFO

Commentary

- Itanium architecture does **not** represent a significant breakthrough in scaling ILP or in avoiding the problems of complexity and power consumption
- Instead of pursuing more ILP, architects are increasingly focusing on TLP implemented with single-chip multiprocessors
- In 2000, IBM announced the 1st commercial single-chip, general-purpose multiprocessor, the Power4, which contains 2 Power3 processors and an integrated L2 cache
 - Since then, Sun Microsystems, AMD, and Intel have switch to a focus on single-chip multiprocessors rather than more aggressive uniprocessors.
- Right balance of ILP and TLP is unclear today
 - Perhaps right choice for server market, which can exploit more TLP, may differ from desktop, where single-thread performance may continue to be a primary requirement

And in conclusion ...

- **Limits to ILP (power efficiency, compilers, dependencies ...) seem to limit to 3 to 6 issue for practical options**
- **Explicitly parallel (Data level parallelism or Thread level parallelism) is next step to performance**
- **Coarse grain vs. Fine grained multithreading**
 - Only on big stall vs. every clock cycle
- **Simultaneous Multithreading if fine grained multithreading based on OOO superscalar microarchitecture**
 - Instead of replicating registers, reuse rename registers