Parallel Computer Architecture
Fall 2018
Thread Level Parallelism
Simultaneous Multithreading (SMT)

Nikos Bellas

Computer and Communications Engineering Department
University of Thessaly

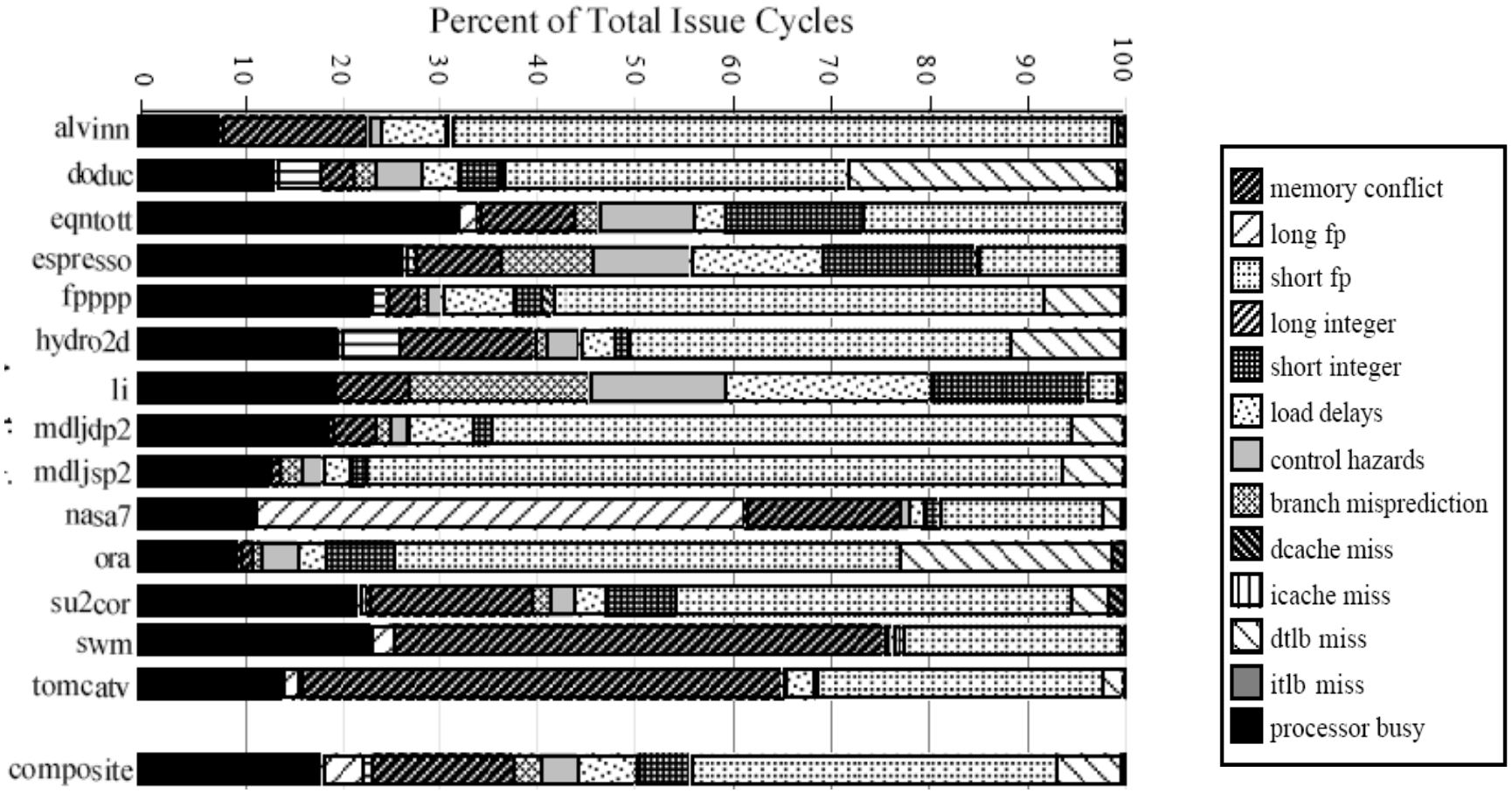
Readings for this lecture

- **Multithreading**
  - H&P, v3.0, section 6.9
  - H&P, v4.0, section 3.5
- **SMT**
  - D. Tullsen et al, " Simultaneous Multithreading: Maximizing On-Chip Parallelism," ISCA 22, June 1995
  - B Sinharoy et al, "Power5 system microarchitecture," IBM Journal of Research & Development, July/September 2005

ILP is not enough

- ILP and DLP our main focus in class
  - Dynamic and static techniques to extract parallelism from a single stream of instructions
- There are a lot of reasons why this is not enough for a wide-issue processor
  - ILP is local. Due to the limited instruction window size, only instructions from a small region are examined for parallelism
  - Function calls, imperfect branch predictions, cache misses, memory alias analysis and large functional latencies have significant negative impact
  - In a lot of apps, ILP is inherently small (e.g. SPECint benchmarks)
- Briefly discussed data level parallelism (DLP)
  - Loop unrolling
  - Same instructions, multiple data streams
  - GPUs, Vector processors, SIMD processors

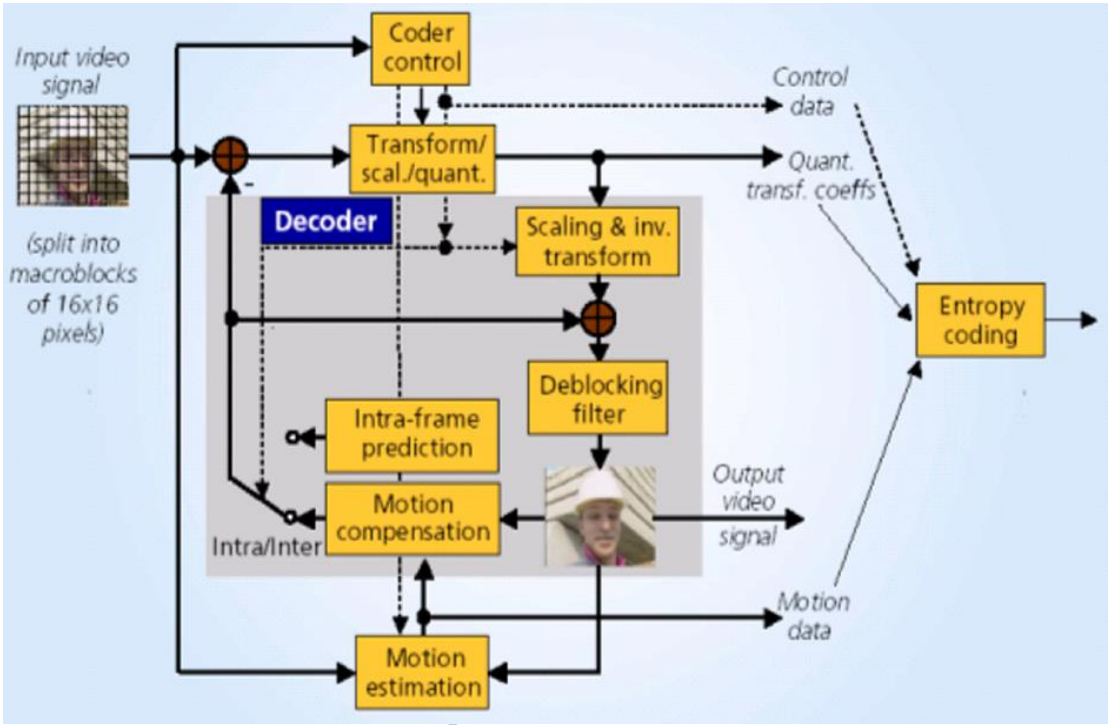
Where do cycles go?



Based on Alpha 21164 (8 issue peak rate)
SPEC92 benchmarks
Only 19% useful cycles

Coarse-grain level parallelism

- **Frequently, abundant parallelism at higher level of the application**
- **Mainly due to the nature of the algorithm**
  - Transaction processing systems (web-servers, DB servers, storage servers)
  - Video compression



Coarse-grain level parallelism

- Different levels of coarse-grain parallelism
- Processes are the “heaviest” unit of granularity
  - Processes own resources allocated by the O/S
  - Resources include memory, file handles, sockets, device handles, and windows
  - Normally, no sharing of resources among processes
  - Processes are typically pre-emptively multitasked by the O/S
  - E.g. Writing a word document, listening to an MP3, web browsing at the same time
  - Only one process runs at any time in a single-threaded core

Coarse-grain level parallelism

- Threads are in the same process and share the address space
- ILP is implicit, the programmer knows nothing about it
- Thread Level Parallelism (TLP) is explicitly represented by the programmer
  - POSIX thread programming :

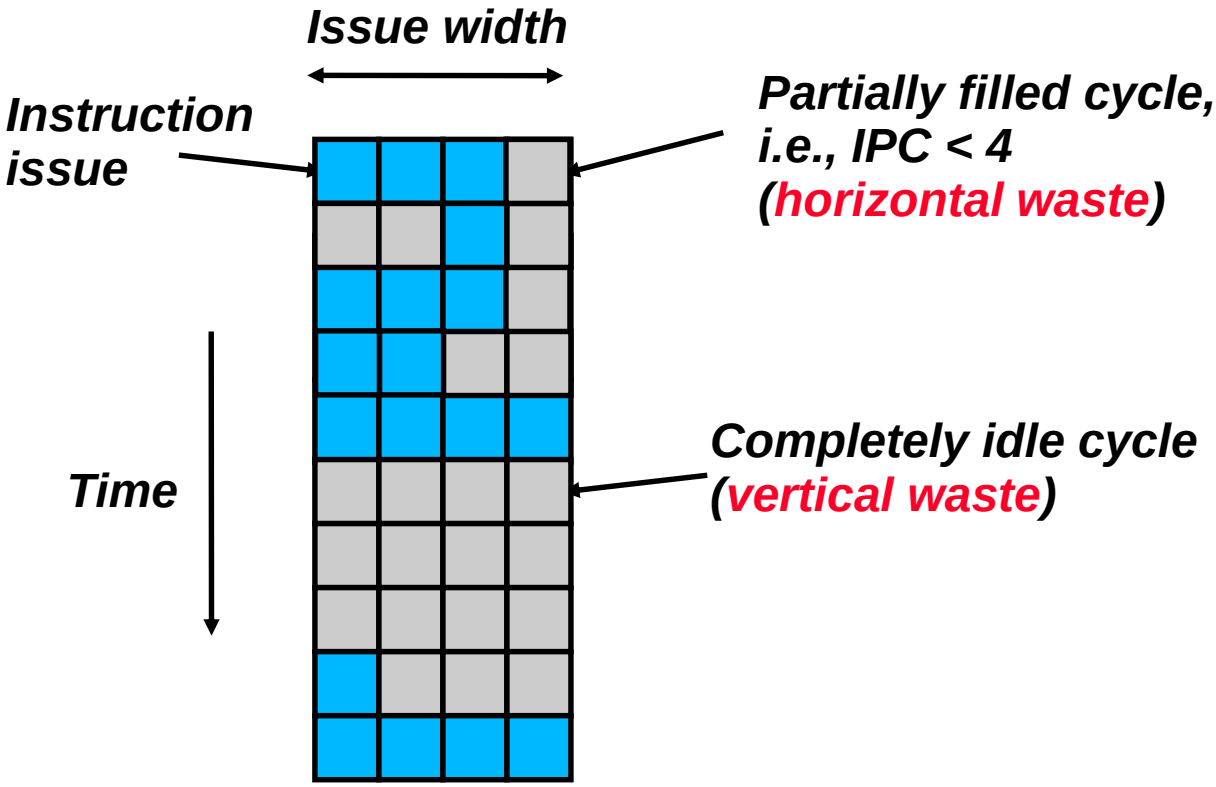
```
int iret1 = pthread_create( &thread1, NULL, &function_name, (void*) message1);
```

  - Previous example with video compression
  - Automatic TLP extraction by the compiler, is a hot research topic.
  - With very limited results….
- ILP and TLP are complementary
- Exploit different sources of parallelism in the program

Coarse-grain level parallelism

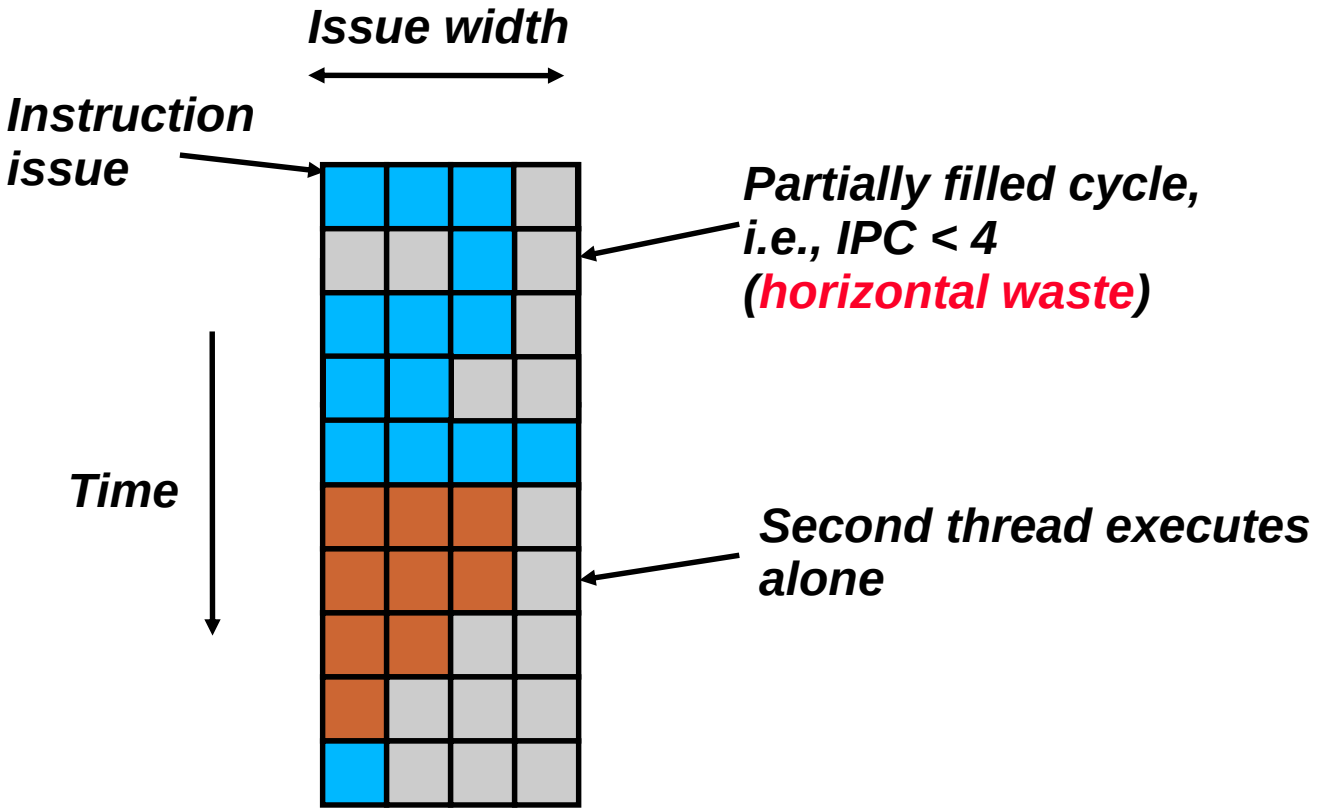
- **Questions**
  - Can a superscalar single-core processor exploit both ILP and TLP?
  - Can the TLP be used to employ idle resources when insufficient ILP exists?
  - If yes, what changes are needed in the architecture?
  - What are the performance gains versus the extra complexity and power dissipation?

Superscalar – single thread



Constrained by lack of ILP

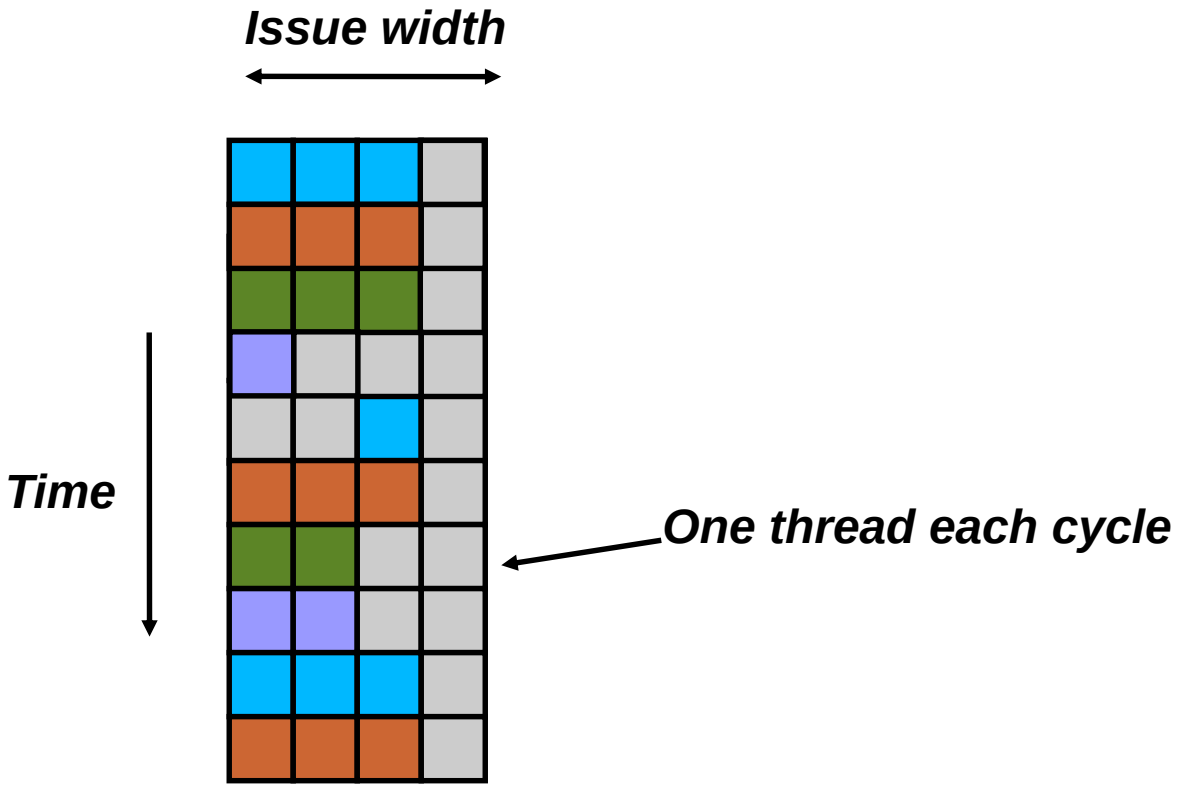
Coarse-grain multithreading



Threads run for more than a cycle. Only one thread every cycle.

Removes vertical waste, but leaves some horizontal waste

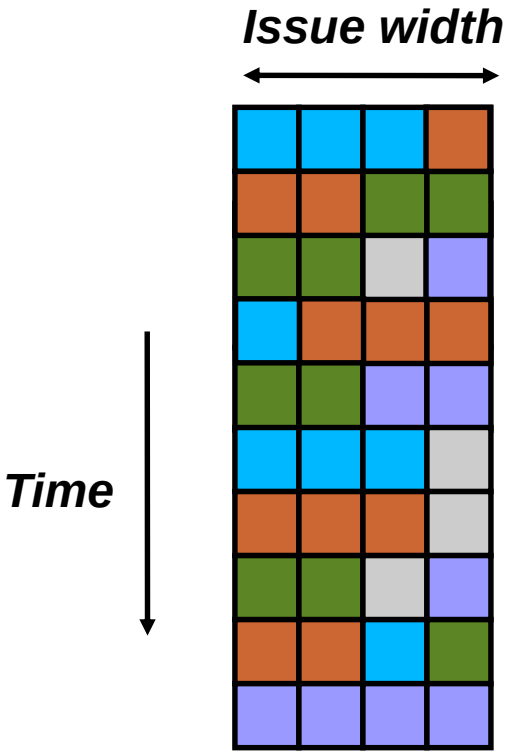
Fine-grain multithreading



Threads run for a single cycle. Only one thread every cycle.

Removes vertical waste, but leaves some horizontal waste

Simultaneous multithreading (SMT)



- Interleave multiple threads to multiple issue slots with no restrictions

SMT issues

- Higher throughput than single thread superscalar
- BUT, higher latency per thread
- Tradeoffs between throughput and latency are possible
- “Preferred” threads are given higher priority
  - Instruction prefetch, resources are given mainly to the preferred thread
  - If it stalls, other threads are considered

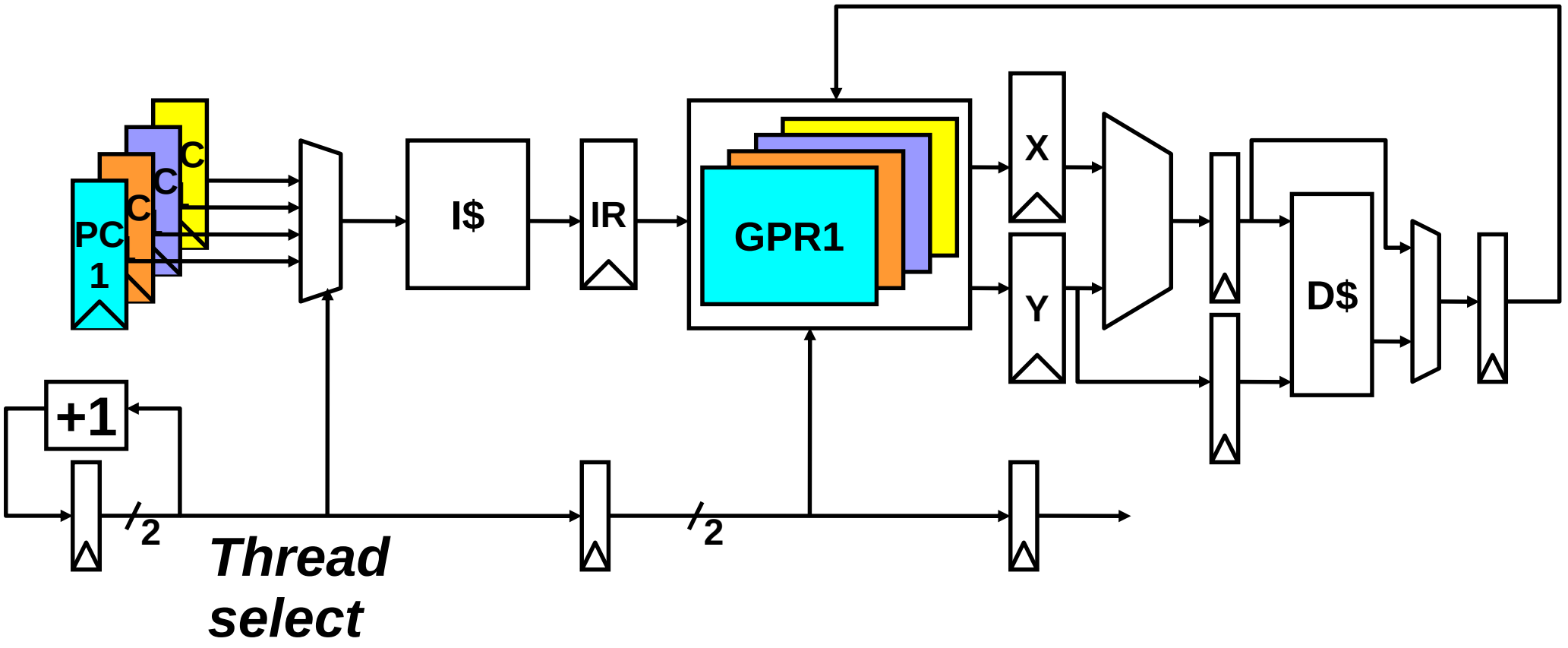
Out of Order SMT

- 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

Multithreading Costs

- Each thread requires its own user state
  - PC
  - Fetch Units
  - Control Registers
- Also, needs its own system state
  - virtual memory page table base register
  - exception handling registers
- Design challenges
  - Larger register file
  - Larger fetch units
  - Clock cycle may be affected

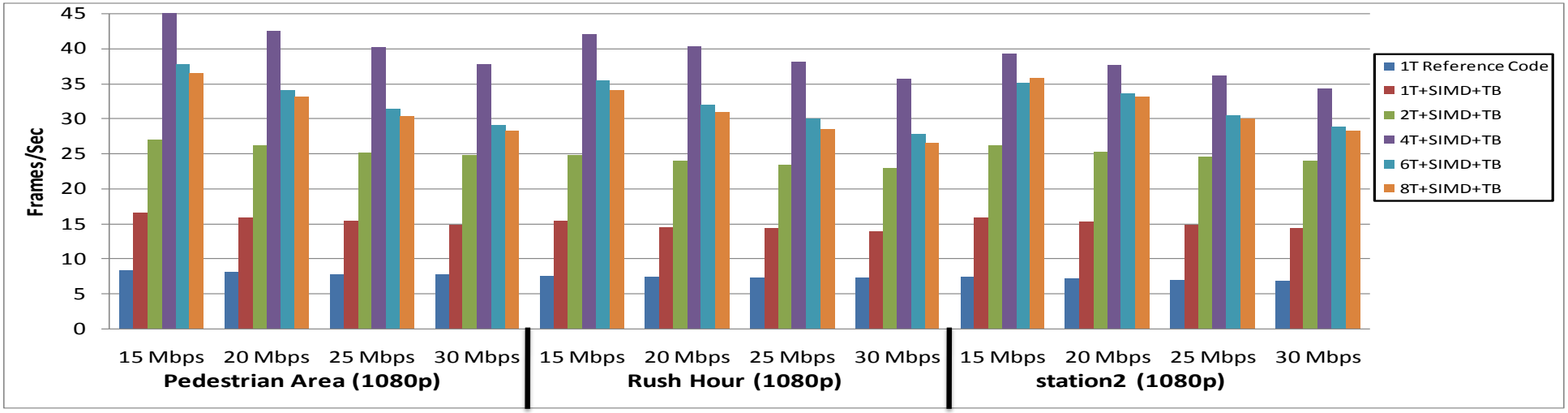
Simple SMT pipeline



- 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

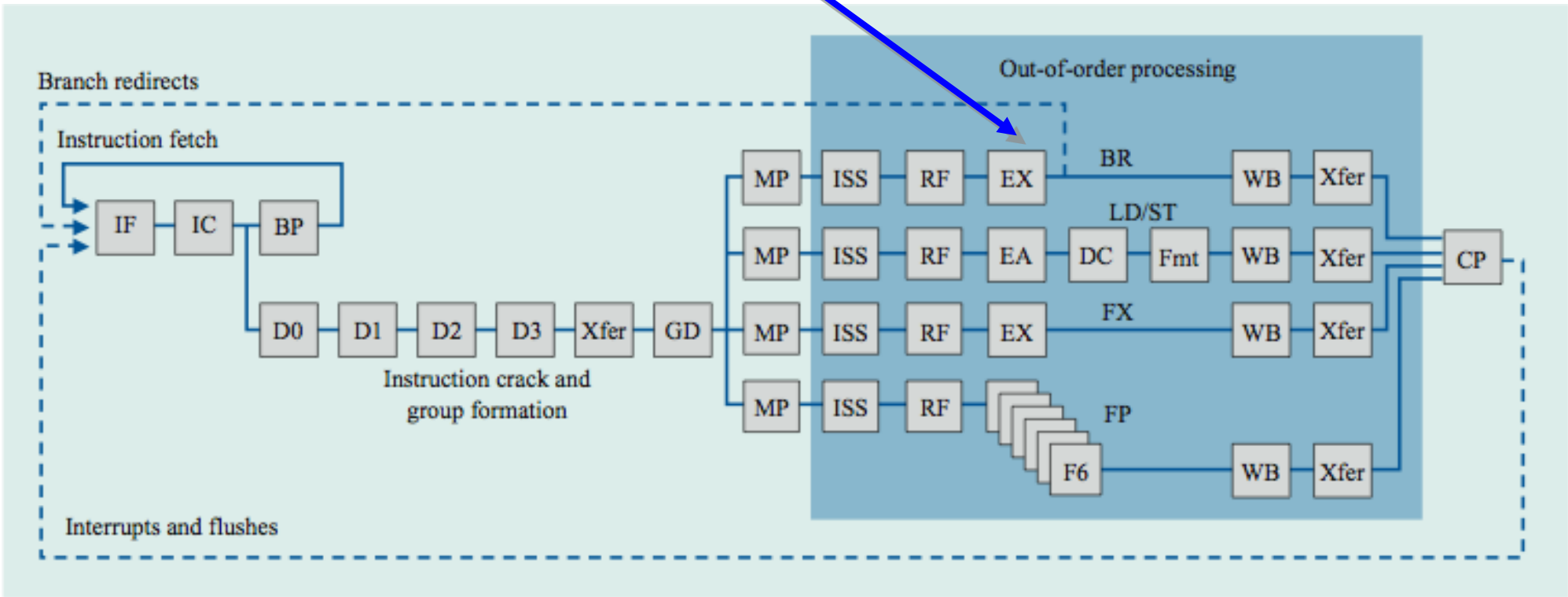
SMT performance

- SMT does not always offer performance improvements
  - AVS Video decoding example : performance (in fps) drops for #of threads > #cores
  - Core i7 with 4 cores and Hyperthreading enabled



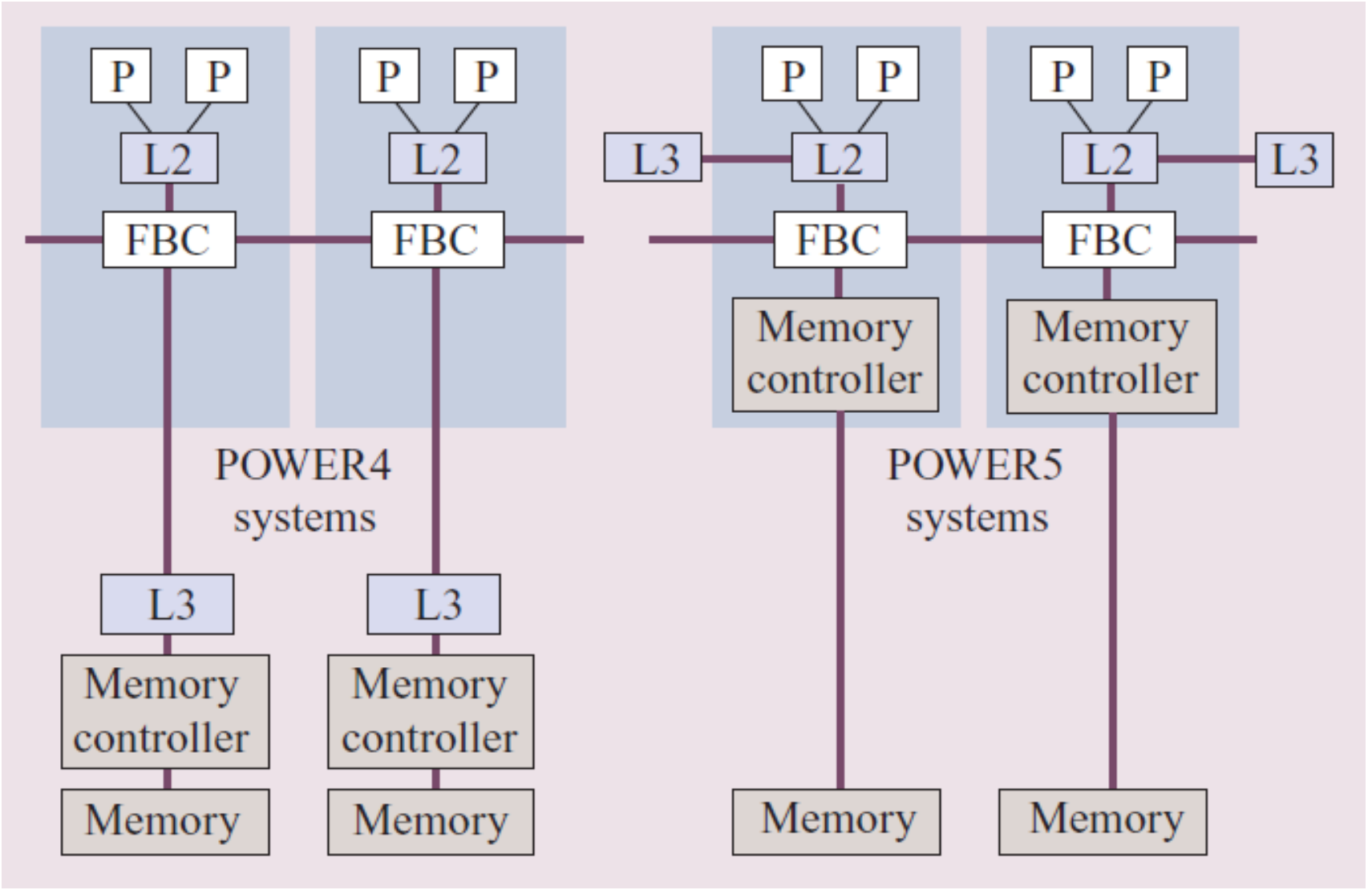
Power 4

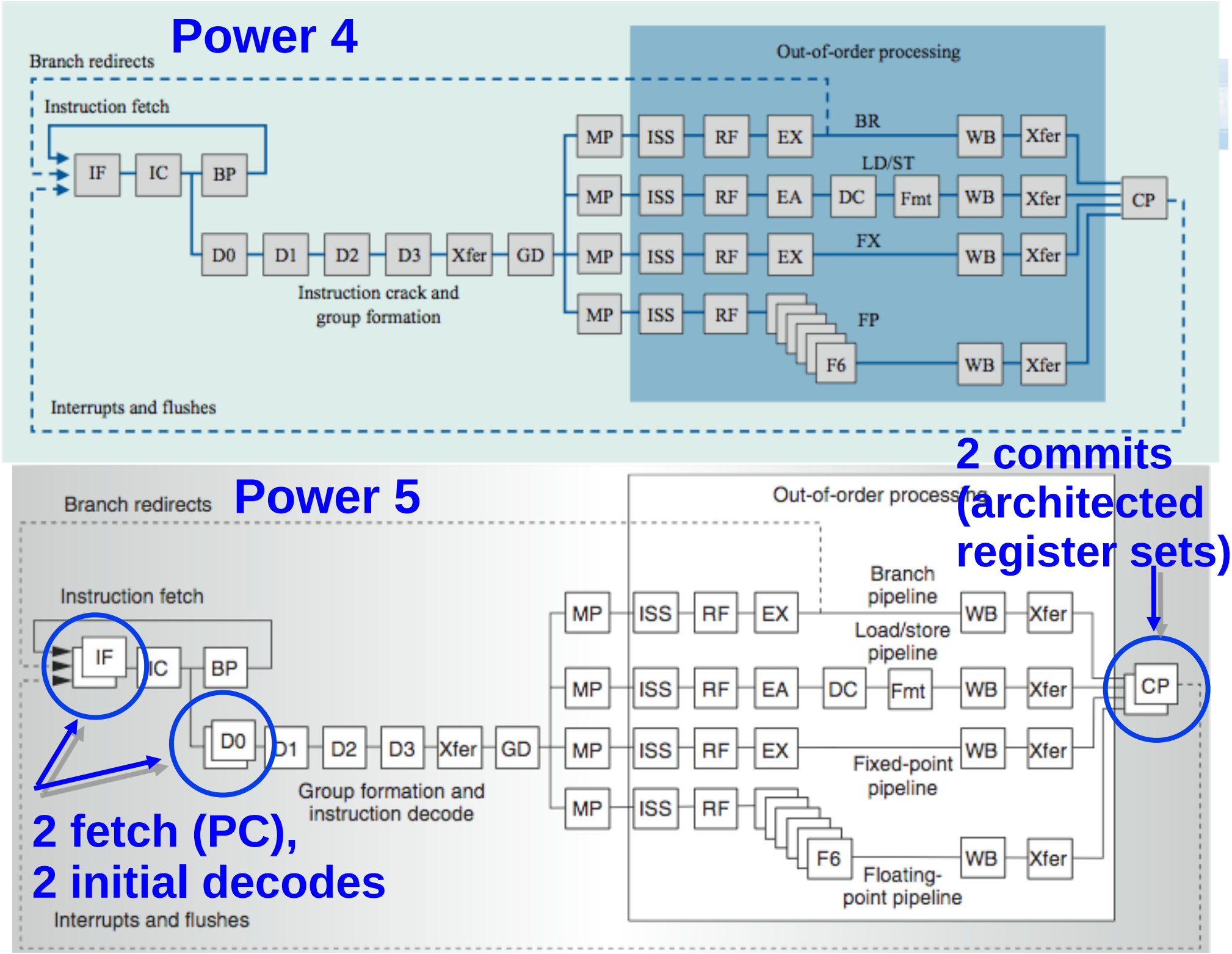
Single-threaded predecessor to Power 5.
8 execution units in
out-of-order engine, each may
issue an instruction each cycle.
Released in 2001



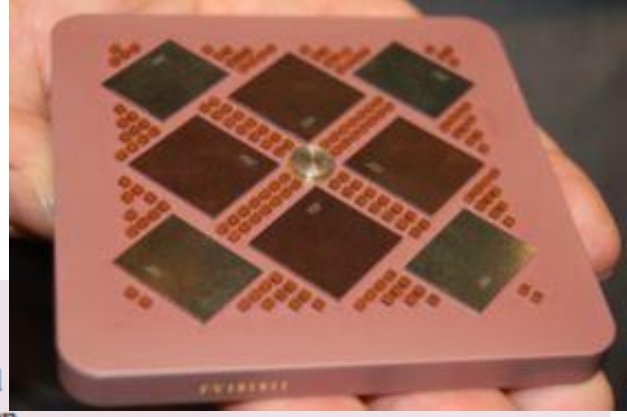
Power4 and Power5 system architecture

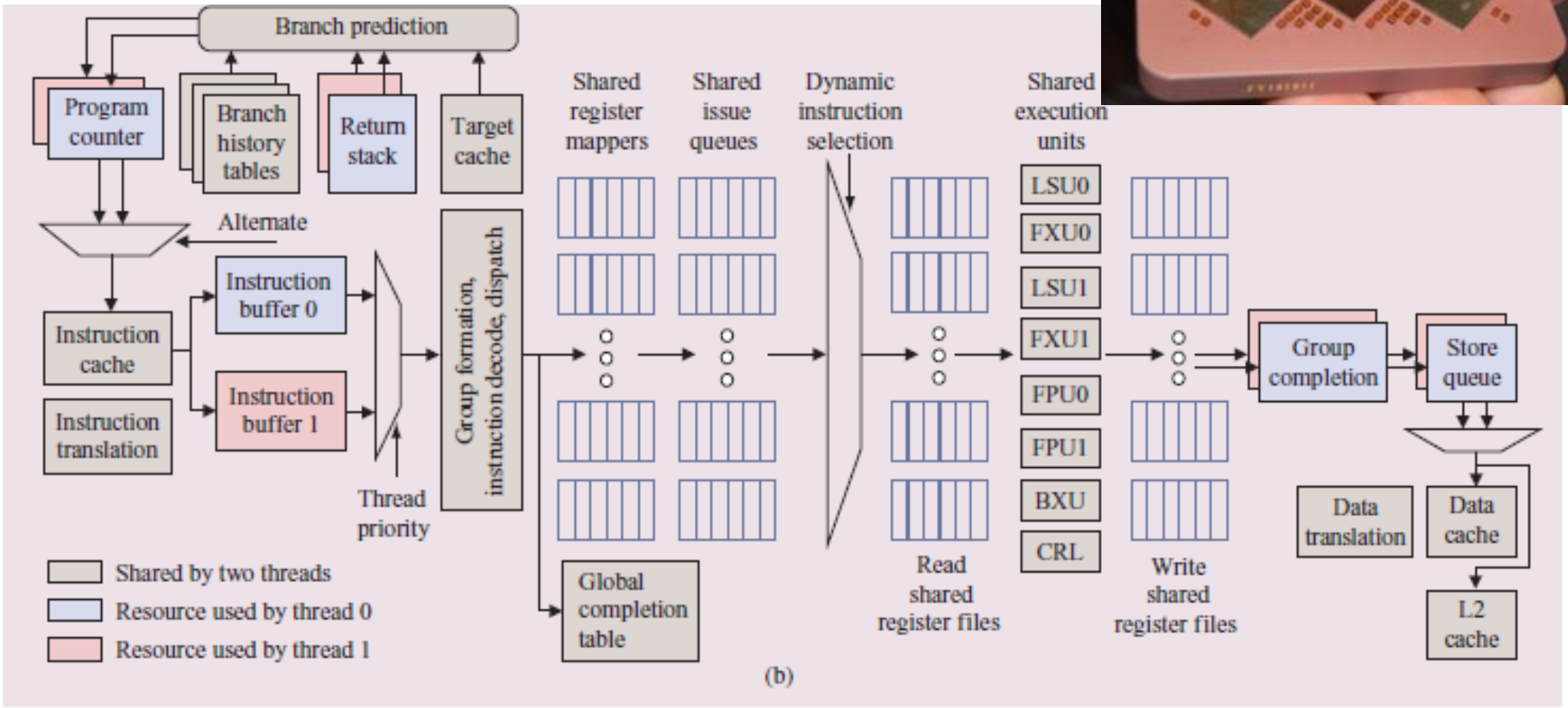
Processors optimized for the server market




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
Out-of-order processing
Instruction fetch
Branch pipeline
Load/store pipeline
Fixed-point pipeline
Floating-point pipeline
Group formation and instruction decode
2 commits (architected register sets)
2 fetch (PC), 2 initial decodes
Interrupts and flushes

Power5 data flow …





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

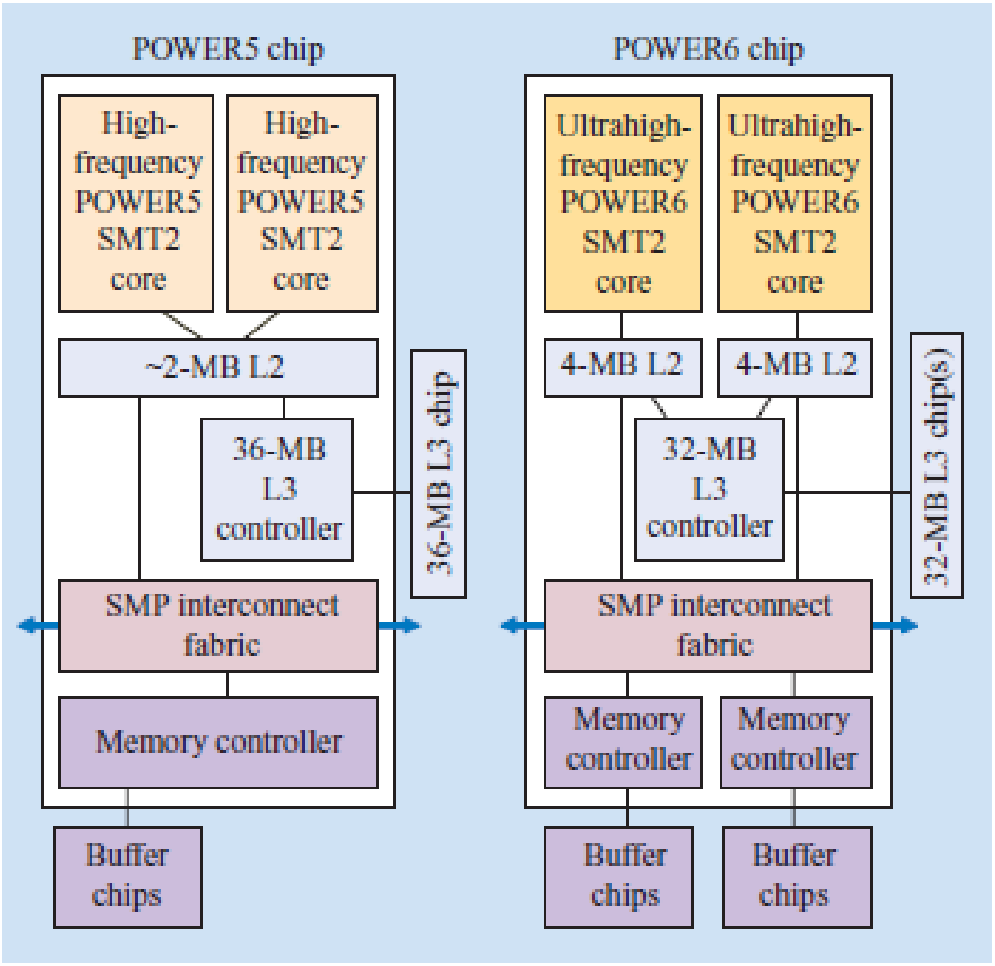
Changes in Power5 to support SMT

- 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

Power6

- Power6 is the follow-up microprocessor to Power5
- 65nm technology – 790 million transistors
- Dual core, 2 threads per core, SMT design
- Improved FP operation, and novel decimal FP unit
- Extensions for vector processing
  - Including an Altivec unit

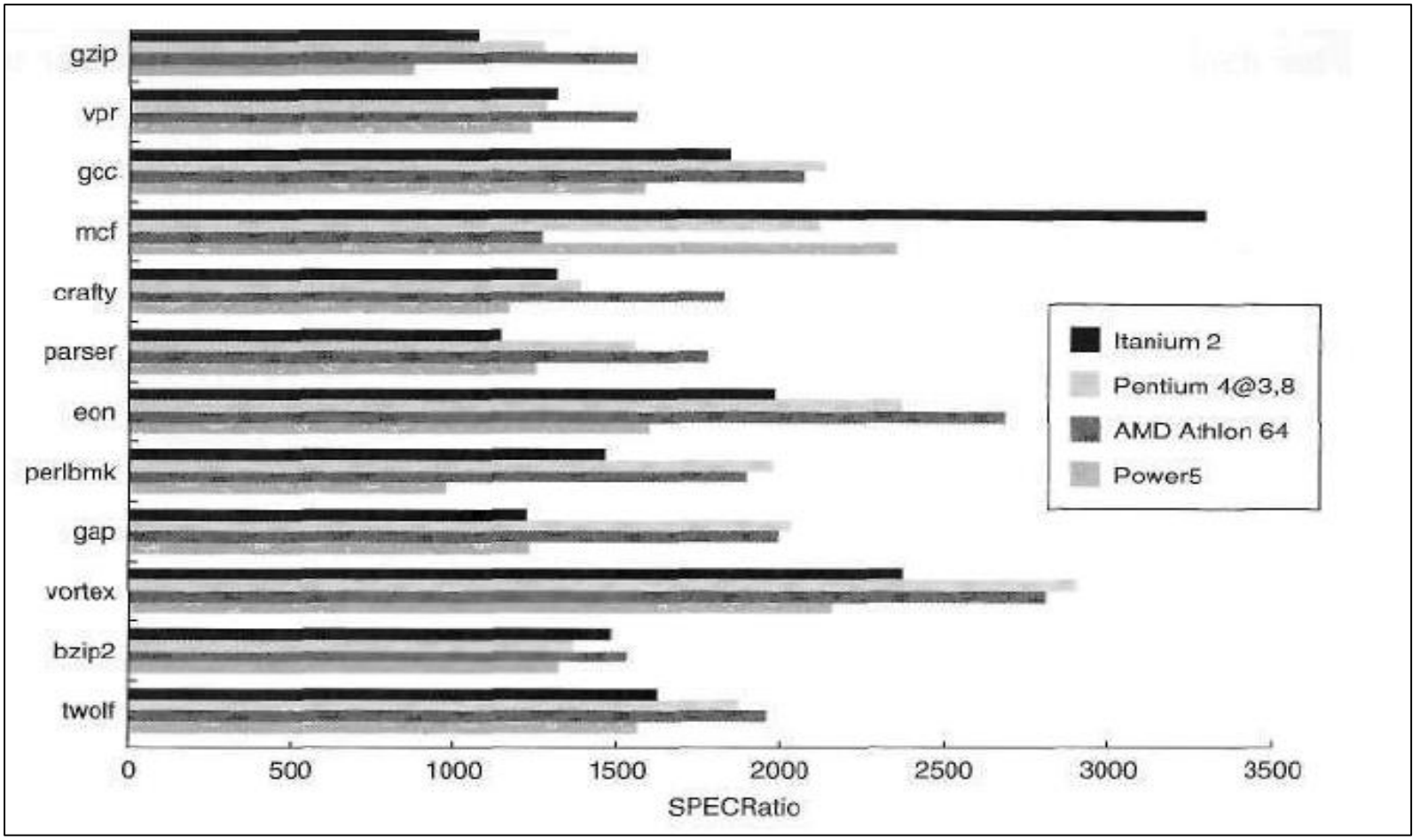
Power5 vs Power6 system architecture



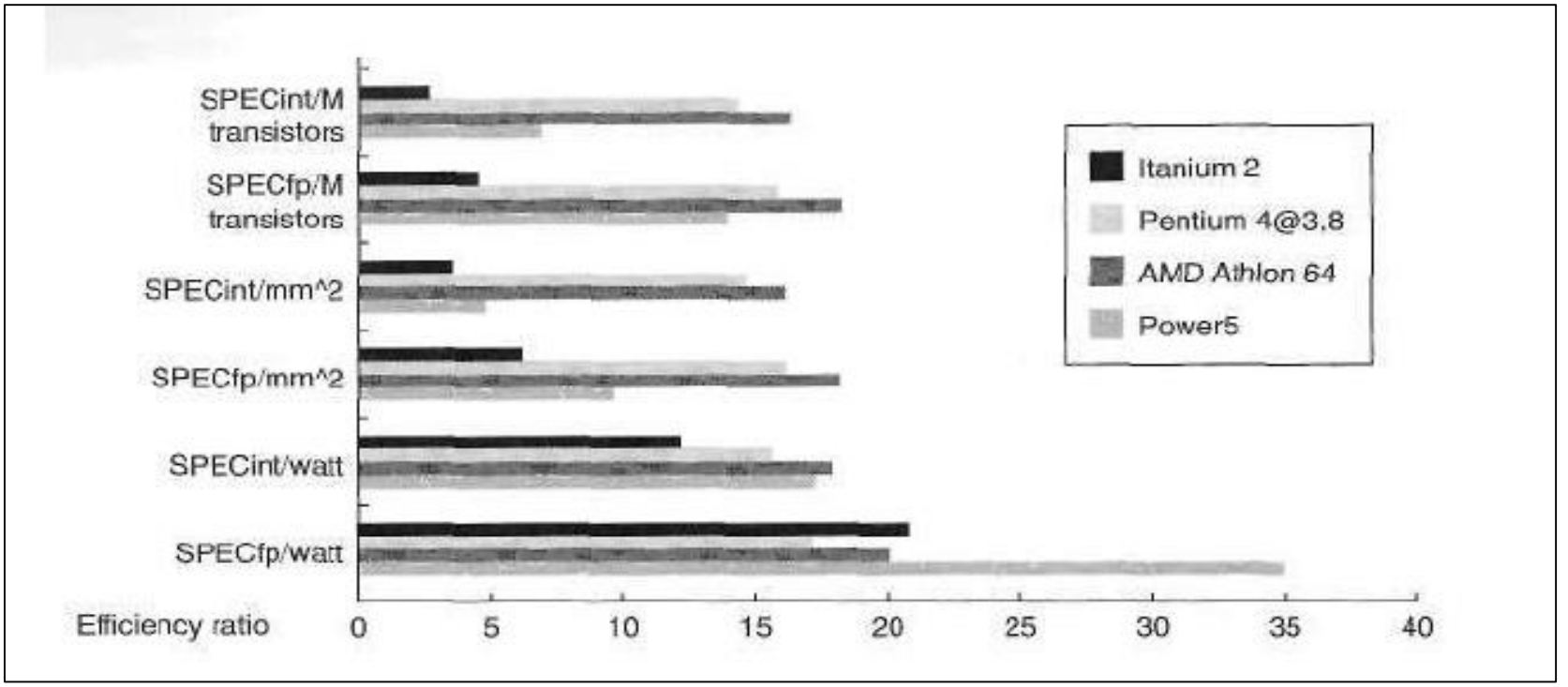
Recent multiple issue processors

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. 1FP	3.8	125M 122 mm^2	115W
AMD Athlon 64 FX-57	Speculative dynamically scheduled	3/3/4	6 int. 3FP	2.8	114M 115 mm^2	104 W
IBM Power5 1 processor	Speculative dynamically scheduled; SMT; two CPU cores/chip	8/4/8	6 int. 2FP	1.9	200M 300 mm^2 (estimated)	80 W (estimated)
Intel Itanium 2	EPIC style; primarily statically scheduled	6/5/11	9 int. 2FP	1.6	592M 423 mm^2	130W

Recent multiple issue processors



Recent multiple issue processors



IA-64 more complex, more power consuming and MUCH more complex compiler.

Unlikely that software-intensive approach will produce superior performance or extract more ILP