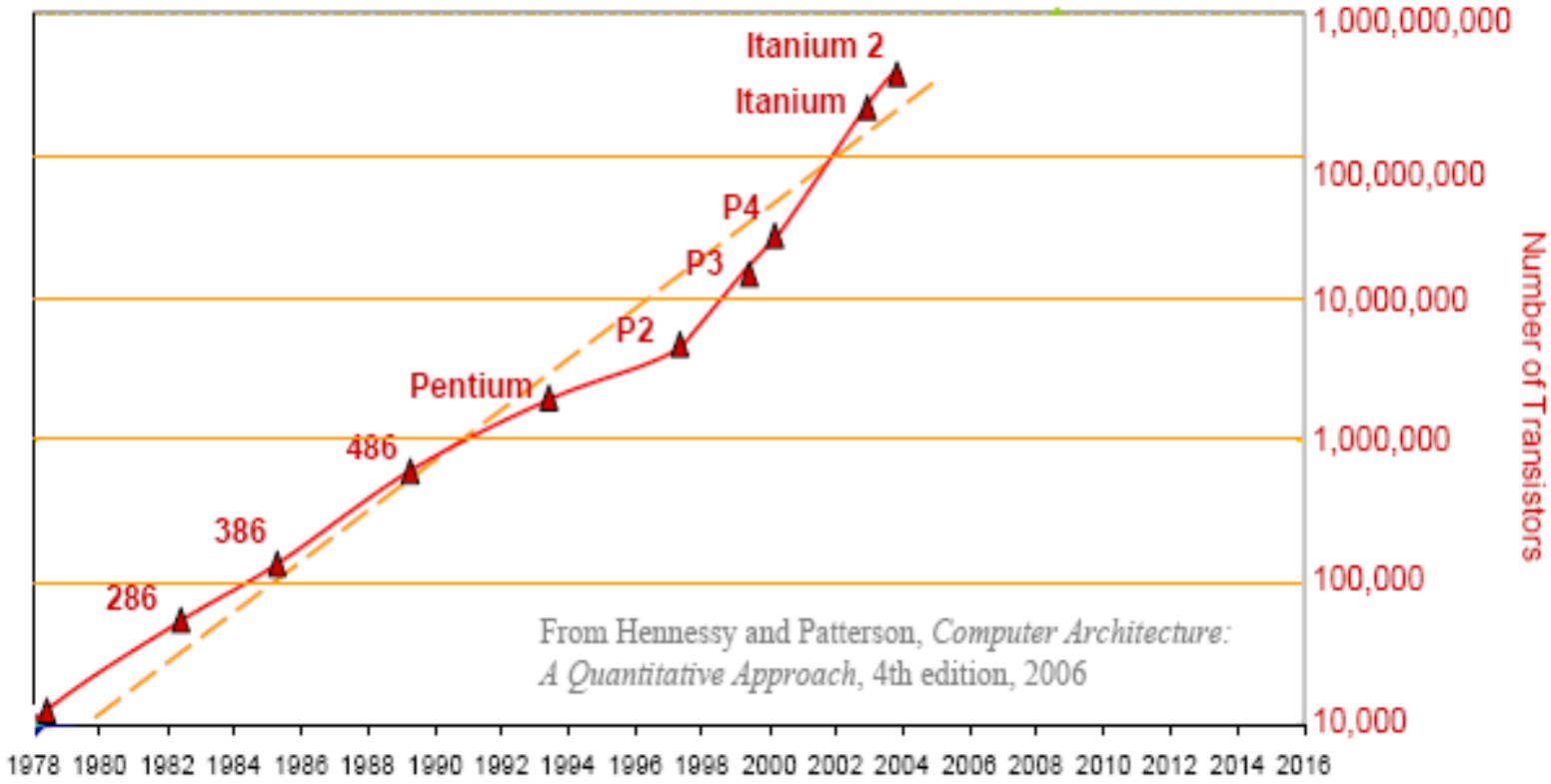
Multithreaded and multicore processors

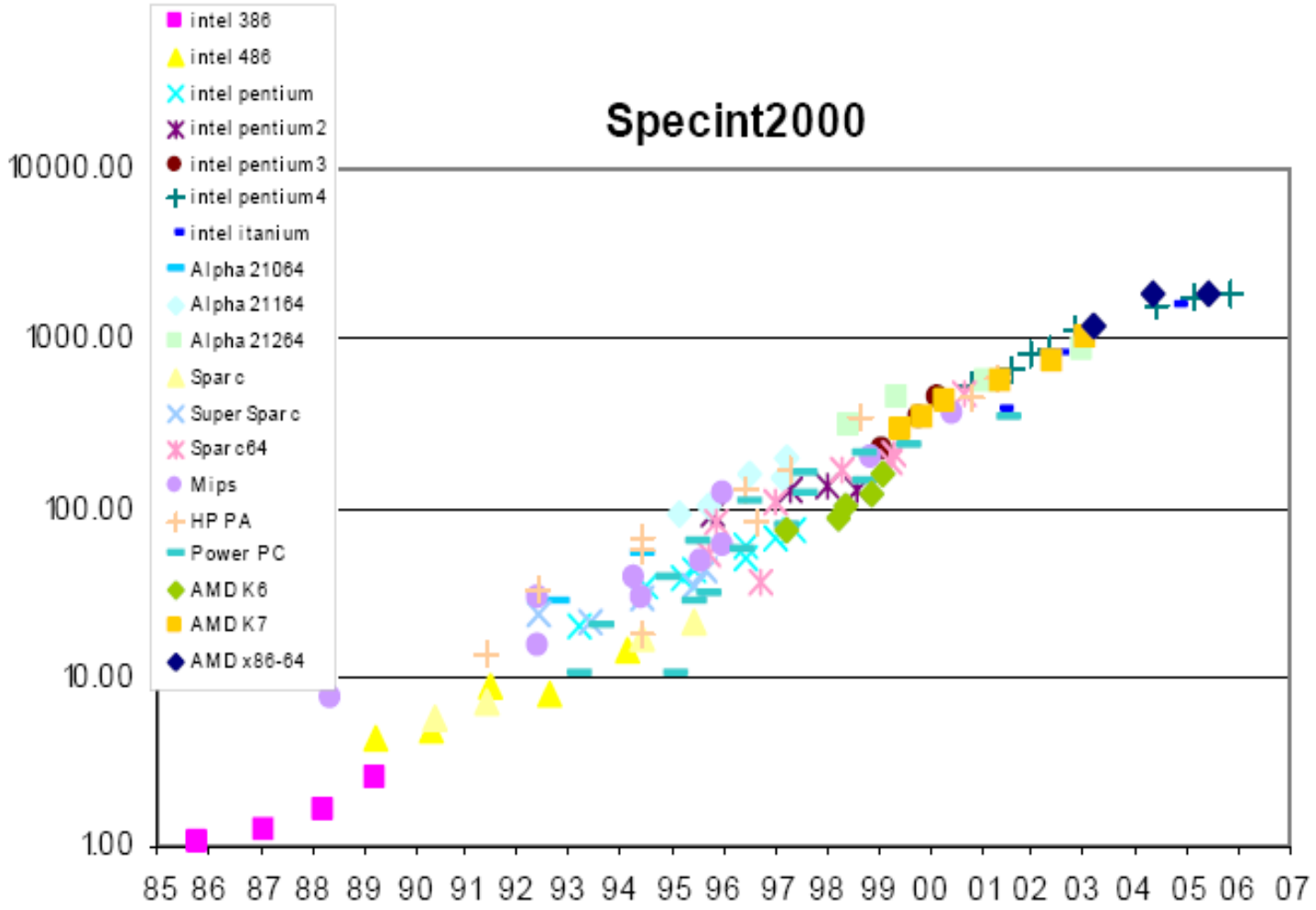
Outline

- Multithreading
- Multicore architectures
- Example architectures

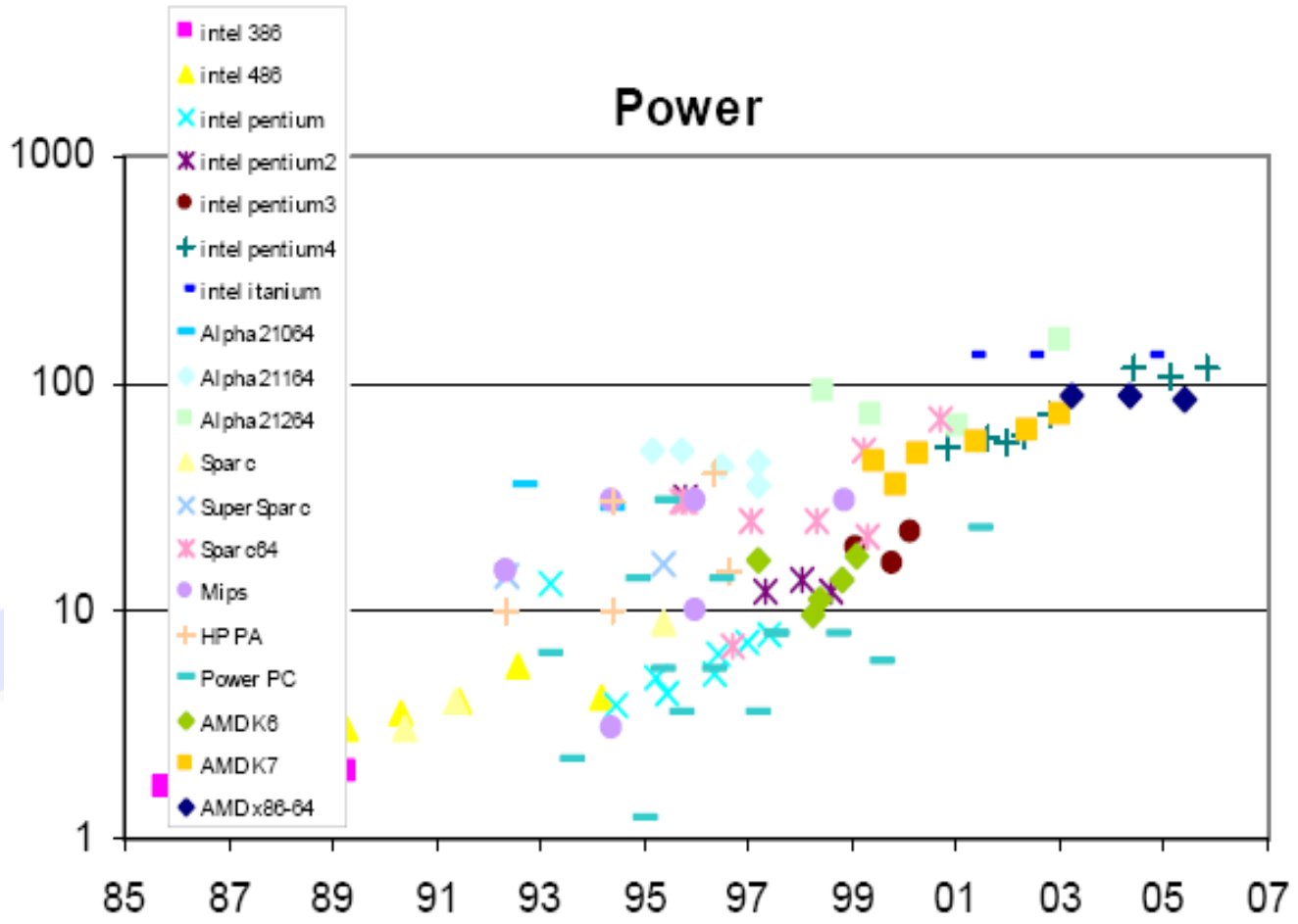
Towards Multicore: Moore's law



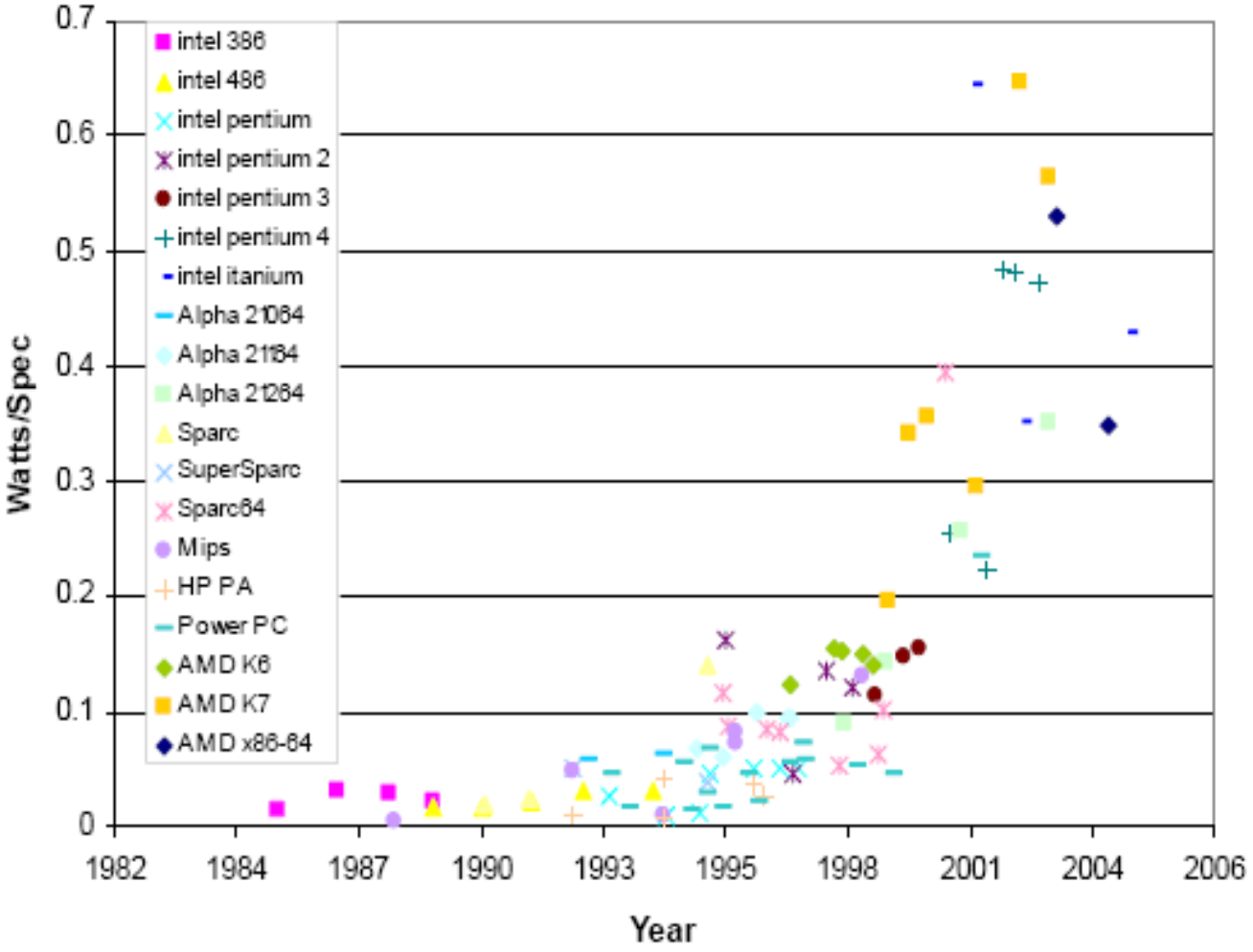
Uniprocessors: performance



Power consumption (W)

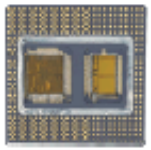


Efficiency (power/performance)



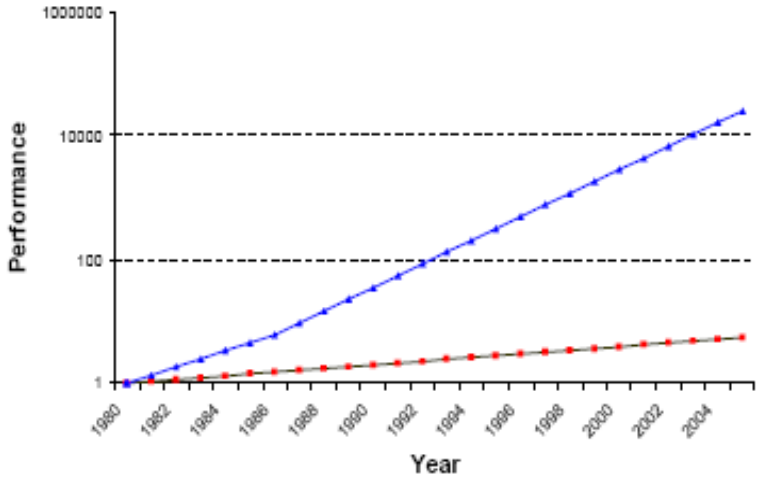
Access latency to DRAM

- Access time correlated to the speed of light
- Memory technologies are changing:
  - SRAM not easily scalable
  - DRAM does not have the best cost/bit anymore
- Efficiency problem in terms of power consumption



µProc
60%/yr.
(2X/1.5yr)

DRAM
9%/yr.
(2X/10 yrs)



Decreasing returns on investment

- ’80s: expansion of superscalar processors
  - Improvement of 50% in performance
  - Transistors used to create implicit parallelism
    - Pipelined Processors (10 CPI to 1 CPI)
- ’90s: era of decreasing returns
  - Exploit at best the implicit parallelism
    - Issue from 2 to 6 ways, issue out-of-order, branch prediction (from 1 CPI to 0.5 CPI)
    - Performance below expectations
    - Delayed and cancelled projects
- 2000: beginning of the multicore era
  - Explicit Parallelism

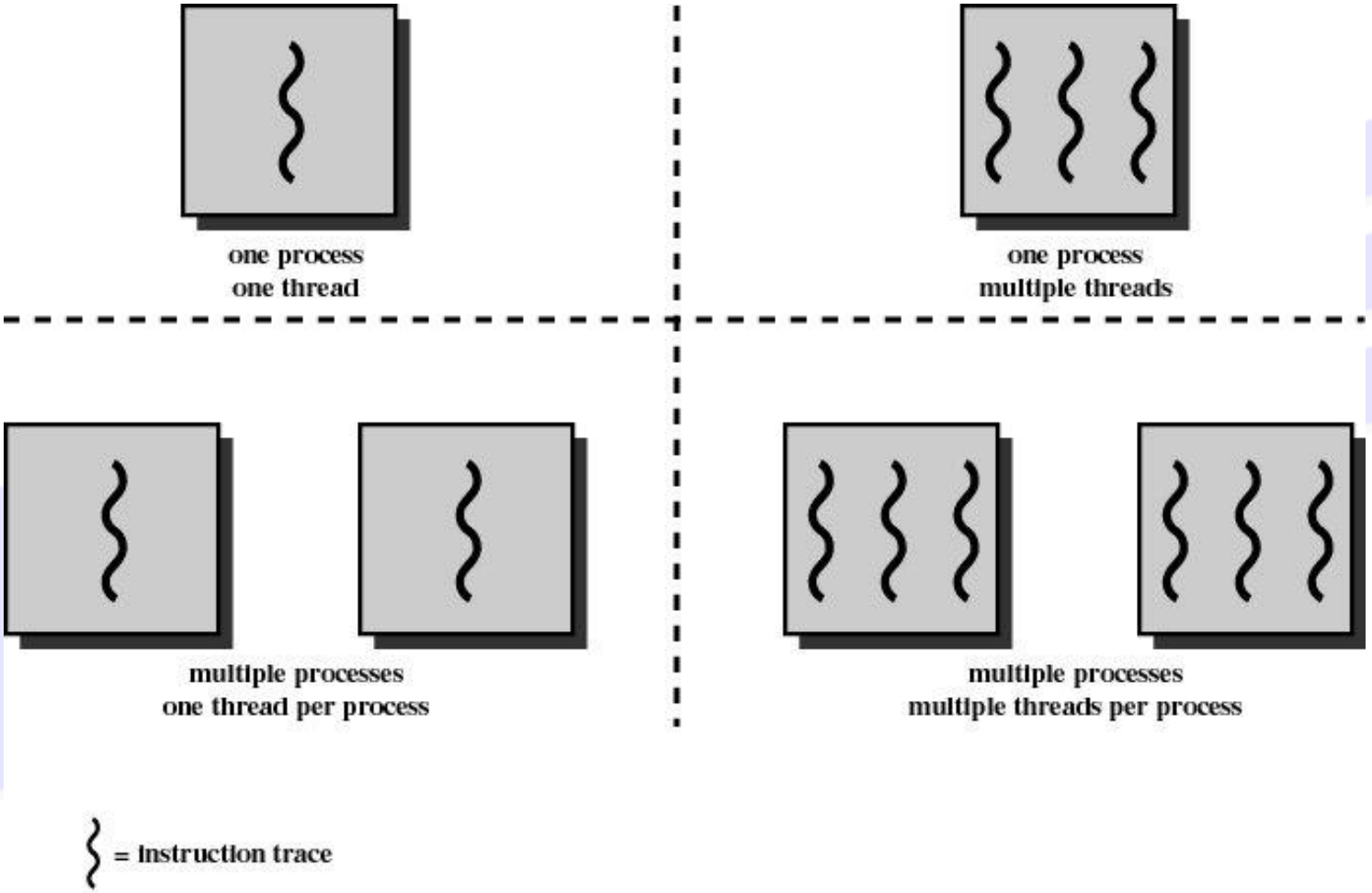
Motivations for paradigm change

- Modern processors fail to utilize execution resources well
- There is no single culprit:
  - Memory conflicts, control hazards, branch misprediction, cache miss….
- Attacking the problems one at a time always has limited effectiveness
- Need for a general latency-tolerance solution which can hide all sources of latency can have a large impact on performance

Parallel programming

- Explicit parallelism implies structuring the applications into *concurrent and communicating tasks*
- Operating systems offer support for different types of tasks. The most important and frequent are:
  - *processes*
  - *threads*
- The operating systems *implement* multitasking differently based on the characteristics of the processor:
  - single core
  - single core with multithreading support
  - multicore

Molteplicity process/thread



A first alternative: CPUs that exploit thread-level parallelism

- ***Multithreading***: more ***threads*** share the functional units of the same CPU, overlapping execution;
- The operating system sees the single physical processor as a symmetric multiprocessor constituted by two processors

Another Approach: Multithreaded Execution

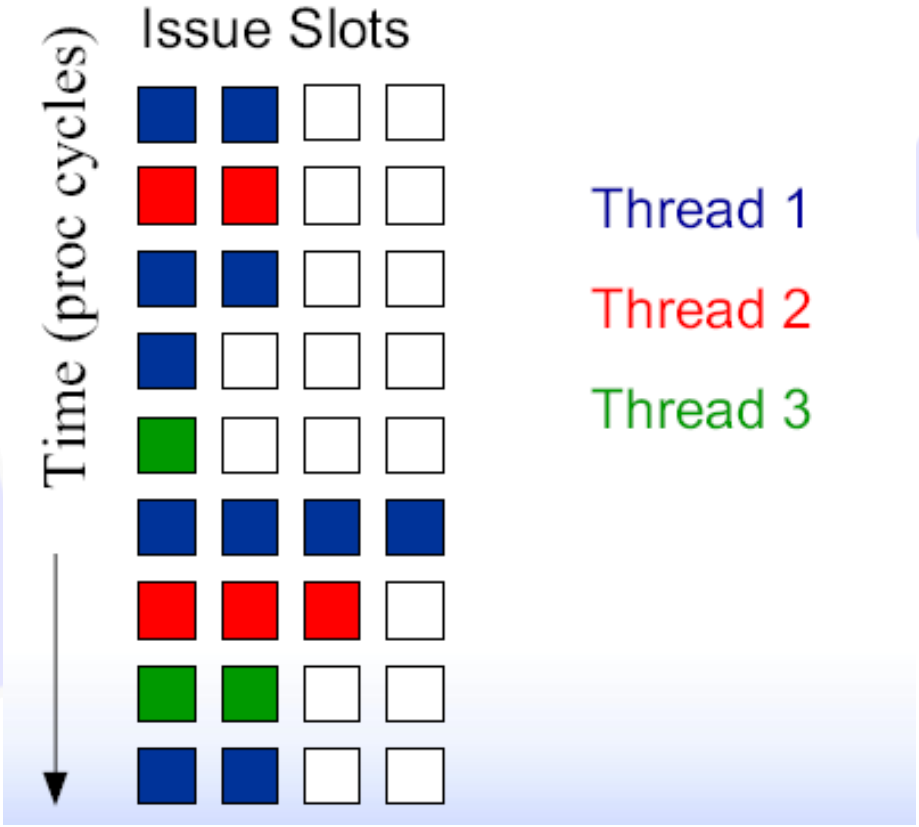
- Multithreading: multiple threads to share the functional units of 1 processor via overlapping
  - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
  - memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch; much faster than full process switch ≈ 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)

Thread-level parallelism (TLP)

Two basic approaches:

- **Fine grained multithreading**: switches from one thread to the other at each instruction – the execution of more threads is interleaved (often the switching is performed taking turns, skipping one thread if there is a stall)
- The CPU must be able to change thread at every clock cycle. It is necessary to duplicated the hardware resources.

Example of fine grained MT in superscalar processor



Thread level parallelism (TLP)

Fine grained multithreading

- 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.

Thread level parallelism

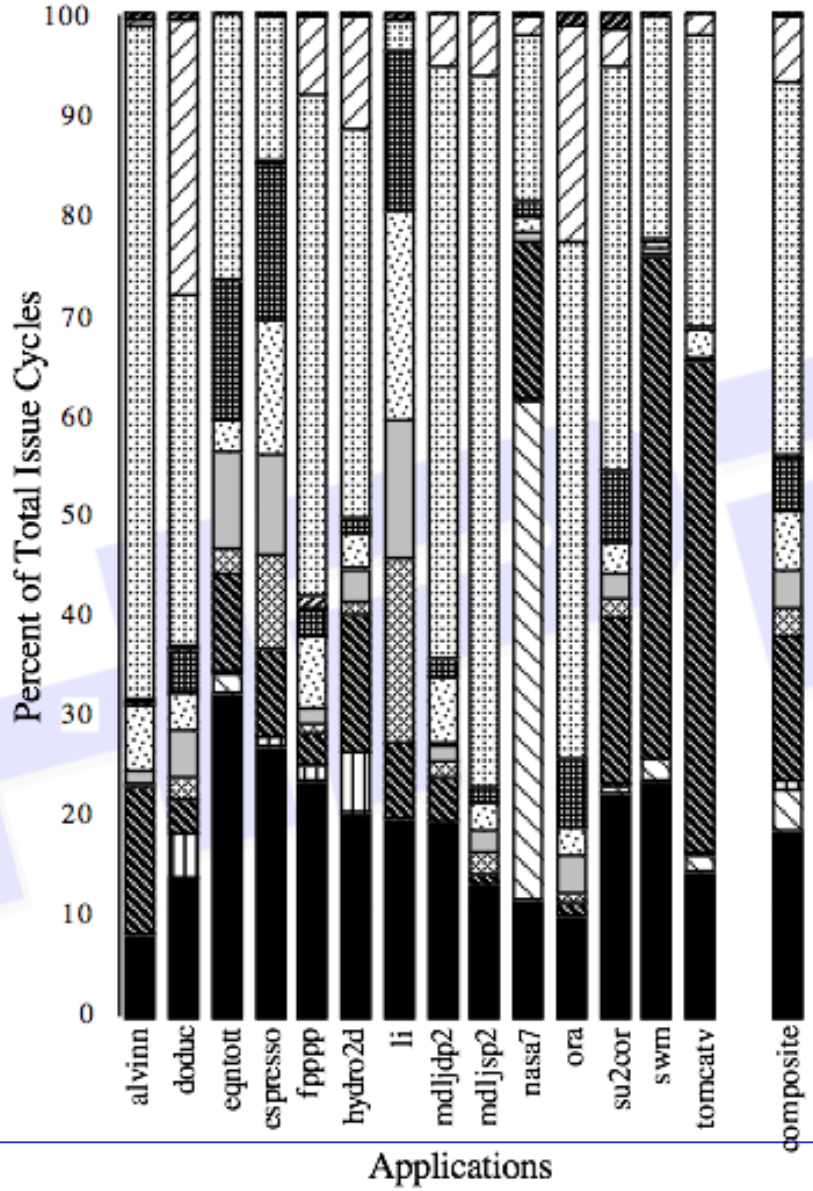
- **Coarse grained multithreading**: switching from one thread to another occurs only when there are long stalls – e.g., for a miss on the second level cache.
- Two threads share many system resources (e.g., architectural registers) ⇨ the switching from one thread to the next requires different clock cycles to save the context.

Thread level parallelism

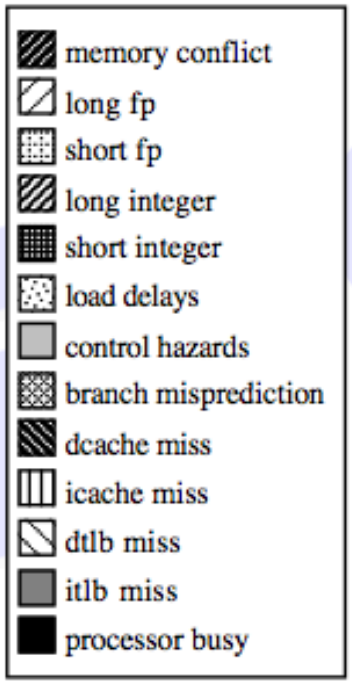
Coarse grained multithreading:

- Advantage: in normal conditions the single thread is not slowed down;
  - 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: for short stalls it does not reduce the throughput loss – the CPU starts the execution of instructions that belonged to a single thread, when there is one stall it is necessary to empty the pipeline before starting the new thread.
- Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill << stall time

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?

Thread level parallelism: simultaneous multithreading

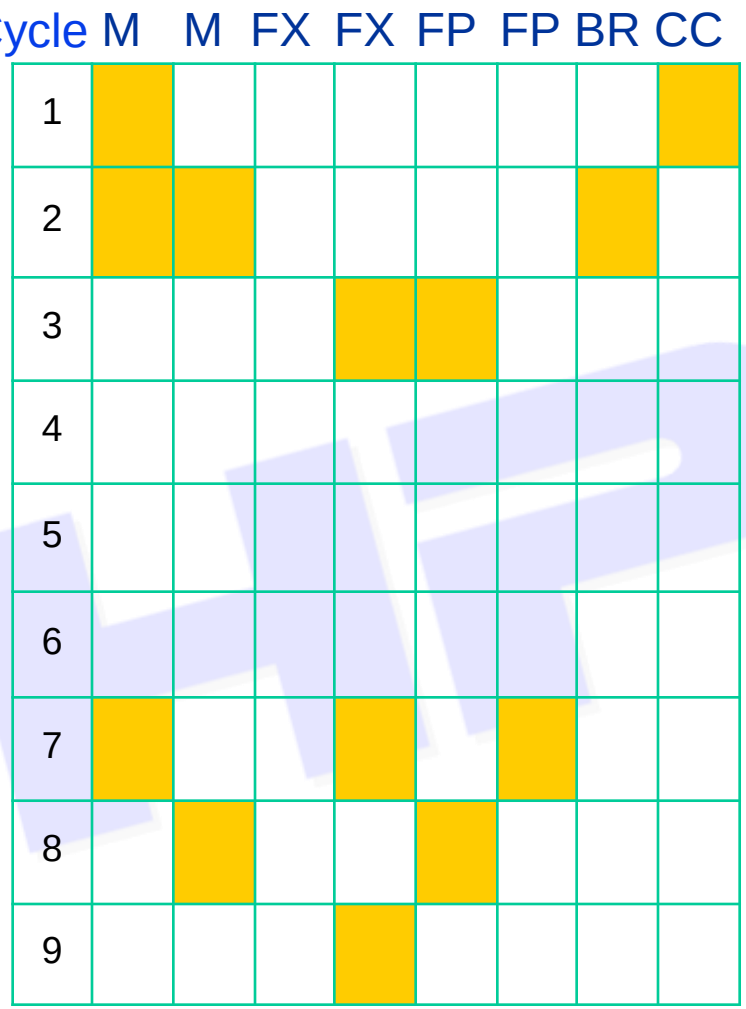
- Uses the resources of one superscalar processor to exploit simultaneously ILP and TLP.
- Key motivation: a CPU today has more functional resources than what one thread can in fact use
- Thanks to register renaming and dynamic scheduling, more independent instructions to different threads may be issued without worrying about dependences (that are solved by the hardware of the dynamic scheduling).
- Simultaneously schedule instructions for execution from all threads

Simultaneous Multithreading (SMT)

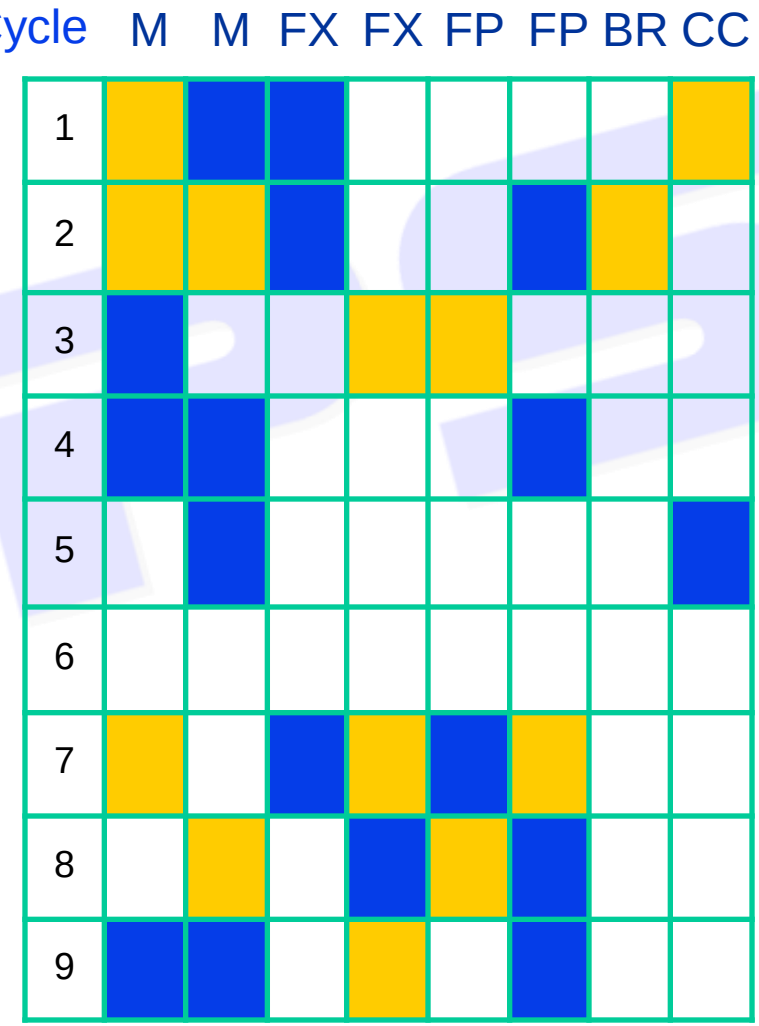
- Simultaneous multithreading (SMT): insight that dynamically scheduled processor already has many HW mechanisms to support multithreading
  - Large set of virtual registers that can be used to hold the register sets of independent threads
  - Register renaming provides unique register identifiers, so 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

Simultaneous Multi-threading ...

One thread, 8 units

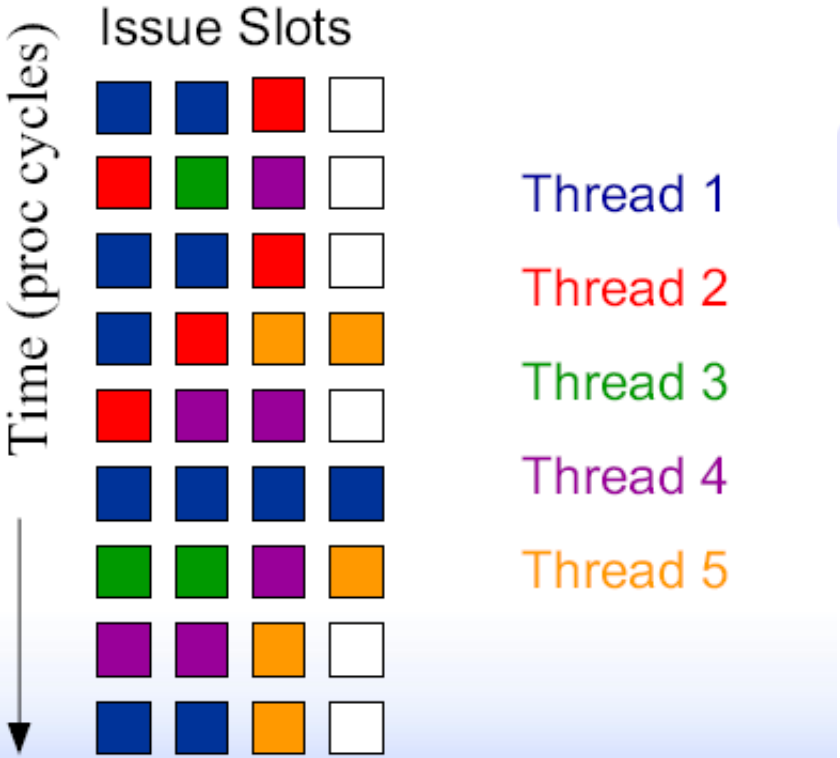


Two threads, 8 units



M = Load/Store, FX = Fixed Point, FP = Floating Point, BR = Branch, CC = Condition Codes

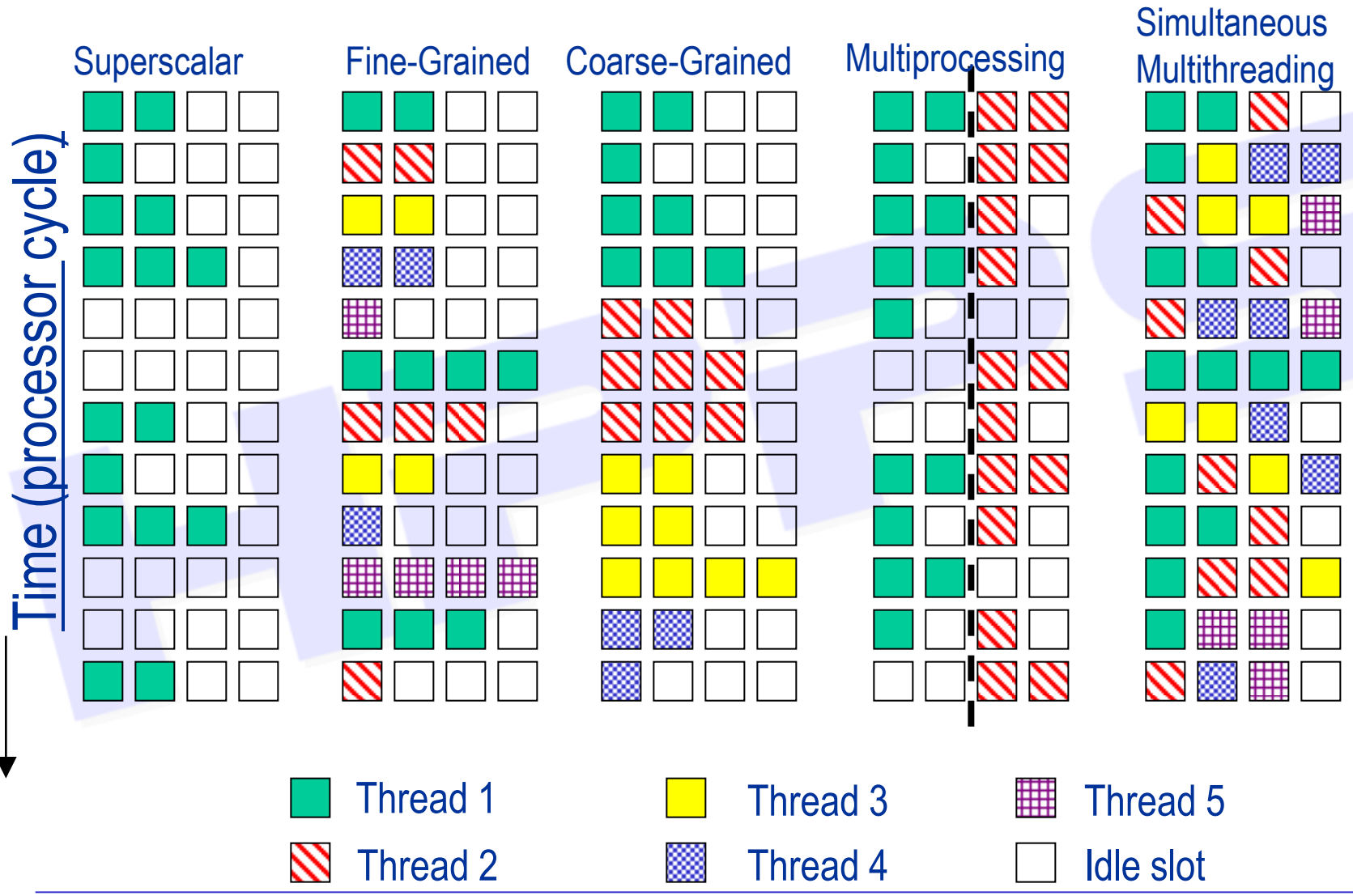
Thread level parallelism: simultaneous multithreading



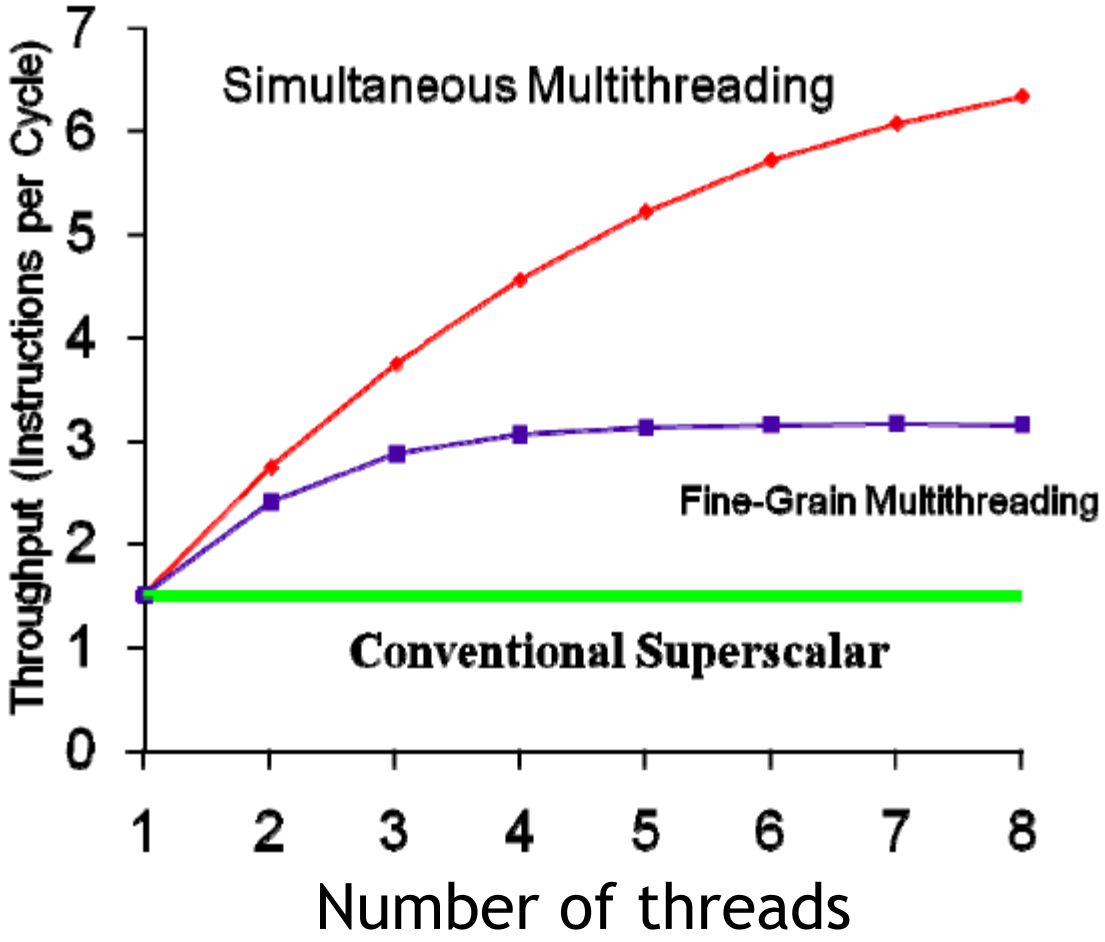
Simultaneous multithreading (SMT)

- The system can be dynamically adapted to the environment, allowing (if possible) the execution of instructions from each thread, and allowing that the instructions of a single thread used all functional units if the other thread incurs in a long latency event.
- More threads use the issues possibilities of the CPU at each cycle; ideally, the exploitation of the issues availabilities is limited only by the unbalance between resources requests and availabilities.

Multithreaded Categories



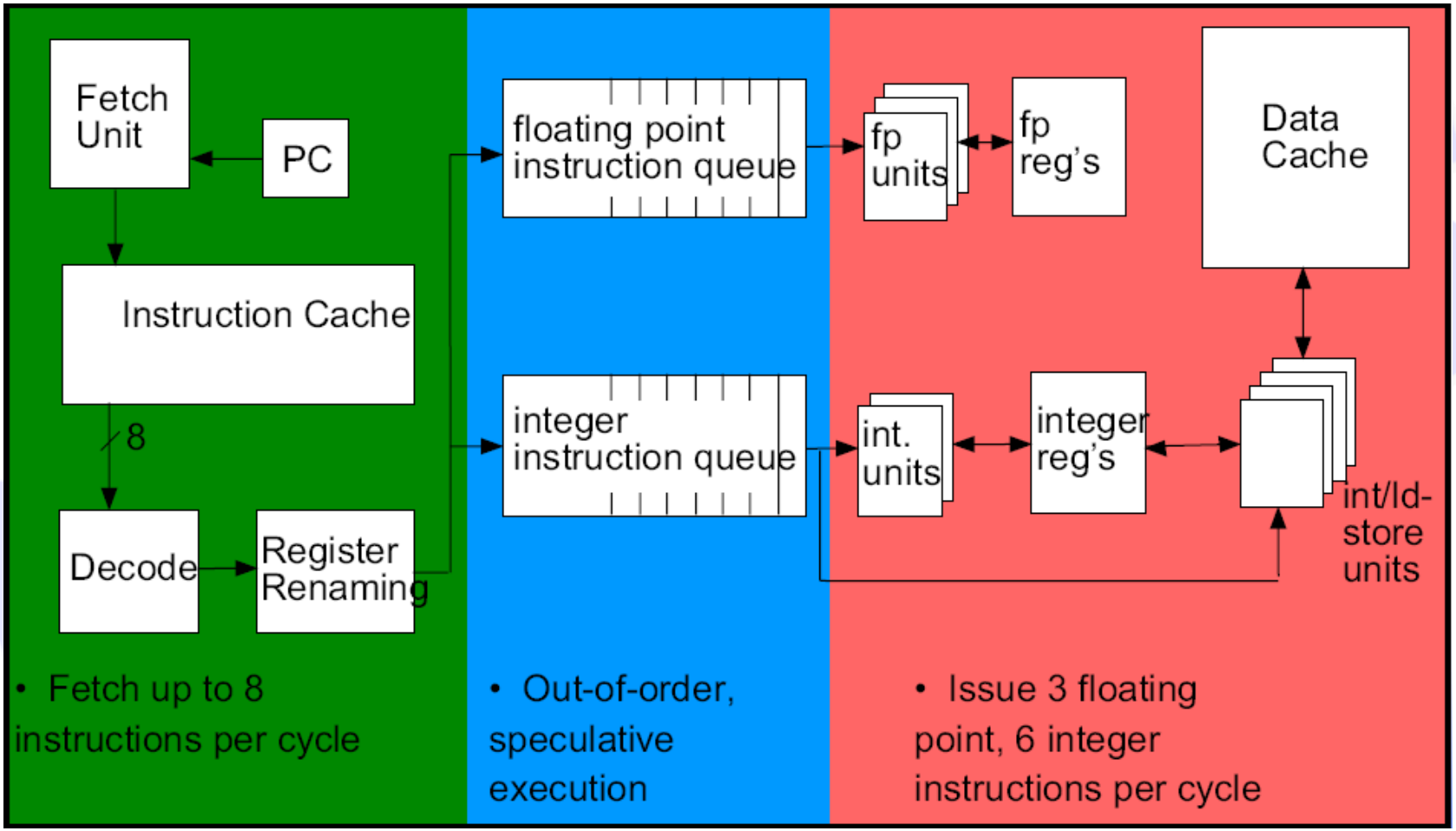
Performance comparison



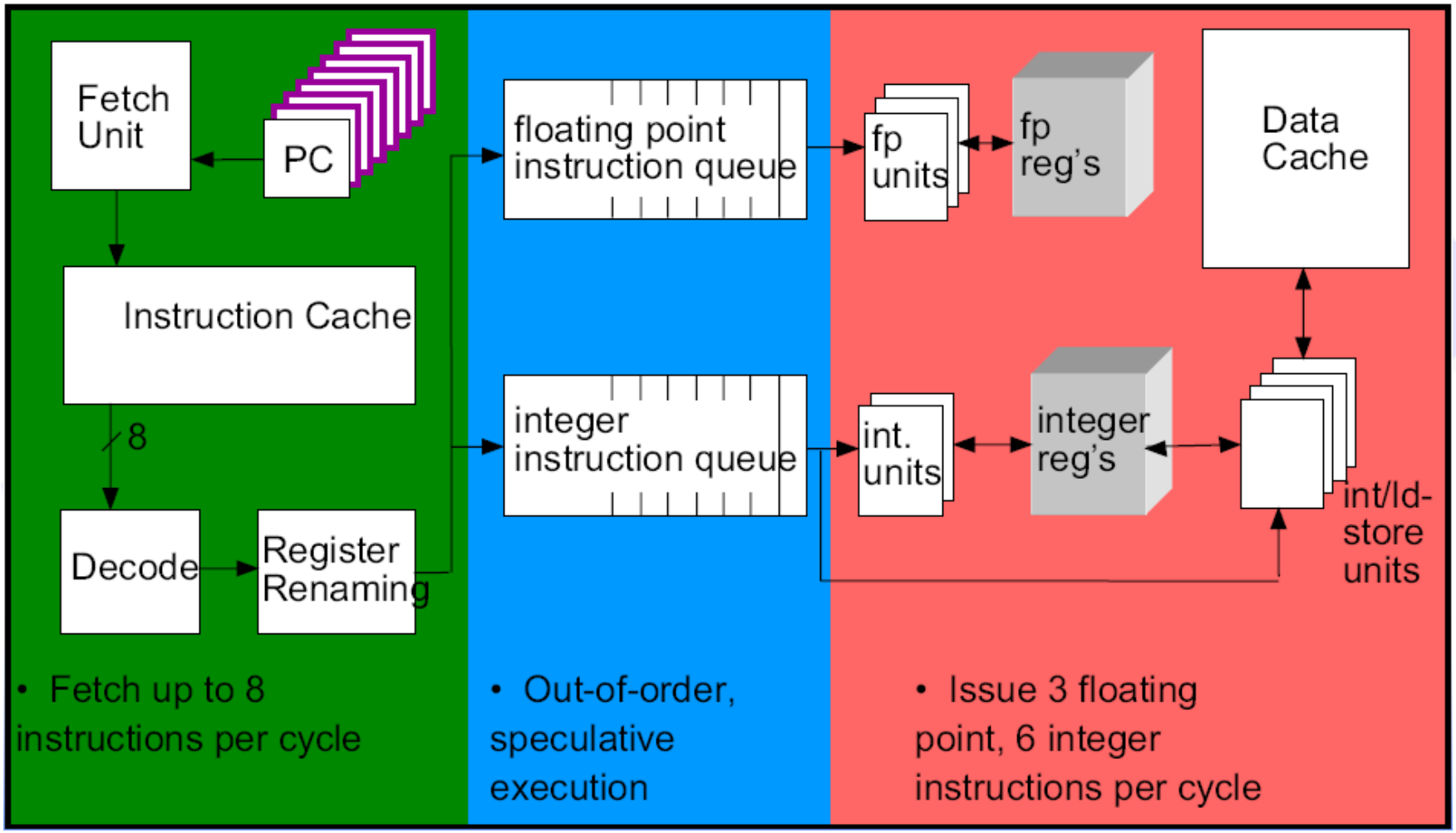
Goals for an SMT architecture implementation

- Minimize the architectural impact on conventional superscalar design
- Minimize the performance impact on a single thread
- Achieve significant throughput gains with many threads

A conventional superscalar architecture



An SMT architecture



Improved Fetch

- The Fetch Unit in a SMT architecture has two distinct advantages over a conventional architecture:
  - Can fetch from multiple threads at once
  - Can choose which threads to fetch

  - Fetching from 2 threads/cycle achieves most of the performance from multiple-thread fetch.

  - How to identify the best candidate to fetch?
  - ✓ fewest unresolved branches
  - ✓ Fewest load misses
  - ✓ Fewest instructions in queue

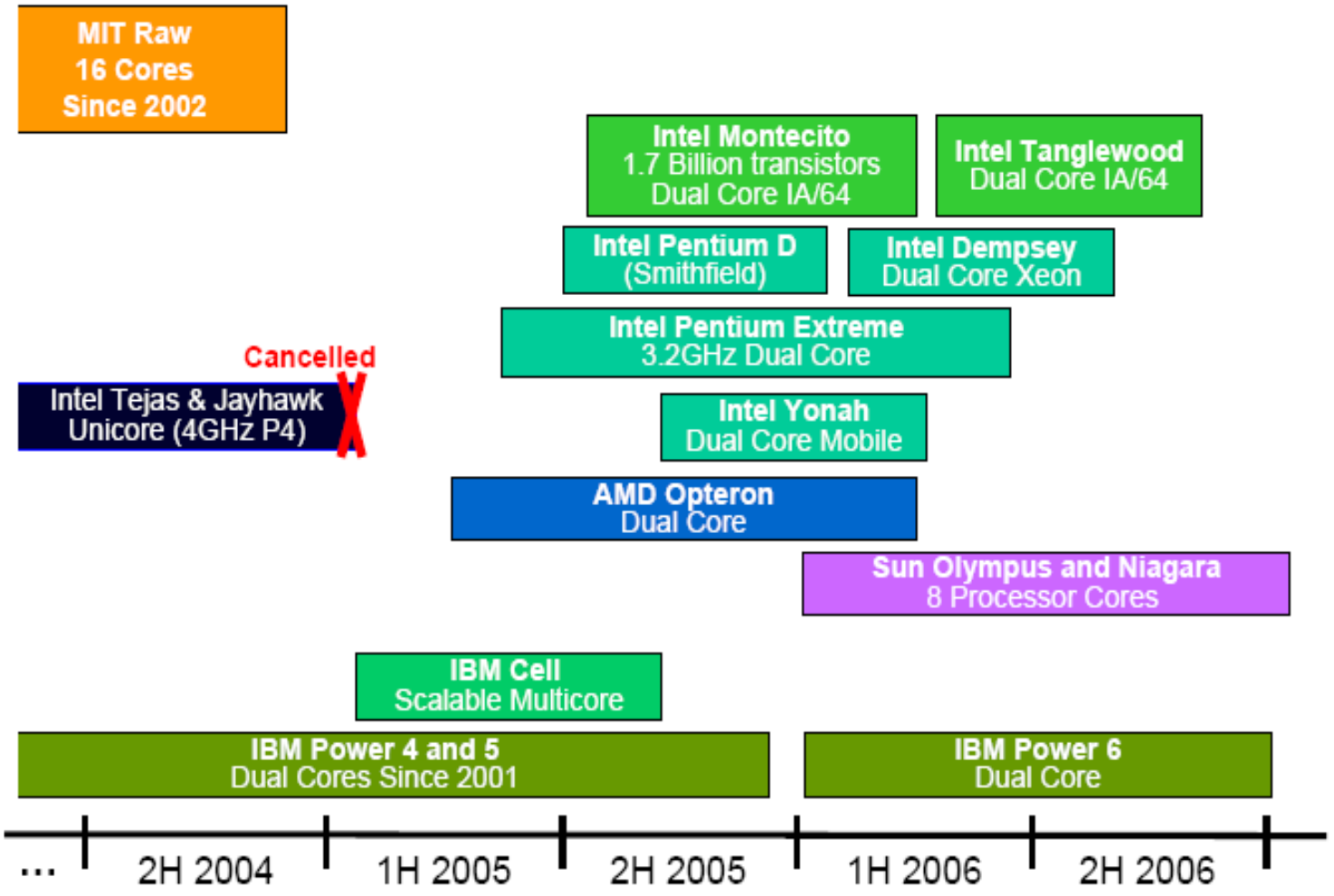
Design Challenges in SMT

- 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

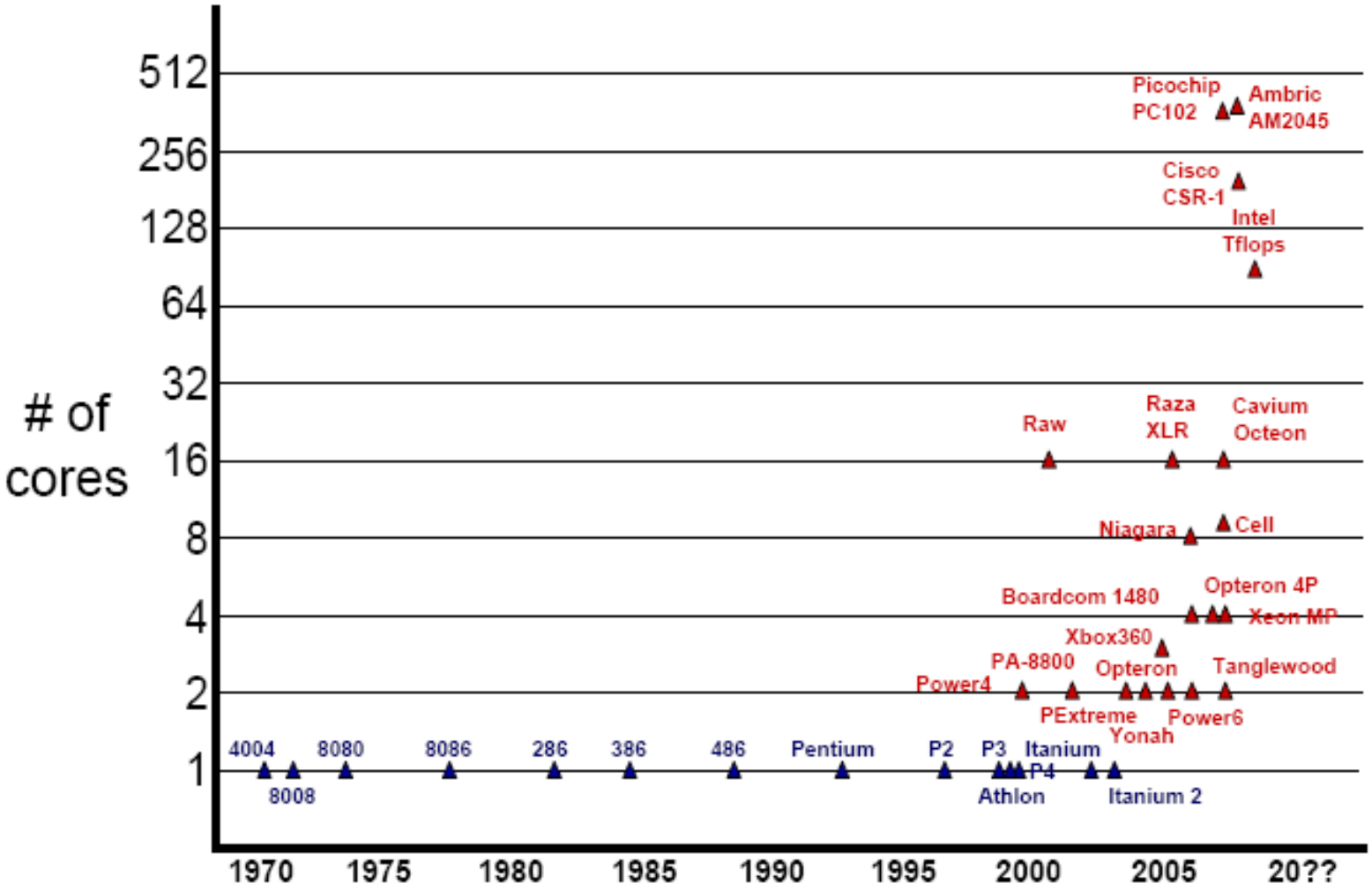
Why multicore?

- Difficult to increase performance and clock frequency of the single core
- Deep pipeline:
  - Heat dissipation problems
  - Speed light transmission problems in wires
  - Difficulties in design and verification
  - Requirement of very large design groups
- Many new applications are multi-threaded

Multicore


MIT Raw
16 Cores
Since 2002
Intel Montecito
1.7 Billion transistors
Dual Core IA/64
Intel Tanglewood
Dual Core IA/64
Intel Pentium D
(Smithfield)
Intel Dempsey
Dual Core Xeon
Intel Pentium Extreme
3.2GHz Dual Core
Cancelled
Intel Tejas & Jayhawk
Unicore (4GHz P4)
Intel Yonah
Dual Core Mobile
AMD Opteron
Dual Core
Sun Olympus and Niagara
8 Processor Cores
IBM Cell
Scalable Multicore
IBM Power 4 and 5
Dual Cores Since 2001
IBM Power 6
Dual Core
...
2H 2004
1H 2005
2H 2005
1H 2006
2H 2006

Multicore


# of cores
512
256
128
64
32
16
8
4
2
1
1970
1975
1980
1985
1990
1995
2000
2005
20??
Picochip PC102
Ambric AM2045
Cisco CSR-1
Intel Tflops
Raw
Raza XLR
Cavium Octeon
Niagara
Cell
Boardcom 1480
Opteron 4P
Xeon MP
Xbox360
PA-8800
Opteron
Tanglewood
Power4
PExtreme
Power6
Yonah
4004
8008
8080
8086
286
386
486
Pentium
P2
P3
Itanium
P4
Athlon
Itanium 2

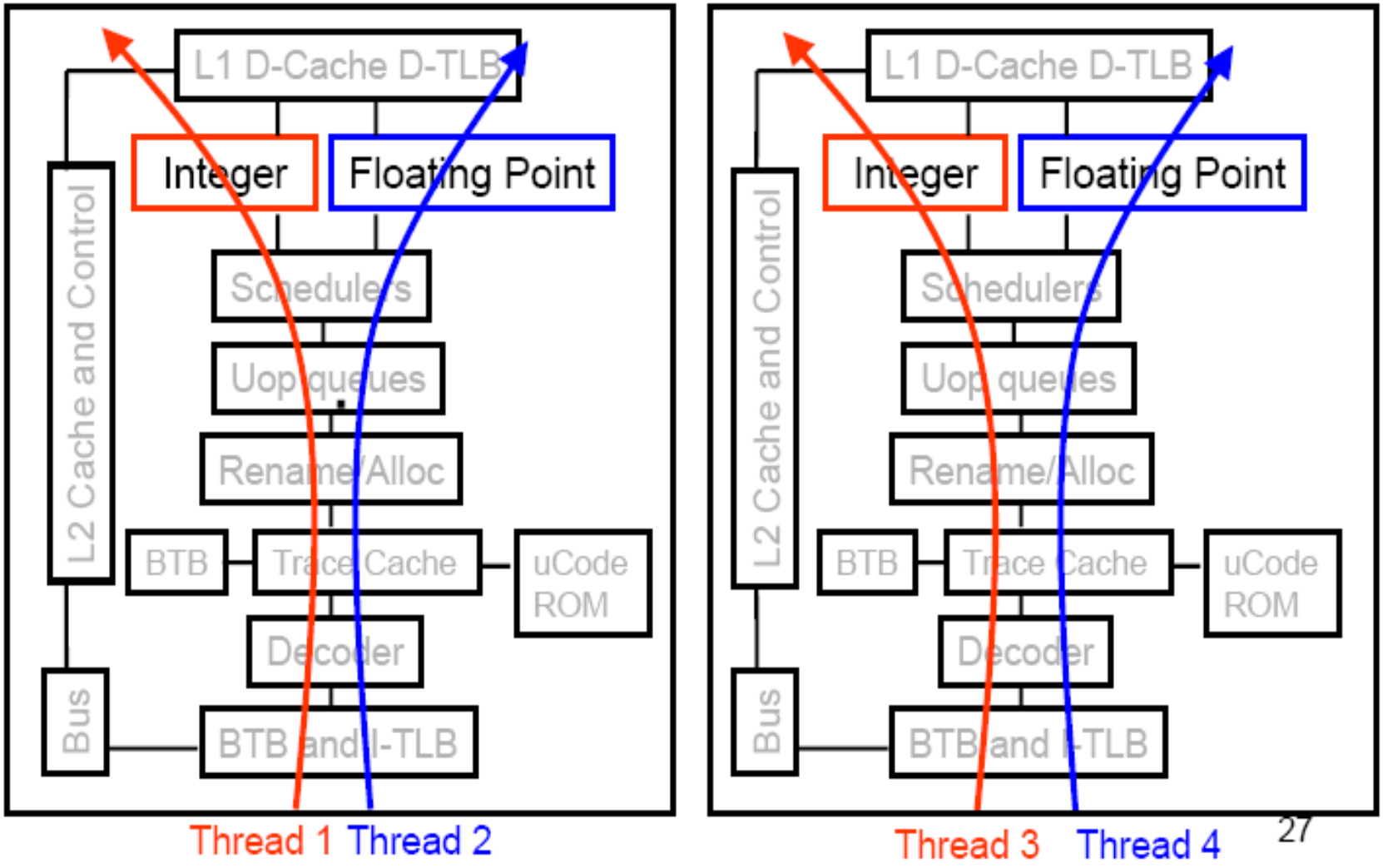
What kind of parallelism?

- MIMD: Multiple Instruction Multiple Data
  - Different cores execute different threads and interact with different parts of the memory
  - Shared memory
    - Intel Yonah, AMD Opteron
    - IBM Power 5 & 6
    - Sun Niagara
  - Shared newtork
    - MIT Raw
    - IBM Cell
  - Mini Cores
    - Intel Tflops
    - Picochip

Multicore and SMT

- Processors can use SMT
- N. threads: 2, 4 or sometime 8
  - (called hyper-threads by Intel)
- Memory hierarchy:
  - If only multithreading: all caches are shared
  - Multicore:
    - Cache L1 private
    - Cache L2 private in some architectures and shared in others
    - Memory always shared

SMT Dual core: 4 concurrent threads

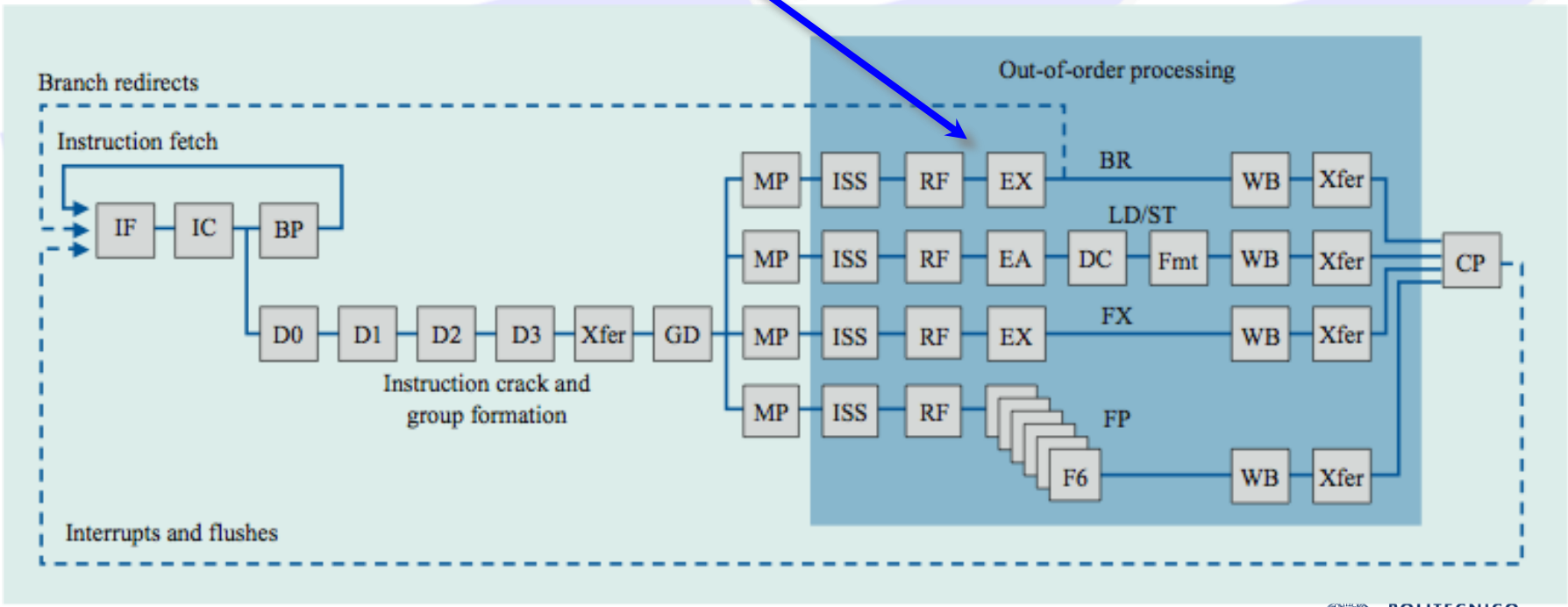


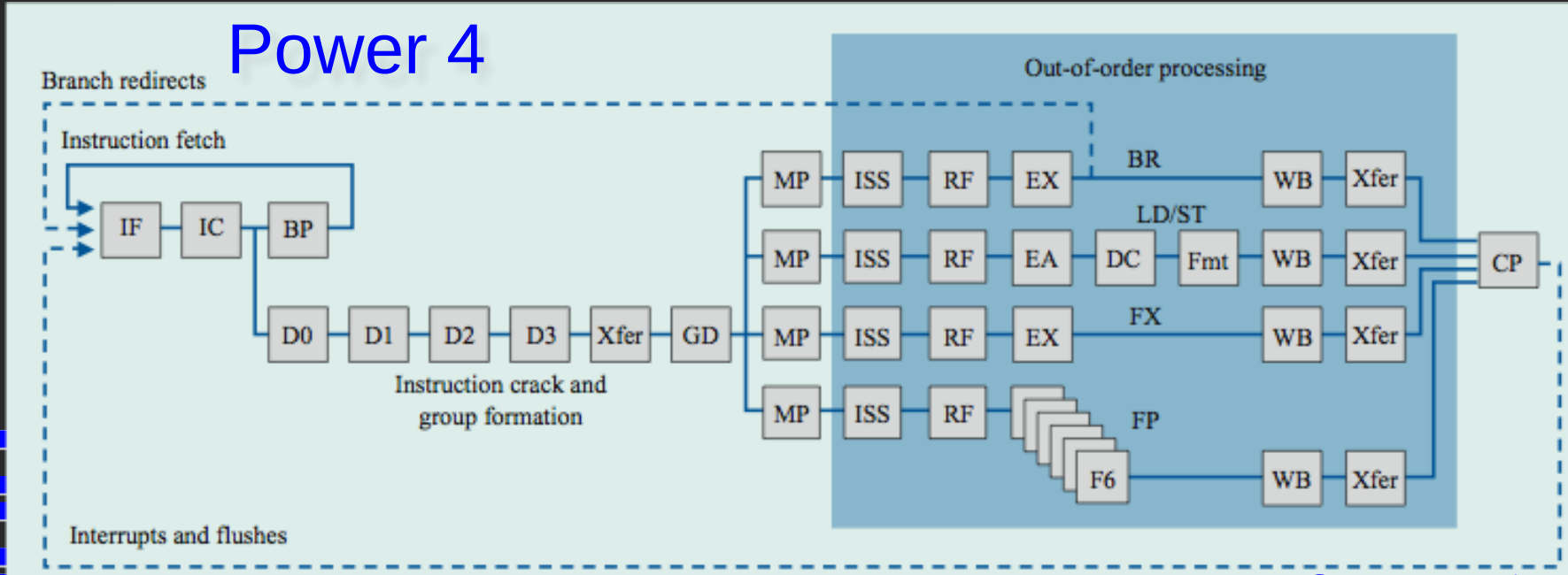
IBM architectures Power 4 and Power 5

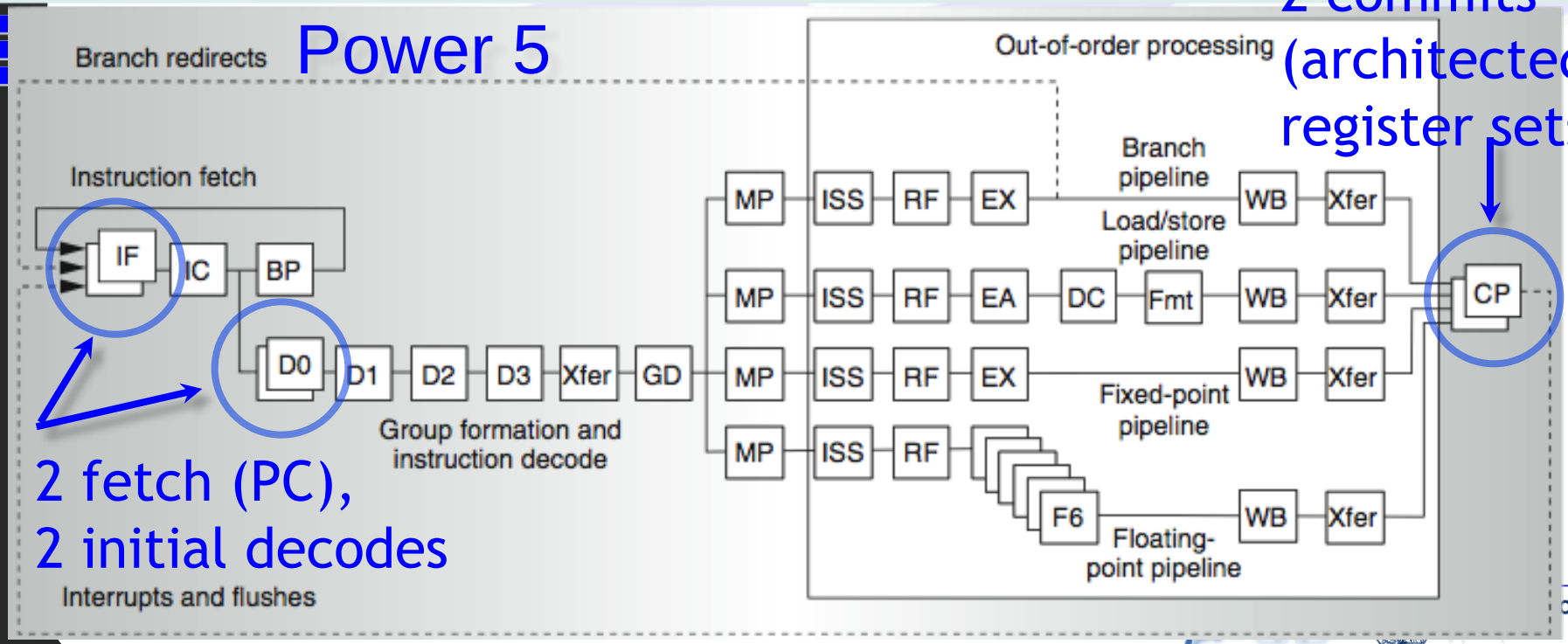
- Power 4 and Power 5: the chip includes *two* processors, each one implementing a two-way SMT
- Higher multithreading (more than 2 threads): possible, but reduced marginal increase in performance (performance may decrease due to *cache thrashing* – the data of one thread may send out the data necessary to another thread).

Power 4

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



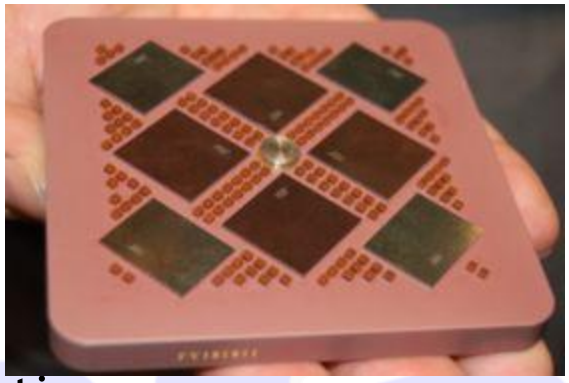




2 commits (architected register sets)

2 fetch (PC),
2 initial decodes

Example: IBM Power 5



IF = instruction fetch,
IC = instruction cache,
BP = branch predict,
D0 = decoding stage 0,
Xfer = transfer,
GD = dispatch a group of instructions to execution,
MP = mapping,
ISS = instruction issue,
RF = register file read,
EX = execution,
EA = address computation,
DC = data cache,
F6 = floating point pipeline (6 stages),
Fmt = data format,
WB = write back,
CP = commit of a group of instructions

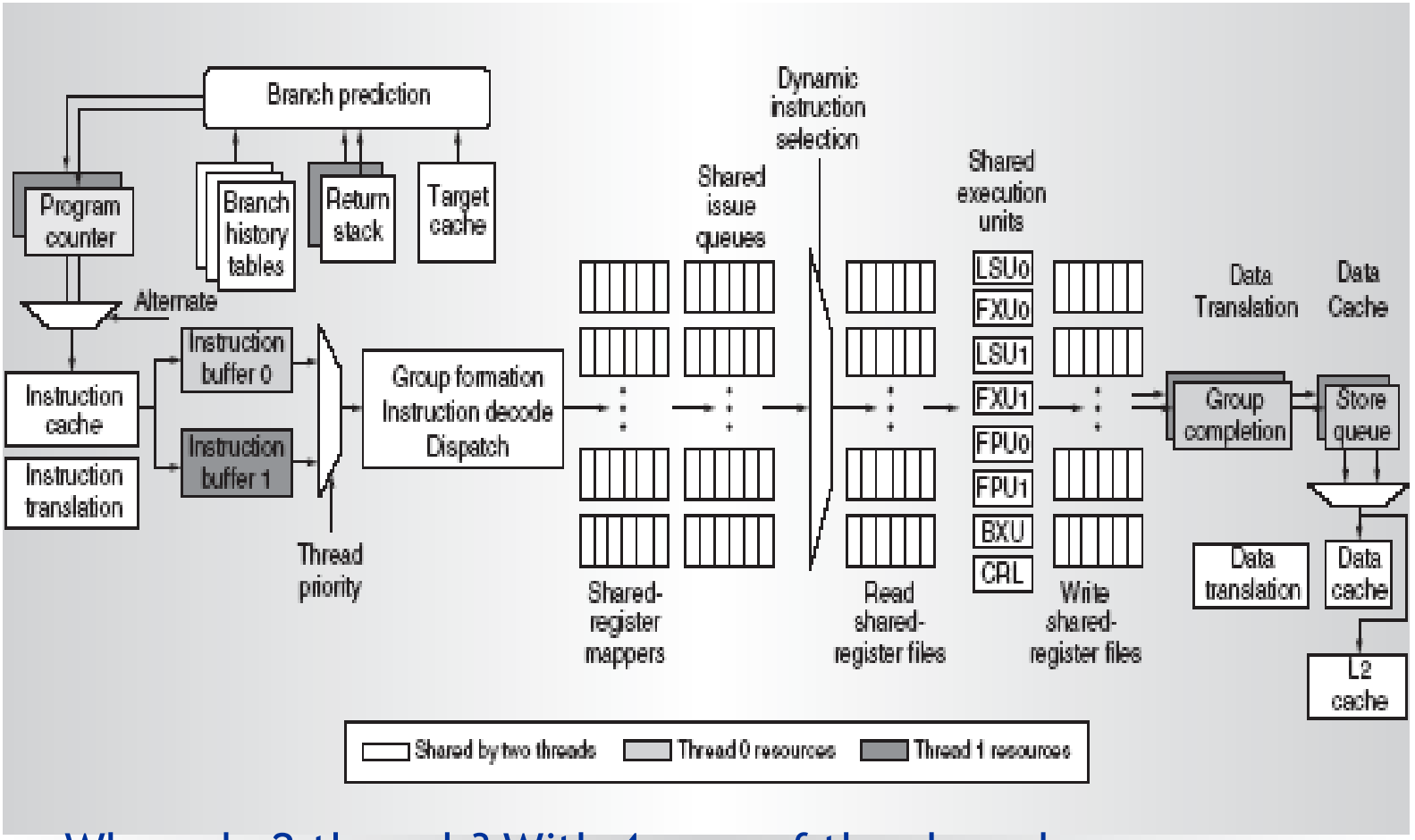
Thread level parallelism: IBM Power 5

- SMT mode: Power 5 uses two different address registers in read to store the PC of the two different threads.
- Instruction fetch (IF) alternates between the 2 threads; in a given cycle, all instructions read come from the same thread.
- ST mode: it uses only one PC, at each cycle we read instructions of one thread.
- The two threads share instruction cache istruzioni e translation unit of the instructions.

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

Instruction flow in Power 5

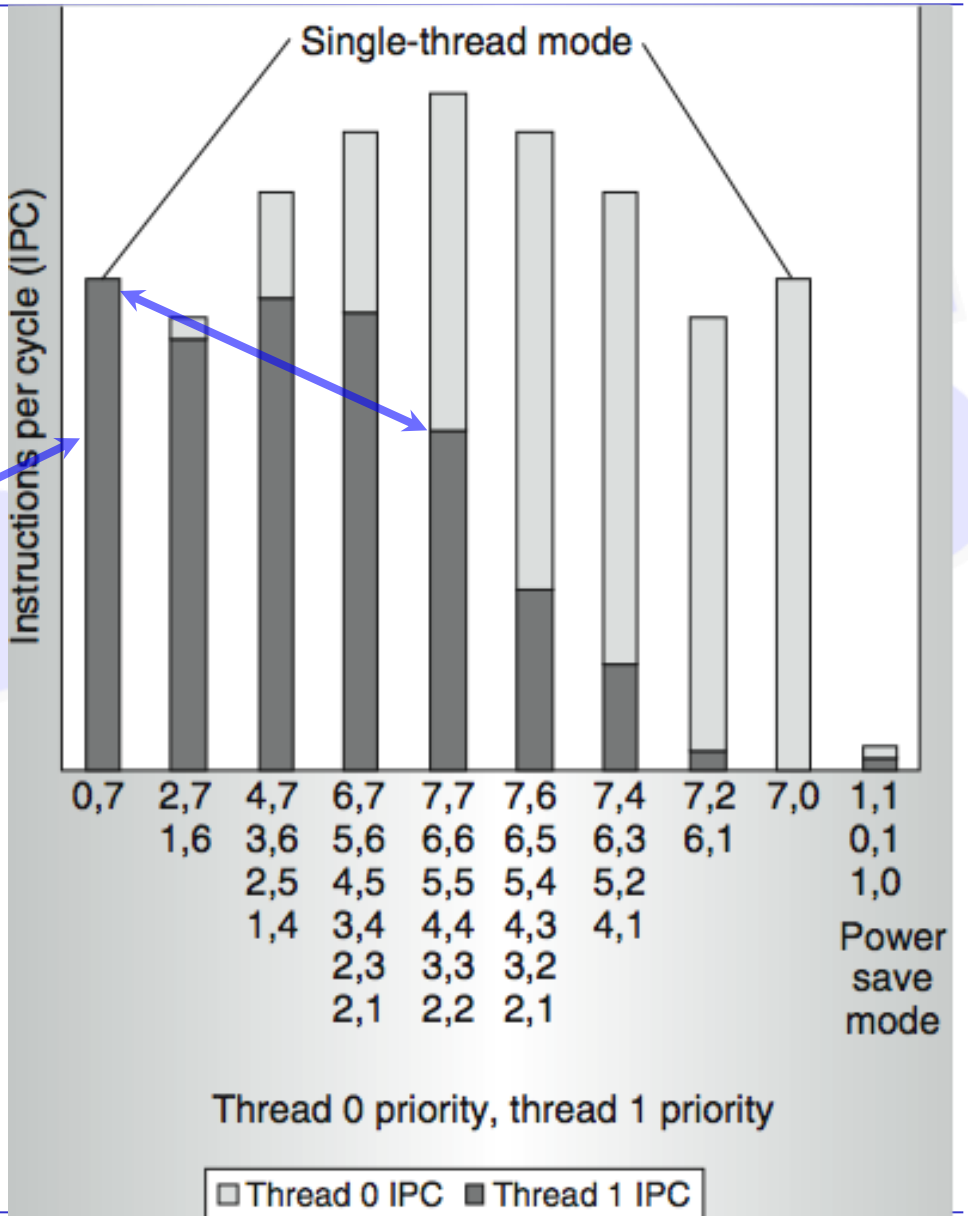


Why only 2 threads? With 4, one of the shared resources (physical registers, cache, memory bandwidth) would be prone to bottleneck

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.



Initial Performance of SMT

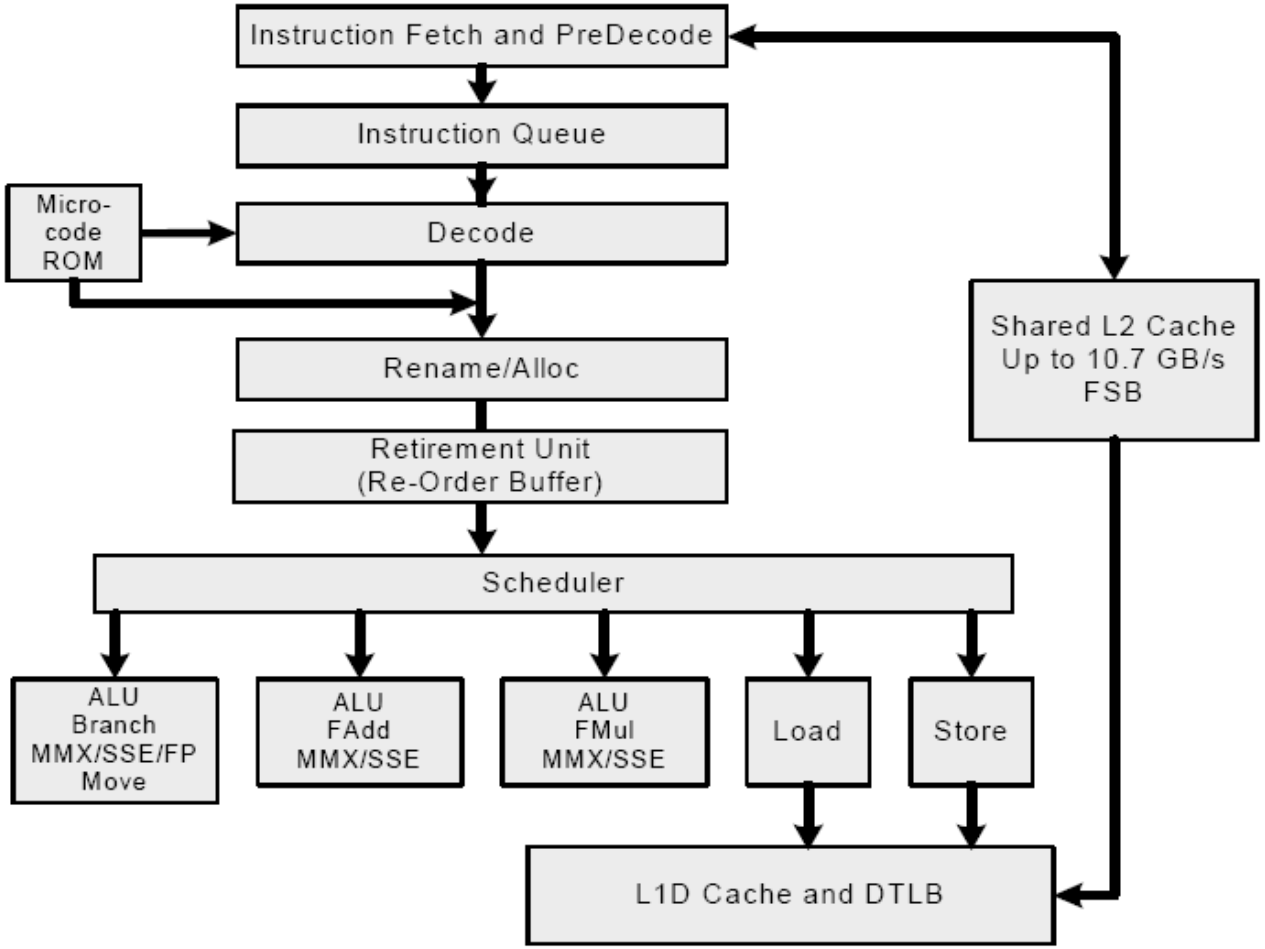
- Pentium 4 Extreme SMT yields 1.01 speedup for SPECint_rate benchmark and 1.07 for SPECfp_rate
  - Pentium 4 is dual threaded SMT
  - SPECRate requires that each SPEC benchmark be run against a vendor-selected number of copies of the same benchmark
- Running on Pentium 4 each of 26 SPEC benchmarks paired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

Intel Core 2 Duo / Quad

Pipeline functionality



Intel Wide Dynamic Execution

- Each *processor core fetches, dispatches, executes and retires* up to four instructions per cycle
  - Fourteen-stage efficient pipeline
  - Three arithmetic logical units
  - Four decoders to decode up to five instruction per cycle
  - Macro-fusion and micro-fusion to improve front-end throughput
  - Peak issue rate of dispatching up to six micro-ops per cycle
  - Peak retirement bandwidth of up to 4 micro-ops per cycle
  - Advanced branch prediction
  - Stack pointer tracker to improve efficiency of executing function/procedure entries and exits

Intel Advanced Smart Cache

- ***Delivers higher bandwidth from the second*** level cache to the core
  - Large second level cache up to 4 MB and 16-way associativity
  - Optimized for multicore and single-threaded execution environments
  - 256 bit internal data path to improve bandwidth from L2 to first-level data cache

Intel Smart Memory Access

- Prefetches data from memory in response to data access patterns and reduces cache-miss exposure of out-of-order execution.
  - Hardware prefetchers to reduce effective latency of second-level cache misses
  - Hardware prefetchers to reduce effective latency of first-level data cache misses
  - Memory disambiguation to improve efficiency of speculative execution execution engine

Intel Advanced Digital Media Boost

- Improves most 128-bit SIMD instruction with single-cycle throughput and floating-point operations
  - Single-cycle throughput of most 128-bit SIMD instructions
  - Up to eight floating-point operation per cycle
  - Three issue ports available to dispatching SIMD instructions for execution

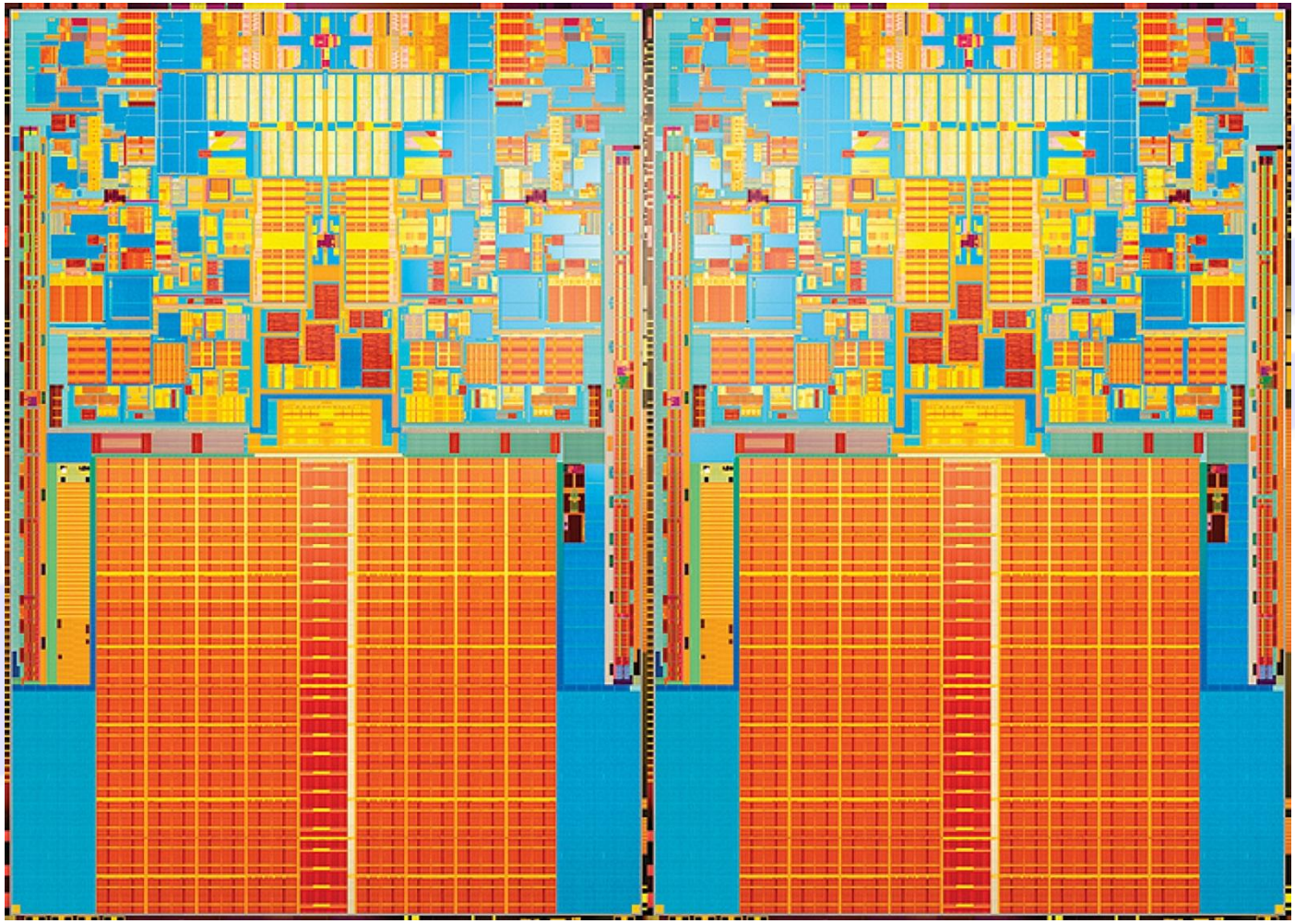
Front end

- Instruction fetch unit prefetches instructions into an instruction queue to maintain steady supply of instruction to the decode units
- Four-wide decode unit can decode 4 instructions per cycle or 5 instructions per cycle with Macrofusion
  - Macrofusion fuses common sequence of two instructions as one decoded instruction (micro-ops) to increase decoding throughput
  - Microfusion fuses common sequence of two micro-ops as one micro-ops to improve retirement throughput
- Instruction queue provides caching of short loops to improve efficiency
- Stack pointer tracker improves efficiency of executing procedure/function entries and exits

Execution Cores

- The execution core of the Intel Core microarchitecture is superscalar and can process instructions out of order to increase the overall rate of instructions executed per cycle (IPC)
  - Up to six micro-ops can be dispatched to execute per cycle
  - Up to four instructions can be retired per cycle
  - Three full arithmetic logical units
  - SIMD instructions can be dispatched through three issue ports
  - Most SIMD instructions have 1-cycle throughput (including 128-bit SIMD instructions)
  - Up to eight floating-point operation per cycle
  - Many long-latency computation operation are pipelined in hardware to increase overall throughput
  - Reduced exposure to data access delays using Intel Smart Memory Access

Penryn

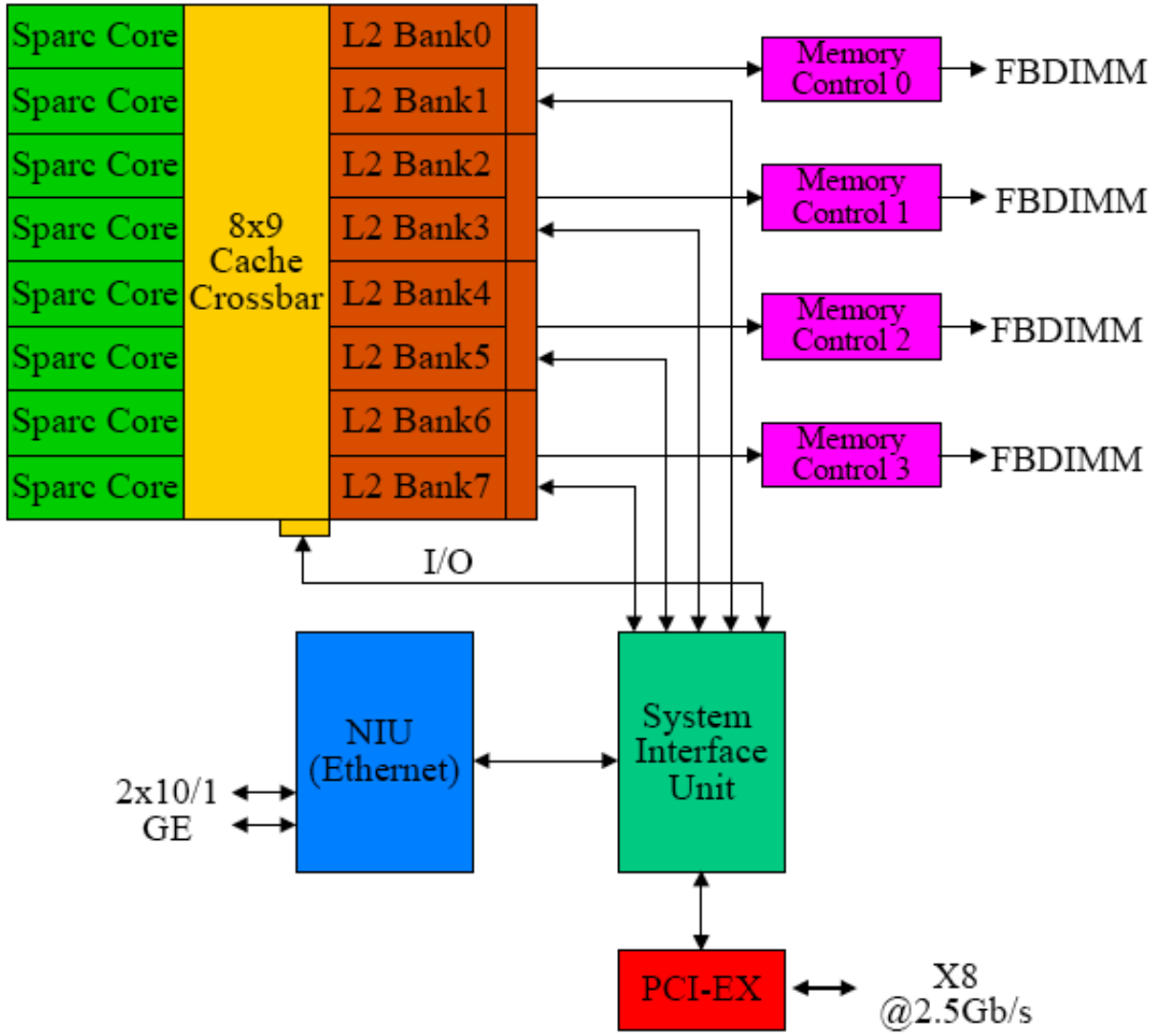


Penryn

- Penryn adds to the Core Micro-architecture
  - A radix-16 divider, faster OS primitives further increases the performance of Intel Wide Dynamic Execution
  - Improves Intel Advanced Smart Cache with up to 50% larger level-two cache (4 MB vs 6 MB) and up to 50% increase in way-set associativity (16 to 24 ways)
  - A 128-bit shuffler engine significantly improves the performance of Intel Advanced Digital Media Boost and SSE4
  - 45 nm vs 65 nm

Sun Niagara T2

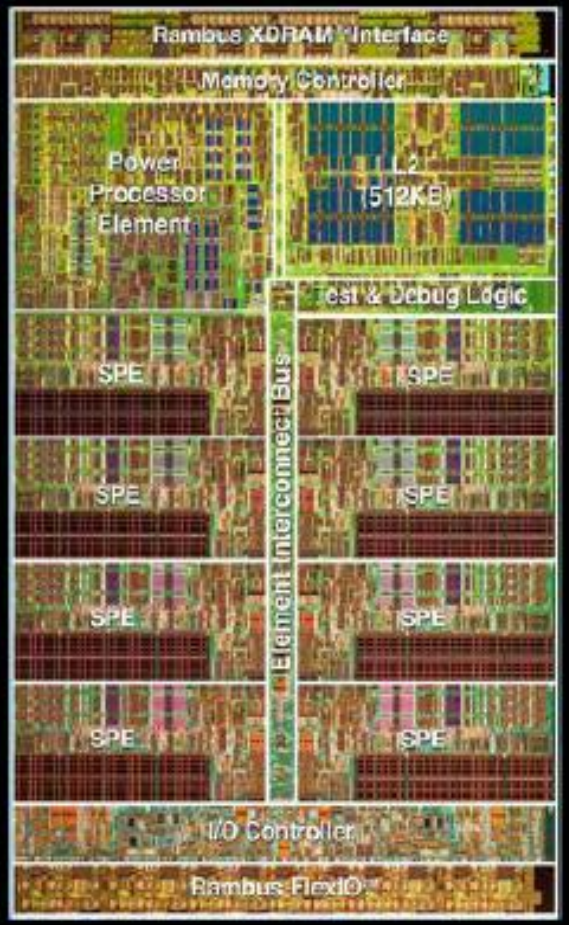
- 8 cores x 8 threads
- In order
- Fine-grained MT



CELL IBM Processor

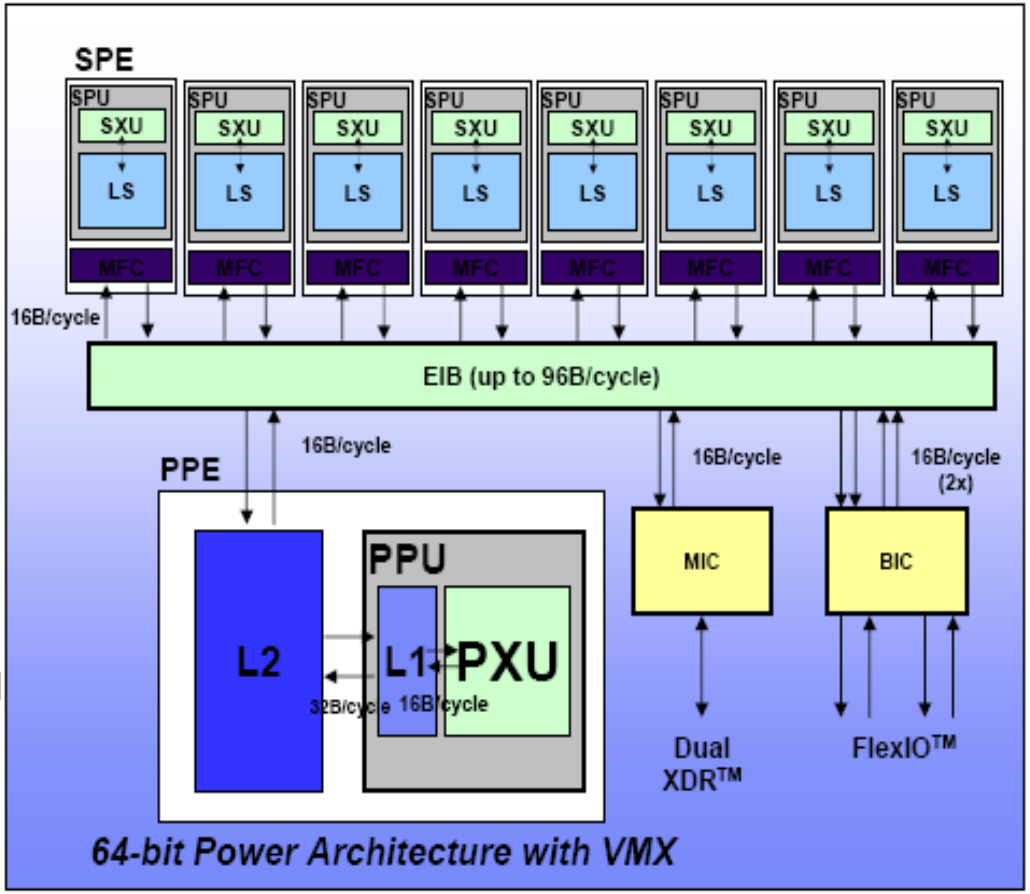
Highlights (3.2 GHz)

- 241M transistors
- 235mm2
- 9 cores, 10 threads
- >200 GFlops (SP)
- >20 GFlops (DP)
- Up to 25 GB/s memory B/W
- Up to 75 GB/s I/O B/W
- >300 GB/s EIB
- Top frequency >4GHz
  (observed in lab)



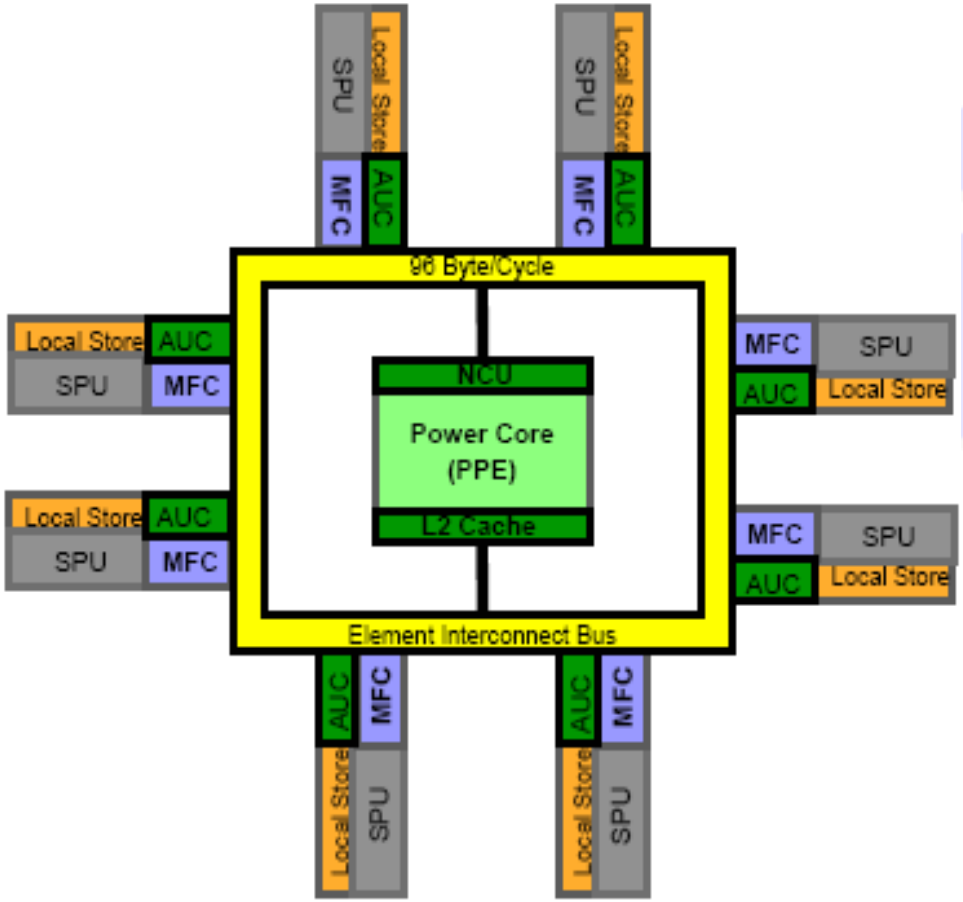
Characteristics

- Multicore with heterogeneous architecture
  - PowerPC processor for control tasks
  - 8 Synergistic Processor Elements for data-intensive processing:
    - Coprocessors with 256 KB dedicated



Synergistic Processor Element

- Power processor element:
  - General purpose, 64 bit RISC processor
  - Two-way multithreaded
  - L1: 32 KB I, 32 KB D
  - L2 512 KB
- Element Interconnect bus
  - 4 rings of 16 data bytes that support simultaneous transfers
- Broadband interface controller
  - To connect to external devices



Intel Nehalem

- Review entire semester by looking at most recent microprocessor from Intel
- Nehalem is code name for microarchitecture at heart of Core i7 and Xeon 5500 series server chips
- First released at end of 2008

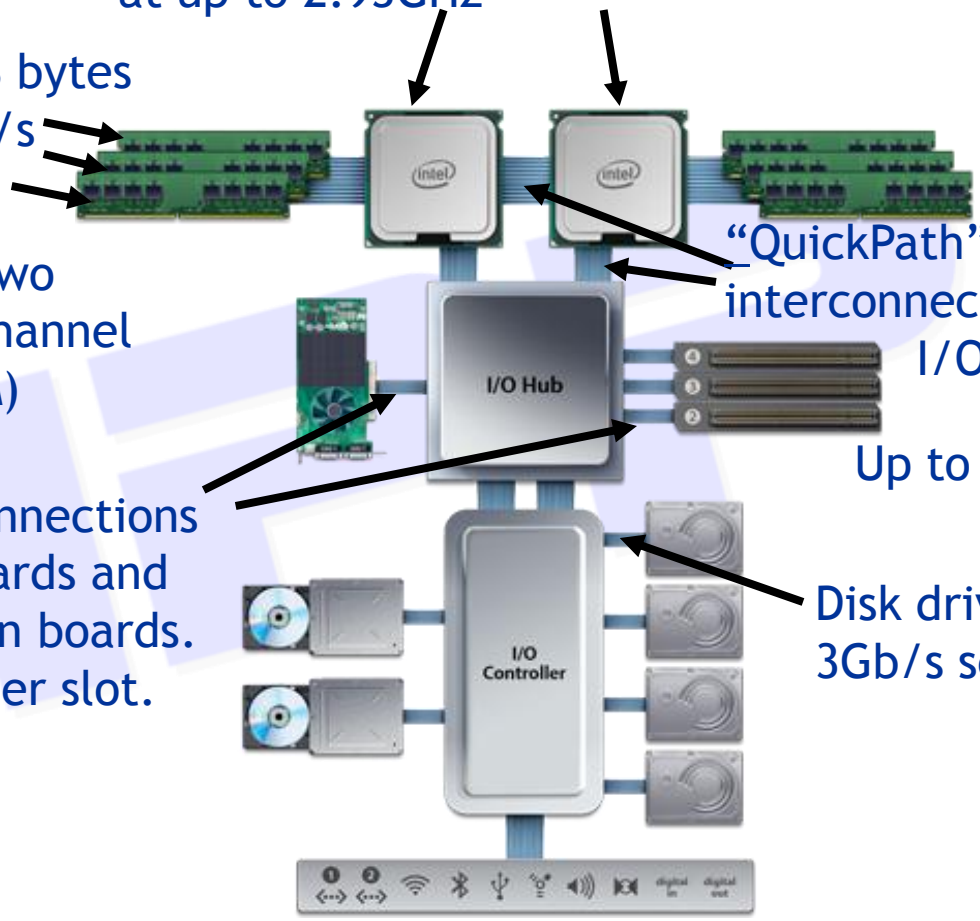
- Figures/Info from Intel, David Kanter at Real World Technologies.

Nehalem System Example: Apple Mac Pro Desktop 2009

Two Nehalem Chips ("Sockets"), each containing four processors ("cores") running at up to 2.93GHz

Each chip has three DRAM channels attached, each 8 bytes wide at 1.066Gb/s (3*8.5GB/s).

Can have up to two DIMMs on each channel (up to 4GB/DIMM)



"QuickPath" point-point system interconnect between CPUs and I/O.

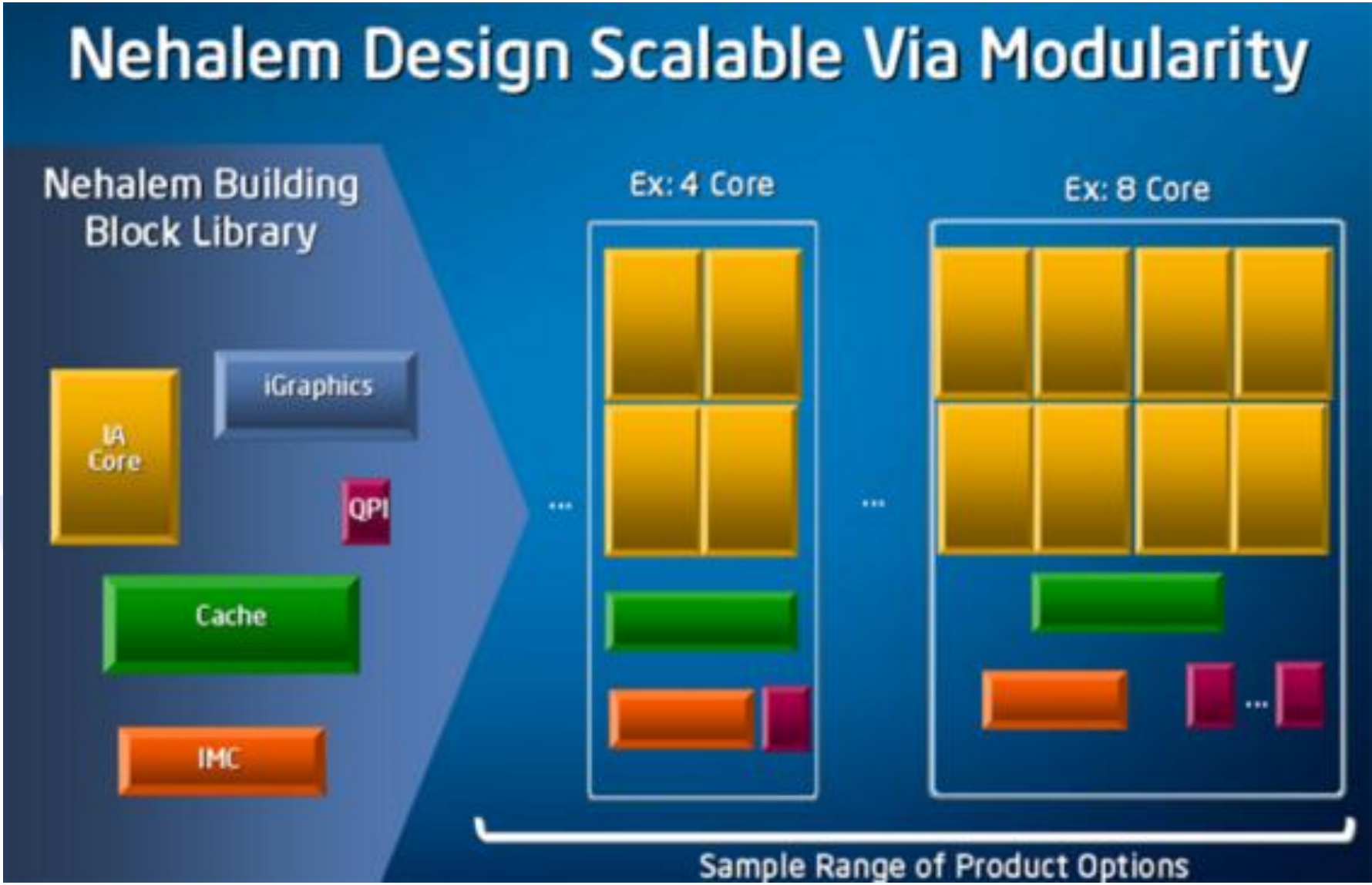
Up to 25.6 GB/s per link.

PCI Express connections for Graphics cards and other extension boards. Up to 8 GB/s per slot.

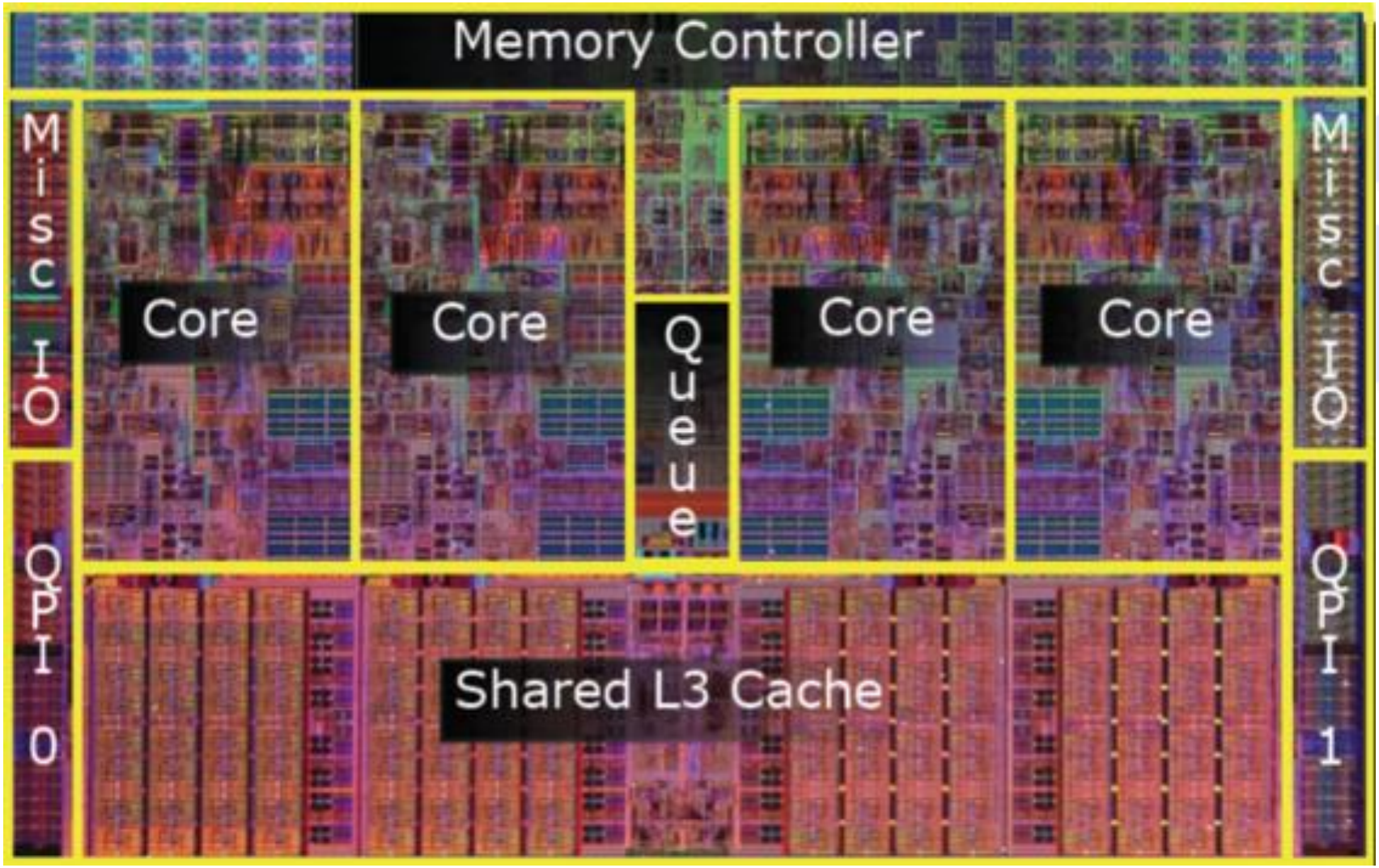
Disk drives attached with 3Gb/s serial ATA link

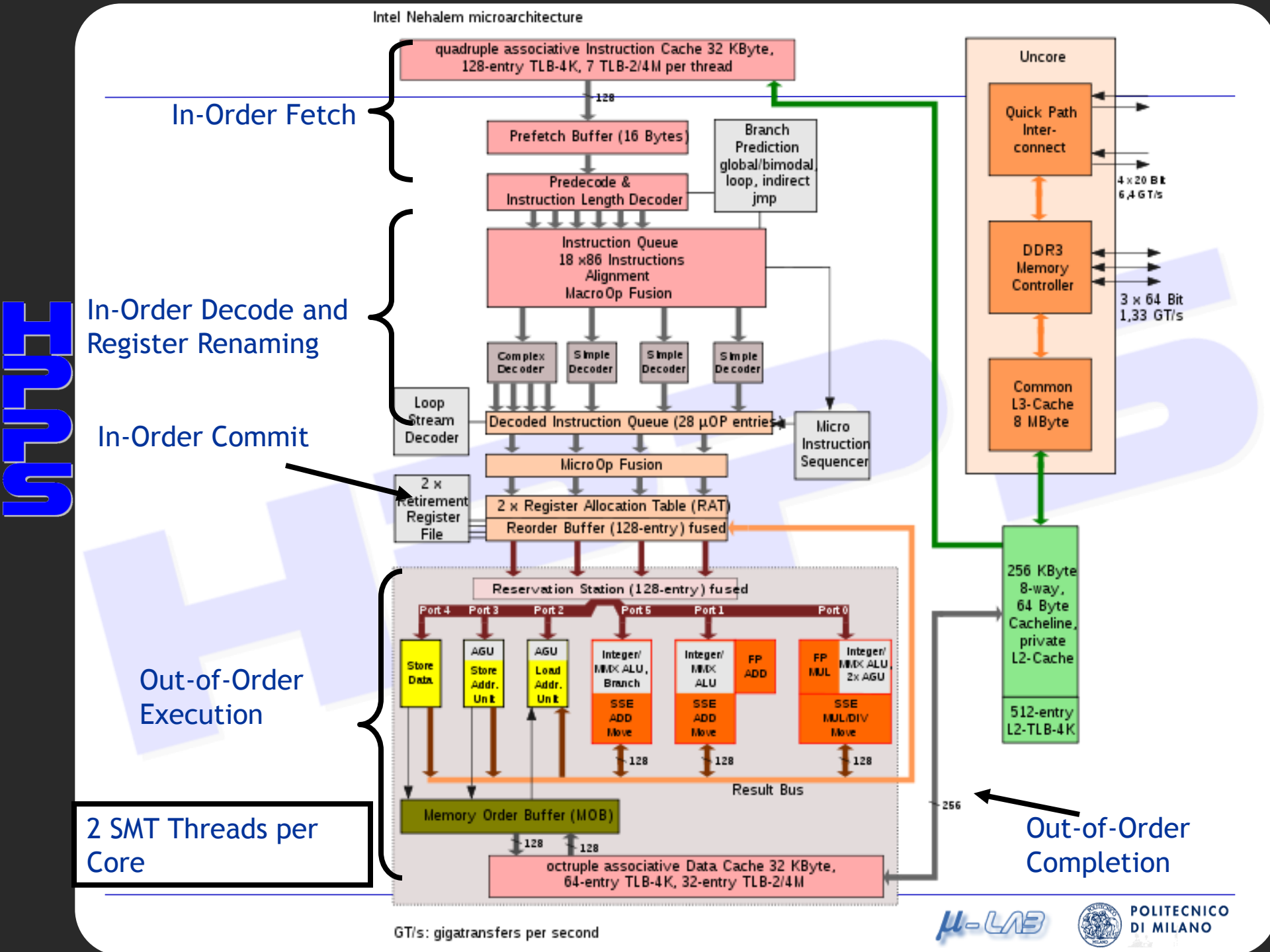
Slower peripherals (Ethernet, USB, Firewire, WiFi, Bluetooth, Audio)

Building Blocks to support “Family” of processors

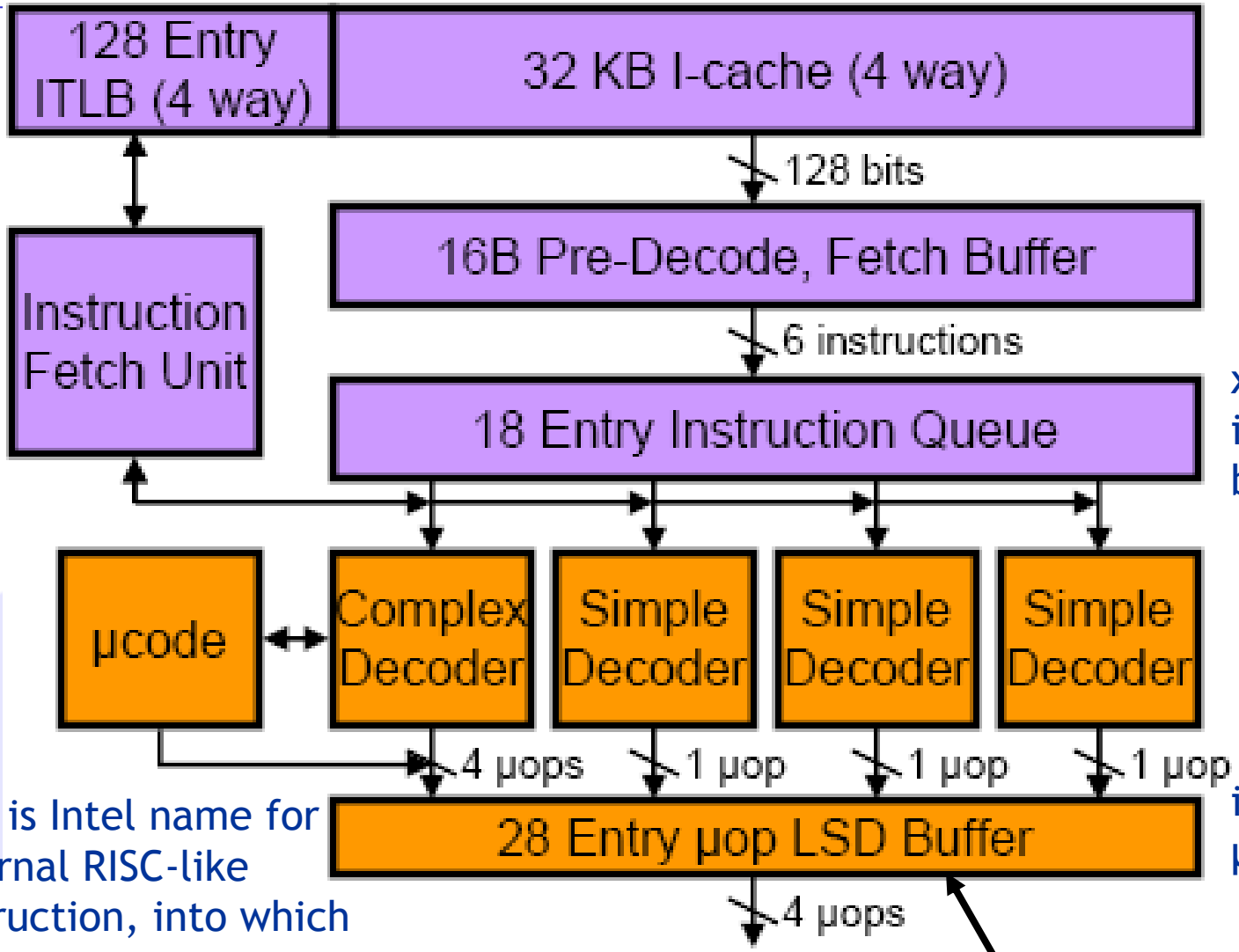


Nehalem Die Photo




Intel Nehalem microarchitecture
In-Order Fetch
In-Order Decode and Register Renaming
In-Order Commit
Out-of-Order Execution
2 SMT Threads per Core
Out-of-Order Completion
quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
FP ADD
SSE ADD Move
FP MUL
Integer/ MMX ALU 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octruple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K
256
GT/s: gigatransfers per second
POLITECNICO DI MILANO

Front-End Instruction Fetch & Decode



µOP is Intel name for internal RISC-like instruction, into which x86 instructions are translated

Loop Stream Detector (can run short loops out of the buffer)

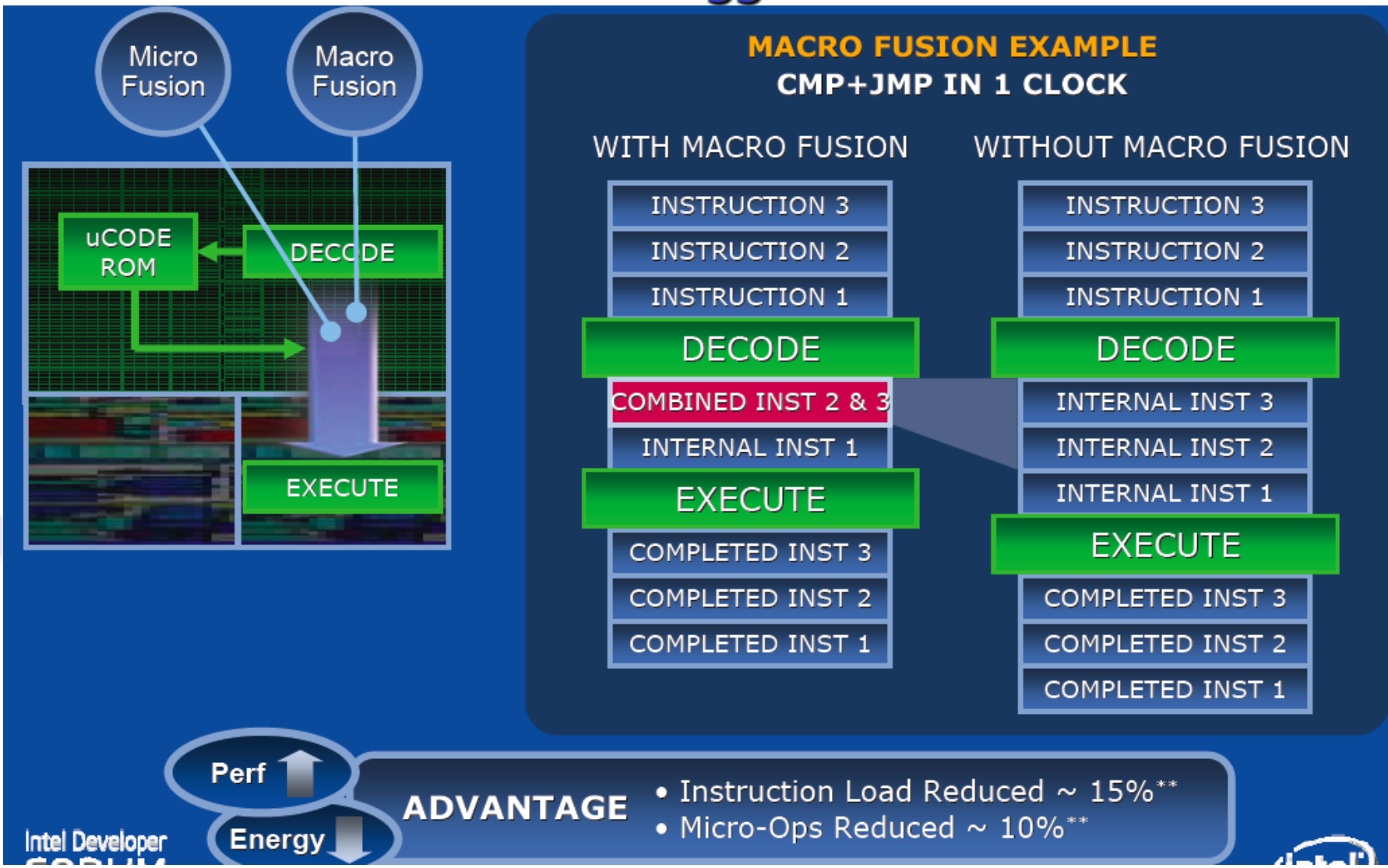
Branch Prediction

- Part of instruction fetch unit
- Several different types of branch predictor
  - Details not public
- Two-level BTB
- Loop count predictor
  - How many backwards taken branches before loop exit
  - (Also predictor for length of microcode loops, e.g., string move)
- Return Stack Buffer
  - Holds subroutine targets
  - Renames the stack buffer so that it is repaired after mispredicted returns
  - Separate return stack buffer for each SMT thread

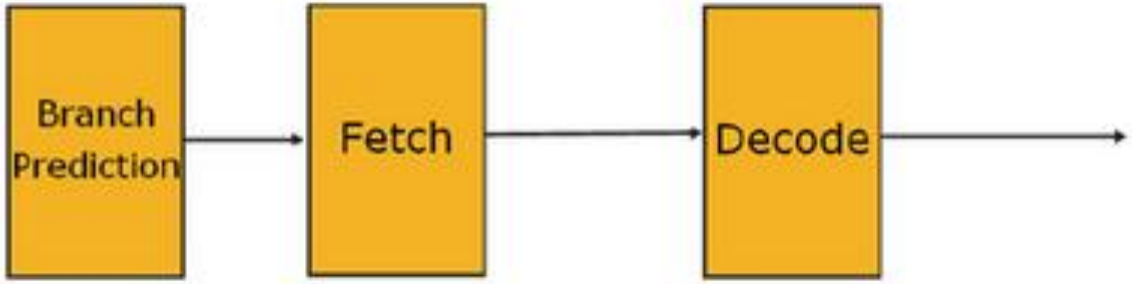
x86 Decoding

- Translate up to 4 x86 instructions into uOPS each cycle
- Only first x86 instruction in group can be complex (maps to 1-4 uOPS), rest must be simple (map to one uOP)
- Even more complex instructions, jump into microcode engine which spits out stream of uOPS

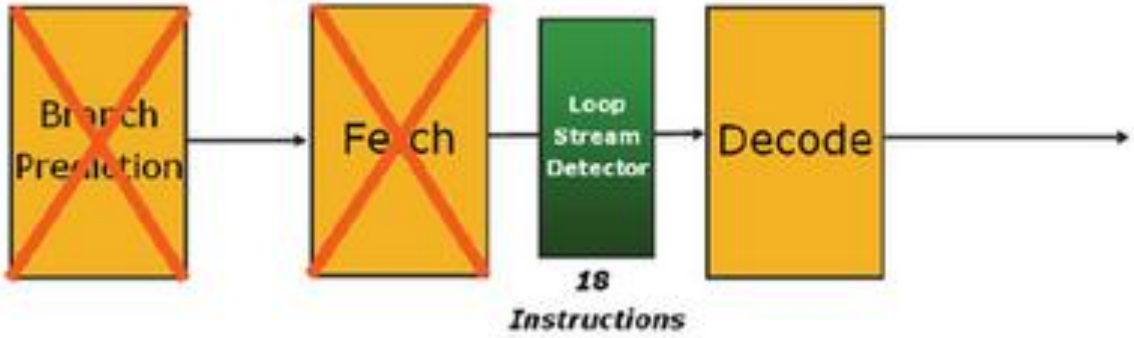
Split x86 in small uOPs, then fuse back into bigger units



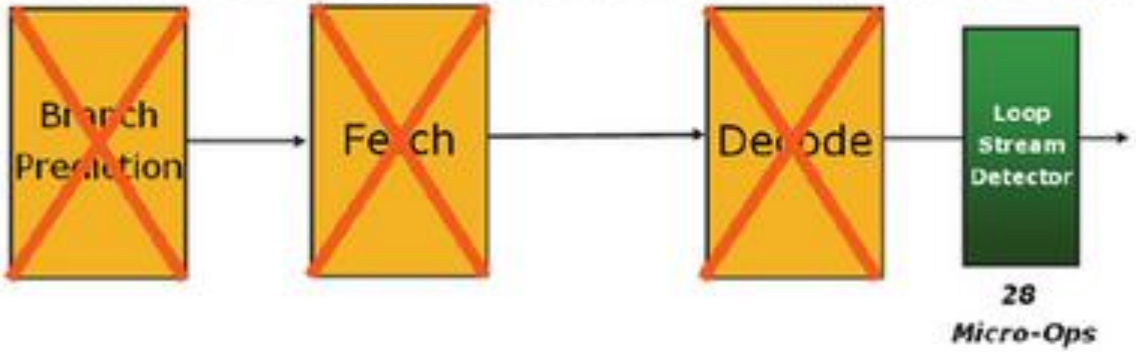
Loop Stream Detectors save Power



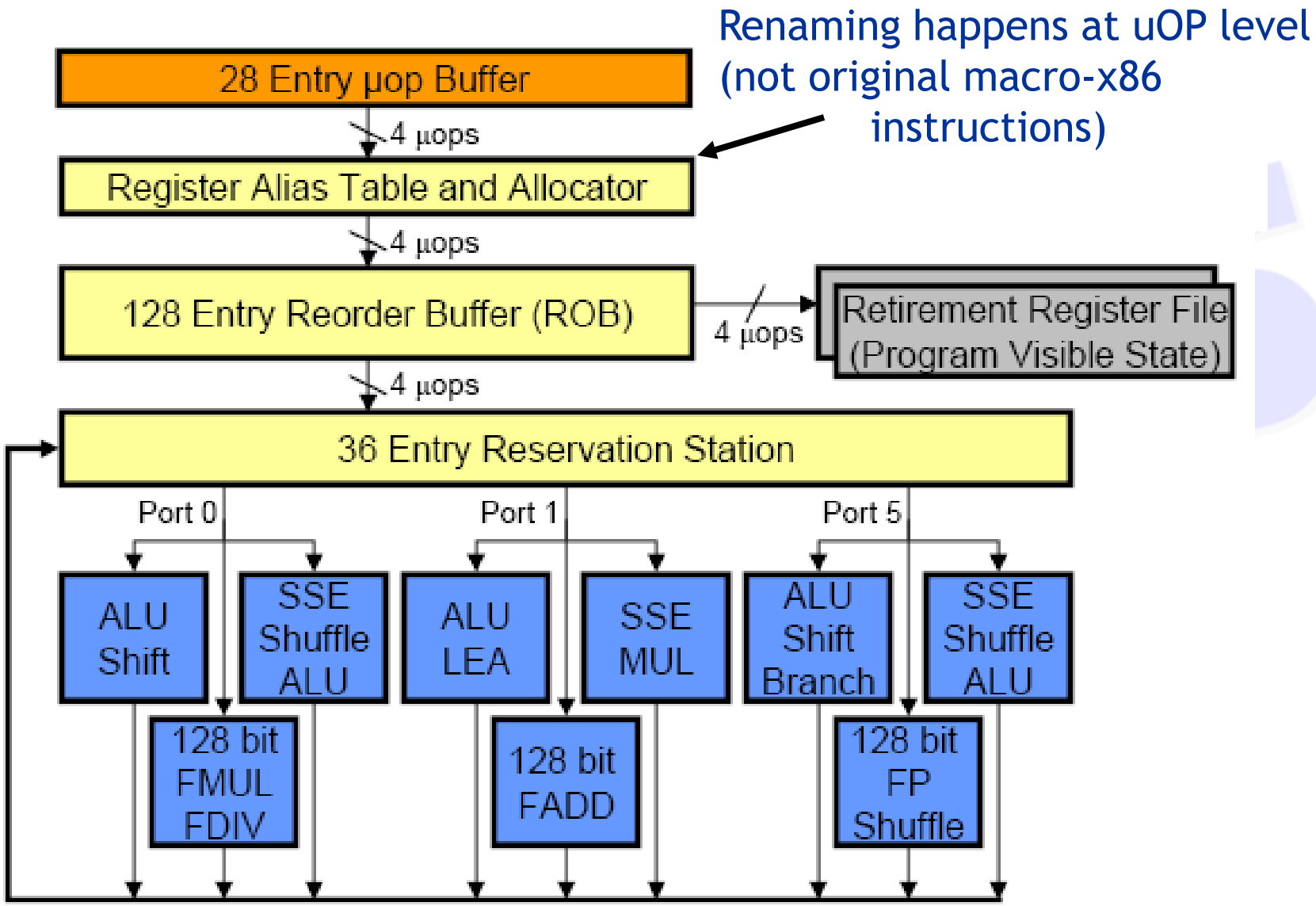
Intel® Core™2 Loop Stream Detector



Intel Core Microarchitecture (Nehalem) Loop Stream Detector



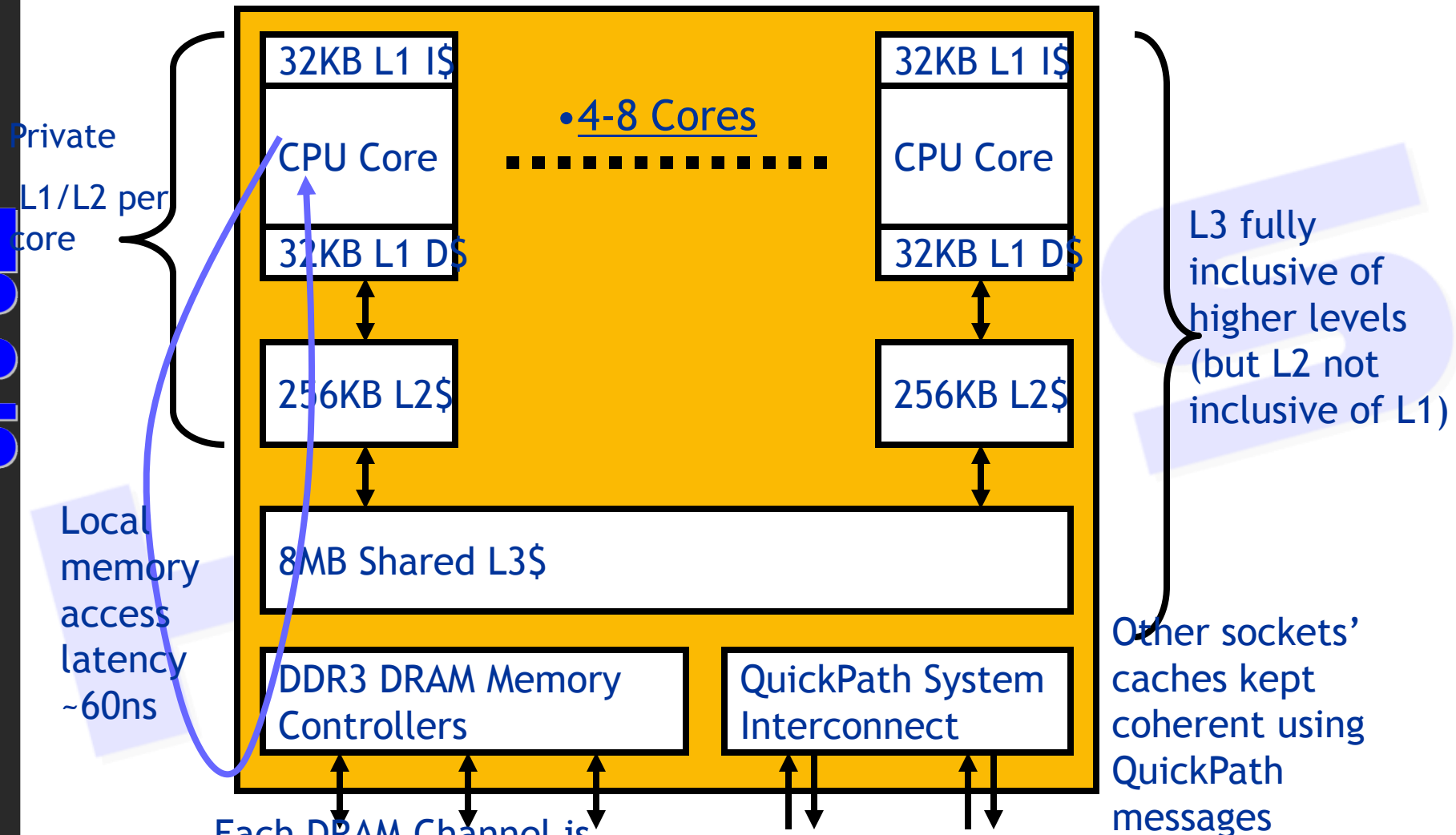
Out-of-Order Execution Engine



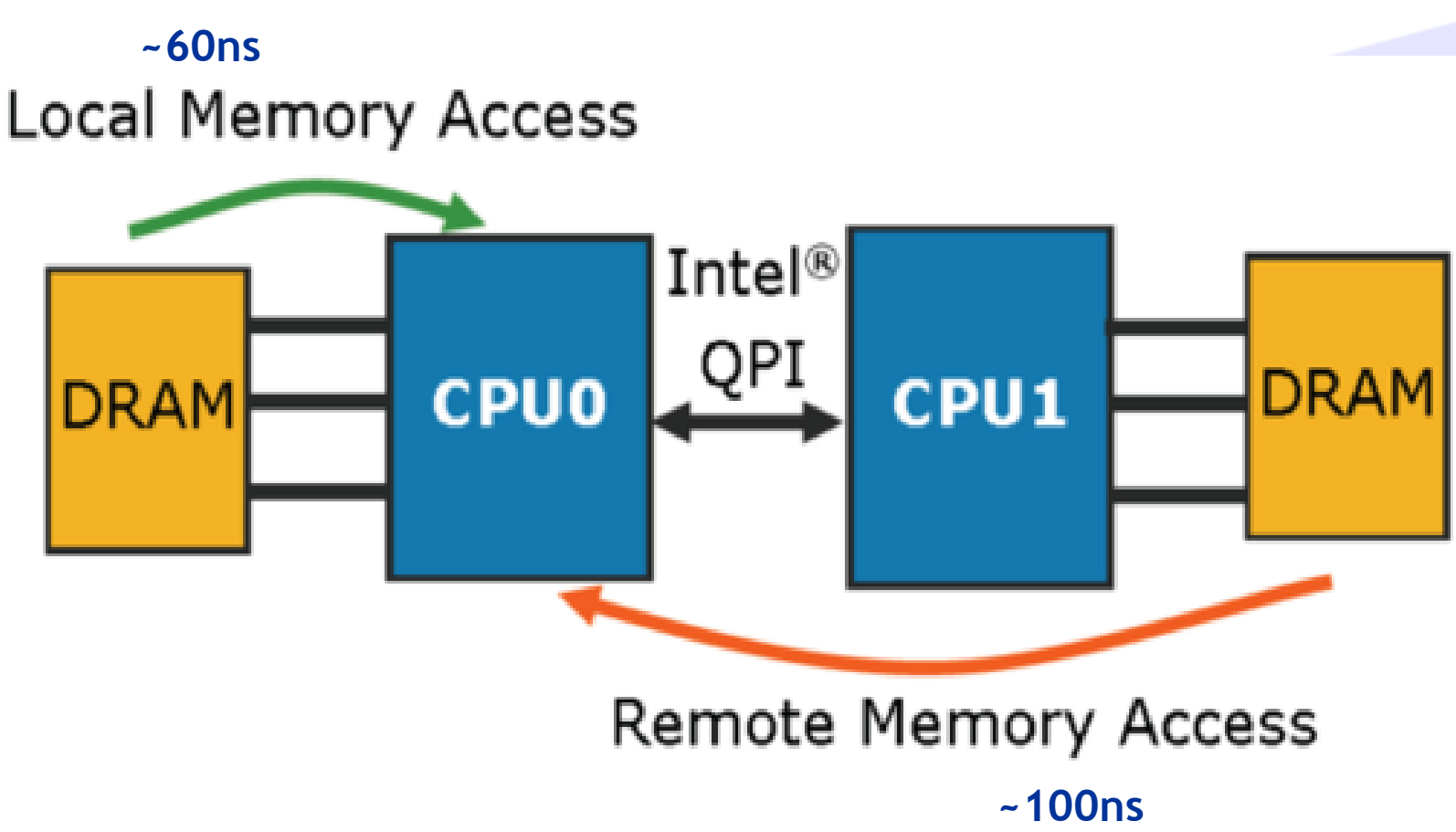
SMT effects in OoO Execution Core

- Reorder buffer (remembers program order and exception status for in-order commit) has 128 entries divided statically and equally between both SMT threads
- Reservation stations (instructions waiting for operands for execution) have 36 entries competitively shared by threads

Nehalem Memory Hierarchy Overview



All Sockets can Access all Data



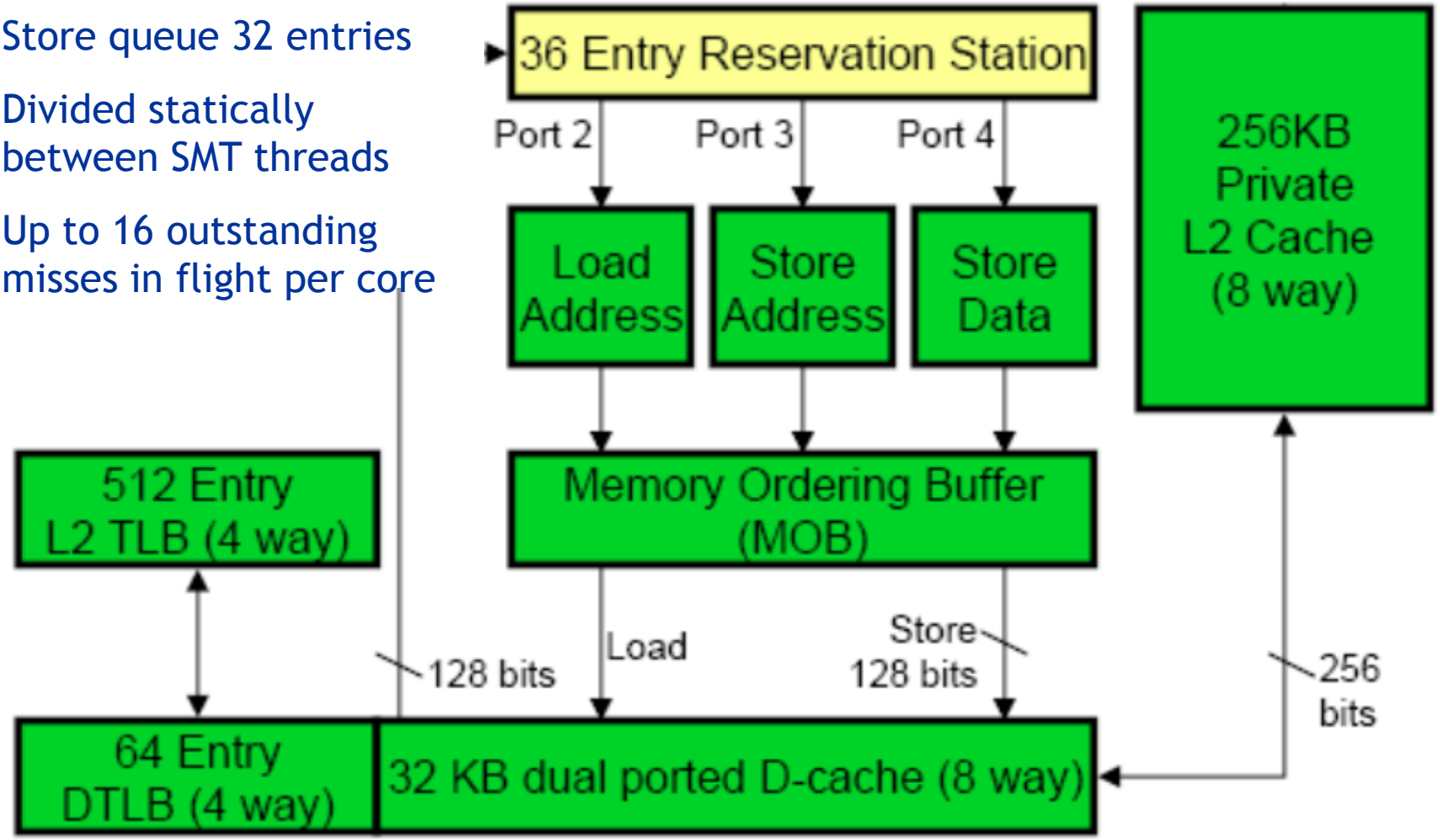
Core's Private Memory System

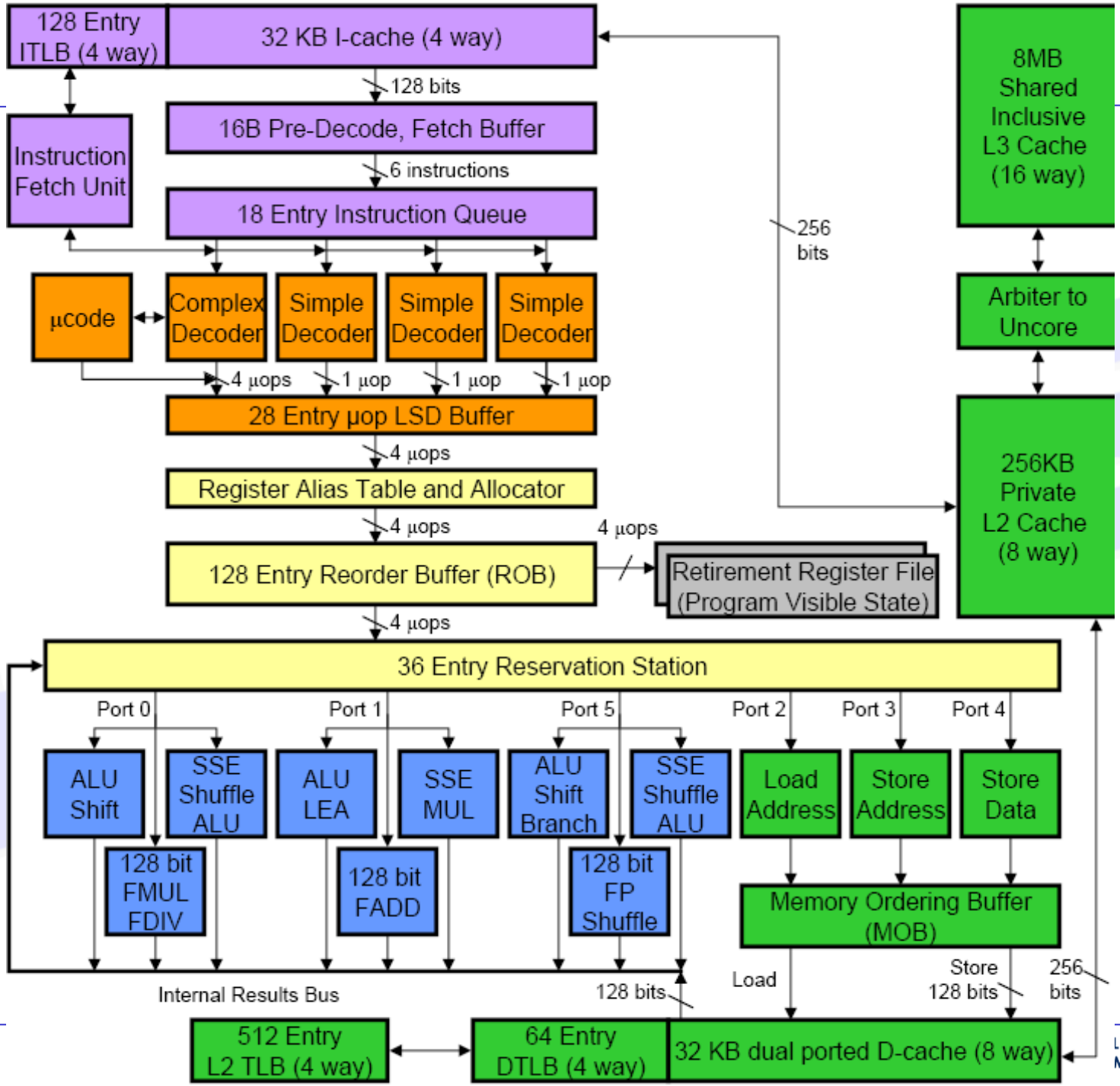
Load queue 48 entries

Store queue 32 entries

Divided statically between SMT threads

Up to 16 outstanding misses in flight per core




128 Entry ITLB (4 way)
32 KB I-cache (4 way)
128 bits
16B Pre-Decode, Fetch Buffer
Instruction Fetch Unit
6 instructions
18 Entry Instruction Queue
256 bits
8MB Shared Inclusive L3 Cache (16 way)
Arbiter to Uncore
μcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 μops
1 μop
1 μop
1 μop
28 Entry μop LSD Buffer
256KB Private L2 Cache (8 way)
4 μops
Register Alias Table and Allocator
4 μops
4 μops
128 Entry Reorder Buffer (ROB)
Retirement Register File (Program Visible State)
4 μops
36 Entry Reservation Station
Port 0
Port 1
Port 5
Port 2
Port 3
Port 4
ALU Shift
SSE Shuffle ALU
ALU LEA
SSE MUL
ALU Shift Branch
SSE Shuffle ALU
Load Address
Store Address
Store Data
128 bit FMUL FDIV
128 bit FADD
128 bit FP Shuffle
Memory Ordering Buffer (MOB)
Load
Store
128 bits
256 bits
Internal Results Bus
128 bits
512 Entry L2 TLB (4 way)
64 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)

Cache Hierarchy Latencies

- L1 32KB 8-way, latency 4 cycles
- L2 256KB 8-way, latency <12 cycles
- L3 8MB, 16-way, latency 30-40 cycles
- DRAM, latency ~180-200 cycles

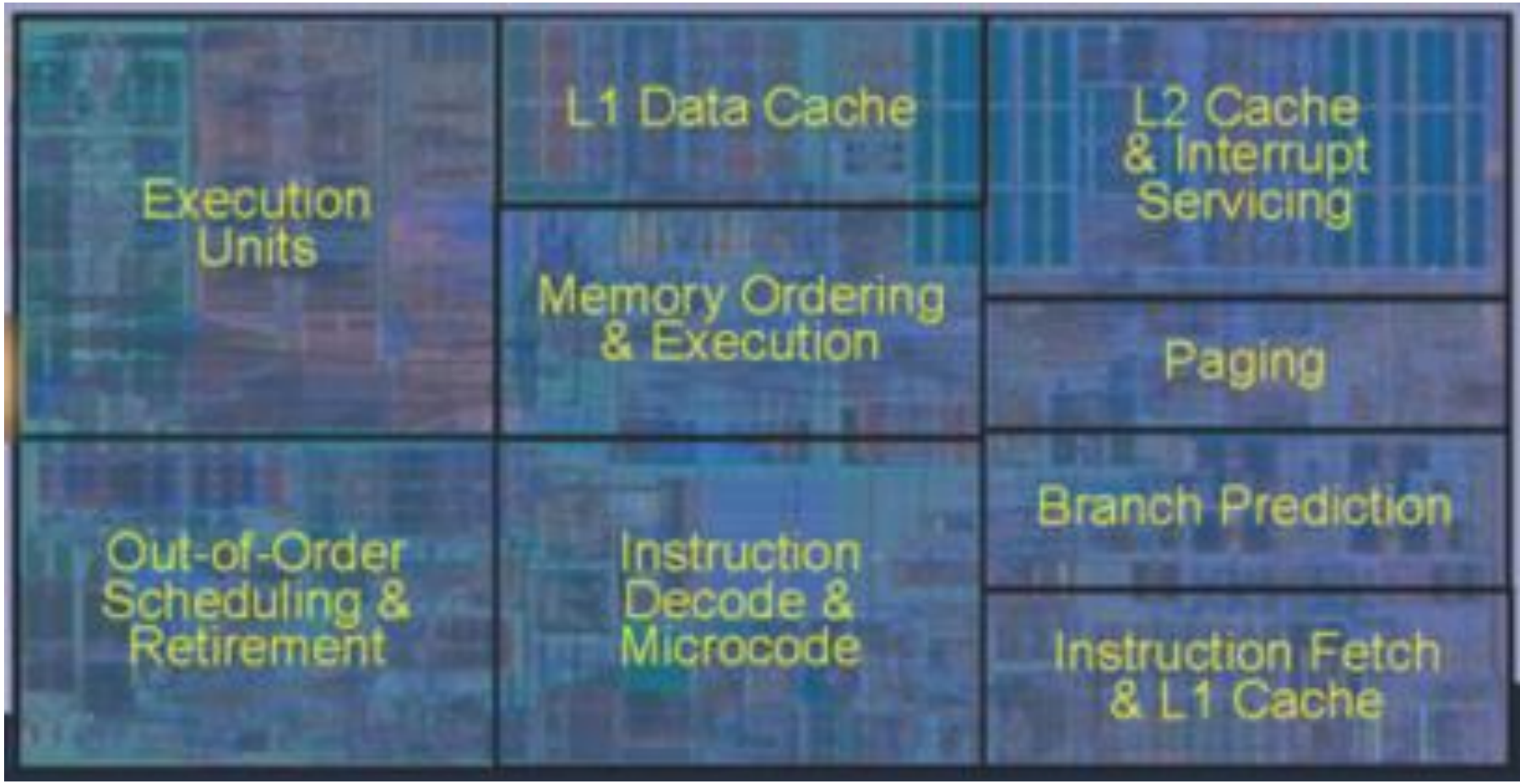
Nehalem Virtual Memory Details

- Implements 48-bit virtual address space, 40-bit physical address space
- Two-level TLB
- I-TLB (L1) has shared 128 entries 4-way associative for 4KB pages, plus 7 dedicated fully-associative entries per SMT thread for large page (2/4MB) entries
- D-TLB (L1) has 64 entries for 4KB pages and 32 entries for 2/4MB pages, both 4-way associative, dynamically shared between SMT threads
- Unified L2 TLB has 512 entries for 4KB pages only, also 4-way associative
- Additional support for system-level virtual machines

Virtualization Support

- TLB entries tagged with virtual machine and address space ID
  - No need to flush on context switches between VMs
- Hardware page table walker can walk guest-physical to host-physical mapping tables
  - Fewer traps to hypervisor

Core Area Breakdown



(Nehalem) Turbo Mode

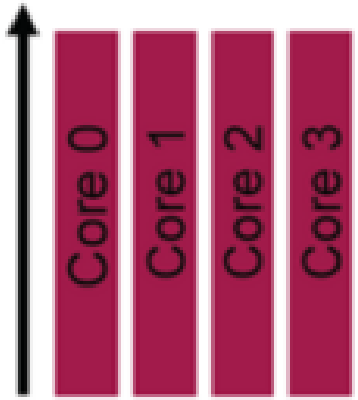
Power Gating

Zero power for inactive cores

Turbo Mode

In response to workload adds additional performance bins within headroom

No Turbo



Workload Lightly Threaded or < TDP