ECE/CS 757: Advanced Computer Architecture II

Instructor:Mikko H Lipasti

Spring 2017

University of Wisconsin-Madison

Lecture notes based on slides created by John Shen, Mark Hill, David Wood, Guri Sohi, Jim Smith, Natalie Enright Jerger, Michel Dubois, Murali Annavaram, Per Stenström and probably others

Lecture 3 Outline

- Multithreaded processors
- Multicore processors

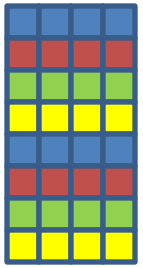
Multithreaded Cores

- Basic idea:
  - CPU resources are expensive and should not be idle
- 1960's: Virtual memory and multiprogramming
  - Virtual memory/multiprogramming invented to tolerate latency to secondary storage (disk/tape/etc.)
  - Processor-disk speed mismatch:
    - microseconds to tens of milliseconds (1:10000 or more)
  - OS context switch used to bring in other useful work while waiting for page fault or explicit read/write
  - Cost of context switch must be much less than I/O latency (easy)

Multithreaded Cores

- 1990's: Memory wall and multithreading
  - Processor-DRAM speed mismatch:
    - nanosecond to fractions of a microsecond (1:500)
  - H/W task switch used to bring in other useful work while waiting for cache miss
  - Cost of context switch must be much less than cache miss latency
- Very attractive for applications with abundant thread-level parallelism
  - Commercial multi-user workloads

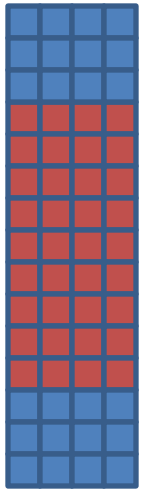
Approaches to Multithreading



- Fine-grain multithreading
  - Switch contexts at fixed fine-grain interval (e.g. every cycle)
  - Need enough thread contexts to cover stalls
  - Example: Tera MTA, 128 contexts, no data caches
- Benefits:
  - Conceptually simple, high throughput, deterministic behavior
- Drawback:
  - Very poor single-thread performance

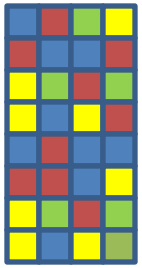
Approaches to Multithreading

- Coarse-grain multithreading
  - Switch contexts on long-latency events (e.g. cache misses)
  - Need a handful of contexts (2-4) for most benefit
- Example: IBM RS64-IV (Northstar), 2 contexts
- Benefits:
  - Simple, improved throughput (~30%), low cost
  - Thread priorities mostly avoid single-thread slowdown
- Drawback:
  - Nondeterministic, conflicts in shared caches



Approaches to Multithreading

- Simultaneous multithreading
  - Multiple concurrent active threads (no notion of thread switching)
  - Need a handful of contexts for most benefit (2-8)
- Examples: Intel P4, Intel Nehalem, IBM Power 5/6/7/8, Alpha EV8/21464, AMD Zen
- Benefits:
  - Natural fit for OOO superscalar
  - Improved throughput
  - Low incremental cost
- Drawbacks:
  - Additional complexity over OOO superscalar
  - Cache conflicts



Approaches to Multithreading

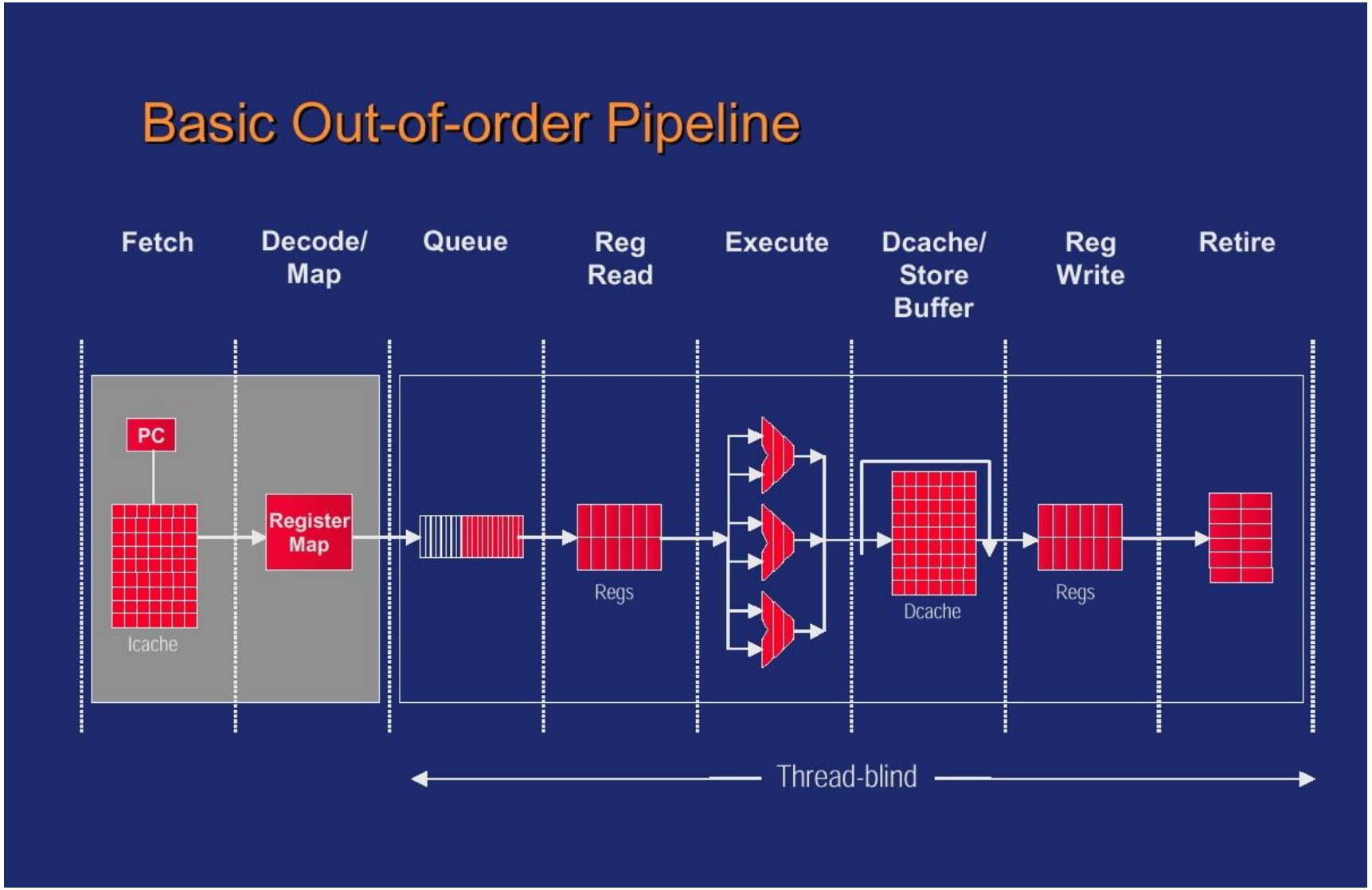
- Chip Multithreading (CMT)
  - Similar to CMP
- Share something in the core:
  - Expensive resource, e.g. floating-point unit (FPU)
  - Also share L2, system interconnect (memory and I/O bus)
- Example: Sun Niagara, 8 cores per die, one FPU
- Benefits:
  - Same as CMP
  - Further: amortize cost of expensive resource over multiple cores
- Drawbacks:
  - Shared resource may become bottleneck
  - 2nd generation (Niagara 2) does **not** share FPU

Multithreaded/Multicore Processors

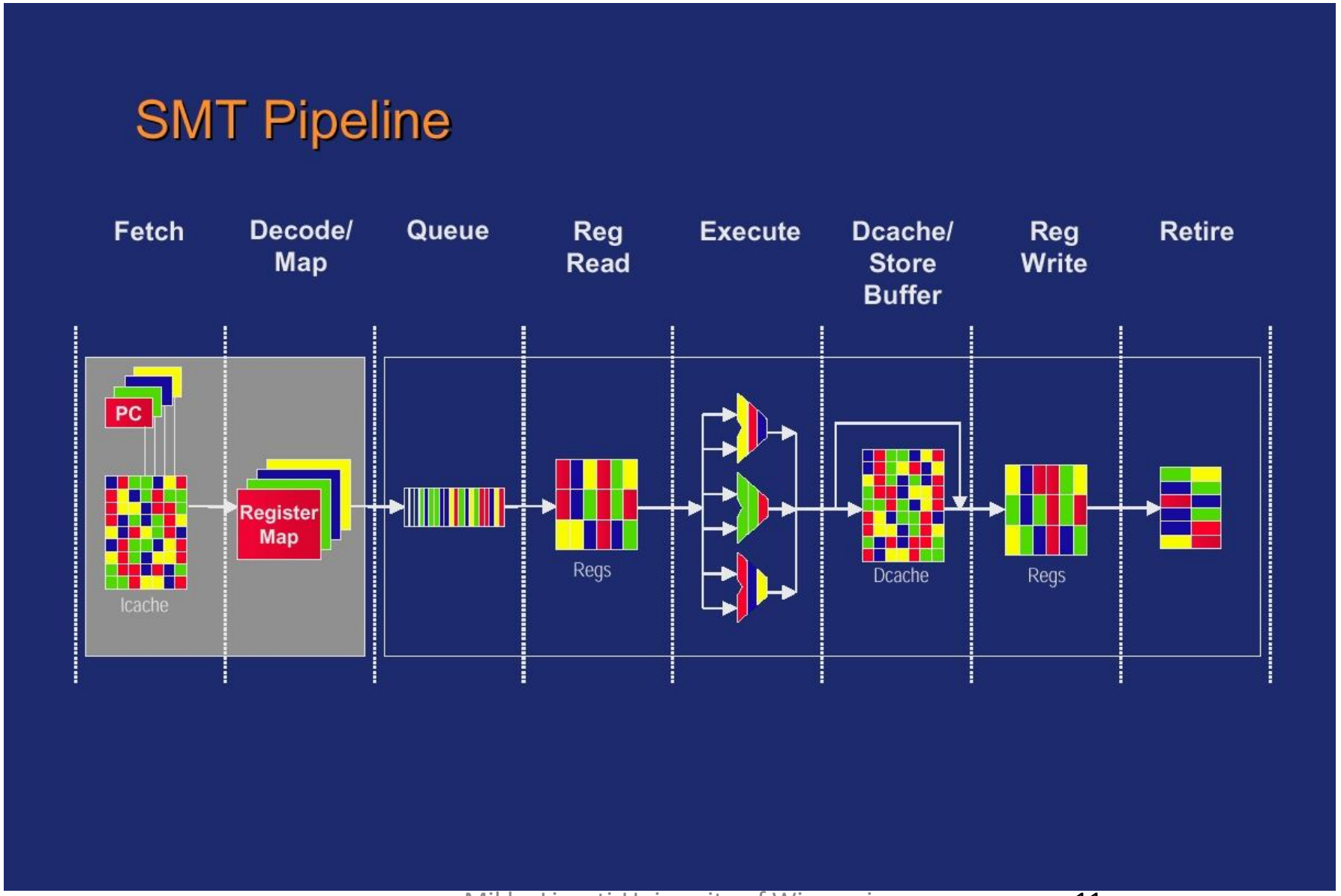
MT Approach	Resources shared between threads	Context Switch Mechanism
None	Everything	Explicit operating system context switch
Fine-grained	Everything but register file and control logic/state	Switch every cycle
Coarse-grained	Everything but I-fetch buffers, register file and con trol logic/state	Switch on pipeline stall
SMT	Everything but instruction fetch buffers, return address stack, architected register file, control logic/state, reorder buffer, store queue, etc.	All contexts concurrently active; no switching
CMT	Various core components (e.g. FPU), secondary cache, system interconnect	All contexts concurrently active; no switching
CMP	Secondary cache, system interconnect	All contexts concurrently active; no switching

- Many approaches for executing multiple threads on a single die
  - Mix-and-match: IBM Power8 CMP+SMT

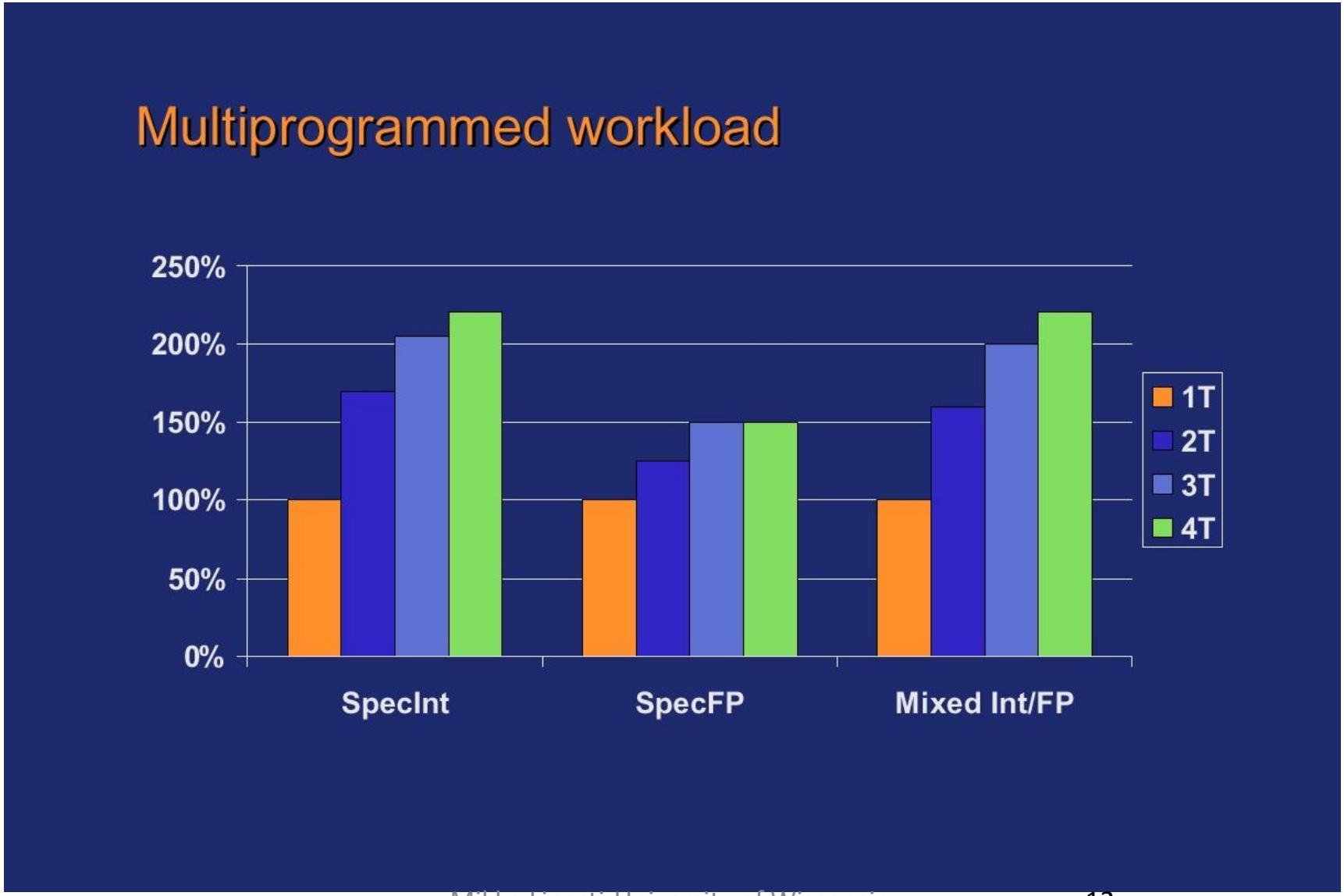
SMT Microarchitecture [Emer,'01]



SMT Microarchitecture [Emer,'01]



SMT Performance [Emer,'01]



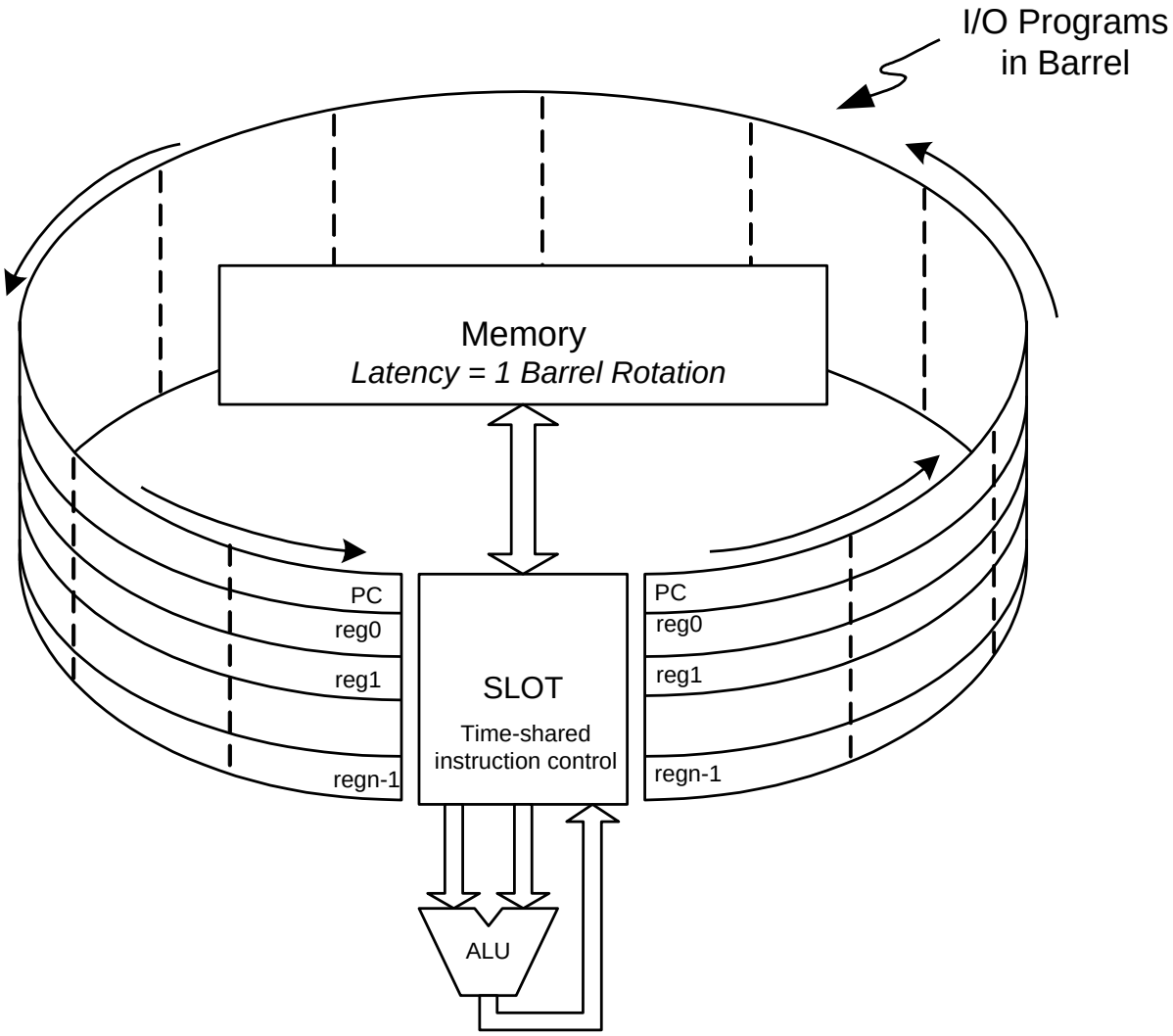
Historical Multithreaded Processors

- CDC6600 PPs
  - I/O processing
- Denelcor HEP
  - General purpose scientific

CDC 6600 Peripheral Processors

- Intended to perform OS and I/O functions
- Used "barrel and slot"
  - register state is arranged around a “barrel”
  - one set of ALU and memory hardware accessed through “slot” in barrel
  - slot in barrel rotates one position each cycle
- Could be used as stand-alone "MP"
- Similar method later used in IBM Channels

CDC 6600 Peripheral Processors



Denelcor HEP

- General purpose scientific computer
- Organized as an MP
  - Up to 16 processors
  - Each processor is multithreaded
  - Up to 128 memory modules
  - Up to 4 I/O cache modules
  - Three-input switches and *chaotic* routing

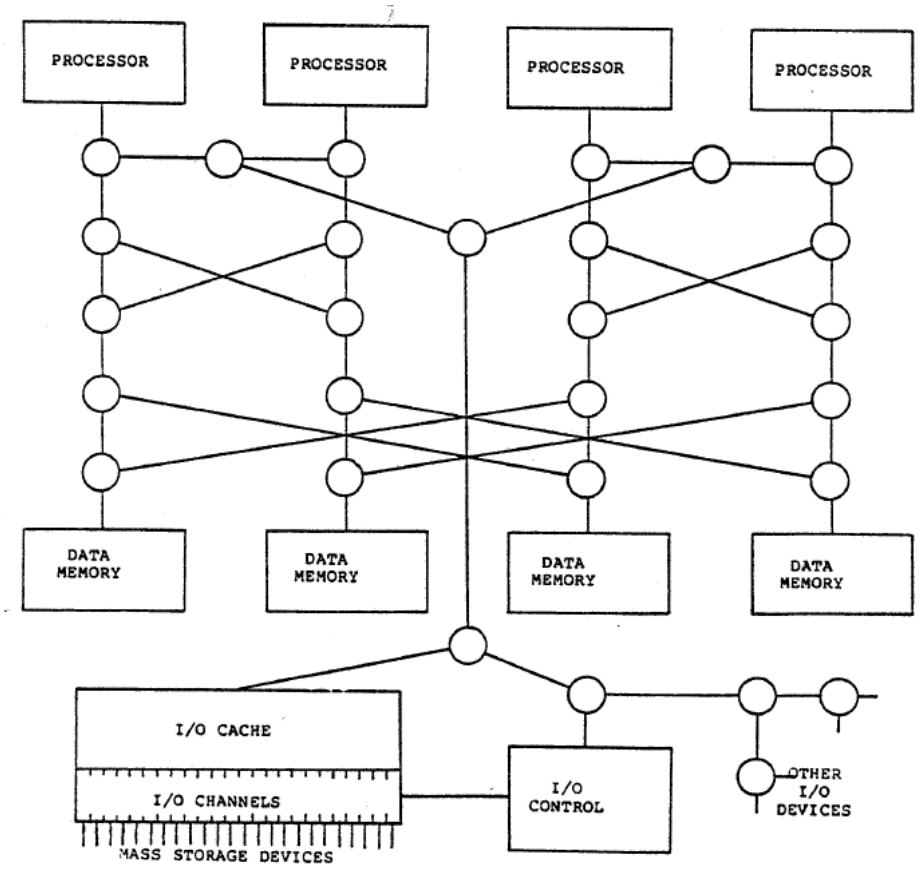


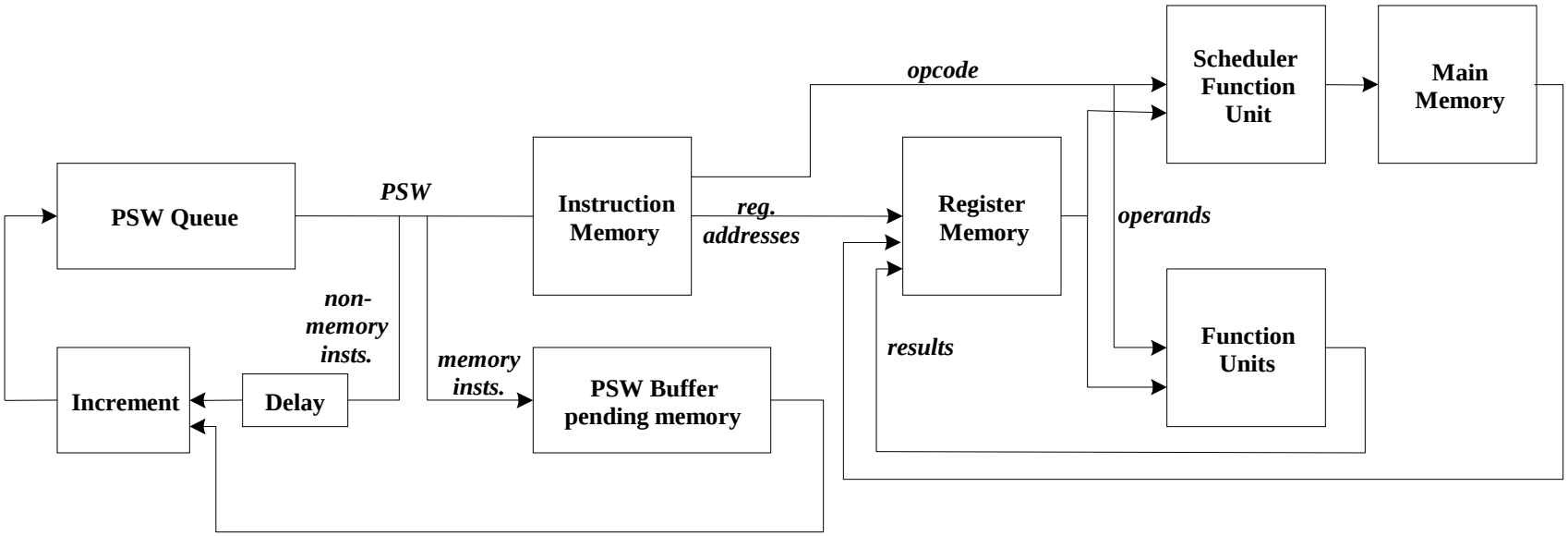
Figure 1. A typical HEP system

HEP Processor Organization

- Multiple contexts (threads) are supported;
  - 120 threads
  - Each with a PSW (program status word)
- PSWs circulate in a control loop
  - control and data loops pipelined 8 deep
  - PSW in control loop can circulate no faster than data in data loop
  - PSW at queue head fetches and starts execution of next instruction
  - No inter-instruction pipeline forwarding or stalls needed
- Clock period: 100 ns
  - 8 PSWs in control loop => 10 MIPS
  - Maximum perf. per thread => 1.25 MIPS

  (They tried to sell this as a supercomputer)

HEP Processor Organization



HEP Processor, contd.

- Address space: 32K to 1Mwords (64 bits)
- 64 bit instructions
- 2048 GP registers + 4096 constants
  - Registers can be shared among threads
- Memory operation
  - Loads and stores performed by scheduler functional unit (SFU)
  - SFU builds network packet and sends it into switch
  - PSW is removed from control loop and placed in SFU queue
  - PSW is placed back into control loop following memory response
- Special operations
  - control instructions to create/terminate threads
  - full/empty bits in memory and registers
    - busy wait on empty/full operands

Switch

- Packet switched, but bufferless
- 3 bi-directional ports per switch
  - Every cycle, take in 3 packets, send out 3 packets
- "Hot Potato" routing
  - Form of adaptive routing
  - Do not enqueue on a port conflict
    - Send anyway on another port and raise priority
  - At top priority (15) traverse a circuit through the net
  - Revisited: C. Fallin, “CHIPPER: …” HPCA 2011

Modern Day Multi-Threading

- Apply to superscalar pipelines
  - More resources to share
- Also one-wide in-order processors
  - Provide high efficiency for throughput-oriented servers
- Start with case study
  - Intel Pentium 4 *Hyperthreading*

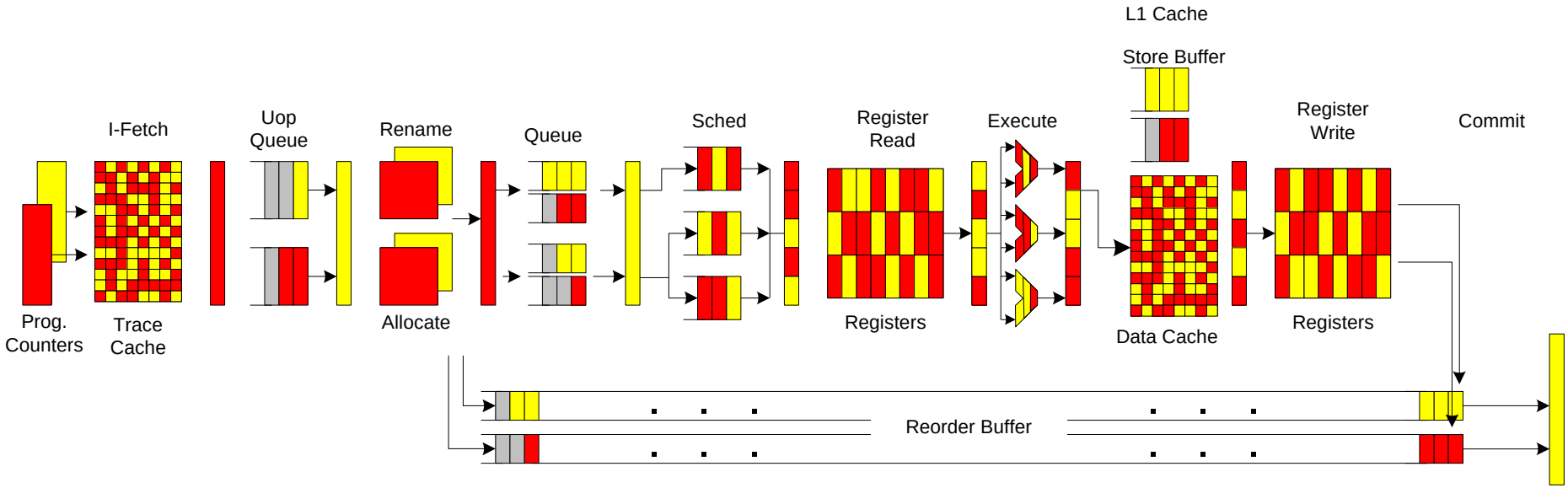
D. Marr et al. Hyper-Threading Technology Architecture and Microarchitecture. Intel Technology Journal, Feb. 2002

Intel Hyperthreading

- Part of Pentium 4 design (Xeon)
- Two threads per processor
- Goals
  - Low cost – less than 5% overhead for replicated state
  - Assure forward progress of both threads
    - Make sure both threads get some buffer resources
    - through partitioning or budgeting
  - Single thread running alone does not suffer slowdown

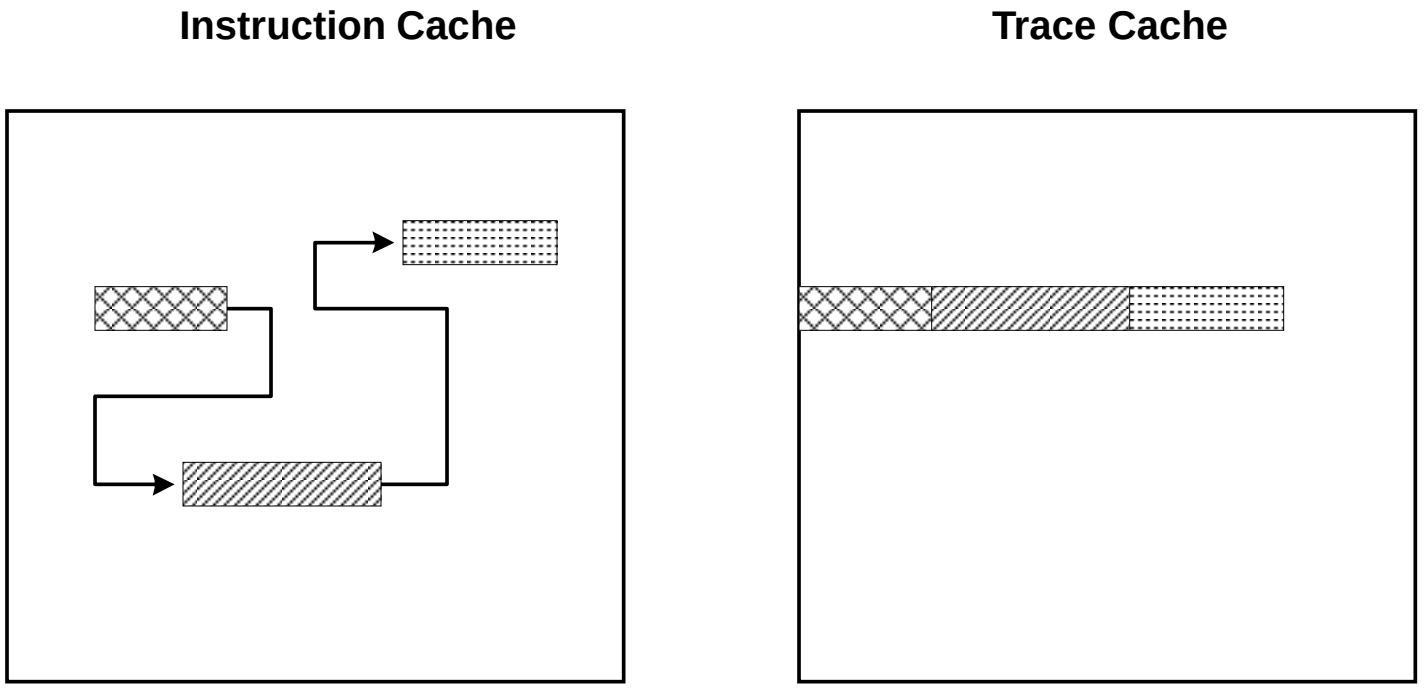
Intel Hyperthreading

- Main pipeline
  - Pipeline prior to trace cache not shown
- Round-Robin instruction fetching
  - Alternates between threads
  - Avoids dual-ported trace cache
  - BUT trace cache is a shared resource



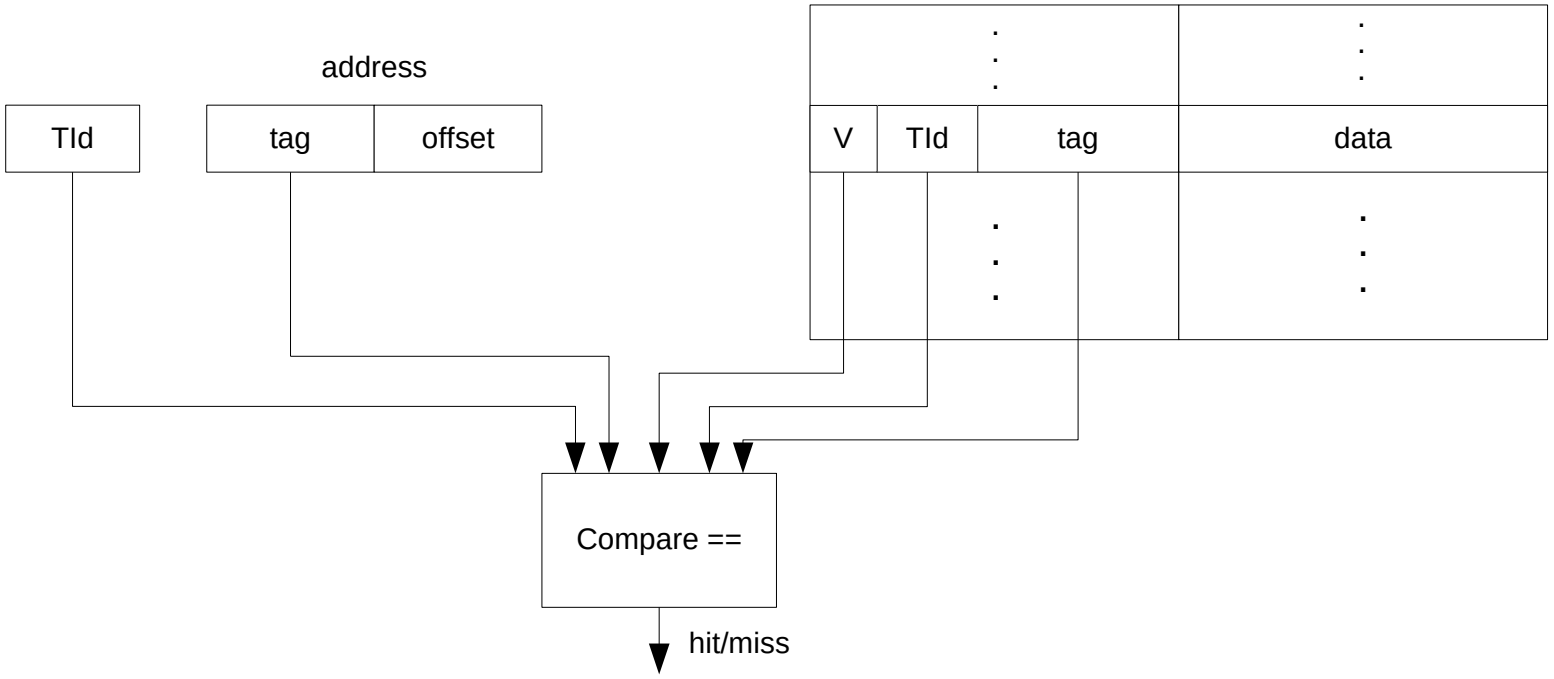
Trace Caches

- Trace cache captures dynamic traces
- Increases fetch bandwidth
- Help shorten pipeline (if predecoded)



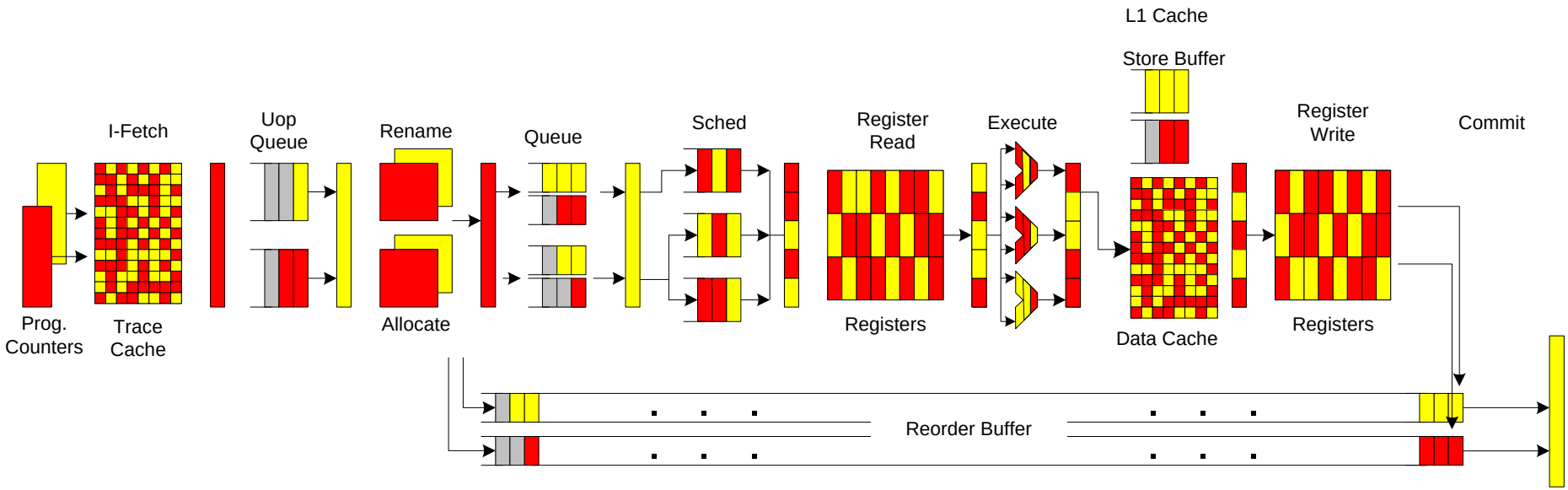
Capacity Resource Sharing

- Append thread identifier (TId) to threads in shared capacity (storage) resource
- Example: cache memory



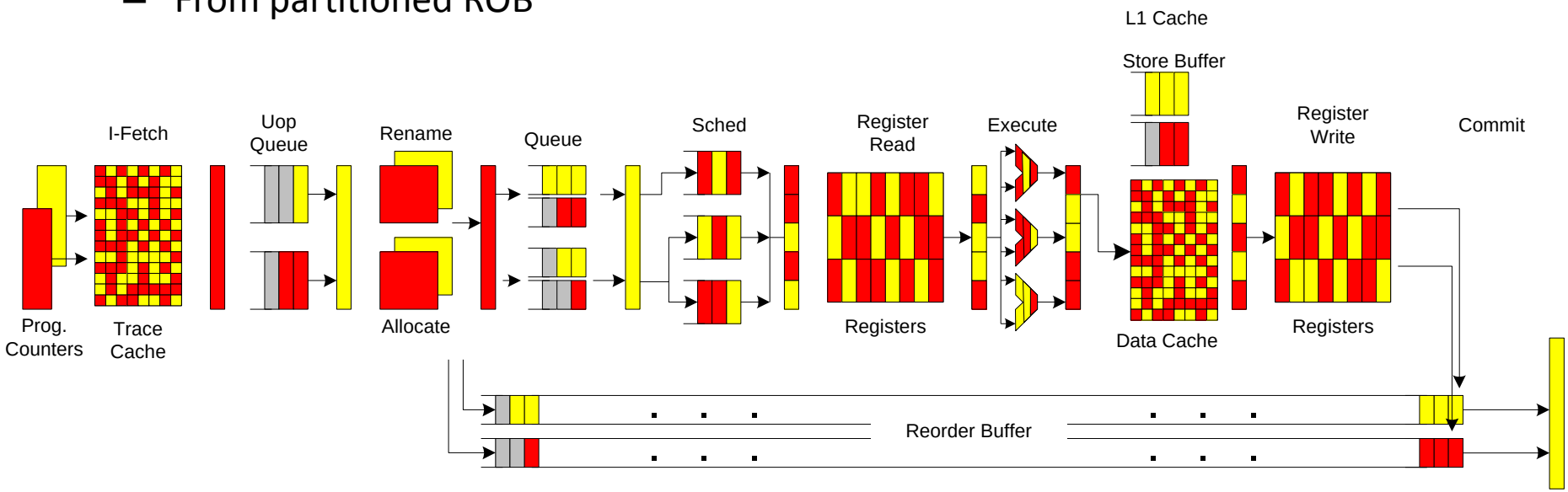
Frontend Implementation

- Partitioned front-end resources
  - Fetch queue (holds uops)
  - Rename and allocate tables
  - Post-rename queues
- Partitioning assures forward progress if other thread is blocked
  - Round-robin scheduling



Backend Implementation

- Physical registers are pooled (shared)
- Five instruction buffers (schedulers)
  - Shared
  - With an upper limit
- Instruction issue is irrespective of thread ID
- Instruction commit is round-robin
  - From partitioned ROB

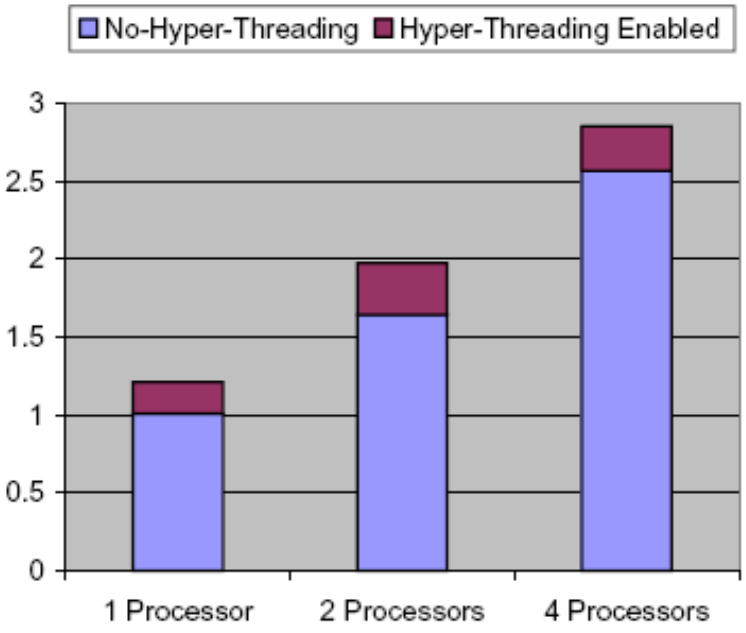


Operating Modes and OS Support

- MT-mode – two active logical processors; shared/partitioned resources
- ST-mode (ST0 or ST1) – one logical processor; combined resources
- HALT – privileged instruction => (normally) low power mode
  - IN MT mode => transition to ST0 or ST1
    (depending on the thread that HALTed)
  - In ST mode => low power mode
- Interrupt to HALTed thread => transition to MT mode
- OS manages two "processors"
  - OS code should HALT rather than idle loop
  - Schedule threads with priority to ST mode
    - (require OS knowledge of hyperthreading)

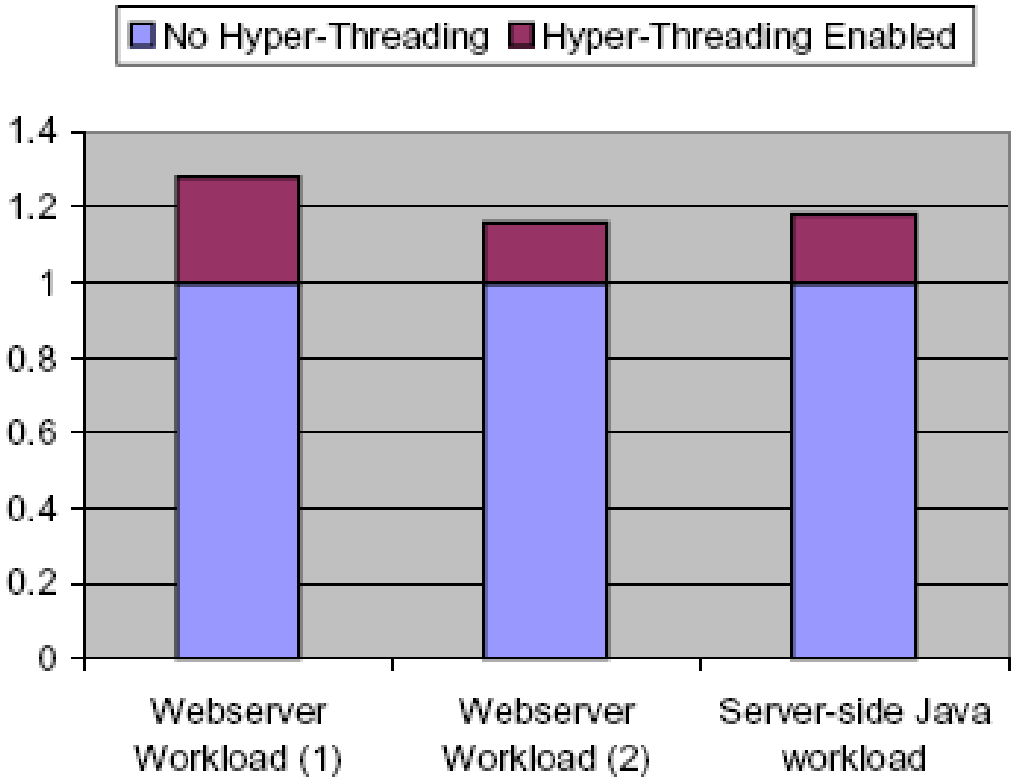
Performance

- OLTP workload
  - 21% gain in single and dual systems
  - Likely external bottleneck in 4 processor systems
    - Most likely front-side bus (FSB), i.e. memory bandwidth



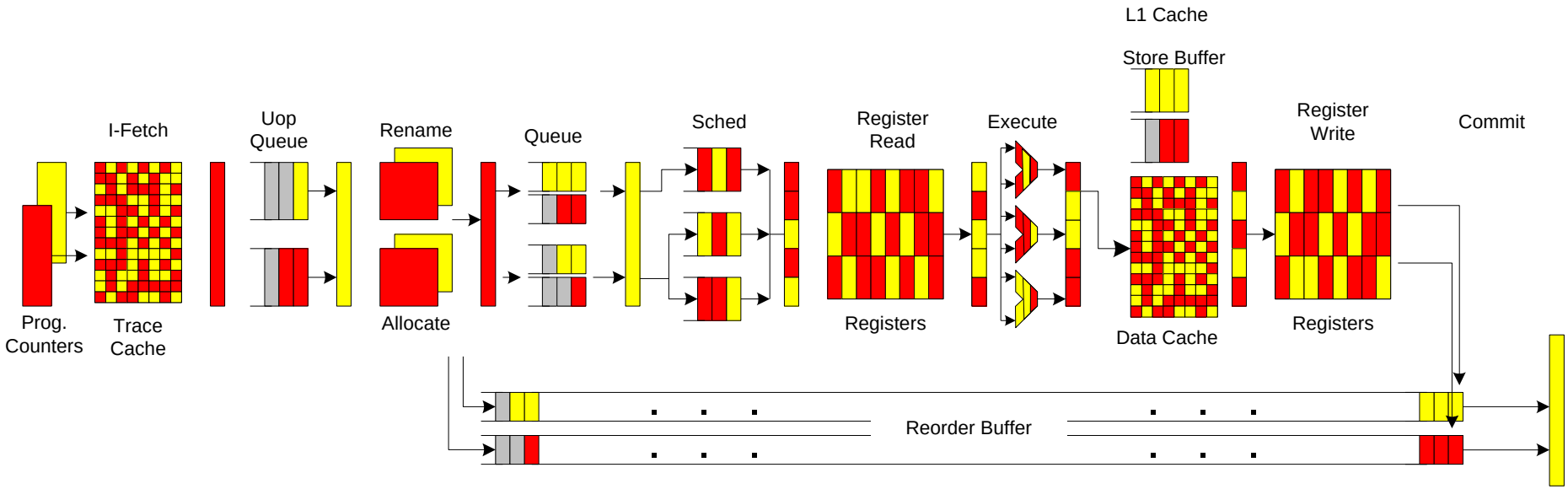
Performance

- Web server apps



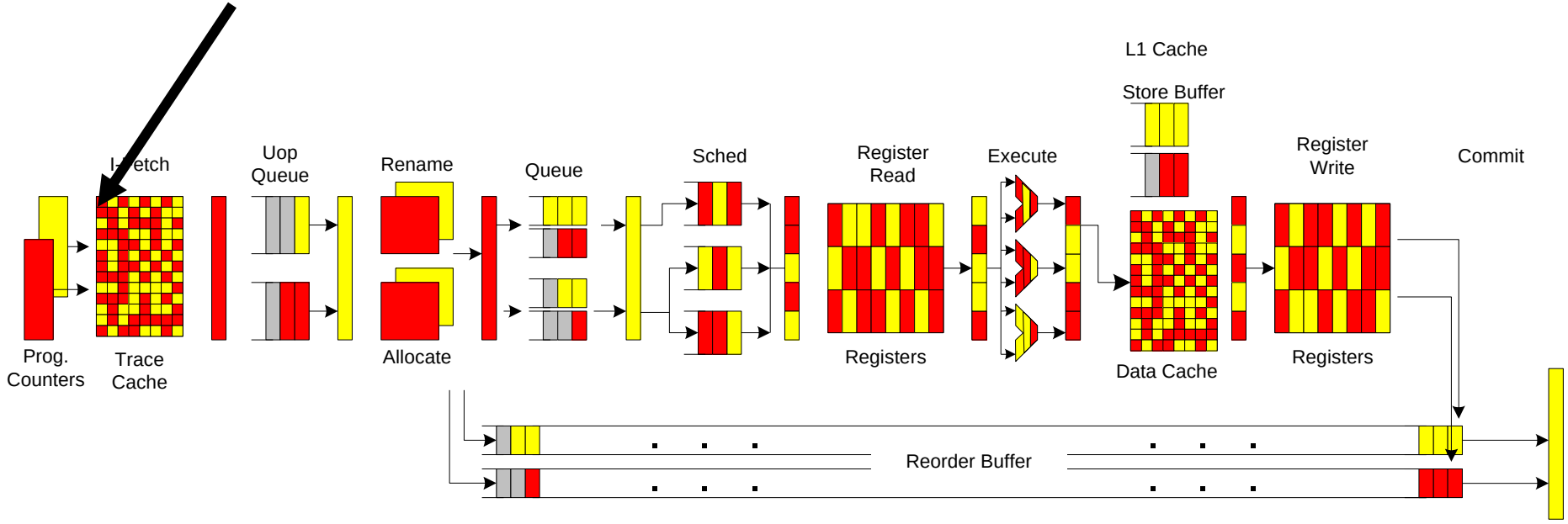
Intel Hyperthreading Summary

- Mix of partitioned and shared resources
- Mostly round-robin scheduling
- Primary objective: performance
- Secondary objective: fairness



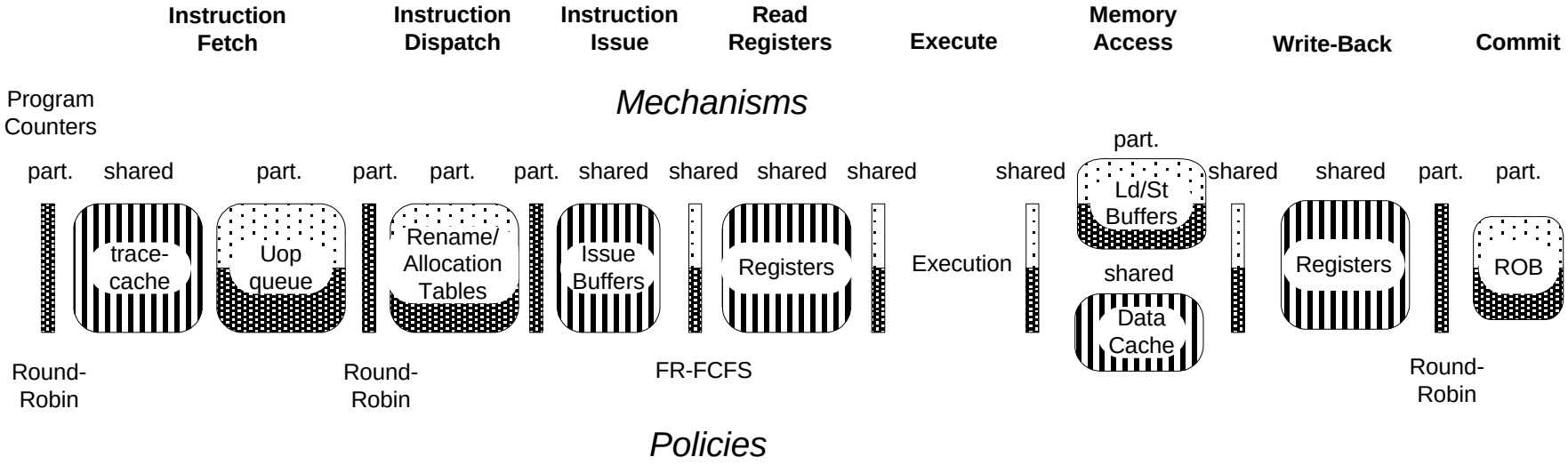
Policies and Mechanisms

- Separate primitives (mechanisms) from solutions (policies)
  - Generally good computer engineering
  - Allows flexibility in policies (during and after design)
- Example
  - Mechanism: Program counter multiplexer in IFetch stage
  - Policy: mux control – round-robin (or priorities)



Example: Hyperthreading

- Mechanisms (and Policies) in Pentium 4



Dispatch Policy

- Primary point for resource management
- Intel design uses software-settable priorities
  - Set via software writeable control register
  - Application software can control some of the priority settings
  - Priority 0 => idle, priority 1 => spinning
  - Both threads at level 1 => throttle to save power
- Hardware feedback can adjust priorities
- Reference:
  - D.M. Tullsen et al., ”Exploiting choice:…” ISCA 1996.

SMT Register File Capacity

- IBM Power8: 12 cores, 8T each, (32 FX+32FP) registers per thread. Consider FX only:
  - 8x32 = 256 physical registers minimum
  - Also need registers for inflight instructions
    - GCT (ROB) is 28x8=224 instructions x 75% = 168 registers
  - Total demand: 168+256 = 424 !!!
  - Very large, slow, power hungry
  - Instead: two-level RF for SMT8 mode:
    - 1st level is a cache of 124+124)
- More: B. Sinharoy et al., “IBM POWER8 processor core microarchitecture, ” IBM J R&D 59(1), Jan 2015.

Multithreading Summary

- Goal: increase throughput
  - Not latency
- Utilize execution resources by sharing among multiple threads:
  - Fine-grained, coarse-grained, simultaneous
- Usually some hybrid of fine-grained and SMT
  - Front-end is FG, core is SMT, back-end is FG
- Resource sharing
  - I$, D$, ALU, decode, rename, commit – shared
  - IQ, ROB, LQ, SQ – partitioned vs. shared
- Historic multithreaded machines
- Recent examples

Lecture 3 Outline

- Multithreaded processors
- Multicore processors

Processor Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) **(CPI)** **(cycle time)**

- In the 1980's (decade of pipelining):
  - CPI: 5.0 => 1.15
- In the 1990's (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case)
- In the 2000's (decade of multicore):
  - Core CPI unchanged; chip CPI scales with #cores

Multicore Objectives

- Use available transistors to add value
  - Provide better perf, perf/cost, perf/watt
- Effectively share expensive resources
  - Socket/pins:
    - DRAM interface
    - Coherence interface
    - I/O interface
  - On-chip area/power
    - Mem controller
    - Cache
    - FPU? (Conjoined cores, e.g. Niagara)

High-Level Design Issues

1. Where to connect cores?
   - Time to market:
     - at off-chip bus (Pentium D)
     - at coherence interconnect (Opteron Hypertransport)
   - Requires substantial (re)design:
     - at L2 (Power 4, Core Duo, Core 2 Duo, etc.)
     - at L3 (Opteron, Itanium, etc.)

High-Level Design Issues

2. Share caches?
   - yes: all designs that connect at L2 or L3
   - no: initial designs that connected at “bus”
3. Coherence?
   - Private caches? Reuse existing MP/socket coherence
     - Opportunity missed to optimize for on-chip sharing?
   - Shared caches?
     - Need new coherence protocol for on-chip caches
     - Could be write-through L1 with back-invalidates for other caches

High-Level Design Issues

4. How to connect?
   - Off-chip bus? Time-to-market hack, not scalable
   - Existing pt-to-pt coherence interconnect
     - e.g. AMD's hypertransport
   - Shared L2/L3:
     - Crossbar, up to 3-4 cores
     - 1D "UMA/dancehall" organization with crossbar
   - On-chip bus? Not very scalable
   - Interconnection network
     - scalable, but high overhead, design complexity
     - E.g. ring, 2D tiled organization, mesh interconnect

Shared vs. Private L2/L3

- Bandwidth issues
  - Data: if shared then banked/interleaved
  - Tags: snoop b/w into L2 (L1 if not inclusive)
- Cache misses: per core vs. per chip
  - Compare same on-chip capacity (e.g. 4MB)
  - When cores share data:
    - Cold/capacity/conflict misses fewer in shared cache
    - Communication misses greater in private cache
  - Conflict misses can increase with shared cache
    - Fairness issues between cores

Shared vs. Private L2/L3

- Access latency: fixed vs. NUCA (interconnect)
  - Classic UMA (dancehall) vs. NUMA/NUCA
  - Collocate LLC banks with cores
    - Commonly assumed in research literature
- Complexity due to bandwidth:
  - Arbitration
  - Concurrency/interaction
- Coherent vs. non-coherent shared LLC
  - LLC can be "memory cache" below "coherence"
  - Only trust contents after snoop/coherence has determined that no higher-level cache has a dirty copy

Multicore Coherence

- All private caches:
  - reuse existing protocol, if scalable enough
- Some shared cache
  - New LL shared cache is non-coherent (easy)
    - Use existing protocol to find blocks in private L2/L1
    - Serialize L3 access; use as memory cache
  - New shared LLC is coherent (harder)
    - Complexity of multilevel protocols is underappreciated
    - Could flatten (treat as peers) but:
      - Lose opportunity
      - May not be possible (due to inclusion, WB/WT handling)
    - Combinatorial explosion due to multiple protocols interacting

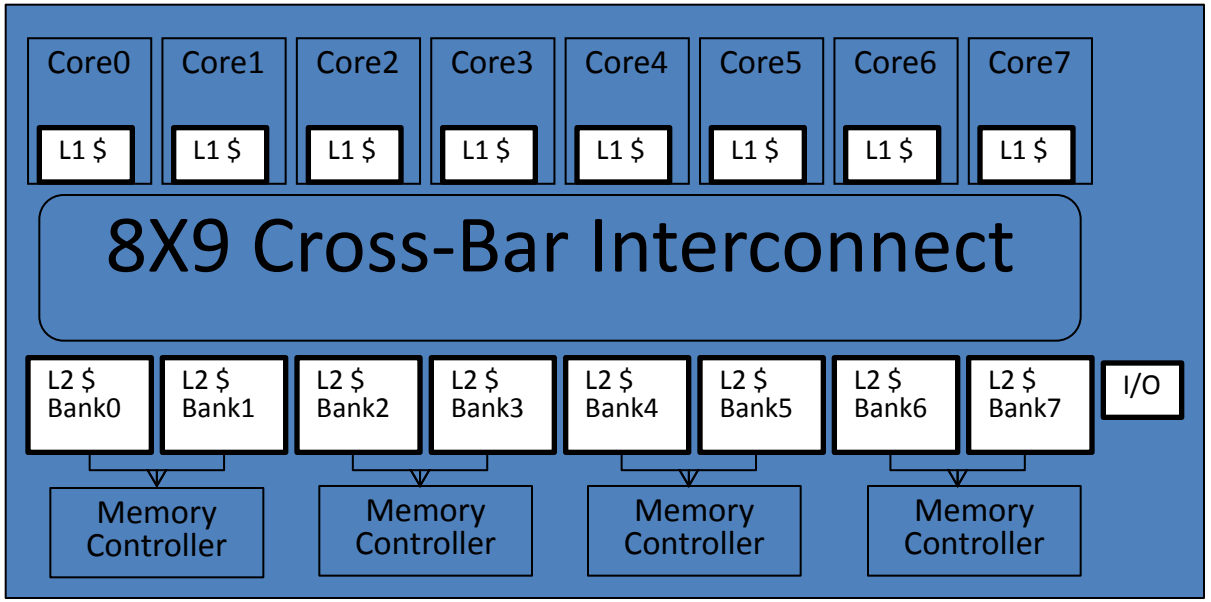
Multicore Coherence

- Shared L2 is coherent via writethru L1
  - Still need sharing list to forward invalidates/writes
  - *Ordering* of WT stores and conflicting loads, coherence messages not trivial
  - WT bandwidth is expensive
- Shared L2 with writeback L1
  - Combinatorial explosion of multiple protocols
  - Fractal coherence (MICRO ‘10), manager-client pairing (MICRO’11) addresses this issue

Multicore Interconnects

- Bus/crossbar - dismiss as short-term solutions?
- Point-to-point links, many possible topographies
  - 2D (suitable for planar realization)
    - Ring
    - Mesh
    - 2D torus
  - 3D - may become more interesting with 3D packaging (chip stacks)
    - Hypercube
    - 3D Mesh
    - 3D torus
- More detail in subsequent NoC unit

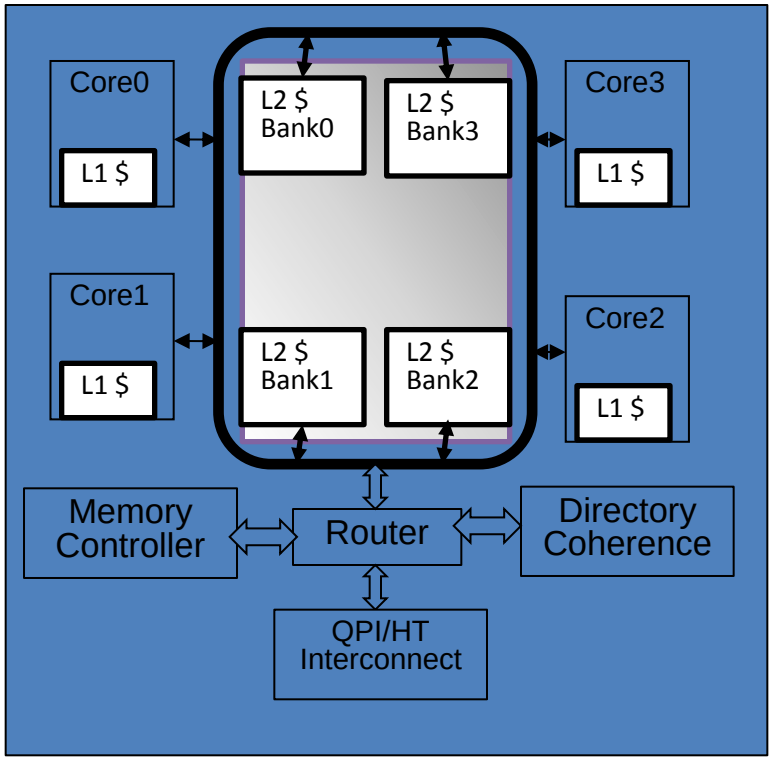
Cross-bar (IBM Power4-Power8)



On-Chip Bus/Crossbar

- Used widely (Power4/5/6/7/8, Piranha, Niagara, etc.)
  - Assumed not scalable
  - Is this really true, given on-chip characteristics?
  - Scales "far enough" : watch out for arguments at the limit
  - e.g. swizzle-switch makes x-bar scalable enough [UMich]
- Simple, straightforward, nice ordering properties
  - Wiring can be a nightmare (for crossbar)
  - Bus bandwidth is weak (even multiple busses)
  - Compare DEC Piranha 8-lane bus (32GB/s) to Power4 crossbar (100+GB/s)
  - Workload BW demands: commercial vs. scientific

On-Chip Ring (e.g. Intel)



On-Chip Ring

- Point-to-point ring interconnect
  - Simple, easy
  - Nice ordering properties (unidirectional)
  - Every request a broadcast (all nodes can snoop)
  - Scales poorly: *O(n)* latency, fixed bandwidth
- Optical ring (nanophotonic)
  - HP Labs Corona project [ISCA 08]
  - Much lower latency (speed of light)
  - Still fixed bandwidth (but lots of it)

On-Chip Mesh

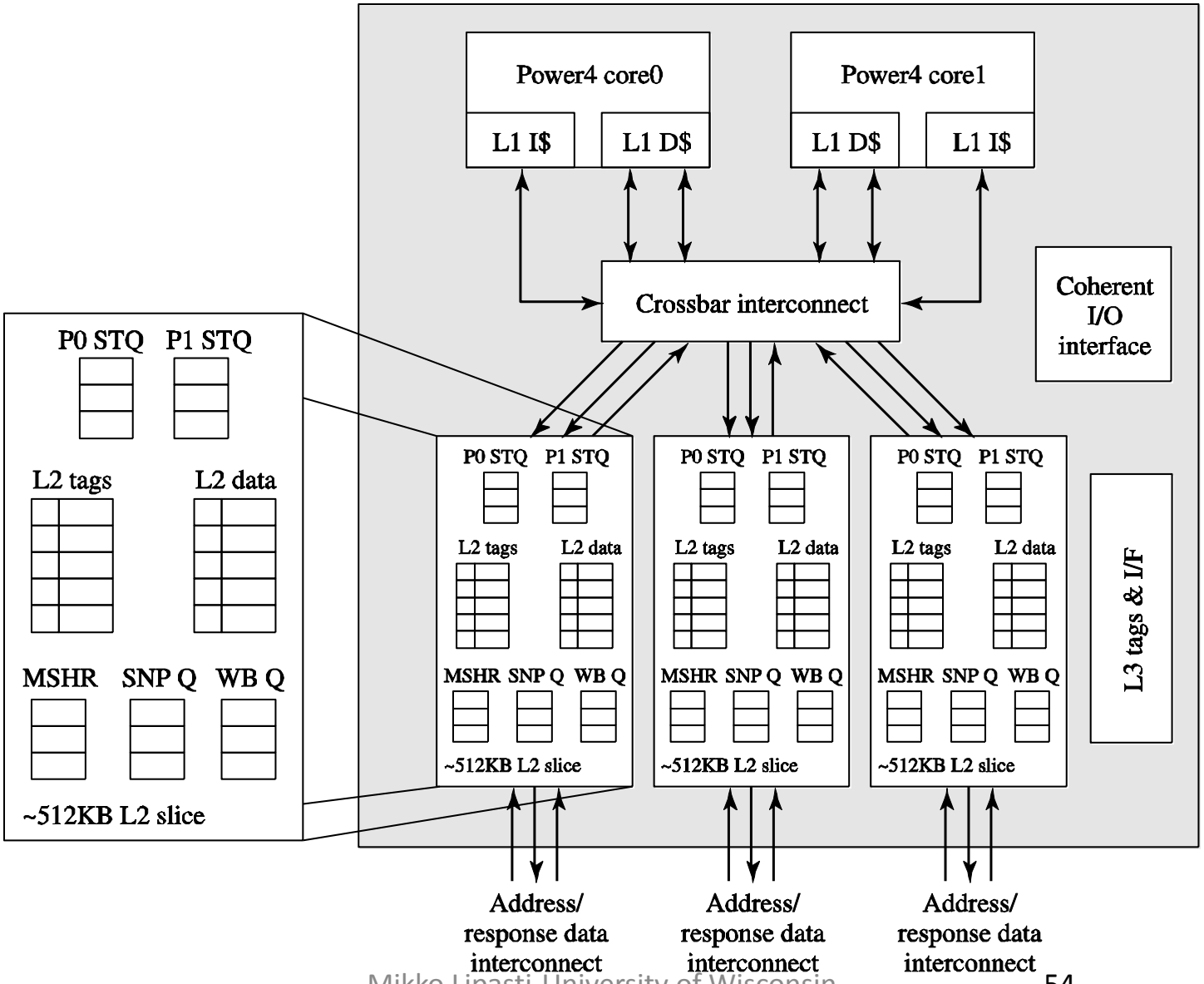
- Widely assumed in academic literature
- Tilera (Wentzlaff reading [19])
- Not symmetric, so have to watch out for load imbalance on inner nodes/links
  - 2D torus: wraparound links to create symmetry
    - Not obviously planar
    - Can be laid out in 2D but longer wires, more intersecting links
- Latency, bandwidth scale well
- Lots of recent research in the literature

CMP Examples

- Chip Multiprocessors (CMP)
- Becoming very popular

Processor	Cores/ chip	Multi-threaded?	Resources shared
IBM Power 4	2	No	L2/L3, system interface
IBM Power7	8	Yes (4T)	Core, L2/L3, system interface
Sun Ultrasparc	2	No	System interface
Sun Niagara	8	Yes (4T)	Everything
Intel Pentium D	2	Yes (2T)	Core, nothing else
AMD Opteron	2	No	System interface (socket)

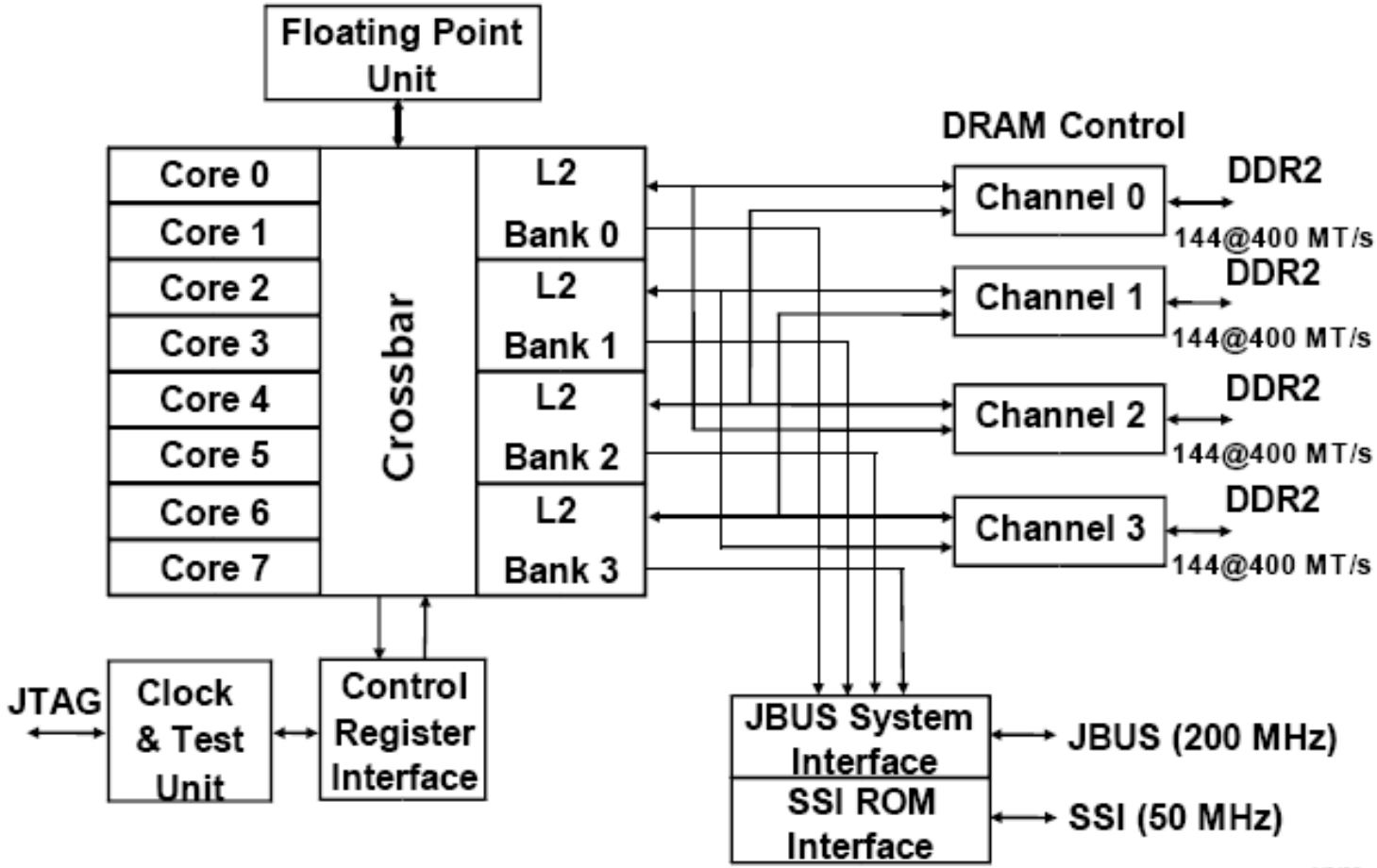
IBM Power4: Example CMP



Niagara Case Study

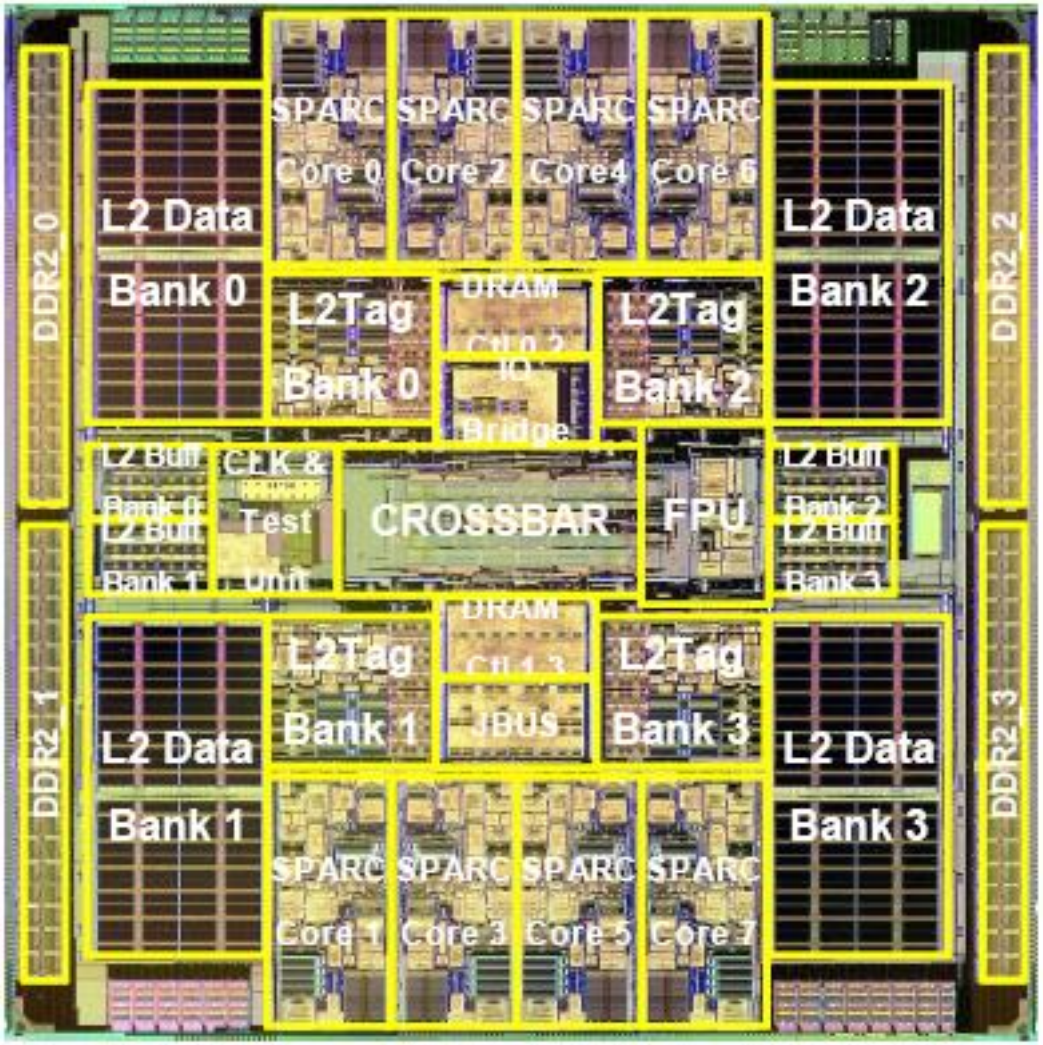
- Targeted application: web servers
  - Memory intensive (many cache misses)
  - ILP limited by memory behavior
  - TLP: Lots of available threads (one per client)
- Design goal: maximize throughput (/watt)
- Results:
  - Pack many cores on die (8)
  - Keep cores simple to fit 8 on a die, share FPU
  - Use multithreading to cover pipeline stalls
  - Modest frequency target (1.2 GHz)

Niagara Block Diagram [Source: J. Laudon]



- 8 in-order cores, 4 threads each
- 4 L2 banks, 4 DDR2 memory controllers

Ultrasparc T1 Die Photo [Source: J. Laudon]



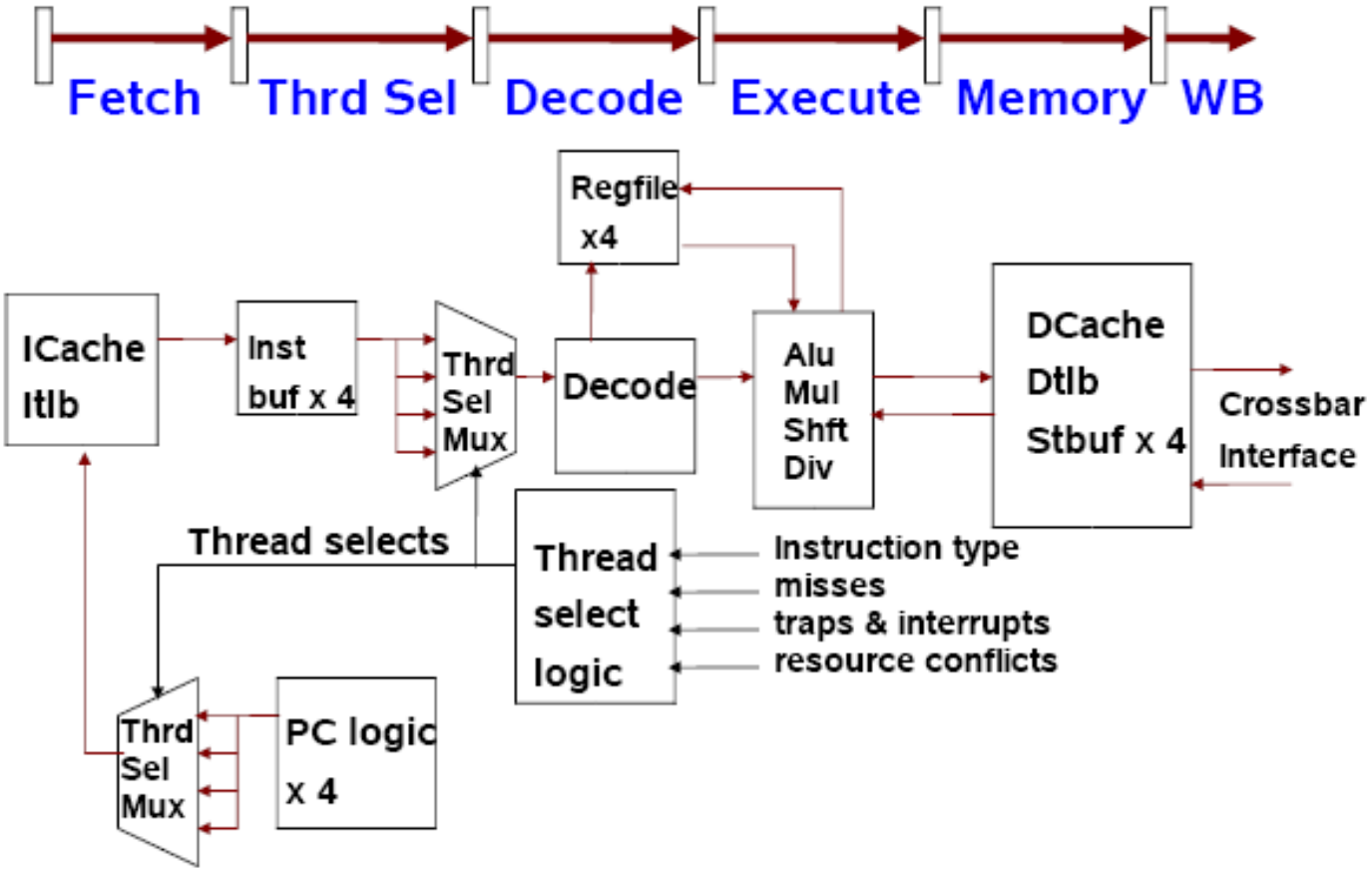
Features:

- 8 64-bit Multithreaded SPARC Cores
- Shared 3 MB, 12-way 64B line writeback L2 Cache
- 16 KB, 4-way 32B line ICache per Core
- 8 KB, 4-way 16B line write-through DCache per Core
- 4 144-bit DDR-2 channels
- 3.2 GB/sec JBUS I/O

Technology:

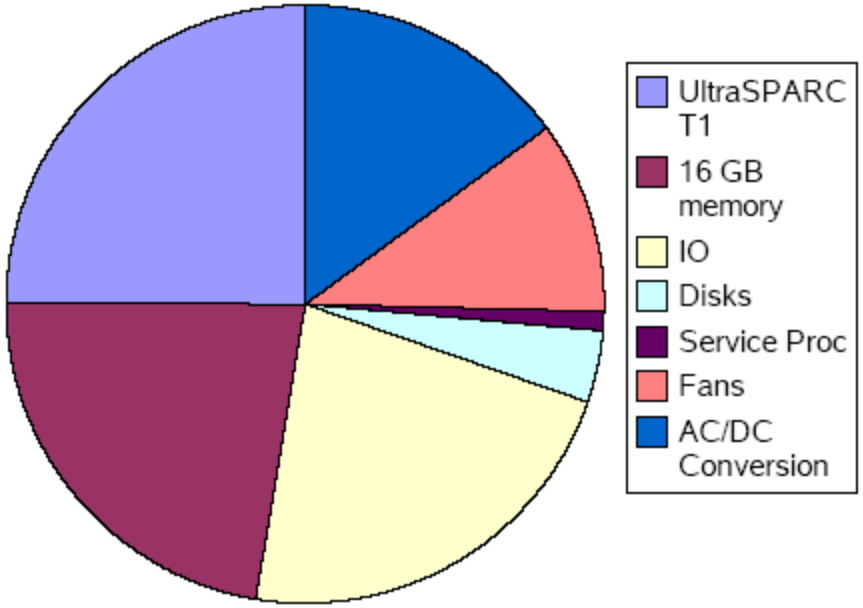
- TI's 90nm CMOS Process
- 9LM Cu Interconnect
- 63 Watts @ 1.2GHz/1.2V
- Die Size: 379mm²
- 279M Transistors
- Flip-chip ceramic LGA

Niagara Pipeline [Source: J. Laudon]



- Shallow 6-stage pipeline
- Fine-grained multithreading

T2000 System Power



- 271W running SpecJBB2000
- Processor is only 25% of total
- DRAM & I/O next, then conversion losses

Niagara Summary

- Example of *application-specific* system optimization
  - Exploit application behavior
    - TLP, cache misses, low ILP
  - Build very efficient solution
- Downsides
  - Loss of *general-purpose* suitability
  - E.g. poorly suited for software development (parallel make, gcc)
  - Very poor FP performance (fixed in Niagara 2)

CMPs WITH HETEROGENEOUS CORES

- Workloads have different characteristics
  - Large number of small cores (applications with high thread count)
  - Small number of large cores (applications with single thread or limited thread count)
  - Mix of workloads
  - Most parallel applications have both serial and parallel sections (Amdahl's Law)

- Hence, heterogeneity
  - Temporal: EPI throttling via DVFS
  - Spatial: Each core can differ either in performance or functionality

- Performance asymmetry
  - Using homogeneous cores and DVFS, or processor with mixed cores (ARM BIG.little)
  - Variable resources: e.g., adapt size of cache via power gating of cache banks
  - Speculation control (unpredictable branches): throttle in-flight instructions (reduces activity factor)

Method	EPI Range	Time to vary EPI
DVFS	1:2 to 1:4	100 us, ramp V_{cc}
Variable Resources	1:1 to 1:2	1 us, Fill L1
Speculation Control	1:1 to 1:1.4	10 ns, Pipe flush
Mixed Cores	1:6 to 1:11	10 us, Migrate L2

CMPs WITH HETEROGENEOUS CORES (Functional Asymmetry)

- Use heterogeneous cores
  - E.g., GP cores, GPUs, cryptography, vector cores, floating point coprocessors
  - Heterogeneous cores may be programmed differently
  - Mechanisms must exist to transfer activity from one core to another
    - Fine-grained: e.g. FP co-processor, use ISA
    - Coarse-grained: transfer computation using APIs
- Examples:
  - Cores with different ISAs
  - Cores with different cache sizes, different issue width, different branch predictors
  - Cores with different micro-architectures (in-order vs. out-of-order)
  - Different types of cores (GP and SIMD)
- Goals:
  - Save area (more cores)
  - Save power by using cores with different power/performance characteristics for different phases of execution

CMPs WITH HETEROGENEOUS CORES

- Different applications may have better performance/power characteristics on some types of core (static)
- Same application goes through different phases that can use different cores more efficiently (dynamic)
  - Execution moves from core to core dynamically
  - Most interesting case (dynamic)
  - Cost of switching cores (must be infrequent: such as O/S time slice)
- Assume cores with same ISA but different performance/energy ratio
  - Need ability to track performance and energy to make decisions
  - Goal: minimize energy-delay product (EDP)
  - Periodically sample performance and energy spent
    - Run application on one or multiple cores in small intervals
  - Possible heuristics
    - Neighbor: pick one of the two neighbors at random, sample, switch if better
    - Random: select a core at random and sample, switch if better
    - All: sample all cores and select the best
    - Consider the overhead of sampling

Multicore Summary

- Objective: resource sharing, power efficiency
  - Where to connect
  - Cache sharing
  - Coherence
  - How to connect
- Examples/case studies
- Heterogeneous CMPs
- Readings
  - [6] K. Olukotun, et al., "The Case for a Single-Chip Multiprocessor," ASPLOS-7, October 1996.
  - [7] Kumar, R., et al., "Heterogeneous Chip Multiprocessors", IEEE Computer, pp. 32-38, Nov. 2005

Lecture 3 Summary

- Multithreaded processors
- Multicore processors