Lecture 13:
Advanced CPU Microarchitecture, pt.2

Jangwoo Kim (Seoul National University)
jangwoo@snu.ac.kr

Announcement #1

- Start reading P&H: Chapter 5
  - Cache and memory
  - Not covered by Mid-term

- HW#3
  - **Due: 4/19**

- Mid-term
  - **4/20 (6PM) @ Room 102**

Announcement #2

- Papers to be read
  - **Must understand these papers for your mid-term exam!**

Lec 12 —

(1) Superscalar + Out-of-Order execution // the most important one!

- "**The Microarchitecture of Superscalar Processors**" by Smith, Proceedings of IEEE 1995.

Lec 13 —

(2) Simultaneous Multithreading (SMT)

- "**Simultaneous Multithreading: A Platform for Next-Generation Processors**" by Eggers, et al., IEEE Micro 1997

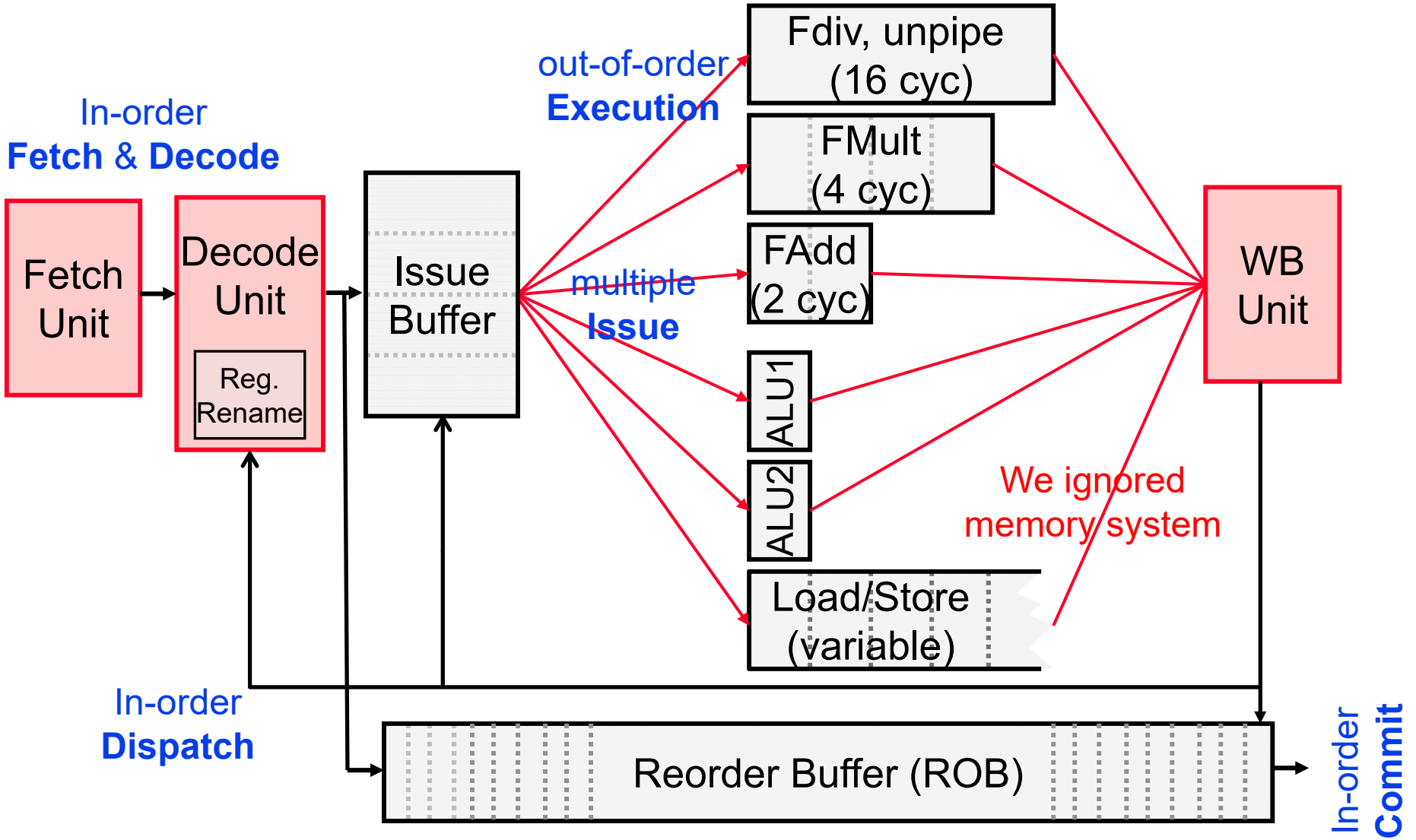
(3) Chip Multiprocessor (CMP)

- "**Niagara: A 32-Way Multithreaded SPARC Processor**" by Kongetira, et al. IEEE Micro 2005

- **What did you learn? Main ideas?**
- **Why important? Any limitations?**

We DO ask questions based on these readings!!

Review: Out-of-Order Superscalar



Read "**The Microarchitecture of Superscalar Processors**" [Smith 1995, Proceedings of IEEE]

Hardware Multithreading

Performance And Utilization

- Performance (IPC) is important

  Utilization (actual IPC / peak IPC) is important too!

- Even moderate superscalars (e.g., 4-way) not fully utilized
  - Average sustained IPC: 1.5~2 → <50% utilization
    - Branch mis-predictions
    - Cache misses, especially L2
    - Data dependences

- **Multi-threading (MT)**
  - One thread cannot fully utilize CPU? Maybe 2, 4 (or 100) can!
  - Improve utilization by executing multiple threads on a single CPU simultaneously.

Latency vs Throughput

- **MT trades (single-thread) latency for throughput**
  - – Sharing processor degrades latency of individual threads
  - \+ But improves aggregate latency of both threads
  - \+ Improves utilization

Example:

- Thread A: individual latency=10s, latency with thread B=15s
- Thread B: individual latency=20s, latency with thread A=25s
  - Sequential latency (first A then B or vice versa): 30s
  - Parallel latency (A and B simultaneously): 25s
- – MT slows each thread by 5s
- \+ But improves total latency by 5s

- **Different workloads have different parallelism**
  - SpecFP has lots of ILP (can use an 8-way superscalar)
  - Server workloads have TLP (can use multiple threads)
    - TLP: thread-level parallelism

MT: Core Sharing

Time sharing:

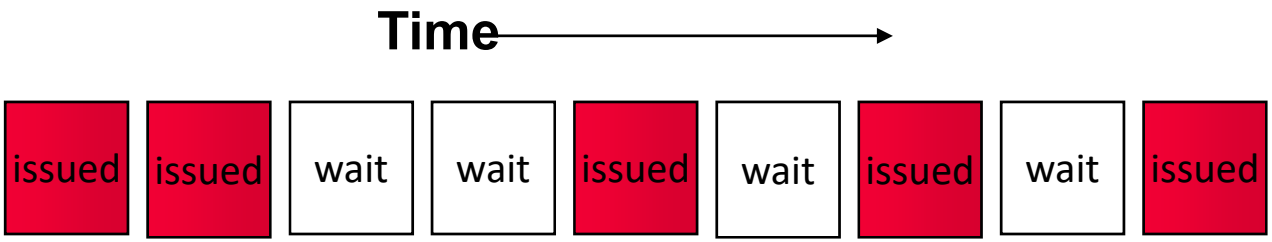
- Run one thread at a time
- On a long-latency operation (e.g., cache miss), switch threads
- Also known as "switch-on-miss" multithreading
  - E.g., Sun UltraSPARC T1/T2: "Niagara"

Space sharing:

- Simple: across pipeline depth (a.k.a., pipeline interleaving)
  - Fetch and issue each cycle from a different thread
- Advanced: both across "pipeline depth" and "width"
  - Fetch and issue each cycle from multiple threads
  - Policy to decide "which to fetch" gets complicated

    (a.k.a., Simultaneous Multithreading (SMT))
  - E.g., Intel Nehalem, IBM POWER7, SPARC T4

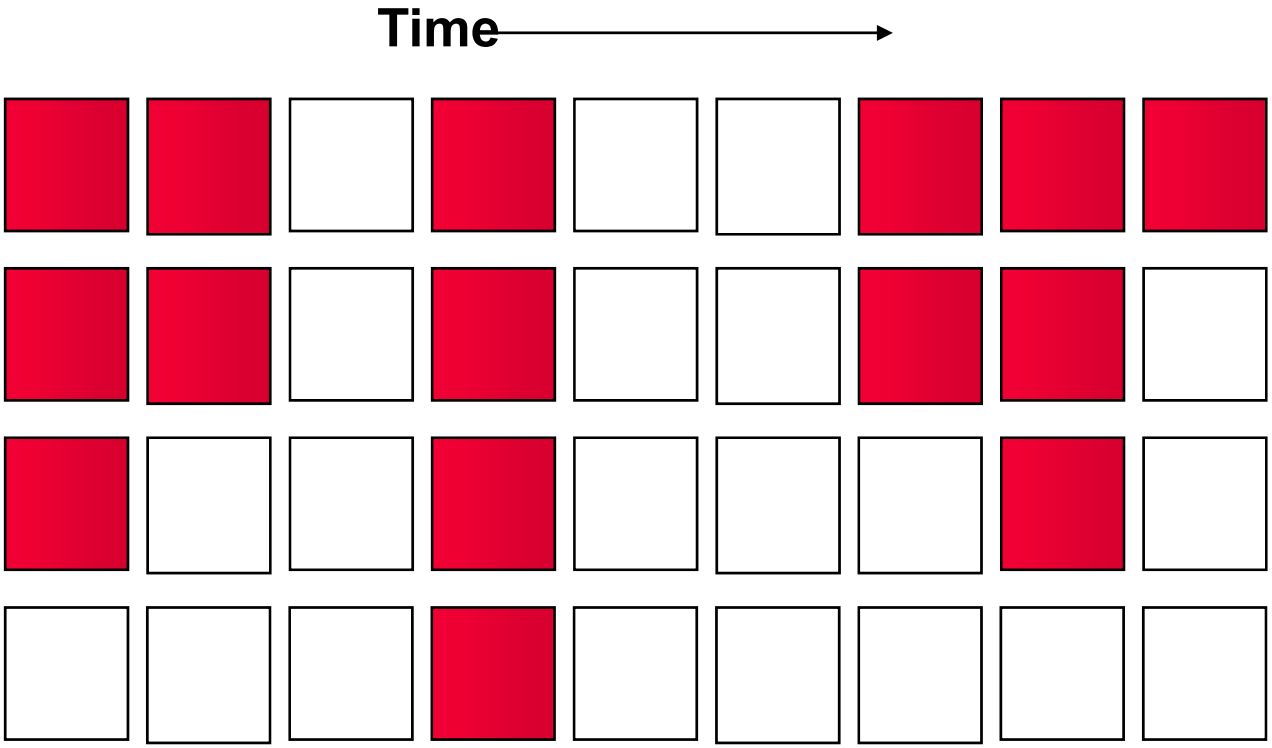
Instruction Issue



1 instruction-width pipeline

→ reduced function unit utilization due to dependencies

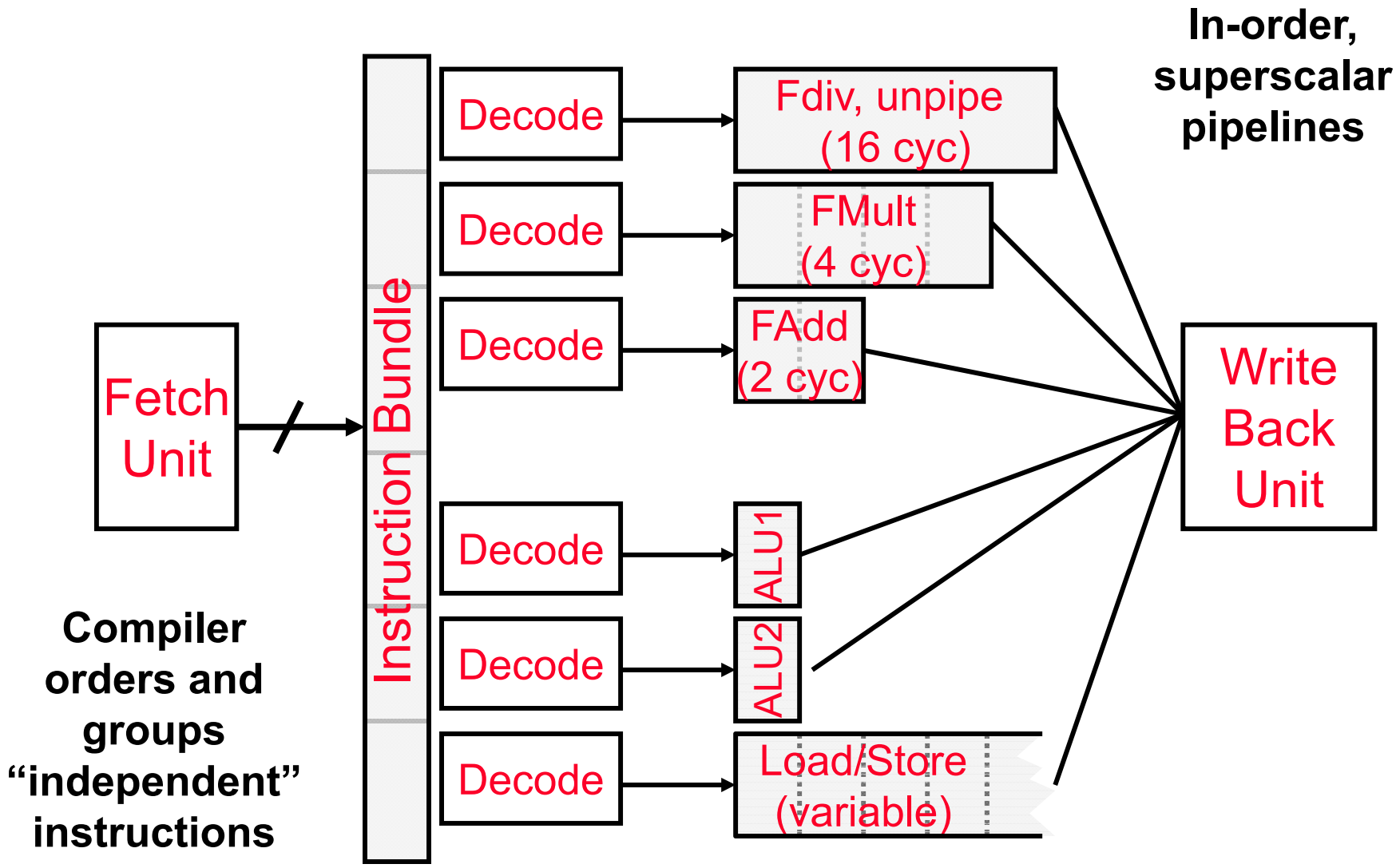
Superscalar Issue



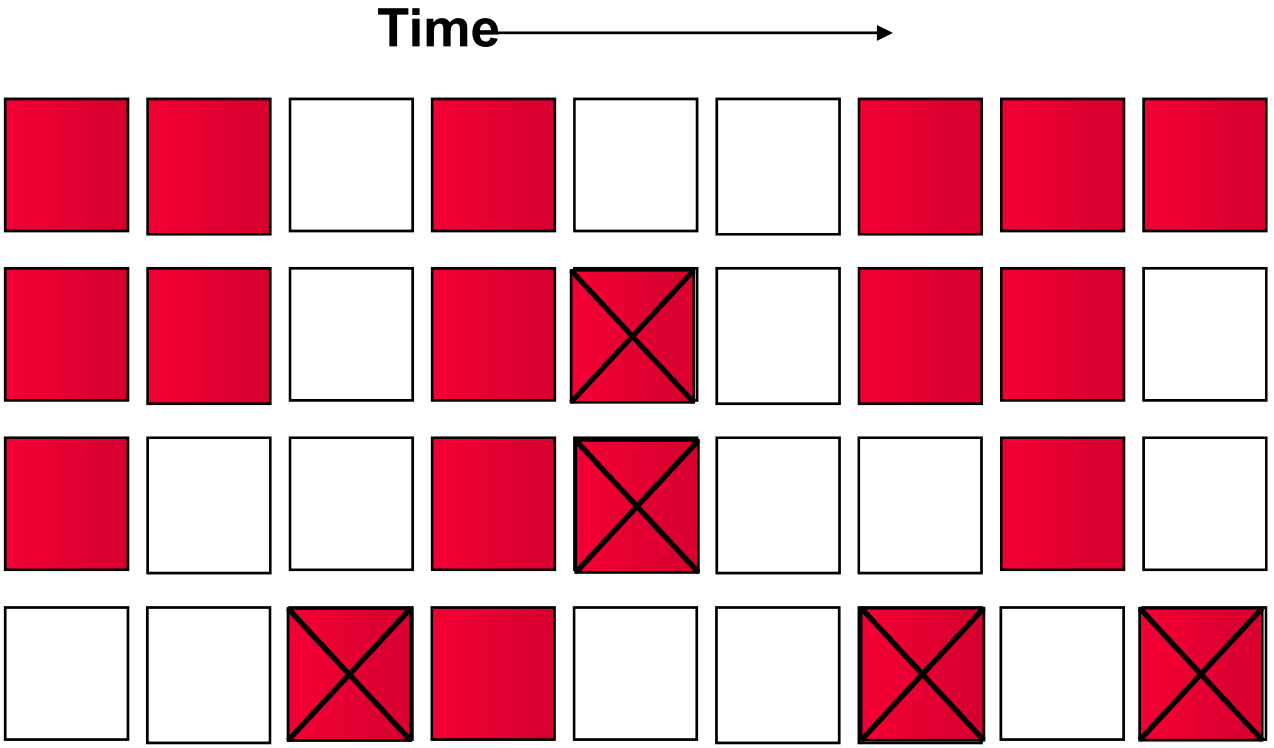
"Superscalar" leads to more performance, but lower utilization

Very Long Instruction Words (VLIW)

(e.g., Intel Itanium)

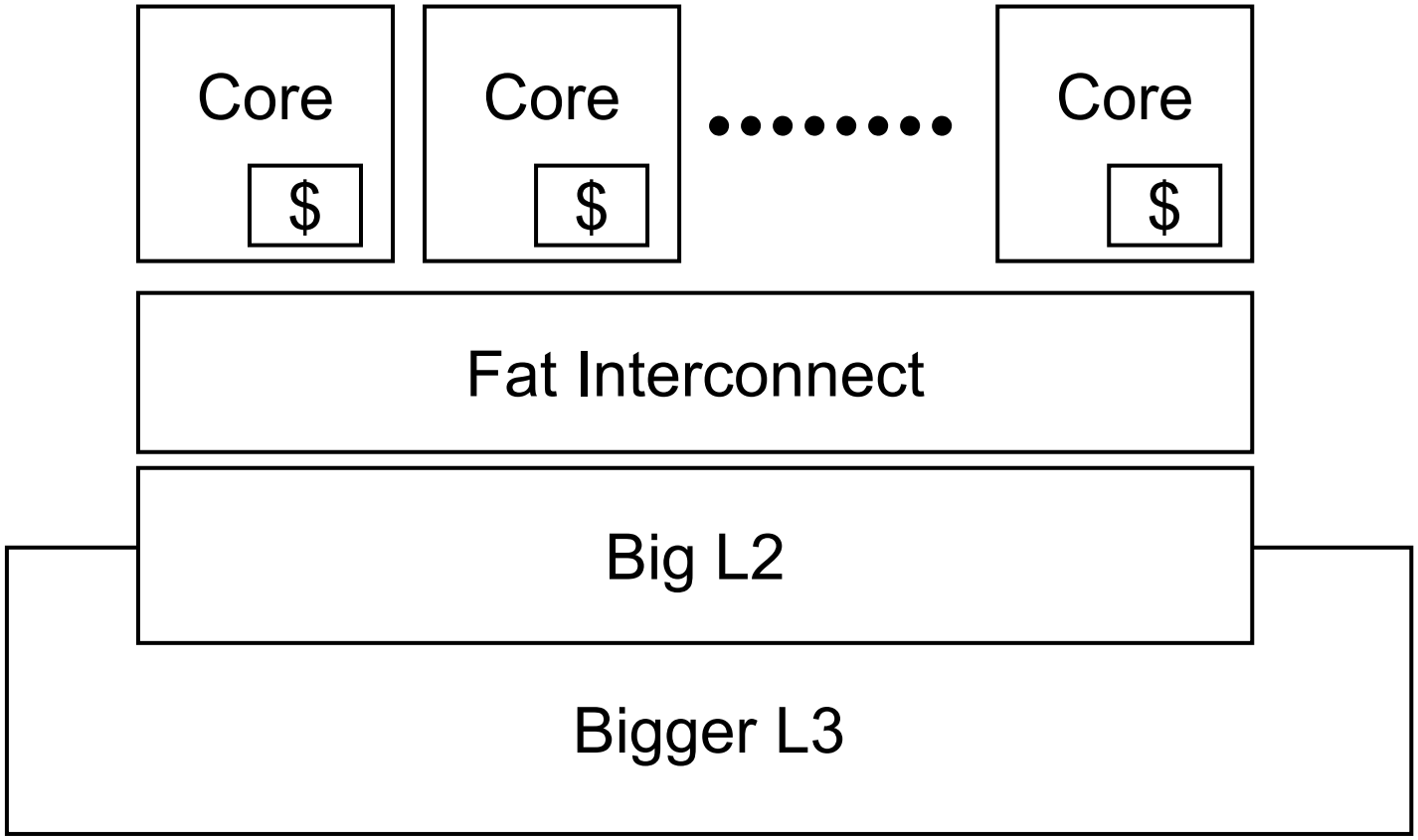


Predicated Issue



Adds to function unit utilization, but results are thrown away

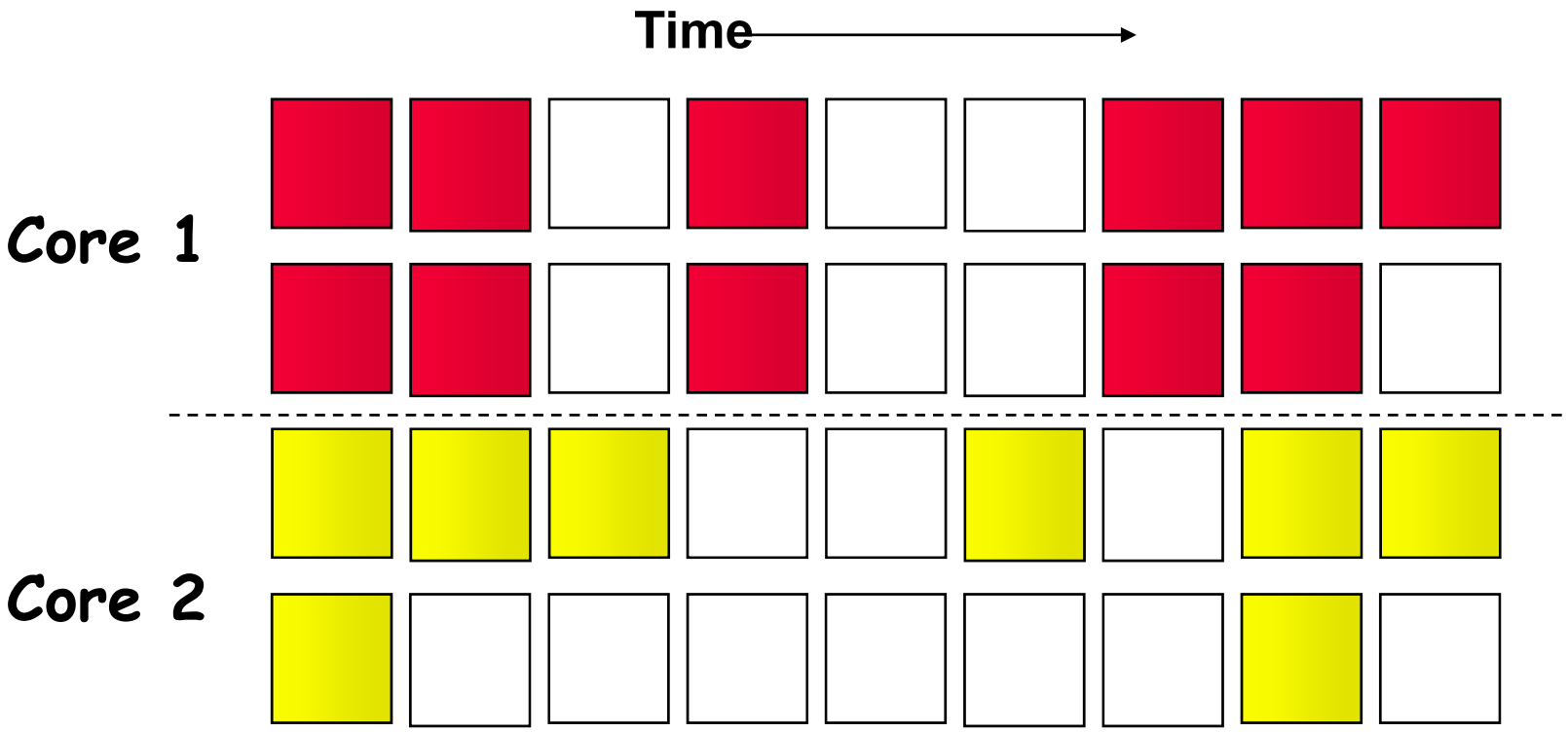
Multi Core: Chip-Multiprocessor (CMP)



Which one is better?

8 In-order cores @ 1Ghz VS single OoO-SS core @ 2ghz ?

Chip Multiprocessor (CMP)



Limited utilization when only running one thread

Introduction: CMP

- **Moore's Law still valid**
  - How can we use many transistors?

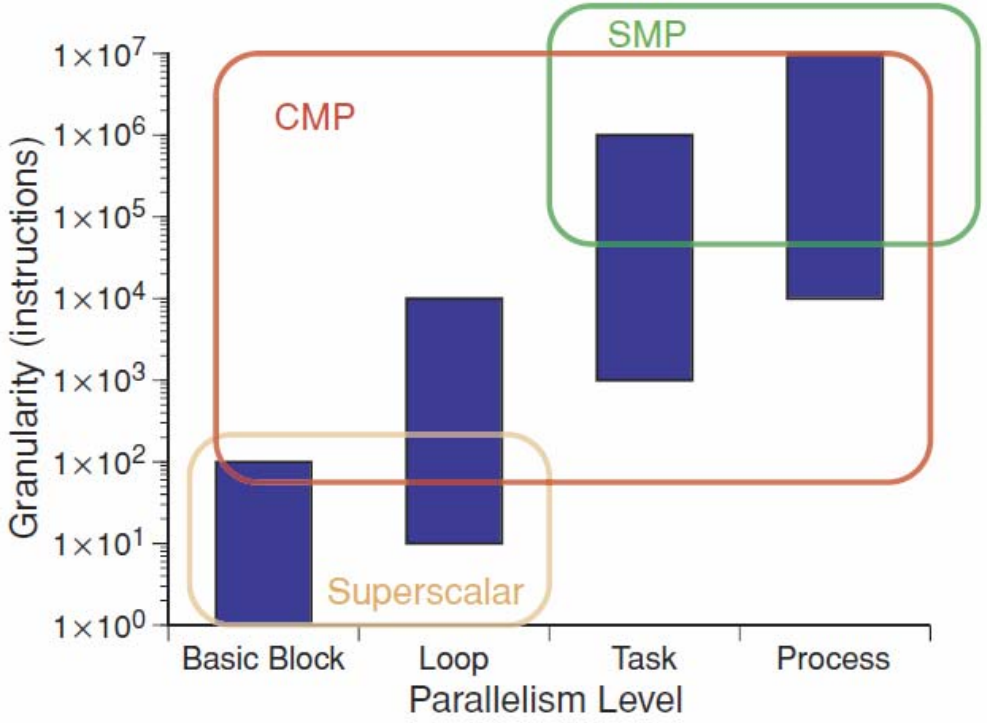
- **Chip Multiprocessor (CMP)**

  (or multicore microprocessor)
  - Putting multiple smaller and simpler cores on chip
  - Easy to build by filling the area with multiple simple cores
  - Necessary to provide enough memory & I/O bandwidth
  - Applications must have enough **Thread Level Parallelism (TLP)**
    - Application must be divided to a group of **threads**.
    - Threads must run in parallel

CMP does not make a sense without TLP.
How much TLP in your everyday workloads?

Introduction: ILP vs TLP

- **Granularities of parallelism**
  - Instruction / basic block // small ILP
  - Loop // large ILP
  - Tasks // threads
  - Processes // programs



Symmetric multiprocessor (SMP)
= Multiple CPUs in a system

Chip multiprocessor (CMP)
= Multiple cores on a chip

Superscalar
= Multiple issue logic on a core

Introduction: Superscalar VS CMP

- **Out-of-Order SS or In-Order CMP on the same area**

6-way OoO SS

Few large units

Big predictor

Large ISQ

Faster L2

	6-WAY SS	4 x 2-WAY MP
♯ of CPUs	1	4
Degree superscalar	6	4 × 2
♯ of architectural registers	32int/32fp	4 × 32int/32fp
♯ of physical registers	160int/160fp	4 × 40int/40fp
♯ of integer functional units	3	4 × 1
♯ of floating pt. functional units	3	4 × 1
♯ of load/store ports	8 (one per bank)	4 × 1
BTB size	2048 entries	4 × 512 entries
Return stack size	32 entries	4 × 8 entries
Instruction issue queue size	128 entries	4 × 8 entries
I cache	32 KB, 2-way S. A.	4 × 8 KB, 2-way S. A.
D cache	32 KB, 2-way S. A.	4 × 8 KB, 2-way S. A.
L1 hit time	2 cycles	1 cycle
L1 cache interleaving	8 banks	N/A
Unified L2 cache	256 KB, 2-way S. A.	256 KB, 2-way S. A.
L2 hit time/L1 penalty	4 cycles	5 cycles
Memory latency/L2 penalty	50 cycles	50 cycles

4 x 2-way IO SS

Many simple units

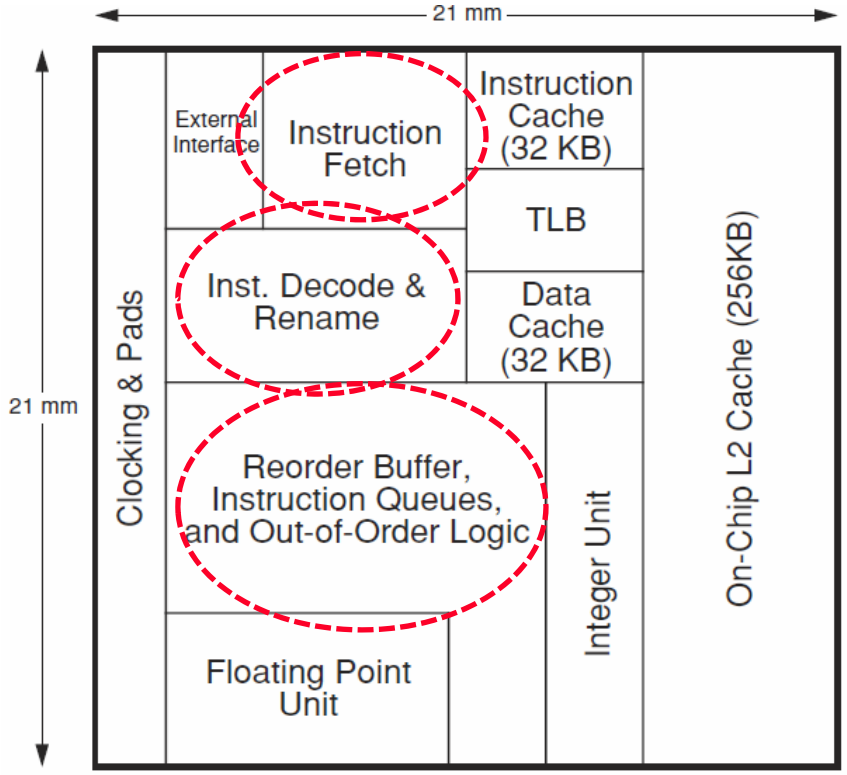
4 x simple predictors

4 x small ISQs

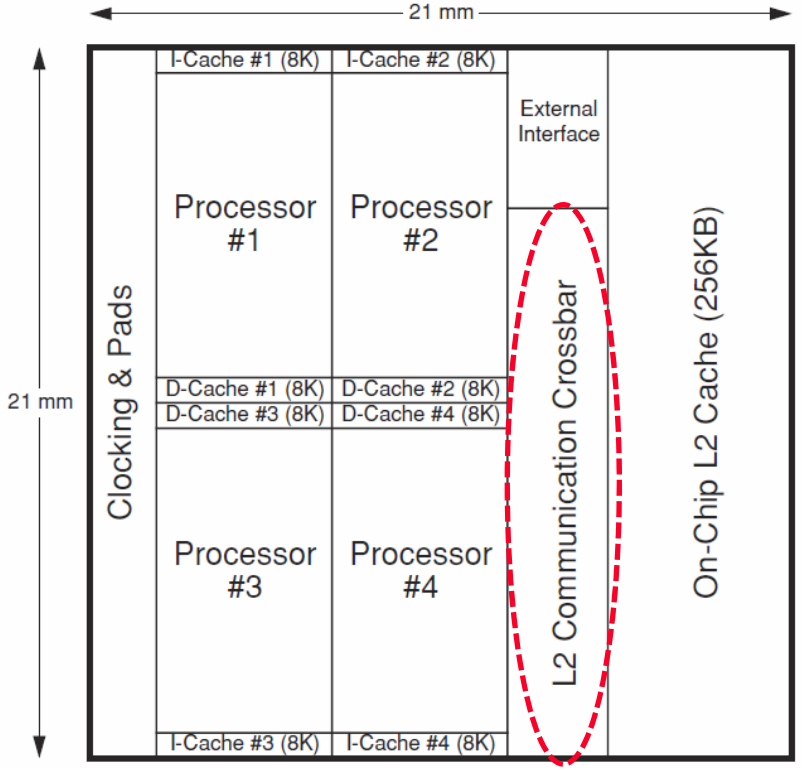
Faster L1

Introduction: Superscalar VS CMP

- **Area is "price": OoO SS or In-Order CMP on the same area**



VS

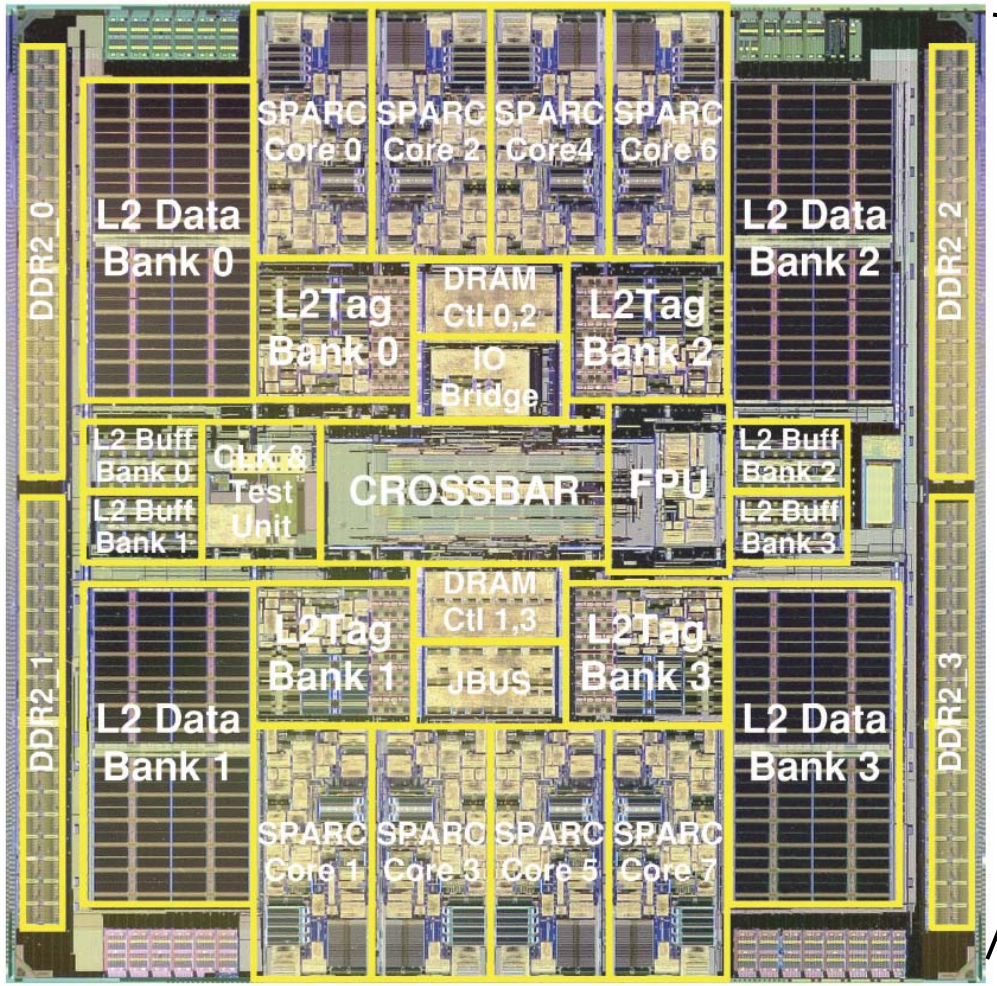


30% area taken for OoO & SS

Many simple cores
(with interconnection)

Niagara: CMP with multi-threading

- **Sun Niagara I (=UltraSPARC T1) : "die" photo**

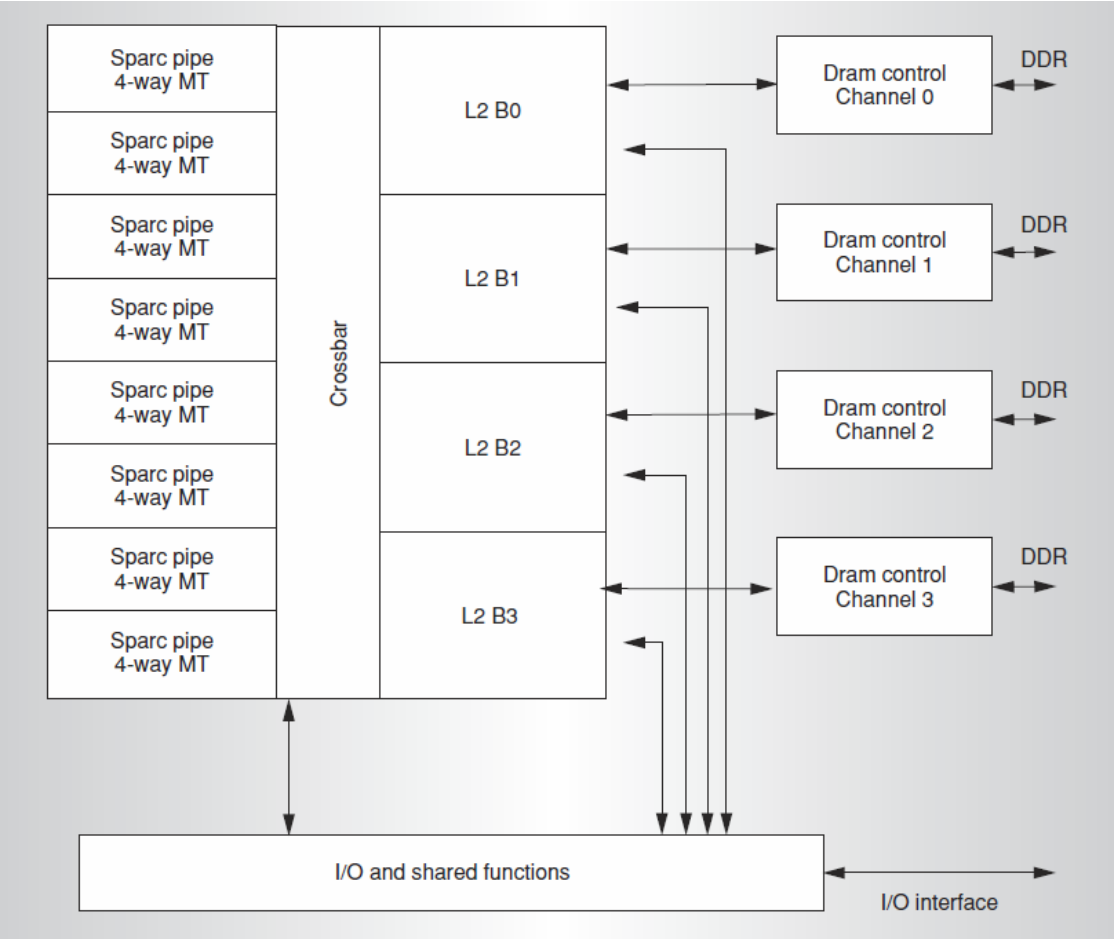


1.2 Ghz
90nm tech

8 * 4mt cores
single FPU
Crossbar in middle

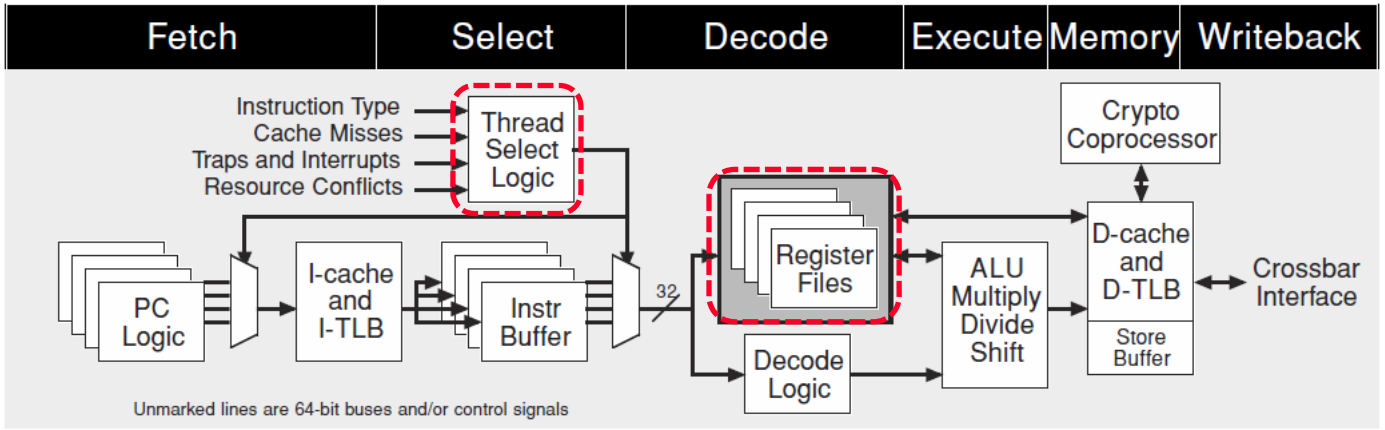
Niagara: CMP with multi-threading

- **Sun Niagara I (=UltraSPARC T1) : block diagram**



Niagara: CMP with multi-threading

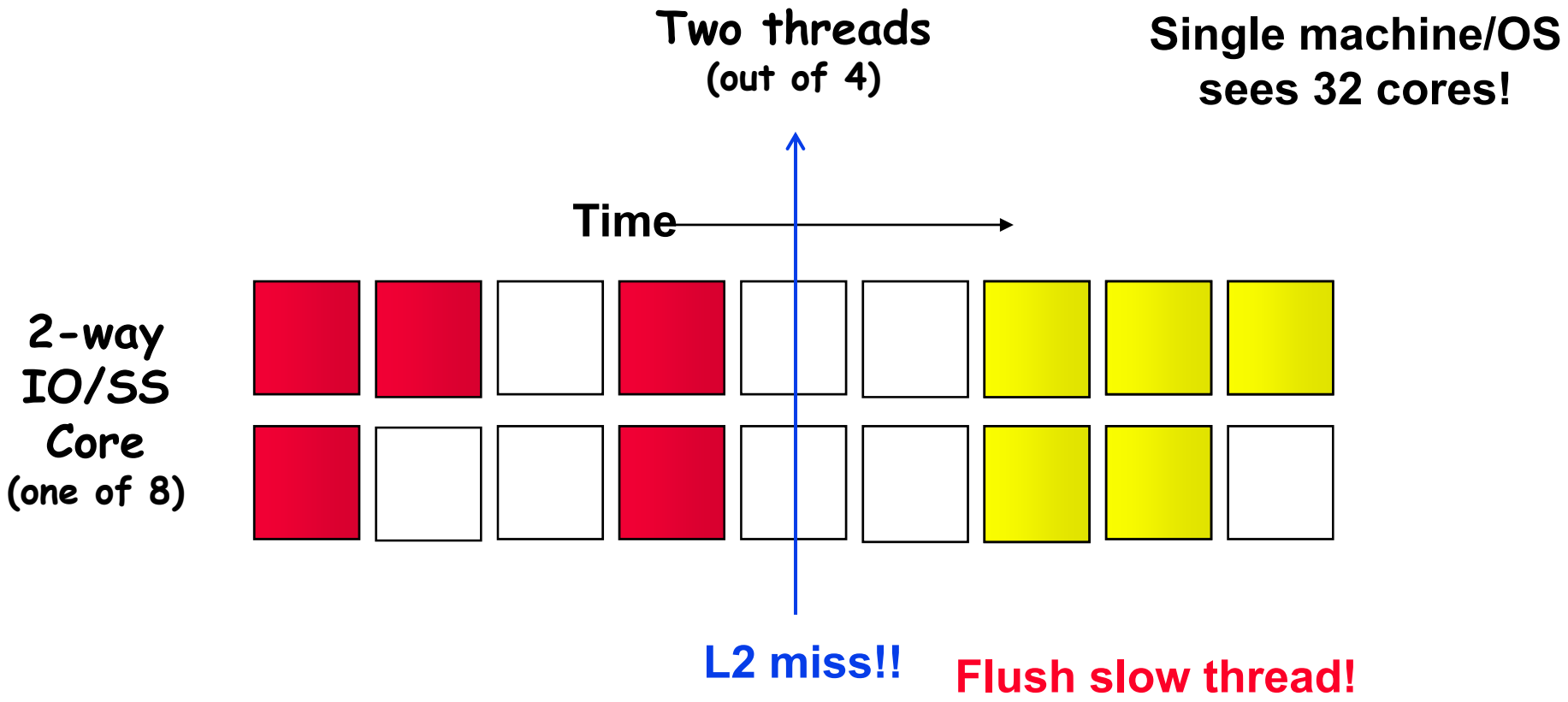
- **Sun Niagara I (=UltraSPARC T1)**
  - **Each "core" pipeline with 4-way in-order multi-threading**



F → S → D → E → M → W

- 6 shallow stages
- **No branch prediction**
- Select "least recently used" thread for fairness.
- Minimal speculation (e.g., only 'load hit' speculation)

Coarse-grain Multithreading (e.g., Niagara)



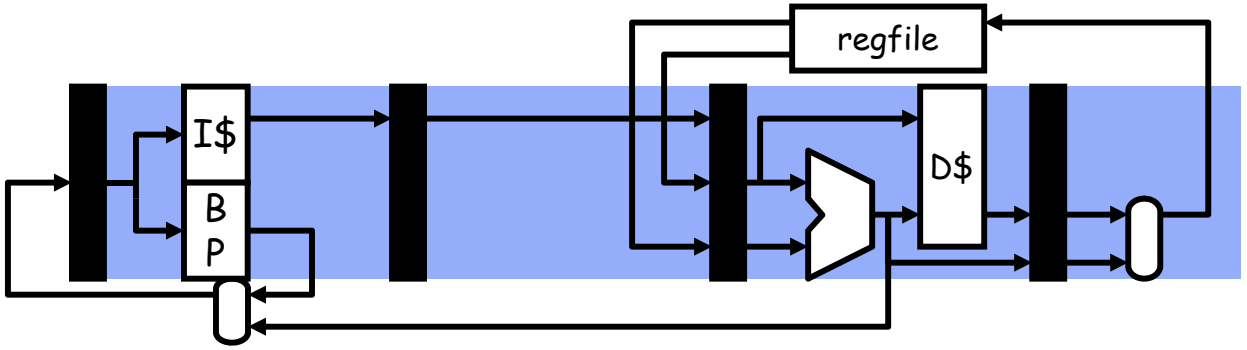
Preserves single-thread performance,
but can only hide very long latencies (i.e., L2/L3 misses)

Coarse-Grain Multithreading (CGMT)

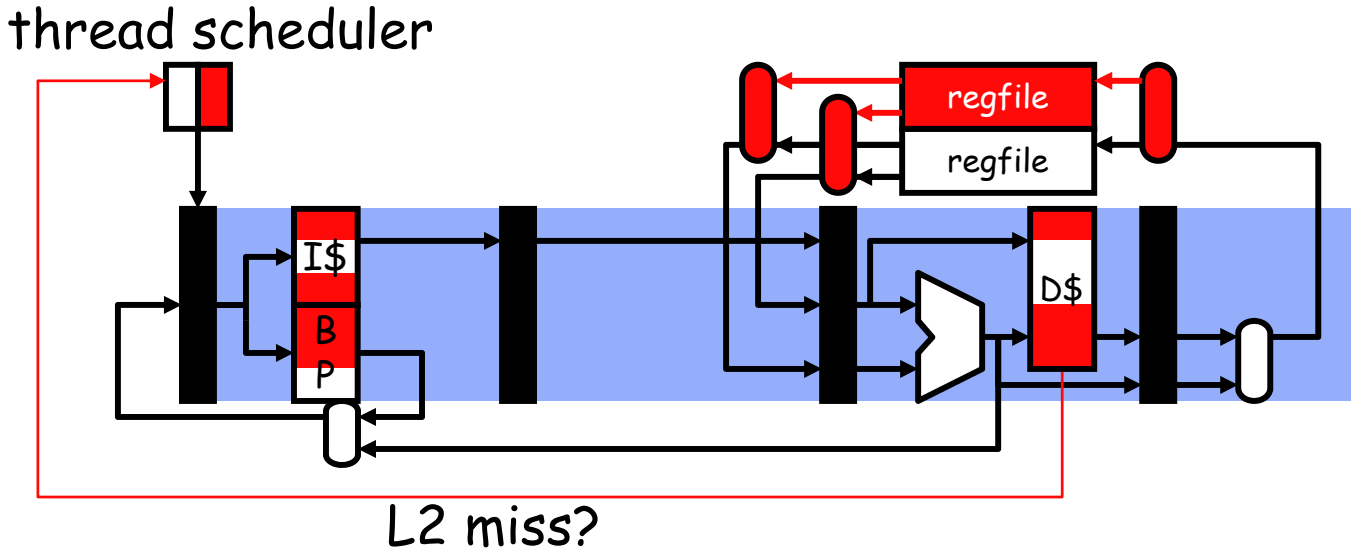
- **Coarse-Grain Multi-Threading (CGMT)**
  - \+ Sacrifices very little single thread performance (of one thread)
  - \- Tolerates only long latencies (e.g., L2 misses)
  - \- Thread scheduling policy
    - Designate a "preferred" thread (e.g., thread A)
    - Switch to thread B on thread A's L2 miss
    - Switch back to A when A's L2 miss returns
  - \- No pipeline partitioning
    - They flush on switch
      - **Can't tolerate latencies shorter than twice pipeline depth**
      - Need short in-order pipeline for good performance

CGMT

- Normal Superscalar

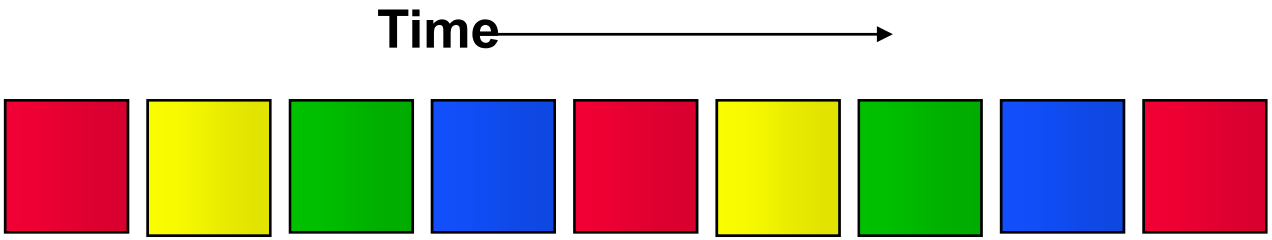


- CGMT

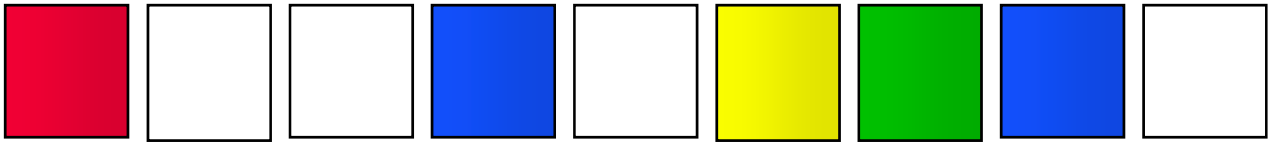


Fine Grained Multithreading

What if we switch threads very fast and frequently?



If saturated workload -> Lots of threads



But, if unsaturated workload -> Lots of stalls

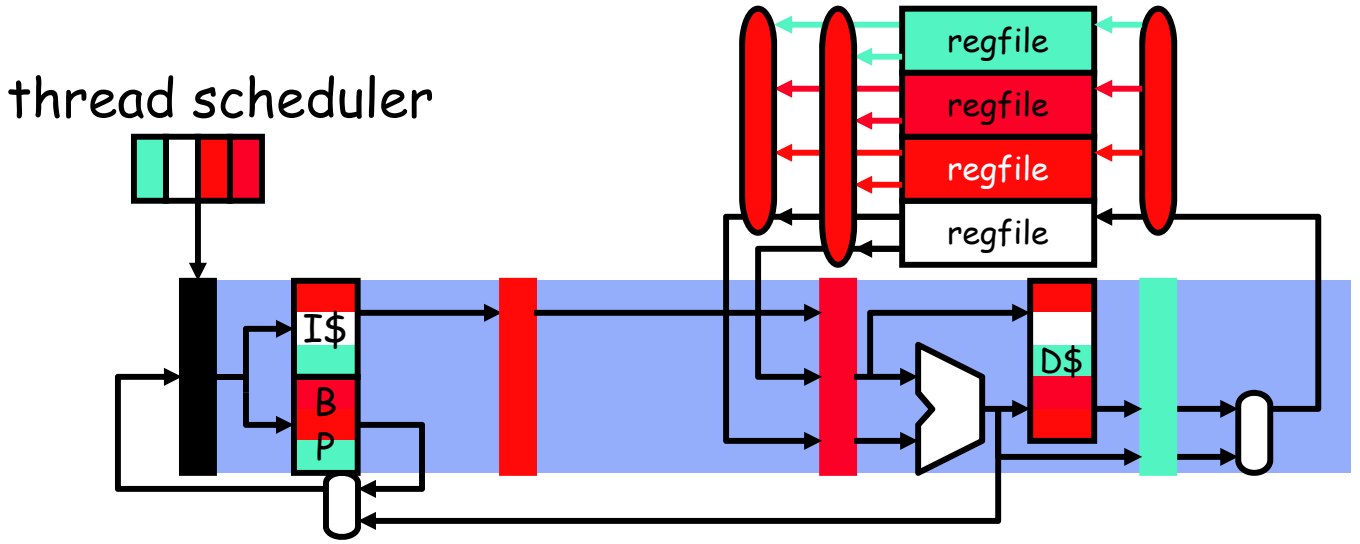
Intra-thread dependencies still limit performance

Fine-Grain Multithreading (FGMT)

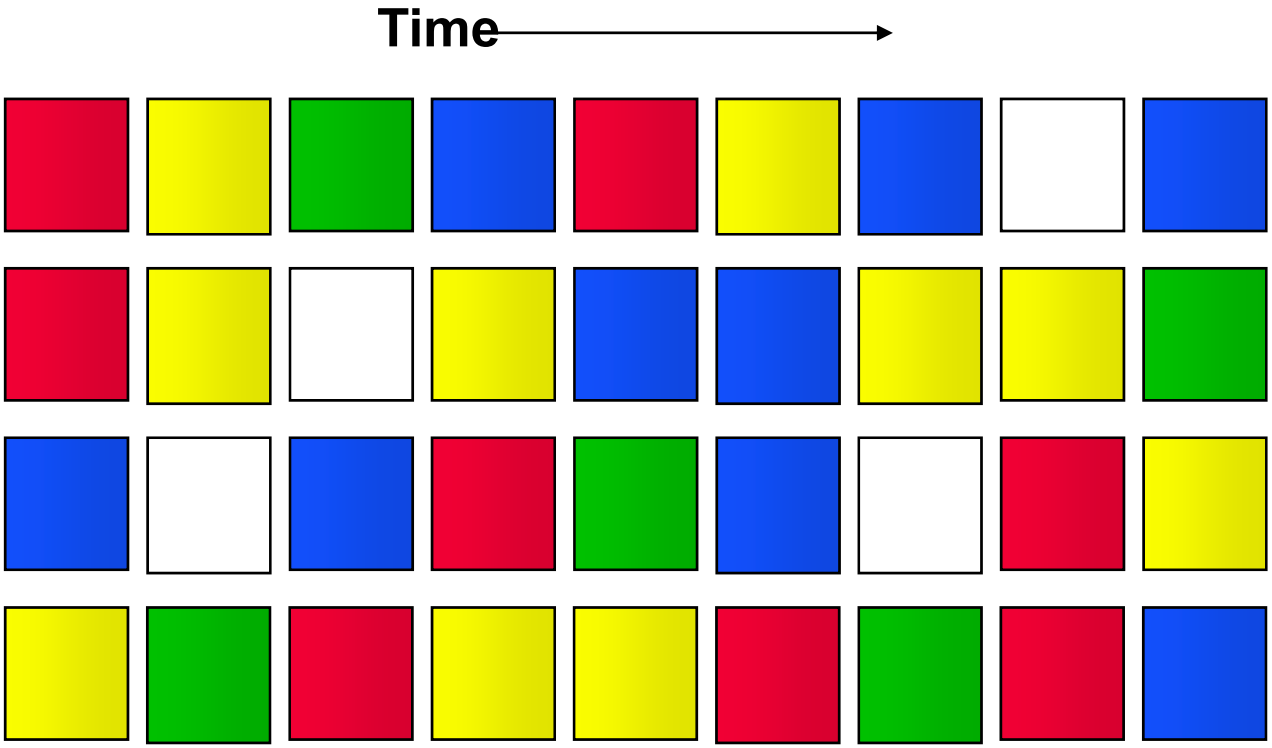
- **Fine-Grain Multithreading (FGMT)**
  - – Sacrifices significant single thread performance
  - \+ Tolerates all latencies (e.g., L2 misses, mispredicted branches, etc.)
  - \- Thread scheduling policy
    - Switch threads every cycle (round-robin), L2 miss or no
  - \- Pipeline partitioning
    - Dynamically partitioned, No flush on switch
    - Length of pipeline doesn't matter
  - – Need a lot of threads
  - – Not popular today
    - Many threads → many register files

Fine-Grain Multithreading

- FGMT
  - (Many) more threads
  - Multiple threads in pipeline at once



Simultaneous Multithreading (SMT)



Maximum utilization of function units by independent operations
- What problem in FGMT is solved by SMT?

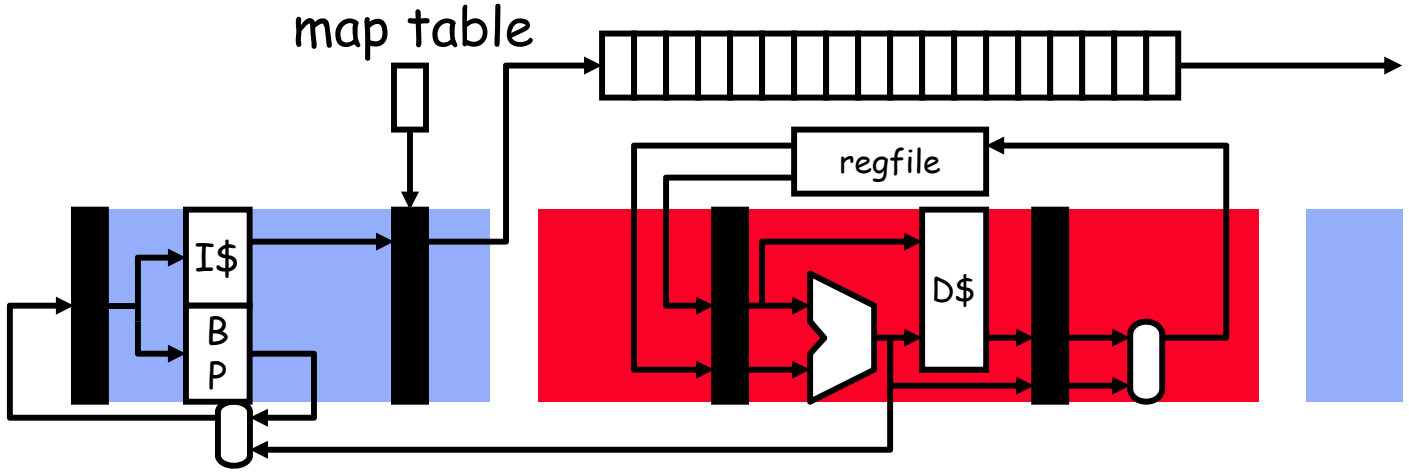
SMT is also known as "Hyperthreading"

Simultaneous Multithreading (SMT)

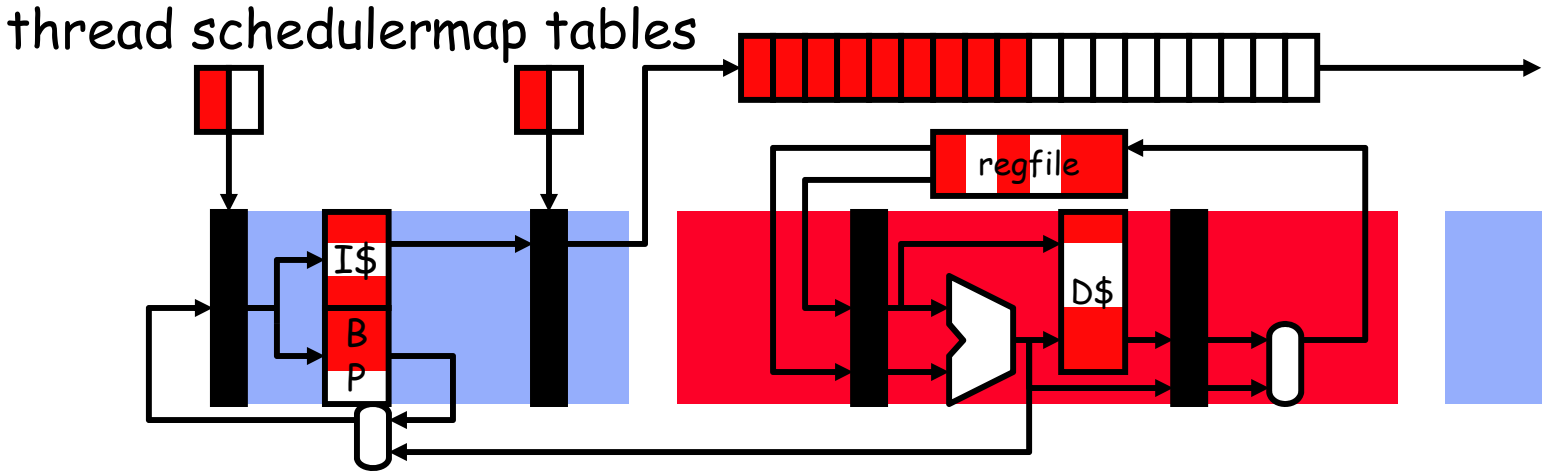
- Can we multithread an out-of-order machine?
  - Don't want to give up performance benefits
  - Don't want to give up natural tolerance of D$ (L1) miss latency
- **Simultaneous multithreading (SMT)**
  - \+ Tolerates all latencies (e.g., L2 misses, mispredicted branches)
  - ± Sacrifices some single thread performance due to resource sharing
  - Thread scheduling policy
    - Round-robin (just like FGMT) or something else?
  - Pipeline partitioning
    - Dynamic, but hard to be optimal
  - Example:
    - Pentium4 (hyper-threading): 5-way issue, 2 threads
    - Alpha 21464: 8-way issue, 4 threads (canceled)

Simultaneous Multithreading (SMT)

- Ordinary Superscalar



- SMT
  - Replicate map table, share physical register file



Issues for SMT

- Fetch
  - Like Niagara, can switch multiple PCs every cycle OR
  - Maybe few independent fetch (e.g., 2.8 fetch policy)

- Cache interference
  - General concern for all MT variants
  - Can the working sets of multiple threads fit in the caches?
  - Memory-sharing thread applications can help here
    - + Same instructions → share I$,
    - + Shared data → less D$ contention
    - • MT is good for "server" workloads (e.g., database, web server)
  - To keep miss rates low, SMT might need a larger L2 (which is OK)
    - • Out-of-order tolerates L1 misses

- Large map table and physical register file
  - #phys-regs = (**#threads** * #arch-regs) + #in-flight insns

SMT Resource Partitioning

- How are ROB/LSQ/ISQ partitioned in SMT?
  - Depends on what you want to achieve
- **Static partitioning**
  - Divide ROB/LSQ/ISQ into T static equal-sized partitions
  + Ensures that low-IPC threads don't starve high-IPC ones
    - Low-IPC threads stall and occupy ROB/LSQ/ISQ slots
  – Low utilization
- **Dynamic partitioning**
  - Divide ROB/LSQ/ISQ into dynamically resizing partitions
  - Let threads fight for amongst themselves
  + High utilization
  - Possible starvation
  - Which thread to fetch every cycle?

  e.g., "ICOUNT" : fetch a thread having fewest in-flight insts.

SMT vs. CMP

- If you wanted to run multiple threads would you build a…
  - Chip multiprocessor (CMP): multiple separate pipelines?
  - A multithreaded processor (SMT): a single larger pipeline?
- **Both will get you throughput on multiple threads**
  - CMP will be simpler, possibly faster clock
  - SMT will get you better performance (IPC) on a single thread
- **Let's use both SMT and CMP on a chip!**
  - Now almost every modern CPU does this
    - Intel, AMD, IBM, Sun (N1/N2/N3),…
  - Oracle (ex-SUN) does more

    e.g., Yosemite Fall (T4/N4)

    : 8 OoO cores with 2-way SMT, each with 8 FGMT threads

    8 cores * 8 mts = 64 threads on-chip with 16 thread issue a cycle

State of the Art

	AMD Opteron 8360SE	Intel Xeon X7460	Intel Itanium 9050	IBM P5	IBM P6	Fijitsu SPARC 7	SUN T2
cores/threads	4x1	6x1 OoO SMT	2x2 VLIW	2x2	2x2 OoO SMT	4x2 OoO SMT	8x8 IO FGMT
Clock (GHz)	2.5	2.67	1.60	2.2	5	2.52	1.8
Issue Rate	3 (x86)	4 (rop)	6	5	7	4	2
Pipeline depth	12/17	14	8	15	13	15	8/12
Out-of-order	72(rop)	96(rop)	inorder	200	limited	64	inorder
on-chip$ (MB)	2+2	9+16	1+12	1.92	8	6	4
Trans (10^6)	463	1900	1720	276	790	600	503
Power (W)	105	130	104	100	>100	135	95
SPECint 2006 per-core/total	14.4/170	22/274	14.5/1534	10.5/197	15.8/1837	10.5/2088	--/142
SPECfp 2006 per-core/total	18.5/156	22/142	17.3/1671	12.9/229	20.1/1822	25.0/1861	--/111

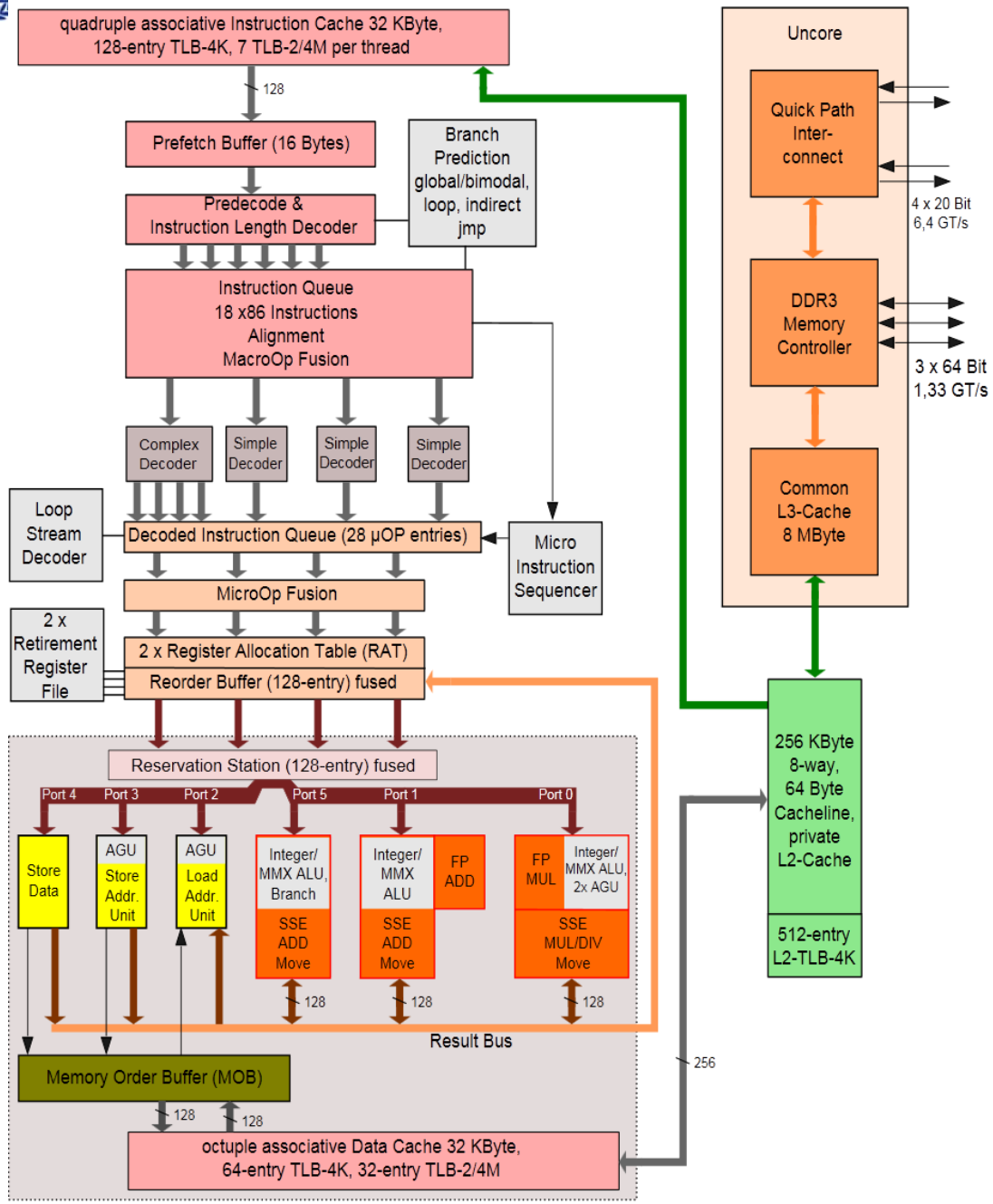
Multiple chips per system (scalability)

Microprocessor Report, Oct 2008

History of Intel CPUs

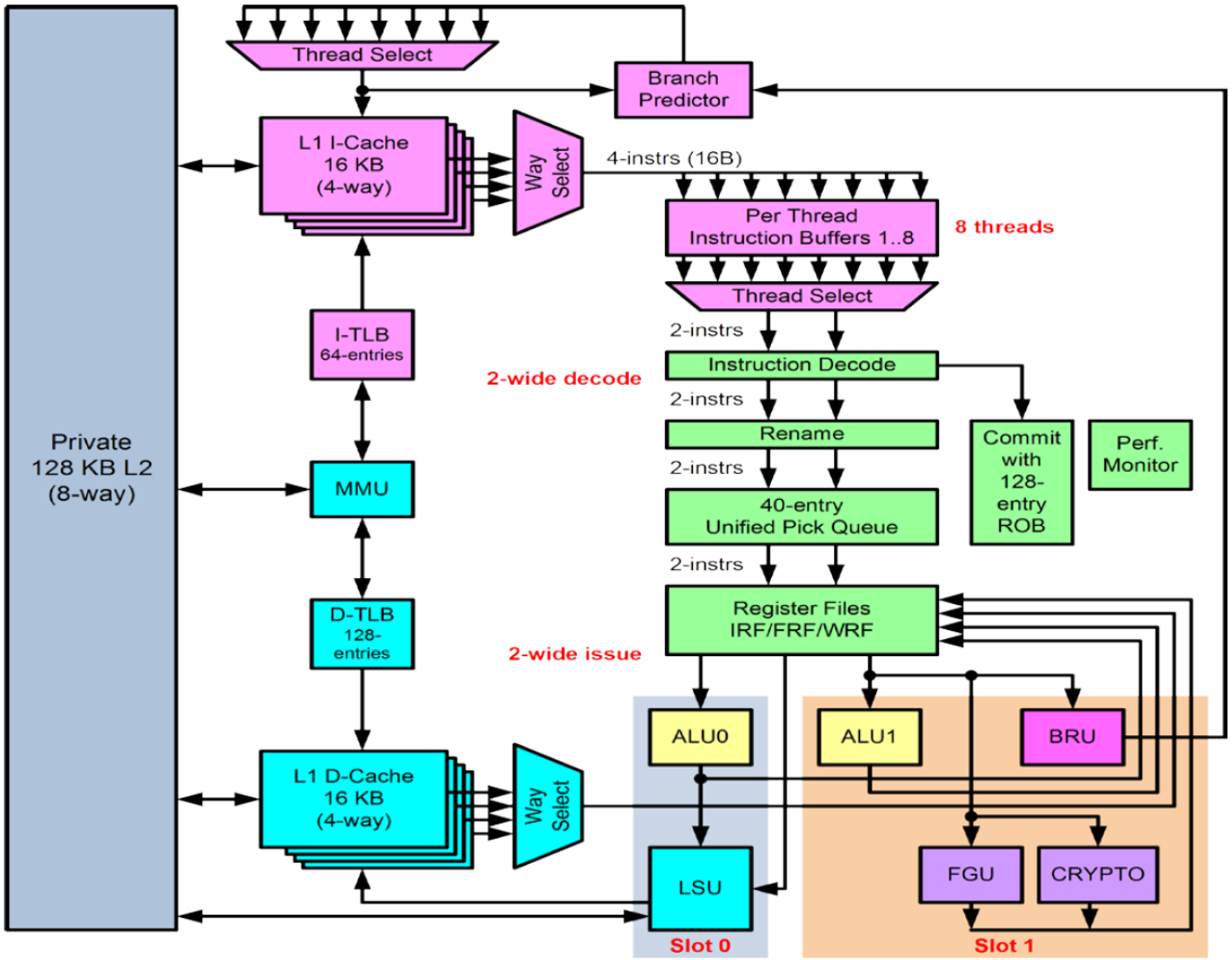
- Intel uArchitecture : # of stages
  - P5 (Pentium)
    - 5 stages, 2-way **in-order superscalar**
  - P6 (Pentium 3, Pentium Pro)
    - 10~14 stages, superpipelining, **OoO superscalar**
  - Netburst (Pentium 4)
    - 20~31 stages, **aggressive OoO superscalar**, **SMT**
  - Core (Pentium M)
    - 14 stages (this is really P6) + **multi-core**
  - Bonnell (ATOM)
    - 16 stages, 2-way in-order superscalar for **low power**
  - Nehalem (i3, i5, i7) & Sandy Bridge
    - up to 8 cores, basically every advanced techniques above.

Intel Nehalem Microarchitecture



- Up to 8 cores
- 2 threads / core
- 2-way Superscalar
- Out-of-Order
- 128-entry ISQ
- 128-entry ROB
- Branch prediction
- One L1 cache / core
- ..

Oracle/Sun Sparc T4 Microarchitecture



- 8 cores on chip
- 8 threads / core
- 2-way superscalar
- Out-of-Order
- 40-entry ISQ
- 128-entry ROB
- Branch prediction
- One L1D / 2 cores
- ..

READ!!

- "The Microarchitecture of Superscalar Processors" by Smith, Proceedings of IEEE 1995.
- "Simultaneous Multithreading: A Platform for Next-Generation Processors" by Eggers, et al., IEEE Micro 1997
- "Niagara: A 32-Way Multithreaded SPARC Processor" by Kongetira, et al. IEEE Micro 2005

Question?

Announcements:

Reading: Start reading Ch. 5 (P&H)

Handouts: None