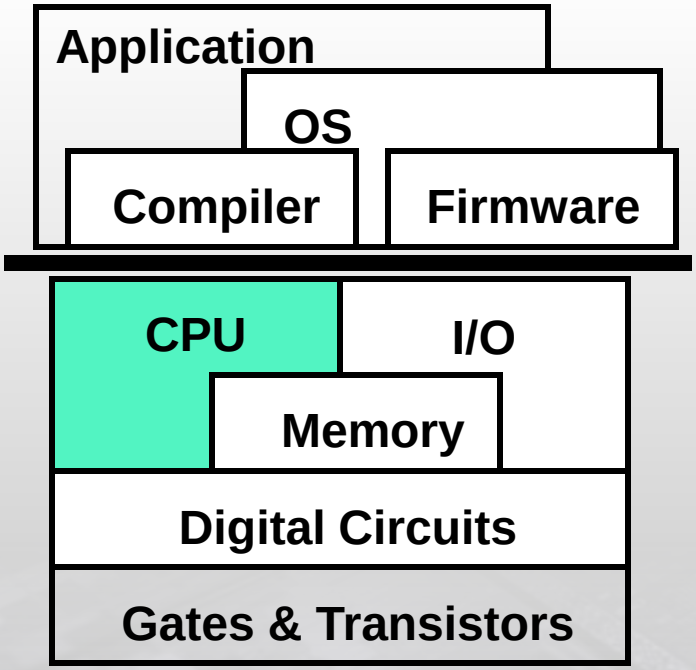
Thread Level Parallelism II: Multithreading

Readings:

H&P: Chapter 3.5

Paper: NIAGARA: A 32-WAY MULTITHREADED …

This Unit: Multithreading (MT)



- Why multithreading (MT)?
  - Utilization vs. performance
- Three implementations
  - Coarse-grained MT
  - Fine-grained MT
  - Simultaneous MT (SMT)
- Example
  - The Sun UltraSparc T1
- Research topics

Performance And Utilization

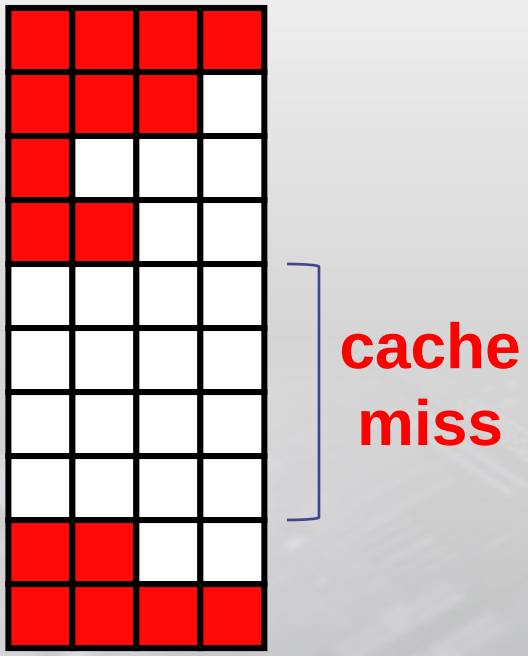
- Performance (IPC) important
- Utilization (actual IPC / peak IPC) important too

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

- **Multi-threading (MT)**
  - Improve utilization by multi-plexing multiple threads on single CPU
  - One thread cannot fully utilize CPU? Maybe 2, 4 (or 100) can
  - Multi-programmed OS generalization:
    - I/O→CPU Stall
    - Context Switches → Thread Switches
    - Stack of software → Zero software

Superscalar Under-utilization

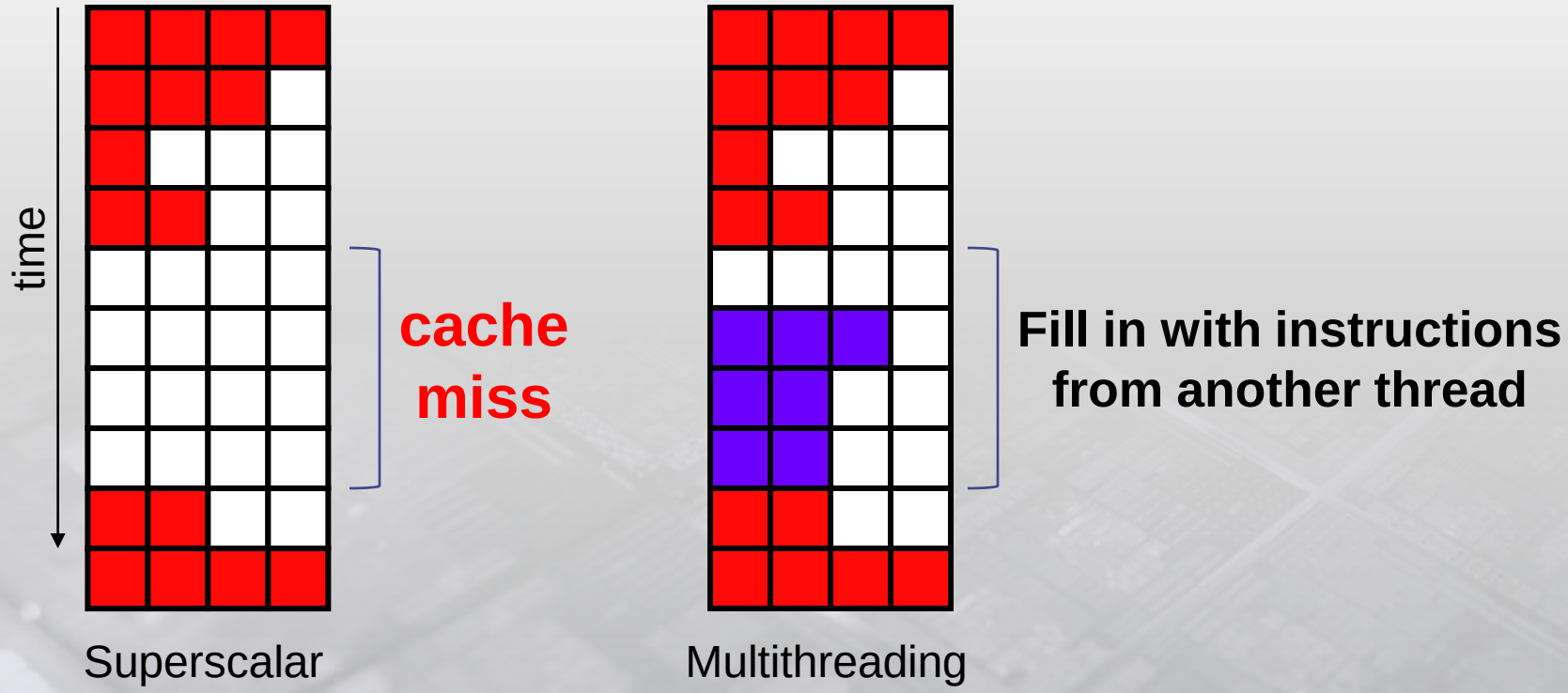
- Time evolution of issue slot
  - 4-issue processor



Superscalar

Simple Multithreading

- Time evolution of issue slot
  - 4-issue processor



- Where does it find a thread? Same problem as multi-core
  - Same shared-memory abstraction

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-wide machine)
  - Server workloads have TLP (can use multiple threads)

MT Implementations: Similarities

- How do multiple threads share a single processor?
  - Different sharing mechanisms for different kinds of structures
  - Depend on what kind of state structure stores

- **No state**: ALUs
  - Dynamically shared
- **Persistent hard state (aka "context")**: PC, registers
  - Replicated
- **Persistent soft state**: caches, bpred
  - Dynamically partitioned (like on a multi-programmed uni-processor)
    - TLBs need thread ids, caches/bpred tables don't (PA/VA???)
  - Exception: **ordered "soft" state** (BHR, RAS) is replicated
- **Transient state**: pipeline latches, ROB, RS
  - Partitioned … somehow

MT Implementations: Differences

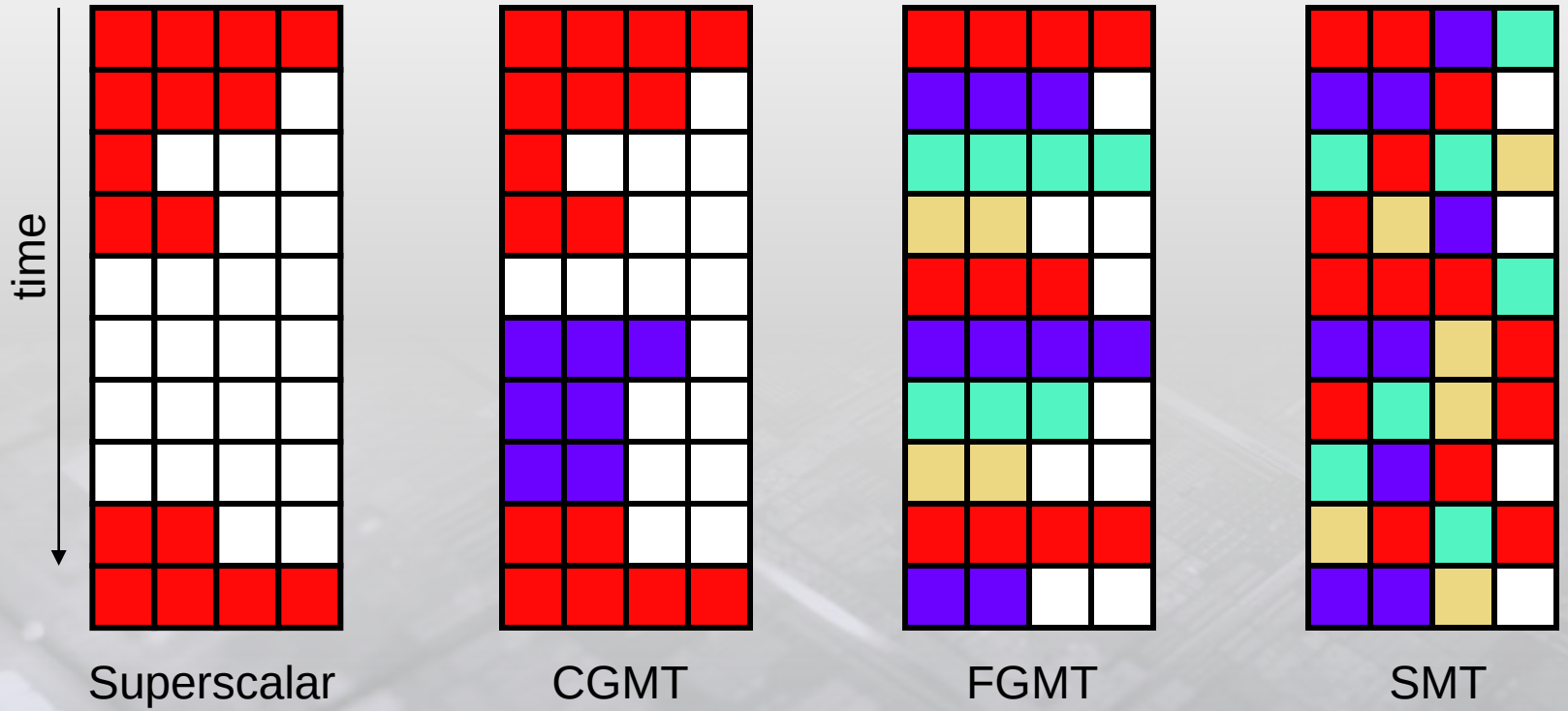
- Main question: **thread scheduling policy**
  - When to switch from one thread to another?
- Related question: **pipeline partitioning**
  - How exactly do threads share the pipeline itself?

- Choice depends on
  - What kind of latencies (specifically, length) you want to tolerate
  - How much single thread performance you are willing to sacrifice

- Three designs
  - Coarse-grain multithreading (CGMT)
  - Fine-grain multithreading (FGMT)
  - Simultaneous multithreading (SMT)

The Standard Multithreading Picture

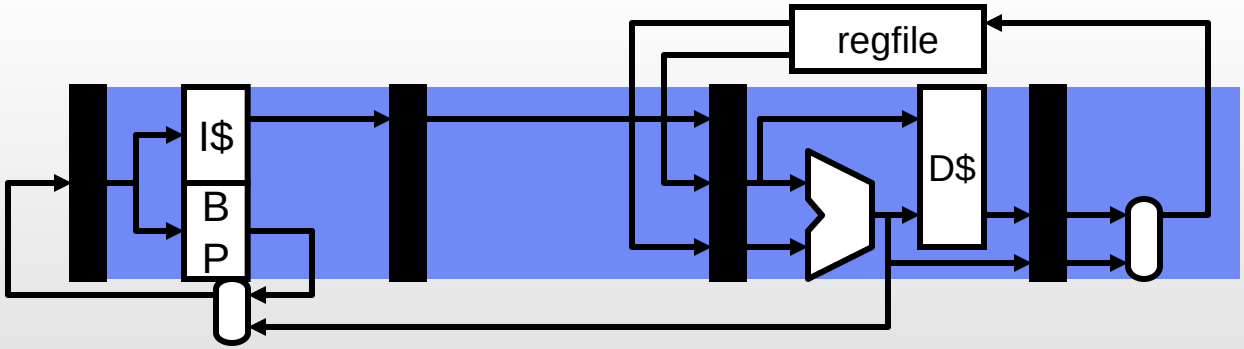
- Time evolution of issue slots
  - Color = thread



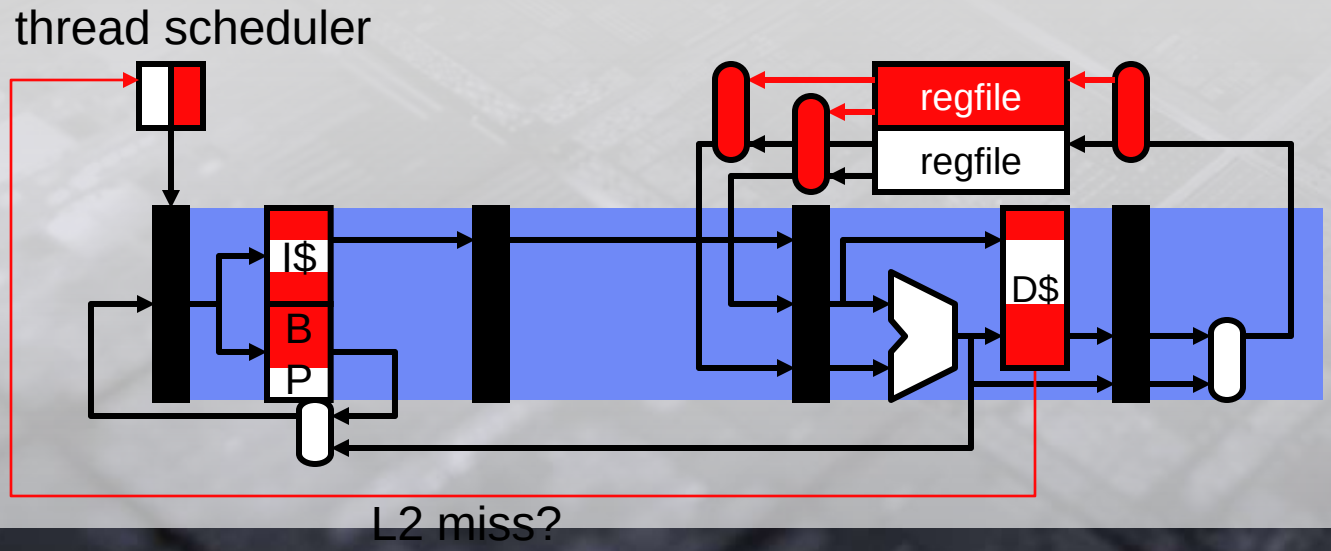
Coarse-Grain Multithreading (CGMT)

- **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 L2 miss
    - Switch back to A when A L2 miss returns
  - Pipeline partitioning
    - None, flush on switch
    - – Can't tolerate latencies shorter than twice pipeline depth
    - Need short in-order pipeline for good performance
  - Example: IBM Northstar/Pulsar (RS/6000)

CGMT


regfile
I$
B
P
D$

- CGMT

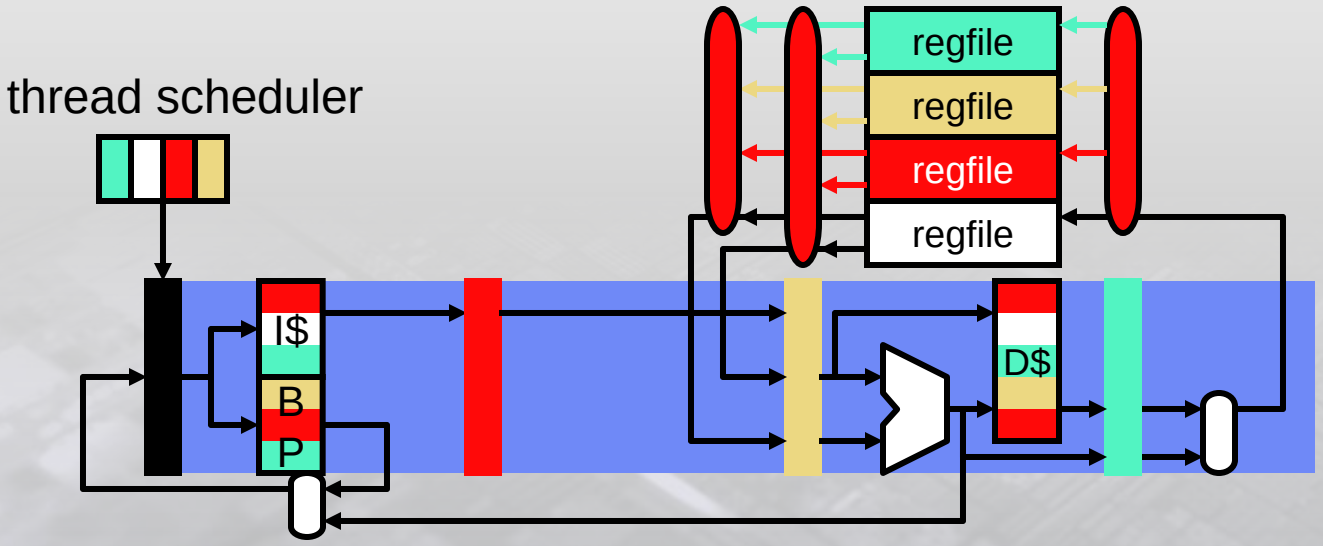

thread scheduler
regfile
regfile
I$
B
P
D$
L2 miss?

Fine-Grain Multithreading (FGMT)

- **Fine-Grain Multithreading (FGMT)**
  - – Sacrifices significant single thread performance
  - \+ Tolerates latencies (e.g., L2 misses, mispredicted branches, etc.)
  - Thread scheduling policy
    - Switch threads every cycle (round-robin), L2 miss or no
  - Pipeline partitioning
    - Dynamic, no flushing
    - Length of pipeline doesn't matter so much
  - – Need a lot of threads
  - Extreme example: Denelcor HEP ('81-'85)
    - So many threads (100+), it didn't even need caches
    - Failed commercially
  - Not popular today
    - Many threads → many register files

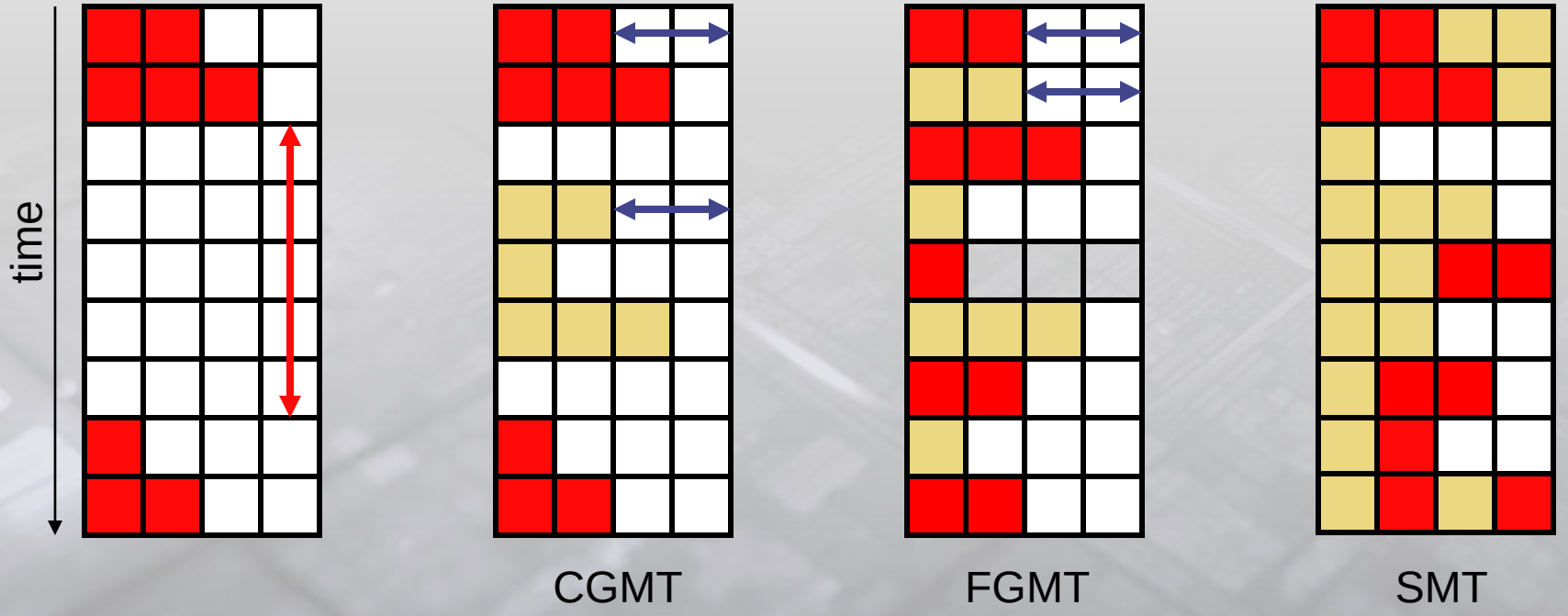
Fine-Grain Multithreading

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



Vertical and Horizontal Under-Utilization

- FGMT and CGMT reduce **vertical under-utilization**
  - Loss of all slots in an issue cycle
- Do not help with **horizontal under-utilization**
  - Loss of some slots in an issue cycle (in a superscalar processor)



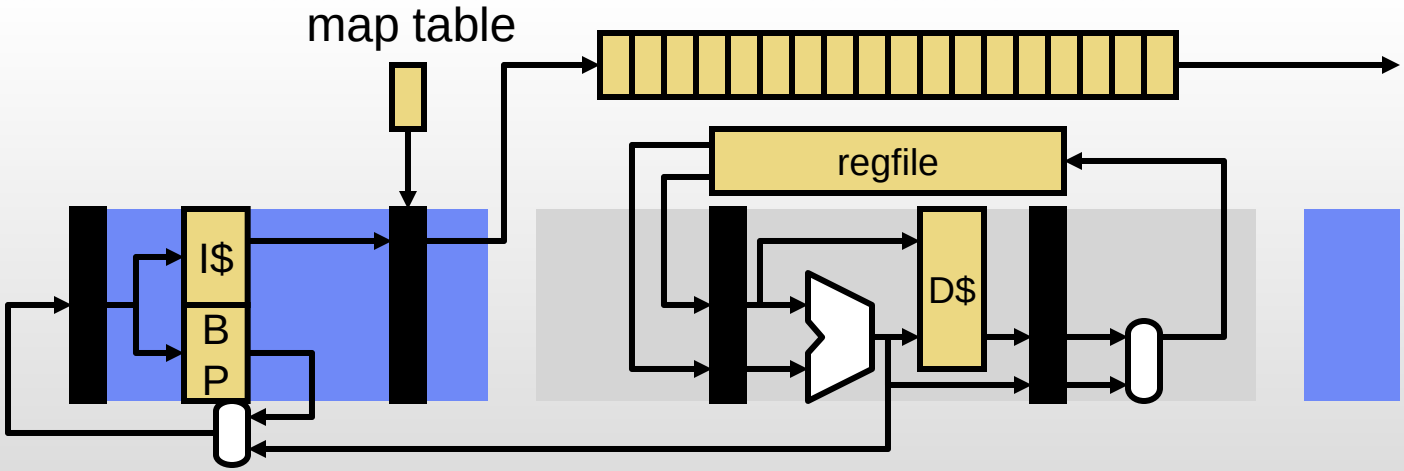
SMT

Simultaneous Multithreading (SMT)

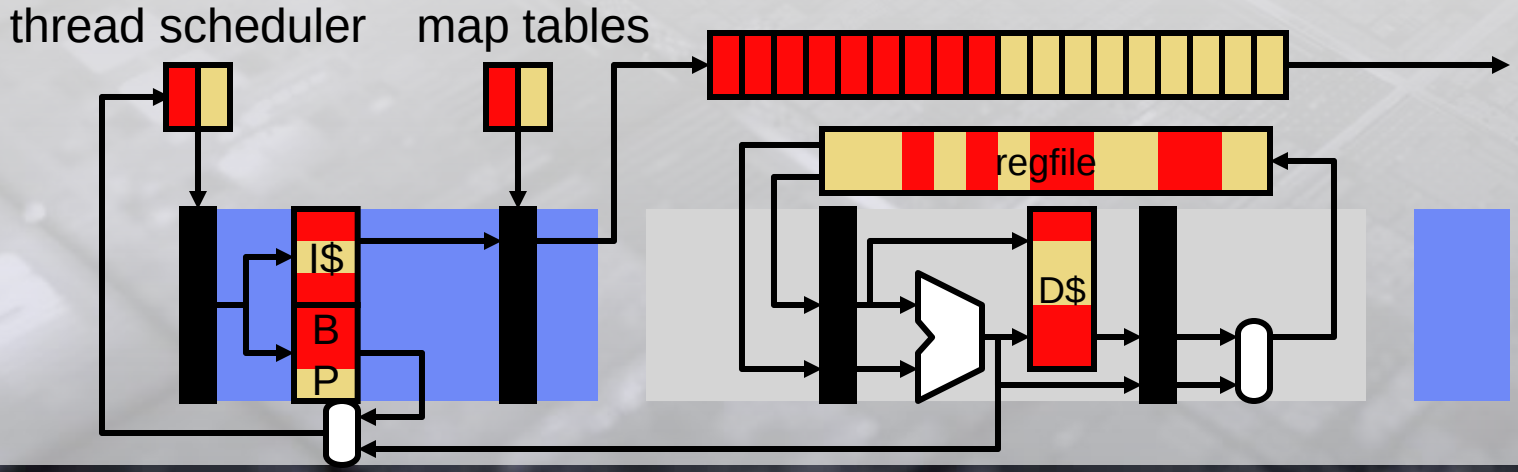
- What can issue insns from multiple threads in one cycle?
  - Same thing that issues insns from multiple parts of same program…
  - …out-of-order execution
  - *How different PCs are managed?*

- **Simultaneous multithreading (SMT)**: OOO + FGMT
  - Aka **"hyper-threading"**
  - Observation: once insns are renamed, scheduler doesn't care which thread they come from (well, for non-loads at least)
  - Some examples
    - IBM Power5: 4-way issue, 2 threads
    - Intel Pentium4: 3-way issue, 2 threads
    - Intel "Nehalem": 4-way issue, 2 threads
    - Alpha 21464: 8-way issue, 4 threads (canceled)
    - Notice a pattern? #threads (T) * 2 = #issue width (N)

Simultaneous Multithreading (SMT)

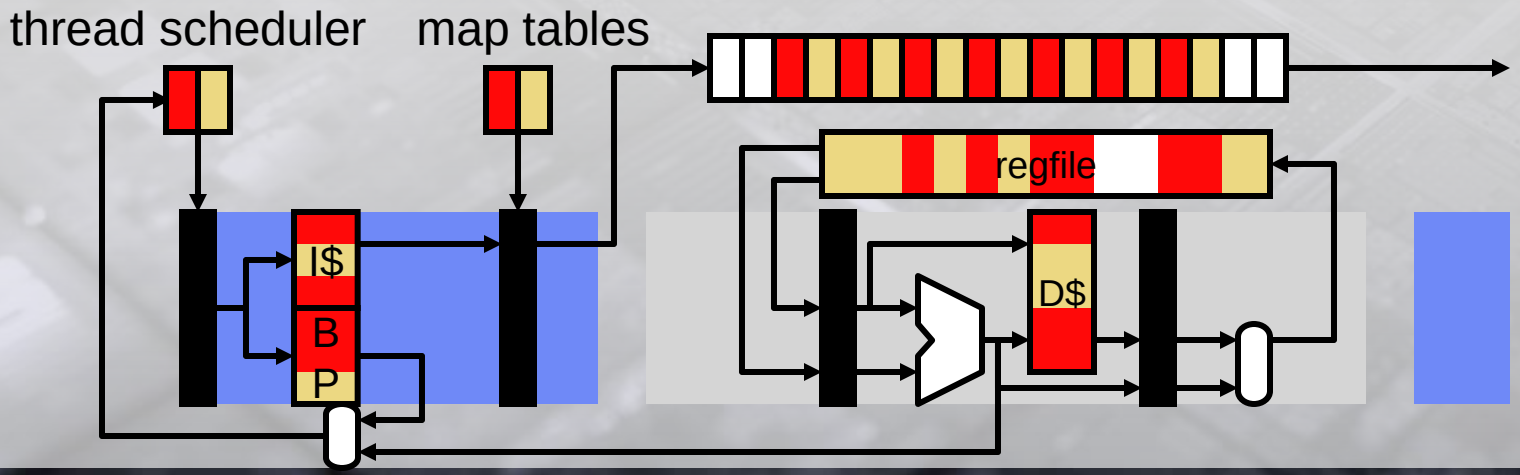


- SMT
  - Replicate map table, share (larger) physical register file



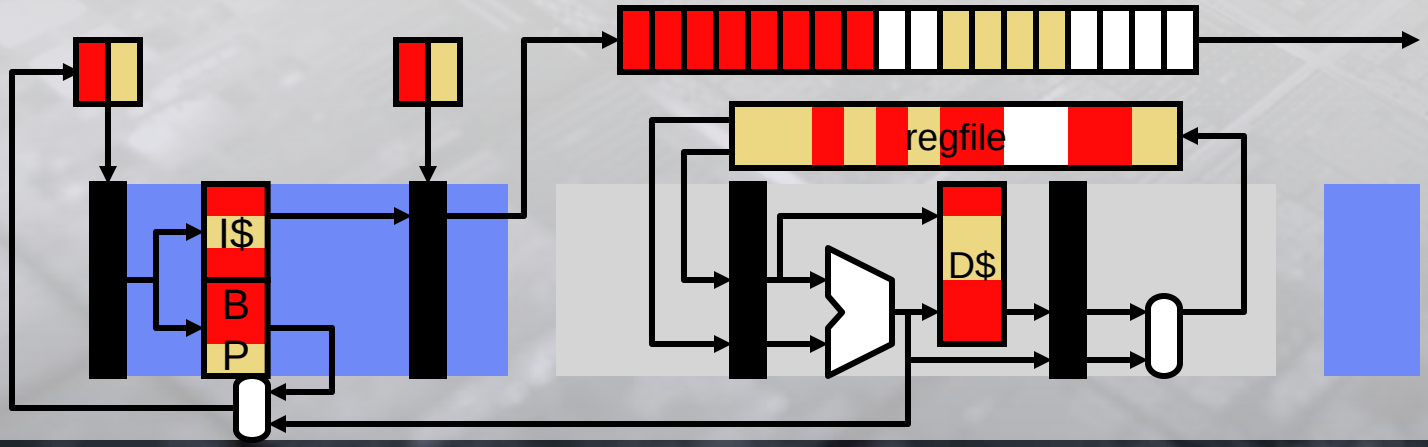
SMT Resource Partitioning

- Physical regfile and insn buffer entries shared at fine-grain
  - Physically unordered and so fine-grain sharing is possible
- How are physically ordered structures (ROB/LSQ) shared?
  - Fine-grain sharing (below) would entangle retire (and flush)
  - Allowing threads to commit independently is important



Static & Dynamic Resource Partitioning

- **Static partitioning** (below)
  - T equal-sized contiguous partitions
  - ± No starvation, sub-optimal utilization (fragmentation)
- **Dynamic partitioning**
  - P > T partitions, available partitions assigned on need basis
  - ± Better utilization, possible starvation
  - ICOUNT: fetch policy prefers thread with fewest in-flight insns
- Couple both with larger ROBs/LSQs



Multithreading Issues

- Shared soft state (caches, branch predictors, TLBs, etc.)
- Key example: **cache interference**
  - General concern for all MT variants
  - Can the working sets of multiple threads fit in the caches?
  - Shared memory SPMD threads help here
    - + Same insns → share I$
    - + Shared data → less D$ contention
    - MT is good for workloads with shared insn/data
  - To keep miss rates low, SMT might need a larger L2 (which is OK)
    - Out-of-order tolerates L1 misses

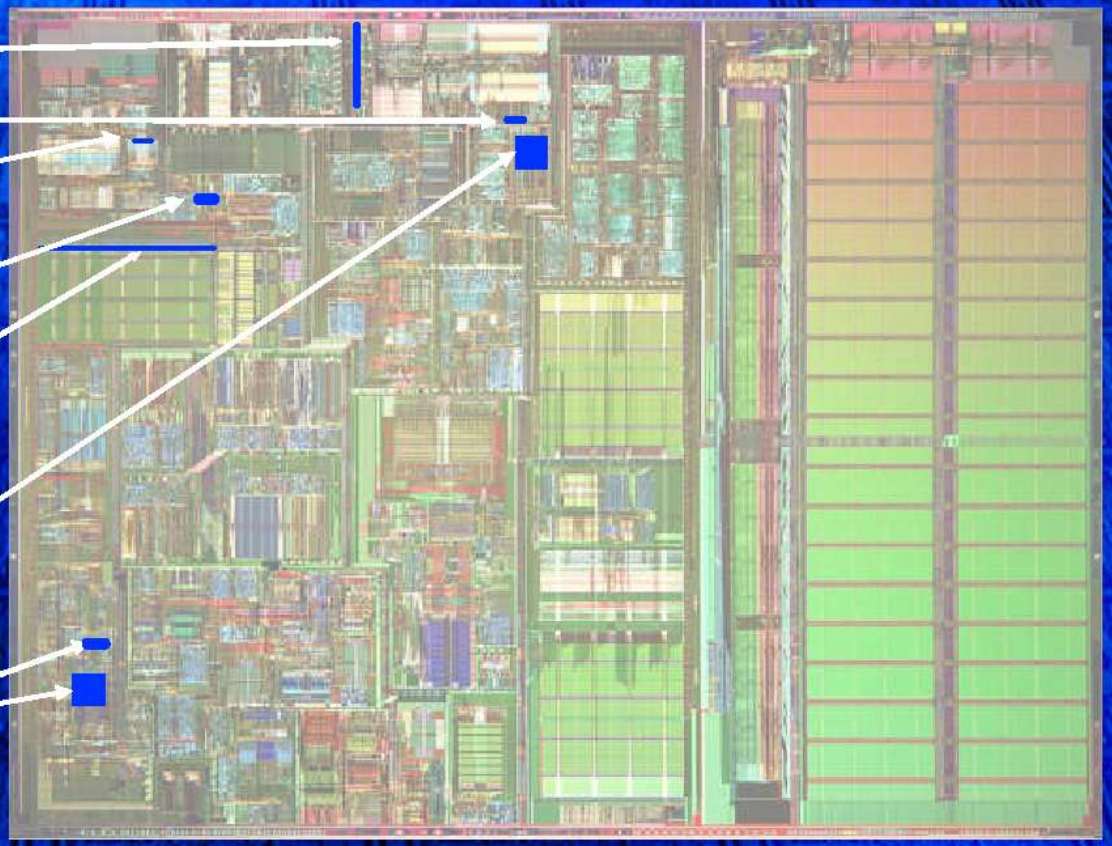
- Large physical register file (and map table)
  - physical registers = (**#threads** * #arch-regs) + #in-flight insns
  - map table entries = (**#threads** * #arch-regs)

Notes About Sharing Soft State

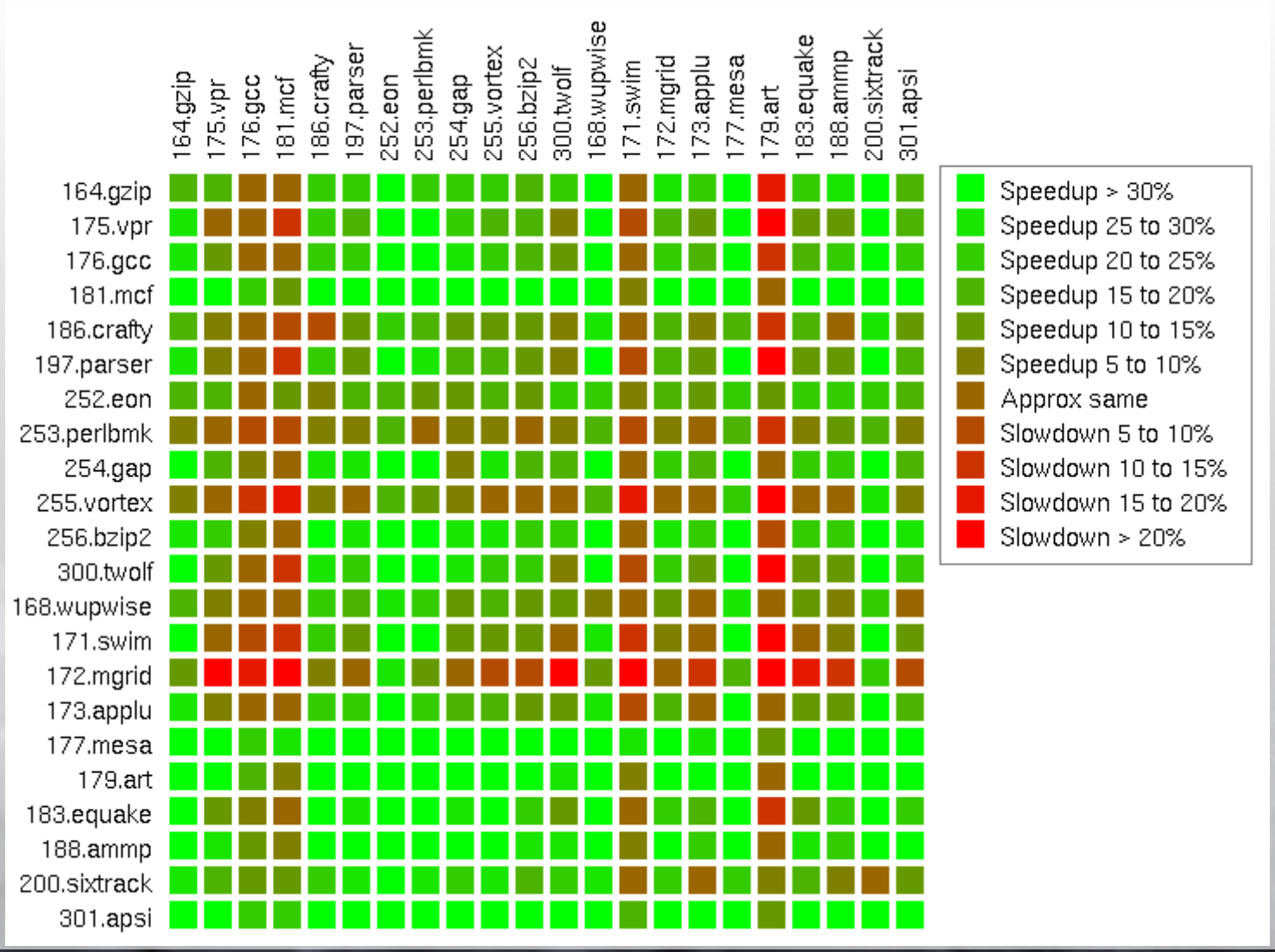
- Caches are shared naturally…
  - Physically-tagged: address translation distinguishes different threads
- … but TLBs need explicit thread IDs to be shared
  - Virtually-tagged: entries of different threads indistinguishable
  - Thread IDs are only a few bits: enough to identify on-chip contexts
- Thread IDs make sense on BTB (branch target buffer)
  - BTB entries are already large, a few extra bits / entry won't matter
  - Different thread's target prediction → automatic mis-prediction
- … but not on a BHT (branch history table)
  - BHT entries are small, a few extra bits / entry is huge overhead
  - Different thread's direction prediction → mis-prediction not automatic
- Ordered soft-state should be replicated
  - Examples: Branch History Register (BHR), Return Address Stack (RAS)
  - Otherwise it becomes meaningless… Fortunately, it is typically small

The real cost of SMT

HyperThreading Technology: What was added?



Not always a good thing



Multithreading vs. Multicore

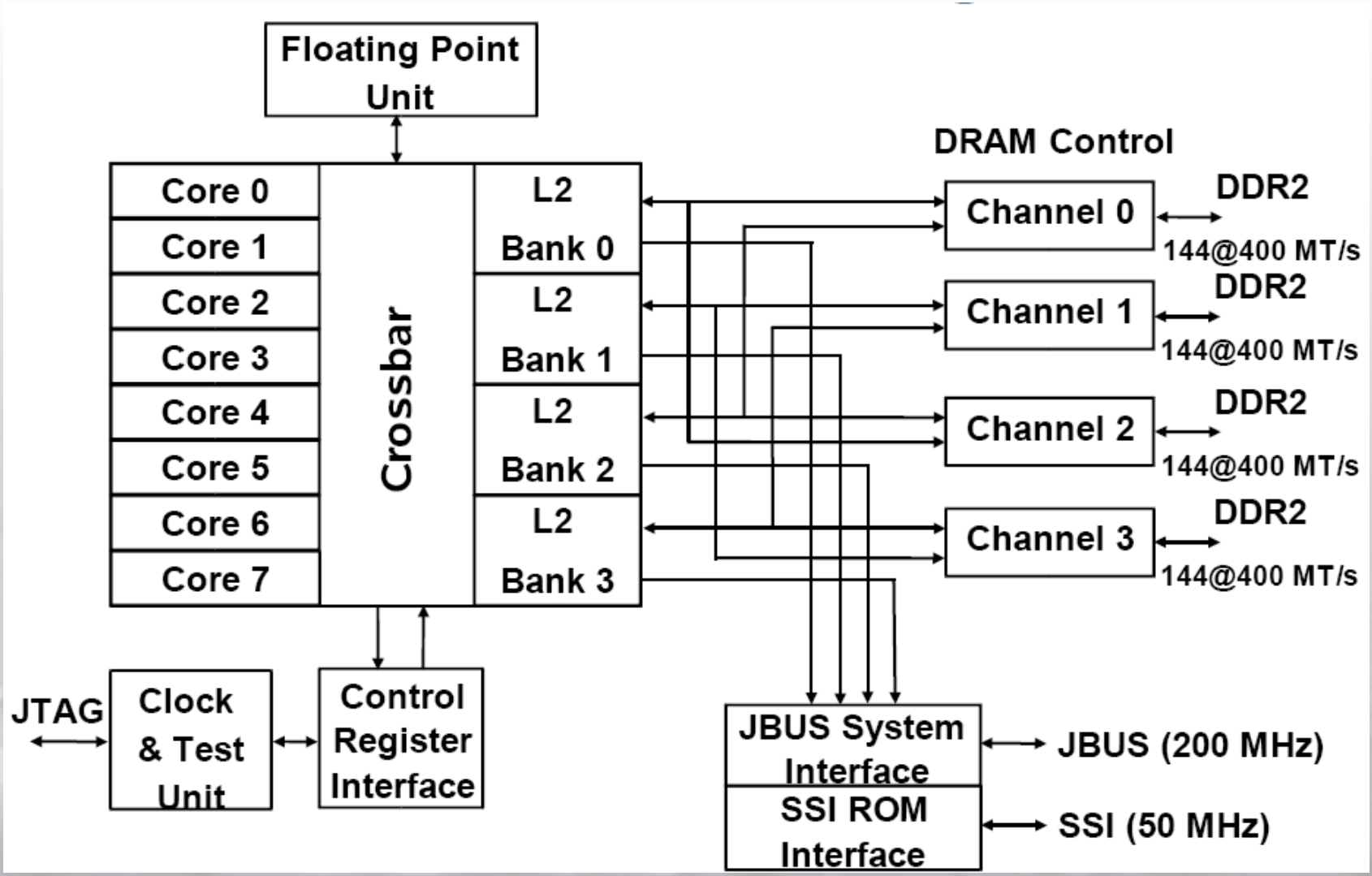
- If you wanted to run multiple threads would you build a…
  - A multicore: multiple separate pipelines?
  - A multithreaded processor: a single larger pipeline?
- **Both will get you throughput on multiple threads**
  - Multicore core will be simpler, possibly faster clock
  - SMT will get you better performance (IPC) on a single thread
    - SMT is basically an ILP engine that converts TLP to ILP
    - Multicore is mainly a TLP (thread-level parallelism) engine
- **Do both**
  - Sun's Niagara (UltraSPARC T1), ***Chip Multiprocessor***
  - 8 processors, each with 4-threads (non-SMT threading)
  - 1Ghz clock, in-order, short pipeline (6 stages or so)
  - Designed for power-efficient "throughput computing"

Multithreading Summary

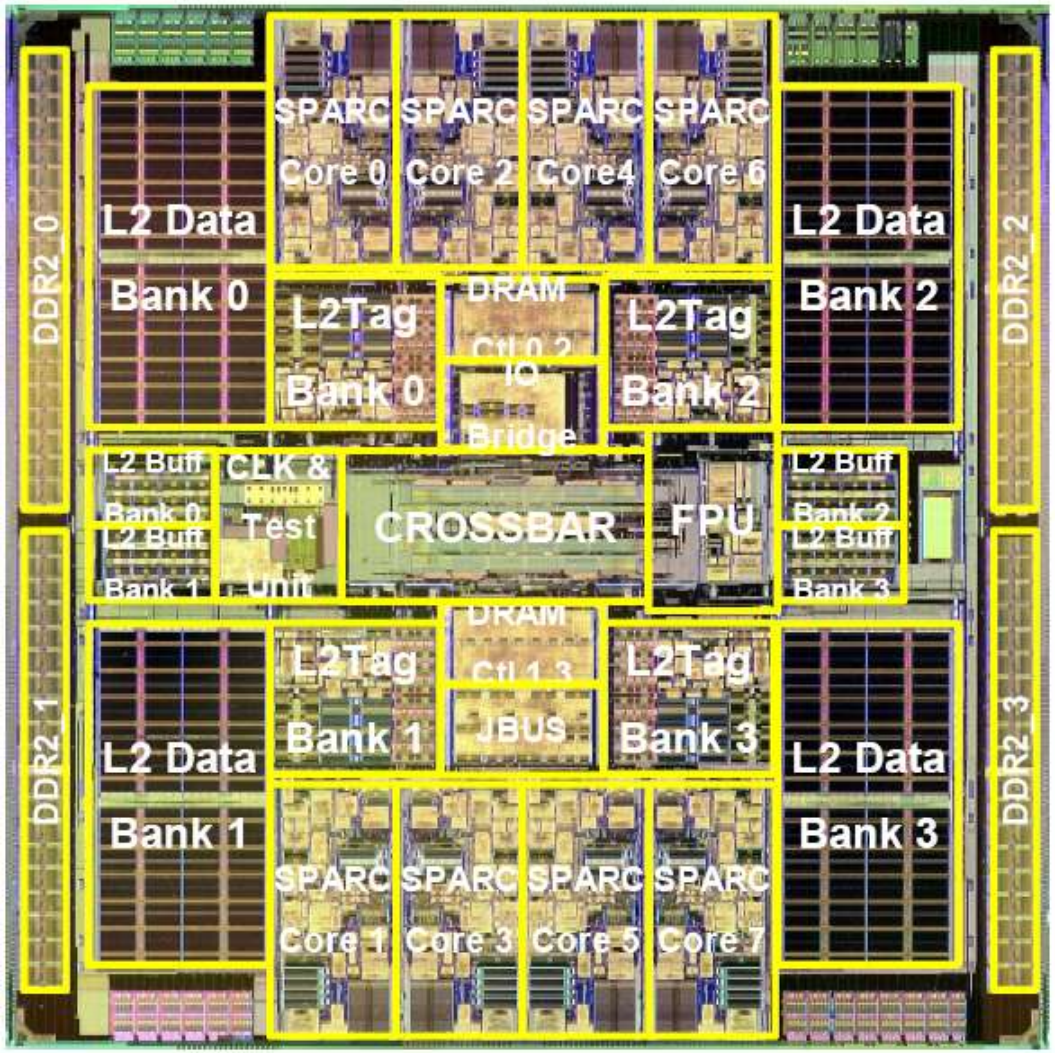
- Latency vs. throughput
- Partitioning different processor resources
- Three multithreading variants
  - Coarse-grain: no single-thread degradation, but long latencies only
  - Fine-grain: other end of the trade-off
  - Simultaneous: fine-grain with out-of-order
- Multithreading vs. chip multiprocessing

Example: The Sun UnltraSparc T1 Multiprocessor

The SUN UltraSparc T1



Floor plan



Features:

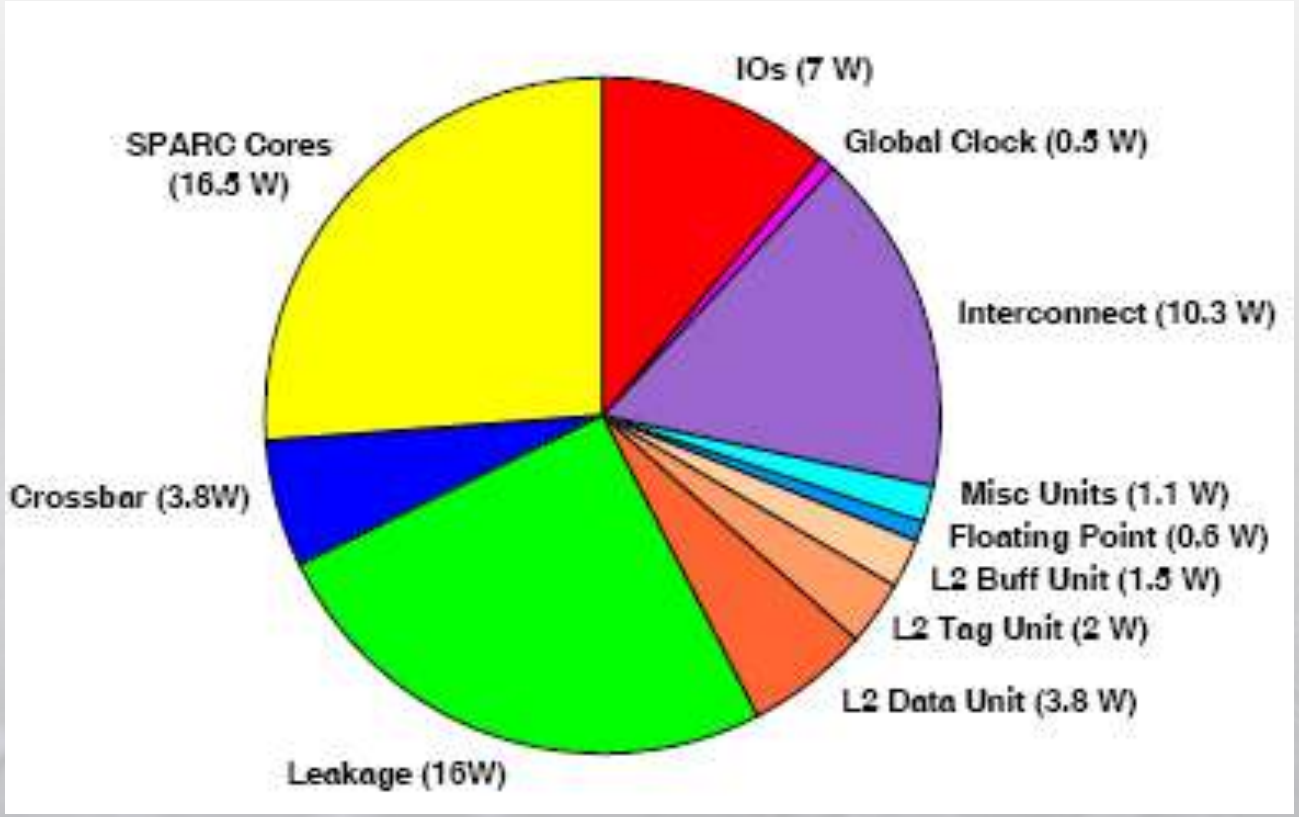
- 8 64-bit Multithreaded SPARC Cores
- Shared 3 MB, 12-way 64B line writeback L2 Cache
- 16 KB, 4-way 32B line ICache per Core
- 8 KB, 4-way 16B line write-through DCache per Core
- 4 144-bit DDR-2 channels
- 3.2 GB/sec JBUS I/O

Technology:

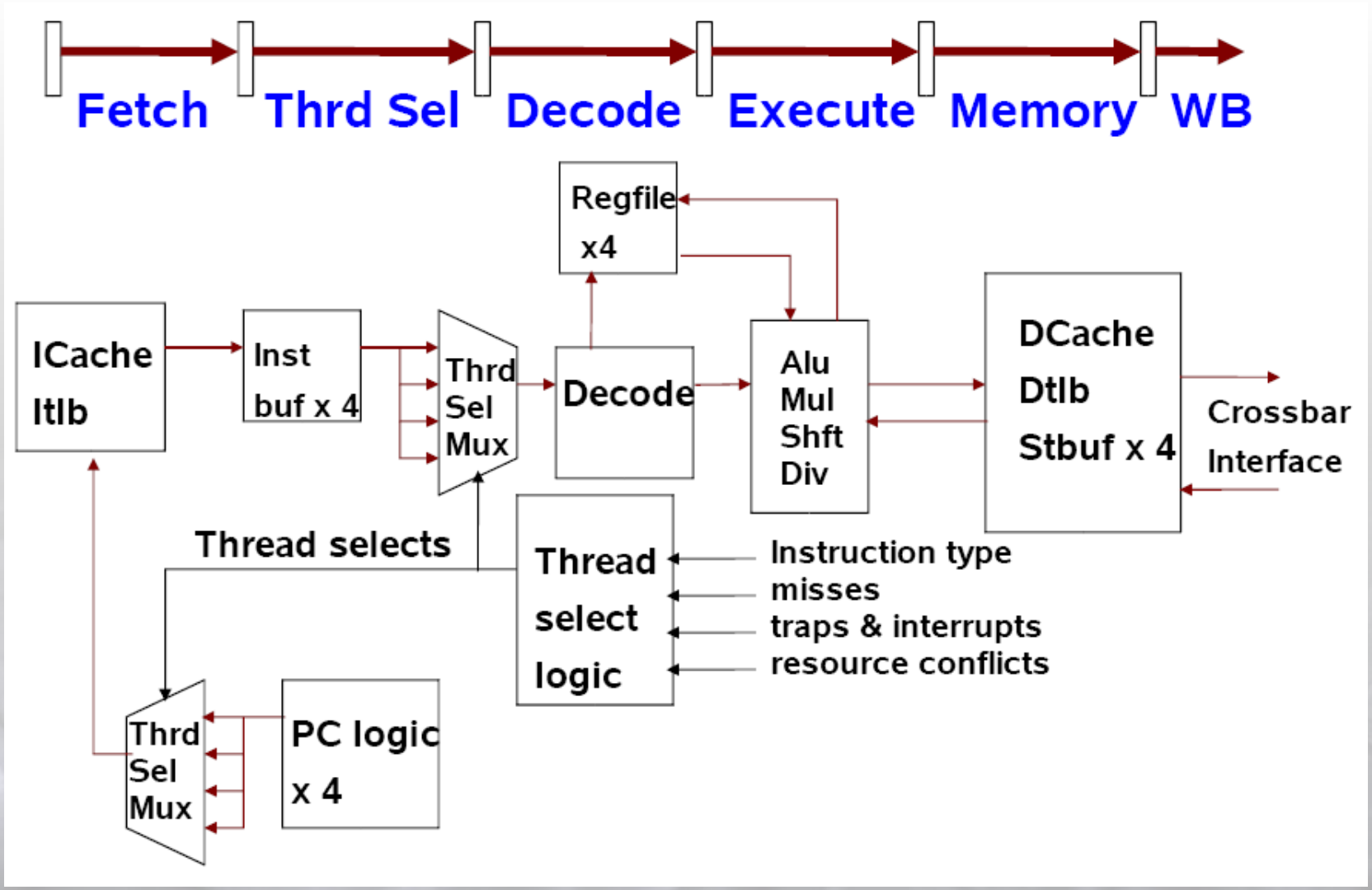
- TI's 90nm CMOS Process
- 9LM Cu Interconnect
- 63 Watts @ 1.2GHz/1.2V
- Die Size: 379mm^2
- 279M Transistors
- Flip-chip ceramic LGA

Enclosure (SunFire T2000)

Power Breakdown



The SUN UltraSparc T1 Pipeline



Cost

Sun Fire T2000		
CPU		UltraSPARC T1
Sockets		1
Height		2U
SpecWeb 2005	Performance	14001
	Power	330 W
	Perf/Watt	42.4
SpecJBB 2005	Performance	63378 BOPS
	Power	298 W
	Perf/Watt	212.7

E10K

1997
32 x US2
77.4 ft³
2000 lbs
13,456 W
52,000 BTUs/hr

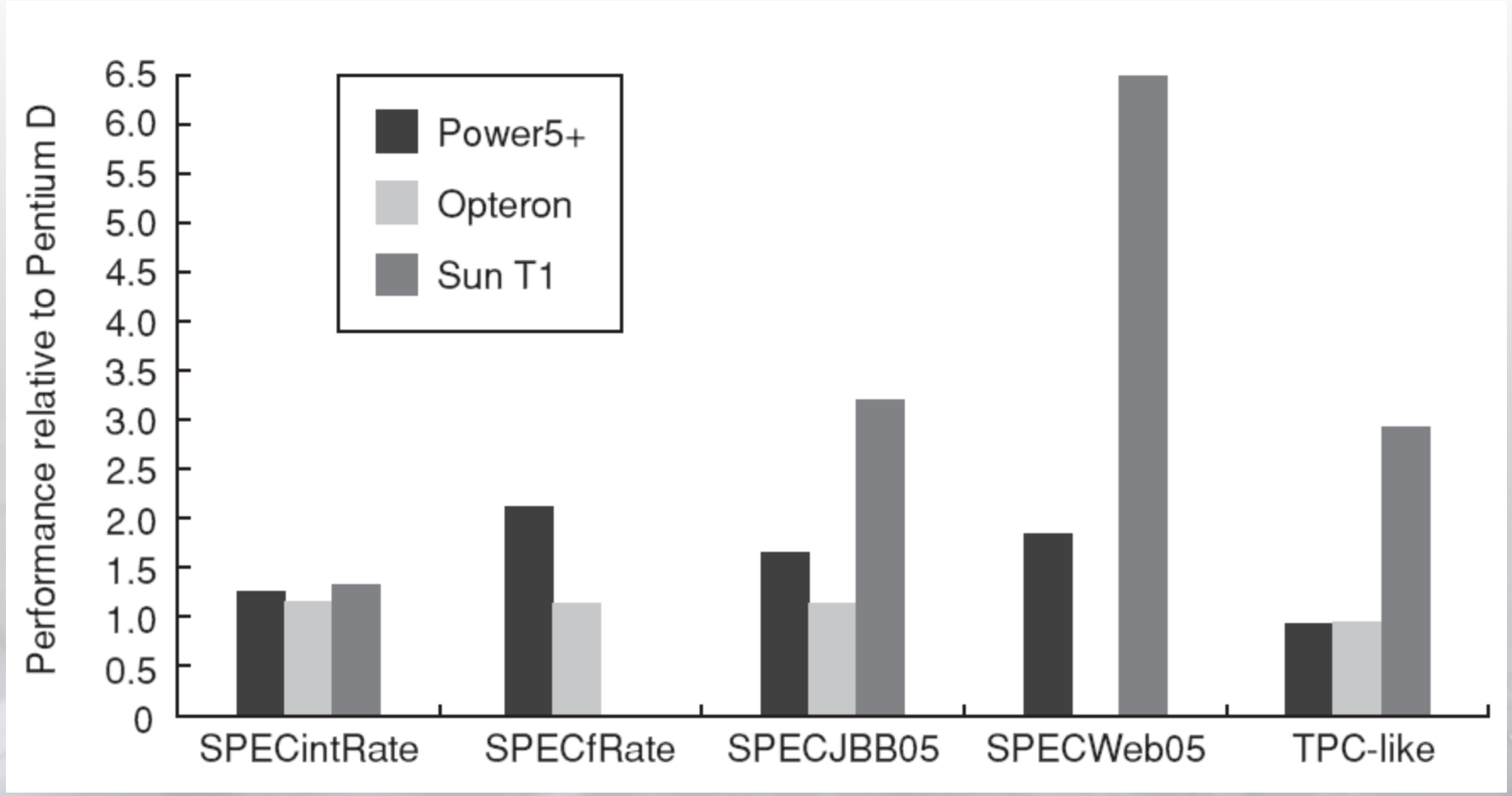
T2000

2005
1 x US T1
0.85 ft³
37 lbs
~300 W
1,364 BTUs/hr

Performance

Characteristic	SUN T1	AMD Opteron	Intel Pentium D	IBM Power5
Cores	8	2	2	2
Instruction issues per clock per core	1	3	3	4
Multithreading	Fine-grained	No	SMT	SMT
Caches L1 I/D in KB per core L2 per core/shared L3 (off-chip)	16/8 3 MB shared	64/64 1 MB/core	12K uops/16 1 MB/core	64/32 L2: 1.9 MB shared L3: 36 MB
Peak memory bandwidth (DDR2 DRAMs)	34.4 GB/sec	8.6 GB/sec	4.3 GB/sec	17.2 GB/sec
Peak MIPS FLOPS	9600 1200	7200 4800 (w. SSE)	9600 6400 (w. SSE)	7600 7600
Clock rate (GHz)	1.2	2.4	3.2	1.9
Transistor count (M)	300	233	230	276
Die size (mm^2)	379	199	206	389
Power (W)	79	110	130	125

Performance (Pentium D normalized)



Sun UltraSparc T2

- Second iteration of Niagara
- 8 pipelined cores, 8-way SMT ➔ 64 hardware threads per chip
- 4MB L2
- 8 Fully pipelined FPU (1 per core)
- Dual 10 GbE and PCIe integrated
- Security processor per core: DES, 3DES, AES, etc…
- 65nm, 1.4Ghz, <95W (chip), +60GB/s BW (4 FB DIMM controllers)

- Sun UltraSparc T3 (a.k.a. Niagara Falls)
  - 16 cores 16-way SMT, 4-chip servers

RESEARCH

Research: Speculative Multithreading

- **Speculative multithreading**
  - Use multiple threads/processors for **single-thread performance**
  - Speculatively parallelize sequential loops, that might not be parallel
    - Processing elements (called PE) arranged in logical ring
    - Compiler or hardware assigns iterations to consecutive PEs
    - Hardware tracks logical order to detect mis-parallelization
  - Techniques for doing this on non-loop code too
    - Detect reconvergence points (function calls, conditional code)
  - Effectively chains ROBs of different processors into one big ROB
    - Global commit "head" travels from one PE to the next
    - Mis-parallelization flushes one PEs, but ***not all*** PEs
  - Also known as split-window or "Multiscalar"
  - Not commercially available yet… eventually will be??
    - But it is the "biggest idea" from academia not yet adopted

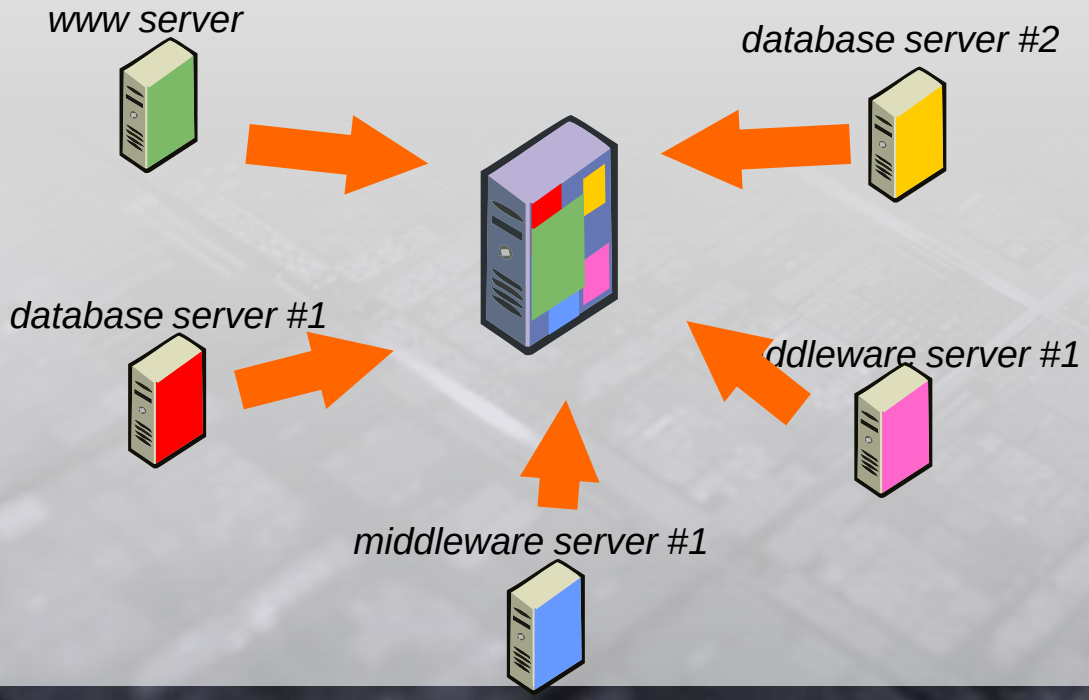
Research: Multithreading for Reliability

- Can multithreading help with reliability?
  - Design bugs/manufacturing defects? No
  - Gradual defects, e.g., thermal wear? No
  - Transient errors? Yes

- **Staggered redundant multithreading (SRT)**
  - Run two copies of program at a slight stagger
  - Compare results, difference? Flush both copies and restart
  - Significant performance overhead

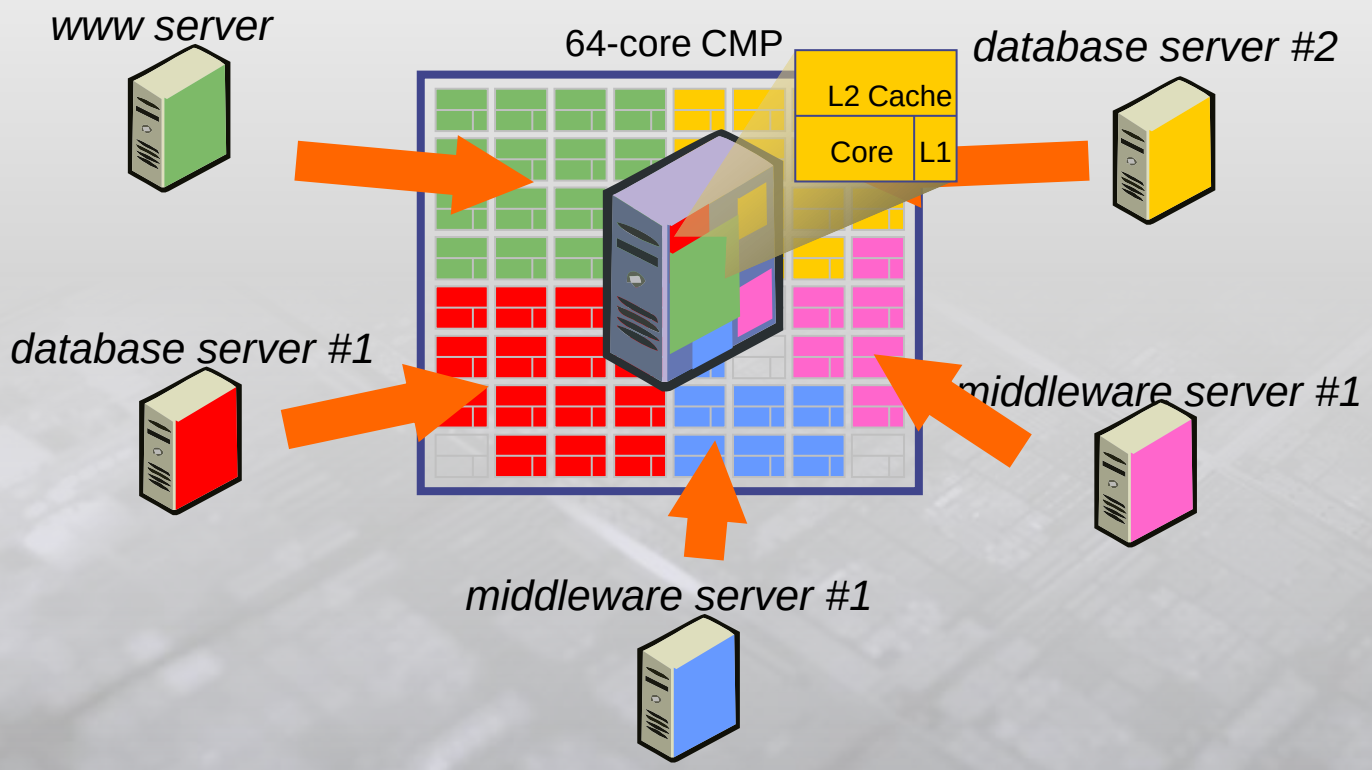
Research: QoS CMP

- **Server Consolidation** Definition :
  - Server consolidation is an approach to the efficient usage of computer server resources in order to reduce the total number of servers or server locations that an organization requires



Motivation: CMP & Server consolidation

- CMP based system are pervasive and an ideal hardware platform (in terms of cost) to deploy server consolidation



Virtualization

- Virtualization is the most suitable tool to achieve server consolidation
  - Hides all the nasty details to the user
  - Eases system administration
  - Performance?

- Most virtualization layers provide a huge set of parameters to regulate the access to shared resources
  - Disk
  - Memory
  - Network
  - Time sharing CPU

Performance Isolation & CMP

- CMP introduces a new dimension in shared resources
  - A large portion of the memory hierarchy will be shared
- Two mayor potential problems
  - On-chip cache capacity and/or bandwidth
  - Off-chip bandwidth
- Question that arises:
  - Too fine grained to be effective at software level?
  - Has to be the CMP aware of virtualization layer?
  - Has to provide the hardware some mechanism to apply virtualization policies?
  - …

Is it a problem?

- Let's analyze the problem with state-of-the-art infrastructure

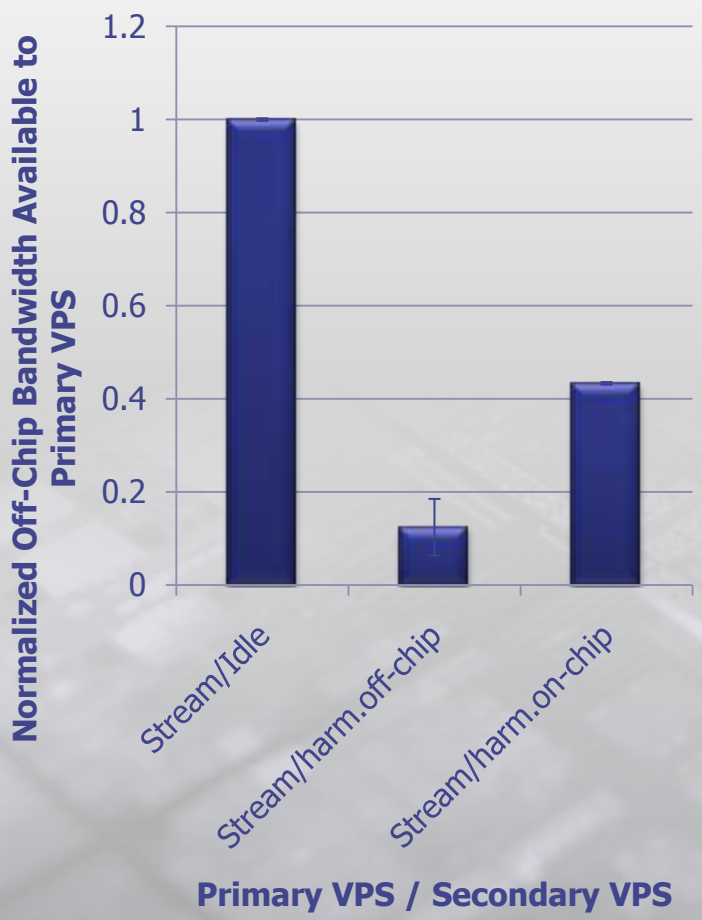
- Consolidated servers
  - Significant workload to be consolidates (SPECWeb2005) with Apache Tomcat
  - Misbehaving workload that stresses CMP (on-chip capacity & off-chip bandwidth)

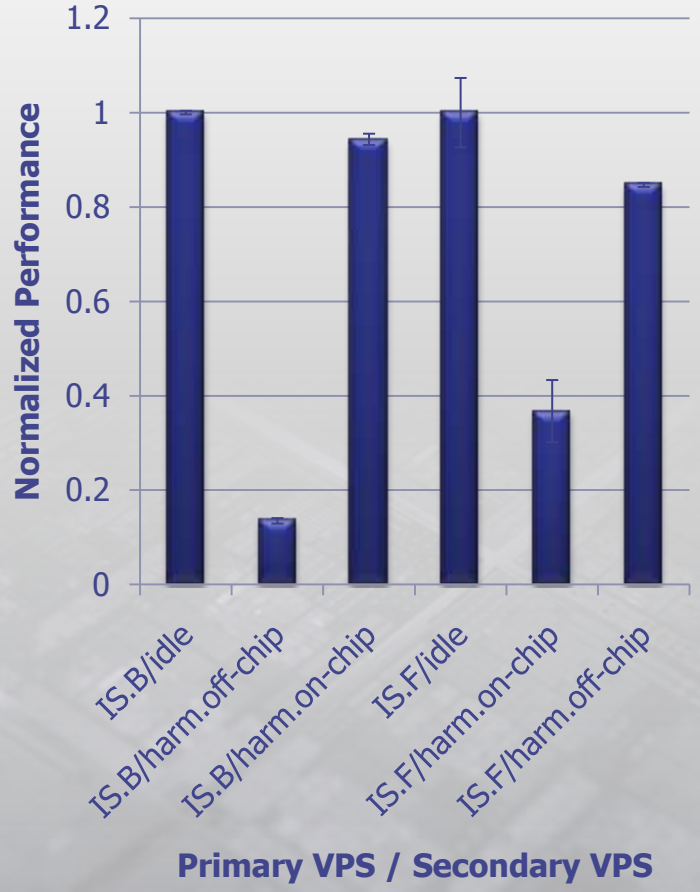
- Software stack
  - Virtualization layer (XEN & Solaris Zones) with resource controls
  - Realistic Operating systems (GNU/Linux Debian5, Solaris 10)

Hardware Setup

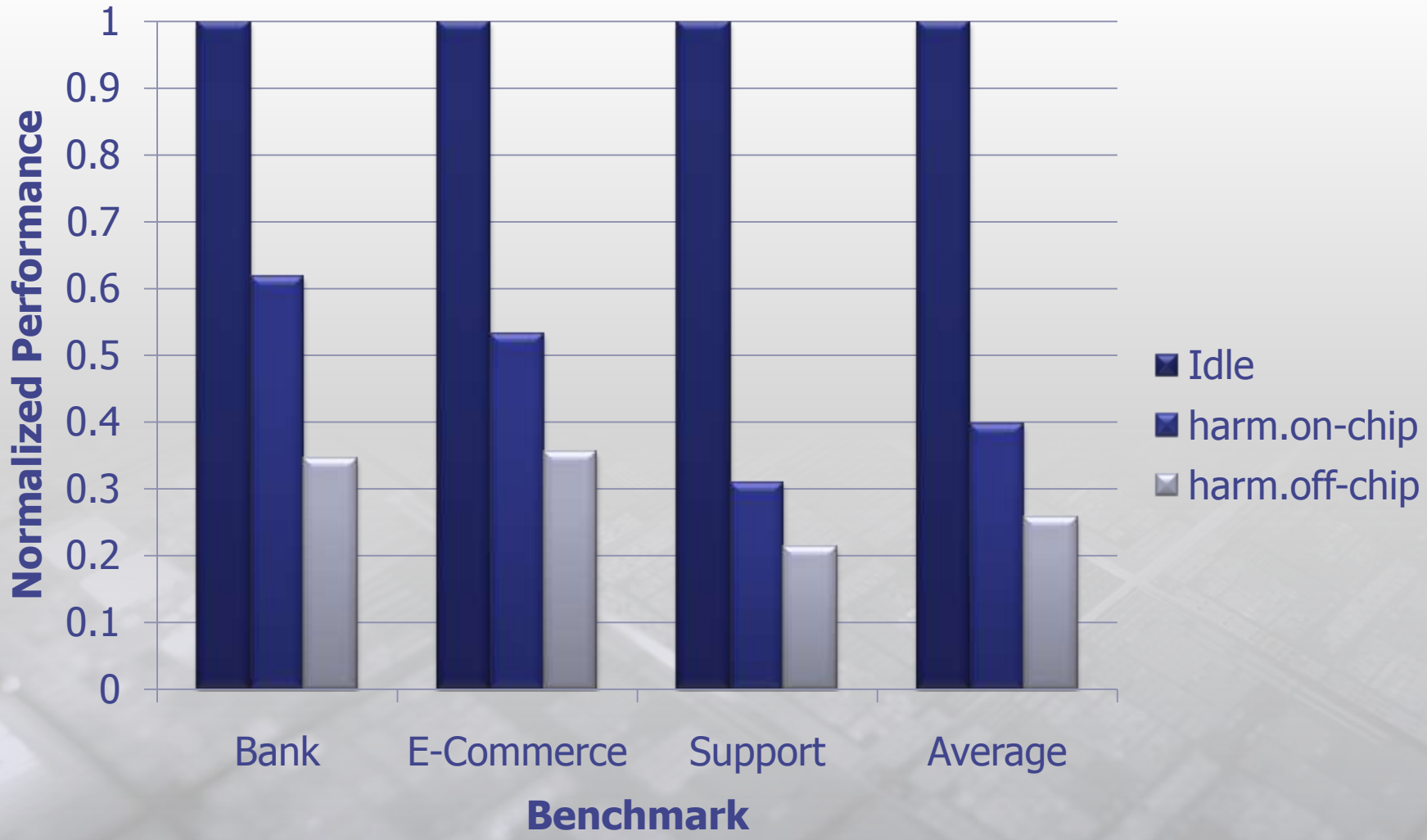
	Sun Fire T1000	HP Proliant DL160 G5
Processor Model	Sun UltraSparc T1	Intel Xeon X5472
Threads, Cores, dies, chips	32, 8, 1, 1	8, 8, 4, 2
L1I	Private, 16KB 4-way set associative, 32-byte lines, 1-per core	Private, 32KB, 8-way set associative, 64-byte lines, 1-per core
L1D	Private, 8KB 4-way set associative, 16-byte lines, write-through, 1 per core	Private, 32KB, 8-way set associative, 64-byte lines, 1-per core
L2	3MB Shared per chip, 12-way set associative, 64-byte lines	6MB Shared per die, 24-way set associative, 64 byte lines
Memory	8GB, 533Mhz, DDR2	16GB, 667Mhz FB-DIMM
Hard disk	1 SATA	2 SATA RAID 0
NIC	Dual Gigabit Ethernet (bounded)	Dual Gigabit Ethernet (bounded)

Misbehaving application on T1000

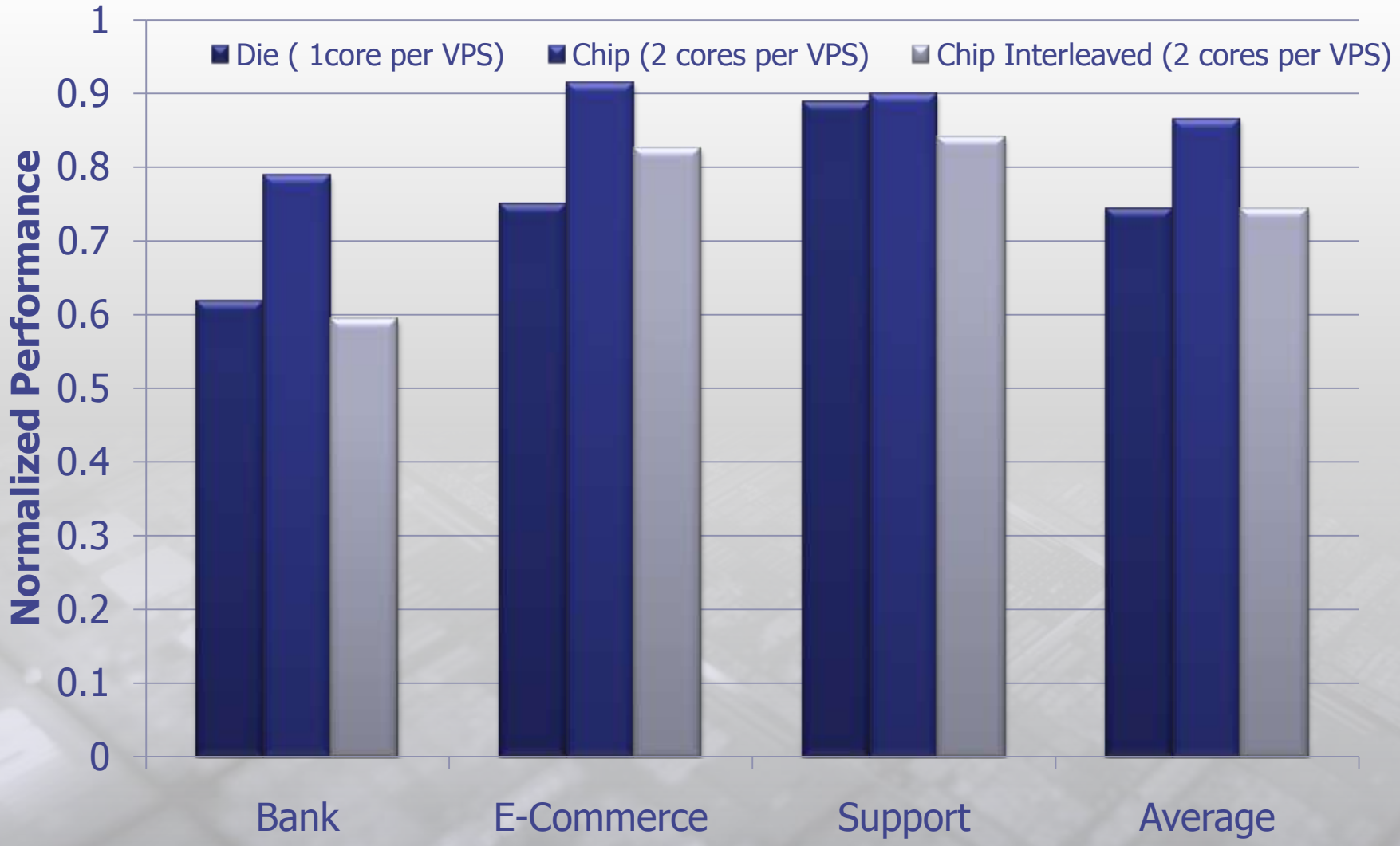




Performance isolation on T1000



Performance Isolation on Xeon



What to do?

- Current virtualization + hardware is not enough to provide the expected level of performance
- Could a provider guarantee SLA?
  - Potentially could reduce the server consolidation degree
  - Potentially could loss its customers
- With many core-CMP advent the problem will exacerbate
- We need some sort of QoS provision
  - Hardware level mechanisms to control the access to common resources
  - Software could specify policies for those mechanisms

Acknowledgments

- Slides developed by Amir Roth of University of Pennsylvania with sources that included University of Wisconsin slides by Mark Hill, Guri Sohi, Jim Smith, and David Wood.
- Slides enhanced by Milo Martin and Mark Hill with sources that included Profs. Asanovic, Falsafi, Hoe, Lipasti, Shen, Smith, Sohi, Vijaykumar, and Wood
- Slides re-adapted by V. Puente of University of Cantabria