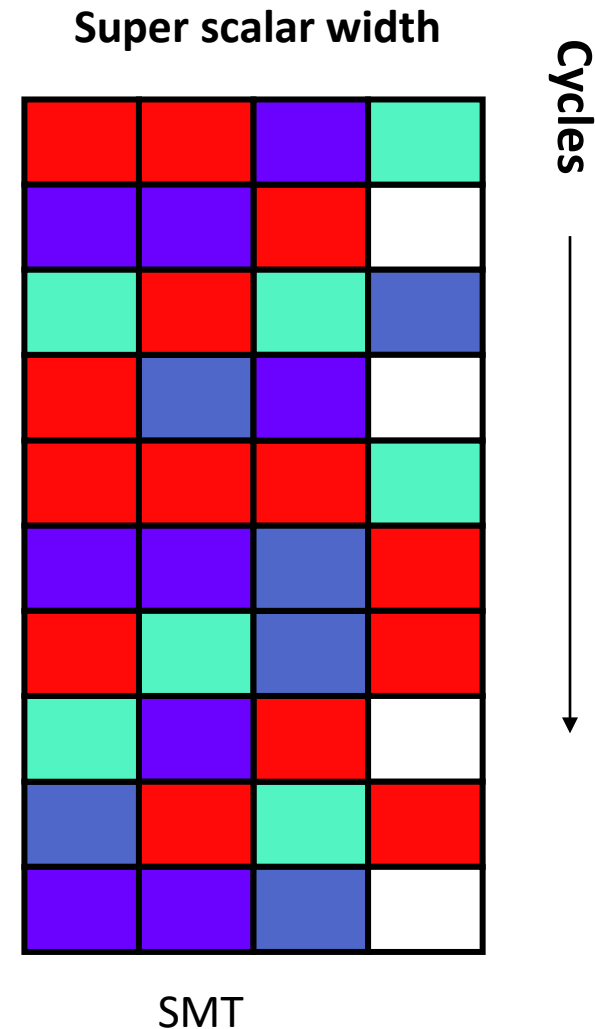




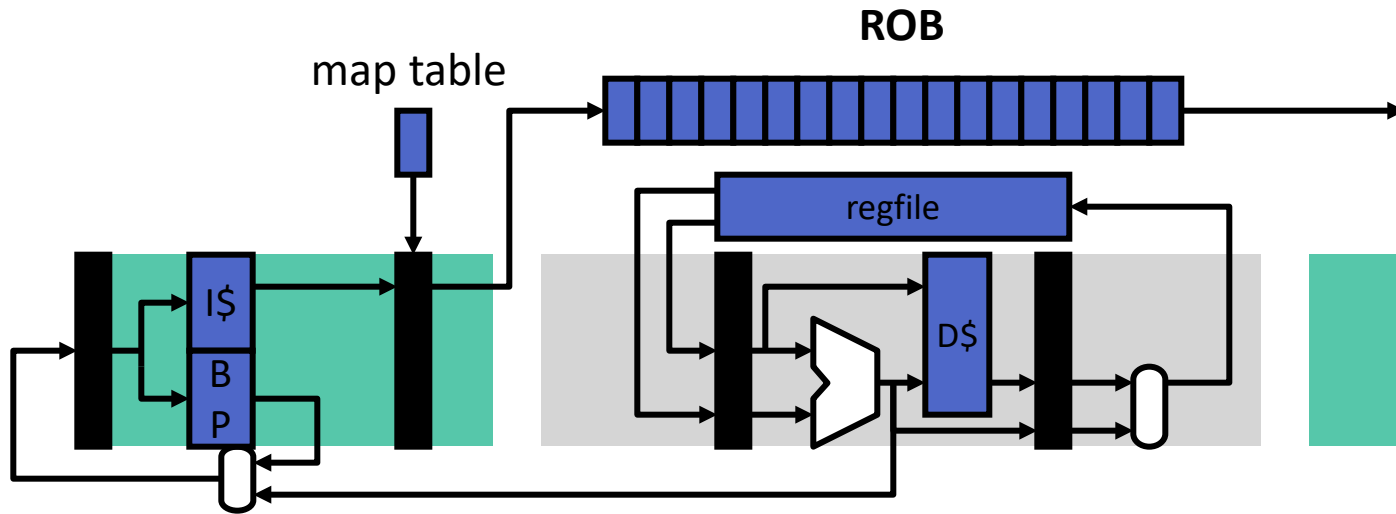
Multi -threading (part 2) + Multicores

Simultaneous multithreading (SMT)

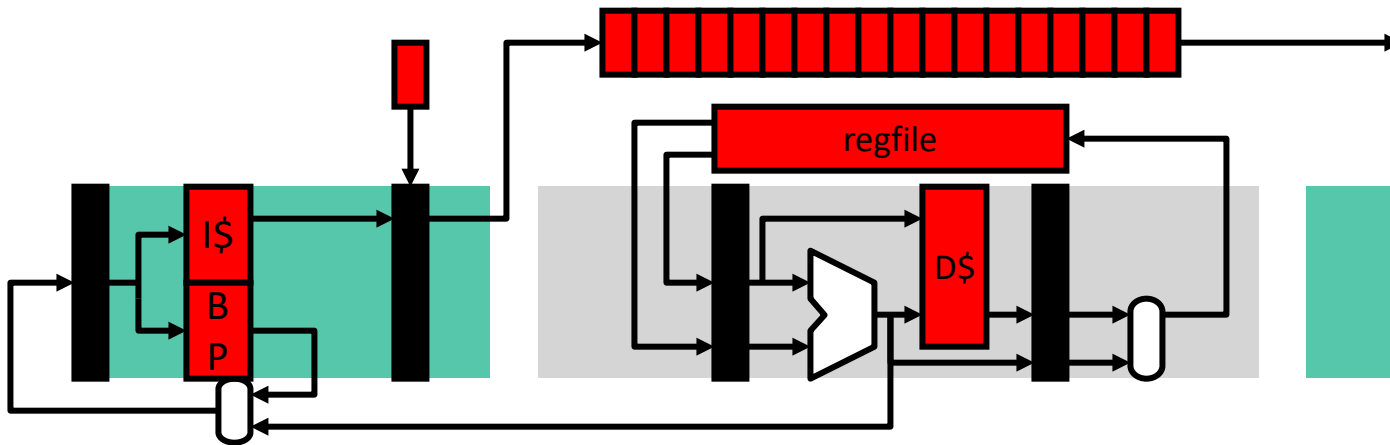
- Can we multithread an out-of-order machine?
 - ▶ Don't want to give up performance benefits
 - ▶ Don't want to give up natural tolerance of D\$ (L1) miss latency
- In each cycle, instruction can be issued from whichever thread ready
- Also called **hyperthreading** by Intel/AMD



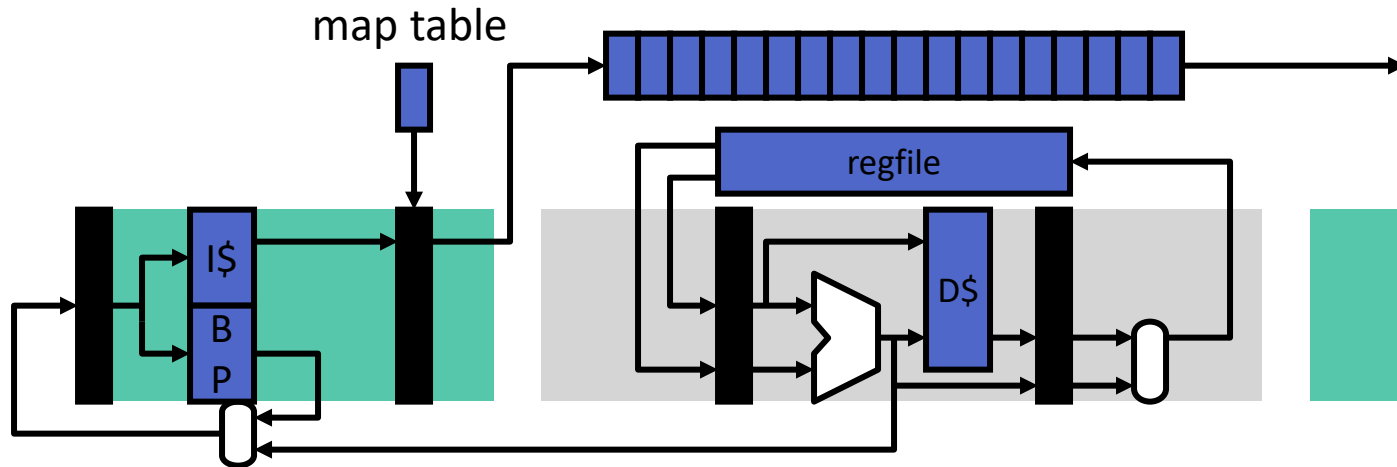
SMT



- + Goal: merge OoO pipelines into one

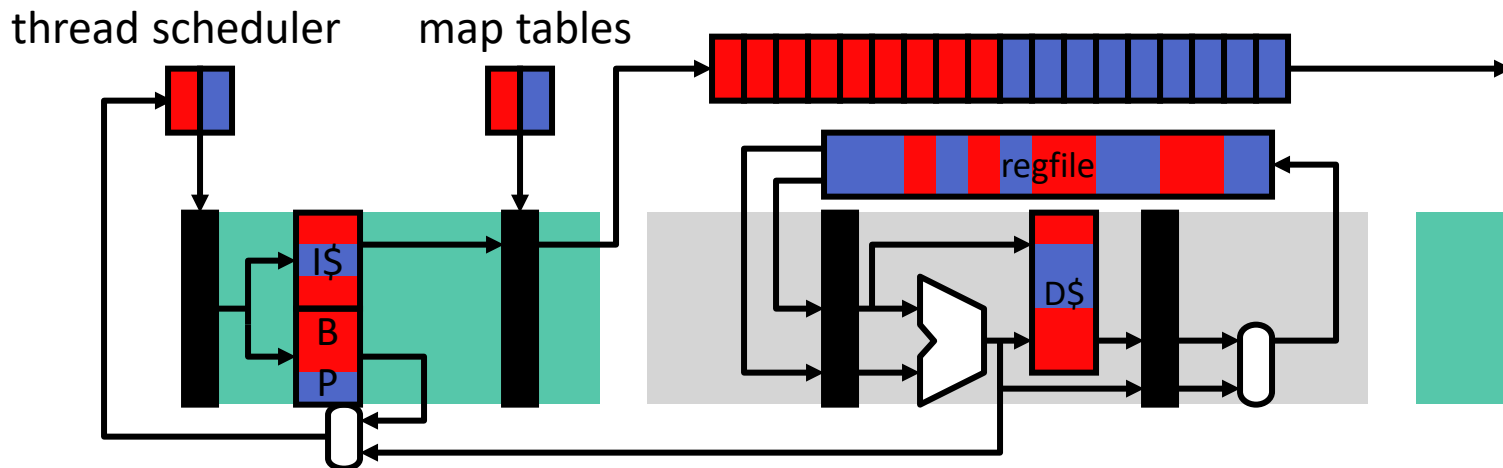


SMT



- SMT

– Replicate map table, share (larger) physical register file



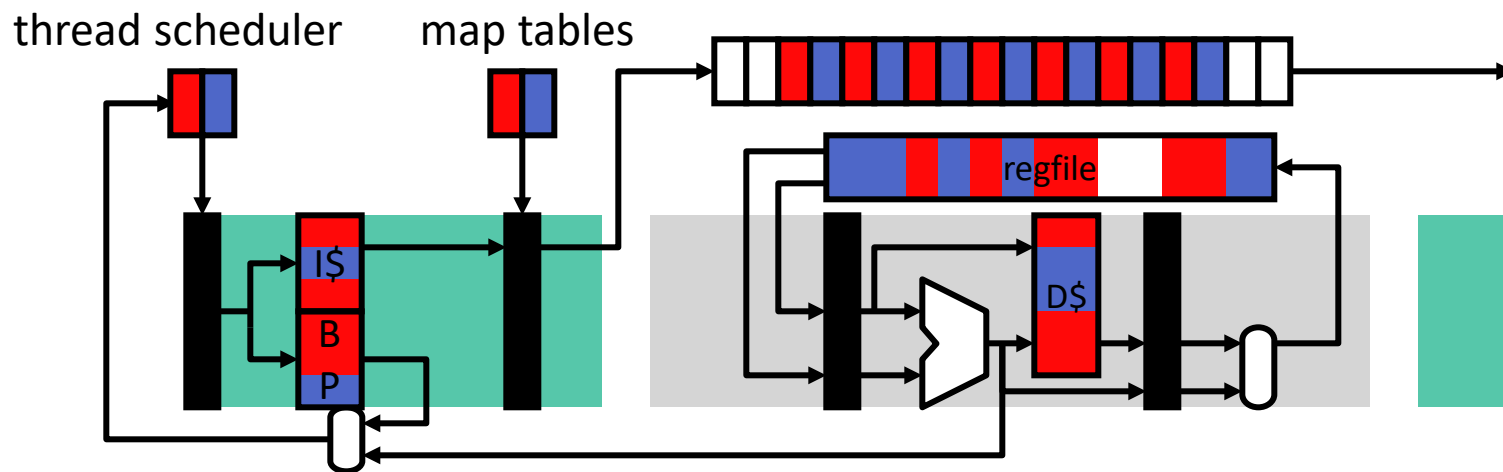


SMT Resource Partitioning

- Physical reg. file and instruction buffer entries shared at fine-grain
 - Physically unordered and so fine-grain sharing is possible
- How are physically ordered structures (ROB/LSQ) shared?

SMT Resource Partitioning

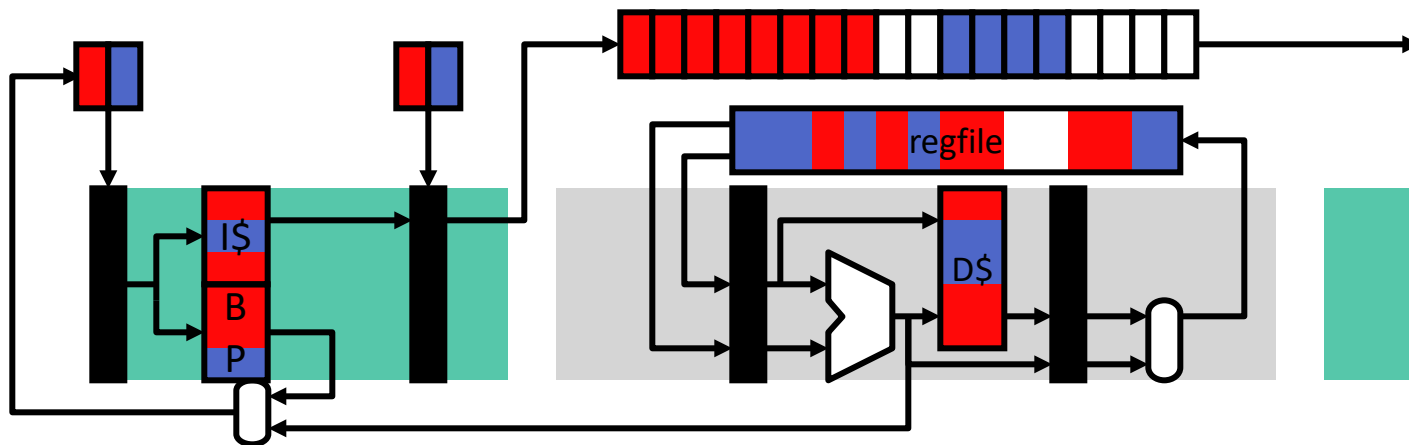
- Physical reg. file and instruction buffer entries shared at fine-grain
 - Physically unordered and so fine-grain sharing is possible
- How are physically ordered structures (ROB/LSQ) shared?
 - Fine-grain sharing (below) would entangle commit (and squash)
 - Allowing threads to commit independently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



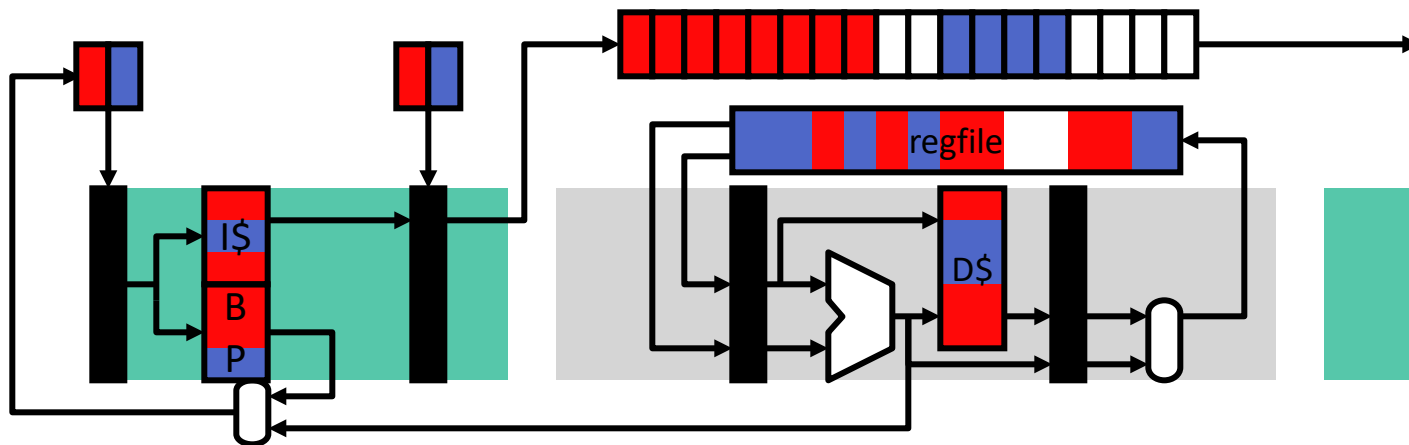
Static & Dynamic Resource Partitioning

- **Static partitioning** (below)
 - T equal-sized contiguous partitions
 - ± No starvation, sub-optimal utilization (fragmentation)

- Couple both with larger ROB/LSQs



- **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 ROB/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?
 - To keep miss rates low, SMT might need a larger L2
 - Out-of-order tolerates L1 misses
- Large physical register file (and map table)
 - Larger ROB, LSQ is also needed



Multithreading Issues

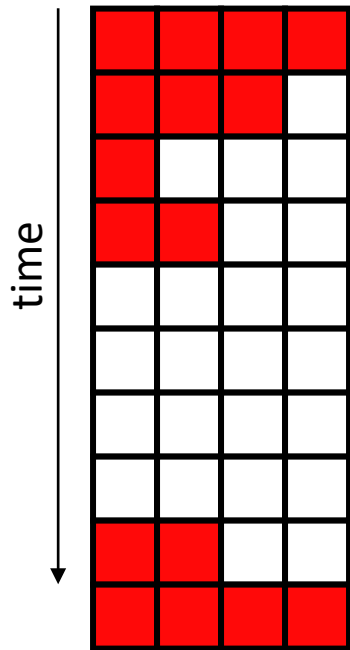
- Shared soft state (caches, branch predictors, TLBs, etc.)
- Key example: **cache interference**
 - General concern for all MT variants
 - Can the working sets of multiple threads fit in the caches?
 - To keep miss rates low, SMT might need a larger L2
 - Out-of-order tolerates L1 misses
- Large physical register file (and map table)
 - physical registers = (**#threads** * #arch-regs) + #in-flight insns
 - map table entries = (**#threads** * #arch-regs)
- Larger ROB, LSQ is also needed

Simultaneous Multithreading

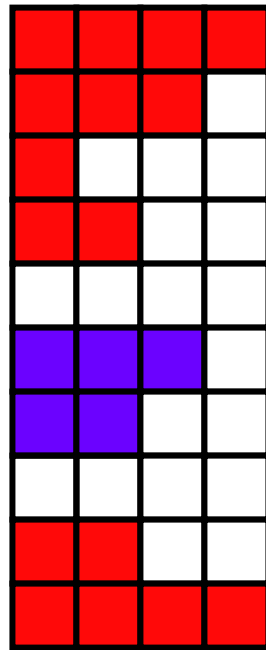
- ✗ Sacrifices some single thread performance
- ✓ Tolerates all latencies
- ✓ Increases pipeline utilization as much as possible
- ✓ A natural extension of superscalar OOO pipelines
 - ▶ Front-end handles fetching and dispatching from separate threads
 - ▶ The back-end can just mix instructions from all threads

Multithreading summary

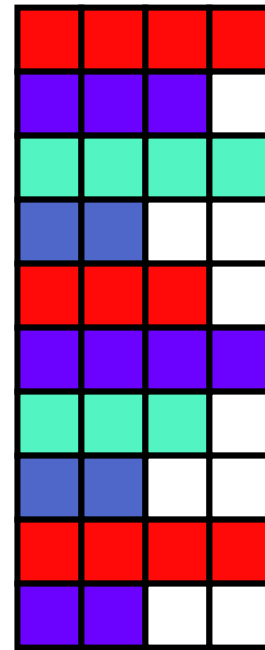
- Time evolution of issue slots
 - Different color = different (h/w) thread. White = unused



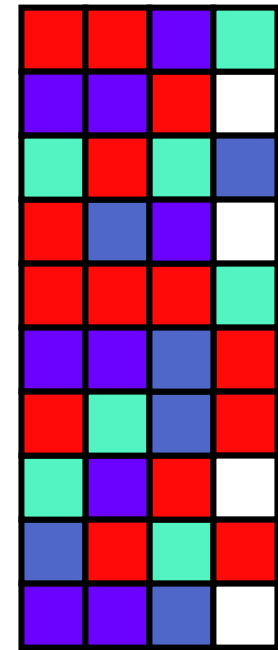
Superscalar



CGMT



FGMT



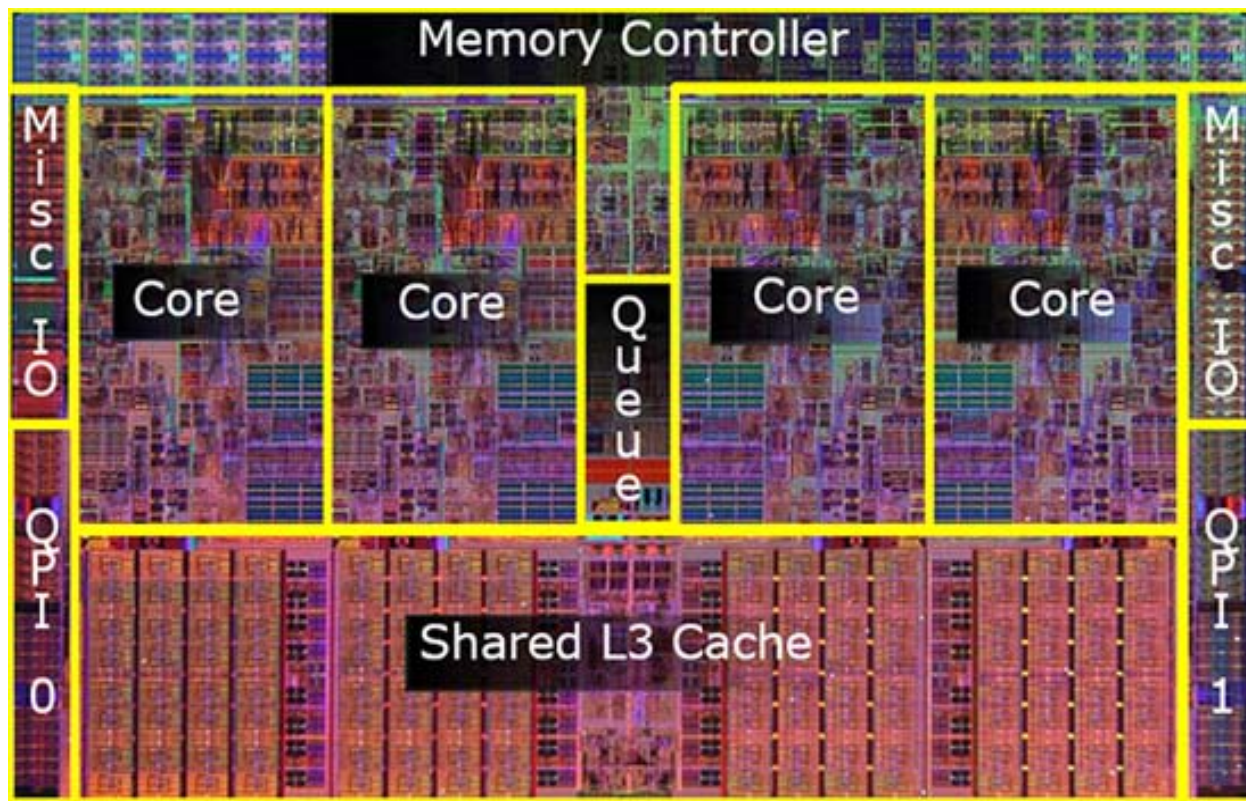
SMT

Architectures to Exploit TLP

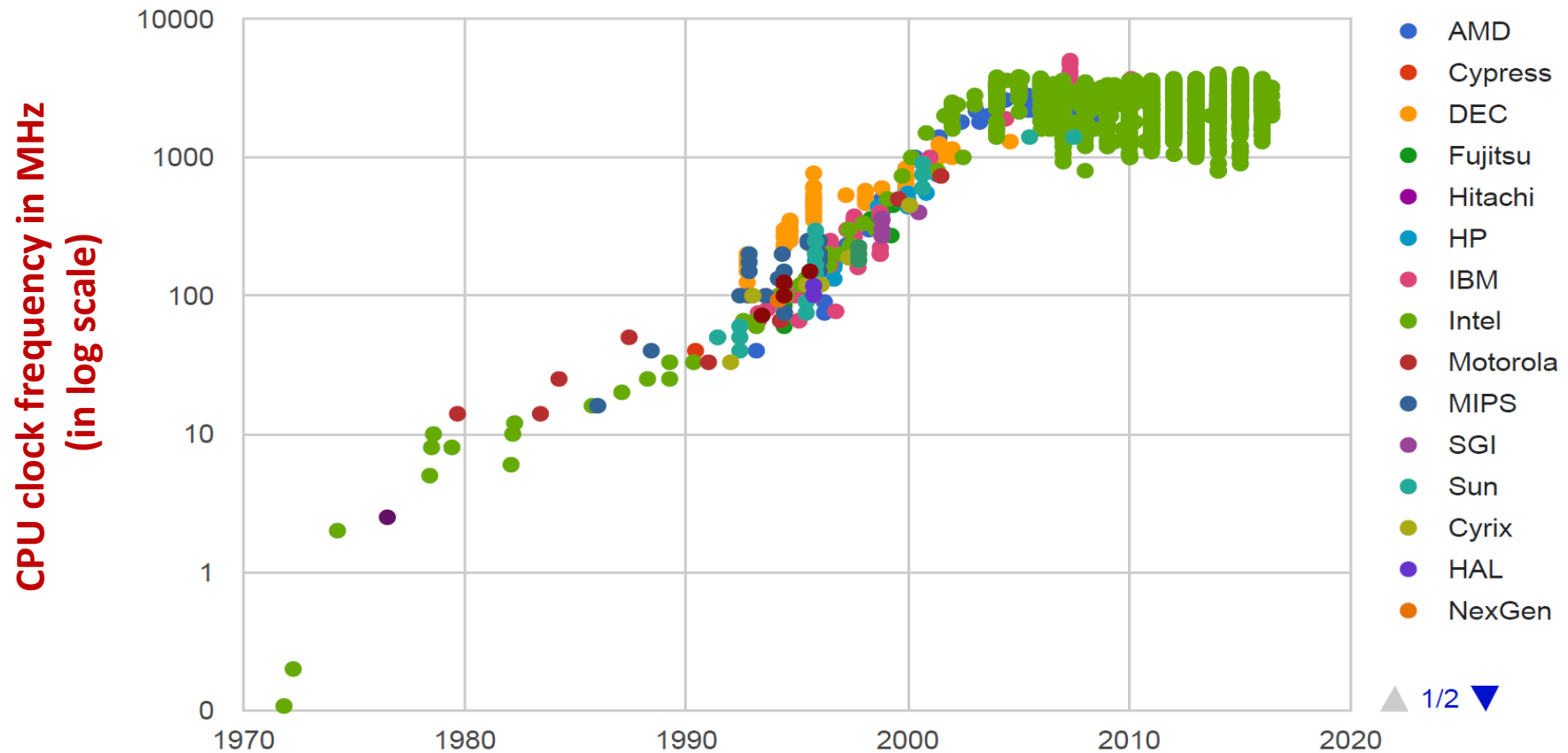
- ***Hardware Multithreading (MT)***: Multiple threads share the same processor pipeline
 - ▶ Coarse-grained MT (CGMT)
 - ▶ Fine-grained MT (FMT)
 - ▶ Simultaneous MT (SMT)
- ***Multiprocessors (MP)***: Different threads run on **different** processors (**separate pipelines**)
 - ▶ Multiple processor chips
 - ▶ Chip Multiprocessors (CMP), a.k.a. Multicore processors
 - ▶ A combination of the above

Multicores/ Chip multiprocessor

- Multiple duplicated cores on a single chip

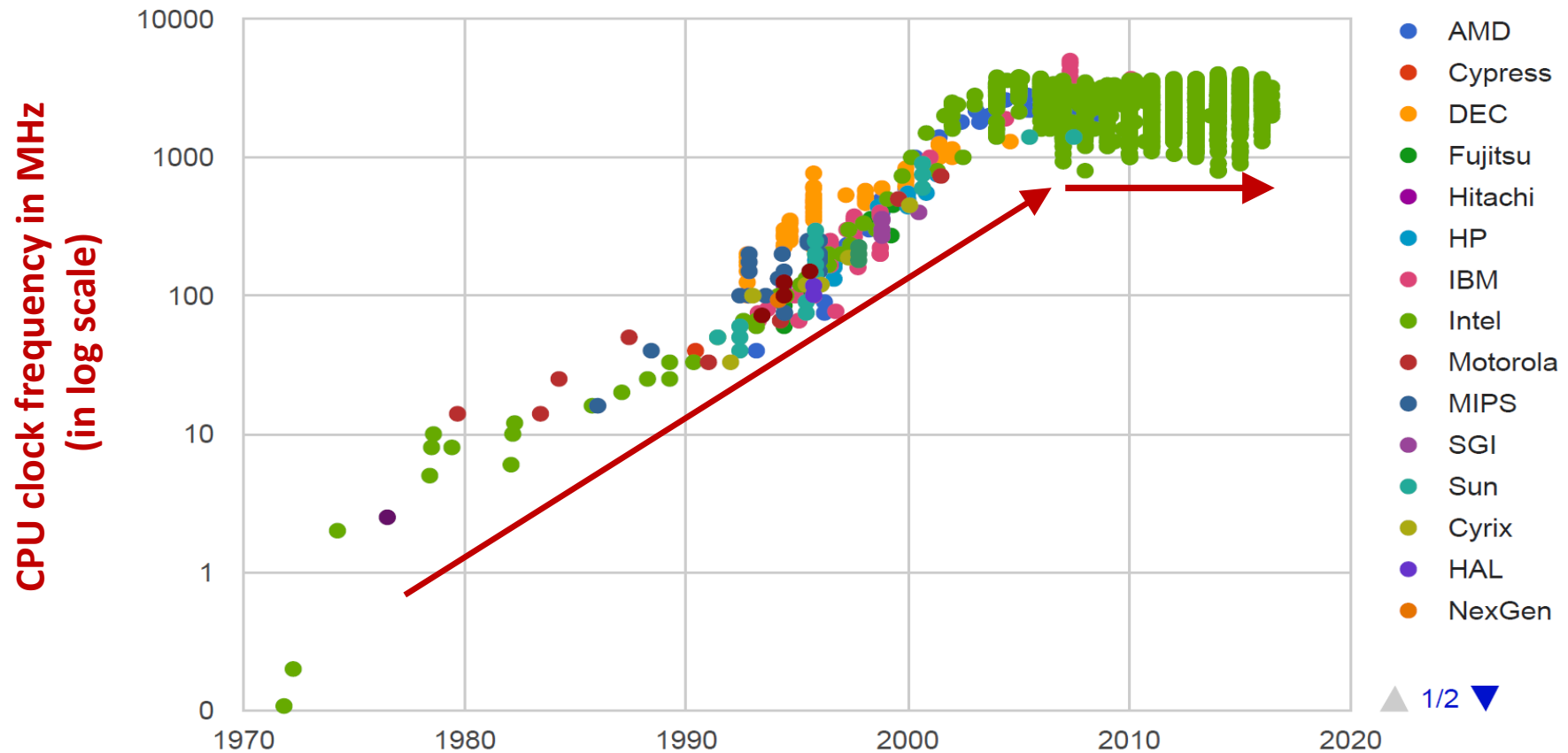


What led to multi-cores?



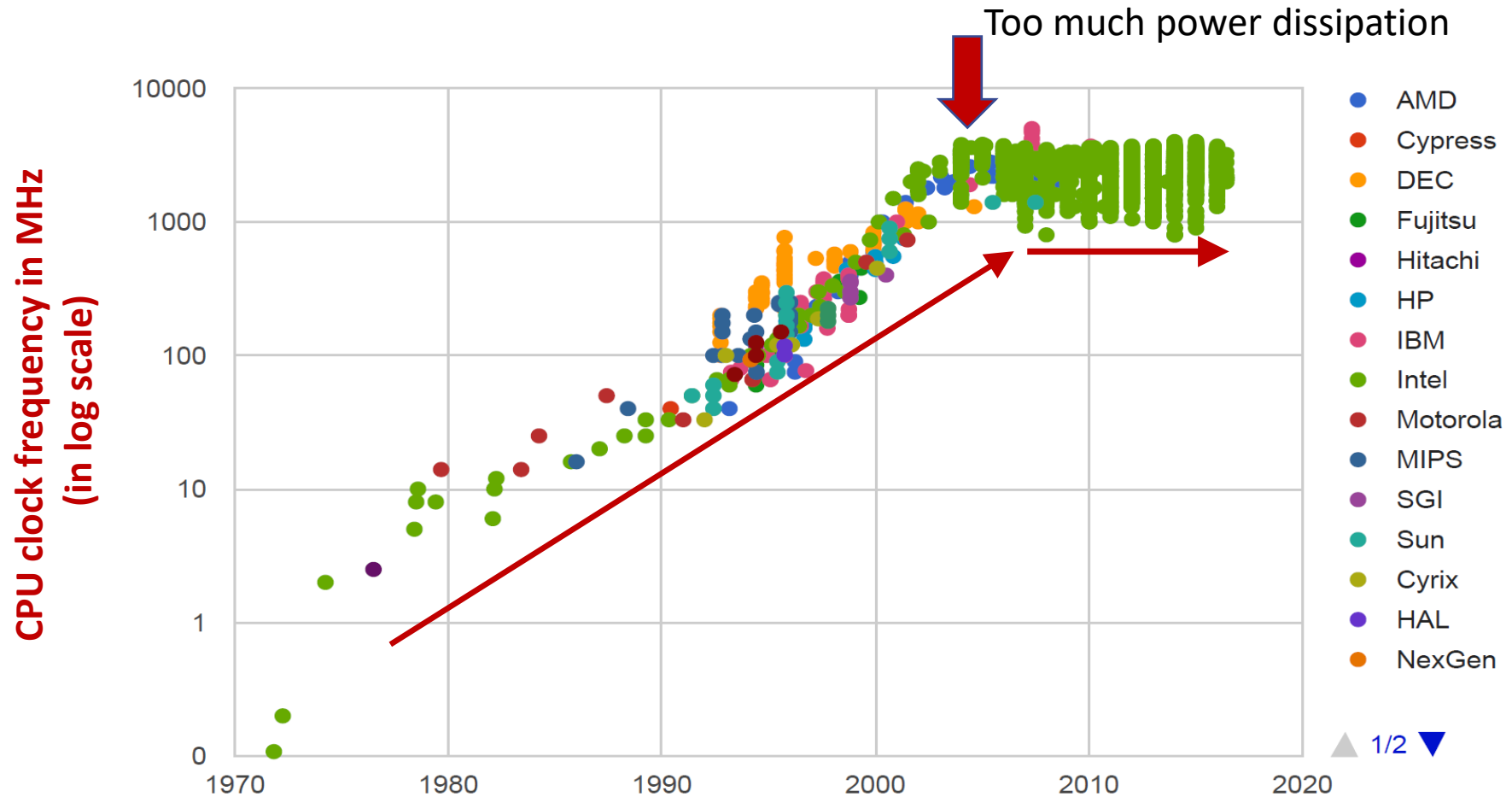
Historical data from Stanford CUPDB

What led to multi-cores?



Historical data from Stanford CUPDB

What led to multi-cores?



Historical data from Stanford CUPDB

What drove single -core perf.?

Two “laws” that drove transistor technology

- Moore’s law: In 1965, Gordon E. Moore—co-founder of Intel —postulated that the number of transistors that can be packed into a given unit of space will double about every ~two years.

What drove single -core perf.?

Two “laws” that drove transistor technology

- Moore’s law: In 1965, Gordon E. Moore—co-founder of Intel —postulated that the number of transistors that can be packed into a given unit of space will double about every ~two years.
- Dennard’s scaling: In 1974 ISSC paper, Robert H. Dennard – as the transistor got smaller, transistor got faster, but power density remained constant.

What drove single -core perf.?

- Effect of Moore's law + Dennard's scaling:
 - ▶ Smaller, faster transistors without increase in power density
 - ▶ Faster transistors → Higher core frequency
 - ▶ More transistors → More architectural innovation/ bigger caches etc.
 - ▶ Power-efficient transistors → No impact of higher clock frequency on power density


What drove single -core perf.?

- Effect of Moore's law + Dennard's scaling:
 - ▶ Smaller, faster transistors without increase in power density
 - ▶ Faster transistors → Higher core frequency
 - ▶ More transistors → More architectural innovation/ bigger caches etc.
 - ▶ Power-efficient transistors → No impact of higher clock frequency on power density


What drove single -core perf.?

- Effect of Moore's law + Dennard's scaling:
 - ▶ Smaller, faster transistors without increase in power density
 - ▶ Faster transistors → Higher core frequency
 - ▶ More transistors → More architectural innovation/ bigger caches etc.
 - ▶ ~~Power-efficient transistors → No impact of higher clock frequency on power density~~

What drove single -core perf.?

- Effect of Moore's law ↓ + Dennard's scaling: 
 - ▶ Smaller, faster transistors without increase in power density
 - ▶ Faster transistors → Higher core frequency
 - ▶ More transistors → More architectural innovation/ bigger caches etc.
 - ▶ ~~Power-efficient transistors → No impact of higher clock frequency on power density~~

What drove single -core perf.?

- Effect of Moore's law ↓ + Dennard's scaling: 
 - ▶ Smaller, faster transistors without increase in power density
 - ▶ Faster transistors → Higher core frequency
 - ▶ More transistors → More architectural innovation/ bigger caches etc.
 - ▶ ~~Power efficient transistors → No impact of higher clock frequency on power density~~
- Effect of slowing Moore's law and stalled Dennard's scaling:
 - ▶ Increasing frequency → Power budget is busted
 - ▶ Need more architectural innovation for smarter use of transistors to uplift performance



Multithreading or Multicore?

- If you wanted to run multiple threads would you build a...
 - A multicore: multiple separate pipelines?
 - A multithreaded processor: a single larger pipeline?



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 could be simpler, possibly faster clock
 - SMT could 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**
 - Intel core i9-10850K processors
 - 2-way hyperthreaded (SMT), 10 cores (20 threads)
 - AMD Ryzen 9 5950X desktop processors
 - 2-way SMT, 16 cores (32 threads)



How to connect multiple cores?

- Physical connection:

- ▶ How multiple cores and caches are connected on a CMP?
- ▶ Different on-chip interconnection topologies
 - Also called Network-on-chip or NOC

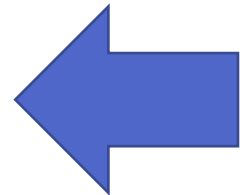
- Logical connection/Logical view:

- ▶ What is the software view?
- ▶ Fundamentally how the memory is shared?

How to connect multiple cores?

- Physical connection:

- ▶ How multiple cores and caches are connected on a CMP?
- ▶ Different on-chip interconnection topologies
 - Also called Network-on-chip or NOC



- Logical connection/Logical view:

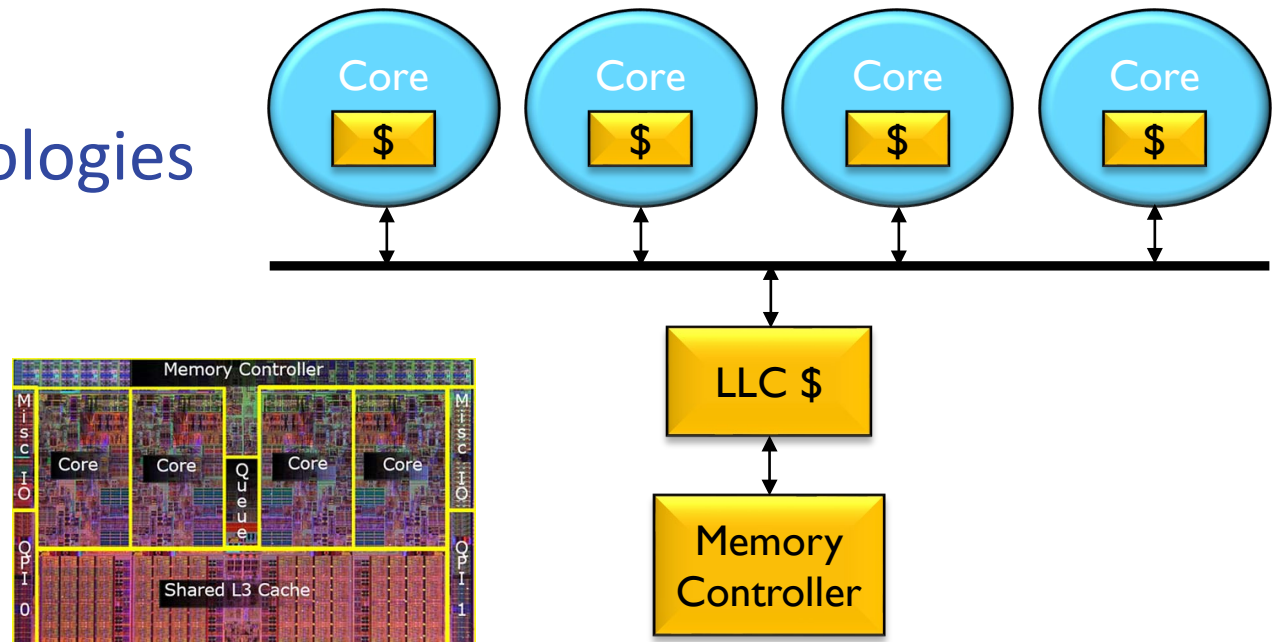
- ▶ What is the software view?
- ▶ Fundamentally how the memory is shared?

On-chip Interconnects (1/5)

- Today, (Core+L1+L2) = “core”
 - (L3+I/O+Memory) = “uncore”
- How to interconnect multiple “core”s to “uncore”?

- Possible topologies

- Bus
- Crossbar
- Ring
- Mesh
- Torus





Limitation with bus

- Wire delays does not scale
 - ▶ Limits the number of cores that can be connected
- Bus arbitration is necessary
 - ▶ Only one core or LLC can drive the bus at a time
 - ▶ Limits b/w



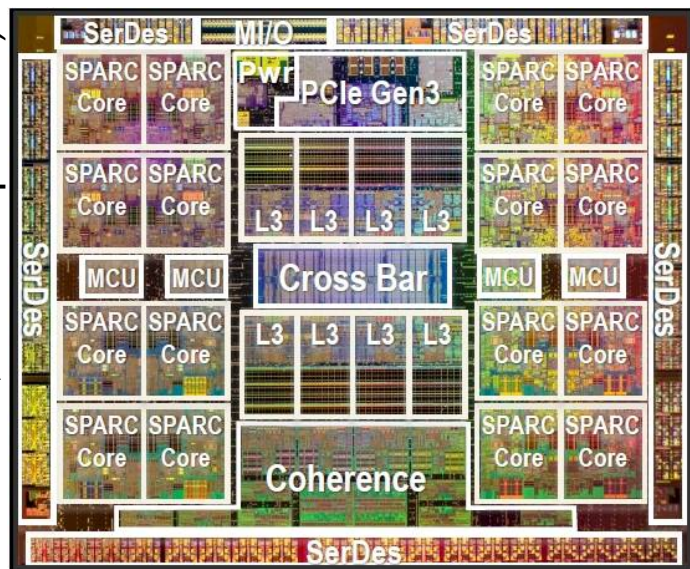
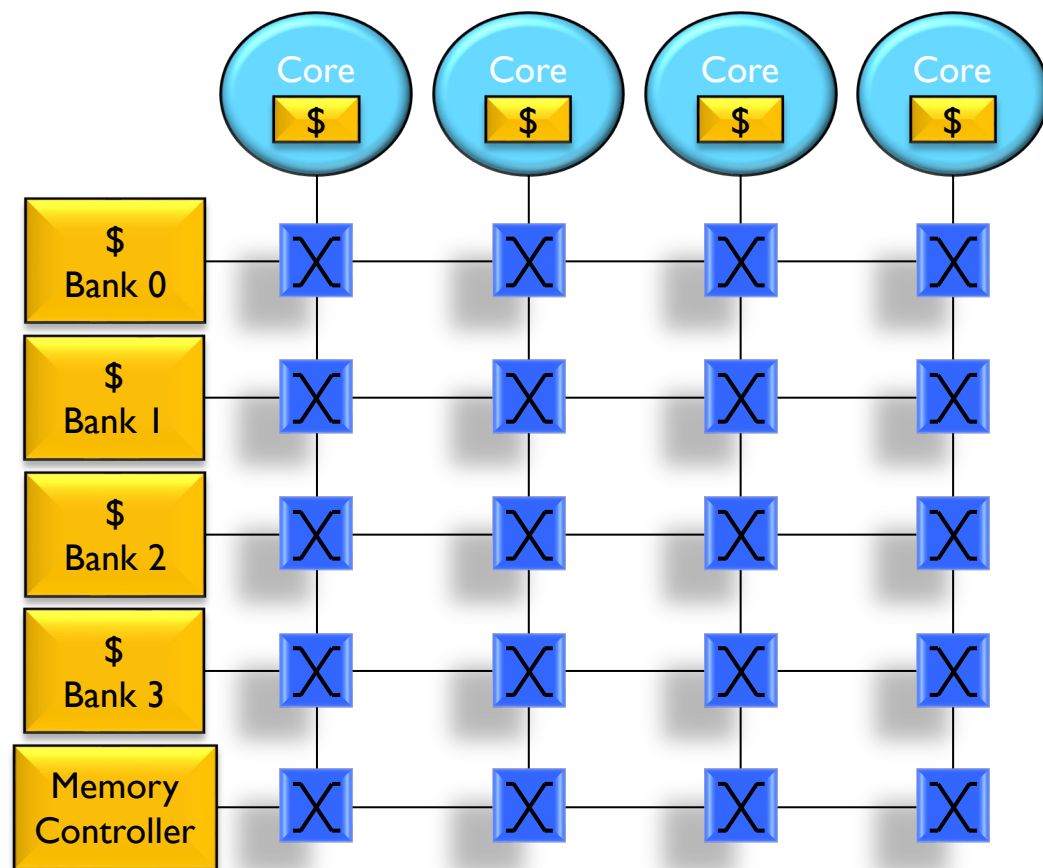
On chip packet -switched networks

- Packetized messages over **point-to-point connections**
- On chip switches and routers
 - Packet routing algorithms
- Different network topologies

On-chip Interconnects (2/5)

■ Possible topologies

- ▶ Bus
- ▶ Crossbar
- ▶ Ring
- ▶ Mesh
- ▶ Torus

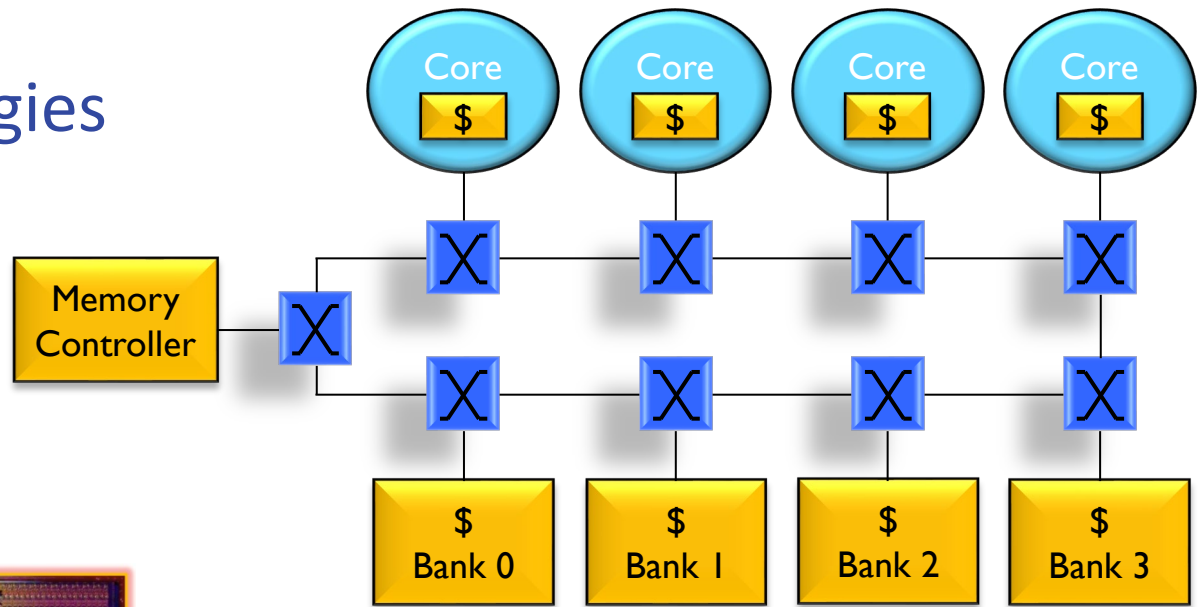


Oracle UltraSPARC T5 (3.6GHz,
16 cores, 8 threads per core)

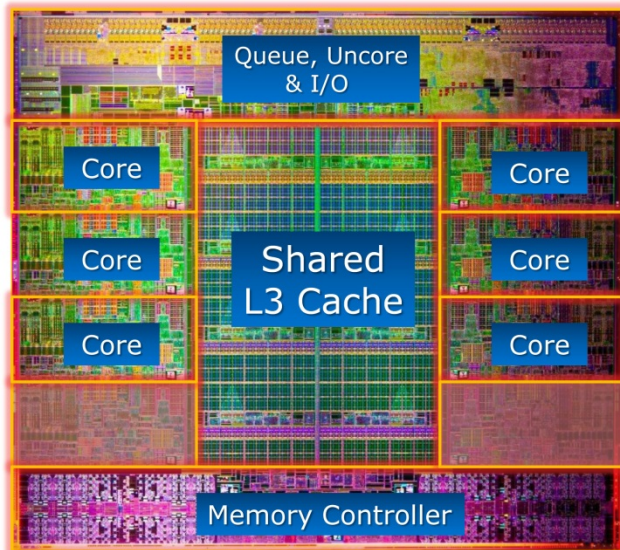
On-chip Interconnects (3/5)

■ Possible topologies

- ▶ Bus
- ▶ Crossbar
- ▶ Ring
- ▶ Mesh
- ▶ Torus



Intel Sandy Bridge (3.5GHz,
6 cores, 2 threads per core)

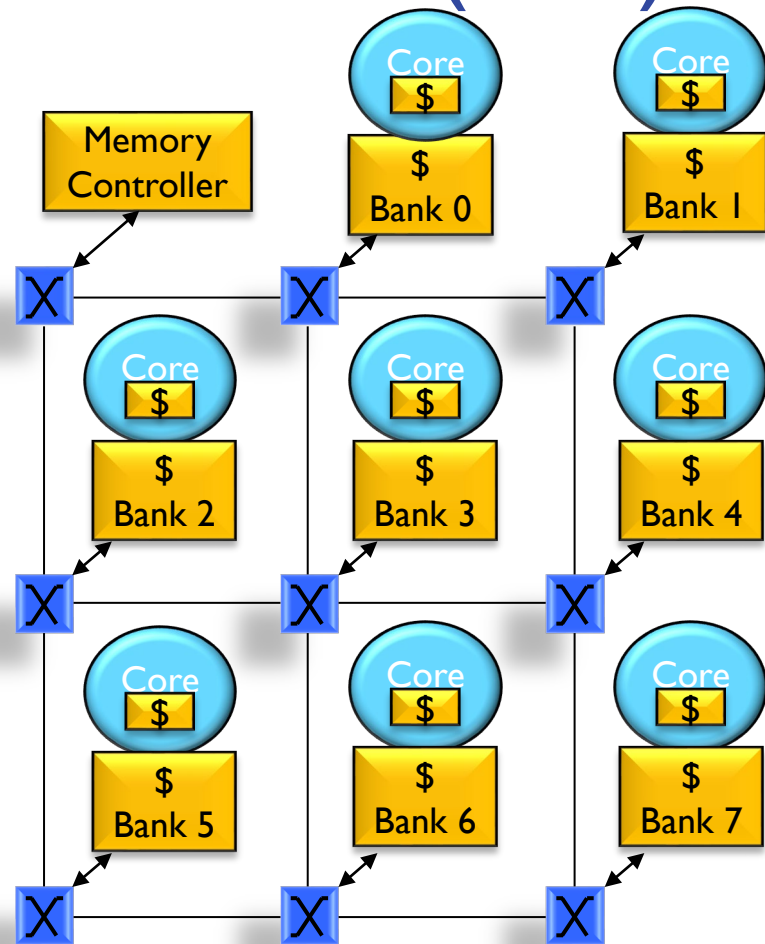


- 3 ports per switch
- Smaller number of switches
- Simple and cheap
- Can be bi-directional to reduce latency

On-chip Interconnects (4/5)

■ Possible topologies

- ▶ Bus
- ▶ Crossbar
- ▶ Ring
- ▶ Mesh
- ▶ Torus



- Up to 5 ports per switch

Tilera Tile64 (866MHz, 64 cores)

