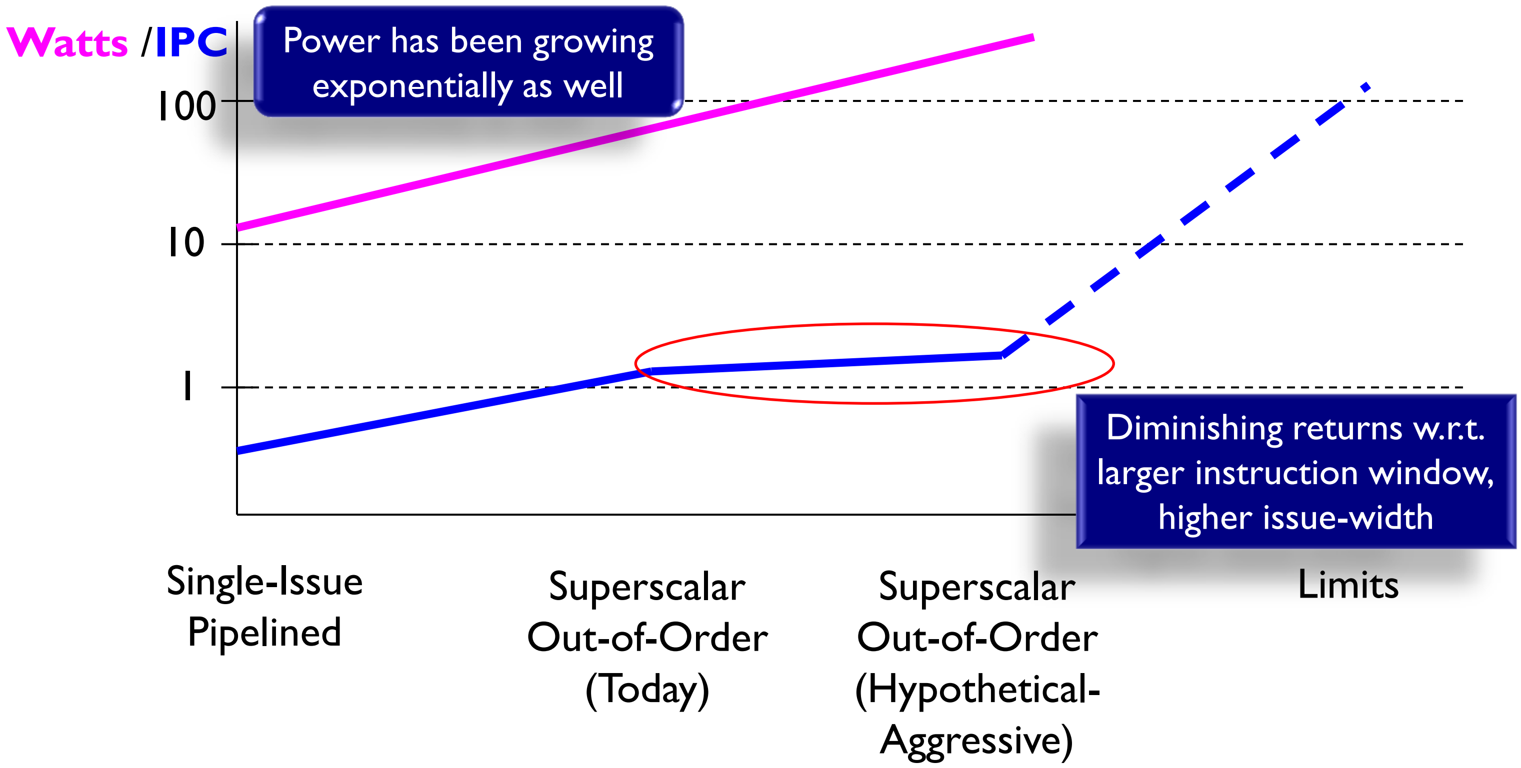
CSE 502: Computer Architecture

Multi-{Socket,Core,Thread}

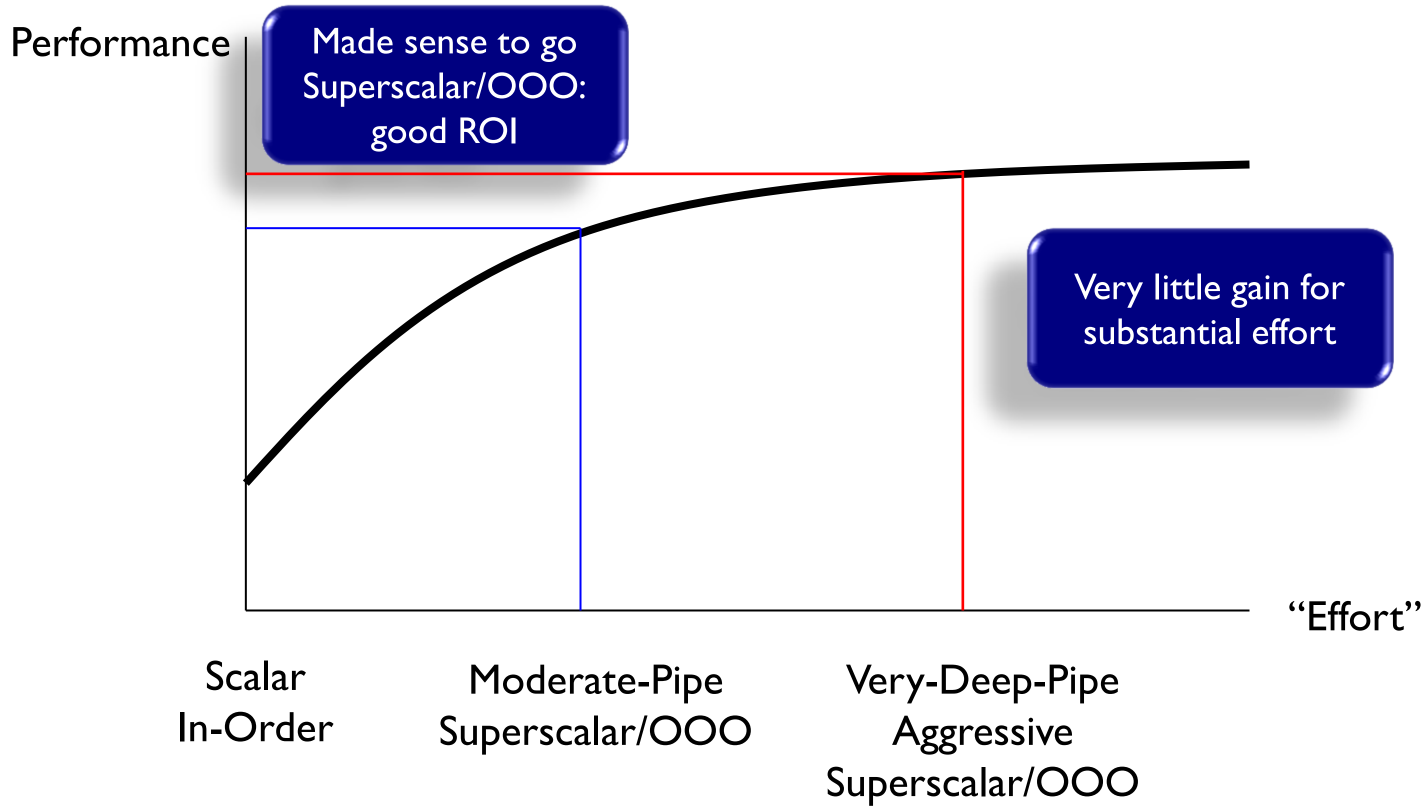
Getting More Performance

- Keep pushing IPC and/or frequenecy
  - Design complexity (time to market)
  - Cooling (cost)
  - Power delivery (cost)
  - …
- Possible, but too costly

Bridging the Gap



Higher Complexity not Worth Effort



User Visible/Invisible

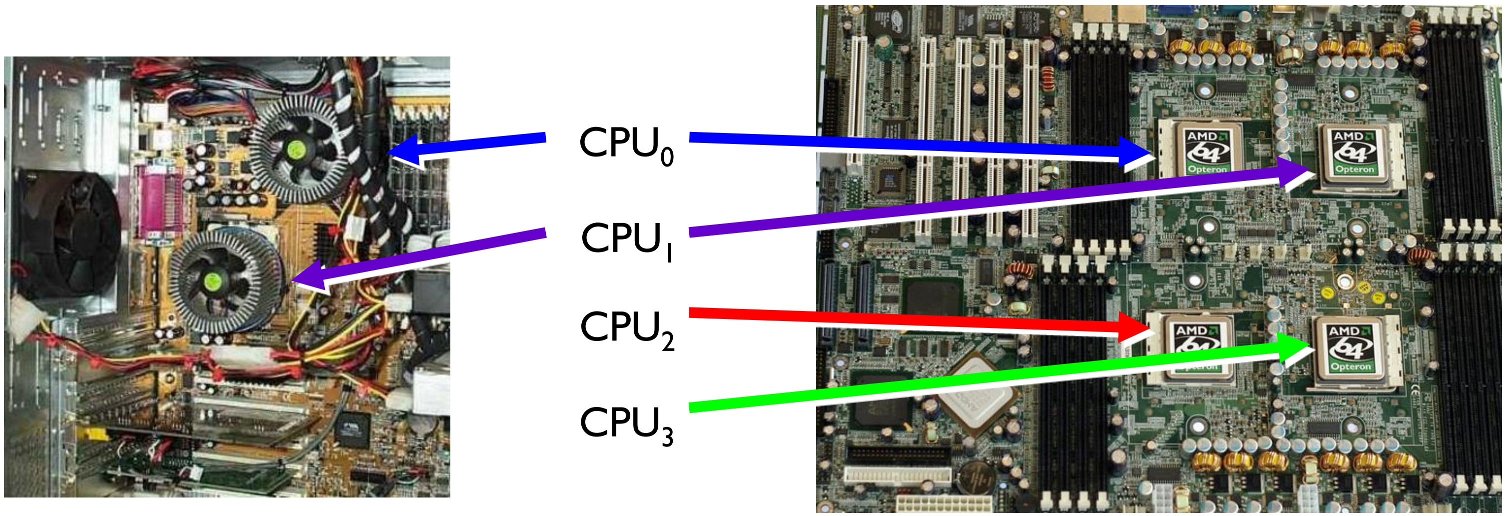
- All performance gains up to this point were “free”
  - No user intervention required (beyond buying new chip)
    - Recompilation/rewriting could provide even more benefit
  - Higher frequency & higher IPC
  - Same ISA, different micro-architecture
- Multi-processing pushes parallelism above ISA
  - Coarse grained parallelism
    - Provide multiple processing elements
  - User (or developer) responsible for finding parallelism
    - User decides how to use resources

Sources of (Coarse) Parallelism

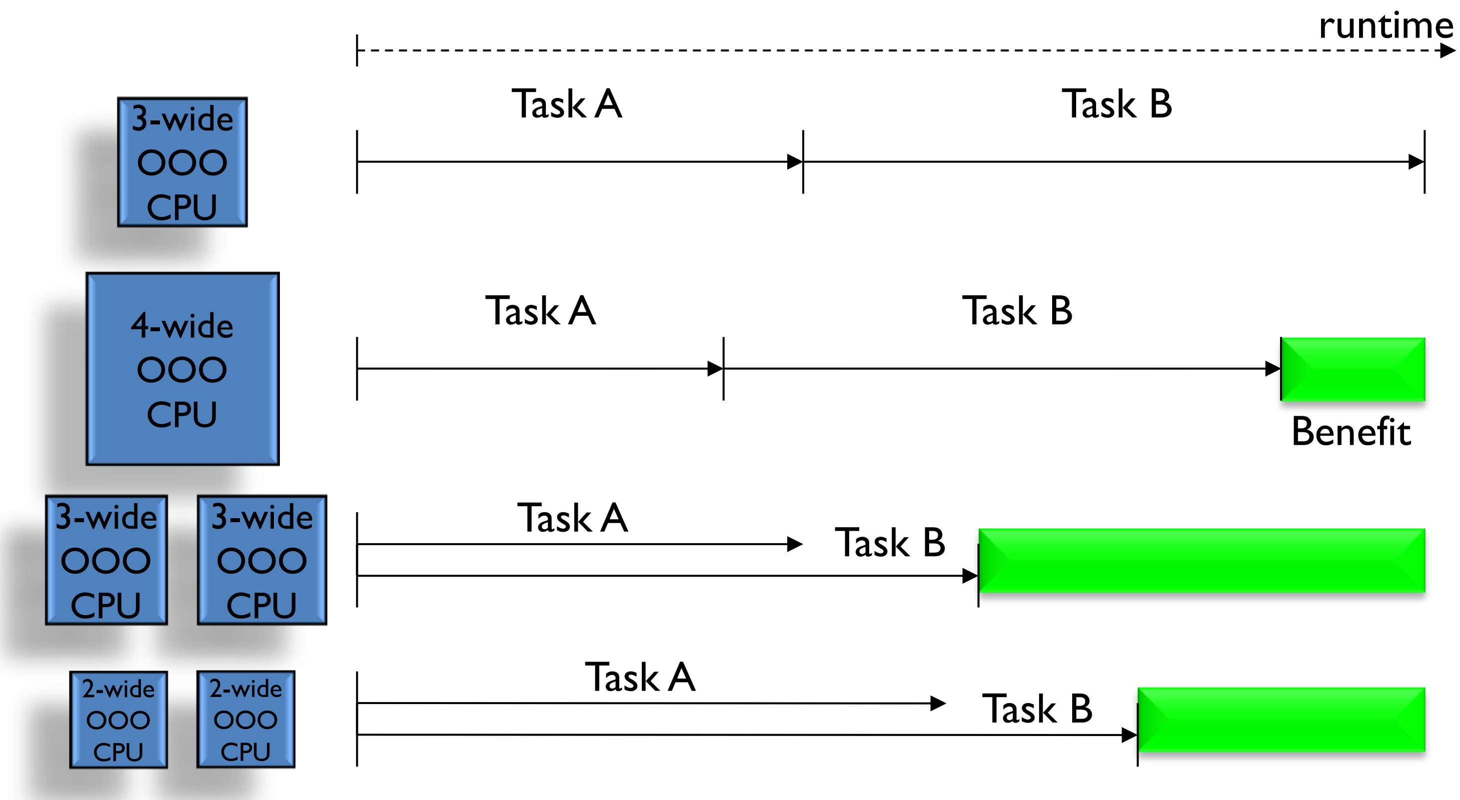
- Different applications
  - MP3 player in background while you work in Office
  - Other background tasks: OS/kernel, virus check, etc…
  - Piped applications
    - gunzip -c foo.gz | grep bar | perl some-script.pl
- Threads within the same application
  - Java (scheduling, GC, etc...)
  - Explicitly coded multi-threading
    - pthreads, MPI, etc…

SMP Machines

- *SMP* = Symmetric Multi-Processing
  - Symmetric = All CPUs have “equal” access to memory
- OS seems multiple CPUs
  - Runs one process (or thread) on each CPU

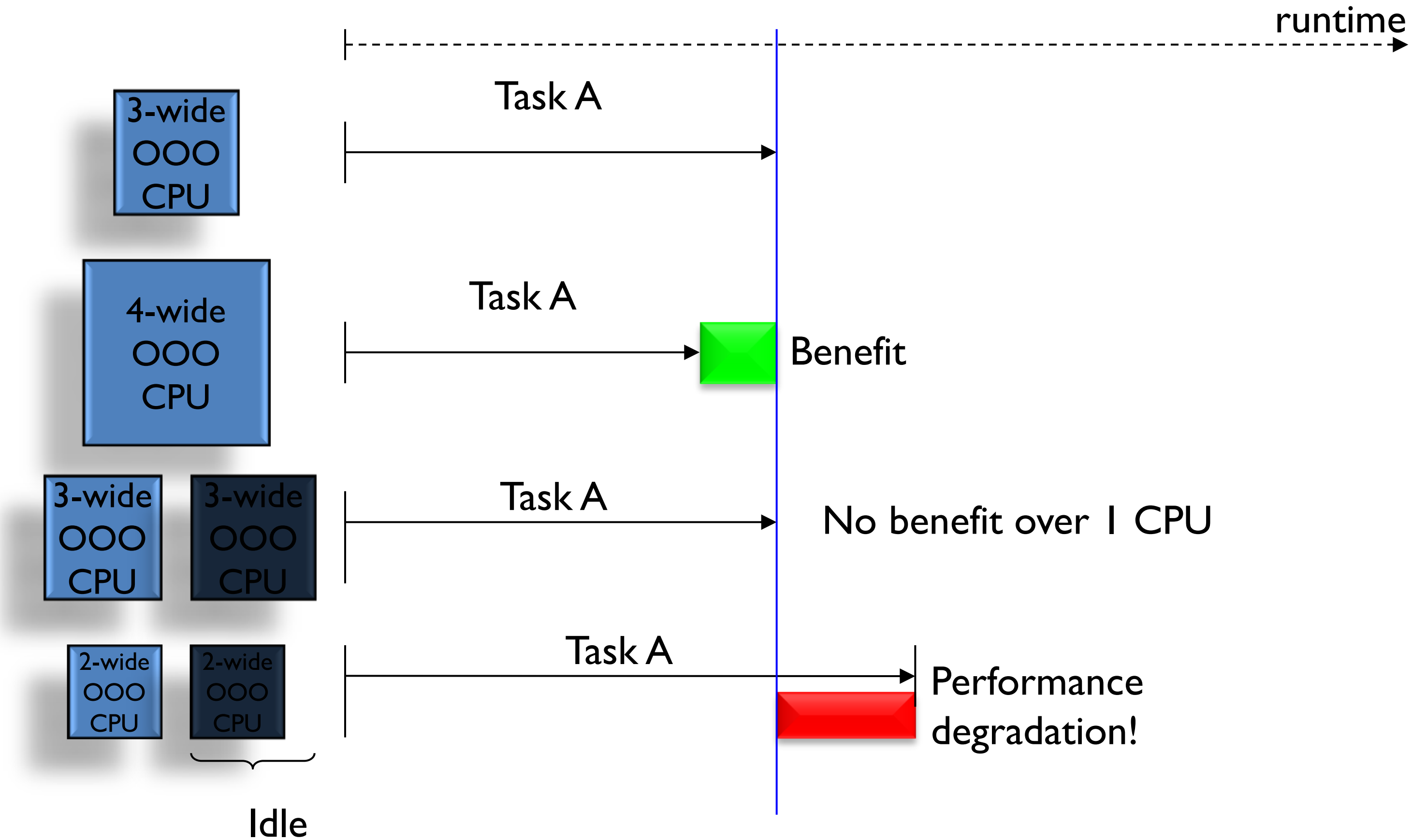


MP Workload Benefits



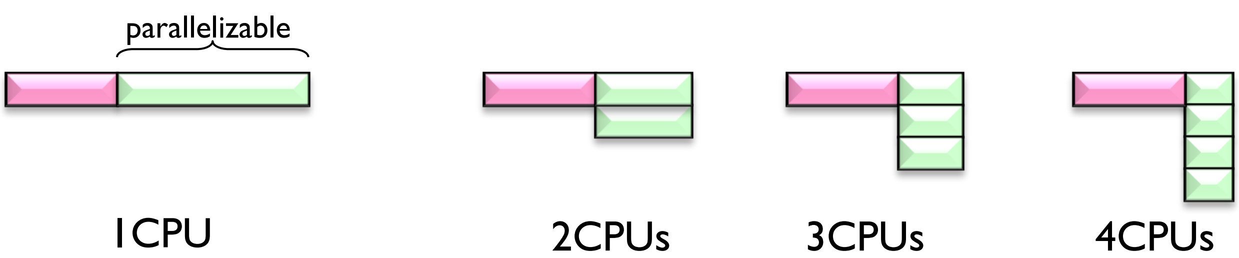
Assumes you have multiple tasks/programs to run

… If Only One Task Available



Benefit of MP Depends on Workload

- Limited number of parallel tasks to run on PC
  - Adding more CPUs than tasks provide zero benefit
- For parallel code, Amdahl's law curbs speedup

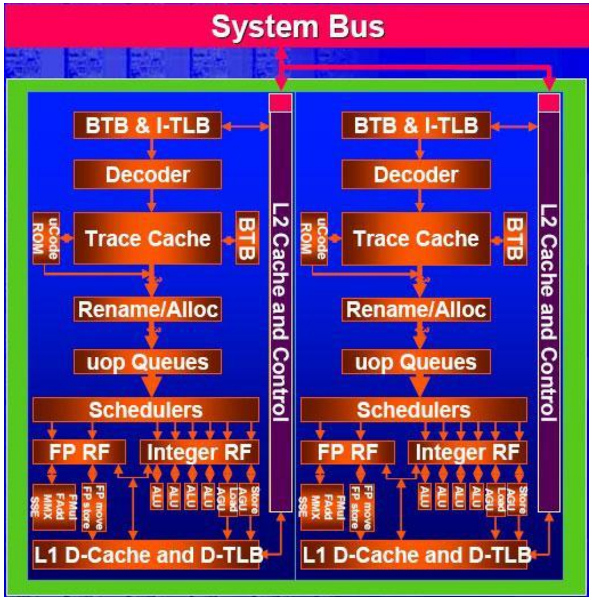


Hardware Modifications for SMP

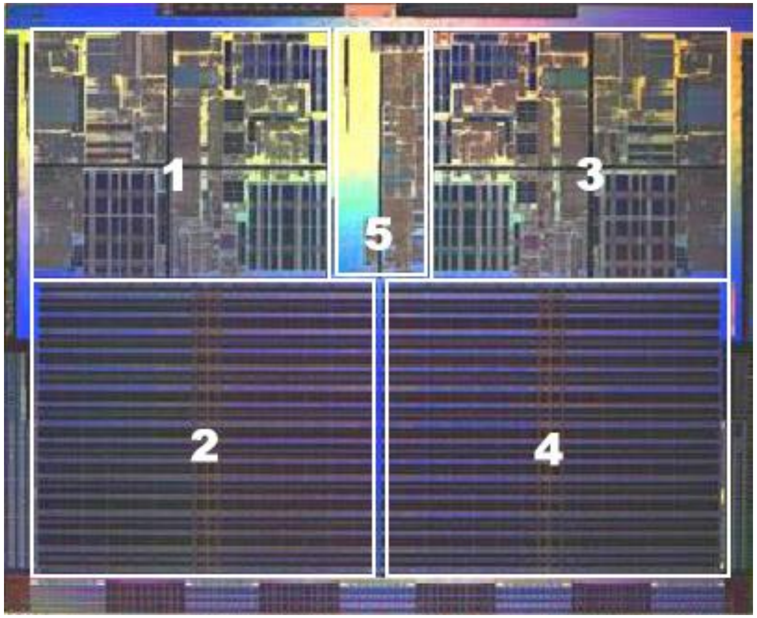
- Processor
  - Memory interface
- Motherboard
  - Multiple sockets (one per CPU)
  - Datapaths between CPUs and memory
- Other
  - Case: larger (bigger motherboard, better airflow)
  - Power: bigger power supply for N CPUs
  - Cooling: more fans to remove N CPUs' worth of heat

Chip-Multiprocessing (*CMP*)

- Simple SMP on the same chip
  - CPUs now called “cores” by hardware designers
  - OS designers still call these “CPUs”



Intel “Smithfield” Block Diagram



AMD Dual-Core Athlon FX

Benefits of CMP

- Cheaper than multi-chip SMP
  - All/most interface logic integrated on chip
    - Fewer chips
    - Single CPU socket
    - Single interface to memory
  - Less power than multi-chip SMP
    - Communication on-die is more power-efficient than chip-to-chip
- Efficiency
  - Use for transistors instead of wider/more aggressive OoO
  - Potentially better use of hardware resources

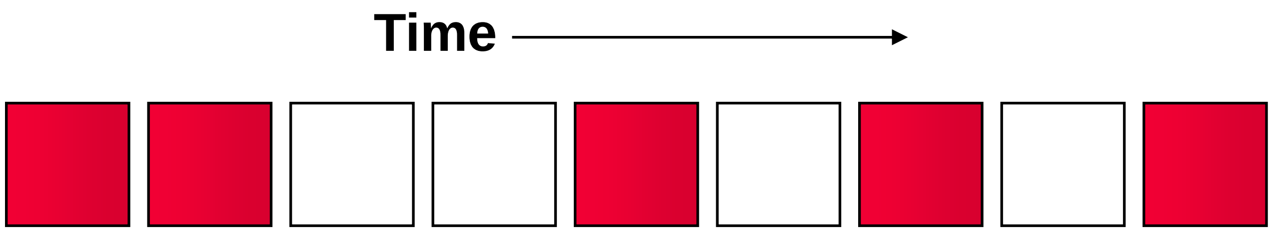
CMP Performance vs. Power

- 2x CPUs not necessarily equal to 2x performance
- 2x CPUs → ½ power for each
  - Maybe a little better than ½ if resources can be shared
- Back-of-the-Envelope calculation:
  - 3.8 GHz CPU at 100W
  - Dual-core: 50W per CPU
  - $P \propto V^3$: $V_{orig}{}^3/V_{CMP}{}^3 = 100W/50W$ → $V_{CMP} = 0.8\ V_{orig}$
  - $f \propto V$: $f_{CMP} = 3.0GHz$

Multi-Threading

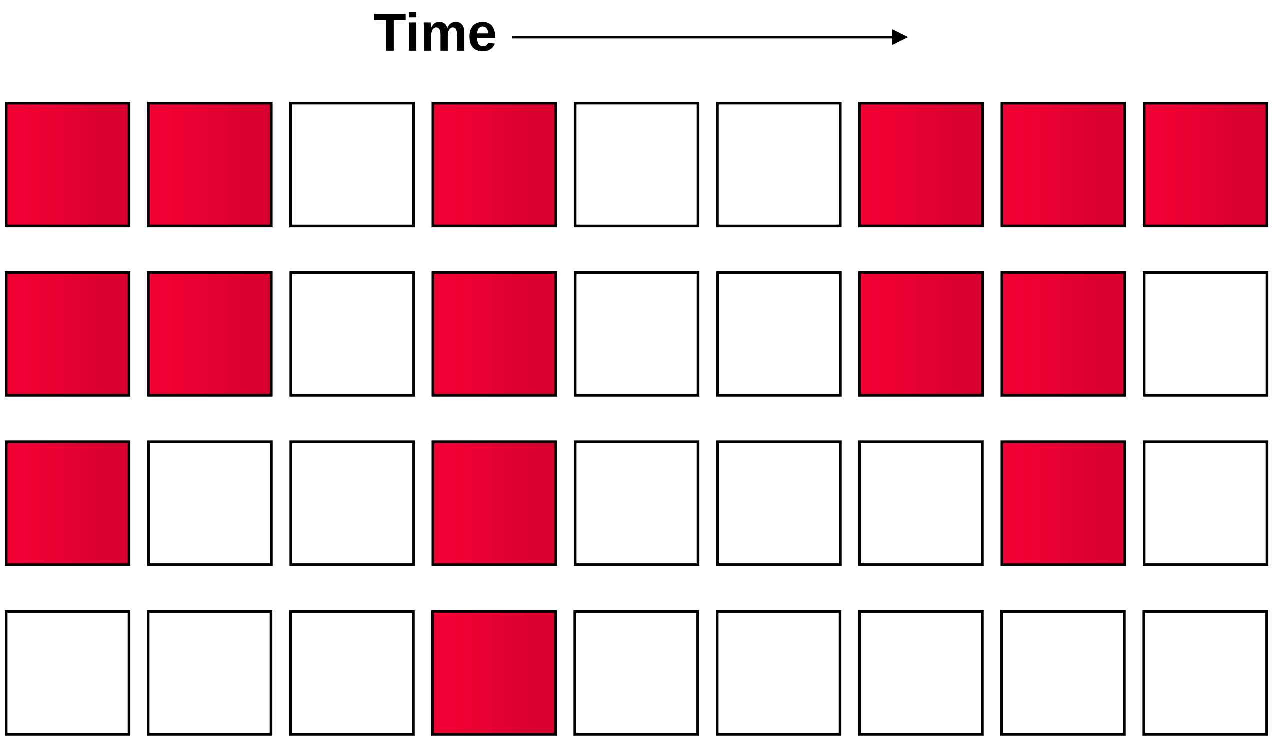
- Uni-Processor: 4-6 wide, lucky if you get 1-2 IPC
  - Poor utilization of transistors
- SMP: 2-4 CPUs, but need independent threads
  - Poor utilization as well (if limited tasks)
- *{Coarse-Grained,Fine-Grained,Simultaneous}-MT*
  - Use single large uni-processor as a multi-processor
    - Core provide multiple hardware contexts (threads)
      - Per-thread PC
      - Per-thread ARF (or map table)
  - Each core appears as multiple CPUs
    - OS designers still call these “CPUs”

Scalar Pipeline



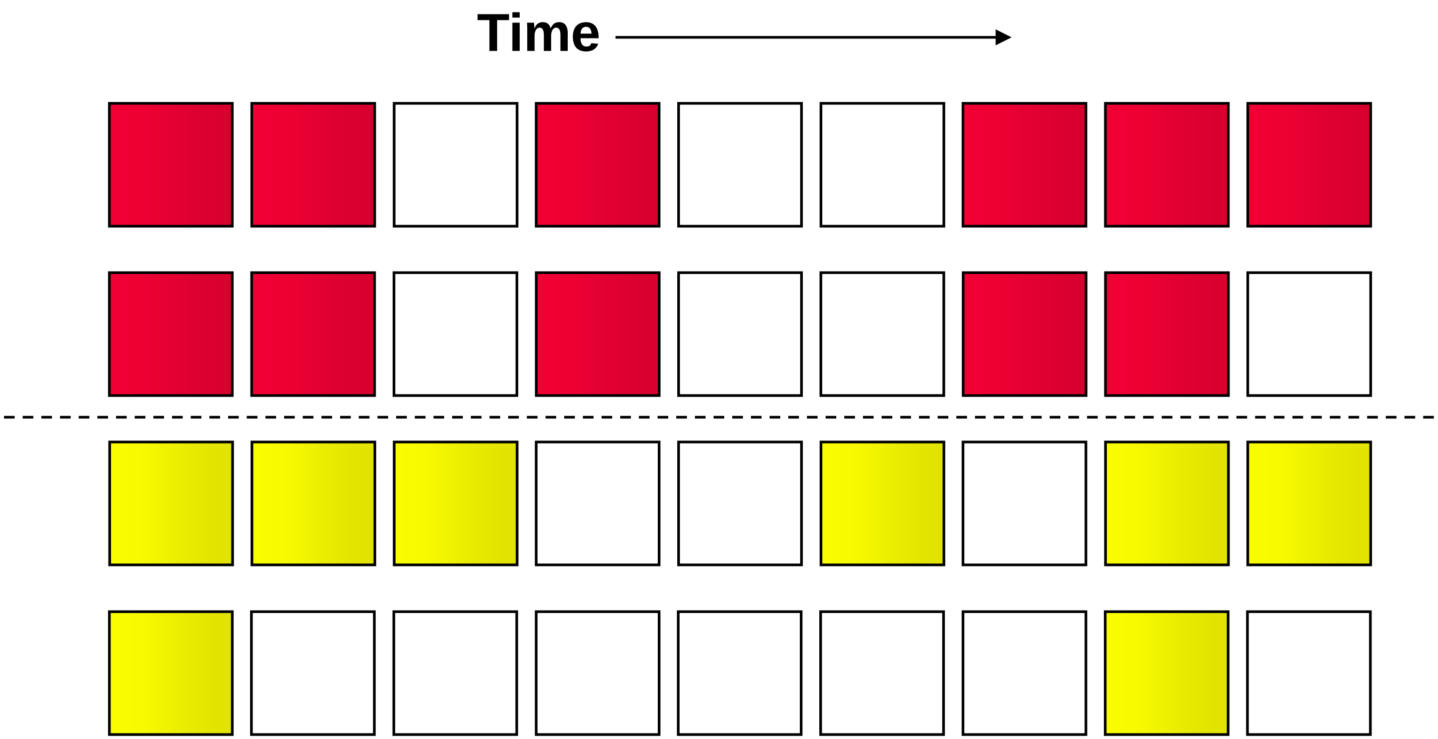
Dependencies limit functional unit utilization

Superscalar Pipeline



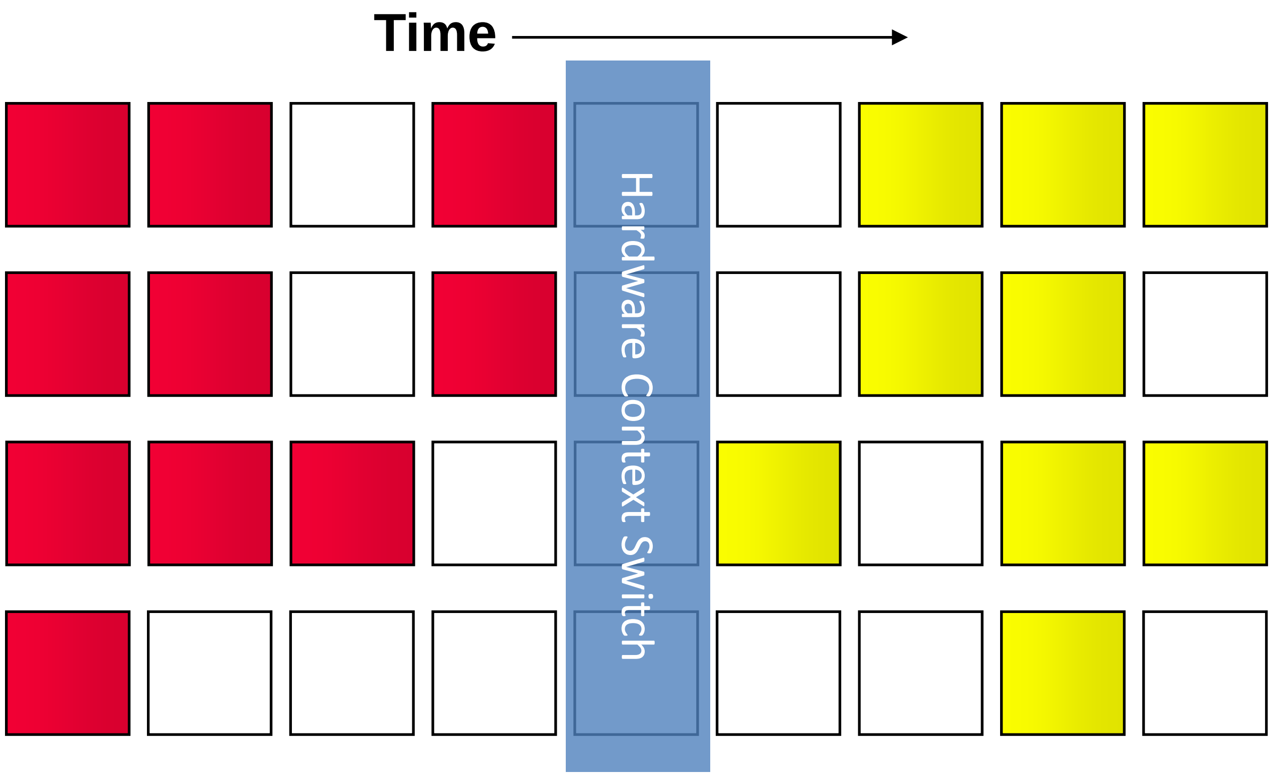
Higher performance than scalar, but lower utilization

Chip Multiprocessing (CMP)



Limited utilization when running one thread

Coarse-Grained Multithreading (1/3)



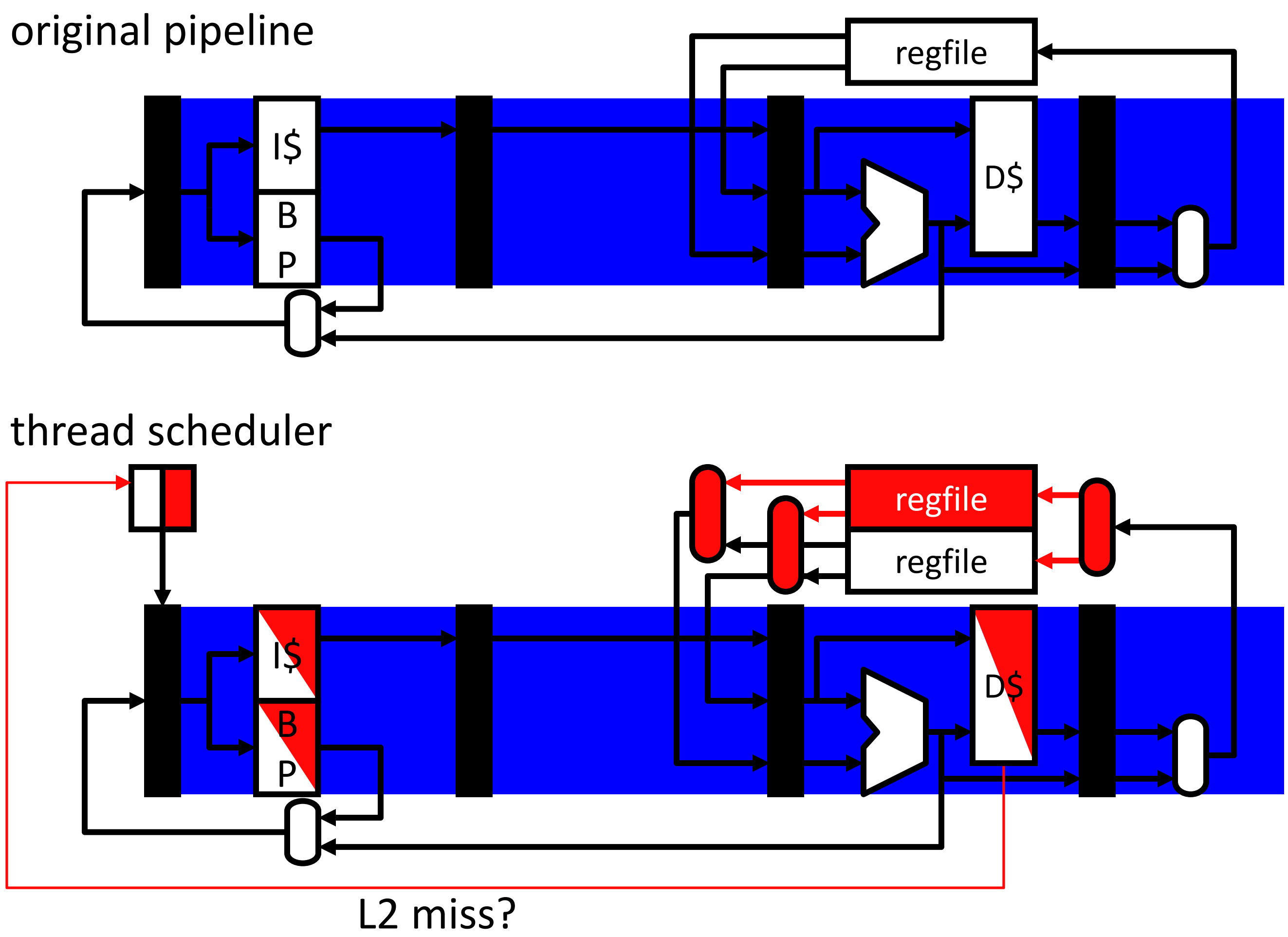
Only good for long latency ops (i.e., cache misses)

Coarse-Grained Multithreading (2/3)

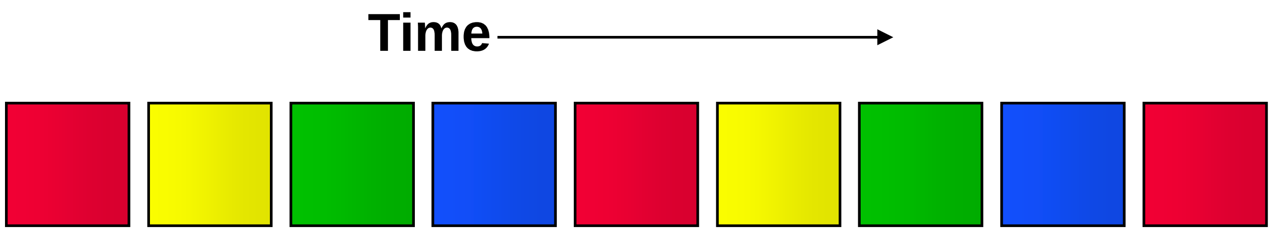
+ Sacrifices a little single thread performance
– Tolerates only long latencies (e.g., L2 misses)

- 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

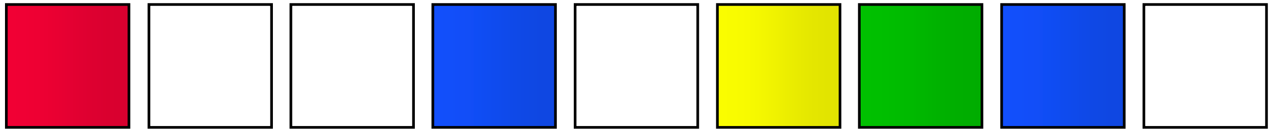
Coarse-Grained Multithreading (3/3)



Fine-Grained Multithreading (1/3)



Saturated workload -> Lots of threads



Unsaturated workload -> Lots of stalls

Intra-thread dependencies still limit performance

Fine-Grained Multithreading (2/3)

– Sacrifices significant single-thread performance

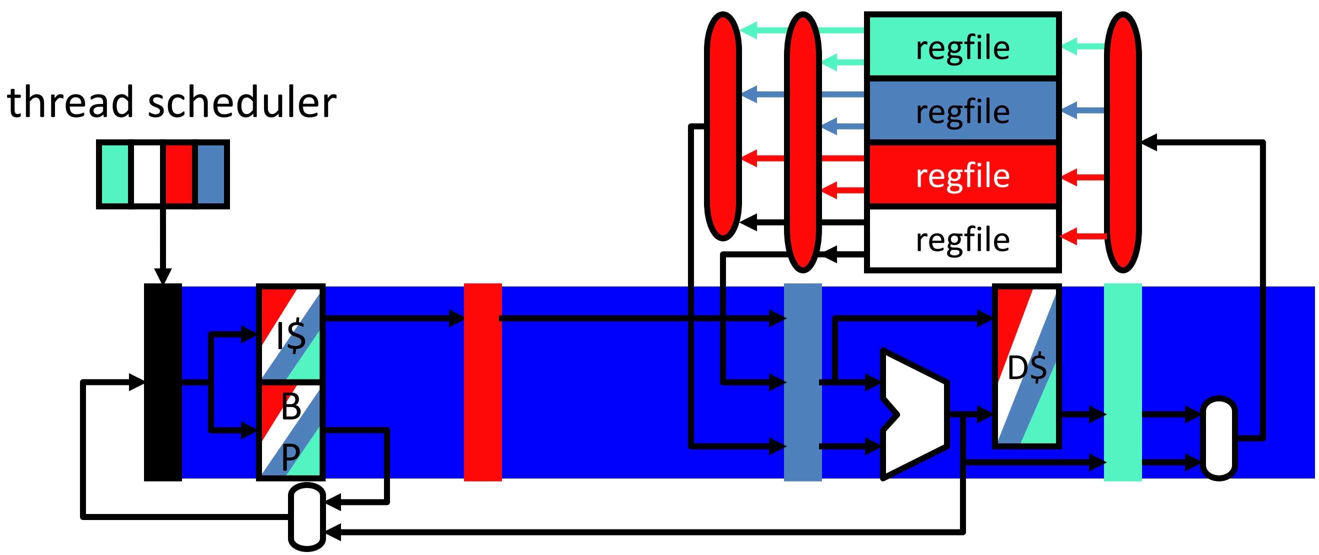
\+ Tolerates everything

  + L2 misses
  + Mispredicted branches
  + etc...

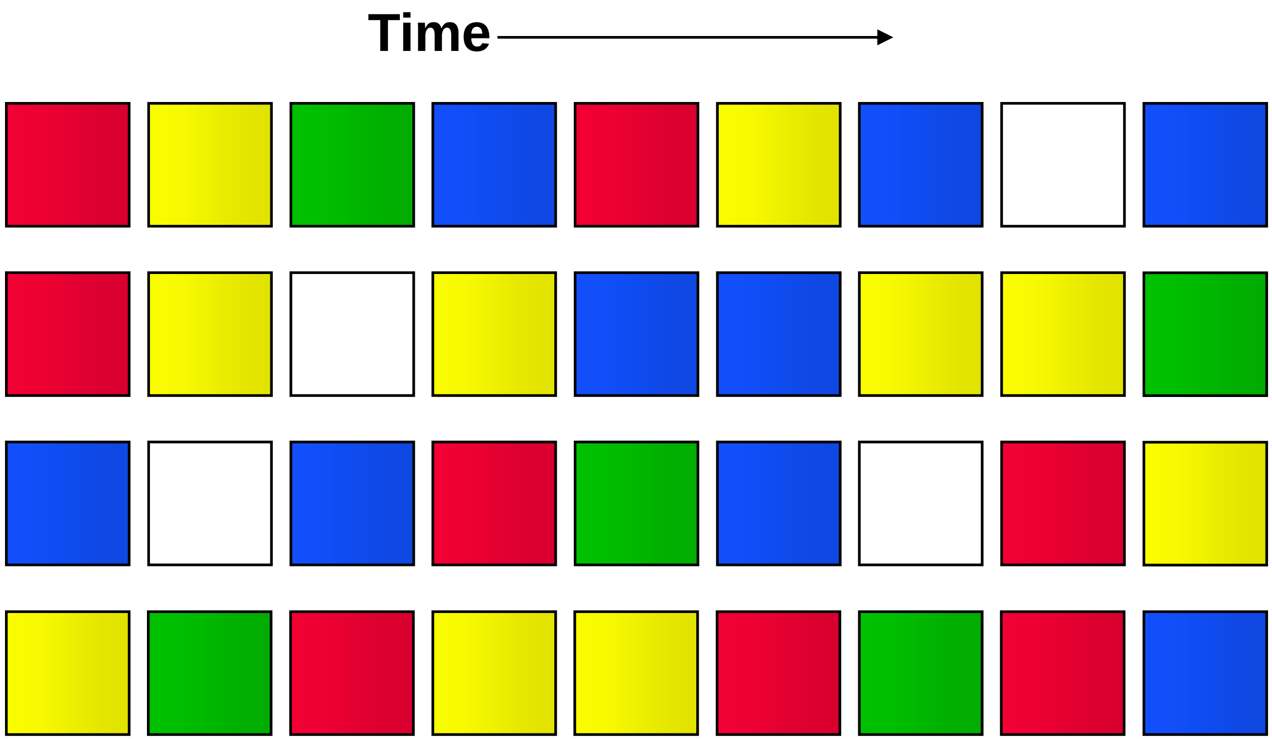
- Thread scheduling policy
  - Switch threads often (e.g., every cycle)
  - Use round-robin policy, skip threads with long-latency ops
- Pipeline partitioning
  - Dynamic, no flushing
  - Length of pipeline doesn’t matter

Fine-Grained Multithreading (3/3)

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



Simultaneous Multithreading (1/3)

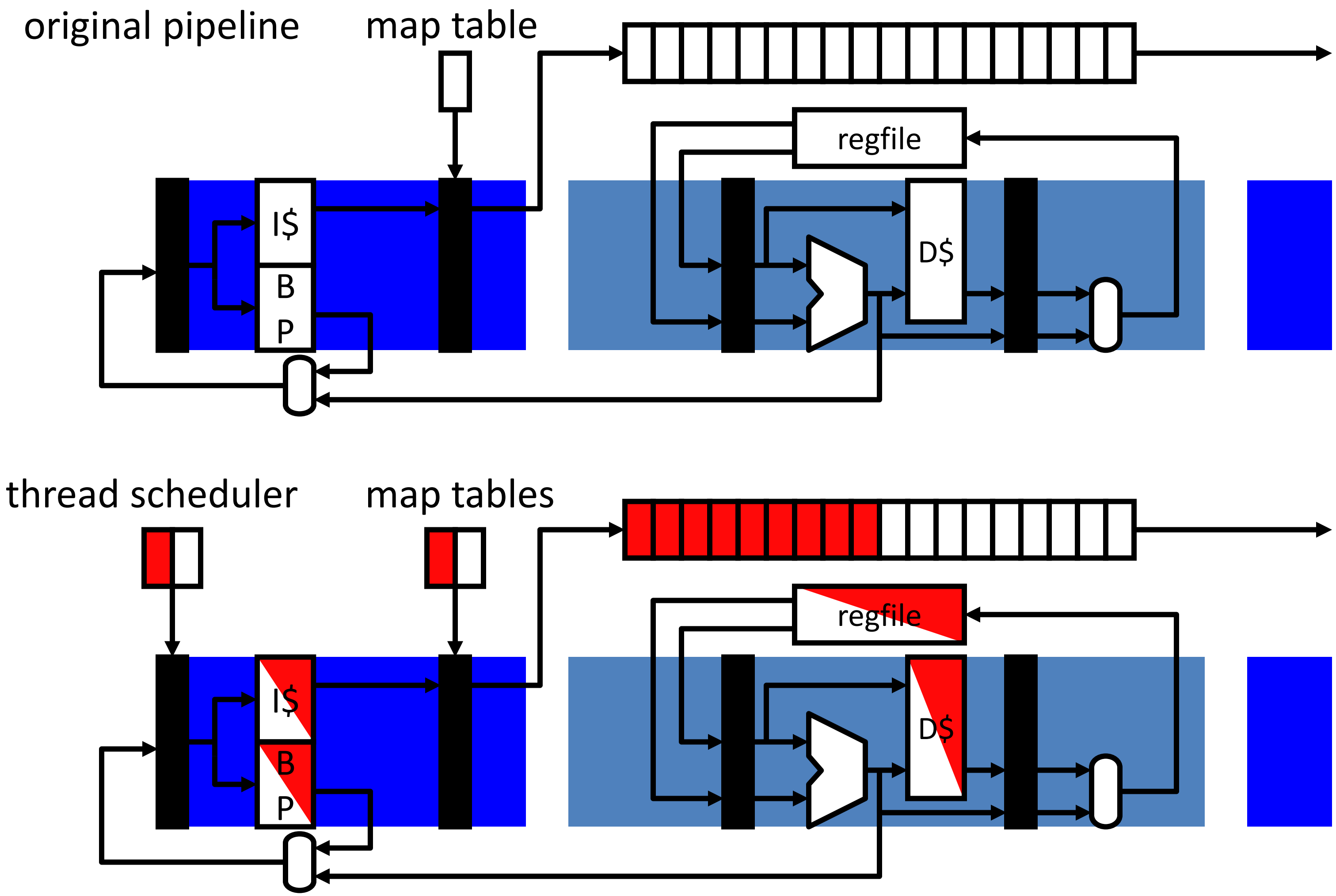


Max utilization of functional units

Simultaneous Multithreading (2/3)

+ Tolerates all latencies
± Sacrifices some single thread performance
– Thread scheduling policy
    - Round-robin (like Fine-Grained MT)
– Pipeline partitioning
    - Dynamic
– Examples
    – Pentium4 (*hyper-threading*): 5-way issue, 2 threads
    – Alpha 21464: 8-way issue, 4 threads (canceled)

Simultaneous Multithreading (3/3)



Issues for SMT

- Cache interference
  - Concern for all MT variants
  - Shared memory SPMD threads help here
    - Same insns. → share I$
    - Shared data → less D$ contention
    - MT is good for “server” workloads
  - SMT might want a larger L2 (which is OK)
    - Out-of-order tolerates L1 misses
- Large map table and physical register file
  - #maptable-entries = (#threads * #arch-regs)
  - #phys-regs = (#threads * #arch-regs) + #in-flight insns

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

Benefits of MT depend on workload

CMP vs MT

- If you wanted to run multiple threads would you build a…
  - Chip multiprocessor (CMP): multiple separate pipelines?
  - A multithreaded processor (MT): 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
    - SMT is basically an ILP engine that converts TLP to ILP
    - CMP is mainly a TLP engine
- Do both (CMP of MTs), e.g., Sun UltraSPARC T1
  - 8 processors, each with 4-threads (fine-grained threading)
  - 1Ghz clock, in-order, short pipeline
  - Designed for power-efficient “throughput computing”

Combining MP Techniques (1/2)

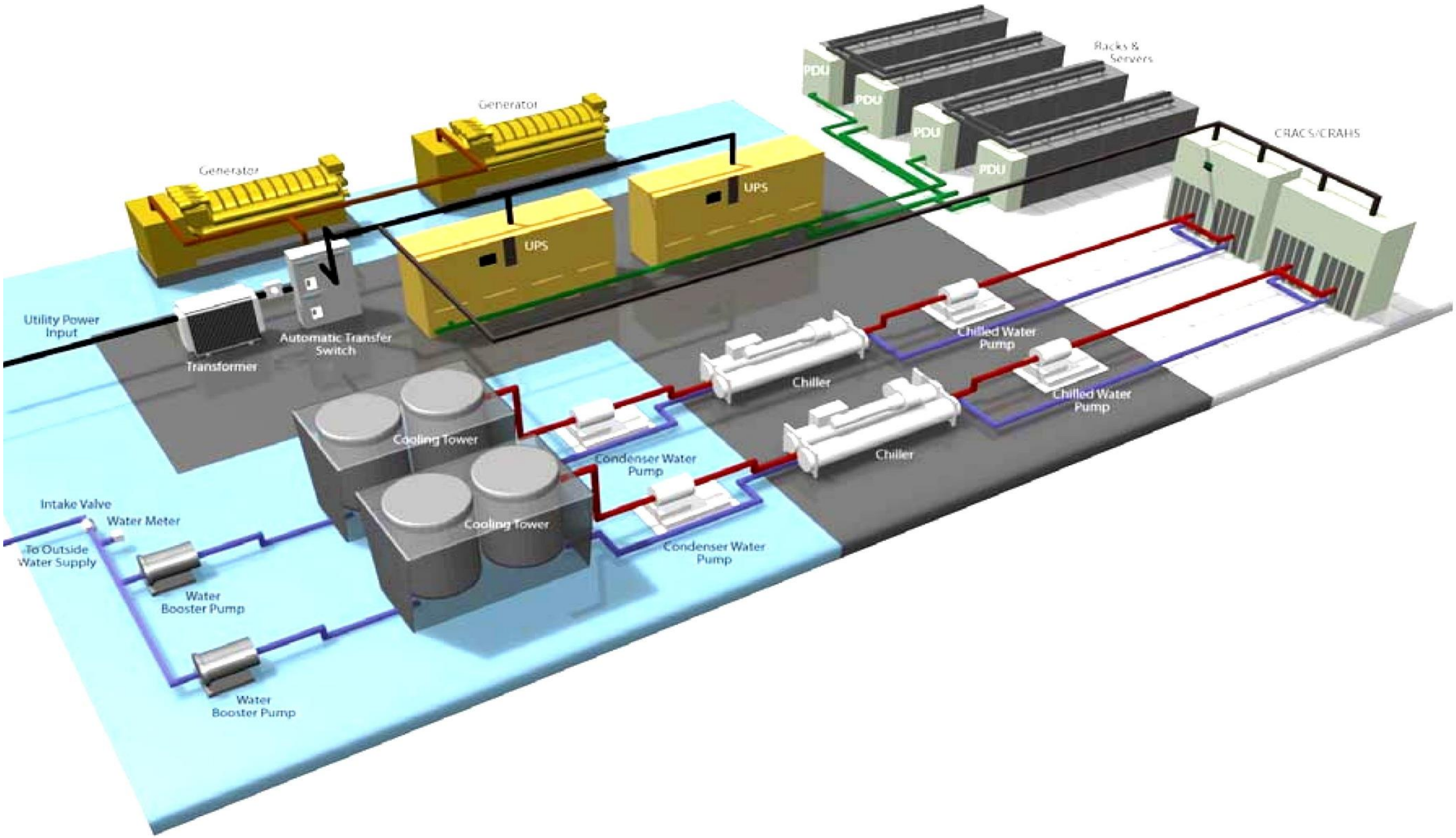
- System can have SMP, CMP, and SMT at the same time
- Example machine with 32 threads
  - Use 2-socket SMP motherboard with two chips
  - Each chip with an 8-core CMP
  - Where each core is 2-way SMT
- Makes life difficult for the OS scheduler
  - OS needs to know which CPUs are…
    - Real physical processor (SMP): highest independent performance
    - Cores in same chip: fast core-to-core comm., but shared resources
    - Threads in same core: competing for resources
  - Distinct apps. scheduled on different CPUs
  - Cooperative apps. (e.g., pthreads) scheduled on same core
  - Use SMT as last choice (or don't use for some apps.)

Combining MP Techniques (2/2)

Scalability Beyond the Machine

Server Racks

Datacenters (1/2)



Datacenters (2/2)