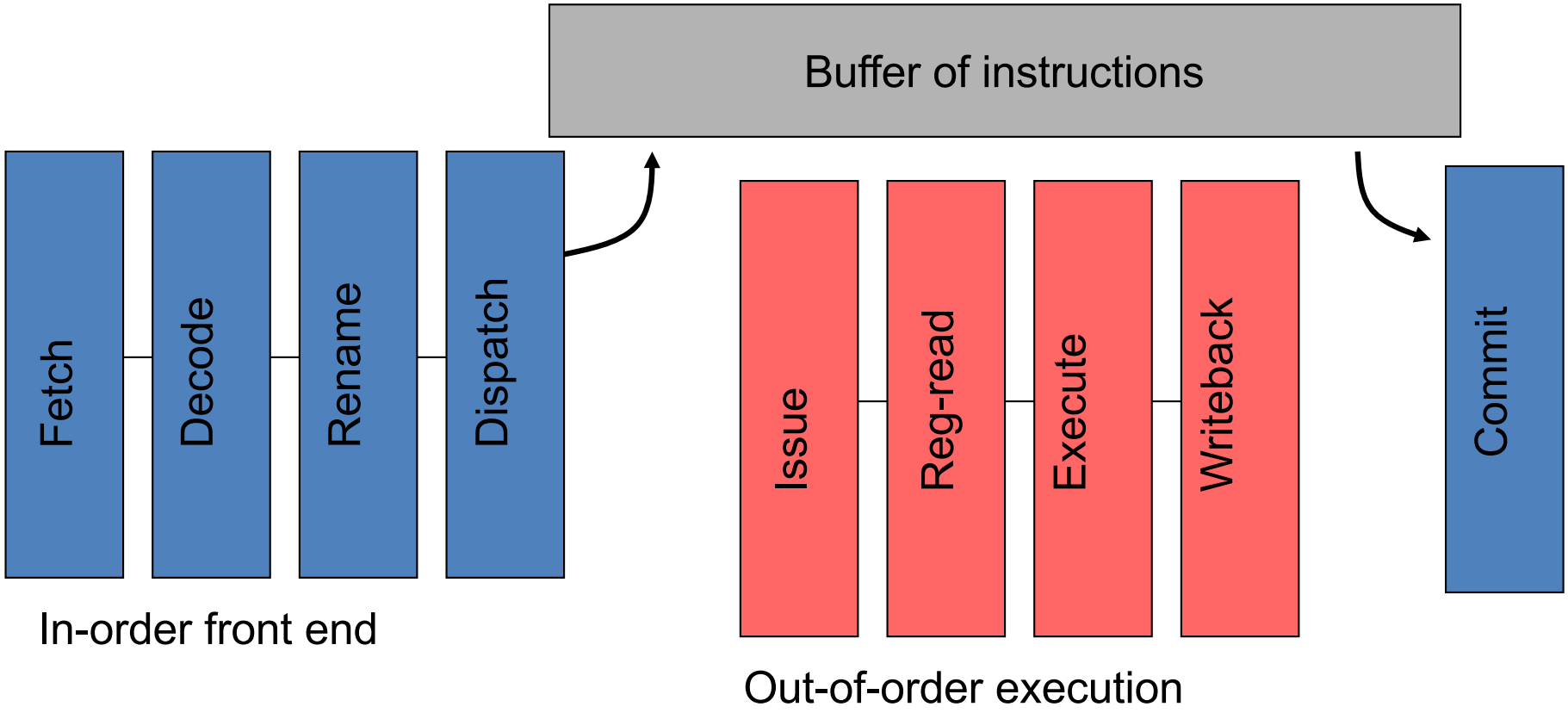
CSEP 548: Computer Systems Architecture

Hardware Multithreading (SMT)

Luis Ceze, Spring 2017

based on slides from friends at UPenn, UIUC, UW, MIT.

Out-of-order pipeline



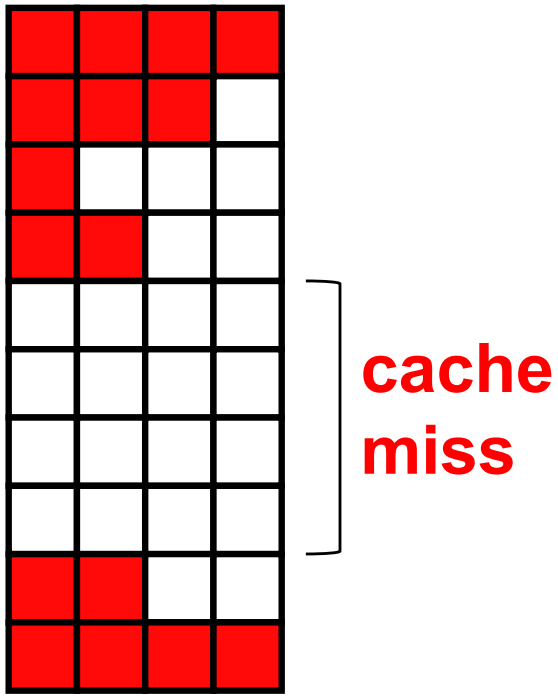
How does this tolerate latencies?

What is multithreading?

- In SW?
- In HW? How?

Superscalar Under-utilization

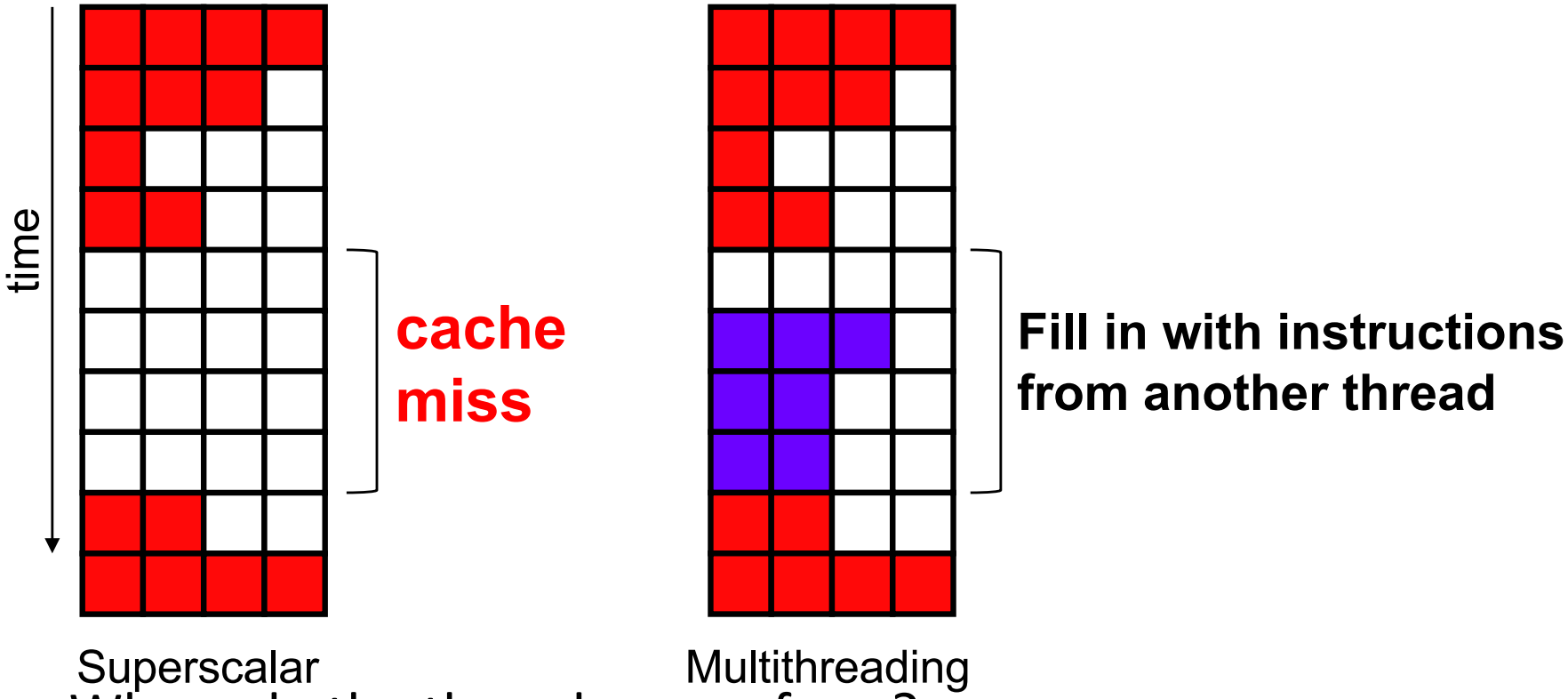
- Time evolution of issue slot
  - 4-issue processor



Superscalar

Simple Multithreading

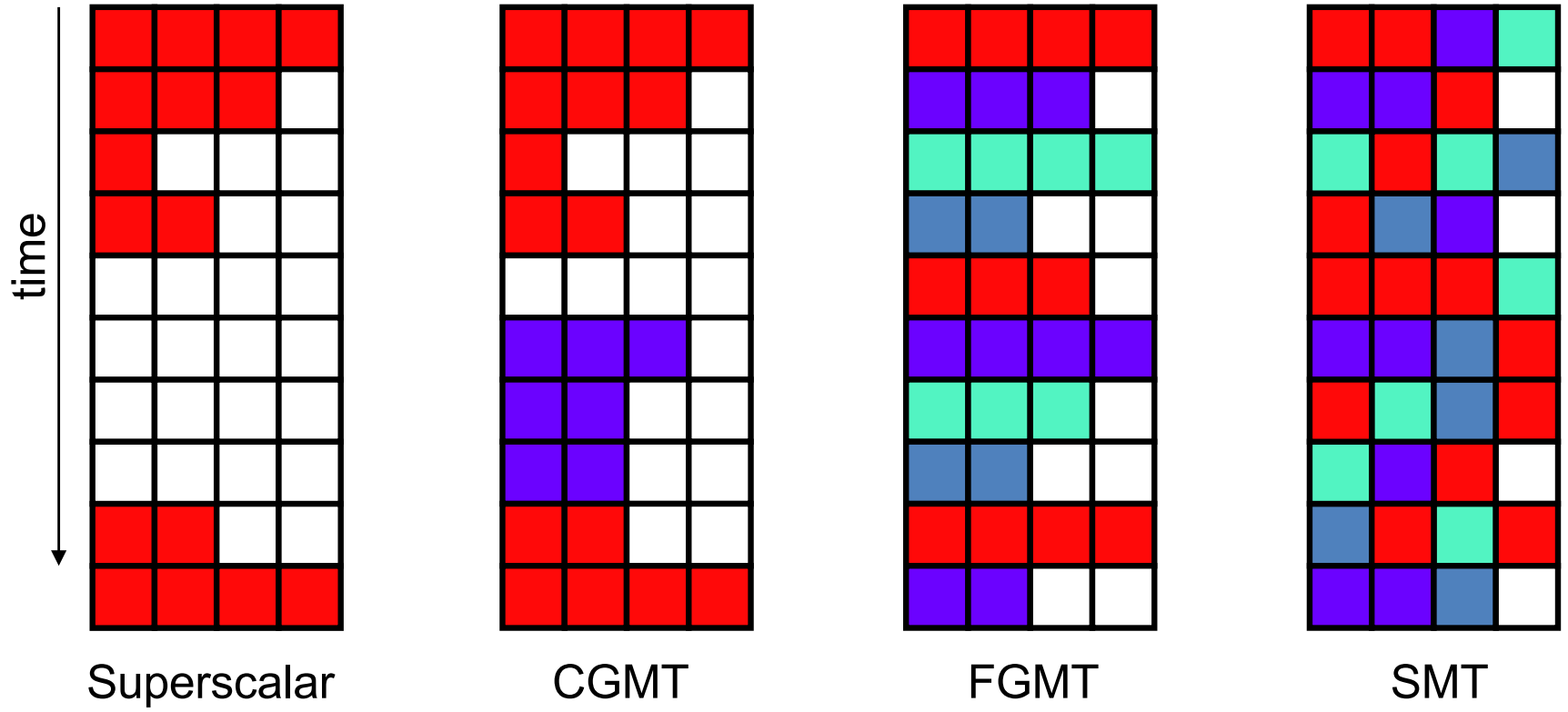
- Time evolution of issue slot
  - 4-issue processor



- Where do the threads come from?

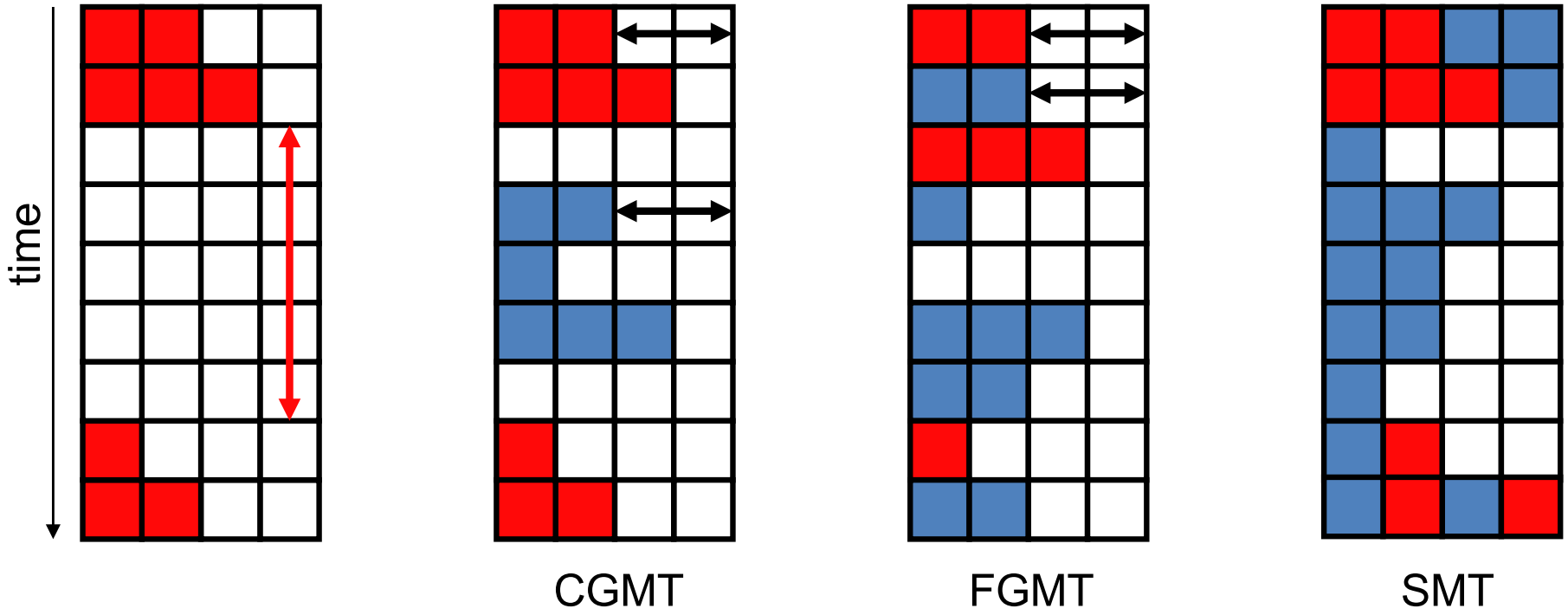
“The” Multithreading Picture

- Time evolution of issue slots
  - Color = thread



Vertical and Horizontal Under-Utilization

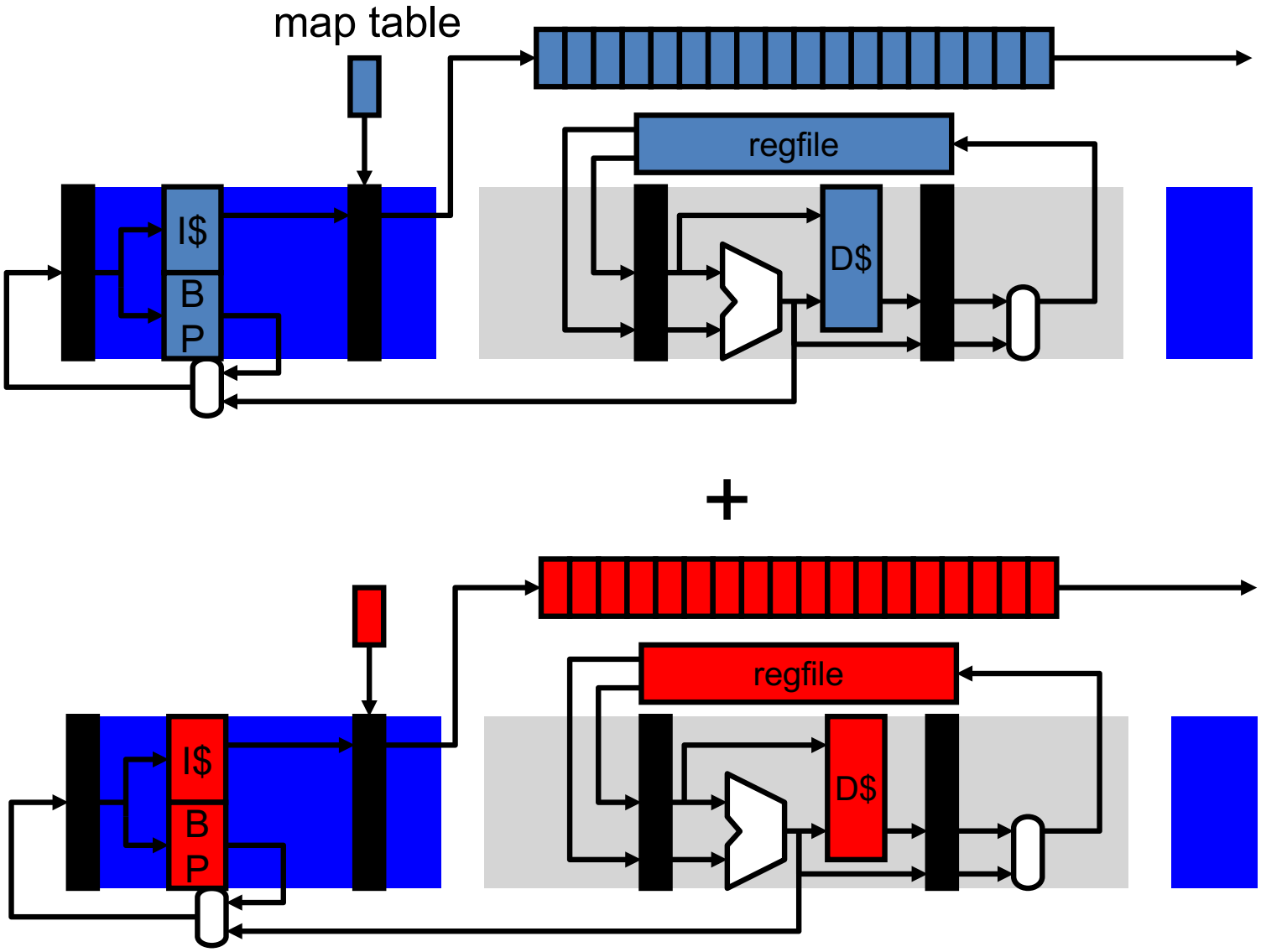
- FGMT and CGMT reduce **vertical under-utilization**
  - Loss of all slots in an issue cycle
- Do not help with **horizontal under-utilization**
  - Loss of some slots in an issue cycle (in a superscalar processor)



Simultaneous Multithreading (SMT)

- How can we issue insns from multiple threads in one cycle?

Simultaneous Multithreading (SMT)

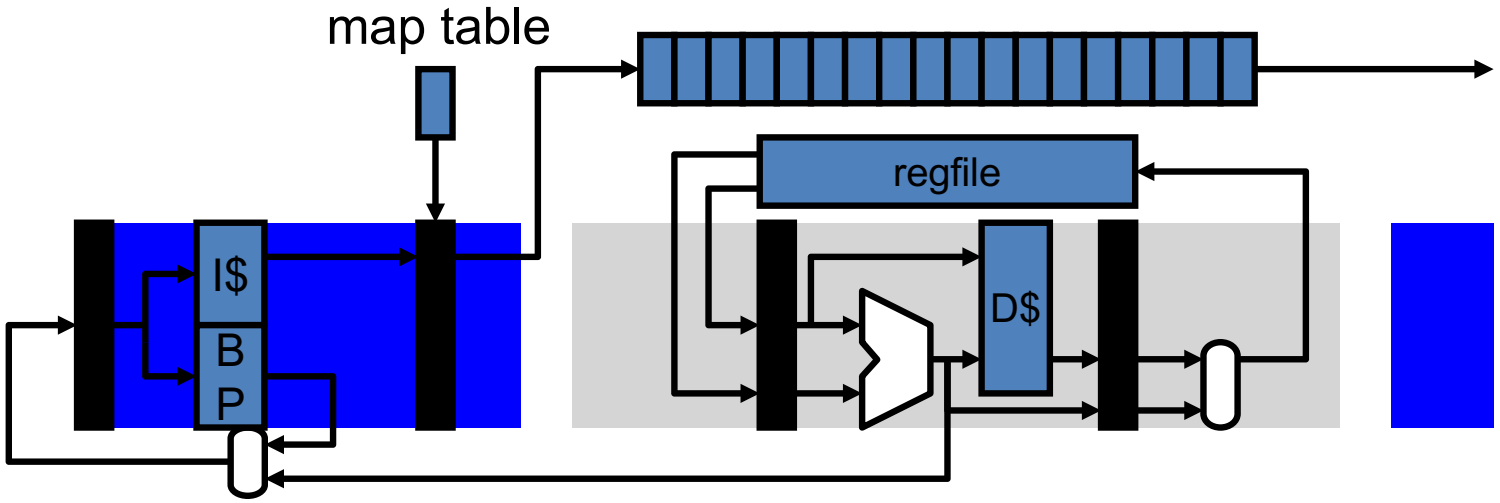


Simultaneous Multithreading (SMT)

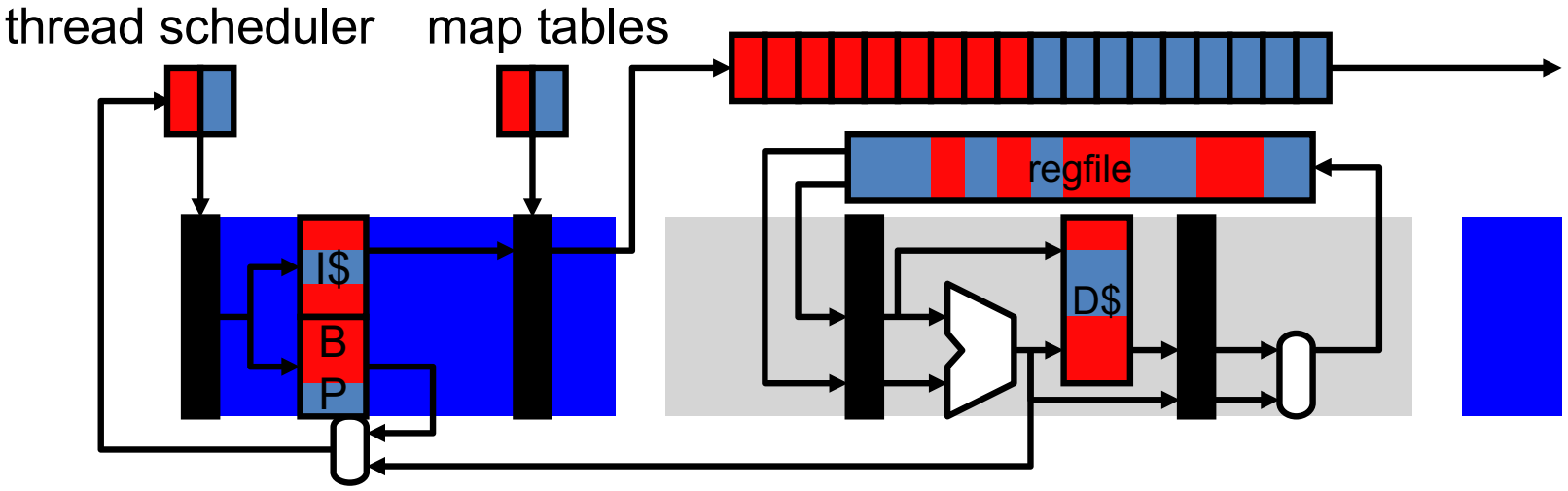
- What can issue insns from multiple threads in one cycle?
  - Same thing that issues insns from multiple parts of same program…
  - …out-of-order execution

- **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)

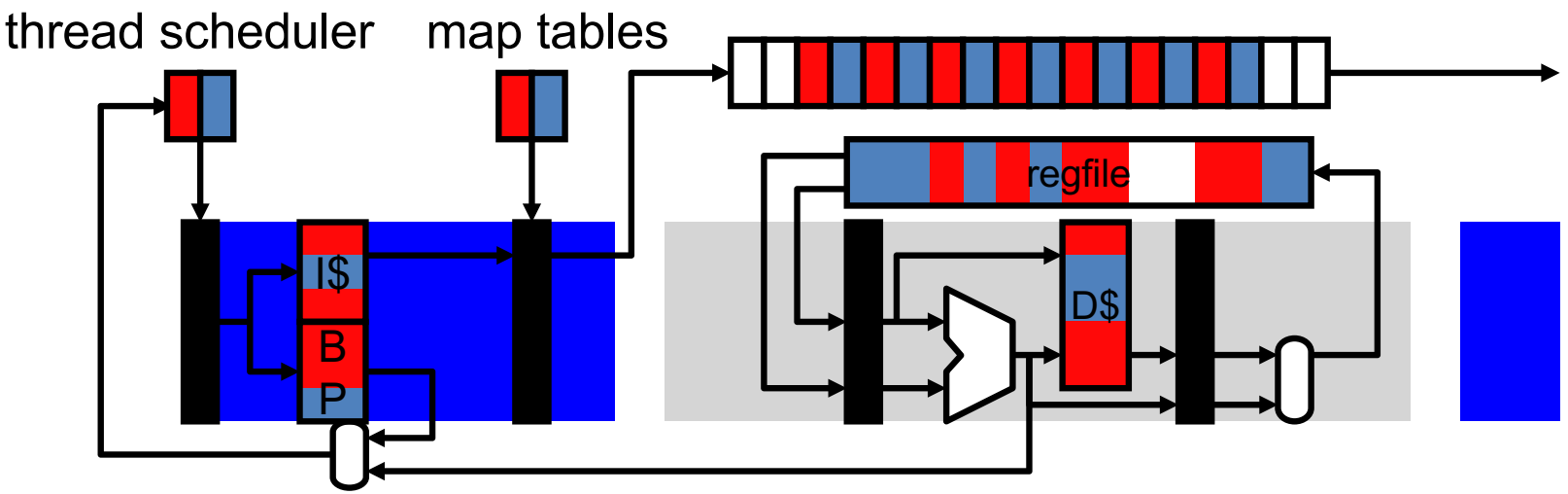


- SMT
  - Replicate map table, share (larger) physical register file



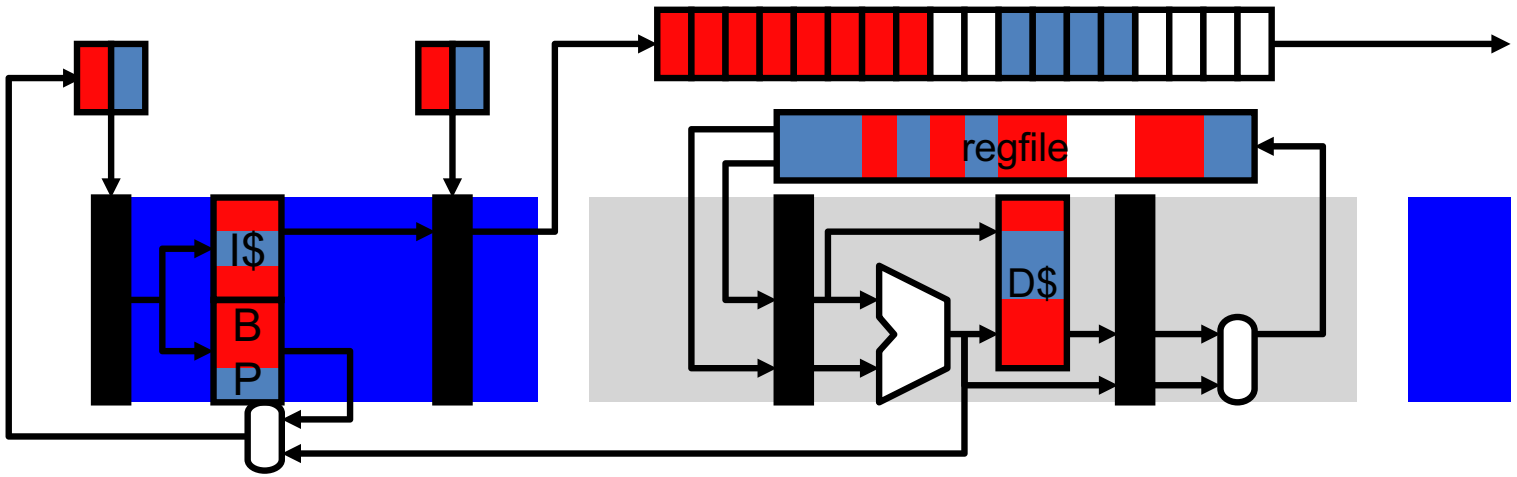
SMT Resource Partitioning

- 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 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)
- **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)

Sharing Soft State

- BTBs?
- BHT (branch history table)?
- Branch History Register (BHR)?
- Return Address Stack (RAS)?
- Caches are shared naturally…
- TLBs need explicit thread IDs to be shared, Why?
  - More on this later…

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 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)
  - 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"

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…
    - But it is one of the "big idea" from academia not yet adopted

Research: Multithreading for Reliability

- Can multithreading help with reliability?
  - Design bugs/manufacturing defects?
  - Gradual defects, e.g., thermal wear?
  - Transient errors?

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