CSE 502: Computer Architecture

Instruction Commit

The End of the Road (um… Pipe)

- Commit is typically the last stage of the pipeline
- Anything an insn. does at this point is *irrevocable*
  - Only actions following sequential execution allowed
  - E.g., wrong path instructions may not commit
    - They do not exist in the sequential execution

Everything In-Order

- ISA defines program execution in sequential order
- To the outside, CPU must appear to execute in order
- What does it mean to "appear"?
  - … when someone looks
  - ok, so what does it mean to "look"?

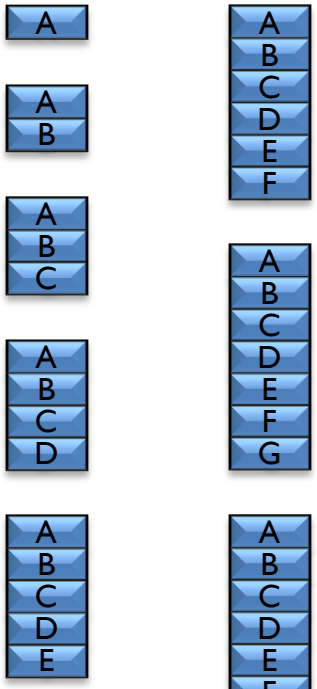
"Looking" at CPU State

- When OS swaps contexts
  - OS saves the current program state (requires "looking")
  - Allows restoring the state later
- When program has a fault (e.g., page fault)
  - OS steps in and "looks" at the "current" CPU state

- Superscalar must "retire" at least N insns. per cycle
  - Update processor state same as sequential execution

Superscalar Commit is like Sampling

Scalar Commit Processor States



Superscalar Commit Processor States

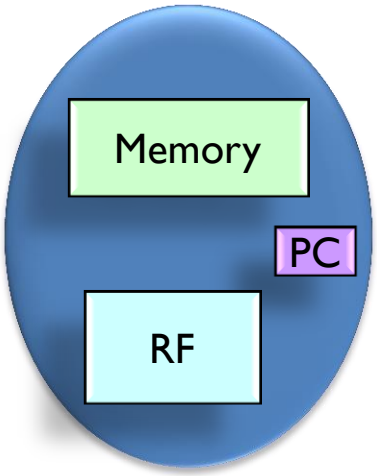
Each "state" in the superscalar machine always corresponds to one state of the scalar machine (but not necessarily the other way around), and the ordering of states is preserved

Implementation in the CPU

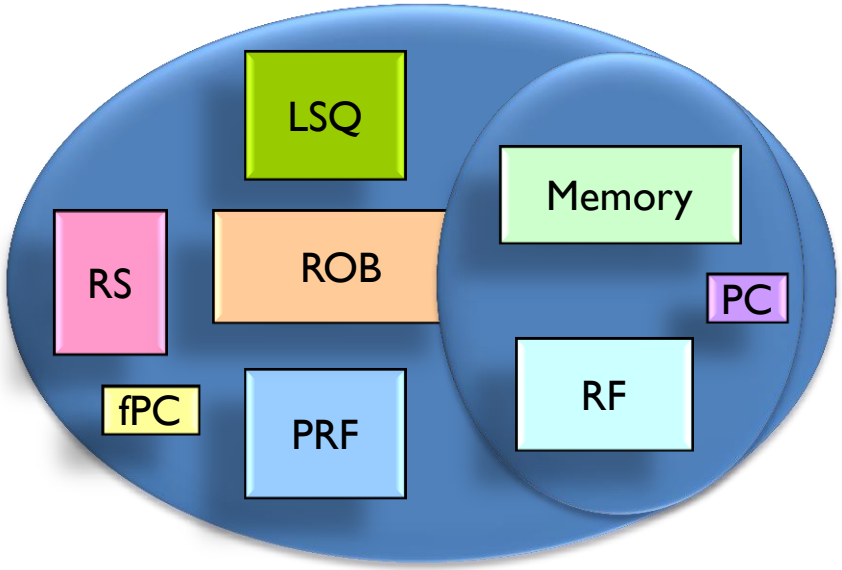
- ARF keeps state corresponding to committed insns.
  - Commit from ROB happens in order
  - ARF always contains some RF state of sequential execution

- Whoever wants to "look" should look in ARF
  - What about insns. that executed out of order?

Only the Sequential Part Matterns



Sequential View of the Processor

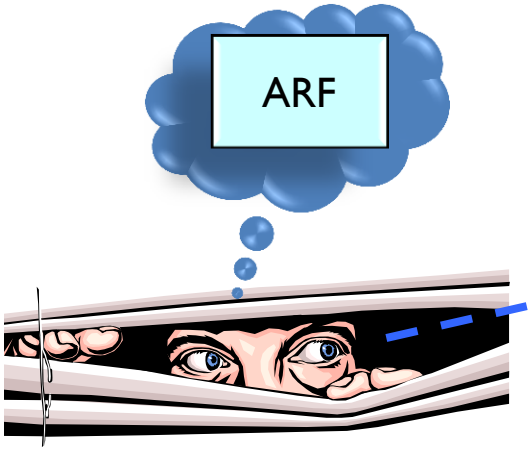


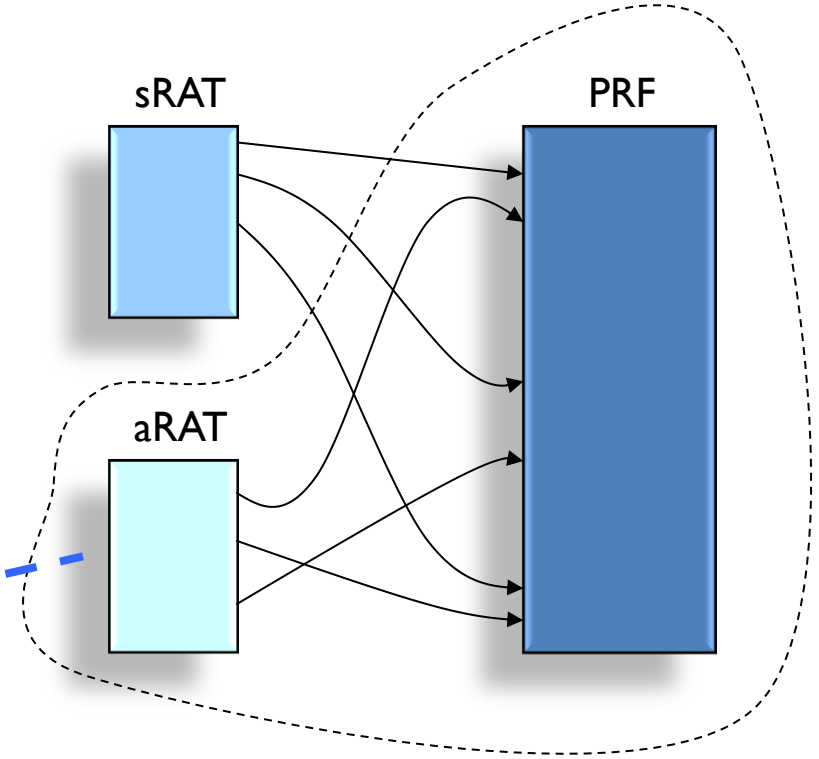
State of the Superscalar Out-of-Order Processor

What if there's no ARF?

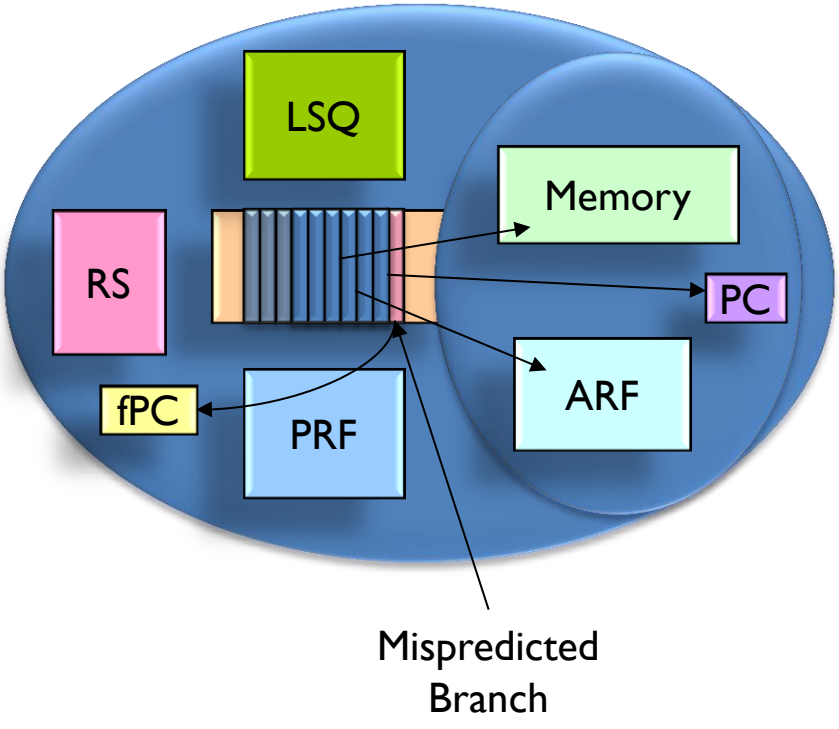
View of the Unified Register File

If you need to "see" a register, you go through the aRAT first.





View of Branch Mispredictions



Wrong-path instructions are flushed…

architected state has never been touched

Fetch correct path instructions

Which can update the architected state when they commit

Committing Instructions (1/2)

- "Retire" vs. "Commit"
  - Sometimes people use this to mean the same thing
  - Sometimes they mean different things
    - Check the context!

- Insn. commits by making "effects" visible
  - Architected state: (A)RF, Memory/$, PC
  - Speculative state: everything else (ROB, RS, LSQ, etc…)

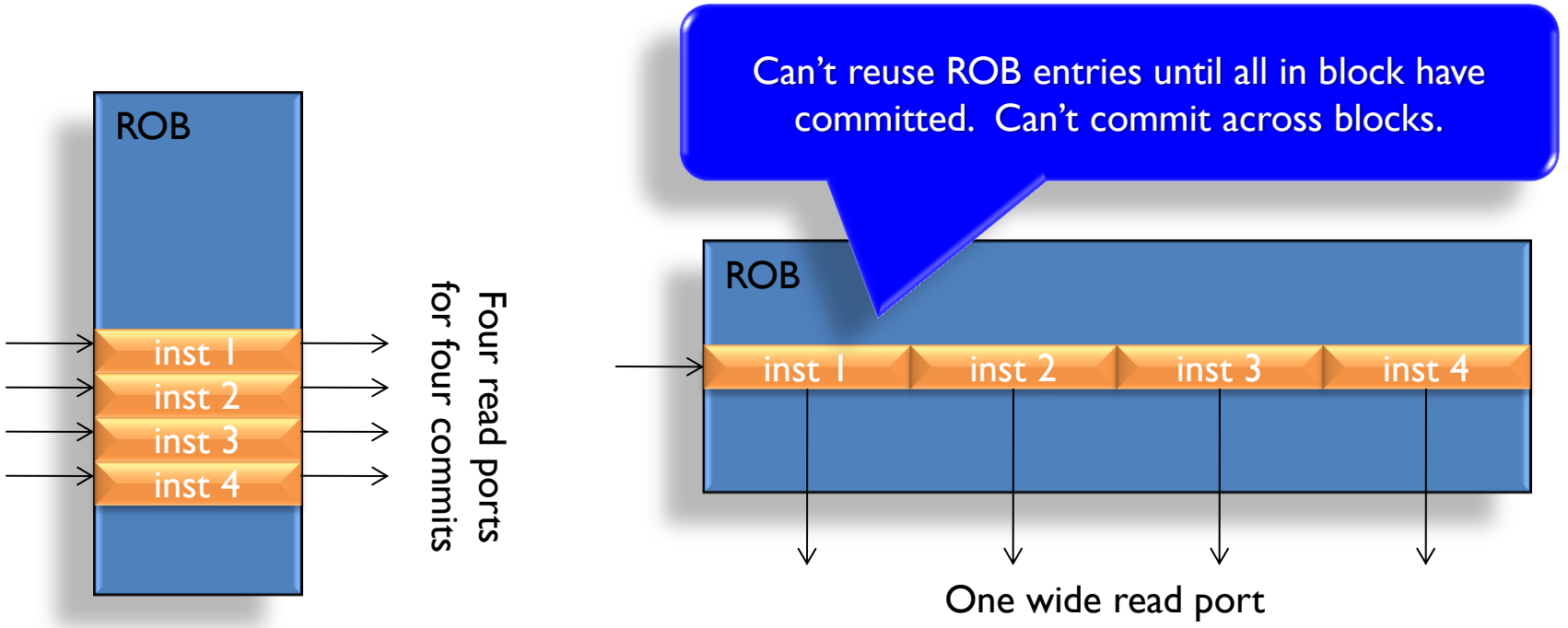
Committing Instructions (2/2)

- When an insn. executes, it modifies processor state
  - Update a register
  - Update memory
  - Update the PC (almost all instructions do this)

- To make "effects" visible, core copies values
  - Value from Physical Reg to Architected Reg
  - Value from LSQ to memory/cache
  - Value from ROB to Architected PC

Blocked Commit

- To commit N insns. per cycle, ROB needs N ports
  - (in ***addition to*** ports for dispatch, issue, exec, and WB)



Reduces cost, lowers IPC due to constraints.

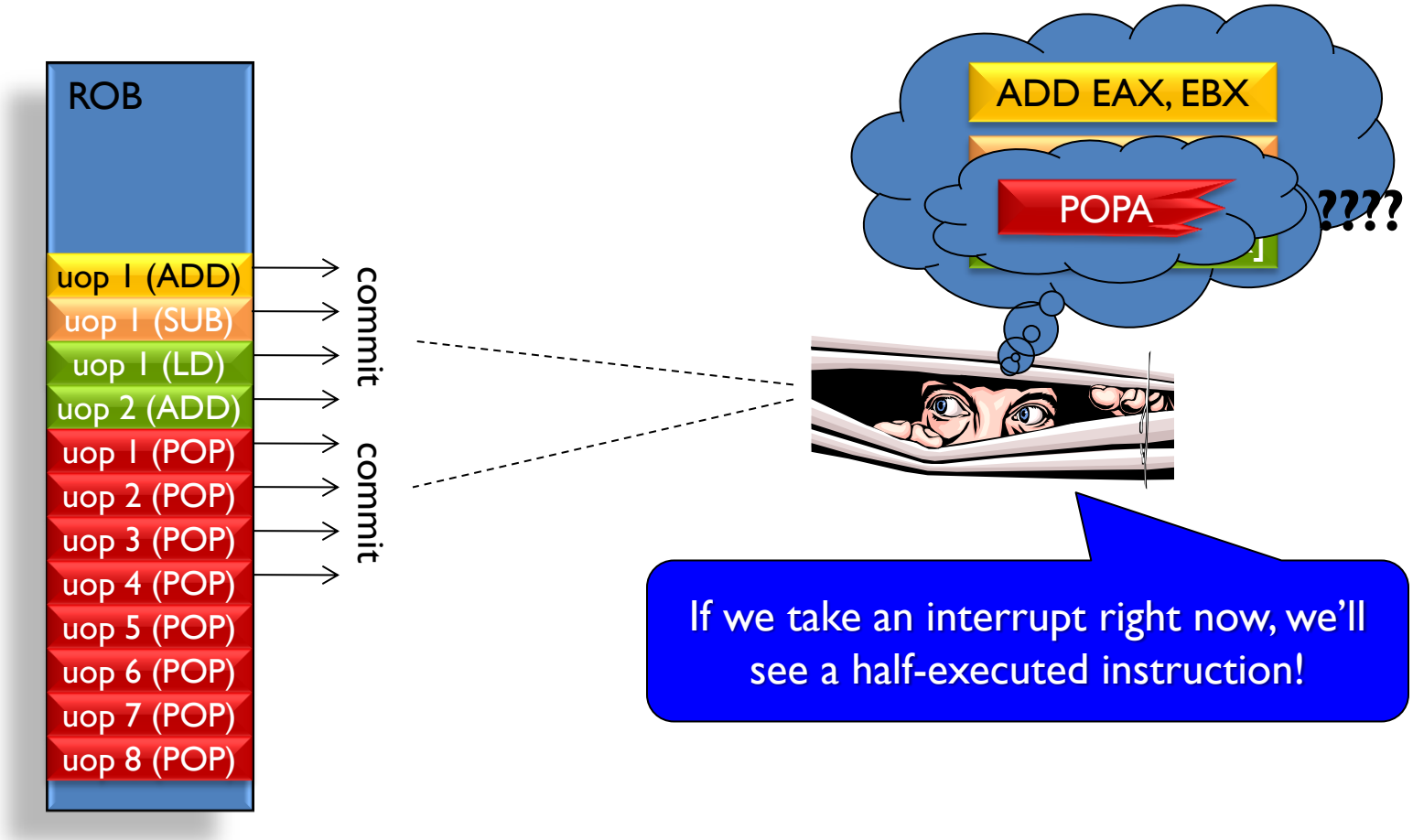
Commit Restrictions

- If any N insns. can commit per cycle
  - May require heavy multi-porting of other structures
    - Stores
      - N extra DL1 write ports
      - N extra DTLB read ports
    - Branches
      - N branch predictor update ports
      - Deallocate N RAT checkpoints
- Solution: Limit max commits per cycle of each type
  - Example: Max one branch per cycle

Don't we check DTLB during store-address computation anyway? Do we need to do it again here?

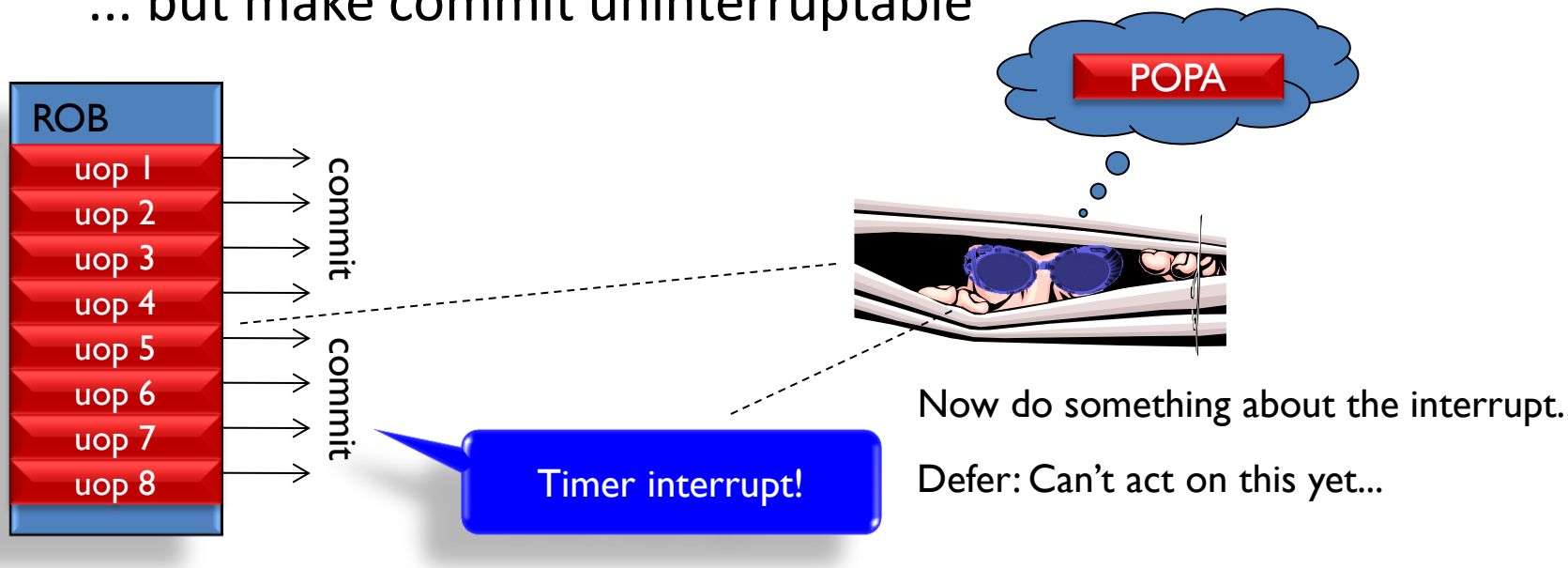
x86 Commit (1/2)

- ROB contains uops, outside world knows insns.



x86 Commit (2/2)

- Works when uop-flow length ≤ commit width
- What to do with long flows?
  - In all cases: can't commit until *all* uops in a flow completed
  - Just commit N uops per cycle
    ... but make commit uninterruptable

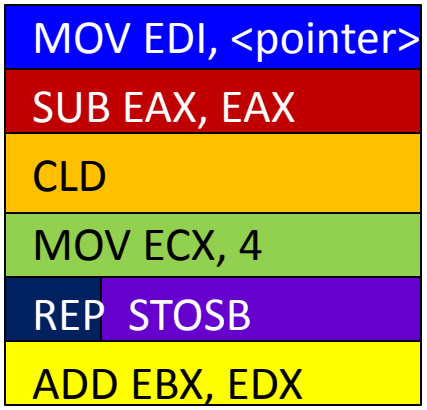


Handling REP-prefixed Instructions (1/2)

- Ex. REP STOSB (memset EAX value ECX times)
  - Entire sequence is one x86 instruction
  - What if REPs for 1,000,000,000 iterations?
    - Can't "lock up" for a billion cycles while uops commit
    - Can't wait to commit until all uops are done
      - Can't even fetch the entire instruction – not enough space in ROB

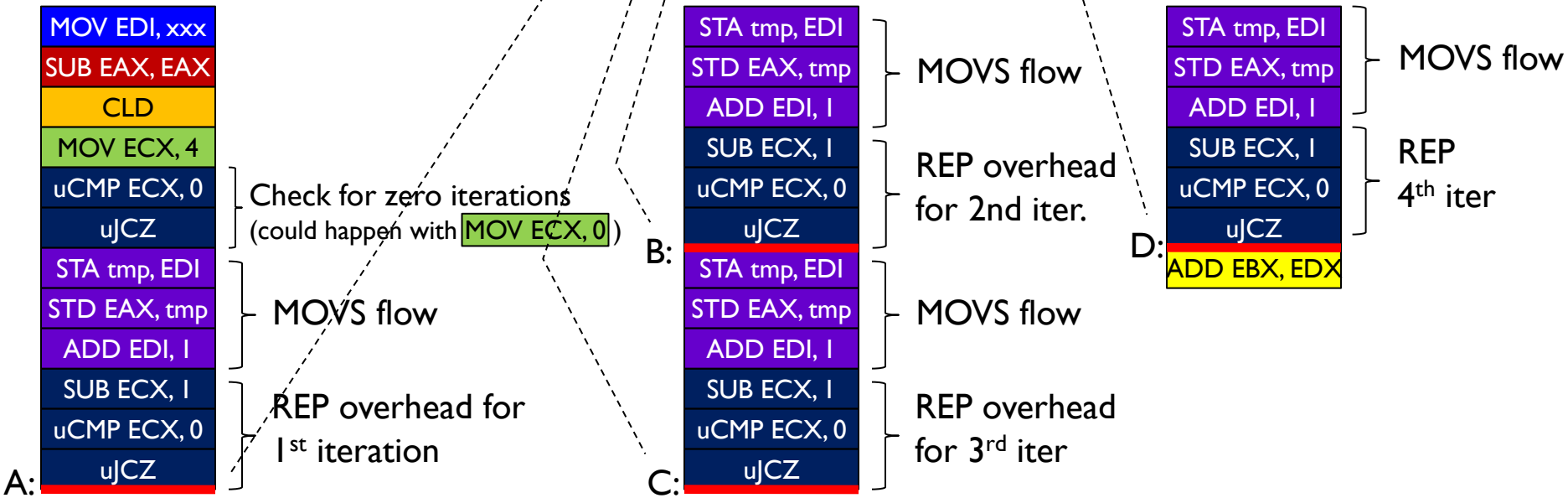
- At the ISA level, REP iterations are interruptible...
  - Treat each iteration as a separate "macro-op"

Handling REP-prefixed Instructions (2/2)



All of these are interruptible points (commit can stop and effects be seen by outside world), since they all have well-defined ISA-level states:

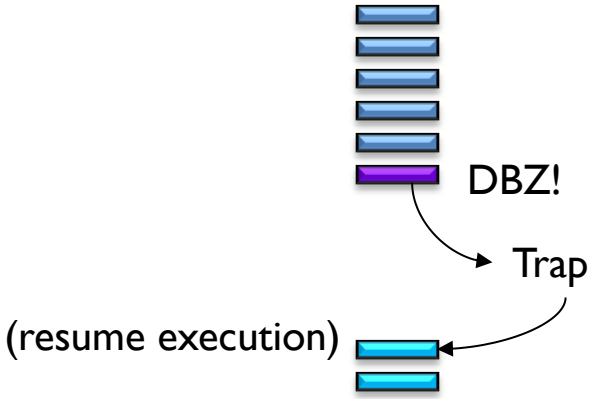
A: ECX=3, EDI = ptr+1
B: ECX=2, EDI = ptr+2
C: ECX=1, EDI = ptr+3
D: ECX=0, EDI = ptr+4

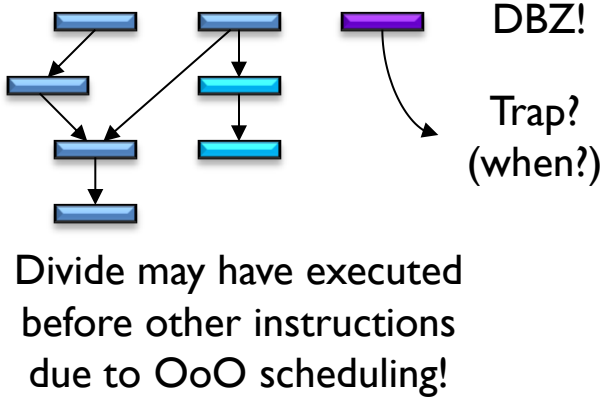


Faults

- Divide-by-Zero, Overflow, Page-Fault

- All occur at a specific point in execution (precise)

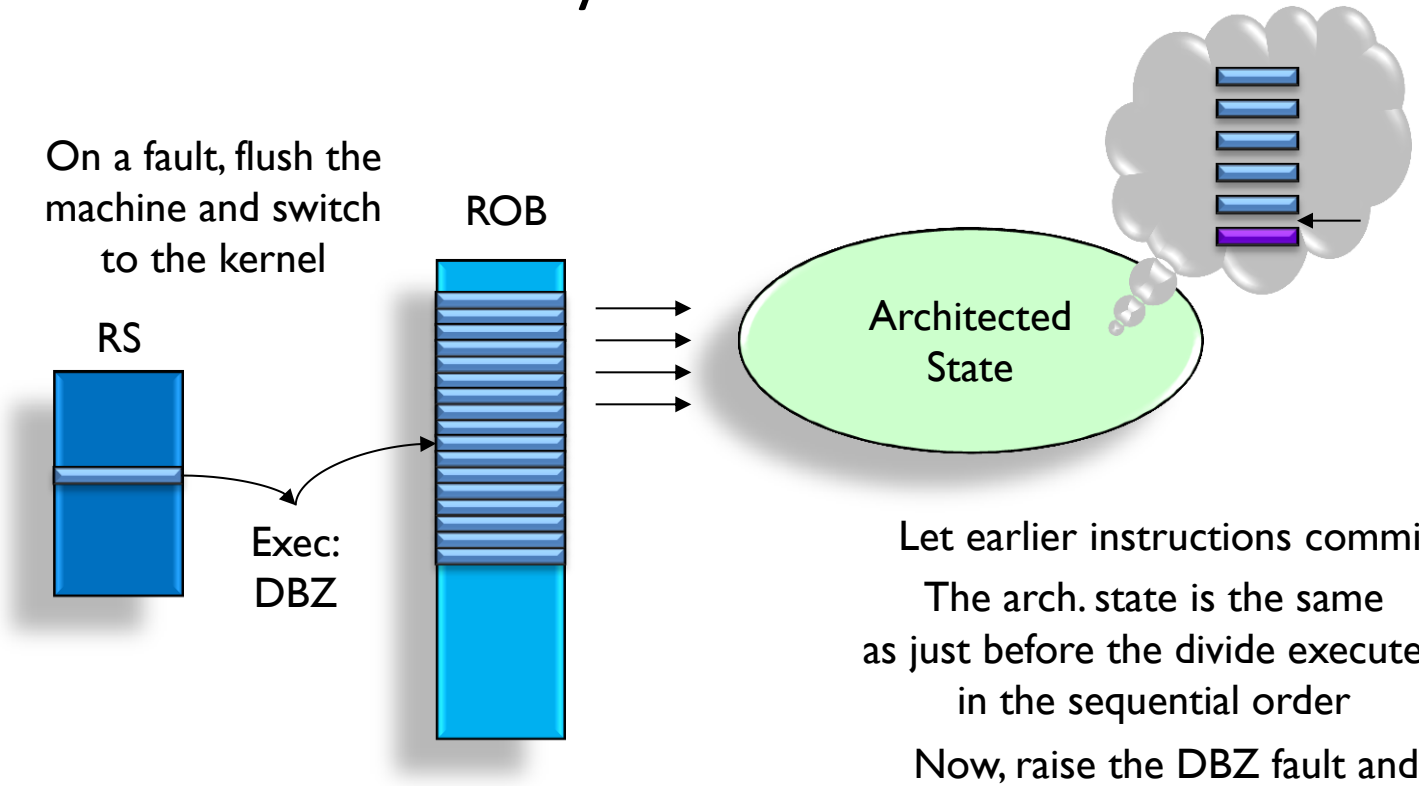




CPU maintains appearance of sequential execution

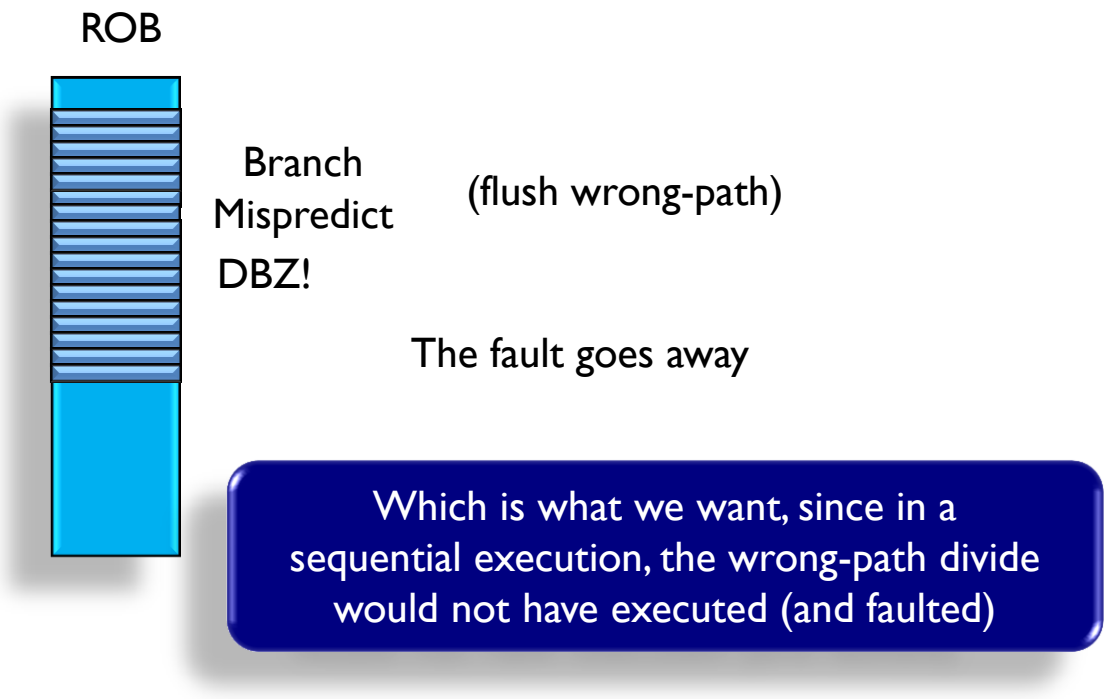
Timing of DBZ Fault

- Need to hold on to your faults



Speculative Faults

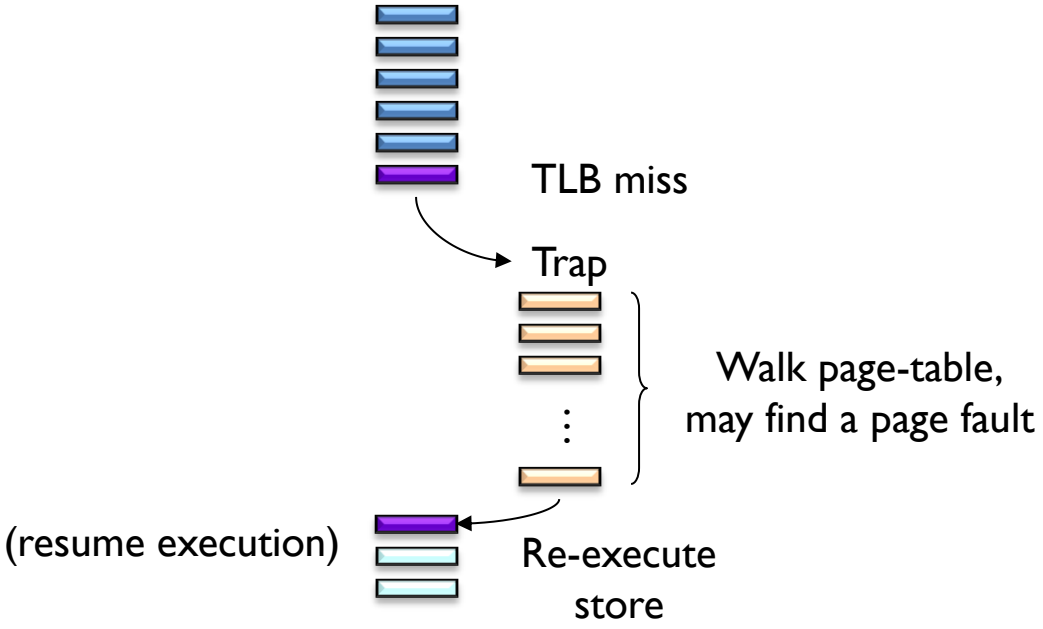
- Faults might not be faults…



Buffer faults until commit to avoid speculative faults

Timing of TLB Miss

- Store must re-execute (or re-commit)
  - Cannot leave the ROB



Store TLB miss can stall the core

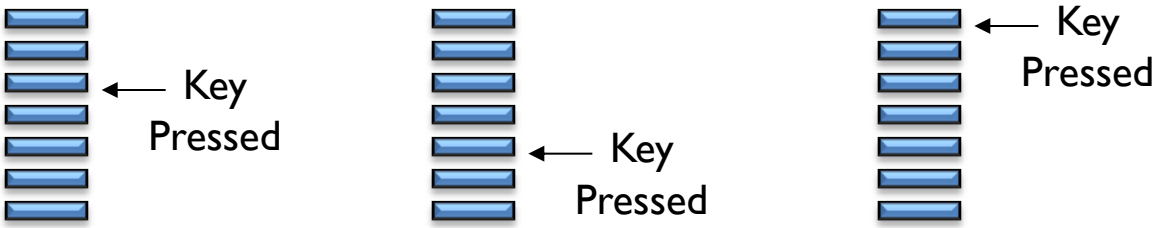
Load Faults are Similar

- Load issues, misses in TLB
  - When load is oldest, switch to kernel for page-table walk
- …could be painful; there are lots of loads

- Modern processors use hardware page-table walkers
  - OS loads a few registers with PT information (pointers)
  - Simple logic fetches mapping info from memory
  - Requires page-table format is specified by the ISA

Asynchronous Interrupts

- Some interrupts are not associated with insns.
  - Timer interrupt
  - I/O interrupt (disk, network, etc…)
  - Low battery, UPS shutdown
- When the CPU "notices" doesn't matter (too much)

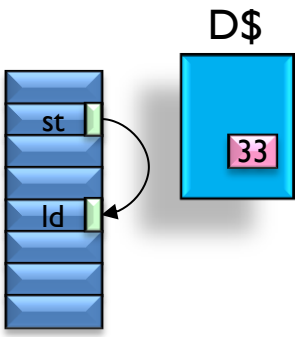


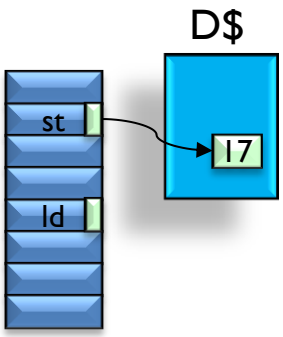
Two Options for Handling Async Interrupts

- Handle immediately
  - Use current architected state and flush the pipeline
- Deferred
  - Stop fetching, let processor drain, then switch to handler
    - What if CPU takes a fault in the mean time?
    - Which came “first”, the async. interrupt or the fault?

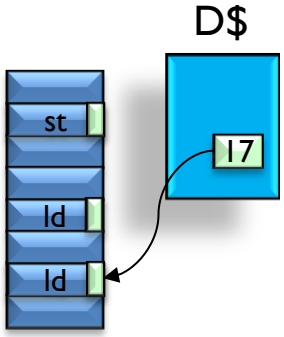
Store Retirement (1/2)

- Stores forward to later loads (for same address)
  - Normally, LSQ provides this facility





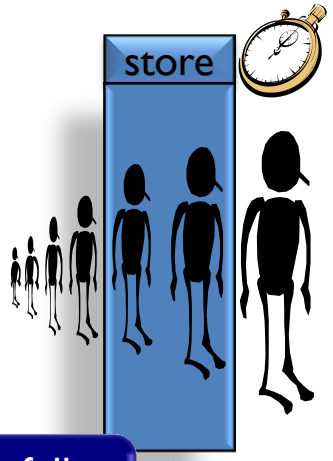
At commit, store Updates cache



After store has left the LSQ, the D$ can provide the correct value

Store Retirement (2/2)

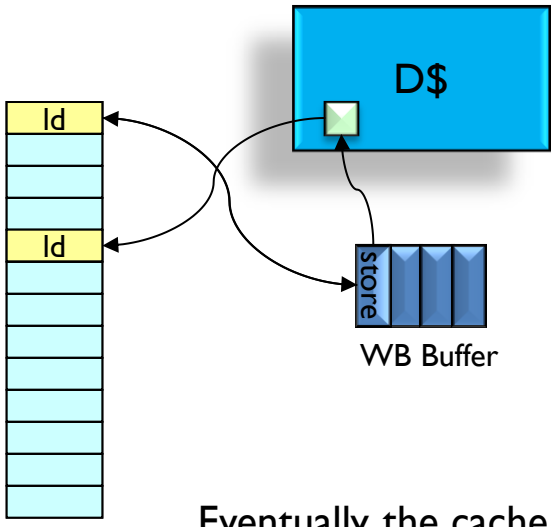
- Can't free LSQ Store entry until write is done
  - Enables forwarding until loads can get value from cache
- Have to re-check TLB when doing write
  - TLB contents at Execute were speculative
- Store may stall commit for a long time
  - If there's a cache miss
  - If there's a TLB miss (with HW TLB walk)



All instructions may have successfully executed, but none can commit!

Writeback Buffer (1/2)

- Want to get stores out of the way quickly



Even if store misses in cache, entering WB buffer counts as committing.

Allows other insns. to commit.

WB buffer is part of the cache hierarchy. May need to provide values to later loads.

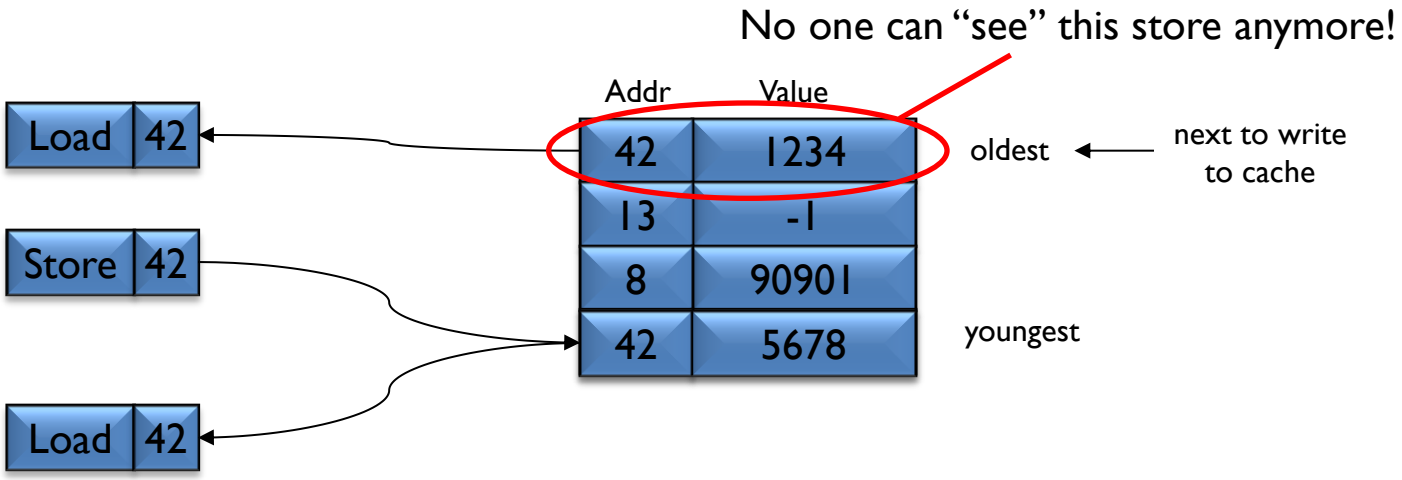
Eventually, the cache update occurs, the WB buffer entry is emptied.

Cache can now provide the correct value.

Usually fast, but potential structural hazard

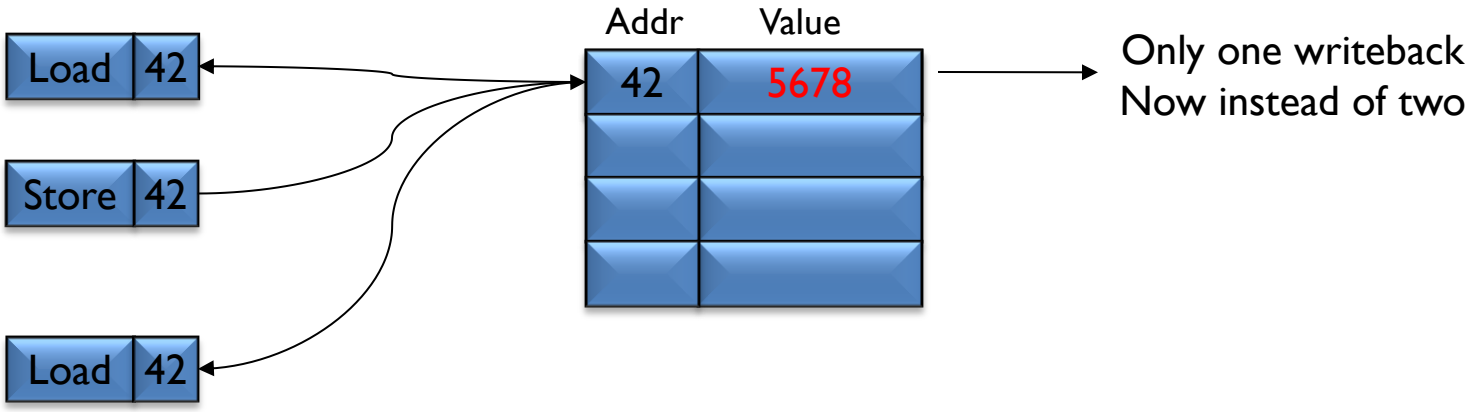
Writeback Buffer (2/2)

- Stores enter WB Buffer in program order
- Multiple stores can exist to same address
  - Only the last store is "visible"



Write Combining Buffer (1/2)

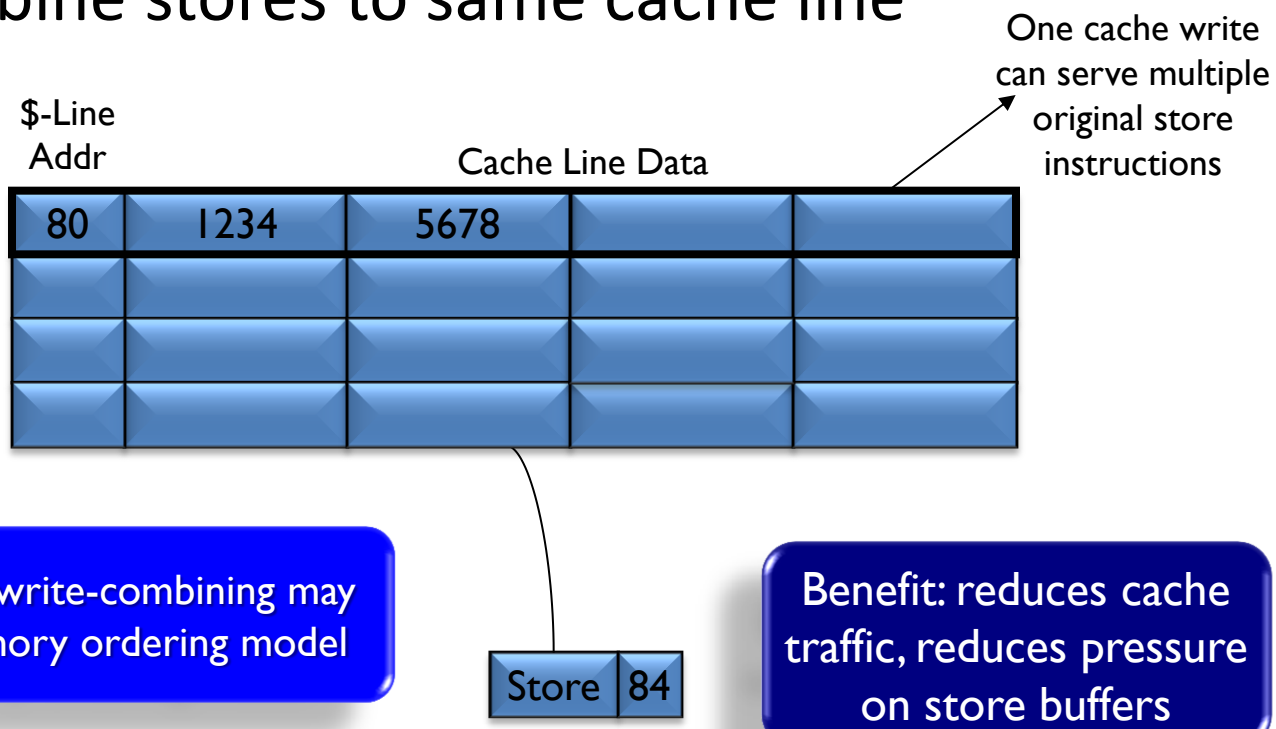
- Augment WBB to combine writes together



If Stores to same address, combine the writes

Write Combining Buffer (2/2)

- Can combine stores to same cache line



Aggressiveness of write-combining may be limited by memory ordering model

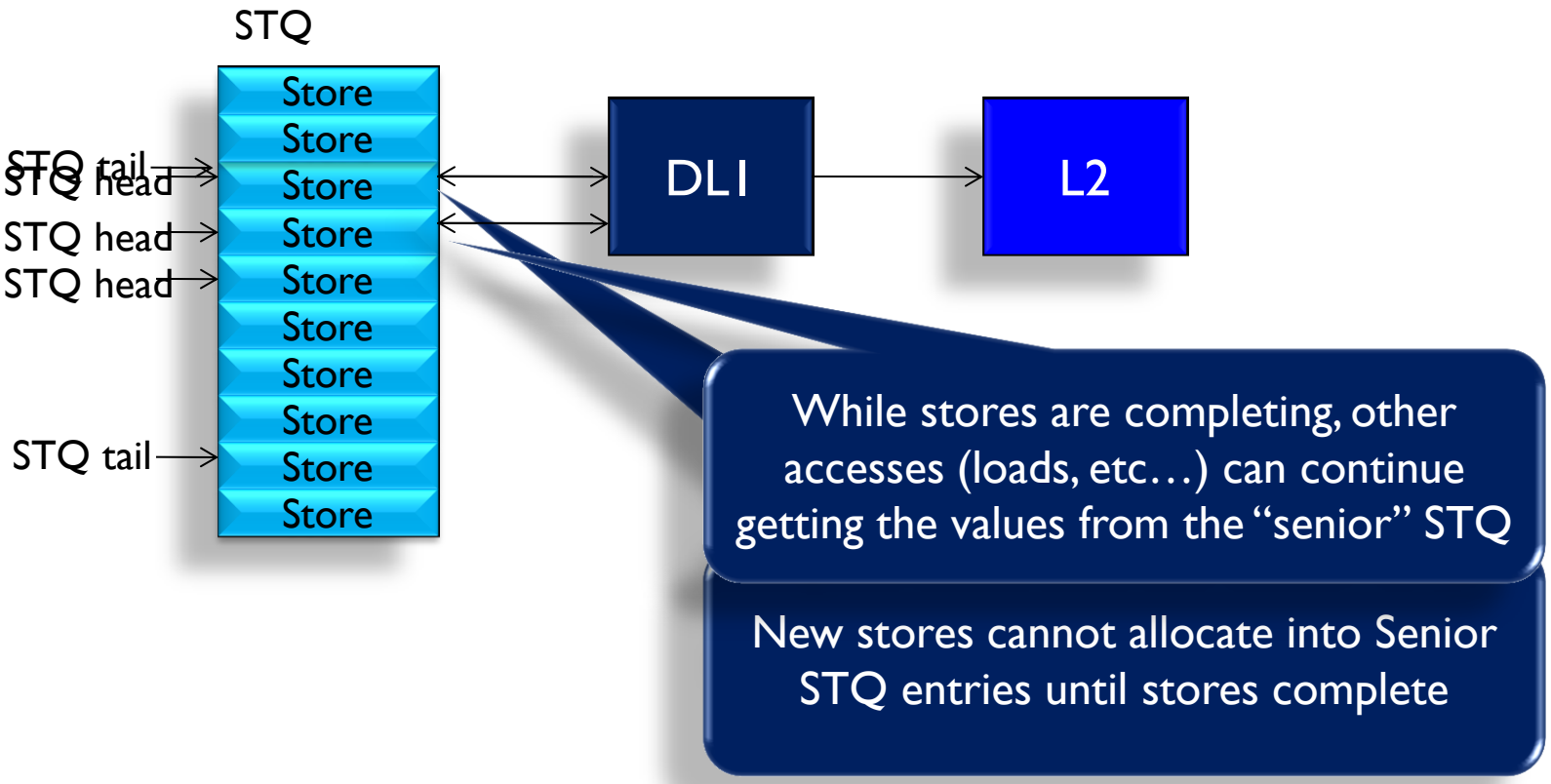
Benefit: reduces cache traffic, reduces pressure on store buffers

Writeback/combining buffer can be implemented in/integrated with the MSHRs

Only certain memory regions may be "write-combinable" (e.g., USWC in x86)

Senior Store Queue

- Use STQ as WBB (not necessarily write combining)



No WBB and no stall on Store commit

Cleanup on Commit (Retire)

- Besides updating architected state
  … commit needs to deallocate resources
  - ROB/LSQ entries
  - Physical register
  - "colors" of various sorts
  - RAT checkpoints
- Most are FIFO's or Queues
  - Alloc/dealloc is usually just inc/dec head/tail pointers
- Unified PRF requires a little more work
  - Have to return "old" mapping to free list