

# CSE 502:

# Computer Architecture

Out-of-Order Memory Access

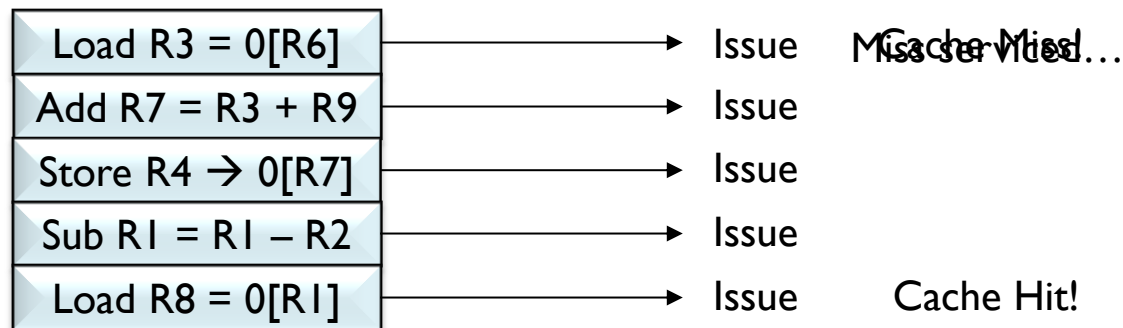
# Dynamic Scheduling Summary

- Out-of-order execution: a performance technique
- Feature I: Dynamic scheduling (iO  $\rightarrow$  OoO)
  - “Performance” piece: re-arrange insns. for high perf.
  - Decode (iO)  $\rightarrow$  dispatch (iO) + issue (OoO)
  - Two algorithms: Scoreboard, Tomasulo
- Feature II: Precise state (OoO  $\rightarrow$  iO)
  - “Correctness” piece: put insns. back into program order
  - Writeback (OoO)  $\rightarrow$  complete (OoO) + retire (iO)
  - Two designs: P6, R10K

One remaining piece: OoO memory accesses

# Executing Memory Instructions

- If  $R1 \neq R7$ 
  - Then Load R8 gets correct value from cache
- If  $R1 == R7$ 
  - Then Load R8 should get value from the Store
  - *But it didn't!*



But there was a later load...

# Memory Disambiguation Problem

- Ordering problem is a data-dependence violation
- Imprecise memory worse than imprecise registers
- Why can't this happen with non-memory insts?
  - Operand specifiers in non-memory insns. are absolute
    - “R1” refers to one specific location
  - Operand specifiers in memory insns. are ambiguous
    - “R1” refers to a memory location specified by the value of R1.
    - When pointers (e.g., R1) change, so does this location

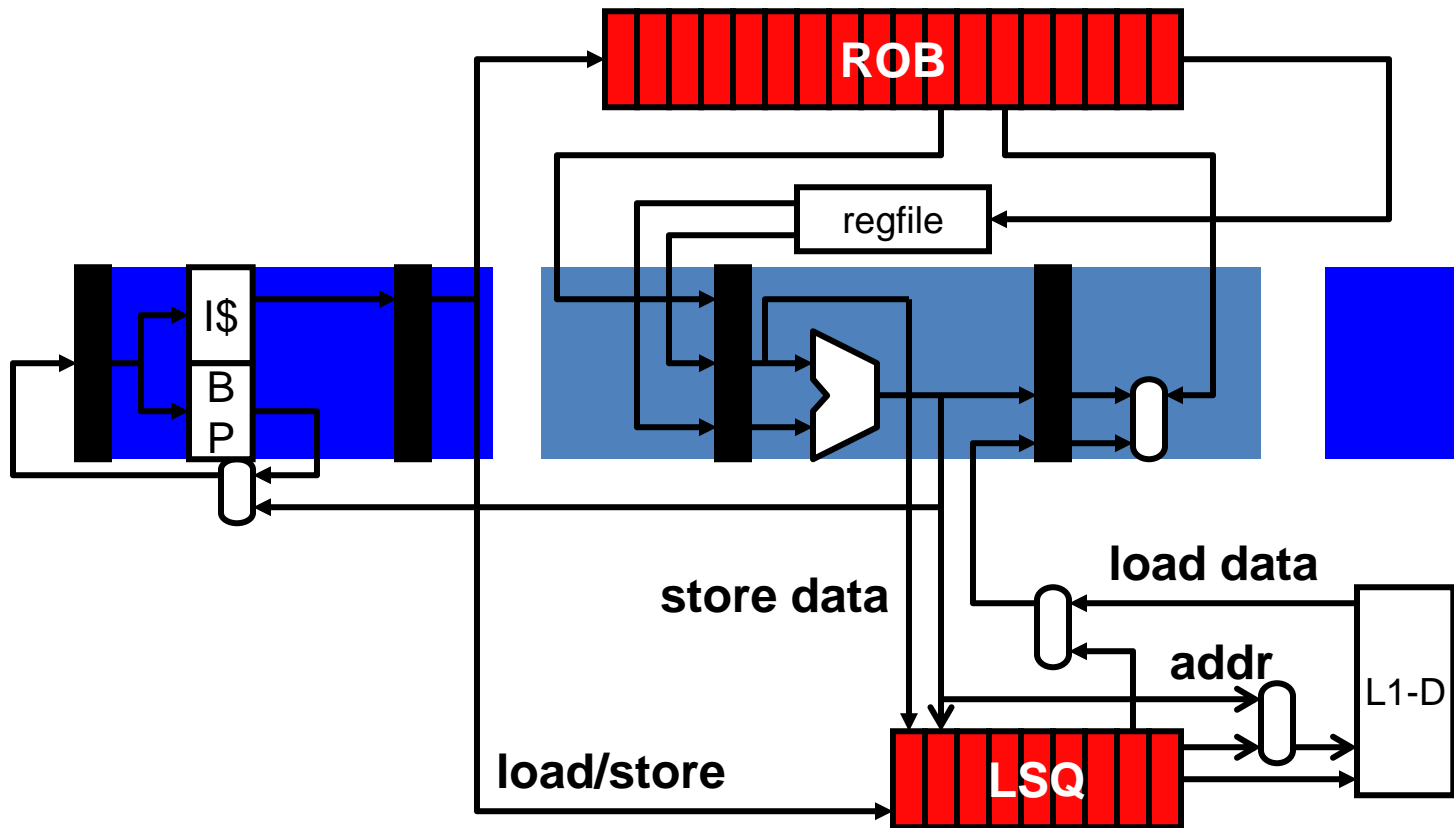
# Two Problems

- Memory disambiguation on loads
  - Do earlier unexecuted stores to the same address exist?
    - Binary question: answer is yes or no
- Store-to-load forwarding problem
  - I'm a load: Which earlier store do I get my value from?
  - I'm a store: Which later load(s) do I forward my value to?
    - Non-binary question: answer is one or more insn. identifiers

# Load/Store Queue (1/3)

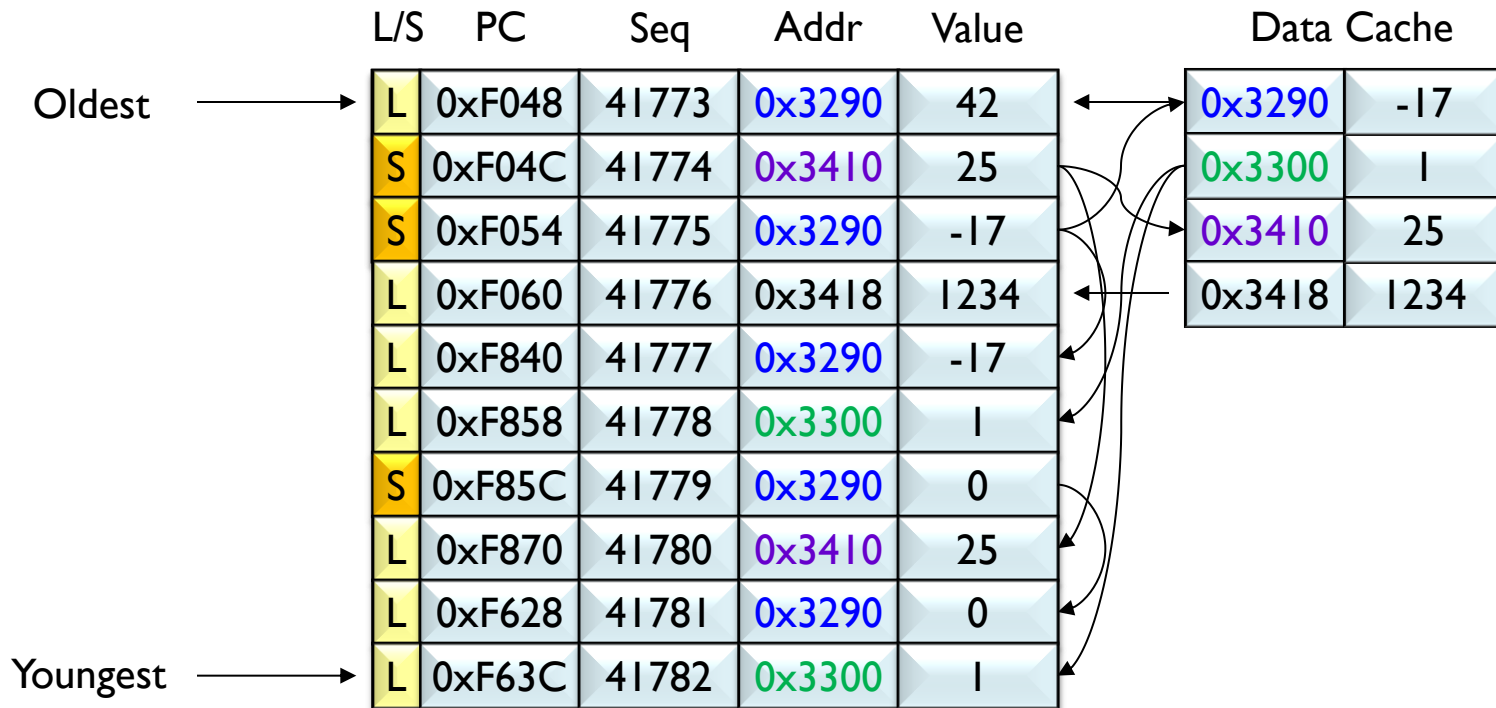
- Load/store queue (LSQ)
  - Completed stores write to LSQ
  - When store retires, head of LSQ written to L1-D
  - When loads execute, access LSQ and L1-D in parallel
    - Forward from LSQ if older store with matching address

# Load/Store Queue (2/3)



Almost a “real” processor diagram

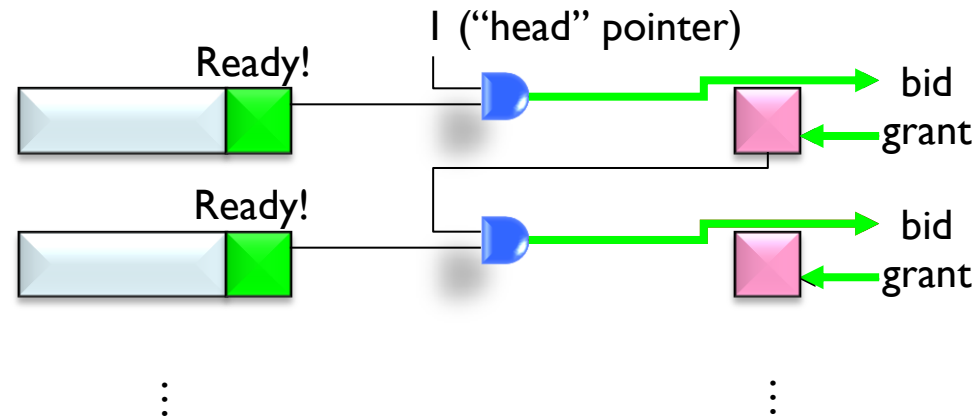
# Load/Store Queue (3/3)





# In-order Memory (Policy 1/4)

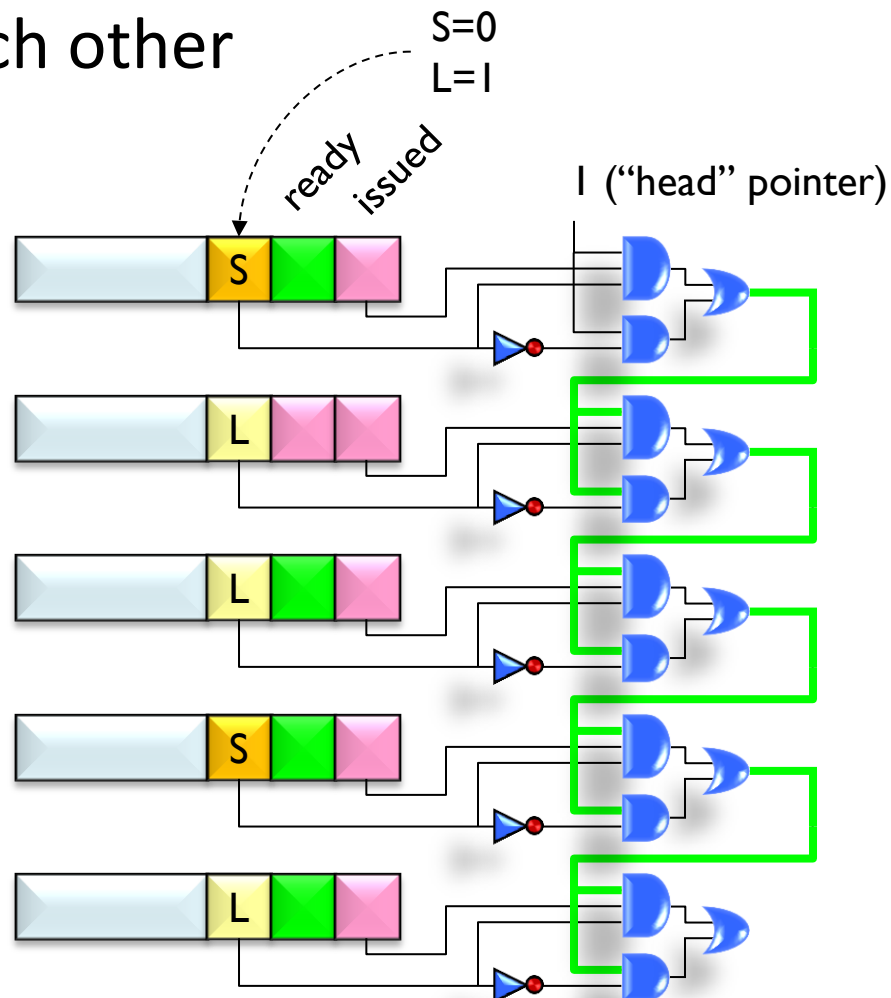
- No memory reordering
- LSQ still needed for forwarded data (last slide)
- Easy to schedule



Fairly simple, but low performance

# Loads OoO between Stores (Policy 2/4)

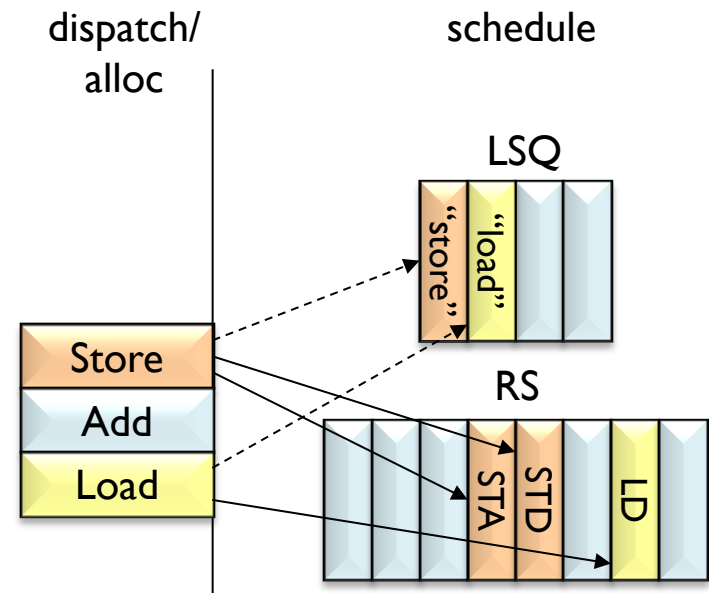
- Loads exec OoO w.r.t. each other
  - Stores block everything



Still simple, but better performance

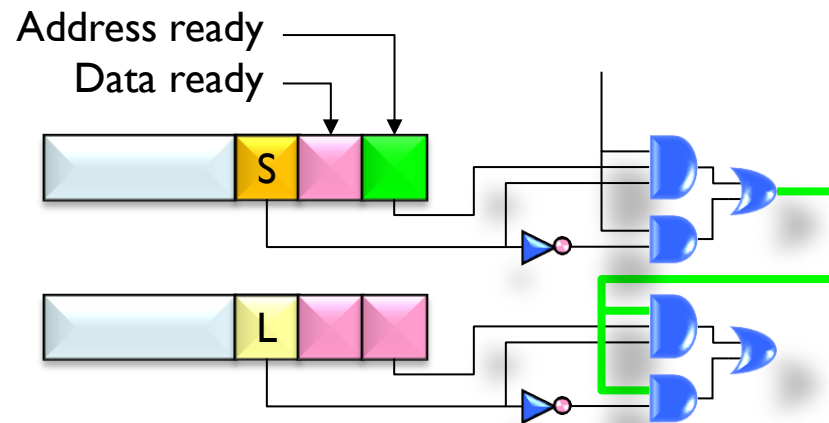
# Stores Can be Split into STA/STD

- STA: SToRe Address
- STD: SToRe Data
- Makes some designs easier
  - RS/ROB store one value
  - Stores need two (A & D)



# Loads Wait for STAs Only (Policy 3/4)

- Only address is needed to disambiguate
- May be ready earlier to allow checking for violations
  - No need to wait for data



Still simple, even better performance

# Loads Execute When Ready (Policy 4/4)

- Most aggressive approach
- Relies on fact that store → load forwarding is *rare*
- Greatest potential IPC – loads never stall
- Potential for incorrect execution
  - Need to be able to “undo” bad loads

Very complex, but high performance

# Detecting Ordering Violations (1/2)

- Case 1: Older store execs before younger load
  - No problem; if same address st→ld forwarding happens
- Case 2: Older store execs after younger load
  - Store scans all younger loads
  - Address match → ordering violation

# Detecting Ordering Violations (2/2)

(Load 41773 ignores broadcast because it has a lower seq #)

L/S	PC	Seq	Addr	Value
L	0xF048	41773	0x3290	42
S	0xF04C	41774	0x3410	25
S	0xF054	41775	0x3290	-17
L	0xF060	41776	0x3418	1234
L	0xF840	41777	0x3290	-17
L	0xF858	41778	0x3300	1
S	0xF85C	41779	0x3290	0
L	0xF870	41780	0x3410	25
L	0xF628	41781	0x3290	-17
L	0xF63C	41782	0x3300	1

Store broadcasts value,  
address and sequence #  
(-17,0x3290,41775)

IF younger load hadn't executed, and  
address matches, grab broadcasted value

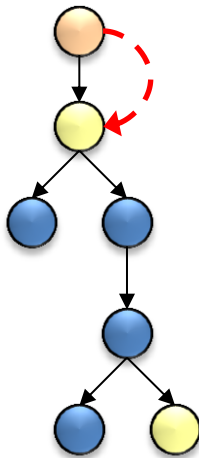
Loads CAM-match on  
address, only care if  
store seq-# is lower than  
own seq  
(0,0x3290,41779)

An instruction may be involved in  
more than one ordering violation  
IF younger load has executed, and  
address matches, then **ordering violation!**

Must flush *all* later accesses after violation

# Dealing with Misspeculations

- Loads are not the only thing which are wrong
  - Loads propagate wrong values to all dependents
- These must somehow be re-executed



- Easiest: flush all instructions after (and including?) the misspeculated load, and just refetch
- Load uses forwarded value
- Correct value propagated when instructions re-execute



# Flushing Complications

- Exactly same mispredicted branches
  - Checkpoint at every load in addition to branches
    - Very large number of checkpoints needed
  - Rollback to previous branch (which has its own checkpoint)
    - Make sure load doesn't misspeculate on 2<sup>nd</sup> try
    - Must redo work between the branch and the load
  - Can work with undo-list style of recovery
- Not all younger insns. are dependent on bad load
- Pipeline latency due to *refetch* is exposed

# Selective Re-Execution

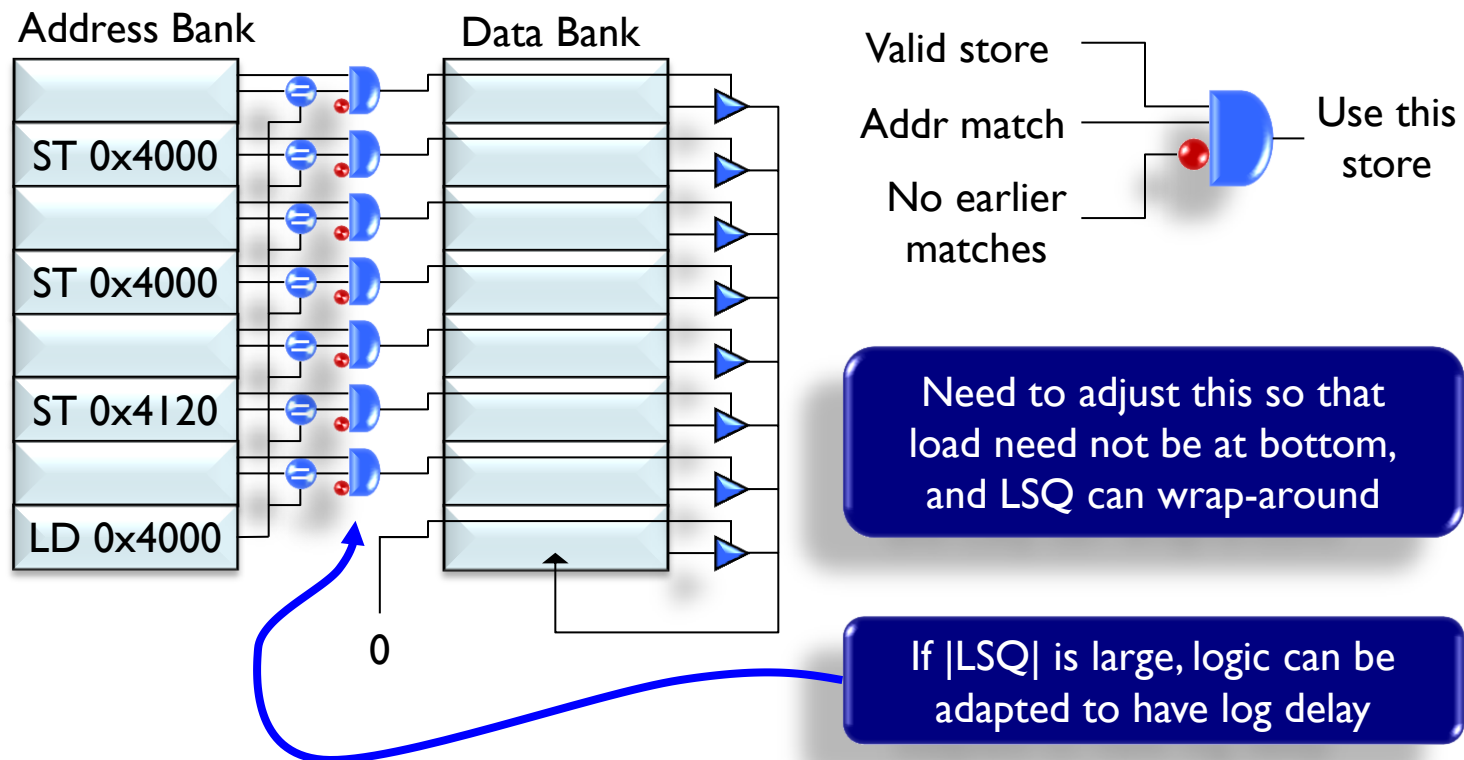
- Re-execute only the dependent insns.
- Ideal case w.r.t. maintaining high IPC
  - No need to re-fetch/re-dispatch/re-rename/re-execute
- Very complicated
  - Need to hunt down only data-dependent insns.
  - Some bad insns. already executed (now in ROB)
  - Some bad insns. didn't execute yet (still in RS)
- P4 does something like this (called “replay”)

# LSQ Hardware in More Detail

- Very complicated CAM logic
  - Need to quickly look up based on value
  - May find multiple values / need age based search
- No need for age-based search in ROB
  - Physical regs. are renamed, guarantees one writer
  - No easy way to prevent multiple stores to same address

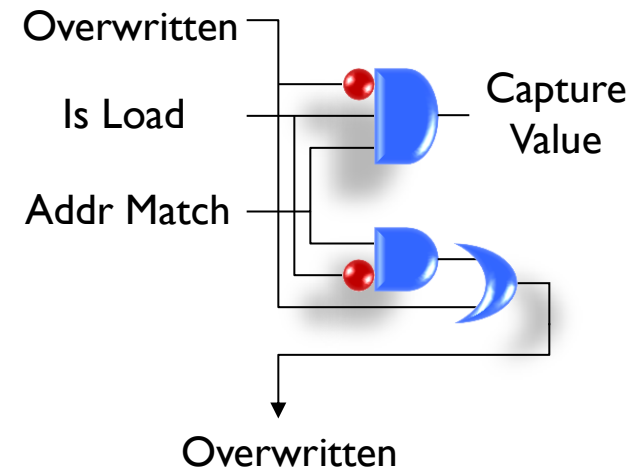
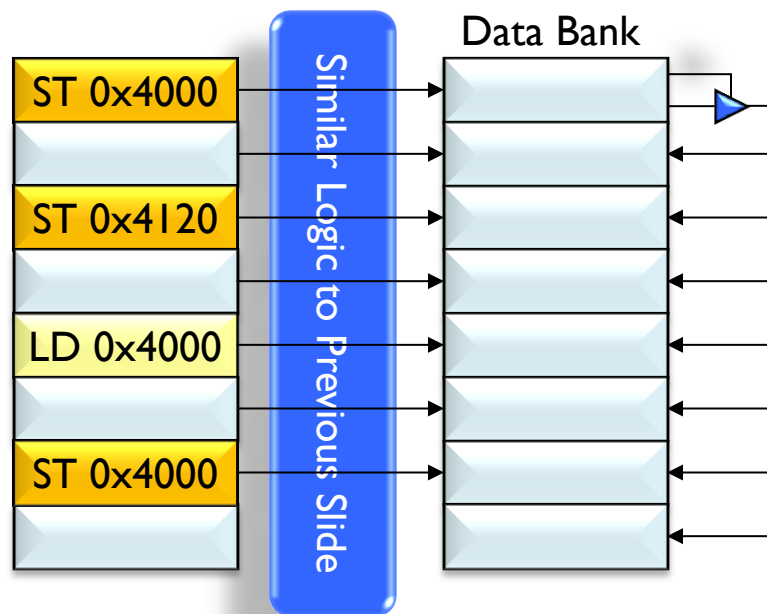
# Loads Checking for Earlier Stores

- On Load dispatch, find data from earlier Store



# Data Forwarding

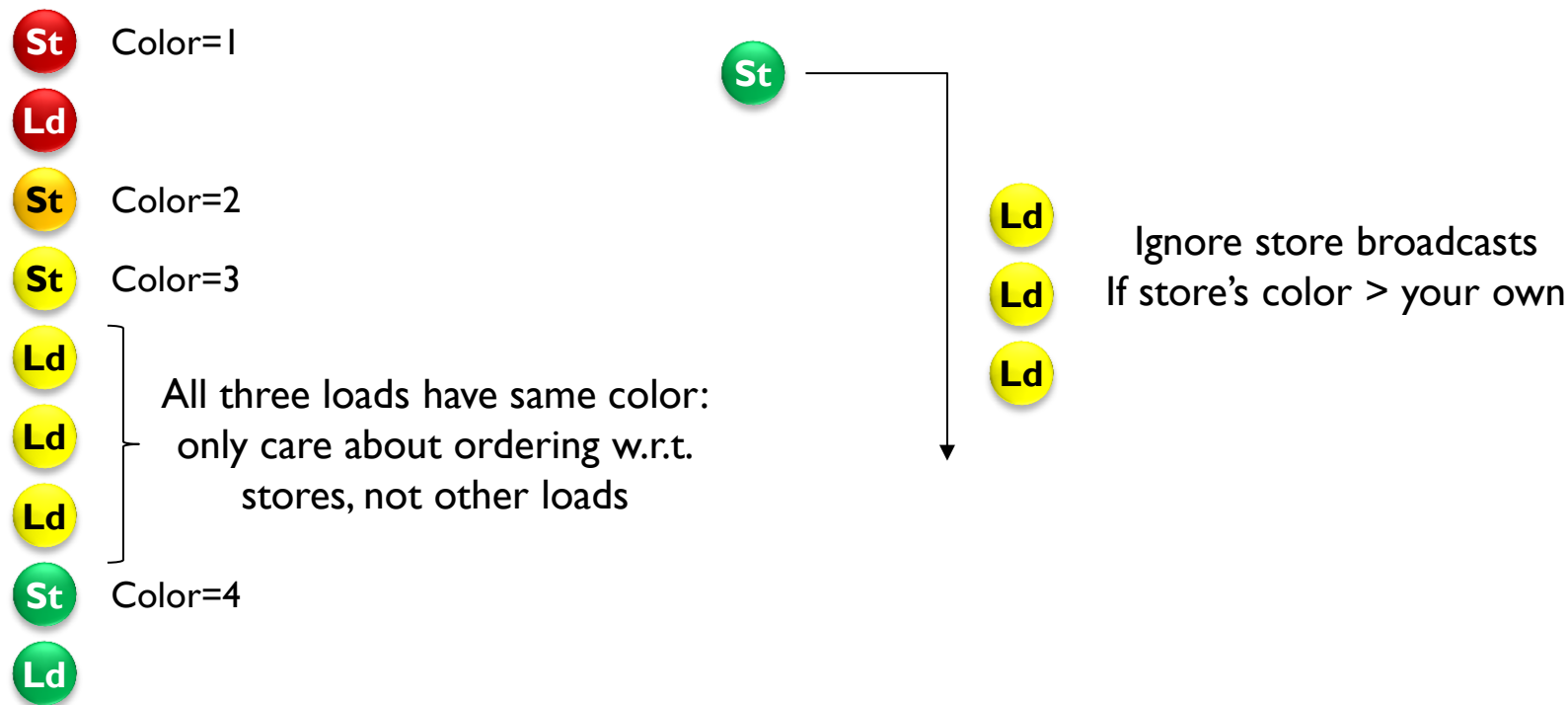
- On execute Store (STA+STD), check for later Loads



This is ugly, complicated, slow, and power hungry

# Alternative Data Forwarding: Store Colors

- Each store assigned unique number (its color)
- Loads inherit the color of the most recent store



# Split Load Queue/Store Queue

- Stores don't need to broadcast address to stores
- Loads don't need to check against earlier loads

