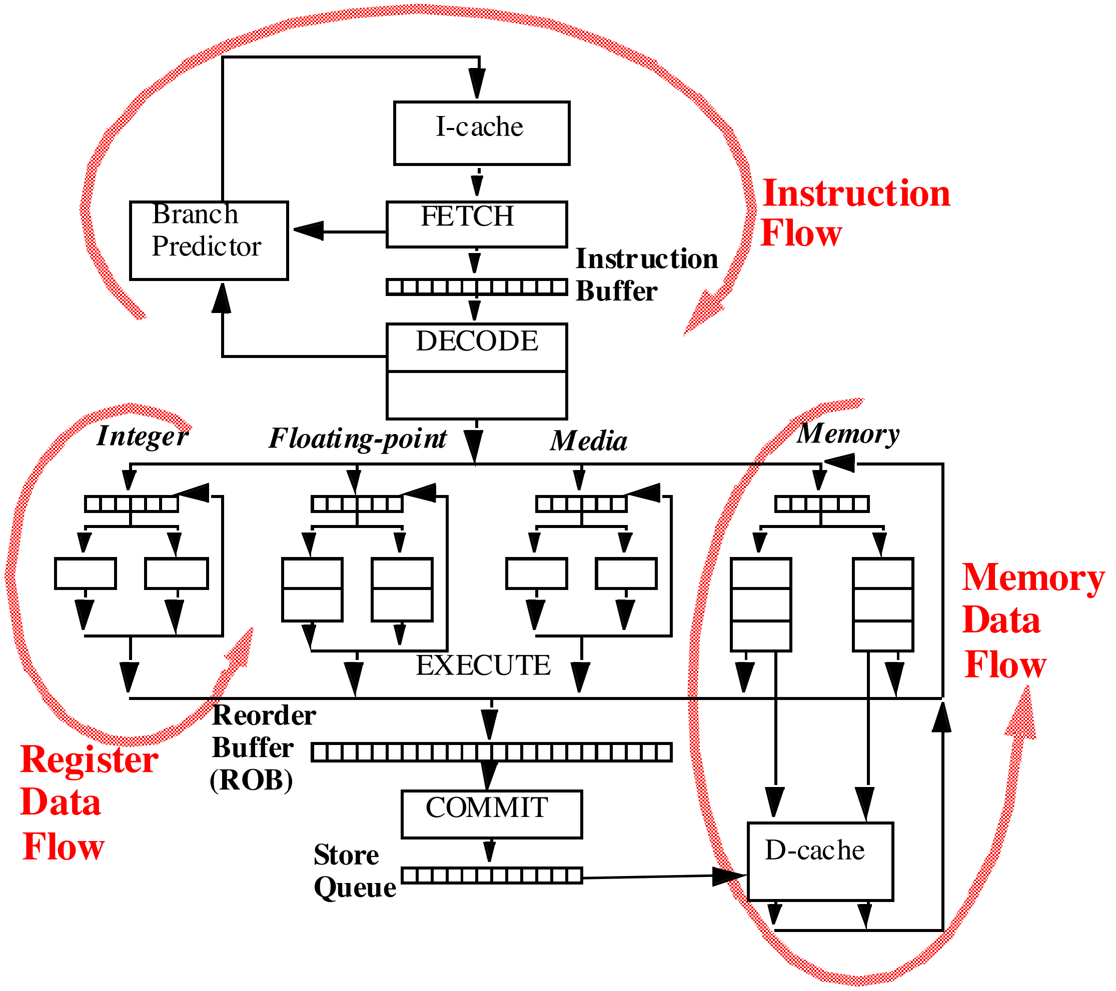
Superscalar Design

Memory Data Flow

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-739: Processor Design

Impediments to High IPC



Memory Data Flow Techniques

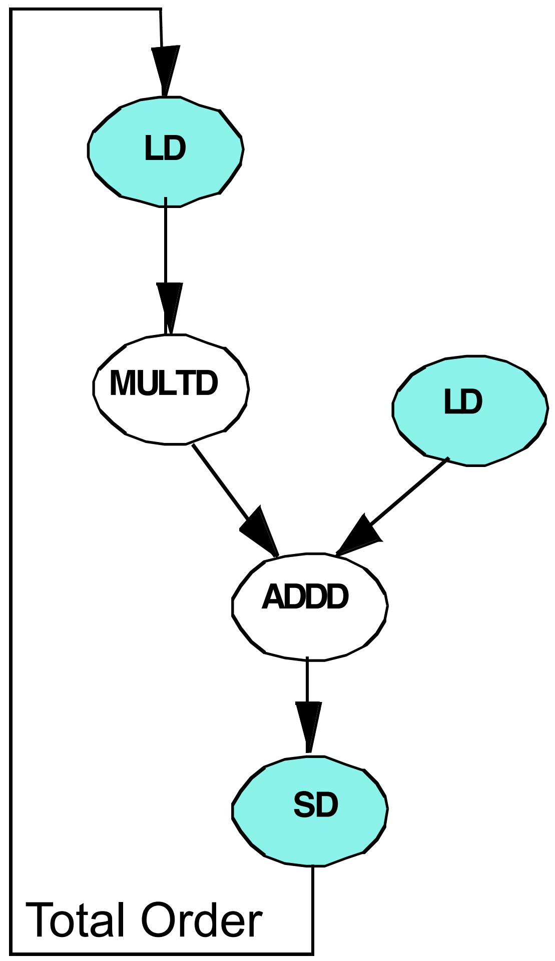
- Move data between memory and RF
- Long Latency
- Bottleneck
- Operations
  - Address Generation
  - Address Translation
  - Read/write data

The DAXPY Example

Y(i) = A * X(i) + Y(i)

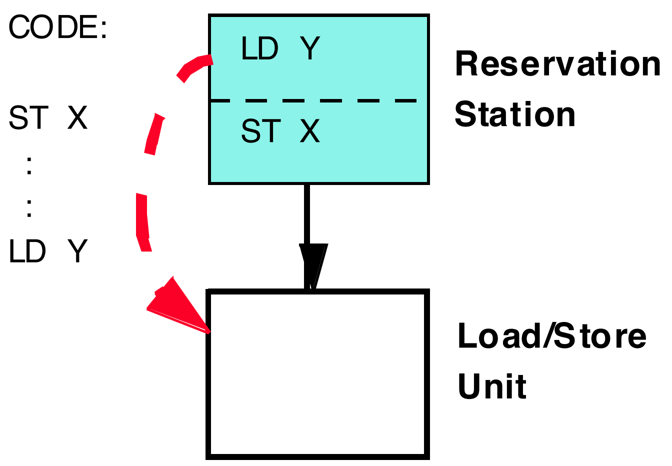
```
        LD      F0, a
        ADDI    R4, Rx, #512        ; last address

Loop:
        LD      F2, 0(Rx)           ; load X(i)
        MULTD   F2, F0, F2          ; A*X(i)
        LD      F4, 0(Ry)           ; load Y(i)
        ADDD    F4, F2, F4          ; A*X(i) + Y(i)
        SD      F4, 0(Ry)           ; store into Y(i)
        ADDI    Rx, Rx, #8          ; inc. index to X
        ADDI    Ry, Ry, #8          ; inc. index to Y
        SUB     R20, R4, Rx         ; compute bound
        BNZ     R20, loop           ; check if done
```

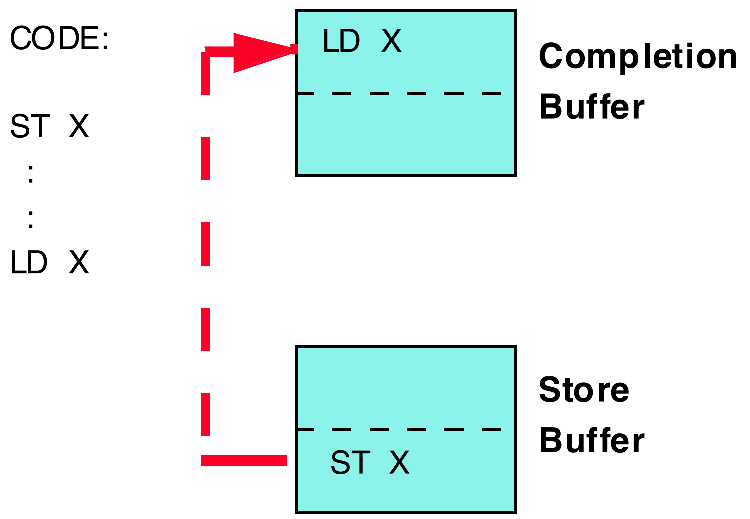


Performance Gains From Weak Ordering

Load Bypassing:



Load Forwarding:



Performance gain:

Load bypassing: 11%-19% increase over total ordering

Load forwarding: 1%-4% increase over load bypassing

Optimizing Load/Store Disambiguation

- Non-speculative load/store disambiguation
  1. Loads wait for addresses of all prior stores
  2. Full address comparison
  3. Bypass if no match, forward if match

✧ (1) can limit performance:

load r5, MEM[r3] → cache miss

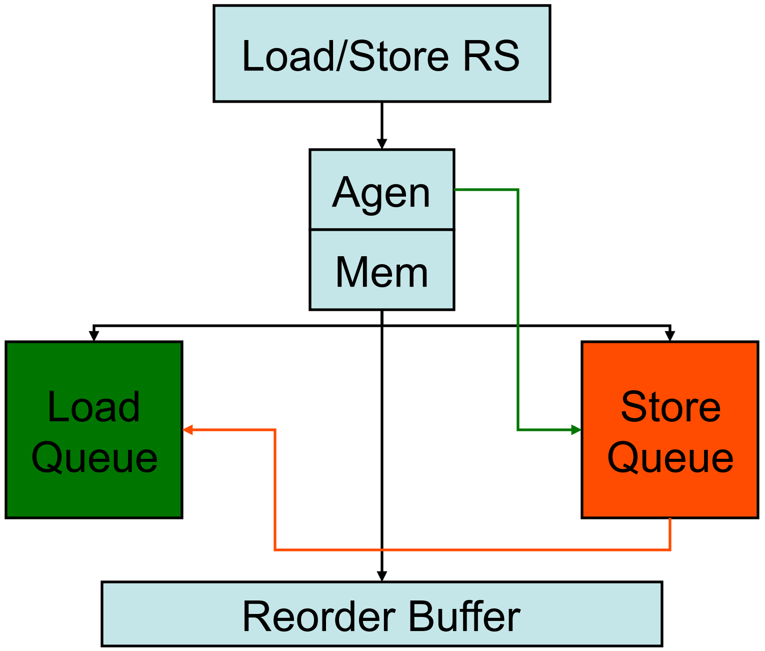
store r7, MEM[r5] → RAW for agen, stalled

…

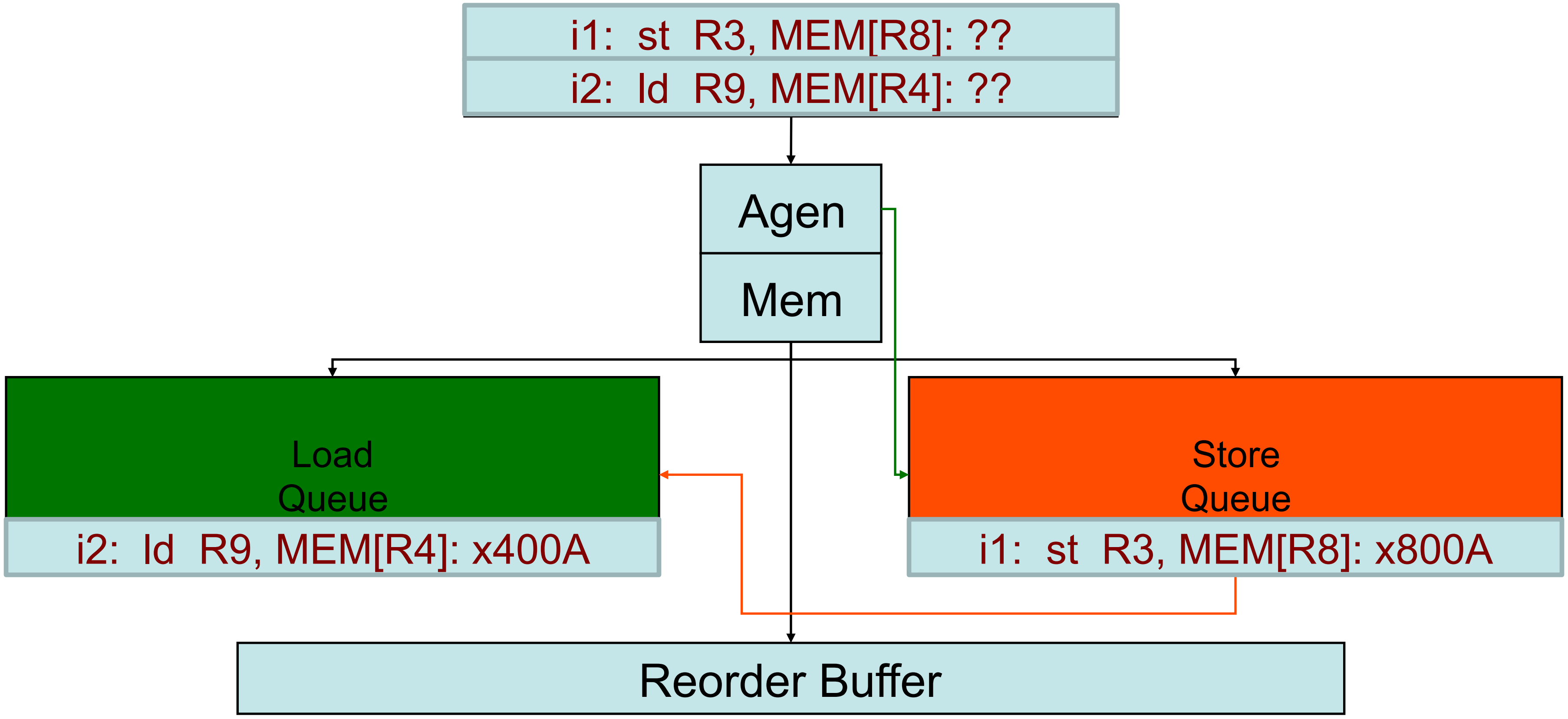
load r8, MEM[r9] → independent load stalled

Speculative Disambiguation

- What if aliases are rare?
  1. Loads don't wait for addresses of all prior stores
  2. Full address comparison of stores that are ready
  3. Bypass if no match, forward if match
  4. Check all store addresses when they commit
     - No matching loads – speculation was correct
     - Matching unbypassed load – incorrect speculation
  5. Replay starting from incorrect load

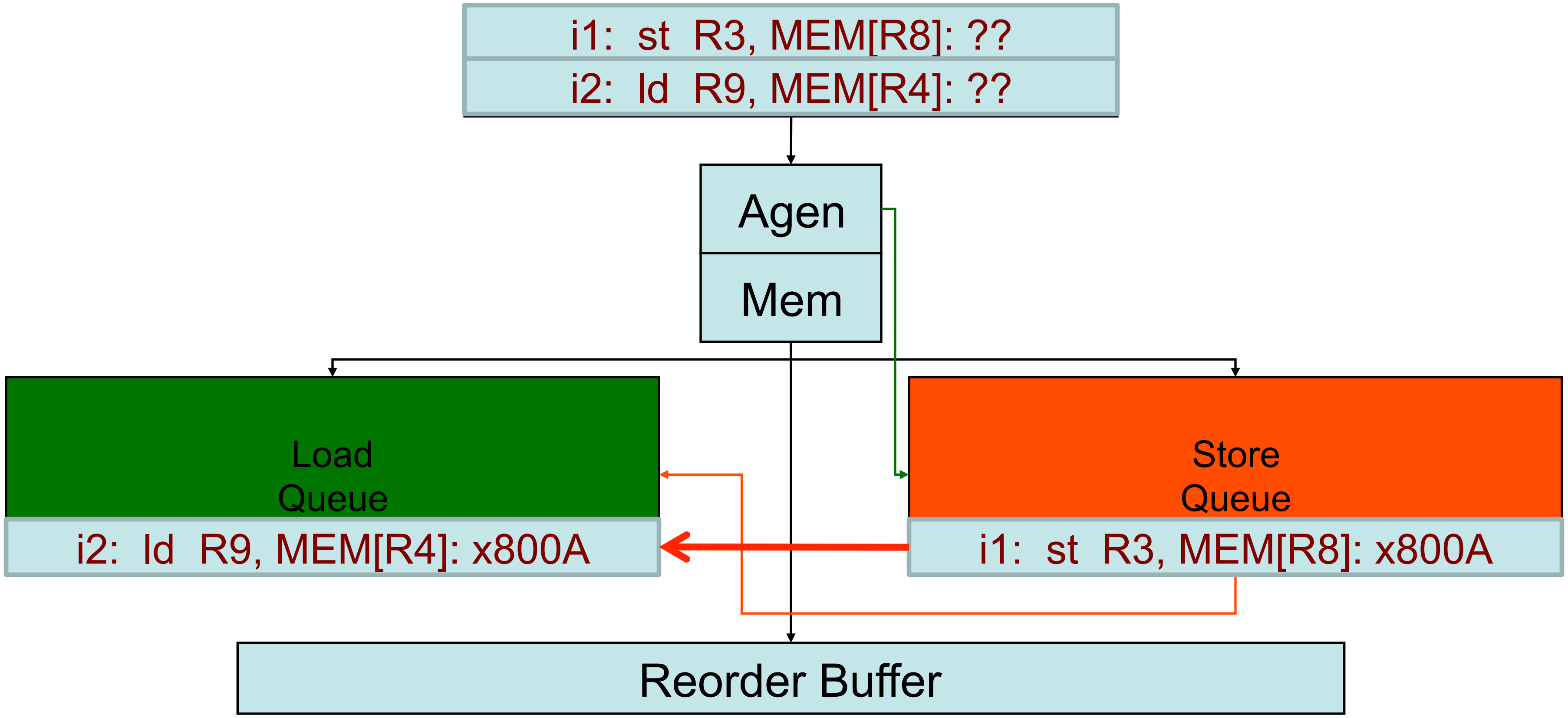


Speculative Disambiguation: Load Bypass



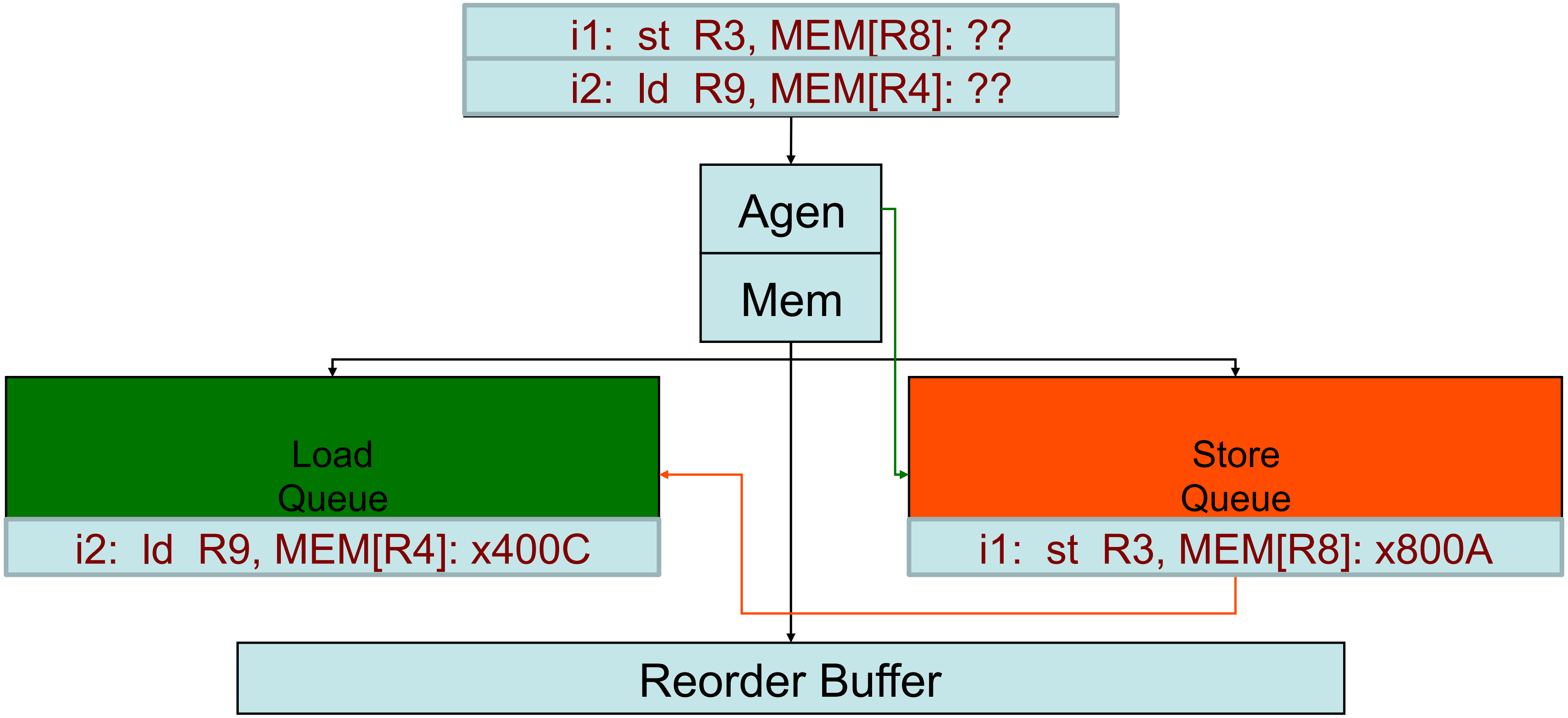
- i1 and i2 issue in program order
- i2 checks store queue (no match)

Speculative Disambiguation: Load Forward



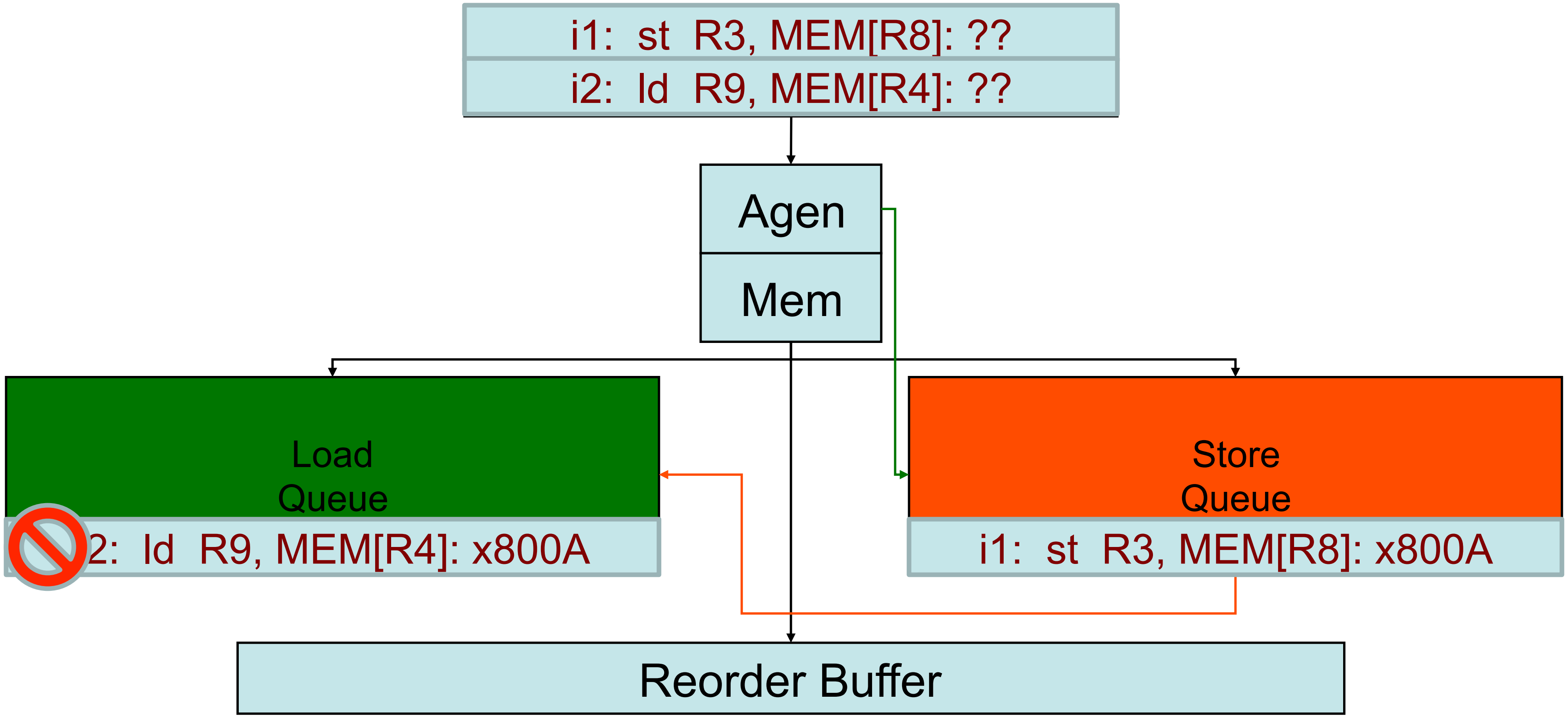
- i1 and i2 issue in program order
- i2 checks store queue (match=>forward)

Speculative Disambiguation: Safe Speculation



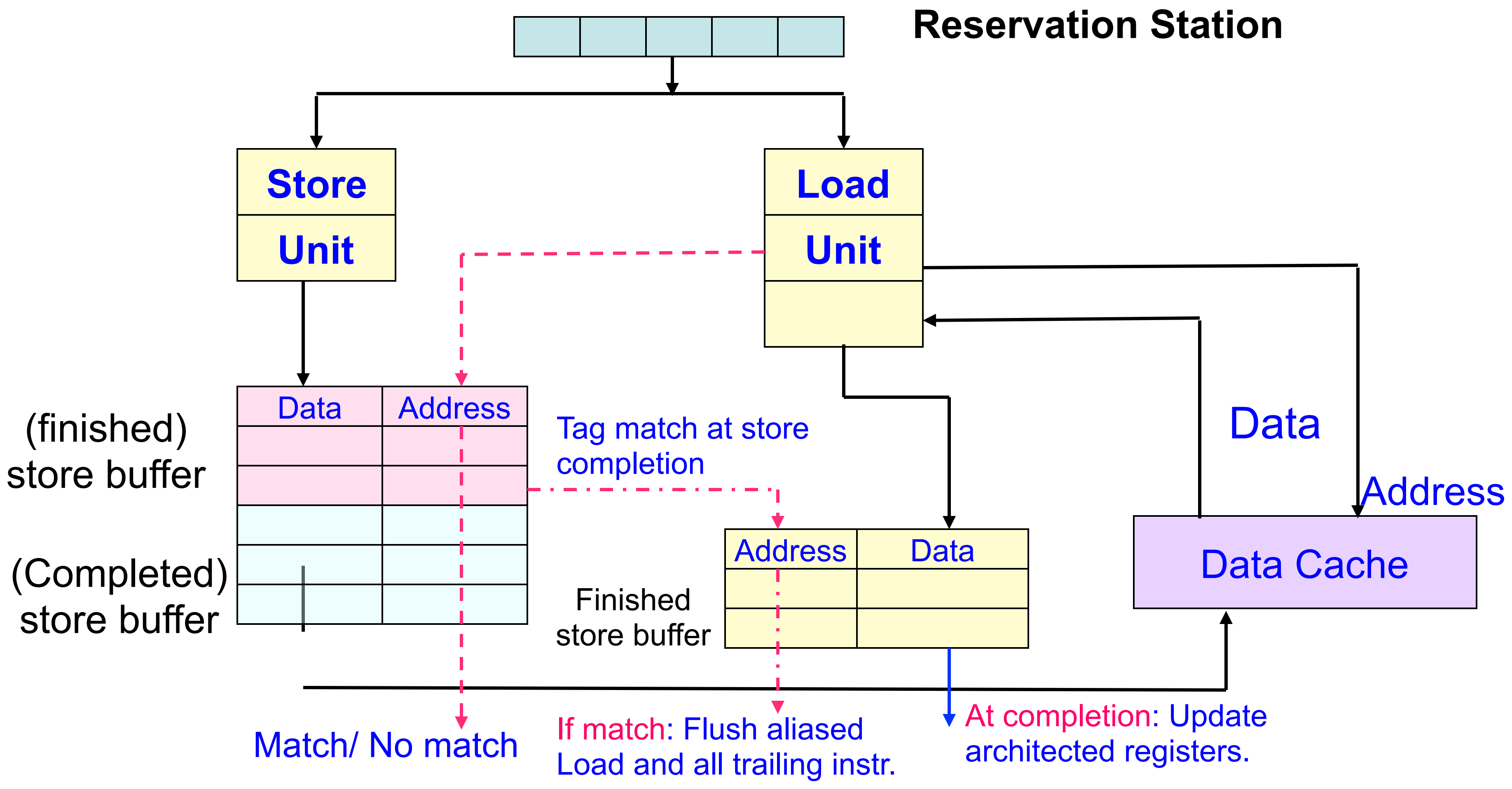
- i1 and i2 issue out of program order
- i1 checks load queue at commit (no match)

Speculative Disambiguation: Violation



- i1 and i2 issue out of program order
- i1 checks load queue at commit (match)
  - ➢ i2 marked for replay

Load/ Store Handling



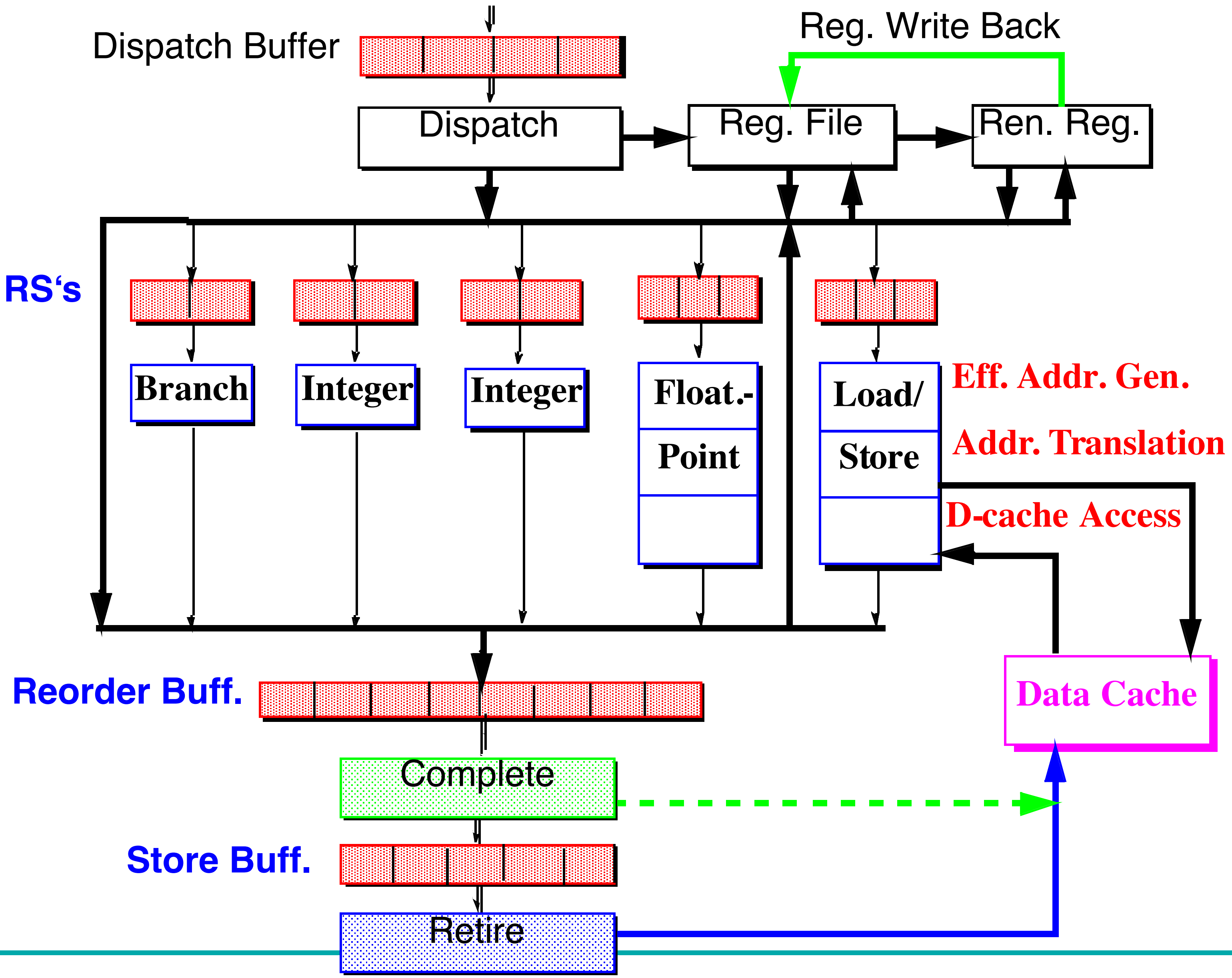
Use of Prediction

- If aliases are rare: static prediction
  - Predict no alias every time
    - Why even implement forwarding? PowerPC 620 doesn't
  - Pay misprediction penalty rarely
- If aliases are more frequent: dynamic prediction
  - Use PHT-like history table for loads
    - If alias predicted: delay load
    - If aliased pair predicted: forward from store to load
      - More difficult to predict pair [store sets, Alpha 21264]
  - Pay misprediction penalty rarely
- Memory cloaking [Moshovos, Sohi]
  - Predict load/store pair
  - Directly copy store data register to load target register
  - Reduce data transfer latency to absolute minimum

Load/Store Disambiguation Discussion

- RISC ISA:
  - Many registers, most variables allocated to registers
  - Aliases are rare
  - Most important to not delay loads (bypass)
  - Alias predictor may/may not be necessary
- CISC ISA:
  - Few registers, many operands from memory
  - Aliases much more common, forwarding necessary
  - Incorrect load speculation should be avoided
  - If load speculation allowed, predictor probably necessary
- Address translation:
  - Can't use virtual address (must use physical)
  - Wait till after TLB lookup is done
  - Or, use subset of untranslated bits (page offset)
    - Safe for proving inequality (bypassing OK)
    - Not sufficient for showing equality (forwarding not OK)

The Memory Bottleneck



Load/Store Processing

For both Loads and Stores:

1. **Effective Address Generation:**
   - Must wait on register value
   - Must perform address calculation
2. **Address Translation:**
   - Must access TLB
   - Can potentially induce a page fault (exception)

Load/Store Processing

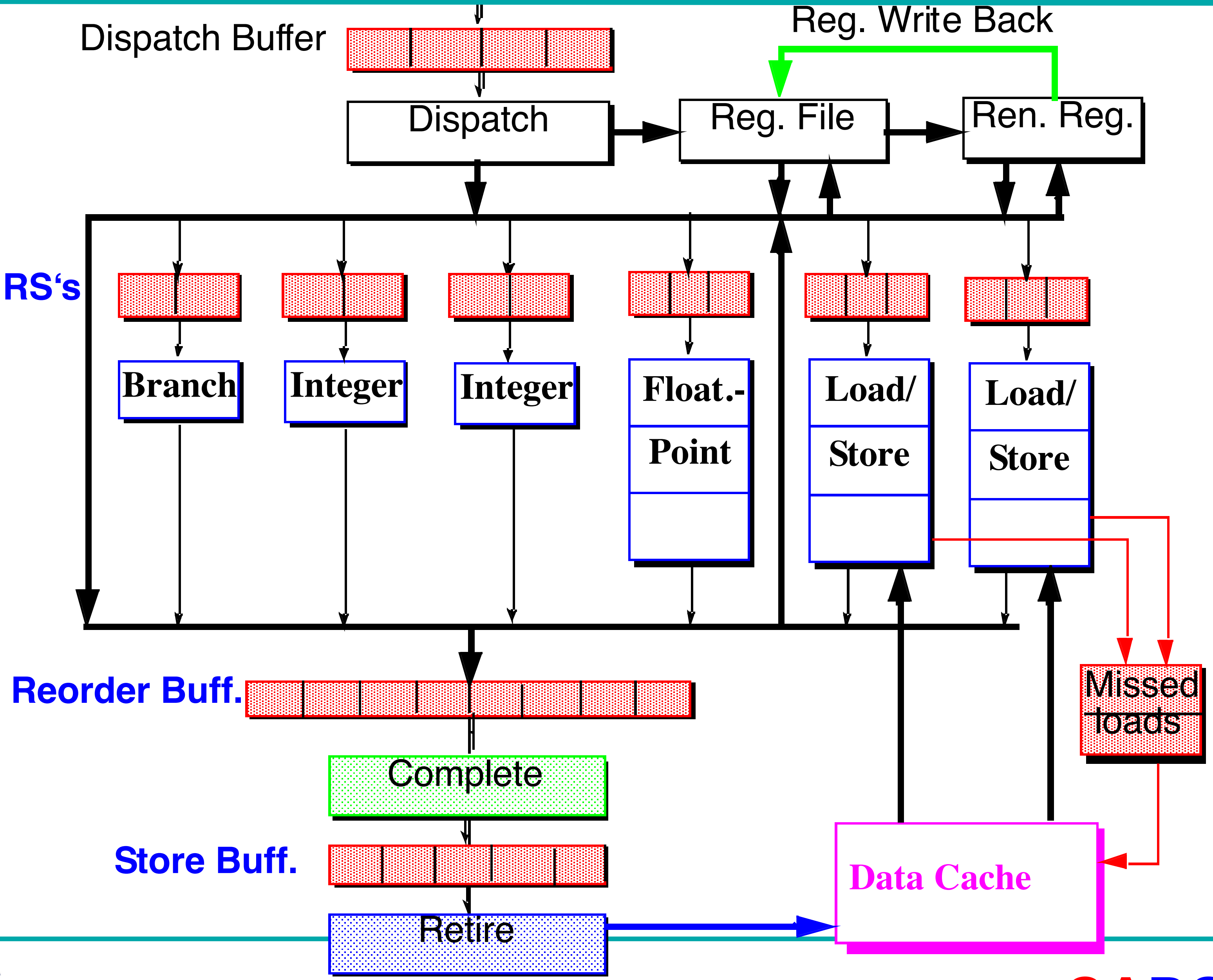
For Loads: D-cache Access (Read)

- Can potentially induce a D-cache miss
- Check aliasing against store buffer for possible load forwarding
- If bypassing store, must be flagged as speculative load until completion

For Stores: D-cache Access (Write)

- When completing must check aliasing against speculative loads
- After completion, wait in store buffer for access to D-cache
- Can potentially induce a D-cache miss

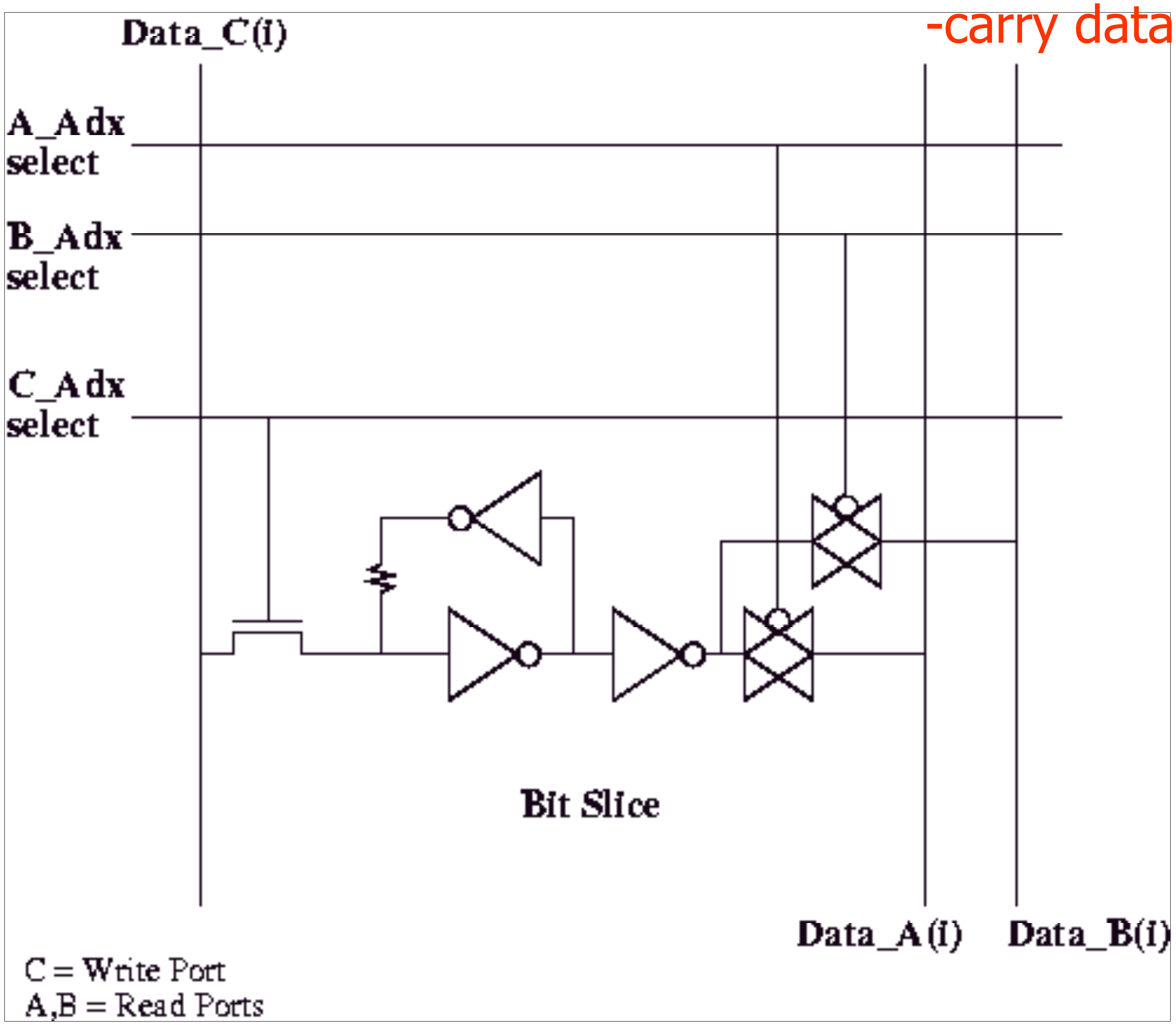
Easing The Memory Bottleneck



True Multi-porting of SRAM

"Bit" Lines
-carry data in/out

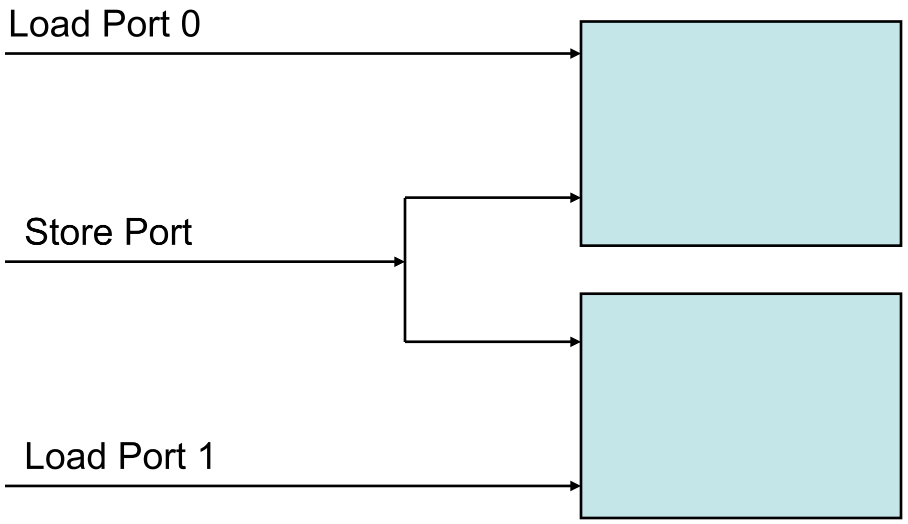
"Word" Lines
-select a row



True Multi-porting of SRAM

- Would be ideal
- Increases cache area
  - Array becomes wire-dominated
- Slower access
  - Wire delay across larger area
  - Cross-coupling capacitance between wires
- SRAM access difficult to pipeline

Multiple Cache Copies



- Used in DEC Alpha 21164, IBM Power4
- Independent load paths
- Single shared store path
  - May be exclusive with loads, or internally dual-ported
- Bottleneck, not practically scalable beyond 2 paths
- Provides some fault-tolerance
  - Parity protection per copy

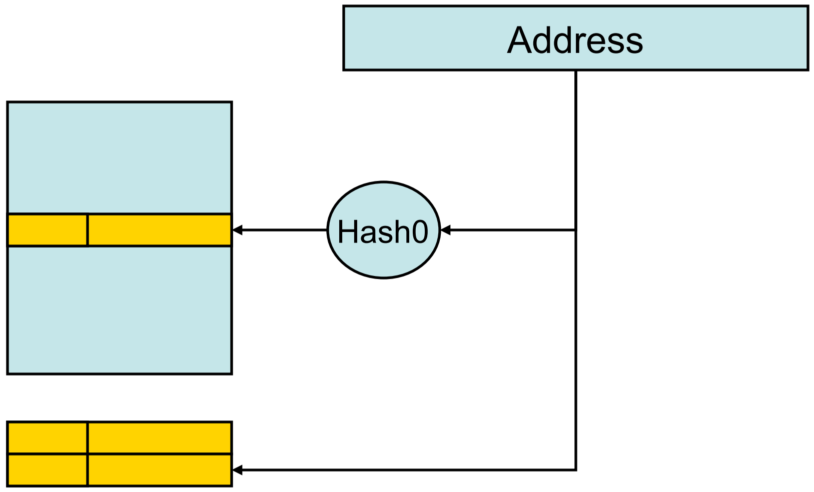
Memory Bottleneck Techniques

Dynamic Hardware (Microarchitecture):

- Use Multiple Load/Store Units (need multiported D-cache)
- Use More Advanced Caches (victim cache, stream buffer)
- Use Hardware Prefetching (need load history and stride detection)
- Use Non-blocking D-cache (need missed-load buffers/MSHRs)
- Large instruction window (memory-level parallelism)

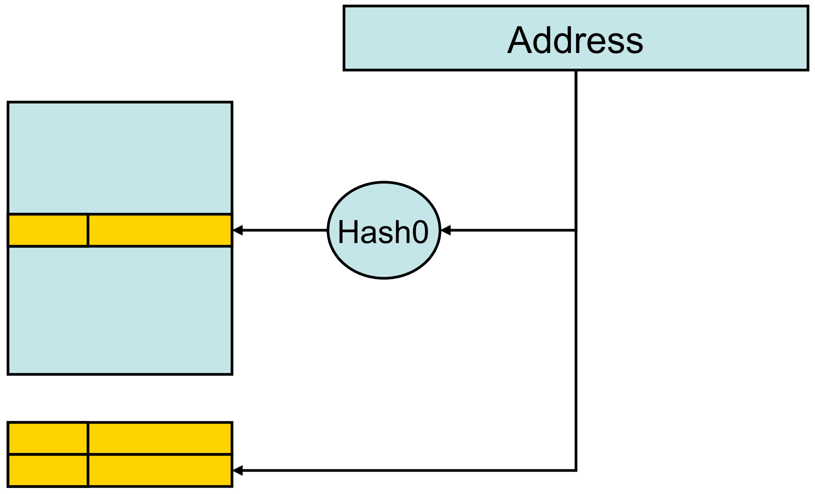
Jouppi's Victim Cache

- Targeted at conflict misses
- Victim cache: a small fully associative cache
  - holds victims replaced in direct-mapped or low-associativity
  - LRU replacement
  - a miss in cache + a hit in victim cache
    - => move line to main cache



Jouppi's Victim Cache

- Removes conflict misses, mostly useful for DM or 2-way SA
  - Even one entry helps some benchmarks
  - I-cache helped more than D-cache
- Versus cache size
  - Generally, victim cache helps more for smaller caches
- Versus line size
  - helps more with larger line size
- Used in Pentium Pro (P6) I-cache



Prefetching

- Even "demand fetching" prefetches other words in block
  - Spatial locality
- Prefetching is useless
  - Unless a prefetch costs less than demand miss
- Ideally, prefetches should
  - ✓ Always get data before it is referenced
  - ✓ Never get data not used
  - ✓ Never prematurely replace data
  - ✓ Never interfere with other cache activity

Software Prefetching

- Use compiler to try to
  - Prefetch early
  - Prefetch accurately
- Prefetch into
  - Register (binding)
    - Use normal loads? Stall-on-use (Alpha 21164)
    - What about page faults? Exceptions?
  - Caches (non-binding) – preferred
    - Needs ISA support

Software Prefetching

- For example:

```
do j= 1, cols
   do ii = 1 to rows by BLOCK
     prefetch (&(x[i,j])+BLOCK)    # prefetch one block ahead
      do i = ii to ii + BLOCK-1
        sum = sum + x[i,j]
```

- How many blocks ahead should we prefetch?
  - Affects timeliness of prefetches
  - Must be scaled based on miss latency

Hardware Prefetching

- What to prefetch
  - One block spatially ahead
  - N blocks spatially ahead
  - Based on observed stride, track/prefetch multiple strides
- Training hardware prefetcher
  - On every reference (expensive)
  - On every miss (information loss)
  - Misses at what level of cache?
  - Prefetchers at every level of cache?
- Pressure for nonblocking miss support (MSHRs)

Stream or Prefetch Buffers

- Prefetching causes capacity and conflict misses (pollution)
  - Can displace useful blocks
- Aimed at compulsory and capacity misses
- Prefetch into buffers, NOT into cache
  - On miss start filling stream buffer with successive lines
  - Check both cache and stream buffer
    - Hit in stream buffer => move line into cache (promote)
    - Miss in both => clear and refill stream buffer
- Performance
  - Very effective for I-caches, less for D-caches
  - Multiple buffers to capture multiple streams (better for D-caches)
- Can use with any prefetching scheme to avoid pollution

Summary : Prefetching

- Prefetching anticipates future memory references
  - Software prefetching
  - Next-block, stride prefetching
  - Global history buffer prefetching
- Issues/challenges
  - Accuracy
  - Timeliness
  - Overhead (bandwidth)
  - Conflicts (displace useful data)

Memory Bottleneck Techniques

Static Software (Code Transformation):

- Insert Prefetch or Cache-Touch Instructions (mask miss penalty)
- Array Blocking Based on Cache Organization (minimize misses)
- Reduce Unnecessary Load/Store Instructions (redundant loads)
- Software Controlled Memory Hierarchy (expose it to above DSI)

Thank You