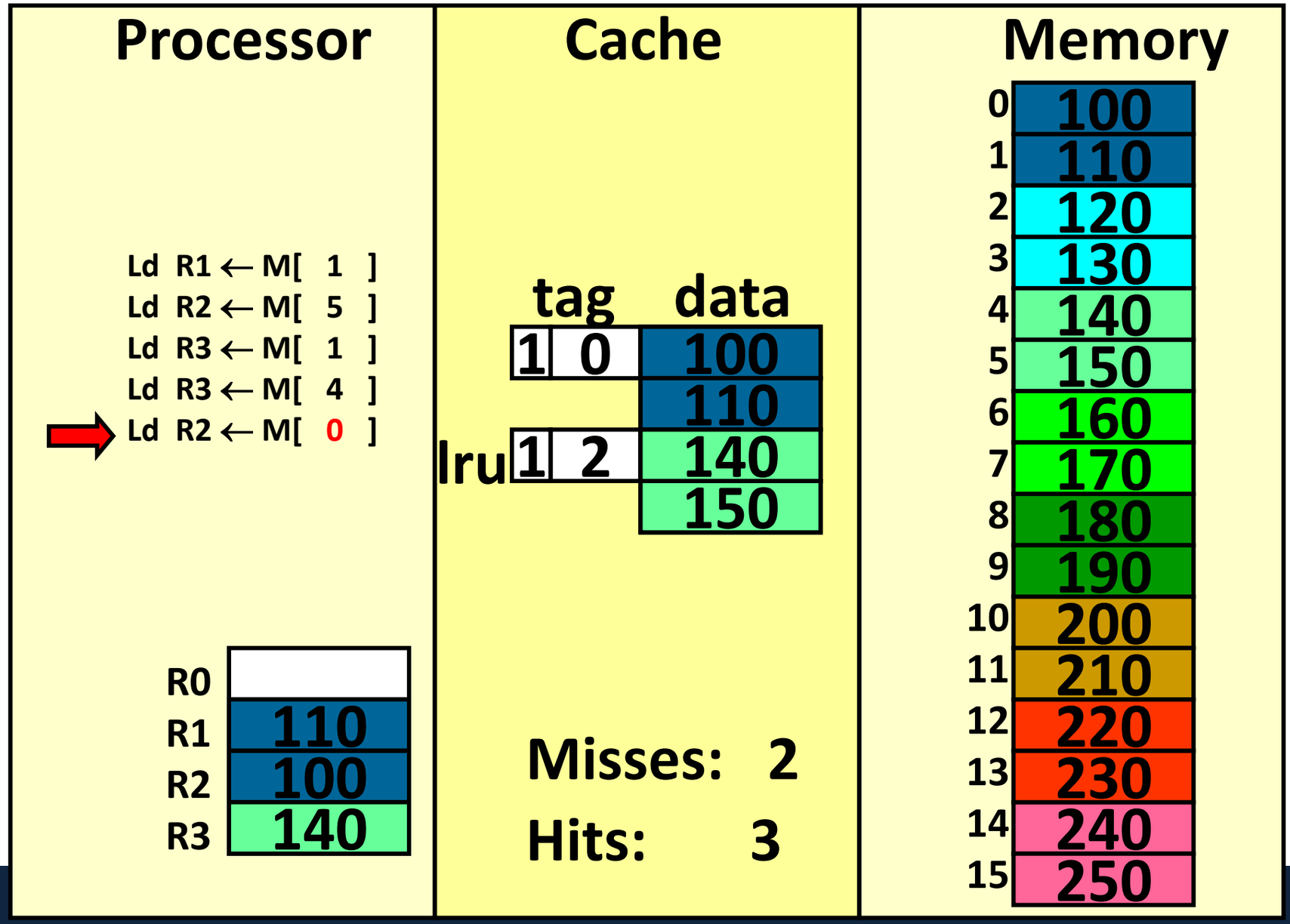
# EECS 370

## Direct Mapped Caches

# Announcements

- Project 3
  - Checkpoint due Thursday
  - Full submission due next Thursday
- HW 3
  - Due next Friday
  - Everything relevant has been covered in lecture
- Labs
  - No lab this Friday / next Monday (no prelab either)
  - Extra OH for P3 available instead (see calendar)

# Previously


Processor
Cache
Memory
Ld R1 ← M[ 1 ]
Ld R2 ← M[ 5 ]
Ld R3 ← M[ 1 ]
Ld R3 ← M[ 4 ]
Ld R2 ← M[ 0 ]
tag data
1 0 100
110
lru 1 2 140
150
R0
R1 110
R2 100
R3 140
Misses: 2
Hits: 3
0 100
1 110
2 120
3 130
4 140
5 150
6 160
7 170
8 180
9 190
10 200
11 210
12 220
13 230
14 240
15 250

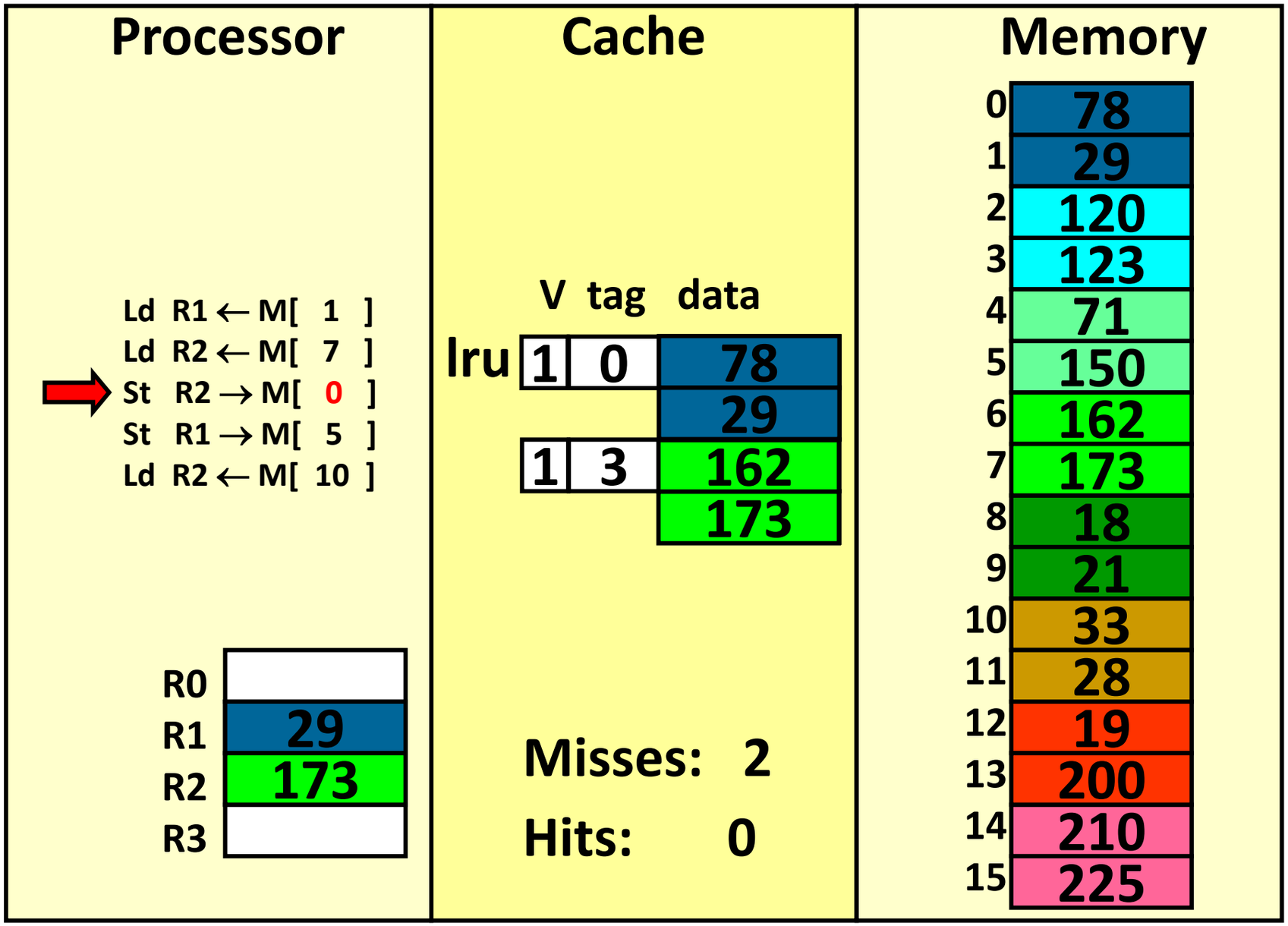
# What about stores?

- Where should you write the result of a store?
  - If that memory location is in the cache:
    - Send it to the cache.
    - Should we also send it to memory?
      (write-through policy)
  - If it is not in the cache:
    - Allocate the line (put it in the cache)?
      (allocate-on-write policy)
    - Write it directly to memory without allocation?
      (no allocate-on-write policy)

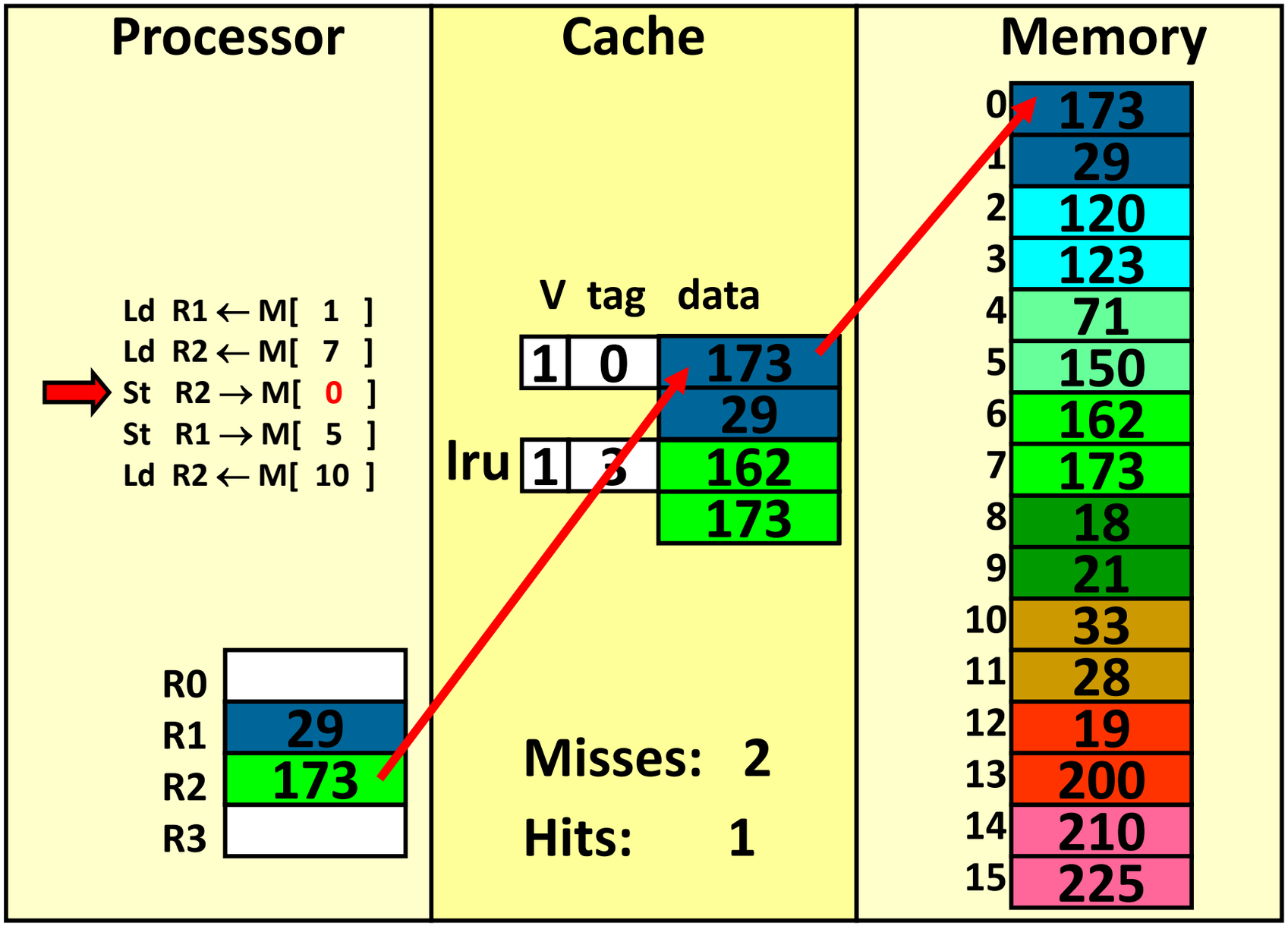
# Agenda

- Larger Cache Blocks
- Extra Problems
- LRU with More than Two Blocks
- **Write-Through Cache**
- Write-Back Cache

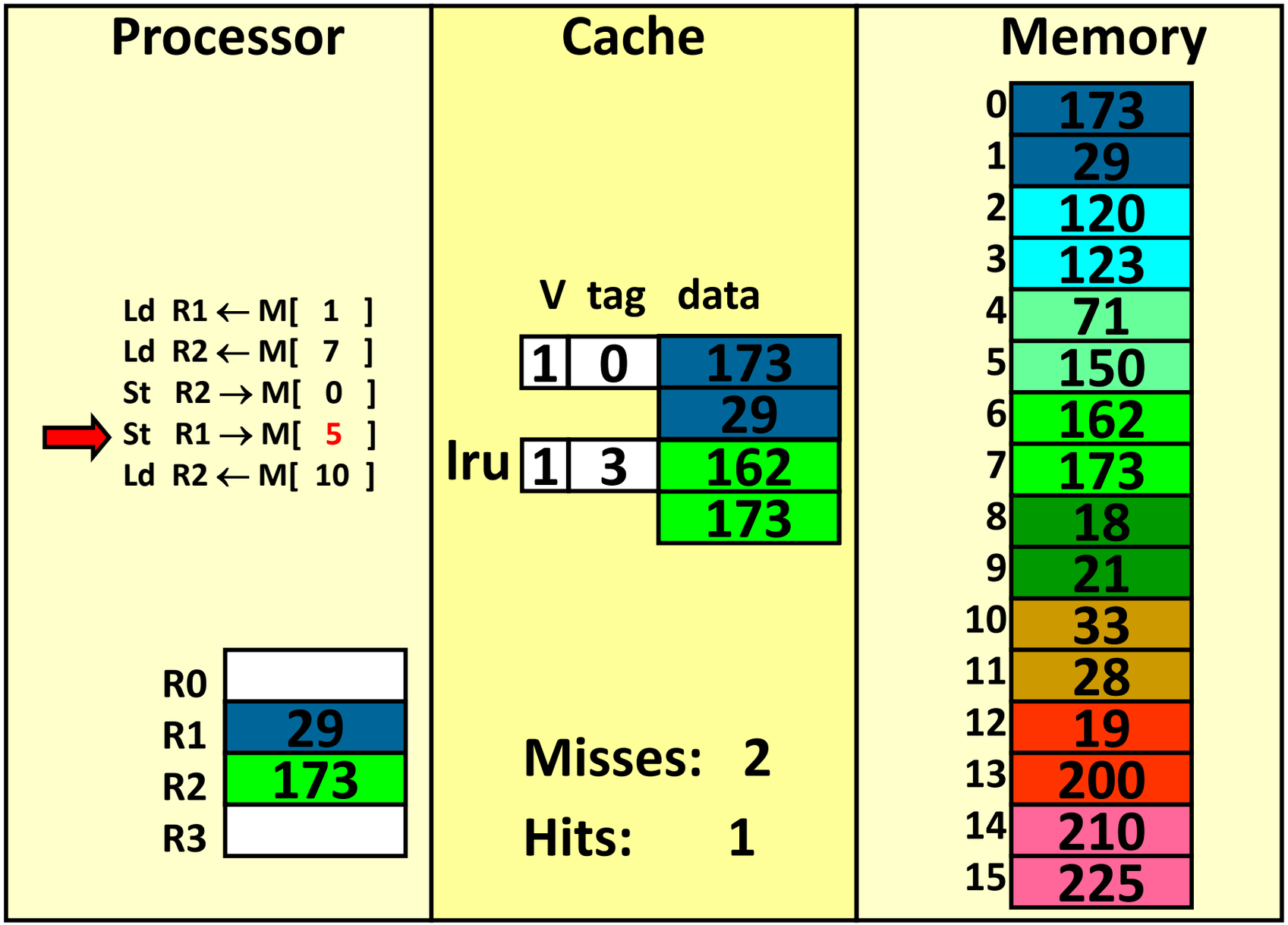
# write-through, allocate on write (REF 3)



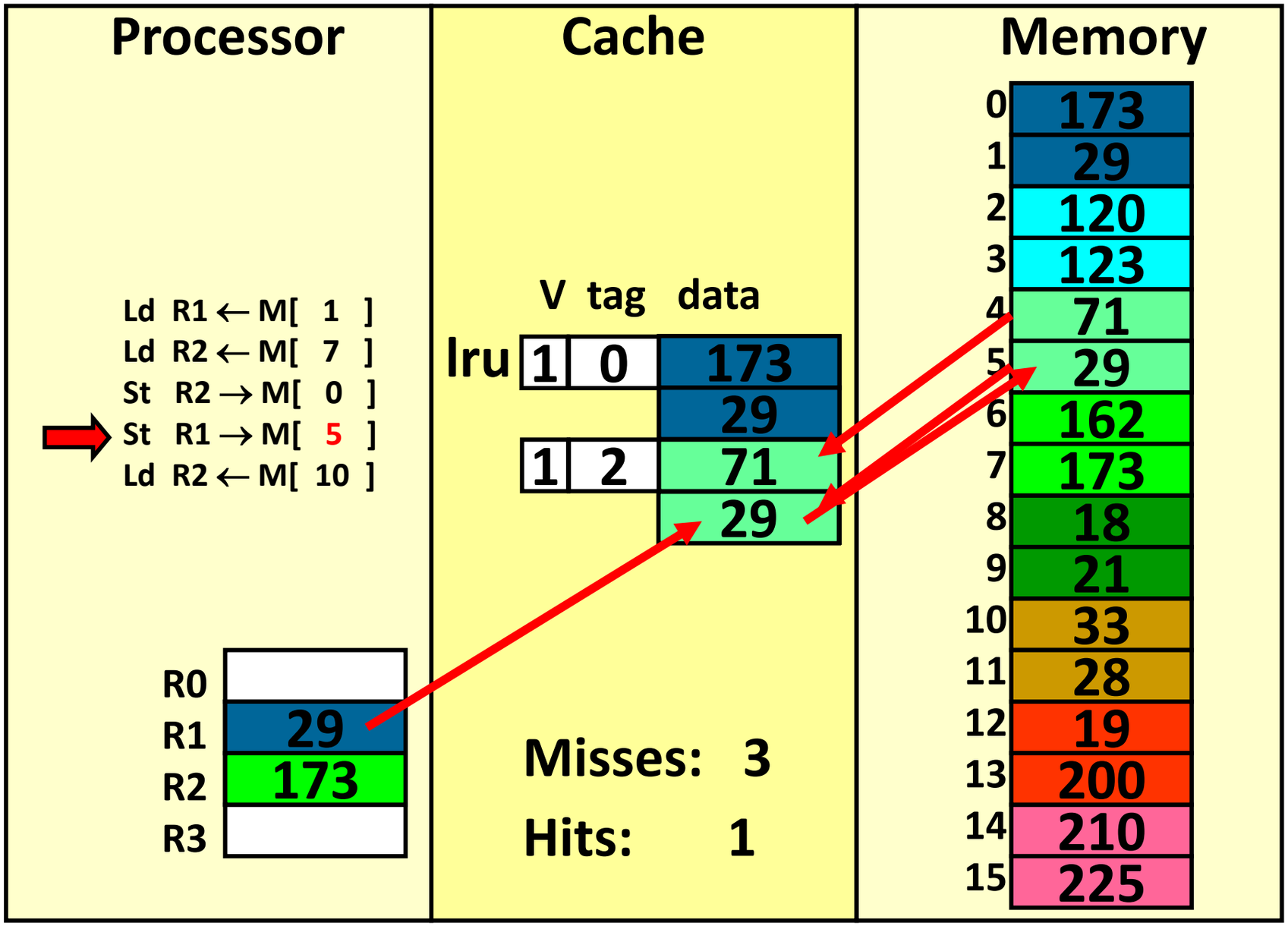
# write-through, allocate on write (REF 3)



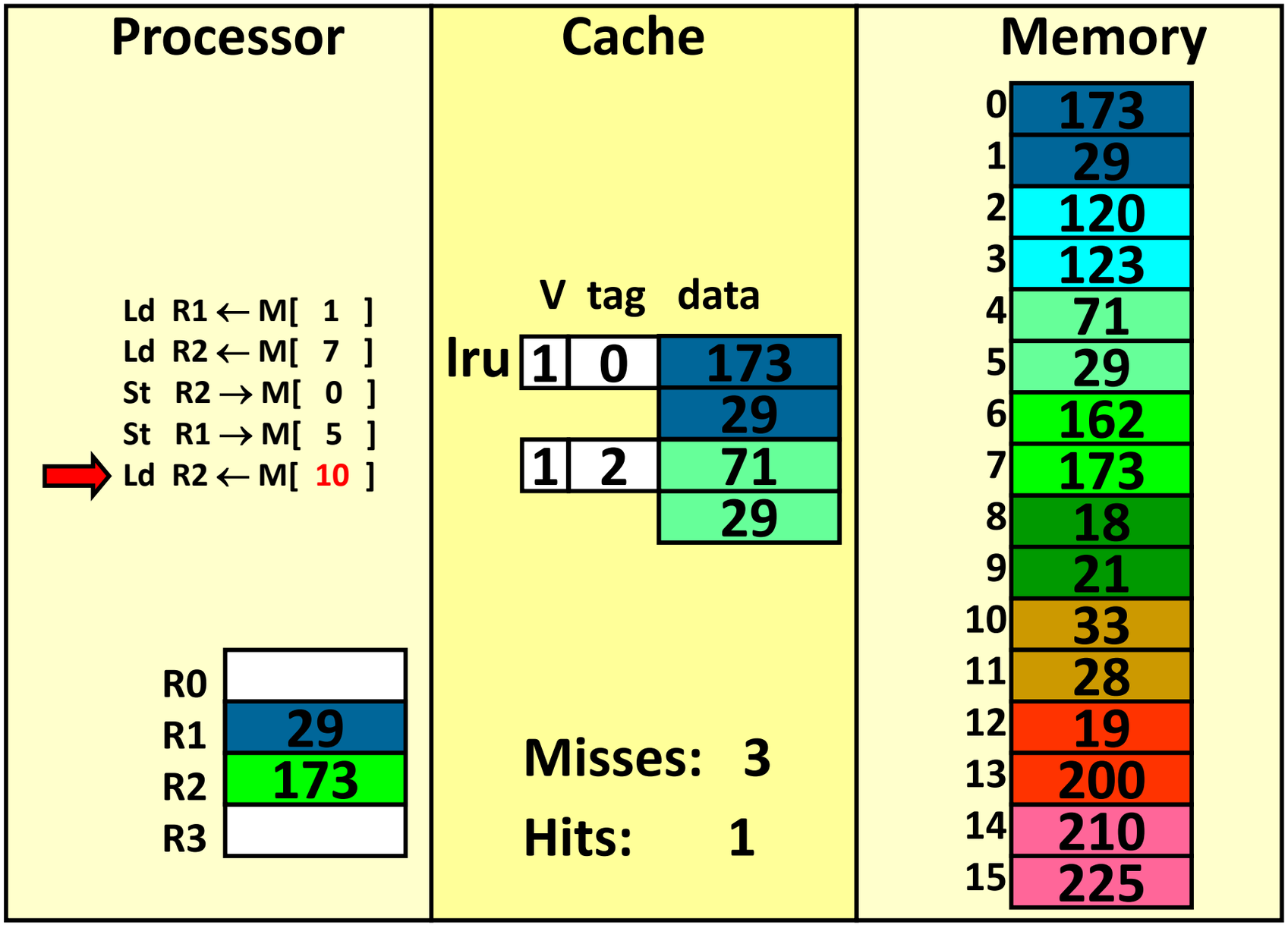
# write-through, allocate on write (REF 4)



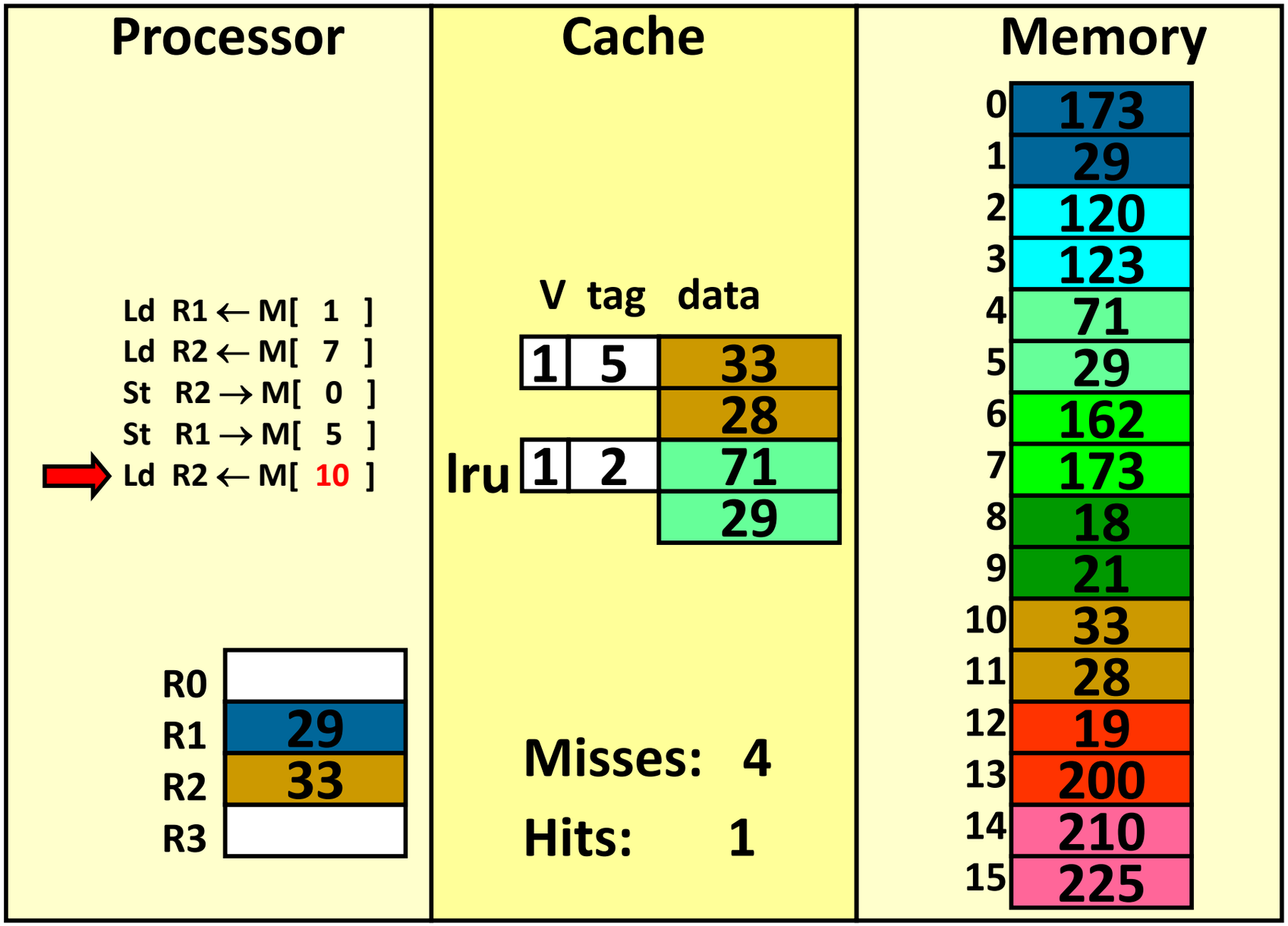
# write-through, allocate on write (REF 4)



# write-through, allocate on write (REF 6)



# write-through, allocate on write (REF 6)



# How many memory references?

- Each miss reads a block
  - 2 bytes in this cache
- Each store writes a byte
- Total reads: 8 bytes
- Total writes: 2 bytes

- but caches generally miss < 20%
  - Can we take advantage  of that?
  - Multi-core processors have limited bandwidth between caches and memory
  - Extra stores also cost power

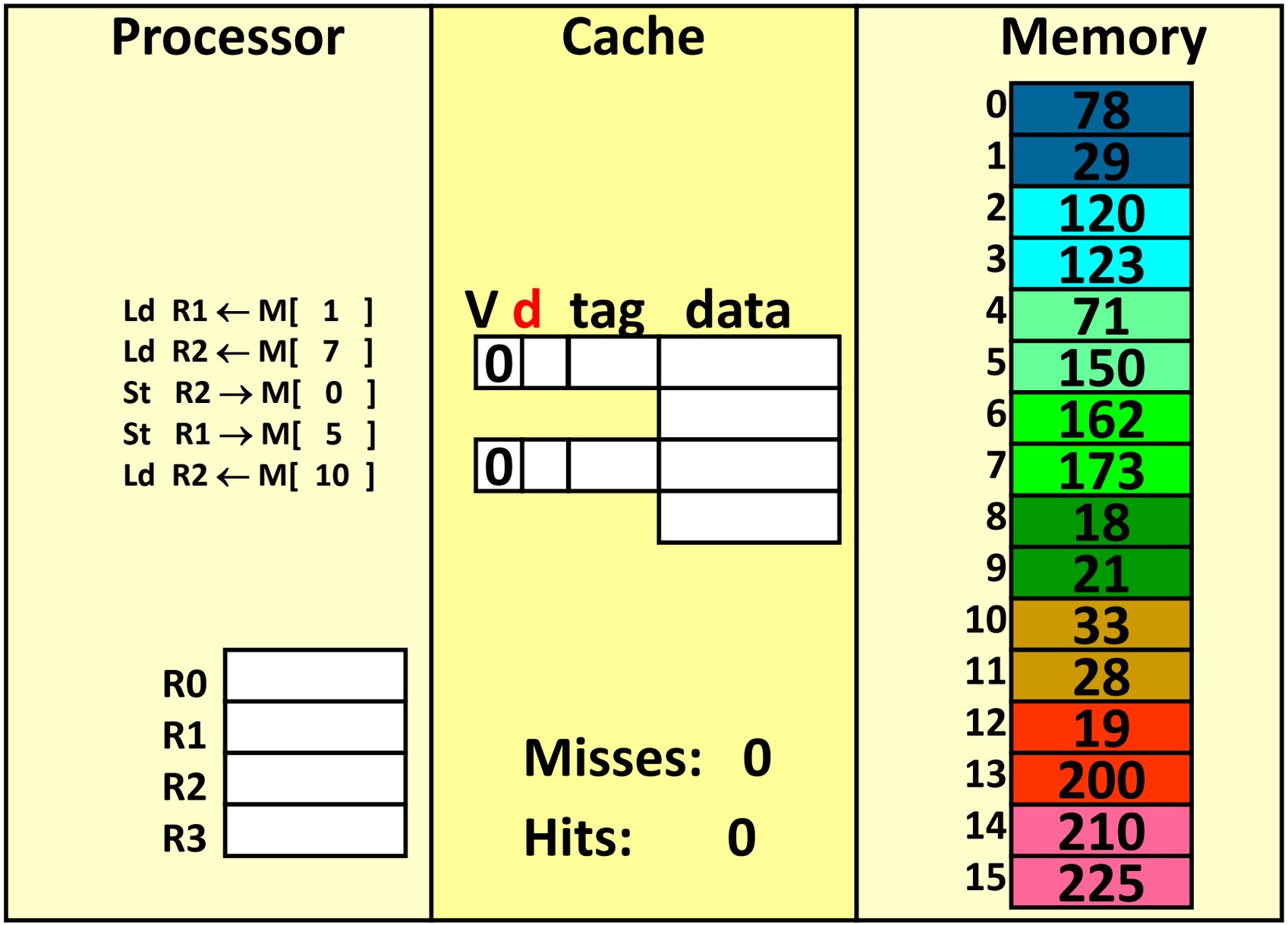
# Agenda

- Larger Cache Blocks
- Extra Problems
- LRU with More than Two Blocks
- Write-Through Cache
- **Write-Back Cache**

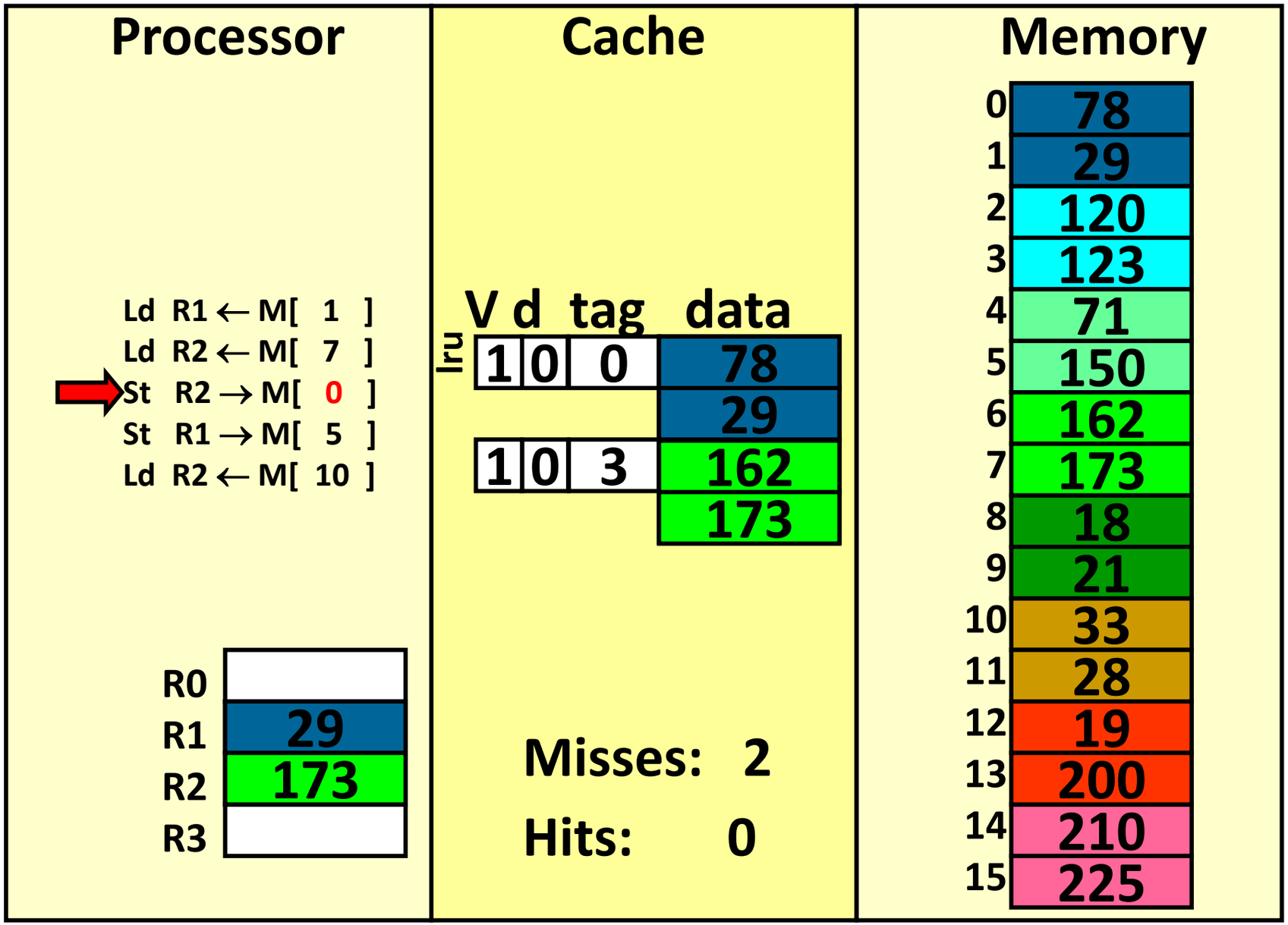
# Write-through vs write-back

- Can we design the cache to NOT write all stores to memory immediately?
  - Keep the most recent copy in the cache and update the memory **only when** that data is evicted from the cache (write-back)
  - Do we need to write-back all evicted lines?
    - No, only blocks that have been modified
    - Keep a "dirty bit", reset when the line is allocated, set when the block is stored into. If a block is "dirty" when evicted, write its data back into memory.

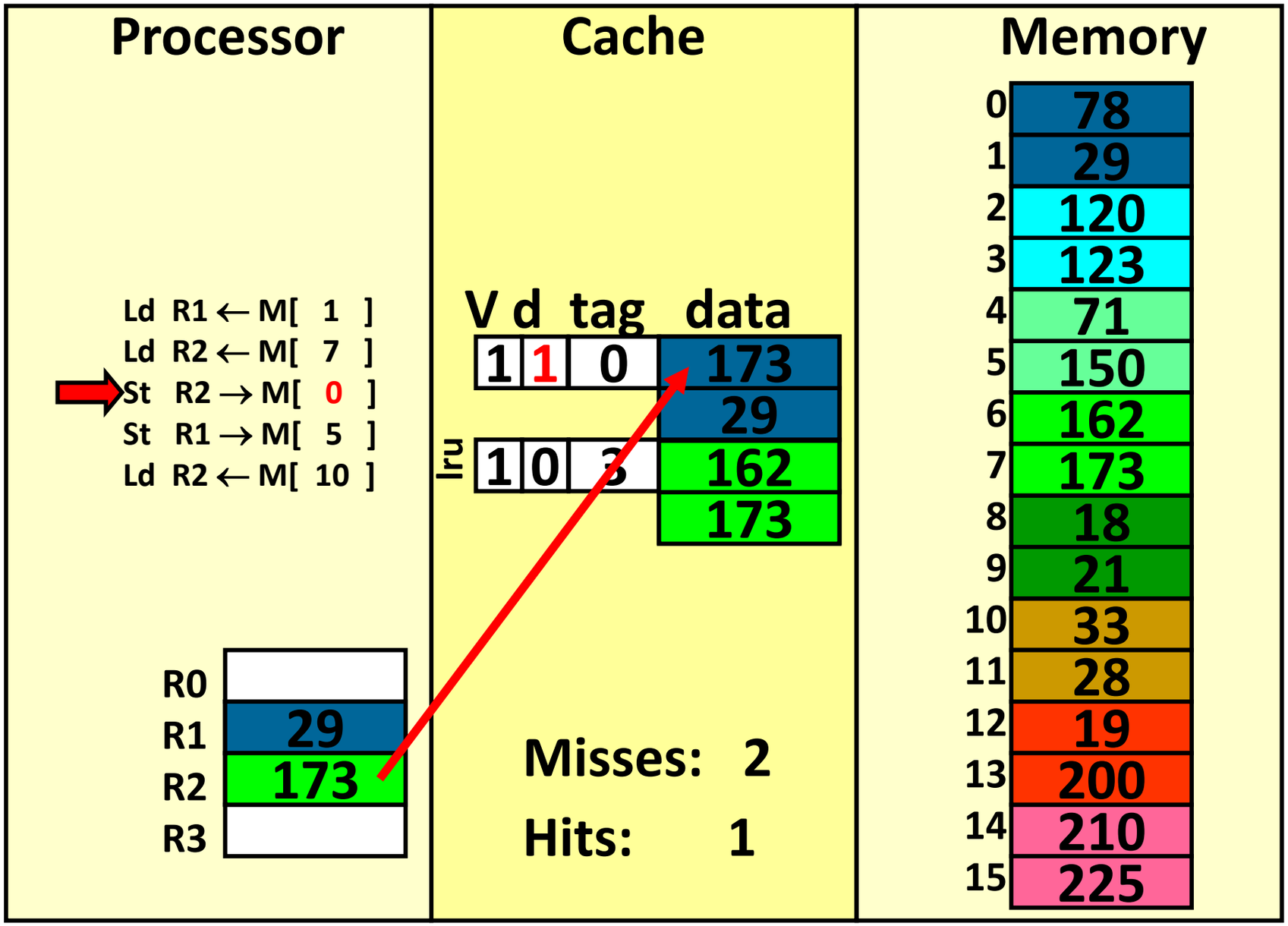
# Handling stores (write-back)



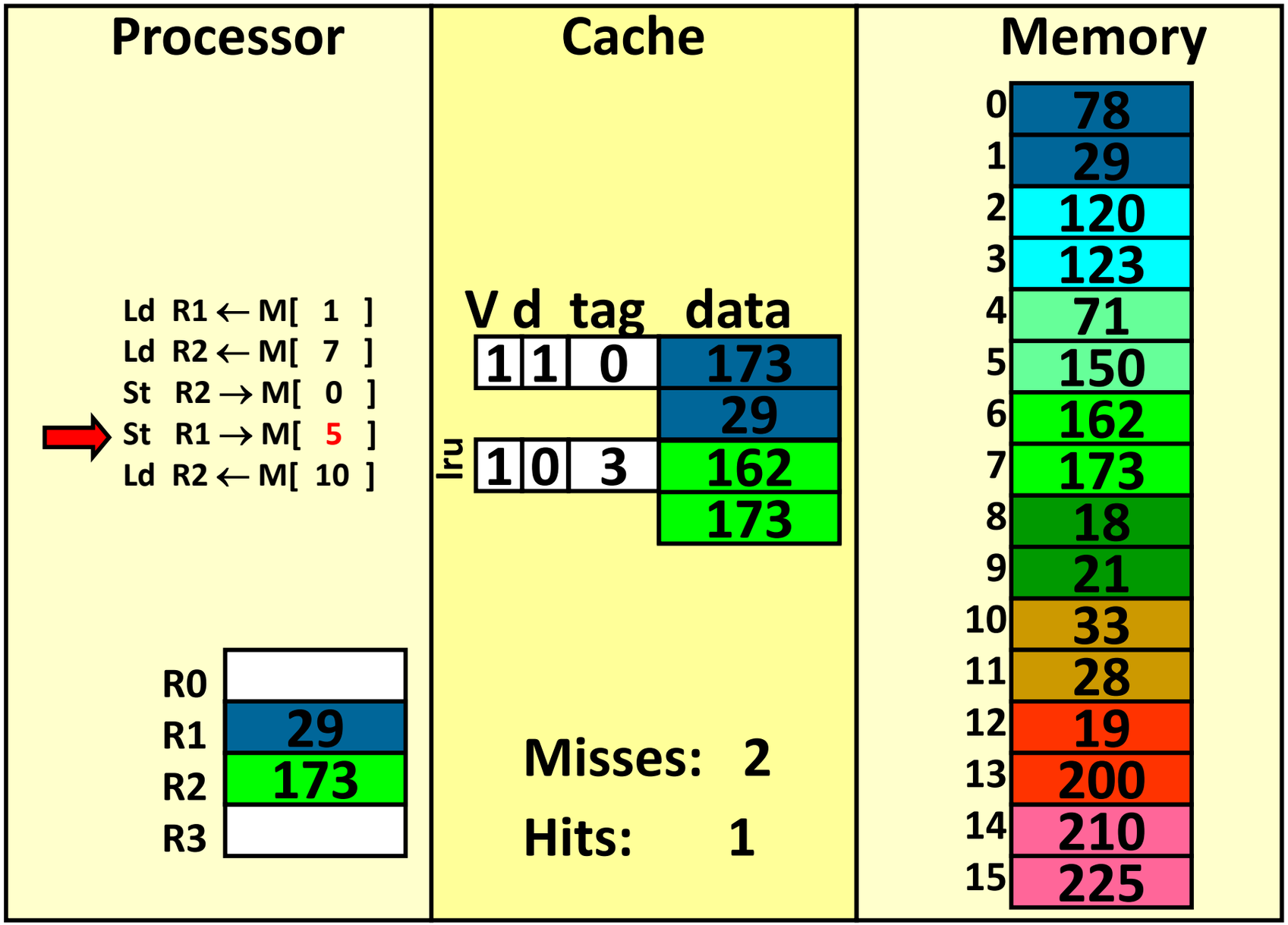
# write-back (REF 3)



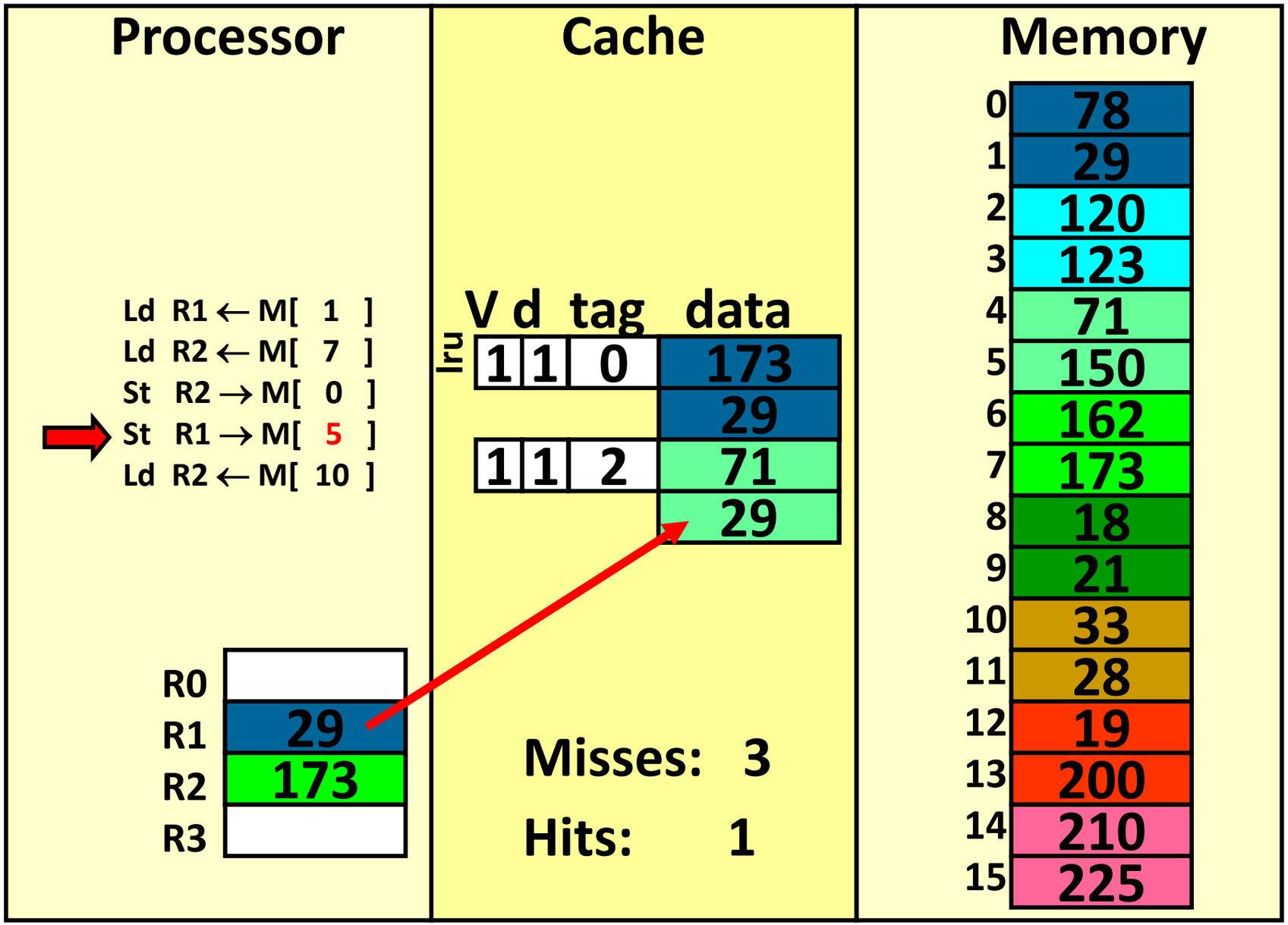
# write-back (REF 3)



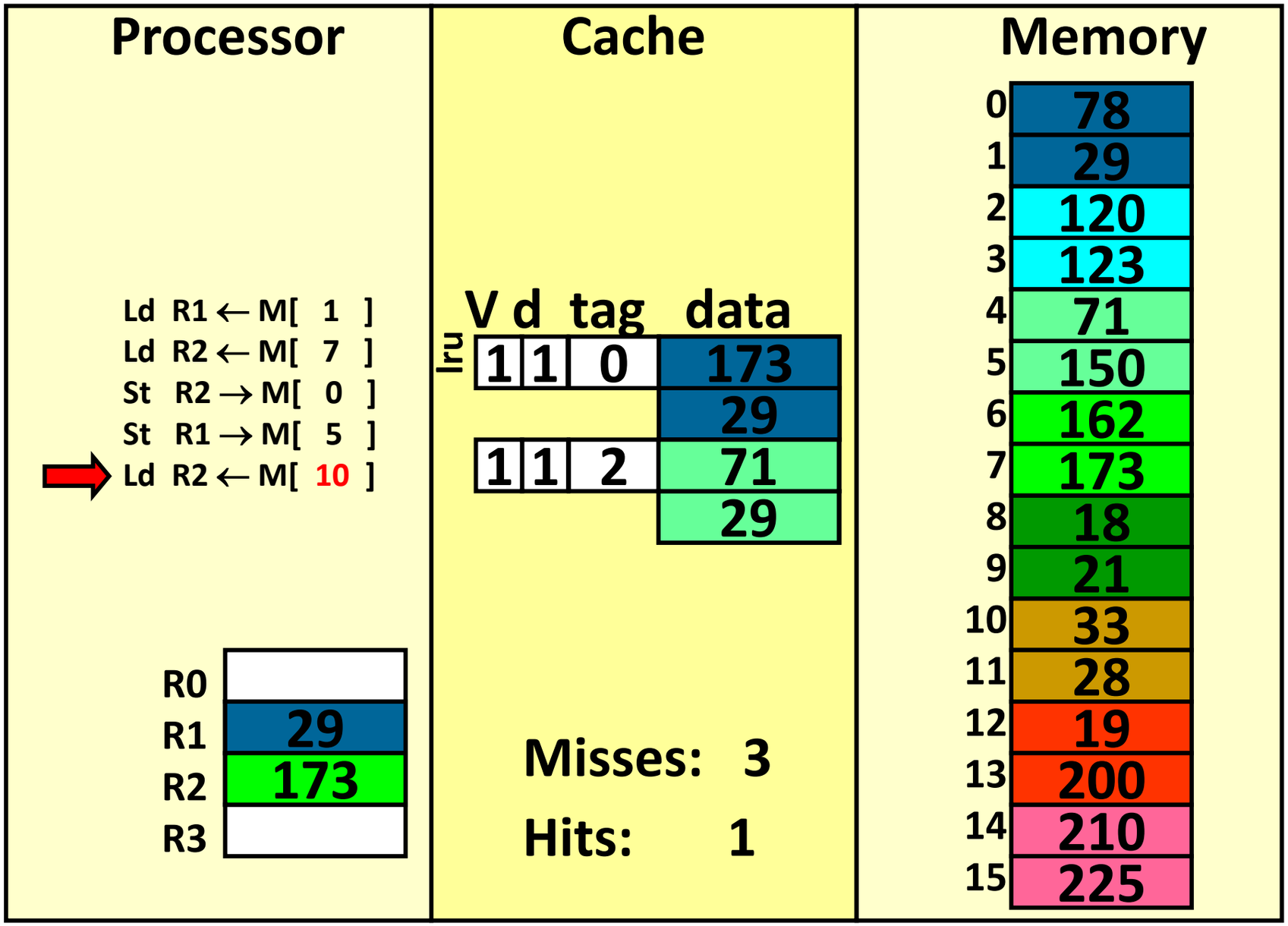
# write-back (REF 4)



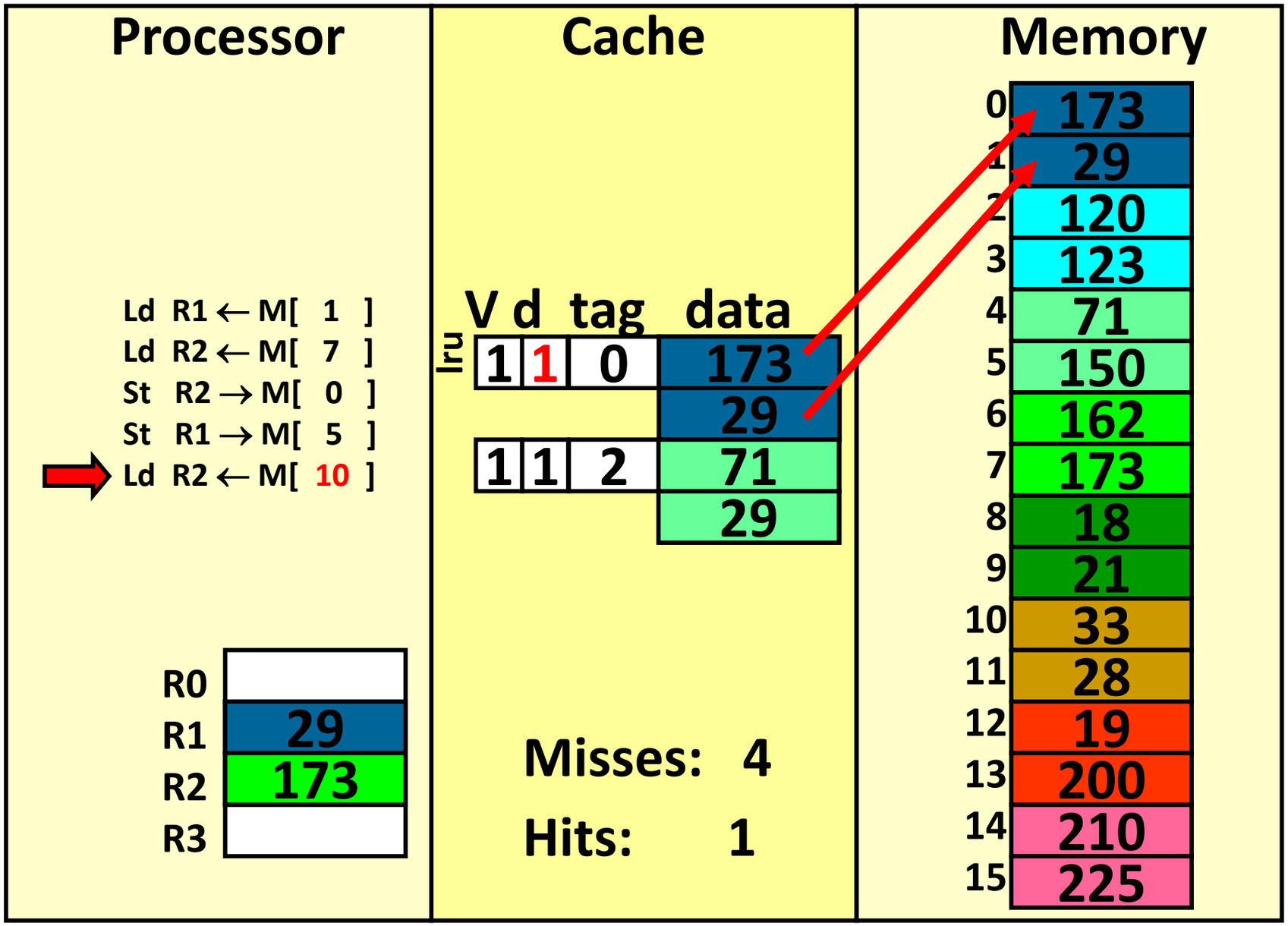
# write-back (REF 4)



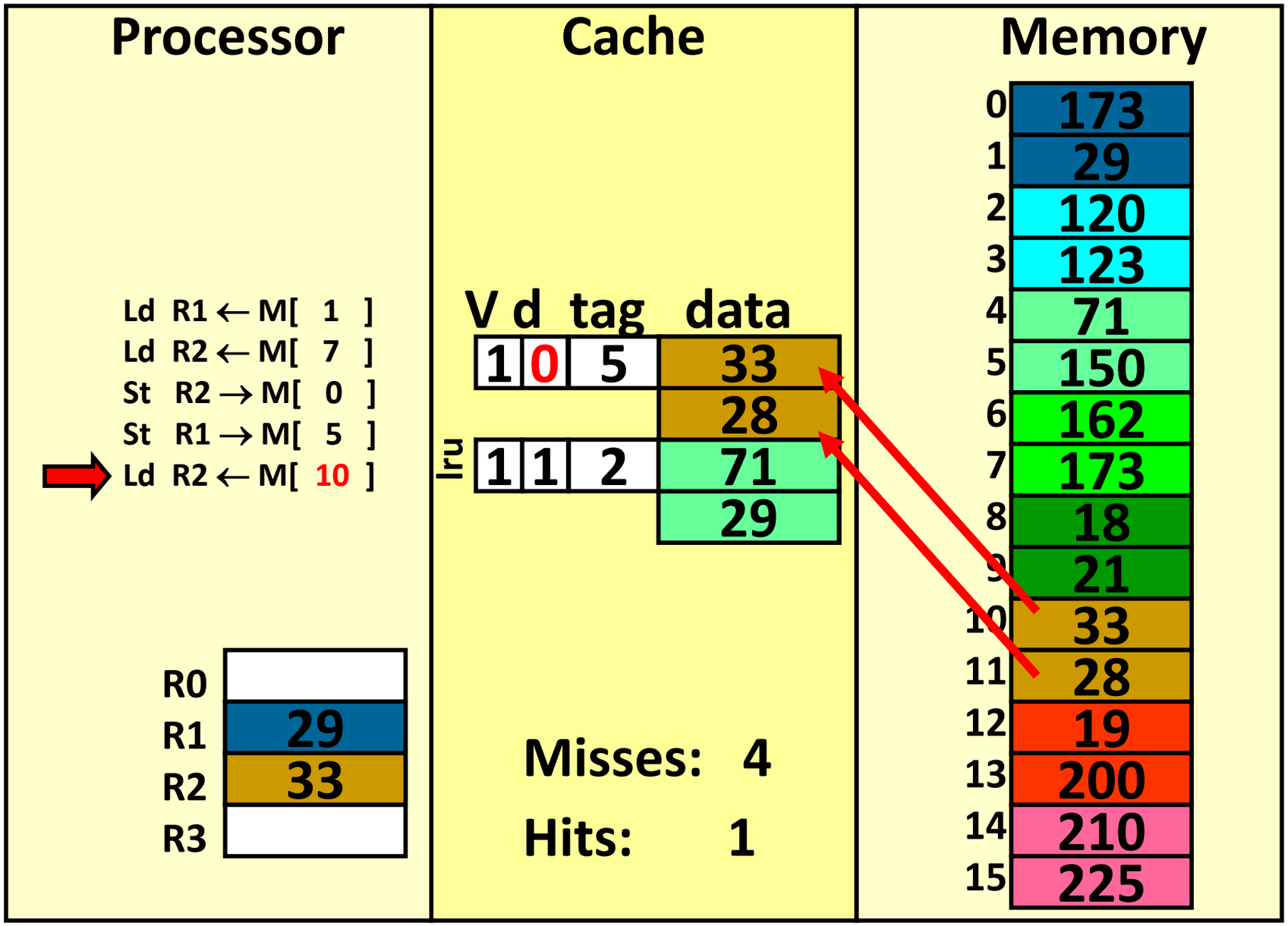
# write-back (REF 5)



# write-back (REF 5)



# write-back (REF 5)



# How many memory references?

- Each miss reads a block
  - 2 bytes in this cache
- Each evicted dirty cache line writes a block
- Total reads: 8 bytes
- Total writes: 4 bytes (after final eviction)

For this example, would you choose write-back or write-through?

Write-back works best when we write to a particular address multiple times before evicting

# Review: Writes

| Store w No Allocate | Write-Back | Write-Through |
|---|---|---|
| Hit? | Write Cache | Write to Cache **+ Memory** |
| Miss? | Write to Memory | Write to Memory |
| Replace block? | If evicted block is dirty, write to Memory | Do Nothing |

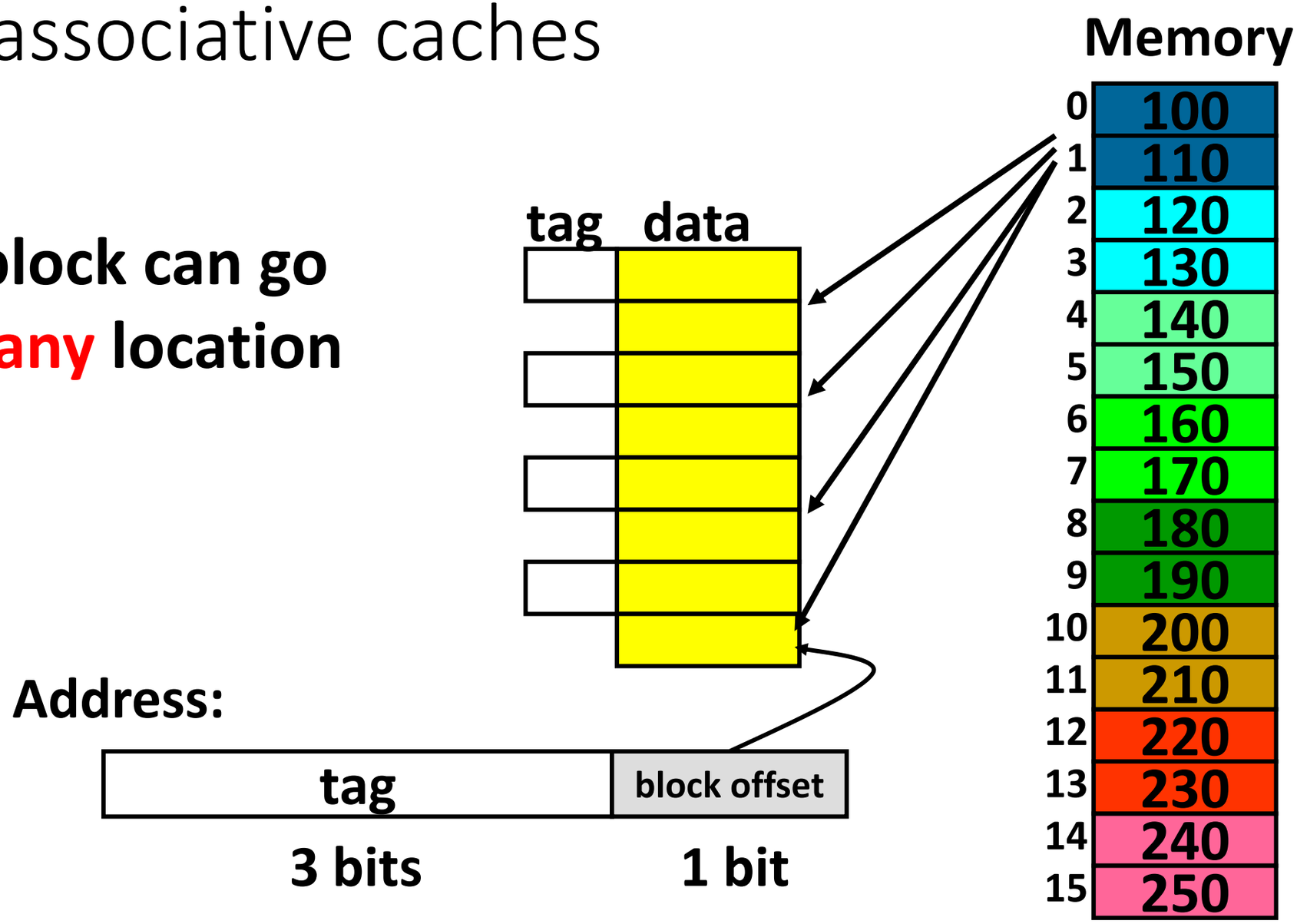
| Store w Allocate | Write-Back | Write-Through |
|---|---|---|
| Hit? | Write Cache | Write to Cache **+ Memory** |
| Miss? | Read from Memory to Cache,<br>Allocate to LRU block<br>Write to Cache | Read from Memory to Cache,<br>Allocate to LRU block<br>Write to Cache **+ Memory** |
| Replace block? | If evicted block is dirty, write to Memory | Do Nothing |

# Agenda

- **Fully-associative vs Direct Mapped Cache**
- Example
- Class Problems

# Fully-associative caches

**A block can go to any location**



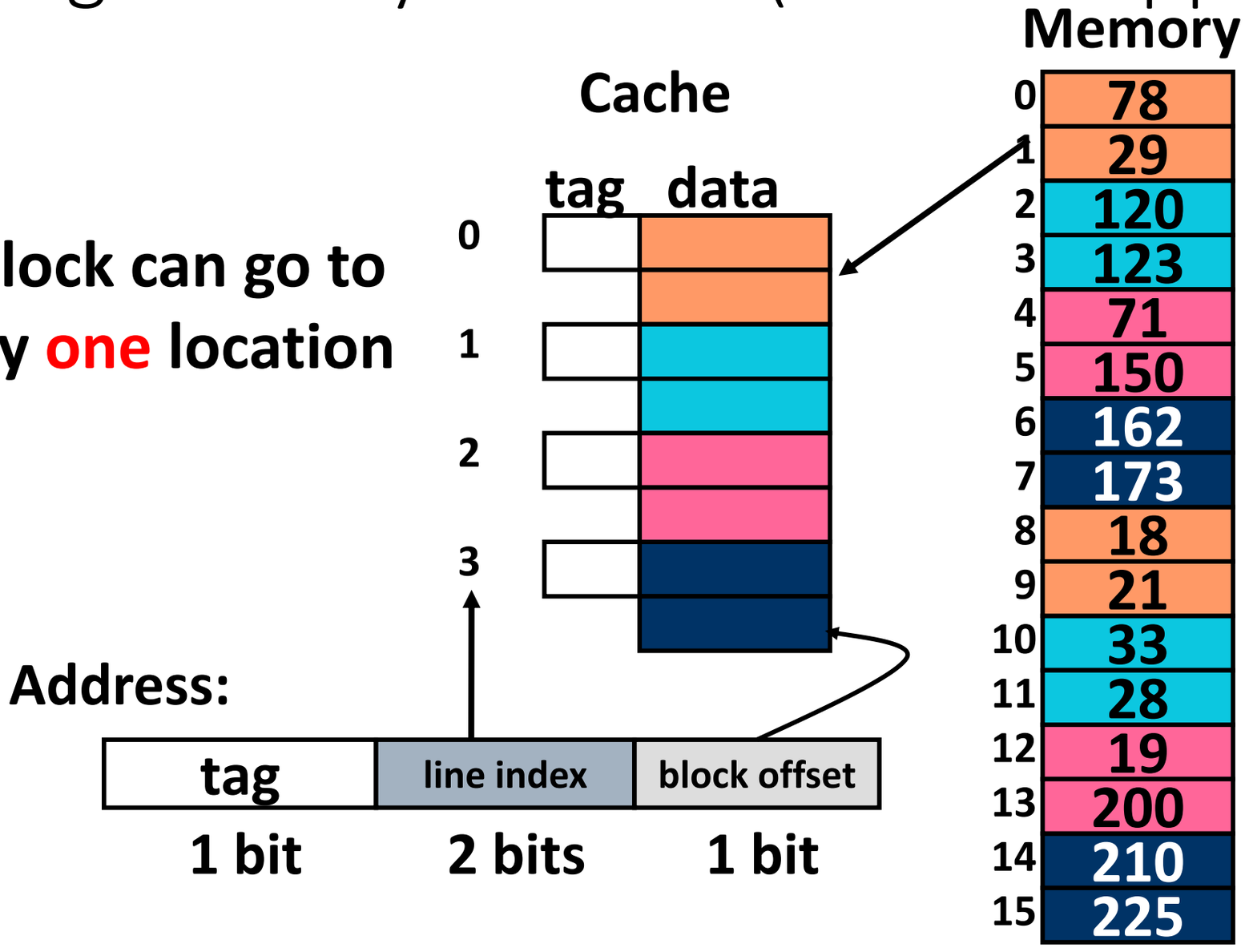
# Fully-associative caches

- We designed a fully-associative cache
  - Any memory location can be copied to any cache line.
  - We check every cache tag to determine whether the data is in the cache
- This approach can be too slow sometimes
    - Parallel tag searches are expensive and can be slow

# Direct mapped caches

- We can redesign the cache to eliminate the requirement for parallel tag lookups
  - Direct mapped caches partition memory into as many regions as there are cache lines
  - Each memory region maps to a **single cache line** in which data can be placed
  - Now only **one tag** needs to be checked – the one associated with the region the reference is in

# Mapping memory to cache (Direct-mapped)



# Direct mapped caches

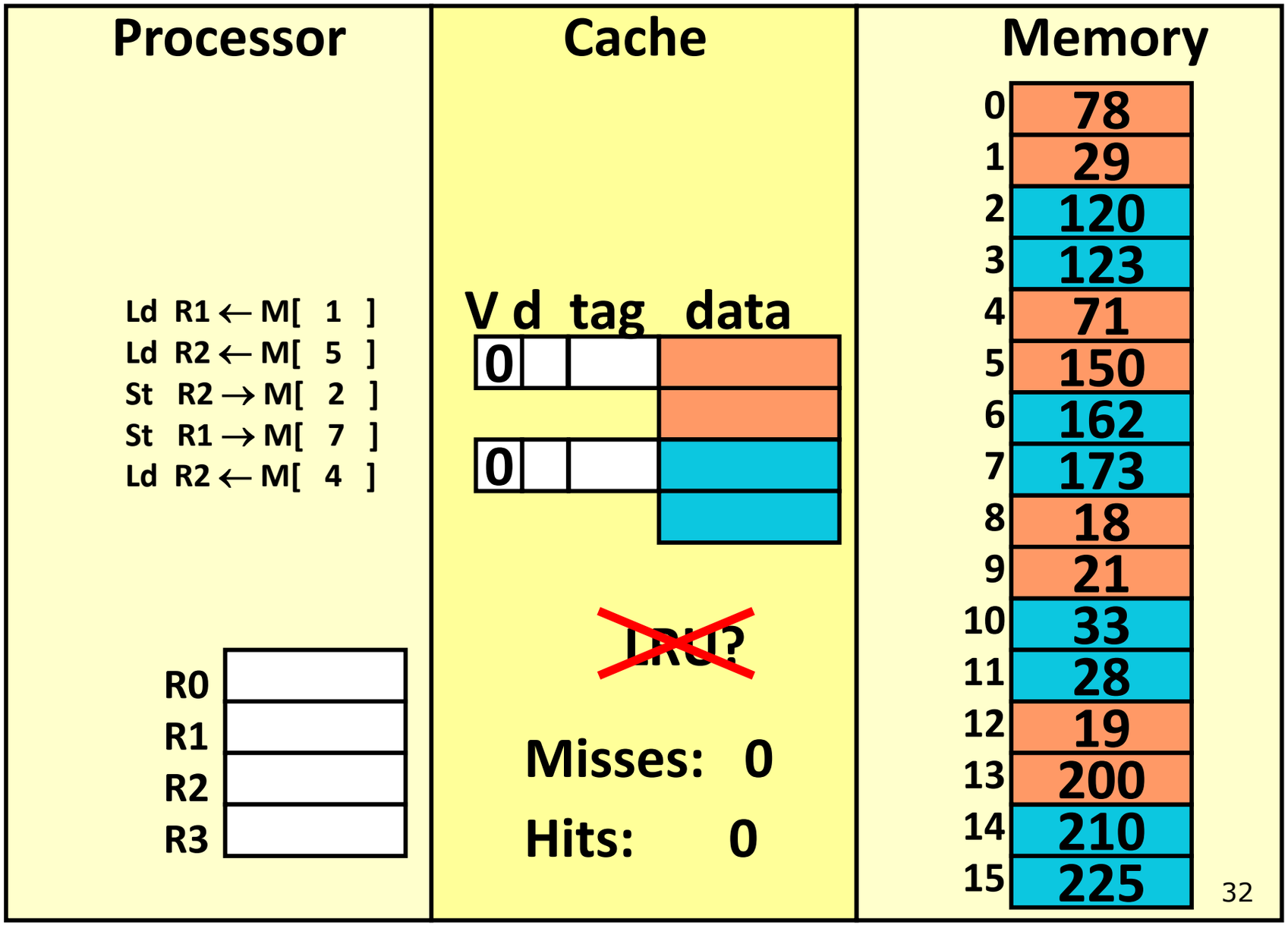
- Two blocks in memory that map to the same index in the cache cannot be present in the cache at the same time
  - One index → one entry
- Can lead to 0% hit rate if more than one block accessed in an interleaved manner map to the same index
  - Assume addresses A and B have the same index bits but different tag bits
  - A, B, A, B, A, B, A, B, …
  - All accesses are conflict misses

# Agenda

- Fully-associative vs Direct Mapped Cache
- **Example**
- Class Problems

# Direct-mapped cache

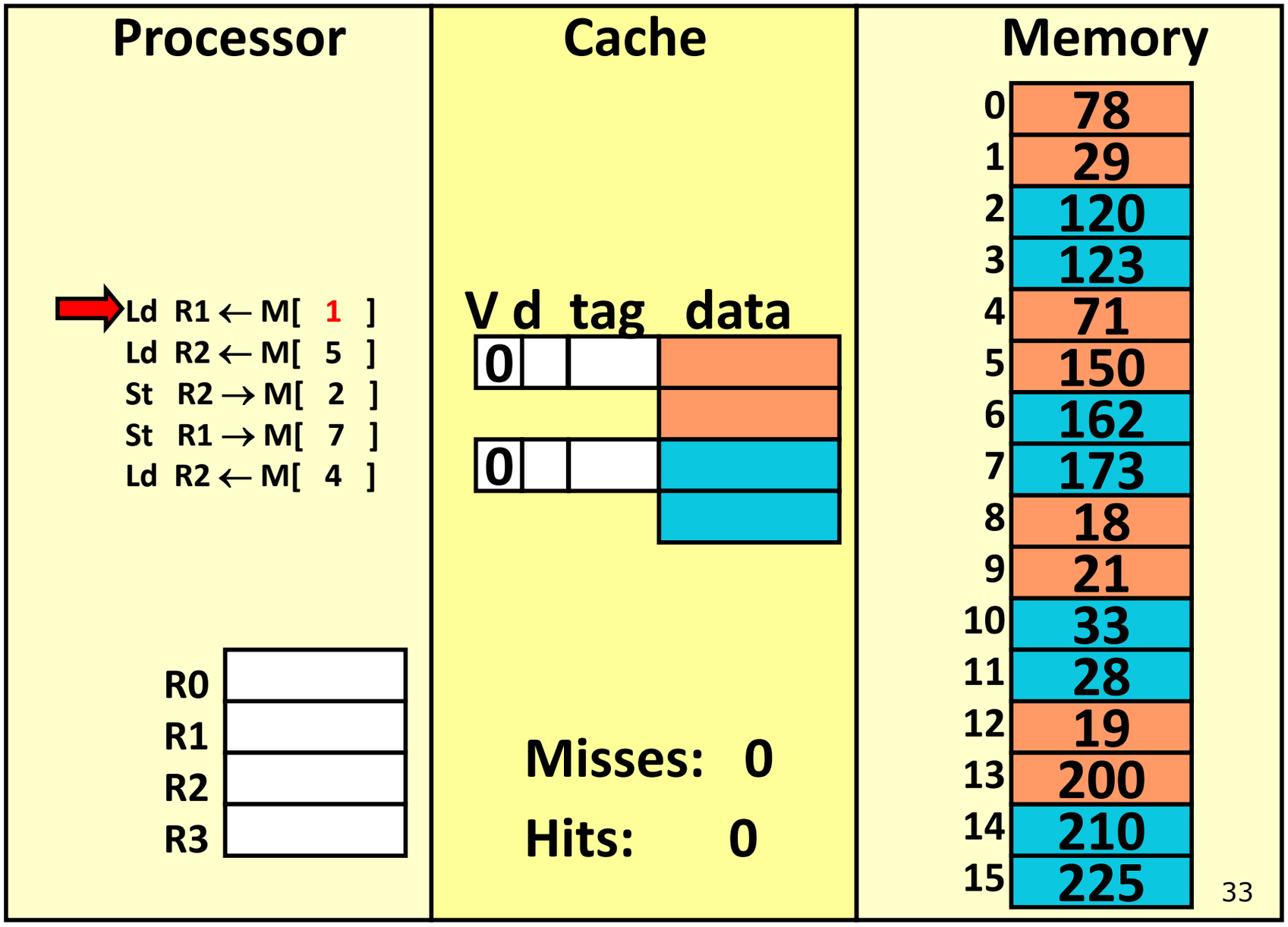
**Poll:** How many bits for each field?



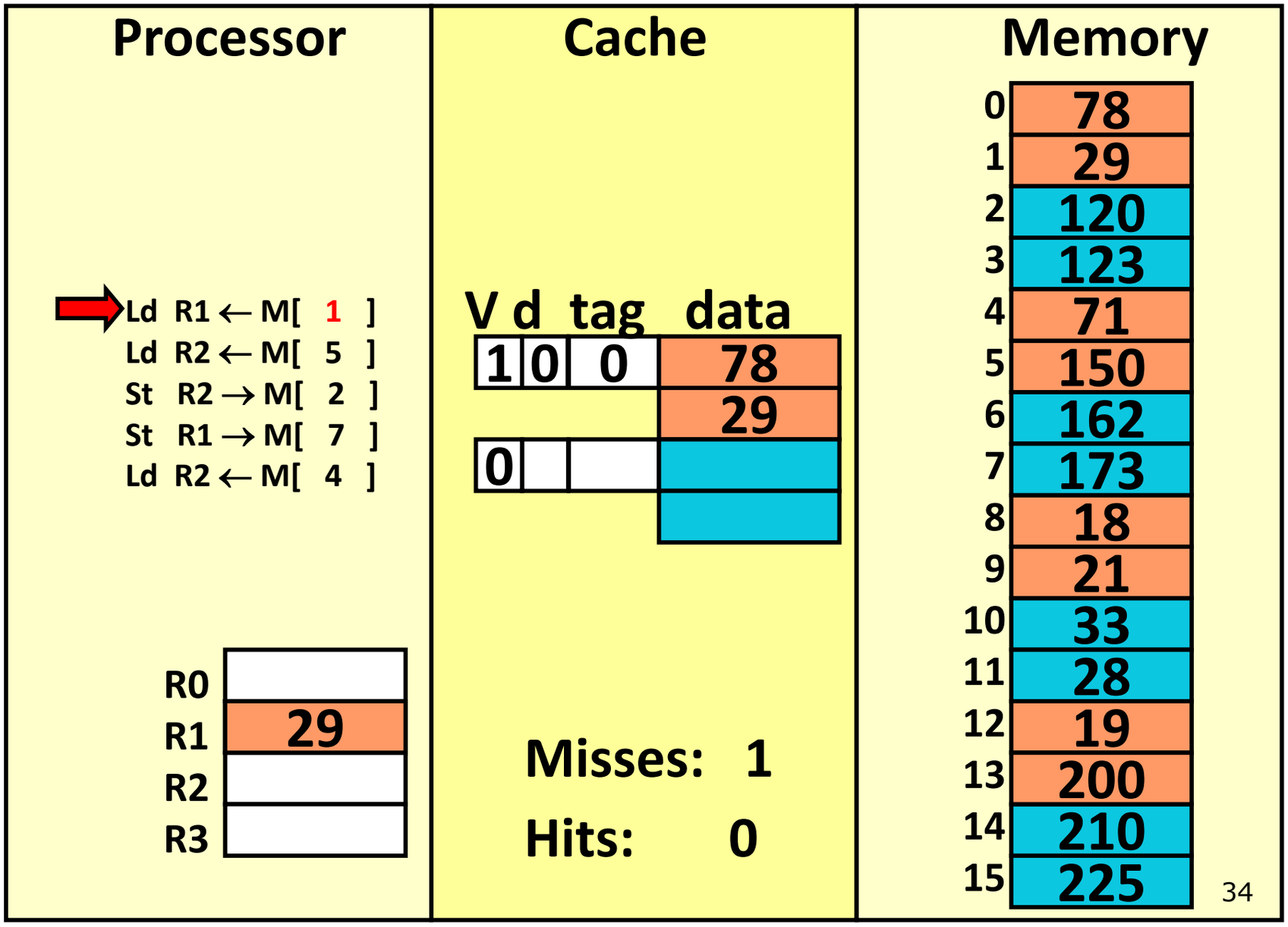



Live Poll + Q&A: slido.com #eecs370

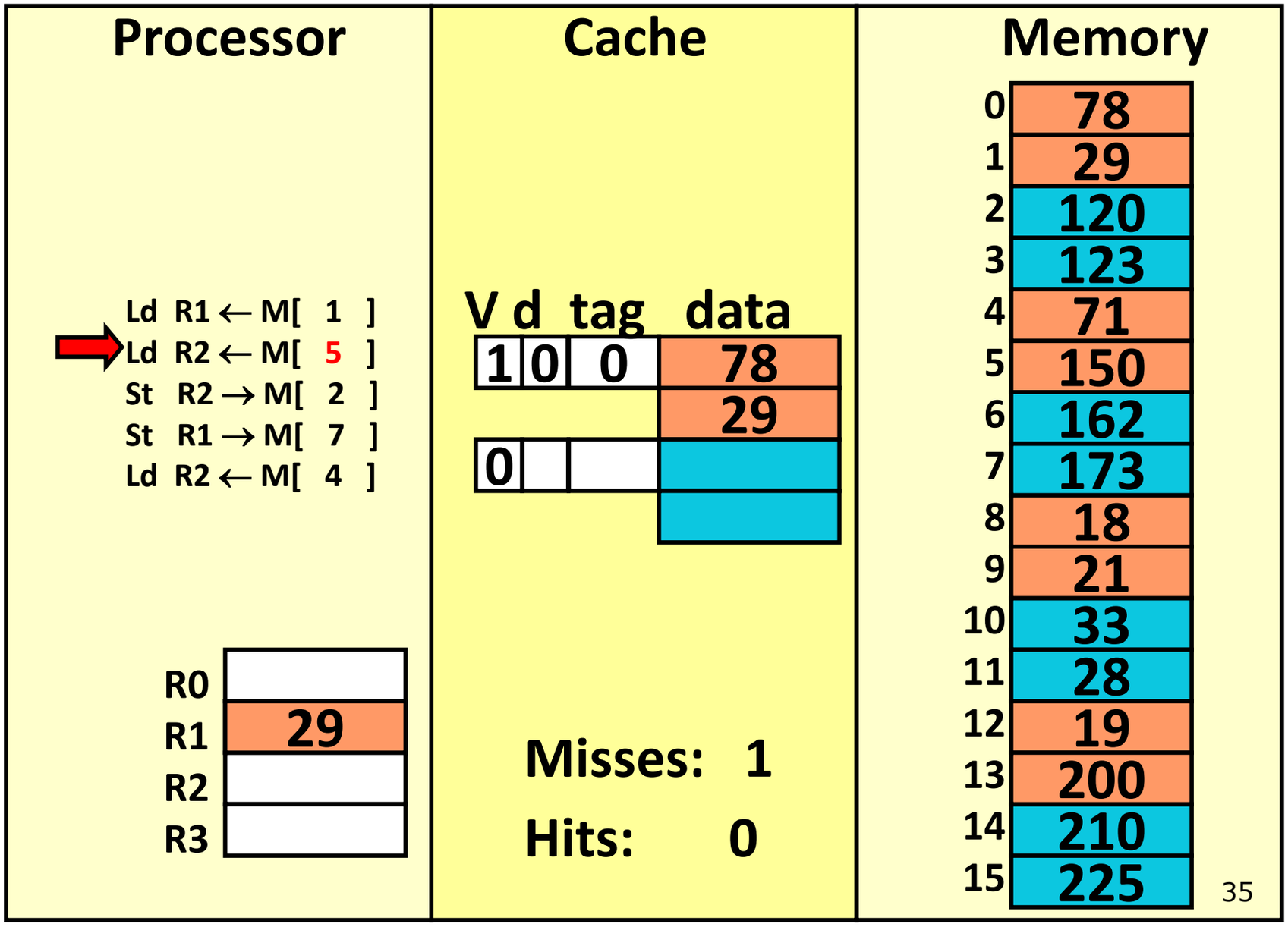
# Direct-mapped (REF 1)



# Direct-mapped (REF 1)

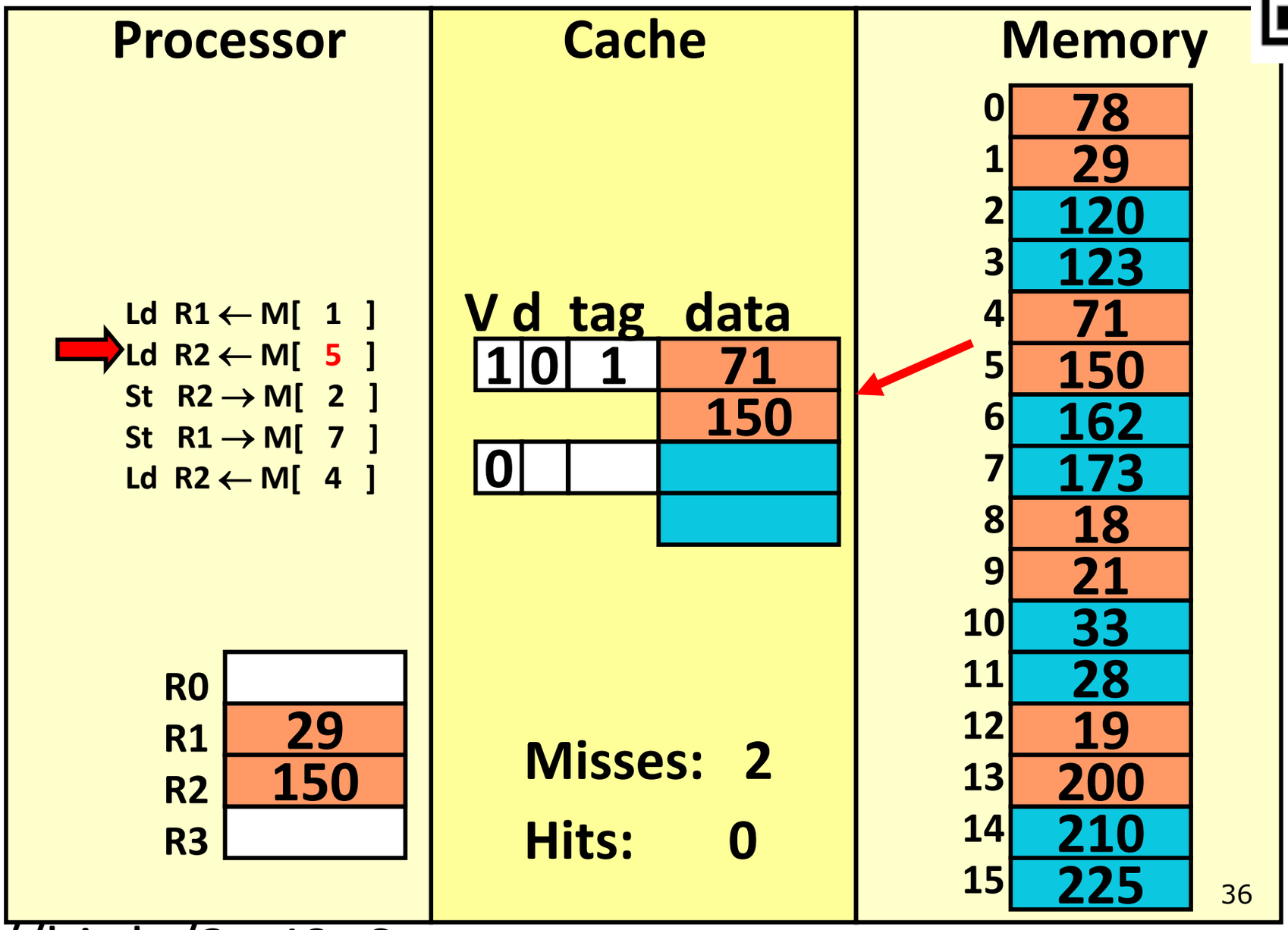


# Direct-mapped (REF 2)



# Direct-mapped (REF 2)

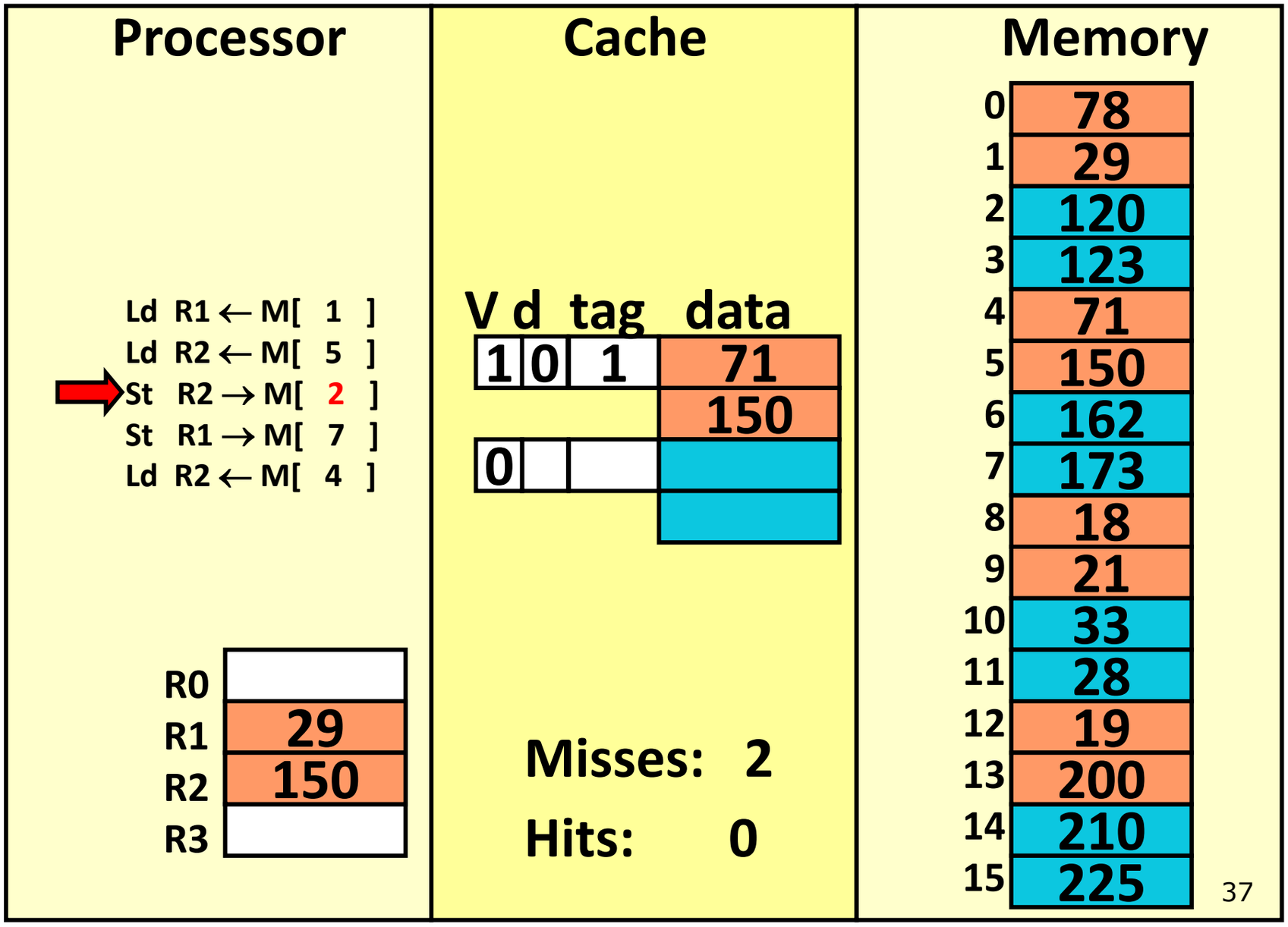
**Poll:** Complete the last few instructions yourself



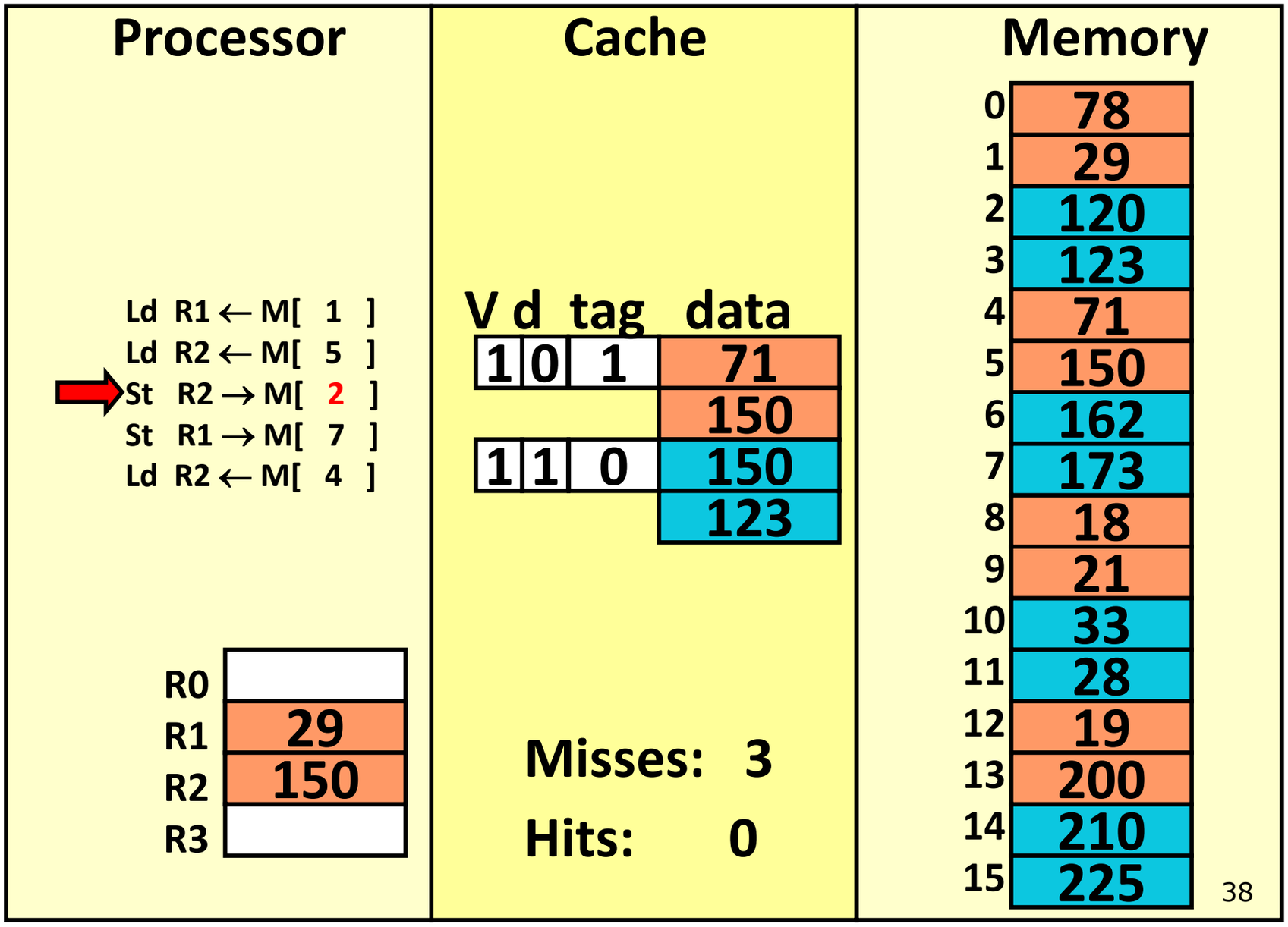



https://bit.ly/3tnI0nS

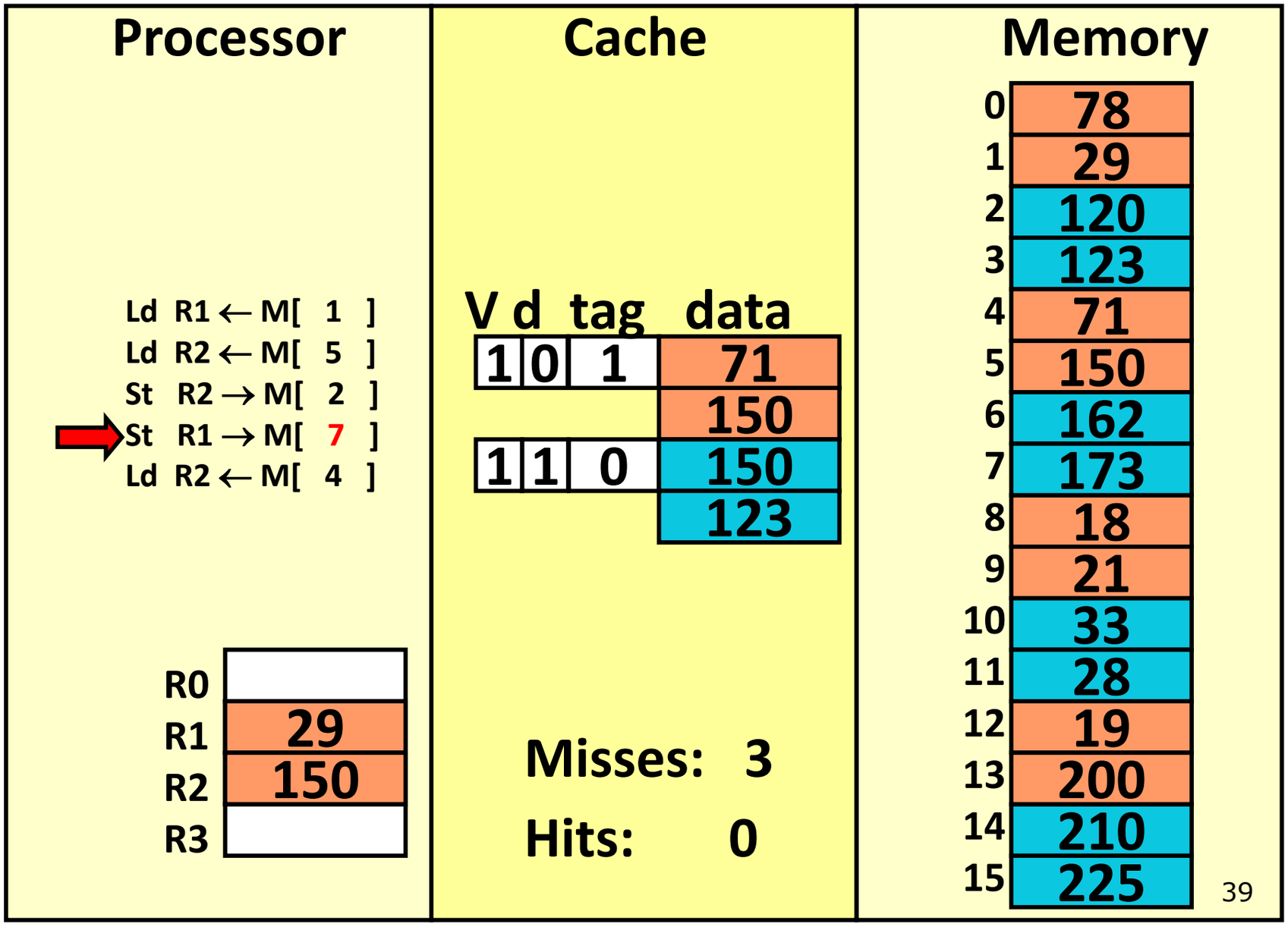
# Direct-mapped (REF 3)



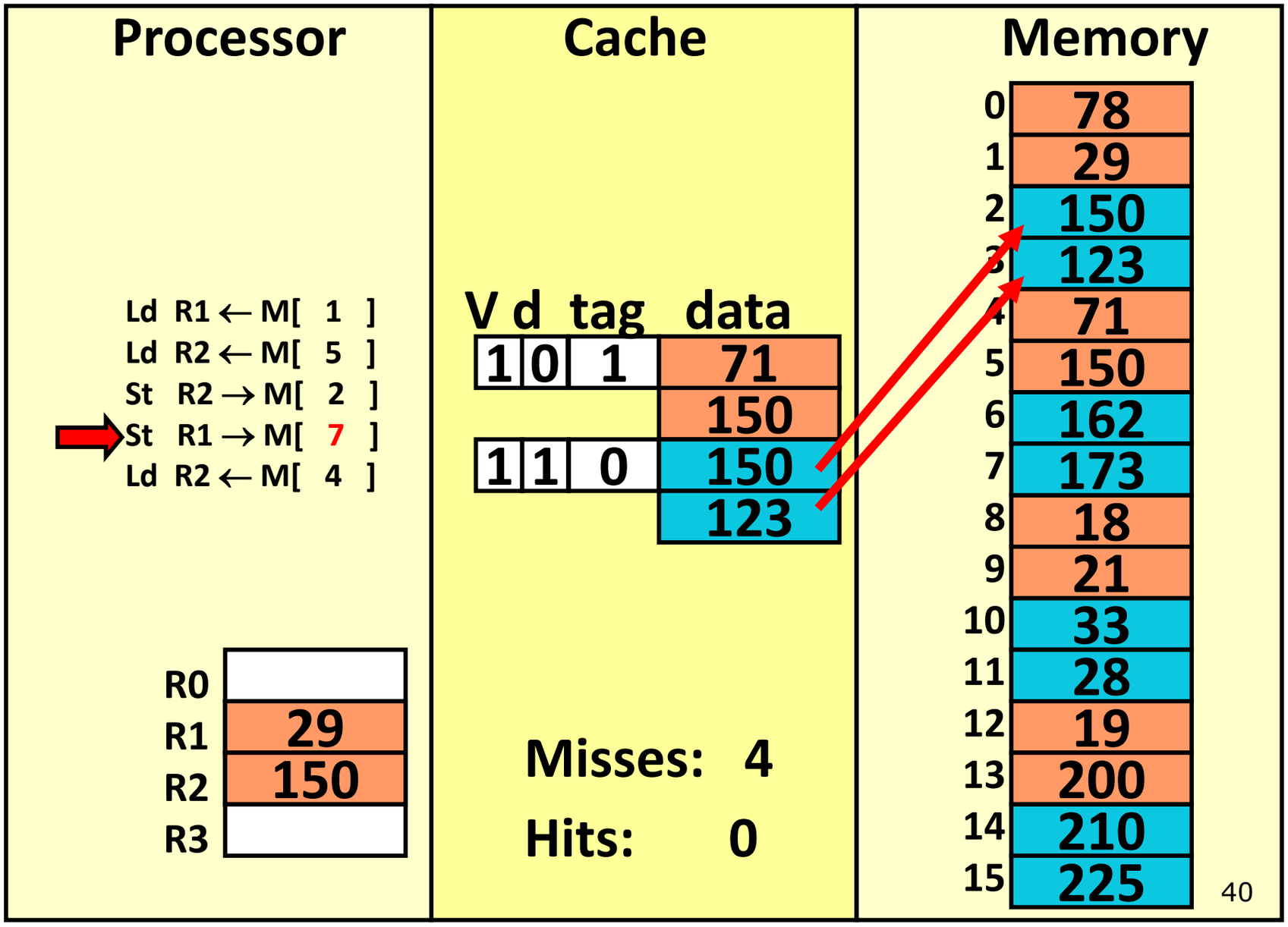
# Direct-mapped (REF 3)



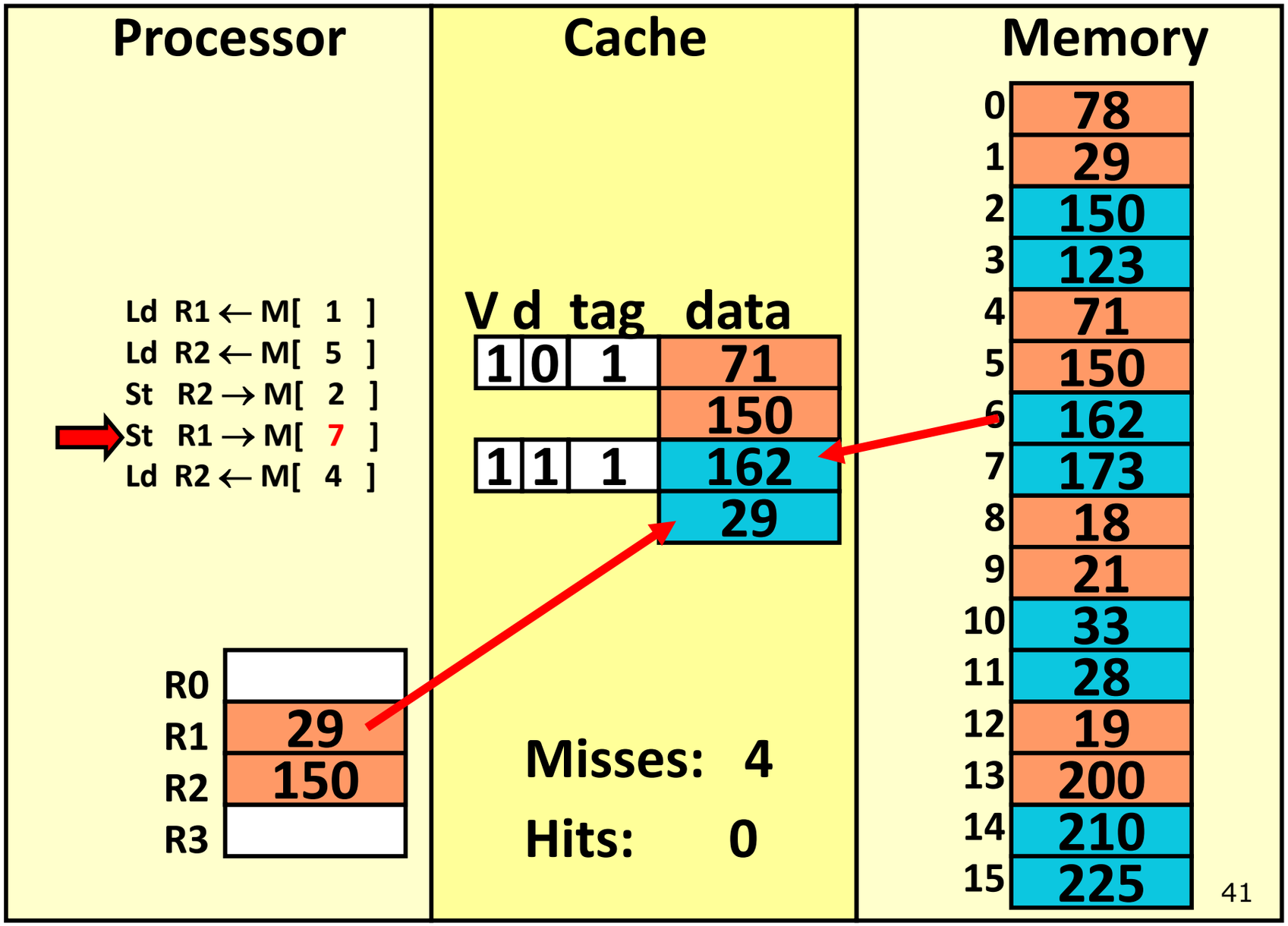
# Direct-mapped (REF 4)



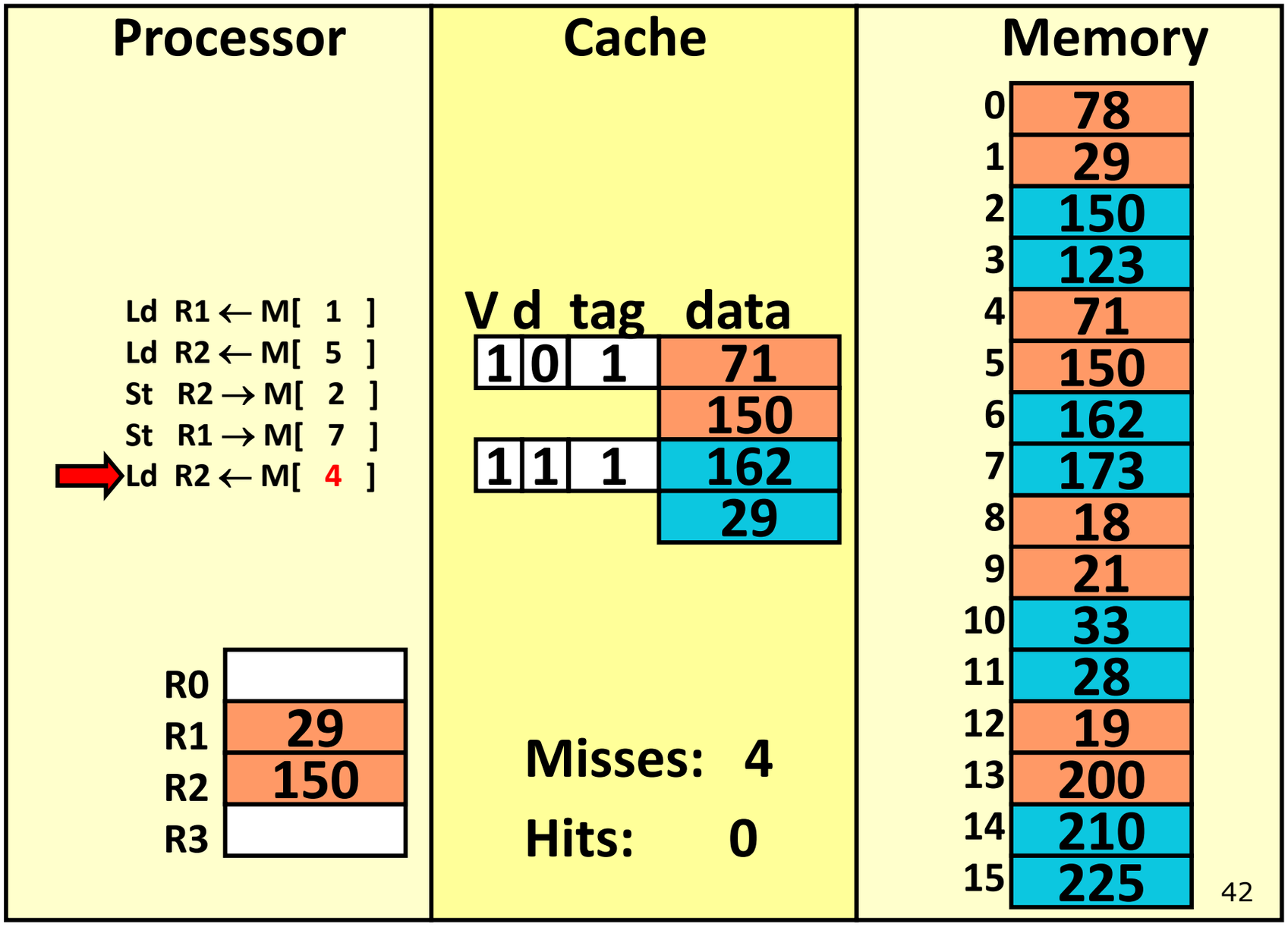
# Direct-mapped (REF 4)



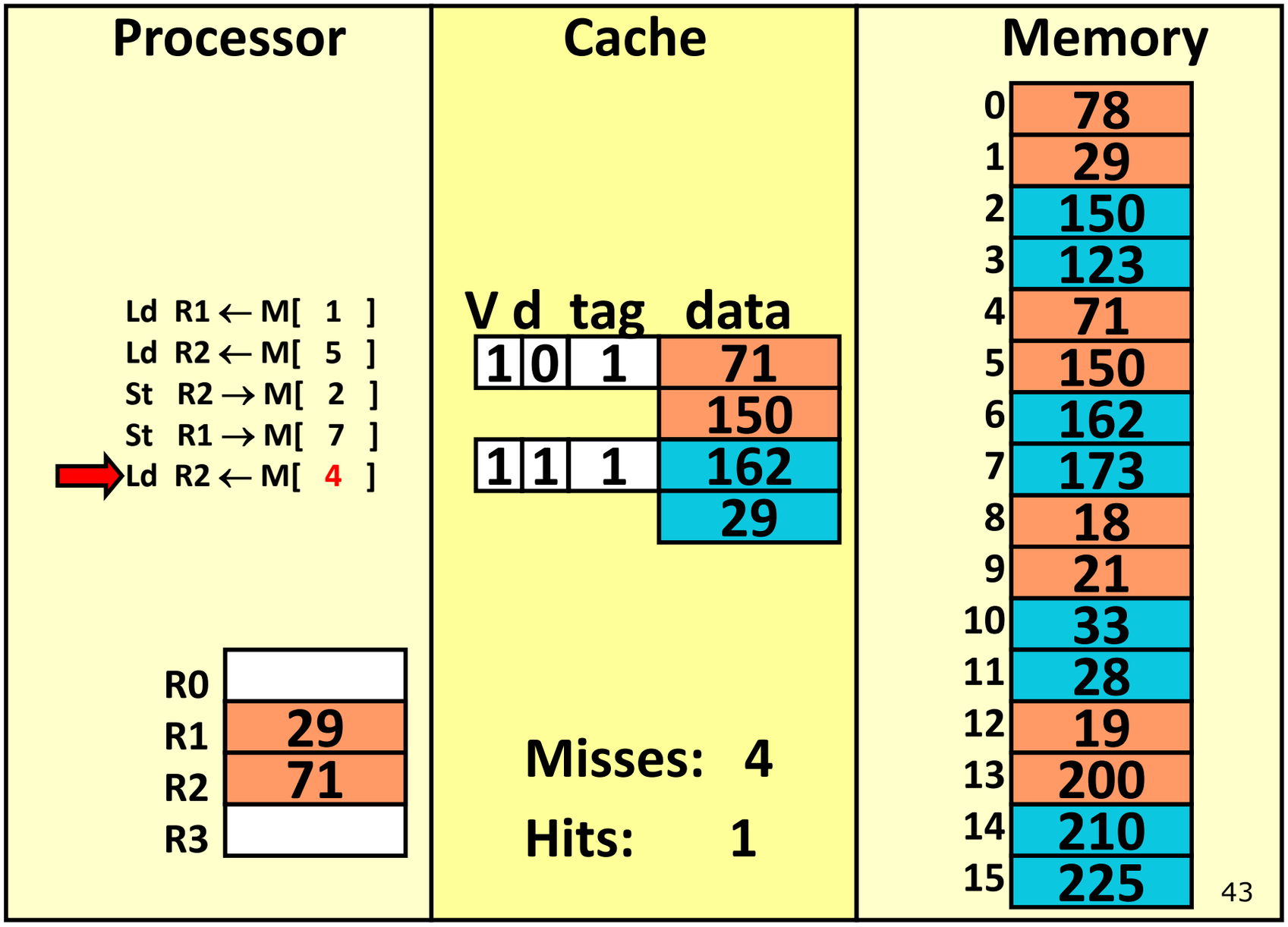
# Direct-mapped (REF 4)



# Direct-mapped (REF 5)



# Direct-mapped (REF 5)



# Next time

- Discuss intermediate between fully-associative and direct mapped caches
  - "Set Associative" caches

# Agenda

- Fully-associative vs Direct Mapped Cache
- Example
- **Class Problems**

**Poll:** **What storage is needed per block?**

# Class Problem—Storage overhead

- Consider the following cache:
  32-bit memory addresses, byte addressable, 64KB cache
  64B cache block size, write-allocate, write-back, *fully associative*

  This cache will need 512 kilobits for the data area (64 kilobytes times 8 bits per byte). Note that in this context, 1 kilobyte = 1024 bytes (NOT 1000 bytes!) Besides the actual cached data, this cache will need other storage. Consider tags, valid bits, dirty bits, bits to keep track of LRU, and anything else that you think is necessary.
- How many additional bits (not counting the data) will be needed to implement this cache ?

**Poll:** What storage is needed per block?

# Class Problem—Storage overhead

- Consider the following cache:
  32-bit memory addresses, byte addressable, 64KB cache
  64B cache block size, write-allocate, write-back, *fully associative*

  This cache will need 512 kilobits for the data area (64 kilobytes times 8 bits per byte). Note that in this context, 1 kilobyte = 1024 bytes (NOT 1000 bytes!) Besides the actual cached data, this cache will need other storage. Consider tags, valid bits, dirty bits, bits to keep track of LRU, and anything else that you think is necessary.
- How many additional bits (not counting the data) will be needed to implement this cache ?

**Tag bits = 32 – log(64) = 26 bits**

**#lines = 64KB/64B = 1024**

**LRU = log(1024) = 10 bits**

**1 valid bit, 1 dirty bit**

Poll

# Class Problem—Analyze performance

- Suppose that accessing a cache takes 10ns while accessing main memory in case of cache-miss takes 100ns. What is the average memory access time if the cache hit rate is 97%?

- To improve performance, the cache size is increased. It is determined that this will increase the hit rate by 1%, but it will also increase the time for accessing the cache by 2ns. Will this improve the overall average memory access time?

# Class Problem—Analyze performance

- Suppose that accessing a cache takes 10ns while accessing main memory in case of cache-miss takes 100ns. What is the average memory access time if the cache hit rate is 97%?

  AMAT = 10 + (1 - 0.97)*100 = 13 ns

- To improve performance, the cache size is increased. It is determined that this will increase the hit rate by 1%, but it will also increase the time for accessing the cache by 2ns. Will this improve the overall average memory access time?

  AMAT = 12 + (1 - 0.98)*100 = 14 ns