19. Cache Organization: Direct-Mapped & Intro to Set-Associative Caches

EECS 370 – Introduction to Computer Organization – Winter 2023

EECS Department
University of Michigan in Ann Arbor, USA

Stuff that needs doing

- Project 3 is due this Thursday
- Homework 5 is posted

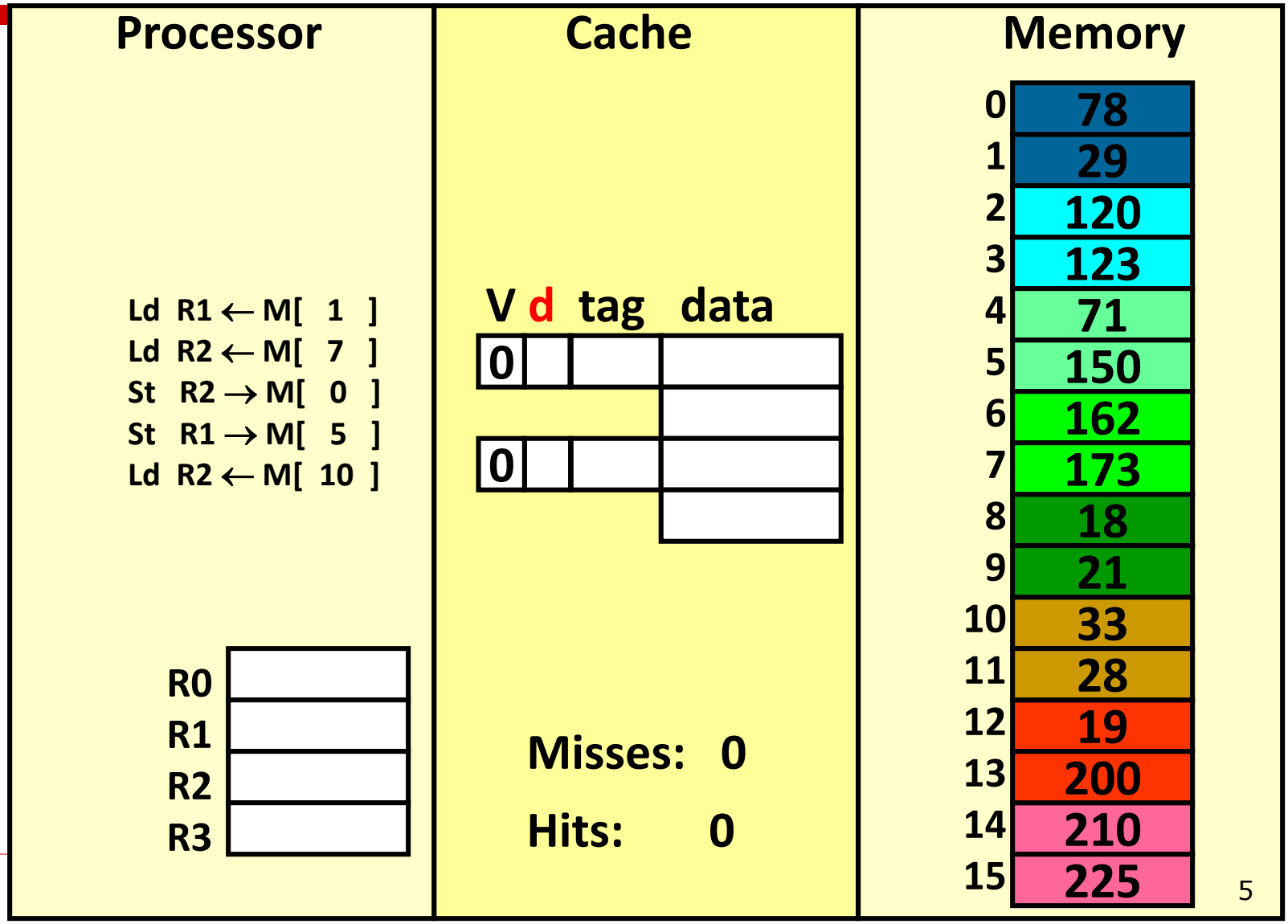
Today

- Finish up writing to a cache
  - Write-back caches
  - Review Write Options
- Some review questions
- Direct-mapped caches

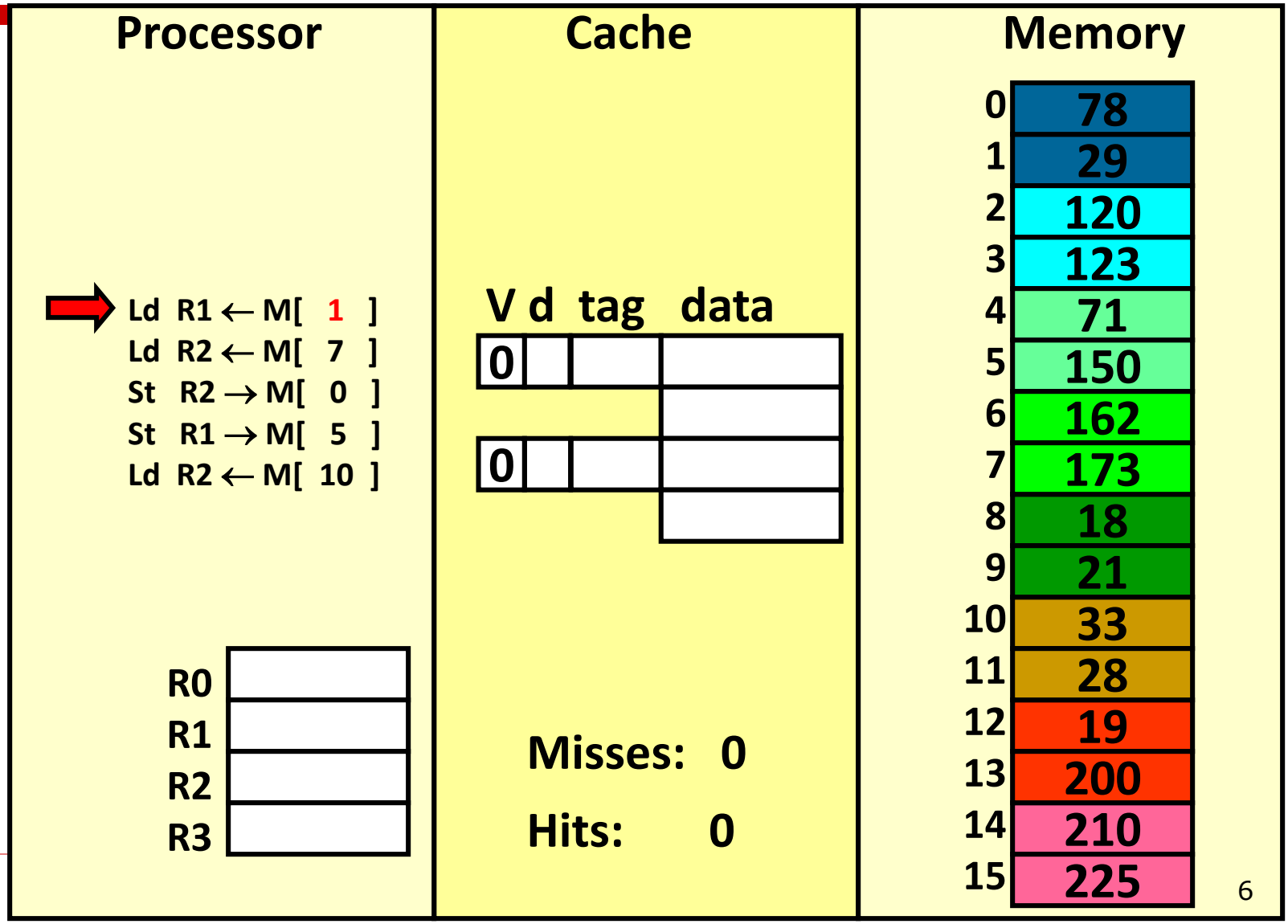
Write-through vs. write-back

- Can we also design the cache to NOT write all stores to memory immediately?
  - We can keep the most recent copy in the cache and update the memory **only when** that data is evicted from the cache (a write-back policy also called copy-back).
  - Do we need to write-back all evicted lines?
    - No, only blocks that have been stored into
    - 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.

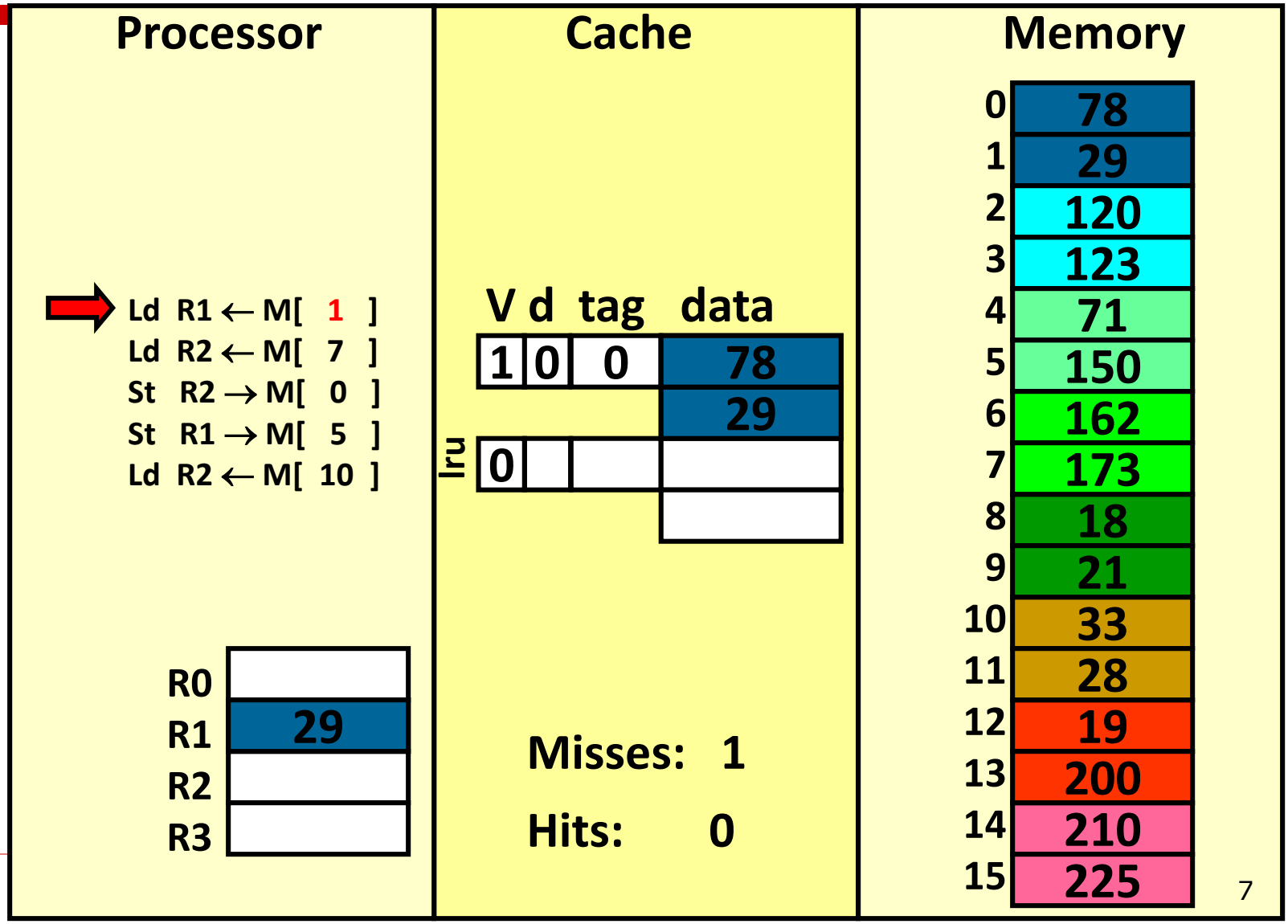
Handling stores (write-back)



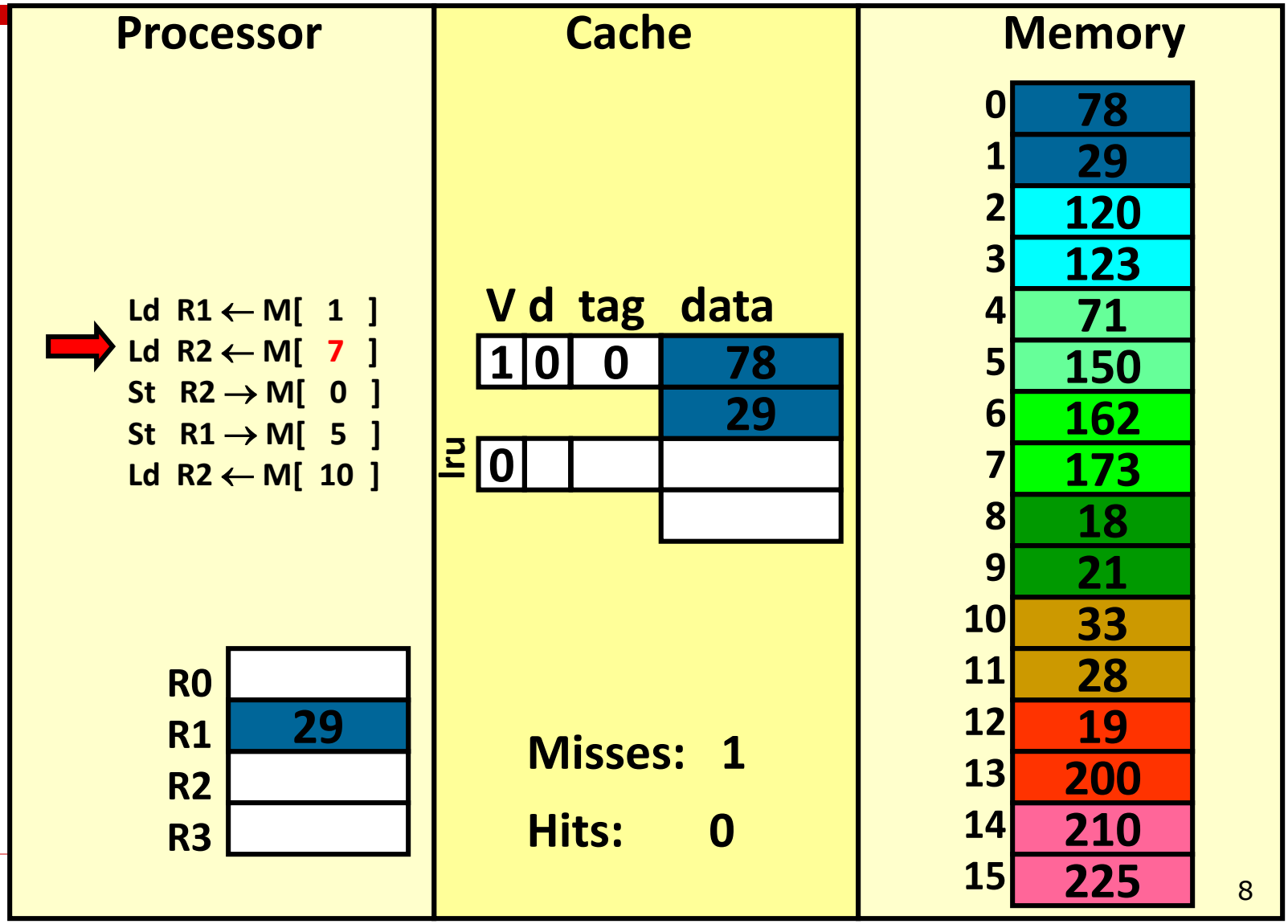
write-back (REF 1)



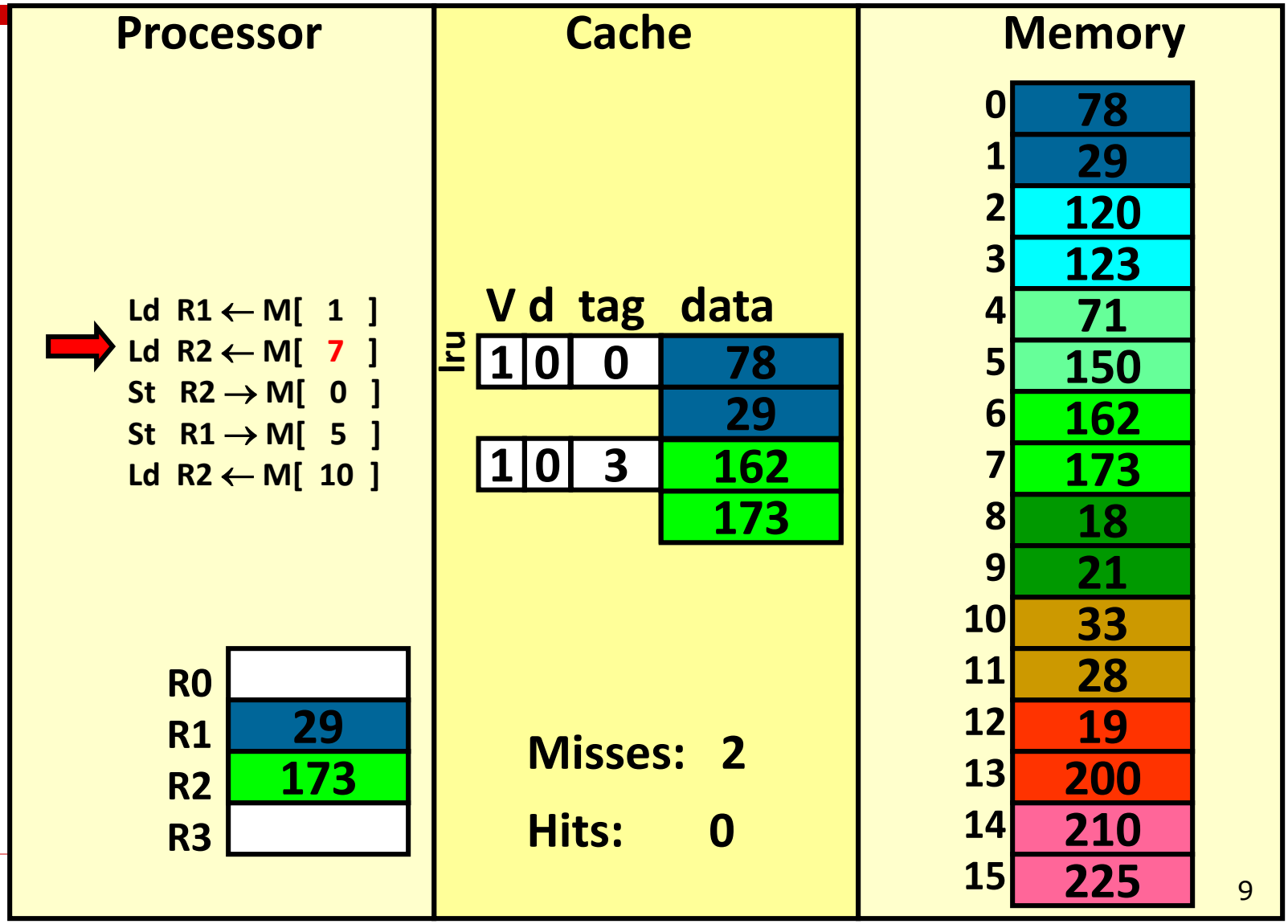
write-back (REF 1)



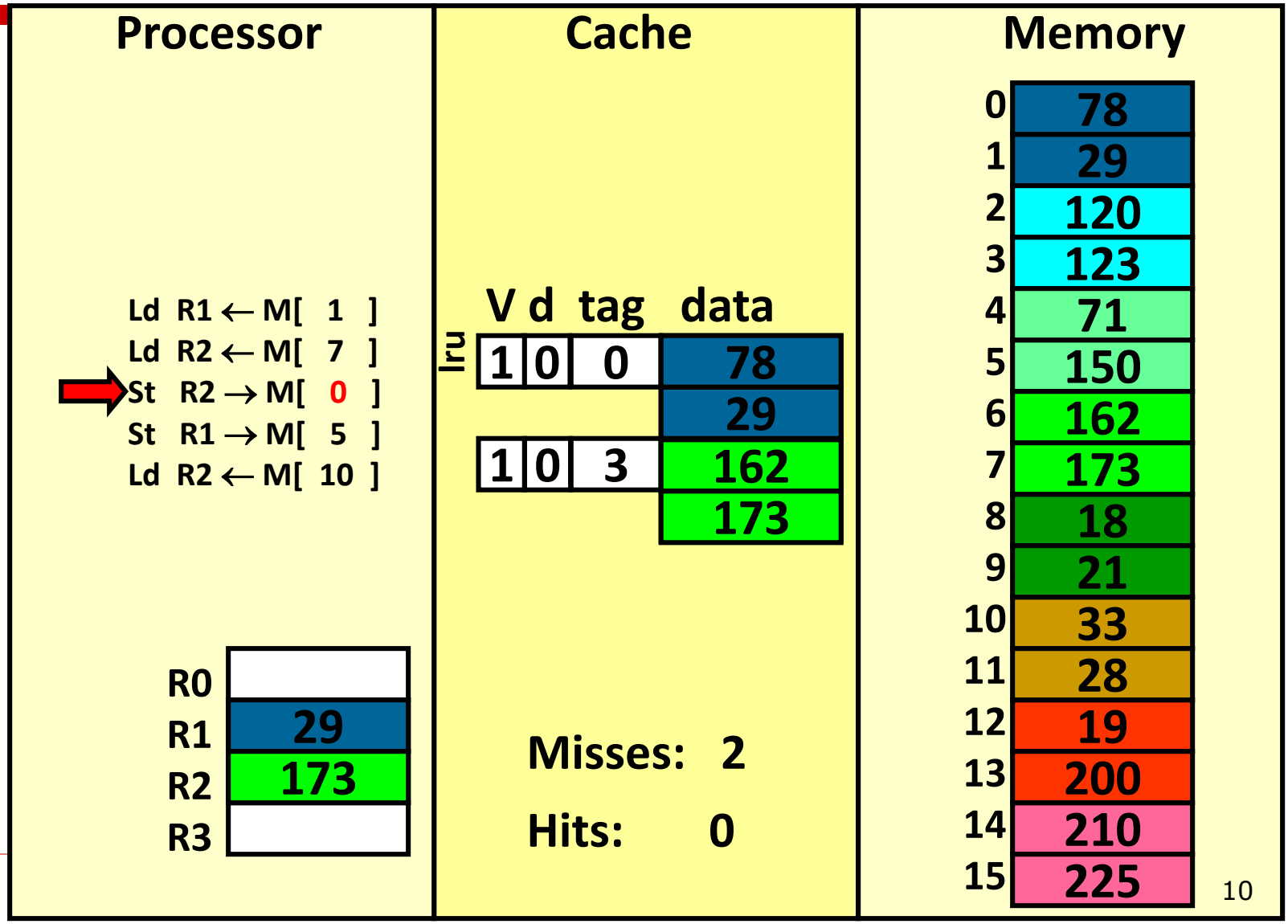
write-back (REF 2)



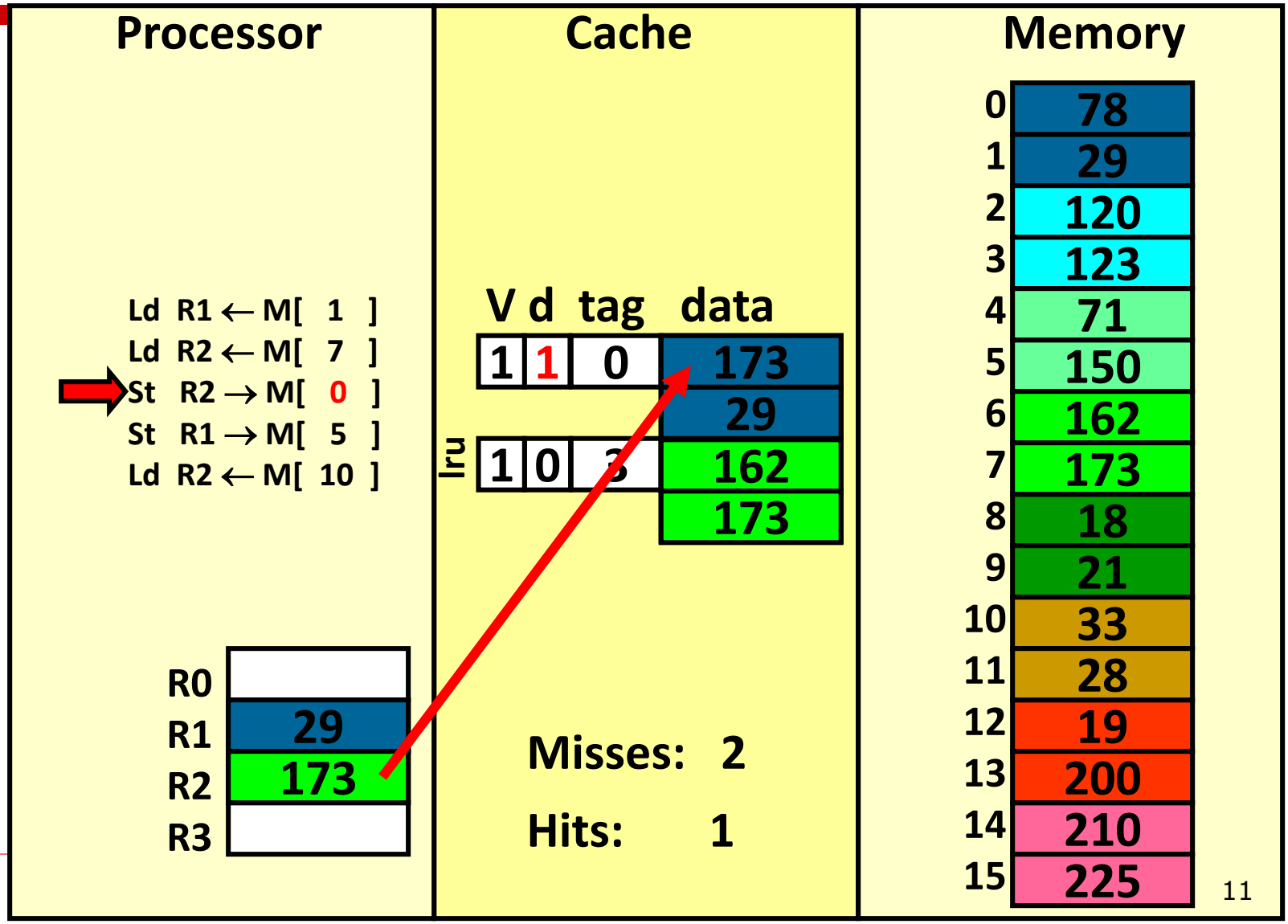
write-back (REF 2)



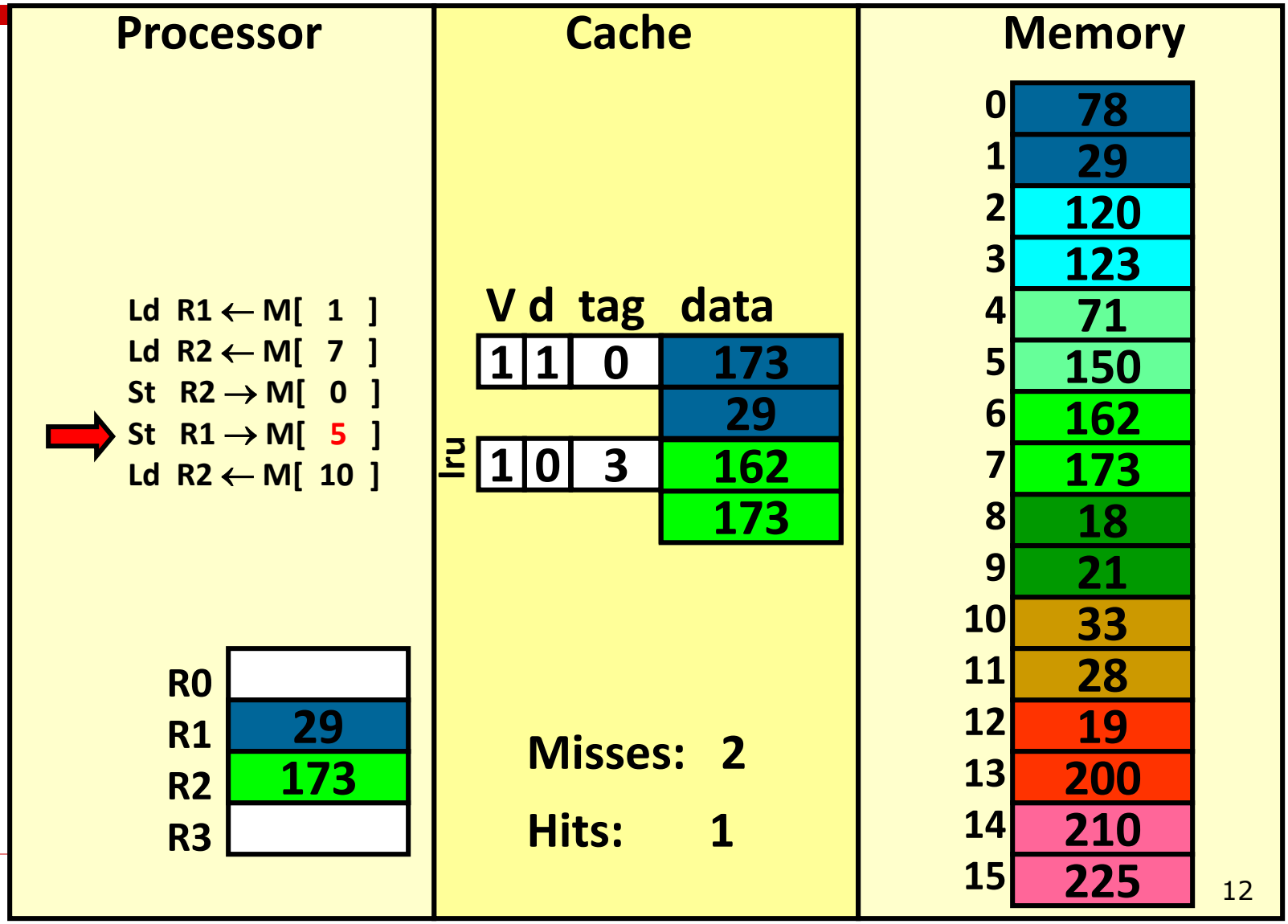
write-back (REF 3)



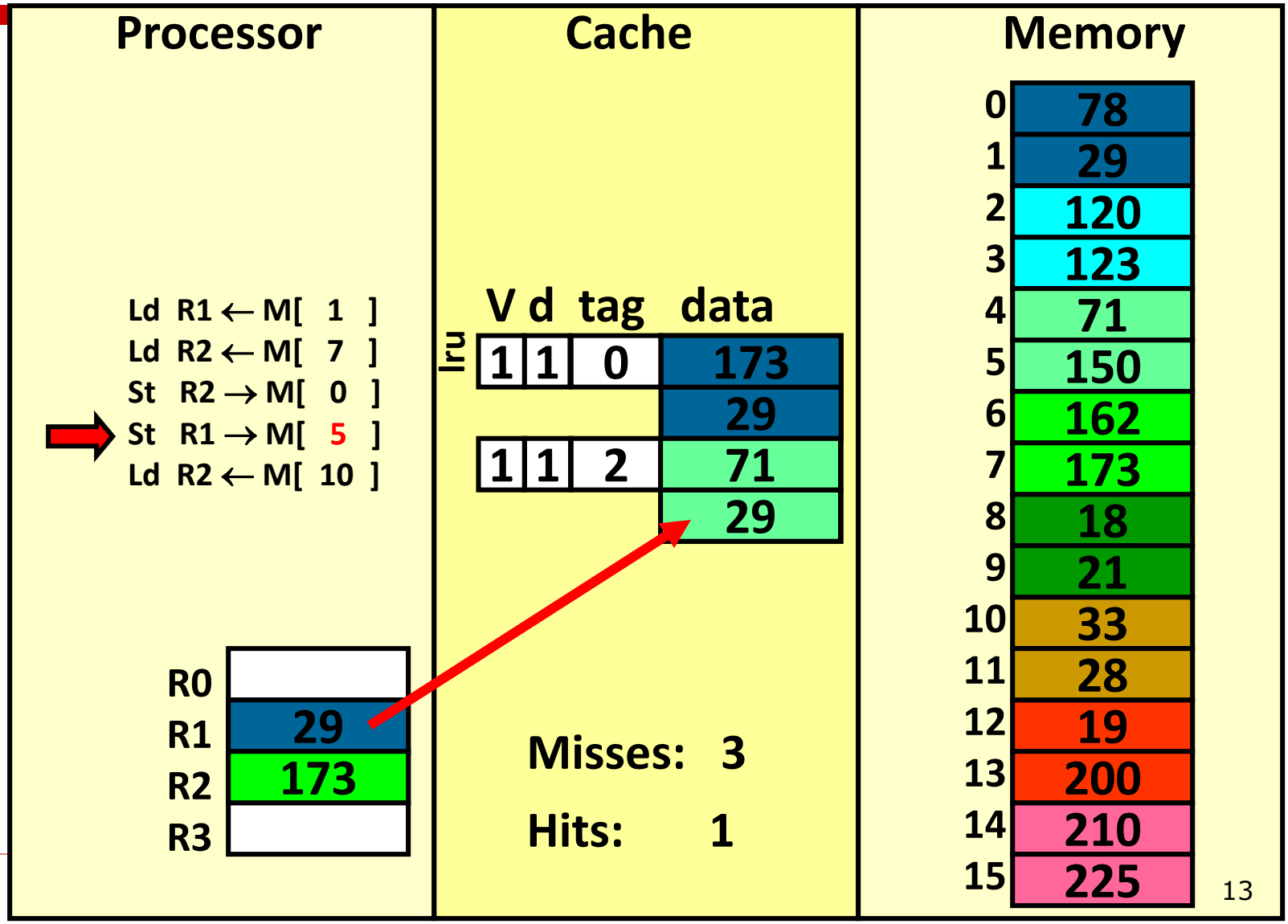
write-back (REF 3)



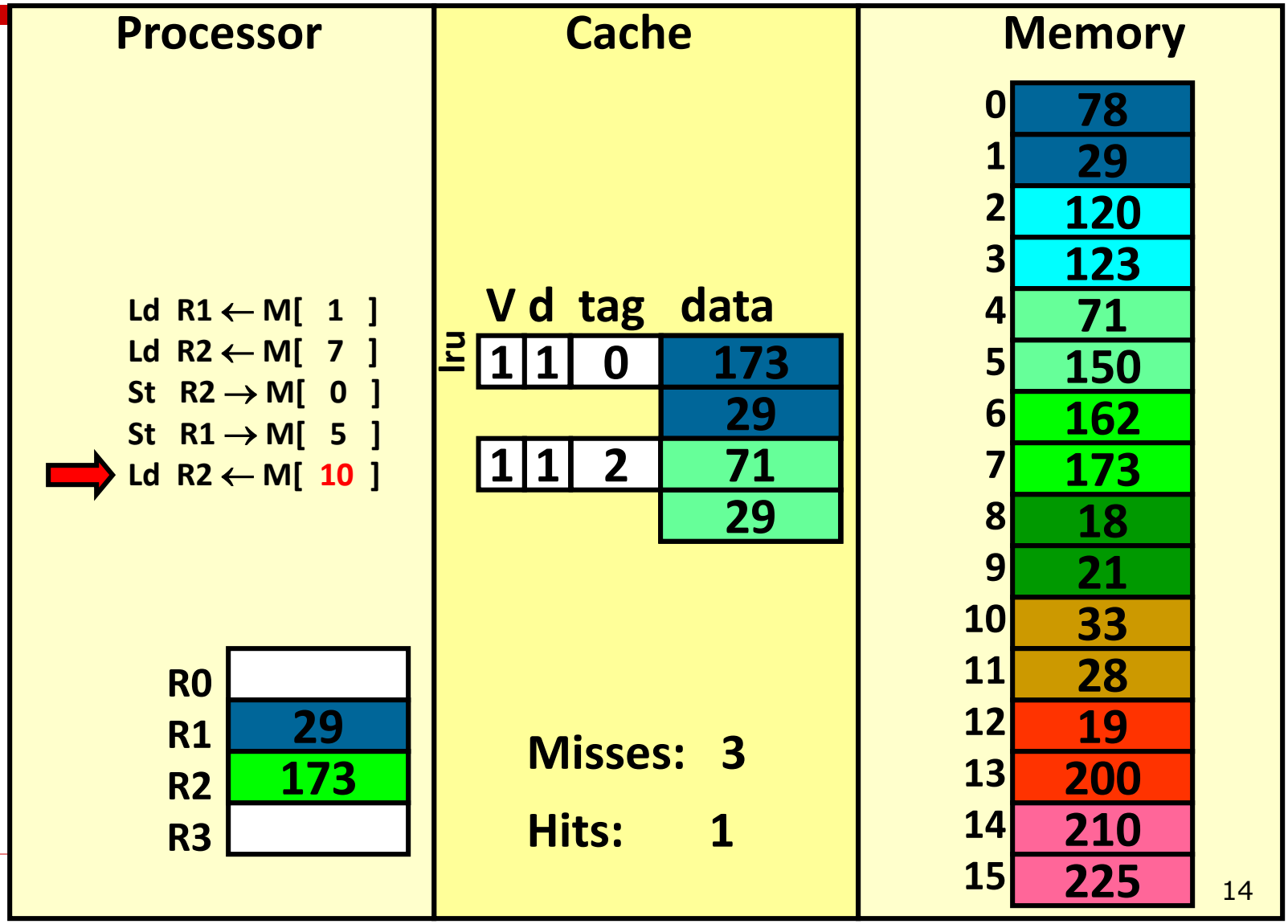
write-back (REF 4)



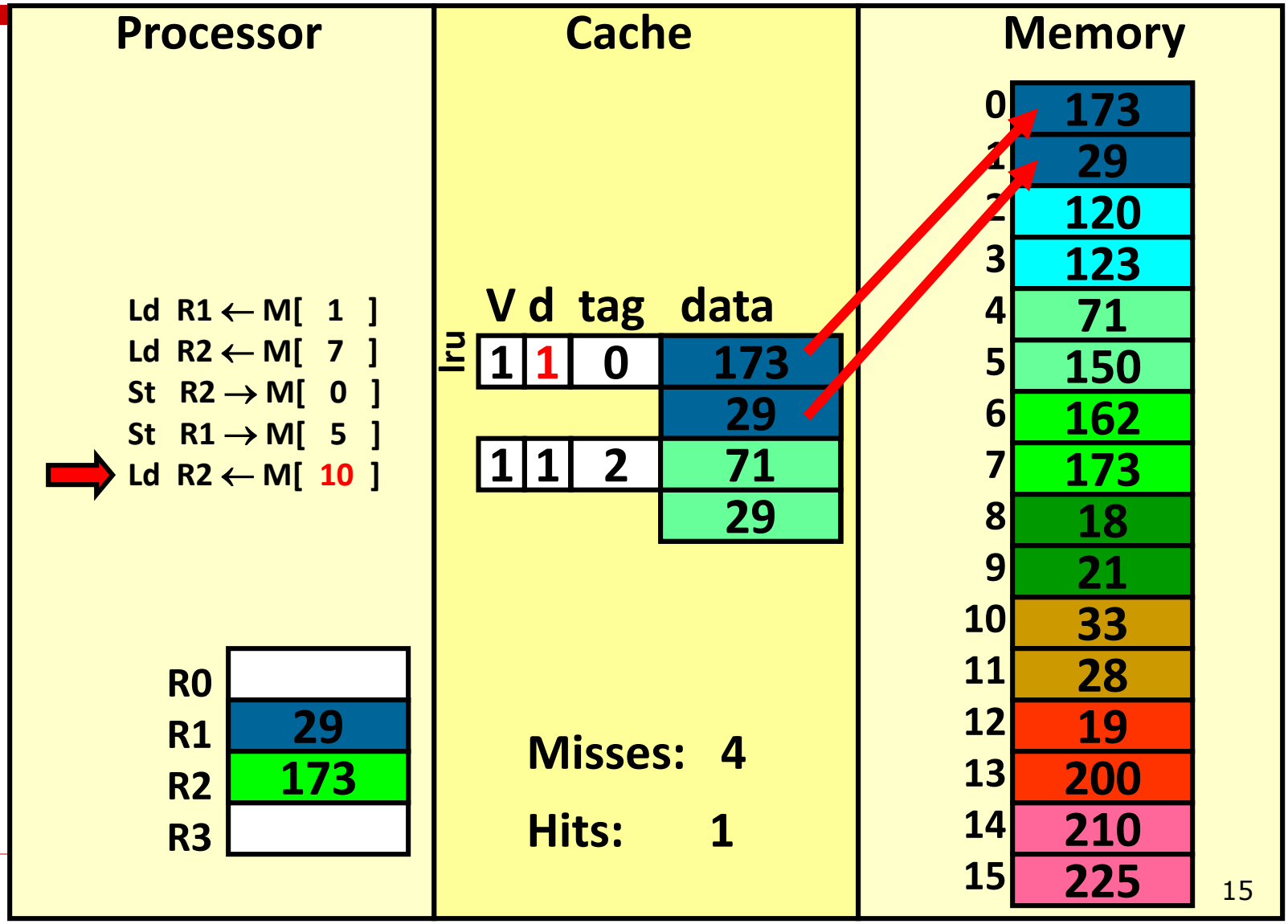
write-back (REF 4)



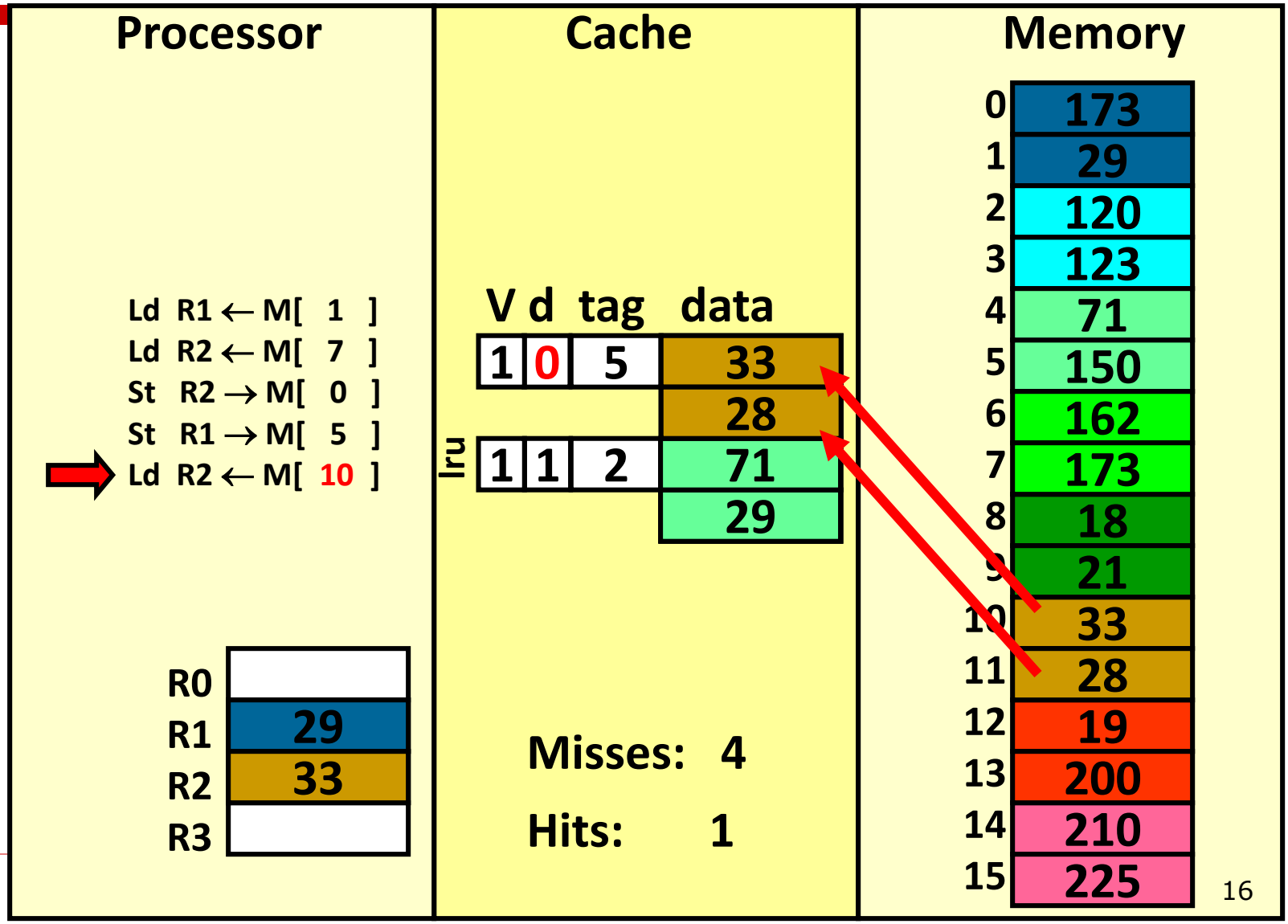
write-back (REF 5)



write-back (REF 5)



write-back (REF 5)



Review: Writes

- Write-allocate vs. no-write-allocate caches
- Policy that decides what to do with a cache-miss on a store instruction.
  - Write-allocate: First bring data from memory into the cache, then write
  - No-write-allocate: do not bring data in the cache, just write directly to the memory, not to the cache

Review: Writes

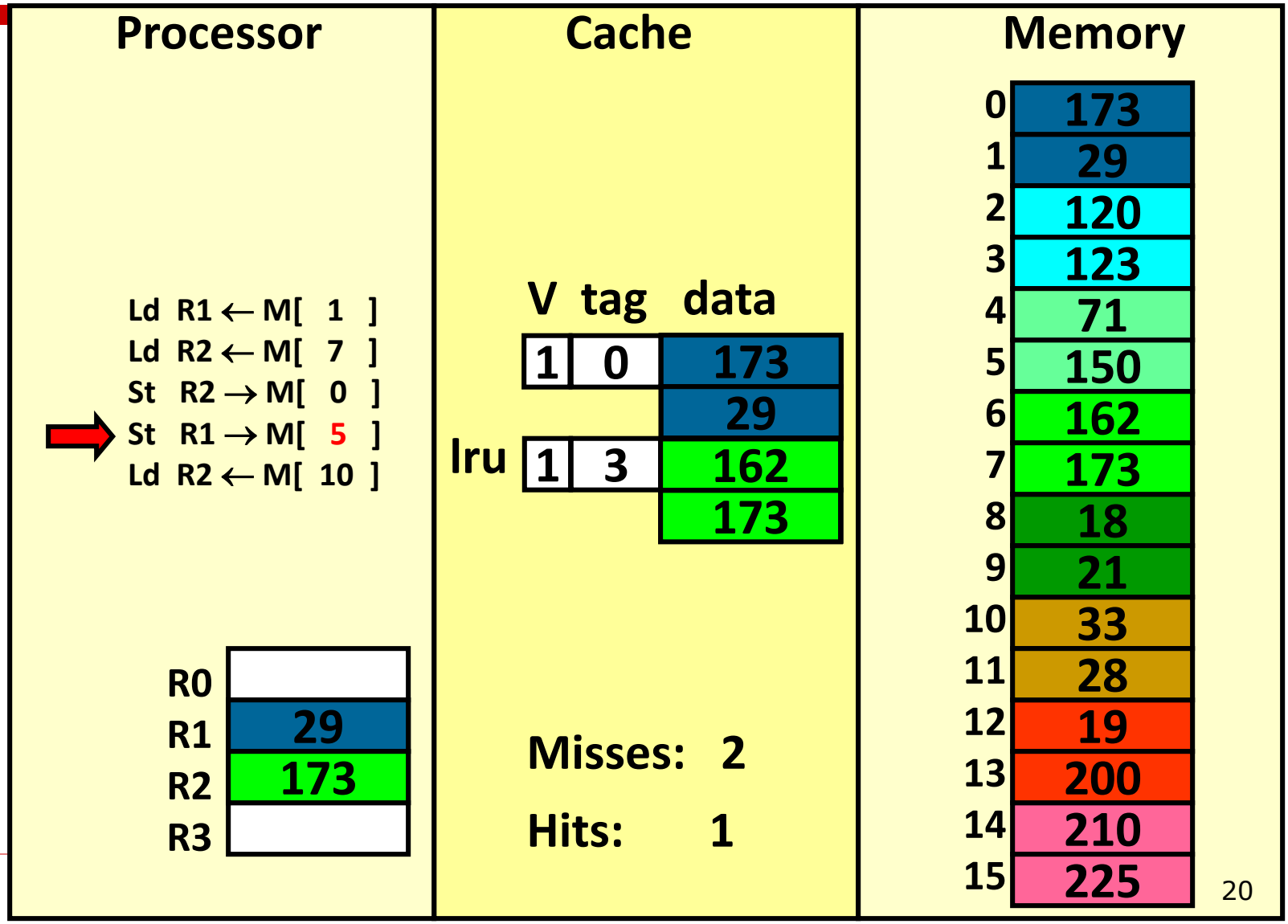
- Write-through vs. write-back caches
  - Policy that decides when to **write to cache vs. memory vs. both**
  - Write-through: write to both cache and memory
  - Write-back: write only to cache, keep track of dirty cache line, write to memory when dirty cache line is evicted

Writes

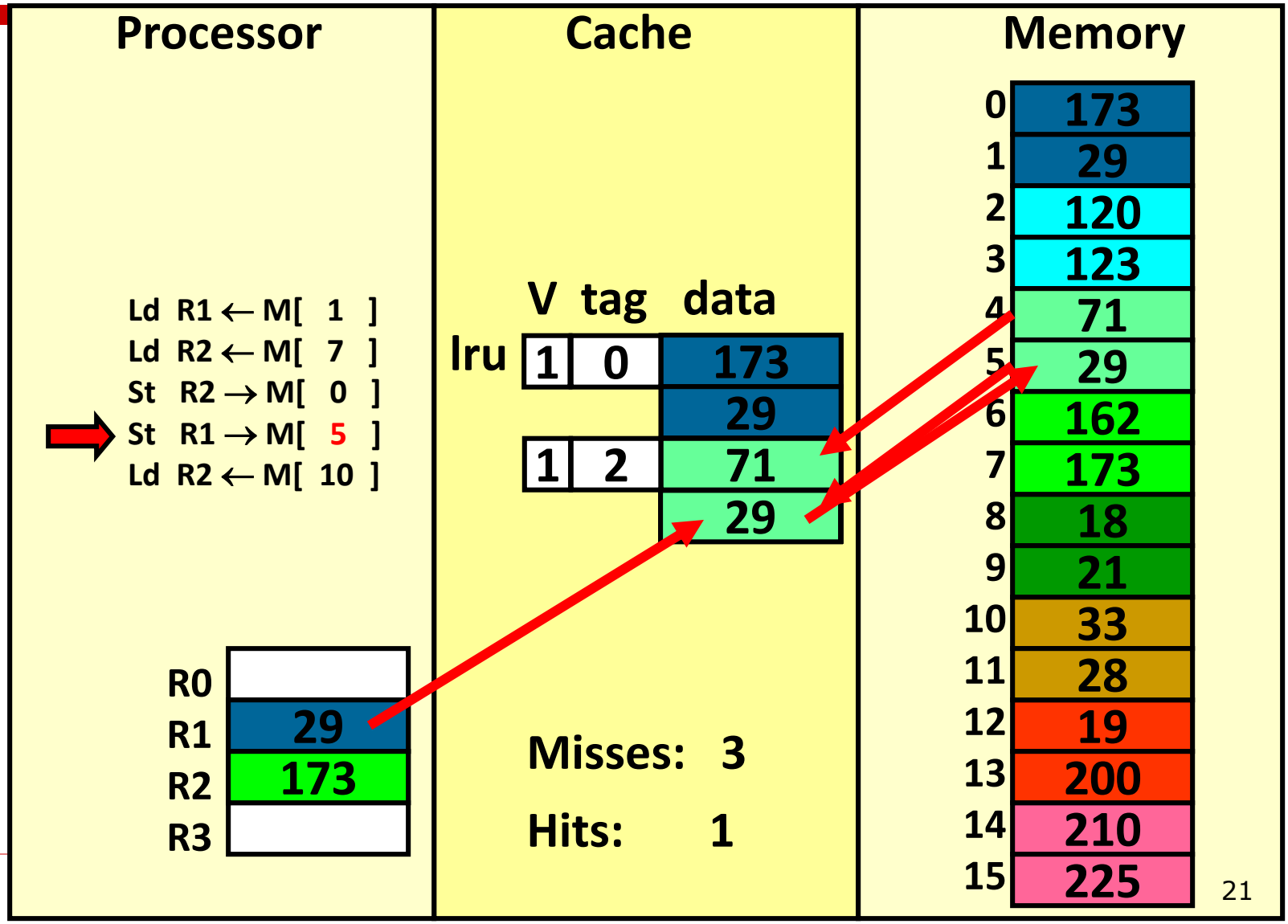
Store w No Allocate	Write-Back	Write-Through
Hit?	Write Cache	Write to Cache **+ Memory**
Miss?	Write to Memory	Write to Memory
Replace block?	If evicted block is dirty, write to Memory	Do Nothing

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

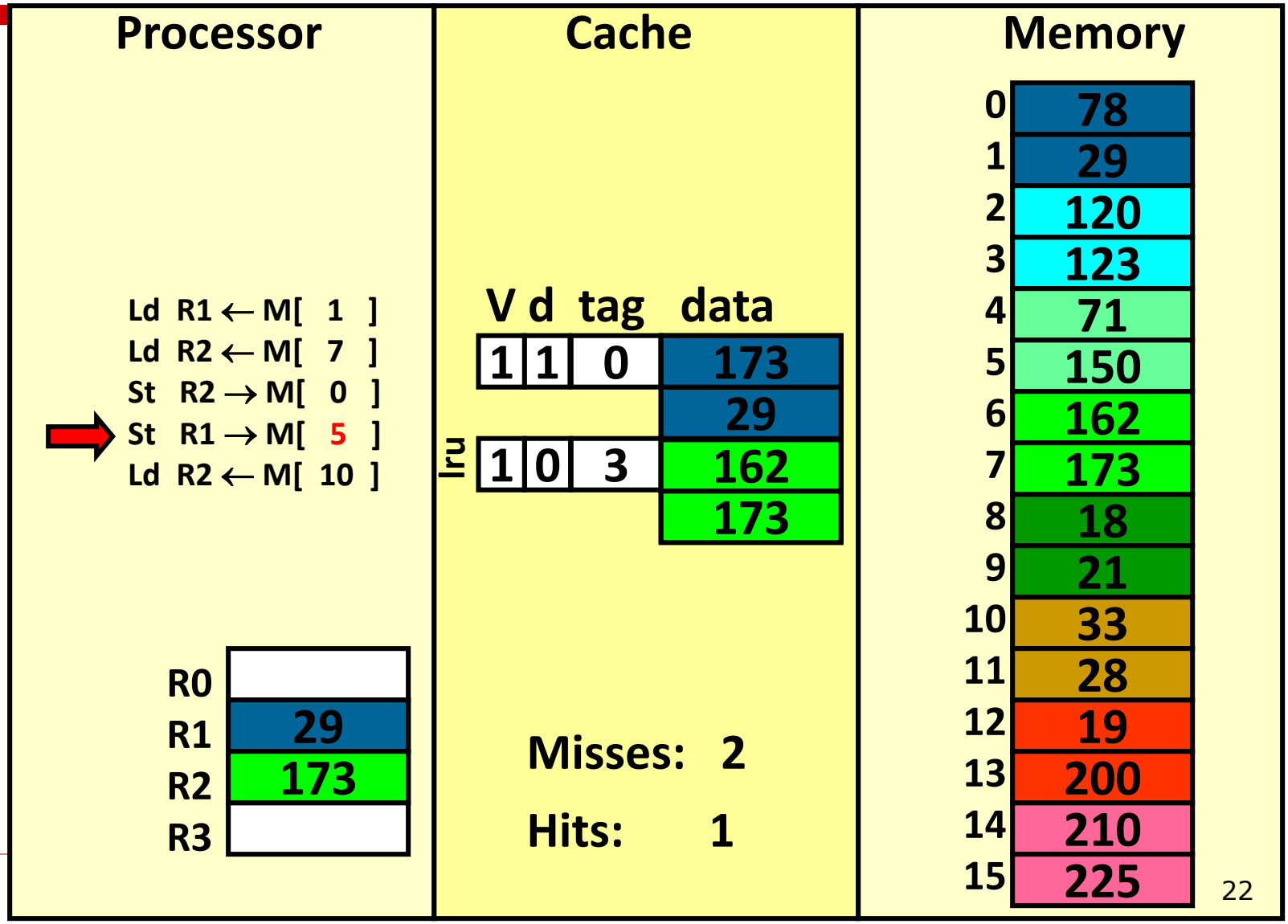
Review: write-through



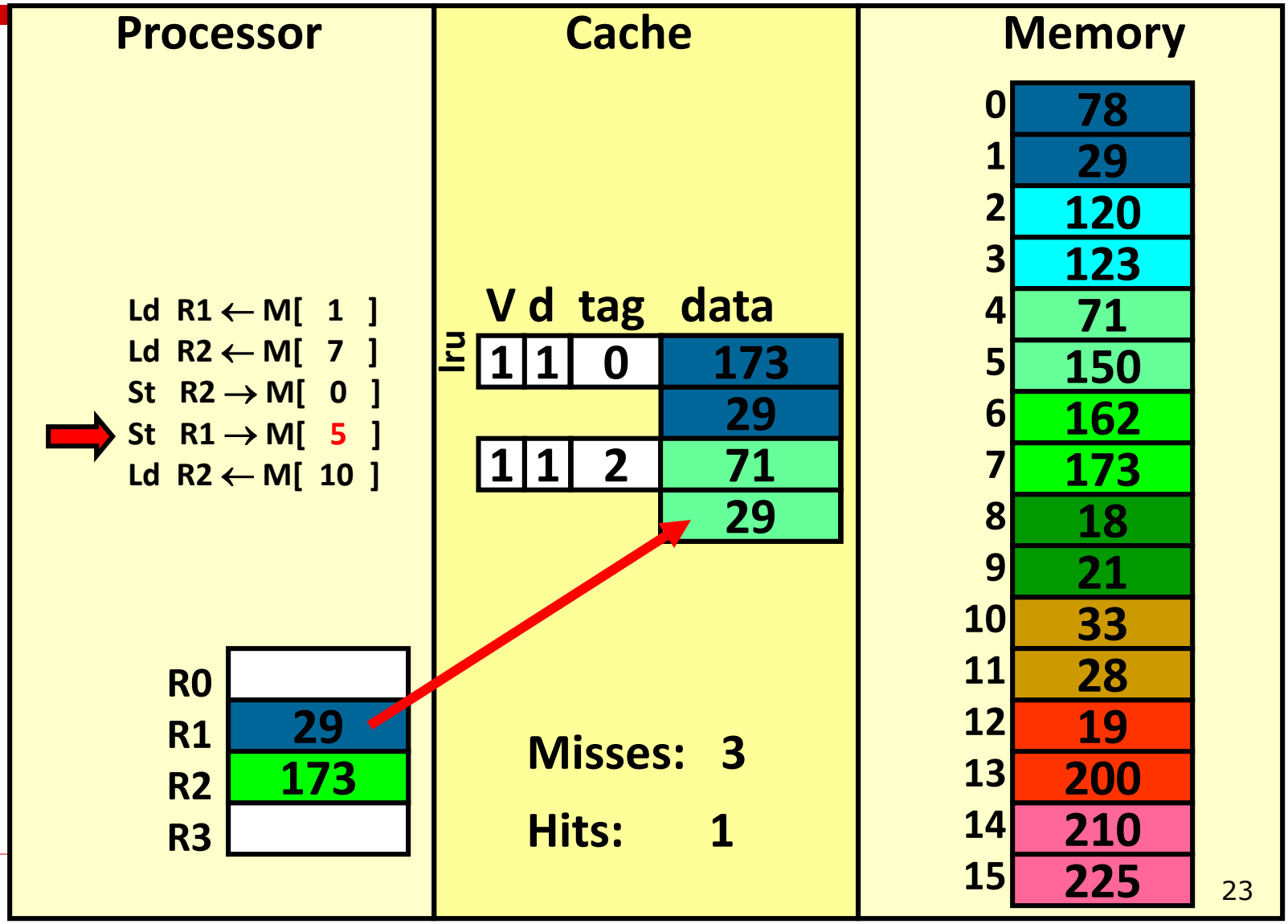
Review: write-through



Review: write-back



Review: write-back



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 ?

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 = 32 (Address) – 6 (block offset) = 26 bits
#blocks = 64K / 64 = 1024 ➜ LRU bits = 10
Overhead per block = 26(Tag) + 1(V) + 1(D) + 10 (LRU) = 38 bits

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%?

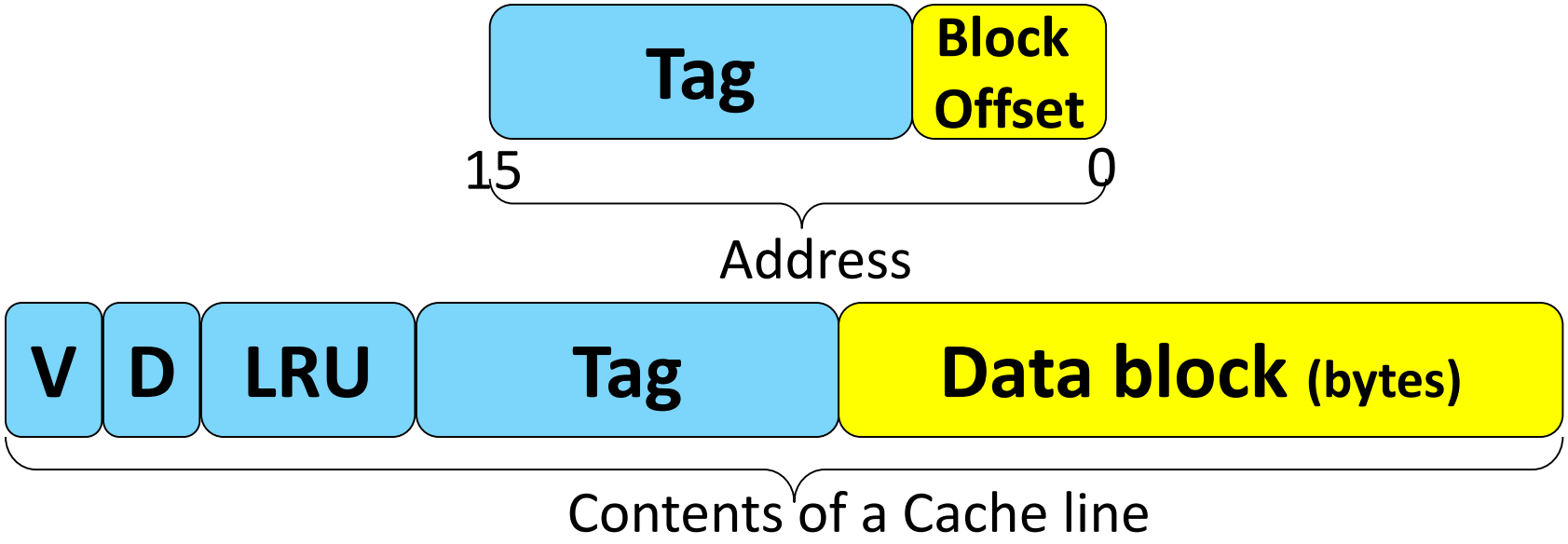
  AMAT = 10 + (1 - 0.97)*100 = 13 ns

- 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

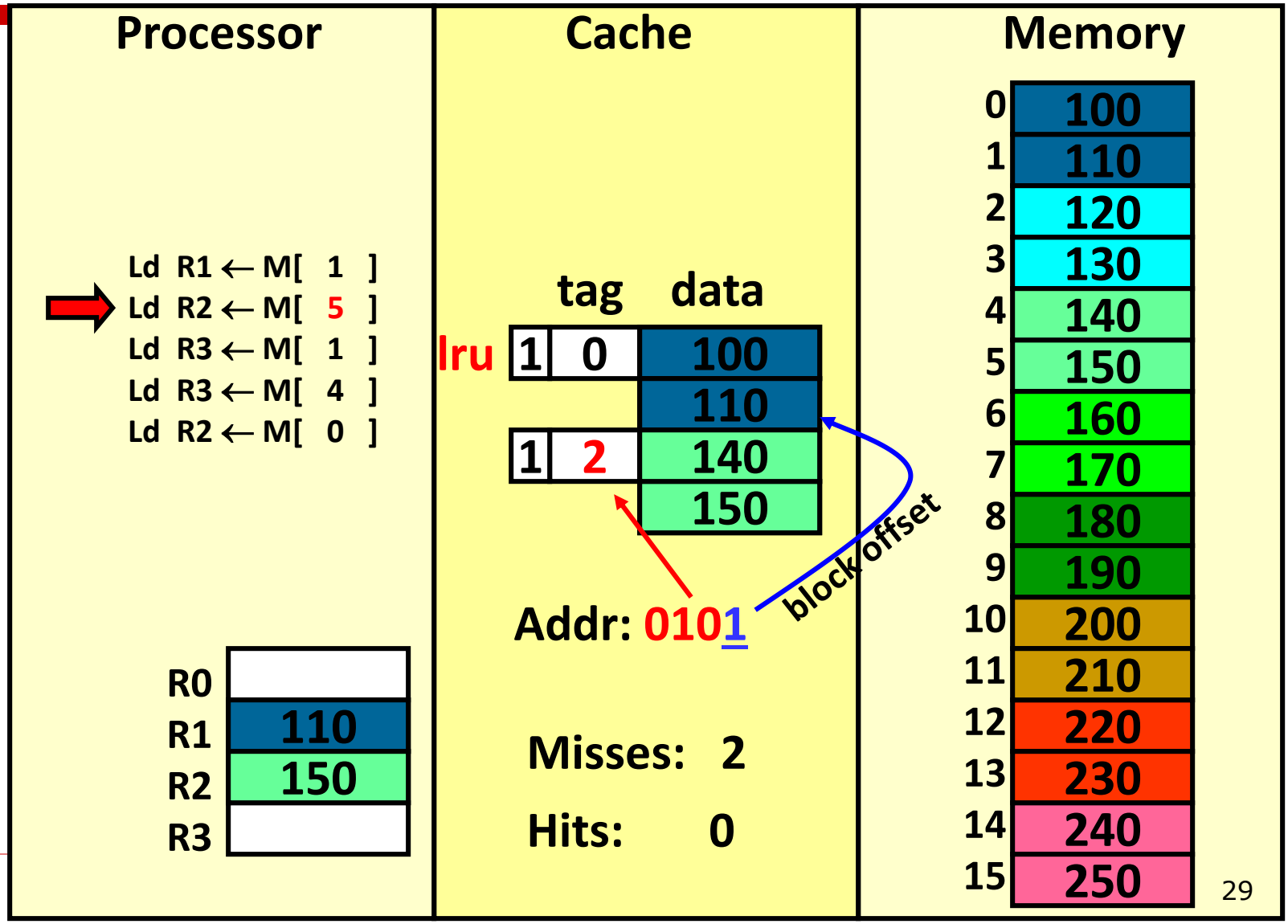
Review: Cache Organization

- Capture spatial locality (for performance)
- Reduce tag overhead (chip area is expensive)
- Example: 16-bit system => All addresses are 16-bit long



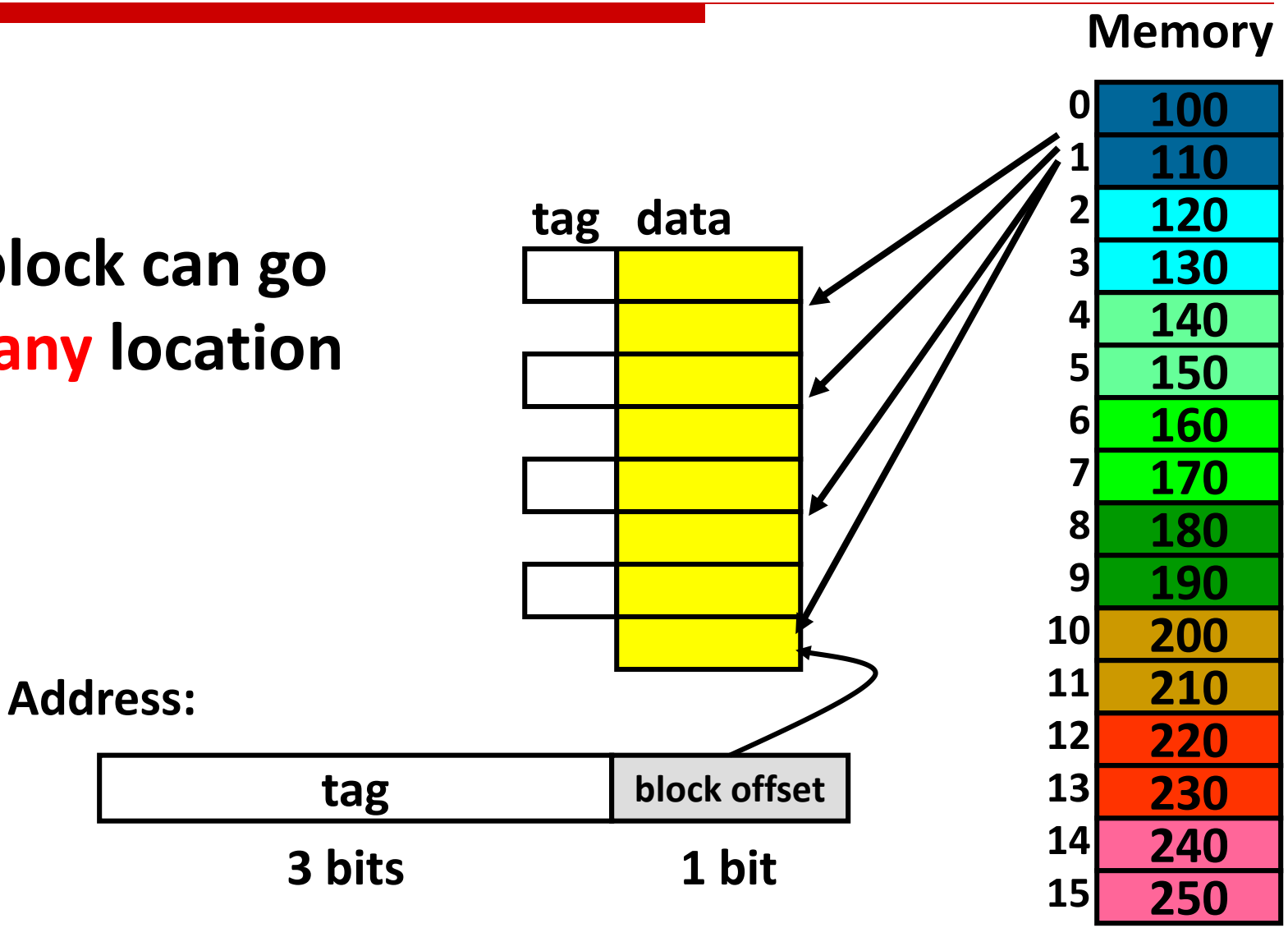
- Do not store block offset in the cache line
  - Determine byte to be read/written from the address directly

Review: How to find tag from address?



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.

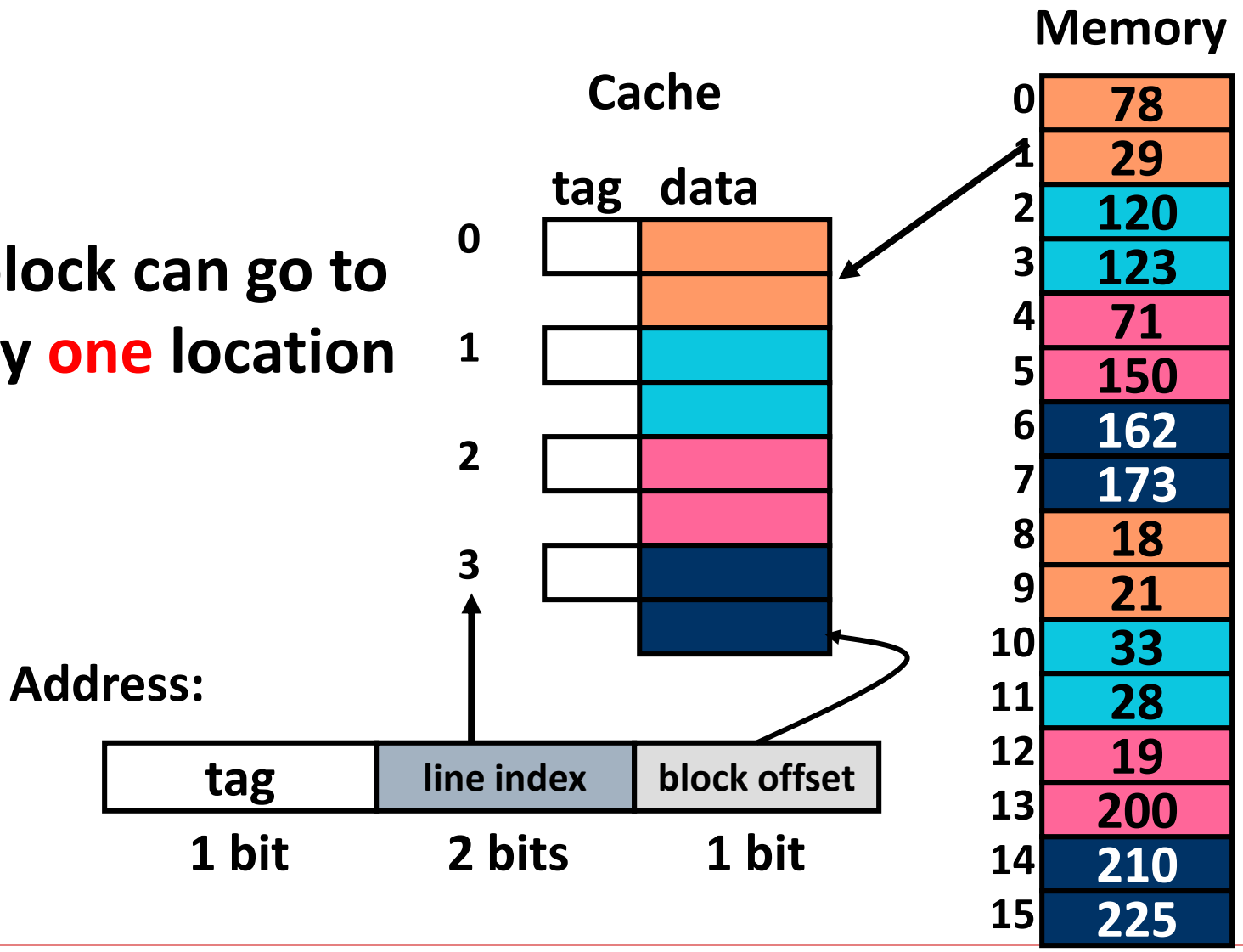
- This approach can be too slow sometimes
  - Parallel tag searches are expensive and can be slow Why?

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
  - You then only need to check a single tag – the one associated with the region the reference is located in

Mapping memory to cache (Direct-mapped)

A block can go to only one location

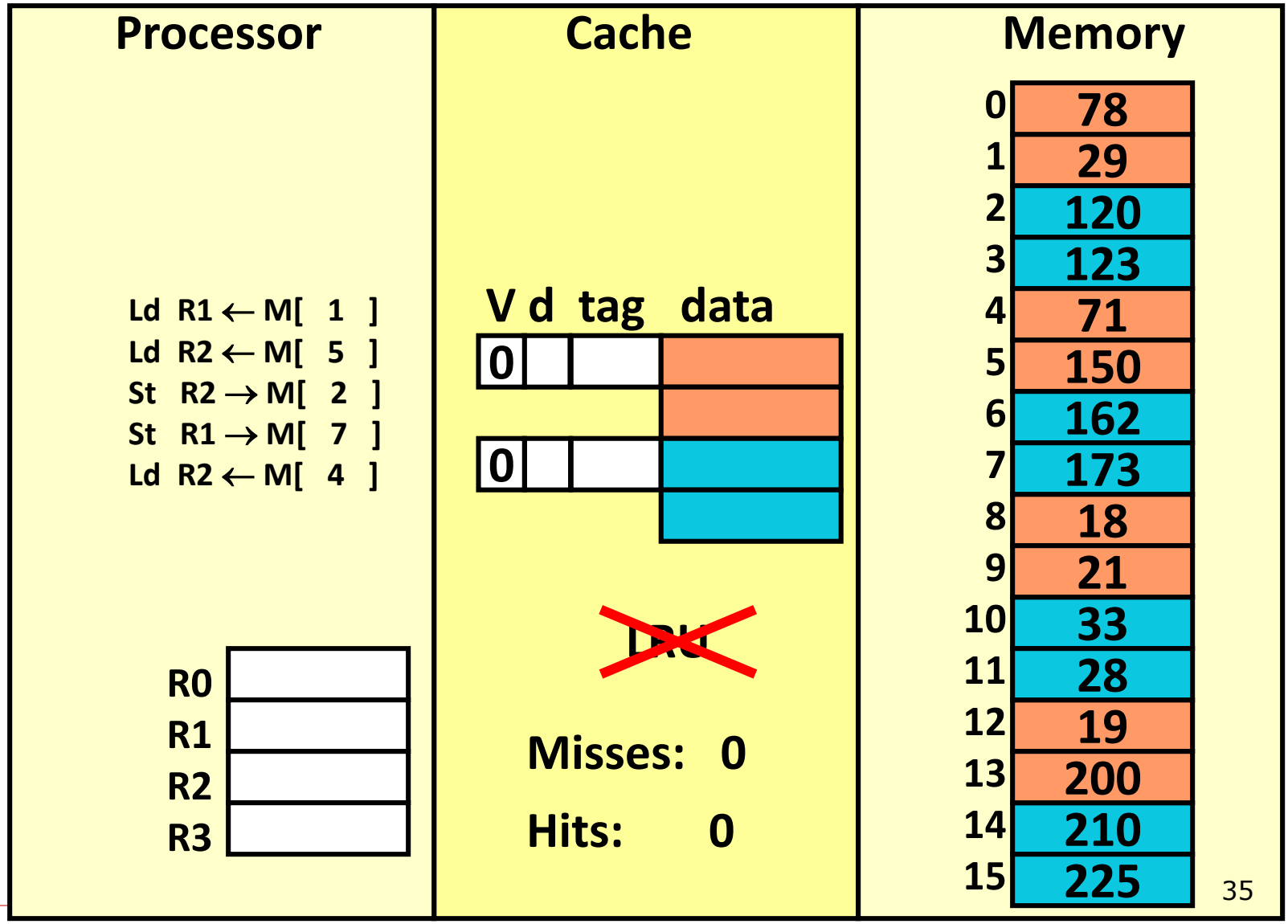


Direct mapped caches

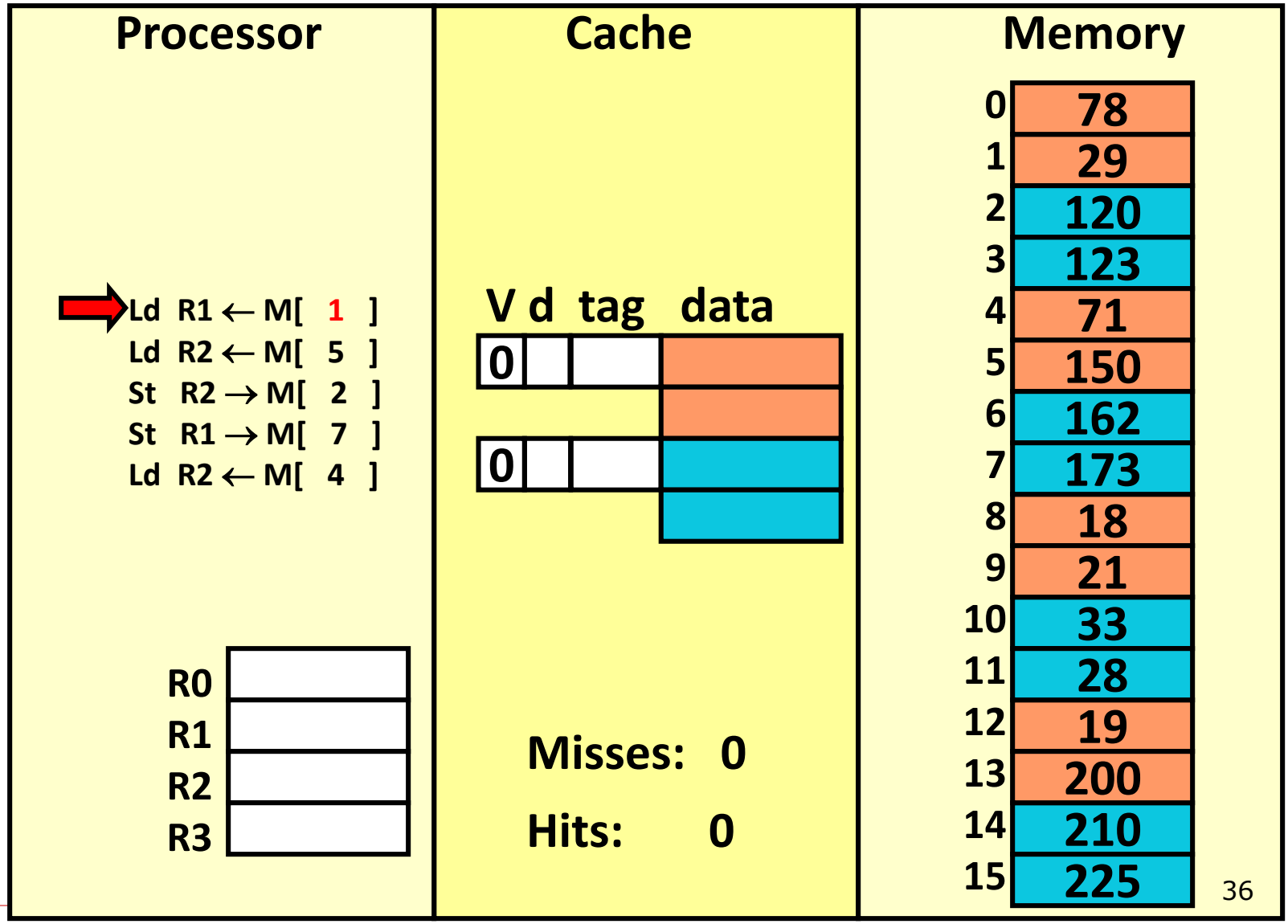
- Two blocks in memory that map to the same index in the cache cannot be present in the cache at the same time
  - One index → one entry

- 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

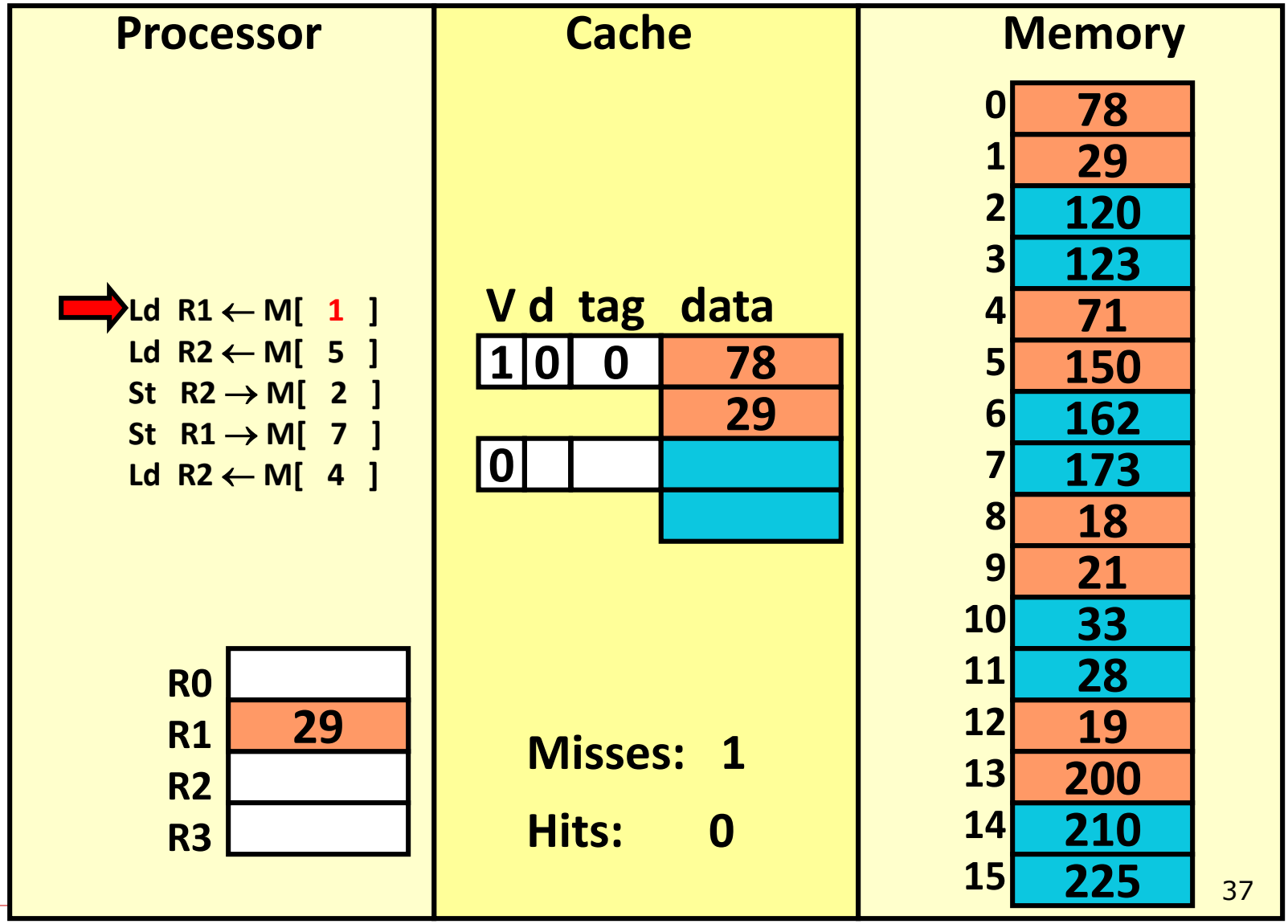
Direct-mapped cache



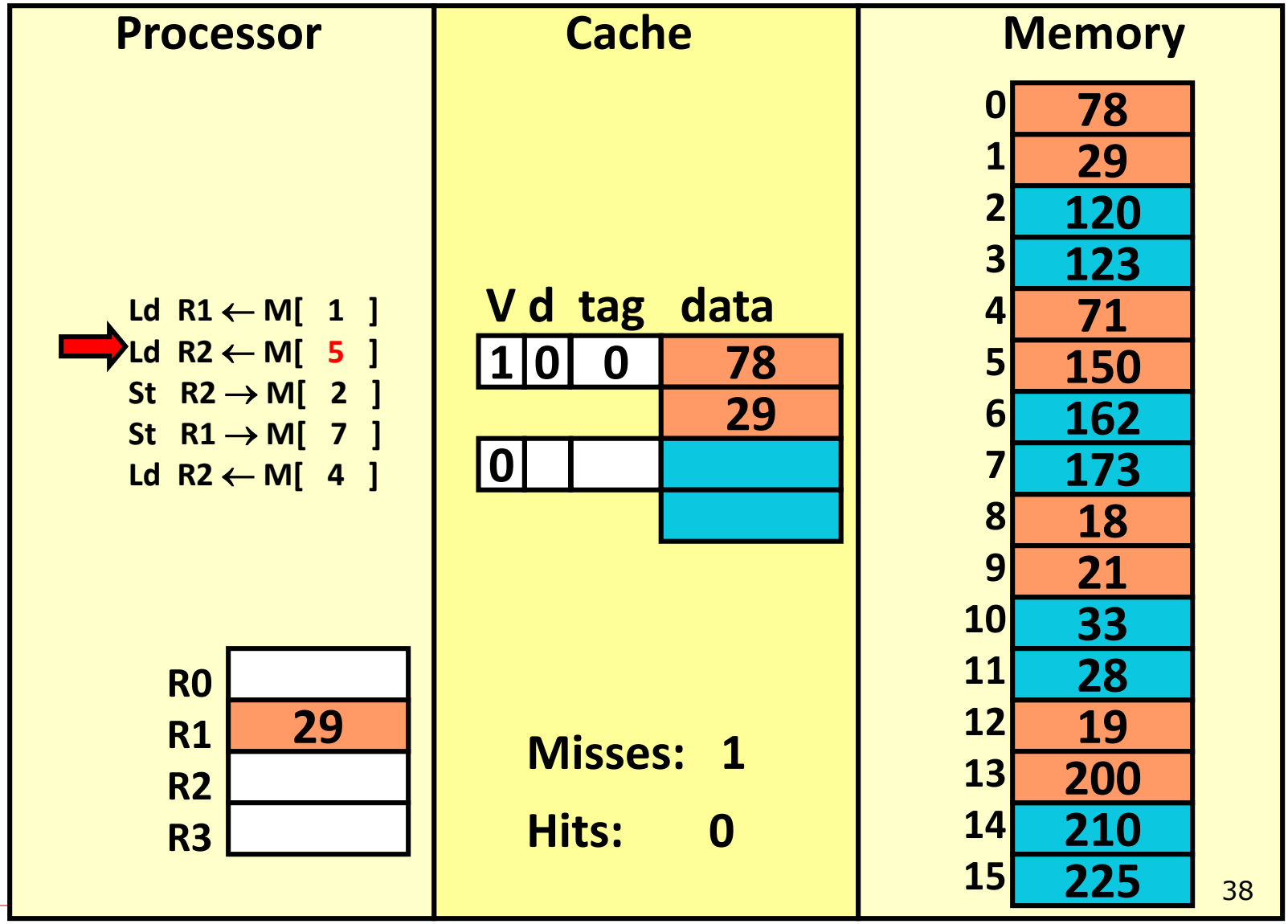
Direct-mapped (REF 1)



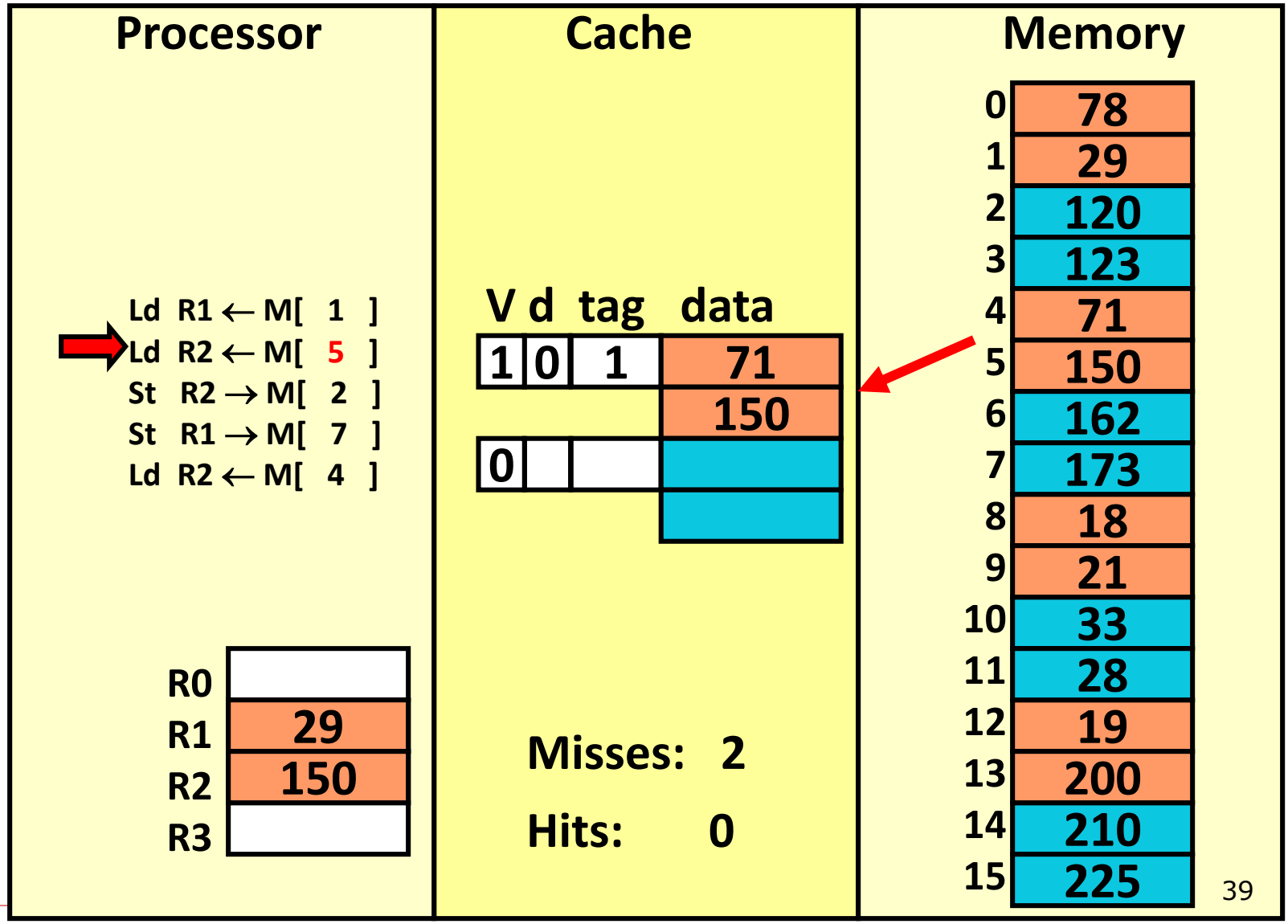
Direct-mapped (REF 1)



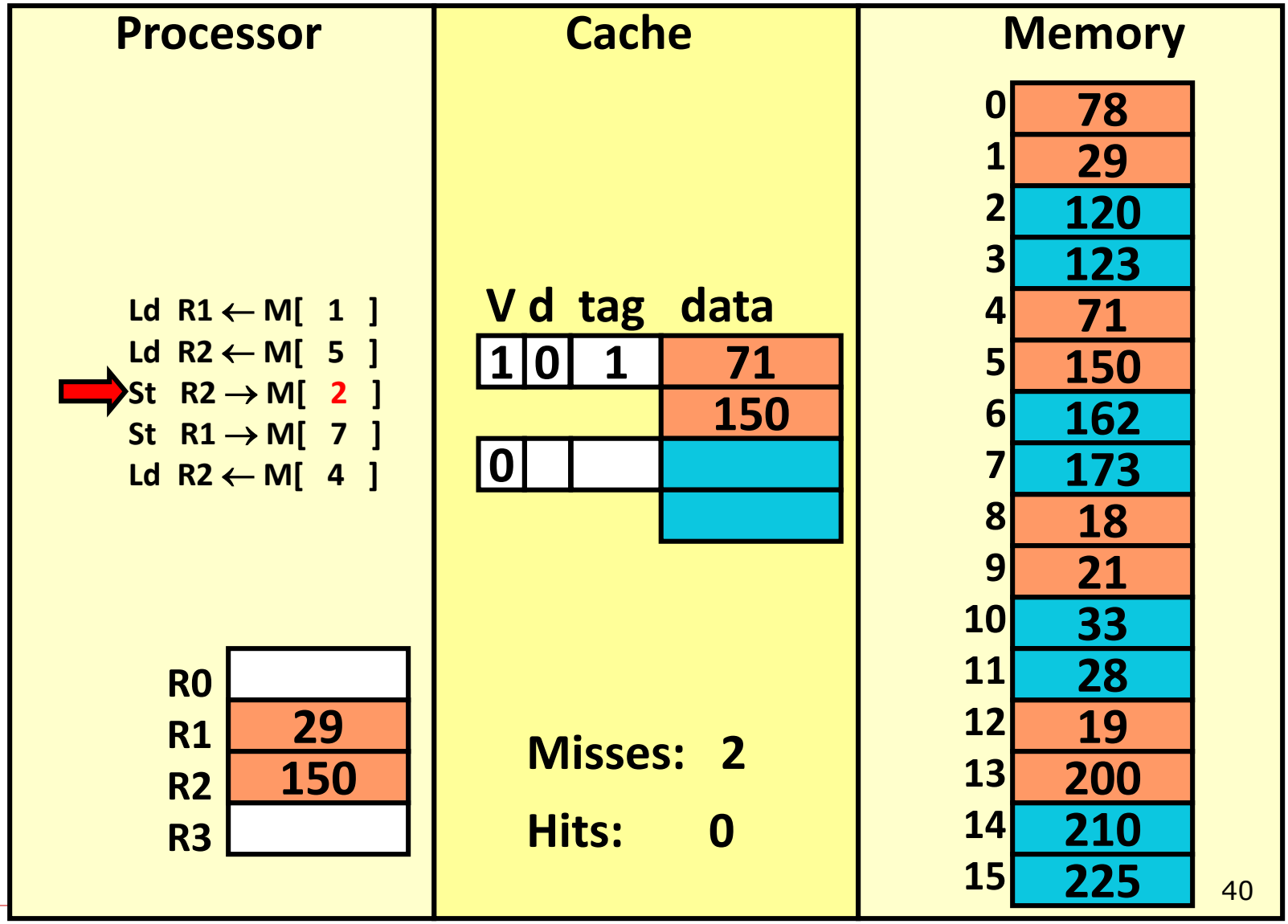
Direct-mapped (REF 2)



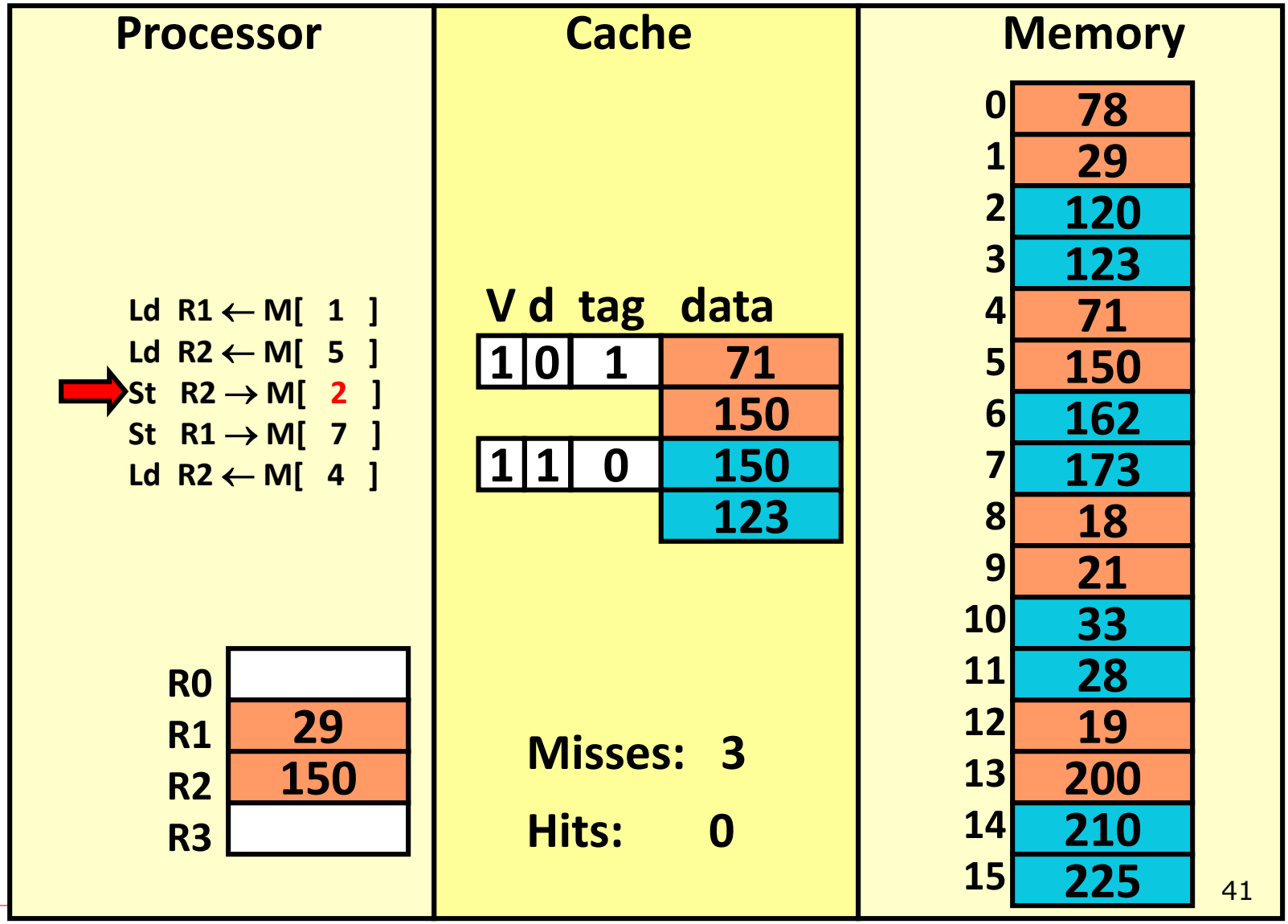
Direct-mapped (REF 2)



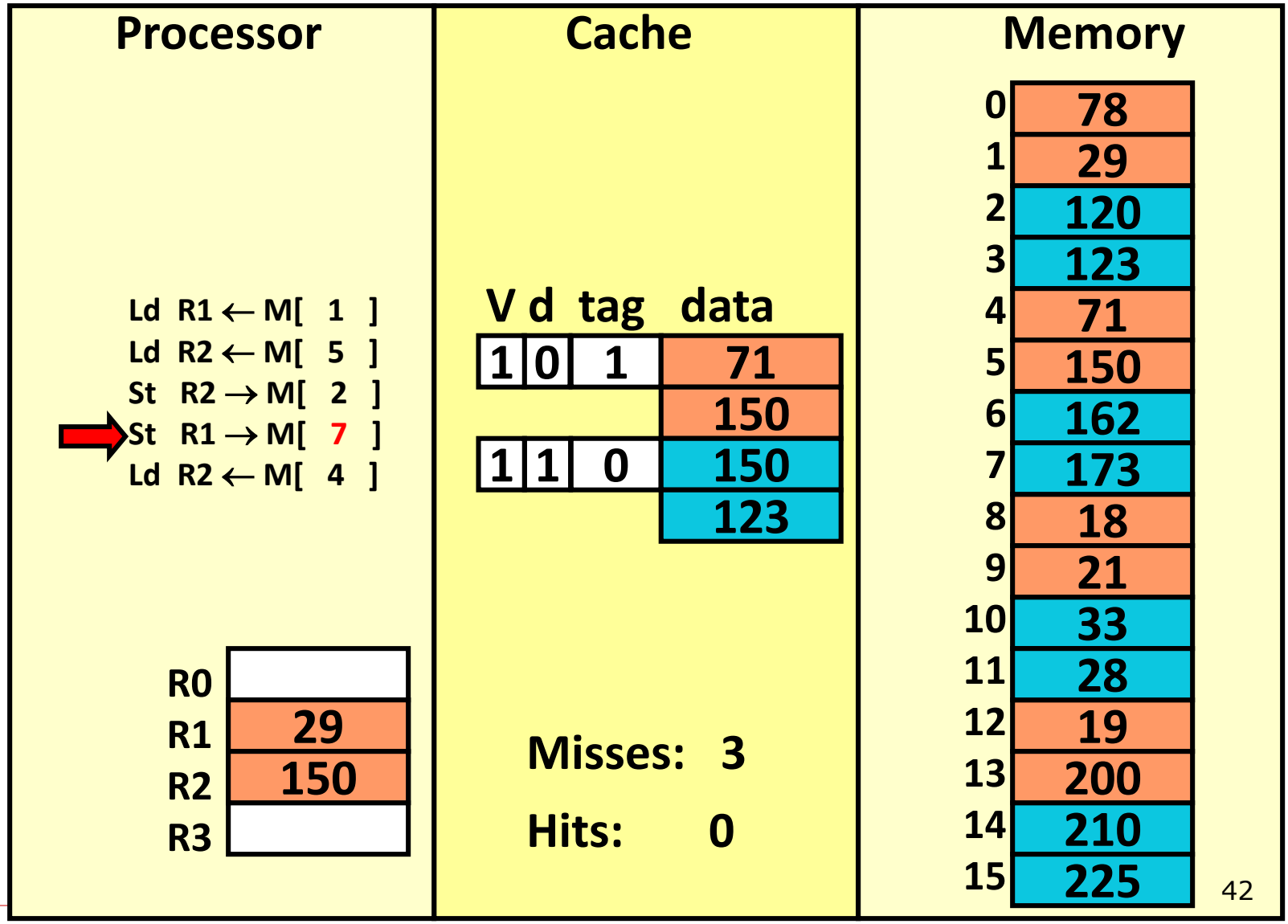
Direct-mapped (REF 3)



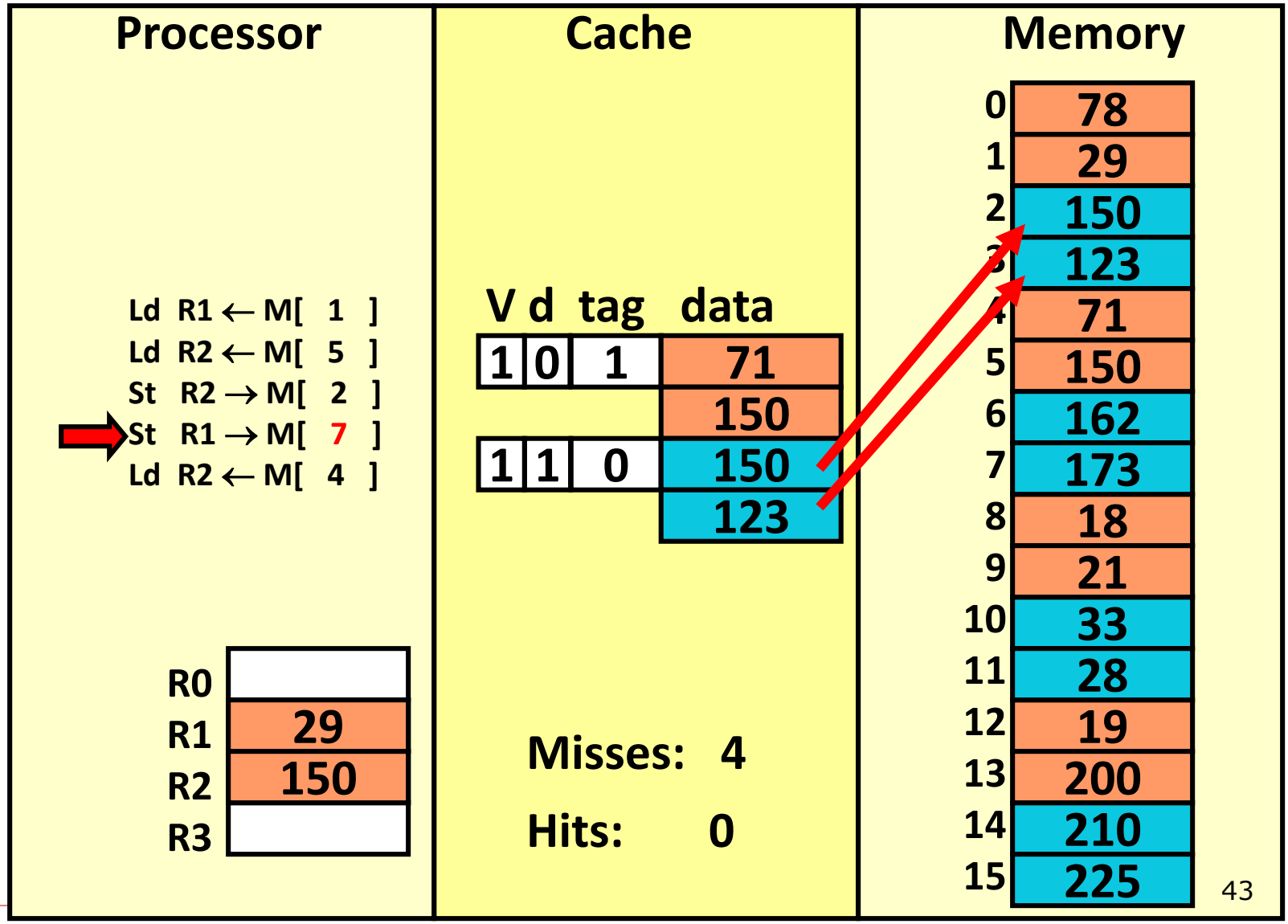
Direct-mapped (REF 3)



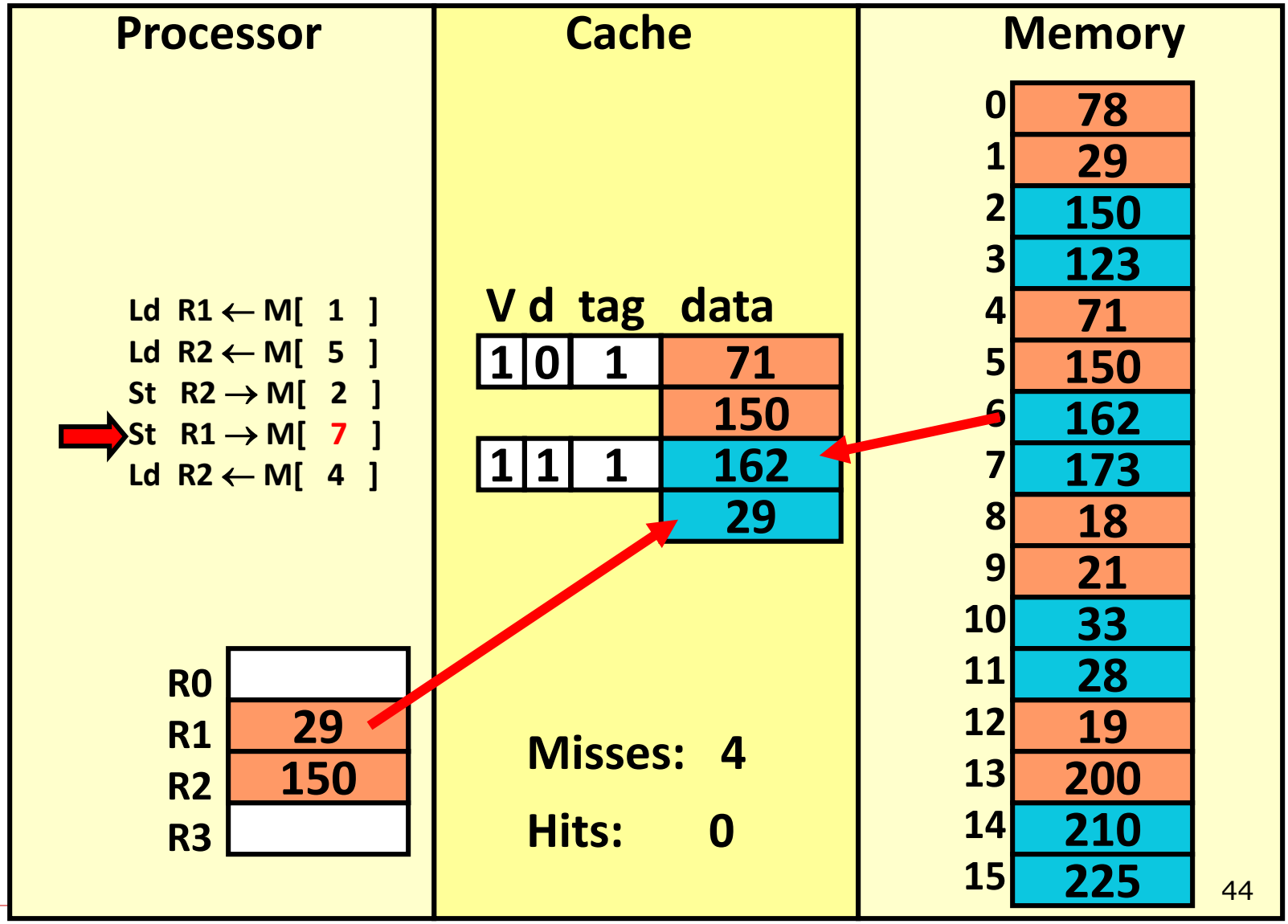
Direct-mapped (REF 4)



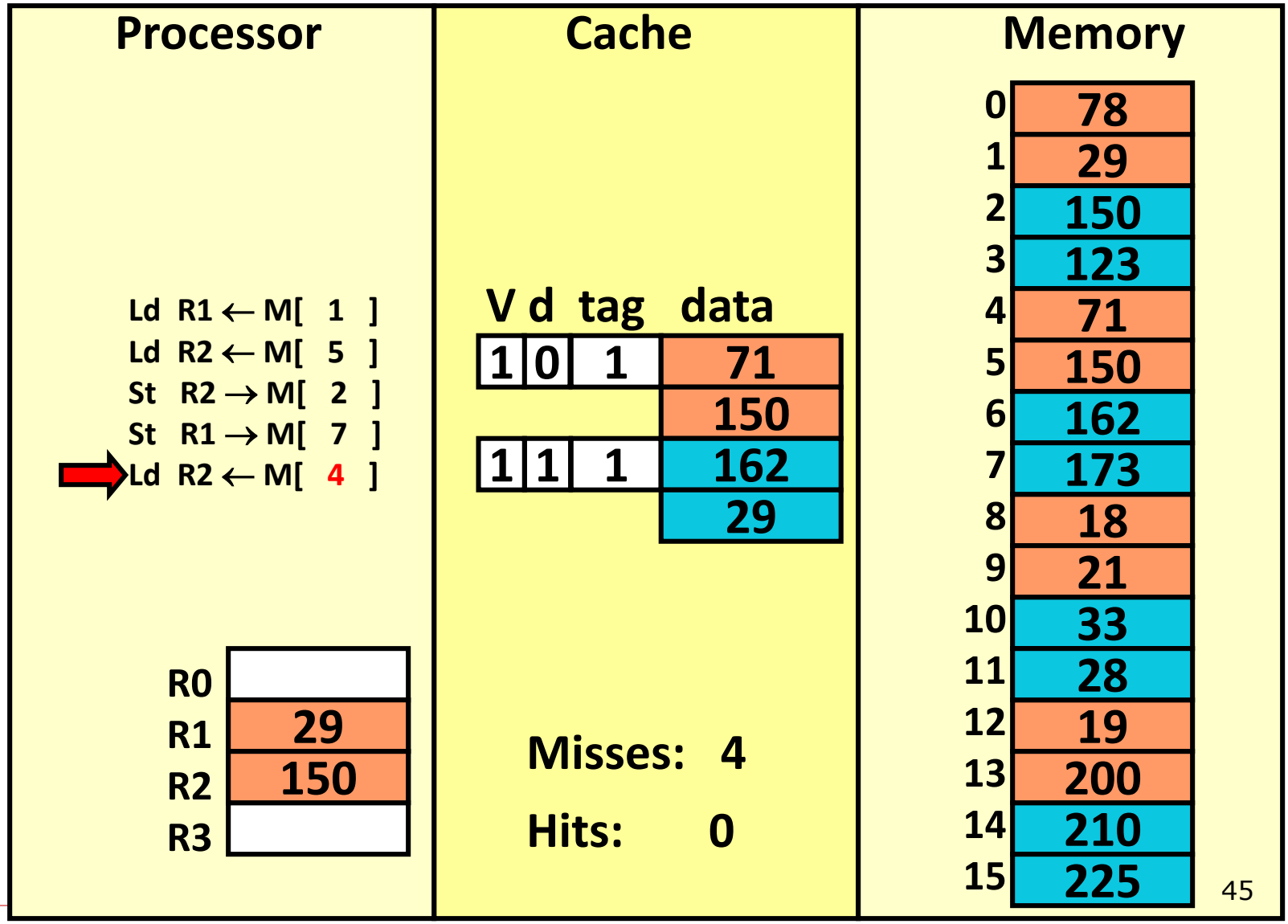
Direct-mapped (REF 4)



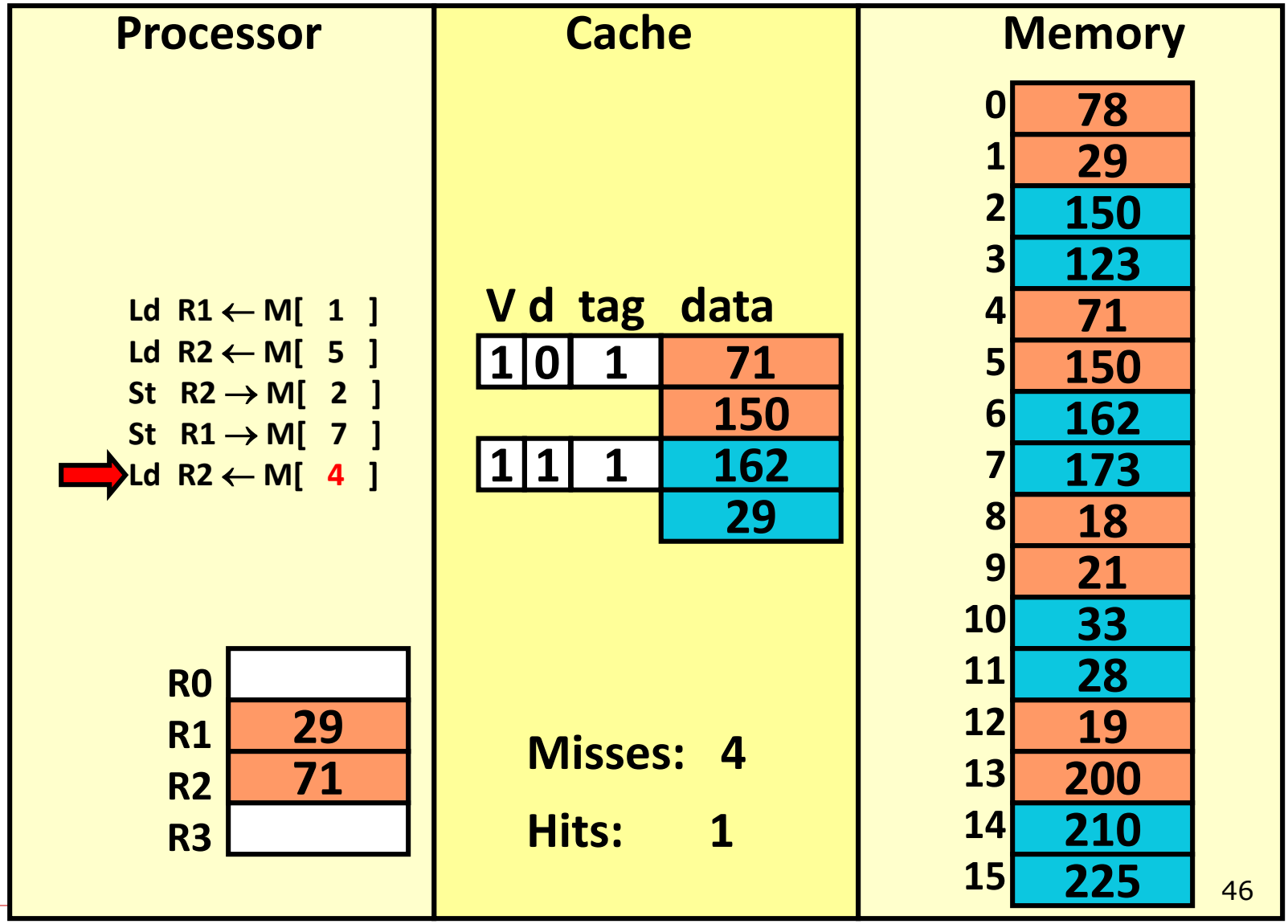
Direct-mapped (REF 4)



Direct-mapped (REF 5)



Direct-mapped (REF 5)



Class Problem

- How many tag bits are required for:
  - 32-bit address, byte addressed, direct-mapped 32k cache, 128 byte block size, write-back

- What are the overheads of this cache?

Class Problem

- How many tag bits are required for:
  - 32-bit address, byte addressed, direct-mapped 32k cache, 128 byte block size, write-back

    # Bytes in block = 128 ➜ Block offset = 7 bits (*byte addressable*)

    # Lines = 32k / 128 = 256 ➜ Line index = 8 bits

    Tag bits = 32 – 7 – 8 = 17 bits

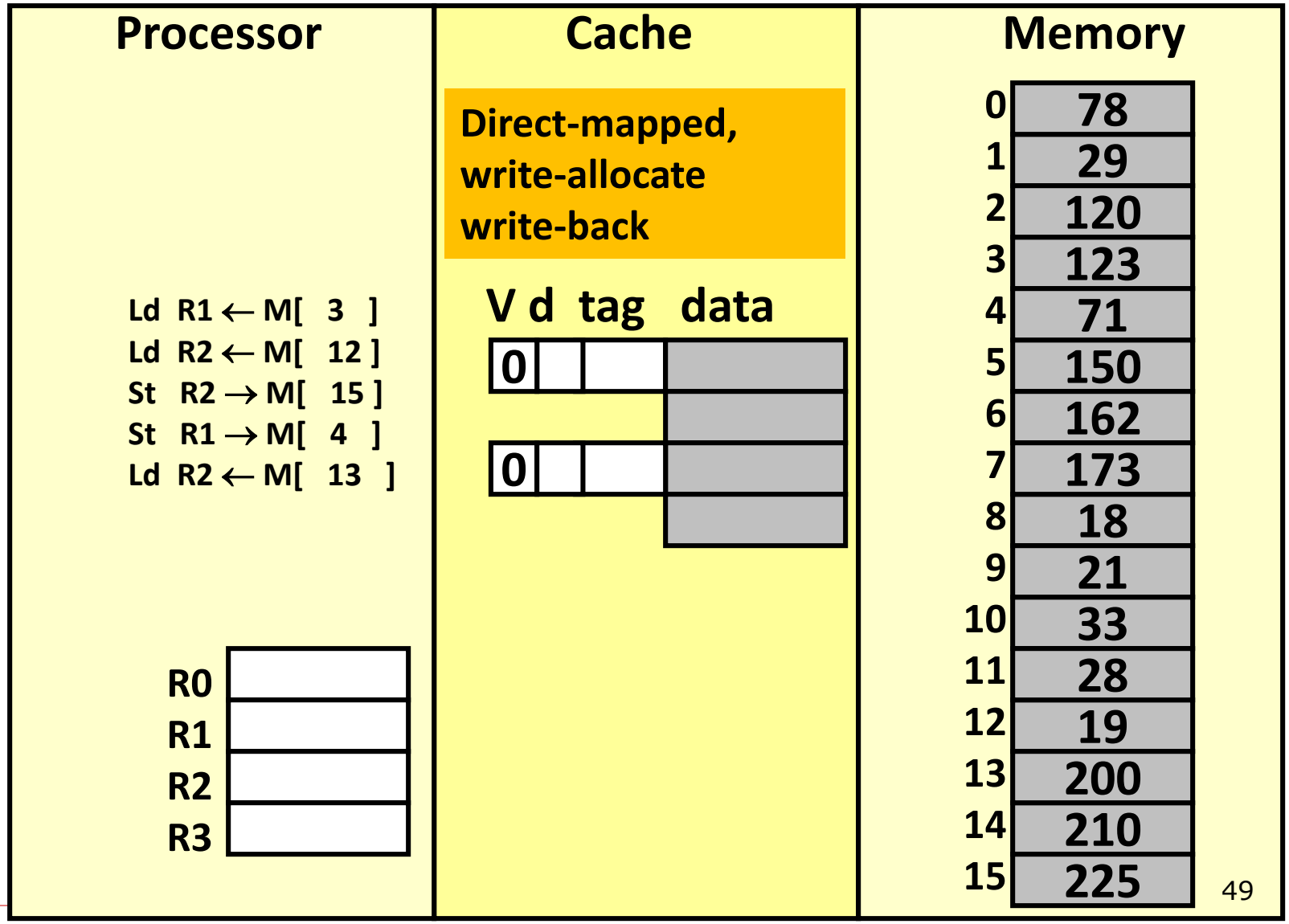
- What is the overhead of this cache?

  17 bits (Tag) + 1 bit (Valid) + 1 bit (Dirty) = 19 bits / line

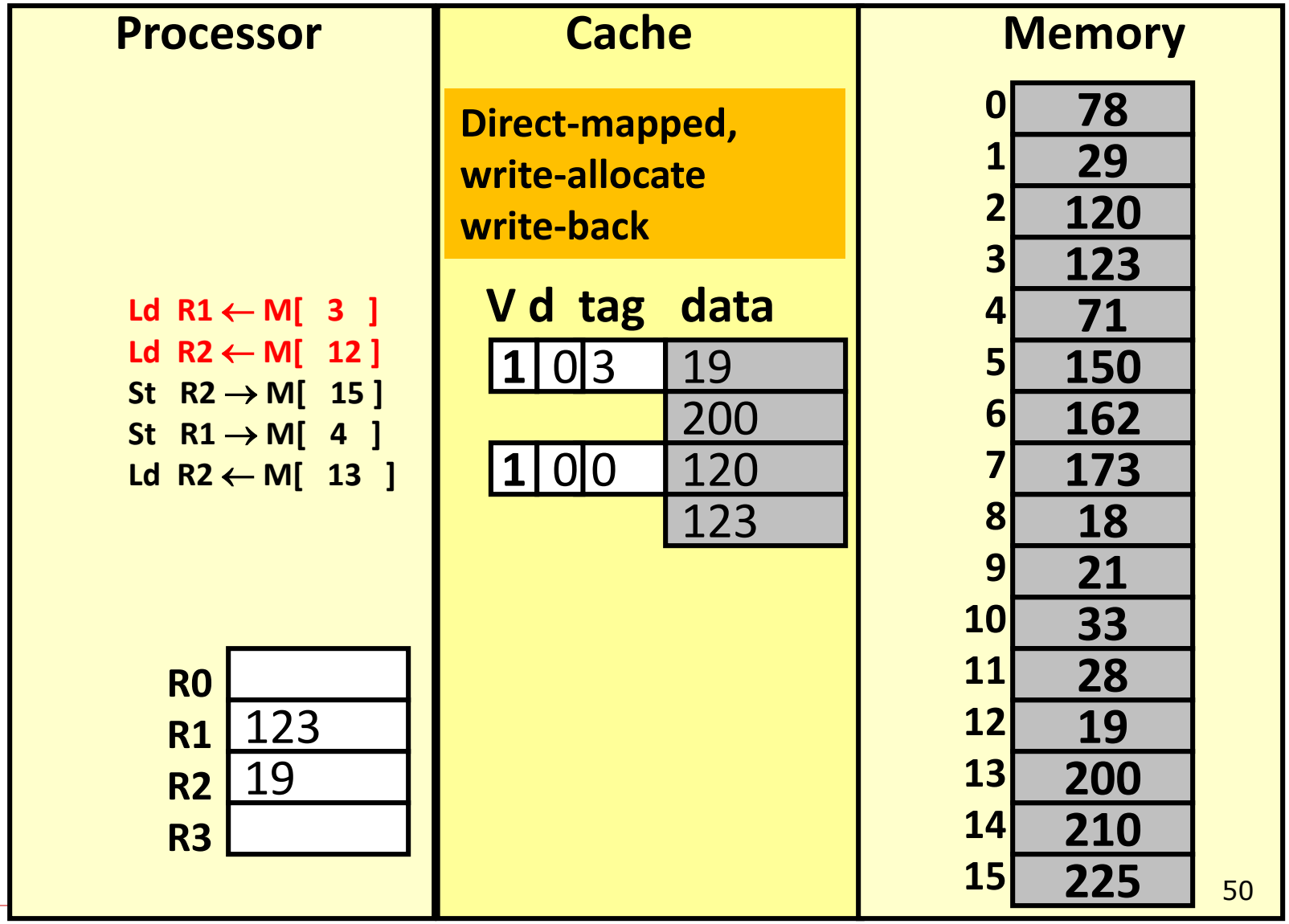
  19 bits / line * 256 lines = 4864 bits

  4864 bits / 32KB = 1.9% overheads

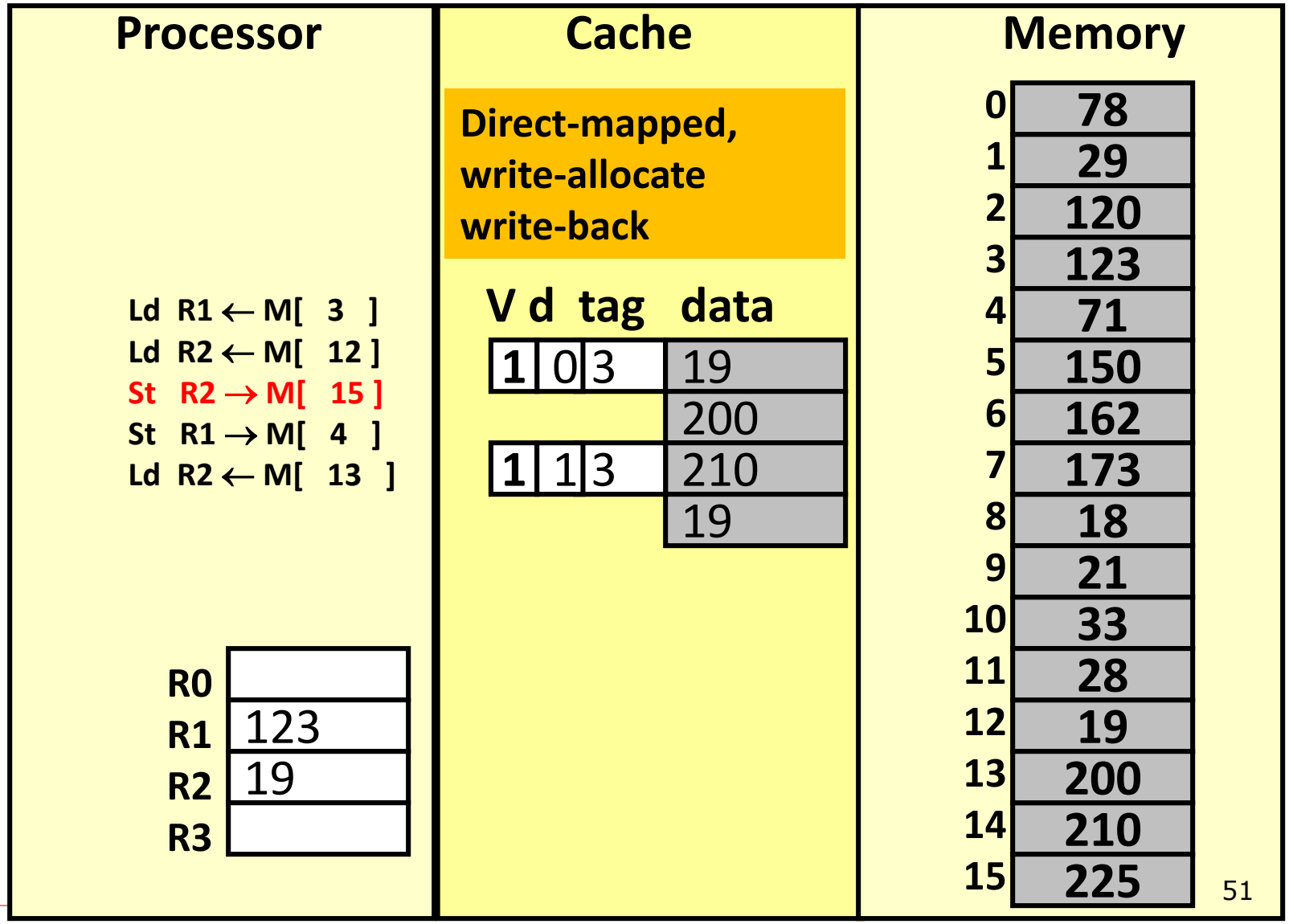
Class Problem – What is the state of the cache after executing the following instruction sequence?



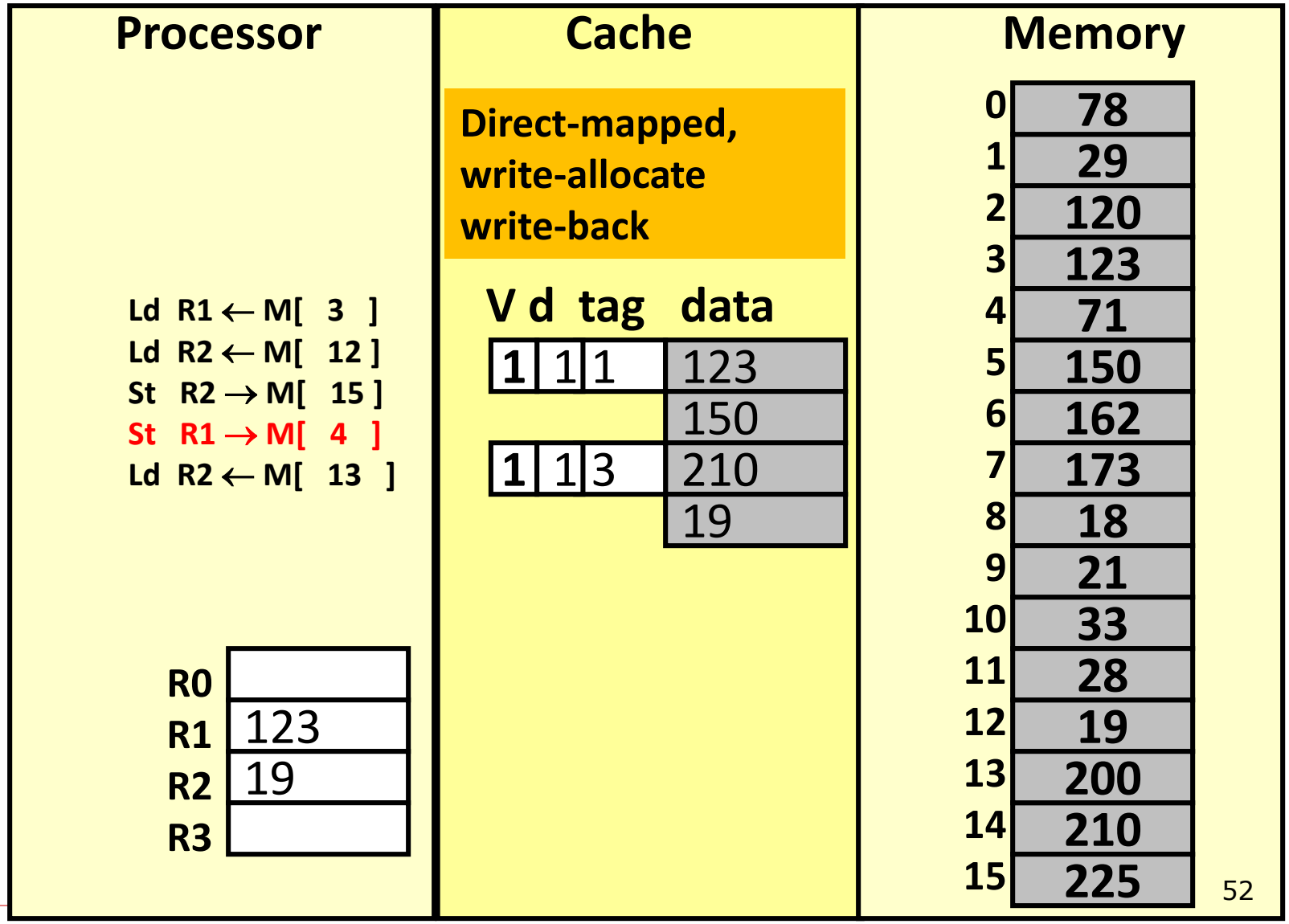
Class Problem – What is the state of the cache after executing the following instruction sequence?



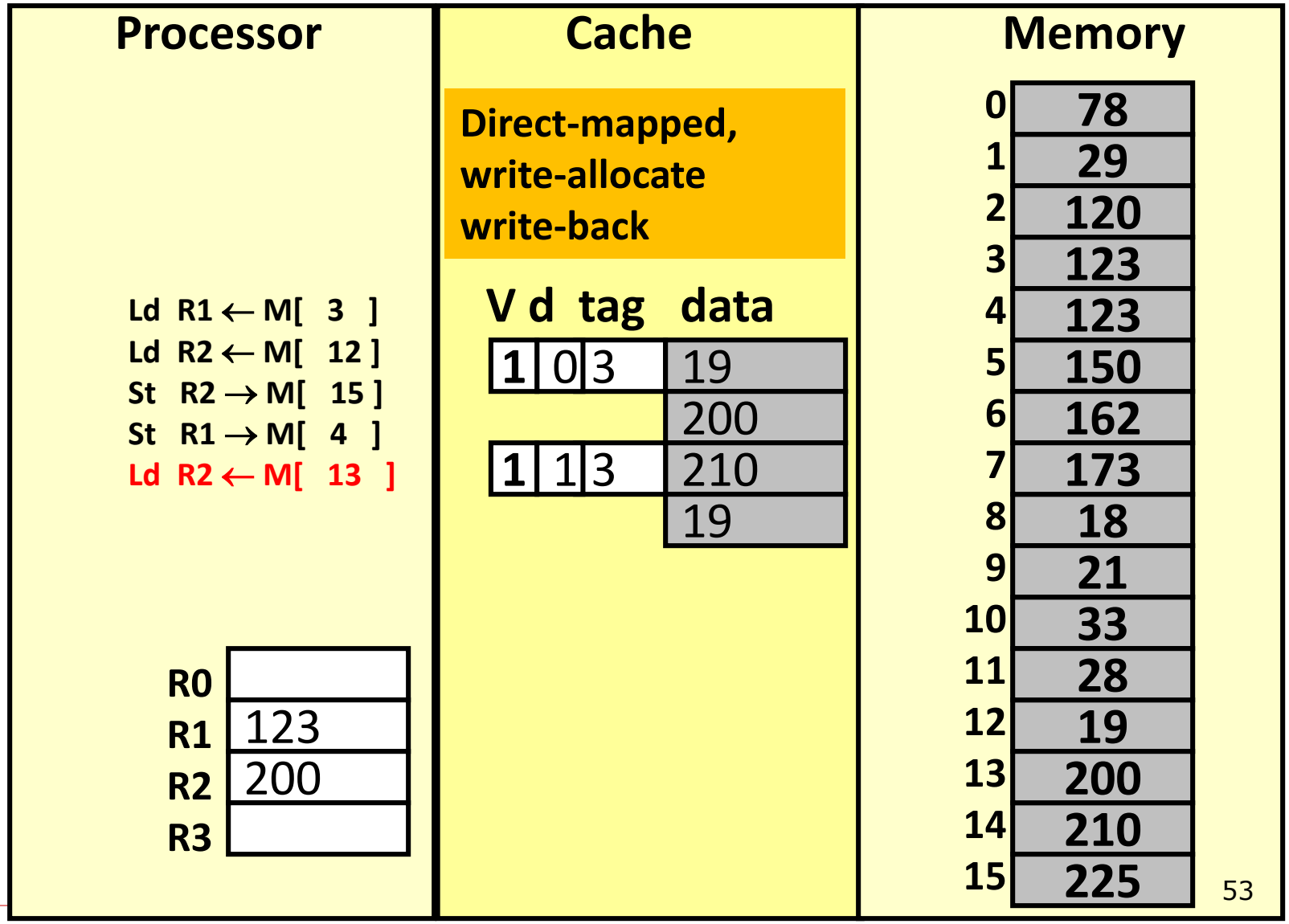
Class Problem – What is the state of the cache after executing the following instruction sequence?



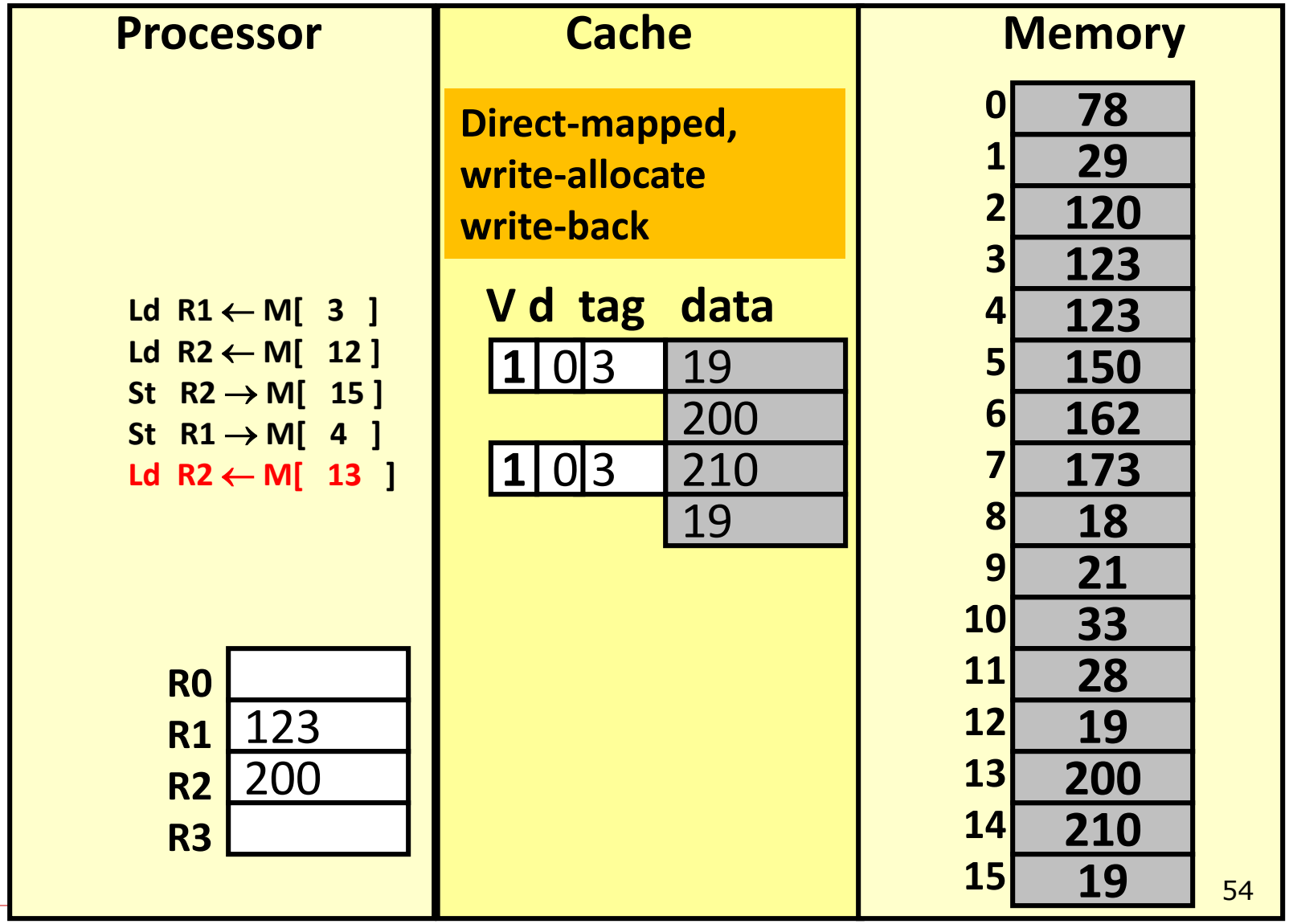
Class Problem – What is the state of the cache after executing the following instruction sequence?



Class Problem – What is the state of the cache after executing the following instruction sequence?



Class Problem – What is the state of the cache after executing the following instruction sequence?



What about cache for instructions?

- Instructions should be cached as well
- We have two choices:
  1. Treat instruction fetches as normal data and allocate cache lines when fetched
  2. Create a second cache (called the instruction cache or iCache) which caches instructions only

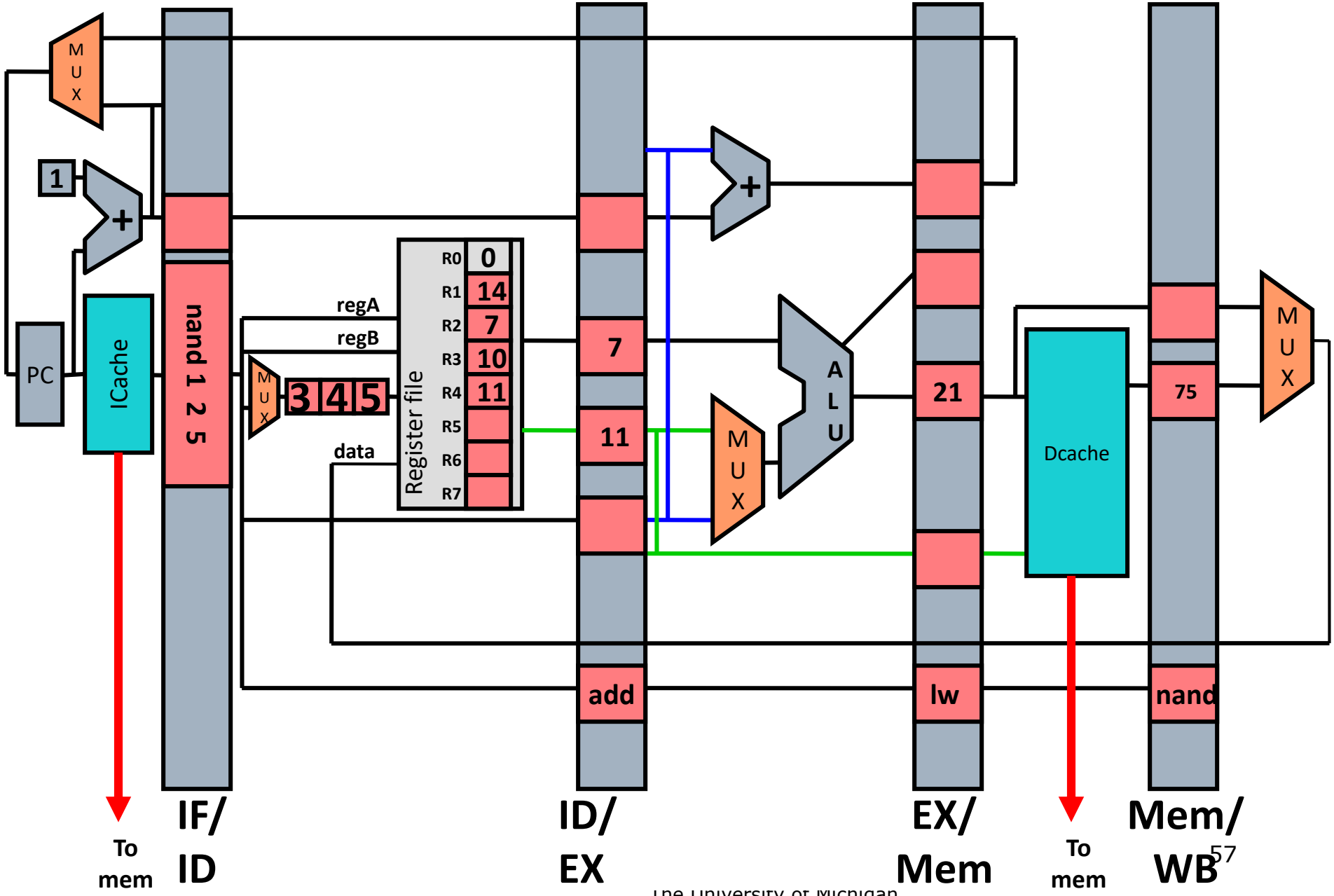
How do you know which cache to use?

What are advantages of a separate iCache?

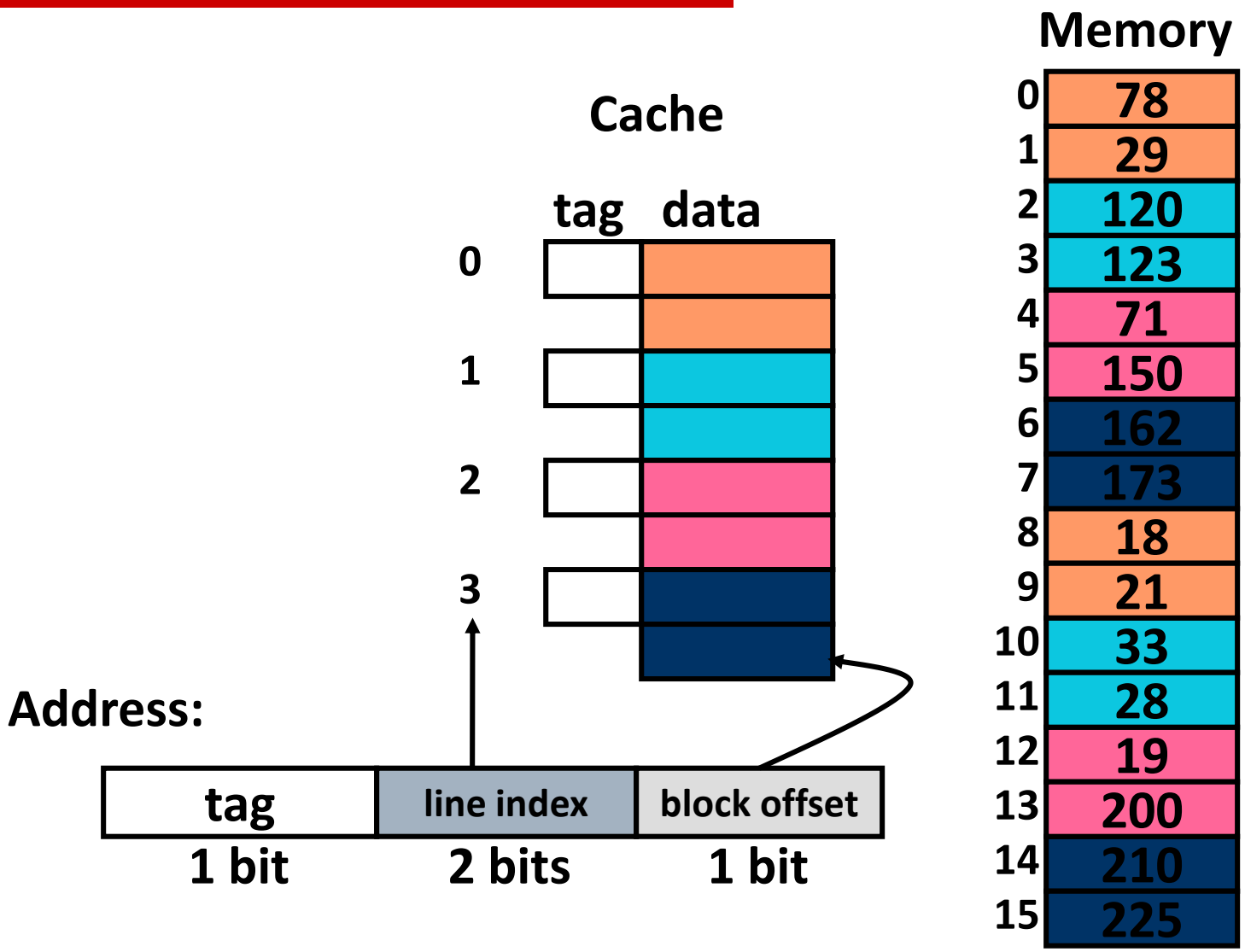
Integrating Caches into a Pipeline

- How are caches integrated into a pipelined implementation?
  - Replace instruction memory with Icache
  - Replace data memory with Dcache
- Issues:
  - Memory accesses now have variable latency
  - Both caches may miss at the same time

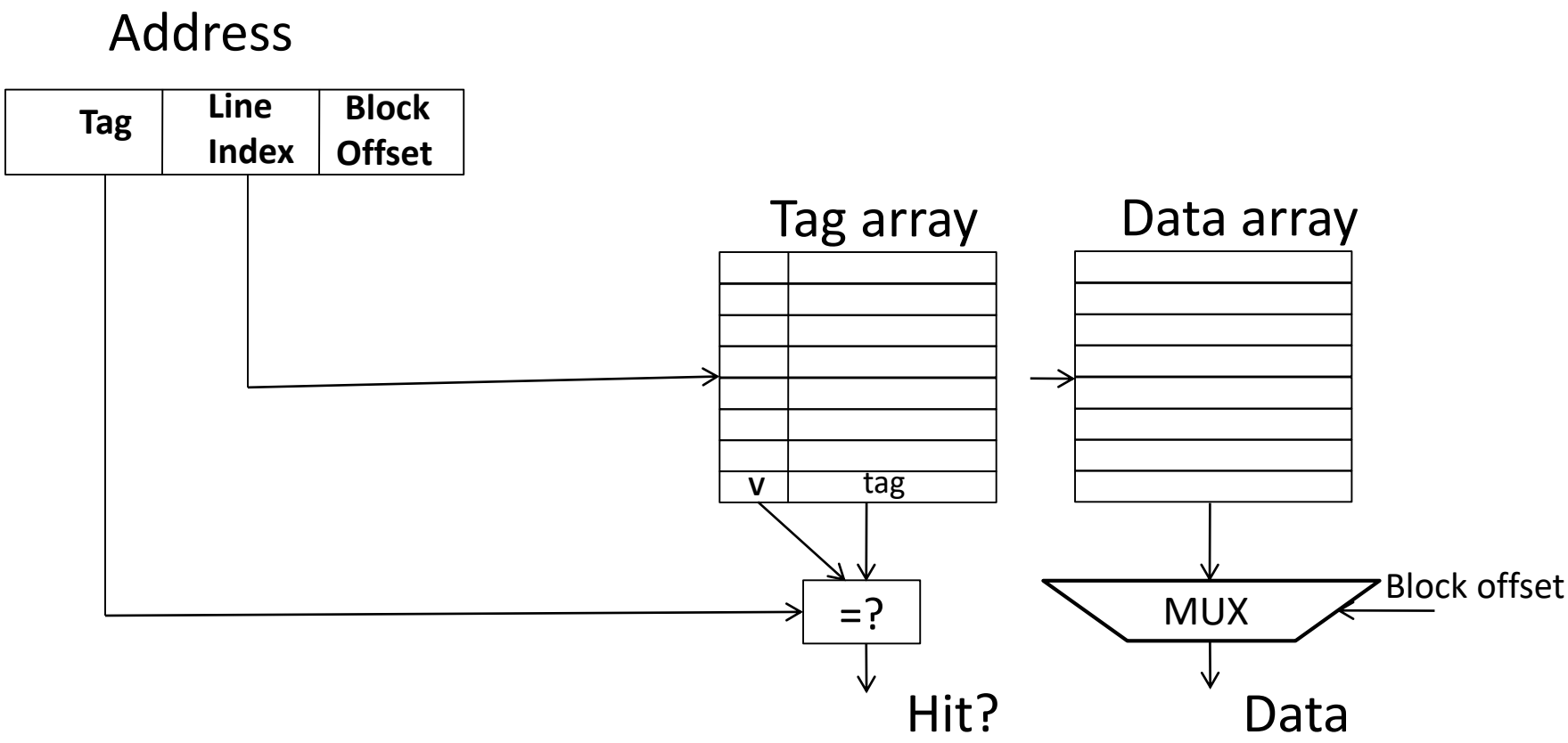
LC2K Pipeline with Caches



Summary: Direct-mapped caches



Direct-mapped cache: Placement & Access



Get the advantage of both…

- Set associative caches:
  - Partition memory into regions
    - like direct mapped but fewer partitions
  - Associate a region to a set of cache lines
    - Check tags for all lines in a set to determine a HIT
- Treat each line in a set like a small fully associative cache
  - LRU (or LRU-like) policy generally used

Set-associative cache