18. Cache organization: Block Size & Writes

EECS 370 – Introduction to Computer Organization – Winter 2023

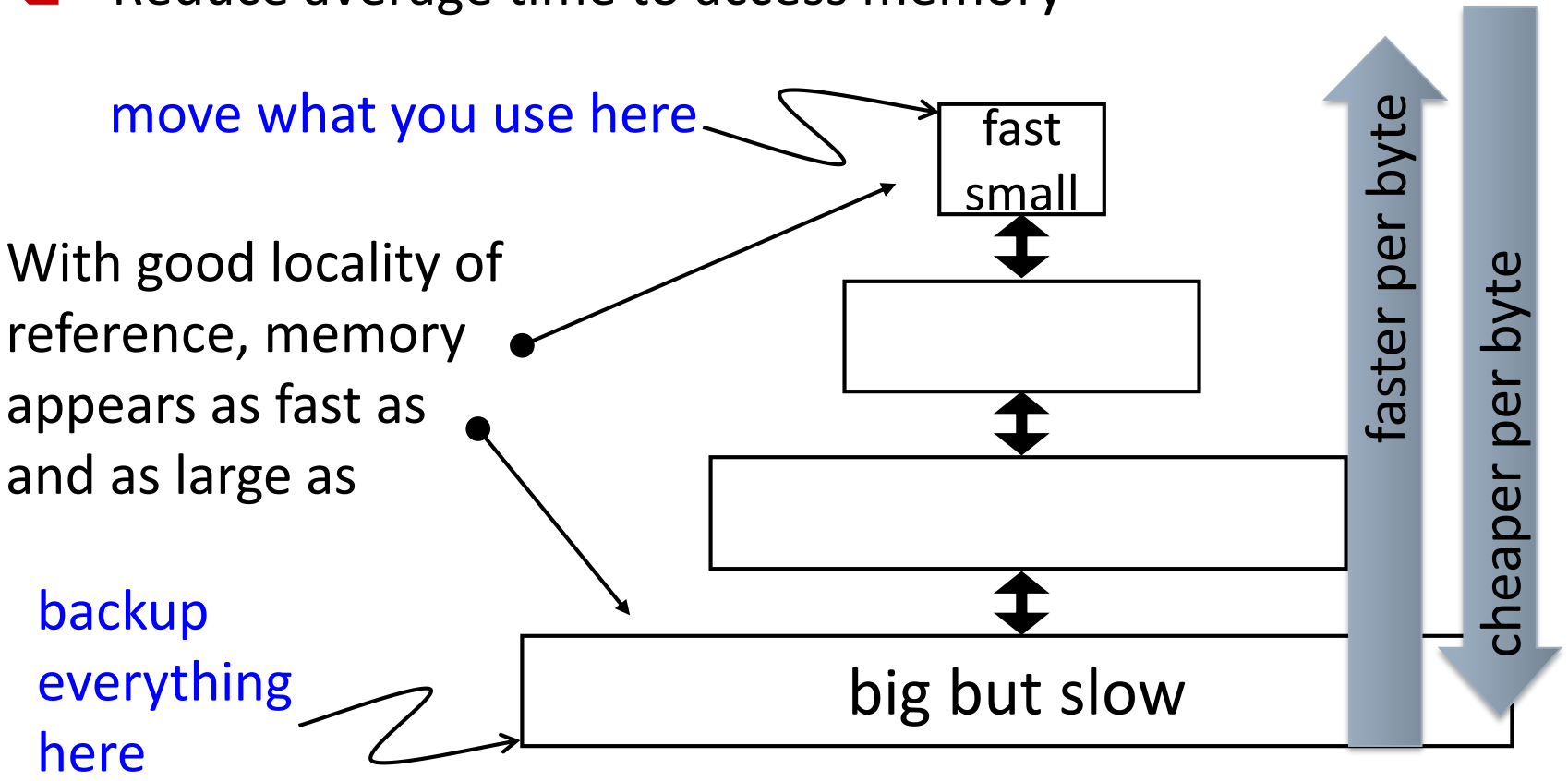
EECS Department
University of Michigan in Ann Arbor, USA

Checking in and stuff upcoming

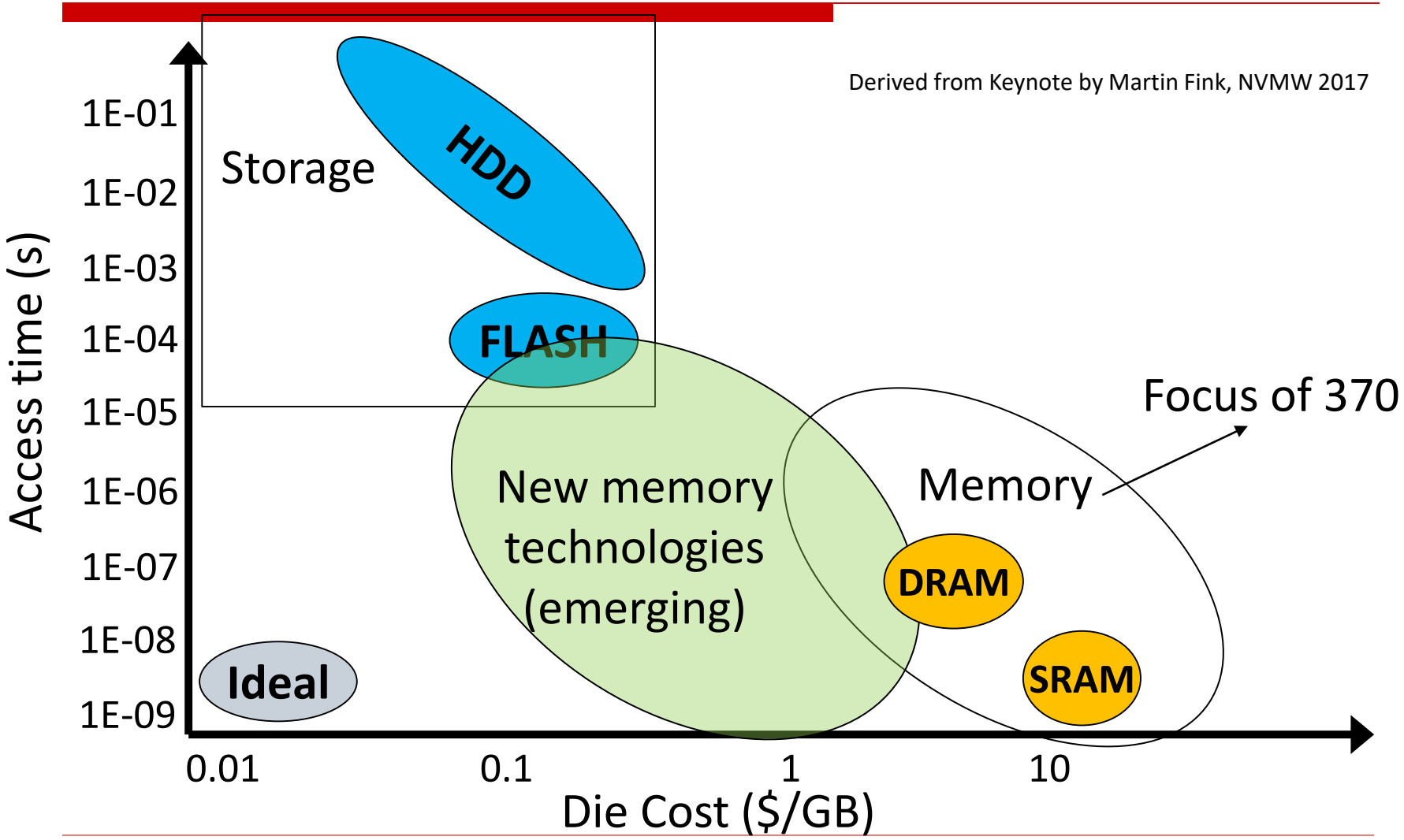
- Thursday 3/16: Project 3 milestone
- Monday 3/20: Homework 4 due
- Thursday 3/23: Project 3 due

Review: Memory hierarchy goals

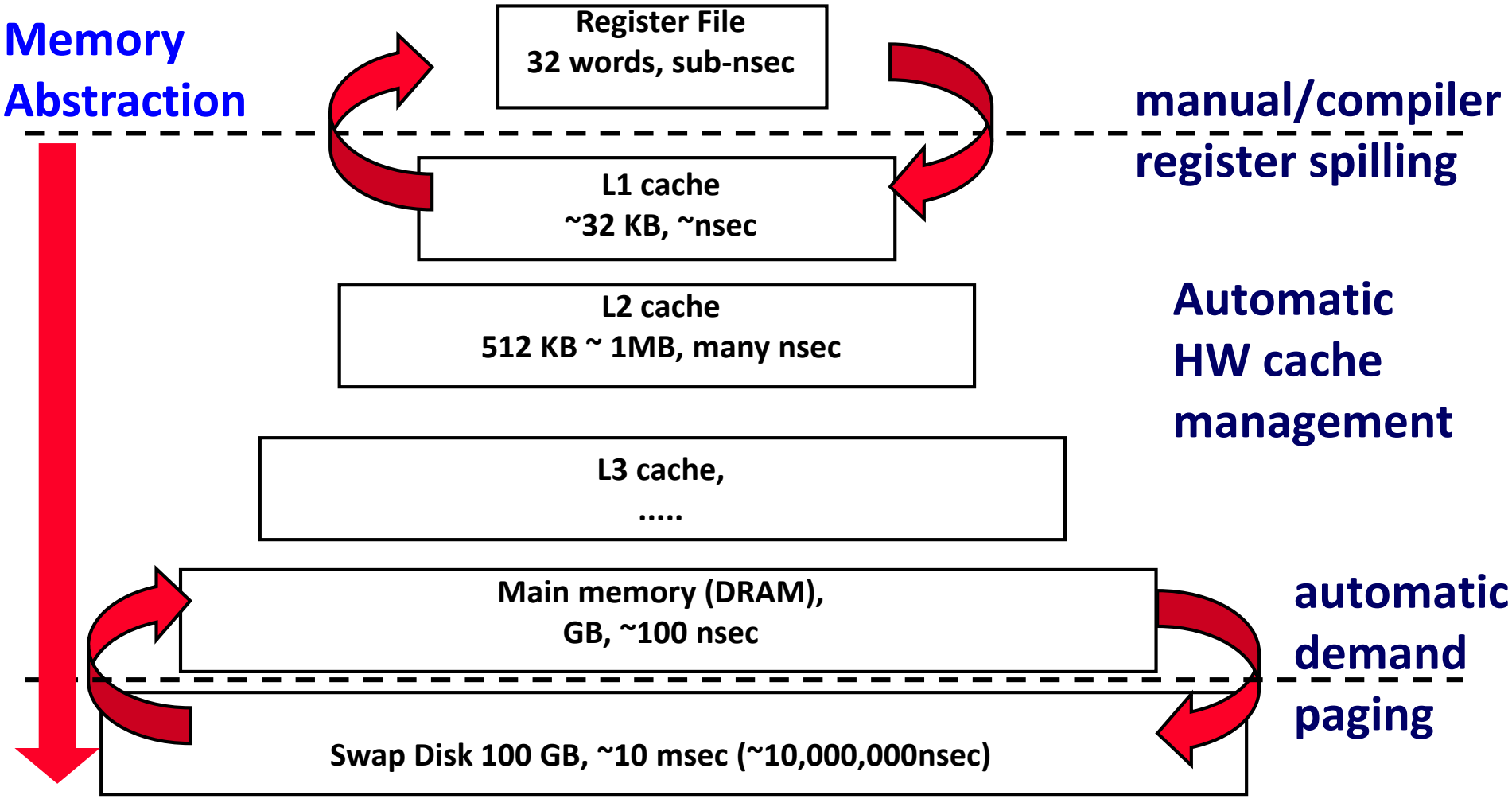
- Minimize processor stalls due to wait on memory
- Reduce average time to access memory



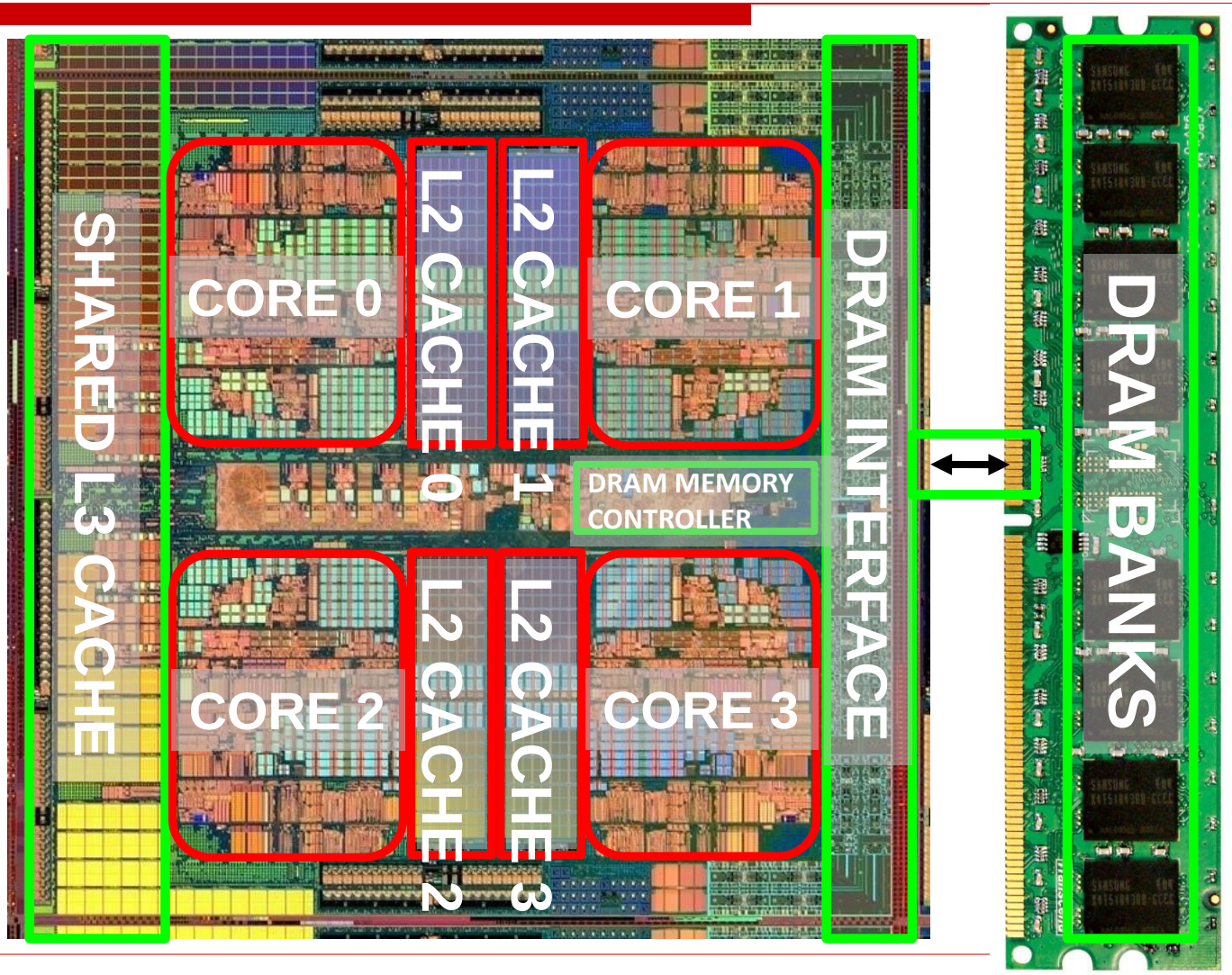
Memory Technology Design Space



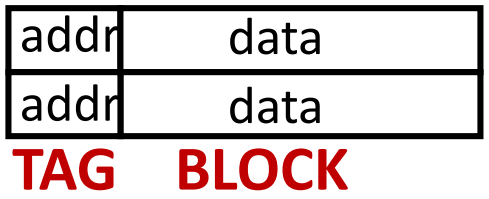
Memory in a Modern System



Memory in a Modern System



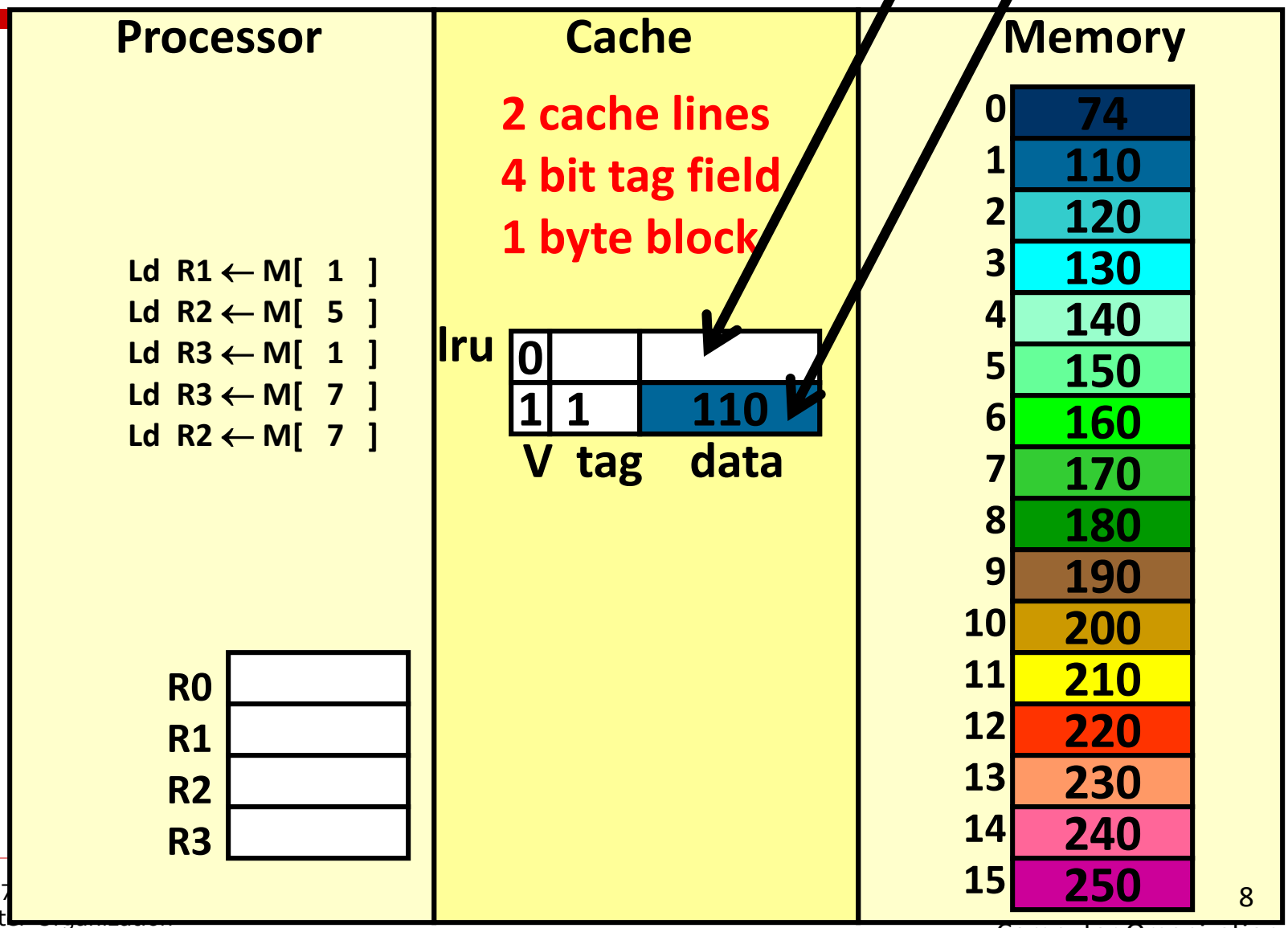
Review: Cache Organization



- A cache memory consists of multiple tag/block pairs (called cache lines)
  - Searches can be done in parallel (within reason)
  - At most one tag will match
- If there is a tag match, it is a cache HIT
- If there is no tag match, it is a cache MISS

Our goal is to keep the data we think will be accessed in the near future in the cache

Review: A Very Simple Memory System



LRU Implementation with counters

- **2 ways**
  - 1 bit per set to mark latest way accessed in set
  - Evict way not pointed by bit

- **k-way set associative LRU**
  - Requires full ordering of way accesses
  - Hold a $\log_2 k$ bit counter per line
  - When a way *i* is accessed

```
X = Counter[i]
Counter[i] = k-1
for (j = 0 to k-1)
    if ((j≠i) AND (Counter[j]>X)) Counter[j]--;
```

  - When replacement is needed
    - evict way with counter = 0
  - Expensive for even small k's

Initial State

Way	0	1	2	3
Count	0	1	2	3

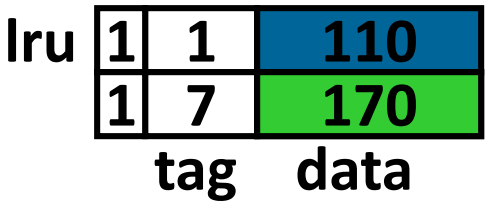
Access way 2

Way	0	1	2	3
Count	0	1	3	2

Access way 0

Way	0	1	2	3
Count	3	1	2	1

How can we reduce the overhead for each cache line?



- Cache bigger units than bytes
  - Each block has a single tag, and blocks can be whatever size we choose.

How increasing block size reduces overhead

Case 1:
Block size: **1** bytes

V	tag	data (block)
1	0	74
1	6	160

How many bits needed per tag?
= $\log_2$(number of blocks in memory) = $\log_2(16)$
= 4 bits

Case 2:
Block size: **2** bytes

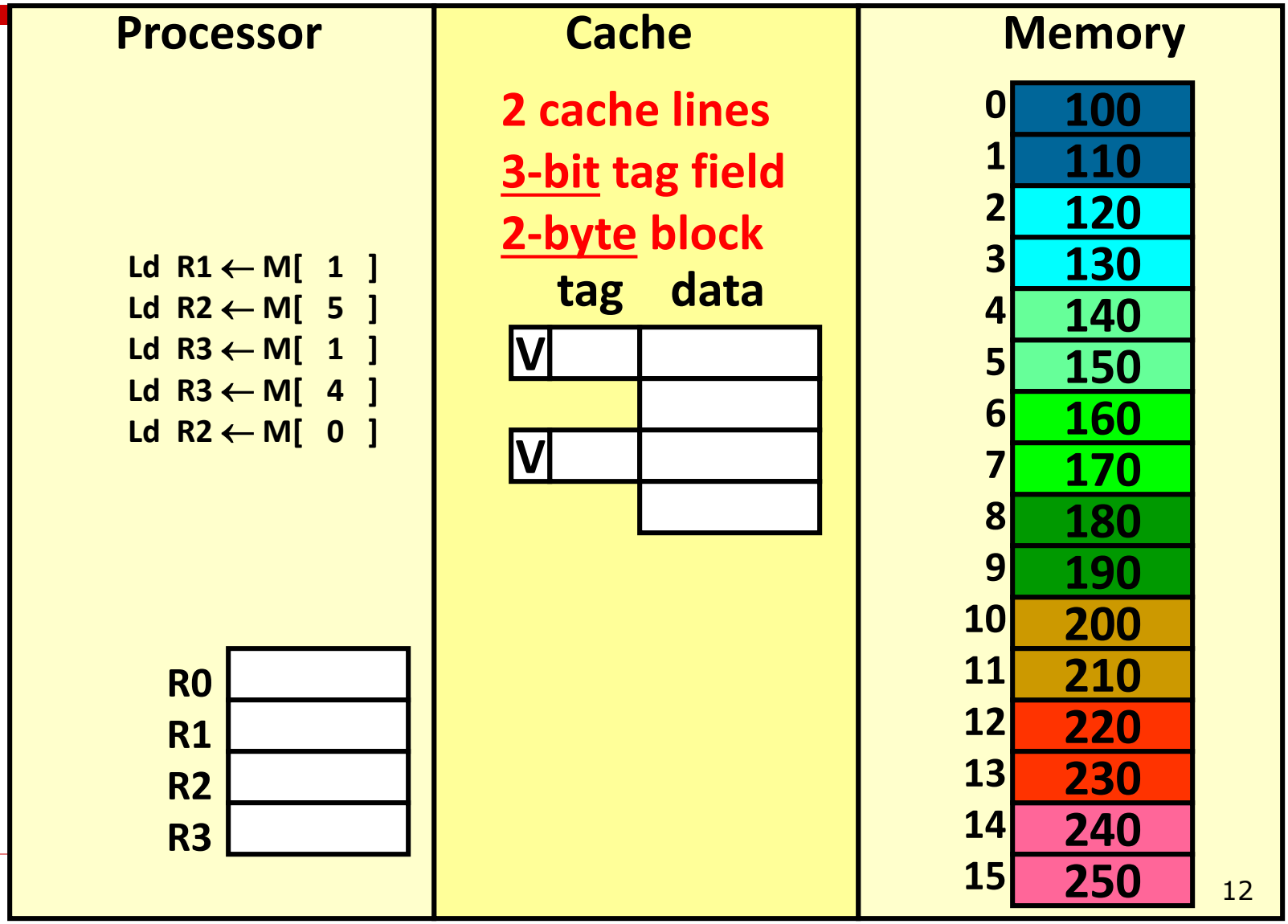
V	tag	data (block)	
1	0	74	110
1	3	160	170

How many bits needed per tag?
= $\log_2$(number of blocks in memory) = $\log_2(8)$
= 3 bits

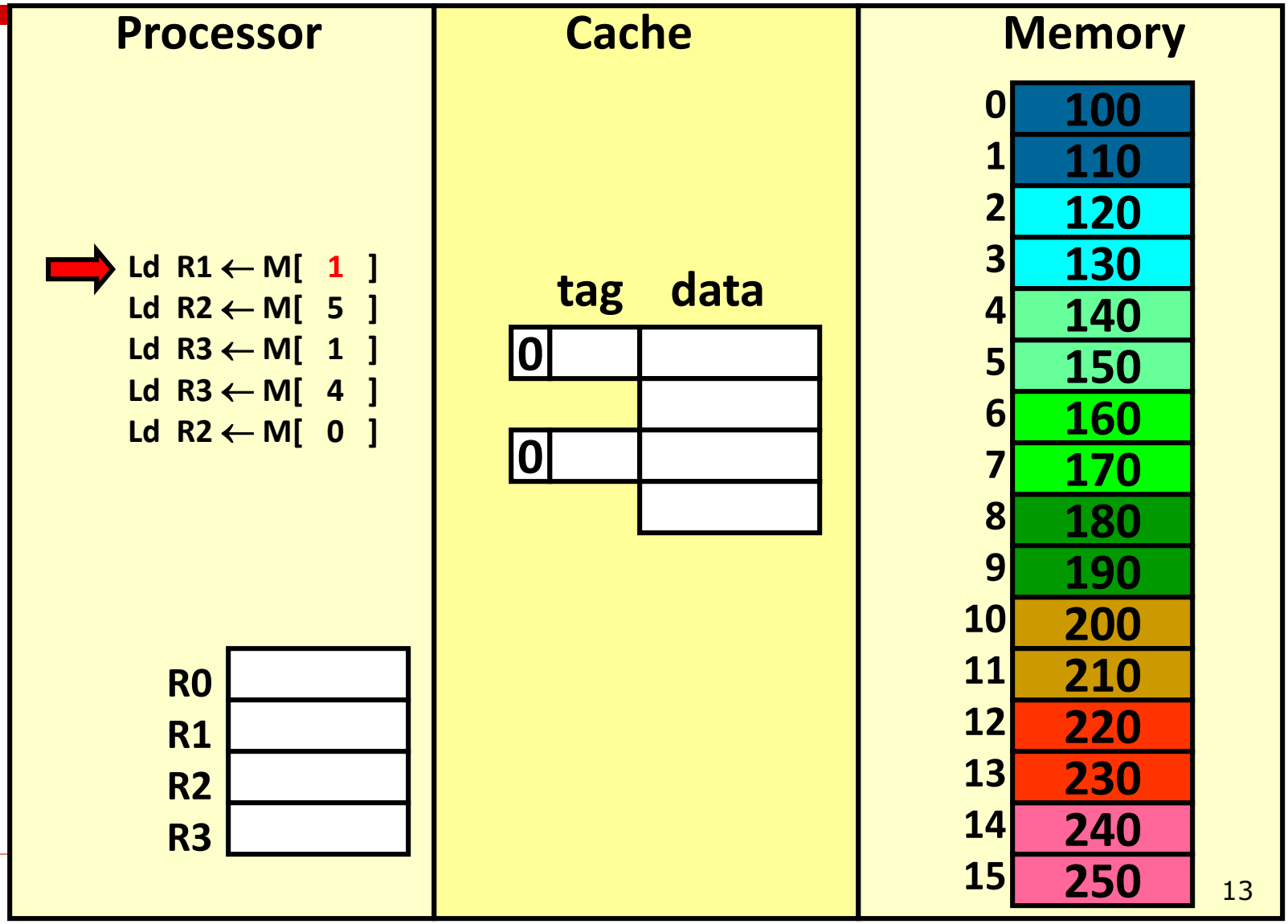
Keep more data per cache line

	Memory	Block Case1	Block Case2
0	74	0	0
1	110	1	0
2	120	2	1
3	130	3	1
4	140	4	2
5	150	5	2
6	160	6	3
7	170	7	3
8	180	8	4
9	190	9	4
10	200	10	5
11	210	11	5
12	220	12	6
13	230	13	6
14	240	14	7
15	250	15	7

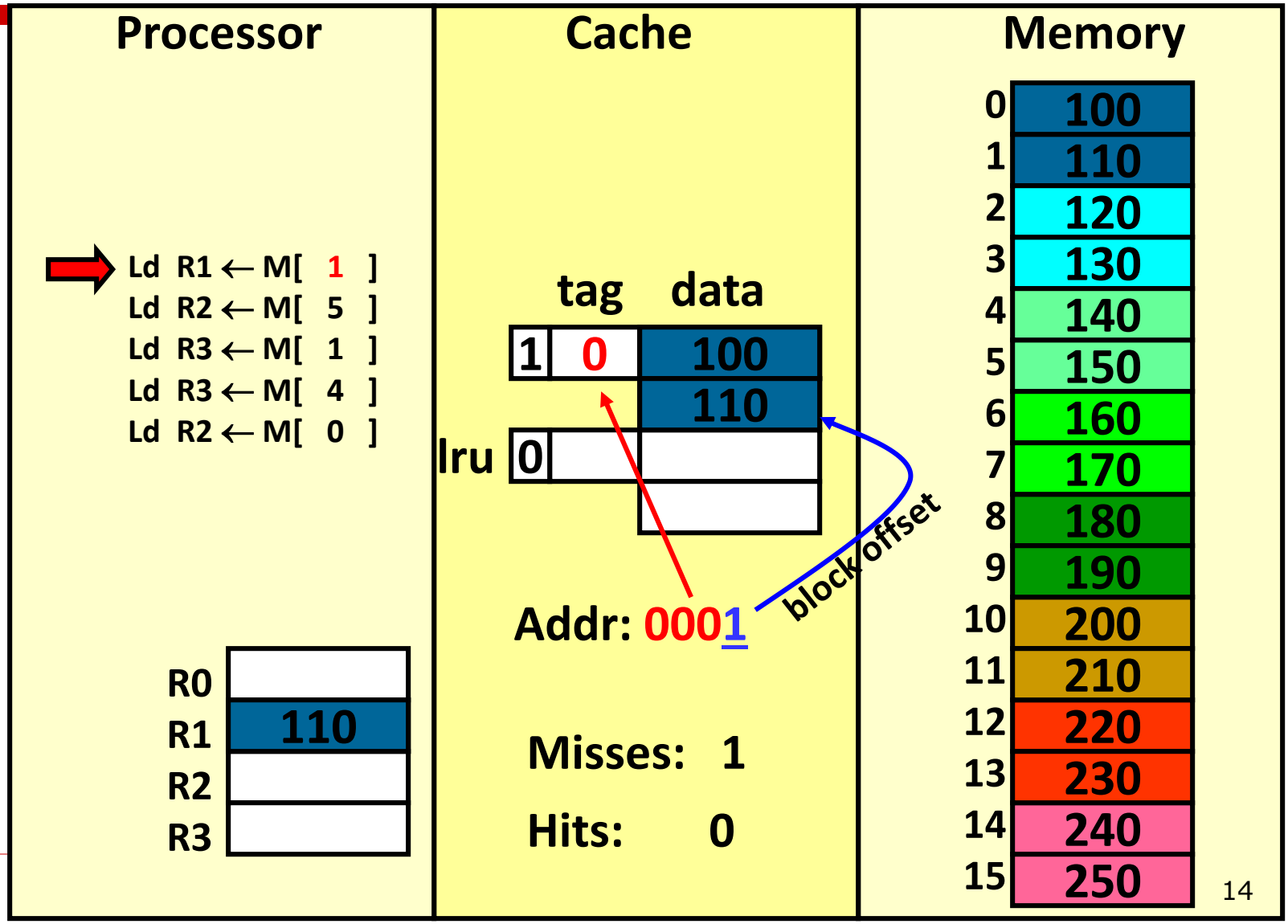
Block size for caches



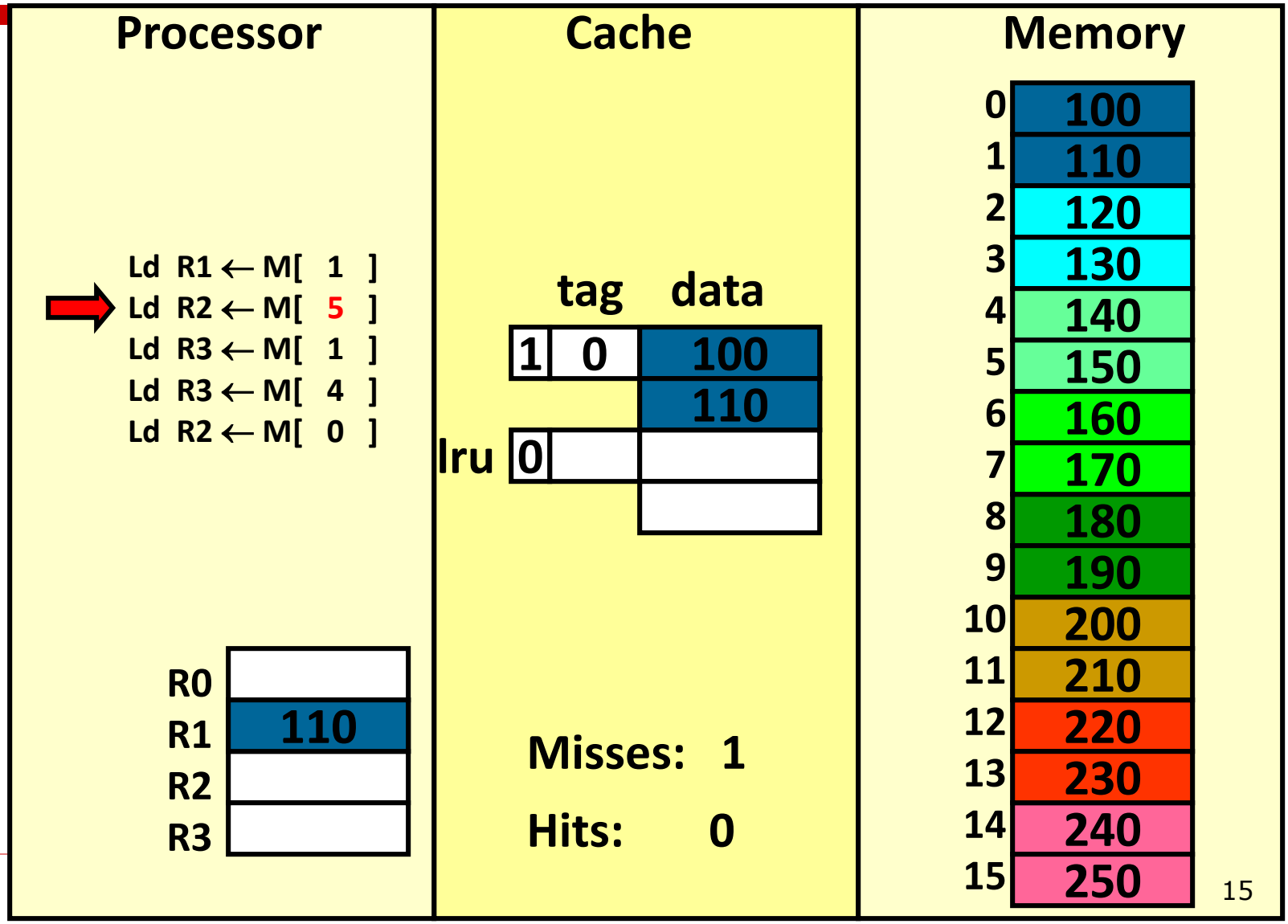
Block size for caches



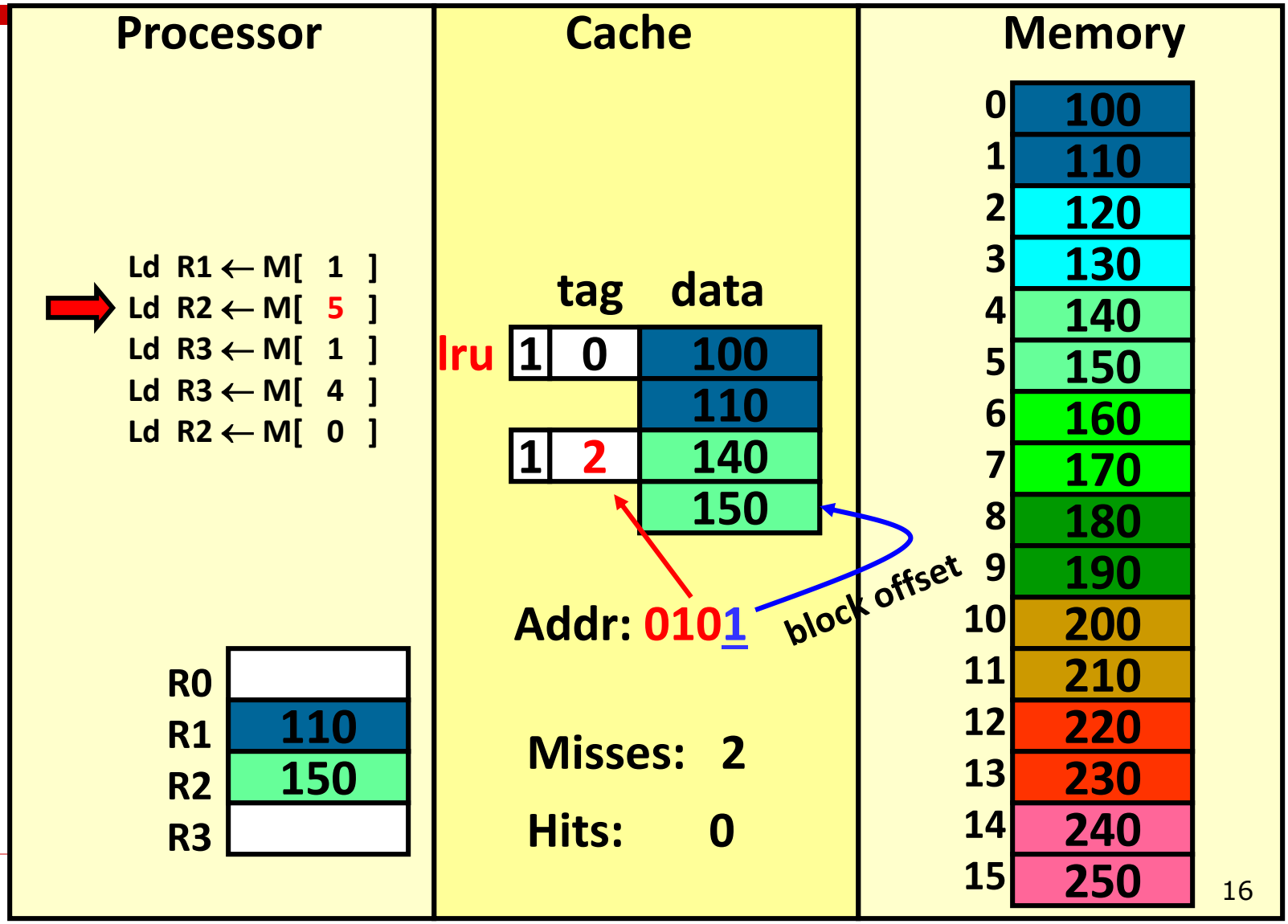
Block size for caches



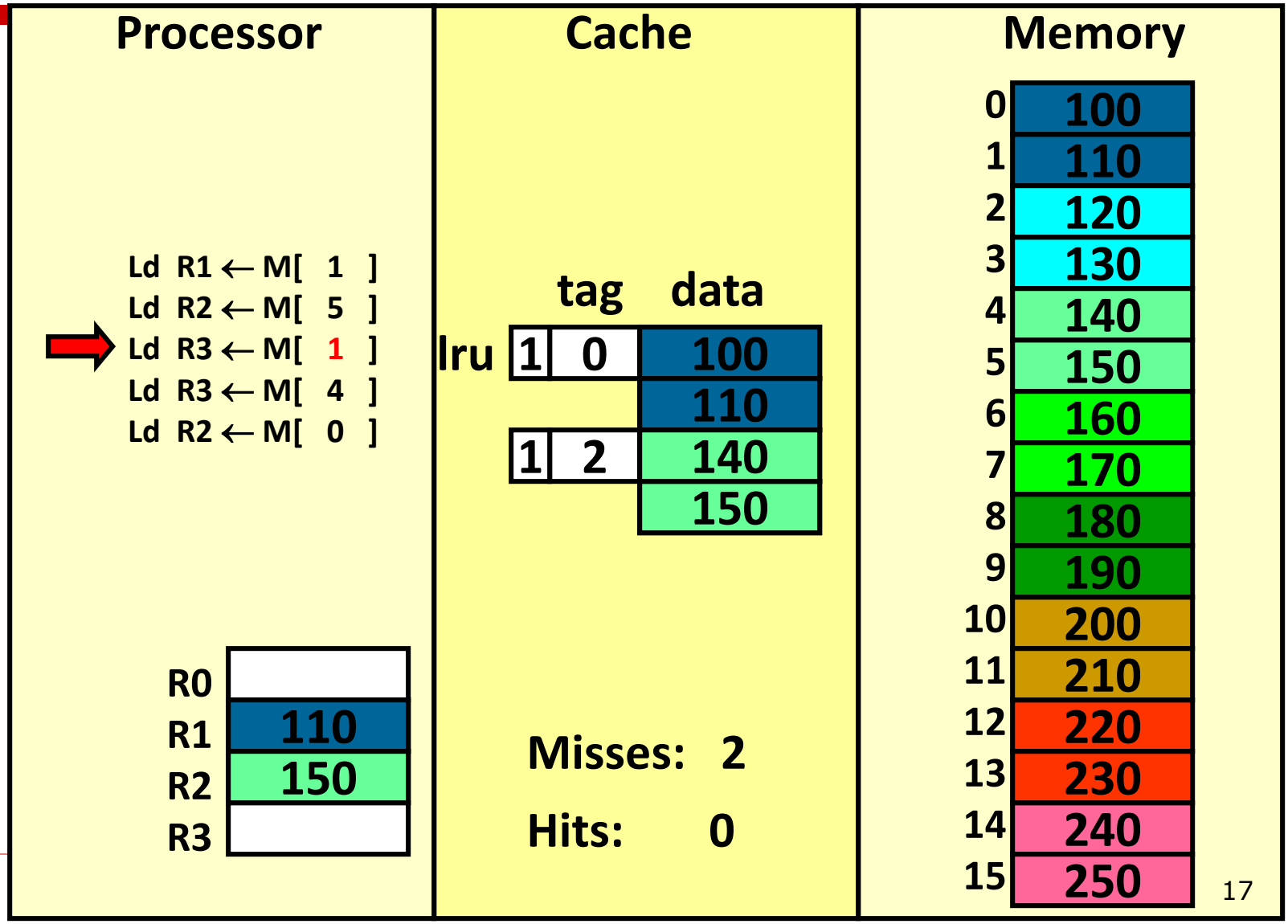
Block size for caches



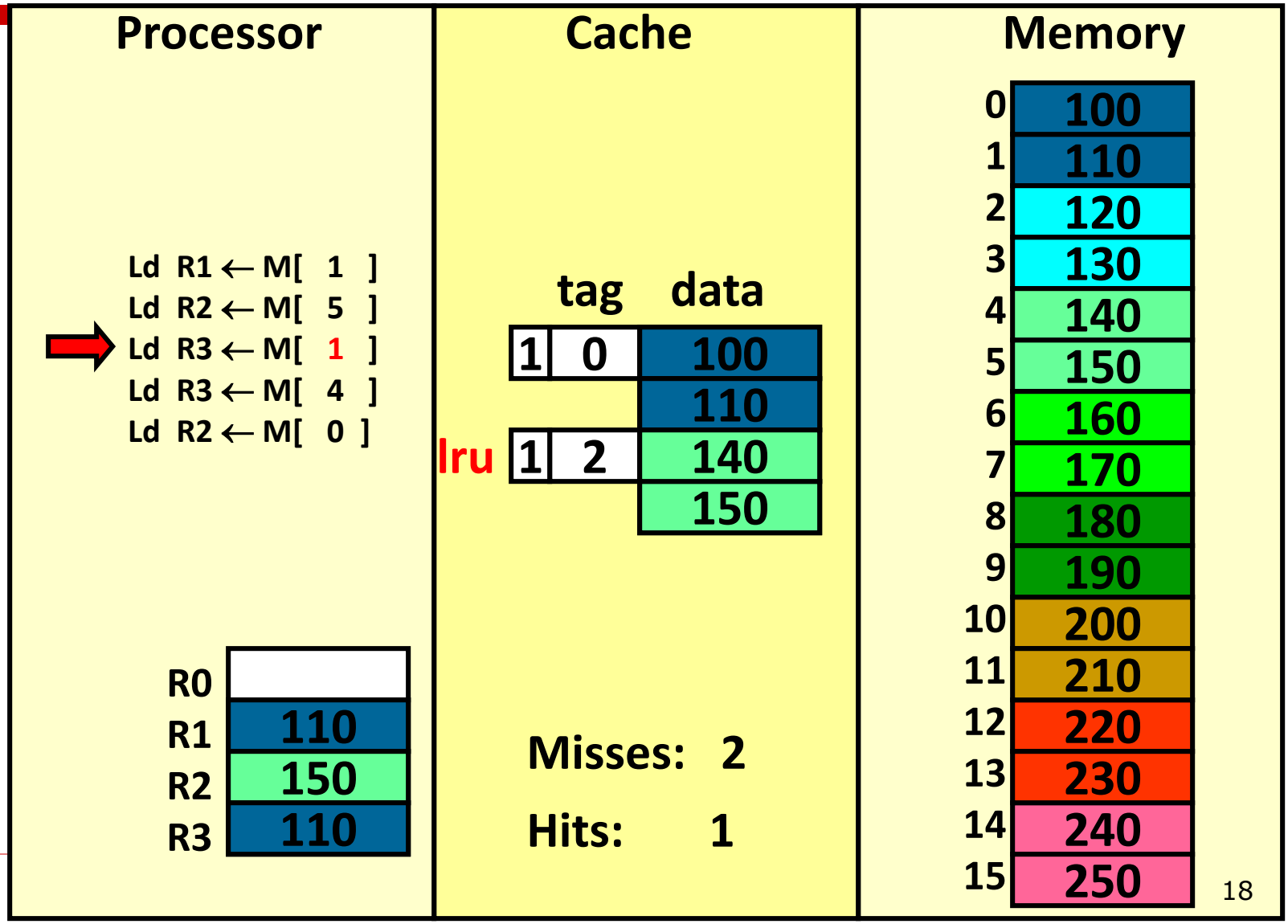
Block size for caches



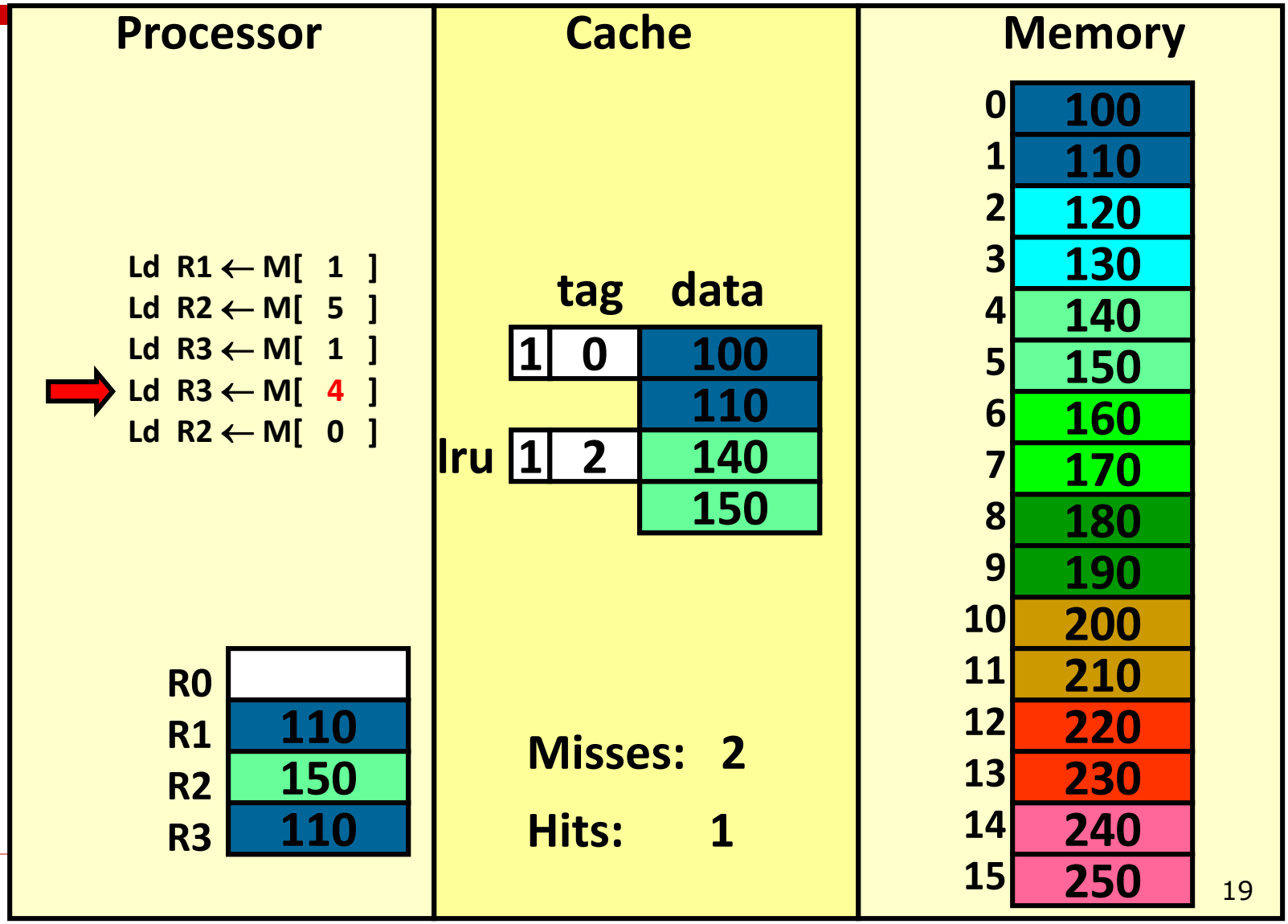
Block size for caches



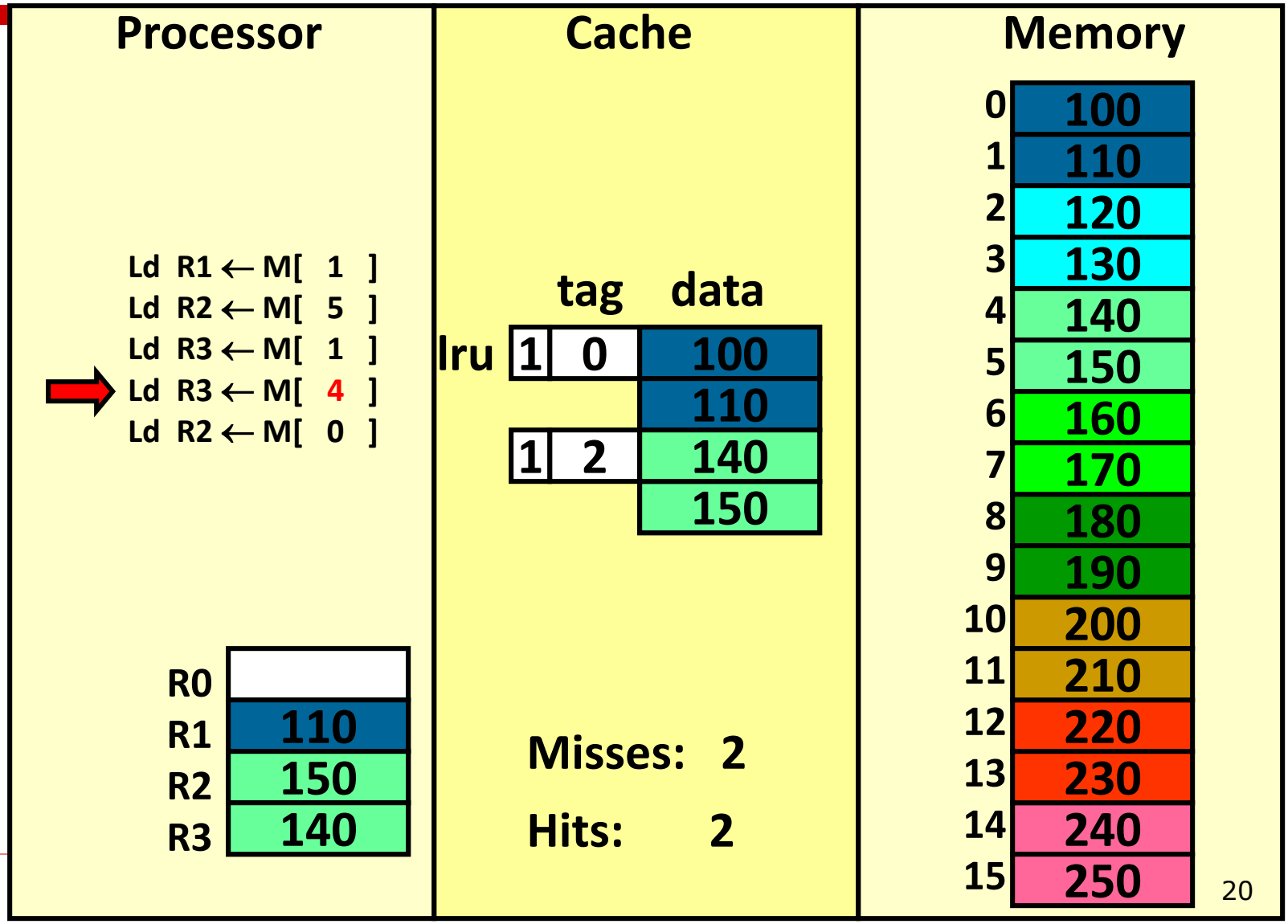
Block size for caches



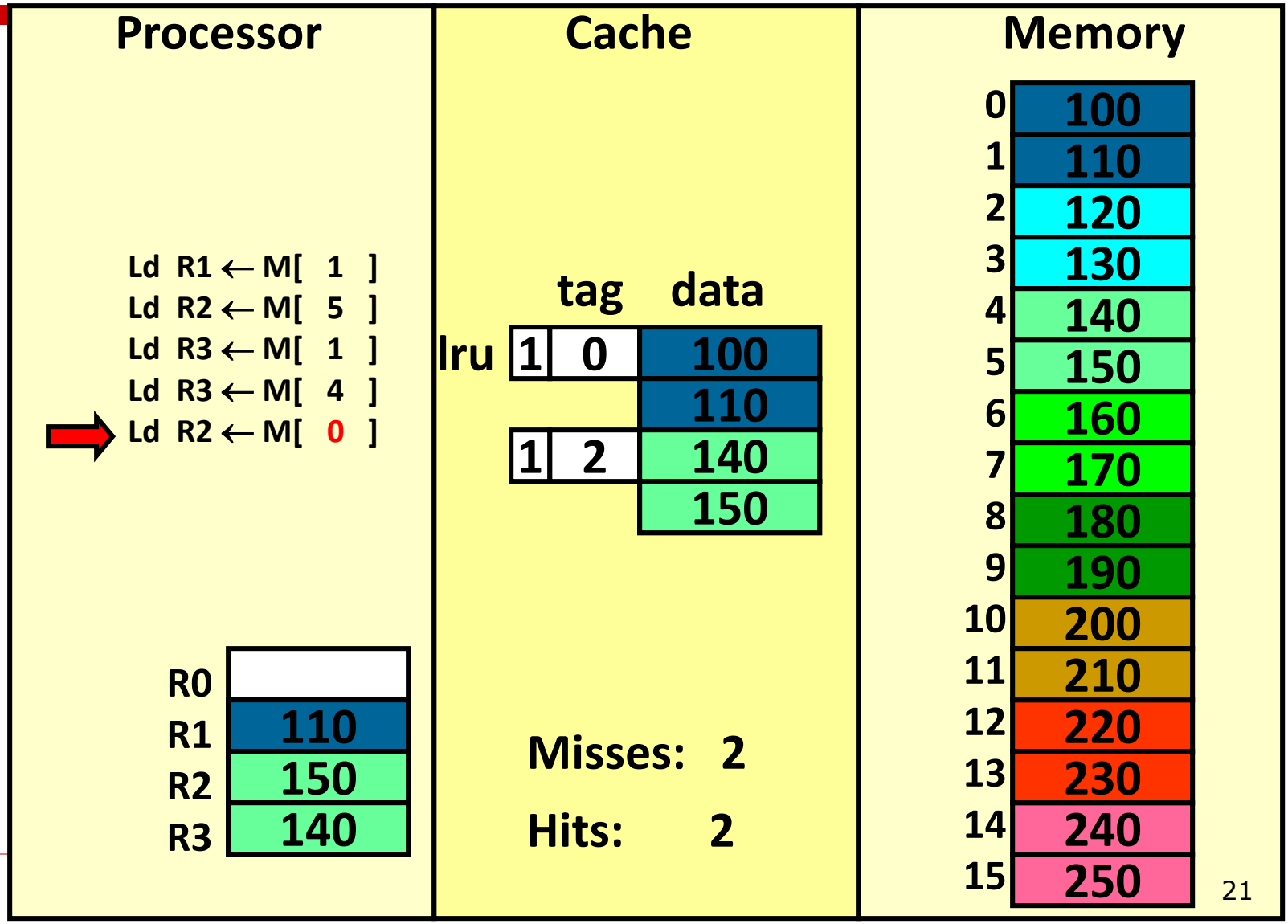
Block size for caches



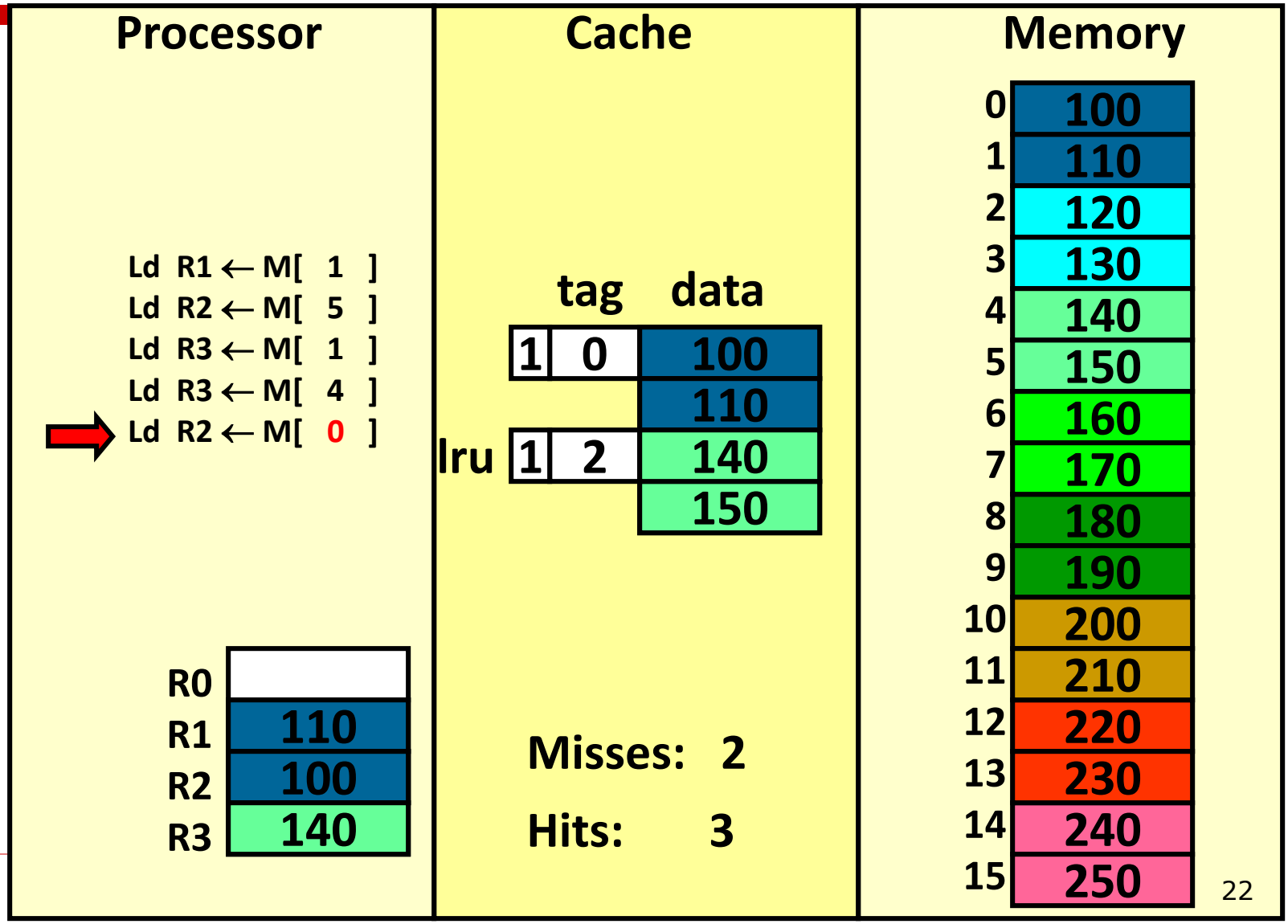
Block size for caches



Block size for caches



Block size for caches

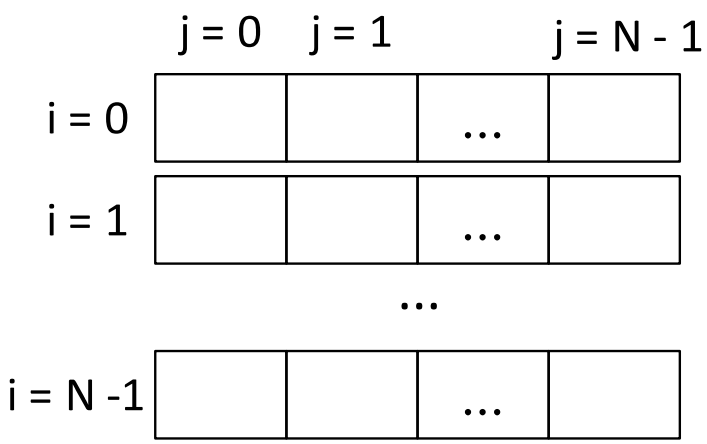


Spatial Locality

- Notice that when we accessed address 1, we also brought in address 0.
  - This turned out to be a good thing since we later referenced address 0 and found it in the cache.
- Spatial locality in a program says that if we reference a memory location (e.g., 1000), we are more likely to reference a location near it (e.g. 1001) than some random location.

Spatial Locality

- *Observation*: Applications access data near to what they just accessed



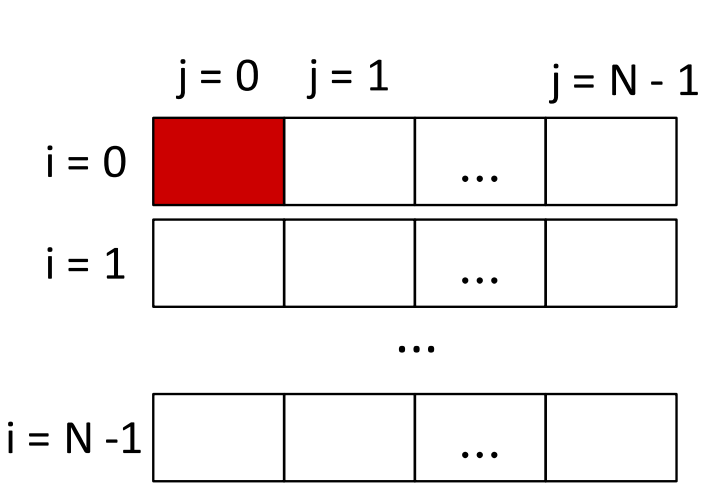
Access sequence

i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 0, j = 3
...

```
for(i=0; i< N; i++)
  for(j = 0; j < N; j++ )
  {
      count++;
      arrayInt[i][j] = 10;
  }
```

Spatial Locality

- *Observation*: Applications access data near to what they just accessed



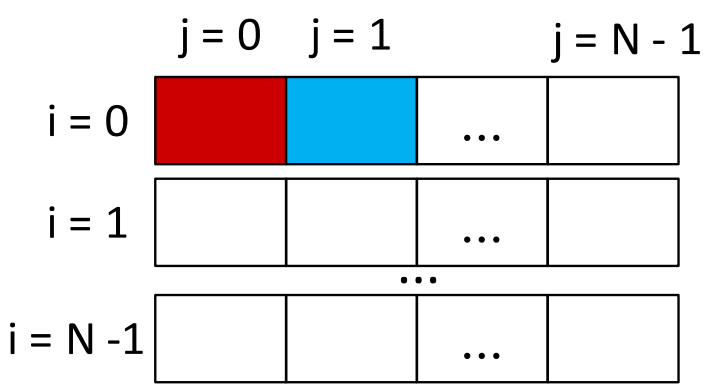
Access sequence

i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 0, j = 3
...

```
for(i=0; i< N; i++)
  for(j = 0; j < N; j++ )
  {
      count++;
      arrayInt[i][j] = 10;
  }
```

Spatial Locality

- *Observation*: Applications access data near to what they just accessed



Access sequence

i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 0, j = 3
...

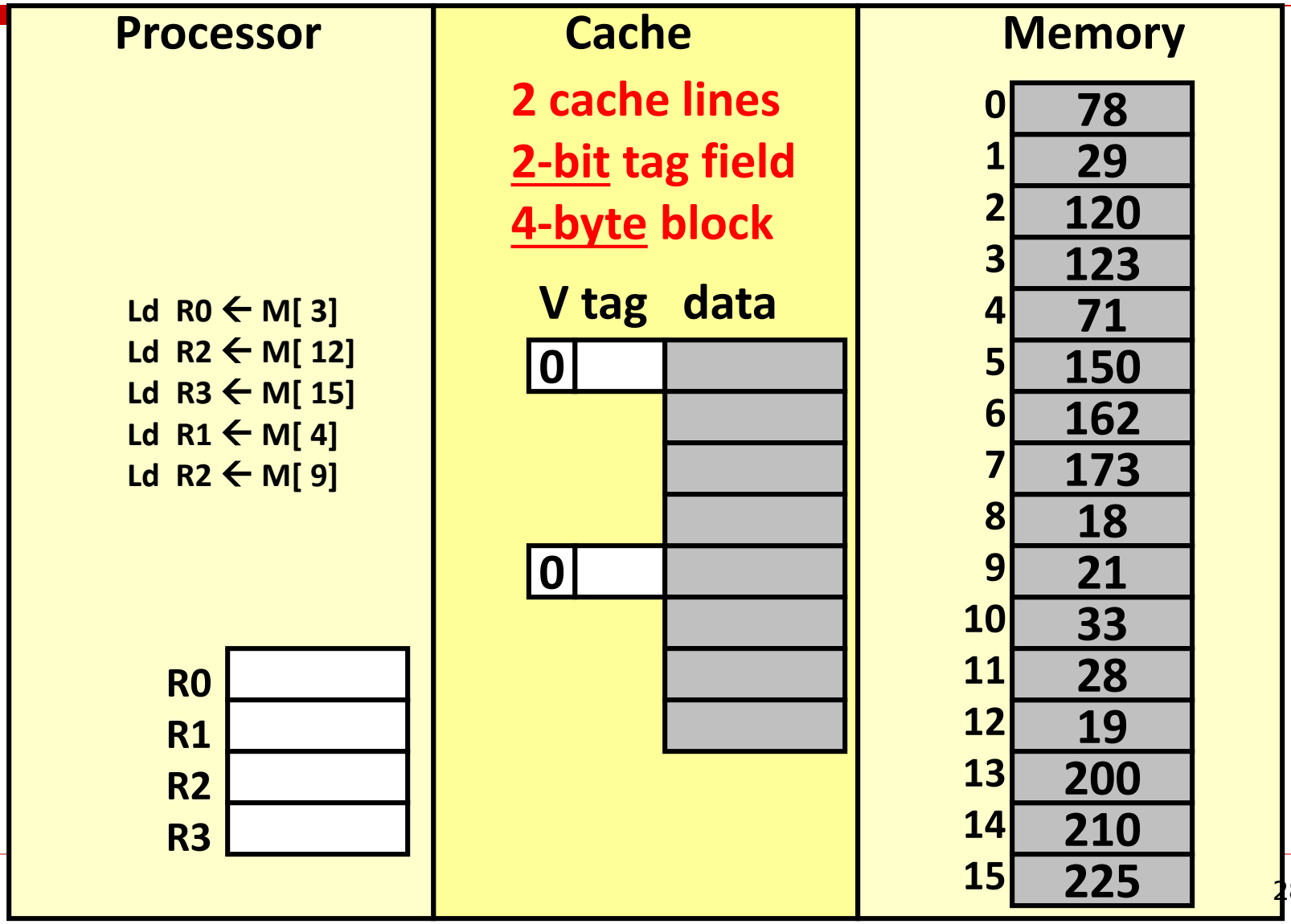
```
for(i=0; i< N; i++)
  for(j = 0; j < N; j++ )
  {
       count++;
       arrayInt[i][j] = 10;
  }
```

Applications have ***SPATIAL LOCALITY***

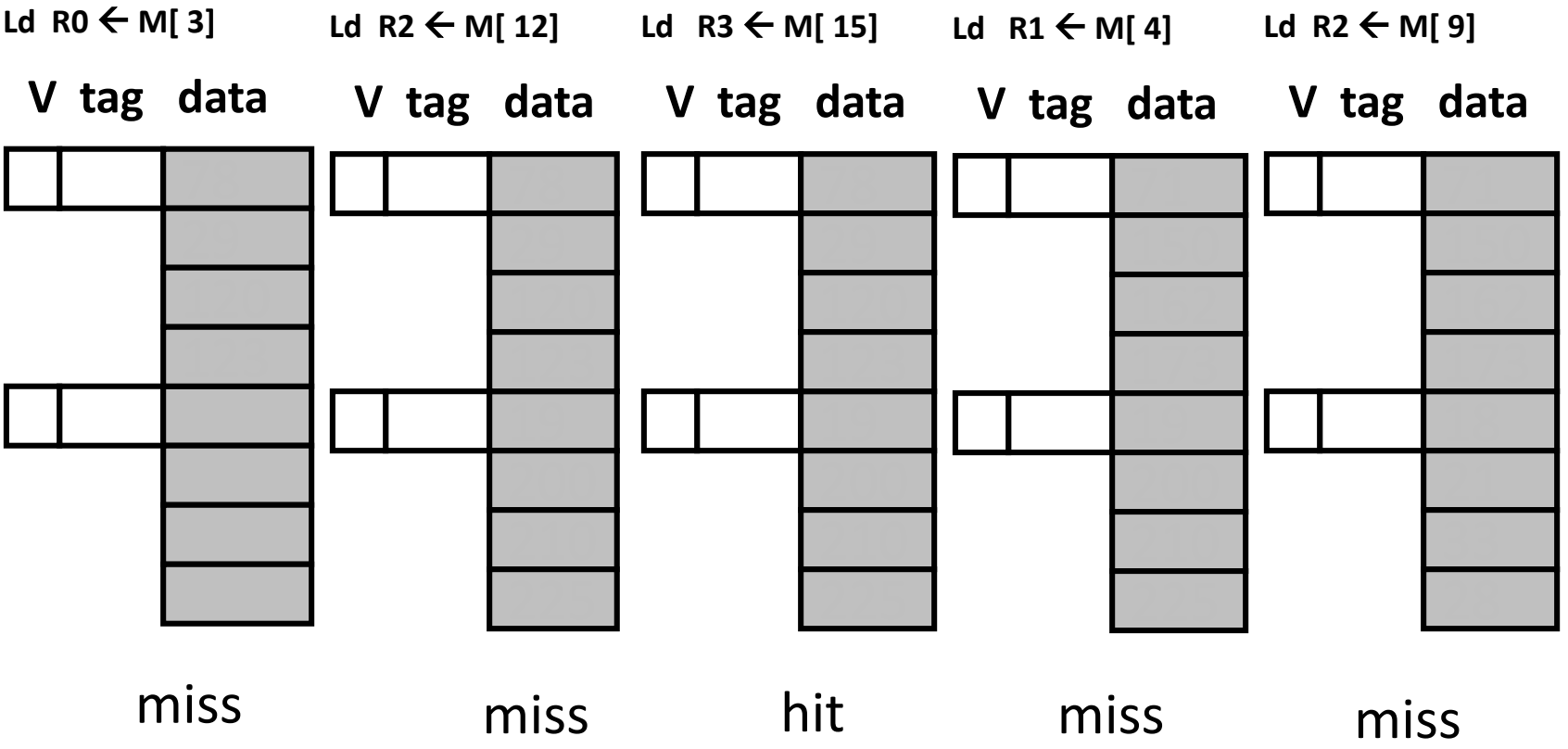
Basic Cache organization

- Decide on the block size
  - How? Simulate lots of different block sizes and see which one gives the best performance
  - Most systems use a block size between 32 bytes and 128 bytes
  - Longer sizes reduce the overhead by:
    - Reducing the number of CAM entries
    - Reducing the size of each CAM entry

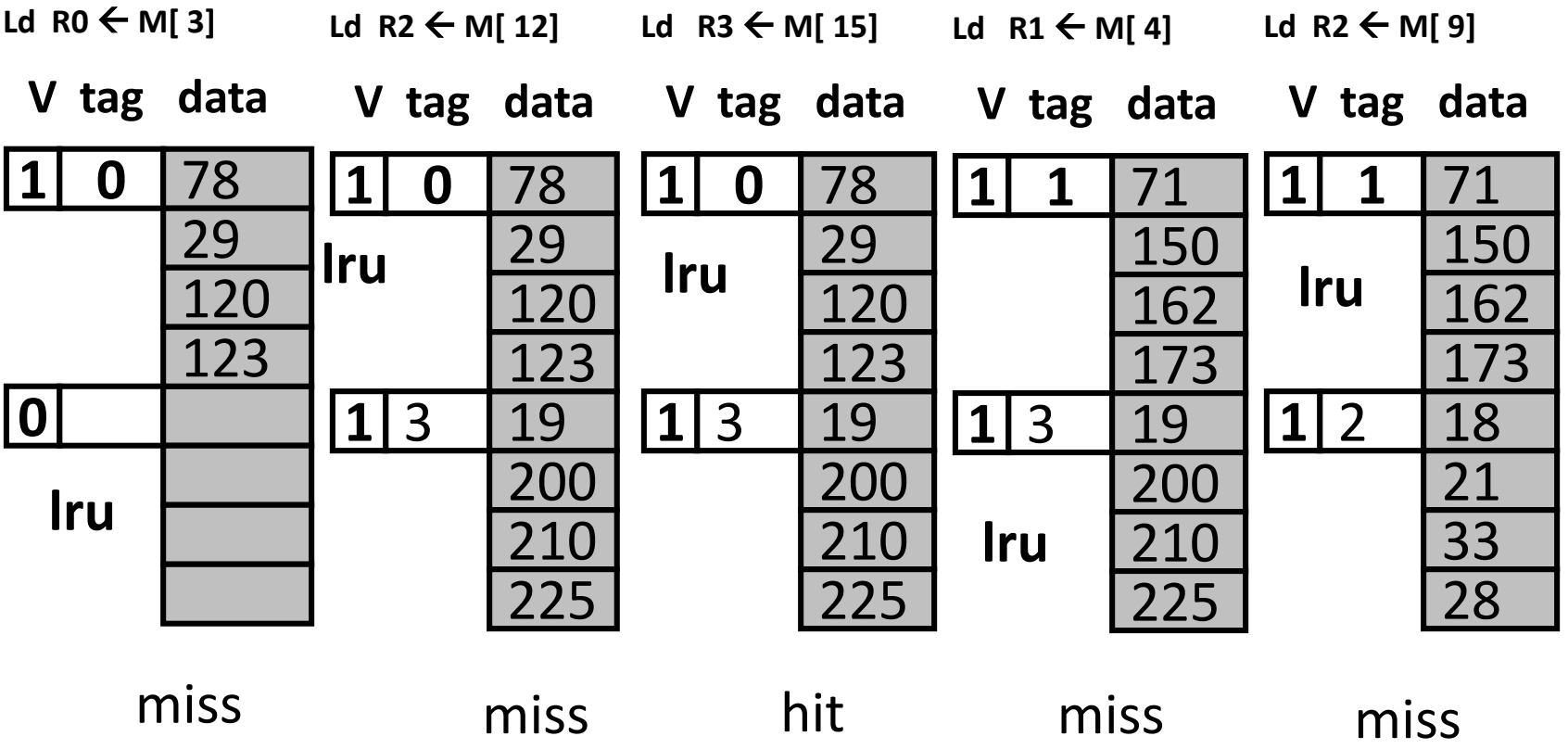
Practice Problem– Compute the register and cache state after executing the instruction sequence



Solution to Practice Problem



Solution to Practice Problem



Class Problem

- Given a cache with the following configuration: total size is 8 bytes, block size is 2 bytes*, fully associative*, LRU replacement. The memory address size is 16 bits and is byte addressable.
  1. How many bits are for each tag? How many blocks in the cache?
  2. For the following reference stream, indicate whether each reference is a hit or miss: 0, 1, 3, 5, 12, 1, 2, 9, 4
  3. What is the hit rate?
  4. How many bits are needed for storage overhead for each block?

Class Problem

- Given a cache with the following configuration: total size is 8 bytes, block size is 2 bytes*, fully associative*, LRU replacement. The memory address size is 16 bits and is byte addressable.

1. How many bits are for each tag? How many blocks in the cache?

   Tag = 16 – 1 (block offset) = 15 bits;
   2 byte blocks, 8 bytes total = 4 blocks.

2. For the following reference stream, indicate whether each reference is a hit or miss: 0, 1, 3, 5, 12, 1, 2, 9, 4

   M,H,M,M,M,H,H,M,M

3. What is the hit rate? 3/9 = 33 %

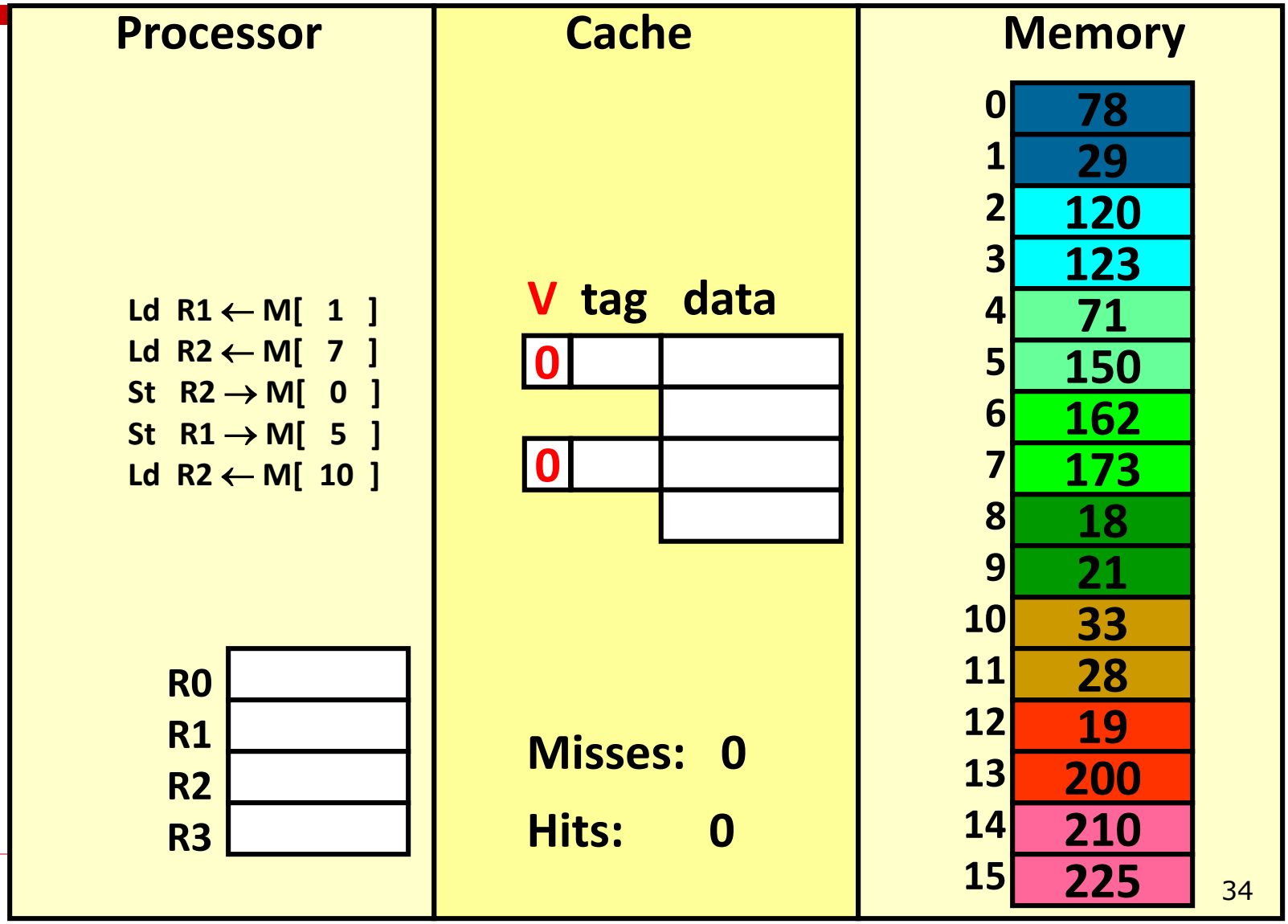
4. How many bits are needed for storage overhead for each block?

   Overhead = 15 (Tag) + 1 (V) + 2 (LRU) = 18 bits

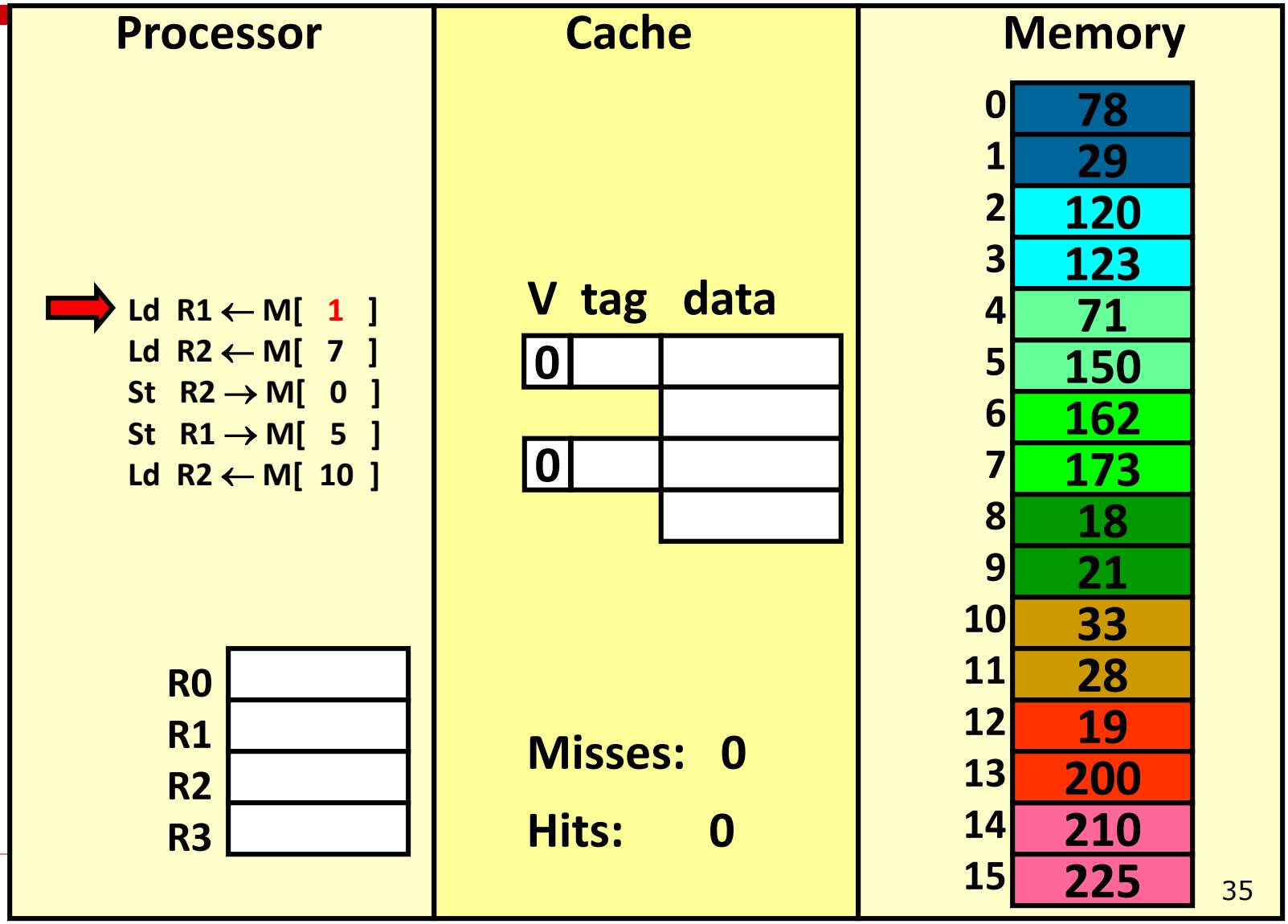
What about stores?

- 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)

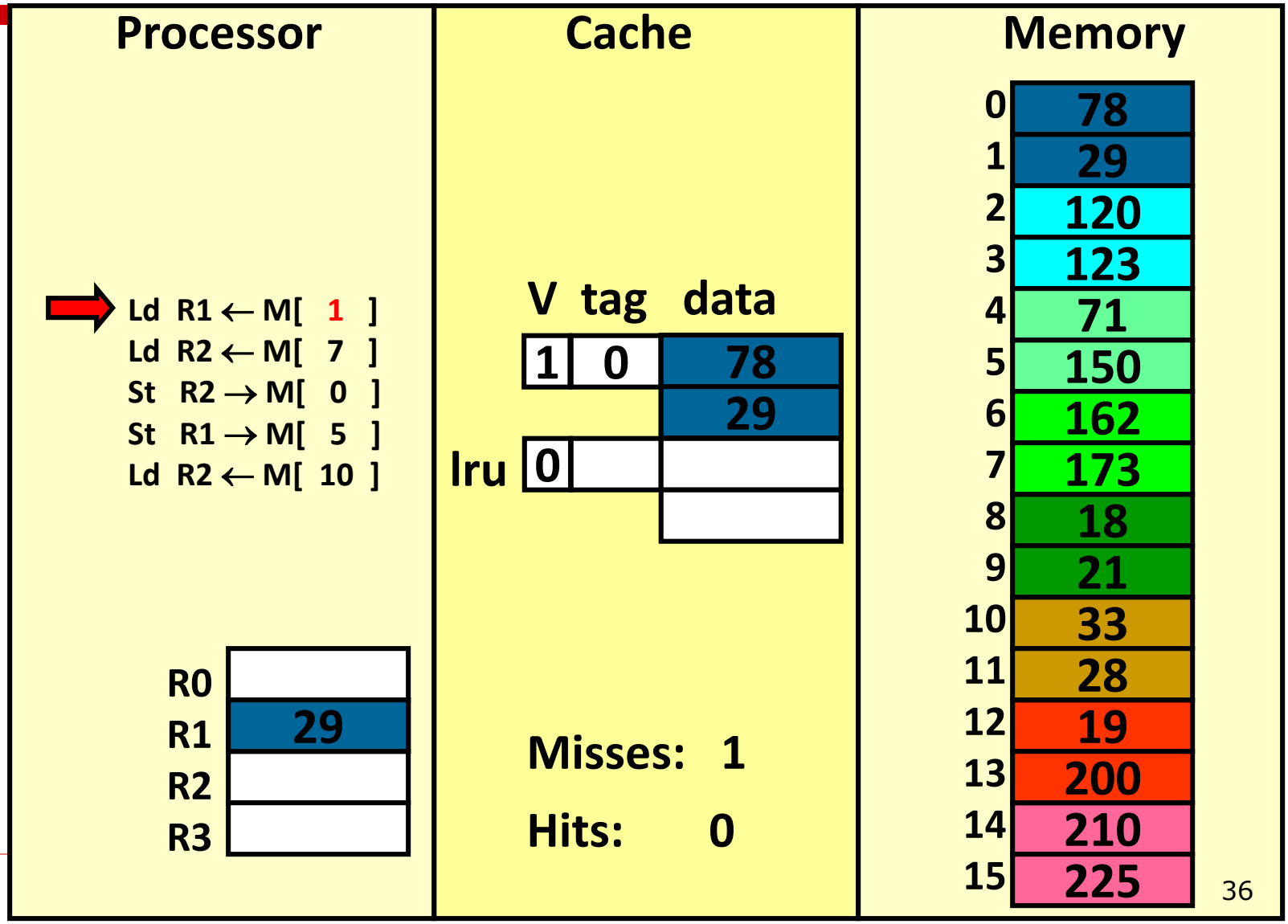
Handling stores (write-through)



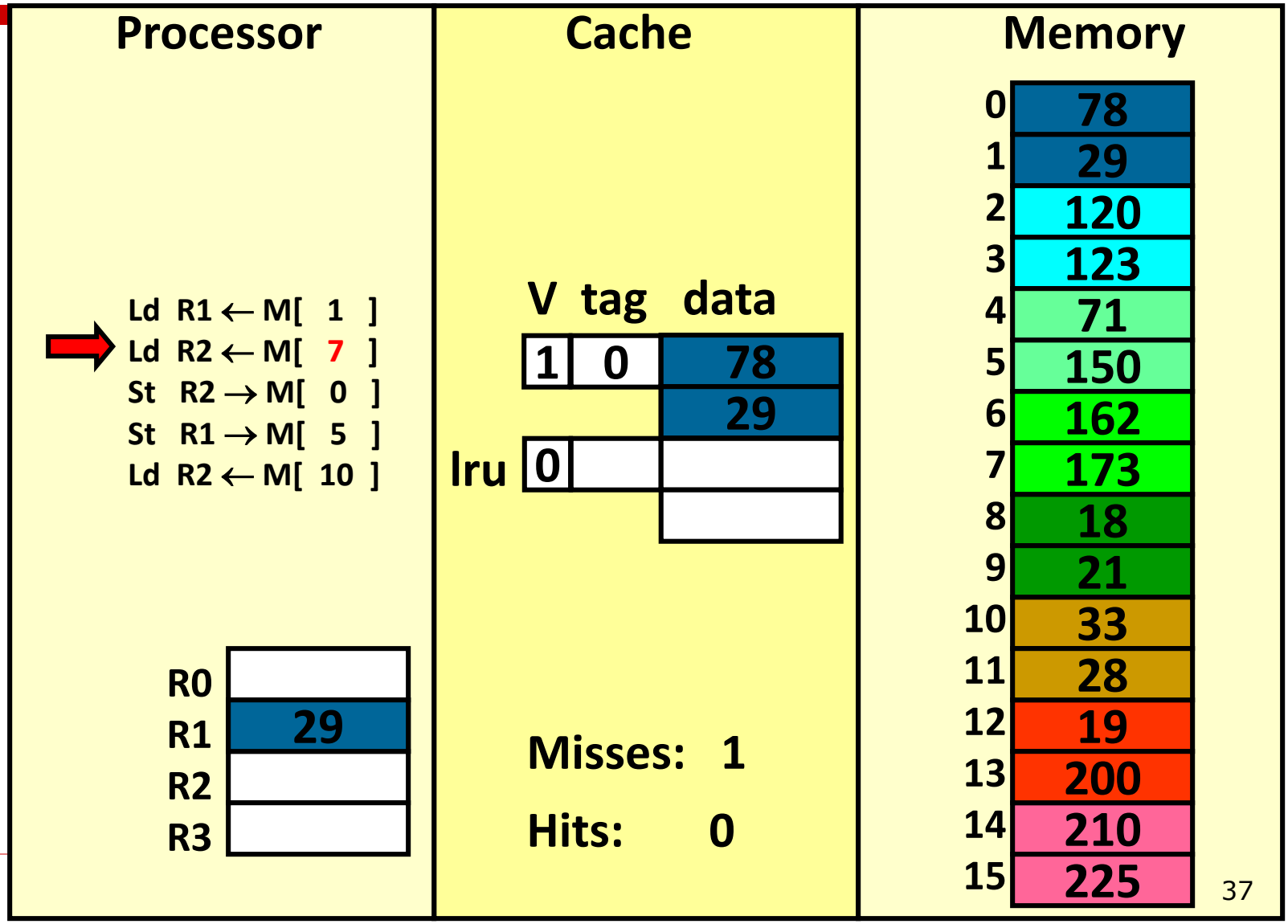
write-through (REF 1)



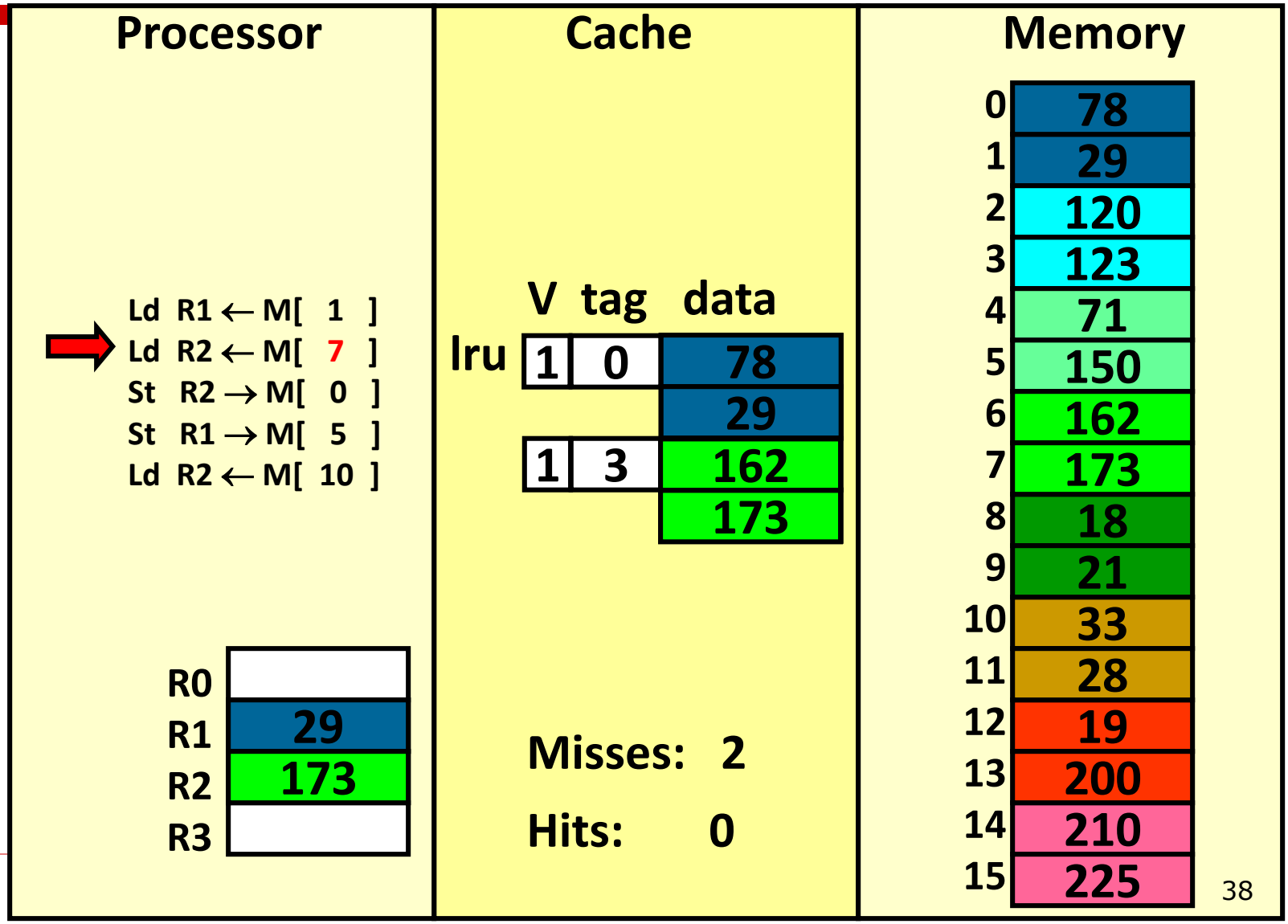
write-through (REF 1)



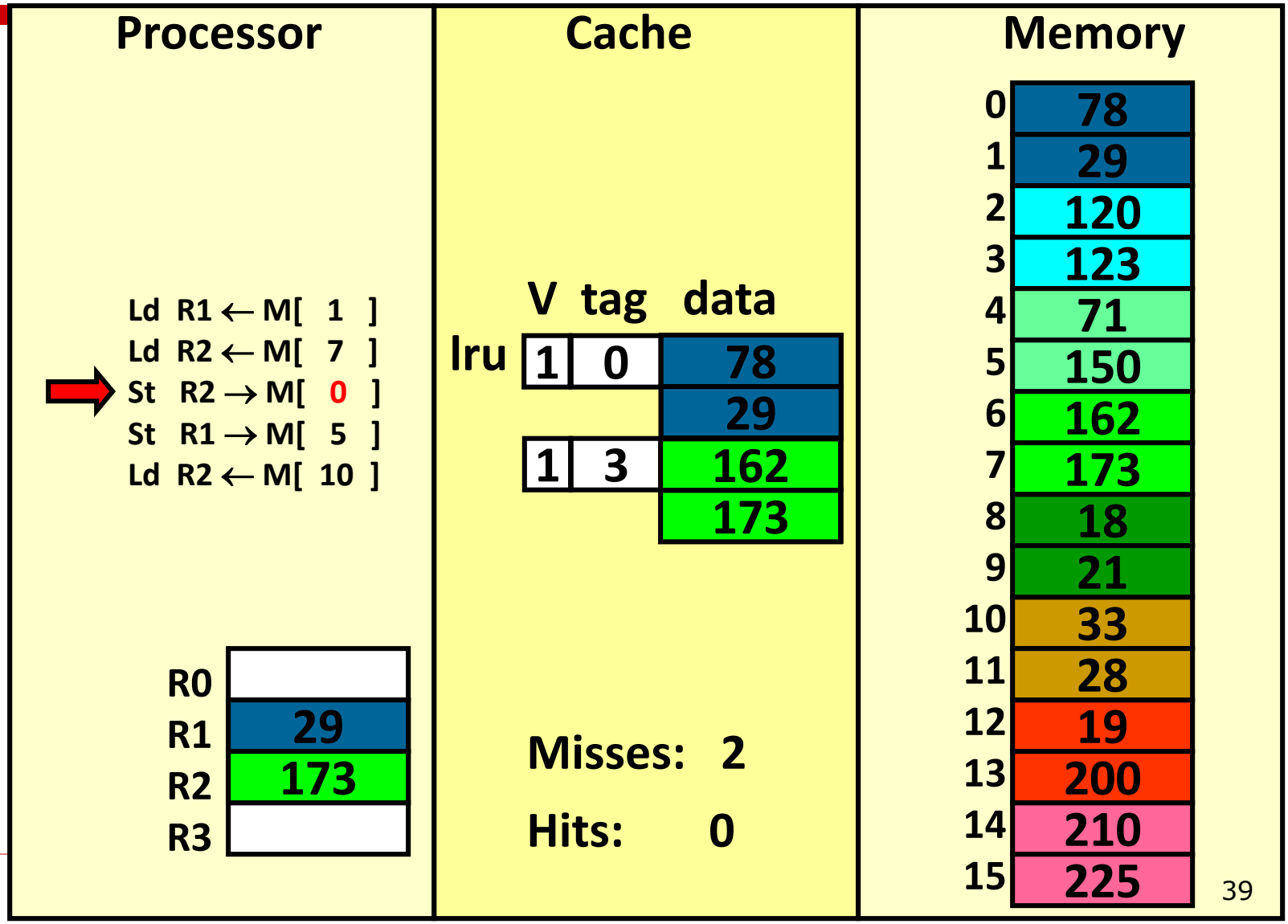
write-through (REF 2)



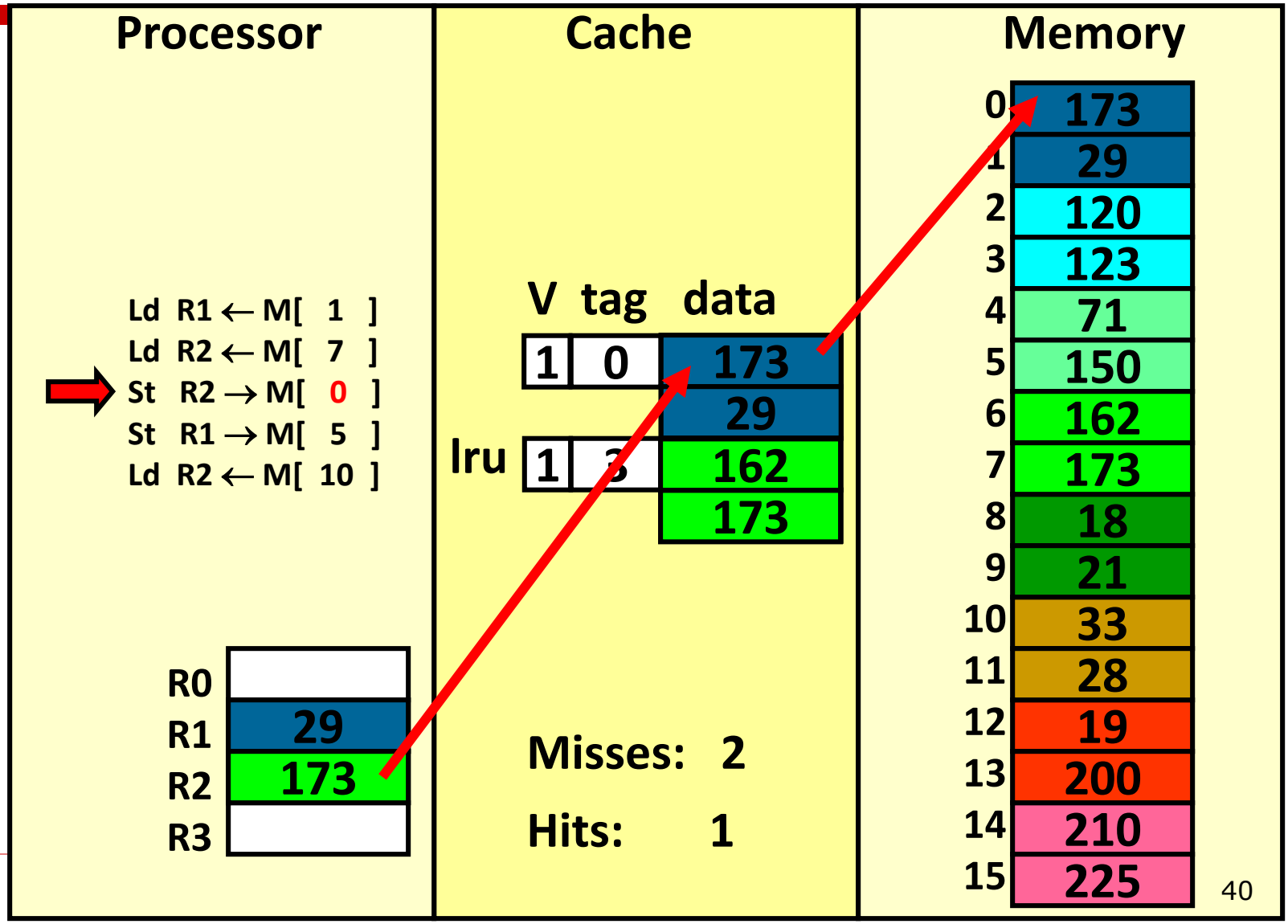
write-through (REF 2)



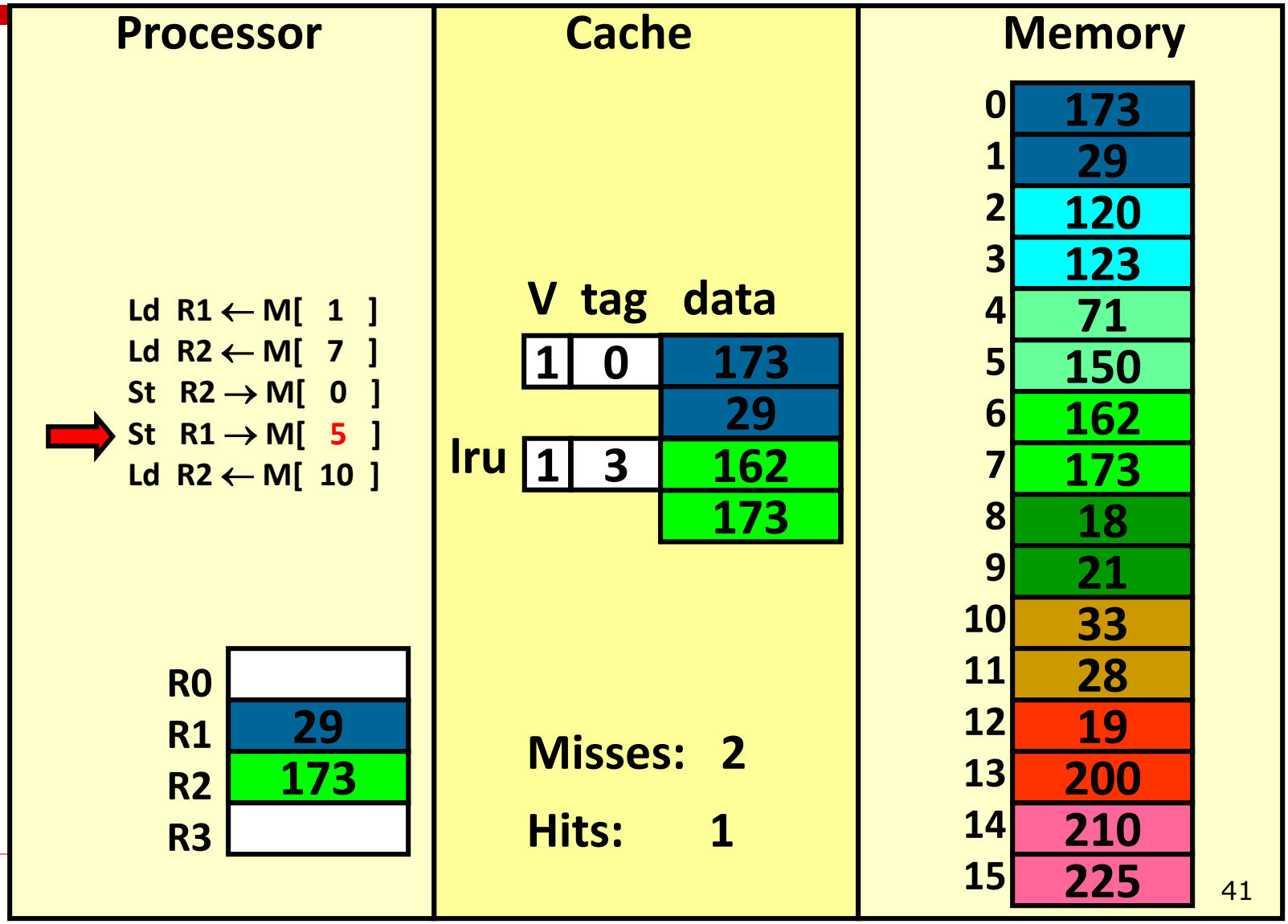
write-through (REF 3)



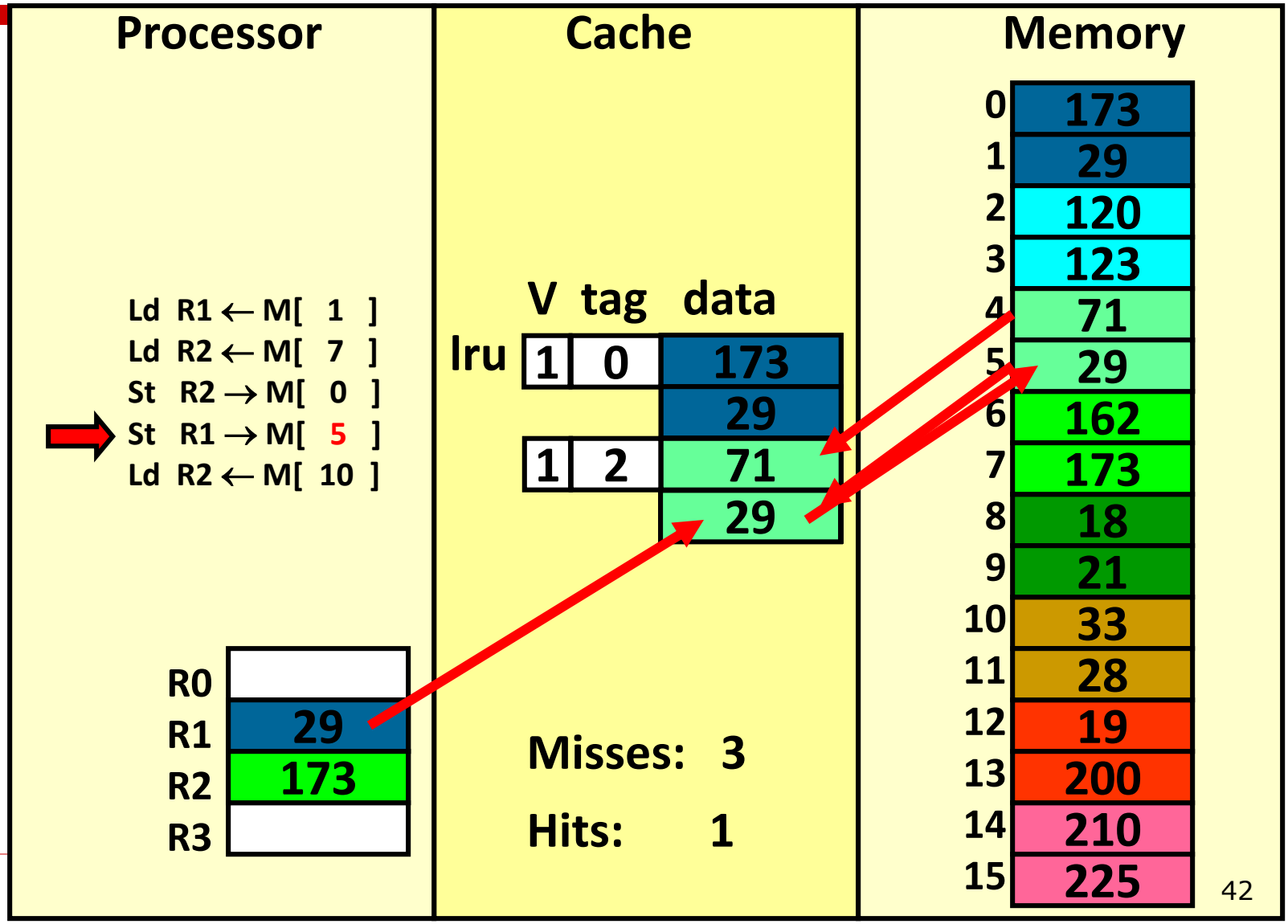
write-through (REF 3)



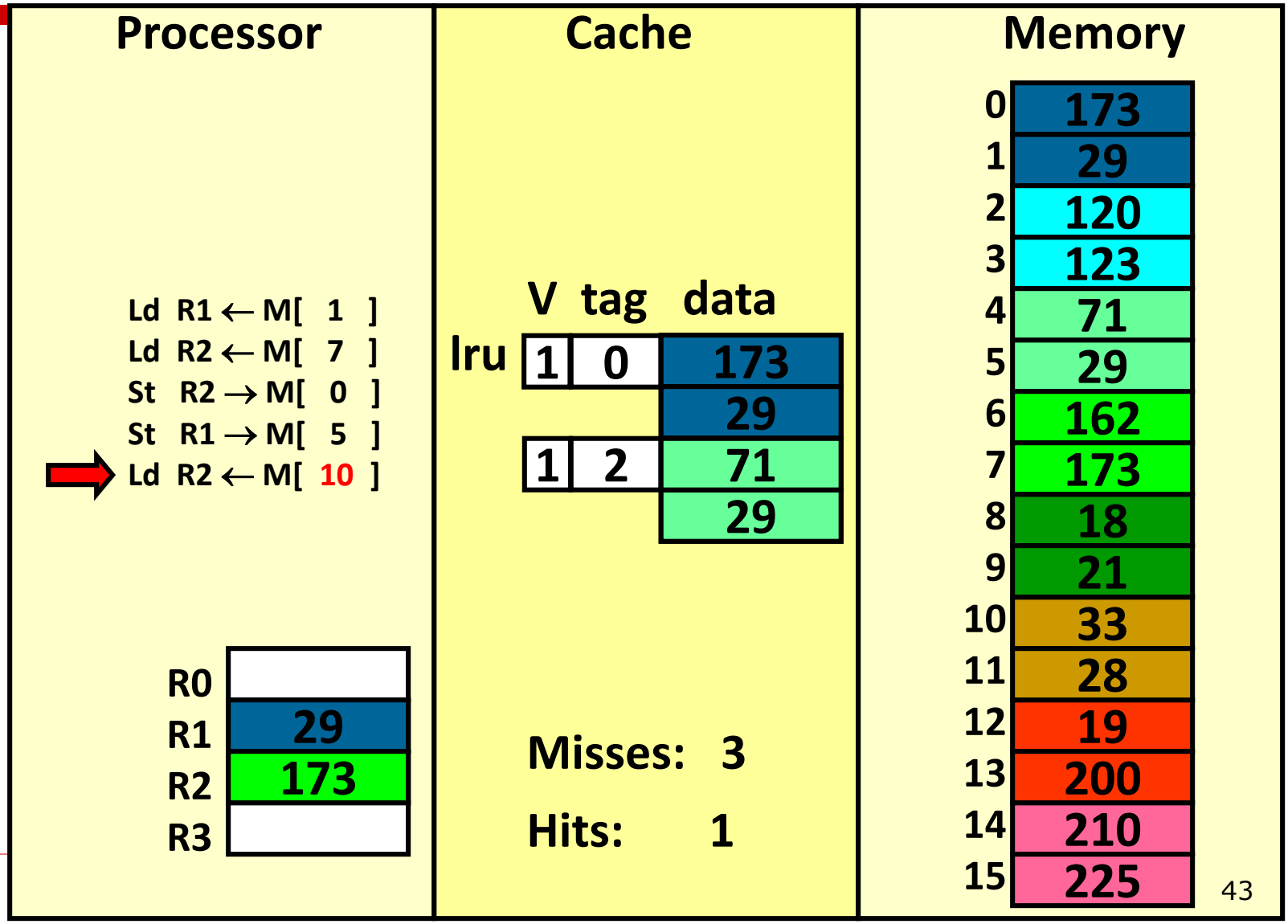
write-through (REF 4)



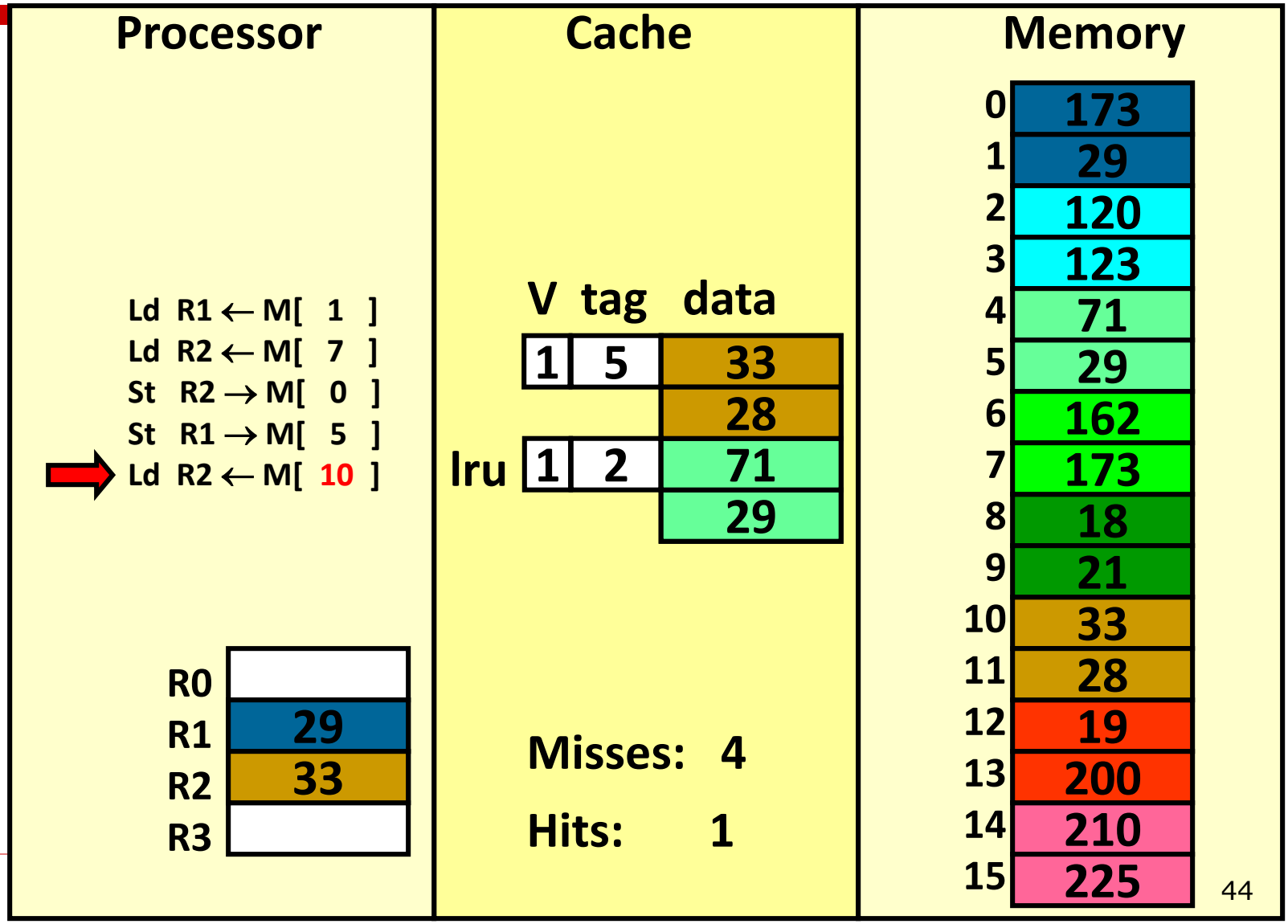
write-through (REF 4)



write-through (REF 6)



write-through (REF 6)



How many memory references?

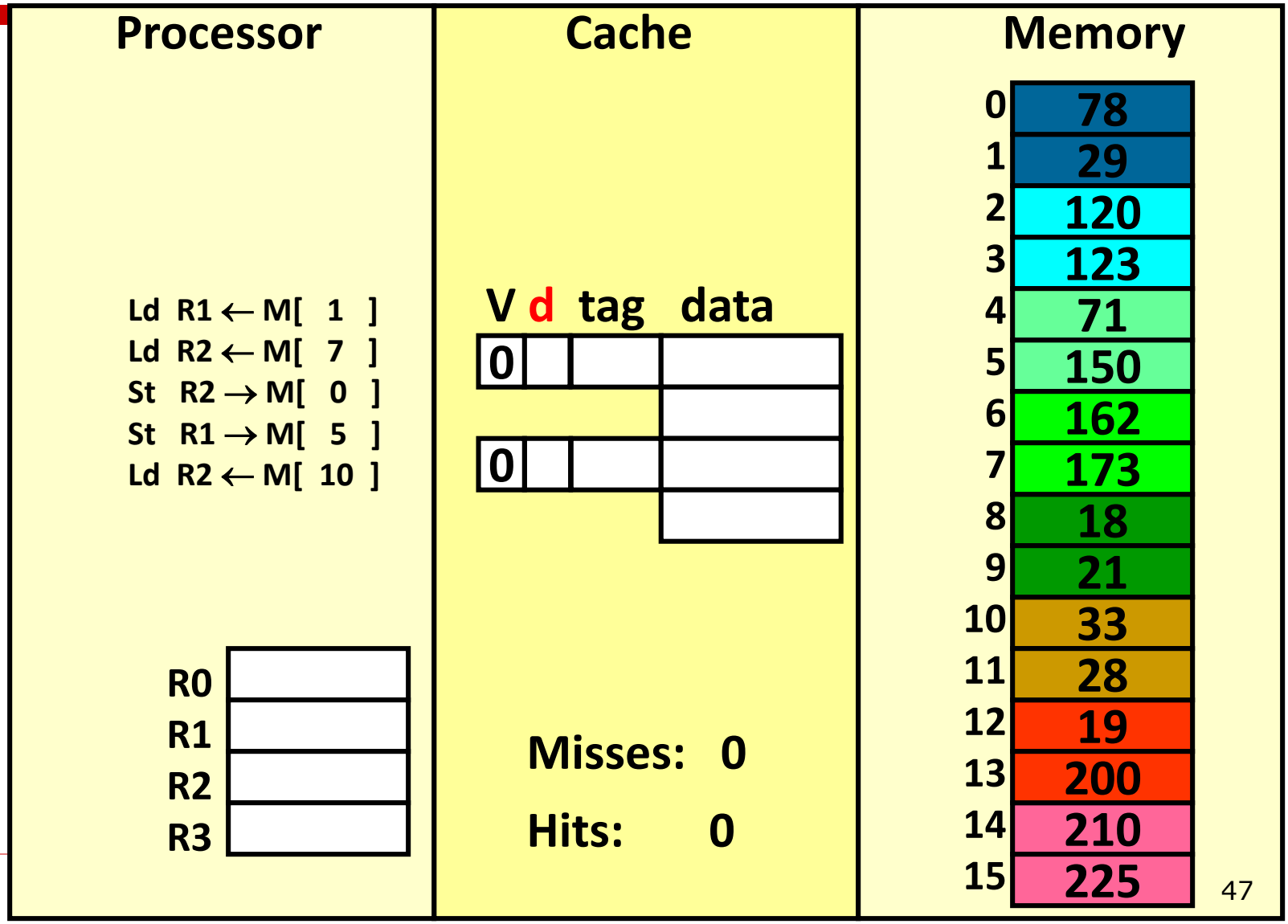
- Each miss reads a block
  - 2 bytes in this cache
- Each store writes a byte
- Total reads: 8 bytes
- Total writes: 2 bytes

but caches generally miss < 20%

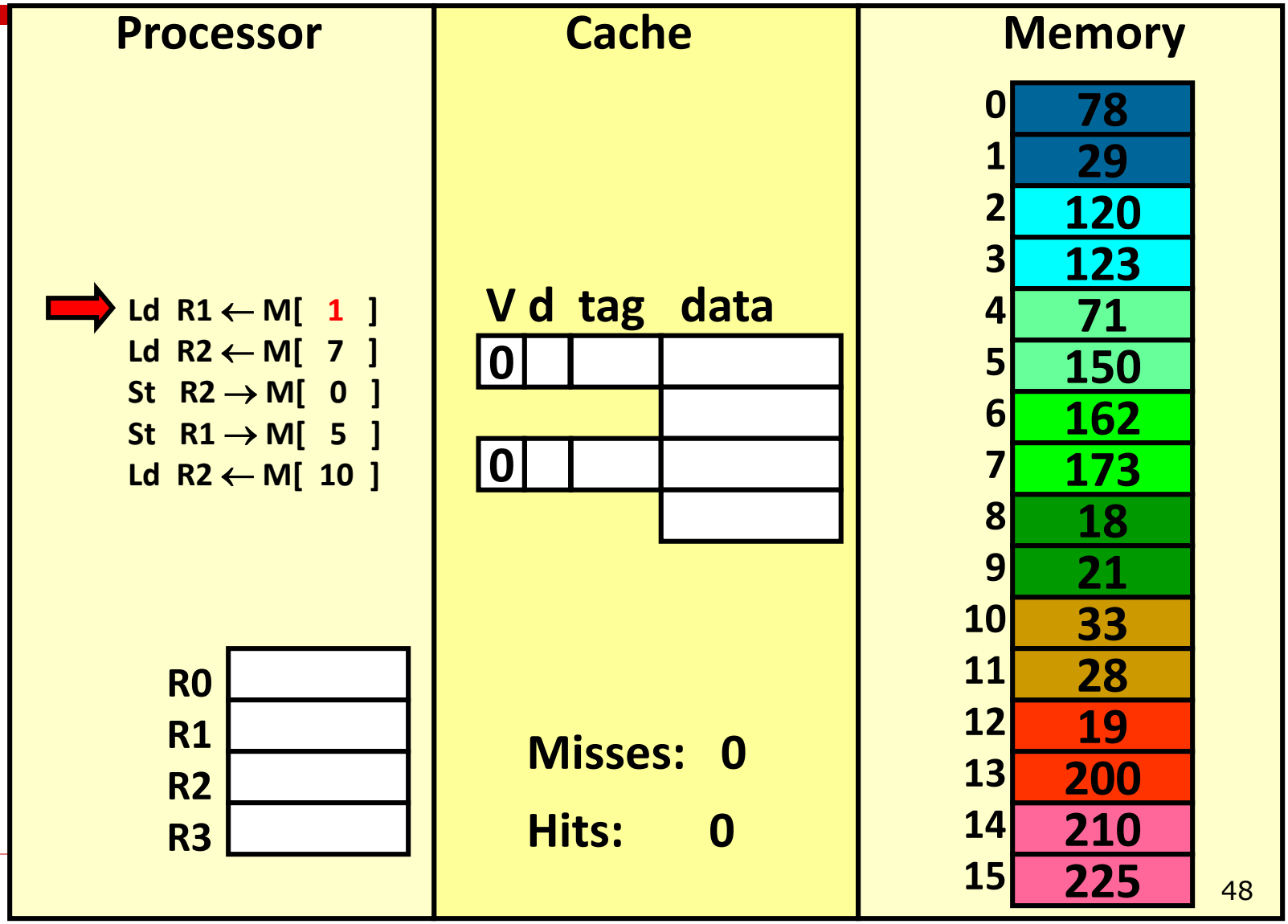
Write-through vs. write-back

- 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.

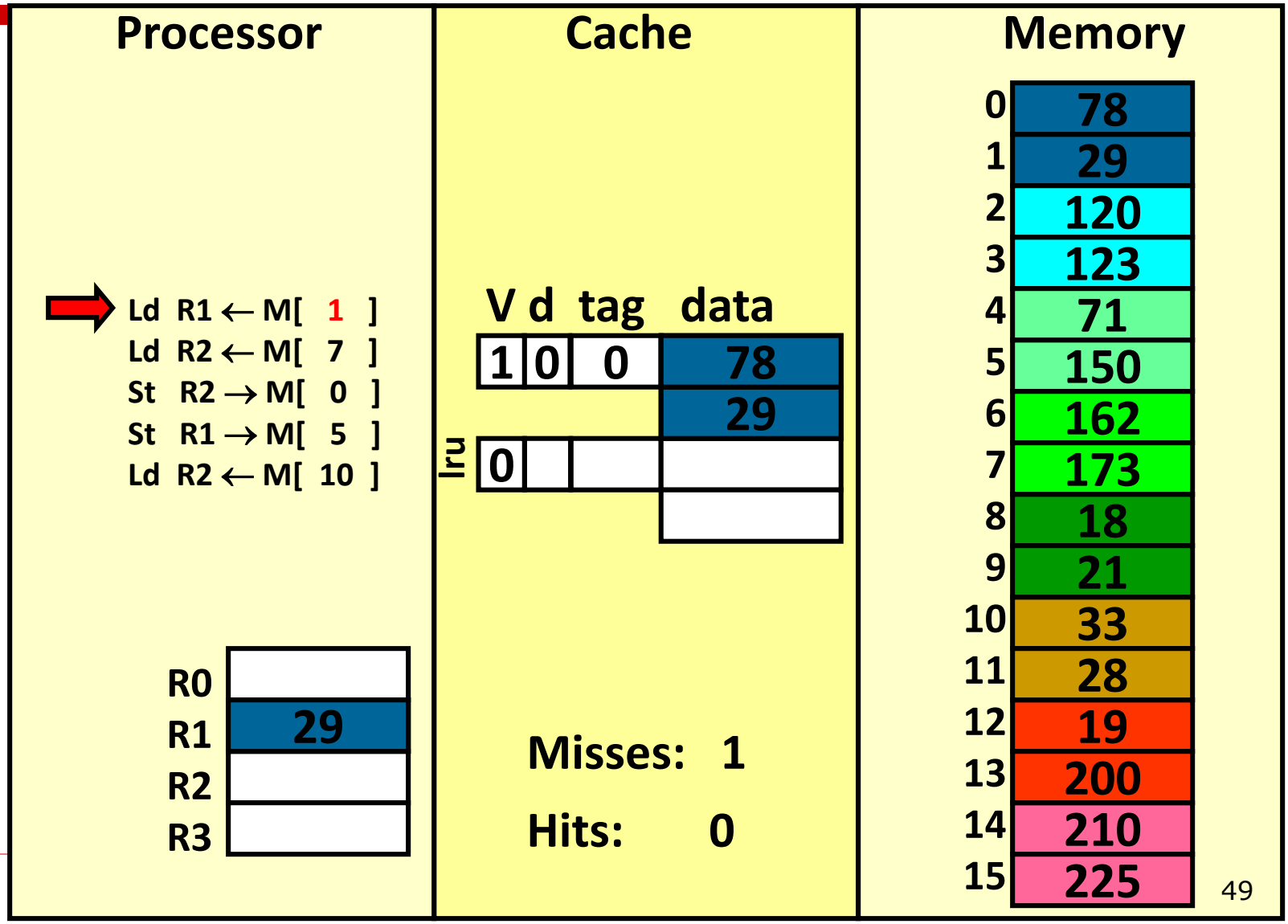
Handling stores (write-back)



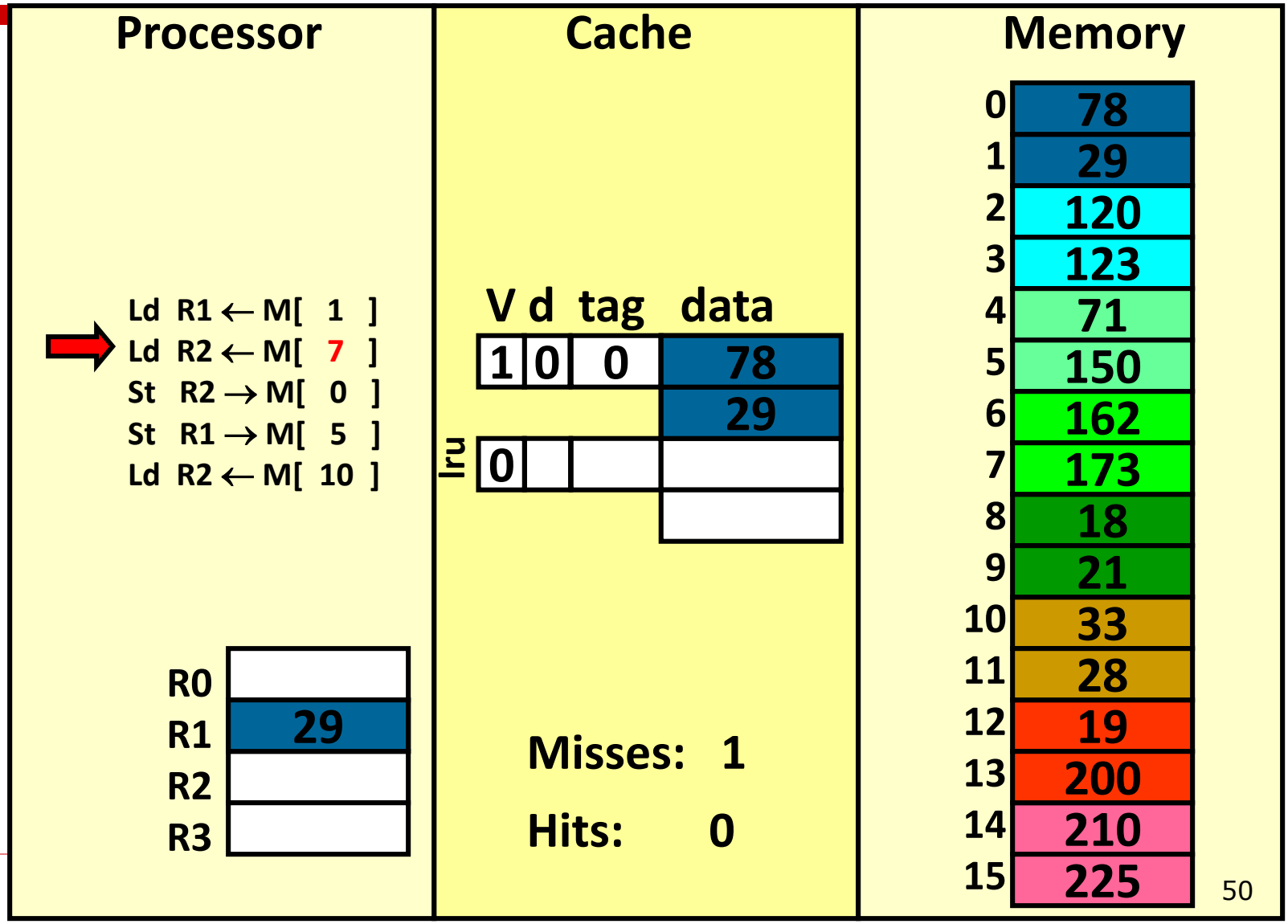
write-back (REF 1)



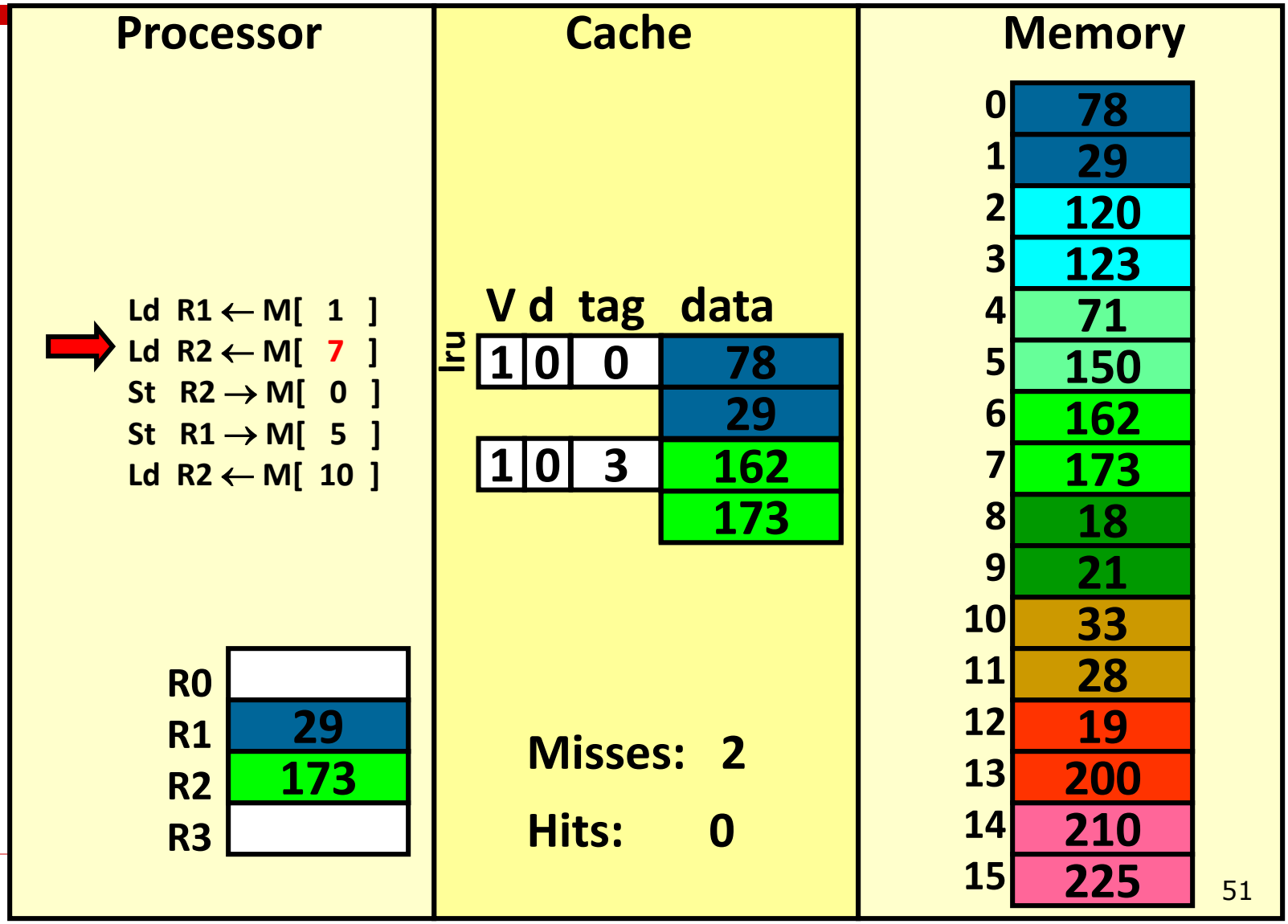
write-back (REF 1)



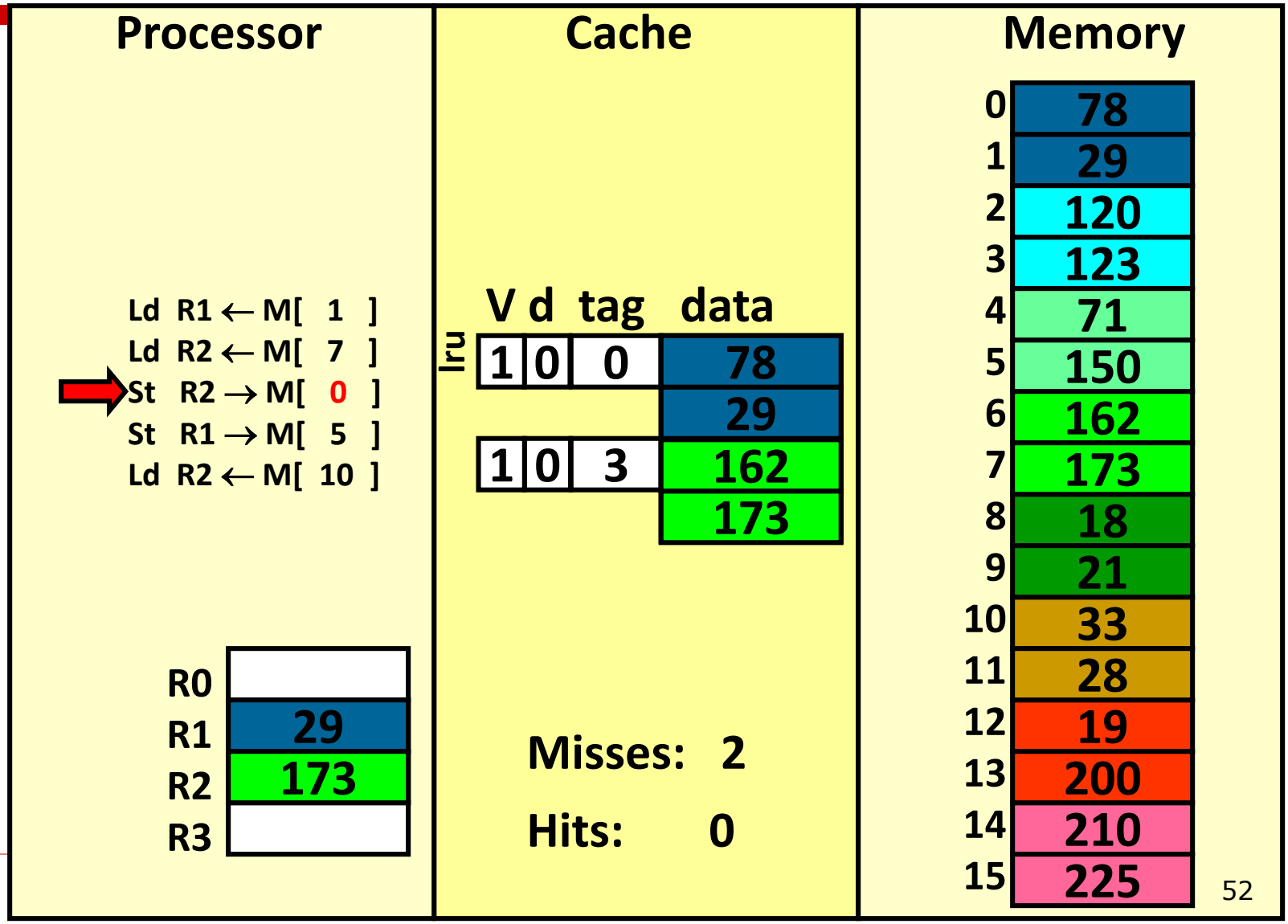
write-back (REF 2)



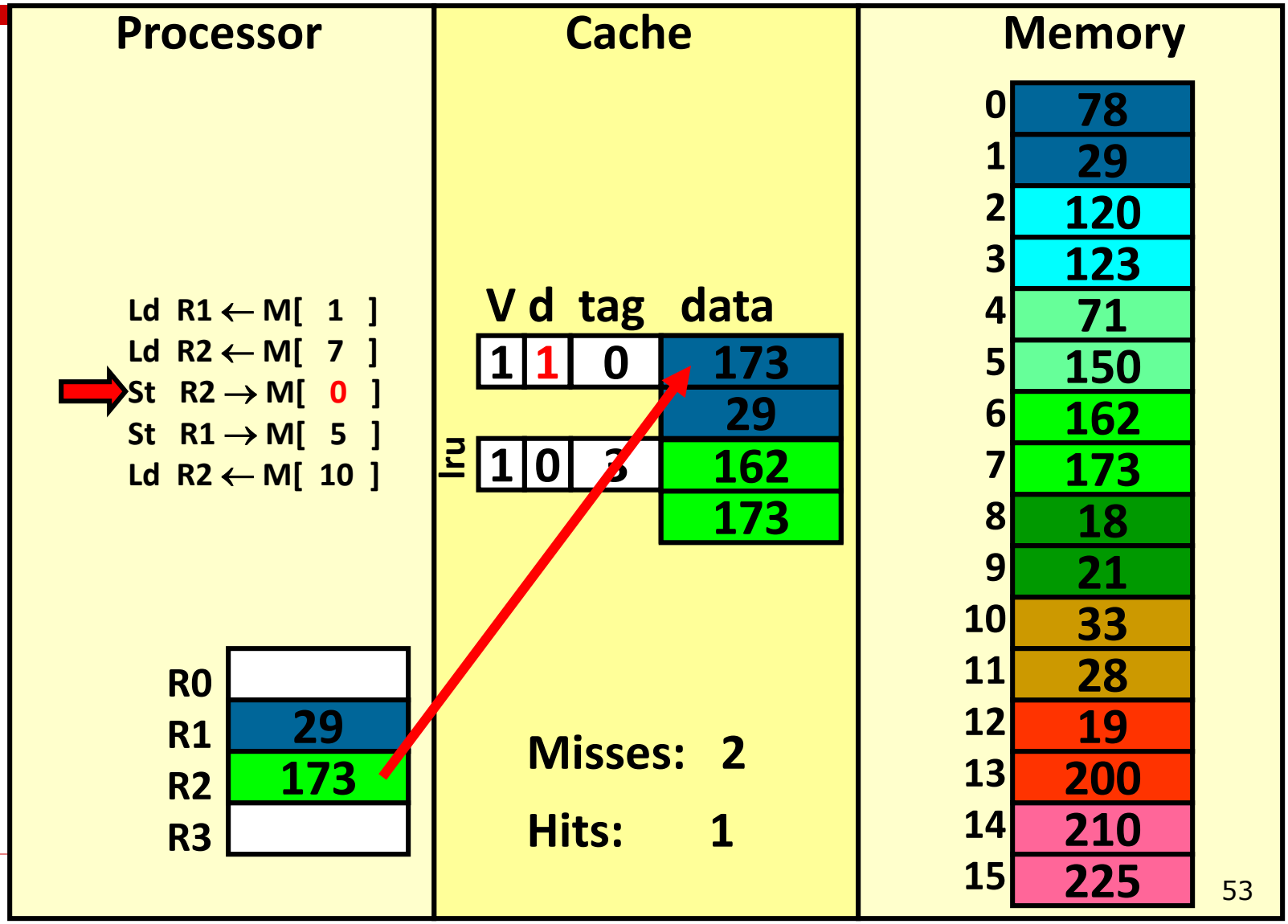
write-back (REF 2)



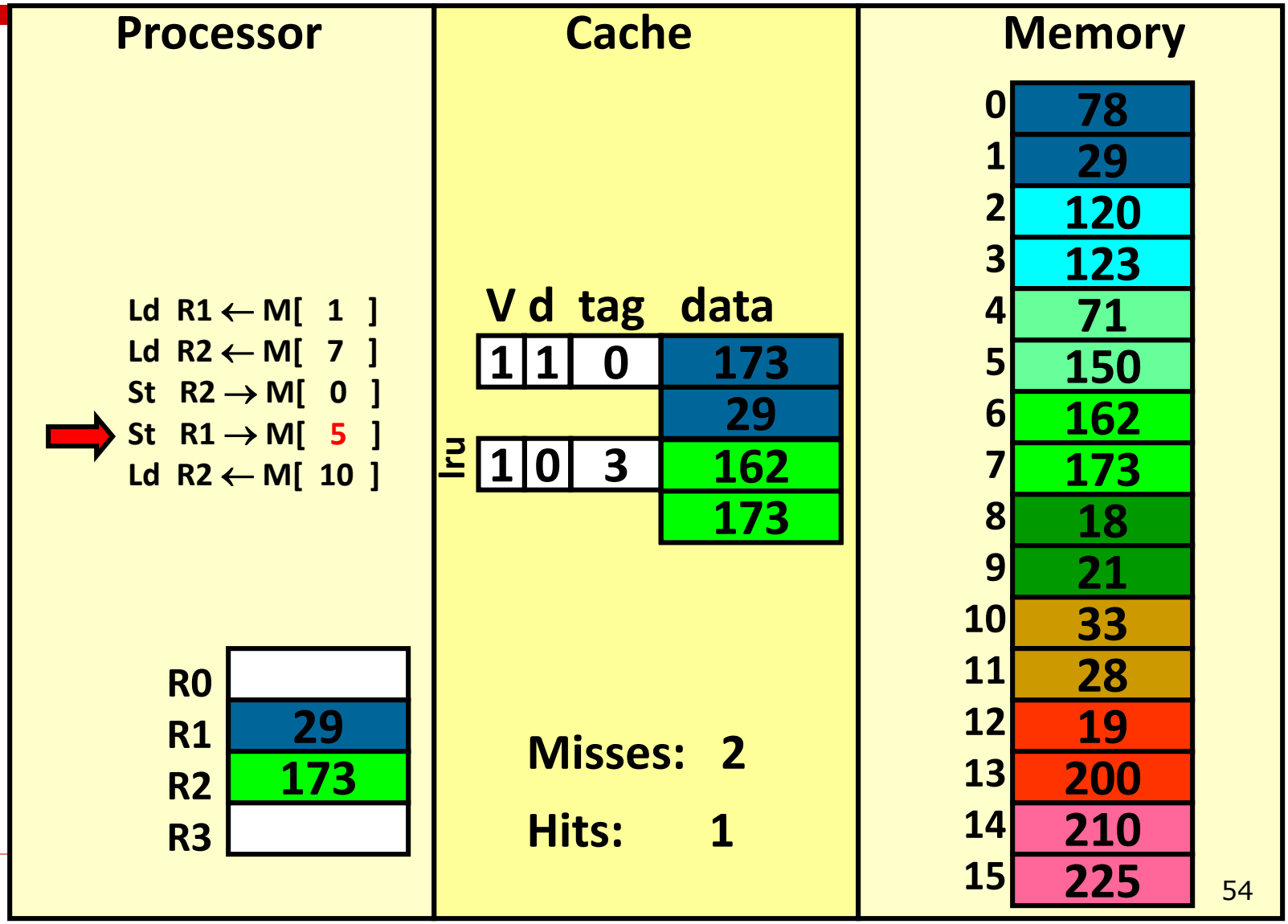
write-back (REF 3)



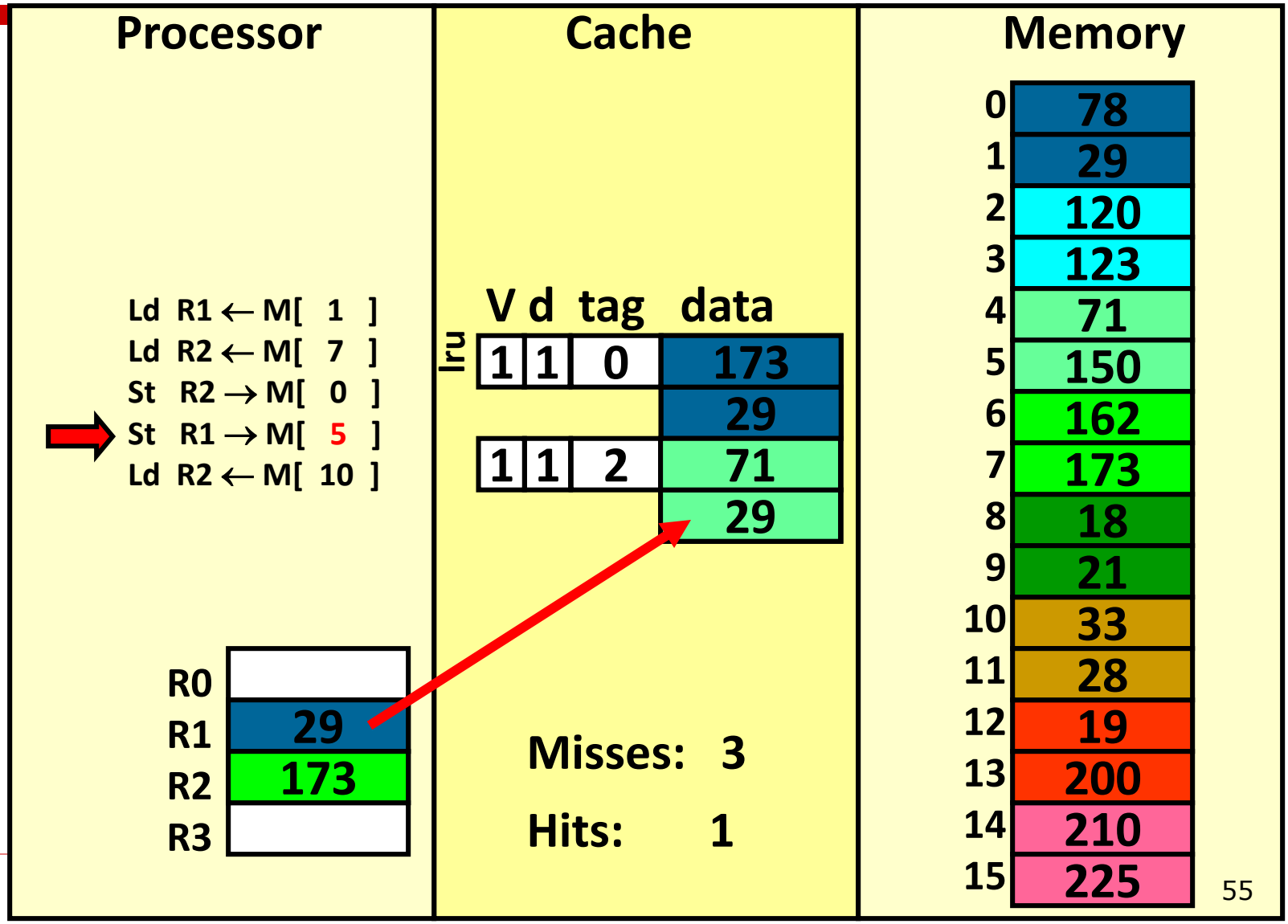
write-back (REF 3)



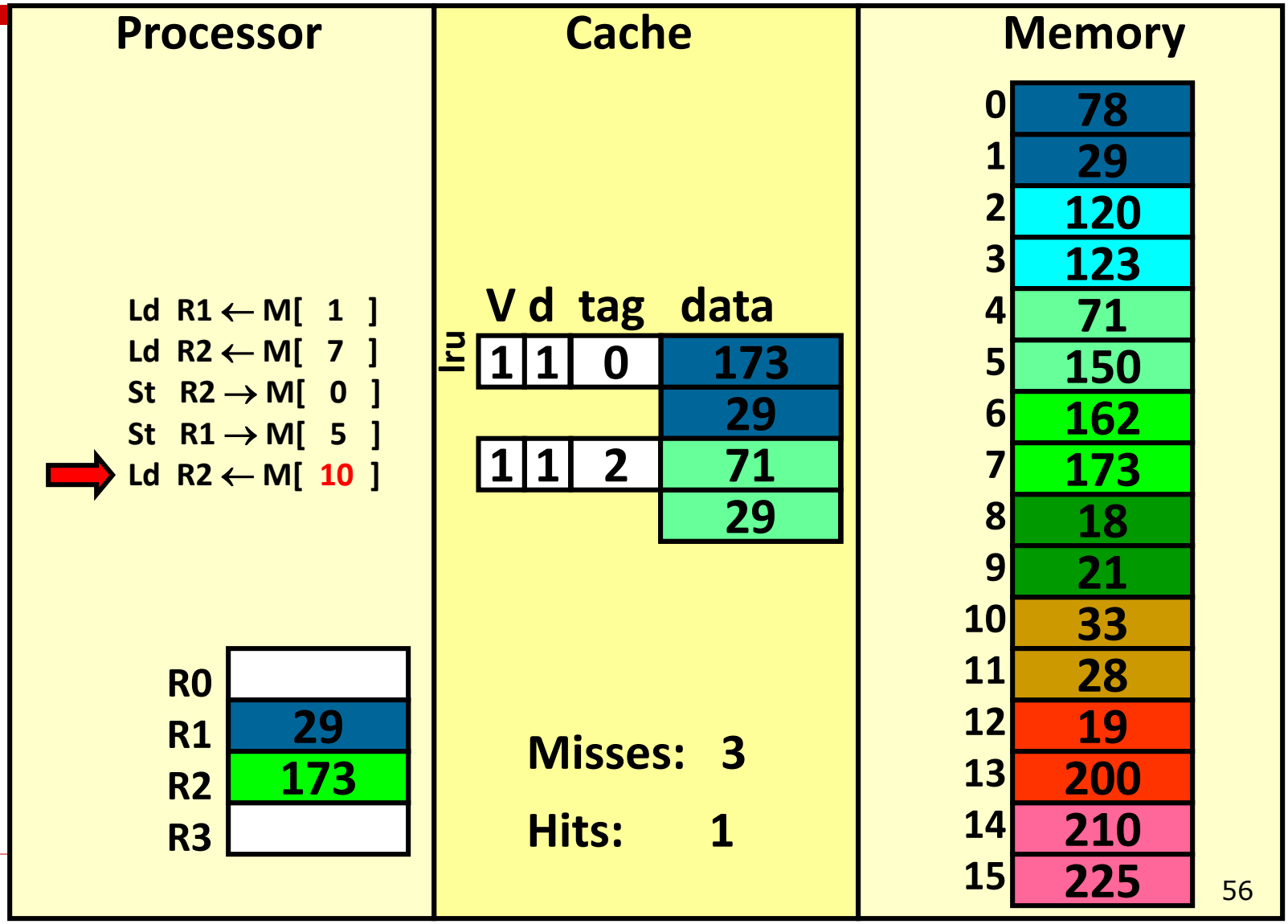
write-back (REF 4)



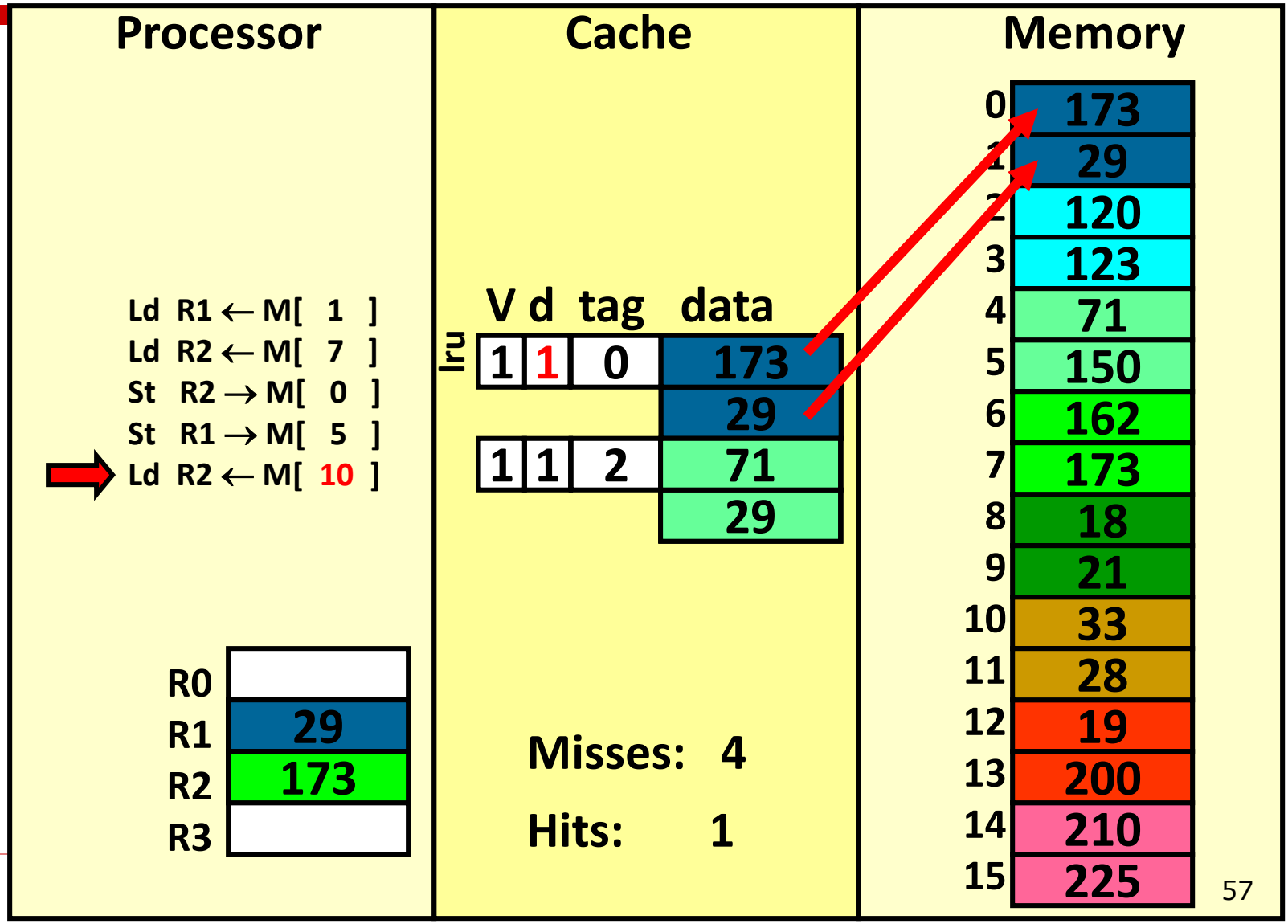
write-back (REF 4)



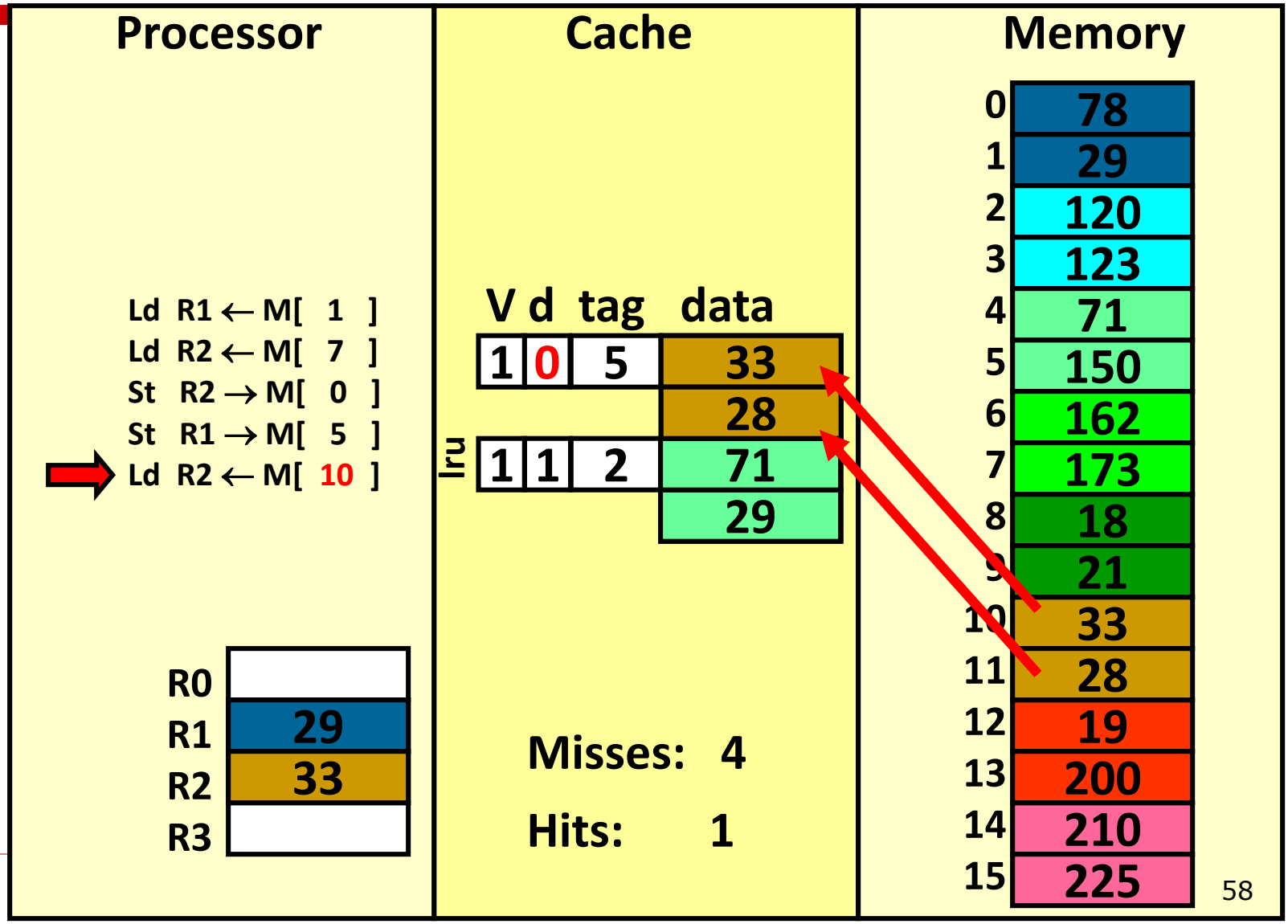
write-back (REF 5)



write-back (REF 5)



write-back (REF 5)



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?

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

Questions to ask about a cache

- What is the block size?
- How many lines?
- How many bytes of data storage?
- How much overhead storage?
- What is the hit rate?
- What is the latency of an access?
- What is the replacement policy ?
  - LRU? LFU? FIFO? Random?

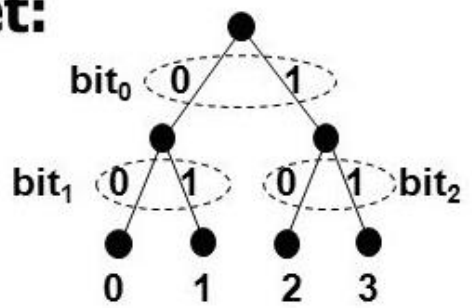
Bonus material
You can skip this, it's just for fun

EECS 370 – Introduction to Computer Organization – Winter 2023

EECS Department
University of Michigan in Ann Arbor, USA

Pseudo LRU (PLRU)

- **PLRU records a partial order**
- **Example: 4-ways, using 3 bits per-set:**
  - Bit_0 specifies if LRU in {0,1} or on {2,3}
  - Bit_1 specifies the LRU between {0,1}
  - Bit_2 specifies the LRU between {2,3}



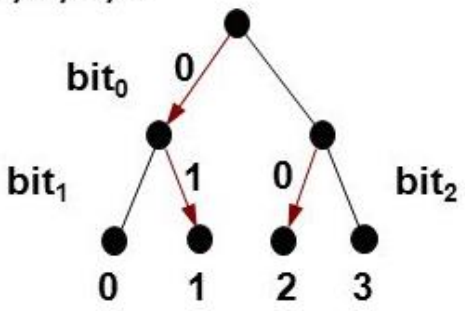
- **Victim selection**
  - Following the opposite directions pointed by the bits

- **Example**
  - Ways are accessed in the following order: 3,0,2,1

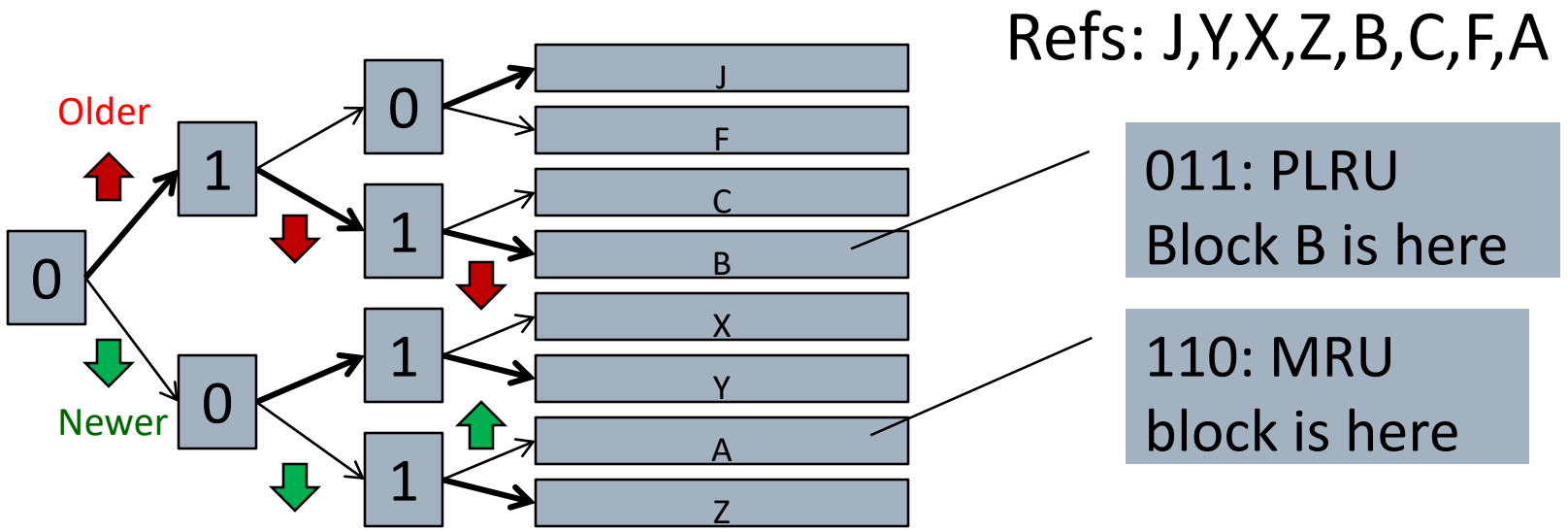
    bit_0=0 points to the pair with 1

    bit_1=1 points to 1

    bit_2=0 2 was accessed after 3
  - Victim: right, right ⇒ 3



PLRU with 8 blocks



- Each access: flip nodes along path to block
- Eviction: follow LRU path
- Overhead: *(a-1)/a* bits per block
  - <1 per line!