

# Caches (Writing)

**Hakim Weatherspoon**

**CS 3410, Spring 2012**

Computer Science

Cornell University

# Goals for Today

---

Cache Parameter Tradeoffs

Cache Conscious Programming

Writing to the Cache

- Write-through vs Write back

---

# Cache Design Tradeoffs

# Cache Design

---

Need to determine parameters:

- Cache size
- Block size (aka line size)
- Number of ways of set-associativity (1, N,  $\infty$ )
- Eviction policy
- Number of levels of caching, parameters for each
- Separate I-cache from D-cache, or Unified cache
- Prefetching policies / instructions
- Write policy

# A Real Example

```
> dmidecode -t cache
```

## Cache Information

```
Configuration: Enabled, Not Socketed, Level 1  
Operational Mode: Write Back  
Installed Size: 128 KB  
Error Correction Type: None
```

## Cache Information

```
Configuration: Enabled, Not Socketed, Level 2  
Operational Mode: Varies With Memory Address  
Installed Size: 6144 KB  
Error Correction Type: Single-bit ECC
```

```
> cd /sys/devices/system/cpu/cpu0; grep cache/*/*
```

```
cache/index0/level:1  
cache/index0/type:Data  
cache/index0/ways_of_associativity:8  
cache/index0/number_of_sets:64  
cache/index0/coherency_line_size:64  
cache/index0/size:32K  
cache/index1/level:1  
cache/index1/type:Instruction  
cache/index1/ways_of_associativity:8  
cache/index1/number_of_sets:64  
cache/index1/coherency_line_size:64  
cache/index1/size:32K  
cache/index2/level:2  
cache/index2/type:Unified  
cache/index2/shared_cpu_list:0-1  
cache/index2/ways_of_associativity:24  
cache/index2/number_of_sets:4096  
cache/index2/coherency_line_size:64  
cache/index2/size:6144K
```

Dual-core 3.16GHz Intel  
(purchased in 2011)

# A Real Example

Dual-core 3.16GHz Intel  
(purchased in 2009)

## Dual 32K L1 Instruction caches

- 8-way set associative
- 64 sets
- 64 byte line size

## Dual 32K L1 Data caches

- Same as above

## Single 6M L2 Unified cache

- 24-way set associative (!!!)
- 4096 sets
- 64 byte line size

4GB Main memory

1TB Disk

# Basic Cache Organization

---

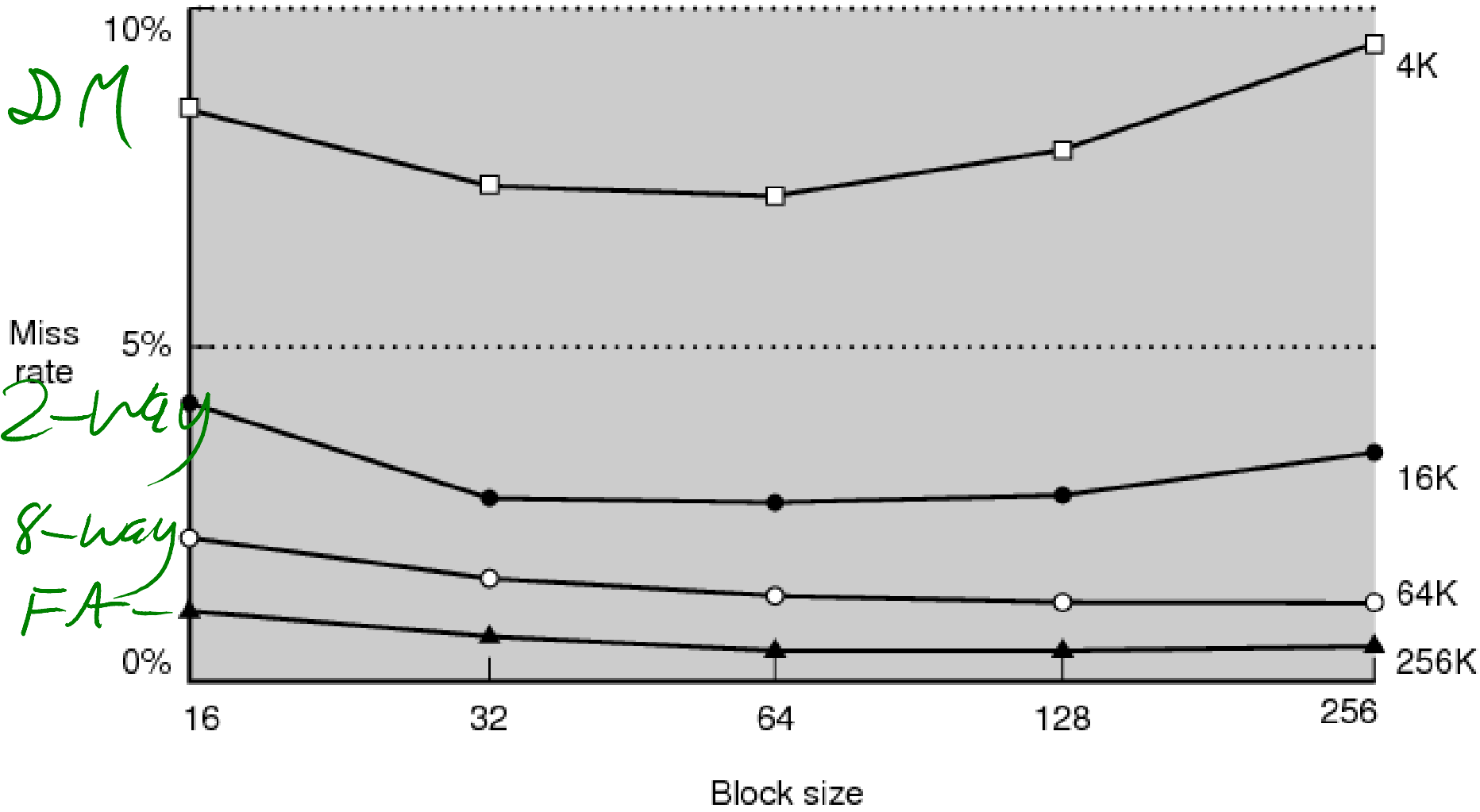
Q: How to decide block size?

A: Try it and see

But: depends on cache size, workload,  
associativity, ...

Experimental approach!

# Experimental Results





# Tradeoffs

---

For a given total cache size,  
larger block sizes mean....

- fewer lines
- so fewer tags (and smaller tags for associative caches)
- so less overhead
- and fewer cold misses (within-block “prefetching”)

But also...

- fewer blocks available (for scattered accesses!)
- so more conflicts
- and larger miss penalty (time to fetch block)

---

# Cache Conscious Programming

# Cache Conscious Programming

```
// H = 12, W = 10
```

```
int A[H][W];
```

```
for(x=0; x < W; x++)
```

```
    for(y=0; y < H; y++)
```

```
        sum += A[y][x];
```

row col



1	11	21							
		2	12	22					
				3	13	23			
						4	14	24	
								5	15
25									
6	16	26							
		7	17	...					
				8	18				
						9	19		
								10	20

Every access is a cache miss!

(unless *entire* matrix can fit in cache)

# Cache Conscious Programming

```
// H = 12, W = 10
```

```
int A[H][W];
```

```
for(y=0; y < H; y++)
```

```
  for(x=0; x < W; x++)
```

```
    sum += A[y][x];
```

*col*  
*row*

*col* *row*

Block size = 4 → 75% hit rate

Block size = 8 → 87.5% hit rate

Block size = 16 → 93.75% hit rate

And you can easily prefetch to warm the cache.

1	2	3	4	5	6	7	8	9	10
11	12	13	...						

---

# Writing with Caches

# Eviction

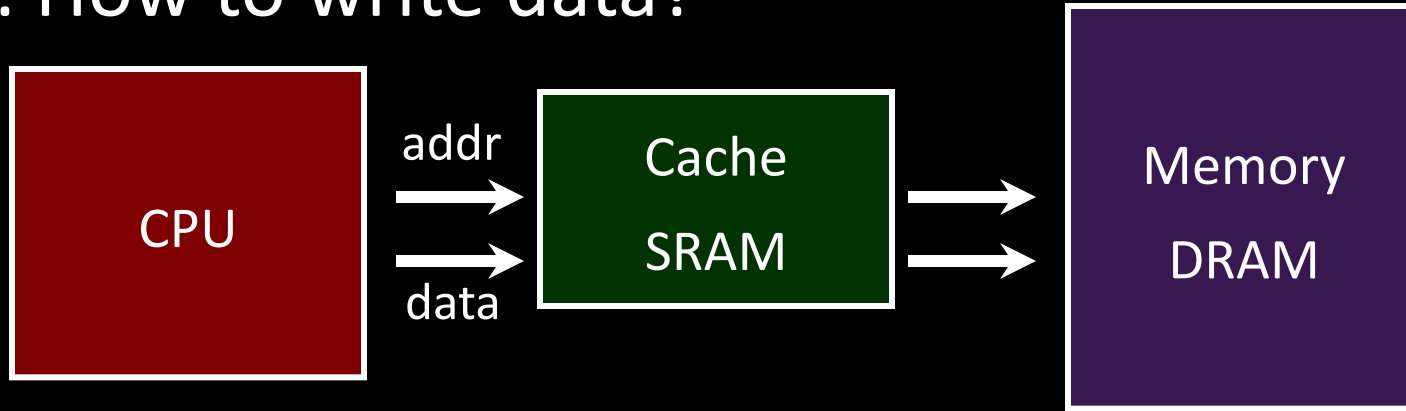
---

Which cache line should be evicted from the cache to make room for a new line?

- Direct-mapped
  - no choice, must evict line selected by index
- Associative caches
  - random: select one of the lines at random
  - round-robin: similar to random
  - FIFO: replace oldest line
  - LRU: replace line that has not been used in the longest time

# Cached Write Policies

Q: How to write data?



If data is already in the cache...

## No-Write

- writes invalidate the cache and go directly to memory

Write-Through — *consistent w/ mem*

- writes go to main memory and cache

## Write-Back

- CPU writes only to cache
- cache writes to main memory later (when block is evicted)

*avoid many writes to mem*

# What about Stores?

---

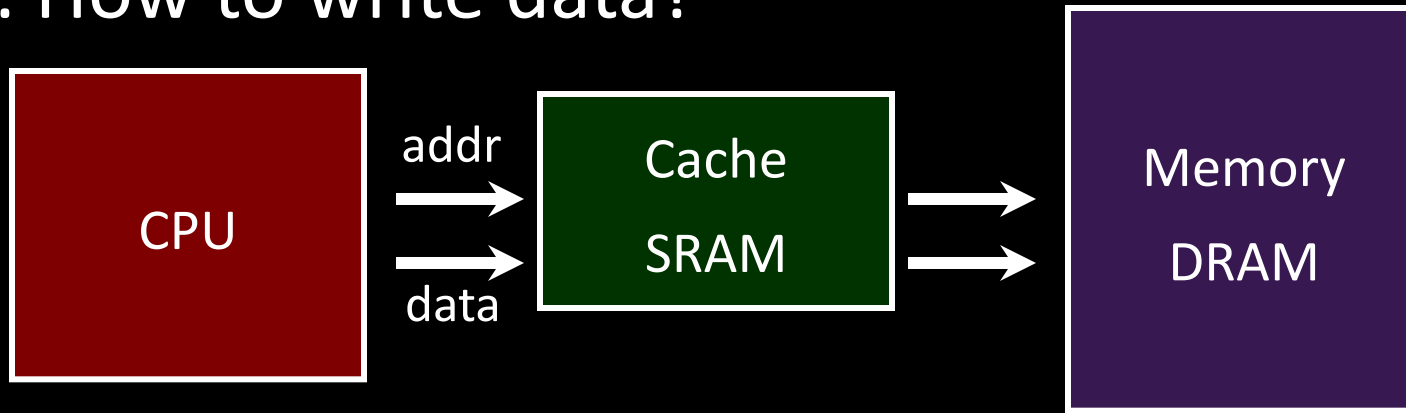
Where should you write the result of a store?

- If that memory location is in the cache?
  - Send it to the cache
  - Should we also send it to memory right away?  
(write-through policy)
  - Wait until we kick the block out (write-back policy)
- If it is not in the cache?
  - Allocate the line (put it in the cache)?  
(write allocate policy)
  - Write it directly to memory without allocation?  
(no write allocate policy)



# Write Allocation Policies

Q: How to write data?



If data is not in the cache...

## Write-Allocate

- allocate a cache line for new data (and maybe write-through)

## No-Write-Allocate

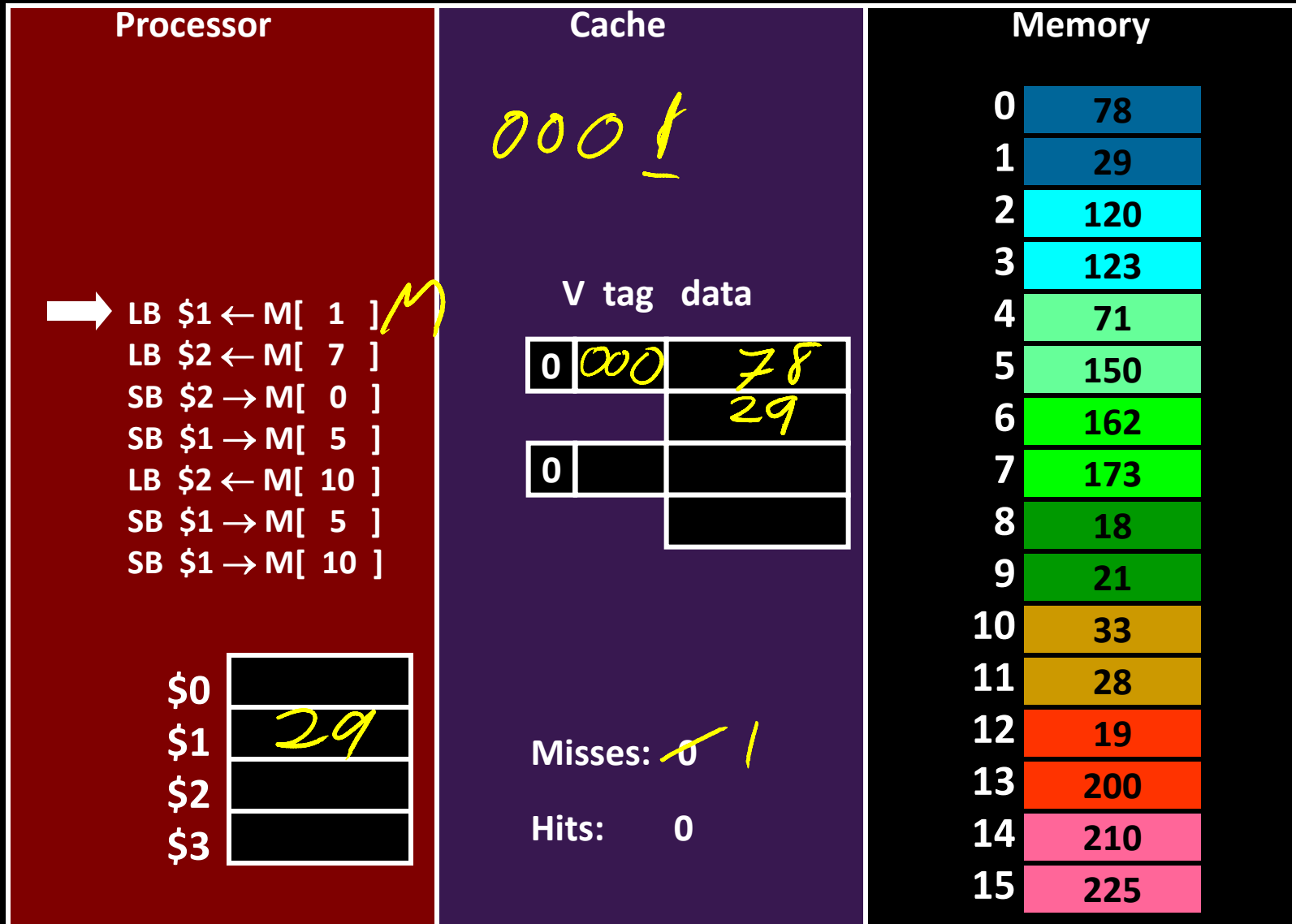
- ignore cache, just go to main memory

# Handling Stores (Write-Through)

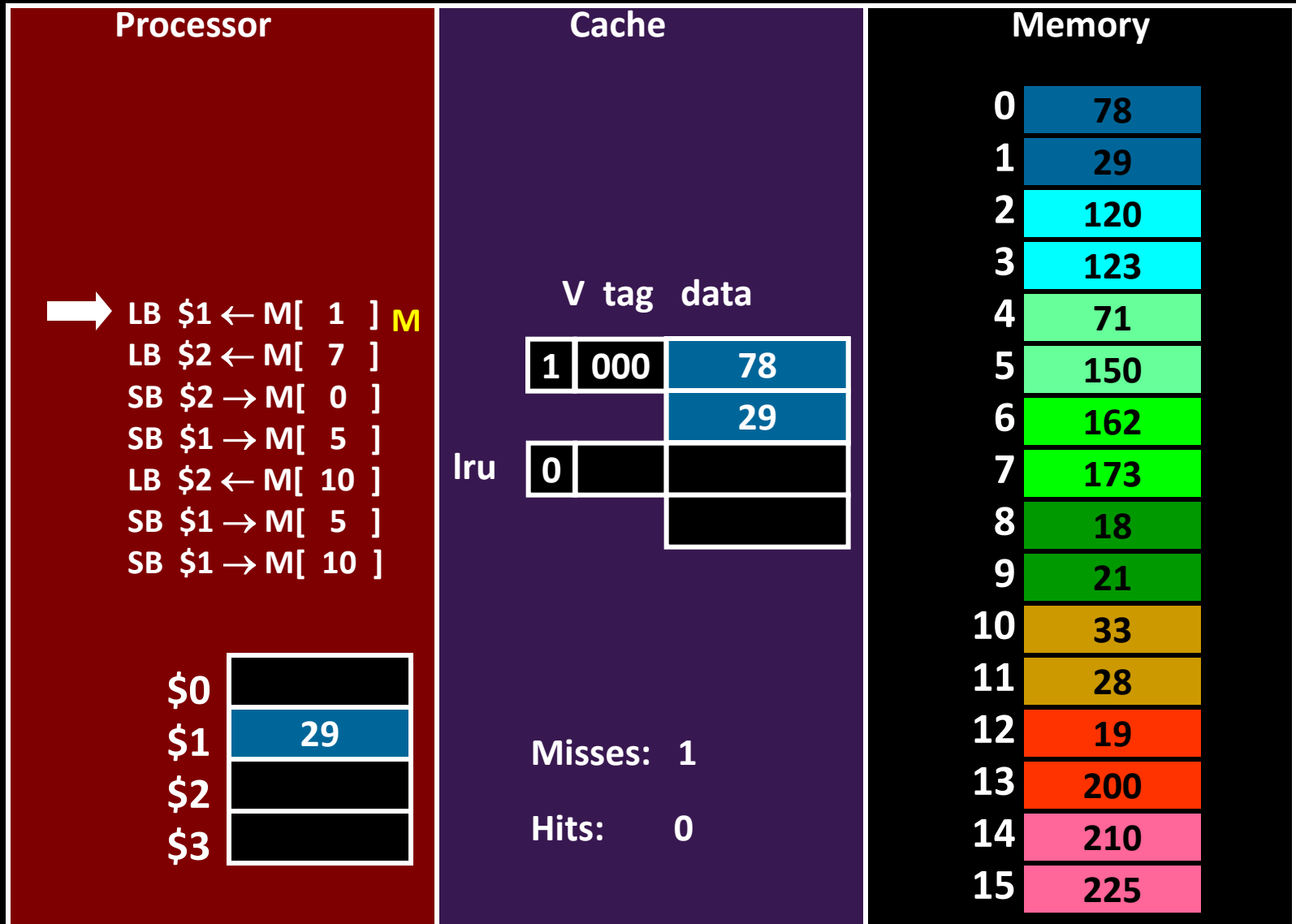
Using **byte addresses** in this example! Addr Bus = ~~5~~<sub>4</sub> bits

Processor	Cache Fully Associative Cache	Memory																																												
<p><b>Assume write-allocate policy</b></p> <p>LB \$1 ← M[ 1 ]            LB \$2 ← M[ 7 ]            SB \$2 → M[ 0 ]            SB \$1 → M[ 5 ]            LB \$2 ← M[ 10 ]            SB \$1 → M[ 5 ]            SB \$1 → M[ 10 ]</p> <div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">\$0</div> <div style="border: 1px solid white; width: 100px; height: 20px; background-color: black;"></div> </div> <div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">\$1</div> <div style="border: 1px solid white; width: 100px; height: 20px; background-color: black;"></div> </div> <div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">\$2</div> <div style="border: 1px solid white; width: 100px; height: 20px; background-color: black;"></div> </div> <div style="display: flex; align-items: center;"> <div style="margin-right: 5px;">\$3</div> <div style="border: 1px solid white; width: 100px; height: 20px; background-color: black;"></div> </div>	<p>2 cache lines  <u>2</u> word block  <u>3</u> bit tag field            1 bit block offset field            V tag data</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%; text-align: center;">0</td> <td style="width: 20%;"></td> <td style="width: 70%;"></td> </tr> <tr> <td style="width: 10%;"></td> <td style="width: 20%;"></td> <td style="width: 70%;"></td> </tr> <tr> <td style="width: 10%; text-align: center;">0</td> <td style="width: 20%;"></td> <td style="width: 70%;"></td> </tr> <tr> <td style="width: 10%;"></td> <td style="width: 20%;"></td> <td style="width: 70%;"></td> </tr> </table> <p>Misses: 0            Hits: 0</p>	0						0						<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 5%; text-align: center;">0</td><td style="width: 95%; text-align: center;">78</td></tr> <tr><td style="text-align: center;">1</td><td style="text-align: center;">29</td></tr> <tr><td style="text-align: center;">2</td><td style="text-align: center;">120</td></tr> <tr><td style="text-align: center;">3</td><td style="text-align: center;">123</td></tr> <tr><td style="text-align: center;">4</td><td style="text-align: center;">71</td></tr> <tr><td style="text-align: center;">5</td><td style="text-align: center;">150</td></tr> <tr><td style="text-align: center;">6</td><td style="text-align: center;">162</td></tr> <tr><td style="text-align: center;">7</td><td style="text-align: center;">173</td></tr> <tr><td style="text-align: center;">8</td><td style="text-align: center;">18</td></tr> <tr><td style="text-align: center;">9</td><td style="text-align: center;">21</td></tr> <tr><td style="text-align: center;">10</td><td style="text-align: center;">33</td></tr> <tr><td style="text-align: center;">11</td><td style="text-align: center;">28</td></tr> <tr><td style="text-align: center;">12</td><td style="text-align: center;">19</td></tr> <tr><td style="text-align: center;">13</td><td style="text-align: center;">200</td></tr> <tr><td style="text-align: center;">14</td><td style="text-align: center;">210</td></tr> <tr><td style="text-align: center;">15</td><td style="text-align: center;">225</td></tr> </table>	0	78	1	29	2	120	3	123	4	71	5	150	6	162	7	173	8	18	9	21	10	33	11	28	12	19	13	200	14	210	15	225
0																																														
0																																														
0	78																																													
1	29																																													
2	120																																													
3	123																																													
4	71																																													
5	150																																													
6	162																																													
7	173																																													
8	18																																													
9	21																																													
10	33																																													
11	28																																													
12	19																																													
13	200																																													
14	210																																													
15	225																																													

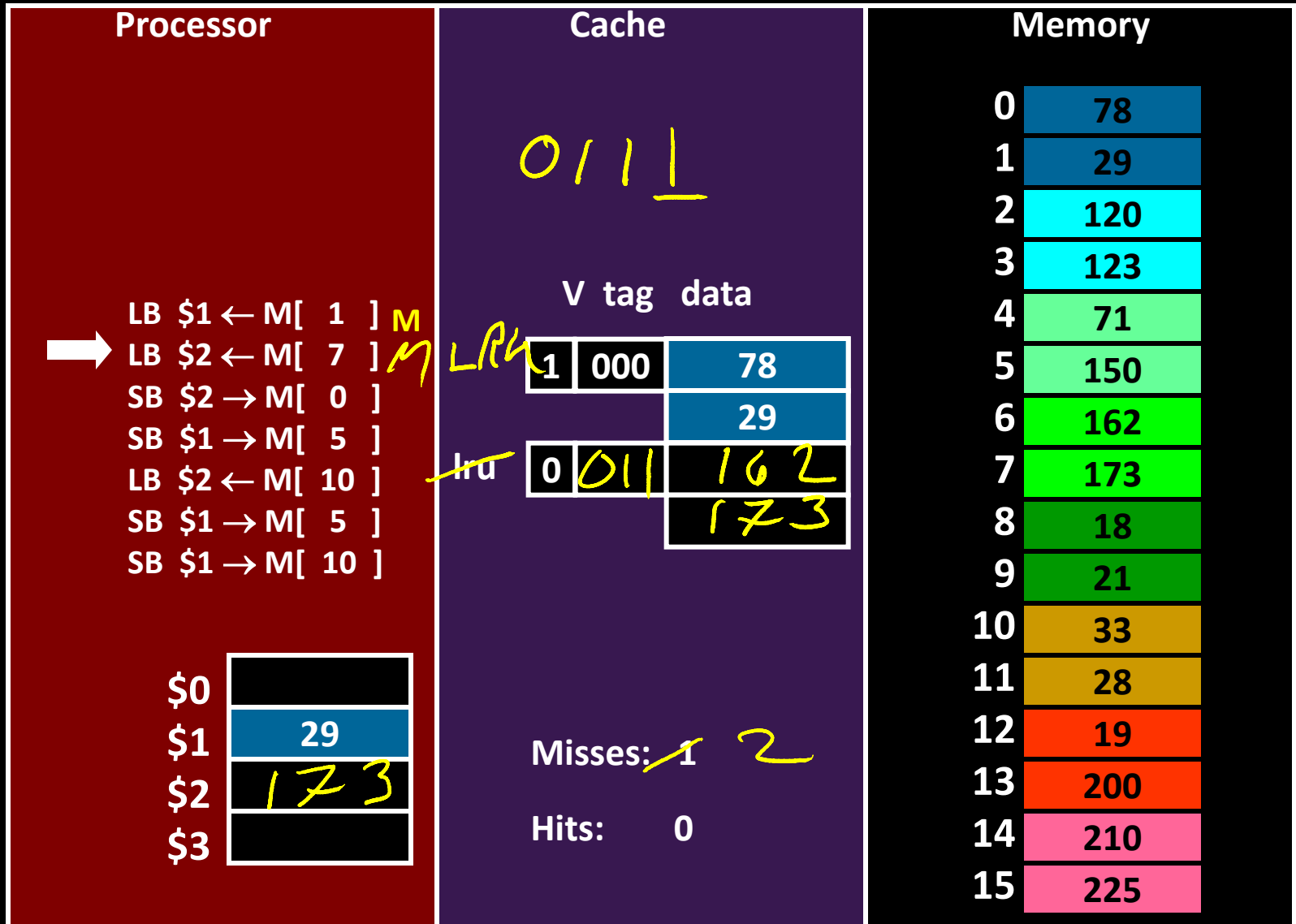
# Write-Through (REF 1)



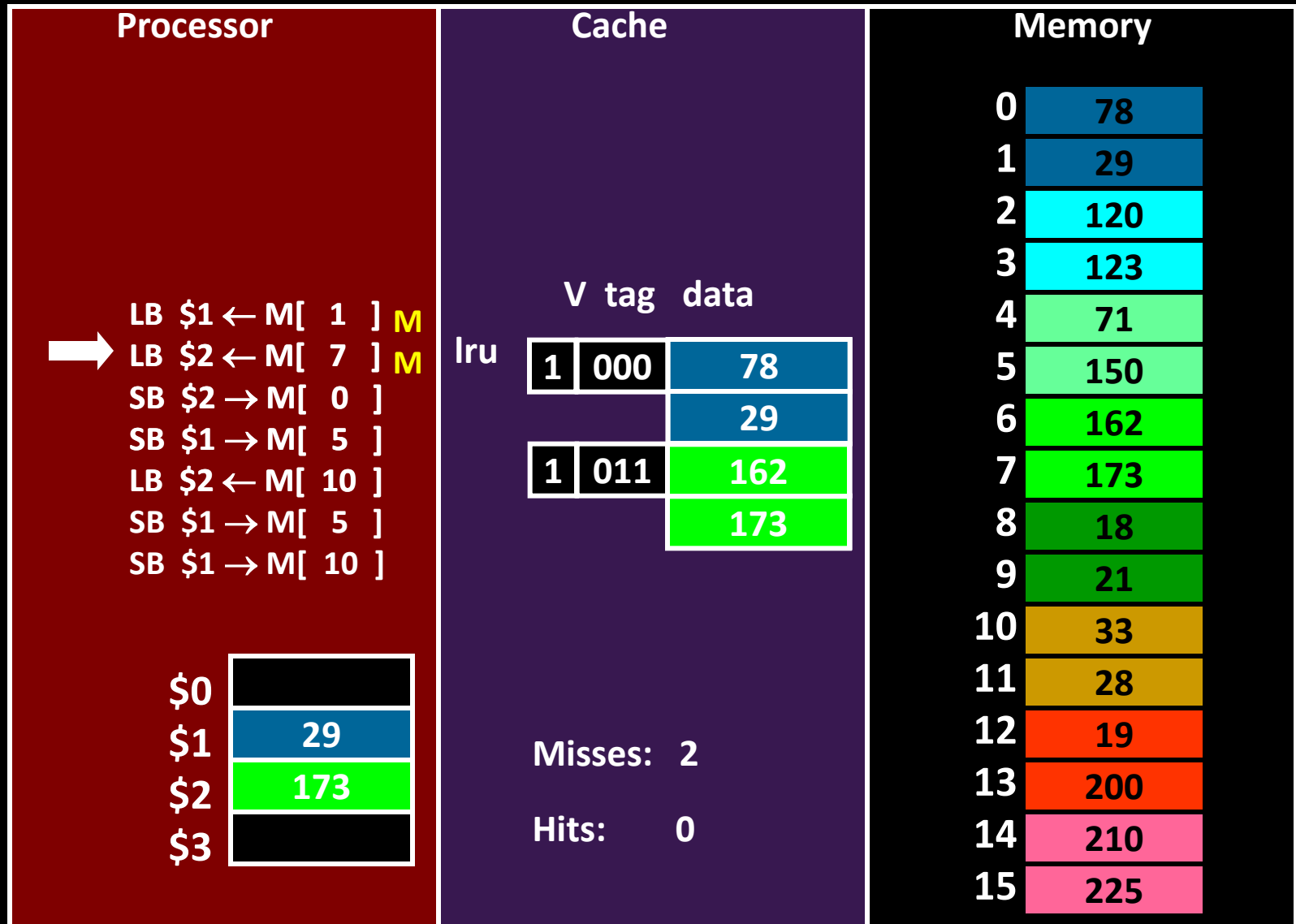
# Write-Through (REF 1)



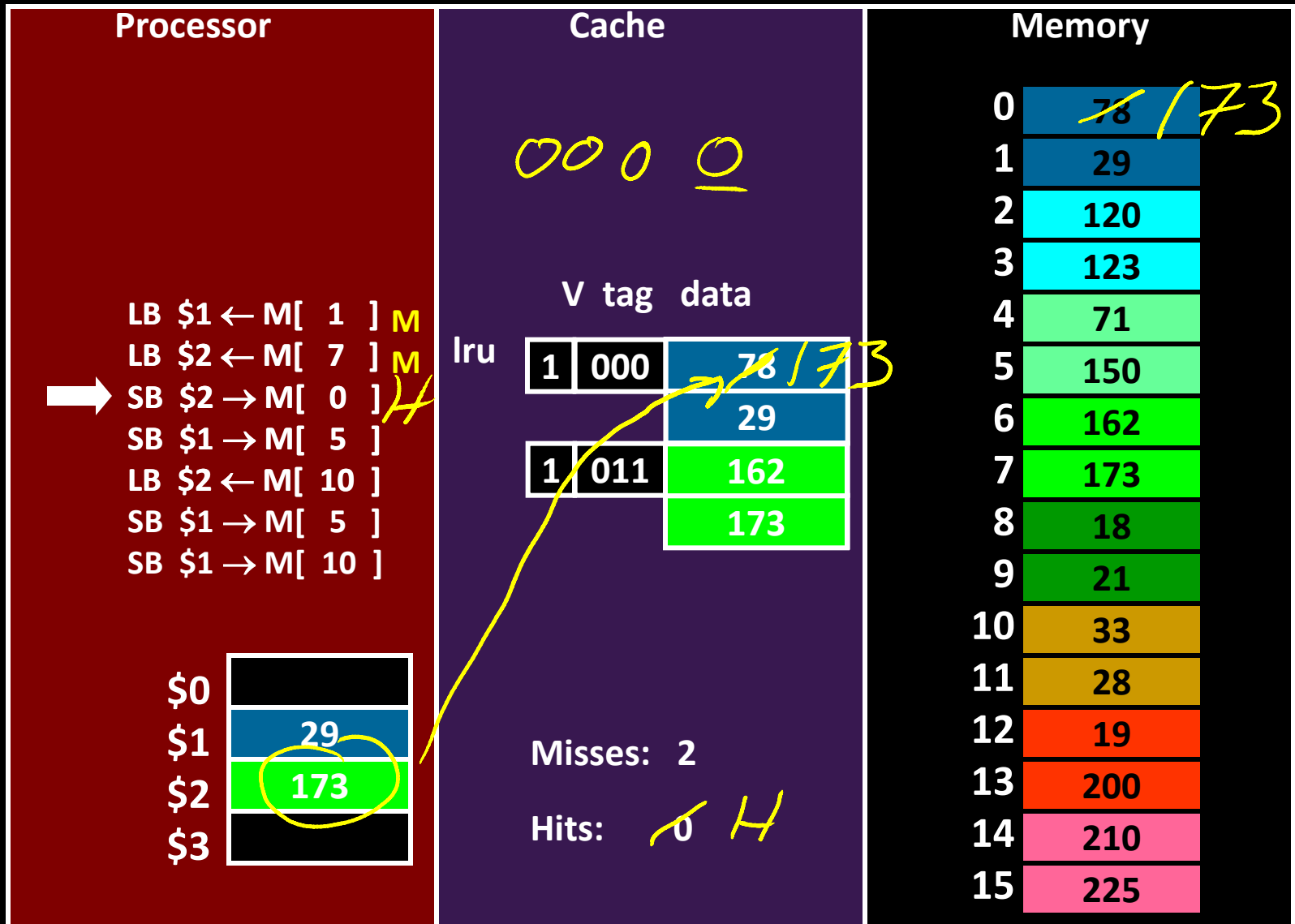
# Write-Through (REF 2)



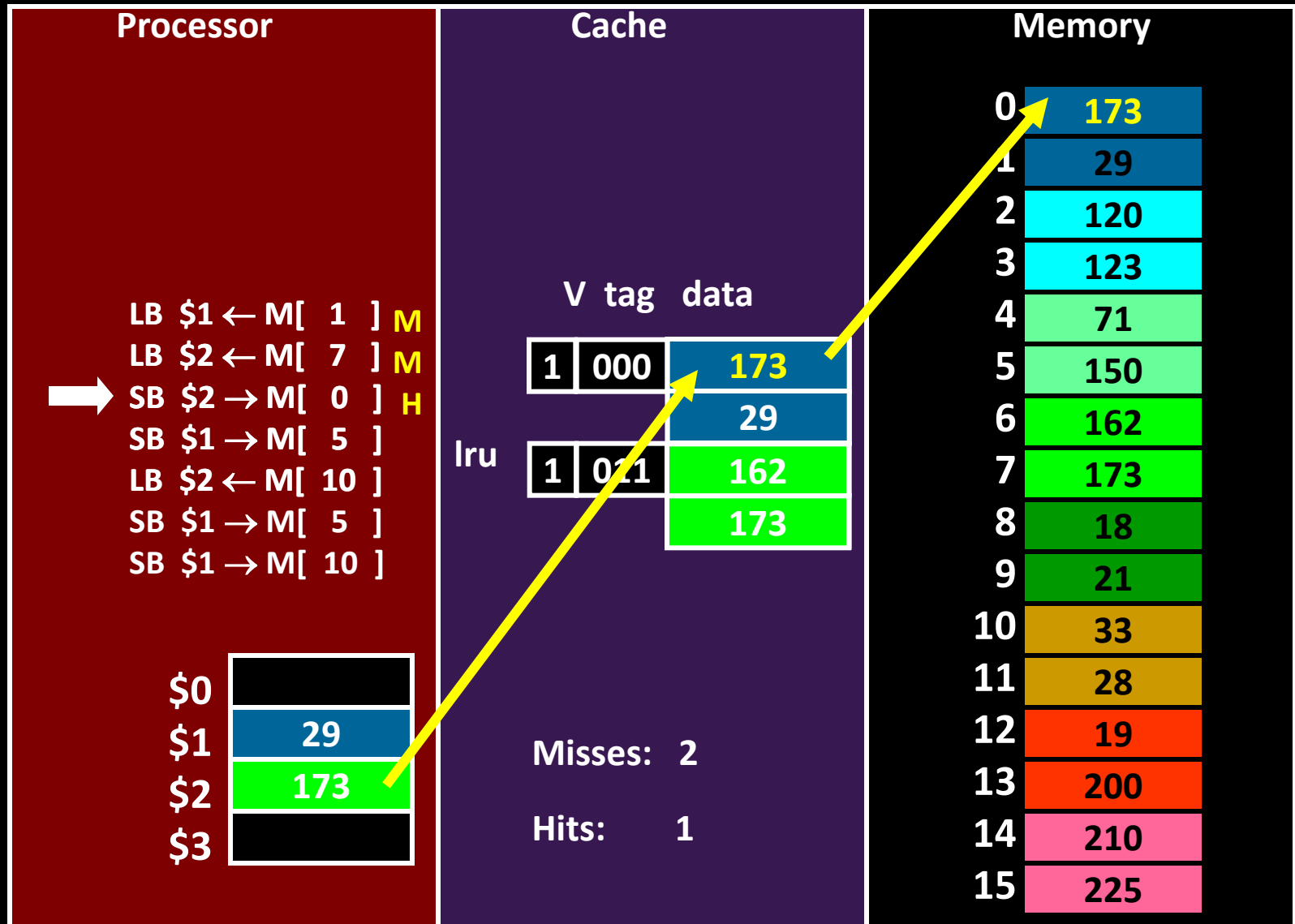
# Write-Through (REF 2)



# Write-Through (REF 3)

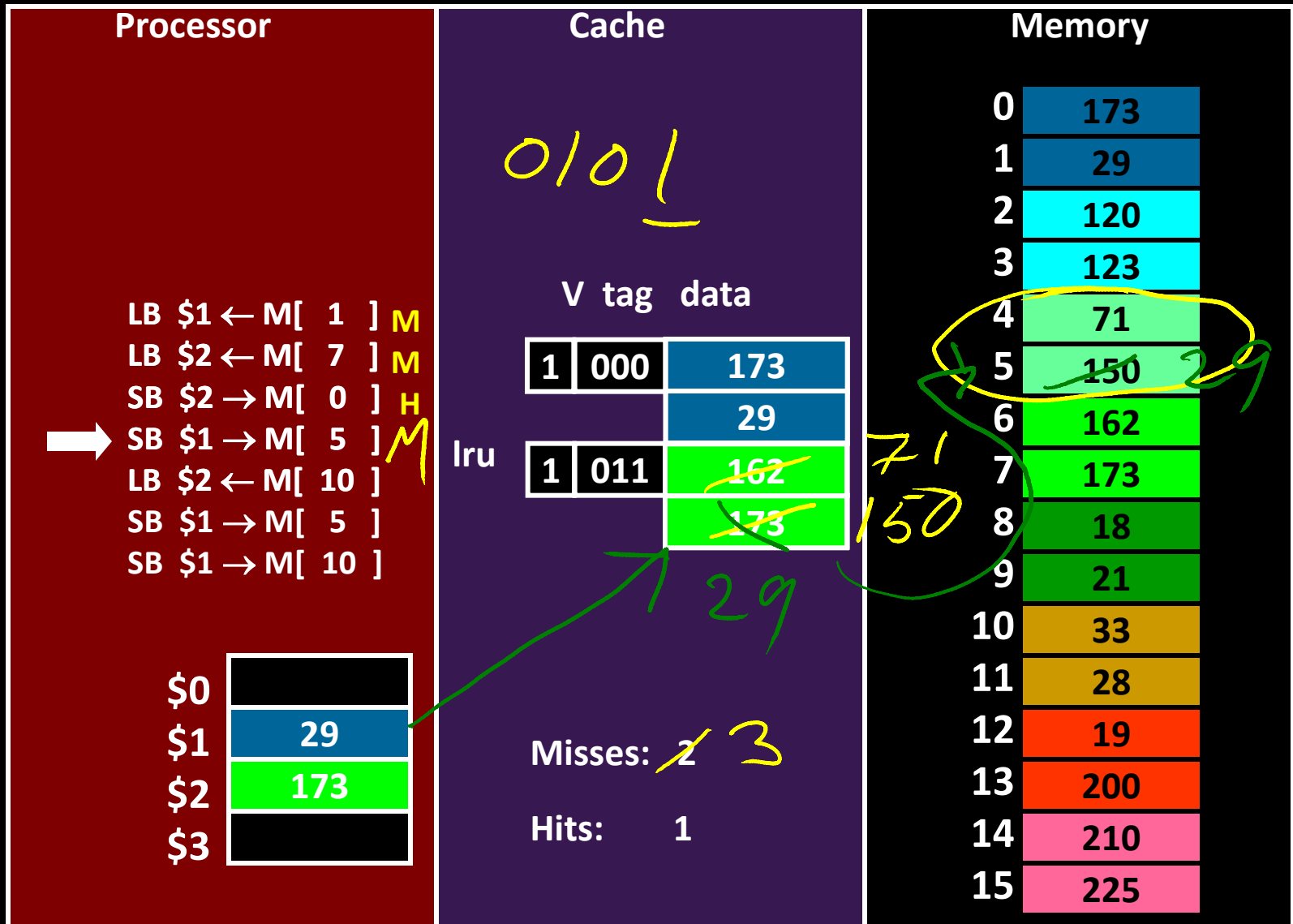


# Write-Through (REF 3)

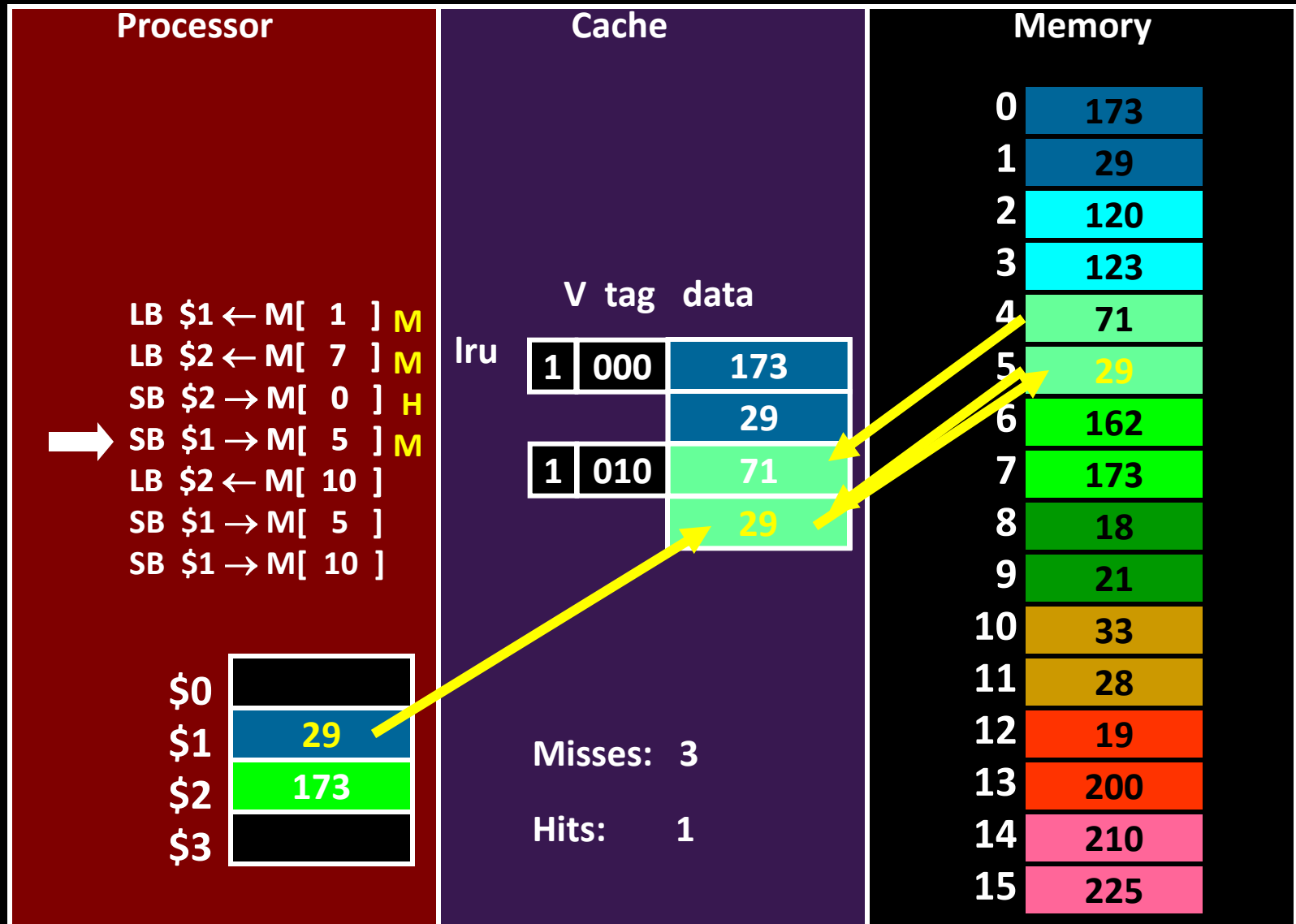




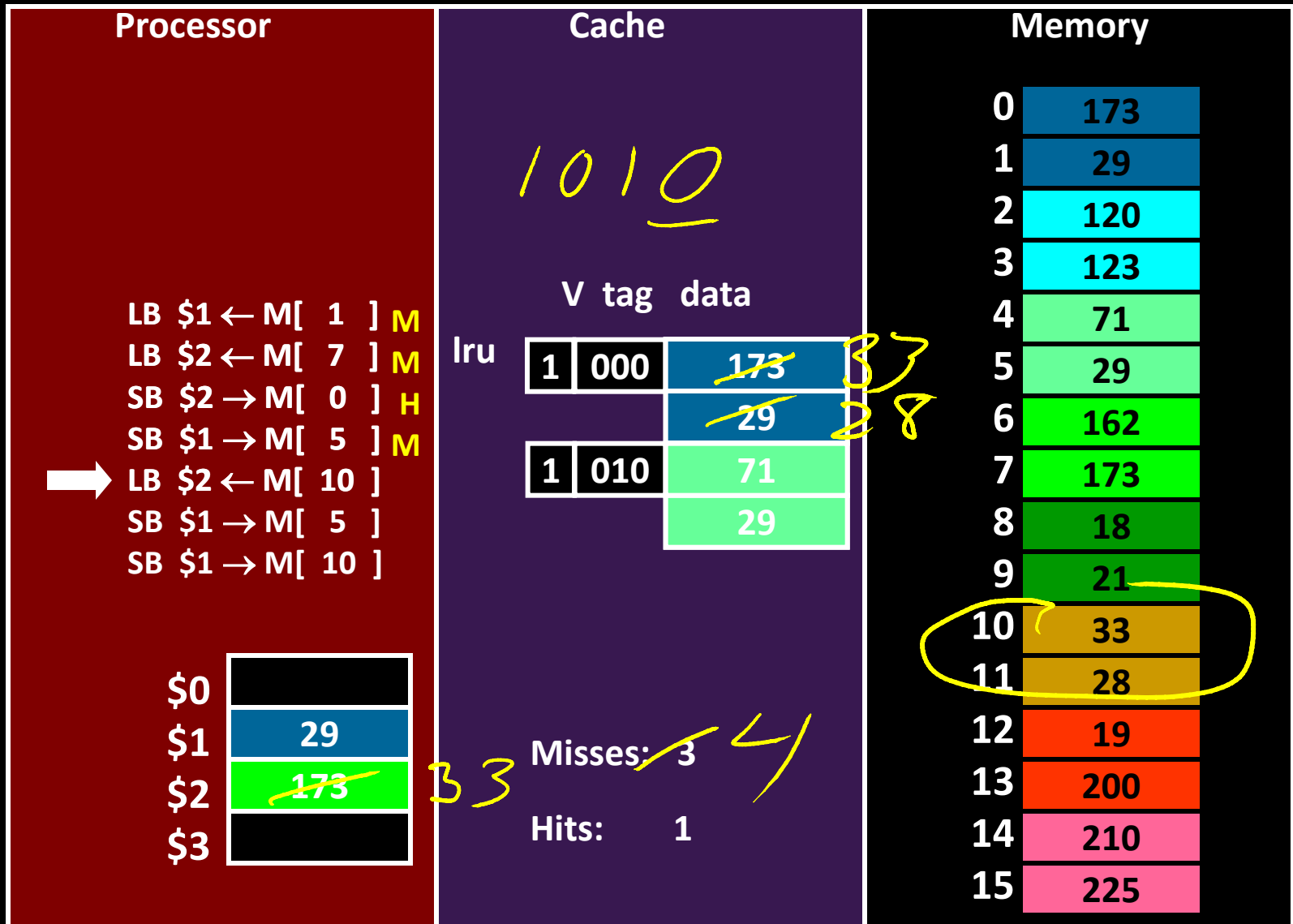
# Write-Through (REF 4)



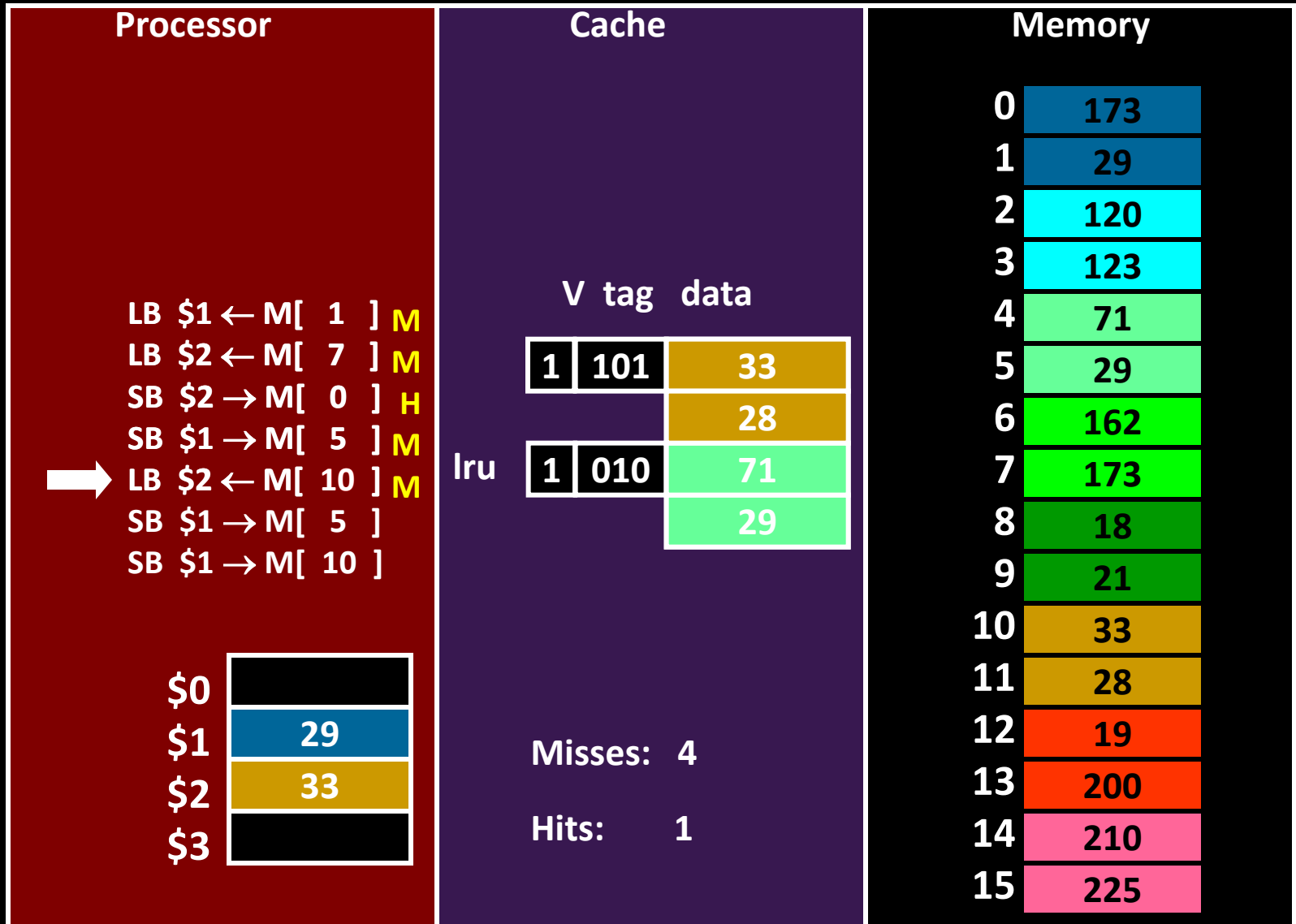
# Write-Through (REF 4)



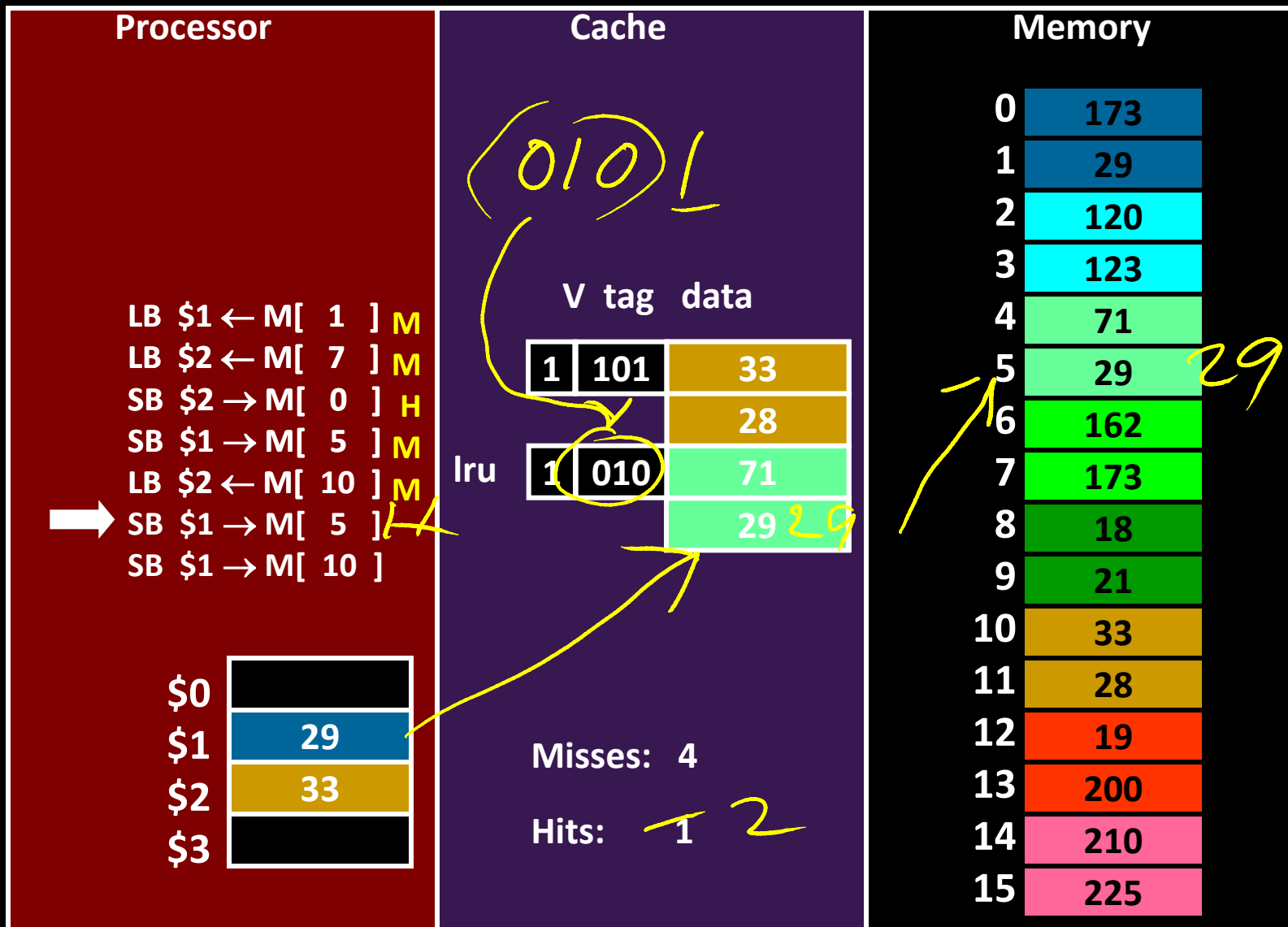
# Write-Through (REF 5)



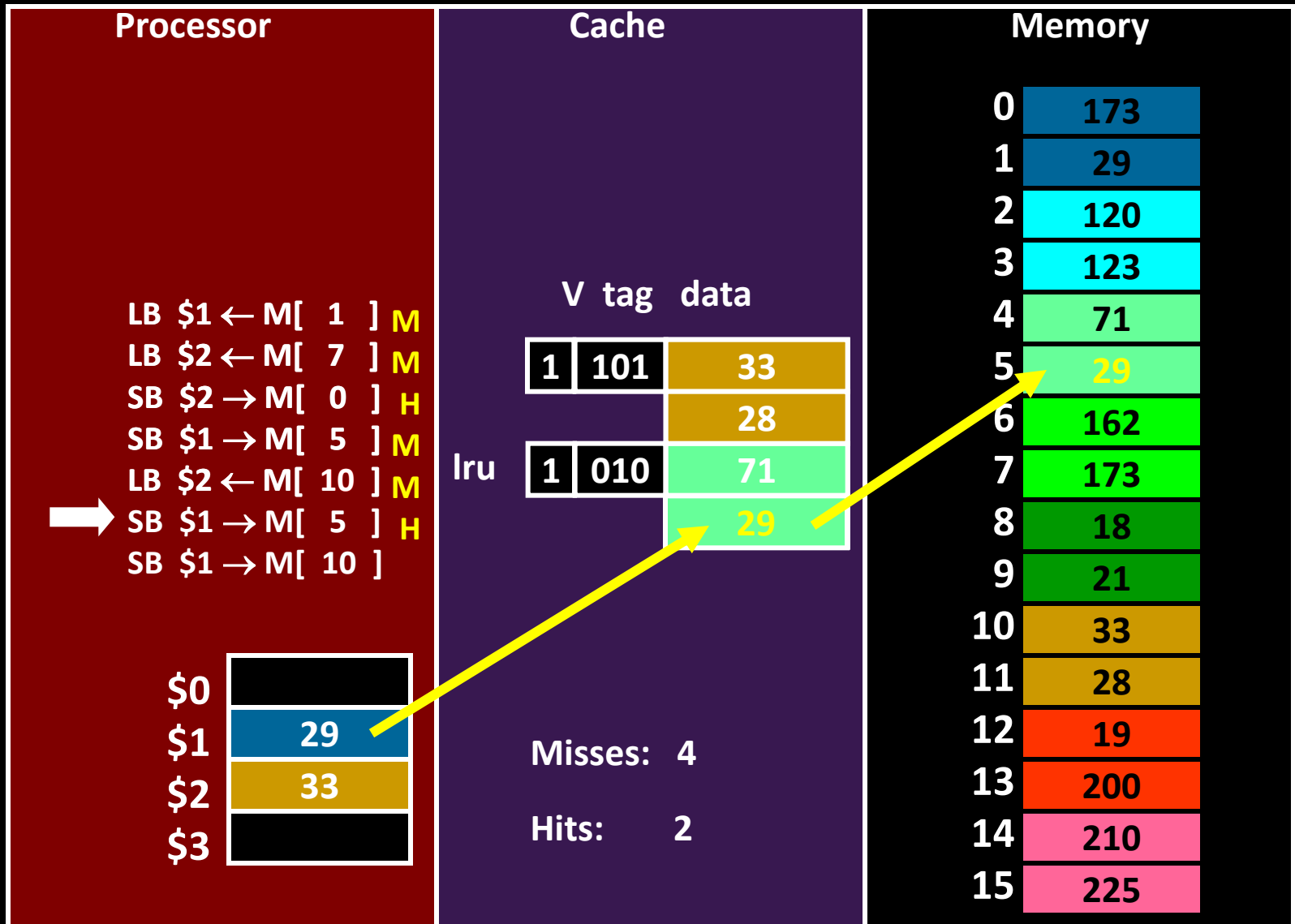
# Write-Through (REF 5)



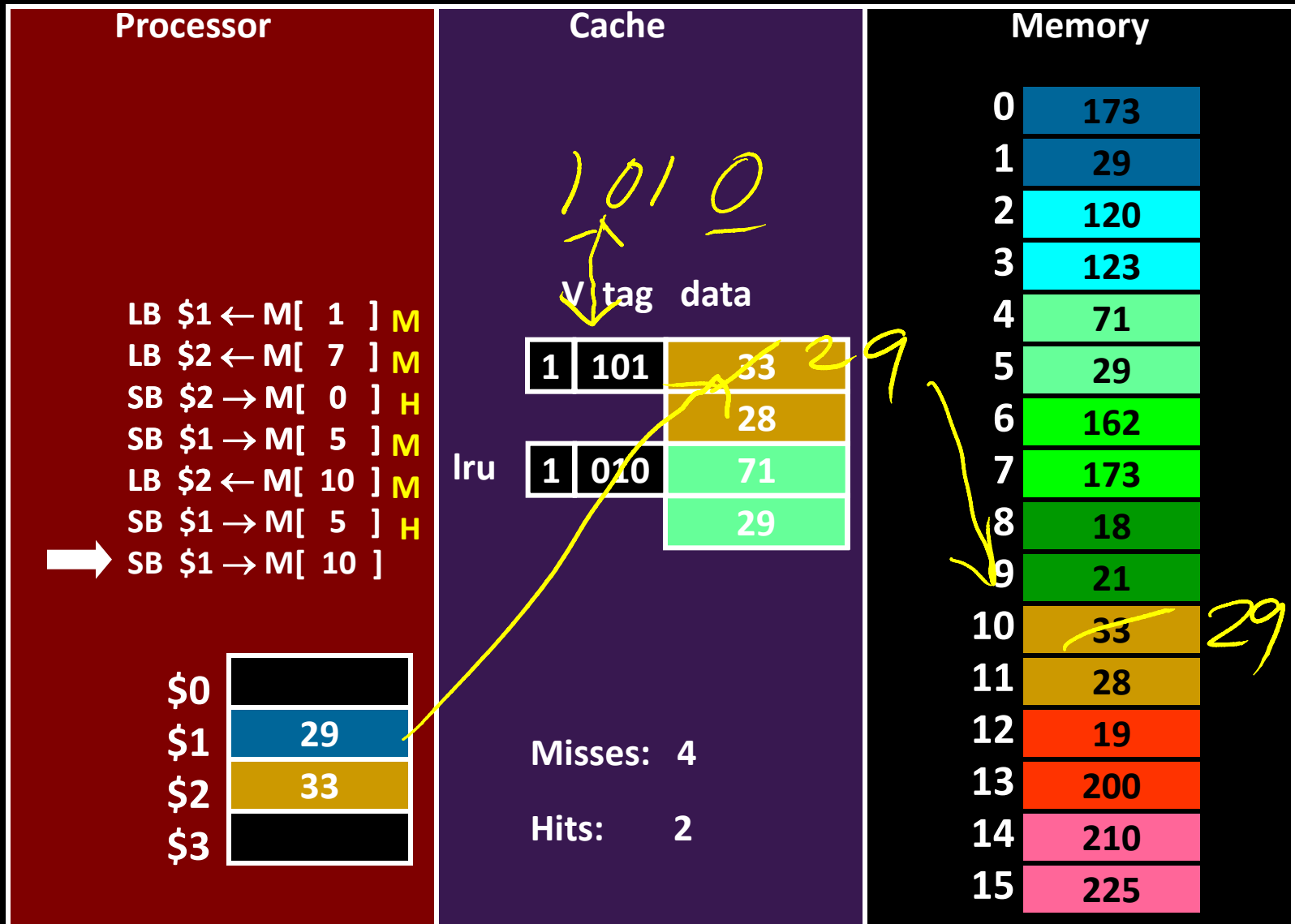
# Write-Through (REF 6)



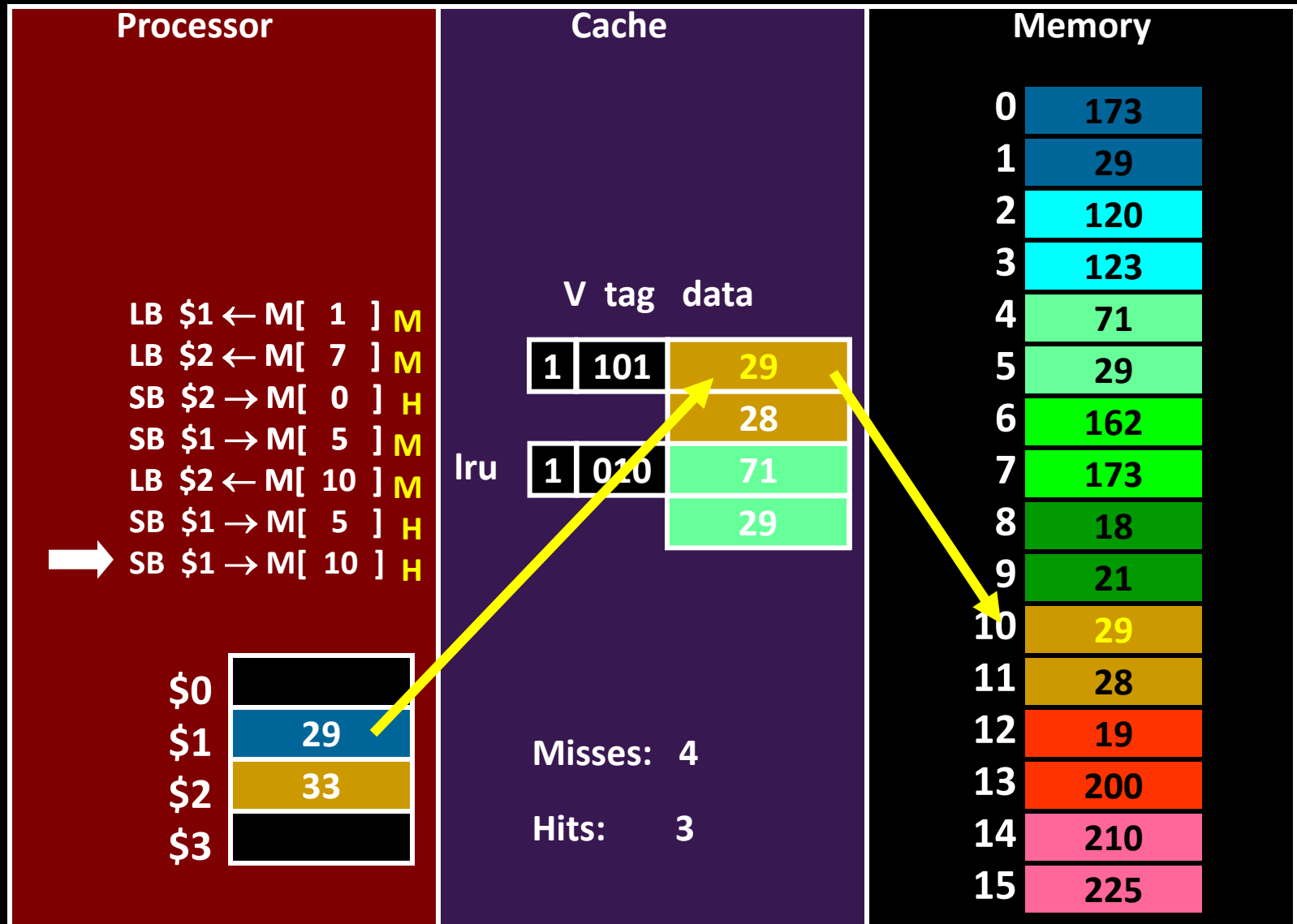
# Write-Through (REF 6)



# Write-Through (REF 7)



# Write-Through (REF 7)





# How Many Memory References?

Write-through performance

Each miss (read or write) reads a **block** from mem

- 4 misses → 8 mem reads

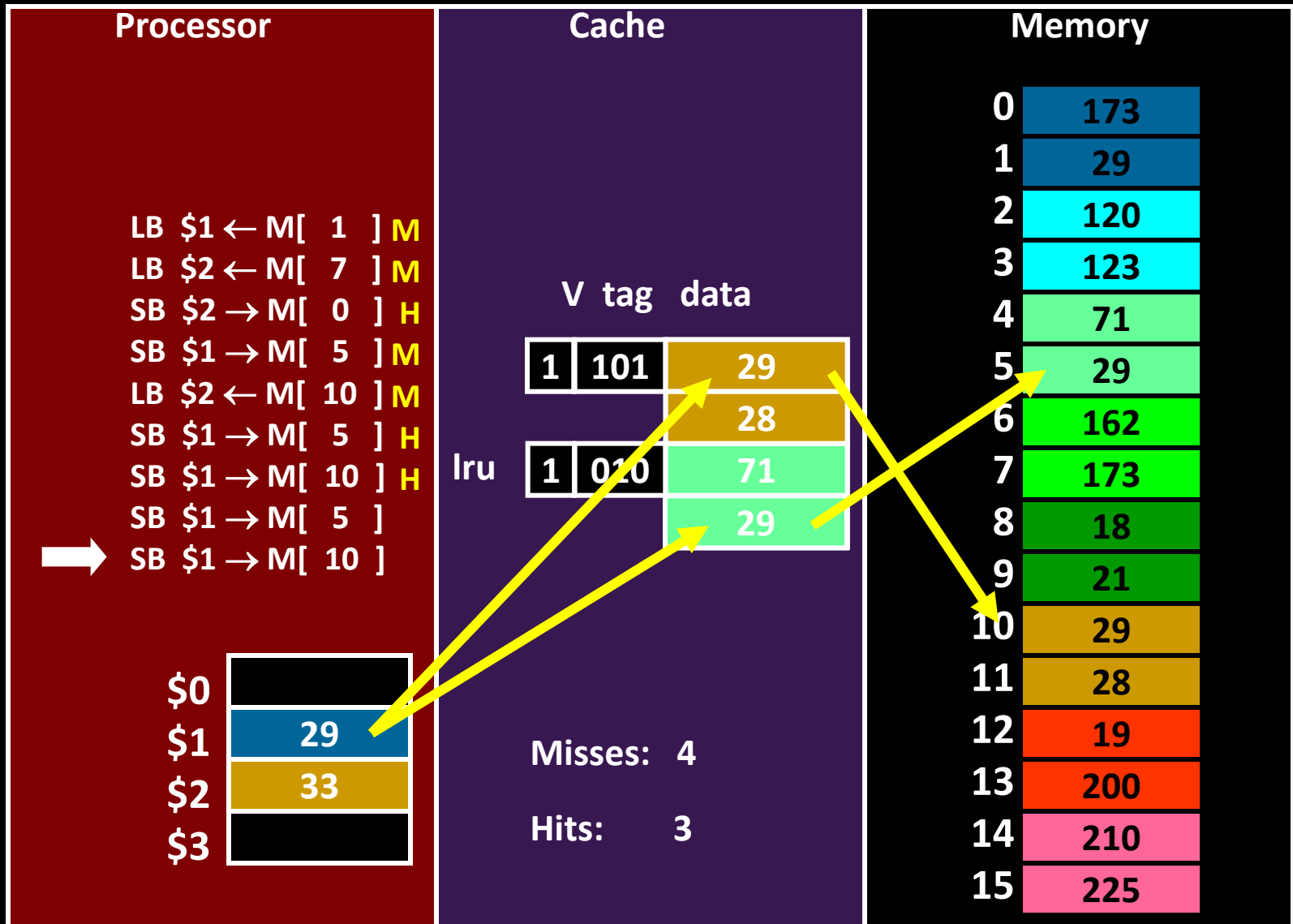
Each store writes an **item** to mem

- 4 mem writes

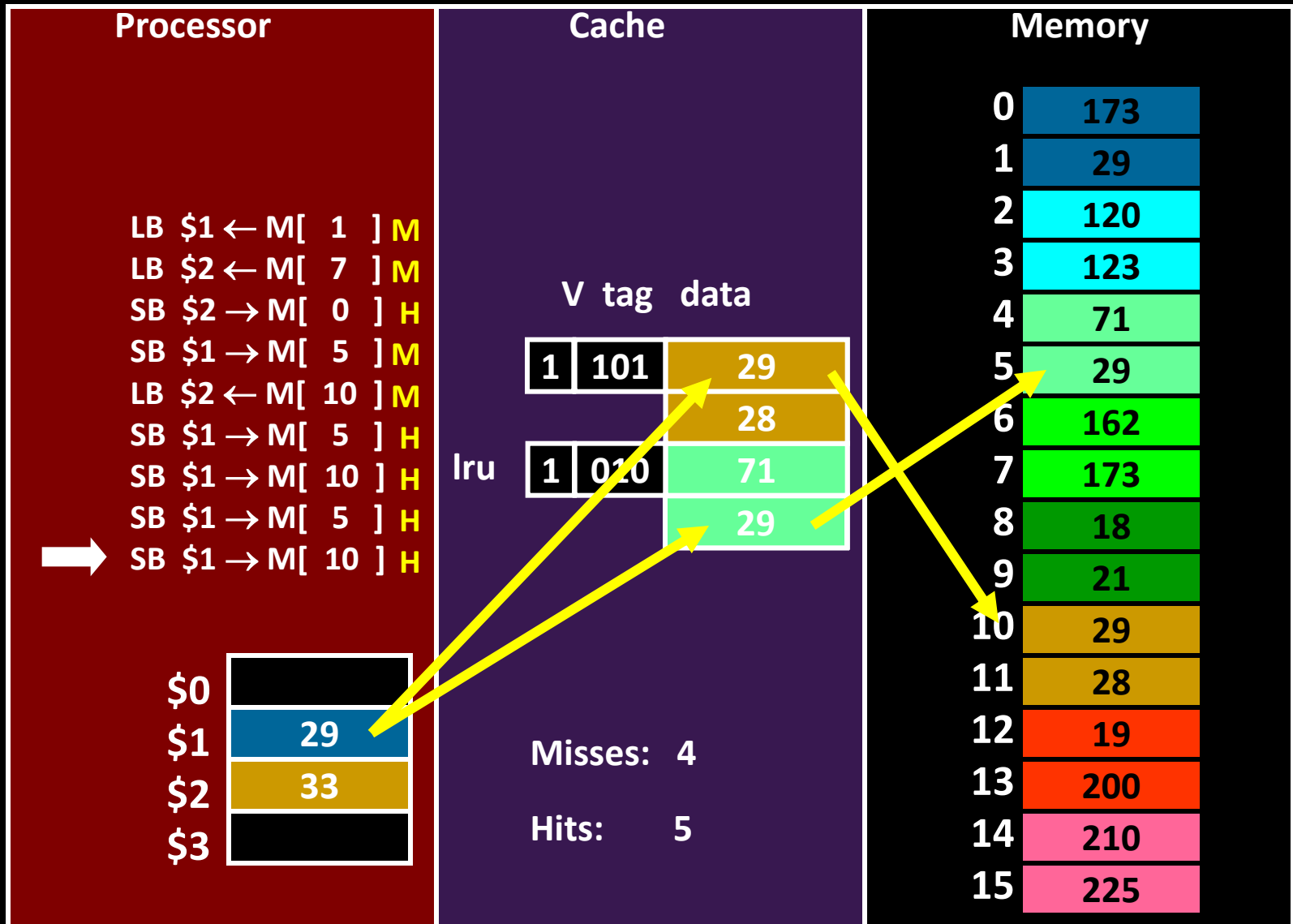
Evictions don't need to write to mem

- no need for dirty bit

# Write-Through (REF 8,9)



# Write-Through (REF 8,9)



# Write-Through vs. Write-Back

---

Can we also design the cache **NOT** to write all stores immediately to memory?

Keep the most current copy in cache, and update memory when that data is evicted (**write-back policy**)

Do we need to write-back all evicted lines?

No, only blocks that have been stored into (written)

# Write-Back Meta-Data

V	D	Tag	Byte 1	Byte 2	... Byte N

V = 1 means the line has valid data

D = 1 means the bytes are newer than main memory

When allocating line:

- Set V = 1, D = 0, fill in Tag and Data

When writing line:

- Set D = 1

When evicting line:

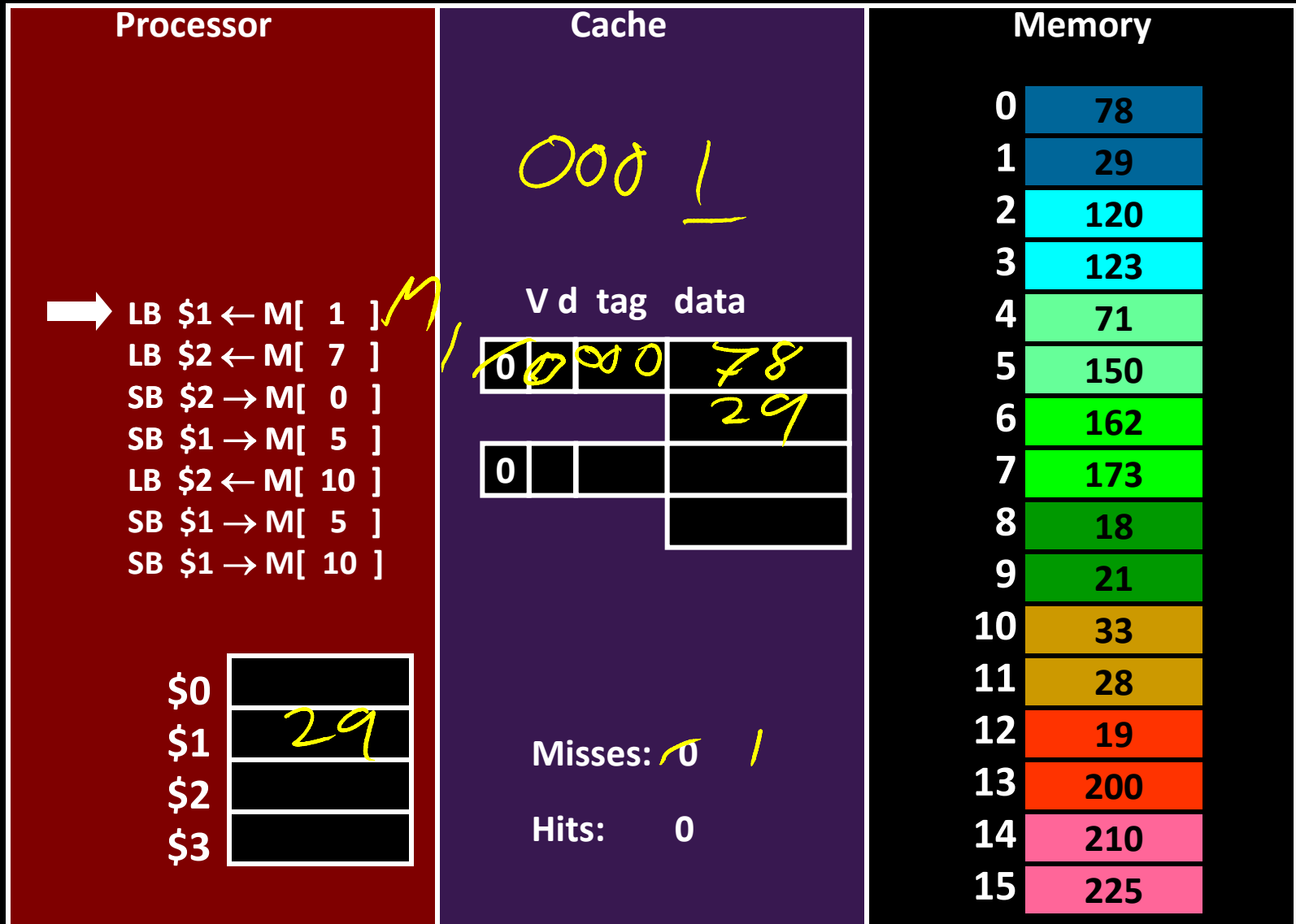
- If D = 0: just set V = 0
- If D = 1: write-back Data, then set D = 0, V = 0

# Handling Stores (Write-Back)

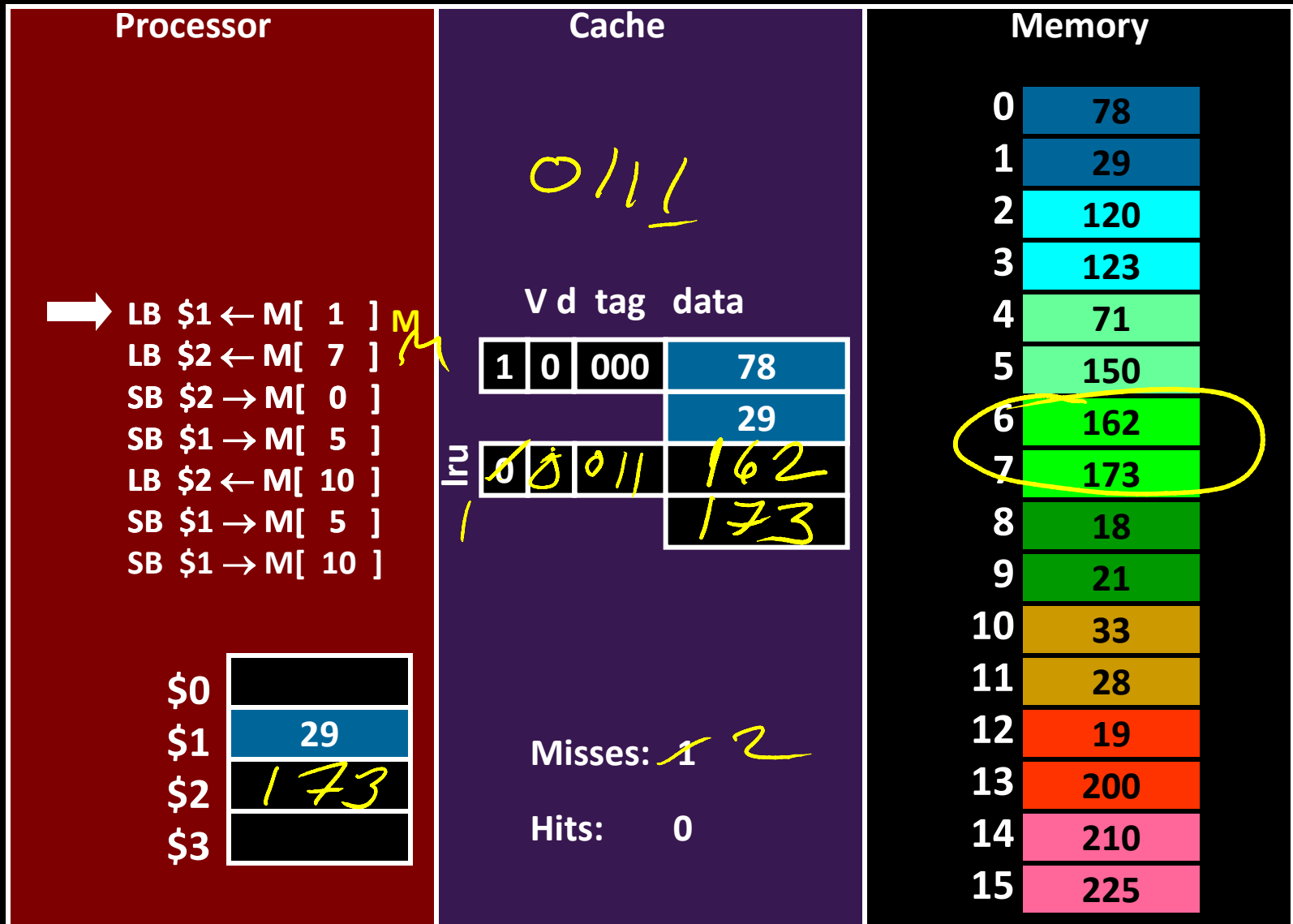
Using **byte addresses** in this example! Addr Bus = 4 bits

Processor	Cache Fully Associative Cache	Memory																																																
<p><b>Assume write-allocate policy</b></p> <p>LB \$1 ← M[ 1 ]            LB \$2 ← M[ 7 ]            SB \$2 → M[ 0 ]            SB \$1 → M[ 5 ]            LB \$2 ← M[ 10 ]            SB \$1 → M[ 5 ]            SB \$1 → M[ 10 ]</p> <table style="margin-left: auto; margin-right: auto;"> <tr><td style="background-color: #800000; color: white;">\$0</td><td style="width: 40px; height: 20px;"></td></tr> <tr><td style="background-color: #800000; color: white;">\$1</td><td style="width: 40px; height: 20px;"></td></tr> <tr><td style="background-color: #800000; color: white;">\$2</td><td style="width: 40px; height: 20px;"></td></tr> <tr><td style="background-color: #800000; color: white;">\$3</td><td style="width: 40px; height: 20px;"></td></tr> </table>	\$0		\$1		\$2		\$3		<p>2 cache lines  <u>2</u> word block  <u>3</u> bit tag field  <u>1</u> bit block offset field</p> <p style="text-align: center;">V d tag data</p> <table style="margin-left: auto; margin-right: auto;"> <tr><td style="border: 1px solid white; padding: 2px;">0</td><td style="border: 1px solid white; width: 20px; height: 20px;"></td><td style="border: 1px solid white; width: 20px; height: 20px;"></td><td style="border: 1px solid white; width: 20px; height: 20px;"></td></tr> <tr><td style="border: 1px solid white; padding: 2px;">0</td><td style="border: 1px solid white; width: 20px; height: 20px;"></td><td style="border: 1px solid white; width: 20px; height: 20px;"></td><td style="border: 1px solid white; width: 20px; height: 20px;"></td></tr> </table> <p style="margin-top: 20px;">Misses: 0 Hits: 0</p>	0				0				<table style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 30px;">0</td><td style="background-color: #0056b3; color: white; text-align: center;">78</td></tr> <tr><td>1</td><td style="background-color: #0056b3; color: white; text-align: center;">29</td></tr> <tr><td>2</td><td style="background-color: #00ffff; color: white; text-align: center;">120</td></tr> <tr><td>3</td><td style="background-color: #00ffff; color: white; text-align: center;">123</td></tr> <tr><td>4</td><td style="background-color: #90ee90; color: white; text-align: center;">71</td></tr> <tr><td>5</td><td style="background-color: #90ee90; color: white; text-align: center;">150</td></tr> <tr><td>6</td><td style="background-color: #00ff00; color: white; text-align: center;">162</td></tr> <tr><td>7</td><td style="background-color: #00ff00; color: white; text-align: center;">173</td></tr> <tr><td>8</td><td style="background-color: #228b22; color: white; text-align: center;">18</td></tr> <tr><td>9</td><td style="background-color: #228b22; color: white; text-align: center;">21</td></tr> <tr><td>10</td><td style="background-color: #d4af37; color: white; text-align: center;">33</td></tr> <tr><td>11</td><td style="background-color: #d4af37; color: white; text-align: center;">28</td></tr> <tr><td>12</td><td style="background-color: #ff4500; color: white; text-align: center;">19</td></tr> <tr><td>13</td><td style="background-color: #ff4500; color: white; text-align: center;">200</td></tr> <tr><td>14</td><td style="background-color: #ff69b4; color: white; text-align: center;">210</td></tr> <tr><td>15</td><td style="background-color: #ff69b4; color: white; text-align: center;">225</td></tr> </table>	0	78	1	29	2	120	3	123	4	71	5	150	6	162	7	173	8	18	9	21	10	33	11	28	12	19	13	200	14	210	15	225
\$0																																																		
\$1																																																		
\$2																																																		
\$3																																																		
0																																																		
0																																																		
0	78																																																	
1	29																																																	
2	120																																																	
3	123																																																	
4	71																																																	
5	150																																																	
6	162																																																	
7	173																																																	
8	18																																																	
9	21																																																	
10	33																																																	
11	28																																																	
12	19																																																	
13	200																																																	
14	210																																																	
15	225																																																	

# Write-Back (REF 1)

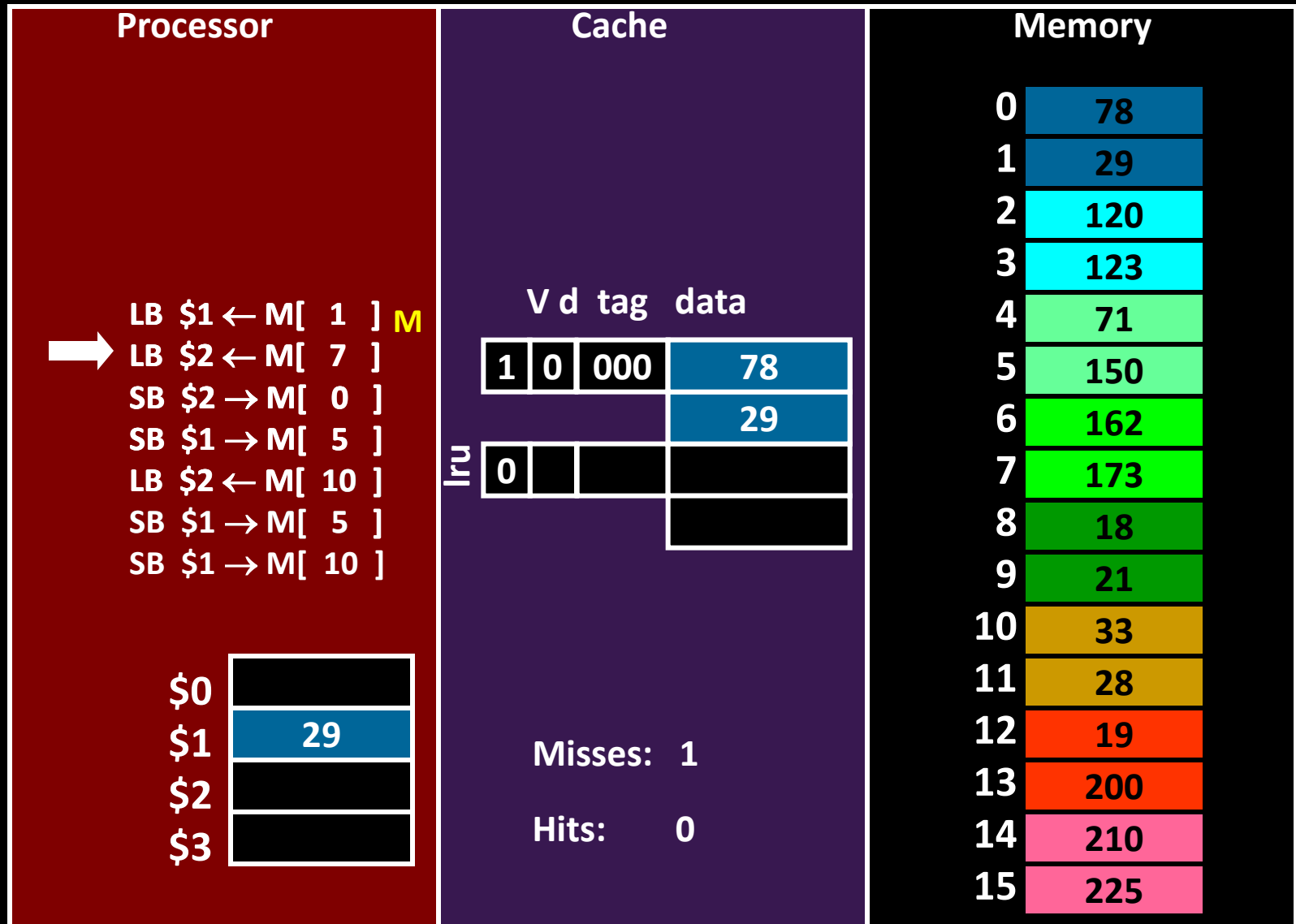


# Write-Back (REF 1)

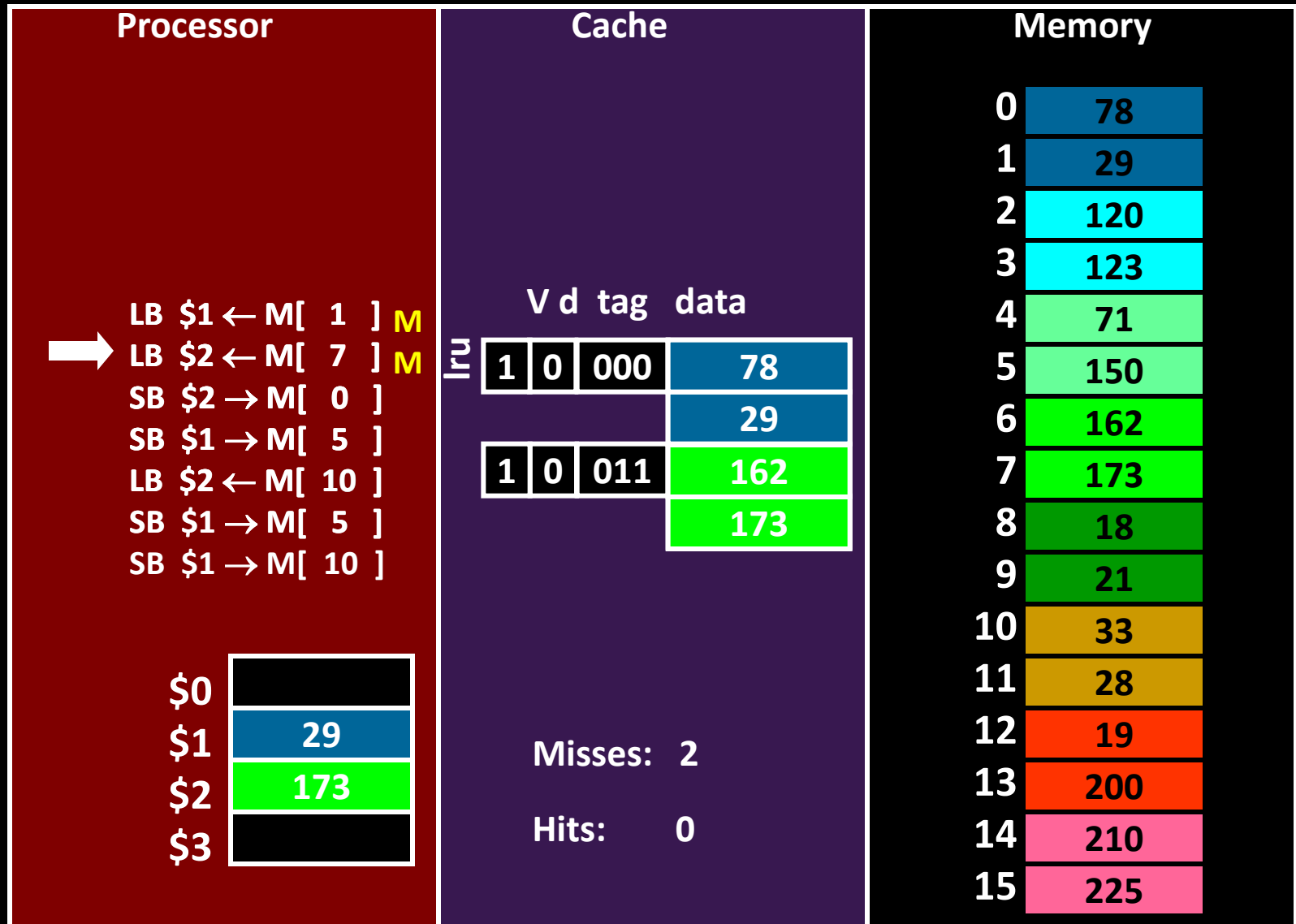




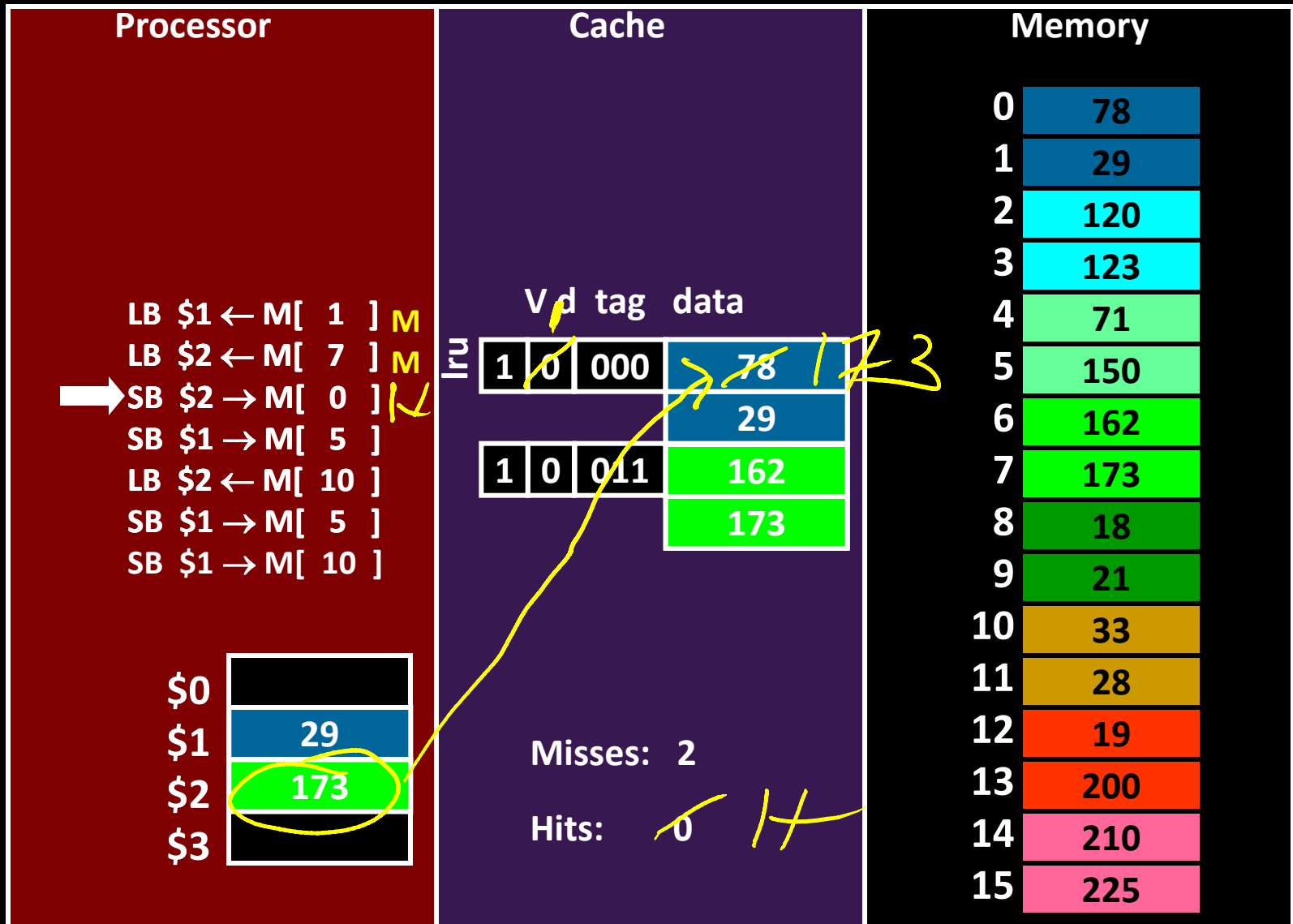
# Write-Back (REF 2)



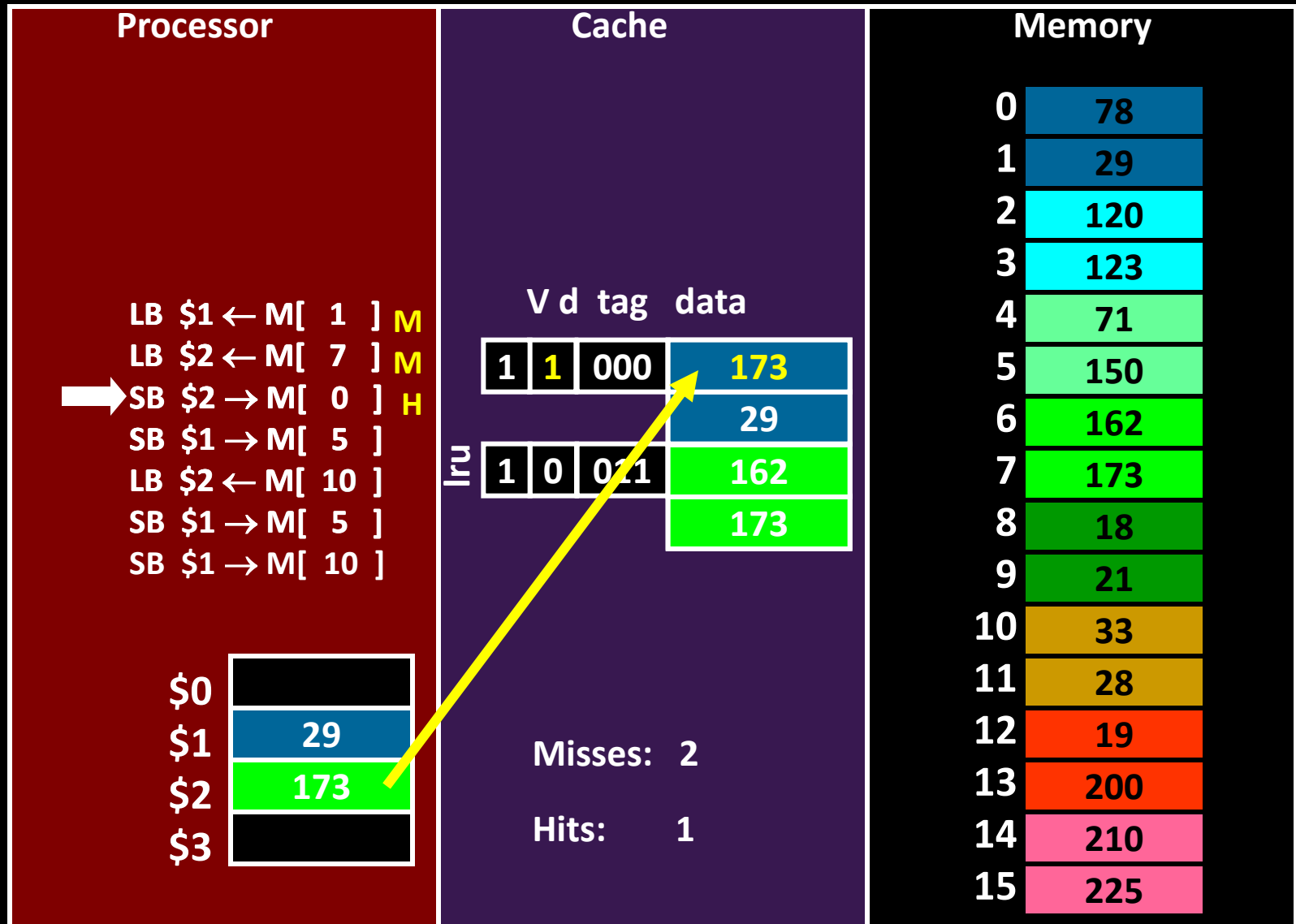
# Write-Back (REF 2)



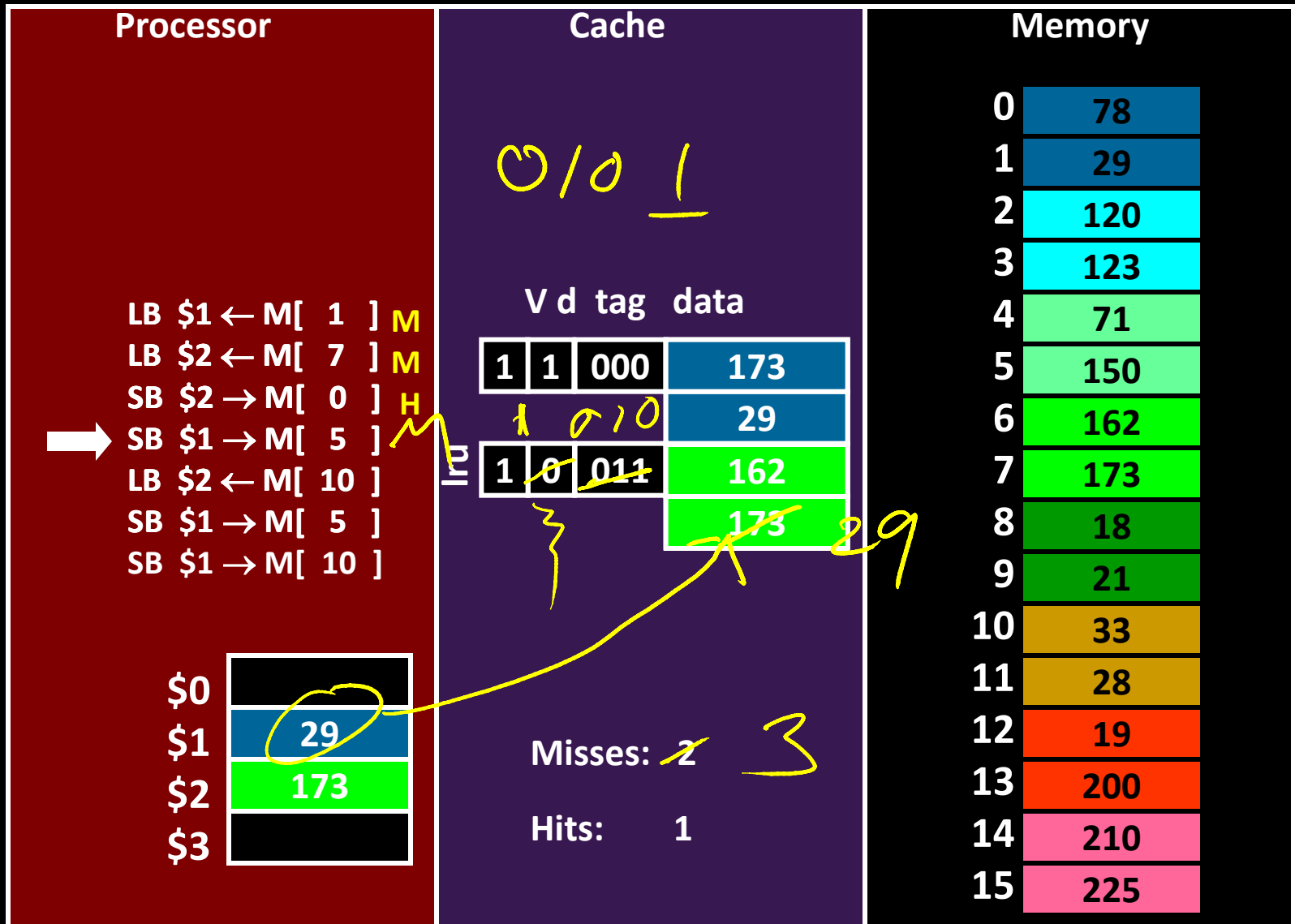
# Write-Back (REF 3)



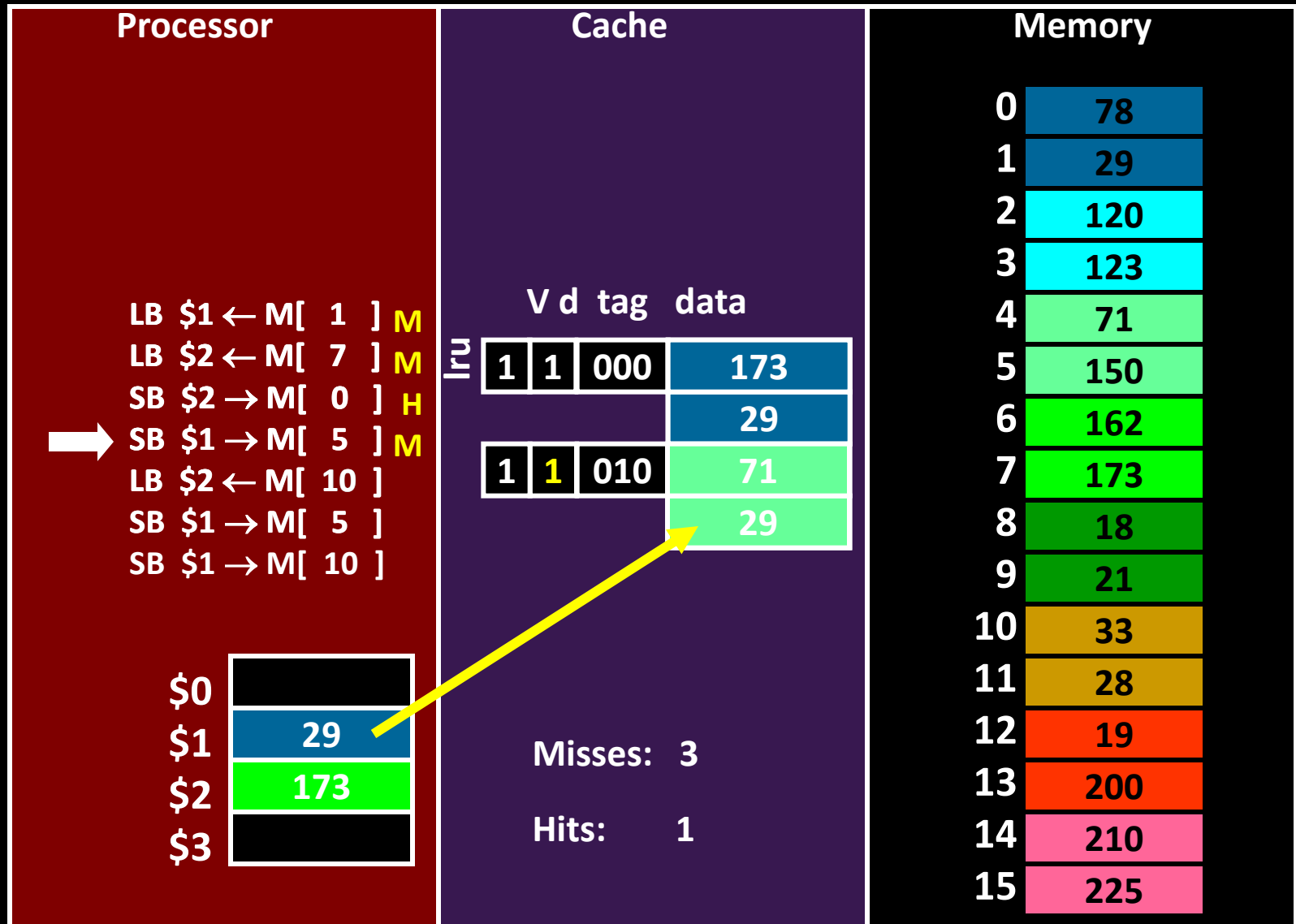
# Write-Back (REF 3)



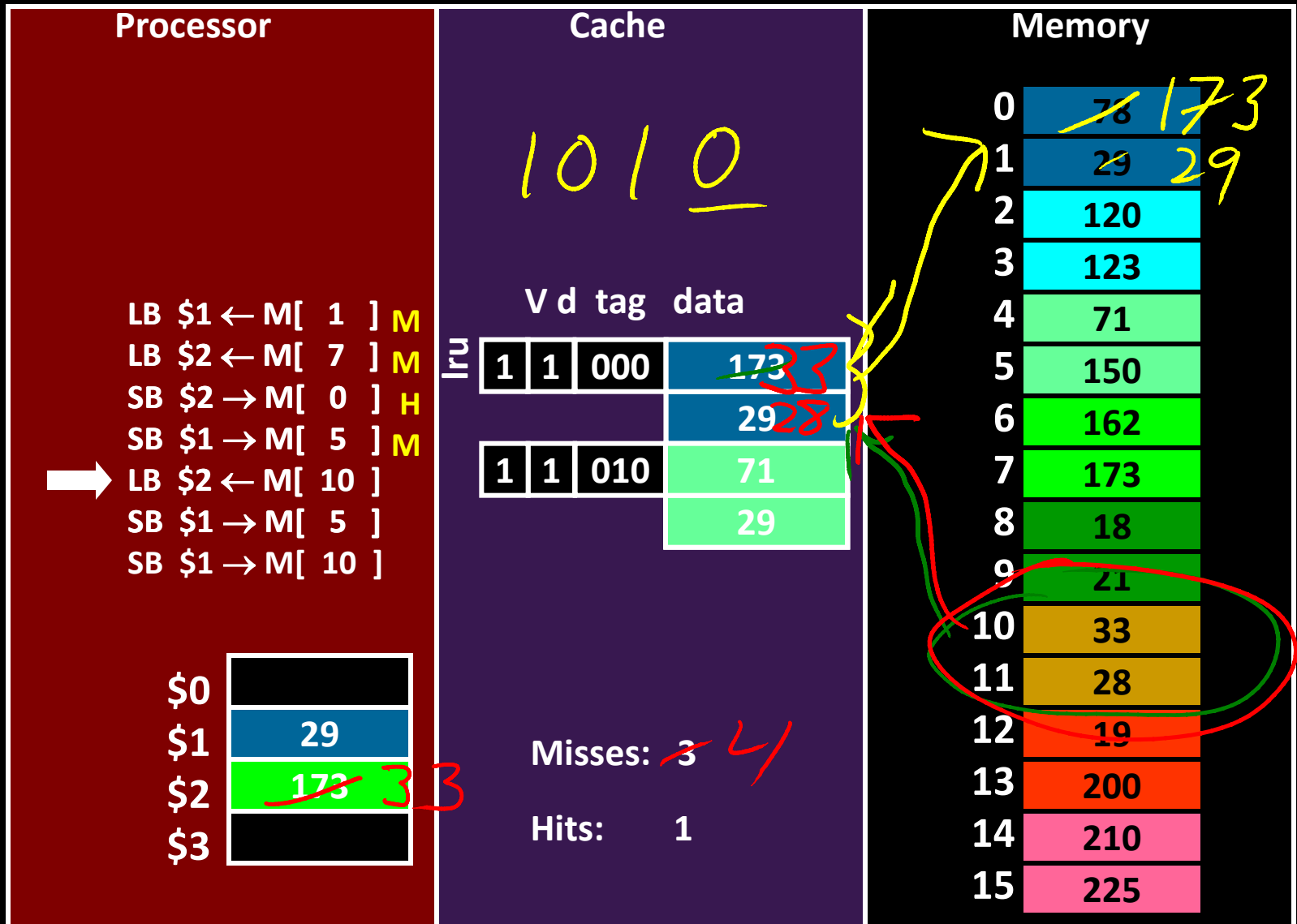
# Write-Back (REF 4)



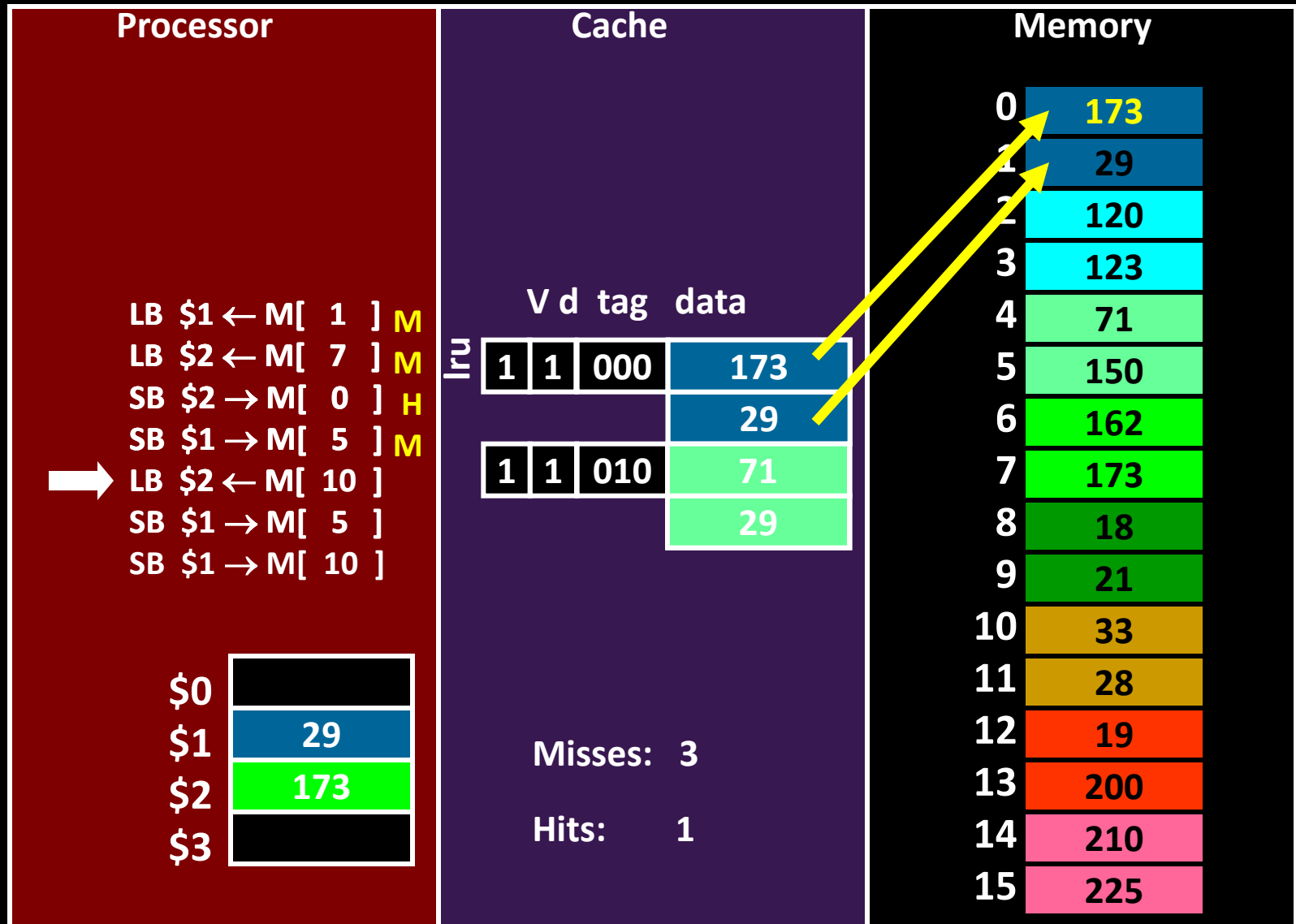
# Write-Back (REF 4)



# Write-Back (REF 5)

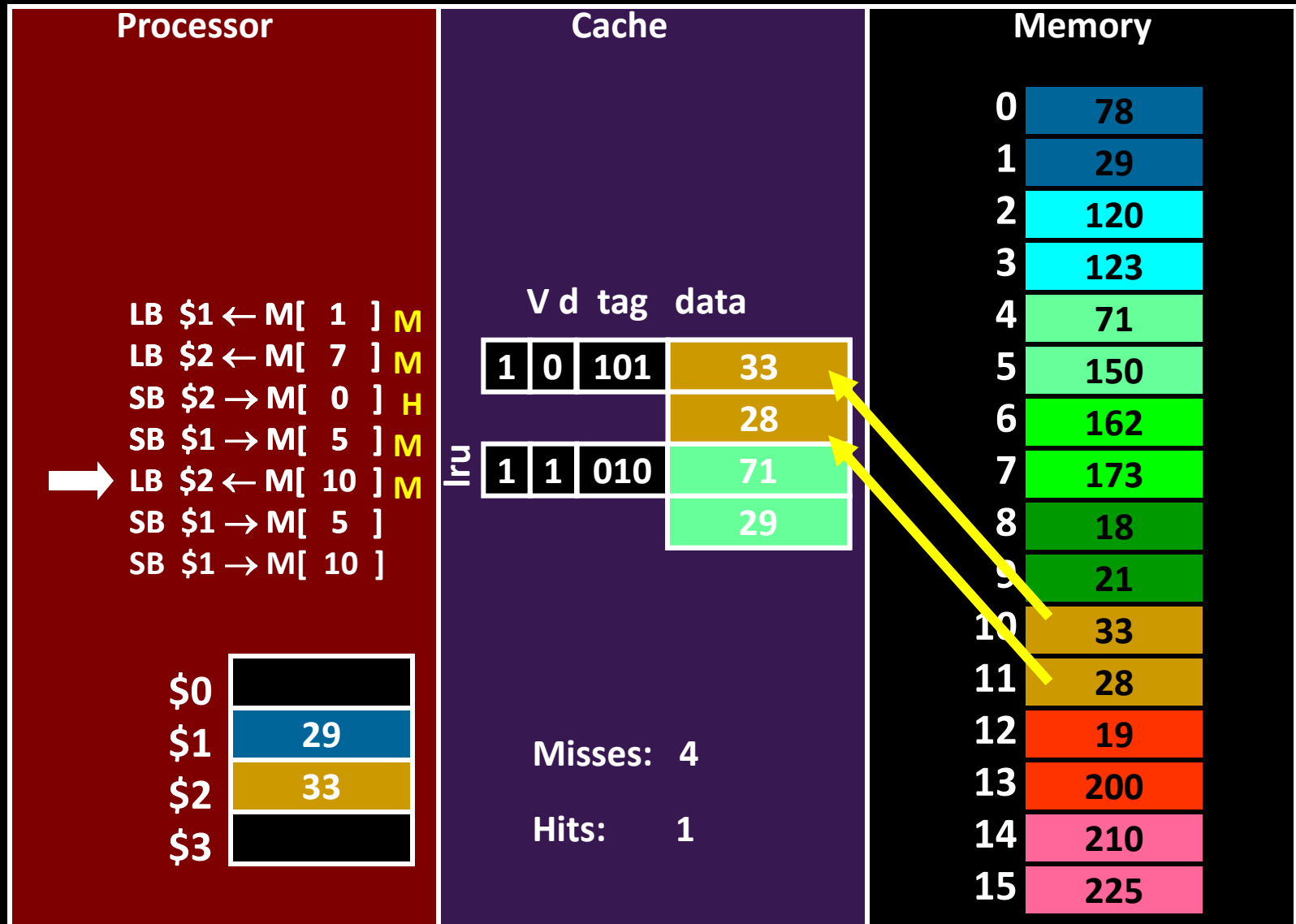


# Write-Back (REF 5)

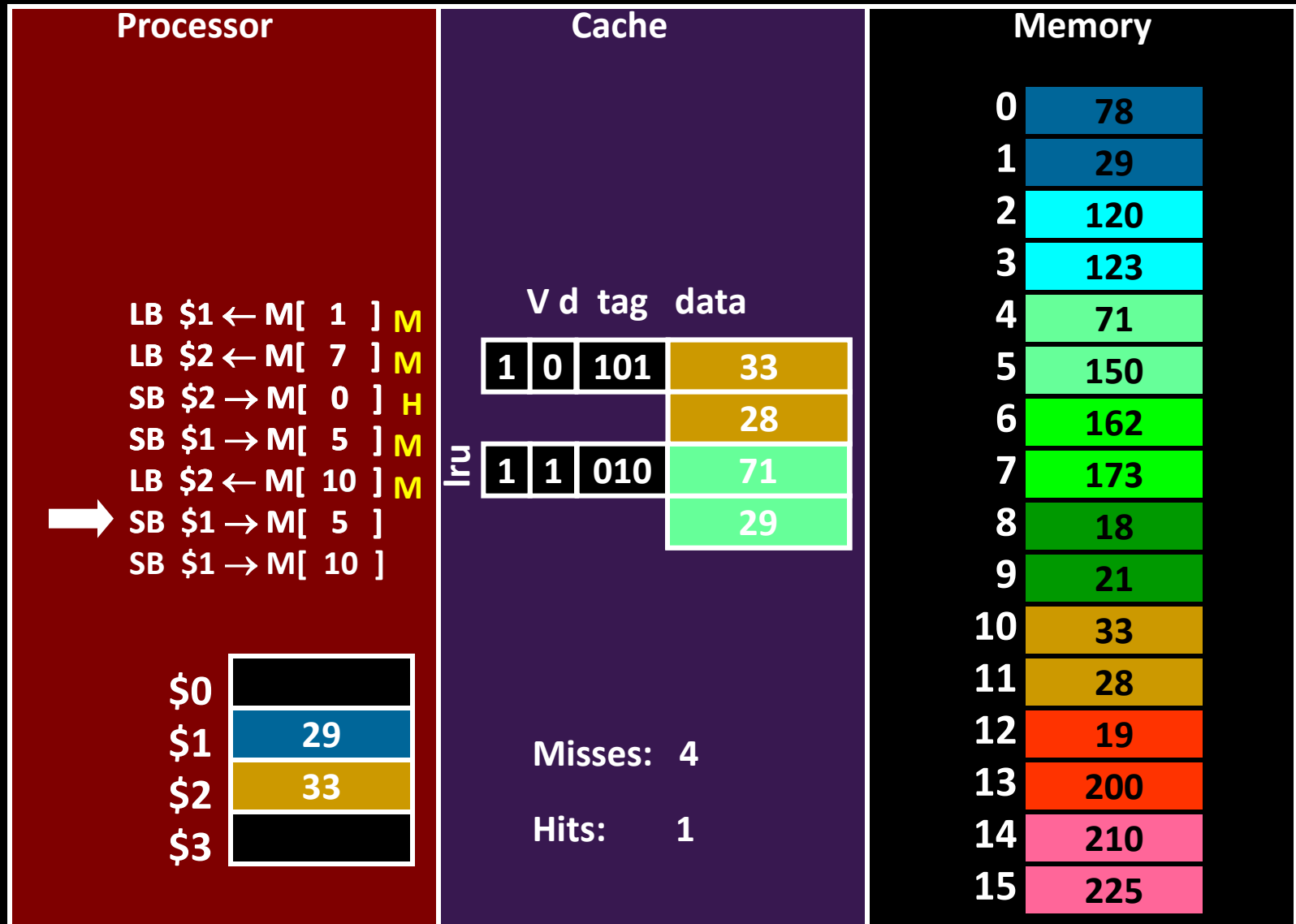




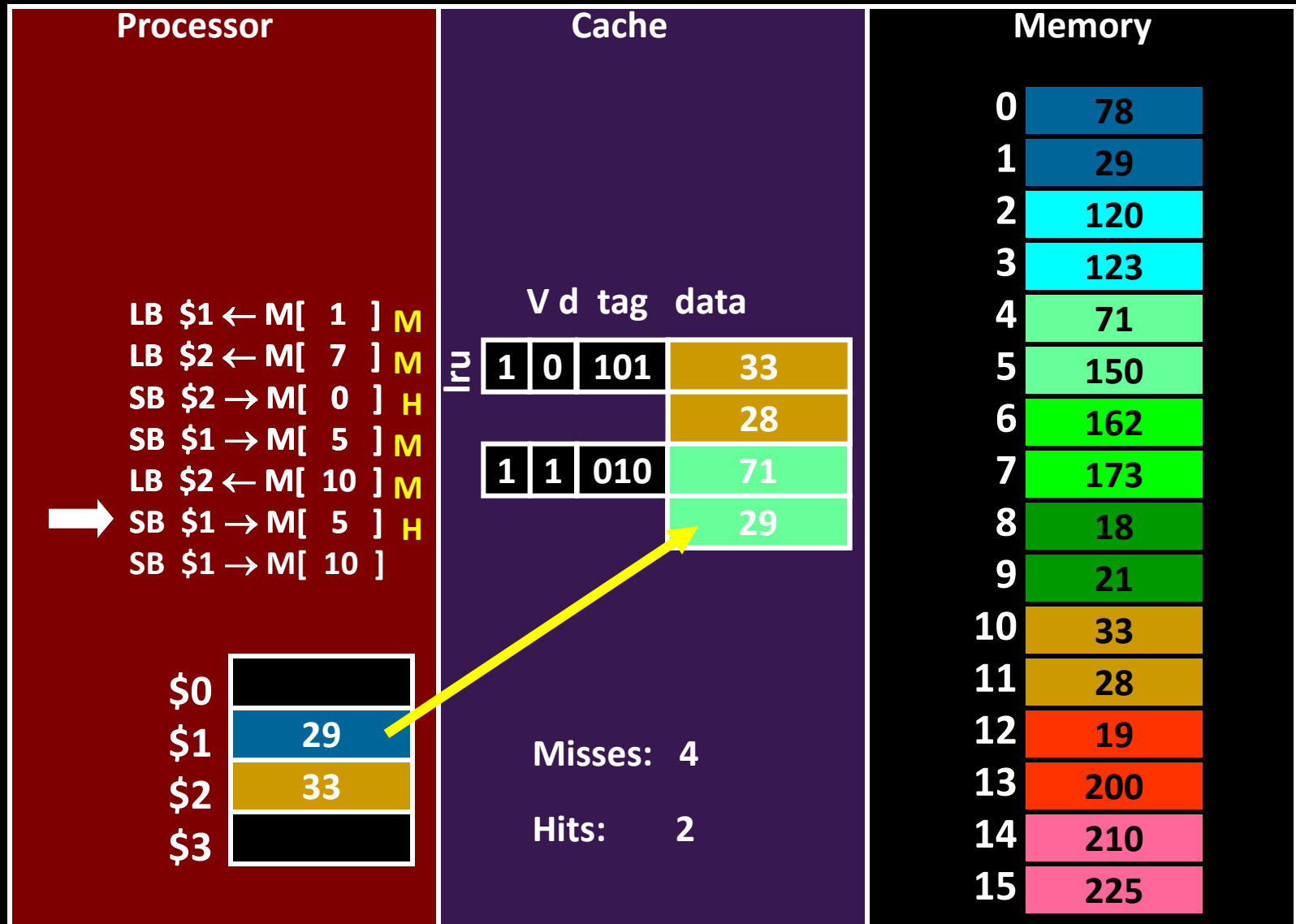
# Write-Back (REF 5)



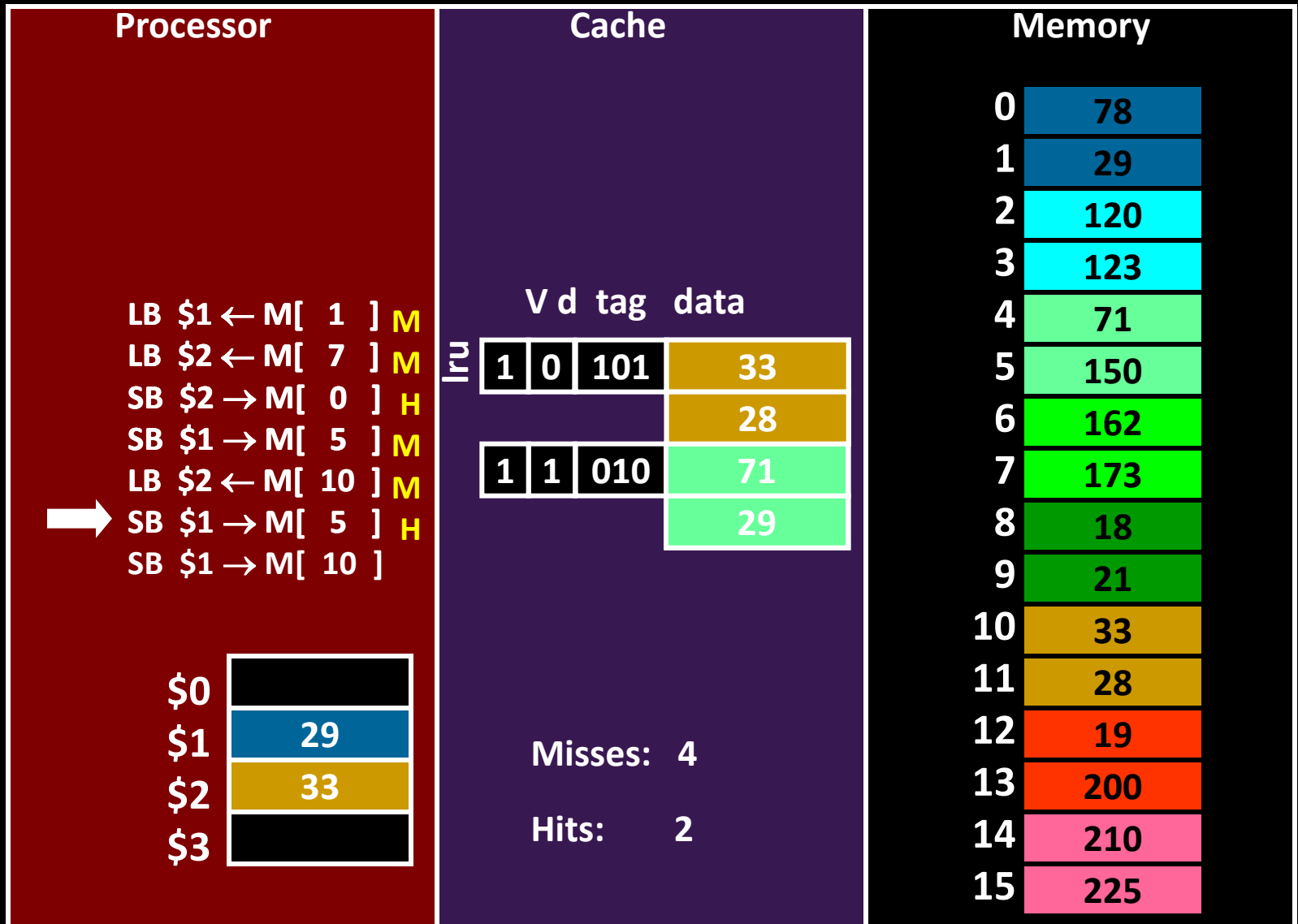
# Write-Back (REF 6)



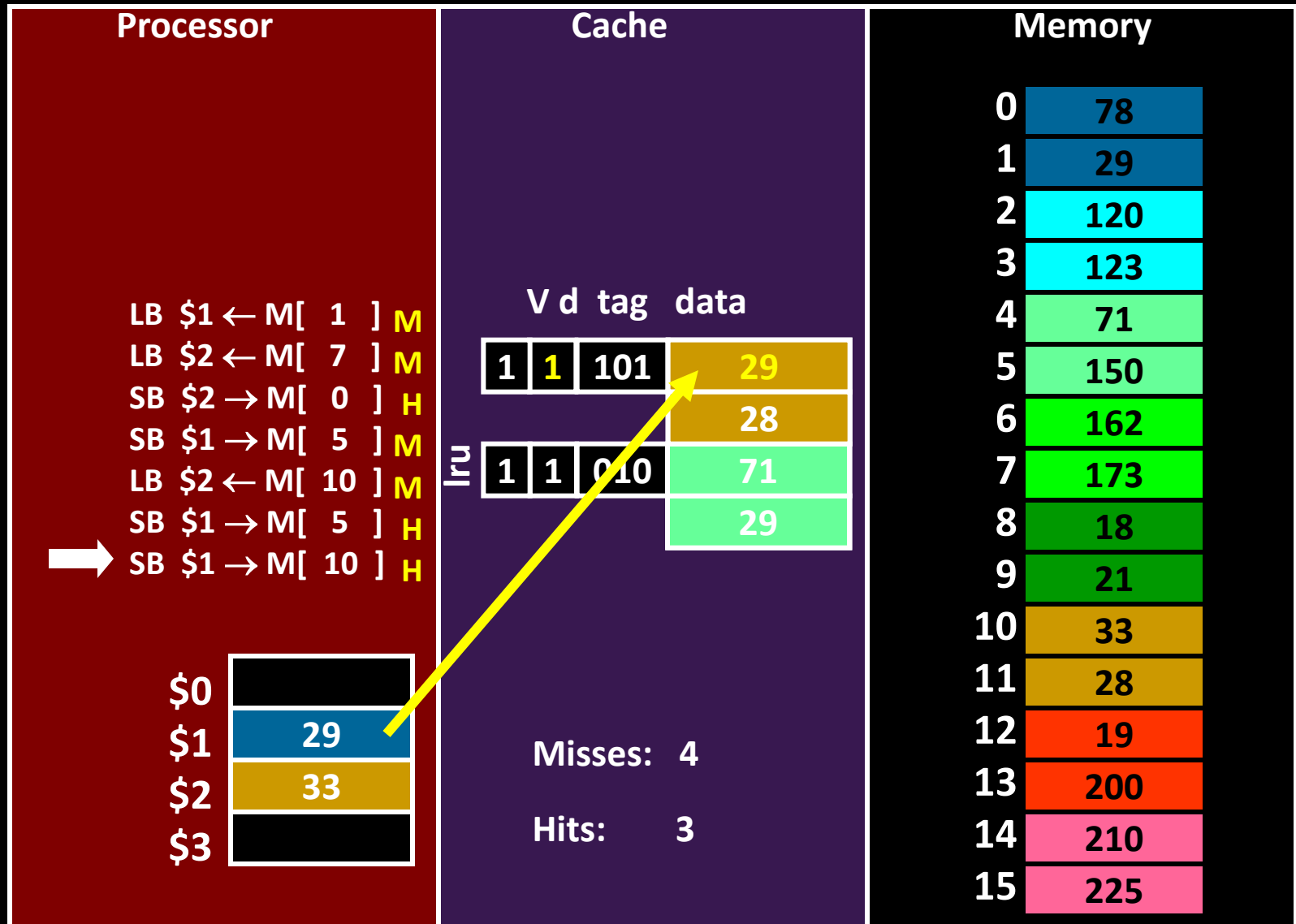
# Write-Back (REF 6)



# Write-Back (REF 7)



# Write-Back (REF 7)



# How Many Memory References?

Write-back performance

Each miss (read or write) reads a block from mem

- 4 misses → 8 mem reads

*Some* evictions write a block to mem

- 1 dirty eviction → 2 mem writes
- (+ 2 dirty evictions later → +4 mem writes)

# How many memory references?

Each miss reads a block

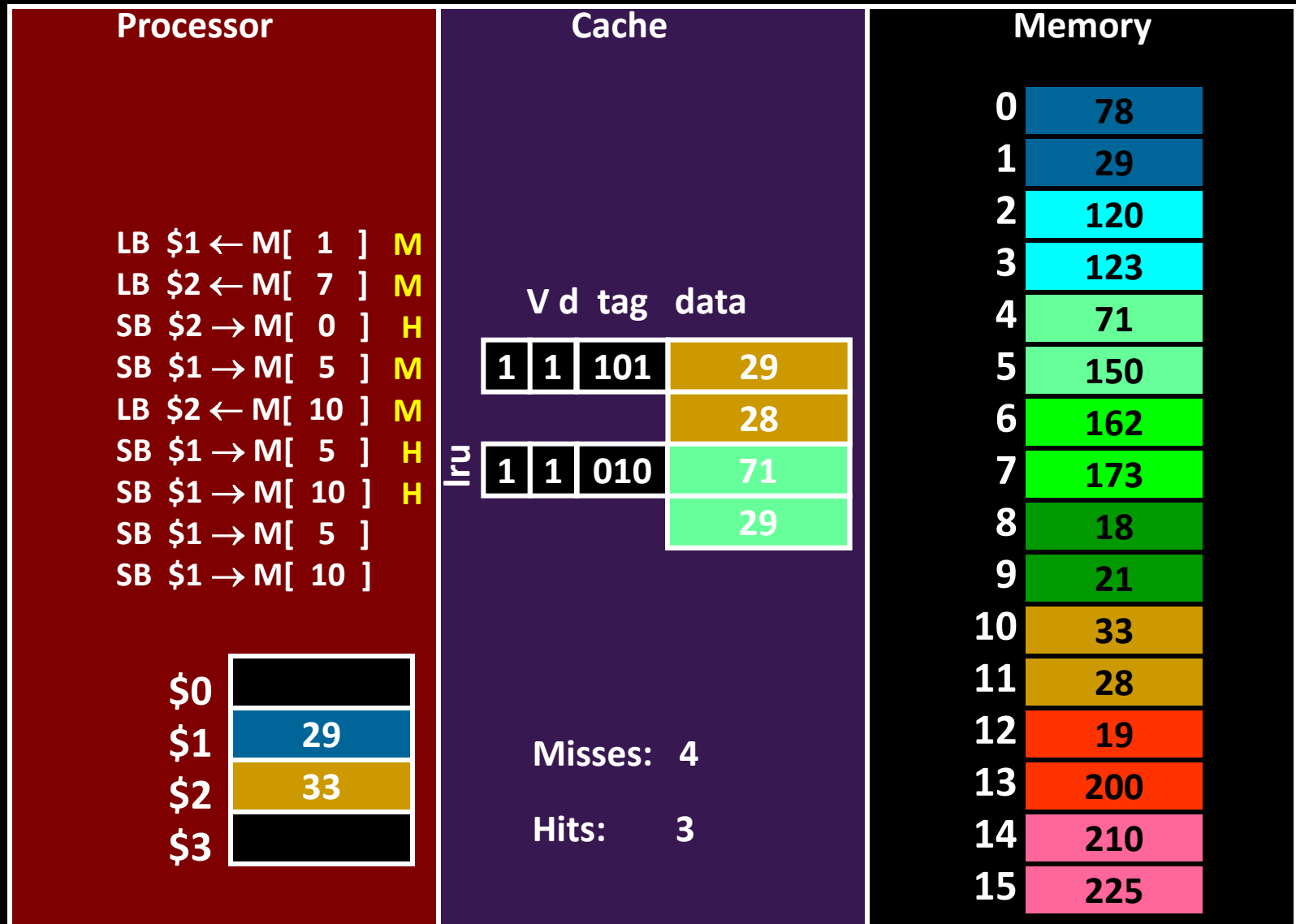
Two words in this cache

Each evicted dirty cache line writes a block

Total reads: six words

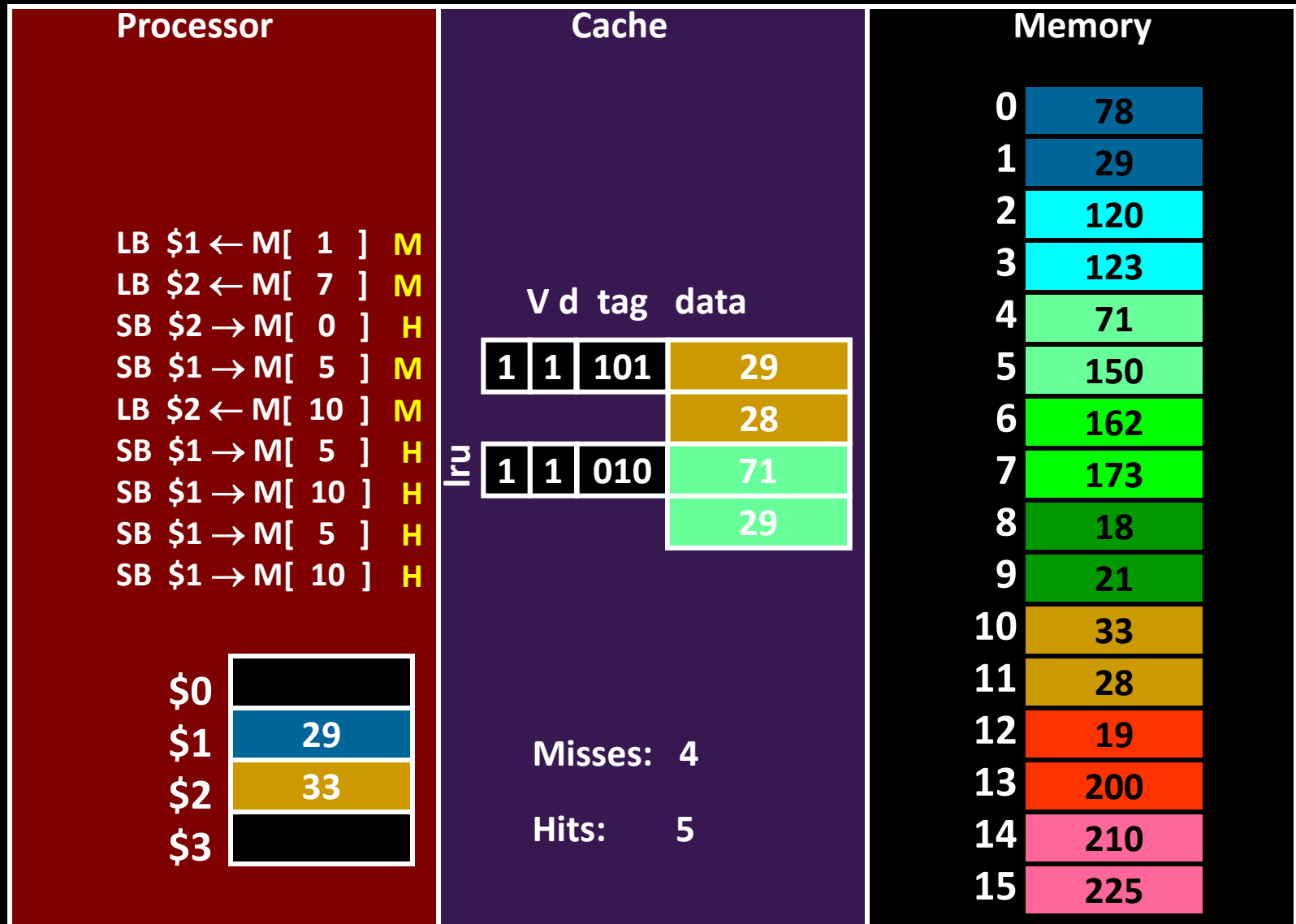
Total writes: 4/6 words (after final eviction)

# Write-Back (REF 8,9)





# Write-Back (REF 8,9)



# How Many Memory References?

Write-back performance

Each miss (read or write) reads a block from mem

- 4 misses → 8 mem reads

*Some* evictions write a block to mem

- 1 dirty eviction → 2 mem writes
- (+ 2 dirty evictions later → +4 mem writes)

By comparison write-through was

- Reads: eight words
- Writes: 4/6/8 etc words
- Write-through or Write-back?

# Write-through vs. Write-back

---

Write-through is slower

- But cleaner (memory always consistent)

Write-back is faster

- But complicated when multi cores sharing memory

# Performance: An Example

Performance: Write-back versus Write-through

Assume: large associative cache, 16 byte lines

```
for (i=1; i<n; i++)
```

```
    A[0] += A[i];
```

*Handwritten notes:*  
WT:  $n/16$  rds  
WP:  $n$  wrs  
WT:  $n$  rds  
1 wr

```
for (i=0; i<n; i++)
```

```
    B[i] = A[i]
```

*Handwritten notes:*  
WT:  $2 \times n / 16$  rds  
WP:  $2 \times n / 16$  rds  
Both:  $2 \times n$  wr

# Performance Tradeoffs

---

Q: Hit time: write-through vs. write-back?

A: Write-through slower on writes.

Q: Miss penalty: write-through vs. write-back?

A: Write-back slower on evictions.

# Write Buffering

Q: Writes to main memory are **slow!**

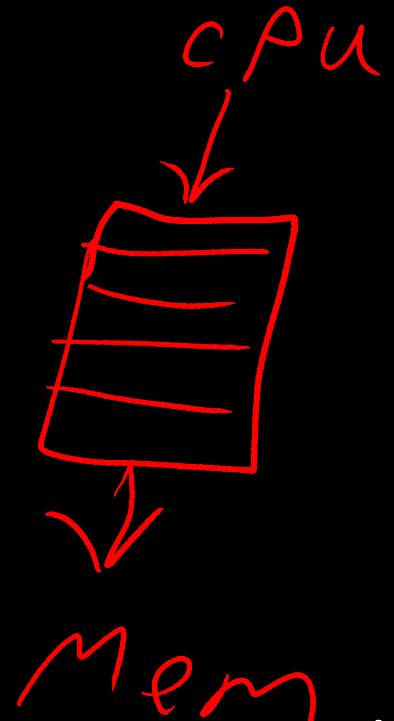
A: Use a **write-back buffer**

- A small queue holding dirty lines
- Add to end upon eviction
- Remove from front upon completion

Q: What does it help?

A: short bursts of writes (but not sustained writes)

A: fast eviction reduces miss penalty



# Write-through vs. Write-back

---

Write-through is slower

- But simpler (memory always consistent)

Write-back is almost always faster

- write-back buffer hides large eviction cost
- But what about multiple cores with separate caches but sharing memory?

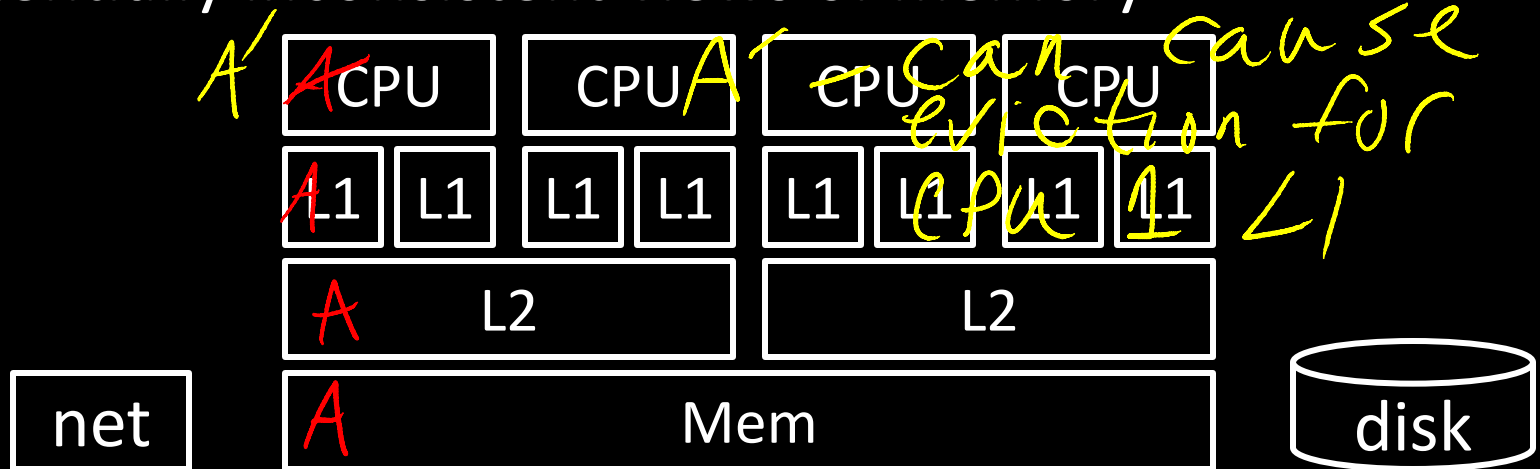
**Write-back requires a cache coherency protocol**

- Inconsistent views of memory
- Need to “snoop” in each other’s caches
- Extremely complex protocols, very hard to get right

# Cache-coherency

Q: Multiple readers and writers?

A: Potentially inconsistent views of memory



## Cache coherency protocol

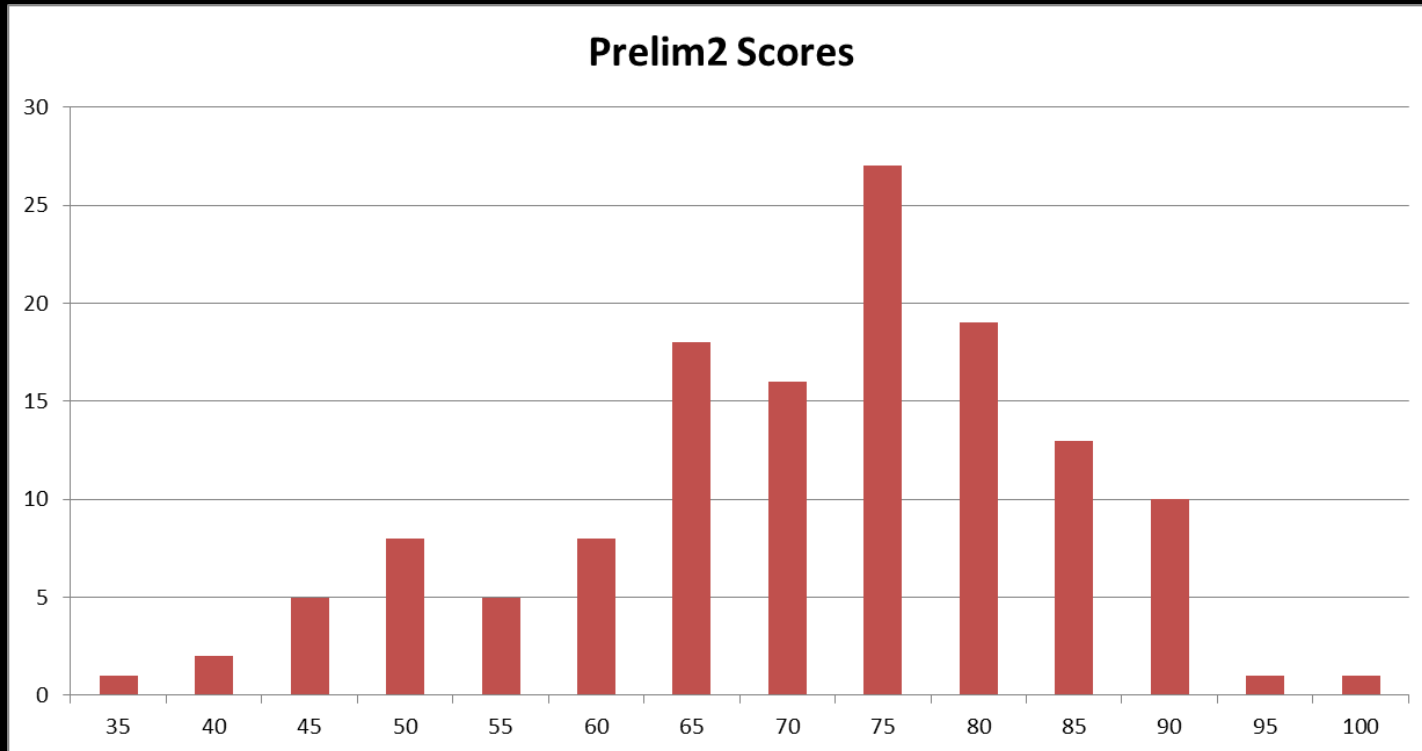
- May need to **snoop** on other CPU's cache activity
- **Invalidate** cache line when other CPU writes
- **Flush** write-back caches before other CPU reads
- Or the reverse: Before writing/reading...
- Extremely complex protocols, very hard to get right



# Administrivia

## Prelim2 results

- Mean 68.9 (median 71), standard deviation 13.0



- Prelims available in Upson 360 after today
- Regrade requires written request
  - ***Whole test is regraded***

# Administrivia

---

Lab3 due **next** Monday, April 9<sup>th</sup>

HW5 due **next** Tuesday, April 10<sup>th</sup>

# Summary

---

## Caching assumptions

- small working set: 90/10 rule
- can predict future: spatial & temporal locality

## Benefits

- (big & fast) built from (big & slow) + (small & fast)

## Tradeoffs:

associativity, line size, hit cost, miss penalty, hit rate

# Summary

---

## Memory performance matters!

- often more than CPU performance
- ... because it is the bottleneck, and not improving much
- ... because most programs move a LOT of data

## Design space is huge

- Gambling against program behavior
- Cuts across all layers:  
users → programs → os → hardware

## Multi-core / Multi-Processor is complicated

- Inconsistent views of memory
- Extremely complex protocols, very hard to get right