Caches 3

CS 3410, Spring 2014

Computer Science

Cornell University

See P&H Chapter: 5.1-5.4, 5.8, 5.15

Overview

Cache Organization

Writing to caches: policies, performance

Cache performance

Compromise

Set-associative cache

Like a direct-mapped cache

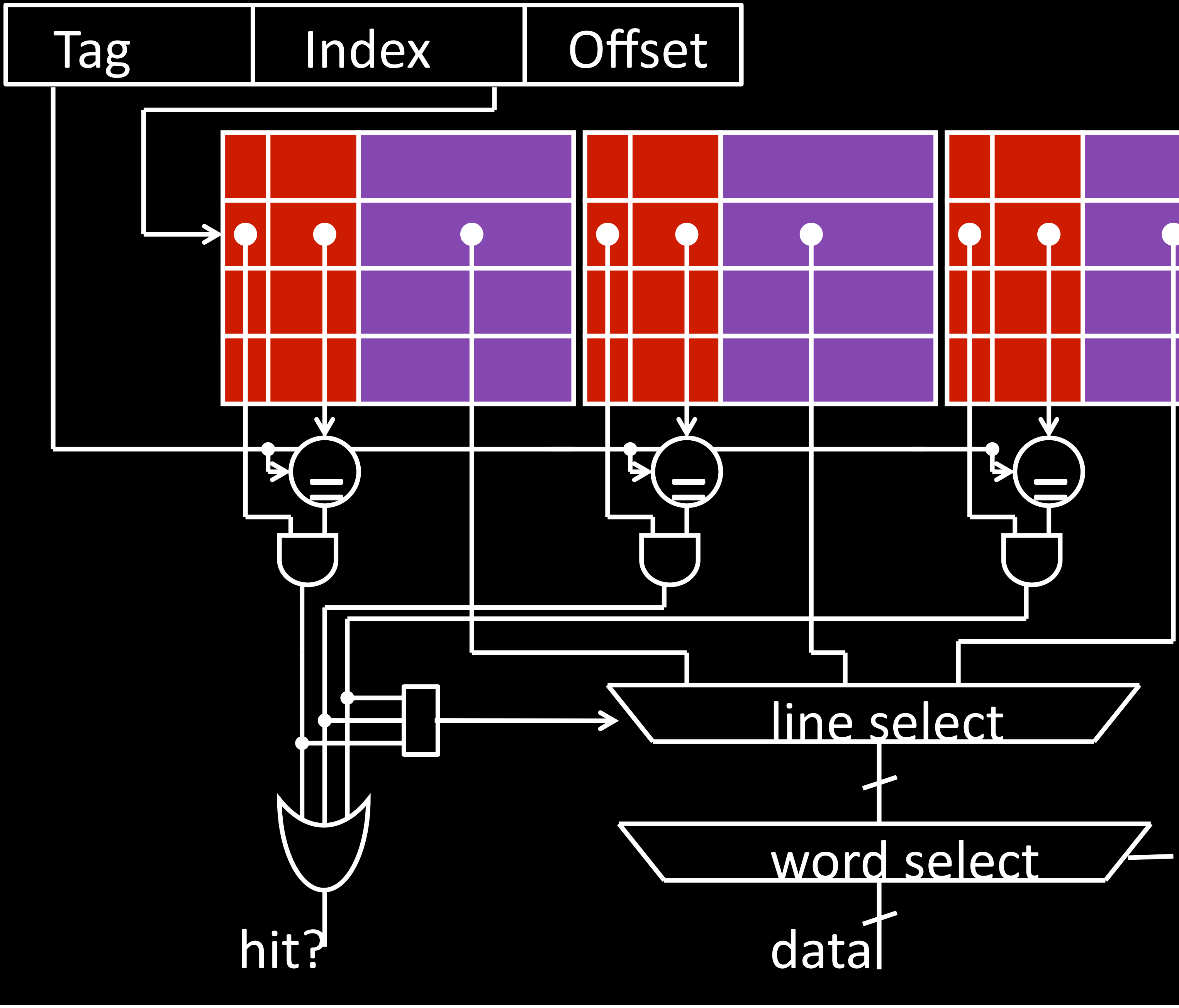
- Index into a location
- Fast

Like a fully-associative cache

- Can store multiple entries
  - decreases conflicts
- Search in each element

n-way set assoc means n possible locations

3-Way Set Associative Cache (Reading)



Basic Cache Organization

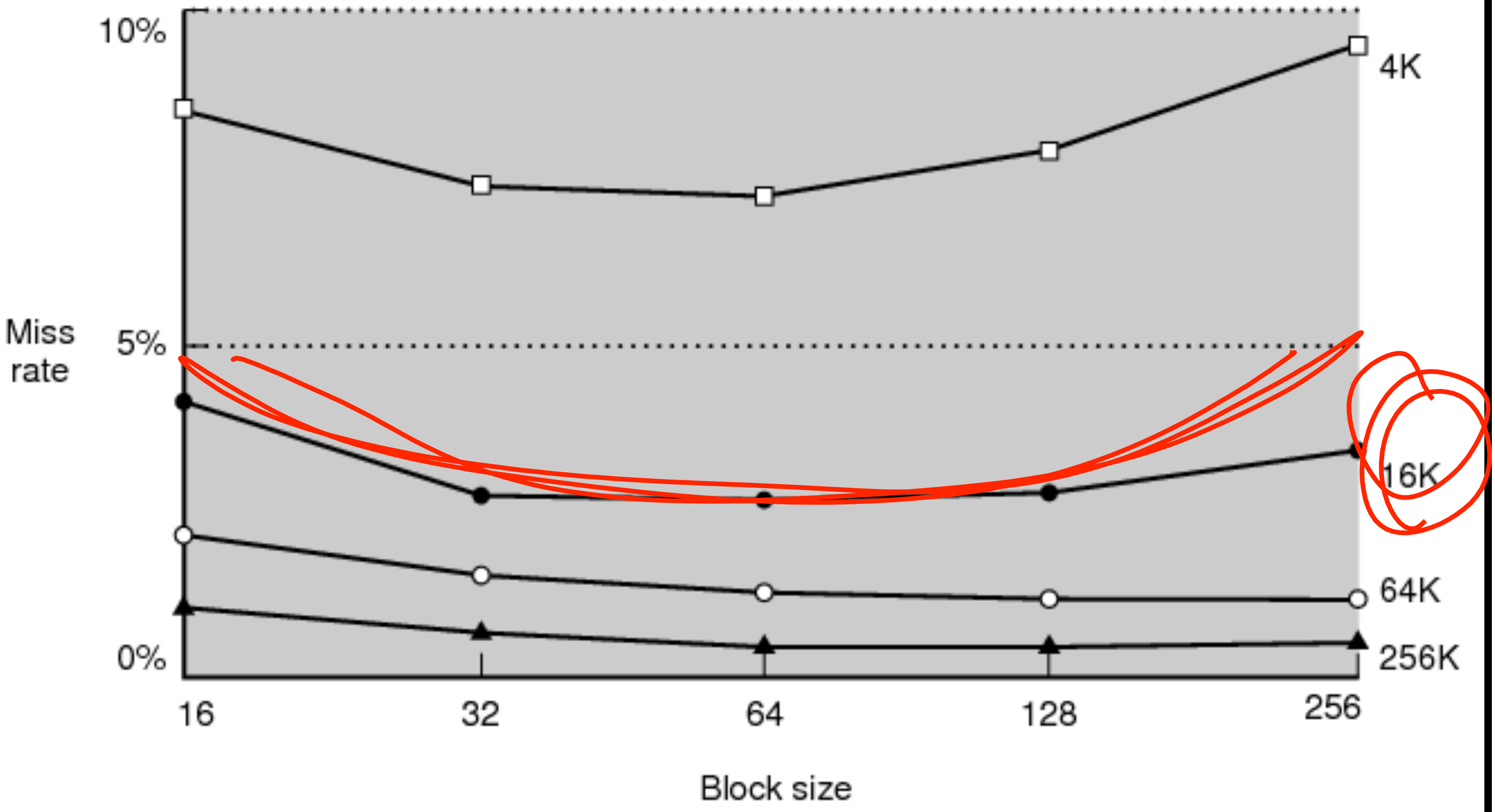
Q: How to decide block size?

A: Try it and see

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

Experimental approach!

Experimental Results



Tradeoffs

For a given total cache size,

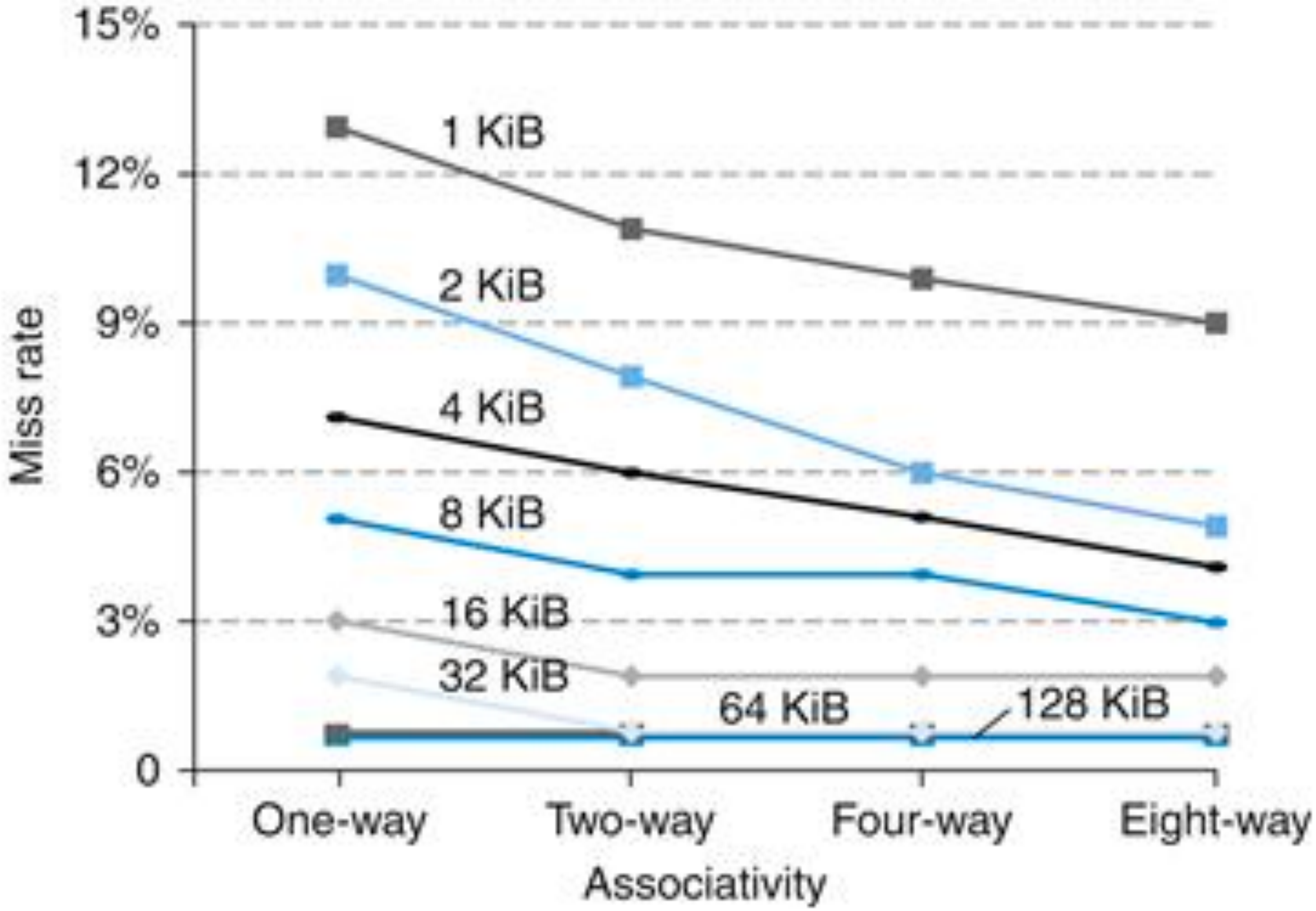
larger block sizes mean….

- fewer lines
- so fewer tags, 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)

Associativity



Other Designs

Multilevel caches

Next Goal

What about writes?

What happens when the CPU writes to a register and calls a store instruction?!

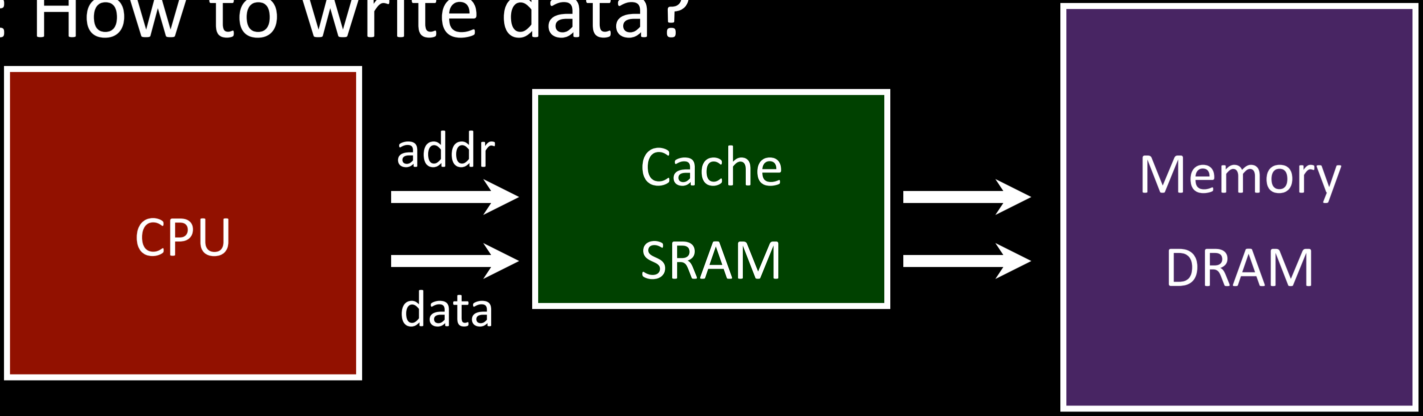
What about Stores?

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 evict the block (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)

Cache 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

writes go to main memory and cache

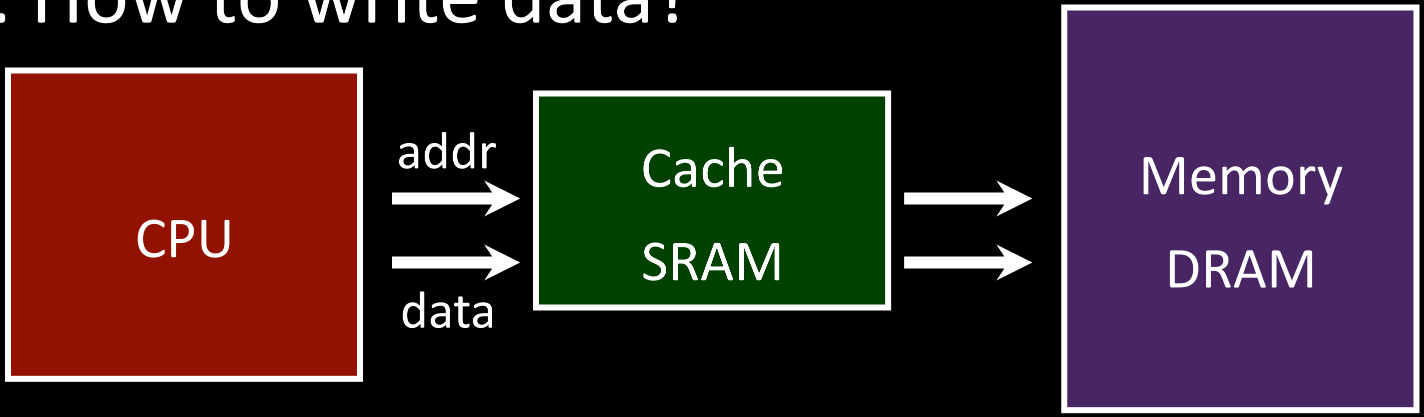
Write-Back

CPU writes only to cache

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

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

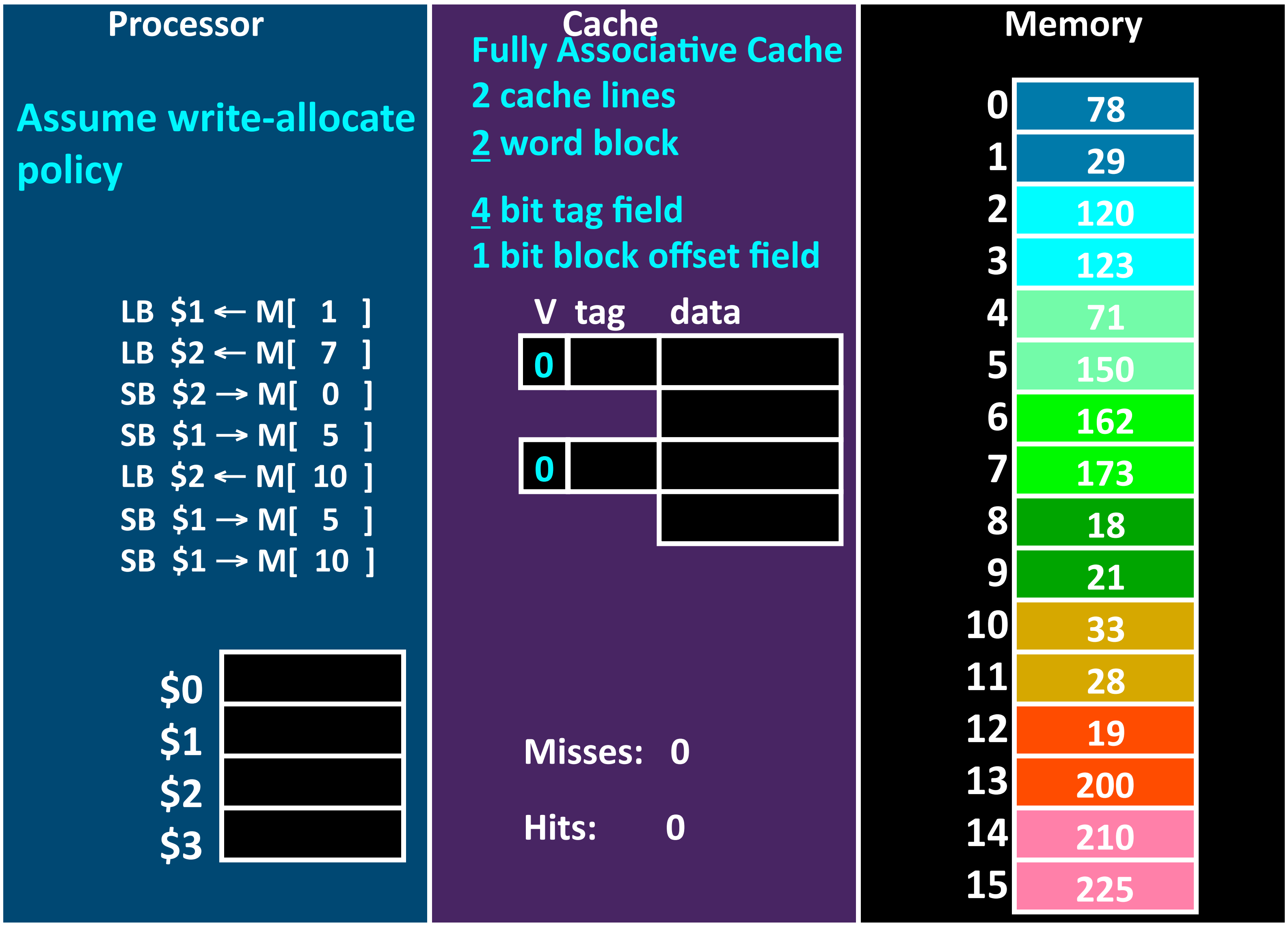
Next Goal

How does a write-through cache work?

Assume write-allocate

Handling Stores (Write-Through)

Using **byte addresses** in this example! Addr Bus = 5 bits



Processor

Assume write-allocate policy

```
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  ]
```

$0	
$1	
$2	
$3	

Cache

Fully Associative Cache

2 cache lines

2 word block

4 bit tag field

1 bit block offset field

V	tag	data
0		
0		

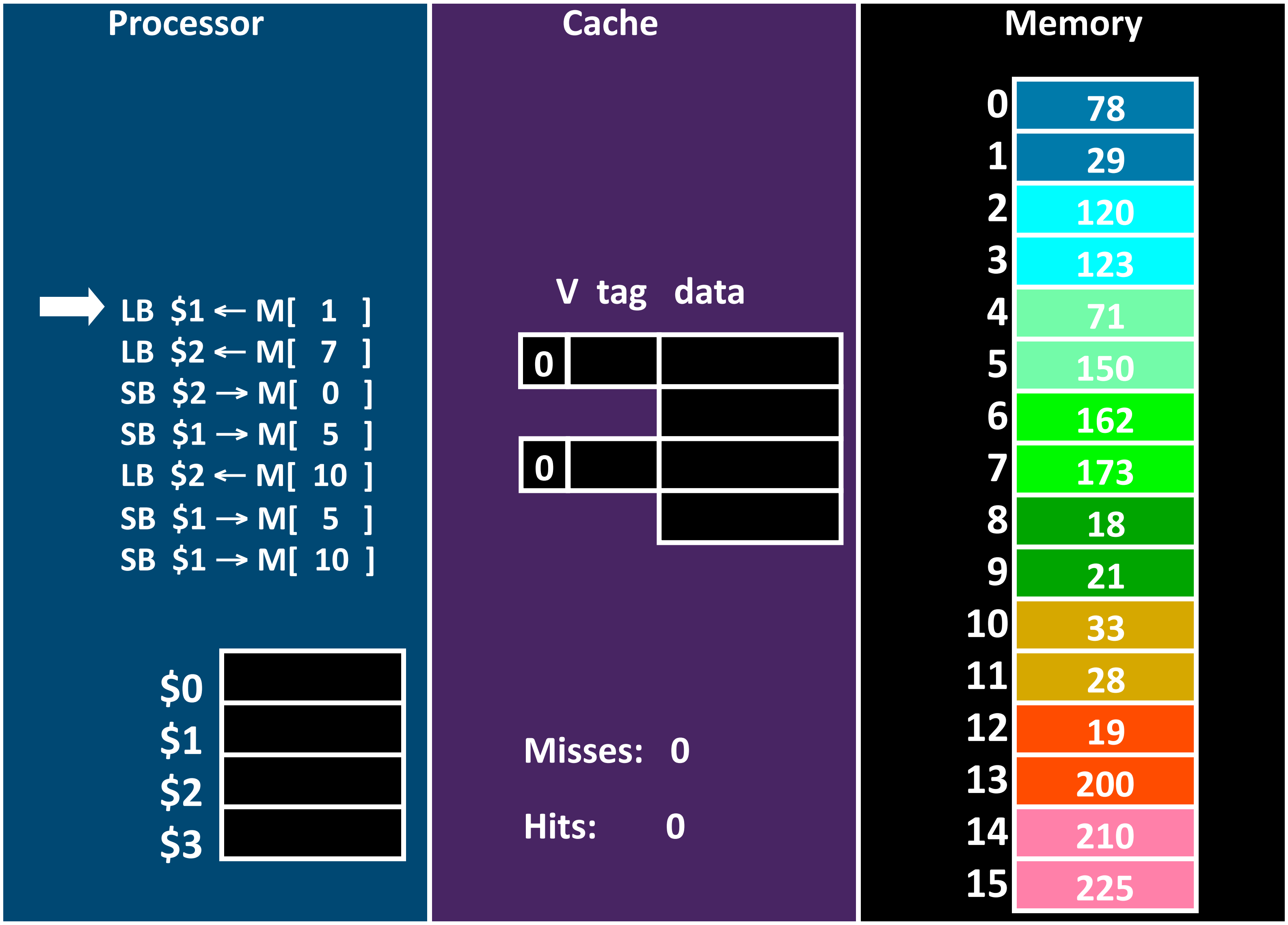
Misses: 0

Hits: 0

Memory

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)



Processor

```
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  ]
```

$0	
$1	
$2	
$3	

Cache

V	tag	data
0		
0		

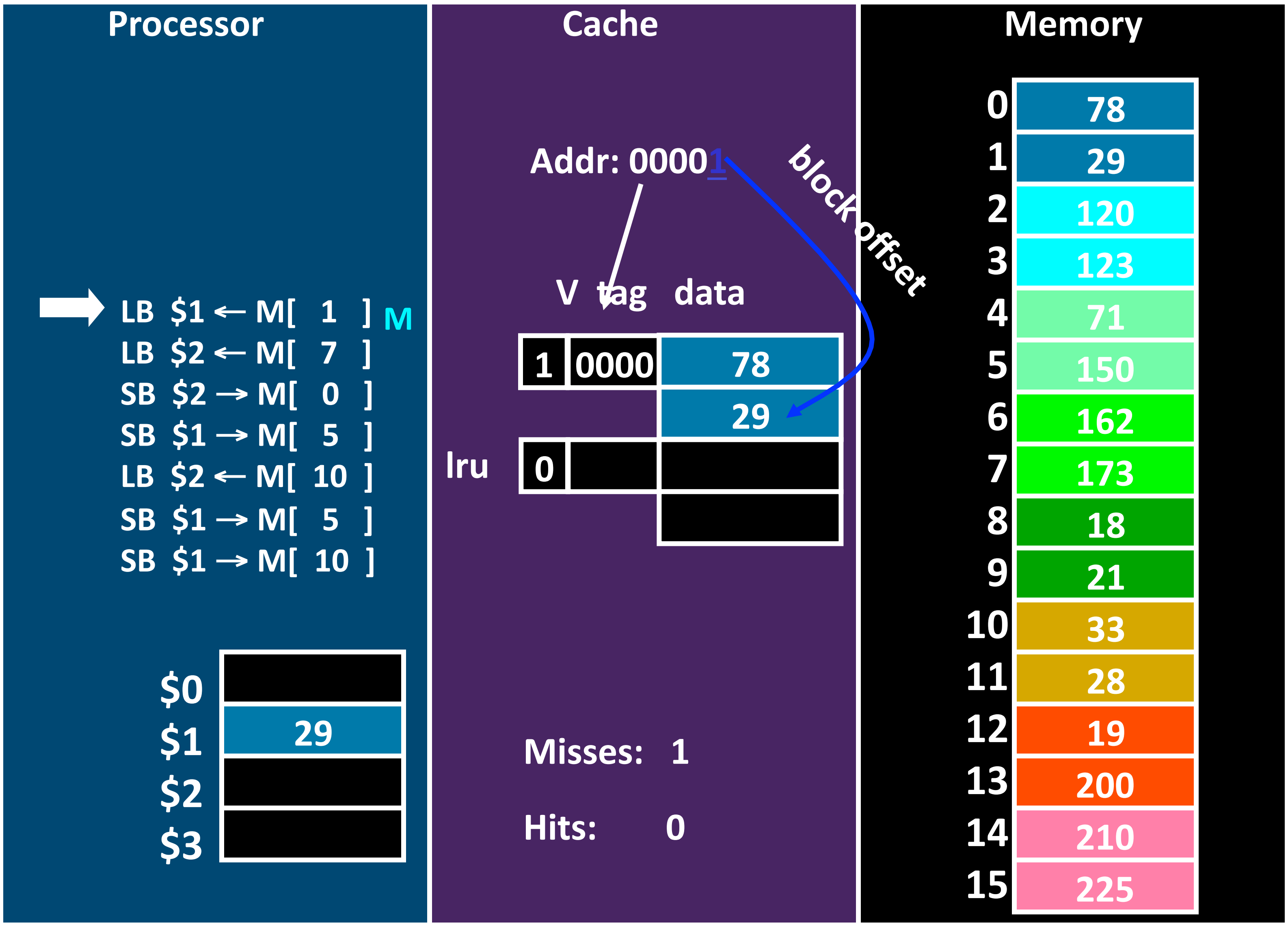
Misses: 0

Hits: 0

Memory

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)



Processor

```
➡ LB  $1 ← M[  1  ] M
   LB  $2 ← M[  7  ]
   SB  $2 → M[  0  ]
   SB  $1 → M[  5  ]
   LB  $2 ← M[ 10  ]
   SB  $1 → M[  5  ]
   SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	
$3	

Cache

Addr: 00001

	V	tag	data
	1	0000	78
			29
lru	0		

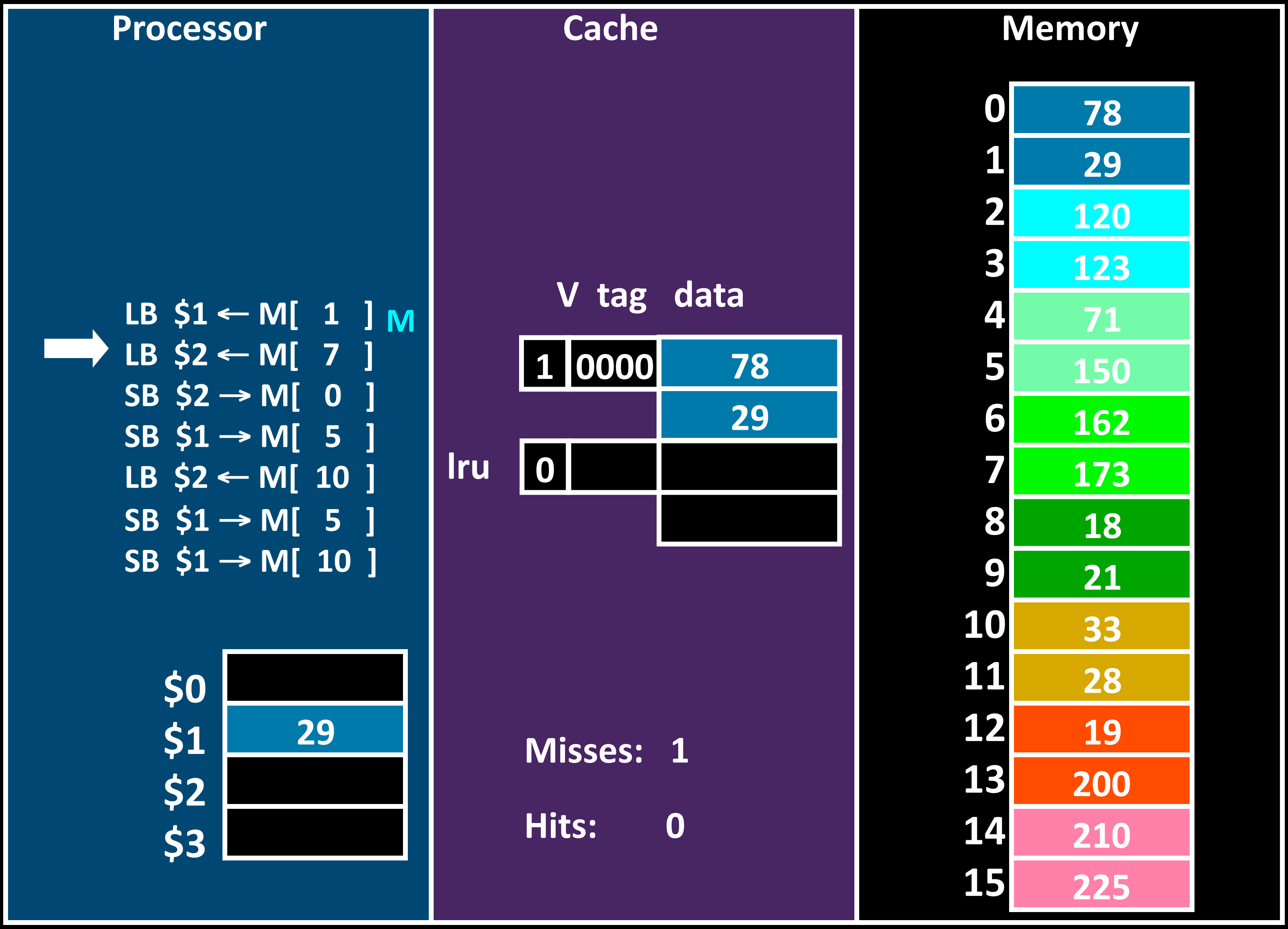
Misses: 1

Hits: 0

Memory

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 2)



Processor

LB $1 ← M[1] M
LB $2 ← M[7]
SB $2 → M[0]
SB $1 → M[5]
LB $2 ← M[10]
SB $1 → M[5]
SB $1 → M[10]

Register	Value
$0	
$1	29
$2	
$3	

Cache

	V	tag	data
	1	0000	78
			29
lru	0		

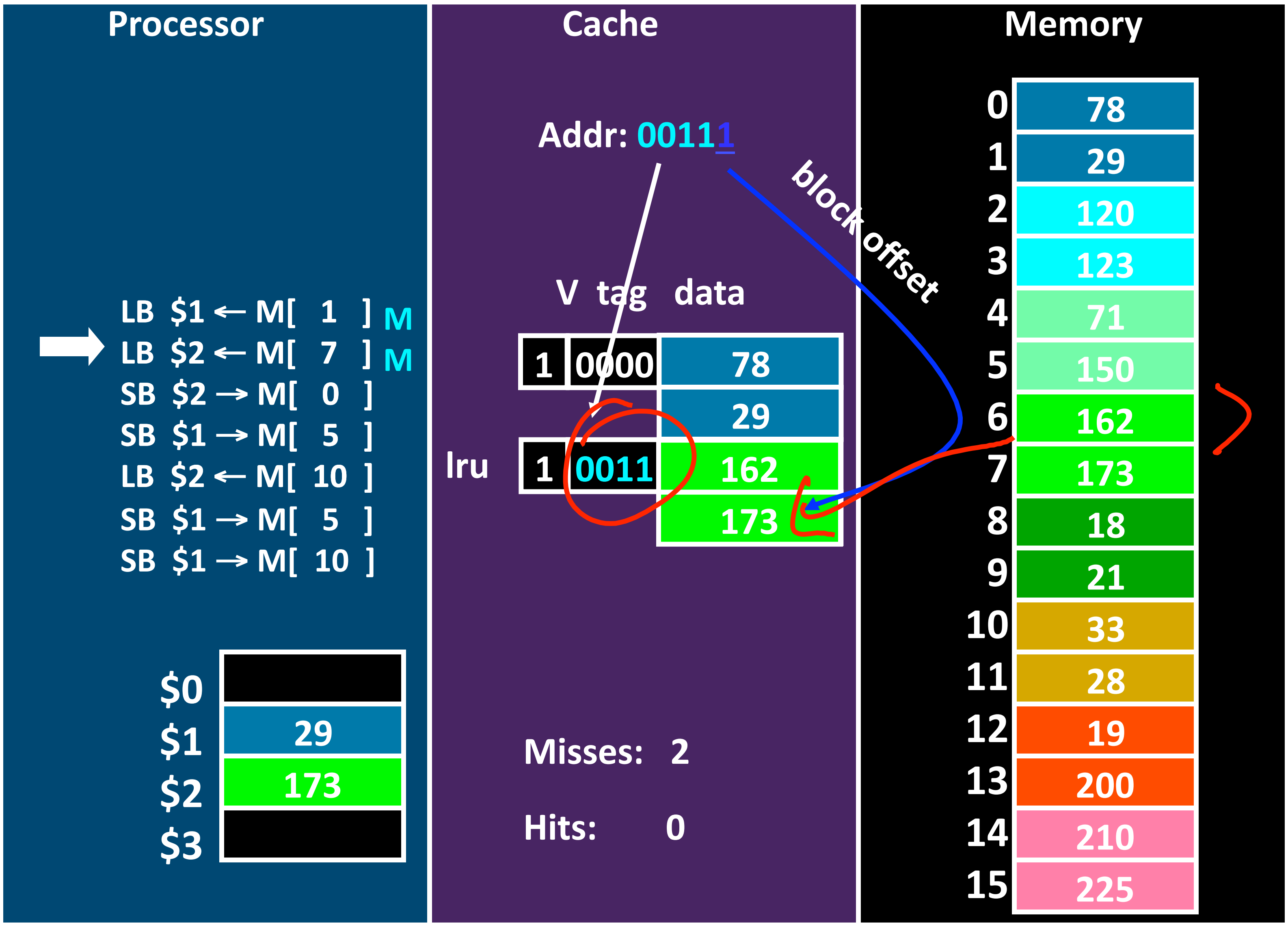
Misses: 1

Hits: 0

Memory

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 2)



```
LB $1 ← M[ 1 ] M
LB $2 ← M[ 7 ] M
SB $2 → M[ 0 ]
SB $1 → M[ 5 ]
LB $2 ← M[ 10 ]
SB $1 → M[ 5 ]
SB $1 → M[ 10 ]
```

$0	
$1	29
$2	173
$3	

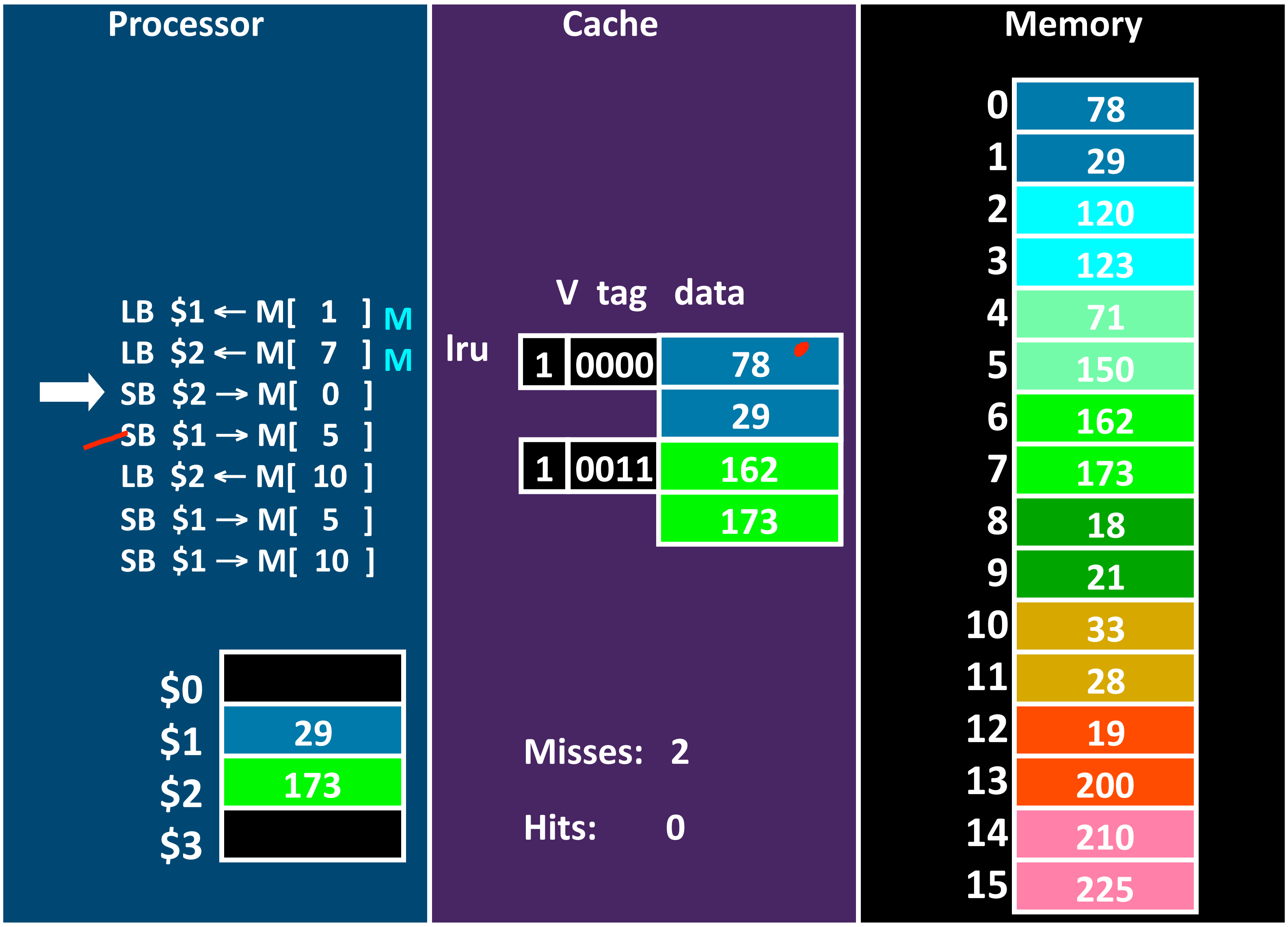
V	tag	data
1	0000	78
		29
1	0011	162
		173

Misses: 2

Hits: 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 3)



Processor

```
LB  $1 ← M[  1  ] M
LB  $2 ← M[  7  ] M
SB  $2 → M[  0  ]
SB  $1 → M[  5  ]
LB  $2 ← M[ 10  ]
SB  $1 → M[  5  ]
SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

Cache

lru

V	tag	data
1	0000	78
		29
1	0011	162
		173

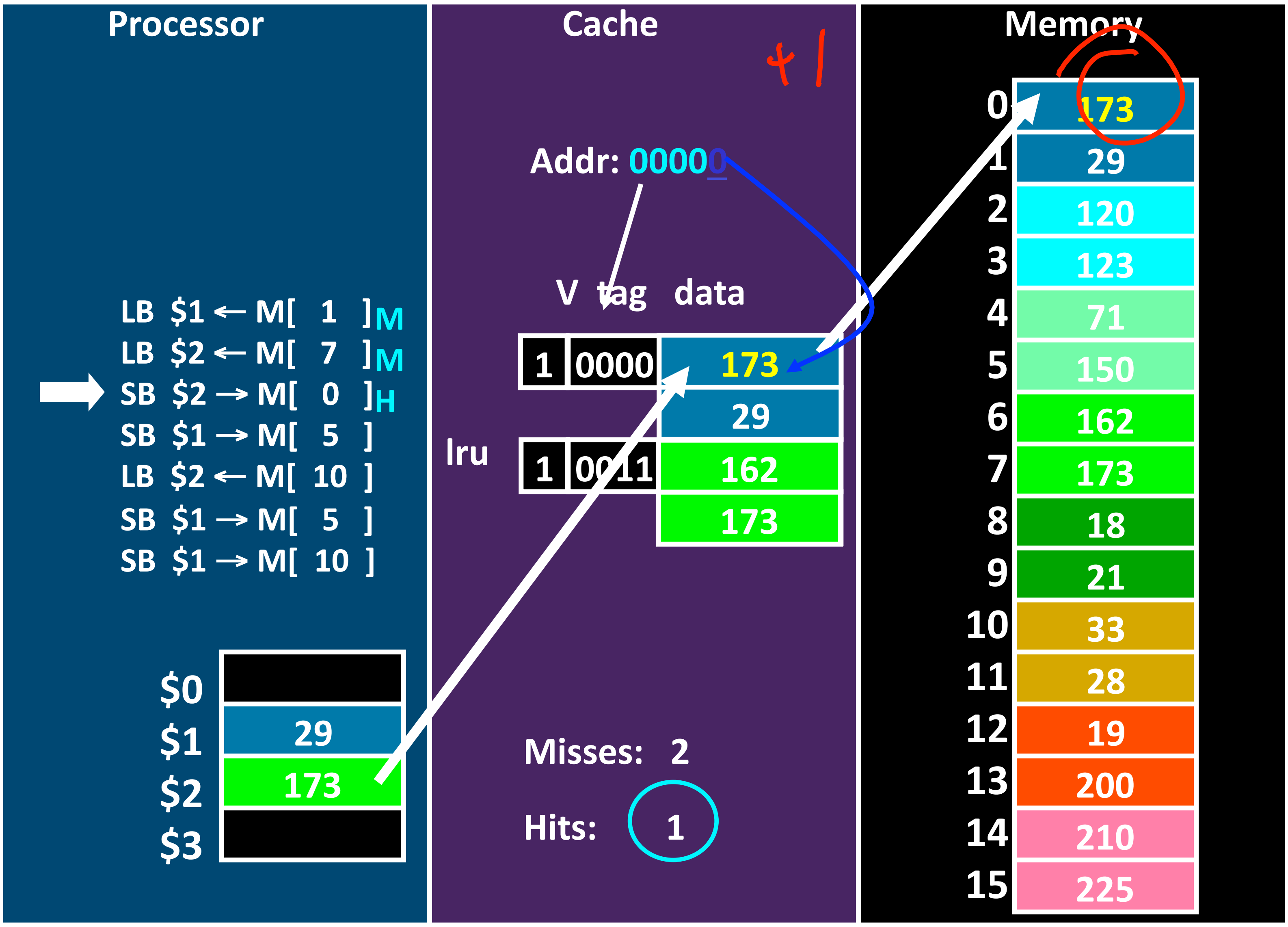
Misses: 2

Hits: 0

Memory

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 3)



Processor

```
LB  $1 ← M[  1  ] M
LB  $2 ← M[  7  ] M
SB  $2 → M[  0  ] H
SB  $1 → M[  5  ]
LB  $2 ← M[ 10  ]
SB  $1 → M[  5  ]
SB  $1 → M[ 10  ]
```

Register	Value
$0	
$1	29
$2	173
$3	

Cache

+1

Addr: 00000

	V	tag	data
	1	0000	173
			29
lru	1	0011	162
			173

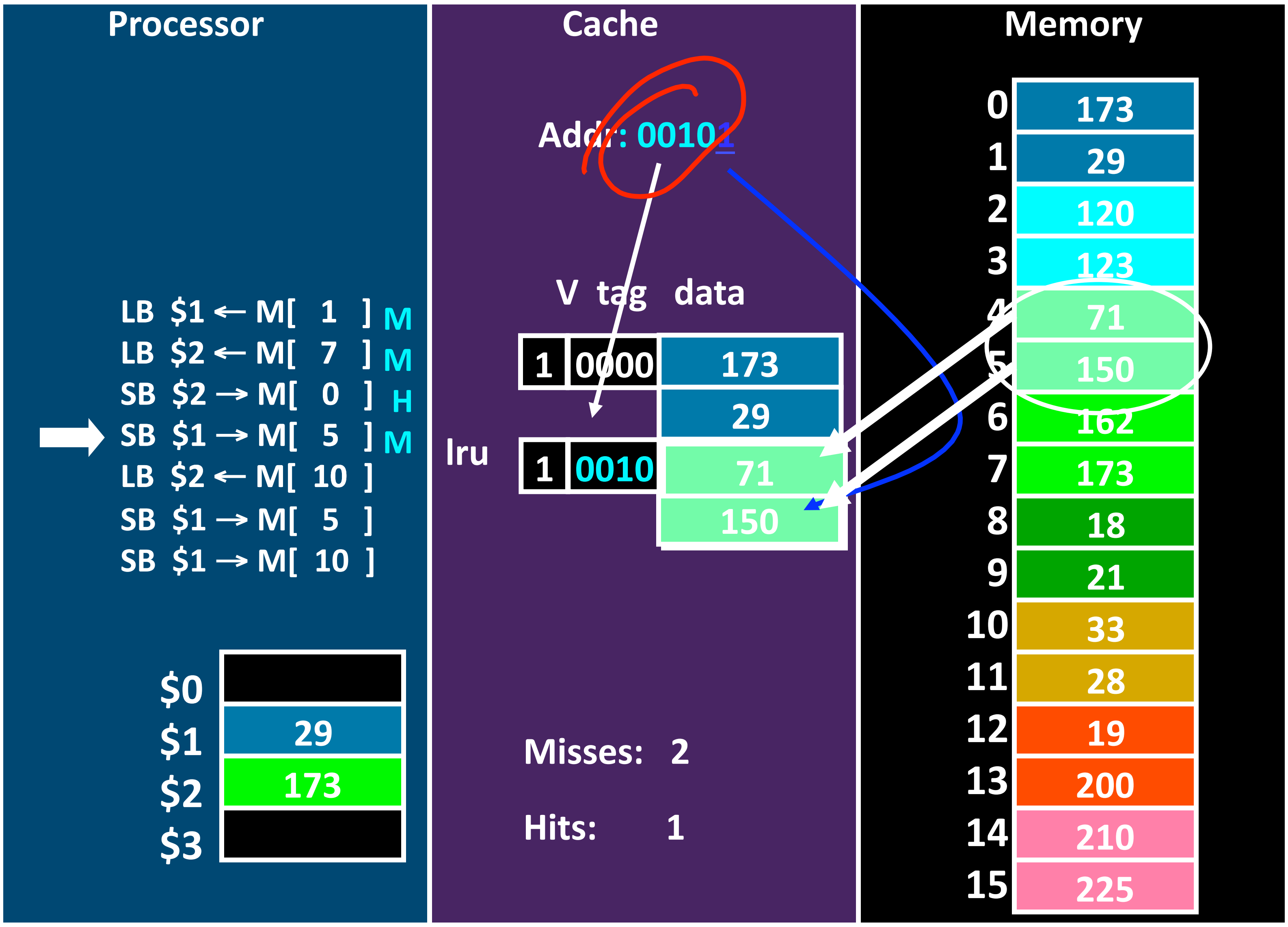
Misses: 2

Hits: 1

Memory

Address	Value
0	173
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 4)



Processor

LB $1 ← M[1] M
LB $2 ← M[7] M
SB $2 → M[0] H
SB $1 → M[5] M
LB $2 ← M[10]
SB $1 → M[5]
SB $1 → M[10]

$0	
$1	29
$2	173
$3	

Cache

Addr: 00101

	V	tag	data
	1	0000	173
			29
lru	1	0010	71
			150

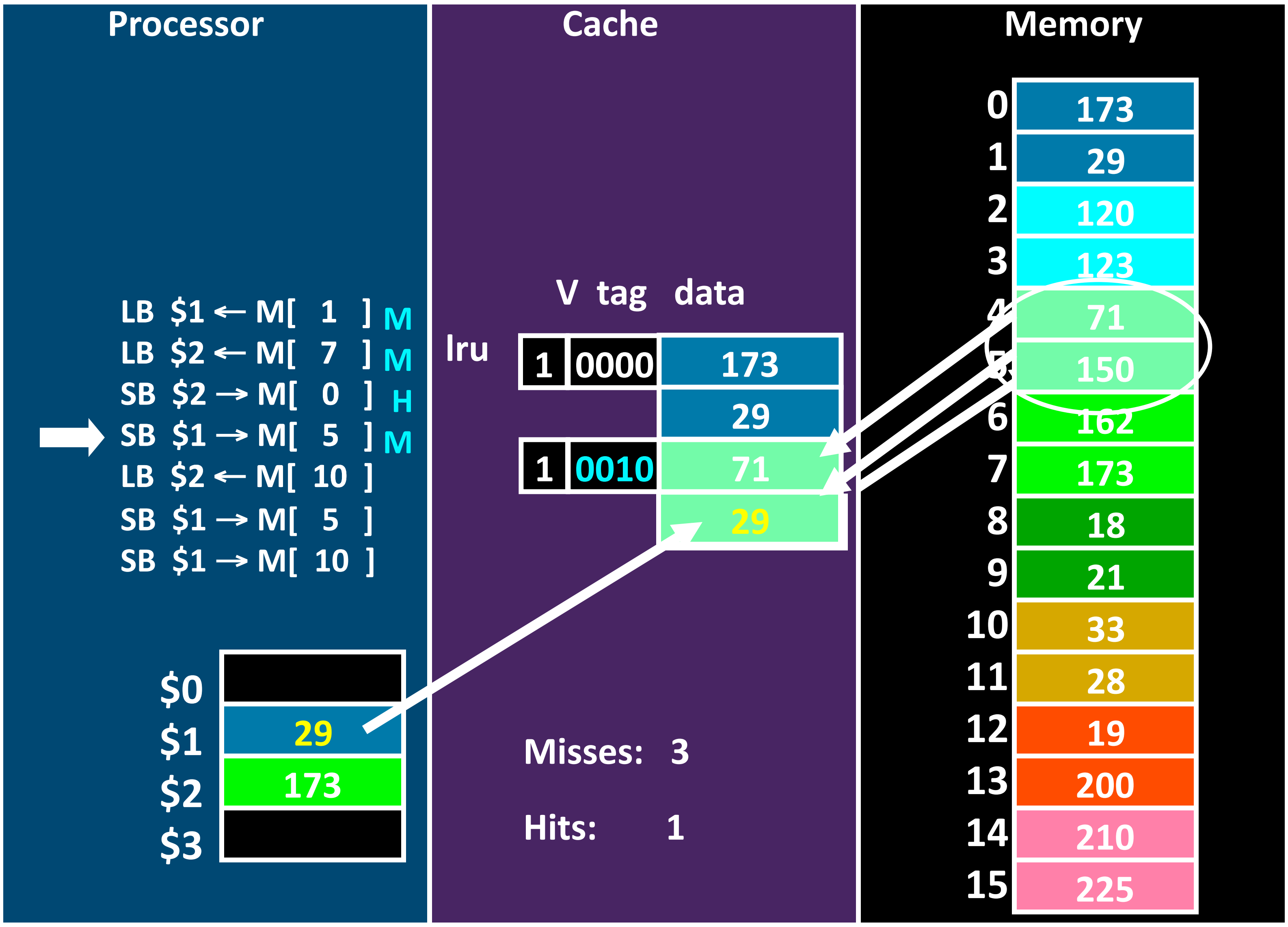
Misses: 2

Hits: 1

Memory

0	173
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 4)



```
   LB  $1 ← M[  1  ]  M
   LB  $2 ← M[  7  ]  M
   SB  $2 → M[  0  ]  H
⇨  SB  $1 → M[  5  ]  M
   LB  $2 ← M[ 10  ]
   SB  $1 → M[  5  ]
   SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

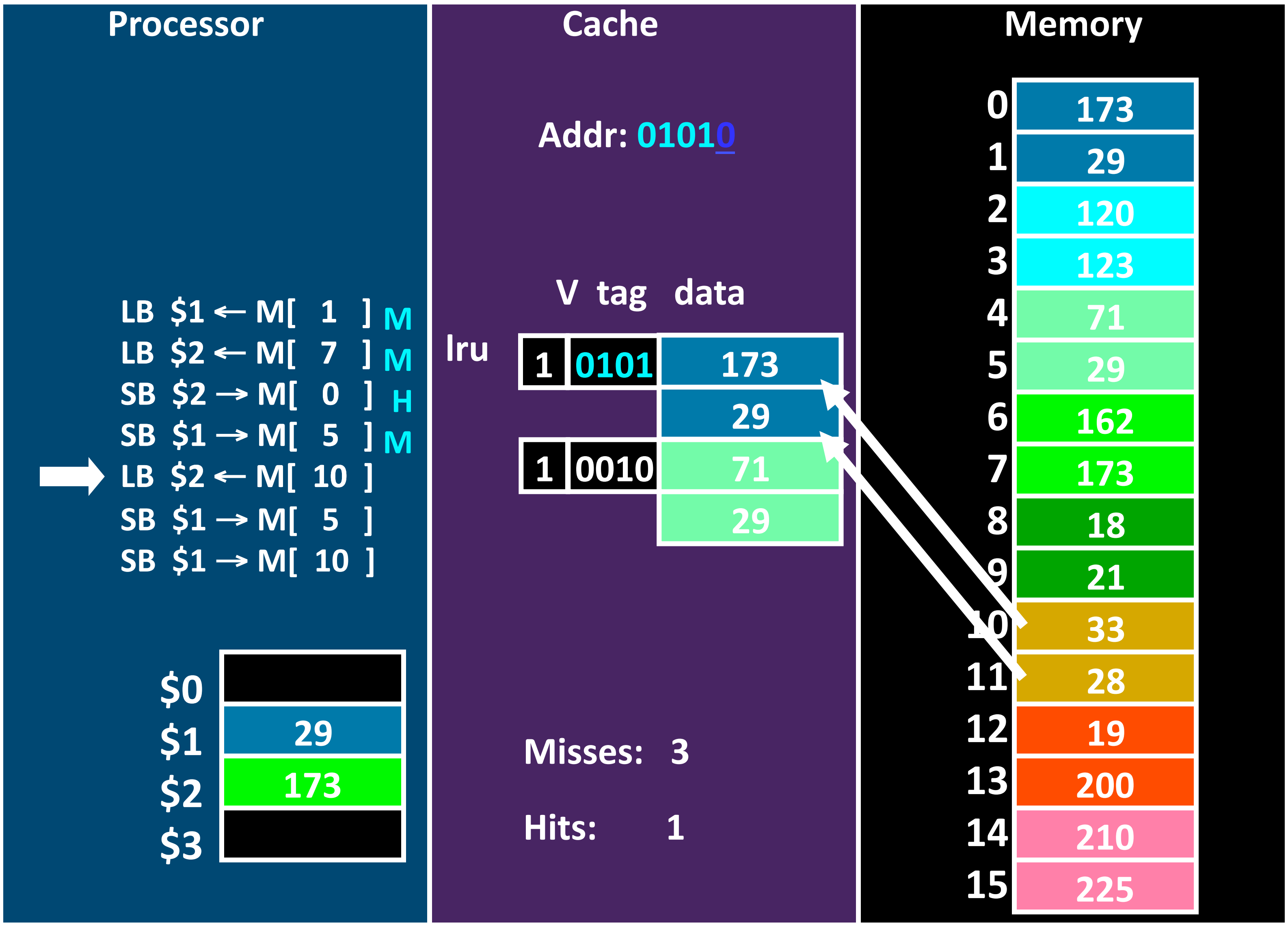
V	tag	data
1	0000	173
		29
1	0010	71
		29

Misses: 3

Hits: 1

0	173
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 5)



Processor

```
  LB  $1 ← M[  1  ] M
  LB  $2 ← M[  7  ] M
  SB  $2 → M[  0  ] H
  SB  $1 → M[  5  ] M
➡ LB  $2 ← M[ 10  ]
  SB  $1 → M[  5  ]
  SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

Cache

Addr: 01010

	V	tag	data
lru	1	0101	173
			29
	1	0010	71
			29

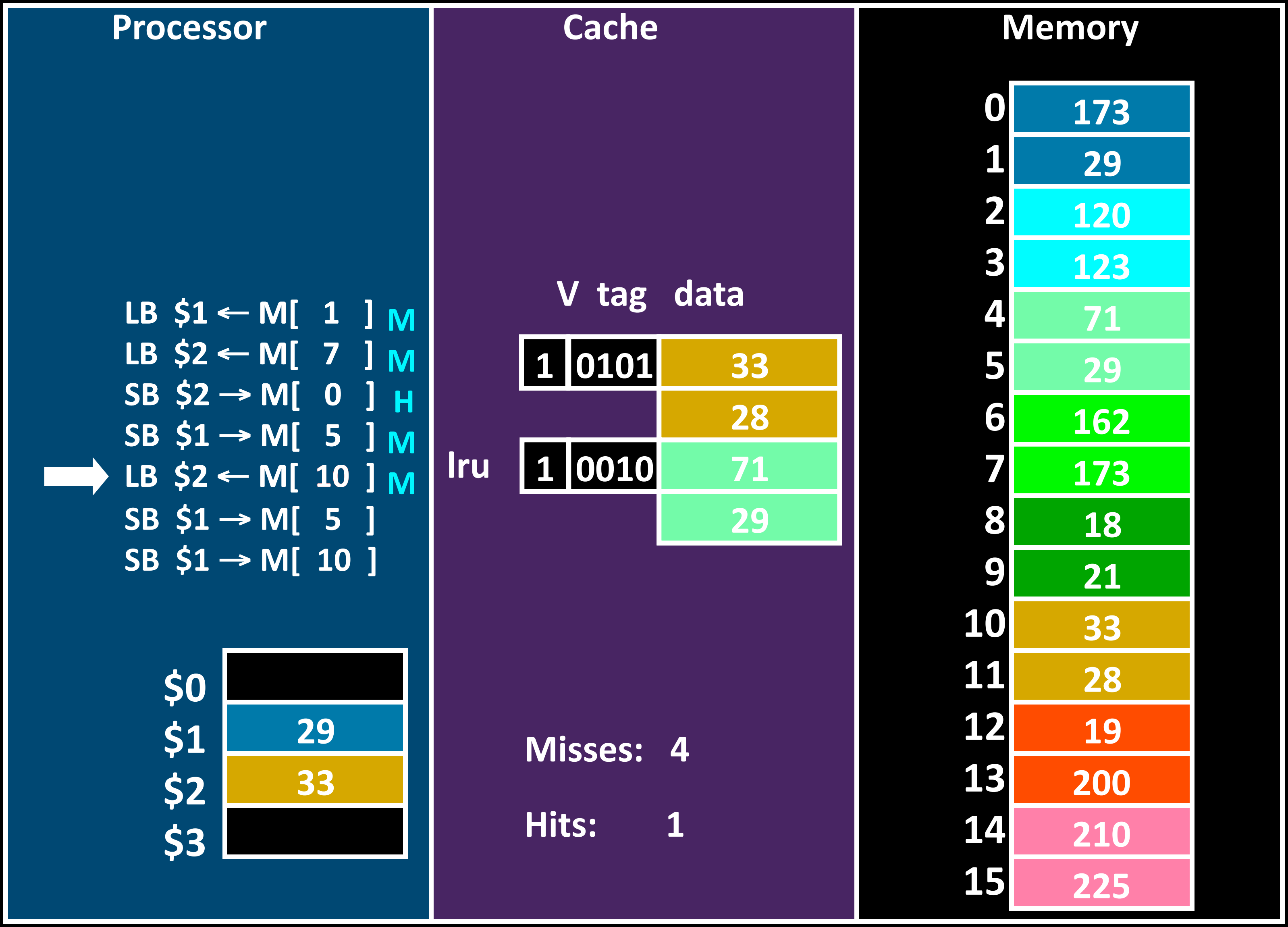
Misses: 3

Hits: 1

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	33
11	28
12	19
13	200
14	210
15	225

Write-Through (REF 5)



Processor

```
LB $1 ← M[ 1 ]  M
LB $2 ← M[ 7 ]  M
SB $2 → M[ 0 ]  H
SB $1 → M[ 5 ]  M
LB $2 ← M[ 10 ] M
SB $1 → M[ 5 ]
SB $1 → M[ 10 ]
```

$0	
$1	29
$2	33
$3	

Cache

	V	tag	data
	1	0101	33
			28
lru	1	0010	71
			29

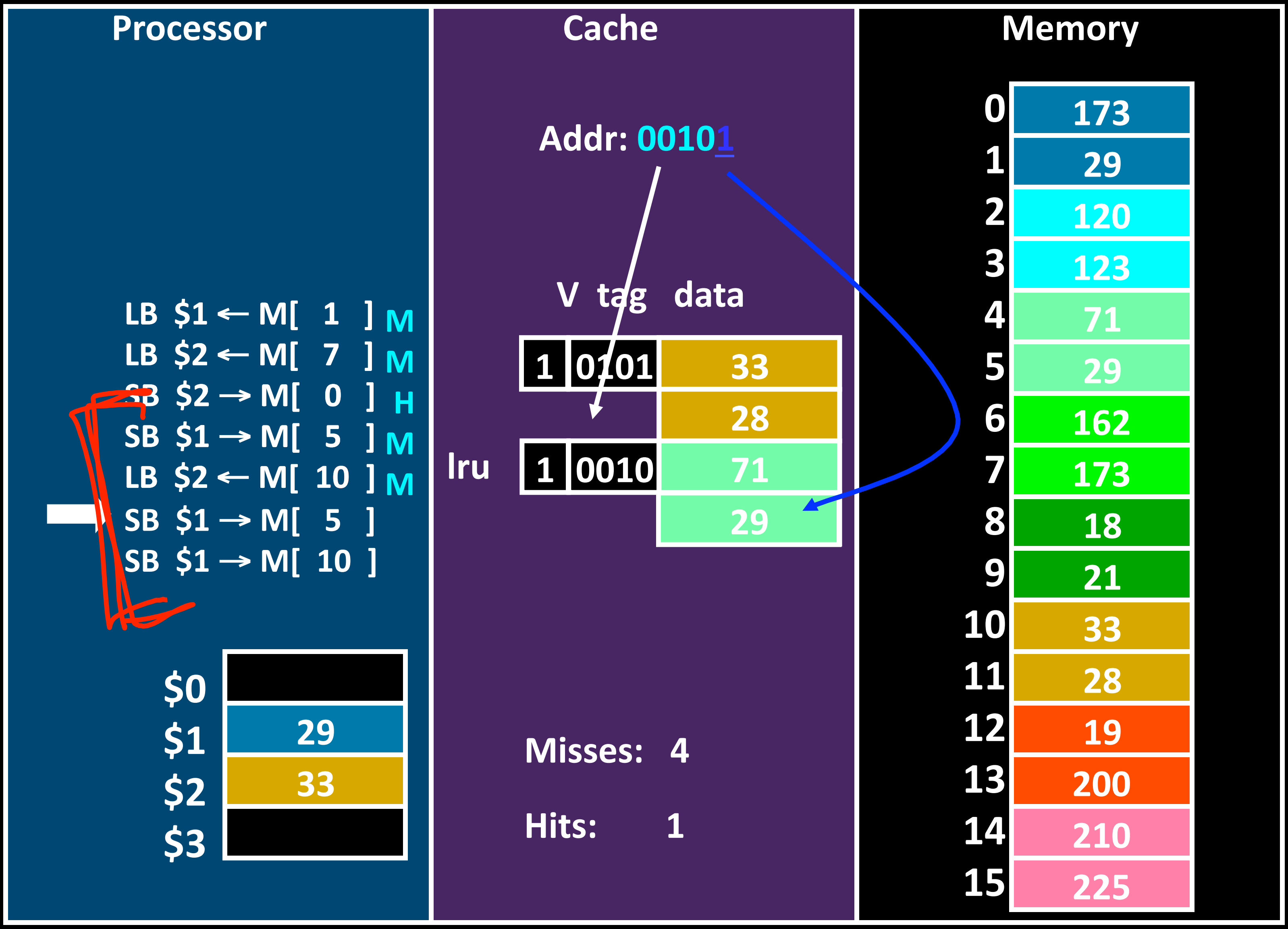
Misses: 4

Hits: 1

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	33
11	28
12	19
13	200
14	210
15	225

Write-Through (REF 6)



Processor

```
LB  $1 ← M[  1  ] M
LB  $2 ← M[  7  ] M
SB  $2 → M[  0  ] H
SB  $1 → M[  5  ] M
LB  $2 ← M[ 10  ] M
SB  $1 → M[  5  ]
SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	33
$3	

Cache

Addr: 00101

	V	tag	data
	1	0101	33
			28
lru	1	0010	71
			29

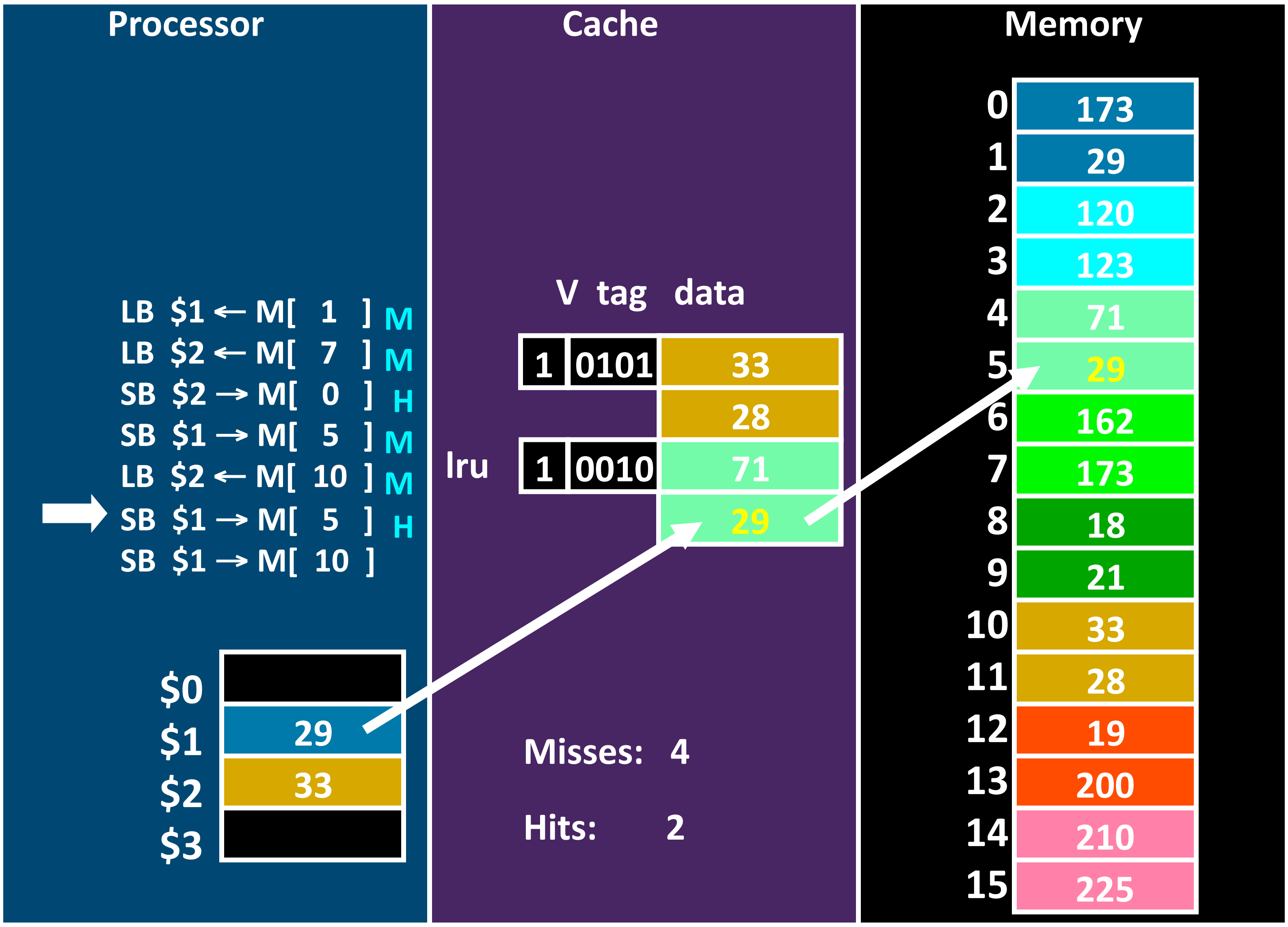
Misses: 4

Hits: 1

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	33
11	28
12	19
13	200
14	210
15	225

Write-Through (REF 6)



Processor

```
LB  $1 ← M[  1  ]  M
LB  $2 ← M[  7  ]  M
SB  $2 → M[  0  ]  H
SB  $1 → M[  5  ]  M
LB  $2 ← M[ 10  ]  M
SB  $1 → M[  5  ]  H
SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	33
$3	

Cache

	V	tag	data
	1	0101	33
			28
lru	1	0010	71
			29

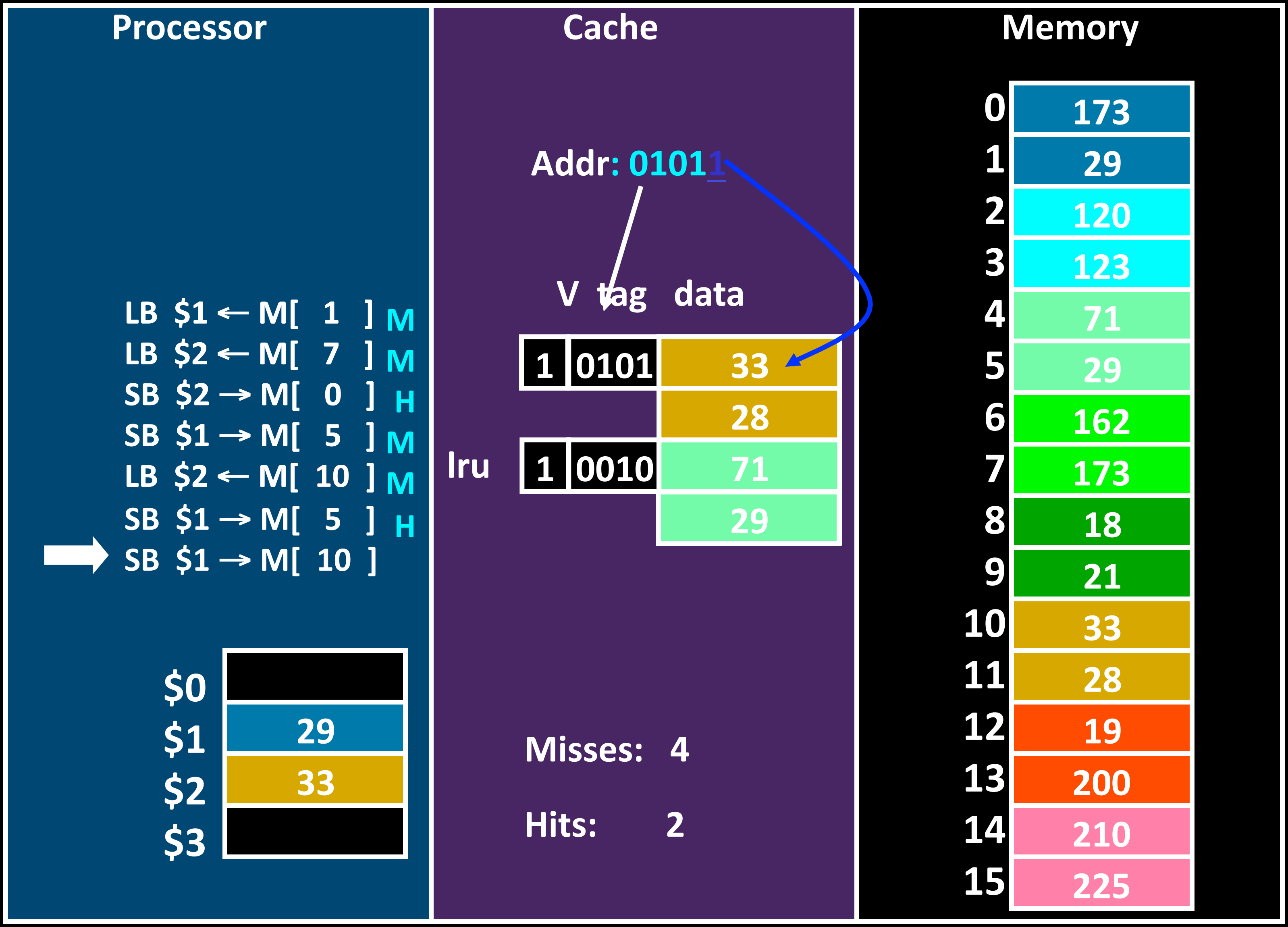
Misses: 4

Hits: 2

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	33
11	28
12	19
13	200
14	210
15	225

Write-Through (REF 7)



Processor

LB $1 ← M[1] M
LB $2 ← M[7] M
SB $2 → M[0] H
SB $1 → M[5] M
LB $2 ← M[10] M
SB $1 → M[5] H
SB $1 → M[10]

$0	
$1	29
$2	33
$3	

Cache

Addr: 01011

	V	tag	data
	1	0101	33
			28
lru	1	0010	71
			29

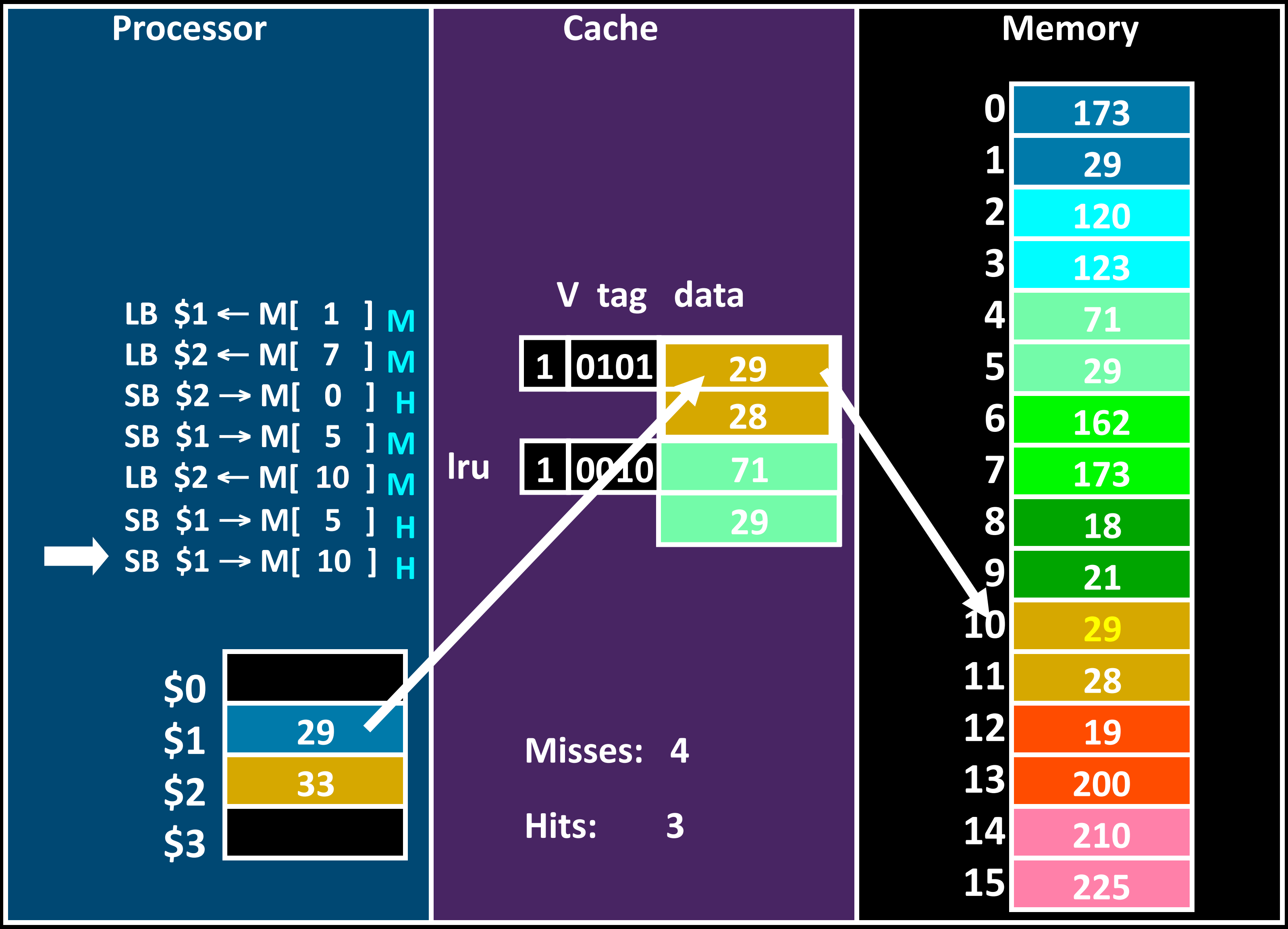
Misses: 4

Hits: 2

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	33
11	28
12	19
13	200
14	210
15	225

Write-Through (REF 7)



Processor

```
  LB  $1 ← M[  1 ]  M
  LB  $2 ← M[  7 ]  M
  SB  $2 → M[  0 ]  H
  SB  $1 → M[  5 ]  M
  LB  $2 ← M[ 10 ]  M
  SB  $1 → M[  5 ]  H
➡ SB  $1 → M[ 10 ]  H
```

$0	
$1	29
$2	33
$3	

Cache

	V	tag	data
	1	0101	29
			28
lru	1	0010	71
			29

Misses: 4

Hits: 3

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	29
11	28
12	19
13	200
14	210
15	225

How Many Memory References?

Write-through performance

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

- 4 misses → 8 mem reads

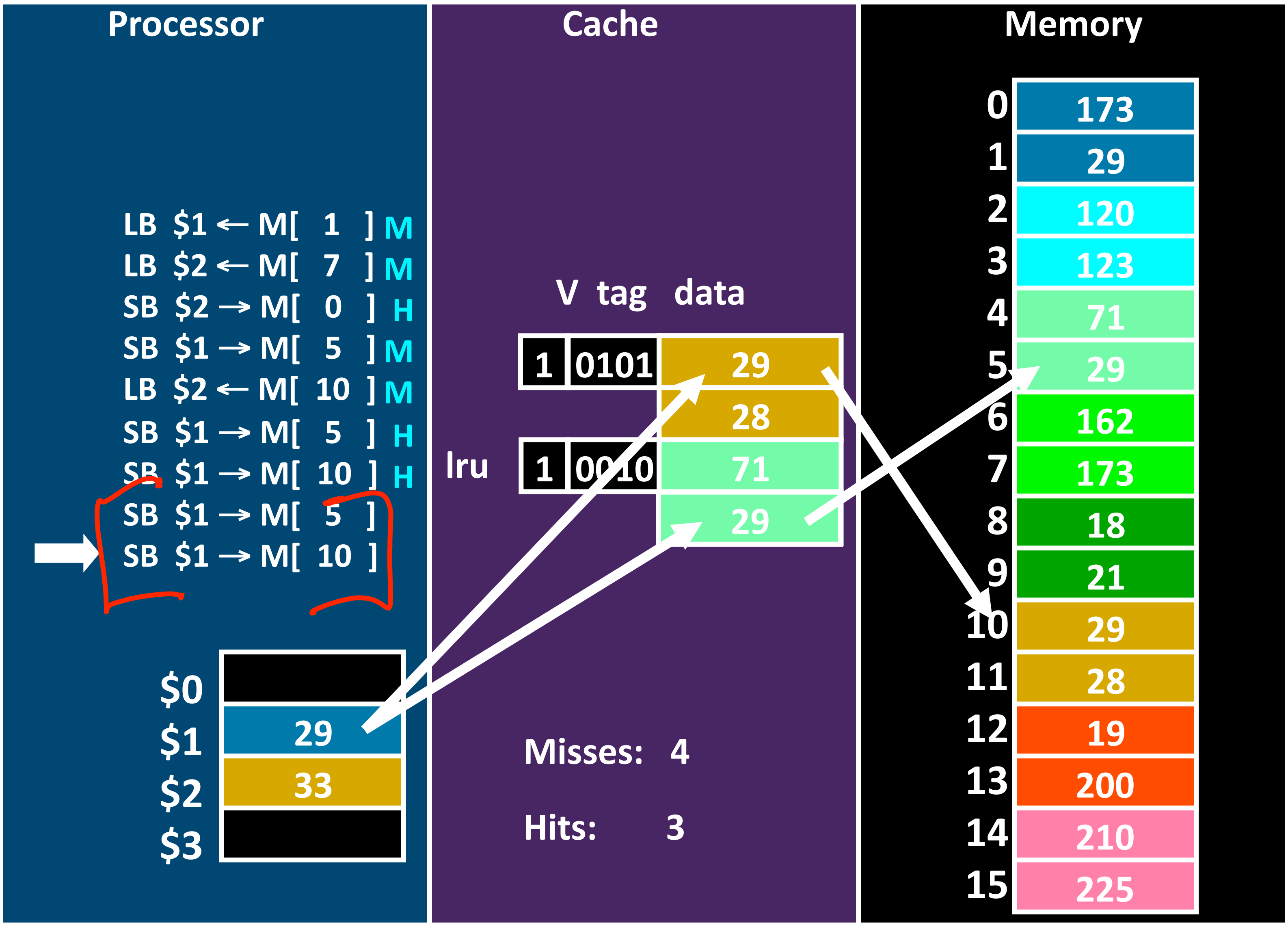
Each store writes an item to mem

- 4 mem writes

Evictions don’t need to write to mem

- no need for dirty bit

Write-Through (REF 8,9)



Processor

```
LB  $1 ← M[  1  ] M
LB  $2 ← M[  7  ] M
SB  $2 → M[  0  ] H
SB  $1 → M[  5  ] M
LB  $2 ← M[ 10  ] M
SB  $1 → M[  5  ] H
SB  $1 → M[ 10  ] H
SB  $1 → M[  5  ]
SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	33
$3	

Cache

	V	tag	data
	1	0101	29
			28
lru	1	0010	71
			29

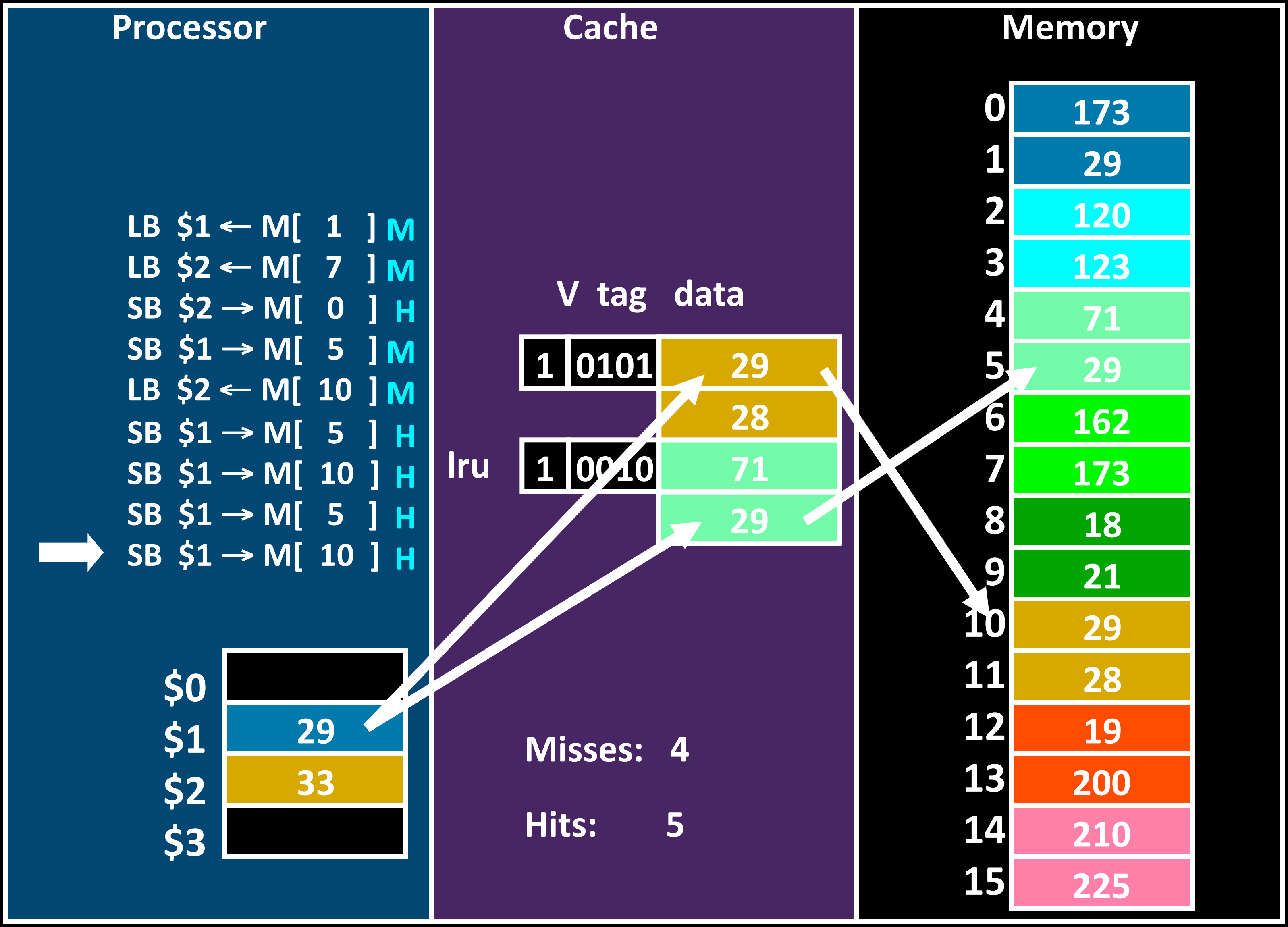
Misses: 4

Hits: 3

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	29
11	28
12	19
13	200
14	210
15	225

Write-Through (REF 8,9)



Processor

LB $1 ← M[1] M
LB $2 ← M[7] M
SB $2 → M[0] H
SB $1 → M[5] M
LB $2 ← M[10] M
SB $1 → M[5] H
SB $1 → M[10] H
SB $1 → M[5] H
➡ SB $1 → M[10] H

$0	
$1	29
$2	33
$3	

Cache

	V	tag	data
	1	0101	29
			28
lru	1	0010	71
			29

Misses: 4

Hits: 5

Memory

0	173
1	29
2	120
3	123
4	71
5	29
6	162
7	173
8	18
9	21
10	29
11	28
12	19
13	200
14	210
15	225

Summary: Write Through

Write-through policy with write allocate

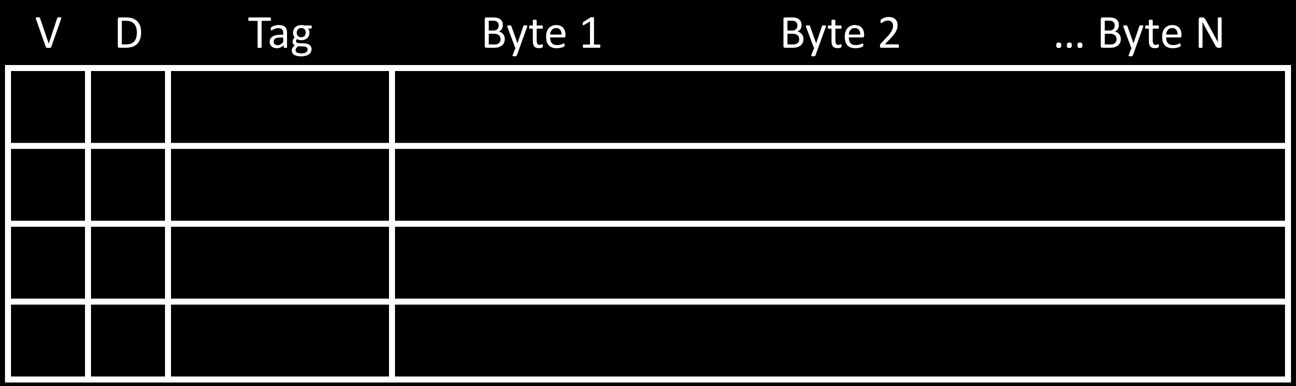
- Cache miss: read entire block from memory
- Write: write only updated item to memory
- Eviction: no need to write to memory

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 = 1 means the line has valid data

D = 1 means the bytes are newer than main memory

When allocating line: 2

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

When writing line: 3

- Set D = 1

When evicting line: 1

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

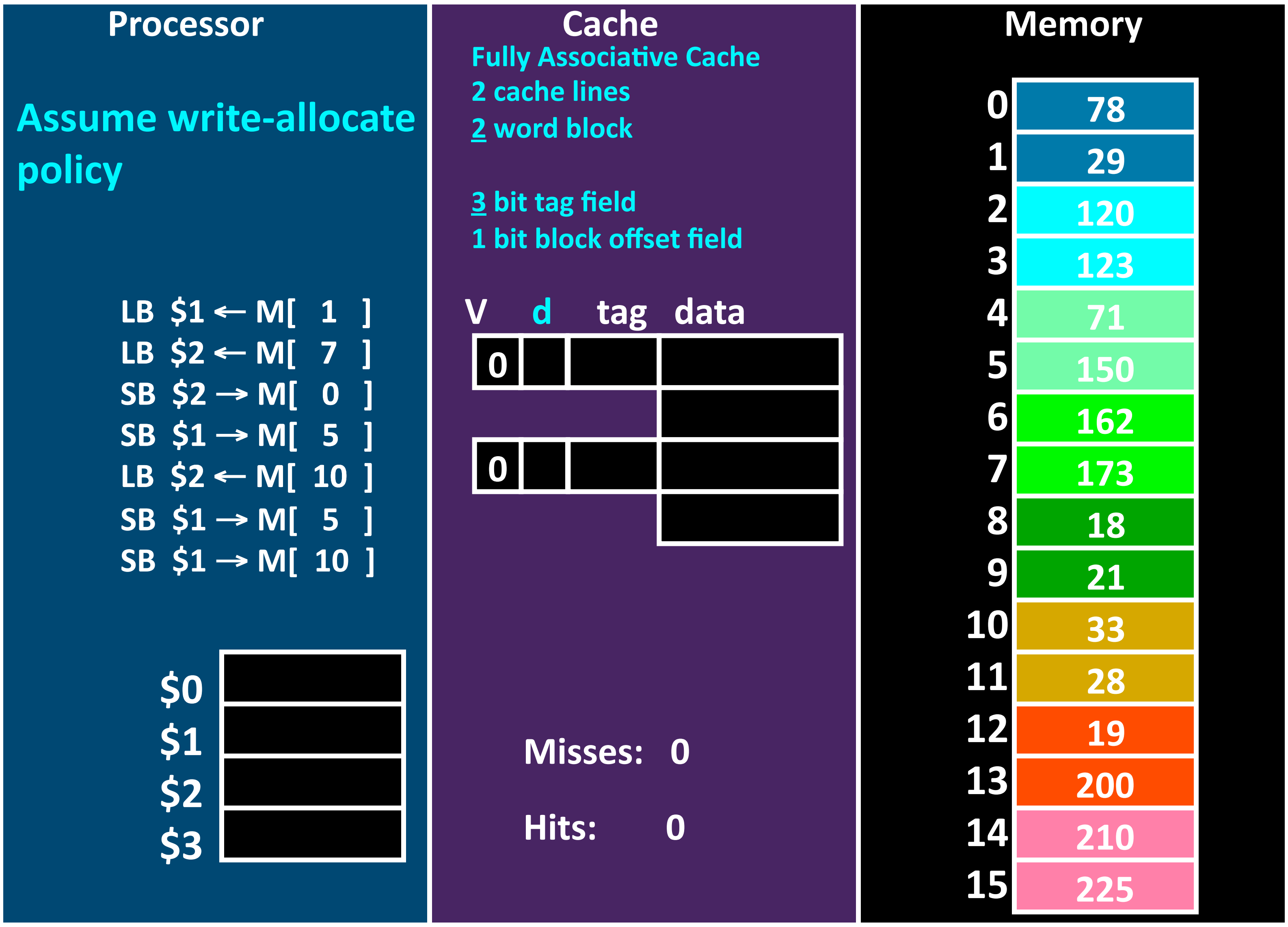
Write-back Example

Example: How does a write-back cache work?

Assume write-allocate

Handling Stores (Write-Back)

Using **byte addresses** in this example! Addr Bus = 5 bits



Processor

Assume write-allocate policy

```
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  ]
```

$0	
$1	
$2	
$3	

Cache

Fully Associative Cache
2 cache lines
2 word block

3 bit tag field
1 bit block offset field

V	d	tag	data
0			
0			

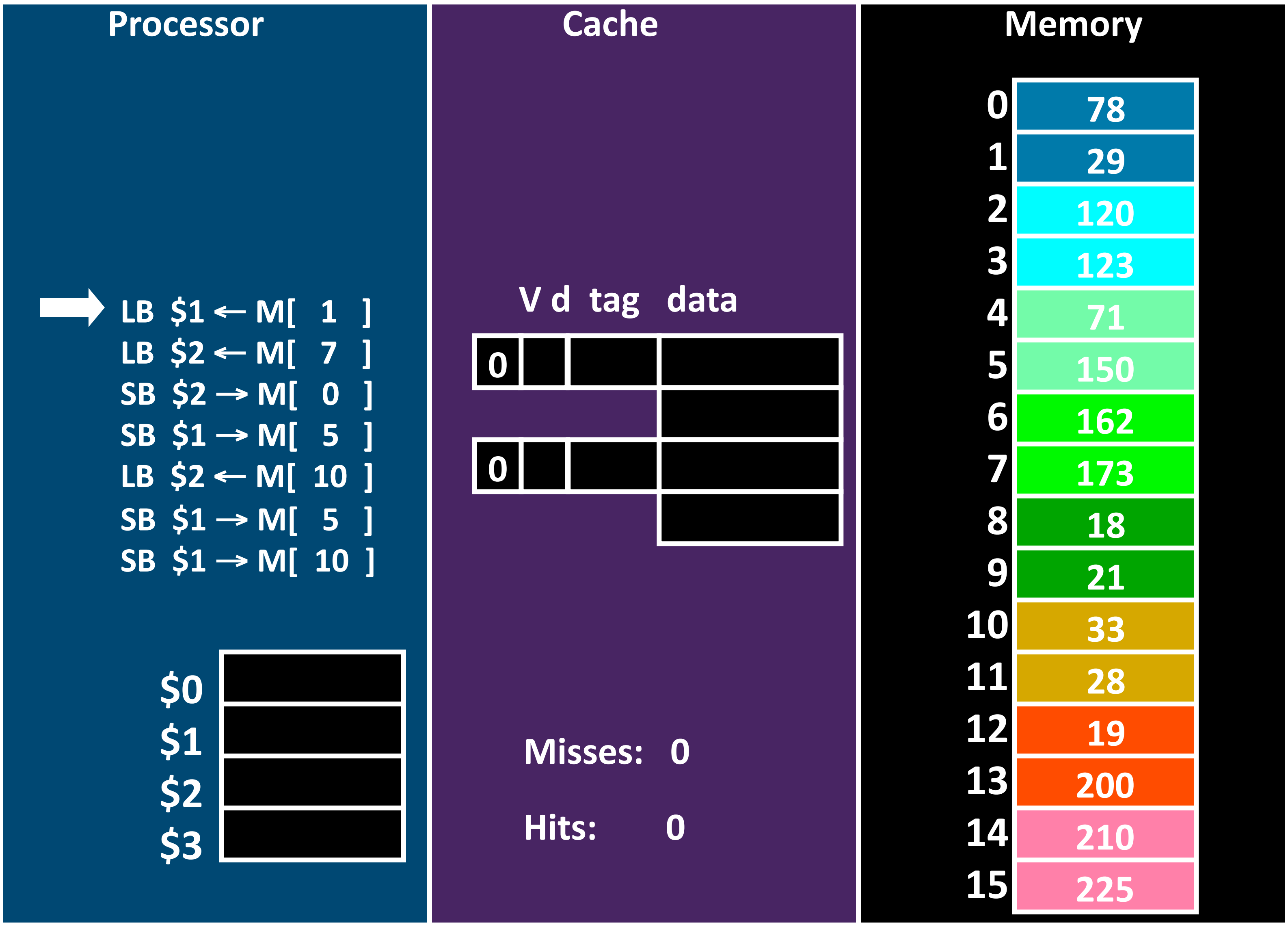
Misses: 0

Hits: 0

Memory

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)



Processor

```
➡ 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  ]
```

$0	
$1	
$2	
$3	

Cache

V	d	tag	data
0			
0			

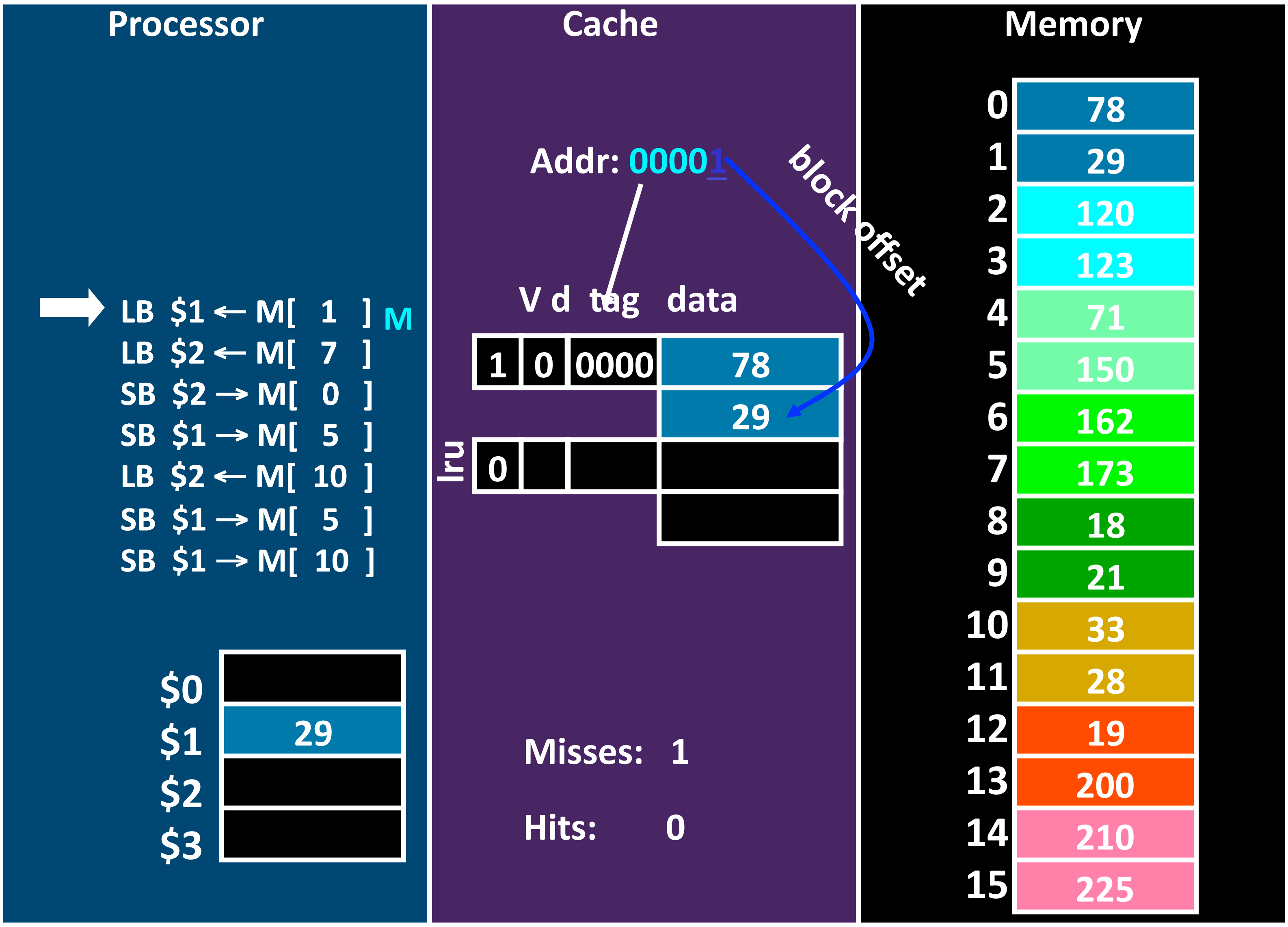
Misses: 0

Hits: 0

Memory

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)



Processor

Cache

Memory

LB $1 ← M[1] M
LB $2 ← M[7]
SB $2 → M[0]
SB $1 → M[5]
LB $2 ← M[10]
SB $1 → M[5]
SB $1 → M[10]

$0	
$1	29
$2	
$3	

Addr: 00001

block offset

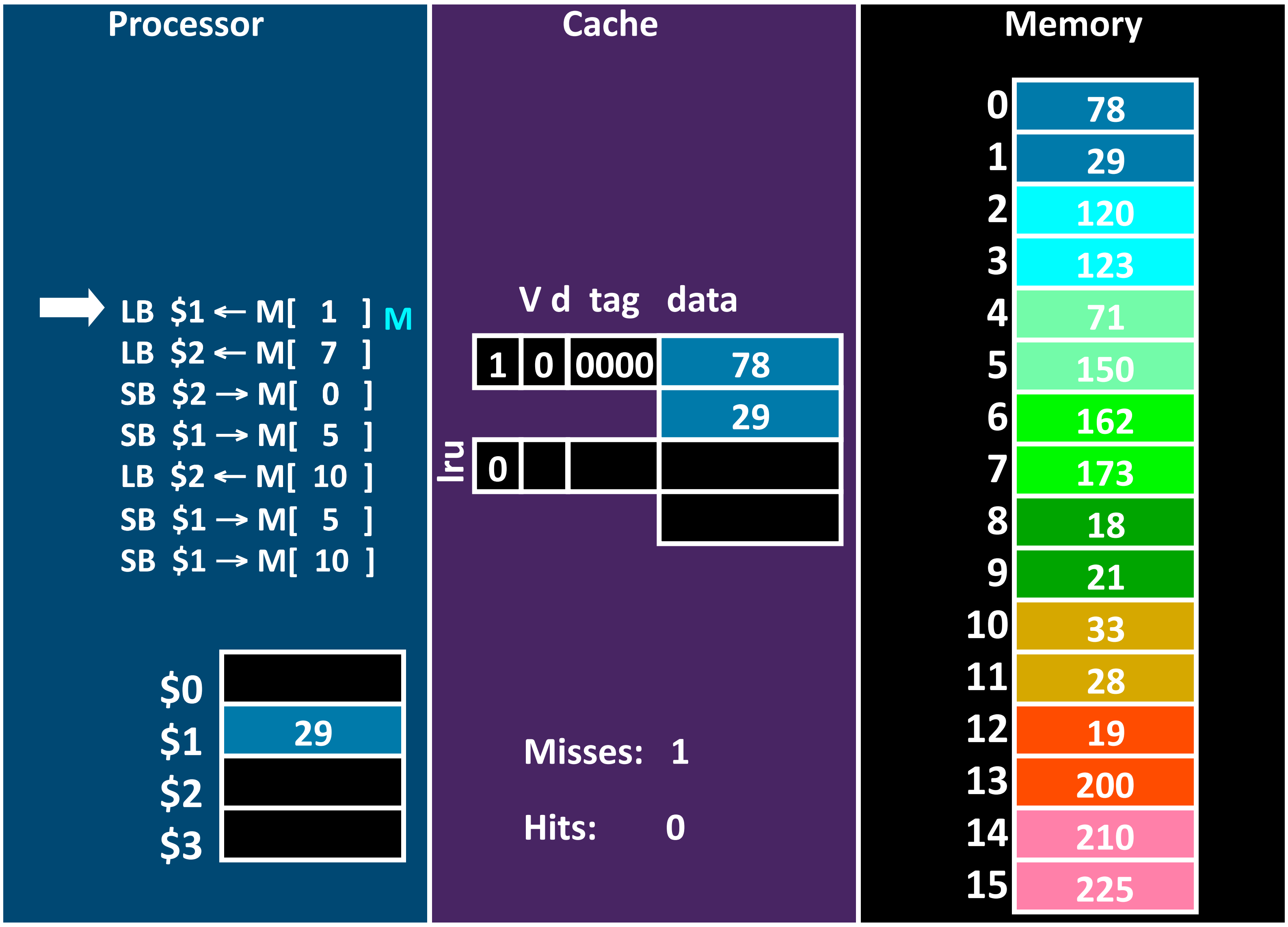
	V	d	tag	data
	1	0	0000	78
				29
lru	0			

Misses: 1

Hits: 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)



Processor

```
➡ LB  $1 ← M[  1  ] M
  LB  $2 ← M[  7  ]
  SB  $2 → M[  0  ]
  SB  $1 → M[  5  ]
  LB  $2 ← M[ 10  ]
  SB  $1 → M[  5  ]
  SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	
$3	

Cache

	V	d	tag	data
	1	0	0000	78
				29
lru	0			

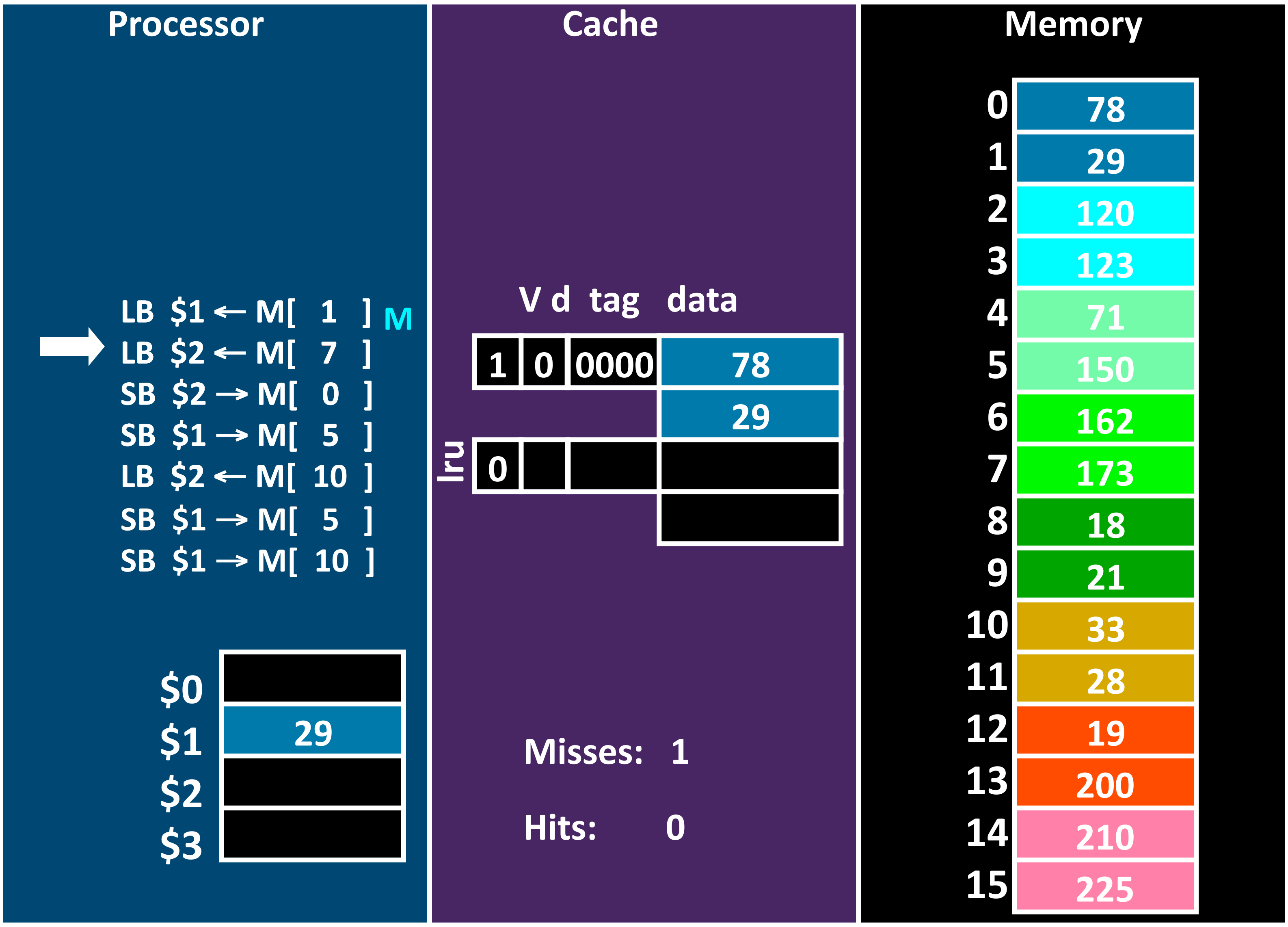
Misses: 1

Hits: 0

Memory

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 2)



Processor

```
   LB  $1 ← M[  1  ] M
➡  LB  $2 ← M[  7  ]
   SB  $2 → M[  0  ]
   SB  $1 → M[  5  ]
   LB  $2 ← M[ 10  ]
   SB  $1 → M[  5  ]
   SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	
$3	

Cache

	V	d	tag	data
	1	0	0000	78
				29
lru	0			

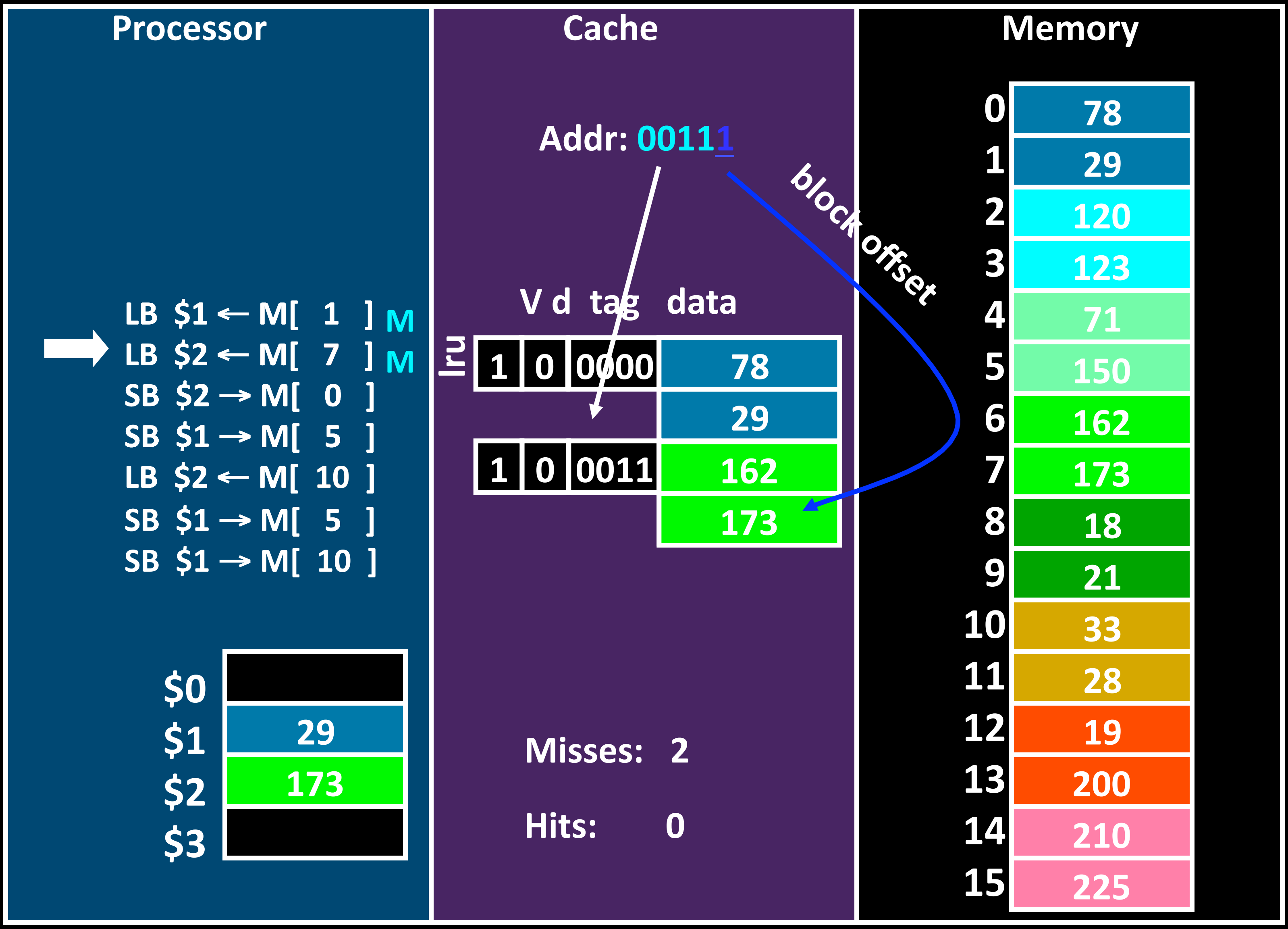
Misses: 1

Hits: 0

Memory

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 2)



Processor

```
    LB  $1 ← M[  1  ] M
⇨   LB  $2 ← M[  7  ] M
    SB  $2 → M[  0  ]
    SB  $1 → M[  5  ]
    LB  $2 ← M[ 10  ]
    SB  $1 → M[  5  ]
    SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

Cache

Addr: 00111

V	d	tag	data
1	0	0000	78
			29
1	0	0011	162
			173

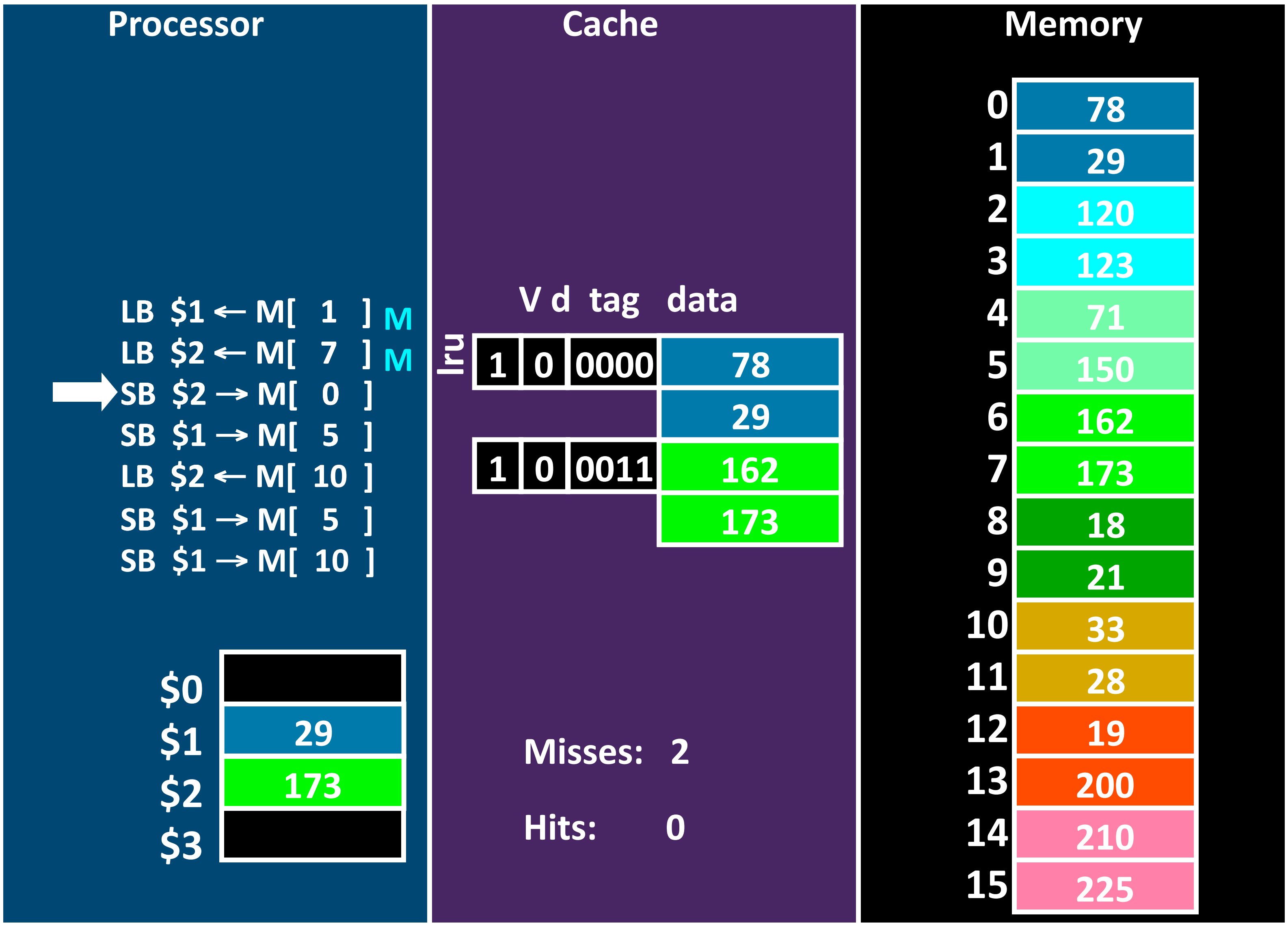
Misses: 2

Hits: 0

Memory

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 3)



Processor

```
  LB  $1 ← M[  1  ] M
  LB  $2 ← M[  7  ] M
⇨ SB  $2 → M[  0  ]
  SB  $1 → M[  5  ]
  LB  $2 ← M[ 10  ]
  SB  $1 → M[  5  ]
  SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

Cache

lru

V	d	tag	data
1	0	0000	78
			29
1	0	0011	162
			173

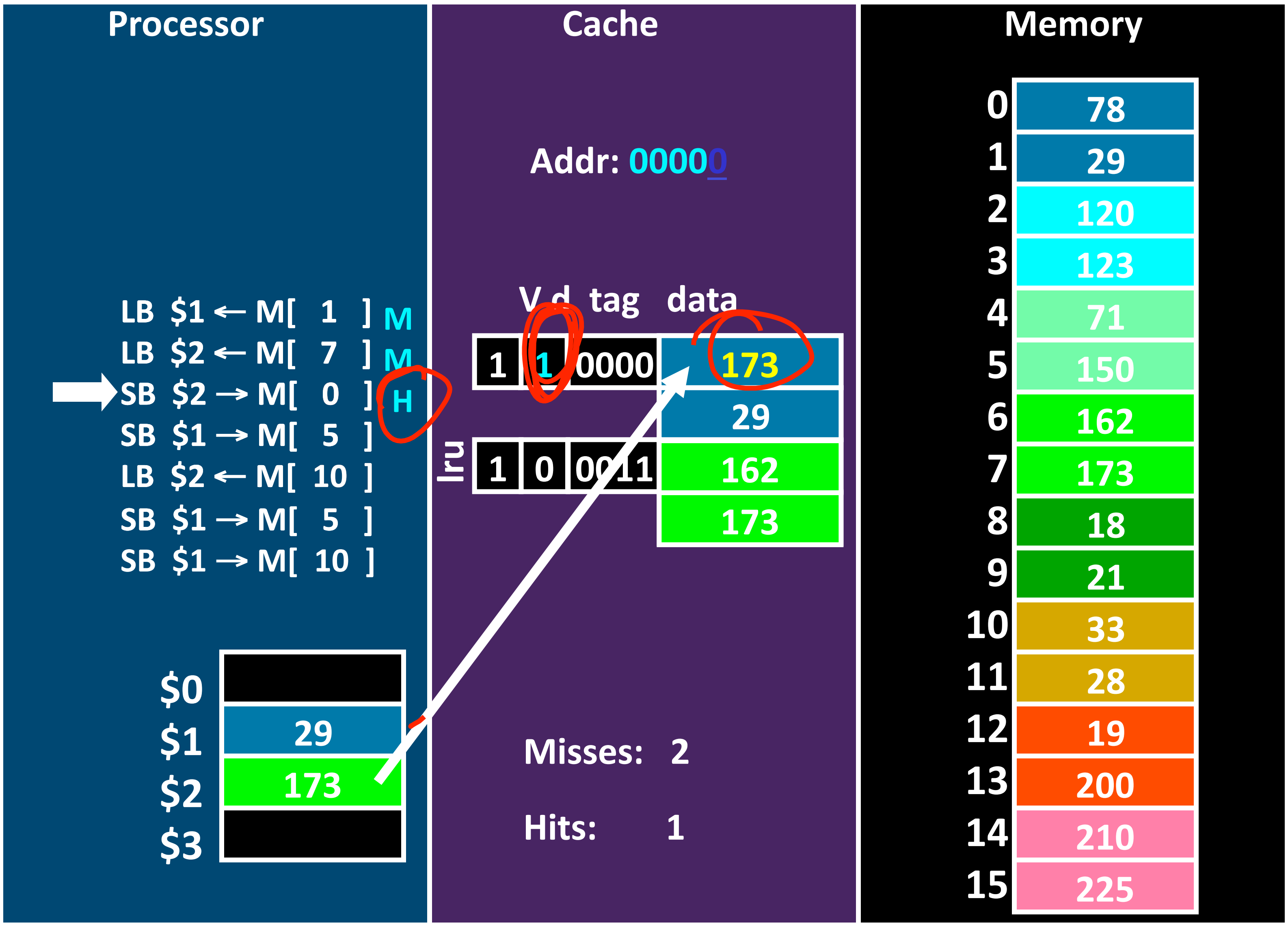
Misses: 2

Hits: 0

Memory

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 3)



Processor

```
LB $1 ← M[ 1  ] M
LB $2 ← M[ 7  ] M
SB $2 → M[ 0  ] H
SB $1 → M[ 5  ]
LB $2 ← M[ 10 ]
SB $1 → M[ 5  ]
SB $1 → M[ 10 ]
```

$0	
$1	29
$2	173
$3	

Cache

Addr: 00000

	V	d	tag	data
	1	1	0000	173 29
lru	1	0	0011	162 173

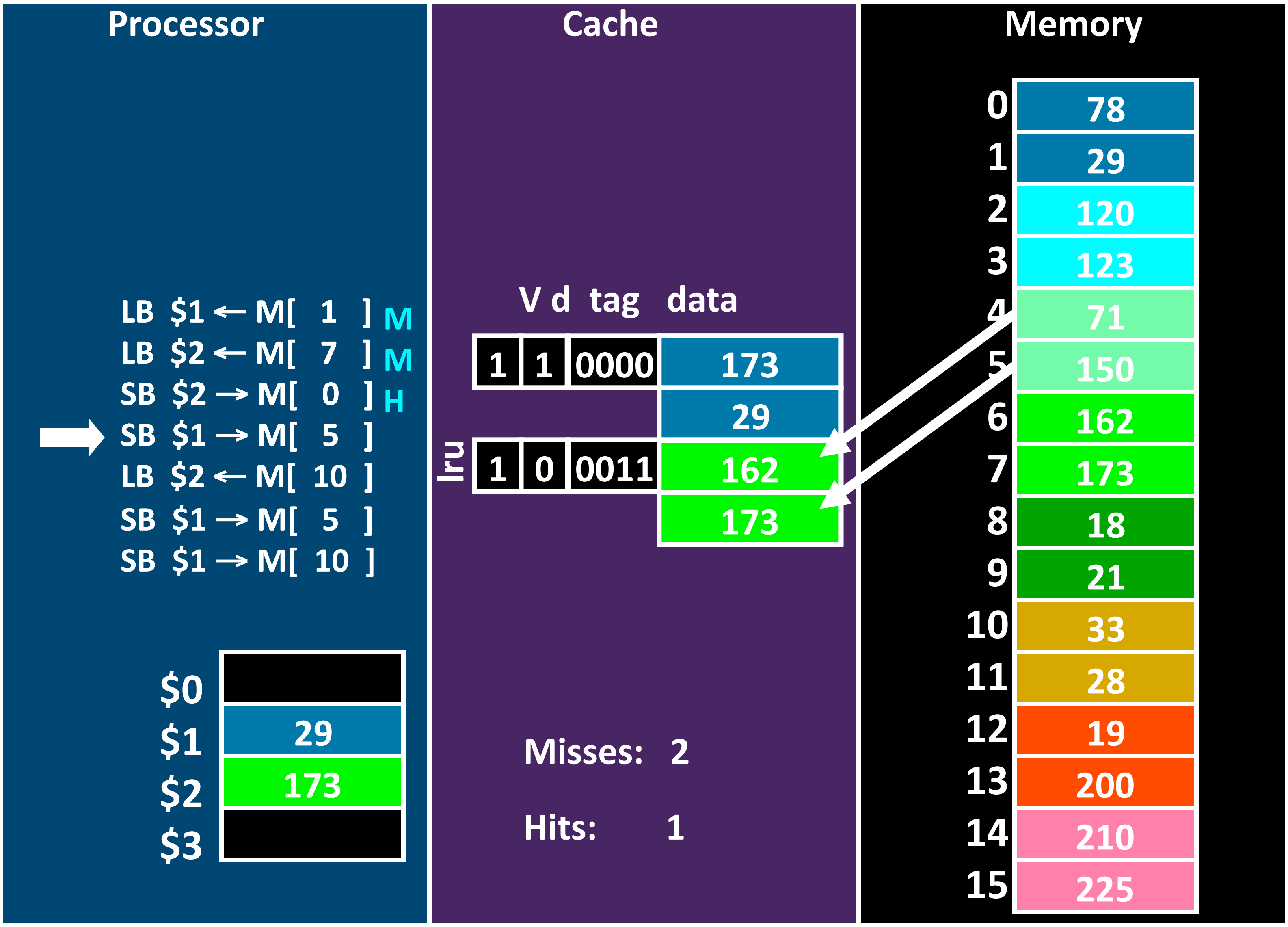
Misses: 2

Hits: 1

Memory

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 4)



Processor

```
LB $1 ← M[ 1 ]  M
LB $2 ← M[ 7 ]  M
SB $2 → M[ 0 ]  H
SB $1 → M[ 5 ]
LB $2 ← M[ 10 ]
SB $1 → M[ 5 ]
SB $1 → M[ 10 ]
```

$0	
$1	29
$2	173
$3	

Cache

	V	d	tag	data
	1	1	0000	173
				29
lru	1	0	0011	162
				173

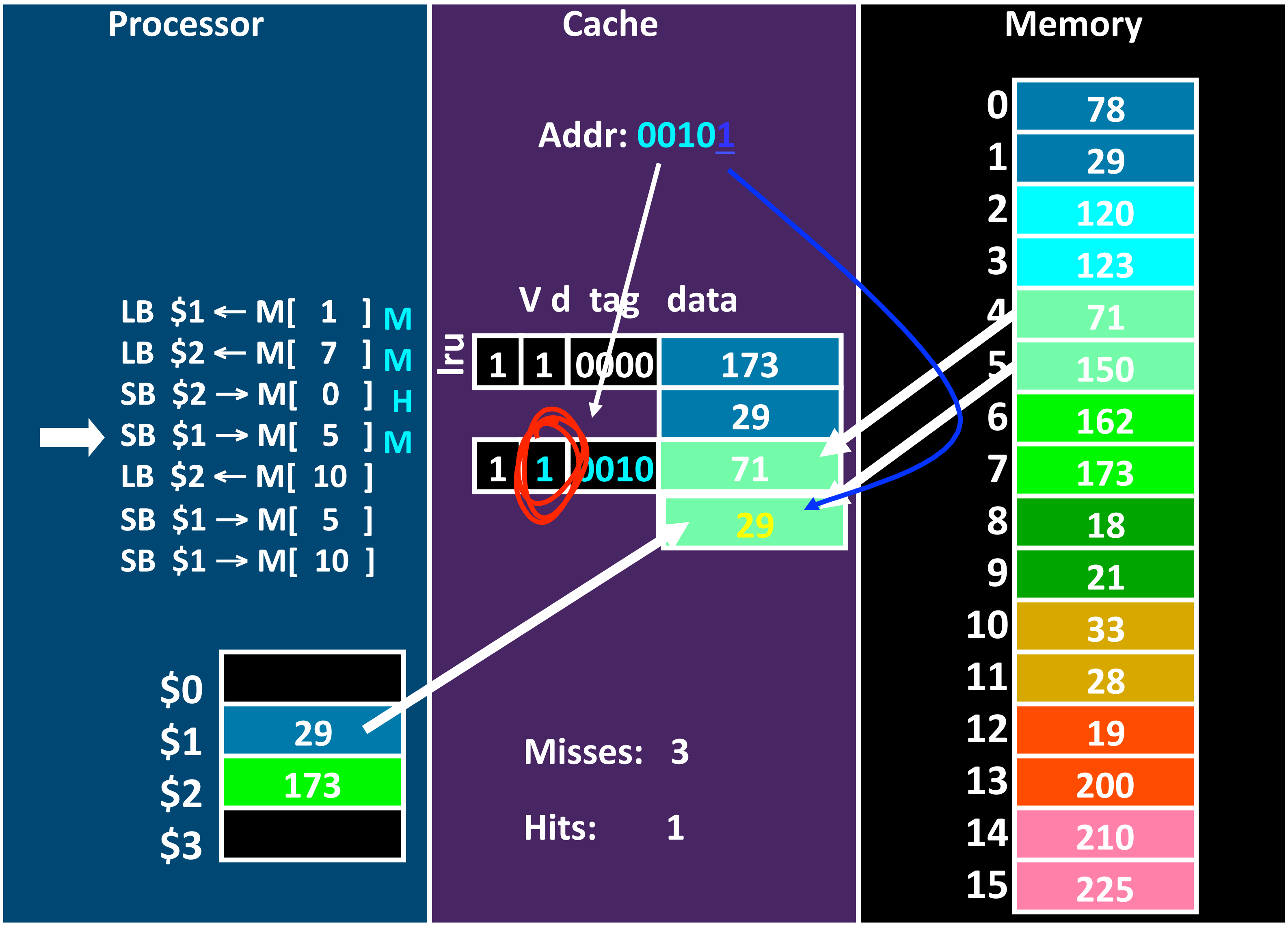
Misses: 2

Hits: 1

Memory

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 4)



Processor

```
LB  $1 ← M[  1  ]  M
LB  $2 ← M[  7  ]  M
SB  $2 → M[  0  ]  H
SB  $1 → M[  5  ]  M
LB  $2 ← M[ 10  ]
SB  $1 → M[  5  ]
SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

Cache

Addr: 00101

lru	V	d	tag	data
lru	1	1	0000	173
				29
	1	1	0010	71
				29

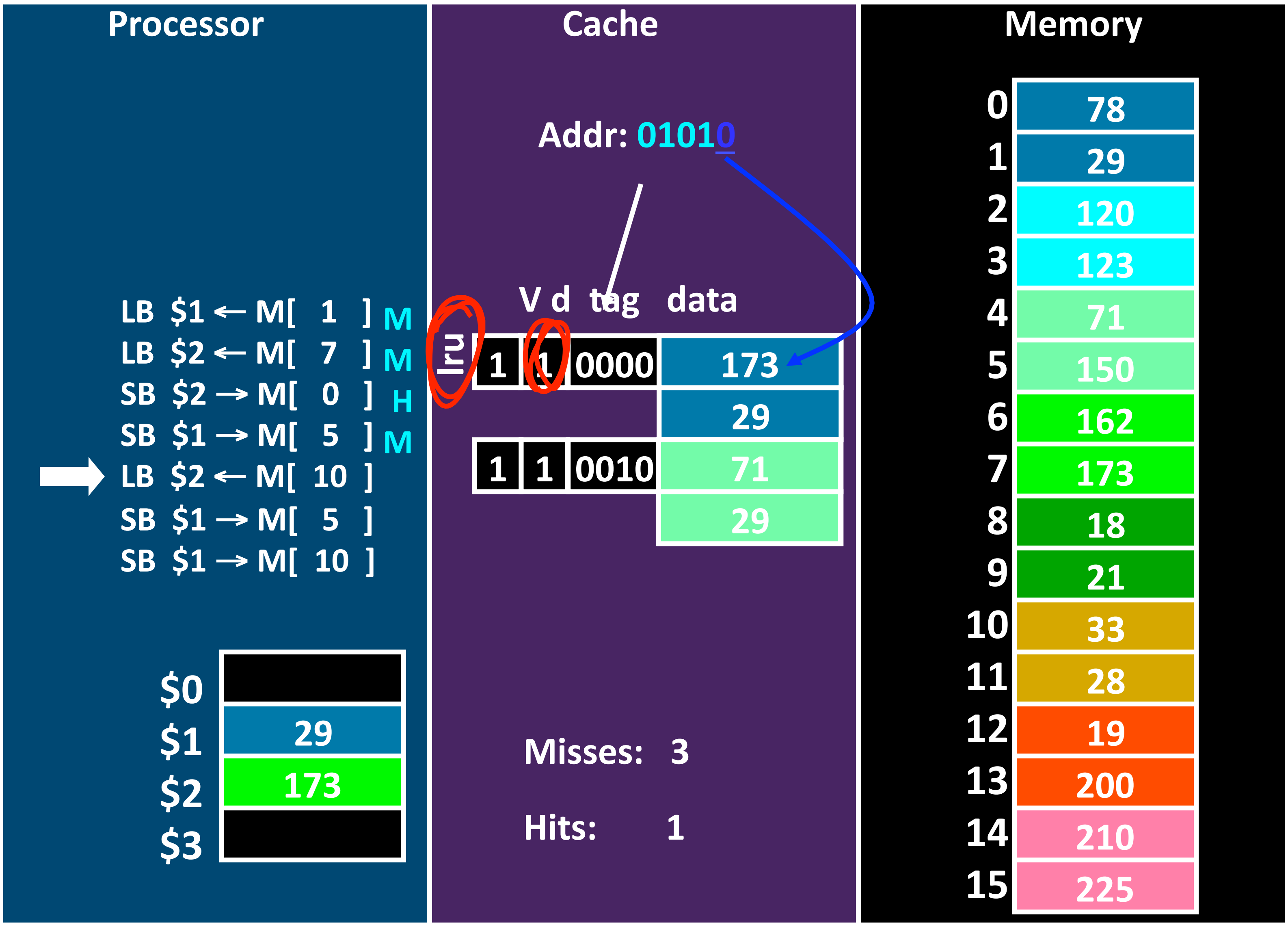
Misses: 3

Hits: 1

Memory

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 5)



Processor

```
   LB  $1 ← M[  1  ]  M
   LB  $2 ← M[  7  ]  M
   SB  $2 → M[  0  ]  H
   SB  $1 → M[  5  ]  M
⇨  LB  $2 ← M[ 10  ]
   SB  $1 → M[  5  ]
   SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

Cache

Addr: 01010

lru	V	d	tag	data
	1	1	0000	173
				29
	1	1	0010	71
				29

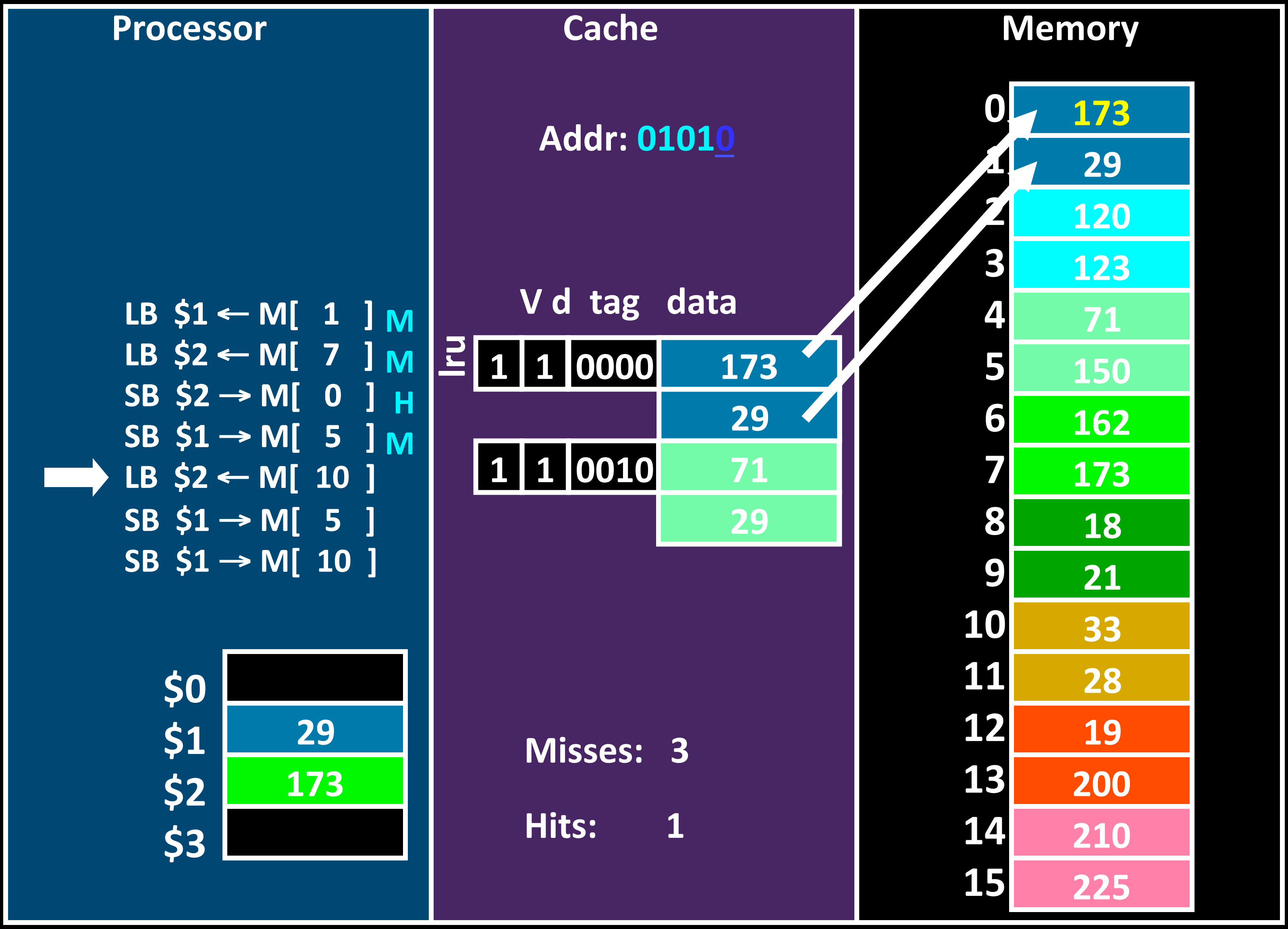
Misses: 3

Hits: 1

Memory

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 5)



Processor

```
LB  $1 ← M[  1  ] M
LB  $2 ← M[  7  ] M
SB  $2 → M[  0  ] H
SB  $1 → M[  5  ] M
LB  $2 ← M[ 10  ]
SB  $1 → M[  5  ]
SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	173
$3	

Cache

Addr: 01010

	V	d	tag	data
lru	1	1	0000	173
				29
	1	1	0010	71
				29

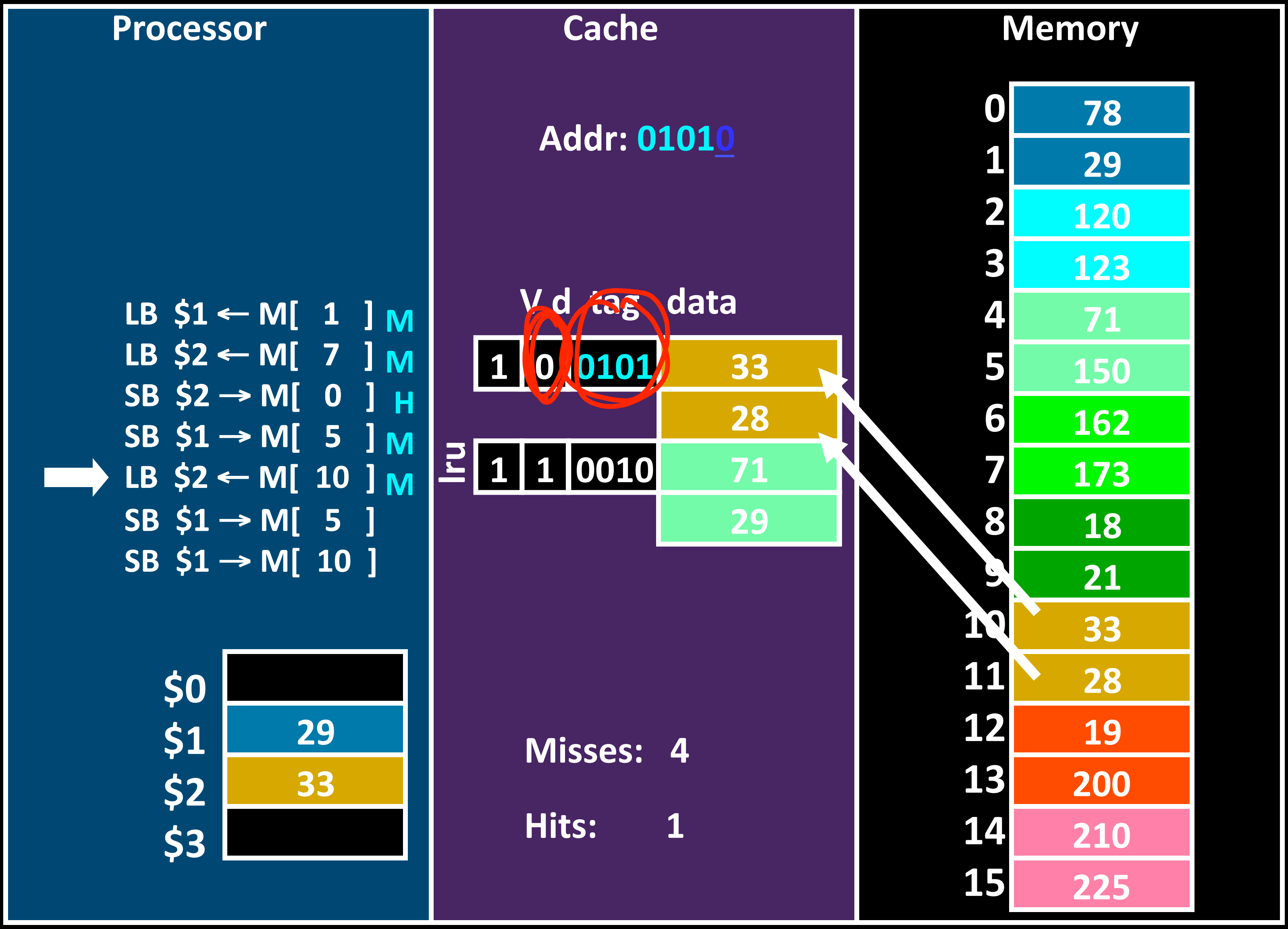
Misses: 3

Hits: 1

Memory

0	173
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 5)



Processor

```
LB $1 ← M[  1 ] M
LB $2 ← M[  7 ] M
SB $2 → M[  0 ] H
SB $1 → M[  5 ] M
LB $2 ← M[ 10 ] M
SB $1 → M[  5 ]
SB $1 → M[ 10 ]
```

$0	
$1	29
$2	33
$3	

Cache

Addr: 01010

V	d	tag	data
1	0	0101	33
			28
1	1	0010	71
			29

lru

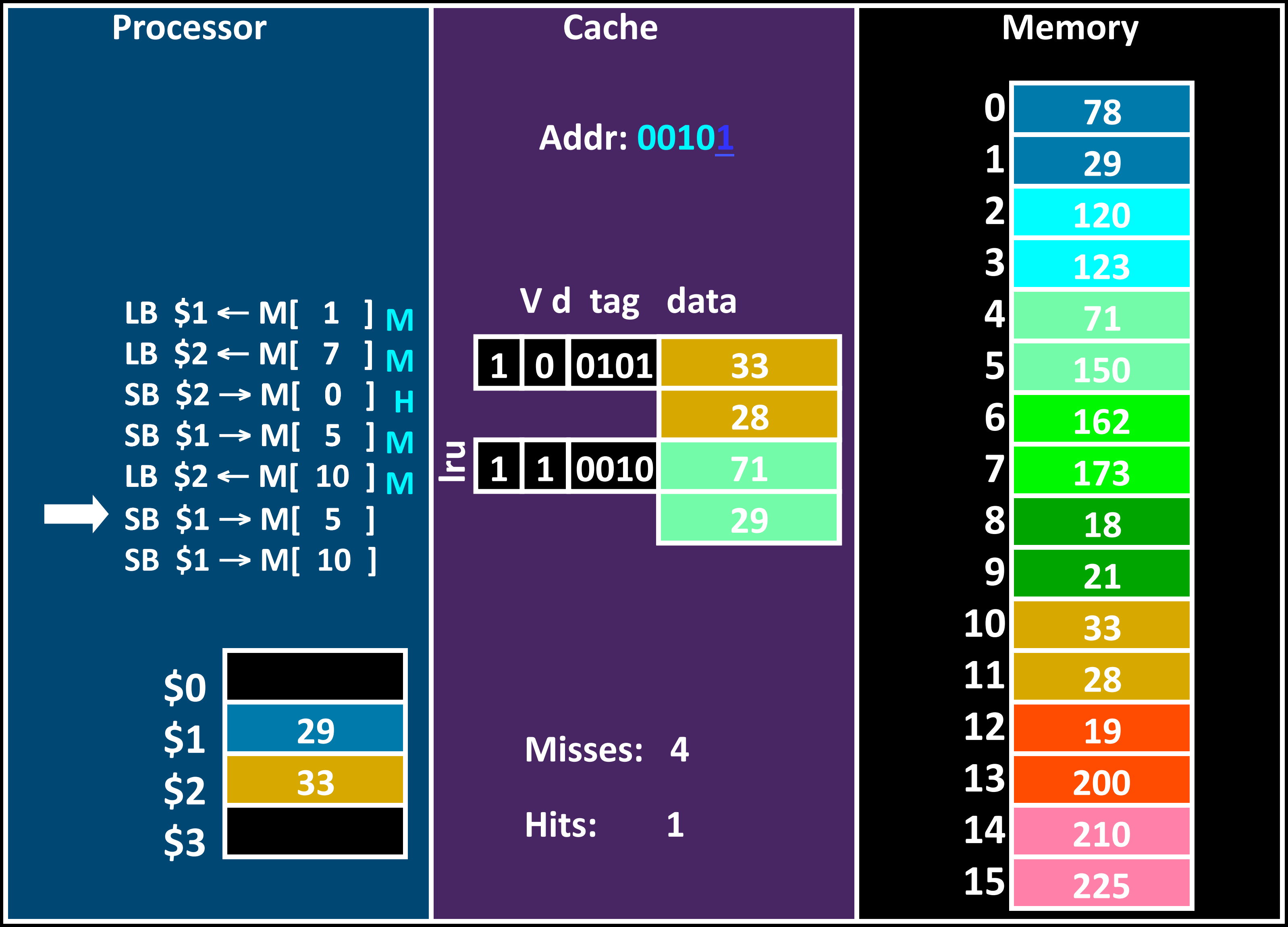
Misses: 4

Hits: 1

Memory

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 6)



Processor

LB $1 ← M[1] M
LB $2 ← M[7] M
SB $2 → M[0] H
SB $1 → M[5] M
LB $2 ← M[10] M
➡ SB $1 → M[5]
SB $1 → M[10]

$0	
$1	29
$2	33
$3	

Cache

Addr: 00101

V	d	tag	data
1	0	0101	33
			28
1	1	0010	71
			29

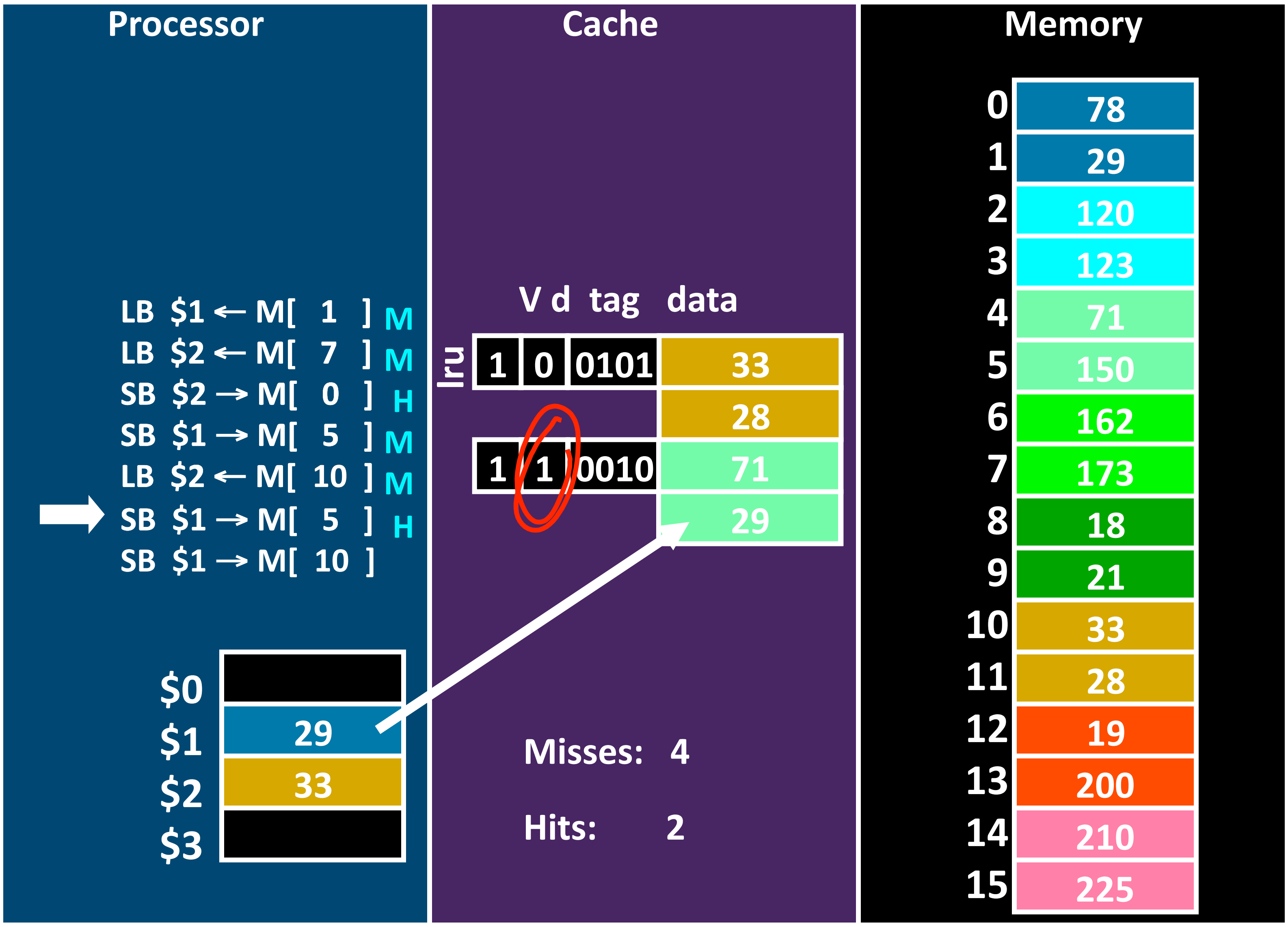
Misses: 4

Hits: 1

Memory

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 6)



Processor

```
  LB $1 ← M[  1 ] M
  LB $2 ← M[  7 ] M
  SB $2 → M[  0 ] H
  SB $1 → M[  5 ] M
  LB $2 ← M[ 10 ] M
➡ SB $1 → M[  5 ] H
  SB $1 → M[ 10 ]
```

$0	
$1	29
$2	33
$3	

Cache

lru	V	d	tag	data
lru	1	0	0101	33
				28
	1	1	0010	71
				29

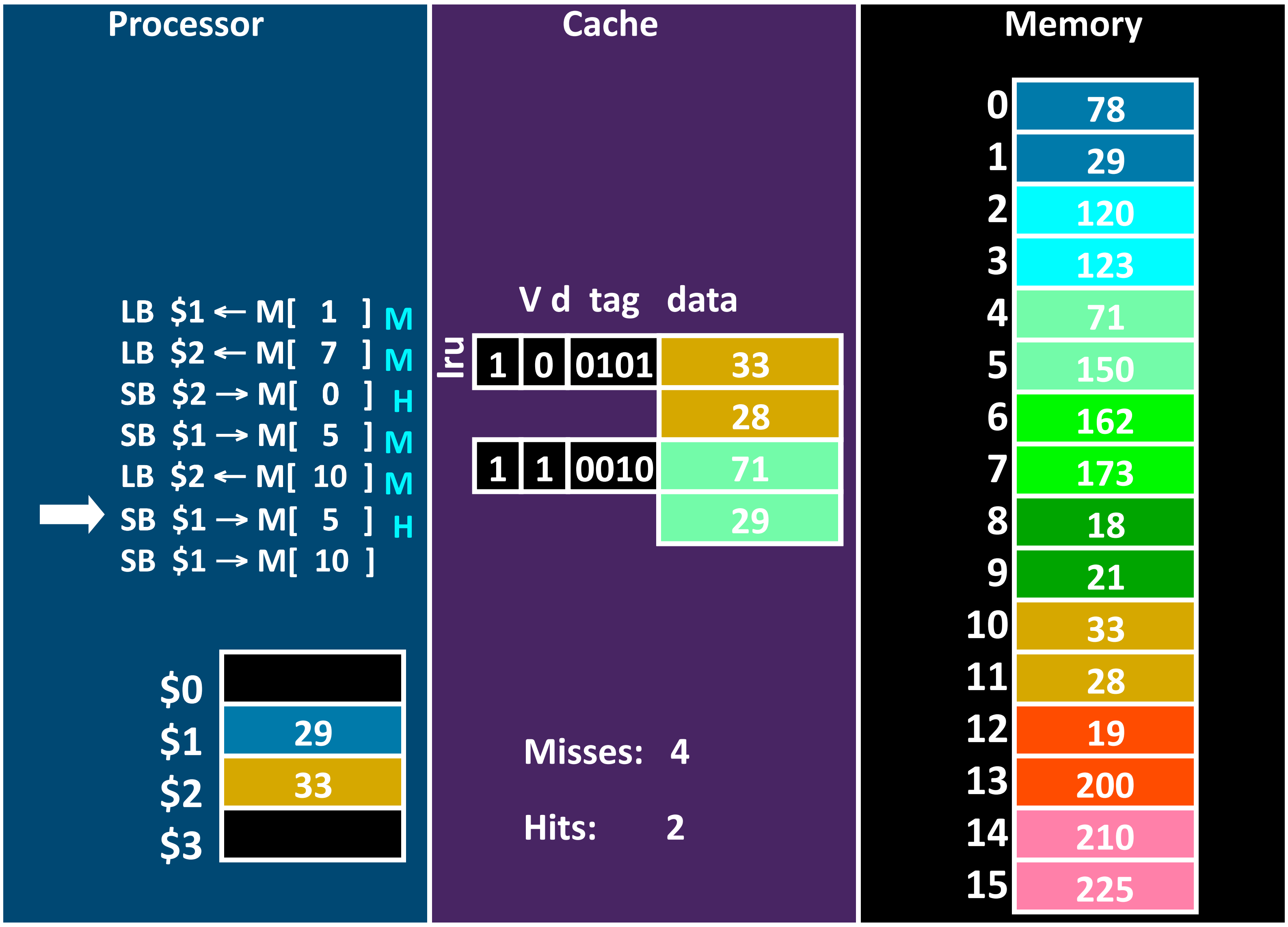
Misses: 4

Hits: 2

Memory

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 7)



Processor

```
  LB  $1 ← M[  1  ]  M
  LB  $2 ← M[  7  ]  M
  SB  $2 → M[  0  ]  H
  SB  $1 → M[  5  ]  M
  LB  $2 ← M[ 10  ]  M
⇨ SB  $1 → M[  5  ]  H
  SB  $1 → M[ 10  ]
```

$0	
$1	29
$2	33
$3	

Cache

lru	V	d	tag	data
lru	1	0	0101	33
				28
	1	1	0010	71
				29

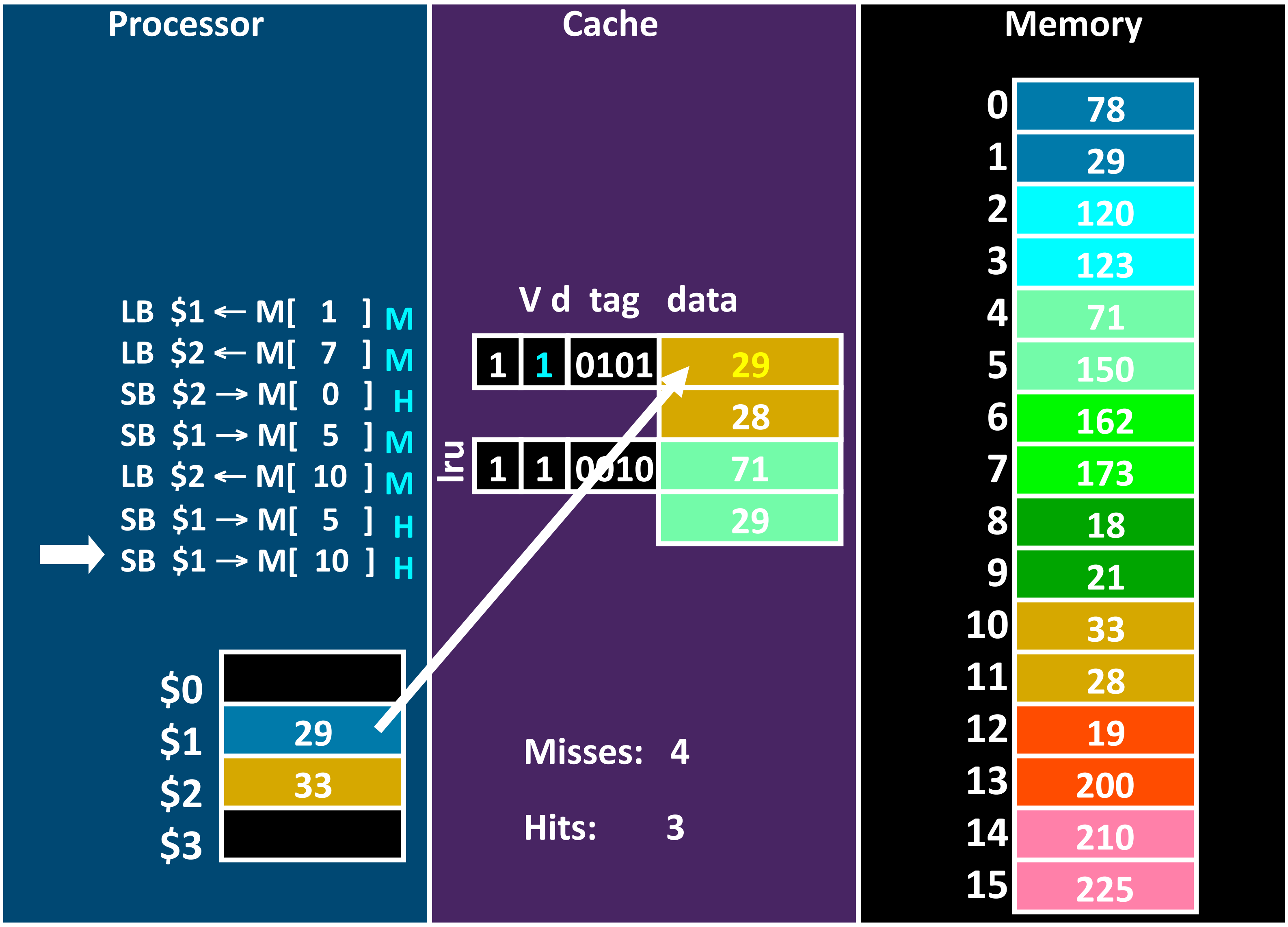
Misses: 4

Hits: 2

Memory

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 7)



Processor

```
LB  $1 ← M[  1 ]  M
LB  $2 ← M[  7 ]  M
SB  $2 → M[  0 ]  H
SB  $1 → M[  5 ]  M
LB  $2 ← M[ 10 ]  M
SB  $1 → M[  5 ]  H
SB  $1 → M[ 10 ]  H
```

$0	
$1	29
$2	33
$3	

Cache

	V	d	tag	data
	1	1	0101	29
				28
lru	1	1	0010	71
				29

Misses: 4

Hits: 3

Memory

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

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)

Write-Back (REF 8,9)

Processor

LB $1 ← M[1] M
LB $2 ← M[7] M
SB $2 → M[0] H
SB $1 → M[5] M
LB $2 ← M[10] M
SB $1 → M[5] H
SB $1 → M[10] H
SB $1 → M[5]
SB $1 → M[10]

$0	
$1	29
$2	33
$3	

Cache

	V	d	tag	data
	1	1	0101	29
				28
lru	1	1	0010	71
				29

Misses: 4

Hits: 3

Memory

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 8,9)

Processor

LB $1 ← M[1] M
LB $2 ← M[7] M
SB $2 → M[0] H
SB $1 → M[5] M
LB $2 ← M[10] M
SB $1 → M[5] H
SB $1 → M[10] H
SB $1 → M[5] H
SB $1 → M[10] H

$0	
$1	29
$2	33
$3	

Cache

	V	d	tag	data
	1	1	0101	29
				28
lru	1	1	0010	71
				29

Misses: 4

Hits: 5

Memory

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

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: 4/6/8/10/12/… etc words

So is write back just better?

What are other performance tradeoffs between write-through and write-back?

How can we further reduce penalty for cost of writes to memory?

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!**

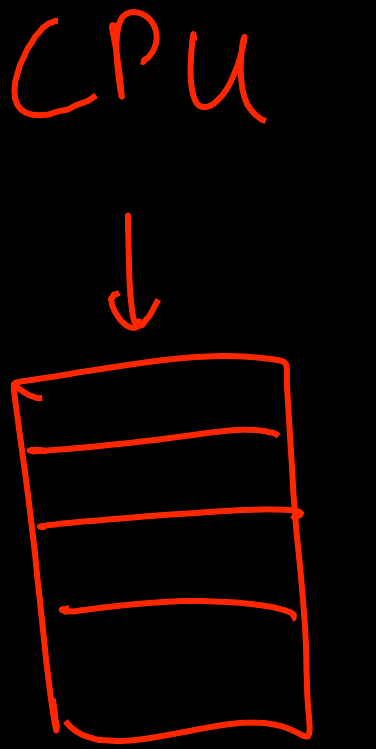
A: Use a write-back buffer

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

Q: When 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?

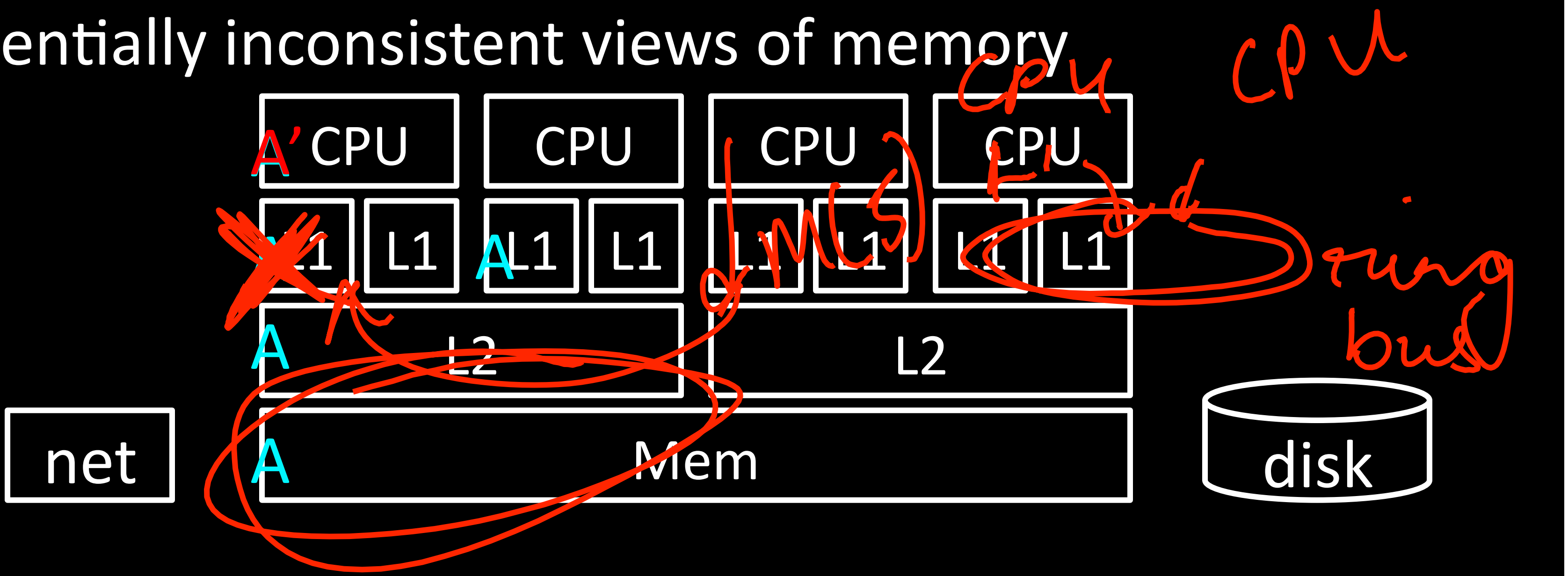
Write-back requires a cache coherency protocol

- 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

- 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

Summary: Write Through

Write-through policy with write allocate

- Cache miss: read entire block from memory
- Write: write only updated item to memory
- Eviction: no need to write to memory
- Slower, but cleaner

Write-back policy with write allocate

- Cache miss: read entire block from memory
  - But may need to write dirty cacheline first
- Write: nothing to memory
- Eviction: have to write to memory, entire cacheline because don’t know what is dirty (only 1 dirty bit)
- Faster, but complicated with multicore

Next Goal

Performance: What is the average memory access time (AMAT) for a cache?

AMAT = %hit x hit time + % miss x miss time

↓ ↓ ↓ ↓

90% 1 10% 100

.9 + 10 = 10.9

Cache Performance Example

Average Memory Access Time (AMAT)

Cache Performance (very simplified):

L1 (SRAM): 512 x 64 byte cache lines, direct mapped

Data cost: 3 cycle per word access

Hit = 5

Lookup cost: 2 cycle

16 words (i.e. 64 / 4 = 16)

Mem (DRAM): 4GB

Data cost: 50 cycle for first word, plus 3 cycles per subsequent word

$50 + 15 \times 3 = 95$

AMAT = %hit x hit time + % miss x miss time

Hit time = 5 cycles

Miss time = hit time + 50 (first word) + 15 x 3 (words)

= 100 cycles

$0.9 \times 5 + 0.1 \times 100 = 14.5$

If %hit = 90%, then

AMAT = .9 x 5 + .1 x 100 = 14.5 cycles

Cache Performance Example

Average Memory Access Time (AMAT)

Cache Performance (very simplified):

L1 (SRAM): 512 x 64 byte cache lines, direct mapped

Data cost: 3 cycle per word access

Lookup cost: 2 cycle

16 words (i.e. 64 / 4 = 16)

Mem (DRAM): 4GB

Data cost: 50 cycle for first word, plus 3 cycles per subsequent word

Multi Level Caching

Cache Performance (very simplified):

L1 (SRAM): 512 x 64 byte cache lines, direct mapped

Hit time: 5 cycles

L2 cache: bigger

Hit time = 20 cycles

Mem (DRAM): 4GB

Hit rate: 90% in L1, 90% in L2

5+20+95

= 126

AMAT = %hit x hit time + % miss x miss time

AMAT = .9 x 5 + .1 (.9 x 20 + .1 x 120) = 4.5 + .1 (18 + 12) = 7.5

L1 10%

Often: L1 fast and direct mapped, L2 bigger and higher associativity

Performance Summary

Average memory access time (AMAT)

depends on cache architecture and size

access time for hit,

miss penalty, miss rate

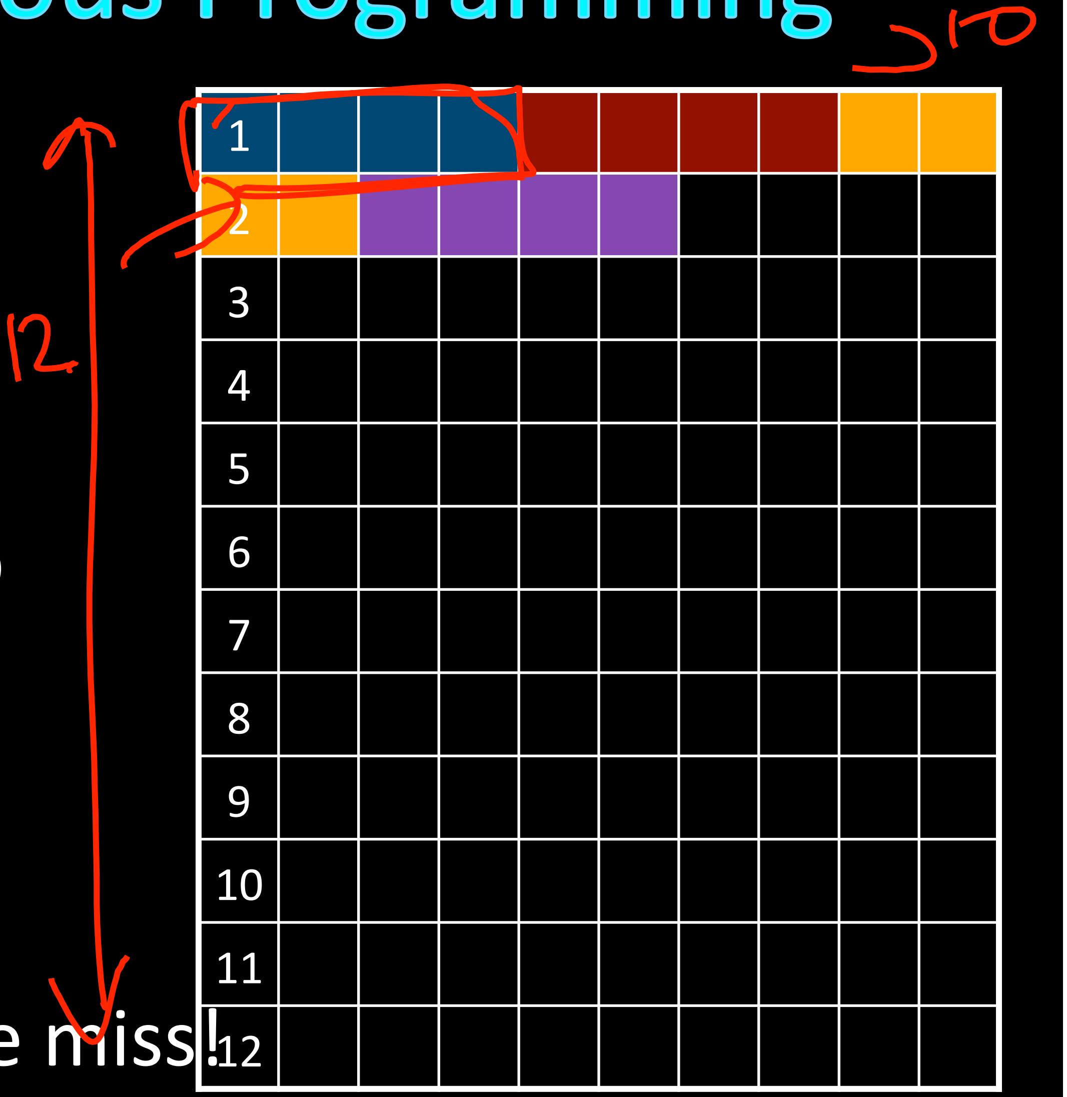
Cache design a very complex problem:

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

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];
```



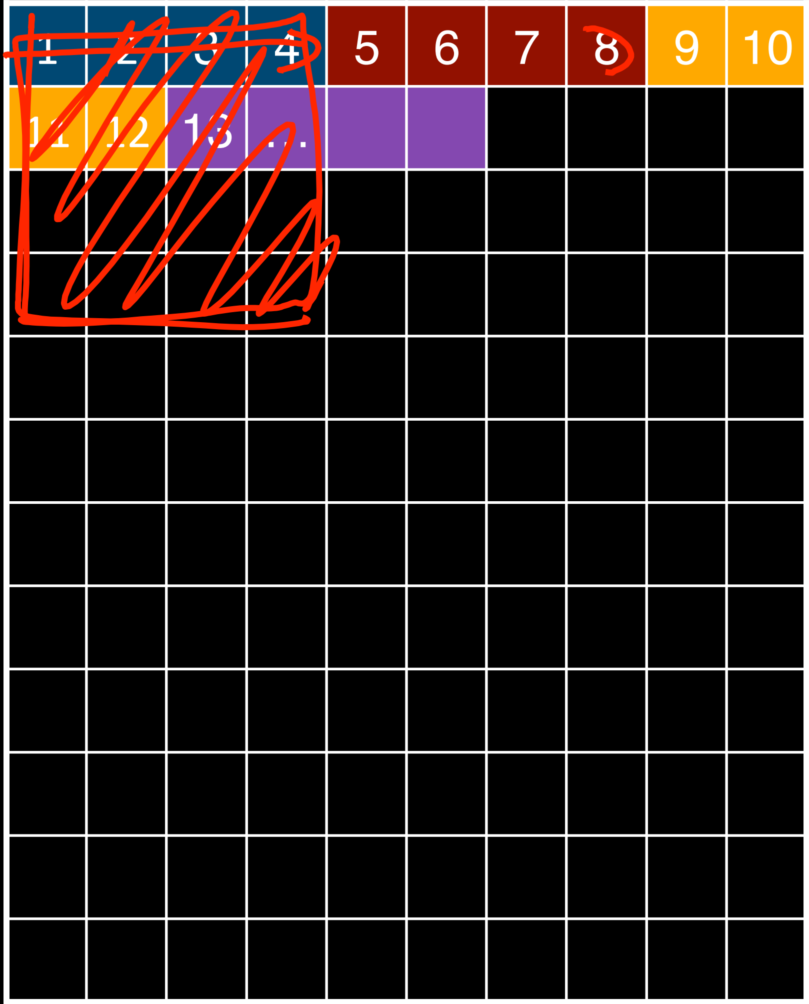
Every access is a cache miss!

(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];
```



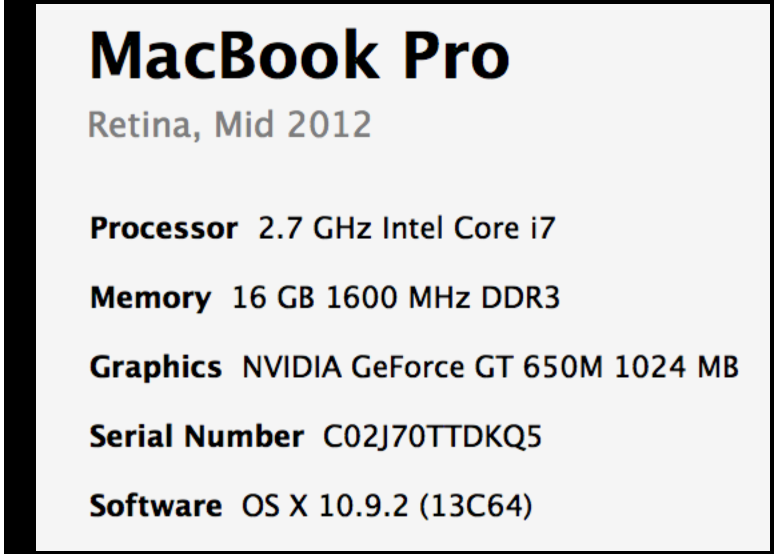
Block size = 4 →

Block size = 8 → 87.5% hit rate

Block size = 16 → 93.75% hit rate

And you can easily prefetch to warm the cache

By the end of the cache lectures…



MacBook Pro

Retina, Mid 2012

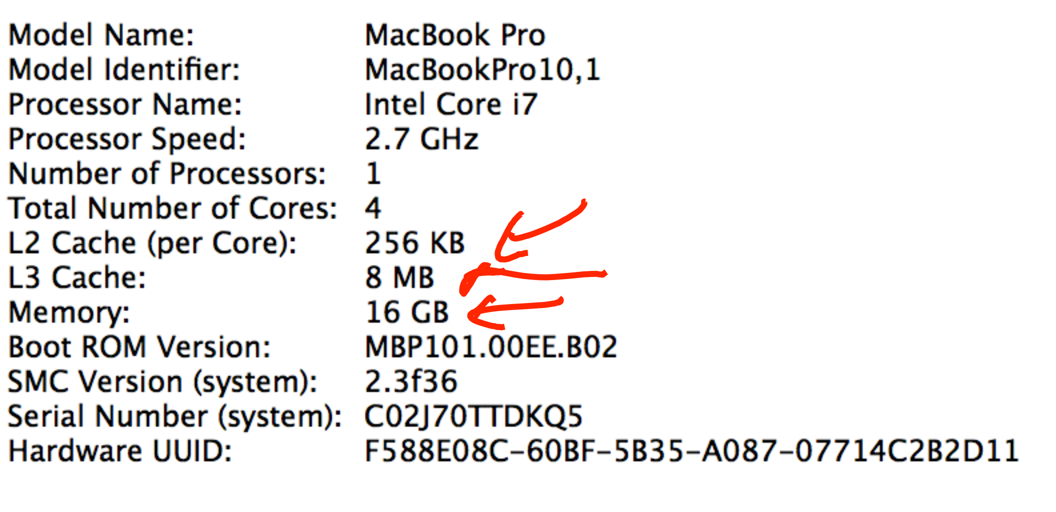
Processor 2.7 GHz Intel Core i7

Memory 16 GB 1600 MHz DDR3

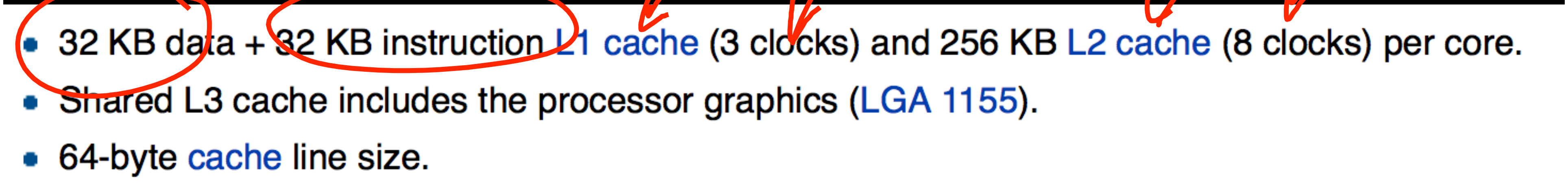
Graphics NVIDIA GeForce GT 650M 1024 MB

Serial Number C02J70TTDKQ5

Software OS X 10.9.2 (13C64)



```
Model Name:                MacBook Pro
Model Identifier:          MacBookPro10,1
Processor Name:            Intel Core i7
Processor Speed:           2.7 GHz
Number of Processors:      1
Total Number of Cores:     4
L2 Cache (per Core):       256 KB
L3 Cache:                  8 MB
Memory:                    16 GB
Boot ROM Version:          MBP101.00EE.B02
SMC Version (system):      2.3f36
Serial Number (system):    C02J70TTDKQ5
Hardware UUID:             F588E08C-60BF-5B35-A087-07714C2B2D11
```



- 32 KB data + 32 KB instruction L1 cache (3 clocks) and 256 KB L2 cache (8 clocks) per core.
- Shared L3 cache includes the processor graphics (LGA 1155).
- 64-byte cache line size.

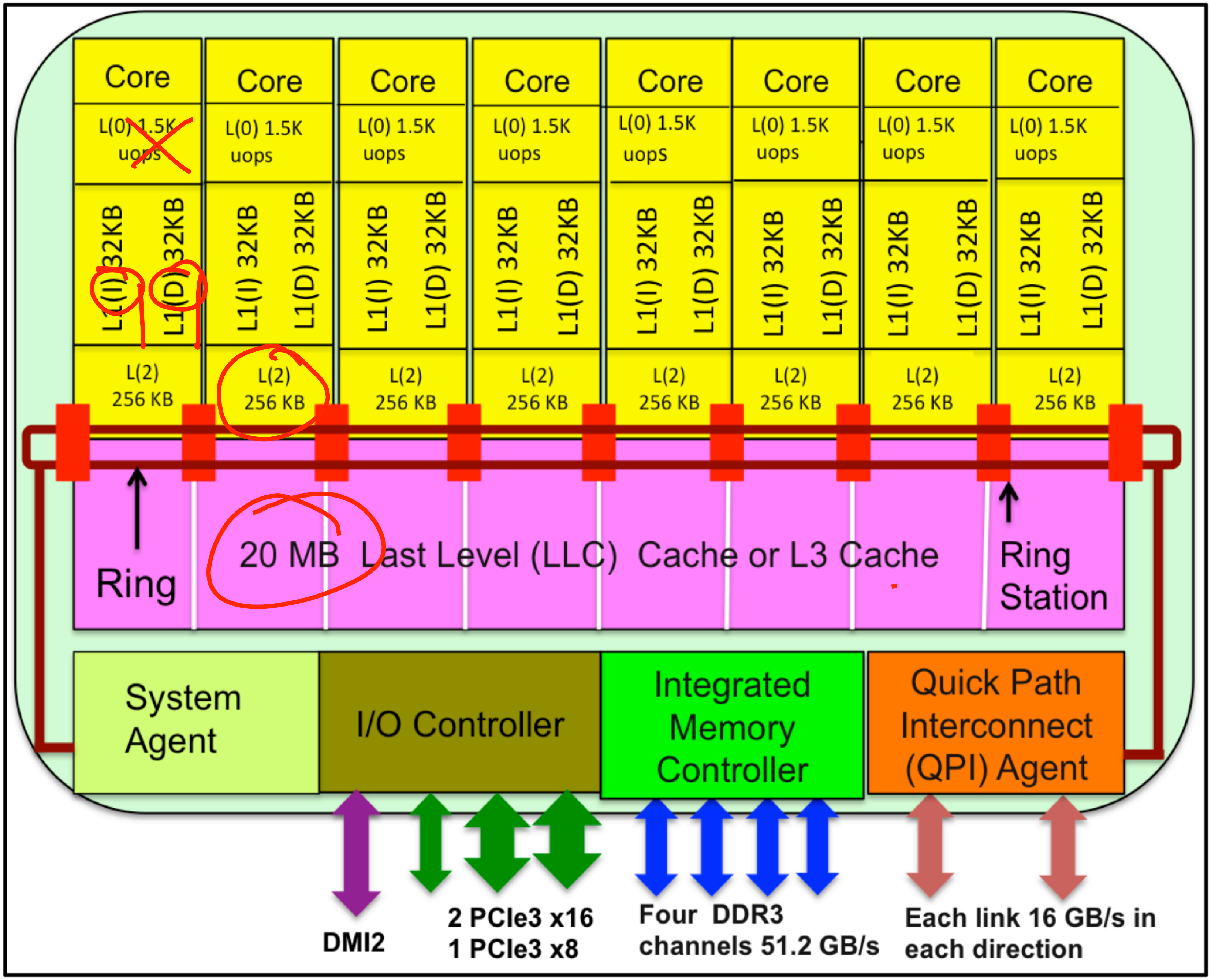

Core
L(0) 1.5K uops
L1(I) 32KB
L1(D) 32KB
L(2) 256 KB
Ring
20 MB Last Level (LLC) Cache or L3 Cache
Ring Station
System Agent
I/O Controller
Integrated Memory Controller
Quick Path Interconnect (QPI) Agent
DMI2
2 PCIe3 x16
1 PCIe3 x8
Four DDR3 channels 51.2 GB/s
Each link 16 GB/s in each direction

Figure 1. Schematic diagram of a Sandy Bridge processor.

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

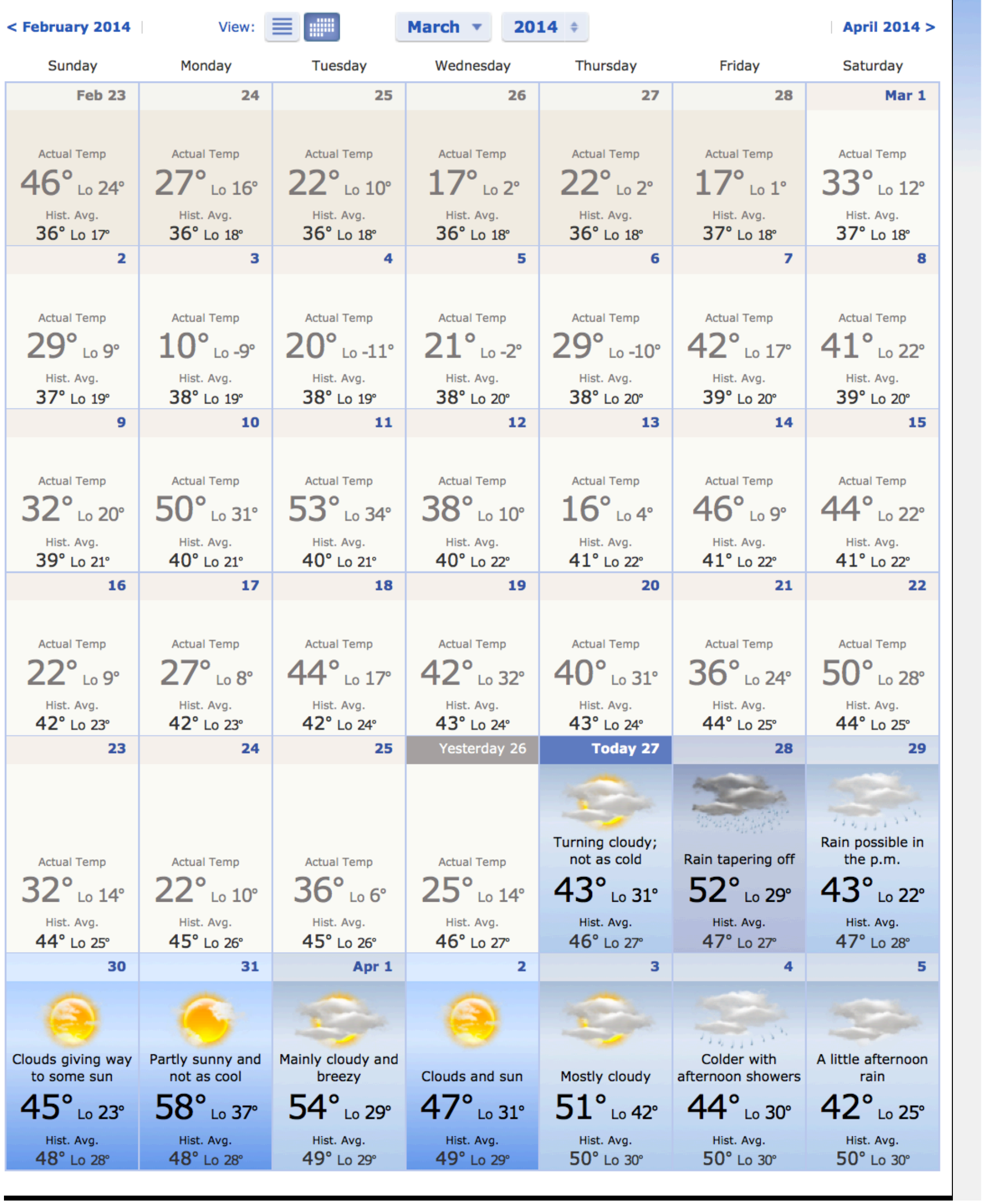
Design space is huge

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

Multi-core / Multi-Processor is complicated

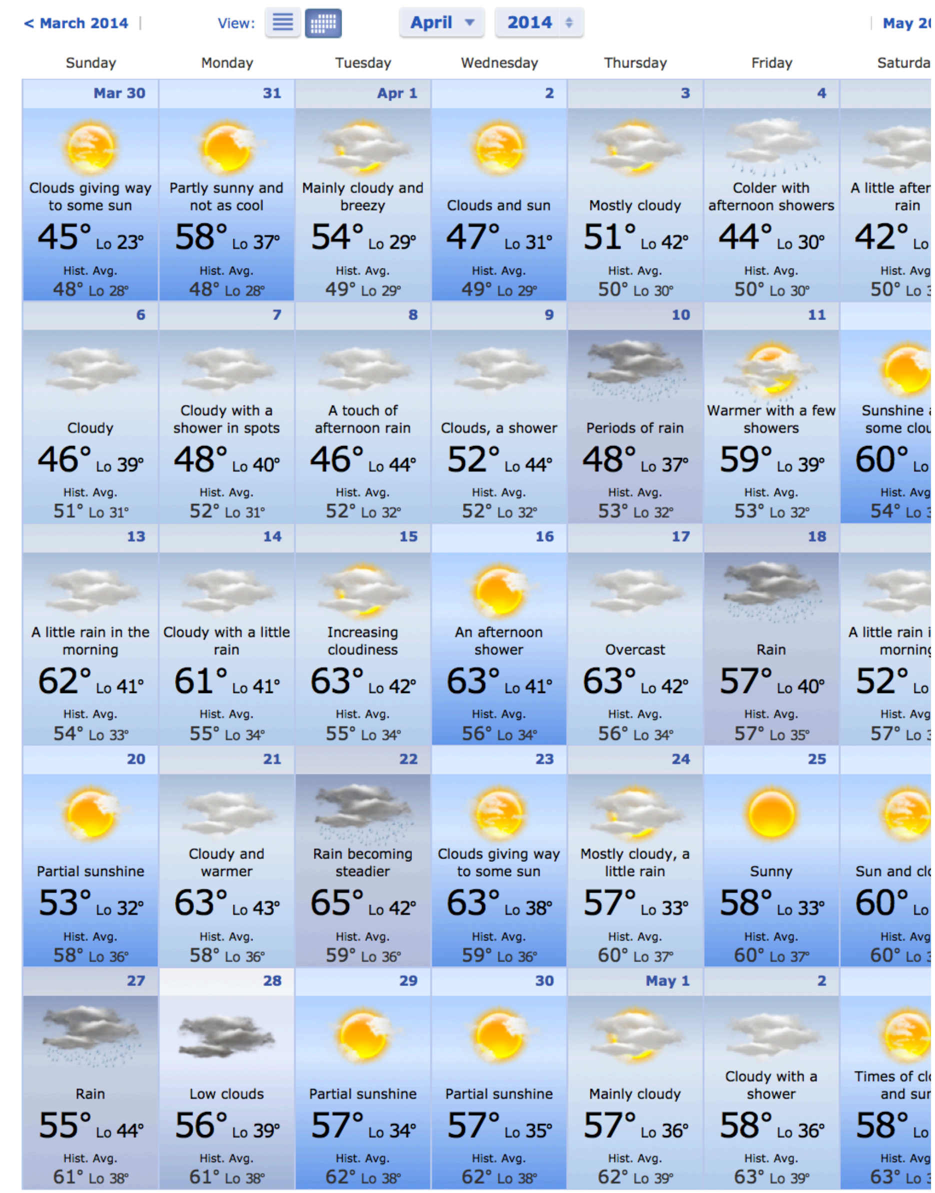
- Inconsistent views of memory
- Extremely complex protocols, very hard to get right

Spring is here!



< February 2014 | View: March 2014 | April 2014 >

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Feb 23 Actual Temp 46° Lo 24° Hist. Avg. 36° Lo 17°	24 Actual Temp 27° Lo 16° Hist. Avg. 36° Lo 18°	25 Actual Temp 22° Lo 10° Hist. Avg. 36° Lo 18°	26 Actual Temp 17° Lo 2° Hist. Avg. 36° Lo 18°	27 Actual Temp 22° Lo 2° Hist. Avg. 36° Lo 18°	28 Actual Temp 17° Lo 1° Hist. Avg. 37° Lo 18°	Mar 1 Actual Temp 33° Lo 12° Hist. Avg. 37° Lo 18°
2 Actual Temp 29° Lo 9° Hist. Avg. 37° Lo 19°	3 Actual Temp 10° Lo -9° Hist. Avg. 38° Lo 19°	4 Actual Temp 20° Lo -11° Hist. Avg. 38° Lo 19°	5 Actual Temp 21° Lo -2° Hist. Avg. 38° Lo 20°	6 Actual Temp 29° Lo -10° Hist. Avg. 38° Lo 20°	7 Actual Temp 42° Lo 17° Hist. Avg. 39° Lo 20°	8 Actual Temp 41° Lo 22° Hist. Avg. 39° Lo 20°
9 Actual Temp 32° Lo 20° Hist. Avg. 39° Lo 21°	10 Actual Temp 50° Lo 31° Hist. Avg. 40° Lo 21°	11 Actual Temp 53° Lo 34° Hist. Avg. 40° Lo 21°	12 Actual Temp 38° Lo 10° Hist. Avg. 40° Lo 22°	13 Actual Temp 16° Lo 4° Hist. Avg. 41° Lo 22°	14 Actual Temp 46° Lo 9° Hist. Avg. 41° Lo 22°	15 Actual Temp 44° Lo 22° Hist. Avg. 41° Lo 22°
16 Actual Temp 22° Lo 9° Hist. Avg. 42° Lo 23°	17 Actual Temp 27° Lo 8° Hist. Avg. 42° Lo 23°	18 Actual Temp 44° Lo 17° Hist. Avg. 42° Lo 24°	19 Actual Temp 42° Lo 32° Hist. Avg. 43° Lo 24°	20 Actual Temp 40° Lo 31° Hist. Avg. 43° Lo 24°	21 Actual Temp 36° Lo 24° Hist. Avg. 44° Lo 25°	22 Actual Temp 50° Lo 28° Hist. Avg. 44° Lo 25°
23 Actual Temp 32° Lo 14° Hist. Avg. 44° Lo 25°	24 Actual Temp 22° Lo 10° Hist. Avg. 45° Lo 26°	25 Actual Temp 36° Lo 6° Hist. Avg. 45° Lo 26°	Yesterday 26 Actual Temp 25° Lo 14° Hist. Avg. 46° Lo 27°	Today 27 Turning cloudy; not as cold 43° Lo 31° Hist. Avg. 46° Lo 27°	28 Rain tapering off 52° Lo 29° Hist. Avg. 47° Lo 27°	29 Rain possible in the p.m. 43° Lo 22° Hist. Avg. 47° Lo 28°
30 Clouds giving way to some sun 45° Lo 23° Hist. Avg. 48° Lo 28°	31 Partly sunny and not as cool 58° Lo 37° Hist. Avg. 48° Lo 28°	Apr 1 Mainly cloudy and breezy 54° Lo 29° Hist. Avg. 49° Lo 29°	2 Clouds and sun 47° Lo 31° Hist. Avg. 49° Lo 29°	3 Mostly cloudy 51° Lo 42° Hist. Avg. 50° Lo 30°	4 Colder with afternoon showers 44° Lo 30° Hist. Avg. 50° Lo 30°	5 A little afternoon rain 42° Lo 25° Hist. Avg. 50° Lo 30°



< March 2014 | View: April 2014 | May 20

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturda
Mar 30 Clouds giving way to some sun 45° Lo 23° Hist. Avg. 48° Lo 28°	31 Partly sunny and not as cool 58° Lo 37° Hist. Avg. 48° Lo 28°	Apr 1 Mainly cloudy and breezy 54° Lo 29° Hist. Avg. 49° Lo 29°	2 Clouds and sun 47° Lo 31° Hist. Avg. 49° Lo 29°	3 Mostly cloudy 51° Lo 42° Hist. Avg. 50° Lo 30°	4 Colder with afternoon showers 44° Lo 30° Hist. Avg. 50° Lo 30°	A little after rain 42° Lo Hist. Avg 50° Lo 3
6 Cloudy 46° Lo 39° Hist. Avg. 51° Lo 31°	7 Cloudy with a shower in spots 48° Lo 40° Hist. Avg. 52° Lo 31°	8 A touch of afternoon rain 46° Lo 44° Hist. Avg. 52° Lo 32°	9 Clouds, a shower 52° Lo 44° Hist. Avg. 52° Lo 32°	10 Periods of rain 48° Lo 37° Hist. Avg. 53° Lo 32°	11 Warmer with a few showers 59° Lo 39° Hist. Avg. 53° Lo 32°	Sunshine a some clou 60° Lo Hist. Avg 54° Lo 3
13 A little rain in the morning 62° Lo 41° Hist. Avg. 54° Lo 33°	14 Cloudy with a little rain 61° Lo 41° Hist. Avg. 55° Lo 34°	15 Increasing cloudiness 63° Lo 42° Hist. Avg. 55° Lo 34°	16 An afternoon shower 63° Lo 41° Hist. Avg. 56° Lo 34°	17 Overcast 63° Lo 42° Hist. Avg. 56° Lo 34°	18 Rain 57° Lo 40° Hist. Avg. 57° Lo 35°	A little rain i morning 52° Lo Hist. Avg 57° Lo 3
20 Partial sunshine 53° Lo 32° Hist. Avg. 58° Lo 36°	21 Cloudy and warmer 63° Lo 43° Hist. Avg. 58° Lo 36°	22 Rain becoming steadier 65° Lo 42° Hist. Avg. 59° Lo 36°	23 Clouds giving way to some sun 63° Lo 38° Hist. Avg. 59° Lo 36°	24 Mostly cloudy, a little rain 57° Lo 33° Hist. Avg. 60° Lo 37°	25 Sunny 58° Lo 33° Hist. Avg. 60° Lo 37°	Sun and cl 60° Lo Hist. Avg 60° Lo 3
27 Rain 55° Lo 44° Hist. Avg. 61° Lo 38°	28 Low clouds 56° Lo 39° Hist. Avg. 61° Lo 38°	29 Partial sunshine 57° Lo 34° Hist. Avg. 62° Lo 38°	30 Partial sunshine 57° Lo 35° Hist. Avg. 62° Lo 38°	May 1 Mainly cloudy 57° Lo 36° Hist. Avg. 62° Lo 39°	2 Cloudy with a shower 58° Lo 36° Hist. Avg. 63° Lo 39°	Times of cl and sur 58° Lo Hist. Avg 63° Lo 3