Caches and Memory

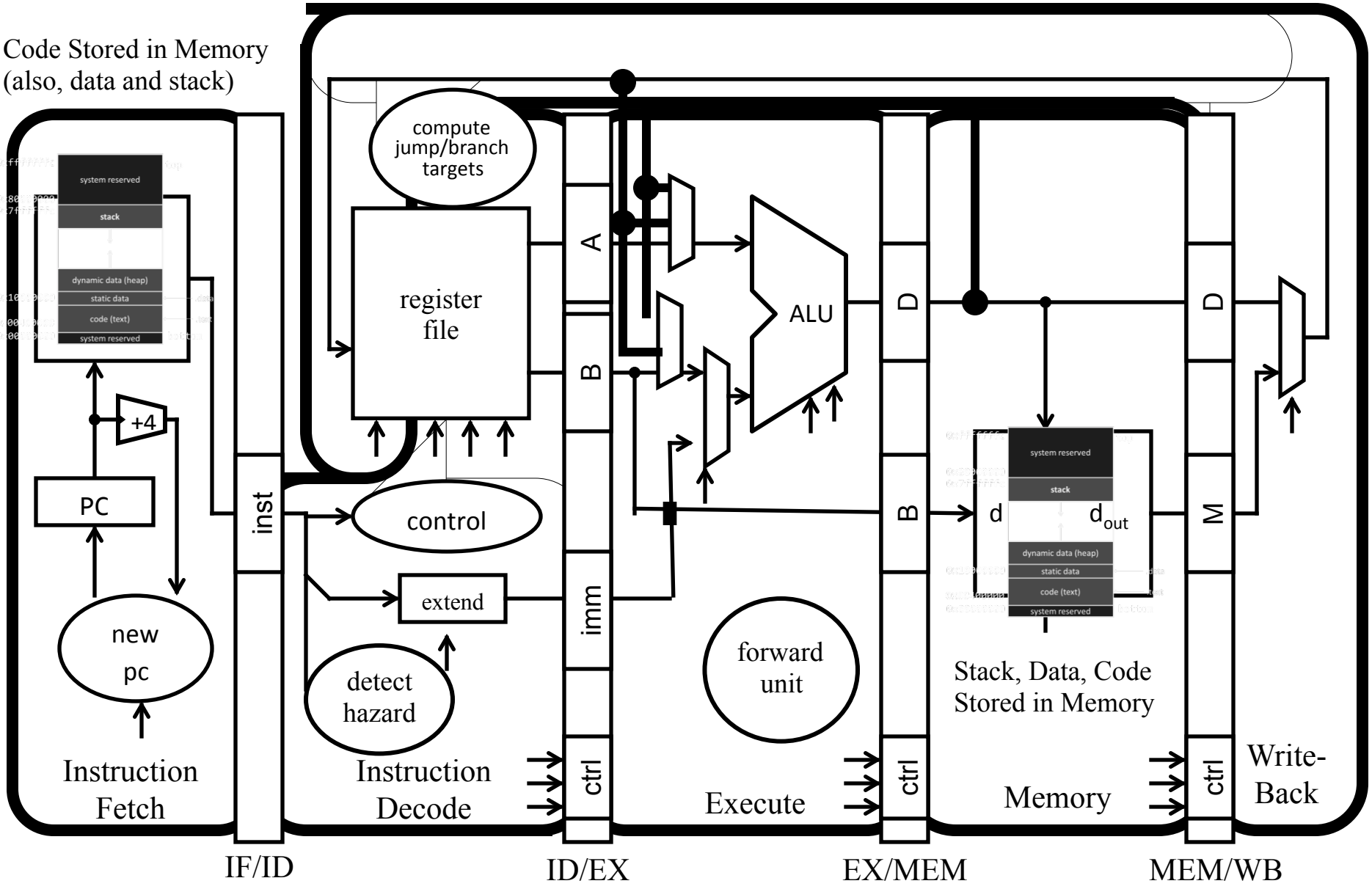
CS 3410, Spring 2015

Computer Science

Cornell University

See P&H Chapter: 5.1-5.3 (except writes)

Big Picture: Memory

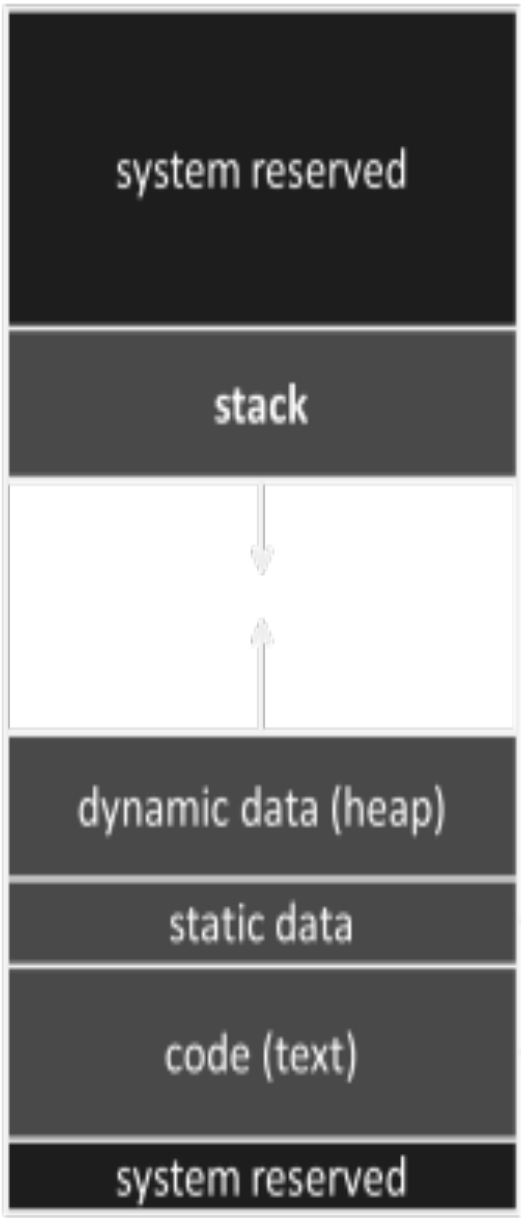


What is in Memory?

```
int n = 4;
int k[] = {3, 14, 0, 10};

int fib(int i) {
   if (i <= 2) return i;
   else return fib(i-1)+fib(i-2);
}

int main(int ac, char **av) {
   for (int i = 0; i < n; i++) {
      printi(fib(k[i]));
      prints("\n");
   }
}
```



Problem

Main memory is very very very slow!

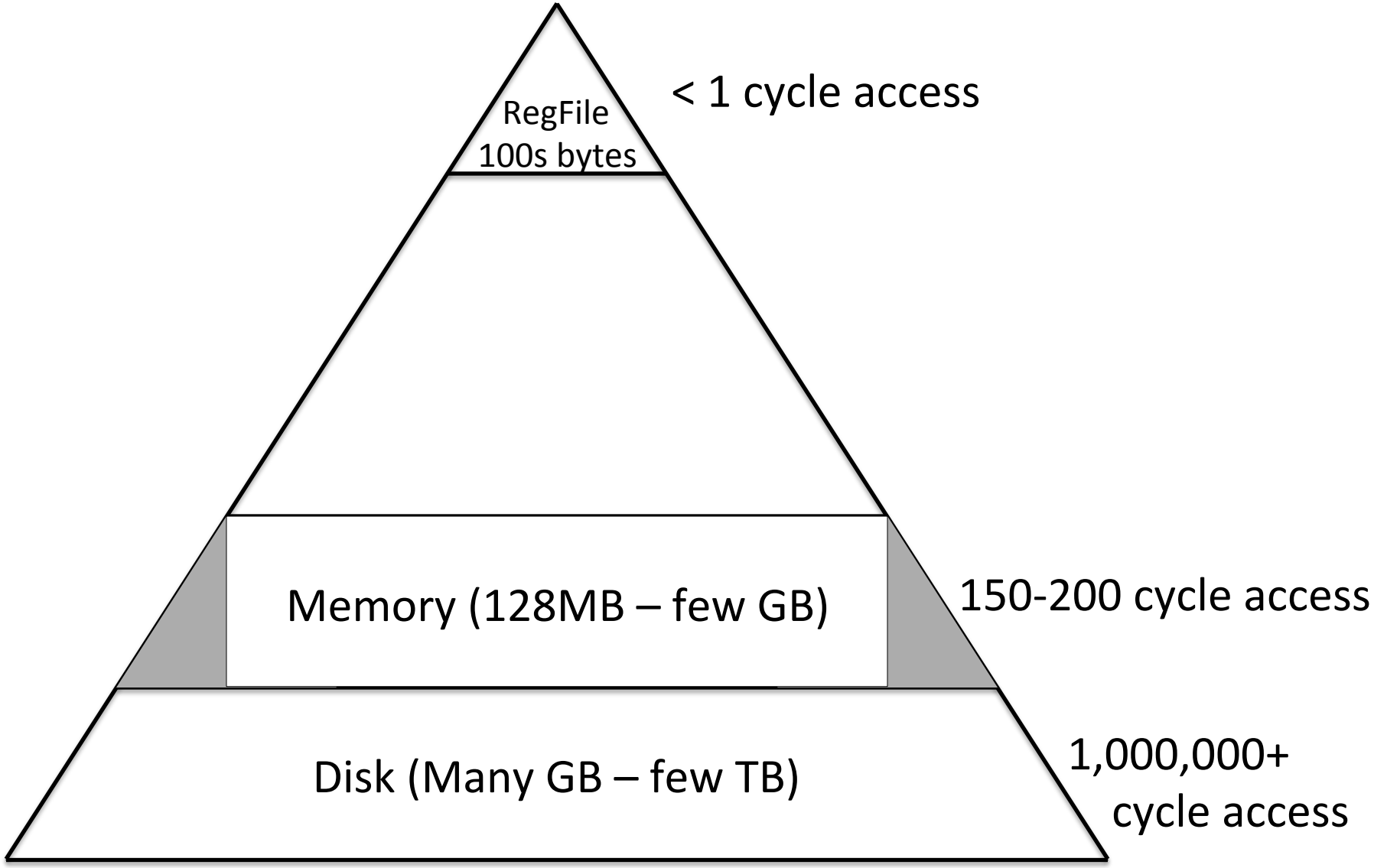
- **DRAM**
  - 1 transistor, cheaper/bit, slower, needs refresh
- **SRAM**
  - 6-8 transistors, costlier/bit, faster, no refresh
- **D-Flip Flop, Registers**
  - 10-100s of transistors, very expensive, very fast

Problem

CPU clock rates ~0.33ns – 2ns (3GHz-500MHz)

Memory technology	**Access time in nanosecs (ns)**	**Access time in cycles**	**$ per GIB in 2012**	**Capacity**
SRAM (on chip)	0.5-2.5 ns	1-3 cycles	$4k	256 KB
SRAM (off chip)	1.5-30 ns	5-15 cycles	$4k	32 MB
DRAM	50-70 ns	150-200 cycles	$10-$20	8 GB
SSD (Flash)	5k-50k ns	Tens of thousands	$0.75-$1	512 GB
Disk	5M-20M ns	Millions	$0.05-$0.1	4 TB

Memory Hierarchy



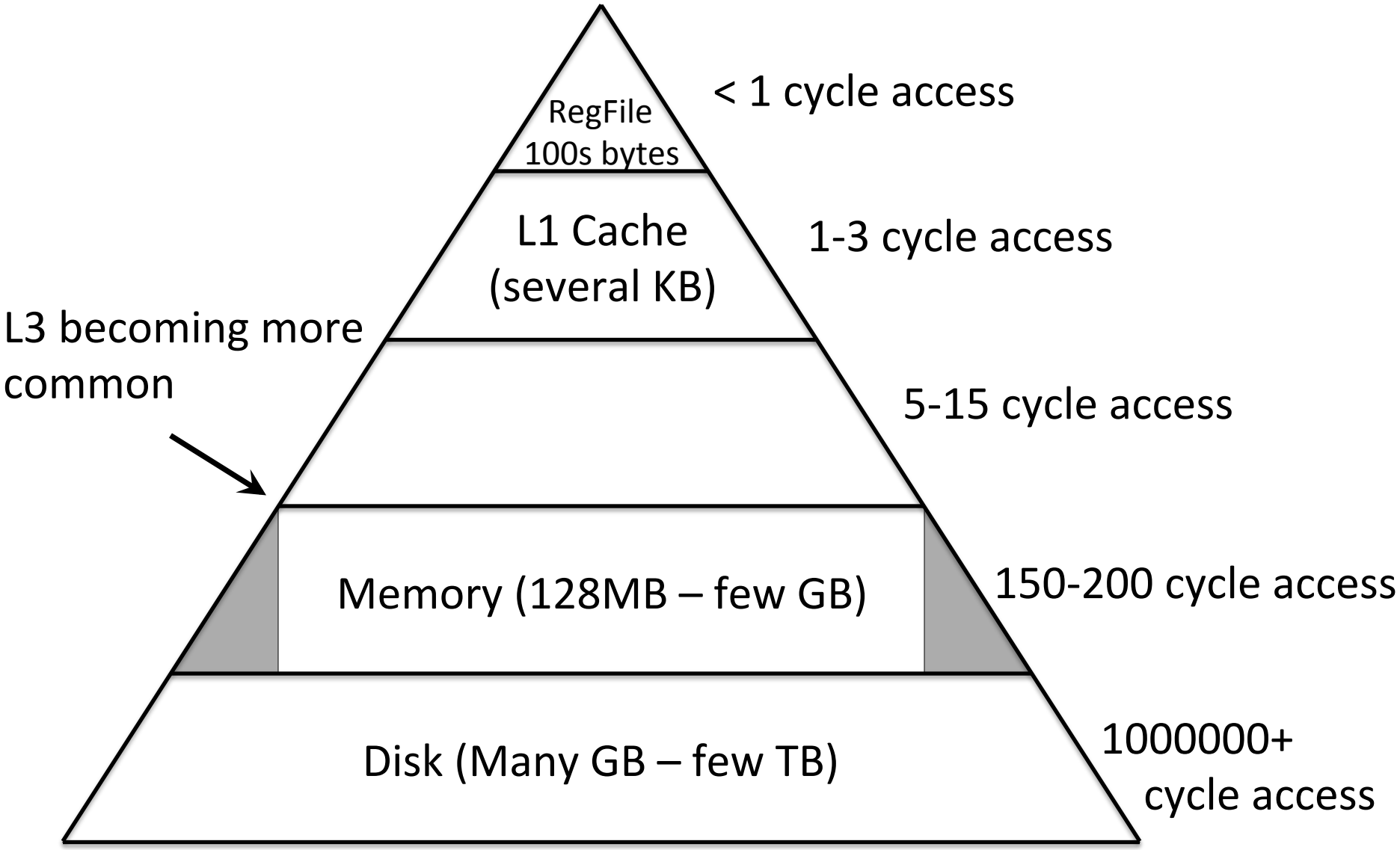
***These are rough numbers, mileage may vary.**

Goal

- memory?

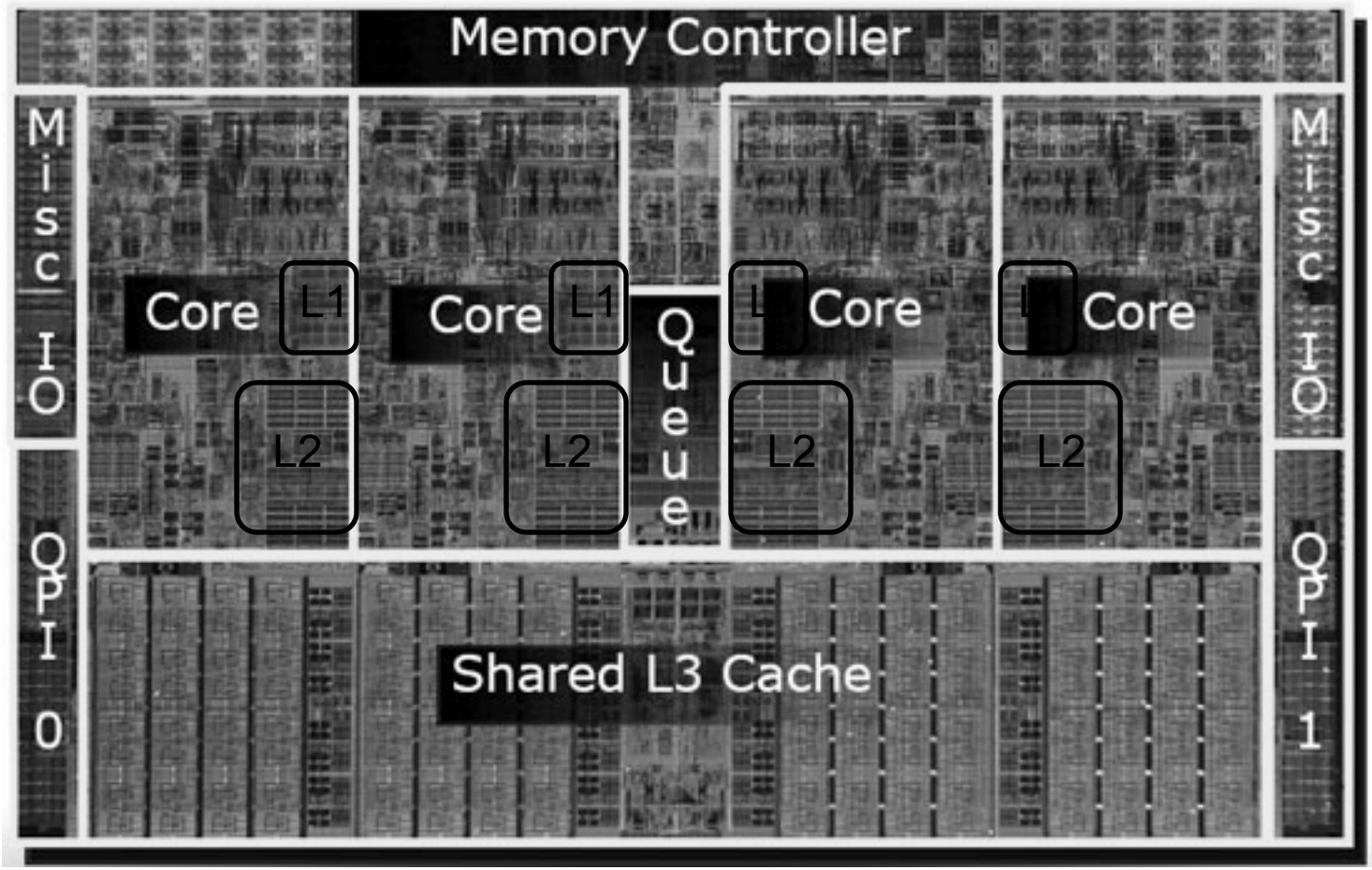
- YES! Caches!
  - Using fast technology (SRAM)
  - Being smart about access and placement of data
    - Temporal locality
    - Spatial locality

Memory Hierarchy



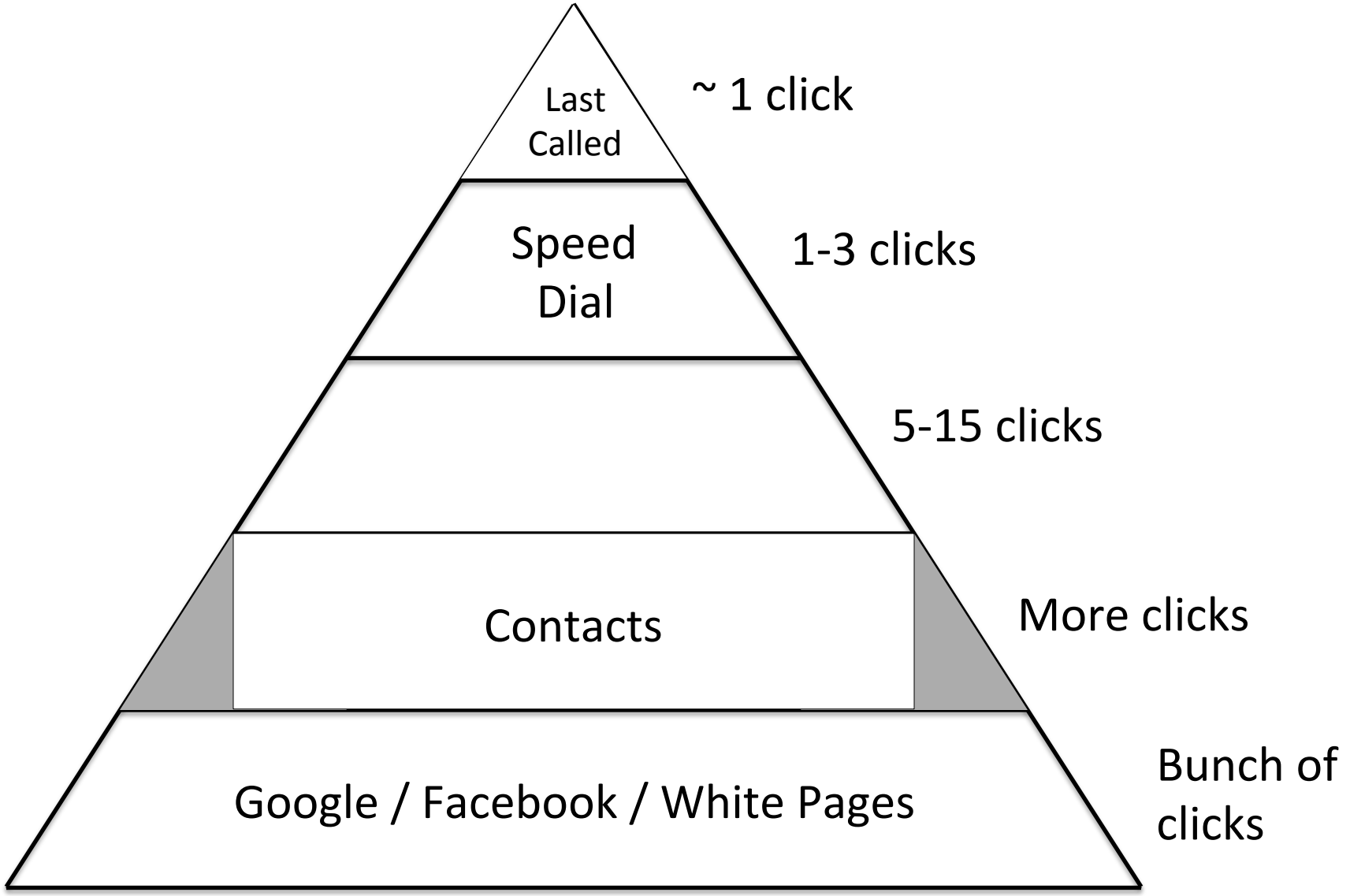
***These are rough numbers, mileage may vary.**

Caches on Today's CPUs



This Core i7 CPU has 4 cores, a memory controller and the QuickPath Interconnect (QPI at bottom left and right) on a single chip (die).

Just like your Contacts!



*** Will refer to this analogy using BLUE in the slides.**

Memory Hierarchy

- Storage (memory/DRAM) farther from processor
  - big & slow
  - stores inactive data
  - contacts

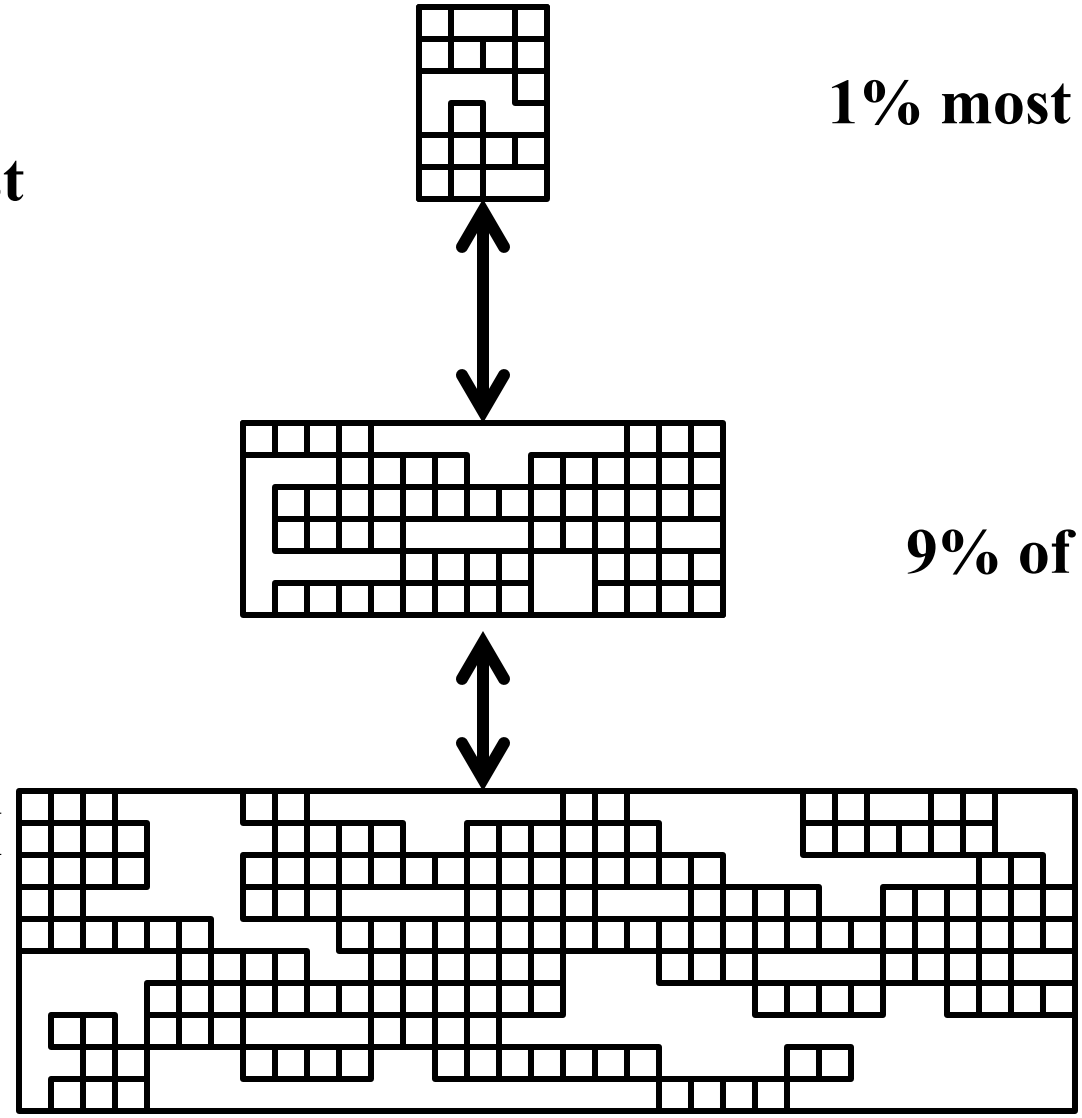
- Storage (cache/SRAM) closer to processor
  - small & fast
  - stores active data
  - speed dial & favorites

Memory Hierarchy

L1 Cache
SRAM-on-chip
1% of data most accessed

L2/L3 Cache
SRAM
9% of data is "active"

Memory DRAM
90% of data inactive (not accessed)



Speed Dial
1% most called people

Favorites
9% of people called

Contacts
90% people rarely called

Insight for Caches

If Mem[x] was accessed *recently*...

… then Mem[x] is likely to be accessed *soon*

- temporal locality:
  - Mem[x] higher in memory hierarchy since it will likely be accessed again soon
  - If you call a contact, you are more likely to call them again soon

… then Mem[x ± ε] is likely to be accessed *soon*

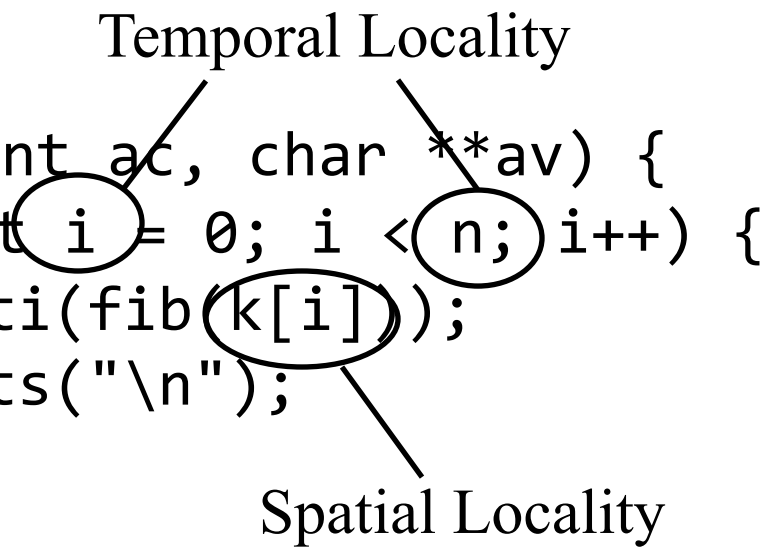
- Exploit spatial locality:
  - Put entire block containing Mem[x] and surrounding addresses higher in memory hierarchy since nearby address will likely be accessed
  - If you call a member of a family, you are more likely to call other members

What is in Memory?

```
int n = 4;
int k[] = {3, 14, 0, 10};

int fib(int i) {
   if (i <= 2) return i;
   else return fib(i-1)+fib(i-2);
}

int main(int ac, char **av) {
   for (int i = 0; i < n; i++) {
      printi(fib(k[i]));
      prints("\n");
   }
}
```



Cache Lookups (Read)

Processor tries to access Mem[x]

You try to speed dial “Smith, Paul”

Check: is block containing Mem[x] in the cache?

Check: is Smith in your Speed Dial?

Yes: cache hit!

- return requested data from cache line!
- call Smith, P.!

Cache Lookups (Read)

Processor tries to access Mem[x]

You try to speed dial “Smith, Paul”

Check: is block containing Mem[x] in the cache?

Check: is Smith in your Speed Dial?

No: cache miss

- read block from memory (or lower level cache)
- find Smith, P. in Contacts (or Favorites)
- (evict an existing cache line to make room)
- (delete a speed dial number to make room for Smith, P.)
- place new block in cache
- store Smith, P. in speed dial
- return requested data
- call Smith, P.!

→ and stall the pipeline while all of this happens

Definitions and Terms

- Block (or cacheline or line)
  - Minimum unit of information that is present/or not in the cache
  - Minimum # of contact numbers that is either present or not
- Cache hit
  - Contact in Speed Dial
- Cache miss
  - Contact not in Speed Dial
- Hit rate
  - The fraction of memory accesses found in a level of the memory hierarchy
  - The fraction of contacts found in Speed Dial
- Miss rate
  - The fraction of memory accesses not found in a level of the memory hierarchy
  - The fraction of contacts not found in Speed Dial

Cache Questions

- What structure to use?
  - Where to place a block?
  - How to find a block?
- When miss, which block to replace?
- What happens on write?
  - Next lecture

Three common designs

A given data block can be placed…

- … in exactly one cache line → Direct Mapped
- … in any cache line → Fully Associative
- … in a small set of cache lines → Set Associative

Direct Mapped Cache

- Each contact can be mapped to a single digit

Contacts

Baker, J.	111-111-1111
Baker, S.	222-222-2222
Dunn, A.	333-333-3333
Foster, D.	444-444-4444
Gill, D.	555-555-5555
Harris, F.	666-666-6666
Jones, N.	777-777-7777
Lee, V.	888-888-8888
Mann, B.	111-111-1119
Moore, F.	222-222-2229
Powell, C.	333-333-3339
Sam, J.	444-444-4449
Taylor, B.	555-555-5559
Taylor, O.	666-666-6669
Wright, T.	777-777-7779
Zhang, W.	888-888-8889

Direct Mapped Cache

- Each contact can be mapped to a single digit

Baker, J.

↓

ABC ➔ 2

↓

2 ➔ 111-111-1111

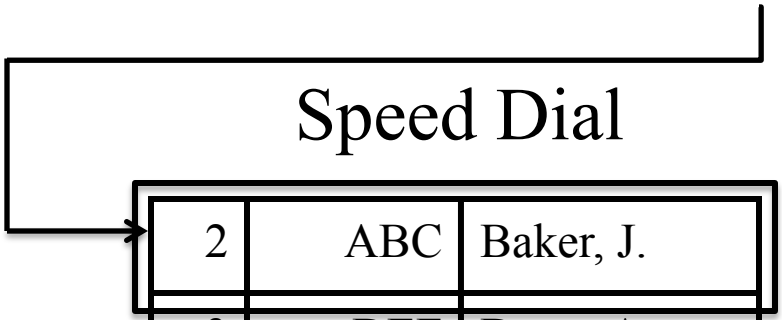
Your brain is doing indexing!

Contacts

Baker, J.	111-111-1111
Baker, S.	222-222-2222
Dunn, A.	333-333-3333
Foster, D.	444-444-4444
Gill, D.	555-555-5555
Harris, F.	666-666-6666
Jones, N.	777-777-7777
Lee, V.	888-888-8888
Mann, B.	111-111-1119
Moore, F.	222-222-2229
Powell, C.	333-333-3339
Sam, J.	444-444-4449
Taylor, B.	555-555-5559
Taylor, O.	666-666-6669
Wright, T.	777-777-7779
Zhang, W.	888-888-8889

Direct Mapped Cache

- Use the first letter to index!



Speed Dial

2	ABC	Baker, J.
3	DEF	Dunn, A.
4	GHI	Gill, S.
5	JKL	Jones, N.
6	MNO	Mann, B.
7	PQRS	Powell, J.
8	TUV	Taylor, B.
9	WXYZ	Wright, T.

Contacts

Baker, J.	111-111-1111
Baker, S.	222-222-2222
Dunn, A.	333-333-3333
Foster, D.	444-444-4444
Gill, D.	555-555-5555
Harris, F.	666-666-6666
Jones, N.	777-777-7777
Lee, V.	888-888-8888
Mann, B.	111-111-1119
Moore, F.	222-222-2229
Powell, C.	333-333-3339
Sam, J.	444-444-4449
Taylor, B.	555-555-5559
Taylor, O.	666-666-6669
Wright, T.	777-777-7779
Zhang, W.	888-888-8889

Block Size: 1 Contact Number

Direct Mapped Cache

- Use the first letter to index!
- Use the initial to offset!

Speed Dial

2	ABC	Baker, J.	Baker, S.
3	DEF	Dunn, A.	Foster, D.
4	GHI	Gill, D.	Harris, F.
5	JKL	Jones, N.	Lee, V.
6	MNO	Mann, B.	Moore, F.
7	PQRS	Powell, C.	Sam, J.
8	TUV	Taylor, B.	Taylor, O.
9	WXYZ	Wright, T.	Zhang, W.

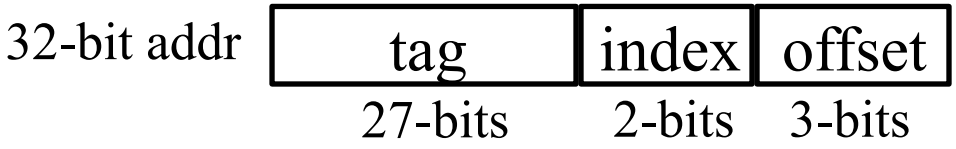
Block Size: 2 Contact Numbers

Contacts

Baker, J.	111-111-1111
Baker, S.	222-222-2222
Dunn, A.	333-333-3333
Foster, D.	444-444-4444
Gill, D.	555-555-5555
Harris, F.	666-666-6666
Jones, N.	777-777-7777
Lee, V.	888-888-8888
Mann, B.	111-111-1119
Moore, F.	222-222-2229
Powell, C.	333-333-3339
Sam, J.	444-444-4449
Taylor, B.	555-555-5559
Taylor, O.	666-666-6669
Wright, T.	777-777-7779
Zhang, W.	888-888-8889

Direct Mapped Cache

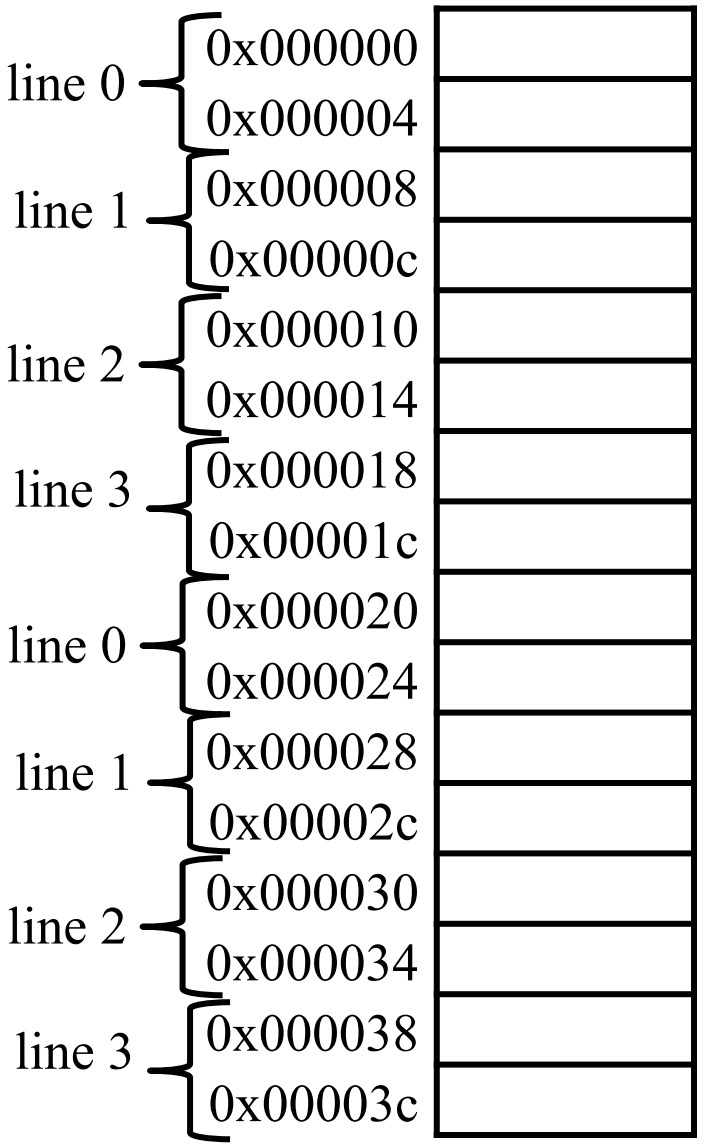
- Each block number mapped to a single cache line index
- Simplest hardware



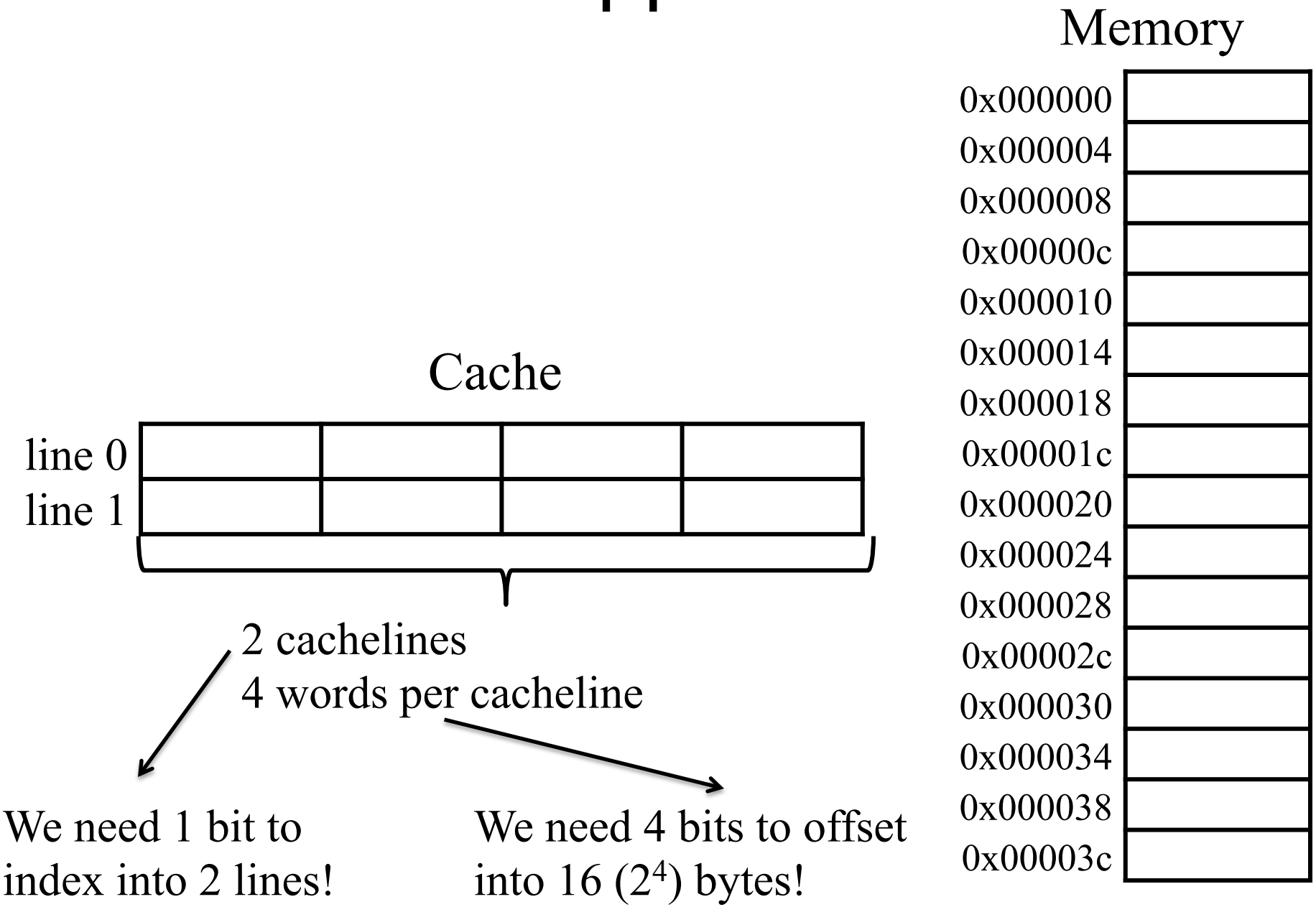
Cache

line 0	0x000000	0x000004
line 1		
line 2		
line 3		

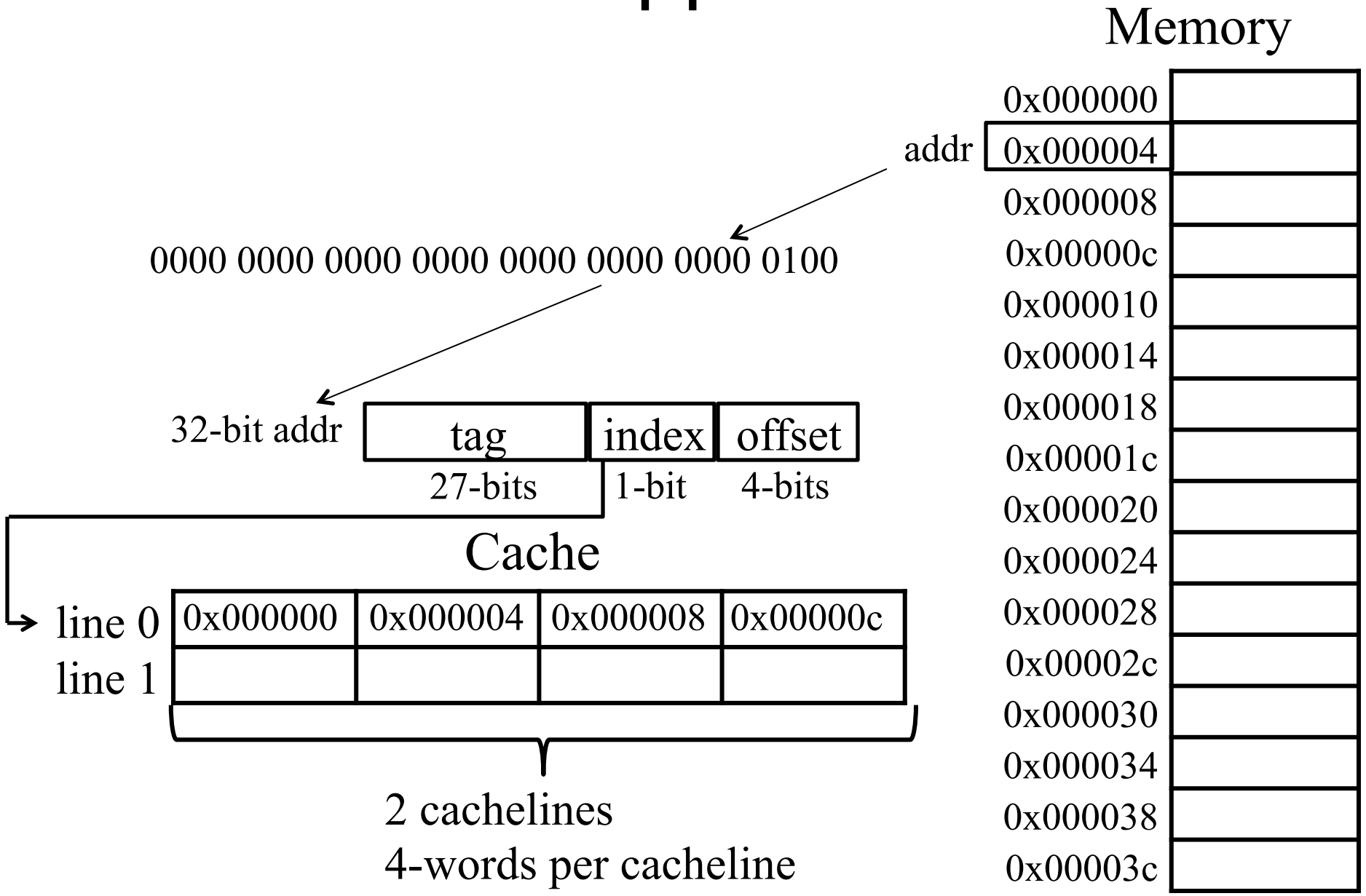
4 cachelines
2 words per cacheline



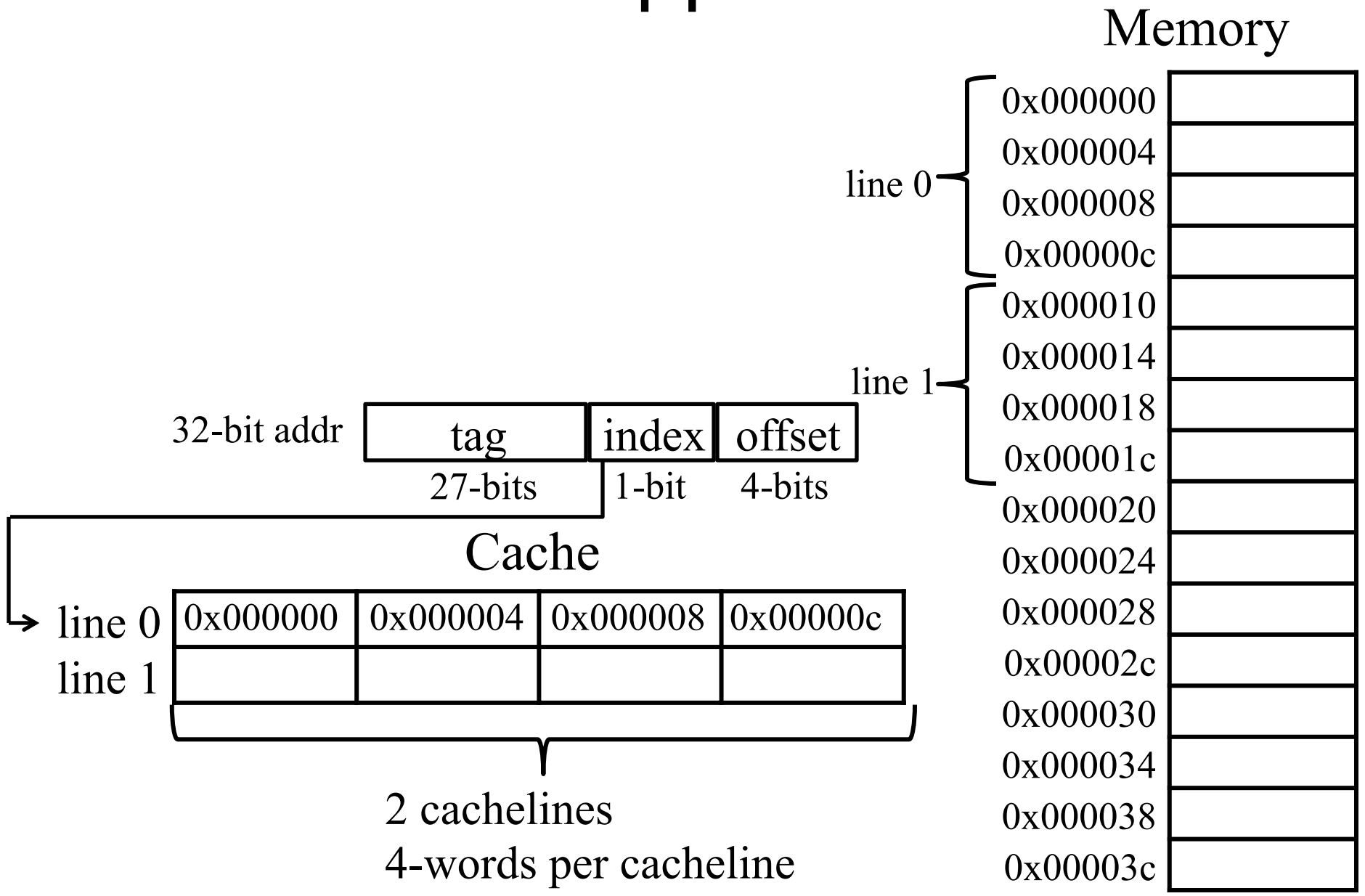
Direct Mapped Cache



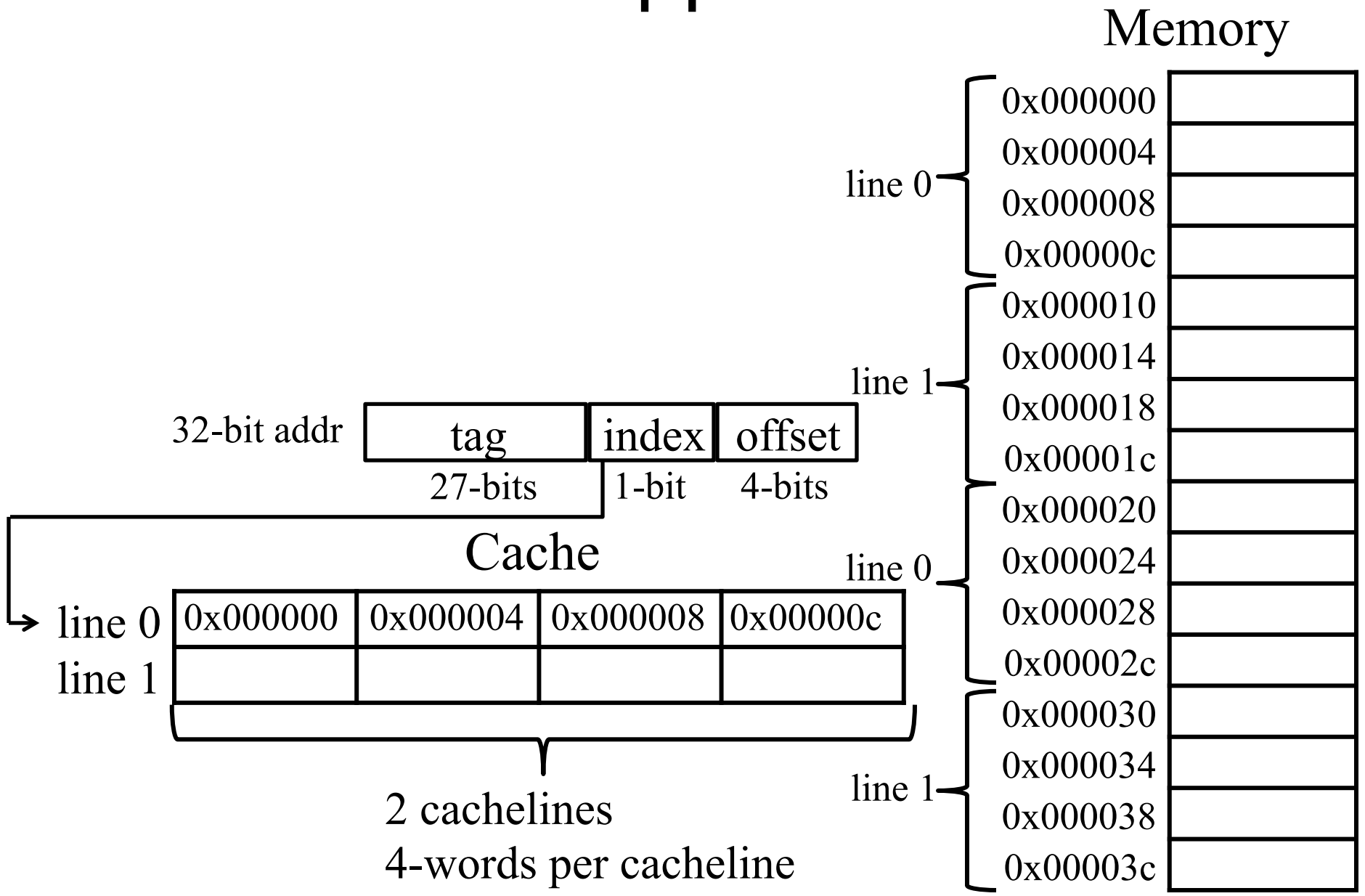
Direct Mapped Cache



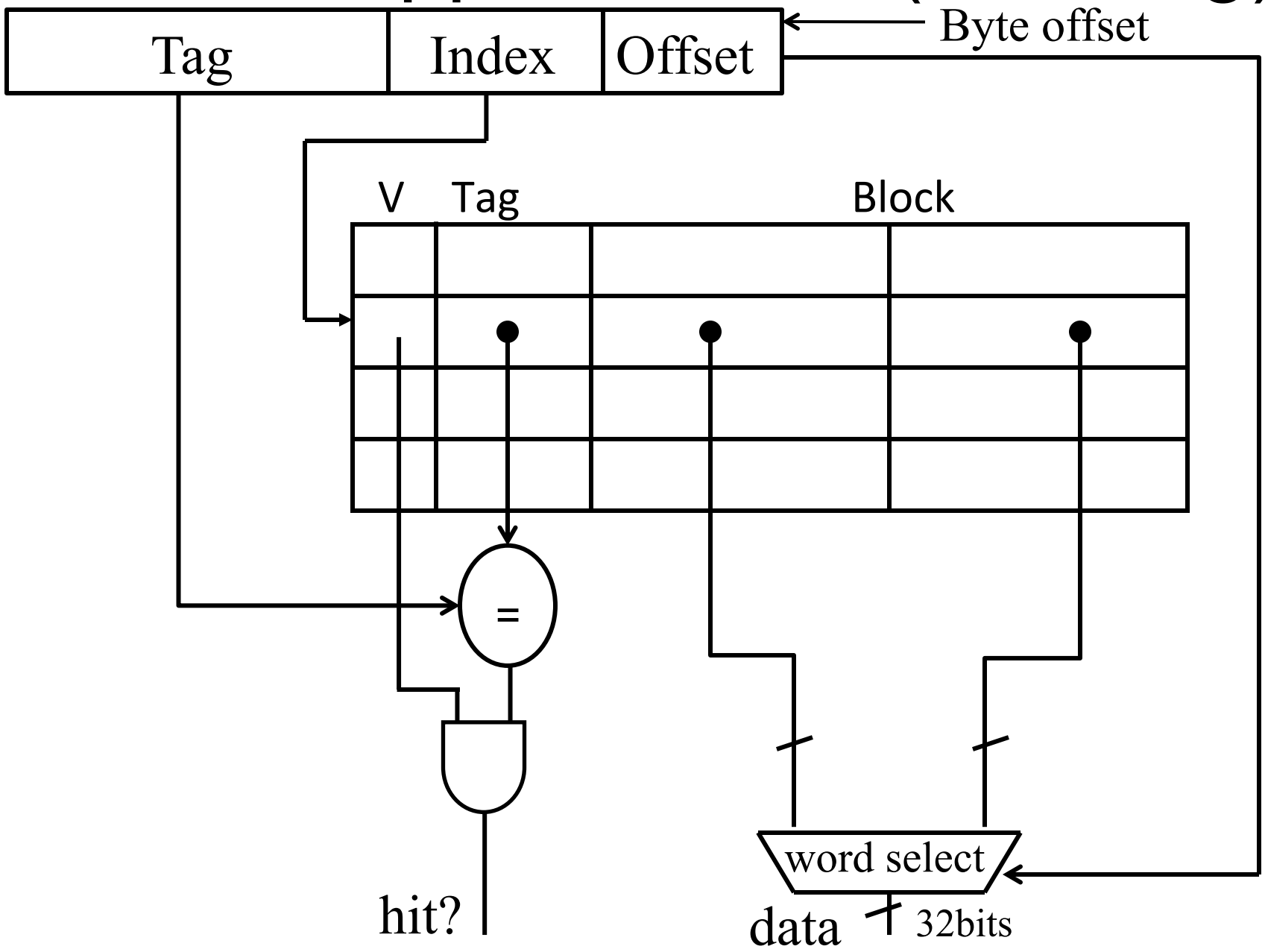
Direct Mapped Cache



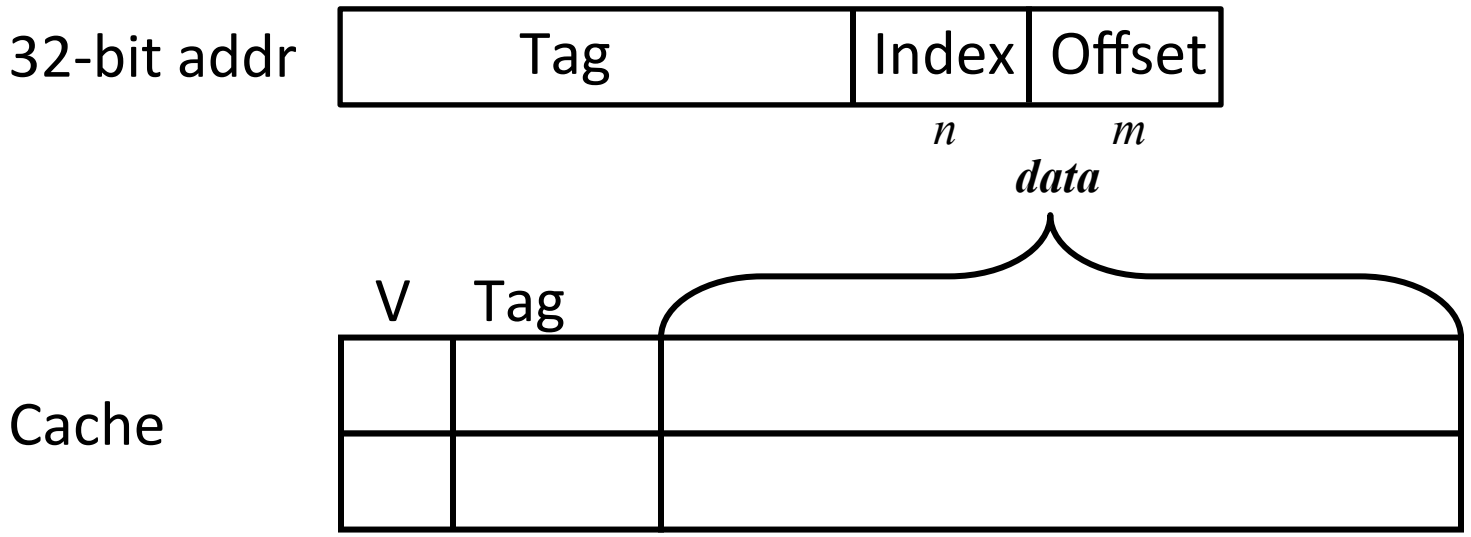
Direct Mapped Cache



Direct Mapped Cache (Reading)



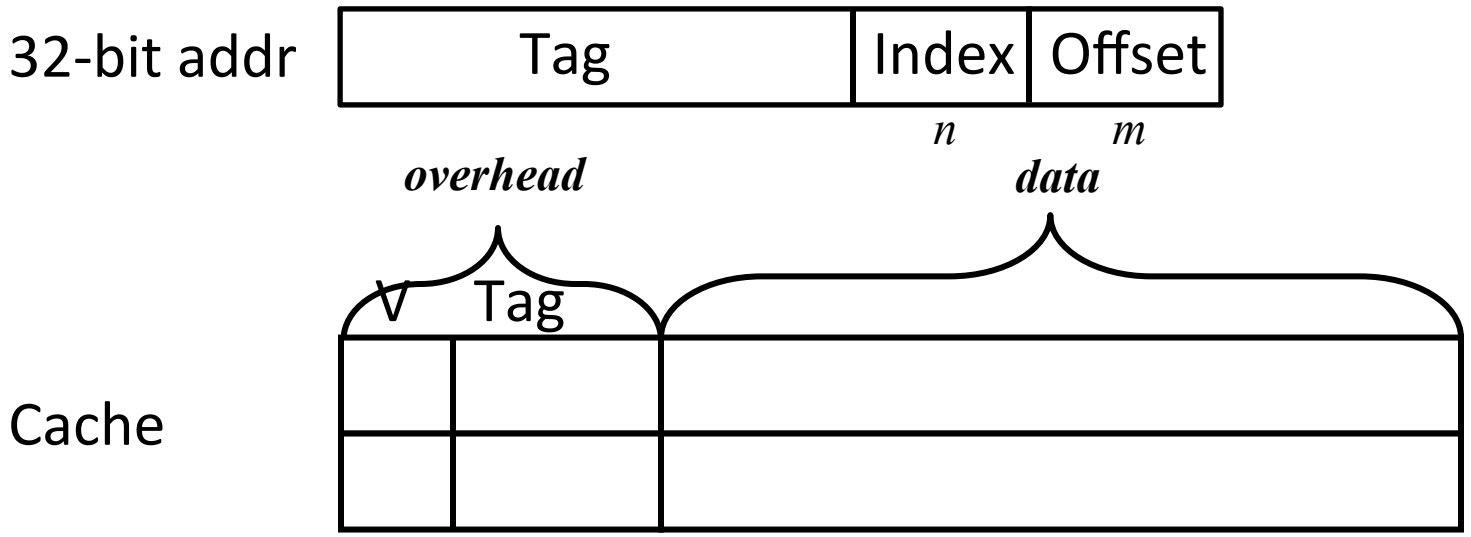
Direct Mapped Cache (Reading)



n bit index, *m* bit offset

Q: How big is the cache (***data only***)?

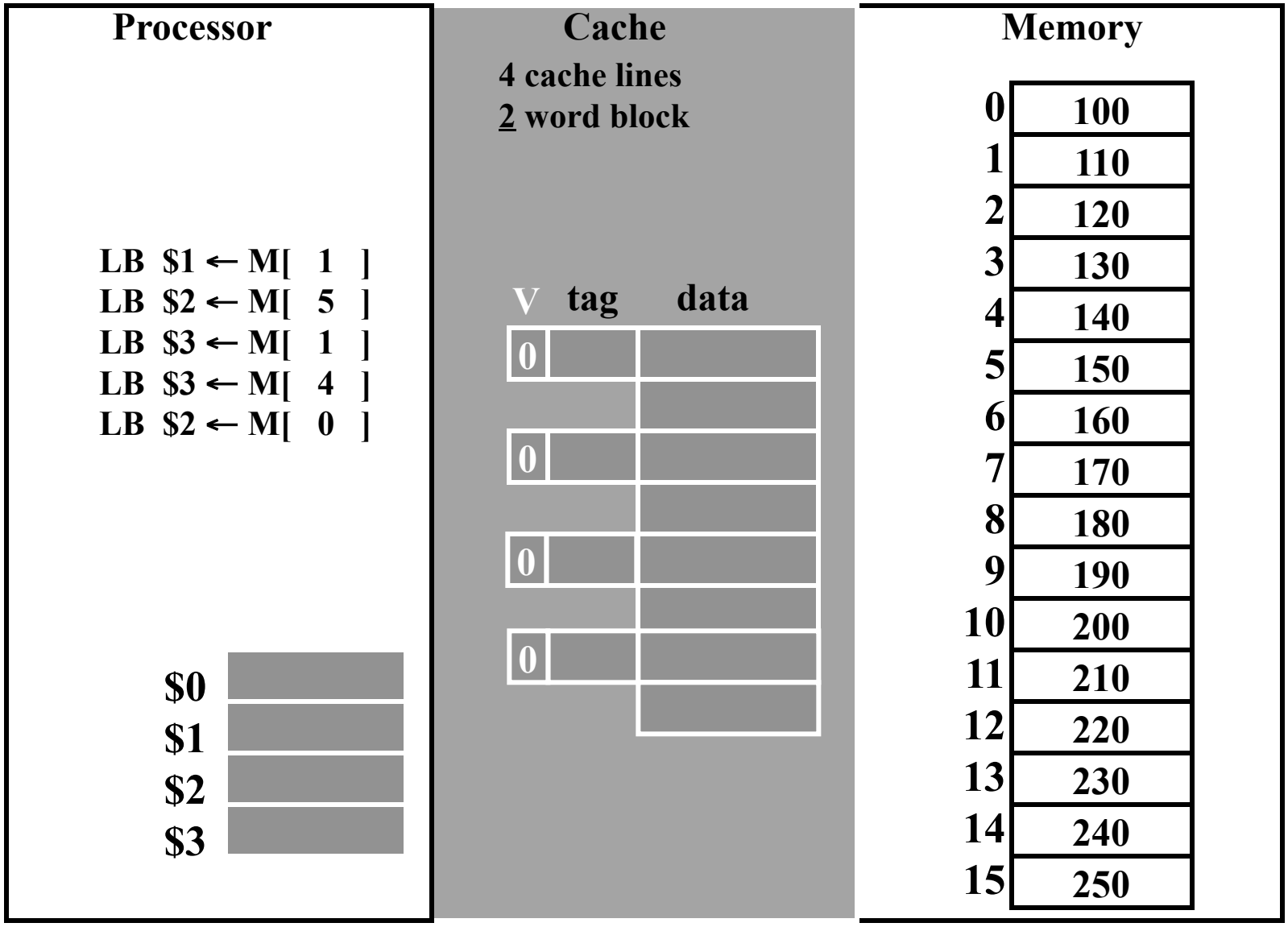
Direct Mapped Cache (Reading)



n bit index, *m* bit offset

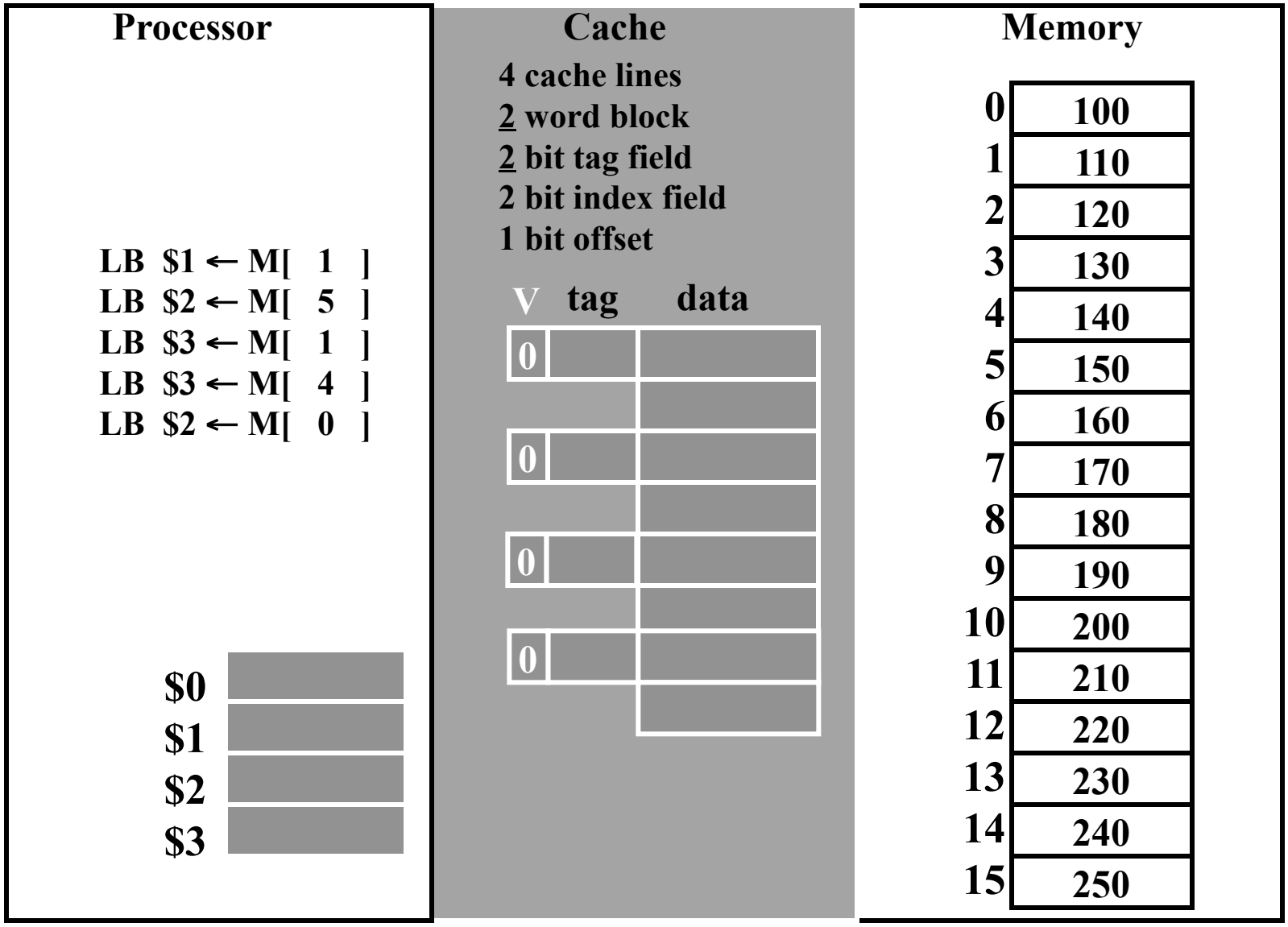
Q: How much SRAM is needed (***data*** + ***overhead***)?

Example: Direct Mapped Cache



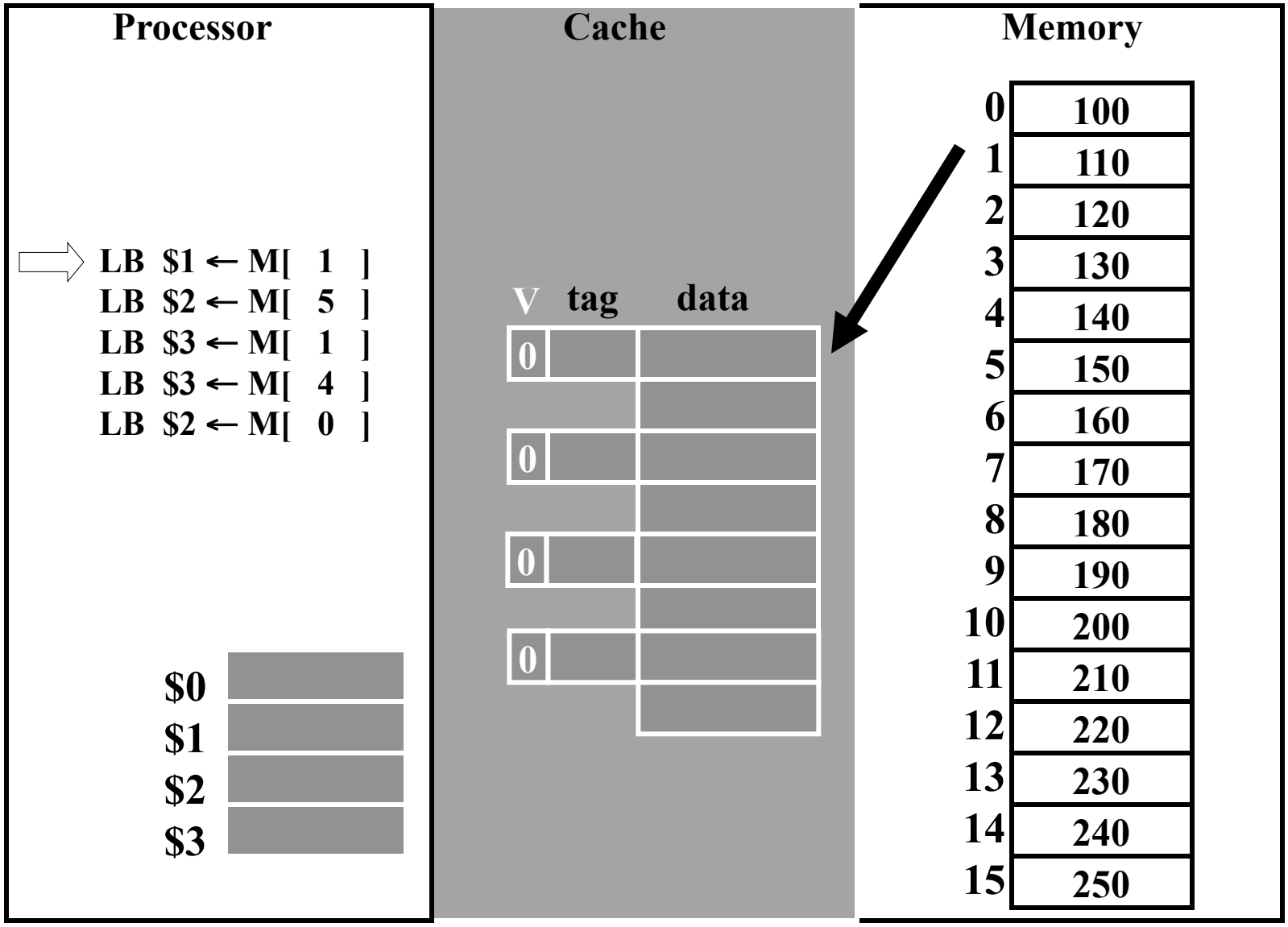
Using **byte addresses** in this example! Addr Bus = 5 bits

Example: Direct Mapped Cache



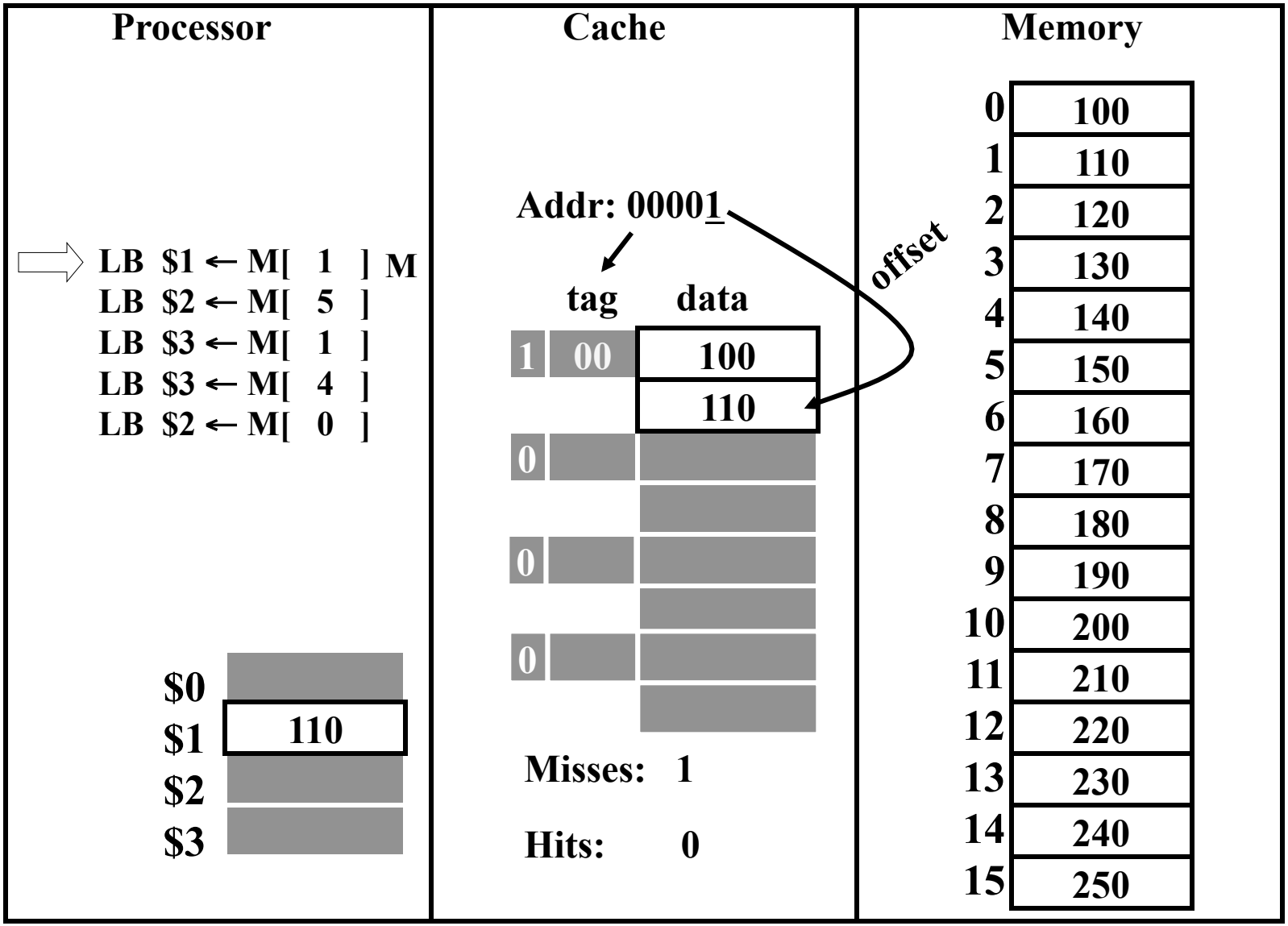
Using **byte addresses** in this example! Addr Bus = 5 bits

1st Access



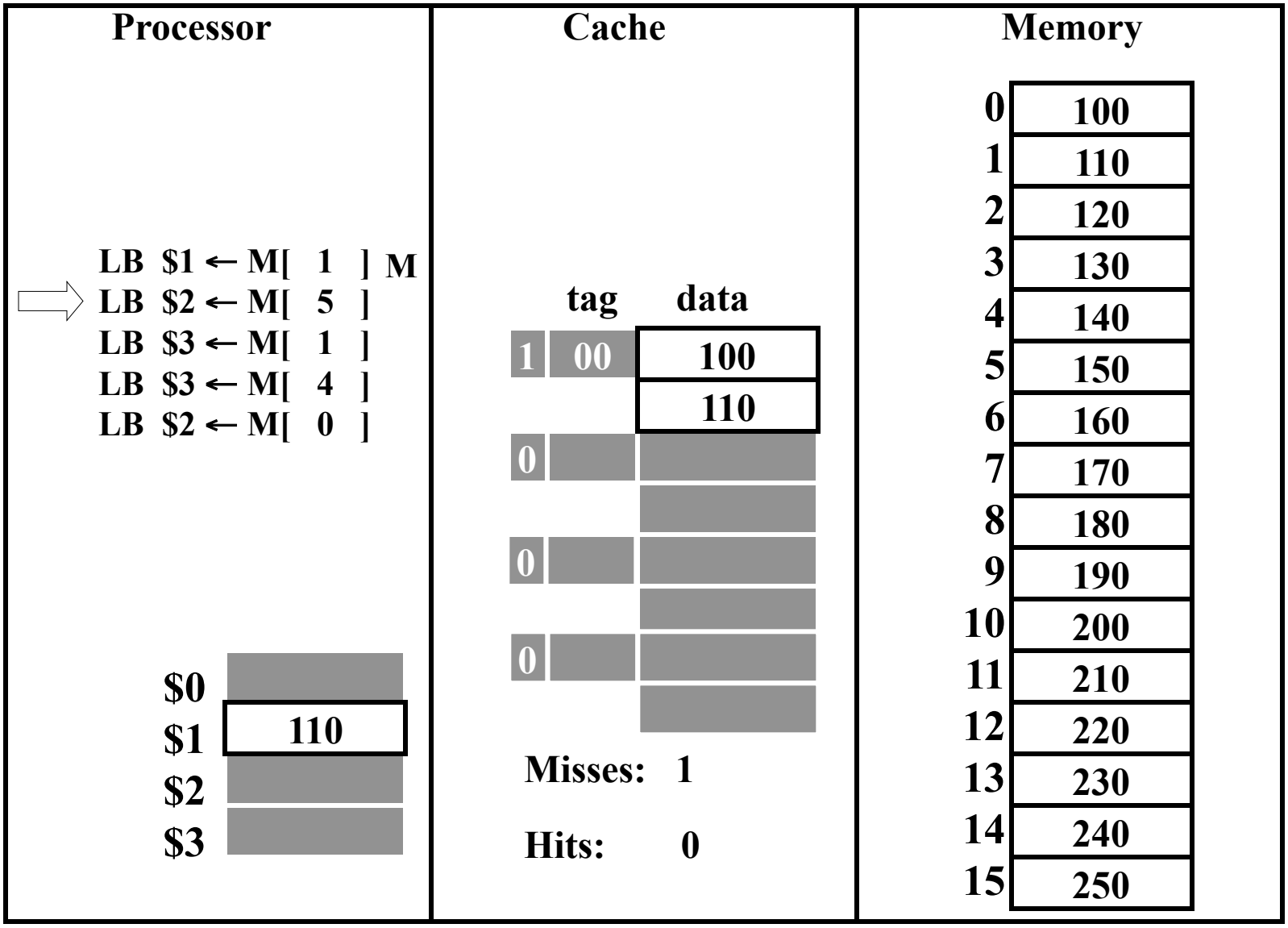
Using **byte addresses** in this example! Addr Bus = 5 bits

1st Access



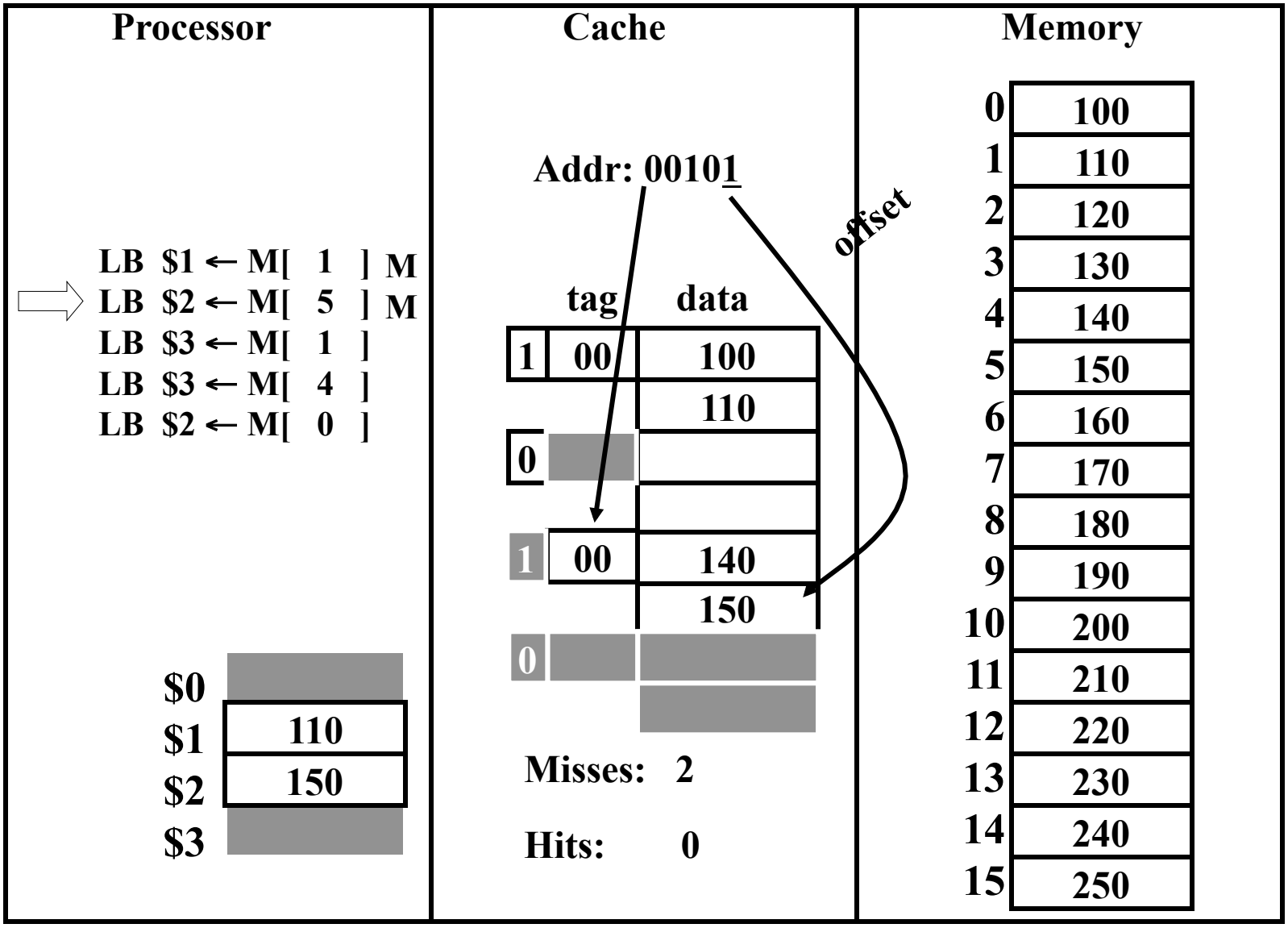
Using **byte addresses** in this example! Addr Bus = 5 bits

2nd Access



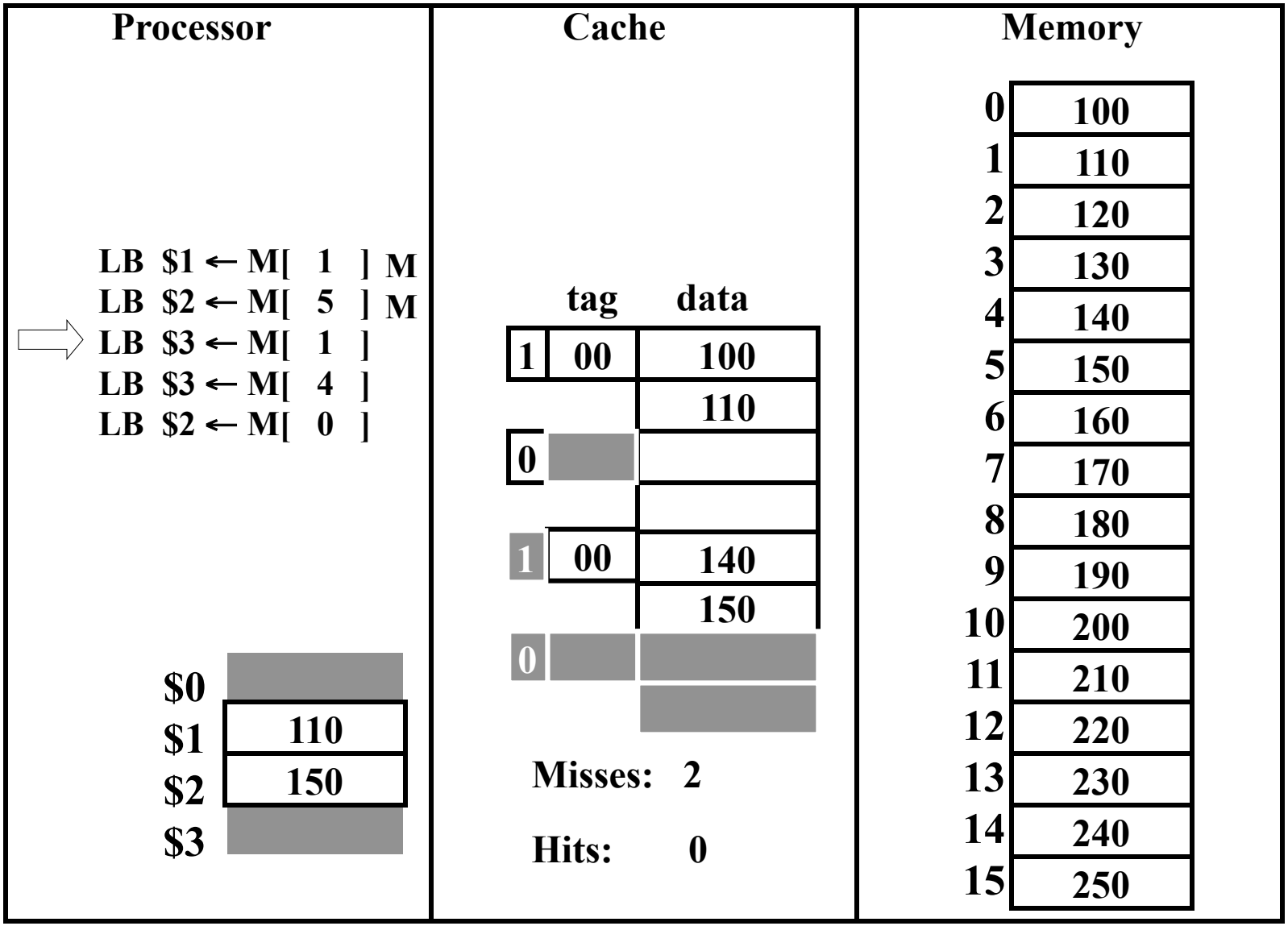
Using **byte addresses** in this example! Addr Bus = 5 bits

2nd Access



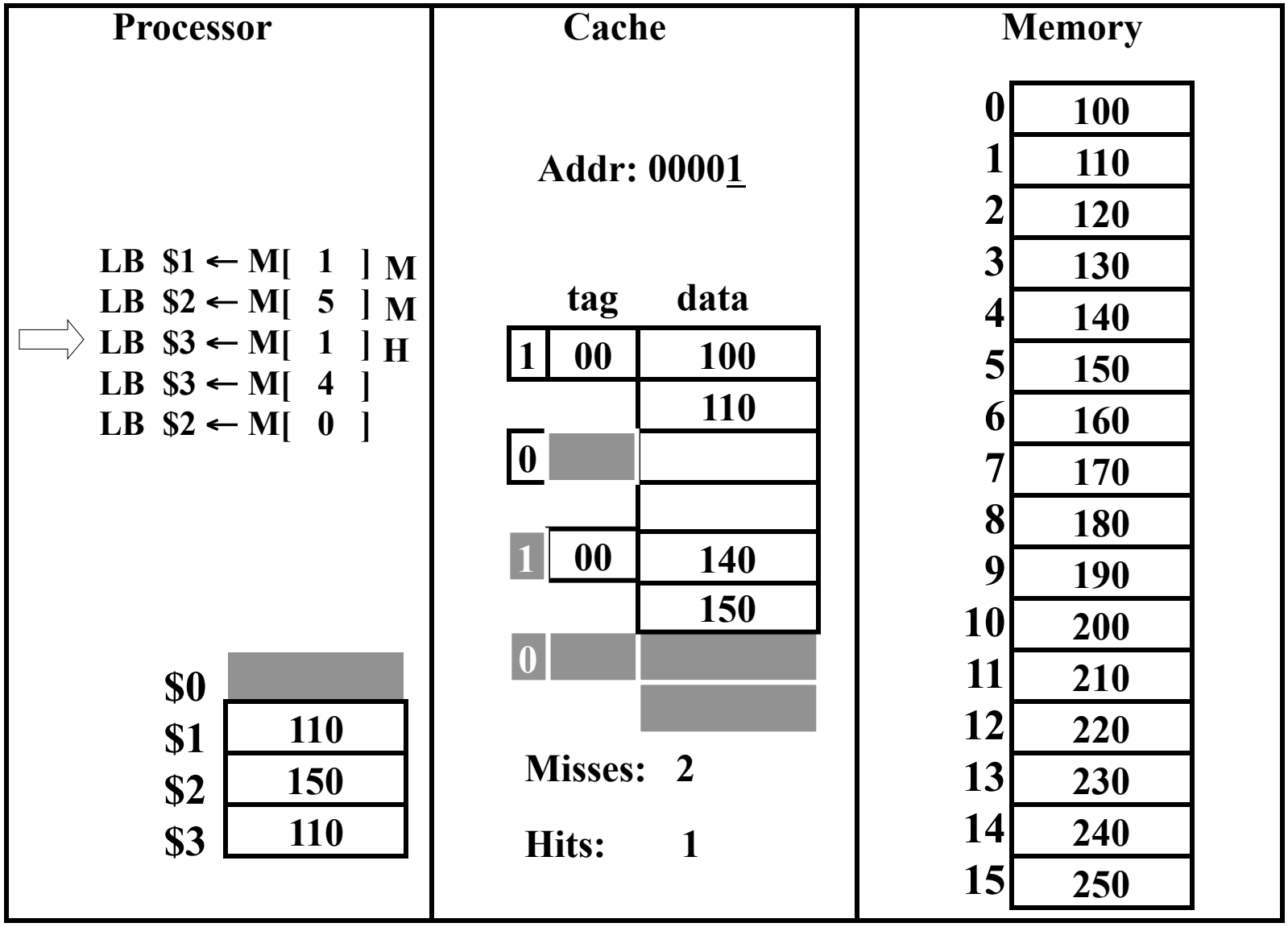
Using **byte addresses** in this example! Addr Bus = 5 bits

3rd Access



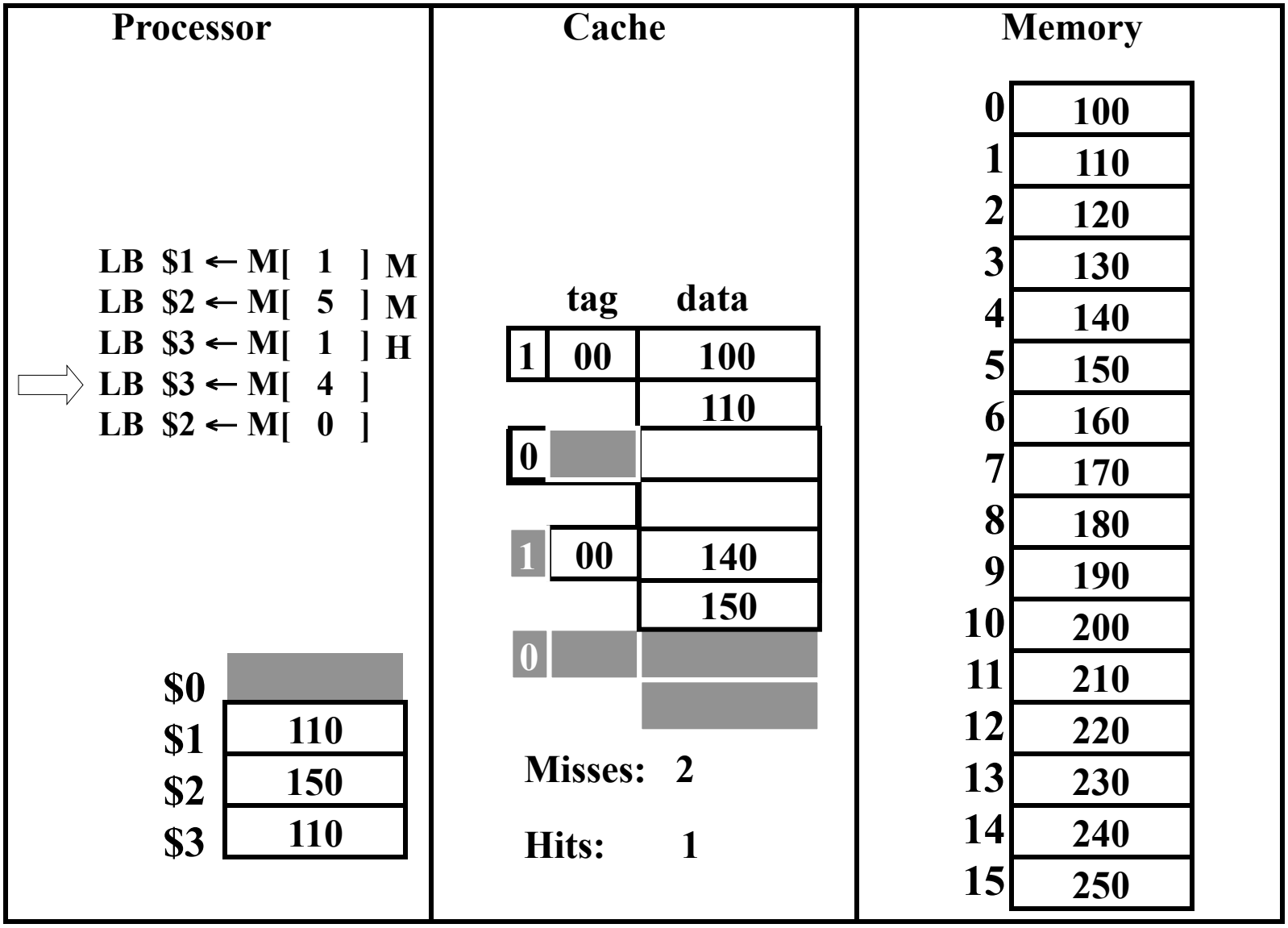
Using **byte addresses** in this example! Addr Bus = 5 bits

3rd Access



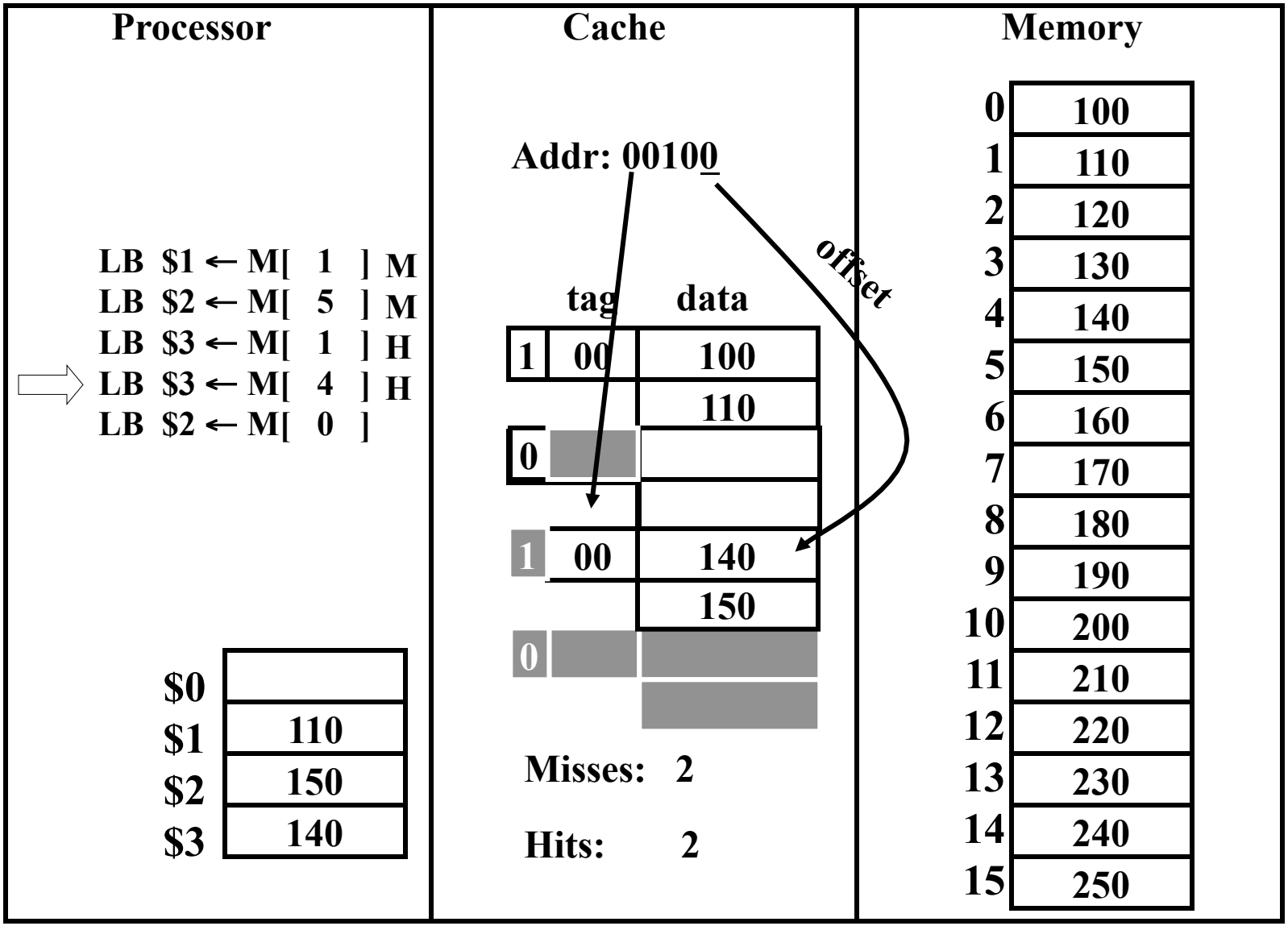
Using **byte addresses** in this example! Addr Bus = 5 bits

4th Access



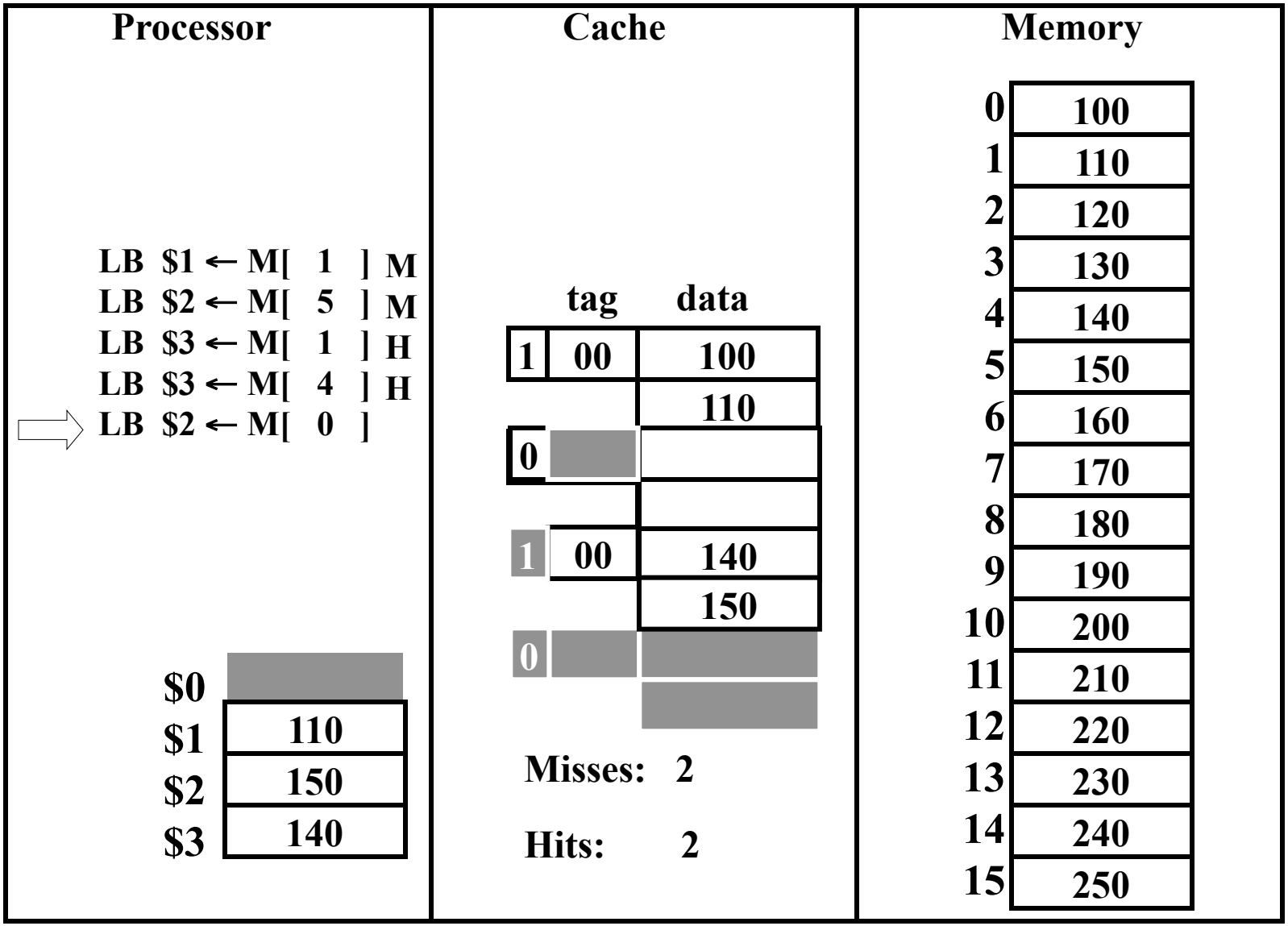
Using **byte addresses** in this example! Addr Bus = 5 bits

4th Access



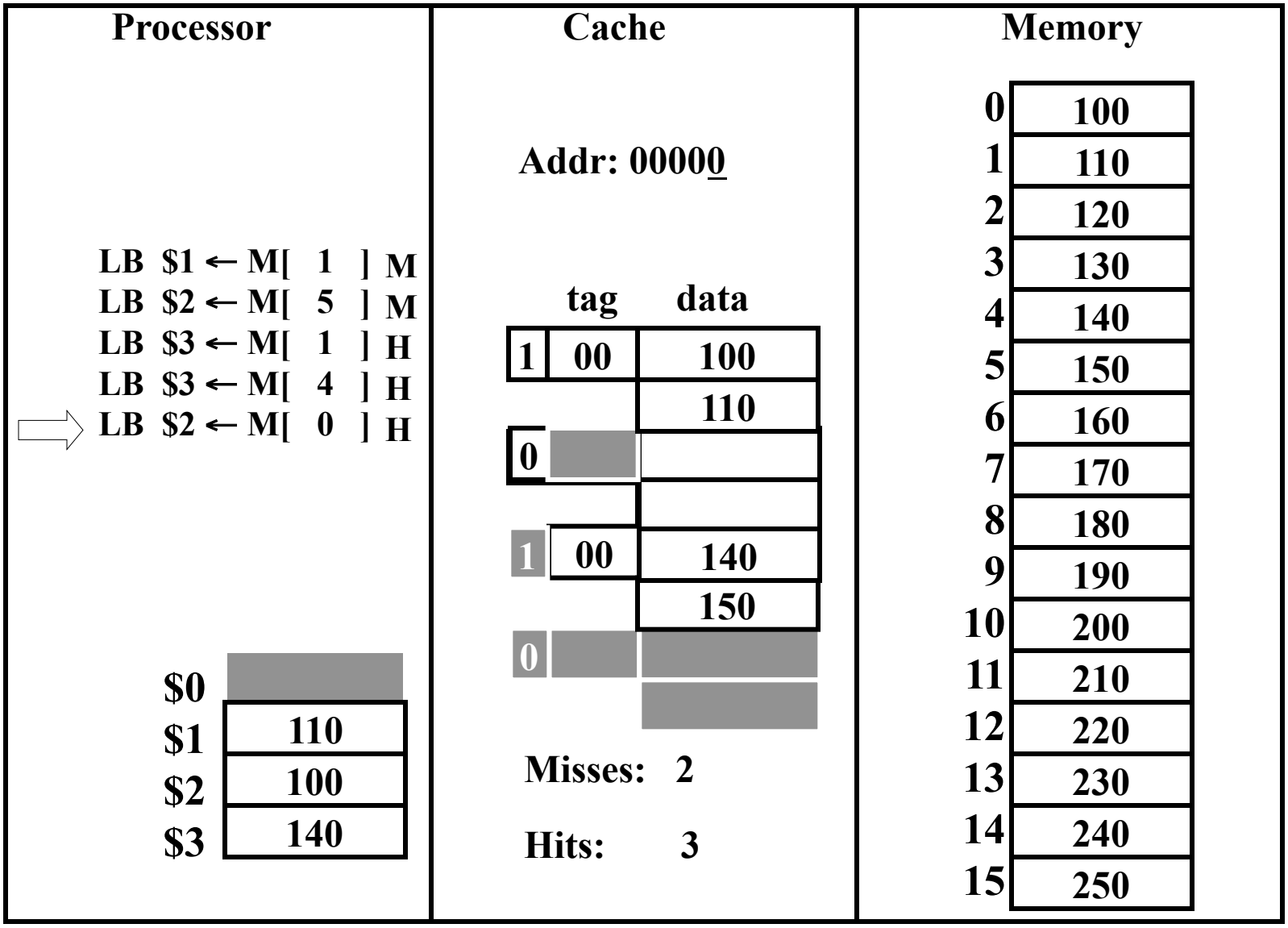
Using **byte addresses** in this example! Addr Bus = 5 bits

5th Access



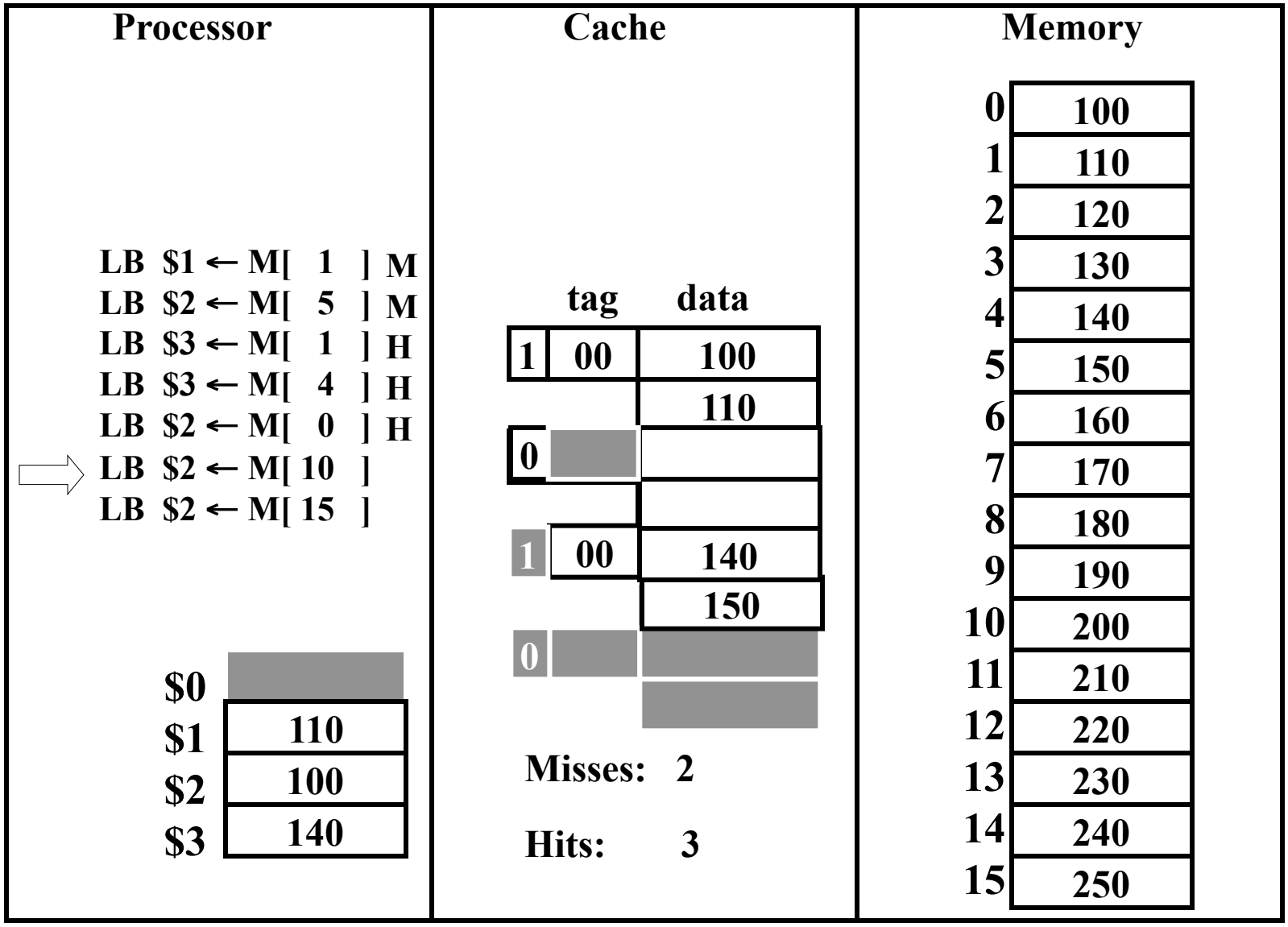
Using **byte addresses** in this example! Addr Bus = 5 bits

5th Access



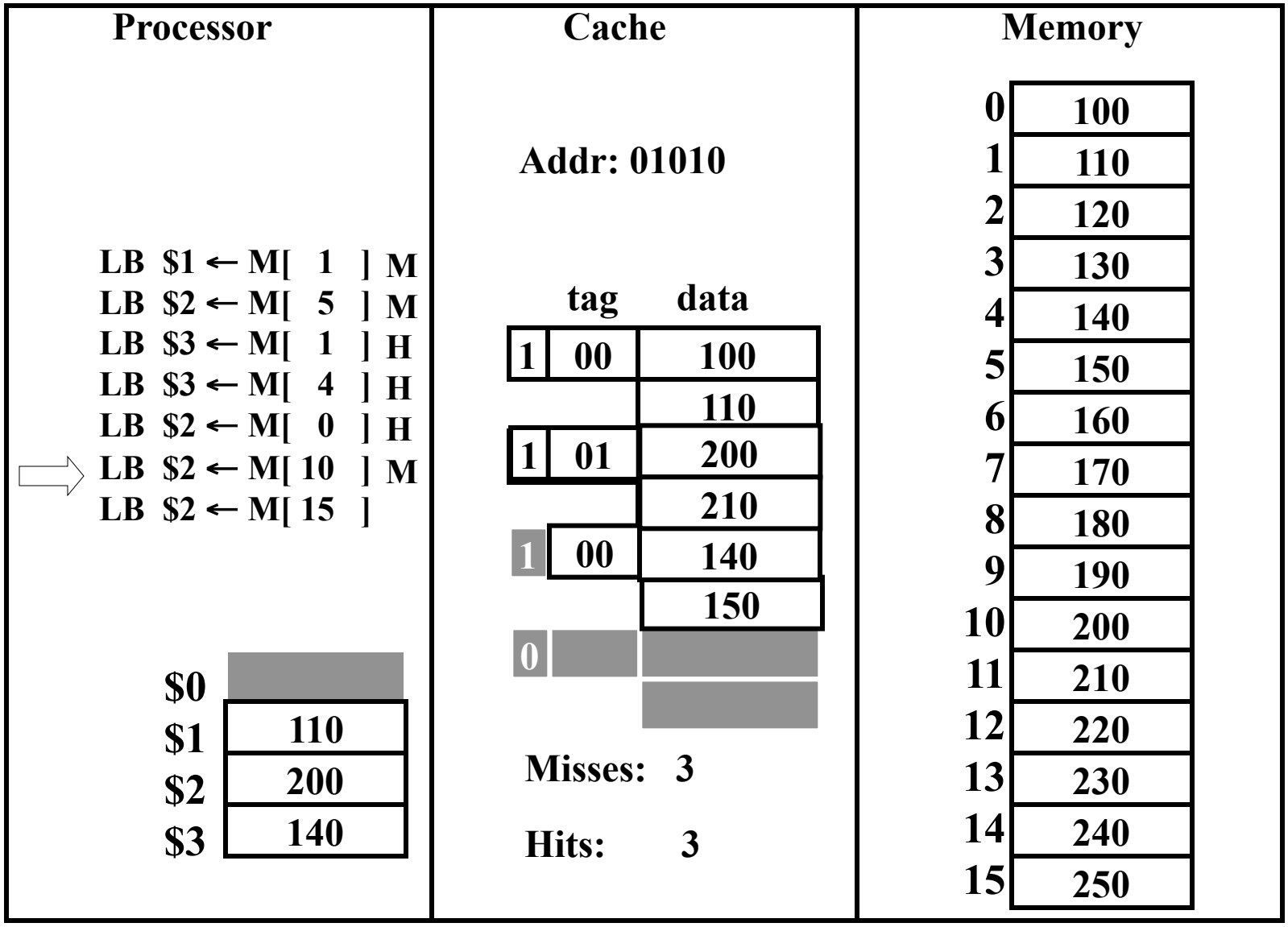
Using **byte addresses** in this example! Addr Bus = 5 bits

6th Access



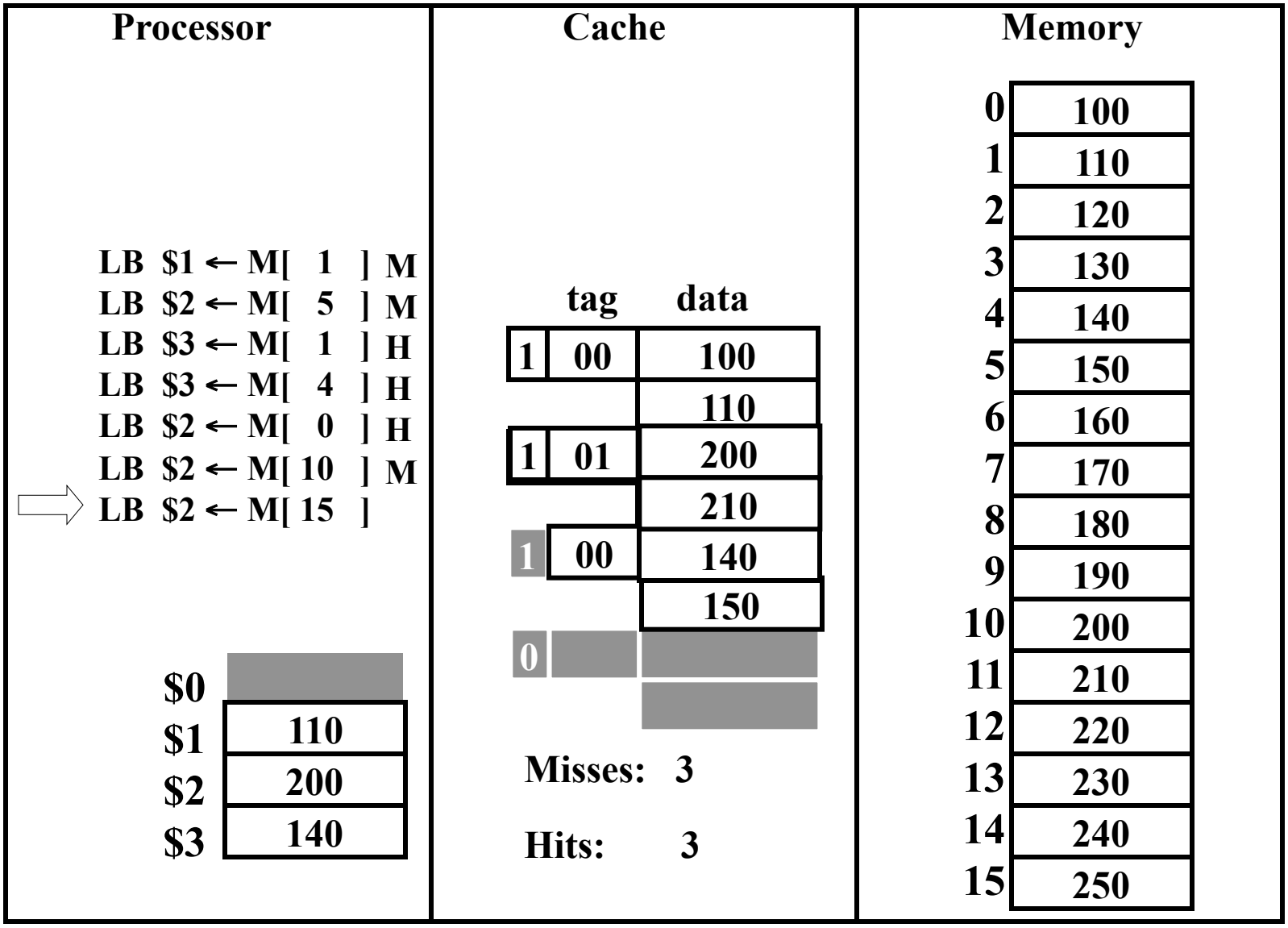
Using **byte addresses** in this example! Addr Bus = 5 bits

6th Access



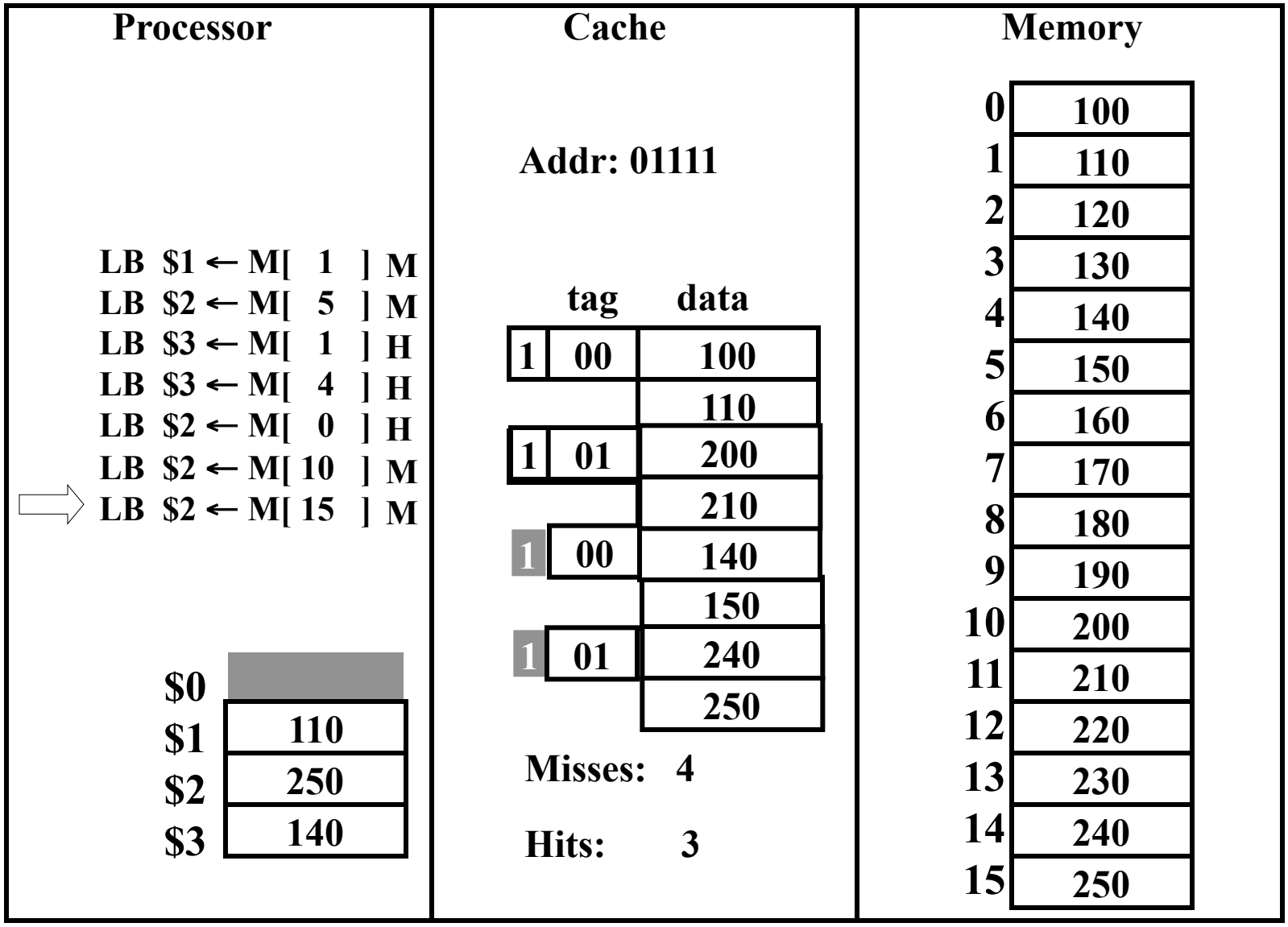
Using **byte addresses** in this example! Addr Bus = 5 bits

7th Access



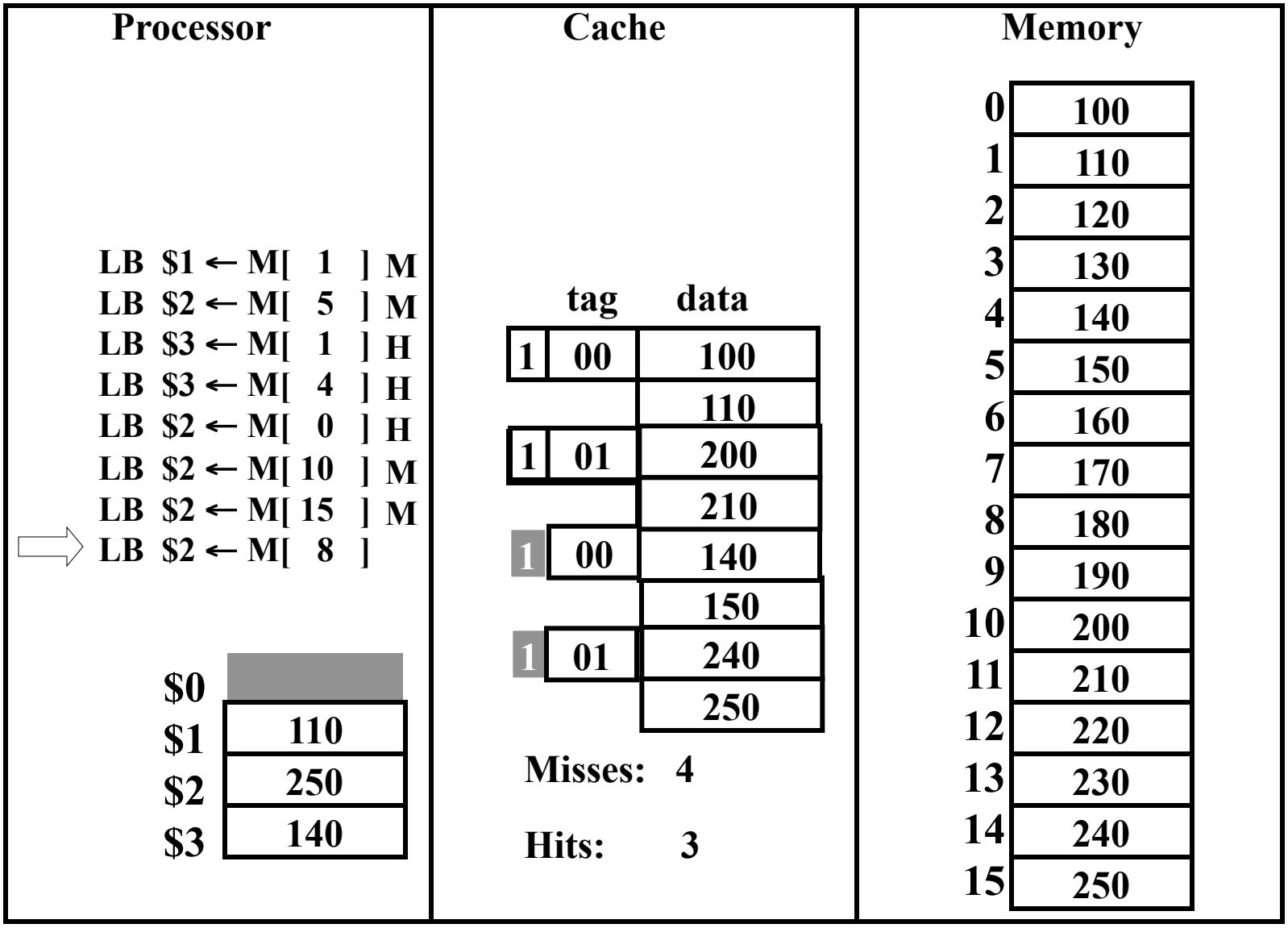
Using **byte addresses** in this example! Addr Bus = 5 bits

7th Access



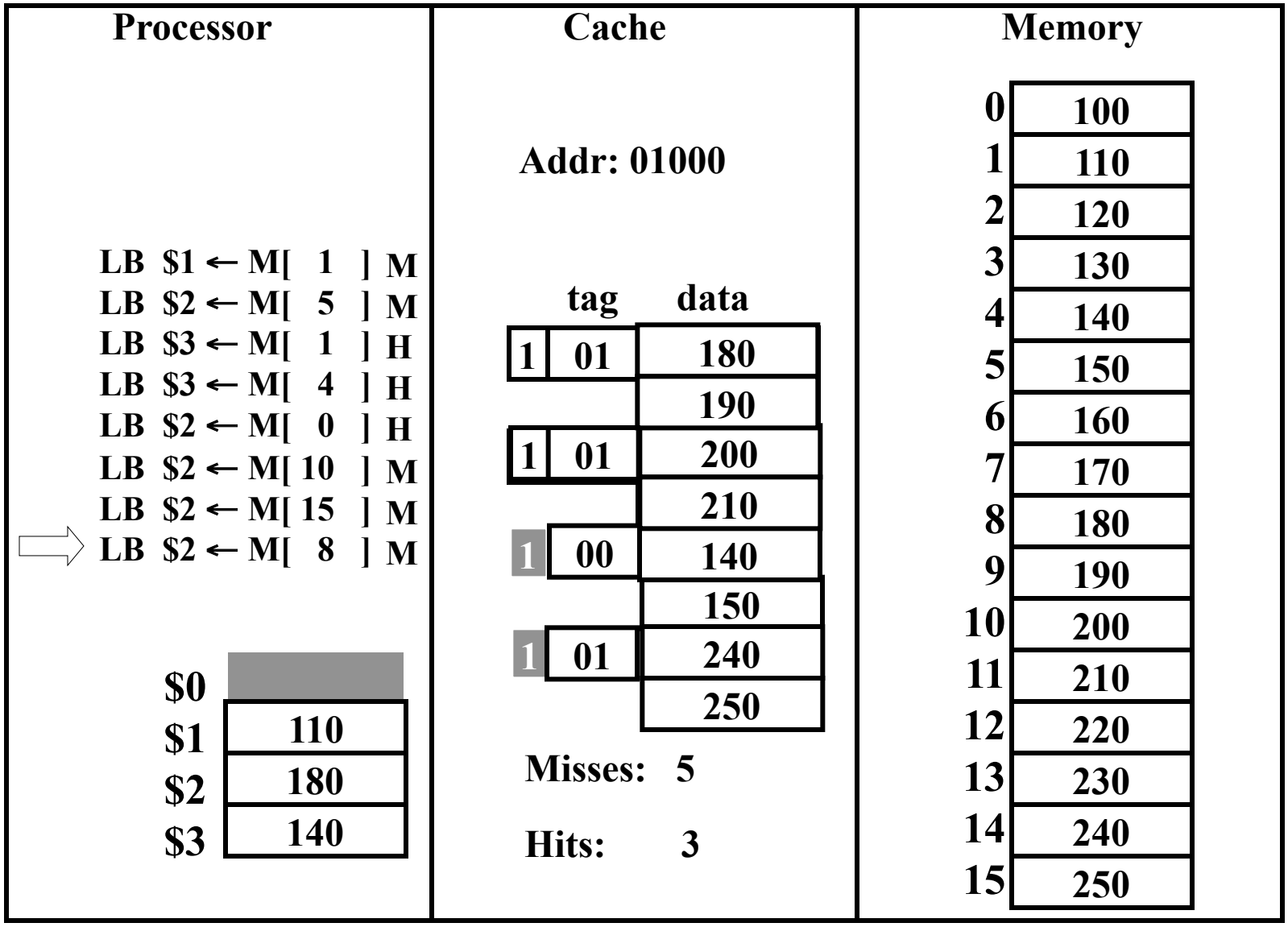
Using **byte addresses** in this example! Addr Bus = 5 bits

8th Access



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

8th Access



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

Misses

Three types of misses

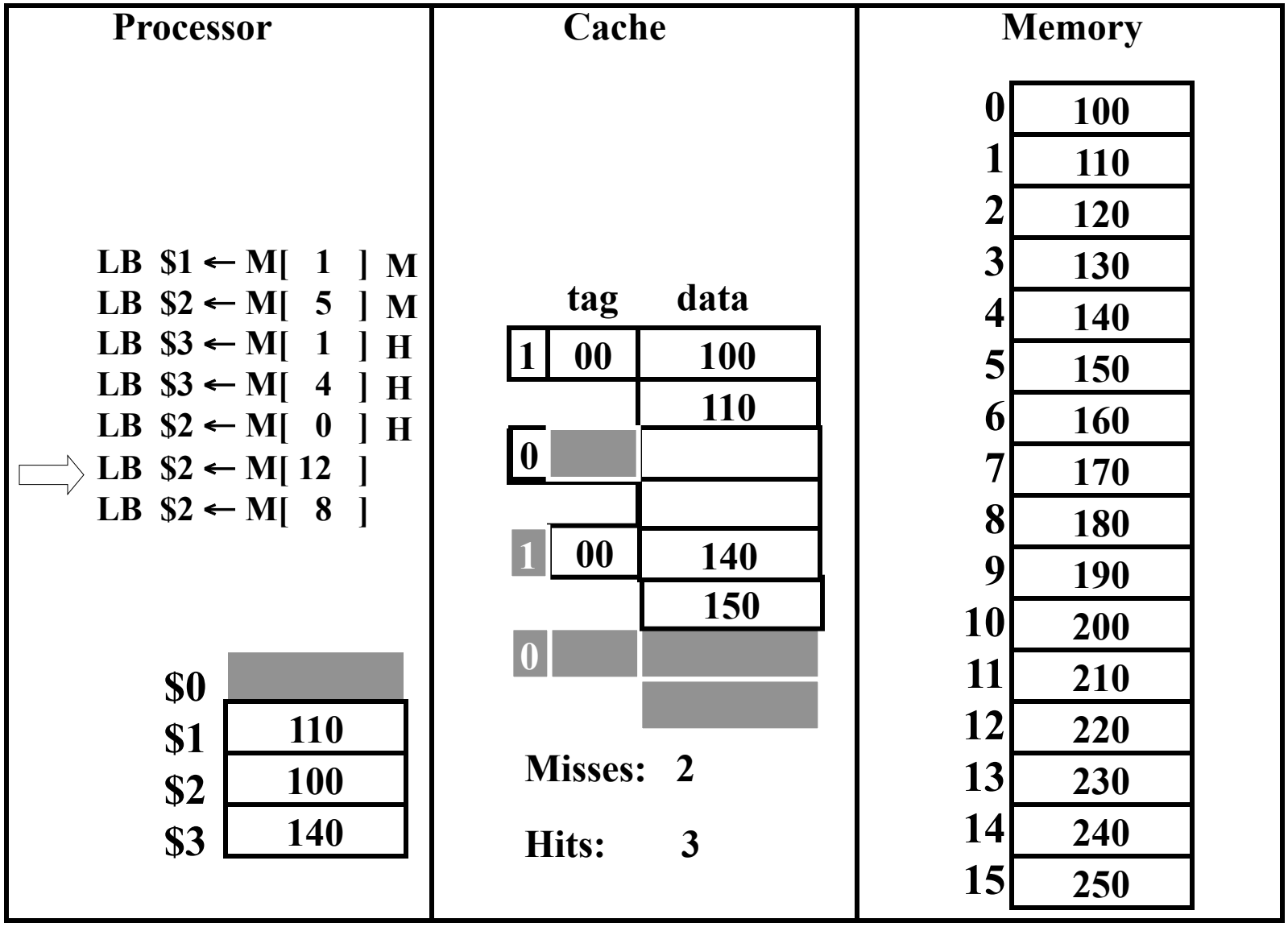
- Cold (aka Compulsory)
  - The line is being referenced for the first time
- Capacity
  - The line was evicted because the cache was not large enough
- Conflict
  - The line was evicted because of another access whose index conflicted

Misses

How to avoid…

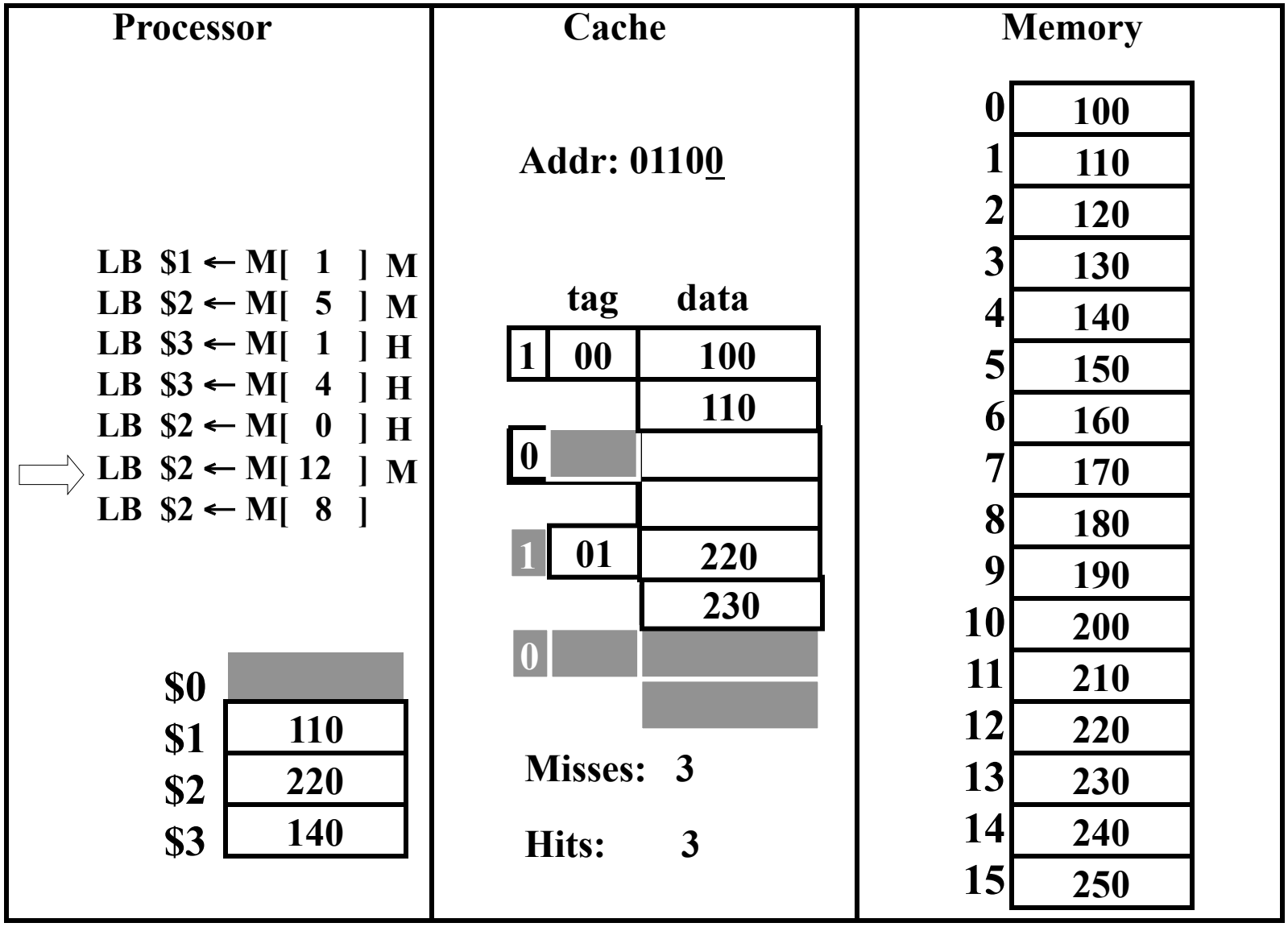
- Cold Misses
  - Unavoidable? The data was never in the cache…
  - Prefetching!
- Capacity Misses
  - Buy more SRAM
- Conflict Misses
  - Use a more flexible cache design

6th Access



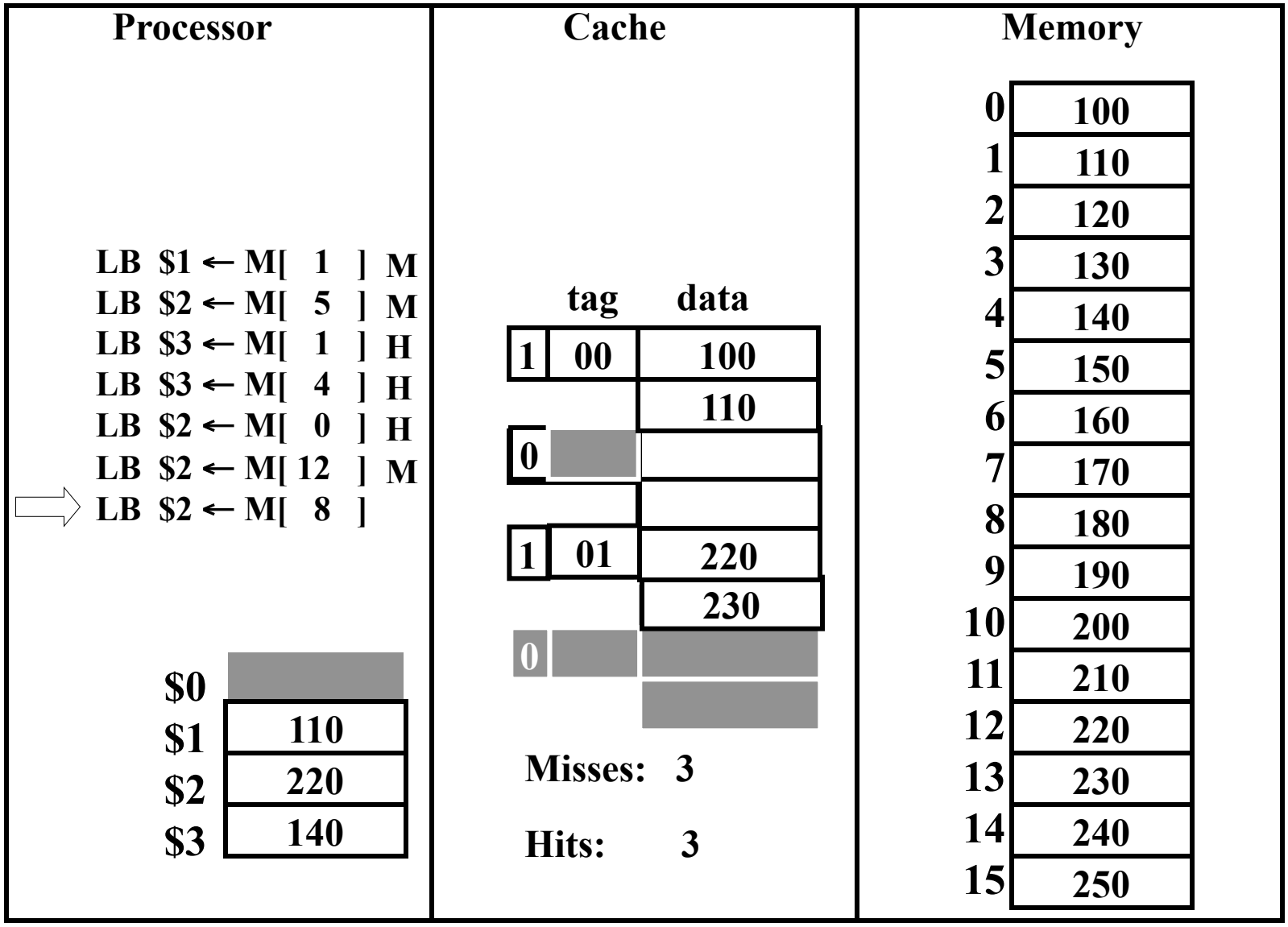
Using **byte addresses** in this example! Addr Bus = 5 bits

6th Access



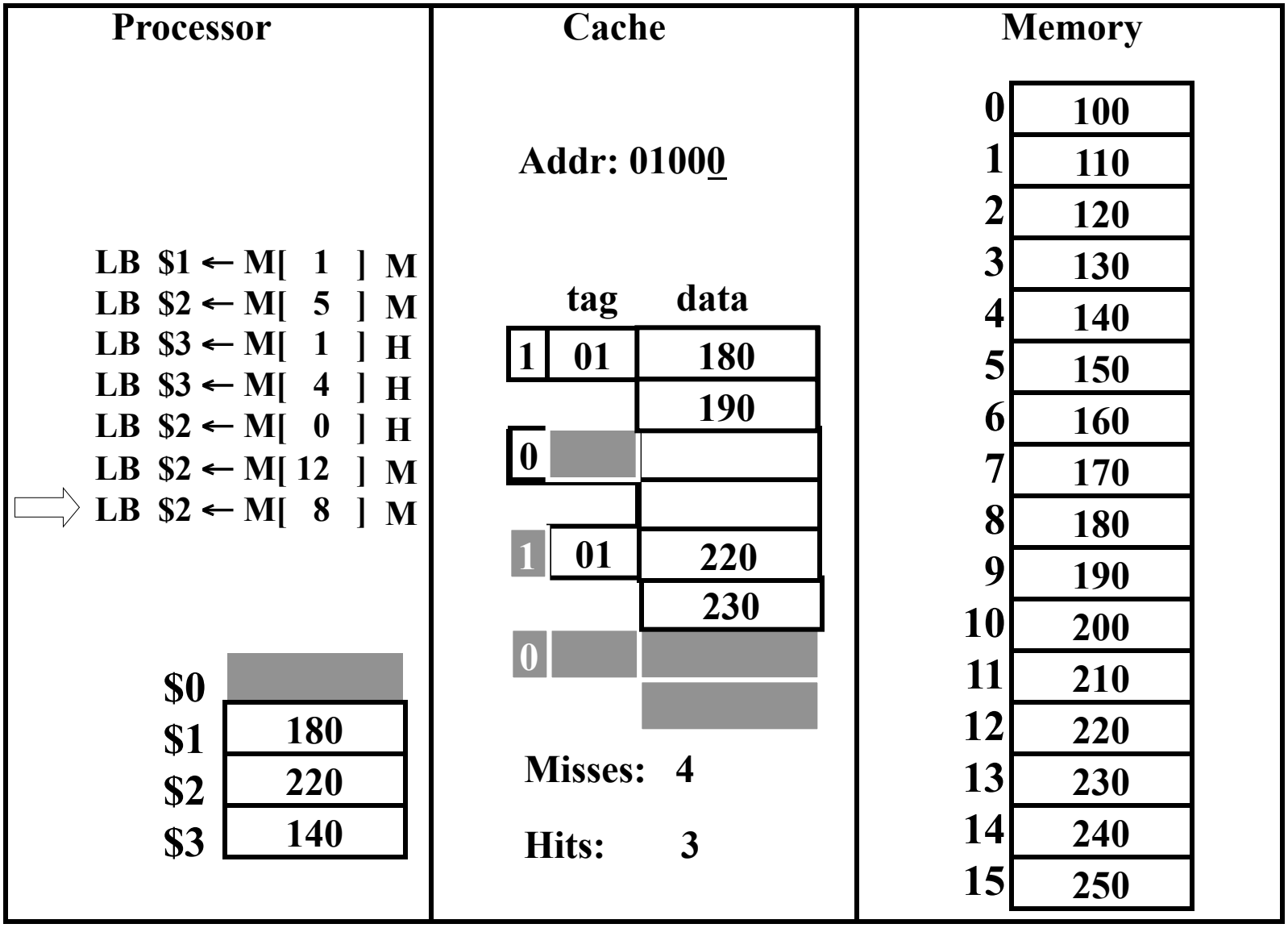
Using **byte addresses** in this example! Addr Bus = 5 bits

7th Access



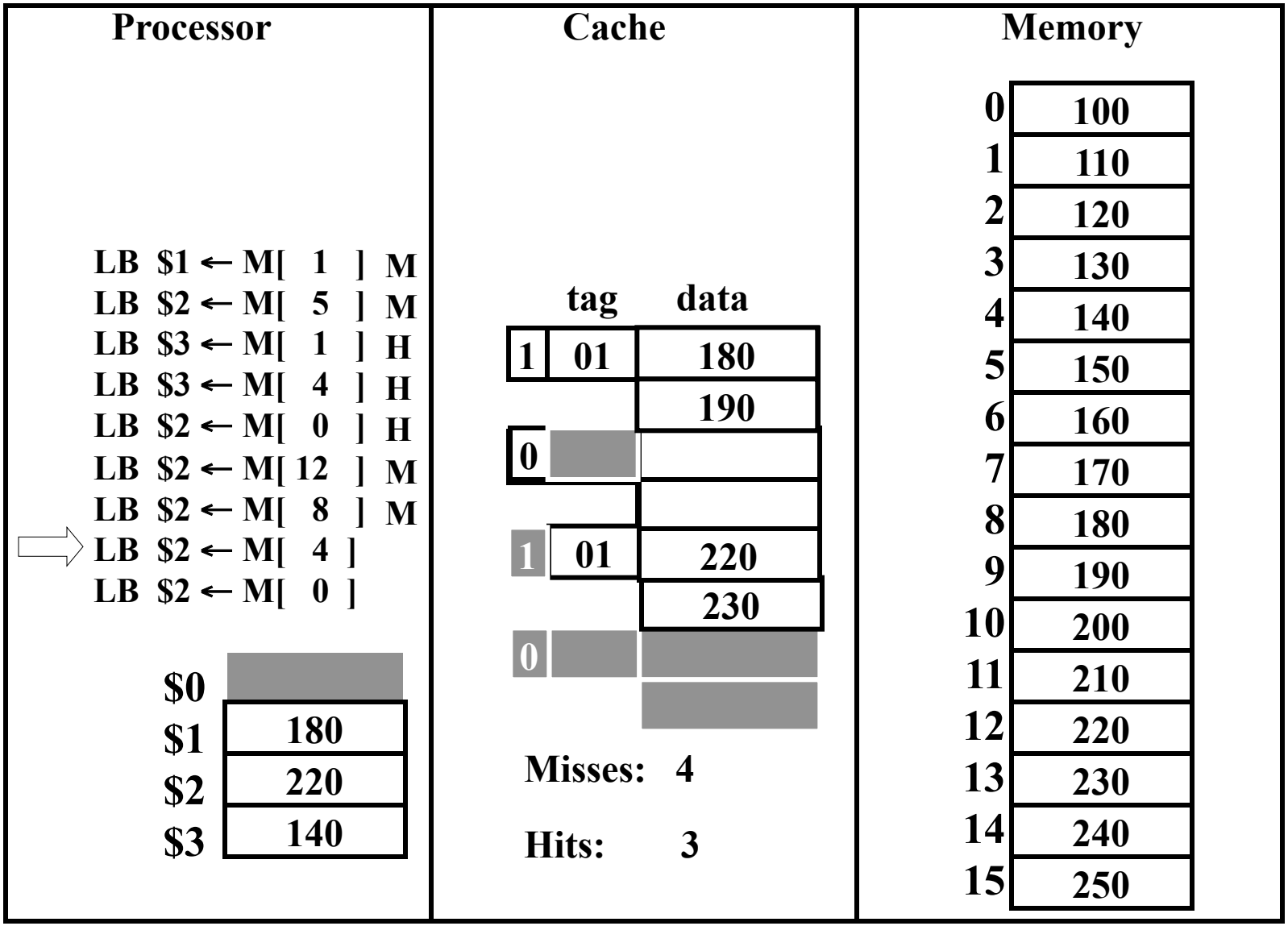
Using **byte addresses** in this example! Addr Bus = 5 bits

7th Access



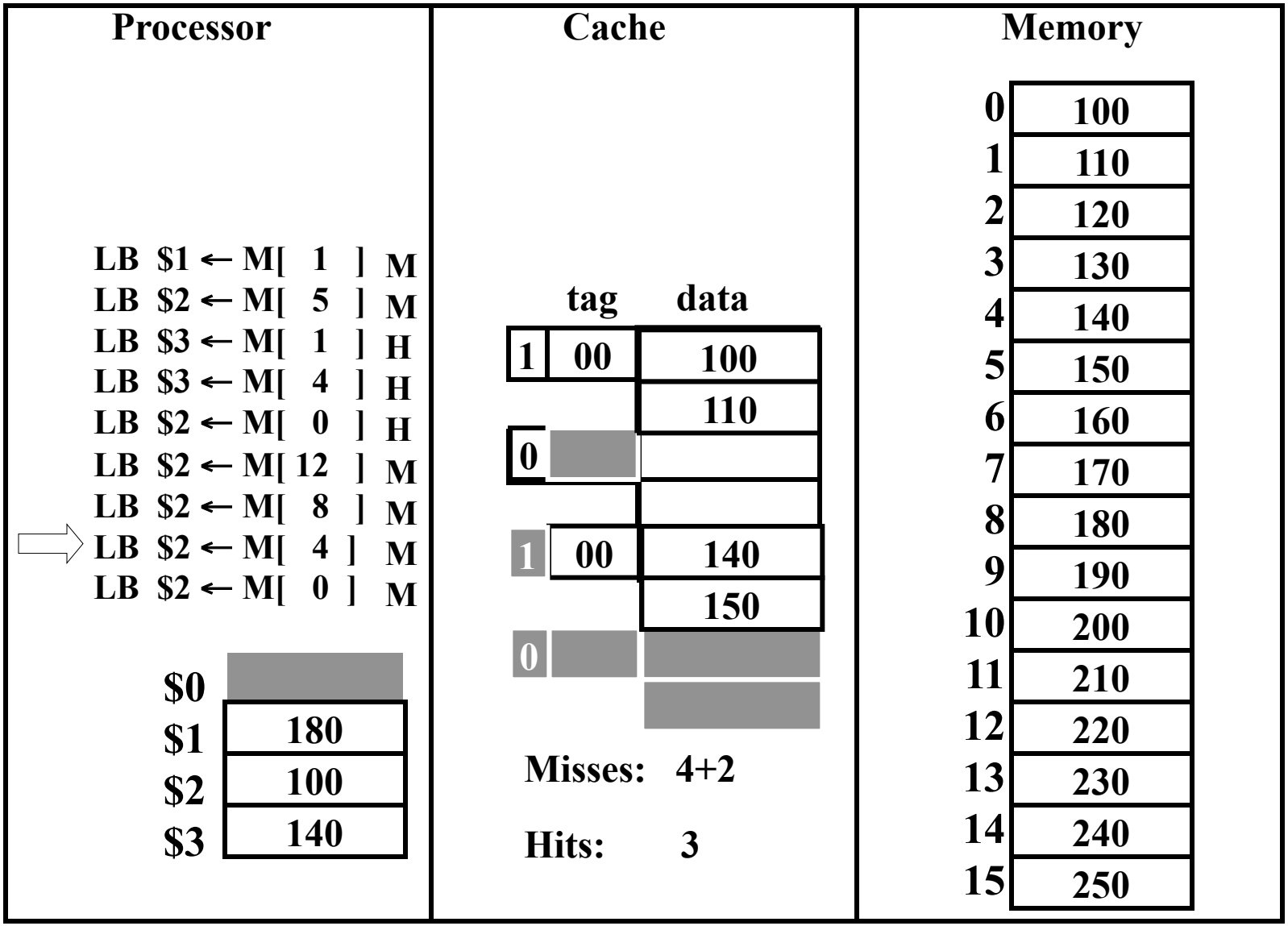
Using **byte addresses** in this example! Addr Bus = 5 bits

8th and 9th Accesses



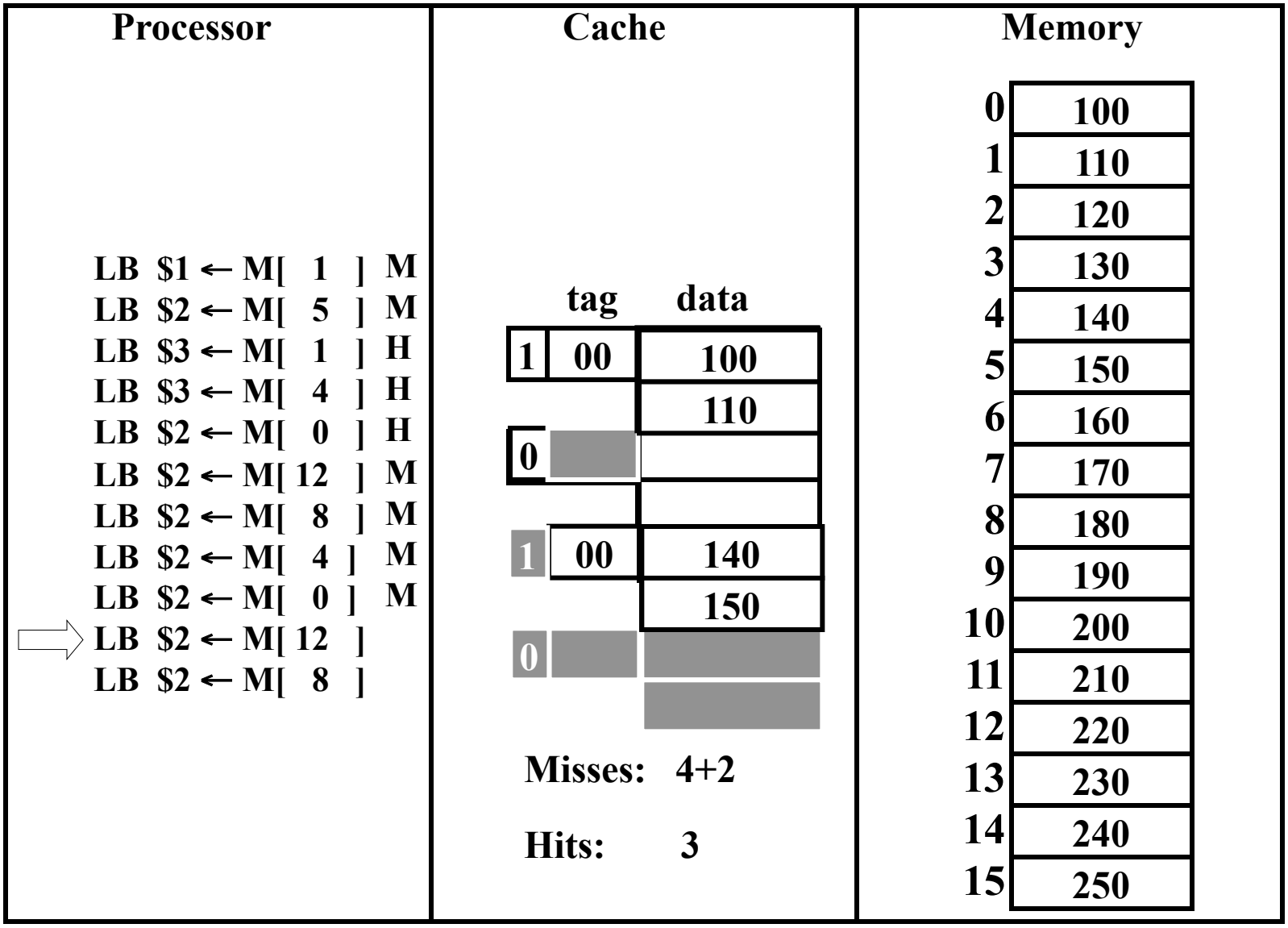
Using **byte addresses** in this example! Addr Bus = 5 bits

8th and 9th Accesses



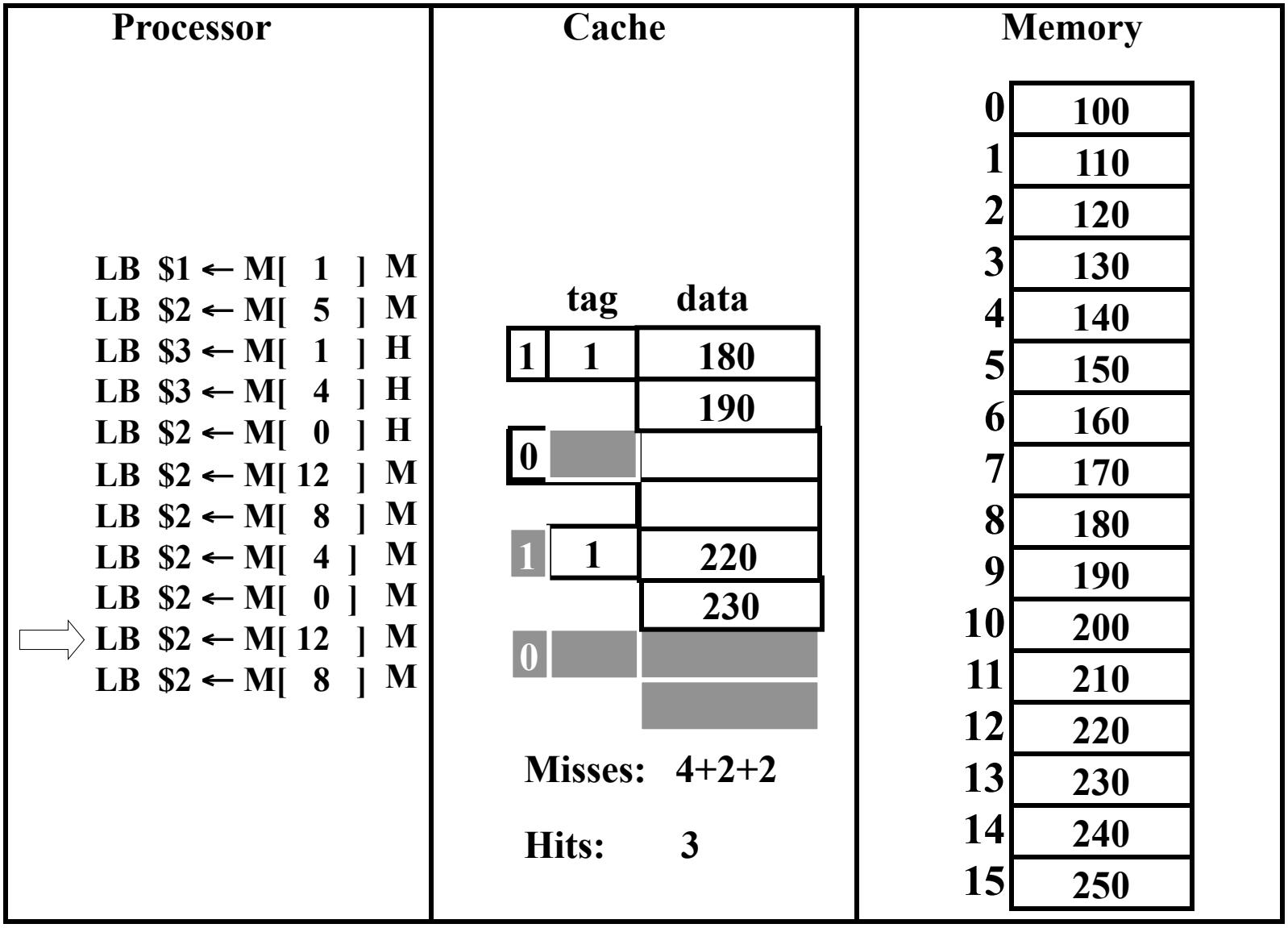
Using **byte addresses** in this example! Addr Bus = 5 bits

10th and 11th Accesses



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

10th and 11th Accesses



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

Cache Organization

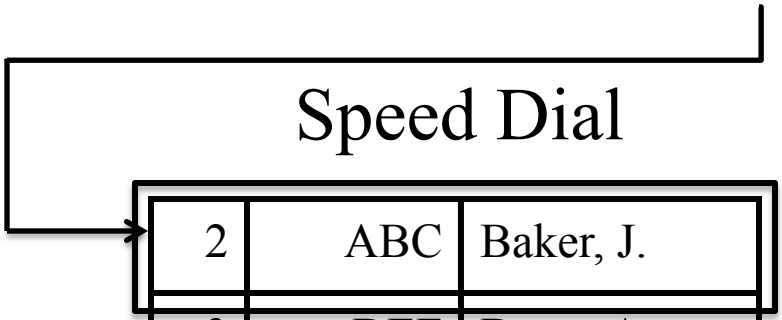
How to avoid Conflict Misses?!

Three common designs

- Direct mapped: Block can only be in one line in the cache
  - Contact numbers are assigned in the Speed Dial by first letter.
- Fully associative: Block can be anywhere in the cache
  - Contact numbers are assigned without any rules.
- Set-associative: Block can be in a few (2 to 8) places in the cache
  - Every digit in the Speed Dial maps to a new Speed Dial

Direct Mapped Cache

- Use the first letter to index!



Speed Dial

2	ABC	Baker, J.
3	DEF	Dunn, A.
4	GHI	Gill, S.
5	JKL	Jones, N.
6	MNO	Mann, B.
7	PQRS	Powell, J.
8	TUV	Taylor, B.
9	WXYZ	Wright, T.

Contacts

Baker, J.	111-111-1111
Baker, S.	222-222-2222
Dunn, A.	333-333-3333
Foster, D.	444-444-4444
Gill, D.	555-555-5555
Harris, F.	666-666-6666
Jones, N.	777-777-7777
Lee, V.	888-888-8888
Mann, B.	111-111-1119
Moore, F.	222-222-2229
Powell, C.	333-333-3339
Sam, J.	444-444-4449
Taylor, B.	555-555-5559
Taylor, O.	666-666-6669
Wright, T.	777-777-7779
Zhang, W.	888-888-8889

Block Size: 1 Contact Number

Fully Associative Cache

- No index!

Speed Dial

2	Mann, B.
3	Dunn, A.
4	Taylor, B.
5	Wright, T.
6	Baker, J.
7	Powell, J.
8	Gill, S.
9	Jones, N.

Contacts

Baker, J.	111-111-1111
Baker, S.	222-222-2222
Dunn, A.	333-333-3333
Foster, D.	444-444-4444
Gill, D.	555-555-5555
Harris, F.	666-666-6666
Jones, N.	777-777-7777
Lee, V.	888-888-8888
Mann, B.	111-111-1119
Moore, F.	222-222-2229
Powell, C.	333-333-3339
Sam, J.	444-444-4449
Taylor, B.	555-555-5559
Taylor, O.	666-666-6669
Wright, T.	777-777-7779
Zhang, W.	888-888-8889

Block Size: 1 Contact Number

Fully Associative Cache

- No index!
- Use the initial to offset!

Speed Dial

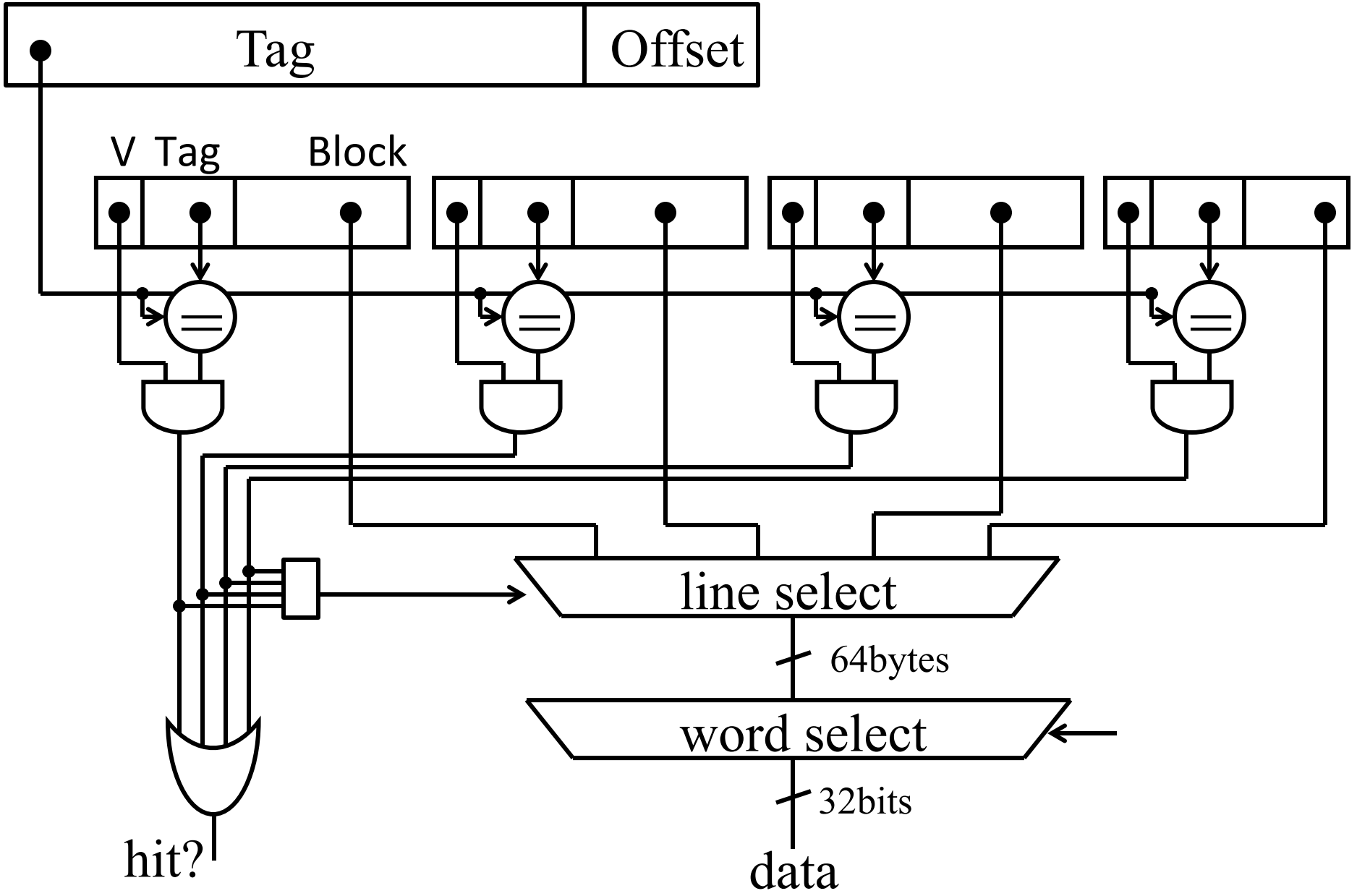
2	Mann, B.	Moore, F.
3	Powell, C.	Sam, J.
4	Gill, D.	Harris, F.
5	Wright, T.	Zhang, W.
6	Baker, J.	Baker, S.
7	Dunn, A.	Foster, D.
8	Taylor, B.	Taylor, O.
9	Jones, N.	Lee, V.

Block Size: 2 Contact Numbers

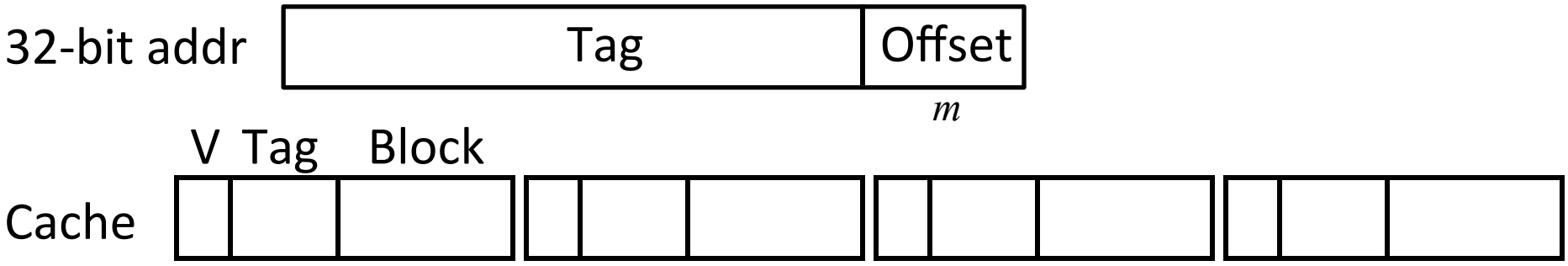
Contacts

Baker, J.	111-111-1111
Baker, S.	222-222-2222
Dunn, A.	333-333-3333
Foster, D.	444-444-4444
Gill, D.	555-555-5555
Harris, F.	666-666-6666
Jones, N.	777-777-7777
Lee, V.	888-888-8888
Mann, B.	111-111-1119
Moore, F.	222-222-2229
Powell, C.	333-333-3339
Sam, J.	444-444-4449
Taylor, B.	555-555-5559
Taylor, O.	666-666-6669
Wright, T.	777-777-7779
Zhang, W.	888-888-8889

Fully Associative Cache (Reading)



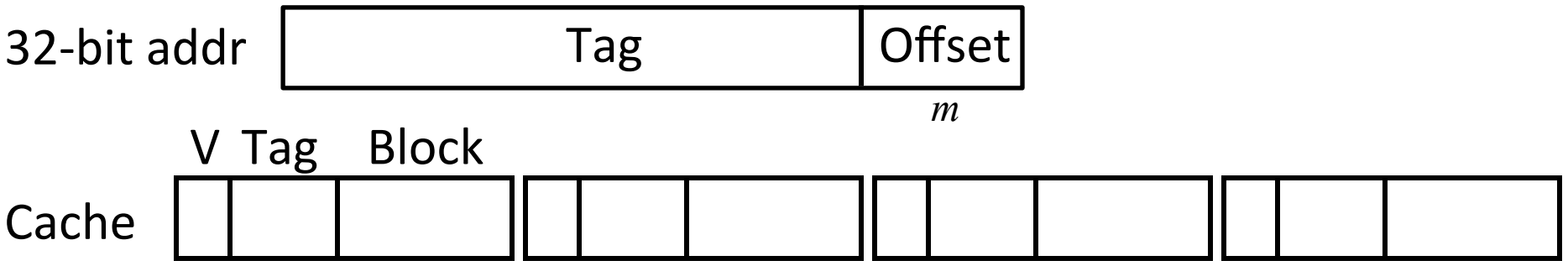
Fully Associative Cache (Reading)



m 2^n

Q: How big is the cache (data only)?

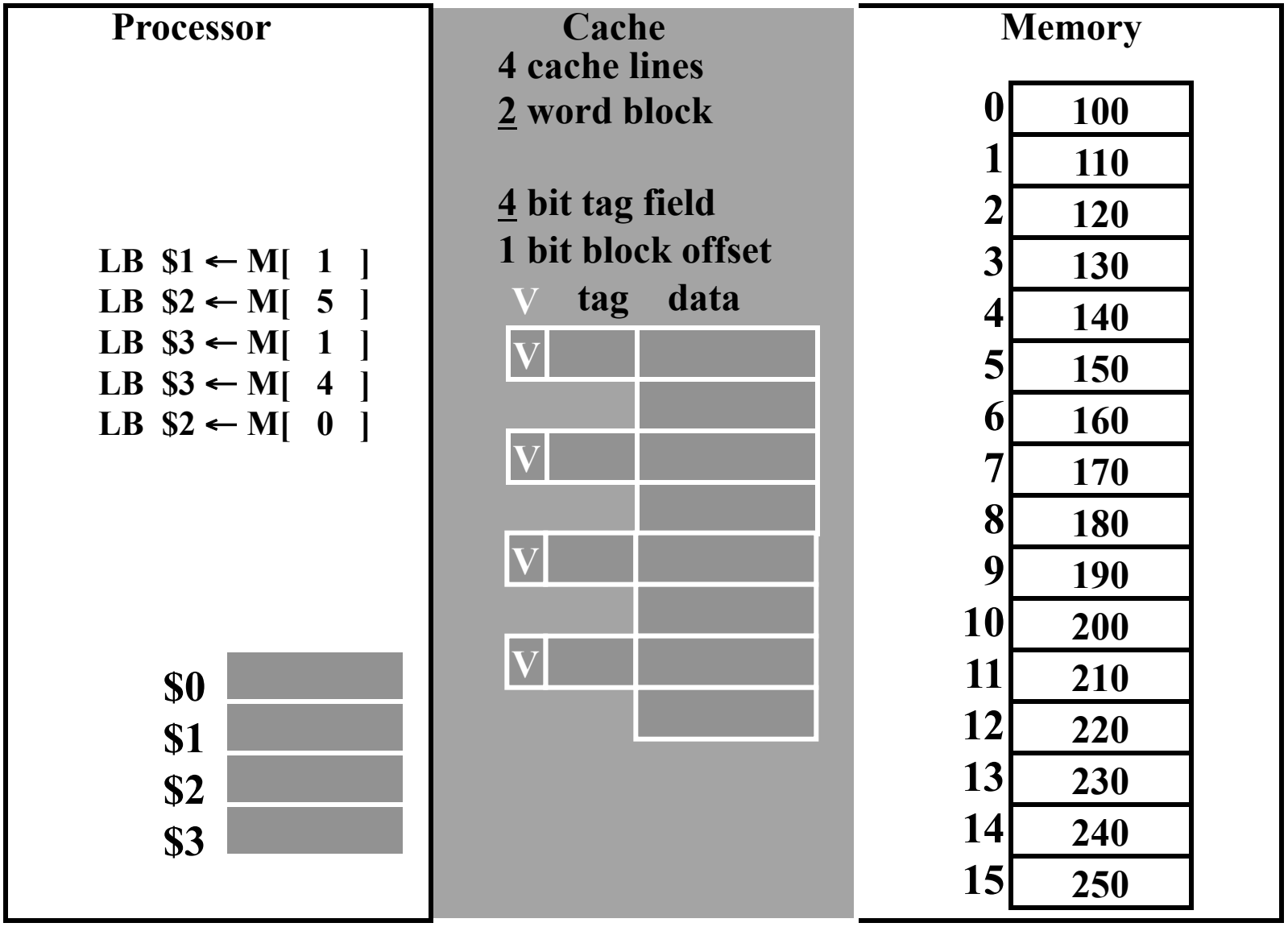
Fully Associative Cache (Reading)



m bit offset, 2^n blocks (cache lines)

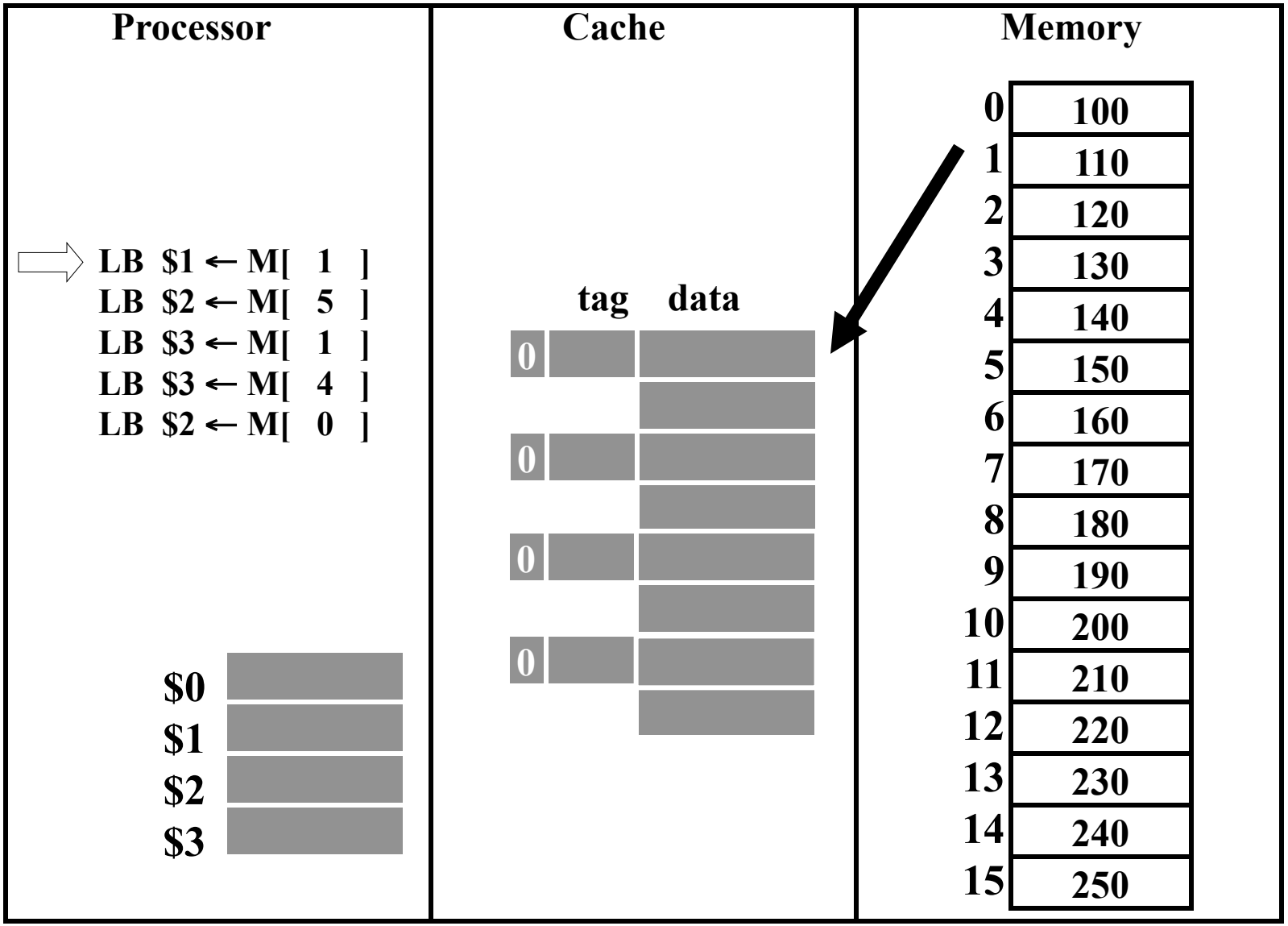
Q: How much SRAM needed (***data*** + ***overhead***)?

Example: Fully Associative Cache



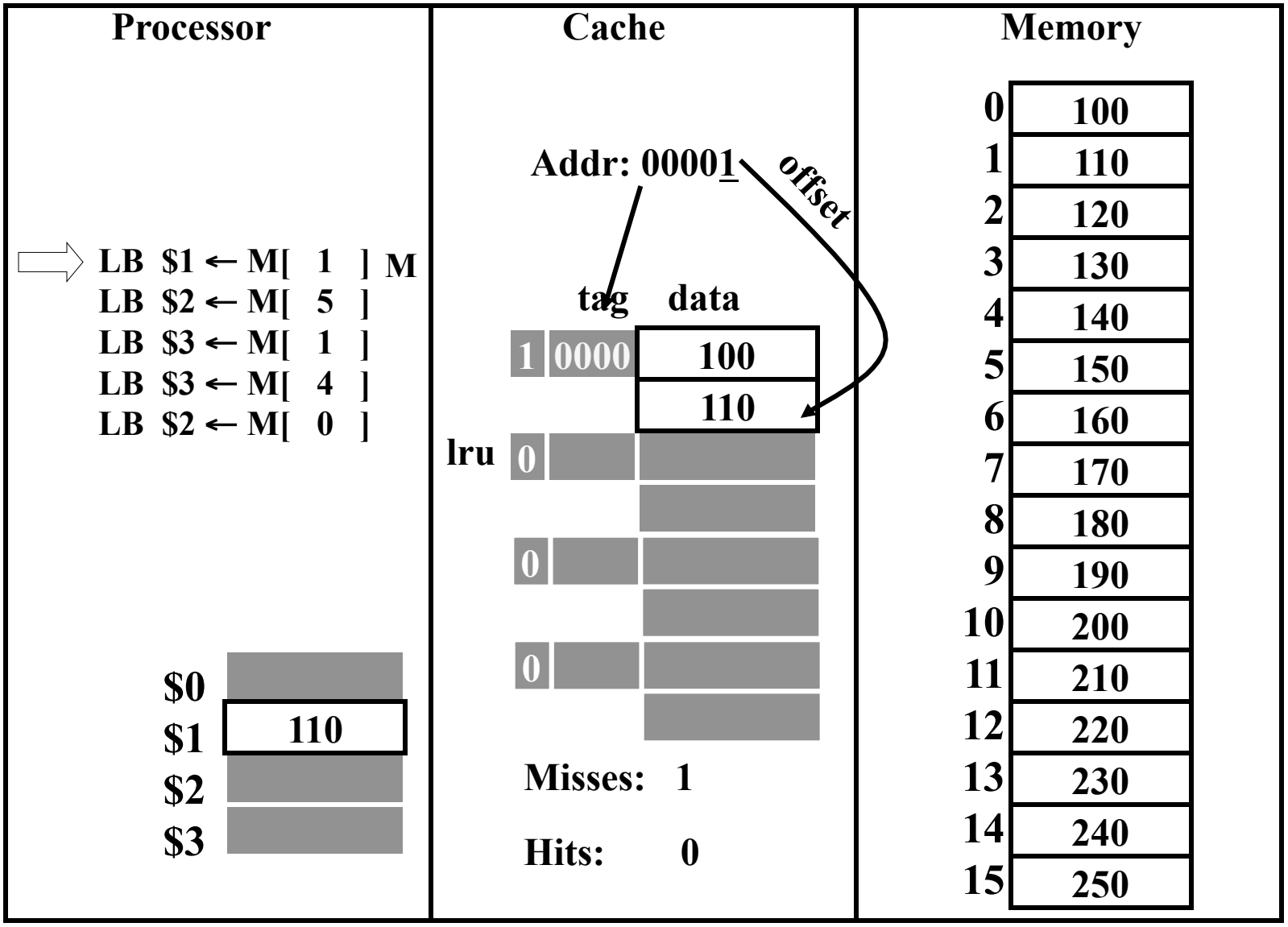
Using **byte addresses** in this example! Addr Bus = 5 bits

1st Access



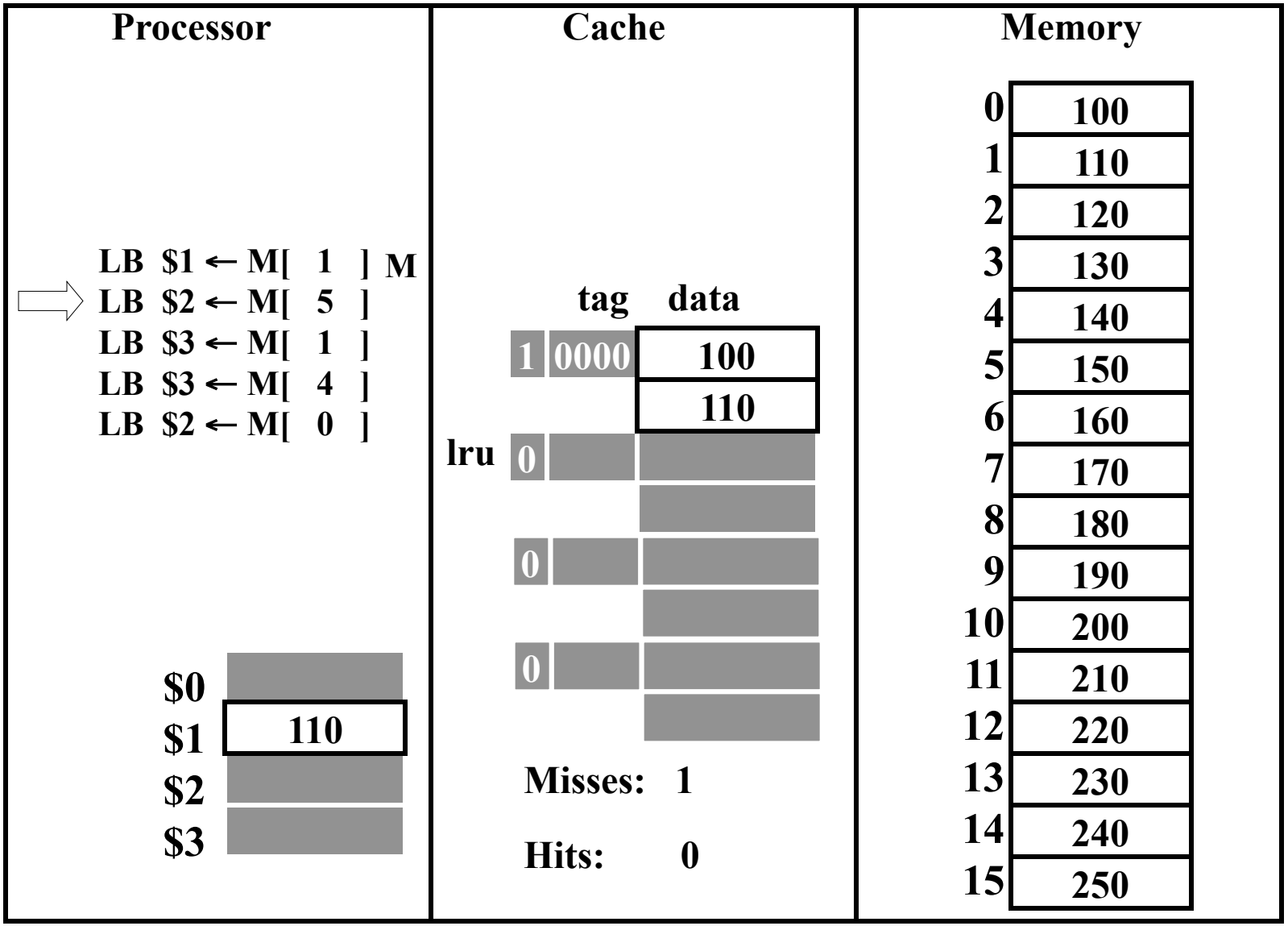
Using **byte addresses** in this example! Addr Bus = 5 bits

1st Access



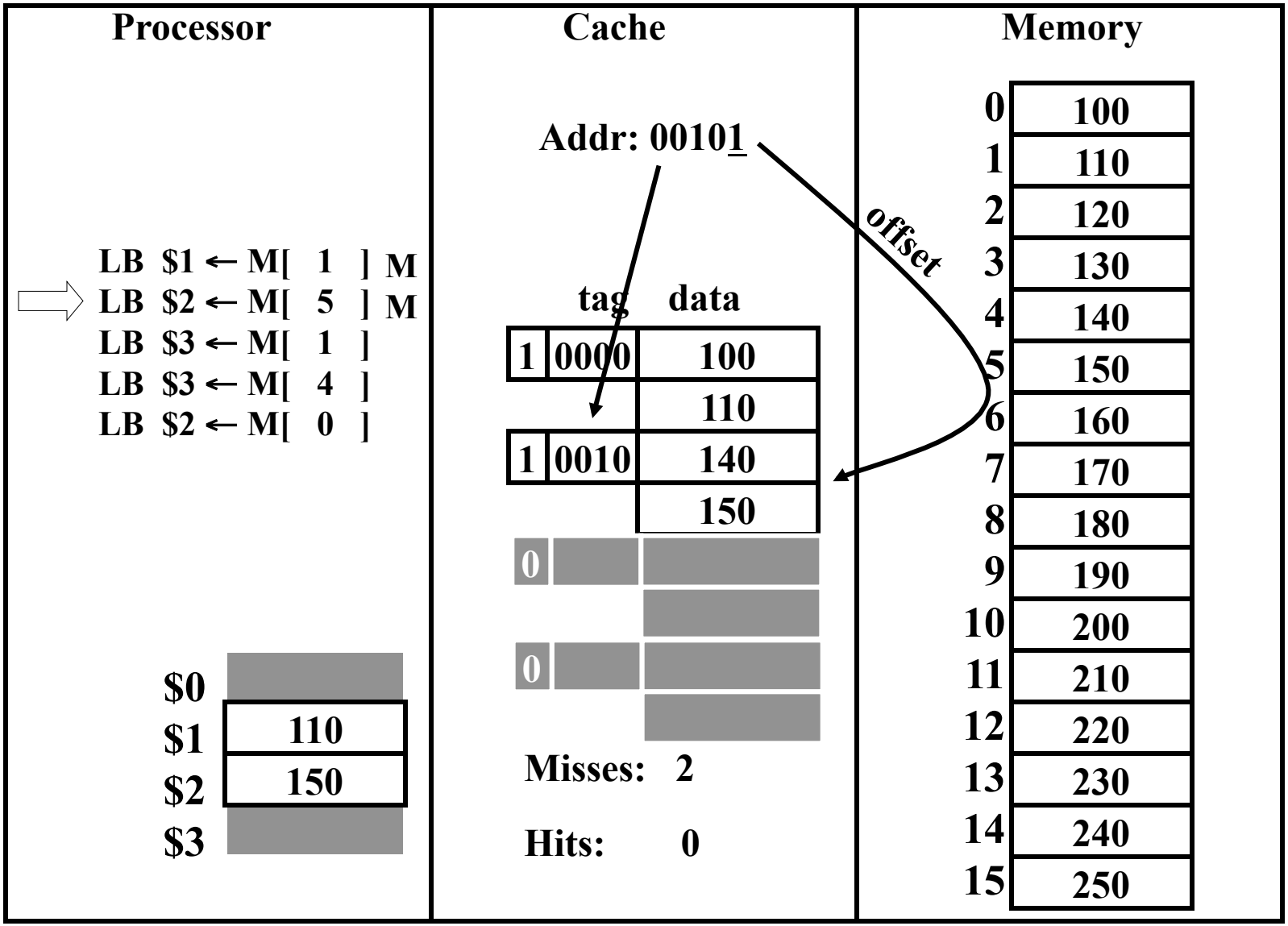
Using **byte addresses** in this example! Addr Bus = 5 bits

2nd Access



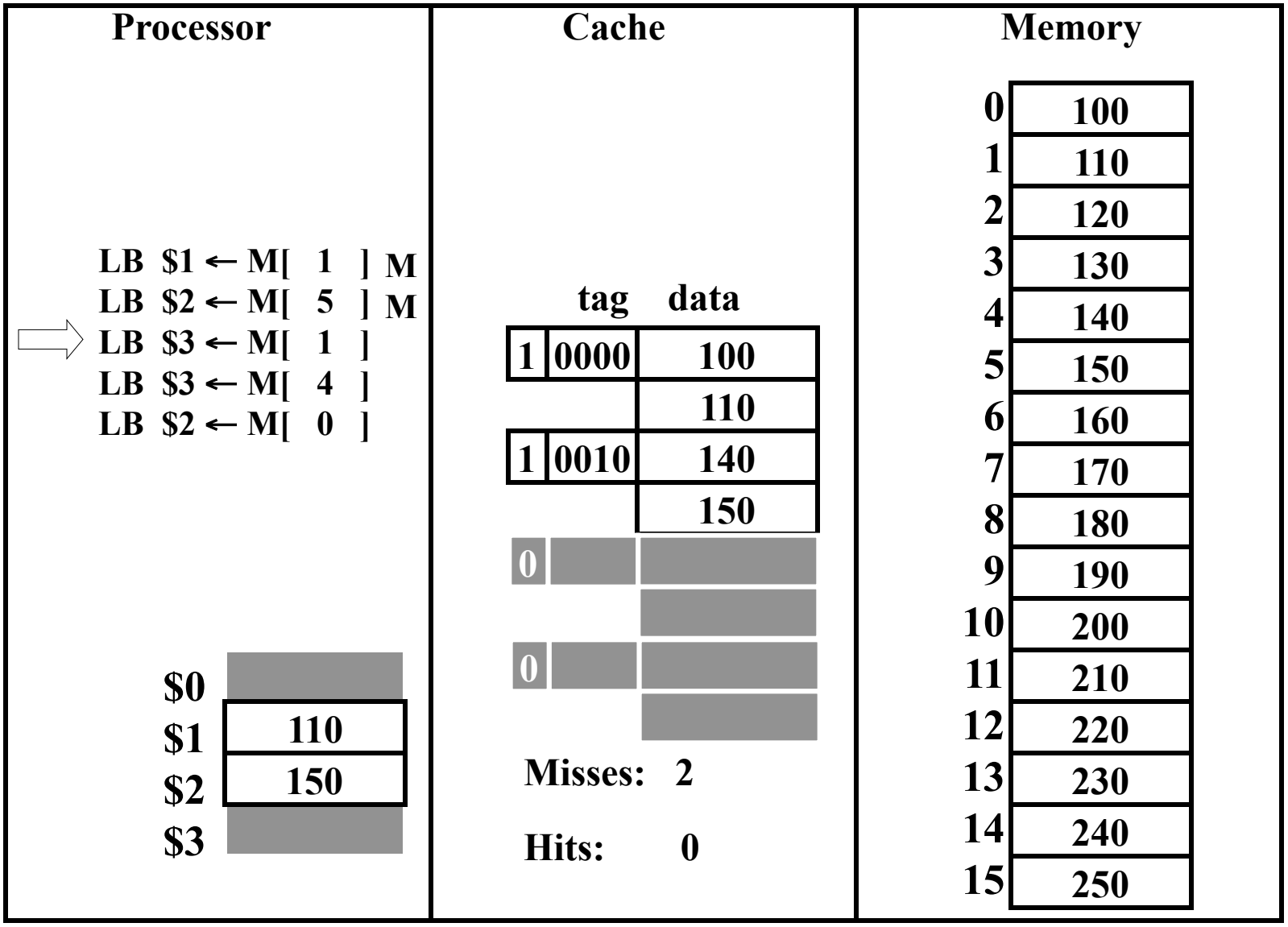
Using **byte addresses** in this example! Addr Bus = 5 bits

2nd Access



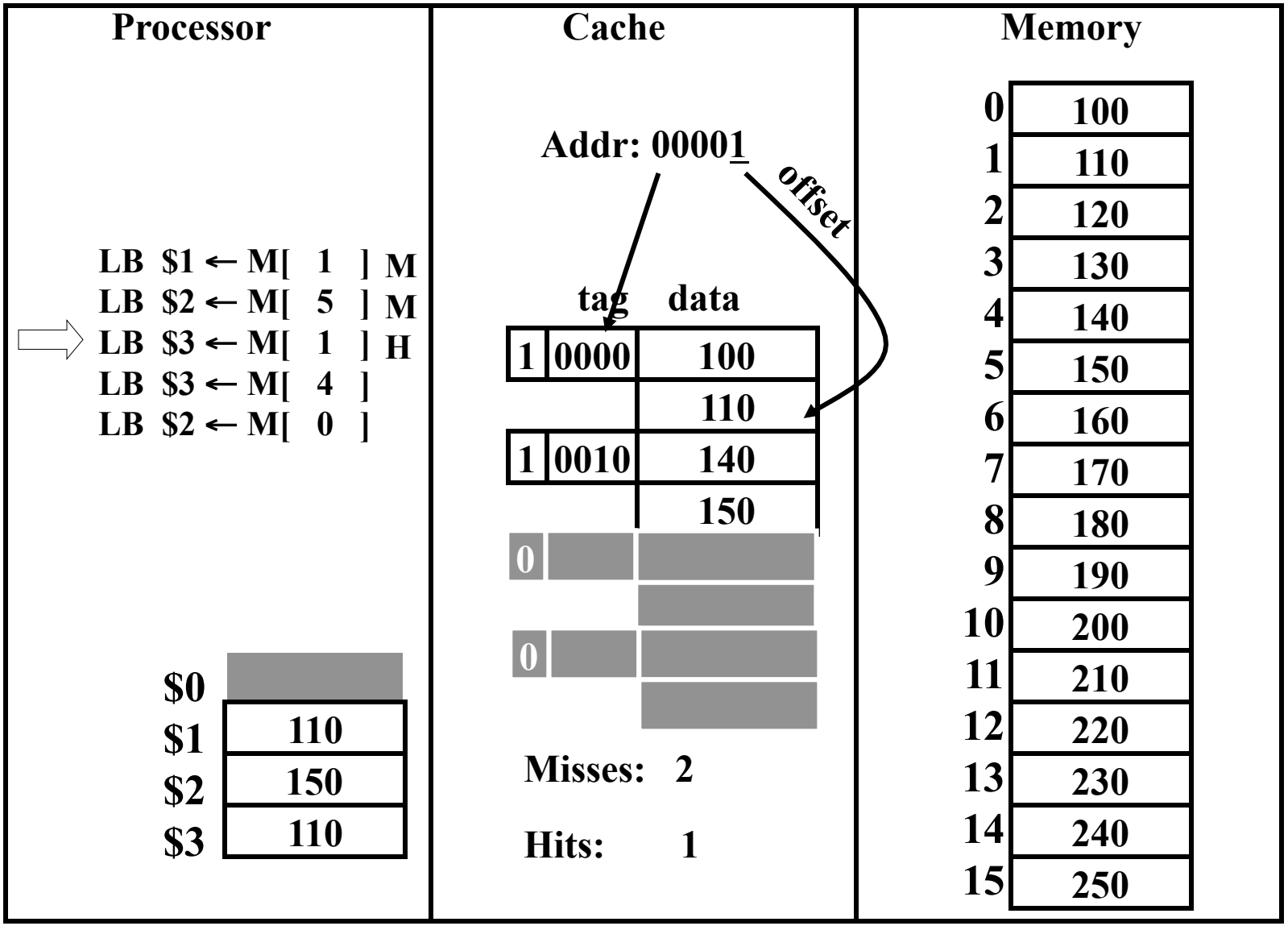
Using **byte addresses** in this example! Addr Bus = 5 bits

3rd Access



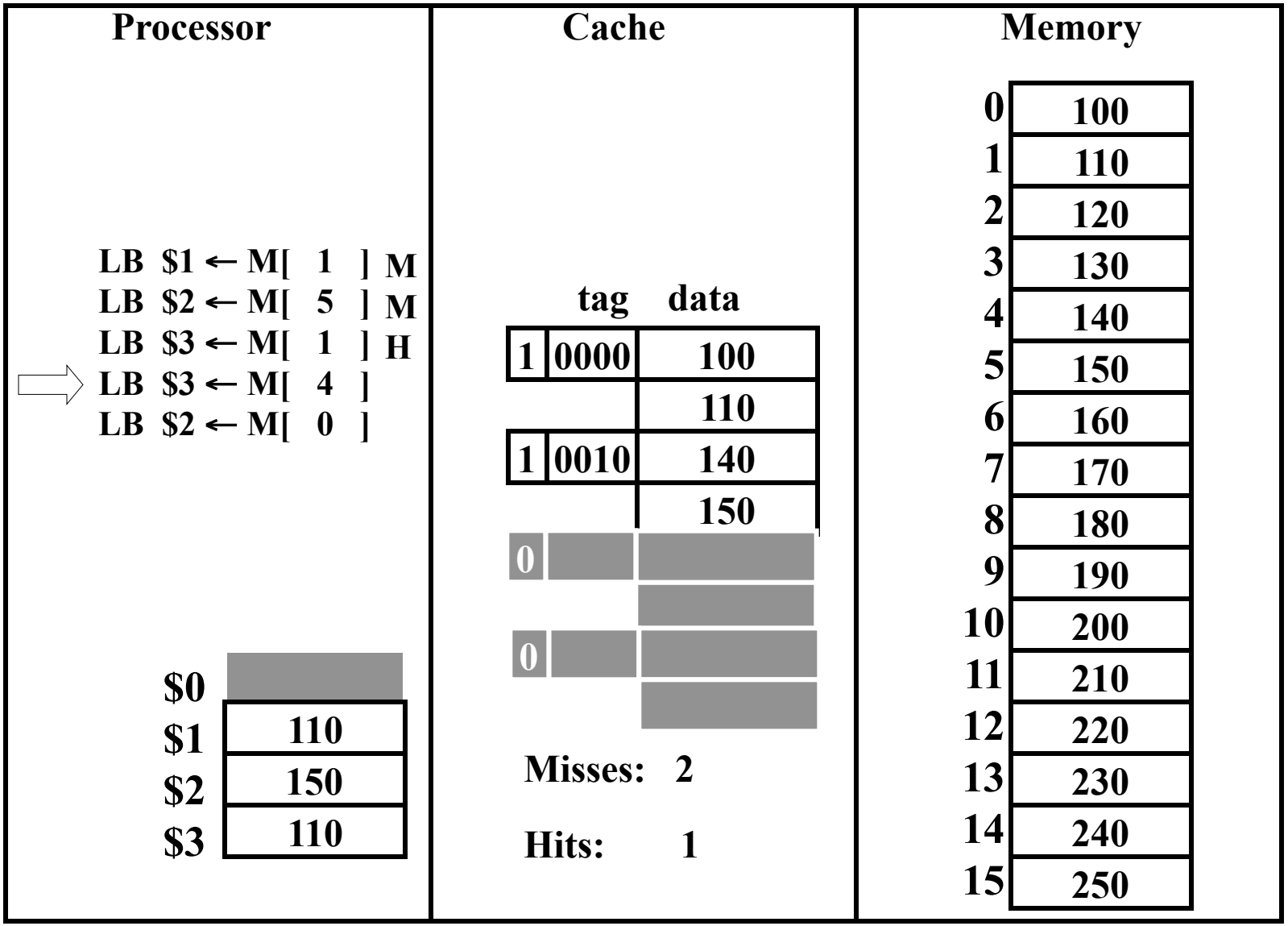
Using **byte addresses** in this example! Addr Bus = 5 bits

3rd Access



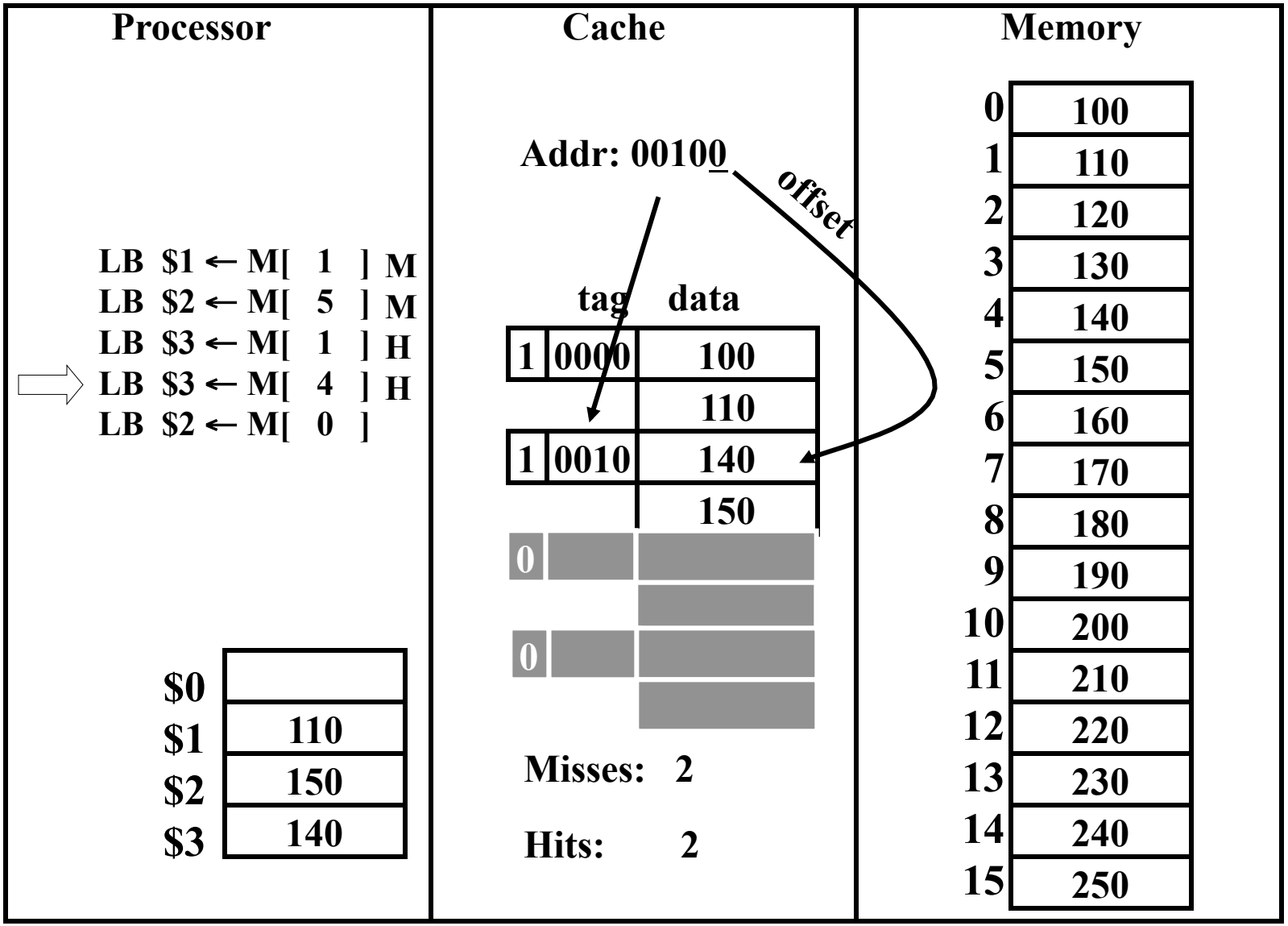
Using **byte addresses** in this example! Addr Bus = 5 bits

4th Access



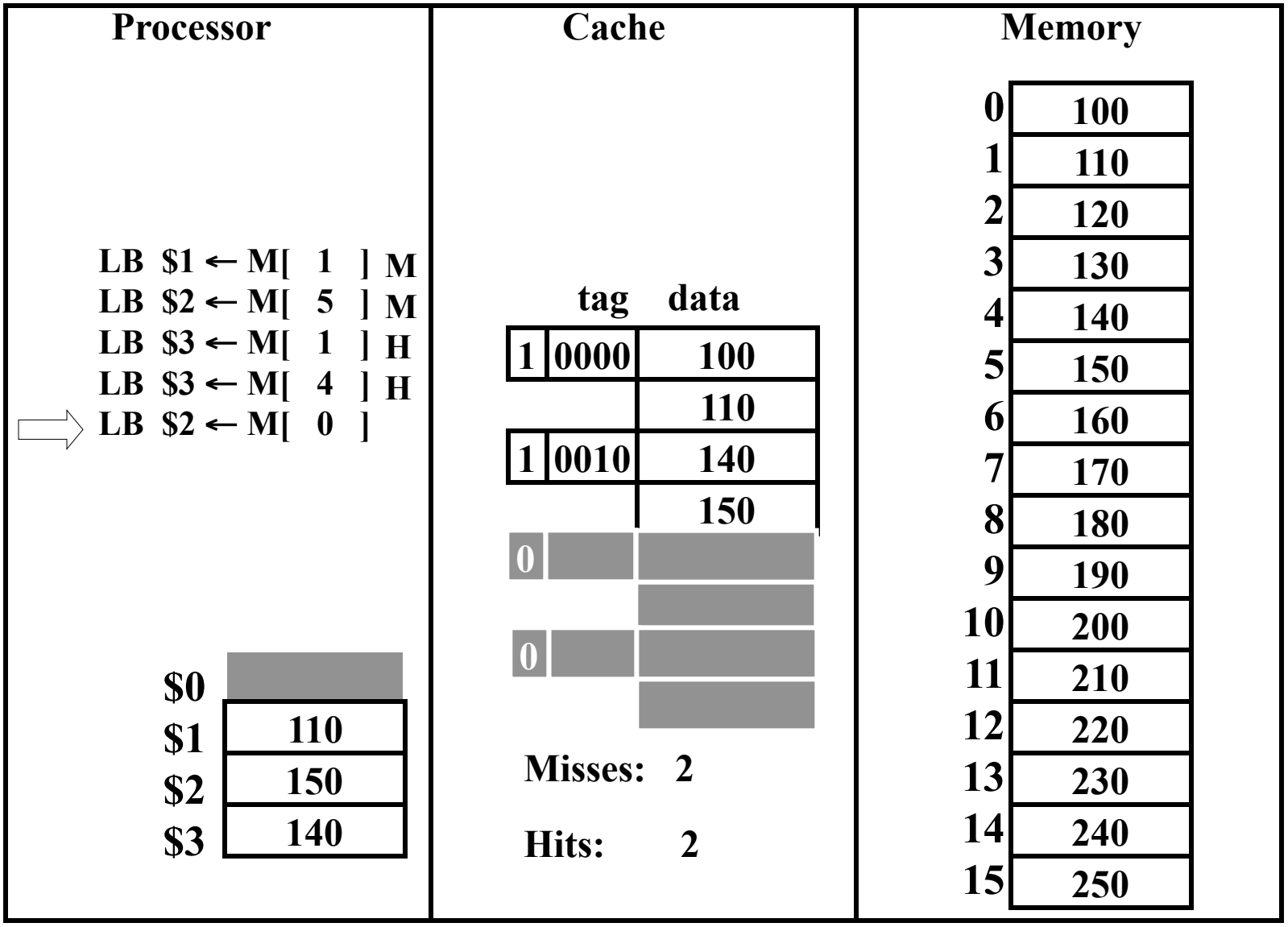
Using **byte addresses** in this example! Addr Bus = 5 bits

4th Access



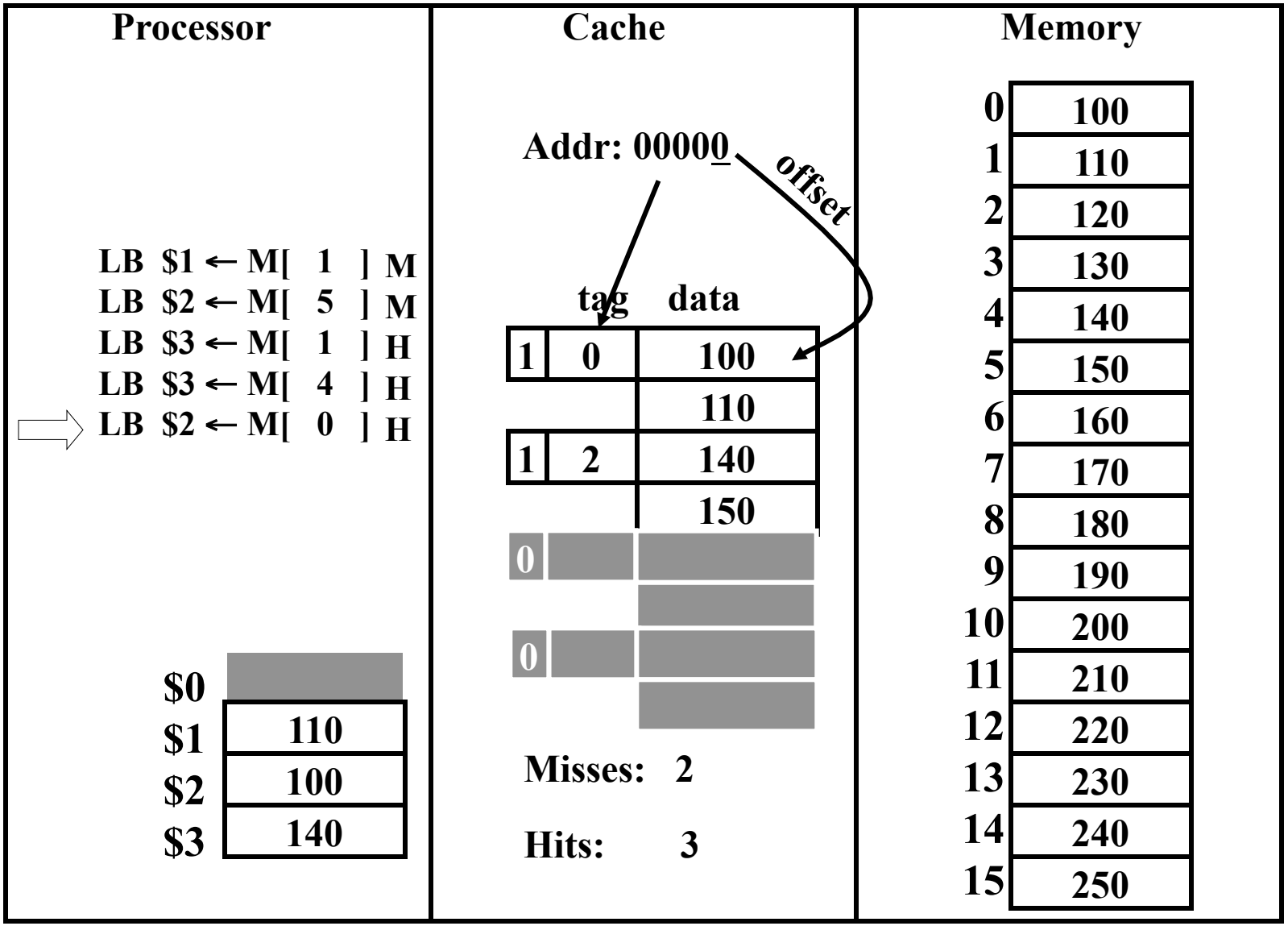
Using **byte addresses** in this example! Addr Bus = 5 bits

5th Access



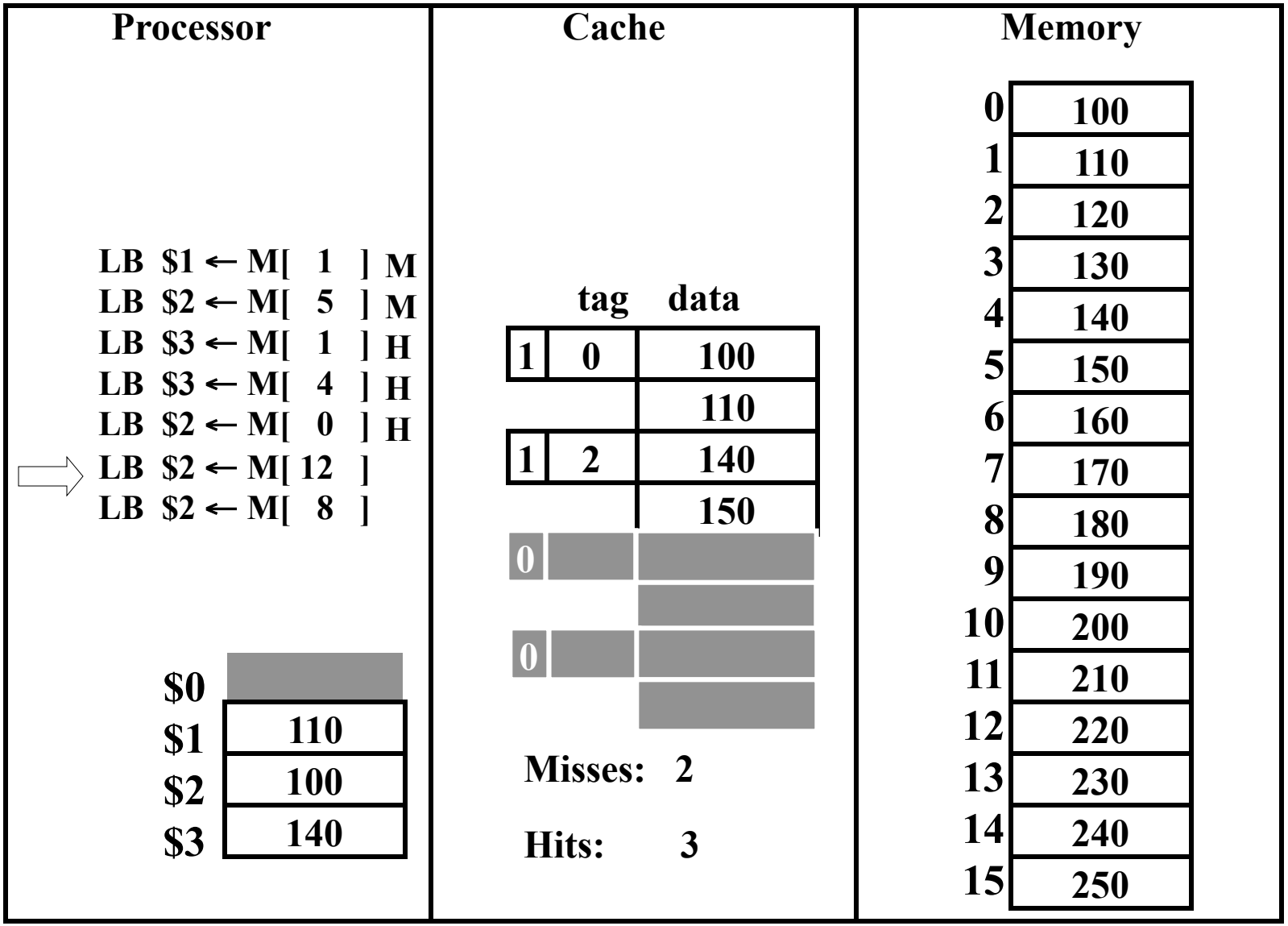
Using **byte addresses** in this example! Addr Bus = 5 bits

5th Access



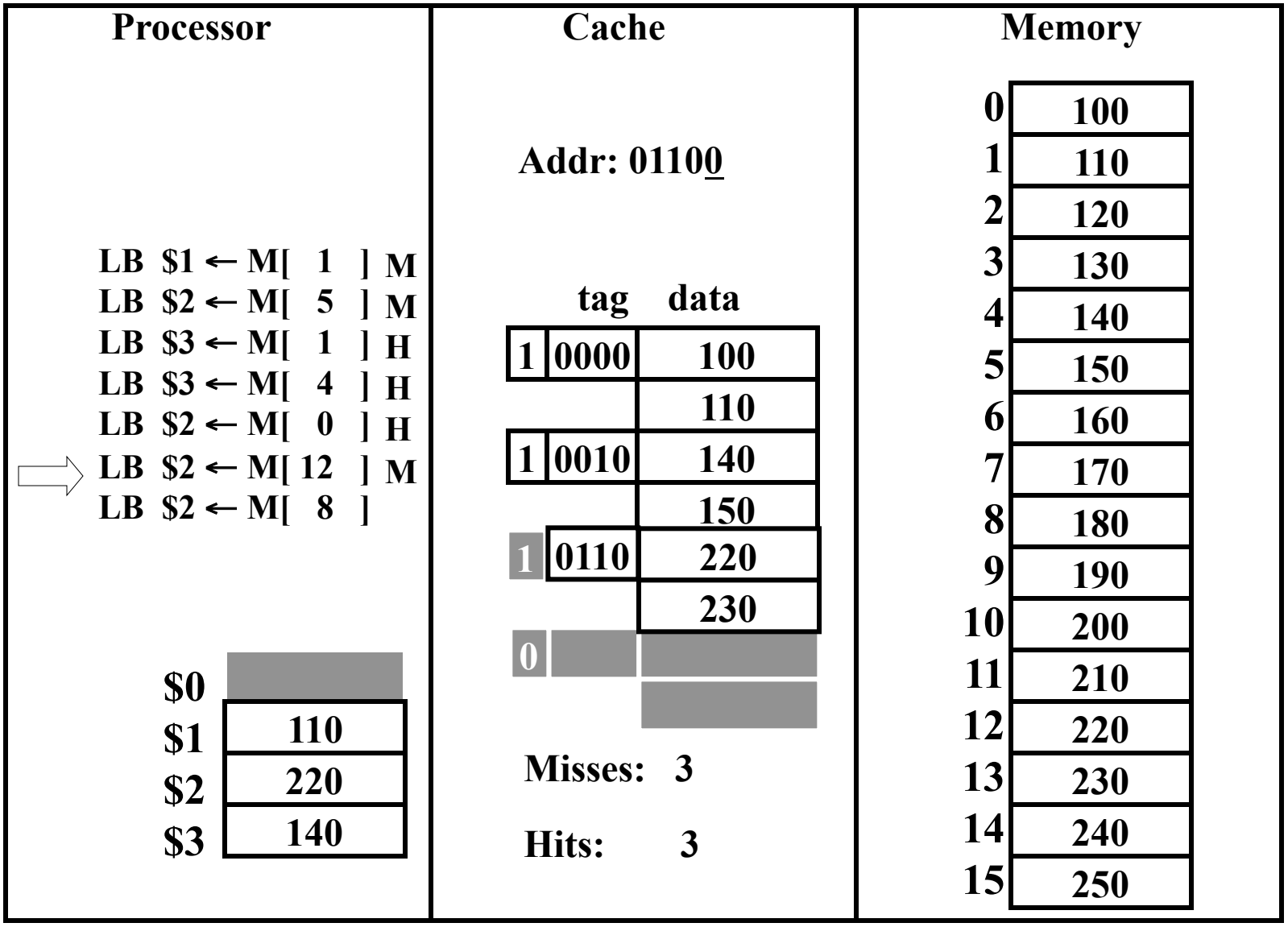
Using **byte addresses** in this example! Addr Bus = 5 bits

6th Access



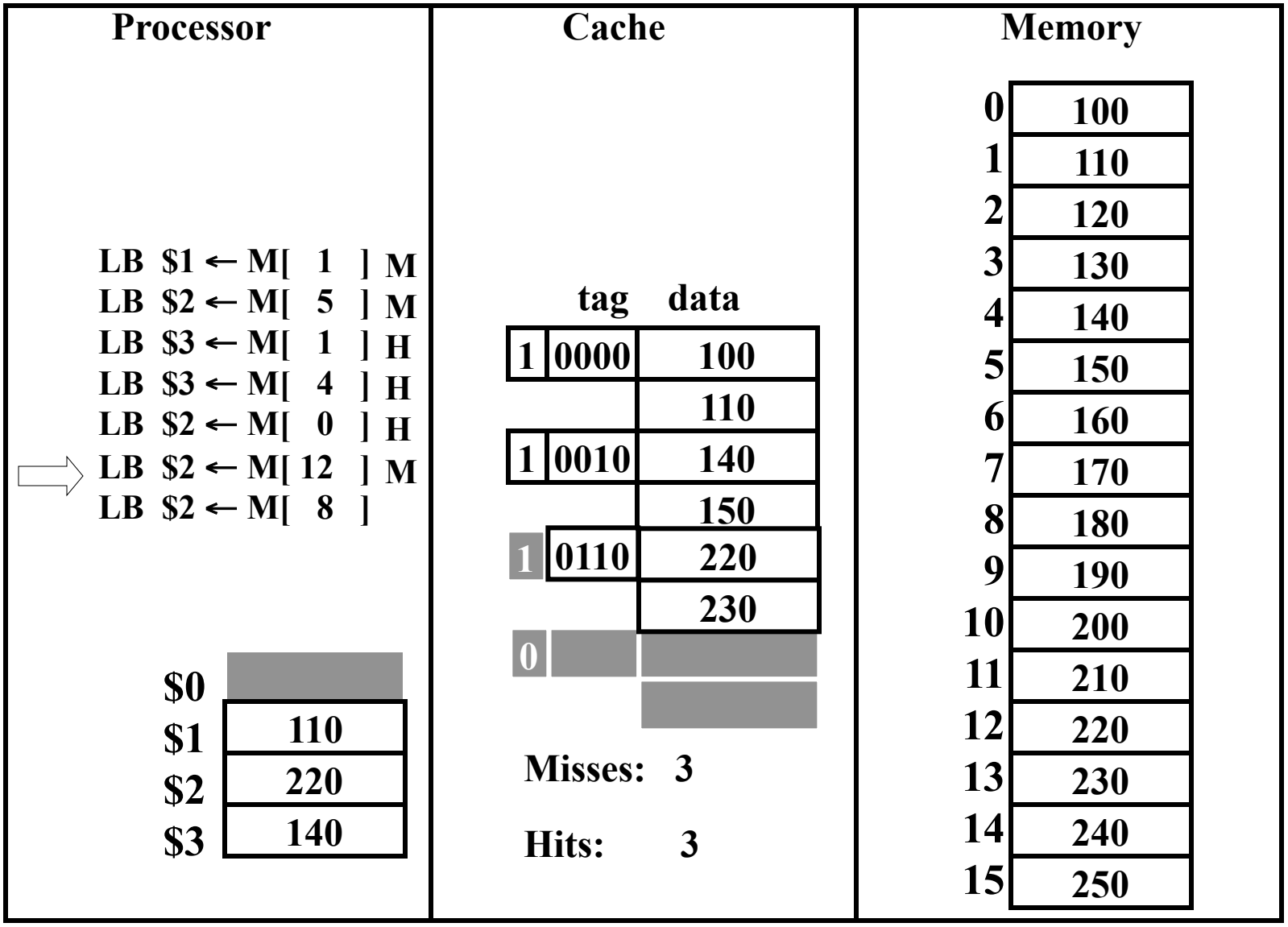
Using **byte addresses** in this example! Addr Bus = 5 bits

6th Access



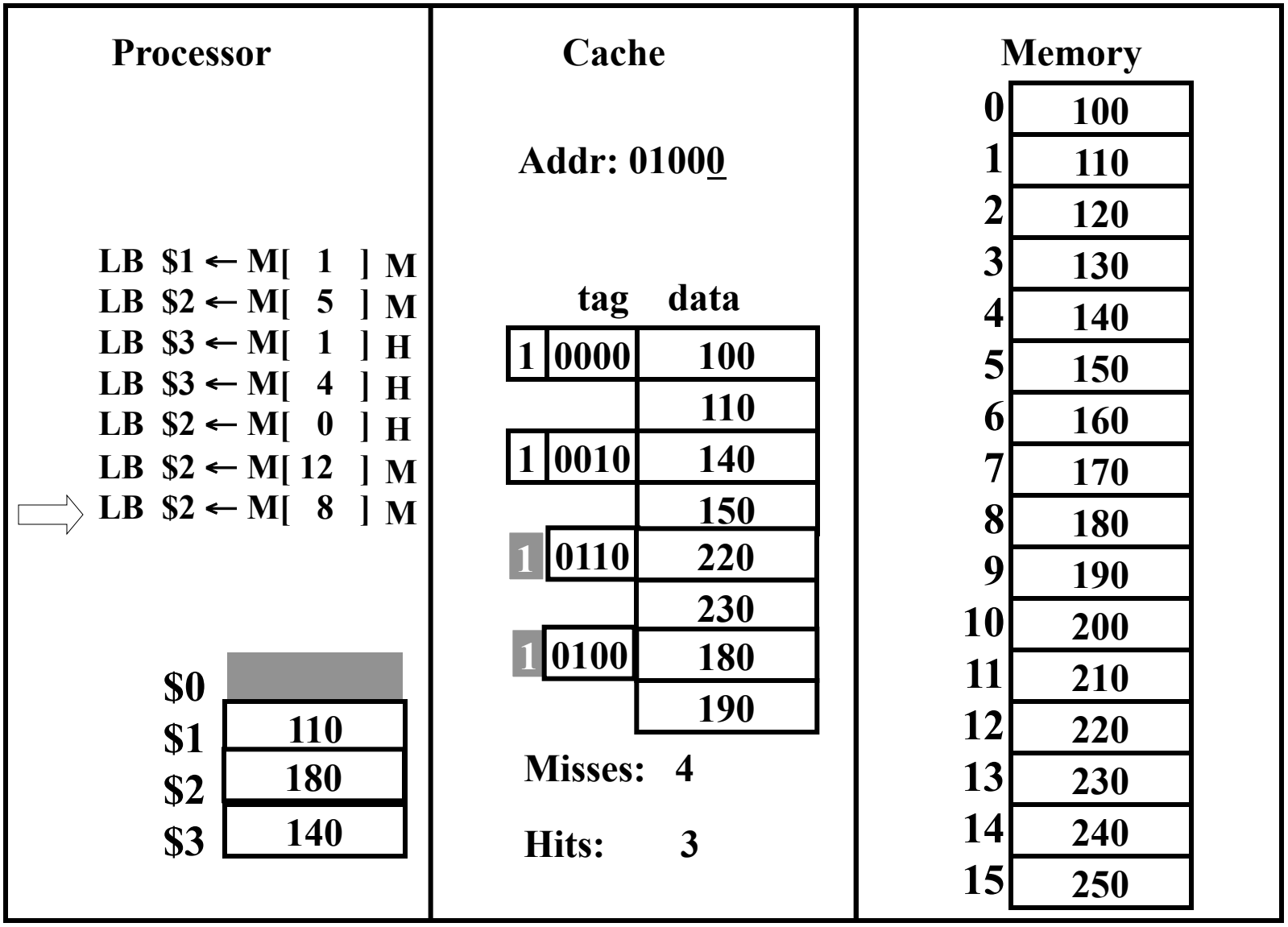
Using **byte addresses** in this example! Addr Bus = 5 bits

7th Access



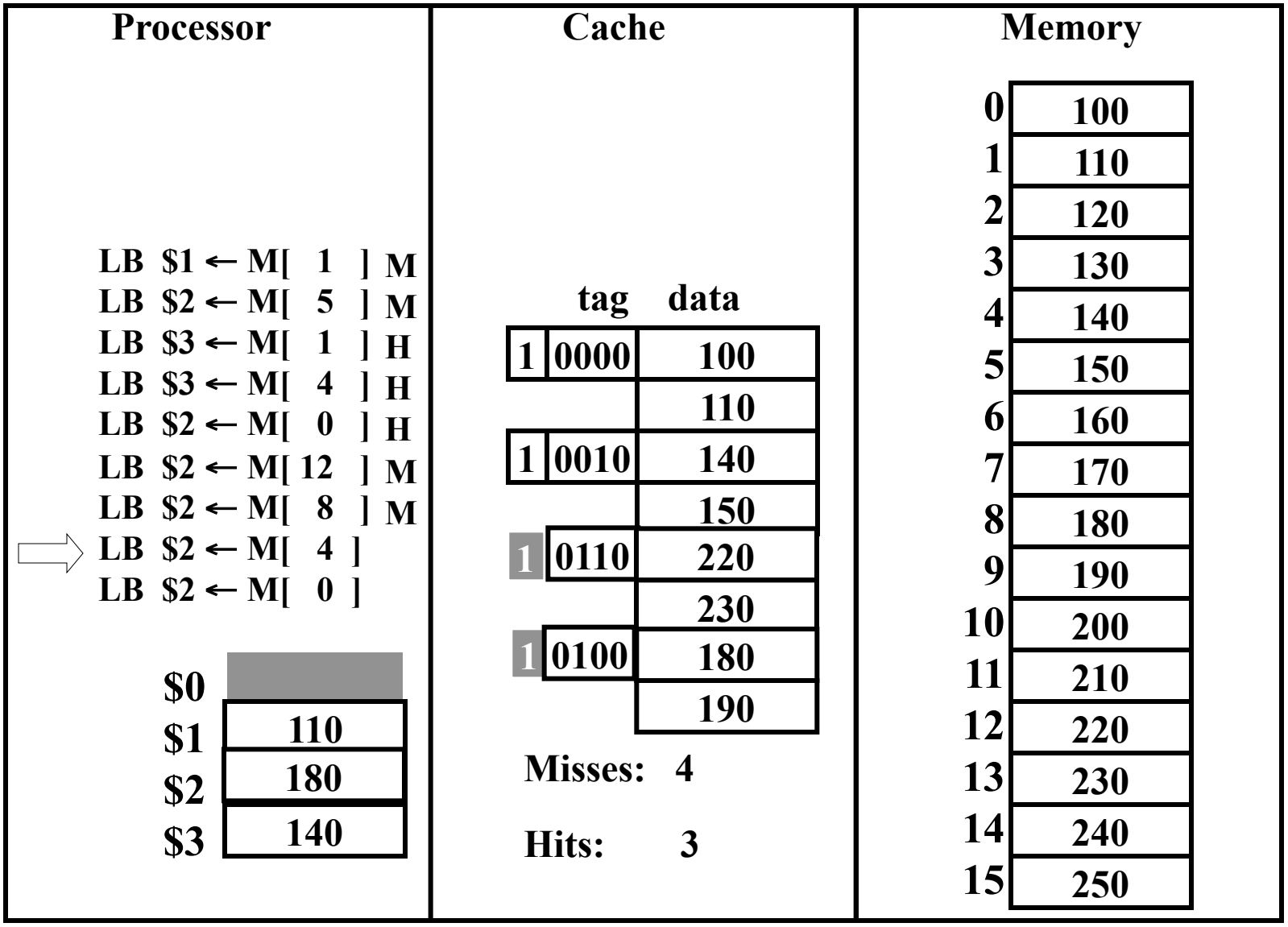
Using **byte addresses** in this example! Addr Bus = 5 bits

7th Access



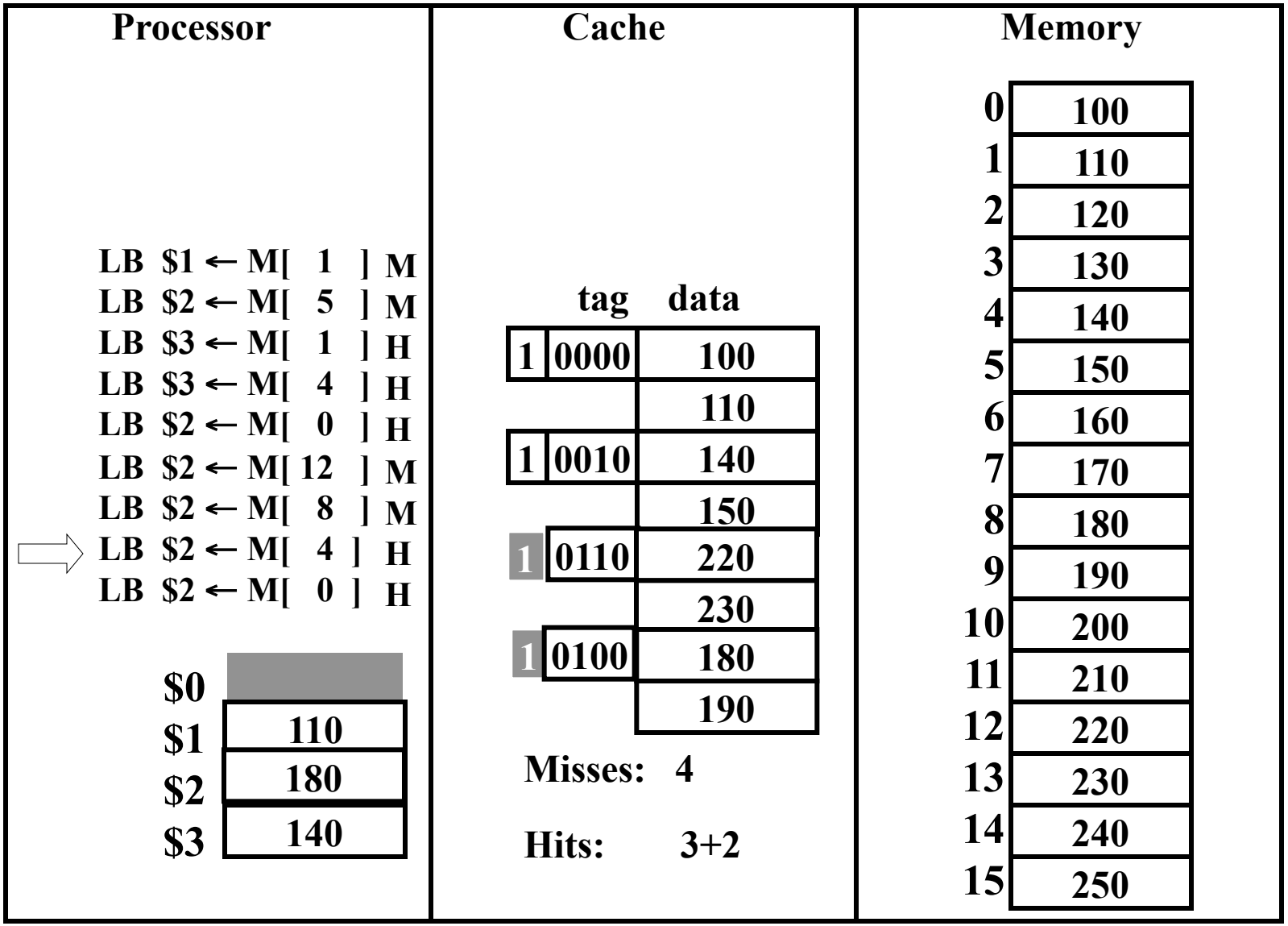
Using **byte addresses** in this example! Addr Bus = 5 bits

8th and 9th Accesses



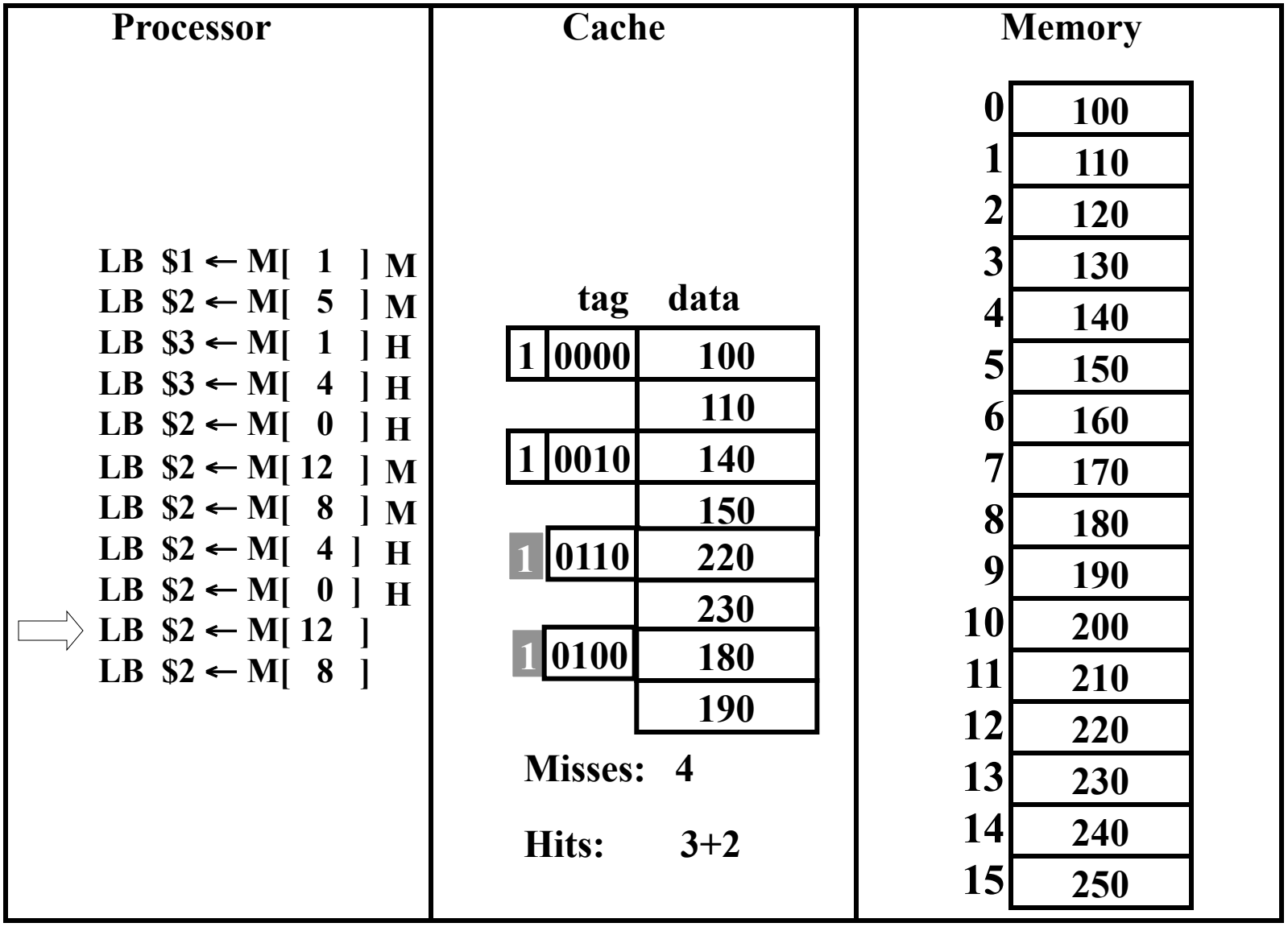
Using **byte addresses** in this example! Addr Bus = 5 bits

8th and 9th Accesses



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

10th and 11th Accesses



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

10th and 11th Accesses

Processor

	LB	$1 ← M[1]	M
	LB	$2 ← M[5]	M
	LB	$3 ← M[1]	H
	LB	$3 ← M[4]	H
	LB	$2 ← M[0]	H
	LB	$2 ← M[12]	M
	LB	$2 ← M[8]	M
	LB	$2 ← M[4]	H
	LB	$2 ← M[0]	H
⇨	LB	$2 ← M[12]	H
	LB	$2 ← M[8]	H

Cache

	tag	data
1	0000	100
		110
1	0010	140
		150
1	0110	220
		230
1	0100	180
		190

Misses: 4

Hits: 3+2+2

Memory

0	100
1	110
2	120
3	130
4	140
5	150
6	160
7	170
8	180
9	190
10	200
11	210
12	220
13	230
14	240
15	250

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

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

Cache Tradeoffs

	Direct Mapped	**Fully Associative**
Tag Size	Smaller	Larger
SRAM Overhead	Less	More
Controller Logic	Less	More
Speed	Faster	Slower
Price	Less	More
Scalability	Very	Not Very
# of conflict misses	Lots	Zero
Hit Rate	Low	High
Pathological Cases	Common	?

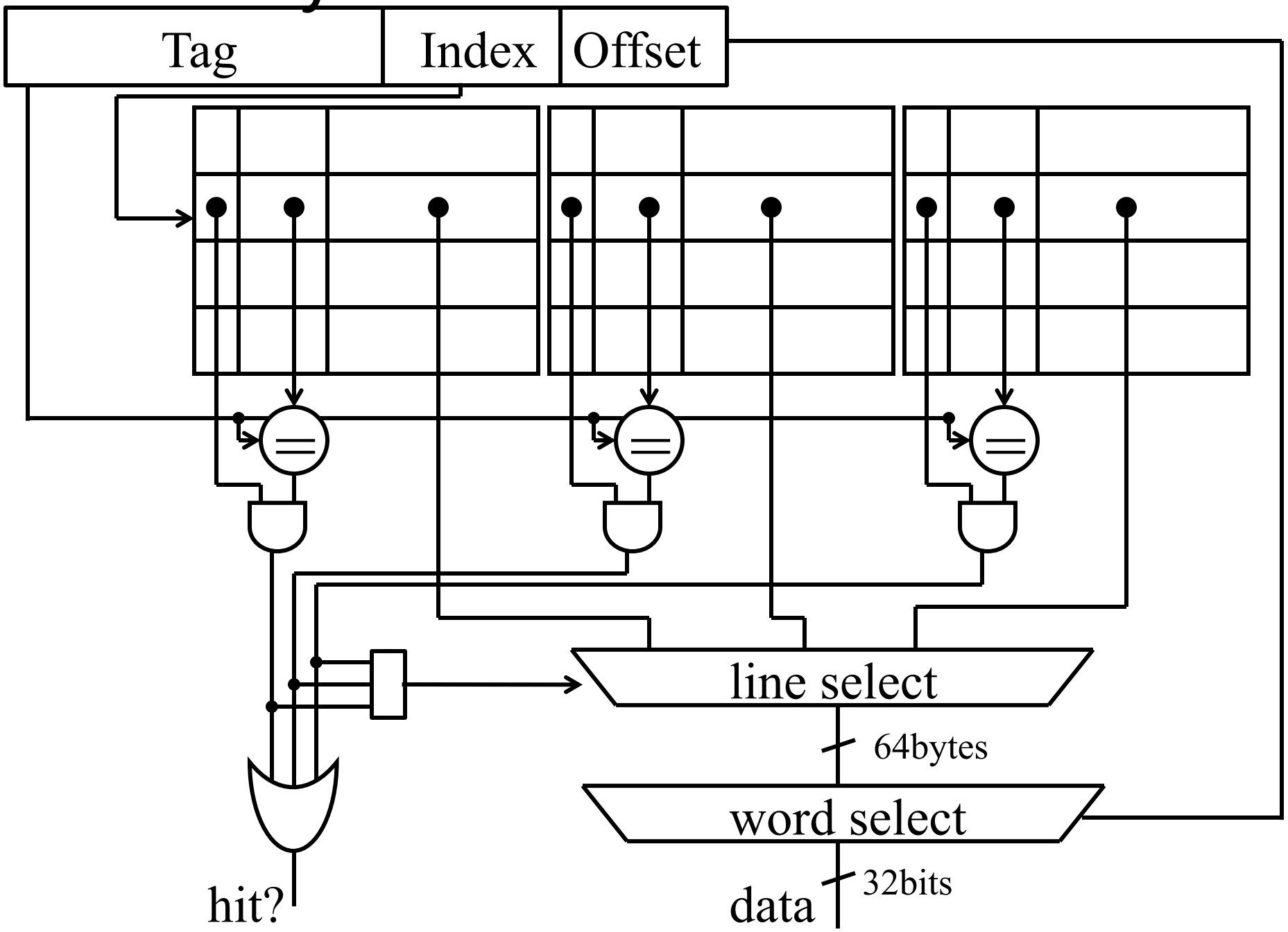
Hybrid Approach

- Set-associative cache!

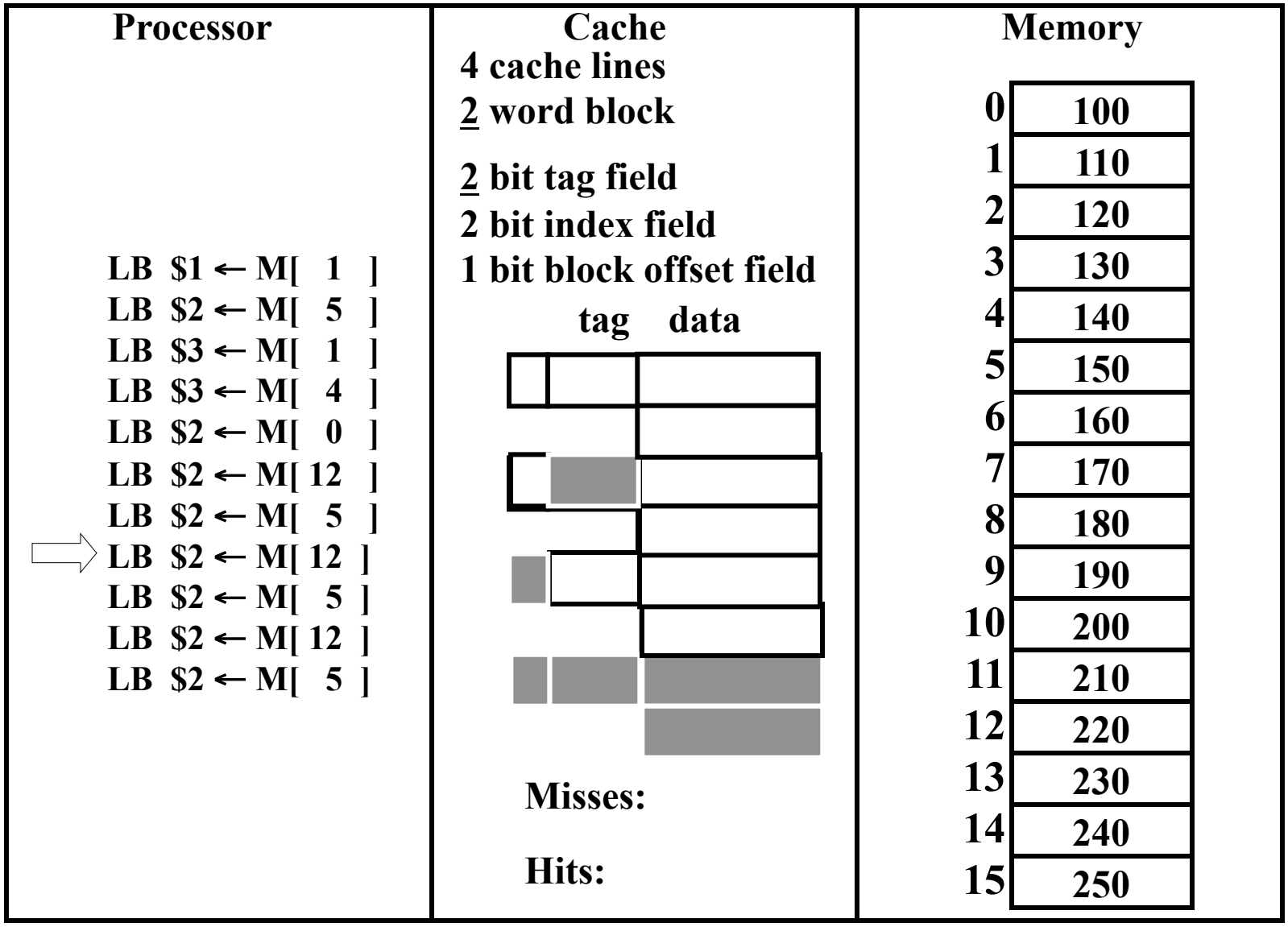
- Like a direct-mapped cache
  - Index into a location
  - Fast

- Like a fully-associative cache
  - Can store multiple entries
    - decreases thrashing in cache
  - Search in each element

3-Way Set Associative Cache

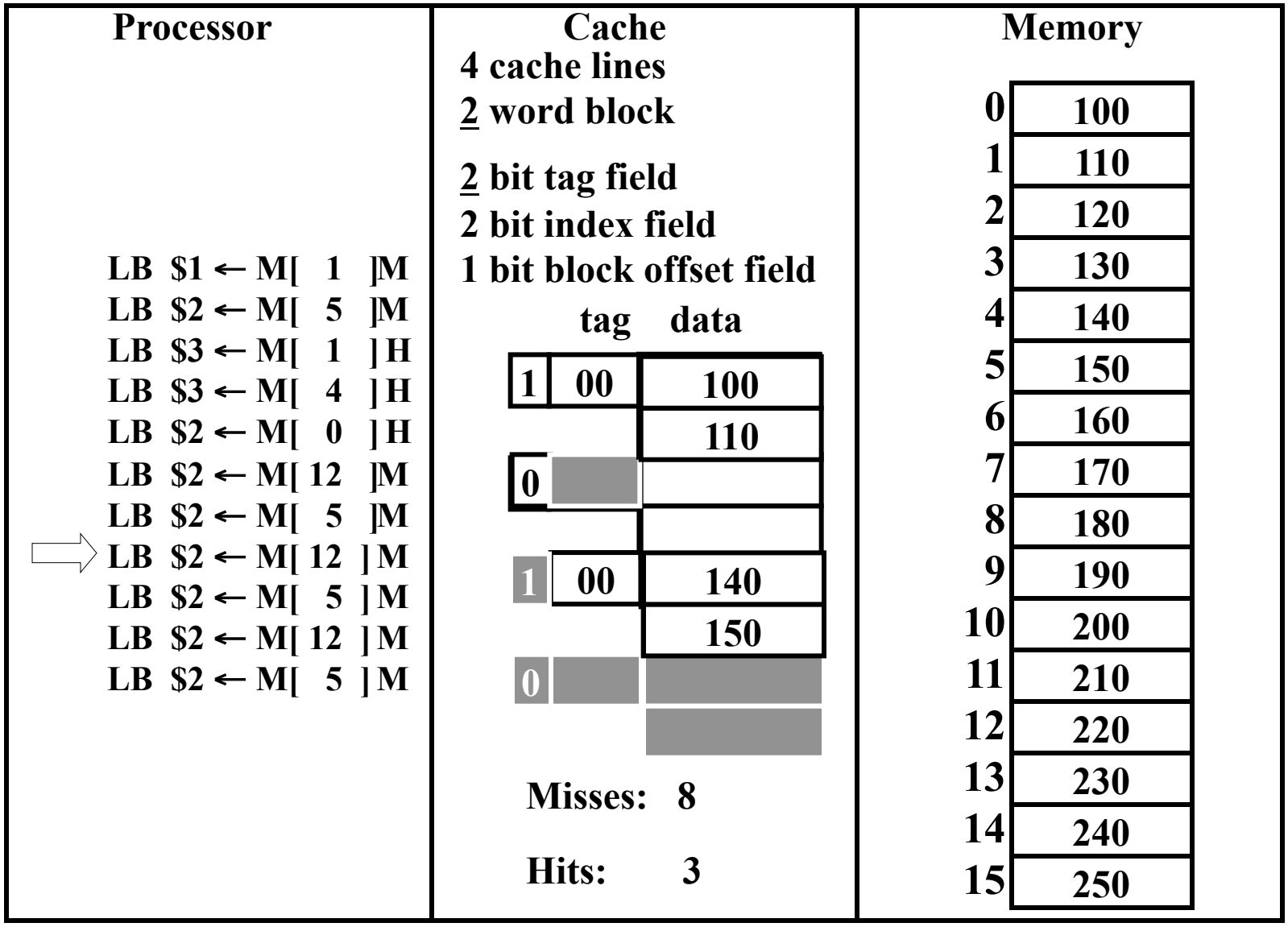


Comparison: Direct Mapped



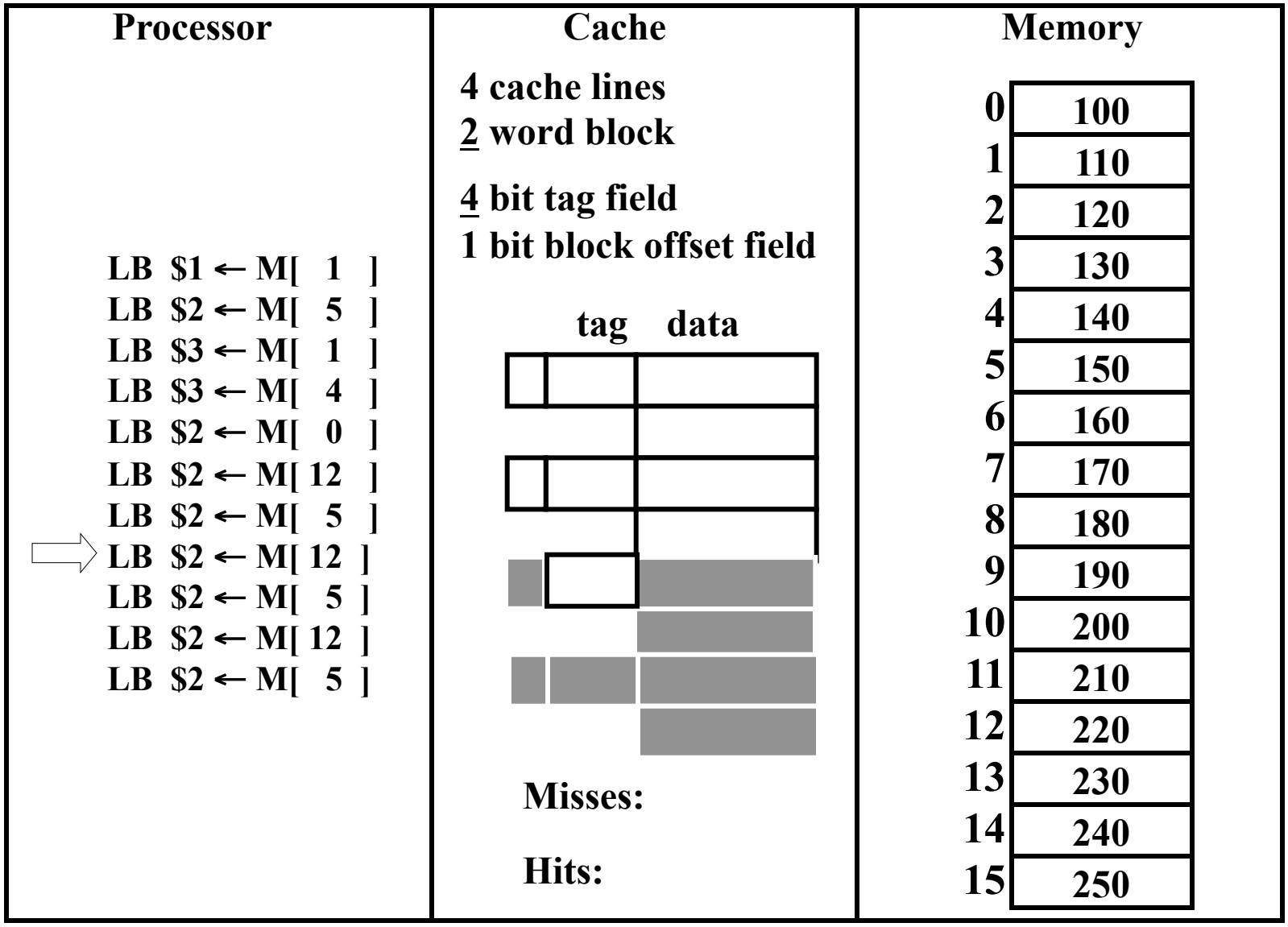
Using **byte addresses** in this example! Addr Bus = 5 bits

Comparison: Direct Mapped



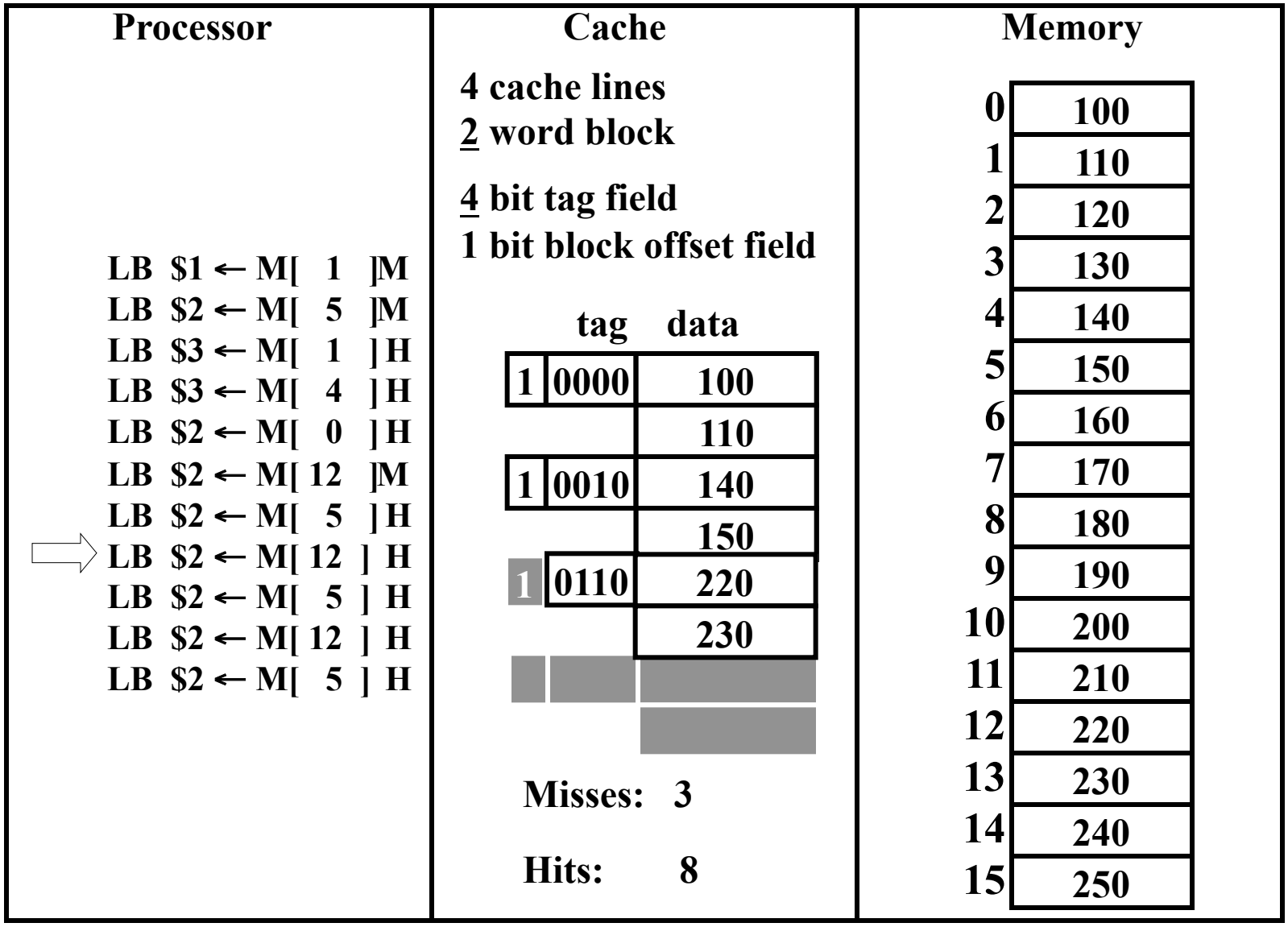
Using **byte addresses** in this example! Addr Bus = 5 bits

Comparison: Fully Associative



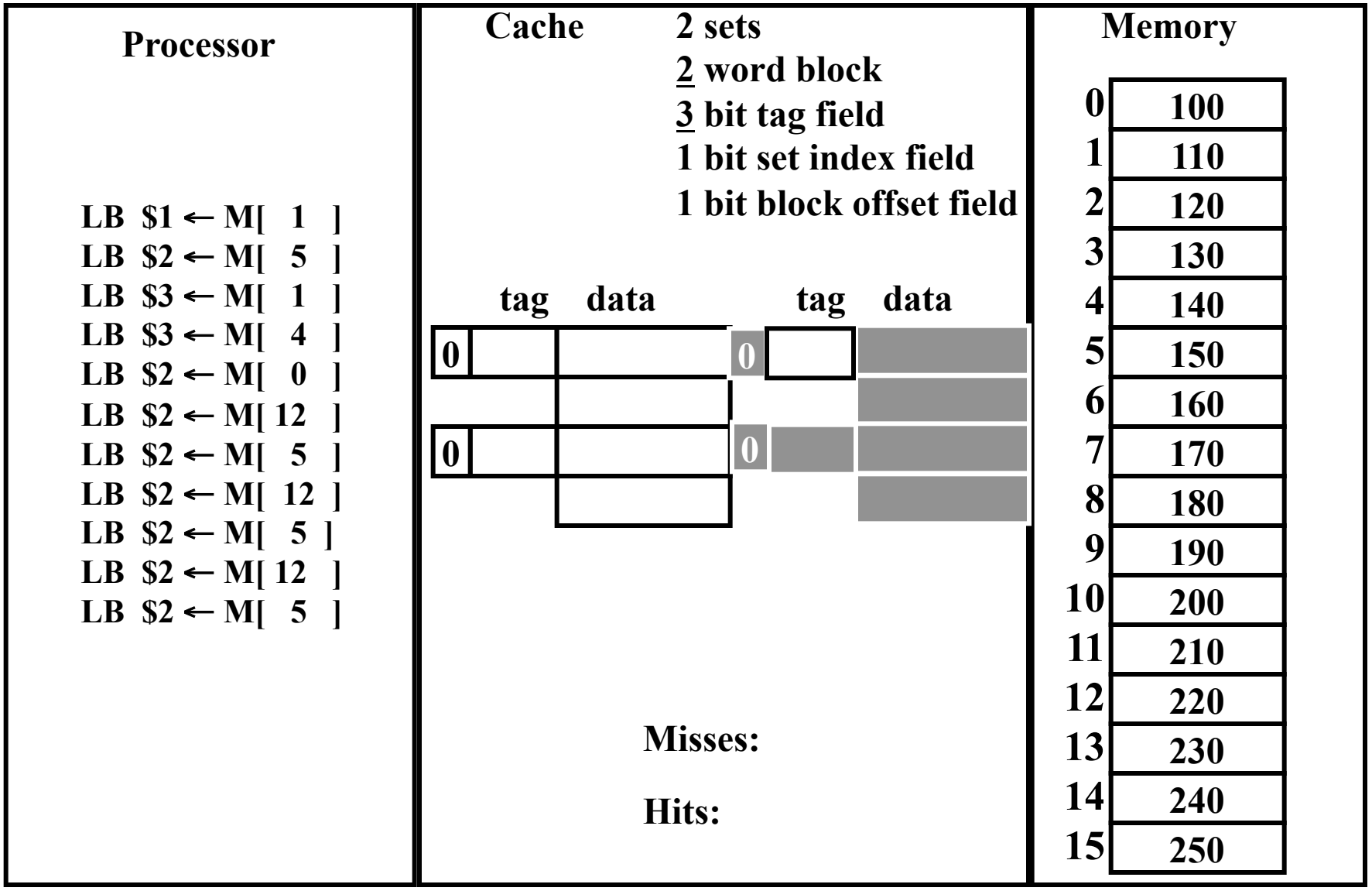
Using **byte addresses** in this example! Addr Bus = 5 bits

Comparison: Fully Associative



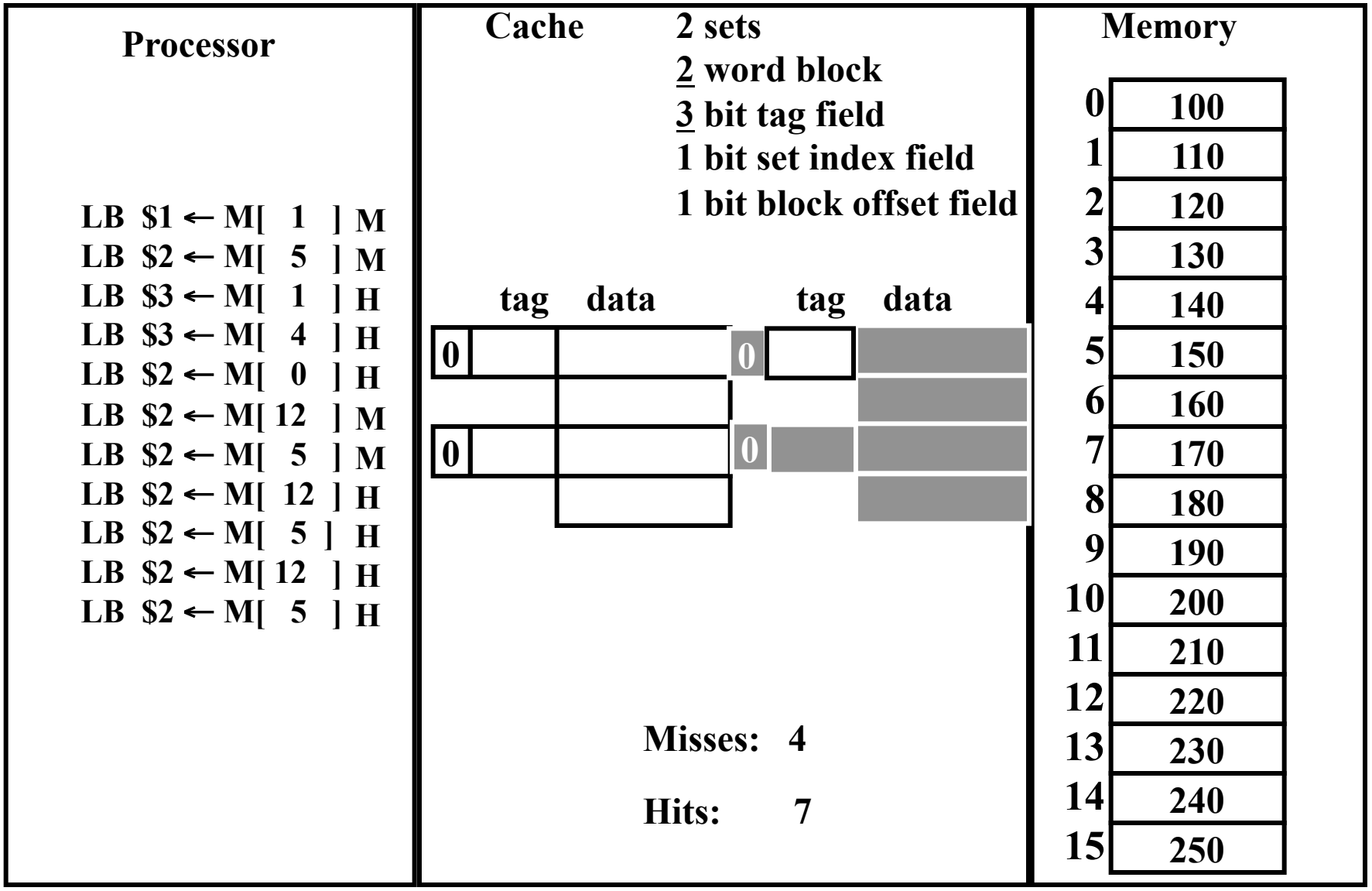
Using **byte addresses** in this example! Addr Bus = 5 bits

Comparison: 2-Way Set Associative



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

Comparison: 2-Way Set Associative



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

Summary

- Caching assumptions
  - small working set: 90/10 rule
  - can predict future: spatial & temporal locality
- Benefits
  - big & fast memory built from (big & slow) + (small & fast)
- Tradeoffs: associativity, line size, hit cost, miss penalty, hit rate
  - Fully Associative → higher hit cost, higher hit rate
  - Larger block size → lower hit cost, higher miss penalty

Remaining Issues

To Do:

- Evicting cache lines
- Picking cache parameters
- Writing using the cache