Caches

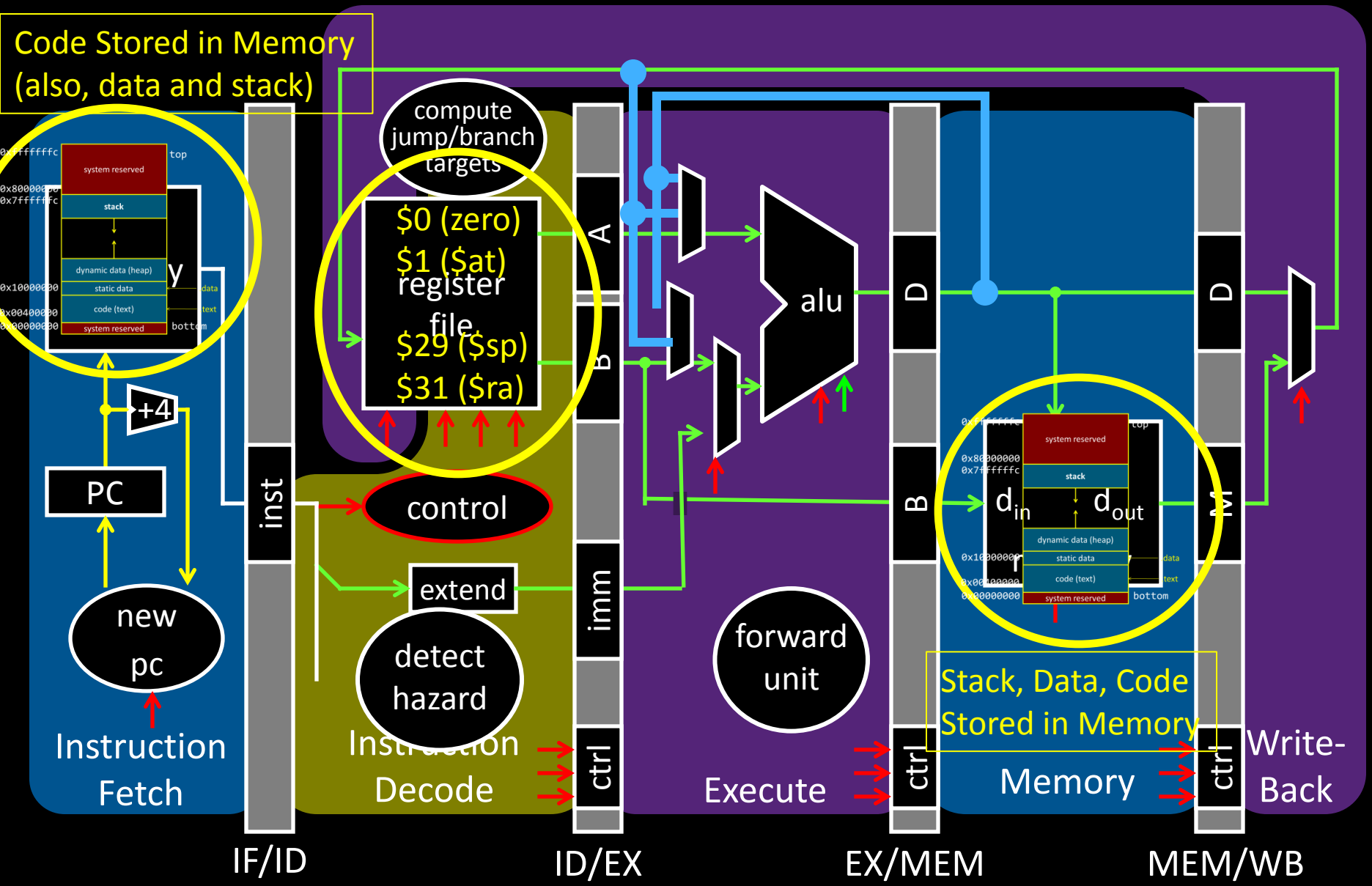
Hakim Weatherspoon
CS 3410, Spring 2013
Computer Science
Cornell University

See P&H 5.1, 5.2 (except writes)

What will you do over Spring Break?

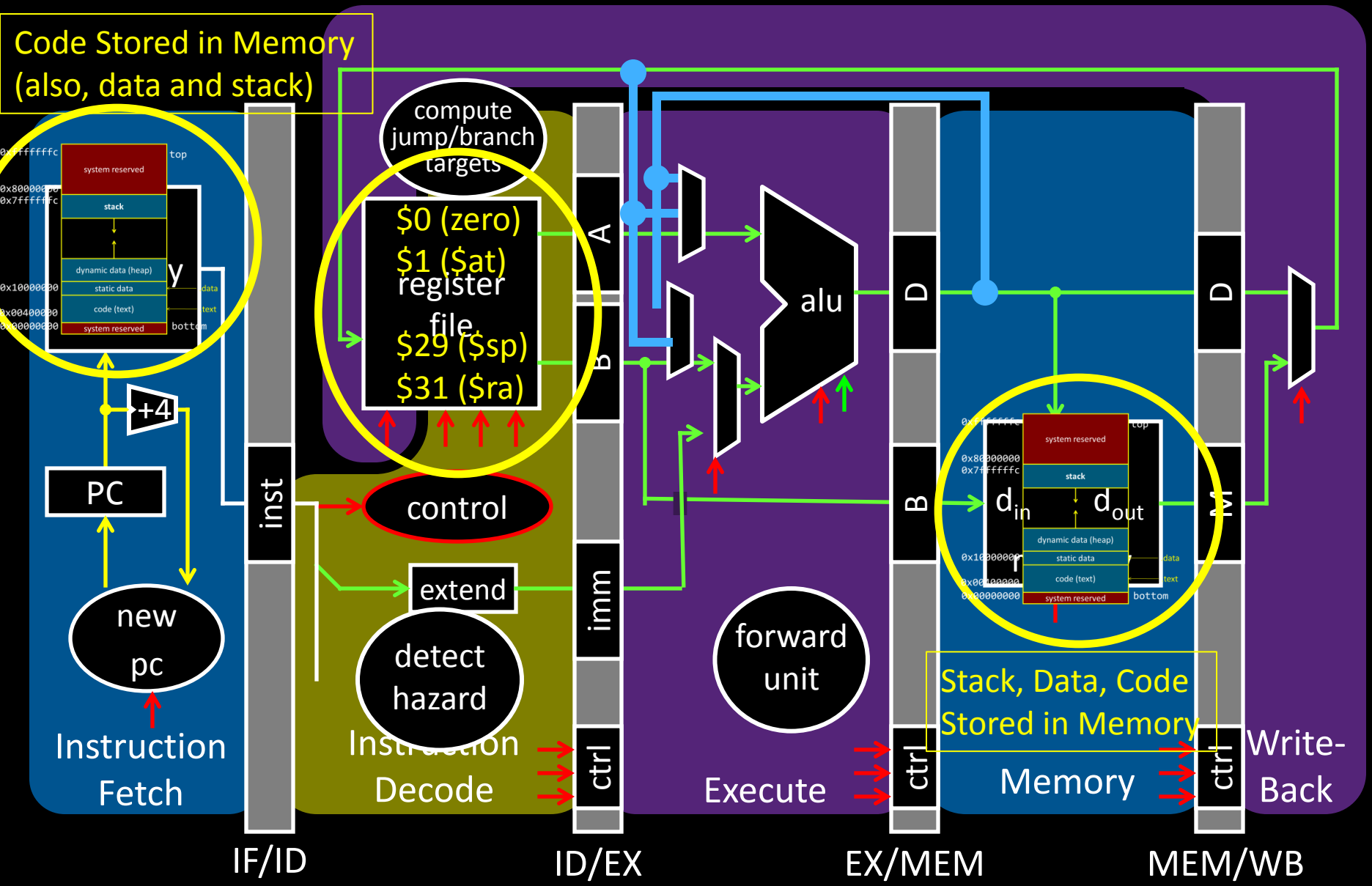
A) Relax
B) Head home
C) Head to a warm destination
D) Stay in (frigid) Ithaca
E) Study

Big Picture: Memory



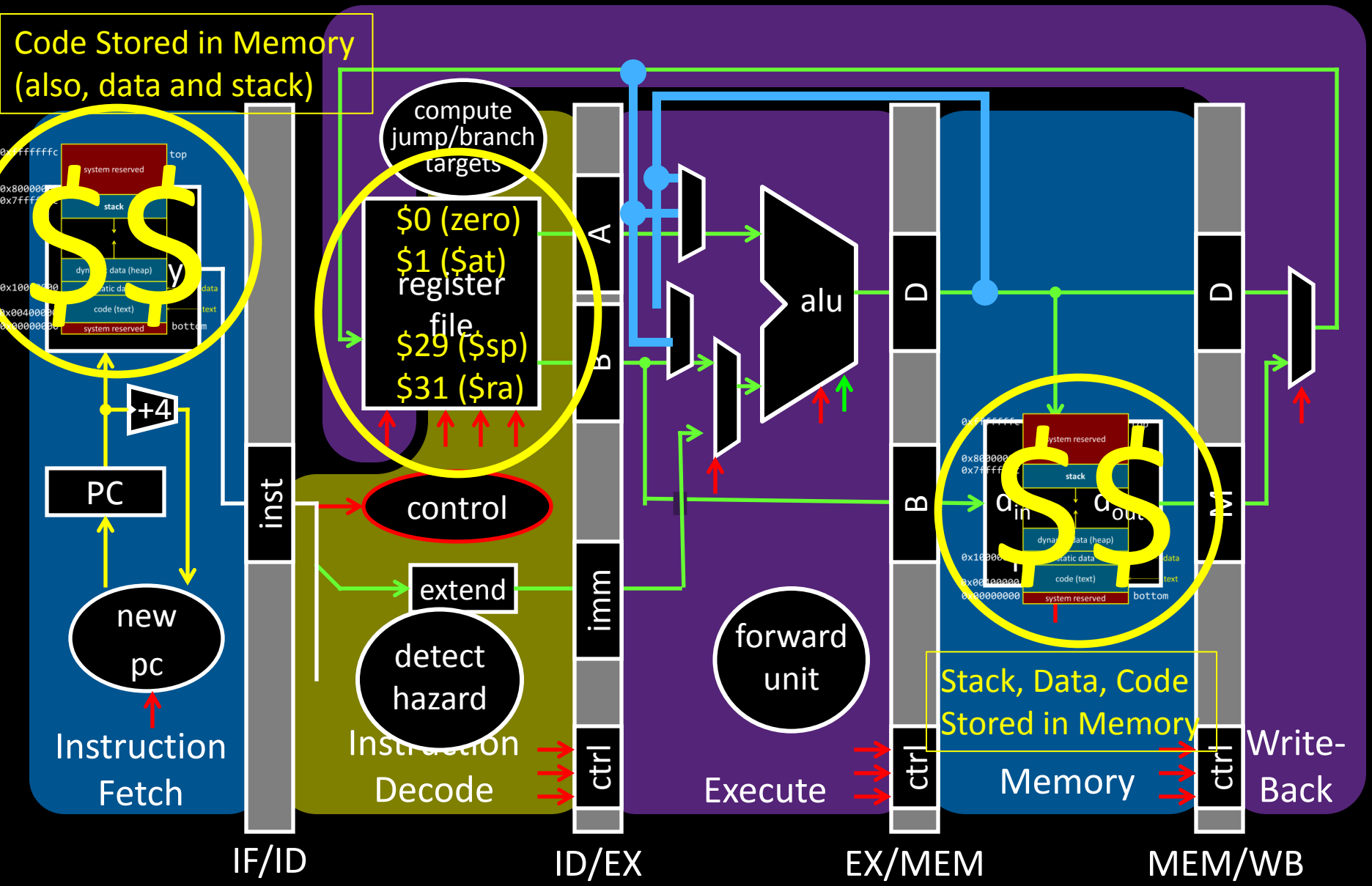
Big Picture: Memory

Memory: big & slow vs Caches: small & fast



Big Picture: Memory

Memory: big & slow vs Caches: small & fast



Goals for Today: caches

Caches vs memory vs tertiary storage

- Tradeoffs: big & slow vs small & fast
- working set: 90/10 rule
- How to predict future: temporal & spacial locality

Examples of caches:

- Direct Mapped
- Fully Associative
- N-way set associative

Caching Questions

- How does a cache work?
- How effective is the cache (hit rate/miss rate)?
- How large is the cache?
- How fast is the cache (AMAT=average memory access time)

Big Picture

How do we make the processor fast,

Given that memory is VEEERRRYYYY SLLOOOWWW!!

Performance

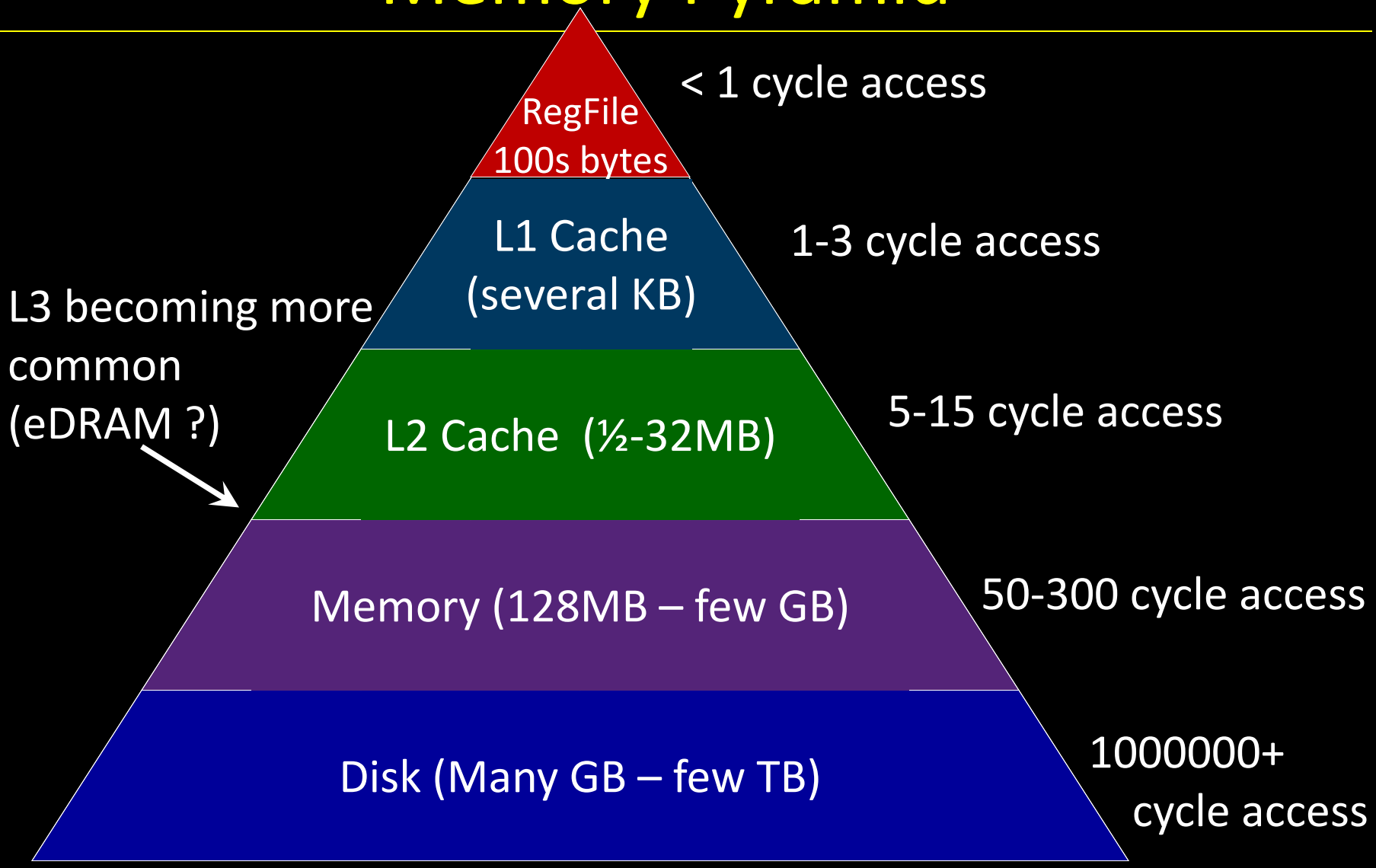
CPU clock rates ~0.2ns – 2ns (5GHz-500MHz)

Technology	Capacity	$/GB	Latency
Tape	1 TB	$.17	100s of seconds
Disk	4 TB	$.03	Millions of cycles (ms)
SSD (Flash)	512 GB	$2	Thousands of cycles (us)
DRAM	8 GB	$10	50-300 cycles (10s of ns)
SRAM off-chip	32 MB	$4000	5-15 cycles (few ns)
SRAM on-chip	256 KB	???	1-3 cycles (ns)

Others: eDRAM aka 1T SRAM , FeRAM, CD, DVD, …

Q: Can we create illusion of cheap + large + fast?

Memory Pyramid



These are rough numbers: mileage may vary for latest/greatest

Caches usually made of SRAM (or eDRAM)

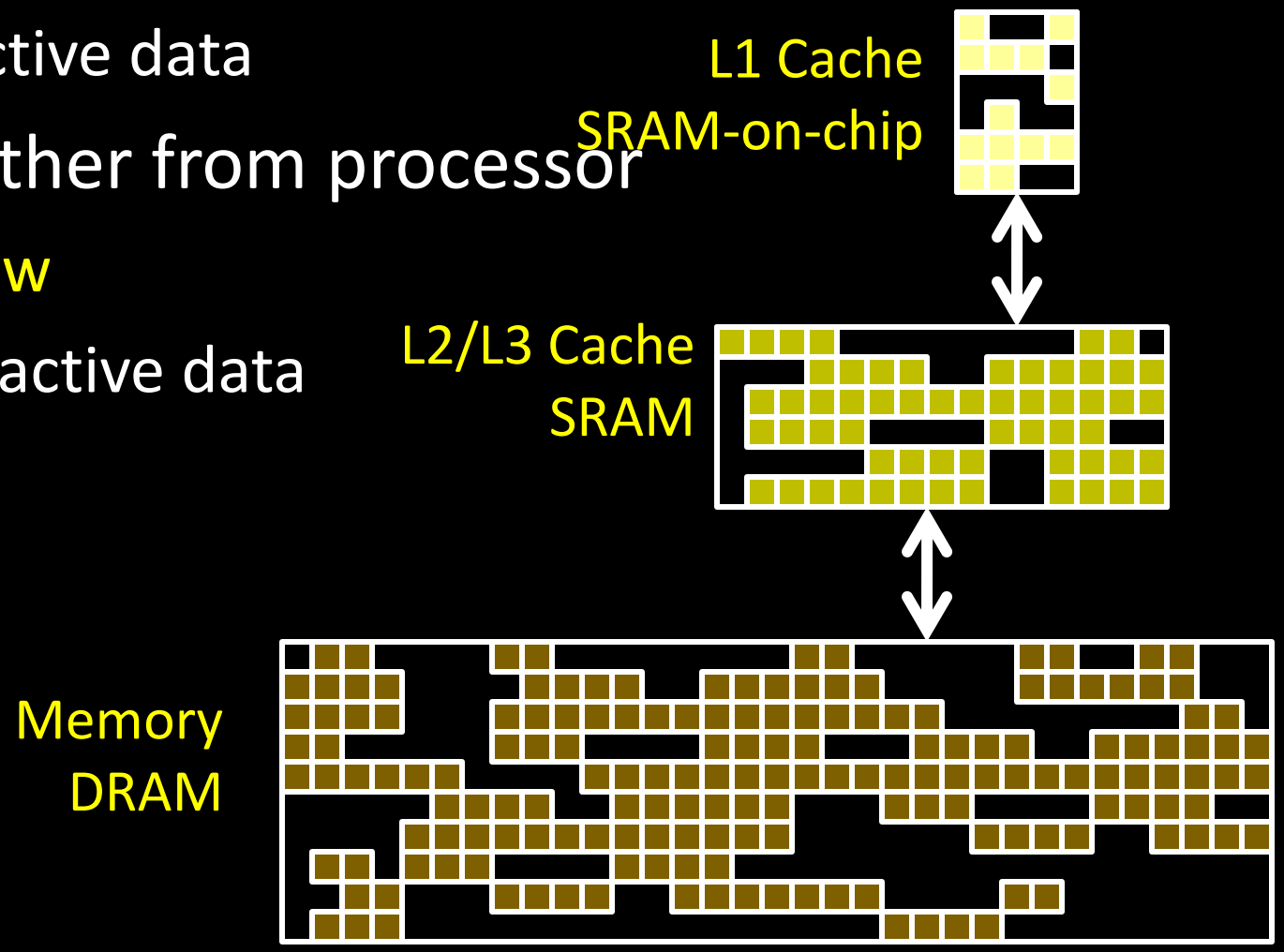
Memory Hierarchy

Memory closer to processor

- small & fast
- stores active data

Memory farther from processor

- big & slow
- stores inactive data



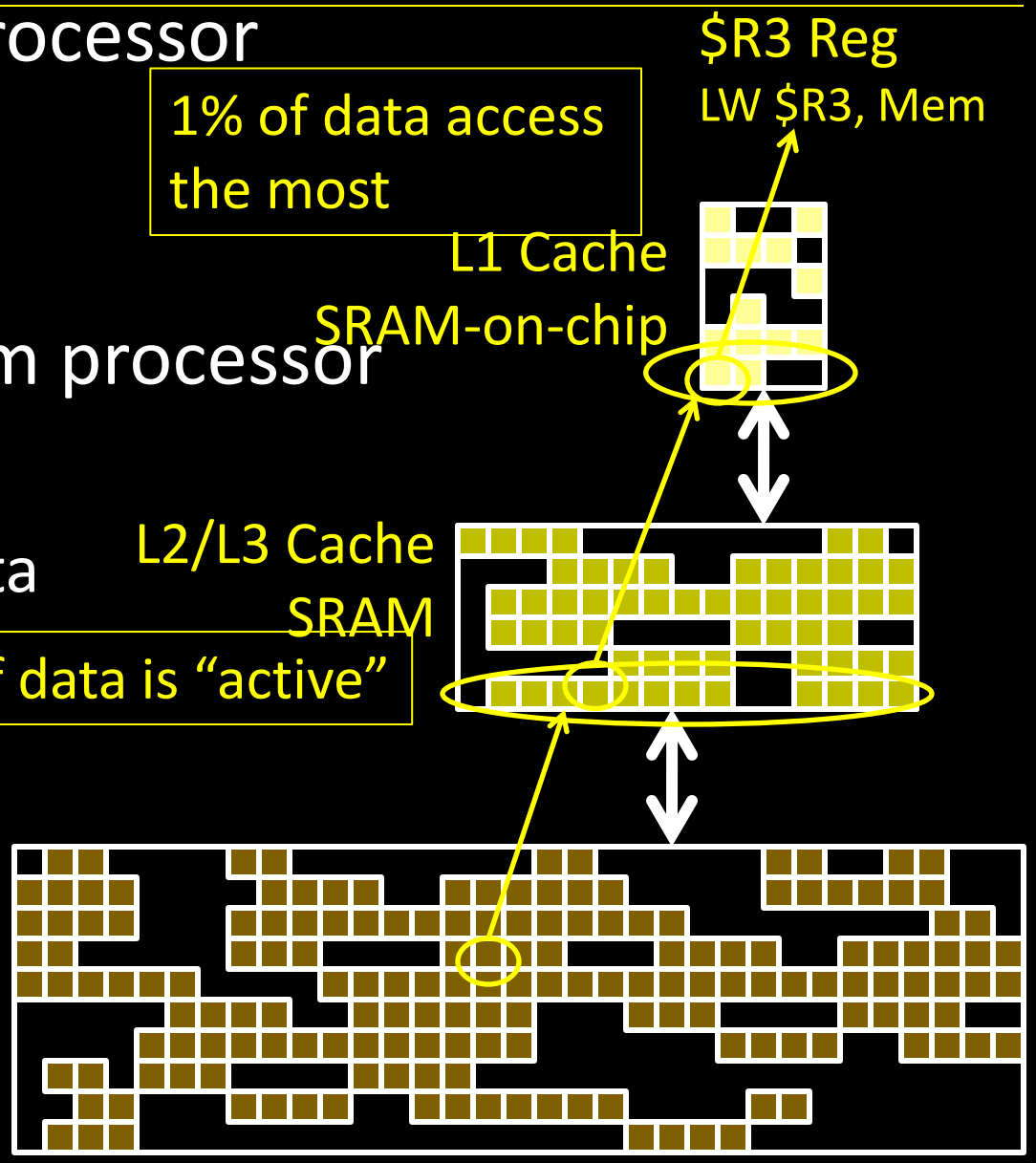
Memory Hierarchy

Memory closer to processor

- small & fast
- stores active data

Memory farther from processor

- big & slow
- stores inactive data



Memory Hierarchy

Insight for Caches

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

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

- Exploit temporal locality:
  - Put recently accessed Mem[x] higher in memory hierarchy since it will likely be accessed again soon

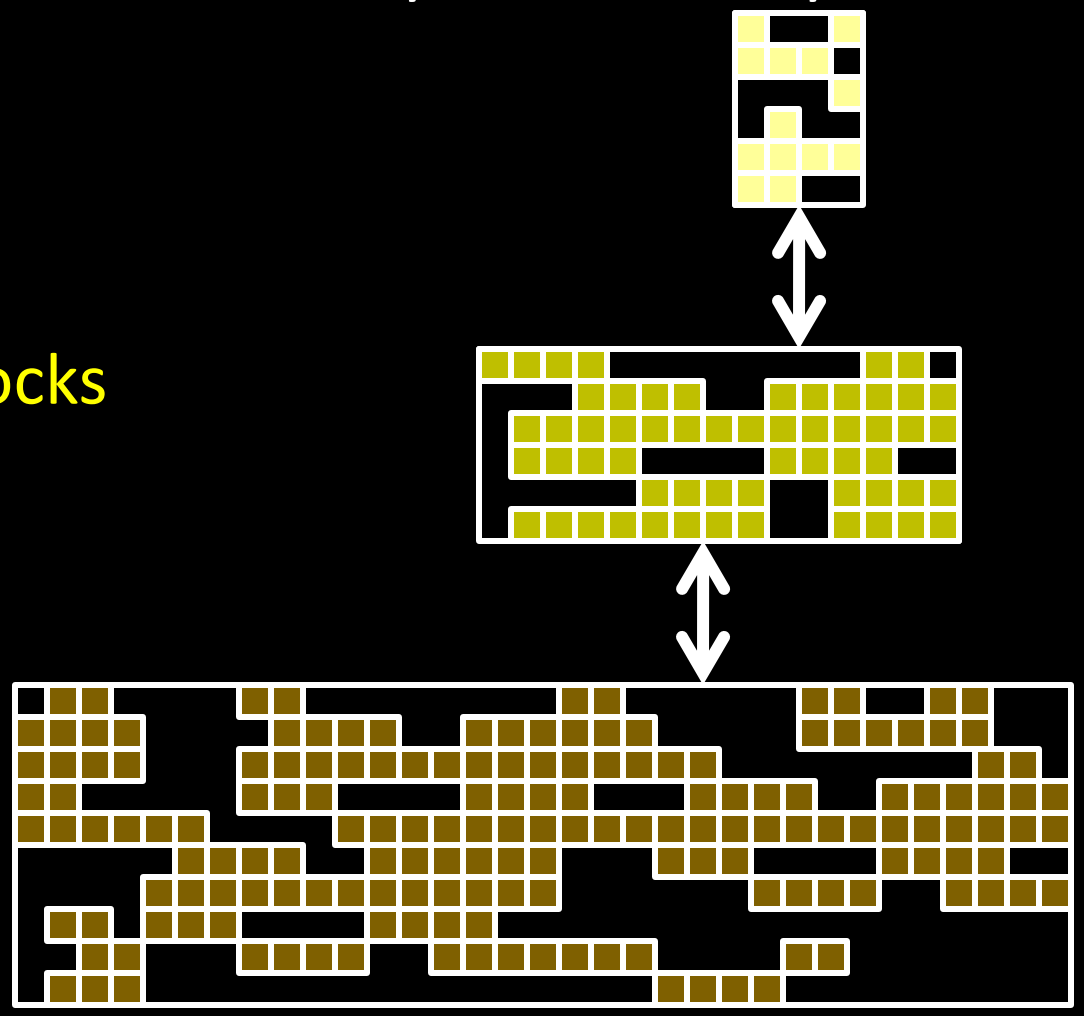
... then Mem[$x \pm \varepsilon$] is likely to be accessed *soon*

- Exploit spatial locality:
  - Put entire block containing Mem[x] and surrounding addresses higher in memory hierarchy since nearby address will likely be accessed

Memory Hierarchy

Memory closer to processor is fast but small

- usually stores subset of memory farther away
  - "strictly inclusive"



- Transfer whole blocks (cache lines):

  4kb: disk ↔ ram

  256b: ram ↔ L2

  64b: L2 ↔ L1

Memory Hierarchy

Memory trace

```
0x7c9a2b18
0x7c9a2b19
0x7c9a2b1a
0x7c9a2b1b
0x7c9a2b1c
0x7c9a2b1d
0x7c9a2b1e
0x7c9a2b1f
0x7c9a2b20
0x7c9a2b21
0x7c9a2b22
0x7c9a2b23
0x7c9a2b28
0x7c9a2b2c
0x0040030c
0x00400310
0x7c9a2b04
0x00400314
0x7c9a2b00
0x00400318
0x0040031c
...
```

```
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");
   }
}
```

Temporal Locality

Spatial Locality

Cache Lookups (Read)

Processor tries to access Mem[x]

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

- Yes: cache hit
  - return requested data from cache line
- No: cache miss
  - read block from memory (or lower level cache)
  - (evict an existing cache line to make room)
  - place new block in cache
  - return requested data
  - → and stall the pipeline while all of this happens

Three common designs

A given data block can be placed…

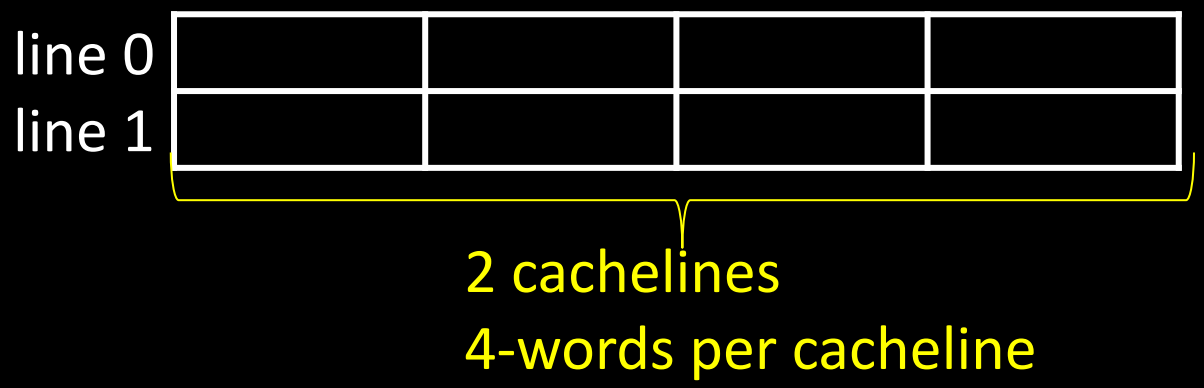
- … in exactly one cache line → Direct Mapped
- … in any cache line → Fully Associative
- … in a small set of cache lines → Set Associative

Direct Mapped Cache

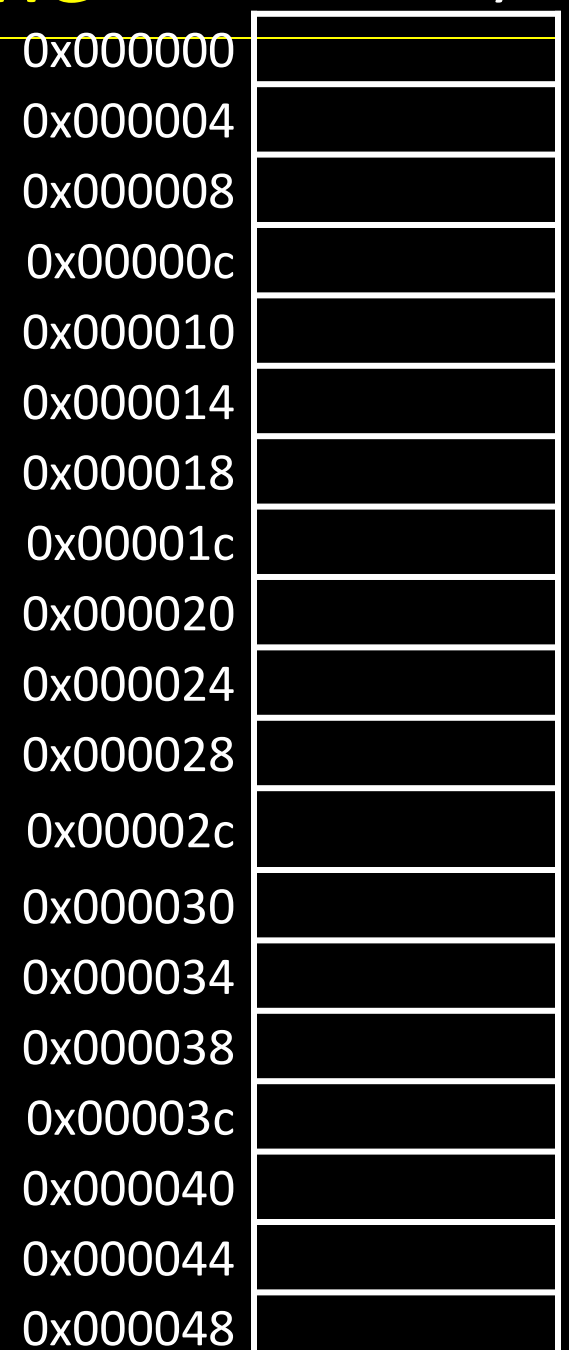
Direct Mapped Cache

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

Cache



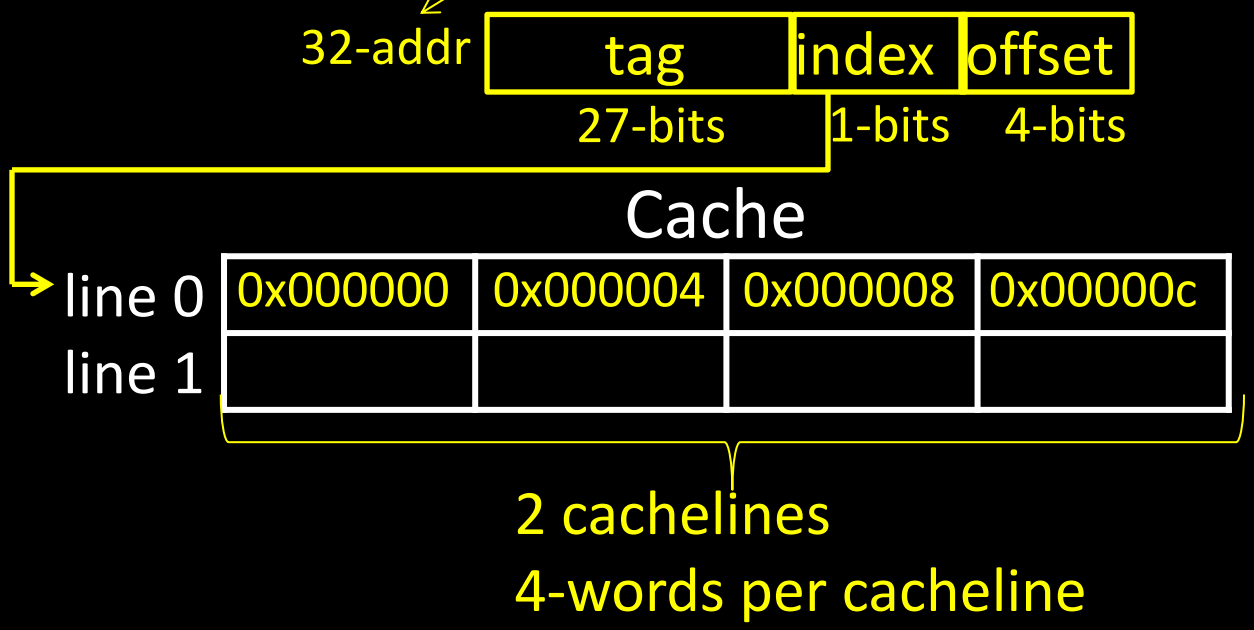
Memory

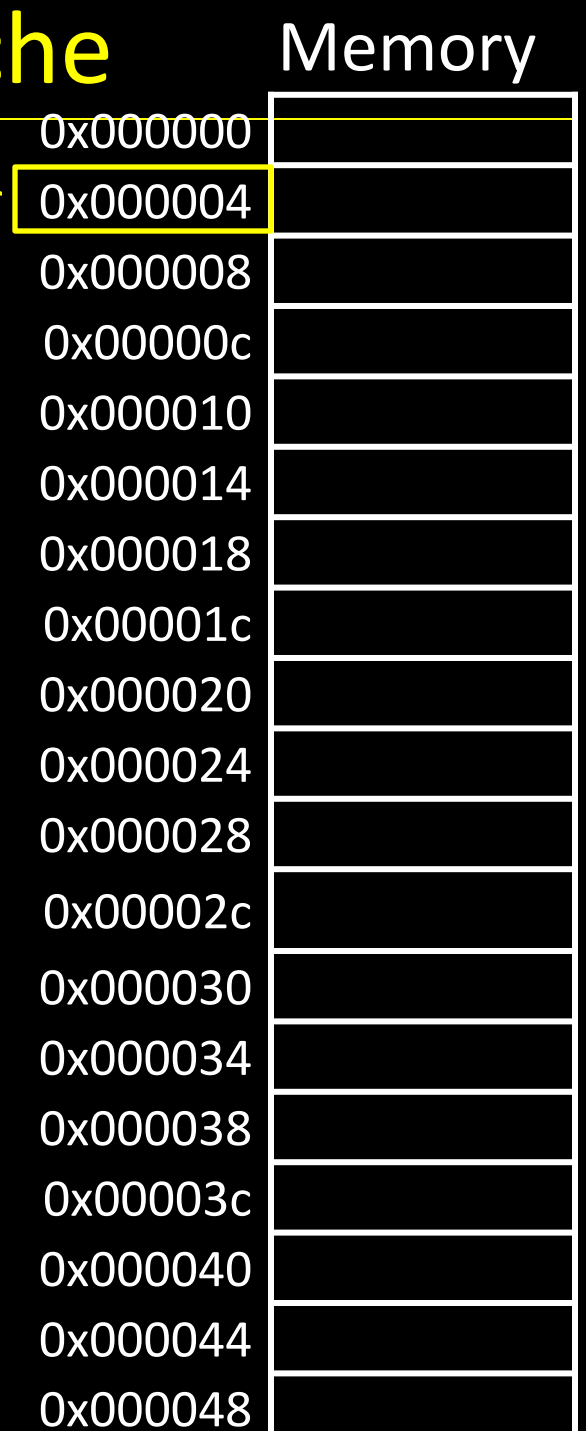


Direct Mapped Cache

Direct Mapped Cache

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





Direct Mapped Cache

Direct Mapped Cache

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

32-addr	tag	index	offset
	27-bits	1-bits	4-bits

Cache

line 0	0x000000	0x000004	0x000008	0x00000c
line 1				

2 cachelines
4-words per cacheline

Memory

	0x000000	
line 0	0x000004	
	0x000008	
	0x00000c	
	0x000010	
line 1	0x000014	
	0x000018	
	0x00001c	
	0x000020	
	0x000024	
	0x000028	
	0x00002c	
	0x000030	
	0x000034	
	0x000038	
	0x00003c	
	0x000040	
	0x000044	
	0x000048	

Direct Mapped Cache

Direct Mapped Cache

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

32-addr	tag	index	offset
	27-bits	2-bits	3-bits

Cache

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

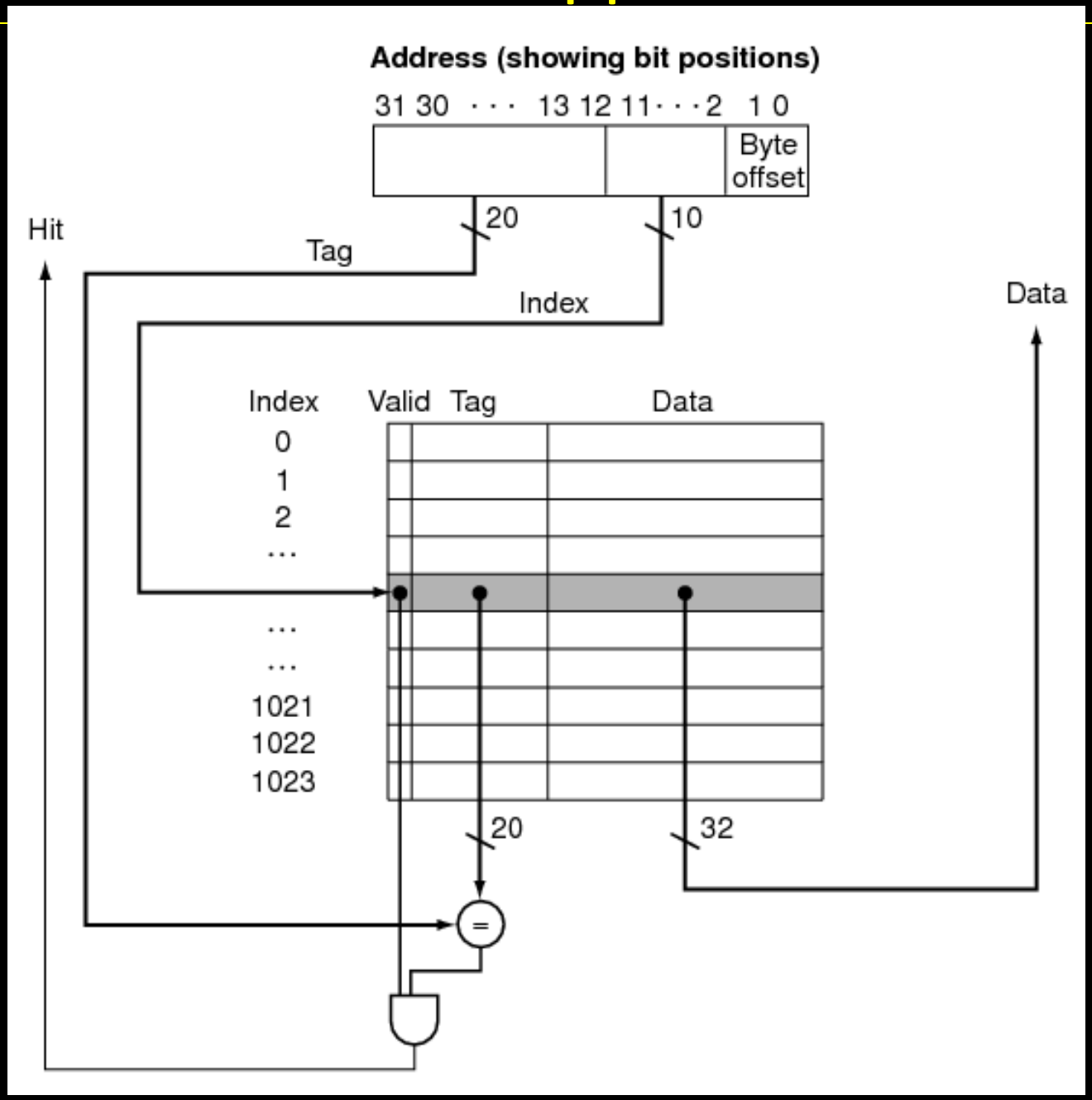
4 cachelines

2-words per cacheline

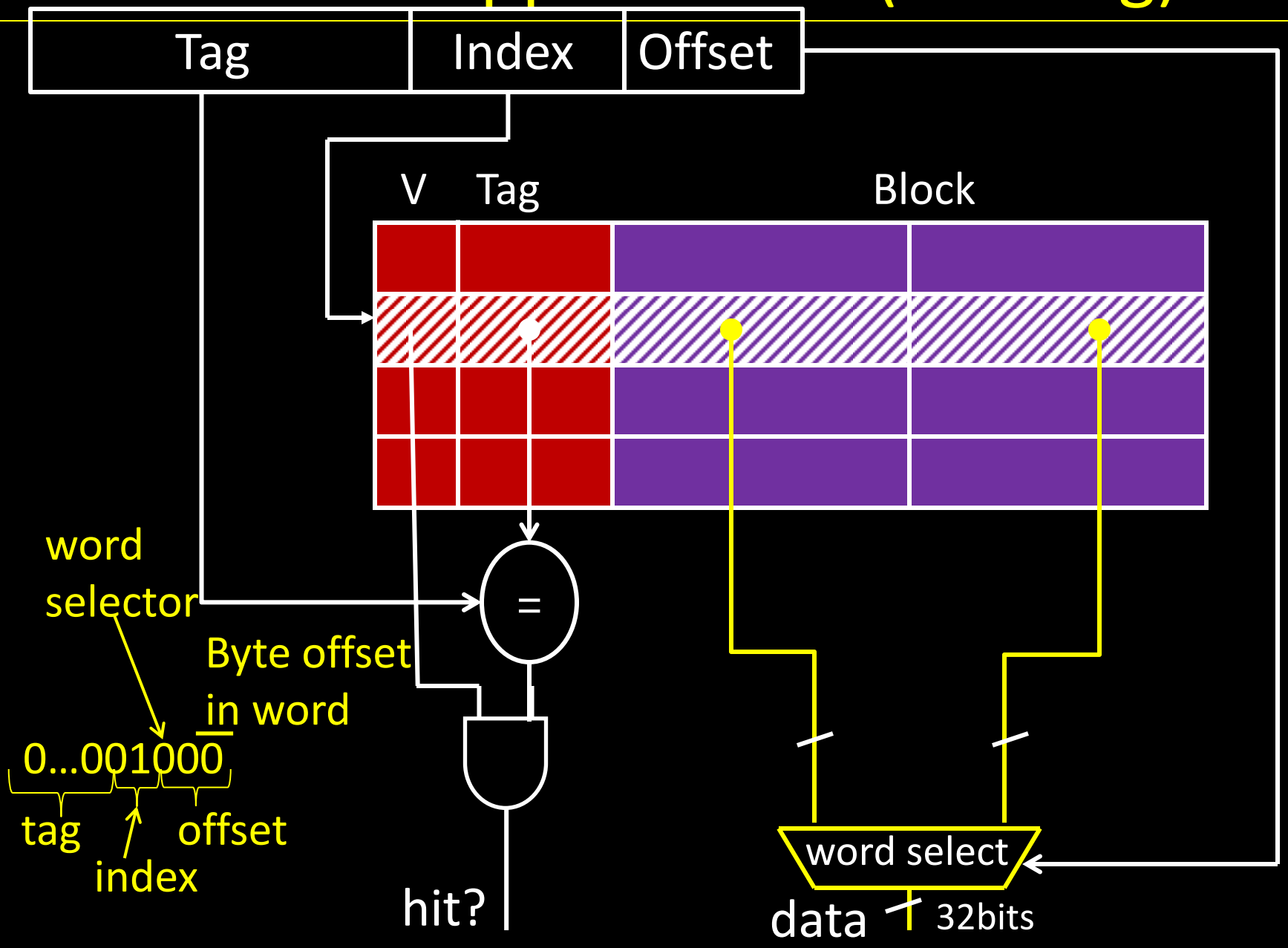
Memory

line 0	0x000000	
	0x000004	
line 1	0x000008	
	0x00000c	
line 2	0x000010	
	0x000014	
line 3	0x000018	
	0x00001c	
line 0	0x000020	
	0x000024	
line 1	0x000028	
	0x00002c	
line 2	0x000030	
	0x000034	
line 3	0x000038	
	0x00003c	
	0x000040	
	0x000044	
	0x000048	

Direct Mapped Cache

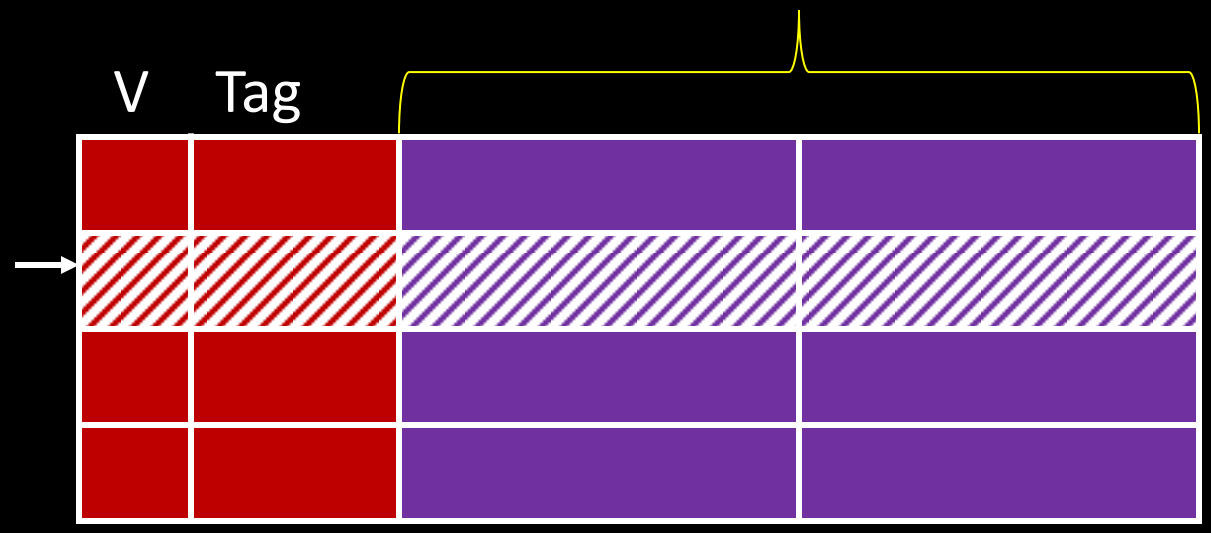


Direct Mapped Cache (Reading)



Direct Mapped Cache (Reading)

Tag	Index	Offset

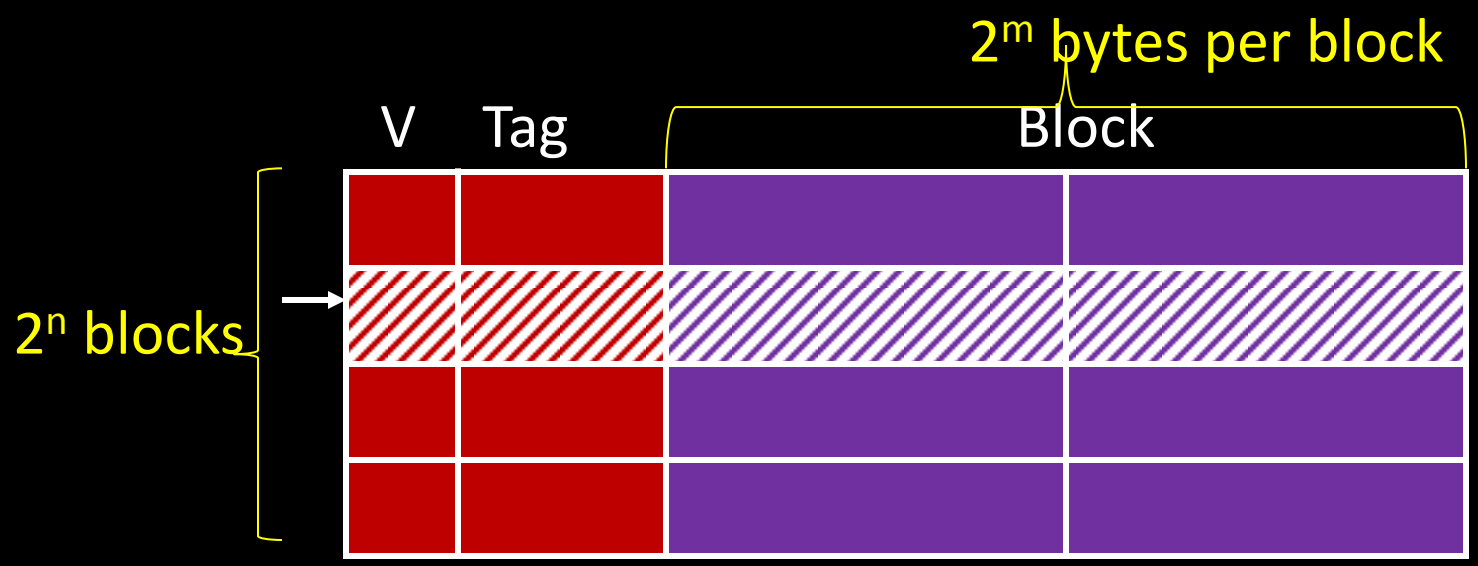


n bit index, *m* bit offset

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

Direct Mapped Cache (Reading)

Tag	Index	Offset



n bit index, *m* bit offset

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

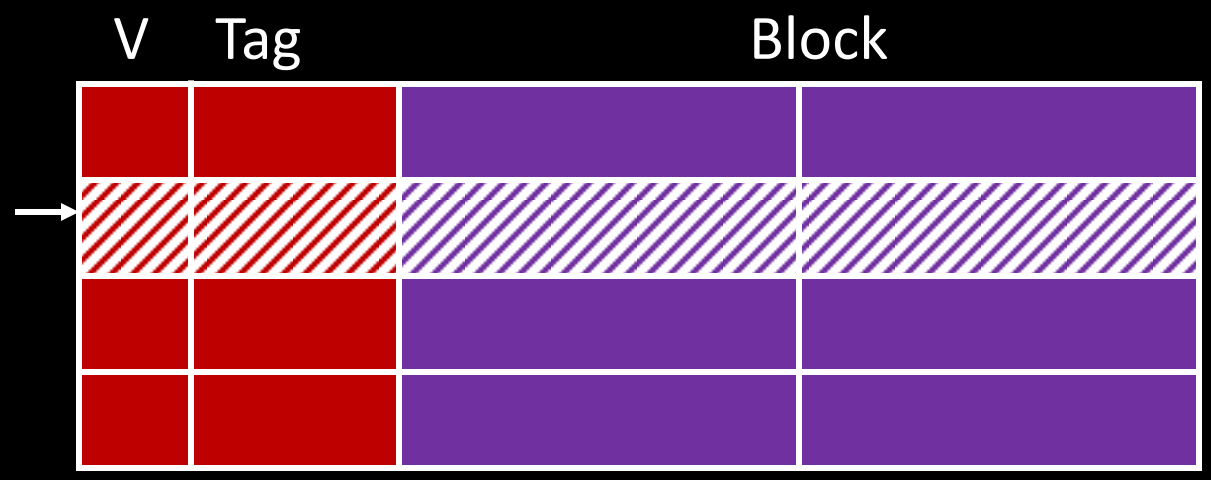
Cache of size 2^n blocks

Block size of 2^m bytes

Cache Size: 2^n bytes per block x 2^n blocks = 2^{n+m} bytes

Direct Mapped Cache (Reading)

Tag	Index	Offset

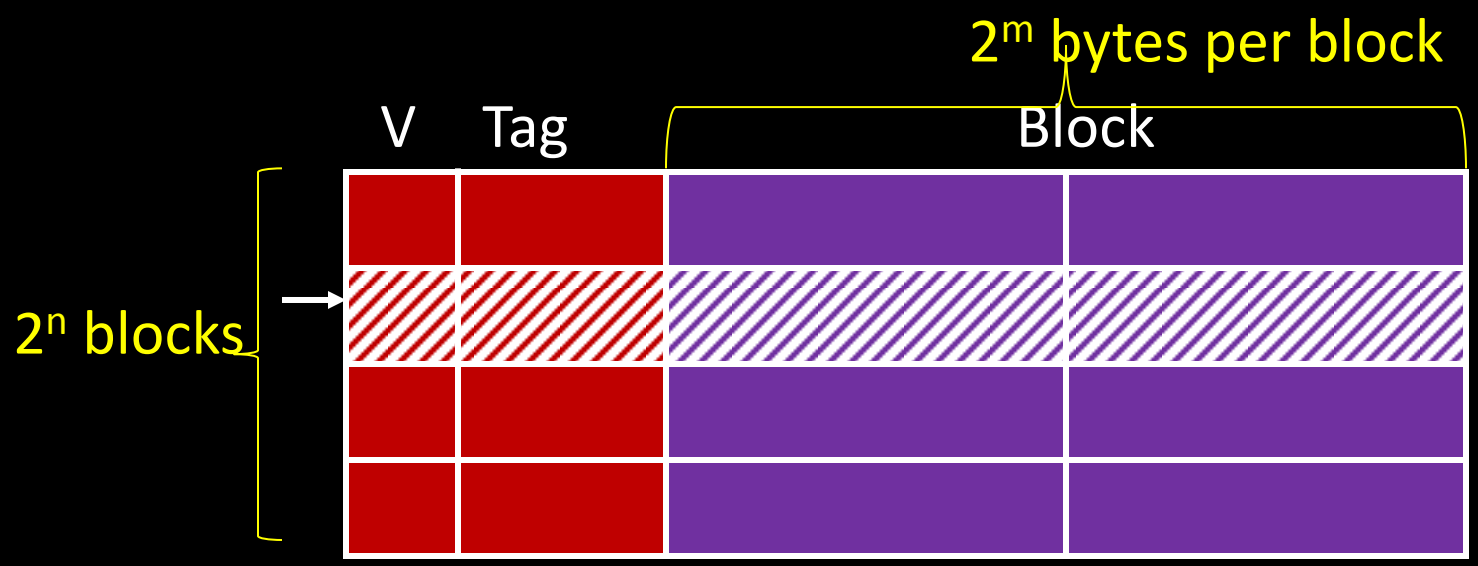


n bit index, *m* bit offset

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

Direct Mapped Cache (Reading)

Tag	Index	Offset



n bit index, *m* bit offset

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

Cache of size 2^n blocks

Block size of 2^m bytes

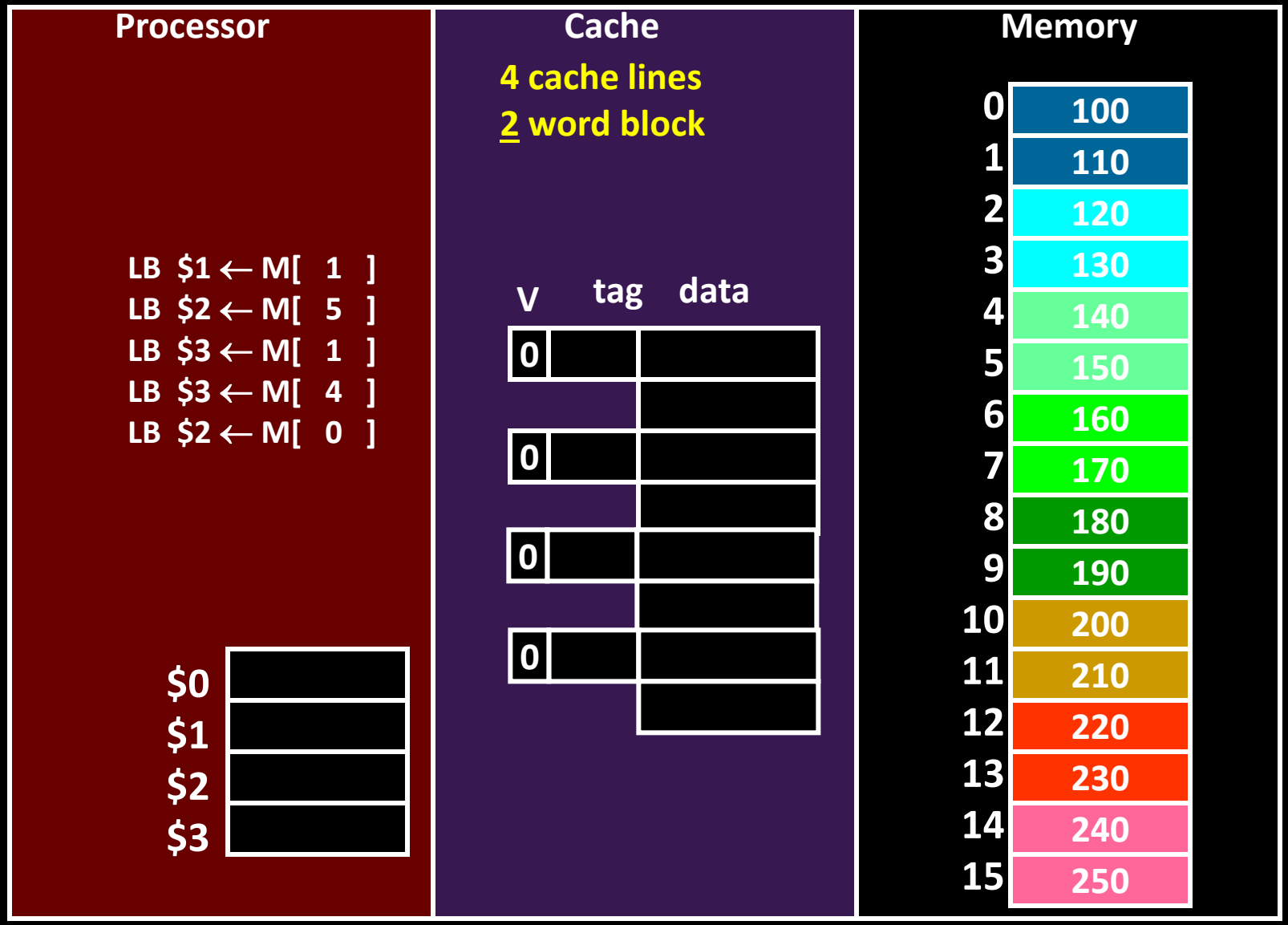
Tag field: 32 – (n + m), Valid bit: 1

SRAM Size: 2^n x (block size + tag size + valid bit size)

= 2^n x (2^m bytes x 8 bits-per-byte + (32–n–m) + 1)

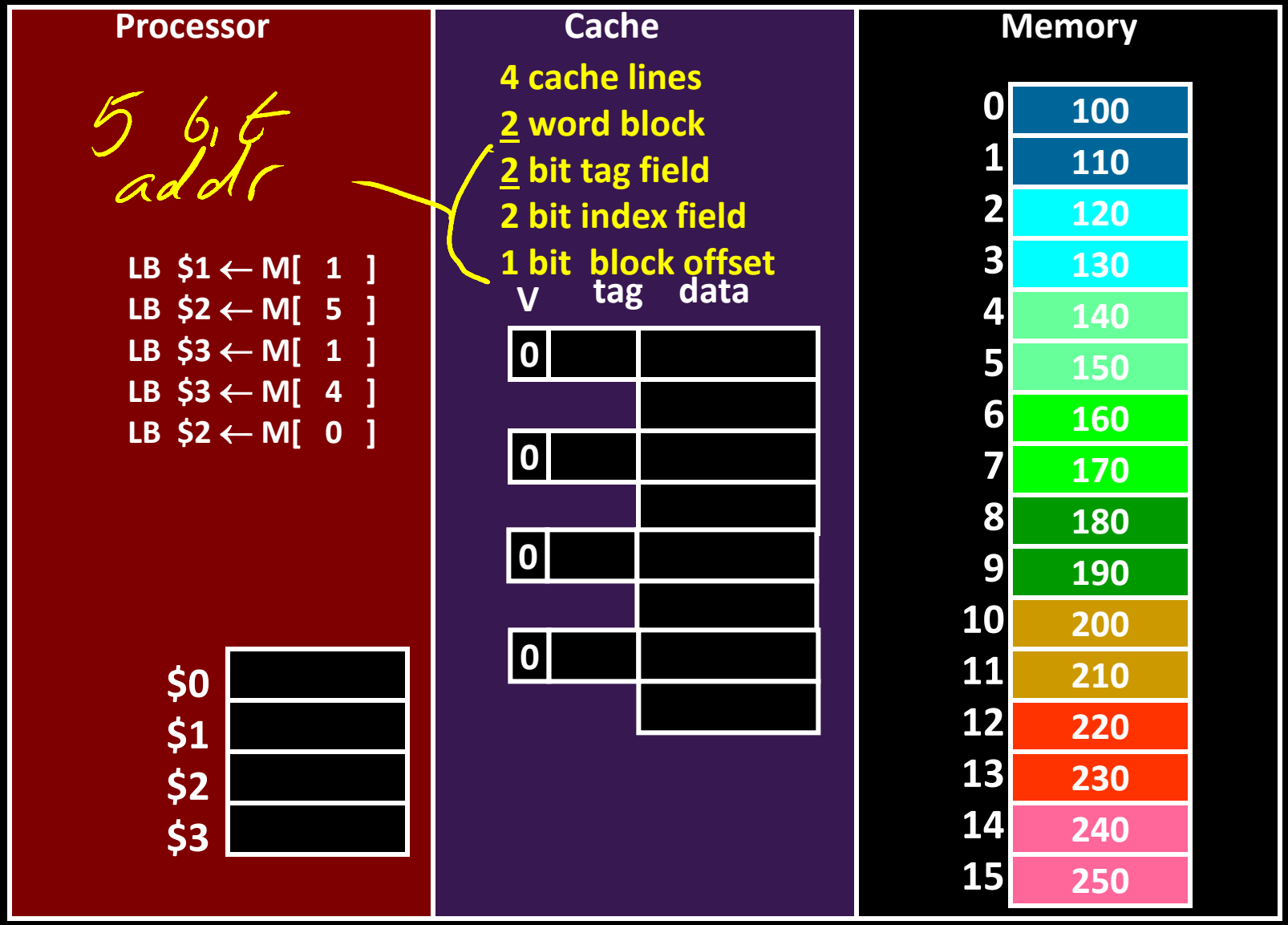
Example:A Simple Direct Mapped Cache

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

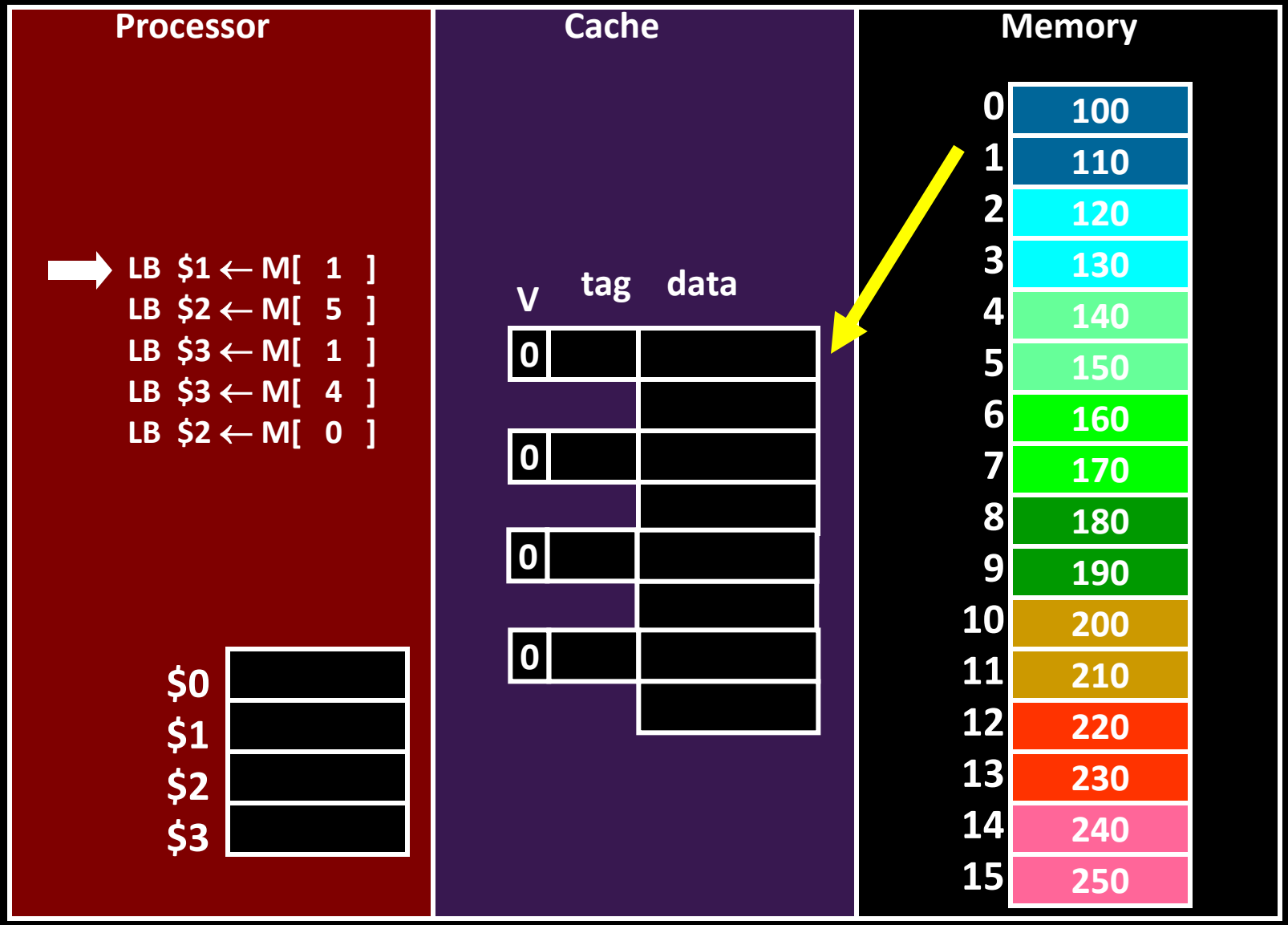


Example:A Simple Direct Mapped Cache

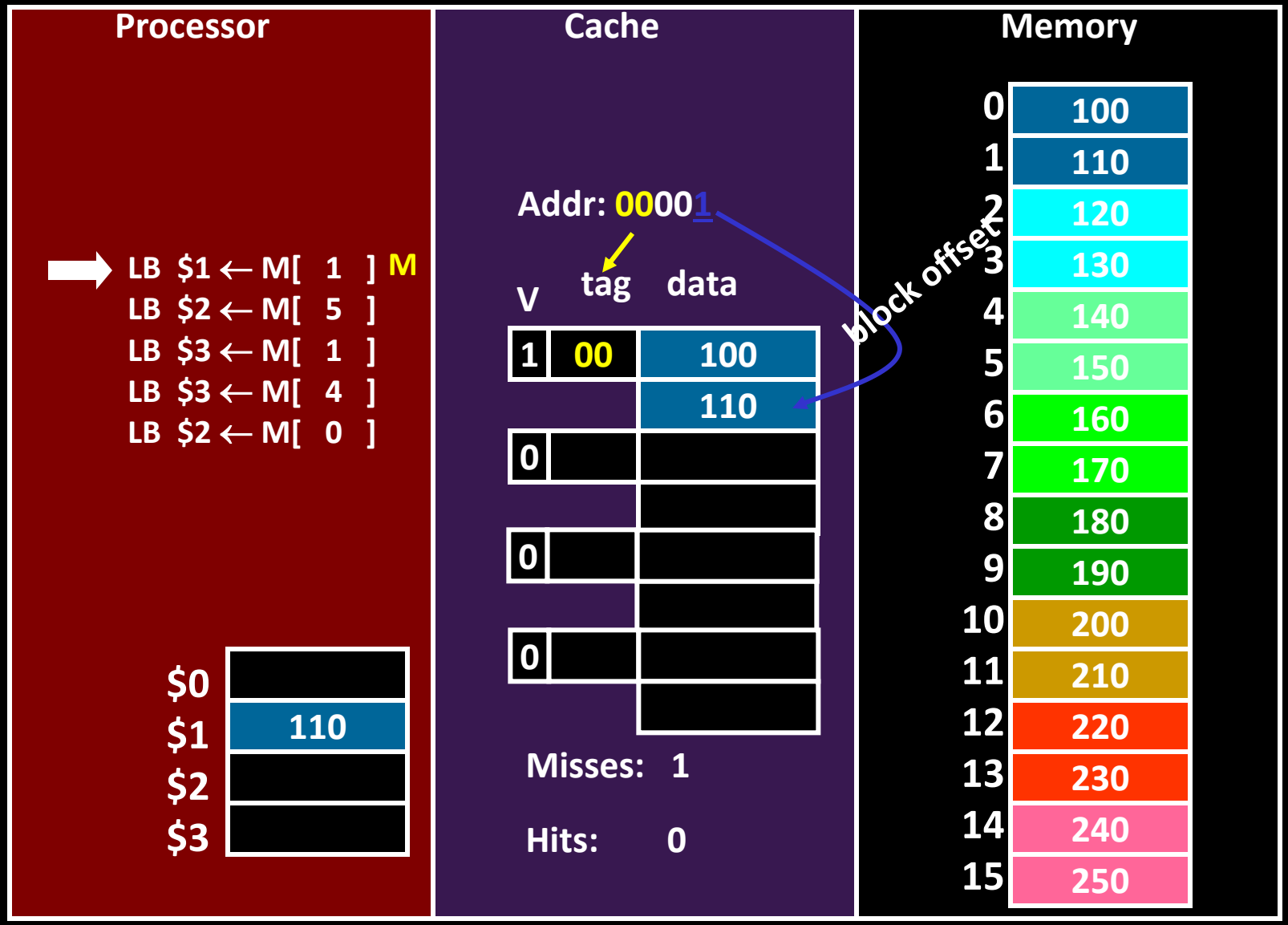
Using **byte addresses** in this example! Addr Bus = 5 bits



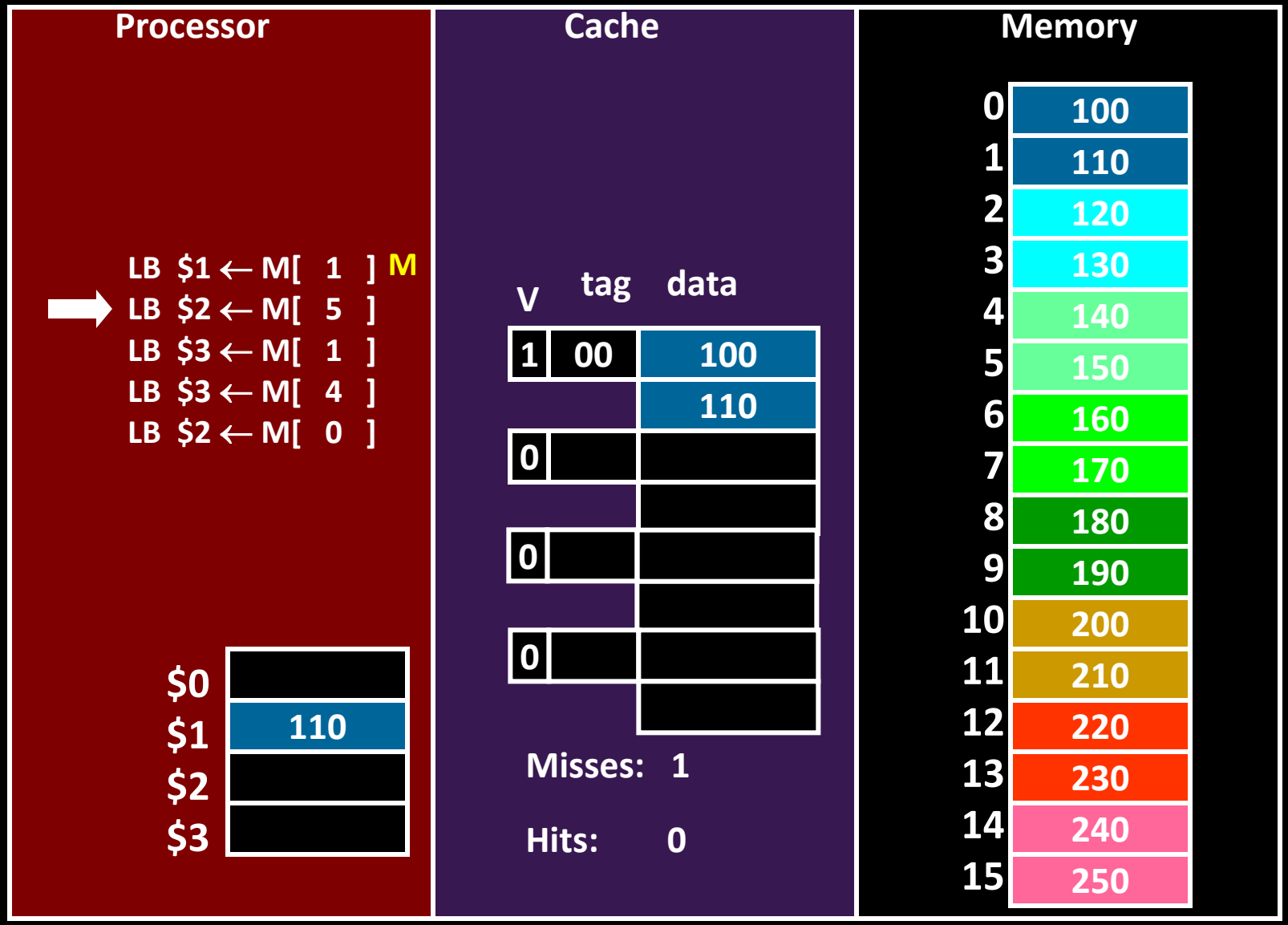
1st Access


Processor
LB $1 ← M[1]
LB $2 ← M[5]
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1
$2
$3
Cache
V
tag
data
0
0
0
0
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

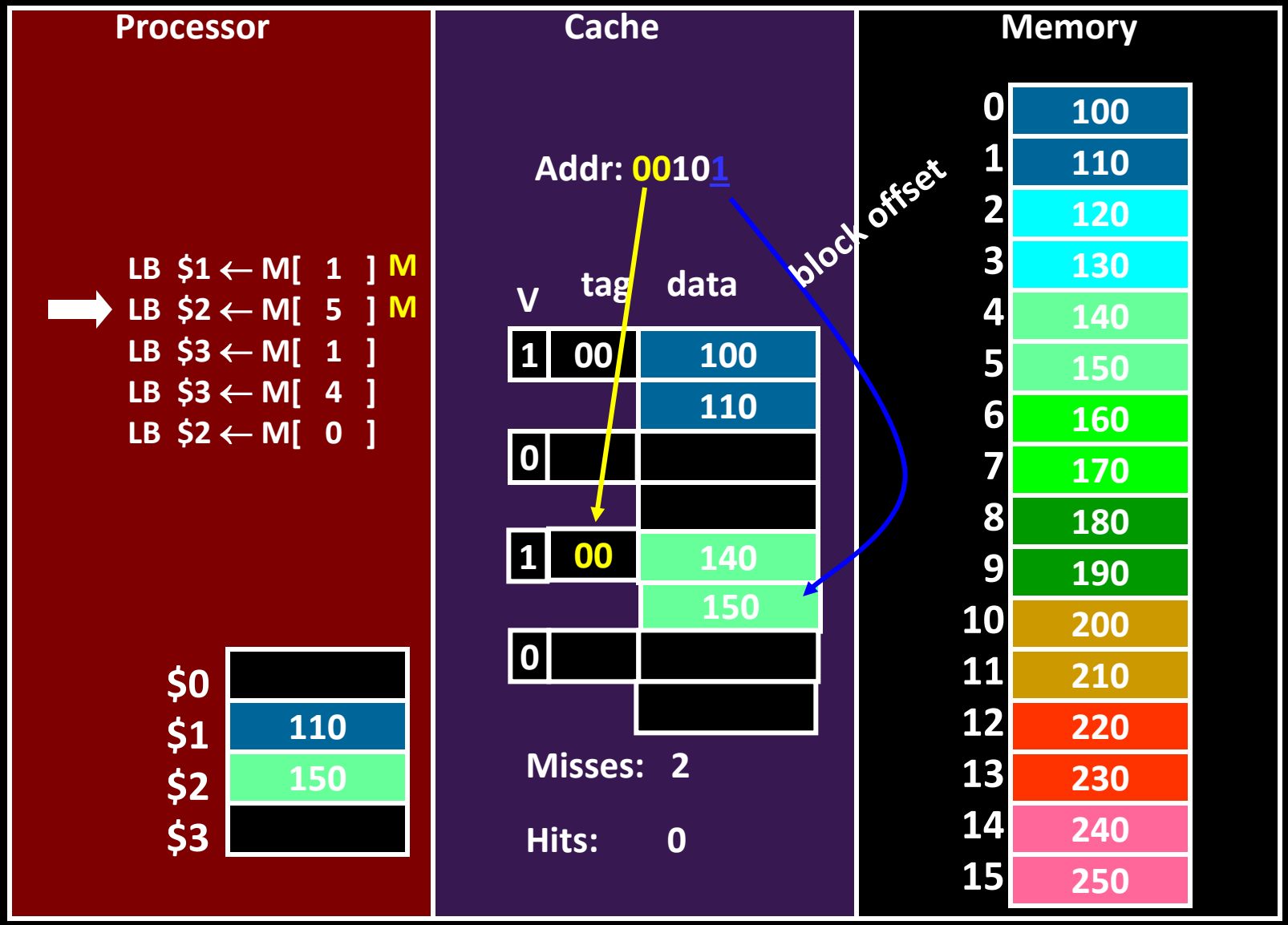
1st Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5]
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1
110
$2
$3
Cache
Addr: 00001
V
tag
data
1
00
100
110
0
0
0
Misses: 1
Hits: 0
block offset
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

2nd Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5]
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1
110
$2
$3
Cache
V
tag
data
1
00
100
110
0
0
0
Misses: 1
Hits: 0
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

2nd Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5] M
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1 110
$2 150
$3
Cache
Addr: 00101
block offset
V tag data
1 00 100
110
0
1 00 140
150
0
Misses: 2
Hits: 0
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

3rd Access

Processor

LB $1 ← M[1] M
LB $2 ← M[5] M
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]

$0	
$1	110
$2	150
$3	

Cache

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 2

Hits: 0

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

3rd Access

Processor

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

$0	
$1	110
$2	150
$3	110

Cache

Addr: 00001

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 2

Hits: 1

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

4th Access

Processor

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

$0	
$1	110
$2	150
$3	110

Cache

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

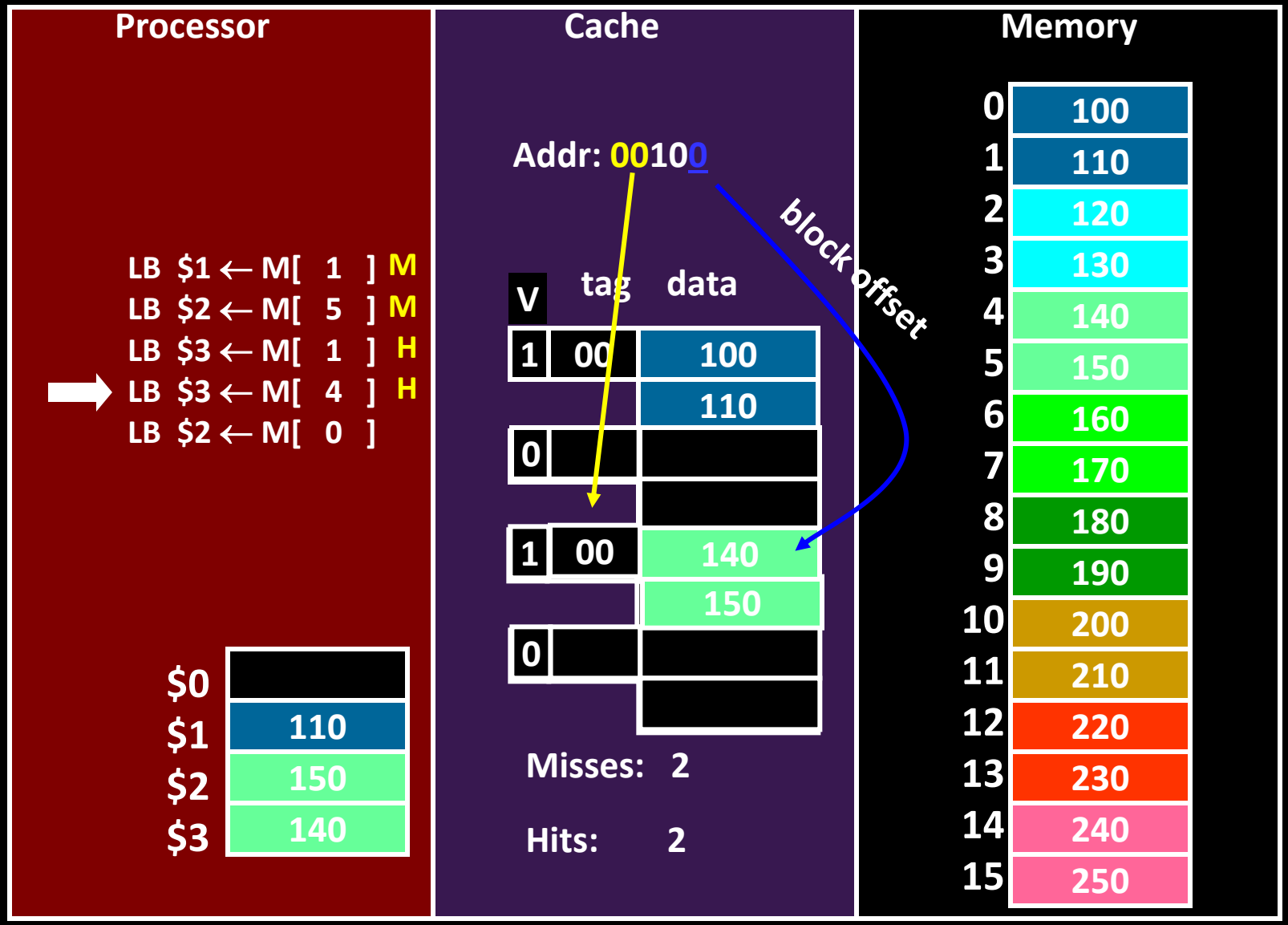
Misses: 2

Hits: 1

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

4th Access


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]
$0
$1 110
$2 150
$3 140
Cache
Addr: 00100
block offset
V tag data
1 00 100
110
0
1 00 140
150
0
Misses: 2
Hits: 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

5th Access

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]

$0	
$1	110
$2	150
$3	140

Cache

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 2

Hits: 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

5th Access

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

$0	
$1	110
$2	100
$3	140

Cache

Addr: 00000

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 2

Hits: 3

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

6th Access

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[10]
LB $2 ← M[15]

$0	
$1	110
$2	100
$3	140

Cache

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 2

Hits: 3

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

6th Access

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[10] M
LB $2 ← M[15]

$0	
$1	110
$2	200
$3	140

Cache

Addr: 01010

V	tag	data
1	00	100
		110
1	01	200
		210
1	00	140
		150
0		

Misses: 3

Hits: 3

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

7th Access

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[10] M
LB $2 ← M[15]

$0	
$1	110
$2	200
$3	140

Cache

V	tag	data
1	00	100
		110
1	01	200
		210
1	00	140
		150
0		

Misses: 3

Hits: 3

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

7th Access

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[10] M
LB $2 ← M[15] M

$0	
$1	110
$2	250
$3	140

Cache

Addr: 01111

V	tag	data
1	00	100
		110
1	01	200
		210
1	00	140
		150
1	01	240
		250

Misses: 4

Hits: 3

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

8th Access

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[10] M
LB $2 ← M[15] M
LB $2 ← M[8]

$0	
$1	110
$2	250
$3	140

Cache

V	tag	data
1	00	100
		110
1	01	200
		210
1	00	140
		150
1	01	240
		250

Misses: 4

Hits: 3

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

8th Access

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[10] M
LB $2 ← M[15] M
LB $2 ← M[8] M

$0	
$1	110
$2	180
$3	140

Cache

Addr: 01000

V	tag	data
1	01	180
		190
1	01	200
		210
1	00	140
		150
1	01	240
		250

Misses: 5

Hits: 3

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

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

Q: 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

Direct Mapped Example: 6th Access

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

Pathological example

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]
LB $2 ← M[8]

$0	
$1	110
$2	100
$3	140

Cache

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 2

Hits: 3

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

6th Access

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]

$0	
$1	110
$2	220
$3	140

Cache

Addr: 01100

V	tag	data
1	00	100
		110
0		
1	01	220
		230
0		

Misses: 3

Hits: 3

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

7th Access

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]

$0	
$1	110
$2	220
$3	140

Cache

V	tag	data
1	00	100
		110
0		
1	01	220
		230
0		

Misses: 3

Hits: 3

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

7th Access

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

$0	
$1	180
$2	220
$3	140

Cache

Addr: 01000

V	tag	data
1	01	180
		190
0		
1	01	220
		230
0		

Misses: 4

Hits: 3

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

8th and 9th Access

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]
LB $2 ← M[0]

$0	
$1	180
$2	220
$3	140

Cache

V	tag	data
1	01	180
		190
0		
1	01	220
		230
0		

Misses: 4

Hits: 3

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

8th and 9th Access

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]	M
LB	$2 ← M[0]	M

$0	
$1	180
$2	100
$3	140

Cache

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 4+2

Hits: 3

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

10th and 11th Access

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]	M
LB	$2 ← M[0]	M
LB	$2 ← M[12]	
LB	$2 ← M[8]	

Cache

V	tag	data
1	00	100
		110
0		
1	00	140
		150
0		

Misses: 4+2

Hits: 3

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

10th and 11th Access

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] M
LB $2 ← M[0] M
→ LB $2 ← M[12] M
LB $2 ← M[8] M

Cache

V	tag	data
1	1	180
		190
0		
1	1	220
		230
0		

Misses: 4+2+2

Hits: 3

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

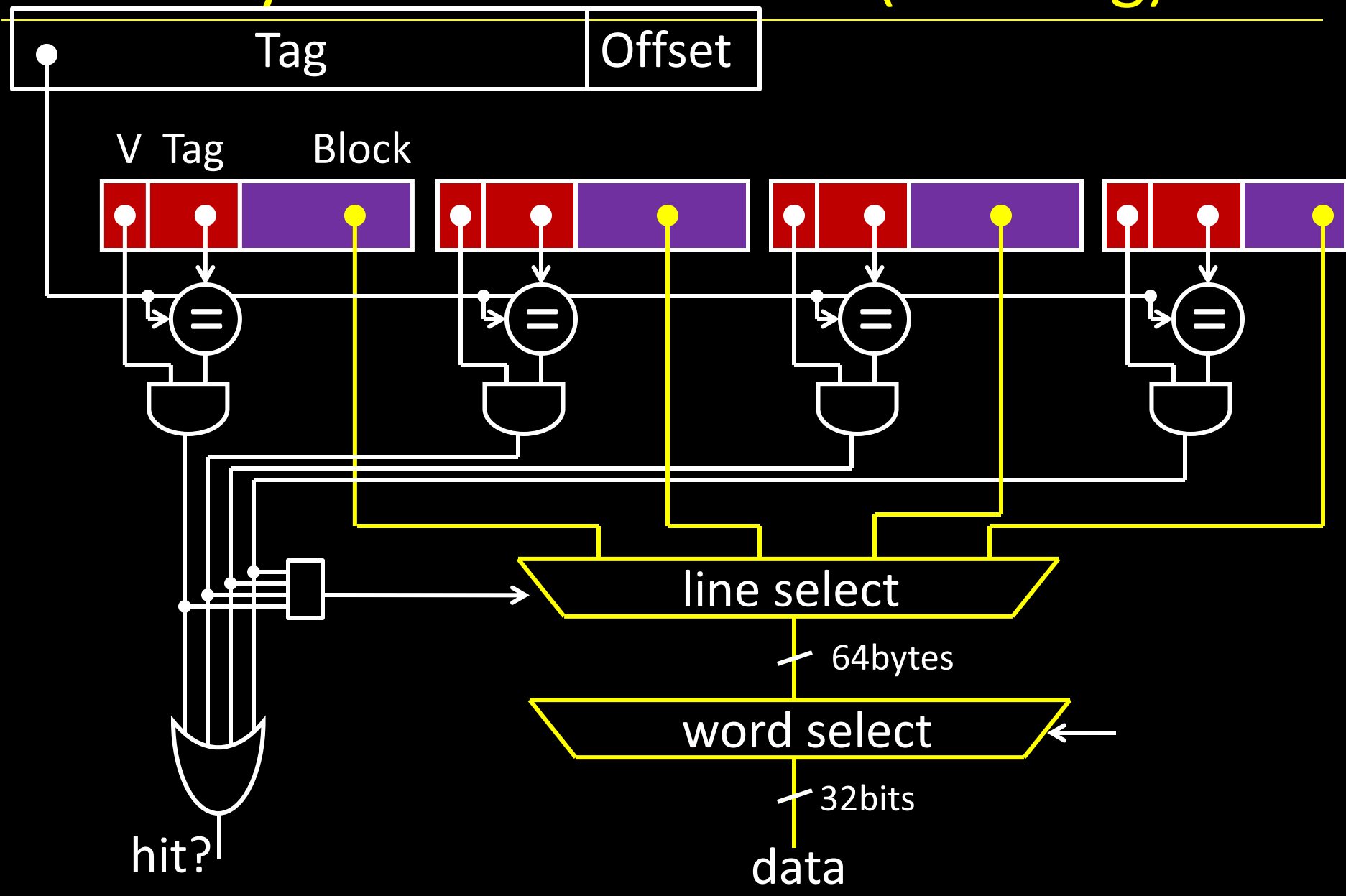
Cache Organization

How to avoid Conflict Misses

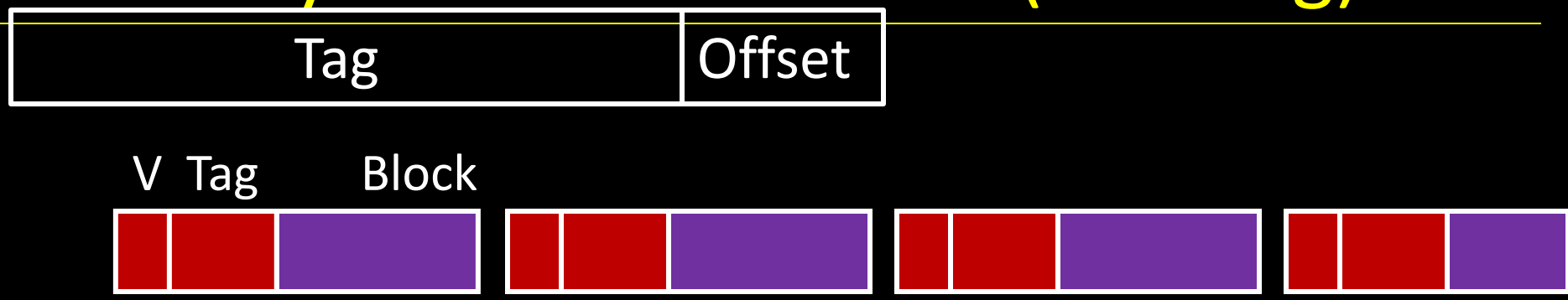
Three common designs

- Fully associative: Block can be anywhere in the cache
- Direct mapped: Block can only be in one line in the cache
- Set-associative: Block can be in a few (2 to 8) places in the cache

Fully Associative Cache (Reading)



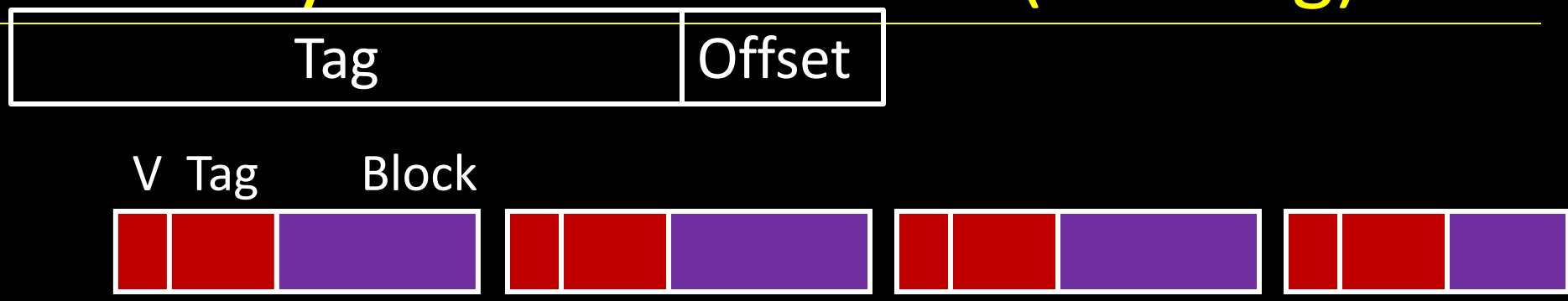
Fully Associative Cache (Reading)



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

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

Fully Associative Cache (Reading)



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

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

Cache of size 2^n blocks

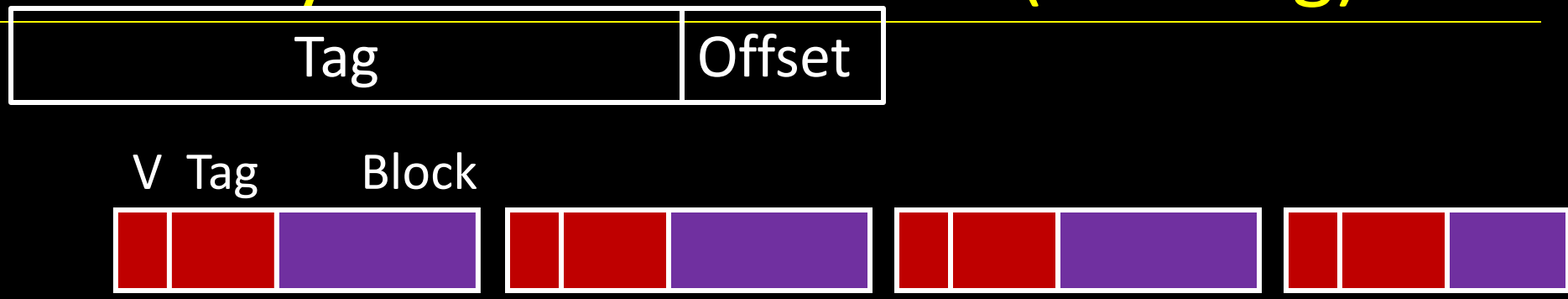
Block size of 2^m bytes

Cache Size: number-of-blocks x block size

$= 2^n \times 2^m$ bytes

$= 2^{n+m}$ bytes

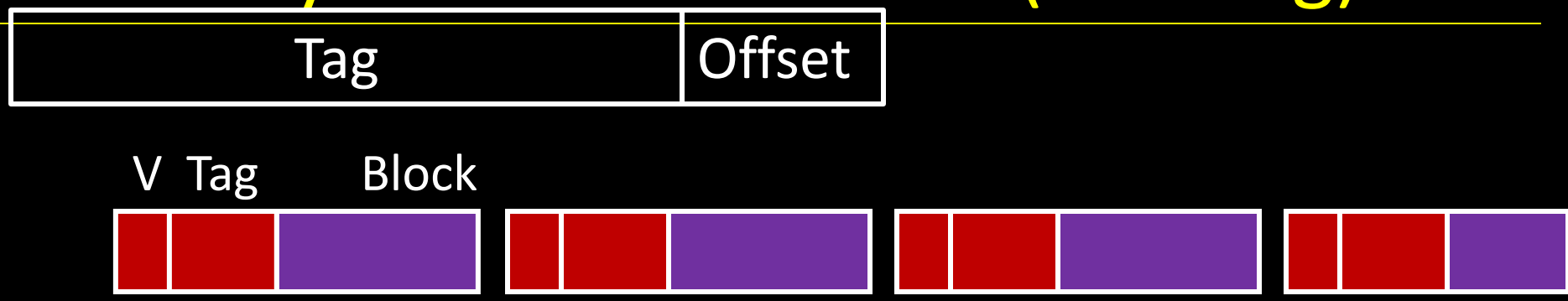
Fully Associative Cache (Reading)



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

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

Fully Associative Cache (Reading)



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

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

Cache of size 2^n blocks

Block size of 2^m bytes

Tag field: 32 – m

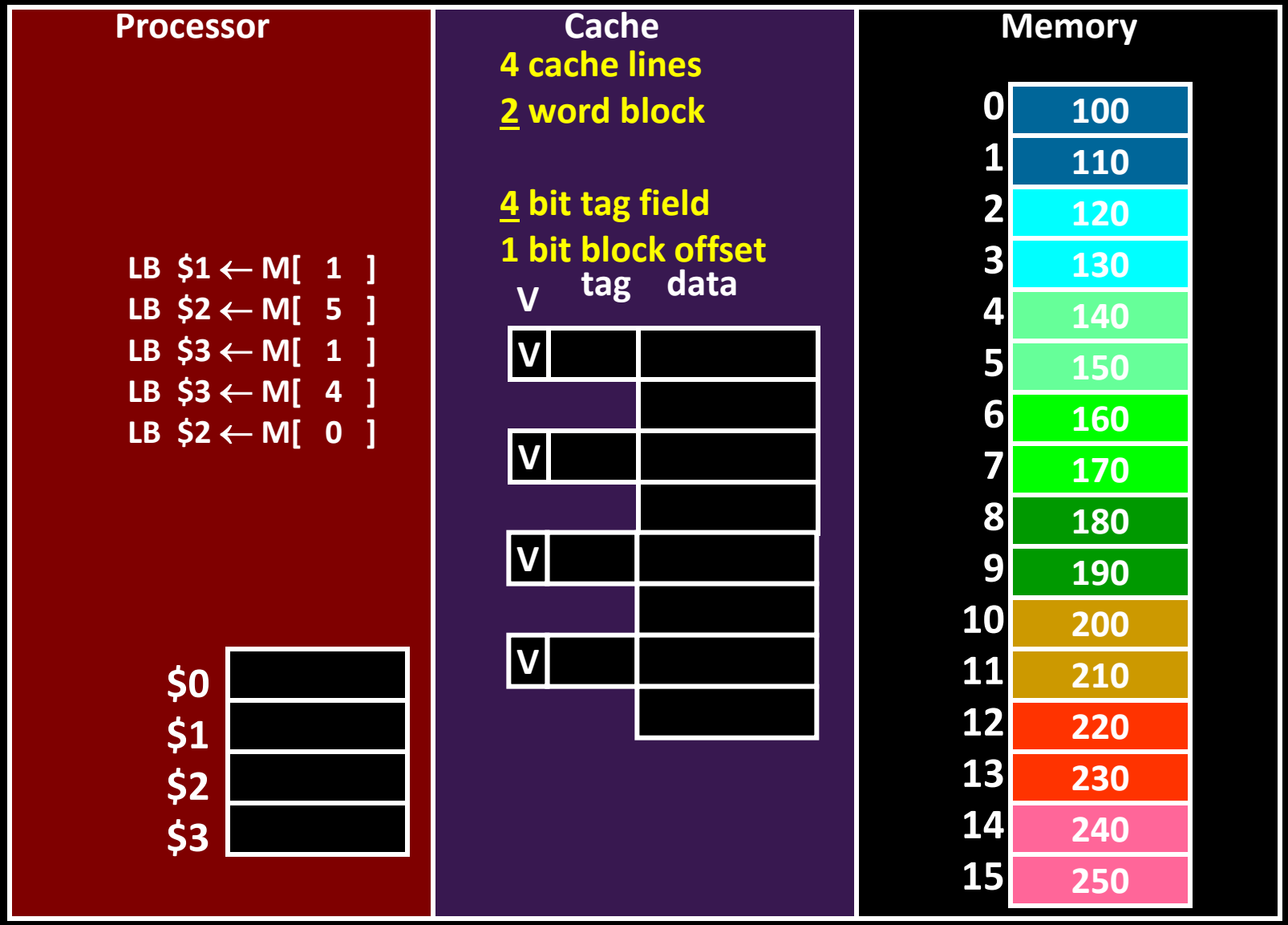
Valid bit: 1

SRAM size: 2^n x (block size + tag size + valid bit size)

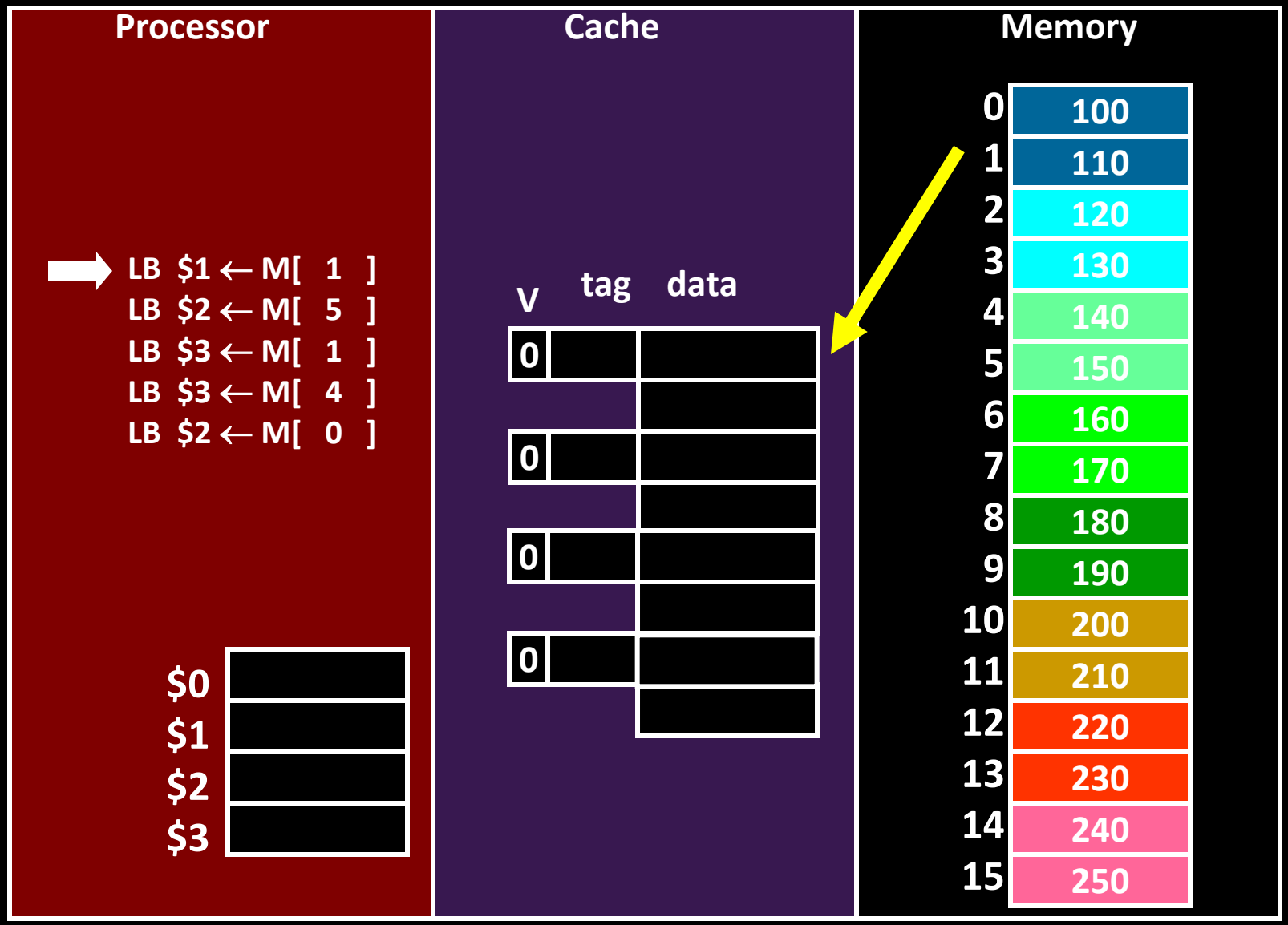
$= 2^n x (2^m \text{ bytes} \times 8 \text{ bits-per-byte} + (32-m) + 1)$

Example:Simple Fully Associative Cache

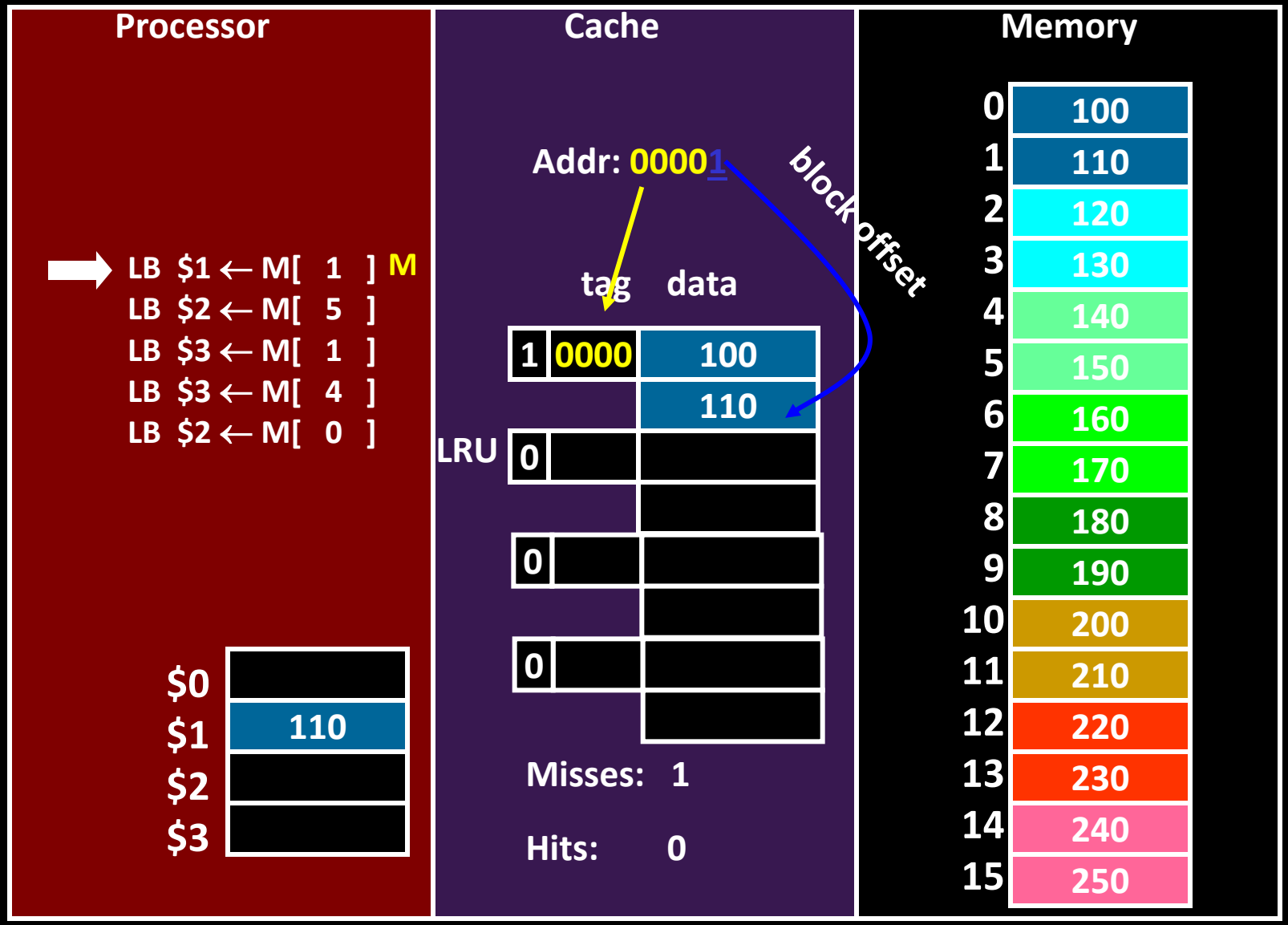
Using **byte addresses** in this example! Addr Bus = 5 bits



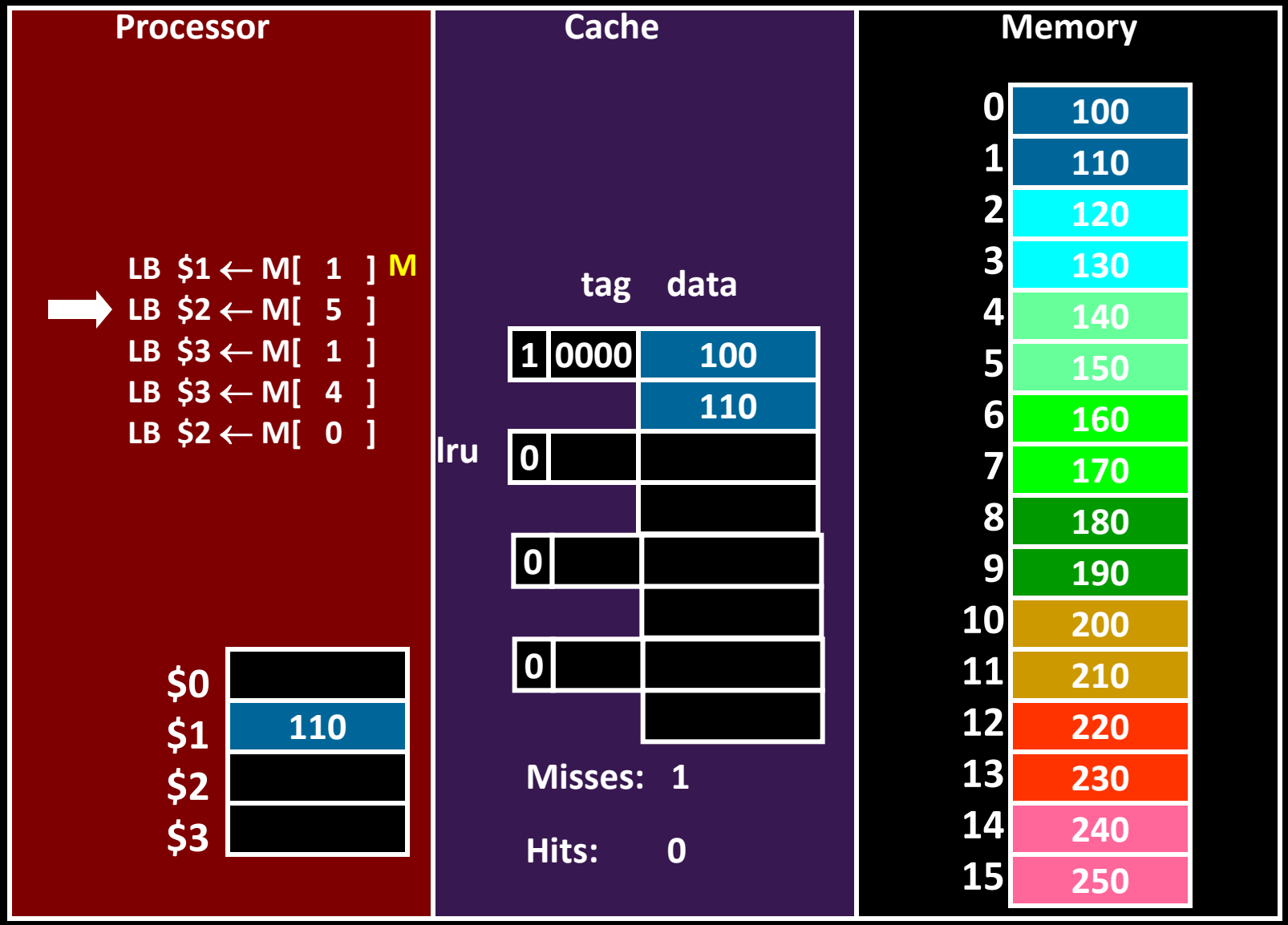
1st Access


Processor
LB $1 ← M[1]
LB $2 ← M[5]
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1
$2
$3
Cache
V
tag
data
0
0
0
0
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

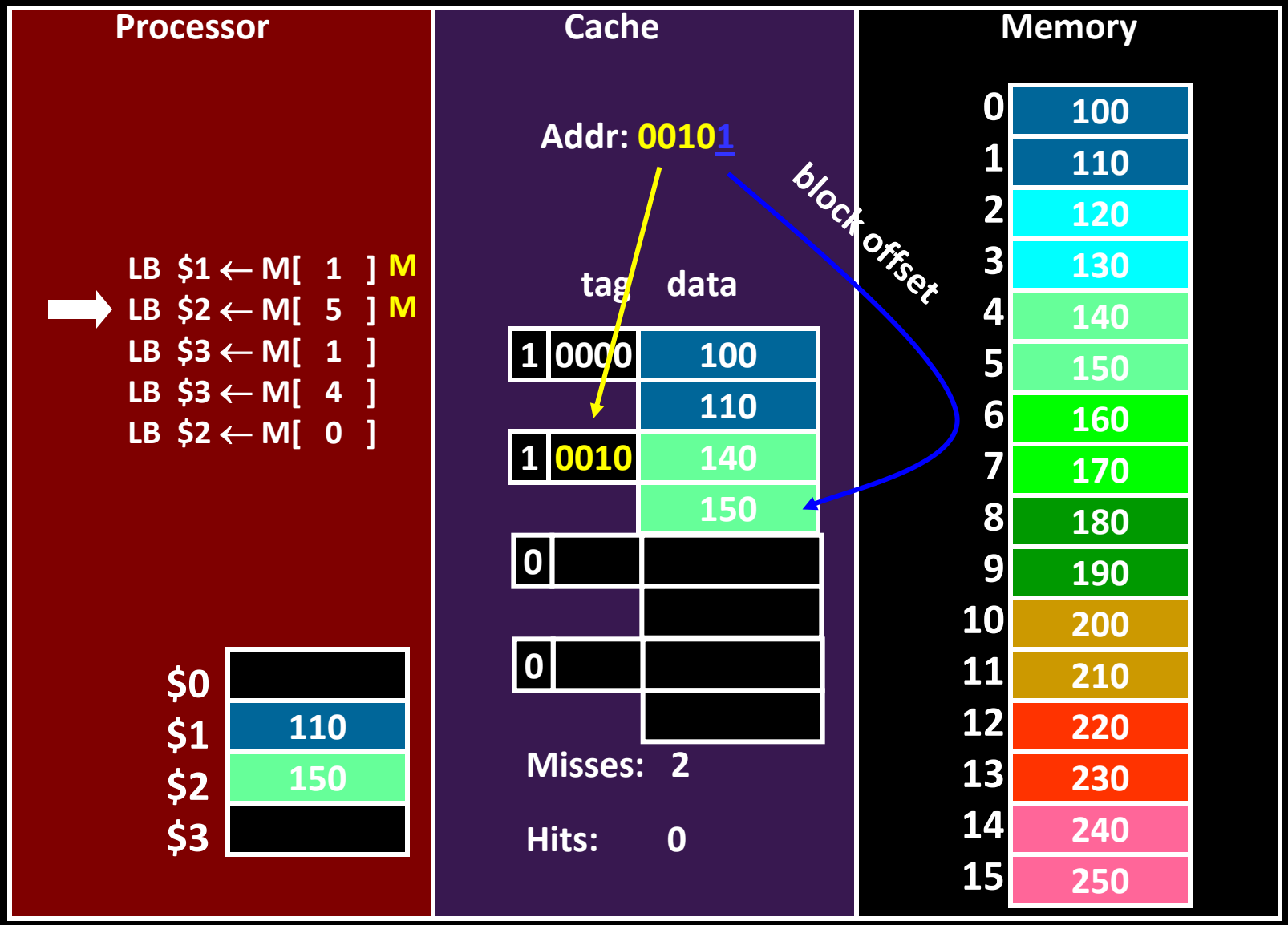
1st Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5]
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1
110
$2
$3
Cache
Addr: 00001
block offset
tag
data
1
0000
100
110
LRU
0
0
0
Misses: 1
Hits: 0
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

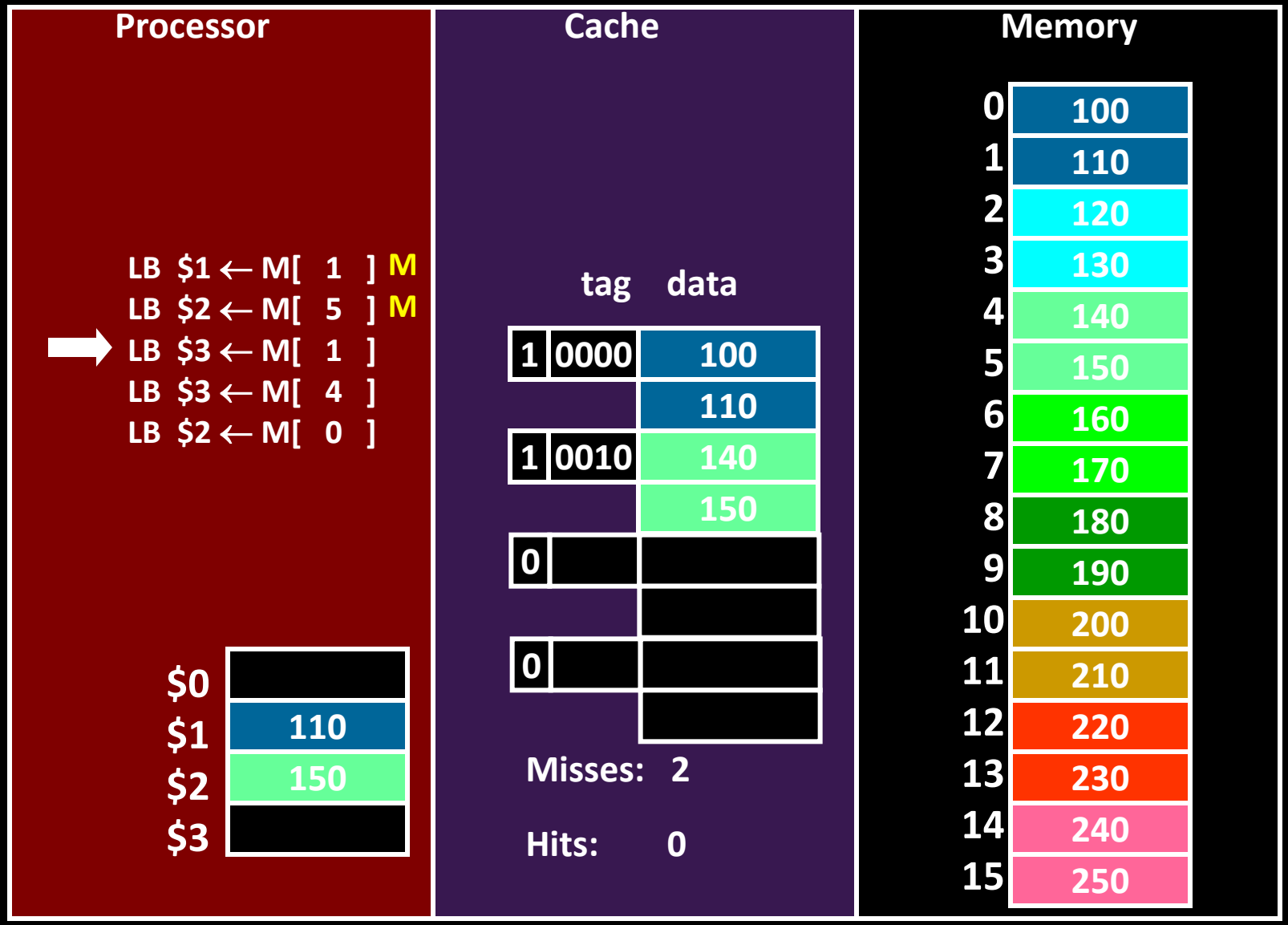
2nd Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5]
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1
110
$2
$3
Cache
tag
data
1
0000
100
110
lru
0
0
0
Misses: 1
Hits: 0
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

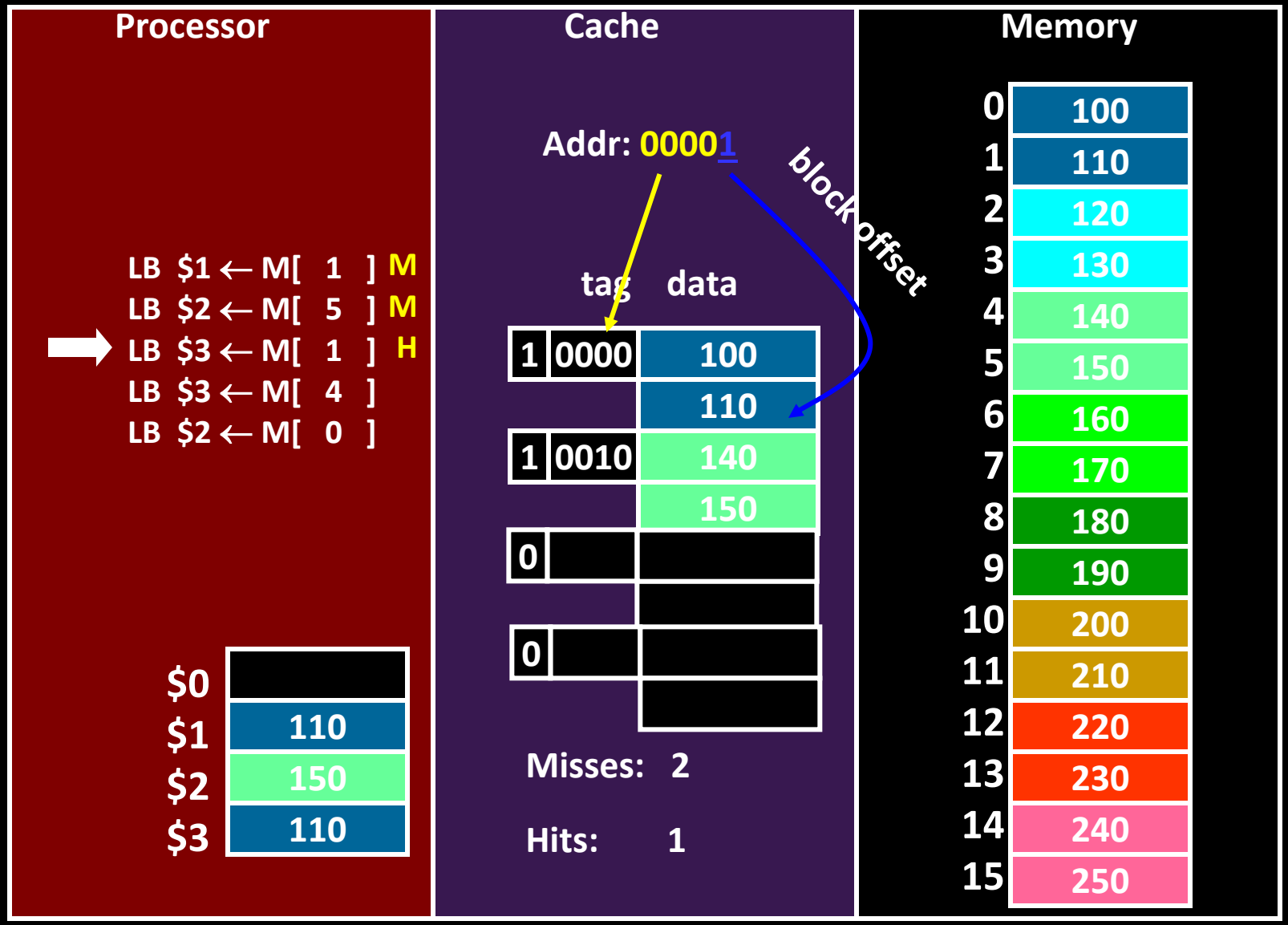
2nd Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5] M
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1 110
$2 150
$3
Cache
Addr: 00101
block offset
tag data
1 0000 100
110
1 0010 140
150
0
0
Misses: 2
Hits: 0
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

3rd Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5] M
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1 110
$2 150
$3
Cache
tag data
1 0000 100
110
1 0010 140
150
0
0
Misses: 2
Hits: 0
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

3rd Access


Processor
LB $1 ← M[1] M
LB $2 ← M[5] M
LB $3 ← M[1] H
LB $3 ← M[4]
LB $2 ← M[0]
$0
$1 110
$2 150
$3 110
Cache
Addr: 00001
block offset
tag data
1 0000 100
110
1 0010 140
150
0
0
Misses: 2
Hits: 1
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

4th Access

Processor

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

$0	
$1	110
$2	150
$3	110

Cache

	tag	data
1	0000	100
		110
1	0010	140
		150
0		
0		

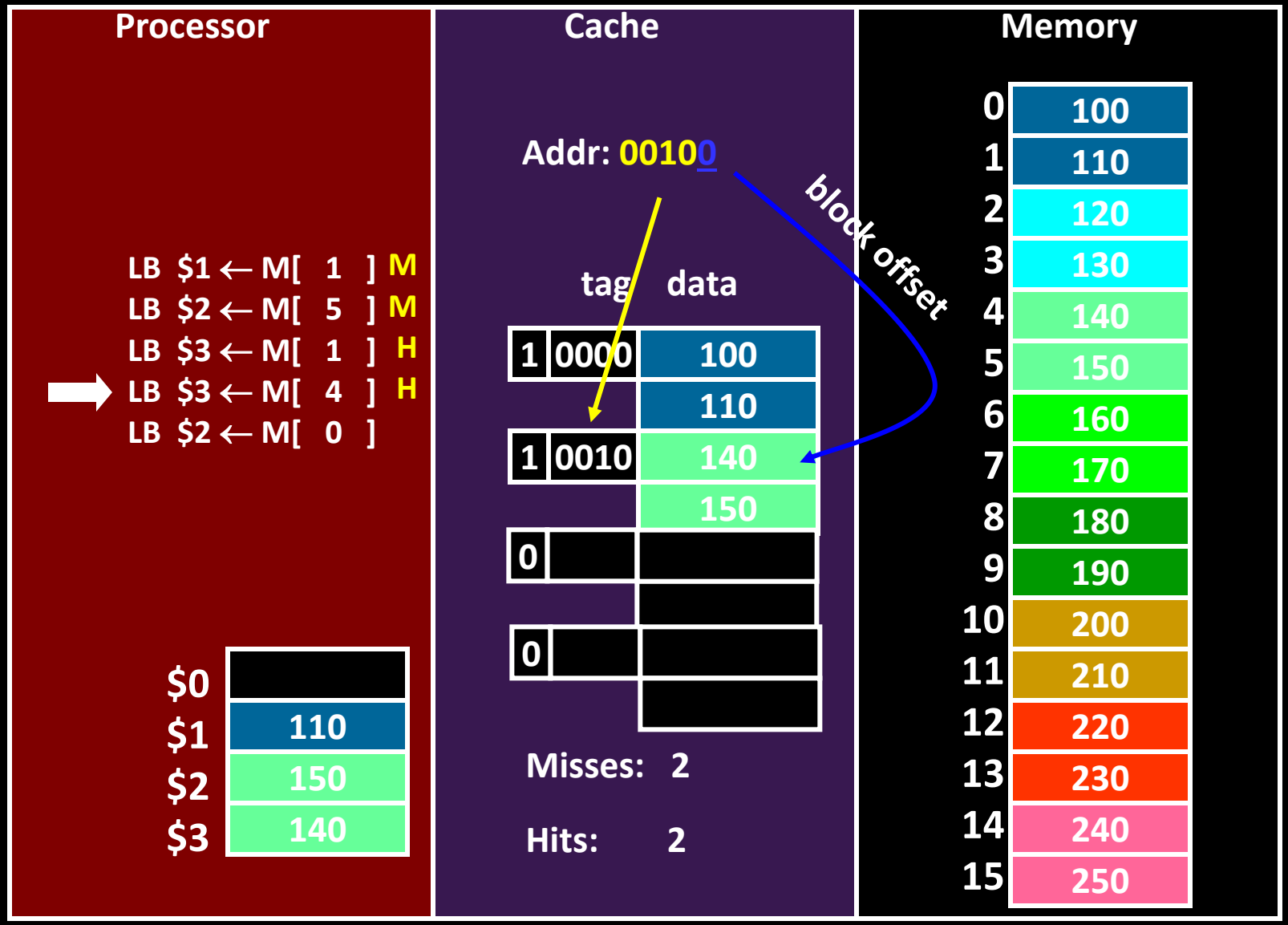
Misses: 2

Hits: 1

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

4th Access


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]
$0
$1 110
$2 150
$3 140
Cache
Addr: 00100
tag data
1 0000 100
110
1 0010 140
150
0
0
Misses: 2
Hits: 2
block offset
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

5th Access

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]

$0	
$1	110
$2	150
$3	140

Cache

	tag	data
1	0000	100
		110
1	0010	140
		150
0		
0		

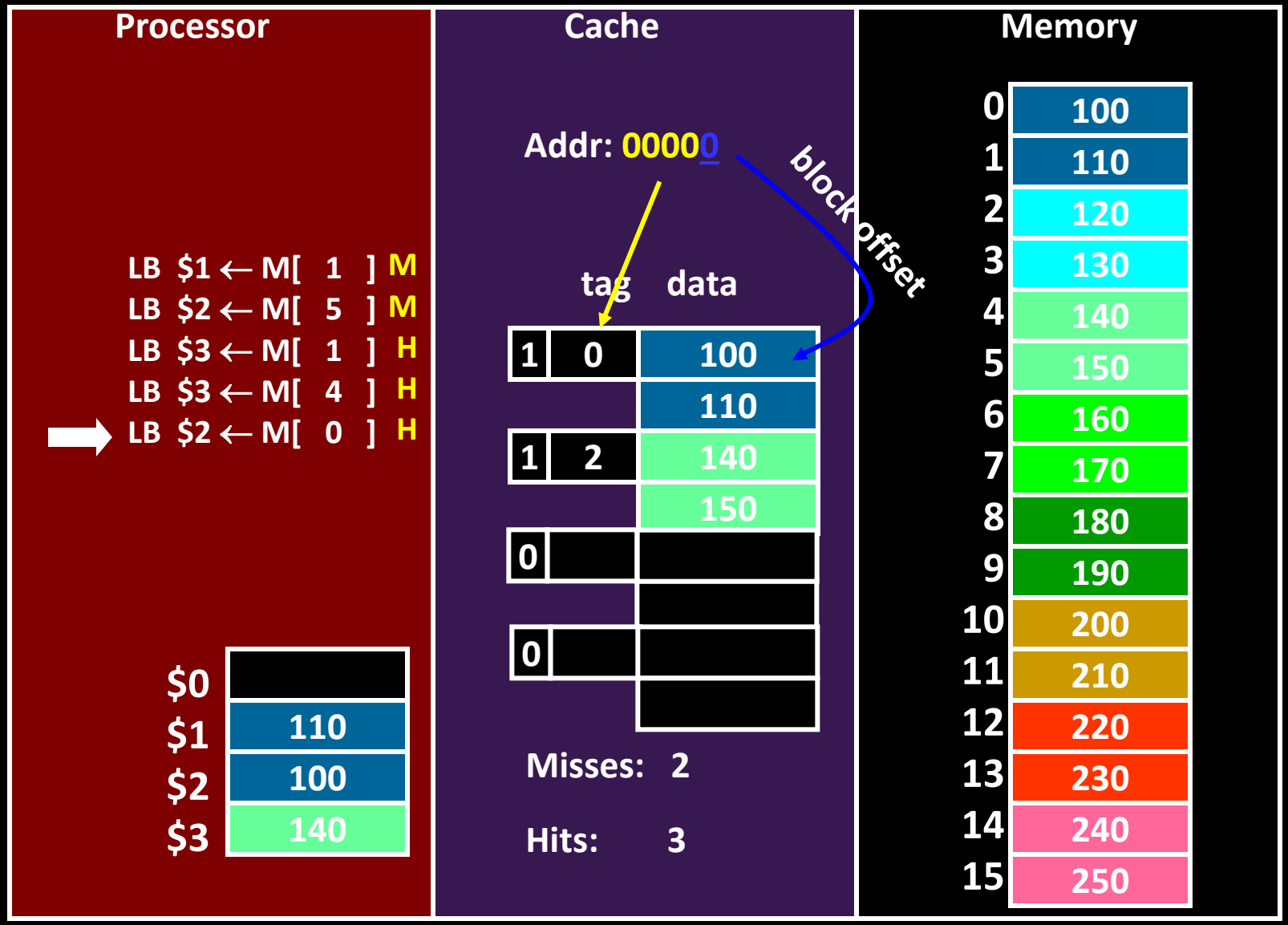
Misses: 2

Hits: 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

5th Access


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
$0
$1 110
$2 100
$3 140
Cache
Addr: 00000
block offset
tag data
1 0 100
110
1 2 140
150
0
0
Misses: 2
Hits: 3
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

6th Access

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]
LB $2 ← M[8]

$0	
$1	110
$2	100
$3	140

Cache

	tag	data
1	0	100
		110
1	2	140
		150
0		
0		

Misses: 2

Hits: 3

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

6th Access

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]

$0	
$1	110
$2	220
$3	140

Cache

Addr: 01100

	tag	data
1	0000	100
		110
1	0010	140
		150
1	0110	220
		230
0		

Misses: 3

Hits: 3

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

7th Access

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]

$0	
$1	110
$2	220
$3	140

Cache

	tag	data
1	0000	100
		110
1	0010	140
		150
1	0110	220
		230
0		

Misses: 3

Hits: 3

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

7th Access

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

$0	
$1	110
$2	180
$3	140

Cache

Addr: 01000

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

Misses: 4

Hits: 3

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

8th and 9th Access

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]
LB $2 ← M[0]

$0	
$1	110
$2	180
$3	140

Cache

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

Misses: 4

Hits: 3

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

8th and 9th Access

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

$0	
$1	110
$2	180
$3	140

Cache

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

Misses: 4

Hits: 3+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

10th and 11th Access

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]
LB $2 ← M[8]

Cache

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

Misses: 4

Hits: 3+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

10th and 11th Access

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

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
+ Smaller	Tag Size	Larger –
+ Less	SRAM Overhead	More –
+ Less	Controller Logic	More –
+ Faster	Speed	Slower –
+ Less	Price	More –
+ Very	Scalability	Not Very –
– Lots	# of conflict misses	Zero +
– Low	Hit rate	High +
– Common	Pathological Cases?	?

Compromise

Set-associative cache

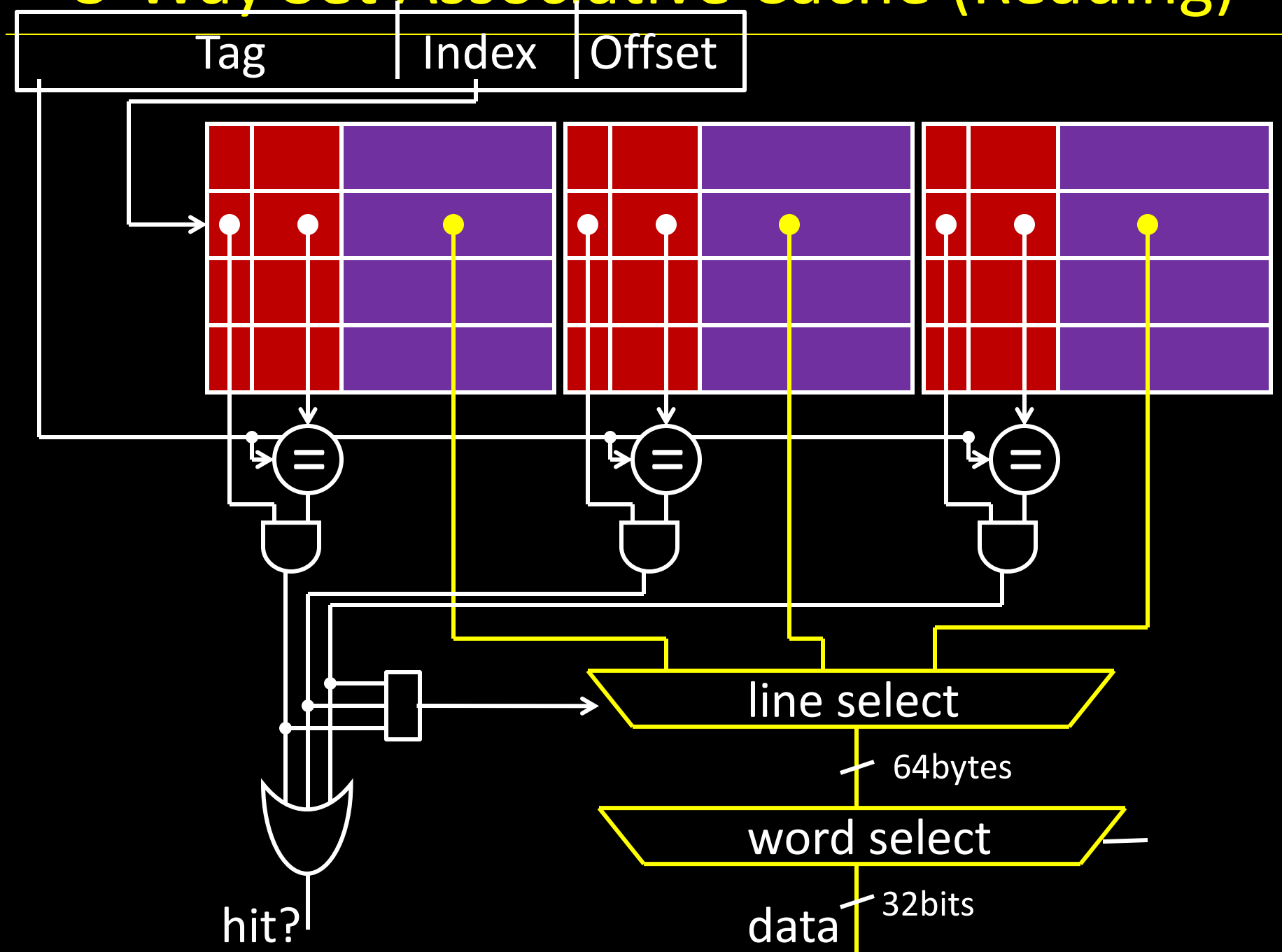
Like a direct-mapped cache

- 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 (Reading)



Comparison: Direct Mapped

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

Processor

LB $1 ← M[1]
LB $2 ← M[5]
LB $3 ← M[1]
LB $3 ← M[4]
LB $2 ← M[0]
LB $2 ← M[12]
LB $2 ← M[5]
→ LB $2 ← M[12]
LB $2 ← M[5]
LB $2 ← M[12]
LB $2 ← M[5]

Cache

4 cache lines

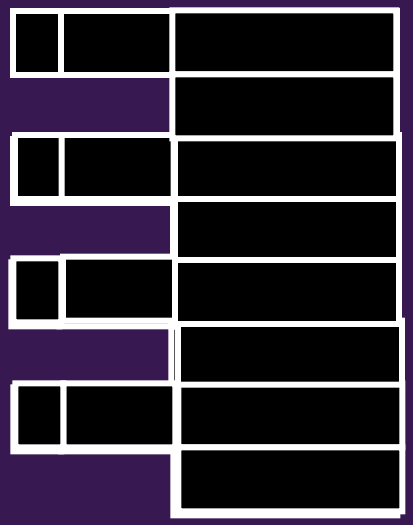
2 word block

2 bit tag field

2 bit index field

1 bit block offset field

tag data



Misses:

Hits:

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

Comparison: Direct Mapped

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

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[5]	M
LB $2 ← M[12]	M
LB $2 ← M[5]	M
LB $2 ← M[12]	M
LB $2 ← M[5]	M

Cache

4 cache lines

2 word block

2 bit tag field

2 bit index field

1 bit block offset field

	tag	data	
1	00	100	
		110	
0			
1	00	140	
		150	
0			

Misses: 8

Hits: 3

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

Comparison: Fully Associative

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

Processor

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

Cache

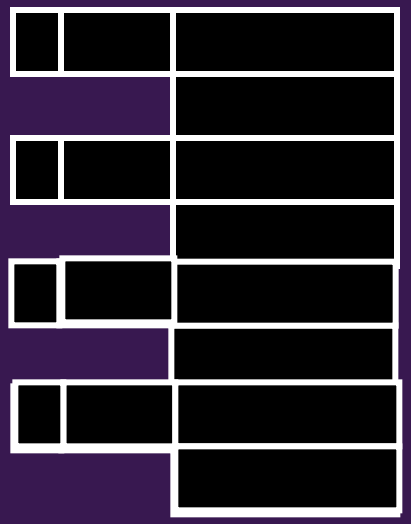
4 cache lines

2 word block

4 bit tag field

1 bit block offset field

tag data



Misses:

Hits:

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

Comparison: Fully Associative

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

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[5]	H
LB	$2 ← M[12]	H
LB	$2 ← M[5]	H
LB	$2 ← M[12]	H
LB	$2 ← M[5]	H

Cache

4 cache lines

2 word block

4 bit tag field

1 bit block offset field

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

Misses: 3

Hits: 8

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

Comparison: 2 Way Set Assoc

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

Processor

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

Cache

2 sets
2 word block
3 bit tag field
1 bit set index field
1 bit block offset field

	tag	data		tag	data
0			0		
0			0		

Misses:

Hits:

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

Comparison: 2 Way Set Assoc

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

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[5] M
LB $2 ← M[12] H
LB $2 ← M[5] H
LB $2 ← M[12] H
LB $2 ← M[5] H

Cache

2 sets
2 word block
3 bit tag field
1 bit set index field
1 bit block offset field

	tag	data		tag	data
0			0		
0			0		

Misses: 4

Hits: 7

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

Remaining Issues

To Do:

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

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

Next up: other designs; writing to caches

Administrivia

Upcoming agenda

- HW3 was due ***yesterday*** Wednesday, March 13th
- PA2 Work-in-Progress circuit due ***before*** spring break
- Spring break: Saturday, March 16th to Sunday, March 24th
- Prelim2 Thursday, March 28th, right after spring break
- PA2 due Thursday, April 4th

Have a great Spring Break!!!