

Caches

Hakim Weatherspoon
CS 3410, Spring 2012
Computer Science
Cornell University

See P&H 5.1, 5.2 (except writes)

Administrivia

HW4 due *today* , March 27th

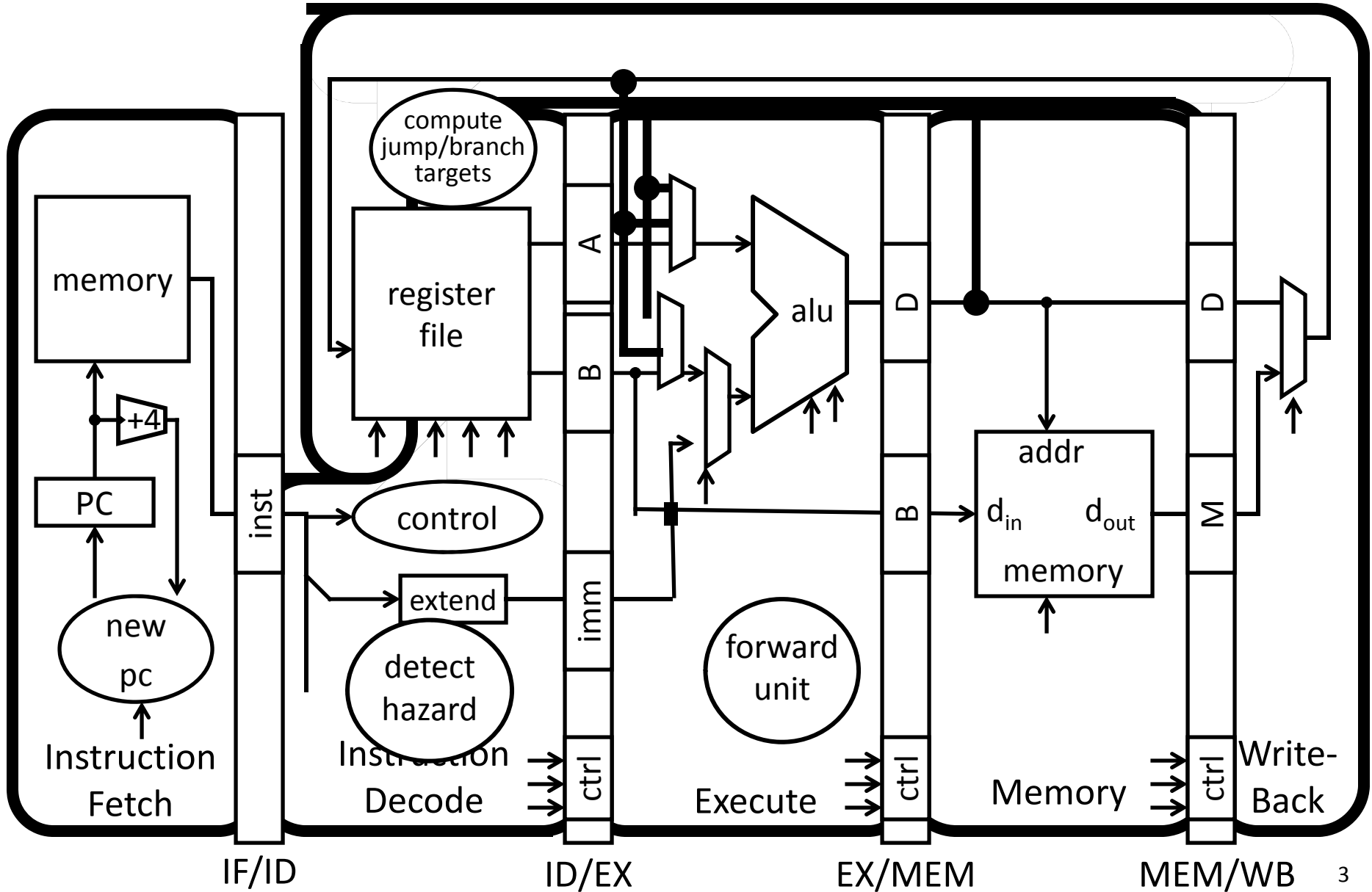
Project2 due next Monday, April 2nd

Prelim2

- Thursday, March 29th at 7:30pm in Philips 101
- Review session *today* 5:30-7:30pm in Phillips 407

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
 - Best of both worlds
- working set: 90/10 rule
- How to predict future: temporal & spacial locality

Examples of caches:

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

Performance

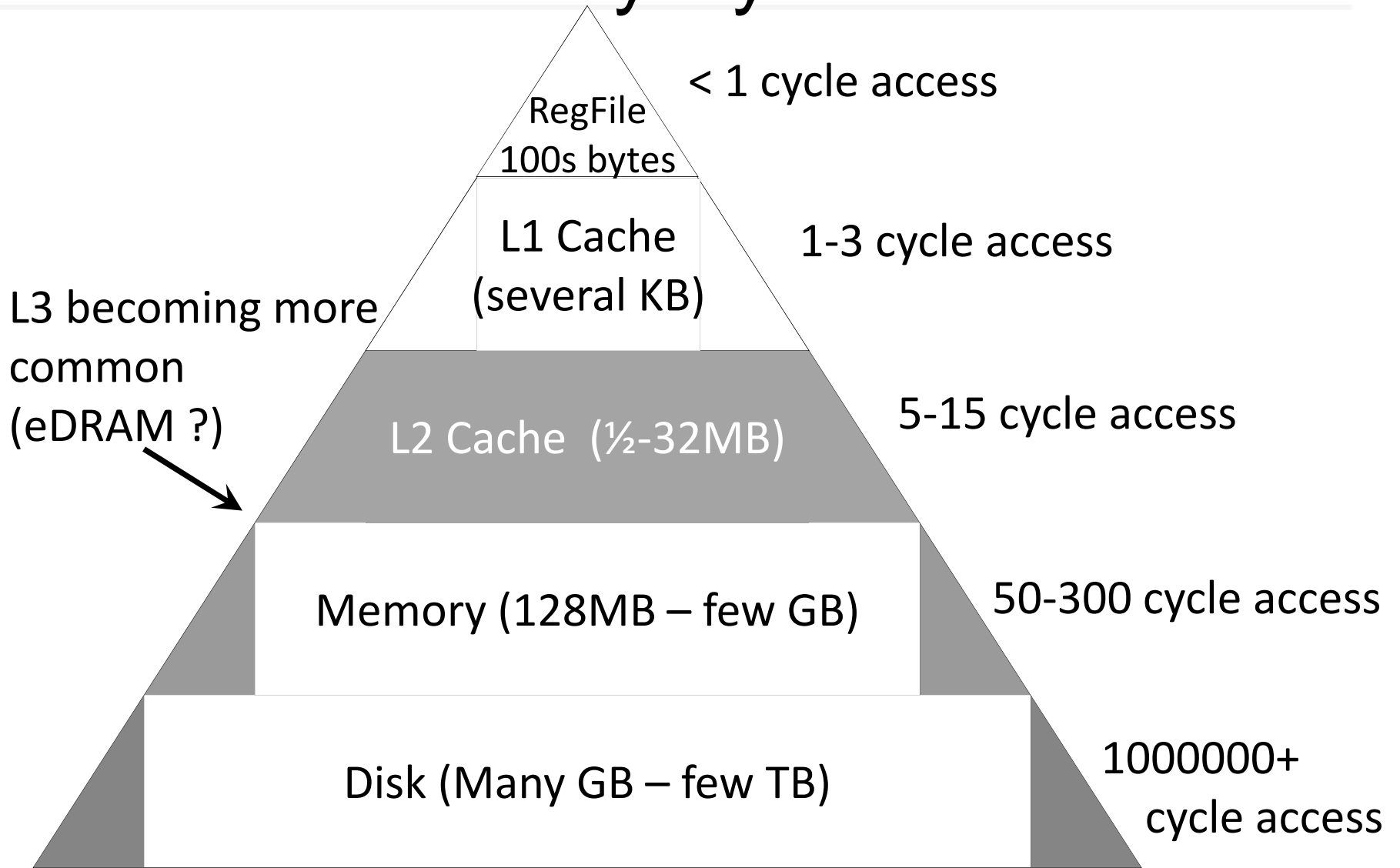
CPU clock rates $\sim 0.2\text{ns} - 2\text{ns}$ (5GHz-500MHz)

Technology	Capacity	\$/GB	Latency
Tape	1 TB	\$.17	100s of seconds
Disk	2 TB	\$.03	Millions of cycles (ms)
SSD (Flash)	128 GB	\$2	Thousands of cycles (us)
DRAM	8 GB	\$10	50-300 cycles (10s of ns)
SRAM off-chip	8 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)

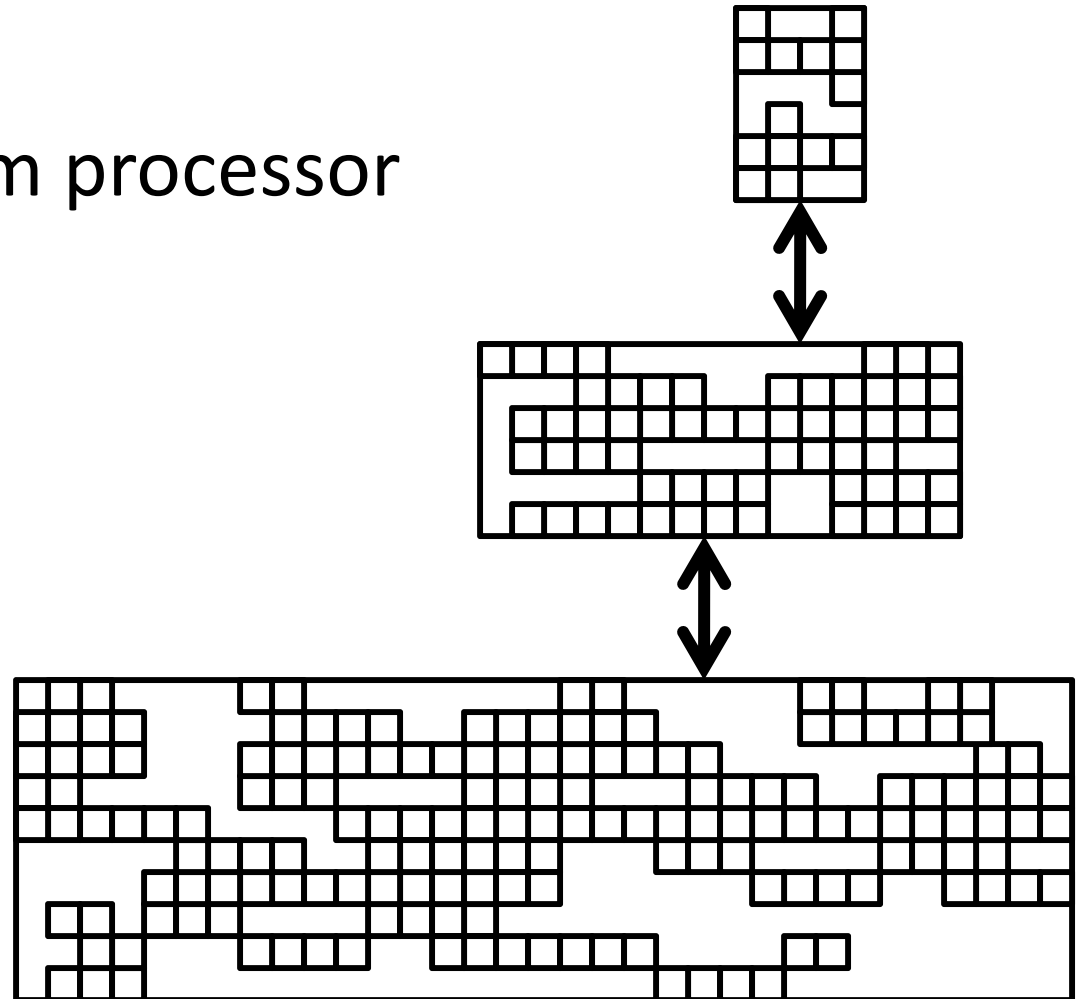
Memory Hierarchy

Memory closer to processor

- small & fast
- stores active data

Memory farther from processor

- big & slow
- stores inactive data



Memory Hierarchy

Insight for Caches

If Mem[x] is 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 $\pm \epsilon$] 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

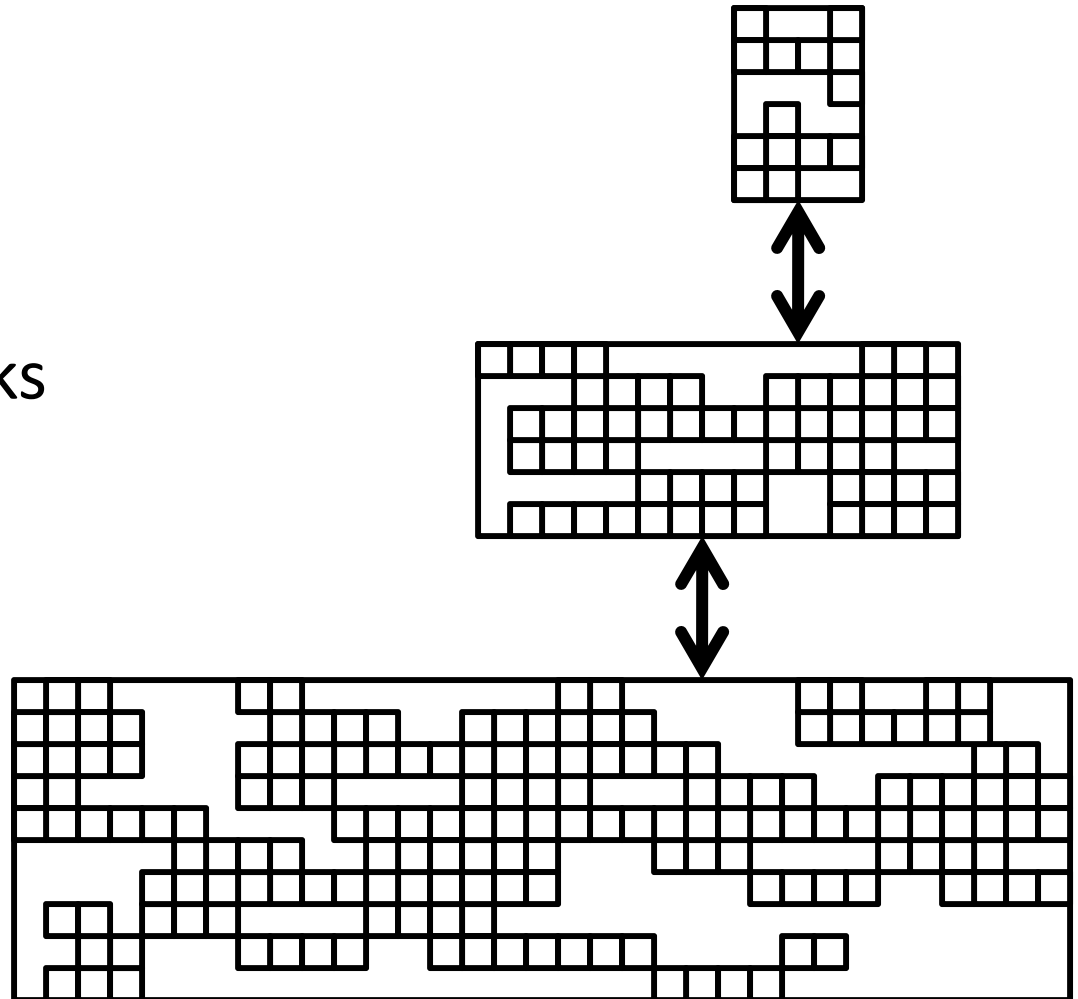
- usually stores subset of memory farther away
 - “strictly inclusive”

- Transfer whole blocks (cache lines):

4kb: disk \leftrightarrow ram

256b: ram \leftrightarrow L2

64b: L2 \leftrightarrow L1



Memory Hierarchy

Memory trace

```
0x7c9a2b18    int n = 4;
0x7c9a2b19    int k[] = { 3, 14, 0, 10 };
0x7c9a2b1a
0x7c9a2b1b
0x7c9a2b1c    int fib(int i) {
0x7c9a2b1d        if (i <= 2) return i;
0x7c9a2b1e        else return fib(i-1)+fib(i-2);
0x7c9a2b1f    }
0x7c9a2b20
0x7c9a2b21
0x7c9a2b22    int main(int ac, char **av) {
0x7c9a2b23        for (int i = 0; i < n; i++) {
0x7c9a2b28            printi(fib(k[i]));
0x7c9a2b2c            prints("\n");
0x0040030c        }
0x00400310    }
0x7c9a2b04
0x00400314    }
0x7c9a2b00
0x00400318
0x0040031c
...
```

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

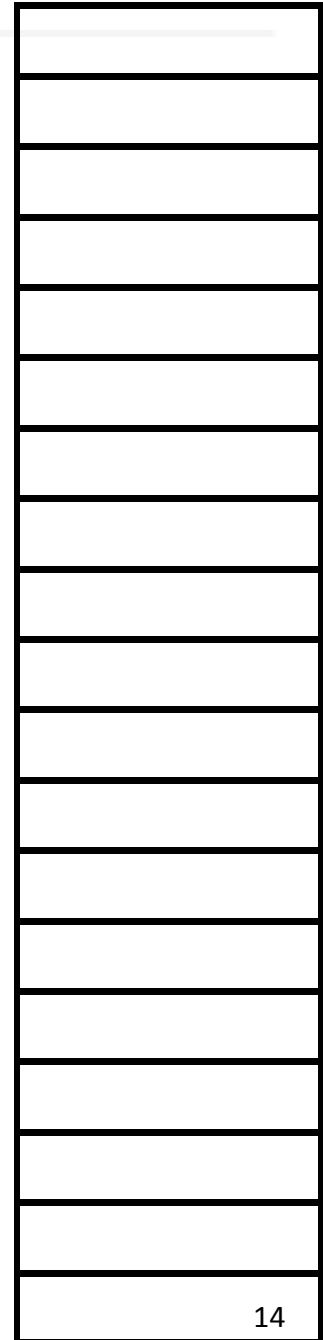
Direct Mapped Cache

Direct Mapped Cache

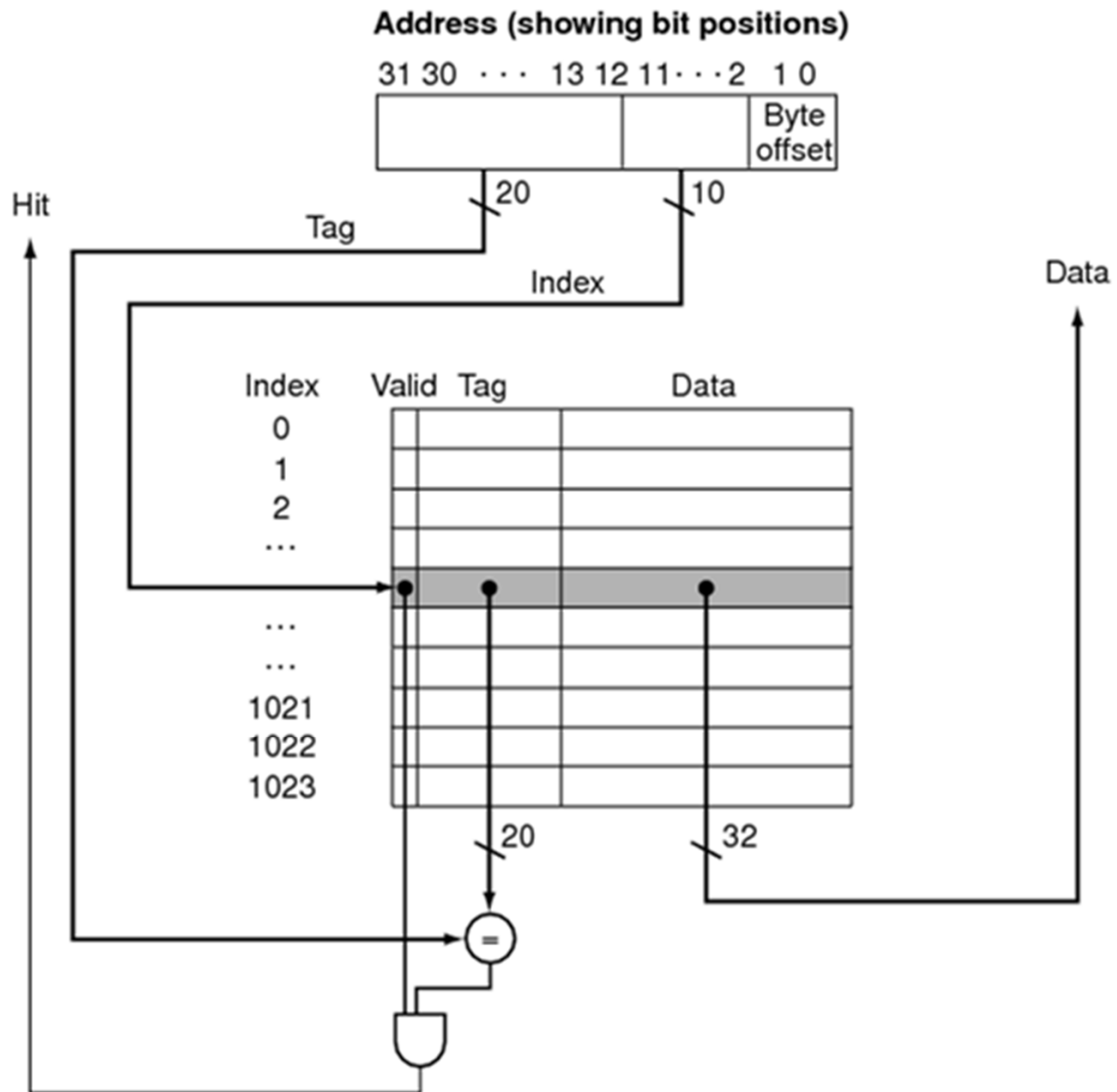
- Each block number mapped to a single cache line index
- Simplest hardware

line 0		
line 1		
line 2		
line 3		

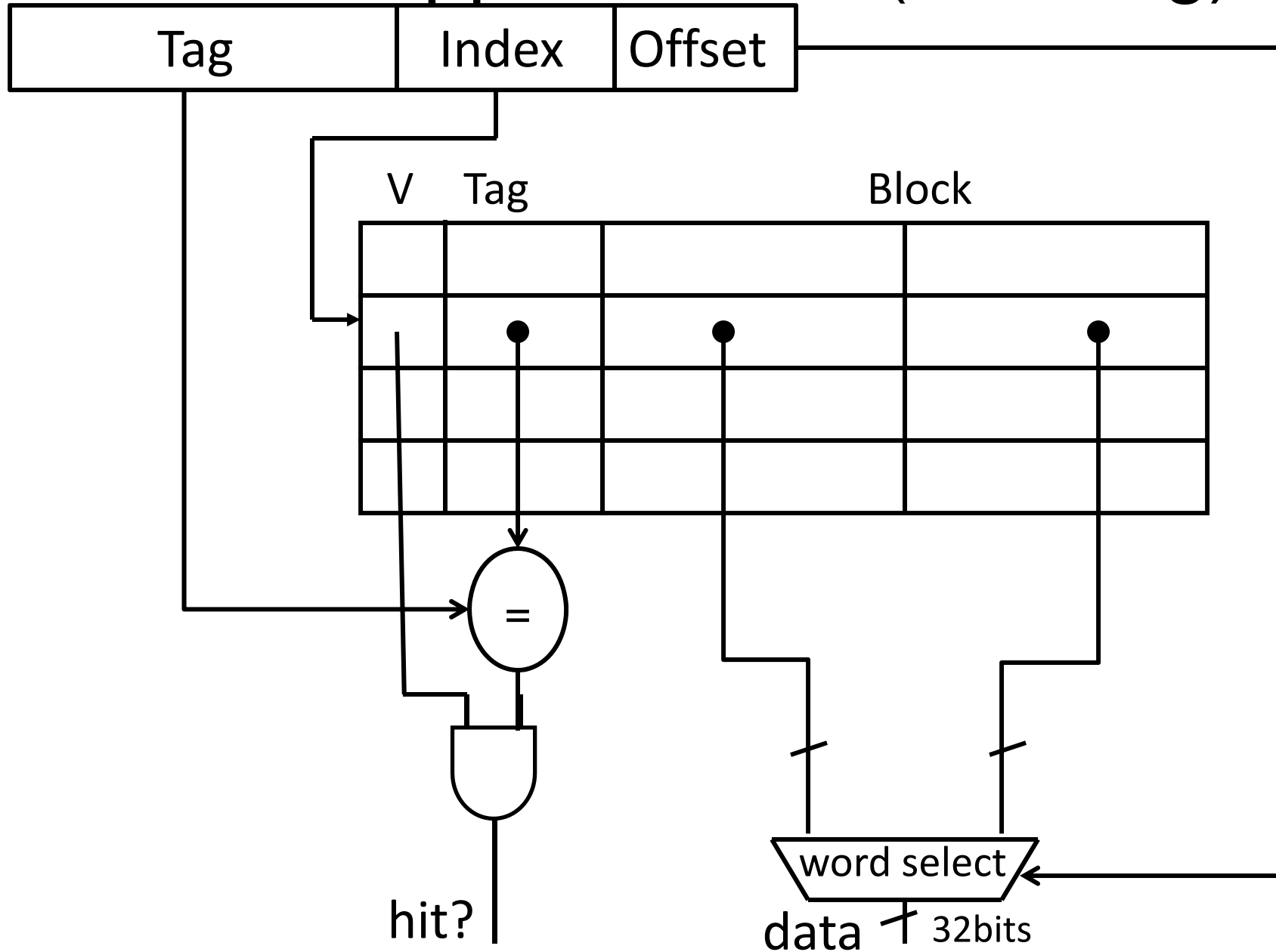
0x000000
0x000004
0x000008
0x00000c
0x000010
0x000014
0x000018
0x00001c
0x000020
0x000024
0x000028
0x00002c
0x000030
0x000034
0x000038
0x00003c
0x000040
0x000044
0x000048



Direct Mapped Cache

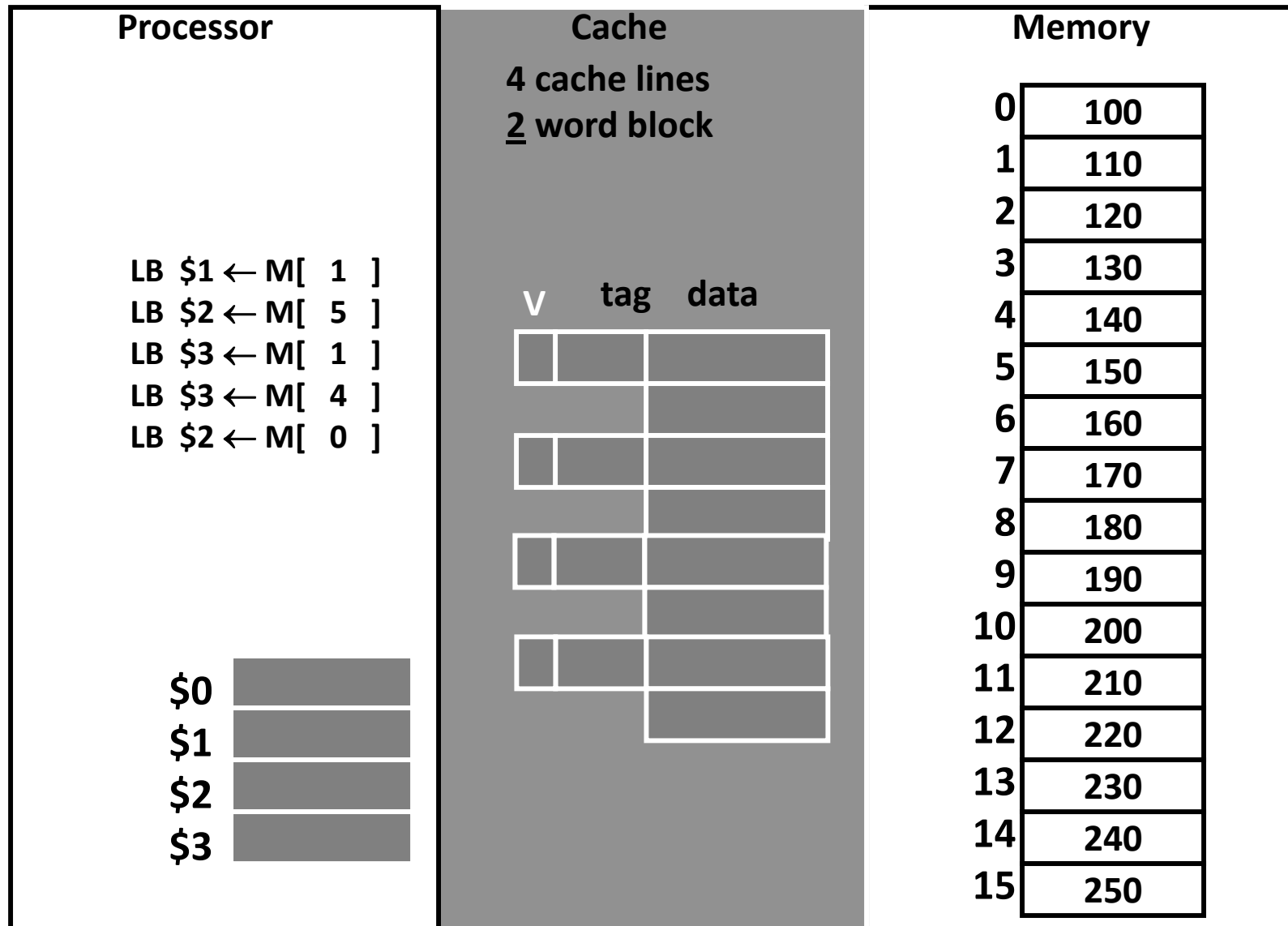


Direct Mapped Cache (Reading)



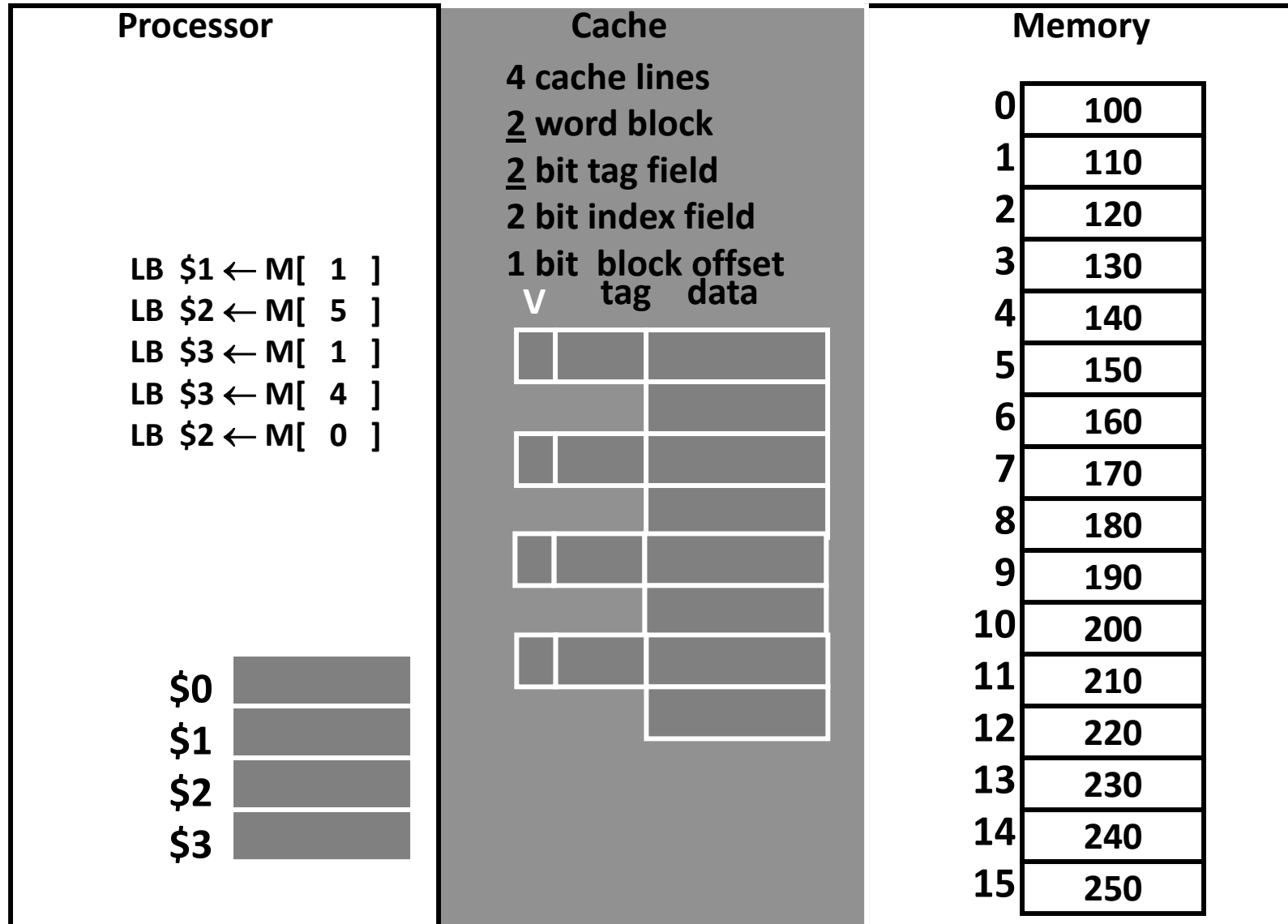
Example: A Simple Direct Mapped Cache

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

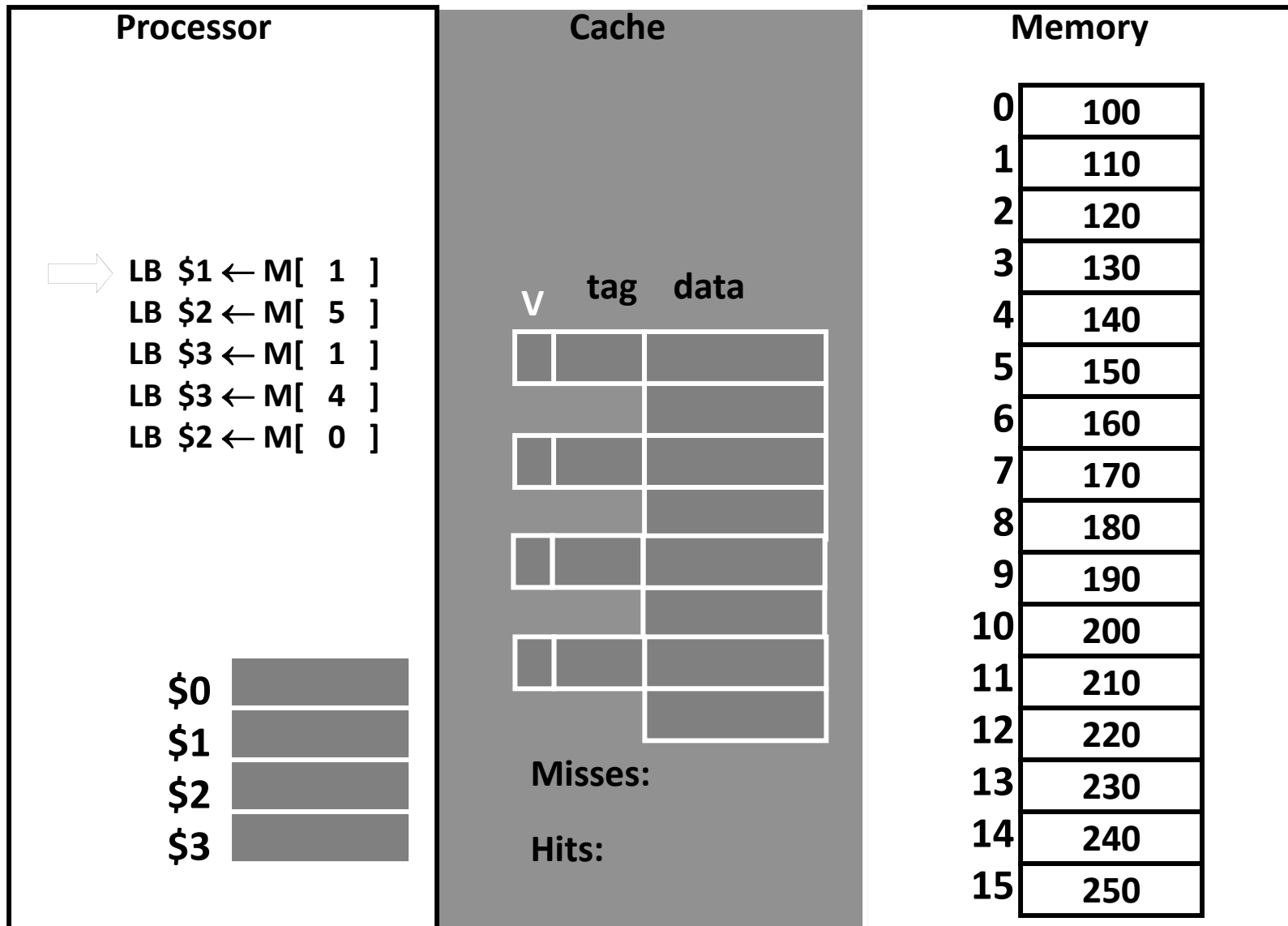


Example: A Simple Direct Mapped Cache

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



1st Access



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

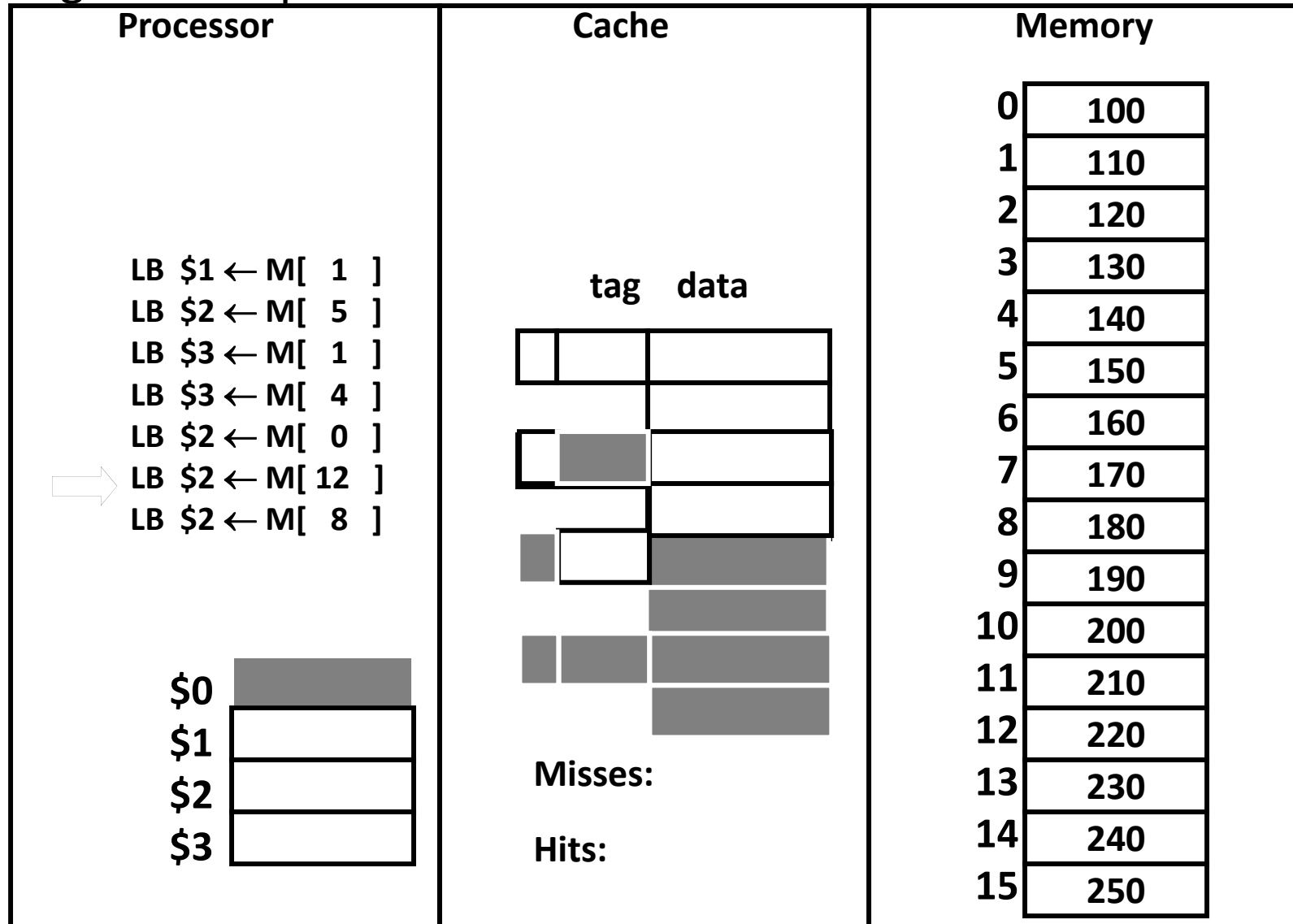
Conflict Misses

- Use a more flexible cache design

Direct Mapped Example: 6th Access

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

Pathological example



Cache Organization

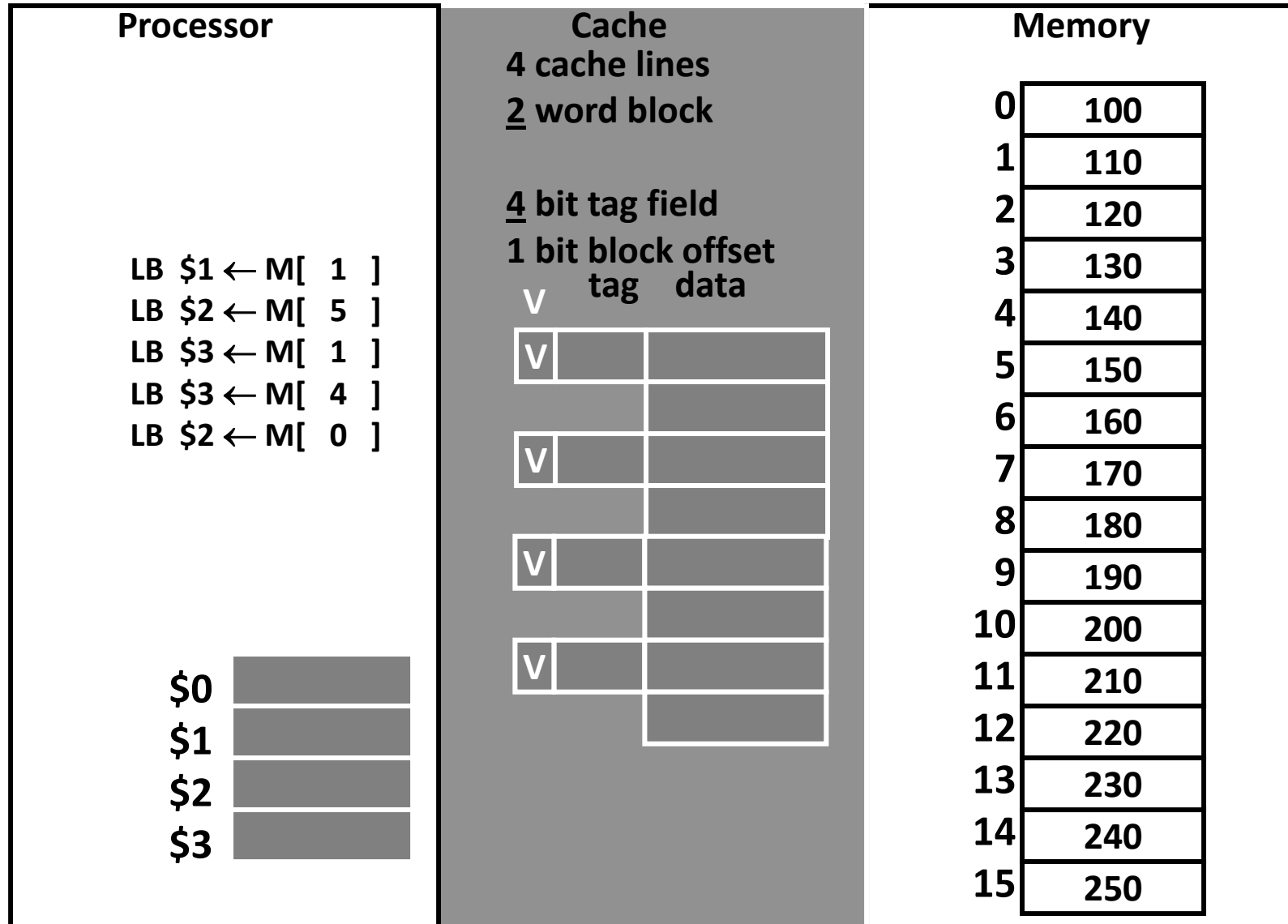
How to avoid Conflict Misses

Three common designs

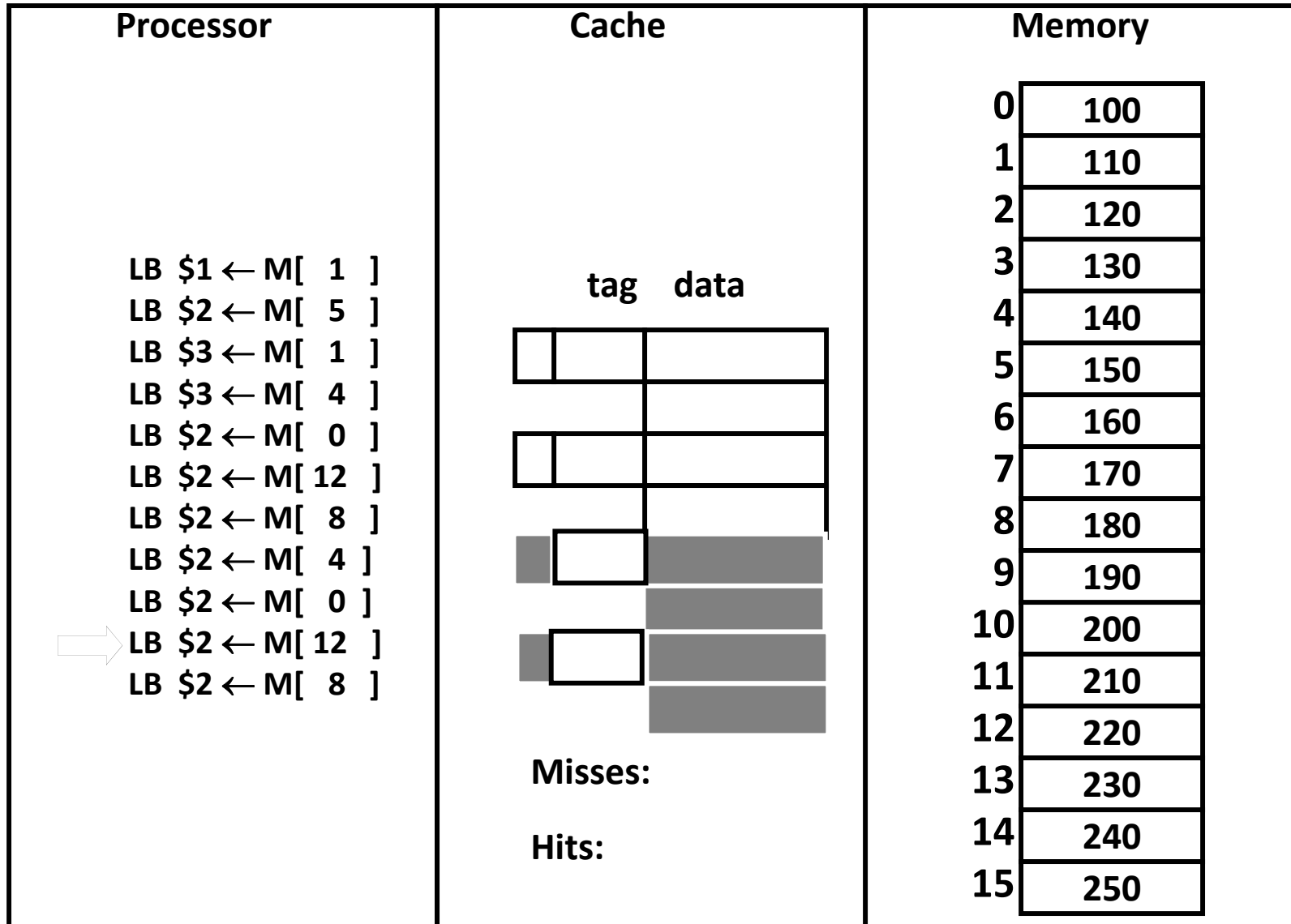
- Fully associative: Block can be anywhere in the cache
- Direct mapped: Block can only be in one line in the cache
- Set-associative: Block can be in a few (2 to 8) places in the cache

Example: Simple Fully Associative Cache

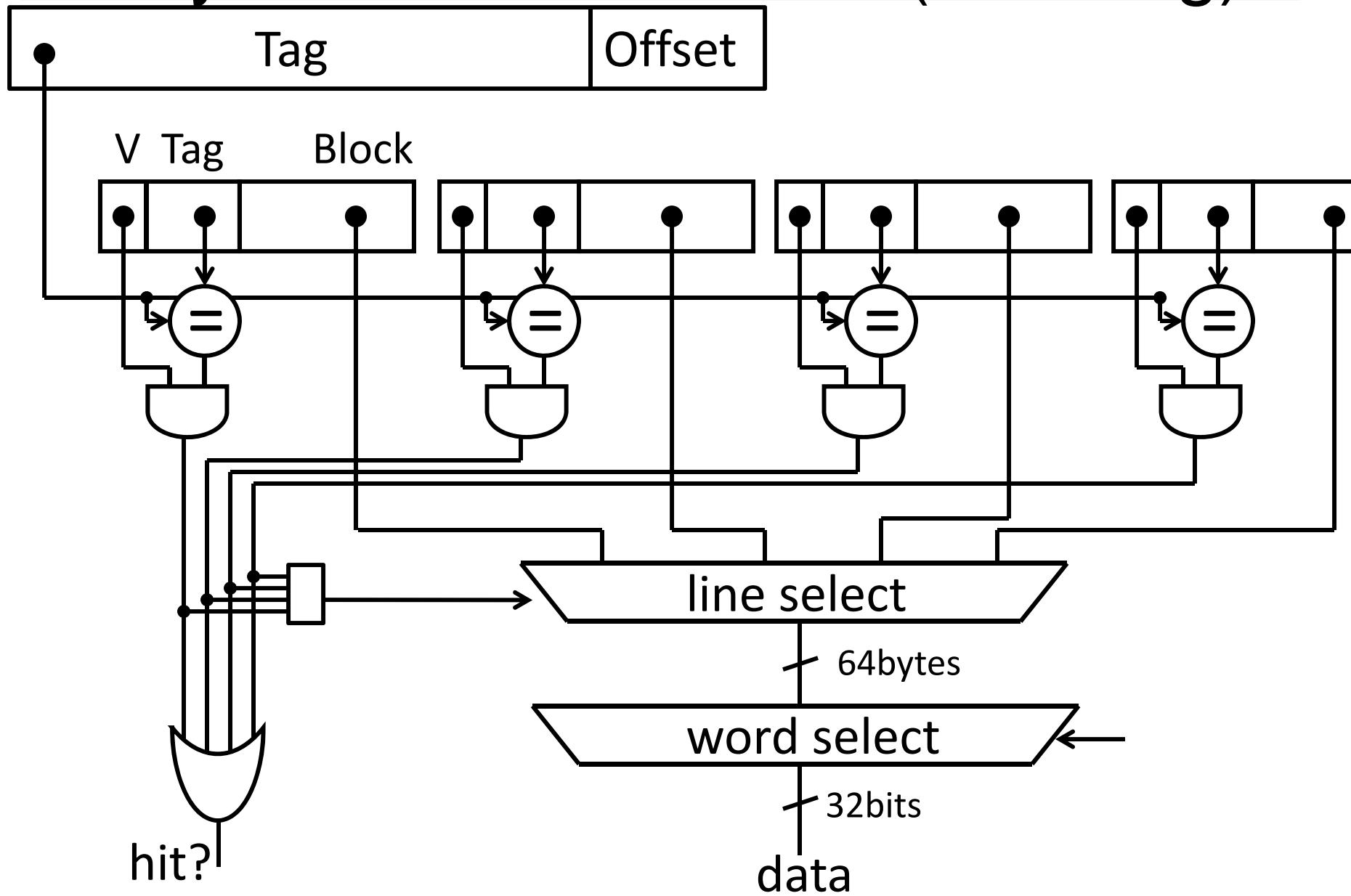
Using **byte addresses** in this example! Addr Bus = 5 bits



10th and 11th Access



Fully Associative Cache (Reading)



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

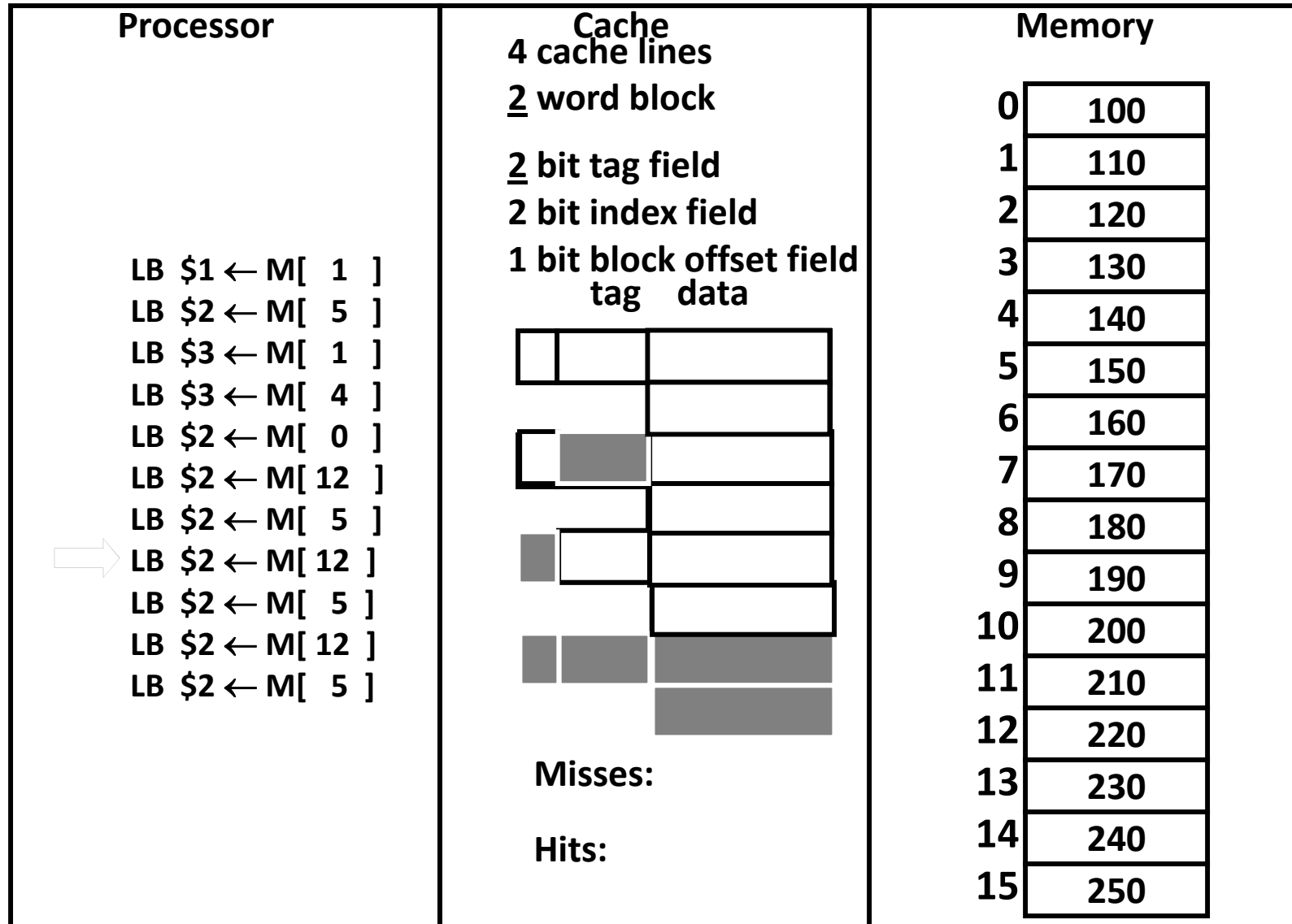
- Index into a location
- Fast

Like a fully-associative cache

- Can store multiple entries
 - decreases thrashing in cache
- Search in each element

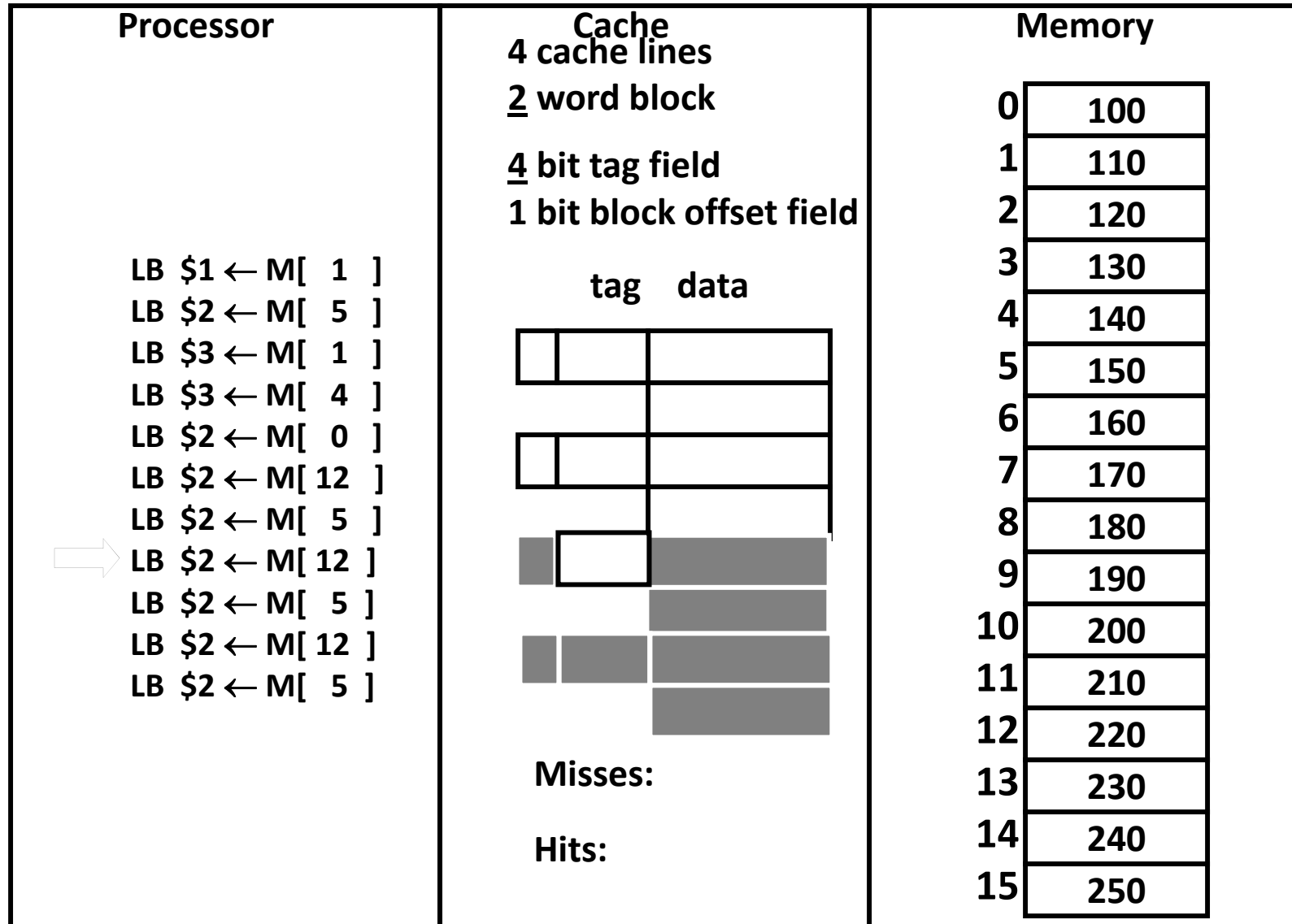
Comparison: Direct Mapped

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



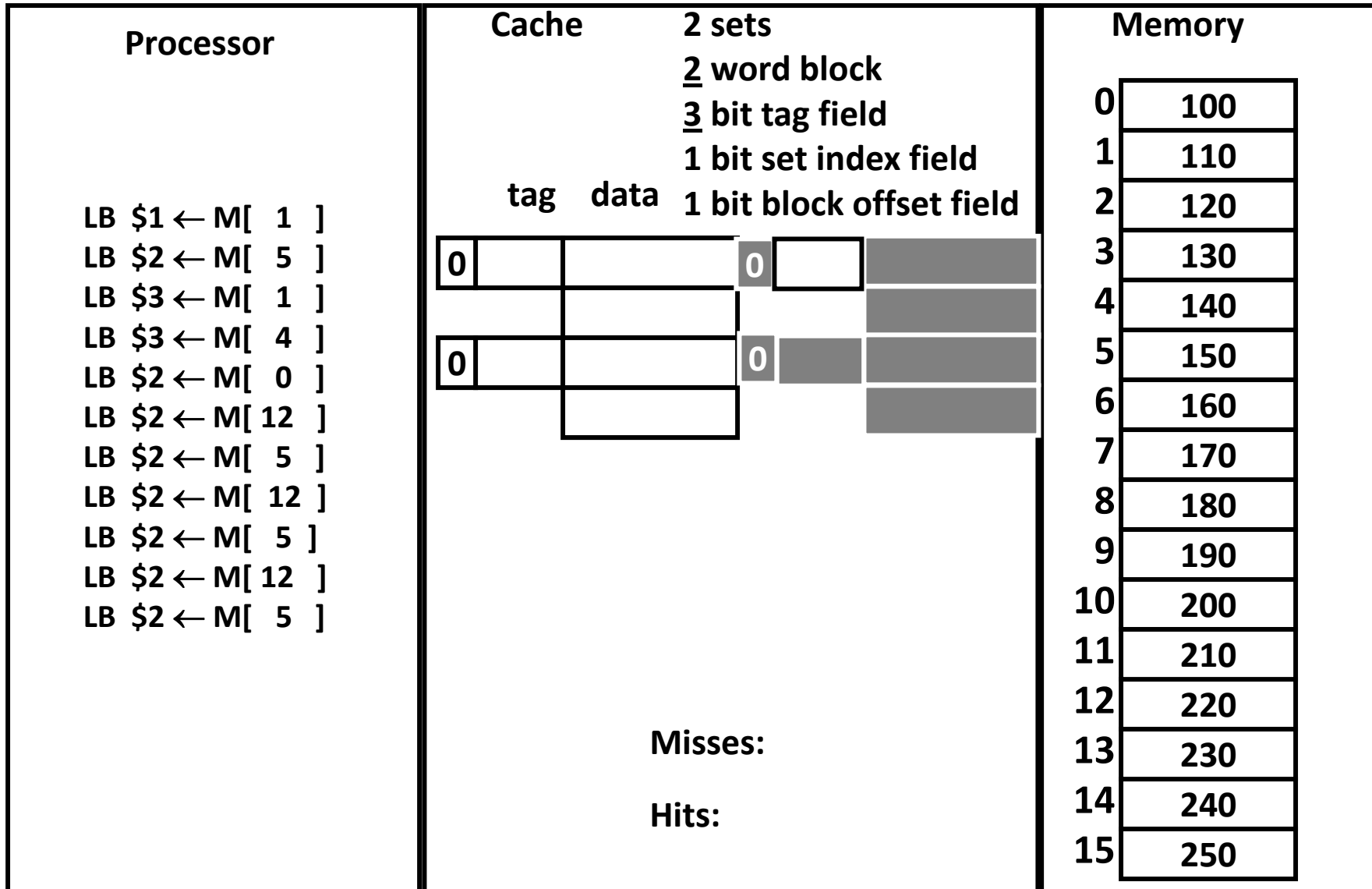
Comparison: Fully Associative

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

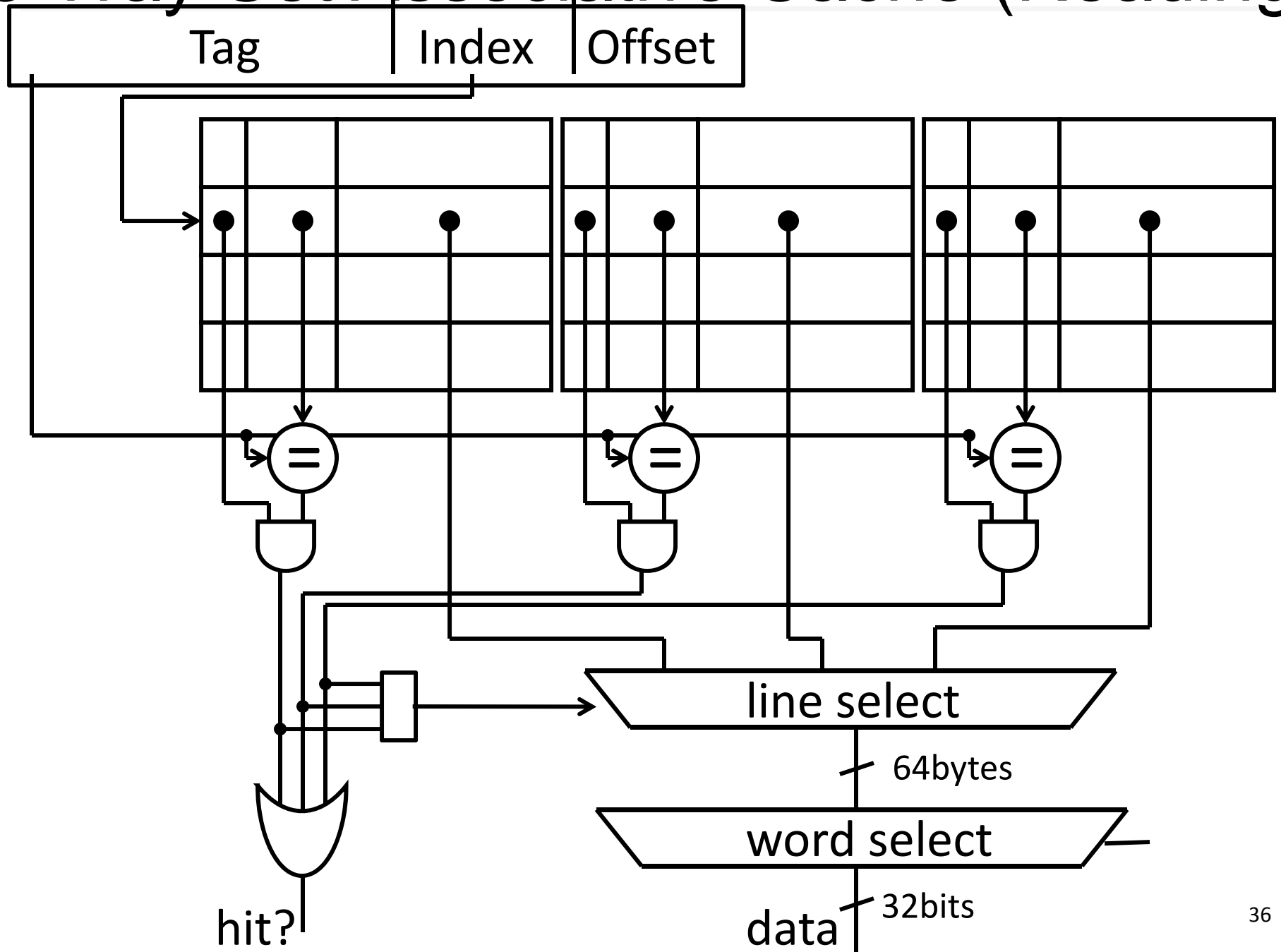


Comparison: 2 Way Set Assoc

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



3-Way Set Associative Cache (Reading)



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