

Caches

Han Wang
CS 3410, Spring 2012
Computer Science
Cornell University

See P&H 5.1, 5.2 (except writes)

Announcements

This week:

- PA2 Work-in-progress submission

Next six weeks:

- Two labs and two projects
- Prelim2 will be Thursday, March 29th

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

Cache organization, parameters and 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

Agenda

- Memory Hierarchy Overview
- The Principle of Locality
- Direct-Mapped Cache
- Fully Associative Cache

Performance

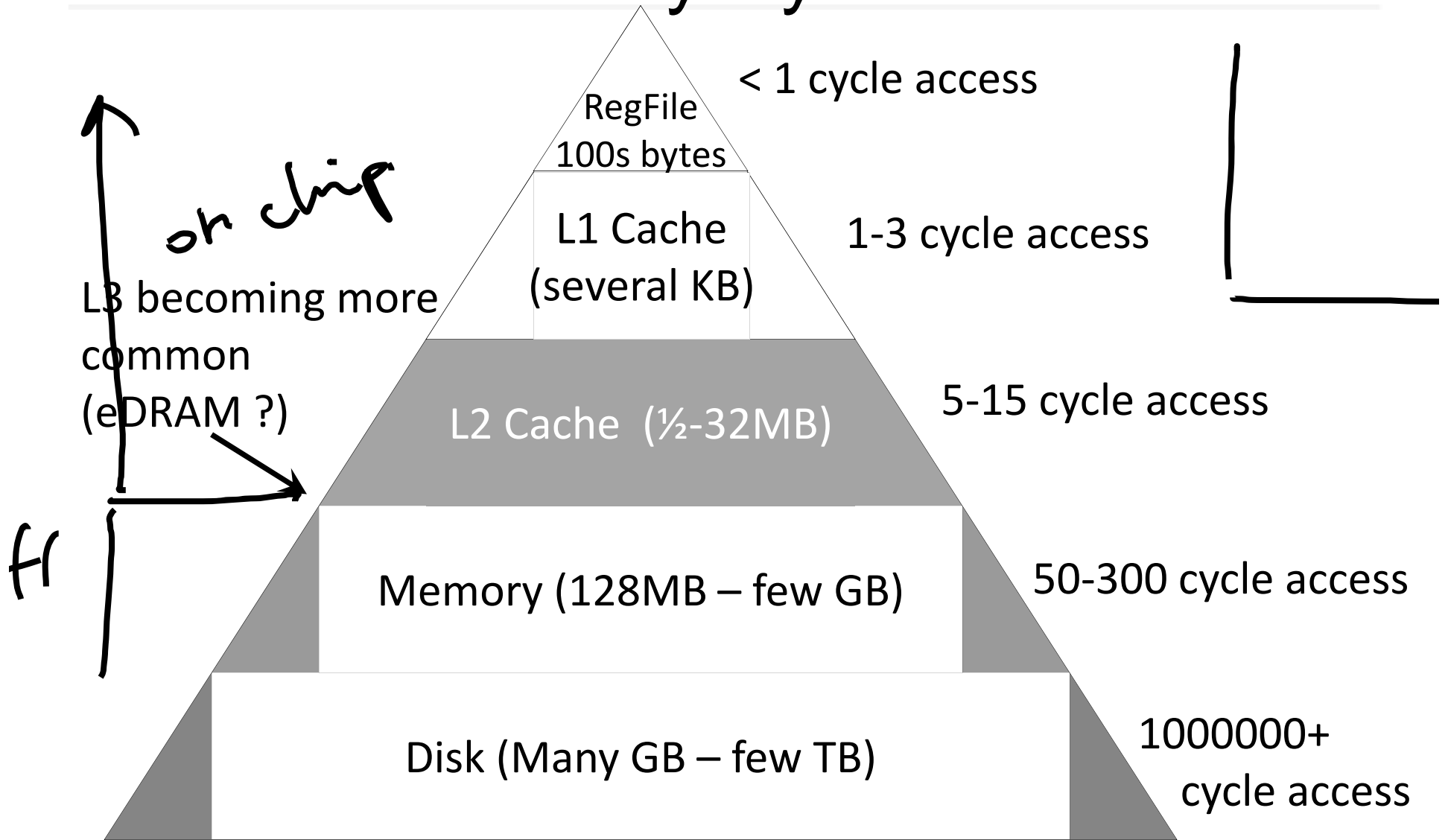
CPU clock rates $\sim 0.2\text{ns} - 2\text{ns}$ (5GHz-500MHz)

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 1-T 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)

A Typical System

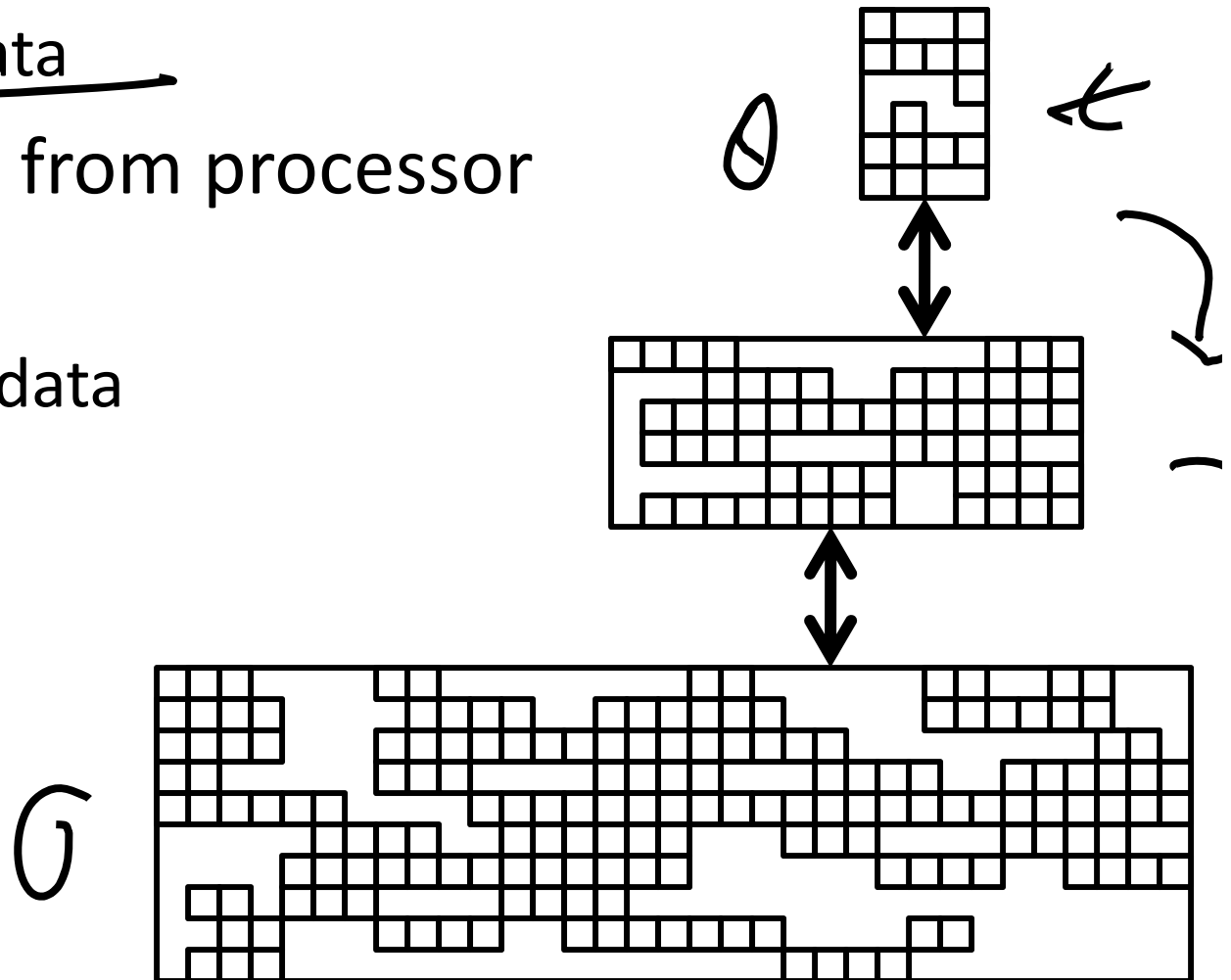
Memory Hierarchy

Memory closer to processor

- small & fast
- stores active data

Memory farther from processor

- big & slow
- stores inactive data



Active vs Inactive Data

Assumption: Most data is not active.

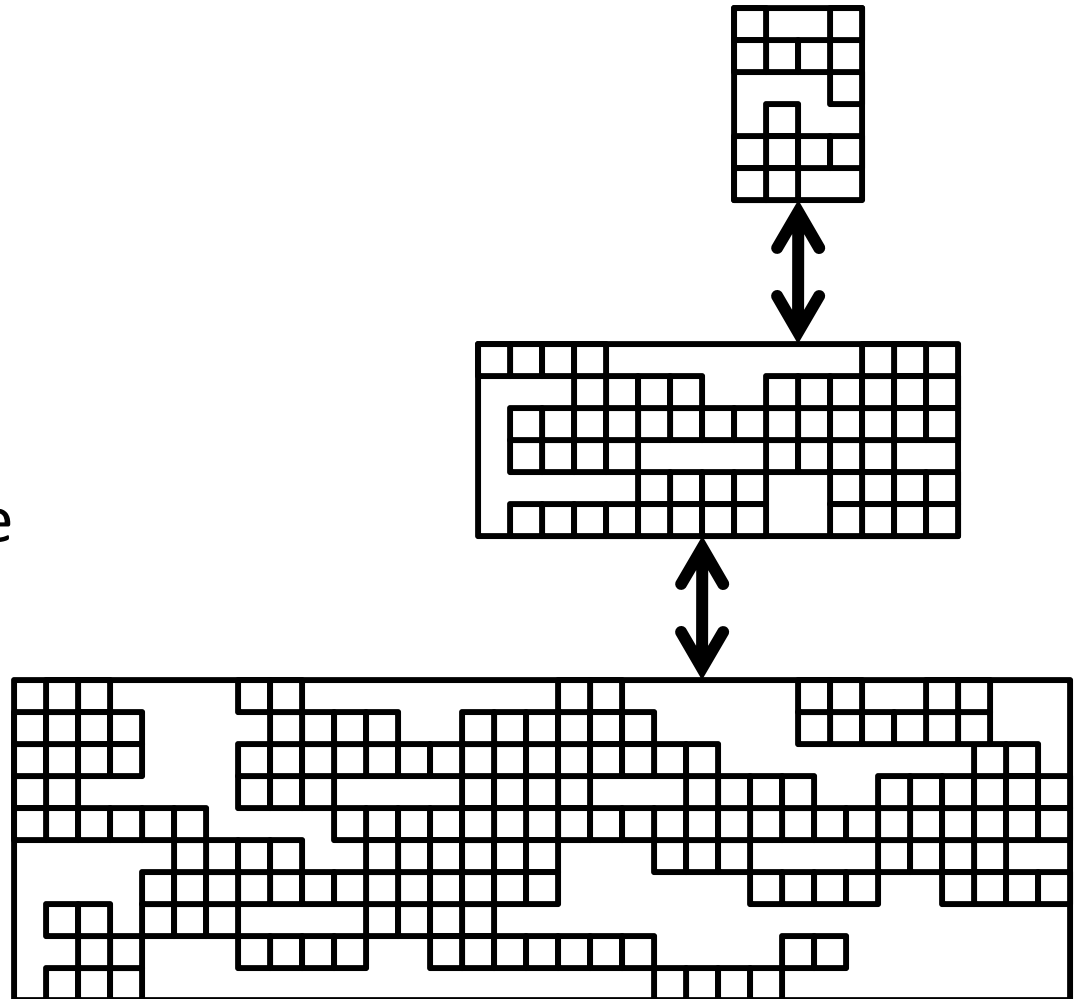
Q: How to decide what is active?

A: Programmer decides

A: Compiler decides

A: OS decides at run-time

A: Hardware decides
at run-time



Insight of Caches

Q: What is “active” data?

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

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

- Exploit temporal locality:

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

- Exploit spatial locality:

Locality Analogy

- Writing a report on a specific topic.
- While at library, check out books and keep them on desk.
- If need more, check them out and bring to desk.
 - But don't return earlier books since might need them
 - Limited space on desk; Which books to keep?
- You hope this collection of ~20 books on desk enough to write report, despite 20 being only 0.00002% of books in Cornell libraries

Two types of Locality

Temporal Locality (locality in time)

- If a memory location is referenced then it will tend to be referenced again soon
- ⇒ Keep most recently accessed data items closer to the processor

Spatial Locality (locality in space)

- If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon
- ⇒ Move blocks consisting of contiguous words closer to the processor

Locality

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");
    }
}
```

Memory Hierarchy

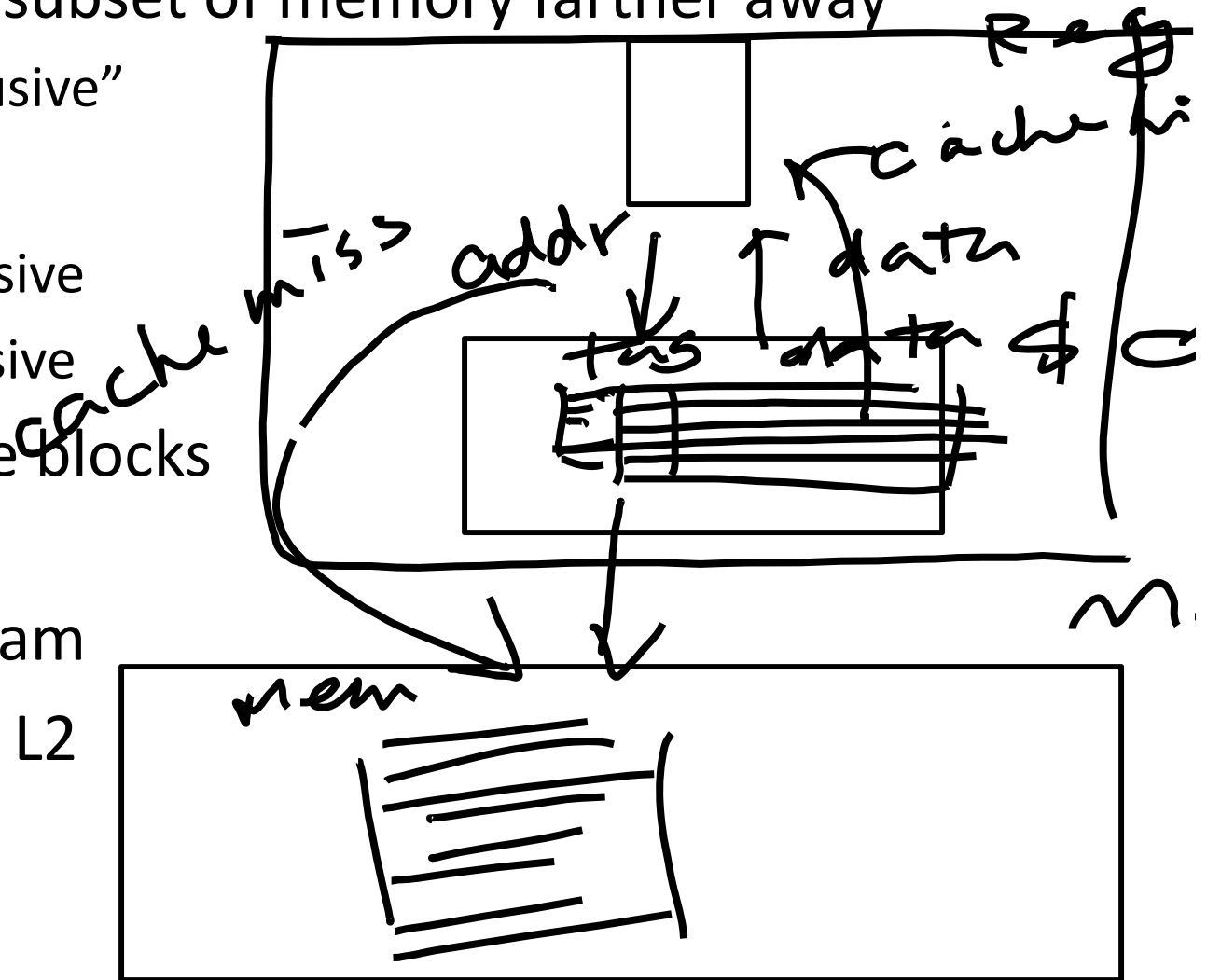
Memory closer to processor is fast but small

- usually stores subset of memory farther away
 - “strictly inclusive”
- alternatives:
 - strictly exclusive
 - mostly inclusive
- Transfer whole blocks (cache lines):

4kb: disk ↔ ram

256b: ram ↔ L2




64b: L2 ↔ L1



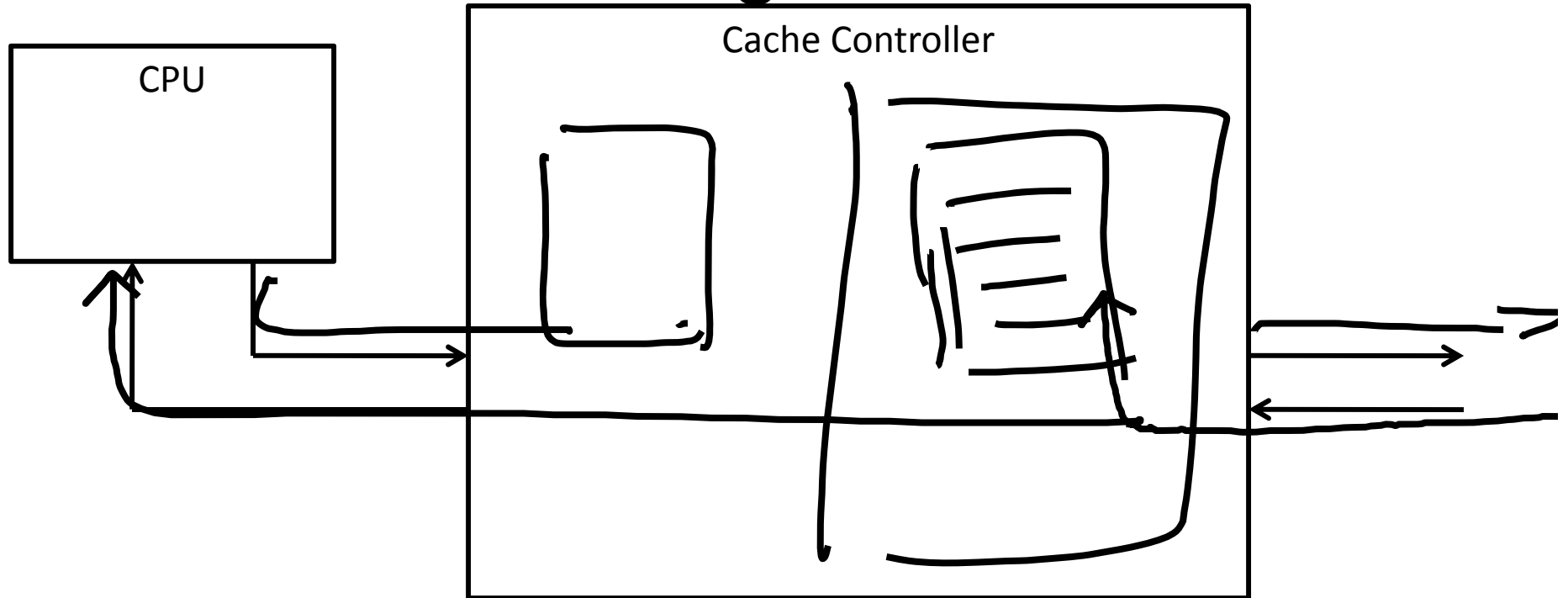
Cache Lookups (Read)

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

Cache Organization



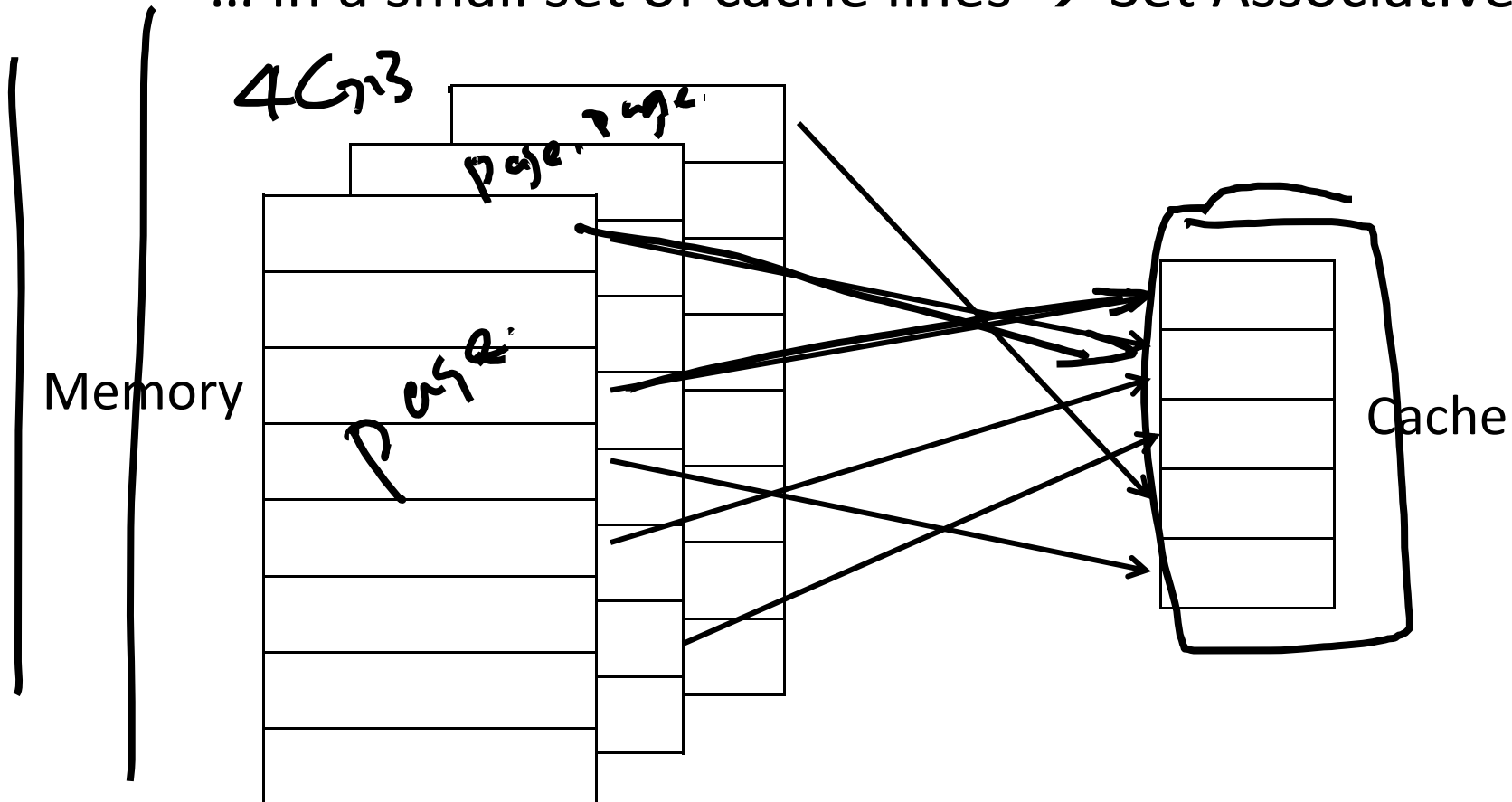
Cache has to be fast and dense

- Gain speed by performing lookups in parallel
 - but requires die real estate for lookup logic
- Reduce lookup logic by limiting where in the cache a block might be placed
 - but might reduce cache effectiveness

Three common designs

A given data block can be placed...

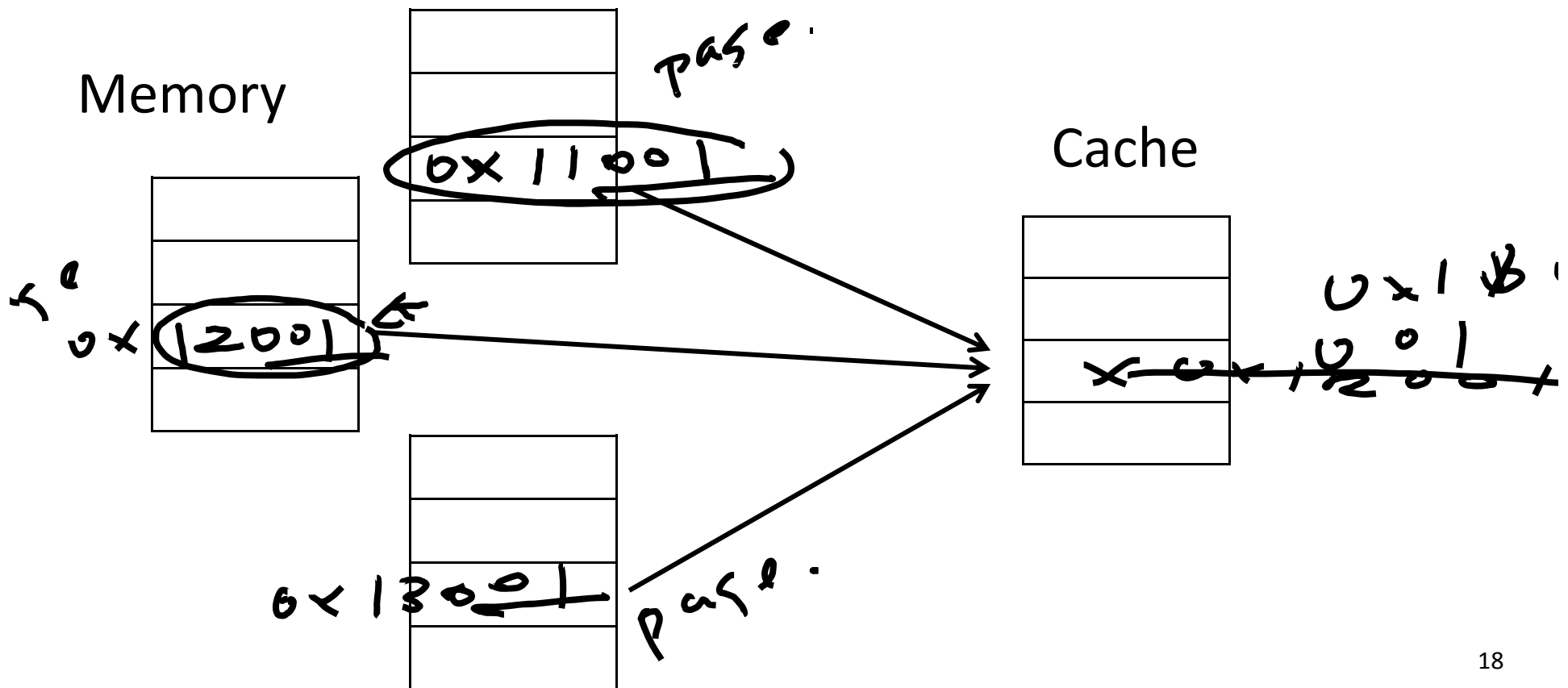
- ... in any cache line → Fully Associative
- ... in exactly one cache line → Direct Mapped
- ... in a small set of cache lines → Set Associative



Three common designs

A given data block can be placed...

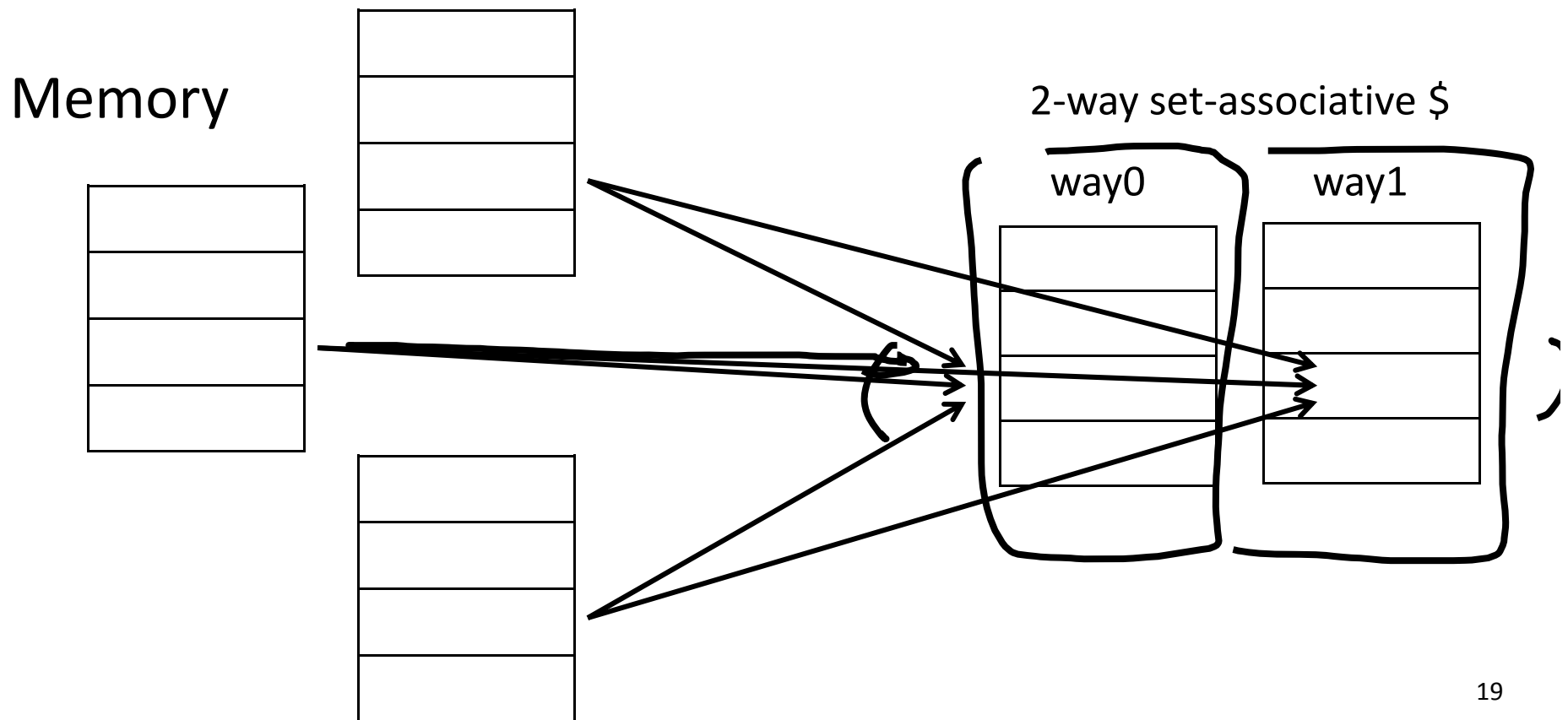
- ... in any cache line → Fully Associative
- ... in exactly one cache line → Direct Mapped
- ... in a small set of cache lines → Set Associative



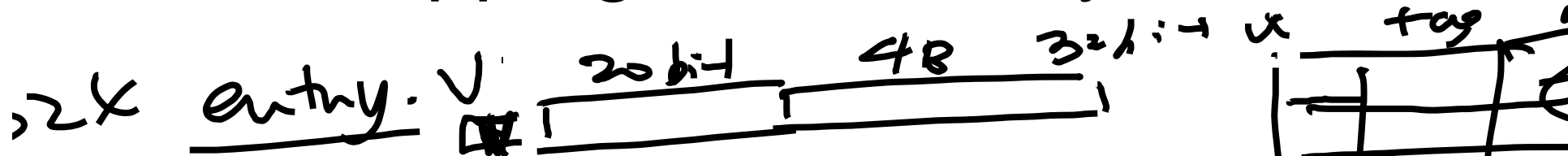
Three common designs

A given data block can be placed...

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

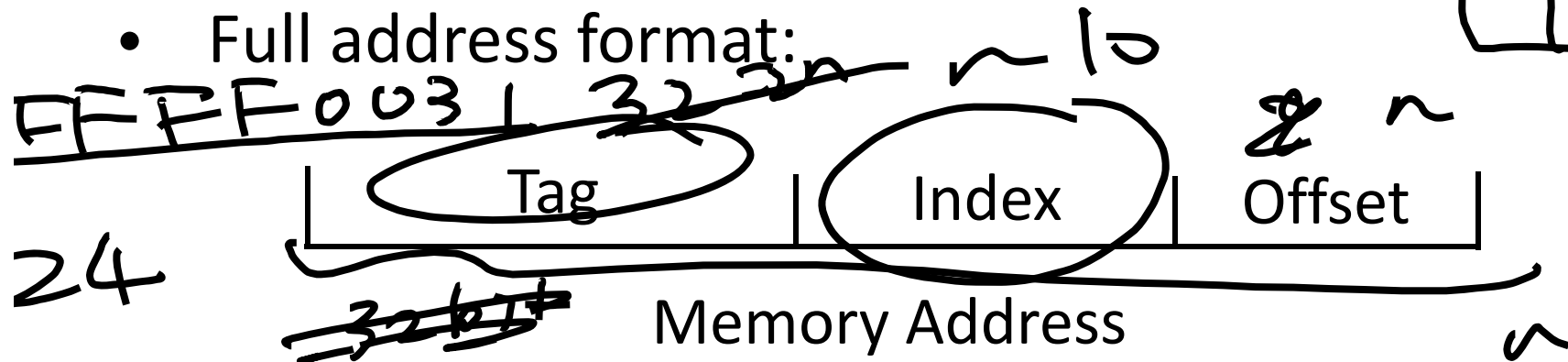


TIO: Mapping the Memory Address



- Lowest bits of address (*Offset*) determine *which byte within a block* it refers to.

- Full address format:



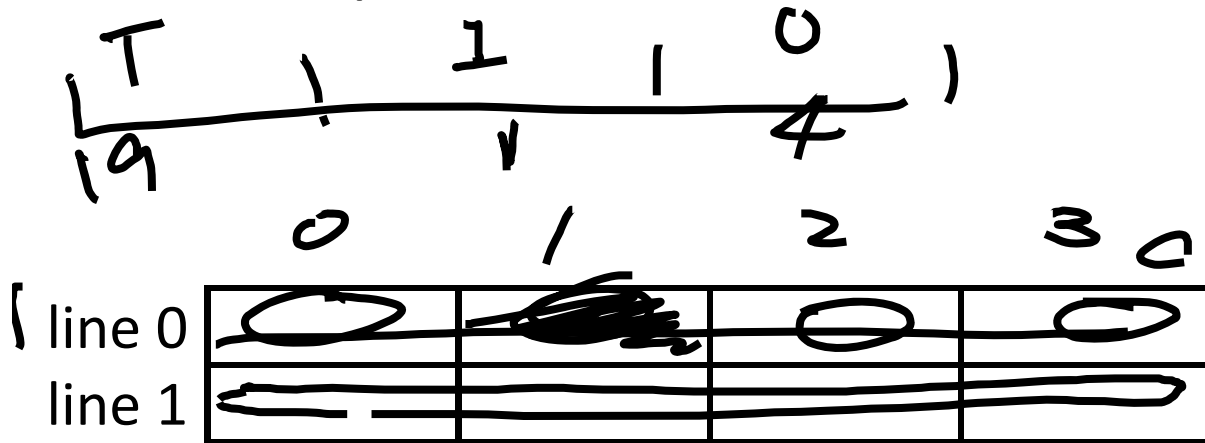
- n-bit Offset means a block is how many bytes?
- n-bit Index means cache has how many blocks?

$$2^n$$

Direct Mapped Cache

Direct Mapped Cache

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



0x000000

0x000004

0x000008

0x00000c

0x000010

0x000014

0x000018

0x00001c

0x000020

0x000024

0x000028

0x00002c

0x000030

0x000034

0x000038

0x00003c

0x000040

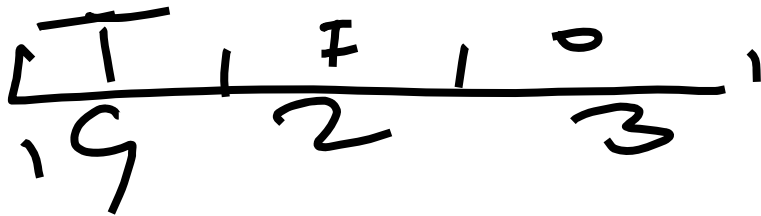
0x000044

0x000048

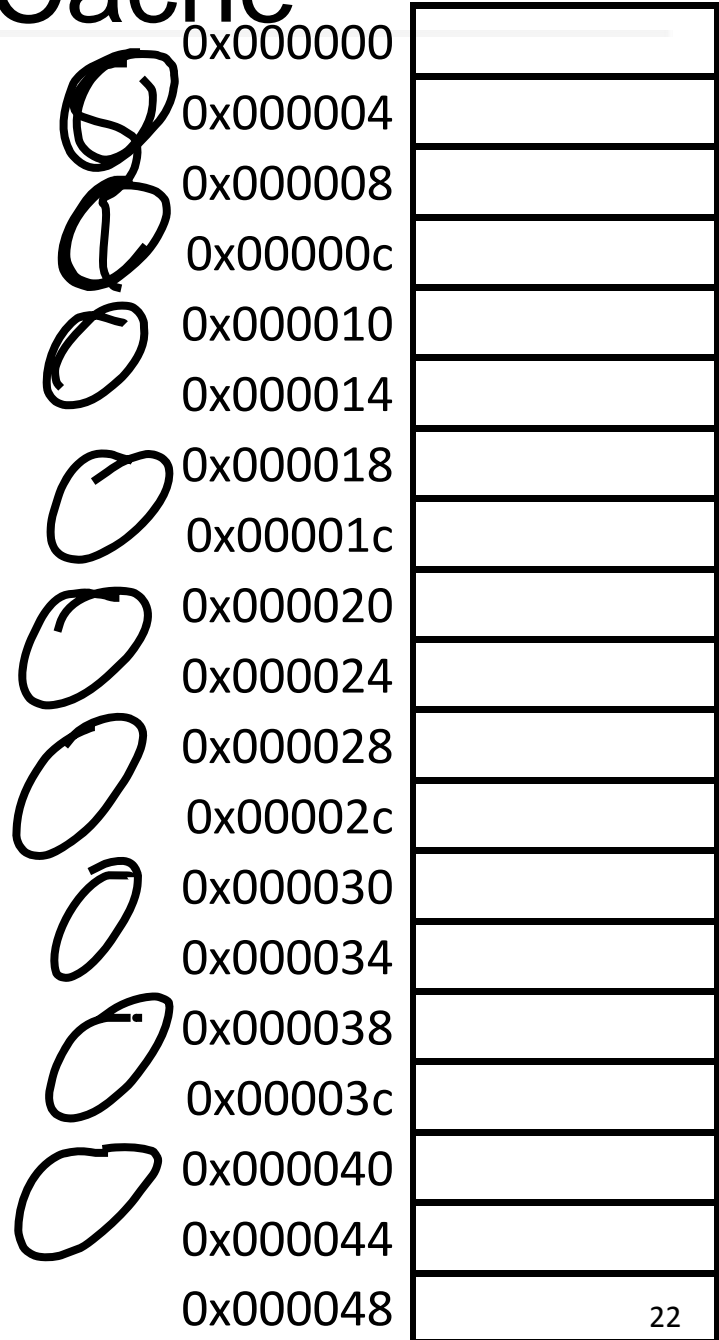
Direct Mapped Cache

Direct Mapped Cache

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



line 0		
line 1		
line 2		
line 3		



Tags and Offsets

Assume sixteen 64-byte cache lines

0x7FFF3D4D

= 0111 1111 1111 1111 0011 1101 0100 1101

T

I

O

$\log_2 64$

Need meta-data for each cache line:

- • valid bit: is the cache line non-empty?
- • tag: which block is stored in this line (if valid)

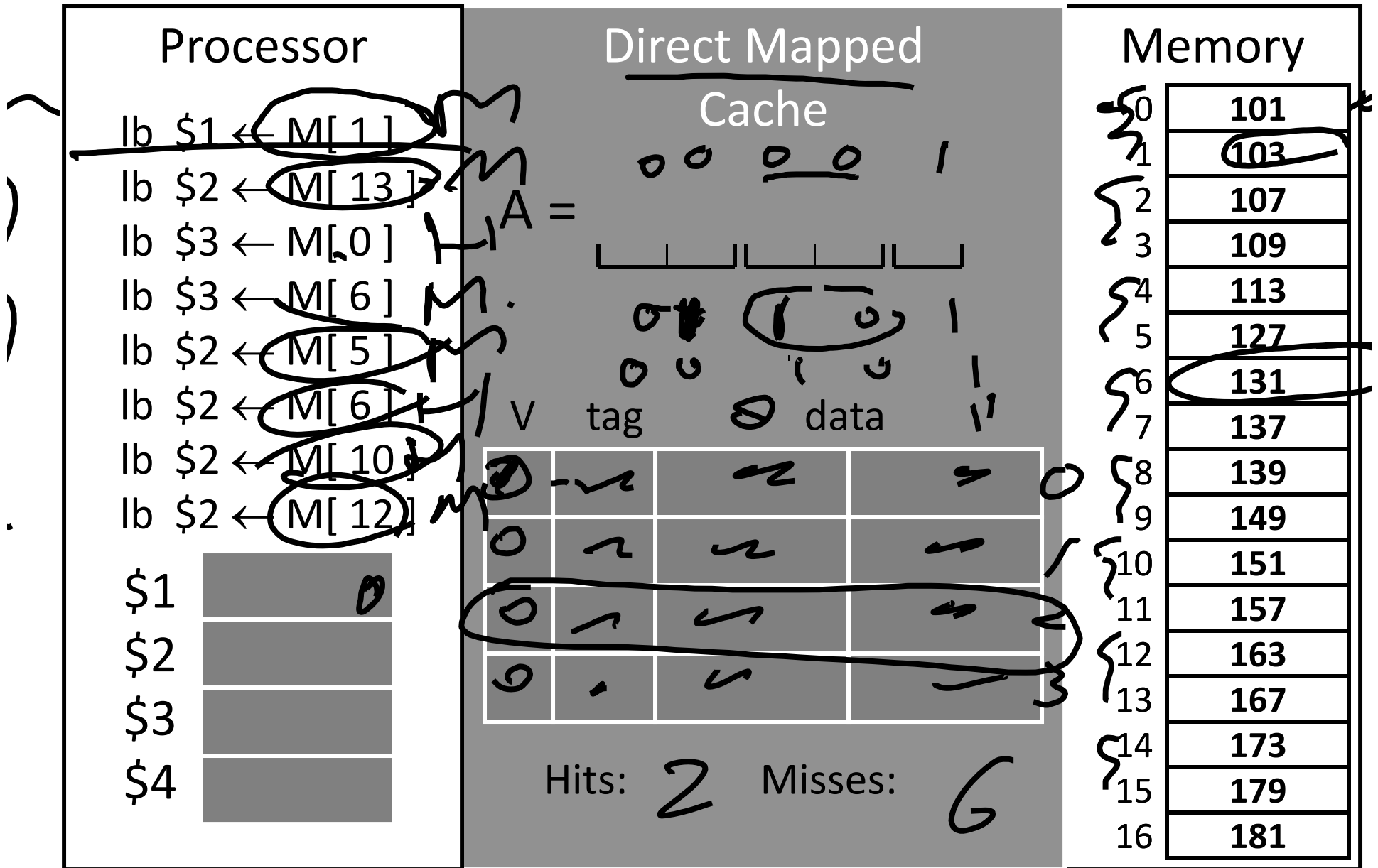
Q: how to check if X is in the cache?

Q: how to clear a cache line?

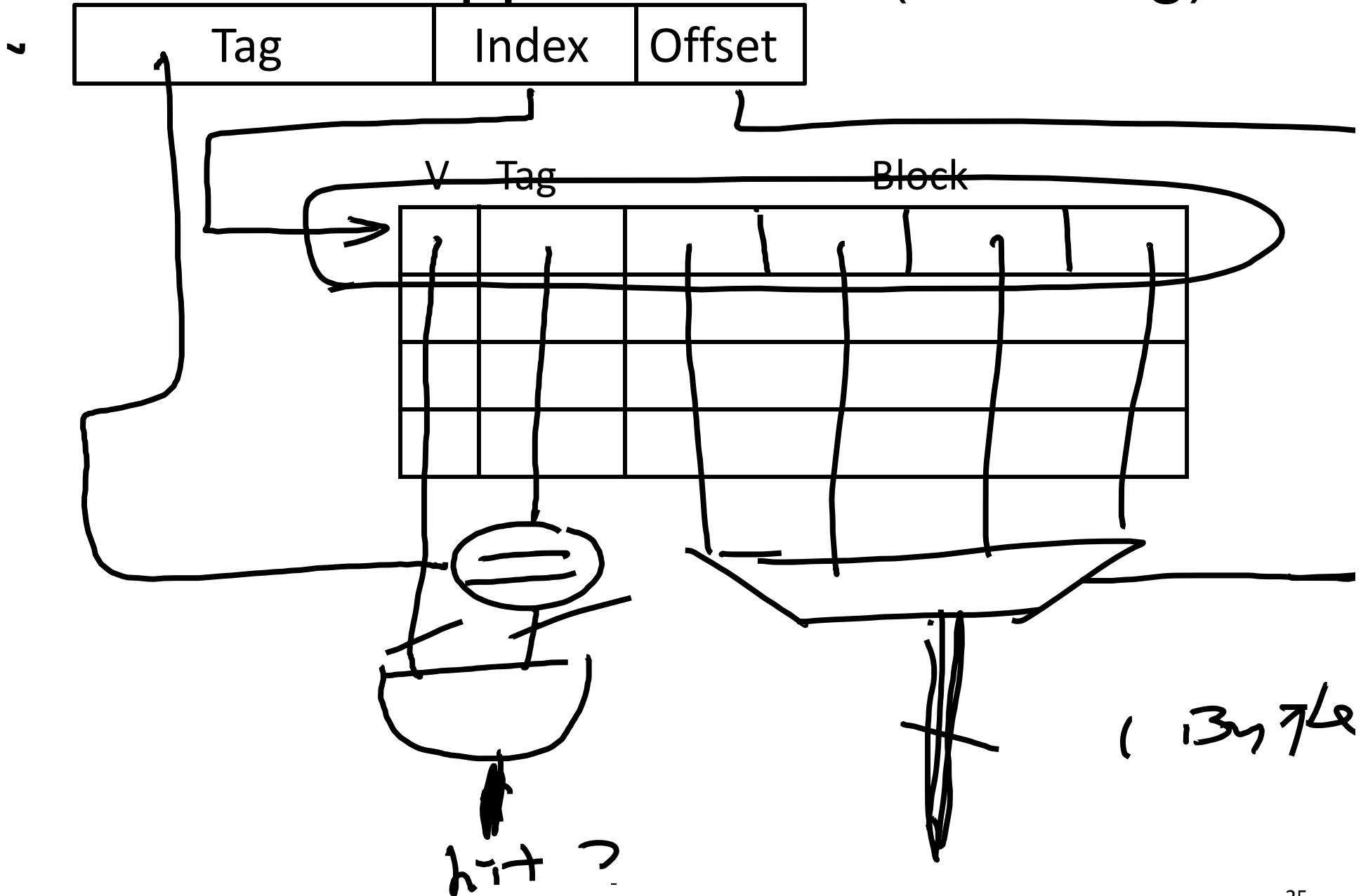
→

A Simple Direct Mapped Cache

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



Direct Mapped Cache (Reading)



Direct Mapped Cache Size



n bit index, m bit offset

Q: How big is cache (data only)?

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

Cache Performance

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

Mem (DRAM): 4GB

Data cost: 50 cycle per word, plus 3 cycle per consecutive word

$$T = T_{hit} \times \% + T_{miss} \times \%$$

$$T_{hit} = 5$$

$$T_{miss} = 5 + 50 + 15 \times 3 = 90$$

Performance depends on:

$$\text{Access time for hit, miss penalty, hit rate} = 5 \times 90\% + 100 \times 10\% = 14.5$$

Misses

Cache misses: classification

The line is being referenced for the first time

- Cold (aka Compulsory) Miss

The line was in the cache, but has been evicted

Avoiding Misses

Q: How to avoid...

Cold Misses

- Unavoidable? The data was never in the cache...
- Prefetching!

Other Misses

- Buy more SRAM
- Use a more flexible cache design

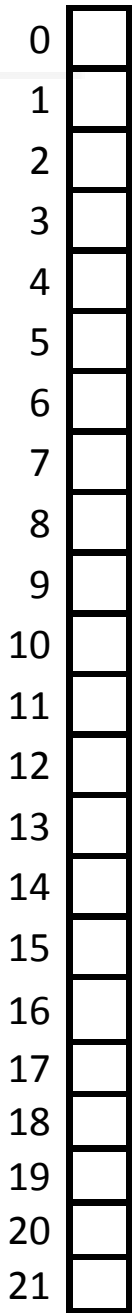
Bigger cache doesn't always help...

Mem access trace: 0, 16, 1, 17, 2, 18, 3, 19, 4, ...

Hit rate with four direct-mapped 2-byte cache lines?

With eight 2-byte cache lines?

With four 4-byte cache lines?



Misses

Cache misses: classification

The line is being referenced for the first time

- Cold (aka Compulsory) Miss

The line was in the cache, but has been evicted...

... because some other access with the same index

- Conflict Miss

... because the cache is too small

- i.e. the *working set* of program is larger than the cache
- Capacity Miss

Avoiding Misses

Q: How to avoid...

Cold Misses

- Unavoidable? The data was never in the cache...
- Prefetching!

Capacity Misses

- Buy more SRAM

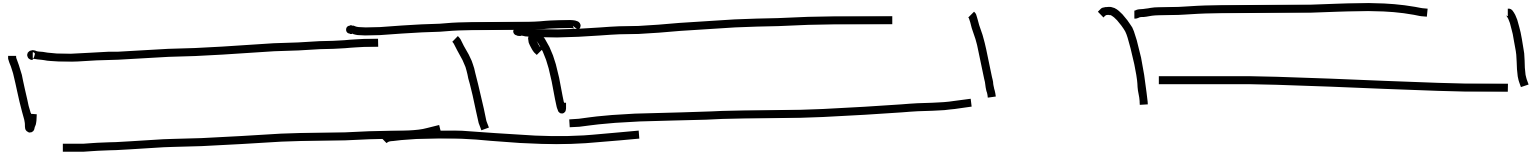
Conflict Misses

- Use a more flexible cache design

Three common designs

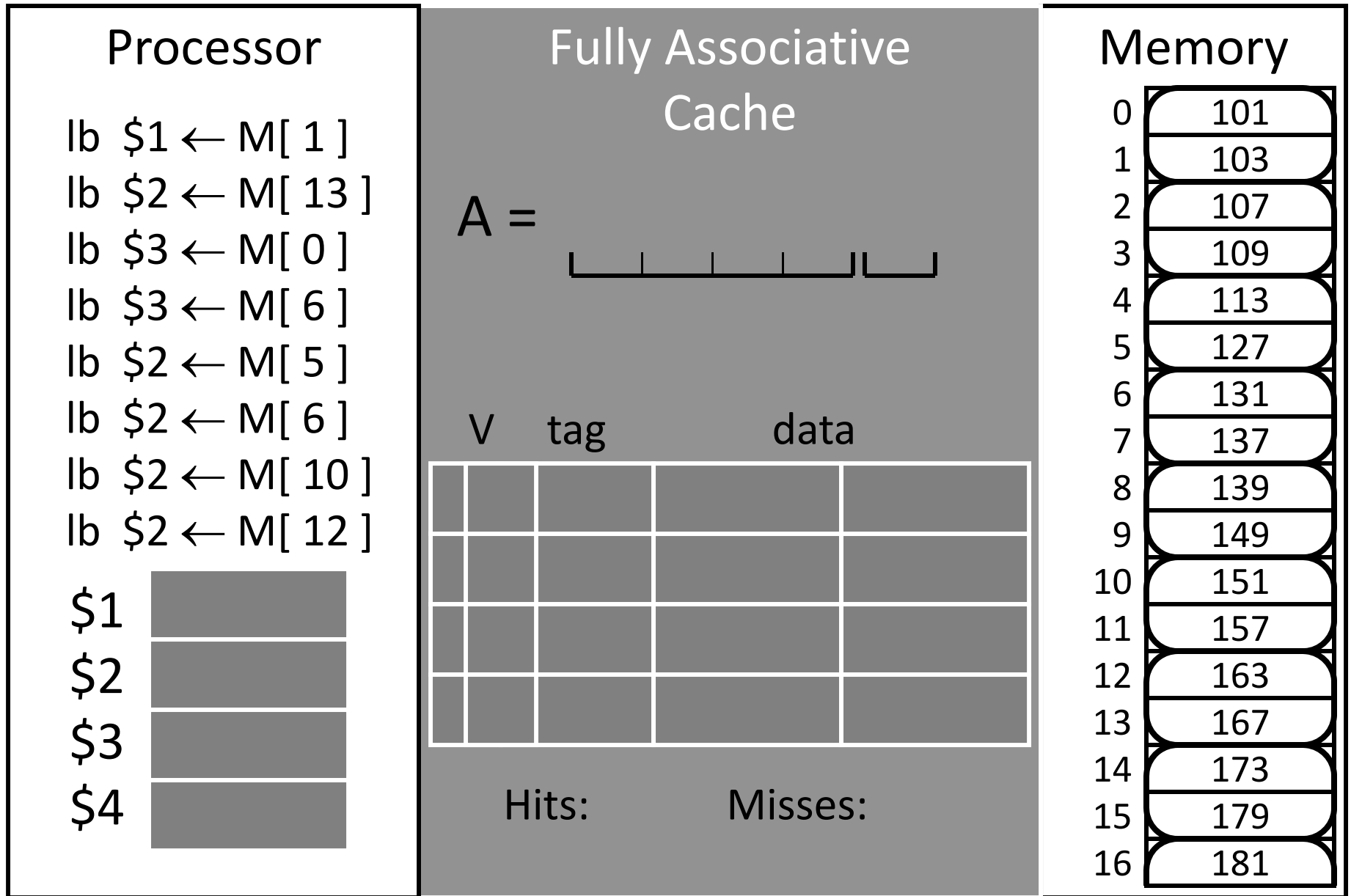
A given data block can be placed...

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

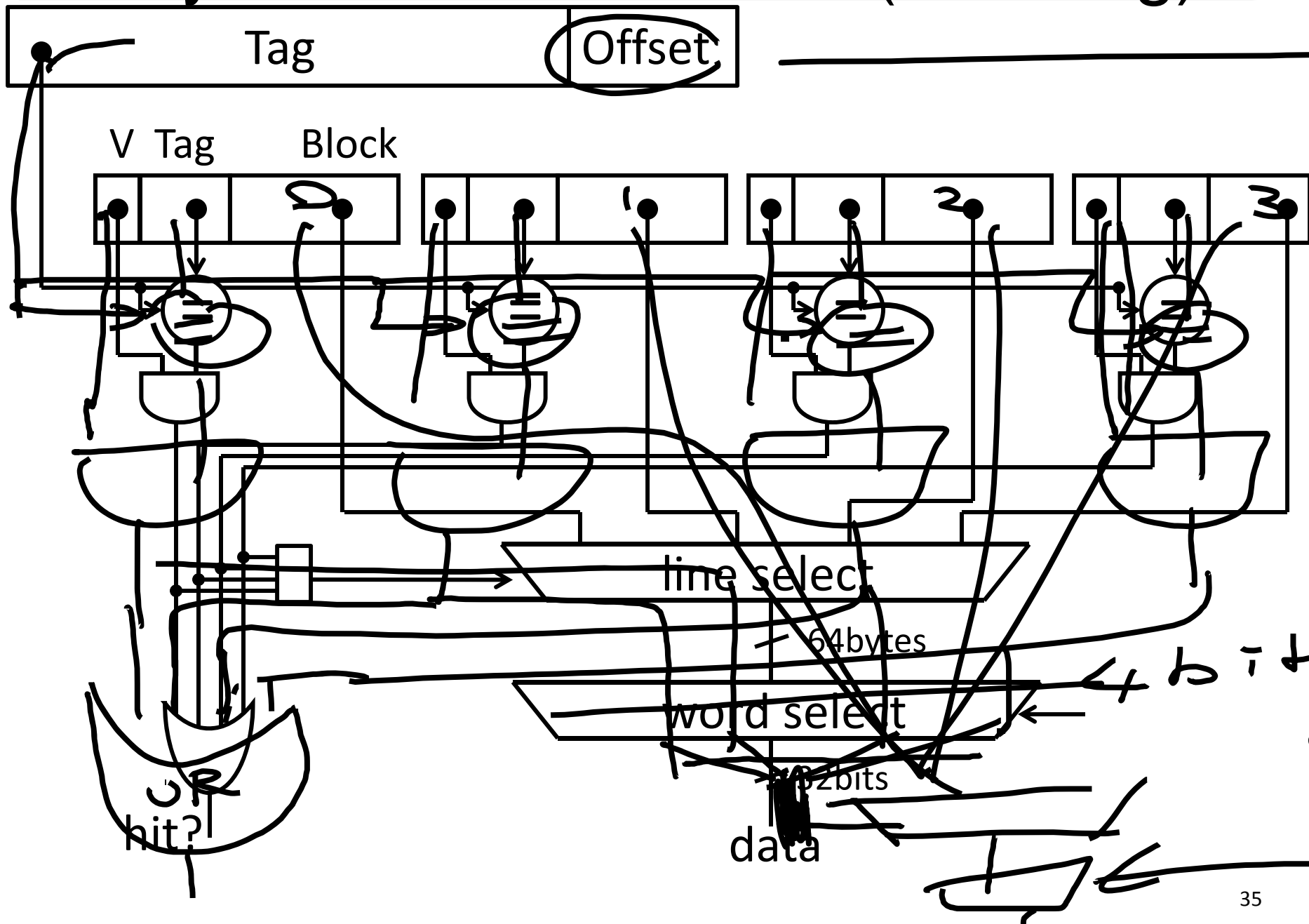


A Simple Fully Associative Cache

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



Fully Associative Cache (Reading)



Fully Associative Cache Size



m bit offset , 2^n cache lines

Q: How big is cache (data only)?

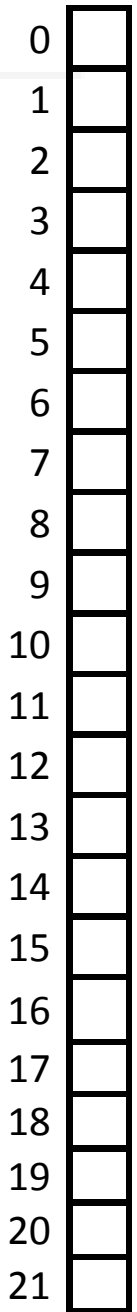
Q: How much SRAM needed (data + overhead)?

Fully-associative reduces conflict misses...

... assuming good eviction strategy

Mem access trace: 0, 16, 1, 17, 2, 18, 3, 19, 4, 20, ...

Hit rate with four fully-associative 2-byte cache lines?



... but large block size can still reduce hit rate

vector add trace: 0, 100, 200, 1, 101, 201, 2, 202, ...

Hit rate with four fully-associative 2-byte cache lines?

With two fully-associative 4-byte cache lines?

Misses

Cache misses: classification

Cold (aka Compulsory)

- The line is being referenced for the first time

Capacity

- The line was evicted because the cache was too small
- i.e. the *working set* of program is larger than the cache

Conflict

- The line was evicted because of another access whose index conflicted

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