Caches

Hakim Weatherspoon
CS 3410, Spring 2011
Computer Science
Cornell University

See P&H 5.1, 5.2 (except writes)

Announcements

HW3 available due *next* Tuesday

- Work with alone ~~partner~~
- Be responsible with new knowledge

Use your resources

- FAQ, class notes, book, Sections, office hours, newsgroup, CSUGLab

Next six weeks

- Two homeworks and two projects
- *Optional* prelim1 tomorrow, Wednesday, in Philips 101
- Prelim2 will be Thursday, April 28th
- PA4 will be final project (no final exam)

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
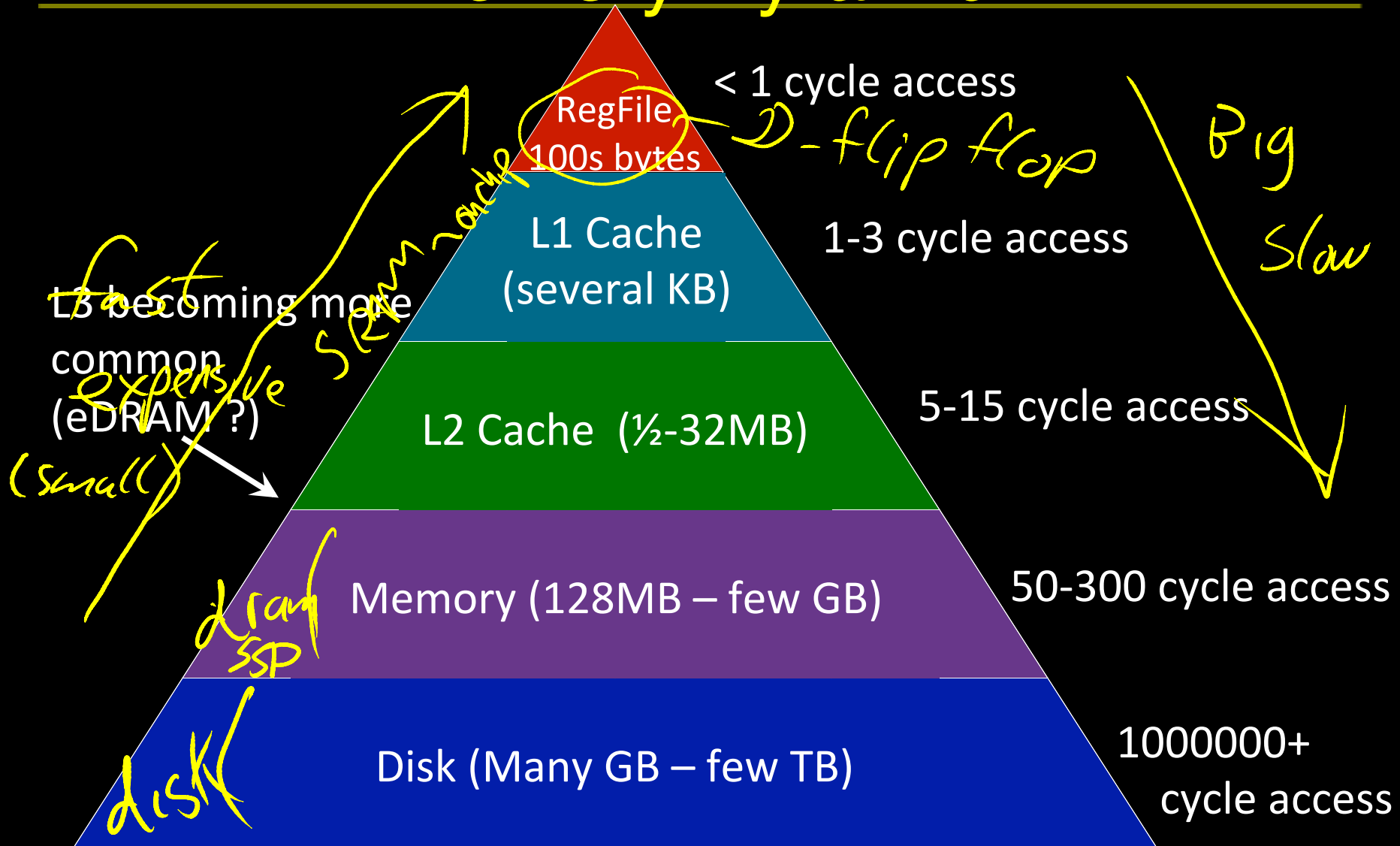- Larger block size → lower hit cost, higher miss penalty

Performance

CPU clock rates ~0.2ns – 2ns (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)

(handwritten annotations: Capacity to latency inversely prop)

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

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

Memory Pyramid

RegFile
100s bytes

D-flip flop

< 1 cycle access

Big

Slow

L1 Cache
(several KB)

1-3 cycle access

L3 becoming more
common
(eDRAM ?)

fast

expensive SRAM caches

(small)

L2 Cache (½-32MB)

5-15 cycle access

dram
SSD

Memory (128MB – few GB)

50-300 cycle access

disk

Disk (Many GB – few TB)

1000000+
cycle access

These are rough numbers: mileage may vary for latest/greatest
Caches usually made of SRAM (or eDRAM)

5

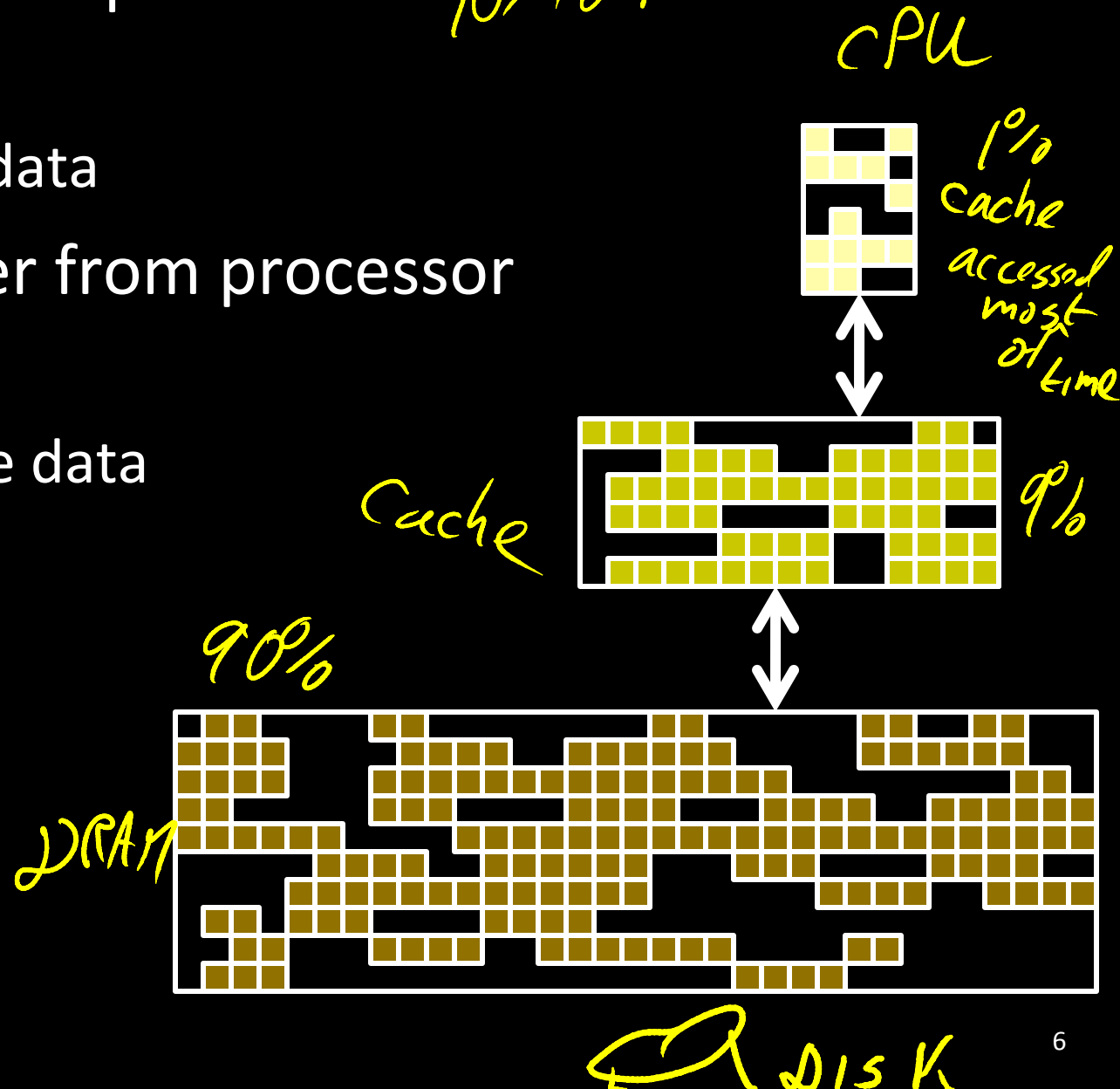# Memory Hierarchy

Memory closer to processor

- small & fast

- stores active data

Memory farther from processor

- big & slow

- stores inactive data

90/10 rule

CPU

1% cache accessed most of time

Cache

90%

90%

DRAM

DISK

Active vs Inactive Data

Assumption: Most data is not active.
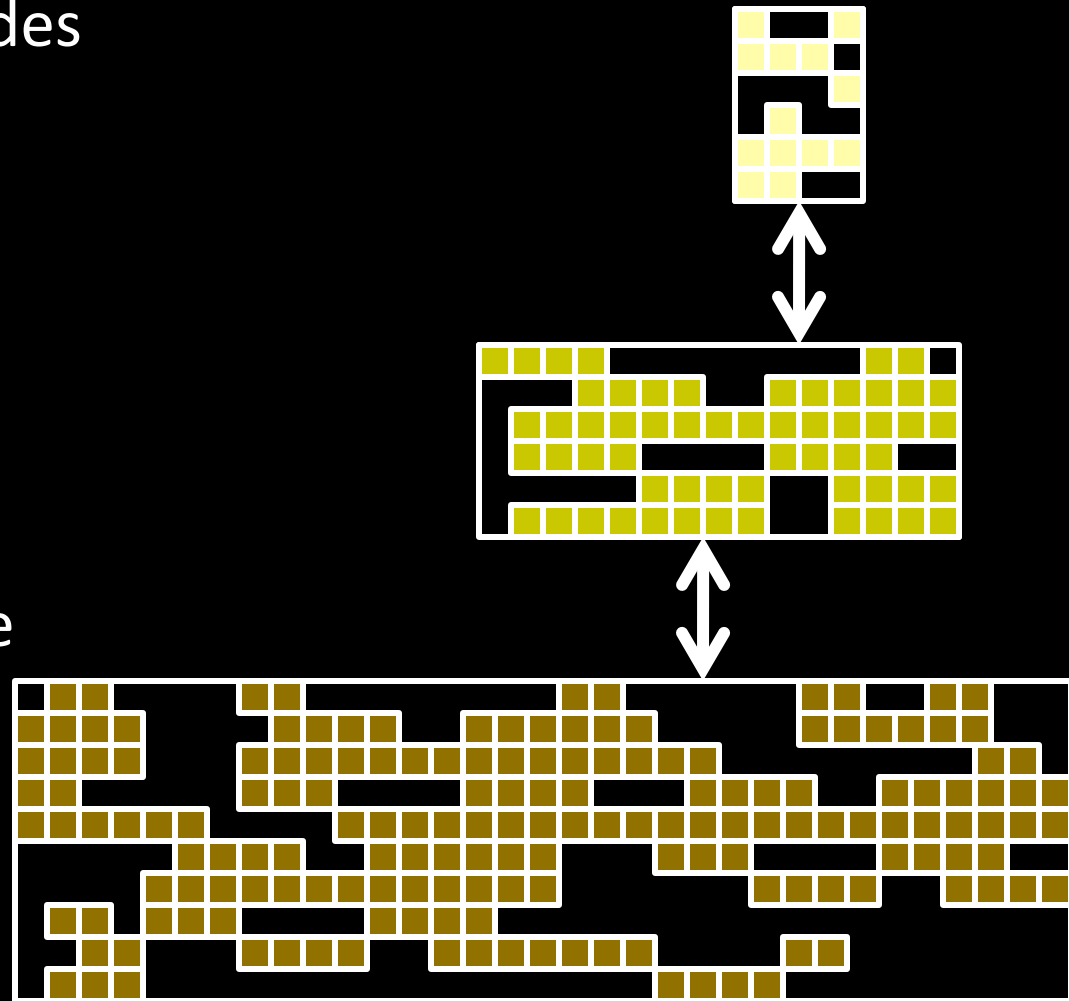
Q: How to decide what is active?

A: Some committee decides

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?

data that will be accessed soon

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

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

- Exploit spatial locality:

  Put entire block containing Mem[x] higher

Locality

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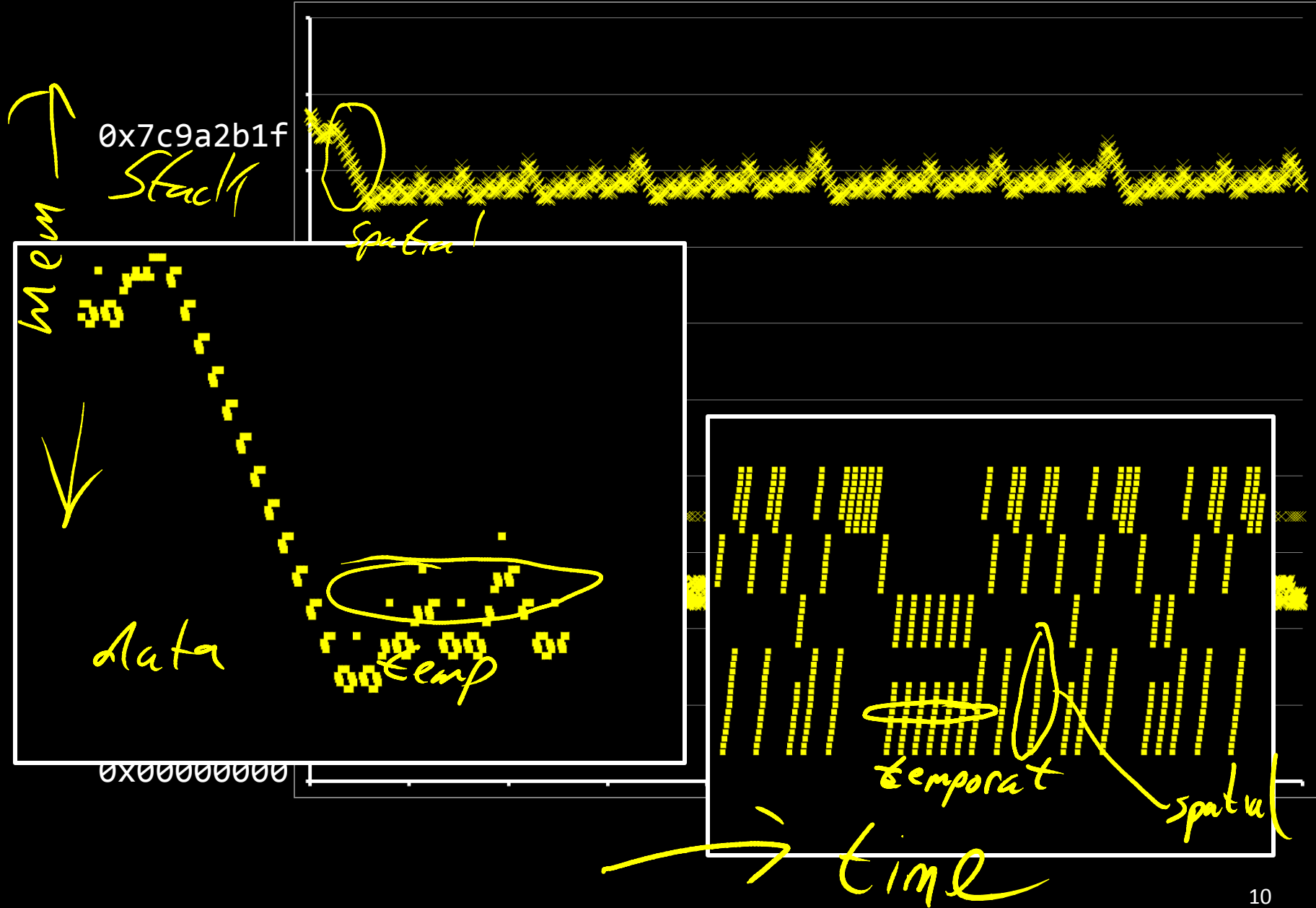
high
Stack
low

Stack
temp
local

Spatial

```c
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");
    }
}
```

9

Memory Hierarchy

Memory closer to processor is fast but small

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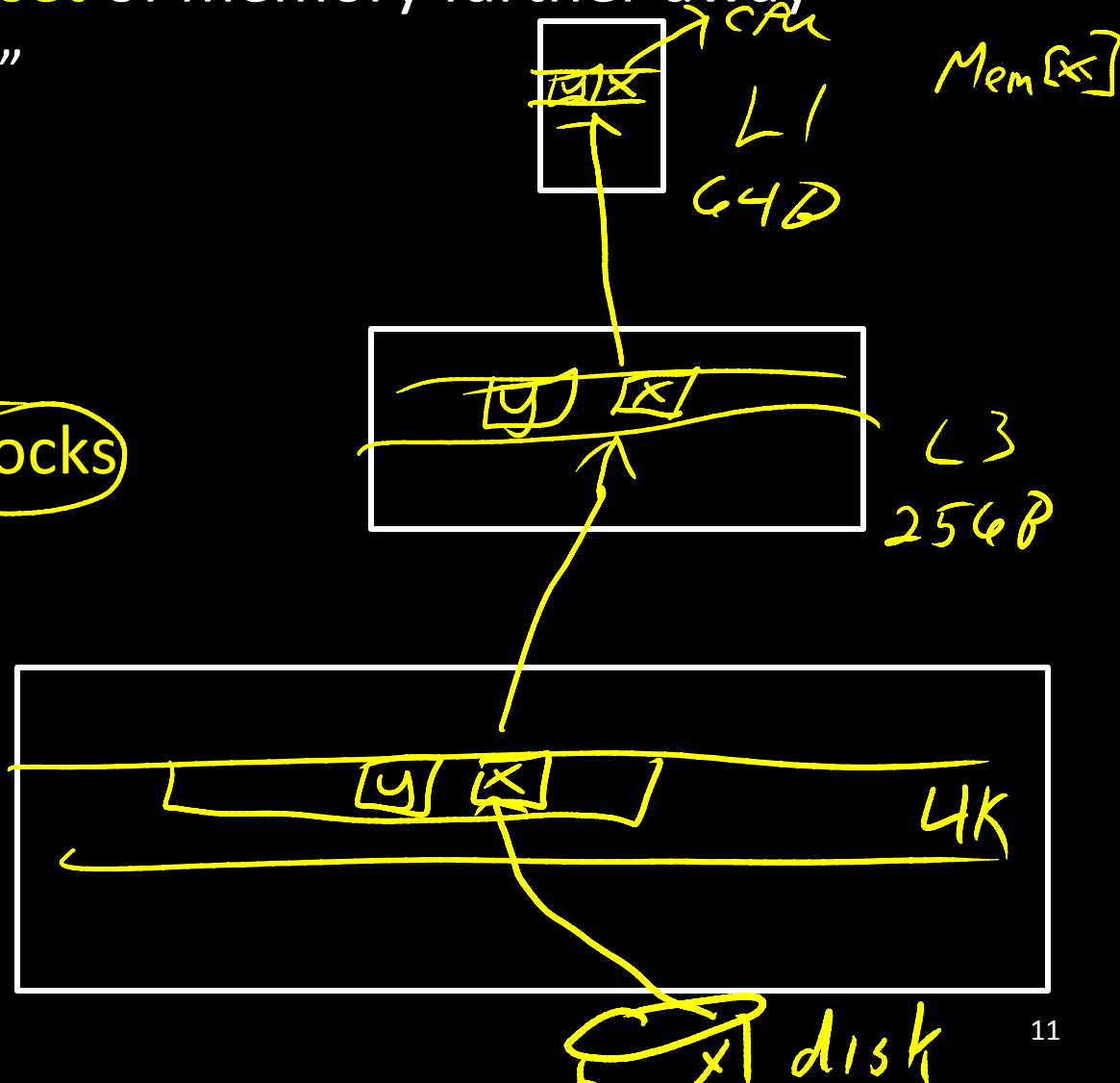 - "strictly inclusive"

- alternatives:
  - strictly exclusive
  - mostly inclusive

- Transfer whole blocks
  (cache lines):

  4kb: disk ⟷ ram

  256b: ram ⟷ L2

  64b: L2 ⟷ DRAM

Cache Lookups (Read)

Processor tries to access Mem[x]

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

- Yes: cache hit
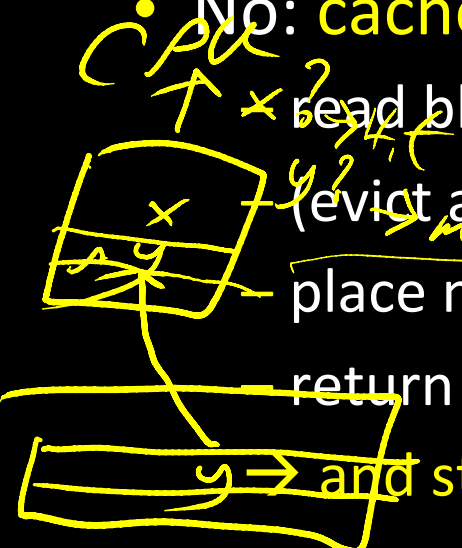 - return requested data from cache line

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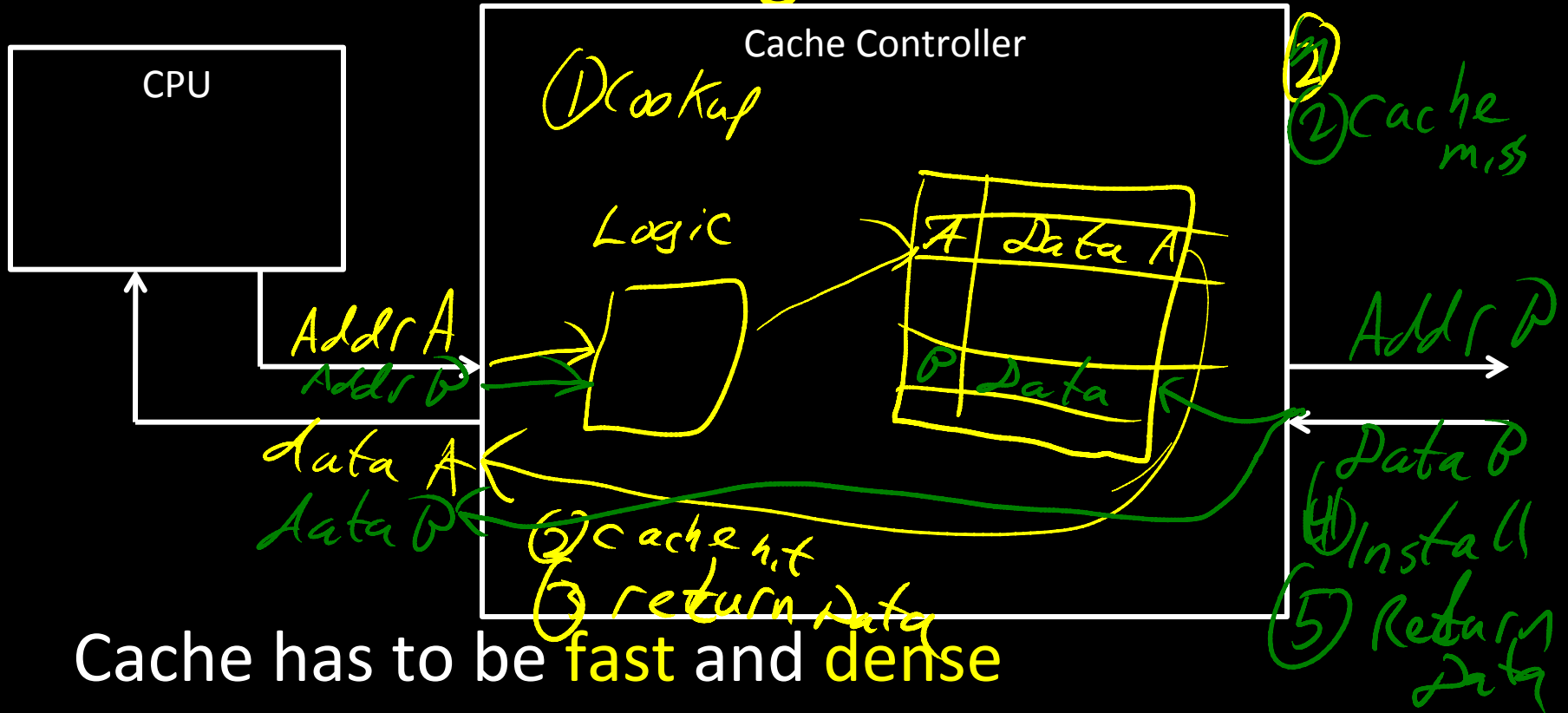 - (evict an existing cache line to make room)
  - place new block in cache
  - return requested data
  - and stall the pipeline while all of this happens

Cache Organization

CPU

Cache Controller

① Lookup

Logic

① A Data A

② Cache miss

② Cache hit
③ return Data

Addr A
Addr B
data A
Data B

Addr B

Data B
④ Install
⑤ Return Data

Cache has to be fast and dense

- Gain speed by performing lookups in parallel
  - but requires die real estate for lookup logic *mor. more $$*

- Reduce lookup logic by limiting where in the cache a block might be placed *better port*
  - 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

Lot flexibility
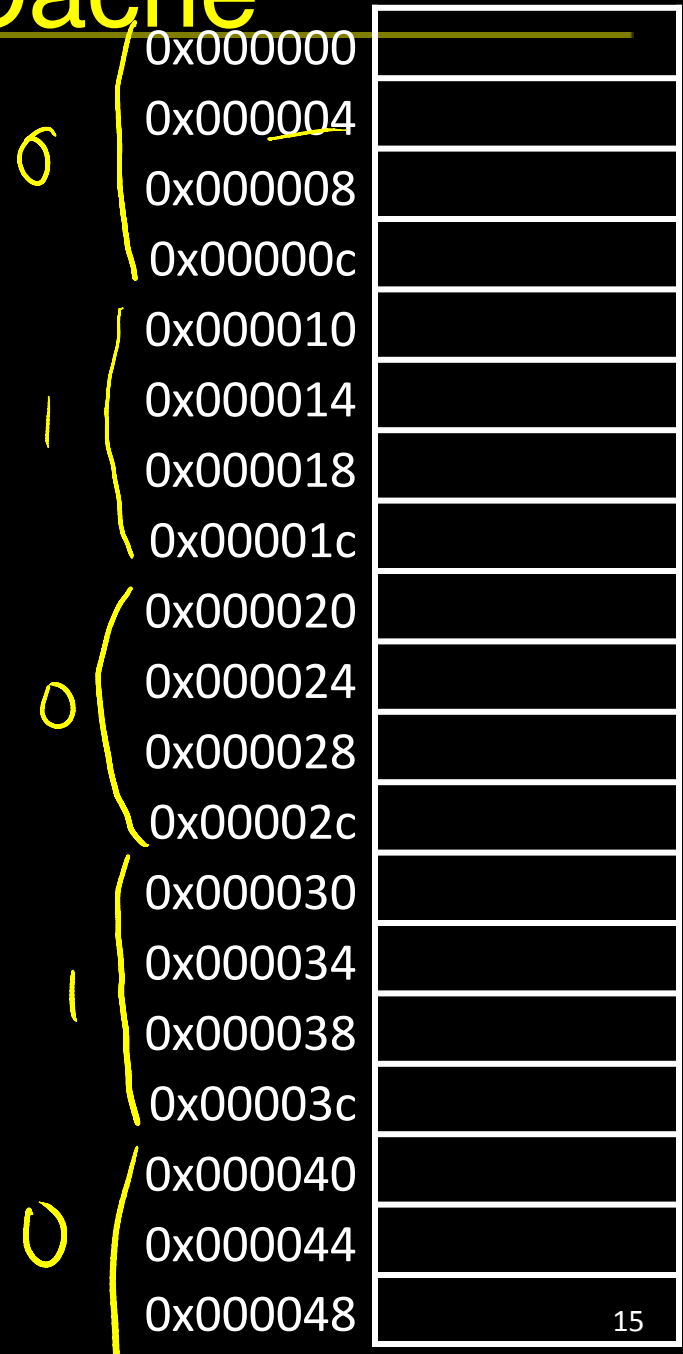
no flexibilit

little flex

Best Hit ratio

easiest
cheapest (*real estate*)
imp

Direct Mapped Cache

Direct Mapped Cache

- Each block number mapped to a single cache line index

- Simplest hardware

2 cache lines

line 0	0x0... 0	0x0... 4	0x0... 8	0x0... c
line 1				

4 words

0x000000
0x000004
0x000008   0
0x00000c

0x000010
0x000014   1
0x000018
0x00001c

0x000020
0x000024   0
0x000028
0x00002c

0x000030
0x000034   1
0x000038
0x00003c

0x000040
0x000044   0
0x000048

15

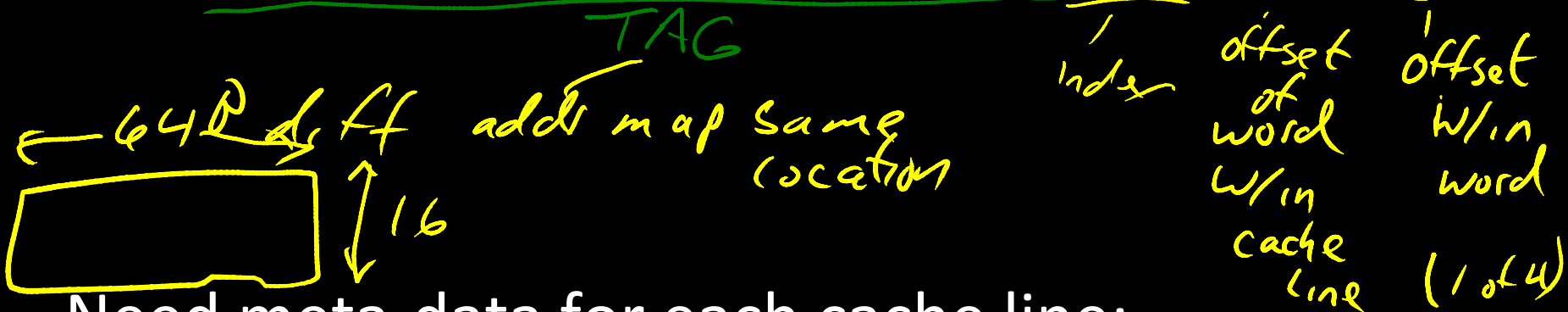## Direct Mapped Cache

- Each block number mapped to a single cache line index

- Simplest hardware

4 cache lines

line 0	X a	Y b
line 1		
line 2		
line 3		

2 words

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

16

Tags and Offsets

Assume sixteen 64-byte cache lines

0x7FFF3D4D

= 0111 1111 1111 1111 0011 1101 0100 1101

index 6-bit offset

$2^4 = 64$

TAG

index offset of word w/in cache line

offset w/in word (1 of 4)

64 Bdiff

16

add r map same location

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

Processor

lb $1 ← M[1]
lb $2 ← M[13]
lb $3 ← M[0]
lb $3 ← M[6]
lb $2 ← M[5]
lb $2 ← M[6]
lb $2 ← M[10]
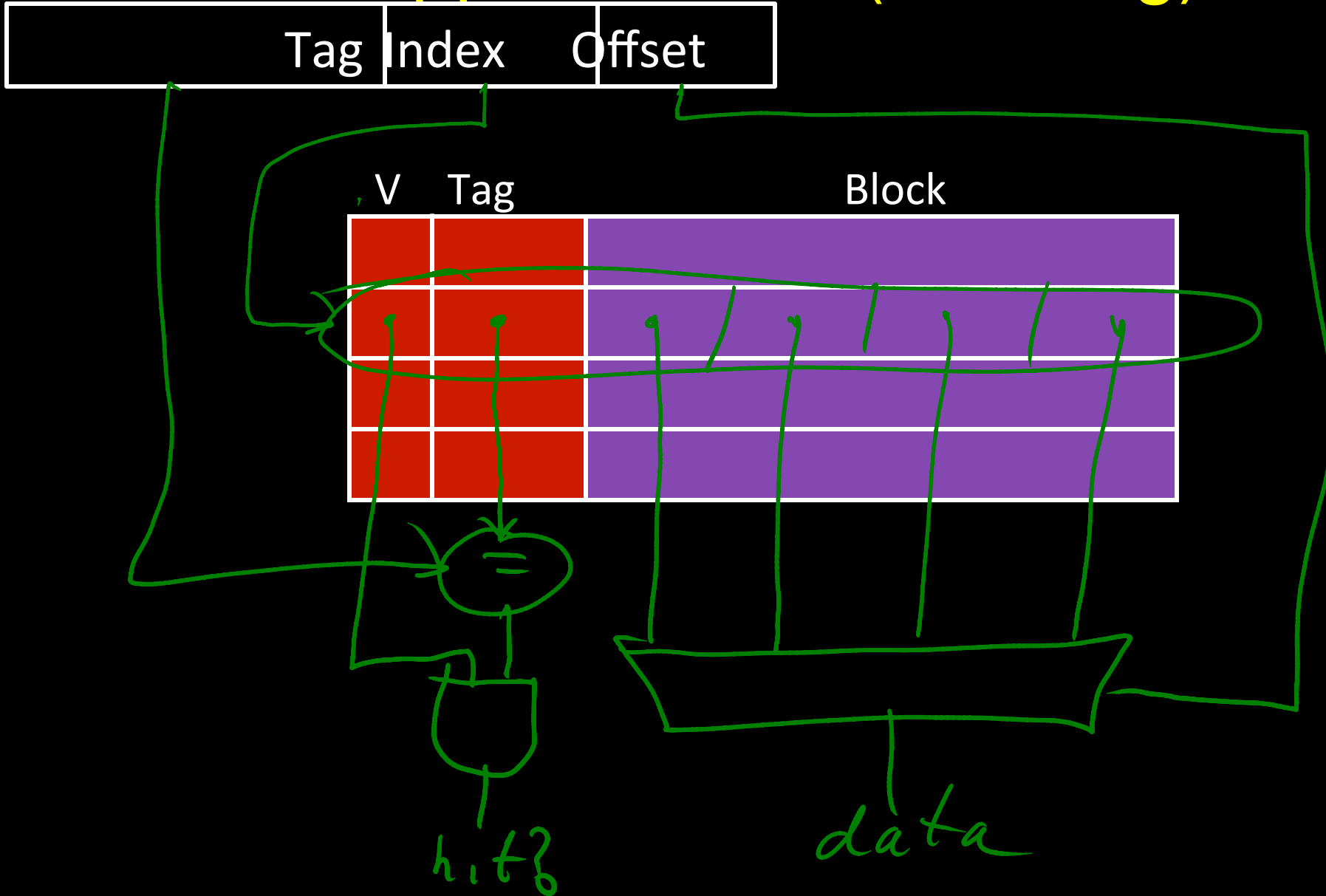lb $2 ← M[12]

$1
$2
$3
$4

Direct Mapped Cache

00110 = 6

01101 = 13

A =

A = 10000 11

00101 = 5

tag   index   off   index

V   tag

1	00	101	103
1	01	113 127	
1	00	131	137

Hits: | Misses: ||||

Memory

0	101
1	103
2	107
3	109
4	113
5	127
6	131
7	137
8	139
9	149
10	151
11	157
12	163
13	167
14	173
15	179
16	181

Direct Mapped Cache (Reading)

Tag Index Offset

V Tag Block

hit?

data

Direct Mapped Cache Size

	Tag	Index	Offset	

$$32-n-m \qquad n \qquad m$$

n bit index, *m* bit offset

Q: How big is cache (data only)?

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

$$2^n \text{ Lines} \cdot 2^m \text{ bytes per line} = 2^{n+m} \text{ bytes}$$

$$((32-n-m)+1) \cdot 2^n = \text{ overhead}$$

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

5 cycles per hit

90/10 rule
90% hit

$2 + 50 + 16 * 3 = 100$ cycles for a miss

$.9 * 5 + .1 * 100 =$

$5 + 50 + 15 * 3 = 100$

$4.5 \qquad 10 = 14.5$

Performance depends on:

Access time for hit, miss penalty, hit rate

Misses

Cache misses: classification

The line is being referenced for the first time

- Cold (aka Compulsory) Miss → *Prefetch*

The line was in the cache, but has been *Predicted* evicted

Avoiding Misses

Q: How to avoid…

Cold Misses

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

Other Misses
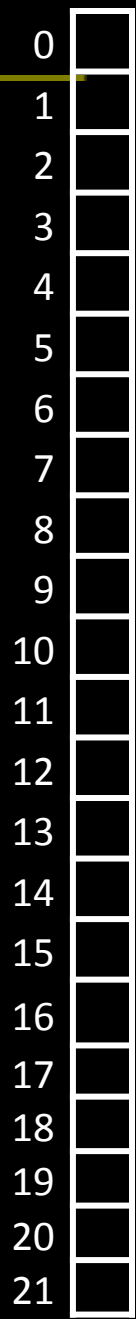
- Buy more SRAM
- Use a more flexible cache design

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

Processor

lb $1 ← M[1]
lb $2 ← M[13]
lb $3 ← M[0]
lb $3 ← M[6]
lb $2 ← M[5]
lb $2 ← M[6]
lb $2 ← M[10]
lb $2 ← M[12]

$1
$2
$3
$4

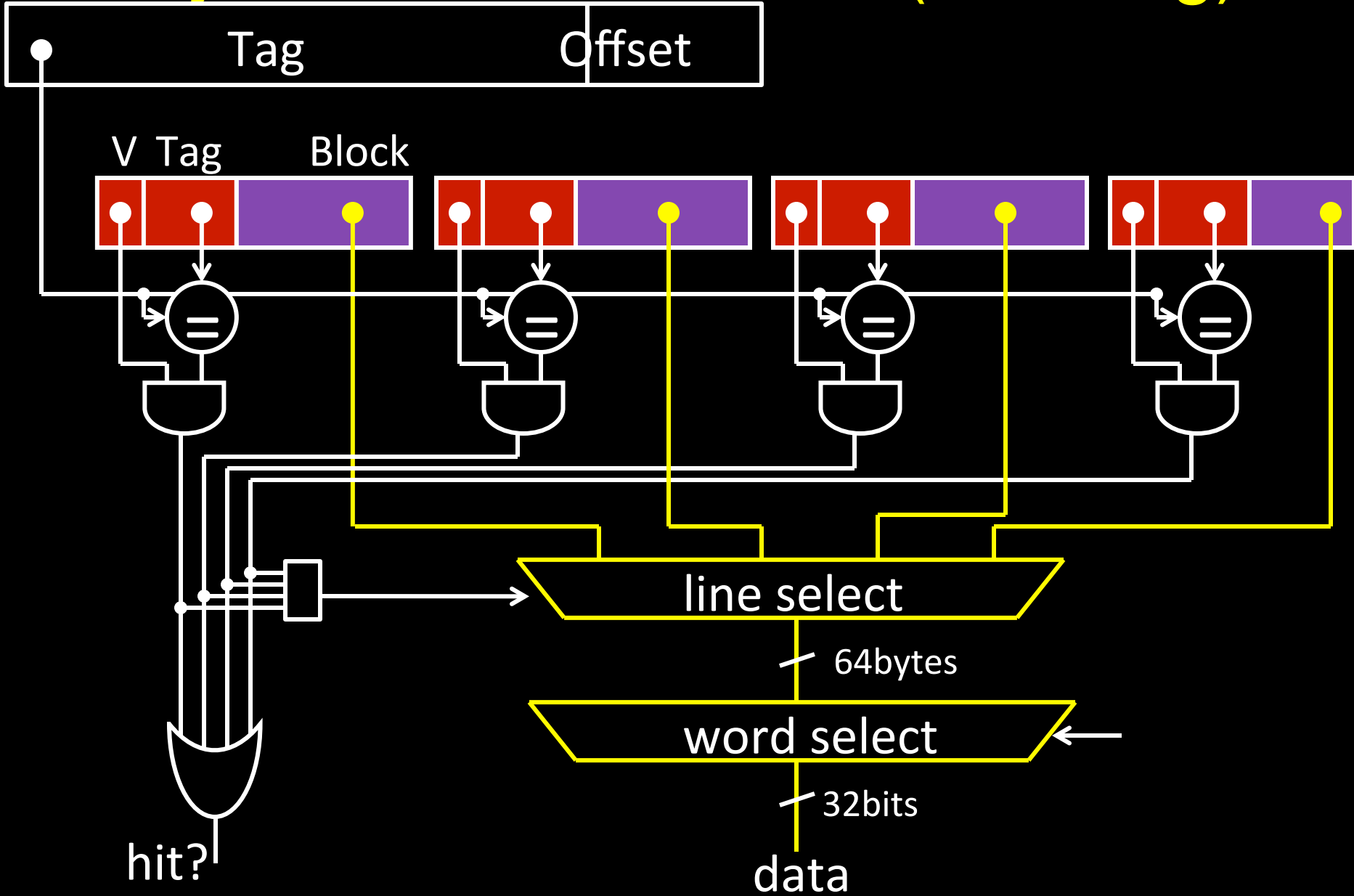## Fully Associative Cache

A =

V   tag

Hits: Misses:

Memory

0	101
1	103
2	107
3	109
4	113
5	127
6	131
7	137
8	139
9	149
10	151
11	157
12	163
13	167
14	173
15	179
16	181

Fully Associative Cache (Reading)

Tag | Offset

V Tag | Block

line select

64bytes

word select

32bits

hit?

data

Fully Associative Cache Size

Tag	Offset

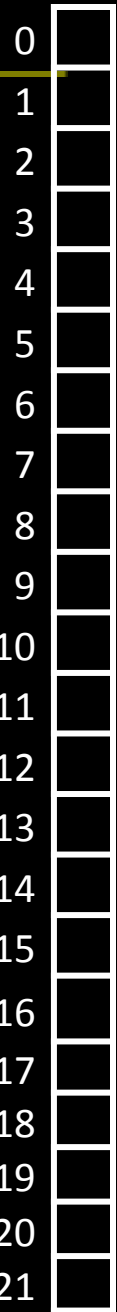m bit offset , 2^n cache lines

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?

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

... 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