Lecture 23: Caching out

CMPE 120: Computer Organization & Architecture

Olivia Weng

Logistics

- Reminder: check-in with professor is 1% of your final grade
  - Sign up link
- Grade remediation plan
  - 30+ students have already signed up
  - Slots are getting booked quickly → sign up soon!
- HW5 released last Thursday. Start now!

Grade remediation plan

- HW1 Quiz Remediation: meet with professor for 15 minutes
  - Bring a newly hand-written solution. Explain what you got wrong on your quiz and your new solution.
  - Replaces HW1 quiz grade & drops lowest HW Quiz grade
  - Schedule a time on my calendar in the next ~3 weeks (10/30 - 11/20)
    - Link: https://calendar.app.google/d3D4hm3JbmpXcrVt5
    - 100 appointment slots available for 67 students
    - If none of these times work, email me to schedule a time during this window.

- Final exam grade can replace the midterm grade if higher
  - → Midterm grade replacement = max(final exam grade, midterm exam grade)

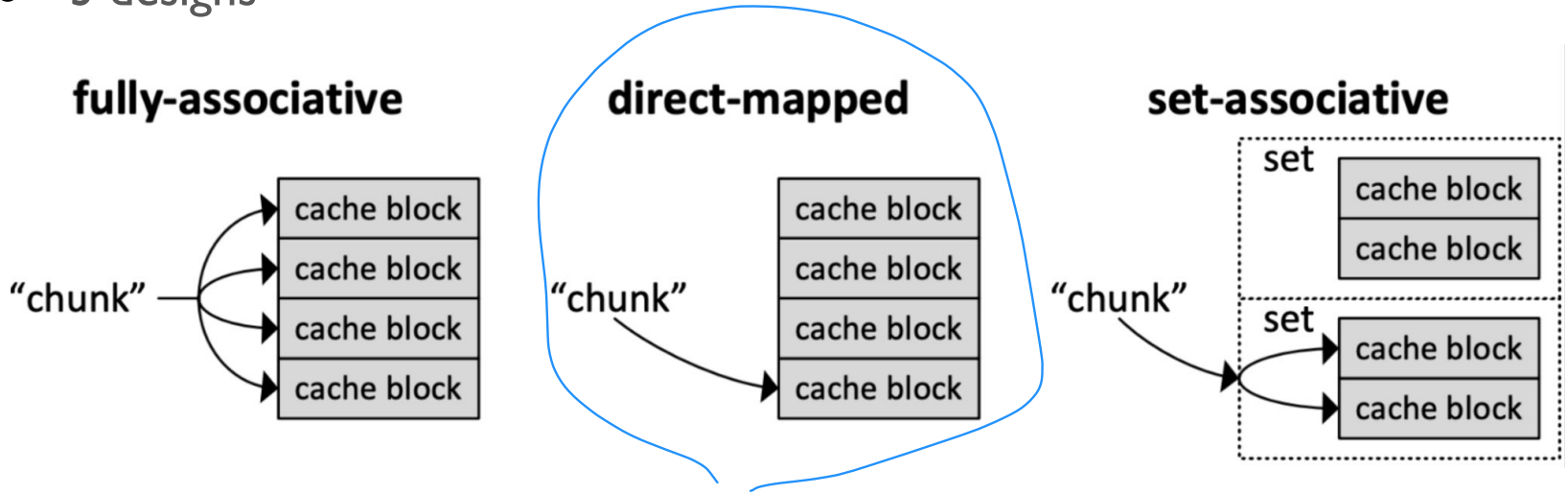
Cache design

- How to map data chunks from main memory to blocks in the cache?

Cache design

- How to map data chunks from main memory to blocks in the cache?

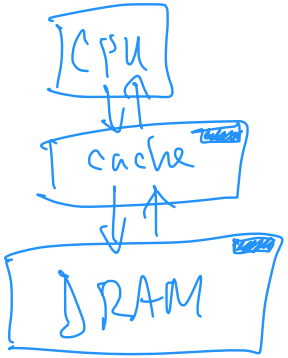
- 3 designs



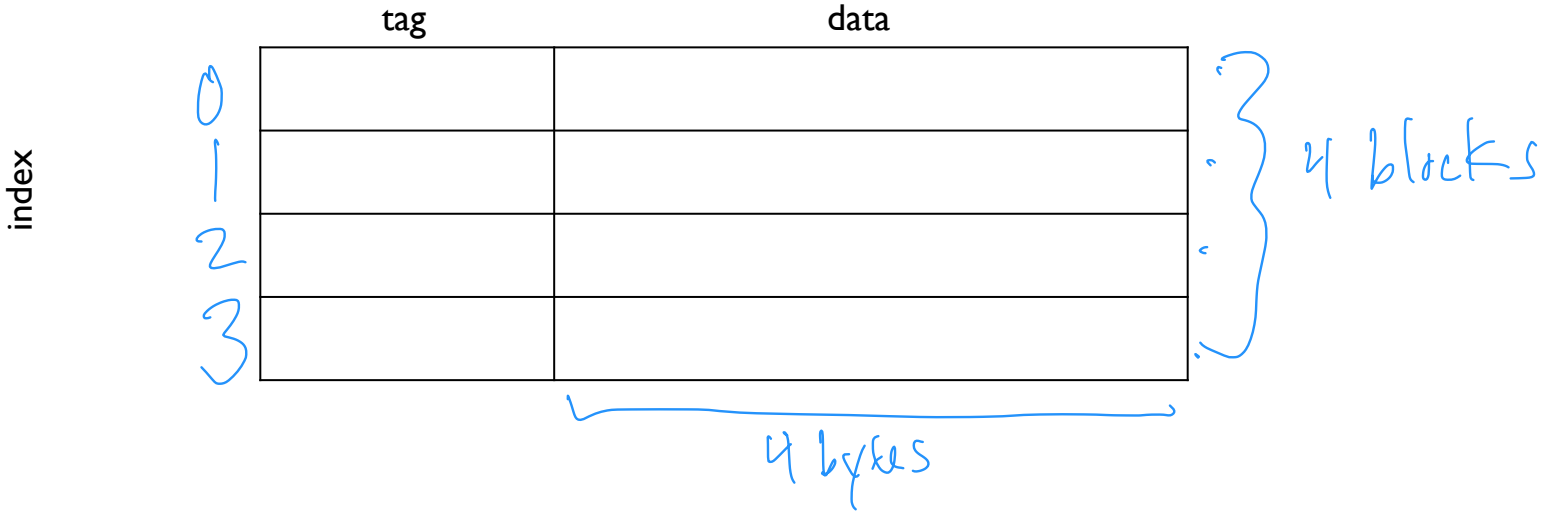
Direct mapped Cache

- A cache that can put a block of data in exactly 1 place

Direct mapped Cache



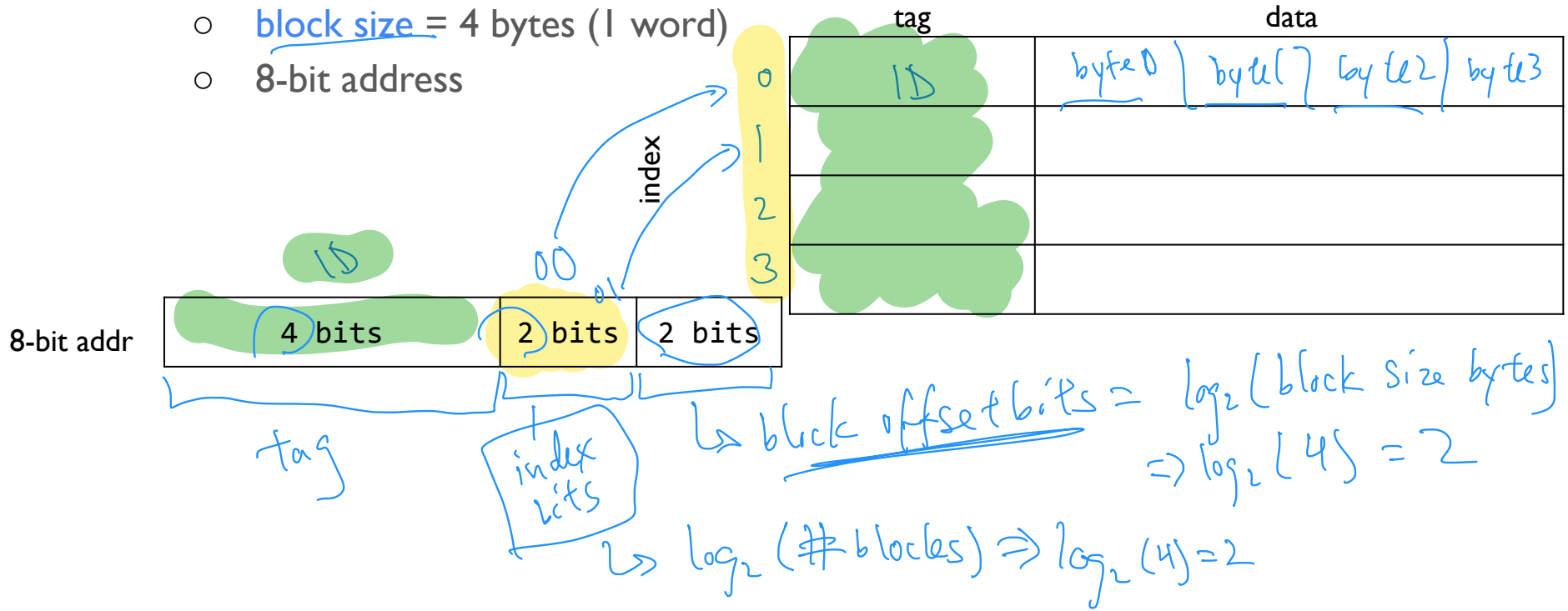
- A cache that can put a block of data in exactly 1 place
- Example: 4-block direct mapped cache
  - block size = 4 bytes (1 word)
  - each word in memory maps to exactly 1 cache location



Direct mapped Cache

data byte-addressable

- Example: 4-block direct mapped cache
  - block size = 4 bytes (1 word)
  - 8-bit address



Extracting addresses from code

- A and B are int (4-byte) arrays with 2 elements.
  - Assume `i`, `j`, and `sum` are all in registers
  - A's base address: 0x10
  - B's base address: 0x38

```
int sum = 0;
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        sum += A[j] - B[j];
    }
}
```

What is the stream of addresses?

A. A[0], B[0], A[1], B[0], A[0], B[1], A[1], B[1]

B. A[0], B[0], A[0], B[0], A[1], B[1], A[1], B[1]

C. A[0], B[0], A[1], B[1], A[0], B[0], A[1], B[1]

D. A[0], B[0], A[1], B[1], A[0], B[1], A[0], B[1]

Miss types

- Compulsory miss: first access of data

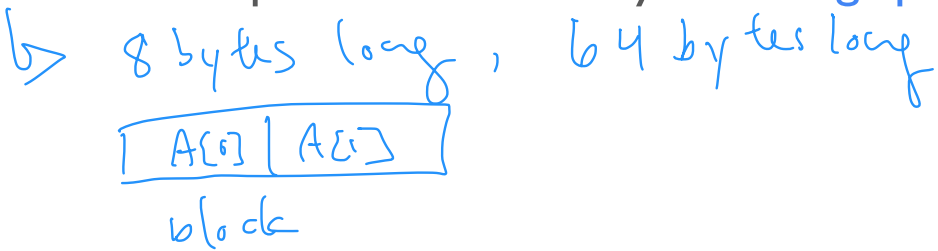
Miss types

- Compulsory miss: first access of data

- Capacity miss: missed only because cache is not big enough
  - If only have room for 2 items but trying to store 4, then you're out of space

Larger block sizes

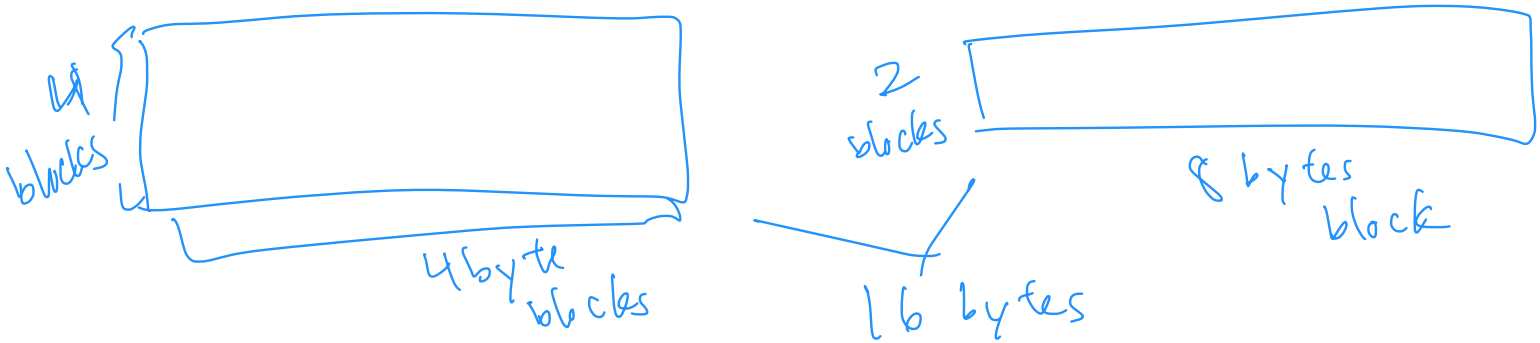
- Larger blocks help reduce misses by lowering spatial locality misses



Larger block sizes

- Larger blocks help reduce misses by lowering spatial locality misses

- The larger the block size, the fewer blocks can be stored in same sized cache
  - Tradeoff!
  - Most modern machines have block size of 64 bytes



Key caching formulas

- With direct mapped caches
  - `num_block_offset_bits = log2(block_size_in_bytes)`
  - `num_cache_entries = cache_size / block_size`
  - `num_index_bits = log2(num_cache_entries)`
  - `num_tag_bits = address_size - num_index_bits - num_block_offset_bits`

New base address for B

- A and B are int (4-byte) arrays with 2 elements.
  - Assume `i`, `j`, and `sum` are all in registers
  - A's base address: 0x10
  - B's base address: 0x30

```
int sum = 0;
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        sum += A[j] - B[j];
    }
}
```

New base address for B

Item	Addr b10	Addr b2	H/M?
A[0]	16	0001 0 000	M
B[0]	48	0011 0 000	M
A[1]	20	0001 0 100	
B[1]	52	0011 0 100	
A[0]	16	0001 0 000	
B[0]	48	0011 0 000	
A[1]	20	0001 0 100	
B[1]	52	0011 0 100	

Each block now holds 2 words (8 bytes)

tag	data (8 bytes)
0011	B[0] B[1]

What is the hit rate on this cache?

A. 75%
B. 50%
C. 25%
D. 12.5%
E. 0%

Why do we miss on A[1]?

Why do we miss on A[1]?

- Conflict miss: Had capacity to fit all blocks, but indexing caused miss
  - Although had room to hold both A[0-1] & B[0-1], all elements indexed to 0
  - Analogy: collisions in hash table (index is hash function)

Fully associative cache

- Fully associative cache: block can be placed in any cache location
  - No rules on what data can go where

Fully associative cache

- Fully associative cache: block can be placed in any cache location
  - No rules on what data can go where

- But, some new considerations:
  - Where do you look to find an element?

  - How to decide what to remove?

to replacement policy

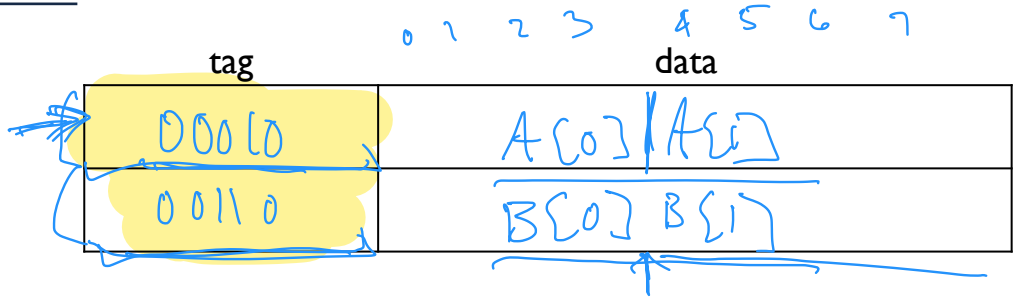
replacement policy : Least Recently Used

Fully associative cache

Item	Addr b10	Addr b2	H/M?
A[0]	16	00010 000	M
B[0]	48	00110 000	M
A[1]	20	00010 100	H
B[1]	52	00110 100	H
A[0]	16	00010 000	H
B[0]	48	00110 000	H
A[1]	20	00010 100	H
B[1]	52	00110 100	H

tag | block offset bits

Each block holds 2 words (8 bytes)



6 hits / 8 accesses = 75%

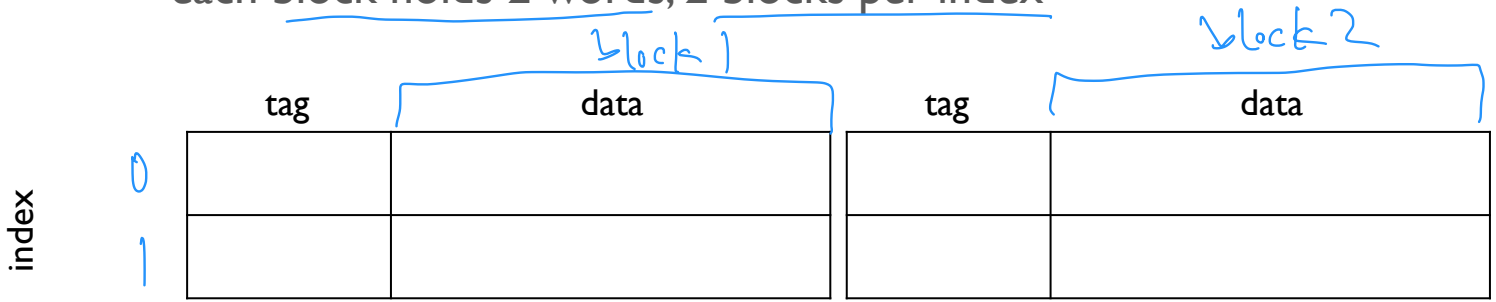
What is the hit rate on this cache?

A. 75%
B. 50%
C. 25%
D. 12.5%
E. 0%

Set associativity

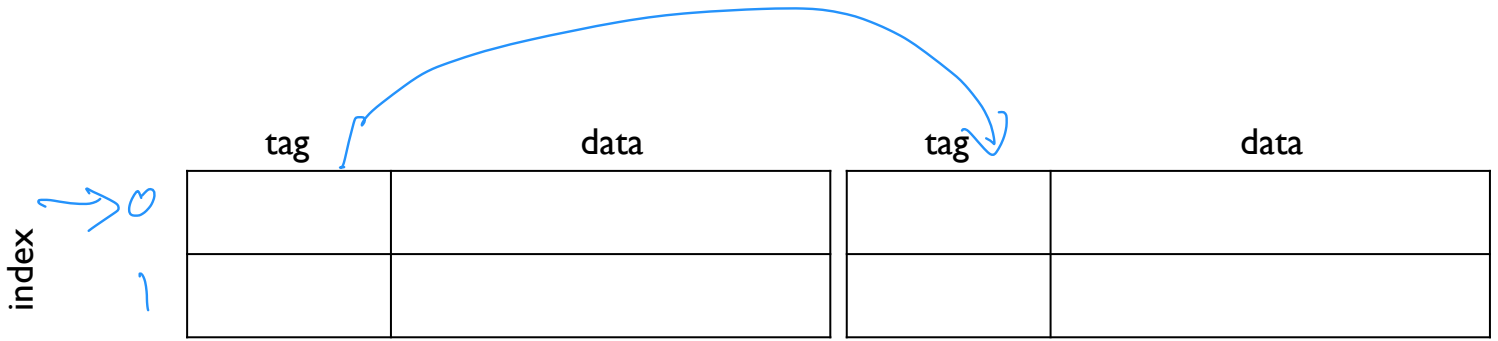
- n-way set-associative cache
  - Each index can store n blocks
  - All blocks sharing an index are in a set

- Example: 4 blocks
  - each block holds 2 words, 2 blocks per index



Set associativity

- Consider again:
  - Where do you look to find an element?
  - How to decide what to remove? LRU



2-way set-associative cache

Each block holds 2 words (8 bytes)

Item	Addr b10	Addr b2	H/M?
A[0]	16	0001 0 000	M
B[0]	48	0011 0 000	M
A[1]	20	0001 0 100	H
B[1]	52	0011 0 100	H
A[0]	16	0001 0 000	H
B[0]	48	0011 0 000	H
A[1]	20	0001 0 100	H
B[1]	52	0011 0 100	H

tag | idx | offset

6 hits / 8 accesses

75%

idx	tag	data	tag	data
0	0001	A[0] A[1]	0011	B[0] B[1]
1				

What is the hit rate on this cache?

A. 75%
B. 50%
C. 25%
D. 12.5%
E. 0%

Set associative caches

- Performance is middle-ground between directed mapped & fully associative
  - Decent hit times
  - Decent use of cache space

- Hybrid design between direct mapped & fully associative

- Hit rates generally higher than direct mapped

↳ reduced conflict misses