

# Memory & Caches III

CSE 351 Winter 2024

## Instructor:

Justin Hsia

## Teaching Assistants:

Adithi Raghavan

Aman Mohammed

Connie Chen

Eyoel Gebre

Jiawei Huang

Malak Zaki

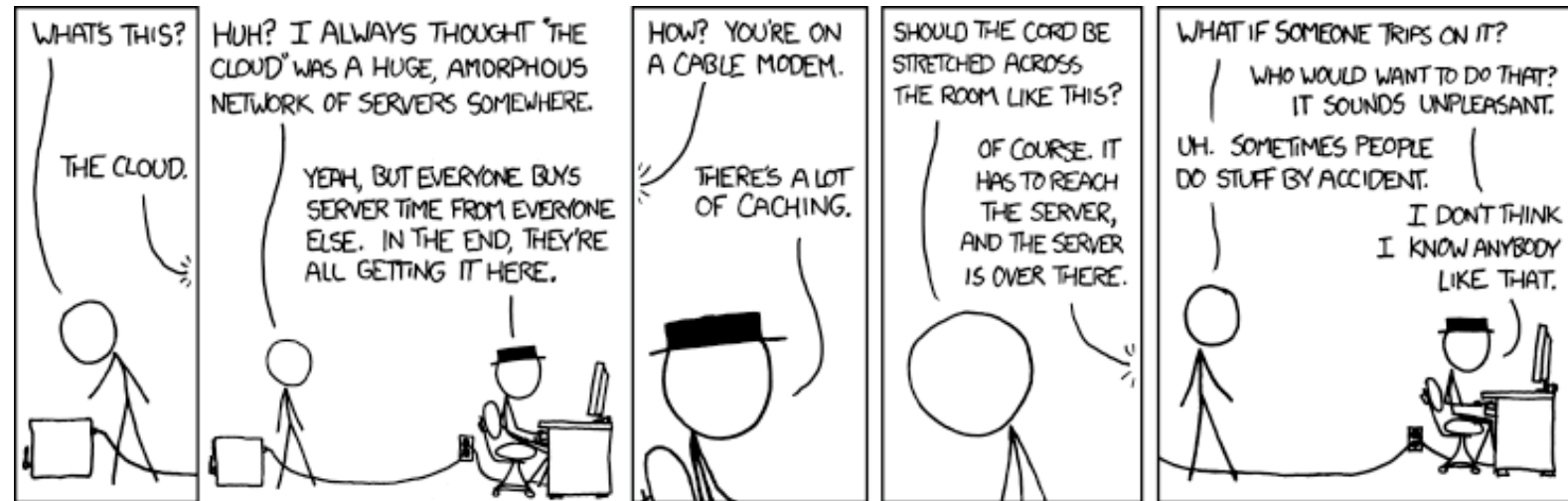
Naama Amiel

Nathan Khuat

Nikolas McNamee

Pedro Amarante

Will Robertson



<http://xkcd.com/908/>

# Relevant Course Information

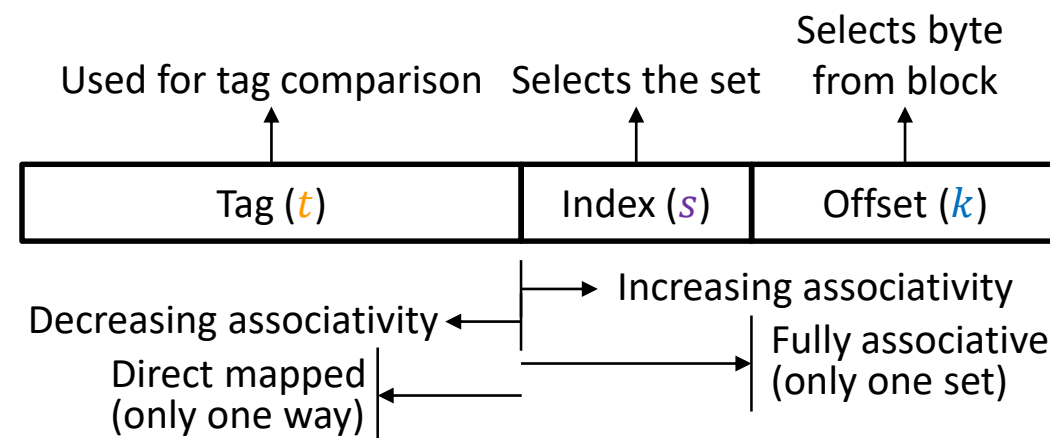
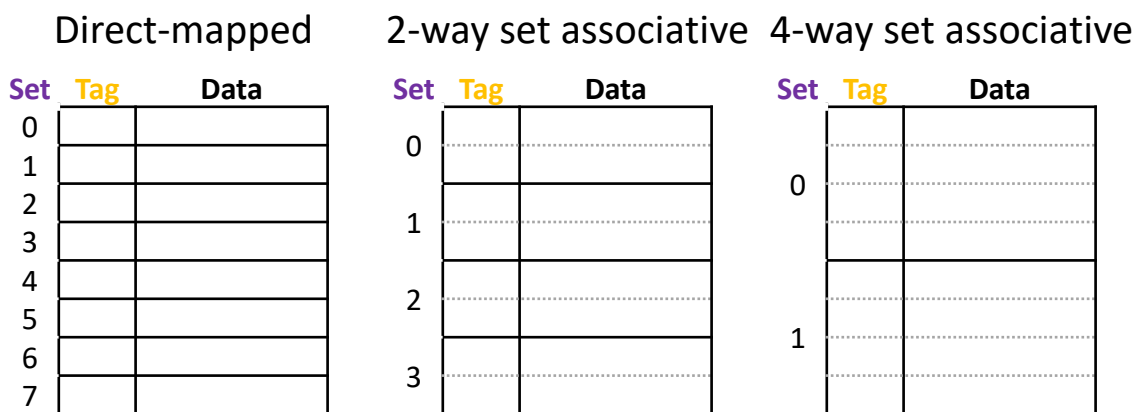
- ❖ HW15 due tonight, HW16 due Friday, HW17 due Wednesday (2/21)
- ❖ Lab 3 due Friday (2/16)
- ❖ Lab 4 released Friday, due two weeks later (3/1)
  - Can do Part 1 after today; will need Lesson 18 to do Part 2
- ❖ No lecture on Monday for President's Day!

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions. The colors include shades of purple, blue, yellow, green, and red, representing different functional blocks and interconnects.

# Caches III

# Lesson Summary (1/2)

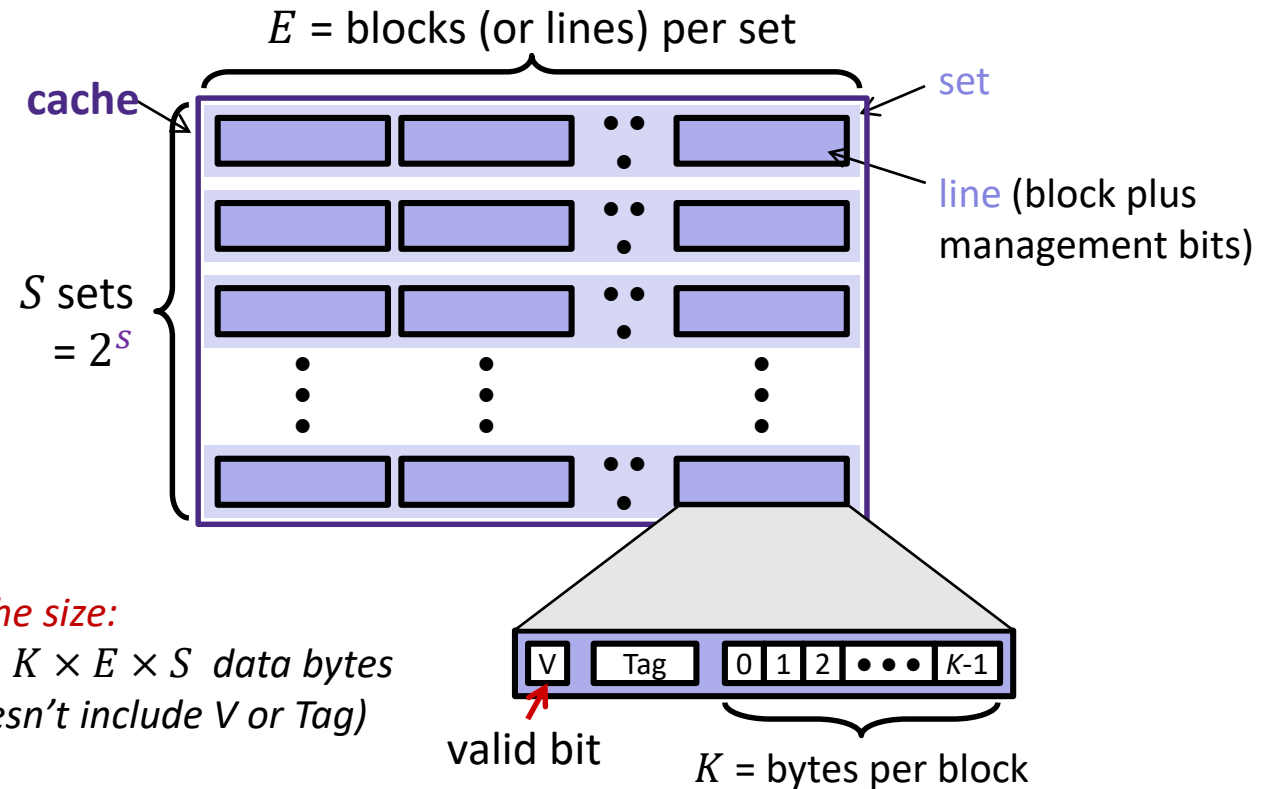
- ❖ **Associativity** gives us flexibility in where to place blocks in the cache
  - Group  $E$  slots into **sets**, means there are  $E$  **ways** to place block within each set
    - Direct-mapped is  $E = 1$ , **fully associative** is  $E = \#$  of slots in cache (*i.e.*,  $S = 1$ )
    - Helps avoid conflicts in each set at the expense of slightly longer and more complex searching & placing in the cache
    - By default, we will replace the **least-recently used** block in a set
  - Index  $\rightarrow$  Set,  $S = (C/K)/E$ , still  $s = \log_2(S)$



# Lesson Summary (2/2)

## ❖ Management bits

- Information needed for proper management of the cache & its data, but not counted in the cache size
- **Valid bit** for validity of data
- **Tag bits** for identifying which block



# Lesson Q&A

- ❖ Learning Objectives:
  - Determine how memory addresses and data interact with the cache (*i.e.*, cache lookups, data movement).
  - Analyze how changes to cache parameters and policies affect performance metrics such as AMAT.
  
- ❖ What lingering questions do you have from the lesson?
  - Chat with your neighbors about the lesson for a few minutes to come up with questions

A detailed, colorful micrograph of a microchip die, showing a complex grid of circuitry and various colored regions (purple, blue, yellow, green, red) representing different functional blocks and interconnects.

# Caches III – Practice

# Polling Questions

- ❖ We have a cache of size 2 KiB with block size of 128 B. If our cache has 2 sets, what is its associativity?
  - A. 2
  - B. 4
  - C. 8
  - D. 16
  
- ❖ If addresses are 16 bits wide, how wide is the Tag field?



# Homework Setup

- ❖ Addresses are 13 bits wide, and the cache is two-way set-associative ( $E=2$ ) with 4-byte block size ( $K=4$ ) and eight sets ( $S=8$ ).

2-way set-associative cache

Set Index	Line 0						Line 1					
	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	21	1	68	65	79	20	00	0	F3	29	84	55
1	43	1	74	68	65	72	21	1	63	F7	E3	D1
2	7F	1	65	20	69	74	0B	1	22	AA	BB	CC
3	29	0	27	73	20	73	3B	0	2F	5C	93	4B
4	3D	1	61	6D	6D	79	00	0	–	–	–	–
5	2C	0	20	77	6F	6C	00	0	–	–	–	–
6	1A	1	66	64	61	77	2F	1	23	24	25	F3
7	2B	1	67	0	0	0	C1	0	49	AB	DF	CC

- What are the widths of the tag, index, and offset fields?

# Homework Setup

- Addresses are 13 bits wide, and the cache is two-way set-associative ( $E=2$ ) with 4-byte block size ( $K=4$ ) and eight sets ( $S=8$ ).

2-way set-associative cache

Set Index	Line 0						Line 1					
	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	21	1	68	65	79	20	00	0	F3	29	84	55
1	43	1	74	68	65	72	21	1	63	F7	E3	D1
2	7F	1	65	20	69	74	0B	1	22	AA	BB	CC
3	29	0	27	73	20	73	3B	0	2F	5C	93	4B
4	3D	1	61	6D	6D	79	00	0	–	–	–	–
5	2C	0	20	77	6F	6C	00	0	–	–	–	–
6	1A	1	66	64	61	77	2F	1	23	24	25	F3
7	2B	1	67	0	0	0	C1	0	49	AB	DF	CC

- What addresses will hit in Set 0?



# Caches III – Context

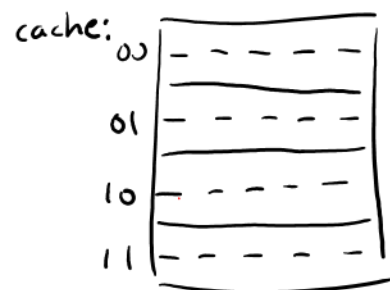
# Example Code Analysis Problem

❖ Assuming the cache starts cold (all blocks invalid) and `sum`, `i`, and `j` are stored in registers, calculate the **miss rate**: 100%

- $m = 10$  bits,  $C = 64$  B,  $K = 8$  B,  $E = 2$   $t = 5$  bits,  $s = 2$  bits,  $k = 3$  bits

2 bytes  
per element

```
#define SIZE 8
short ar[SIZE][SIZE], sum = 0; // &ar=0x200
for (int i = 0; i < SIZE; i++)
    for (int j = 0; j < SIZE; j++)
        sum += ar[j][i];
```



	address	tag	index	offset		
<code>ar[0][0]</code>	$0b\ 10\ 0000\ 0000$	10	0000	0000	$\rightarrow M$	(1 <sup>st</sup> way)
<code>ar[1][0]</code>	$0b\ 10\ 0001\ 0000$	10	0001	0000	$\rightarrow M$	(1 <sup>st</sup> way)
<code>ar[2][0]</code>	$0b\ 10\ 0010\ 0000$	10	0010	0000	$\rightarrow M$	(2 <sup>nd</sup> way)
<code>ar[3][0]</code>	$0b\ 10\ 0011\ 0000$	10	0011	0000	$\rightarrow M$	(2 <sup>nd</sup> way)
<code>ar[4][0]</code>	$0b\ 10\ 0100\ 0000$	10	0100	0000	$\rightarrow M$	(replacement!)
$\vdots$			$\vdots$			
<code>ar[0][1]</code>	$0b\ 10\ 0000\ 0010$	10	0000	0010	$\rightarrow M$	(conflict!)

matrix ar:



Cache block holds 4 elements of a row of the matrix

jumping by a row skips two block numbers (and sets)

# Cache Simulator

- ❖ <https://courses.cs.washington.edu/courses/cse351/cachesim/>
  - From course website: Simulators → Cache Simulator
  - Allows you to play around with the effects of cache parameters and policies
  - Lots of neat features like highlighting, hover text, ability to rewind and replay accesses, and copy-and-paste access patterns
- ❖ Ways to use:
  - Take advantage of “explain mode” and navigable history to test your own hypotheses and answer your own questions
  - Self-guided Cache Sim Demo posted along with Section 7
  - Will be used in HW18 – Lab 4 Preparation

# Cache Simulator Demo

- ❖ <https://courses.cs.washington.edu/courses/cse351/cachesim/>
  - From course website: Simulators → Cache Simulator
  - Allows you to play around with the effects of cache parameters and policies
  - Lots of neat features like highlighting, hover text, ability to rewind and replay accesses, and copy-and-paste access patterns
- ❖ Let's simulate the example problem from the lesson:

```
#define SIZE 8
short ar[SIZE][SIZE], sum = 0; // &ar=0x200
for (int i = 0; i < SIZE; i++)
    for (int j = 0; j < SIZE; j++)
        sum += ar[j][i];
```

# Group Work Time

- ❖ During this time, you are encouraged to work on the following:
  - 1) If desired, continue your discussion
  - 2) Work on the homework problems
  - 3) Work on the lab (if applicable)
  
- ❖ Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support