# Lecture 21: Nothing beats cold, hard cache

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
  - 20+ students have already signed up
  - Slots are getting booked quickly → sign up soon!

- HW4 due this Thursday. HW4 Quiz next Tuesday.

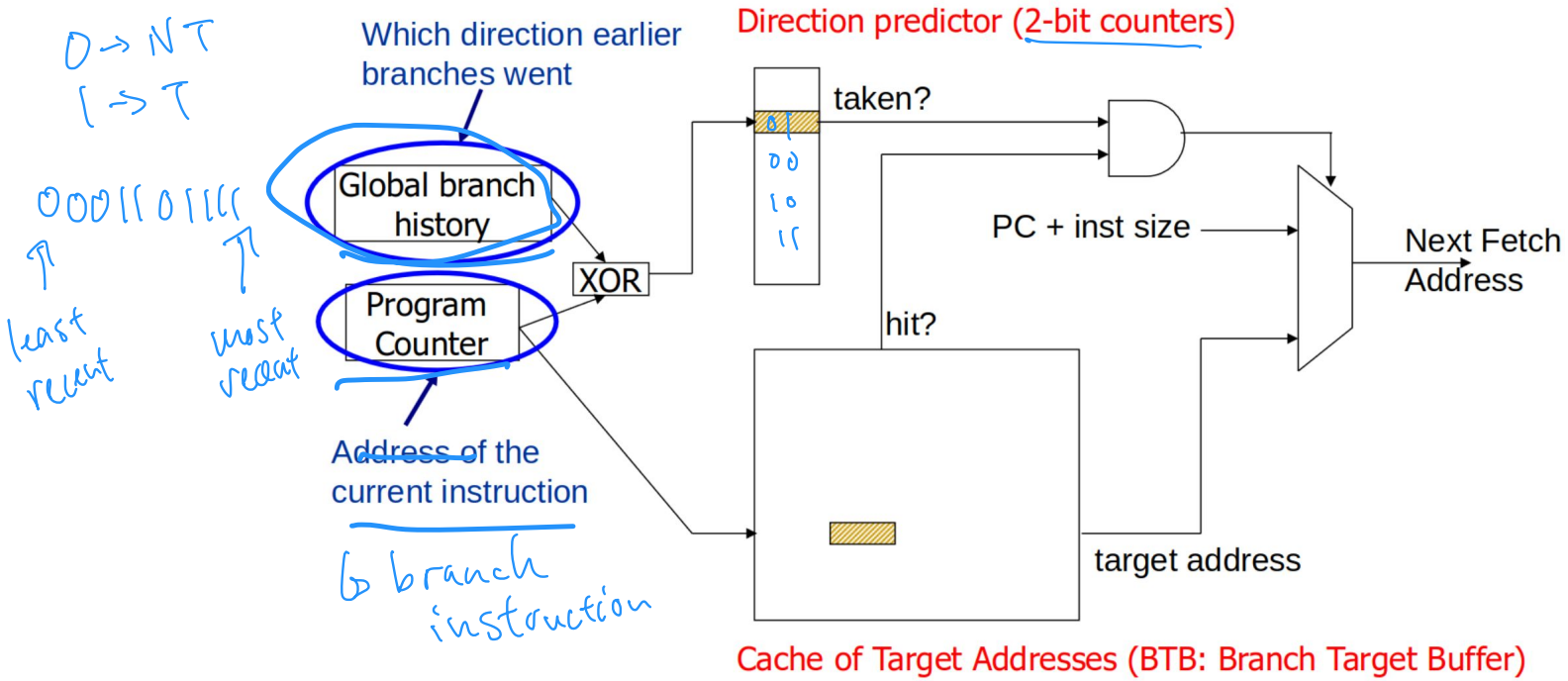
# Grade remediation plan

- 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)

# Two-level gshare branch predictor

- gshare: global shared pattern history table greatly reduces mispredictions




Paper: https://american.cs.ucdavis.edu/academic/readings/papers/mcfarling.pdf

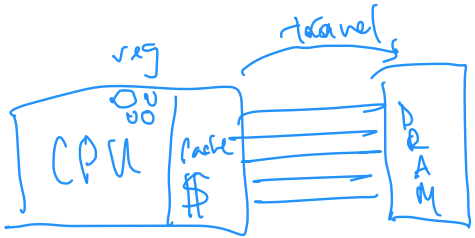
# What is a cache?

- A small memory (usually on-chip) where we can store data for easy access

# What is a cache?

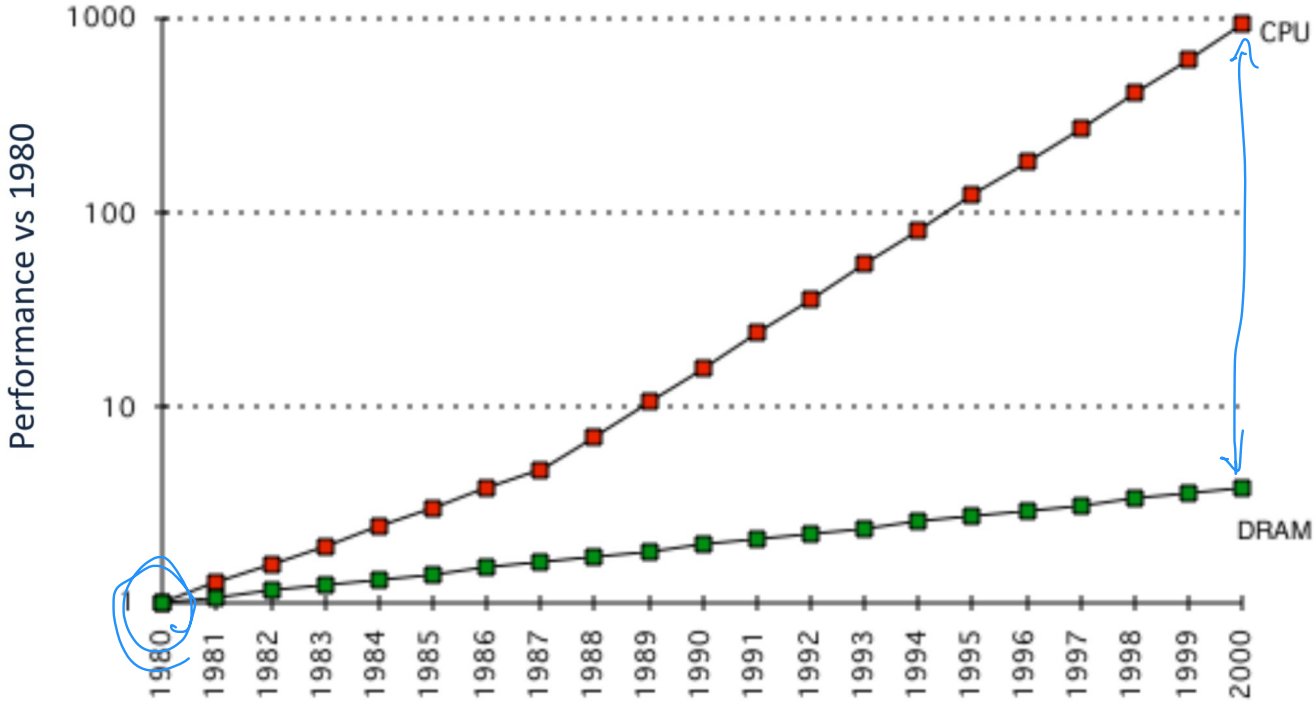
- A small memory (usually on-chip) where we can store data for easy access

- Why is a cache useful?
  - DRAM accesses are super slow
  - So far we've assumed DRAM access takes 1 cycle in our pipeline
    - In reality, it takes 350-500 cycles



# Processor vs memory performance

- Memory is super slow compared to processors



# What is a cache?

- A small memory (usually on-chip) where we can store data for easy access

- Why is a cache useful?
  - DRAM accesses are super slow
  - So far we've assumed DRAM access takes 1 cycle in our pipeline
    - In reality, it takes 350-500 cycles
  - Cache accesses can take <10s cycles

# What is a cache?

- A small memory (usually on-chip) where we can store data for easy access

- Why is a cache useful?
  - DRAM accesses are super slow
  - So far we've assumed DRAM access takes 1 cycle in our pipeline
    - In reality, it takes 350-500 cycles
  - Cache accesses can take <10s cycles

How do caches fit in a computer's memory system?

# Memory vs Storage

- Computer has many kinds of memory and storage

Memory

volatile = needs power to remember

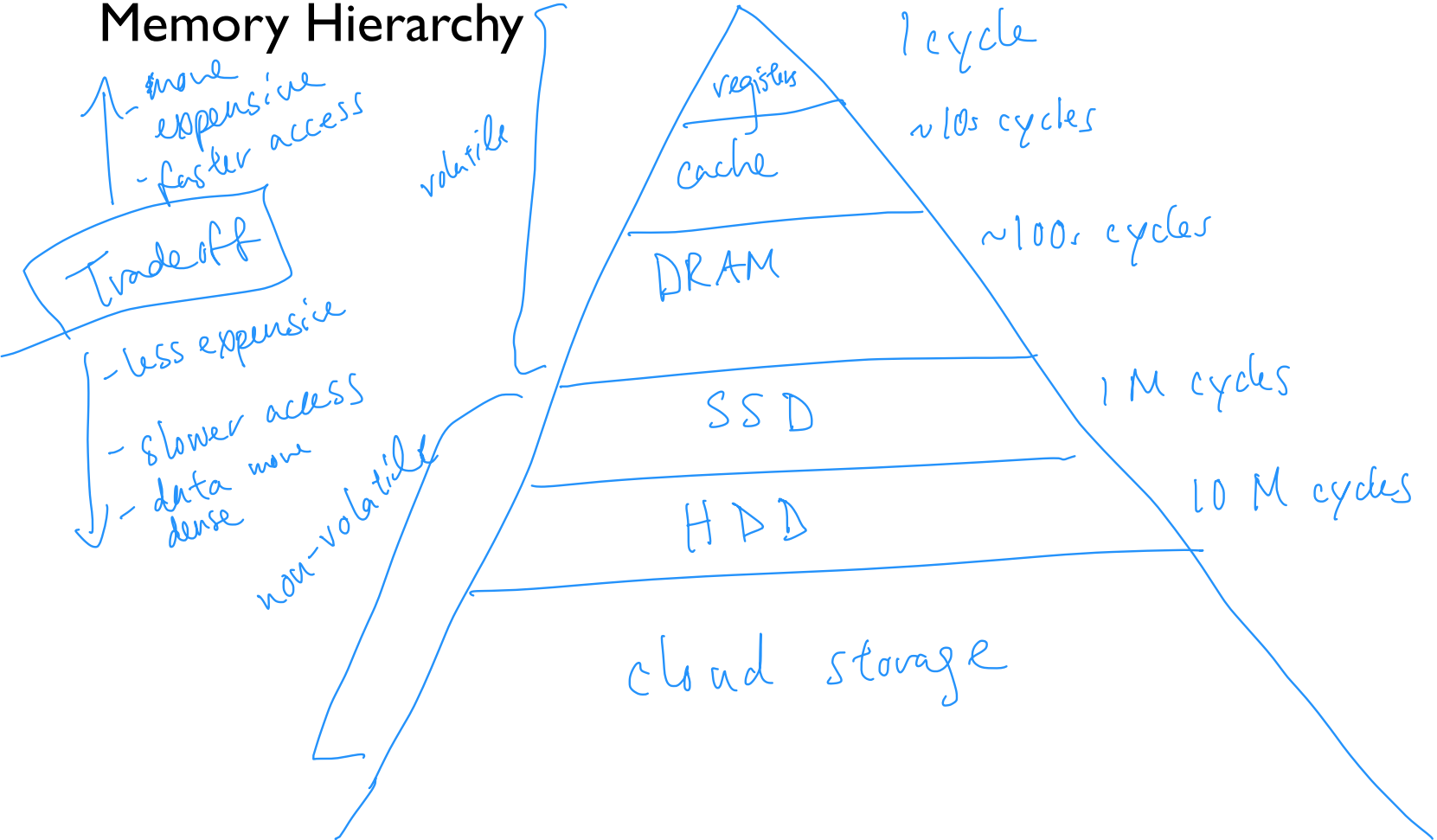
- regs.
- cache
- DRAM

Storage

non-volatile = does NOT need power to remember

- SSDs
- HDD
- storing files

# Memory Hierarchy



# Memory locality

- Memory hierarchies take advantage of memory locality

- Memory locality: future memory accesses are near past accesses

# Locality, locality, locality

- Temporal locality: near in time
  - We will often access the same data again soon

# Locality, locality, locality

- Temporal locality: near in time
  - We will often access the same data again soon

- Spatial locality: near in space/distance
  - Next access is often close to our last/recent accesses

# Locality, locality, locality

- Temporal locality: near in time
  - We will often access the same data again soon

- Spatial locality: near in space/distance
  - Next access is often close to our last/recent accesses

- Memories take advantage of both temporal and spatial locality

# Cache examples

- What are some examples?







# Ideal memory system

- We want a large and fast memory
  - But, we cannot have large and fast memory

# Ideal memory system

- We want a large and fast memory
  - But, we cannot have large and fast memory

- We have
  - small and fast memory — registers, caches — different sizes
  - large and slow memory — DRAM, SSD
  - speeds/sizes in between

- With memory locality, we can make this work

# Ideal memory system

- We want a large and fast memory
  - But, we cannot have large and fast memory

Without locality and caching, modern computing is impossible

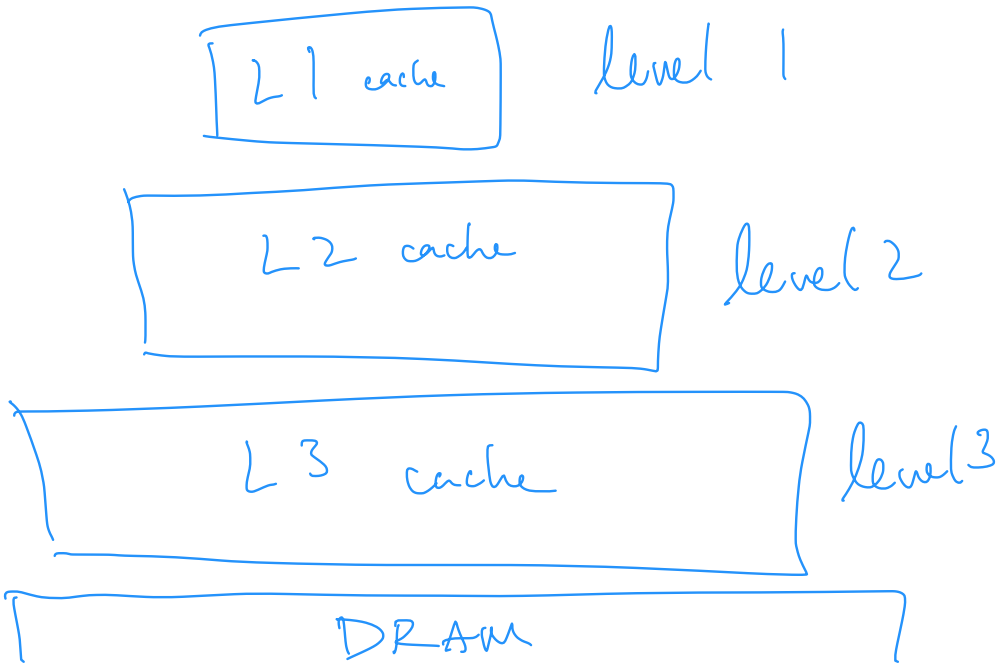
- We have
  - small and fast memory
  - large and slow memory
  - speeds/sizes in between

- With memory locality, we can make this work

# Cache levels

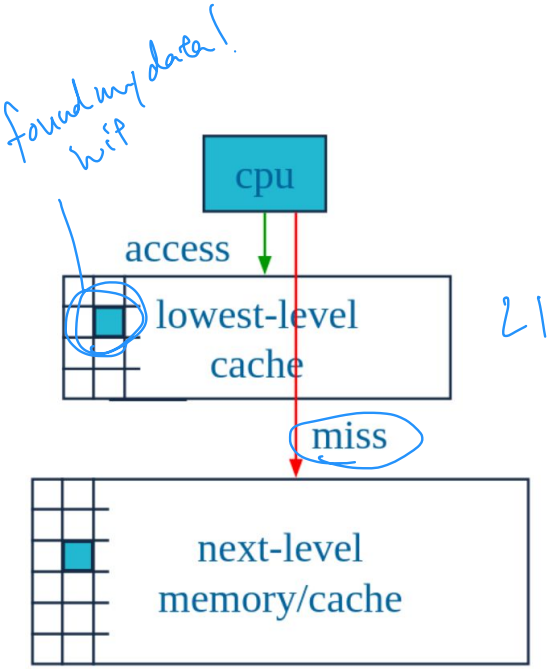
smaller caches have faster accesses

- Computers typically have 3 levels of cache
  - They vary in size and access times



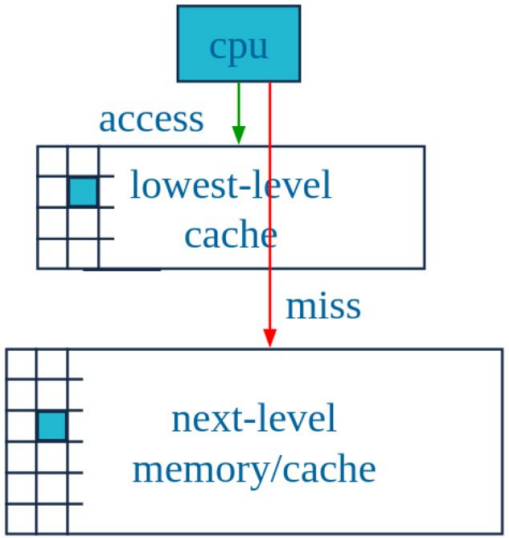
# Caching issues

- On memory access
  - How do I know if this is a hit or miss?

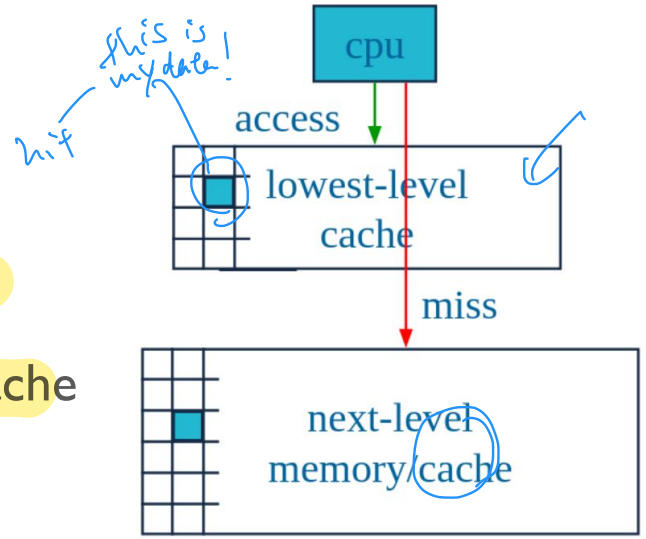


# Caching issues

- On memory access
  - How do I know if this is a hit or miss?
- On cache miss
  - Where to put the new data?
  - What data to kick out?
    - Typically select the Least Recently Used (LRU)
  - How to keep track of what data this is?



# Cache Fundamentals




- Cache hit: access where data found in cache
- Cache miss: access where data is NOT in cache
- Hit time: time to access the cache
- Miss penalty: time to move data from further level to closer, then to cpu
- Hit ratio: percentage of time data is found in cache
- Miss ratio: 1 - hit ratio

$$\frac{\#\ \text{hits}}{\#\ \text{accesses}}$$

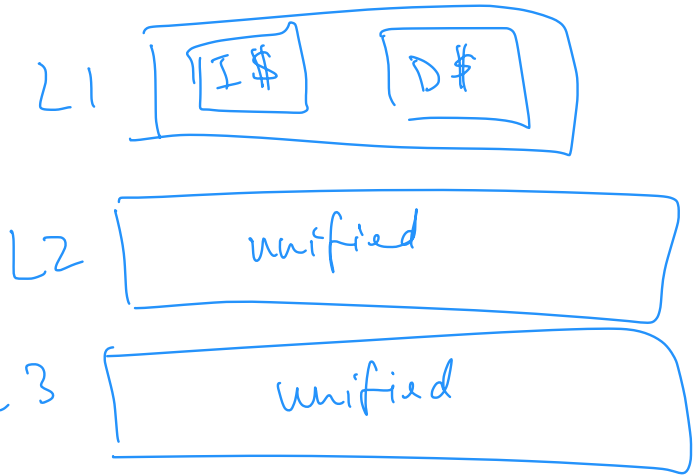
# Cache Fundamentals

F D E M W

I$ D$

↳ instr. cache ↳ data cache.

- Cache block size: amount of data transferred on cache miss
- Instruction cache: cache that only holds instructions
- Data cache: cache that only holds data
- Unified cache: cache that holds both



# Hardware implications on cache design

- Caches are the thing that make real workloads fast

# Hardware implications on cache design

- Caches are the thing that make real workloads fast



- Size of cache inversely proportional to its speed
  - Smaller caches are faster

SRAM more expensive than DRAM

- Every bit counts
  - Caches use as few bits as possible to do their work

Caches built using SRAM.

# 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];
    }
}
```

A[0] B[0] A[1] B[1]
A[0] B[0] A[1] B[1]

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]