# MEMORY HIERARCHY DESIGN

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

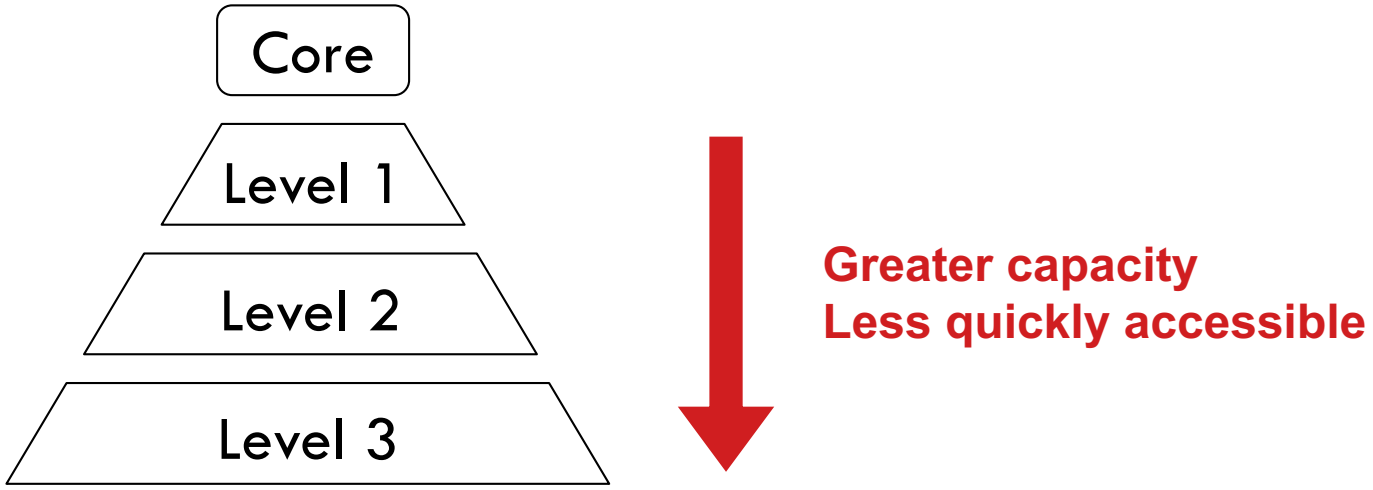
# Overview

- Announcement
  - Homework 4 will be released on Oct. 30st
- This lecture
  - Memory hierarchy
  - Memory technologies
  - Principle of locality
- Cache concepts

# Memory Hierarchy

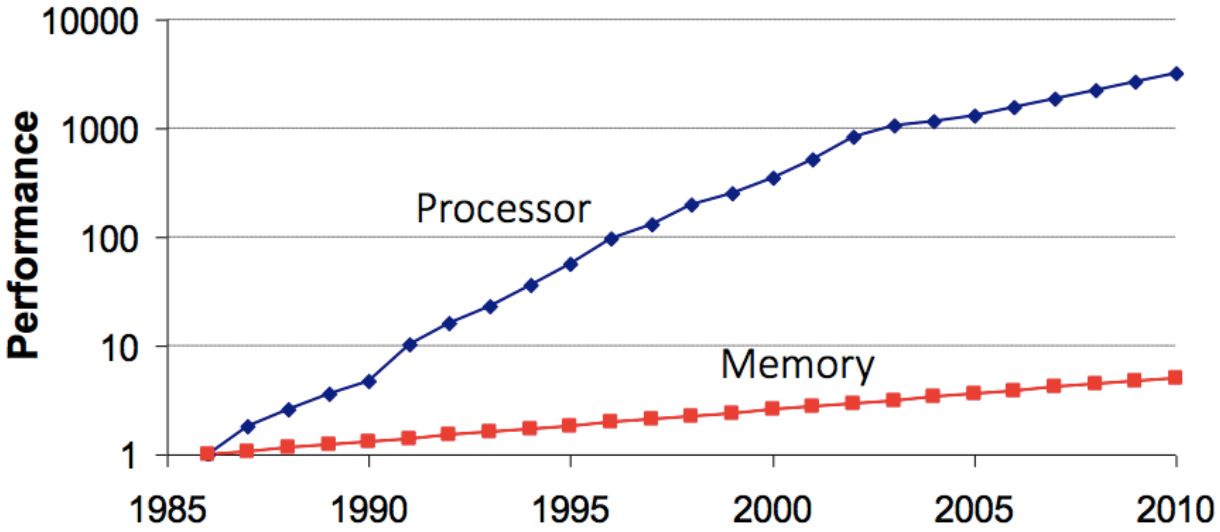
"Ideally one would desire an indefinitely large memory capacity such that any particular [...] word would be immediately available [...] We are [...] forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible."

*-- Burks, Goldstine, and von Neumann, 1946*



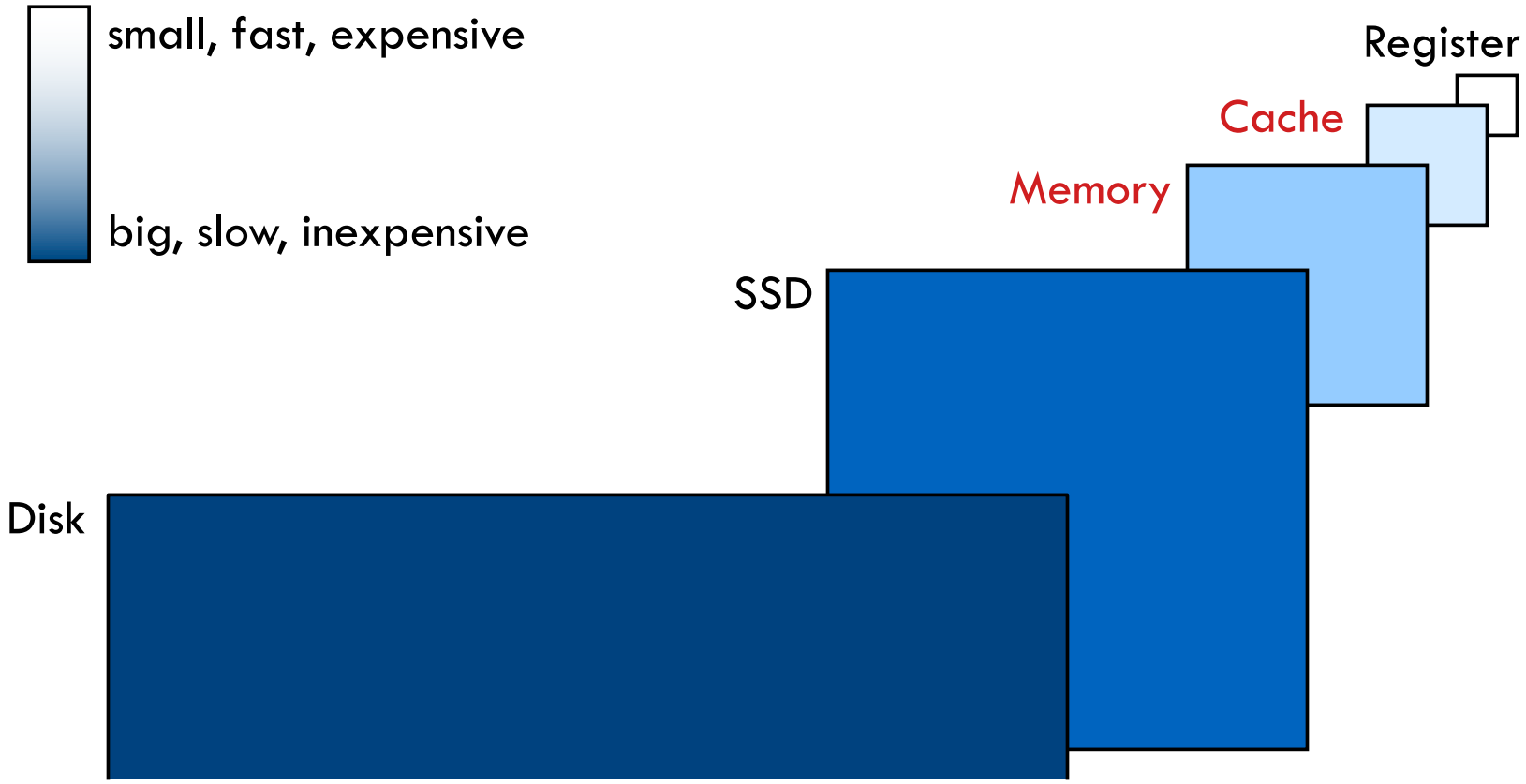
# The Memory Wall

- Processor-memory performance gap increased over 50% per year
  - Processor performance historically improved ~60% per year
  - Main memory access time improves ~5% per year



# Modern Memory Hierarchy

- Trade-off among memory speed, capacity, and cost

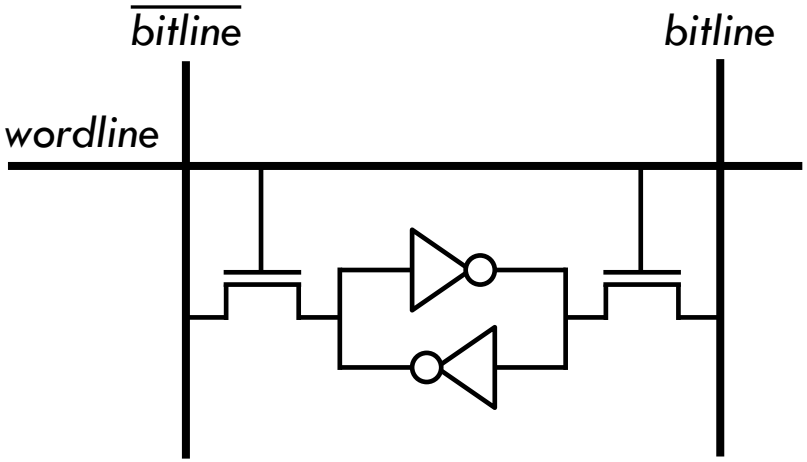


# Memory Technology

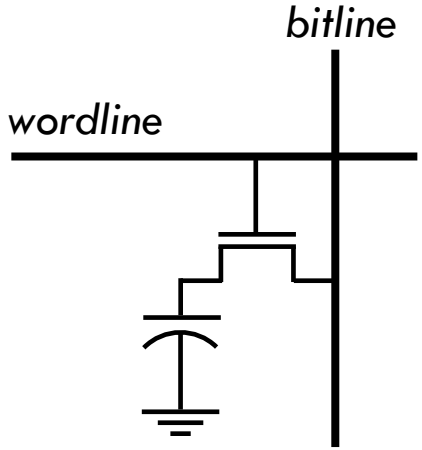
- Random access memory (RAM) technology
  - access time same for all locations (not so true anymore)
  - Static RAM (SRAM)
    - typically used for caches
    - 6T/bit; fast but – low density, high power, expensive
  - Dynamic RAM (DRAM)
    - typically used for main memory
    - 1T/bit; inexpensive, high density, low power – but slow

# RAM Cells

- 6T SRAM cell
  - internal feedback maintains data while power on




- 1T-1C DRAM cell
  - needs refresh regularly to preserve data



# Processor Cache

- Occupies a large fraction of die area in modern microprocessors

3-3.5 GHz
~$1000 (2014)

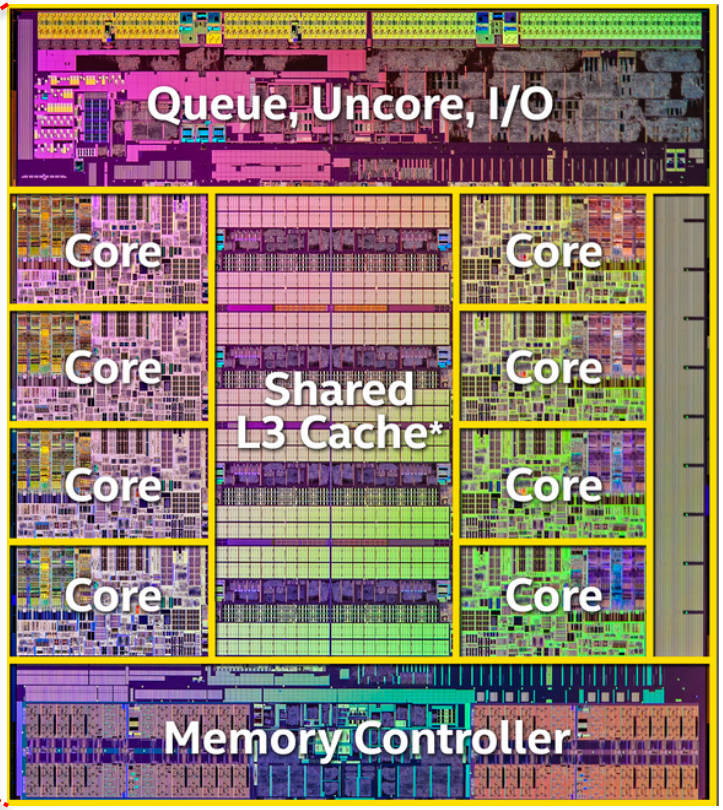

*Source: Intel Core i7*

# Processor Cache

- Occupies a large fraction of die area in modern microprocessors

3-3.5 GHz
~$1000 (2014)

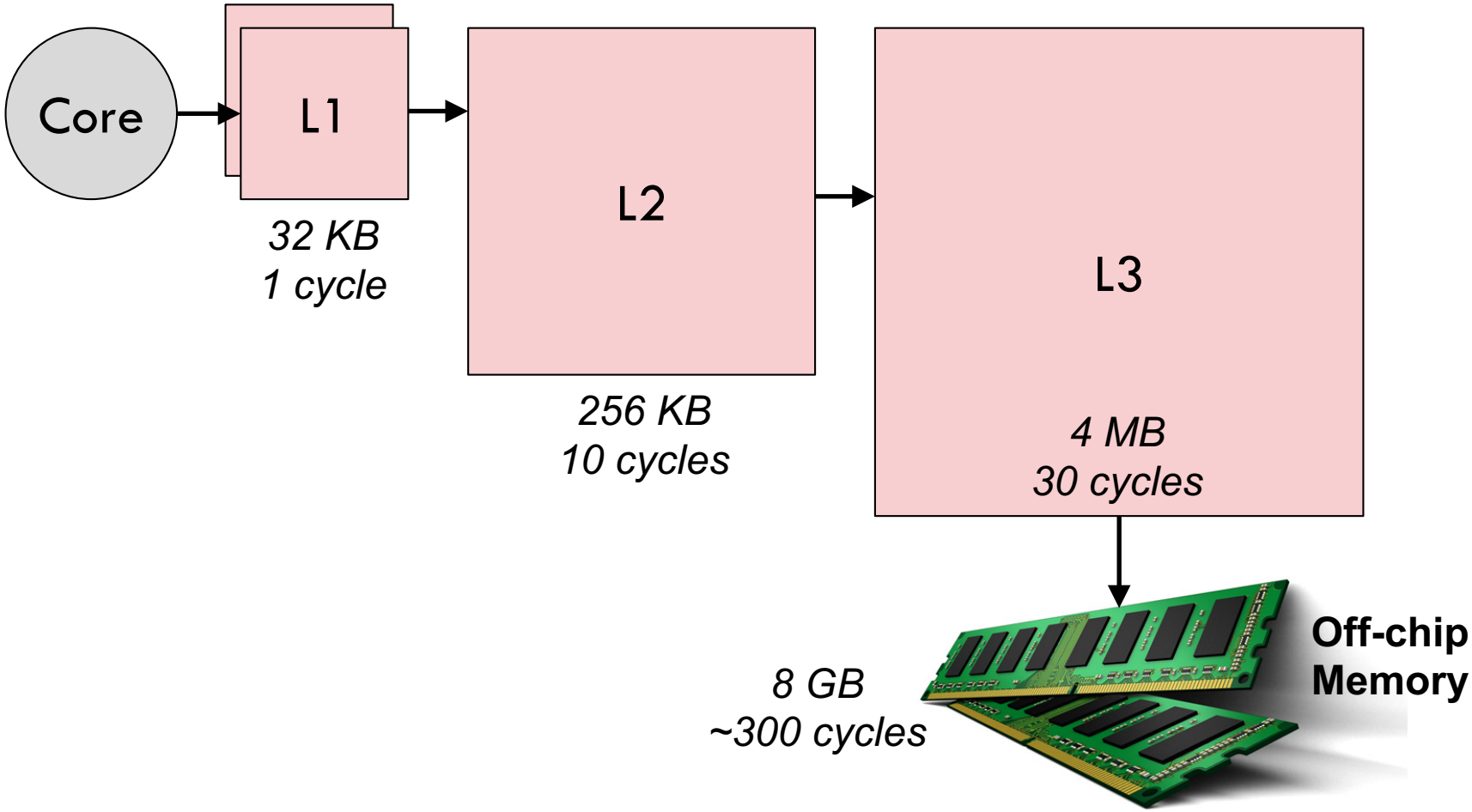

20MB of cache




*Source: Intel Core i7*

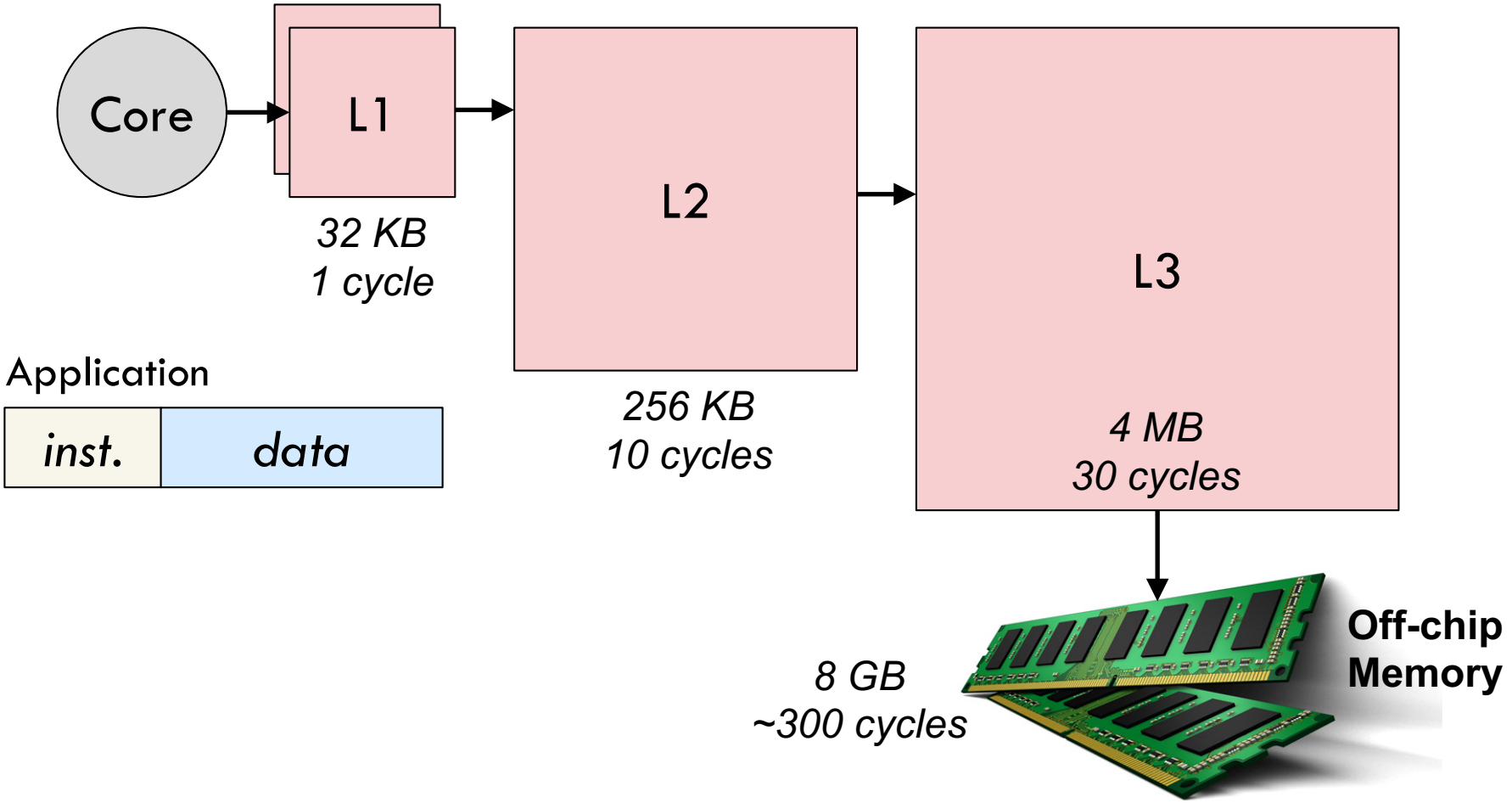
# Cache Hierarchy

- Example three-level cache organization



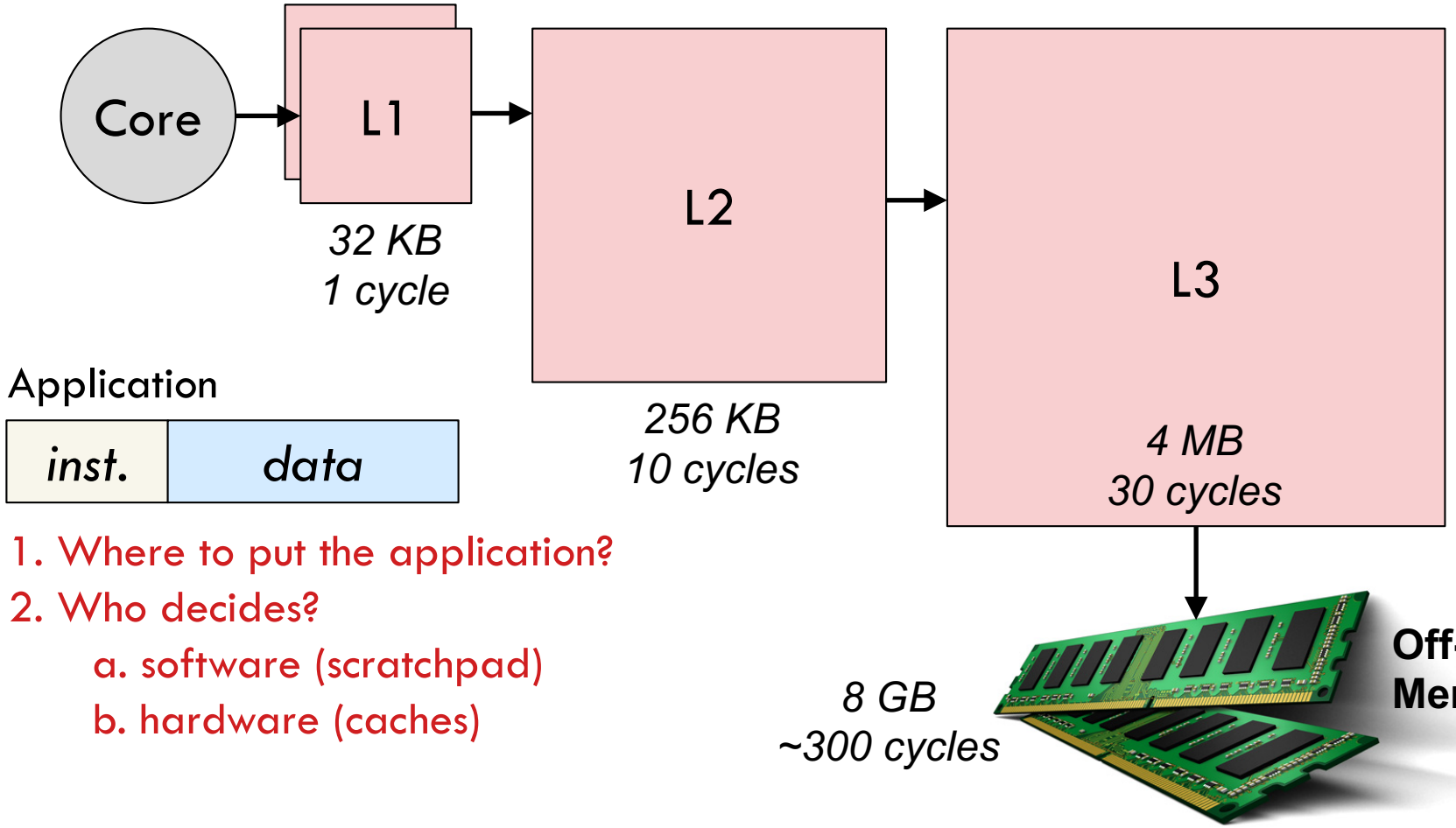
# Cache Hierarchy

- Example three-level cache organization



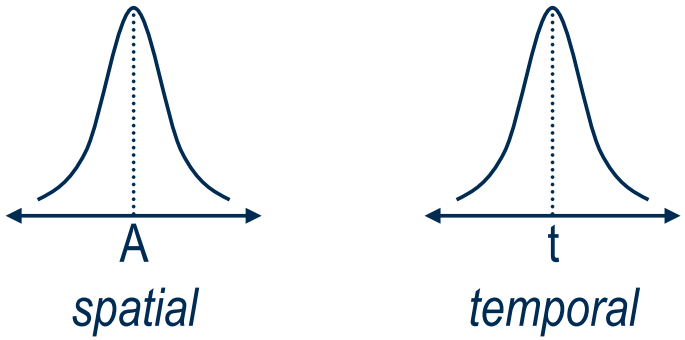
# Cache Hierarchy

- Example three-level cache organization



# Principle of Locality

- Memory references exhibit localized accesses
- Types of locality
  - *spatial*: probability of access to $A+\delta$ at time $t+\varepsilon$ highest when $\delta \rightarrow 0$
  - *temporal*: probability of accessing $A+\varepsilon$ at time $t+\delta$ highest when $\delta \rightarrow 0$

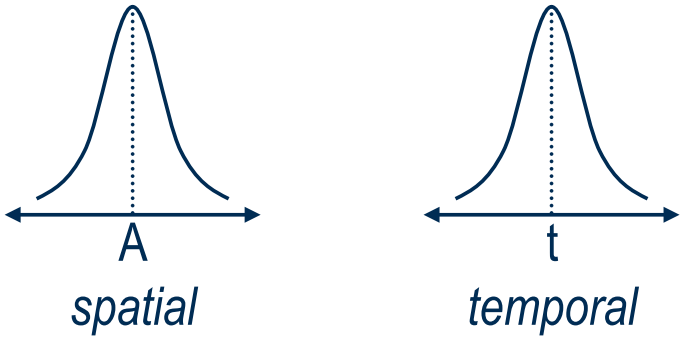



```
for (i=0; i<1000; ++i) {
        sum = sum + a[i];
}
```

Key idea: store local data in fast cache levels

# Principle of Locality

- Memory references exhibit localized accesses
- Types of locality
  - *spatial*: probability of access to $A+\delta$ at time $t+\varepsilon$ highest when $\delta \rightarrow 0$
  - *temporal*: probability of accessing $A+\varepsilon$ at time $t+\delta$ highest when $\delta \rightarrow 0$




```
for (i=0; i<1000; ++i) {
        sum = sum + a[i];
}
```

temporal spatial

Key idea: store local data in fast cache levels

# Cache Terminology

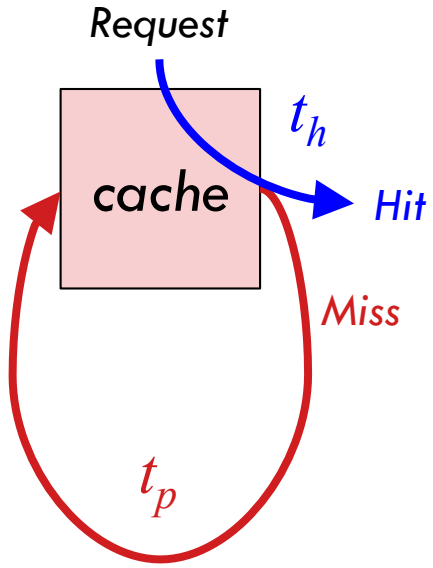
- Block (cache line): unit of data access
- Hit: accessed data found at current level
  - *hit rate: fraction of accesses that finds the data*
  - *hit time: time to access data on a hit*
- Miss: accessed data NOT found at current level
  - miss rate: 1 – hit rate
  - miss penalty: time to get block from lower level

*hit time << miss penalty*

# Cache Performance

- Average Memory Access Time (AMAT)

| Outcome | Rate | Access Time |
|---|---|---|
| Hit | $r_h$ | $t_h$ |
| Miss | $r_m$ | $t_h + t_p$ |




$$AMAT = r_h t_h + r_m (t_h + t_p)$$
$$r_h = 1 - r_m$$

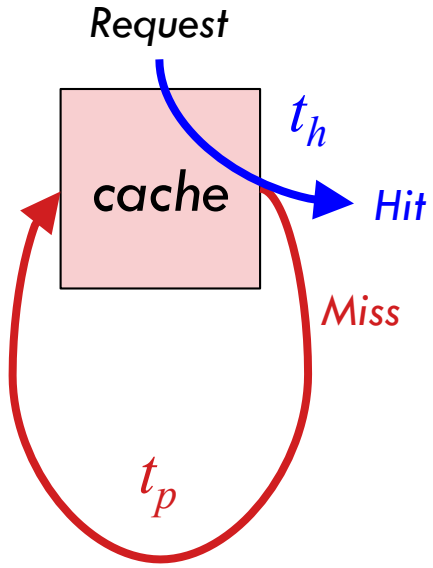
$$AMAT = t_h + r_m t_p$$

*problem: hit rate is 90%; hit time is 2 cycles; and accessing the lower level takes 200 cycles; find the average memory access time?*

# Cache Performance

- Average Memory Access Time (AMAT)

| Outcome | Rate | Access Time |
|---|---|---|
| Hit | $r_h$ | $t_h$ |
| Miss | $r_m$ | $t_h + t_p$ |




$$AMAT = r_h t_h + r_m (t_h + t_p)$$
$$r_h = 1 - r_m$$

$$AMAT = t_h + r_m t_p$$

*problem: hit rate is 90%; hit time is 2 cycles; and accessing the lower level takes 200 cycles; find the average memory access time?*

AMAT = 2 + 0.1x200 = 22 cycles

# Example Problem

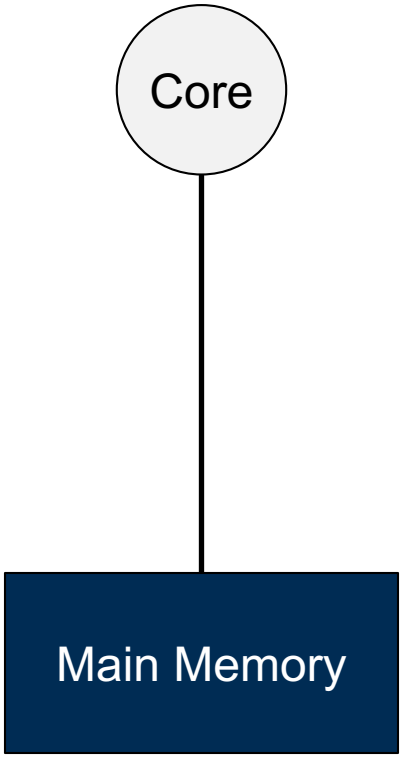
- Assume that the miss rate for instructions is 5%; the miss rate for data is 8%; the data references per instruction is 40%; and the miss penalty is 20 cycles; find performance relative to perfect cache with no misses

# Example Problem

- Assume that the miss rate for instructions is 5%; the miss rate for data is 8%; the data references per instruction is 40%; and the miss penalty is 20 cycles; find performance relative to perfect cache with no misses
  - misses/instruction = 0.05 + 0.08 x 0.4 = 0.082
  - Assuming hit time =1
    - AMAT = 1 + 0.082x20 = 2.64
    - Relative performance = 1/2.64

# Summary: Cache Performance

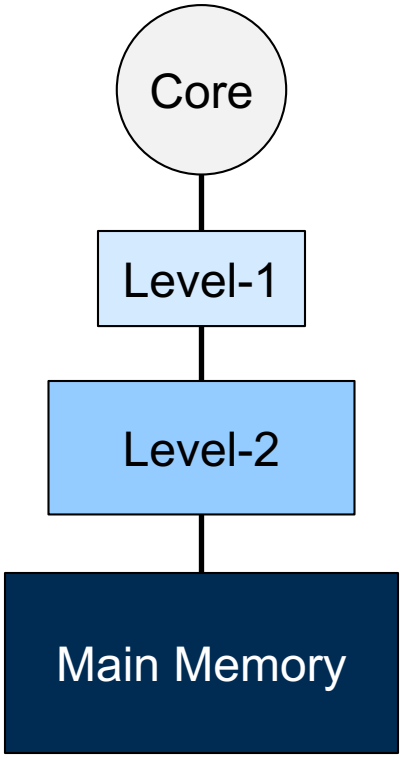
- Bridging the processor-memory performance gap




Main memory access time: 300 cycles

# Summary: Cache Performance

- Bridging the processor-memory performance gap




Main memory access time: 300 cycles

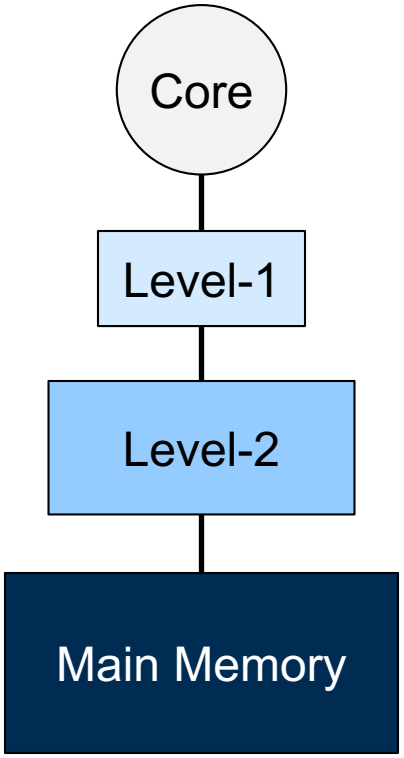
Two level cache

- L1: 2 cycles hit time; 60% hit rate
- L2: 20 cycles hit time; 70% hit rate

What is the average mem access time?

# Summary: Cache Performance

- Bridging the processor-memory performance gap




Main memory access time: 300 cycles

Two level cache

- L1: 2 cycles hit time; 60% hit rate
- L2: 20 cycles hit time; 70% hit rate

What is the average mem access time?

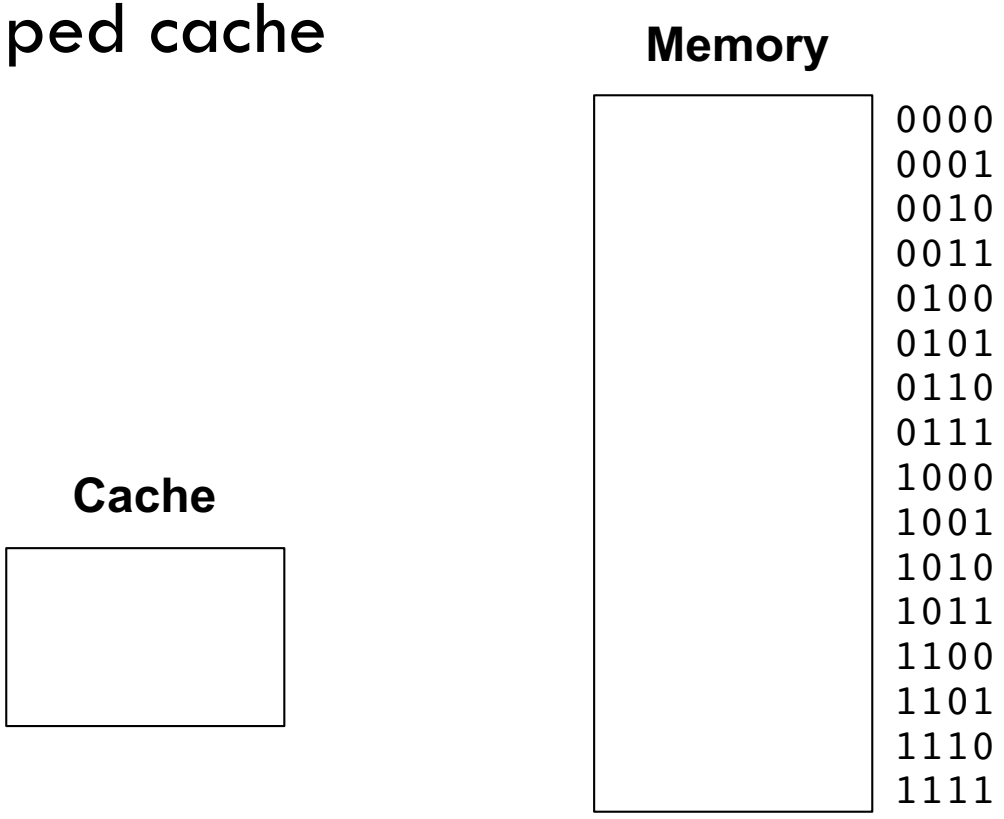
$$AMAT = t_{h1} + r_{m1}\, t_{p1}$$

$$t_{p1} = t_{h2} + r_{m2}\, t_{p2}$$

$$AMAT = 46$$

# Cache Addressing

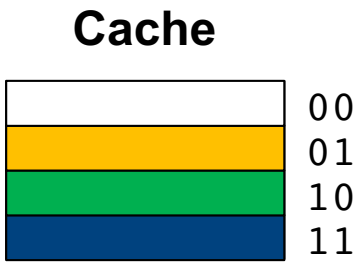
- Instead of specifying cache address we specify main memory address
- Simplest: direct-mapped cache

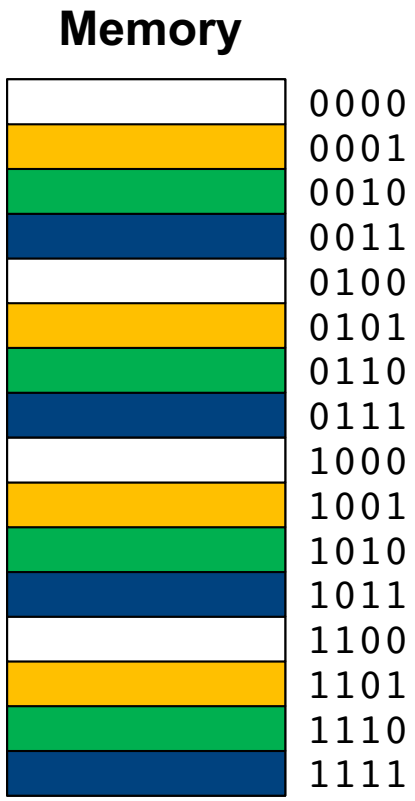


# Cache Addressing

- Instead of specifying cache address we specify main memory address
- Simplest: direct-mapped cache

Note: each memory address maps to a single cache location determined by modulo hashing



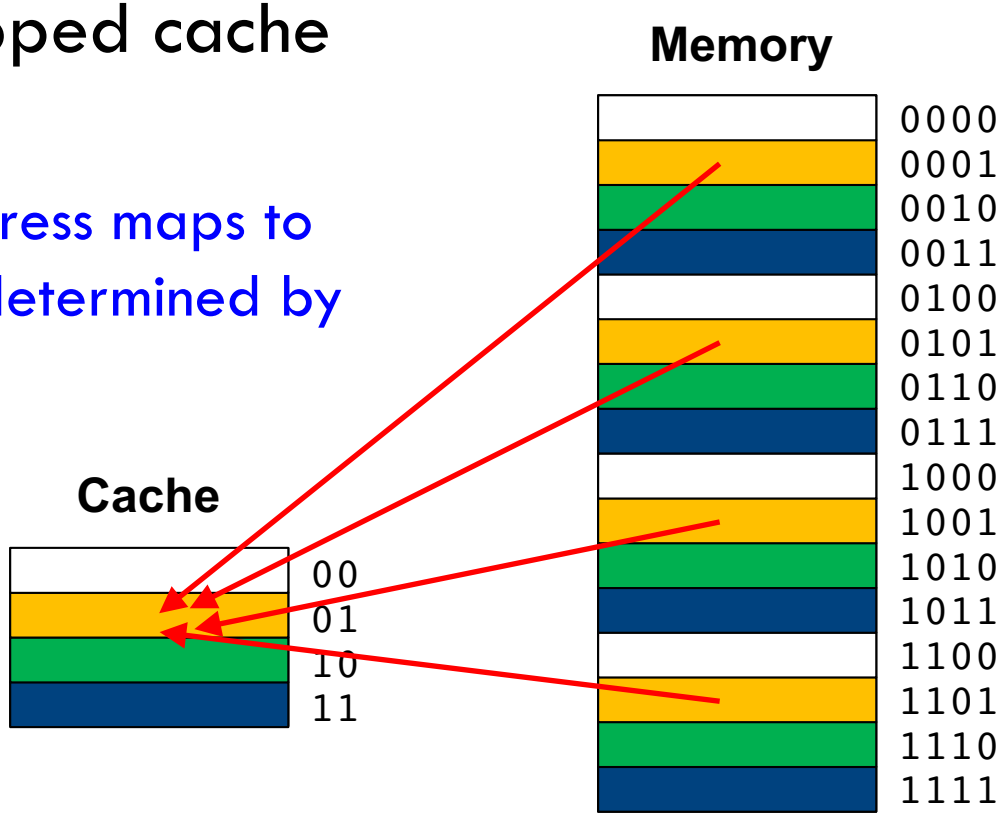



# Cache Addressing

- Instead of specifying cache address we specify main memory address
- Simplest: direct-mapped cache

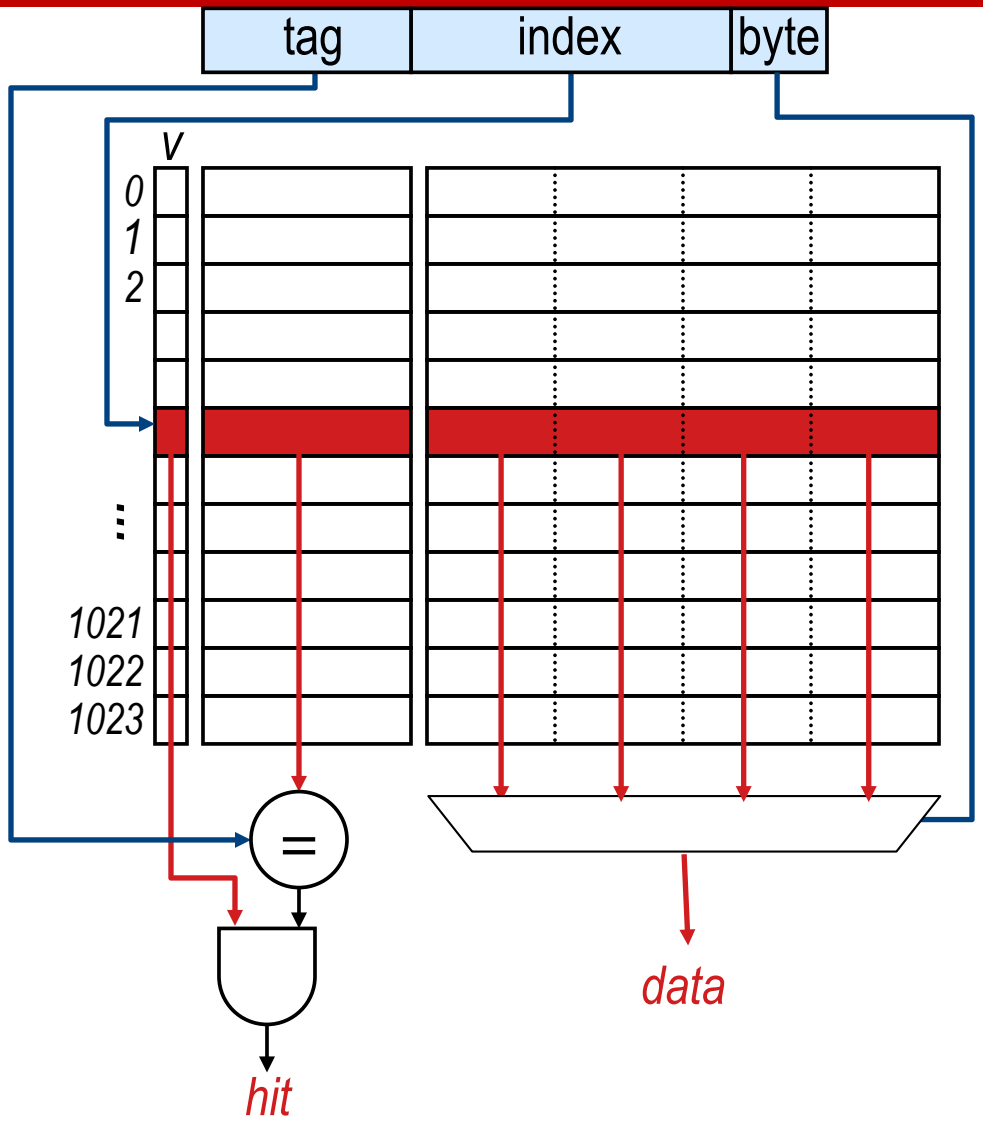
Note: each memory address maps to a single cache location determined by modulo hashing

**How to exactly specify which blocks are in the cache?**



# Direct-Mapped Lookup

- Byte offset: to select the requested byte
- Tag: to maintain the address
- Valid flag (v): whether content is meaningful
- Data and tag are always accessed



# Example Problem

- Find the size of tag, index, and offset bits for an 8MB, direct-mapped L3 cache with 64B cache blocks. Assume that the processor can address up to 4GB of main memory.

# Example Problem

- Find the size of tag, index, and offset bits for an 8MB, direct-mapped L3 cache with 64B cache blocks. Assume that the processor can address up to 4GB of main memory.
- $4GB = 2^{32}$ B → address bits = 32
- $64B = 2^{6}$ B → byte offset bits = 6
- $8MB/64B = 2^{17}$ → index bits = 17
- tag bits = 32 – 6 – 17 = 9

# Cache Optimizations

- How to improve cache performance?

$$AMAT = t_h + r_m \, t_p$$

- Reduce hit time ($t_h$)
- Improve hit rate ($1 - r_m$)
- Reduce miss penalty ($t_p$)

# Cache Optimizations

- How to improve cache performance?

$$AMAT = t_h + r_m \, t_p$$

- Reduce hit time ($t_h$)
  - Memory technology, critical access path
- Improve hit rate ($1 - r_m$)
- Reduce miss penalty ($t_p$)

# Cache Optimizations

- How to improve cache performance?

$$AMAT = t_h + r_m\, t_p$$

- Reduce hit time ($t_h$)
  - Memory technology, critical access path
- Improve hit rate ($1 - r_m$)
  - Size, associativity, placement/replacement policies
- Reduce miss penalty ($t_p$)

# Cache Optimizations

- How to improve cache performance?

$$AMAT = t_h + r_m \, t_p$$

- Reduce hit time ($t_h$)
  - Memory technology, critical access path
- Improve hit rate ($1 - r_m$)
  - Size, associativity, placement/replacement policies
- Reduce miss penalty ($t_p$)
  - Multi level caches, data prefetching