

# Memory & Caches I

CSE 351 Autumn 2022

## Instructor:

Justin Hsia

## Teaching Assistants:

Angela Xu

Arjun Narendra

Armin Magness

Assaf Vayner

Carrie Hu

Clare Edmonds

David Dai

Dominick Ta

Effie Zheng

James Froelich

Jenny Peng

Kristina Lansang

Paul Stevans

Renee Ruan

Vincent Xiao

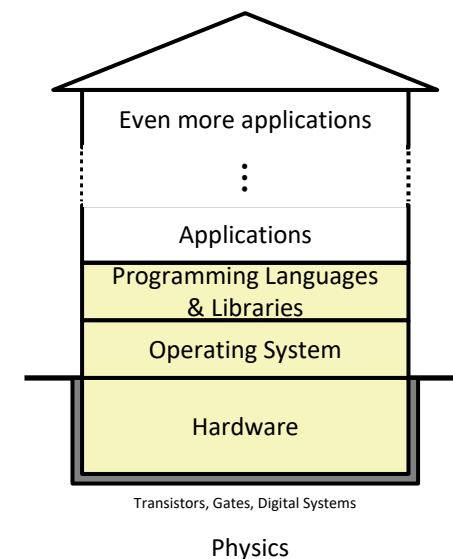


# Relevant Course Information

- ❖ hw15 due Monday, hw16 due Wednesday
  - Veteran's Day next Friday (11/11); no lecture
- ❖ Lab 3 due next Friday (11/11)
  - Make sure to look at HW15 before starting
- ❖ Midterm starts tomorrow (11/3-5)
  - Private posts on Ed Discussion, please!

# The Hardware/Software Interface

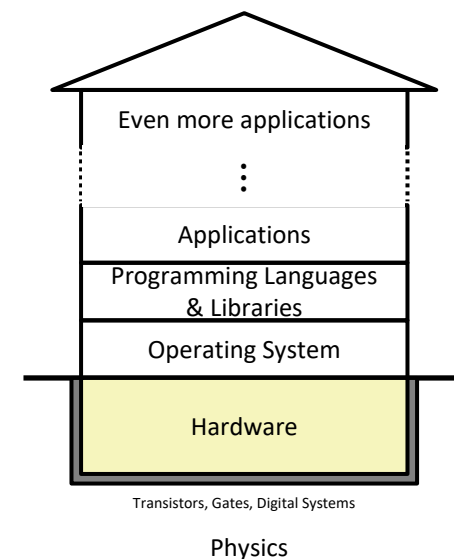
- ❖ Topic Group 1: **Data**
  - Memory, Data, Integers, Floating Point, Arrays, Structs
- ❖ Topic Group 2: **Programs**
  - x86-64 Assembly, Procedures, Stacks, Executables
- ❖ Topic Group 3: **Scale & Coherence**
  - Caches, Processes, Virtual Memory, Memory Allocation



# The Hardware/Software Interface

## ❖ Topic Group 3: **Scale & Coherence**

- **Caches**, Processes, Virtual Memory, Memory Allocation



- ❖ How do we maintain logical consistency in the face of more data and more processes?
  - How do we support control flow both within many processes and things external to the computer?
  - How do we support data access, including dynamic requests, across multiple processes?

# Aside: Units and Prefixes (Review)

- ❖ Here focusing on large numbers (exponents  $> 0$ )
- ❖ Note that  $10^3 \approx 2^{10}$
- ❖ SI prefixes are *ambiguous* if base 10 or 2
- ❖ IEC prefixes are *unambiguously* base 2

SIZE PREFIXES ( $10^x$  for Disk, Communication;  $2^x$  for Memory)

SI Size	Prefix	Symbol	IEC Size	Prefix	Symbol
$10^3$	Kilo-	K	$2^{10}$	Kibi-	Ki
$10^6$	Mega-	M	$2^{20}$	Mebi-	Mi
$10^9$	Giga-	G	$2^{30}$	Gibi-	Gi
$10^{12}$	Tera-	T	$2^{40}$	Tebi-	Ti
$10^{15}$	Peta-	P	$2^{50}$	Pebi-	Pi
$10^{18}$	Exa-	E	$2^{60}$	Exbi-	Ei
$10^{21}$	Zetta-	Z	$2^{70}$	Zebi-	Zi
$10^{24}$	Yotta-	Y	$2^{80}$	Yobi-	Yi

# How to Remember?

- ❖ Will be given to you on Final reference sheet
  
- ❖ Mnemonics
  - There unfortunately isn't one well-accepted mnemonic
    - But that shouldn't stop you from trying to come with one!
  - **K**iller **M**echanical **G**iraffe **T**eaches **P**et, **E**xting **Z**ebra to **Y**odel
  - **K**irby **M**issed **G**anondorf **T**erribly, **P**otentially **E**xterminating **Z**elda and **Y**oshi
  - xkcd: **K**arl **M**arx **G**ave **T**he **P**roletariat **E**leven **Z**eppelin, **Y**o
    - <https://xkcd.com/992/>
  - Post your best on Ed Discussion!

# Reading Review

- ❖ Terminology:
  - Caches: cache blocks, cache hit, cache miss
  - Principle of locality: temporal and spatial
  - Average memory access time (AMAT): hit time, miss penalty, hit rate, miss rate
  
- ❖ Questions from the Reading?

# Review Questions

- ❖ Convert the following to or from IEC:
  - 512 Ki-books
  - $2^{27}$  caches
  
- ❖ Compute the average memory access time (AMAT) for the following system properties:
  - Hit time of 1 ns
  - Miss rate of 1%
  - Miss penalty of 100 ns

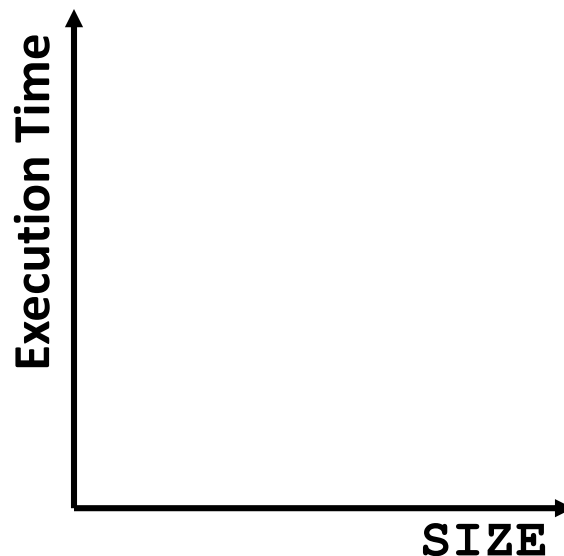


# How does execution time grow with SIZE?

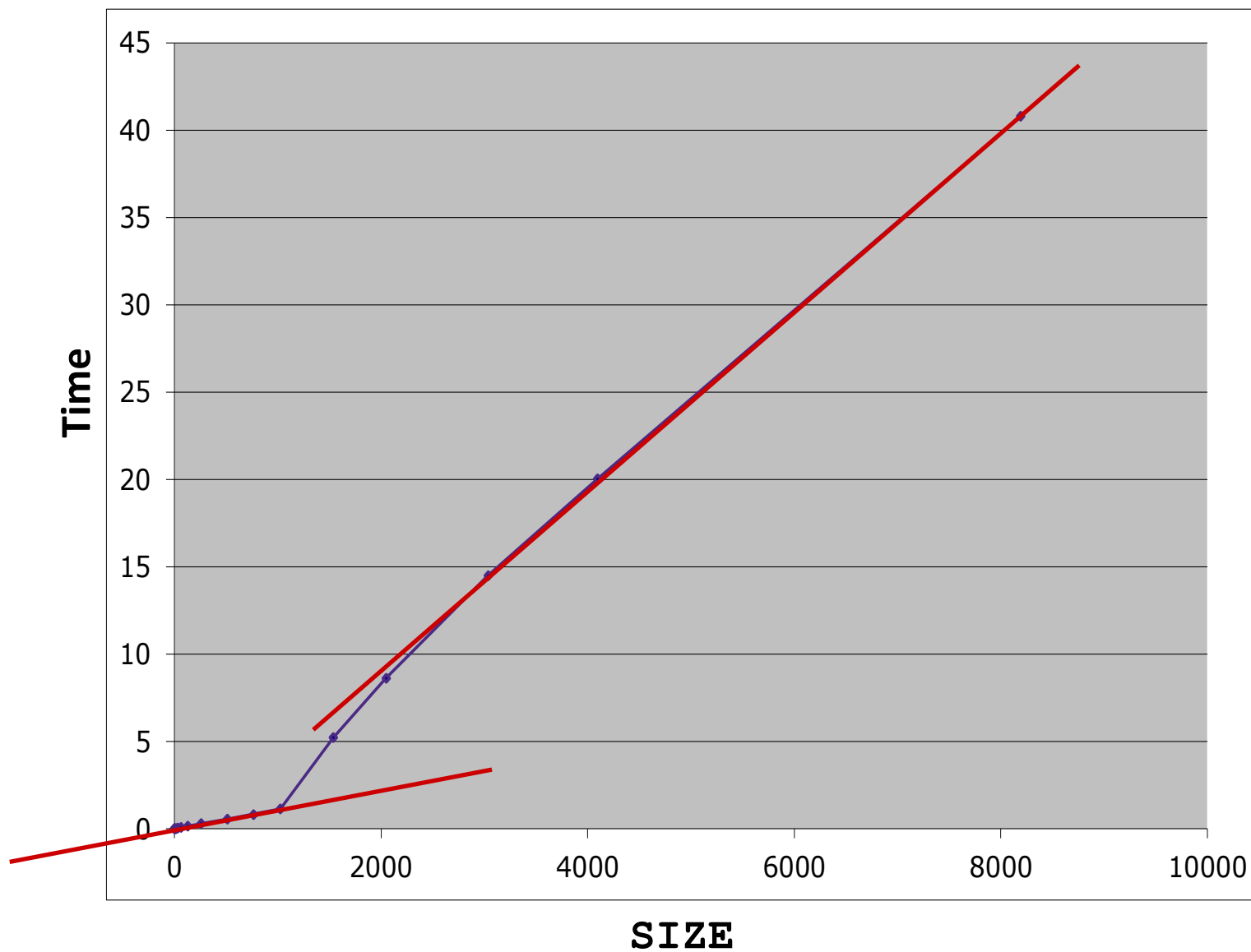
```
int array[SIZE];
int sum = 0;

for (int i = 0; i < 200000; i++) {
    for (int j = 0; j < SIZE; j++) {
        sum += array[j];
    }
}
```

**Plot:**



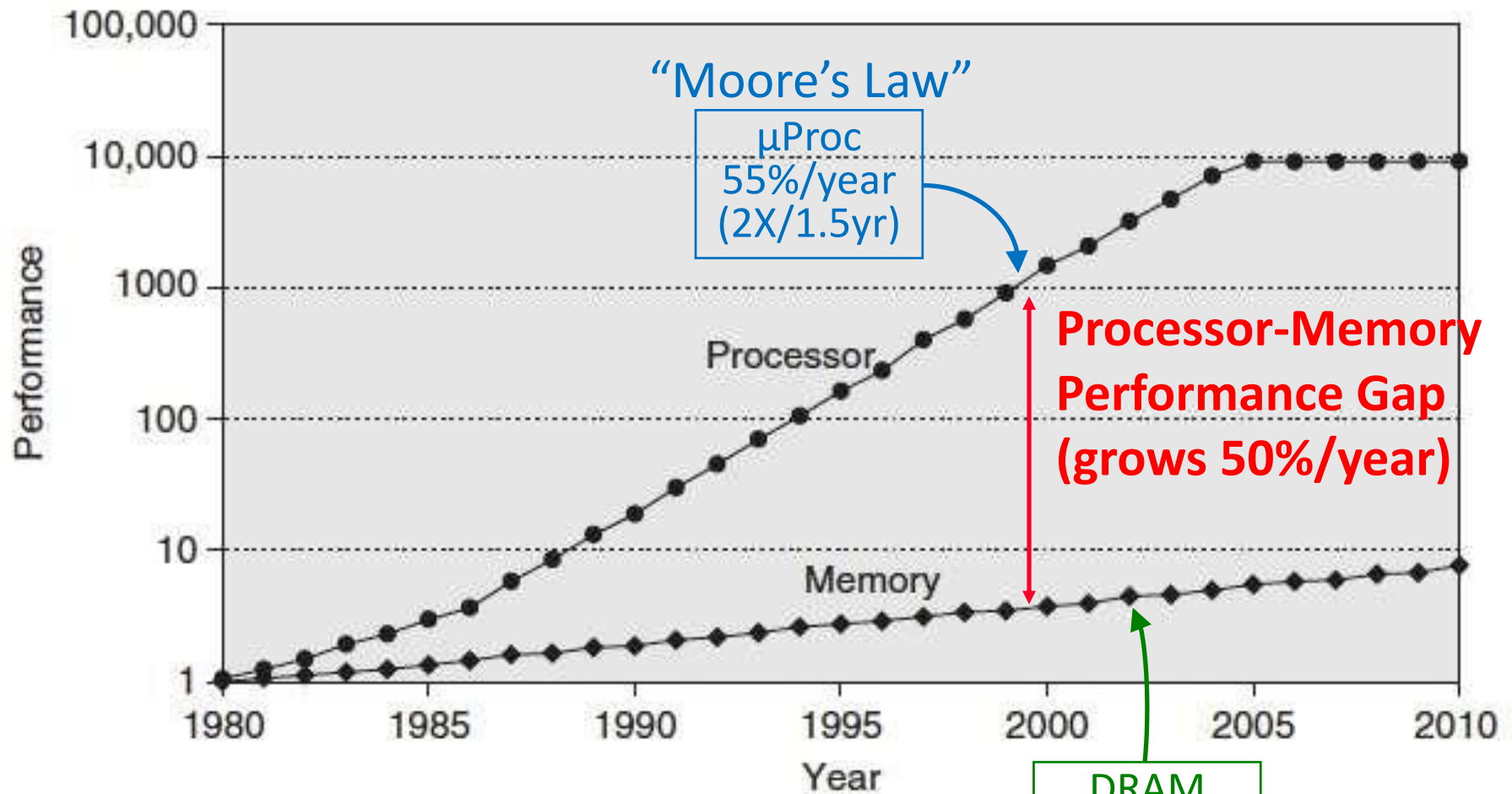
# Actual Data



# Making memory accesses fast!

- ❖ **Cache basics**
- ❖ **Principle of locality**
- ❖ **Memory hierarchies**
- ❖ Cache organization
- ❖ Program optimizations that consider caches

# Processor-Memory Gap



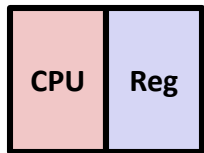
**1989** first Intel CPU with cache on chip

**1998** Pentium III has two cache levels on chip

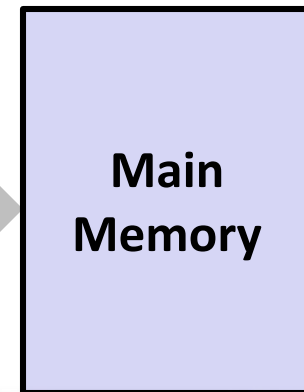
DRAM  
7%/year  
(2X/10yrs)

# Problem: Processor-Memory Bottleneck

Processor performance  
doubled about  
every 18 months



Bus latency / bandwidth  
evolved much slower



**Core 2 Duo:**  
Can process at least  
256 Bytes/cycle

**Core 2 Duo:**  
Bandwidth  
2 Bytes/cycle  
Latency  
100-200 cycles (30-60ns)

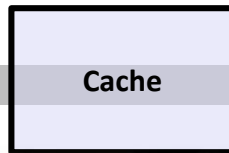
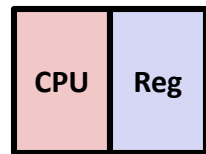


**Problem: lots of waiting on memory**

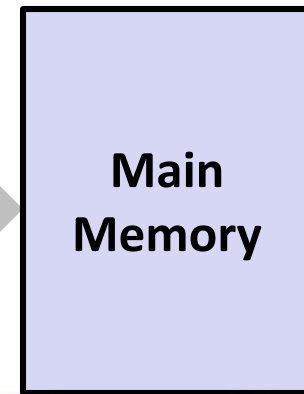
*cycle: single machine step (fixed-time)*

# Problem: Processor-Memory Bottleneck

Processor performance  
doubled about  
every 18 months



Bus latency / bandwidth  
evolved much slower



**Core 2 Duo:**  
Can process at least  
256 Bytes/cycle

**Core 2 Duo:**  
Bandwidth  
2 Bytes/cycle  
Latency  
100-200 cycles (30-60ns)



**Solution: caches**



*cycle: single machine step (fixed-time)*

# Cache

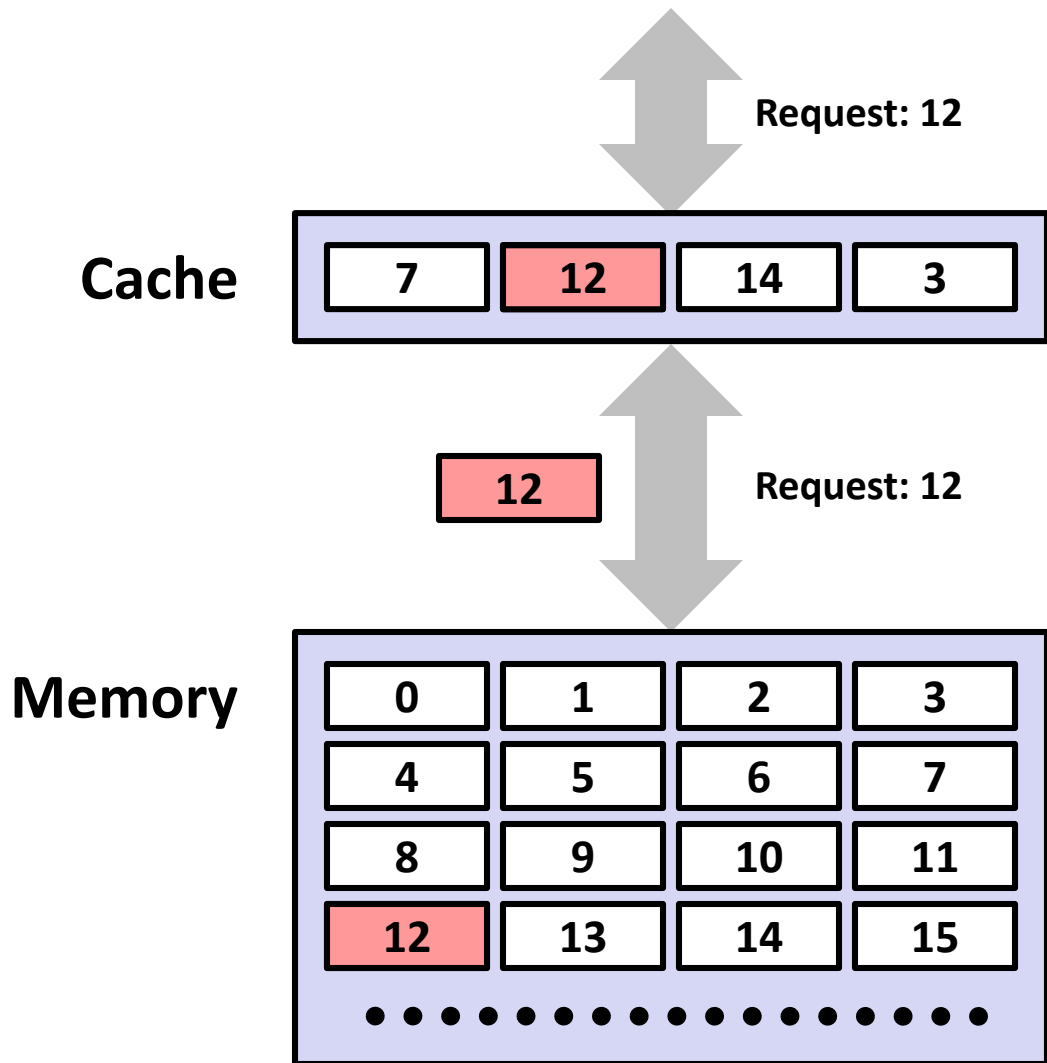
- ❖ Pronunciation: “cash”
  - We abbreviate this as “\$”
- ❖ English: A hidden storage space for provisions, weapons, and/or treasures
- ❖ Computer: Memory with short access time used for the storage of frequently or recently used instructions (i-cache/I\$) or data (d-cache/D\$)
  - *More generally*: Used to optimize data transfers between any system elements with different characteristics (network interface cache, I/O cache, etc.)







# General Cache Concepts: Miss (Review)



*Data in block b is needed*

*Block b is not in cache:  
Miss!*

*Block b is fetched from  
memory*

*Block b is stored in cache*

- **Placement policy:**  
determines where b goes
- **Replacement policy:**  
determines which block  
gets evicted (victim)

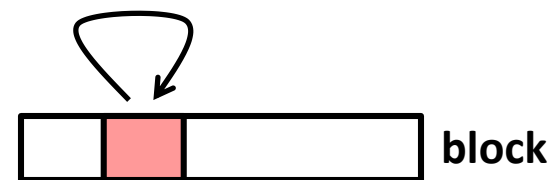
*Data is returned to CPU*

# Why Caches Work (Review)

- ❖ **Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently

# Why Caches Work (Review)

- ❖ **Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently



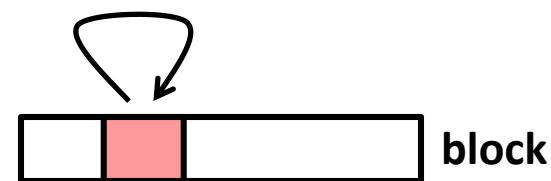
- ❖ **Temporal locality:**
  - Recently referenced items are *likely* to be referenced again in the near future

# Why Caches Work (Review)

- ❖ **Locality:** Programs tend to use data and instructions with addresses near or equal to those they have used recently

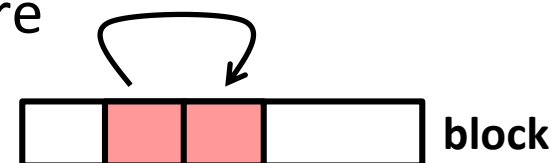
- ❖ **Temporal locality:**

- Recently referenced items are *likely* to be referenced again in the near future



- ❖ **Spatial locality:**

- Items with nearby addresses *tend* to be referenced close together in time



- ❖ How do caches take advantage of this?

# Example: Any Locality?

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

## ❖ Data:

- Temporal: sum referenced in each iteration
- Spatial: consecutive elements of array a [ ] accessed

## ❖ Instructions:

- Temporal: cycle through loop repeatedly
- Spatial: reference instructions in sequence

# Locality Example #1

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];

    return sum;
}
```

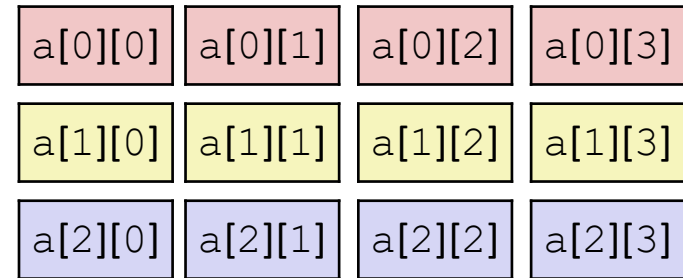
# Locality Example #1

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];

    return sum;
}
```

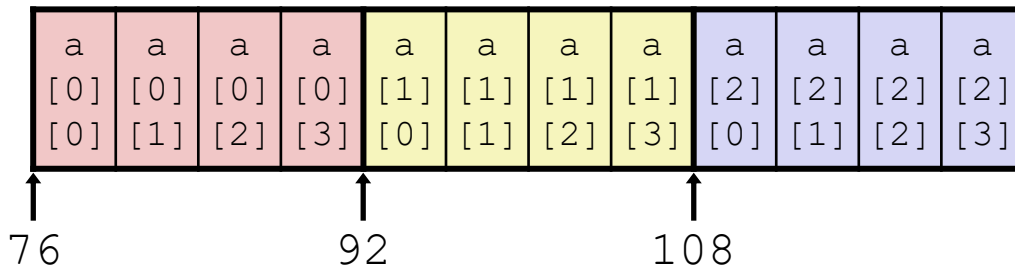
**M = 3, N = 4**



**Access Pattern:**  
stride = ?

- 1) a[0][0]
- 2) a[0][1]
- 3) a[0][2]
- 4) a[0][3]
- 5) a[1][0]
- 6) a[1][1]
- 7) a[1][2]
- 8) a[1][3]
- 9) a[2][0]
- 10) a[2][1]
- 11) a[2][2]
- 12) a[2][3]

## Layout in Memory



**Note:** 76 is just one possible starting address of array a



# Locality Example #2

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];

    return sum;
}
```

# Locality Example #2

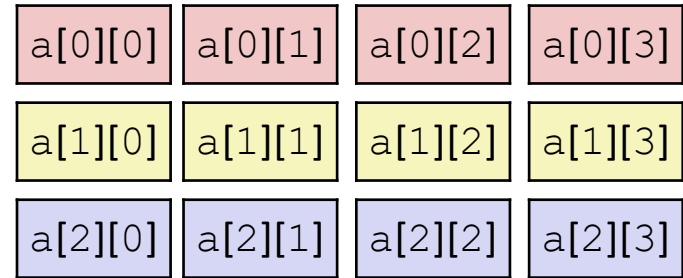
```

int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];

    return sum;
}
    
```

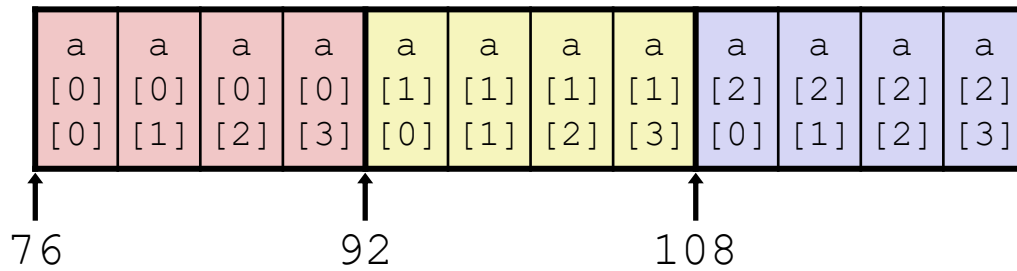
**M = 3, N = 4**



**Access Pattern:**  
stride = ?

- 1) a[0][0]
- 2) a[1][0]
- 3) a[2][0]
- 4) a[0][1]
- 5) a[1][1]
- 6) a[2][1]
- 7) a[0][2]
- 8) a[1][2]
- 9) a[2][2]
- 10) a[0][3]
- 11) a[1][3]
- 12) a[2][3]

## Layout in Memory



# Locality Example #3

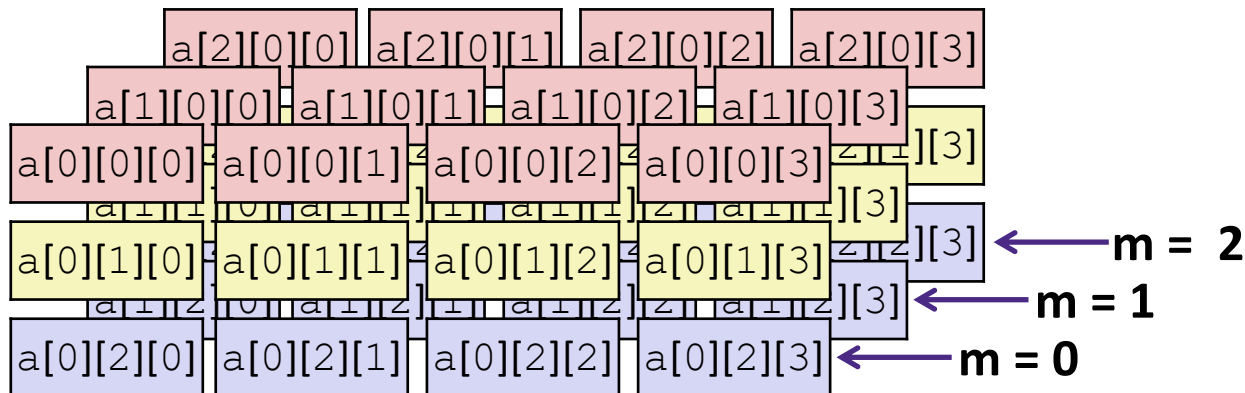
```

int sum_array_3D(int a[M][N][L])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < L; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];

    return sum;
}
    
```

- ❖ What is wrong with this code?
- ❖ How can it be fixed?



# Locality Example #3

```

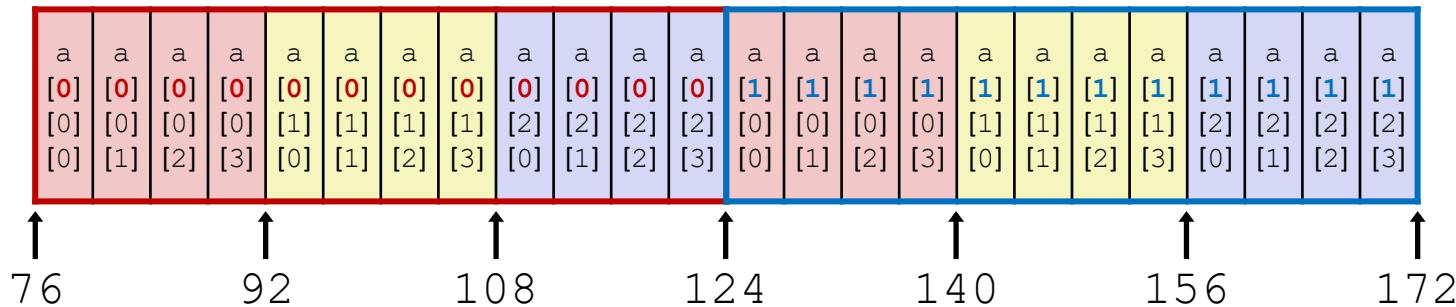
int sum_array_3D(int a[M][N][L])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < L; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];

    return sum;
}
    
```

- ❖ What is wrong with this code?
- ❖ How can it be fixed?

**Layout in Memory (M = ?, N = 3, L = 4)**



# Cache Performance Metrics (Review)

- ❖ Huge difference between a cache hit and a cache miss
  - Could be 100x speed difference between accessing cache and main memory (measured in *clock cycles*)
- ❖ **Miss Rate (MR)**
  - Fraction of memory references not found in cache (misses / accesses) =  $1 - \text{Hit Rate}$
- ❖ **Hit Time (HT)**
  - Time to deliver a block in the cache to the processor
    - Includes time to determine whether the block is in the cache
- ❖ **Miss Penalty (MP)**
  - Additional time required because of a miss

# Cache Performance (Review)

- ❖ Two things hurt the performance of a cache:
  - Miss rate and miss penalty
- ❖ *Average Memory Access Time (AMAT)*: average time to access memory considering both hits and misses
  - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$**   
(abbreviated  $AMAT = HT + MR \times MP$ )
- ❖ 99% hit rate twice as good as 97% hit rate!
  - Assume HT of 1 clock cycle and MP of 100 clock cycles
  - 97%:  $AMAT =$
  - 99%:  $AMAT =$

# Practice Question

- ❖ **Processor specs:** 200 ps clock, MP of 50 clock cycles, MR of 0.02 misses/instruction, and HT of 1 clock cycle

AMAT =

- ❖ Which improvement would be best?

A. 190 ps clock

B. Miss penalty of 40 clock cycles

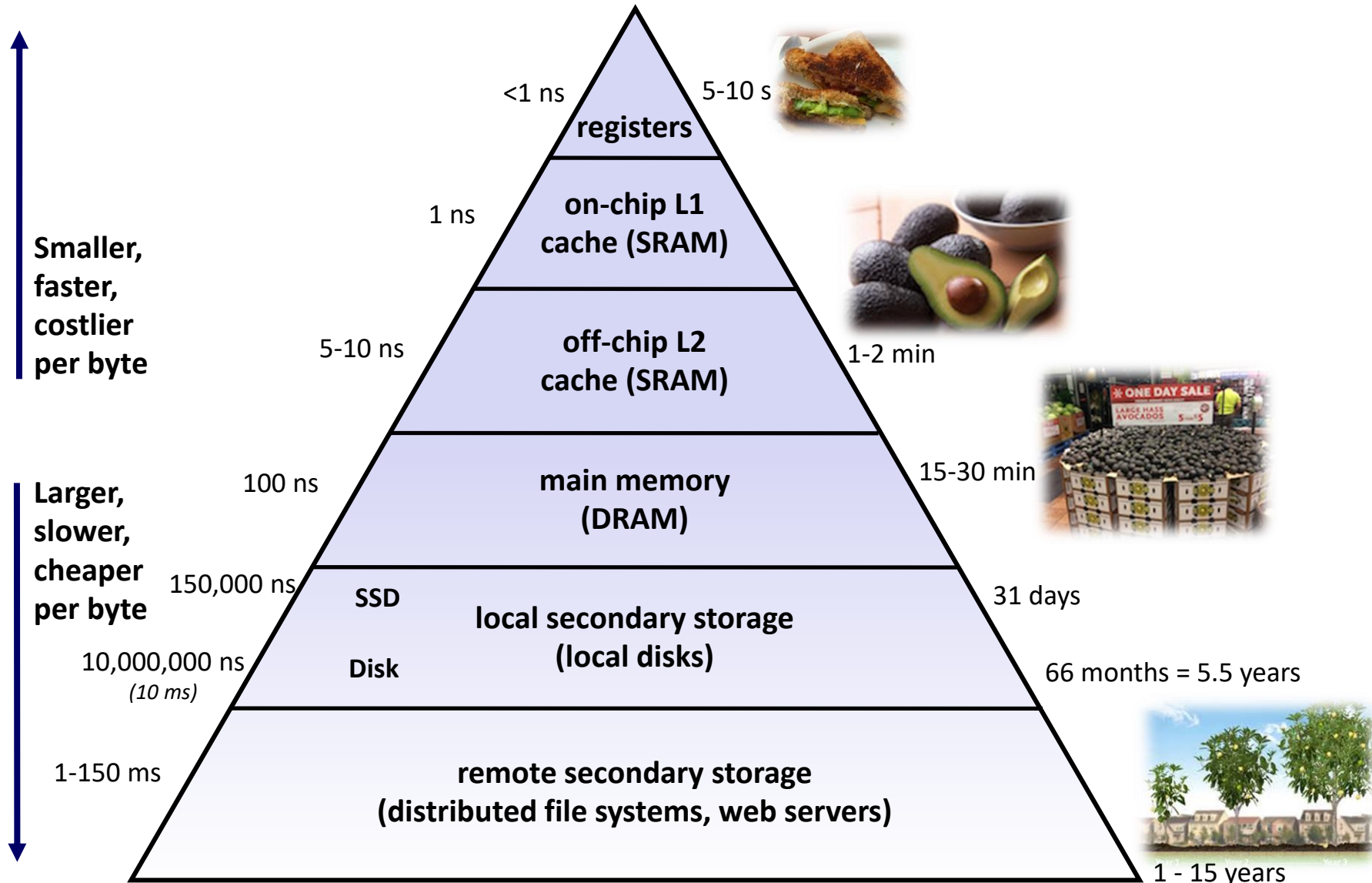
C. MR of 0.015 misses/instruction

# Can we have more than one cache?

- ❖ Why would we want to do that?
  - Avoid going to memory!
- ❖ Typical performance numbers:
  - Miss Rate
    - L1 MR = 3-10%
    - L2 MR = Quite small (*e.g.*, < 1%), depending on parameters, etc.
  - Hit Time
    - L1 HT = 4 clock cycles
    - L2 HT = 10 clock cycles
  - Miss Penalty
    - P = 50-200 cycles for missing in L2 & going to main memory
    - Trend: increasing!



# An Example Memory Hierarchy



# Summary

## ❖ Memory Hierarchy

- Successively higher levels contain “most used” data from lower levels
- Exploits *temporal and spatial locality*
- Caches are intermediate storage levels used to optimize data transfers between any system elements with different characteristics

## ❖ Cache Performance

- Ideal case: found in cache (hit)
- Bad case: not found in cache (miss), search in next level
- Average Memory Access Time (AMAT) =  $HT + MR \times MP$ 
  - Hurt by Miss Rate and Miss Penalty