

# Memory & Caches I

CSE 351 Spring 2024

## Instructor:

Elba Garza

## Teaching Assistants:

Ellis Haker

Adithi Raghavan

Aman Mohammed

Brenden Page

Celestine Buendia

Chloe Fong

Claire Wang

Hamsa Shankar

Maggie Jiang

Malak Zaki

Naama Amiel

Nikolas McNamee

Shananda Dokka

Stephen Ying

Will Robertson

My CPU when the L1 cache misses



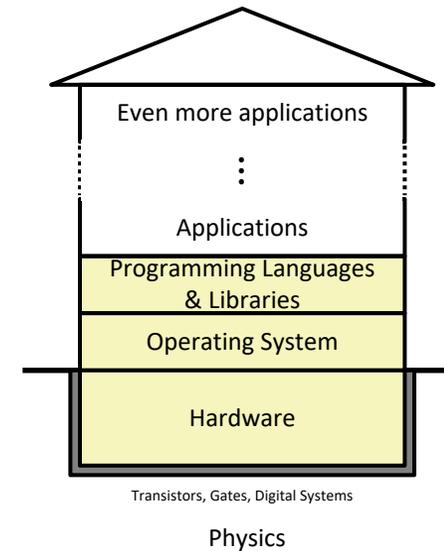
Playlist: [CSE 351 24Sp Lecture Tunes!](#)

# Relevant Course Information

- ❖ HW13/14 due Wednesday, May 1<sup>st</sup>; HW 15 due Friday May 3<sup>rd</sup>
- ❖ Mid-Quarter Survey (on Canvas) available Wednesday, May 1<sup>st</sup>
  - Due by Monday, May 6<sup>th</sup> — Let us know how the course is going for you, especially if you couldn't join for the in-person assessment last week!
- ❖ Take-home Midterm (May 6th & 7th)
  - Instructions will be posted in Ed discussion later tonight!
  - You may discuss *high-level concepts and give hints*, but **may not** solve the problems together. No group work!
  - We will be available on Ed Discussion (private posts, pls) and office hours to answer **clarifying questions**.
  - In-class Vote: Do y'all want in-person lecture on the 6<sup>th</sup>, or lecture recording?

# The Hardware/Software Interface

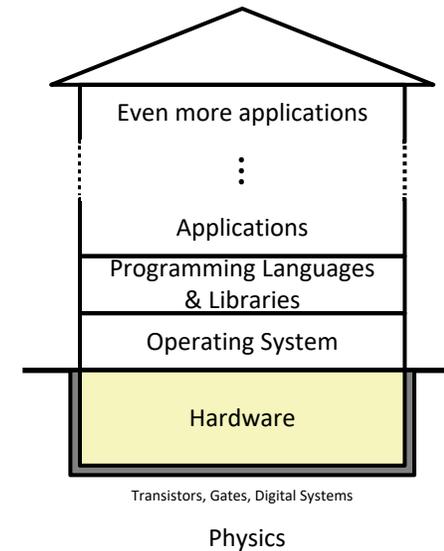
- ❖ Topic Group 1: **Data**
  - Memory, Data, Integers, Floating Point, Arrays, Structs
- ❖ Topic Group 2: **Programs**
  - x86-64 Assembly, Procedures, Stacks, Executables
- ❖ Topic Group 3: **Scale & Coherence**
  - Caches, Processes, Virtual Memory, Memory Allocation



# 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 base 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 reference sheets
  - And you can just look it up as needed 😊
- ❖ 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**eppelins, **Y**o
    - <https://xkcd.com/992/>

# 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

# Review Questions

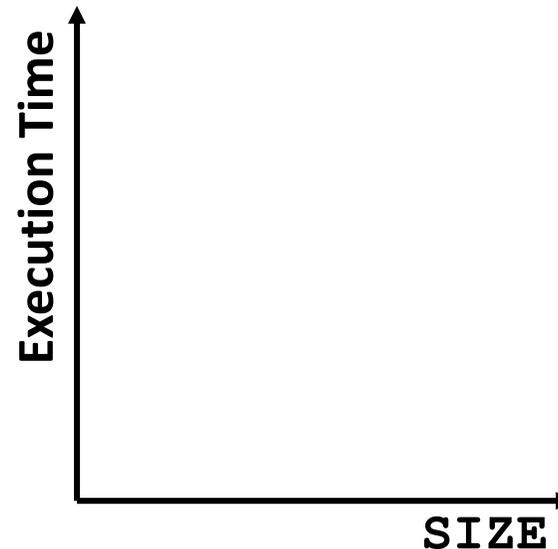
- ❖ Convert the following to or from IEC:
  - 512 Mi-students
  - $2^{33}$  cats
  
- ❖ Compute the average memory access time (AMAT) for the following system properties:
  - Hit time of 2 ns
  - Miss rate of 1%
  - Miss penalty of 300 ns

# How does execution time grow with SIZE?

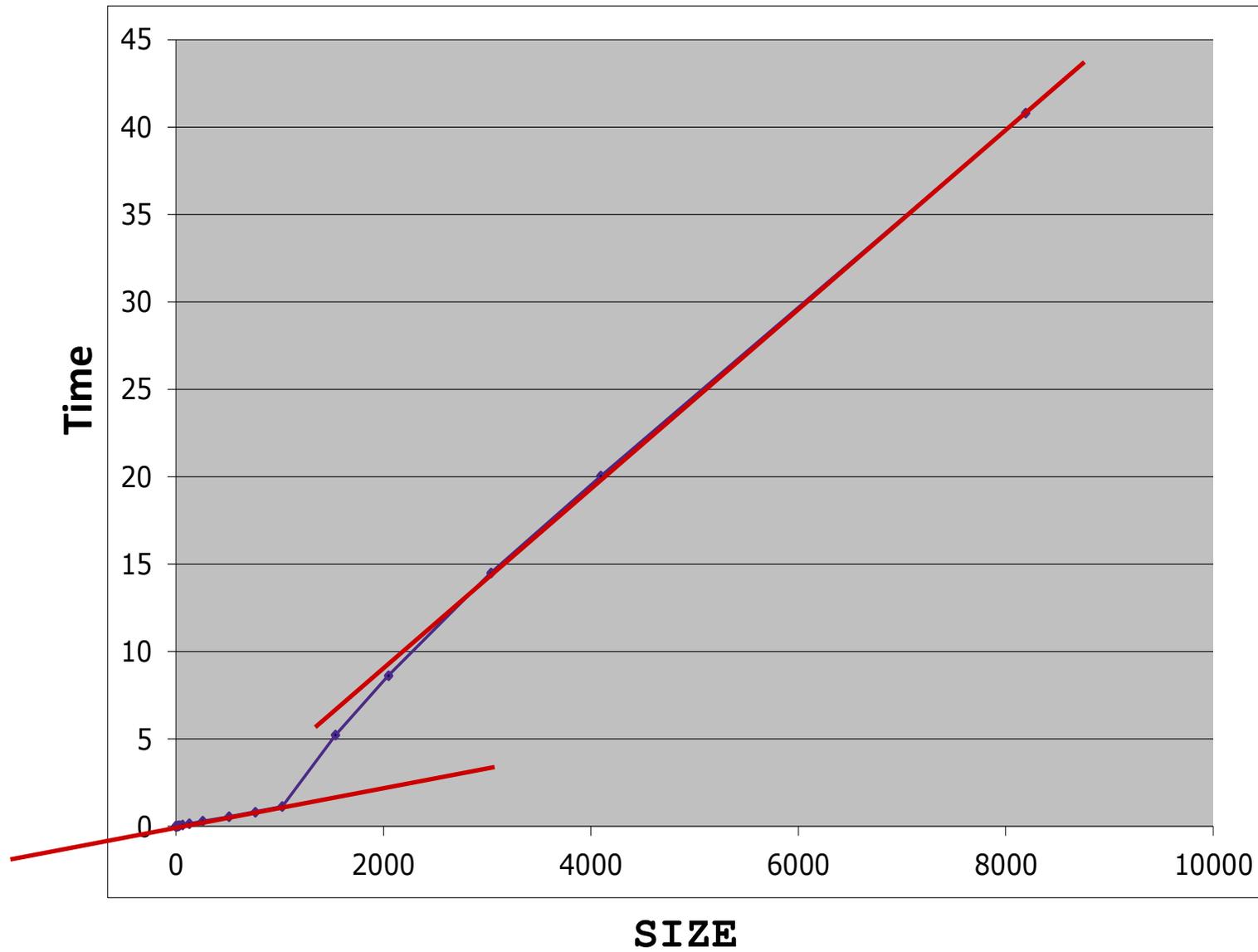
```
int array[SIZE];
int sum = 0;

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

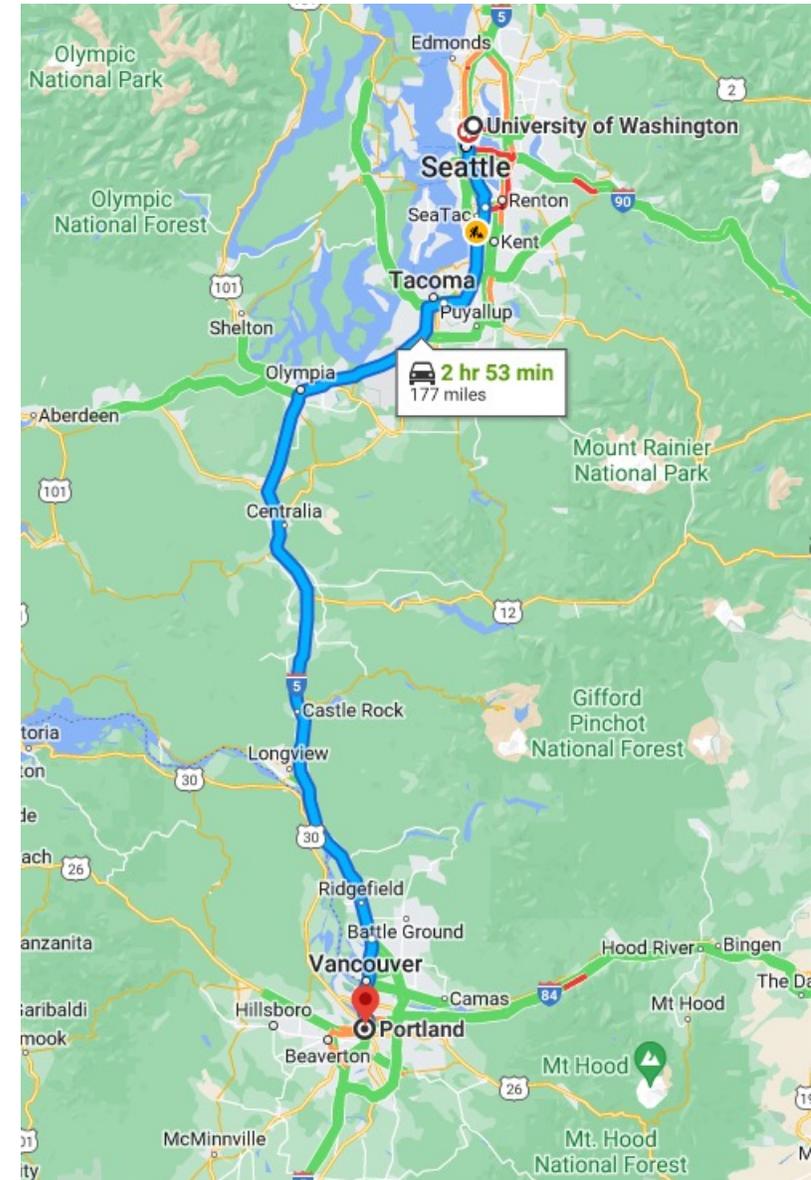
**Plot:**



# Actual Data



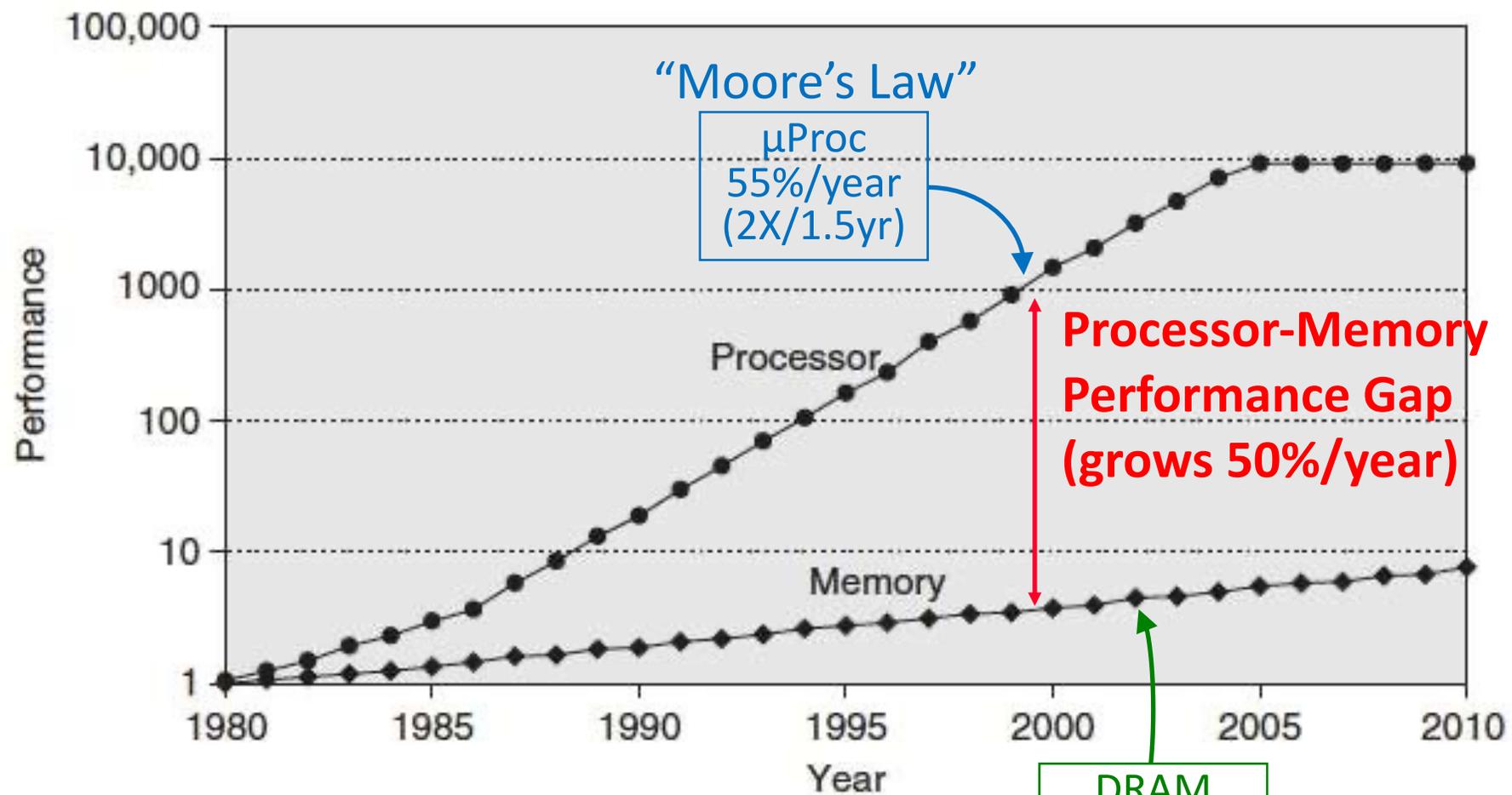
# A timely analogy...



# Making memory accesses fast!

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

# Processor-Memory Gap



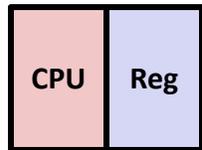
**1989** first Intel CPU with cache on chip

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

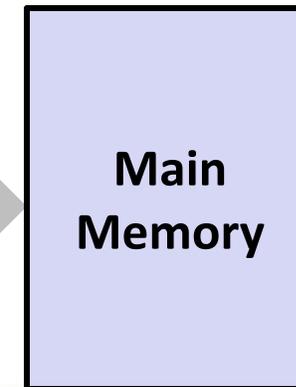
DRAM  
7%/year  
(2X/10yrs)

# 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



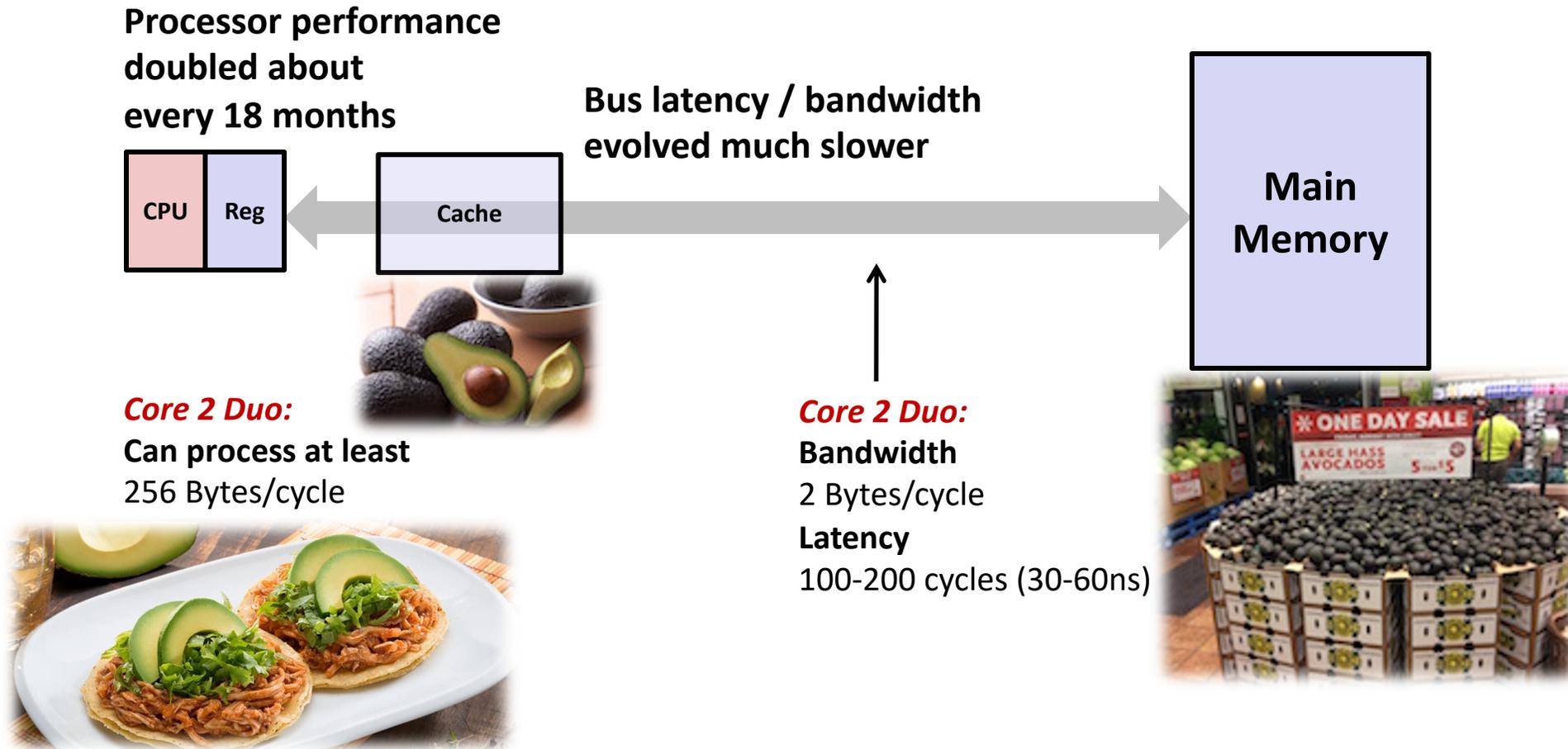
**Core 2 Duo:**  
Bandwidth 2 Bytes/cycle  
Latency 100-200 cycles (30-60ns)



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

**Problem: lots of waiting on memory...**

# Problem: Processor-Memory Bottleneck



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

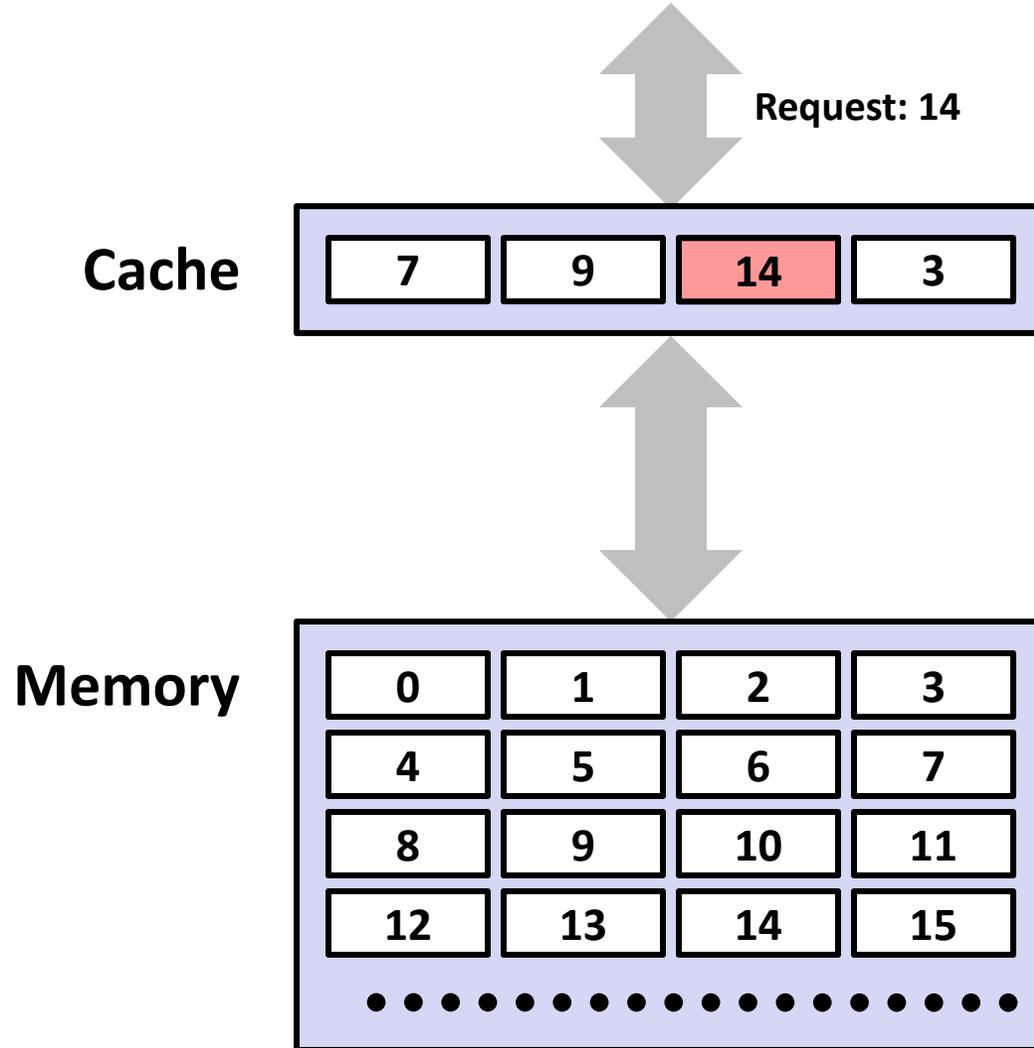
**Solution: caches**

# 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: Hit (Review)

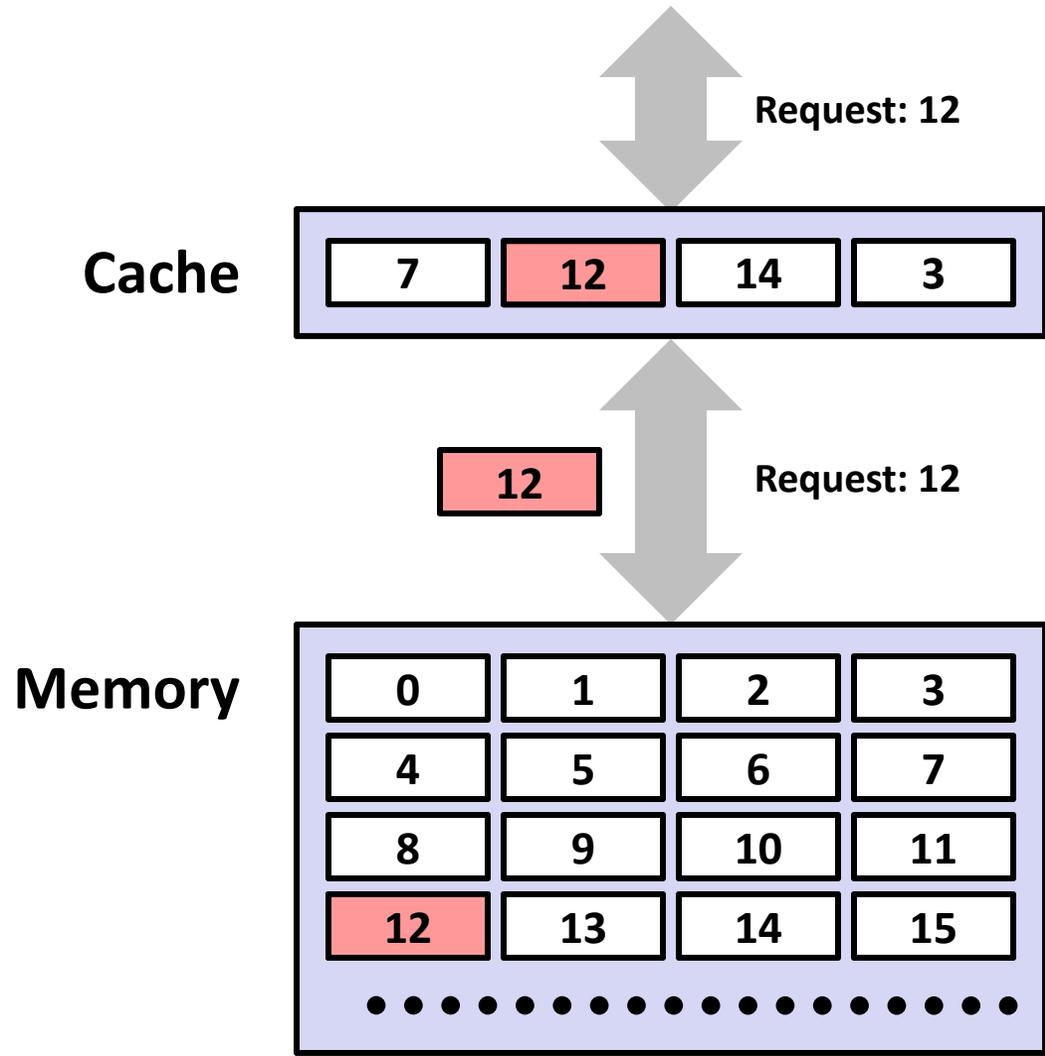


*Data in block b is needed*

*Block b is in cache:  
Hit!*

*Data is returned to CPU*

# 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.

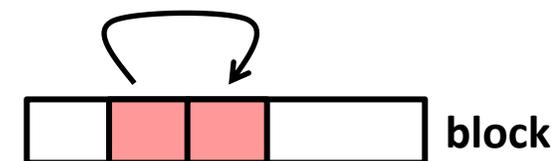
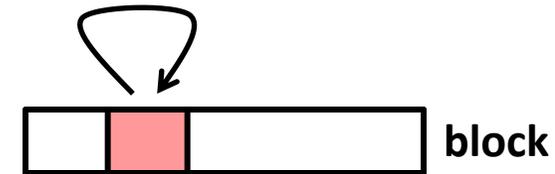
# Why Caches Work (Review)

- ❖ **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?

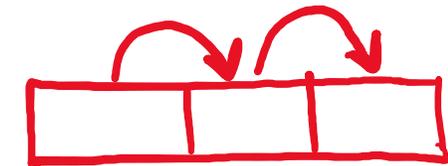


# Example: Any Locality?

```
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;
}
```

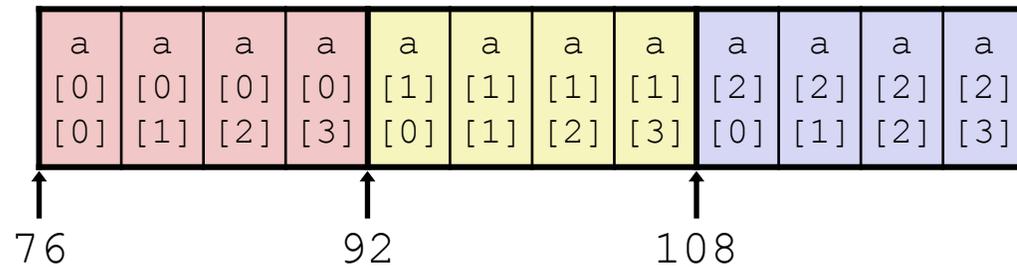
# Locality Example #1

```
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];

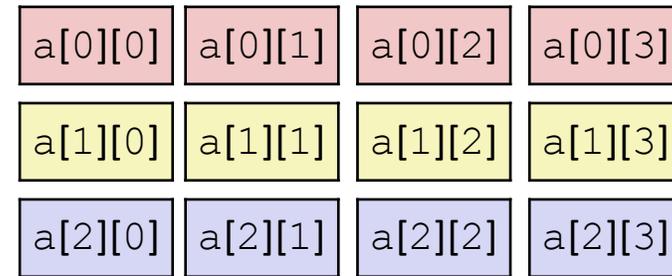
    return sum;
}
```

## Layout in Memory



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

**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]

# 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

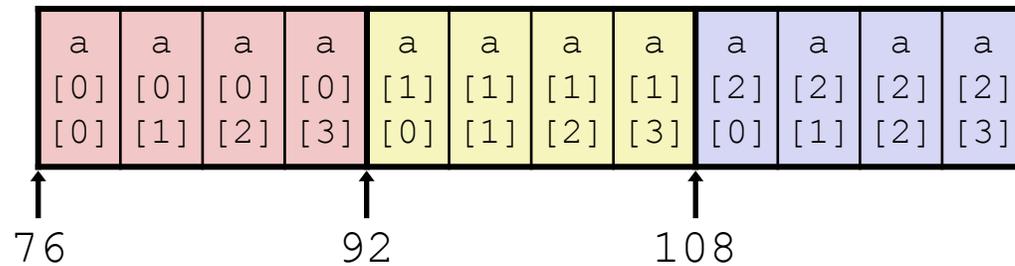
```

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

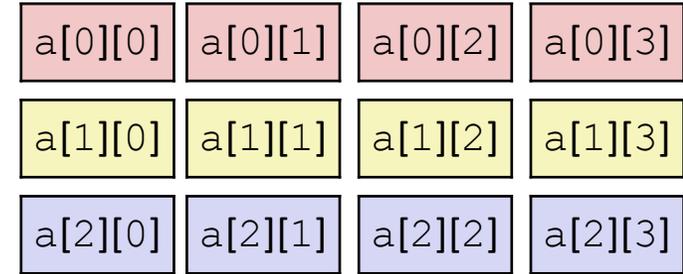
    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];

    return sum;
}
    
```

Layout in Memory



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]

# Locality Example #3

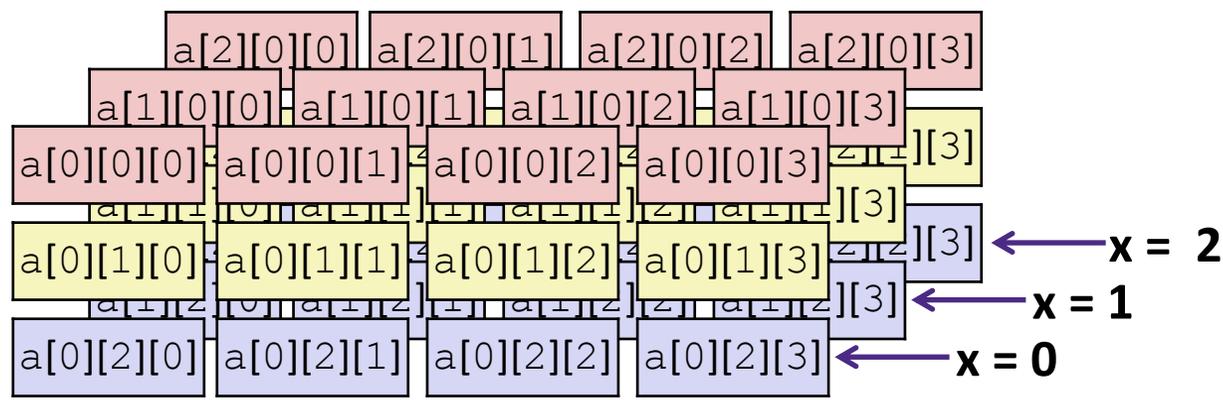
```

int sum_array_3D(int a[X][Y][Z])
{
    int i, j, k, sum = 0;

    for (i = 0; i < Y; i++)
        for (j = 0; j < Z; j++)
            for (k = 0; k < X; k++)
                sum += a[k][i][j];

    return sum;
}
    
```

- ❖ What is wrong with this code?
- ❖ How can it be fixed?
  - Want inner loop to be j, since those are only one int apart



# 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



# 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 =

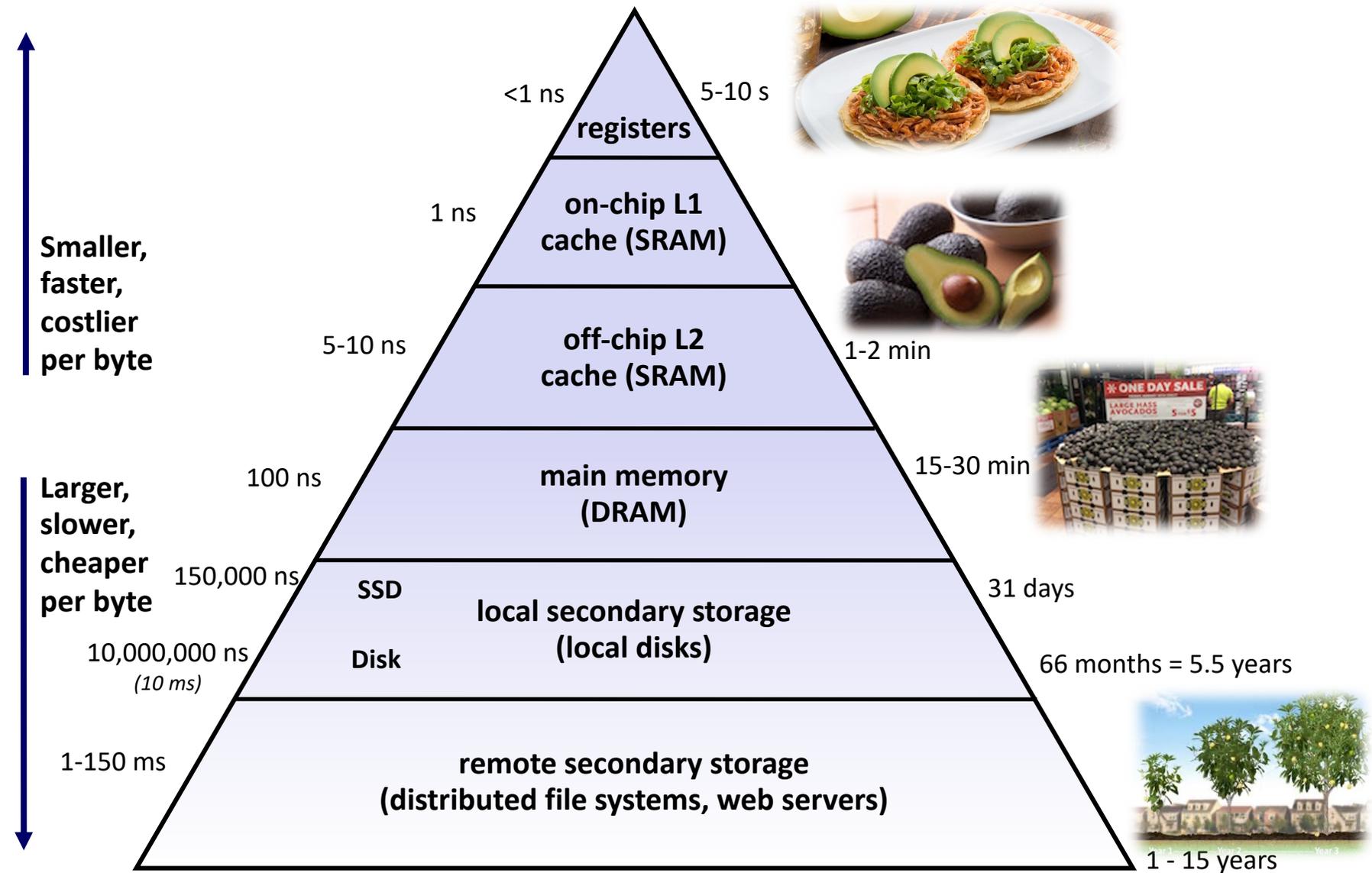
- ❖ Which improvement would be best?
  - A. 190 ps clock
  - B. Miss penalty of 40 clock cycles
  - C. MR of 0.015 misses/instruction

# Can we have more than one cache?

Sure!

- ❖ 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
- Makes use of *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