

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

# Caches I

CSE 351 Autumn 2017

**Instructor:**  
Justin Hsia

**Teaching Assistants:**  
Lucas Wotton  
Michael Zhang  
Parker DeWilde  
Ryan Wong  
Sam Gehman  
Sam Wolfson  
Savanna Yee  
Vinny Palaniappan

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

# Administrivia

- ❖ Homework 3 due tonight
- ❖ Lab 3 due next Thursday (11/9)
- ❖ Midterm grades released tomorrow
  - Regrade requests done on Gradescope and due Wednesday (11/8)
  - **Midterm Clobber Policy**
    - Final will be cumulative (half midterm, half post-midterm)
    - If you perform better on the midterm portion of the final, you replace your midterm score!
    - Replacement score =  $(F_{MT} \text{ score} - F_{MT} \text{ avg}) \times \frac{MT \text{ stdev}}{F_{MT} \text{ stdev}} + MT \text{ mean}$

2

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

# Roadmap

**C:**

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

**Java:**

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
  c.getMPG();
```

Memory & data  
Integers & floats  
x86 assembly  
Procedures & stacks  
Executables  
Arrays & structs  
**Memory & caches**  
Processes  
Virtual memory  
Memory allocation  
Java vs. C

**Assembly language:**

```
get_mpg:
  pushq  %rbp
  movq   %rsp, %rbp
  ...
  popq   %rbp
  ret
```

**Machine code:**

```
0111010000011000
100011010000010000000010
1000100111000010
11000001111101000011111
```

**OS:** Windows 10, OS X Yosemite, Linux

**Computer system:**

3

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

# Aside: Units and Prefixes

- ❖ 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

4

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

# 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 Giraffe Teaches Pet, Extinct **Z**ebra to Yodel
  - **K**irby Missed **G**anondorf Terribly, Potentially Exterminating **Z**elda and Yoshi
  - **xkcd**: **K**arl **M**arx **G**ave **T**he **P**roletariat **E**leven **Z**eppelins, **Y**o
    - <https://xkcd.com/992/>
  - Post your best on Piazza!

5

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

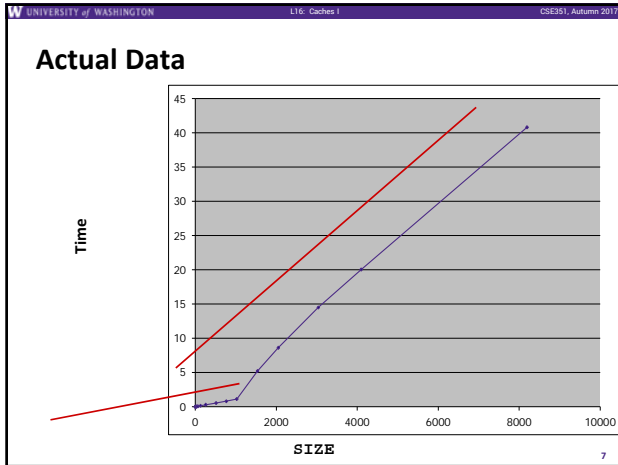
# How does execution time grow with SIZE?

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

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

**Plot**

6

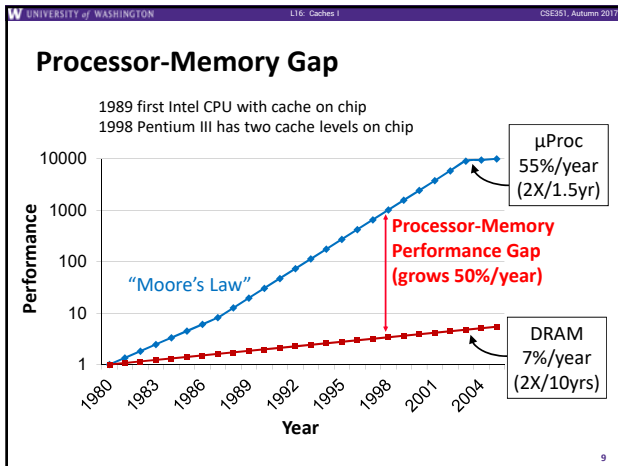


UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Making memory accesses fast!

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

8



UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

Bus latency / bandwidth evolved much slower

CPU Reg

Main Memory

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)

10

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

Bus latency / bandwidth evolved much slower

CPU Reg

Cache

Main Memory

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)

11

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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.)

12

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

## General Cache Mechanics

Cache: [ 7 | 9 | 14 | 3 ]

- Smaller, faster, more expensive memory.
- Caches a subset of the **blocks** (a.k.a. *lines*)

Data is copied in **block-sized** transfer units

Memory:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

- Larger, slower, cheaper memory.
- Viewed as partitioned into "**blocks**" or "*lines*"

13

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

## General Cache Concepts: Hit

Request: 14

Cache: [ 7 | 9 | 14 | 3 ]

Data in block *b* is needed

Block *b* is in cache: **Hit!**

Data is returned to CPU

Memory:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

14

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

## General Cache Concepts: Miss

Request: 12

Cache: [ 7 | 12 | 14 | 3 ]

Data in block *b* is needed

Block *b* is not in cache: **Miss!**

Block *b* is fetched from memory

Request: 12

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

Memory:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

15

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

## Why Caches Work

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

16

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

## Why Caches Work

- 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

17

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

## Why Caches Work

- 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?

18

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Example: Any Locality?

```

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

```

- ❖ **Data:**
  - **Temporal:** sum referenced in each iteration
  - **Spatial:** array a[] accessed in stride-1 pattern
- ❖ **Instructions:**
  - **Temporal:** cycle through loop repeatedly
  - **Spatial:** reference instructions in sequence

19

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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;
}

```

20

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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**

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

**Access Pattern:**

- 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]

stride = ?

**Layout in Memory**

a	a	a	a	a	a	a	a	a	a	a	a
[0]	[0]	[0]	[0]	[1]	[1]	[1]	[1]	[2]	[2]	[2]	[2]
[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]

76                      92                      108

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

21

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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;
}

```

22

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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**

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

**Access Pattern:**

- 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]

stride = ?

**Layout in Memory**

a	a	a	a	a	a	a	a	a	a	a	a
[0]	[0]	[0]	[0]	[1]	[1]	[1]	[1]	[2]	[2]	[2]	[2]
[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]

76                      92                      108

23

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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**

a[2][0][0]	a[2][0][1]	a[2][0][2]	a[2][0][3]
a[1][0][0]	a[1][0][1]	a[1][0][2]	a[1][0][3]
a[0][0][0]	a[0][0][1]	a[0][0][2]	a[0][0][3]
a[2][1][0]	a[2][1][1]	a[2][1][2]	a[2][1][3]
a[1][1][0]	a[1][1][1]	a[1][1][2]	a[1][1][3]
a[0][1][0]	a[0][1][1]	a[0][1][2]	a[0][1][3]
a[2][2][0]	a[2][2][1]	a[2][2][2]	a[2][2][3]
a[1][2][0]	a[1][2][1]	a[1][2][2]	a[1][2][3]
a[0][2][0]	a[0][2][1]	a[0][2][2]	a[0][2][3]

m = 2  
m = 1  
m = 0

24

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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 = 7, N = 3, L = 4)

25

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Cache Performance Metrics

- ❖ 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 - 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

26

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Cache Performance

- ❖ 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
  - $AMAT = Hit\ time + Miss\ rate \times Miss\ penalty$   
(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 =

27

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Peer Instruction 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?
  - Vote at <http://PollEv.com/justinh>
  - A. 190 ps clock
  - B. Miss penalty of 40 clock cycles
  - C. MR of 0.015 misses/instruction

28

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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!

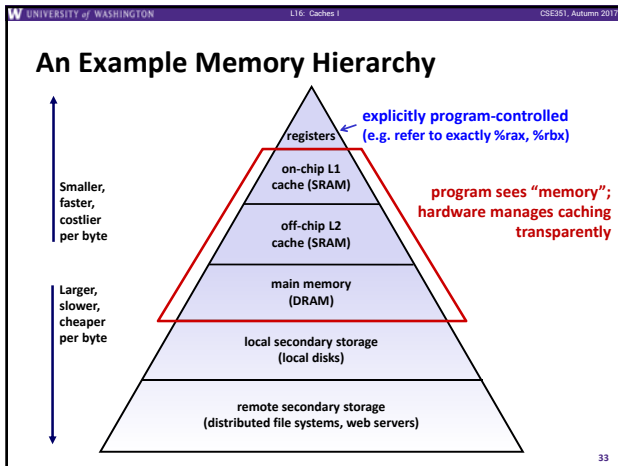
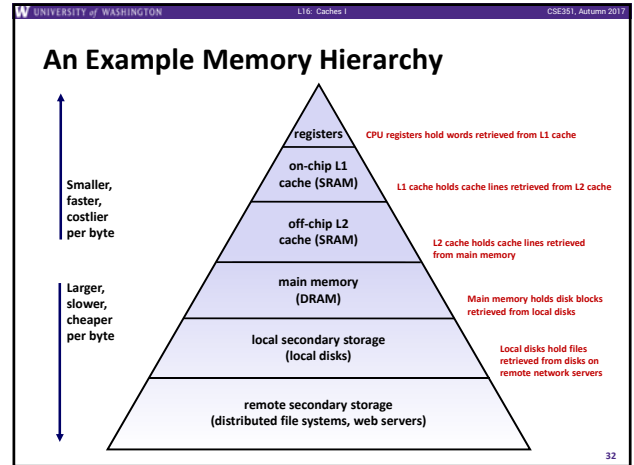
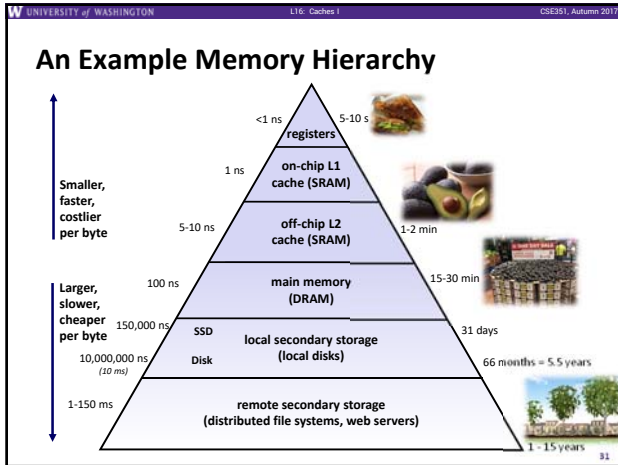
29

UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Memory Hierarchies

- ❖ Some fundamental and enduring properties of hardware and software systems:
  - Faster storage technologies almost always cost more per byte and have lower capacity
  - The gaps between memory technology speeds are widening
    - True for: registers ↔ cache, cache ↔ DRAM, DRAM ↔ disk, etc.
  - Well-written programs tend to exhibit good locality
- ❖ These properties complement each other beautifully
  - They suggest an approach for organizing memory and storage systems known as a memory hierarchy

30

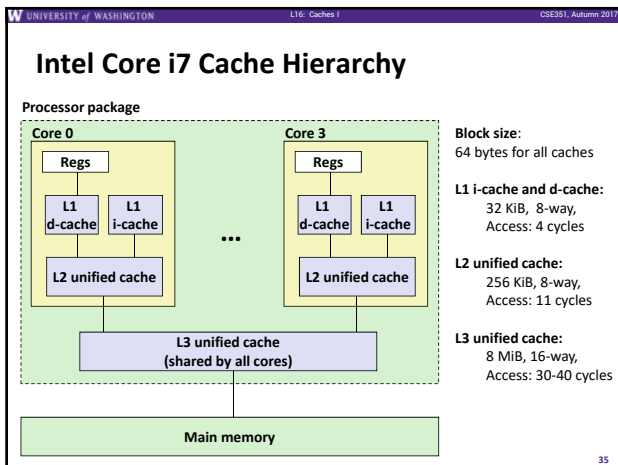


UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### Memory Hierarchies

- ❖ Fundamental idea of a memory hierarchy:
  - For each level  $k$ , the faster, smaller device at level  $k$  serves as a cache for the larger, slower device at level  $k+1$
- ❖ Why do memory hierarchies work?
  - Because of locality, programs tend to access the data at level  $k$  more often than they access the data at level  $k+1$
  - Thus, the storage at level  $k+1$  can be slower, and thus larger and cheaper per bit
- ❖ **Big Idea:** The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top

34



UNIVERSITY of WASHINGTON L16: Caches I CSE351, Autumn 2017

### 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

36