The Hardware/Software Interface

Memory & Caches I

Instructors:

Justin Hsia, Amber Hu

Teaching Assistants:

Anthony Mangus
Grace Zhou
Jiuyang Lyu
Kurt Gu
Mendel Carroll
Naama Amiel
Rose Maresh
Violet Monserate
Divya Ramu
Jessie Sun
Kanishka Singh
Liander Rainbolt
Ming Yan
Pollux Chen
Soham Bhosale

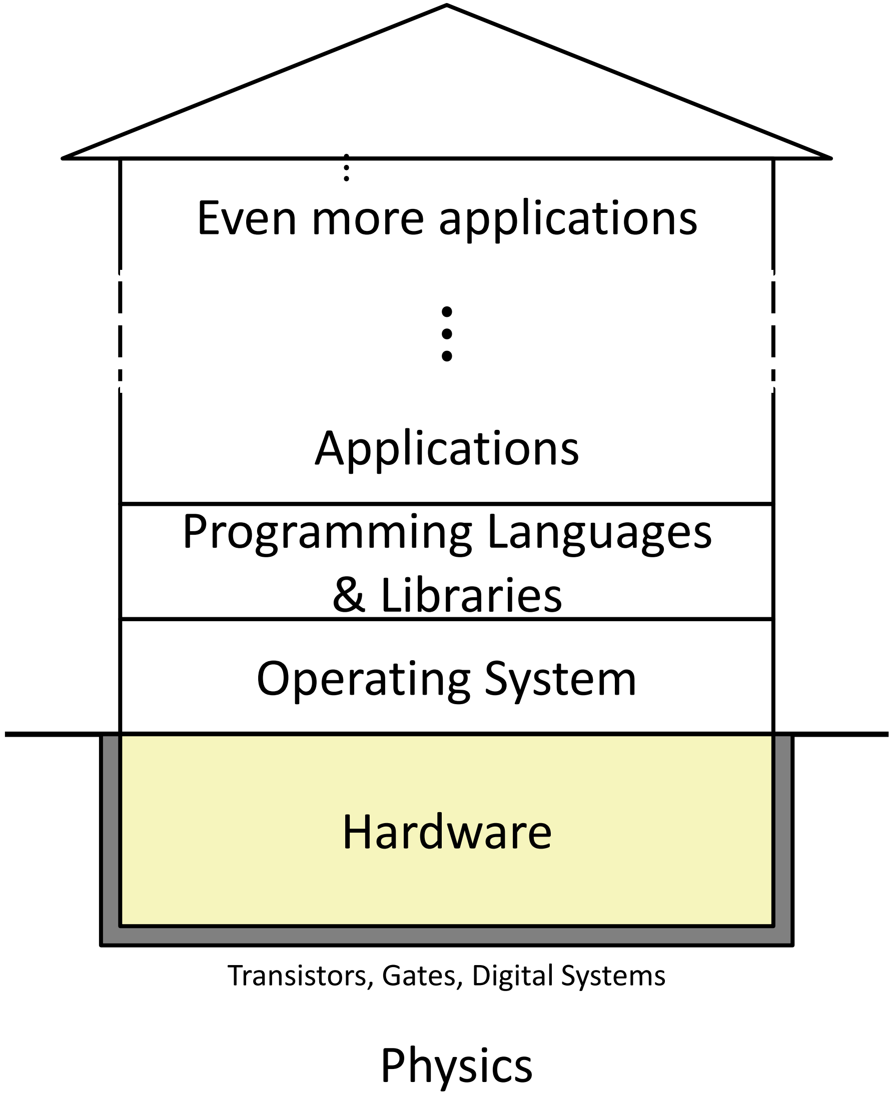
Relevant Course Information

- Mid-quarter Survey due tonight
- HW14 due tonight, HW15 due Monday, HW16 due Wednesday
- Lab 3 due next Friday (11/7)
  - Lots of resources: HW15, Section 06, GDB Stack Tutorial lesson
- Midterm grading is ongoing, will be released when available
  - Midterm clobber policy allows you to overwrite your midterm score!
  - Solutions file will be posted, Gradescope rubric items will have more scoring details

House of Computing Check-In

- Topic Group 3: **Scale & Coherence**
  - **Caches**, Memory Allocation, Processes, Virtual Memory

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



Lecture Outline (1/4)

- **IEC Prefixes**
- Caches and Cache Mechanics
- Cache Performance Metrics
- The Memory Hierarchy

Prefixes (Review)

- Here focusing on large numbers (exponents > 0)
- SI prefixes are *ambiguous* if base 10 or 2
  - 1000 ≈ 1024
  - Note that $10^3 \approx 2^{10}$
- IEC prefixes are *unambiguously* base 2

my hard drive:
advertised: 512 GB
$= 512 \times 10^9$ B
actually: $\frac{512 \times 10^9}{2^{30}} \approx 477$ GiB

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

Large Powers of 2 and Units (Review)

- Because IEC prefixes are powers of 2^{10}, we can convert any large power of 2 as follows:
  - Note that we are only changing the *quantity* and the *units* remain the same

$$2^{XY}\text{ “things”} = \left\{\begin{array}{l} Y=0 \to 1 \\ Y=1 \to 2 \\ Y=2 \to 4 \\ Y=3 \to 8 \\ Y=4 \to 16 \\ Y=5 \to 32 \\ Y=6 \to 64 \\ Y=7 \to 128 \\ Y=8 \to 256 \\ Y=9 \to 512 \end{array}\right\} + \left\{\begin{array}{l} X=0 \to \\ X=1 \to \text{Kibi-} \\ X=2 \to \text{Mebi-} \\ X=3 \to \text{Gibi-} \\ X=4 \to \text{Tebi-} \\ X=5 \to \text{Pebi-} \\ X=6 \to \text{Exbi-} \\ X=7 \to \text{Zebi-} \\ X=8 \to \text{Yobi-} \end{array}\right\} + \text{ “things”}$$

- Examples:
  - 2^{32} bits into IEC: $2^2 \to 4$, $2^{30} \to$ Gibi $\boxed{\text{4 Gibi bits}}$
  - How many address bits to use 13.2 TiB of memory? 16 TiB $\to$ 2^{44} bytes $\to$ $\boxed{\text{44 address bits}}$
    >8, <16

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**xtinct **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/
  - Post your best on Ed Discussion!

Polling Questions (1/2)

- Convert the following to or from IEC:
  - 512 Ki-books $= \boxed{2^{19} \text{ books}}$
    - $2^9 \times 2^{10}$
  - 2^{27} caches $= \boxed{128 \text{ Mi-caches}}$
    - $2^7 \times 2^{20}$

Lecture Outline (2/4)

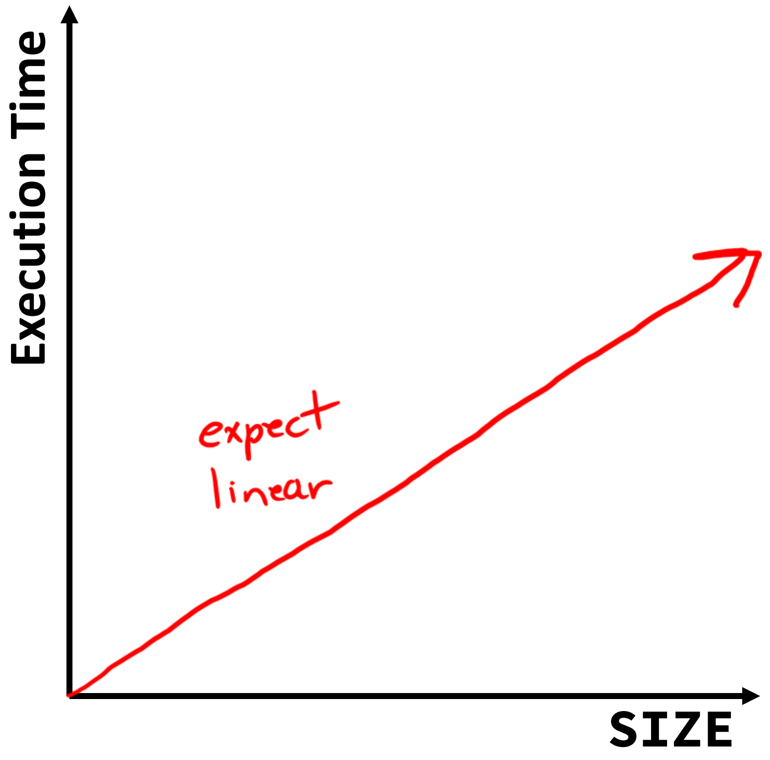
- IEC Prefixes
- **Caches and Cache Mechanics**
- Cache Performance Metrics
- The Memory Hierarchy

How does execution time grow with SIZE?

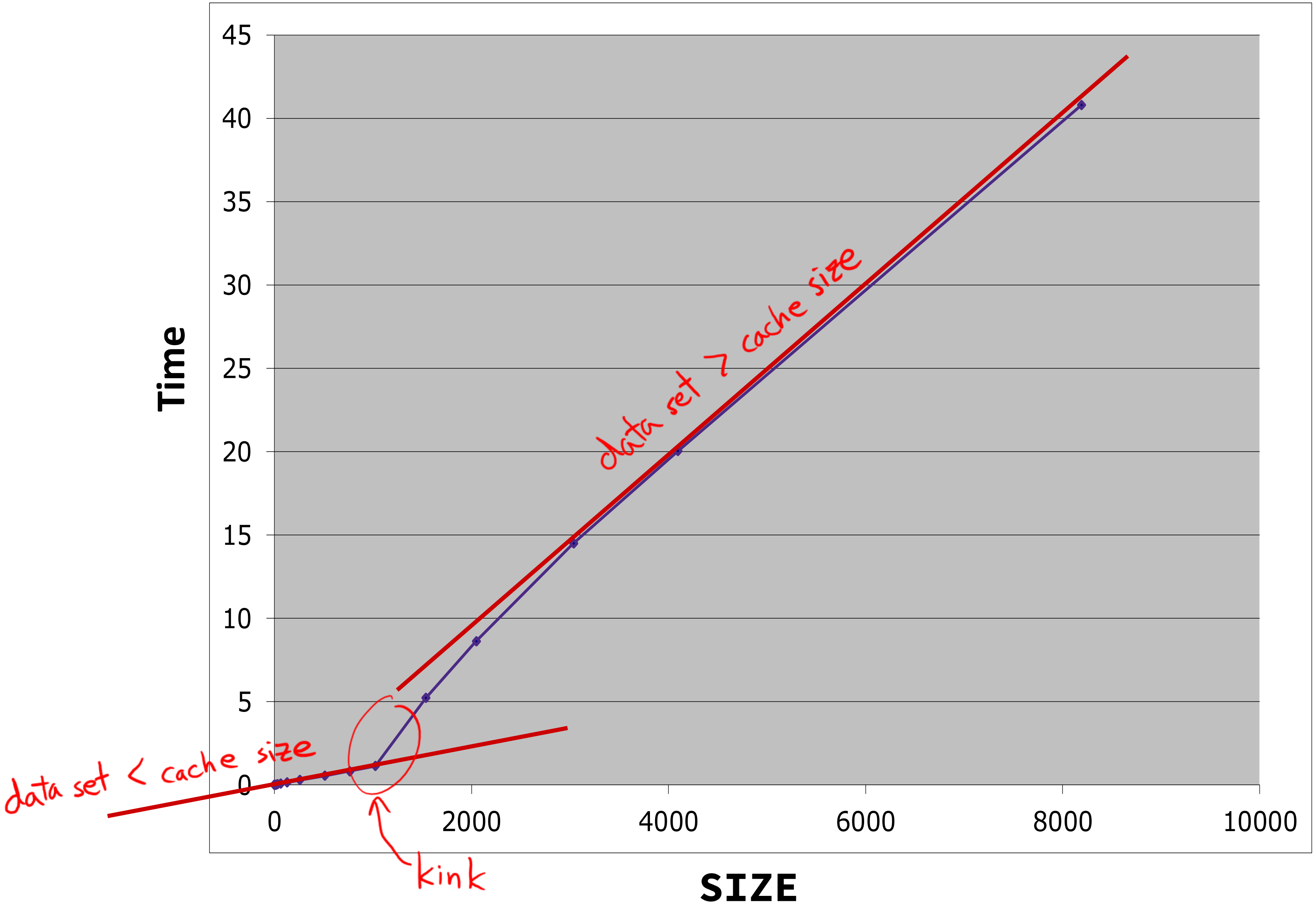
```
int array[SIZE];
int sum = 0;

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

repeat 200,000 times

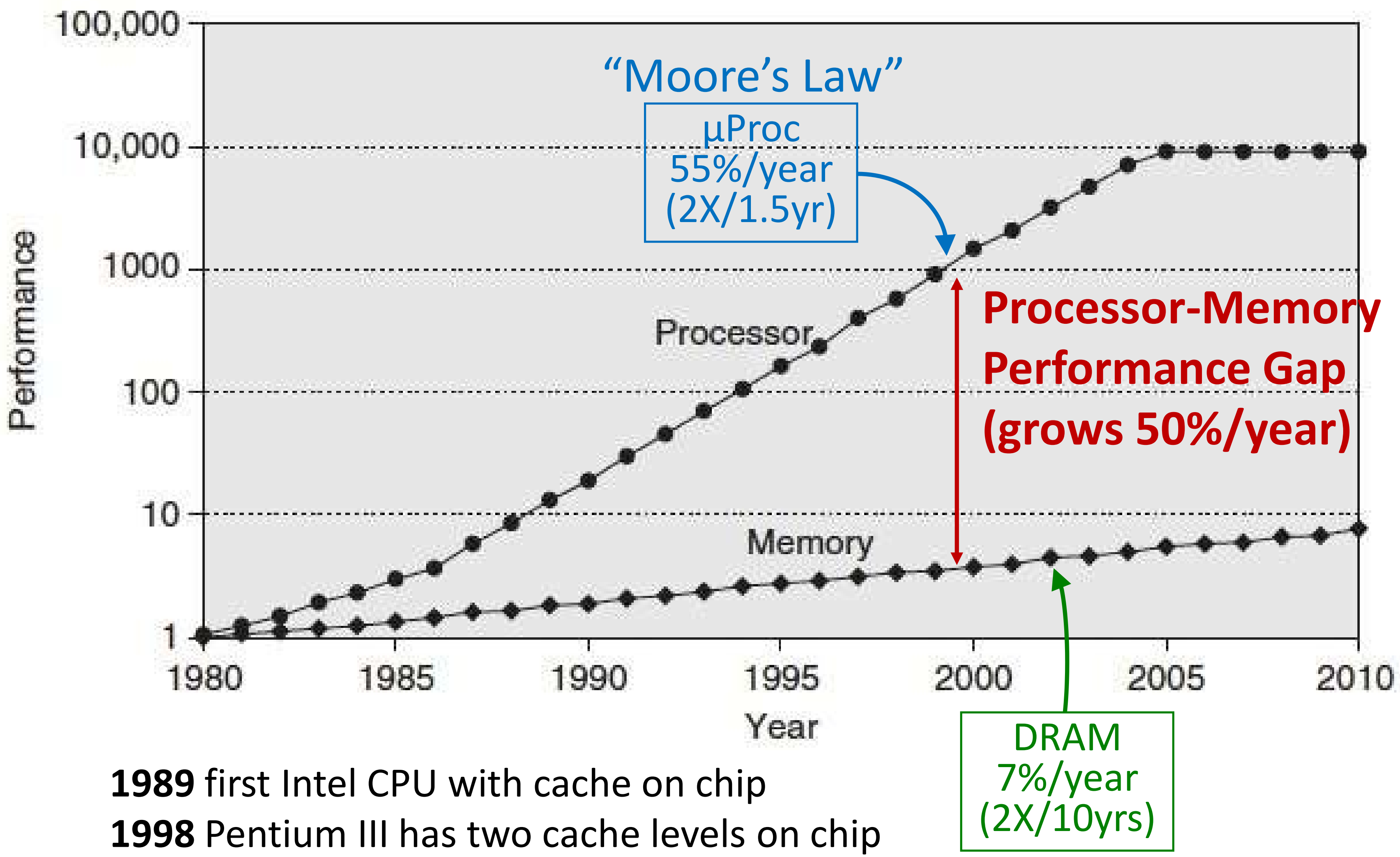
← execute SIZE×200,000 times



Actual Cache Timing Data



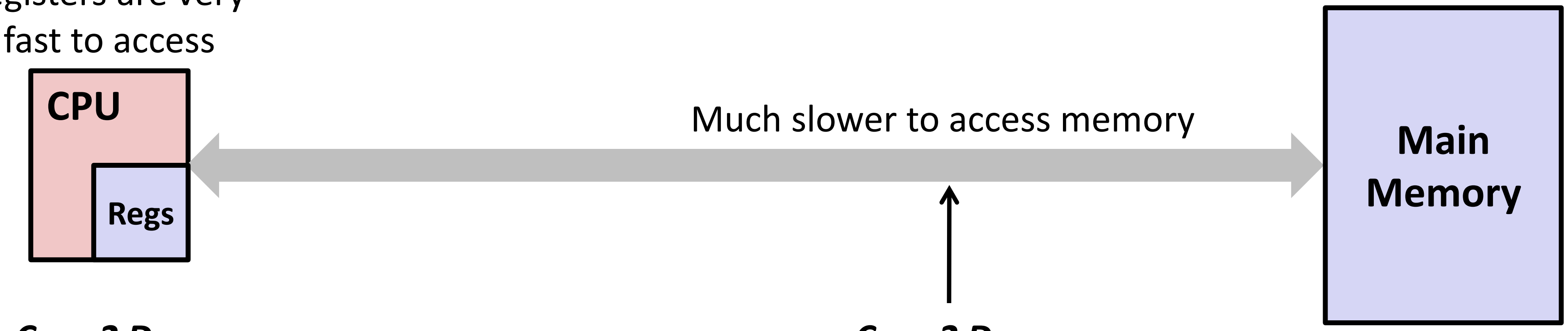
Processor-Memory Gap



1989 first Intel CPU with cache on chip
1998 Pentium III has two cache levels on chip

Processor-Memory Bottleneck

Registers are very fast to access



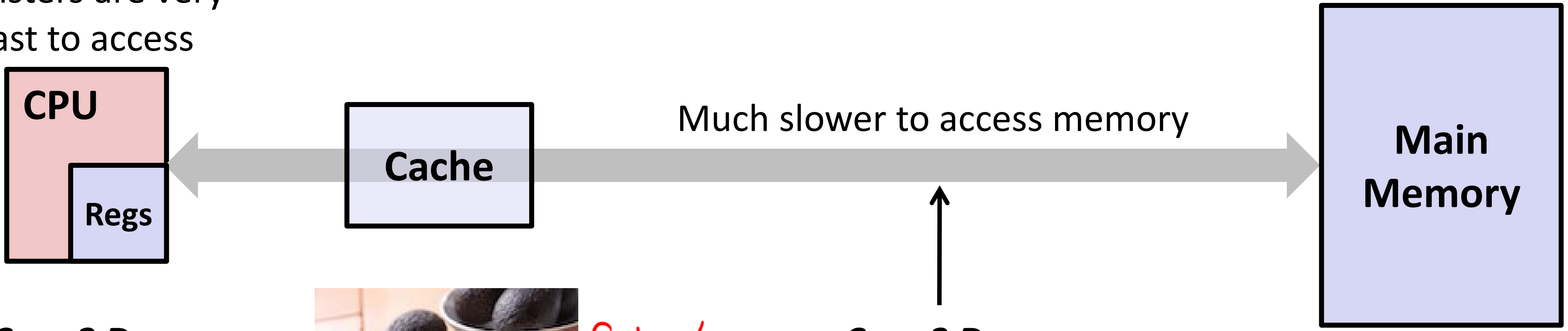
Core 2 Duo:
Can process at least
256 Bytes/cycle

Core 2 Duo:
Bandwidth: 2 Bytes/cycle
Latency: 100-200 cycles !

Problem: *lots of waiting on memory*

Processor-Memory Bottleneck Fix

Registers are very fast to access



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

fridge/pantry

Core 2 Duo:
Bandwidth: 2 Bytes/cycle
Latency: 100-200 cycles

sandwich to mouth

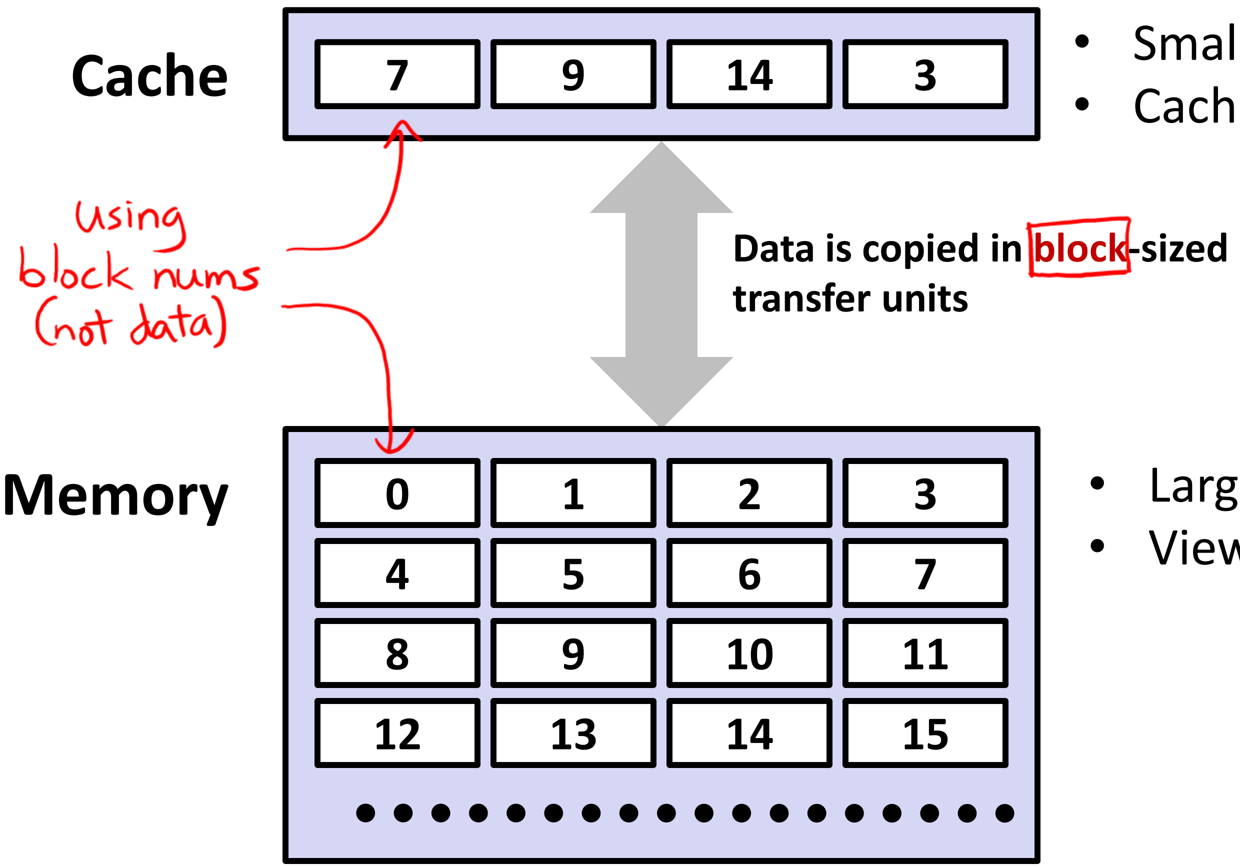
Problem: *lots of waiting on memory*
Solution: *caches*

grocery store

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.)

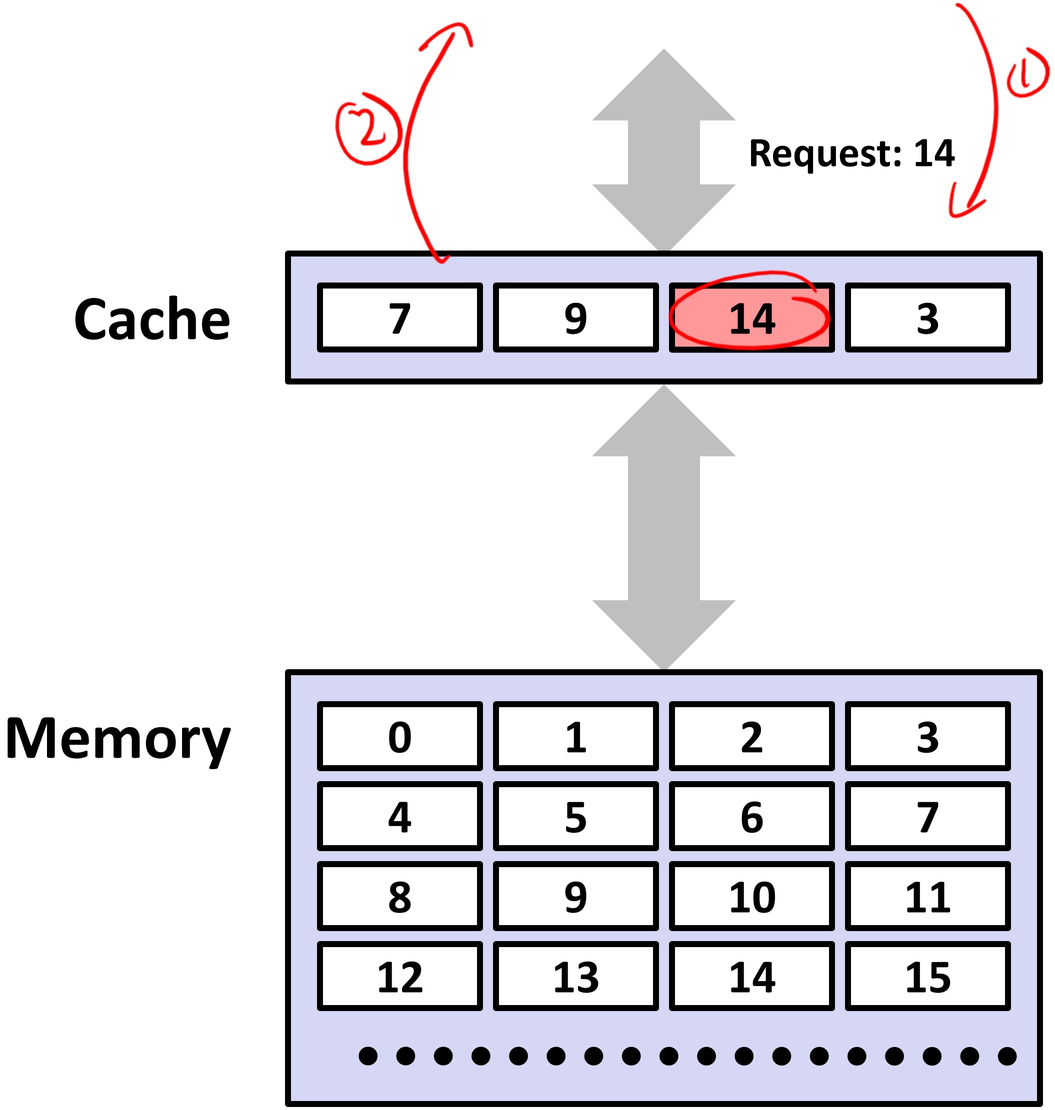
General Cache Mechanics (Review)



- Smaller, faster, more expensive memory
- Caches a subset of the blocks

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

General Cache Concepts: Hit (Review)

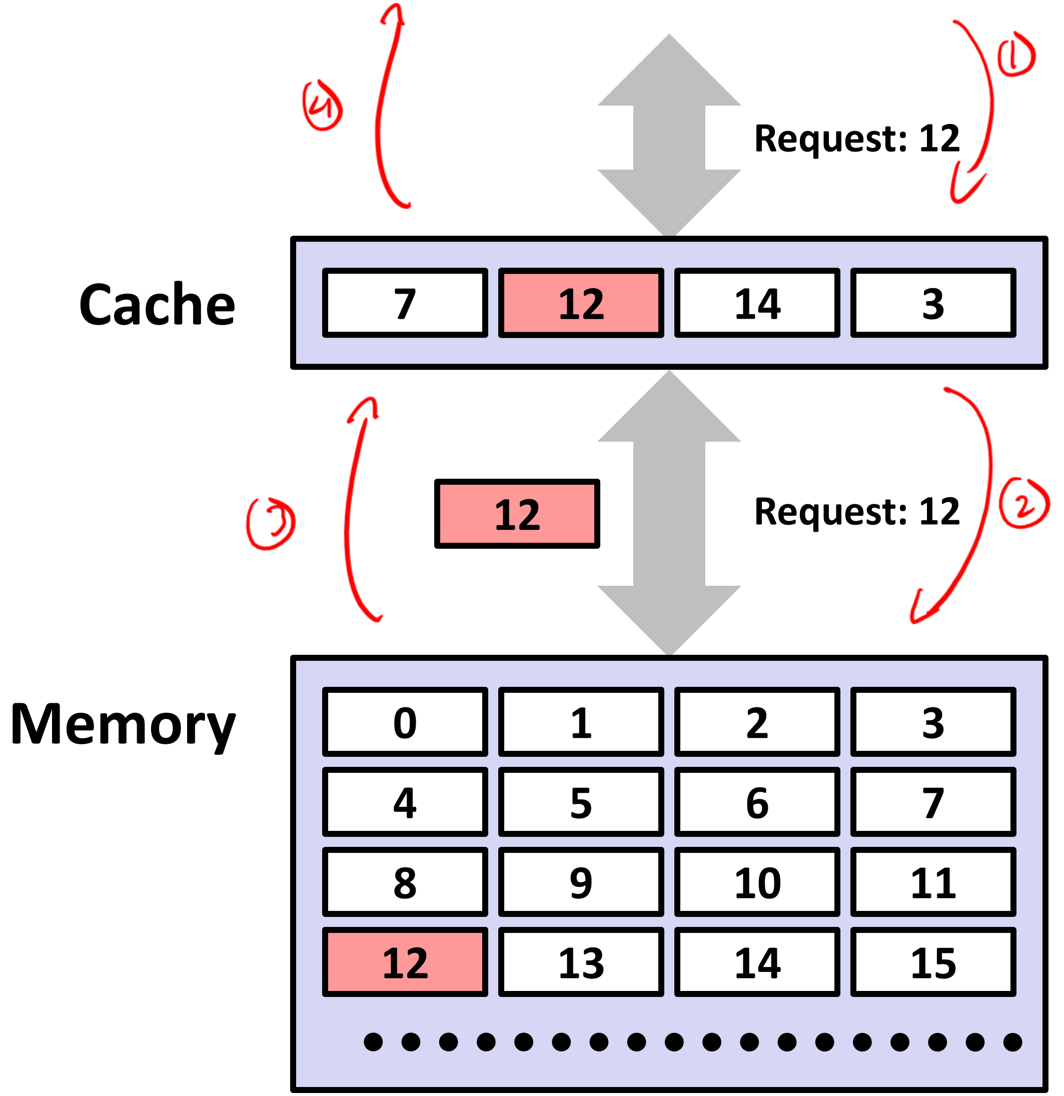


① *Data in block b is needed*

Block b is in cache:
Hit!

② *Data is returned to CPU*

General Cache Concepts: Miss (Review)



1. ***Data in block b is needed***

Block b is not in cache: Miss!

2. ***Block b is fetched from memory***

3. ***Block b is stored in cache***
   - Placement policy: Where should b go if there is room in the cache
   - Replacement policy: Which block should b replace if the ache is full

4. ***Data is returned to CPU***

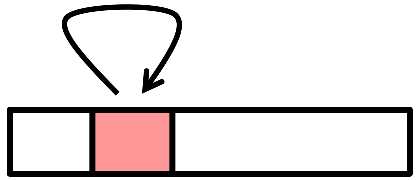
Why Caches Work: Locality (Review)

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

Why Caches Work: Temporal Locality (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

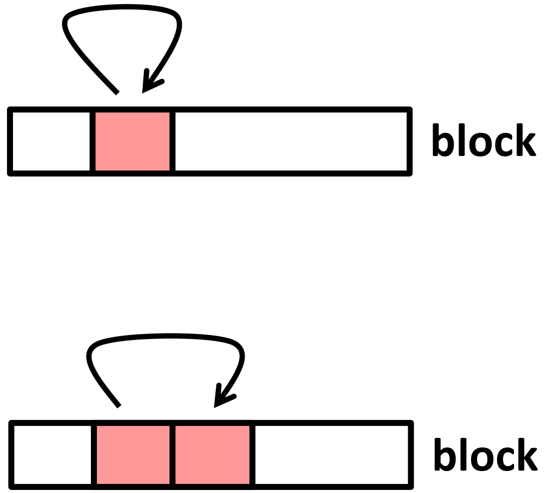


Why Caches Work: Spatial Locality (Review)

- 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?

Locality in Practice

```
sum = 0;
for (i = 0; i < n; i++) {
  sum += a[i];  a[0]
}               a[1]
return sum;     a[2]
```

Loop:

je Loop

- **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 Code

```
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;
}
```

rows cols

Locality Example #1 Access Pattern

```
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;
}
```

a[0][0]
0 1
0 2

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:
stride = ?

"stride-1"
1 int = 4B

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 (arbitrary starting address)

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]

76 92 108

Locality Example #2 Access Pattern

```
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;
}
```

0 1 2 ⋮   0 0 0

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:
stride = ?

stride-4
stride-N

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 (arbitrary starting address)

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]
76				92				108			

Lecture Outline (3/4)

- IEC Prefixes
- Caches and Cache Mechanics
- **Cache Performance Metrics**
- The Memory Hierarchy

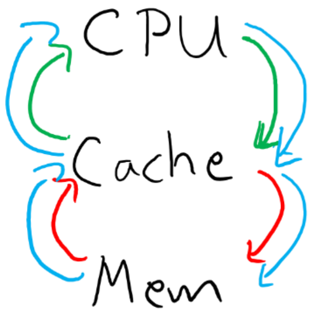
Cache Performance Parameters (Review)

- 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*)

- 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

Hit takes HT

Miss takes HT + MP



- Miss Penalty (MP)
  - Additional time required because of a miss
- Miss Rate (MR)
  - Fraction of memory references not found in cache (misses / accesses) = 1 - Hit Rate

Cache Performance Measurement (Review)

- *Average Memory Access Time* (AMAT): average time to access memory considering both hits and misses

AMAT = Hit time + Miss rate × Miss penalty = HT + MR × MP

  - HT and MP generally fixed, so minimize MR

HT*HR + MT*MR

HT*(1-MR)+(HT+MP)*MR

HT-~~HT*MR~~+~~HT*MR~~+MP*MR

- Example: Assume HT of 1 clock cycle and MP of 100 clock cycles
  - 97%: AMAT = 1+ 0.03*100 = 4 clock cycles
  - 99%: AMAT = 1 +0.01*100 = 2 clock cycles
  - 99% hit rate *twice* as good as 97% hit rate!!?!

Polling Questions (2/2)

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

  AMAT = HT+MR*MP = 1+0.02*50 = 2 clock cycles = 400 ps

- Which improvement would be best?

  A. 190 ps clock (overclocking, faster CPU)

  2 clock cyles = 380ps

  B. Miss penalty of 40 clock cycles (reduced Mem size)

  1+0.02*40 = 1.8 clock cycles = 360ps

  C. MR of 0.015 misses/instruction (write better code)

  1+0.015*50 = 1.75 clock cycles = 350ps

Lecture Outline (4/4)

- IEC Prefixes
- Caches and Cache Mechanics
- Cache Performance Metrics
- **The Memory Hierarchy**

More than one cache? (Review)

- Why would we want to do that?
  - Avoid going to memory!

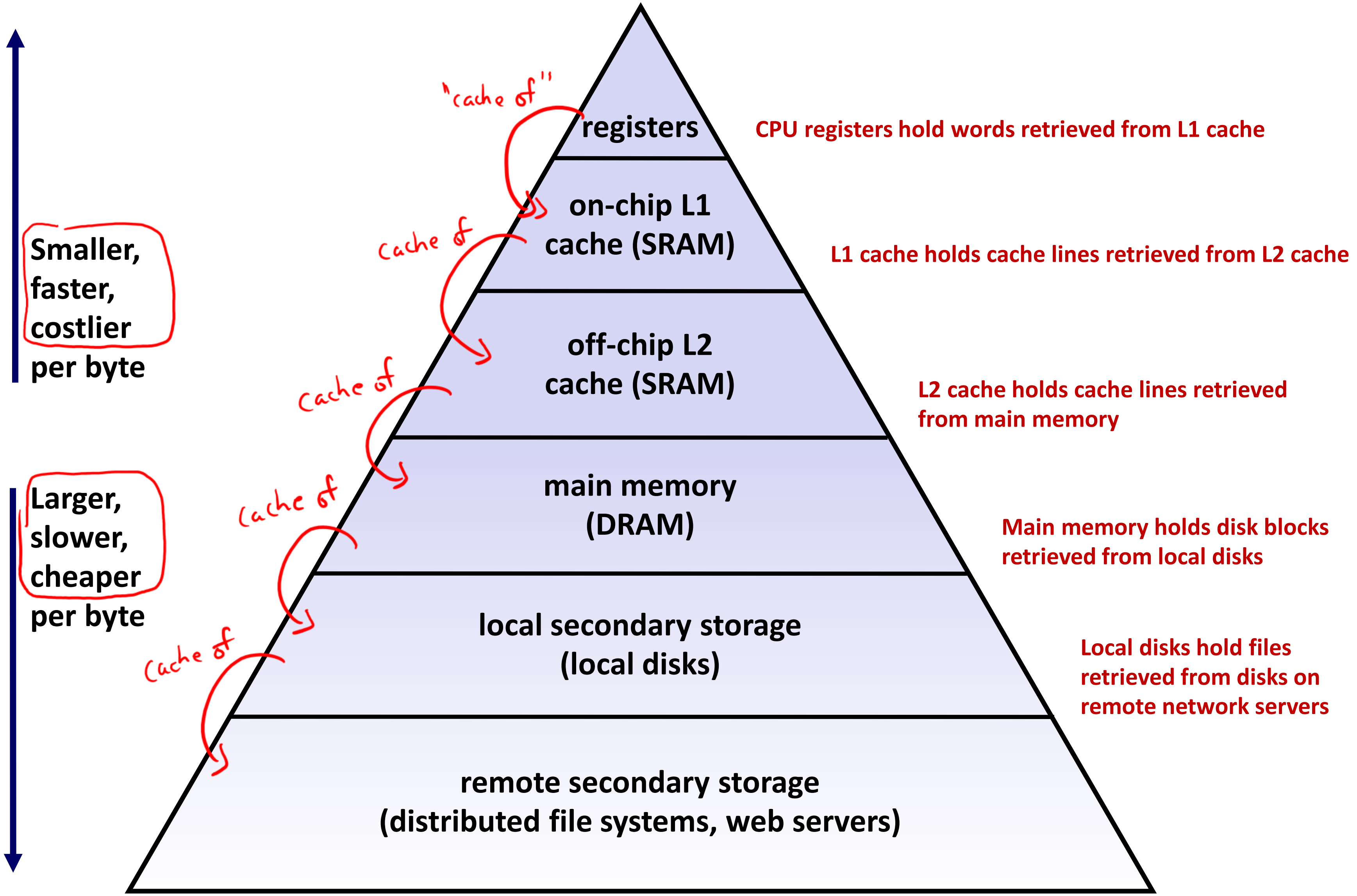
- Extra considerations for multilevel caching:
  - Block size may differ between different levels of cache
  - A memory access can miss in one level and then hit in the next, causing the block to be copied into the higher level
  - AMAT, however, is computed for your overall system caching, taking all levels of cache into account

Memory Hierarchies (Review)

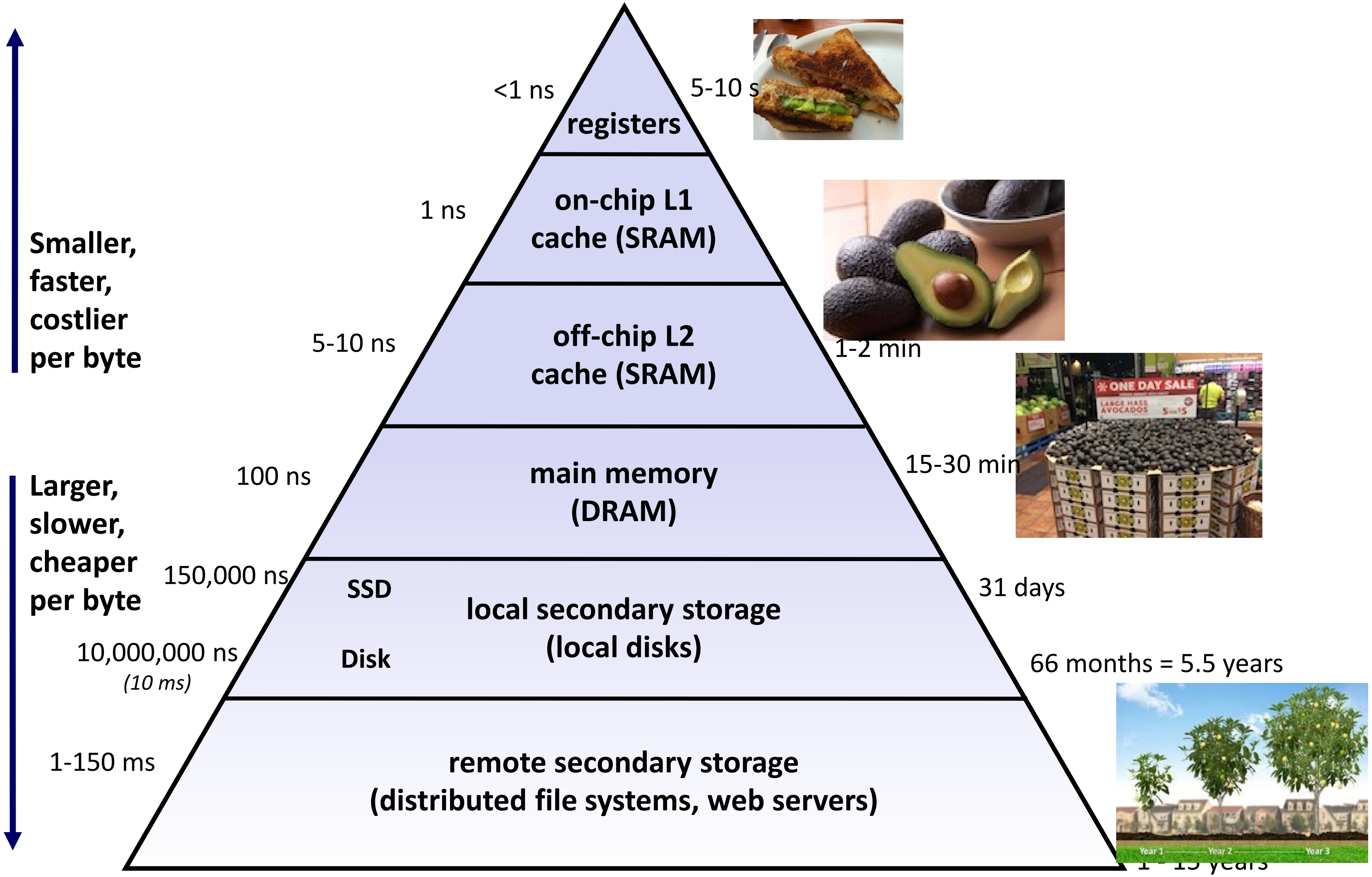
- Some fundamental properties of hardware and software systems:
  - The gaps between memory technology speeds are widening
    - True for: registers ↔ cache, cache ↔ main memory, main memory ↔ disk, etc.
  - Faster storage technologies almost always cost more per byte and have lower capacity
  - Well-written programs tend to exhibit good locality

- These properties complement each other beautifully and suggest an approach for organizing memory and storage systems known as 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

An Example Memory Hierarchy (Review)



An Example Memory Hierarchy: Analogy Revisited



Summary (1/3)

- IEC prefixes are unambiguously powers of 2:

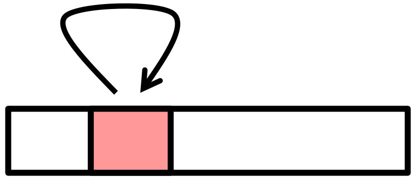
SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

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

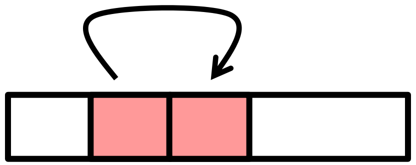
2^{XY} "things" = [Y = 0 → 1, Y = 1 → 2, Y = 2 → 4, Y = 3 → 8, Y = 4 → 16, Y = 5 → 32, Y = 6 → 64, Y = 7 → 128, Y = 8 → 256, Y = 9 → 512] + [X = 0 →, X = 1 → Kibi-, X = 2 → Mebi-, X = 3 → Gibi-, X = 4 → Tebi-, X = 5 → Pebi-, X = 6 → Exbi-, X = 7 → Zebi-, X = 8 → Yobi-] + "things"

Summary (2/3)

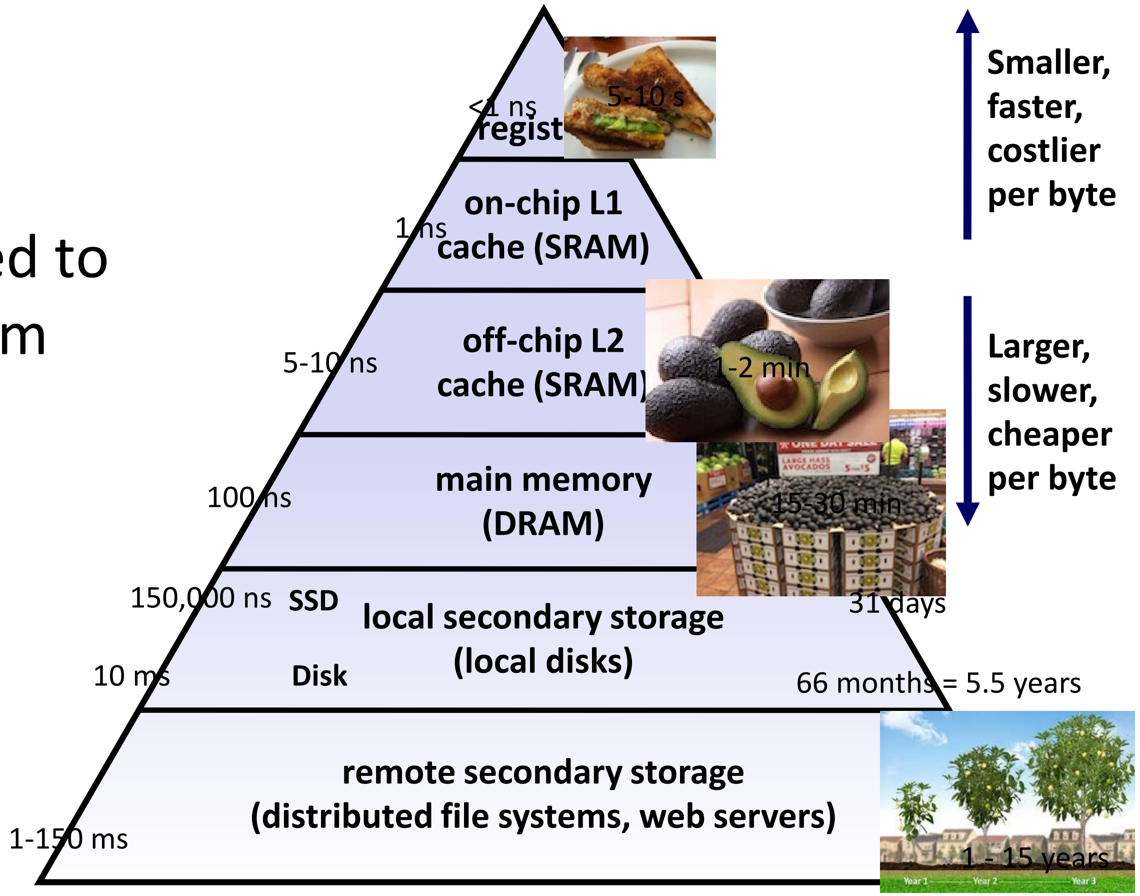
- Memory Hierarchy
  - Successively higher levels contain "most used" data from lower levels
  - Caches are intermediate storage levels used to optimize data transfers between any system elements with different characteristics
  - Exploits *temporal and spatial locality*:



block



block



Summary (3/3)

- Cache Performance
  - Ideal case: found in cache (***cache hit***), return requested data immediately
  - Bad case: not found in cache (***cache miss***), search in next level
    - Bring entire ***cache block*** containing requested data into this cache once found
  - **Average Memory Access Time** (AMAT) = HT + MR × MP
    - Hurt by Miss Rate and Miss Penalty

Hit takes HT

Miss takes HT + MP