Lecture Participation Poll #17

Log onto pollev.com/cse374
Or
Text CSE374 to 22333

Lecture 17: Memory Architecture

CSE 374: Intermediate Programming Concepts and Tools

Administrivia

Be kind to yourself and one another ☺

Reminder: Midpoint Deadline Friday November 6th at 9pm PST

Thought experiment

```
public int sum1(int n, int m, int[][] table) {
   int output = 0;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         output += table[i][j];
      }
   }
   return output;
}
```

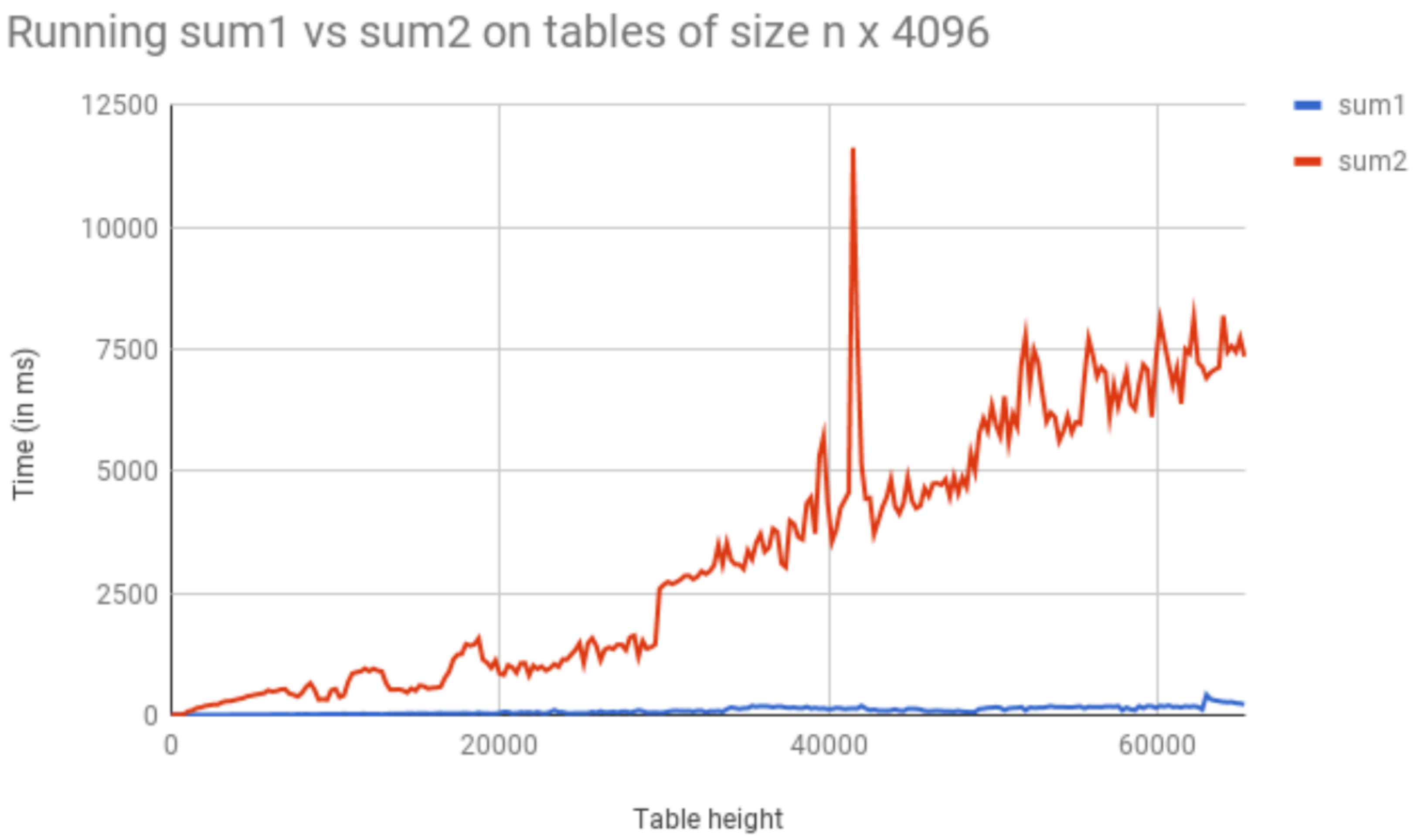
```
public int sum2(int n, int m, int[][] table) {
   int output = 0;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         output += table[j][i];
      }
   }
   return output;
}
```

What do these two methods do?

What is the big-Θ

Θ(n*m)

Thought Experiment Graphed



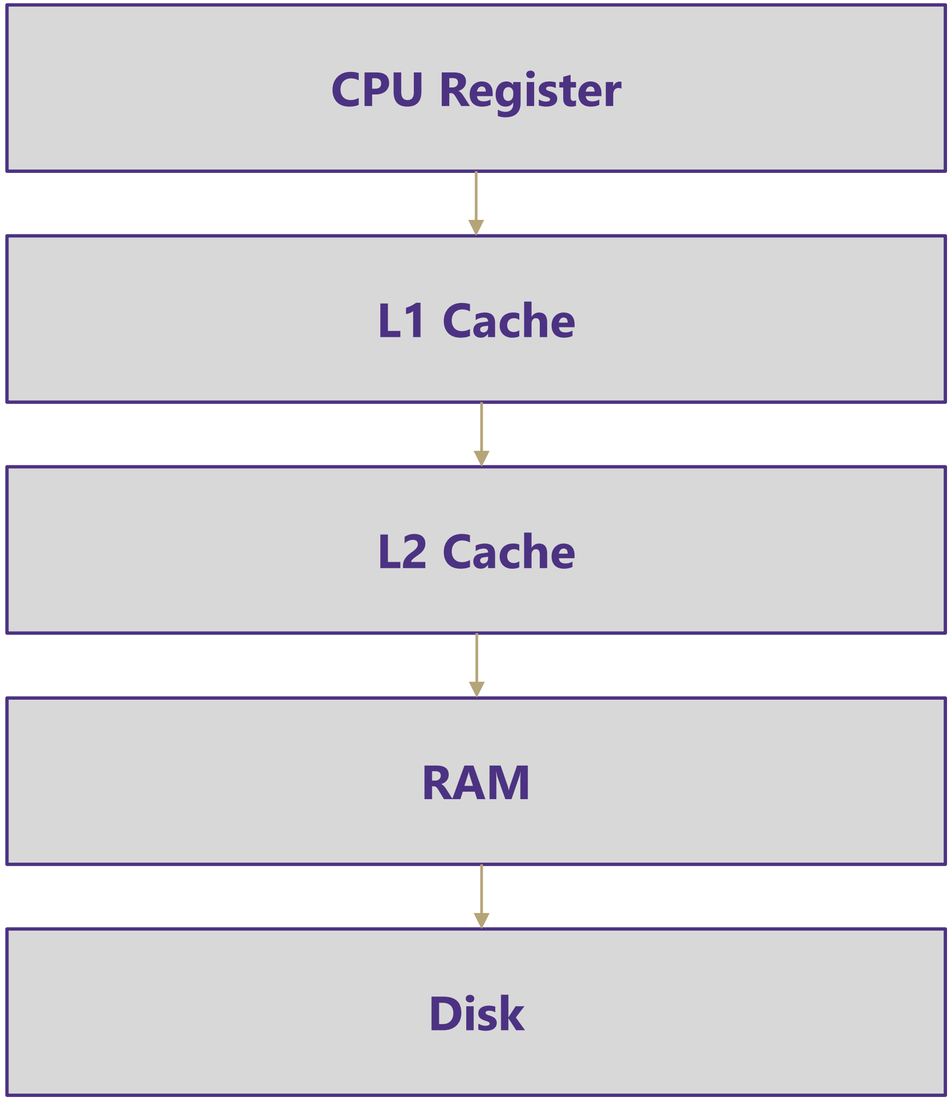
Incorrect Assumptions

- Accessing memory is a quick and constant-time operation

Lies!

- Sometimes accessing memory is cheaper and easier than at other times
- Sometimes accessing memory is very slow

Memory Architecture

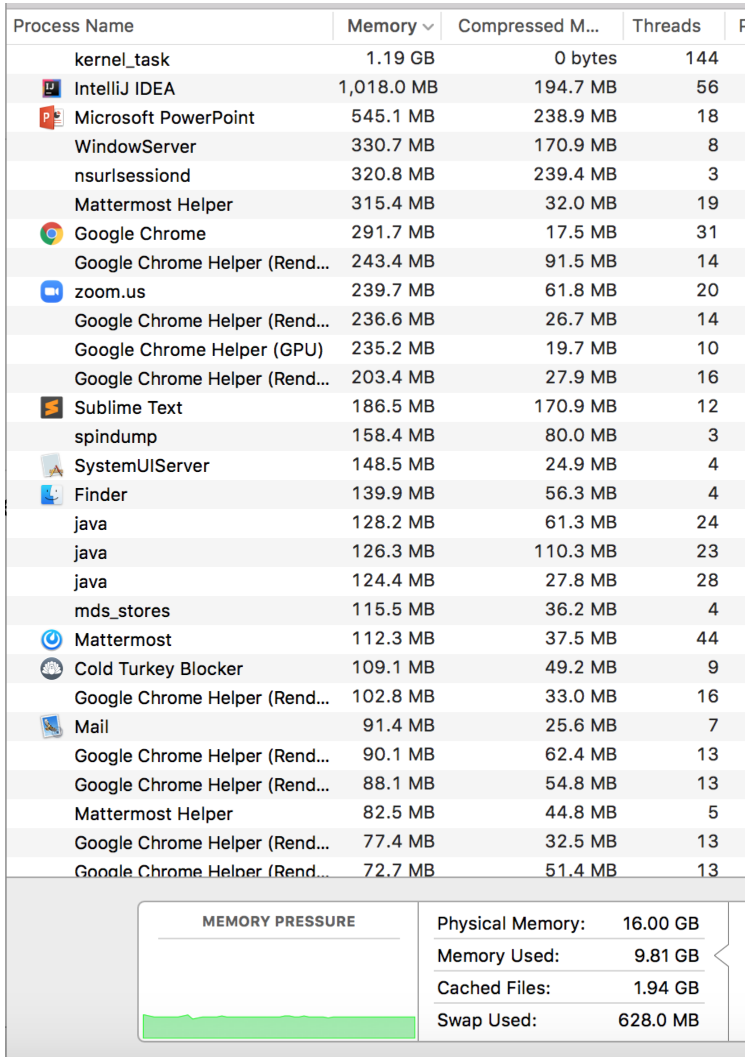


What is it?	Typical Size	Time
The brain of the computer!	32 bits	≈free
Extra memory to make accessing it faster	128KB	0.5 ns
Extra memory to make accessing it faster	2MB	7 ns
Working memory, what your programs need	8GB	100 ns
Large, longtime storage	1 TB	8,000,000 ns

RAM (Random-Access Memory)

- - RAM is where data gets stored for the programs you run. Think of it as the main memory storage location for your programs.

- RAM goes by a ton of different names: memory, main memory, RAM are all names for this same thing.



Process Name	Memory	Compressed M...	Threads
kernel_task	1.19 GB	0 bytes	144
IntelliJ IDEA	1,018.0 MB	194.7 MB	56
Microsoft PowerPoint	545.1 MB	238.9 MB	18
WindowServer	330.7 MB	170.9 MB	8
nsurlsessiond	320.8 MB	239.4 MB	3
Mattermost Helper	315.4 MB	32.0 MB	19
Google Chrome	291.7 MB	17.5 MB	31
Google Chrome Helper (Rend...	243.4 MB	91.5 MB	14
zoom.us	239.7 MB	61.8 MB	20
Google Chrome Helper (Rend...	236.6 MB	26.7 MB	14
Google Chrome Helper (GPU)	235.2 MB	19.7 MB	10
Google Chrome Helper (Rend...	203.4 MB	27.9 MB	16
Sublime Text	186.5 MB	170.9 MB	12
spindump	158.4 MB	80.0 MB	3
SystemUIServer	148.5 MB	24.9 MB	4
Finder	139.9 MB	56.3 MB	4
java	128.2 MB	61.3 MB	24
java	126.3 MB	110.3 MB	23
java	124.4 MB	27.8 MB	28
mds_stores	115.5 MB	36.2 MB	4
Mattermost	112.3 MB	37.5 MB	44
Cold Turkey Blocker	109.1 MB	49.2 MB	9
Google Chrome Helper (Rend...	102.8 MB	33.0 MB	16
Mail	91.4 MB	25.6 MB	7
Google Chrome Helper (Rend...	90.1 MB	62.4 MB	13
Google Chrome Helper (Rend...	88.1 MB	54.8 MB	13
Mattermost Helper	82.5 MB	44.8 MB	5
Google Chrome Helper (Rend...	77.4 MB	32.5 MB	13
Google Chrome Helper (Rend...	72.7 MB	51.4 MB	13

MEMORY PRESSURE

Physical Memory:	16.00 GB
Memory Used:	9.81 GB
Cached Files:	1.94 GB
Swap Used:	628.0 MB

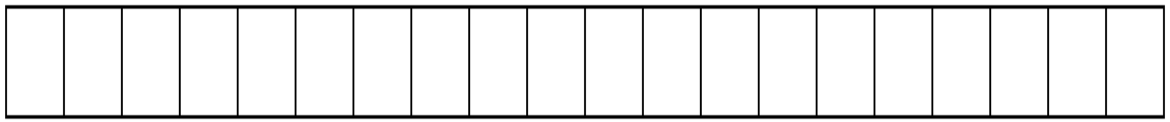
RAM can be represented as a huge array

RAM:
- addresses, storing stuff at specific locations
- random access

Arrays
- indices, storing stuff at specific locations
- random access

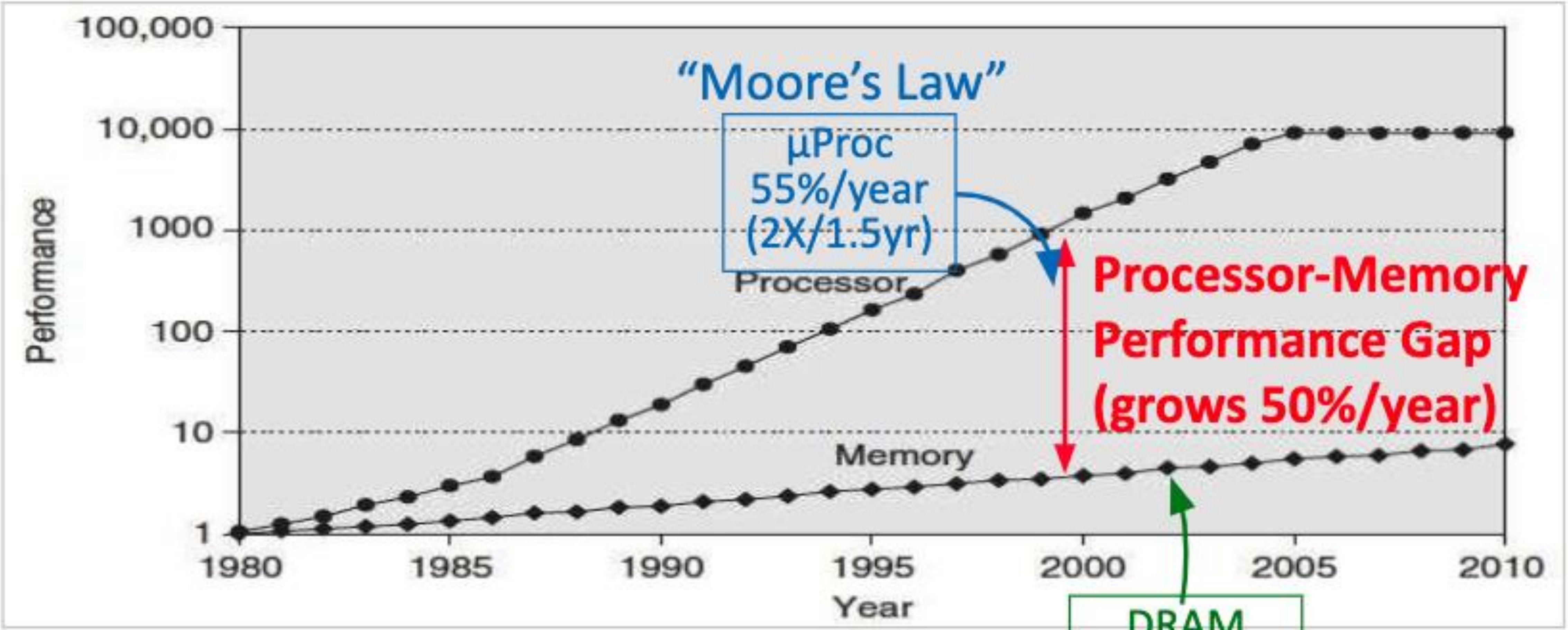
This is a main takeaway

=



If you're interested in deeper than this : https://www.youtube.com/watch?v=fpnE6UAfbtU or take some EE classes?

Processor – Memory Gap



1989 first Intel CPU with cache on chip

1998 Pentium III has two cache levels on chip

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

CPU | Reg

Bus latency / bandwidth evolved much slower

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)*

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

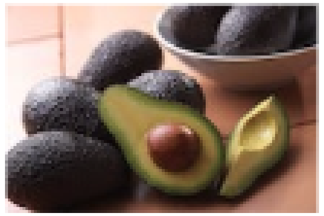
Bus latency / bandwidth evolved much slower

CPU | Reg

Cache

Main Memory

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

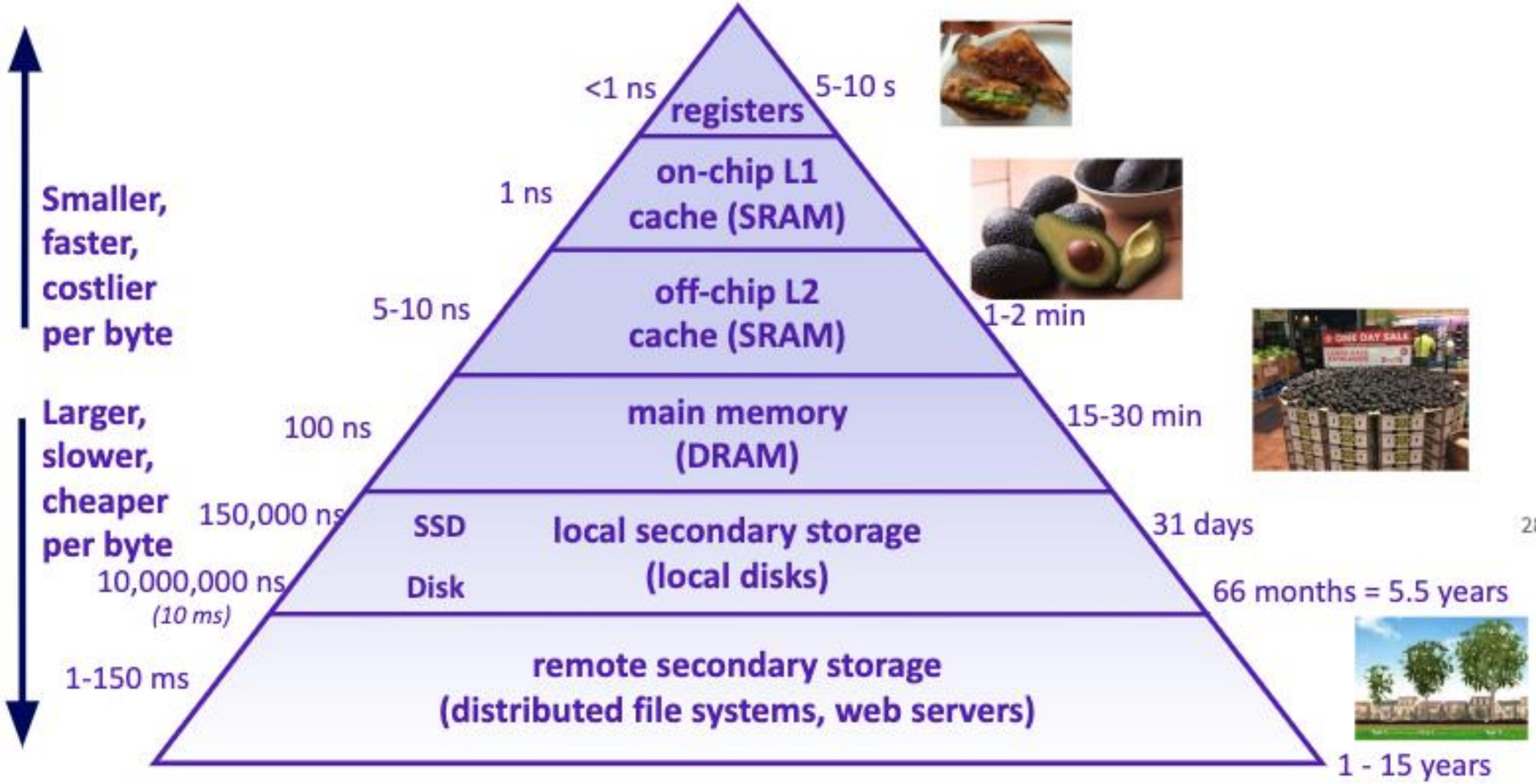


Core 2 Duo:
Bandwidth
2 Bytes/cycle
Latency
100-200 cycles (30-60ns)

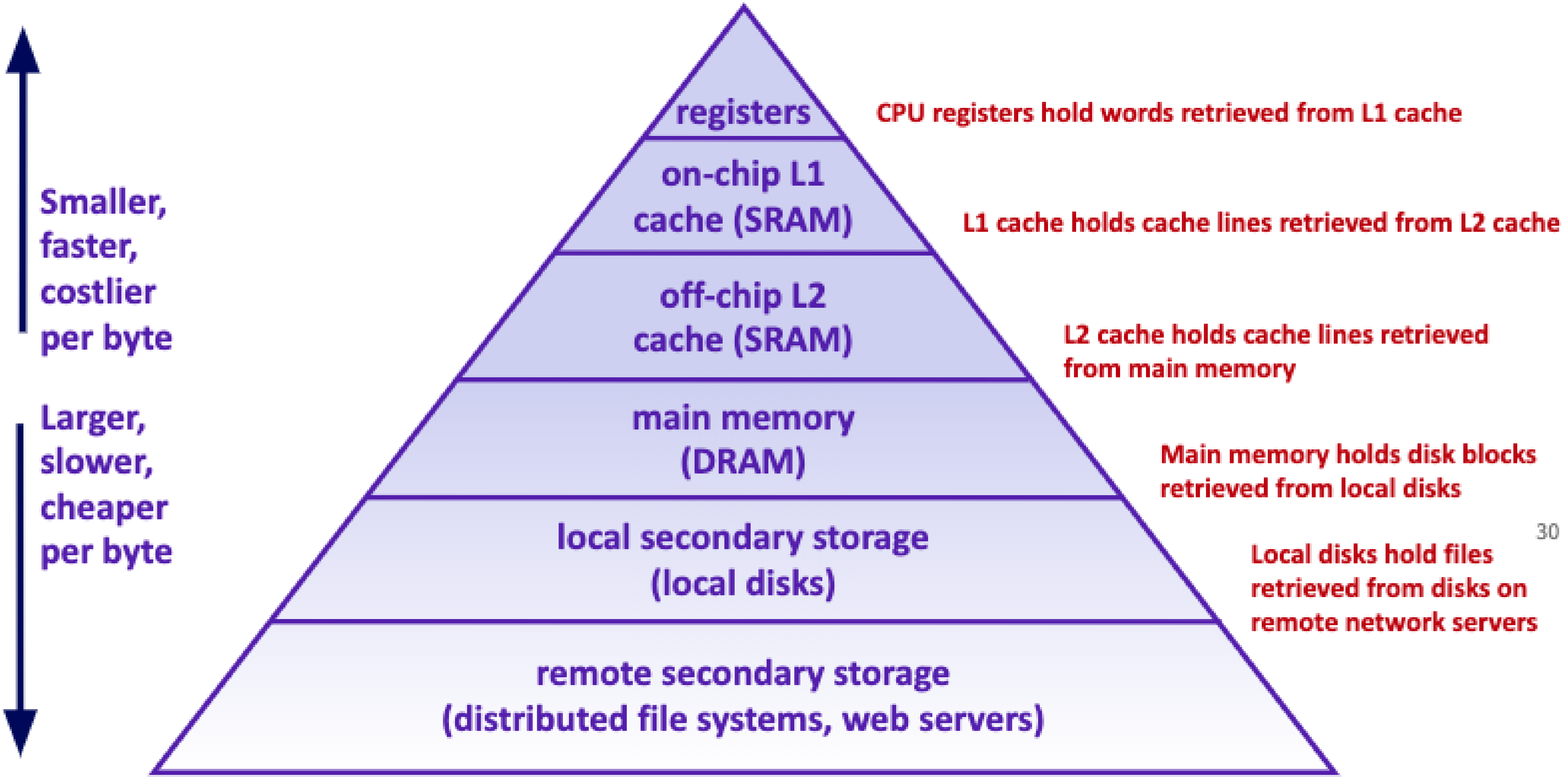
Solution: caches

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

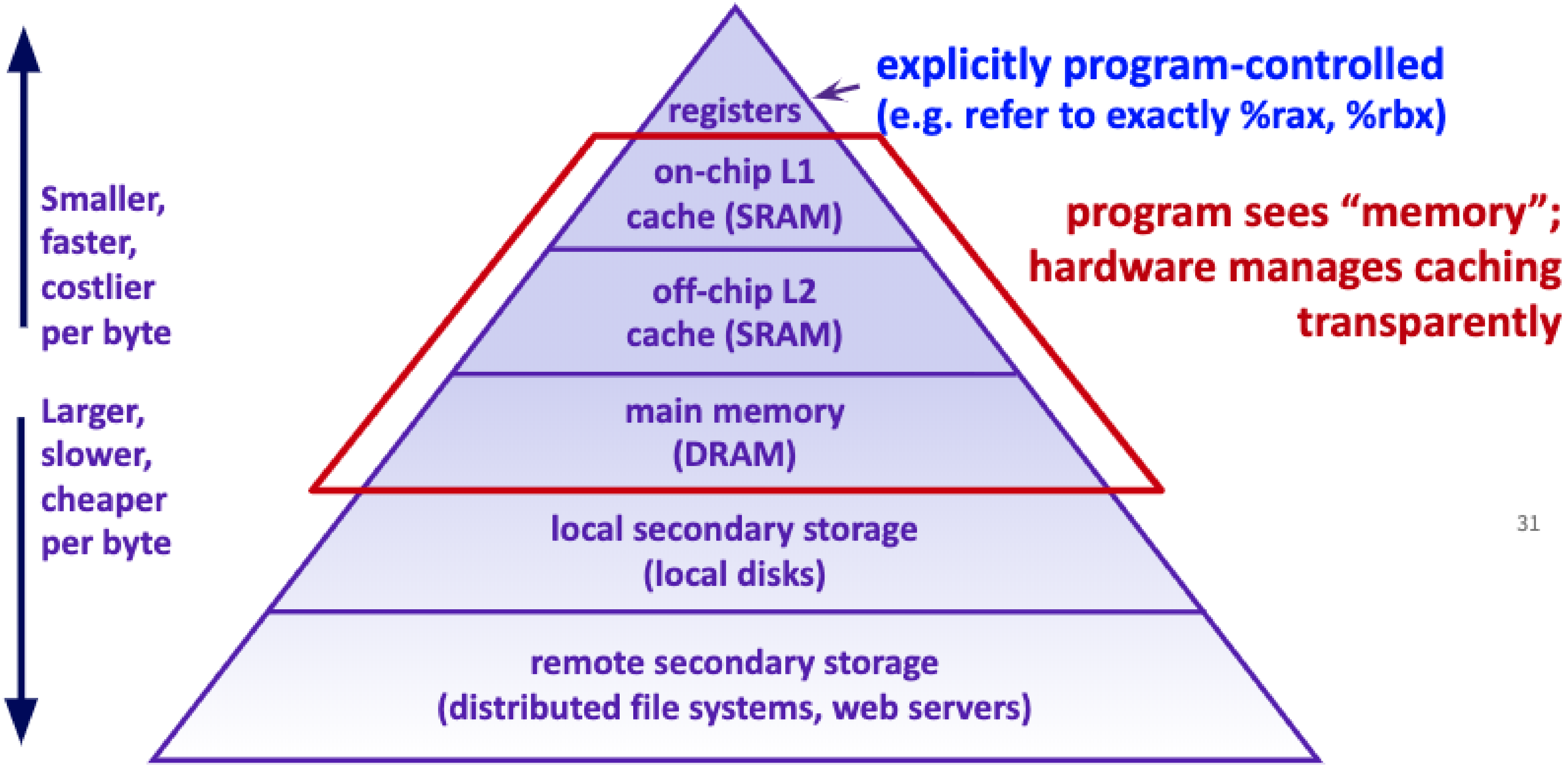
Example Memory Hierarchy



Example Memory Hierarchy



Example Memory Hierarchy



Review: Binary, Bits and Bytes

- binary
- A base-2 system of representing numbers using only 1s and 0s
- - vs decimal, base 10, which has 9 symbols

- bit
- The smallest unit of computer memory represented as a single binary value either 0 or 1

byte

The most commonly referred to unit of memory, a grouping of 8 bits

Can represent 265 different numbers (28)

1 Kilobyte = 1 thousand bytes (kb)

1 Megabyte = 1 million bytes (mb)

1 Gigabyte = 1 billion bytes (gb)

Decimal	Decimal Break Down	Binary	Binary Break Down
0	$(0 * 10^0)$	0	$(0 * 2^0)$
1	$(1 * 10^0)$	1	$(1 * 2^0)$
10	$(1 * 10^1) + (0 * 10^0)$	1010	$(1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)$
12	$(1 * 10^1) + (2 * 10^0)$	1100	$(1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (0 * 2^0)$
127	$(1 * 10^2) + (1 * 10^1) + (2 * 10^0)$	01111111	$(0 * 2^7) + (1 * 2^6) + (1 * 2^5) + (1 * 2^4)(1 * 2^3) + (1 * 2^2) + (1 * 2^1) + (1 * 2^0)$

Memory Architecture

Takeaways:

- the more memory a layer can store, the slower it is (generally)

- accessing the disk is **very** slow

Computer Design Decisions

- Physics
  - Speed of light
  - Physical closeness to CPU
- Cost
  - "good enough" to achieve speed
  - Balance between speed and space

Locality

How does the OS minimize disk accesses?

Spatial Locality

Computers try to partition memory you are likely to use close by

- Arrays
- Fields

Temporal Locality

Computers assume the memory you have just accessed you will likely access again in the near future

Leveraging Spatial Locality

When looking up address in “slow layer”

- bring in more than you need based on what’s near by
- cost of bringing 1 byte vs several bytes is the same
- Data Carpool!

Leveraging Temporal Locality

When looking up address in “slow layer”

Once we load something into RAM or cache, keep it around or a while

- But these layers are smaller
  - When do we “evict” memory to make room?

Moving Memory

Amount of memory moved from **disk** to **RAM**

- Called a "**block**" or "**page**"
  - ≈4kb
  - Smallest unit of data on disk

Amount of memory moved from **RAM** to **Cache**

- called a "**cache line**"
  - ≈64 bytes

Operating System is the Memory Boss

- controls page and cache line size
- decides when to move data to cache or evict

Example Revisited

```
public int sum1(int n, int m, int[][] table) {
   int output = 0;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         output += table[i][j];
      }
   }
   return output;
}
```

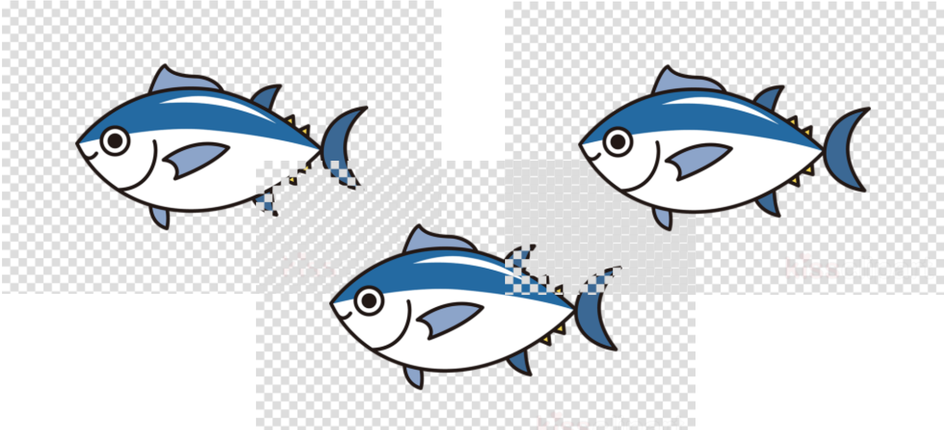
```
public int sum2(int n, int m, int[][] table) {
   int output = 0;
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         output += table[j][i];
      }
   }
   return output;
}
```

Why does sum1 run so much faster than sum2?
sum1 takes advantage of spatial and temporal locality

0			1			2			3			4		
0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
‘a’	‘b’	‘c’	‘d’	‘e’	‘f’	‘g’	‘h’	‘i’	‘j’	‘k’	‘l’	‘m’	‘n’	‘o’

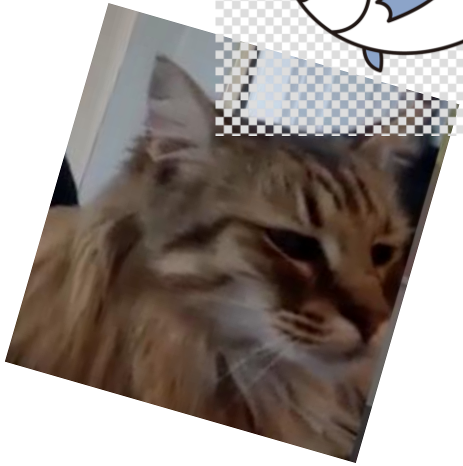
How memory is used and moves around

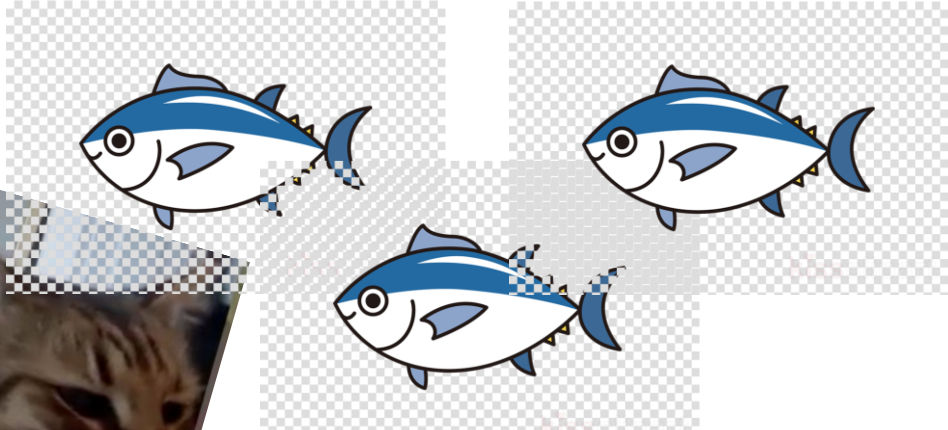

shutterstock.com • 298428176




SAFEWAY

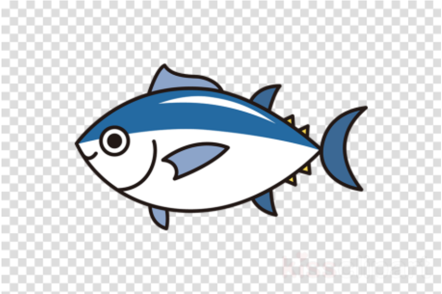

shutterstock.com • 298428176

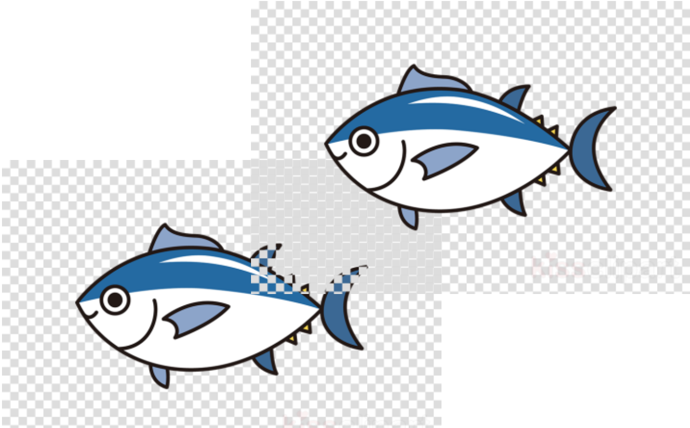





SAFEWAY

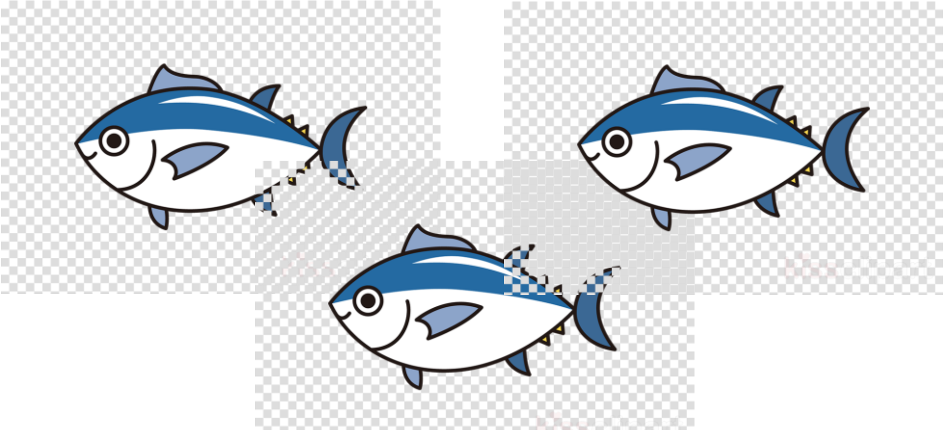

shutterstock.com • 298428176





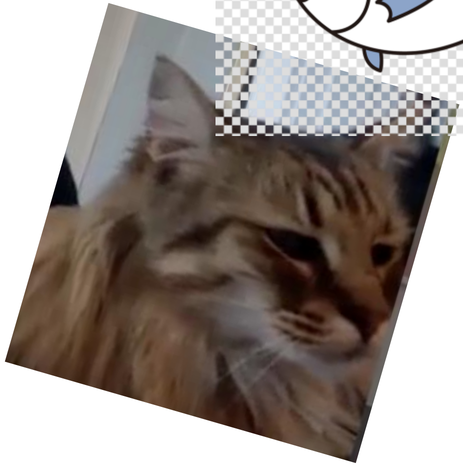

SAFEWAY

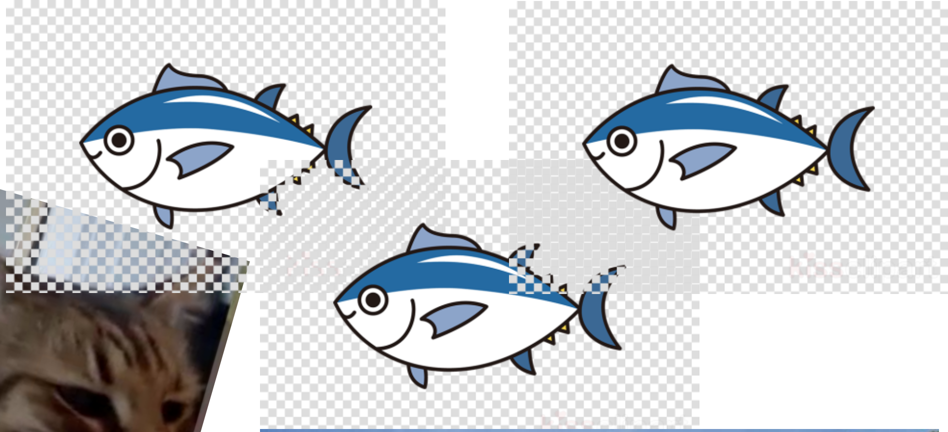

shutterstock.com • 298428176




SAFEWAY

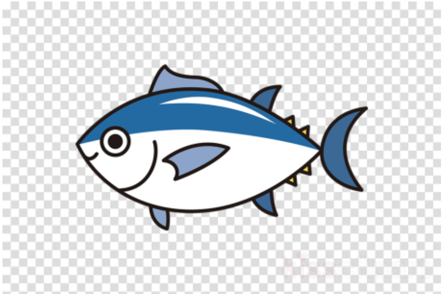

shutterstock.com • 298428176

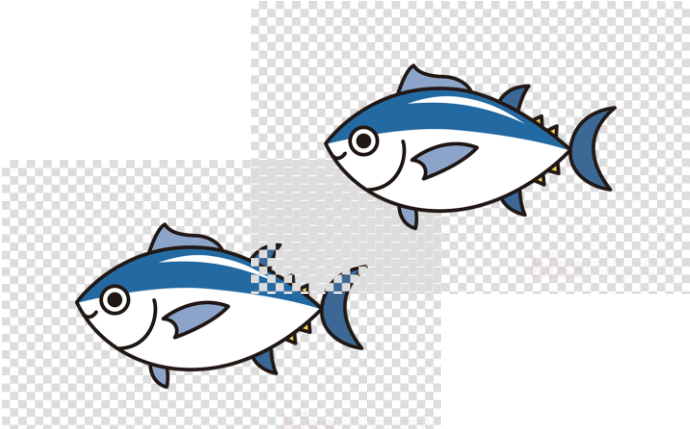





SAFEWAY

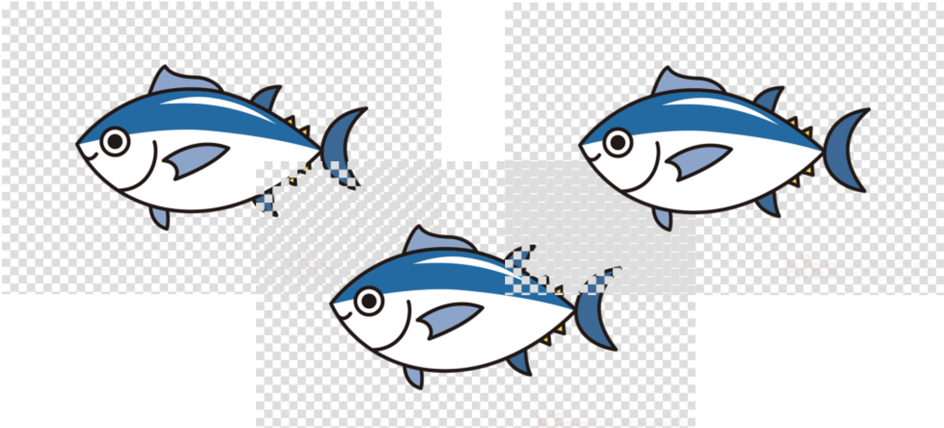

shutterstock.com • 298428176





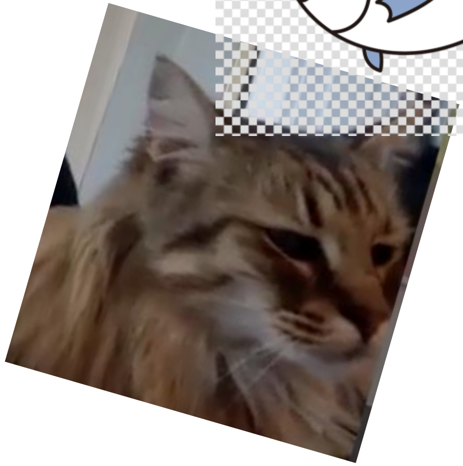

SAFEWAY

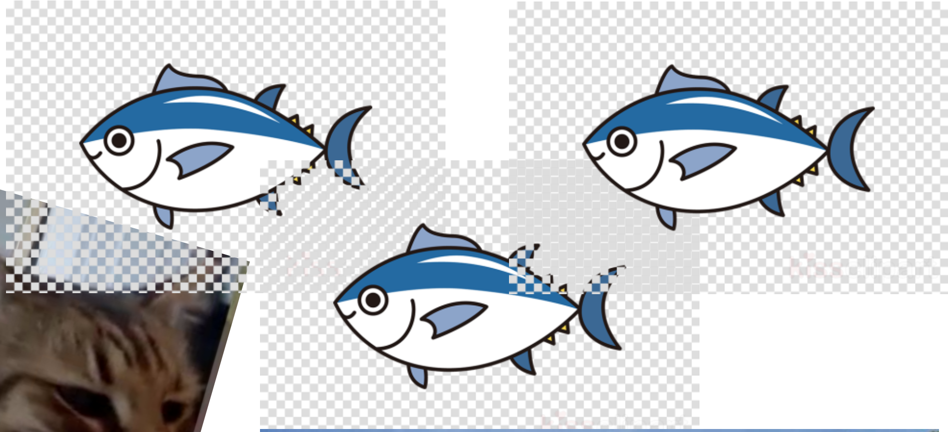

shutterstock.com • 298428176




SAFEWAY

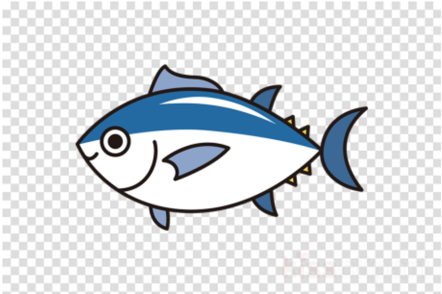

shutterstock.com • 298428176

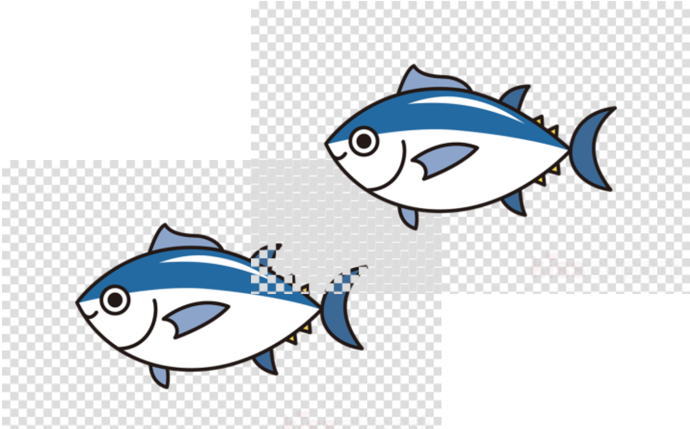





SAFEWAY

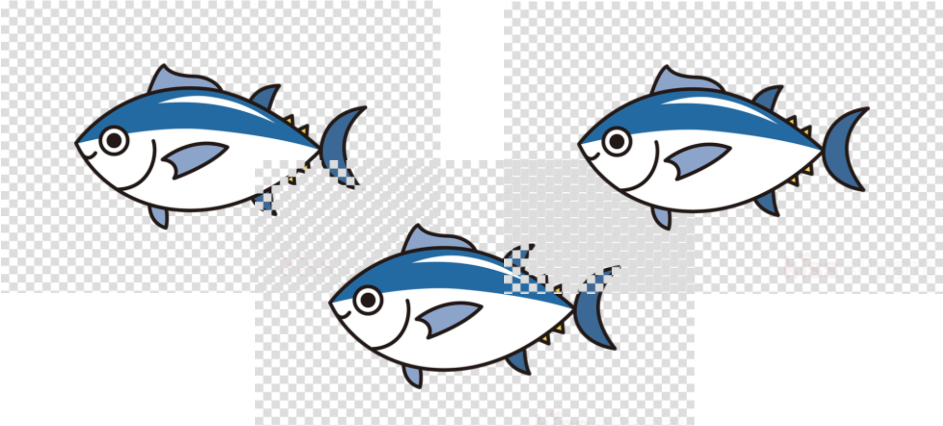

shutterstock.com • 298428176





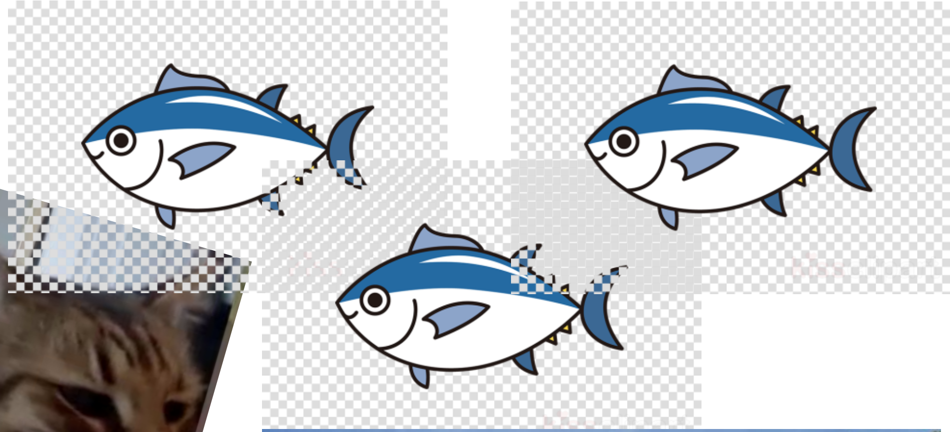

SAFEWAY

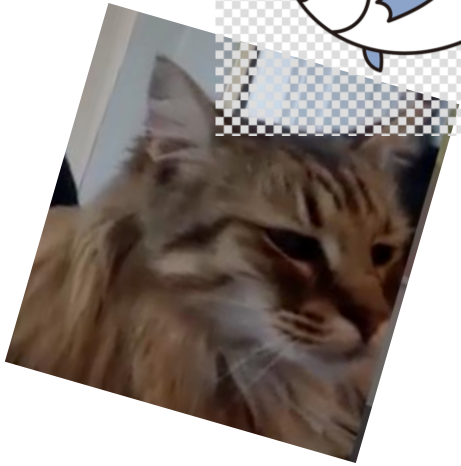

shutterstock.com • 298428176




SAFEWAY

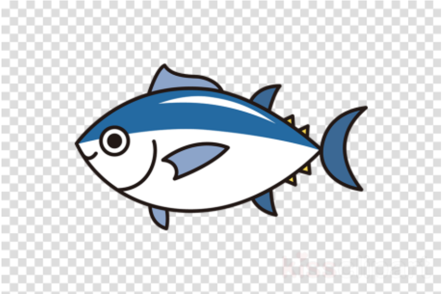
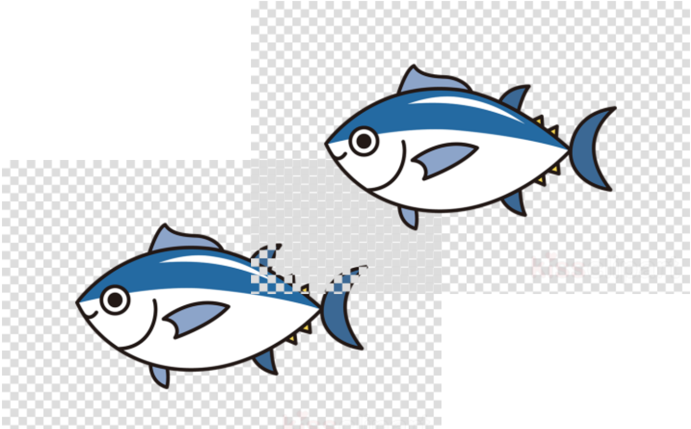

shutterstock.com • 298428176





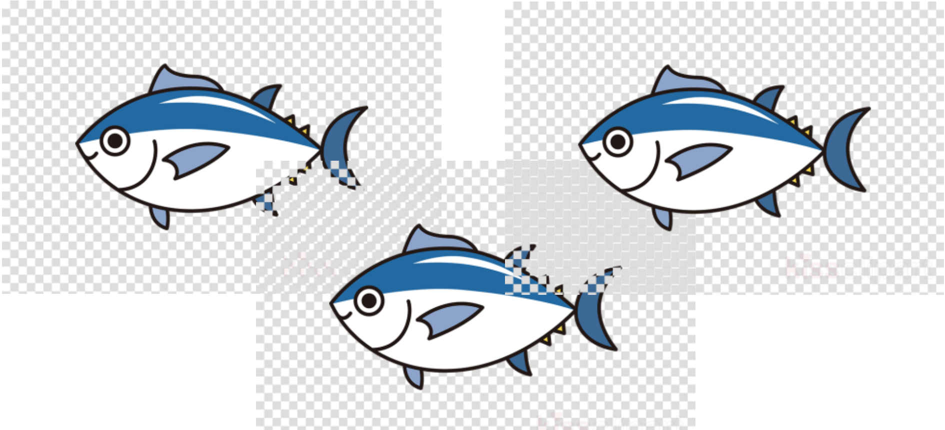

SAFEWAY


shutterstock.com • 298428176

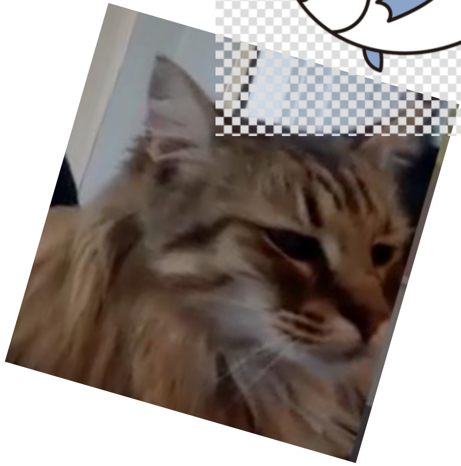
SAFEWAY

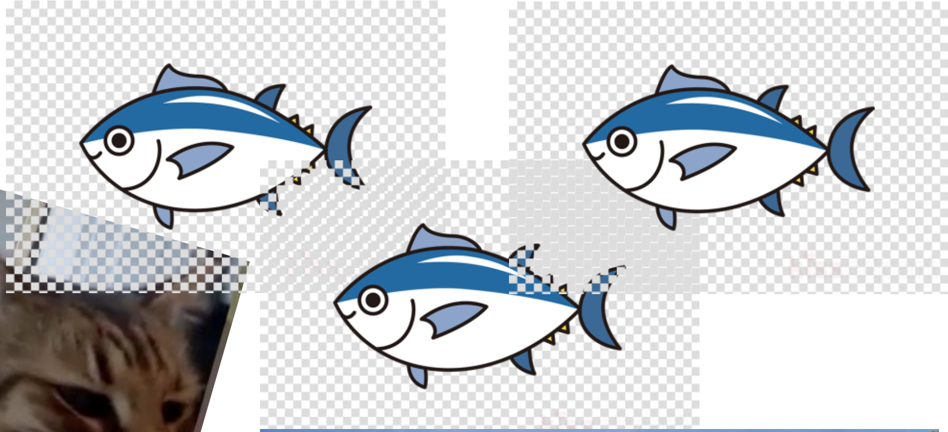

shutterstock.com • 298428176




SAFEWAY

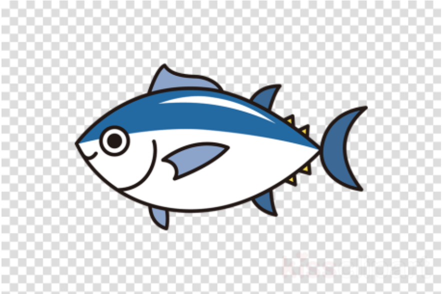

shutterstock.com • 298428176

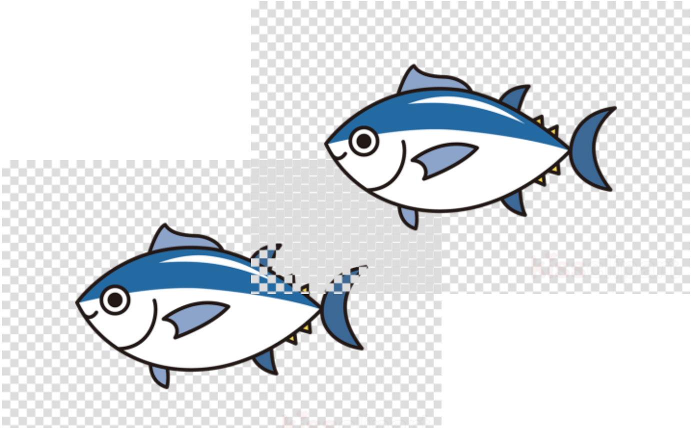





SAFEWAY

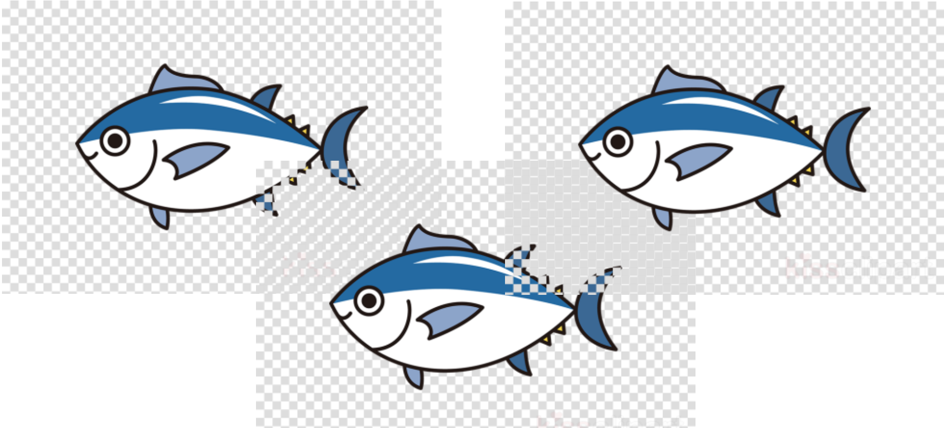

shutterstock.com • 298428176






SAFEWAY


shutterstock.com • 298428176



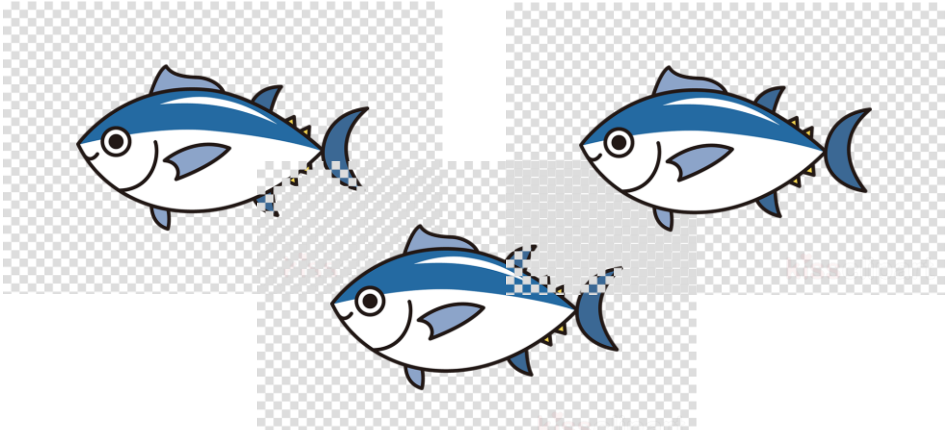

SAFEWAY

Solution to Mercy’s traveling problem

- If we know Mercy is going to keep eating tuna . . . . Why not buy a bunch during a single trip and save them all somewhere closer than the store?

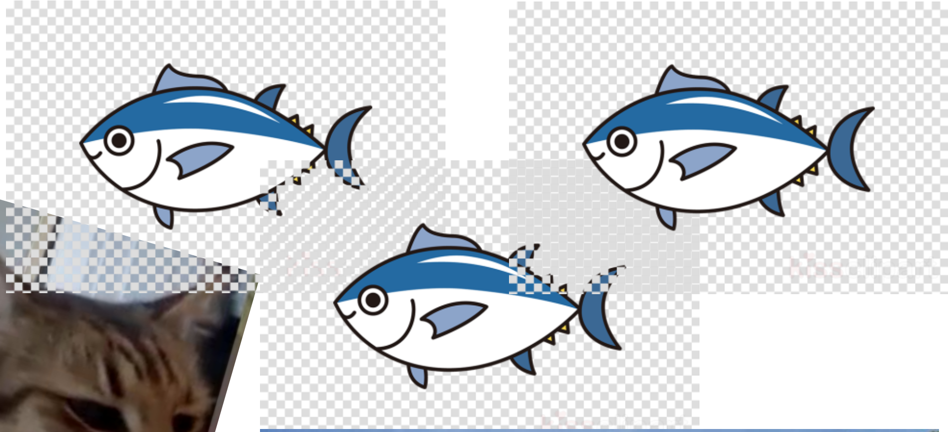
- Let’s get Mercy a refrigerator!

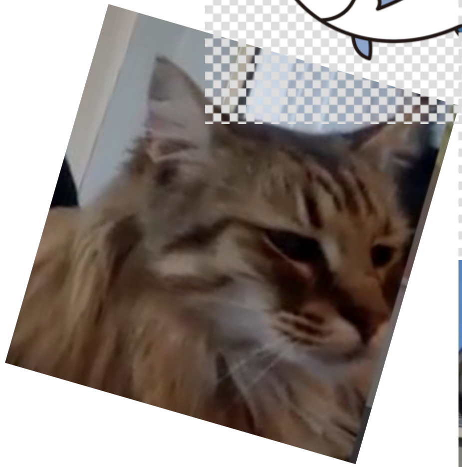

shutterstock.com • 298428176




SAFEWAY

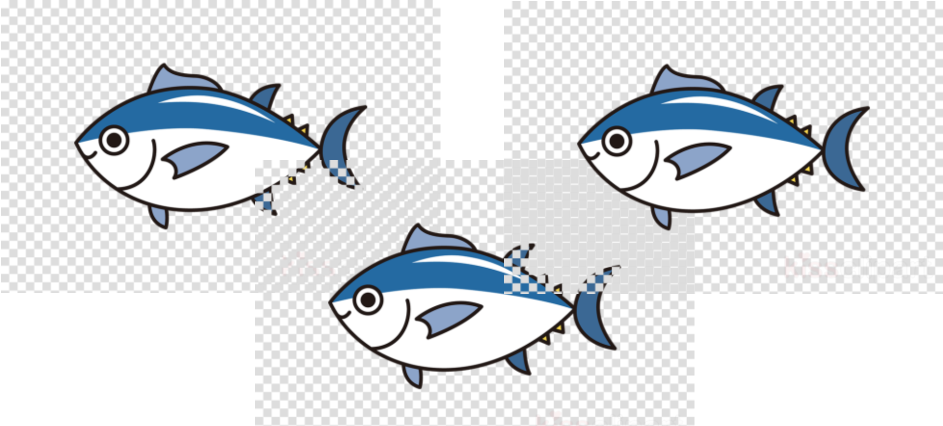

shutterstock.com • 298428176






SAFEWAY


shutterstock.com • 298428176




SAFEWAY

Recap + connecting analogy back to computer

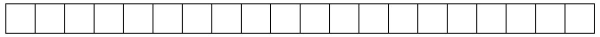
Before

CPU

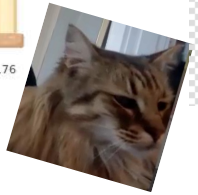
CPU – kind of like the home / brain of your computer. Pretty much all computation is done here and data needs to move here to do anything significant with it (math, if checks, normal statement execution).

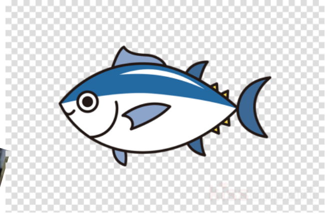
Data travels between RAM and the CPU, but it's slow

RAM



After





Bring a bunch of food back when you go all the way to the store

CPU

Cache!

Bring a bunch of data back when you go all the way to RAM

RAM

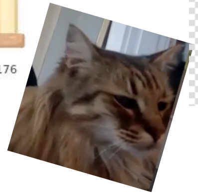
Cache

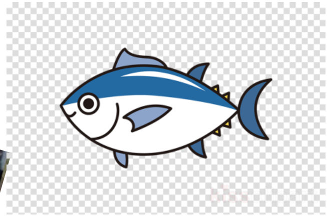
- Rough definition: a place to store some memory that's smaller and closer to the CPU compared to RAM. Because caches are closer to the CPU (where your data generally needs to go to be computed / modified / acted on) getting data from cache to CPU is a lot quicker than from RAM to CPU. This means we love when the data we want to access is conveniently in the cache.

- Generally we always store some data here in hopes that it will be used in the future and that we save ourselves the distance / time it takes to go to RAM.

- Analogy from earlier: The refrigerator (a cache) in your house to store food closer to you than the store. Walking to your fridge is much quicker than walking to the store!

After





Bring a bunch of food back when you go all the way to the store

CPU

This is a big idea!

Cache!

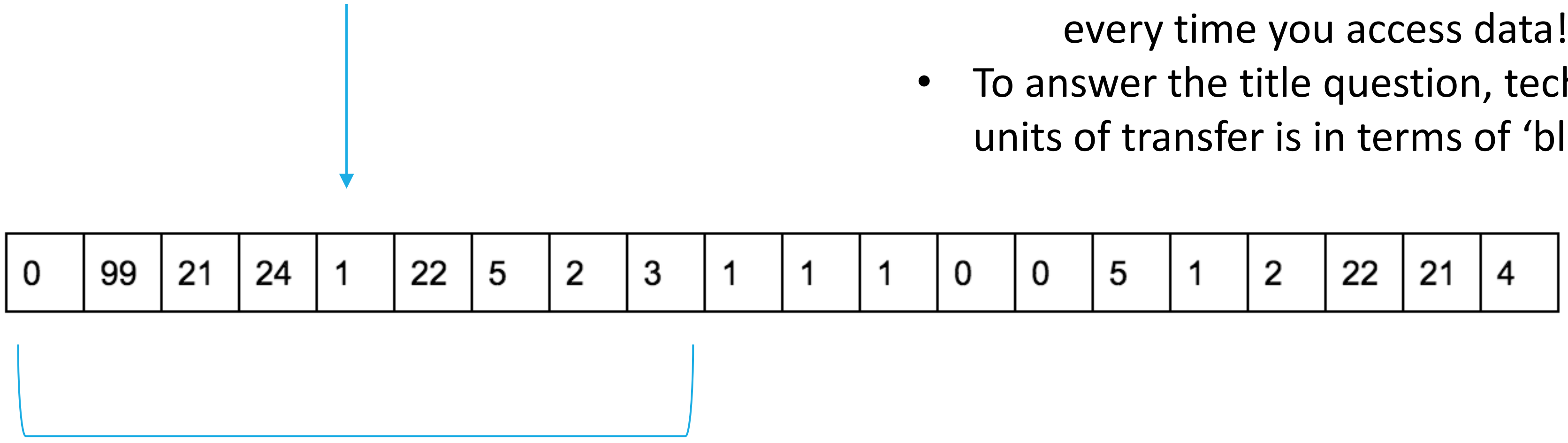
Bring a bunch of data back when you go all the way to RAM

RAM

How is a bunch of memory taken from RAM?

This is a big idea (continued)!

- Imagine you want to retrieve the 1 at index 4 in RAM
- Your computer is smart enough to know to grab some of the surrounding data because computer designers think that it's reasonably likely you'll want to access that data too.
  - (You don't have to do anything in your code for this to happen – it happens automatically every time you access data!)
- To answer the title question, technically the term / units of transfer is in terms of 'blocks'.



How is a bunch of memory taken from RAM? (continued)

CPU

original data (the 1) we wanted to look up gets passed back to the cpu

cache

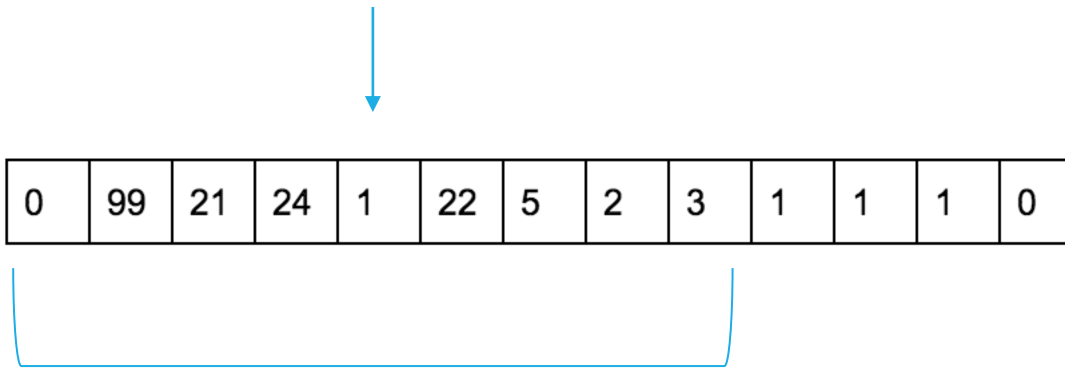
all the data from the block gets brought to the cache

0	99	21	24	1	22	5	2	3	1	1	1	0	0	5	1	2	22	21	4

How does this pattern of memory grabbing affect our programs?

- This should have a major impact on programming with arrays. Say we access an index of an array that is stored in RAM. Because we grab a whole bunch of contiguous memory even when we just access one index in RAM, we'll probably be grabbing other nearby parts of our array and storing that in our cache for quick access later.

Imagine that the below memory is just an entire array of length 13, with some data in it.



Just by accessing one element we bring the nearby elements back with us to the cache. In this case, it's almost all of the array!

Another demo, but timed

- https://repl.it/repls/MistyroseLinedTransformation

- (takes about 15 seconds to run)

Appendix