Lecture 24: Memory and Locality

CSE 373: Data Structures and Algorithms

Warm Up

Merge Sort		Quick Sort first element as pivot		Quick Sort in-place median of values as pivot	
Worst case runtime?	$\Theta(nlogn)$	Worst case runtime?	$\Theta(n^2)$	Worst case runtime?	$\Theta(n \log n)$
Best case runtime?	$\Theta(nlogn)$	Best case runtime?	$\Theta(nlogn)$	Best case runtime?	$\Theta(nlogn)$
In-practice runtime?	$\Theta(nlogn)$	In-practice runtime?	$\Theta(nlogn)$	In-practice runtime?	$\Theta(n \log n)$
Stable?	Yes	Stable?	No	Stable?	No
In-place?	No	In-place?	No	In-place?	Yes

Announcements

Things are tough all over the world right now
- Everyone gets +2 late days (thanks TAs!)
- Extending the late turn in from 3 days after due date to 5 days after due date

P4 Spec Quiz Due today!
- For extra credit
- No late submissions accepted
- P4 due Wednesday June 2nd

Office Hours slight change
- Tas have been instructed to help with ONE step of debugging: identify bug, reproduce bug or resolve bug
- Goal is to move through OH queue faster so you have more questions answered in smaller chunks
- OH Form will be added to OH page and bot

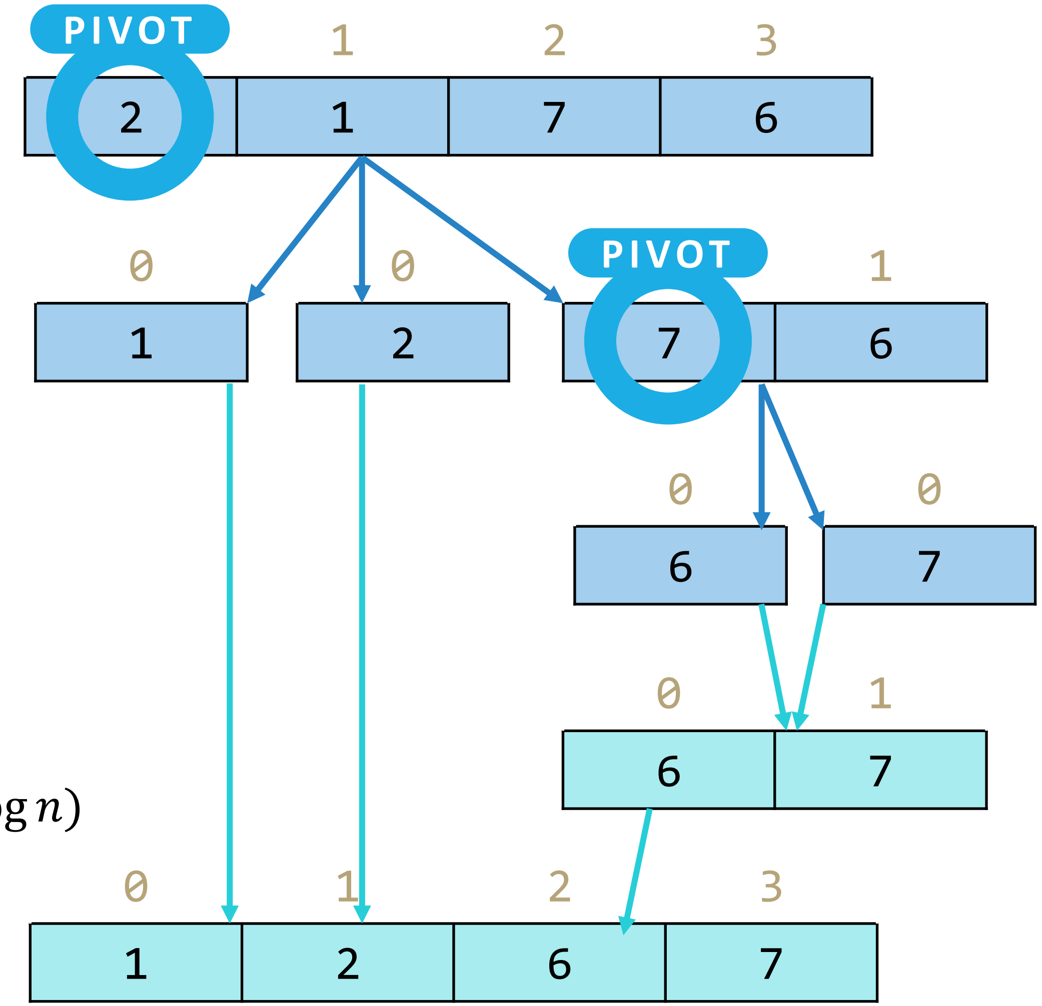
Tech Career Resources
- No BS CS Career Talk recording: https://courses.cs.washington.edu/courses/cse142/21sp/explore.html
- Section 9 Thursday 5/27 Interview Prep

Quick Sort (v1)

```
quickSort(list) {
   if (list.length == 1):
      return list
   else:
      pivot = choosePivot(list)
      smallerHalf = quickSort(getSmaller(pivot, list))
      largerHalf = quickSort(getBigger(pivot, list))
      return smallerHalf + pivot + largerHalf
}
```

Worst case: Pivot only chops off one value

Best case: Pivot divides each array in half



Worst case runtime? $T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ T(n-1) + n & \text{otherwise} \end{cases} = \Theta(n^2)$

Best case runtime? $T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + n & \text{otherwise} \end{cases} = \Theta(n \log n)$

In-practice runtime? Just trust me: $\Theta(n \log n)$

(absurd amount of math to get here)

Stable? No

In-place? Can be done!

Can we do better?

How to avoid hitting the worst case?

- It all comes down to the pivot. If the pivot divides each array in half, we get better behavior

Here are four options for finding a pivot. What are the tradeoffs?

- Just take the first element
- Take the median of the full array
- Take the median of the first, last, and middle element
- Pick a random element

Strategies for Choosing a Pivot

Just take the first element

- Very fast!
- But has worst case: for example, sorted lists have $\Omega(n^2)$ behavior

Take the median of the full array

- Can actually find the median in $O(n)$ time (google QuickSelect). It's **complicated.**
- $O(n\ log\ n)$ even in the worst case… but the constant factors are **awful**. No one does quicksort this way.

Take the median of the first, last, and middle element

Most commonly used

- Makes pivot slightly more content-aware, at least won't select very smallest/largest
- Worst case is still $\Omega(n^2)$, but on real-world data tends to perform well!

Pick a random element

- Get $O(n \log n)$ runtime with probability at least $1 - 1/n^2$
- No simple worst-case input (e.g. sorted, reverse sorted)

Quick Sort (v2: In-Place)

Divide

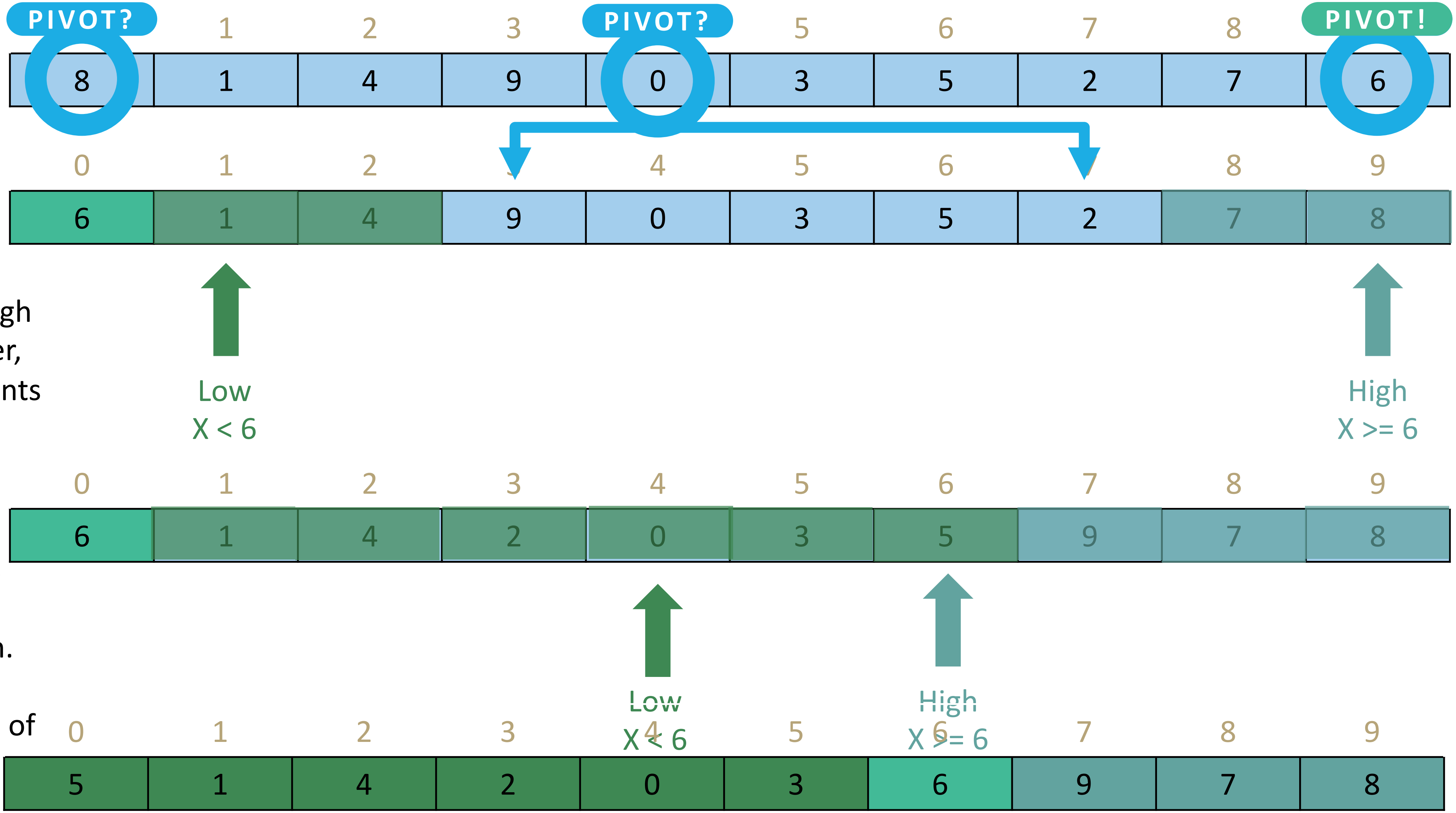
Select a pivot

Move pivot out of the way

Bring low and high pointers together, swapping elements if needed

Meeting point is where pivot belongs; swap in.

Now recurse on smaller portions of same array!



Quick Sort (v2: In-Place)

```
quickSort(list) {
   if (list.length == 1):
      return list
   else:
      pivot = choosePivot(list)
      smallerPart, largerPart = partition(pivot, list)
      smallerPart = quickSort(smallerPart)
      largerPart = quickSort(largerPart)
      return smallerPart + pivot + largerPart
}
```

choosePivot:
- Use one of the pivot selection strategies

partition:
- For in-place Quick Sort, series of swaps to build both partitions at once
- Tricky part: moving pivot out of the way and moving it back!
- Similar to Merge Sort divide step: two pointers, only move smaller one

Worst case runtime? $T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ T(n-1) + n & \text{otherwise} \end{cases} = \Theta(n^2)$

Best case runtime? $T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + n & \text{otherwise} \end{cases} = \Theta(n \log n)$

In-practice runtime? Just trust me: $\Theta(n \log n)$

(absurd amount of math to get here)

Stable? No

In-place? Yes

0	1	2	3	4	5
0	3	6	9	7	8

Can we do better?

We'd really like to avoid hitting the worst case.

Key to getting a good running time, is always cutting the array (about) in half.

How do we choose a good pivot?

Here are four options for finding a pivot. What are the tradeoffs?

- Just take the first element
- Take the median of the first, last, and middle element
- Take the median of the full array
- Pick a random element as a pivot

Pivots

Just take the first element

- fast to find a pivot
- But (e.g.) nearly sorted lists get $\Omega(n^2)$ behavior overall

Take the median of the first, last, and middle element

- Guaranteed to not have the absolute smallest value.
- On real data, this works quite well…
- But worst case is still $\Omega(n^2)$

Median of three is a common choice in practice

Take the median of the full array

- Can actually find the median in $O(n)$ time (google QuickSelect). It's **complicated.**
- $O(n\ log\ n)$ even in the worst case….but the constant factors are **awful**. No one does quicksort this way.

Pick a random element as a pivot

- somewhat slow constant factors
- Get $O(n \log n)$ running time with probability at least $1 - 1/n^2$
- "adversaries" can't make it more likely that we hit the worst case.

Sorting: Summary

	Best-Case	Worst-Case	Space	Stable
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$	No
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(1)$	Yes
Heap Sort	$\Theta(n)$	$\Theta(nlogn)$	$\Theta(n)$	No
In-Place Heap Sort	$\Theta(n)$	$\Theta(nlogn)$	$\Theta(1)$	No
Merge Sort	$\Theta(nlogn)$	$\Theta(nlogn)$	$\Theta(nlogn)$ $\Theta(n)$* optimized	Yes
Quick Sort	$\Theta(nlogn)$	$\Theta(n^2)$	$\Theta(n)$	No
In-place Quick Sort	$\Theta(nlogn)$	$\Theta(n^2)$	$\Theta(1)$	No

What does Java do?

- Actually uses a combination of *3 different sorts*:
  - If objects: use Merge Sort* (stable!)
  - If primitives: use Dual Pivot Quick Sort
  - If "reasonably short" array of primitives: use Insertion Sort
    - Researchers say 48 elements

Key Takeaway: No single sorting algorithm is "the best"!

- Different sorts have different properties in different situations
- The "best sort" is one that is well-suited to your data

* They actually use Tim Sort, which is very similar to Merge Sort in theory, but has some minor details different

STRATEGY 1:
ITERATIVE IMPROVEMENT

Insertion Sort

WORST $\Theta(n^2)$
BEST $\Theta(n)$

Simple, stable, low-overhead, great if already sorted.

IN-PLACE STABLE SPACE $\Theta(1)$

Selection Sort

WORST $\Theta(n^2)$
BEST $\Theta(n^2)$

Minimizes array writes, otherwise never preferred.

IN-PLACE SPACE $\Theta(1)$

STRATEGY 2:
IMPOSE STRUCTURE

Heap Sort

WORST $\Theta(n \log n)$
BEST $\Theta(n)$

Always good runtimes

IN-PLACE SPACE $\Theta(1)$

STRATEGY 3:
DIVIDE AND CONQUER

Merge Sort

WORST $\Theta(n \log n)$
BEST $\Theta(n \log n)$

Stable, very reliable! In-place variant is slower.

STABLE SPACE $\Theta(n)$

Quick Sort

WORST $\Theta(n^2)$
BEST $\Theta(n \log n)$

Fastest in practice (constant factors), bad worst case.

IN-PLACE SPACE $\Theta(1)$

Insertion Sort

WORST $\theta(n^2)$

BEST $\theta(n)$

Simple, stable, low-overhead, great if already sorted.

IN-PLACE STABLE SPACE $\theta(1)$

Merge Sort

WORST $\theta(n \log n)$

BEST $\theta(n \log n)$

Stable, very reliable! In-place variant is slower.

STABLE SPACE $\theta(n)$

Heap Sort

WORST $\theta(n \log n)$

BEST $\theta(n)$

Always good runtimes

IN-PLACE SPACE $\theta(1)$

Selection Sort

WORST $\theta(n^2)$

BEST $\theta(n^2)$

Minimizes array writes, otherwise never preferred.

IN-PLACE SPACE $\theta(1)$

Quick Sort

WORST $\theta(n^2)$

BEST $\theta(n \log n)$

Fastest in practice (constant factors), bad worst case.

IN-PLACE SPACE $\theta(1)$

Can we do better than n log n?

- For comparison sorts, **NO**. A proven lower bound!
  - Intuition: n elements to sort, no faster way to find "right place" than log n
- However, niche sorts can do better in specific situations!

Many cool niche sorts beyond the scope of 373!

Radix Sort (Wikipedia, VisuAlgo) - Go digit-by-digit in integer data. Only 10 digits, so no need to compare!

Counting Sort (Wikipedia)

Bucket Sort (Wikipedia)

External Sorting Algorithms (Wikipedia) - For big data™

But Don't Take it From Me…

Here are some excellent visualizations for the sorting algorithms we've talked about!

Comparing Sorting Algorithms

- Different Types of Input Data: https://www.toptal.com/developers/sorting-algorithms
- More Thorough Walkthrough: https://visualgo.net/en/sorting?slide=1

Comparing Sorting Algorithms

DANCE EDITION

Insertion Sort: https://www.youtube.com/watch?v=ROalU379l3U

Selection Sort: https://www.youtube.com/watch?v=Ns4TPTC8whw

Heap Sort: https://www.youtube.com/watch?v=Xw2D9aJRBY4

Merge Sort: https://www.youtube.com/watch?v=XaqR3G_NVoo

Quick Sort: https://www.youtube.com/watch?v=ywWBy6J5gz8

Memory & Locality!

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)

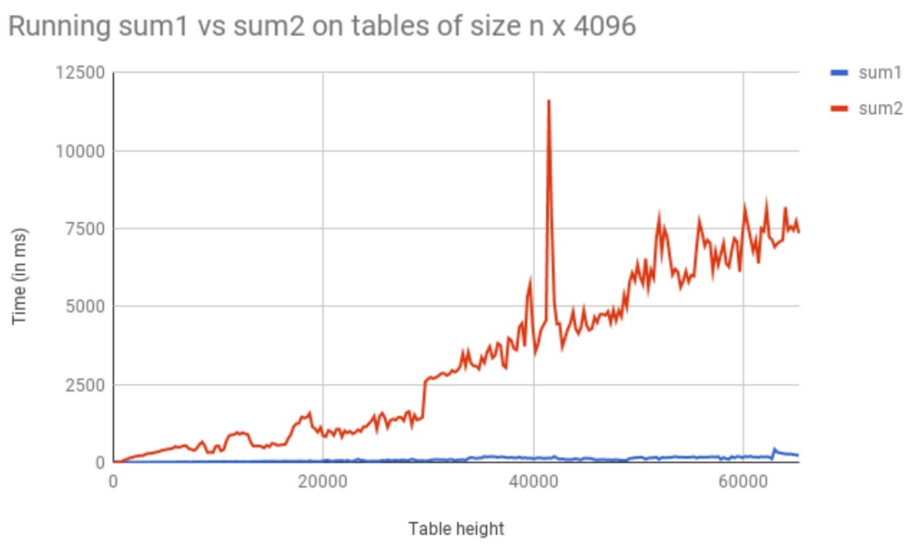
Decimal	Decimal Break Down	Binary	Binary Break Down
0	$(0 * 10^0)$	0	$(0 * 2^0)$
1	$(1 * 10^0)$	1	$(1 * 2^0)$
10	$(1 * 10^1) + (0 * 10^0)$	1010	$(1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)$
12	$(1 * 10^1) + (2 * 10^0)$	1100	$(1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (0 * 2^0)$
127	$(1 * 10^2) + (1 * 10^1)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

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)



Incorrect Assumptions

Accessing memory is a quick and constant-time operation Lies!

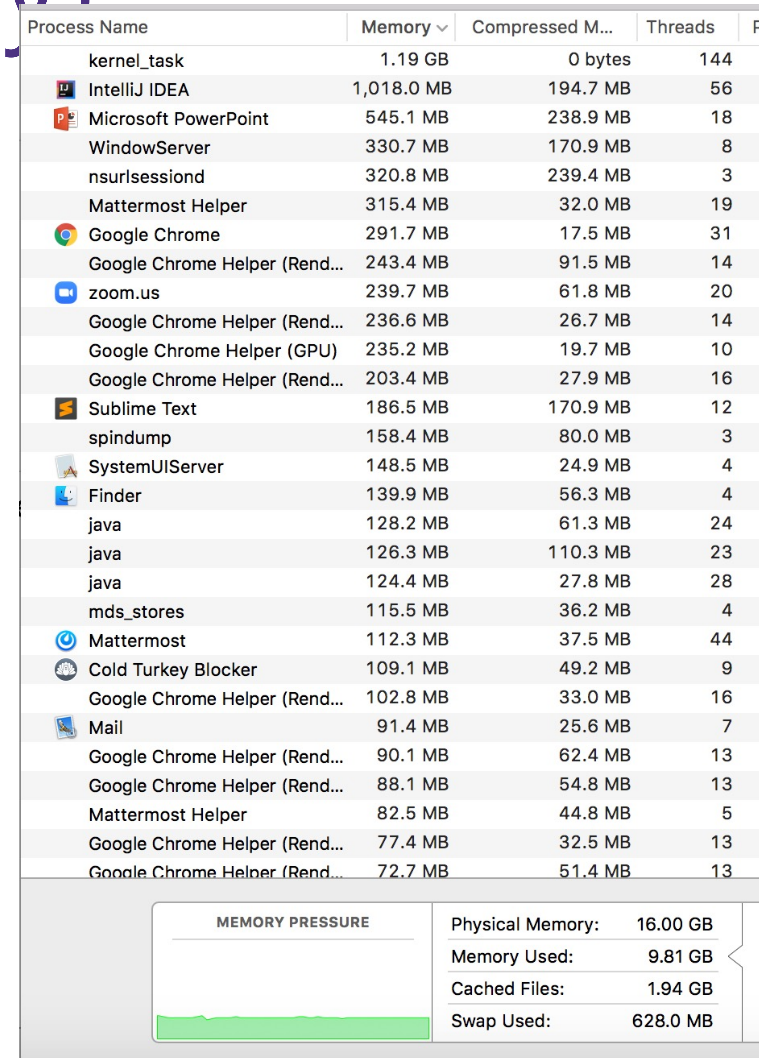
Sometimes accessing memory is cheaper and easier than at other times

Sometimes accessing memory is very slow

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

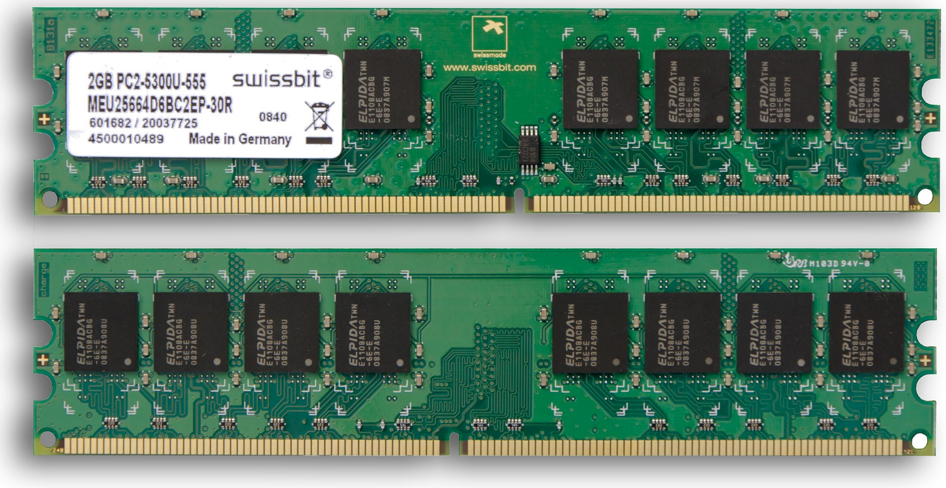
Physical Memory:	16.00 GB
Memory Used:	9.81 GB
Cached Files:	1.94 GB
Swap Used:	628.0 MB

RAM can be represented as a huge array

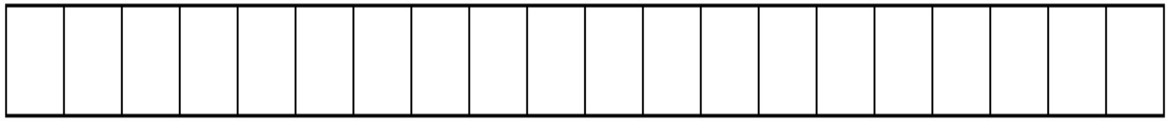
RAM:
- addresses, storing stuff at specific locations
- random access

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

This is a main takeaway



=



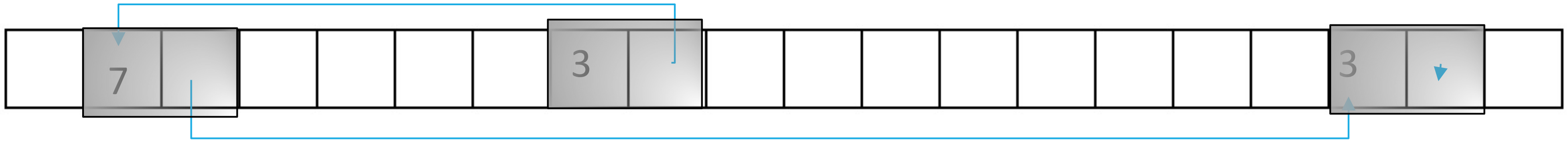
If you're interested in deeper than this : https://www.youtube.com/watch?v=fpnE6UAfbtU or take some EE classes?

A rough view of arrays and linked lists

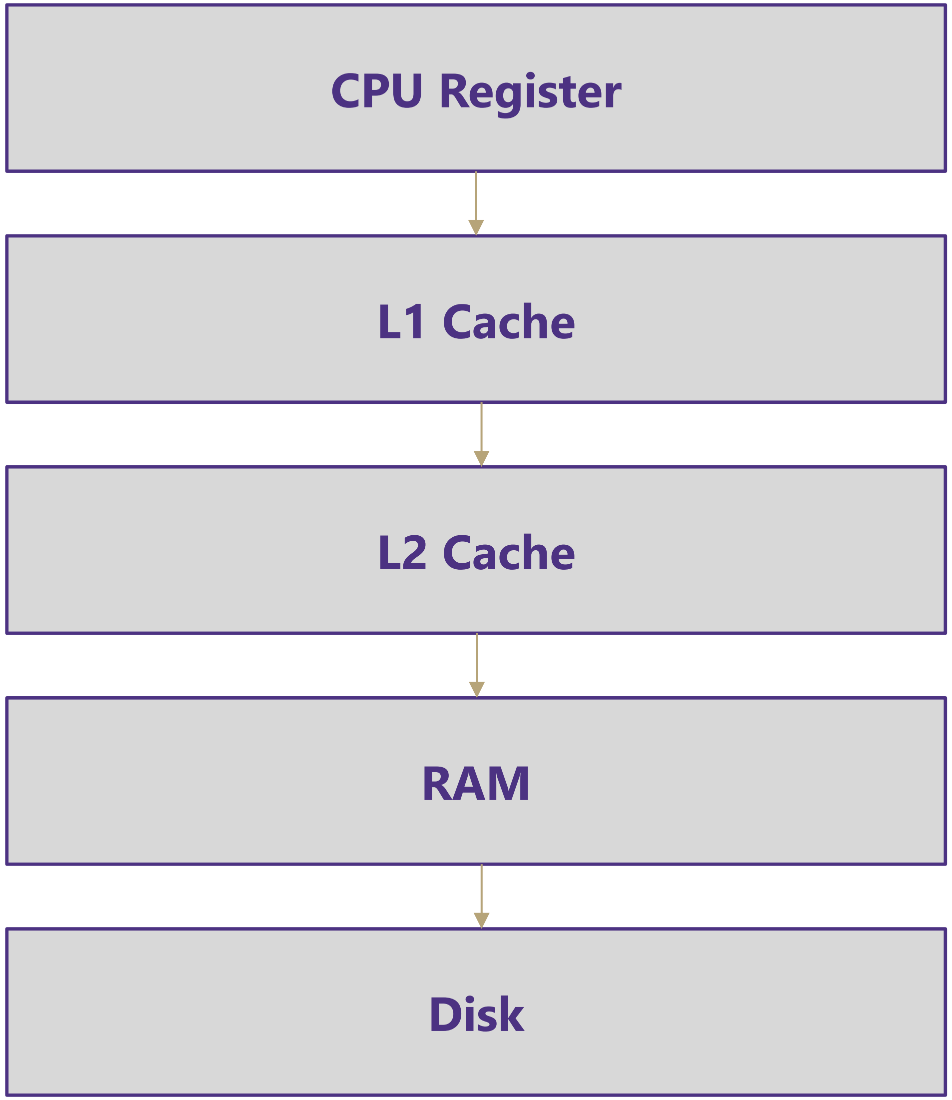
```
int[] array = new int[3];
array[0] = 3;
array[1] = 7;
array[2] = 3;
```


```
Node front = new Node(3);
front.next = new Node(7);
front.next.next = new Node(3);
```

(drawing singly linked list instead of doubly because drawings are hard / the two are similar)



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

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

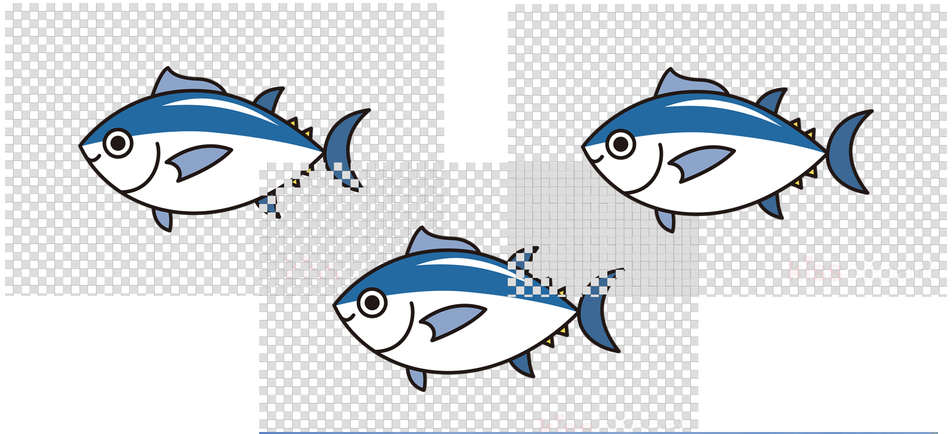
Leveraging Spatial Locality

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!

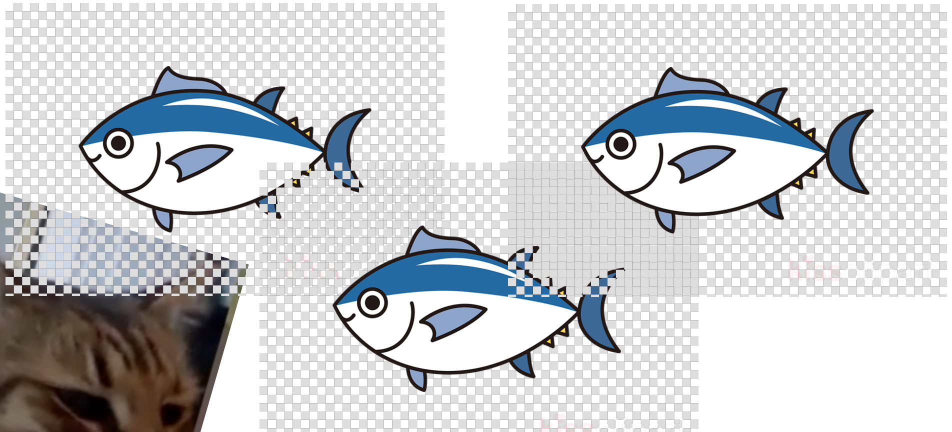
How memory is used and moves around

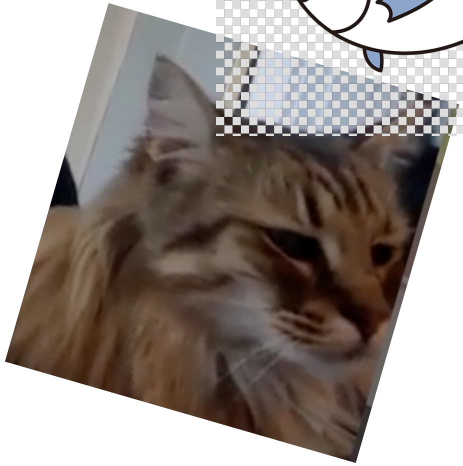

shutterstock.com • 298428176




SAFEWAY

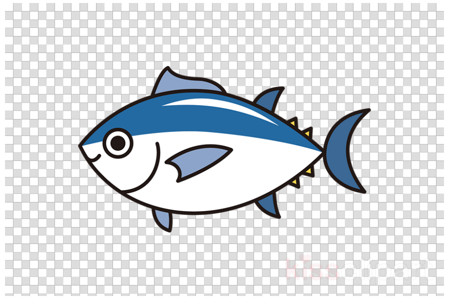

shutterstock.com • 298428176

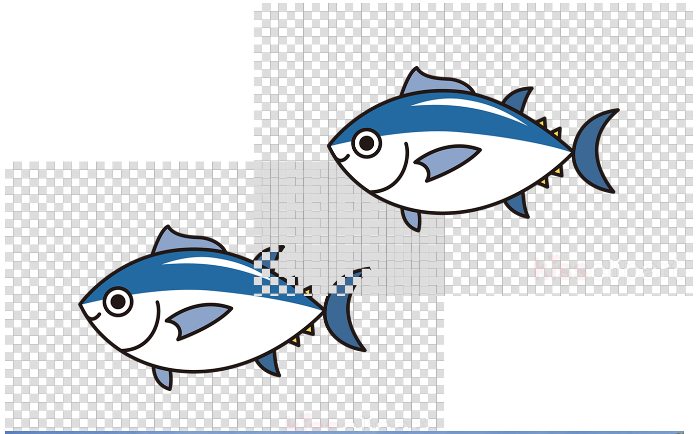





SAFEWAY

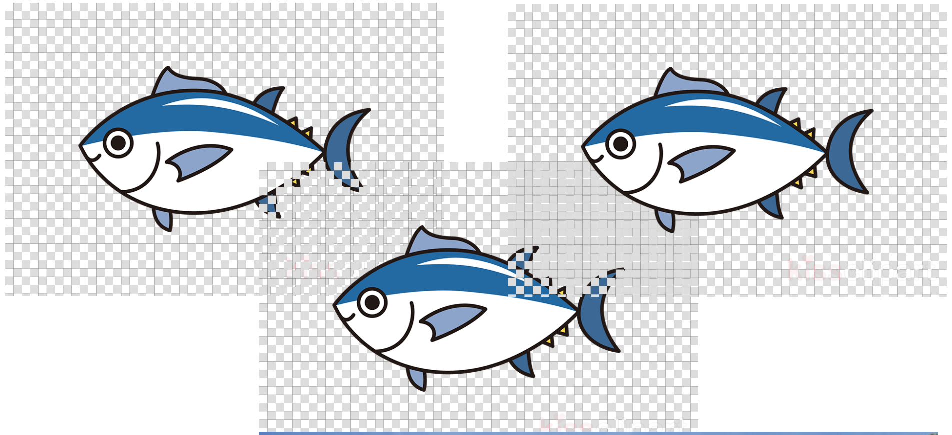

shutterstock.com • 298428176





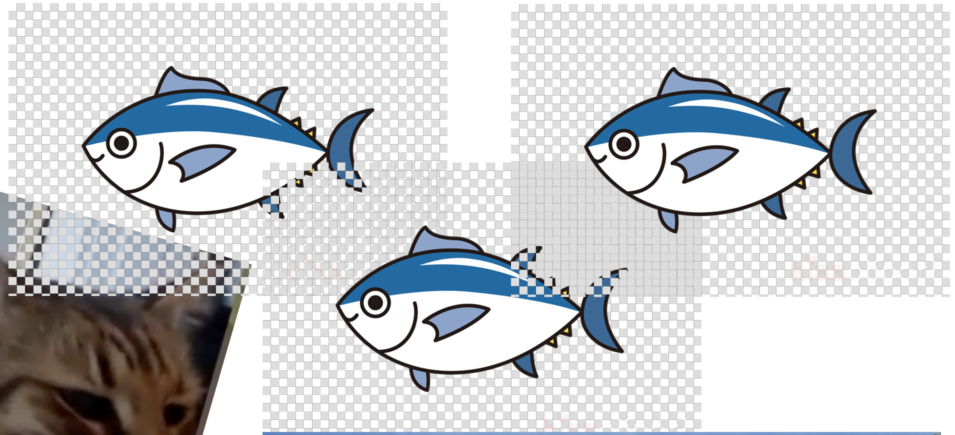

SAFEWAY

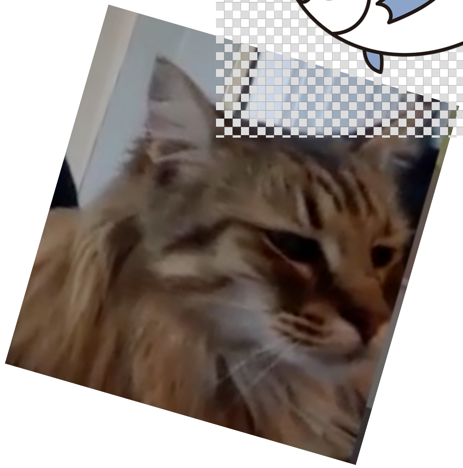

shutterstock.com • 298428176




SAFEWAY

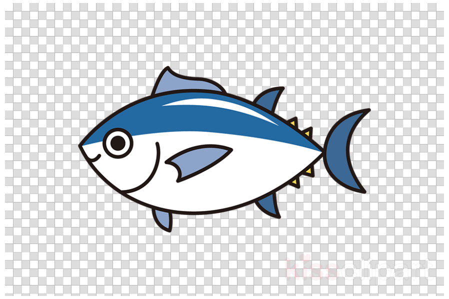

shutterstock.com • 298428176

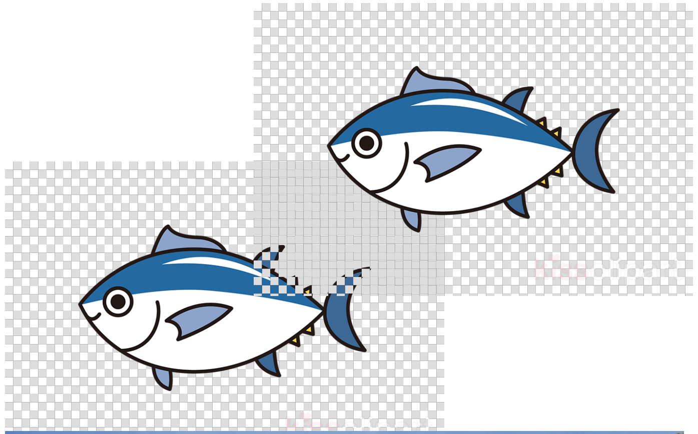





SAFEWAY

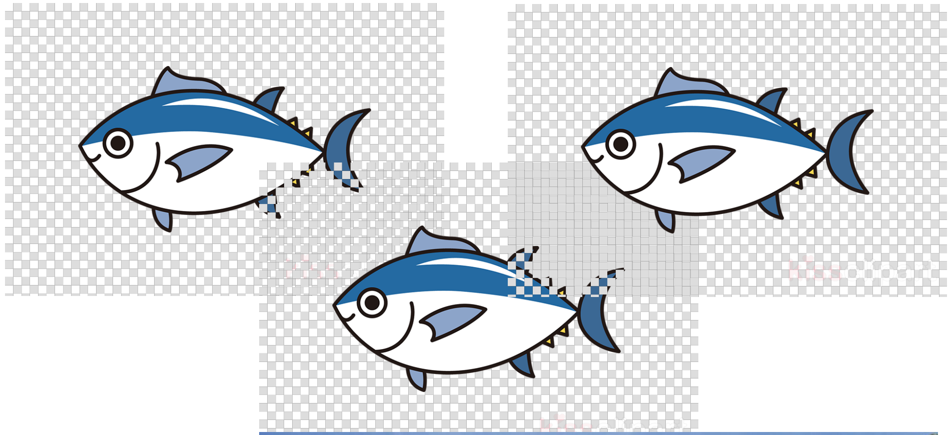

shutterstock.com • 298428176





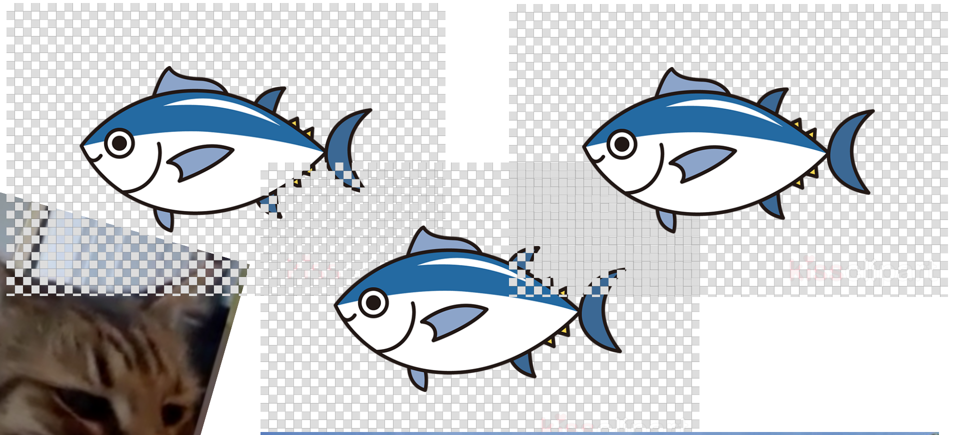

SAFEWAY

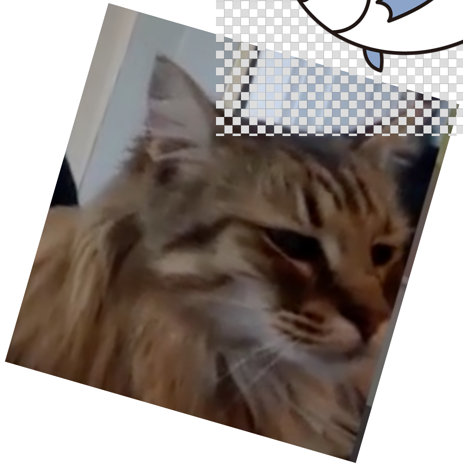

shutterstock.com • 298428176




SAFEWAY

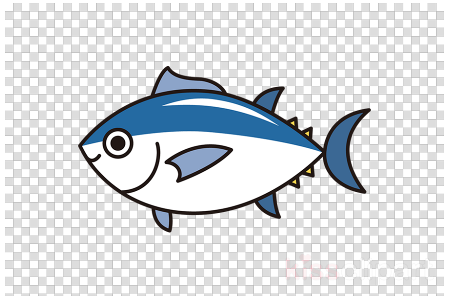

shutterstock.com • 298428176

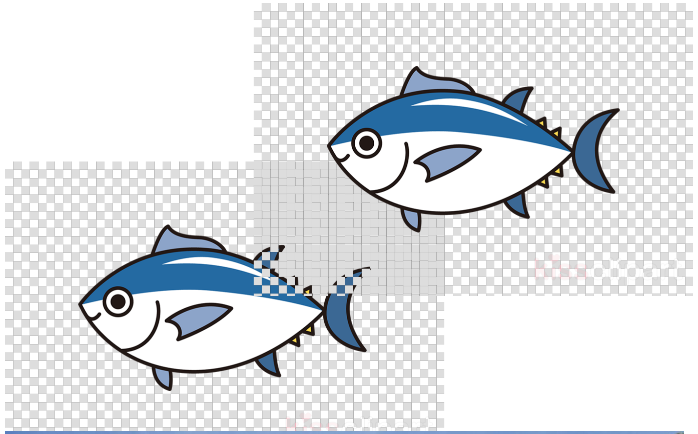





SAFEWAY


shutterstock.com • 298428176





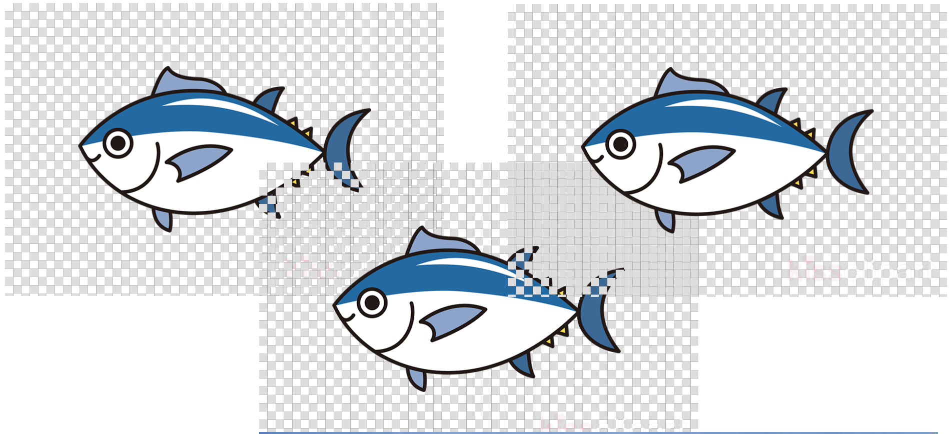

SAFEWAY

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?

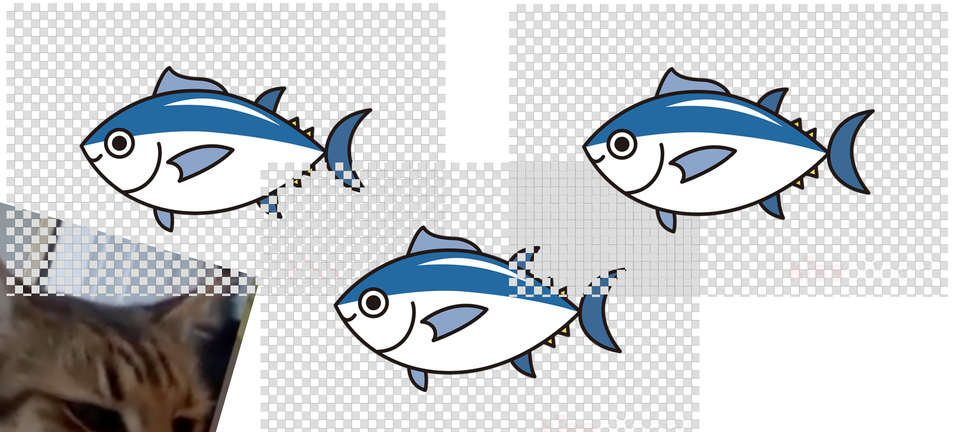
Let’s get Mercy a refrigerator!

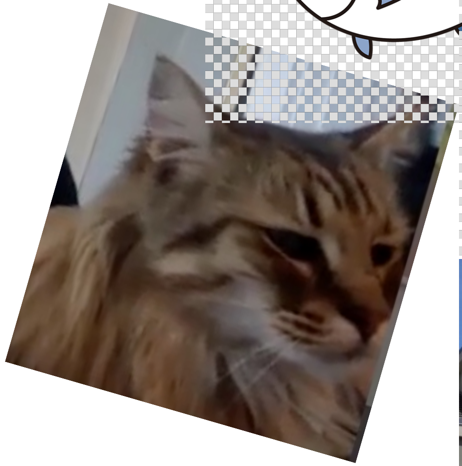

shutterstock.com • 298428176



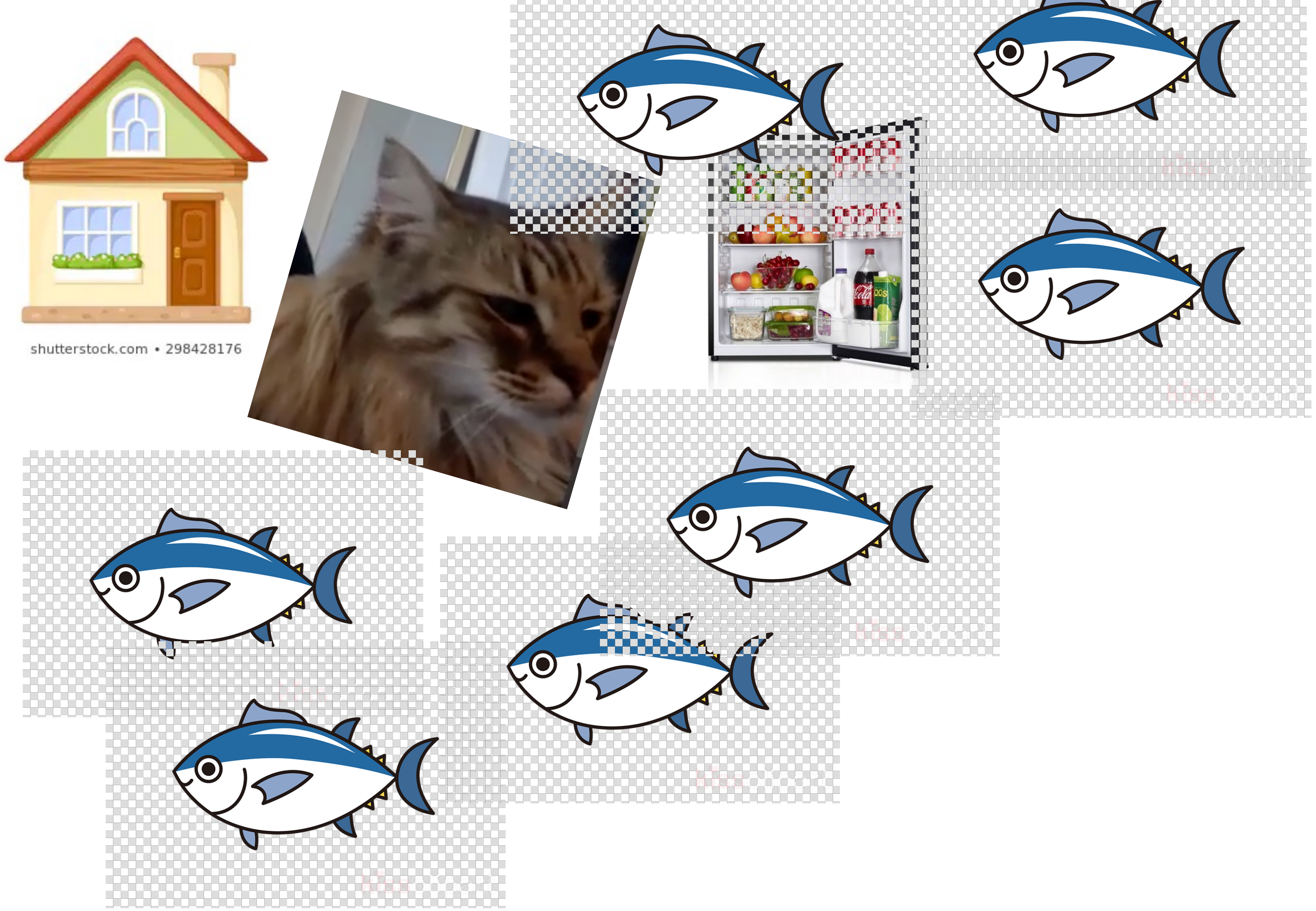

SAFEWAY

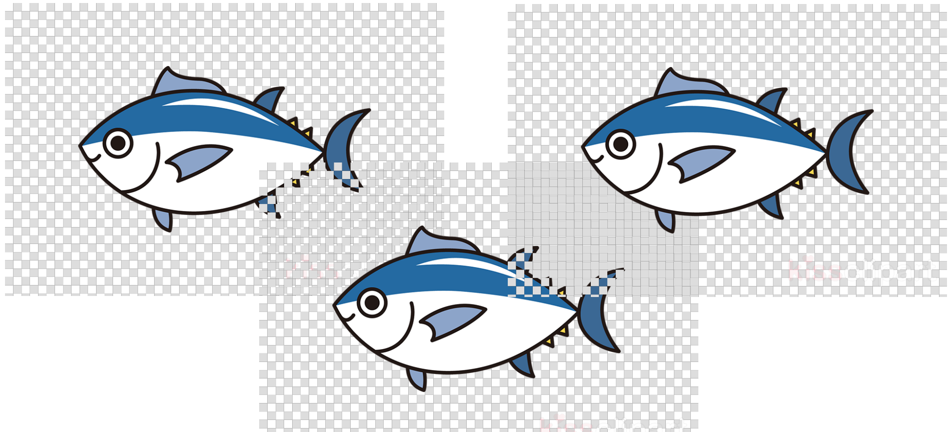

shutterstock.com • 298428176





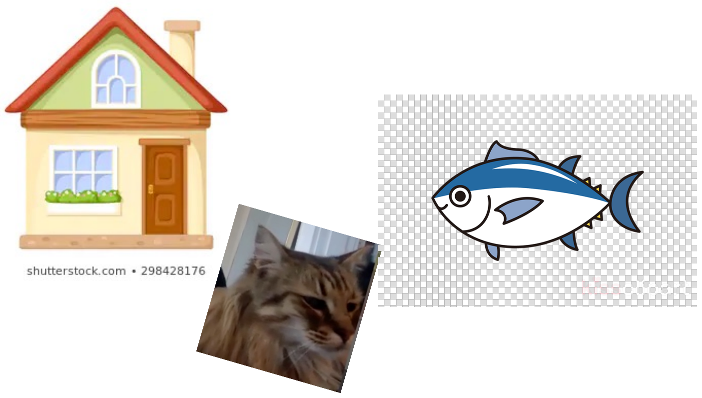

SAFEWAY


shutterstock.com • 298428176




SAFEWAY

Before



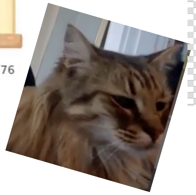
CPU

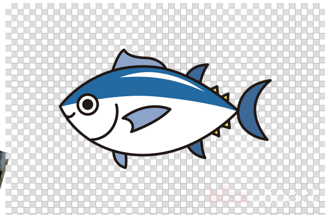
CPU – kind of like the home / brain of your computer. Pretty much all computation is done here and data needs to move here to do anything significant with it (math, if checks, normal statement execution).

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

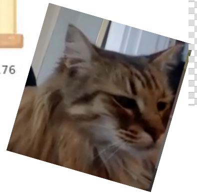
Cache

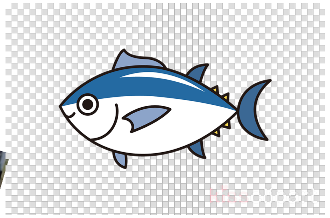
- Rough definition: a place to store some memory that's smaller and closer to the CPU compared to RAM. Because caches are closer to the CPU (where your data generally needs to go to be computed / modified / acted on) getting data from cache to CPU is a lot quicker than from RAM to CPU. This means we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

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

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.

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

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!

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

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;
}
```

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’

Java and Memory

What happens when you use the "**new**" keyword in Java?

- Your program asks the Java Virtual Machine for more memory from the "heap"
  - Pile of recently used memory
- If necessary the JVM asks Operating System for more memory
  - Hardware can only allocate in units of page
  - If you want 100 bytes you get 4kb
  - Each page is contiguous

What happens when you create a new array?

- Program asks JVM for one long, contiguous chunk of memory

What happens when you create a new object?

- Program asks the JVM for any random place in memory

What happens when you read an array index?

- Program asks JVM for the address, JVM hands off to OS
- OS checks the L1 caches, the L2 caches then RAM then disk to find it
- If data is found, OS loads it into caches to speed up future lookups

What happens when we open and read data from a file?

- Files are always stored on disk, must make a disk access

Array v Linked List

Is iterating over an ArrayList faster than iterating over a LinkedList?

Answer:

LinkedList nodes can be stored in memory, which means the don't have spatial locality. The ArrayList is more likely to be stored in contiguous regions of memory, so it should be quicker to access based on how the OS will load the data into our different memory layers.