



Lecture Participation Poll #18

Log onto pollev.com/cse374

Or

Text CSE374 to 22333

Lecture 18: Memory Architecture

CSE 374: Intermediate
Programming Concepts and
Tools

Administrivia

Grade round up coming on Canvas:

Everyone gets +3 late days for the quarter

HW1 Autograder is up and running

- Reminder HW1 final turn in Dec 3

HW2 & HW3 Published

- Regrades open tomorrow, due by Nov 22

HW4 due today

- Locks on Monday

- Individual assignment posting later today, will be due Friday 11/19

HW 5 releasing today

- Due 11/29

HW6 releases 11/24

- Due 12/16

Want to talk grades? Kasey adding more Calendly's for next week

•Homework: 65%

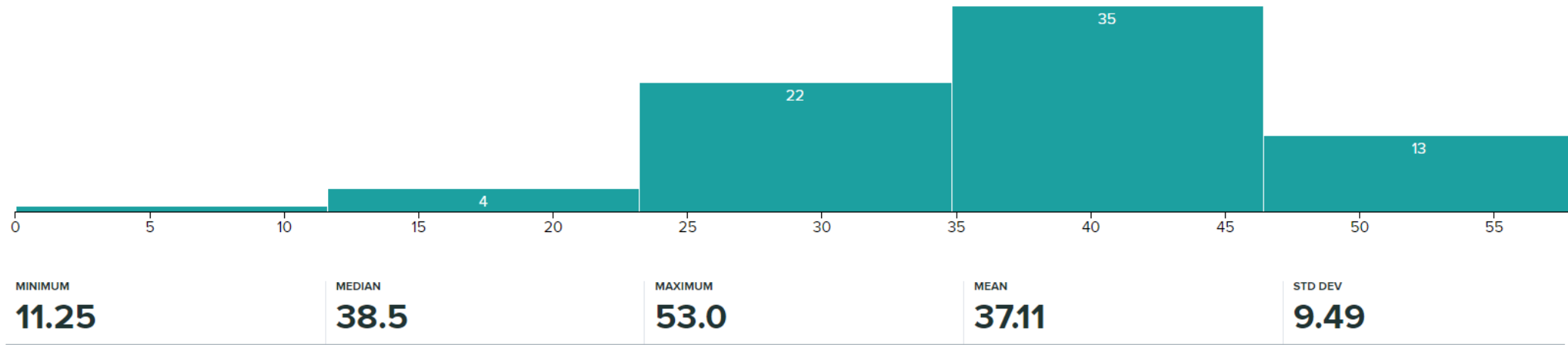
- HWs 1-4 & 6: 10% each
- HW 5: 15%

•Individual HW Assessments: 10%

•Midterm Exam: 10%

•Final Exam: 15%

Midterm



- Everyone gets +5 to their score
 - Max score is 58
 - Median is 43.5 or 75%
- Final Exam
 - Will include a pointer mystery similar to midterm style that was removed
 - Will be written as a 1 hour exam with 2 hours of work time

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 for 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;
}
```

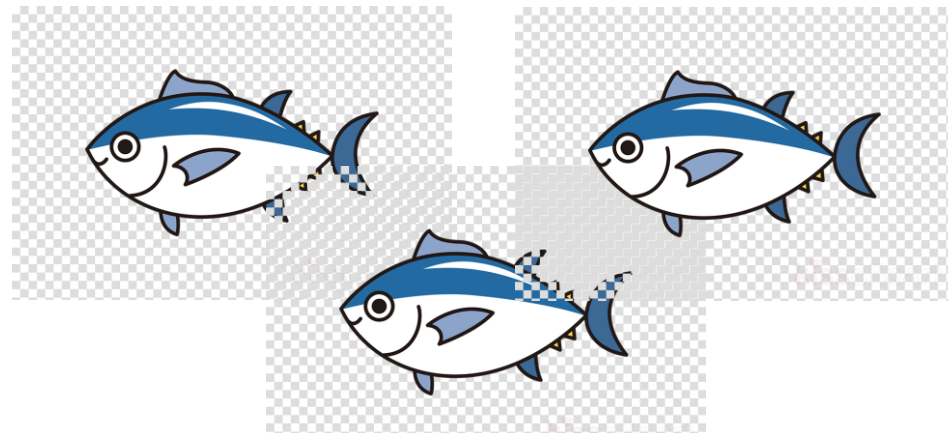
Why does sum1 run so much faster than sum2?
sum1 takes advantage of spatial and temporal locality

0	1	2	3	4																														
<table border="1"><thead><tr><th>0</th><th>1</th><th>2</th></tr></thead><tbody><tr><td>'a'</td><td>'b'</td><td>'c'</td></tr></tbody></table>	0	1	2	'a'	'b'	'c'	<table border="1"><thead><tr><th>0</th><th>1</th><th>2</th></tr></thead><tbody><tr><td>'d'</td><td>'e'</td><td>'f'</td></tr></tbody></table>	0	1	2	'd'	'e'	'f'	<table border="1"><thead><tr><th>0</th><th>1</th><th>2</th></tr></thead><tbody><tr><td>'g'</td><td>'h'</td><td>'i'</td></tr></tbody></table>	0	1	2	'g'	'h'	'i'	<table border="1"><thead><tr><th>0</th><th>1</th><th>2</th></tr></thead><tbody><tr><td>'j'</td><td>'k'</td><td>'l'</td></tr></tbody></table>	0	1	2	'j'	'k'	'l'	<table border="1"><thead><tr><th>0</th><th>1</th><th>2</th></tr></thead><tbody><tr><td>'m'</td><td>'n'</td><td>'o'</td></tr></tbody></table>	0	1	2	'm'	'n'	'o'
0	1	2																																
'a'	'b'	'c'																																
0	1	2																																
'd'	'e'	'f'																																
0	1	2																																
'g'	'h'	'i'																																
0	1	2																																
'j'	'k'	'l'																																
0	1	2																																
'm'	'n'	'o'																																

How memory is used and moves around



shutterstock.com • 298428176



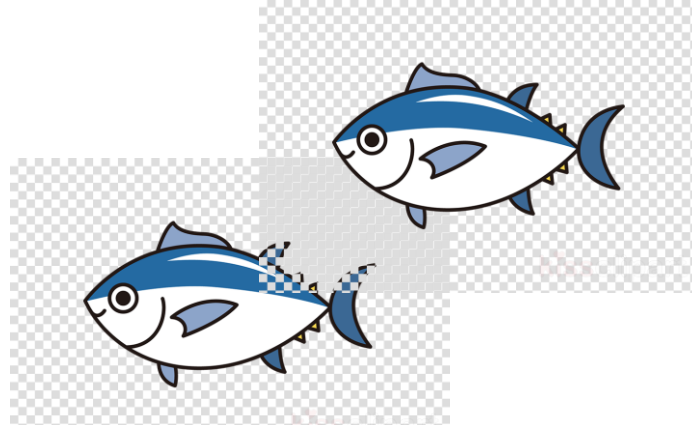
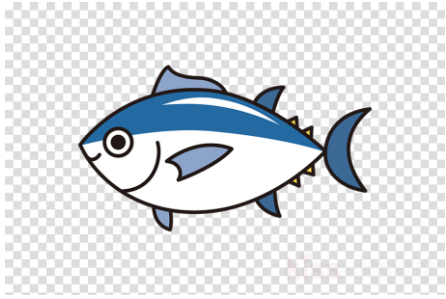


shutterstock.com • 298428176



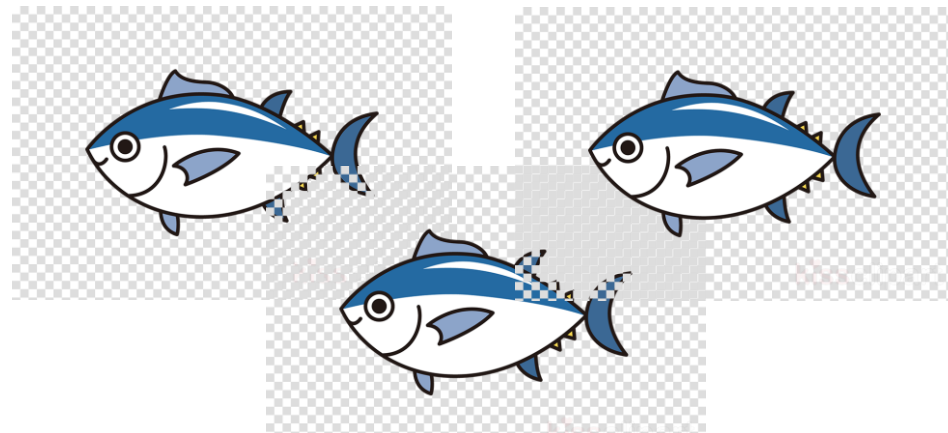


shutterstock.com • 298428176





shutterstock.com • 298428176



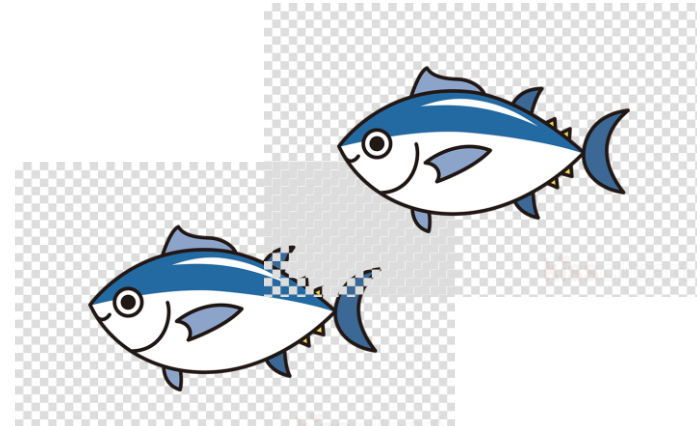
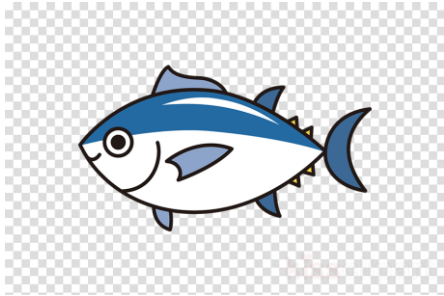


shutterstock.com • 298428176



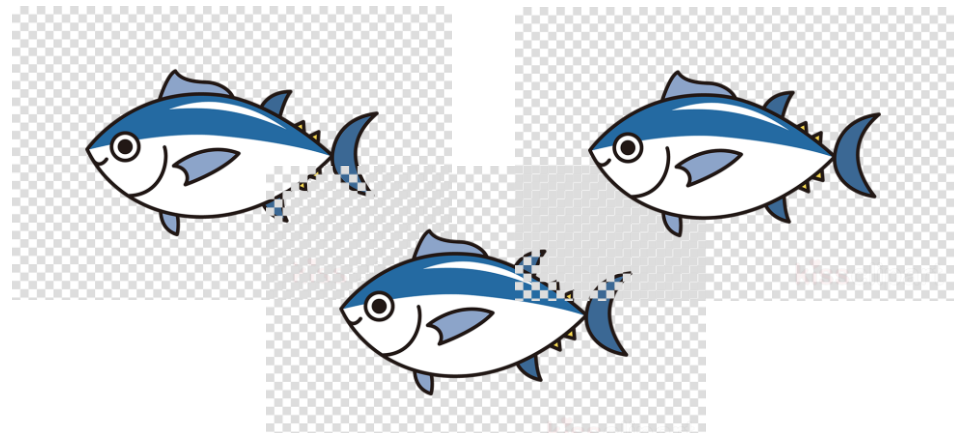


shutterstock.com • 298428176





shutterstock.com • 298428176



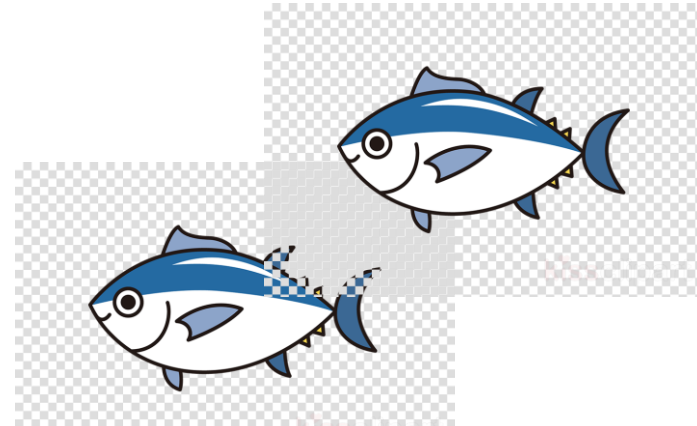
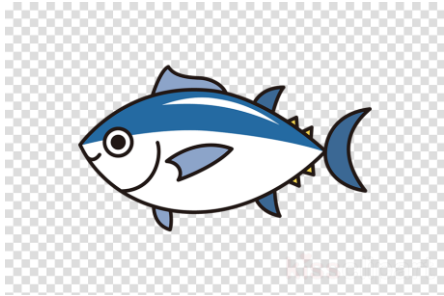


shutterstock.com • 298428176



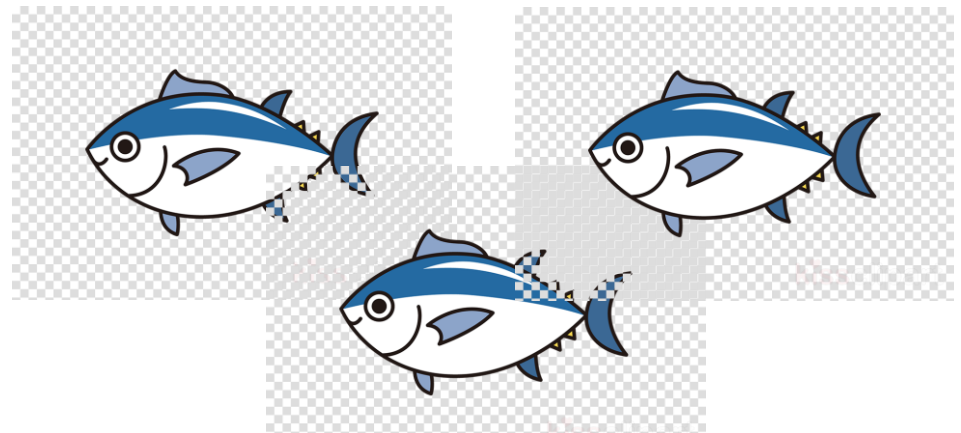


shutterstock.com • 298428176





shutterstock.com • 298428176



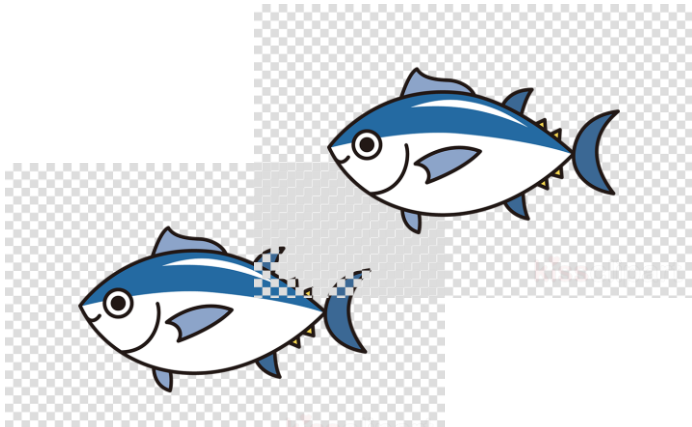
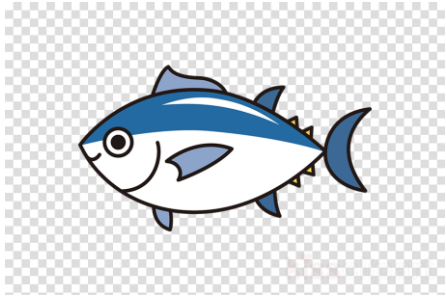


shutterstock.com • 298428176



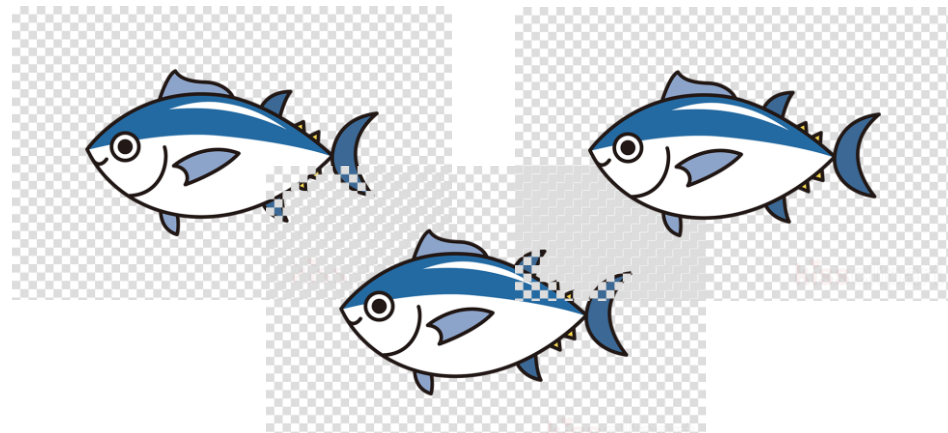


shutterstock.com • 298428176





shutterstock.com • 298428176



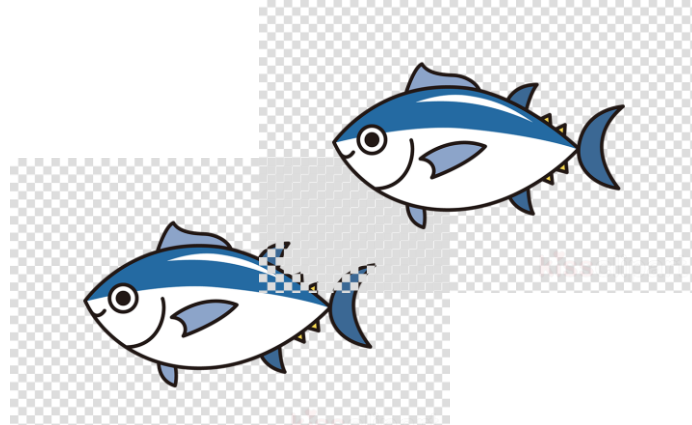
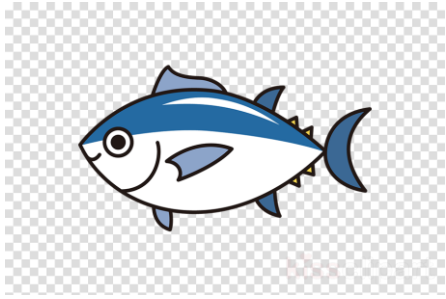


shutterstock.com • 298428176



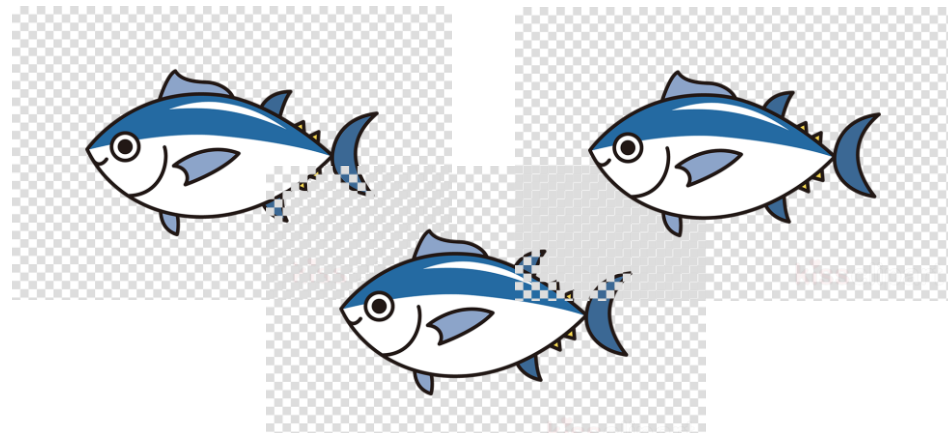


shutterstock.com • 298428176





shutterstock.com • 298428176

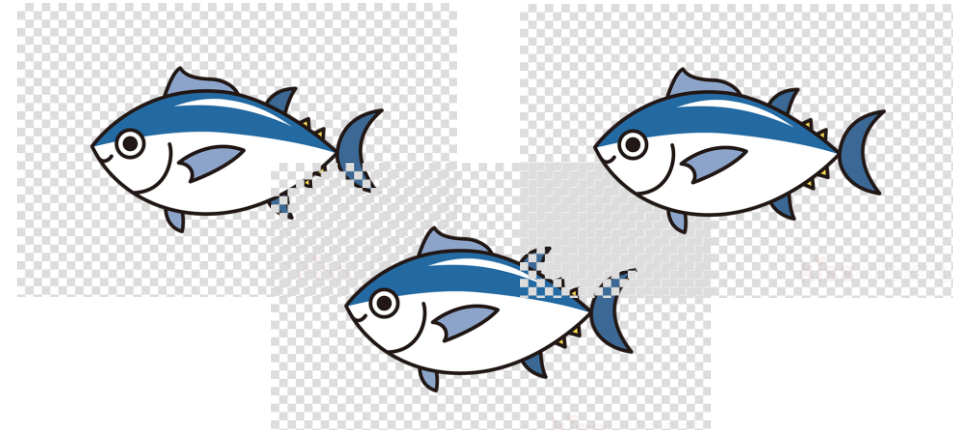


Solution to Mercy's traveling problem

- 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





shutterstock.com • 298428176





shutterstock.com • 298428176

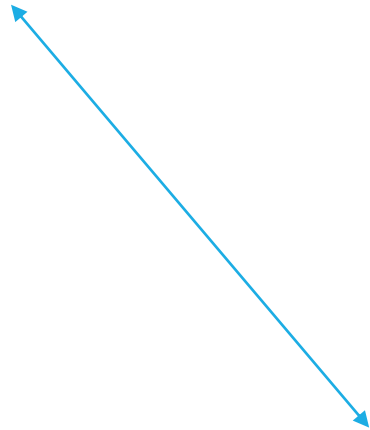


Recap + connecting analogy back to computer

Before CPU



shutterstock.com • 298428176



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



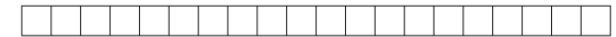
After CPU



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 CPU



Cache!

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

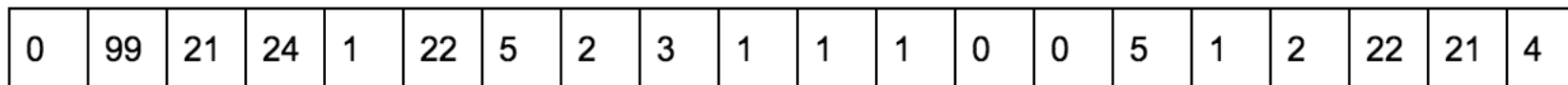
RAM



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

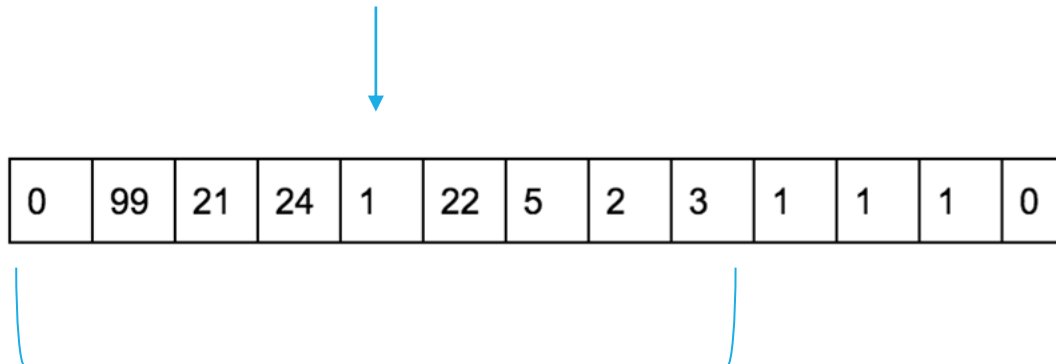
0	99	21	24	1	22	5	2	3	1	1	1	0	0	5	1	2	22	21	4
---	----	----	----	---	----	---	---	---	---	---	---	---	---	---	---	---	----	----	---



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.

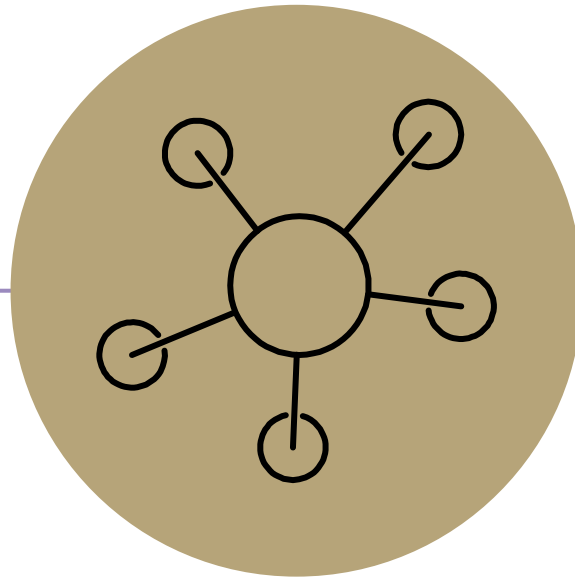
Imagine that the below memory is just an entire array of length 13, with some data in it.



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