Computer Memory

Data Structures and Algorithms

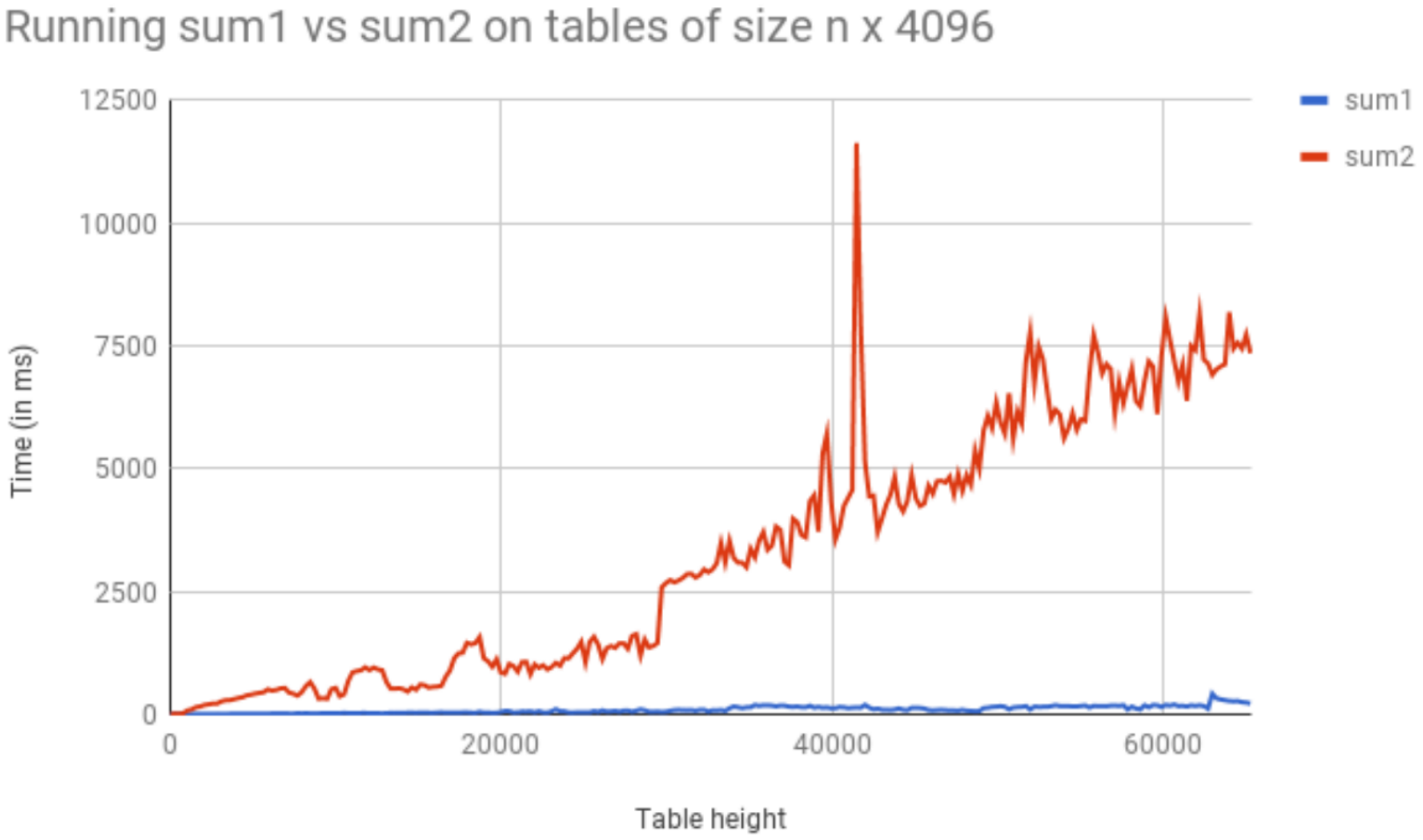
Warm Up

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

What do these two methods do?
What is the big-Θ
Θ(n*m)

Warm Up


Running sum1 vs sum2 on tables of size n x 4096
12500
10000
7500
5000
2500
0
Time (in ms)
0
20000
40000
60000
Table height
sum1
sum2

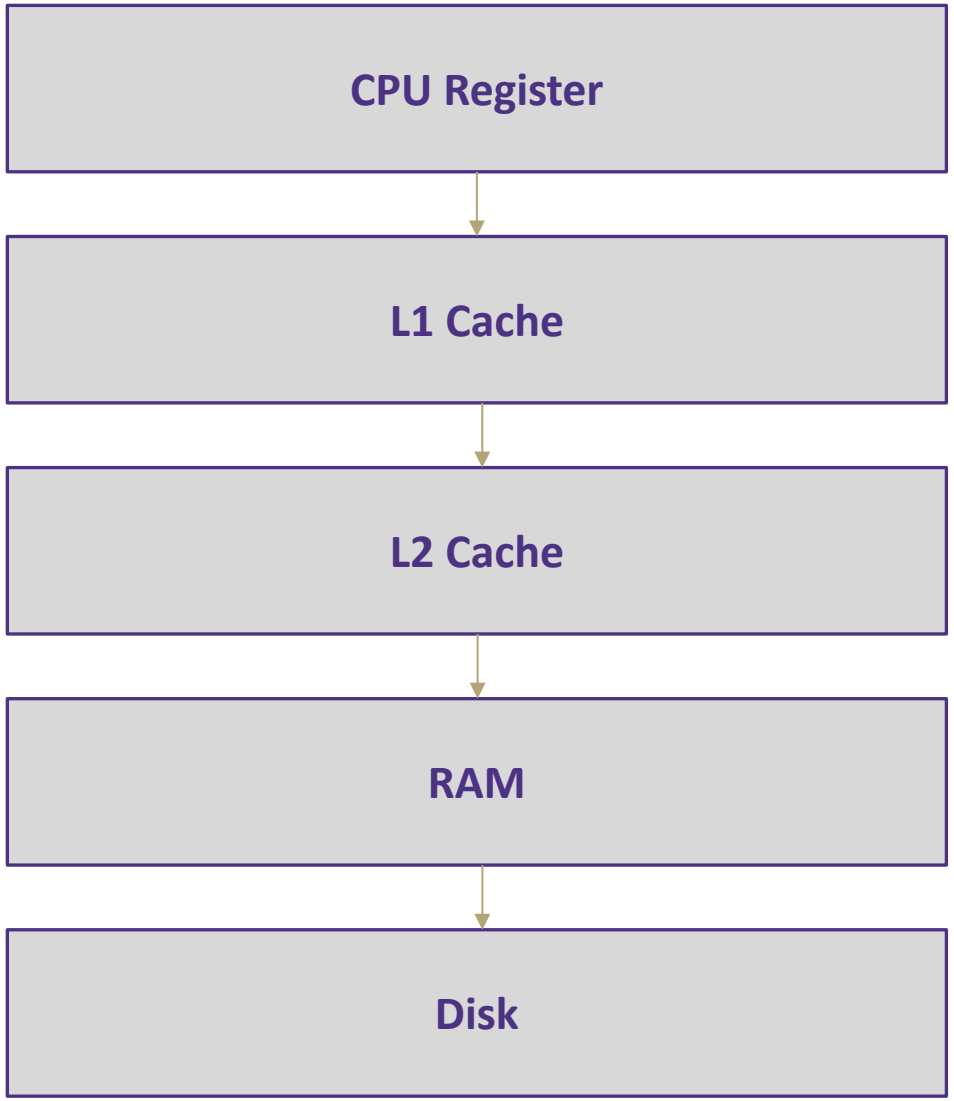
Incorrect Assumptions

Accessing memory is a quick and constant-time operation

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

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

Warm Up

```
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 < m; i++) {
        for (int j = 0; j < n; 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 **J**ava **V**irtual **M**achine 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.