Arrays and Memory

The limitations of arrays

Arrays are useful, but they do have limitations.

Two important limitations:

- single datatype, not mixed datatypes.
- fixed size, cannot expand or shrink after creation.

These limitations (and the fact that array indexing starts from zero) make sense if we understand what is going on behind the scenes in computer memory when arrays are created.

Computer memory: bits and bytes

Bit – Short for "binary digit," i.e. 1 or 0. The smallest meaningful unit of information.

Bits of information are stored in memory cells inside of your computer in a memory array.

A 1 stands for high electric potential and a 0 stands for low electric potential in a capacitor.

0	1	0	1	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	0	0
1	0	0	1	0	1	0	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	1	1
1	1	0	1	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1	0	0	0	0	0
1	1	0	1	0	0	1	1	1	1	0	0	0	1	1	0	1	1	1	0	0	0	0	0
0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	1
0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	0	1	1	1
1	0	1	1	0	0	0	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0

Computer memory: bits and bytes

Byte – A grouping of 8 contiguous bits. This is the smallest grouping of memory that the computer assigns an address.

A byte can have $2^8 = 256$ different values… enough to encode small pieces of information like shades of red, green, or blue for a color value or a basic character like 'A' or '?'.

0	1	0	1	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	0	0
1	0	0	1	0	1	0	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	1	1
1	1	0	1	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1	0	0	0	0	0
1	1	0	1	0	0	1	1	1	1	0	0	0	1	1	0	1	1	1	0	0	0	0	0
0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	1
0	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	0	1	1	1
0	1	1	0	1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	0	1	1	1
1	0	1	1	0	0	0	1	1	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0

Datatypes occupy a fixed number of bytes

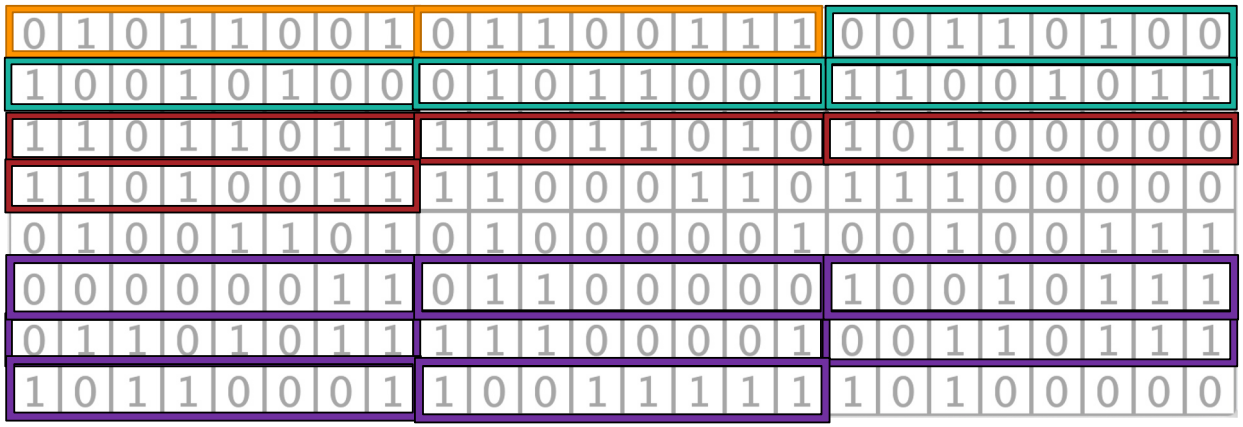
Each primitive datatype has a different width in bytes.

char — 2 bytes

int — 4 bytes

float — 4 bytes

double — 8 bytes



Arrays are references to blocks of reserved memory

When you declare an array, Java asks your operating system to find an unused block of bytes large enough to store your data.

The OS gives you back a **reference** to the memory address where the block of bytes are located. In BlueJ, these references are represented as arrows that "point" to the values in memory.

```
int[] myNumbers = new int[5];
```

Arrays are references to blocks of reserved memory

```
int[] myNumbers = new int[5];
```

In fact, if you try to print `myNumbers`, you get some strange stuff that looks nothing like an array:

```
System.out.println(myNumbers);
```

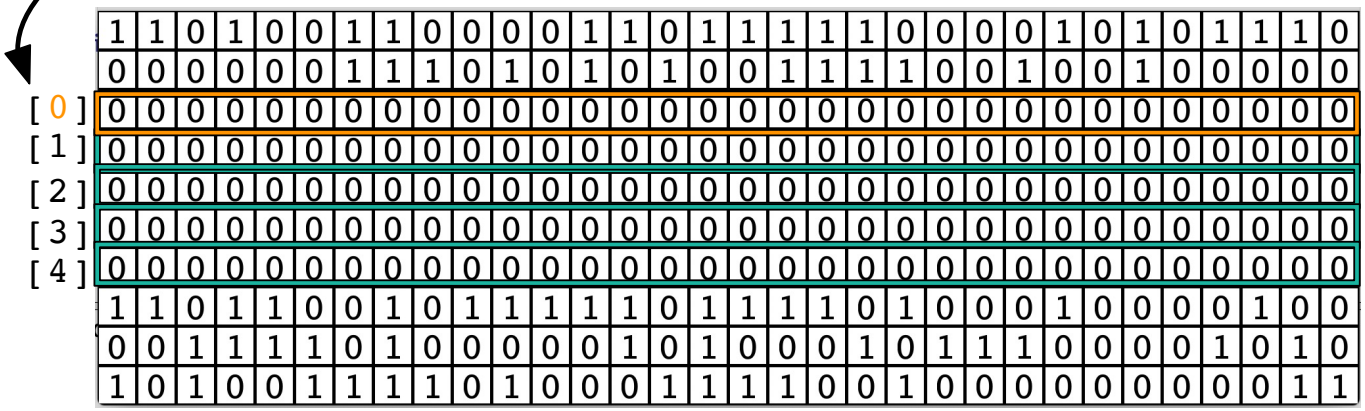
Output to console:

```
I@5d548e7a
```

This is the memory reference!

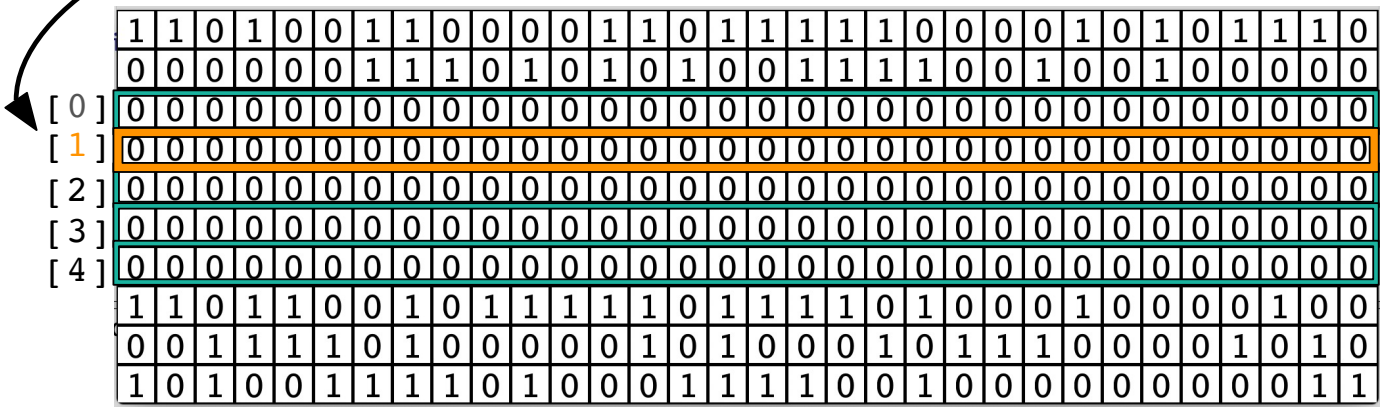
Arrays are references to blocks of reserved memory

```
int[] myNumbers = new int[5];
myNumbers[0] stored at (I@5d548e7a + 0 * 4 bytes)
```



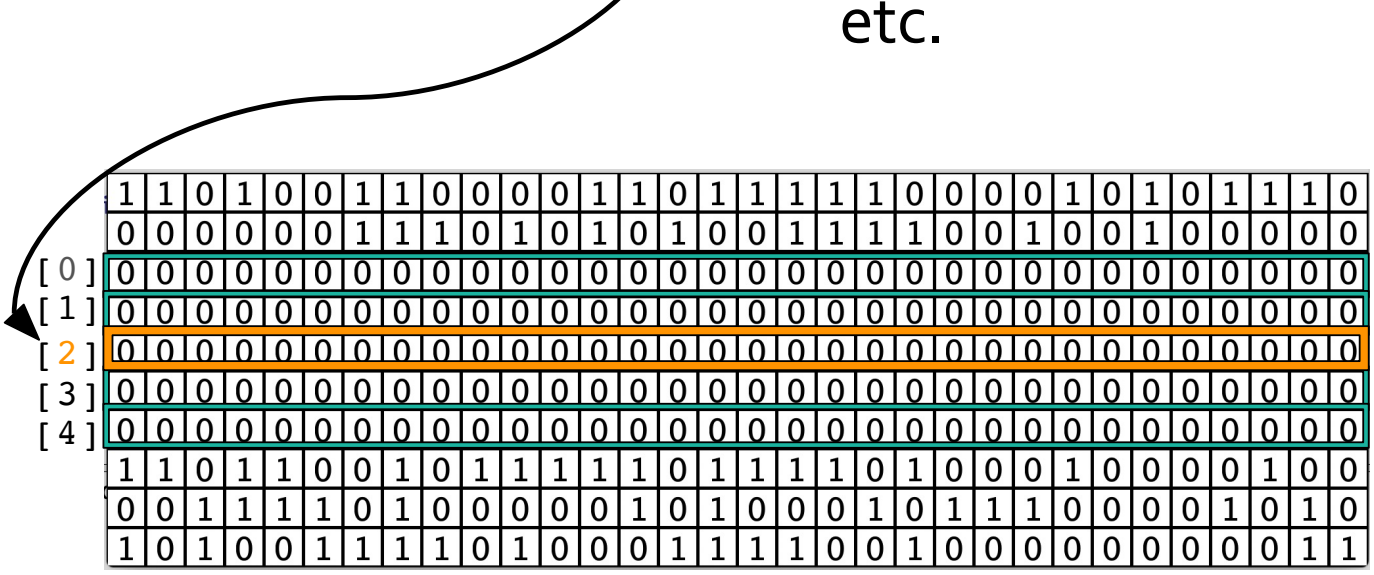
Arrays are references to blocks of reserved memory

```
int[] myNumbers = new int[5];
myNumbers[1] stored at (I@5d548e7a + 1 * 4 bytes)
```



Arrays are references to blocks of reserved memory

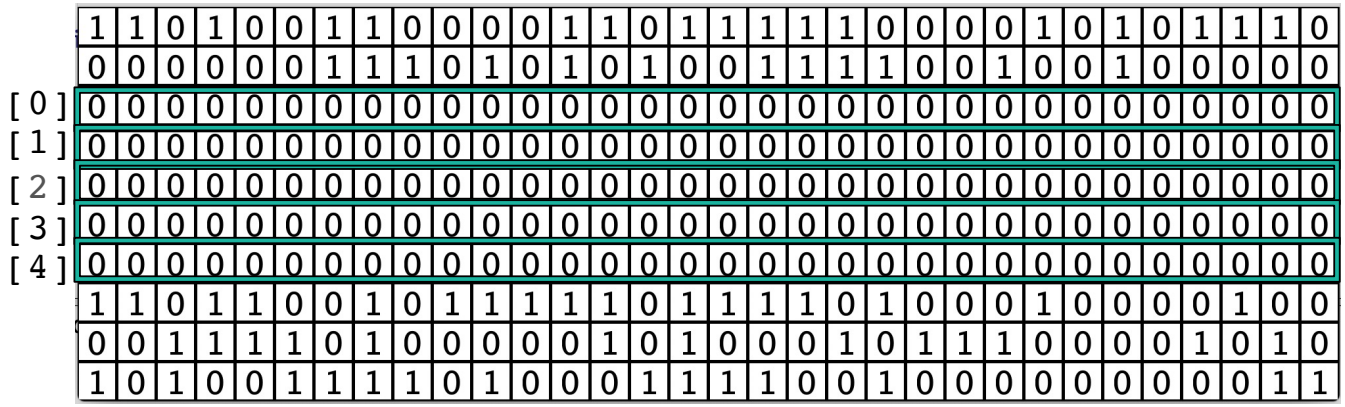
```
int[] myNumbers = new int[5];
myNumbers[2] stored at (I@5d548e7a + 2 * 4 bytes)
```



Arrays are references to blocks of reserved memory

```
int[] myNumbers = new int[5];
myNumbers[i] stored at (I@5d548e7a + i * 4 bytes)
```

The fact that arrays hold only *one* kind of datatype means that the computer can jump straight to the next element of the array using the index.



Arrays are references to blocks of reserved memory

```
int[] myNumbers = new int[5];
myNumbers[i] stored at (I@5d548e7a + i * 4 bytes)
```

Arrays cannot be expanded after they are created. That would risk overwriting other data nearby!

Memory references are addresses

The value `I@5d548e7a` is itself an int that is printed as a hexadecimal (base 16) number which is an address:

```
5d548e7a = 1,565,822,586 bytes deep into RAM
```

When arrays are passed into a method, only this address is copied over.

The entire array is not copied over each time a method is called—that would be very time consuming for large arrays.

Computer RAM: Random Access Memory

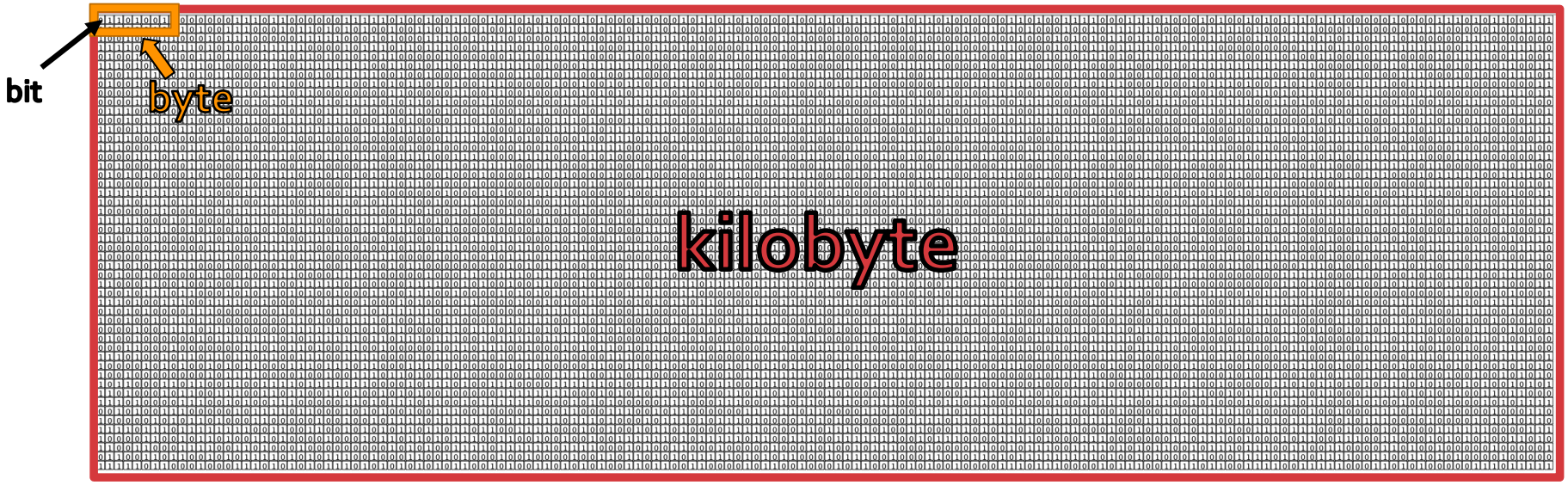
Even though the address `I@5d548e7a` represents a location over 1 billion bytes deep into RAM, the data stored there can be retrieved nearly instantly.

"Random Access" refers to the ability to jump almost instantly to a value stored anywhere in memory.

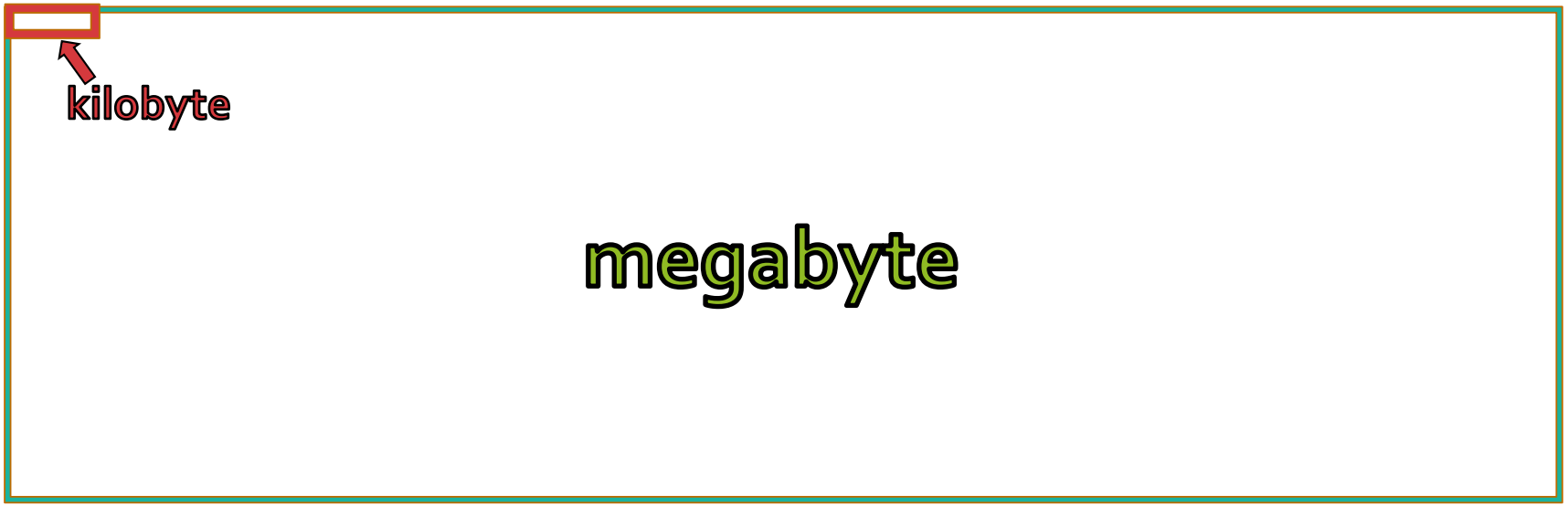
The opposite of random access is serial access, like using a cassette tape or vinyl record where it takes time to skip ahead to the part you want to access.

Computer RAM: Random Access Memory

Modern RAM is measured in Gigabytes, which is a *lot* of addressable space in memory.

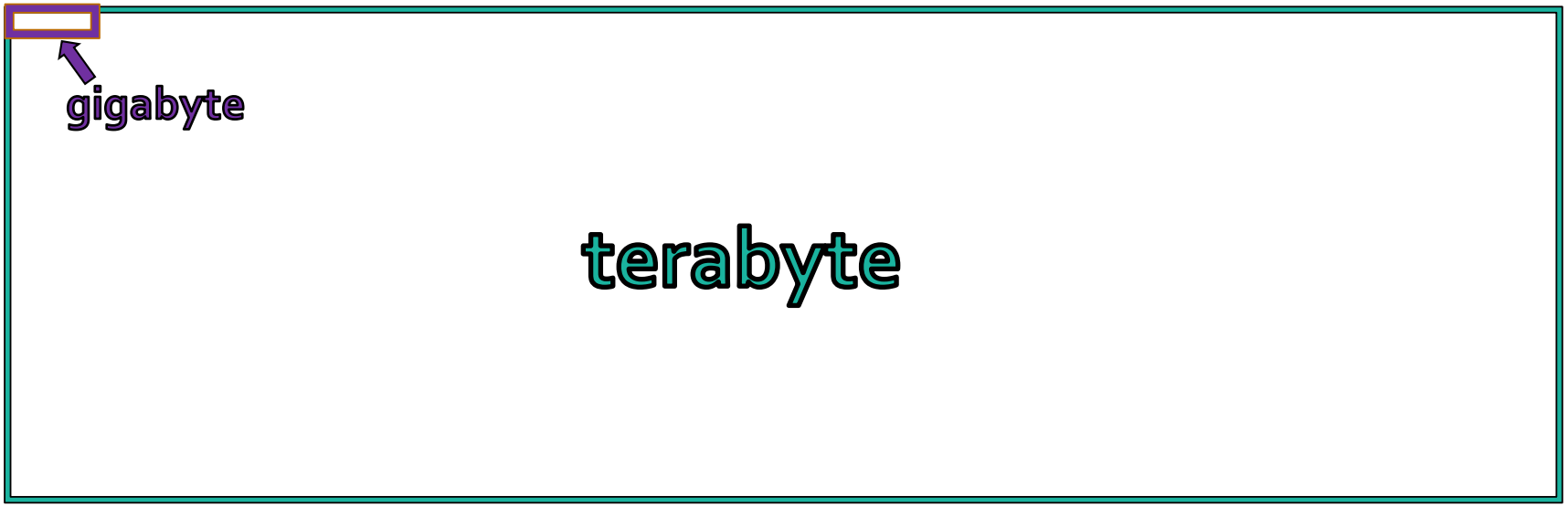


1 kilobyte = 1,000 bytes



1 megabyte = 1,000 kilobytes (1 million bytes)

1 gigabyte = 1,000 megabytes (1 billion bytes)



1 terabyte = 1,000 gigabytes (1 trillion bytes)

RAM is "short term memory"

The hard drive is "long term memory"

When your computer turns off or loses power, your RAM is reset.

RAM is only useful for doing computations and helping active programs complete their tasks.

Your photos, games, music, and documents are all stored in long term memory on the hard disk drive (HDD). The HDD is a serial access device that can store information with much greater density.

A hard disk drive