Memory Maps

Section 2.3

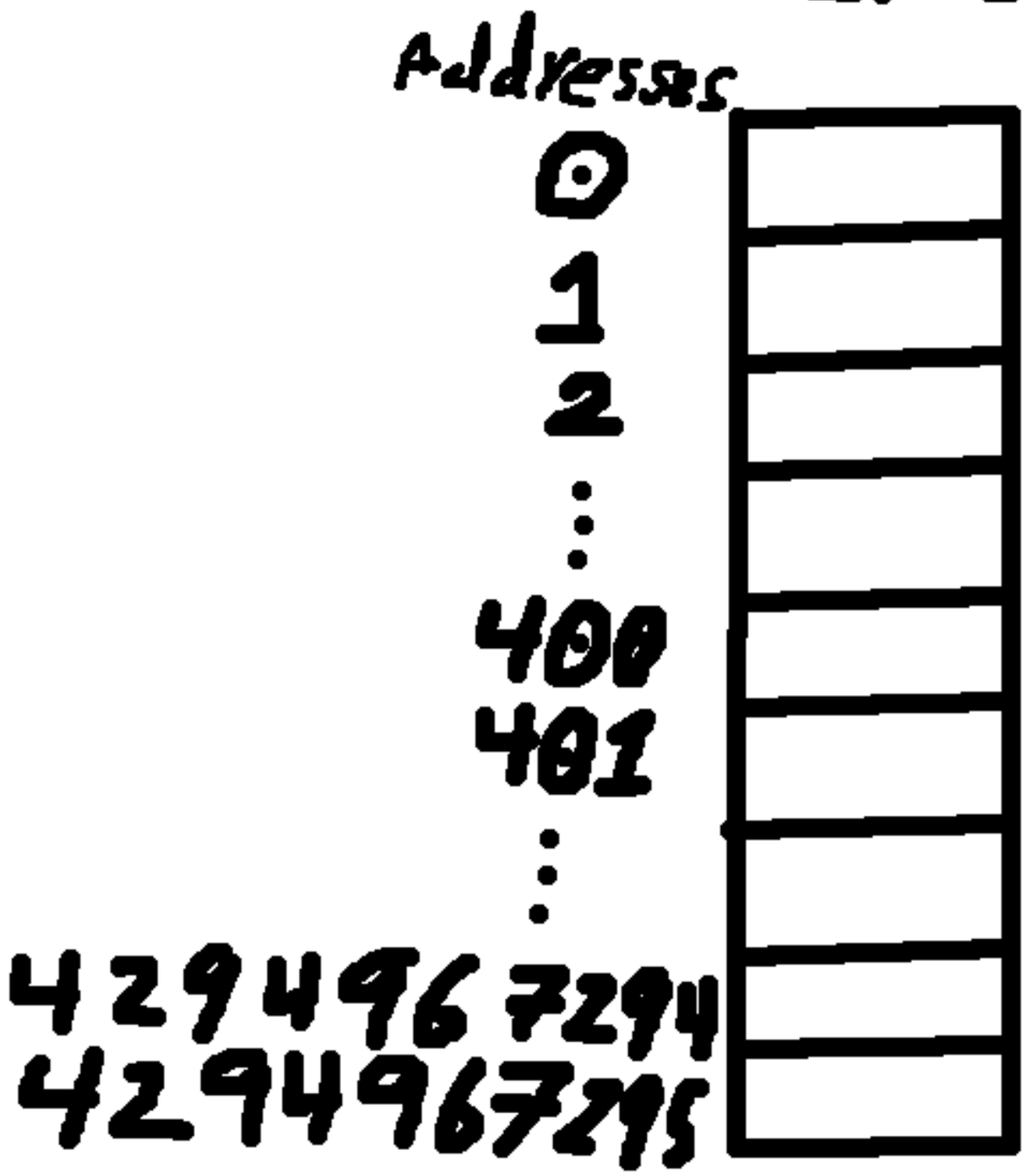
Memory

- Basic unit of memory is a **bit**
- But smallest unit computer works with is a **byte**
- Every byte of memory has an **address**
- An **address** is basically an (often larger) **integer**
  - Commonly, addresses are written in hexadecimal

- A computer's memory is essentially a very large array
- In this analogy, each element of the array is a byte
- Each index of the array is an address

- Often, portions of the computer's memory are designated for specific uses
- A variable in our program will correspond to one or more of bytes of memory

Memory



4GB =

4,294,967,296 Bytes

Memory Maps

- A **memory map** is a table that lists variables, their addresses, and their values in a program
  - A way to visualize contents of memory
- Each address is assumed to represent 1 byte
- Actual value of address will likely vary each time the program runs
  - Often just uses 400 as the starting address in examples in the textbook

```
char a;
char b;
char c;
```

```
a = 7;
b = -13;
c = 0;
```

Variable	Address	Value
a	400	7
b	401	-13
c	402	0

```
char a;
int b;
float x;
double y;
```

```
a = 7;
b = -13;
x = 0.1;
y = 42.5
```

Variable	Address	Value
a	400	7
b	401 – 404	-13
x	405 – 408	0.1
y	409 – 416	42.5

- Memory maps can include bit values as well
- Usually not necessary and thus omitted

```
char a;
short int b;
char c;
```

```
a = 6;
b = 13;
c = '6';
```

Variable	Address	Bits	Value
a	400	0000 0110	6
b	401 – 402	0000 0000 0000 1101	13
c	403	0011 0110	'6'

- In Java, variables have default values
  - 0 for numerical types
- In C, variables do not have default values
  - Practically a random value, whatever the bits in that memory were last set to

```
int i;
int sum;
printf("%d\n", sum);

for (i = 1; i <= 10; i++)
{
    if (i % 2 == 0)
    {
        sum = sum + i;
    }
}

printf("%d\n", sum);
```

Variable	Address	Value
i	400 – 403	~~1 2 3 4 5 6 7 8 9 10~~ 11
sum	404 – 407	?