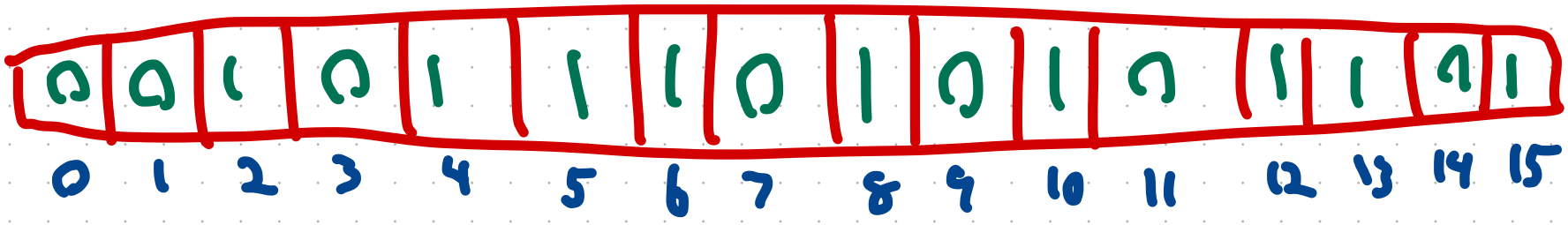
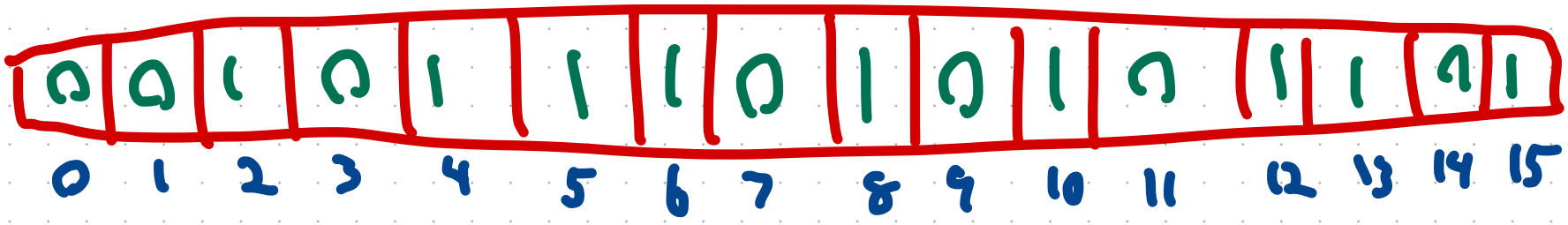
# Bit Arrays

Suppose You Want a Array/List of 0/1 values



# Suppose You Want a Array/List of 0/1 values




For example, you want to store a set of small positive integers

{2, 4, 5, 6, 8, 10, 12, 13, 15}

# What about a List?

# What about a List?

Lets make a list to store 1000 values:

# What about a List?

Lets make a list to store 1000 values:

```
L = [0 for i in range 1000]
```

(or `L = [0]*1000`)

# What about a List?

Lets make a list to store 1000 values:

```
L = [0 for i in range 1000]
```

(or `L = [0]*1000`)

How big is L?

# What about a List?

Lets make a list to store 1000 values:

```
L = [0 for i in range 1000]
```

(or `L = [0]*1000`)

How big is L?

16,056 Bytes = 128,448 Bits

# What about a List?

Lets make a list to store 1000 values:

```
L = [0 for i in range 1000]
```
(or `L = [0]*1000`)

How big is L?

16,056 Bytes = 128,448 Bits

To store 1000 Bits!!

# What about a List?

Lets make a list to store 1000 values:

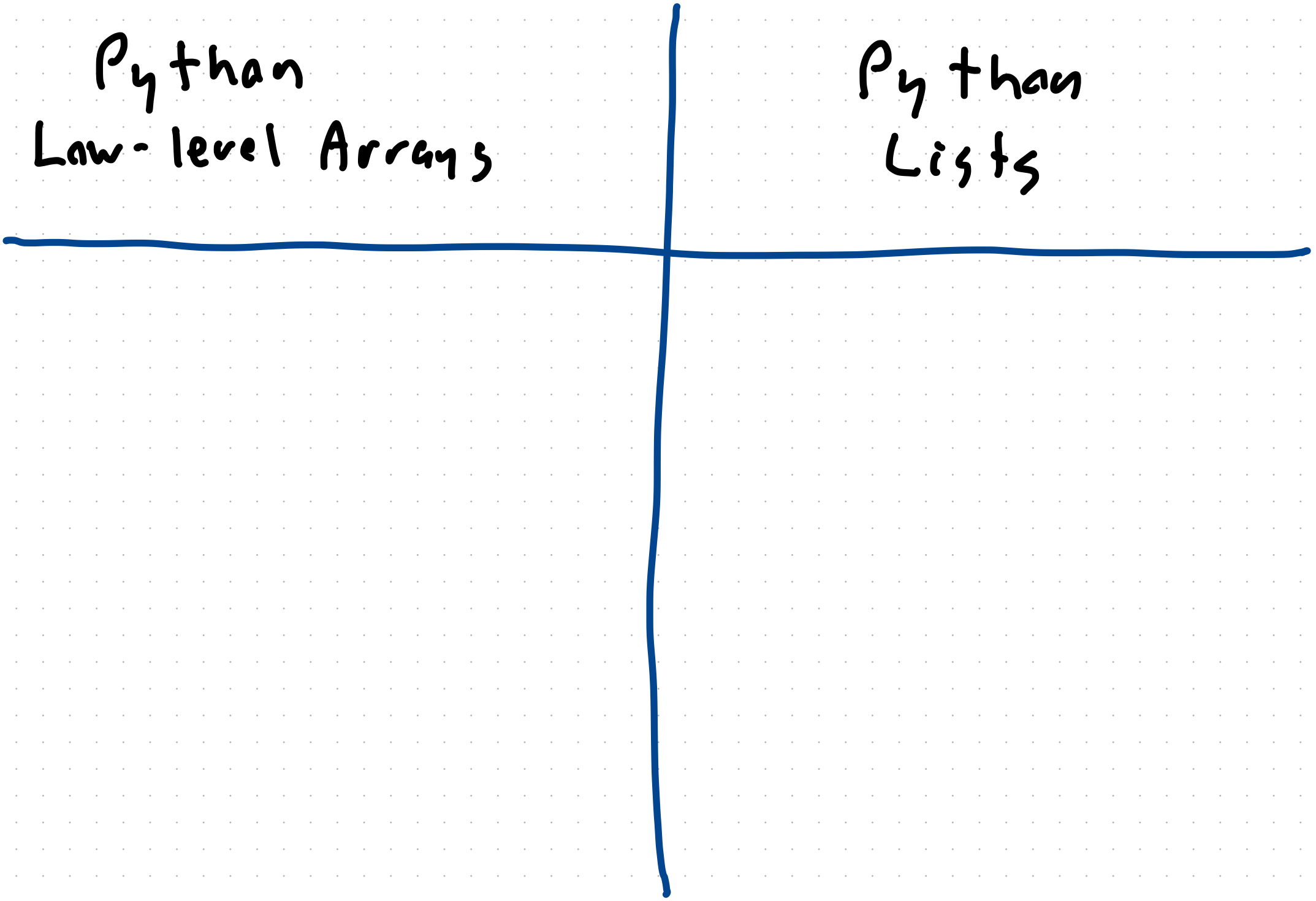
```
L = [0 for i in range 1000]
```
(or `L = [0]*1000`)

How big is L?

16,056 Bytes = 128,448 Bits

To store 1000 Bits!!

Can we do better?


Python
Low-level Arrays
Python
Lists

## Python Low-level Arrays

Can only hold one kind of data which must be specified in advance (e.g. byte, float)

## Python Lists

Can hold anything

## Python Low-level Arrays

Can only hold one kind of data which must be specified in advance (e.g. byte, float)

Compact

## Python Lists

Can hold anything

Big

## Python Low-level Arrays

Can only hold one kind of data which must be specified in advance (e.g. byte, float)

Compact

Fast

## Python Lists

Can hold anything

Big

Slower

## Python Low-level Arrays

Can only hold one kind of data which must be specified in advance (e.g. byte, float)

Compact

Fast

Limited Features

## Python Lists

Can hold anything

Big

Slower

Can do many things

# Creating a low-level Array of Bytes

```
A = array.array("B", initial values)
```

# Creating a low-level Array of Bytes

```
A = array.array("B", initial values)
```

↑ Code that means Byte - 8 bits

i.e. 0..255

# Bitarray

It should work like this:

# Bitarray

It should work like this:

```
B = BitArray(10)
```

size

# Bit array

It should work like this:

```
B = BitArray(10)
B[3] = 1
```

(10) ← size

# Bit array

It should work like this:

```
B = BitArray(10)
```

size

```
B[3] = 1
B[7] = 1
```

# Bitarray

It should work like this:

```
B = BitArray(10)
B[3] = 1
B[7] = 1
print(B[7], B[8])
```

size

1 0

# Bitarray

It should work like this:

```
B = BitArray(10)
B[3]=1
B[7]=1
print(B[7], B[8])
    1   0
print(str(B))
    0001000100
```

size

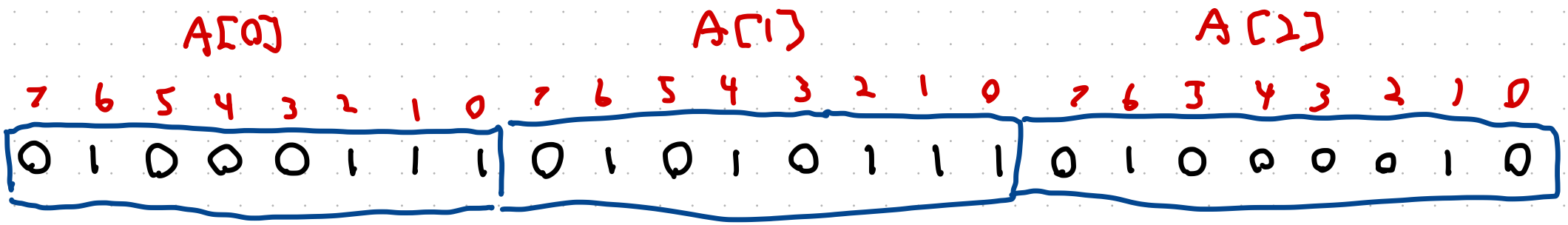
# Bytes to Bits

The smallest array data type supported is a byte = 8 bits.

# Bytes to Bits

The smallest array data type supported is a byte = 8 bits.

We need to do a little work to make a class where we can read/write individual bits.

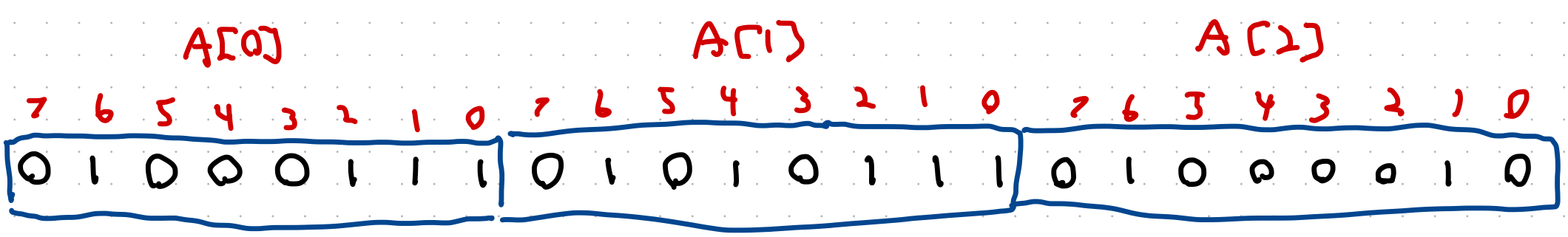



Where would we keep the 22^ND Bit?

In the 6^TH Bit of the 2^ND Byte

22 mod 8

$\lfloor \frac{22}{8} \rfloor$




Where would we keep the 22^ND Bit?

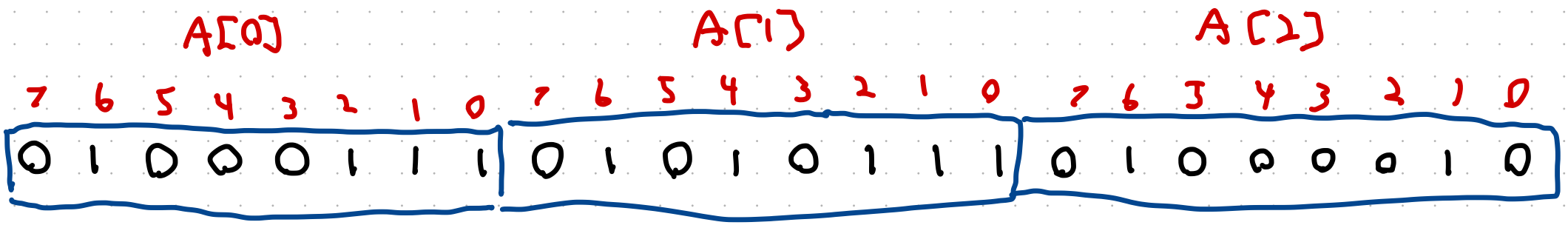
In the 6^TH Bit of the 2^ND Byte

22 mod 8

$\left\lfloor \frac{22}{8} \right\rfloor$

How do we get this value?

3rd Byte 0 1 0 0 0 0 1 0



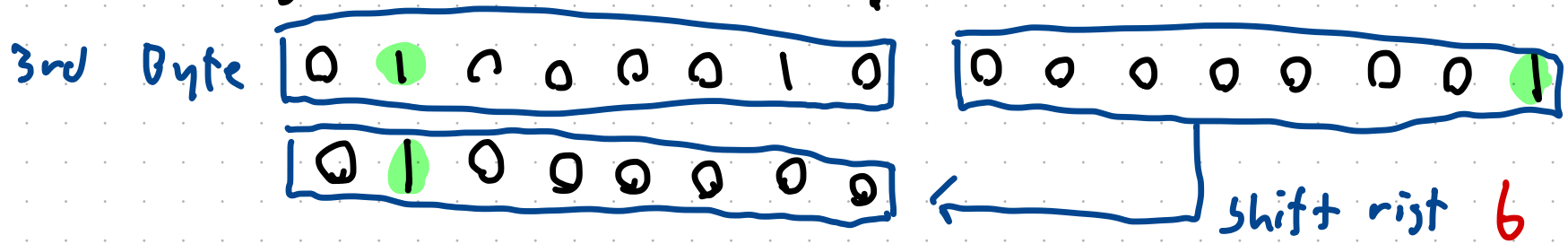

Where would we keep the 22^ND Bit?

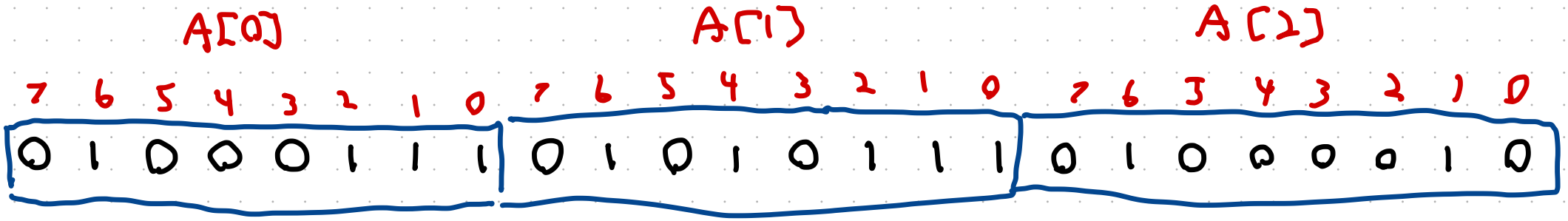
In the 6^TH Bit of the 2^ND Byte

22 mod 8

$\lfloor \frac{22}{8} \rfloor$

How do we get this value?





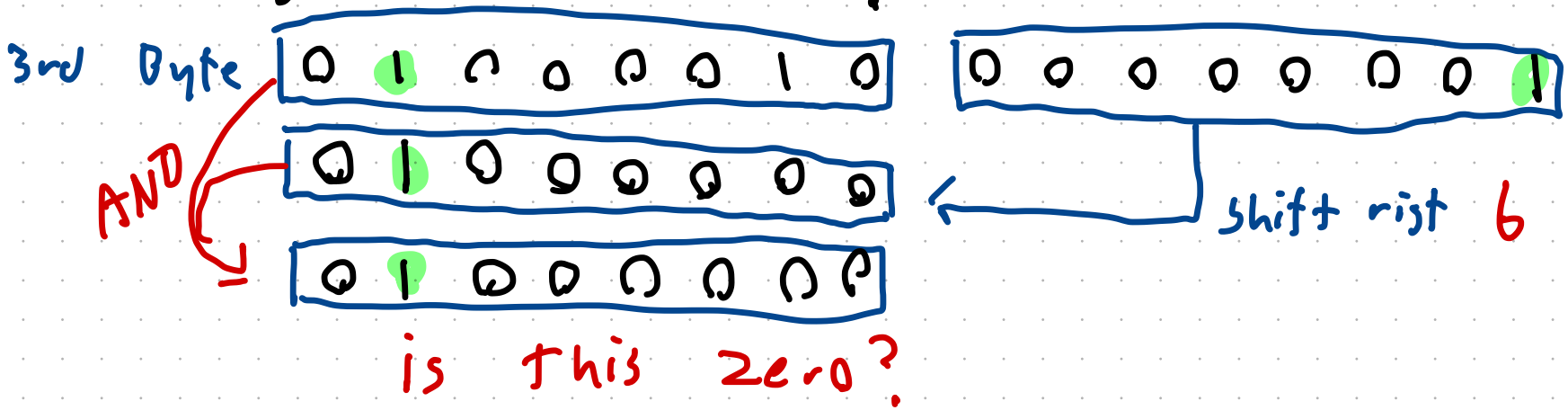

Where would we keep the 22^ND Bit?

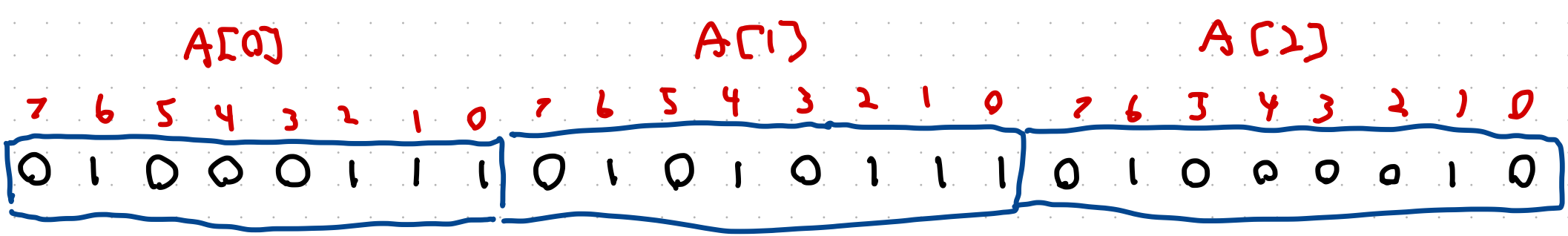
In the 6^TH Bit of the 2^ND Byte

22 mod 8

$\lfloor \frac{22}{8} \rfloor$

How do we get this value?






Python

Where would we keep the 22^ND Bit?

In the 6^TH Bit of the 2^ND Byte

22 mod 8

22 % 8

$\lfloor \frac{22}{8} \rfloor$

22 // 8

How do we get this value?