CS 112

How to approach programming problems:

The BigInt

Spring 2021

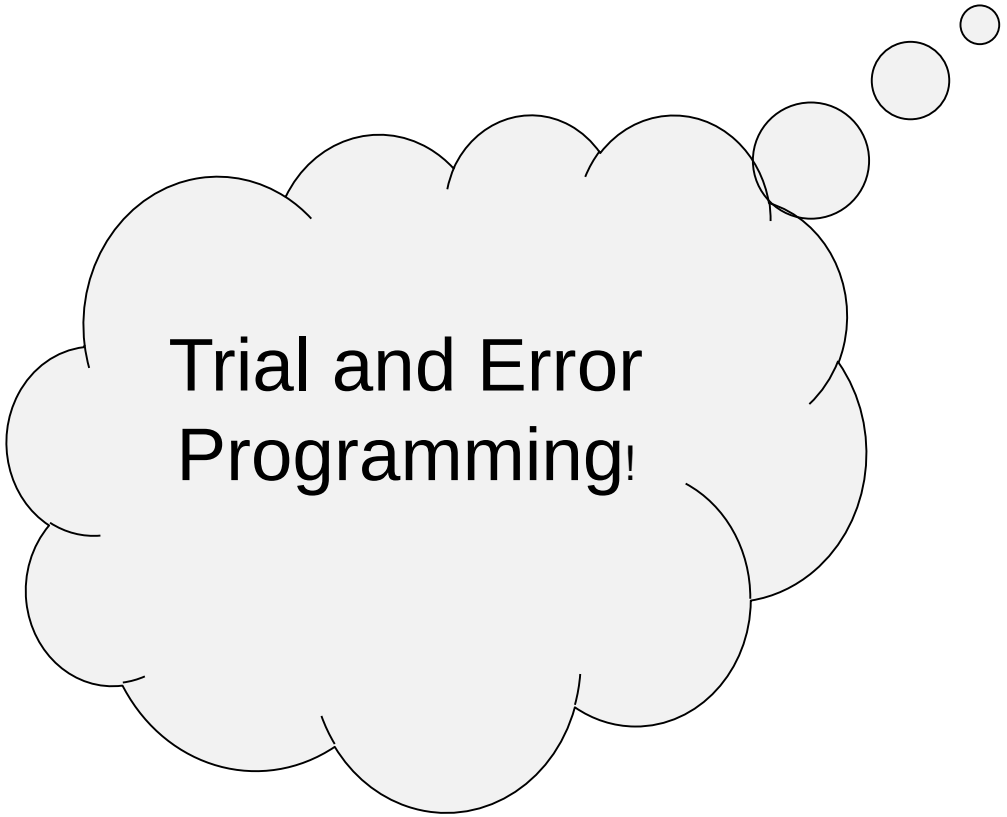
Christine Papadakis-Kanaris

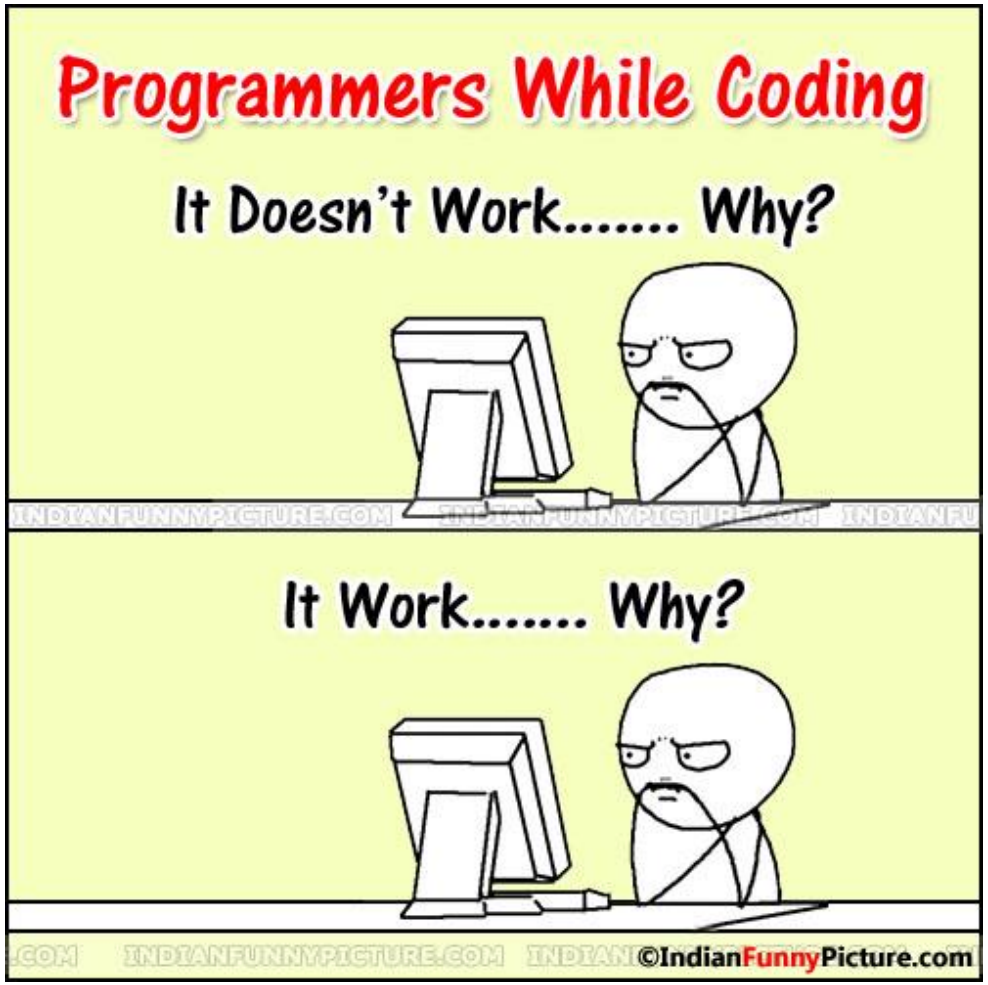
How to Approach Programming Problems

What not to do:

How to Approach Programming Problems

What not to do:





How to Approach Programming Problems

What to do:

#1
Understand the Problem!

How to Approach Programming Problems

What to do:

How to Approach Programming Problems

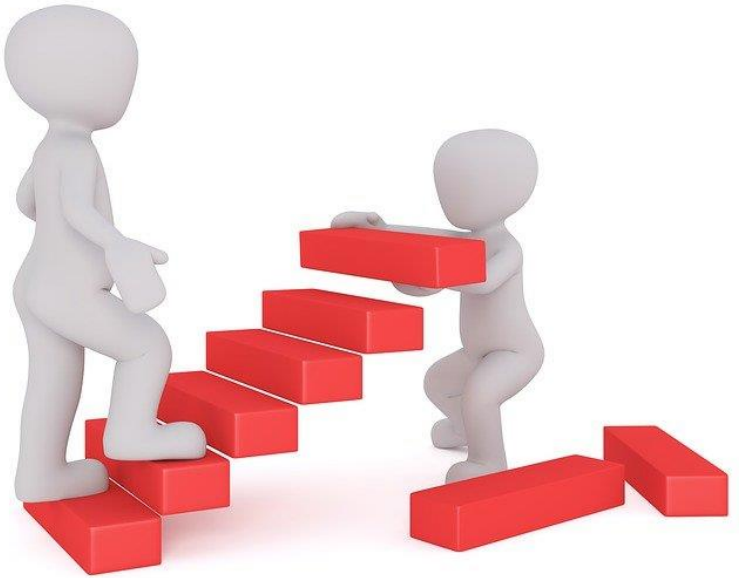
What to do:

#2
Form the solution

How to Approach Programming Problems

What to do:

#2
Identify the BASIC steps



Understanding BigInt

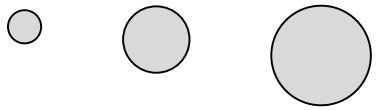
- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.

- How can we represent very large integers in Java?

Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- How is it that Python does not have this issue?

```
number = 5
number = 50000
number = 500000000000
```



The variable `number` does not contain the assigned value, but a reference to an object that represents that value.

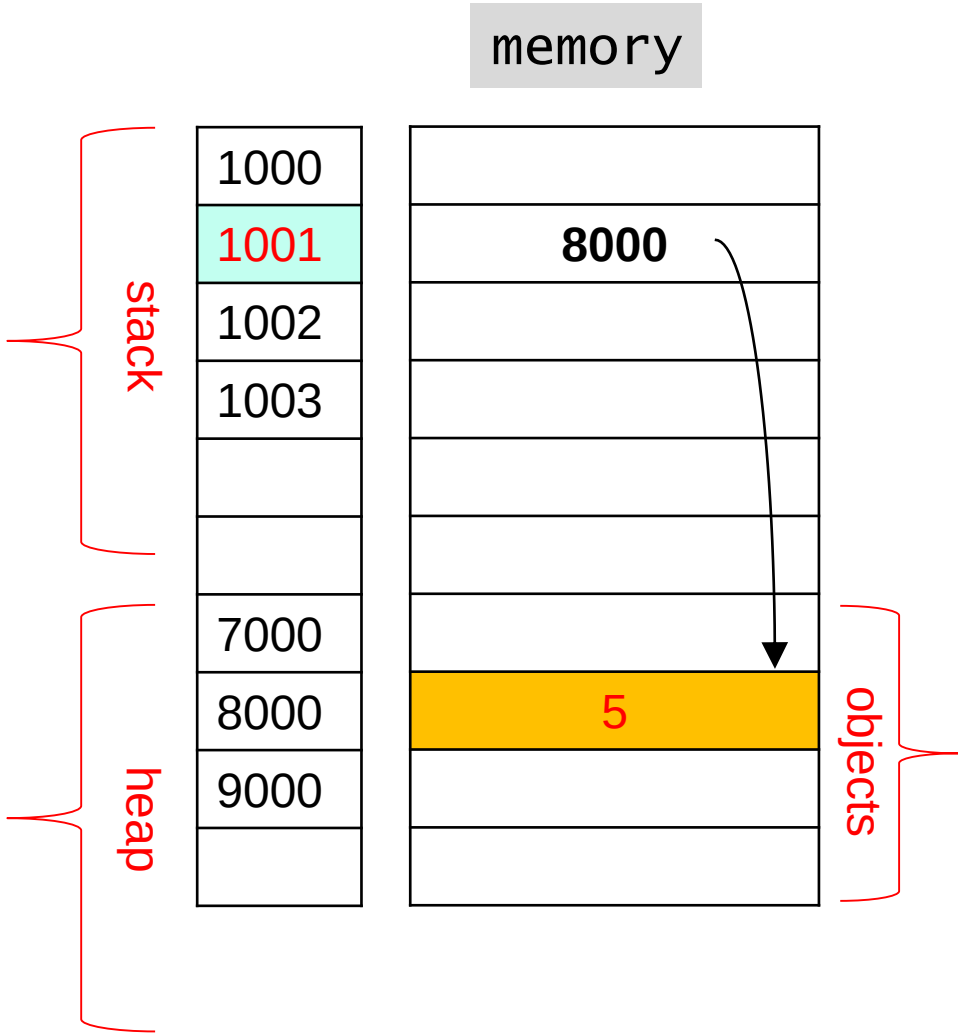
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- How is it that Python does not have this issue?

```
number = 5
number = 50000
number = 500000000000
```

mapping

number	1001



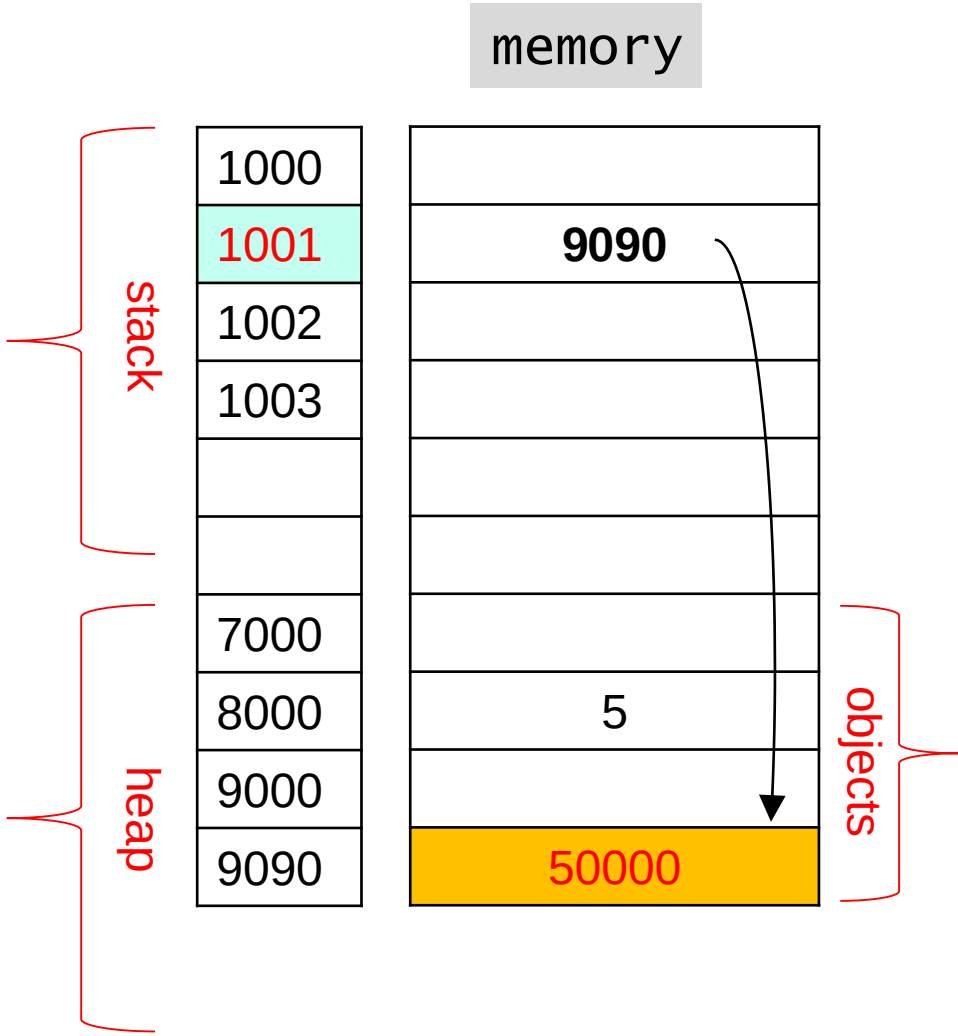
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- How is it that Python does not have this issue?

```
number = 5
number = 50000
number = 50000000000
```

mapping

number	1001



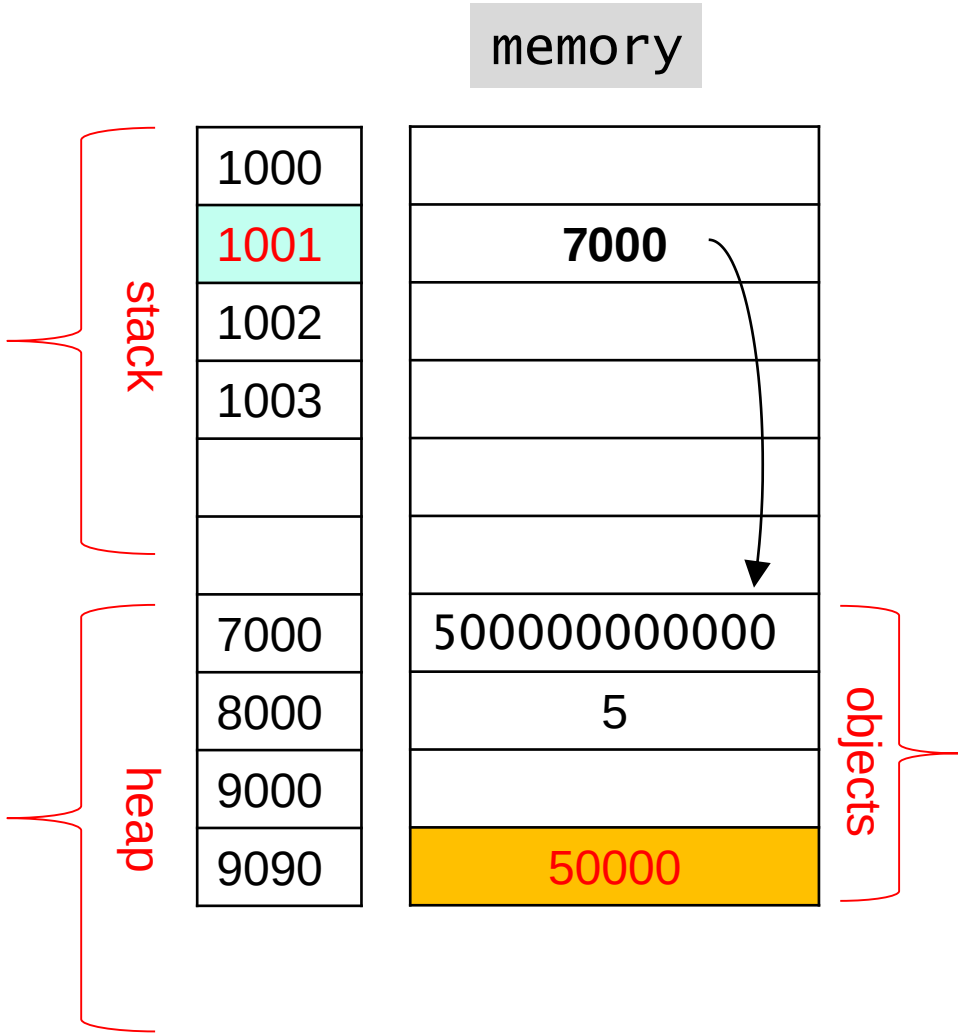
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- How is it that Python does not have this issue?

```
number = 5
number = 50000
number = 50000000000
```

mapping

number	1001



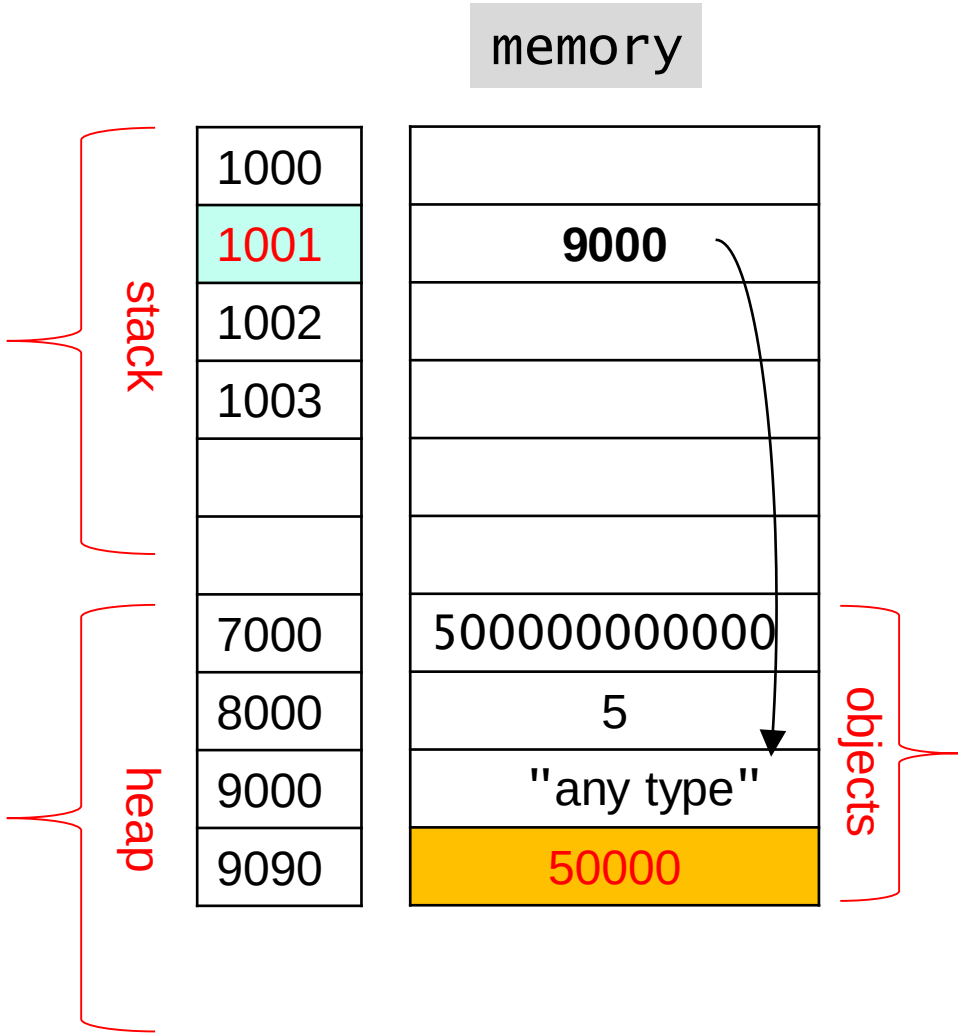
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- How is it that Python does not have this issue?

```
number = 5
number = 50000
number = "any type"
```

mapping

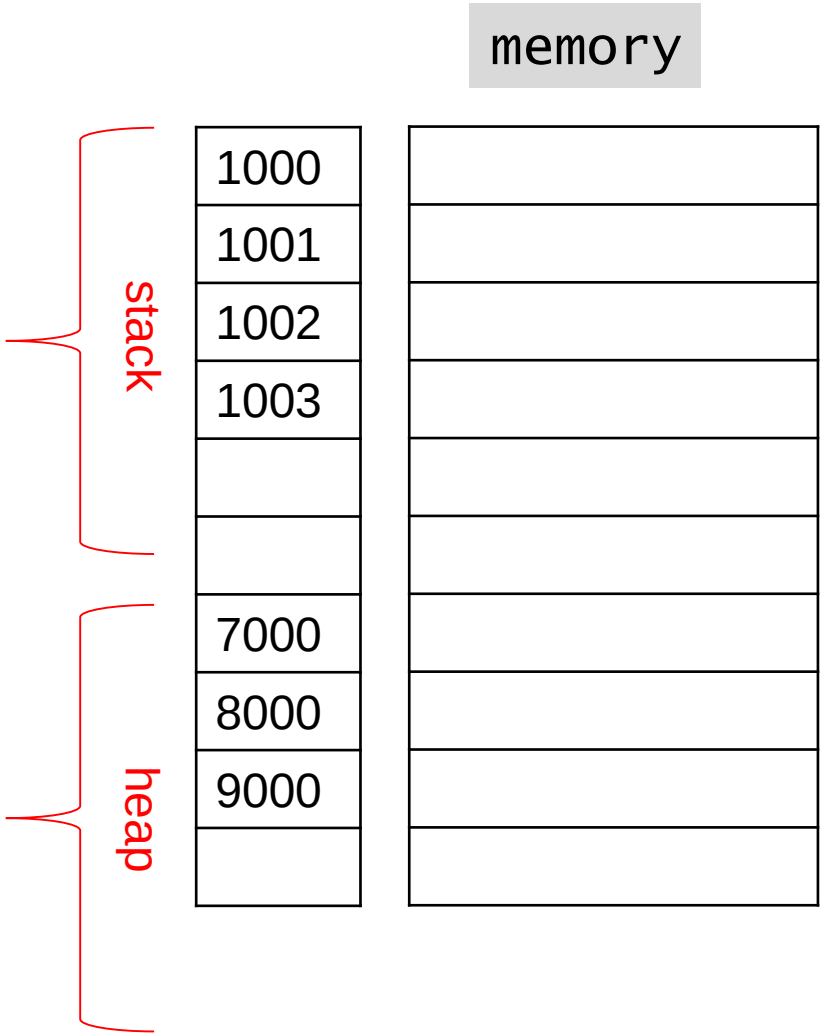
number	1001



Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- Java is a strongly typed compiled language.

```
int number = 5;
number = 50000;
number = 500000000000;
```



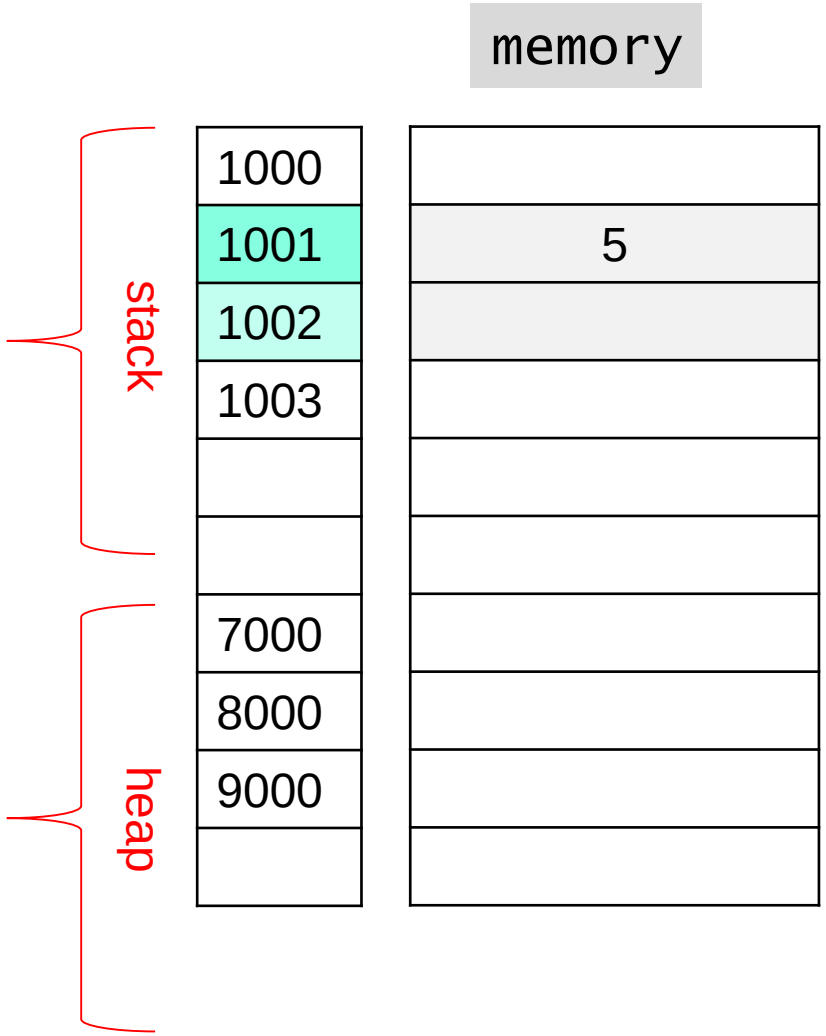
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- Java is a strongly typed compiled language.

```
int number = 5; // assume 2 bytes per integer
number = 50000;
number = 500000000000;
```

mapping

number	1001



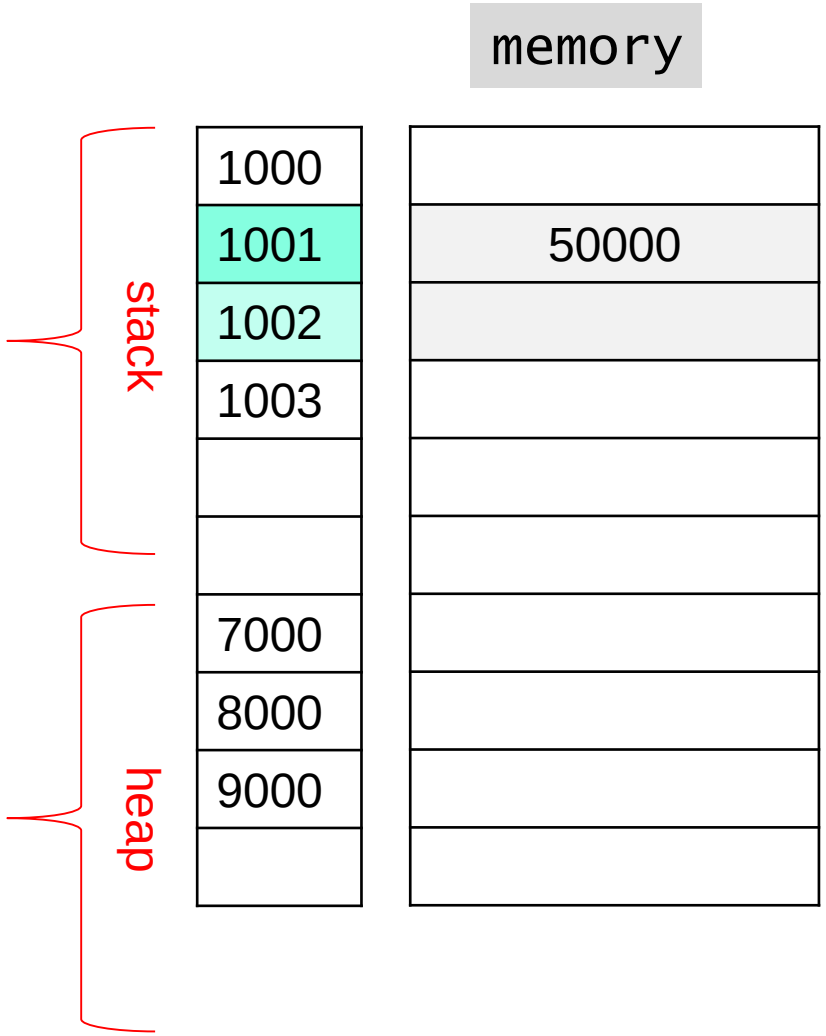
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- Java is a strongly typed compiled language.

```
int number = 5; // assume 2 bytes per integer
number = 50000;
number = 500000000000;
```

mapping

number	1001



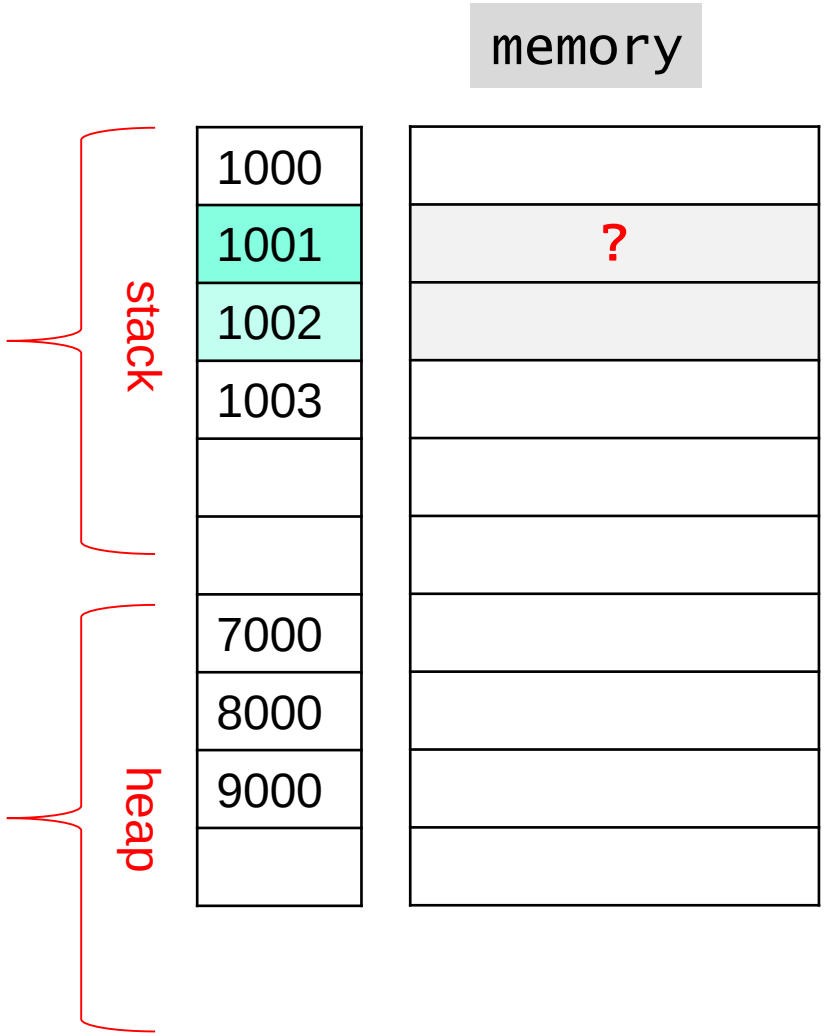
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- Java is a strongly typed compiled language.

```
int number = 5; // assume 2 bytes per integer
number = 50000;
number = 500000000000;
```

mapping

number	1001



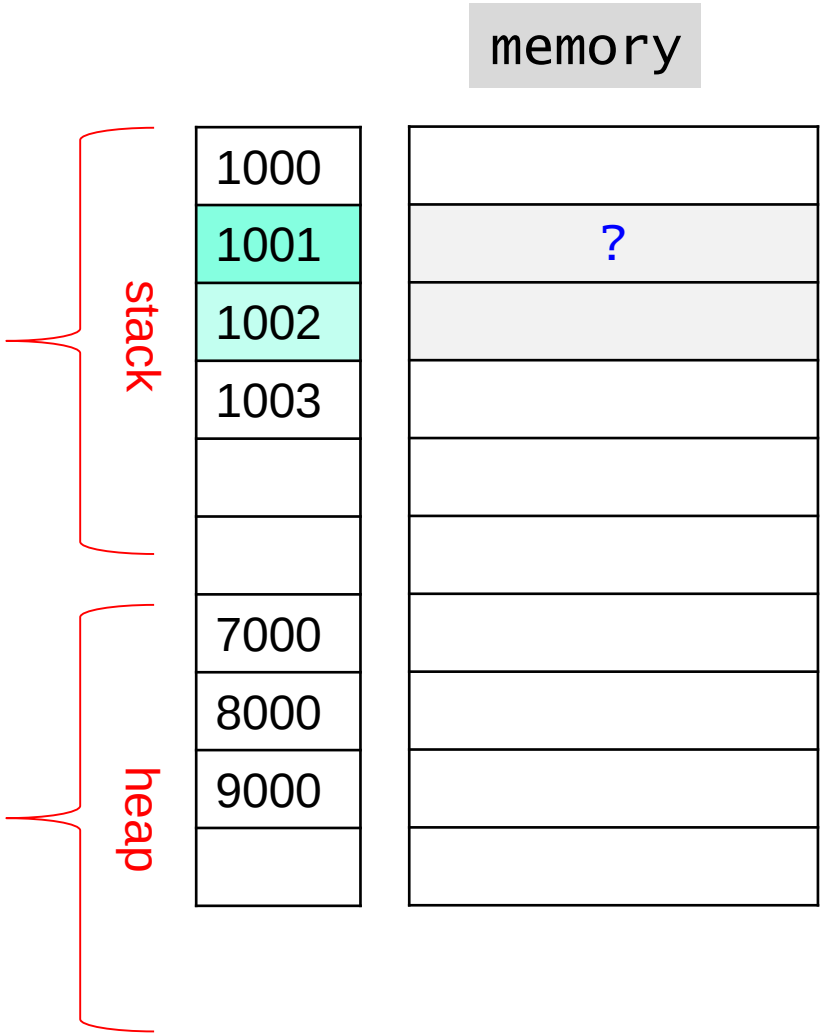
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- How can we represent very large integers in Java?
- Java is a strongly typed compiled language.

```
int number = 5; // assume 2 bytes per integer
number = 50000;
number = 500000000000;
number = 10.34; // needs double the bytes
```

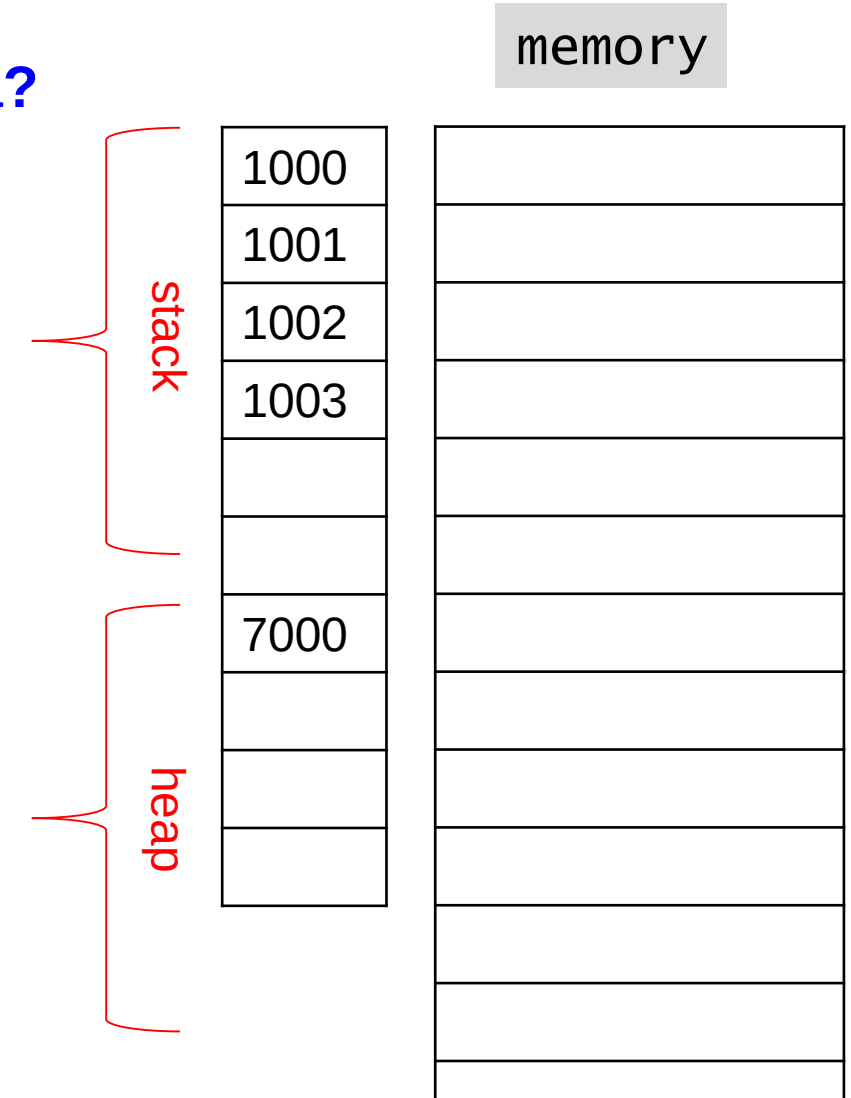
mapping

number	1001



Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Java's `BigInteger` class!** `BigInteger`



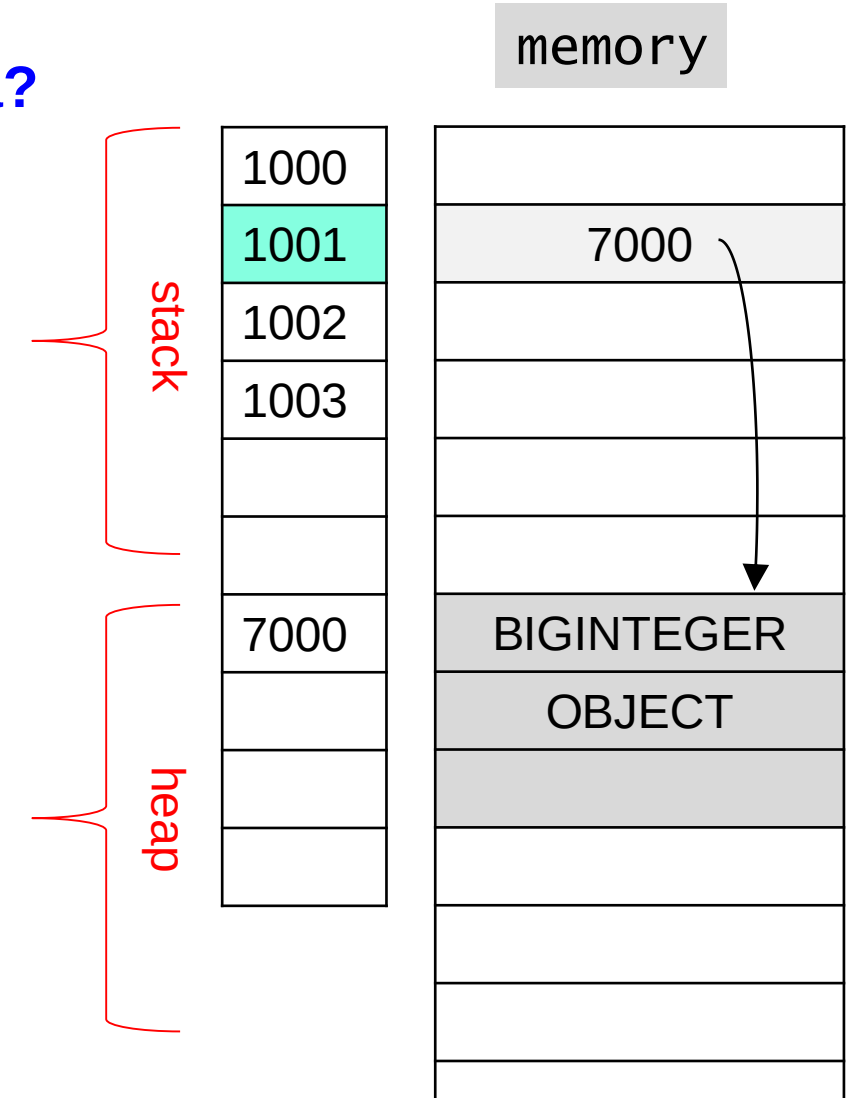
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Java's `BigInteger` class!** `BigInteger`

```
BigInteger number = new BigInteger( 5000000 );
```

mapping

number	1001



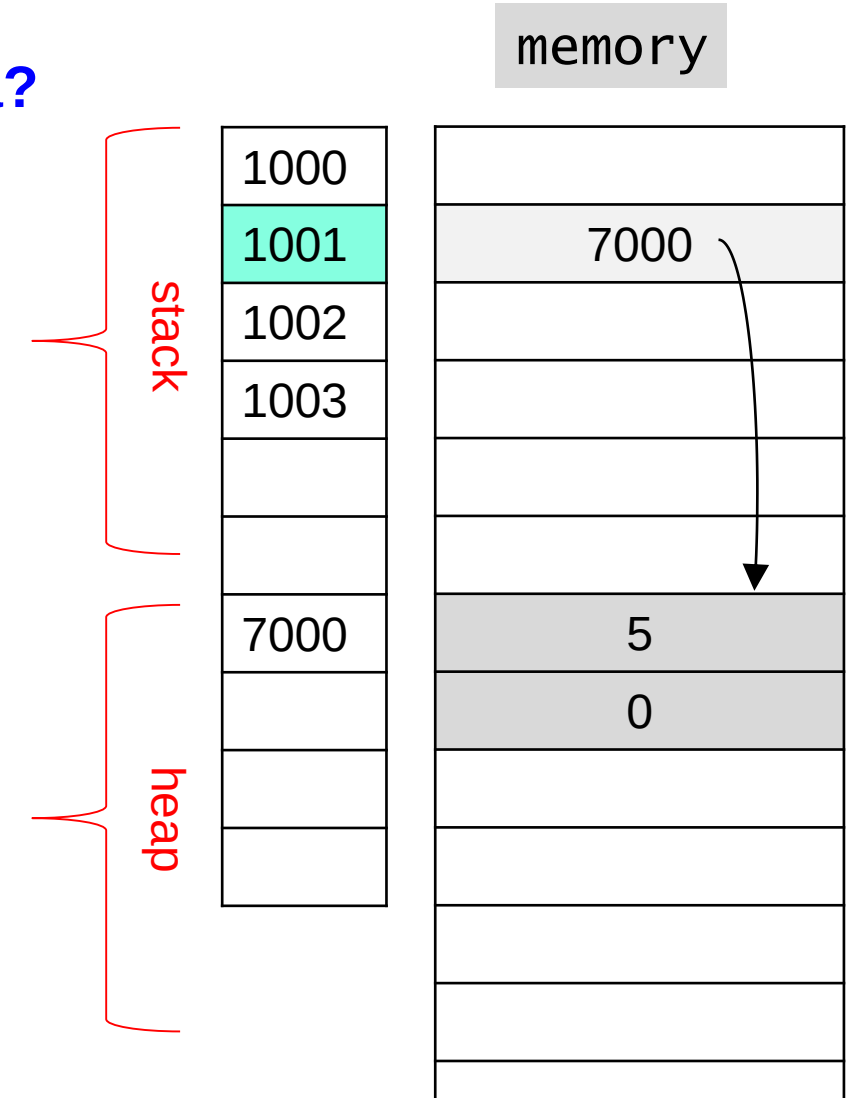
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

```
int[] number = {5,0}; // 50
```

mapping

number	1001



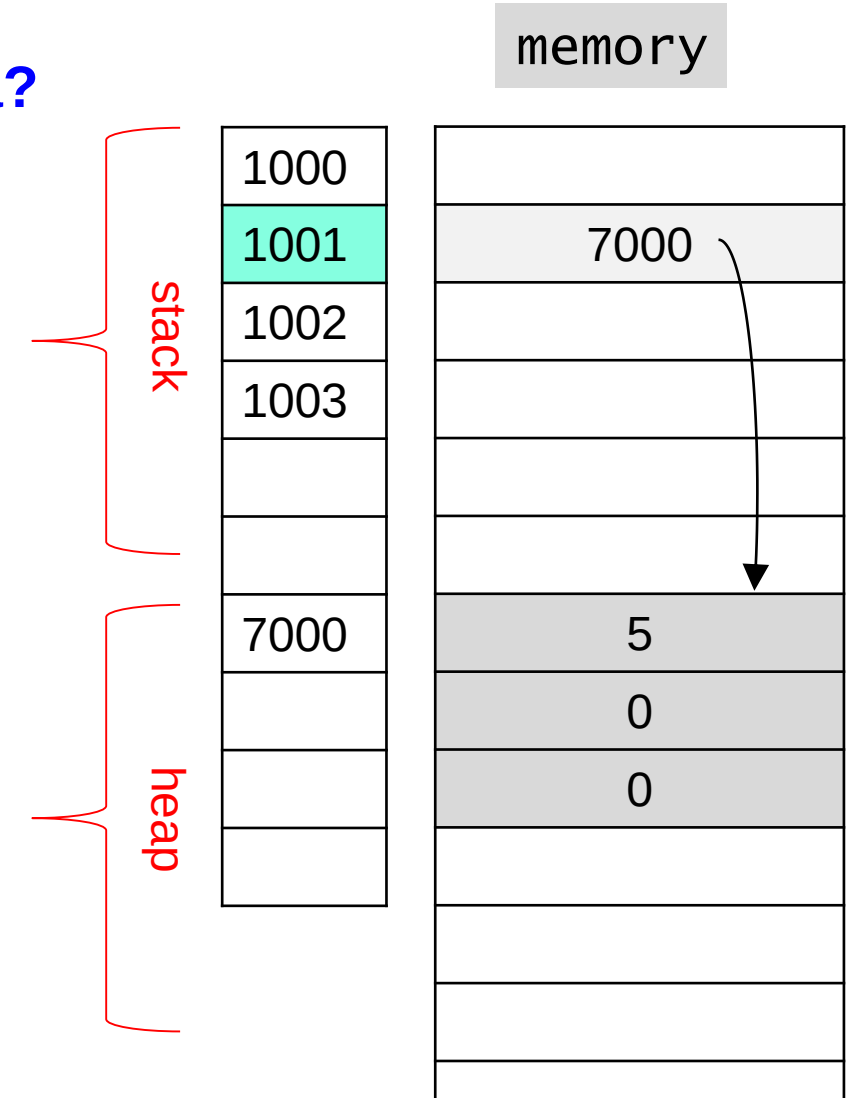
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

```
int[] number = {5,0,0}; // 500
```

mapping

number	1001



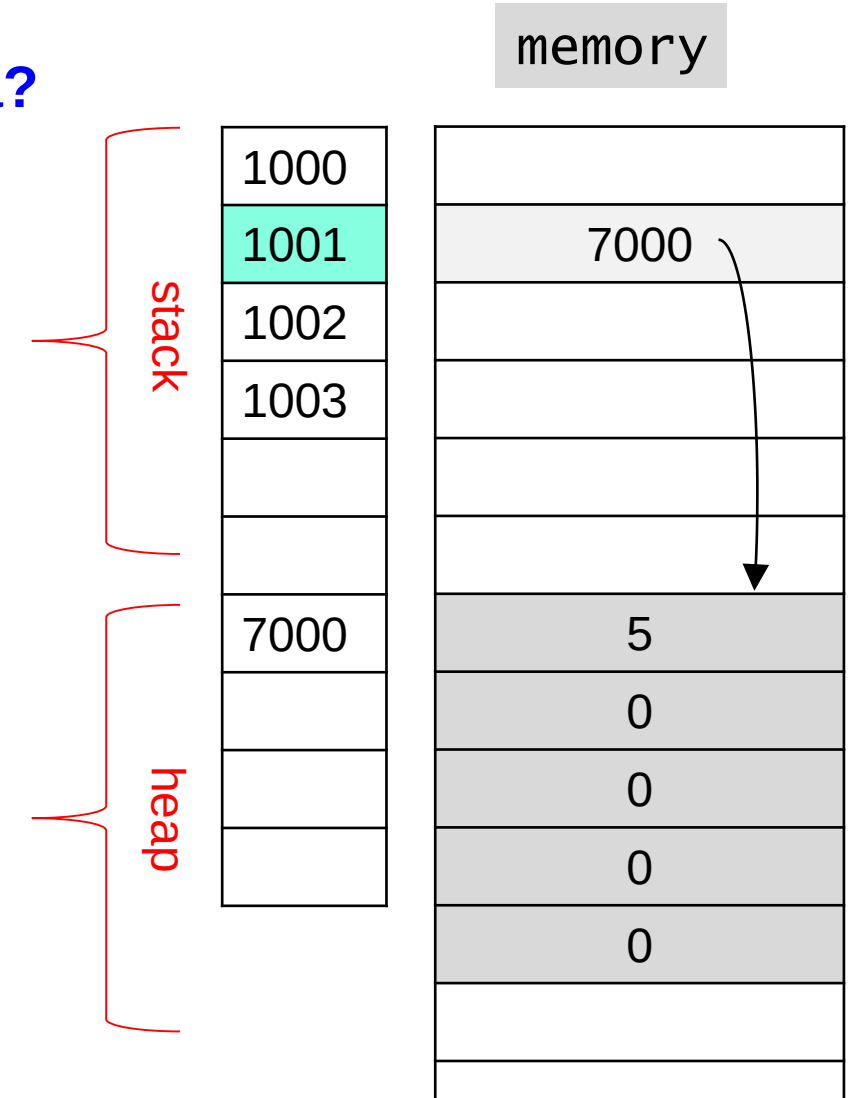
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

```
int[] number = {5,0,0,0,0}; // 50000
```

mapping

number	1001



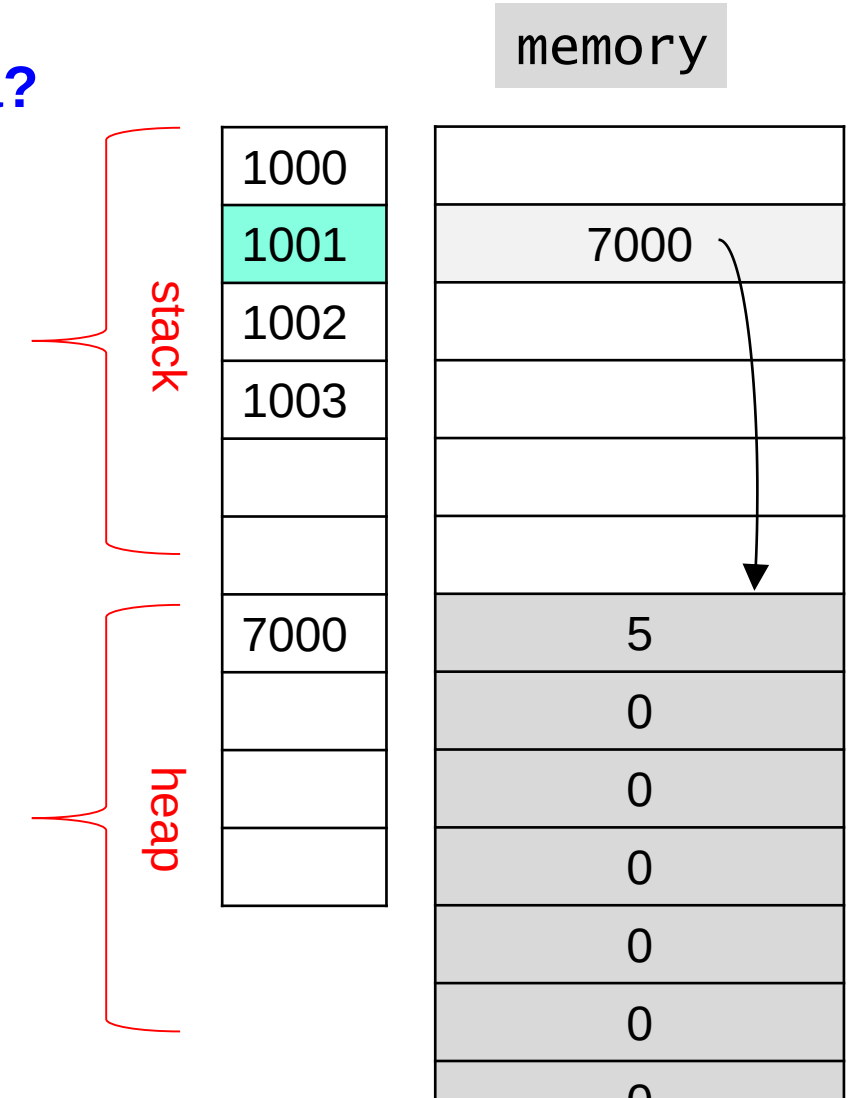
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

```
int[] number = {5,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

mapping

number	1001



Understanding BigInt

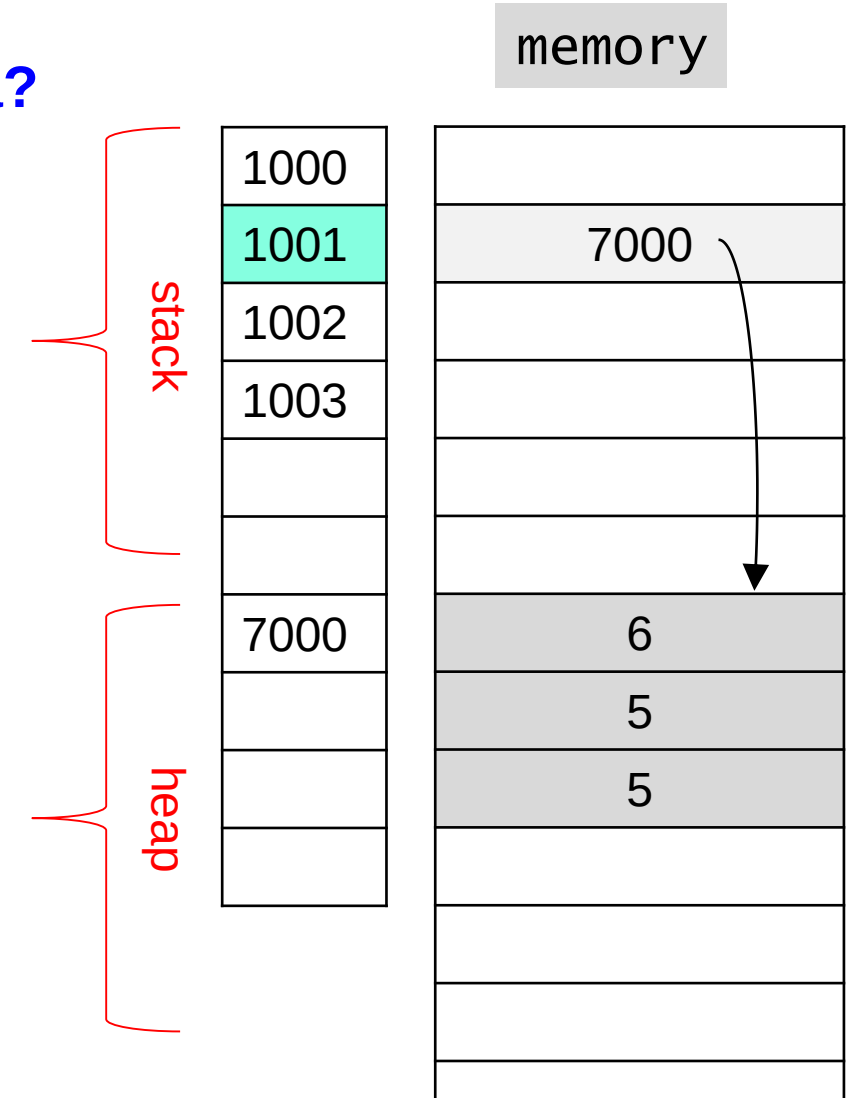
- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

```
int[] number = {6,5,5};
```

3 significant digits

mapping

number	1001



Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

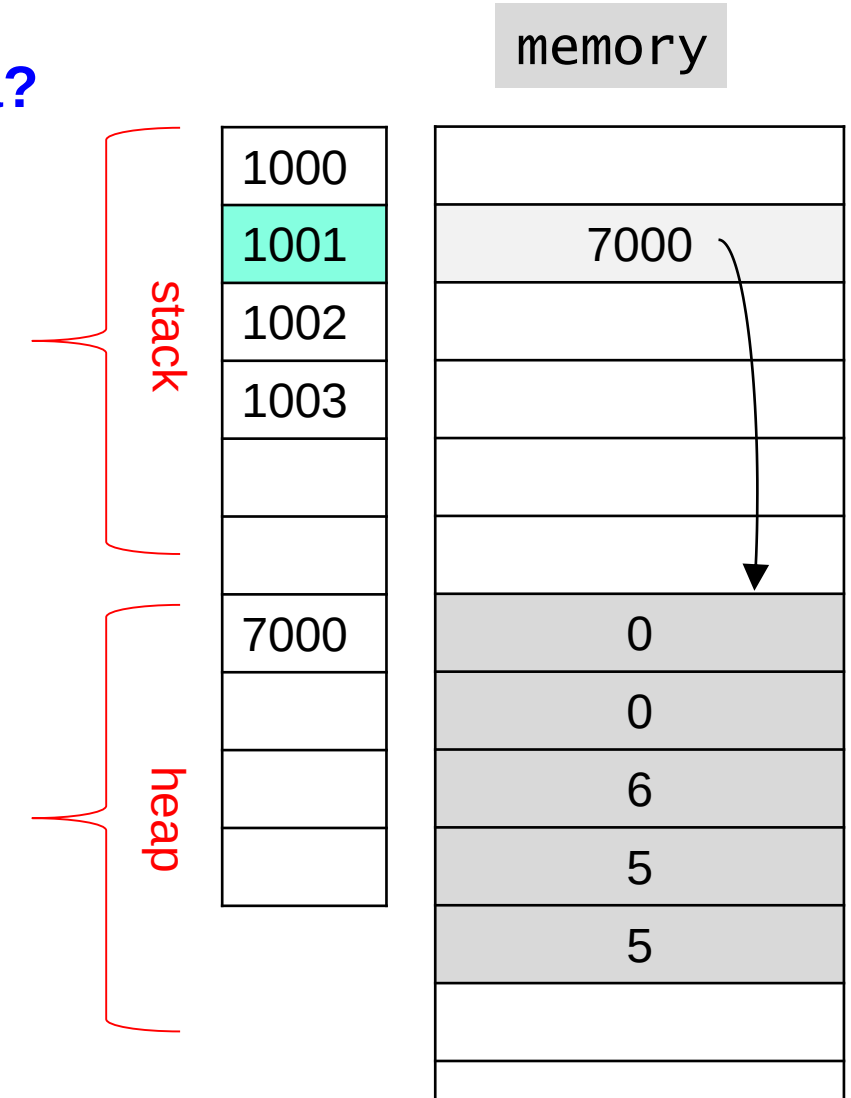
```
int[] number = {0,0,6,5,5};
```

hundred tens ones

also 3 significant digits

mapping

number	1001



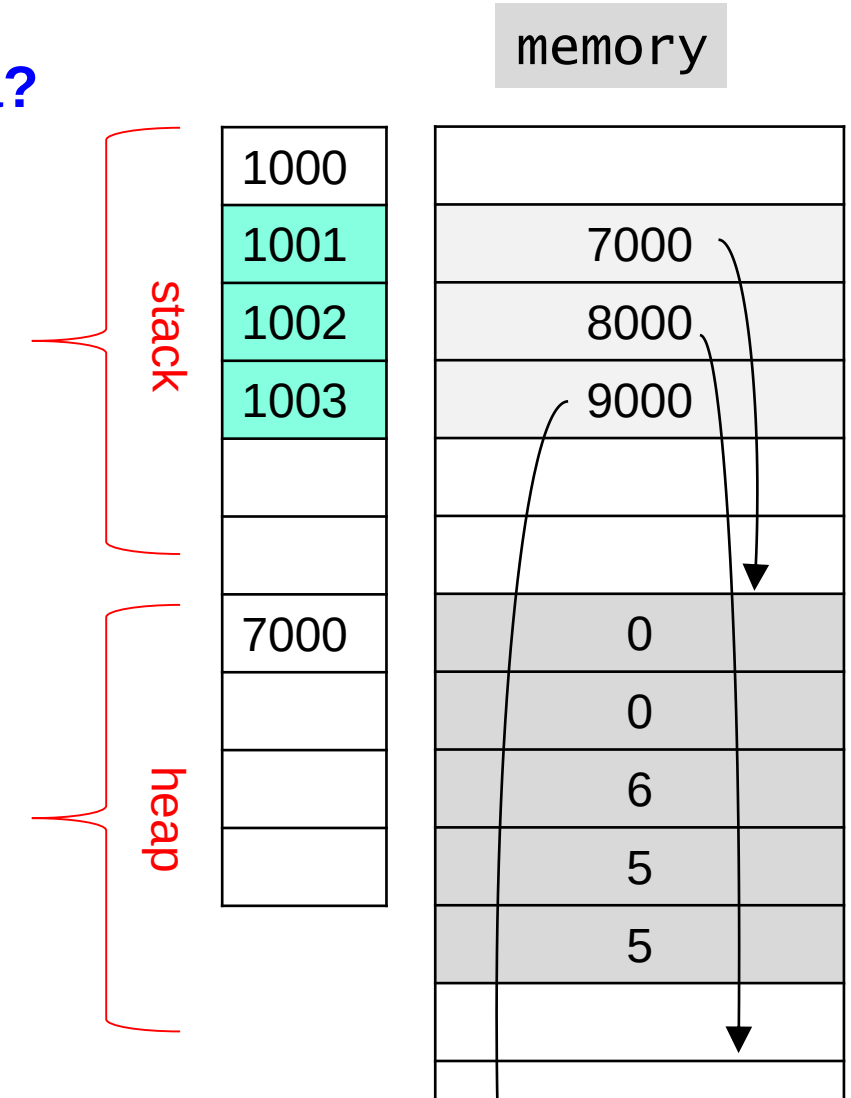
Understanding BigInt

- We know that the largest integer we can store in a primitive variable in Java is an `unsigned long`.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

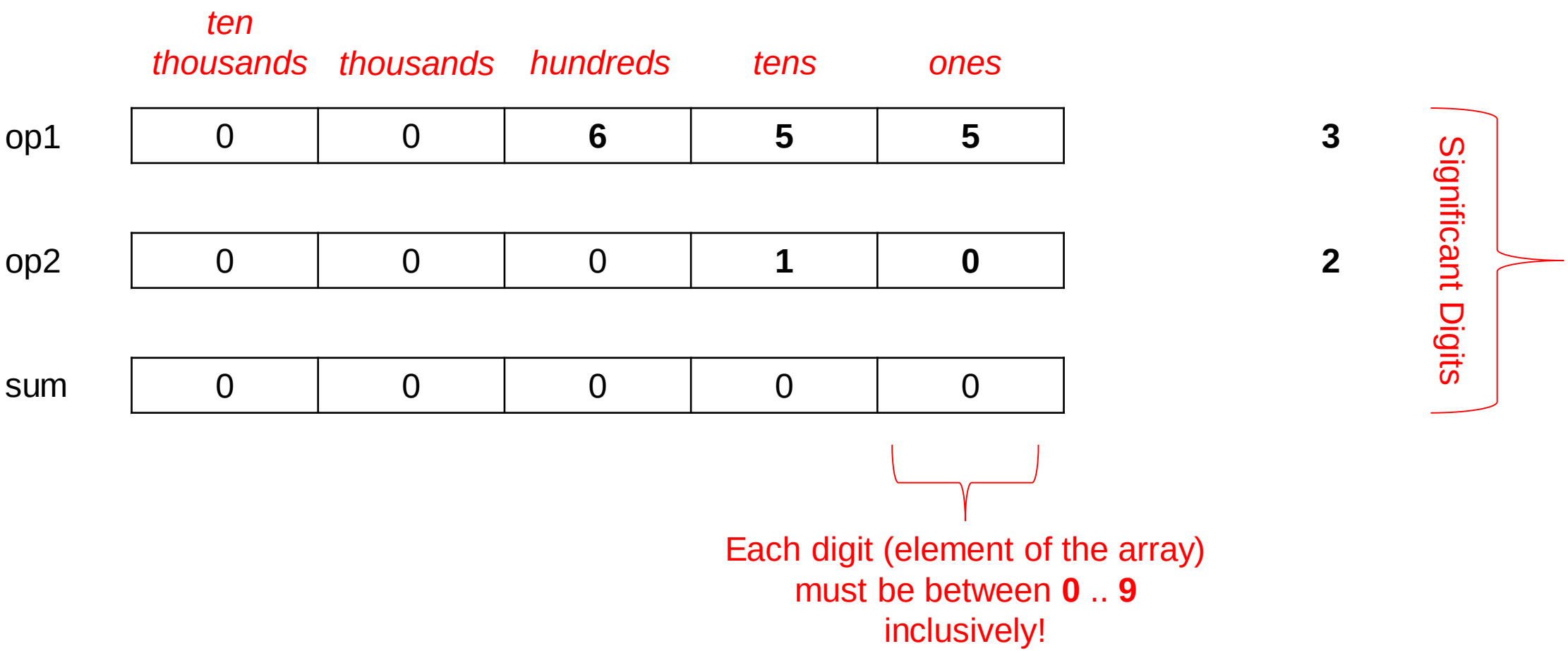
```
int[] op1 = {0,0,6,5,5};   // 655
int[] op2 = {0,0,0,1,0};   // 10
int[] sum = {0,0,0,0,0};   // 0
```

mapping

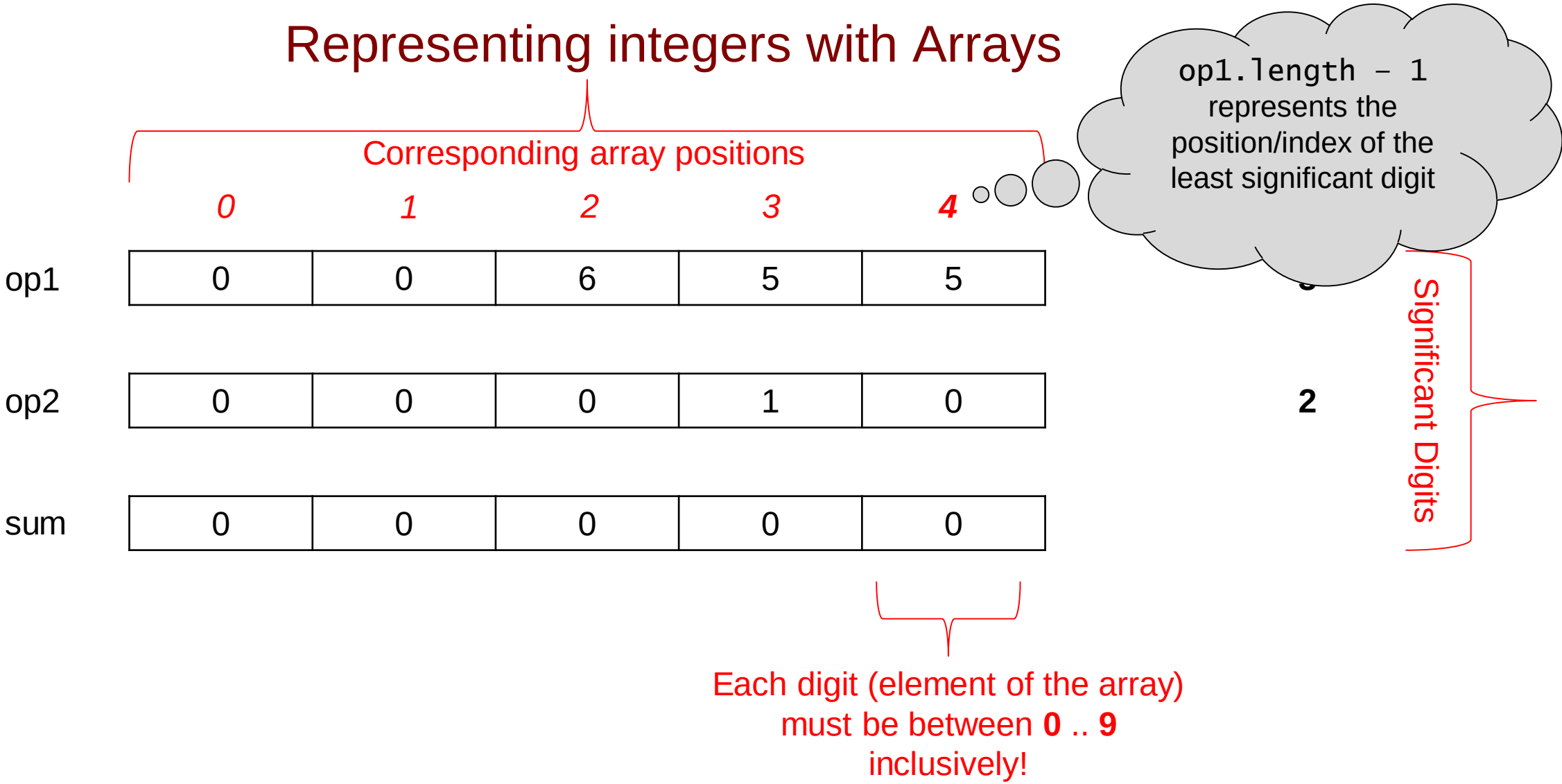
op1	1001
op2	1002
sum	1003



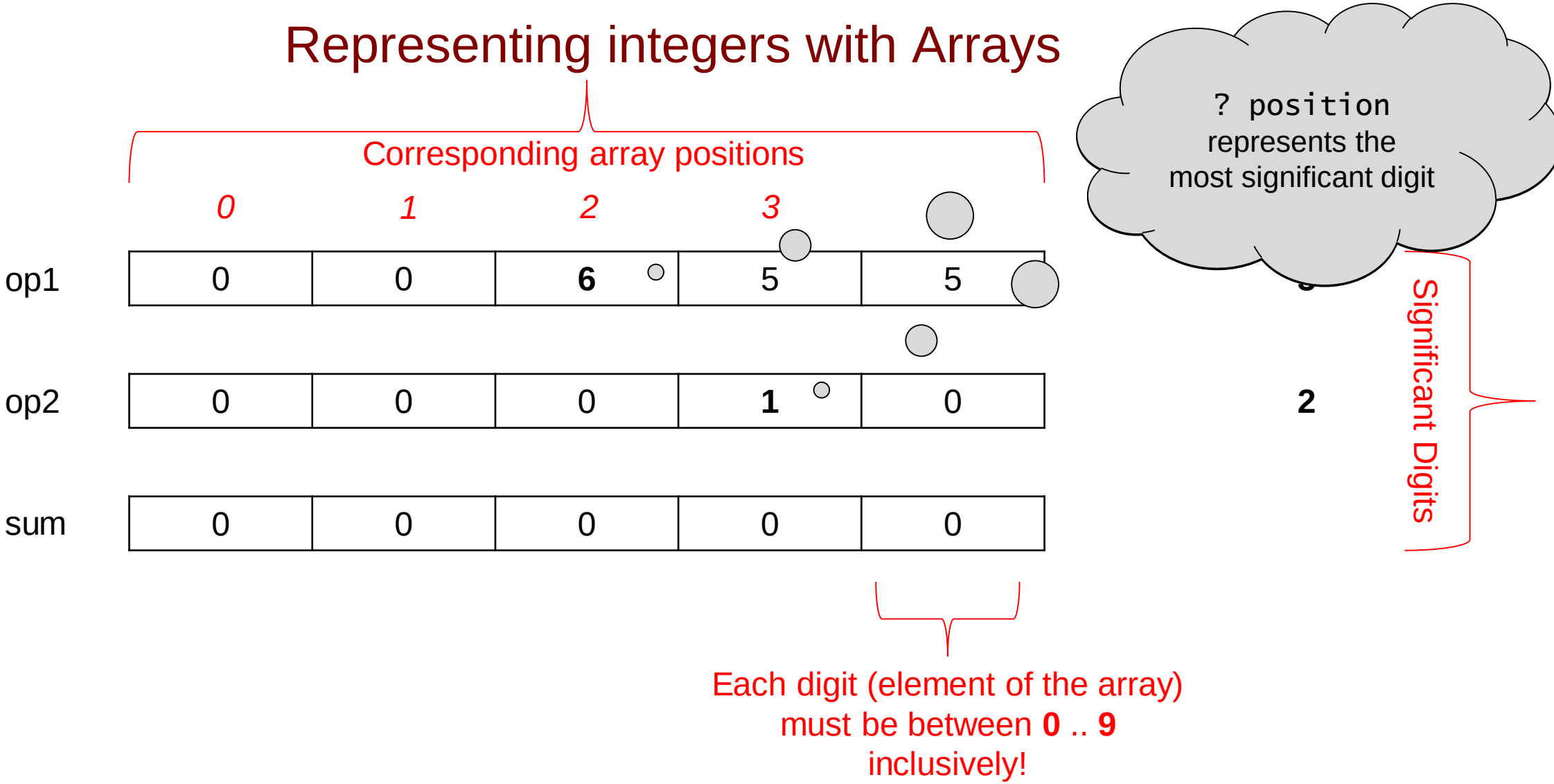
Representing integers with Arrays



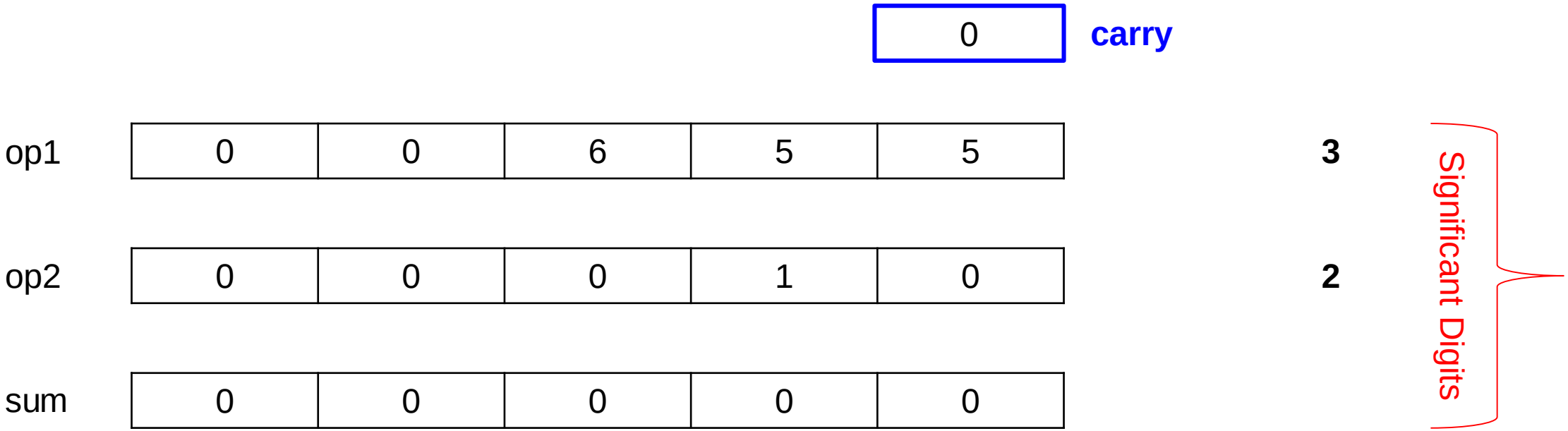
Representing integers with Arrays



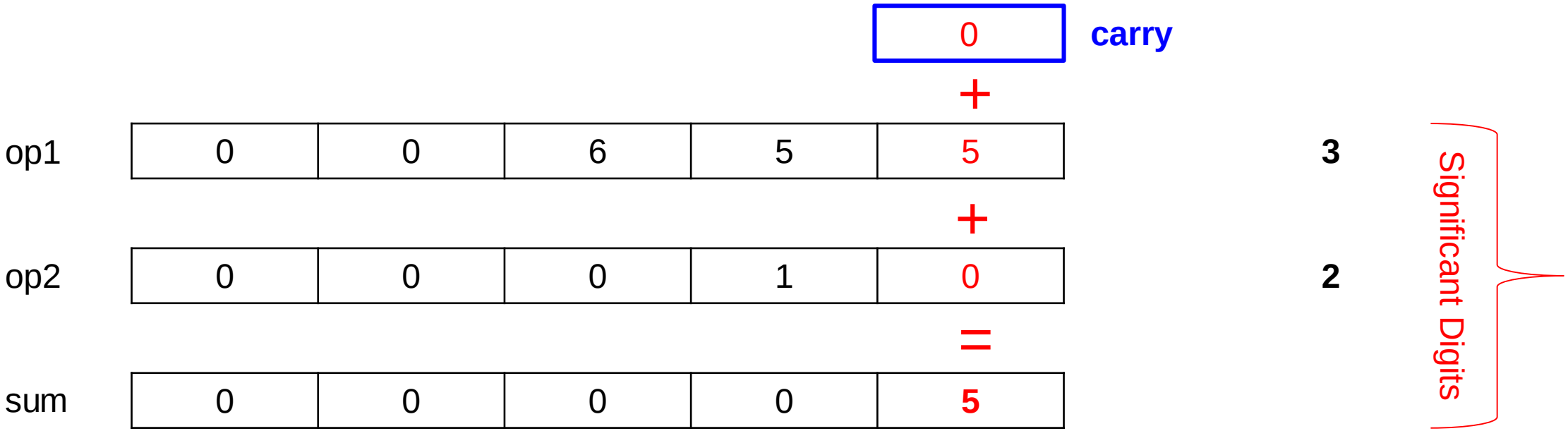
Representing integers with Arrays



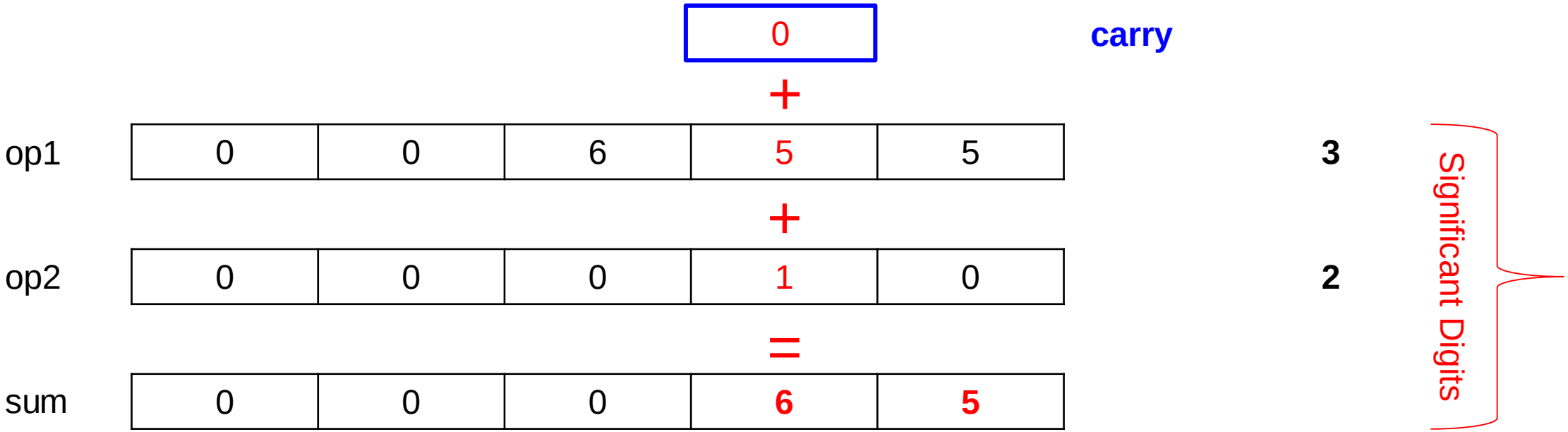
Representing integers with Arrays



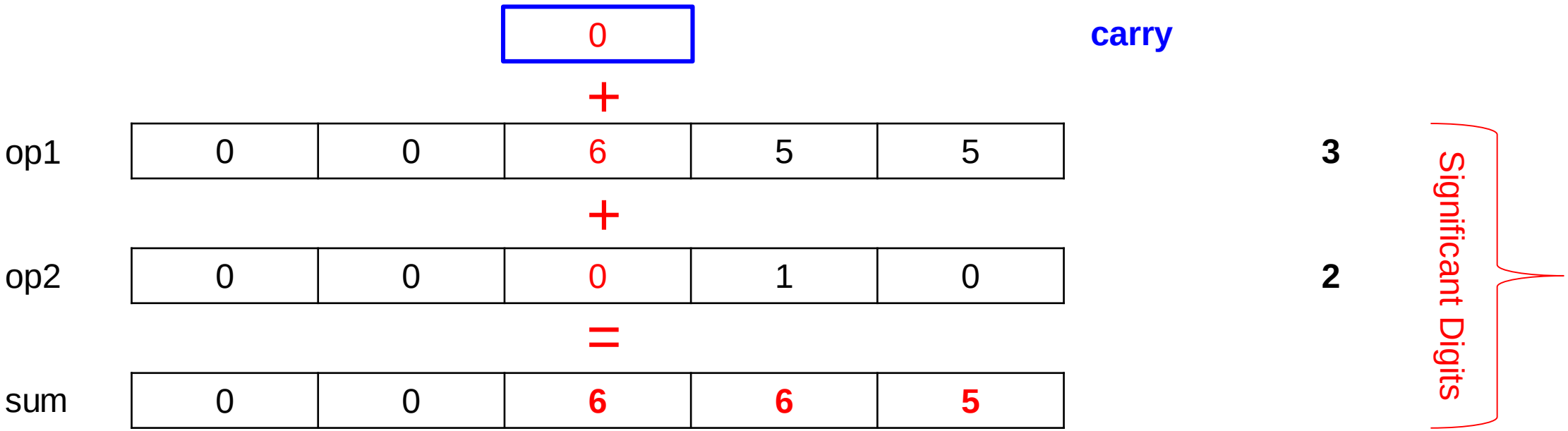
Representing integers with Arrays



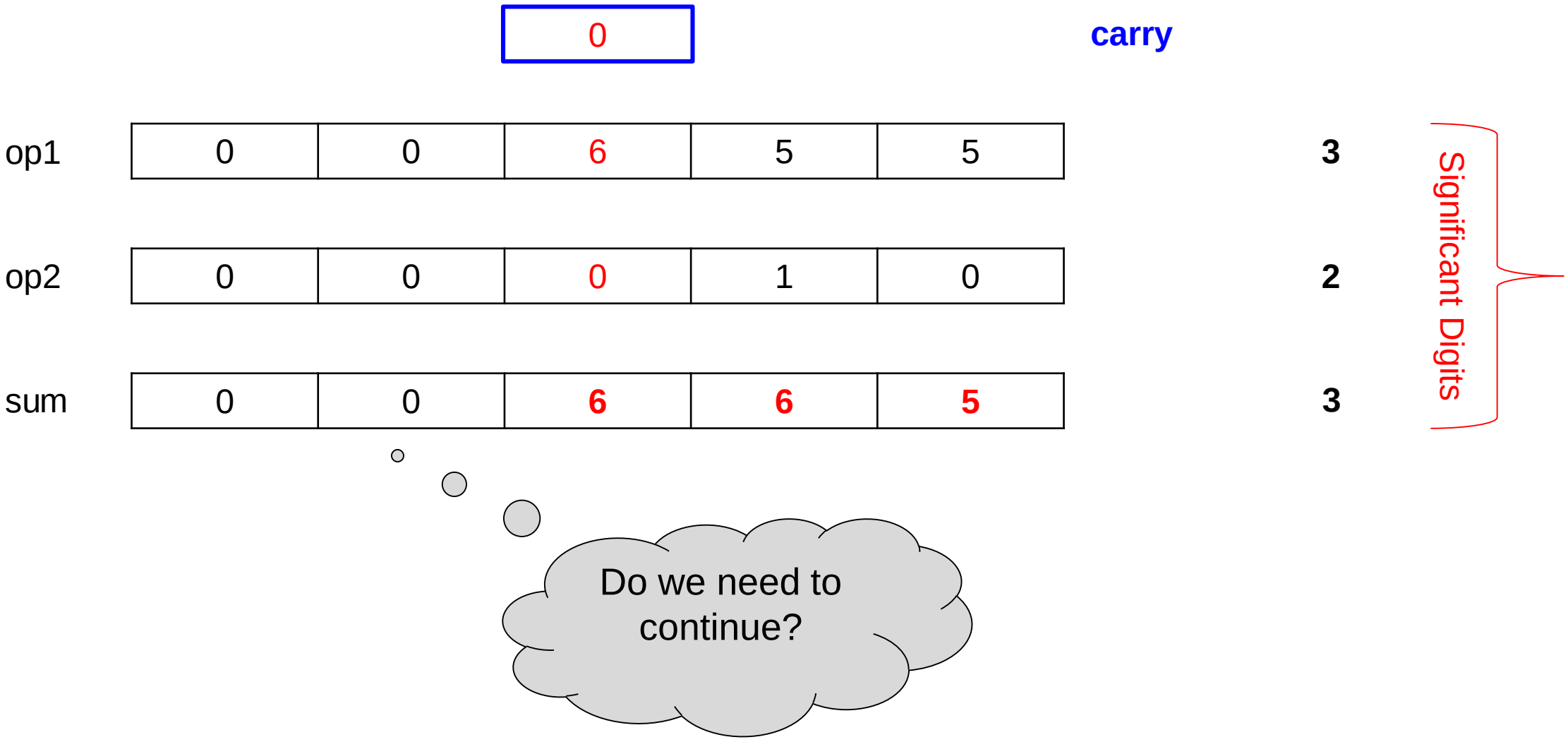
Representing integers with Arrays



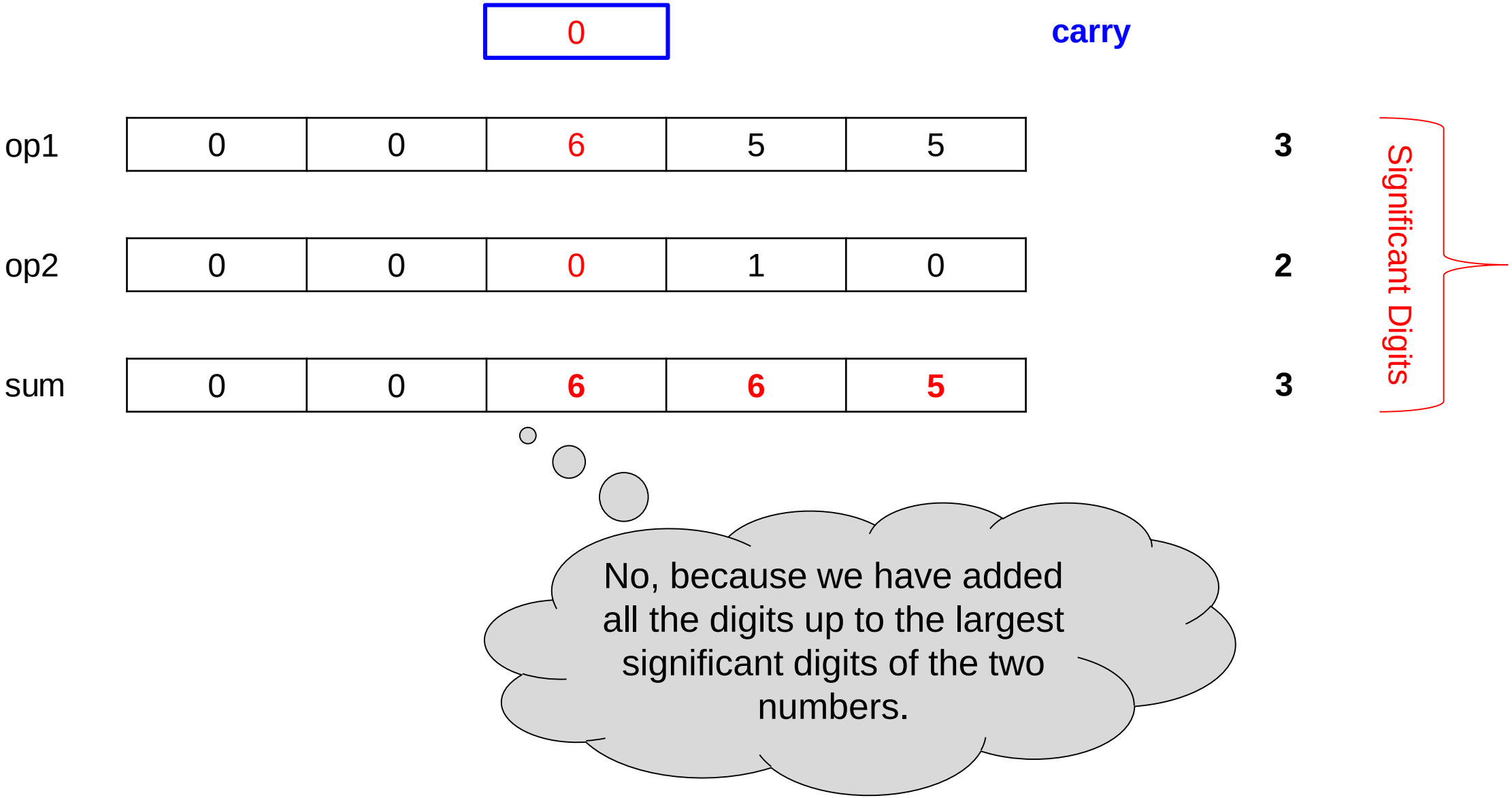
Representing integers with Arrays



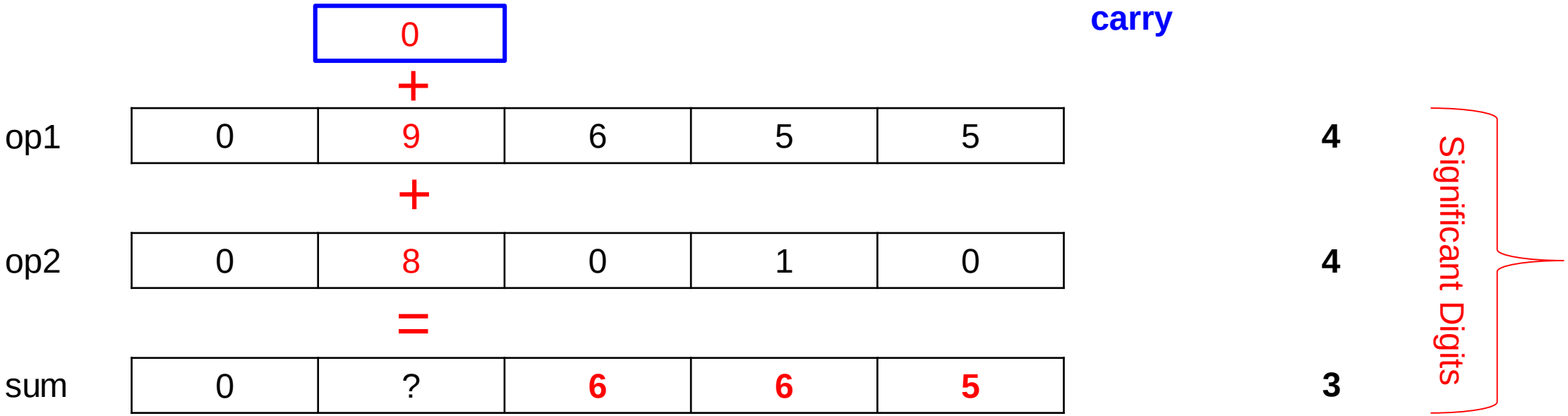
Representing integers with Arrays



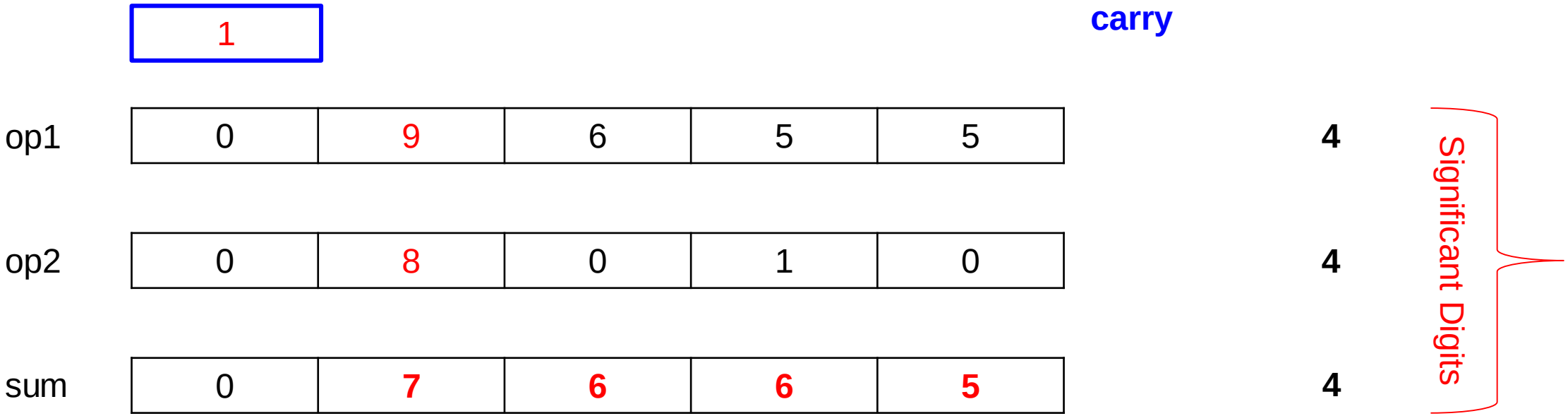
Representing integers with Arrays



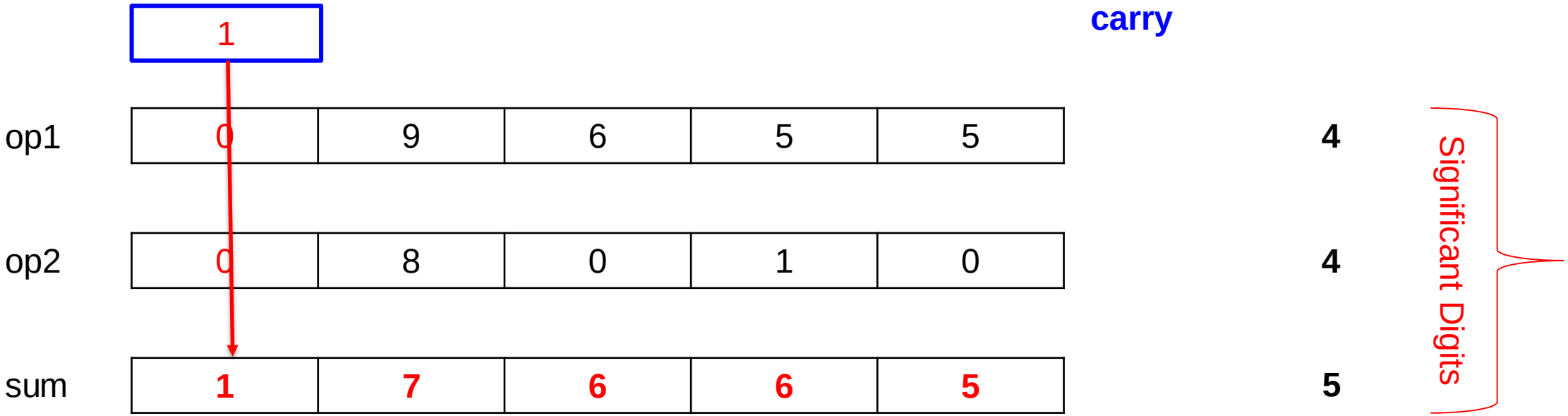
Representing integers with Arrays:
what if?



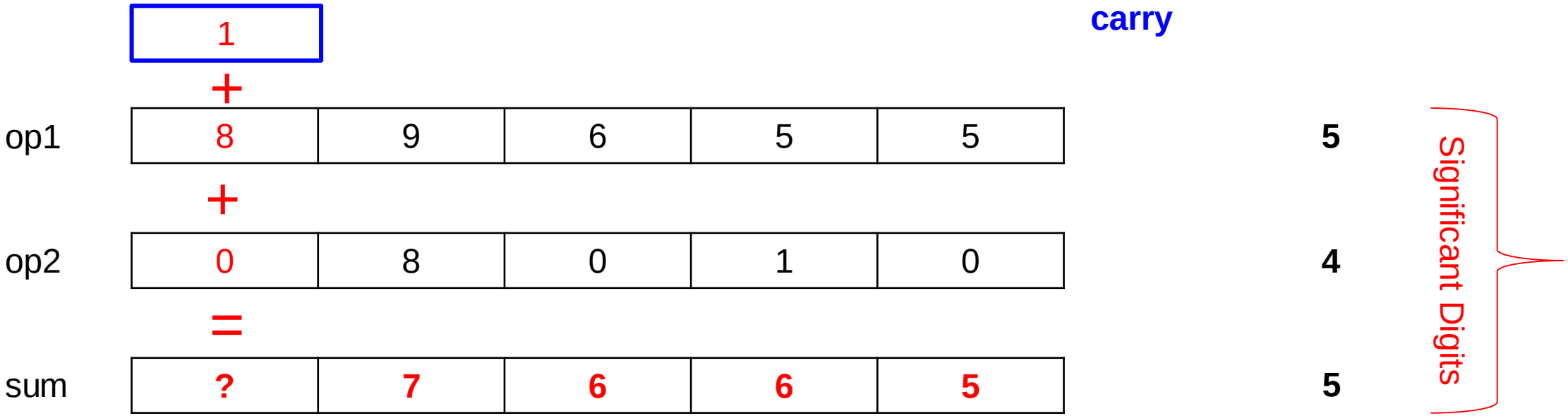
Representing integers with Arrays: what if?



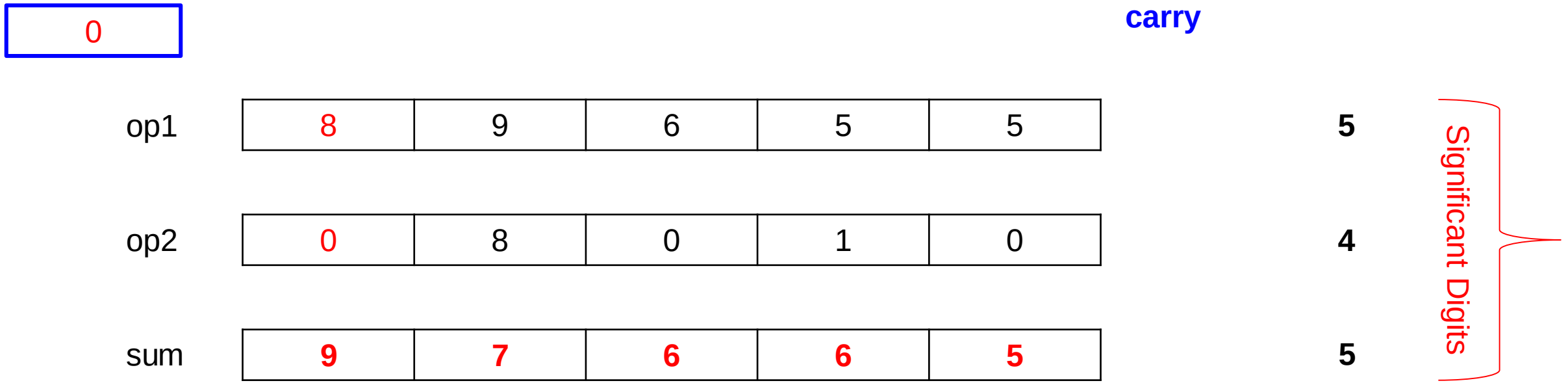
Representing integers with Arrays: what if?



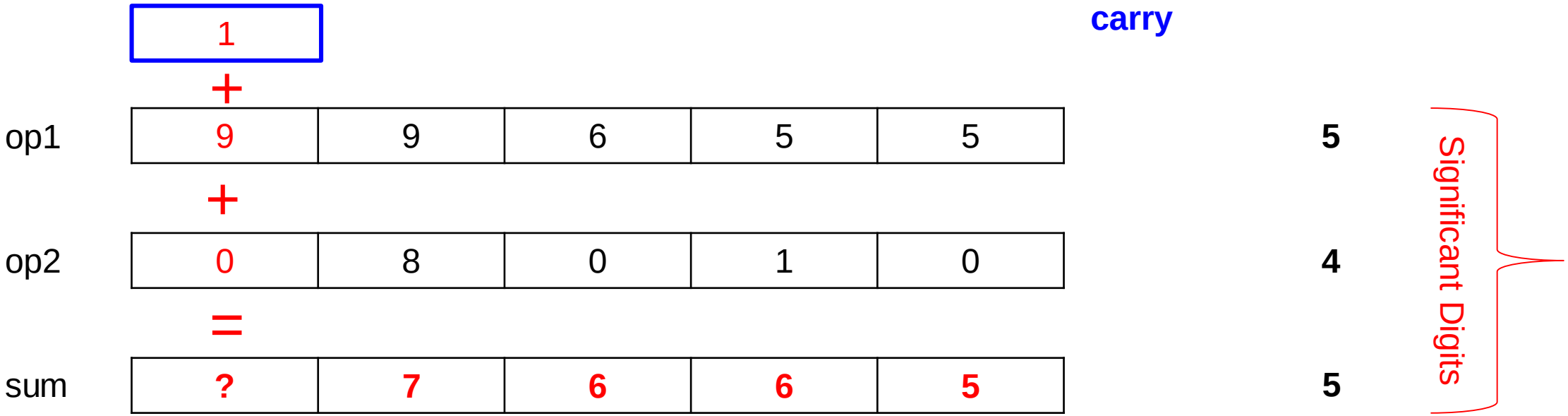
Representing integers with Arrays:
another what if?



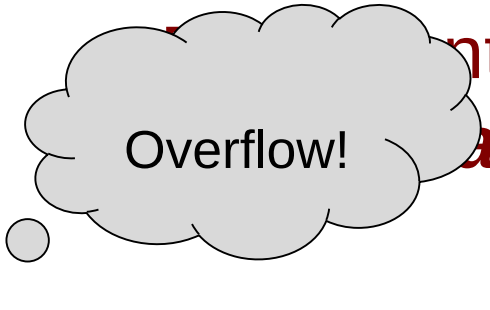
Representing integers with Arrays: ***another*** what if?



Representing integers with Arrays: yet ***another*** what if?



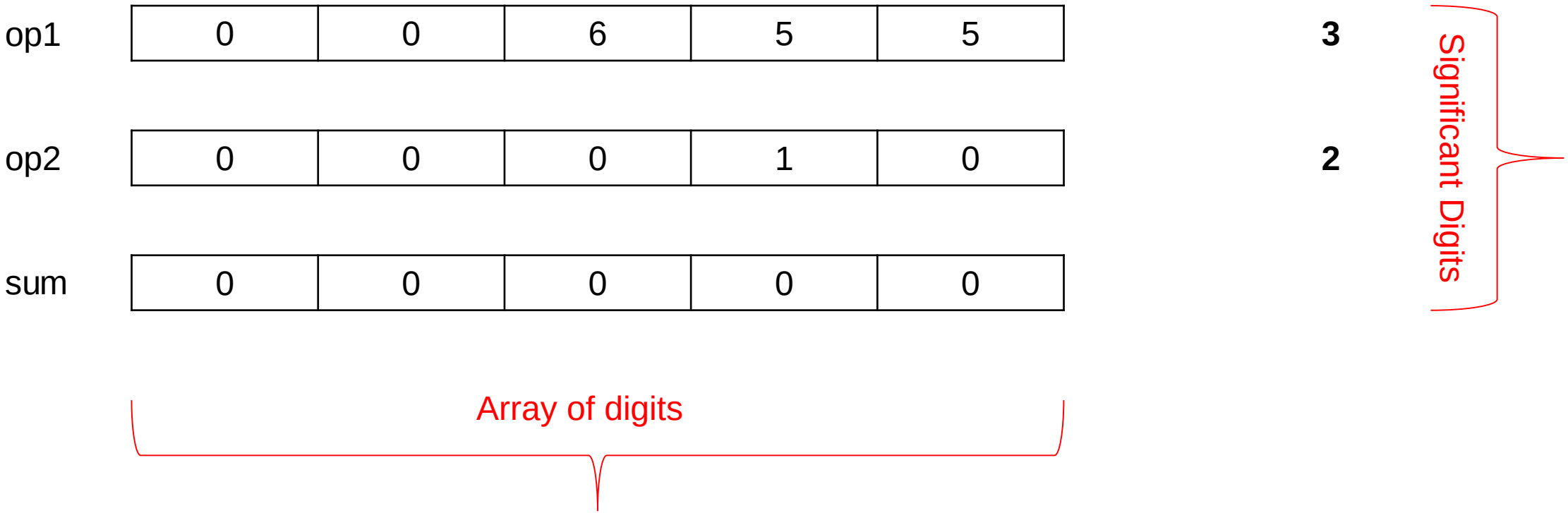
nting integers with Arrays:
another what if?



	1					carry
op1	9	9	6	5	5	**5**
op2	0	8	0	1	0	**4**
sum	**0**	**7**	**6**	**6**	**5**	**5**

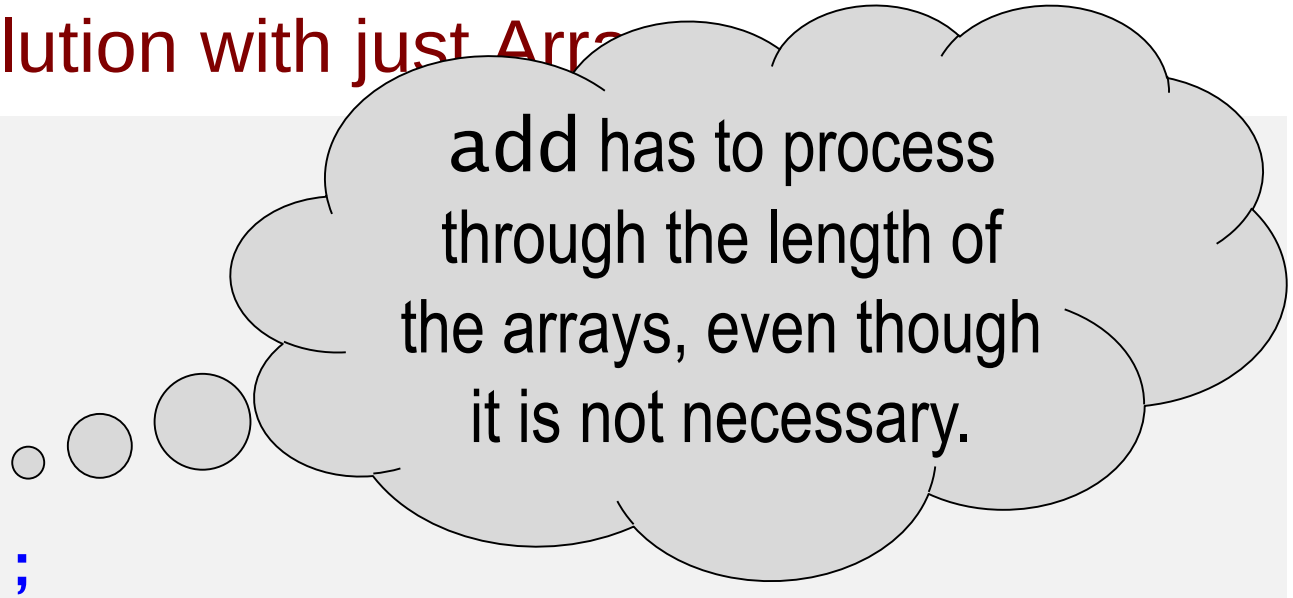
Significant Digits

Representing integers with Arrays



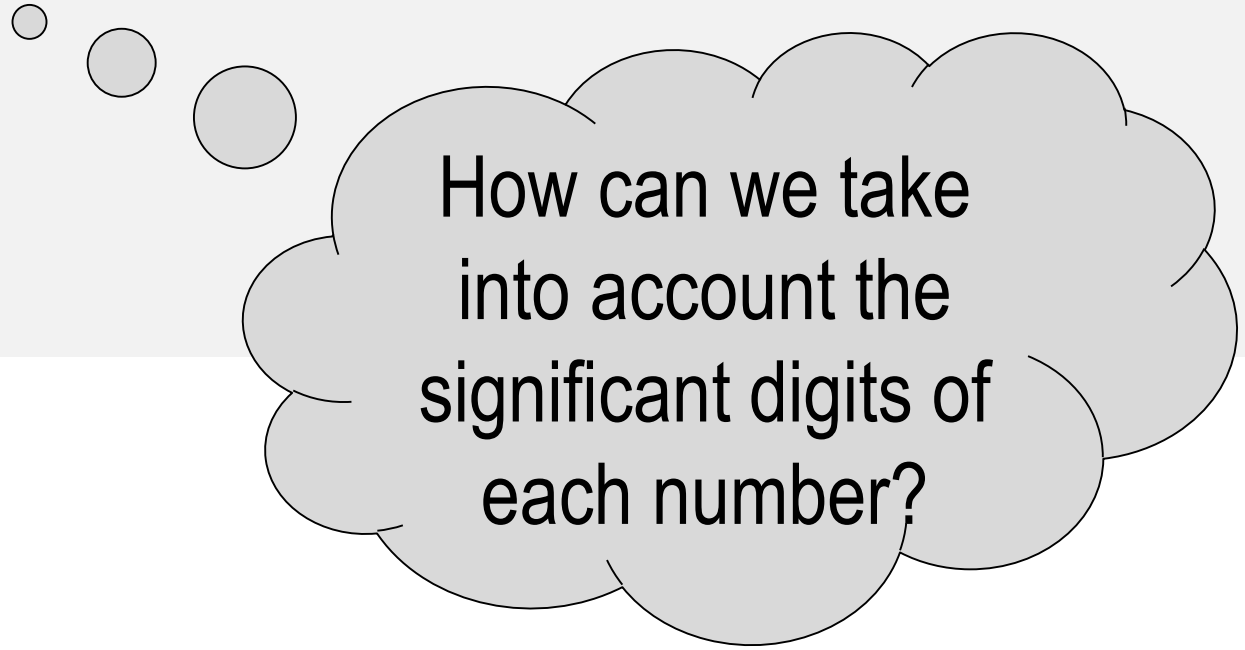
Procedural solution with just Arra

```
{

    int[] op1 = {0,0,3,4,5};

    int[] op2 = {0,0,0,9,8};

    int [] sum = add(op1, op2);

}
```



Procedural solution with just Arrays

```
{

    int[] op1 = {0,0,3,4,5}, op2 = {0,0,0,9,8};



}
```



Procedural solution with just Arrays

```
{

   int[] op1 = {0,0,3,4,5}, op2 = {0,0,0,9,8};

   int s_op1 = 3; // significant digits of op1

   int s_op2 = 2; // significant digits of op2

}
```

Procedural solution with just Arrays

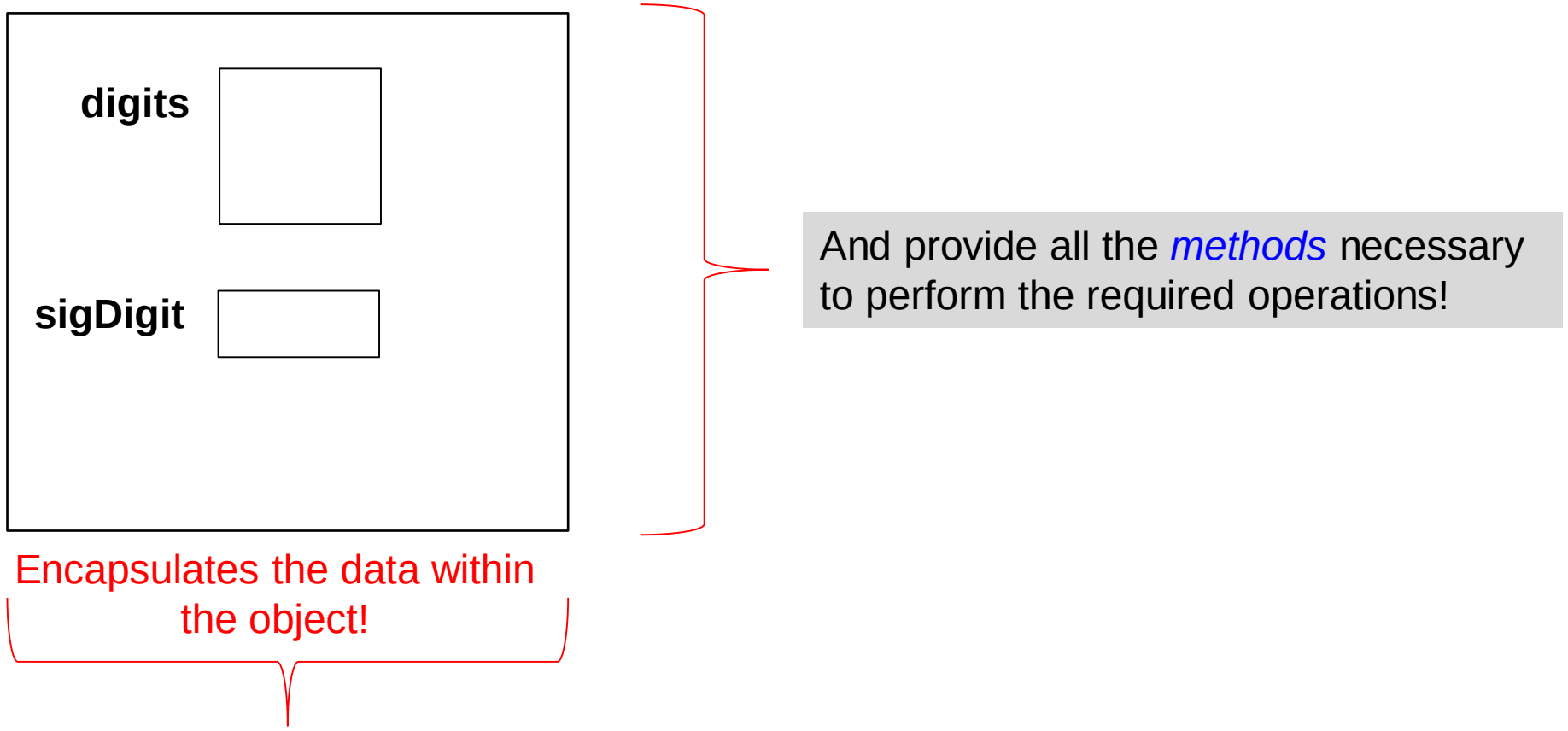
```
{

    int[] op1 = {0,0,3,4,5}, op2 = {0,0,0,9,8};

    int s_p1 = 3, s_op2 = 2;

    int[] sum = add(op1, s_p1, op2, s_op2);

}
```

Procedural solution with just Arrays

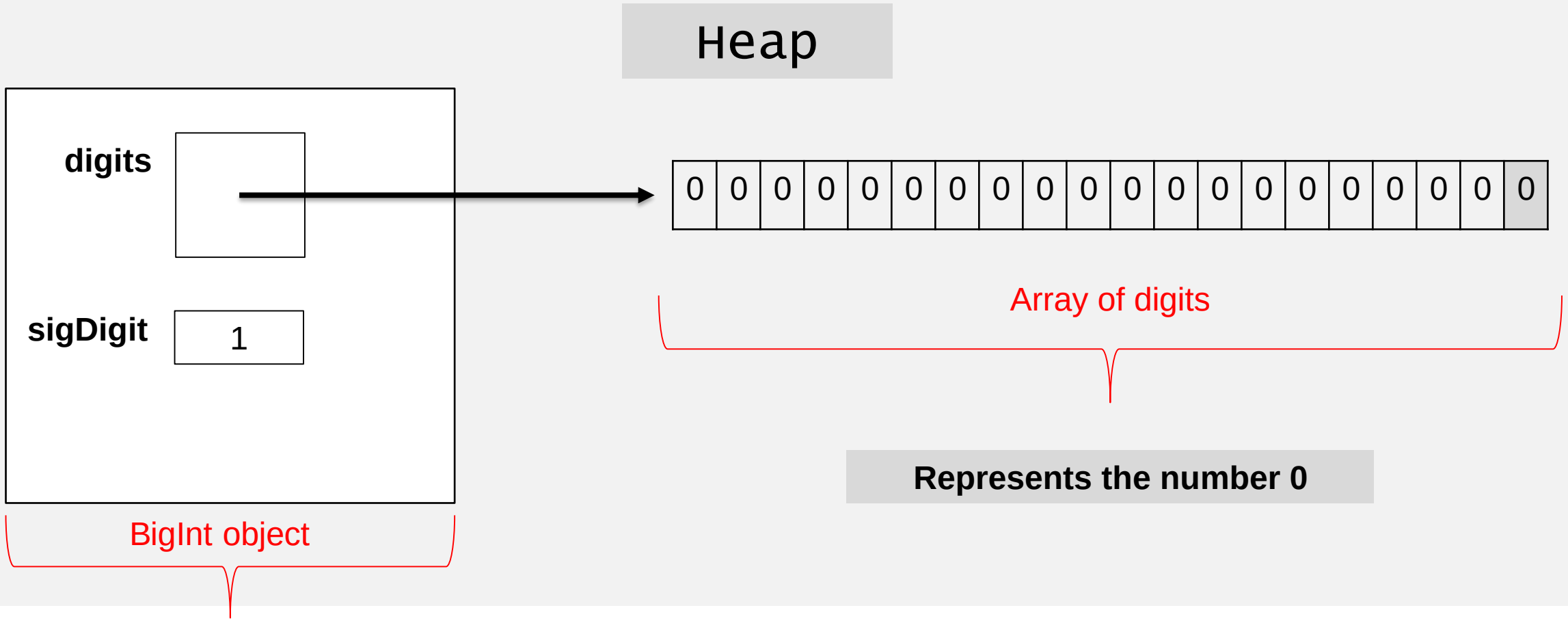
```
{
    int[] op1 = {0,0,3,4,5}, op2 = {0,0,0,9,8};
    int s_p1 = 3, s_op2 = 2;
    int[] sum = add(op1, s_p1, op2, s_op2);
}
```

```
public static int[] add(int[] o1, int s1, int[] o2, int s2) {
    int[] sum = {0,0,0,0,0};
    . . . .
    return sum;
}
```

Object Oriented Design:
Encapsulating the data into a **BigInt** object

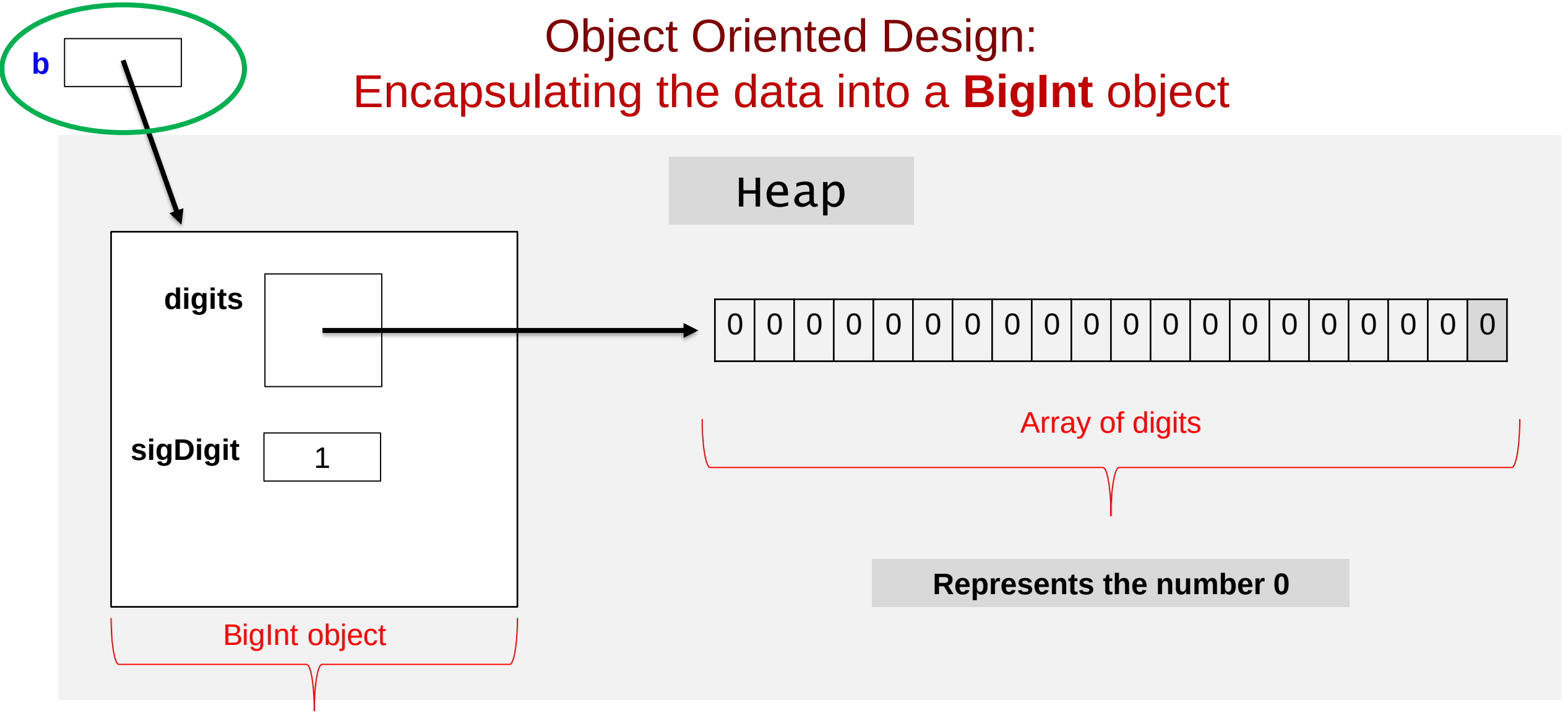


Object Oriented Design:
Encapsulating the data into a **BigInt** object



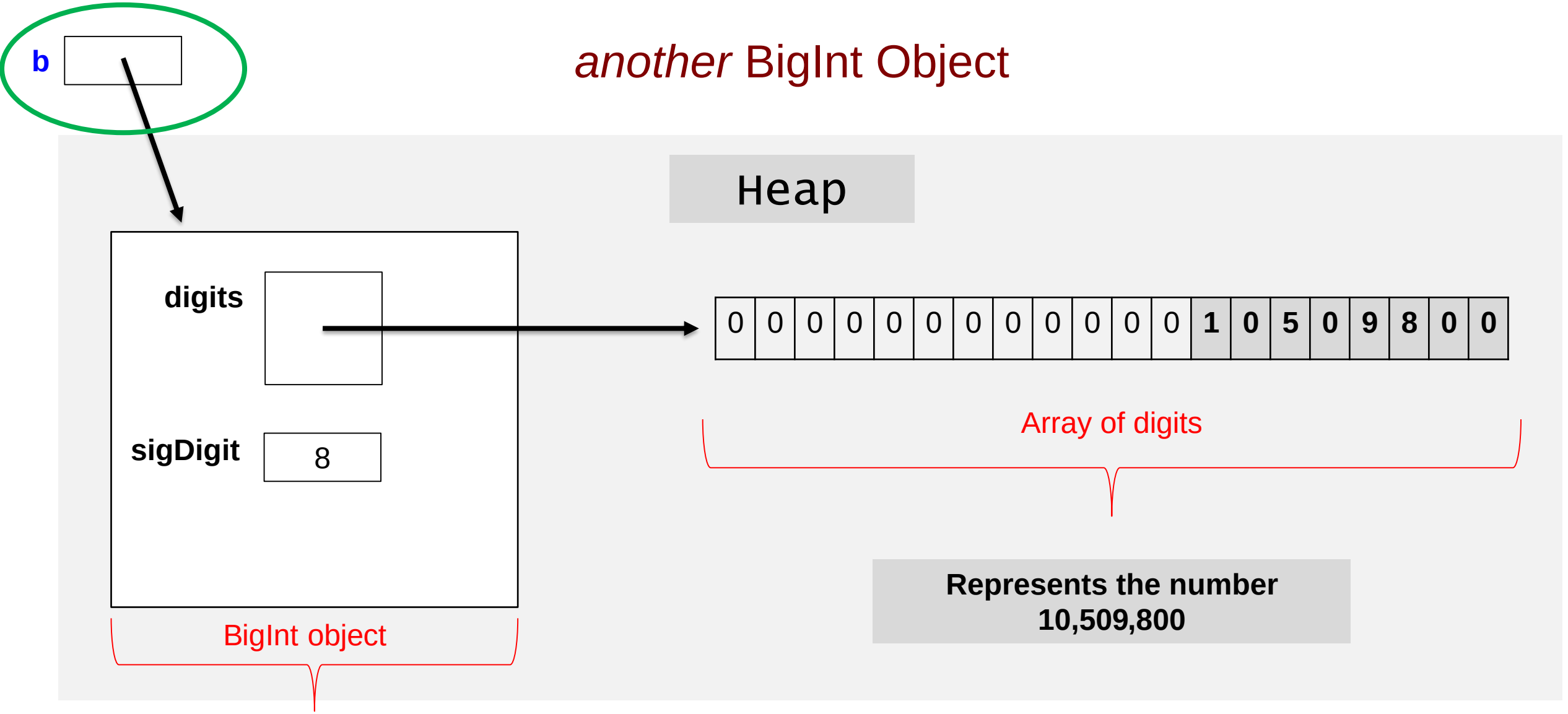

```
BigInt b = new BigInt();  // no-arg constructor
```

Object Oriented Design:
Encapsulating the data into a **BigInt** object




```
BigInt b = new BigInt();  // b references the object
```

another BigInt Object




```
BigInt b = new BigInt(10509800); // custom constructor
```

Programming with Objects:

Mathematical Operations

```
{

    BigInt b = new BigInt();        // should create a BigInt object representing 0

    System.out.println( b );        // output the contents - need toString()









}
```

Programming with Objects:

Mathematical Operations

```
{

    BigInt b = new BigInt();       // should create a BigInt object representing 0

    System.out.println( b.toString() );    // this is the call Java will make…









}
```

Programming with Objects:

Mathematical Operations

```
{
    BigInt b = new BigInt();        // should create a BigInt object representing 0

    System.out.println( b );        // output the contents of the object

    int[] arr = {1,2,3};          // array representing the number 123

    b = new BigInt( arr );        // create a new BigInt object from the array passed

    System.out.println( b );









}
```

Programming with Objects:

Mathematical Operations

```
{
    BigInt b = new BigInt();        // should create a BigInt object representing 0

    System.out.println( b );        // output the contents of the object

    int[] arr = {1,2,3};          // array representing the number 123

    b = new BigInt( 3567 );     // or, create a new BigInt object from the integer passed

    System.out.println( b );









}
```

Programming with Objects:

Mathematical Operations

```
{
    BigInt op1 = new BigInt(13456);     // create a BigInt object representing 13,456

    BigInt op2 = new BigInt( 223 );     // create a BigInt object representing 223

    BigInt sum = op1.add(op2);    // Compute the sum and store in another Bigint object

    System.out.println( op1 + “+” + op2 + “=“ + sum );









}
```

Baby steps…