

CS 112



How to approach programming problems:

The BigInt

Spring 2021

Christine Papadakis-Kanaris

How to Approach Programming Problems

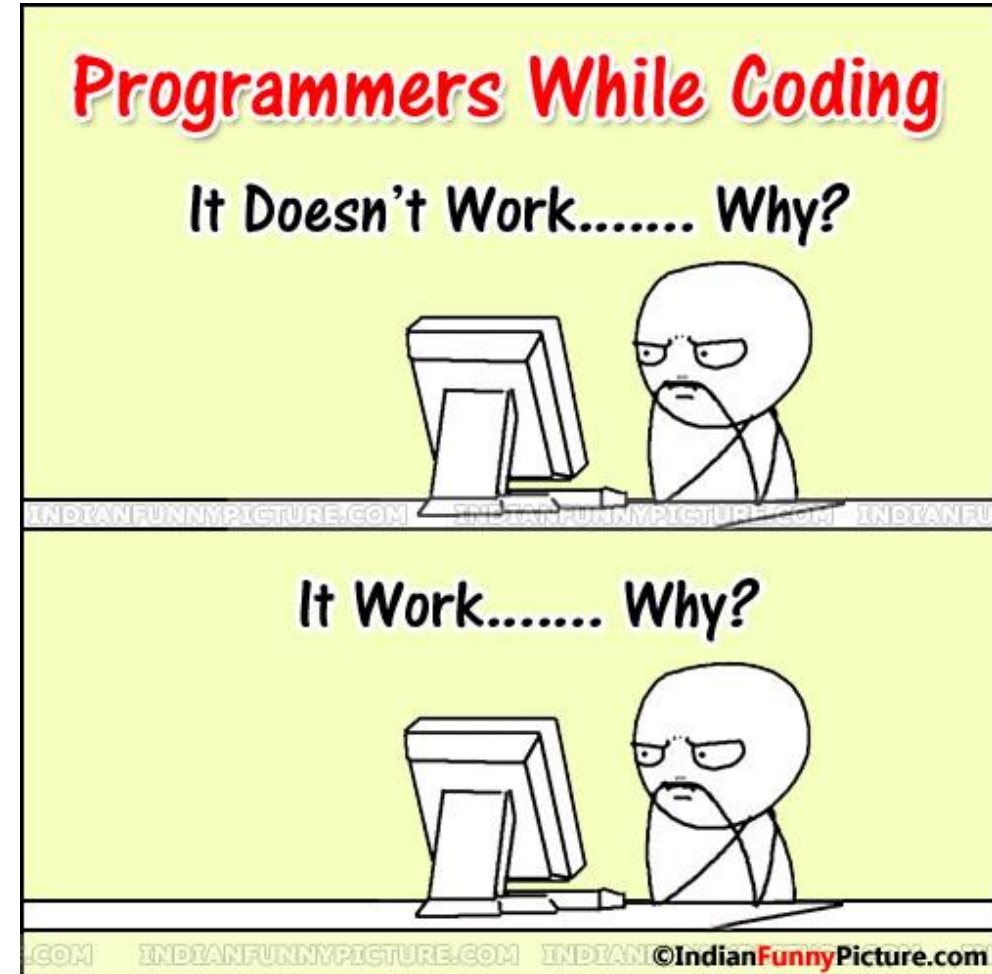
What not to do:



How to Approach Programming Problems

What not to do:

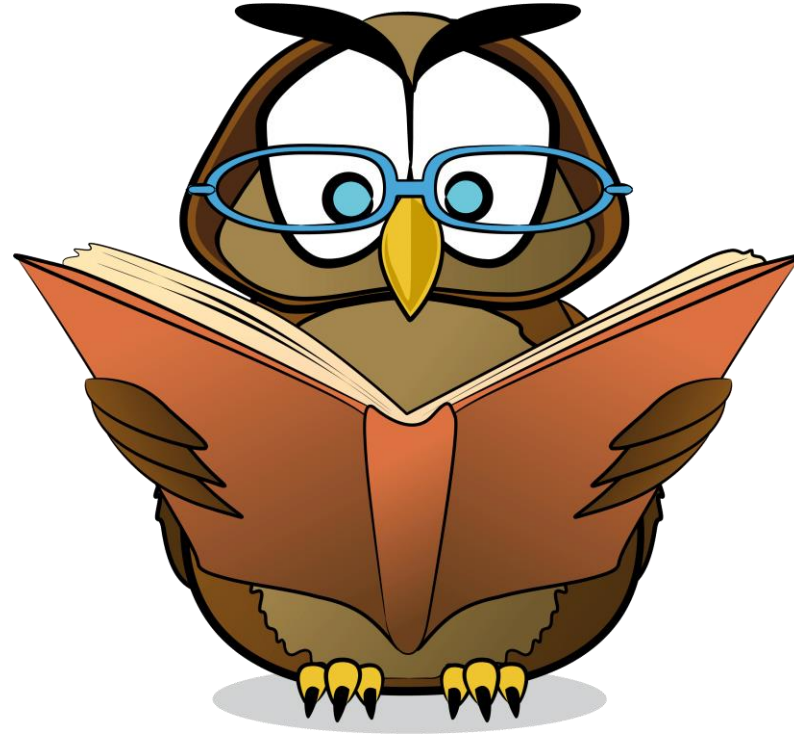
Trial and Error
Programming!



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:

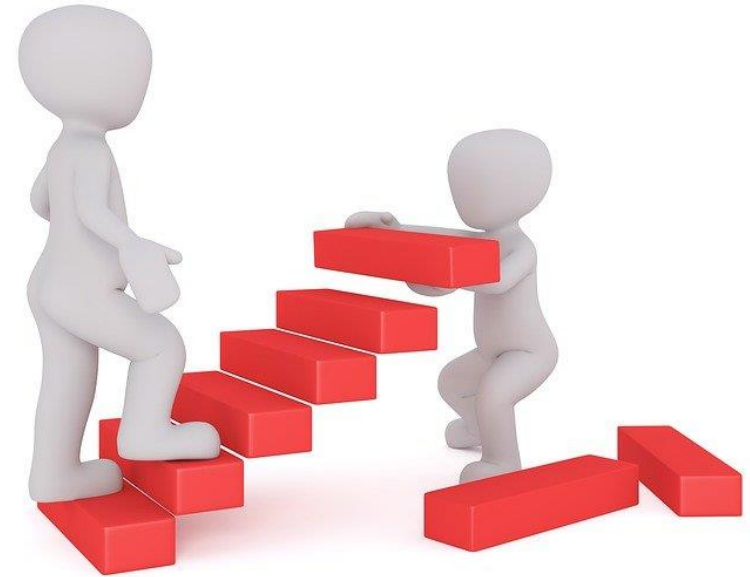
#2
Form the
solution



How to Approach Programming Problems

What to do:

#2
Identify the
BASIC steps



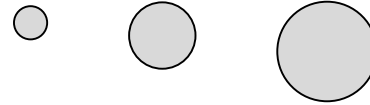
Understanding BigInt

- 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

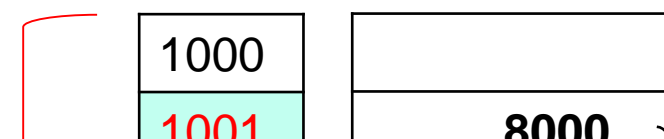
- We know that the largest integer we can store in a primitive variable in Java is an unsigned long.
- How can we represent very large integers in Java?
- How is it that Python does not have this issue?

```
number = 5  
number = 50000  
number = 5000000000000
```



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

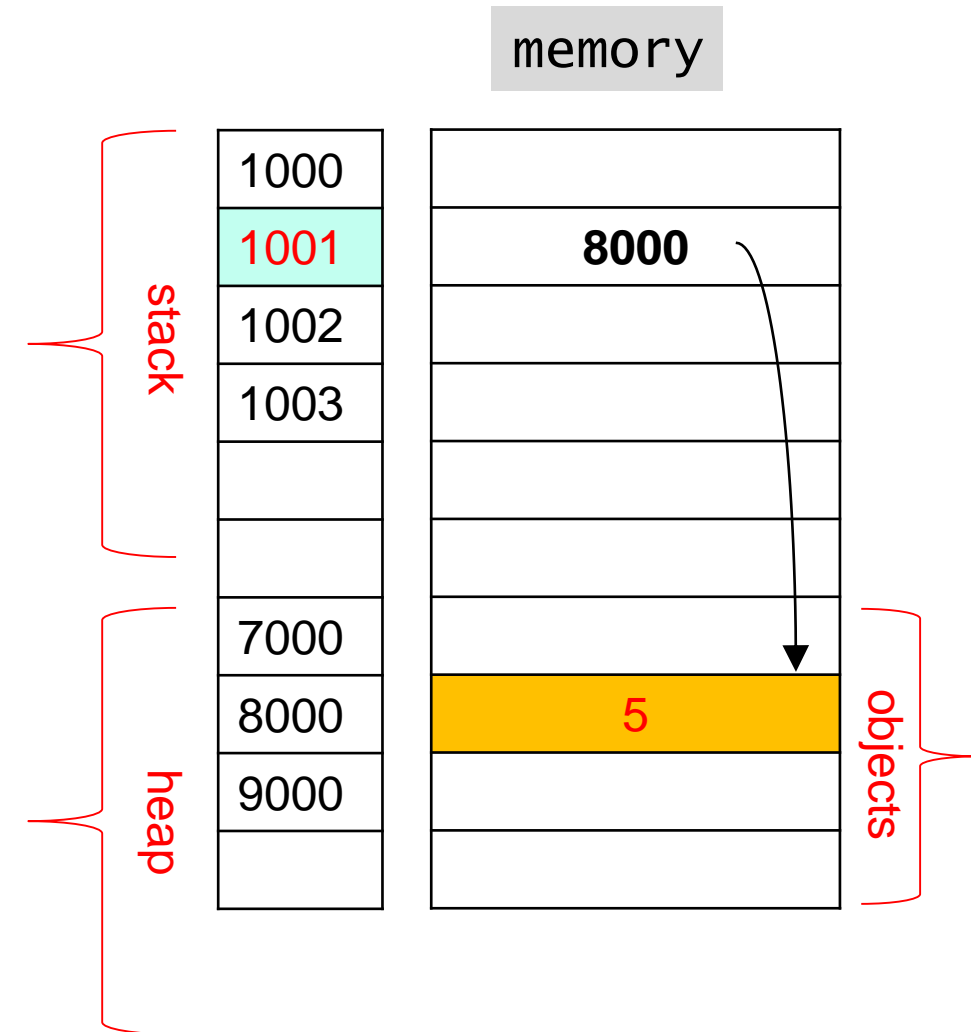
Understanding BigInt

- 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?
- 
- | |
|------|
| 1000 |
| 1001 |
- | |
|------|
| |
| 8000 |
- memory

```
number = 5
number = 50000
number = 50000000000000
```

mapping

number	1001
--------	------



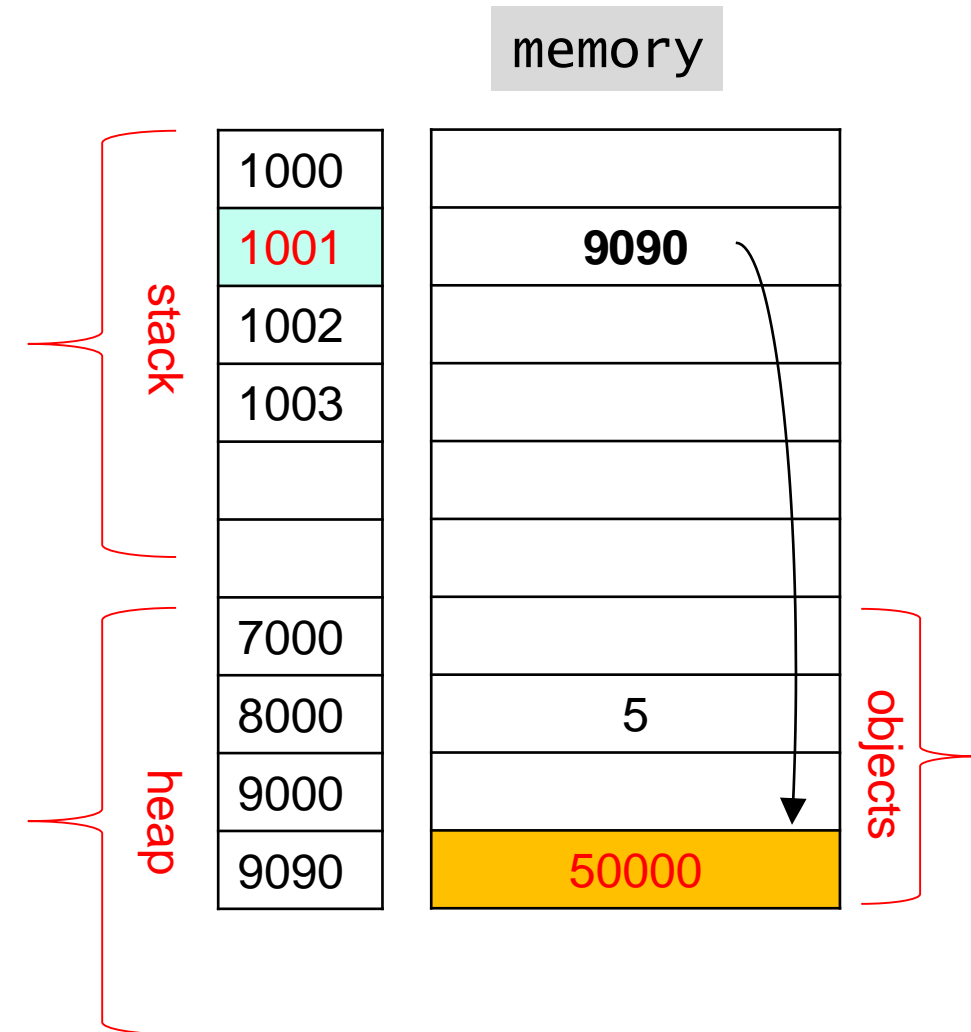
Understanding BigInt

- 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 = 500000000000
```

mapping

number	1001
--------	------



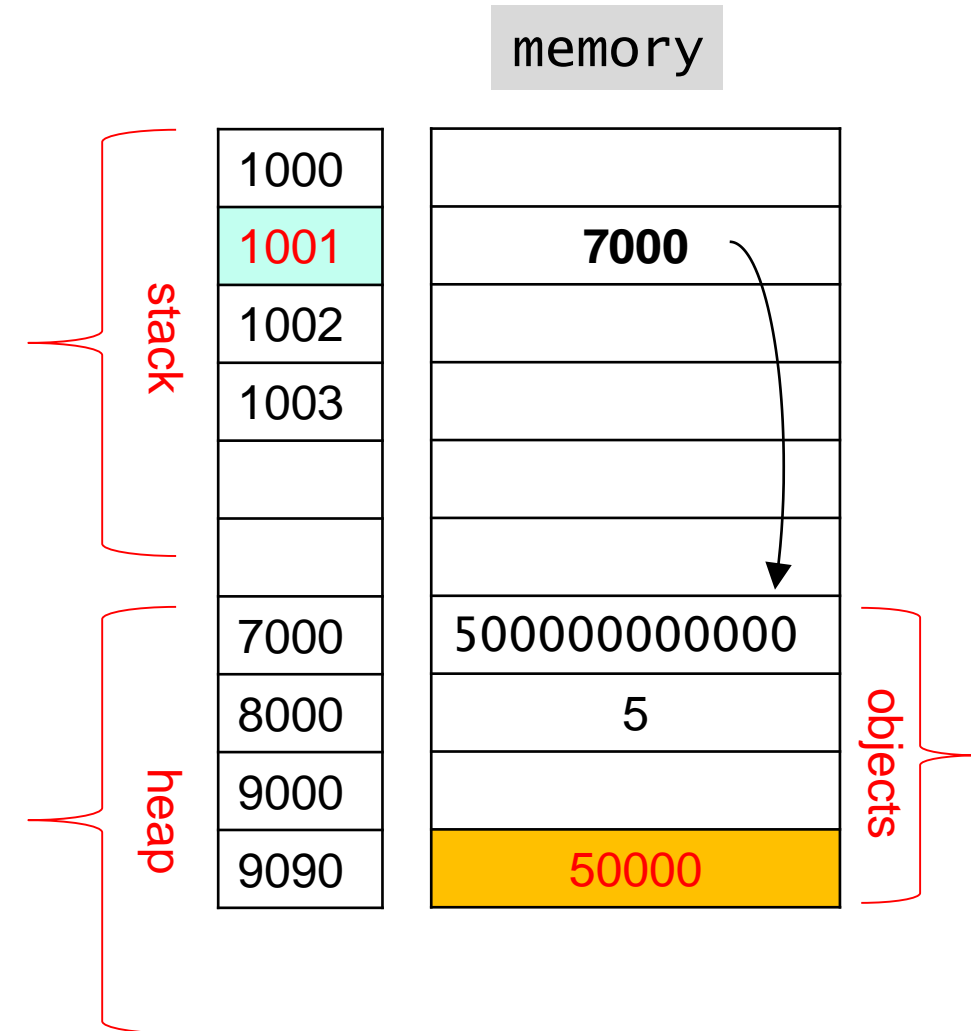
Understanding BigInt

- 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 = 500000000000
```

mapping

number	1001
--------	------



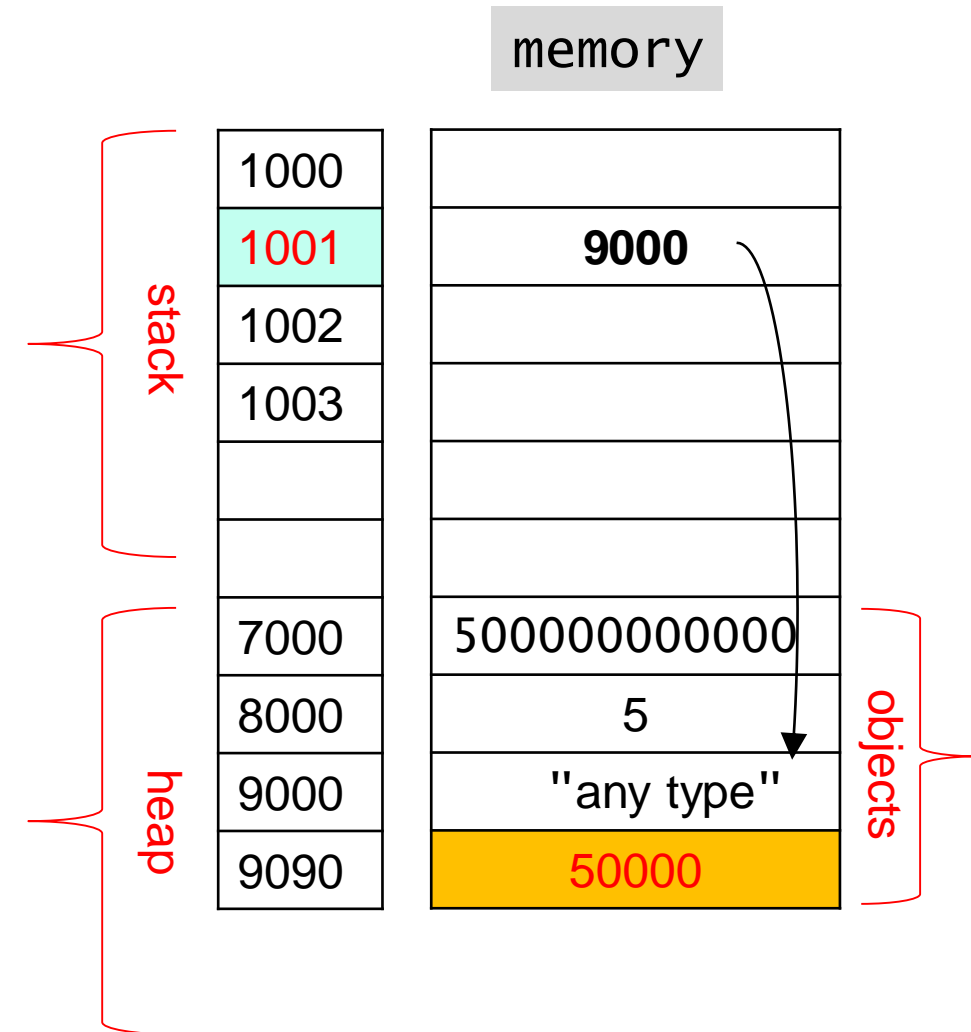
Understanding BigInt

- 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 = "any type"
```

mapping

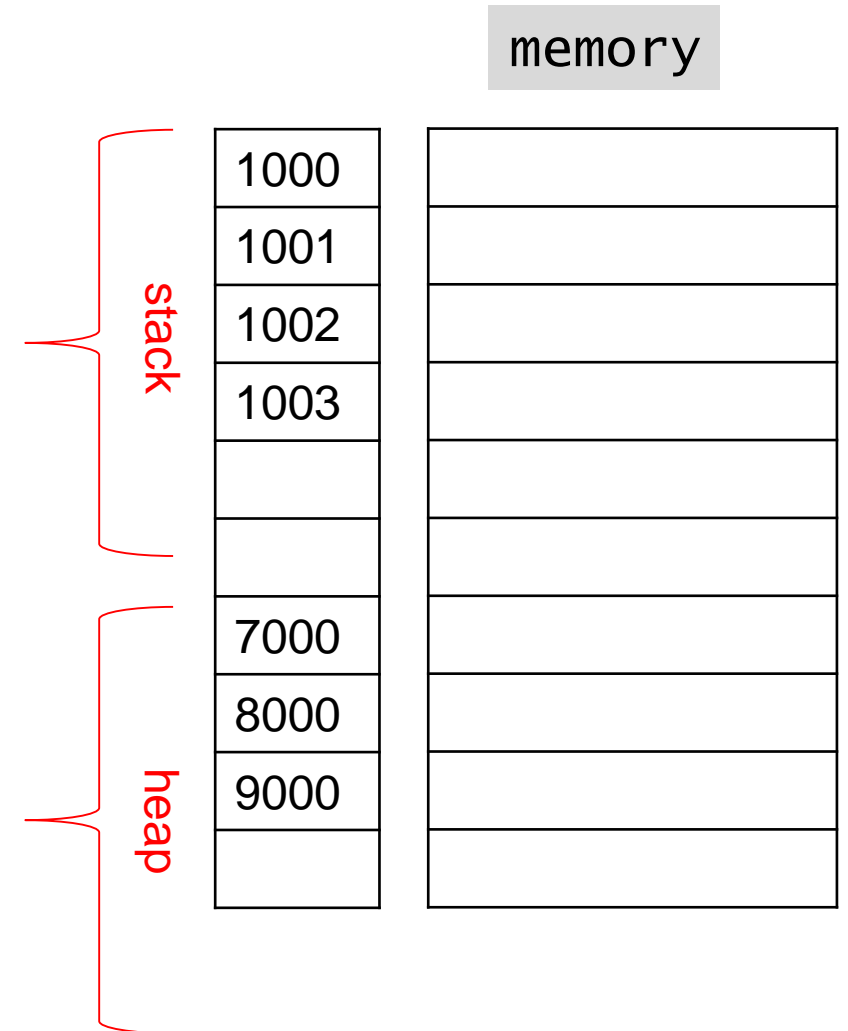
number	1001
--------	------




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 = 5000000000000;
```



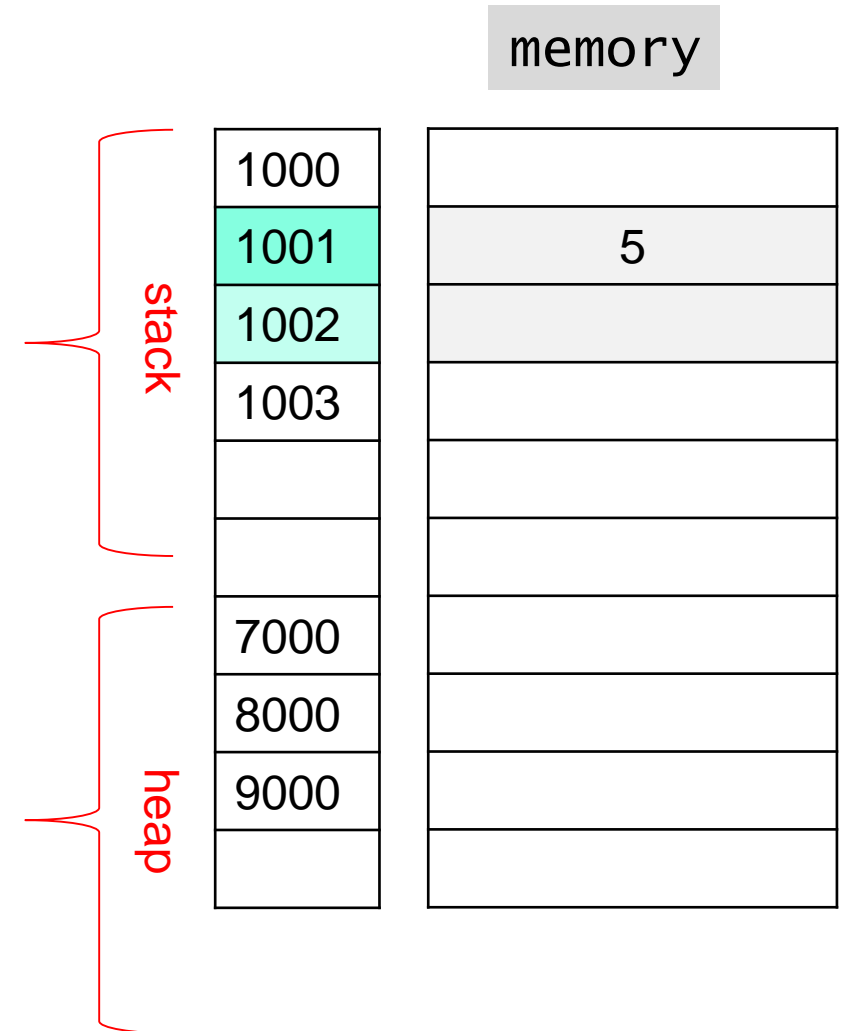
Understanding BigInt

- 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.
- 
- The diagram illustrates a memory layout. On the right, a grey box is labeled "memory". Below it, a stack of boxes is shown. A red bracket on the left indicates a sequence of boxes. The first box is white and contains the number "1000". Below it is a cyan box containing the number "1001". The second box is white and contains the number "5" in its bottom half, with its top half being empty.

```
int number = 5; // assume 2 bytes per integer
number = 50000;
number = 500000000000000;
```

mapping

number	1001
--------	------



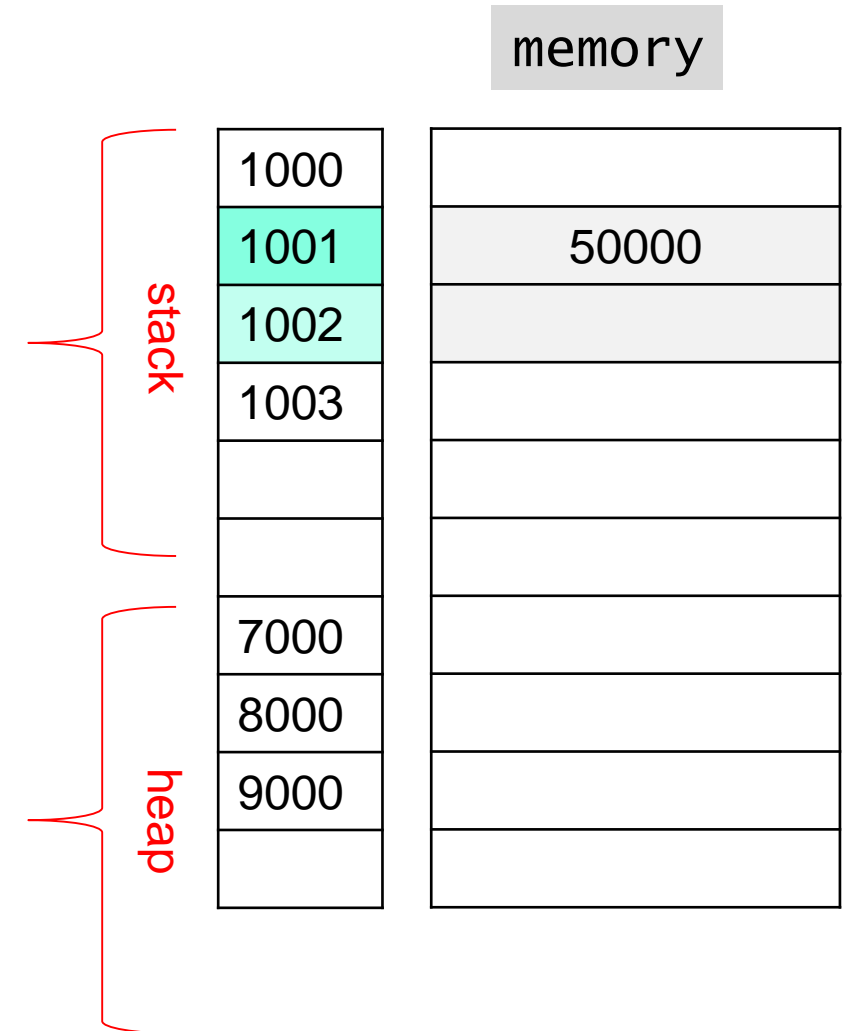
Understanding BigInt

- 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 = 5000000000000;
```

mapping

number	1001
--------	------



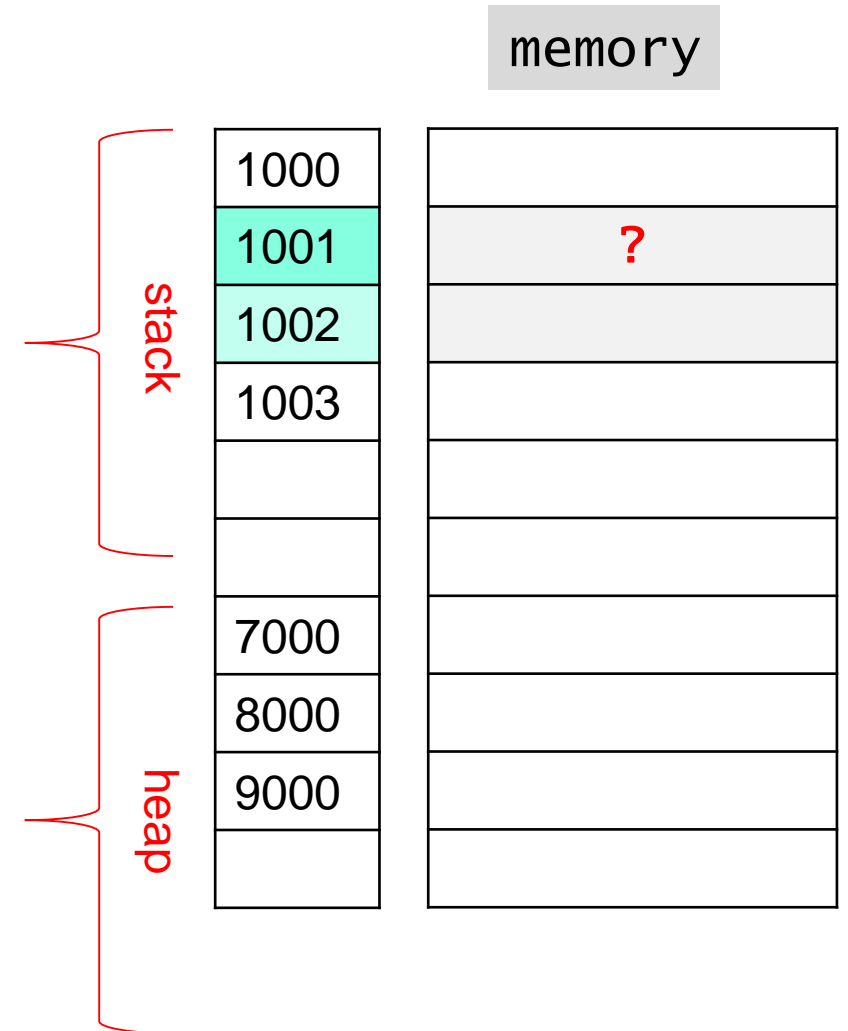
Understanding BigInt

- 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.
- 
- memory
- | | |
|------|---|
| 1000 | |
| 1001 | ? |

```
int number = 5; // assume 2 bytes per integer
number = 50000;
number = 500000000000000;
```

mapping

number	1001
--------	------



Understanding BigInt

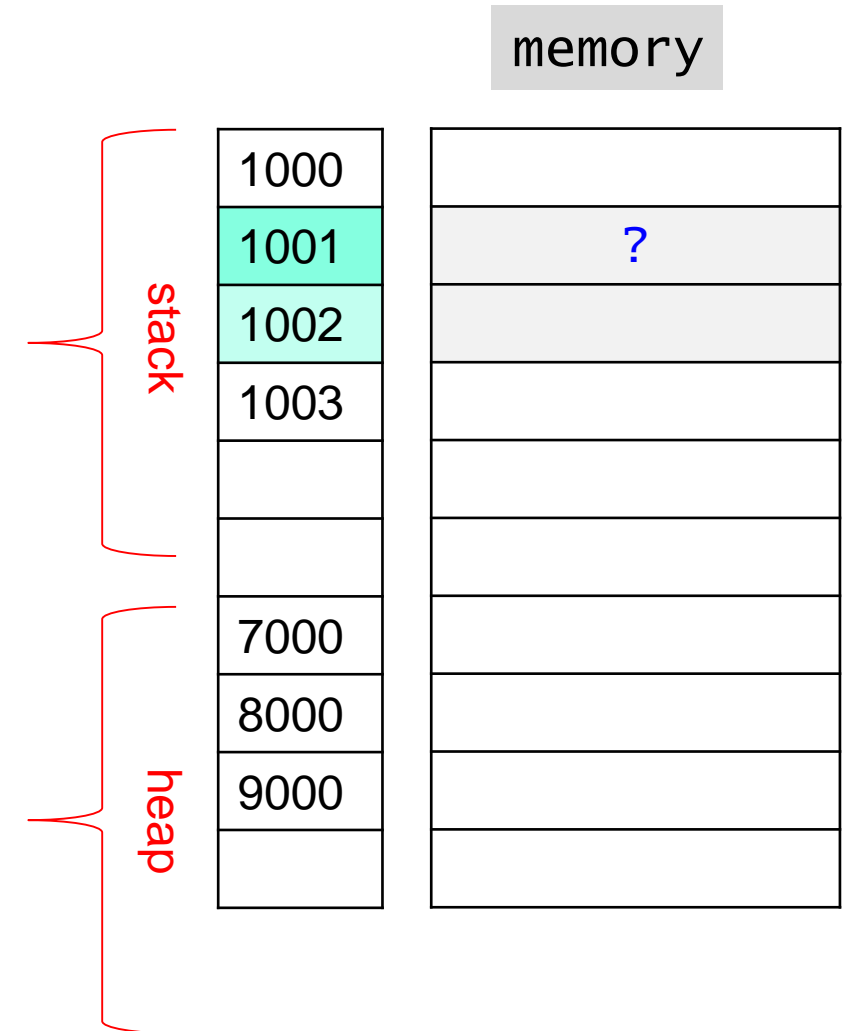
- We know that the largest integer we can store in a primitive variable in Java is an unsigned long.
- How can we represent very large integers in Java?
- Java is a strongly typed compiled language.

```
int number = 5; // assume 2 bytes per integer
number = 50000;
number = 500000000000000;
number = 10.34; // needs double the bytes
```



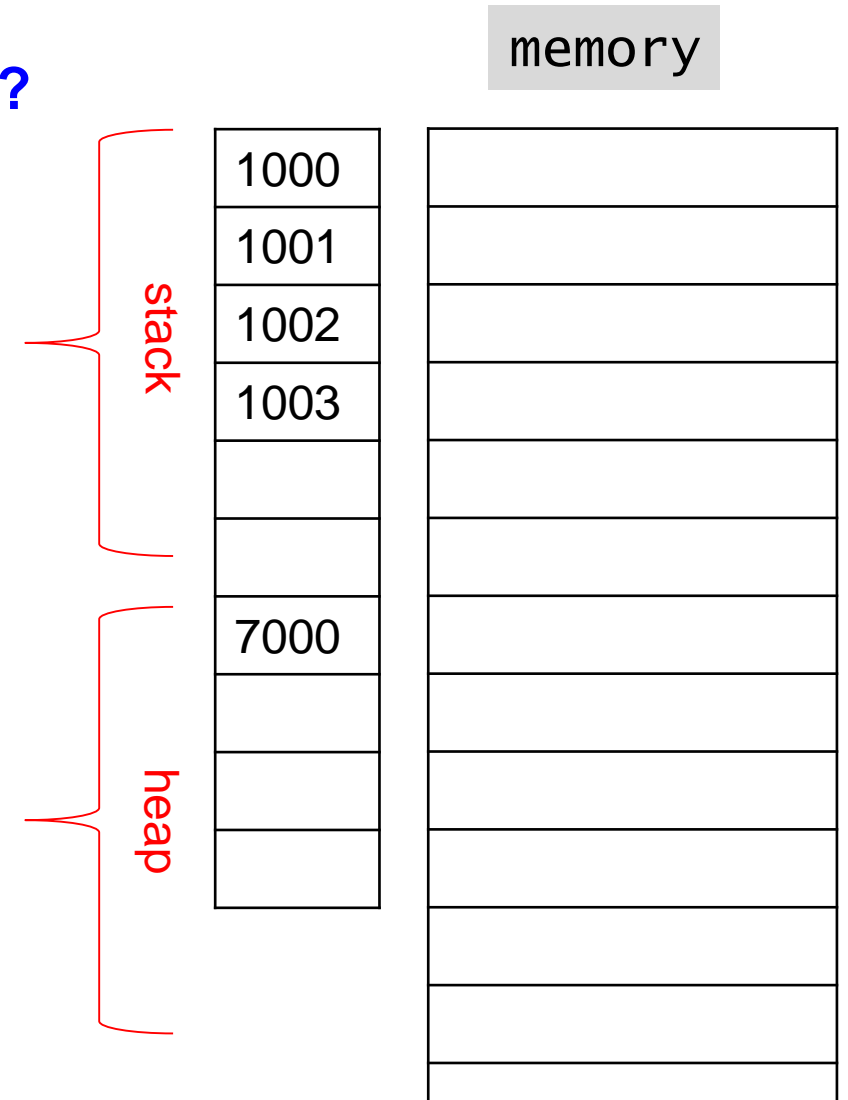
mapping

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!
-
- The diagram illustrates how a large integer is stored in memory. A red bracket groups two boxes containing the binary values '1000' and '1001'. To the right of these is a larger box labeled 'memory', representing the storage for the entire BigInteger object.



BigInteger

Understanding BigInt

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

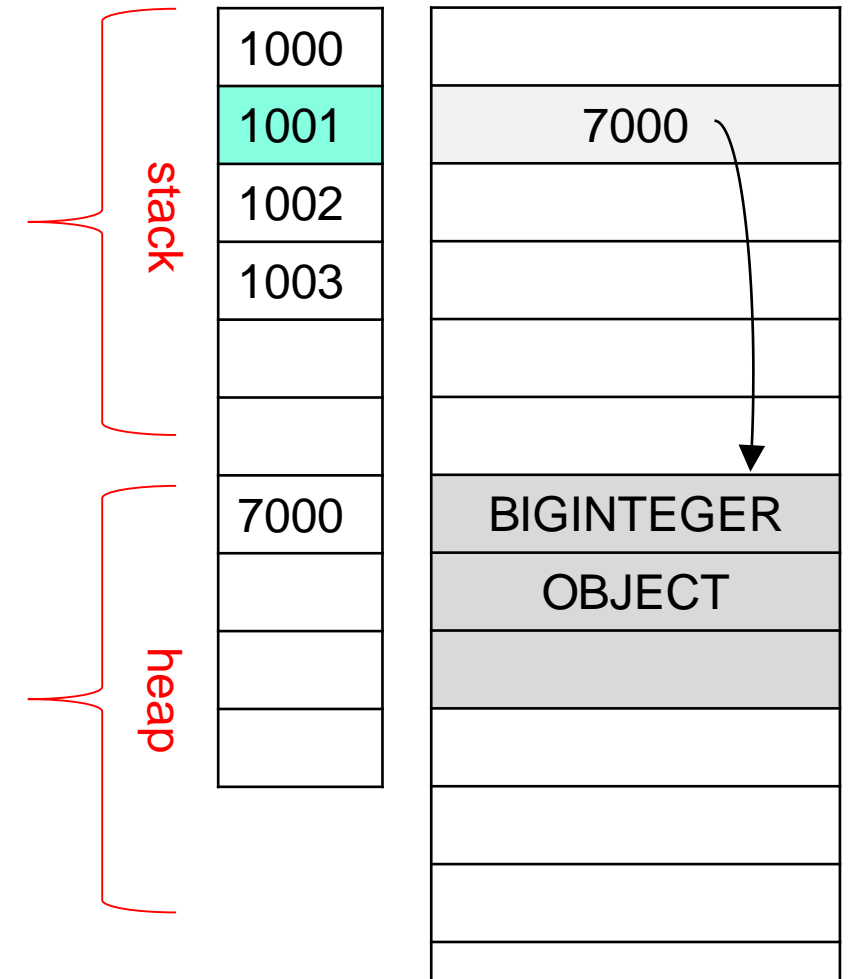
- **How can we represent very large integers in Java?**

- **Java's BigInteger class!** `BigInteger`

`BigInteger number = new BigInteger(5000000);`

mapping

number	1001
--------	------



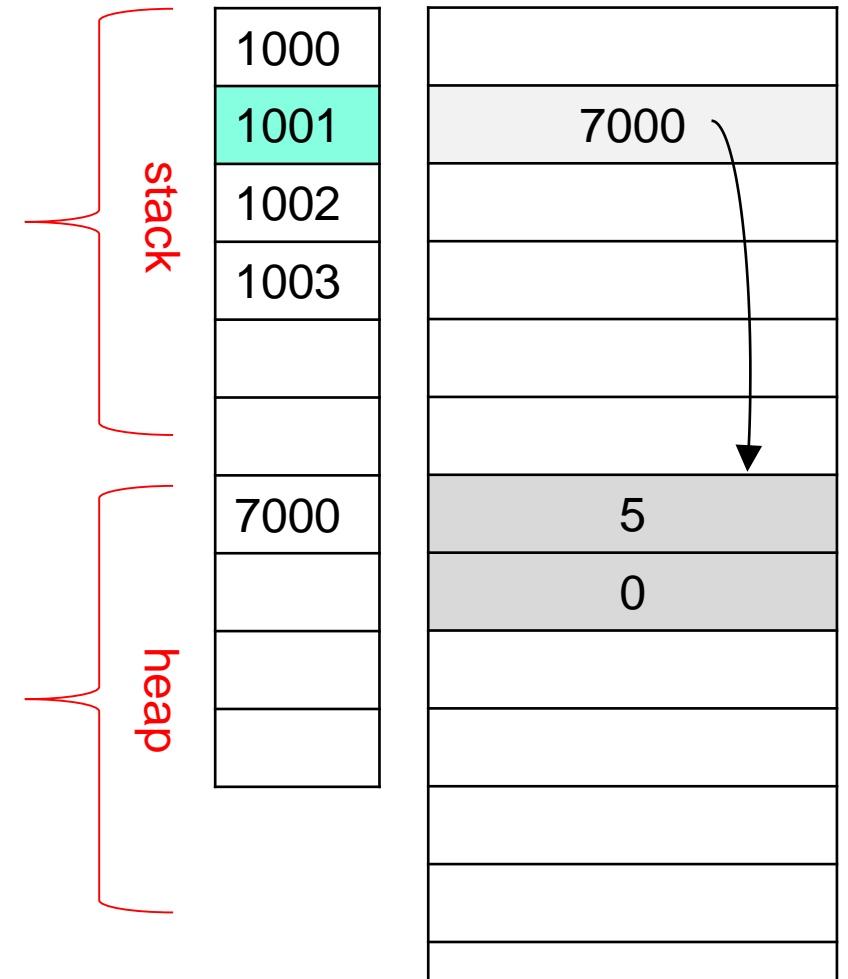
Understanding BigInt

- 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
--------	------



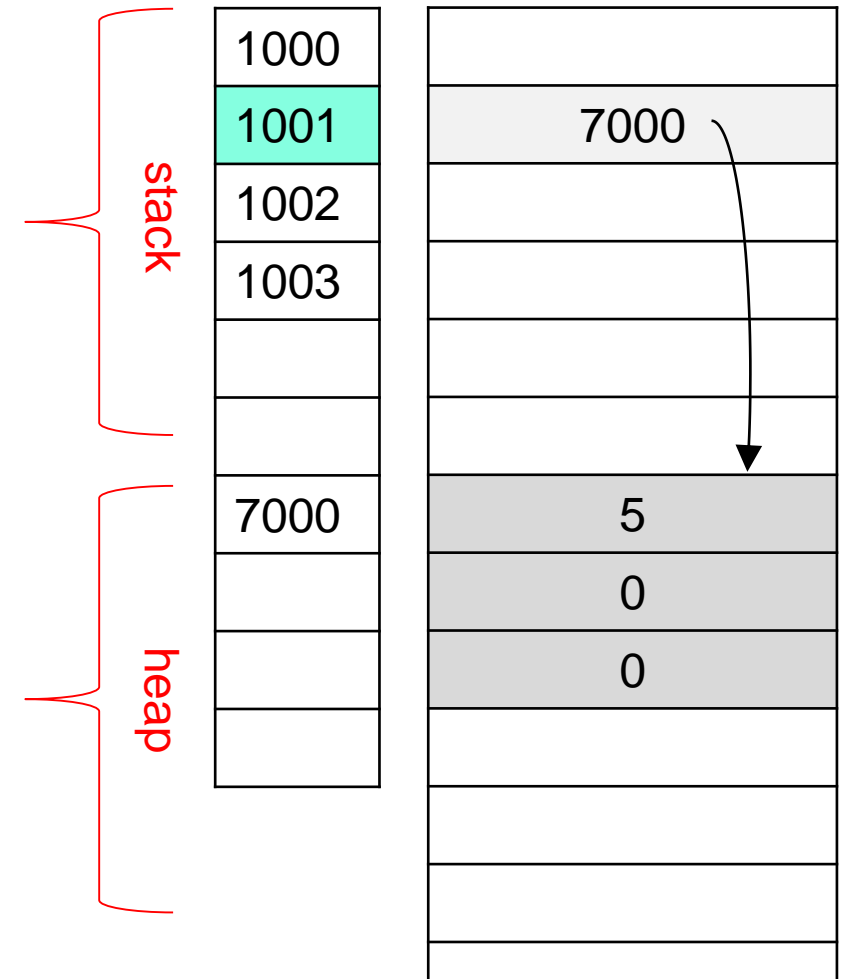
Understanding BigInt

- 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.

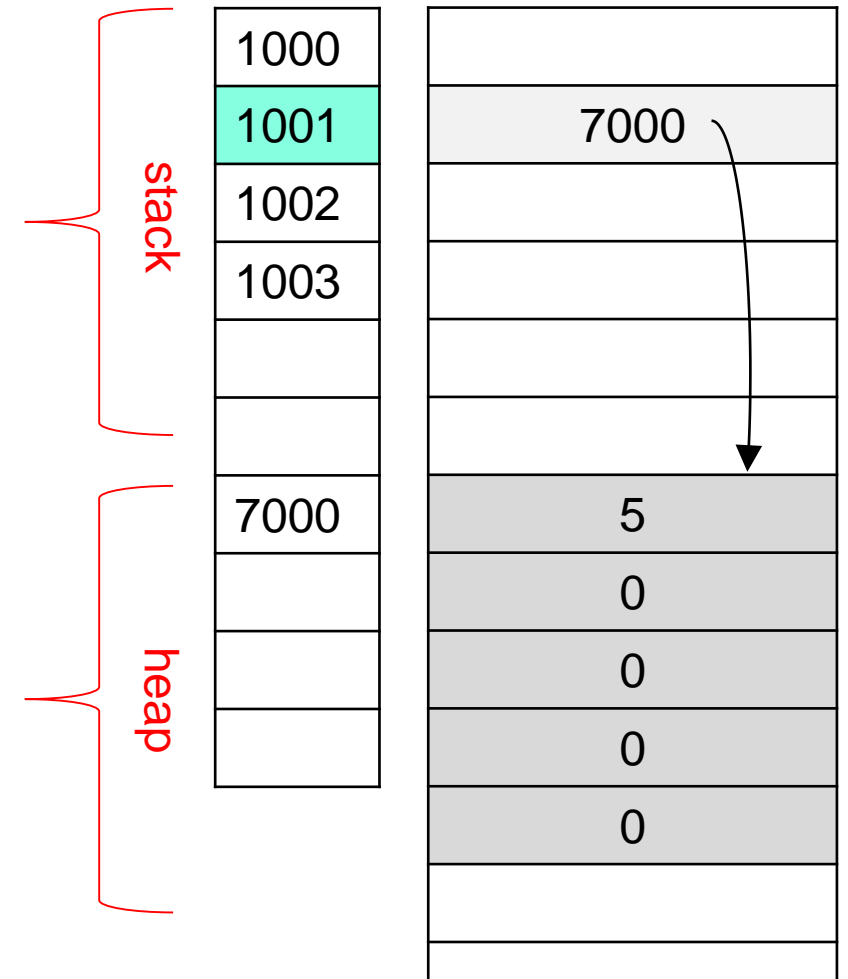
- **How can we represent very large integers in Java?**

- **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.

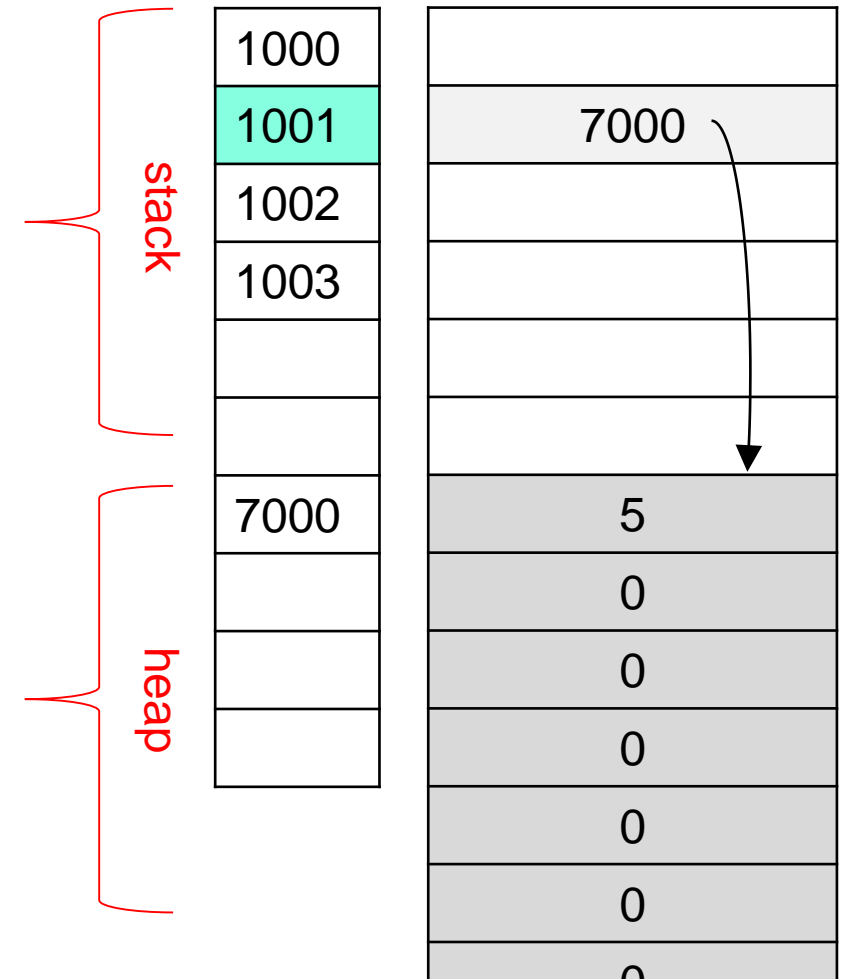
- **How can we represent very large integers in Java?**

- **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

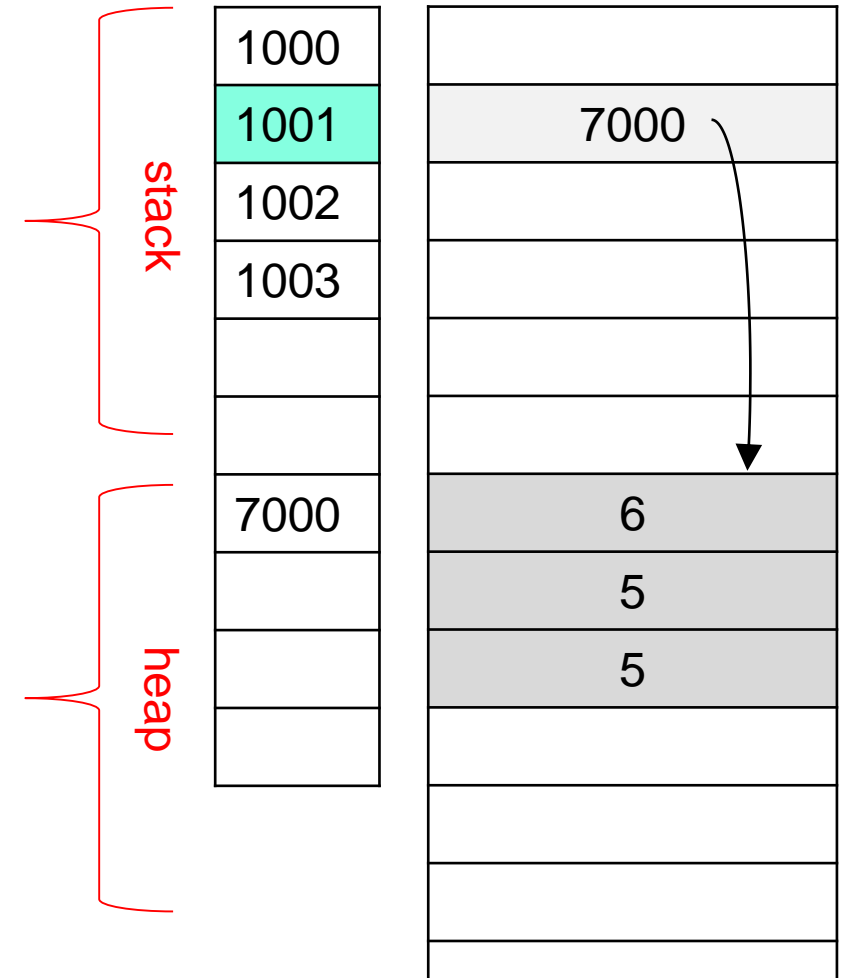
- We know that the largest integer we can store in a primitive variable in Java is an unsigned long.
- **How can we represent very large integers in Java?**
- **Using an array to represent an integer!**

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

3 significant
digits

mapping

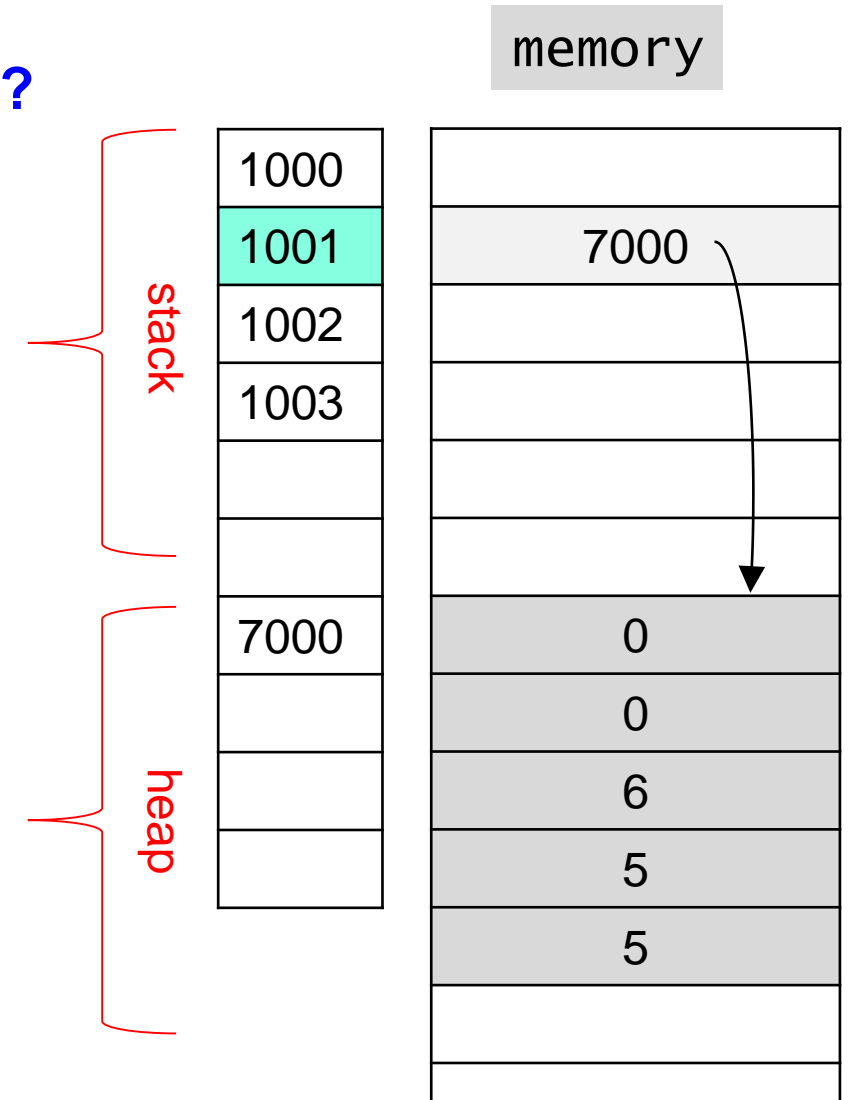
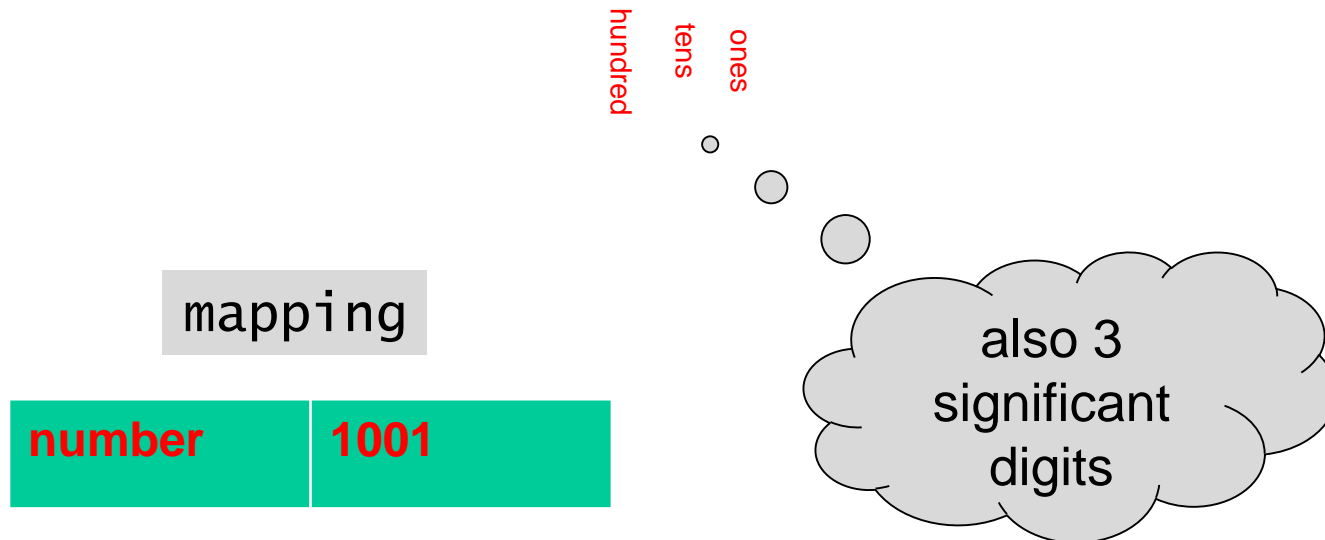
number	1001
--------	------



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};
```

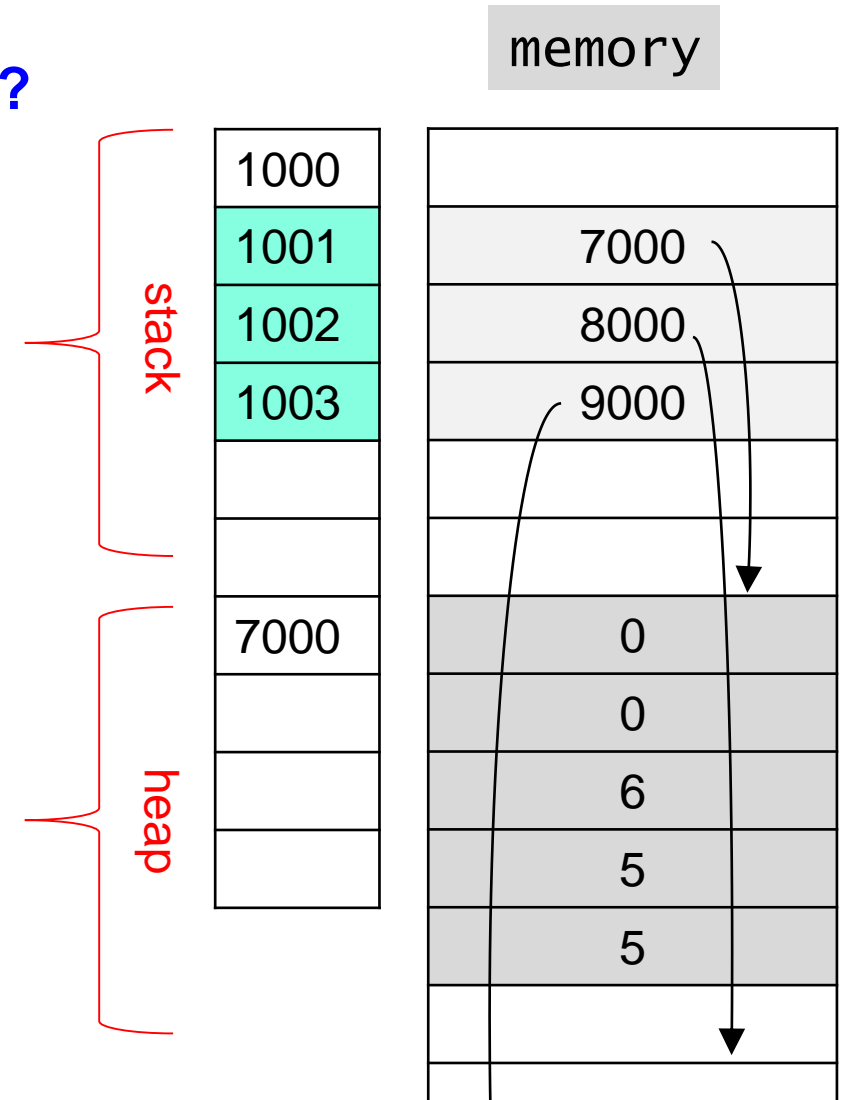


Understanding BigInt

- 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!**
-
- The diagram illustrates memory representation. A red bracket groups the first two boxes (1000 and 1001) under the 'memory' label. The third box contains '7000'.

```
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



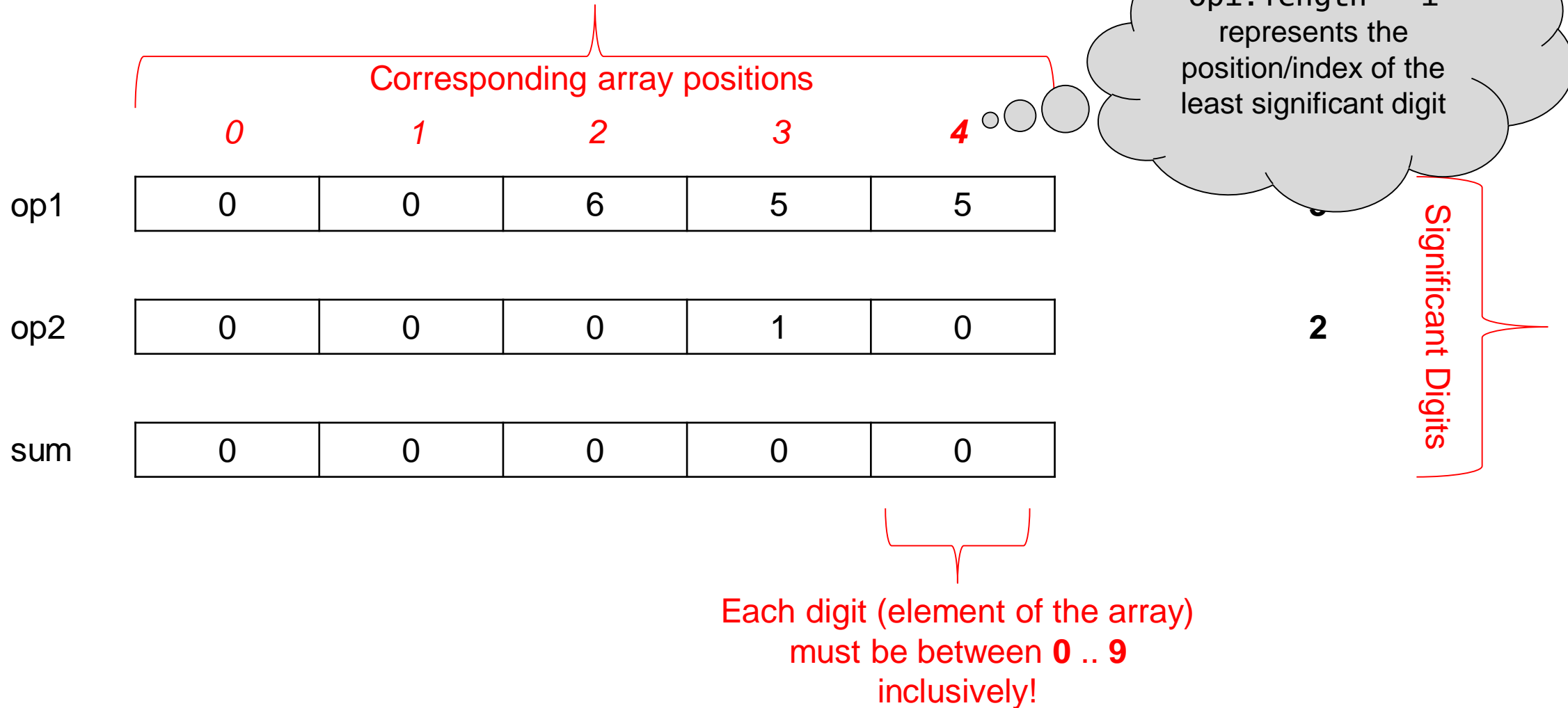
Representing integers with Arrays

	<i>ten thousands</i>	<i>thousands</i>	<i>hundreds</i>	<i>tens</i>	<i>ones</i>	
op1	0	0	6	5	5	3
op2	0	0	0	1	0	2
sum	0	0	0	0	0	

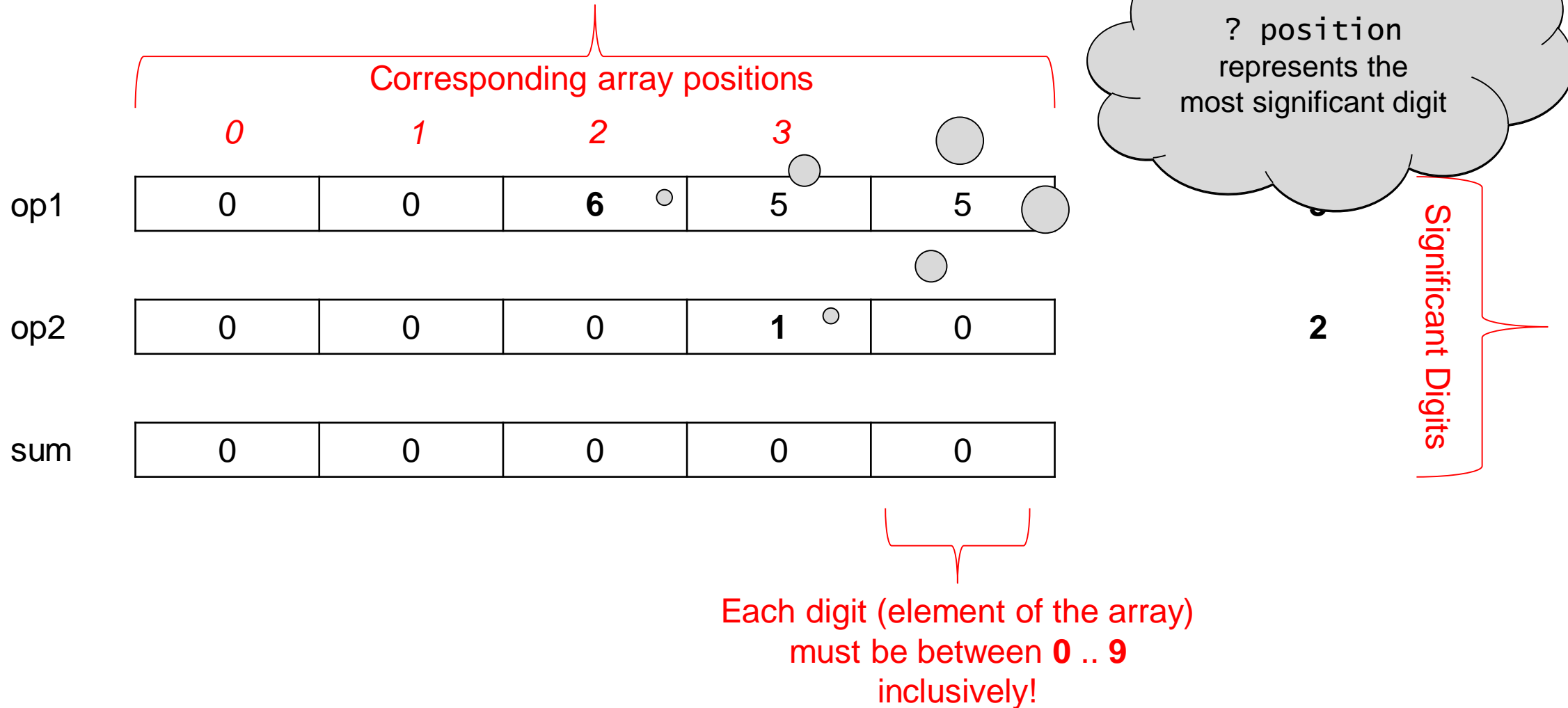
Significant Digits

Each digit (element of the array)
must be between **0 .. 9**
inclusively!

Representing integers with Arrays



Representing integers with Arrays



Representing integers with Arrays

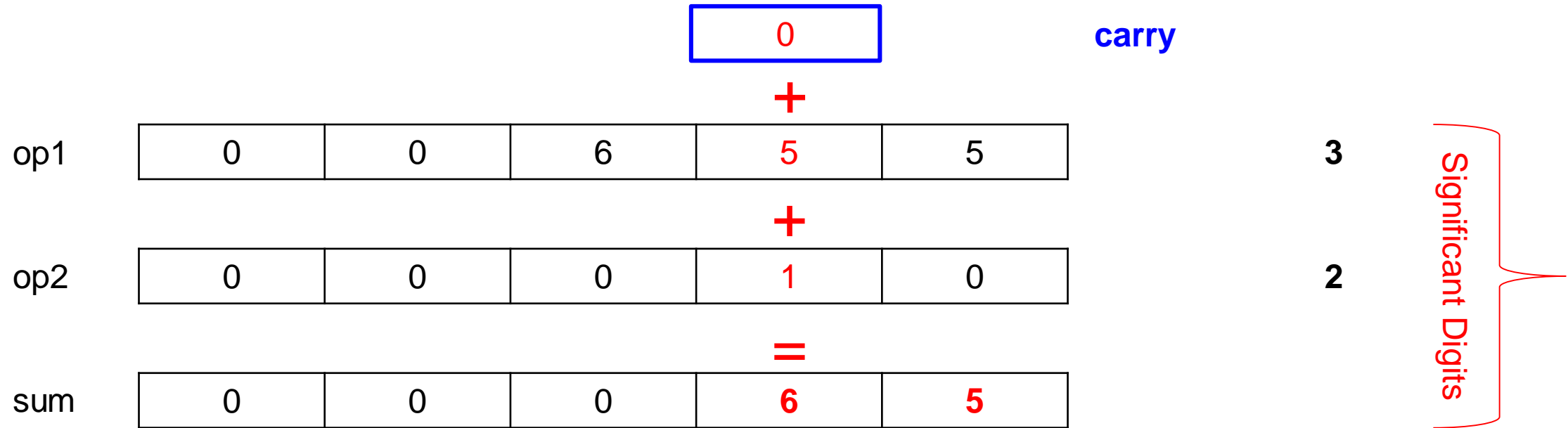
					<div>0</div>	carry		
op1	0	0	6	5	5		3	Significant Digits
op2	0	0	0	1	0		2	
sum	0	0	0	0	0			

Representing integers with Arrays

					<div>0</div>	carry	
					+		
op1	0	0	6	5	5		3
					+		
op2	0	0	0	1	0		2
					=		
sum	0	0	0	0	5		

Significant Digits

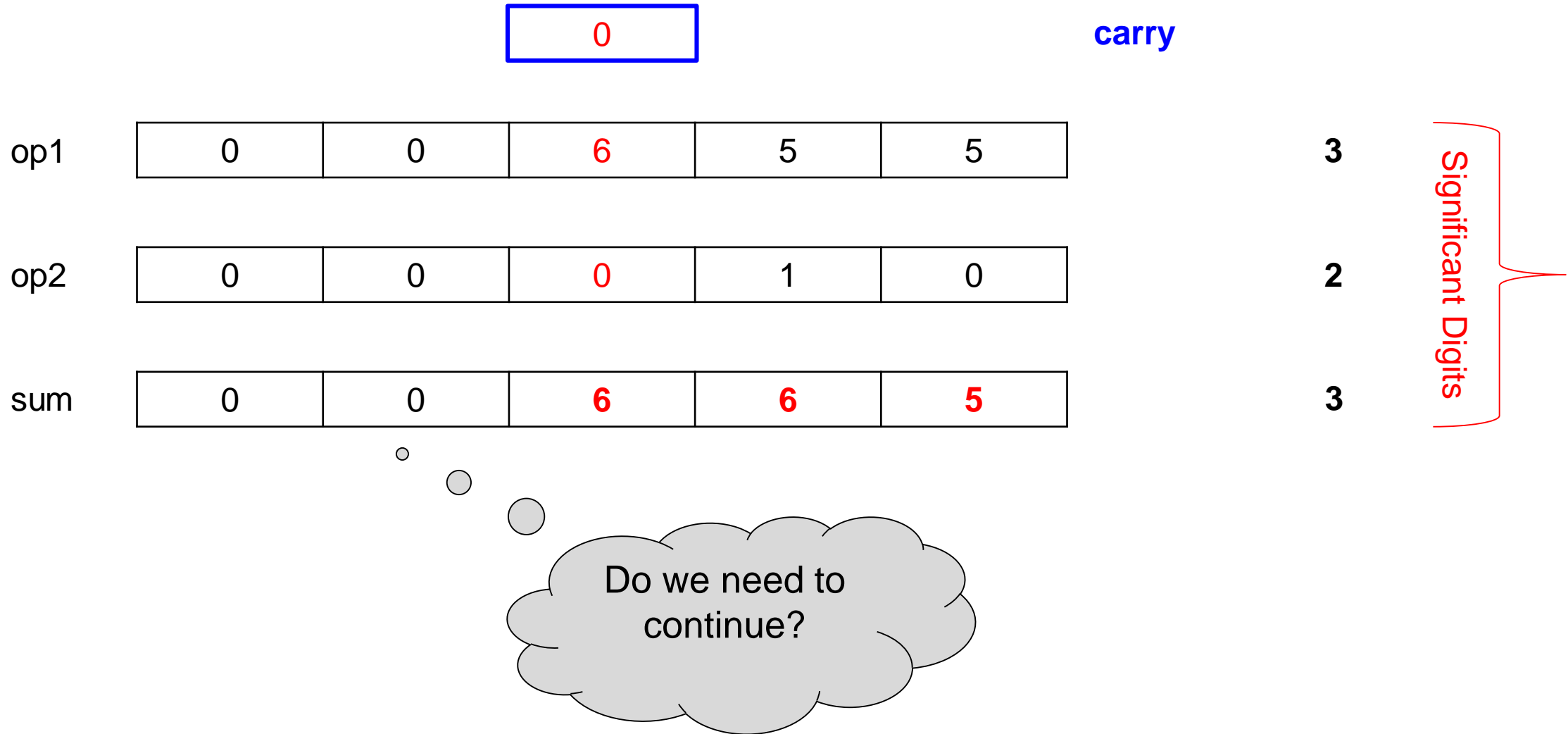
Representing integers with Arrays



Representing integers with Arrays

			<div>0</div>			carry	
			+				
op1	0	0	6	5	5	3	Significant Digits
			+				
op2	0	0	0	1	0	2	
			=				
sum	0	0	6	6	5		

Representing integers with Arrays



Representing integers with Arrays

			0			carry	
op1	0	0	6	5	5	3	Significant Digits
op2	0	0	0	1	0	2	
sum	0	0	6	6	5	3	

No, because we have added all the digits up to the largest significant digits of the two numbers.

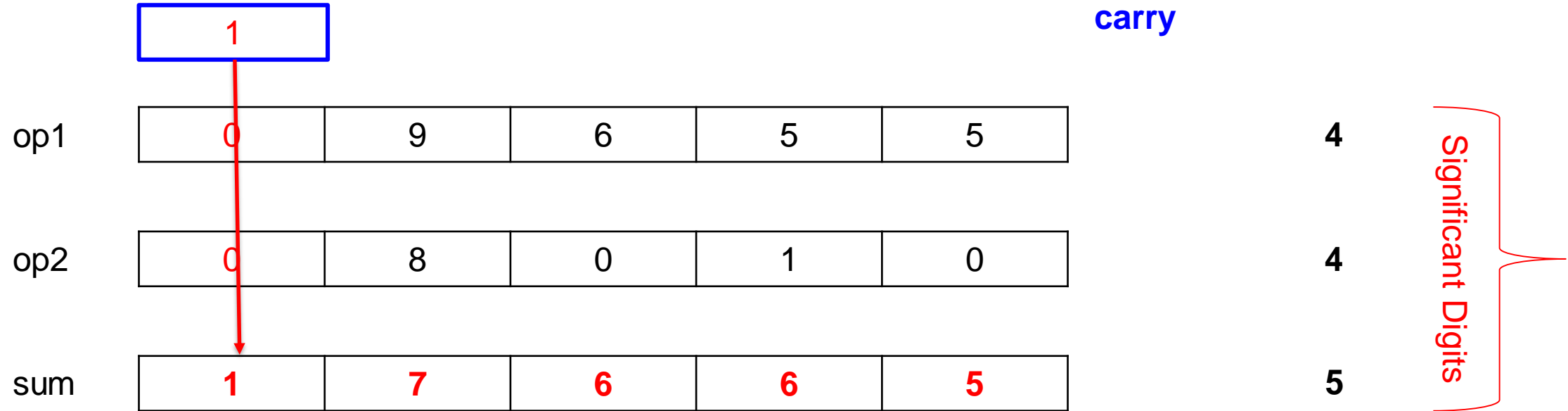
Representing integers with Arrays: *what if?*

		<div>0</div>				carry	
		+					
op1	0	9	6	5	5	4	Significant Digits
		+					
op2	0	8	0	1	0	4	
		=					
sum	0	?	6	6	5	3	

Representing integers with Arrays: what if?

	<div>1</div>					carry		
op1	0	9	6	5	5		4	Significant Digits
op2	0	8	0	1	0		4	
sum	0	7	6	6	5		4	

Representing integers with Arrays: what if?



Representing integers with Arrays: *another* what if?

	<div>1</div>					carry	
	+						
op1	8	9	6	5	5	5	Significant Digits
	+						
op2	0	8	0	1	0	4	
	=						
sum	?	7	6	6	5	5	

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

0

carry

op1

8	9	6	5	5
---	---	---	---	---

5

op2

0	8	0	1	0
---	---	---	---	---

4

sum

9	7	6	6	5
---	---	---	---	---

5

Significant Digits

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

	<div>1</div>					carry	
	+						
op1	9	9	6	5	5	5	Significant Digits
	+						
op2	0	8	0	1	0	4	
	=						
sum	?	7	6	6	5	5	

Representing integers with Arrays: *another* what if?

Overflow!

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

op1	0	0	6	5	5
op2	0	0	0	1	0
sum	0	0	0	0	0

3


2

Significant Digits

Array of digits

Procedural solution with just Arrays

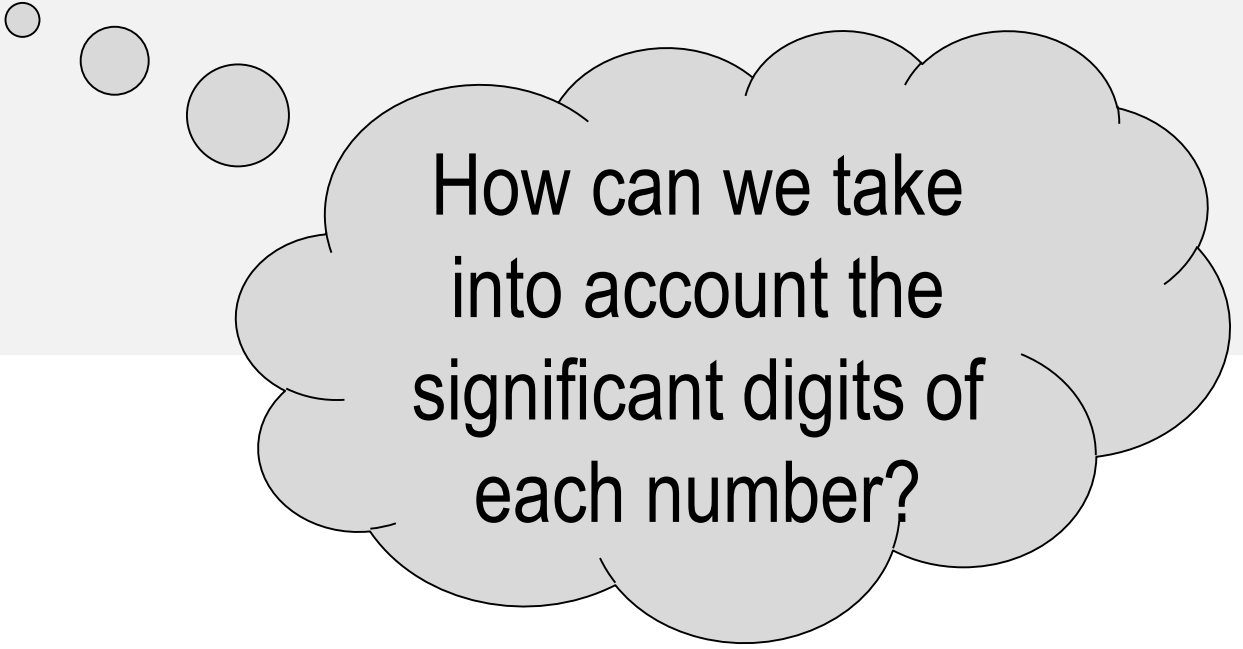
```
{  
    int[] op1 = {0,0,3,4,5};  
    int[] op2 = {0,0,0,9,8};  
    int [] sum = add(op1, op2);  
}
```



add has to process
through the length of
the arrays, even though
it is not necessary.

Procedural solution with just Arrays

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




How can we take
into account the
significant digits of
each number?

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

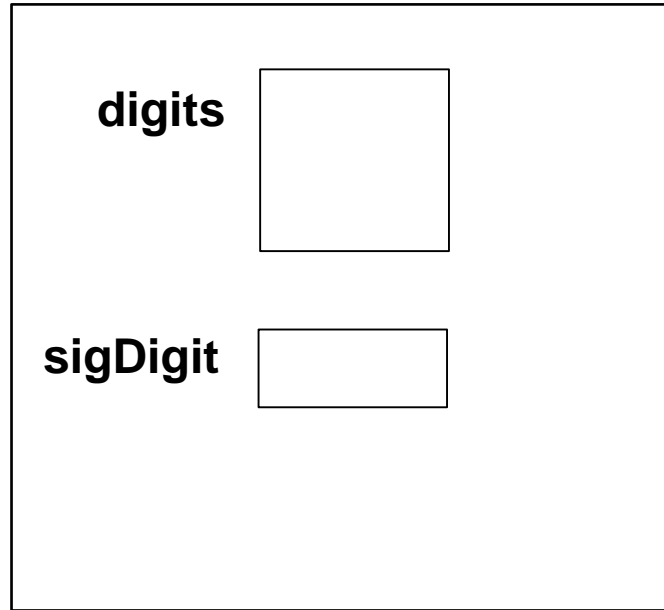
```
{  
    int[] op1 = {0,0,3,4,5}, op2 = {0,0,0,9,8};  
    int s_p1 = 3, s_op2 = 2;  
    int[] sum = add(op1, s_p1, op2, s_op2);  
}
```



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

Object Oriented Design:

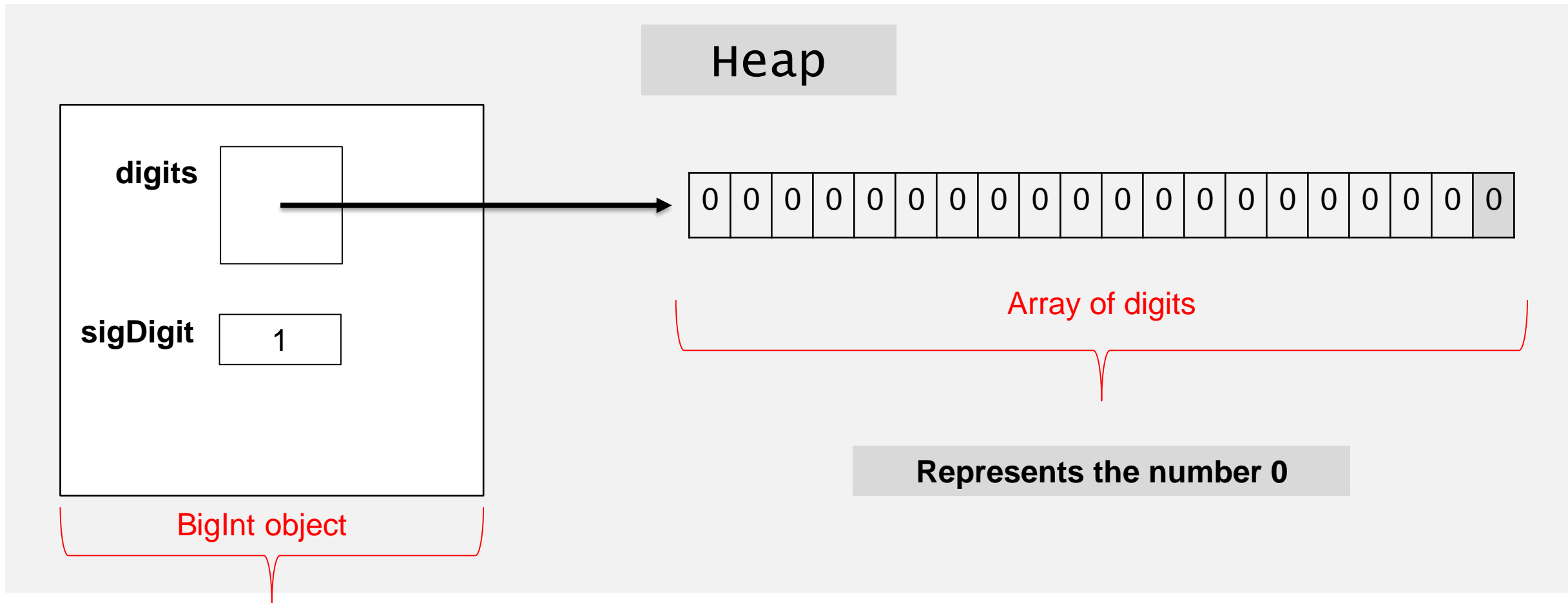
Encapsulating the data into a **BigInt** object



Encapsulates the data within
the object!

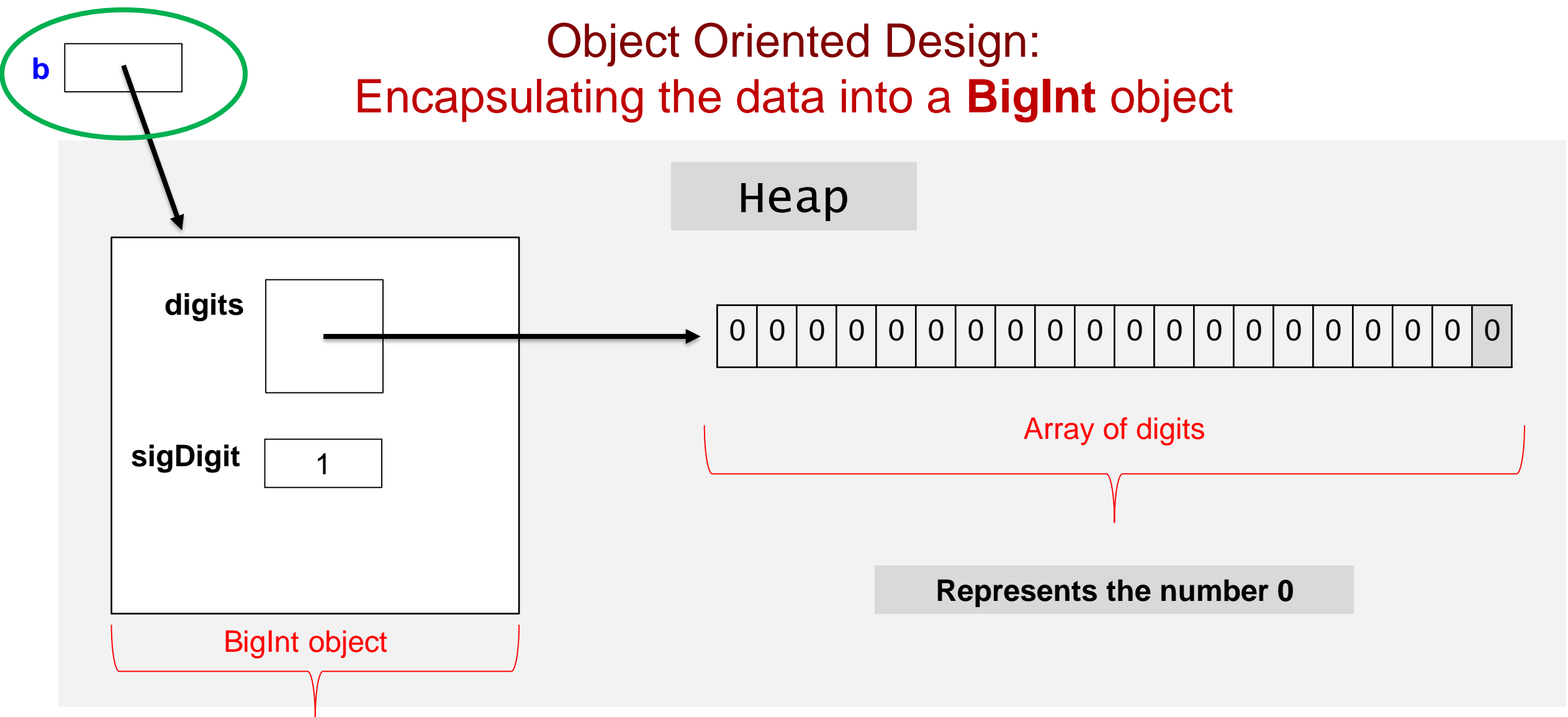
And provide all the *methods* necessary
to perform the required operations!

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

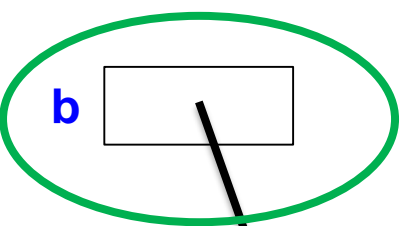


```
BigInt b = new BigInt(); // no-arg constructor
```

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

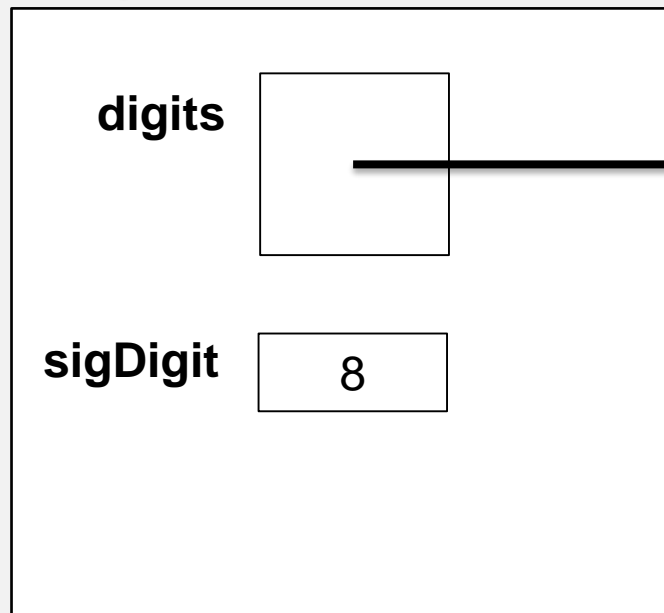


```
BigInt b = new BigInt(); // b references the object
```



another BigInt Object

Heap



0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	5	0	9	8	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Array of digits

Represents the number
10,509,800

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...

