Announcements/Follow-ups

- Final Exam 2 weeks from today
- Follow-ups
  - Questions (midterm 2, lab 8, quiz 4)
  - For each loop mistake
    - Updated slides and examples will be posted
  - Finish generics

For-each loop mistake

This doesn't modify the elements!

```
ArrayList<Double> myDoubleCollection;
…
for(Double x : myDoubleCollection) {
      x += 100.0;
}
```

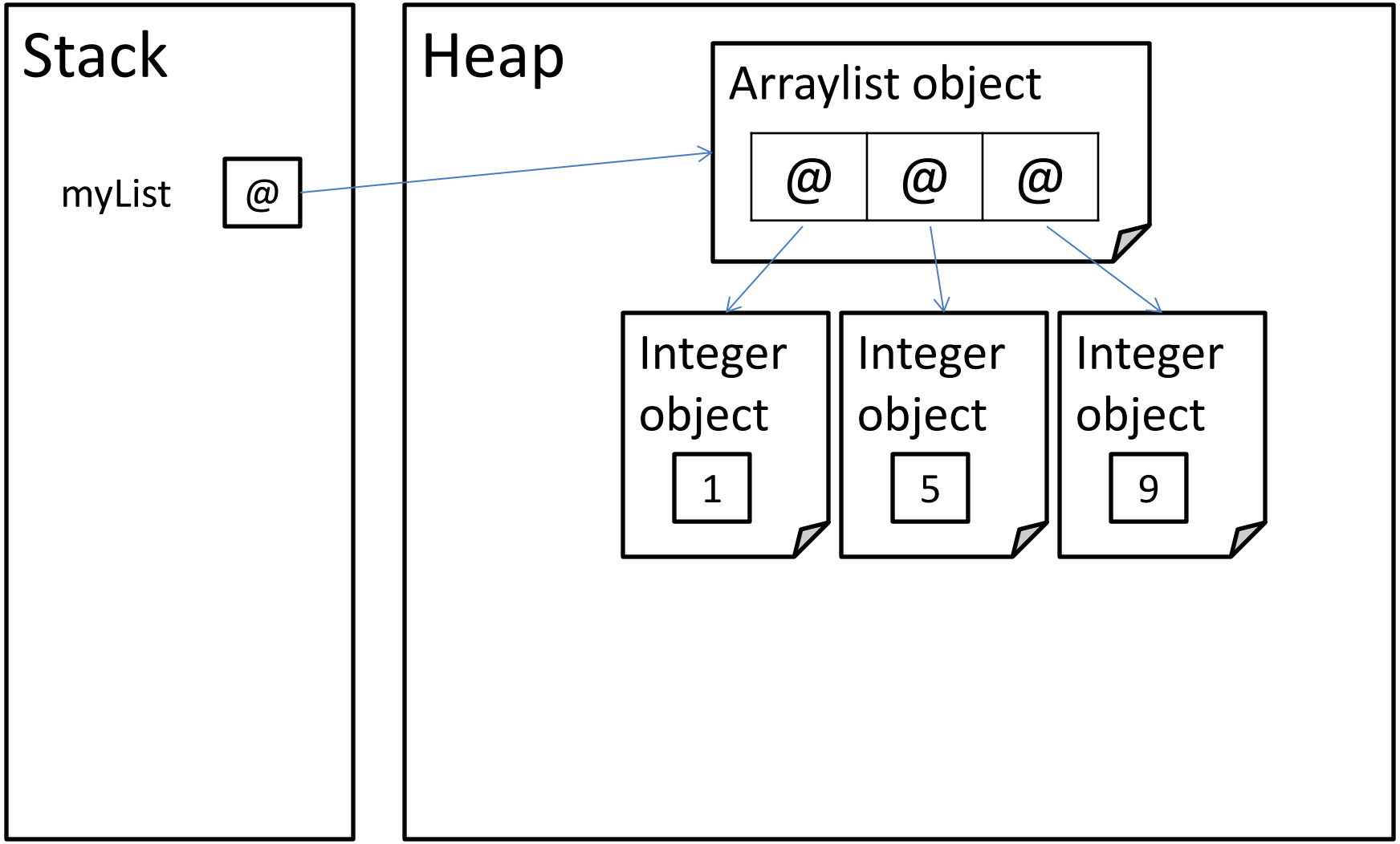
- ForEachAndIterators example revisited

For-each loop mistake

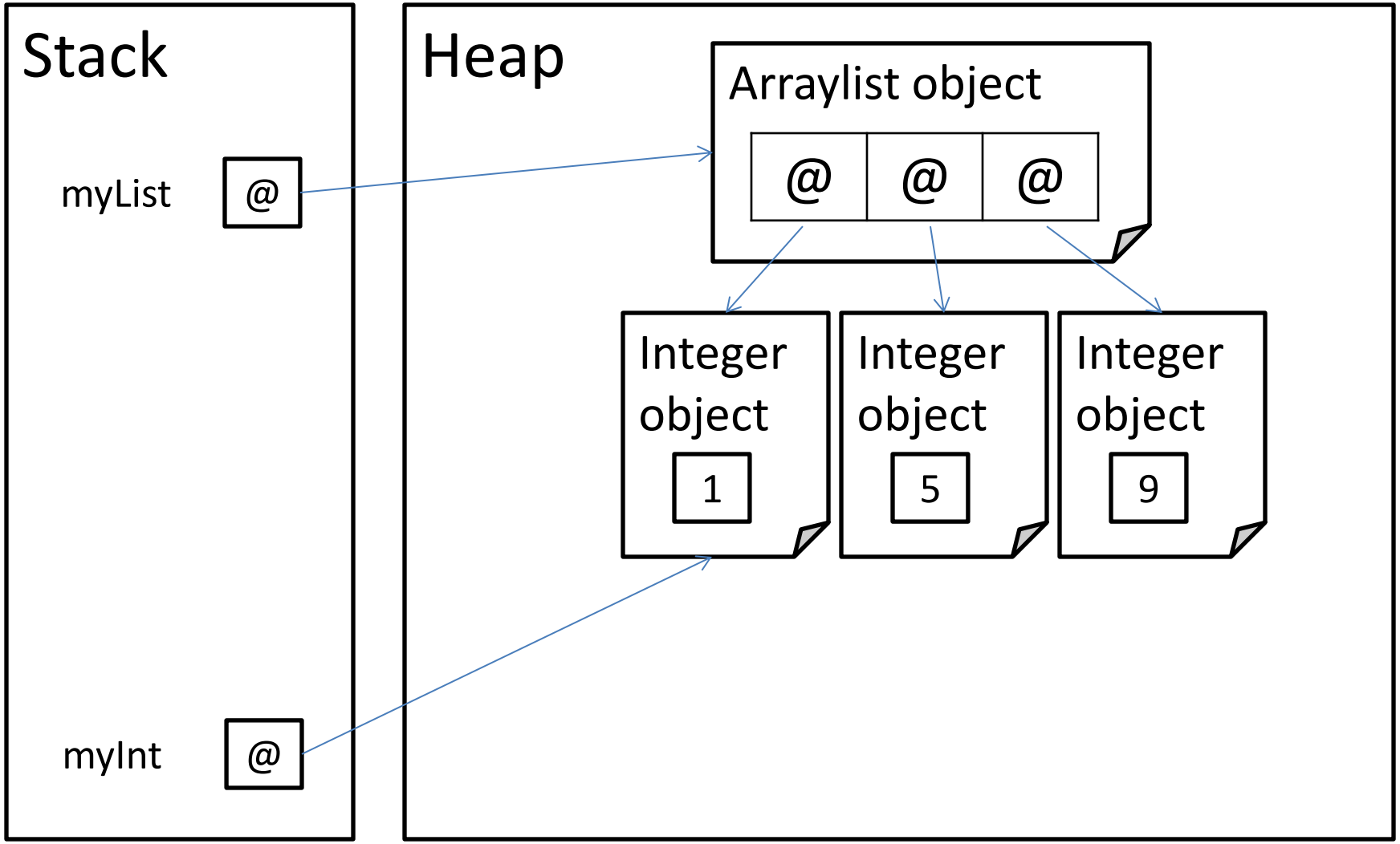
This would modify the elements:

```
ArrayList<MutableDouble> myMutables;
…
for(MutableDouble x : myMutables) {
     x.increaseValueBy(100.0);
}
```

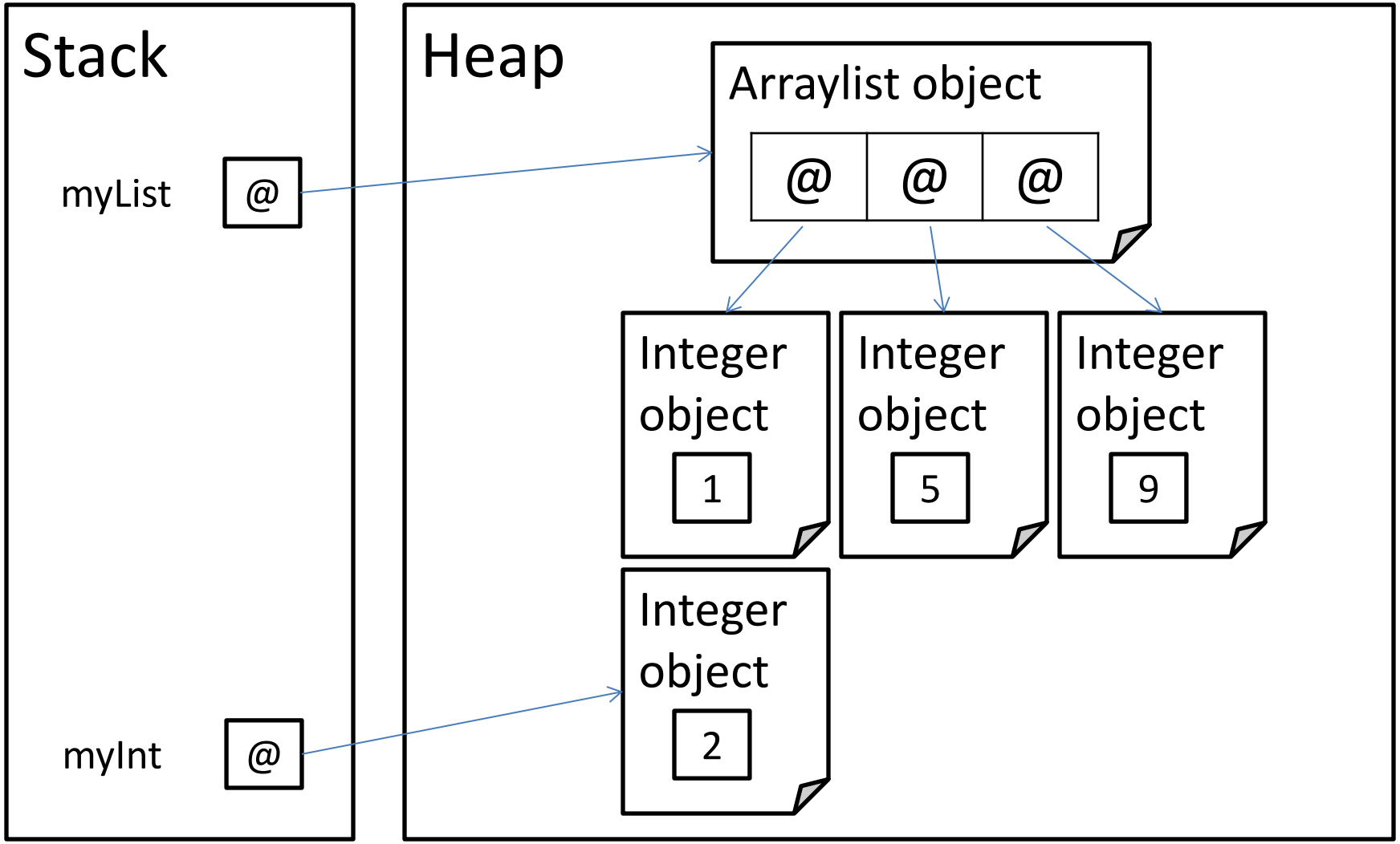
ArrayList<Integer> myList;



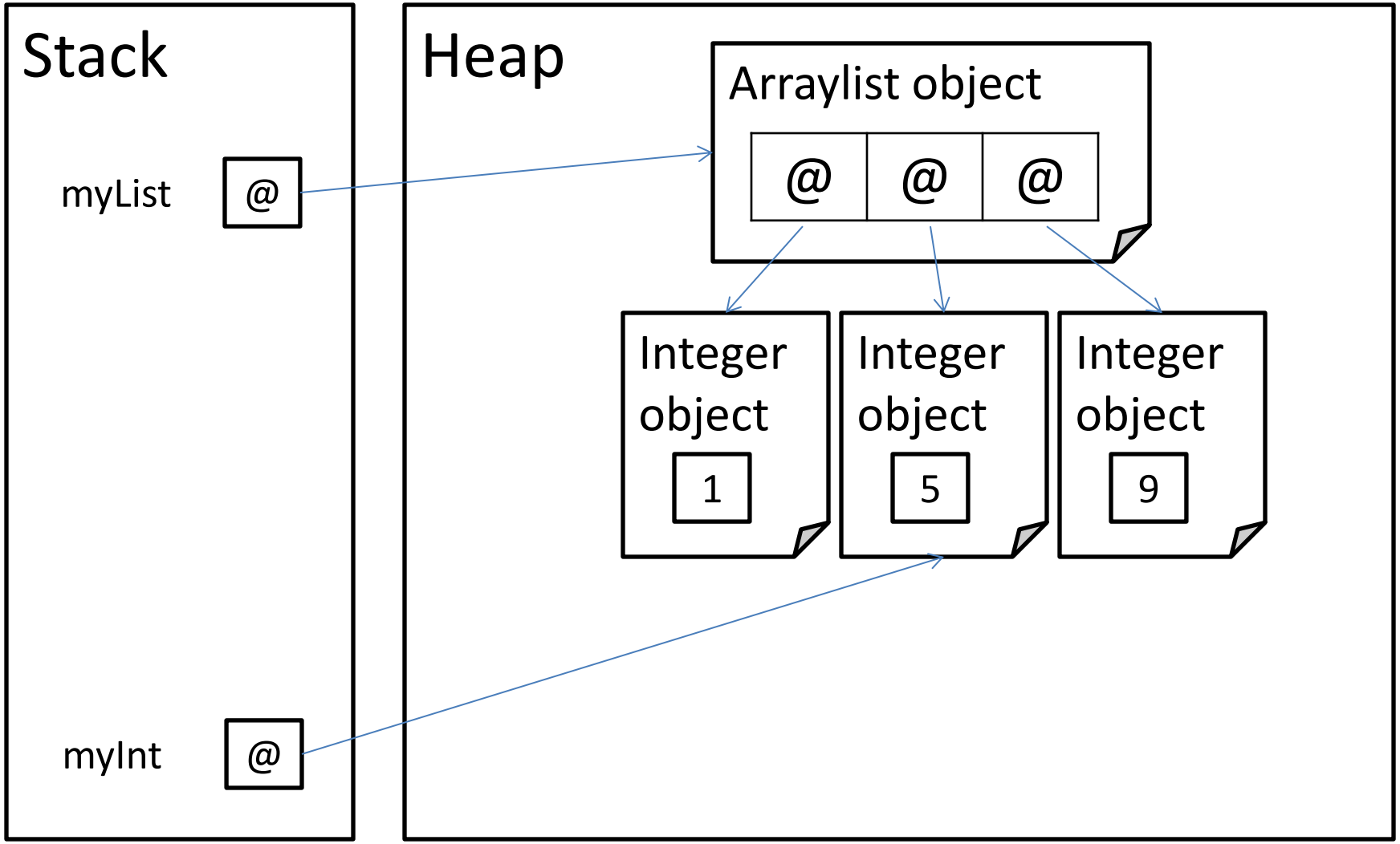
for(Integer myInt : myList)



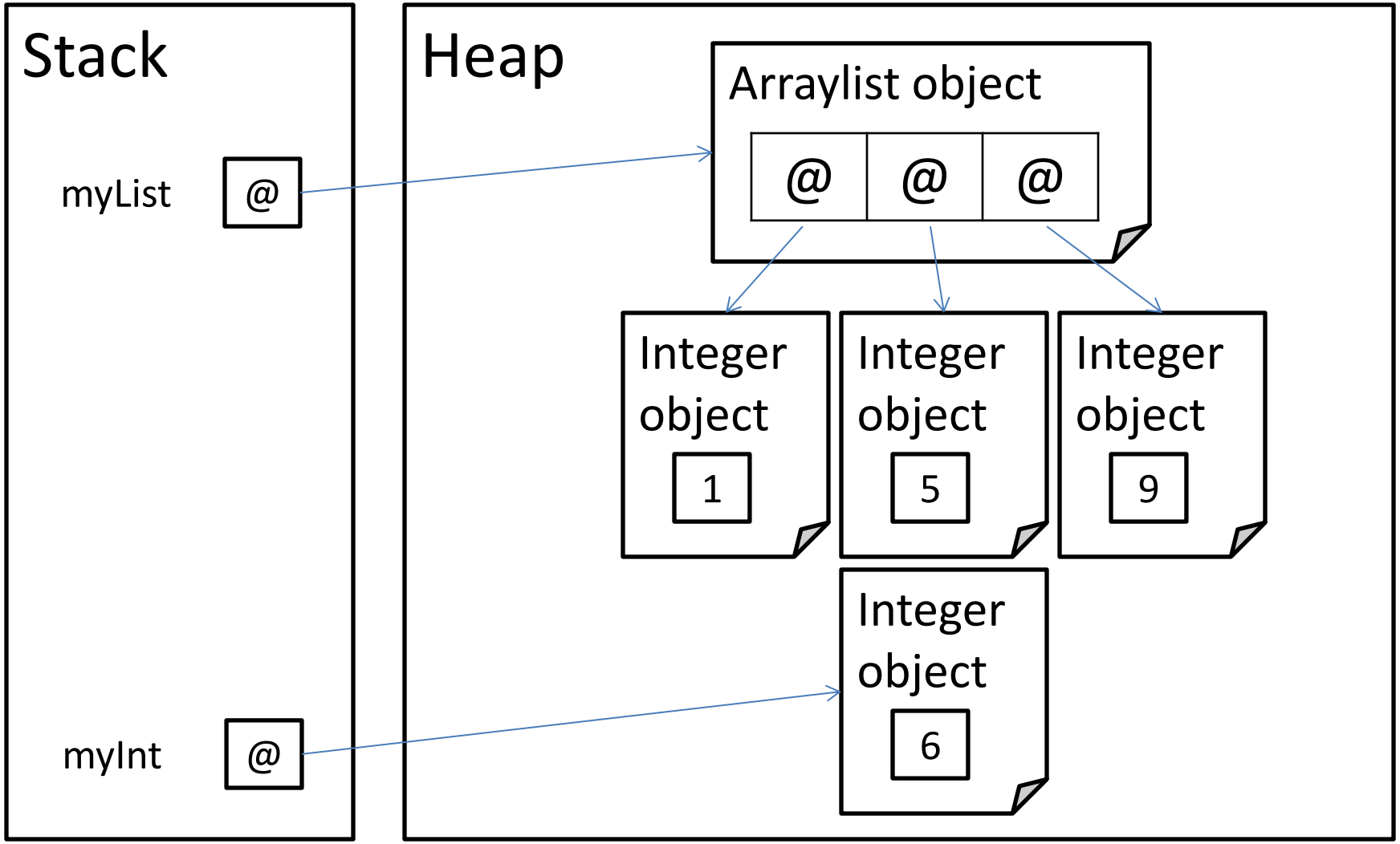
```
myInt++; //myInt = myInt+1;
```



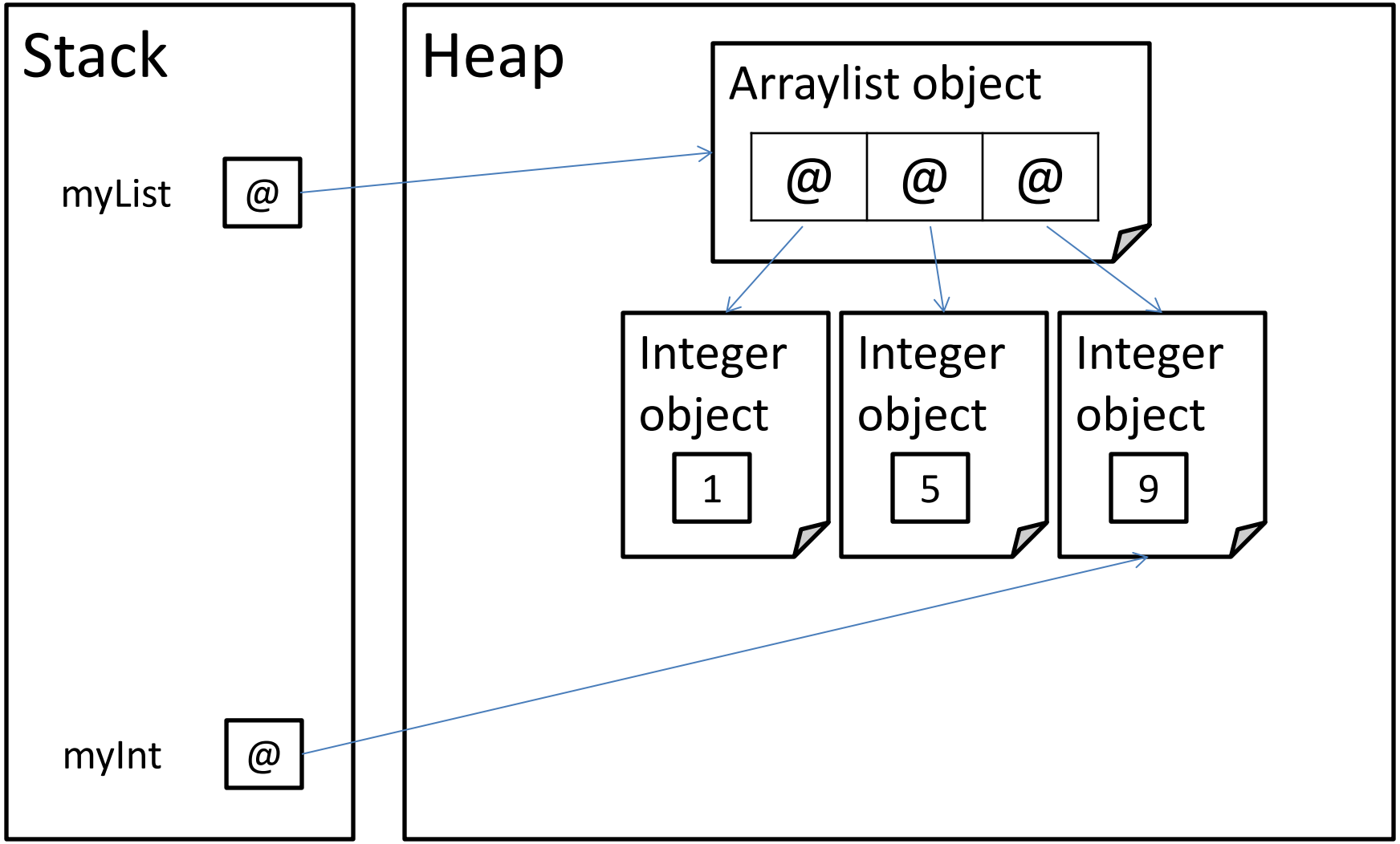
for(Integer myInt : myList)



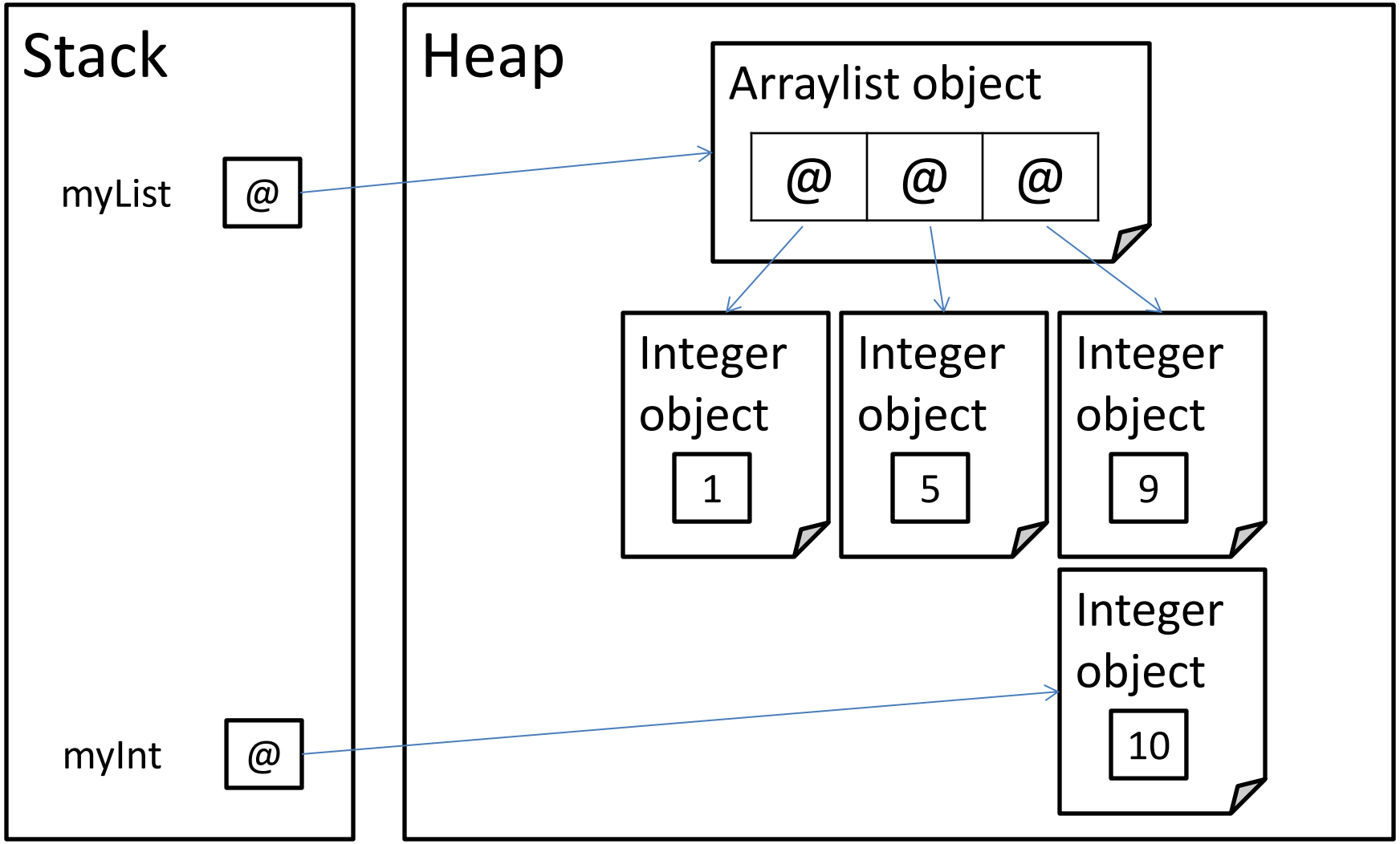
```
myInt++; //myInt = myInt+1;
```



for(Integer myInt : myList)



```
myInt++; //myInt = myInt+1;
```



Generics

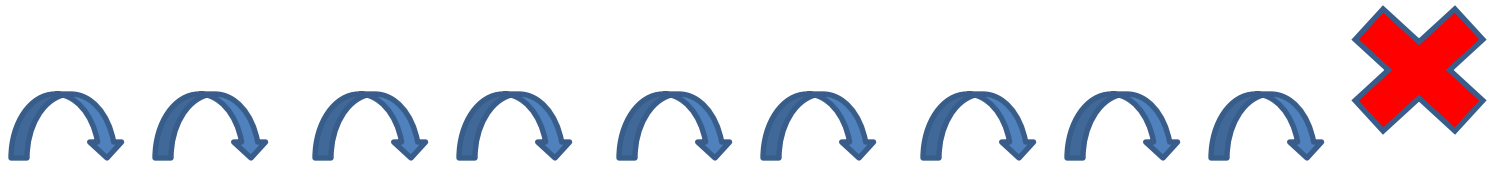
- Wednesday Slides
  - Implementing generic methods
  - Implementing generic interfaces
  - Collections algorithms
- Generics don’t play well with arrays
  - GenericSwap revisited
  - When using generics, use collections
- LexArray example

Data Structures

- Any way of organizing data in memory
  - Objects
  - Arrays
  - Linked lists
  - Collections
  - Databases
- Typically optimized for a specific purpose
  - Inserting new data
  - Finding existing data (“queries”)
  - Etc.

Linear search

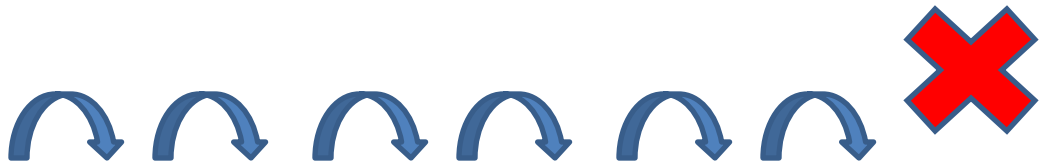
10?



0	1	2	3	4	5	6	7	8	9
16	3	22	7	9	8	11	1	18	4

Linear search on sorted array

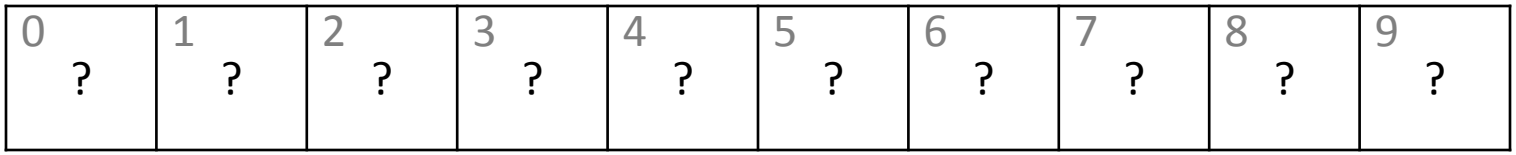
10?



0	1	2	3	4	5	6	7	8	9
1	3	4	7	8	9	11	16	18	22

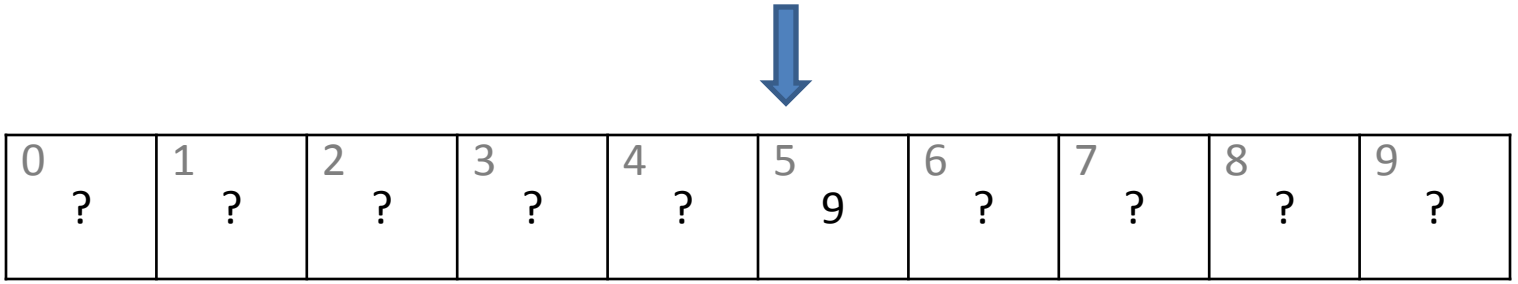
Binary search on sorted array

10?



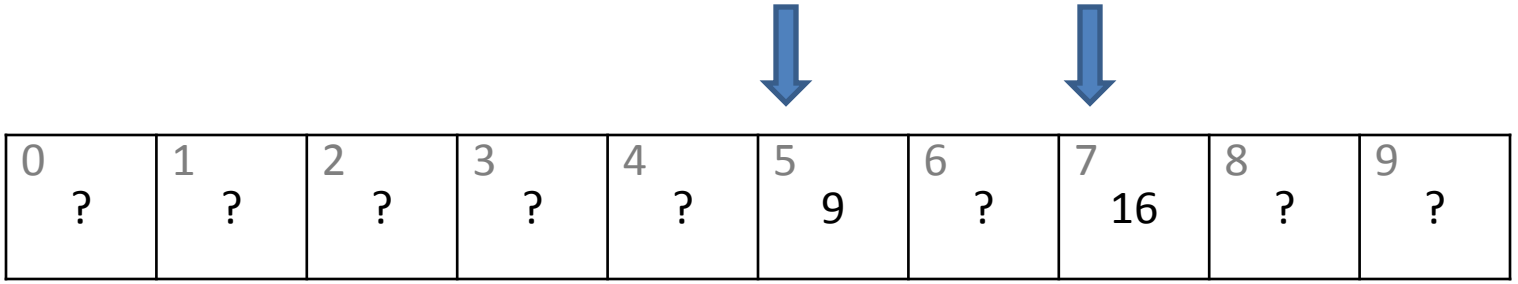
Binary search on sorted array

10?



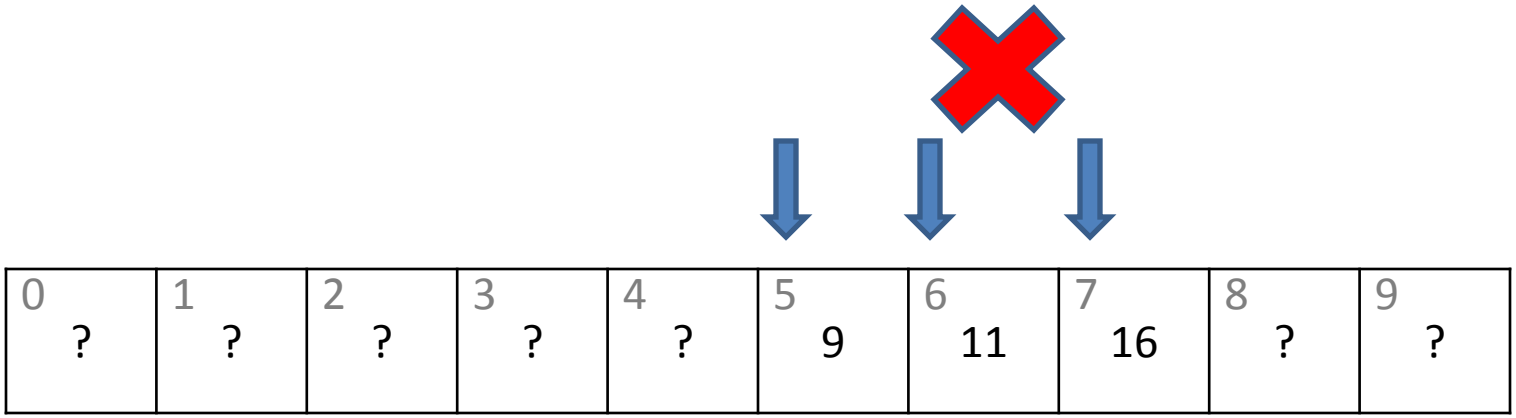
Binary search on sorted array

10?



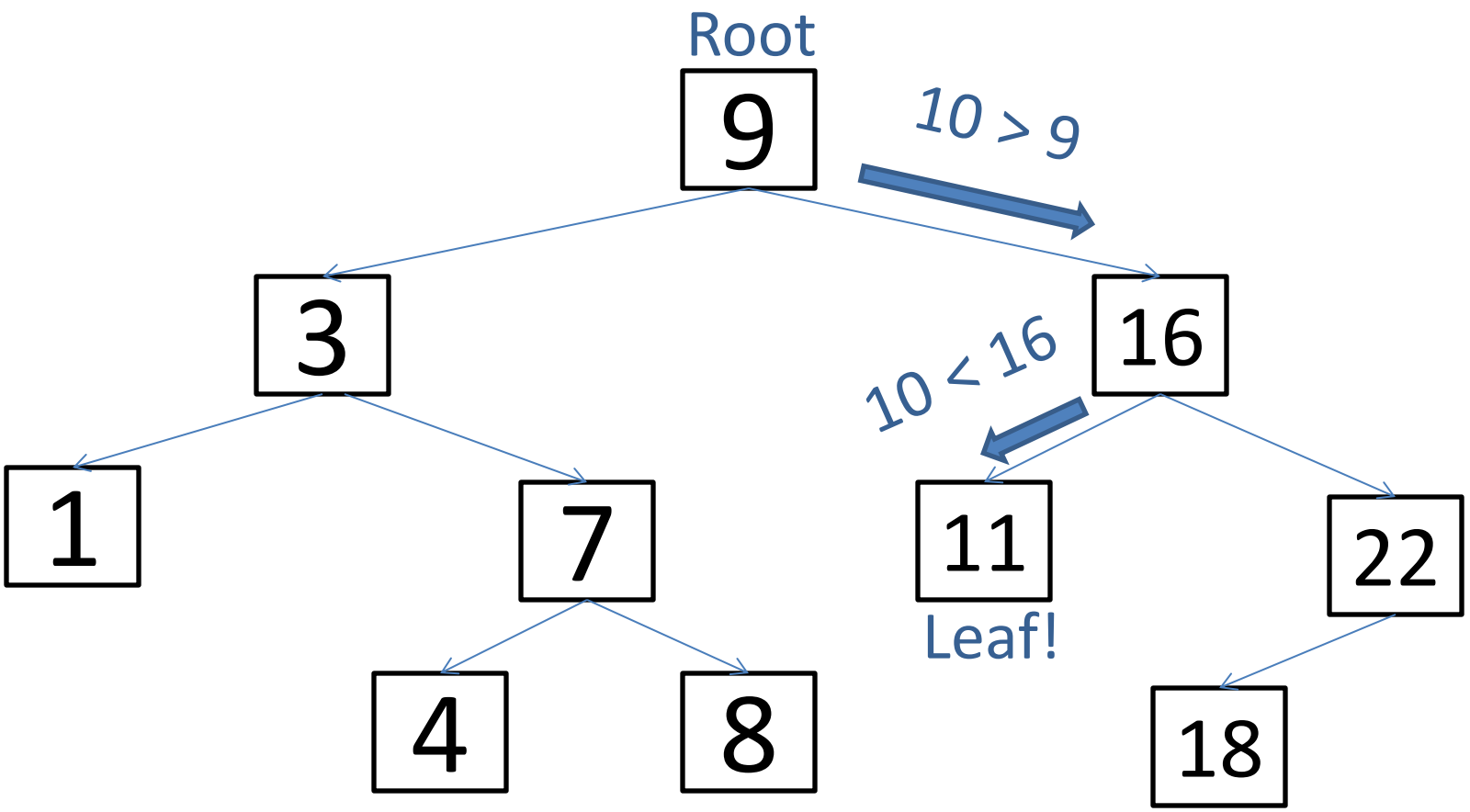
Binary search on sorted array

10?



- BinarySearch example

Binary search tree



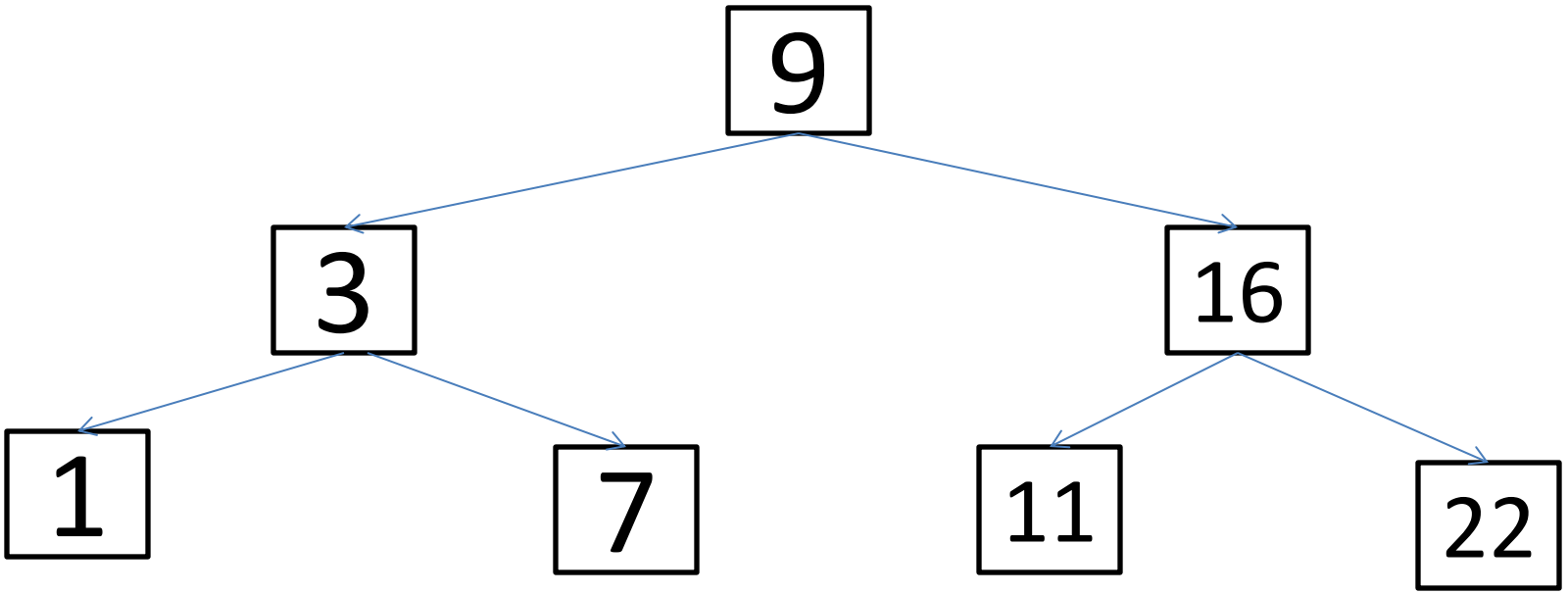
Binary tree

- Each entry is called a “node” or “vertex”
- Arrows connecting nodes are called “edges”
- Arrows point from “parent” nodes to “child” nodes
- Each node has two “children”, left and right.
- Children of children are called “descendents”
- All descendents of a node are called a “sub-tree”
- The top node is called the “root”
- Nodes without children are called “leaves”

Binary **search** tree

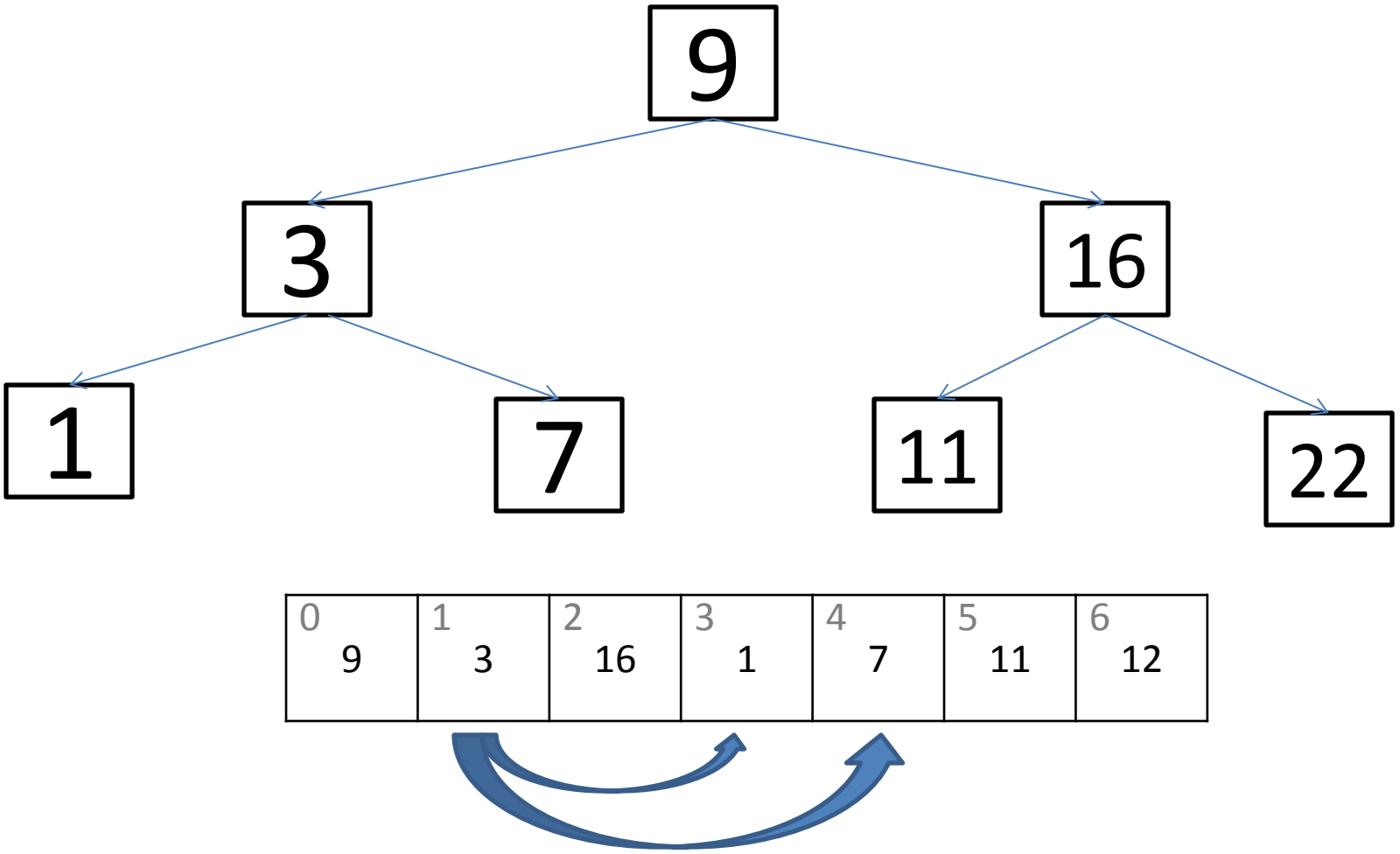
- Every node in the left sub-tree has a value less than the root
- Every node in the right sub-tree has a value greater than the root
- To “search” the tree:
  - Start at the root
  - Compare the query with the root value
  - Move left or right depending on the comparison
  - Repeat on the resulting sub-tree
  - Stop when the value is found, or a leaf is reached

Binary search tree – with array

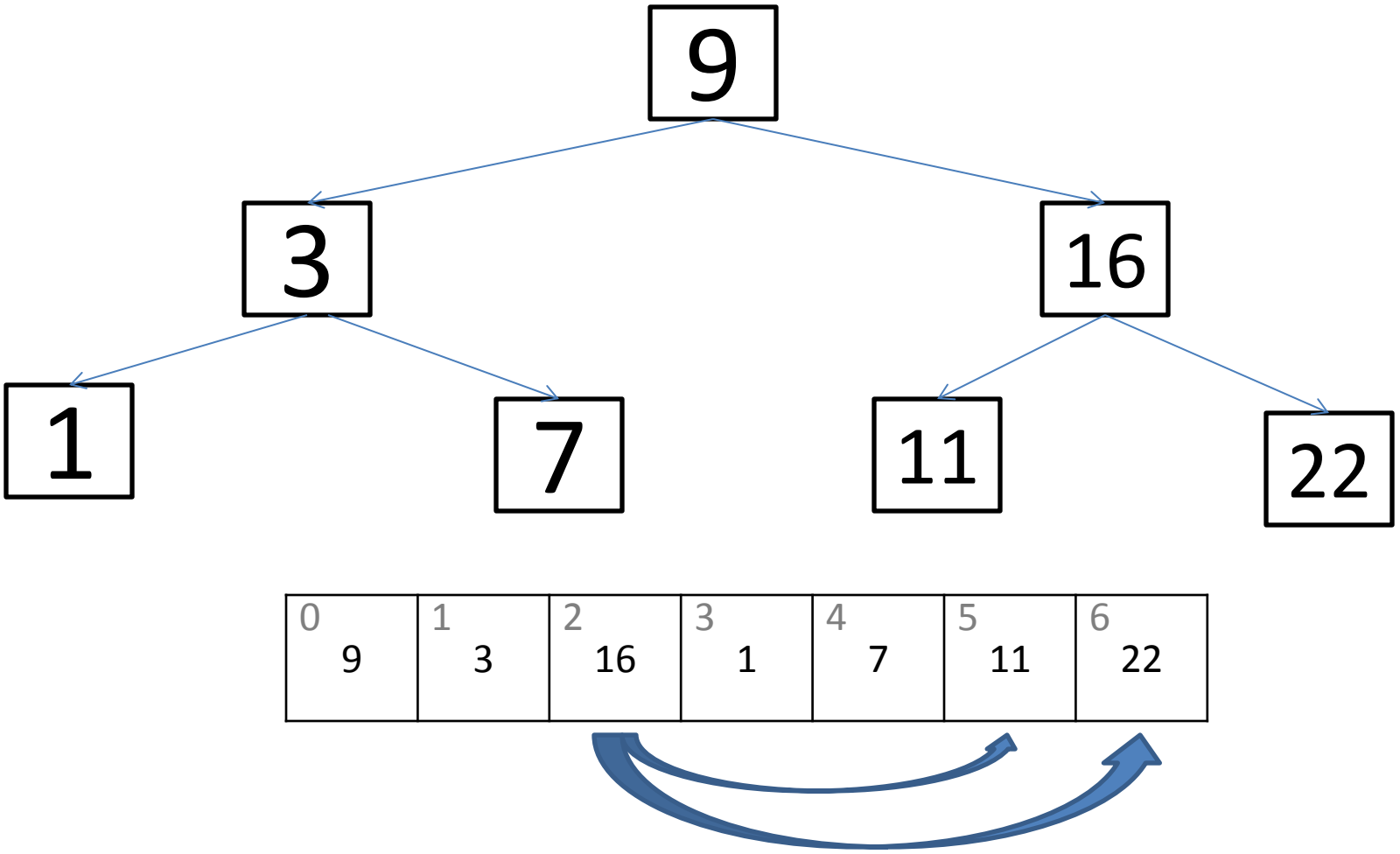


0	1	2	3	4	5	6
9	3	16	1	7	11	12

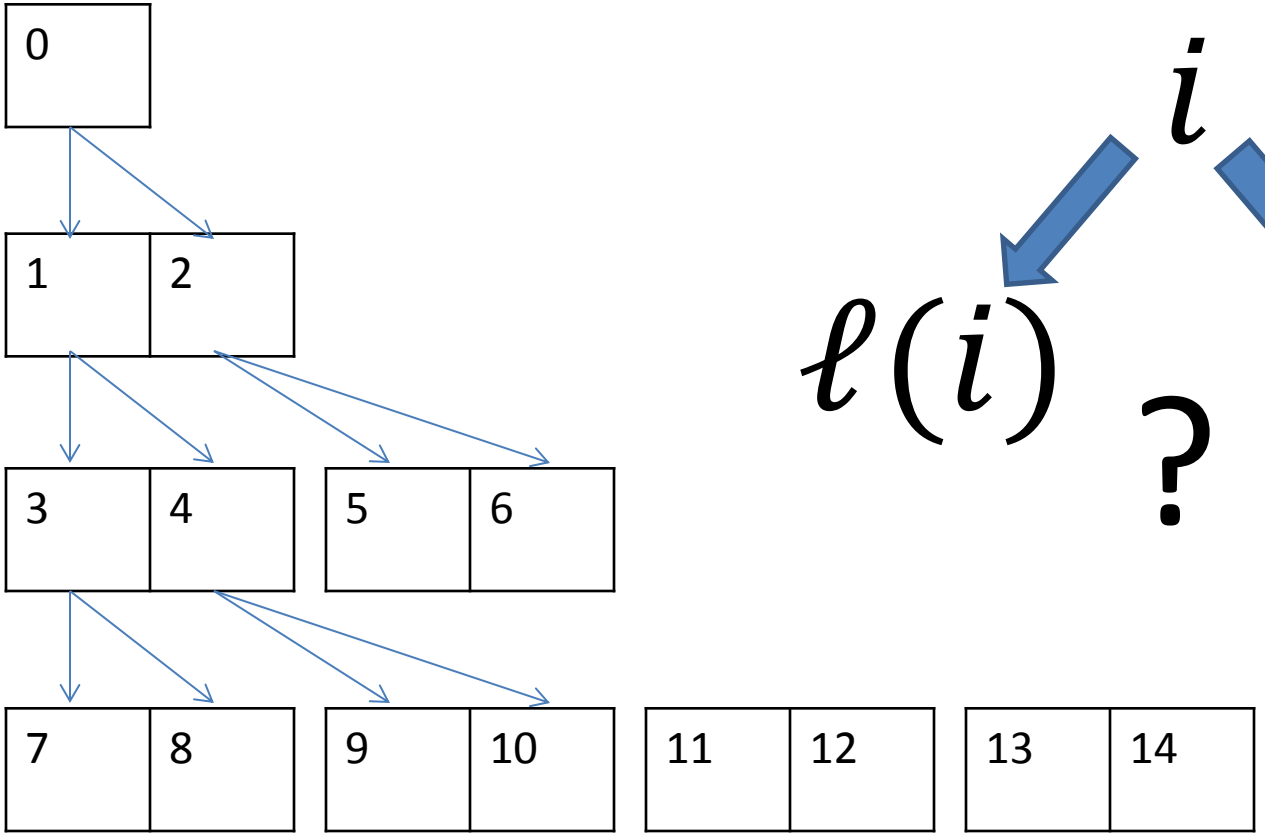
Binary search tree – with array



Binary search tree – with array



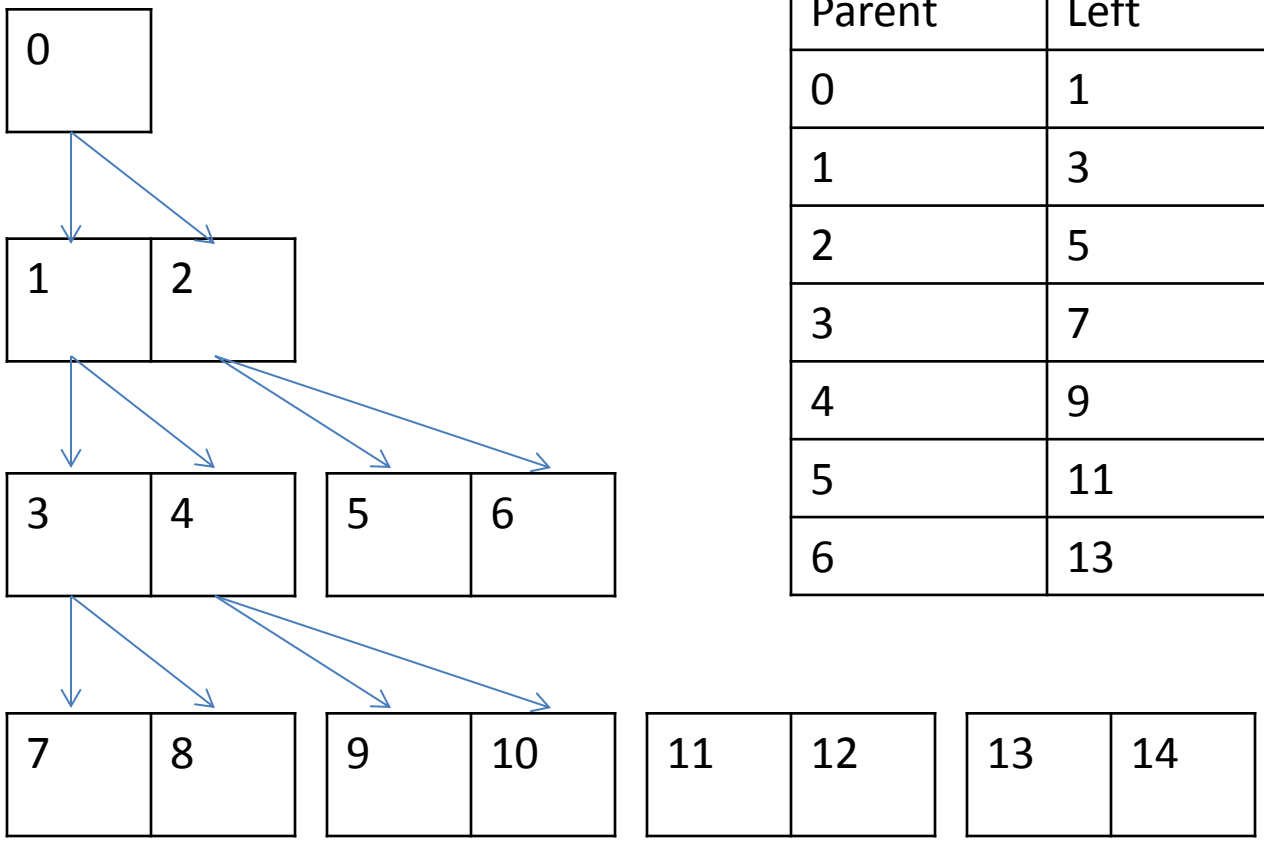
Binary search tree – with array



$$i$$

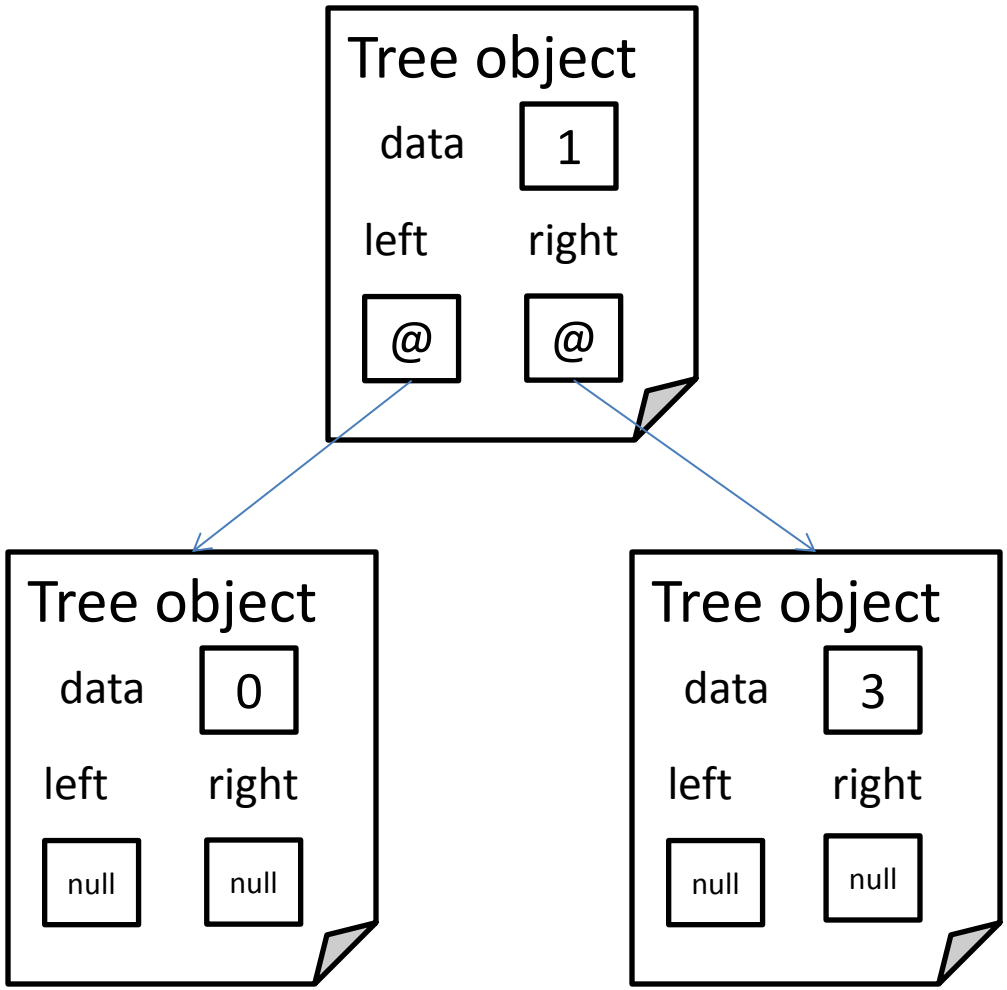
$$\ell(i) \quad ? \quad r(i)$$

Binary search tree – with array



Parent	Left	Right
0	1	2
1	3	4
2	5	6
3	7	8
4	9	10
5	11	12
6	13	14

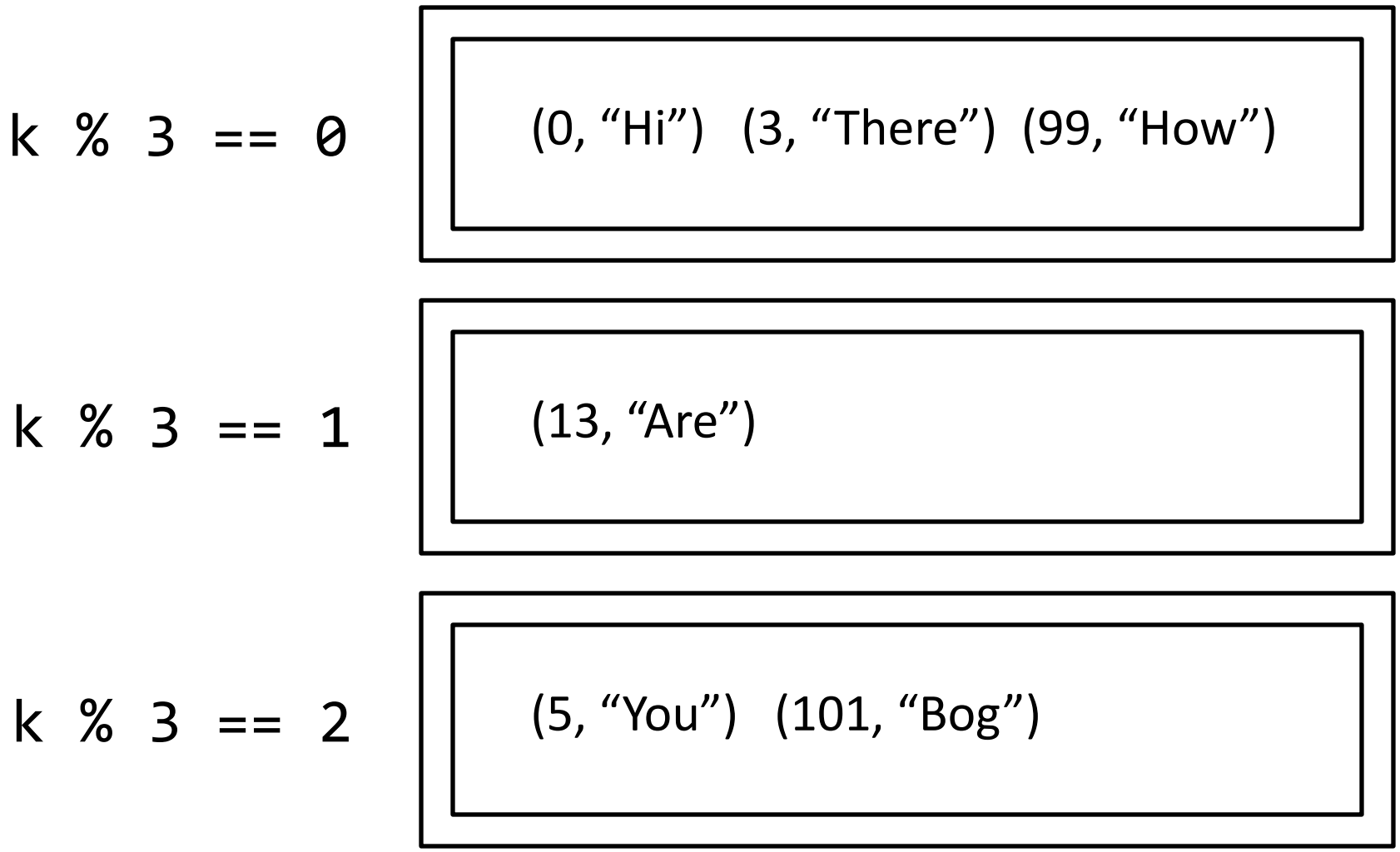
Binary search tree - references



Hash Tables

- A mapping from (potentially non-integer) keys to values.
  - E.g. White pages
- Supports fast value look-up for a given key
- Keys are mapped to a set of buckets
  - Typically they should be distributed evenly
- Look-up by jumping directly to the correct bucket
- MyHash example

Hash Tables



Hash Tables