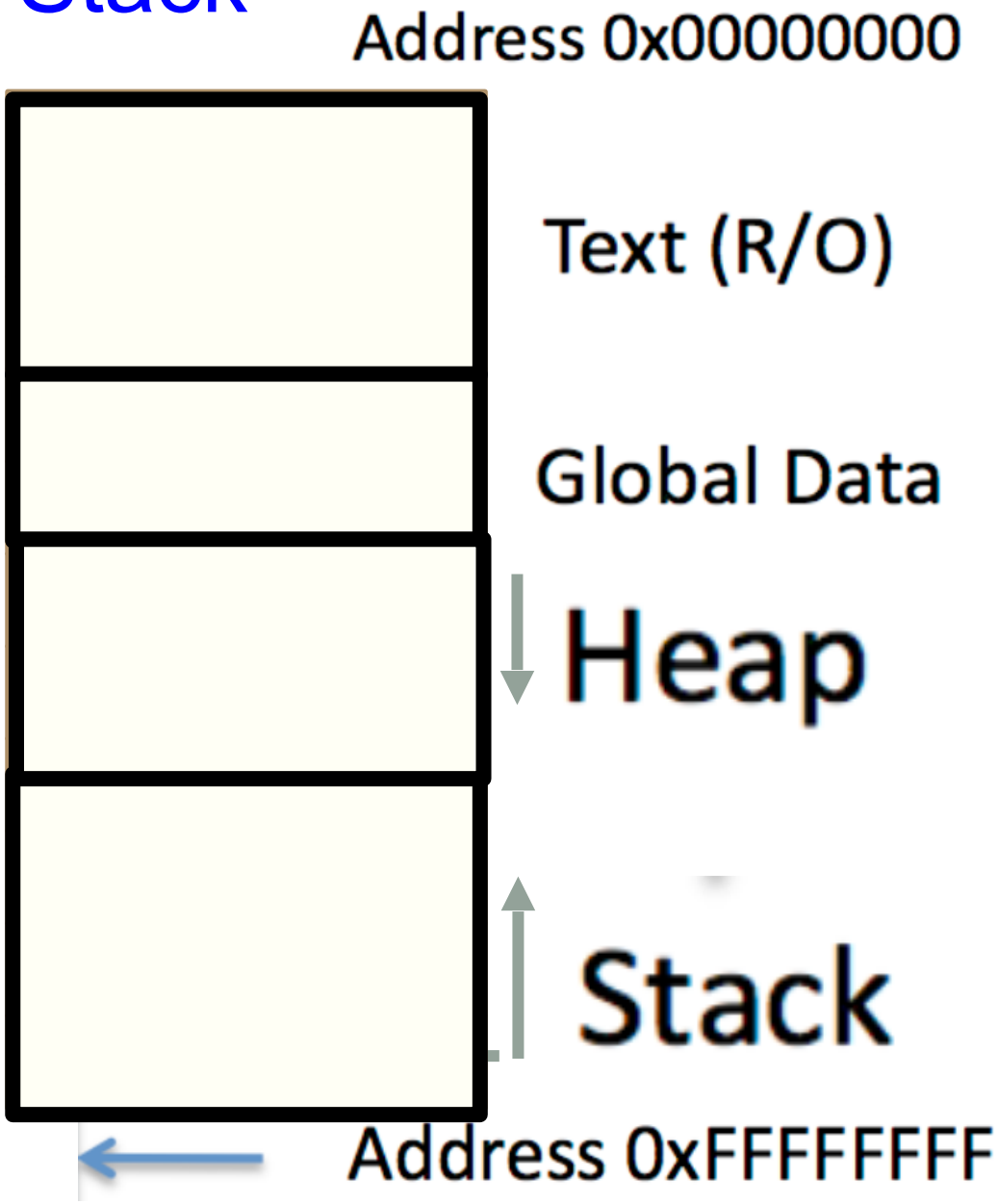
HEAP VS STACK
DYNAMIC MEMORY
LINKED LISTS

Problem Solving with Computers-II

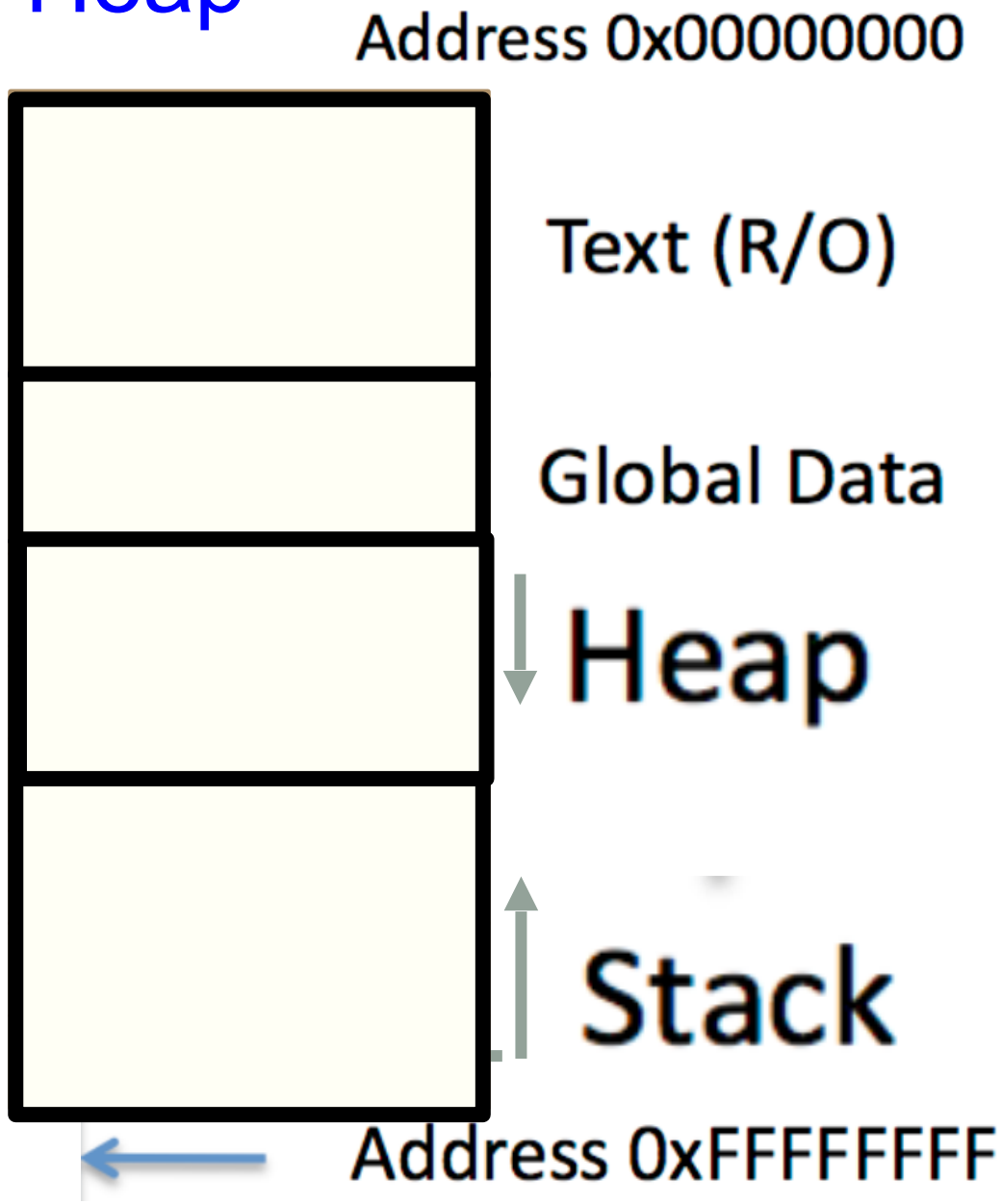
C++ Memory Model: Stack

- Stack: Segment of memory managed automatically using a Last in First Out (LIFO) principle
- Think of it like a stack of books!



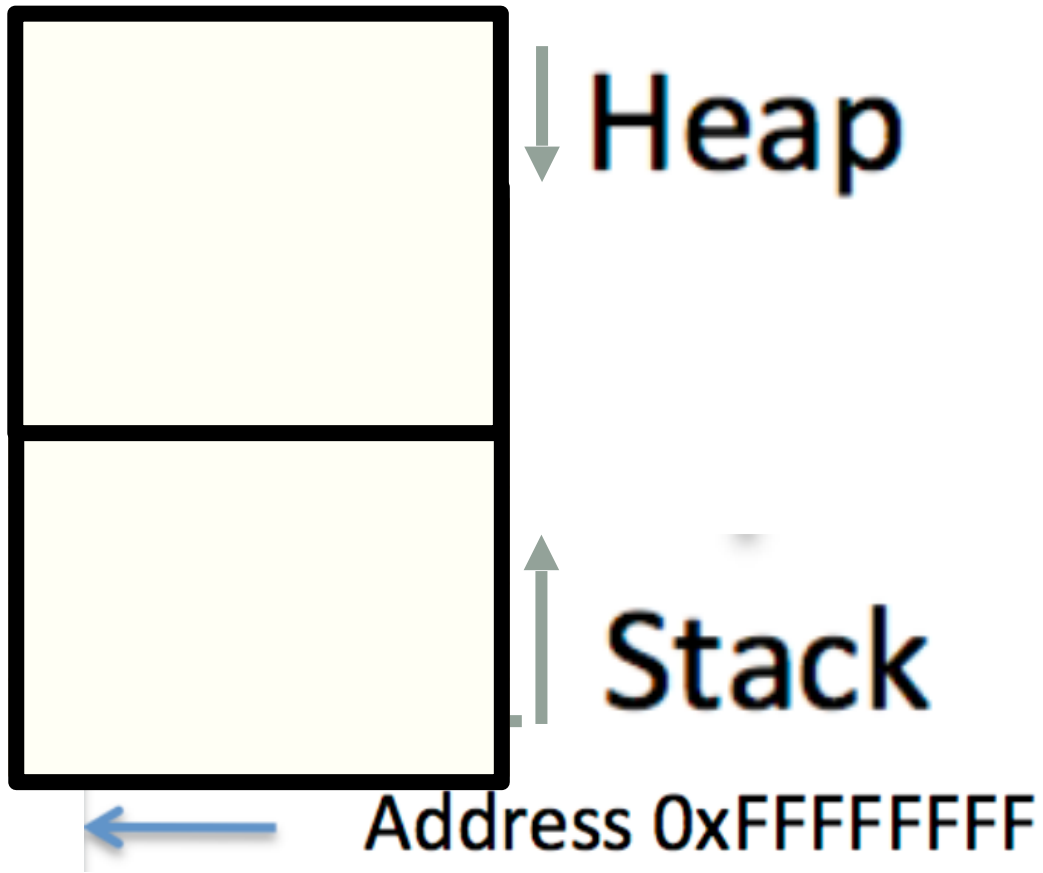
C++ Memory Model: Heap

- Heap: Segment of memory managed by the programmer
- Data created on the heap stays there
  - FOREVER or
  - until the programmer explicitly deletes it



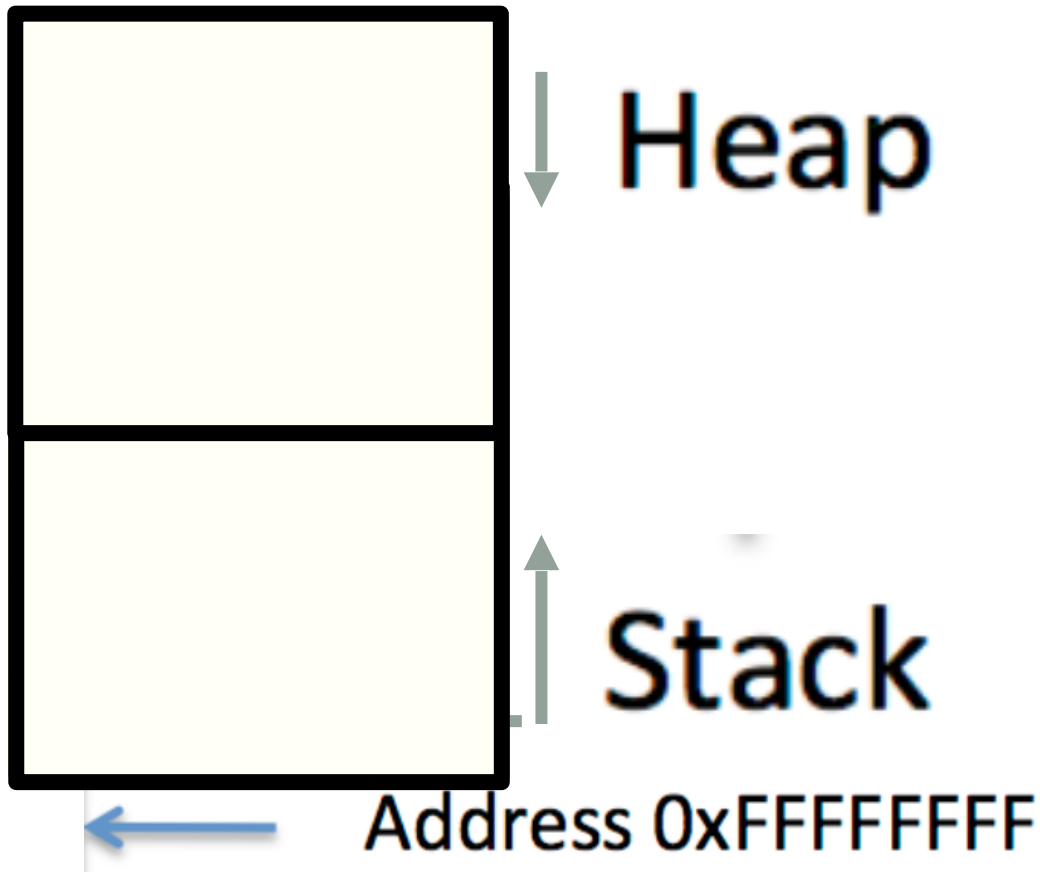
Creating data on the Heap: new

To **allocate** memory on the heap use the **new** operator



Deleting data on the Heap: delete

To **free** memory on the heap use the **delete** operator



Dynamic memory management = Managing data on the heap

```
int* p= new int; //creates a new integer on the
heap

SuperHero* n = new SuperHero;

                  //creates a new Student on the
heap

delete p; //Frees the integer

delete n; //Frees the Student
```

The case of the disappearing data!

```
int getInt(){
      int x=5;
      return x;
}
int* getAddressOfInt(){
      int x=10;
      return &x;
}
int main(){
      int y=0, *p=nullptr, z=0;
      y = getInt();
      p = getAddressOfInt();
      z = *p;
     cout<<y<<", "<<z<<", "<<*p<<endl;
}
```

What is the output?

A. 5, 0, 10

B. 5, 10, 10

C. Something else

Heap vs. stack

```
1 #include <iostream>
2 using namespace std;
3
4 int* createAnIntArray(int len){
5
6     int arr[len];
7     return arr;
8
9 }
```

Does the above function correctly return an array of integers?

A. Yes

B. No

Memory Errors

- Memory Leak: Program does not free memory allocated on the heap.

- Segmentation Fault: Code tries to access an invalid memory location

Linked Lists

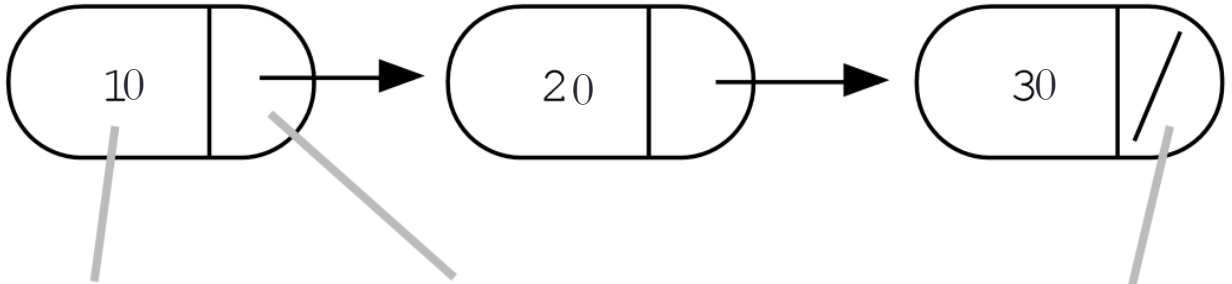
Array List

Stack

Heap

The overall list is built by connecting the nodes together by their next pointers. The nodes are all allocated in the heap.

Linked List



Each node stores one data element (int in this example).

Each node stores one next pointer.

The next field of the last node is NULL.

What is the key difference between the two?

Questions you must ask about any data structure:

- **What operations does the data structure support?**

  A linked list supports the following operations:

  1. Insert (a value to the head)
  2. Append (a value to the tail)
  3. Delete (a value)
  4. Search (for a value)
  5. Min
  6. Max
  7. Print all values

- **How do you implement each operation?**
- **How fast is each operation?**

Linked-list as an Abstract Data Type (ADT)

```
class LinkedList {
public:
    LinkedList();
    ~LinkedList();
    // other public methods


private:
    struct Node {
        int info;
        Node* next;
    };
    Node* head;
    Node* tail;
};
```

Next time

- Rule of Three
- Linked Lists contd.