Watch this video! We'll come back to hash tables at the end of the class.

tinyurl.com/phyllis-hashtable

CS50 Section 5

Linked Lists

- We refer to this combination of structs and pointers, when used to create a “chain” of nodes a **linked list.**

- A linked list **node** is a special type of struct with two fields:
  - Data of some type
  - A pointer to another linked list node.

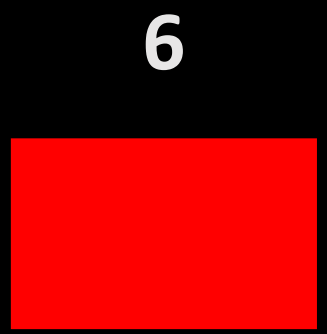
```
typedef struct node
{
        int value;
        struct node *next;
}
node;
```

- Create a linked list:
  - Dynamically allocate space for a new (your first!) node.
  - Check to make sure you didn’t run out of memory (does node == NULL?)
  - Initialize the value field.
  - Initialize the next field (specifically, to NULL).
  - Return a pointer to your newly created node.

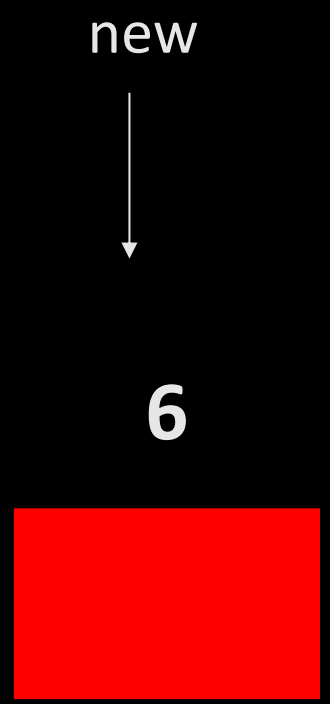
- Create a linked list:
  - Dynamically allocate space for a new (your first!) node.
  - Check to make sure you didn’t run out of memory.
  - Initialize the value field.
  - Initialize the next field (specifically, to NULL).
  - Return a pointer to your newly created node.

6

- Create a linked list:
  - Dynamically allocate space for a new (your first!) node.
  - Check to make sure you didn’t run out of memory.
  - Initialize the value field.
  - Initialize the next field (specifically, to NULL).
  - Return a pointer to your newly created node.



- Create a linked list:
  - Dynamically allocate space for a new (your first!) node.
  - Check to make sure you didn’t run out of memory.
  - Initialize the value field.
  - Initialize the next field (specifically, to NULL).
  - Return a pointer to your newly created node.

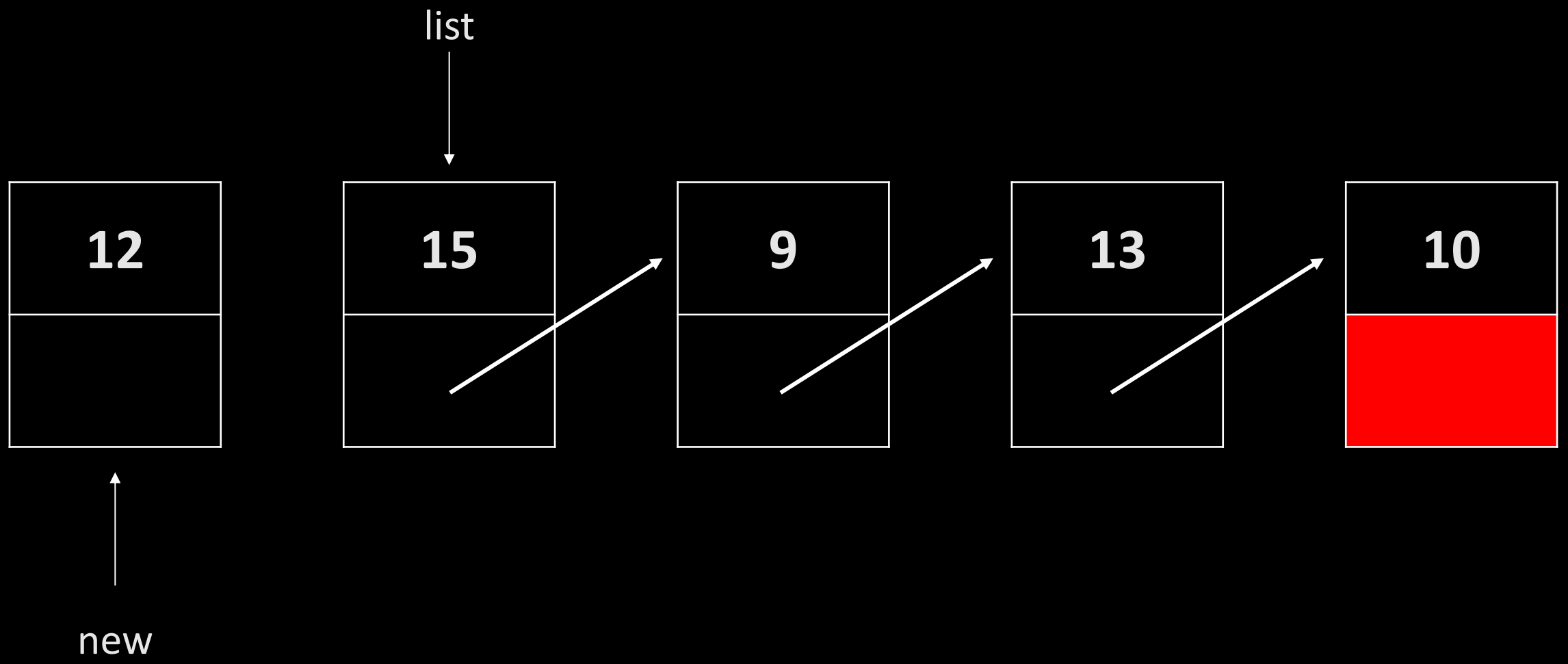


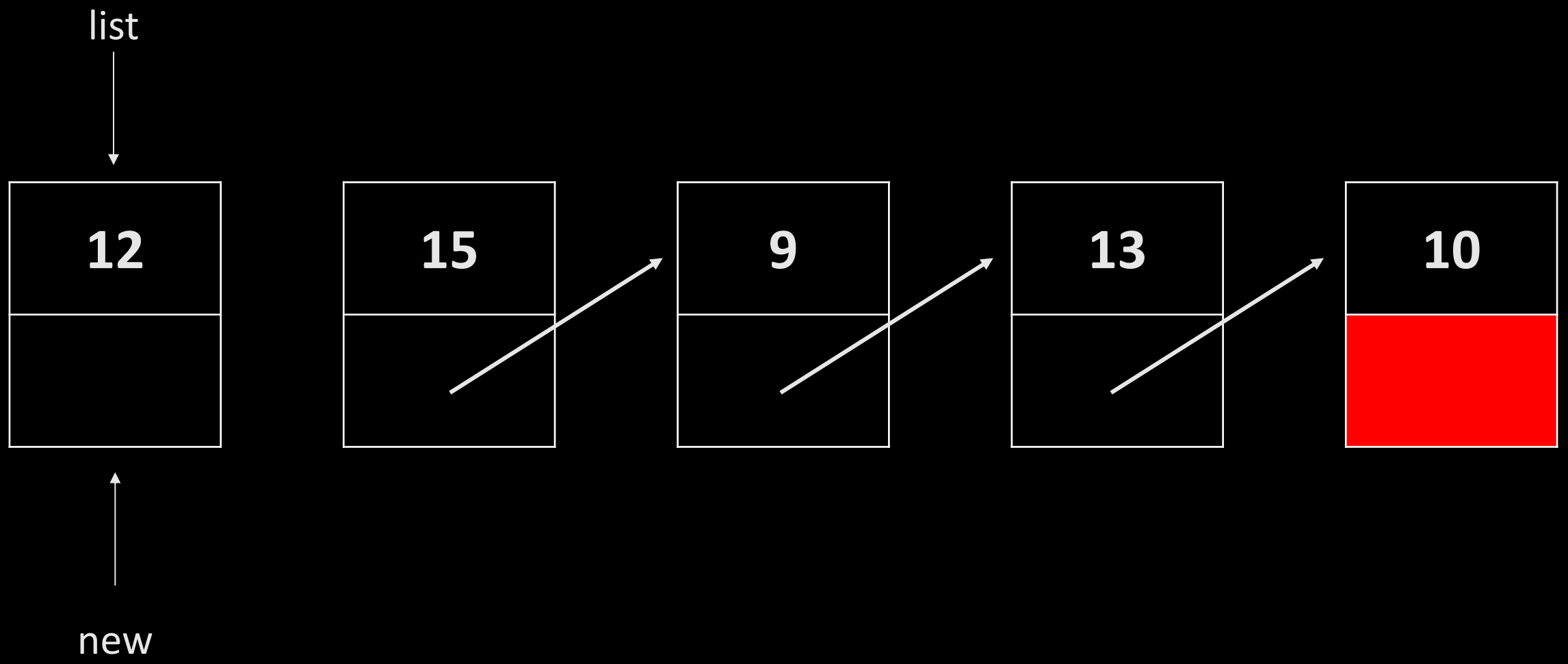
- Find an element:
  - Create a traversal pointer pointing to the list's head (first element).
  - If the current node's value field is what we're looking for, return true.
  - If not, set the traversal pointer to the next pointer in the list and go back to the previous step.
  - If you've reached the element of the list, return false.

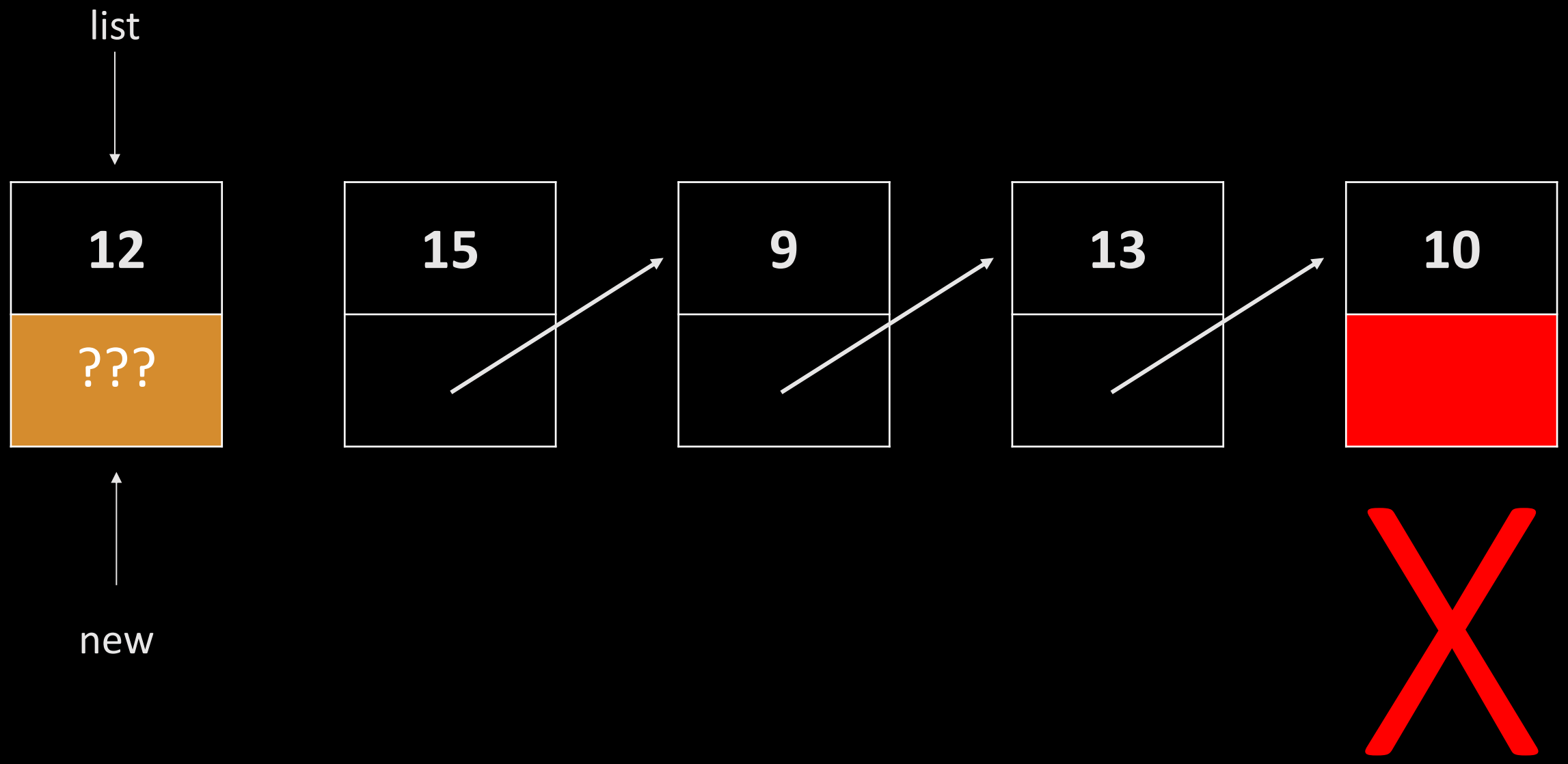
- Insert an element:
  - Dynamically allocate space for a new linked list node.
  - Populate and insert the node at the beginning of the linked list.
  - Return a pointer to the new head of the linked list.

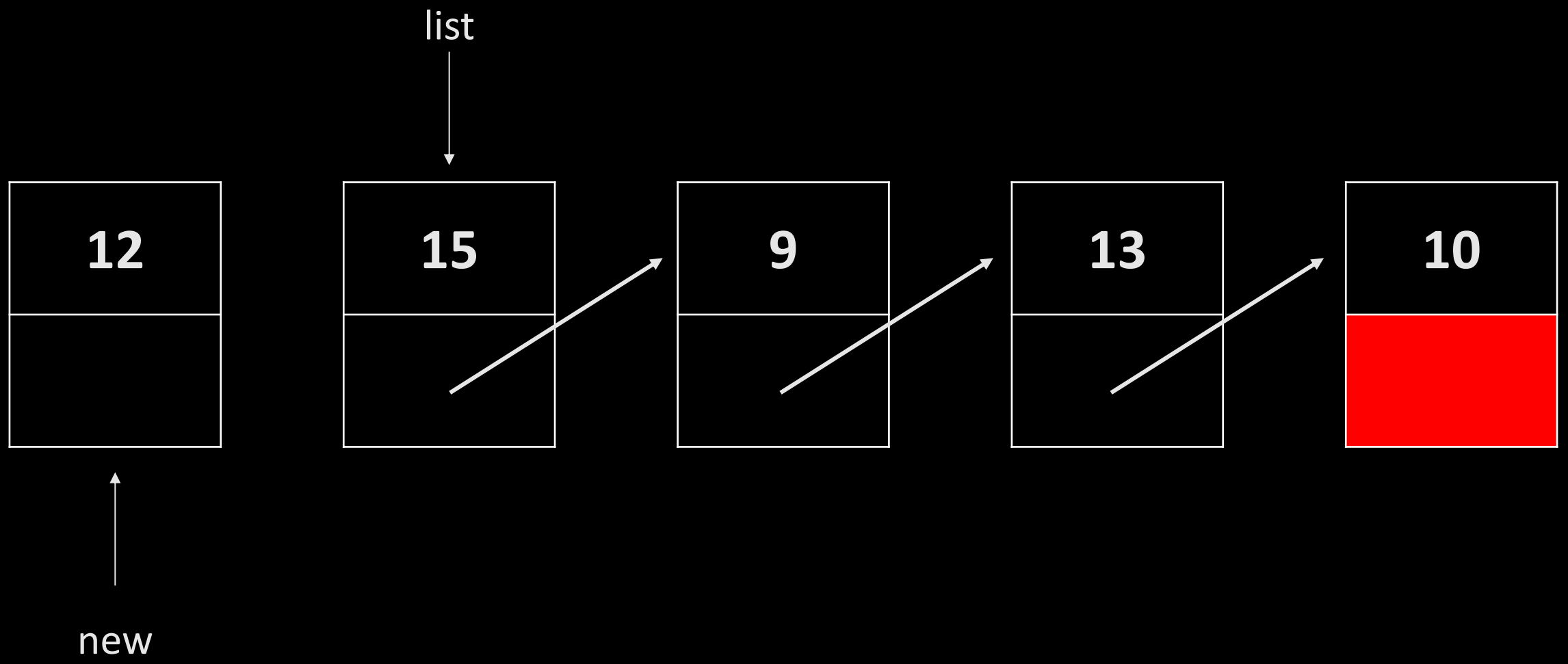
- Insert an element:
  - Dynamically allocate space for a new linked list node.
  - Check to make sure we didn't run out of memory.
  - Populate and insert the node at the beginning of the linked list.
    - So which pointer do we move first? The pointer in the newly created node, or the pointer pointing to the *original* head of the linked list?
    - This choice matters!
  - Return a pointer to the new head of the linked list.

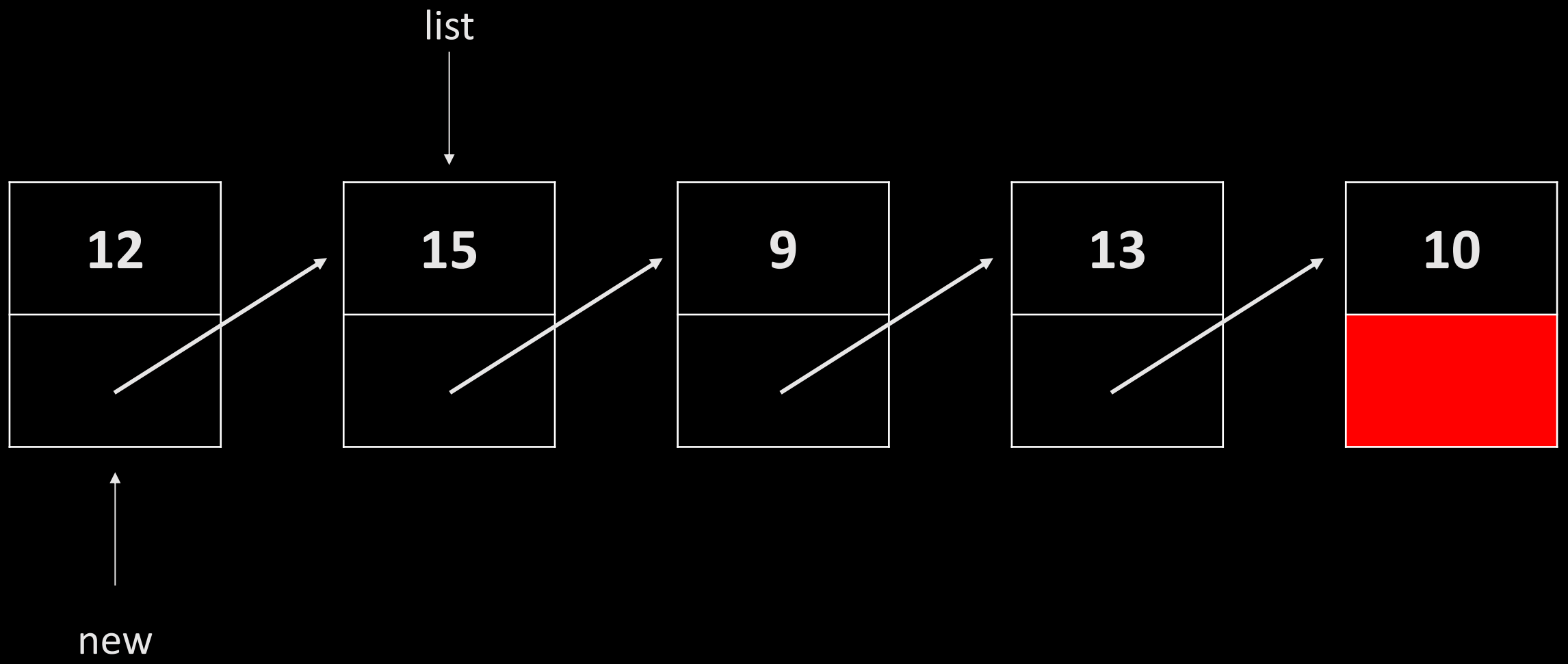
ORDER MATTERS!

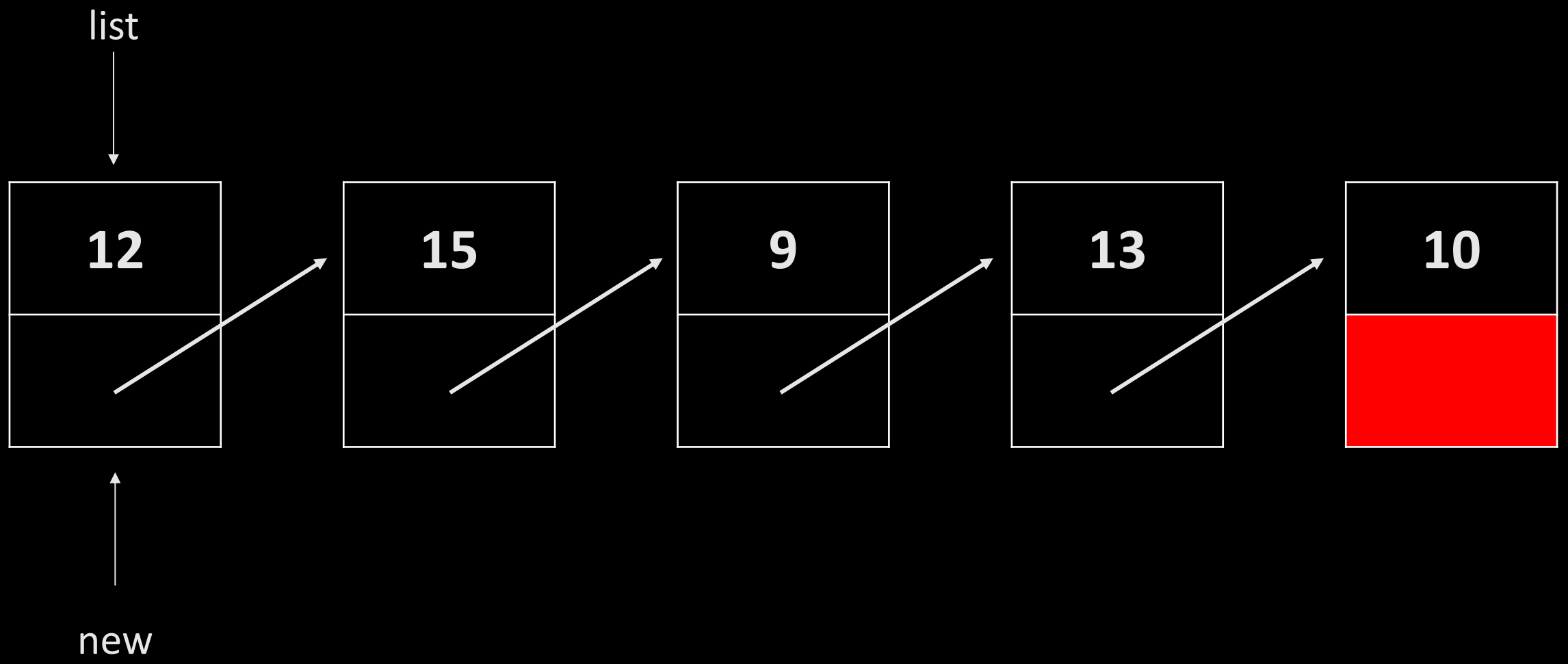

list
12
15
9
13
10
new


list
12
15
9
13
10
new


list
12
???
15
9
13
10
new


list
12
15
9
13
10
new


list
12
15
9
13
10
new


list
12
15
9
13
10
new

- Delete an entire linked list:
  - If you've reached a NULL pointer, stop.
  - Delete the rest of the list.
  - Free the current node.

- Sounds like recursion!

Exercise

Write a program that prompts the user to type in integers, adds each integer one at a time to the head of a linked list, and then prints out the integers in the linked list (they'll be in reverse order from the input).

User should stop typing in integers when 5 is inputted.

If you finish early, change the implementation to add a new node to the end of the list!

```
#include <cs50.h>
#include <stdio.h>

typedef struct node
{
    int number;
    struct node *next;
} node;
```

```
int main(void)
{
    node *list = NULL;
    while (true)
    {
        int x = get_int("Number: ");
        if (x == 5)
        {
            printf("\n");
            break;
        }
    // TODO: Allocate a new node.
    // TODO: Add new node to head of linked list.
    }
// TODO: Print all nodes.
// TODO: Free all nodes.
}
```

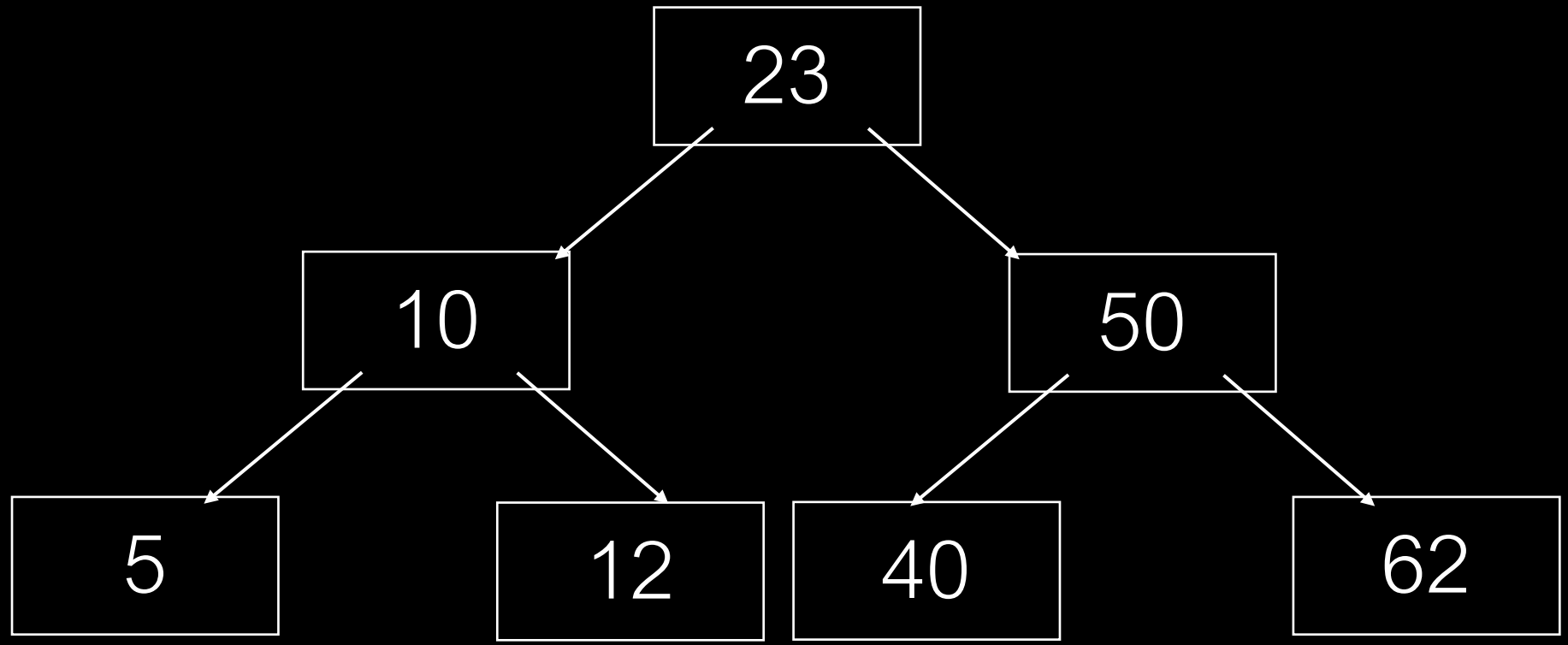
Exercise

Binary Search Trees

Binary Search Tree

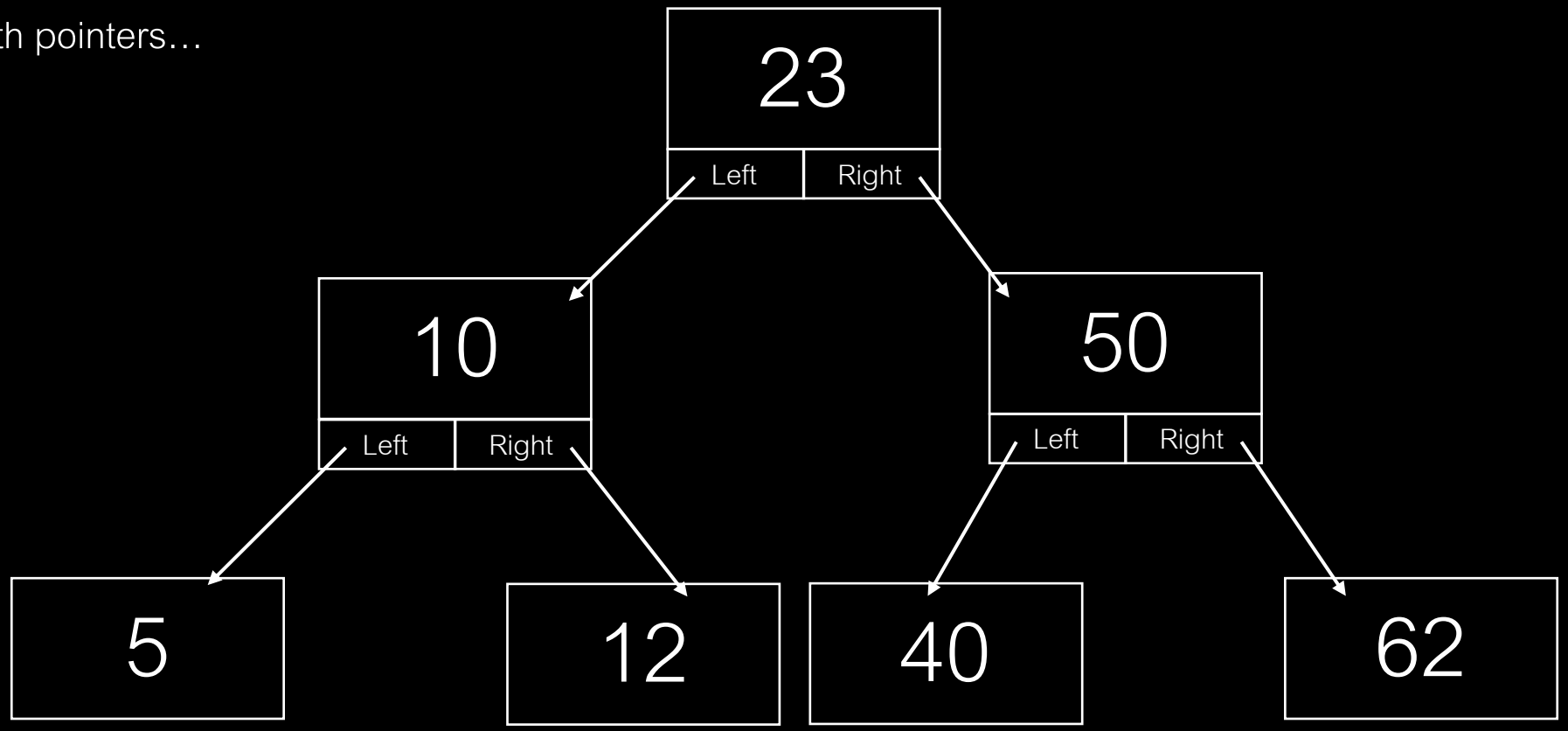
- Let's turn this list into a data structure!

5	10	12	23	40	50	62



Binary Search Tree

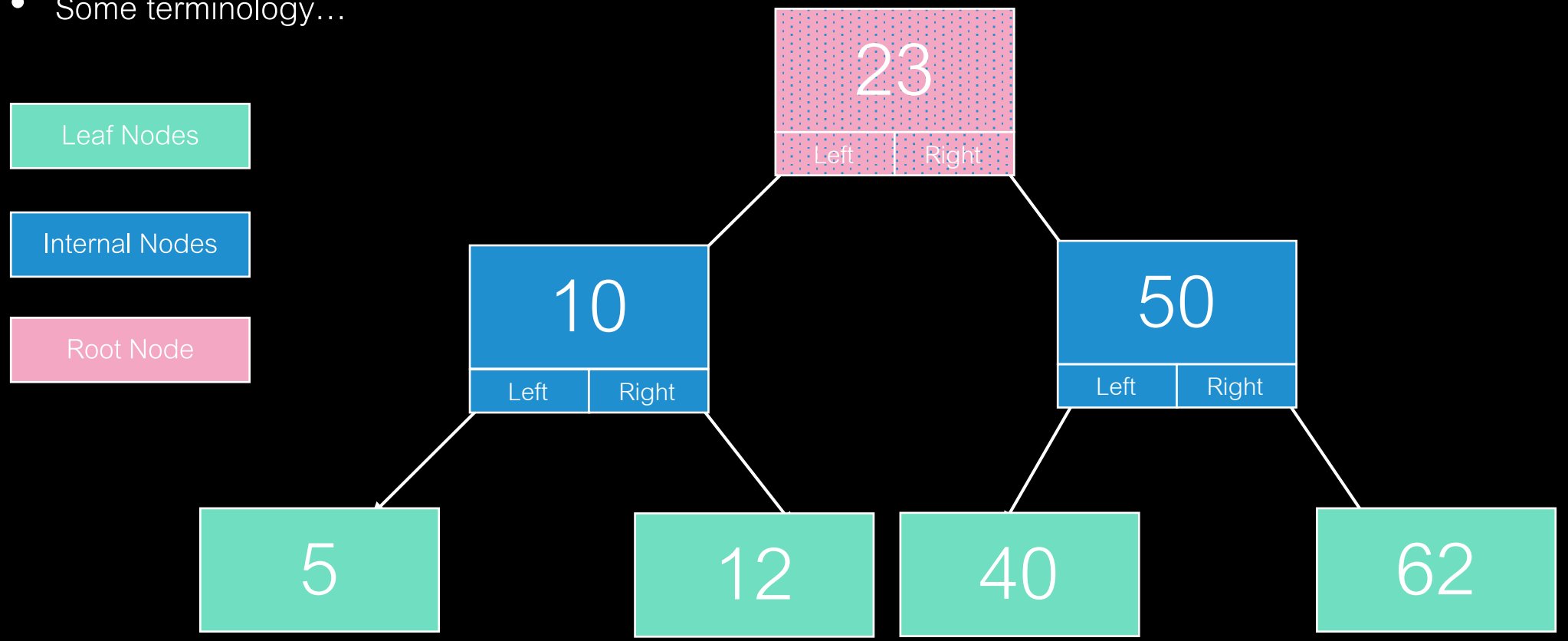
- With pointers…



- At each memory address, there is a node containing a value, an address to the left child, and an address to the right child.

Binary Search Tree

- Some terminology…

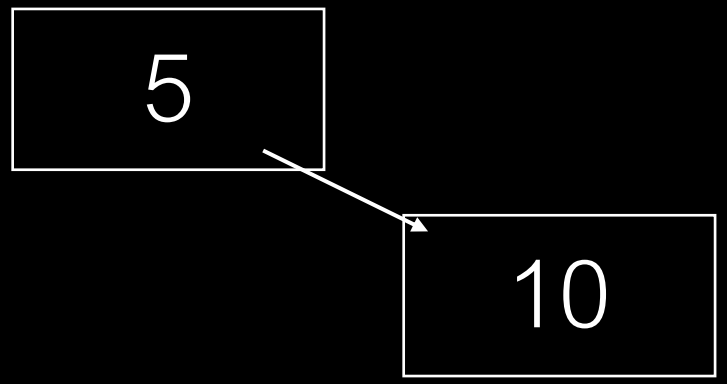


Binary Search Tree – Insertion

- Now to create a binary search tree one node at a time from these numbers… 5, 10, 30, 50

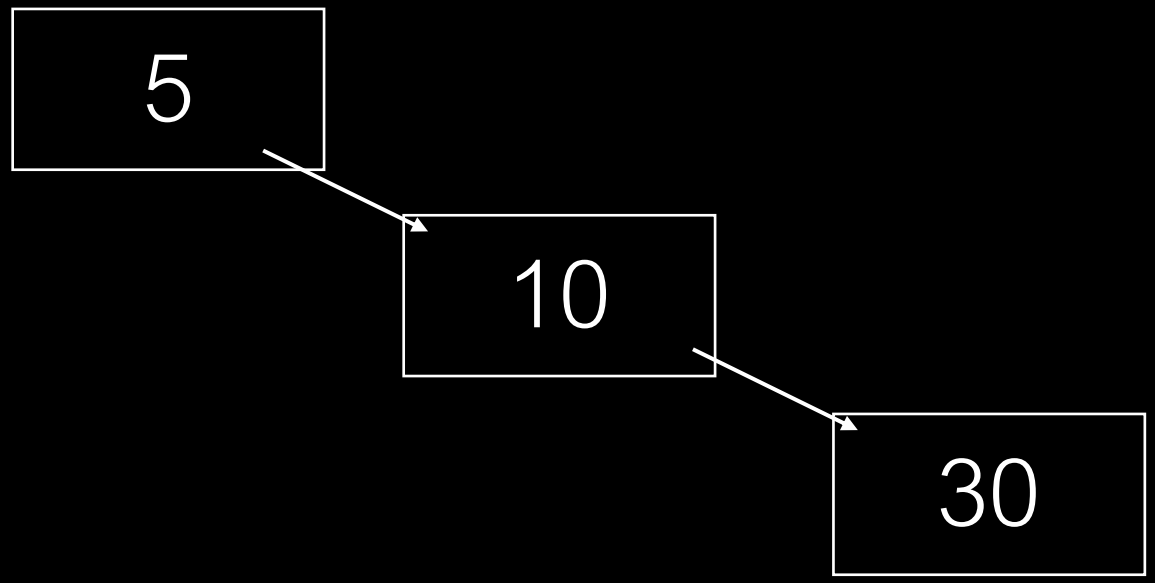
Binary Search Tree – Insertion

- Now to create a binary search tree one node at a time from these numbers… 5, 10, 30, 50



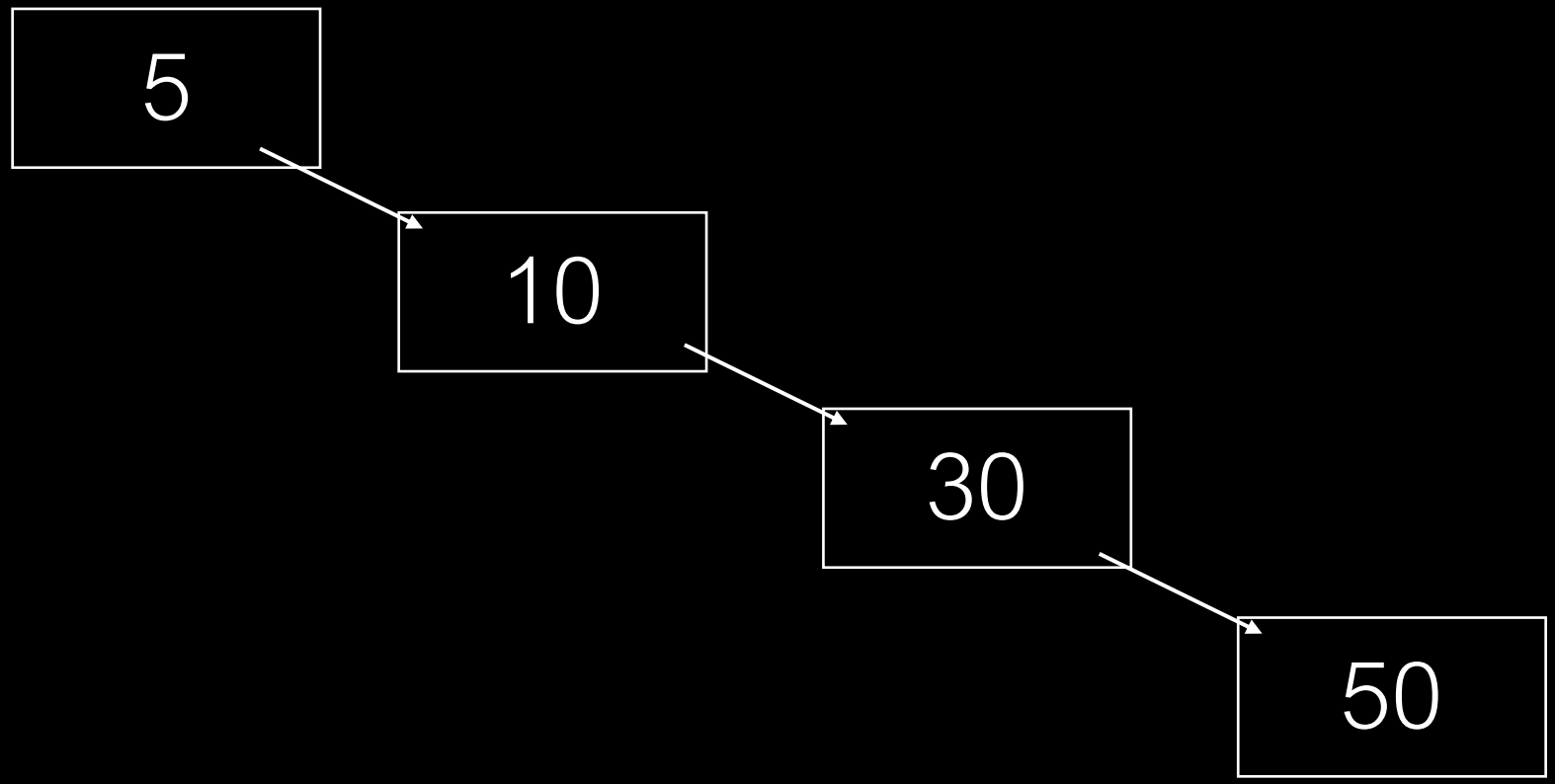
Binary Search Tree – Insertion

- Now to create a binary search tree one node at a time from these numbers… 5, 10, 30, 50



Binary Search Tree – Insertion

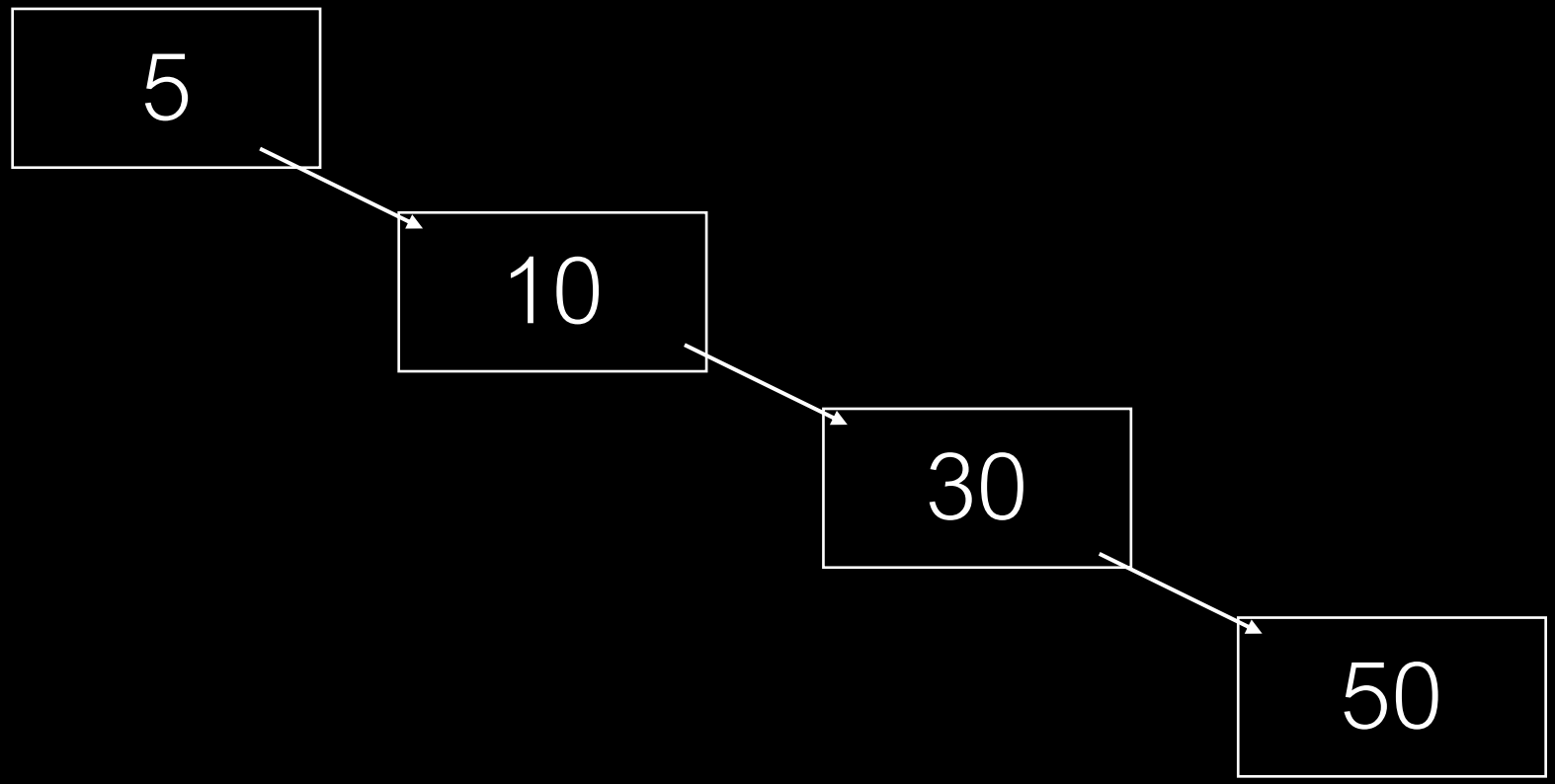
- Now to create a binary search tree one node at a time from these numbers… 5, 10, 30, 50



- What data structure does this look like?

Binary Search Tree

- Let's search for 50…



Efficiency

Binary Search Tree

- Searching:

Efficiency

Binary Search Tree

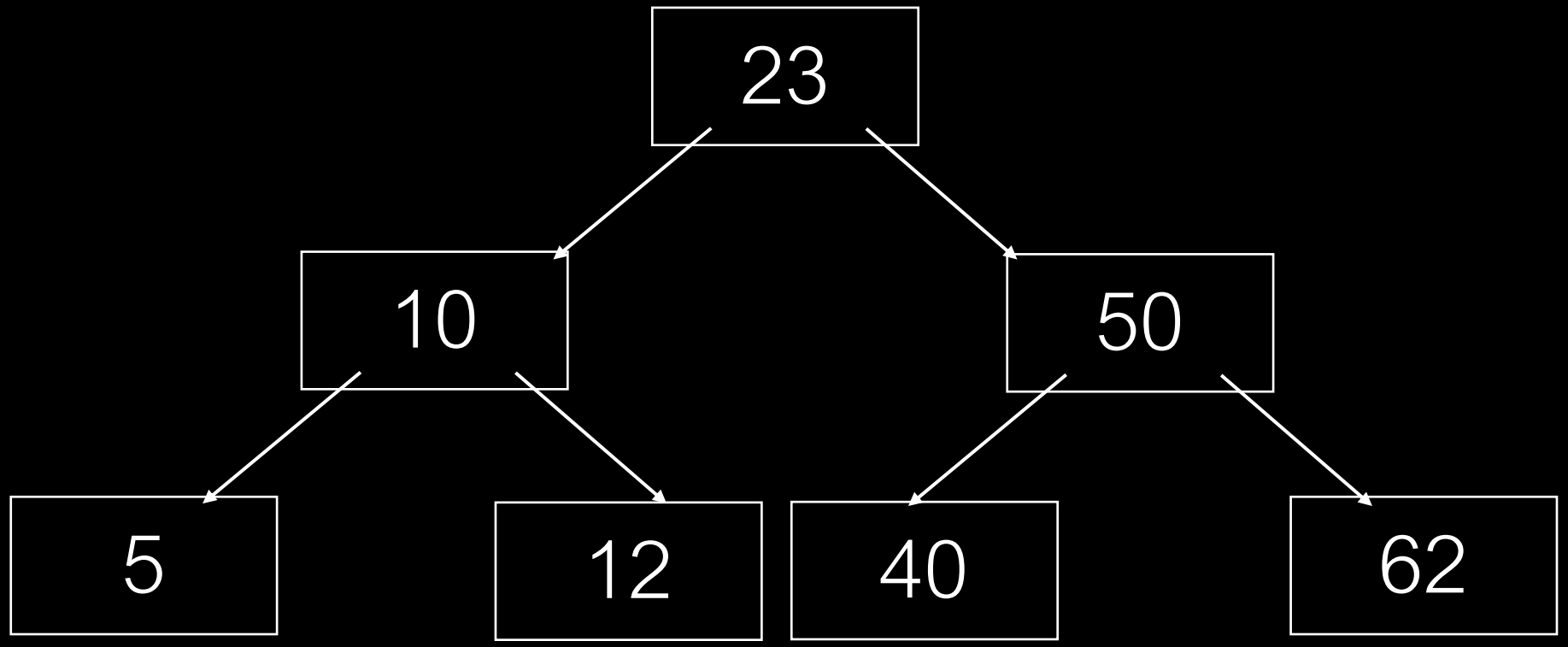
- Searching: O(n)
- Insertion/Deletion:

Efficiency

Binary Search Tree

- Searching: O(n)
- Insertion/Deletion: O(n)

Efficiency



Efficiency

Binary Search Tree

- Searching: O(n)
- Insertion/Deletion: O(n)

Balanced Binary Search Tree

- Searching:
- Insertion/Deletion:

Efficiency

Binary Search Tree

- Searching: O(n)
- Insertion/Deletion: O(n)

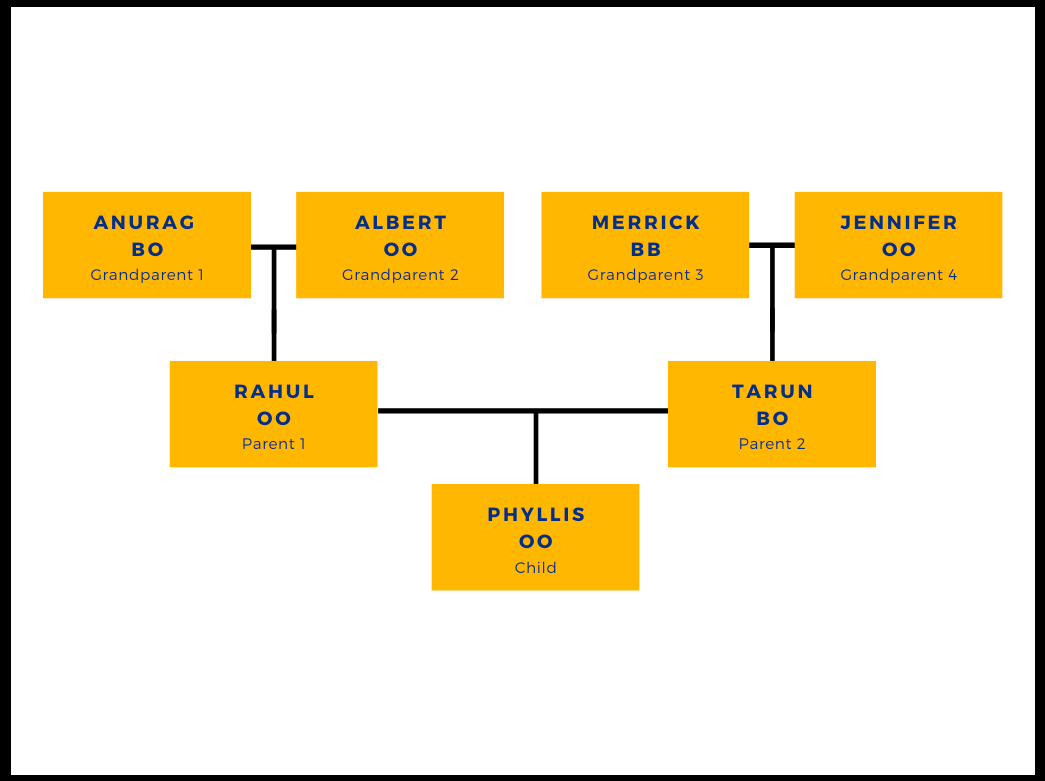
Balanced Binary Search Tree

- Searching: O(log n)
- Insertion/Deletion: O(log n)

Stacks

Queue

Lab


ANURAG
BO
Grandparent 1
ALBERT
OO
Grandparent 2
MERRICK
BB
Grandparent 3
JENNIFER
OO
Grandparent 4
RAHUL
OO
Parent 1
TARUN
BO
Parent 2
PHYLLIS
OO
Child

Hashtables