Review Session 1

R.J. Aquino, ’14

Quiz 1

Quiz 1 Information

- https://cs50.harvard.edu/quizzes/2013/1
- Cumulative, but with an emphasis on material covered since Quiz 0
- Typically more challenging than Quiz 0
- Use CS50 Discuss and take practice quizzes!

Quiz 1 Review Session

- This is NOT an exhaustive list of topics
- This is NOT necessarily everything you need to know about any given topic
- This IS meant to review topics we covered in lecture and section

File I/O

Week 7 Monday, Section 6, Problem Set 5

File I/O

- fopen, fclose, fwrite, fread, fseek
- You should be pretty familiar with these functions after pset5!
- What are common file-related bugs?
  - Forgetting to check if fopen returned NULL or succeeded
  - Forgetting to fclose a file that you fopen'd
  - Forgetting to check if you have reached the end of a file

Structs

Week 7 Monday

Structs

```
// structure representing a student
typedef struct
{
     string name;
     int age;
}
student;
```

Structs, cont.

```
// declare an instanct of struct like any variable
student s;

// set fields of a struct with '.'
s.name = "RJ";
s.age = 21;

// update fields the same way
s.name = "R.J.";

// access fields the same way
printf("%s is %d years old\n.", s.name, s.age);
```

Structs, cont.

```
// you often will have a pointer to a struct
student* ptr = &s;

// to get to the fields, you first need to dereference
(*ptr).age = 22;

// the arrow syntax is a nice shortcut for this!
ptr->age = 22;
```

Data Structures

Data Structures

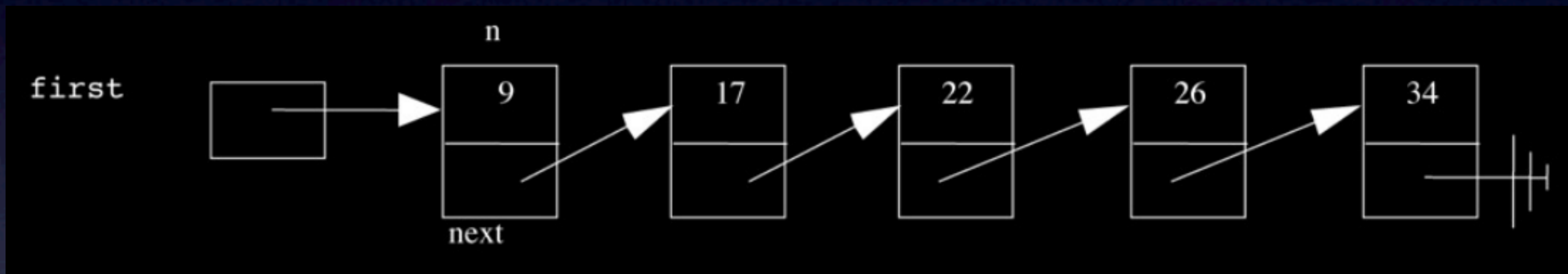
1. Understand each structure at a high level
   - Can you explain how it works in English?
2. Understand the implementation/operations
   - E.g., can you insert into a linked list?
   - Can you write C code related to these structures?
   - Understand pointers and structs
3. Know the runtimes/limitations
   - E.g., how fast is a hash table lookup?
   - Understand "Big-O" notation

Linked Lists

Week 7 Monday and Wednesday, Section 7

Linked Lists

High Level



Linked Lists

High Level

- Easy to insert - $O(1)$ for unsorted lists
- Hard to find - $O(n)$
- Compare with arrays - when is a linked list better? When is an array better?

Linked Lists

Implementation

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

Linked Lists

Implementation

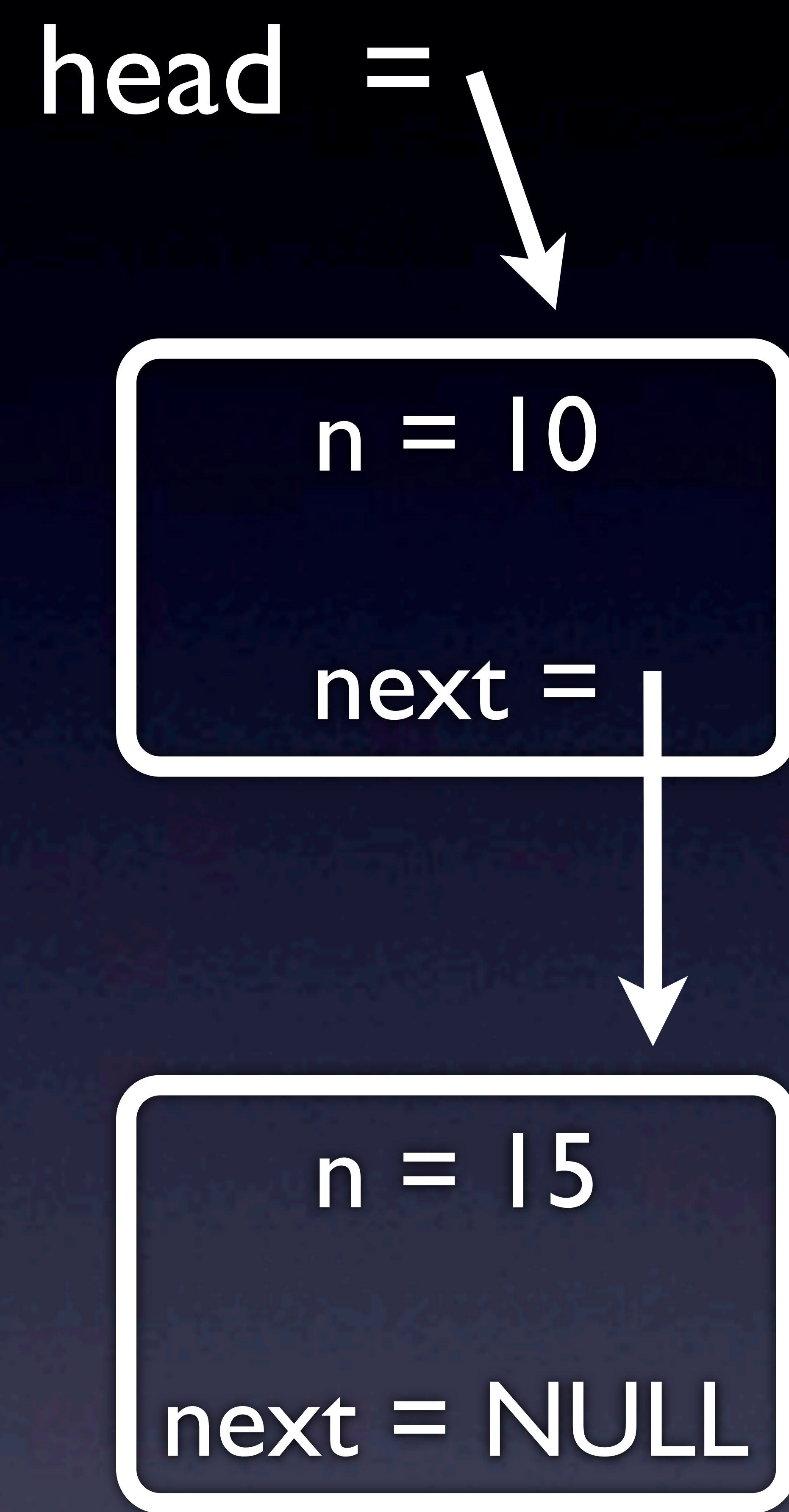
```
typedef struct node
{
    int n;
    struct node* next;
}
node;
```

Could be any type. In pset6, we stored char* or char arrays!

Linked Lists

Operations

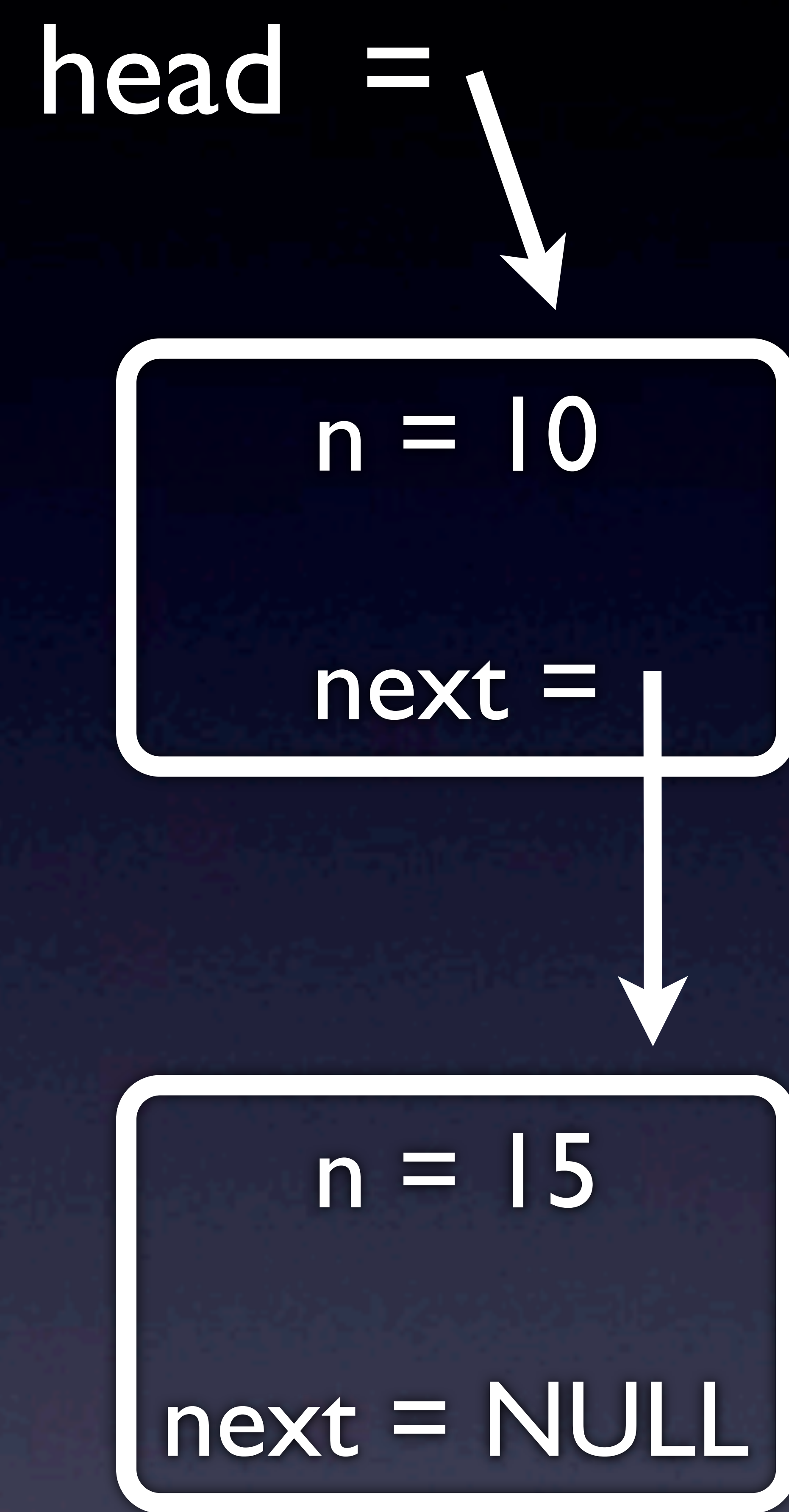
```
node* head;
bool insert(int new_n)
{



}
```



Linked Lists

Operations

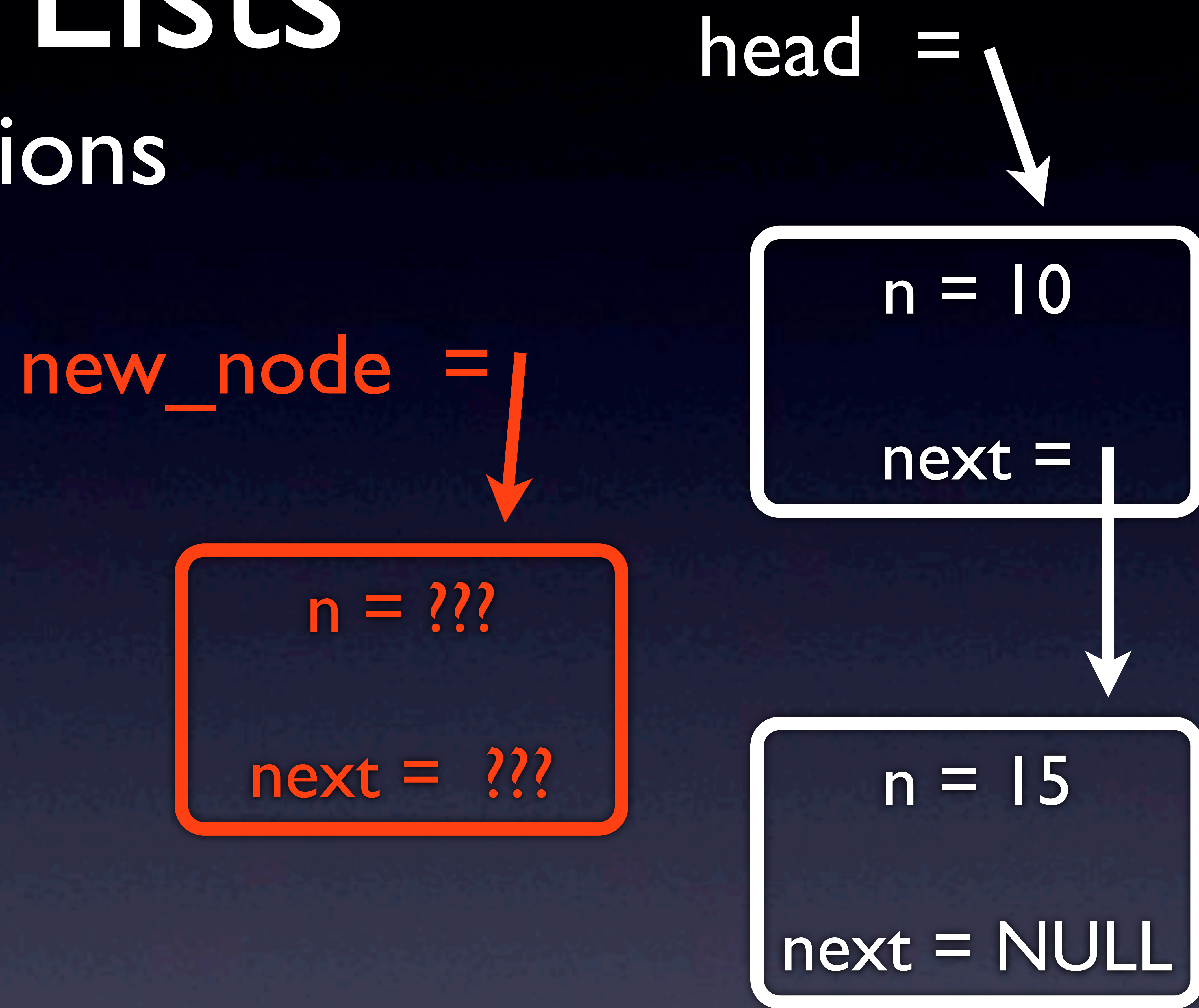
```
node* head;
bool insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
       return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

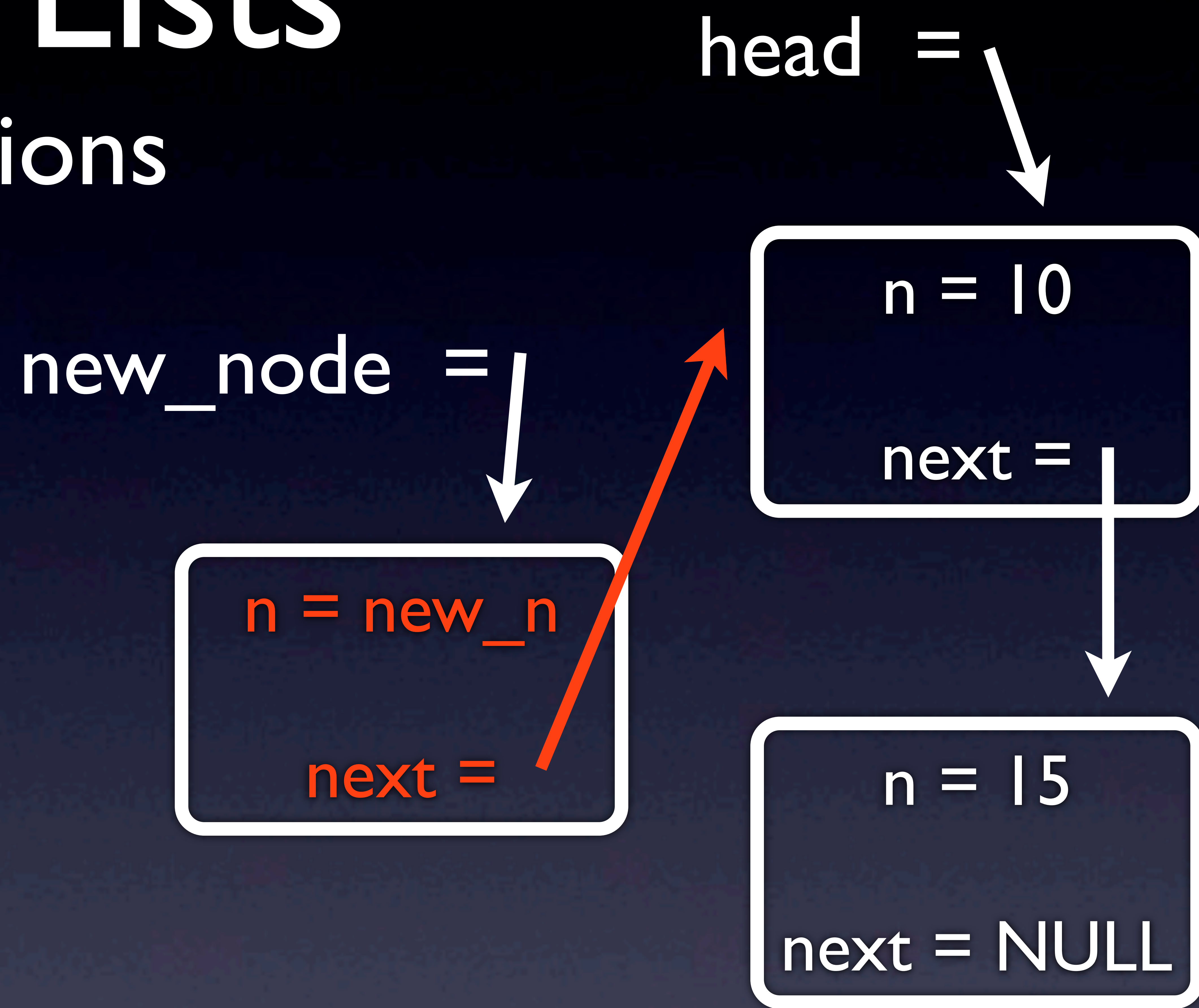
Operations

```
node* head;
bool insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
        return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists
Operations

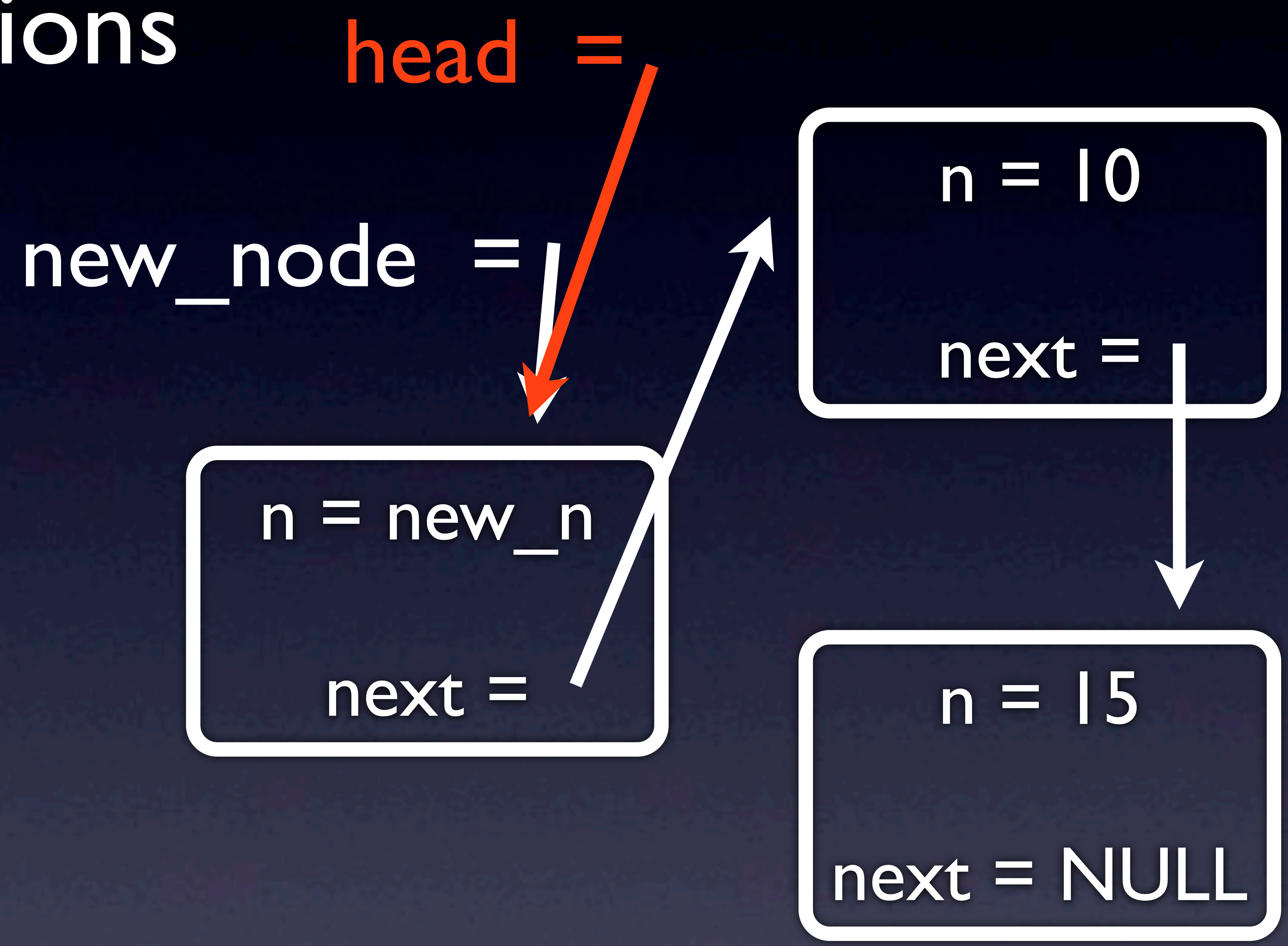
```
node* head;
bool insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
        return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

Operations

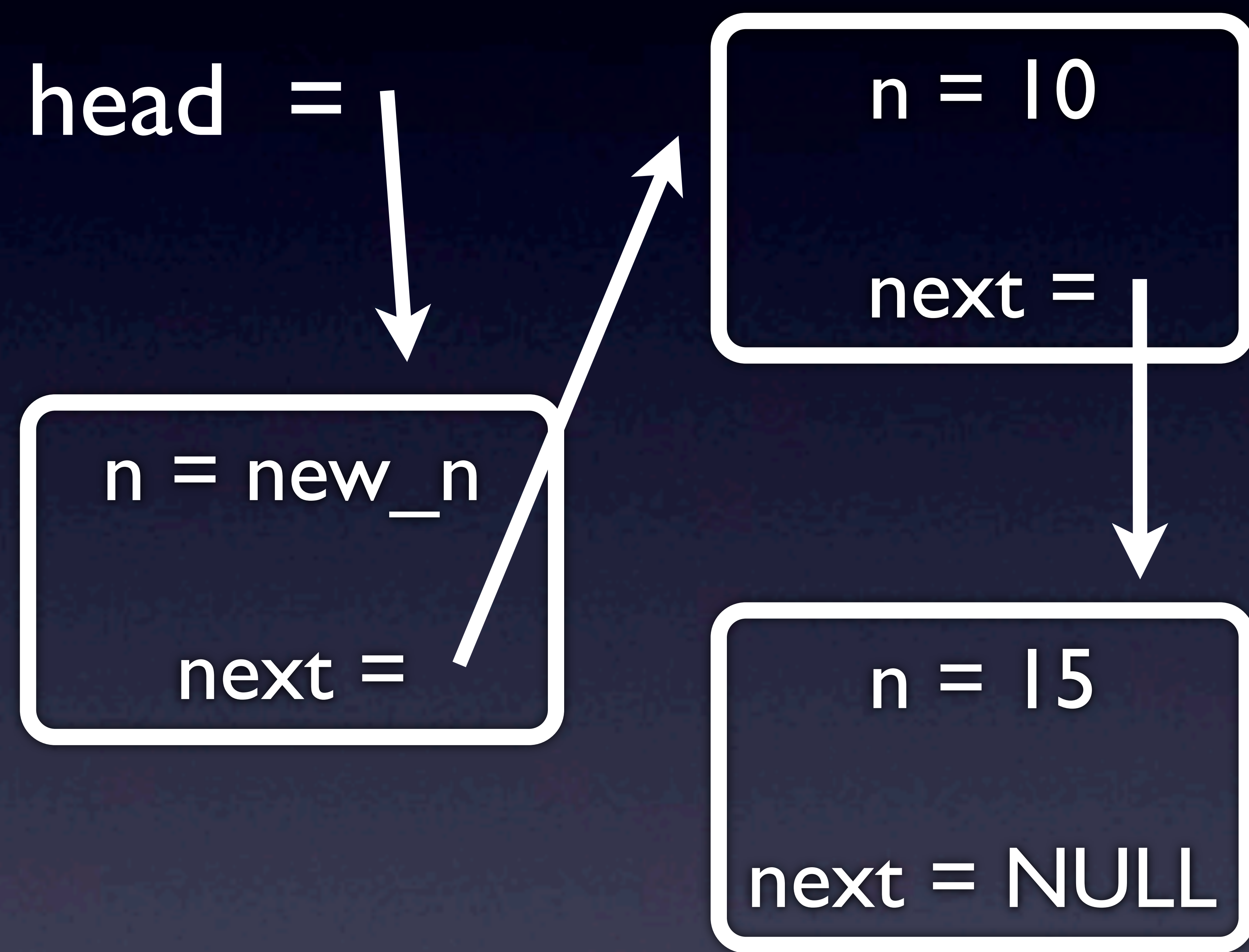
```
node* head;
void insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
        return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists

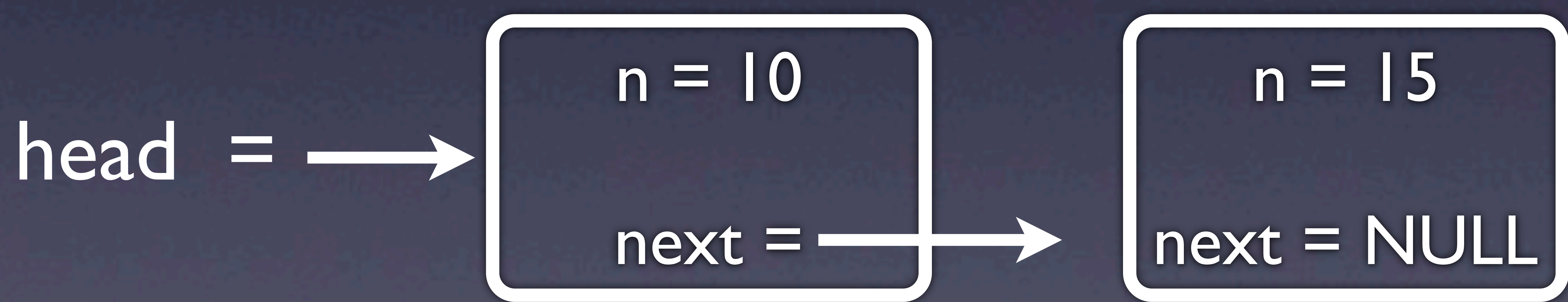
Operations

```
node* head;
void insert(int new_n)
{
    // make a new node
    node* new_node = malloc(sizeof(node));
    if (new_node == NULL)
    {
        return false;
    }

    // add value to node
    new_node->n = new_n;
    new_node->next = head;

    // set head to our new node
    head = new_node;
    return true;
}
```



Linked Lists
Operations

- When in doubt, draw a picture!
- Try to implement delete and find!
- Also note that there are “doubly” linked lists, where each node stores a “prev” pointer too!



Stacks

Week 8 Monday

Stacks

High Level

Stacks

High Level

- "Last in, first out" - LIFO
- Two operations - push and pop
- We can implement these functions using an array.

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

How would we implement push?

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

```
stack s;
bool push(int n)
{
    s.trays[s.size] = n;
    s.size++;
    return true;
}
```

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

Does this work?

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

Fails if size == CAPACITY

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

```
stack s;
bool push(int n)
{
    if (s.size == CAPACITY)
    {
        return false;
    }

    s.trays[s.size] = n;
    s.size++;
    return true;
}
```

Stacks

Array Implementation

```
typedef struct
{
    int trays[CAPACITY];
    int size;
}
stack;
```

What else could we ask about?

- implementation of pop
- non-array implementation
- non-int implemenation
- look at past quizzes!!

Queues

Week 8 Monday

Queues

- “First in, first out” - FIFO
- Two operations - enqueue, dequeue
- Again, can be implemented using an array

Queues

```
typedef struct
{
    int numbers[CAPACITY];
    int front;
    int size;
}
queue;
```

Queues

```
typedef struct
{
    int numbers[CAPACITY];
    int front;
    int size;
}
queue;
```

The index of the next element to dequeue (starts at 0)

Queues

```
typedef struct
{
    int numbers[CAPACITY];
    int front;
    int size;
}
queue;
```

Important things to keep track of:

- Wrapping around if front + size > CAPACITY

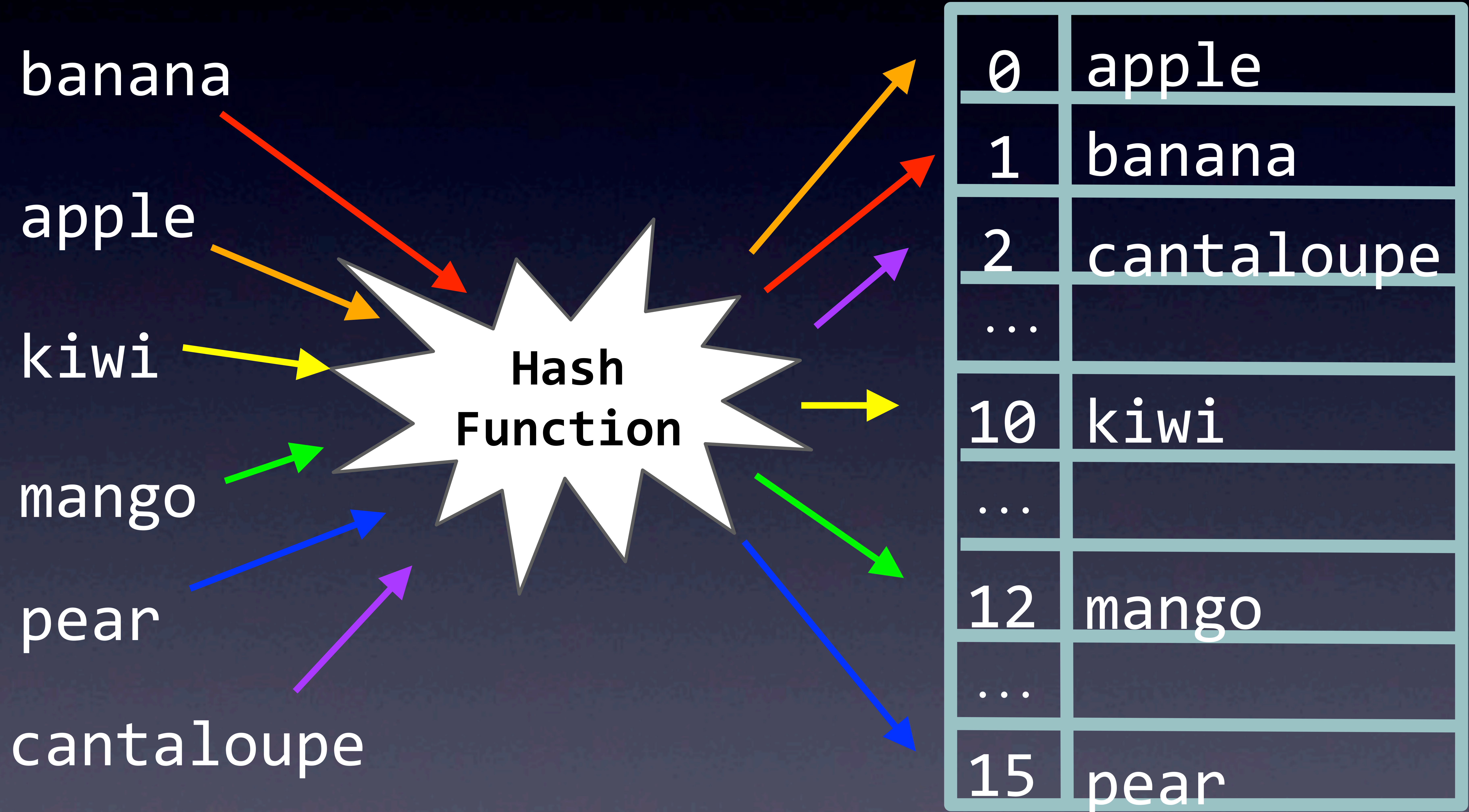
Hash Tables

Week 7 Wednesday, Section 7

Hash Tables

- A structure that aims for $O(1)$ insertion and $O(1)$ lookup
- In CS50, implemented as an array of linked lists
- Key component - hash function
  - Converts our input (say, a word) into a number
  - Used as an index into our array.

Hash Tables


banana
apple
kiwi
mango
pear
cantaloupe
Hash Function
0 apple
1 banana
2 cantaloupe
...
10 kiwi
...
12 mango
...
15 pear

Hash Tables

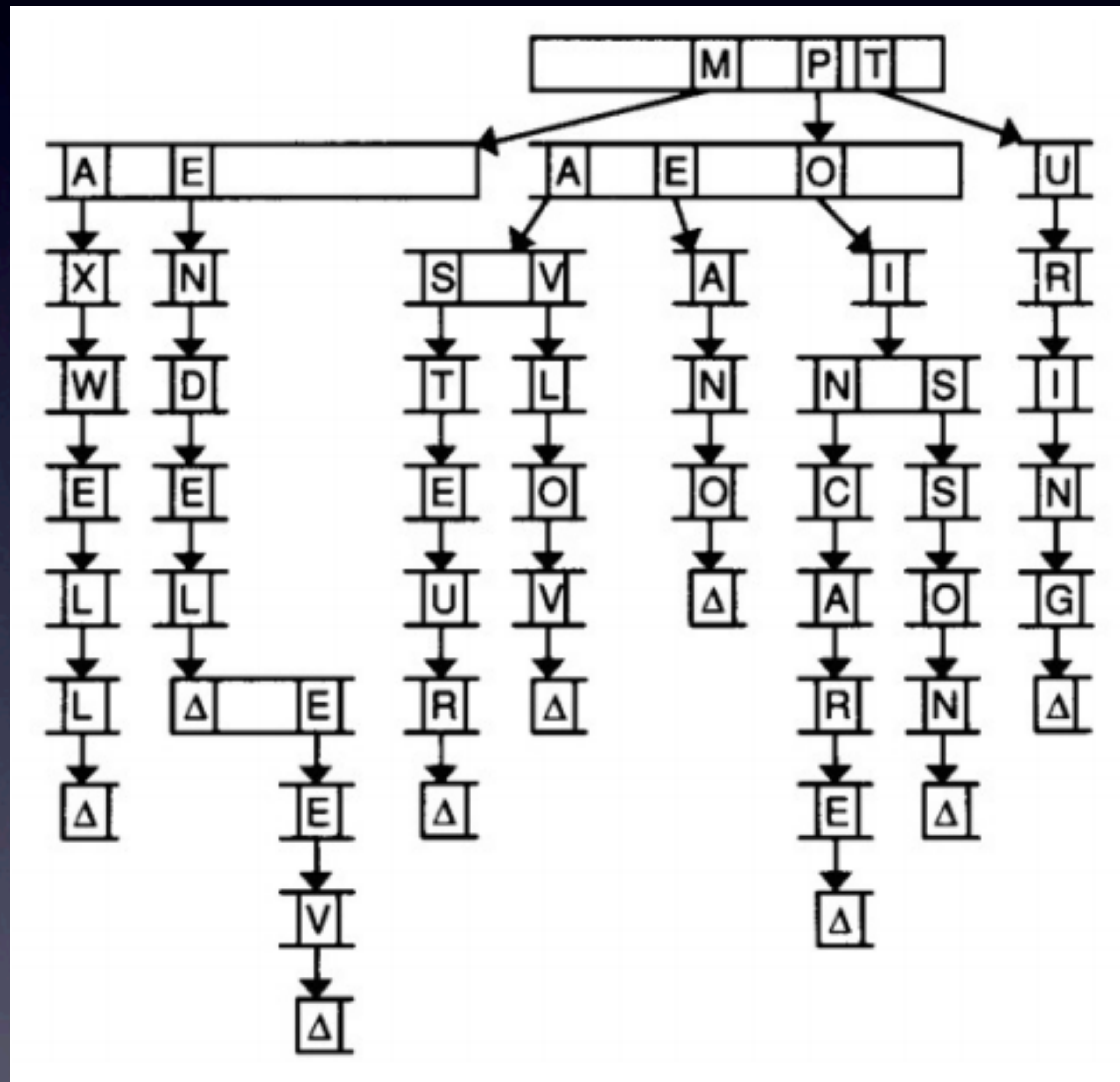
- What happens on collision?
  - Instead of storing one value at, say, hashtable[3], store a linked list!
  - Most of you implemented this for pset6, but check out Rob's postmortem for more implementation details!

Tries

Week 7 Wednesday

Tries

High Level



Tries

High Level

- Designed to store data alongside a keyword input, like a hash table.
- In the case of pset6, the data is “am I a word”
- Insertion and lookup in O(length of word)

Tries

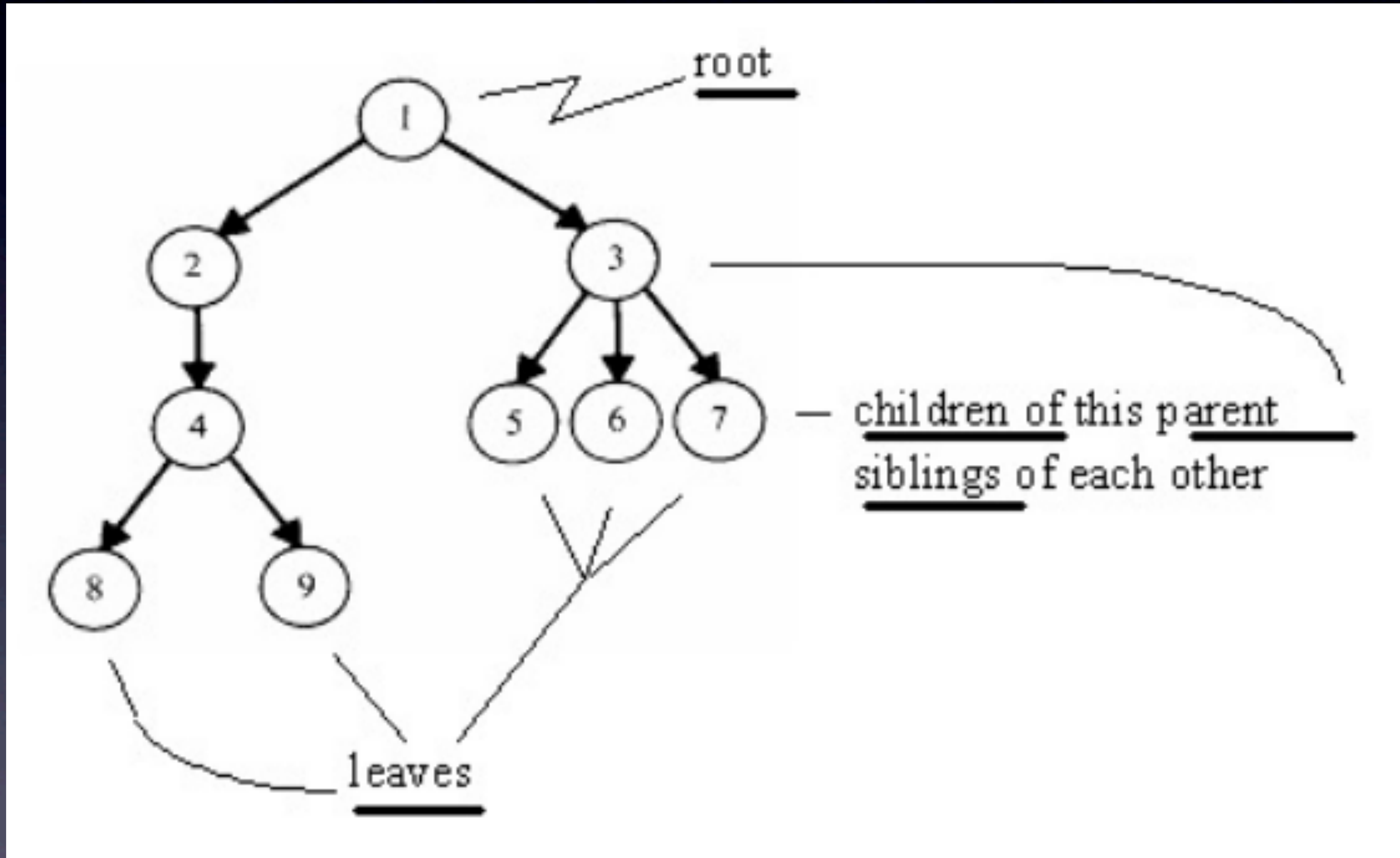
Implementation

```
typedef struct node
{
    bool is_word;
    struct node* children[27];
}
node;
```

Trees/Binary Search Trees

Week 8 Monday

Trees


root
1
2
3
4
5
6
7
8
9
children of this parent
siblings of each other
leaves

Trees

- Like a trie, a tree is a structure of nodes, where each node has 0 or more children. In a trie, we stated that each node had up to 27 children.
- A common type of tree is a "binary tree", where each node has 0, 1, or 2 children.

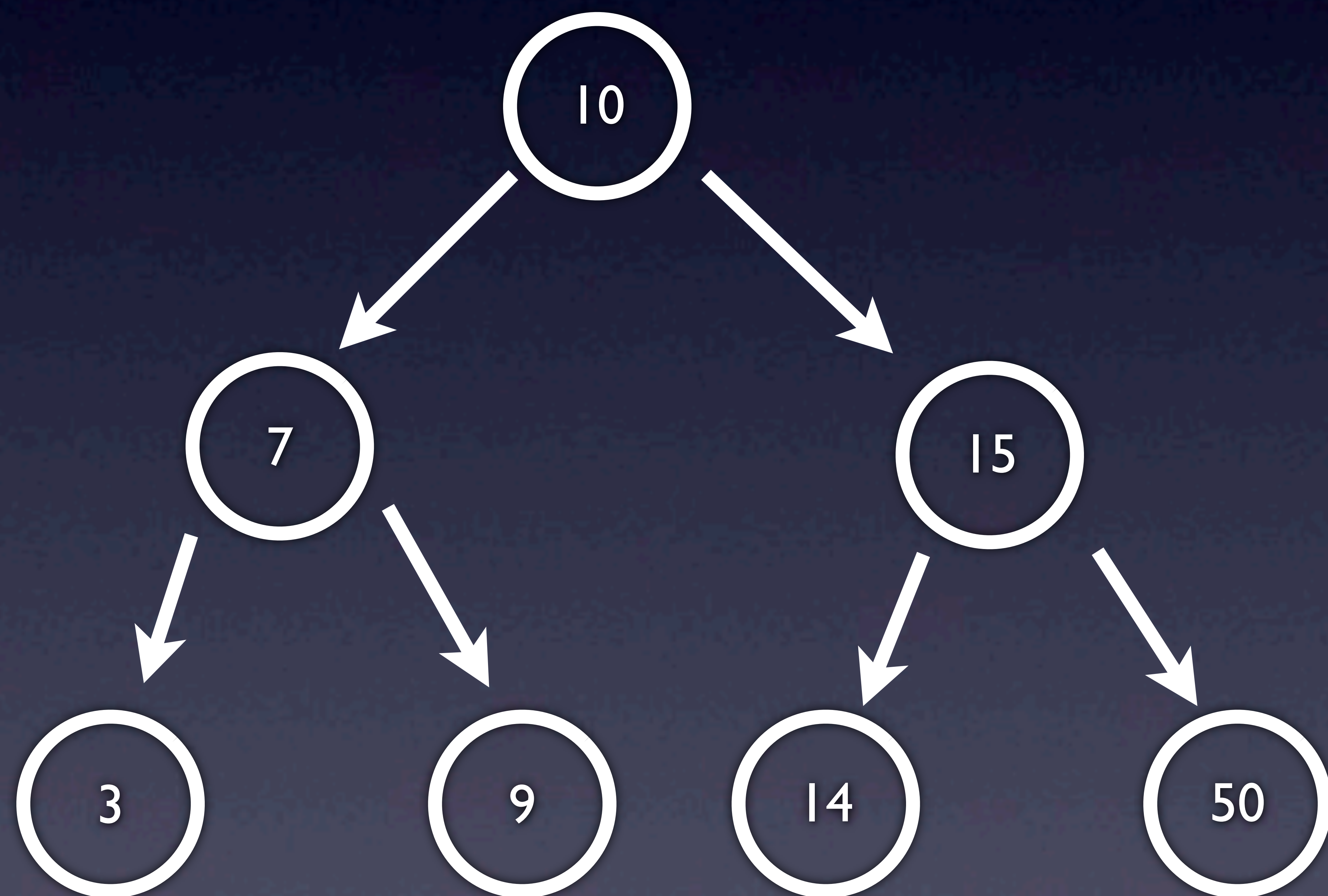
Binary Trees

```
typedef struct node
{
    int n;
    struct node* left;
    struct node* right;
}
node;
```

Binary Trees

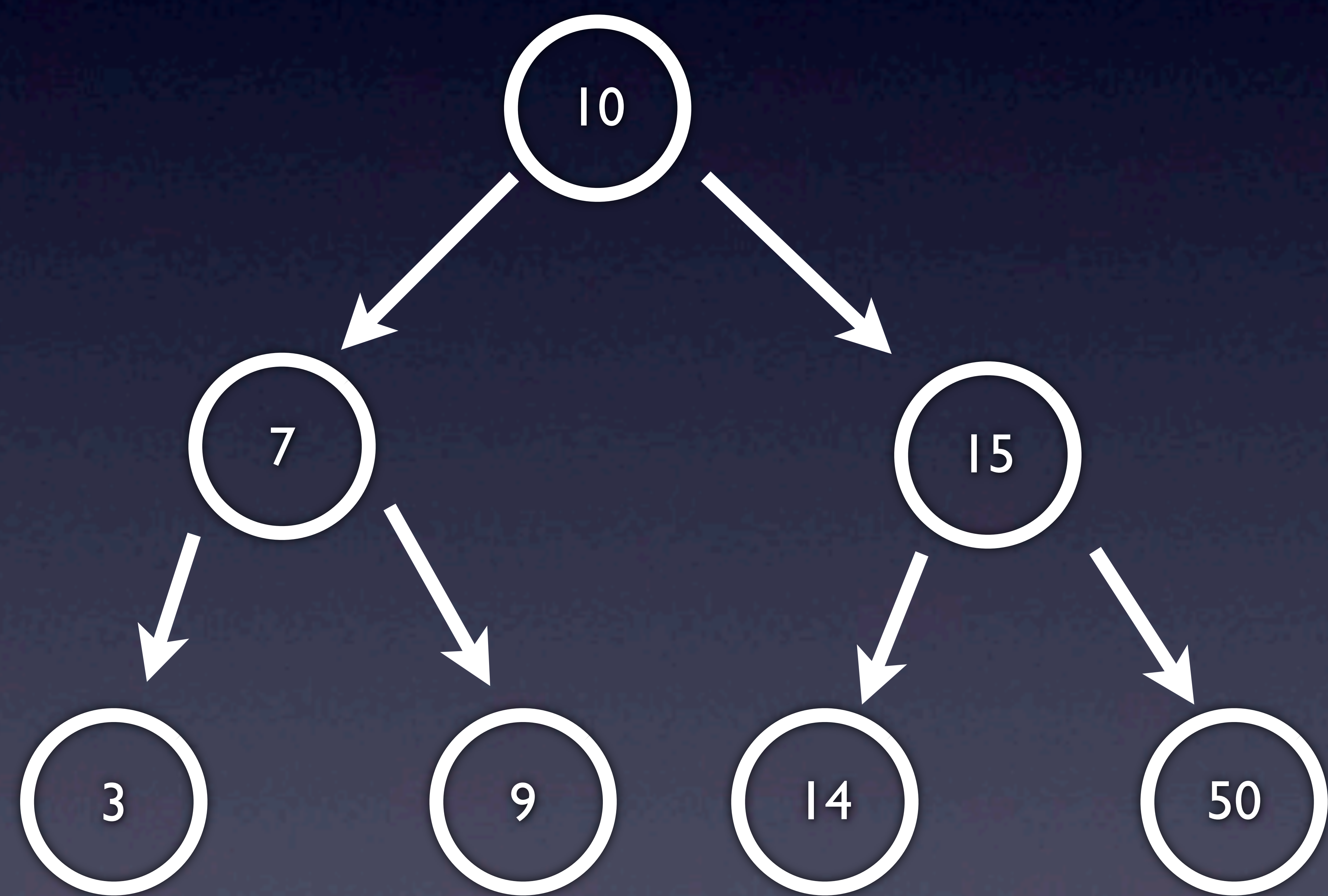
- How is a binary tree useful?
- If we make rules about where we put nodes, we can make search faster.
- In a binary search tree, all nodes on the left subtree of a node have a smaller value than the root node, and all nodes on the right subtree have a greater value than the root node.

"In a binary search tree, all nodes on the left subtree of a node have a smaller value than the root node, and all nodes on the right subtree have a greater value than the root node."



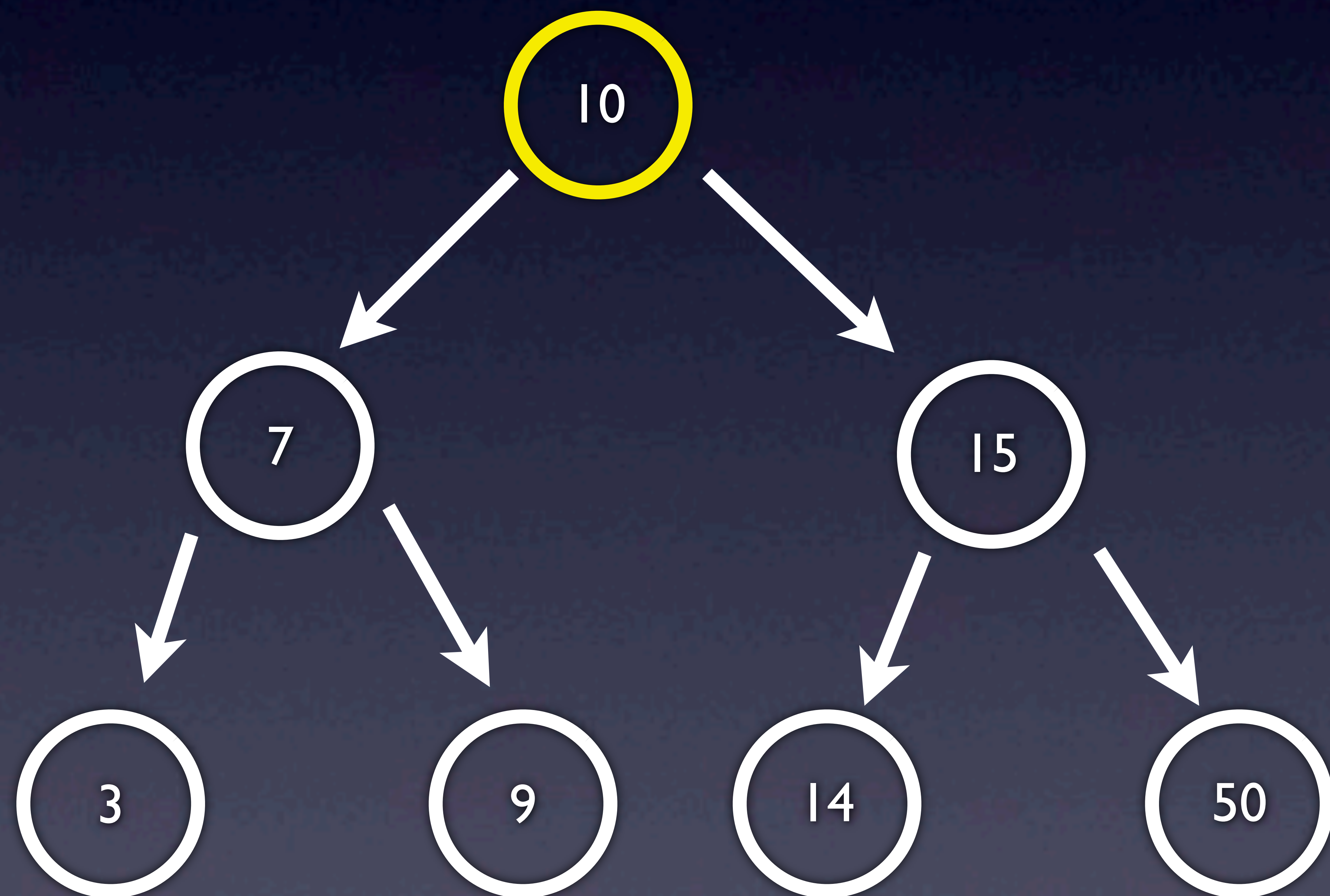
Search:

Find 14



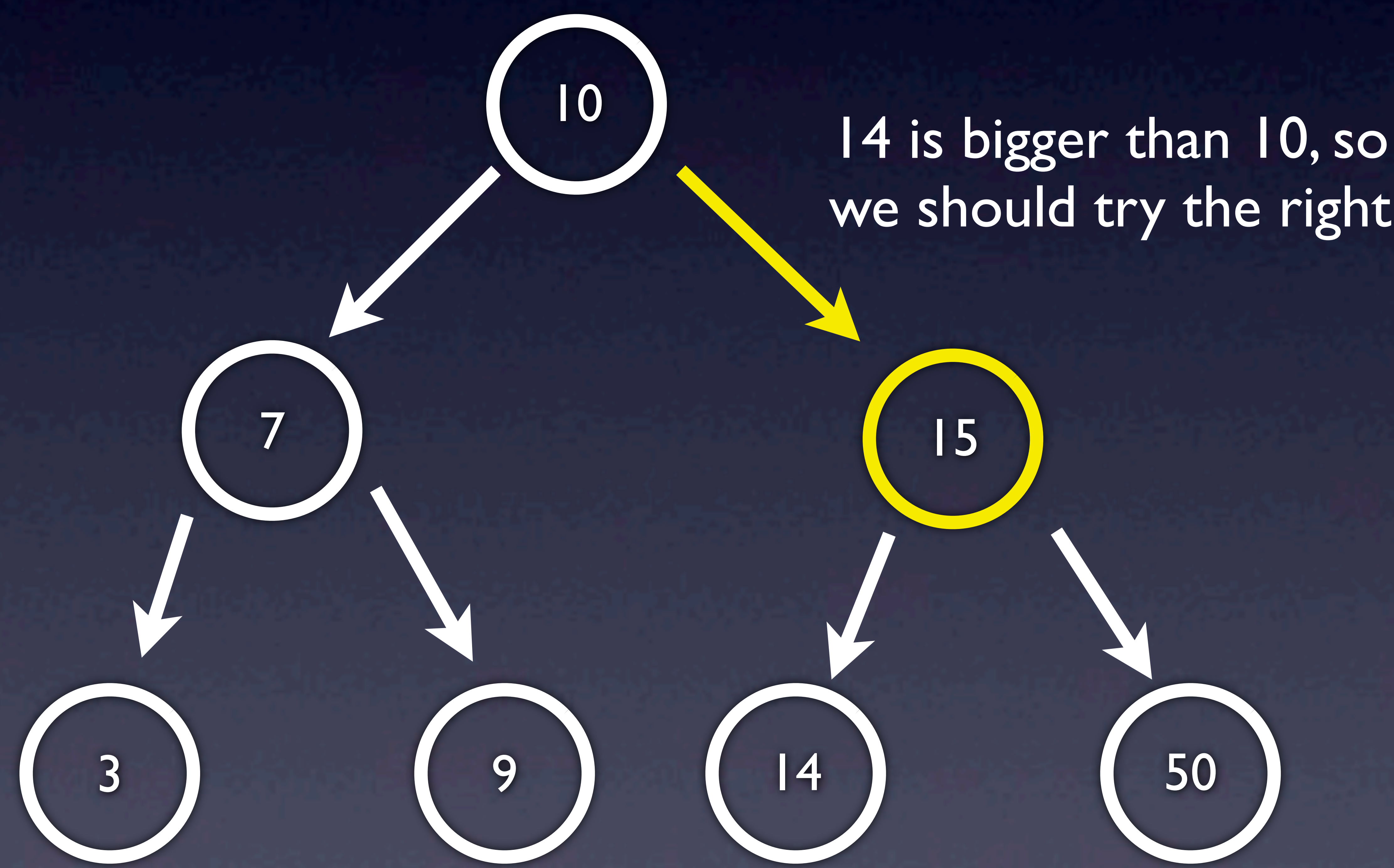
Search:

Find 14



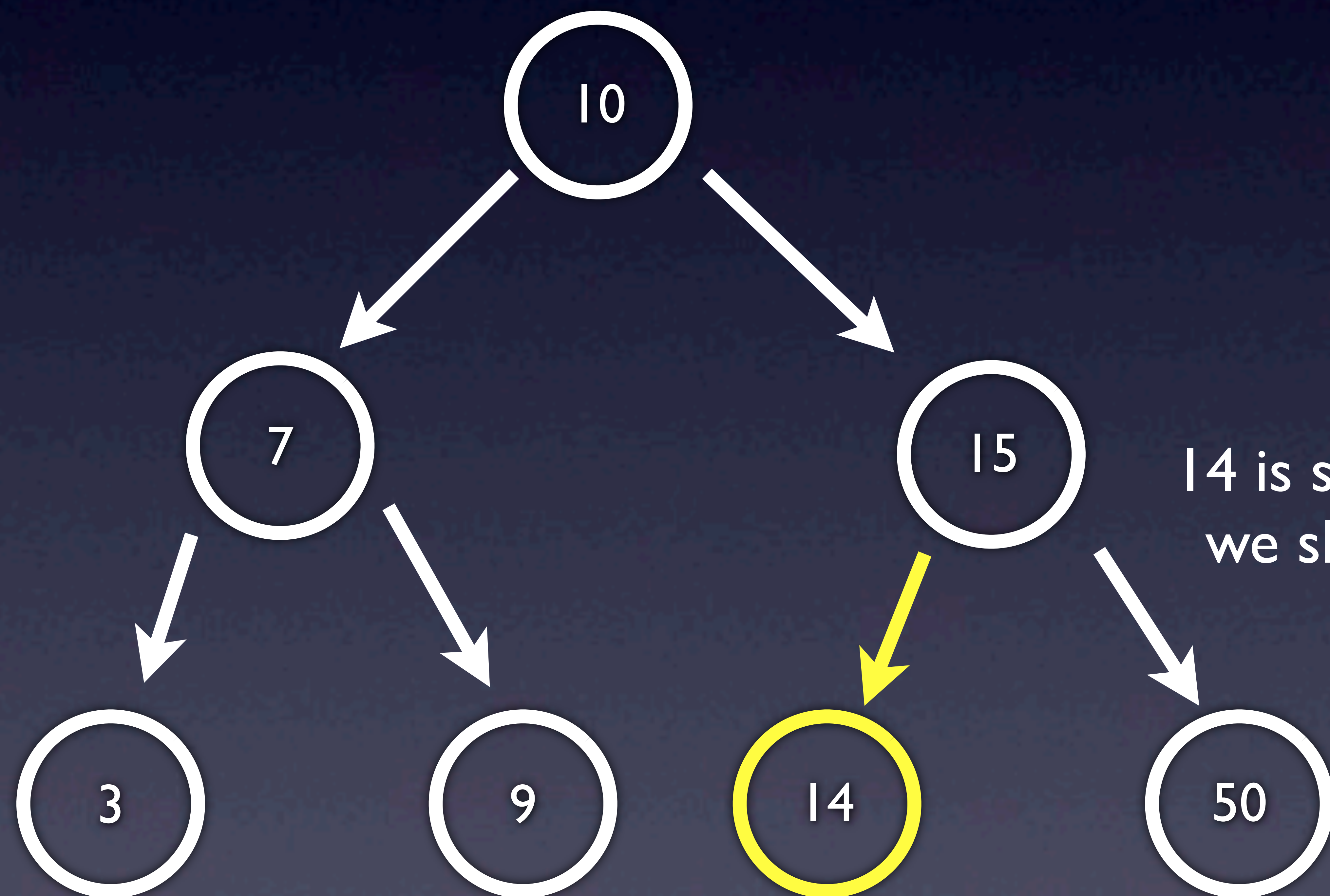
Search:

Find 14



Search:

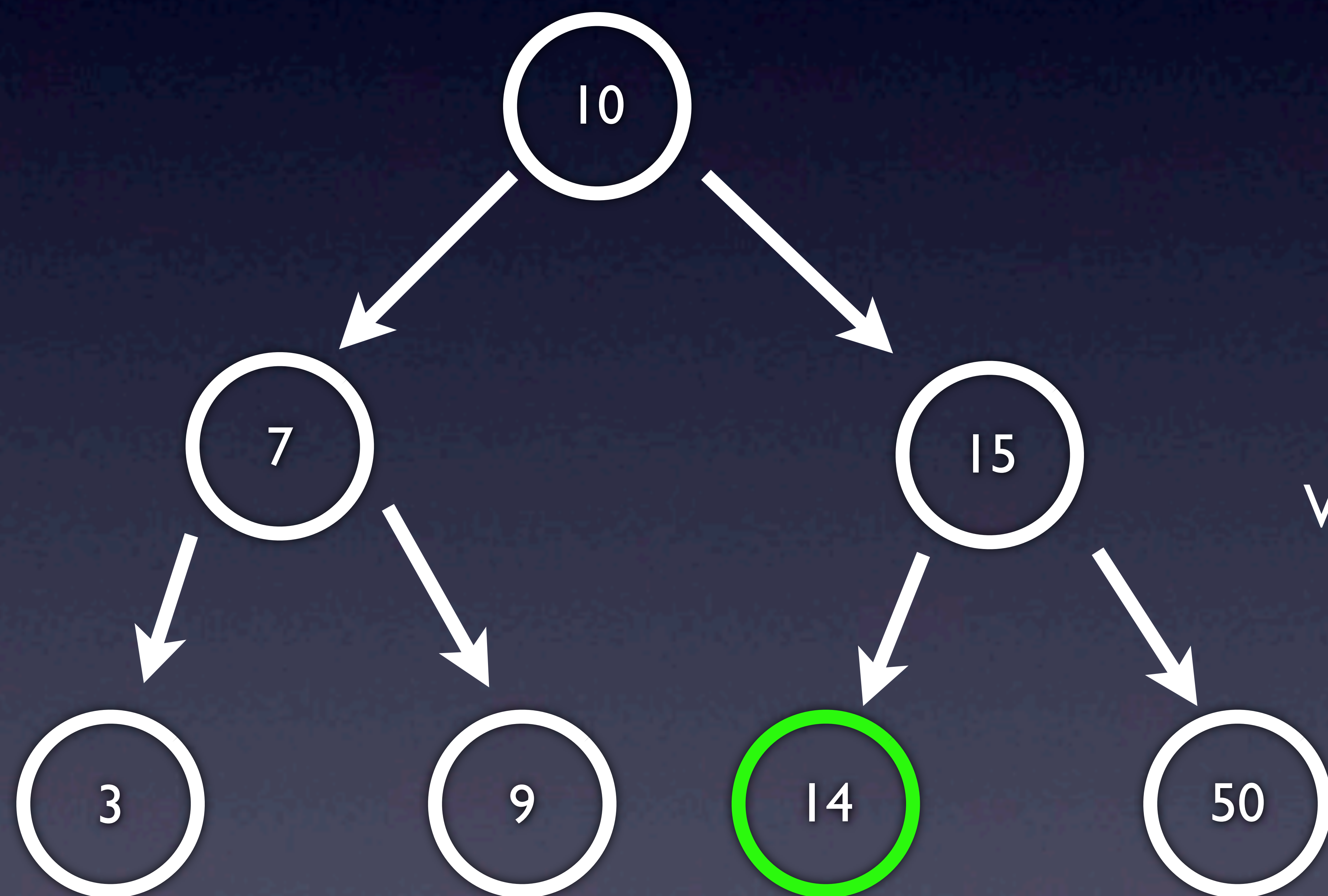
Find 14



14 is smaller than 15, so
we should try the left

Search:

Find 14



We found 14!

Search:

Implementation

```
bool search(int n, node* tree)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (n < tree->n)
    {
        return search(n, tree->left);
    }
    else if (n > tree->n)
    {
        return search(n, tree->right);
    }
    else
    {
        return true;
    }
}
```

BSTs

- Things we could ask you to do:
  - Write `insert`
  - Write an iterative version
  - Compare runtimes/explain when you would want to use a BST over a hashtable, for instance.