

# Data Structures

Lucas Freitas



# Data Structures

- Linked lists
- Stacks
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Data Structures

- **Understand** the conceptual description of each, and why you would use each

# Data Structures

- **Understand** the conceptual description of each, and why you would use each
- **Study and implement** in C the most common operations in each

# Data Structures

- **Understand** the conceptual description of each, and why you would use each
- **Study and implement** in C the most common operations in each
- **Review** pointers and structs

# Data Structures

- Linked lists
- Stacks
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Linked lists

- Insert or remove elements in  $O(1)$  if the linked list is unsorted
- Flexible length
- Remember to `malloc/free` nodes!

# Linked lists

```
typedef struct node  
{
```

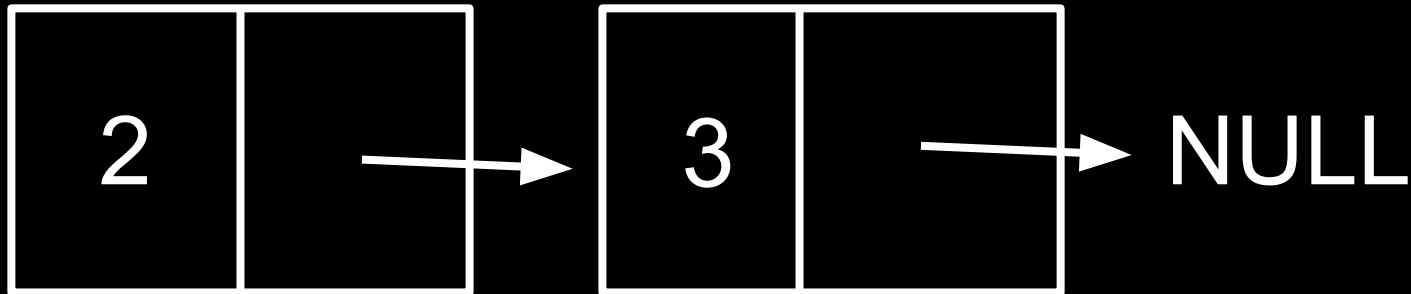
```
    int n;
```

```
    struct node* next;
```

```
}
```

```
node;
```

← or another type you  
want to store



# Insert node

```
void insert(int value)
{

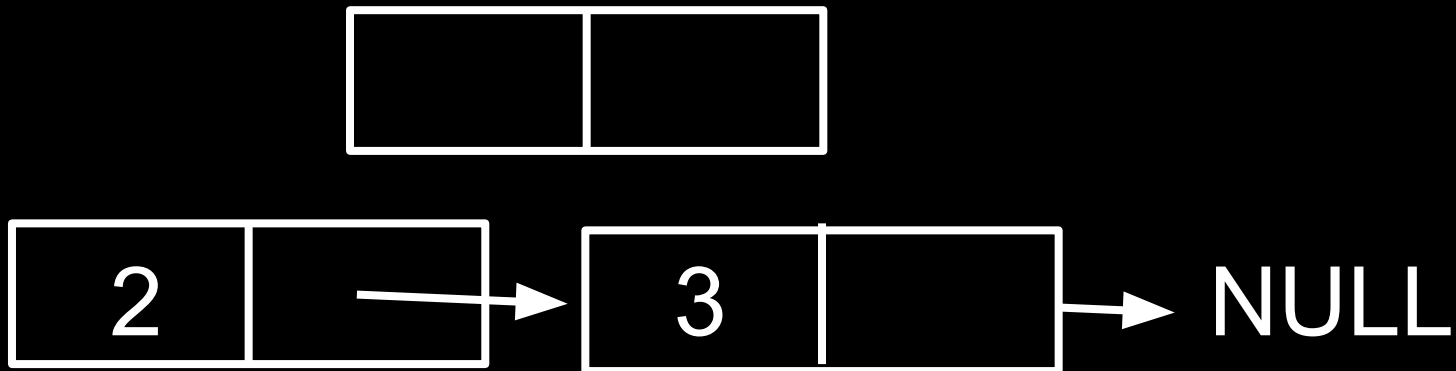
}
```



# Insert node

```
void insert(int value)
{
    node* newnode = malloc(sizeof(node));

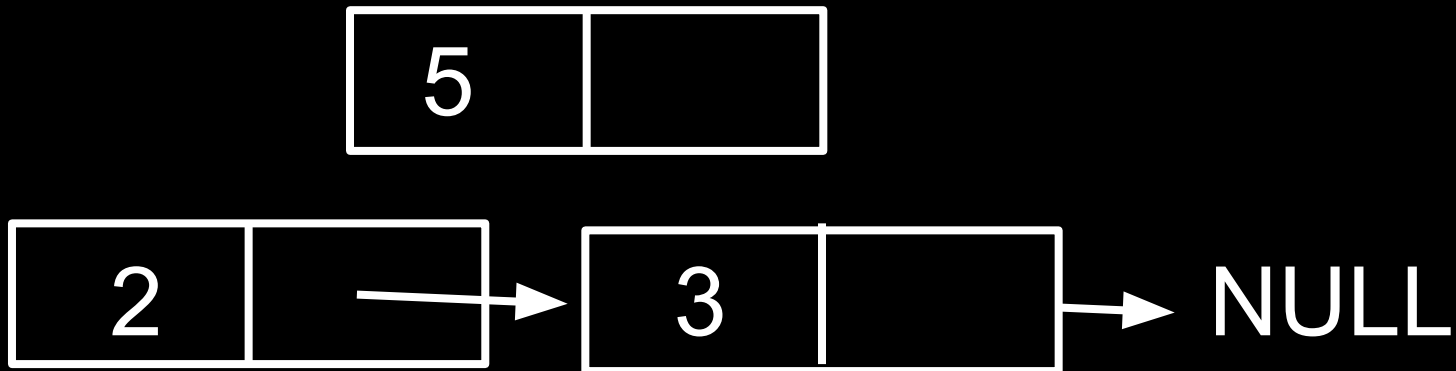
}
```



# Insert node

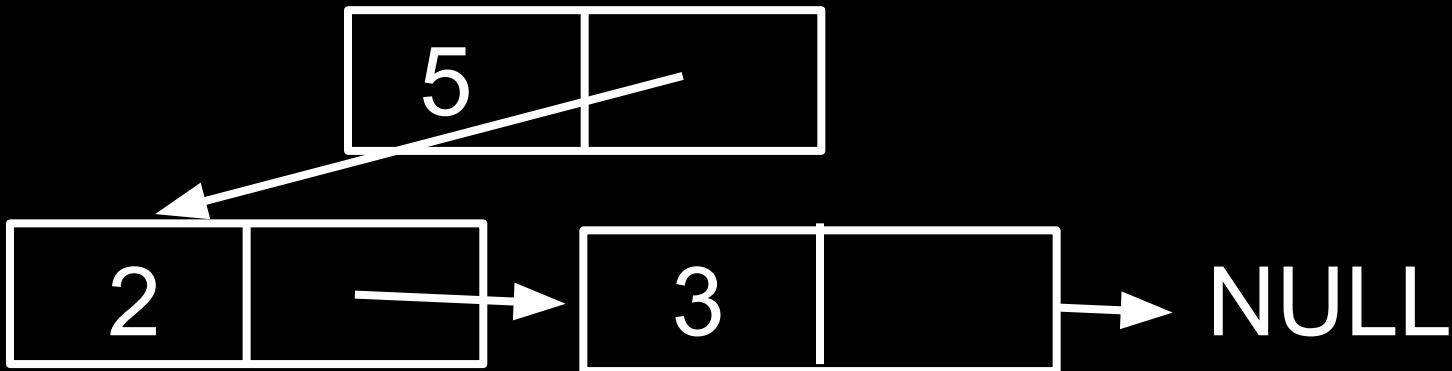
```
void insert(int value)
{
    node* newnode = malloc(sizeof(node));
    newnode->n = value;

}
```



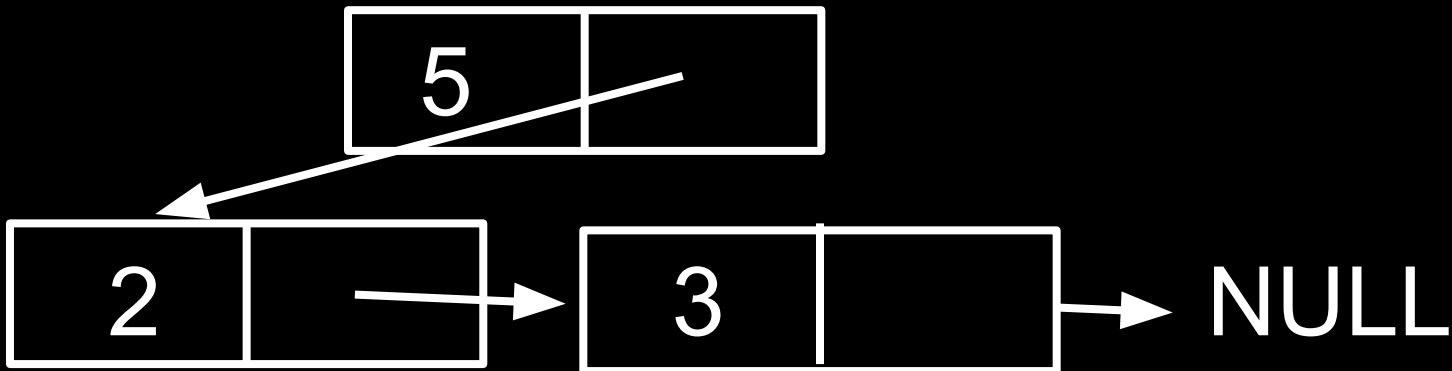
# Insert node

```
void insert(int value)
{
    node* newnode = malloc(sizeof(node));
    newnode->n = value;
    newnode->next = head;
}
```



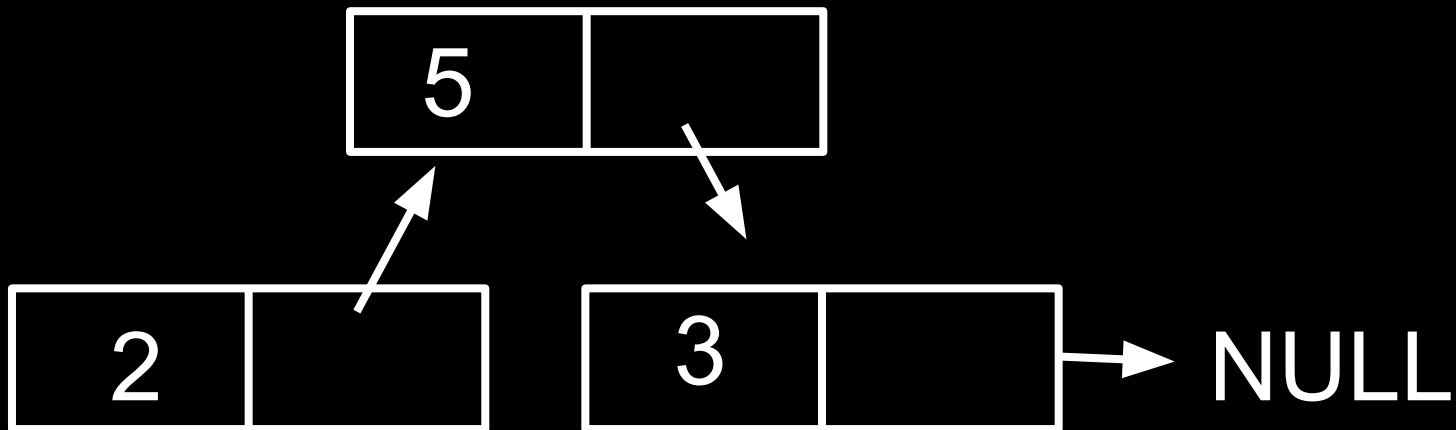
# Insert node

```
void insert(int value)
{
    node* newnode = malloc(sizeof(node));
    newnode->n = value;
    newnode->next = head;
    head = newnode;
}
```



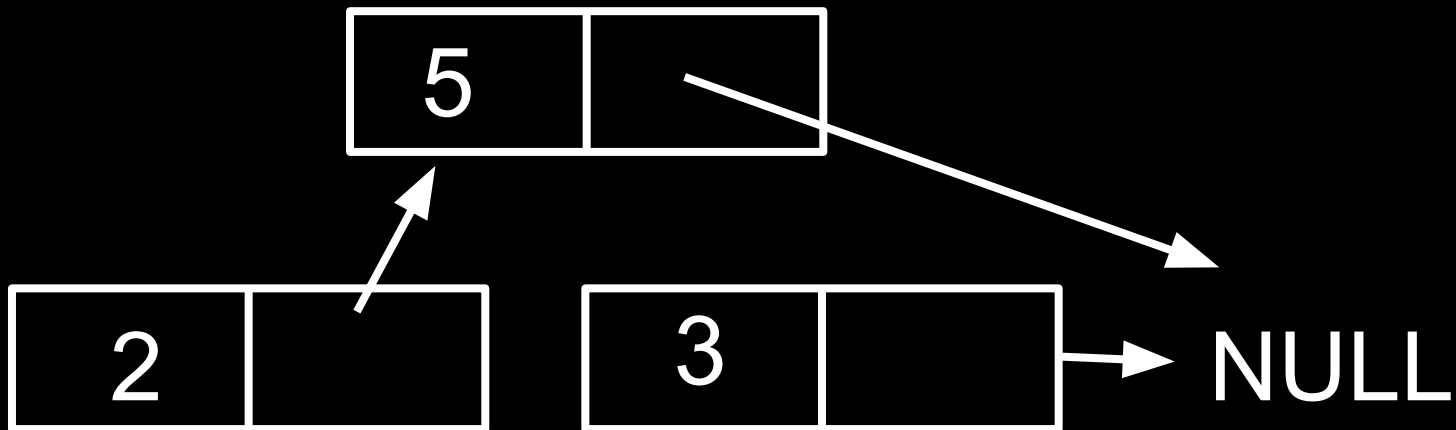
# Delete node

Implement at home for practice!



# Delete node

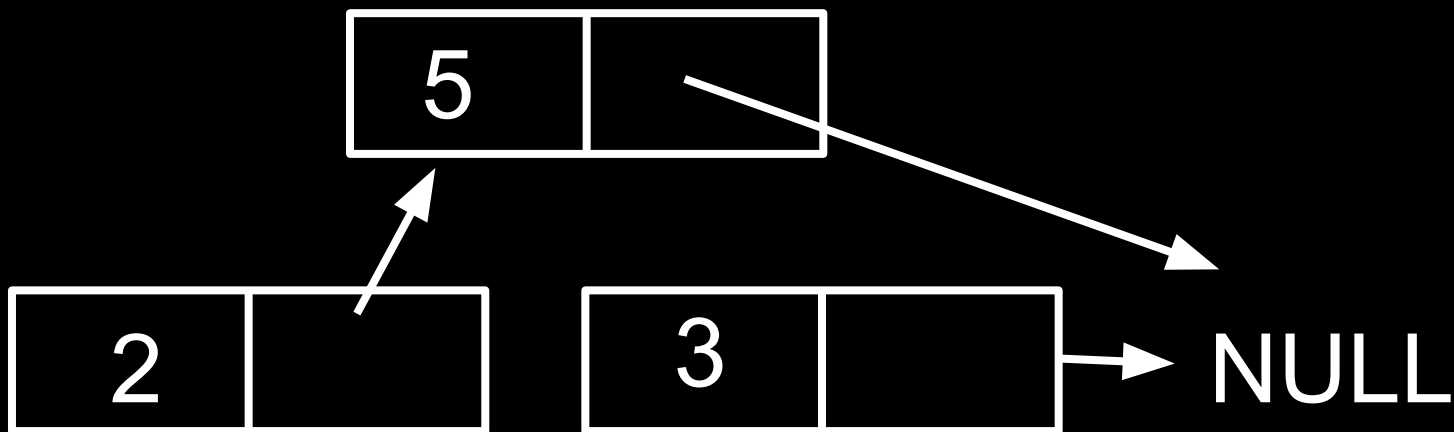
Implement at home for practice!



# Delete node

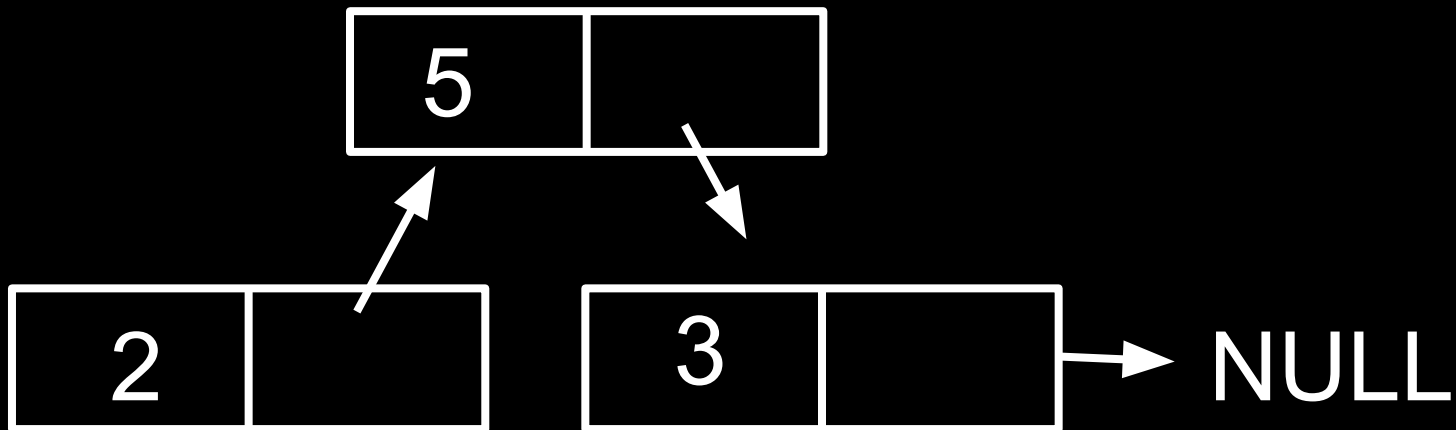
Implement at home for practice!

Memory leak!



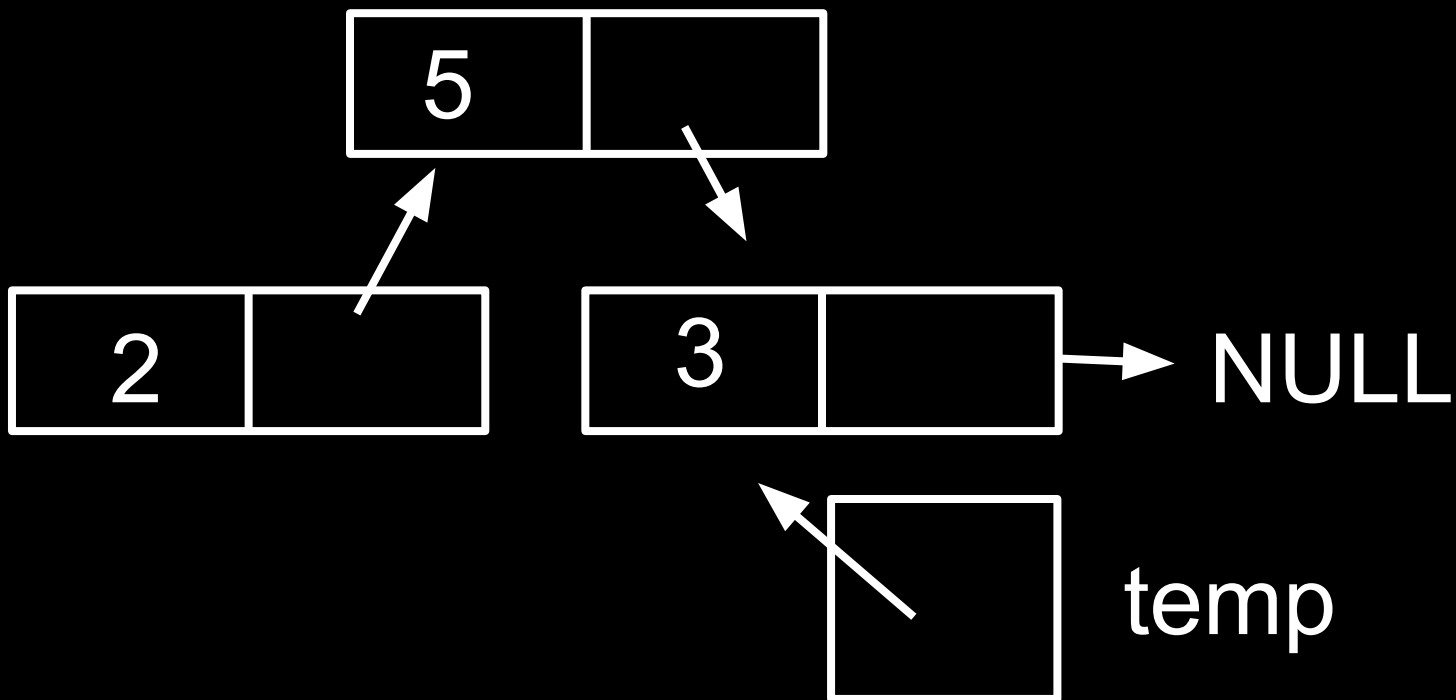
# Delete node

Implement at home for practice!



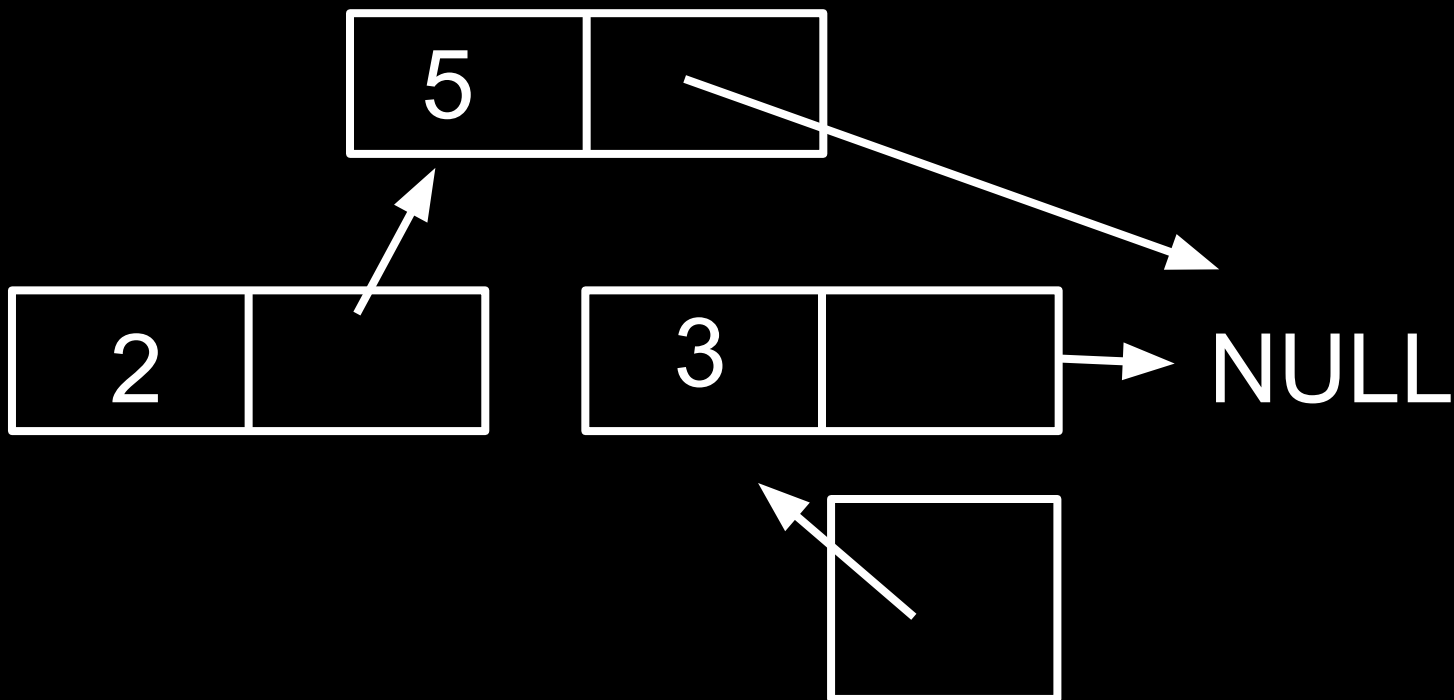
# Delete node

Implement at home for practice!



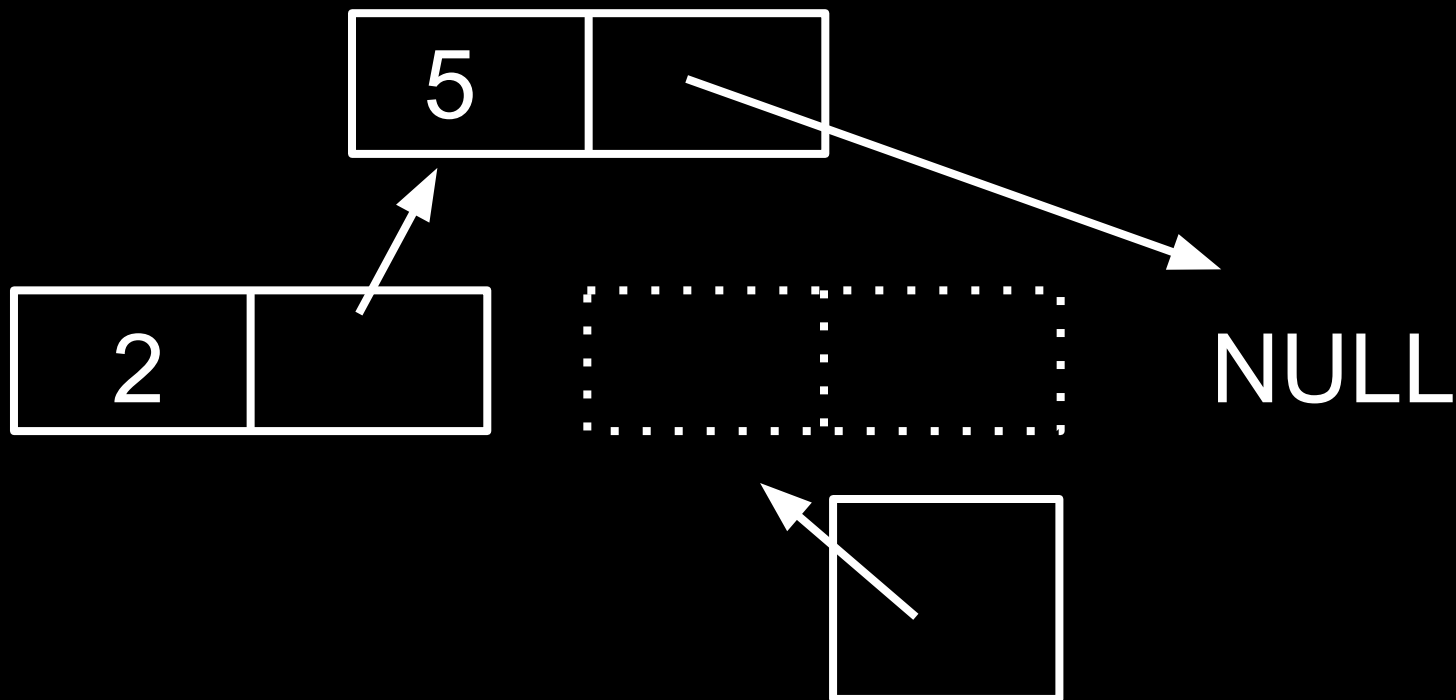
# Delete node

Implement at home for practice!



# Delete node

Implement at home for practice!



# Data Structures

- Linked lists
- **Stacks**
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Stacks

## Last In, First Out (LIFO)

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

# Push (array implementation)

```
bool push(int num)
{
    if (s.size >= CAPACITY)
        return false;

}
```



# Push (array implementation)

```
bool push(int num)
{
    if (s.size >= CAPACITY)
        return false;
    s.numbers[s.size] = num;
}
```



# Push (array implementation)

```
bool push(int num)
{
    if (s.size >= CAPACITY)
        return false;
    s.numbers[s.size] = num;
    s.size++;
}
```



# Pop (array implementation)

```
int pop(void)
{
    if (s.size <= 0)
        return -1;

}
```



# Pop (array implementation)

```
int pop(void)
{
    if (s.size <= 0)
        return -1;
    s.size--;
}
```



# Pop (array implementation)

```
int pop(void)
{
    if (s.size <= 0)
        return -1;
    s.size--;
    return s.numbers[s.size];
}
```

|   |   |   |   |  |  |  |  |  |
|---|---|---|---|--|--|--|--|--|
| 3 | 7 | 1 | 2 |  |  |  |  |  |
|---|---|---|---|--|--|--|--|--|

# Data Structures

- Linked lists
- Stacks
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Queues

## First In, First Out (FIFO)

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

# Enqueue (array implementation)

```
bool enqueue(int num)
{
    if (q.size >= CAPACITY)
        return false;

}
```



# Enqueue (array implementation)

```
bool enqueue(int num)
{
    if (q.size >= CAPACITY)
        return false;
    q.numbers[q.size] = num;
}
```



# Enqueue (array implementation)

```
bool enqueue(int num)
{
    if (q.size >= CAPACITY)
        return false;
    q.numbers[q.size] = num;
    q.size++;
}
```



# Enqueue (array implementation)

```
bool enqueue(int num)
{
    if (q.size >= CAPACITY)
        return false;
    q.numbers[q.size] = num;
    q.size++;
}
```



# Why is that wrong?

```
enqueue (5) ;  
enqueue (7) ;  
enqueue (1) ;  
enqueue (4) ;  
enqueue (6) ;  
dequeue () ;  
enqueue (1) ;
```



# Why is that wrong?

```
enqueue (5) ;
```

```
enqueue (7) ;
```

```
enqueue (1) ;
```

```
enqueue (4) ;
```

```
enqueue (6) ;
```

```
dequeue () ;
```

```
enqueue (1) ;
```



# Why is that wrong?

enqueue (5) ;

enqueue (7) ;

enqueue (1) ;

enqueue (4) ;

enqueue (6) ;

dequeue () ;

enqueue (1) ;



# Why is that wrong?

enqueue (5) ;

enqueue (7) ;

enqueue (1) ;

enqueue (4) ;

enqueue (6) ;

dequeue () ;

enqueue (1) ;



# Why is that wrong?

enqueue (5) ;

enqueue (7) ;

enqueue (1) ;

enqueue (4) ;

enqueue (6) ;

dequeue () ;

enqueue (1) ;

|   |   |   |   |   |   |   |   |  |
|---|---|---|---|---|---|---|---|--|
| 3 | 7 | 1 | 2 | 5 | 7 | 1 | 4 |  |
|---|---|---|---|---|---|---|---|--|

# Why is that wrong?

enqueue (5) ;

enqueue (7) ;

enqueue (1) ;

enqueue (4) ;

enqueue (6) ;

dequeue () ;

enqueue (1) ;

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 3 | 7 | 1 | 2 | 5 | 7 | 1 | 4 | 6 |
|---|---|---|---|---|---|---|---|---|

# Why is that wrong?

enqueue (5) ;

enqueue (7) ;

enqueue (1) ;

enqueue (4) ;

enqueue (6) ;

dequeue ( ) ;

enqueue (1) ;

|  |   |   |   |   |   |   |   |   |
|--|---|---|---|---|---|---|---|---|
|  | 7 | 1 | 2 | 5 | 7 | 1 | 4 | 6 |
|--|---|---|---|---|---|---|---|---|

# Why is that wrong?

enqueue (5) ;

enqueue (7) ;

enqueue (1) ;

enqueue (4) ;

enqueue (6) ;

dequeue () ;

enqueue (1) ;

|  |   |   |   |   |   |   |   |   |
|--|---|---|---|---|---|---|---|---|
|  | 7 | 1 | 2 | 5 | 7 | 1 | 4 | 1 |
|--|---|---|---|---|---|---|---|---|

# Enqueue (array implementation)

```
bool enqueue(int num)
{
    if (q.size >= CAPACITY)
        return false;

}
```



# Enqueue (array implementation)

```
bool enqueue(int num)
{
    if (q.size >= CAPACITY)
        return false;
    q.numbers[(q.size + q.head) % CAPACITY] = num;
}
```



# Enqueue (array implementation)

```
bool enqueue(int num)
{
    if (q.size >= CAPACITY)
        return false;
    q.numbers[(q.size + q.head) % CAPACITY] = num;
    q.size++;
}
```



# Deque

Implement at home!



# Note on stacks and queues

- Can also be implemented using linked lists
- Practice the implementations at home!!!
- Be sure that you understand the differences between the two!

**Let's relax for 10 seconds with Pokemons!**







**Now back to data structures!**



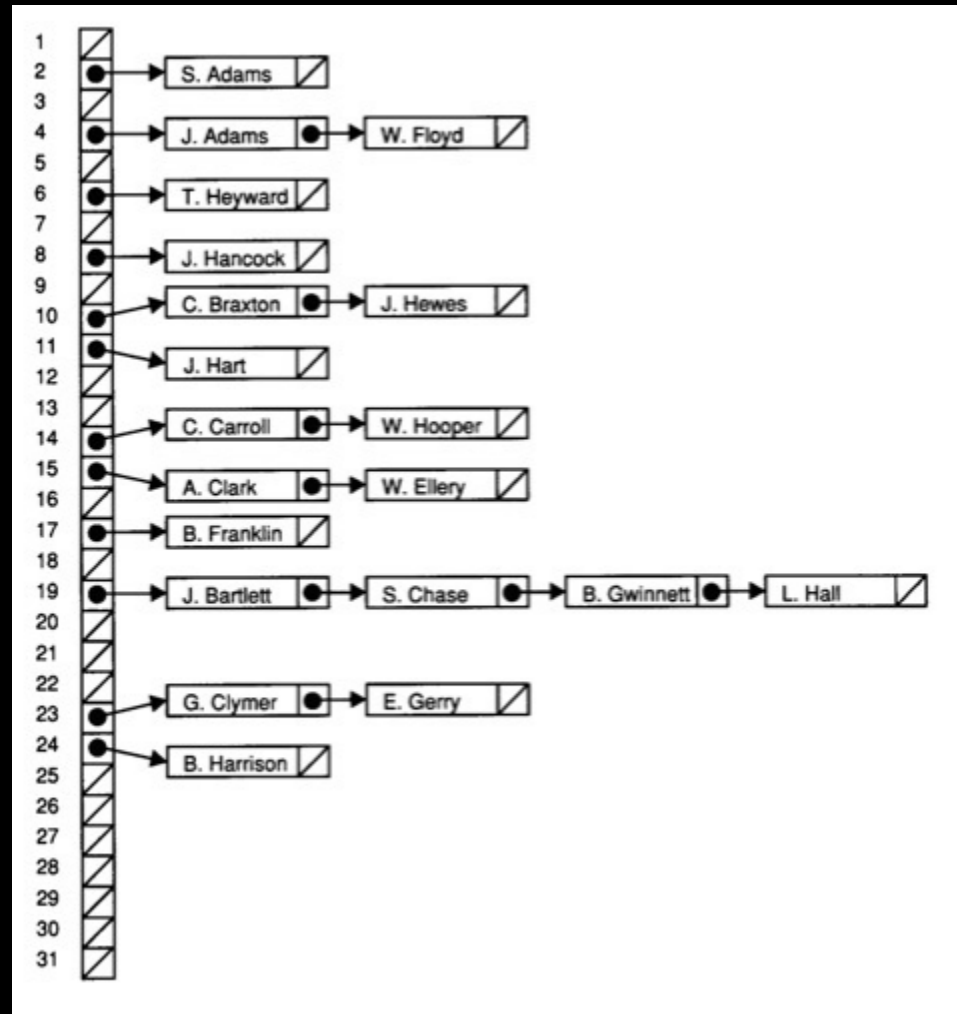
# Data Structures

- Linked lists
- Stacks
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Hash tables

- Array of linked lists
- Hash function
  - turns key (usually a `string`) into an index (`int`)
  - good ones are deterministic and well distributed
  - collisions (that's why we need the linked lists)
- Easy to check if a value is in the hash table
  - spellchecker

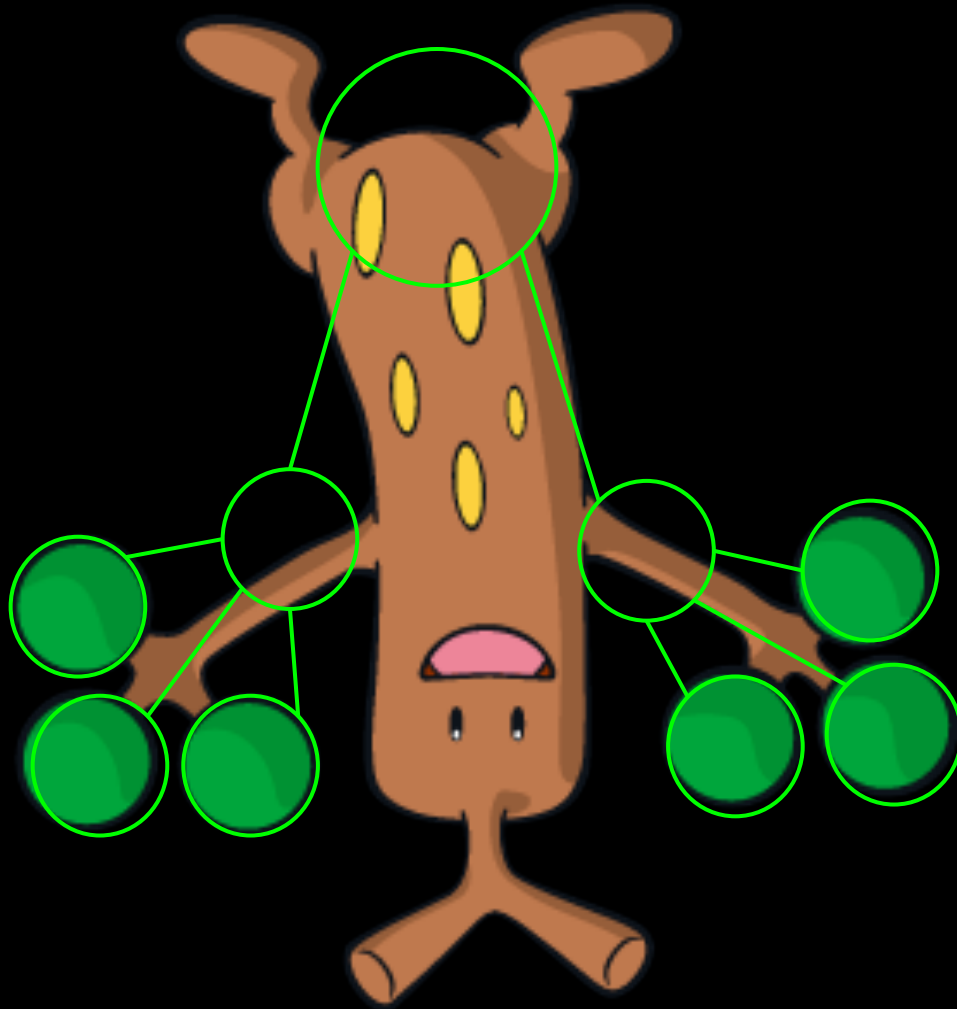
# Hash tables



# Data Structures

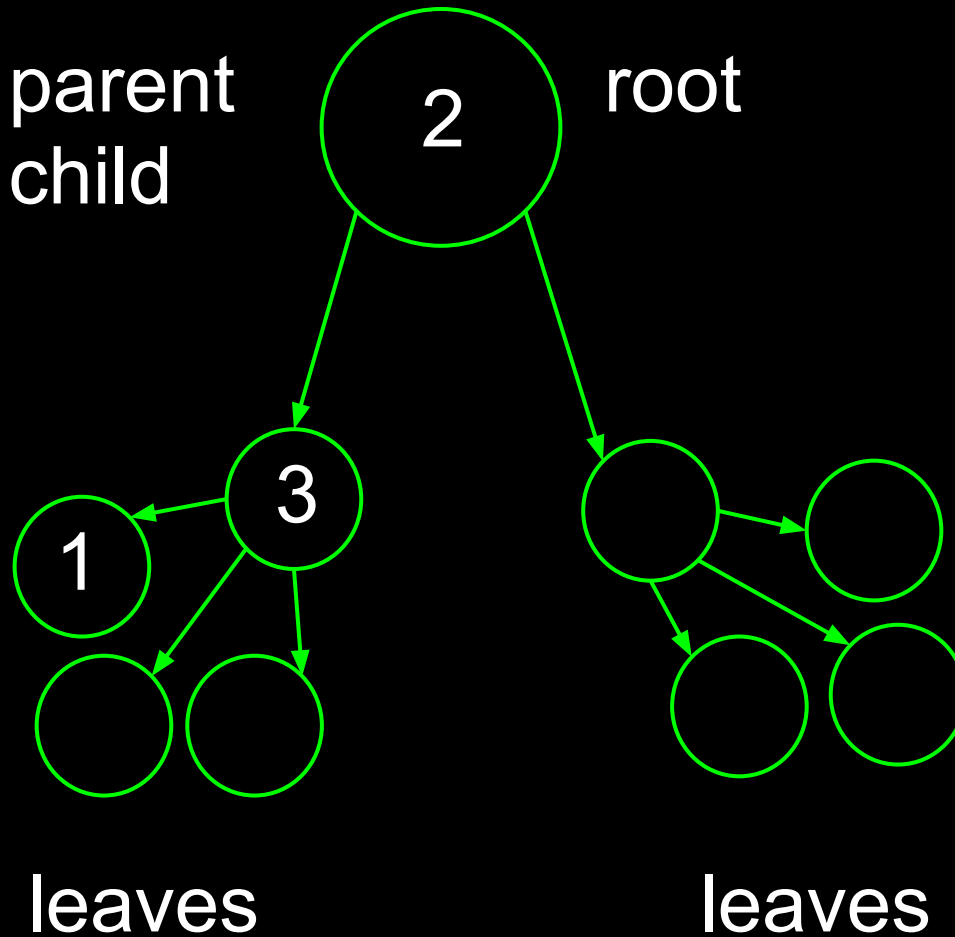
- Linked lists
- Stacks
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Trees



# Trees

- 3 is 1's parent
- 3 is 2's child

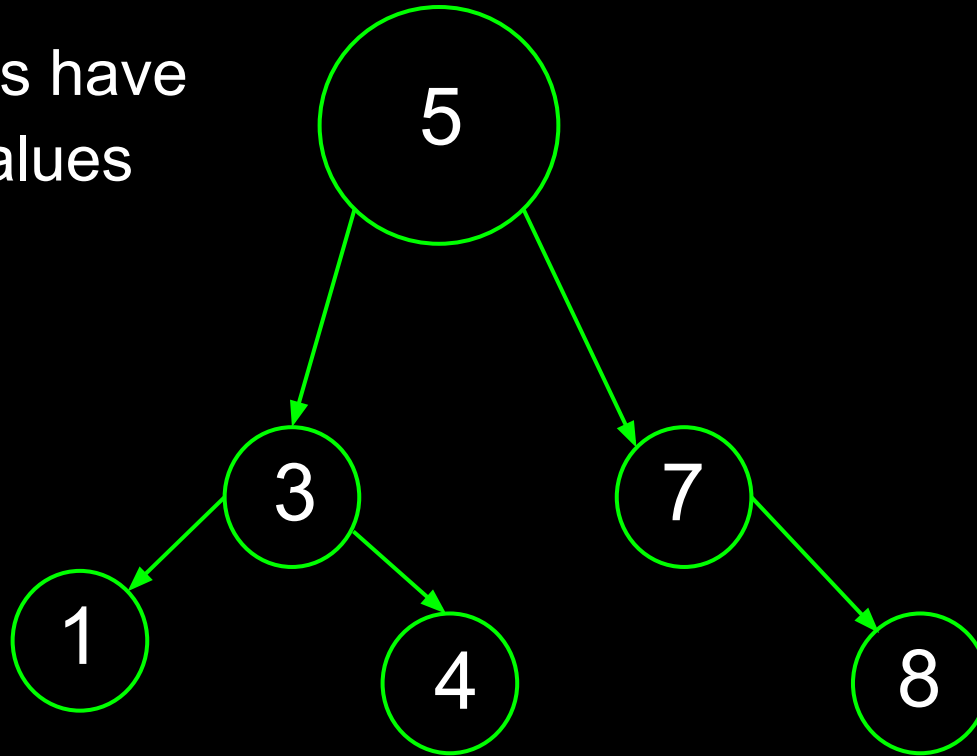


# Data Structures

- Linked lists
- Stacks
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Binary search tree

- Right nodes have "greater" values



# Binary search tree (BST)

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

# Find value in BST

```
bool find(int num, node* root)
{

}

}
```

# Find value in BST

```
bool find(int num, node* root)
{
    if (root == NULL)
        return false;

}
```

# Find value in BST

```
bool find(int num, node* root)
{
    if (root == NULL)
        return false;
    if (num > root->n)
        return find(num, root->right);

}
```

# Find value in BST

```
bool find(int num, node* root)
{
    if (root == NULL)
        return false;
    if (num > root->n)
        return find(num, root->right);
    if (num < root->n)
        return find(num, root->left);
}
```

# Find value in BST

```
bool find(int num, node* root)
{
    if (root == NULL)
        return false;
    if (num > root->n)
        return find(num, root->right);
    if (num < root->n)
        return find(num, root->left);
    return true;
}
```

# Data Structures

- Linked lists
- Stacks
- Queues
- Hash tables
- Trees
- Binary search trees (BST)
- Tries

# Tries (everyone's favorite)

- Tree of arrays
- Fast to lookup values
- Uses a lot of memory
- **Easy to filter words**

# Tries

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

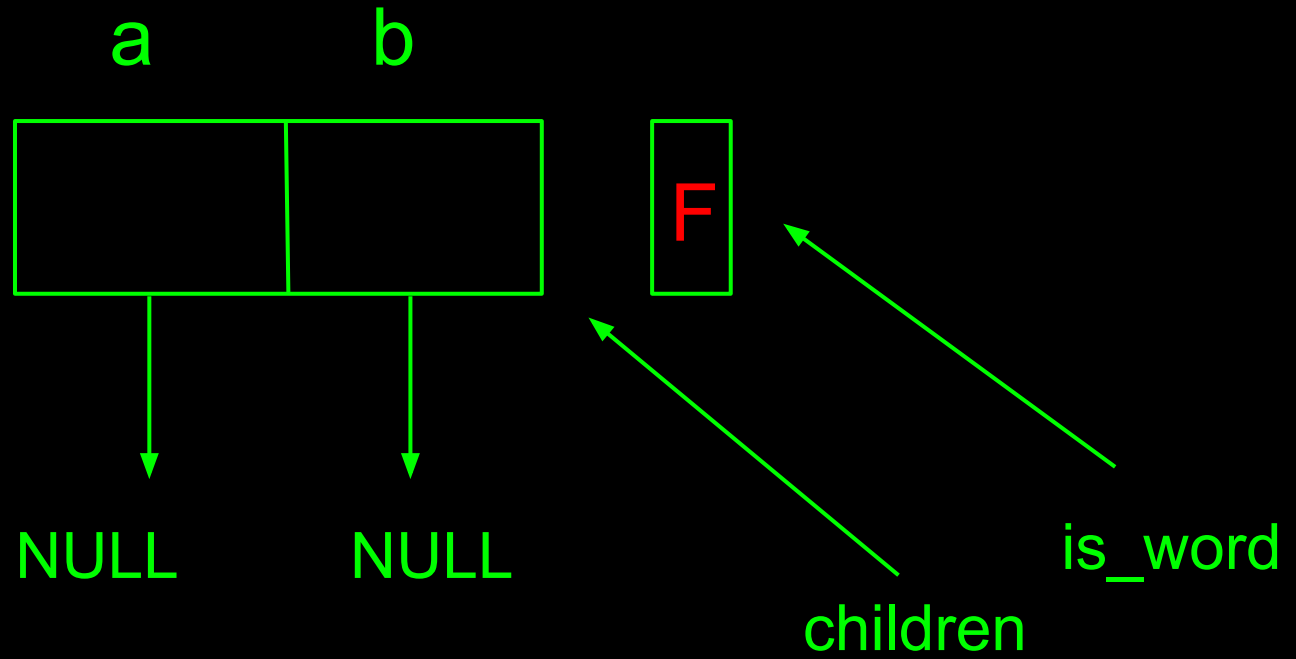
# Tries (for our spellcheck)

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

# Tries (for this review session)

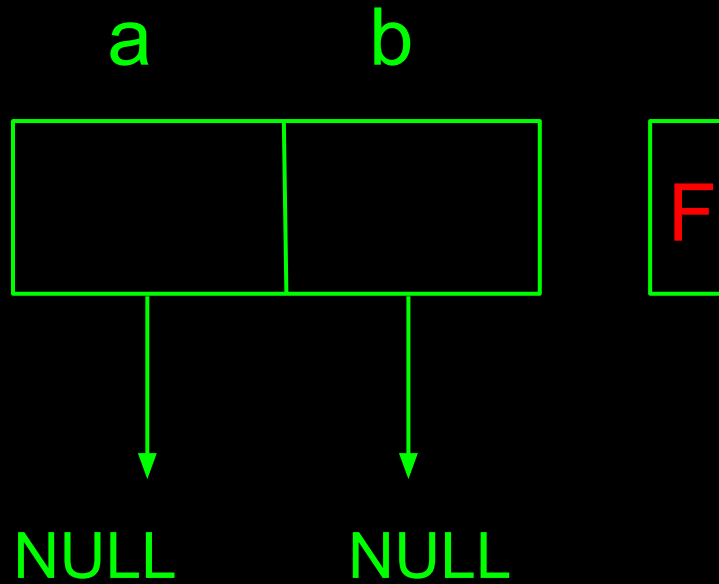
```
typedef struct node
{
    bool is_word;
    struct node* children[2];
}
node;
```

# Tries



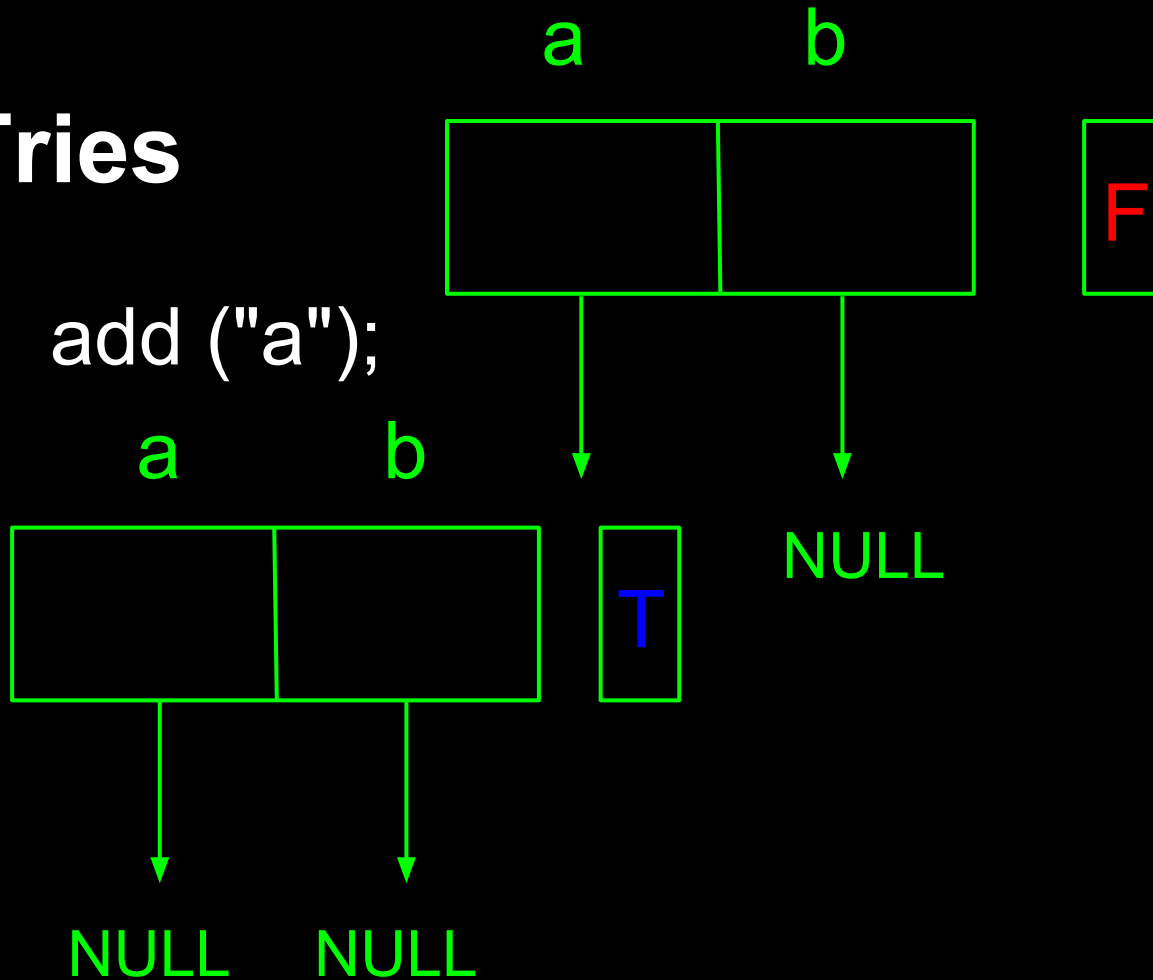
# Tries

- add ("a");



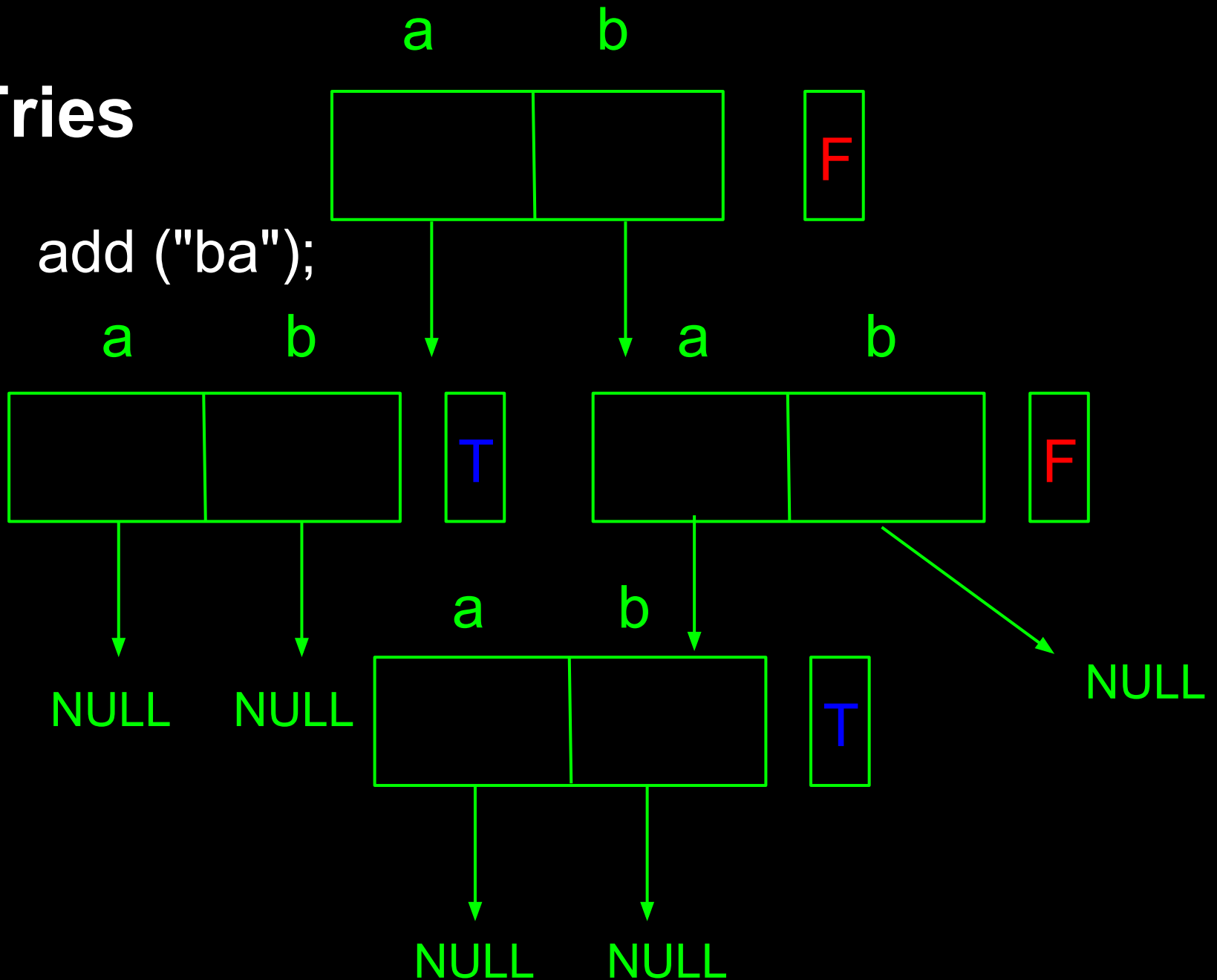
# Tries

- add ("a");



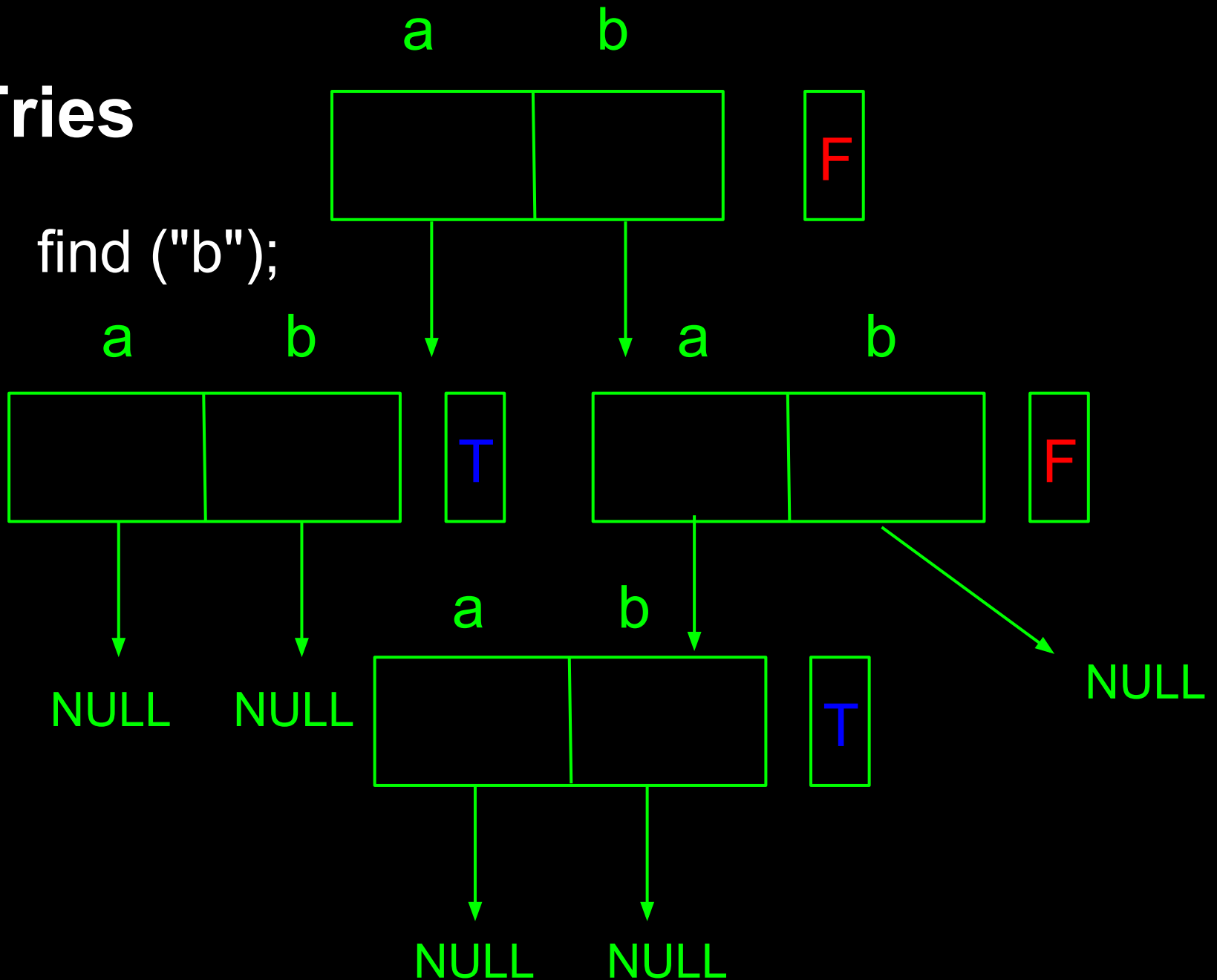
# Tries

- add ("ba");



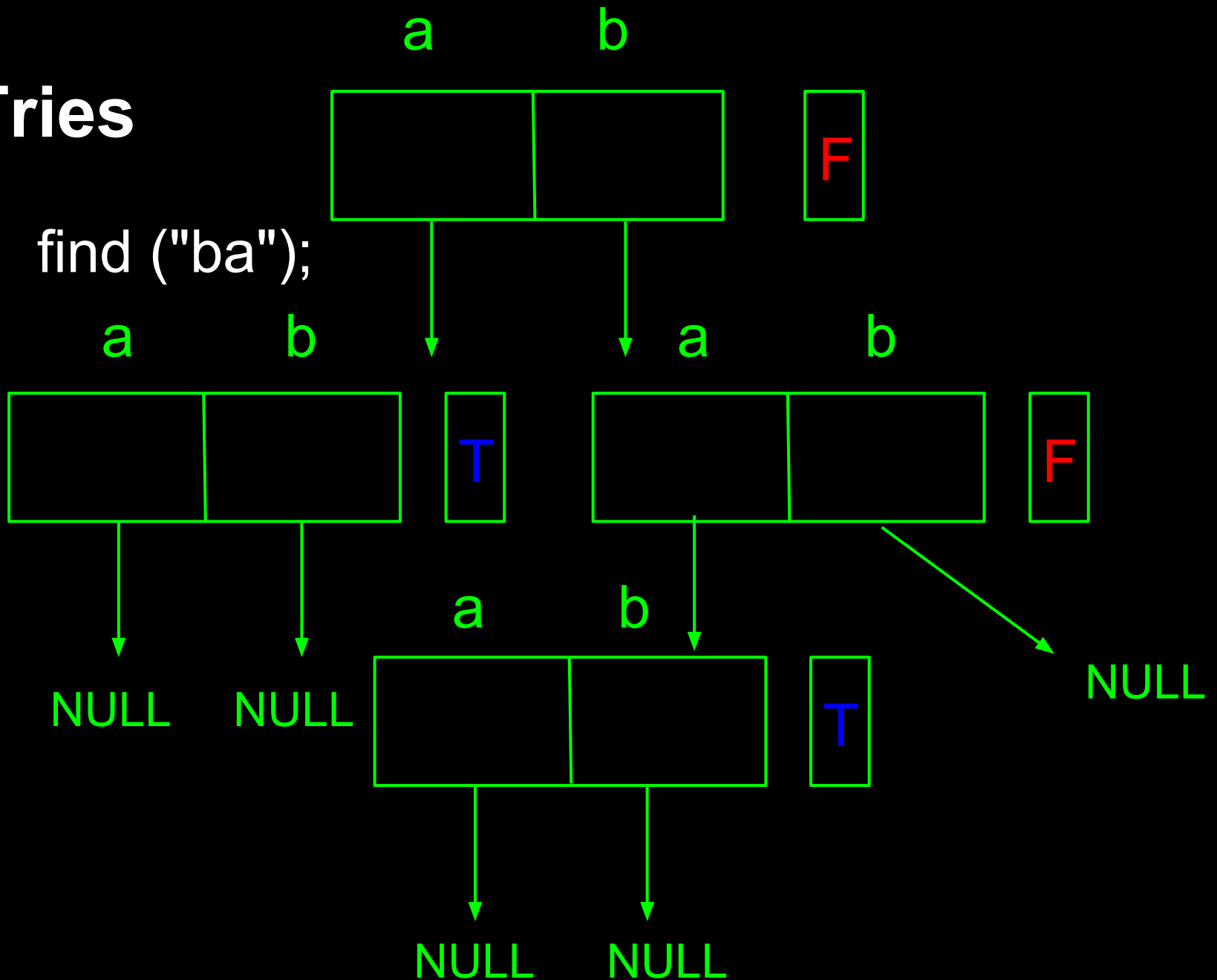
# Tries

- find ("b");



# Tries

- `find ("ba");`



# Huffman Coding!!!!

- Save memory
- Frequent characters shouldn't take as much memory as rare ones

# Huffman tree

```
typedef struct node
{
    char symbol;
    int frequency;
    struct node* left;
    struct node* right;
}
node;
```

# Building huffman tree

1. Pick two trees/nodes in forest with lowest frequencies (use lowest ASCII value if there is a tie);
2. Turn them into a parent tree (combined frequencies) and replace the two children with the parent in the forest;
3. Repeat 1 and 2 until there is only one tree in the forest.

# Example - Huffman tree for ZAMYLE

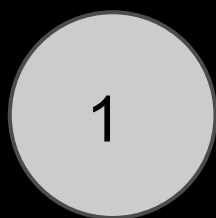
Z - frequency 1

A - frequency 2

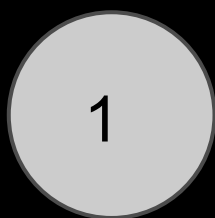
M - frequency 1

Y - frequency 1

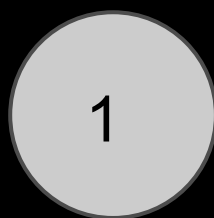
L - frequency 1



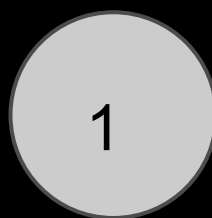
L



M



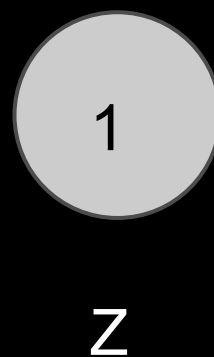
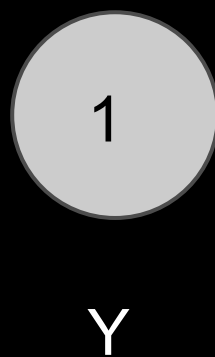
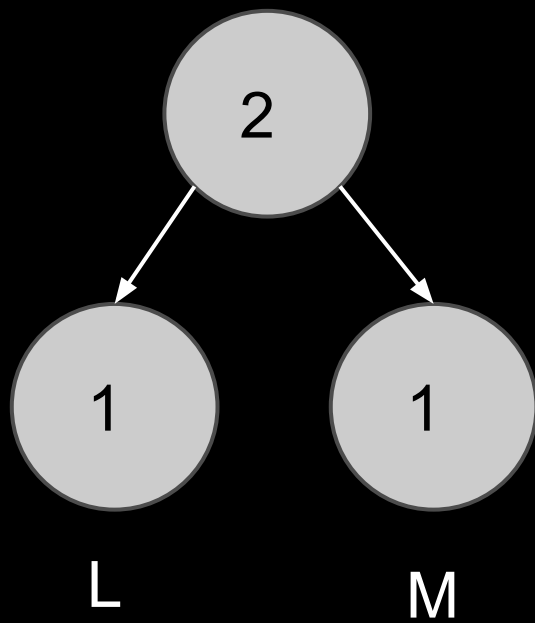
Y

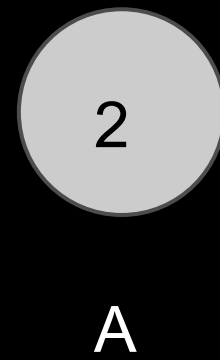
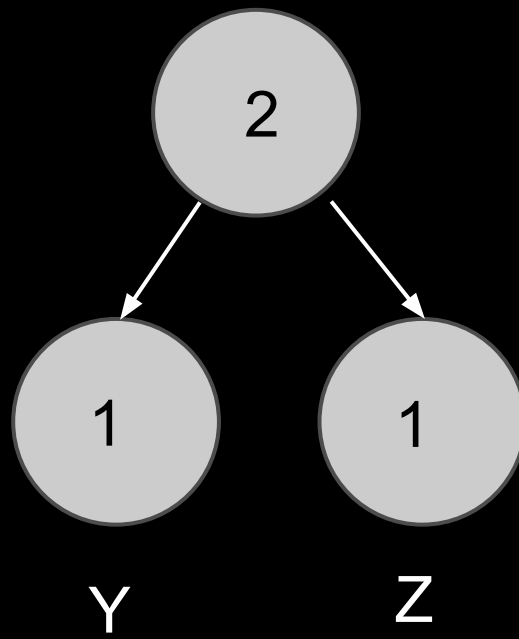
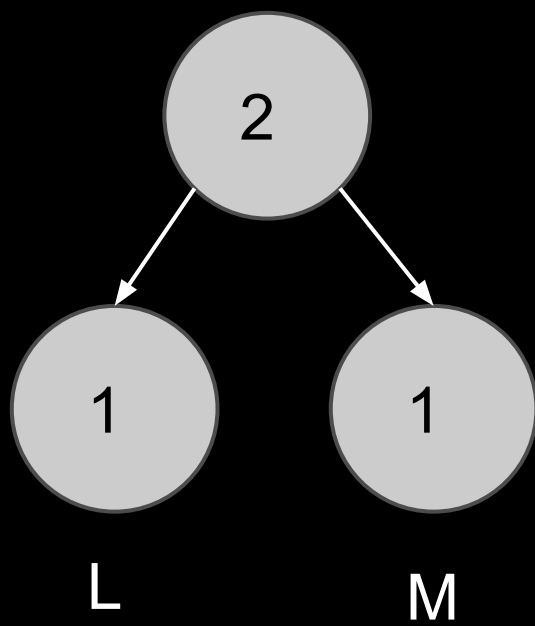


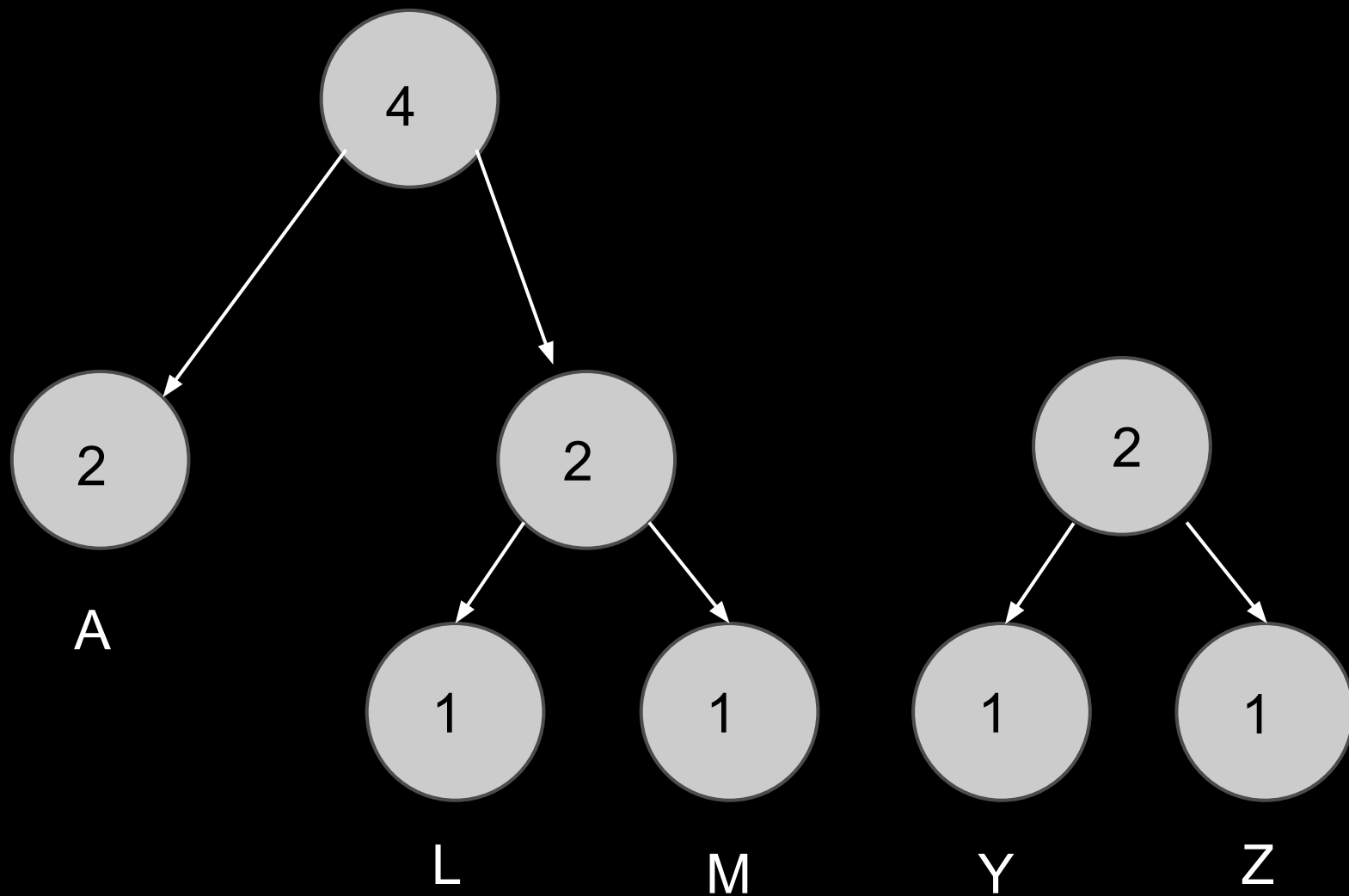
Z

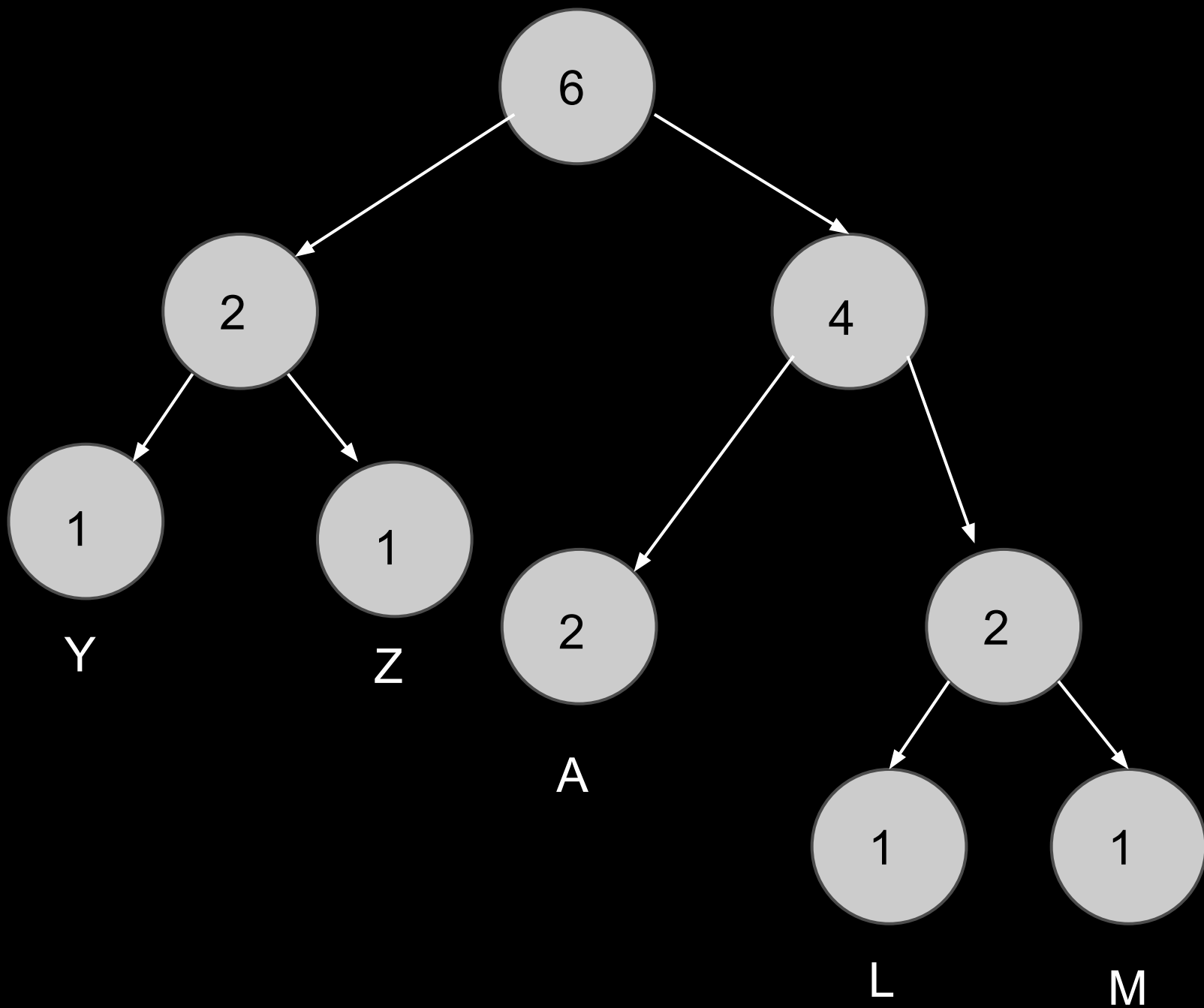


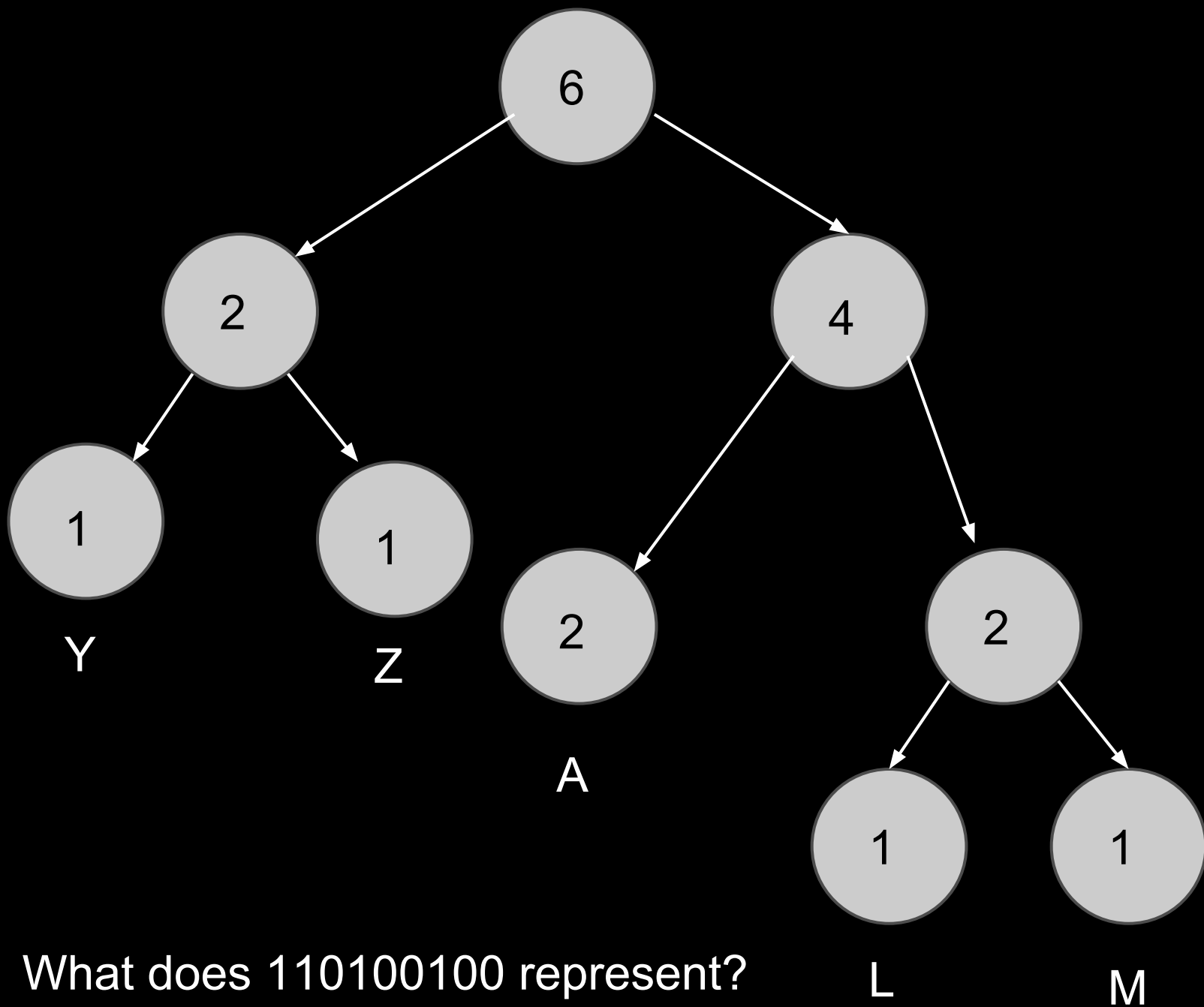
A











**Good luck!!!!**



Week 7

Rob Bowden

# Topics

- Bitwise operators
- Buffer overflow attack
- CS50 Library
- HTML
- HTTP
- CSS

# Bitwise Operators

- $\&$

- $>>$

- $|$

- $<<$

- $\wedge$

- $\sim$

# $\sim$ Bitwise Not

Reverses all bits

$$\sim 1101101 = 0010010$$

Frequently useful if we want “all bits but one”  
equal to one, as we’ll see shortly

# | Bitwise Or

$$\begin{array}{r} 1010 \\ | 1100 \\ = \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array} \quad \begin{array}{l} \\ \\ \end{array}$$

# | Bitwise Or

$$\begin{array}{r} 1010 \\ | 1100 \\ = 1110 \end{array}$$

# | Bitwise Or

'A' | 0x20 = ?

'a' | 0x20 = ?

# | Bitwise Or

`'A' | 0x20 = 'a'`

`'a' | 0x20 = 'a'`

# & Bitwise And

$$\begin{array}{r} 1010 \\ \& 1100 \\ = \quad ???? \end{array}$$

# & Bitwise And

$$\begin{array}{r} 1010 \\ \& 1100 \\ = 1000 \end{array}$$

# & Bitwise And

'A' & ~0x20 = ?

'a' & ~0x20 = ?

# & Bitwise And

`'A' & ~0x20 = 'A'`

`'a' & ~0x20 = 'A'`

$\wedge$  XOR

1010

$\wedge$  1100

= ????

$\wedge$  XOR

1010

$\wedge$  1100

= 0110

$\wedge$  XOR

$\text{'A'} \wedge 0x20 = ?$

$\text{'a'} \wedge 0x20 = ?$

$\wedge$  XOR

$\text{'A'} \wedge 0x20 = \text{'a'}$

$\text{'a'} \wedge 0x20 = \text{'A'}$

# << Left Shift

00001101

<< 3

= ??????????

# << Left Shift

00001101

<< 3

= 01101000

# << Left Shift

$$\begin{array}{r} 00001101 \\ \ll 3 \\ = 01101000 \end{array}$$

$$\begin{array}{r} 13 \\ \ll 3 \\ = 104 \end{array}$$

“ $x \ll y$ ” roughly means  $x * 2^y$

>> Right Shift

01101000

>> 3

= ??????????

>> Right Shift

01101000

>> 3

= 00001101

# << Left Shift

01101000  
>> 3  
= 00001101

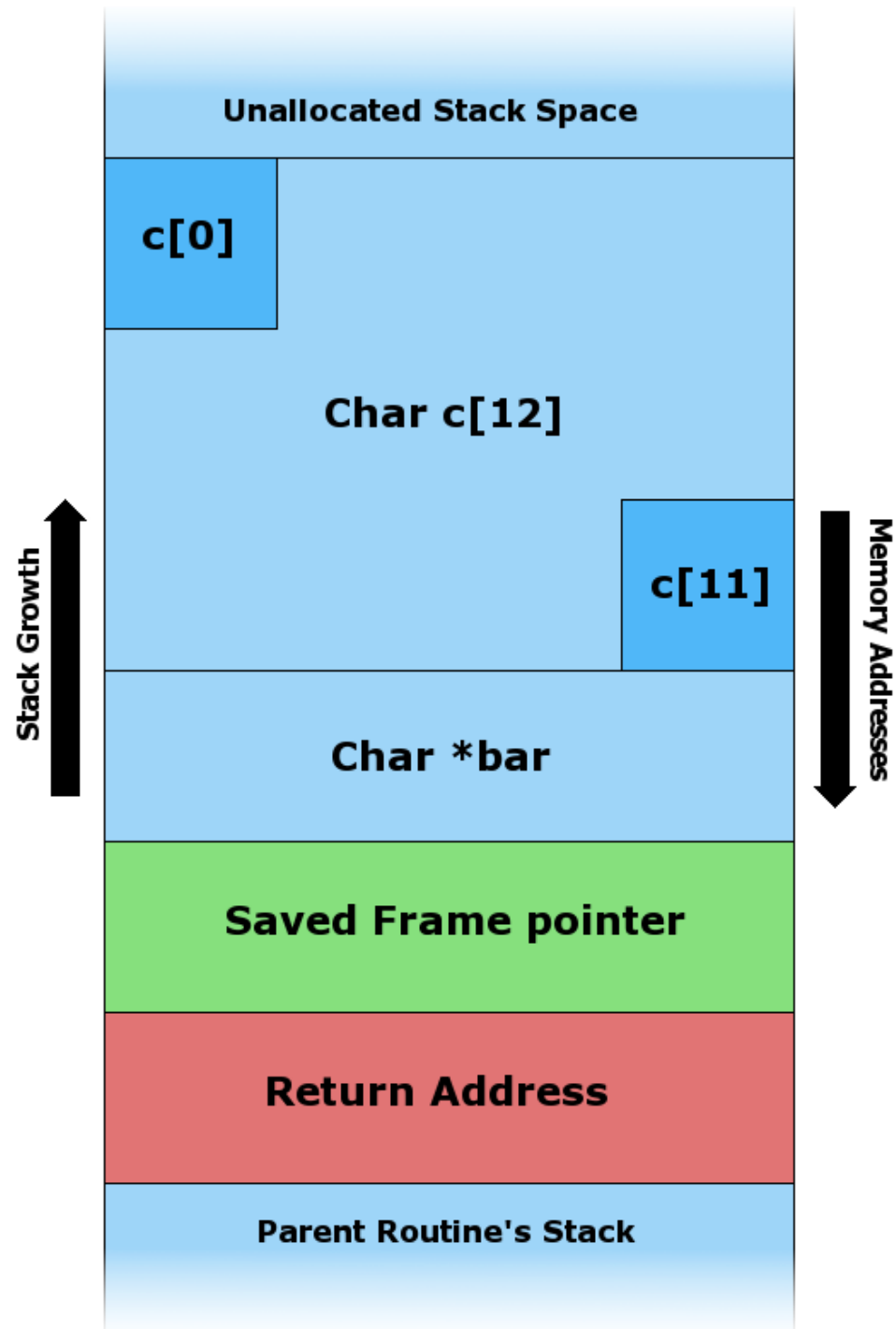
104  
>> 3  
= 13

“ $x \gg y$ ” roughly means  $x / 2^y$   
(What do I mean by “roughly”?)

# Buffer Overflow Attack

What was wrong with this function?

```
void foo(char* bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}
```



Address  
0x80C03508



Stack Growth  
↑



Memory Addresses  
↓

Little Endian  
0x80C03508



# CS50 Library

GetInt, GetString, etc.

```
typedef char* string;
```

# GetString, abridged

```
// ...
while ((c = fgetc(stdin)) != '\n' && c != EOF)
{
    // grow buffer if necessary
    if (n + 1 > capacity)
    {
        // ...
        buffer = realloc(buffer, capacity * 2);
    }

    // append current character to buffer
    buffer[n++] = c;
}
// ...
```

# GetInt

```
while (true)
{
    string line = GetString();
    if (line == NULL)
        return INT_MAX;

    int n; char c;
    if (sscanf(line, " %d %c", &n, &c) == 1)
    {
        free(line);
        return n;
    }
    else
    {
        free(line);
        printf("Retry: ");
    }
}
```

# HTML

HyperText Markup Language

Defines the *structure* and *semantics* of webpages  
(*not* the style)

# HTML

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <link href="styles.css" rel="stylesheet"/>
```

```
    <script src="scripts.js"></script>
```

```
    <title>hello, world</title>
```

```
  </head>
```

```
  <body>
```

```
    hello, world
```

```
    
```

```
  </body>
```

```
</html>
```

# HTML

```
<form action="http://www.google.com/search" method="get">
```

# HTTP

HyperText Transfer Protocol

The protocol of the World Wide Web!!

Transfers *hypertext*, i.e. HTML, but also images, stylesheets, and anything else

# HTTP Request

```
GET /search?q=quick+brown+fox HTTP/1.1  
Host: www.google.com
```

# HTTP Response

HTTP/1.1 200 OK

Followed by whatever I requested

# HTTP Status Codes

- 200 OK
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error

# TCP/IP

HTTP is built on top of TCP/IP

If HTTP is the language of the World Wide Web, IP is the language of the *Internet* (and more)!

# TCP/IP

Every computer on the internet is addressable through an IP address. This includes the servers that host your favorite websites.

DNS lets us remember “google.com” instead of something like 74.125.224.72

# TCP/IP

In addition to IP addresses, every computer has a number of *ports* that different applications can listen on so that the applications don't interfere with one another.

This is part of what the “TCP” part of TCP/IP provides us.

By default, HTTP uses port 80. Other services use other ports, such as SMTP using port 25 for email

# CSS

## Cascading Stylesheets

Used to *style* webpages (remember, HTML is not meant for styling).

# CSS

## Three places to put your styling:

1)

Inline

```
<body style="text-align: center;">
```

2) Between <style> tags

3) In a separate file, which is then “linked” into one or more HTML documents

```
<link href="styles.css" rel="stylesheet"/>
```

# CSS

```
body
{
    text-align: center;
}
#footer
{
    font-size: smaller;
    font-weight: bold;
    margin: 20px;
    text-align: left;
}
```

# PHP and SQL

Ali Nahm



# PHP: Hypertext Preprocessor

- Server-side scripting language
- Let's us develop the backend, or logical underpinnings, of our website

# <?php Syntax ?>

- all PHP code must start with <?php and end with ?> tags
- all variables start with \$
- you do NOT have to note the variable type in your declaration like in C!
  - when you're declaring
  - when you're referring

# Weakly Typed Variables

- Weakly typed means you can freely switch and compare variables between types

```
<?php  
  
$num_int = 1;  
$num_string = "1";  
$num_float = 1.0;  
  
?>
```

- Even though you don't specify the type, there are still variable types!!!

# To Equals or Not to Equals?

`==` checks across types

`===` strict equality, value AND type must match

```
<?php

$num_int = 1;
$num_string = "1";

if ($num_int == $num_string)
echo "Will be echoed";

if ($num_int === $num_string)
echo "Won't be echoed";
?>
```

# String Concatenation

Use the `.` operator!

```
<?php

$string1 = "CAT";
$string2 = "DOG";

$show_name = $string1 . " " . $string2;

// this will print "CAT DOG"
echo $show_name;

// this will print "CAT DOG" via string interpolation
echo ("{$string1} {$string2}");
?>
```



# Arrays

- regular arrays similar to C
- associative arrays

# Arrays

- regular arrays similar to C
- associative arrays

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
?>
```

indices

0

1

2

values

| 0 | 1 | 2 |
|---|---|---|
|   |   |   |

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
// like in C, changes at locations 1 and 2  
$number[0] = 6;
```

```
?>
```

indices

0

1

2

values

6

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
// like in C, changes at locations 1 and 2  
$number[0] = 6;  
$number[1] = 4;
```

```
?>
```

indices

values

| 0 | 1 | 2 |
|---|---|---|
| 6 | 4 |   |

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
// like in C, changes at locations 1 and 2  
$number[0] = 6;  
$number[1] = 4;
```

```
// like in C, access at specific locations 1 and 2  
// this will print "6"  
echo $number[0];
```

```
?>
```

**indices**

**values**

| 0 | 1 | 2 |
|---|---|---|
| 6 | 4 |   |

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
?>
```

indices

0

1

2

values

| 0 | 1 | 2 |
|---|---|---|
|   |   |   |

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
// NOT like in C, you can just append to the end of  
// an array! (pushes 1, then 2, then 3 into array)  
$number[] = 1;
```

```
?>
```

indices

values

| 0 | 1 | 2 |
|---|---|---|
| 1 |   |   |

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
// NOT like in C, you can just append to the end of  
// an array! (pushes 1, then 2, then 3 into array)  
$number[] = 1;  
$number[] = 2;
```

```
?>
```

indices

values

| 0 | 1 | 2 |
|---|---|---|
| 1 | 2 |   |

# Regular Arrays

```
<?php
```

```
// creating an empty array  
$number = [];
```

```
// NOT like in C, you can just append to the end of  
// an array! (pushes 1, then 2, then 3 into array)
```

```
$number[] = 1;
```

```
$number[] = 2;
```

```
$number[] = 3;
```

```
// this will print out "2"
```

```
echo $number[1];
```

```
?>
```

**indices**

**values**

| 0 | 1 | 2 |
|---|---|---|
| 1 | 2 | 3 |

# Arrays

- regular arrays similar to C
- associative arrays

# Associative Arrays

- An array that uses strings as “keys” for each location in an array (aka indices)

indices

X

X

X

values

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |

# Associative Arrays

```
<?php
```

```
    $tf = [];
```

```
    $tf["name"] = "Ali";
```

```
    $tf["calories eaten"] = 1000;
```

```
    $tf["likes"] = ["pig", "milk"];
```

```
?>
```

|         |      |                 |
|---------|------|-----------------|
| indices |      |                 |
| values  | 1000 | ["pig", "milk"] |

# Associative Arrays

```
<?php
```

```
$tf = [];
```

```
// alternatively
```

```
$tf = [
```

```
    "name" => "Ali",
```

```
    "course" => "EC10",
```

```
    "likes" => ["pig", "milk"]
```

```
];
```

```
?>
```

indices

values

1000

["pig", "milk"]

# Loops

We can do the same as we did in C...

```
<?php
```

```
$psets = [1, 2, 3, 4, 5, 6, 7];  
for ($i = 1; $i <= 7; $i++)  
    doProblemSet($i);
```

```
print("DID ALL THE PROBLEM SETS!");
```

```
?>
```

... or we can use foreach loops if we don't know the numerical indices!

```
<?php
```

```
$psets = [1, 2, 3, 4, 5, 6, 7];  
foreach ($psets as $pset_num)  
    doProblemSet($pset_num);
```

```
print("DID ALL THE PROBLEM SETS!");
```

```
?>
```

# Useful Functions

- ▶ `require(pathToFile)` statement includes PHP code from the specified file and evaluates it. Often used to make libraries, etc.
- ▶ `echo` does the same thing as `print`.
- ▶ `exit` stops the further execution of any code.
- ▶ `empty` checks if a variable is empty. These are considered empty:
  - ▶ `""` `0` `0.0` `"0"` `null` `false` `[]` uninitialized variable
- ▶ REALLY good idea to check out the functions you used in pset7!

# Global Scope

```
<?php

    function aFunction($i)
    {
        $i++;
        echo $i . "\n";
    }

for (→ $i = 0; $i < 3; $i++)
    echo $i;

aFunction($i);

echo $i;
?>
```

0

# Global Scope

```
<?php

    function aFunction($i)
    {
        $i++;
        echo $i . "\n";
    }

for (→ $i = 0; $i < 3; $i++)
    echo $i;

aFunction($i);

echo $i;
?>
```



0  
1

# Global Scope

```
<?php

    function aFunction($i)
    {
        $i++;
        echo $i . "\n";
    }

for (→ $i = 0; $i < 3; $i++)
    echo $i;

aFunction($i);

echo $i;
?>
```

```
0
1
2
```

# Global Scope

one exception: functions

```
<?php
```

```
    function aFunction($i)
    {
        $i++;
        echo $i . "\n";
    }
```

```
    for ($i = 0; $i < 3; $i++)
        echo $i;
```

When \$i is  
3, we exit  
the loop.



```
    aFunction($i);
```

```
    echo $i;
?>
```

This is **not**  
local to the  
loop.

```
0
1
2
4
```

# Global Scope

```
<?php
```

```
    function aFunction($i)
    {
        $i++;
        echo $i . "\n";
    }
```

This is local to the function.

```
for ($i = 0; $i < 3; $i++)
    echo $i . "\n";
```

```
aFunction($i);
```

```
→
echo $i;
?>
```

0  
1  
2  
4  
3

# PHP and HTML

- ▶ PHP is used to make web pages dynamic.
  - ▶ With just HTML we serve the same static page to all users.
  - ▶ PHP gives us the power to alter the page's HTML prior to loading, based on the users actions, who they are, logic we've written up, etc.

```
<?= "You are logged in as " . $name ?>
```

You are logged in as Joseph Ong. [About me.](#)

What does your TF do well?  
Funny guy, answers questions w

You are logged in as Tommy MacWilliam. [About me.](#)

What does your TF do well?  
can be funnier.

# Generating HTML

- TWO ways that work

```
<?php
for ($i = 0; $i < 5; $i++)
    print("<img src='{ $memes[$i] }' />");
?>
```

```
<?php for($i = 0; i < 5; i++): ?>
<img src='<?= memes[$i] ?>' />
<?php endfor; ?>
```

↓ ↓

```
<img src='mudkip.png' />
<img src='ditto.jpg' />
<img src='snorlax.gif' />
```



# Forms and Requests

- We can pass data from HTML forms to PHP files

- If we're using a form, then...

```
<form action="printName.php" method="get">
```

- action attribute tells us where to send the data
- method attribute tells us how to send (get or post) the data

# GET request

```
<form action="printName.php" method="get">
First Name: <input type="text" name="firstname"/><br/>
Last Name: <input type="text" name="lastname"/>
</form>
```

First name: david

Last name: malan

```
<?php
echo $_GET["firstname"];
echo $_GET["lastname"];
?>
```

**\$\_GET array**  
(indexed by name attribute)

cloud.cs50.net/~youjustlostthegame/printname.php?firstname=david&lastname=malan

davidmalan

sent in URL

# POST request

```
<form action="printName.php" method="post">
First Name: <input type="text" name="firstname"/><br/>
Last Name: <input type="text" name="lastname"/>
</form>
```

First name: david

Last name: malan

```
<?php
echo $_POST["firstname"];
echo $_POST["lastname"];
?>
```

**\$\_POST** array  
(also indexed by name attribute)

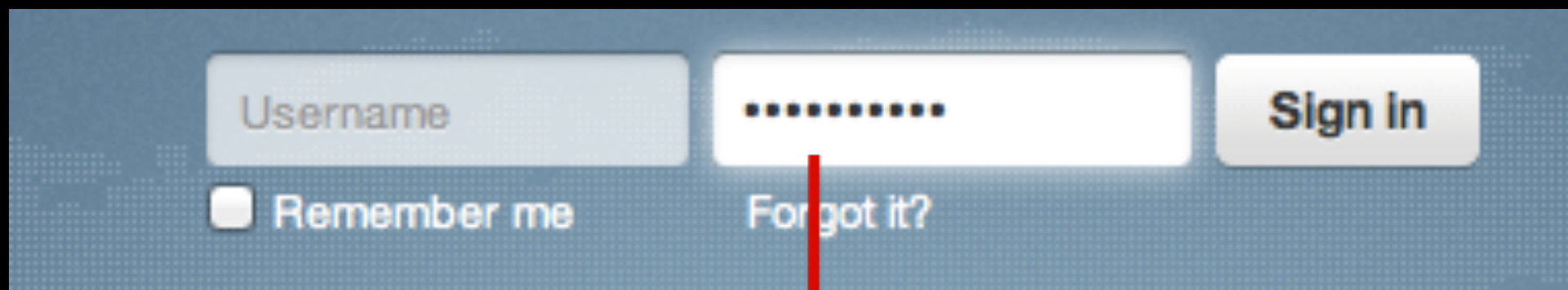
cloud.cs50.net/~youjustlostthegame/printname.php

davidmalan

not sent in URL

# POST and GET equally insecure!

- ▶ It's still sent in plaintext, regardless. One just shows up in the URL, while the other doesn't.



A screenshot of the Twitter login interface. It features a light blue background with a grid pattern. There are two input fields: 'Username' and a password field with masked characters. Below the 'Username' field is a checkbox labeled 'Remember me'. To the right of the password field is a link that says 'Forgot it?'. A 'Sign in' button is positioned to the right of the password field. A red arrow points from the password field down to the network request details below.

Request URL: `https://twitter.com/sessions?phx=1`  
Request Method: POST

▼ Form Data      view URL encoded

`session[username_or_email]:`

`session[password]: thisisacat`

`scribe_log: [{"event_name": "web:front:login_callout:form:login_click\nt", "category": "client_event", "ts": 1321171469570}]`

`redirect after login:`

# \$\_SESSION

- ▶ Used to store information about the current HTTP session.

```
// example from pset7 getting user's cash  
$rows = query("SELECT cash FROM users WHERE id = ?", $_SESSION["id"]);
```

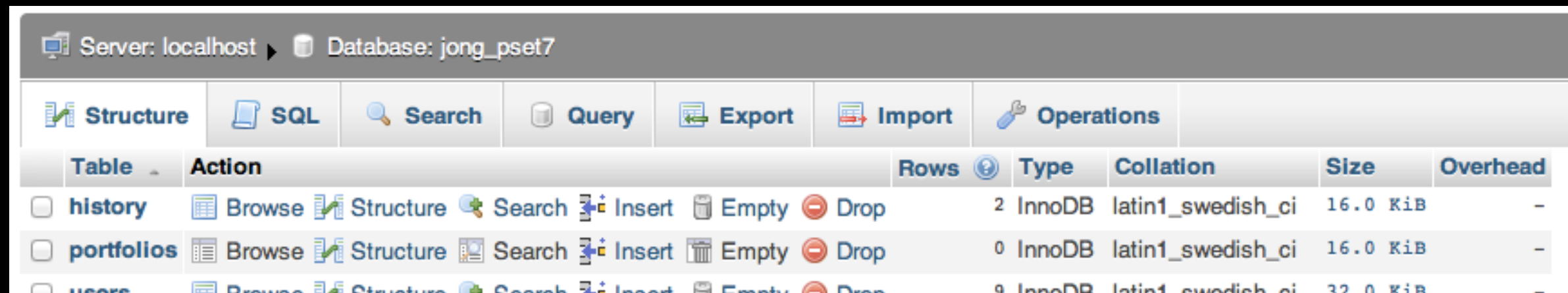
# Structured Query Language

- SQL is a programming language designed for managing databases.



# Databases

- ▶ A database is a collection of tables.



The screenshot shows a database management tool interface. At the top, it displays 'Server: localhost' and 'Database: jong\_pset7'. Below this is a toolbar with icons for 'Structure', 'SQL', 'Search', 'Query', 'Export', 'Import', and 'Operations'. The main area is a table listing database tables. The table has columns: 'Table', 'Action', 'Rows', 'Type', 'Collation', 'Size', and 'Overhead'. Three tables are listed: 'history', 'portfolios', and 'users'.

| Table                               | Action   | Rows | Type   | Collation         | Size     | Overhead |
|-------------------------------------|--|------|--------|-------------------|----------|----------|
| <input type="checkbox"/> history    | Browse  Structure  Search  Insert  Empty  Drop | 2    | InnoDB | latin1_swedish_ci | 16.0 KiB | -        |
| <input type="checkbox"/> portfolios | Browse  Structure  Search  Insert  Empty  Drop | 0    | InnoDB | latin1_swedish_ci | 16.0 KiB | -        |
| <input type="checkbox"/> users      | Browse  Structure  Search  Insert  Empty  Drop | 9    | InnoDB | latin1_swedish_ci | 32.0 KiB | -        |

Each table represents a collection of similar objects.  
For example, a table of users:



The screenshot shows a database table view with columns: 'id', 'username', 'hash', and 'cash'. The table contains three rows of data. Each row has a checkbox, an 'Edit' button, a 'Copy' button, and a 'Delete' button.

|   | id | username | hash                                  | cash       |
|---|----|----------|---------------------------------------|------------|
| <input type="checkbox"/> Edit Copy Delete | 1  | caesar   | \$1\$Y01fprd3\$BA4jQZMm2rmb46EgU7RwN/ | 10000.0000 |
| <input type="checkbox"/> Edit Copy Delete | 2  | chartier | \$1\$NhaqO3f8\$g4zPyTt2KSKdD7HnMI/nK0 | 10000.0000 |
| <input type="checkbox"/> Edit Copy Delete | 3  | guest    | \$1\$3urY0m7m\$PAsveAdEcMgzlyxSKF4cs0 | 10000.0000 |

# Why are they useful?

- ▶ Permanent store for objects, way to track and manage those objects easily -- think of something like user accounts.
- ▶ Very easy paradigms, most essential in SQL are
  - ▶ SELECT
  - ▶ INSERT
  - ▶ DELETE
  - ▶ UPDATE

# SELECT

- ▶ Select rows from a database matching a criterion.

| #                          | Column    | Type         |
|----------------------------|-----------|--------------|
| <input type="checkbox"/> 1 | classid   | int(16)      |
| <input type="checkbox"/> 2 | classname | varchar(255) |
| <input type="checkbox"/> 3 | awesome   | tinyint(1)   |
| <input type="checkbox"/> 4 | slogan    | varchar(255) |







|                          |         |      |                      |
|--------------------------|---------|------|----------------------|
| + Options                |         |      |                      |
| ←T→                      |         |      |                      |
| <input type="checkbox"/> | Edit    | Copy | Delete               |
| 1234                     | CS50    | 1    | Wanna Learn HTML?    |
| <input type="checkbox"/> | Edit    | Copy | Delete               |
| 4321                     | STAT110 | 1    | FIND ALL THE MOMENTS |

SELECT \* FROM classes WHERE awesome = '1';

|                          |         |      |                      |
|--------------------------|---------|------|----------------------|
| + Options                |         |      |                      |
| ←T→                      |         |      |                      |
| <input type="checkbox"/> | Edit    | Copy | Delete               |
| 1234                     | CS50    | 1    | Wanna Learn HTML?    |
| <input type="checkbox"/> | Edit    | Copy | Delete               |
| 4321                     | STAT110 | 1    | FIND ALL THE MOMENTS |

# SELECT

- ▶ How does this look like in code? Integrate PHP!

|                          |  |  |  |         |         |
|--------------------------|--|--|--|---------|---------|
| + Options                |  |  |  |         |         |
| ←T→                      |  | classid  | classname  | awesome | slogan  |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | 1234    | CS50    |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | 4321    | STAT110 |

```
<?php
```

```
//construct sql string  
$rows = query("SELECT * FROM classes WHERE awesome = '1'");  
if ($rows === false)  
    apologize("Query failed. Sadface.");
```

```
foreach ($rows as $row)  
{  
    echo $row["classname"] . ": " . $row["slogan"] . "<br/>";  
}  
?>
```

quit with error message if query didn't work

# SELECT

- ▶ How does this look like in code? Integrate PHP!

| + Options          |  |  |  | classid | classname | awesome | slogan               |
|--------------------|--|--|--|---------|-----------|---------|----------------------|
| ← T →              |  |  |  | 1234    | CS50      | 1       | Wanna Learn HTML?    |
| ☐ Edit Copy Delete |  |  |  | 4321    | STAT110   | 1       | FIND ALL THE MOMENTS |

<?php







```
//construct sql string
$rows = query("SELECT * FROM classes WHERE awesome = '1'");
if ($rows === false)
    apologize("Query failed. Sadface.");
```

```
foreach ($rows as $row)
{
    echo $row["classname"] . ": " . $row["slogan"] . "<br/>";
}
?>
```

CS50: Wanna Learn HTML?

# SELECT

- ▶ How does this look like in code? Integrate PHP!

|                          |  |  |  |         |         |
|--------------------------|--|--|--|---------|---------|
| + Options                |  |  |  |         |         |
| ←T→                      |  | classid  | classname  | awesome | slogan  |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | 1234    | CS50    |
| <input type="checkbox"/> |  Edit |  Copy |  Delete | 4321    | STAT110 |

\$row

```
<?php
```

```
//construct sql string
```

```
$rows = query("SELECT * FROM classes WHERE awesome = '1'");
```

```
if ($rows == false)
```

```
    apologize("Query failed. Sadface.");
```

```
foreach foreach ($rows as $row)
```

```
{
```

```
    echo $row["classname"] . ": " . $row["slogan"] . "<br/>";
```

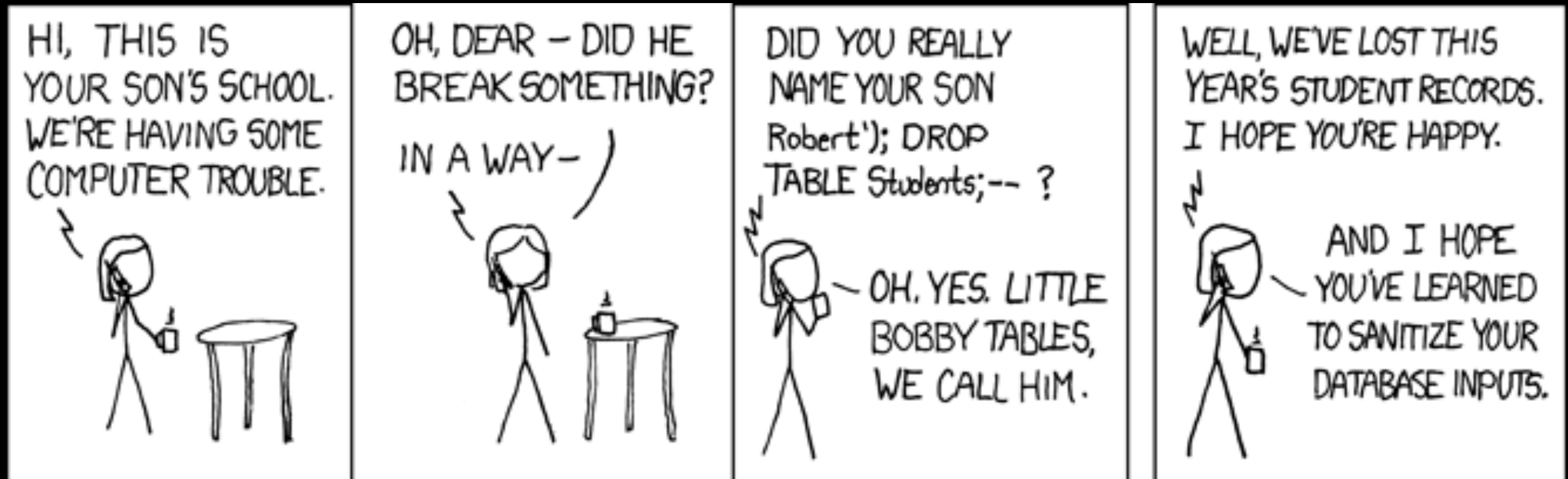
```
}
```

```
?>
```

CS50: Wanna Learn HTML?

STAT110: FIND ALL THE MOMENTS

# SQL Vulnerabilities



# SQL Injection

## ► How does this work?

```
<?php

$rows = query("SELECT * FROM users
              WHERE username = '{$_POST['username']}'
              AND password = '{$_POST['password']}'");

if ($rows === false)
    apologize("Sadface.");

if (count($rows) > 0)
{
    $_SESSION["id"] = $rows[0]["id"];
}
else
{
    apologize("Sadface.");
}
?>
```

# SQL Injection

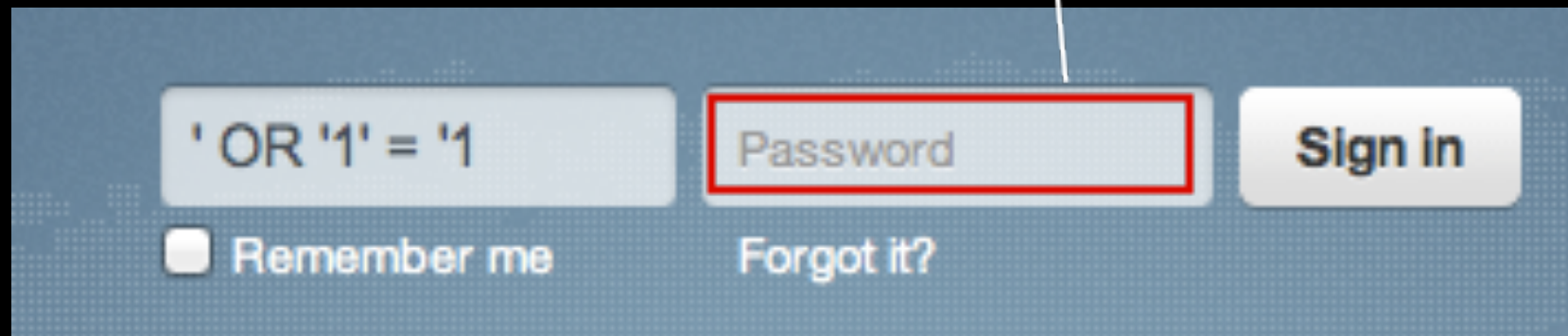
## ► How does this work?

```
<?php
```

```
$rows = query("SELECT * FROM users  
WHERE username = '{$_POST['username']}'  
AND password = '{$_POST['password']}'");
```

```
if ($rows === false)  
    apologize("Sadface.");
```

```
if (count($rows) > 0)  
{  
    $_SESSION["id"] = $rows[0]["id"];  
}  
else  
{  
    apologize("Sadface.");  
}  
?>
```



The image shows a login form with a light blue background. On the left, there is a text input field containing the string `' OR '1' = '1`. To its right is a password input field with a red border, containing the text `Password`. Further right is a button labeled `Sign in`. Below the text input is a checkbox labeled `Remember me`. Below the password input is a link labeled `Forgot it?`. A white arrow points from the password input field to the `{$_POST['password']}` part of the PHP code above.

# SQL Injection

## ► How does this work?

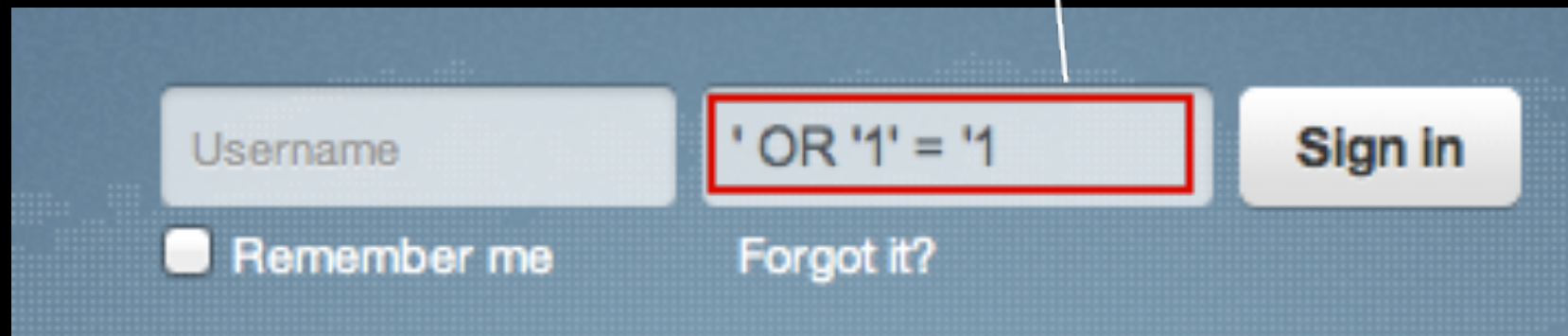
**always true**  
so, returns ALL the rows

```
<?php

$rows = query("SELECT * FROM users
              WHERE username = '{$_POST['username']}'
              AND password = ' OR '1' = '1'");

if ($rows === false)
    apologize("Sadface.");

if (count($rows) > 0)
{
    $_SESSION["id"] = $rows[0]["id"];
}
else
{
    apologize("Sadface.");
}
?>
```



The image shows a login form with a light blue background. It contains a 'Username' input field, a password input field (highlighted with a red border and containing the text `' OR '1' = '1'`), a 'Sign in' button, a 'Remember me' checkbox, and a 'Forgot it?' link. An arrow points from the text box in the top right to the password input field.

# Solution - PDO (question marks!)

```
<?php

$rows = query("SELECT * FROM users
              WHERE username = ?
              AND password = ? ", $_POST["username"],
              $_POST["password"]);

if (count($rows) > 0)
{
    $_SESSION["id"] = $rows[0]["id"];
}
else
{
    apologize("Sadface.");
}
?>
```

note: for one, this is still pretty terrible because you should never, ever, store passwords in plaintext in your database... hash first.

Best of Luck!





# **That quiz 1 review**

by Oreoluwatomiwa Oluwole O. A.  
Babarinsa a.k.a Ore B.

# JavaScript

Or : How I learned to stop worrying and love  
client-side scripting

**JavaScript is kinda cool  
... I guess**

# JavaScript is ...

- useful for client-side scripting
  - Do you really want to have to ping back to the server to handle *\*every\** user interaction
  - Also, you can't animate a button using only server-side scripting

# Things JS also is...

- dynamically-typed
  - The types of your variables are only checked when you run the program
- similar in syntax to C and PHP
  - You have all your good buddies: `if`, `while`, `for`, `x++`, etc.

# A little bit of JS

```
<!DOCTYPE html>
<html>
  <head>
    <script src="jquery.js"></script>
    <script type="text/javascript">
      var x = 10;
      x++;
      console.log(x);
    </script>
  </head>
  <body>
    <p>Yo</p>
  </body>
</html>
```

# JavaScript Arrays

- you would declare an array as such

```
var array = [1, 3, 5];
```

- and access it like this

```
array[1] == 3
```

# C arrays vs. JS arrays

- In JS, `array.length` gives you back the length of an array.
- Unlike C, arrays are dynamically sized!
  - You can add elements to them without worrying about writing out of bounds of memory

# Guilty by Association

- JavaScript lets you use **objects** as **associative arrays**, that let you store key/value pairs
- Declared using { }

```
var assoc = { text : "hello" };  
assoc["name"] = "Ore";  
assoc.name != "Joe";
```

```
for (var slide in pres) ...
```

- So, you've written approx. 10 quadrillion loops where all you do is operate on each element in a data structure
- JS makes this easy using the following :  

```
for (var key in data) { ... }
```

## A bit more about `for`

- If `data` was an object, `key` would be key, such that you'll get some corresponding value if you utilize `data[key]`
- You however, don't want to use this for plain old arrays.

# Objects!

- Objects are a great way in JS to encapsulate related data

```
var obj = { name : "Ore"};  
obj.age = 19;
```

- You can switch between array syntax, and object syntax whenever you want with an exception
  - `obj["key with spaces"]` can't accessed with object syntax

# Scoping it out

- In one of JavaScript's more ... controversial features, it has a markedly different approach to scope
  - If a variable is declared using `var` then its scope is limited to the current function (however, not the current loop or if statement!)
  - without `var` : Limited Scope, What Limited Scope? It's global!

# Document Object Model

- The **DOM** (Document Object Model) gives you a programmatic way to manipulate HTML as objects
  - DOM isn't just available in JS! Many other languages have libraries that let them leverage the DOM.
- It's objects all the way down!
  - Every element is an object
    - attributes are properties of the object
    - nested tags are children of a parent tag

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>hello, world</title>
```

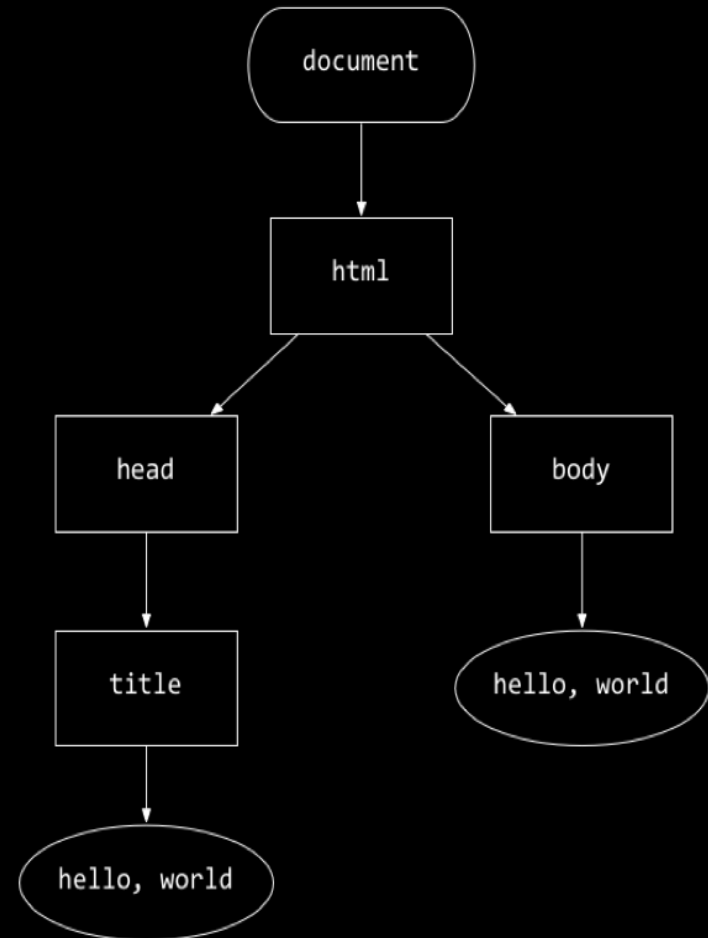
```
  </head>
```

```
  <body>
```

```
    hello, world
```

```
  </body>
```

```
</html>
```



# Where's all the DOM?

- JS loads the DOM into the `document` object
- `document.getElementById("id")`
  - The element with the attribute `id = "id"`

# Events

- the **DOM** lets you attach events to elements
  - events are either some user interaction, or some state change of the page
- For each of these events, we can attach a function that will be executed at that time. We can call this function an **event handler**

# The Events!

- To attach event handlers, you can simply set the value of an HTML attribute for a tag such as `onclick` equal to a function call

```
<button onclick="explode()">  
    Don't Touch  
</button>
```

- You can also grab the DOM element and attach the event handler that way

# jQuery

- jQuery provides many benefits over basic JS, including greater concision.
- It also provides a huge library of cross-browser functions that allows you to do less work
- jQuery provides a slick, syntactically less verbose way of access DOM elements using selectors
  - Same selectors as CSS

# jQuery Selectors

- Let's try to select an element by id!
  - in jQuery this is :  
`$ ( ' #rock ' )`
- Other Selectors
  - `$ ( ".class" )` - all elements of a given class
  - `$ ( "element" )` - all elements with a given tag name

# jQuery Event Handling

- jQuery also gives you a cleaner way of setting up events

```
<script>
$(document).ready(function() {
    $('#myid').click(function() {
        ..
    });
});
</script>
```

# Ajax

- One of the coolest sounding web technologies out there
  - **A**synchronous **J**avaScript **A**nd **X**ML
- Allows you make dynamic HTTP requests without reloading the page!
- Usually has \*nothing\* to with XML anymore

# How does I Ajax?

- In general though you'll need the following
  - A url to send the request to
  - an object containing any data you want to send in your request
  - A function to handle the data you get back

# Ajax Example

```
$(document).ready(function() {  
    // load data via ajax on click  
    $('#greeter').click(function() {  
        $.ajax({  
            url: 'greetings.php',  
            type: 'POST',  
            data: {},  
            success: function(response) {  
                $('#target').text(response);  
            }  
        });  
    });  
});
```

# HTTP Status Codes

- So let's say you sent your Ajax request, but something bad happened to it, you wouldn't be able to tell just by the state of the Ajax request.
- So, Ajax uses HTTP status codes (they aren't specific to Ajax) to report what happened to the request.

# The Actual HTTP Response Codes

200 : All Green

301 : Moved Permanently

401 : UNAUTHORIZED. INTRUDER ALERT.

403 : Forbidden.

404 : Not Found

# Design

Or : How to make the Functional into the Usable

# Design

- It's all about asking the big questions about your applications
  - Who'll be using this app?
  - What will they be using it for?
  - What do my users care about?
  - What DON'T they care about?

# Make it Effortless

- Your User should have to do as little work as possible to leverage your core functionality
  - If I'm using your mobile waffle recipe app, I shouldn't be spending more time trying to find how to make blueberry waffles than actually making them
  - Also, I shouldn't find syrup suggestions for chocolate waffles when I'm looking for how to make blueberry waffles
- Basically, using your applications should be easy for someone who knows nothing about its internals.

# Good Practices

- Paper Prototype
  - Even if everyone else goes digital, software designers will still be sketching out things on paper
  - It gives the ability to quick sketch out designs or workflows, show it around to friends, redo portions on the fly, etc.
- Focus Group
  - Having a group of people who will give honest feedback about your App is critical, also they may help catch bugs!