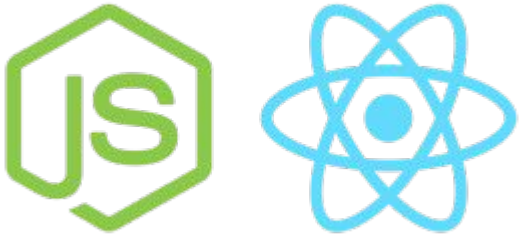
Javascript & React



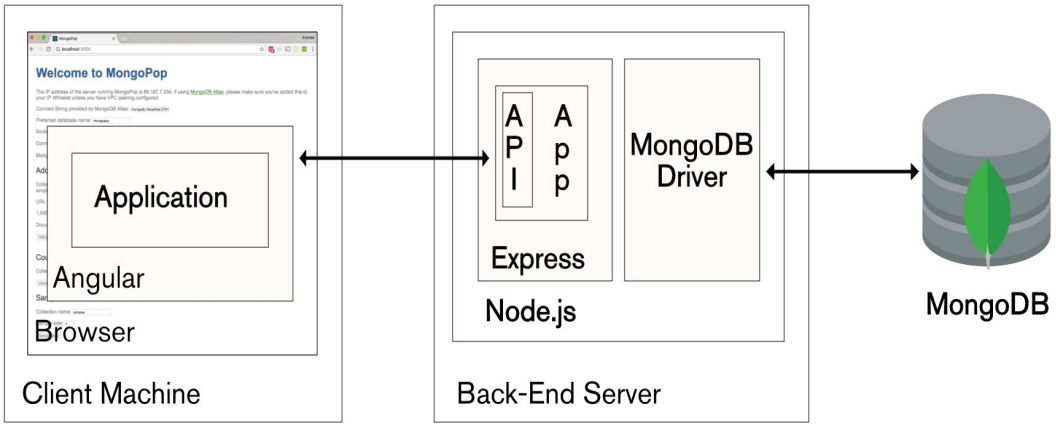
- Anuj Singla
- Pooja Chopra

Agenda

1. Javascript introduction
2. Execution Contexts
3. By value and By reference
4. Immediately Invoked Function Expression(IIFE)
5. Closure
6. bind(), call(), apply() method
7. Prototypal Inheritance
8. ES6 Features
9. Build Javascript library
10. Building game in React

Javascript Introduction

1. What is Javascript?
2. What can you do with it?
3. Where does Javascript code run?
4. Javascript vs ES6 feature?



Execution Contexts

EC is an environment in which javascript code is executed.

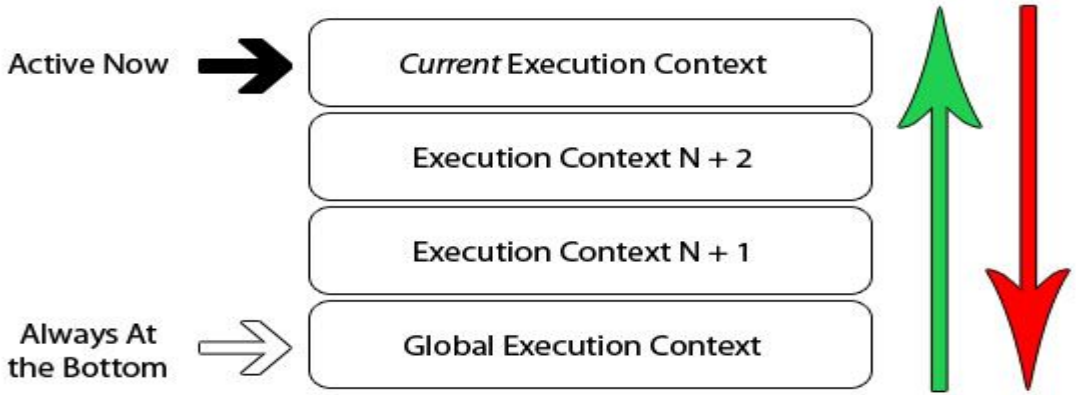
1. Global Execution context
2. Function Execution Context

```
// global context

var sayHello = 'Hello';

function person() {          // execution context

    var first = 'David',
        last  = 'Shariff';

    function firstName() {   // execution context
        return first;
    }

    function lastName() {    // execution context
        return last;
    }

    alert(sayHello + firstName() + ' ' + lastName());

}
```

Execution Contexts

Execution Context Stack

Javascript is a single thread which means only 1 thing is executed at one time in the browser and other action or event queued in the stack called execution stack.



Execution Contexts

1. **Creation Stage**: When we call function it will create function execution context.It will create one object which contain few information.
    a. Create the scope chain
    b. Create variables, functions and arguments.
    c. Determine the value of this object.

```
executionContextObj = {
    'scopeChain': { /* variableObject + all parent execution context's variableObject */ },
    'variableObject': { /* function arguments / parameters, inner variable and function declarations */ },
    'this': {}
}
```

2. Execution stage: assign value and reference to the function and execute the code.

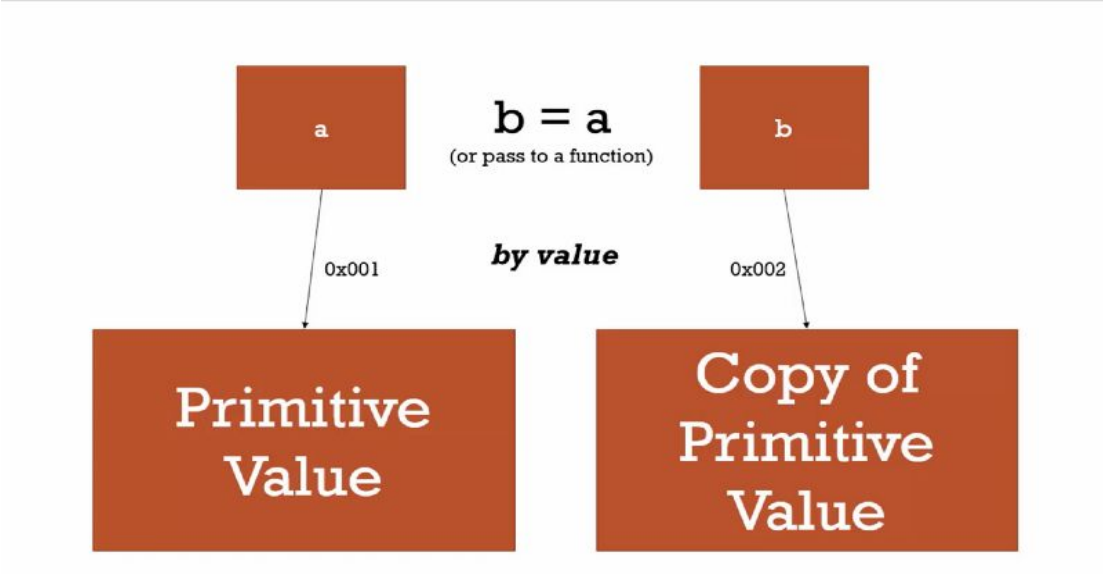
Execution Contexts

Keypoint to remember for Execution Context

1. Single threaded.
2. Synchronous execution.
3. 1 Global context.
4. Infinite function contexts.
5. Each function call creates a new execution context even a call to itself.

By value and By reference

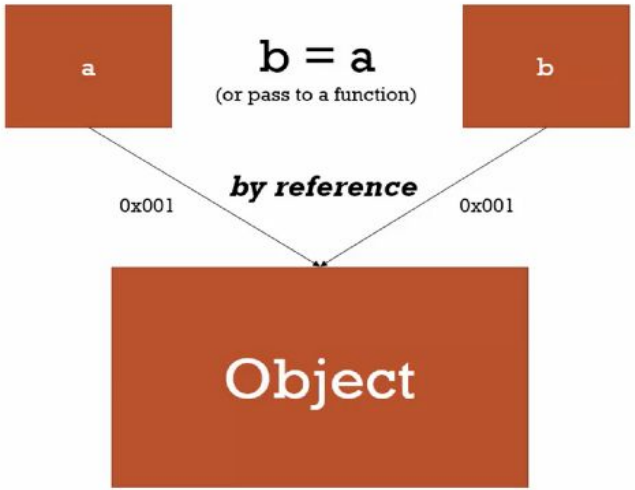
Pass by value (Primitives type):
Javascript always pass by value when you assign primitives value or pass value to the function.



By value and By reference

Pass by Reference (Object)

Javascript always pass by reference when you assign object or pass object to the function.



Immediately Invoked Function Expression(IIFE)

It is javascript function that runs as soon as it is defined.

```
(function(){
  // ...do something...
})();
```

Why IIFE:

1. Local Scope
2. Does not pollute the global namespace
3. Code safety or privacy.
4. Creation — Execution — Discarding of a function in one statement (manage browser memory)

Immediately Invoked Function Expression(IIFE)

```
1  (function () {
2      // logic here
3  })();
4
5  var outsideGreeting = 'outside greeting';
6  (function(name) {
7      var insideGreeting = 'inside greeting';
8      console.log('hello ' +name + ' '+ insideGreeting);
9  }('javascript'));
10
11 console.log(outsideGreeting);
12
13 // IIFE use global variable and expose variable outside from the function
14 var outsideGreeting = 'outside greeting';
15 (function(name, global) {
16     var insideGreeting = 'inside greeting';
17     global.value = 'inside value';
18     console.log('hello ' +name + ' '+ insideGreeting);
19 }('javascript', window));
20
21 console.log(outsideGreeting + ' ' + value);
```

Closure

Closure is a inner function that can access the outer function variables. We are using closure a lot in our project.

The closure has three scope chain.

1. Own scope
2. Outer function scope
3. Global variable scope

```
function greet(whattosay) {

    return function(name) {
        console.log(whattosay + ' '
+ name;
    }

}

var sayHi = greet('Hi');
sayHi('Tony');
```

Closure

Real Time Example:

Suppose, you want to count the number of times user clicked a button on a webpage.

1. Approaches: You could use a global variable, and a function to increase the counter:

Problem: It is global variable and anybody can changes it value.

2. Approaches: You might be thinking of declaring the variable inside the function

Problem: Every time function is called, the counter is set to 1 again.

Closure

Closure (self-invoking function):

```
var ClickCount=(function(){
    var counter=0;

    return function(){
     ++counter;
     // logic
    }
})();

<button onclick="ClickCount()">click me</button>
```

Closure

Closure Rules and side effect:

1. Closures can access outer function variable and parameter even after return outer function.
2. Closures stores reference to the outer function variables.
3. Closures gives wrong output when outer function variables changes value in the loop. We can save using IIFE or let variable.

this object

```
1  //When we use this in the global context(outside the function), this refer to the global object.
2  console.log(this) // Window
3
4  // When we use this inside simple function call this object is represent as a global object.
5  function hello () {
6      console.log(this)
7  }
8
9  // When this object is called inside object method, this object is represent as a current object(Person).
10 let Person = {
11     sayThis: function() {
12       console.log(this);
13    }
14  }
15
16 Person.sayThis() // Person
17
18 // this object is represent an element that fired the event.
19
20 button.addEventListener('click', function() {
21     console.log(this) // button
22 });
```

bind(), call(), apply() method

bind() - It is used to set this object with the specific object when a function is called. It creates a new function with the same body and bind the specific object with the this object.

```
1   var pokemon = {
2       firstname: 'Pika',
3       lastname: 'Chu ',
4       getPokeName: function() {
5           var fullname = this.firstname + ' ' + this.lastname;
6           return fullname;
7       }
8   };
9
10  var pokemonName = function() {
11      console.log(this.getPokeName() + 'I choose you!');
12  };
13
14  // it creates new copy of the function and return it..
15  var logPokemon = pokemonName.bind(pokemon);
16
17  logPokemon(); // 'Pika Chu I choose you!'
```

bind(), call(), apply() method

call() - It is used to set this object with the specific object and also call the function.The first parameter is the this object and second arguments of the function.
Example - .call(this, ‘a’); - > we pass parameter directly.

apply() - it is same like call() method , we pass parameter in the array.
Example - .apply(this, [‘a’]);

bind(), call(), apply() method

```
1  var pokemon = {
2      firstname: 'Pika',
3      lastname: 'Chu ',
4      getPokeName: function() {
5          var fullname = this.firstname + ' ' + this.lastname;
6          return fullname;
7      }
8   };
9
10 var pokemonName = function(lang1, lang2) {
11     console.log(this.getPokeName() + 'I choose you!');
12     console.log(lang1 + lang2);
13 };
14
15 pokemonName.call(pokemon, 'en', 'el');
16 pokemonName.apply(pokemon, ['en', 'el']);
```

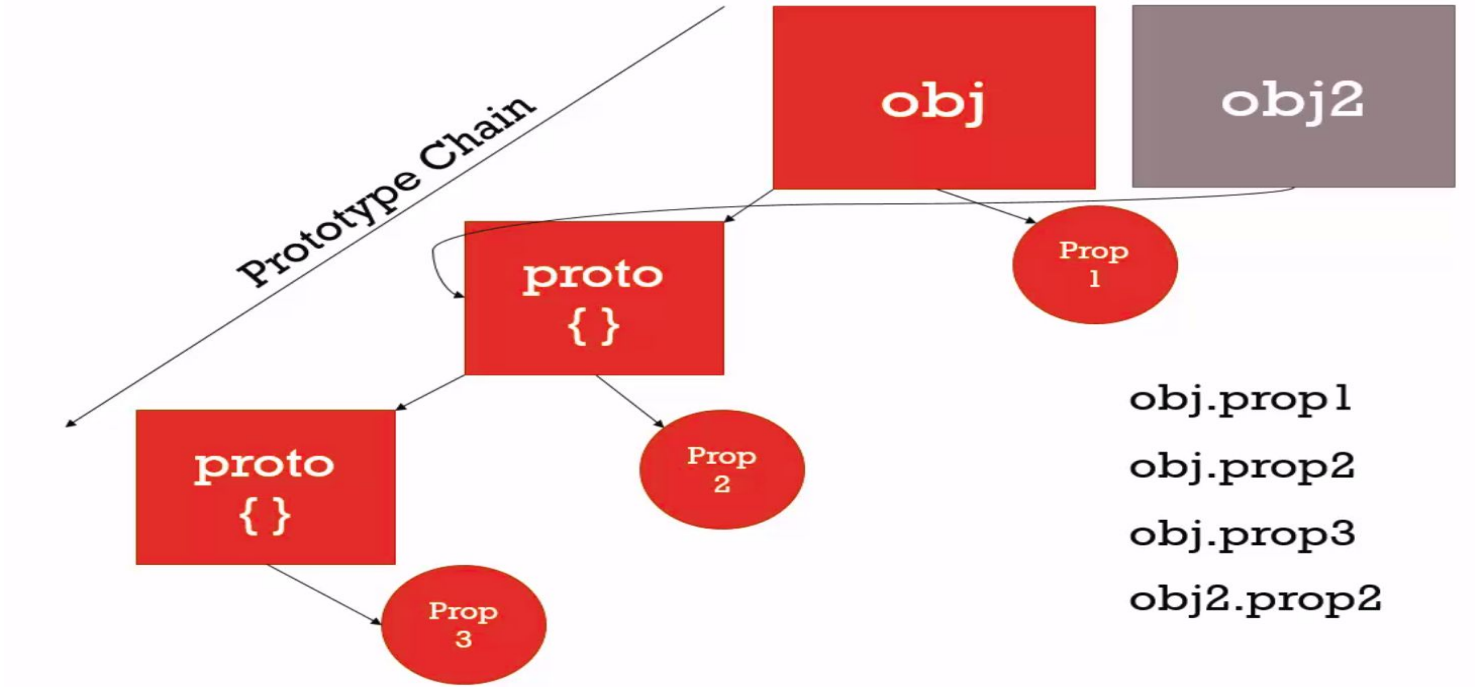
Prototypal Inheritance

Inheritance means one object access properties and method of the other object.

- In JavaScript, all objects have a hidden `[[Prototype]]` property that's either another object or `null`.
- We can use `obj.__proto__` to access it.
- The object referenced by `[[Prototype]]` is called a "prototype".
- If we want to read a property of `obj` or call a method, and it doesn't exist, then JavaScript tries to find it in the prototype.
- If we call `obj.method()`, and the `method` is taken from the prototype, `this` still references `obj`. So methods always work with the current object even if they are inherited.

Function Constructor and .prototype

Prototypal Inheritance



Prototypal Inheritance

```
1  var animal = {
2      eats: true
3  };
4  var rabbit = {
5    jumps: true
6  };
7
8  // rabbit.__proto__ = animal;
9  Object.setPrototypeOf(rabbit, animal);
10
11 // we can find both properties in rabbit now:
12
13 alert( rabbit.eats ); // true
14 alert( rabbit.jumps ); // true
```

Prototypal Inheritance

```
1  // function constructor and .prototype
2  function Person(firstName, lastName) {
3      this.firstName = firstName;
4      this.lastName = lastName;
5  }
6  Person.prototype.getFullName = function() {
7      console.log(this.firstName + ' ' + this.lastName);
8  }
9
10 var john = new Person('John', 'Doe');
11 console.log(john);
12 console.log(john.getFullName());
13
14 var jane = new Person('jane', 'Doe');
15 console.log(john);
16 console.log(jane.getFullName());
17
```

ES6 Features

ECMAScript 6 (ES6) introduced many important features that enhanced JavaScript, making it more powerful

It allowing you to make your code more readable and easy to understand.

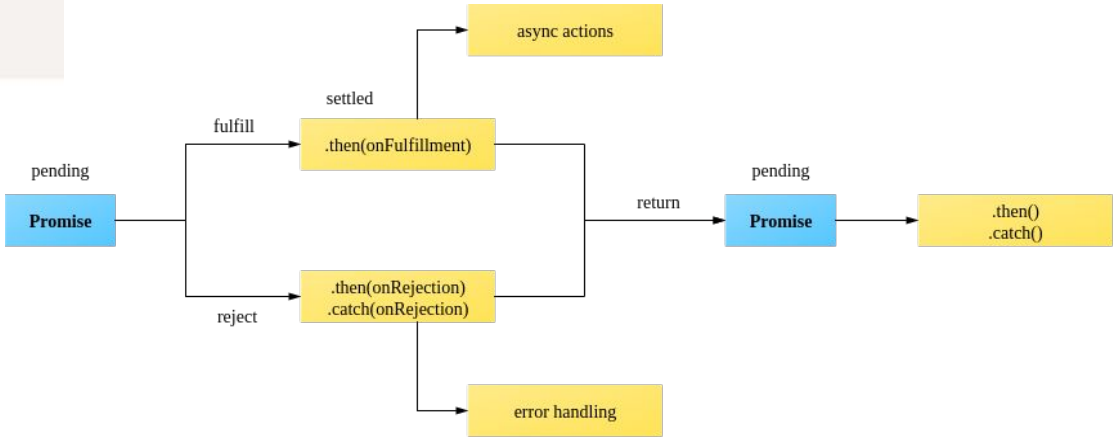
1. Default parameters
2. String Literals
3. Enhanced Object Literals
4. Arrow function
5. Let and const
6. Destructuring Assignment
7. Promises

Promises

Promise help to make asynchronous javascript operations. It will return a value in future or after resolve some operation.

We can create promise with the help of “new Promise”.

```
let promise = new Promise(function(resolve, reject) {
  // executor (the producing code, "singer")
});
```



Async/Await Promises

Async/await is a new way to write asynchronous code.

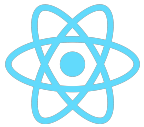
Async/await is actually built on top of promises.

Async : we use async keyword before function and this function always return promise

Await: it will make javascript wait until promise resolved.it will work inside async method.

```
async function foo() {
    return Promise.resolve('promise');
}
foo().then(alert); // promise

const getRequest = async function() {
    const data = await getJSON(); // fetching data from the server
    console.log(data);
    return "done"
}
```

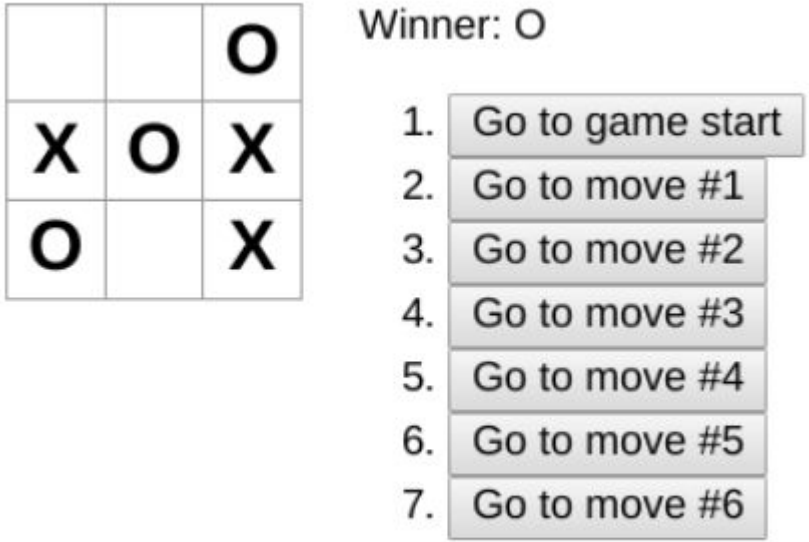
Build Javascript Library



About React

1. A JavaScript library for building user interfaces.
2. M**V**C - React is the **view**.
3. Created and maintained by Facebook.(MIT license)
4. Learn Once, Write Anywhere.
5. It’s all just Javascript(Even JSX technically)
6. Component Based

Component


O
X O X
O X
Winner: O
1. Go to game start
2. Go to move #1
3. Go to move #2
4. Go to move #3
5. Go to move #4
6. Go to move #5
7. Go to move #6

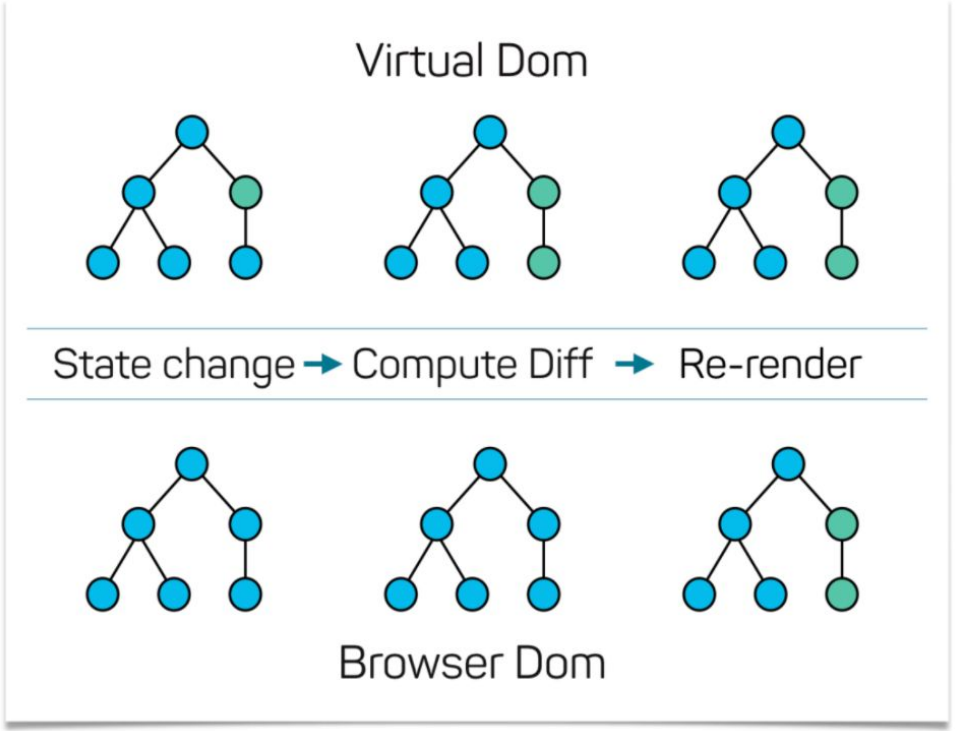
Component

```
1  // Stateless/Functional Component
2  function Hello(props) {
3    return (
4      <div>
5        <h1> Hi </h1>
6        {props.name}
7      </div>
8    );
9  }
```

```
// Stateful Component
class Hello extends React.Component {
    render() {
        return (
            <div>
                <h1> Hi </h1>
                {this.props.name}
            </div>
        );
    }
}
```

<Hello name=”React” />

Virtual DOM

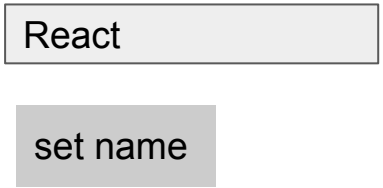

Virtual Dom
State change → Compute Diff → Re-render
Browser Dom

State/Props

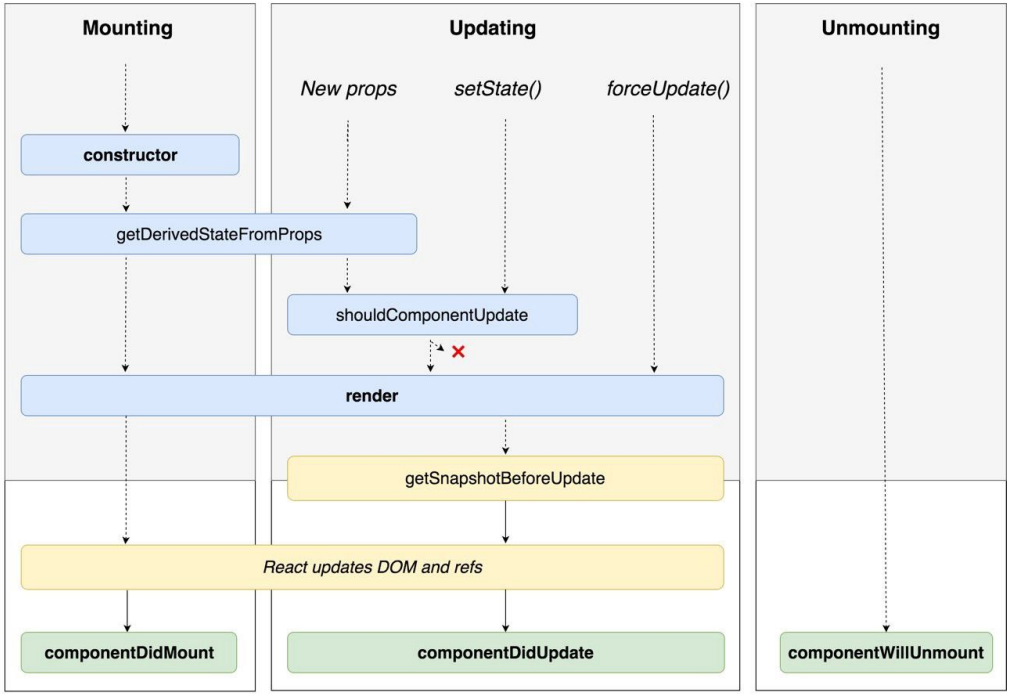

COMPONENT
props
State
DOM
state is used for internal
communication inside a Component

State/Props

```
class Greet extends React.Component {
    state = {
        name: 'react'
    }
    onButtonClick = (e) => {
        e.preventDefault();
        const nameValue = e.target.name.value;
        this.setState({name:nameValue});
    }
    render() {
        return (
            <div className="App">
                <h1>hello {this.state.name}</h1>
                <h1>{this.props.message}</h1>
                <form onSubmit={this.onButtonClick}>
                    <input type="text" name='name'></input>
                    <button>set name</button>
                </form>
            </div>
        )}
};
```



Lifecycle methods



Game Time