ReactJS

Gopalakrishnan Subramani
NodeSense Technologies
gs@nodesen.se
+91 9886991146

NODESEN.SE

A JavaScript library for building user interface

Just the UI

V in M**V**C

React Agenda

- Introduction
- JSX
- Components
- Functional Components
- Virtual DOM
- Props, PropTypes, Default Props
- States
- Component Life Cycles
- Context
- Ref

NODESEN.SE

React

- Single Page Application Development Framework
- Works at Client Side
- Apps are build using EcmaScript 6 (ES6)

Why React?

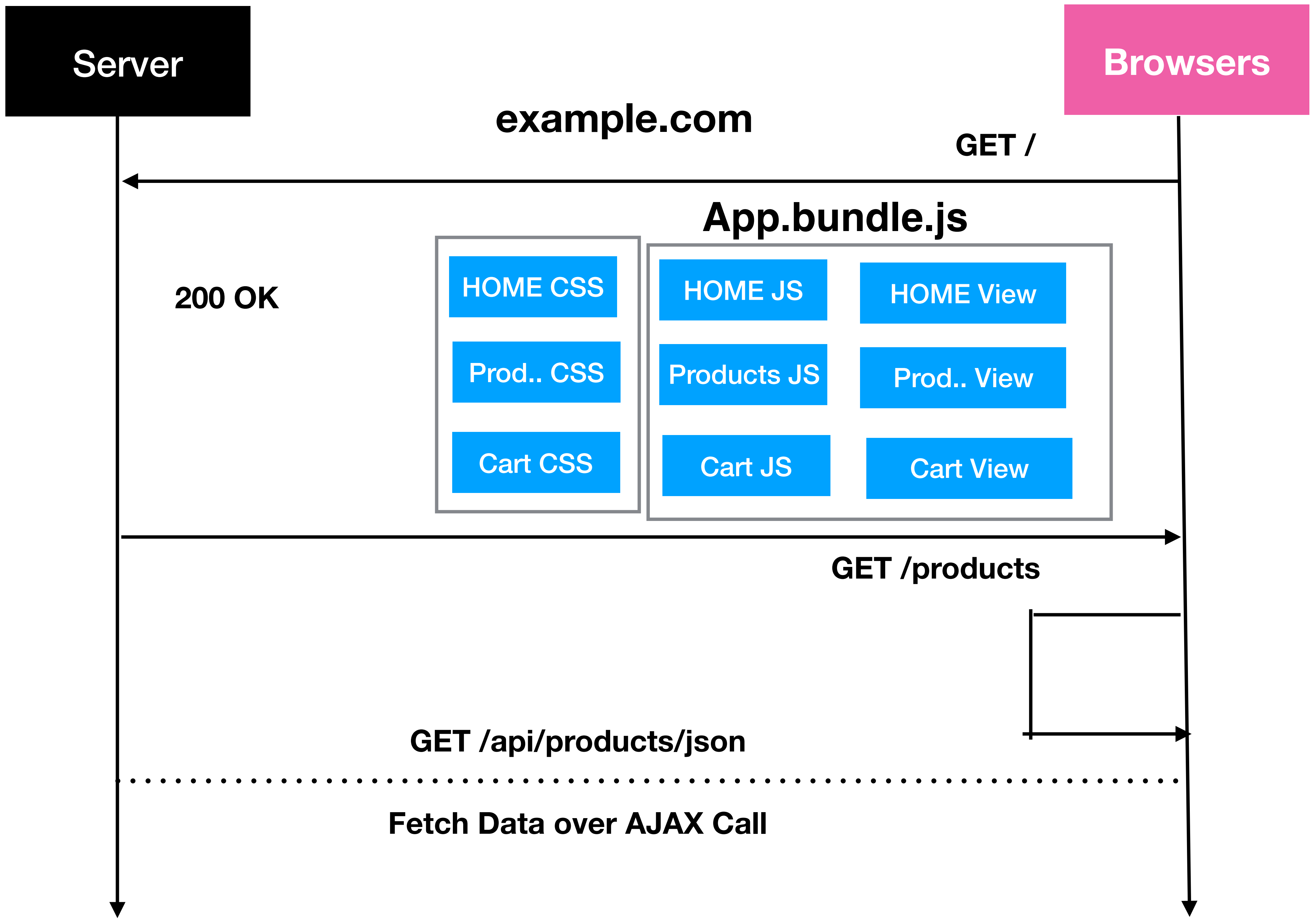
- DOM Mutation is very slow in browsers
- Append, Remove child elements in Real DOM cause performance issues
- Calling setText, attributes, add/remove attributes/ child elements cause slowness

React has Answer for the problems

What is SPA?

- Single Page Application
- A single HTML page, that loads entire application
- SPA is an architectural pattern
- Dynamically update the same page when the user interacts without leaving page
- Uses Ajax & JSON For Data Transfer

Single Page React Application



NODESEN.SE

Advantages of SPA

- Requests to server minimized, faster response
- Parse HTML, create DOM hierarchy only once
- Parse CSS, create CSS structure only once
- Interpret/Compile JavaScript only one
- Dynamic contents, data downloaded on need basics using AJAX
- Consume less battery, CPU resources, mobile friendly

Why DOM Slow

- Technically Browser Layout is slow
- When DOM is touched (add/remove child, update text, update styles) at any way, **Browser Layout algorithms make it slow rendering**
- CSS recalc algorithm, then layout, then repaint, then re-compositing, draw the UI

Read here => http://www.phpied.com/rendering-repaint-reflowrelayout-restyle/

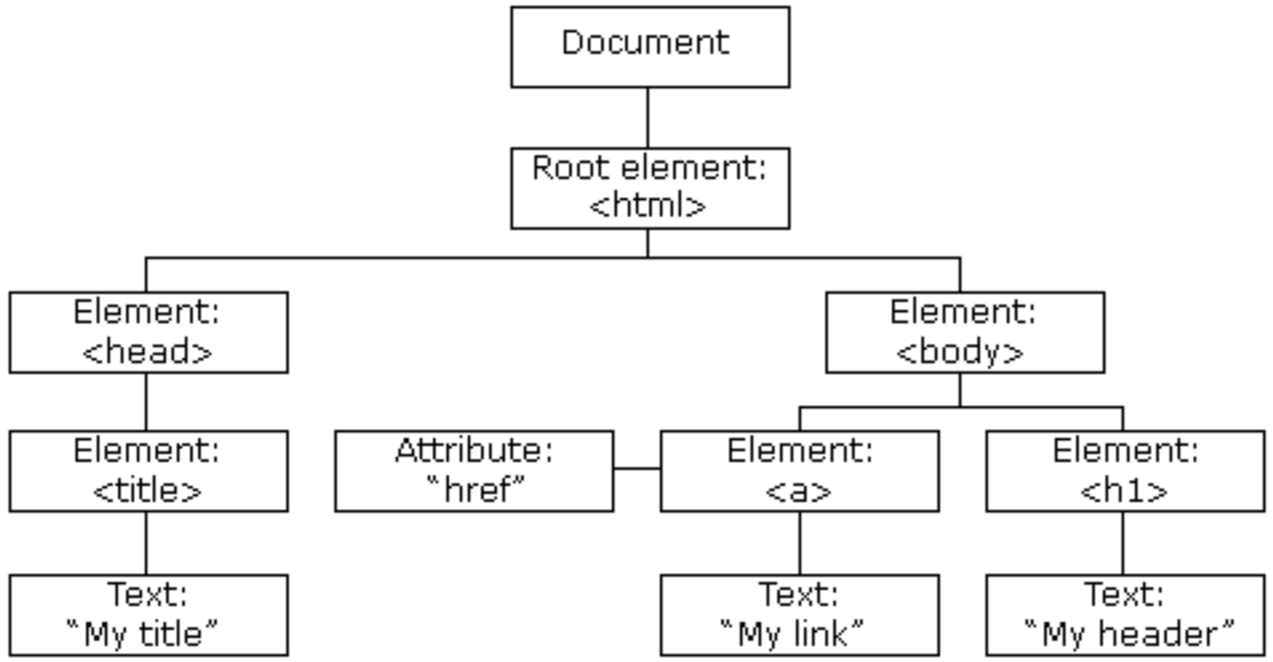
http://gent.ilcore.com/2011/03/how-not-to-trigger-layout-in-webkit.html

How React works?

- Keep developers away from working with real DOM
- Provides **Virtual DOM,** a higher level abstraction over DOM and DOM Mutation (Manipulation)
- Developer works on Virtual DOM
- React update Real DOM in conservative manner

HTML DOM

- Managed by browsers
- Hierarchical, constructed when page loaded
- Use JavaScript to manipulate real DOM (Add/Remove, Show/Hide)



Issues with Real DOM

- Inconsistent across multiple actions
- Hard to Test (Unit Testing, need browsers)
- Expensive, Very slow

Virtual DOM

- Pure JavaScript Object, in-memory representation of the DOM
- Abstraction over Real DOM
- **React Developers works with Virtual DOM, React Framework Sync and Manages Real DOM**

Virtual DOM

- React applies DOM changes on Virtual DOM first
- Then React apply changes on real DOM
- Modify real DOM when changes occurs (like add/ remote DOM elements, DOM Properties)
- Use Efficient **Diffing** algorithm to detect changes, render only changed DOM tree
- Very Fast

Virtual DOM

```
import React, {Component} from "react";

export class Home extends Component {
    // returns a view, a Virtual DOM
    render() {
        return React.createElement(“h1”, {},
               “Welcome to React”)
    }
}
```

A ReactElement is a light, stateless, immutable, virtual representation of a DOM Element.

NODESEN.SE

React.createElement

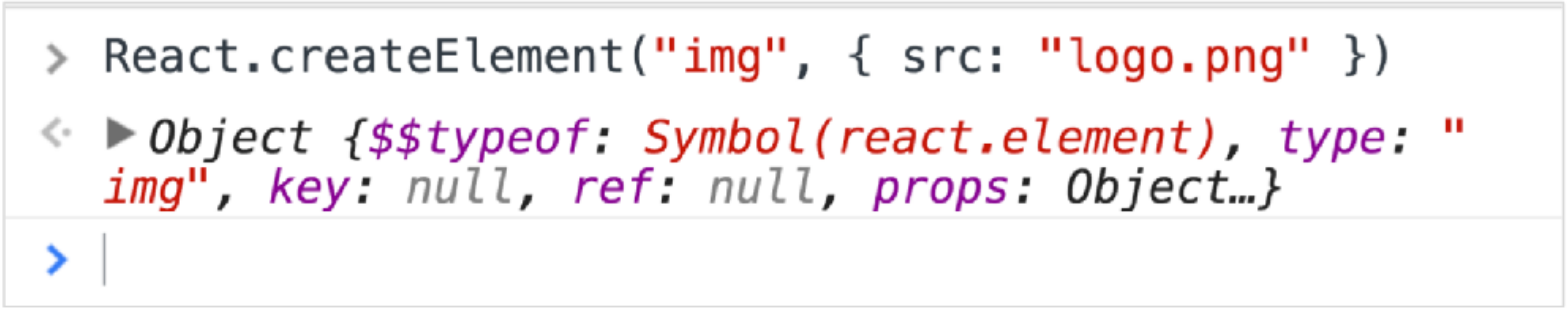
- Creates a virtual DOM, in-memory representation of an element
- Hierarchical structure, similar to HTML DOM

Creating Element

```
// Traditional HTML:
<img src="logo.png" />
```

```
// In React:
React.createElement("img", {src:"logo.png" })
```

The createElement function returns a JavaScript object, but **not really a Browser DOM** element/object



```
> React.createElement("img", { src: "logo.png" })
<· ▶Object {$$typeof: Symbol(react.element), type: "
  img", key: null, ref: null, props: Object…}
> |
```

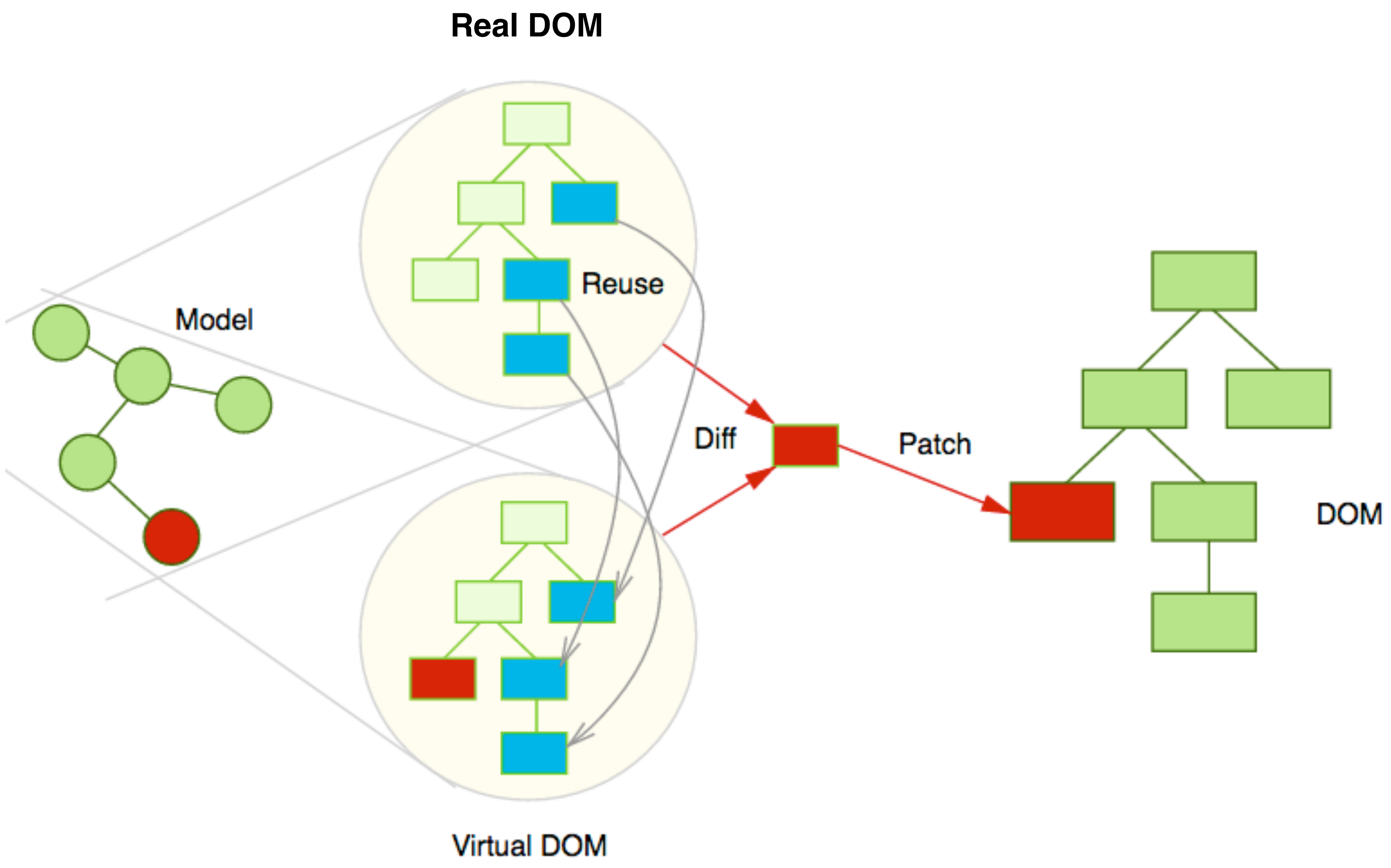
Virtual DOM

- **render()** creates virtual DOM elements as tree hierarchy
- You write the UI in **Virtual DOM** which is NOT real DOM
- React creates/updates Real DOM from Virtual DOM

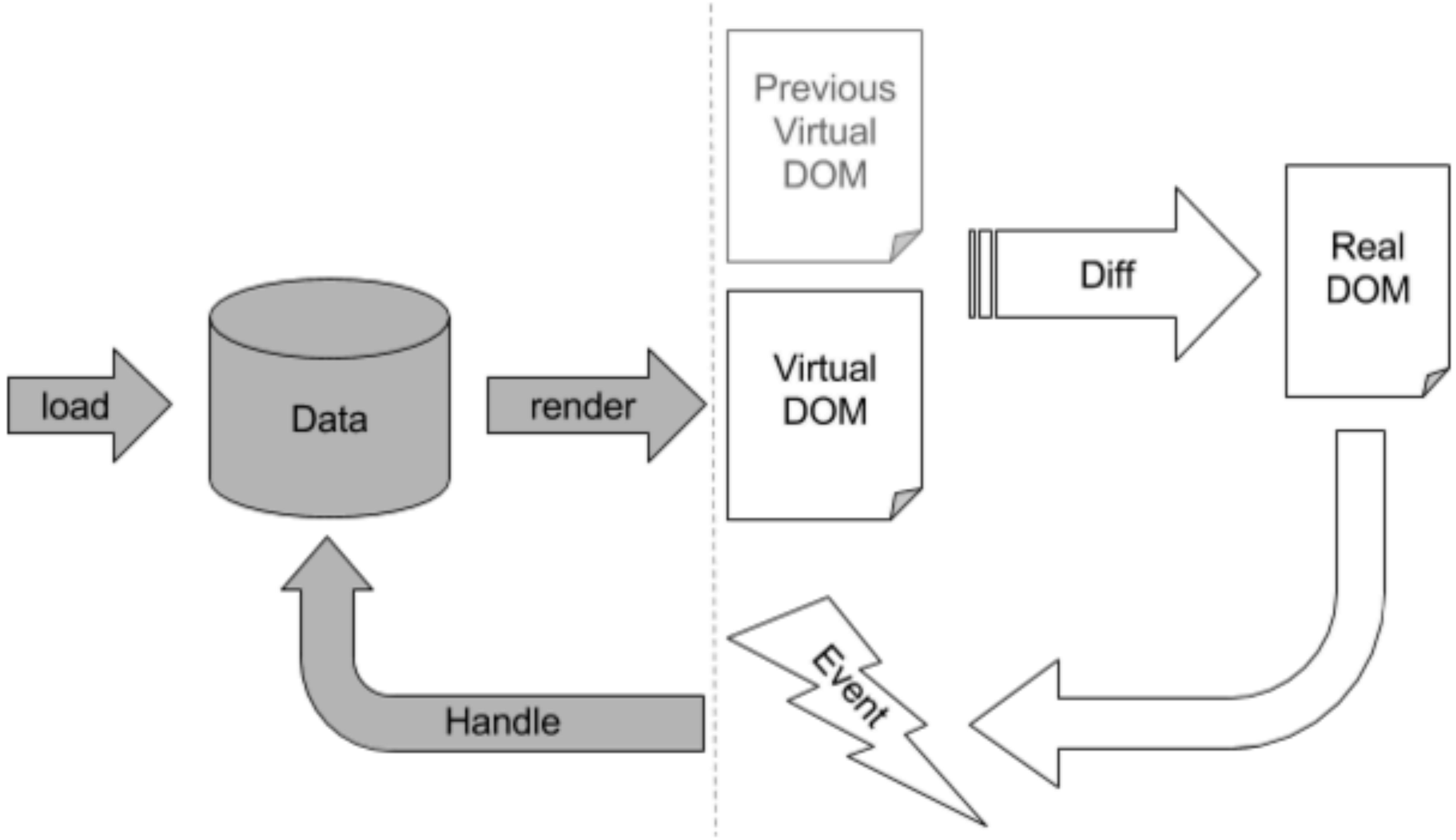
Virtual & Real DOM

- React **Compare** Virtual DOM
- Update Real DOM only if any differences found
- React **Patch** the **differences** in Real DOM

Diff



Virtual DOM

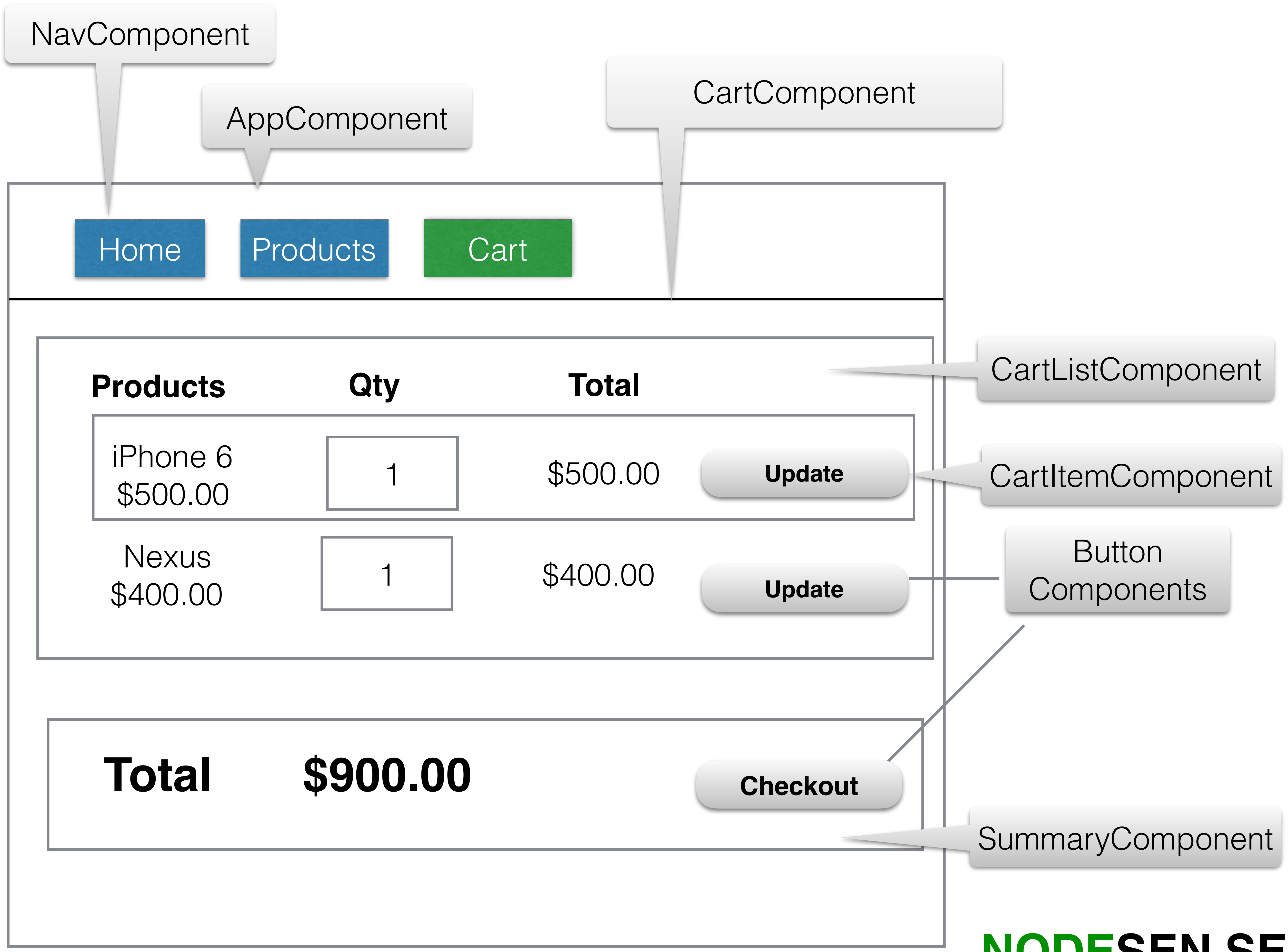


NODESEN.SE

Everything is a Component

Component

- Component is a piece of reusable user interface
- Consists of
- JavaScript Code [Interactions]
- View [Presentation]
- Styles


NavComponent
AppComponent
CartComponent
Home
Products
Cart
Products
Qty
Total
CartListComponent
iPhone 6
$500.00
1
$500.00
Update
CartItemComponent
Nexus
$400.00
1
$400.00
Update
Button Components
Total
$900.00
Checkout
SummaryComponent

NODESEN.SE

ES2015 (ES6)

ECMA Script

European Computer Manufacturers Association

Gopalakrishnan Subramani
NodeSense Technologies
gs@nodesen.se
+91 9886991146

NODESEN.SE

- ES 5 - 2009 (old, popular JavaScript)
- ECMAScript 2015 - June, 2015. [ES6]
- ECMAScript 2016 - June 2016
- ECMAScript 2017 - June 2017
- ES.NEXT Future

let,const for block scope

```
{
    let prefix = ‘Champian’;
    const PI = 3.14;
}

//throws error
console.log(prefix)
```

Scope for prefix, PI is limited within block

const

```
const PI = 3.14
PI = 2.14 //Error

const center = {x:0, y:0}

// Error, can’t change reference
center = {x:20, y:20}
```

for..of loop

```
let points = [10, 20, 30, 40, 50];
for (let point of points) {
    console.log(point)
}
```

Output

```
10
20
30
40
50
```

for ..of - iterates over elements

NODESEN.SE

Default Parameters

```
function add(a = 0, b = 0) {
  return a + b
}
```

add()	a & b are 0
add(10)	a is 10 & b is 0
add(undefined, 20)	a is 0 & b is 20

NODESEN.SE

Rest Operator …

Rest operator helps to work with variable arguments

```
function print(name, …args) {

    for (let arg of args) {
        console.log(arg)
    }
}

print(“hello”, 10, 20, 30, {x: “23”});
```

…args is an array[]

Spread Operator …

```
let numbers = [10, 20]

let list = [0, …numbers, 30]

The list contains 0, 10, 20, 30
```

Spread operator unpack array elements, useful for cloning, merging array elements

DeConstruct Object

Deconstruct helps to declare variables and initialise values from object at same time

```
let data = {a : 10, b: 20, c: 30}

//declare a and b, d variables,
//a = 10
//b = 20
//d = undefined
let {a, b, d} = data;

console.log(a, b, d);
//prints 10 20 undefined
```

$\longrightarrow$

let a = data.a
let b = data.b
let d = data.d

NODESEN.SE

DeConstruct in Function Arguments

```
//has two arguments, a and b
//data.a is passed to argument a,
//data.b is passed to argument b

function add({a, b}) {

    return a + b
}
```

```
let data = {a : 10, b: 20}

add(data)
```

NODESEN.SE

```
class Shape {

    constructor(name2) {
        this.name = name2;
    }

    getName() {
       return this.name;
    }

    setName(name) {
          this.name = name;
    }
}

let s = new Shape(“Circle”)
```

Class Constructor

1. **this** is explicit
2. No Destructor

NODESEN.SE

Inheritance

```
class Circle extends Shape {

    constructor(point) {
        super(“Circle”)
        this.point = point;
    }

    getPoint() {
            return this.point;
    }

    toString() {
        return super.getName() + this.point;
    }
}
```

Base class constructor

Base class method

NODESEN.SE

Static

```
class Circle extends Shape {
    static getType() {
        return “CIRCLE”;
    }
}


//Static Variable at class level
Circle.PI = 3.14;
```

console.log(Circle.getType())

console.log(Circle.PI)

NODESEN.SE

Arrow Operator

```
let power = n => n * n;
```

Single line, one param, implicit return

```
let add = (a, b) => a + b;
```

Single line, two params, implicit return

```
let factorial = (n) => {
    let result = 1;

    if (n <= 1)
        return 1;

    return n * factorial(n-1);
}
```

Multi line, needs braces ‘{‘ & ‘}’, and explicit **return** statement

power(10)

“this” with Arrow Operator

`this` is picked up from surroundings (*lexical*).

```
class Counter {
   start() {
         this.count = 0;

         setInterval( ()=> {

               this.count++;

         }, 2000)
}
```

NODESEN.SE

“this” vs that without Arrow

```
C = new Counter();
c.start()
```

```
class Counter {
  start() {
      this.count = 0;
      var that = this;

      setInterval( function() {
          //this.count++; //won't work
          that.count++; //works

      }, 2000)
}
```

Function called by timer

NODESEN.SE

Template Literals

Template => Backquote ` and ${}

```
let framework = “React”;
let version = “15.4”

//ES6 way BACK QUOTE `
let title = `Framework ${framework} ${version}`;
```

title => Framework React 15.4

Module

- In ES6, Each File is a module
- Module can have classes, variables, const, functions
- A Module can export functions, const, classes, objects
- A File name is used for importing
- File name works as module name resolution

```
//product.js
export class Product {
 constructor() {

  this.name = "iphone"
}

}

//local to product.js
//not exported
class InnerClass {

}

let name = "product"
```

```
//product-edit.component.js

import {Product} from "./product"

var product = new Product()
```

InnerClass and name cannot be imported

Modules

NODESEN.SE

```
//product.js
export class Product {
 constructor() {
   this.name = “iphone”
}
}

export function
    add(a, b) {
  return a + b;
}

export const PI=3.14;
```

```
//product-edit.component.js

import {Product,
         add,
         PI}
             from “./product"

var product = new Product()
add(10, 20)
console.log(PI)
```

Modules

```
//product.js
export class Product {
 constructor() {
   this.name = “iphone”
}
}

export function
     add(a, b) {
  return a + b;
}

export const PI=3.14;
```

```
//product-edit.component.js

import * as pm
            from "./product"


let product = new pm.Product()
pm.add(10, 20)
console.log(pm.PI)
```

Modules

NODESEN.SE

```
//product.js
export class Product {
 constructor() {

  this.name = “iphone”
}

}
```

```
//product.models.js

export class Product {
  ….

}
```

Modules

```
//product-edit.component.js

import {Product} from "./product"

var product = new Product()

//ERROR
//import {Product} from
//          “./product.models”

//or using alias

import {Product as ProductModel}
from “./product.models"

var model = new ProductModel()
```

NODESEN.SE

```
//product.js
export class Product {
 constructor() {

  this.name = "iphone"
}

}

export default class
    ProductService {
}
```

```
//product-edit.component.js

import {Product} from "./product"

import ProductService
             from "./product"

import ProductServiceEx
             from "./product" OR

import ProductService, {Product}
             from "./product"


import ProductServiceEX, {Product}
             from "./product"
```

Each module can export One default, don't use {} for default import

Default export

NODESEN.SE

Other Features of ES6

- Promise class
- Map/Set collections
- Array map, filter, and methods
- Object.assign
- And more

Notable Features of ES6

- let, const for variable declaration over ‘var’
- module export, import
- Classes, Constructor, Inheritance, Polymorphism
- Getter, Setter Properties
- Lambda (Fat Arrow) functions
- Default Parameters
- for..of loop

ES7

- Array.prototype.includes, check if array contains an element
- Exponent operators (**) 2 ** 3 returns 8

ES8

- Async keywords for promise handling
- **async**
- **await**

ES.Next

- @decorator
- static declarations for class variable inside class
- Spread operator for object {…}

A lot more..References

- http://es6-features.org/
- http://www.ecma-international.org/ecma-262/6.0/

Component Types

- Class Component
- Functional Component

React Library

- Consists of Component, PureComponent
- createElement and other useful classes and functions to create react application components

React Package

```
import React, {Component,
               PureComponent} from "react";
```

Contains classes needed for createElement, Component, PureComponents, the core library for react implementation. Creates and returns Virtual DOM on render Methods

Class Component

Default Import

```
import React, {Component} from "react";

export class Home extends Component {
     render() {
          return (
               <div>
                    …
               </div>
          )
     }
}
```

named Import

NODESEN.SE

Class Component

```
import React, {Component} from "react";

export default class Home extends Component {
    // render method, returns a view
    render() {
        return (
            <div>
                <h2>Home</h2>
                <p>Welcome to our site.....</p>
            </div>
        )
    }
}
```

Class Component

- Class Component owns a **State**, **Life Cycle Methods**
- Functional Component are **stateless**, **no life cycle methods**

```
import React, {Component} from "react";

export default class Home extends Component {
    constructor(props, context) {
        super(props, context);
        this.state = {likes: 1000}
    }

    // Life cycle methods
    componentWillMount() {}
    componentDidMount() {}
    componentWillUnmount() {}

    // data update methods
    componentWillReceiveProps(…) {}
    shouldComponentUpdate(…) {}

    render() {
        return (<div>....</div>)
    }
}
```

*** state, props, context discussed later**

*** Life cycle methods discussed later**

NODESEN.SE

Functional Component

```
import React from "react";

// A component function returns a view
export default function Footer() {
    return (
        <div>
            <hr />
            <p> Copyrights@ 2017, NodeSense </p>
        </div>
    )
}
```

Functional Component

*** Props , Context discussed later**

```
import React from "react";

export default function Footer(props, context) {
    return (
        <div>
            <hr />
            <p> Copyrights@ 2017, NodeSense </p>
        </div>
    )
}
```

Expression

```
import React, {Component} from "react";

export default class Home extends Component {

    render() {
       let title = 'Home';
       let description = 'Welcome to our site…..';

        return (
            <div>
                <h2>{title}</h2>

                <p>{description}</p>
            </div>
        );
    }
}
```

NODESEN.SE

Expression

```
import React, {Component} from "react";

export default class Home extends Component {
    constructor() {
        super();

        this.title = 'Home';
        this.description = 'Welcome to our site.....'
    }

    render() {
        return (
            <div>
                <h2>{this.title}</h2>
                <p>{this.description}</p>
            </div>
        )
    }
}
```

XML in JavaScript?

How come a JavaScript can return a XML?

```
 render() {
        return (
            <div>
                <h2 id=“header”>Home</h2>
                <p>Welcome to our site..</p>
            </div>
        )
    }
```

```
<div>
     ^
SyntaxError: Unexpected token <
```

*This breaks JavaScript

NODESEN.SE

But why?

React has no HTML Template.

React Virtual DOM Views are created using Plain JavaScript

What other options?

NODESEN.SE

React View in JS

Builds Hierarchical Virtual DOM Tree, later rendered into Real DOM

```
React.createElement('div', null,
        React.createElement('h2', {id : 'header'},
           ['Home',
             React.createElement('p', null,
                                  'Welcome to our site..')]
        )
)
```

- Elements are the smallest building blocks of React apps, created programatically

NODESEN.SE

```
// JSX
<form>
< label for="email"> Email:</label>
<input type="email" id="email" class="form-control" />
</form>



// React Way
React.createElement("form",null,
    React.createElement("label", { htmlFor: "email" },
         "Email:"),
    React.createElement("input", { type: "email",
                                   id: "email",
                                   className: "form-control
});
```

NODESEN.SE

Writing View in React is painful

So break JavaScript

Solution?

JSX

Java**S**cript + **X**ML => **JSX**

JSX

- Write HTML/XML in JavaScript
- Like HTML, but not really HTML
- JSX is not part of React Core Lib
- React Team recommends JSX for HTML
- We need optional Tools like Babel
- Babel converts JSX to JavaScript Code

Inlining JSX Breaks JavaScript Syntax

Use Babel to rescue
Babel convert JSX to JavaScript

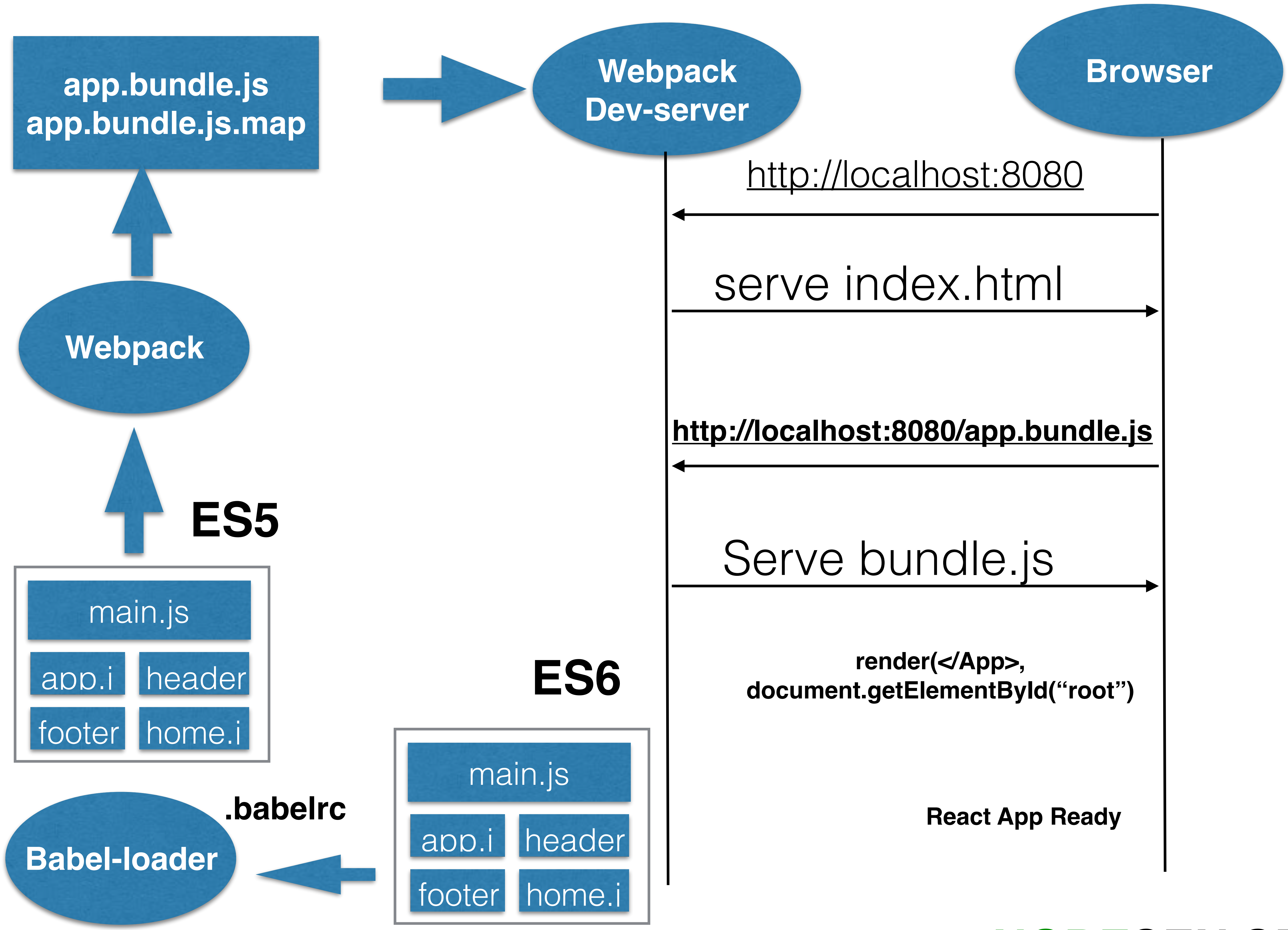
JSX Support

```
>  npm install babel-preset-react --save-dev
```

.babelrc for JSX to JS

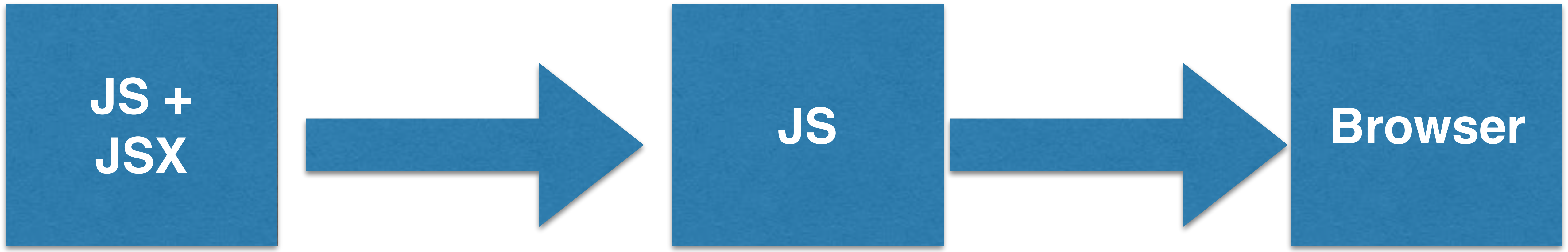
```
{
    "presets": ["es2015", "react"]
}
```

NODESEN.SE


app.bundle.js
app.bundle.js.map
Webpack
Dev-server
Browser
http://localhost:8080
serve index.html
Webpack
http://localhost:8080/app.bundle.js
ES5
Serve bundle.js
main.js
app.j
header
footer
home.j
ES6
render(</App>,
document.getElementById("root")
main.js
.babelrc
React App Ready
Babel-loader
app.j
header
footer
home.j

NODESEN.SE

Babel Preset



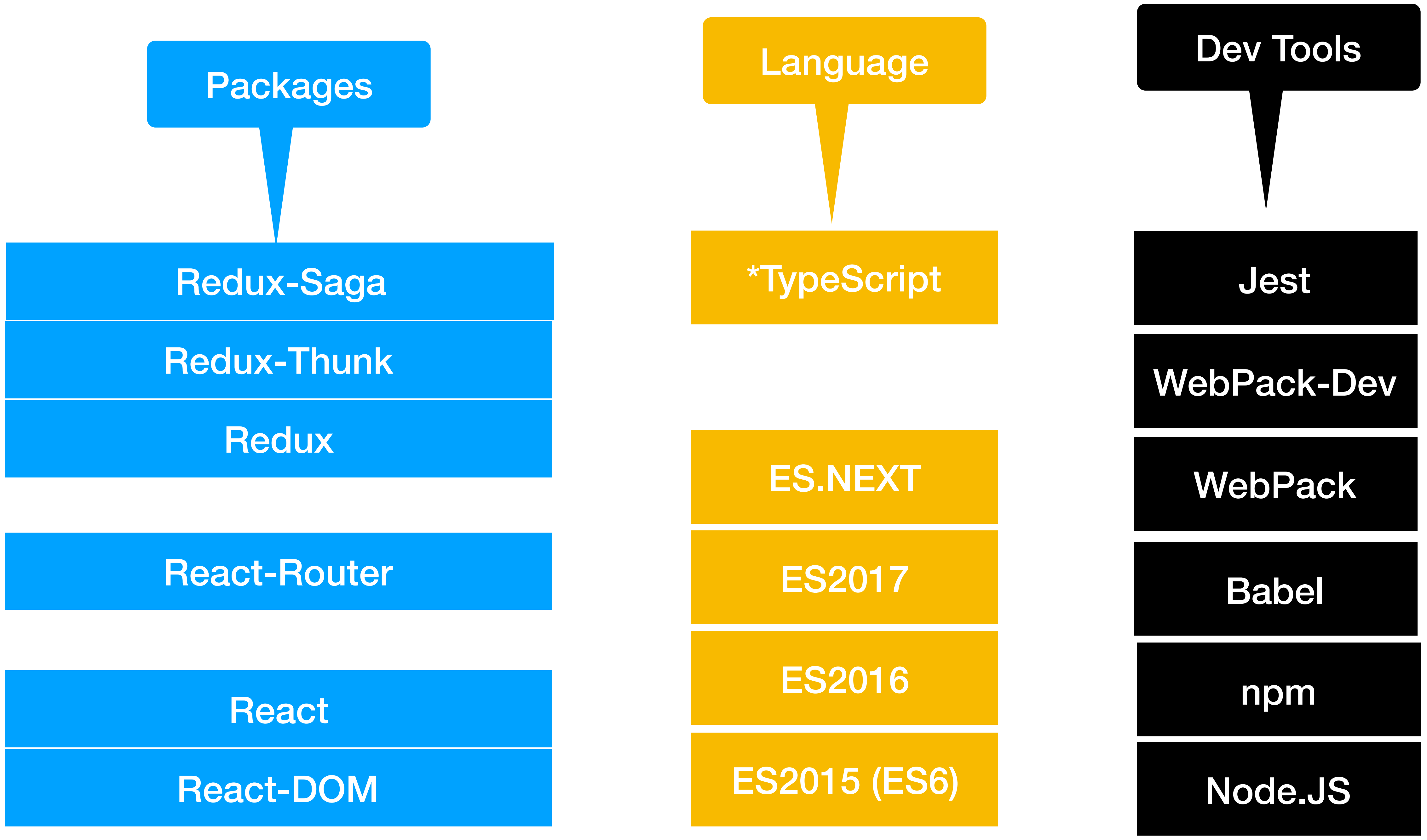
babel
react-preset

Node.js
Development

Browser
No JSX

NODESEN.SE

Building Blocks of React App



*TypeScript is optional

NODESEN.SE

Install React-DOM Library

```
>    npm install   react-dom  --save
```

update package.json

package.json

```
"dependencies": {
  "react": "^15.4.2",
  "react-dom": "^15.4.2"
}
```

NODESEN.SE

React-DOM Package

```
import {render} from “react-dom”;
```

```
render( <App>
        </App>
        , document.getElementById("root"))
```

- Bootstrap react component into Browser DOM
- Manages Diffs between real and virtual DOM
- Patch real DOM if any differences

prop-types

```
>   npm install   prop-types --save
```

```
import PropTypes from "prop-types";

Address.propTypes = {
   city: PropTypes.string.isRequired,
   state: PropTypes.string,
   pincode: PropTypes.number.isRequired
}
```

Useful for describing component props, context types, mandating required properties

React

- Render your application UI
- Responds to events
- Uses Component
- Uses JavaScript as Template (JSX)

Core React Ideas

- Build components, lots of them, App is all about components, App, Pages, Widgets, Routes, everything components
- Re-render DOM, don’t mutate, no show/hide
- Virtual DOM for better performance

Thinking in Components

- Components are compassable
- Reusable
- Maintainable
- Does one thing well
- Testable
- Self-contained

Traditional Way

React Way

```
<div >
   <h1>Main Address</h1>
   <p> MG Road </p>
   <p> Bangalore </p>
   <p> Karnataka </p>
</div>

<div >
   <h1>Branch Address</h1>
   <p> Anna Salai </p>
   <p> Chennai </p>
   <p> TN </p>
</div>
```

```
<Address
         address={mainAddress} />

<Address
         address={branchAddress} />
```

Composability

```
<Invoice>
   <Address
          address={invoiceAddress} />
</Invoice>
```

NODESEN.SE

React has **NO**

- No Controllers
- No Templates
- No directives (like Angular)
- No Globals (like jQuery)
- No models

Composing Components

Compose View

```
import React from “react";
import Header  from "./Header";
import Home  from “./Home";
import Footer  from “./Footer";

export class App extends React.Component {
    render() {
        return (
            <div>
                <Header ></Header>
                <Home ></Home>
                <Footer ></Footer>
            </div>
        )
    }
}
```

NODESEN.SE

Load React into Real DOM

```
import React from 'react';
import {render} from 'react-dom';

import {App} from "./App";

render(<App/>,
       document.getElementById('root'));
```

Component

```
export class App extends Component {
    render() {
        return (
            ....
        )
    }
}
```

render() is called
Multiple times,
not a one time function

May cause impact in performance
When poorly used,
We shall discuss soon..

React DOM Elements

```
import React, {Component} from "react";

export class Address extends Component {
    ...

    render() {
        return (
            <div>

                <h2>City</h2>

            </div>
        )
    }
}
```

div & h2 are element nodes

“City” is a text node

NODESEN.SE

React DOM Elements

```
import React, {Component} from "react";

export class Address extends Component {

    render() {
        let city = ‘Bengaluru’
        return (
            <div>
                <h2>City-{city}</h2>

            </div>
        )
    }
}
```

- React creates two text nodes, one for “City-” and
- Other text node for {city} rendered text
- On data change, it updates only changed text node {city}

NODESEN.SE

ReactDOM

React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.

Default Props

```
export class Address extends Component {

render() {
    return (
        <div>
            <span>City: {this.props.city}</span>
        </div>
    )
}
}

Address.defaultProps = {
    city: 'Bangalore'
}
```

Prop-Type

```
> npm  install prop-types  —save
```

Prop Types defines contract of props passed, data type, required properties

Prop-Type

```
import PropTypes from 'prop-types'; // ES6

export class Address extends Component {
}


Address.propTypes = {
  city: PropTypes.string.isRequired,
  state: PropTypes.string,
  pincode: PropTypes.number.isRequired
}
```

Prop Types

```
// An object taking on a particular properties
address: PropTypes.shape({
  city: PropTypes.string,
  state: PropTypes.number,
  pincode:  PropTypes.number
})

For function: PropTypes.func.isRequired
```

```
optionalEnum: PropTypes.oneOf(['News', 'Photos'])
```

NODESEN.SE

Data Flow

- Bad: No Framework (jQuery), any element/ component can communicate with any other by events
- Backbone: Pub-sub item.on(‘change:name’, function() {…})
- Angular 1.x - 2 Way Binding, $digest loop, $scope, ng-model

React: 1-way data flow

NODESEN.SE

Data Flow

- Data handed from parent to child
- Using HTML styled attributes
- “props” in React (aka, properties)

Props

- One way Data Flow: Parent to Child

Props are immutable

Props in Constructor

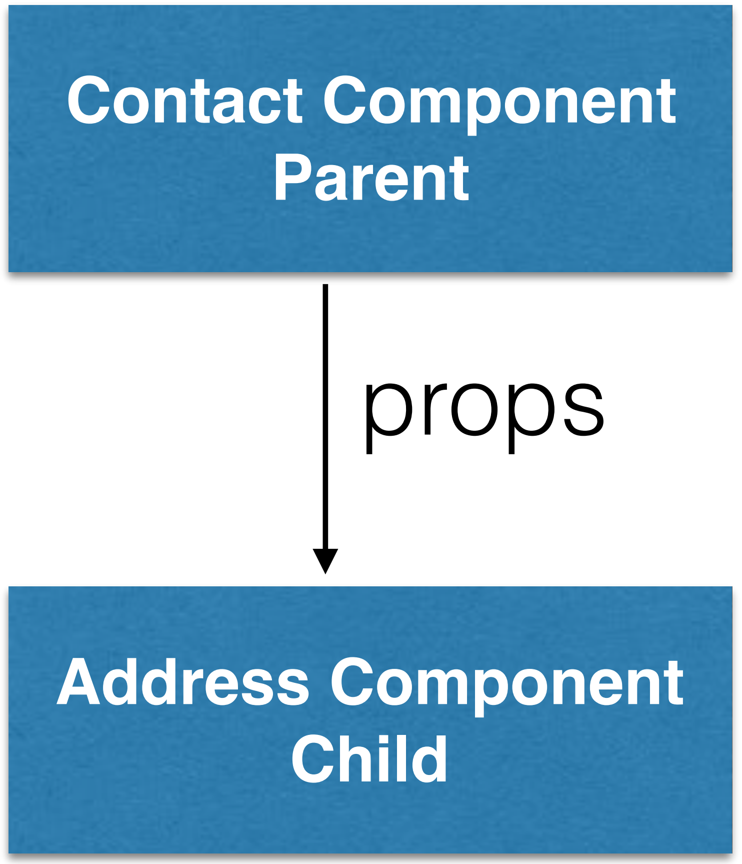
```
import React, {Component} from "react";

export class Address extends Component {

    constructor(props) {
        super(props);
        //initialise component with props
    }

    render() {
        return (
            <div>
                <h2>City-{this.props.title}</h2>
            </div>
        )
    }
}
```

Constructor receives props

NODESEN.SE

```
class Contact extends Component({
   ...
   render() {
      return (
         <div>
            <Address city="Bangalore" state="KA" />
         </div>
      )
   }
})
```

```
class Address extends Component({
   ...
   render() {
      return (
         <div>
            <span> {this.props.city}</span>

            <span> {this.props.state} </span>
         </div>
      )
   }
})
```



NODESEN.SE

Props

- Props are immutable
- i.e. Treat them as read only constants always
- Your component doesn’t own the props
- Props are owned by parent

State is mutable

```
class Home extends React.Component({
    …
    componentDidMount  () {
        this.setState({ likes: 100})
    },

    render() {
        return <Like   likes={this.state.likes} />
    }
})
```

- State is local to component
- State is mutable, can be changed

State can become props/input to child component

```
class RegistrationComponent extends React.Component({
    ...
    render() {
        return (
            <div>
              <TermsComponent  active={this.state.active} />
            </div>
        )
    }
})

class TermsComponent extends React.Component({
    ...
    render() {
        return <input type="checkbox" checked={this.props.active} />
    }
})
```

NODESEN.SE

State APIs

- **Initialisation :** on the constructor, this.state = {show: true}
- **component.setState(nextState, callback)** - set the next state, while merging next state with existing state, after setting new state, calls **render** to update the UI

- **component. forceUpdate(callback),** calls the **render** method, should be avoided, instead use props or setState to update the state and trigger the render to be called

$$\begin{tikzcd}\text{state} \arrow[r] & \text{describe}\end{tikzcd}$$

```
render() {
     …
   <h2>Likes: {this.state.likes} </h2>
   <button  onClick={this.increment}>
      +
    </button>
    …
}
```

for every change of state $\longrightarrow$ describe *the whole user interface*

```
increment(event) {
   this.setState({
     likes: this.state.likes + 1
    })
}
```

NODESEN.SE

render method

- Initial Component Life cycle [constructor, componentWillMount, render(), …]
- Render is called by **forceUpdate**() with in component
- By calling **setState** API with in component
- Or any time there is props change on the parent

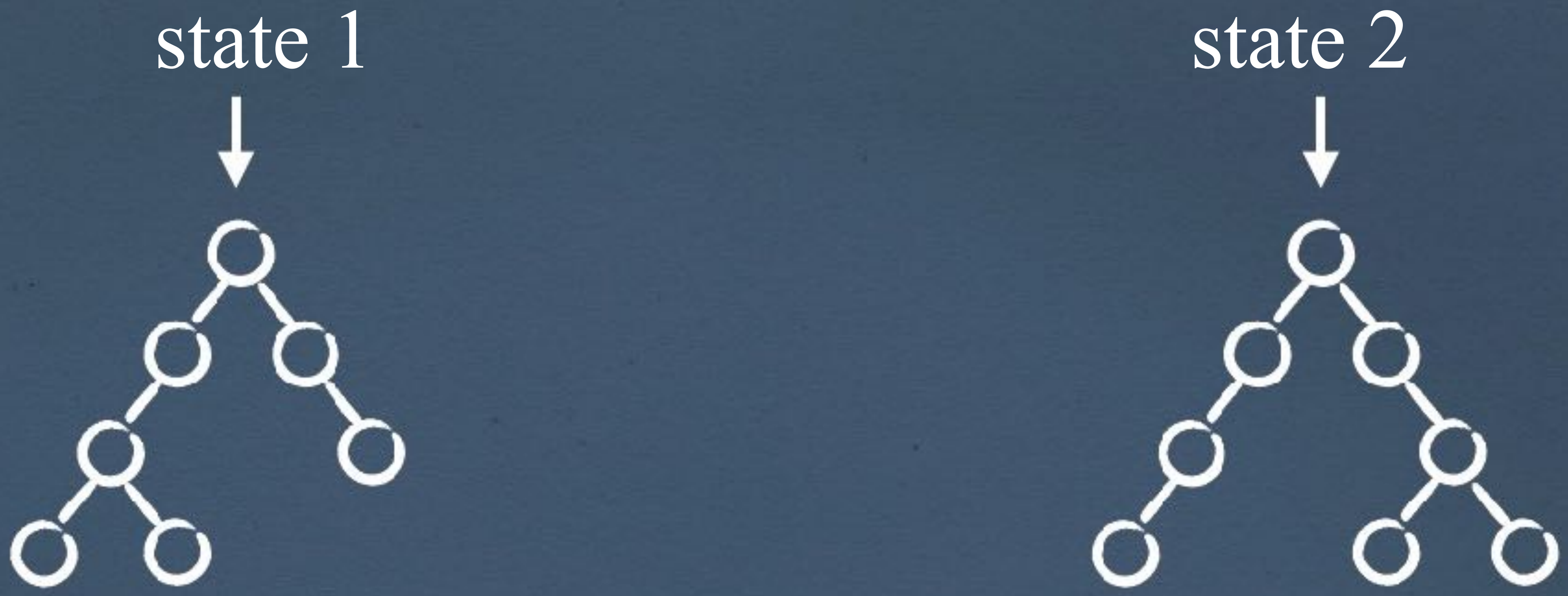
for every change of State & Props

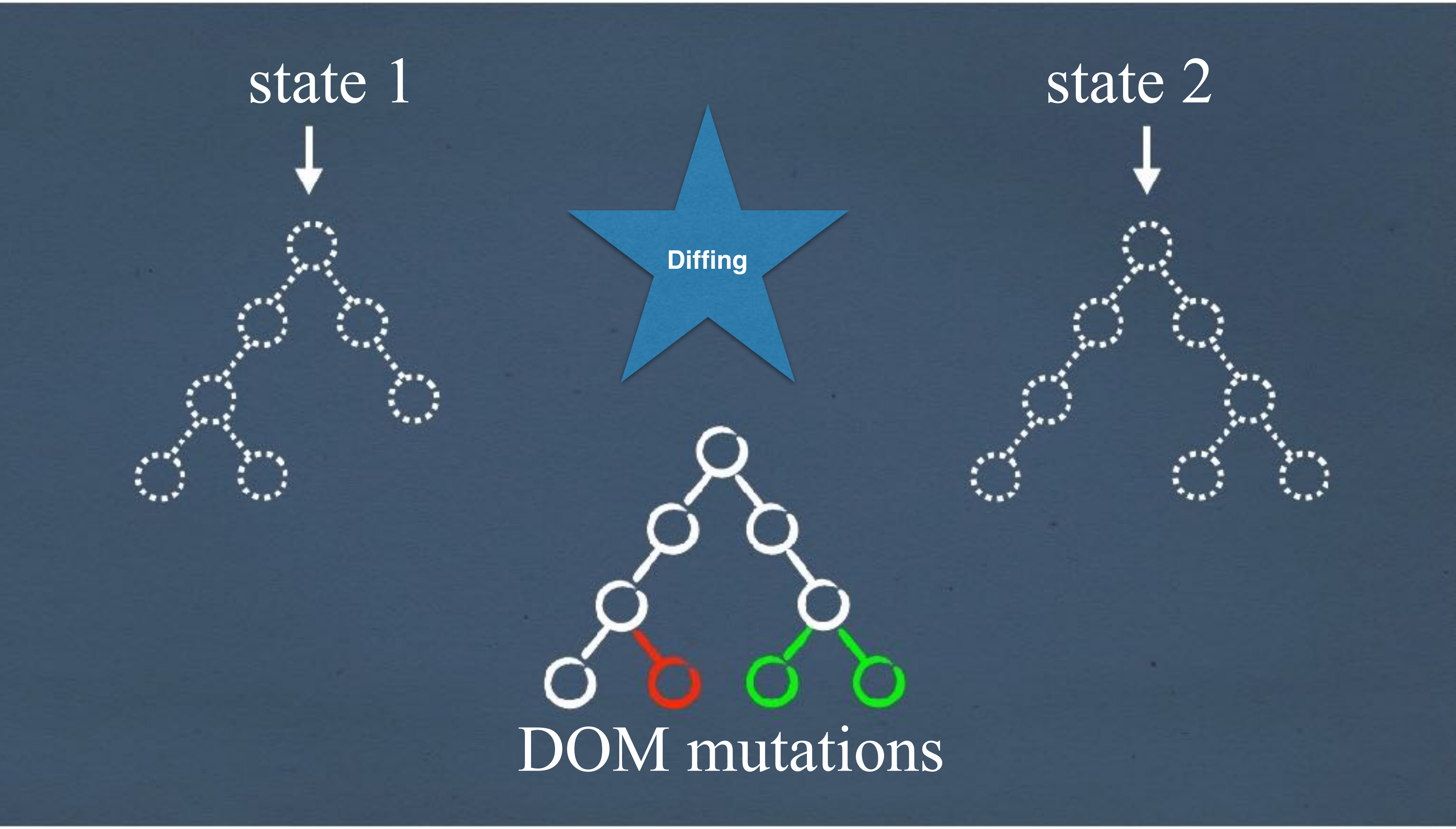
describe *the whole user interface*

React calls render() method for every state change
The Elements returned from render() first **updated on Virtual DOM, not REAL DOM**

React uses diffing algorithms to sync changed DOM nodes between Virtual and Real DOM

NODESEN.SE


state 1
state 2



NODESEN.SE

Props vs State

Props

passed in from parent

```
<Address   city="Bangalore"   />
```

props should be /immutable or
read-only within child component

State

created, managed within component

```
constructor() {
  this.state = {
          products: [],
          loading: false
   }
 }
```

this.state to read

```
this.setState({loading: true})
```

to update date

NODESEN.SE

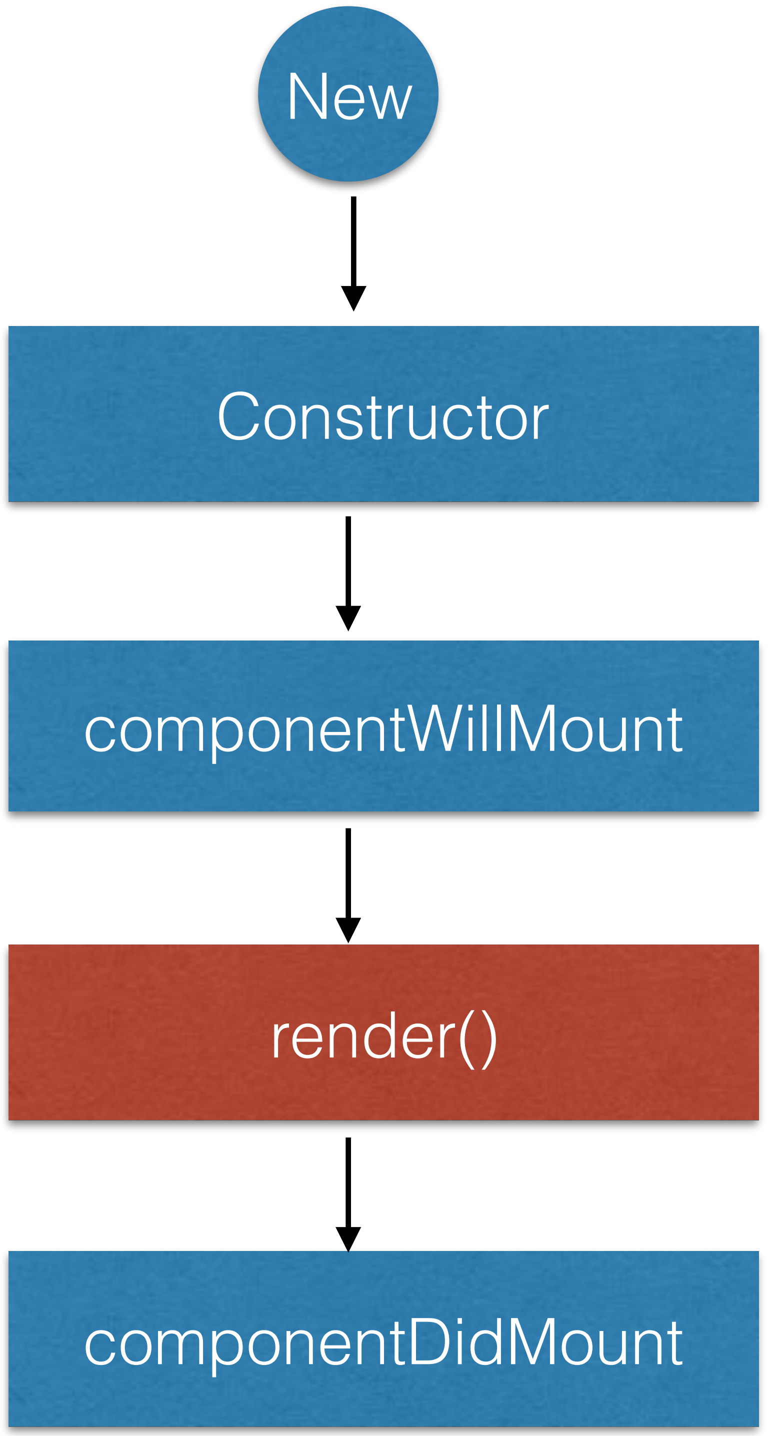
Component Lifecycle

Life cycles

- Creation
- Destruction
- Update

Creation Life Cycle

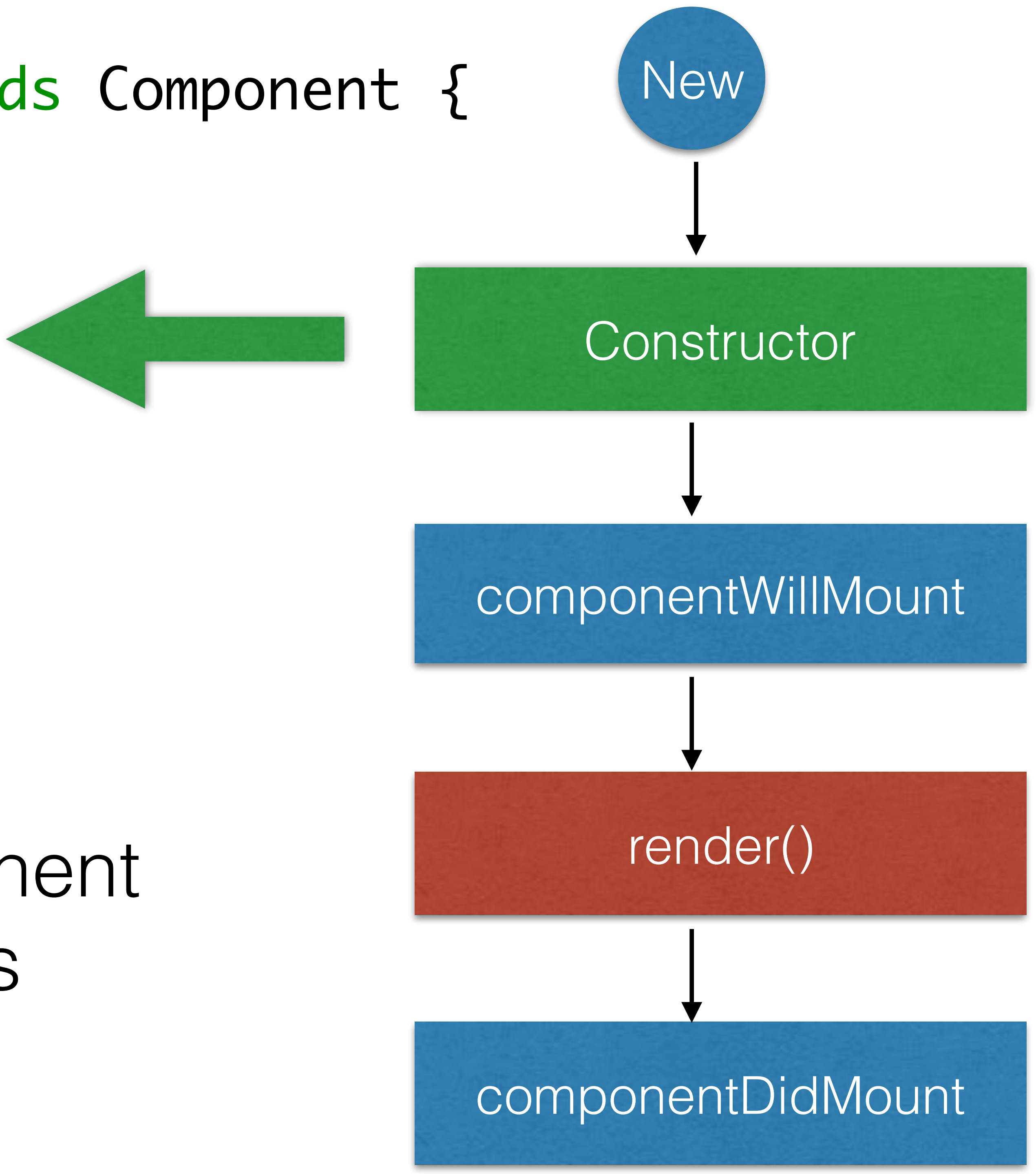
```
export class Like extends Component {
     //View not ready
     constructor(props) {
     }
    //View not ready
    componentWillMount() {
    }
     //returns view
    render() {
         return (…)
    }

    //view ready
    componentDidMount() {
    }
}
```



NODESEN.SE

Constructor

```
export class CartItem extends Component {
    //View not ready
    constructor(props) {
        this.state = {
            qty: props.qty;
        }
    }
}
```



Dos

- Initialise state of component
- Initialise state with props

Don'ts

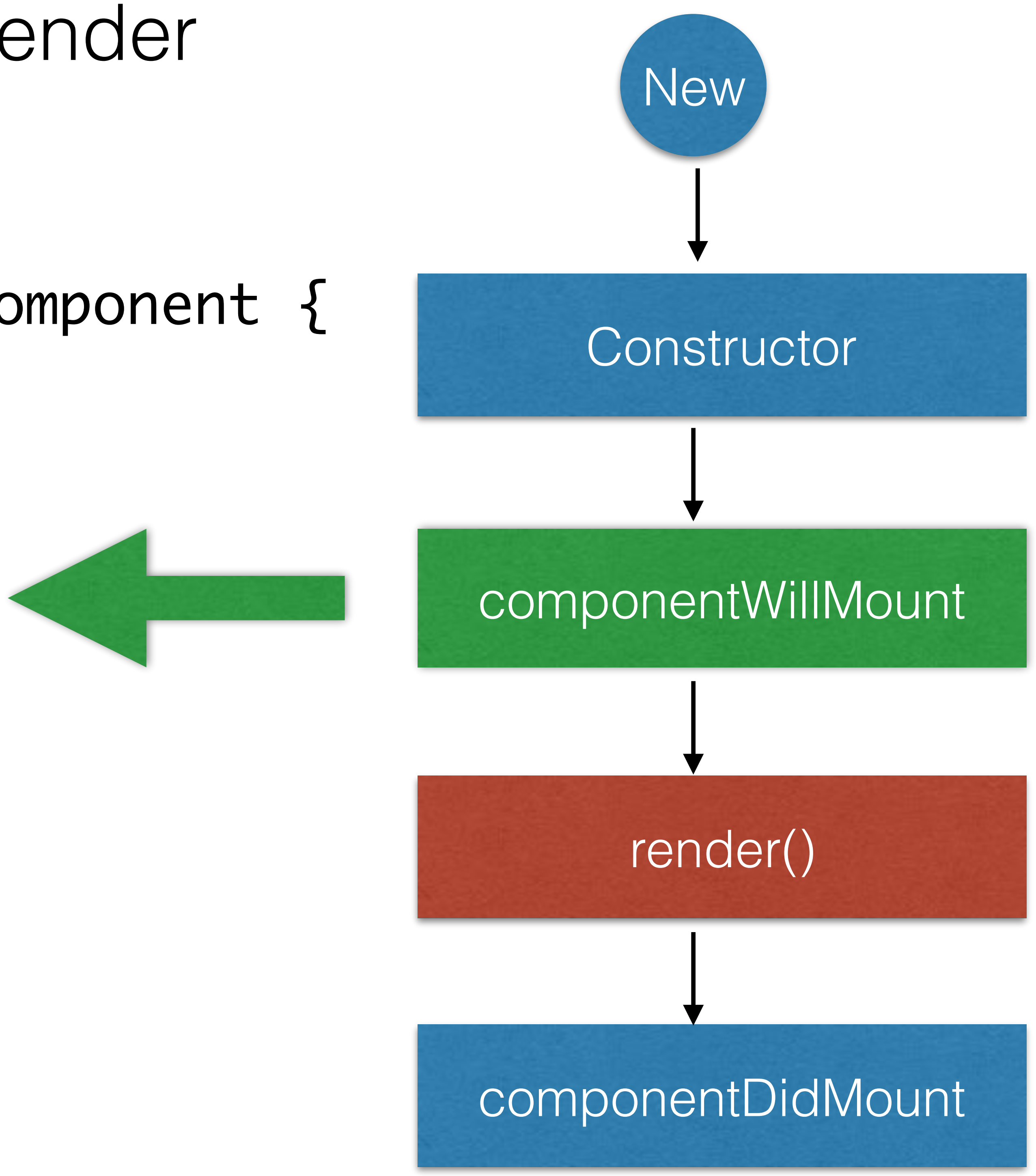
- Access the view
- Calling server APIs

NODESEN.SE

componentWillMount

- Called just before first render
- View is not yet ready

```
export class Cart extends Component {

   //View is not yet ready
   componentWillMount() {
      this.state = {
          showList: true,
          amount: 0
      }
   }
}
```



NODESEN.SE

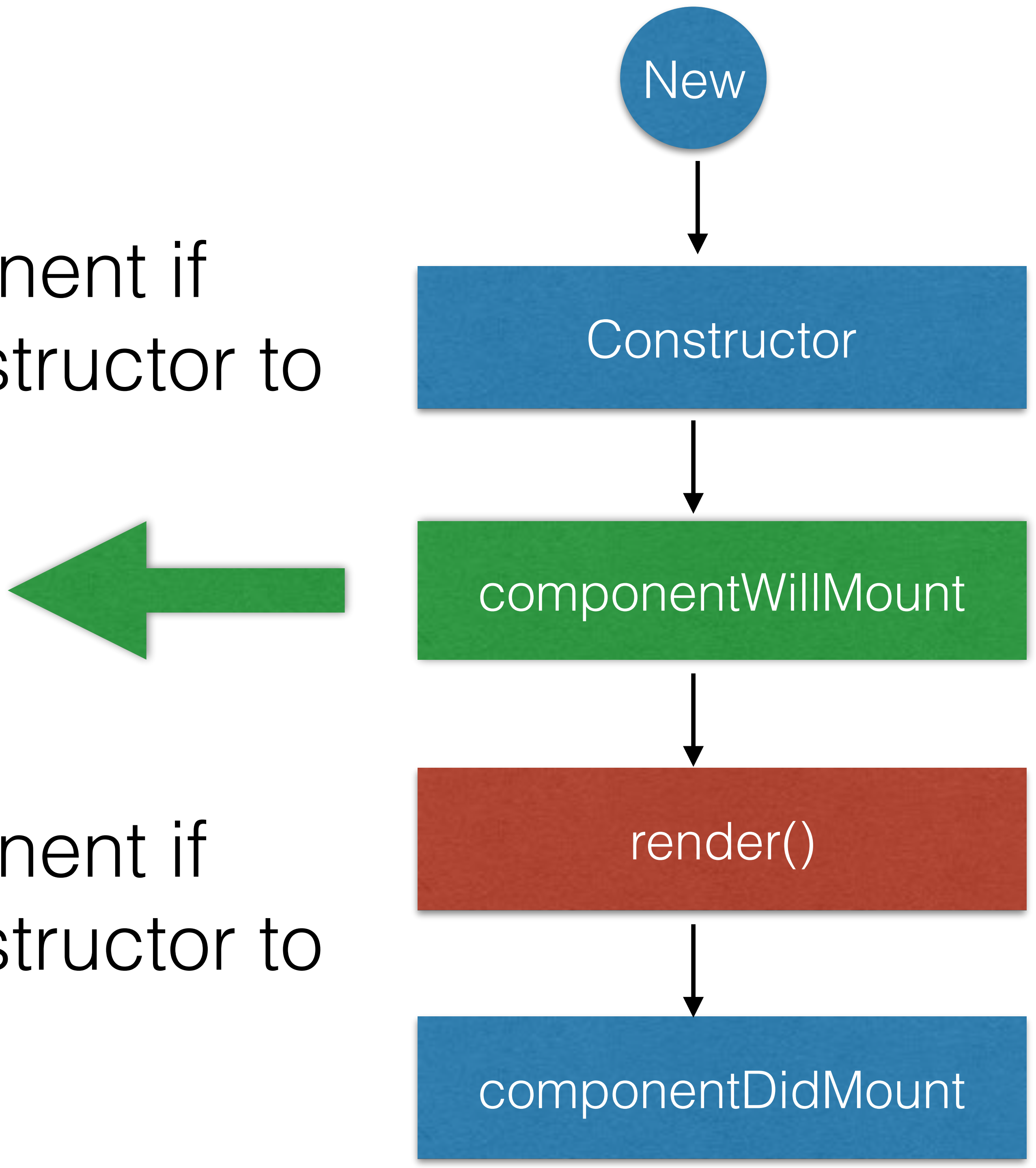
componentWillMount

Dos

Initialise state of the component if you don’t want to write constructor to initialise state

Dont’s

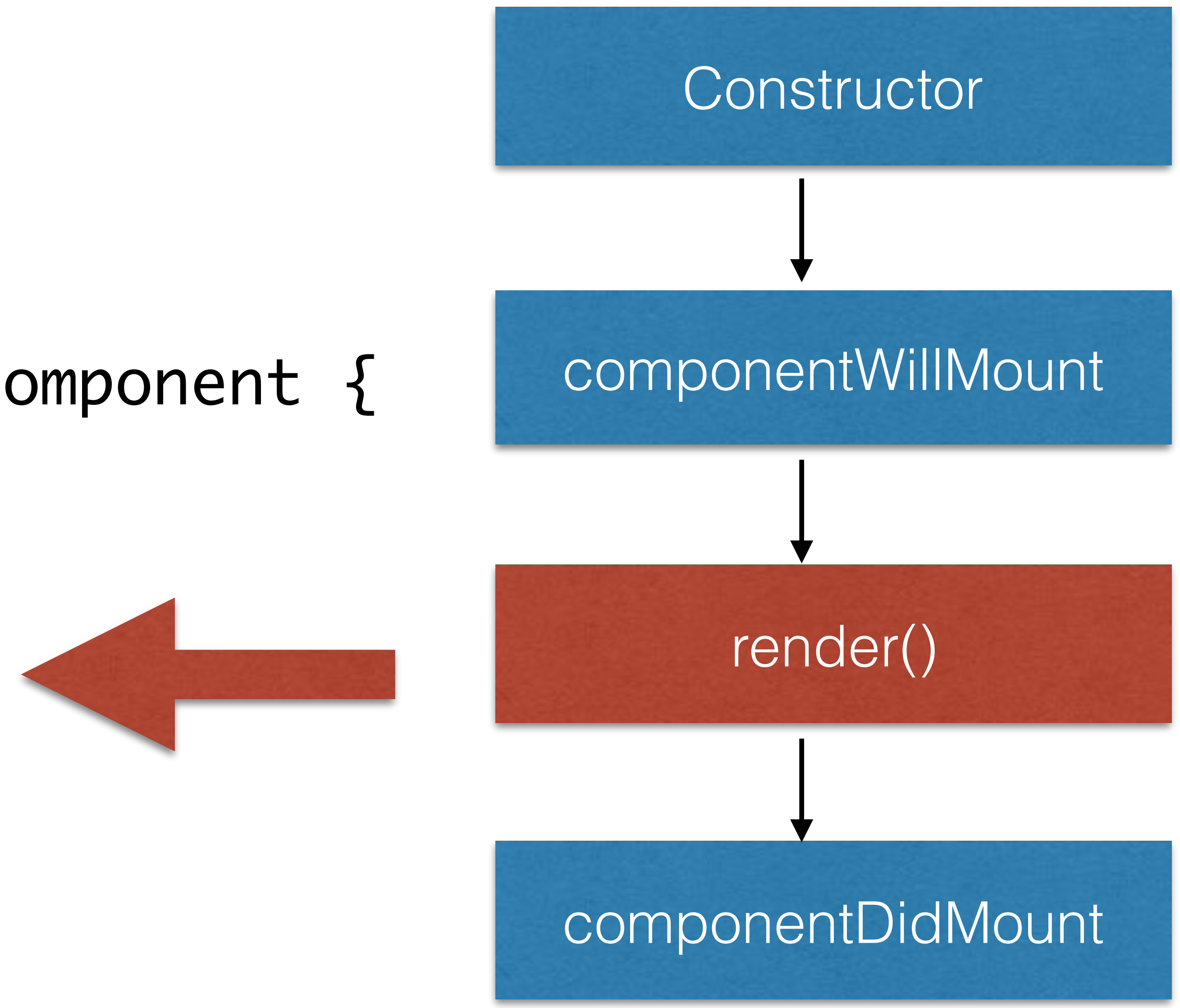
Initialise state of the component if you don’t want to write constructor to initialise state



Render

Called one time during creation life cycle

```
export class Cart extends Component {

     //returns view
    render() {
         return (…)
    }

}
```

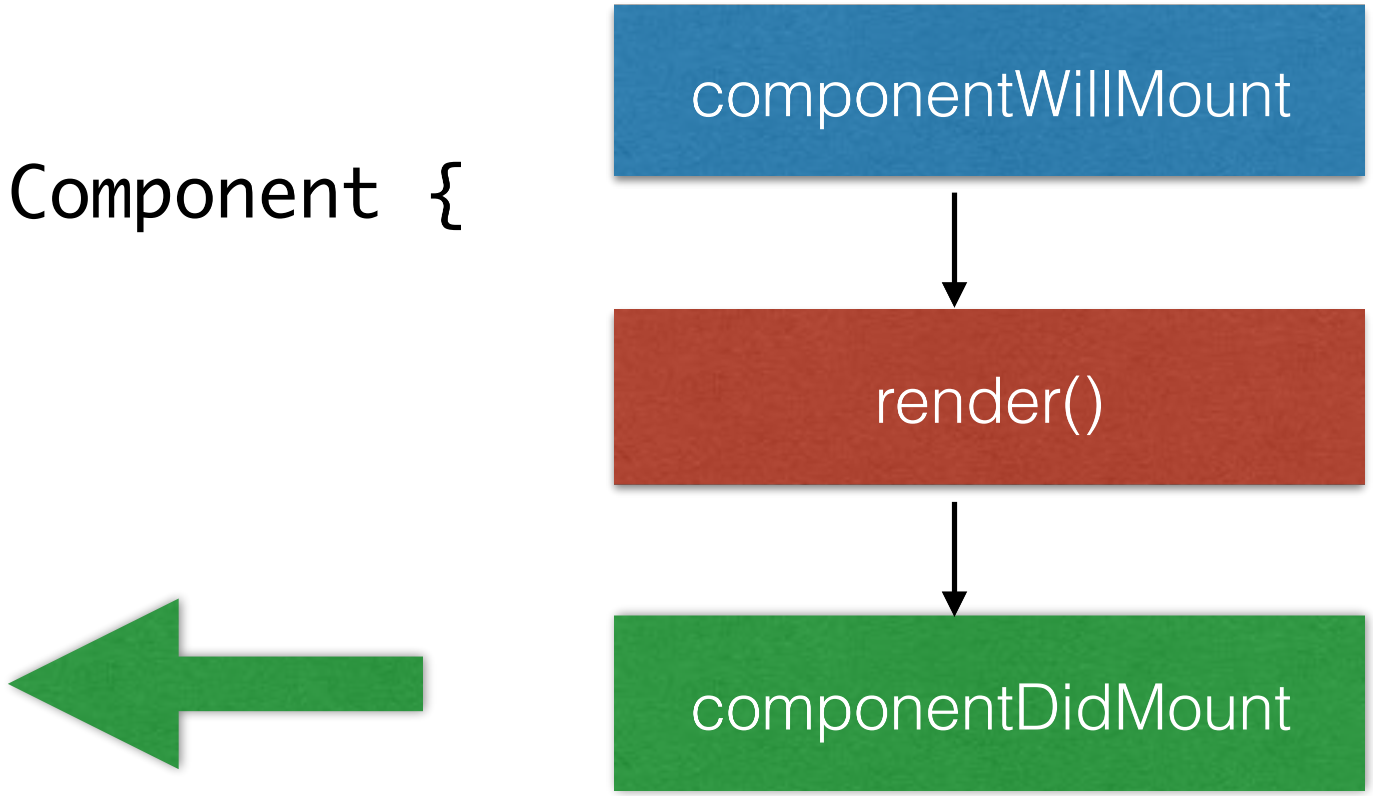


Called multiple times during update cycle

NODESEN.SE

componentDidMount

```
export class CartItem extends Component {
    //View is ready
    componentDidMount() {


    }

}
```



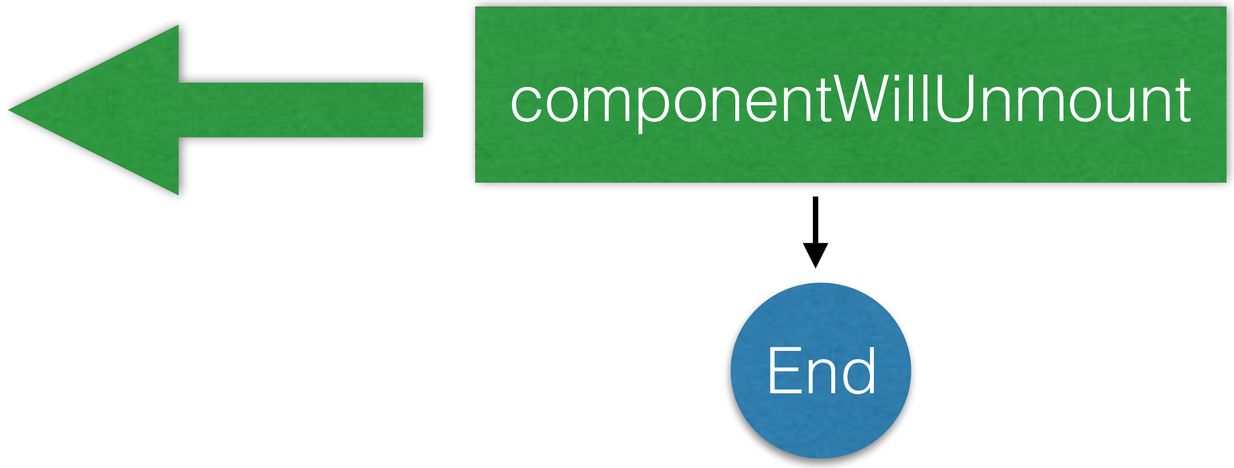
Dos

- Call setStates
- Access View elements
- Initiate AJAX Calls

NODESEN.SE

Destruction Life Cycle

```
export class Cart extends Component {

    //component/view shall be removed
    componentWillUnmount() {


    }
}
```



Dos

- Clean/stop setInterval, setTimeOut
- Unsubscribe from subscription

Dont's

- Access Views
- Calling setState

NODESEN.SE

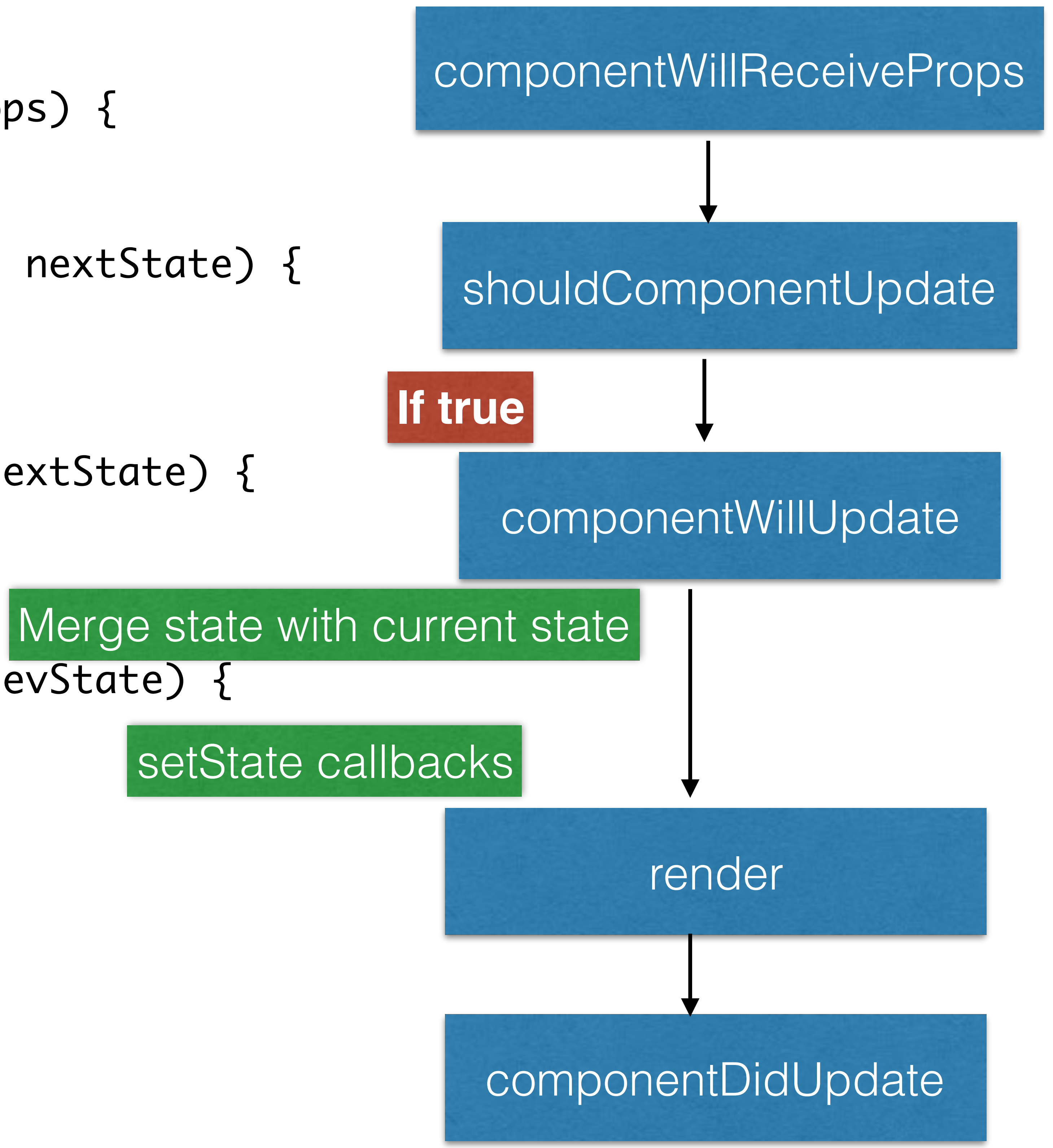
Update Life Cycle

- constructor (props)
- componentWillReceiveProps(nextProps)
- shouldComponentUpdate(nextProps, nextState)
- componentWillUpdate(nextProps, nextState)
- componentDidUpdate(prevProps, prevState)

NODESEN.SE

Update Cycle

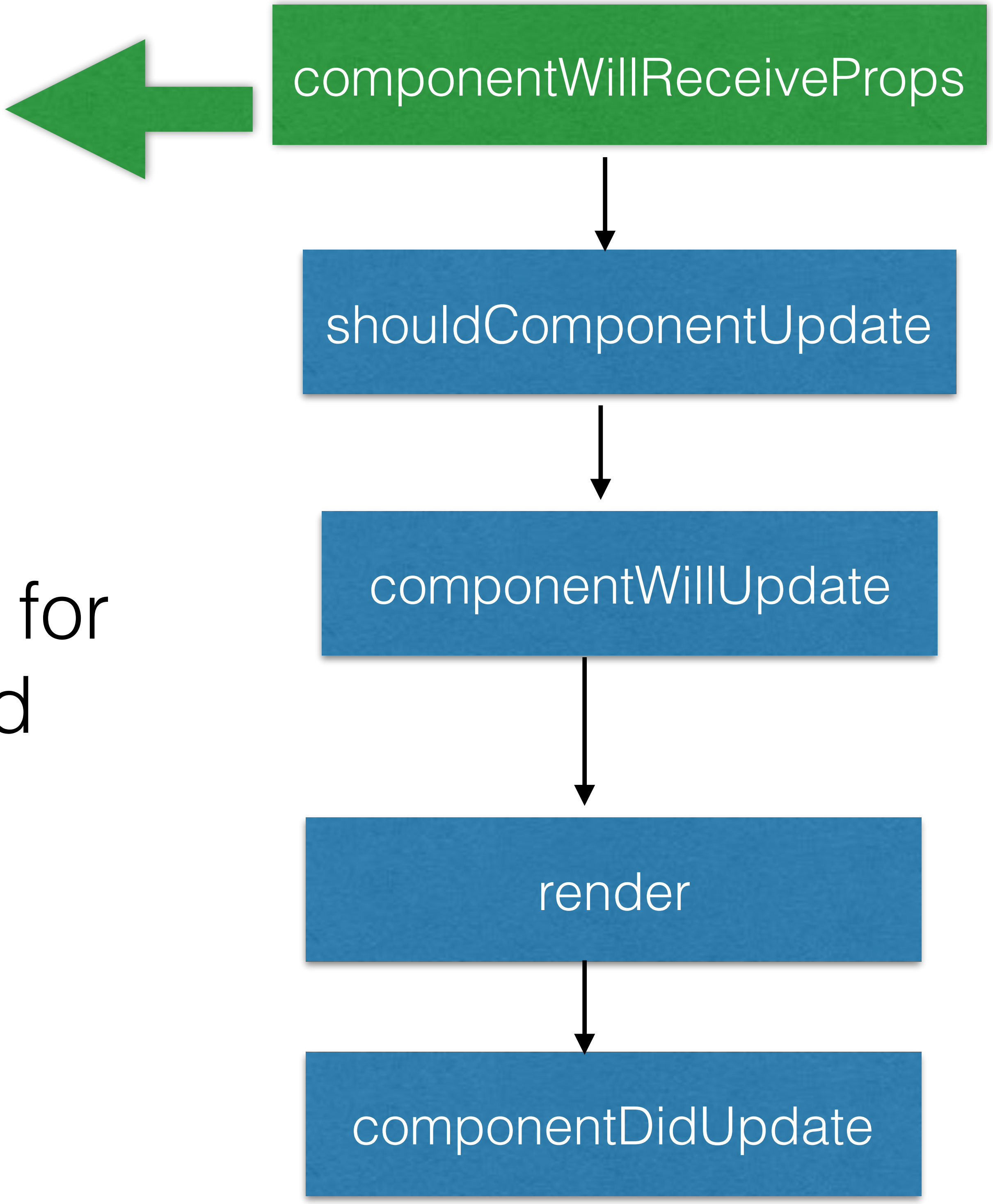
```
class Like extends Component {
     constructor(props) {
         super(props);
     }
    componentWillReceiveProps(nextProps) {

     }
     shouldComponentUpdate(nextProps, nextState) {
         return true; //or false
     }

     componentWillUpdate(nextProps, nextState) {

     }

     componentDidUpdate(prevProps, prevState) {

     }

     render() {
         return (...);
     }
 }
```



NODESEN.SE

Update Cycle

```
class CartItem extends Component {

    componentWillReceiveProps(nextProps) {
      this.setState({
         qty: nextProps.item.qty
      })
    }

 }
```

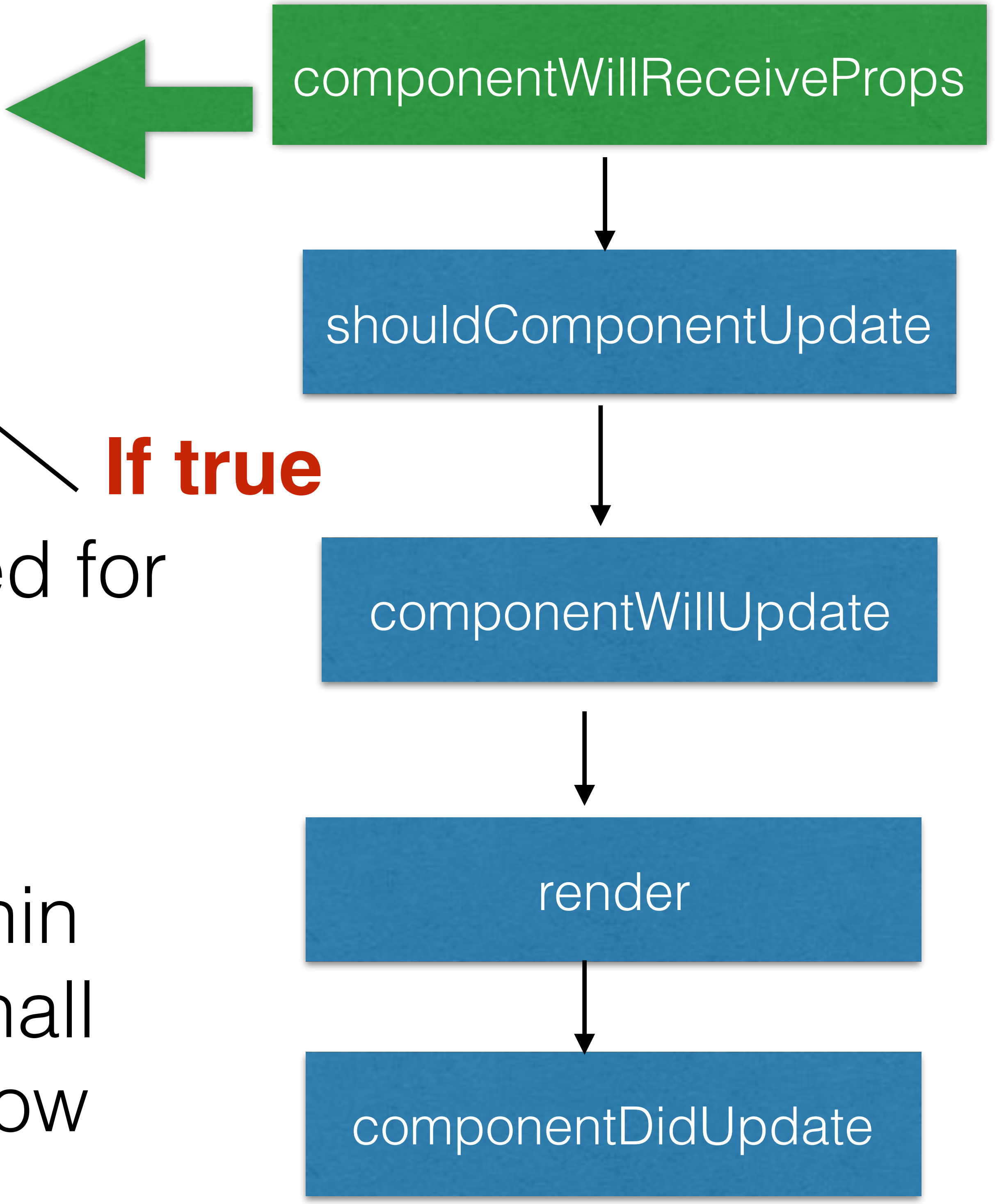


componentWillReceiveProps for **every parent render** method

NODESEN.SE

Update Cycle

```
class CartItem extends Component {

    shouldComponentUpdate(nextProps, nextState) {
        if (nextProps.item != props.item)
        return true;

        return false;

    }

}
```



shouldComponentUpdate called for **every parent render, this.setState** method

SHOULD NOT call **setState** within shouldComponentUpdate, that shall cause recursion, call stack overflow

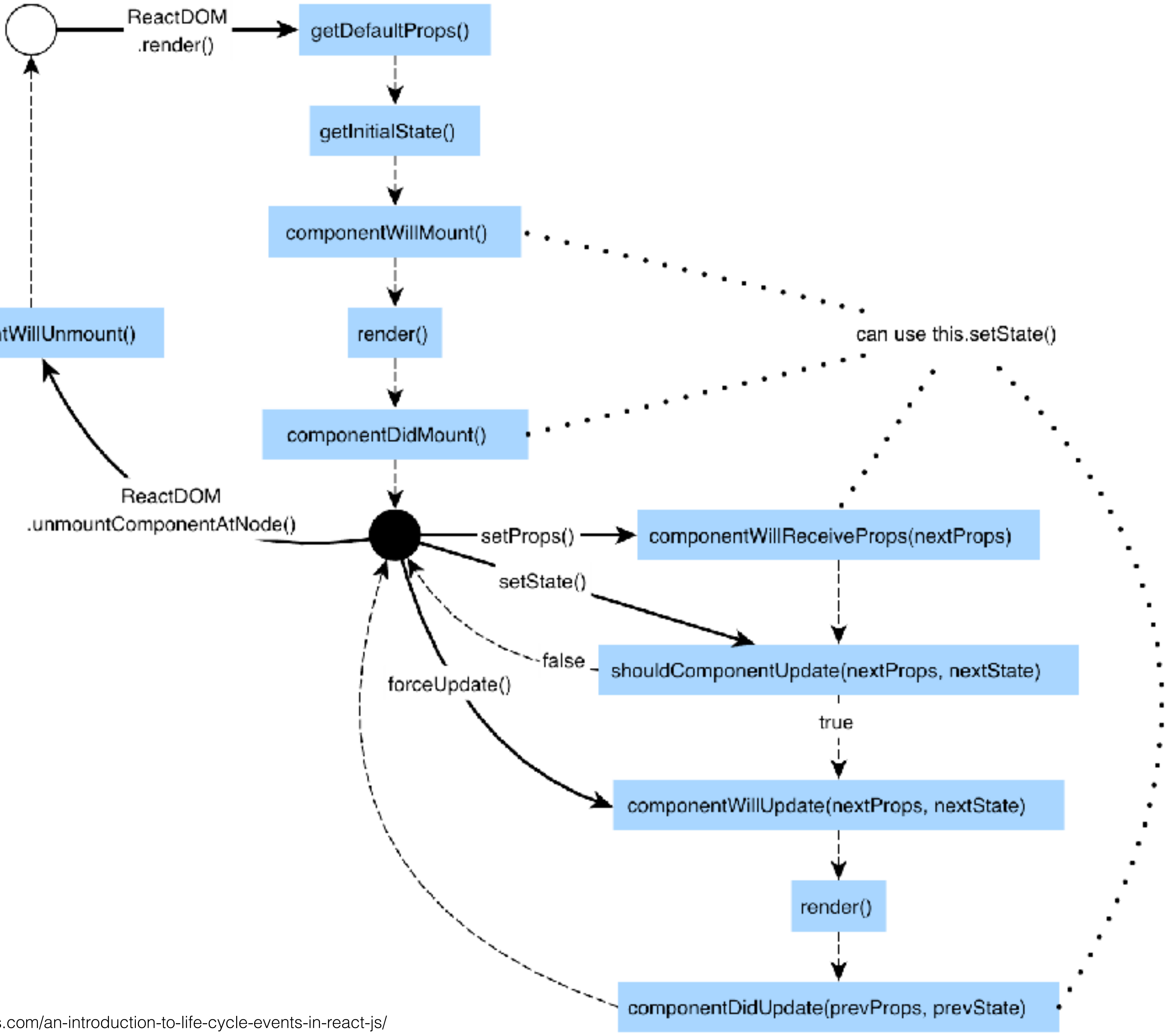
NODESEN.SE

this.setState

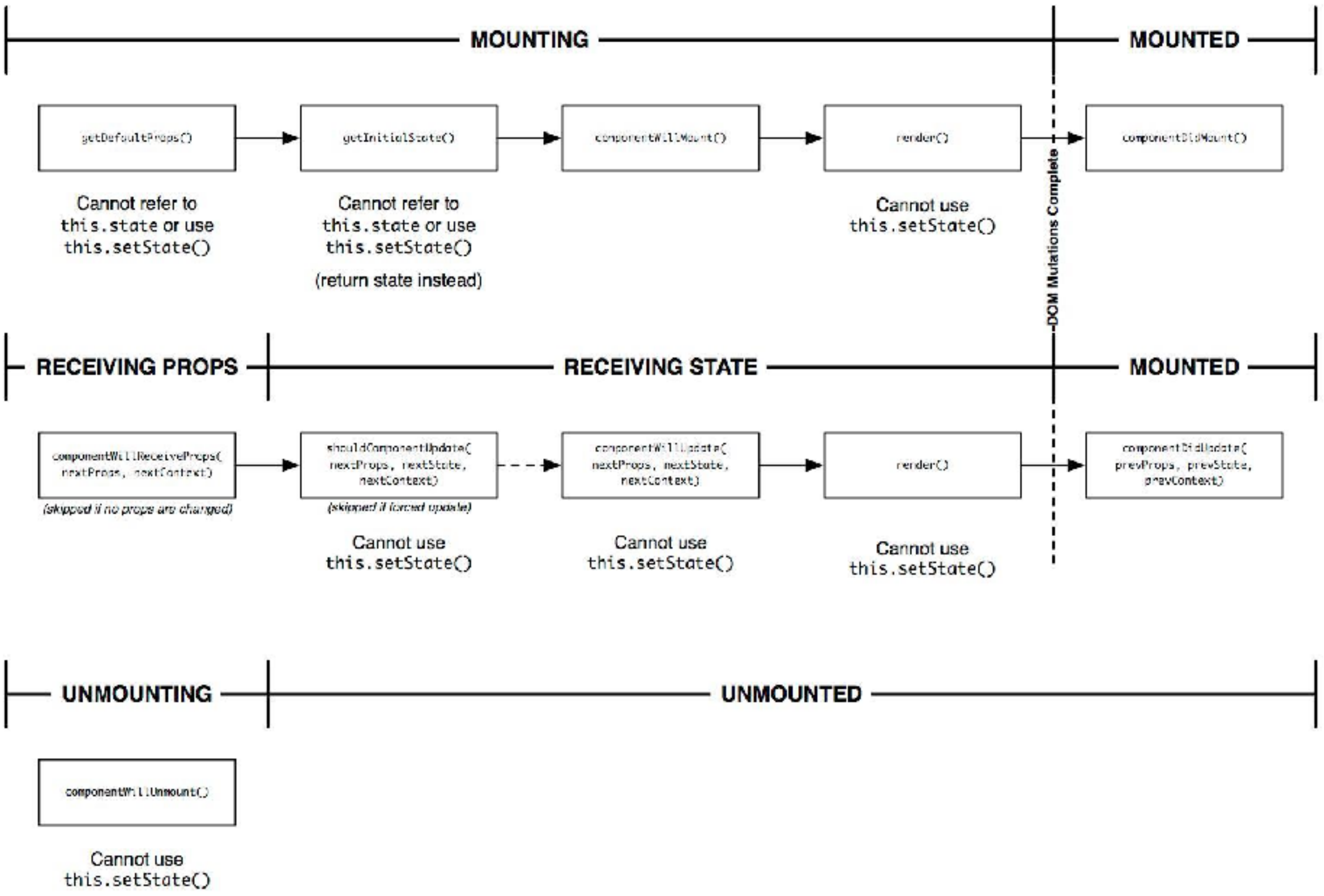
- setState calls shouldComponentUpdate first
- shouldComponentUpdate decides whether render to be called or not

this.forceUpdate

- forceUpdate calls render method directly
- forceUpdate DO NOT call shouldComponentUpdate


ReactDOM .render()
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
componentWillUnmount()
can use this.setState()
ReactDOM .unmountComponentAtNode()
setProps()
componentWillReceiveProps(nextProps)
setState()
false
shouldComponentUpdate(nextProps, nextState)
forceUpdate()
true
componentWillUpdate(nextProps, nextState)
render()
componentDidUpdate(prevProps, prevState)

https://tylermcginnis.com/an-introduction-to-life-cycle-events-in-react-js/

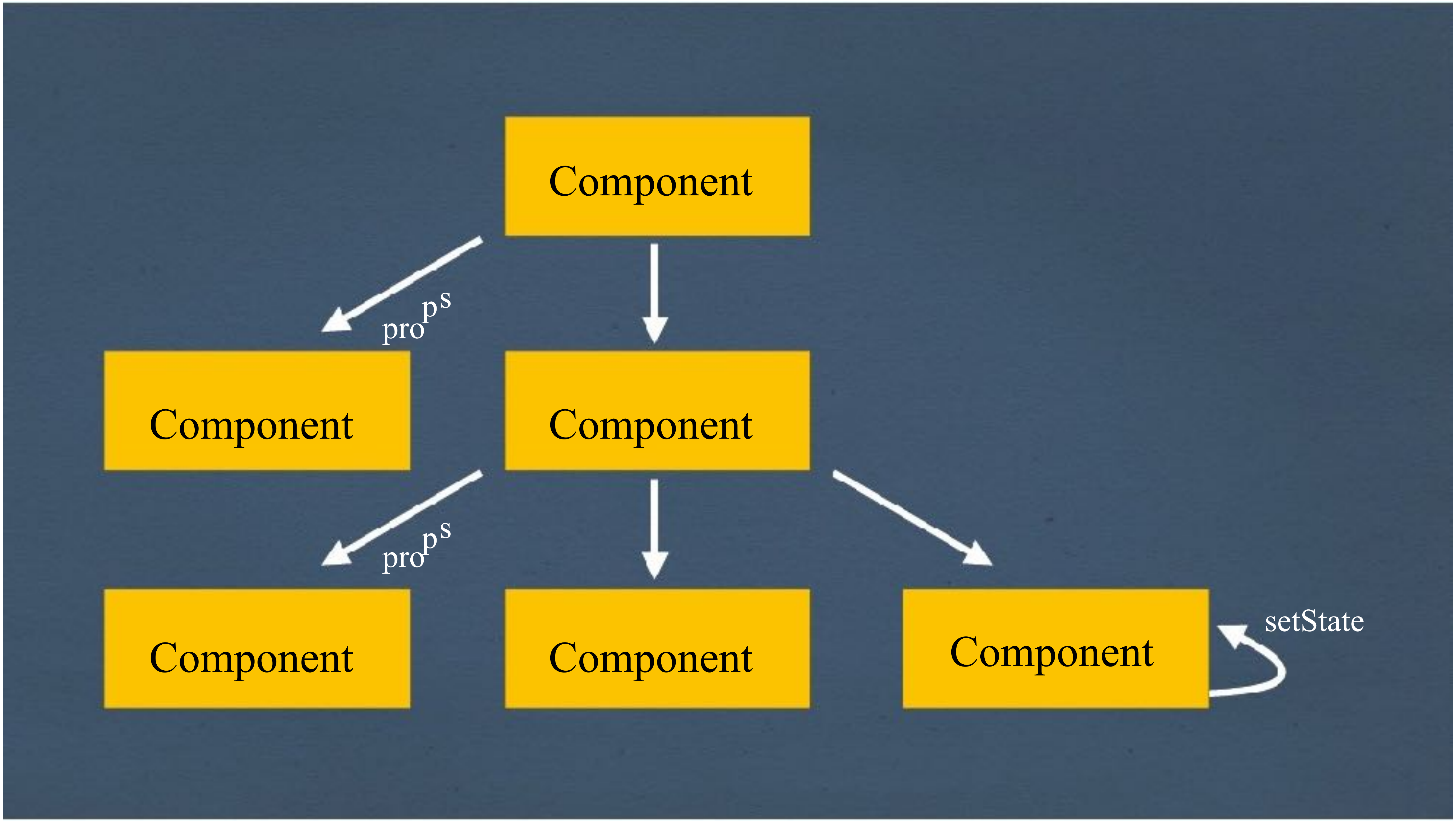

MOUNTING
MOUNTED
getDefaultProps()
getInitialState()
componentWillMount()
render()
componentDidMount()
Cannot refer to this.state or use this.setState()
Cannot refer to this.state or use this.setState()
(return state instead)
Cannot use this.setState()
DOM Mutations Complete
RECEIVING PROPS
RECEIVING STATE
MOUNTED
componentWillReceiveProps(nextProps, nextContext)
(skipped if no props are changed)
shouldComponentUpdate(nextProps, nextState, nextContext)
(skipped if forced update)
componentWillUpdate(nextProps, nextState, nextContext)
render()
componentDidUpdate(prevProps, prevState, prevContext)
Cannot use this.setState()
Cannot use this.setState()
Cannot use this.setState()
UNMOUNTING
UNMOUNTED
componentWillUnmount()
Cannot use this.setState()

https://staminaloops.github.io/undefinedisnotafunction/images/react-lifecycle.jpg

Data Flow

- Data Flow One way
- Props => Parent to Child
- setStates => Within component
- How about forms, input controls?

NODESEN.SE


Component
props
Component
Component
props
Component
Component
Component
setState

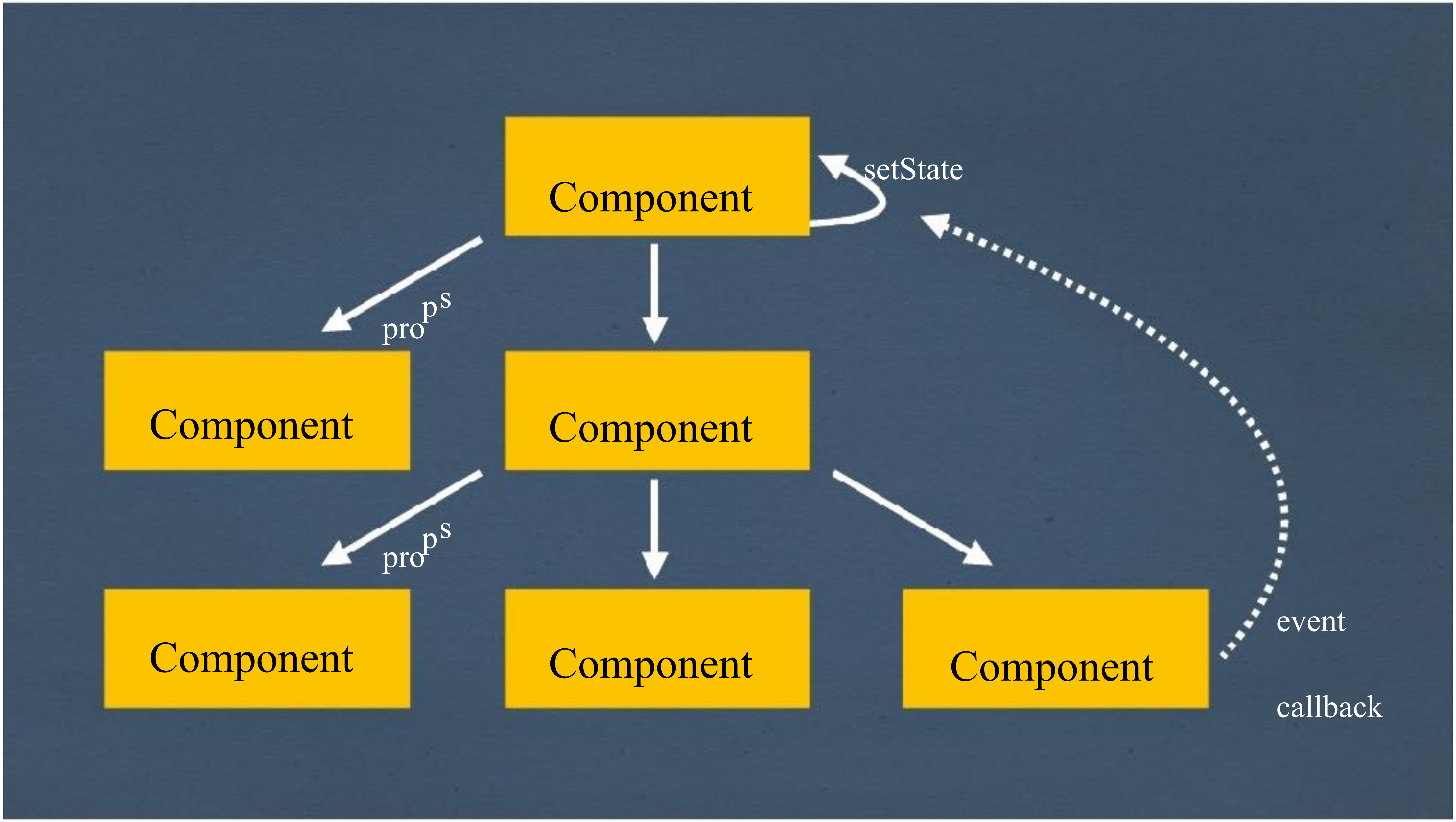
NODESEN.SE

Can Data Flow up?

- Data flows down with handlers
- Other way, we pass function reference from parent to child
- Child call the function, that is defined in parent

Data Flow

- Parent to Child via **props**
- Child to Parent via **handler** function
- Parent should pass handler function to child as props


Component
setState
props
Component
Component
props
Component
Component
Component
event
callback

Child to Parent

Parent

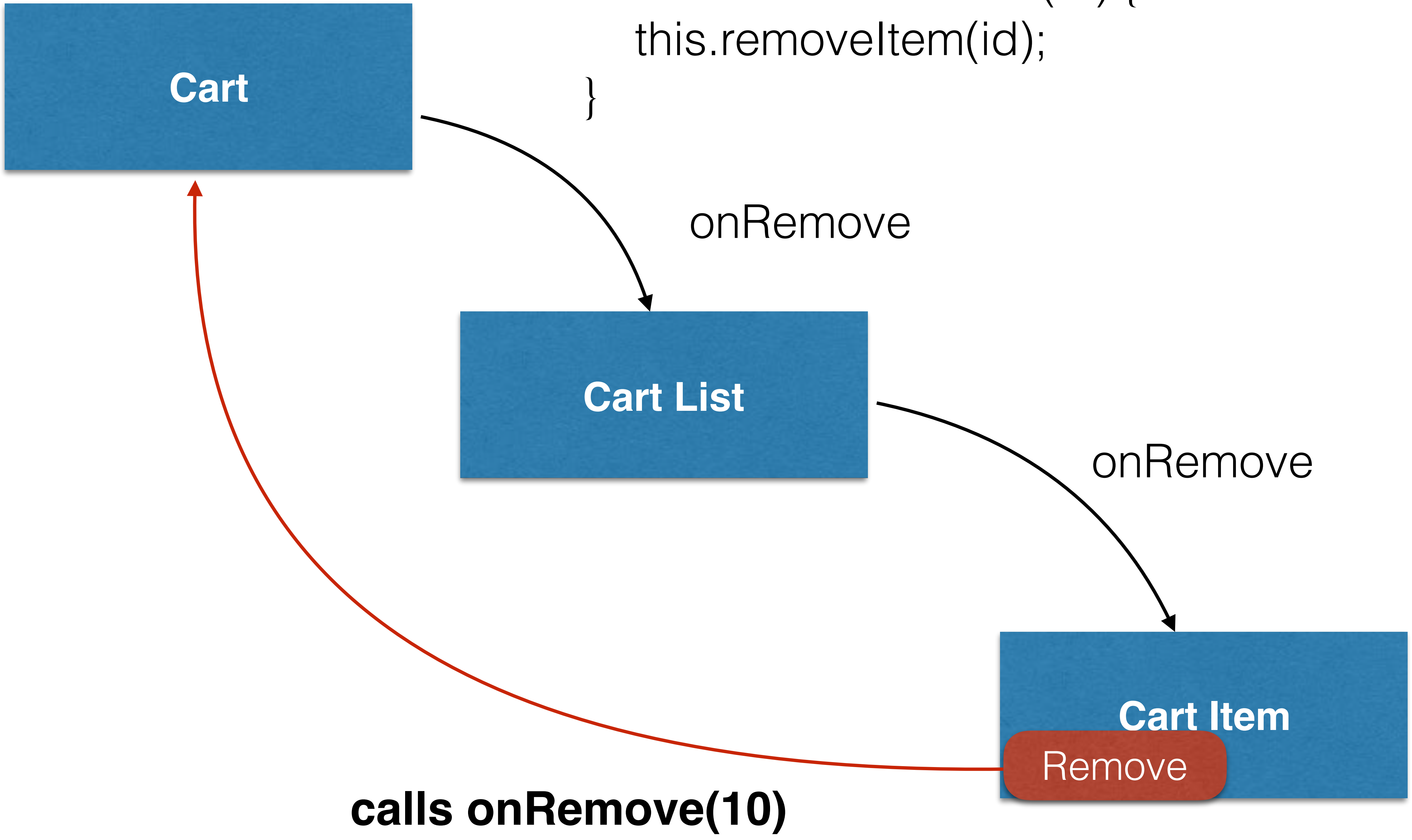
```
class About extends React.Component {

likesChanged(newValue) {
}

render() {
  return (
    <Like likes={this.state.likes}
       onLikeChange={(value}=>this.likesChanged(value)}
  />)
}
```

Child

```
<button onClick={()=> this.props.onLikeChange(likes + 1)}>
            +
</button>
```

NODESEN.SE

```
onRemove = function (id) {
    this.removeItem(id);
}
```



Functional Component

- Function only component
- No Class
- No State
- No Lifecycle methods

```jsx
import React from "react";
import PropTypes from "prop-types";

export function Footer(props) {
    return (
        <div className="footer">
            <hr />
            <span>Copyrights</span>
            <p>@{props.year}, {props.company}</p>
        </div>
    )
}

Footer.defaultProps = {
    year : 2017,
    company: 'NodeSense'
}

Footer.propTypes = {
    year: PropTypes.number.isRequired,
    company: PropTypes.string
}
```

Pure Component

Pure Component

- provides shouldComponentUpdate method implementation
- Allow render to be called only if any changes in props or state items, i.e. shallow compare

```
shouldComponentUpdate(nextProps, nextState) {
        return true; //or false
    }
```

```
import React, {PureComponent} from “react";

export default class LiveWeather extends PureComponent {
   constructor(props) {
        super(props);
   }
   render() {
        return (
            <div>
                <h2>Temp - {this.state.temp}</h2>
            </div>
        )
    }
    //Not needed
    //as PureComponent already provides shouldComponentUpdate
    // shouldComponentUpdate(nextProps, nextState) {
    //     if (state.temp != nextState.temp)
    //         return true;
    //     return false;
    // }
}
```

NODESEN.SE

Refs

- **Helps to access Real DOM Object [not to be overused]**
- **Access child component methods and properties [not to be overused]**

Refs

```
...
componentDidMount()
{
    this.textInput.focus();
    this.elem.textContent = "Paragraph Text"
}
...
render() {
    return ( <div>
        <input
          type="text"
          ref={(input) => { this.textInput = input; }} />
         <p ref={(elem) => {this.elem = input; }} >
            </p>
        </div>
    )
}
```

NODESEN.SE

Refs & Custom Component

https://facebook.github.io/react/docs/refs-and-the-dom.html

```
<Like count={this.state.homeLikes}
        title="Home Likes"
        ref = { (elem) => this.like = elem}
        >
</Like>
```

Context

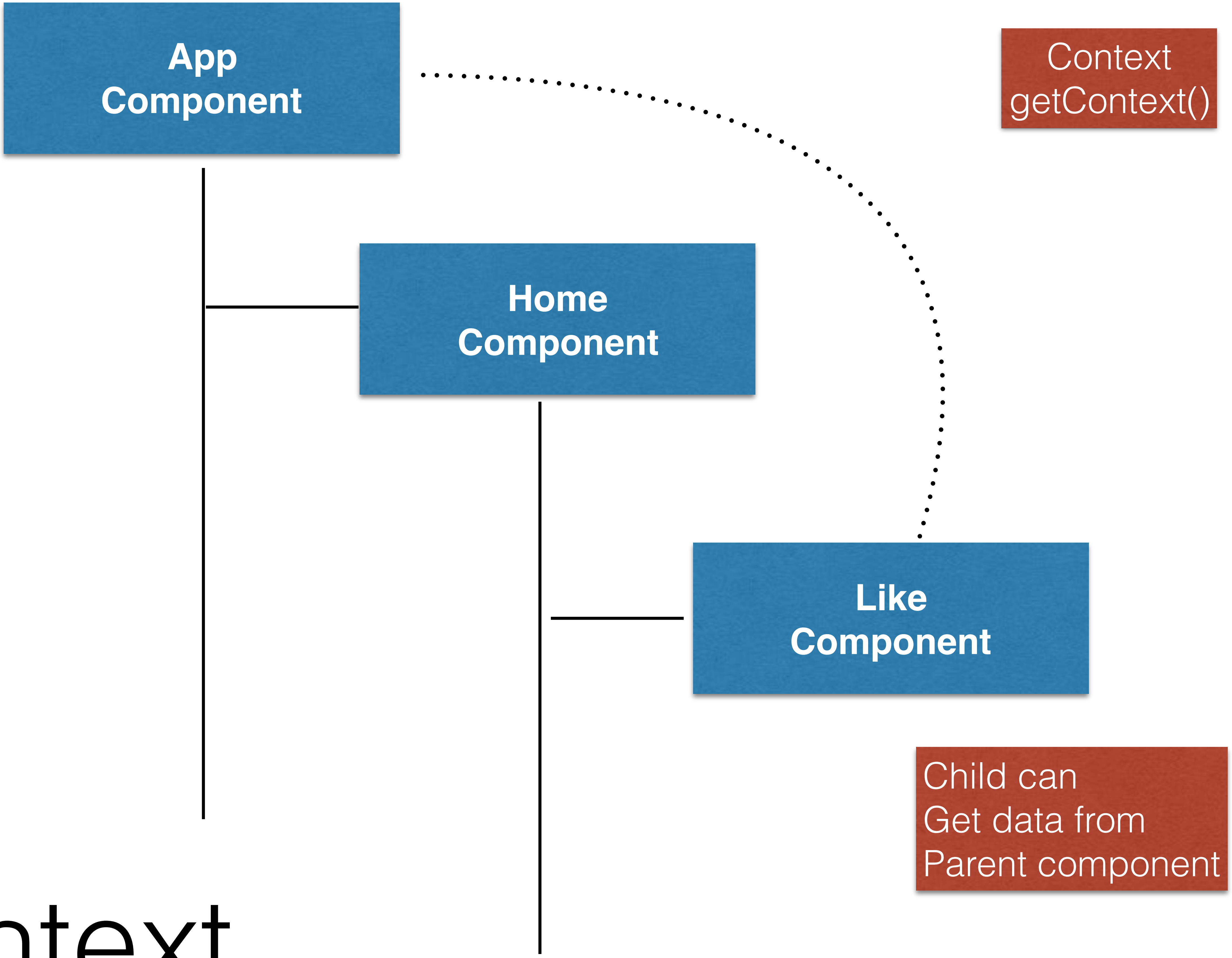
A Hack to pass data from parent to ’N’ level child

React Context

- Helps to pass data from parent to any child at any level [nested]

- **React Team recommends not to over use context, NOT TO UPDATE CONTEXT VARIABLES**

- Useful to pass an global object/variable on through context within react like app title, redux store



Context

NODESEN.SE

Parent Component

```
import PropTypes from "prop-types"

export class App extends Component {
   getChildContext() {
        return {
               color: "purple",
               name: "Product App"
           };
    }

    render() {
        return <Home homeLikes={this.state.homeLikes}>
               </Home>
    }
}

App.childContextTypes = {
  color: PropTypes.string,
  name: PropTypes.string
};
```

Context

NODESEN.SE

```
import PropTypes from "prop-types";
export default class Like extends Component {
  render() {
    return (
        <div>
            <p> Context Name  {this.context.name} </p>
            <p> Context Color {this.context.color} </p>
        </div>
    )
 }
}
```

```
Like.contextTypes = {
    name: PropTypes.string,
    color: PropTypes.string
}
```

Context

NODESEN.SE

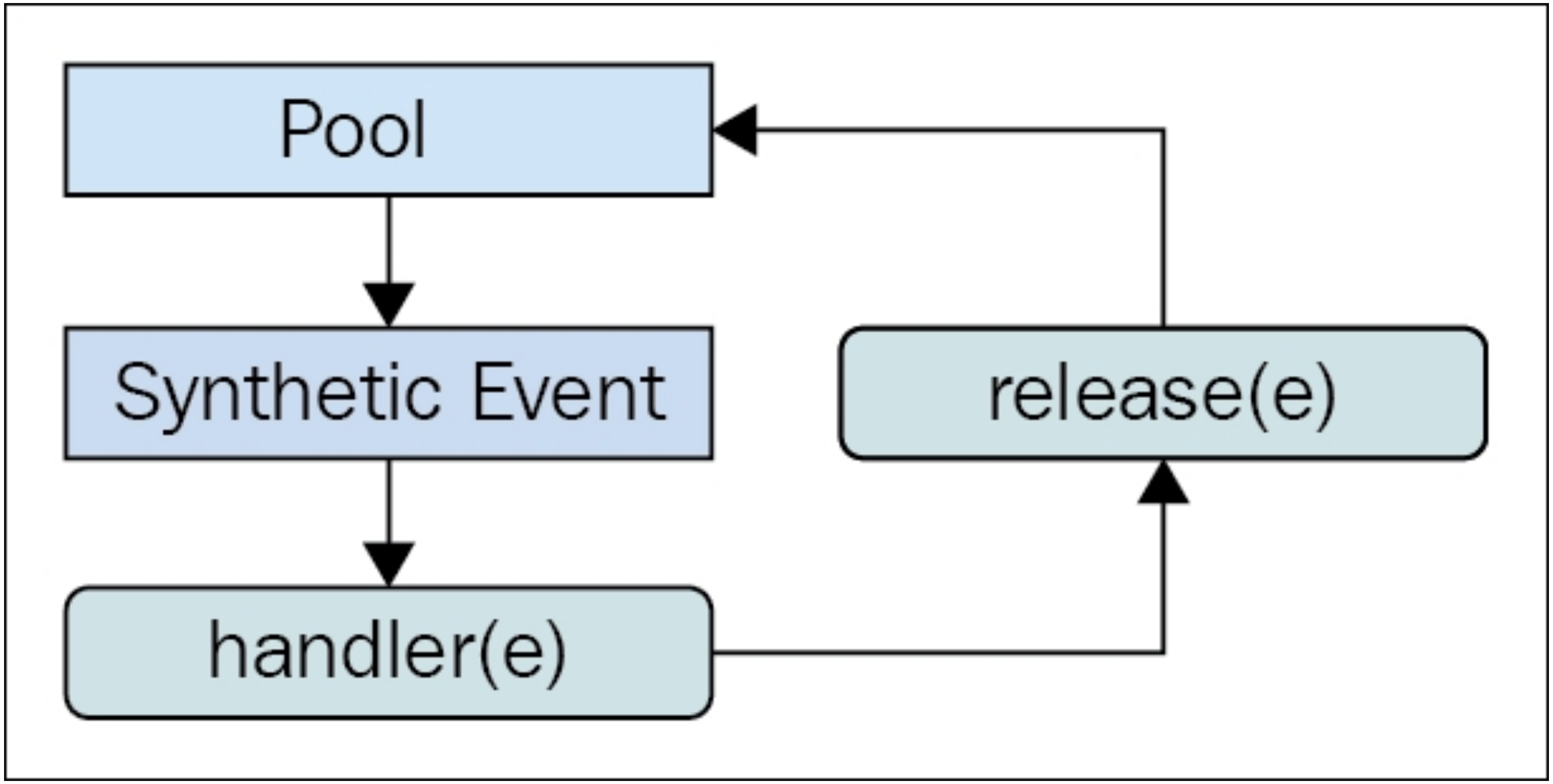
Events

- SyntheticEvent wrapper that forms part of React's Event System
- Provides uniform ways of handling various types of events like Mouse, Key board, Drag/Drop, copy/ paste etc

Events

- React maintain Event Queue for SyntheticEvents, events are executed one after another as per queue
- SyntheticEvents works across all browsers, provides uniform functionalities
- SyntheticEvents wraps real DOM Events
- Events are pooled, recycled, to improve performance

Synthetic Event Pool


Pool
Synthetic Event
handler(e)
release(e)