ReactJS Components

by Vlad Costel Ungureanu
for “Learn Stuff” and “Pentalog Romania”

ReactJS Components

- ✓ Basic Components
- ✓ Props
- ✓ State
- ✓ Lifecycle
- ✓ Working with Components

Basic Components

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

// just some ES6 stuff
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

// usage
const element = <Welcome name="Sara" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Basic Components

- ✓ A components is a pure JS function
- ✓ Always name with a capital letter
- ✓ Receives a single parameter called props (short for properties)
- ✓Returns a HTML/JSX element
- ✓ Only the render/return function is necessary to define a React class

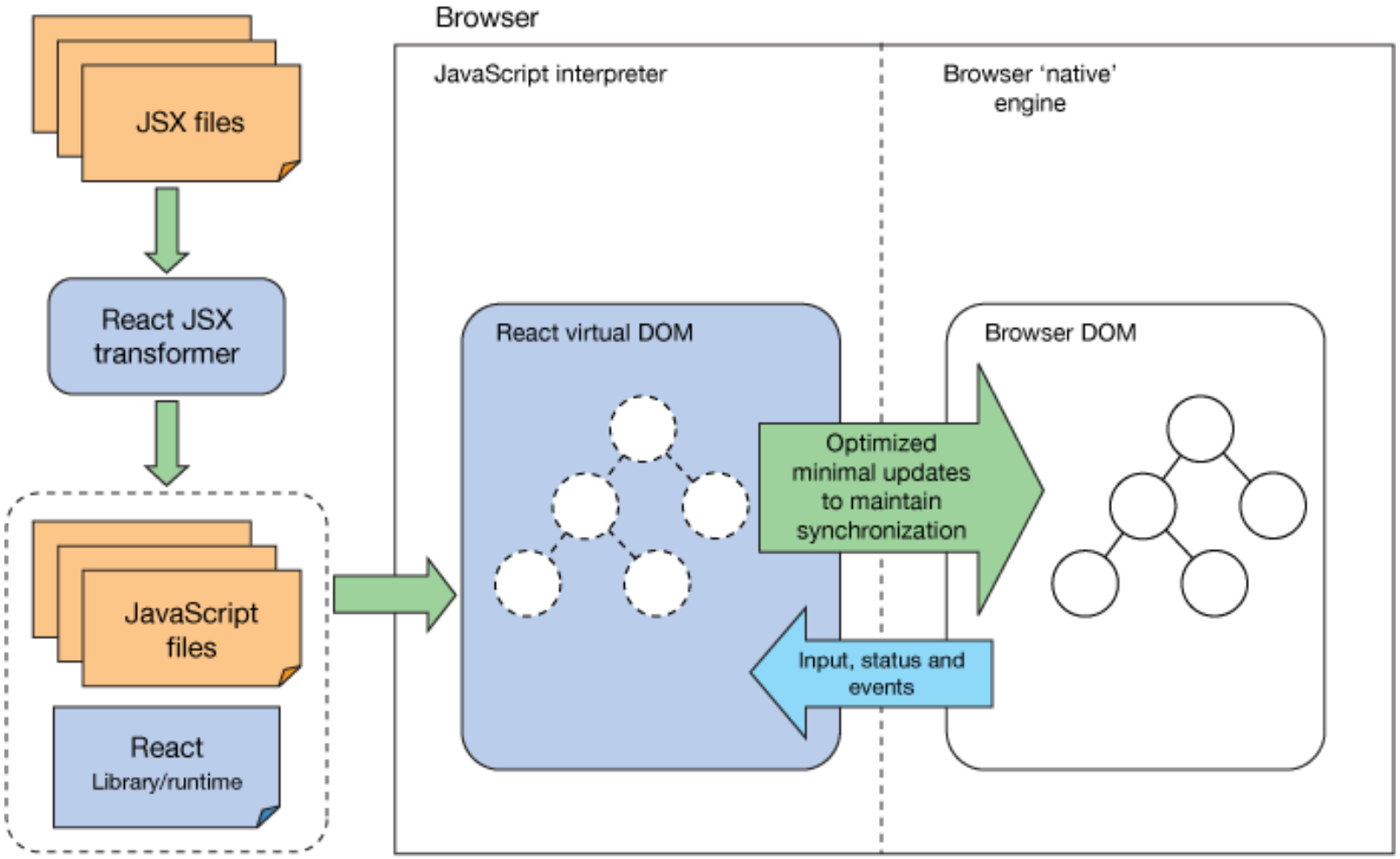
Props

✓ All attributes and their values, given to a component, are grouped into the “props” object

✓ All React components need to act like pure JS functions and not change the props object

```
// pure
function sum(a, b) {
  return a + b;
}

// impure
function withdraw(account, amount) {
  account.total -= amount;
}
```


Browser
JavaScript interpreter
Browser 'native' engine
JSX files
React JSX transformer
JavaScript files
React
Library/runtime
React virtual DOM
Browser DOM
Optimized minimal updates to maintain synchronization
Input, status and events

State

- ✓Similar to props but it is private and fully controlled by the component
- ✓State and props are the same after initialisation
- ✓State represents the current displayed state of the component
- ✓Direct changes to the state do not reflect in the component view
- ✓In order to change the displayed state we need to call “setState()”
- ✓The “setState” method merges given object to existing state allowing for parts of the state to be updated in different methods
- ✓State and props are updated asynchronously and should not be used together

State

- ✓States should be considered isolated/encapsulated
- ✓Neither child nor parent components should care about the others state
- ✓This implies that data flow is **unidirectional** and can only be transmitted from parent to children
- ✓State management is not mandatory for any component
- ✓Components with state are **stateful** and those without state are **stateless**

Lifecycle

"Render Phase"

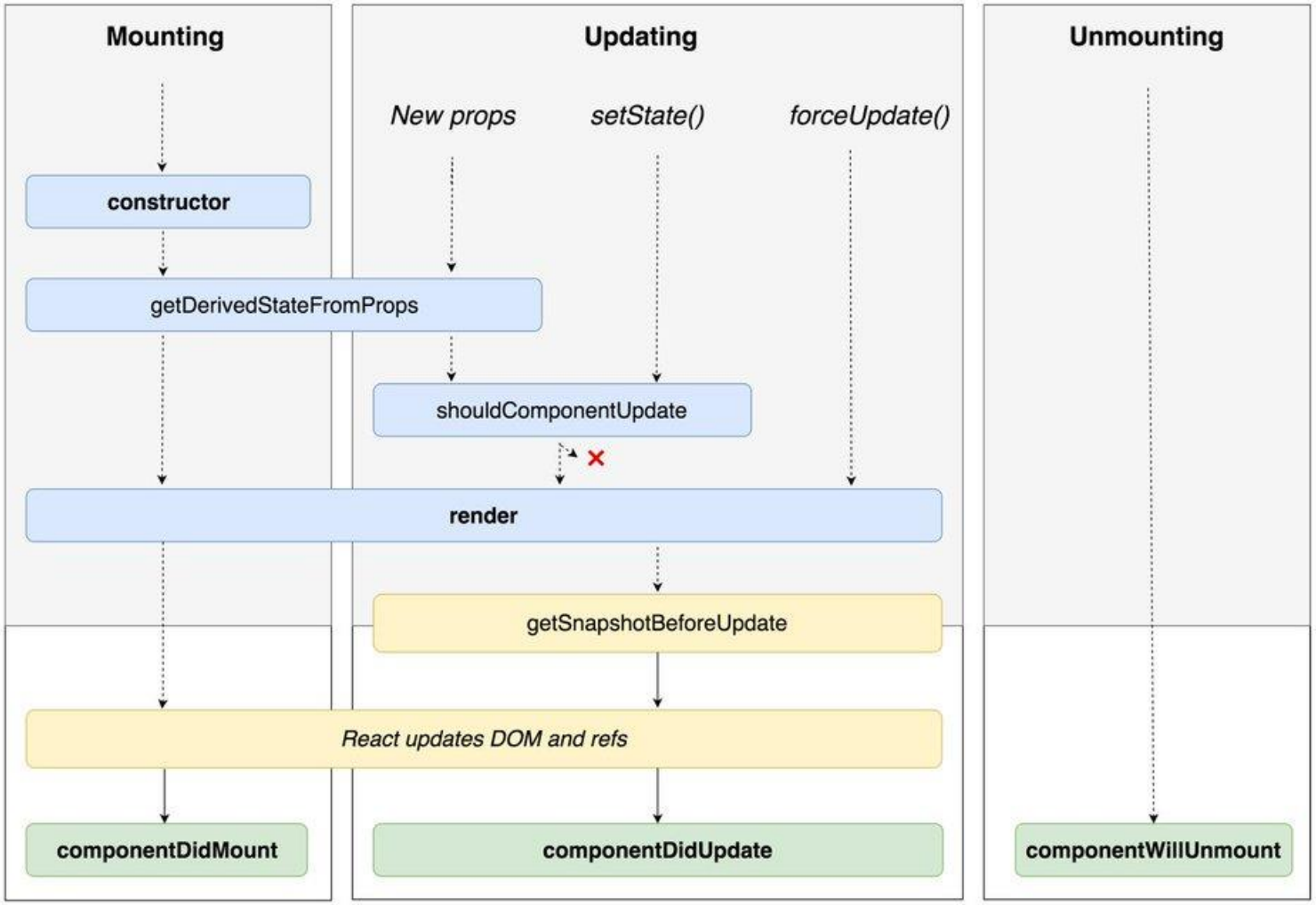
Pure and has no side effects. May be paused, aborted or restarted by React.

"Pre-Commit Phase"

Can read the DOM.

"Commit Phase"

Can work with DOM, run side effects, schedule updates.



Lifecycle - Mount

✓ constructor(props)

// initialize local state, bind event handlers to instance if not using ES6 arrow functions, initialize state instead of calling setState, do not copy props into state, always call base class constructor with props

✓static getDerivedStateFromProps(props, state)

✓render()

// pure, idempotent, does not interact with browser, should not be explicitly called, render can encapsulate if statements in order to achieve conditional rendering

✓componentDidMount()

// called after component mounted, should contain DOM related logic and requests, setState might impact performance as it calls render() again

Lifecycle - Update

✓static getDerivedStateFromProps(props, state)

// return object to update state or nothing, prone to error if manually used, does not have access to component instance

✓shouldComponentUpdate()

✓render()

✓getSnapshotBeforeUpdate()

✓componentDidUpdate(prevProps, prevState, snapshot)

// called immediately after any update except the initial one, DOM peration and requests after update

Lifecycle – Unmount and Error Catching

✓componentWillUnmount()

// called after components is removed from DOM, used for clean up, should not use setState

✓componentDidCatch(error, info)

// we will discuss this in detail in another presentation

Working with Components

- ✓ Even the most basic component should be encapsulated and able to perform independently of other components
- ✓As such, basic components compose more complex components which in turn can be composed to form even more complex components
- ✓One of the most important aspects of component based applications is reusability, taken as the main characteristic of any component
- ✓If done correctly, this ensures that components are in their most simple, yet functional form and any complexity arises only from the incorrect composition of simple elements

Working with Components

- ✓Component composition can be abstract in the sense that you can pass child components as props to other components

- ✓ While similar result can be obtain by using inheritance, there is no specific case in which this proves to be beneficial over the composition approach

THANK YOU!

Vlad Costel Ungureanu
ungureanu_vlad_costel@yahoo.com