React

(Yet Another) Approach to create Browser UI

Training Agenda

- React – Overview
- Change Detection
- Building blocks
  - Components
  - JSX
  - Props
  - State
  - Refs
  - Context API
- Remote Calls
- Creating SPA
- Redux (overview and building blocks)
- Redux integration with React

Prerequisites

- Sound knowledge of latest JavaScript
- Working knowledge of HTML & CSS

What is React

React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

Developed by Facebook and Instagram

Intended to be the View (“V”) or the user interface in MVC

Aims at effortless development of large scale Single Page App (SPA)

Components are defined and eventually becomes HTML

Why React?

- Easy to understand for developers with the knowledge of XML/HTML
- UI state becomes difficult to handle with Vanilla Javascript
- High Performance : renders quick view
- Focus on business logic
- Huge Ecosystem
- Active community
- Easy to test

Getting Started : Installation Steps

To get started with React, install the React CLI Tool (create-react-app)

Run the below command to create new project :

```
> npm install create-react-app -g

> create-react-app <APP_NAME>

> cd <APP_NAME>

> npm start
```

Virtual DOM

“React abstract away the DOM from you, giving a simpler programming model and better performance”

- React

- Virtual DOM is in-memory lightweight representation of actual DOM.
- For every DOM object, there is a corresponding Virtual DOM object.
- Pure JS intermediate representation.
- React never reads from real DOM, only writes to it.
- The process of updating any part of the DOM structure is called “reconciliation”.

Virtual DOM : Update

The virtual DOM gets compared to previous vs current virtual DOM

↓

React figures out what object has changed

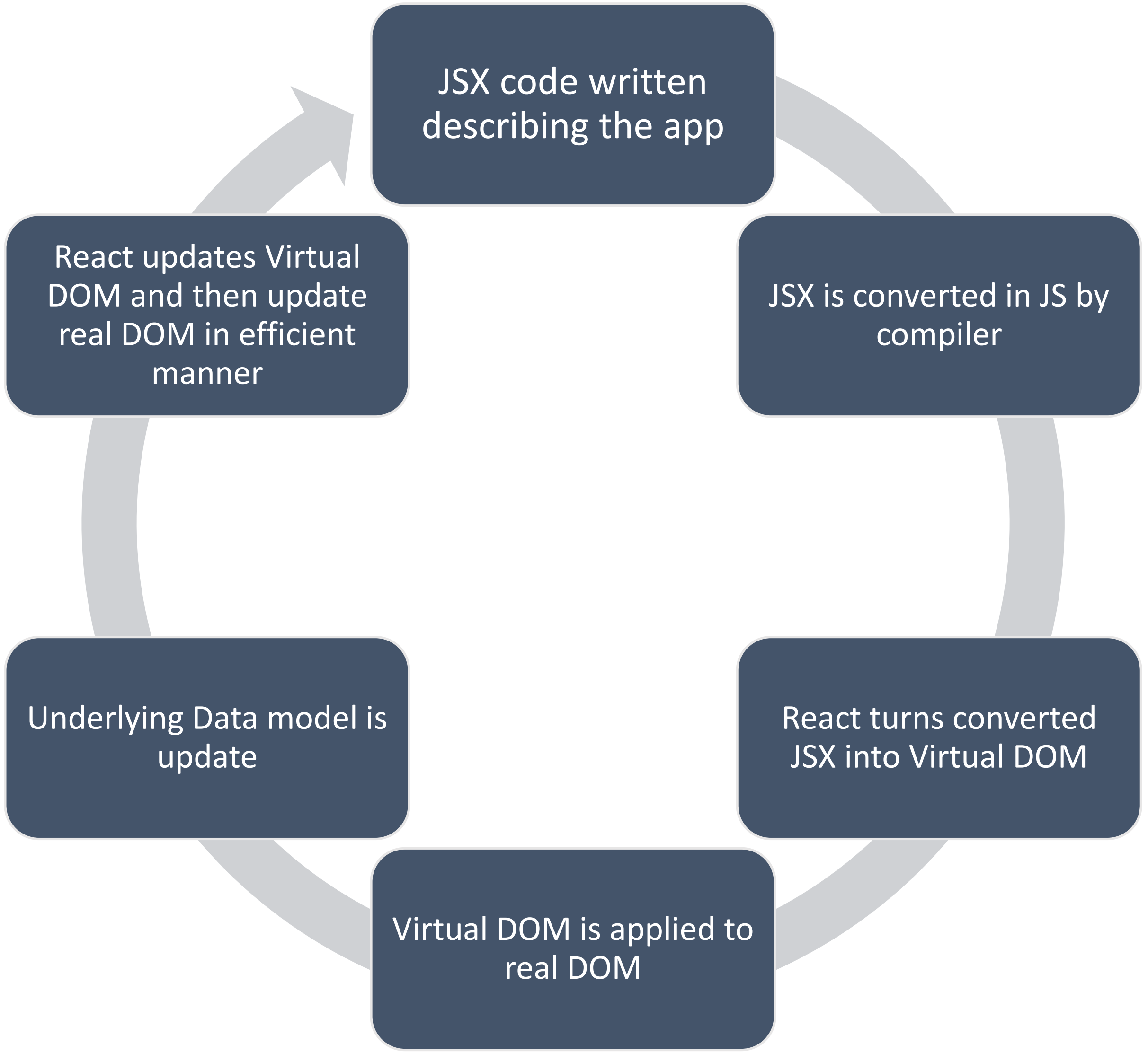
↓

The changed objects, and the changed objects only, get updated on the real DOM.

↓

Changes on the real DOM cause the screen to change.

How React renders the View


JSX code written describing the app
JSX is converted in JS by compiler
React turns converted JSX into Virtual DOM
Virtual DOM is applied to real DOM
Underlying Data model is update
React updates Virtual DOM and then update real DOM in efficient manner

JSX

JSX defines which HTML code React should render to the real DOM in the end.

JSX is not HTML but it looks a lot like it.

JSX is just syntactic sugar for JavaScript, allowing you to write HTMLish code instead of nested React.createElement(...) calls.

Each component needs to return/ render some JSX code

JSX ensures readability and maintainability

JSX finds most of the errors at compile time, hence faster than JS

JSX Restrictions

- Consider the keywords
- One root element per component

Components

Components are the core building block of React apps.

A typical React app therefore could be depicted as a component tree - having one root component ("App") and then an potentially infinite amount of nested child components.

Each component needs to return/ render some JSX code - it defines which HTML code React should render to the real DOM in the end.

React component can be Stateful or Stateless

Components can be nested inside other components

Component Types

Functional Components (Functional)

Class based Components (Container)

Component Life Cycle

Mounting

- **constructor()**
- static getDerivedStateFromProps()
- **render()**
- **componentDidMount()**

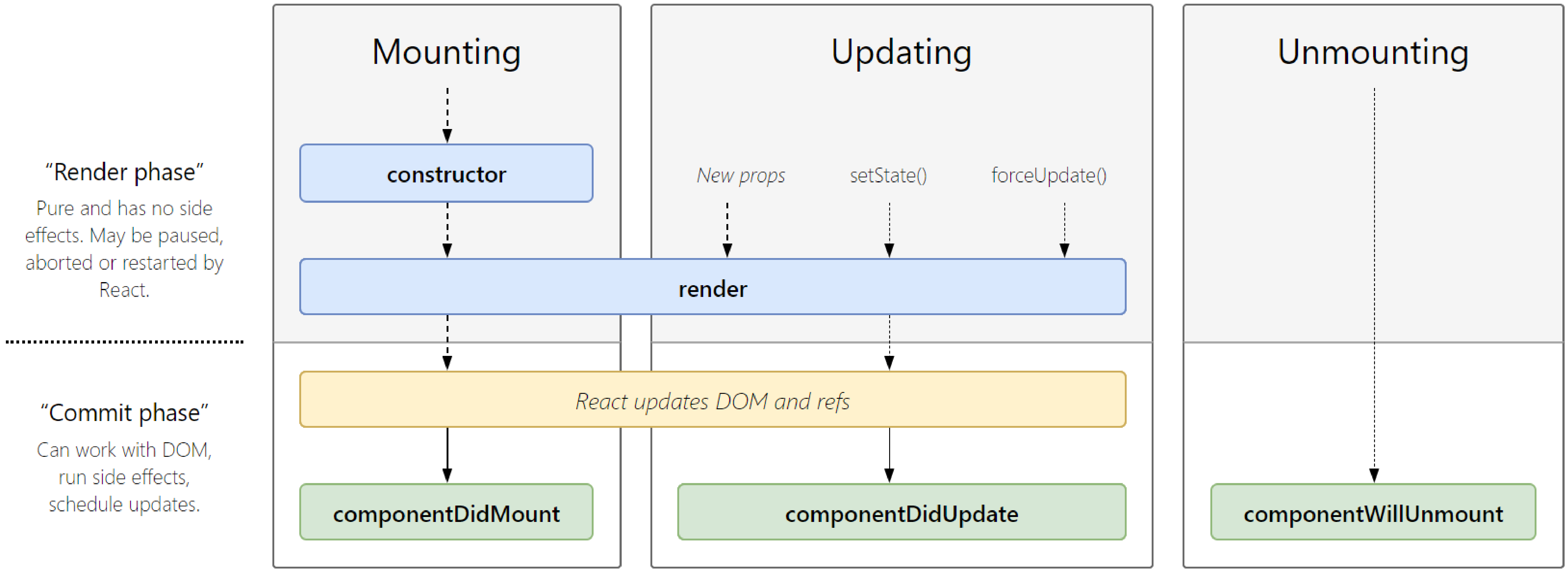
Updating

- static getDerivedStateFromProps()
- shouldComponentUpdate()
- **render()**
- getSnapshotBeforeUpdate()
- **componentDidUpdate()**

Unmounting

- **componentWillUnmount()**

Component Life Cycle



http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram

Props

Props allow you to pass data from a parent (wrapping) component to a child (embedded) component.

Only changes in props and/or state trigger React to re-render your components and potentially update the DOM in the browser.

Props are considered “immutable”

Props are supplied as attribute to components

PropTypes

React.propTypes are used to run type checking the props of a component

Allows to control the presence, or type of certain props passed to the child component

After 15.5, prop-types are moved to library ‘prop-types’

The React.propTypes contains list of Validators:

- String, number, function, Boolean, object, shape, element, any, required etc

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

State

React components can be made dynamic by adding state to it.

State is used when component needs to change independently of its parent.

Changes to state also trigger an UI update.

React component’s state can be updated using setState() with an object map of keys which can be updated with new values. Keys that are not provided will not be affected.

setState() merges the new state with the old state.

Best practice : top level components are stateful which keep all interaction logic, manage UI state, and pass the state down to hierarchy to stateless components using props.

Unidirectional Data Flow

React follows unidirectional data flow via the state and props objects as opposed to the two-way data-binding of libraries like Angular.

State should be updated using setState() method to ensure that the UI is updated and resulting values should be passed down to child components using attributes that are accessible in said children via props.

By keeping the data flow unidirectional you keep a *single source of truth*.

Clean dataflow architecture

Adding Keys for Dynamic Children

Identity and state of each component must be maintained across render passes.

Each child in an array or iteration must be uniquely identified by assigning unique key with the help of “key” prop.

The key should always be supplied directly to the components in an array, not to the container element.

The key is not really about performance, its more about identity (which in turns leads to better performance)

Context API

Context API provide a way to pass data through the component tree without having to pass props down manually at every level : Known as ‘props-drilling’

React Context API gives you solution for instead passing down the props explicitly to each component, you can store the data needed by each component in React Context API and and pass them to all other components implicitly.

If a component needs access to the context, it can consume it on demand

Can be used when data can be considered “global” for a tree of React component.

Forms

Form elements naturally keep some state internally.

Each form elements in HTML maintain their own state and updates it based on user input.

Form element whose value is controlled by React is called “Controlled Components”

Accessing User Input by Refs

- React provides two standard ways to grab values from form elements-
    - Controlled Components
    - Using Refs

- Less code writing, but hinders in optimized working of Babel inline plugin.

Routing

Each React app has been a type of SPA.

React-router library gives us good foundation for building rich applications which have views and URL’s

Routing involves two functionality :

- Modify the location of the app (the URL)
- Determining what component need to render at given location.

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

Redux

A predictable state container for JavaScript apps.

Should I Use Redux?

Some suggestions:

- You have reasonable amounts of data changing over time
- You need a single source of truth for your state
- You find that keeping all your state in a top-level component is no longer sufficient

Redux

Redux is a predictable state container for JavaScript apps.

It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

It provides a great developer experience, such as live code editing combined with a time traveling debugger.

```
> npm install redux react-redux --save
```

Redux : Three Principles

Single source of truth

- The *state* of your whole application is stored in an object tree within a single *store*.

State is read-only

- The only way to change the state is to emit an *action*, an object describing what happened.

Changes are made with pure functions

- To specify how the state tree is transformed by actions, you write pure *reducers*.

Actions

- *Actions* are payloads of information that send data from your application to your *store*.
- They are the only source of information for the store.
- You send them to the store using `store.dispatch()`.

```
{
        type: ADD_TODO,
        text: 'Build my first Redux app’
}
```

Reducers

- The *reducer* is a pure function that takes the previous *state* and an *action*, and returns the next state.
- *Actions* only describe *what happened*, but don't describe how the application's state changes.
- **Reducers** specify how the application's state changes in response to *actions* sent to the *store*.

```
(previousState, action) => newState
```

Reducers : Don’ts

Things you should **never** do inside a reducer:

- Mutate its arguments.
- Perform side effects like API calls and routing transitions.
- Call non-pure functions, e.g. `Date.now()` or `Math.random()`

Store

The Store is the single *object* that has the following responsibilities:

- Holds application state.
- Allows access to state via `getState()`.
- Allows state to be updated via `dispatch(action)`.
- Registers listeners via `subscribe(listener)`.
- Handles unregistering of listeners via the function returned by `subscribe(listener)`.

Creating Store with Root Reducer

```
import { createStore } from 'redux’ ;
import rootReducer from './reducers’ ;

const store = createStore(rootReducer);
```

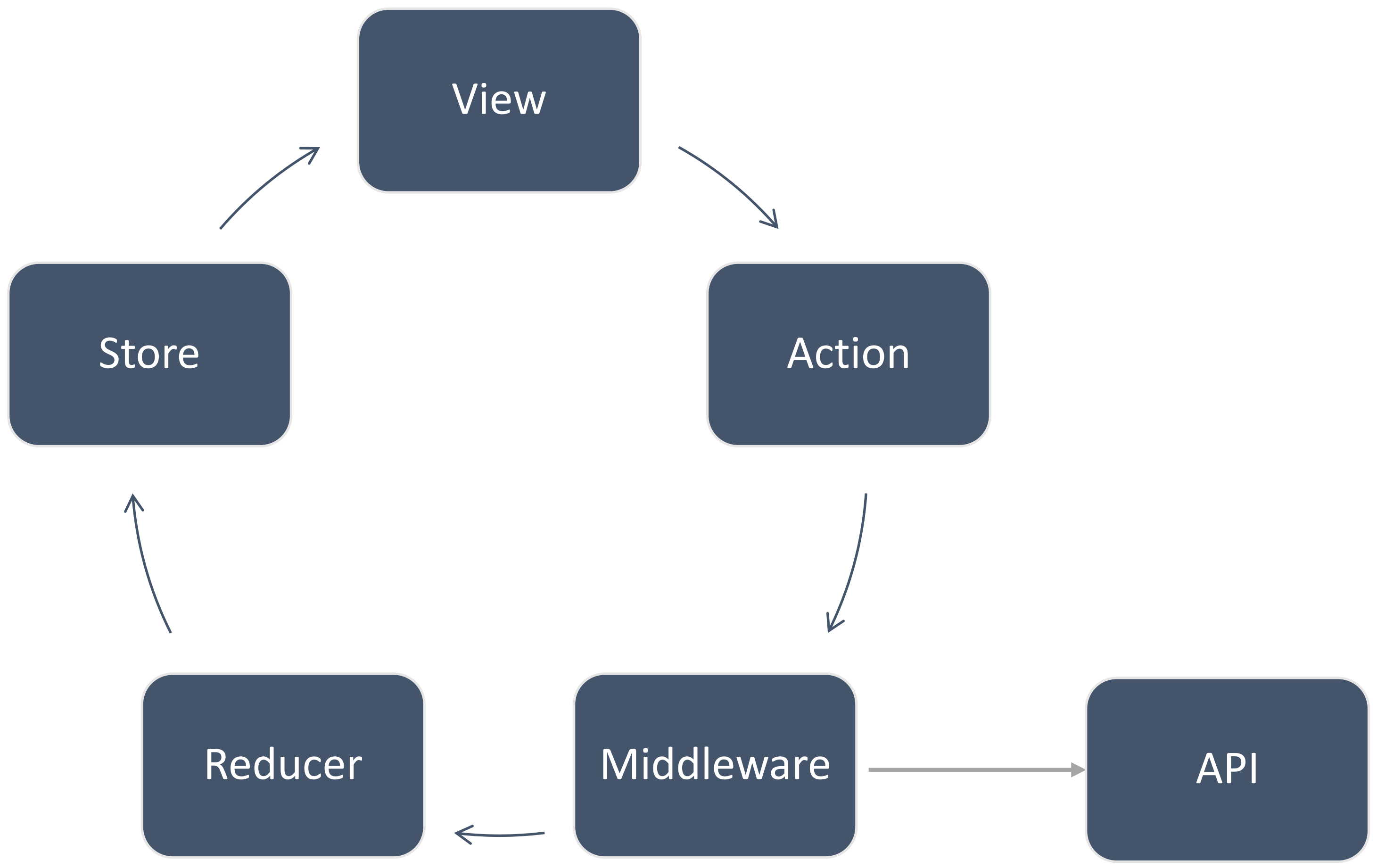
Data Flow

Redux architecture revolves around a **strict unidirectional data flow**.

The data lifecycle in any Redux app follows these 4 steps:

- You call `store.dispatch(action)`.
- The Redux store calls the reducer function you gave it.
- The root reducer may combine the output of multiple reducers into a single state tree.
- The Redux store saves the complete state tree returned by the root reducer.

Redux Architecture & Data Flow



References

- http://javascript.info
- https://reactjs.org
- https://redux.js.org
- https://www.npmjs.com
- https://gist.github.com/danharper/3ca2273125f500429945
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol