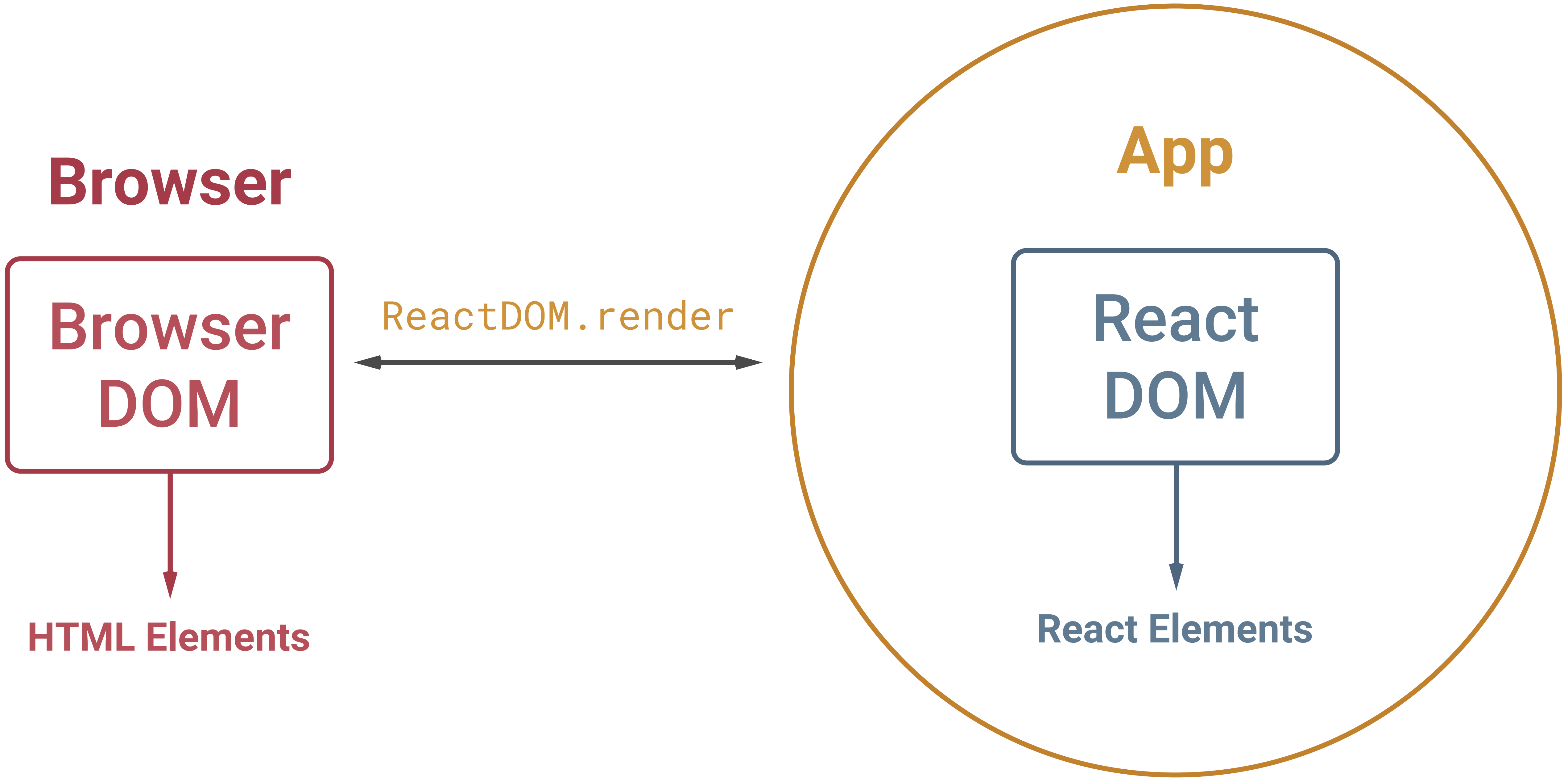
React
Inside the virtual DOM

What is React?

React is a javascript library that let you build a virtual DOM
that get rendered into the browser only when needed
and changes only things that need to be changed.



React DOM

```
{
  App : {
    type : React.element
    children : [
      div : {
        type : React.element,
        className : 'div-element',
        children : [
          MyComponent : {
            type : React.element,
            props : {},
            state : {}
          }
        ]
      }
      div : {
        type : React.element,
        className : 'div-element',
        children : 'some nice text'
      }
    ]
  }
}
```

Simple Application

```
// HTML File:
// defining a root element to render the react app
<div id="root"><!-- React App renders here --></div>


// JS File:
// initial rendering of the application:
ReactDOM.render(
    <div>Hello Styla!</div>,
    document.querySelector( '#root' )
);
```

JSX !== HTML – understanding the React DOM

```
// create React DOM Element:
React.createElement(type, props, children);

// Example React Div Element:
React.createElement( 'div', null, 'Hello Styla!' ); // OR:
React.DOM.div( null, 'Hello Styla!' );

// similar creating DOM element in plain js:
document.createElement('div').innerText = 'Hello Styla!';


// Example React element with complex structure
React.createElement(
    'div',
    {
        title : 'hello styla!'
    },
    // children:
    React.createElement('h1', null, 'hello styla!' ),
    React.createElement('p', null, 'welcome to react!' )
);
```

JSX → Simpler Data Structure

```
// with children
<type prop="some-prop">some child element</type>;
// without children
<type prop="some-prop" { ...spreadObject } />;

// babel translates it to:
React.createElement(
    'type',
    {
        prop : 'some-prop'
    },
    'some child element'
);
```

JSX → Element type

```
// element type is always expected to be a function
typeof type === 'function'

// in jsx html elements start with a small letter:
JSX        : divElement = <div className="something">some content</div>
React does : divElement = React.DOM.div( {
                                     className : 'something'
                                 },
                                 'some content'
                             );

// custom elements start with capitalized letter:
DEFINITION : const MyComponent = () => <div>some content</div>;

JSX        : myComponent = <MyComponent />
React does : myComponent = React.createElement( MyComponent );
```

Types of React Components

– Stateless Components (Functions)

– Pure Components (Classes)
  – rerenders only on stage change

– Components (Classes)

Function Components (Stateless)

```
// simple ES6 function component:
const MyButton = ( props ) =>
    <button className={ `btn btn-${props.type}` }>
        { props.caption }
    </button>

// render to Component:
ReactDOM.render(
    <MyButton
        caption="Stateless Component"
        type="danger"
    >,
    document.querySelector( '#app' )
);
```

Class Components (State lifecycle Components)

```
// simple ES6 function component:
class MyComponent extends React.component
{
    render()
    {
        return (
            <div className="my-component">
                <h1>{ this.props.title }</h1>
            </div>
        );
    }
}

ReactDOM.render( <MyComponent title="title" /> );
```

Props (this.props)

- coming from outside the Component
- immutable (should not be changed)
- JS Object

State (this.state)

- is used inside the components
- can be changed inside a component
- mutatin only with `this.setState` method
- JS Object

React Component lifecycle methods

```
class MyComponent extends React.component
{
    constructor() { // initially on instantiation, set initial state here }
    componentWillReceiveProps(nextProps) {
        // before receiving new props from outside ( but not on initial call )
    }
    shouldComponentUpdate(nextProps, nextState) {
        // before rendering after setState. return true or false
        // to make sure the component runs the lifecycle or not
    }
    componentWillUpdate( nextProps, nextState ) {
        // before the props or state will change
        // not allowed to run setState here!
    }
    render() { // render / mount element to the dom }
    componentDidMount() { // after rendered the first time }
    componentDidUpdate(prevProps, prevState) {
        // after setState and render, but not initially
    }
    componentWillUnmount() { // before element will be removed }
}
```

```
// read more: https://facebook.github.io/react/docs/react-component.html
```

Props validation

propTypes

```
import PropTypes from 'prop-types'; // since react v15.5

...
static propTypes = {
    optionalNumber : PropTypes.number,
    requiredNumber : PropTypes.number.isRequired,
}
...
```

defaultProps

```
...
static defaultProps = {
    title : 'default title',
    content : 'default content',
}
...
```

https://facebook.github.io/react/docs/typechecking-with-proptypes.html

Refs

```
class MyComponent extends React.Component
{
  componentDidMount()
  {
    ~~console.log( this.refs.inputNode )~~ //--> deprecated
    console.log( this.inputNode ); // --> the input field node
    console.log( this.other );     // --> the React Element of other
  }
  render()
  {
    return (
      <div className="input-refs">
        <OtherComponent ref={ other => this.other = other } />
        <input type="text" ref={ input => this.inputNode = input } />
        ~~<input type="text" ref="inputNode" />~~
      </div>
    }
}
```

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

Concepts

How to structure react

1. Lifting State Up

Whenever two components relate to the same state the parent component should handle the state

```
Input = props => <input type="text" onChange={ props.setValue } />


class StateComponent extends React.Component
{
    setA = ( e ) => { this.setState( { a : e.currentTarget.value }
    setB = ( e ) => { this.setState( { b : e.currentTarget.value }

    render() {
        return (
            <div>
                <Input setValue={ this.setA } />
                <Input setValue={ this.setB } />

                <div>StateA : { this.state.a }</div>
                <div>StateB : { this.state.b }</div>
            </div>
        );
    }
}
```

https://facebook.github.io/react/docs/lifting-state-up.html

2. Composition vs Inheritance

> ”React has a powerful composition model, and we recommend using composition instead of inheritance to reuse code between components.“

```
WrapperComponent = props => <div>{ props.children }</div>


InnerComponent = props =>
{
    return (
        <WrapperComponent>
            <div>I’m a child element</div>
        </WrapperComponent>
    );
}
```

https://facebook.github.io/react/docs/composition-vs-inheritance.html

3. Higher-Order Components

A higher-order component (HOC) is an advanced technique in React for reusing component logic. Not part of the React API, but a pattern that works nicely with the nature of react.

```
higherOrderComponent = ( Component ) =>
{
    const someProps = { foo : 'bar' }
    return (
        <Component { ...someProps } >
    );
}

WrappedComponent = props => <div>{ props.foo }</div>;

EnhancedComponent = higherOrderComponent( WrappedComponent );
```

https://facebook.github.io/react/docs/higher-order-components.html

ReactDOMServer

The ReactDOMServer class allows you to render your components on the server.

```
MyServerComponent = ( props ) =>
{
    return (
        <div>Ill be rendered on the server</div>
    );
}

htmlReactString = ReactDOMServer.renderToString( MyServerComponent );
htmlString = ReactDOMServer.renderToStaticMarkup( MyServerComponent );
```

https://facebook.github.io/react/docs/react-dom-server.html

Thank you!