React

JS Library to build
Complex Interactive User Interfaces

$$\begin{tikzcd}
\text{Old School Web} \arrow[d] \\
\text{AJAXified Web} \arrow[d] \\
\text{Single Page Applications}
\end{tikzcd}$$

(Complex Interactive Components)

https://en.wikipedia.org/wiki/Single-page_application

Motivation for React

- Increasing complexity on front-end
- SPAs - complex interactive user interfaces
- More sophisticated JavaScript evolved
- Frameworks like Angular, libraries like React

Today

- React Pre-reqs
- Set up Node, Server
- Set up Express to serve static index.html which just displays Hello World
    - app.use (express.static(‘static’))
- We will React-ify this eventually …

Hello World - Static

(WhiteBoard)

We will upgrade this diagram as we learn more …

Pre-Requisites

More productive / more organised
React code

Tools to make our
“more productive/more organised”
React code
compatible with browsers

ES6 (Classes and Modules)

JSX

Babel

WebPack

ECMAScript (ES)

- Standard
- Defines core features of an ECMAScript Language
- JavaScript - most popular ECMAScript Language
- JavaScript = ECMAScript as Core + Additional Features

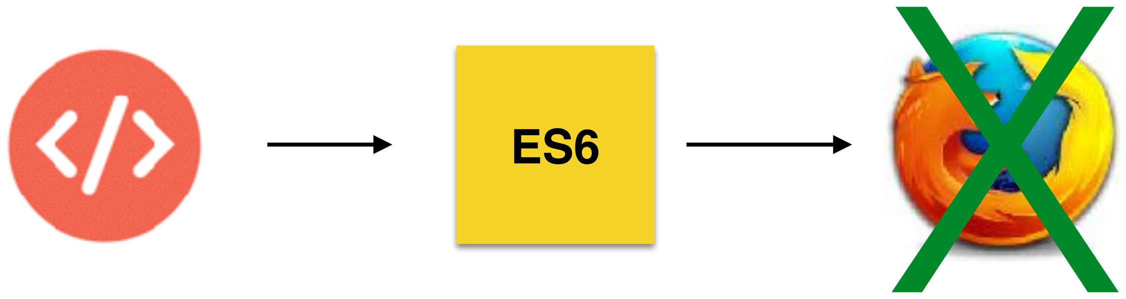
ES Versions - ES6 and ES5

ES5	**ES6 / ES2015**
5th Edition of ES	6th Edition of ES
2009	2015
Less Features	**New - More Features!**
Most Modern Browsers Understand ES5	**Only Some Browsers Understand ES6**

ES6 Modules, ES6 Classes

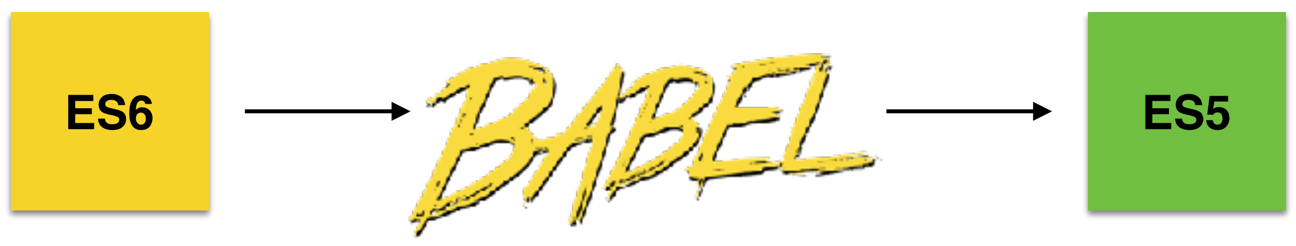
Browsers and ES

- What runs your JavaScript code? Browsers!
- Only few browsers understand ES6
- Problem
  - Want to code with new features in ES6
  - Want most modern browsers to run our code

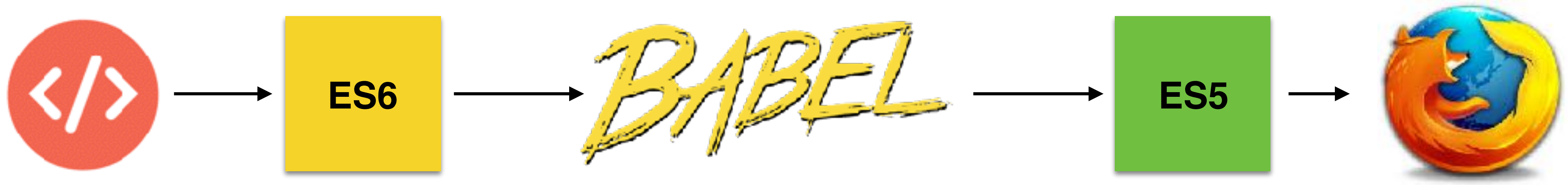


Transpilers

- Transpiler - source to source compiler. Eg. Babel



- Solution



ES6 Classes

ES6 “Classes”

- Objects and Prototypical Inheritance in JS
- ES5 “Classes”
- ES6 “Classes”

“Classes” in JavaScript

- There are no classes in JavaScript!
- Everything is an object
- Objects inherit from objects.
- Classes don’t inherit from classes
- Functions somewhat help us simulate classes

ES5 “Classes”

- ES5 “Classes” - A way to build new objects (from a “template”)
- Keywords
  - Objects in JS
  - Prototypes in JS
  - Prototypical Inheritance

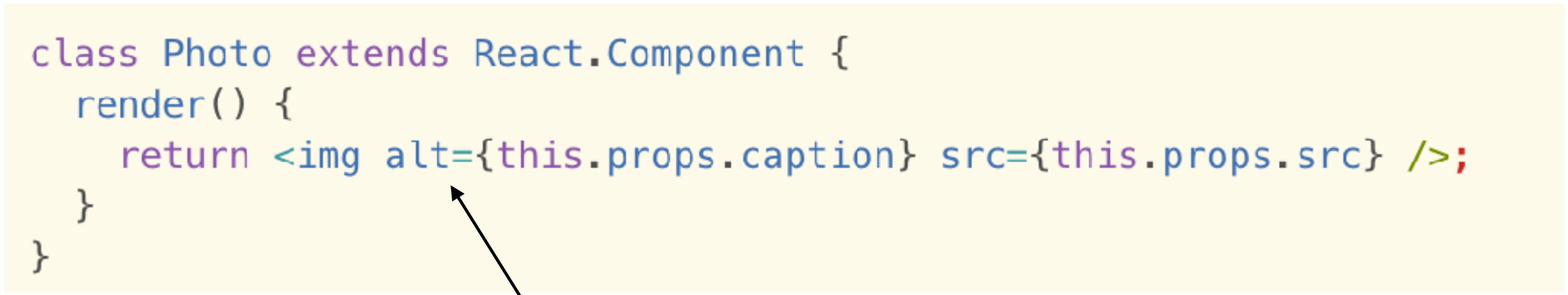
  - Function Constructors
  - Data Members and Methods
  - New Keyword
  - Inheritance - ES5 Classes + Node.js utils.inherit

ES6 “Classes”

- ES6 brings ‘cleaner’ syntax to ES5 Classes.
- ES6 is Syntactic Sugar
- Under the hood - same behaviour/functionality as ES5 ‘classes’!
- Appearance wise
  - cleaner syntax
  - syntax similar to Java/C++ classes
- Warning - Looks are deceptive
  - Don’t confuse them with Java/C++ classes
  - Still object prototypical inheritance

ES6 “Class” React Example

- A React Component corresponds to a UI element in your SPA.
- Each React Component inherits from React.Component
- Each React Component has to define a render() method



```
class Photo extends React.Component {
  render() {
    return <img alt={this.props.caption} src={this.props.src} />;
  }
}
```

Weird HTML in JavaScript ??? - JSX

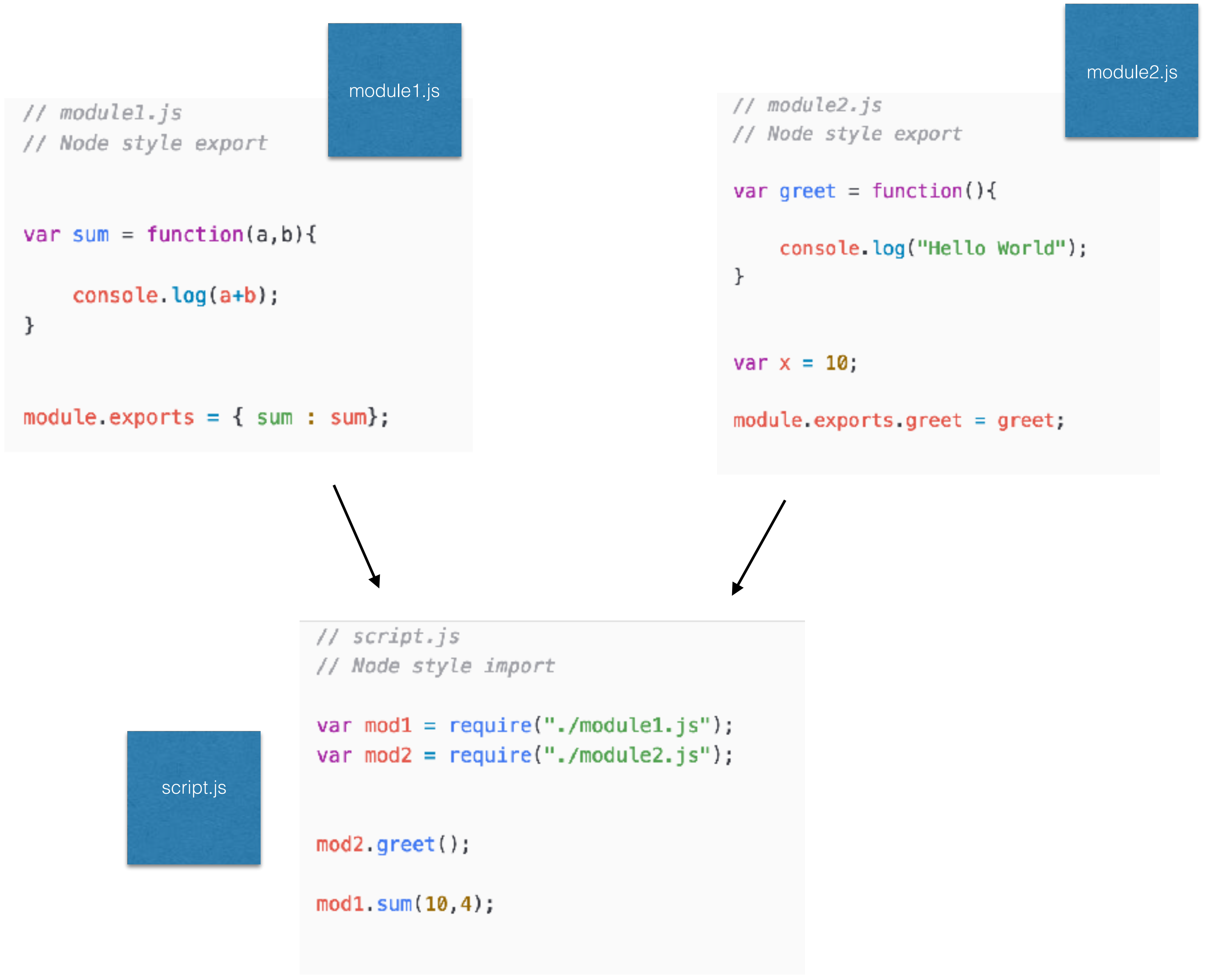
ES6 Modules

ES6 modules

- ES5 - No modules
- Node - enables modules in ES5 JS
- ES6 - Has modules!
- *Motivation - ES6 modules will help us organise our React code*

Node Modules - Revision

- Module
    - Reusable piece of code. One file = One module
    - Defining a module
    - Exporting from a module
    - Importing a module
        - Modules from same directory / specific directory
        - Standard modules



```
// module1.js
// Node style export


var sum = function(a,b){

    console.log(a+b);
}


module.exports = { sum : sum};
```

```
// module2.js
// Node style export

var greet = function(){

    console.log("Hello World");
}


var x = 10;

module.exports.greet = greet;
```

```
// script.js
// Node style import

var mod1 = require("./module1.js");
var mod2 = require("./module2.js");


mod2.greet();

mod1.sum(10,4);
```

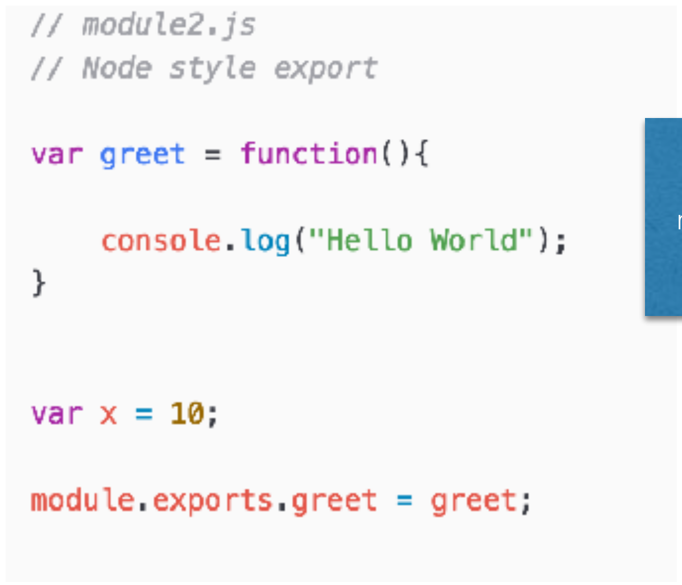
Modules
Node vs ES6

	Node Modules	ES6 Modules
Defining A Module	One File = One Module	One File = One Module
Exporting	**module.exports**	**export, default export**
Importing	**require function**	**import keyword**
Standard	CommonJS	CommonJS

Exporting

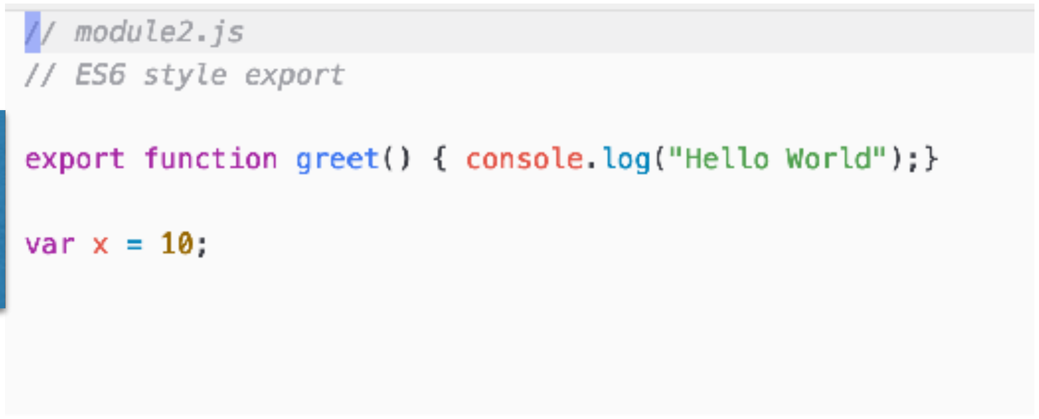
- Node modules : To export an object, add it to the module.exports object

  ES6 modules : To export an object, prefix it with the 'export' keyword.

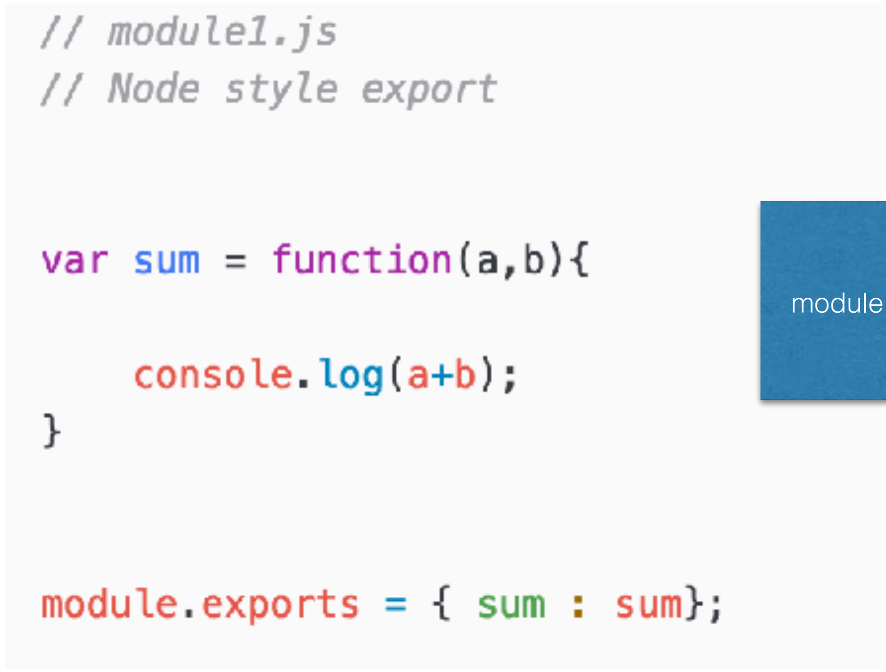


```javascript
// module2.js
// Node style export

var greet = function(){

    console.log("Hello World");
}


var x = 10;

module.exports.greet = greet;
```

module2.js

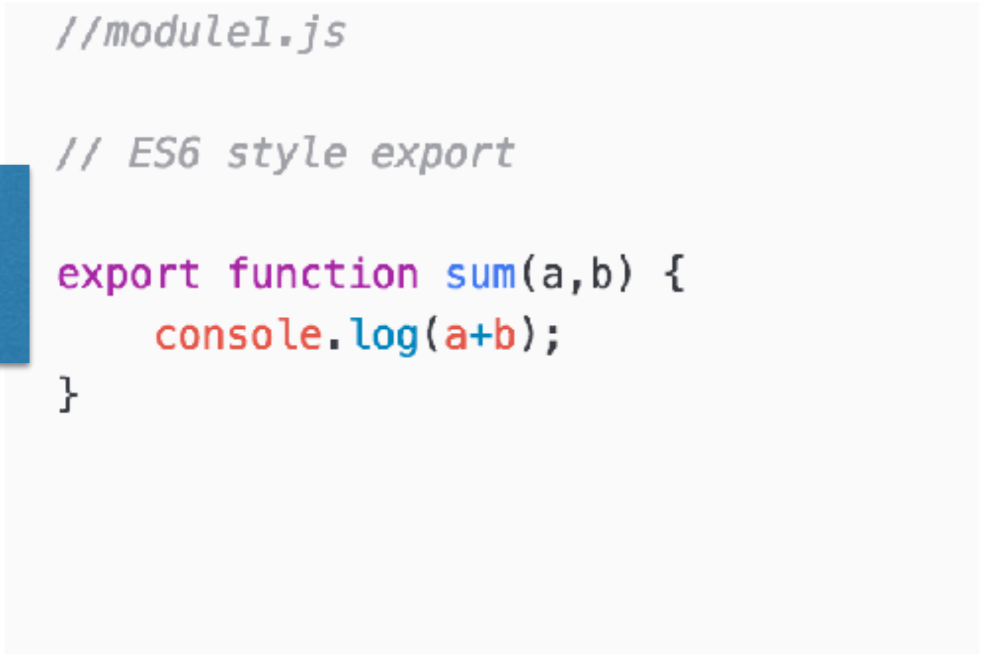


```javascript
// module2.js
// ES6 style export

export function greet() { console.log("Hello World");}

var x = 10;
```



```javascript
// module1.js
// Node style export


var sum = function(a,b){

    console.log(a+b);
}


module.exports = { sum : sum};
```

module1.js

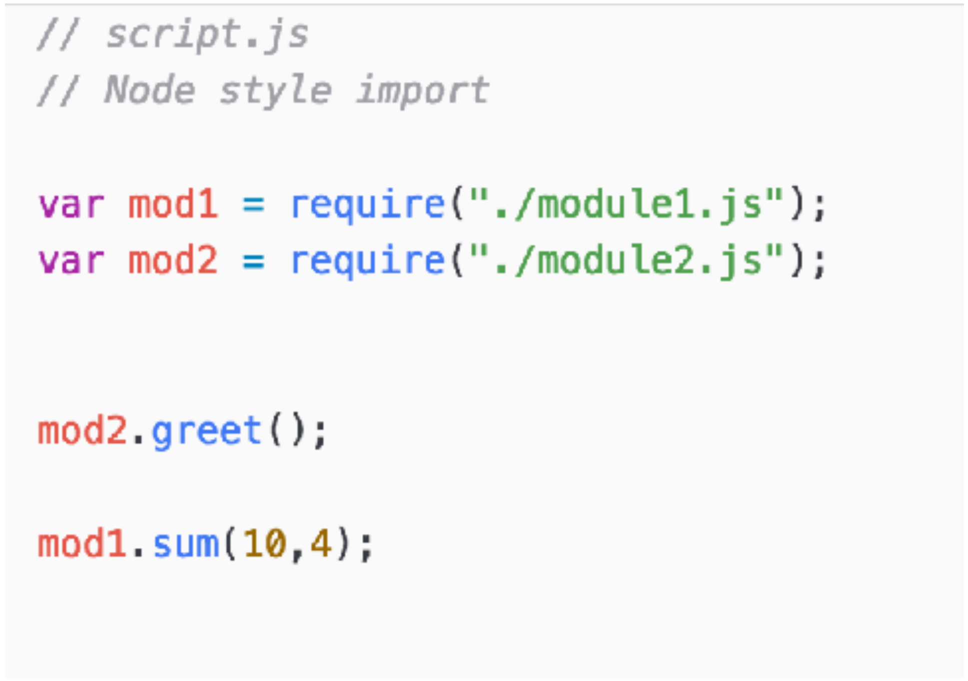


```javascript
//module1.js

// ES6 style export

export function sum(a,b) {
    console.log(a+b);
}
```

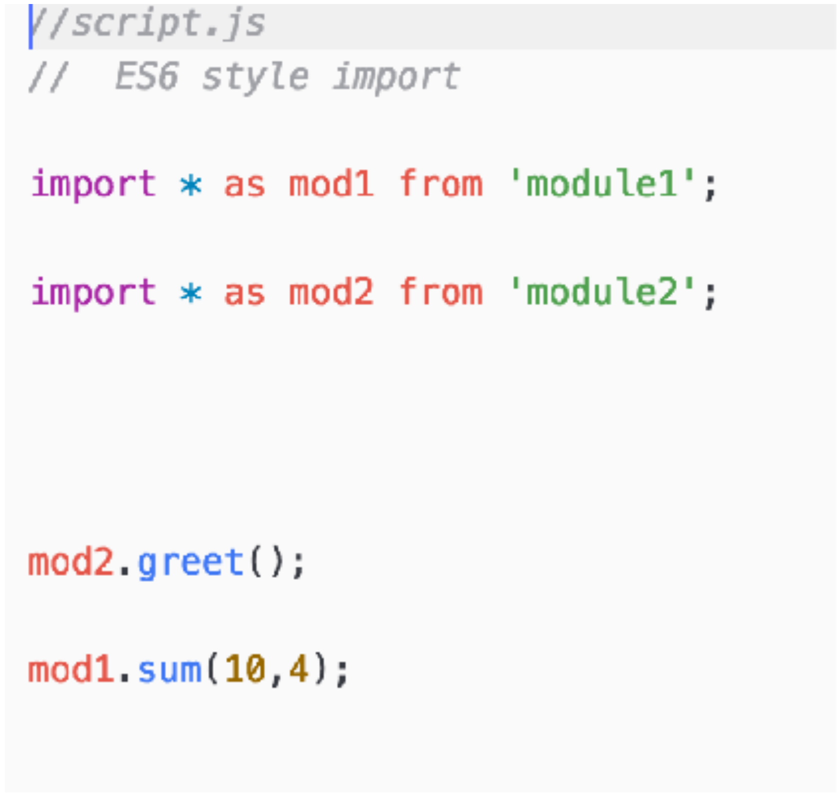
Importing

script.js

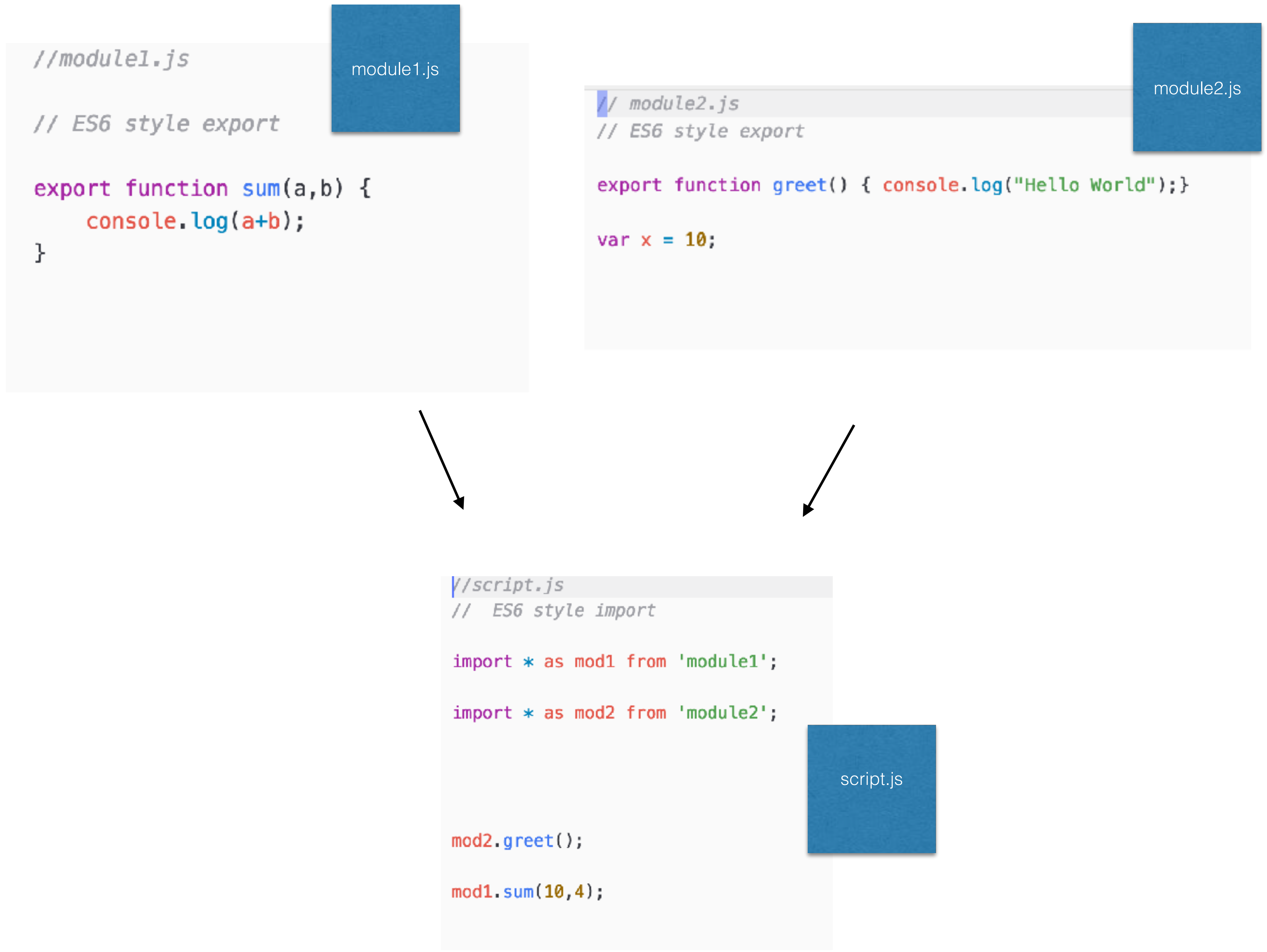
- Node modules : Use the require function
- ES6 modules : Use the import keyword.



```
// script.js
// Node style import

var mod1 = require("./module1.js");
var mod2 = require("./module2.js");


mod2.greet();

mod1.sum(10,4);
```



```
//script.js
//  ES6 style import

import * as mod1 from 'module1';

import * as mod2 from 'module2';



mod2.greet();

mod1.sum(10,4);
```



```
//module1.js

// ES6 style export

export function sum(a,b) {
    console.log(a+b);
}
```

```
// module2.js
// ES6 style export

export function greet() { console.log("Hello World");}

var x = 10;
```

```
//script.js
//  ES6 style import

import * as mod1 from 'module1';

import * as mod2 from 'module2';



mod2.greet();

mod1.sum(10,4);
```

More…
Variations on import, export

- More …
  - Variations on import
  - export, export default
  - HW - Check out
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

- For now
  - we have modules in ES6
  - ‘import keyword’ instead of ‘require function’
  - ‘export keyword’ instead of ‘module.exports object’

Browsers and ES6

- Browsers don’t understand ES6 syntax
    - Eg. (class, extends, import, export)
    - Solution - Transpile - Babel
- Browsers don’t understand Node/ES6 modules.
    - Eg. (multiple files, require/import, module.exports/export)
    - Solution - Module Bundling - WebPack (?)

ES6

JSX

Babel

WebPack

JSX

- JSX - JavaScript Extension that makes React code neater - when it comes to HTML/XML expressions

- **Syntactic Sugar**
  - for writing HTML/XML expressions in React code
  - Again : Better appearance, same functionality

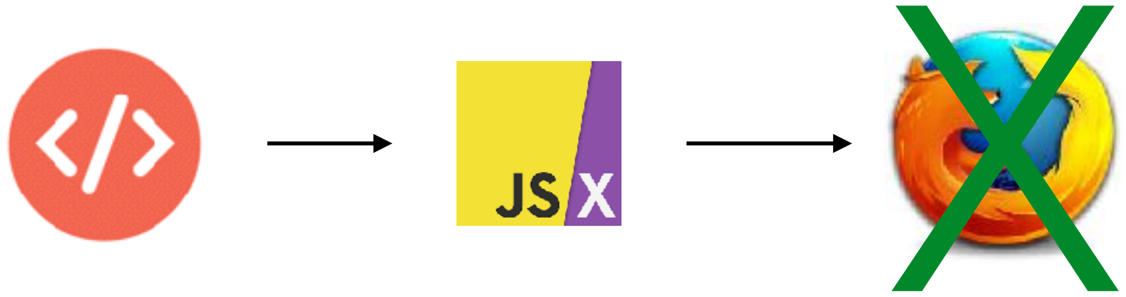
Creating a div element in React

Without JSX	With JSX

`var x = React.createElement(`
`  "div",`
`  null,`
`  " Hello World "`
`);`
	`var x = <div> Hello World </div>;`

- HW - https://facebook.github.io/react/docs/introducing-jsx.h

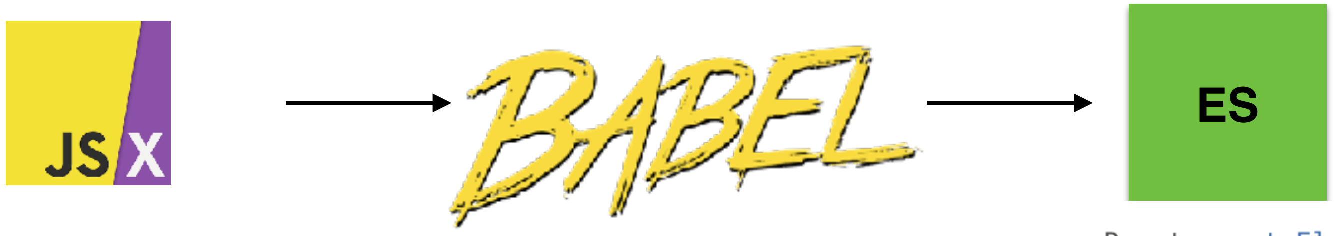
Browsers and JSX



- JSX is a XML-like syntax extension to ECMAScript.
- It’s NOT intended to be implemented by engines or browsers.

Transpilers Again…

- Transpiler - source to source compiler. Eg. Babel

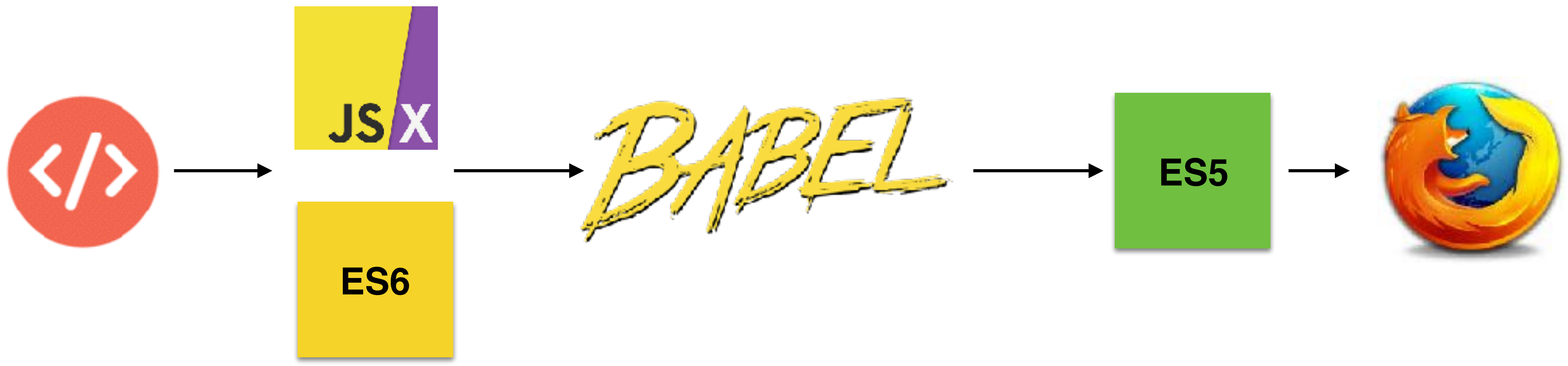


```
var x = <div> Hello World </div>;
```

```
var x = React.createElement(
  "div",
  null,
  " Hello World "
);
```

Hello World

(WhiteBoard)



ES6

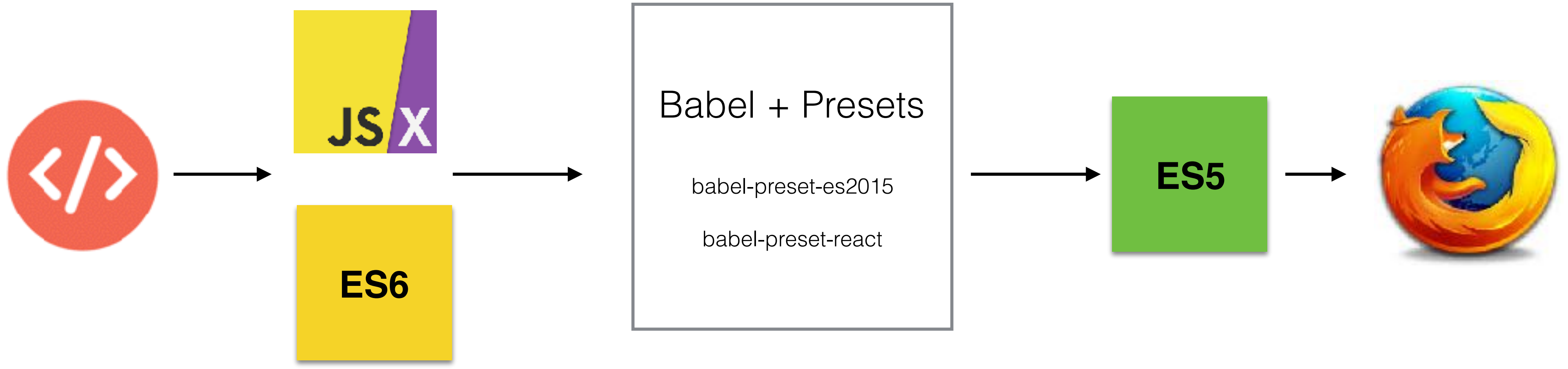
JSX

Babel

WebPack

Babel

- Babel uses presets (collection of plugins) to transpile code.
    - Eg one preset for JSX, one preset for ES6
        - babel-preset-es2015 [ES6 -> ES5]
        - babel-preset-react [JSX -> ES5]



Babel Demo

https://babeljs.io/repl/

```
var x = <div>Hello World</div>
```

Using Babel Manually

- Install babel-core babel-cli
- Install presets
  - babel-preset-es2015
  - babel-preset-react
- Set up .babelrc
- Alternative : Load it using babel-loader (Webpack)

Hello World

(WhiteBoard)

ES6

JSX

Babel

WebPack

Modules & Front-End Code

npm install —save-dev webpack

- We wish to use ES6 modules to better organise our React code
- **React code = Front-end code**
- Modules in back-end code - runs on server
  - Node understands modules
- Modules in front-end code - runs on browsers
  - **Problem - Browsers don’t understand modules!**

Solution : Module Bundling

- Write front-end / React code using Node/ES6 modules
- *Bundle* all module files together as one file - say - bundle.js
- Send bundle.js to browser
- To bundle all code - use a “module bundler

webpack

Hello World

(WhiteBoard)

Hands - On

- Install Node, Express
- Serve Static HTML file
- Install Babel - set up .babelrc
- Install WebPack - set up webpack.config.js
    - http://cb.lk/2qh81
- Install React
    - https://www.npmjs.com/package/react-dom
    - https://www.npmjs.com/package/react
- Create src/react.jsx
- Setup up index.html to use bundle.js

Summary

- Set up development environment for React
- JSX, ES6 - to write more productive/organised React code
  - JSX - HTML expressions
  - ES6 classes
  - ES6 modules
- WebPack, Babel - Generate browser-friendly React code
  - WebPack to load Babel Loader
  - WebPack to bundle modules (will explore in next class)
  - Babel to transform syntax for browsers
- Hello World