

React is a high-performance, reactive UI library for client-side web applications.



Syed Awase Khirni

RESEARCHER | ENTREPRENEUR | TECHNOLOGY COACH
@sak008 | sak@sycliq.com | +91. 9035433124

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project (www.geo-spirit.org). He currently provides consulting services through his startup www.territorialprescience.com and www.sycliq.com. He empowers the ecosystem by sharing his technical skills worldwide, since 2008.

Terms of Use

- You shall not circulate these slides without written permission from Territorial Prescience Research I Pvt Ltd.
- If you use any material, graphics or code or notes from these slides, you shall seek written permission from TPRI and acknowledge the author Dr. Syed Awase Khirni
- If you have not received this material, post-training session, you shall destroy it immediately and not use it for unauthorized usage of the material.
- If this material, has been shared to any organization prior to the training and the organization does not award the contract to TPRI, it should not use the training material internally. If by any chance, the organization is using this training material without written permission, the organization is liable to pay for the damages to TPRI and is subjected to legal action, jurisdiction being Bangalore.
- Without unauthorized usage, TPRI has right to claim damages ranging from USD 5000 to USD 10,000 dollars as damages.
- Any organization, which does not intend to go ahead with training or does not agree with the terms and conditions, should destroy the material from its network immediately. The burden of proof lies on the client, with whom this material has been shared.
- Recovery of the damages and legal fees will be born by the client organization, which has violated the terms and conditions.
- ReactJS is a copyright of Facebook
- Only candidates who have attended the training session in person from Dr. Syed Awase Khirni, TPRI are entitled to hold this training material.
- TPRI reserves all the rights to this material and code plays and right to modify them as and when it deems fit.
- If you agree with the terms and conditions, please go ahead with using the training material. Else please close and destroy the slide and inform TPRI immediately.



Slide Version Updates

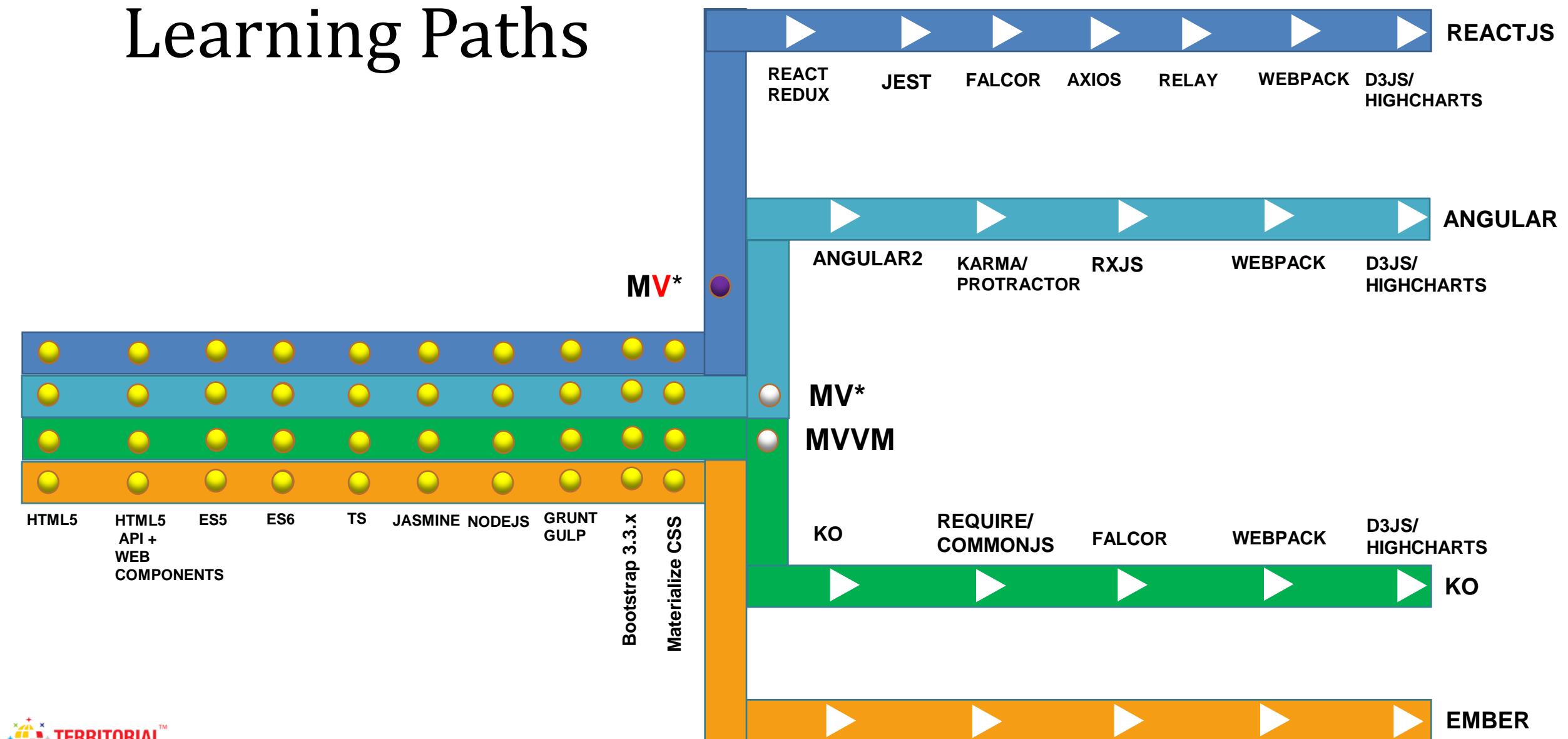
Last Updated	Version	Release Date	Updated by	Code Plays Done @
	0.13.		Syed Awase	
Oct 2016	15.0		Syed Awase	
April 2017	15.5	April 7 2017	Syed Awase	



React Version Updates

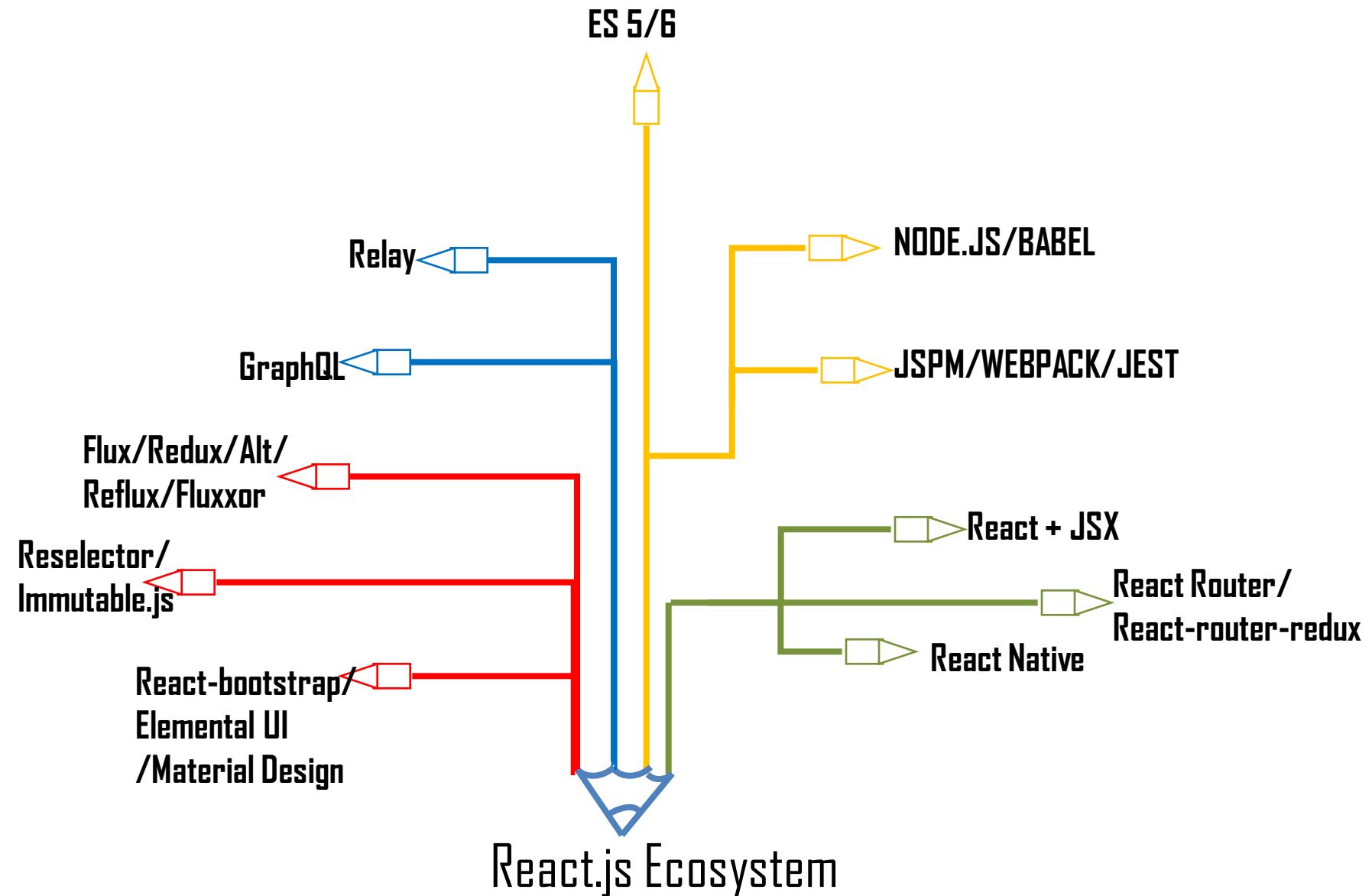
Last Updated	Version	Release Date	Updated by	Code Plays Done @
	0.13.			
Oct 2016	15.0			
7 April 2017	15.5.0	7 April 2017		

Learning Paths



Pre-requisite Knowledge

- HTML5 /CSS3
- Basic JavaScript
- jQuery
- Basic programming skills



Who's using React?



NETFLIX

YAHOO!

SBERBANK

facebook

Atlassian

KHANACADEMY

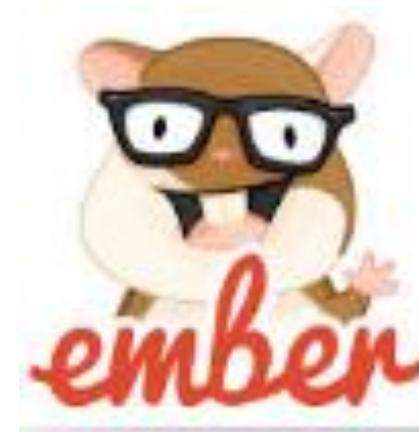
TESLA

323 Companies worldwide and counting!

Client-side frameworks



Size: 766k



Size: 435k

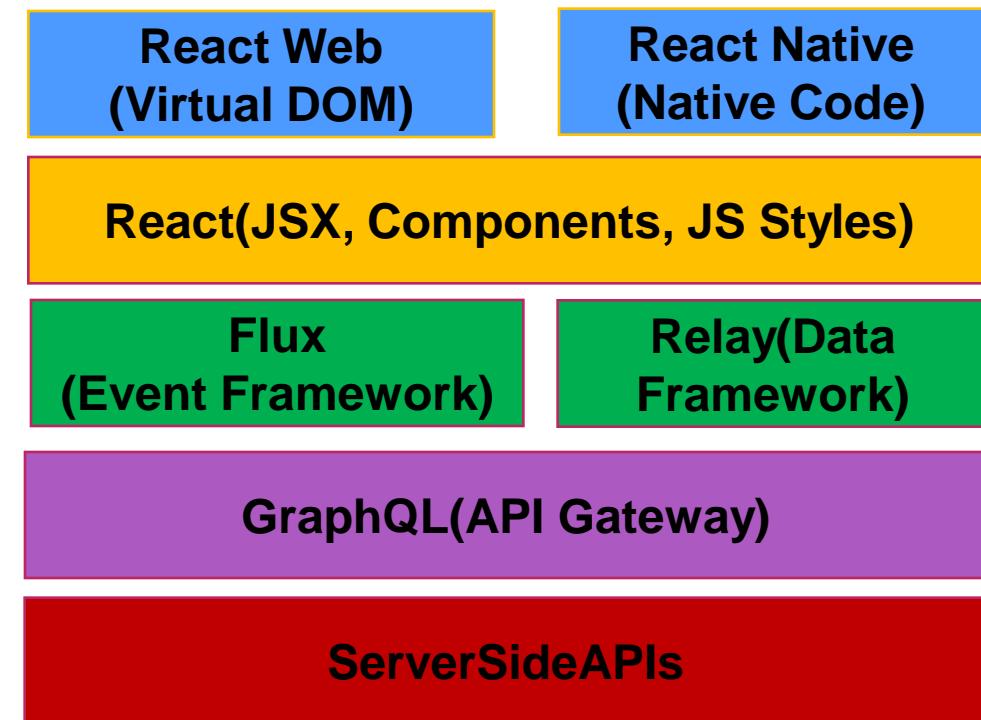


Size: 329k

Cover entire client-side requirements

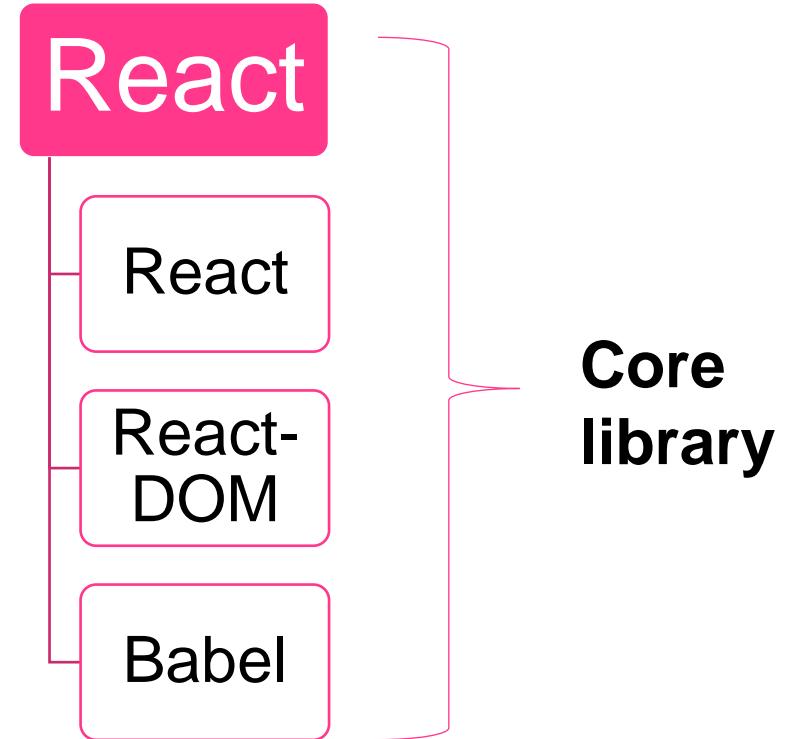
React Ecosystem

- React
- React Router (resource access through routing)
- WebPack (code bundling)
- Babel (code transformation)
- Axios (http requests)/ jQuery ajax



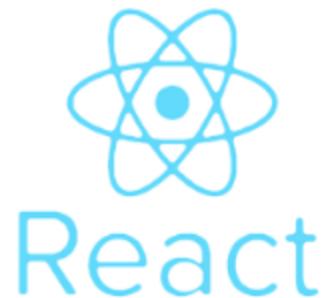
Workshop Structure

- ❑ JavaScript
- ❑ HTML5 Web Components
- ❑ ES6
- ❑ Node.js
- ❑ Webpack
- ❑ Require.js
- ❑ React.js with ES5
 - ❑ JSX
- ❑ React.js with ES6
- ❑



Package	Version	Dependencies
babel-core	npm v6.24.1	dependencies up to date
babylon	npm v6.17.0	dependencies none
babel-traverse	npm v6.24.1	dependencies up to date
babel-generator	npm v6.24.1	dependencies out of date

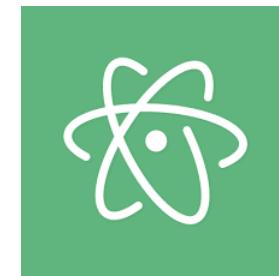
Development Tools



Babelify

Watchify

Babel-React-Preset



React Developer Tools

[offered by Facebook](#)

★★★★★ (253) | [Developer Tools](#) | 190,547 users

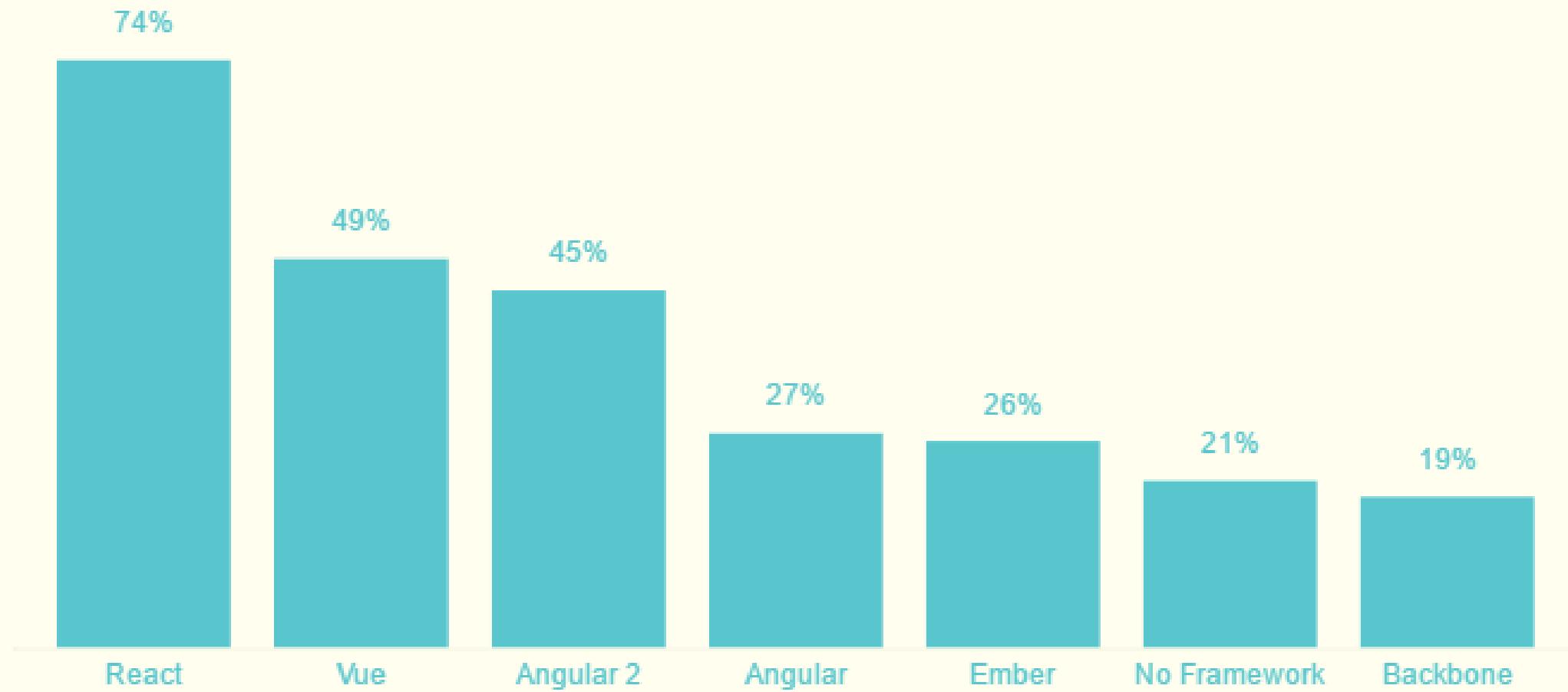
Development Tools

- Browserify : Allows the browser to use NPM modules
- Babel: compiles ES6/JSX into browser-readable javascript
- Babel-React-Preset : Tells Babel how to compile JSX
- Babelify: Allows you to use Babel with Browserify
- Watchify: Watches for changes and instantly starts compiling

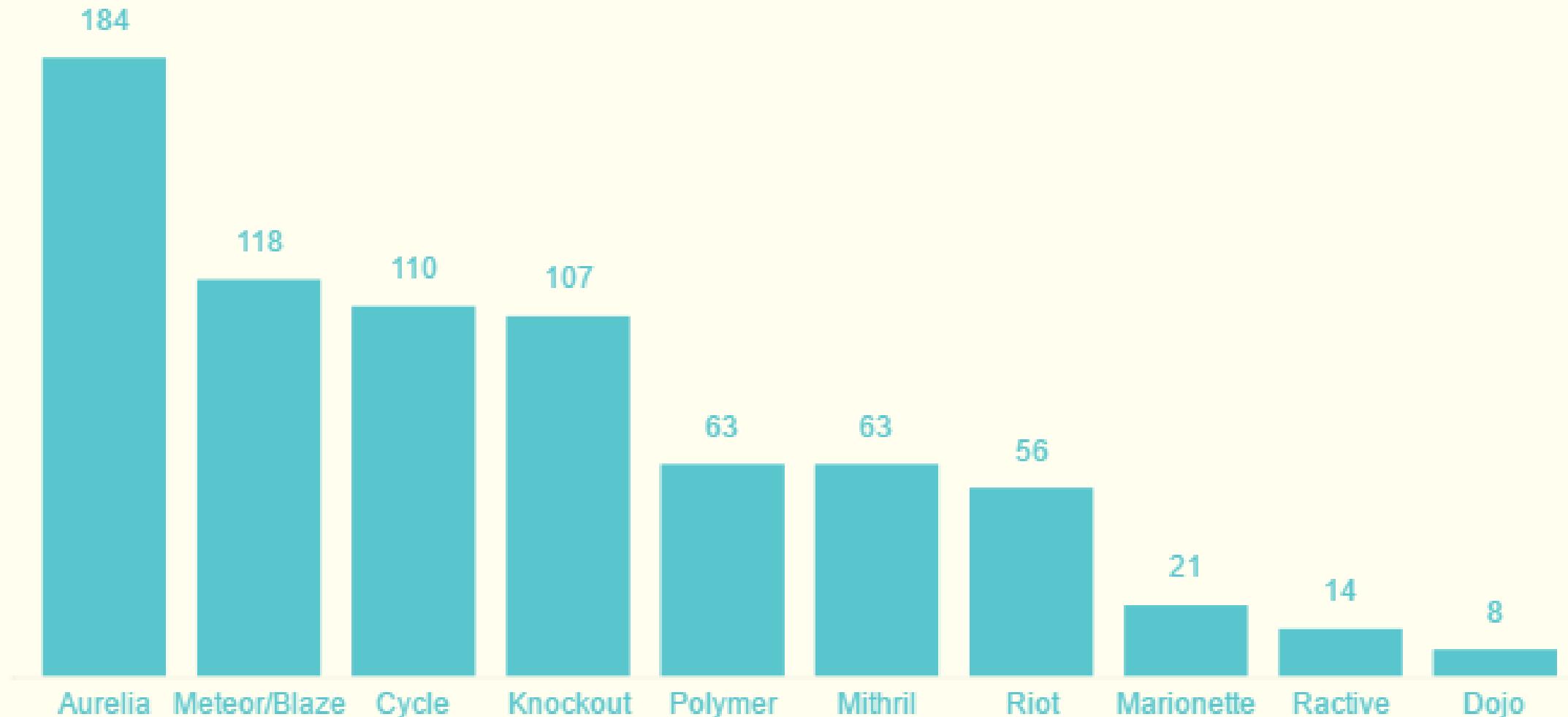


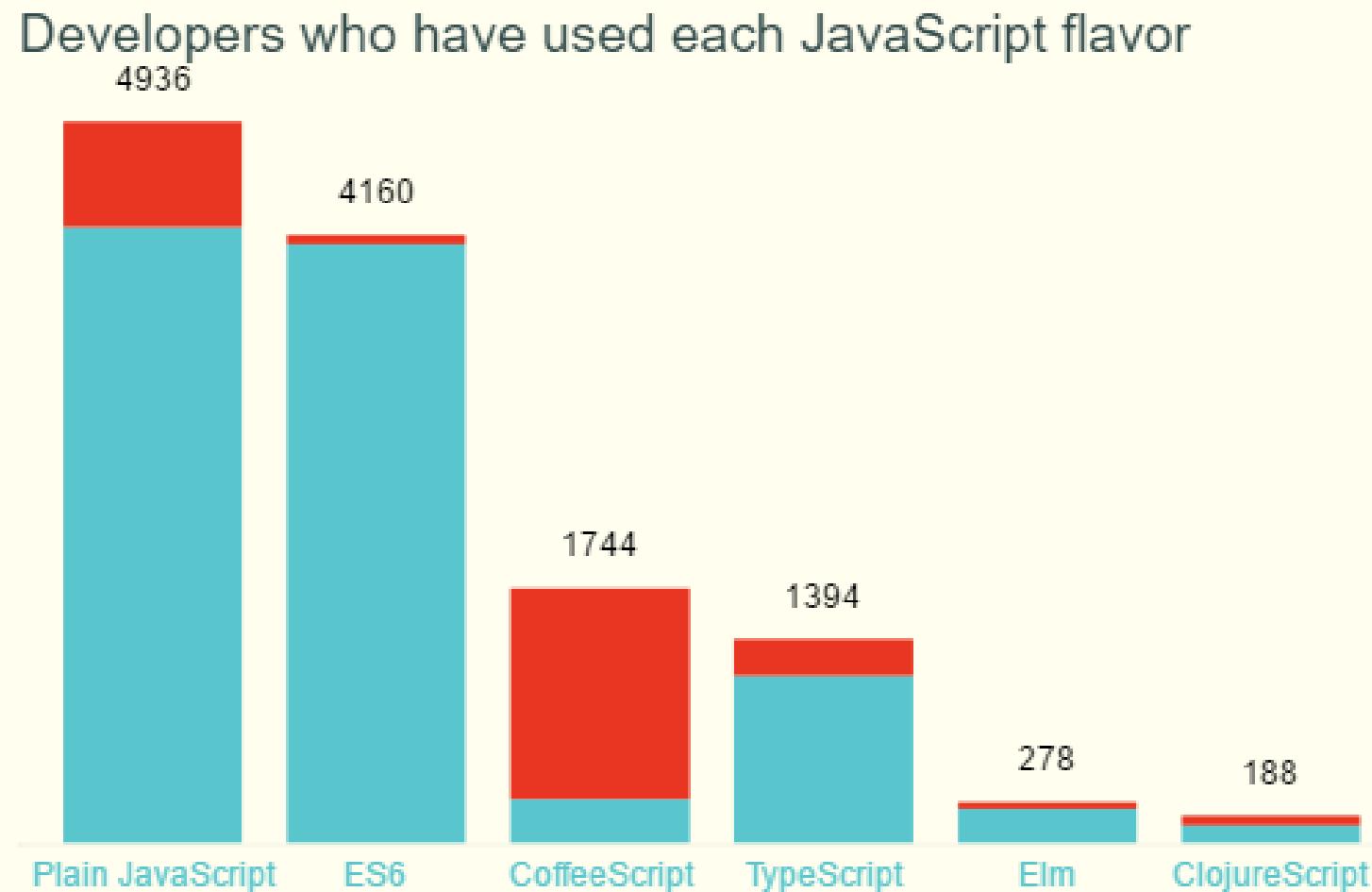
.NET	
Ruby	
Java	
Python	
PHP	
The Web	
Node	

Percentage of non-users who want to learn a given framework



Other front-end frameworks (mentions)





chartblocks

Out of 5328 total respondents (interactive graph)

© TPRI Syed Awase 2015-16 - ReactJS Ground Up!

Evolution

- Created by [Jordan Walker](#) at Facebook now working for reasonml
- 2010 facebook released an extension for PHP called XHP
- XHP helped to decrease XSS attack and make front-end both readable and understandable.
- Dynamic web applications require **many roundtrips to the server.**
- React Compatibility
 - works with any other framework
 - Angular
 - React-based Addons
 - Backbone
 - Ruby on Rails

Isomorphic Application

mERN

React provides
fast client-side
views

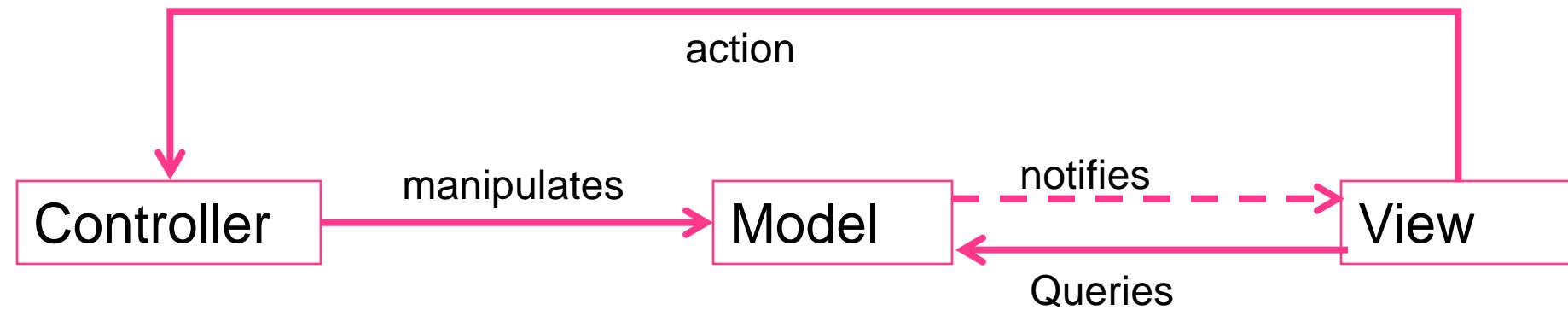


Express provides a
Powerful backend

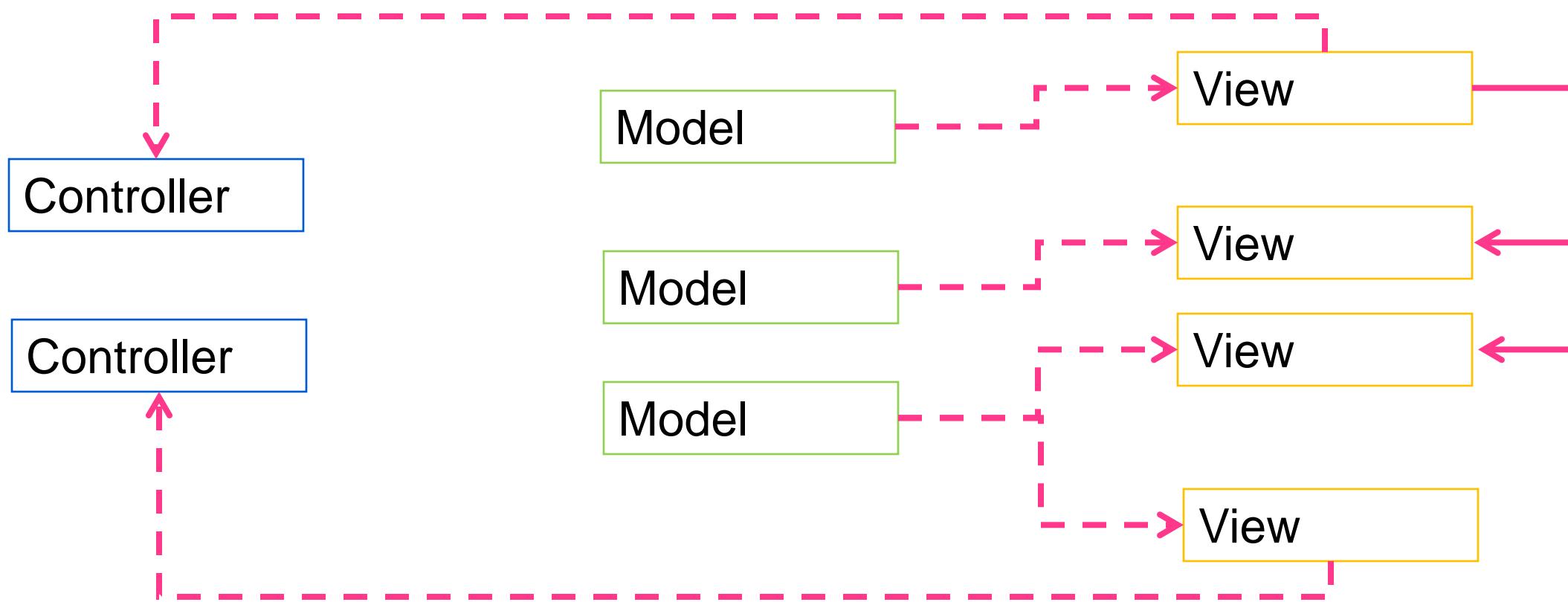
React and Comparison to other Libraries

- React and Knockout
 - Both provide
 - Rendering a model to the DOM
 - Events (publishing, handling and managing events)
- Angular JS and Ember JS are larger and more generalized with routing, built-in services
- Backbone is unique, library that does not provide declarative binding, developers have to imperatively define. Backbone is full comprehensive library to define our own custom framework. Infact , react is roughly comparable to use React as an alternative view component for Backbone.

MVC Application



Enterprise Class Application With MVC



- JSX
- Virtual-DOM
- Props
- PropTypes
- State
- Refs
- LifeCycle
- Flux Architecture
- Thinking in React
- Routing

Strict mode is supported in:

- IE from version 10. Firefox from version 4.
- Chrome from version 13. Safari from version 5.1.
- Opera from version 12.



React

- React focuses on **the rendering and event handling of client-side user interface components (HTML5 Web Components)** leaving the programmer with the responsibility and opportunity to choose other specialized components to complete their application.

- Open-source
- Client-side web library for building composable user interfaces developed by Facebook and Instagram.
- <http://facebook.github.io/react>
- Current version – React v15.3.1

React + Redux

Size: 151k



React is

- Declarative
- Unidirectional Data flow
- Composition
- Explicit Mutation
- “It’s just JavaScript”

- Data Binding in React
 - React utilizes a one way reactive data flow
 - Makes it much more simple than traditional 2 way data binding and also reduces boiler plate code
 - when properties are updated, components are **re-rendered**
 - **2-way data binding is possible with addons**

Programming Paradigm

- Four Main paradigms
 - Imperative
 - Describes the computation in terms of statements that change a program state. They are programming commands or mathematical assertions.
 - Declarative
 - Expresses the logic of computation(what to do) without describing its control flow (How to do) e.g. html, MXML,XAML, XSLT
 - Functional (sub-set of declarative)
 - Treats computation as the evaluation of mathematical functions and avoids state and mutable data, emphasizes on the application of functions, in contrast to the imperative style, which emphasizes changes in state. E.g: Haskell
 - Object-Oriented

Declarative

- We write code that describes what we want, but not necessarily how to get it (declare the desired results, but not the step-by-step)
- when you say *what* you want, i.e. not specifying the implementation details.
- This approach allows the compiler to make decisions that might result in better code.
- Declarative programming tells the machine what you would like to happen (and the computer figures out how to do it)
- CSS,HTML,XML,XSLT,RegX

```
// Declarative (What)

var numbers = [4,2,3,6]
numbers.reduce(function (previous, current)
  return previous + current
})
```

Imperative

- We tell the compiler what we want to happen, step by step
- when you say *how* to get what you want.
- Imperative programming tells the machine how to do something (resulting in what you want to happen)
- C/C++, Java, COBOL, FORTRAN, Perl, JavaScript

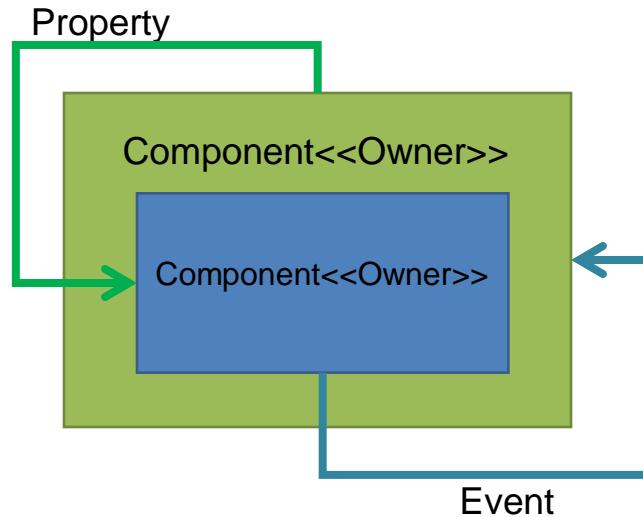
```
// Imperative (How)

var numbers = [4,2,3,6]
var total = 0
for (var i = 0; i < numbers.length; i++) {
  total += numbers[i]
}
```

Why Declarative?

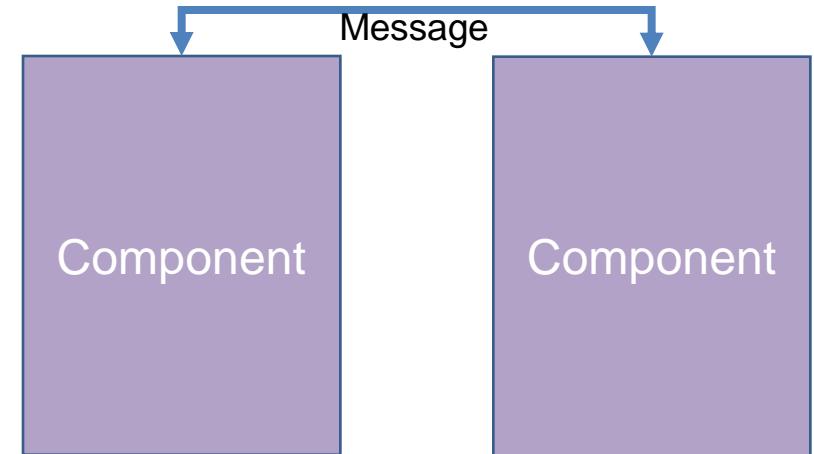
- Reduce Side Effects
- Minimize Mutability
- More readable code
- Less bugs
- React for the most part is declarative in nature.

Component Composition With ReactJS



- **Coupled Composition:** one component is the owner of another component. The owning component can be considered as a parent, while the owned component becomes a child. This kind of coupling leads to a typical component hierarchy with parent and children components.

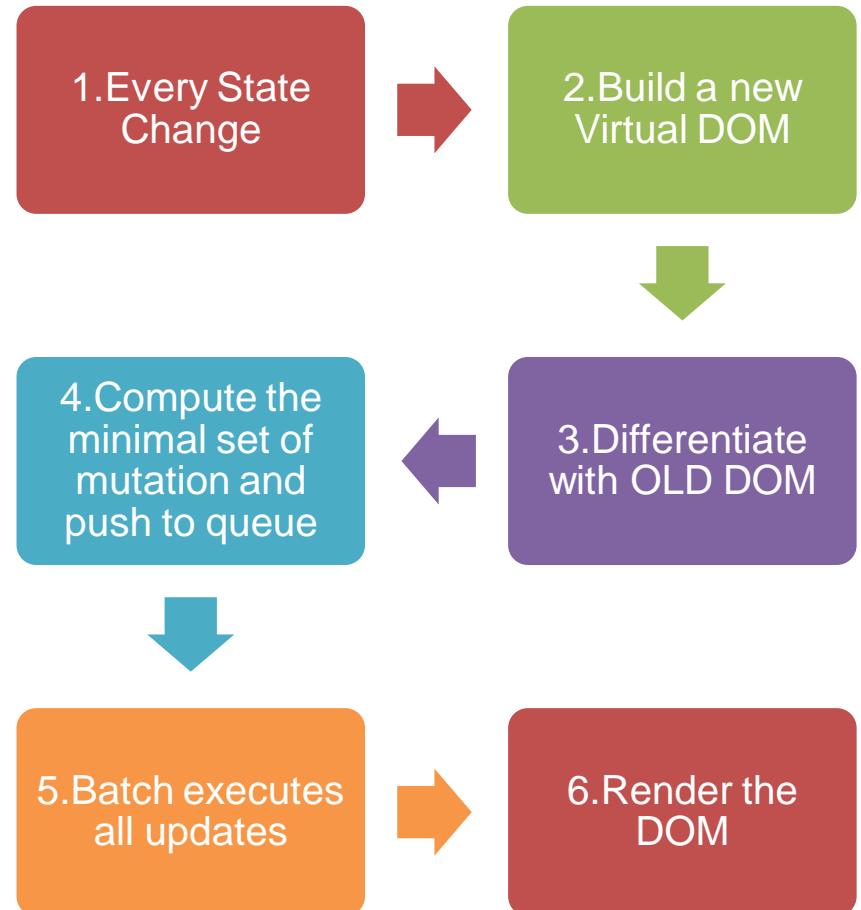
- **Most usual way to build up a UI with React.**



- **Decoupled Composition:** used when none of the components is owner, nor owned. These components exists on the same screen, nevertheless they are not bound or related to each other.
 - **Proportionally difficult to maintain, when the UIs complexity grows.**

Explicit Mutation

- Data that can be modified by user input or ajax call or time.
- When the state is changed, the component subtree is re-rendered., i.e. Whenever a component is dirty(its state changed), that component and its children are re-rendered.
- React doesn't compare state data. **When setState is called, it marks the component as dirty(needs to be re-rendered).**
- **Render method of the component is called, the real DOM is only updated if the output is different from current DOM tree.**



Unidirectional Data Flow

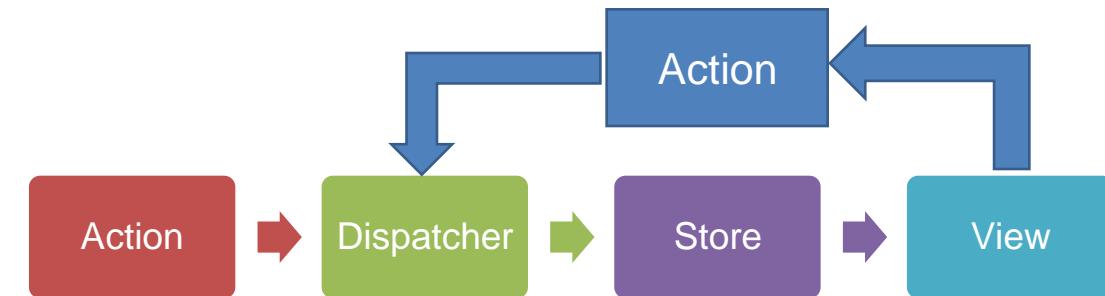
- A technique found in ***functional reactive programming, to ensure clean data flow architecture.*** The major benefit of this approach is that **data flows throughout your application in a single direction and user has better control over it.**
- Single Source of truth, where views are just functions of the application state. Change the state and your view changes.
- Flux takes a functional approach. The view is a function of the application state. As a result, when the state changes the view automatically re-renders itself. The same state produces the same view which give better predictability.
- Flux (pattern) is an application architecture for React's composable view components by utilizing a unidirectional data flow.

Action creators are helper methods, collected into a library, that create an action from method parameters, assign it a type and provide it to the dispatcher.

Every action is sent to all stores via the callbacks the stores register with the dispatcher.



After stores update themselves in response to an action, they emit a change event.



Special views called **controller-views**, listen for change events, retrieve the new data from the stores and provide the new data to the entire tree of their child views.

It's plain JavaScript

- React is plain javascript with JSX

```
var listItems = this.props.items.map(function(item, index){  
  return (  
    <li style={styles.listGroup}>  
      <button  
        style={styles.removeItem}  
        onClick={this.props.remove.bind(null, index)} />  
      <span>  
        {item}  
      </span>  
    </li>  
  )  
},bind(this));
```

React Native

- Allows the development of mobile applications using React.
- Transpiles React Code to the native application code , such as Objective-C for iOS applications

React Advantage

- Some of the areas where React fares well against other competing Javascript User Interface Libraries.
- **Mobile Rendering at 60 fps**
- **Impressive rendering algorithm that minimizes the amount of work**
- **React chooses a declarative style.**
- **A React application is a set of components (HTML5 web components), each of which declaratively defines a mapping between some state and the desired UI.**
- **React Components are self-contained units of functionality. They publish a simple interface that defines their inputs as properties and their outputs as callbacks.**

React vs Angular 2

React

- React is a library
- Typically pull a number of other libraries off the shelf to build a real application.
- Achieved by using libraries such as Redux, react-router, immutable and more.
- Can be written in ES5 and ES6, currently offers **3 different ways to declare components**.

Angular 2

- Angular 2 is a platform
- Provides significantly more opinions and functionality out of the box.
- Platform comes with routing engine, web calls, testing, dependency management.
- Enforced consistency
- Built using TypeScript

React vs Angular 2

React

- React tries to bring dependent javascript libraries to build a comprehensive solution solely maintained by YOU.
- The power of JSX – which compiles down to JavaScript. Markup and code are composed in the same file.
- Missing closing tag – fails at compile time, provides line number and unclosed tag mentioned.

Angular 2

- Structured, methodical reinvention of a mature and comprehensive framework.
- Less likely to churn in painful ways after release.
- Embraces web component's standard.
- Missing closing tag – fails at run-time, no line number provided and no unclosed tag mentioned.

React vs Angular 2

React

- **JSX won't compile if there is typo.**
- ReactJS is JavaScript Centric, while React puts “HTML” into JS.
- To reach react –learn Javascript

Angular 2

- Angular 2 remains HTML5 centric- Angular 2 continues to put “JS” into HTML.
- To read angular- learn a long list of angular-specific syntax

React vs Angular 2

Attribute	AngularJS	Angular 2	React
Version	1.5.0-rc1 / 1.49	2.0.0 - In Beta	0.14.6
Author	Google	Google	Facebook
Language	JavaScript/HTML	TypeScript	JSX
Size	143k	764k	151k
Github Stars	46.4k	8.4k	34.4k
Github Contributors	1,386	189	604

Attribute	AngularJS	Angular 2	React
Churn	Reduced	Reduced	High
Tooling	Low	High	High
Code Design	JS into HTML	JS into HTML	JavaScript Centric
JavaScript "Fatigue"	Less	Less	More

React vs Angular 2

Attribute	AngularJS	Angular 2	React
<i>DOM</i>	Regular DOM	Regular DOM	Virtual DOM
<i>Learning Curve</i>	High	Medium	Low
<i>Packaging</i>	Weak	Medium	Strong
<i>Abstraction</i>	Weak	Strong	Strong
<i>Debugging General</i>	Good HTML / Bad JS	Good JS/Good HTML	Good JS / Bad HTML
<i>Debug Line NO</i>	No	No	Yes
<i>Unclosed Tag Mentioned?</i>	No	No	Yes
<i>Fails When?</i>	Runtime	Runtime	Compile-Time
<i>Binding</i>	2 Way	2 Way	Uni-Directional
<i>Templating</i>	In HTML	In TypeScript Files	In JSX Files
<i>Component Model</i>	Weak	Strong	Medium
<i>Building Mobile?</i>	Ionic Framework	Ionic Framework	React Native
<i>MVC</i>	Yes	Yes	View Layer Only
<i>Rendering</i>	Client Side	Server Side	Server Side

Installation and Dev Environment



- Falcor :
<https://github.com/Netflix/falcor>
 - An efficient library for data fetching.
- ES6 Compiler : Babeljs.io
<https://babeljs.io/>
- Visual Studio Code or Sublime Text

A screenshot of the Chrome Web Store interface. It displays two extensions:

- React Developer Tools** by Facebook: Adds React debugging tools to the Chrome Developer Tools. It has a green "ADDED" badge, a green icon, and a rating of 4.2 stars (428 reviews).
- Redux DevTools** by remotedev.io: DevTools for Redux with actions history, undo and replay. It has a green "ADDED" badge, a colorful icon featuring two ants, and a rating of 4.5 stars (109 reviews).

Some Tips

1. React JS is a **view library**
2. Keep your components small
3. Write functional components either in ES5 or ES2015
4. Write Stateless Components
 - State makes components difficult to test
 - State makes components difficult to reason about
 - State make it too easy to put business logic in the component.
5. Use Redux Framework /Immutable.js/Redux-thunk/reselect
6. Always use propTypes -adds type safety to react components.
7. Shallow rendering – testing components as a single unit, without delving into any of its child components.
8. Use JSX, ES6, Babel, Webpack and NPM
9. Use React and Redux dev Tools



ES6 : ECMA Script 2015

SYED AWASE

Why ECMA 2015/ES6?

- Javascript has no standard library
- Javascript won't run outside the browser
- The DOM is too slow for rendering
- Javascript is single threaded by design
- Asynchronous programming -> callback hell
- Javascript has no packaging or a linker to tie packages together
- Web resources need to be minified and zipped for performance
- Javascript is too slow for videogames
- Machine generated output is more difficult to debug
- Poor performance on mobile devices.
- Ballooning project size and complexity.



<http://kangax.github.io/compat-table/es6/>

Can ES6 be used in browsers?

<http://pointedears.de/scripts/test/es-matrix/>



96%

<https://bugs.chromium.org/p/v8/issues/list?q=label:Harmony>

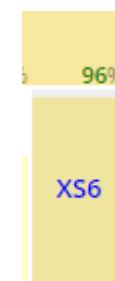


90%

https://bugzilla.mozilla.org/show_bug.cgi?id=694100



86%



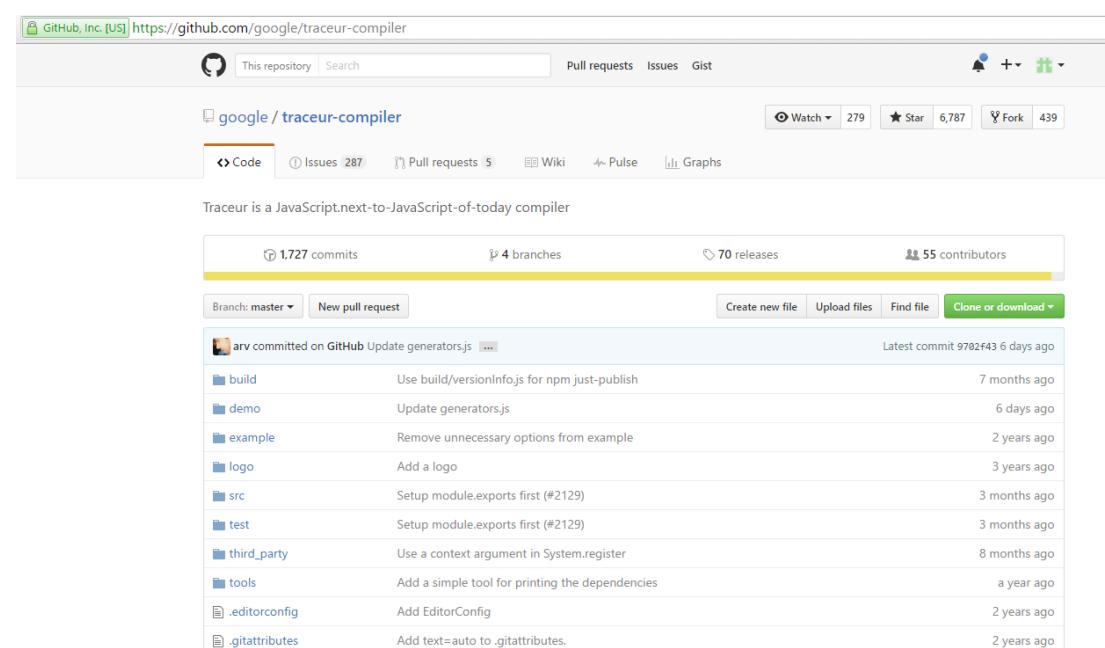
Transpilers

Babel



<https://babeljs.io/>

Traceur



The screenshot shows the GitHub repository for Traceur at <https://github.com/google/traceur-compiler>. The repository has 1,727 commits, 4 branches, 70 releases, and 55 contributors. The latest commit was made 6 days ago. The repository description is "Traceur is a JavaScript.next-to-JavaScript-of-today compiler".

Commit	Description	Date
arv committed on GitHub	Update generators.js	6 days ago
build	Use build/versionInfo.js for npm just-publish	7 months ago
demo	Update generators.js	6 days ago
example	Remove unnecessary options from example	2 years ago
logo	Add a logo	3 years ago
src	Setup module.exports first (#2129)	3 months ago
test	Setup module.exports first (#2129)	3 months ago
third_party	Use a context argument in System.register	8 months ago
tools	Add a simple tool for printing the dependencies	a year ago
.editorconfig	Add EditorConfig	2 years ago
.gitattributes	Add text=auto to .gitattributes.	2 years ago

<https://github.com/google/traceur-compiler>

ES6 New Features

- Classes
 - Modules
 - New Methods for Strings and Arrays
 - Promises
 - Maps, Sets
 - Completely new features
 - Generators
 - Proxies
 - WeakMaps
 - Reflect API
 - Proxy API
- **Static typing is not part of ES6.**
 - **Static typing is available in**
 - **Microsoft Typescript : transpiled to ES5 and throws away the type information while doing so.**
 - **Facebook Flow: A type checker for ECMAScript 6 that is based on flow analysis. It only adds optional type annotations to the language and infers and checks types. DOES NOT HELP IN COMPILING ES6 to ES5.**

Benefits of Static Typing

- Allows you to detect a certain category of errors earlier, because the code is analyzed statically (during development, without running code).
- Static typing is complementary to testing and catches different errors.

ECMAScript 6 Tools

Transpilers

Babel	Turns Es6+ code into Vanilla ES5 with no runtime
Traceur	ES6 Features -> ES5 includes classes, generators, promises, destructuring pattern, default parameters
ES6IFY	Traceur compiler wrapped as a Browserify V2 Platform
Babelify	Babel transpiler wrapped as a Browserify transform
Es6-transpiler	ES6-> ES5 Includes classes, destructuring, default parameters, spread
ES6-module-transpiler	ES6 modules to AMD or CJS
Regenerator	Transform ES6 yeild/generator functions to ES5
Jstransform	A simple utility for pluggable JS syntax transforms. Comes with a small set of Es6-> ES5 transforms
Defs	ES6 block-scoped const and let variables to ES3 vars
Es6_module_transpiler-rails	ES6 modules in the Rails Asset Pipeline

ECMA Script Tools

Transpiler	
Sweet.js	Macros that compile from ES6 to ES5
Bitovi's Transpile	Converts ES6 to AMD, CHS and StealJS
Regexpu	Transform unicode-aware ES6 regular expression to ES5
Lebab	Transformation for ES5 Code to ES6 (approximates)

Strict mode and ECMAScript 6

http://exploringjs.com/es6/ch_one-javascript.html

- Strict mode was introduced in ECMAScript 5 to clean up the language.
- Strict mode introduces three kinds of breaking changes
 1. Syntactic changes: some previously legal syntax is forbidden in strict model
 - **with - lets users add arbitrary objects to the chain of variable scope**
 - **Deleting an unqualified identifier**
 2. More errors
 - Assigning to an undeclared variable causes a ReferenceError
 - Changing read-only properties causes a TypeError
 3. Different semantics
 - Arguments doesn't track the current values of parameters
 - this is undefined in non-method functions

Choosing a package manager

- [npm](#): package manager that was originally created for Node.js, but has grown in popularity for client-side development thanks to module packaging and loading tools such as browserify and webpack. Packages contain CommonJS modules and/or other content such as command line tools.
- [Bower](#): package manager for client-side code. The most popular packages are AMD modules, but CommonJS modules, CSS, HTML and other artifacts can also be managed via it.
- [jspm](#): package manager for SystemJS (see next bullet list). It can install modules from a variety of sources, including GitHub and npm. One key feature of jspm is that external modules can also be written in ES6 (and will be transpiled), not just your own modules.

Choosing a module system

- [RequireJS](#): is a loader for AMD modules, which can be statically created via TypeScript, Traceur, Babel and Closure Compiler. *Loader plugins* (based on Traceur and Babel) enable it to load ES6 modules.
- [Browserify](#): packages CommonJS modules (including ones installed via npm) so that they can be loaded in browsers. Supports ES6 modules via *transforms* (plugins) based on Traceur and Babel.
- [webpack](#): a packager and loader for either CommonJS modules (including ones installed via npm) or AMD modules (including ones installed via Bower). Supports ES6 modules via *custom loaders* (plugins) based on Traceur and Babel.
- [SystemJS](#): A module system based on the ES6 Module Loader Polyfill that supports ES6 modules and the ES5 module formats CommonJS, AMD and “ES6 module loader API”.

Linters and Checkers

Linters and checkers analyze source code statically and report problems related to style, typing, etc.

- Linters
 - [JSLint](#) (focus: enforcing coding practices)
 - [JSHint](#) (focus: enforcing coding practices)
 - [ESLint](#) (focus: letting people implement their own style rules)
 - [JSCS](#) (focus: enforcing code style)
- Checkers
 - [Flow](#): type-checks code. It receives type information from three sources:
 - Type annotation syntax (non-standard)
 - [Type annotations in comments](#)
 - Type inference (which deduces types by statically analyzing source code, making Flow useful even for completely unannotated code)
 - [TypeScript](#): In addition to being a transpiler, the TypeScript compiler also works similarly to Flow and warns about type problems.
 - [Closure Compiler](#): type-checks code and understands type information stored in JSDoc tags.

Choosing a transpiler

- [Microsoft TypeScript](#): Is basically ECMAScript 6 plus optional type annotations.
- [Google Traceur](#): is the first popular ES6 transpiler.
- [Babel](#): is a newer ES6 transpiler that has become the de-facto standard. Babel supports React's JSX syntax in addition to ES6. Pronounced "babble" (think Australian accent – Babel's creator, Sebastian McKenzie is Australian).
- [Closure Compiler](#): can be used as a static transpiler from, e.g., ECMAScript 6 to ECMAScript 5, if you use [the following two command line options](#):
 - Specify the input language: `--language ECMASCIPT6_STRICT`
 - Specify the output language: `--language_out ECMASCIPT5`

Shims/polyfills

- Shims/polyfills enable you to use much of the ECMAScript 6 standard library in ES5 code:
 - [es6-shim](#)
 - [Core.js](#) (used by Babel)
- ES6 Parsers
 - [Esprima](#)
 - [Acorn](#)
 - Babel has also recently become more of a framework for statically analyzing and transforming JavaScript source code.

Running ES6

```
//installing traceur  
npm install --g traceur --save -dev  
//running your es6 file  
traceur --out build.js --script myfile.js  
//running build  
traceur build.js
```



<http://google.github.io/traceur-compiler/demo/repl.html#>

<http://www.es6fiddle.net/> <http://es6console.com/>

<http://babeljs.io/repl/#?babili=false&evaluate=true&lineWrap=false&presets=es2015%2Cract%2Cstage-2&code=>

New in ECMA 2015

- Arrow function
- Classes
- Modules
- Block Scope (let/const)
- Extended Object Literal
- Default Params
- Rest Params
- Spread Operator
- Destructuring
- Iterator
- Generator
- Template Literal
- Tail Call Optimization

New built-in classes and objects

- Promise
- Map
- Set
- WeakMap/WeakSet
- TypedArray
- Symbol
- Proxy/Reflect
- Improvement of existing classes
 - String
 - RegExp
 - Array
 - Object
 - Math
 - Number

Default export/import

- Export : expose a class so that others can use it by importing it

```
//import export example
function exim(){return 'exim';}
function eximbar(){return 'eximbar';}
//export - to expose the functions so that others can use
export {exim, eximbar};
```

- Import : import external classes to implement reusable functionality

```
import {exim, eximbar} from '1-Example';
exim();
eximbar();
```

```
import * as lib from '1-Example';
lib.exim();
lib.eximbar();
```

Let and Block scoping

Destructured assignment

- No Hoisting takes place when we use **let**, making sure that the variable declaration takes place before it is being used.

```
// block scoping without hoisting
let [firstson, secondson, thirdson] = [1,2,3];
let {firstplace, secondplace, thirdplace} = {firstplace:1, secondplace:2, thirdplace:3};

console.log(firstson, secondson, thirdson, firstplace, secondplace, thirdplace);

//Block scoping
let productId = 254;
{
  let productId=23213;
}
console.log(productId);
```

```
let a=[12,12,123,123,4123,145,1235,14123123];
let b= [543,623423,6234,234,63,232,23,2,4,2];
for (let i = 0; i < a.length; i++) {
  let x = a[i]
  console.log(x);
}

for (let i = 0; i < b.length; i++) {
  let y = b[i]
  console.log(y);
}

let callbacks = []
for (let i = 0; i <= 2; i++) {
  callbacks[i] = function () { return i * 2 }
  console.log(callbacks[i]());
}

callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
```

```
var updateFunctions =[];  
for (var i =0; i<2;i++){  
    updateFunctions.push(function(){return i;});  
}  
console.log(updateFunctions[0]());|  
  
//ES2015  
var updateFunctions =[];  
for (let i =0; i<2;i++){  
    updateFunctions.push(function(){return i;});  
}  
console.log(updateFunctions[0]());|
```

Const

- constant is a variable whose value cannot be changed.

```
//ES2015
const DB_CONNEC = "oracle db";
console.log(DB_CONNEC);
```

Block Scope in ES6

```
function updateEmployeeId(){
  employeeId = 123123;
}
let employeeId = null;
updateEmployeeId();
console.log(employeeId); ➔ 123
```

Temporal Dead ZONE

```
let itemId=2112;
for(let itemId=0; itemId<100; itemId++){
  console.log(itemId);
}
console.log(itemId); ➔ 2112
```

Arrow Functions

```
> document.addEventListener('click',
  ()=>console.log(this));
< undefined
```

VM223:2

*Window {speechSynthesis: SpeechSynthesis,
caches: CacheStorage, localStorage: Storage,
sessionStorage: Storage, webkitStorageInfo:
DeprecatedStorageInfo...}*

Arrow Functions

```
// we cannot bind the value of a function to a new value with =>
var employee={
    empid: 12123,
    callName: function(){
        return()=>console.log(this.empid);
    }
};
var newEmployee={
    empid: 51222
};
employee.callName().bind(newEmployee)(); //value 12123 is printed
```

Arrow Functions => (fat arrow symbol)

- they are shorthand form of anonymous function expression that already exist in javascript.
- *this* in arrow function uses lexical scoping, its value is always *inherited* from the enclosing scope.

```
const arrvalues = [41, 121, 51];
const squares = arrvalues.map(x => x * x);

console.log(squares);
```

```
let getPrice =()=> 99.99;
console.log(typeof getPrice);
console.log(getPrice());
```

```
//ES2015
var getTotal = noofitems=> noofitems*99.99;
console.log(getTotal(9));
```

Arrow Functions

```
//ES2015
let computeTax = (salary, tax)=> salary-(salary*tax);
console.log(computeTax(1000, 0.28));
```

```
//ES2015
let salesTax = (salary,tax)=>{
    let g_salary= salary+ salary*13.8;
    g_salary = g_salary -(g_salary*tax);
    return g_salary;
}
console.log(salesTax(1000,0.15));
```

String interpolation via Template literals

- We can use string interpolation and multi-line strings via template literals

ES5

```
function printCoord(x, y) {  
  console.log('('+x+', '+y+')');  
}
```

ES6

```
function printCoord(x, y) {  
  console.log(`(${x}, ${y})`);  
}  
  
const HTML5_SKELETON = `<!doctype html>  
<html>  
<head>  
  <meta charset="UTF-8">  
  <title></title>  
</head>  
<body>  
</body>  
</html>`;
```

Default Function Parameters

```
//ES2015
var stampEmployee = function(empId=1111, designation='SoftwareEngineer', salary='40000'){
    console.log(` ${empId} ${designation} ${salary}`);
};
stampEmployee(undefined, undefined, 45000);
```

```
//ES2015
var Payables = function(price, gst=price*0.15){
    console.log(price+gst);
};
Payables(799);
```

Default Function Parameters

```
//ES2015
let gstTax = 0.15;
let Payables = function(price, gst=price*gstTax){
    console.log(price+gst);
};
Payables(799);
```

Rest parameter in ES2015

- A ***rest parameter*** is indicated by three dots (...) preceding a named **which becomes an *Array containing the rest of the parameters passed to the function.***
- Allows users to specify that multiple Independent arguments should be Combined into an array.

```
//ES2015
var showCategories = function(itemId, ...categories){
  console.log(categories instanceof Array);
}
showCategories(123, 'electronic item', 'electronic appliance');
```

Spread operator in ES2015

- Allows you to specify an array that should be split and have its items passed in as separate arguments to a function.

```
//ES2015  
let tickets=[12,24,43,145];  
var maxTicket = Math.max(...tickets);  
console.log(maxTicket);
```

Object Literal Extensions

-

```
//object literal extension
var empid= 123214213, empName="Syed Awase", tax=0.18,salary=10000;
var employeeView={
    empid,
    empName,
    salary,
    tax,
    computeTax(){
        return (this.salary-this.salary*(1-tax));
    }
};

console.log(employeeView);
console.log(employeeView.computeTax());
```

Object Literal Extensions

```
//example3 object literal extension
var attrprps ='nationality';
var dualcitizenship = 'dual nationality';
var dualcitizen={
  [attrprps+"-0001"]:dualcitizenship
}
console.log(dualcitizen);
```

```
//example 4 with getter and setter
var id ='productId';
var productView ={
  get[id](){return true;},
  set[id](value){}
};
console.log(productView.productId);
```

For..of Loop

- Allows us to easily iterate over elements of a collection.
- For..of iterates over the values of elements of a collection not the keys.
- A collection can be an array, set, list, custom collection object etc.

```
//for..of loop example
var names = ['Syed Awase', 'Syed Azeez', 'Syed Rayyan', 'Syed Ameese'];
for(var name of names){
  console.log(name);
}

var places =[ 'Kashmir','Kanyakumari','Kalangote', 'karimnagar',];
for(var place of places){
  console.log(place);
}
```

Octal and Binary Literals

- In ES5, we had to use ‘0’ in front of the numbers to specify it’s a octal number., which often did not work when used in ***‘strict mode’***
 - “0o” is a new way of creating a number using octal value in ES 2015.
- “0b” lets you create a number using binary of the number directly.

```
//binary values
var b = 0b10; //alternative 0B10 gives same value
console.log(b);
```

```
//octal value
var i = 0o10;
console.log(i)
```

Destructuring

- makes it easier to work with objects and arrays in Javascript.
- Using a pattern syntax similar to object and array literal, we can poke into data structures and pick out the information we want into variables.

```
//destructuring
//object pattern matching
let {lName, fName}= {fName:'Awase Khirni', age:39, lName:'Syed'};
console.log(lName+","+fName);
//Array pattern matching
let feedback= [1,2,3,4,5];
let [stronglyDisagree, Disagree, Neutral, Agree, stronglyAgree] = feedback;
let [poor, ...remaining]=feedback;//rest parameters
console.log(Neutral);
//example 3
let salary=['17000', '50000'];
let [low,average, high="88000"]= salary;
console.log(high);
//example 4
let totalearnings = ['38000', '570231', ['123412512', '87712687']];
let [lowearning,averageearning, [monthlylow, monthlyhigh]]= totalearnings;
console.log(monthlylow);
```

Modules

- Modules are one of the most important features of any programming language. JS lacks this very basic feature. ES5, AMD was achieved either using commonJS or requireJS.
- In ES6 each module is defined in its own file.
- The functions or variables defined in a module are not visible outside unless you explicitly export them

- ES6 modules are declarative in nature.
- use
 - Export to `export`
 - Import to `import`

```
//es6-module loader
npm install -g es-module-loader --save
compile-modules convert -I scripts -o out app.js utility.js --format commonjs
compile-modules convert -I scripts -o out app.js utility.js //alternatively
//running the output
node out|
```

Basic Module Demo in ECMA 2015

Onemain.js

```
//onemain.js file using modules
import {projectId as id, projectName} from 'onemodeule.js';
console.log(` ${projectName} has id: ${id}`);
```

```
1 import * as values from 'twomodule.js';
2 console.log(values);
```

Onemodeule.js

```
//onemodeule.js file using modules
export let projectId=89798798798;
export let projectName ="SycliQ IOT Platform";
```

```
1 let projectId=124123;
2 let projectName = "SycliQ GeoAnalytic Platform";
3 export {projectId, projectName};
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>compile-modules convert -I scripts -o out onemain.js onemodeule.js
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>node out
SycliQ IOT Platform has id: 89798798798
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>
```

Named Exports in Modules

- A module can export multiple things by prefixing their declarations with the keyword `export`. These exports are distinguished by their names and are called **named exports**.

Error

```
//named export example one
import {projectId} from 'onemodule.js';
projectId=0912312312;
console.log(projectId);
```

```
//onemodule.js file using modules
export let projectId=89798798798;
export let projectName ="SycliQ IOT Platform";
```

Named Exports in Module

Fourmain.js

```
import {sycliqproject} from 'fourmodule.js';
sycliqproject.Id = 123541412312;
console.log(sycliqproject.Id);
```

Fourmodule.js

```
1 //named export example two
2 export let sycliqproject={
3     Id:3215123123
4 };
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>compile-modules convert -I scripts -o out fourmain.js fourmodule.js
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>node out
123541412312
```



Named Exports in Module

Fivemain.js

```
import {employee, showEmployee} from 'fivemodeule.js';
employee.employeeId = 'RD36785';
showEmployee();
console.log(employee.employeeId);
```

Fivemodeule.js

```
export let employee ={
    employeeId :`RD27865`,
    employeeName:'Syed Awase'
};

export function showEmployee(){
    console.log(employee.employeeId);
}
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>compile-modules convert -I scripts -o out fivemain.js fivemodeule.js
```

```
e:\CT\ReactJS\ES6-CodePlay\21-ModulesExample>node out
RD36785
RD36785
```

Class in ES2015

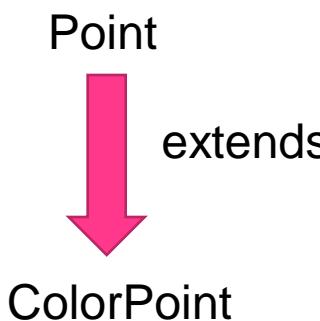
- ES6 classes are not something that is radically new, they are mainly provided with more convenient syntax to create old-school constructor functions.

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script student.js
```

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js
Student { id: 1, name: 'SyedAwase', age: 19, section: 'A', level: '+2' }
function
```

```
1  class Student{
2      //constructor of the class
3  constructor(id,name, age, section,level){
4      //this points to the current object
5      this.id =id;
6      this.name = name;
7      this.age =age;
8      this.section = section;
9      this.level = level;
10 }
11 //member functions
12 getId(){ return this.id;    }
13 getName(){ return this.name;   }
14 getAge(){ return this.age;    }
15 getSection(){ return this.section;   }
16 getLevel(){  return this.level;  }
17
18 setId(id){  this.id = id;    }
19 setName(name){  this.name = name;   }
20 setAge(age){ this.age = age;    }
21 setSection(section){  this.section = section;   }
22 setLevel(level){ this.level=level;    }
23 }
24 //creating a new instance of the class student
25 var aks = new Student(1,'SyedAwase',19,'A','+2');
26 console.log(aks);
27 console.log(typeof Student);
```

Extending Class in ES2015



```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script point.js  
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js  
(50,10) in red  
true  
true
```

```
1 class Point{  
2   constructor(x,y){  
3     this.x = x;  
4     this.y=y;  
5   }  
6  
7   toString(){  
8     return `(${this.x},${this.y})`;  
9   }  
10 }  
11 //extending the class point  
12 class ColorPoint extends Point{  
13   constructor(x,y, color){  
14     super(x,y);  
15     this.color = color;  
16   }  
17   toString(){  
18     return super.toString() + " in " + this.color;  
19   }  
20 }  
21  
22 let cp = new ColorPoint(50,10, 'red');  
23 console.log(cp.toString());  
24 console.log(cp instanceof ColorPoint);  
25 console.log(cp instanceof Point);
```

Class : Additional Insights

```
1 //function
2 function Project(){}
3 console.log(window.Project === Project); //true
4 //class
5 class SoftwareProject{
6
7 };
8 console.log(window.SoftwareProject === SoftwareProject); // false
9 //by creating a class we are not polluting the global context-window
```



Class :extends

```
1 class Human{
2     canBreathe(){
3         return 'can breathe';
4     }
5     canTalk(){
6         return 'can talk';
7     }
8 }
9 class Man extends Human{
10
11     canTalk(){
12         return 'little talk';
13     }
14 }
15 class Woman extends Human{
16     canTalk(){
17         return 'talks relatively more than Man';
18     }
19 }
20 let kapilsharma = new Man();
21 console.log(kapilsharma.canTalk());
22
23 let gutti = new Woman();
24 console.log(gutti.canTalk());
--
```

Class: super

```
1 let Project={  
2     getTaskCount(){return 500;}  
3 };  
4  
5 let AgileProject={  
6     getTaskCount(){  
7         return super.getTaskCount()+200;  
8     }  
9 }  
10 Object.setPrototypeOf(AgileProject, Project);  
11 console.log(AgileProject.getTaskCount());
```

```
1 class Project{  
2     getTaskCount(){  
3         return 1500;  
4     }  
5 }  
6 class ScrumProject extends Project{  
7     getTaskCount(){  
8         return super.getTaskCount()+300;  
9     }  
10 }  
11 let csm = new ScrumProject();  
12 console.log(csm.getTaskCount());
```

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script project.js
```

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js  
1800
```

Static keyword in ES2015

- static properties or class properties are properties of the class.

```
class Employee {  
    static getemployeeId(){  
        return 1231243;  
    }  
}  
  
console.log(Employee.getemployeeId());
```

new.target

- *meta property*
- Sole purpose is to retrieve the current value of the current function environment.
- It is a value that is set when a function is called
- mainly *used in a constructor*

```
1 class Employee{  
2   constructor(){  
3     console.log(typeof new.target);  
4   }  
5 }  
6  
7 var aks = new Employee();|
```

```
1 class Employee{  
2   constructor(){  
3     console.log(new.target);  
4   }  
5 }  
6 class SycliqEmployee extends Employee{  
7   constructor(){  
8     super();  
9   }  
10 }  
11 var sycliqsak = new SycliqEmployee();|
```

Symbols in ES2015

- A symbol is a unique and immutable data type and may be used as an identifier for object properties.
- Purpose of a symbol is to generate a unique identifier, but we never get to access the identifier.

```
1 let Book ={  
2   bookTitle: 'Fountain Head' ,  
3   [Symbol.for('bookAuthor')]: 'Ayn Rand'  
4 };  
5 console.log(Object.getOwnPropertyNames(Book));
```

Symbols enable access control for object state

Symbols are a new primitive type

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur --out build.js --script symbolexample2.js
```

```
e:\CT\ReactJS\ES6-CodePlay\22Classes>traceur build.js  
[ 'bookTitle' ]
```

CODE



```
1 let aks = Symbol("Syed Awase");
2 console.log(typeof aks);
3 console.log(aks.toString());
4 //symbols used with constants
5 const S_AK =Symbol("Syed Awase Khirni");
6 console.log(S_AK.toString());
7 let sak1 = Symbol("Syed Rayyan Awais");
8 let sak2 = Symbol("Syed Rayyan Awais");
9 console.log(sak1==sak2);
10 let sak1_addr = Symbol.for("Syed Rayyan Awais");
11 let sak2_addr = Symbol.for("Syed Rayyan Awais");
12 console.log(sak1_addr.toString());
13 console.log(sak2_addr.toString());
14 console.log(sak1_addr==sak2_addr);
15 let desc = Symbol.keyFor(sak1_addr);
16 console.log(desc);
```

CONSOLE

```
symbol
Symbol(Syed Awase)
Symbol(Syed Awase Khirni)
false
Symbol(Syed Rayyan Awais)
Symbol(Syed Rayyan Awais)
true
Syed Rayyan Awais
```

```
1 let Book ={
2   bookTitle: 'Fountain Head' ,
3   [Symbol.for('bookAuthor')]: 'Ayn Rand'
4 };
5 console.log(Object.getOwnPropertyNames(Book));
6 console.log(Object.getOwnPropertySymbols(Book)); //new method in ES6
```

Well-known Symbol

```
//well known Symbol
let Vehicle= function(){
};

Vehicle.prototype[Symbol.toStringTag]='Bare Bones';
let truck = new Vehicle();
console.log(truck.toString());
```

```
let carManufacturers = ['mercedes','hyundai','audi','bmw','fiat'];
console.log([] .concat(carManufacturers));
```

```
1 //wellknown Symbol
2 let v1 = [12,24,45,56,78];
3 v1[Symbol.toPrimitive]= function(hint){
4     console.log(hint);
5     return 99;
6 };
7 let total = v1+100;
8 console.log(total);
```

default

199



Object Extensions

```
1 //object extensions example
2
3 let m={
4     x:786
5 };
6
7 let n={
8     y:687
9 };
10 Object.setPrototypeOf(m,n);
11 console.log(m.y);
```

687

```
//Object.assign
let s={a:347}, q={b:743};
let target={};
Object.assign(target,s,q);
console.log(target);
//object.is
let amount =0, total=-0;
console.log(Object.is(amount,total));
```

String Extensions

```
//string extensions example
let bookTitle = "Exploratory Representations for Geographic Information Retrieved from the Internet";
console.log(bookTitle.startsWith('Exploratory'));//returns true
console.log(bookTitle.endsWith('Internet'));
console.log(bookTitle.includes('ra'));
```



Number Extensions

```
1 console.log(Number.parseInt === parseInt);
2 console.log(Number.parseFloat === parseFloat);
3 let no='NaN';
4 console.log(isNaN(no));//es5
5 console.log(Number.isNaN(no));//es2015
6 let x='78126';
7 console.log(isFinite(x));//es5
8 console.log(Number.isFinite(x));//es2015
9 let total = 782139.2;
10 console.log(Number.isInteger(total));
11 console.log(Number.isInteger(NaN));
12 console.log(Number.isInteger(Infinity));
13 console.log(Number.isInteger(undefined));
14 console.log(Number.isInteger(10));
15 let a = Math.pow(2,53) -1;
16 console.log(Number.isSafeInteger(a));
17 a=Math.pow(2,53);
18 console.log(Number.isSafeInteger(a));
19 //new constants
20 console.log(Number.EPSILON);
21 console.log(Number.MAX_SAFE_INTEGER);
22 console.log(Number.MIN_SAFE_INTEGER);
```

true
true
true
false
true
false
false
false
false
false
true
true
false
2.220446049250313e-16
9007199254740991
-9007199254740991

Math Extensions

Hyperbolic Functions	
Cosh()	
Acosh()	
Sinh()	
Asinh()	
Tanh()	
Hypot()	

Arithmetic Functions	
Cbrt()	Cube root
Clz32()	Count leading zeros (32 bit integer)
Expm1	Equal to exp(x) - 1
Log2()	Log base 2
Log10()	Log base 10
Log1p()	Equal to log(x+1)
Imul()	32 bit integer multiplication

Sign()	The number' sign: 1, -1, 0, -0, NaN
Trunc()	The integer part of a number
Round()	Round to nearest 32 bit floating point

Math Extensions

```
1 console.log(Math.sign(0));  
2 console.log(Math.sign(-0));  
3 console.log(Math.sign(-20));  
4 console.log(Math.sign(20));  
5 console.log(Math.sign(NaN));
```

```
0  
0  
-1  
1  
NaN
```

Regular Expressions in ES 2015

```
1 //use u at the end of pattern to check for astral patterns
2 let pattern = /\u{1f3c4}/u;
3 console.log(pattern.test(``));
4
5 let newpattern = /^.Surfer/u;
6 console.log(newpattern.test('Surfer'));
7
8 //performing the search from the last index using 'y'
9 let numero = /900/y;
10 console.lof(numero.lastIndex);
11 numero.lastIndex=3;
12 console.log(numero.test('7123123900')));
13 let num=/900/yg;
14 console.log(pattern.flags);
```

```
false
false
```



Function Extensions

```
1 //function expressions
2
3 let fn = function employee(){
4   return "employee";
5 };
6 console.log(fn.name);
7 //anonymous function
8 let fnone = function (){
9   return "book";
10};
11 console.log(fnone.name);
12 let fntwo = fnone;
13 console.log(fntwo.name);
```

```
employee
fnone
fnone
```

```
1 //class expressions
2
3 class Employee{
4   constructor(){
5   }
6   add(){
7   }
8 }
9
10 let sak = new Employee();
11 console.log(Employee.name);
12 console.log(sak.add.name);
```

```
Employee
add
```

Iterators in ES2015

- Iterators are used to iterate through a list of collection.

```
1 //iterators
2
3 let names = ['syed awase', 'syed ameese', 'syed azeez'];
4 console.log(typeof names[Symbol.iterator]);
5 let iter = names[Symbol.iterator]();
6 iter.next();
7 console.log(iter.next());
8
9 //iterator using function and spread operator
10 function process(name1,name2,name3){
11   console.log(name3);
12 }
13 process(...names);
```

```
function
[object Object]
syed azeez
```

Generators in ES2015

- A generator function is a special type of function that when invoked automatically generates a special iterator called a **generator**
- Generator functions are indicated by ***function**** and make use of the **yield** operator to indicate the value to return for each successive calls to **.next()** on the generator.

```
1 function *process(){
2   yield 8000;
3   yield 8001;
4 }
5 let val = process();
6 console.log(val);
7 console.log(val.next());
8 val.next();
9 console.log(val.next());
10 //generator with a loop
11 function *newprocess(){
12   let nextId = 9000;
13   while(true)
14     yield(nextId++);
15 }
16 for(let id of newprocess()){
17   if(id > 9010)break;
18   console.log(id);
19 }
```

```
[object Generator]
function (arg) { return this._invoke(method, arg); }
[object Object]
9000
9001
9002
9003
9004
9005
9006
9007
9008
9009
9010
```

Generators in ES2015

```
1 function* range(start, count) {
2     for (let delta = 0; delta < count; delta++) {
3         yield start + delta;
4     }
5 }
6
7 for (let highschoolyears of range(13, 7)) {
8     console.log(`High School Years start from ${highschoolyears}!`);
9 }
```

High School Years start from 13!
High School Years start from 14!
High School Years start from 15!
High School Years start from 16!
High School Years start from 17!
High School Years start from 18!
High School Years start from 19!

throw and return

```
1 function* valueProcess(){
2     try{
3         yield 1001;
4         yield 1002;
5         yield 1003;
6         yield 1004;
7         yield 1005;
8     }catch(e){
9     }
10}
11
12 let it = valueProcess();
13 console.log(it.next().value);
14 console.log(it.throw('error message'));
15 console.log(it.next());
16 console.log(it.next().value);
17 //only works in firefox
18 console.log(it.return('error message'));
```

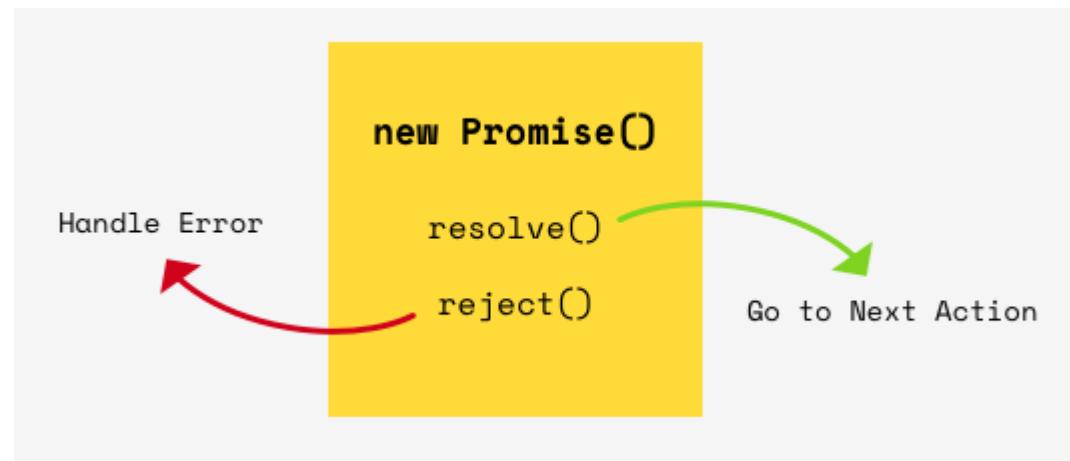
```
1001
[object Object]
[object Object]
[object Object]
```

Promises

- A Promise specifies some code to be executed later(as with events and callbacks and also explicitly indicates whether the code succeeded or failed at its job.
- You can chain promises together based on success or failure in ways that make your code easier to understand and debug.
- JavaScript engines can only execute one piece of code at a time, so they need to keep track of code that meant to run.
- ES6 has native support for promises. In ES5, we used to achieve this using jquery or \$q libraries.

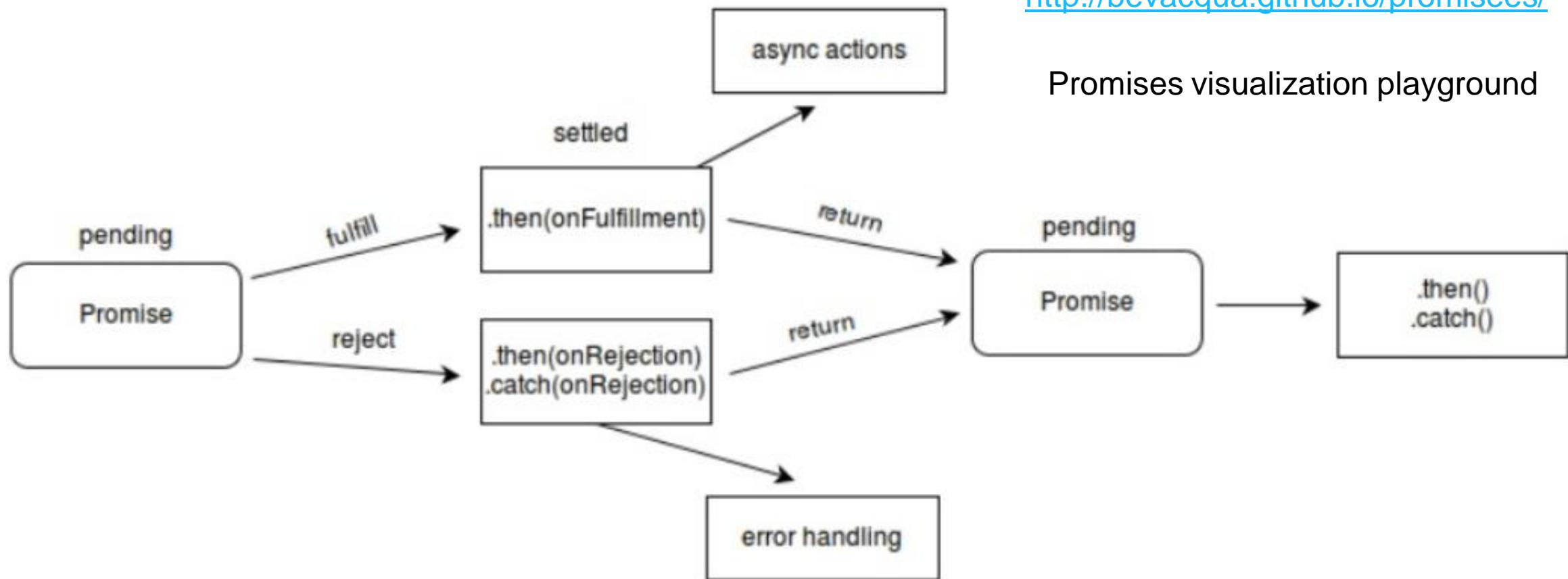
Promises in ES2015

- used for deferred and asynchronous computations. A Promise represents an operation that hasn't completed yet, but is expected in the future.



- Promise has **three** states
 - Pending : initial state, not fulfilled or rejected.
 - Fulfilled: meaning that the operation completed successfully
 - Rejected: meaning that the operation failed.

Promise life cycle in ES2015

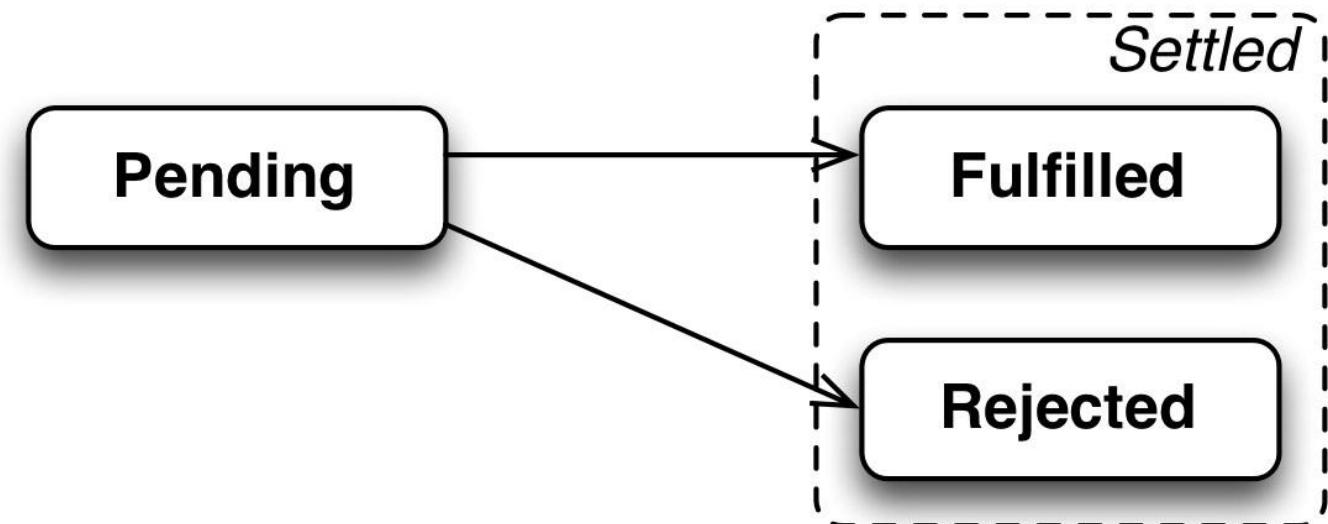
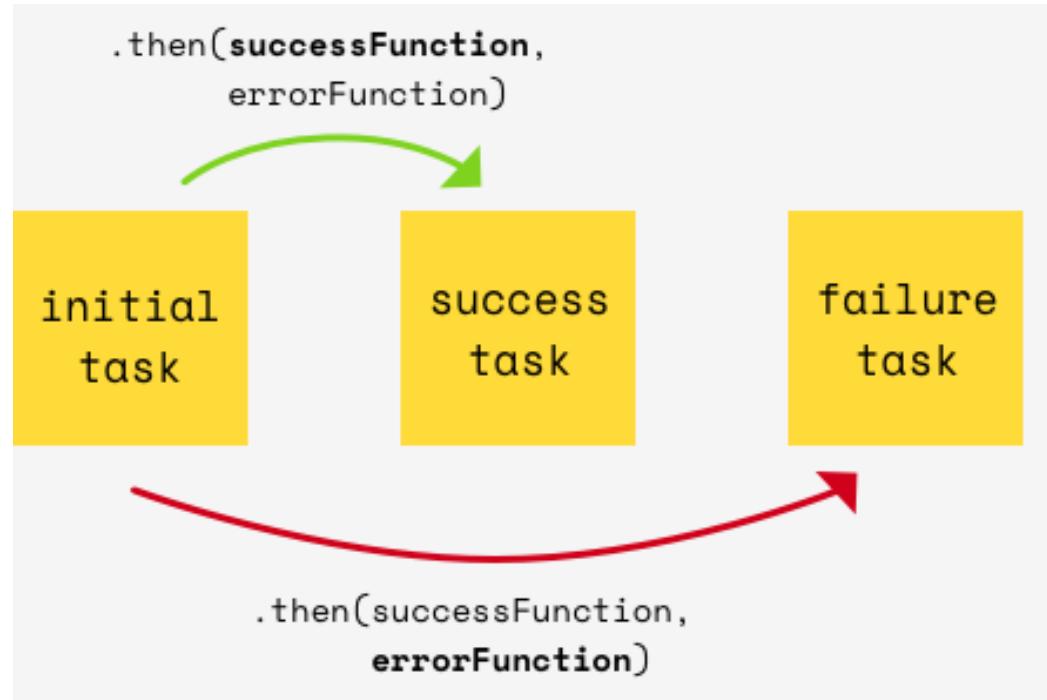


<http://bevacqua.github.io/promisees/>

Promises visualization playground

<https://ponyfoo.com/articles/es6-promises-in-depth>

Promises in ES2015



Creating a Promise

```
var p = new Promise(function(resolve,reject){
  if(/* condition*)){
    resolve(/* value*/); //fulfilled successfully
  }
  else{
    reject(/* reason*/); //error, rejected
  }
});|
```



Promise

```
// A Promise that throws, rather than explicitly reject
var p1 = new Promise((resolve, reject) => {
  if (true)
    throw new Error("rejected!"); // same as rejection
  else
    resolve(4);
});

// trailing .catch() handles rejection
p1.then((val) => val + 2)
  .then((val) => console.log("got", val))
  .catch((err) => console.log("error: ", err.message));
// => error: rejected!
```



Promise

```
new Promise(function(resolve, reject) {
  // A mock async action using setTimeout
  setTimeout(function() { resolve(10); }, 3000);
})
.then(function(num) { console.log('first then: ', num); return num * 2; })
.then(function(num) { console.log('second then: ', num); return num * 2; })
.then(function(num) { console.log('last then: ', num);});

// From the console:
// first then: 10
// second then: 20
// last then: 40
```

```
first then: 10
second then: 20
last then: 40
```



```
const all = (...promises) => {
  const results = [];

  const merged = promises.reduce(
    (acc, p) => acc.then(() => p).then(r => results.push(r)),
    Promise.resolve(null));

  return merged.then(() => results);
};

const p1 = Promise.resolve('Game of Throne: Dragon Queen is Promised the 7 kingdoms by birth right');
const p2 = '----- 2---Game of Throne: Ramzi Bolton-Bastard Son is Promised the Warden of North';
const p3 = new Promise((res, rej) => { setTimeout(res, 100, `-----3---Game of Throne: John Stark
- Fights for justice for his family and regains the title as King in the North` )});

all(p1, p2, p3)
  .then(([a1, a2, a3]) => console.log(a1, a2, a3));

const p4 = Promise.reject('---4---Game of Throne: High Sparrow is burnt alive in public');

all(p1, p2, p3, p4)
  .then(([a1, a2, a3]) => console.log(a1, a2, a3))
  .catch(e => console.log(e));
```

Promise : as a wrapper

```
1 // Creating a promise wrapper for setTimeout
2 function wait(delay = 0) {
3     return new Promise((resolve, reject) => {
4         setTimeout(resolve, delay);
5     });
6 }
7
8 // Using a promise
9 wait(3000)
10    .then(() => {
11        console.log('3 seconds have passed!');
12        return wait(2000);
13    })
14    .then(() => {
15        console.log('5 seconds have passed!');
16        x++; // ReferenceError triggers `catch`
17    })
18    .catch(error => {
19        // output: ReferenceError
20        console.log(error);
21    })
22    .then(() => {
23        // simulate `finally` clause
24        console.log('clean up');
25    });
}
```

```
3 seconds have passed!
5 seconds have passed!
ReferenceError: x is not defined
clean up
```

Promise.all

- There are times when we trigger **multiple async interactions but only want to respond when all of them are completed.**
- **Promise.all method takes an array of promises and fires one callback once they are ALL resolved.**

```
Promise.all([promise1, promise2]).then(function(results) {  
  // Both promises resolved  
})  
.catch(function(error) {  
  // One or more promises was rejected  
});
```

Promises Example

```
const promisedTexts= "Game of Thrones: I am promised to be the King in the North";  
  
Promise.all(promisedTexts)  
.then(texts => {  
    for (const text of texts) {  
        console.log(text);  
    }  
})  
.catch(reason => {  
    console.log("Error to read the promised scroll");  
});
```

Promise.race

- Instead of waiting for all promises to be resolved or rejected, **Promise.race** triggers as soon as any promise in the array is **resolved or rejected**.

```
var req1 = new Promise(function(resolve, reject) {
  // A mock async action using setTimeout
  setTimeout(function() { resolve('I am the first Born'); }, 8000);
});
var req2 = new Promise(function(resolve, reject) {
  // A mock async action using setTimeout
  setTimeout(function() { resolve('I am the second Born'); }, 3000);
});
Promise.race([req1, req2]).then(function(one) {
  console.log('Then: ', one);
}).catch(function(one, two) {
  console.log('Catch: ', one);
});
```

Then: I am the second Born

ES2015 Promises

1. Promises give us the ability to write asynchronous code in a synchronous fashion, with flat indentation and a single exception channel.
2. Help us unify asynchronous APIs and allow us to wrap non-spec compliant Promise APIs or callback APIs with real Promises.
3. Promises give us guarantees of no race conditions and immutability of future value represented by the Promise.
4. Promises cannot be cancelled, once created it will begin execution, if you don't handle rejections or exceptions, they get swallowed.
5. No way to determine the state of a Promise
6. For Recurring values or events, we can use **streams mechanism/pattern**.

Callbacks

1. Callbacks are functions.
2. Just blocks of code which can be run in response to events such as timers going off or messages being received from the server. Any function can be a callback, and every callback is a function
3. They are defined independently of the functions they are called from, they are passed in as arguments. These functions then store the callback and call it when the event actually happens.

Promises

1. Promises are objects.
2. They are objects which store information about whether or not those events have happened yet, and if they have, what their outcome is.
3. They are created inside of asynchronous functions (those which might not return a response until later) and then returned. When an event happens, the asynchronous function will update the promise to notify the outside world.

Callbacks

4. They can be called multiple times by the functions they are passed to.

Promises

4. They can only represent one event- they are either successful once or failed once

Array Extensions

- Array.of
 - is exactly an incarnation of Array Method, similar to ES5.

```
//es5 arrays strange behaviour
let val= Array(100);
console.log(val.length);
//es6 array behaviour as it should be
let es6val = Array.of(100);
console.log(es6val.length);
```

100
1

- Array.from
 - It has 3 arguments, but only the *input is required*.
 - *Input – the arraylike or iterable object you want to cast*
 - *Map – a mapping function that's executed on every item of input*
 - *context – this binding to use when called map.*

```
let scores = [657,785,857,924];
let subjectScore = Array.from(scores, v=>v+20);
console.log(subjectScore);
```

677,805,877,944

Array Extensions

Fill

```
//fill method for an array  
let taxVal =[875,895,9125,1754];  
taxVal.fill(19000);  
console.log(taxVal);  
//start filling from the arra[2]  
taxVal.fill(22000,2);  
console.log(taxVal);  
//array filling with start and stop  
taxVal.fill(25000, 1,2);  
console.log(taxVal);  
// start counting from the end of the array  
taxVal.fill(35000, -1);  
console.log(taxVal);
```

```
19000,19000,19000,19000  
19000,19000,22000,22000  
19000,25000,22000,22000  
19000,25000,22000,35000
```

Find

- Only returns the first value it finds in the array

```
//find method to search for values in an array  
let taxVal =[875,895,9125,1754];  
let result = taxVal.find(value=>value >= 9000);  
console.log(result);
```

9125

Array Extensions

```
1 //find method to search for values in an array
2 let taxVal =[875,895,9125,1754];
3 let result = taxVal.find(value=>value >= 9000);
4 console.log(result);
5
6 //findIndex method
7 let searchResult= taxVal.findIndex(function(value
8   return value==this;
9 },9125);
10 console.log(searchResult);|
```

9125
2

```
//find method to search for values in an array
let taxVal =[875,895,9125,1754];
//copyWithin(destinationPoint, sourceIndex)
taxVal.copyWithin(2,0);
console.log(taxVal);
taxVal.copyWithin(1,3);
console.log(taxVal);|
```

875,895,875,895
875,895,875,895

Array Extensions

```
let numeroUno=[11,12,13,14,15,16,17,18,19];
//copyWithin(destPt,srcIndex, HowManyvaluetoCopy)
numeroUno.copyWithin(3,0,4);
console.log(numeroUno);
```

11,12,13,11,12,13,14,18,19

```
let grades=['A','B','C','D','E','F'];
console.log(...grades.entries());
console.log(...grades.keys());
console.log(...grades.values());
```

0,A 1,B 2,C 3,D 4,E 5,F

0 1 2 3 4 5

A B C D E F

ArrayBuffers

```
let buffer = new ArrayBuffer(1024);
console.log(buffer.byteLength);
buffer[0]=0xff;//hexadecimal
//[] bracket notation can be used with ArrayBuffer
console.log(buffer[0]);|
```

1024

255

Typed Arrays

1. Int8Array()
2. Uint8Array()
3. Uint8ClampedArray()
4. Int32Array()
5. Uint32Array()
6. Int16Array()
7. Uint16Array()
8. Float32Array()
9. Float64Array()



Typed Arrays with ArrayBuffer

```
//typed array and array buffer
let buffer = new ArrayBuffer(1024);
console.log(buffer.byteLength);
buffer[0]=0xff;//hexadecimal
//[] bracket notation can be used with ArrayBuffer
console.log(buffer[0]);
//
let buffertwo = new ArrayBuffer(1024);
let a = new Int8Array(buffertwo);
a[0]=0xff; //hexadecimal value
console.log(a[0]);
let b = new Uint8Array(buffertwo);
b[0] = 0xff;
console.log(b[0]);
let c=new Uint8ClampedArray(buffertwo);
c[0] = -15;
console.log(c[0]);
//
let bufferthree = new ArrayBuffer(1024);
let d= new Uint8Array(bufferthree);
let e=new Uint16Array(bufferthree);
d[0]=1;
console.log(e[0]);
```



Map

- Map is a key/value data structure in ES6. It provides a better data structure to be used for hash-maps.

```
//map example in ES2015
let student1 ={name:"Imad"};
let student2={name:"satnam"};
let students = new Map();
students.set(student1, 'DE12312312');
students.set(student2, 'IN1231232');
console.log(students.get(student1));
console.log(students.get(student2));
// get the size of the map
console.log(students.size);
//delete an entry from the map
students.delete(student2);
console.log(students.size);
//clears out the entire map
students.clear();
console.log(students.size);
let studarr=[
  [student1,'DE12312312'],
  [student2, 'IN1231232']
];
let studentslist = new Map(studarr);
// to check whether a specific key exists or not
console.log(studentslist.has(student2));
let slist = [...studentslist.values()];
console.log(slist);
let smlist = [...studentslist.entries()];
console.log(smlist[0][1]);
```

WeakMap

1. A WeakMap is a subset of Map,
which are not **iterable**
2. **Every key must be an object and
value types are not admitted as
keys.**
3. **WeakMap enables map keys to
be garbage collected when they
are only being referenced as
WeakMap Keys.**

ES2015 Sets

- Sets are yet another collection type in ES6. Sets are very similar to Map.
- Set is also iterable
- They deal with single value or single objects. The purpose of a set is to guarantee a uniqueness.
- Set constructor also accepts an iterable
- Set also has a .size property
- Keys can also be arbitrary values
- Keys must be unique
- NaN equals NaN when it comes to Set



```
1 let PMsOfIndia = new Set();
2 PMsOfIndia.add("Nehru");
3 PMsOfIndia.add("Indira");
4 PMsOfIndia.add("ManMohanSingh");
5 PMsOfIndia.add("PVNRao");
6 console.log(PMsOfIndia);
7 console.log(PMsOfIndia.size);
8 console.log(PMsOfIndia.values());
9 console.log(PMsOfIndia.keys());
10
11 let PresidentsOfIndia = new Set([
12   `Gyani Zail Singh`, `Abdul Kalam`, `Rajendra Prasad`
13 console.log(PresidentsOfIndia.has('Abdul Kalam'));
14 console.log(PresidentsOfIndia.size);
15 console.log(...PresidentsOfIndia.keys());
16 console.log(...PresidentsOfIndia.values());
17 console.log(...PresidentsOfIndia.entries());
18
19 let myval=new Set([11,'11']);
20 console.log(myval.size);
```

[object Set]

4

[object Set Iterator]

[object Set Iterator]

true

3

Gyani Zail Singh Abdul Kalam Rajendra Prasad

Gyani Zail Singh Abdul Kalam Rajendra Prasad

Gyani Zail Singh,Gyani Zail Singh Abdul Kalam,Abdul
Kalam Rajendra Prasad,Rajendra Prasad

2

Subclassing Builtins

- we would like to write extensions to Javascript language builtins.
- The builtin data structures add a huge amount of power to the language, and being able to create new types that leverage that power is amazingly useful

```
//subclassing builtins
class SumArray extends Array{
  sum(){
    let total = 0;
    this.map(v=> total+=v);
    return total;
  }
  let total = SumArray.from([12,13,4212,531,532]);
  console.log(SumArray instanceof Array);
  console.log(SumArray.length);

  let reverseVal = SumArray.reverse();
  console.log(reverseVal instanceof SumArray);
  console.log(reverseVal instanceof Array);
  console.log(SumArray.sum());
```

ES2015 Reflection API

- Object reflection is a language ability to be able to inspect and manipulate object properties at runtime.
- JS has been supporting APIs for object reflection but these APIs were not organized under a namespace and also they threw exception when they fail to complete an operation.
- ES2015 introduces a new object referred as **Reflect** which exposes methods for object reflection.
- These new methods don't throw exception on failure rather they return error, which makes it easy to write code involving object reflection.
- Reflect object cannot be used with **new operator as it is not a constructor**

```
console.log(typeof Reflect); => Object
```

Reflect.construct(target, argumentsList[,newTarget]);

- Not a function Object
- Does not have [[construct]] internal method (can't use new)
- Does not have a [[call]] internal method (can't invoke it as a function)
- Check for browser implementations

Reflect.construct

The [new operator](#) as a function.

```
class Hotel{
    constructor(name, city, rating){
        console.log(` ${name} in ${city} has ${rating}`);
    }
}
function HotelEvaluation(){
    console.log('Evaluate the Quality Standards');
}
let h = Reflect.construct(Hotel, ["Le Meridian", "Bangalore", 8], HotelEvaluation);
Reflect.getPrototypeOf(h);
```

Reflect.apply

Reflect.apply(target, thisArgument, argumentsList)

Calls a target function with arguments as specified by the args parameter

```
class Employee{  
    constructor(){  
        this.empId =7860;  
    }  
    show(){  
        console.log(this.empId);  
    }  
}  
Reflect.apply(Employee.prototype.show, {empId: 7281});  
Reflect.apply(Employee.prototype.show, {empId: 7281}, ['SYCLIQ:']);  
//another example  
Reflect.apply(String.fromCharCode, undefined, [104, 101, 108, 108, 111]);
```

Reflect and Prototypes

Reflect.getPrototypeOf(target)

- It is the same method as Object.getPrototypeOf().
- It returns the prototype of the specified object

```
class LatLong{  
    constructor(){  
        console.log('Stamping out Coordinates');  
    }  
}  
class HotelLocation extends Location{  
}  
console.log(Reflect.getPrototypeOf(HotelLocation));
```

Reflect and Prototypes

`Reflect.setPrototypeOf(target, prototype)`

- It sets the prototype of a specified object to another object or to null.

```
class LatLong{
    constructor(){
        console.log('Stamping out Coordinates');
    }
}
class HotelLocation extends Location{

}
let setup={
    getId(){return 101;}
}
console.log(Reflect.getPrototypeOf(HotelLocation));
let th = new HotelLocation();
Reflect.setPrototypeOf(th,setup);
console.log(th.getId());
```

Reflect and Properties

`Reflect.get(target, propertyKey[, receiver])`

- The static Reflect.get() method works like getting a property from an object.
- It returns the value of the property.
- It allows you to get a property on an object.

```
class Restaurant{  
    constructor(){  
        this.id=9999;  
    }  
}  
let resty = new Restaurant();  
console.log(Reflect.get(resty,"id"));
```

Reflect and Properties

```
Reflect.set(target, propertyKey, value[, receiver])
```

- The static Reflect.set() method works like setting a property on an object.
- It allows you to set a property on an object. It does property assignment and is like the property accessor syntax as a function.

```
class Motel{  
  constructor(){  
    this.id =1111;  
  }  
}  
let m = new Motel();  
Reflect.set(m,'id', 7777);  
console.log(m.id);
```

Reflect and Properties

`Reflect.has(target, propertyKey)`

- The static **Reflect.has()** method works like the **in operator as a function**.
- It allows you to check if a property is in an object.

```
class LandMark{  
    constructor(){  
        this.city = 'Bangalore';  
    }  
}  
class POI extends LandMark{  
    constructor(){  
        super();  
        this.lat = 12.77;  
        this.long = 77.77;  
    }  
}  
let p = new POI();  
console.log(Reflect.has(p, 'city'));  
console.log(Reflect.has(p, 'lat'));  
console.log(Reflect.has(p, 'long'));
```

Reflect and Properties

Reflect.ownKeys(target)

- The **static Reflect.ownKeys() method** returns an array of the target Object's own property keys.
- Its return value is equivalent to `Object.getOwnPropertyNames(target).concat(Object.getOwnPropertySymbols(target))`.

```
class LandMark{  
    constructor(){  
        this.city = 'Bangalore';  
    }  
}  
class POI extends LandMark{  
    constructor(){  
        super();  
        this.lat = 12.77;  
        this.long = 77.77;  
    }  
}  
let p = new POI();  
console.log(Reflect.has(p,'city'));  
console.log(Reflect.has(p, 'lat'));  
console.log(Reflect.has(p, 'long'));  
console.log(Reflect.ownKeys(p));
```

Reflect and Properties

`Reflect.defineProperty(target, propertyKey, attributes)`

- The **static** **Reflect.defineProperty()** method is like `Object.defineProperty()`, but returns a Boolean.
- It allows precise addition to or modification of a property on an object.

```
class Store{  
}  
let st = new Store();  
Reflect.defineProperty(st, 'id', {  
  name:'Walmart',  
  configurable:true,  
  enumerable:true  
});  
console.log(st['name']);
```

Reflect and Properties

`Reflect.deleteProperty(target, propertyKey)`

- The static `Reflect.deleteProperty()` method allows to delete properties.
- It returns a Boolean indicating whether or not the property was successfully deleted.

```
let School= {  
    sid: 1298012,  
    name: 'St.louis'  
};  
console.log(School.sid);  
Reflect.deleteProperty(School, 'sid');  
console.log(School.sid);
```

Reflect and Properties

`Reflect.getOwnPropertyDescriptor(target, propertyKey)`

- The **static `Reflect.getOwnPropertyDescriptor()` method is similar to `Object.getOwnPropertyDescriptor()`.**
- it returns a property descriptor for the given property if it exists on the object, undefined otherwise.

```
let School= {
  sid: 1298012,
  name: 'St.louis'
};
console.log(School.sid);
Reflect.deleteProperty(School, 'sid');
console.log(School.sid);
let d = Reflect.getOwnPropertyDescriptor(School, 'name');
console.log(d);
```

Reflect and Property Extensions

`Reflect.preventExtensions(target)`

- The **static Reflect.preventExtensions()** **method** prevents new properties from ever being added to an object.
- Prevents future extensions of the object.
- Similar to `Object.preventExtensions()`

```
let Person={  
    ssn= 1235412351243  
};  
Person.location="Boston";  
console.log(Person.location);  
Reflect.preventExtensions(Person);  
Person.name="Arijit";  
console.log(Person.name);
```

Reflect and Property Extensions

Reflect.isExtensible(target)

- The static Reflect.isExtensible() method determines if an object is extensible (whether it can have new properties added to it).

```
let Auto={  
    type="three wheeler",  
    make="bajaj"  
};  
console.log(Reflect.isExtensible(Auto));  
Reflect.preventExtenstion(Auto);  
console.log(Reflect.isExtensible(Auto));
```

ES2015 Proxies

- Proxy is used to determine behavior whenever the properties of a **target** object are accessed.
- **A handler object can be used to configure traps for your Proxy.**
- Used for security, profiling,
- New feature in ES2015

ES2015 Proxies : Proxy API

- Proxy is used to determine behavior whenever the properties of a **target** object are accessed.
- **A handler object can be used to configure traps** for your Proxy.
- Used for security, profiling, logging
- New feature in ES2015
- The Proxy object is used to define custom behavior for fundamental operations (e.g. property lookup, assignment, enumeration, function invocation, etc)
- Please check with Kangax for browser compatible implementations.

Proxy Terminology

Wrapper

Handler Object (the Proxy)

TRAPS

Target Object
Or
Function

Get()

Set()

Apply()

Get Proxy

- A trap for getting a property value
- This trap can intercept these operations
 - Property access
 - Inherited property access
 - Reflect.get()

```
function Store(){  
    this.name="WalMart";  
    this.location="Boston";  
    this.MonthlySales = "40000000";  
};  
var ws = new Store();  
var pws = new Proxy(ws, {  
    get:function(target, prop, receiver){  
        return "Successfully Accessed"+ prop;  
    }  
});  
console.log(pws.MonthlySales);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/get



Get Proxy

```
function Store(){
    this.name="WalMart";
    this.location="Boston";
    this.MonthlySales = "40000000";
};

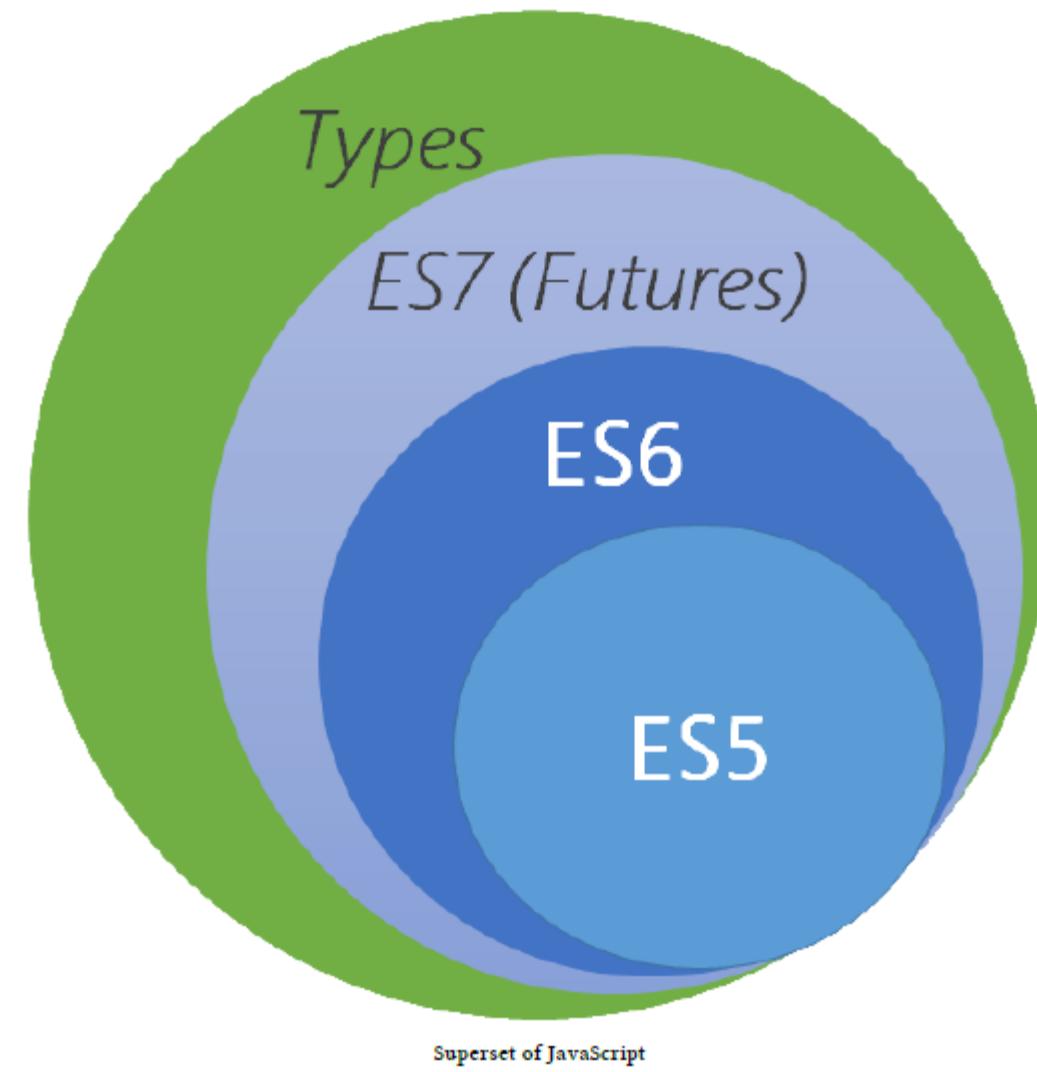
var ws = new Store();
var pws = new Proxy(ws, {
    get:function(target, prop, receiver){
        return "Successfully Accessed"+ prop;
    }
});
var px = new Proxy(ws,{ 
    get:function(target,prop,receiver){
        return Reflect.get(target, prop, receiver);
    }
})
console.log(pws.MonthlySales);
console.log(px.location);
```



TypeScript

Syed Awase Khirni

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project (www.geo-spirit.org). He currently provides consulting services through his startup www.territorialprescience.com and www.sycliq.com



Static Type

Type system

Rigid

Promotes stability and maintainability.

Dynamic Type

Type System
Forgiving

Great for web browser object model.

<https://www.typescriptlang.org/>



Installing TypeScript

Get TypeScript

Node.js

The command-line TypeScript compiler can be installed as a Node.js package.

INSTALL

```
npm install -g typescript
```

COMPILE

```
tsc helloworld.ts
```

Visual Studio



Visual Studio 2015



Visual Studio 2013



Visual Studio Code

And More...



Sublime Text



Emacs



WebStorm



Eclipse

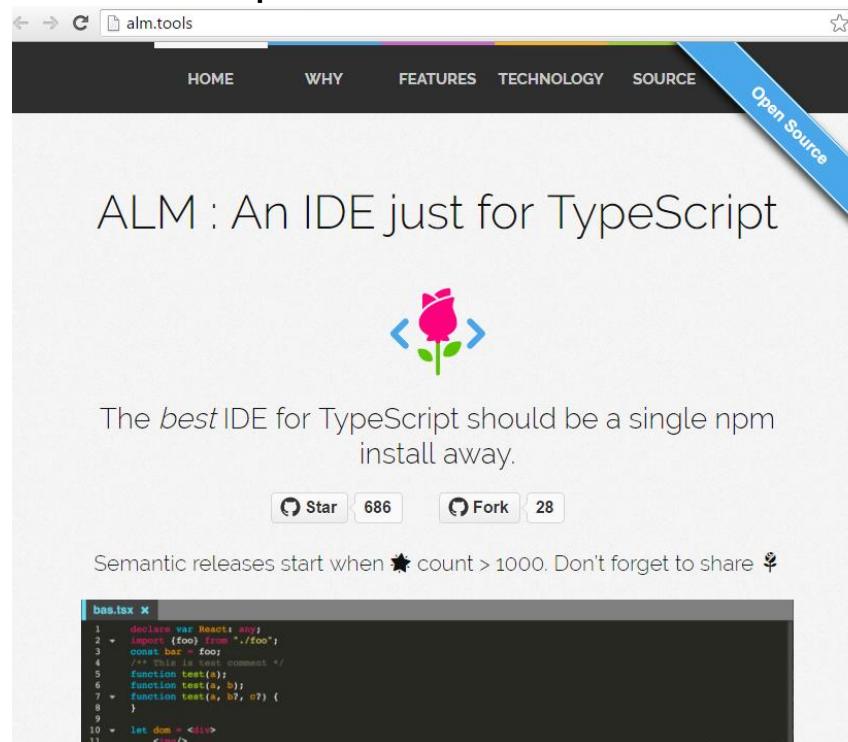


Vim

npm install -g typescript@next

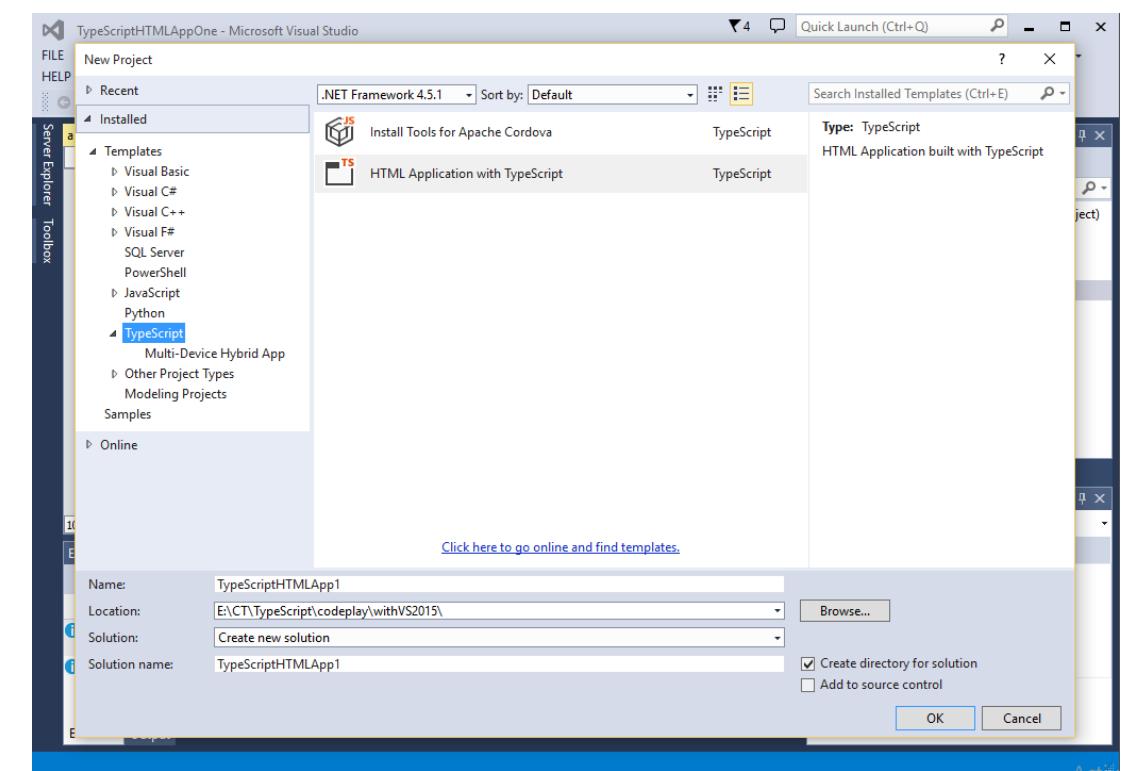
Development Environment

<http://alm.tools/>



```
PS E:\ct\TypeScript\codeplay> npm install -g alm
LoadRequestedDeps -> netw - |#####-----
```

Visual Studio 2015



TypeScriptLang.org/play

TypeScript 1.8 version

TypeScript is a **TRANSPILER**, which takes Your code and changes it, But it's still in the Same language that you started out with.

- Optional static typing and class-based object-oriented Programming, which is a strict superset of JS

The screenshot shows the TypeScript playground interface at <https://www.typescriptlang.org/play/>. The top navigation bar includes links for Documentation, Samples, Download, Connect, and Playground. A banner at the top right says "TypeScript 1.8 is now available. Download our latest version today!" and has a "Fork me on GitHub" button. The main area has tabs for "Using Classes", "TypeScript" (which is selected), and "Share". Below these are two code editors. The left editor contains TypeScript code for a Greeter class:1 class Greeter {
2 greeting: string;
3 constructor(message: string) {
4 this.greeting = message;
5 }
6 greet() {
7 return "Hello, " + this.greeting;
8 }
9 }
10
11 let greeter = new Greeter("world");
12
13 let button = document.createElement('butt
14 button.textContent = "Say Hello";
15 button.onclick = function() {
16 alert(greeter.greet());
17 }
18
19 document.body.appendChild(button);The right editor contains the resulting JavaScript output:1 var Greeter = (function () {
2 function Greeter(message) {
3 this.greeting = message;
4 }
5 Greeter.prototype.greet = function () {
6 return "Hello, " + this.greeting;
7 };
8 return Greeter;
9 }());
10 var greeter = new Greeter("world");
11 var button = document.createElement('butt
12 button.textContent = "Say Hello";
13 button.onclick = function () {
14 alert(greeter.greet());
15 };
16 document.body.appendChild(button);
17 }

Why use TypeScript?

Classic Javascript

Functions are mostly spaghetti code in classic JavaScript unless we use JavaScript design patterns like iife, module pattern or revealing module pattern etc..

JavaScript code encapsulation
difficult to ensure proper types are passed without tests

very few developers use “strict equality”
`(==)`

Complex code for enterprise applications and difficult to maintain

TypeScript

It is a typed superset of JavaScript that compiles to plain JavaScript

More structured and More Organized
Takes care of AMD (Asynchronous Module Definition)

Appeals Enterprise JavaScript Developers with a whole set of features

NOT A totally separate JavaScript

Cross-browser compatible, Works on Any host , Any OS, Open Source, Very Good Tool Support.

Alternatives to TypeScript?

CoffeeScript

The screenshot shows the official CoffeeScript website at coffeescript.org. It features a logo with a coffee cup icon and the text "CoffeeScript". Below the logo are navigation links: TABLE OF CONTENTS, TRY COFFEESCRIPT, and ANNOTATED SOURCE. The main content area contains two sections: "CoffeeScript is a little language that compiles into JavaScript." and "The golden rule of CoffeeScript is: *'It's just JavaScript'*". It also mentions the latest version is 1.10.0.

<http://www.coffeescript.org>

ECMA 6

The screenshot shows a comparison between ECMAScript 6 and ECMAScript 5. The left sidebar lists features: Constants, Scoping, Arrow Functions, Extended Parameter Handling, Template Literals, Extended Literals, and Destructuring Assignment. The right side shows examples for each feature, comparing the ES6 syntax to the traditional ES5 syntax. For example, it shows how the const keyword is used to define constants in ES6.

DART

The screenshot shows the Dart programming language website at <https://www.dartlang.org>. It features a header with the Dart logo and a search bar. The main content area includes a code editor with Dart code for estimating Pi using the Monte Carlo method, and a sidebar with text about Dart being an application programming language and Google's dependency on Dart for large apps. A "Get Started" button is also present.

<http://dartlang.org>

Alternatively you can write classical JavaScript with design patterns

Benefits of TypeScript

Helps in code structuring

Uses class based object oriented
programming

Impose coding guidelines

Offers type checking

Compile time error checking

Intellisense

Optionally statically typed language
**(using any data type, we can assign
any type of value to the variable
by asking the compiler to ignore
the type of a variable)**

Components of TypeScript

Language

Compiler

Language Service

Supports

Modules

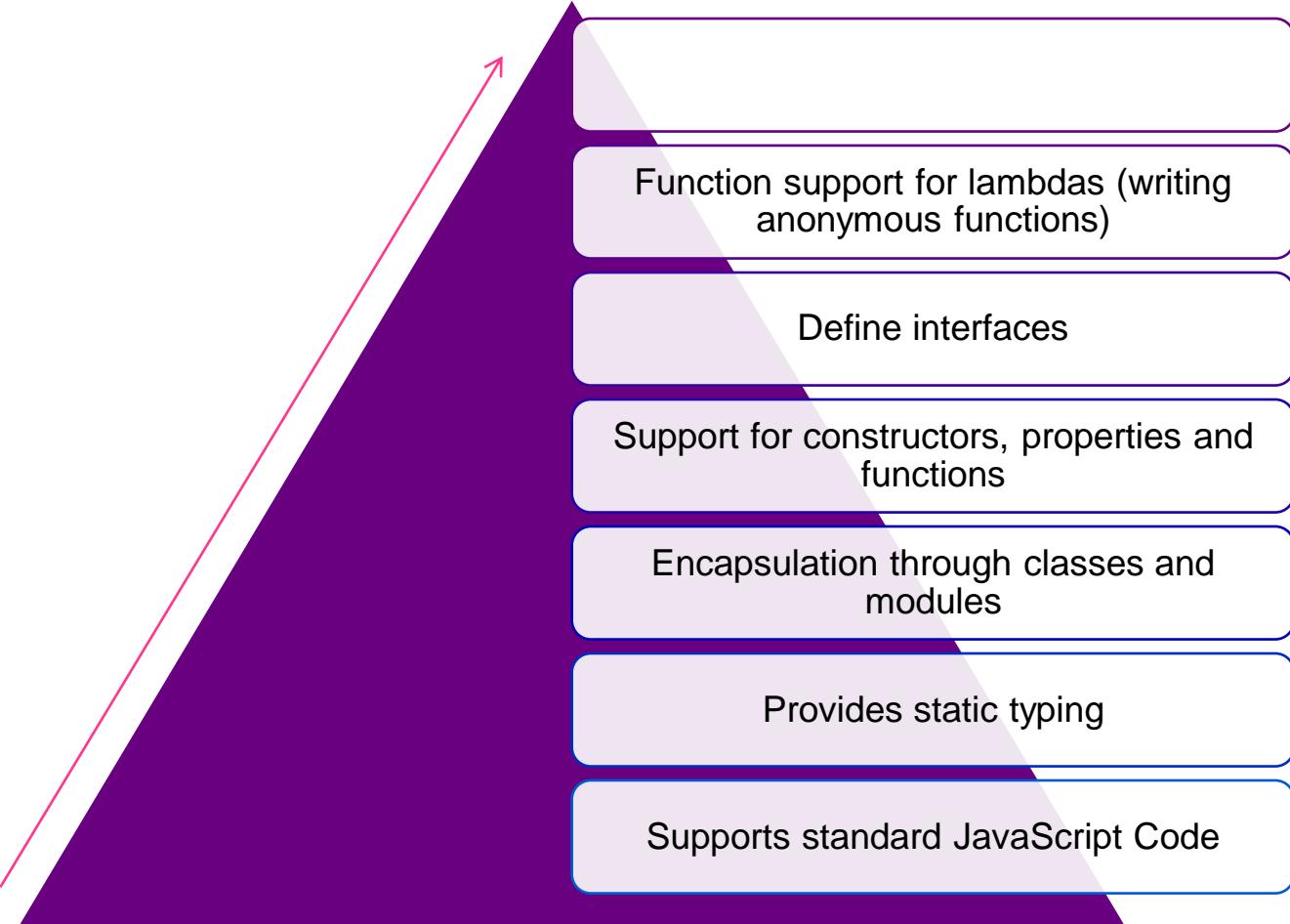
Classes

Interfaces

Data types

Member functions

TypeScript Features



Function support for lambdas (writing anonymous functions)

Define interfaces

Support for constructors, properties and functions

Encapsulation through classes and modules

Provides static typing

Supports standard JavaScript Code

```
1 class Greeter {  
2     greeting: string;  
3     constructor(message: string) {  
4         this.greeting = message;  
5     }  
6     greet() {  
7         return "Hello, " + this.greeting;  
8     }  
9 }  
10  
11 let greeter = new Greeter("world");  
12  
13 let button = document.createElement('button');  
14 button.textContent = "Say Hello";  
15 button.onclick = function() {  
16     alert(greeter.greet());  
17 }  
18  
19 document.body.appendChild(button);
```

```
1 var Greeter = (function () {  
2     function Greeter(message) {  
3         this.greeting = message;  
4     }  
5     Greeter.prototype.greet = function () {  
6         return "Hello, " + this.greeting;  
7     };  
8     return Greeter;  
9 })();  
10 var greeter = new Greeter("world");  
11 var button = document.createElement('button');  
12 button.textContent = "Say Hello";  
13 button.onclick = function () {  
14     alert(greeter.greet());  
15 };  
16 document.body.appendChild(button);  
17
```

```
E:\ct\TypeScript\codeplay\withvs2015> tsc greeter.ts
```

Browser Compatibility

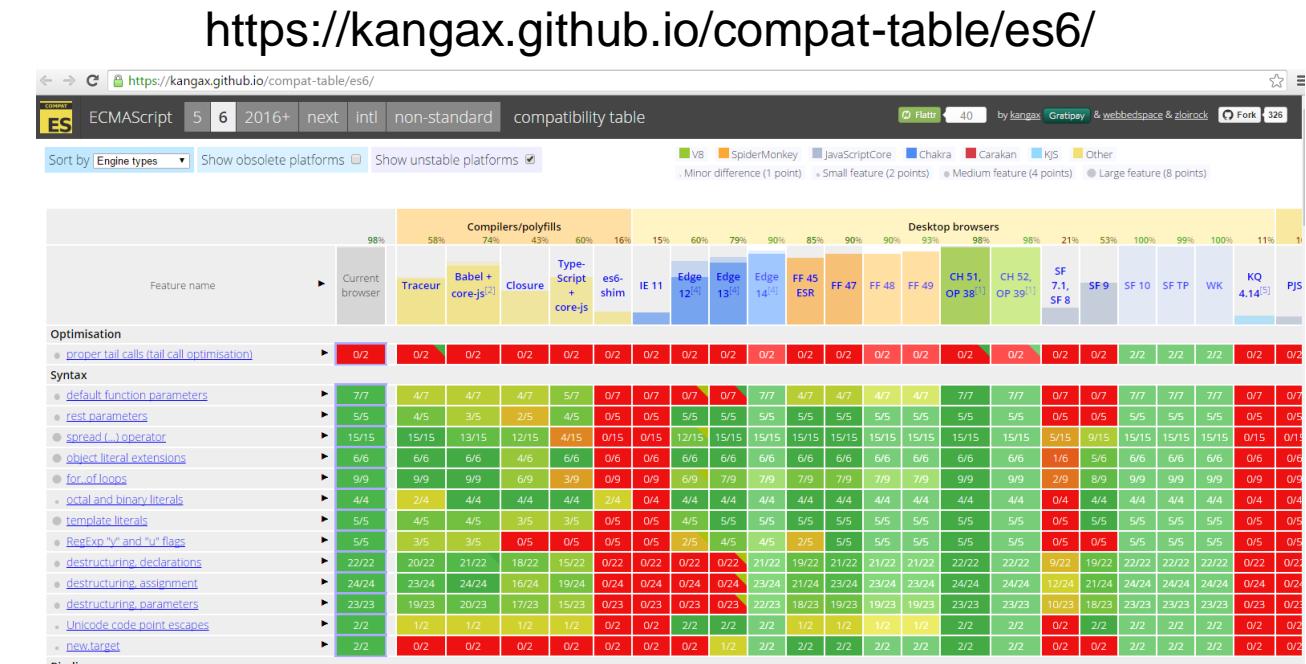
<https://caniuse.com>

May 12, 2015 - Feature suggestions now handle... Compare browsers Index

About News Ads by Google HTML5 CSS3 → HTML5 Browser → HTML5 JavaScript → HTML5 Web Storage →

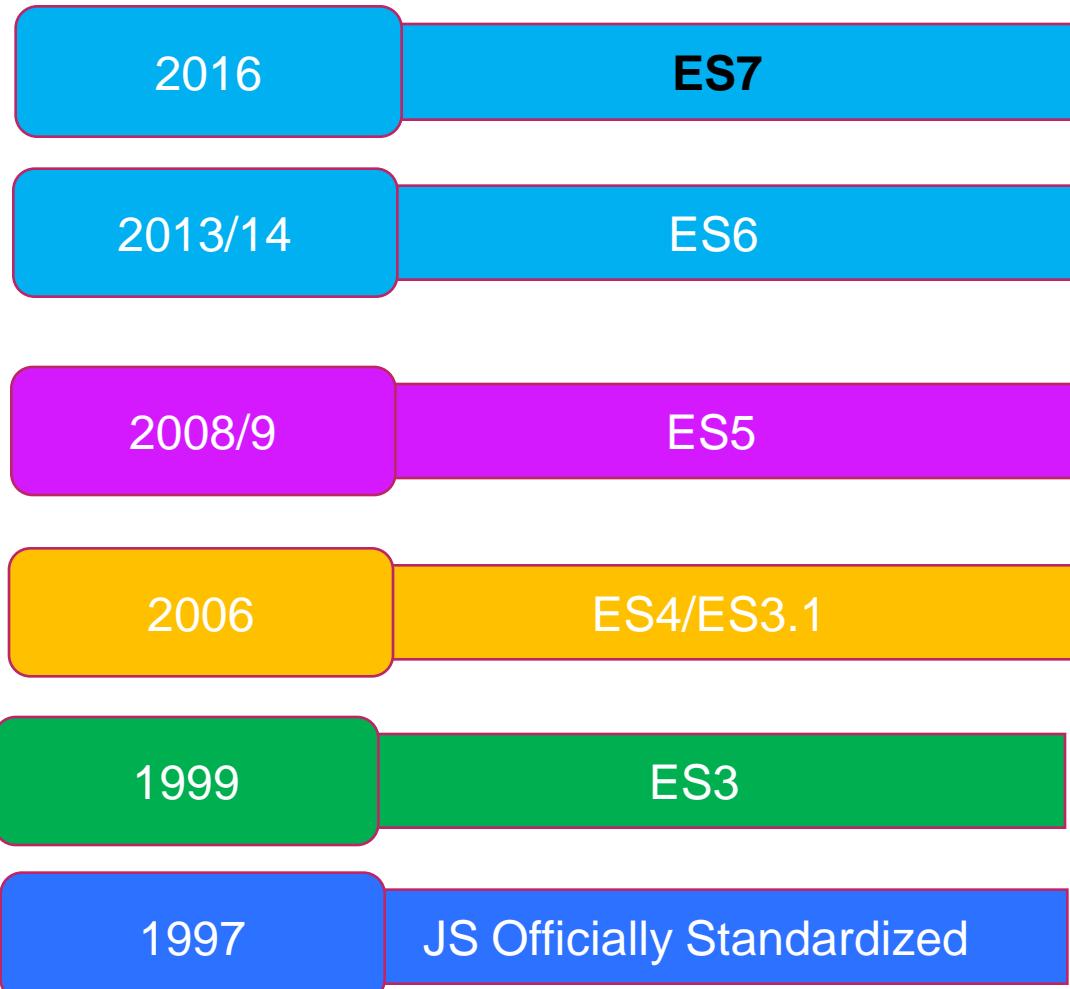
Can I use ? Settings

CSS	HTML5	SVG
▪ ::first-letter CSS pseudo-element selector	▪ accept attribute for file input	▪ Inline SVG in HTML5
▪ ::placeholder CSS pseudo-element	▪ Audio element	▪ SVG (basic support)
▪ ::selection CSS pseudo-element	▪ Audio Tracks	▪ SVG effects for HTML
▪ ::dir() CSS pseudo-class	▪ Autofocus attribute	▪ SVG favicons
▪ ::in-range and ::out-of-range CSS pseudo-classes	▪ Canvas (basic support)	▪ SVG filters
▪ ::matches() CSS pseudo-class	▪ Canvas blend modes	▪ SVG fragment identifiers
▪ @font-face Web fonts	▪ classList (DOMTokenList)	▪ SVG in CSS backgrounds
▪ Blending of HTML/SVG elements	▪ Color input type	▪ SVG in HTML img element
▪ calc() as CSS unit value	▪ contenteditable attribute (basic support)	▪ SVG SMIL animation
▪ ch (character) unit	▪ Custom Elements	▪ SVG fonts
▪ 2.1 selectors	▪ Custom protocol handling	▪ All SVG features
▪ all property	▪ Datalist element	
▪ Animation	▪ dataset & data-* attributes	
▪ Appearance	▪ Date and time input types	
▪ background-attachment	▪ Details & Summary elements	
	▪ Dialog element	
	▪ Base64 encoding and decoding	
	▪ disabled attribute of the fieldset	
	▪ Basic console logging functions	



ECMA TimeLine

European Computer Manufacturers Association



<http://www.ecma-international.org/ecma-262/7.0/index.html>

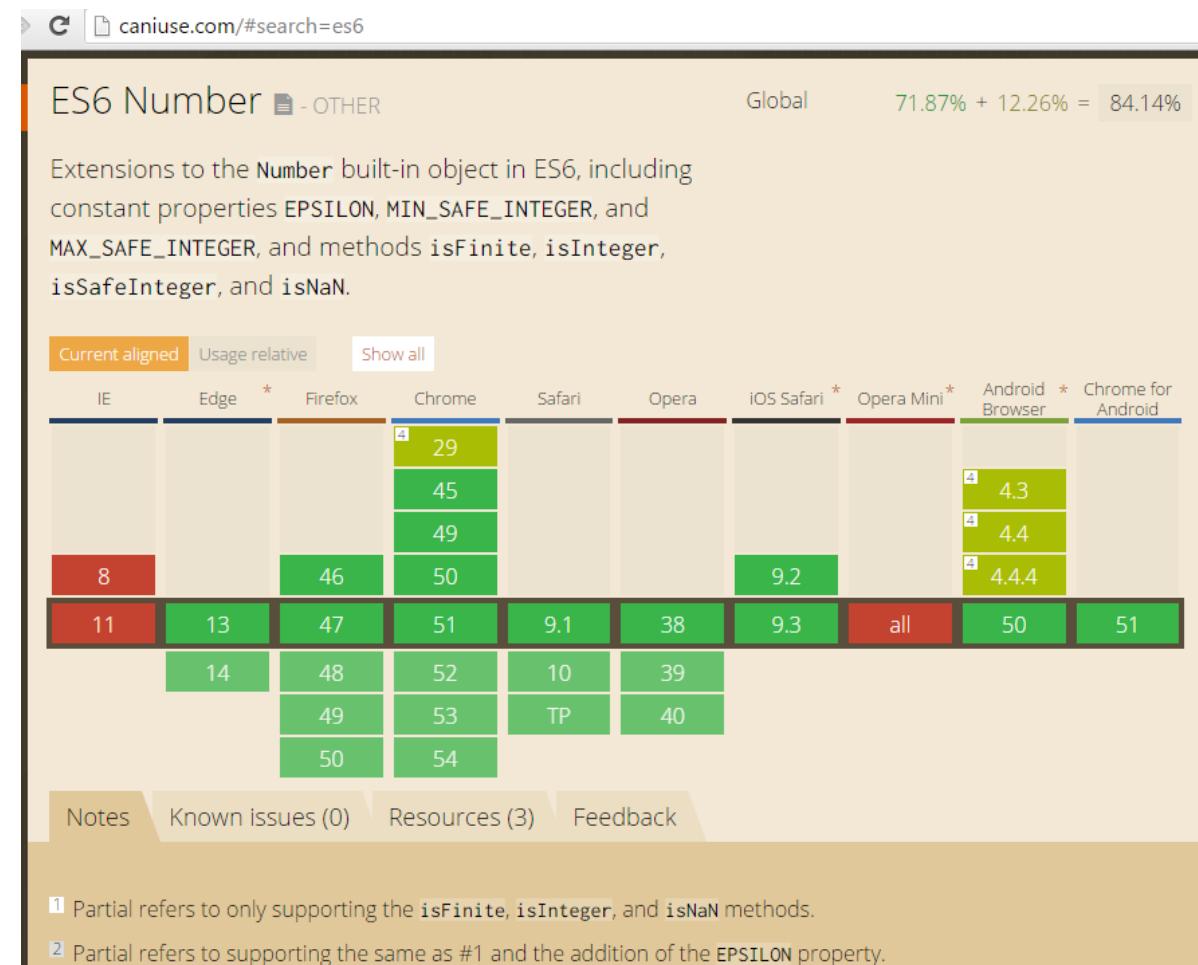
<https://esdiscuss.org/>

TC39 Guides ECMA Specification

ECMAScript -> Language Specification
JavaScript -> implementation of ECMAScript
Java Nashorn -> implementation of ES5.1
Flash/Flex -> implementation of ES3 + custom

ES6 CAN I USE

<http://caniuse.com/#search=es6>



ES6 and TypeScript

Areas of overlap

Classes

Modules

Arrow functions

Rest parameters

Default parameter values

Core TypeScript additions

Types (static, compile-time-only)
supporting static analysis and
tools

compilation to ES3

TypeScript Classes

ES6-> max-min classes

...plus:

Public/private annotations (not
runtime enforced currently)

Field declarations

Statics

Constructor parameter initializers

Static rejection of

...minus (to be added in future)

Class expressions

Class-side inheritance

ES6 and TypeScript

Classes: Static

Statics are used frequently

Imperative update is awkward
when using an otherwise
declarative construct

```
class Point {  
  constructor(x,y) { this.x = x; this.y = y; }  
  static origin = new Point(0,0);  
}
```

Classes: Class Privates

Frequent asks for privacy

TypeScript added compile-time-
only privacy

Not quite same as current private
names syntax proposal

```
class Point {  
  private x: number;  
  private y: number;  
  constructor(x,y) { this.x = x; this.y = y; }  
}
```

ES6 and TypeScript

Classes: Decorators

With classes available, teams want to use them

Biggest blocker so far is when existing class library supported some extra ‘magic’ associated with class/method declarations

ES6 modules

Compiled to JS which uses AMD or commonJS loaders

not yet aligned on import syntax (ES6 not solid here, but may align in future)

much richer notion of internal modules to support “namespace” use cases

ES6 and TypeScript

Modules: Namespaces use case

Two common patterns of large code structure

1. On demand loaded modules (AMD, CommonJS, others)
2. Namespace objects to reduce global pollution

External modules address #1, internal module do not yet handle #2 well

TypeScript allows internal module re-declarations to grow the object

Modules: “modules.exports=“ use case

Not yet addressed in TypeScript

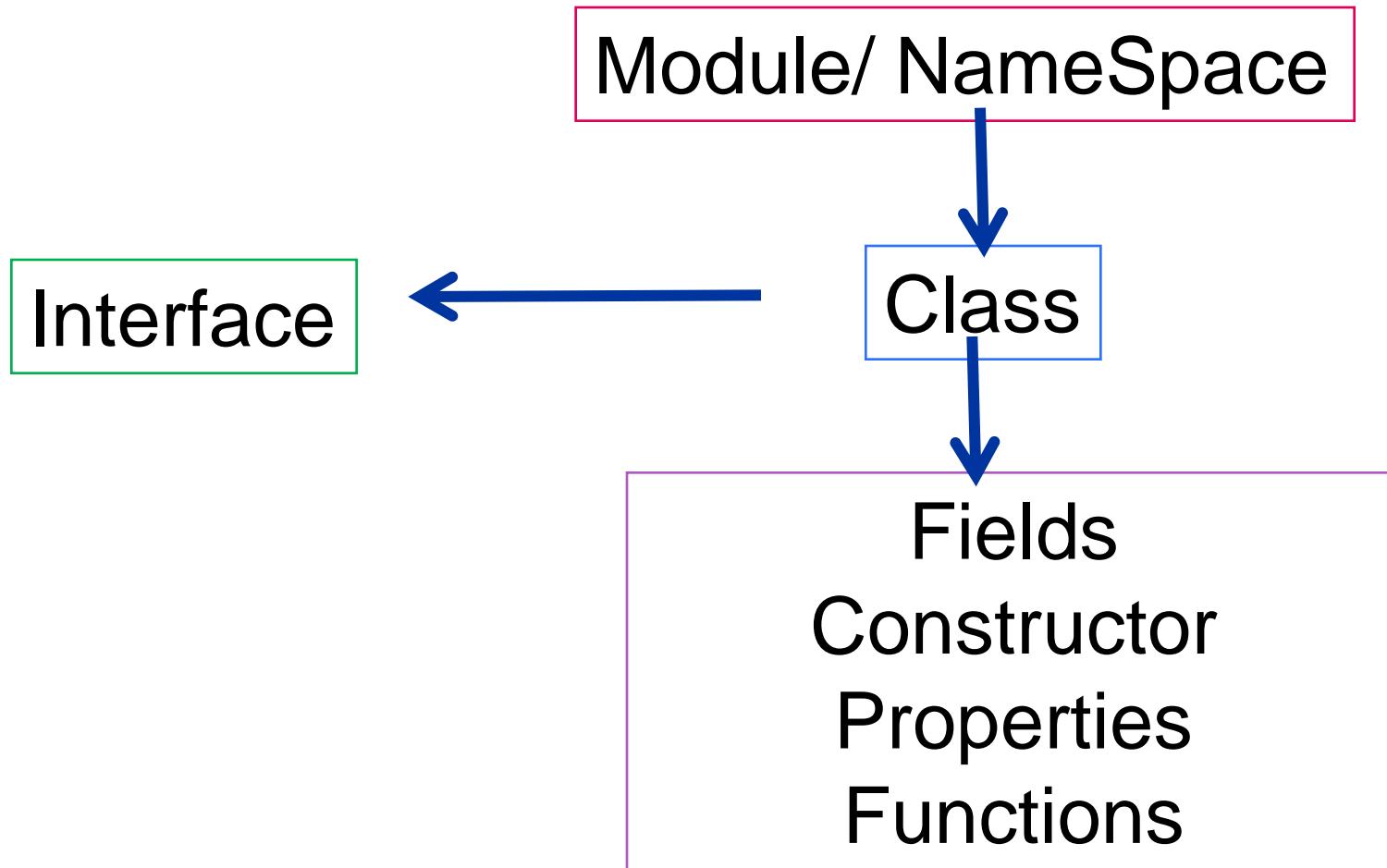
Critical for interop with existing CommonJS/AMD Code

Supportive of “export=“ syntax proposal

TypeScript Keywords and Operators

Keyword	Description
Class	Container for members such as properties and functions
Constructor	Provides initialization functionality in a class
Exports	Export a member from a module
Extends	Extend a class or interface
Implements	Implement an interface
Imports	Imports a module
Interface	Defines code contract that can be implemented by types
Module/namespace	Container for classes and other code
Public/private	Member visibility modifiers
...	Rest parameter syntax
=>	Arrow syntax used with definitions and functions
<typeName>	<> Characters use to cast/convert between types
:	Separator between variable/parameter names and types

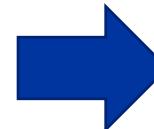
TypeScript Hierarchy



TypeScript Classes

TypeScript

```
class Car {  
    //variables  
    engine: string;  
    //constructor  
    constructor(engine: string) {  
        this.engine = engine;  
    }  
    //function  
    start() {  
        console.log('engine started:' +  
            this.engine);  
    }  
    //function  
    stop() {  
        console.log('engine stopped:' +  
            this.engine);  
    }  
}  
  
window.onload = function () {  
    var car = new Car('V8');  
    car.start();  
    car.stop();  
};
```



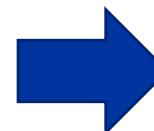
JavaScript

```
var Car = (function () {  
    //constructor  
    function Car(engine) {  
        this.engine = engine;  
    }  
    //function  
    Car.prototype.start = function () {  
        console.log('engine started:' + this.engine);  
    };  
    //function  
    Car.prototype.stop = function () {  
        console.log('engine stopped:' + this.engine);  
    };  
    return Car;  
})();  
  
window.onload = function () {  
    var car = new Car('V8');  
    car.start();  
    car.stop();  
};
```

TypeScript Classes

TypeScript

```
class Employee {  
    //variables  
    fname: string;  
    lname: string;  
    email: string;  
    salary: number;  
    taxrate: number;  
    //constructor  
    constructor(fname: string, lname: string, email:  
        string, salary: number, taxrate: number) {  
        this.fname = fname;  
        this.lname = lname;  
        this.email = email;  
        this.salary = salary;  
        this.taxrate = taxrate;  
    }  
    //function  
    netsalary() {  
        var aftertaxsalary = this.salary - (this.salary *  
            this.taxrate);  
        console.log("After tax salary  
        is :" + aftertaxsalary);  
    }  
}  
window.onload = function () {  
    var sak = new Employee('Awase Khirni', 'Syed',  
        'awasekhirni@gmail.com', 100000, 0.3);  
    sak.netsalary();  
};
```



JavaScript

```
var Employee = (function () {  
    //constructor  
    function Employee(fname, lname, email, salary, taxrate) {  
        this.fname = fname;  
        this.lname = lname;  
        this.email = email;  
        this.salary = salary;  
        this.taxrate = taxrate;  
    }  
    //function  
    Employee.prototype.netsalary = function () {  
        var aftertaxsalary = this.salary - (this.salary * this.taxrate);  
        console.log("After tax salary is :" + aftertaxsalary);  
    };  
    return Employee;  
})();  
  
window.onload = function () {  
    var sak = new Employee('Awase Khirni', 'Syed', 'awasekhirni@gmail.com', 100000, 0.3);  
    sak.netsalary();  
};  
//# sourceMappingURL=Employee.js.map
```

Annotations and inferences

TypeScript

	Description
Var any1;	Type Could be any type (any)
Var num1: numberone;	Type annotation
Var num2: number =2	Type Annotation Setting the value
Var num3 = 3	Type inference (number)
Var num4 = num3 +77	Type inference (number)
Var str1 = num1+ 'this works';	Type inference(string)
Var willnotwork:number = num1+"will give error";	Error

Static and Dynamic typing

TypeScript

Static typing (optional)

Typesafety is a compile-time feature

declare var document; ambient declaration are available in typescript

JavaScript

Dynamic typing

Typesafety happens at run-time debugging

Ambient declarations do not appear anywhere in the JavaScript

Definitely Typed

<http://definitelytyped.org/>

<https://www.nuget.org/packages/jquery.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/knockout.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/angularjs.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/requirejs.TypeScript.DefinitelyTyped/>

<https://www.nuget.org/packages/node.TypeScript.DefinitelyTyped/>

Declaring Variables with var, let and const

var

Variables declared with “**var**” are globally available in the function in which it is declared.

“**Hoisted**” to the top of the function.

Variable name may be declared a second time in the same function.

Let and const

They are scoped to the block they are declared.

Variables declared inside the block will not available/accessible outside of the block.

Not “**Hoisted**” to the top of the block.

Variable name may only be declared once per block.

```
> function ScopeTest(){
  if(true){
    var myVal="use anywhere scope is global";
    let myScope="inside the block";
  }

  console.log(myVal);
  console.log(myScope);

}
```

```
< undefined
```

```
> ScopeTest()
```

```
2017-01-26 16:51:35.152 use anywhere scope is global
```

VM916:7

✖ ► Uncaught ReferenceError: myScope is not defined
at ScopeTest (<anonymous>:8:15)
at <anonymous>:1:1

VM916:8





Basic Types

Boolean

Number

String

Array

Enum (Enumerations)

Any (variable can refer to any type)

Void (absence of a type)

Any Type and Primitives

Represents any JavaScript value
No Static type checking on “any”

```
var data: any;  
var info;
```

Primitive Types

Number => var age: number = 20;
Boolean => var isActive :boolean = true;
String => var firstName:string = “Awase”

Arrays and Indexers

```
var names : string[] = ['sak', 'sas']
```

Null type – is a subtype of all primitives except for void and undefined

```
var num: number = null
```

Undefined => var customer = undefined;
It is a subtype of all types.

Adding Type Annotations

```
> let myName:string="Syed Awase Khirni";
myName=9900911; //error

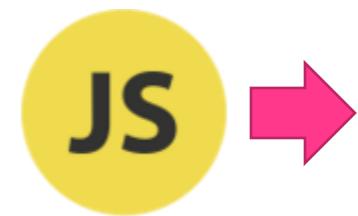
function bitsId():number{
    return 990008;
}

let myCompany:string="SYCLIQ-Systems for Crop Life Cycle Intelligence"
myCompany = bitsId(); //error
```

Enums in TypeScript

```
1 enum ProductType {clothing, electronics, electrical, grocery }
```

```
var ProductType;
(function (ProductType) {
    ProductType[ProductType["clothing"] = 0] = "clothing";
    ProductType[ProductType["electronics"] = 1] = "electronics";
    ProductType[ProductType["electrical"] = 2] = "electrical";
    ProductType[ProductType["grocery"] = 3] = "grocery";
})(ProductType || (ProductType = {}));
```



Arrays

Arrays can be declared in two different ways

Accessed and used much like in JavaScript arrays

Declare as an array of “any” to store any type in the same array

```
// an array of the type string
let spatialTypes:string[]=["point","line", "polygon"];

//using generics Arrays
let spatialGenericType: Array<string>=["point","line", "polygon"];

// with any type |
let anySpatialType: any[] =[242.22, "point", "hexagon"];
```



Tuples

Array where types for the first few elements are specified

Types do not have to be the same

```
//types are defined for the first few elements
let inTuple :[number, boolean]=[23423,true];
// can insert new values
//as long as they are specific to the outlined types
inTuple[2]=false;
inTuple[3]=8237;
```

Functions in TypeScript vs JavaScripts

TypeScript

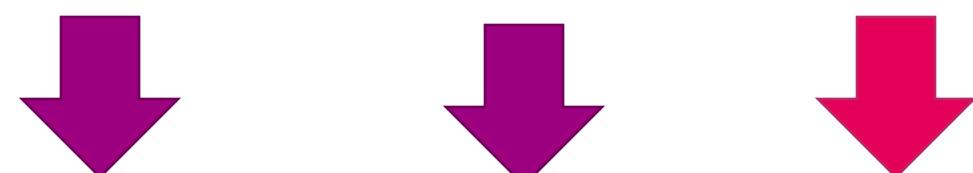
- Declare types for our parameters and return values
- Arrow functions
- Function types
- Allows required and optional parameters
- Explicit support for setting **default parameters**
- **Rest parameters**
- **Allow support for Overloaded functions**

JavaScript

- No types
- Arrow Functions introduced in ES2015, not in earlier version ES5.
- No Function Types
- All parameters are optional (readability and security becomes a challenge)
- Feature support in ES2015, not in ES5.
- Rest parameters in ES2015 not in ES5.
- No overloaded functions

Parameter Types and Return Types

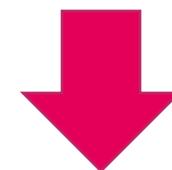
```
function DbConnectionString(url:string, port:number):string{  
    return url+port;  
}
```





Arrow Functions/Lambda Functions

Concise syntax for anonymous functions



```
let listEmployee = AllEmployees.filter(employee=>employee.name==='Syed Awase');
```

```
let listEmployee = AllEmployees.filter(function(employee){  
    return employee.name==='Syed Awase';  
});
```

Functions

Parameter types (required and optional)

Arrow function expressions

Compact form of function expressions

Omit the function keyword

Have scope of “this”

Void

Used as the return type for functions that return no value.

```
//traditional approach
var computearea = function (width:
    number, length: number) {
    return width * length;
}

//using arrow function expression
//omitted the function keyword
//compact return statement with =>
var computevolume = (l: number, w:
    number, h: number) => l * h * w;
```

void

used as the return type of functions
that returns no value

```
//void example
var loginstatus: (msg: string) =>
    void;

loginstatus = function (msg) {
    console.log(msg);
};

loginstatus('Welcome to
    KingsLanding!');
```



Optional and Default Parameters

Optional parameters denoted with "?" after parameter name.

Must appear after all required parameters

Default parameters may be set to a literal value or an expression.

```
//optional parameter
function CreateUser(name:string, age?:number){}
```

```
//default parameter with values
function GetUserById(Id:number="231342"){}  
//or
function GetUserById(Id?:number){}
```



Rest Parameters

Used to collect a group of parameters into a single array.

Denoted with an ellipsis prefix on the last parameter

```
//Rest parameter to collect a group of userIds  
function GetRegUsers(role:string, ...userIds: number[]){ }
```

```
let users = GetRegUsers('general', 232321, 23113, 231112);
```



Function Overloads

TypeScript has function overloading, but ES5 does not have function overloading by type.

Hence, we need to check for all possibilities.

```
//function overloading
//overloading with name type string
function GetUsers(name:string):string[];
//overloading with id type number
function GetUsers(id:number):string[];

function GetUsers(userProperty:any):string[]{
    if(typeof userProperty == 'string'){
        //get users by name
    }
    else if (typeof userProperty == 'number'){
        //get users by id
    }
    return users;
}
```

Object types

Examples are Functions, class, module, interface and literal types

Object types may contain
Properties

Public or private

Required or optional

Call signatures

Construct signatures

Index signatures

```
//object literals
var dimensions = { x: 10, y: 20,
z: 30 };

var room: Object = {
  x: 10, y: 20,
  z: 30
}

//functions are objects in
//typescript
var multiply = function (x:
number) {
  return x * x;
}
```

Interfaces

An interface is a contract that defines a type.

The compiler enforces the contract via type checking.

Interface defines a contract how a contract can and cannot be used.

Interfaces are just used by the compiler for **just type checking**.

Interface is set to specify a specific shape of an object.

JavaScript does not support custom types.

Interfaces do not compile to anything in JavaScript.

Collection of property and method definitions WITHOUT ANY IMPLEMENTATION DETAILS.

DUCK TYPING

Interfaces

Interface is created by using “interface” keyword.

List properties with their types

```
interface User{  
  id: number;  
  name:string;  
  gender:string,  
  age?:number,  
  ethnicity?:string;  
  isRegistered: (status:string)=> void;  
}
```

```
interface Chaplin{  
  walk:() => void;  
  talk:() => void;  
  dance:() => void;  
}  
  
let probablyCharlieChaplin={  
  walk:()=>=> console.log("Walk like Charlie Chaplin"),  
  talk:()=>=> console.log('talk like Charlie Chaplin'),  
  dance:()=>=> console.log('dance like charlie Chaplin')  
}  
  
function fightlikeChaplin(Charlie:Chaplin){ }  
  
fightlikeChaplin(probablyCharlieChaplin);
```

Extending Interfaces

```
interface LibraryResource{  
    catalogNumber:number;  
}  
interface Book{  
    title:string;  
}  
interface Encyclopedia extends LibraryResource,Book{  
    volume:number;  
}  


---

let refBook:Encyclopedia={  
    catalogNumber:32434,  
    title:'Geographic Information Retrieval',  
    volume:212  
}
```

Class

Template/Specification for creating objects

Provides state storage and behaviour

Encapsulates reusable functionality

Provide actual implementation details

Classes in Typescript

Define Types

Support Properties and Methods

Have a constructor
can apply access modifier
(public, private and protected)

Support inheritance

Support Abstract Classes

Classes in TypeScript

Classes act as containers for different members that encapsulate code.

TypeScript class members include

Fields

Constructors

Properties

Functions

Class members are public by default

```
class Book {  
    //fields  
    author: string;  
    isbn: string;  
    publisher: string;  
    bookname: string;  
    //constructor  
    constructor(author: string, bookname:  
        string, publisher: string, isbn: string)  
    {  
        this.author = author;  
        this.bookname = bookname;  
        this.publisher = publisher;  
        this.isbn = isbn;  
    }  
}
```

initialization

Defining Properties in Classes

Properties act as filters and can have a get or set blocks

```
class Vehicle{  
    //fields  
    private _engine: string;  
    //constructor  
    constructor(engine: string) {  
        this.engine = engine;  
    }  
    //property  
    get engine(): string {  
        return this._engine;  
    }  
    //property  
    set engine(value: string) {  
        if (value == undefined) throw 'Supply an  
            Engine!';  
        this._engine = value;  
    }  
}
```

Traditionally in JavaScript, we use `Object.defineProperty()` to set and get the properties of the object (ECMA5 Feature)

```
function Book(name) {  
    Object.defineProperty(this, "name", {  
        get: function() {  
            return "Book: " + name;  
        },  
        set: function(newName) {  
            name = newName;  
        },  
        configurable: false  
    });  
}
```

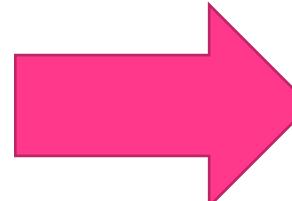
Defining Properties in Classes

```
class Vehicle{  
    //fields  
    private _engine: string;  
    //constructor  
    constructor(engine: string) {  
        this.engine = engine;  
    }  
    //property  
    get engine(): string {  
        return this._engine;  
    }  
    //property  
    set engine(value: string) {  
        if (value == undefined) throw 'Supply an  
            Engine!:';  
        this._engine = value;  
    }  
}
```

```
var Vehicle = (function () {  
    //constructor  
    function Vehicle(engine) {  
        this.engine = engine;  
    }  
    Object.defineProperty(Vehicle.prototype, "engine", {  
        //property  
        get: function () {  
            return this._engine;  
        },  
        //property  
        set: function (value) {  
            if (value == undefined)  
                throw 'Supply an Engine!:';  
            this._engine = value;  
        },  
        enumerable: true,  
        configurable: true  
    });  
    return Vehicle;  
})();  
//# sourceMappingURL=ExampleSix.js.map
```

Using Complex Types

```
1 class Engine {  
2     constructor(public horsePower: number,  
3                  public engineType: string) { }  
4 }  
5 //new class  
6 class Automotive {  
7     //field declarations  
8     private _engine: Engine;  
9     //constructor  
10    constructor(engine: Engine) {  
11        this._engine = engine;  
12    }  
13 }  
14  
15 var toyotaengine = new Engine(300, 'v8');  
16 var innovacrysta = new Automotive(toyotaengine);
```



```
//using complex types  
var Engine = (function () {  
    function Engine(horsePower, engineType) {  
        this.horsePower = horsePower;  
        this.engineType = engineType;  
    }  
    return Engine;  
})();  
;  
  
//new class  
var Automotive = (function () {  
    //constructor  
    function Automotive(engine) {  
        this._engine = engine;  
    }  
    return Automotive;  
})();  
  
//instantiation  
var toyotaengine = new Engine(300, 'v8');  
var innovacrysta = new Automotive(toyotaengine);  
## sourceMappingURL=Examplethree.js.map
```

Access Modifiers

Public : Default property in Typescript

Private : Class membership inside the class only. It cannot be accessed from outside of its containing class.

Protected : acts much like the private modifier with the exception that members declared protected can also be accessed by instances of deriving classes.

Readonly – properties can be made readonly by using readonly keyword. Readonly properties must be initialized at their declaration or in the constructor.

Type Assertions

TypeScript allows you to override its inferred and analyzed view of types any way you want to. Type assertion are purely YOU telling the compiler that you know about the types better than it does, and it should not second guess YOU.

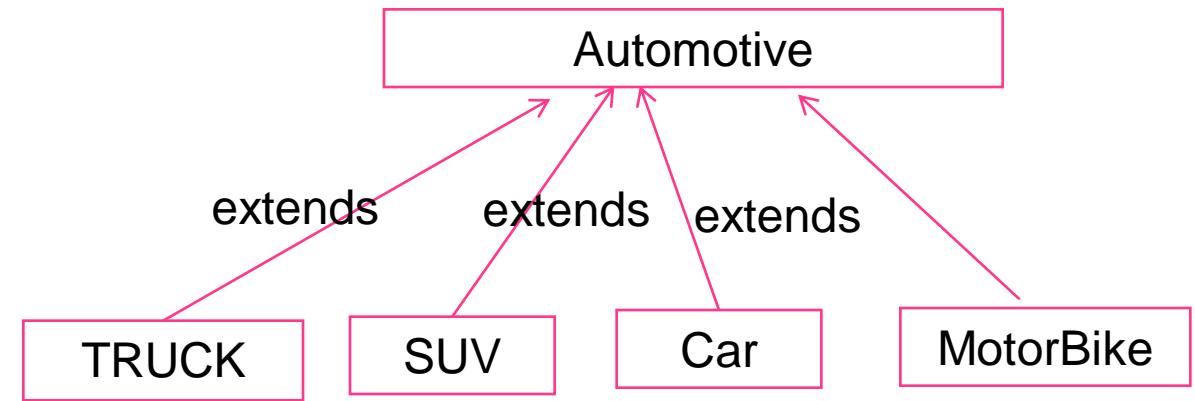
A common use case for type assertion is when porting over code from javascript to typescript.

reason why it is “not called type casting” is that casting generally implies some sort of runtime support. assets are purely a compile time construct and a way to inform the compiler how you want your code to be analyzed.

Extending Types

Types can be extended using the TypeScript “extends” keyword

```
class ChildClass extends ParentClass{  
    constructor() {  
        super();  
    }  
}
```





Abstract Classes

created with the "abstract" keyword

Base classes that may not be instantiated

May contain implementation details

Abstract methods are not implemented.

Extending Types

```
class Auto {  
    engine: Engine;  
    constructor(engine: Engine) {  
        this.engine = engine;  
    }  
}  
  
//truck derives from auto base class  
class Truck extends Auto {  
    fourByFour: boolean;  
    constructor(engine: Engine, fourByFour: boolean) {  
        //calling the base class constructor  
        super(engine);  
        this.fourByFour = fourByFour;  
    }  
}
```

```
1 class Animal {  
2     name: string;  
3     constructor(theName: string) {  
4         this.name = theName;  
5     }  
6     move(distanceInMeters: number= 0) {  
7         console.log(`${this.name} moved ${distanceInMeters}m.`);  
8     }  
9 }  
0  
1 class Horse extends Animal {  
2     constructor(name: string) {  
3         super(name);  
4     }  
5     move(distanceInMeters= 50) {  
6         console.log("Galloping..");  
7         super.move(distanceInMeters);  
8     }  
9 }
```

Using Interface

Interfaces are code contracts. TypeScript supports interfaces.

An interface is defined as a syntactical contract that all classes inheriting the interface should follow. The interface defines the “what” part of the syntactical contract and the deriving classes define the “how” part of the syntactical contract.

Interfaces define properties, methods and events, which are members of the interface.

Interface contain only declaration of the members.

```
1 interface IEngine {  
2     start(callback: (startStatus: boolean, engineType: string) =>  
3         void): void;  
4     stop(callback: (startStatus: boolean, engineType: string) =>  
5         void): void;  
6 }  
7 class AutoEngine implements IEngine {  
8     constructor(public horsePower: number, public engineType:  
9         string) { }  
10    start(callback: (startStatus: boolean, engineType: string) =>  
11        void) {  
12        window.setTimeout(() => {  
13            callback(true, this.engineType);  
14        }, 1000);  
15    }  
16    stop(callback: (startStatus: boolean, engineType: string) =>  
17        void) {  
18        window.setTimeout(() => {  
19            callback(true, this.engineType);  
20        }, 1000);  
21    }  
22 }
```

Module

Modules help us keep separation based on similar functionality

Modules help us in testing individual functionality

Modules help us make the code reusable and maintainable.

We can extend modules within or across files

Each module has a specific role or functionality

Modules are open, we can import, export modules or features

```
1 module dataservice {  
2   export class DataSync {  
3   }  
4 }  
5 //alternatively without module declaration, no exports, no imports  
6 class Library {  
7 }  
8  
9 var universitylib = new Library()
```



TYPESCRIPT TESTING

Unit Testing

unit tests add clarity to your code
They favour decoupling
Encourages simplification
Supports Extensibility



HTML should be a projection of app state, not a source of truth

HTML5 Web Components

SYED AWASE

Existing Challenges with Web Development

- Undescriptive Markup in HTML
- Style conflicts, require highly specific CSS selectors
 - use of !important to force styles
 - No guarantee another style won't conflict
- No Native templates – We use `<script type="text/html">`, store HTML in hidden DOM elements and manually extract
- No bundling , to improve user experience, performance and less roundabout to the server.
- No component standard.

Web-Components

are encapsulated, reusable and composable widgets for the web platform.

Templates

- Reusable markup

Custom Elements

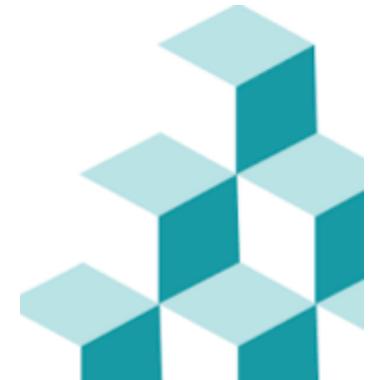
- Define our own custom elements

Shadow DOM

- encapsulated markup and styling
– DOM and Style boundaries

Imports

- Bundle HTML, JS and CSS as a single file





React

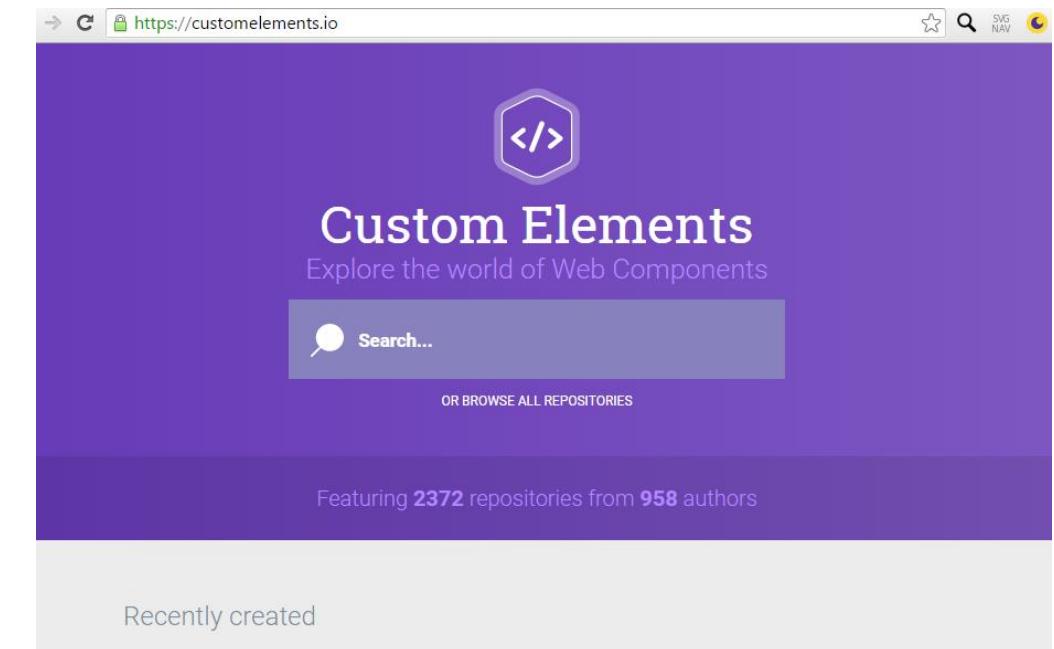
<http://webcomponents.org/>



The screenshot shows the homepage of webcomponents.org. At the top, there's a navigation bar with links for HOME, POLYFILLS, ARTICLES, PRESENTATIONS, PODCASTS, and RESOURCES. A search bar is also present. The main content area features a large orange icon of two interlocking wrenches inside a hexagon. Below the icon, the text "WebComponents.org" is displayed in a large, bold, white font. Underneath, a subtitle reads "a place to discuss and evolve web component best-practices". On the left, a "POLYFILLS" section explains that the webcomponent.js polyfills enable Web Components in (evergreen) browsers. In the center, a "DISCOVER" section links to "CUSTOMELEMENTS.IO", described as "A gallery to display". On the right, an "ARTICLES" section includes a link to "CUSTOM ELEMENTS V1: BUILDING REUSABLE WEB COMPONENTS".

Code Focused Training

<https://customelements.io/>



The screenshot shows the homepage of customelements.io. The header features a purple background with the text "Custom Elements" and "Explore the world of Web Components". Below the header is a search bar with the placeholder "Search...". A link "OR BROWSE ALL REPOSITORIES" is visible. The main content area has a light gray background and displays the text "Featuring 2372 repositories from 958 authors". At the bottom, there's a section titled "Recently created".

<https://html.spec.whatwg.org/multipage/scripting.html#the-template-element>

<https://www.w3.org/TR/html-imports/>

<https://www.w3.org/TR/shadow-dom/>

<https://www.w3.org/TR/custom-elements/>

Browser Readiness

Are We Componentized Yet?

Source: <http://jonrimmer.github.io/are-we-componentized-yet/>

Tracking the progress of **Web Components** through standardisation, polyfillification¹, and implementation.

	Specged	Implementation				
		Polyfill	Chrome / Opera	Firefox	Safari	Edge
Templates			Stable	Stable	8	13
HTML Imports			Stable	On Hold	No Active Development	Vote
Custom Elements			Stable	Flag	Prototype	Vote
Shadow DOM			Stable	Flag	10	Vote

Cells are clickable! Yellow means in-progress; green means mostly finished.

3 Major Libraries

For working with and extending web components

- X-Tags by Mozilla

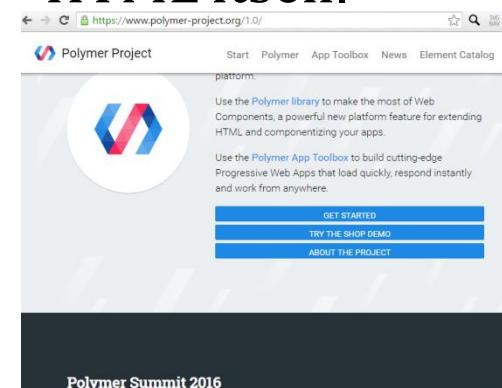
- allows you to easily create elements to encapsulate common behavior or use existing custom elements to quickly get the behavior you are looking for.



<https://x-tag.github.io/>

- Polymer by Google

- encapsulated and interoperable custom elements that extend HTML itself.



<https://www.polymer-project.org/1.0/>

- Bosonic

- Built on top of web components polyfill library.



<http://bosonic.github.io/>



<https://whatwg.org/>



Welcome to the WHATWG community

Maintaining and evolving HTML since 2004

[FAQ](#)

Get answers to your questions
about HTML and the WHATWG

[HTML](#)

Read, use, or implement
the HTML Living Standard

<https://component.kitchen/>



Component.KITCHEN

[HOME](#) [BLOG](#) [TUTORIAL](#) [ABOUT](#)

Web components accelerate app development

Web components let you extend HTML with new capabilities so you can write better web apps faster. Component KITCHEN are experts on this transformational technology. We design and develop web components, we work with a range of industry players to improve the underlying specs and libraries, and we consult with companies who want to use web components to make their products better.

[Learn](#)

[About](#)

JUNE 13, 2016

Can Service Workers service background applications?

I had a thought experiment on how I might port an application I once developed for native Android to a web app, even if it were to run solely on Android devices. The application behaves like a mantle clock: every fifteen or thirty minutes, it wakes up and plays a custom chime. My son used to call it "Big Ben in Your Pocket."

MAY 2, 2016

Why Web Components?

- 1 • Descriptive markup
- 2 • Bootstrapped elements
- 3 • Easily replaced and upgraded
- 4 • Fewer integration mistakes
- 5 • Clearer interface/api

HTML Templates

- contains inert chunks of markup intended to be used later
- Existing strategies have Cross Site Scripting risk from using `.innerHTML`
- Nothing inside runs or renders
- Ng-template (angular)
- Templates in KnockOutJS

Custom template name

```
<script type="text/product-list-stamp">
```

```
<script>
```

Declaring HTML templates

- Contain inert chunks of markup intended to be used later
- Templates are parsed not rendered
- Working directly with DOM
- Hidden from document, Cannot traverse into its DOM

```
document.querySelector('#product-list-template.sycliq-style_QHY') == null  
  
<template id="product-list-stamp">  
  <!--dynamically populated at runtime-->  
  <img src="" alt="">  
  <div class="sycliq-style_QHY"></div> ←  
  <input type="text" name="input_subscribe" value="subscribe"/>  
</template>
```

Using template content

- Two ways
- template.content(a document fragment)
- Template.innerHTML

```
<template id="product-list-stamp">
  <!--dynamically populated at runtime-->
  <img src="" alt="">
  <div class="sycliq-style_QHY"></div>
  <input type="text" name="input_subscribe" value="subscribe"/>
</template>
<script>
  var temp = document.querySelector('#product-list-stamp');
  temp.content.querySelector('img').src ="ebayProduct-list-1.jpg";
  document.body.appendChild(temp.content.cloneNode(true));
</script>
```

Using Template Steps

```
//1.Get a reference to the template  
var template = document.querySelector('templateName');  
//2.Use document.importNode to clone the template's Content  
var clone = document.importNode(template.content,true);  
//3.change the target element within the template as desired  
clone.querySelector('.verb').textContent="Syed Awase Khirni";  
//4 Append element to page  
document.body.appendChild(clone);
```



HTML DOM

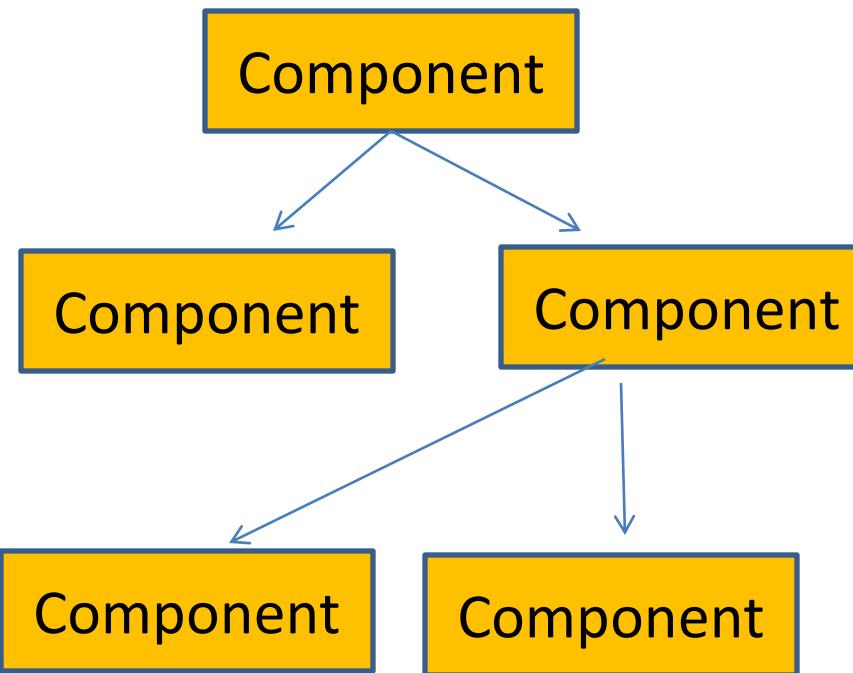
- Inherently very slow
- Needs to be **Rendered(visually represented on to the screen, layout calculations, painting)**
- **HTML**

VIRTUAL DOM

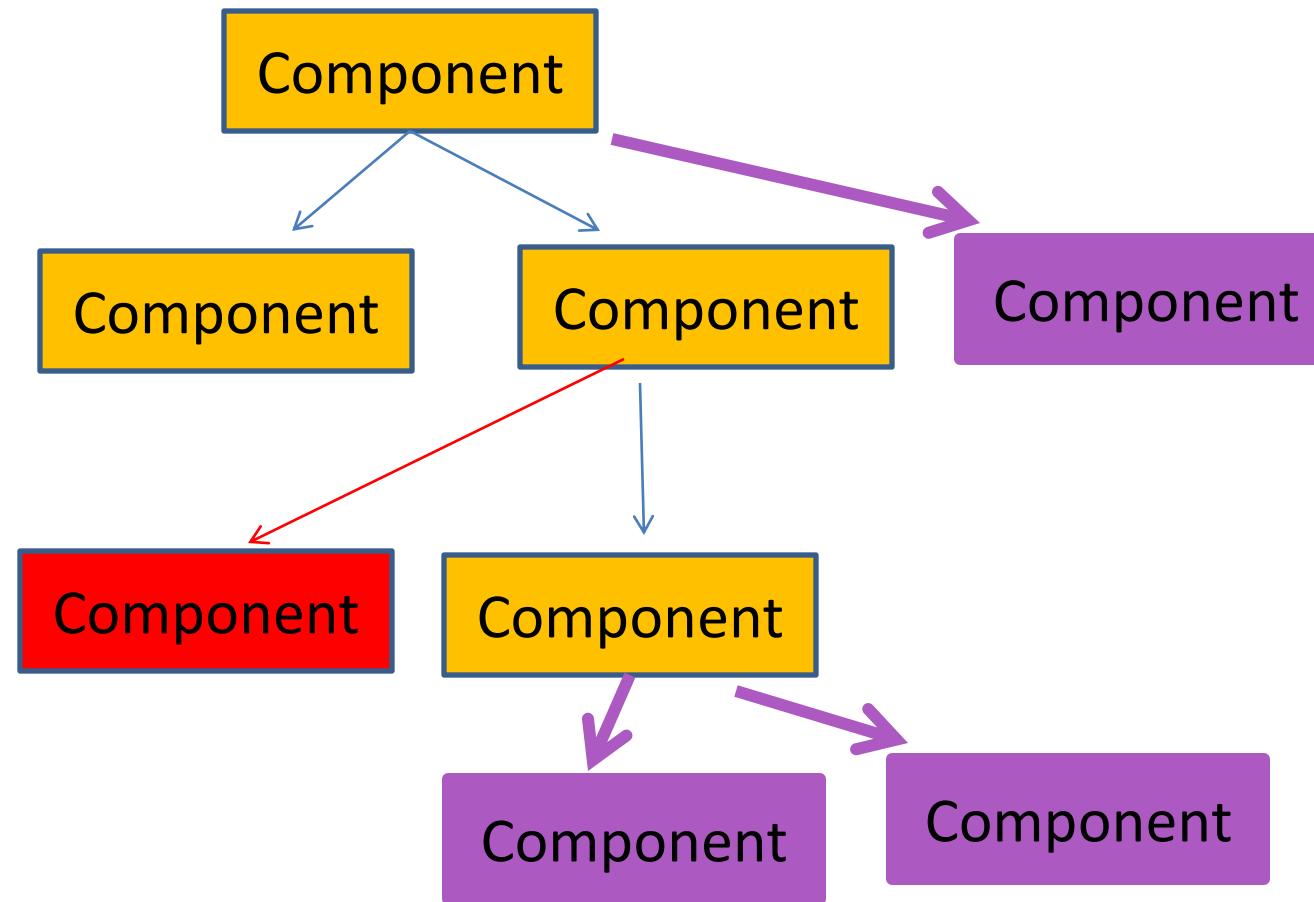
- Very fast
- In-Memory purely done through javascript objects.
- **JSX**

DIFF Phase in Virtual DOM Construction

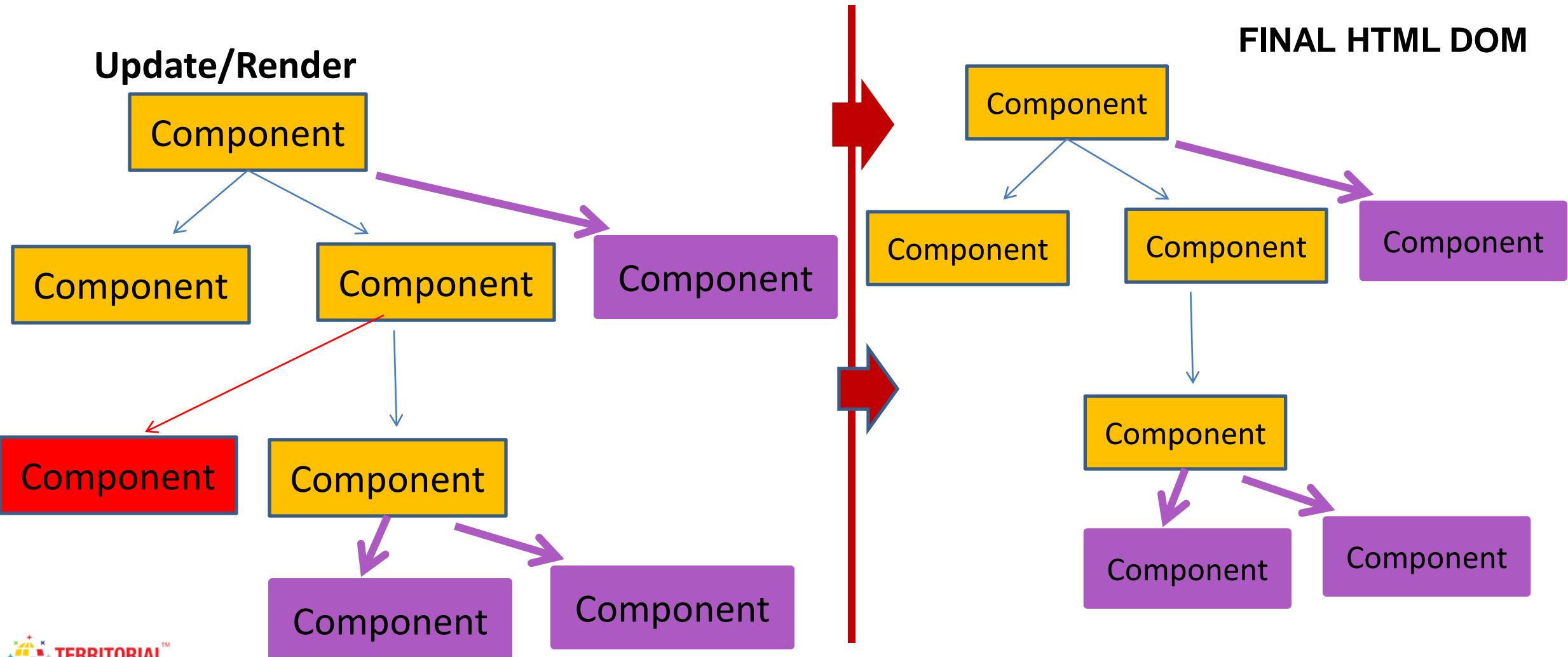
Render



Subsequent Render/Update



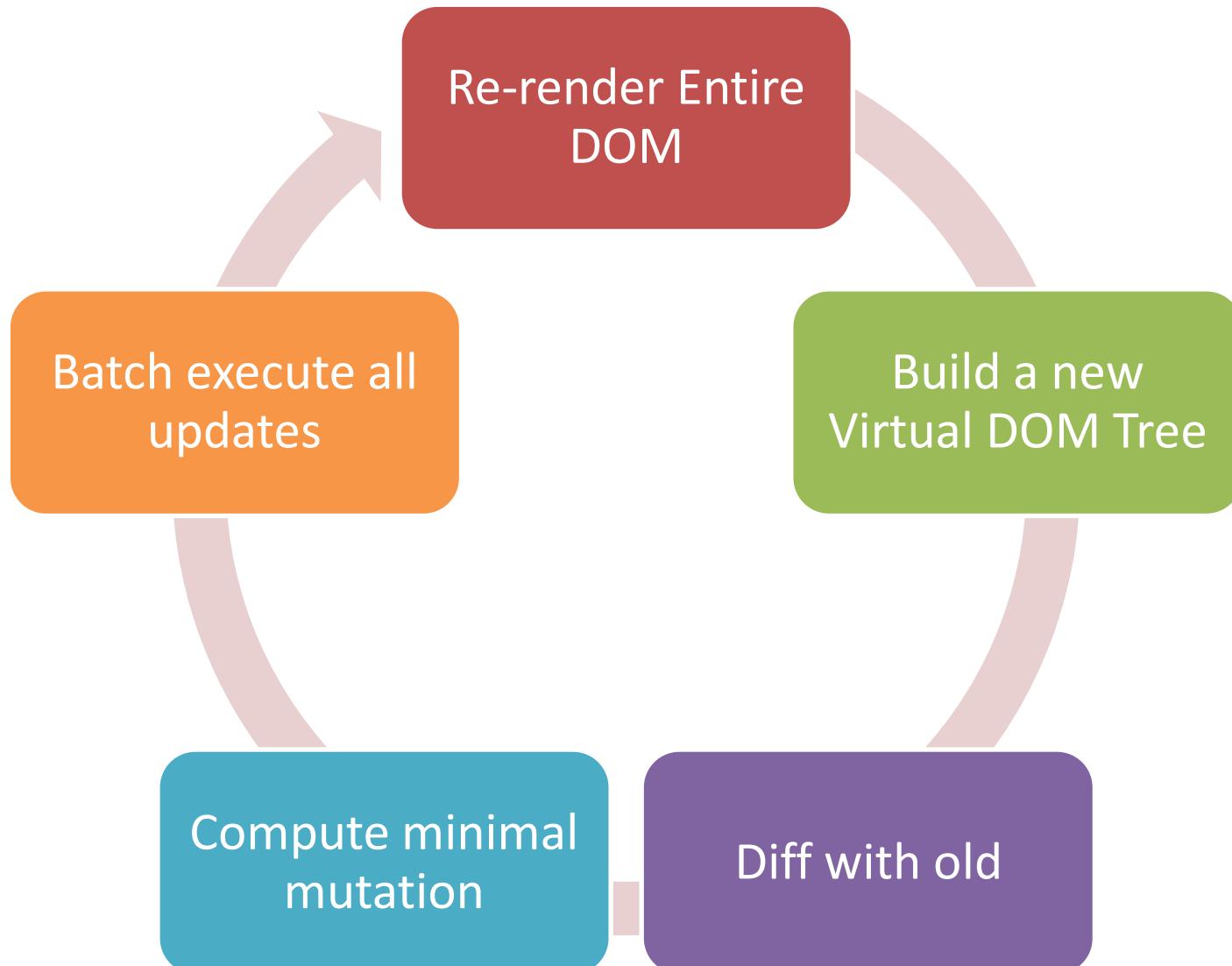
Patch phase in Virtual DOM Construction



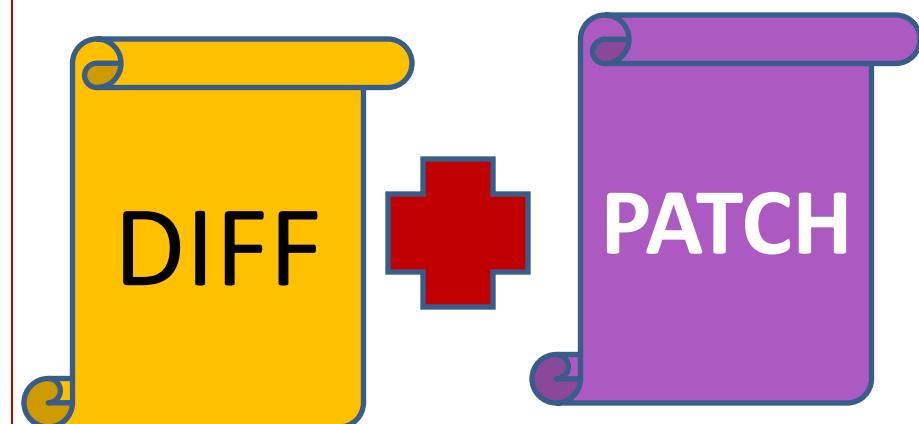
Virtual DOM

- In memory representation of the DOM and events system
- Even HTML5 events in IE8
- Interacting with DOM -> slow process
- Interacting with JS In-Memory Objects – Fast process
 - 60 fps fast
 - Faster parsing
 - Faster validation
- Since no real browser, can easily render on server or client
- Easier to Test
- Autobinding and Event Delegation

Virtual DOM



RENDER ALGORITHM



React looks at both nodes and attributes for minimal set of changes

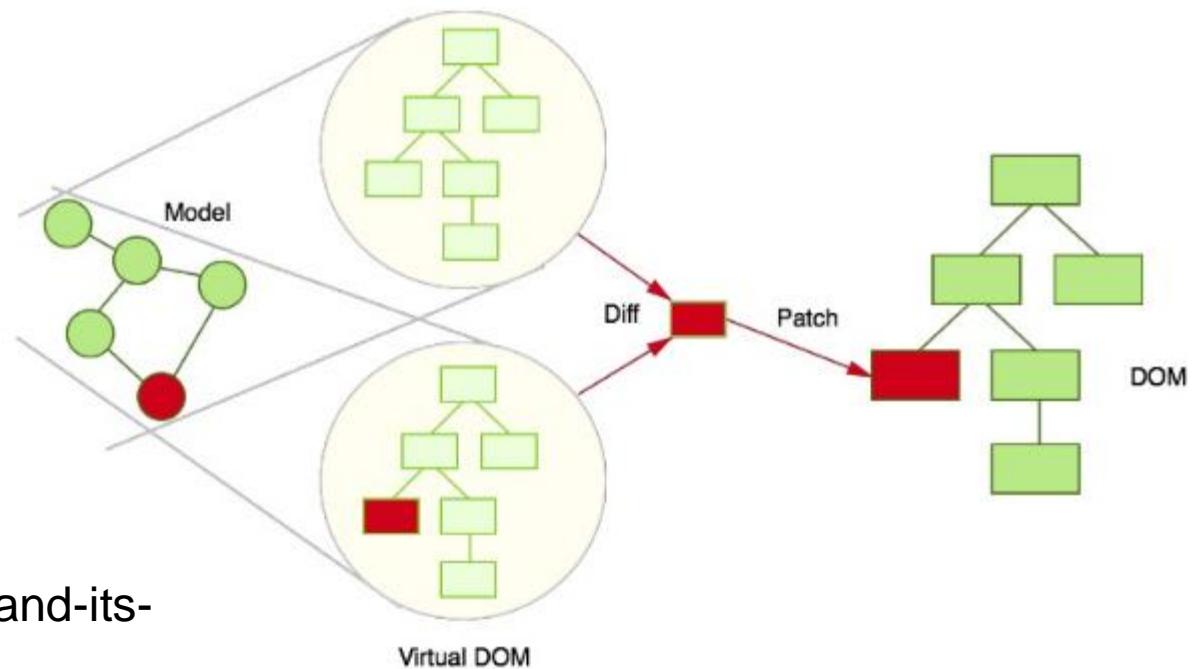
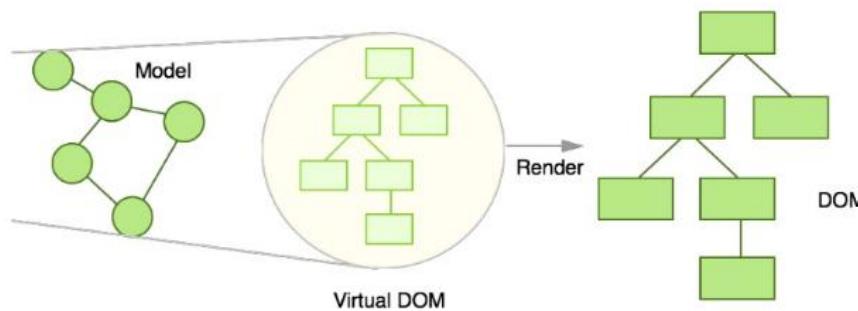
Virtual DOM in React

- React maintains a DOM of its own
 - Helps in identifying which parts have changed by comparing the new version with the one previously stored.
 - renders only the changed parts
 - Determines how to update the browser's DOM more efficiently
 - Fast render of the application



Virtual DOM

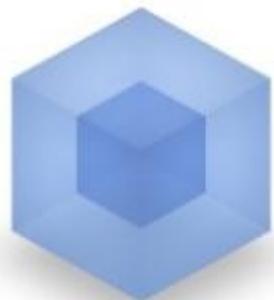
- Batched DOM read/write operations
- Efficient update of sub-tree only



Source: <http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html>

View Rendering in React

- 1 • JSX Code is written describing a basic application
- 2 • JSX is converted into JavaScript by the Compiler
- 3 • React turns converted JSX into a Virtual DOM
- 4 • Virtual DOM is applied to real DOM
- 5 • Underlying Data Model is updated
- 6 • React updates Virtual DOM and then quickly updates the real DOM



Webpack

<http://webpack.github.io>

SYED AWASE

WEB PACK MODULE BUNDLER

Build Systems



<http://broccolajs.com/>



<http://duojs.org/>

SystemJS

<https://github.com/systemjs>



<http://browserify.org/>



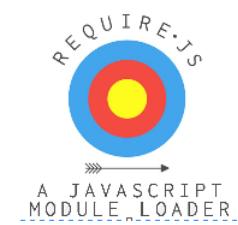
<https://github.com/gobblejs/gobble>



<http://brunch.io/>



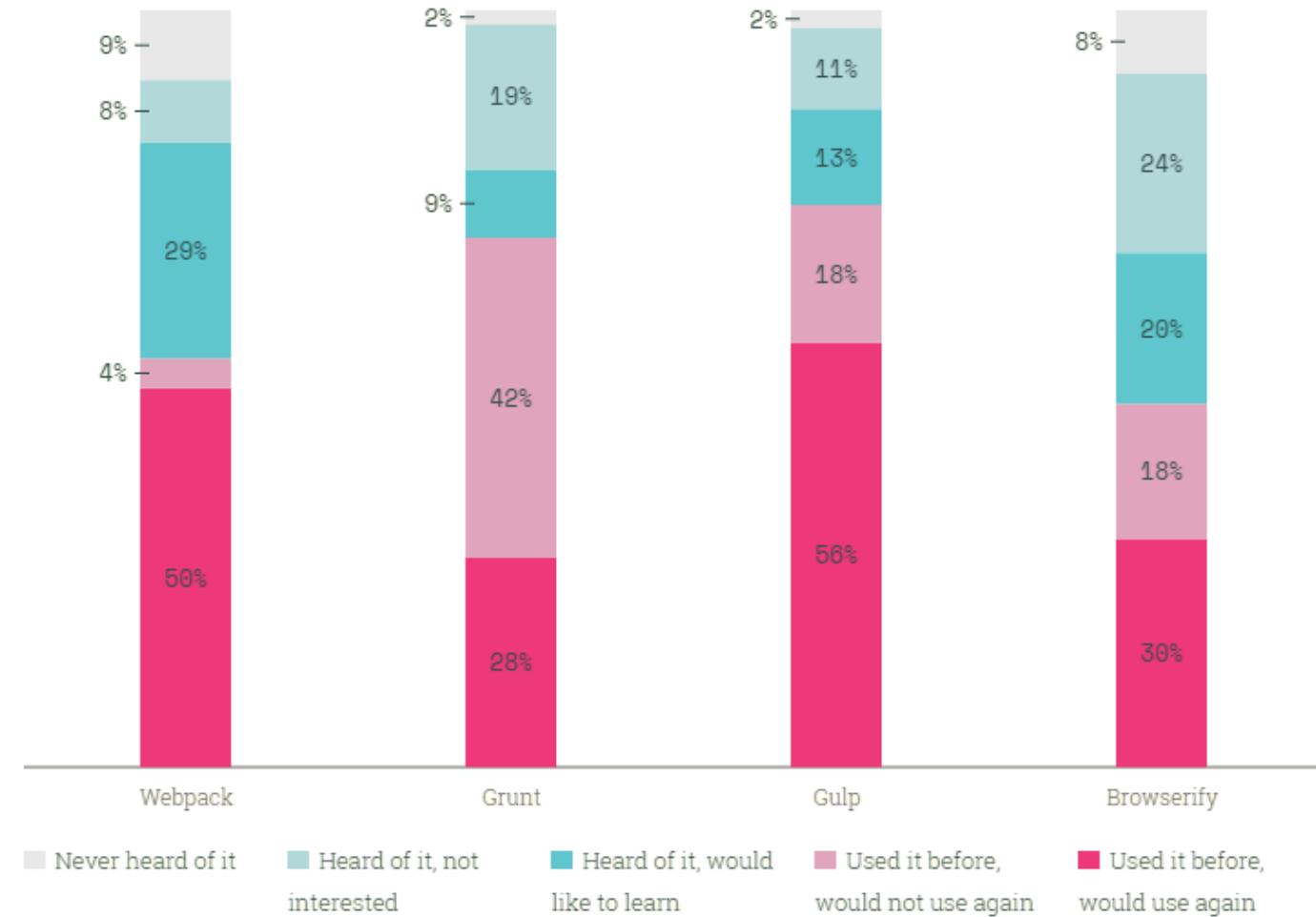
<http://rollupjs.org/>



<https://github.com/flyjs/fly>

Build Tool Statistics

Source: <http://stateofjs.com/2016/buildtools/>



Why webpack?

- JavaScript is unorganized, not modularized, no automatic dependency injection, loosely structured, loosely typed language.
- Simplifies web development by solving a fundamental problem – bundling, transforming the assets to be easily consumed by the browser.
- Allows you to treat your project as a **dependency graph**.
- Webpack does all the preprocessing and gives the bundles needed through configuration.

Server Side Templating

- Back end server creates an HTML document and sends it to the user
- For every user request to the server there is a new HTML document render to the client.

Single Page Application

- Server sends a bare-bones HTML doc to the user. JavaScript runs on the user machine to assemble a full web page.
- For Every user request to the server, only partial pages are added to the master page initially rendered to the client.
- Better Server Performance, UI experience, less round-trips to the server.

Task Runners vs Bundlers

Task Runners

- Task runners such as Grunt and Gulp allow users to perform operations in a cross-platform manner, loading all assets at once.
- They are powerful means of putting assets together to serve specific functionality by loading dependent libraries.

Bundlers

- Bundlers are needed when it is required to splice various assets together and produce bundles on the fly.
- Browserify, Brunch or Webpack are bundlers.
- JSPM pushes package management directly to the browser. It relies on System.JS a dynamic module loader.

Modules in JavaScript

- Components



- API

- CRUD API

Module Systems in JavaScript

- Module Systems are rule sets that define how the various javascript files based on functionality are linked together and loaded on demand.

CommonJS

- A module system implemented by Node.js
- Require
- Module.exports

AMD

- Asynchronous module definition and loading
- Require
- define

ES2015

- ES6/ES2015 module system loading is achieved with export and import
- Import
- Export

CommonJS Pattern

```
//step-1. Get reference to import/includes
var importlib = require('./path/to/file');

//step-2. Module declarations
var myModuleDef = {
    //code goes here
}

//step-3. Export it for other to import/use
module.exports = myModuleDef;
```

Webpack

- It is driven by configuration
- Webpack.config.js
- Configuration defines the inputs and the outputs of your project.
- It describes the types of transformations you perform.
These transformations are defined using loaders and plugins each of which serves as a purpose of its own.

```
e:\CT\ReactJS>mkdir webpack-codeplay
```

```
e:\CT\ReactJS>cd webpack-codeplay
```

```
e:\CT\ReactJS\webpack-codeplay>npm init -y  
Wrote to e:\CT\ReactJS\webpack-codeplay\package.json:
```

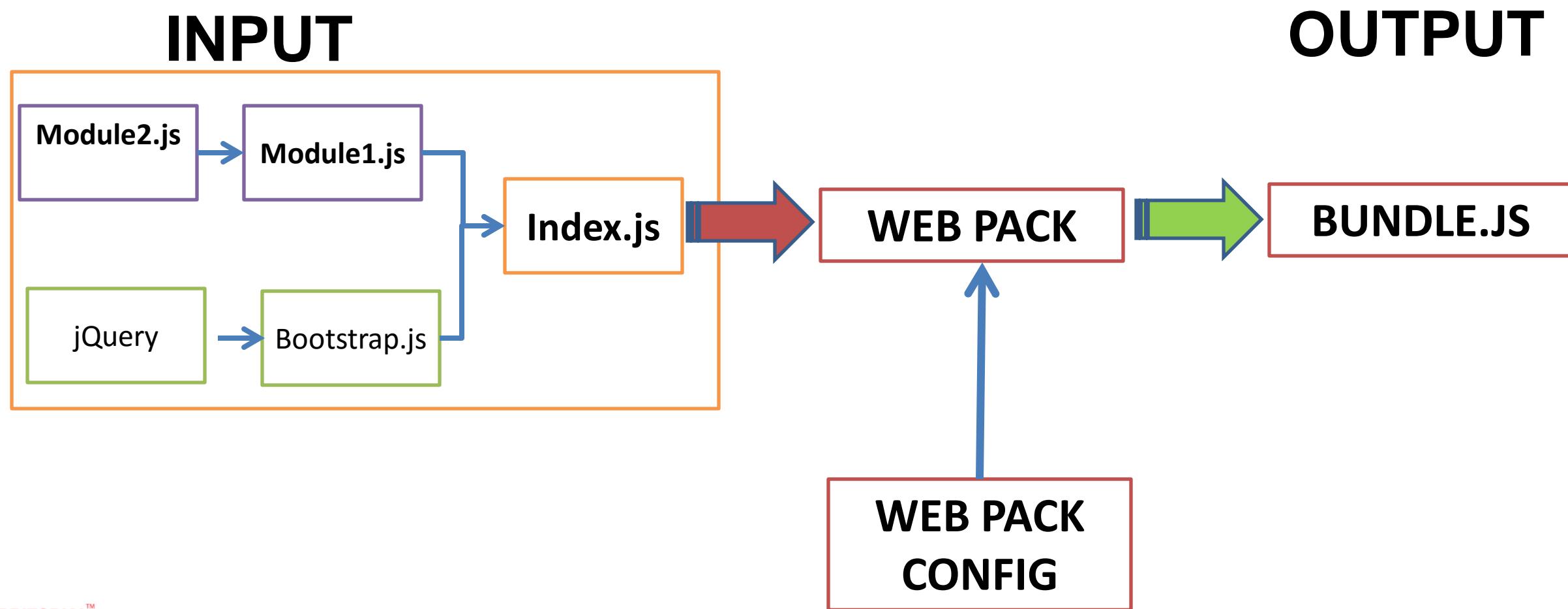
```
{  
  "name": "webpack-codeplay",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC"  
}
```

```
npm i webpack --save-dev # or just -D if you want to save typing
```

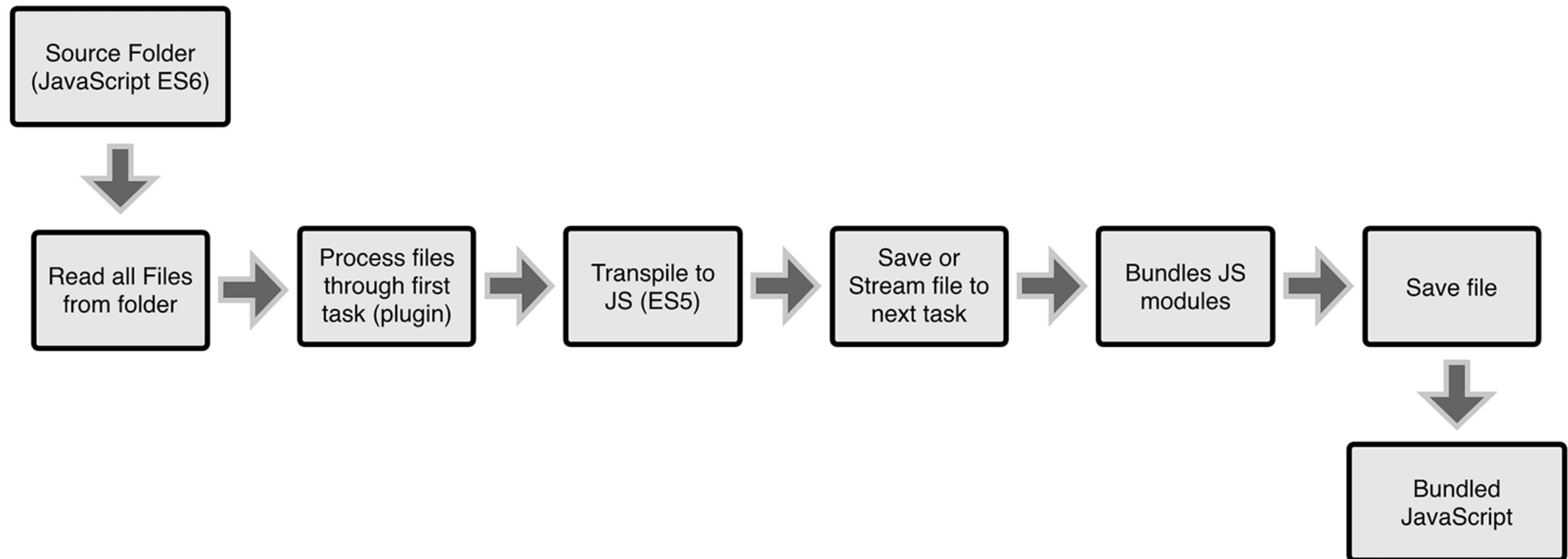
```
e:\CT\ReactJS\webpack-codeplay\node_modules\.bin>webpack  
webpack 1.13.2
```

```
Usage: https://webpack.github.io/docs/cli.html
```

Web pack Structure



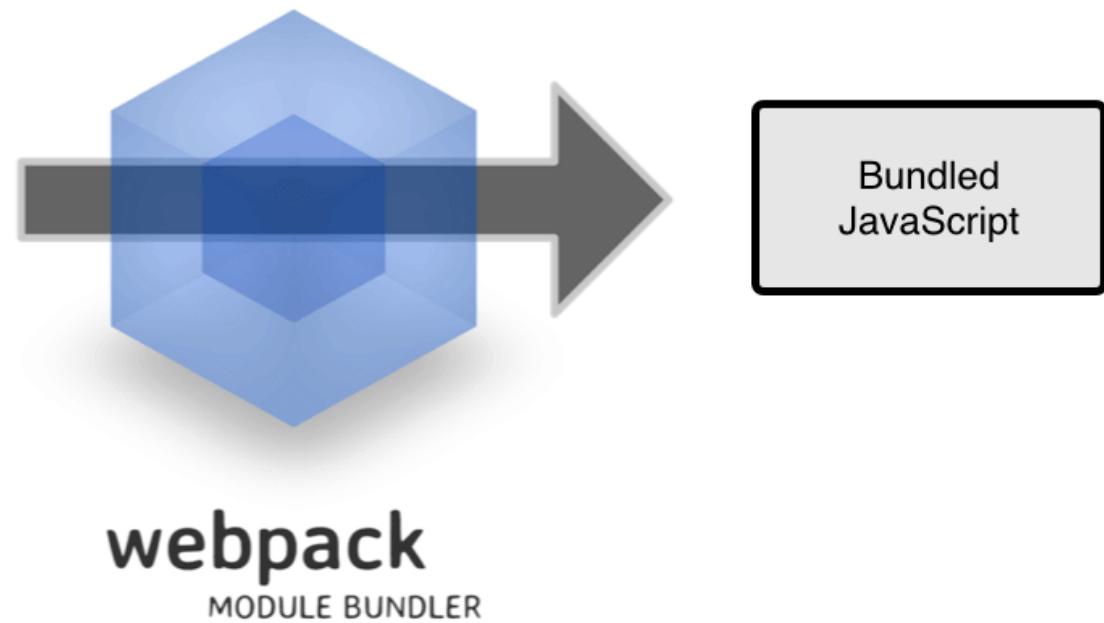
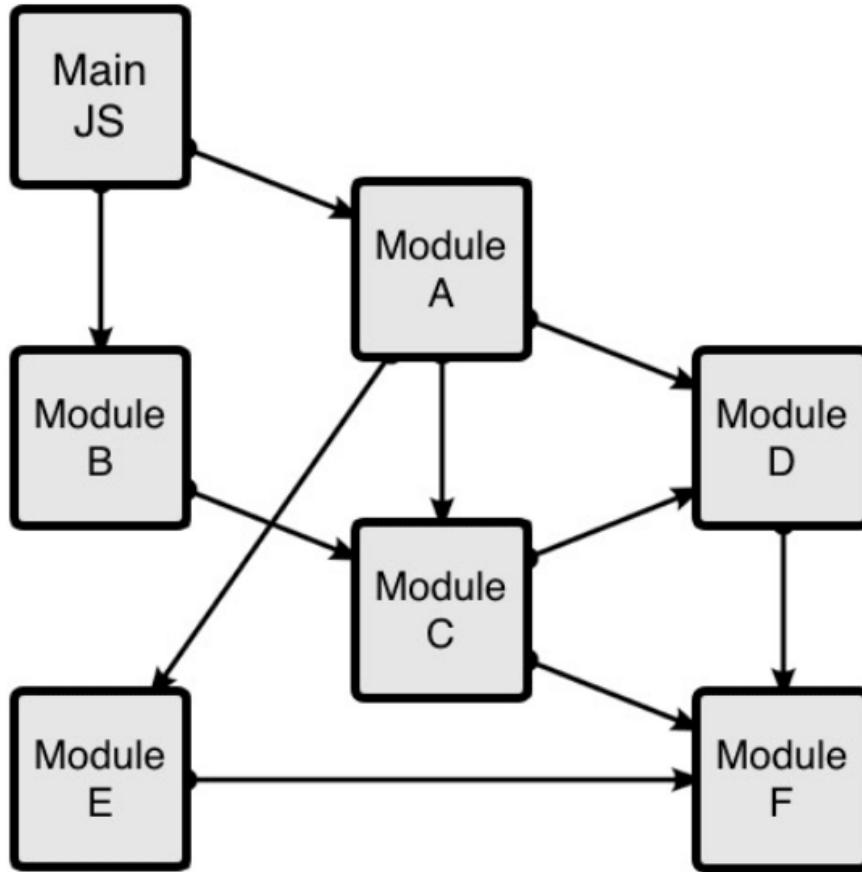
Grunt/Gulp Build Process



Browserify

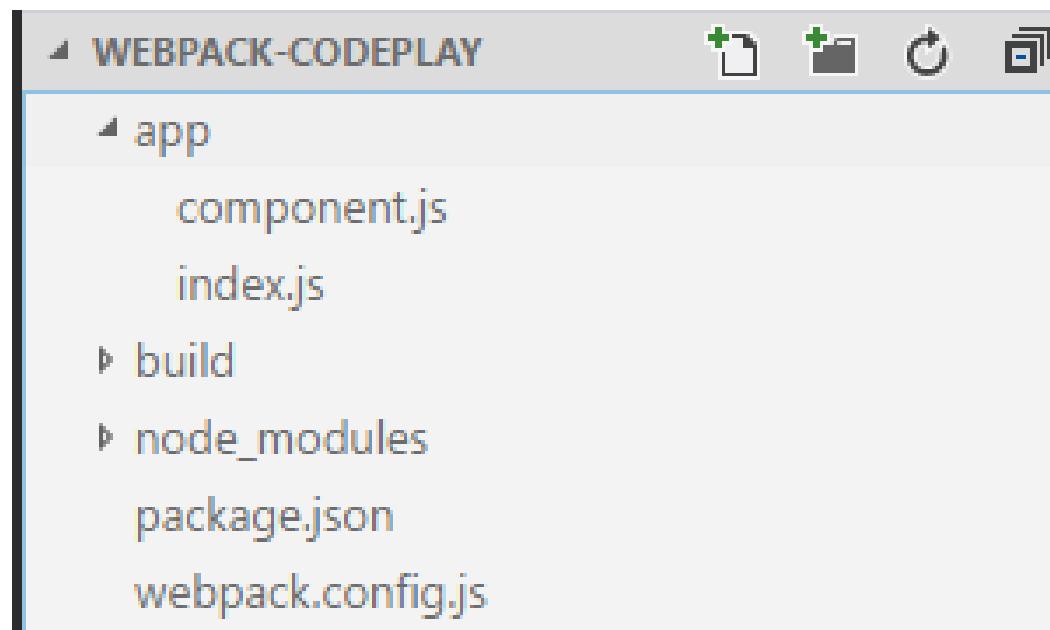
- There are a variety of packaging and bundling tools such as RequireJS (AMD)
- Browserify – alternative to RequireJS as it bundles up node modules to be used in the browser

Webpack



Webpack

Application-Directory Structure



```
› npm install --save webpack@2.2.0-rc.0
```

Webpack configuration

- Using **html-webpack-plugin** to wire up the generated assets with it.

```
e:\CT\ReactJS\webpack-codeplay>npm i html-webpack-plugin --save-dev
```

Why use Webpack?

- Hot Module Replacement
- Bundle Splitting
- Asset Hashing
- Loaders and Plugins

First Webpack Example

Step 1: npm init -y => package.json

```
{  
  "name": "proj1",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "build": "webpack",  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [],  
  "author": "Syed Awase",  
  "license": "MIT",  
  "dependencies": {  
    "webpack": "^2.2.0-rc.0"  
  }  
}
```

Step 2: npm install -save webpack

- Webpack.config.js

```
const path = require('path');  
const config = {  
  entry: './src/index.js',  
  output:{  
    path: path.resolve(__dirname, 'build'),  
    filename: 'bundle.js'  
  }  
};  
module.exports = config;
```

Running your webpack app

```
E:\CT\Webpack\codeplay\proj1>npm run build  
> proj1@1.0.0 build E:\CT\Webpack\codeplay\proj1  
> webpack  
  
keywords if/then/else require v5 option  
Hash: 47583ac714774fec17d4  
Version: webpack 2.2.0-rc.0  
Time: 130ms  
  
 Asset      Size  Chunks              Chunk Names  
bundle.js  2.73 kB     0  [emitted]  main  
 [0] ./src/utilitymodule.js 68 bytes {0} [built]  
 [1] ./src/index.js 92 bytes {0} [built]
```

Babel Modules

- Babel-loader : teaches babel how to work with webpack
- Babel-core : takes in code, parses it and compiles to generate output files
- Babel-preset-env : a Ruleset for telling babel exactly what pieces of ES2015/6/7 syntax to look for and how to turn it into ES5 compatible code.

```
E:\CT\Webpack\codeplay\proj1>npm install --save babel-loader babel-core babel-preset-env
```

Module Notations

Action: CommonJS

- Import a module
 - Const moduleOne = require('./moduleOne');
- Export module
 - Module.exports = moduleOne;

• **javascript loader + rules =>**

Action:ES2015/ES6

- Import a module
 - Import module from './module';
- Export module
 - Export default module

```
-----  
rules:[  
  {  
    use:'babel-loader',  
    test:/\.js$/  
  }  
]
```

Styling/Loaders using Webpack

```
>npm install --save style-loader css-loader
```

CSS-loader

- Gives you the functionality to load/import css files

Style-loader

- Takes CSS imports and adds them to the HTML document

```
module:{  
  rules:[  
    {  
      use:'babel-loader',  
      test:/\.js$/  
    },  
    {  
      use:['style-loader','css-loader'],  
      test:/\.css$/  
    }  
  ]  
}
```

Extract Text Webpack Plugin

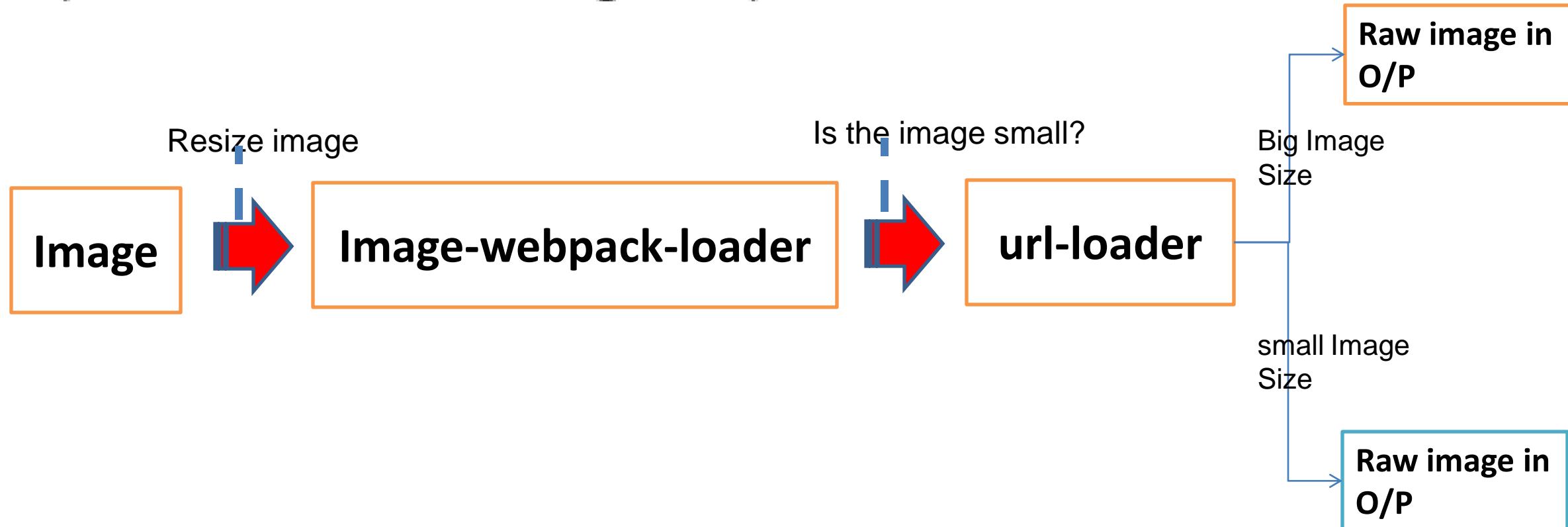
```
npm install --save extract-text-webpack-plugin@2.0.0-beta.4
```

- Extract text webpack plugin emits out style.css as an external stylesheet to be explicitly included in the html file.

```
module:{  
    rules:[  
        {  
            use:'babel-loader',  
            test:/\.js$/  
        },  
        {  
            loader:ExtractTextPlugin.extract({  
                loader:'css-loader'  
            }),  
            test:/\.css$/  
        }  
    ]  
},  
plugins:[  
    new ExtractTextPlugin('style.css')  
]
```

Handling images with webpack

```
npm install --save image-webpack-loader url-loader
```



Code Splitting with webpack

- A feature to split your codebase into “chunks” which are loaded on demand. Some other bundlers call them “layers”, “rollups” or “fragments”.
- AMD and CommonJS specify different methods to load code **on demand.**
- CommonJS
 - `Require.ensure(dependencies,callback)`
- AMD
 - `Require(dependencies,callback)`



Webpack.config.js

```
1 const path = require('path');
2 const HtmlWebpackPlugin=require('html-webpack-plugin');
3 const PATHS ={
4     app:path.join(__dirname,'app'),
5     build:path.join(__dirname, 'build')
6 };
7 module.exports={
8     //Entry accepts a path or any object of entries
9     entry:{
10         app:PATHS.app
11     },
12     output:{
13         path:PATHS.build,
14         filename:'[name].js'
15     },
16     plugins:[
17         new HtmlWebpackPlugin({
18             title:'Webpack demo'
19         })
20     ]
21 };
```

Webpack.config.js :alt config

- As the project grows, it becomes necessary to split it up per environment so as to have enough control over the build result.
- Maintaining configuration with a single file and branching using **webpack-merge**

```
e:\CT\ReactJS\webpack-codeplay>npm i webpack-merge --save-dev
```

```
1 const path = require('path');
2 const HtmlWebpackPlugin=require('html-webpack-plugin');
3 const merge = require('webpack-merge');
4 const PATHS ={
5   app:path.join(__dirname,'app'),
6   build:path.join(__dirname, 'build')
7 };
8 const common={
9   //Entry accepts a path or any object of entries
10  entry:{
11    app:PATHS.app
12  },
13  output:{
14    path:PATHS.build,
15    filename:'[name].js'
16  },
17  plugins:[
18    new HtmlWebpackPlugin({
19      title:'Webpack demo'
20    })
21  ]
22 };
23 var config;
24 switch(process.env.npm_lifecycle_event){
25   case 'build':
26     config=merge(common,{});
27     break;
28   case 'test':
29     config=merge(common,{});
30     break;
31   default:
32     config=merge(common,{});
33 }
34 module.exports = config;
```

Integrating webpack-validator

- Validating your web-pack configuration against a schema and warn if we are trying to do something not sensible.

```
e:\CT\ReactJS\webpack-codeplay>npm i webpack-validator --save-dev
```

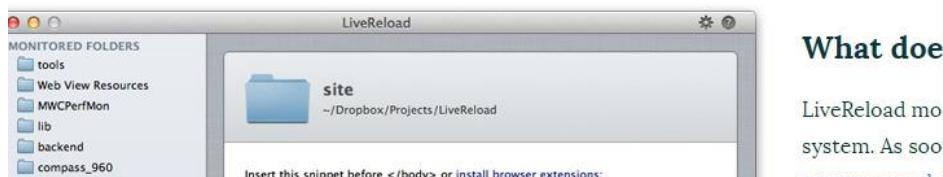
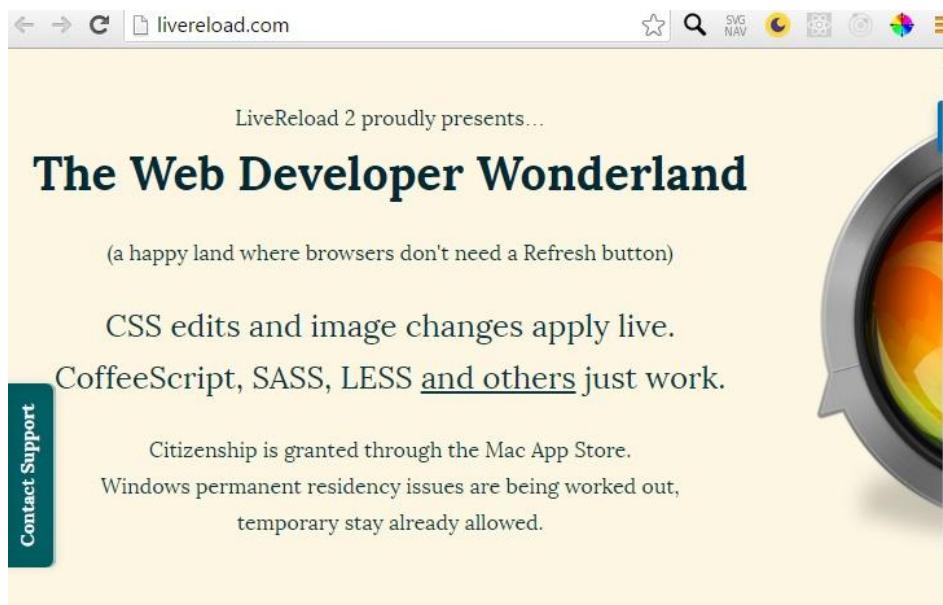
Enable watch mode

- **webpack --watch**

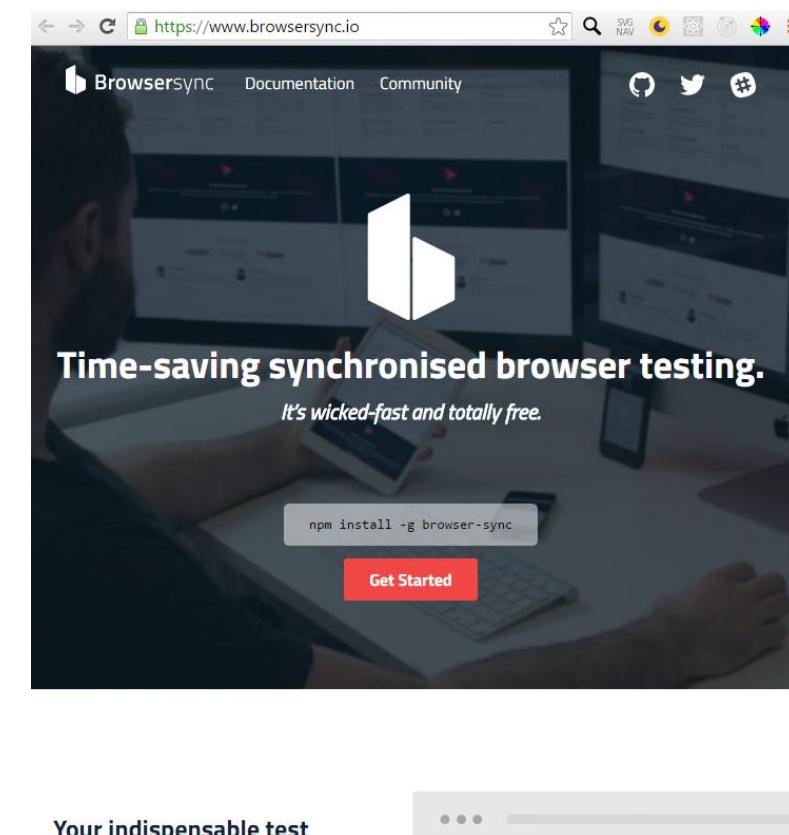
```
const path = require('path');
const HtmlWebpackPlugin=require('html-webpack-plugin');
const merge = require('webpack-merge');
const validate=require('webpack-validator');
const PATHS ={
    app:path.join(__dirname,'app'),
    build:path.join(__dirname, 'build')
};
const common={
    //Entry accepts a path or any object of entries
    entry:{
        app:PATHS.app
    },
    output:{
        path:PATHS.build,
        filename:'[name].js'
    },
    plugins:[
        new HtmlWebpackPlugin({
            title:'Webpack demo'
        })
    ]
};
var config;
//detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event){
    case 'build':
        config = merge(common,{});
        break;
    case 'test':
        config = merge(common,{});
        break;
    default:
        config= merge(common,{});
}
module.exports =validate(config);
```

Tools

LiveReload (www.livereload.com)



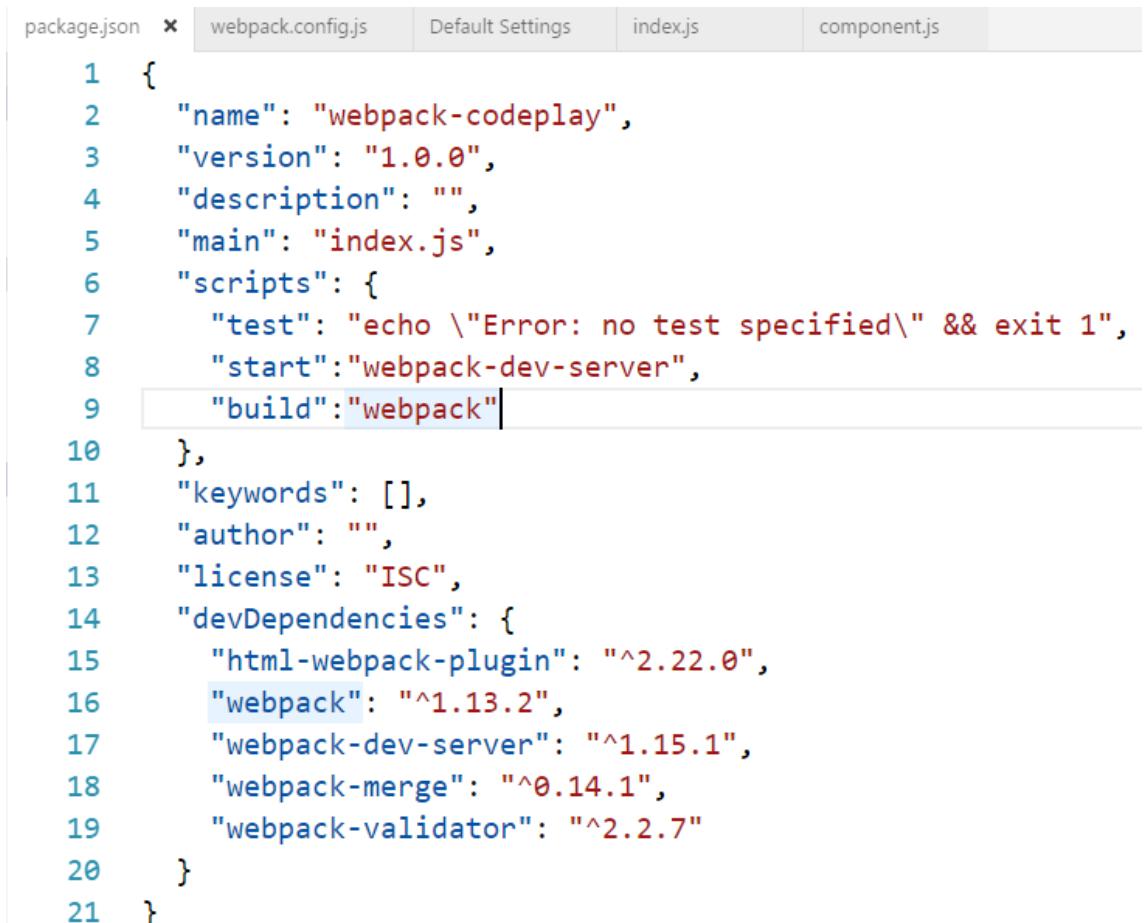
Browsersync (www.browsersync.io)



Webpack-dev-server

e:\CT\ReactJS\webpack-codeplay>npm i webpack-dev-server --save-dev

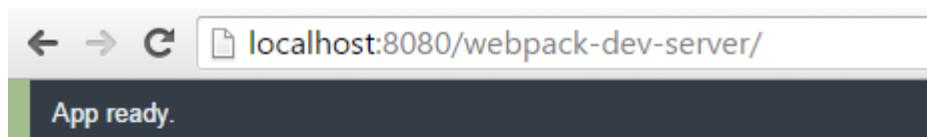
- Webpack-dev-server is a development server running in-memory. It refreshes content automatically in the browser, while you develop your application.
- Also supports an advanced Webpack feature known as **Hot Module Replacement (HMR)** which provides a way to patch the browser state without a full refresh.



```
package.json x webpack.config.js Default Settings index.js component.js
1  {
2    "name": "webpack-codeplay",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\"Error: no test specified\\" && exit 1",
8      "start": "webpack-dev-server",
9      "build": "webpack"
10 },
11 "keywords": [],
12 "author": "",
13 "license": "ISC",
14 "devDependencies": {
15   "html-webpack-plugin": "^2.22.0",
16   "webpack": "^1.13.2",
17   "webpack-dev-server": "^1.15.1",
18   "webpack-merge": "^0.14.1",
19   "webpack-validator": "^2.2.7"
20 }
21 }
```

Running your application with Webpack

```
e:\CT\ReactJS\webpack-codeplay>npm build  
e:\CT\ReactJS\webpack-codeplay>npm start  
> webpack-codeplay@1.0.0 start e:\CT\ReactJS\webpack-codeplay  
> webpack-dev-server  
  
http://localhost:8080/webpack-dev-server/
```





<https://webpack.github.io/docs/comparison.html>

COMPARISON

Feature	webpack/webpack	jrburke/requirejs	substack/node-browserify	jspm/jspm-cli	rollup/rollup
CommonJS <code>require</code>	yes	only wrapping in <code>define</code>	yes	yes	commonjs-plugin
CommonJS <code>require.resolve</code>	yes	no	no	no	no
CommonJS exports	yes	only wrapping in <code>define</code>	yes	yes	commonjs-plugin
AMD <code>define</code>	yes	yes	<code>deamdfy</code>	yes	no
AMD <code>require</code>	yes	yes	no	yes	no
AMD <code>require</code> loads on demand	yes	with manual configuration	no	yes	no
ES2015 <code>import / export</code>	yes(vr. 2)	no	no	yes	yes
Generate a single bundle	yes	yes♦	yes	yes	yes
Load each file separate	no	yes	no	yes	no
Multiple bundles	yes	with manual configuration	with manual configuration	yes	no
Additional chunks are loaded on demand	yes	yes	no	<code>System.import</code>	no
Multi pages build with common	with manual		with manual		

SourceMap

- To enable sourcemaps during development, we can use a default known as eval-source-map

Webpack.config.js

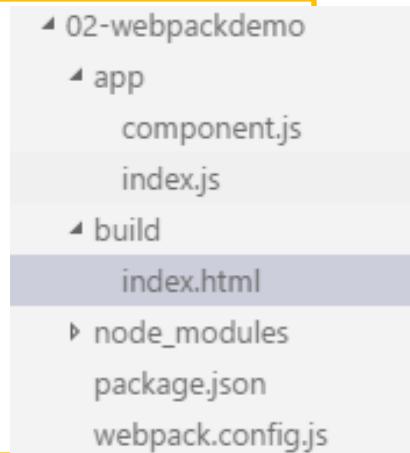
```
//detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event){
  case 'build':
    config = merge(common,{devtool:'eval-source-map'});
    break;
  case 'test':
    config = merge(common,{devtool:'eval-source-map'});
    break;
  default:
    config = merge(common,{devtool:'eval-source-map'});
}
module.exports =validate(config);
```

Webpack Application: Playbook

STEP: I

```
npm init -y
npm i webpack --save-dev
npm i webpack-merge --save-dev
npm i webpack-validator --save-dev
npm i html-webpack-plugin --save-dev
npm i webpack-dev-server --save-dev
npm i npm-install-webpack-plugin --save-dev
```

STEP: II



Webpack Application: Playbook

STEP: III

- create webpack.config.js

```
const path = require('path');
const HtmlWebpackPlugin=require('html-webpack-plugin');
const merge = require('webpack-merge');
const validate=require('webpack-validator');
const PATHS ={
    app:path.join(__dirname,'app'),
    build:path.join(__dirname, 'build')
};
const common={
    //Entry accepts a path or any object of entries
    entry:{
        app:PATHS.app
    },
    output:{
        path:PATHS.build,
        filename:'[name].js'
    },
    plugins:[
        new HtmlWebpackPlugin({
            title:'Webpack demo'
        })
    ]
};
var config;
//detect how npm is run and branch based on that
switch(process.env.npm_lifecycle_event){
    case 'build':
    config = merge(common,{});
    break;
    case 'test':
    config = merge(common,{});
    break;
    default:
    config= merge(common,{});
}
module.exports =validate(config);
```



Webpack Application:Playbook

STEP: IV:
app/component.js

```
module.exports = function(){
  var element = document.createElement('h1');
  element.innerHTML ='Hello Syed Awase';
  return element;
};
```

STEP: V:
app/index.js

```
var component = require('./component');
var app = document.createElement('div');
document.body.appendChild(app);
app.appendChild(component());
```

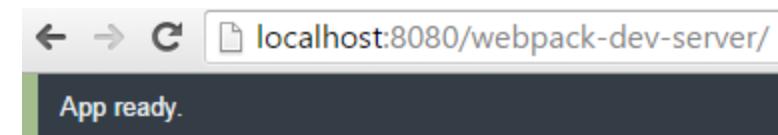
Webpack Application:Playbook

STEP: VI :
build/index.js

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>WebPack Second Demo</title>
</head>
<body>
<div id="app"></div>
<script src="./bundle.js"></script>
</body>
</html>
```

STEP: VII:

```
e:\CT\ReactJS\webpack-codeplay\02-webpackdemo>npm start
> 02-webpackdemo@1.0.0 start e:\CT\ReactJS\webpack-codeplay\02-webpackdemo
> webpack-dev-server --content-base build
http://localhost:8080/webpack-dev-server/
```



RESULT =>

Hello Syed Awase



SYED Awase

PRESENTATION DESIGN PATTERNS

Presentation Design Pattern

UI Centric Design Patterns

MVVM

Model-View-ViewModel

MVP

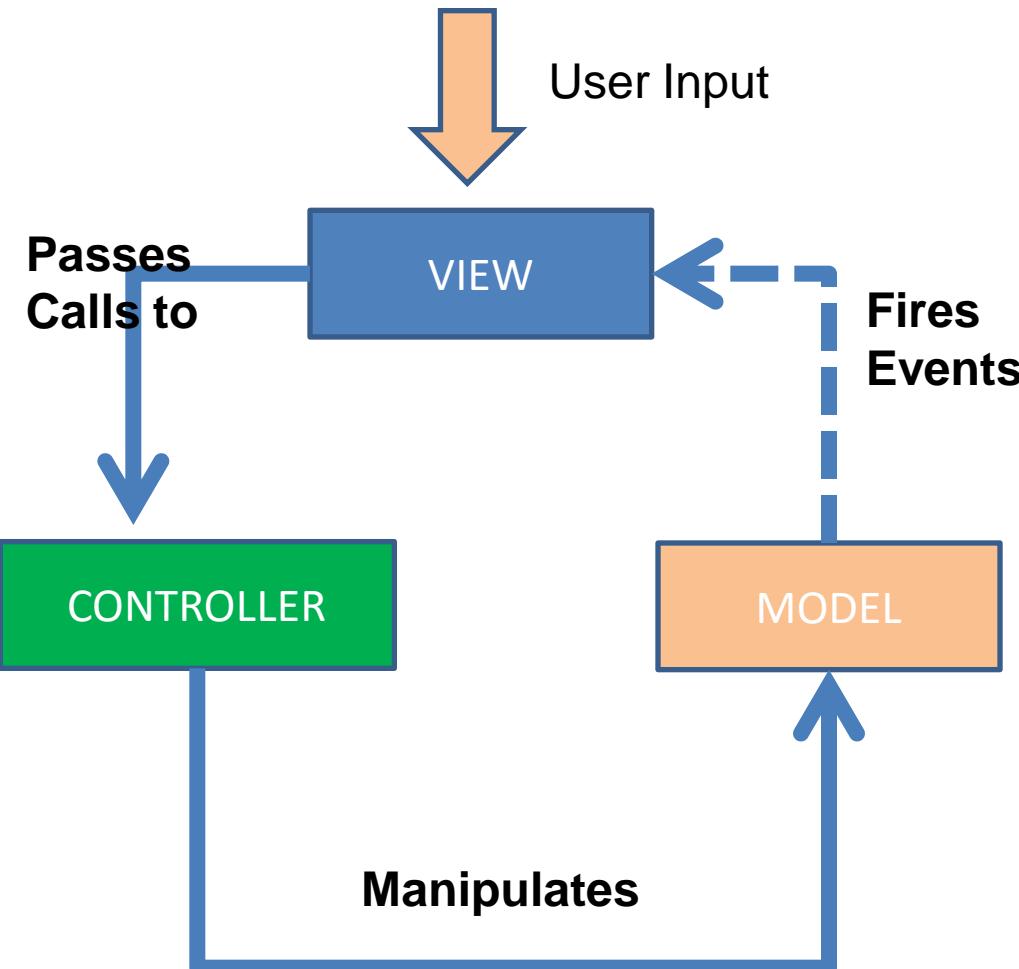
Model-View-Presentation

MVC

Model-View-Controller

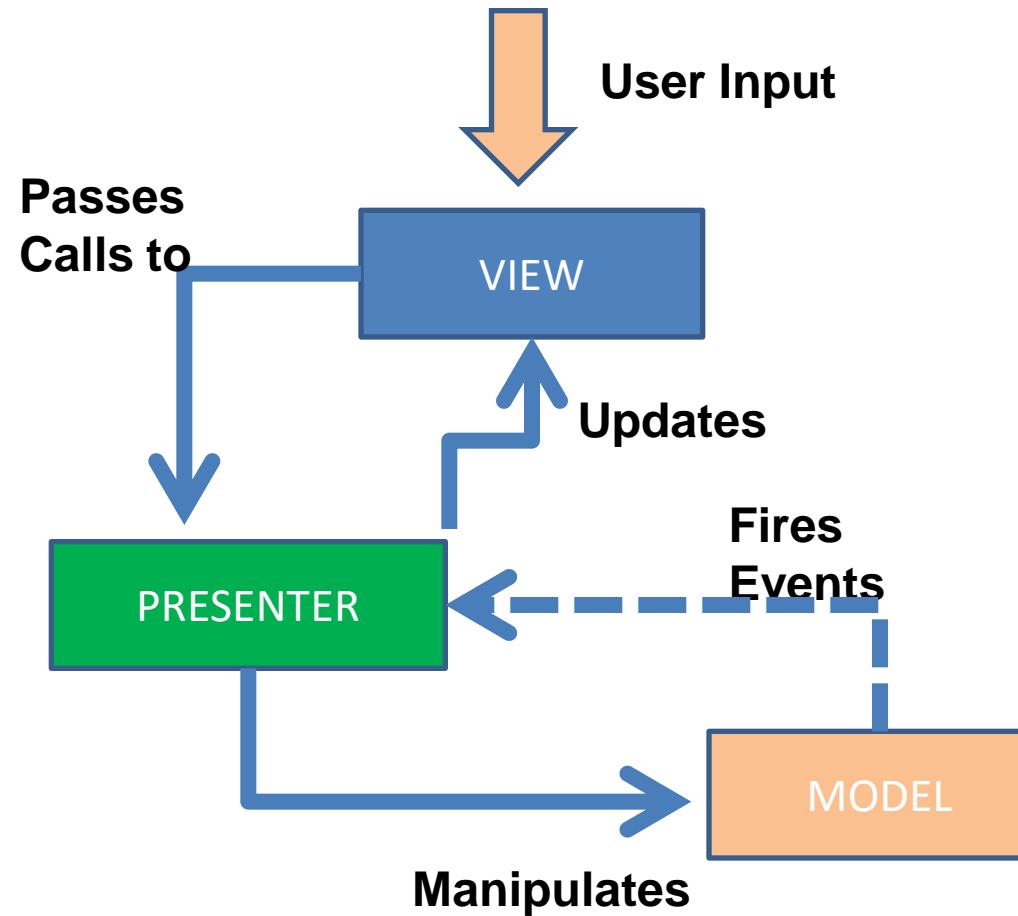
SEPARATION OF CONCERNS**CODE REUSABILITY****UNIT TESTING****SPA****DECOUPLING**

MVC



- Controller contains the logic that alters the model depending on the action triggered by UI.
- View is a composite implementation, A view can switch controllers or a single controller can be used by multiple views. View subscribes to changes done to the model
- controller manipulates the data but asserting the changes from a view perspective

MVP



- Controller is replaced by Presenter , it presents the changes done in model back to view.
- Presenter here takes the responsibility of not only manipulating model but also updating the view.
- MVP Variations

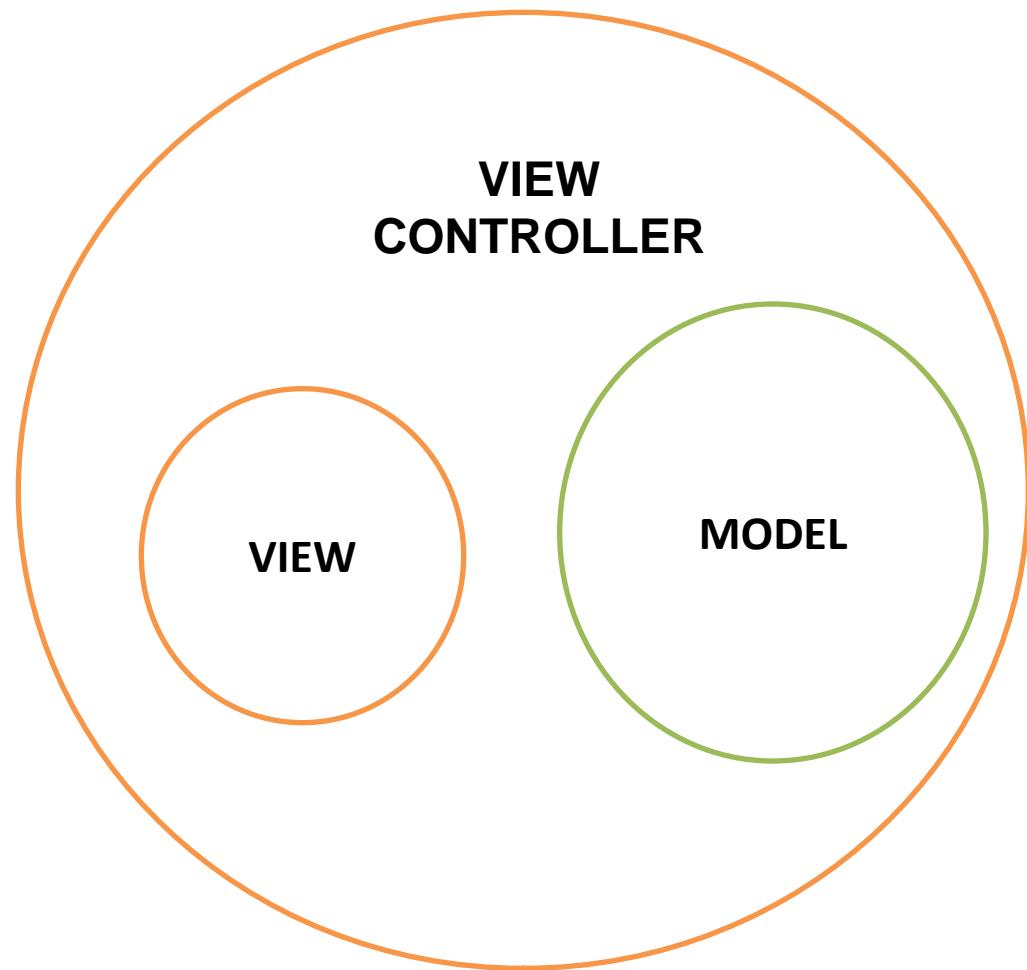
- **Supervising controller** : uses a controller both to handle input response but also to manipulate the view to handle more complex view logic. It leaves simple view behavior to the declarative system, intervening only when effects are needed that are beyond what can be achieved declaratively

- **Passive View** handles this by reducing the behaviour of UI components to the absolute minimum by using a controller that not just handles responses to user events, but also does all the updating of the view. Allows testing to be focused on the controller with little risk of the problem in the view.

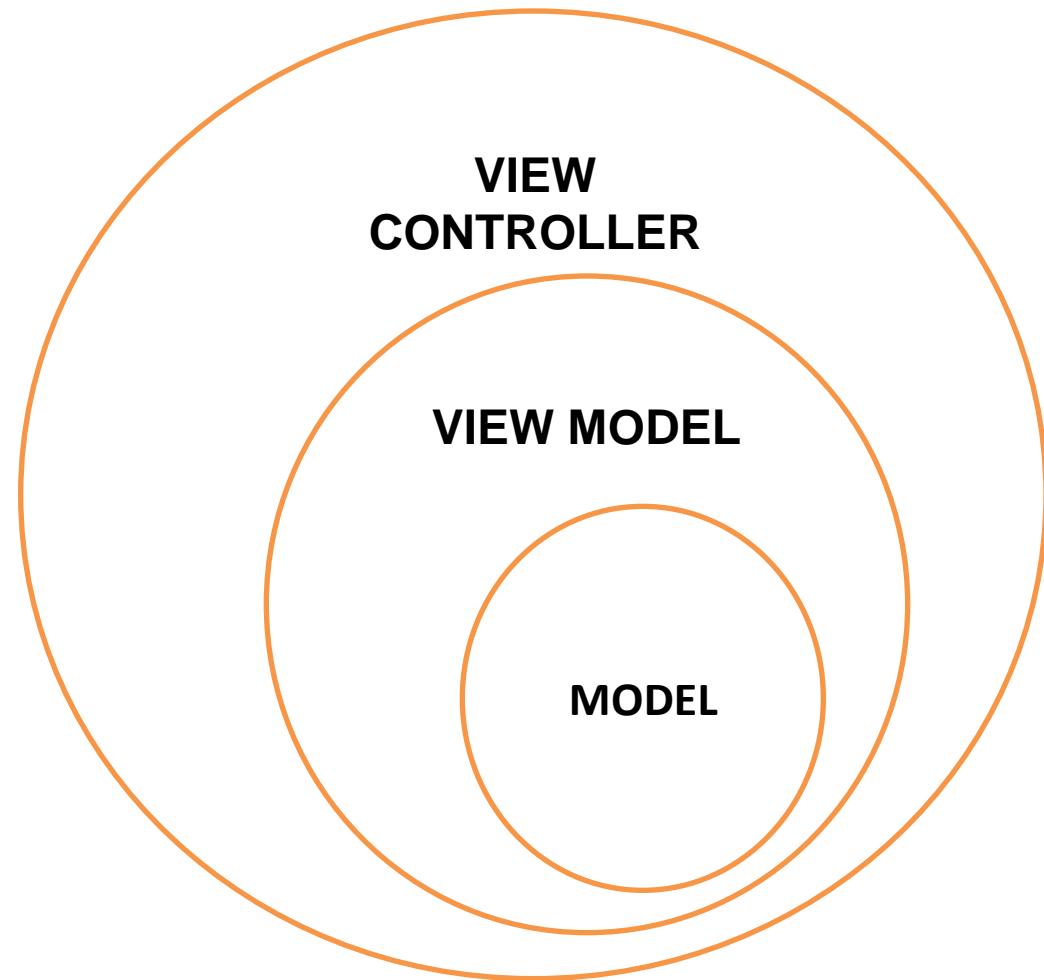
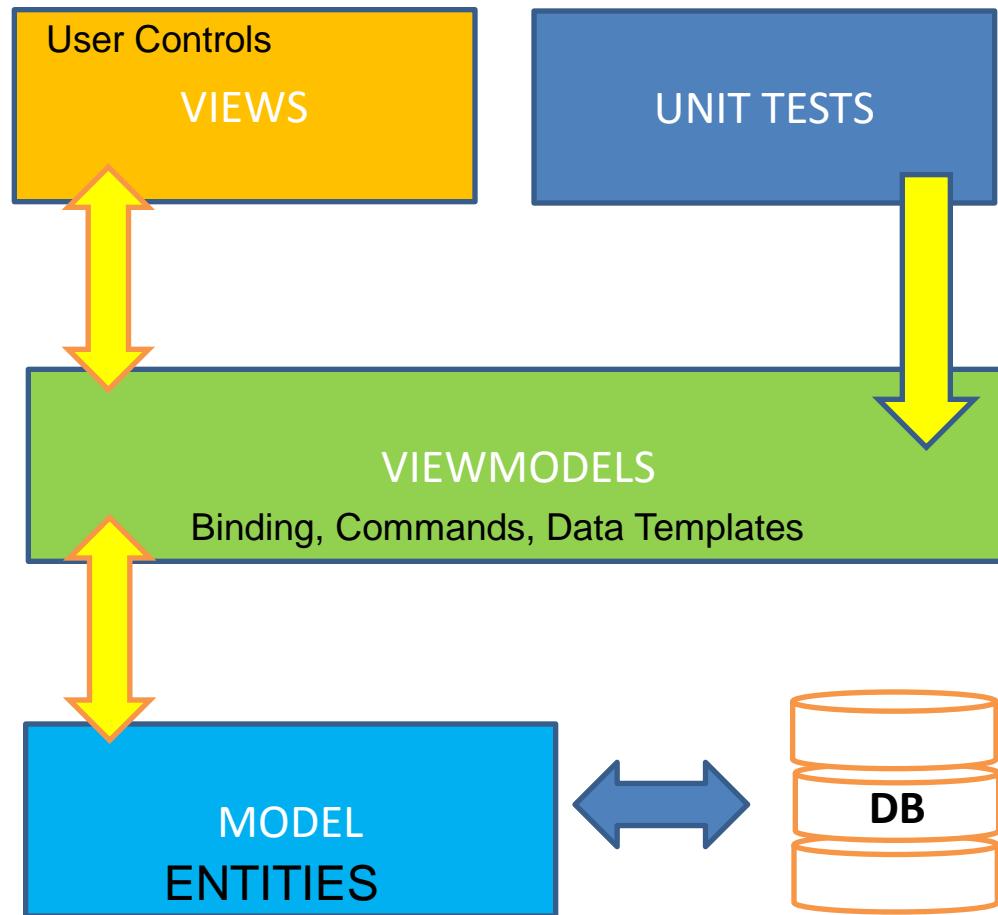
Real World MVC

- Controllers are “almost” not reusable
- Controllers are large
- View is very tight to it’s controller
- Controllers are hard to unit test
-

MVC



MVVM



The Model

- The applications stored data
- Independent of the UI
- Ajax calls to read/write to and from the stored data

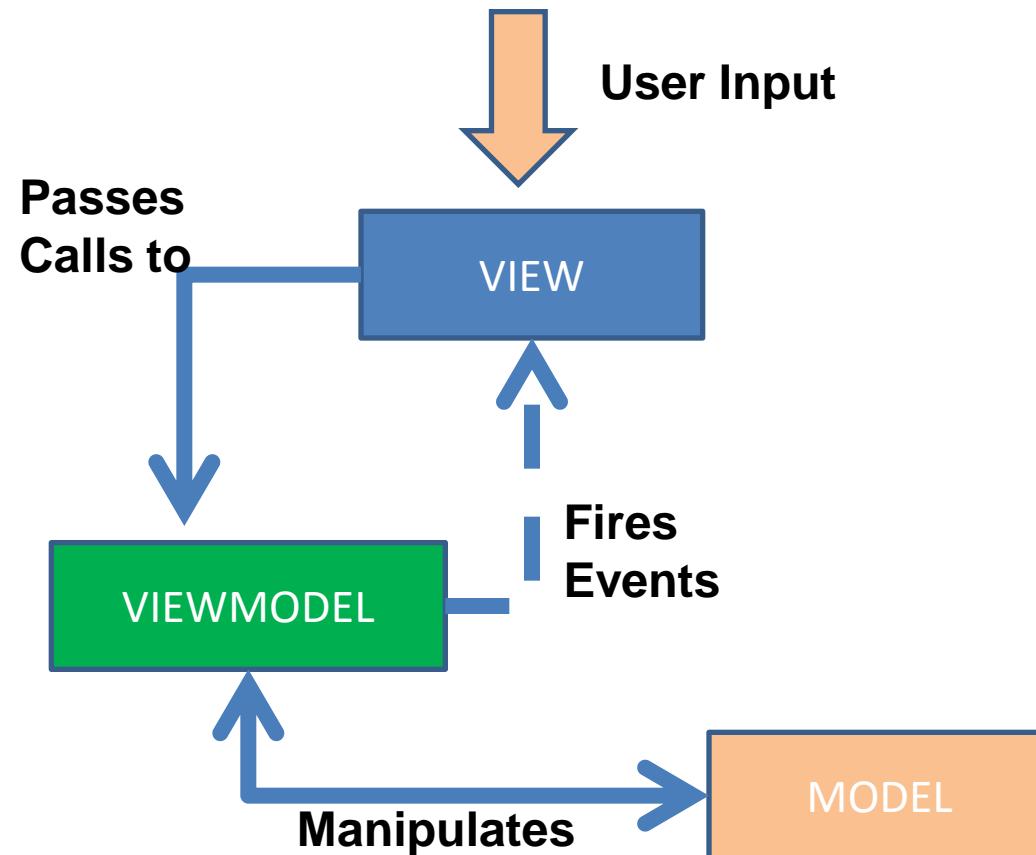
The View Model

- Code representation of the data and operations
- Not the UI itself
- Pure javascript
- Holds unsaved data



The View

- Visible UI
- Displays ViewModel Information
- Updates when the **State** changes
- HTML Document with Declarative Bindings + CSS3 Styling.



MVVM

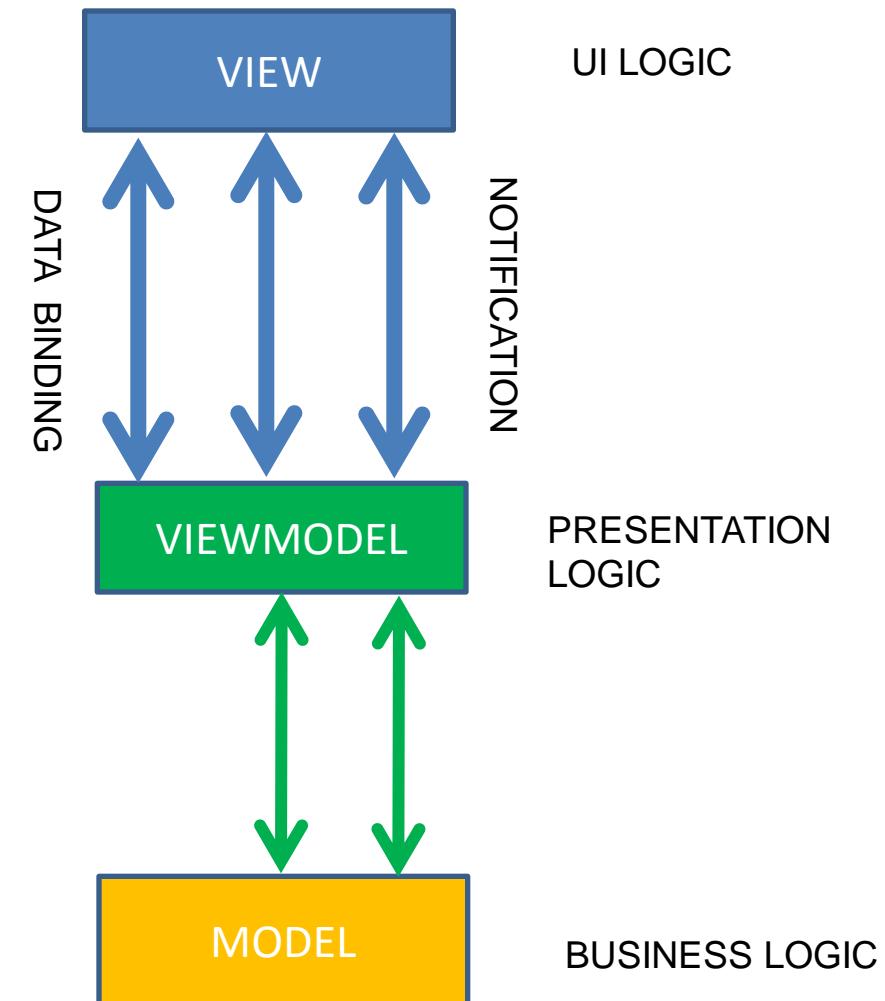
ViewModel does not need a reference to a VIEW

- ViewModel coordinates with one or more models, exposes properties for the view to bind to.
- views know about the ViewModel but not the model
- ViewModel knows about the Model but not the View
- Model Knows about itself
- View binds to properties in the ViewModel
- Changes to properties in the ViewModel automatically propagate to the View – no additional code required
- Data Changes made in the ViewModel, never the View
- More testable than **MVC** or **MVP**

MVVM BENEFITS

- Modularity
 - Decoupling components
 - Allows each component to be versioned independently
- Flexibility
 - Multiple views for one Model (web front end, desktop front end, mobile front end, etc)
 - Replace one component (replace data storage from flat file to database)
- Maintainability
 - Only change one component where bug exists, less risk in late changes
- Testability
 - Each component communicates through

VIEW MODEL ACTS AS A COMPLETE MIRROR OF THE VIEW



View

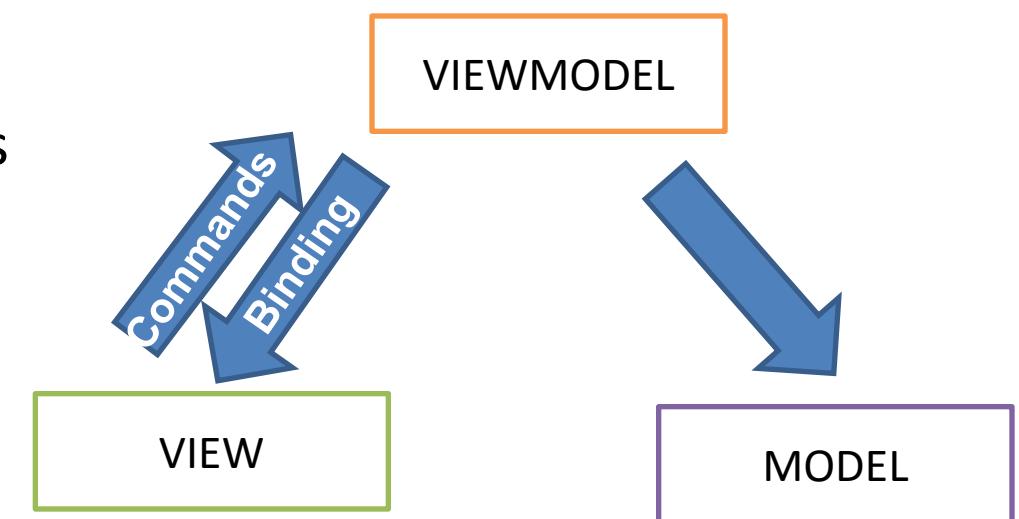
- UserControl based
- XAML
- Minimal code behind
- Data Context set to the associated VIEWMODEL
- No Event Handlers
- Databinding pushes the changes from the ViewModel to View and from View to ViewModel
- Loosely coupled, can easily replace the view without affecting the view model

ViewModel

- Implements INotifyPropertyChanged
- Expose ICommand
- Handle Validation
- Adapter class between the View and the Model
- Listen to the Model's events
- Testable

Model

- Event Based mechanism to signal changes to the ViewModel



MODEL

- Non-visual classes that encapsulate the application's data and business logic
- Can't see ViewModel or View
- Should not contain any use case-specific or user-task-specific behaviour or application logic
- Notifies Other components of any state changes
- May provide data validation and error reporting.

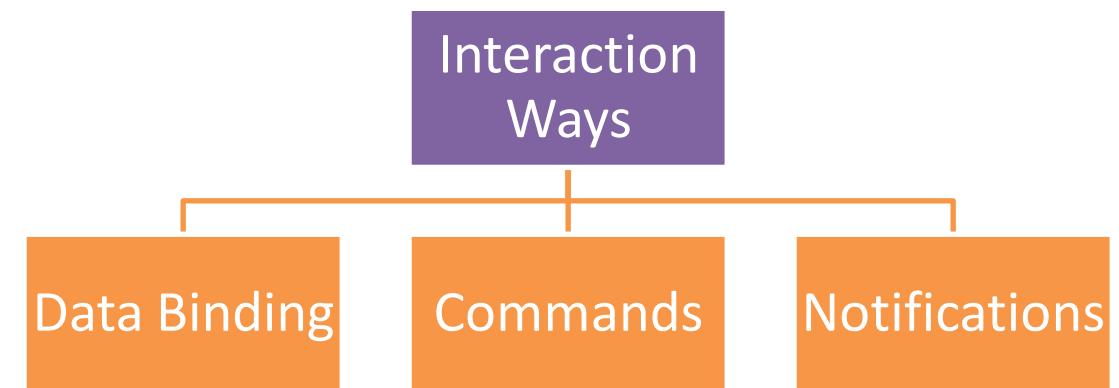
VIEW MODEL

- Non-visual class encapsulates the presentation logic required
- Can't see View (no direct references)
- Coordinates the View's interactions with the model
- May provide data validation and error reporting
- Notifies the view of any state changes

VIEW

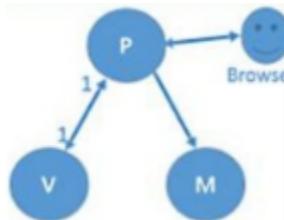
- Visual element defines the controls and their visual layout and styling
- Can see all other components
- Defines and handles UI visual behaviour, such as animations or transitions
- Code behind may contain code that requires direct references to specific UI controls

View \leftrightarrow ViewModel

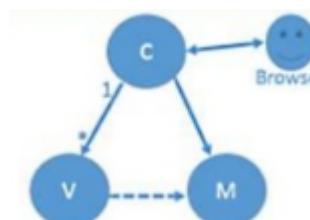


MVP vs. MVC vs. MVVM

MVP(Presenter)	MVC(Controller)	MVVM(ViewModel)
View Communicates with the presenter by directly calling functions on an instance of the presenter	View sends input events to the controller via a callback or registered handler	View binds directly to the View Model
The presenter communicates with the view by taking to an interface implemented by the view	View receives updates directly from the model without having to go through the controller	Changes in view are automatically reflected in ViewModel and viceversa
Use where binding via a data context is not possible	Use where the connection between the view and the rest of the program is not always available	Use where binding via a data context is possible.

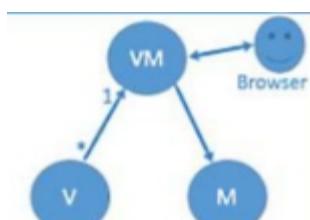


e.g. Windows Forms



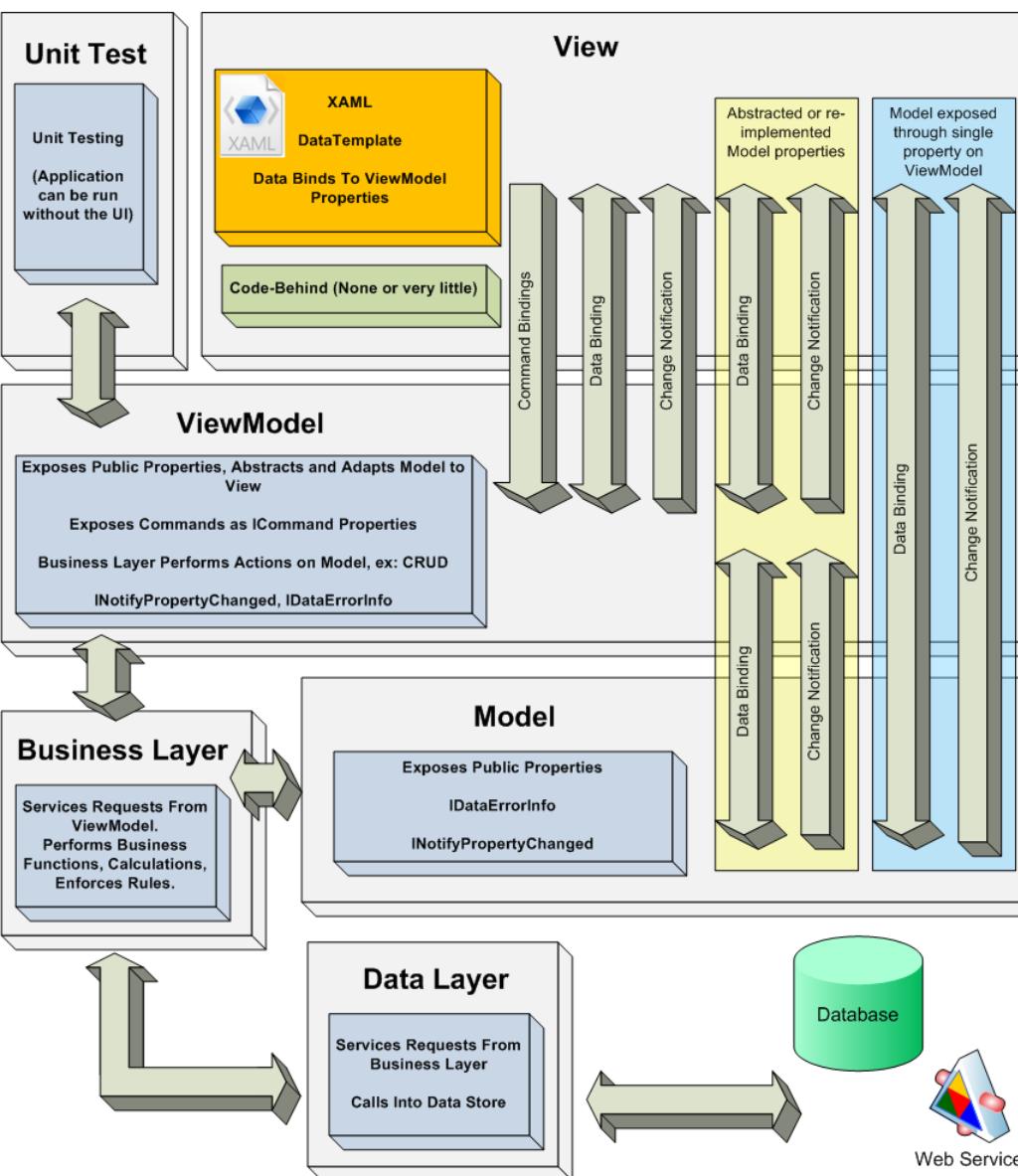
e.g. SmallTalk, ASP.NET MVC

© Syed Awase 2015-16 - KnockOut Ground Up!



e.g. WPF, KO, AngularJS

WPF LOB Application Layers – M-V-VM



MVVM

- **Enable UI Unit Testing**
- **WPF Line of Business Application**
- **Layers represent Separation of concerns and decoupling**
- **Enables designer – developer workflow**
- **Is a natural pattern given WPF's rich data binding that promotes loose coupling**
- **Takes full advantage of WPF's DataTemplates and Commands**
- **UI (view) can be swapped out without touching the UI code**

MVVM

Advantages

- Separation of concerns
- Can use unit tests
- Designers and developers can work synchronously
- Easy to redesign the UI
- Can share code easily

Disadvantages

- Harder to debug
- May affect performance
- More files to serve the architecture



React with ES5

**A JavaScript library for building
composable user interfaces**

SYED AWASE

Resource for React

<https://github.com/zalmoxisus/redux-devtools-extension>

<https://github.com/evgenyrodionov/redux-logger>

<https://github.com/airbnb/enzyme>

<https://github.com/carteblanche/carte-blanche>

<https://github.com/storybooks/react-storybook>

<https://github.com/gaearon/redux-thunk>

<https://github.com/FormidableLabs/victory>

<https://github.com/FormidableLabs/victory-chart>

<https://reactcommunity.org/react-modal/>

Isomorphic JavaScript

- ReactJS can run on both the client or the server
 - Render HTML from JavaScript app on server
 - Browser loads with full HTML
 - JavaScript loads and bootstraps the application
- Universal javascript

Core Technologies

Node &npm	Packages
React	Components
React Router	Routing
Flux	Data Flows
Browserify	Bundler
Gulp	Builds

JS MV* Frameworks

Vanilla JS MV* Issues

1. State
2. Performance
3. Complexity
4. Predictability
5. Testability

Success Criteria for MV*

Framework

1. Isomorphic JavaScript
2. The right simple::powerful ratio
3. The right configurable::opinionated ratio
4. Pluggable modular architecture
5. Performance
6. SEO

React

- A lightweight virtual DOM
- Powerful views without templates
- Unidirectional data flow
- Explicit Mutation
- Battle Proven

React is represented as the **V** in MVC

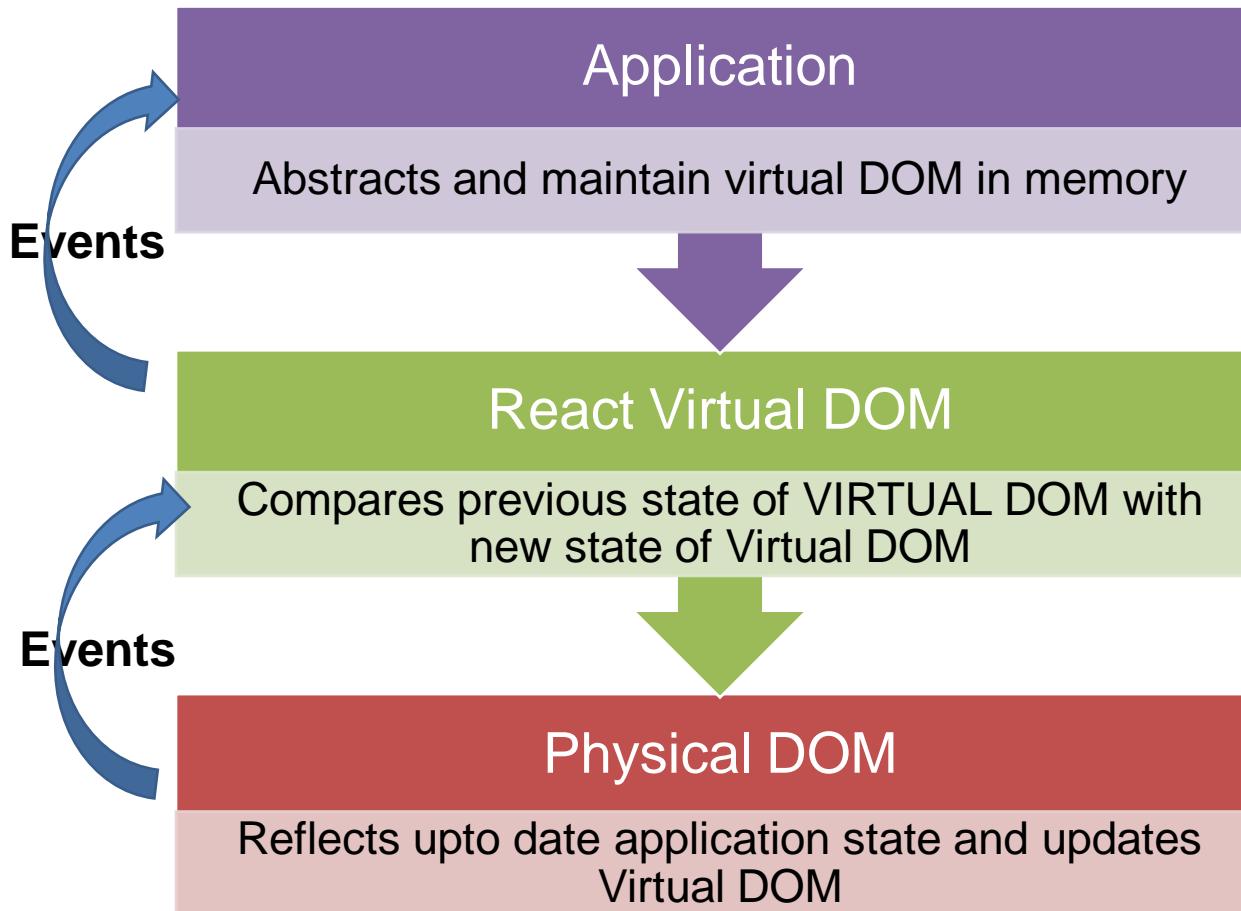
- A React app consists of reusable components.
- Components make code reuse, testing and separation of concerns easy.
- Fast, composable and pluggable.

React has state, it handles mapping from input to state changes, and it renders components.

React

- UI with React can render upto 60 fps
- React render processes has to complete in less than 16 ms.
- React is based on **declarative style of programming**.
- **Declarative style is easier to reason about and prevent bugs.**
- **React components are self-contained units of functionality which publish a simple interface with properties and callbacks.**
- It creates abstract representations of views., which are further broken down into **components**.
- Components encompass both the logic to handle and the display of the view and the view itself.
- DOM manipulation is an expensive operation and should be minimized, it solves this by giving the developer a virtual DOM to render to instead of the actual DOM. It finds difference between the REAL DOM and Virtual DOM and conducts minimum number of DOM operations required to achieve the new state.

How does React Work?



- **Application :** It abstracts application DOM and maintains virtual DOM in memory for handling page interactions and updates from server
- **React Virtual DOM:** It compares previous state of Virtual DOM with new state of Virtual DOM and updates physical DOM with the changed elements.
- **Physical DOM:** It reflects up to date application state and react updates virtual DOM whenever application state changes and which internally updates physical DOM

Why React?

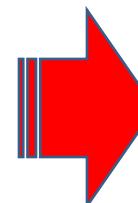
- Fast
- **Unidirectional data flow** from parent components to child components
- Isomorphic
- Easy to debug and Testable
- SPA friendly
- Can co-exist with other frameworks and easy to try it out
- Use JSX with familiar HTML like syntax for writing react applications
- “State” is what makes building UIs hard
- “Data changing over time is the Root of All Evil”
 - React solves this by using composable react components that effectively re-render themselves on each state mutation.
- Freedom from **Domain Specific Language (It's all JavaScript)**
- **Explicit Mutation**

Risk of Two-way Binding

two way data binding allows us to avoid boilerplate code when working with DOM, focus on logic and isolate the logic from templates!

Using visual events, js allows us to build interactions

- field values get stored into a variable
- variable serializes into a JSON and sent through AJAX calls
- DOM gets modified/updated , HTTP Status code changes
- DOM gets updated/modifed.



- Facebook chose to avoid **two-way binding, which it perceives will lead to unpredictable interactions in your applications**
- **Cascading updates**
- **Difficult in debugging**

Approaches to create Components

- ES5createClass
- ES6 class /ES2015 class
- ES5 Stateless function
- ES6 Stateless function /
ES2015 Stateless function
- Many more....

Components

- Two types of components, which are not just react-based but can be visualized in any other component-based UI library or framework.

Presentation Component

- contained components responsible for UI, they are composed with JSX and render using the **render method**.
- **Stateless components**
- **Data is kept in sync using props.**
- **Nearly all markup , Receive data and actions via props**
- **Does not know Redux, typically functional components**

Container Component

- Complements presentation component by providing **states**. **The root component is responsible for managing state**
- **Little to no markup**
- **Pass data and actions down**
- knows about Redux and Often stateful

Source: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.56bb54432

Presentational Component

- Are concerned with how things look
- May contain both presentational and container components inside and usually have some DOM markup and styles of their own
- Often allow containment via **this.props.children**
- Have no dependencies on the rest of the app, such as Flux actions or stores
- Don't specify how the data is loaded or mutated.
- Receive data and callbacks exclusively via props
- Rarely have their own state (when they do, it's UI state rather than data)
- Are written as functional components unless they need state, lifecycle hooks or performance optimization
- Examples: Page, Sidebar, Story, UserInfo, list

Container Component

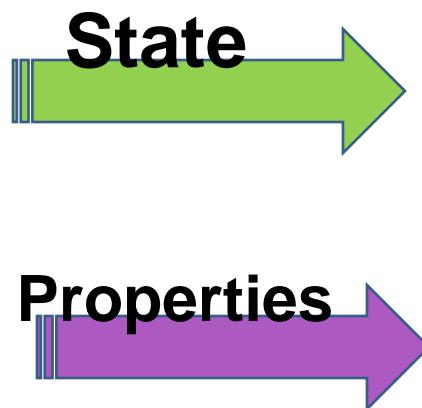
- Are concerned with how things work
- May contain both presentational and container components inside but usually have no DOM markup of their own except for some wrapping divs and never have any styles
- Provide the data and behaviour to presentational or other container components
- Call flux actions and provide these as callbacks to the presentational component
- Are often stateful, as they tend to serve as data sources
- Are usually generated using higher order components such as connect() from React Redux, createContainer() from Relay, or Container.create() from flux utils, rather than written by hand
- Examples: UserPage, Followers Sidebar, Story Container, Followed UserList

Benefits of this approach

Source: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0#.56bb54432

- Better separation of concerns.
Developer understand the app and UI better by writing components this way
- Better reusability. You can use the same presentational component with completely different state sources and turn those into separate container components that can be further reused.
- Presentational components are essentially your app's "palette". You can put them on a single page and let the designer tweak all their variations without touching the app's logic. You can run screen shot regression tests on that page.
- This forces developers to extract "layout components" such as Sidebar, Page, ContextMenu and use `this.props.children` instead of duplicating the same markup and layout in several container components.

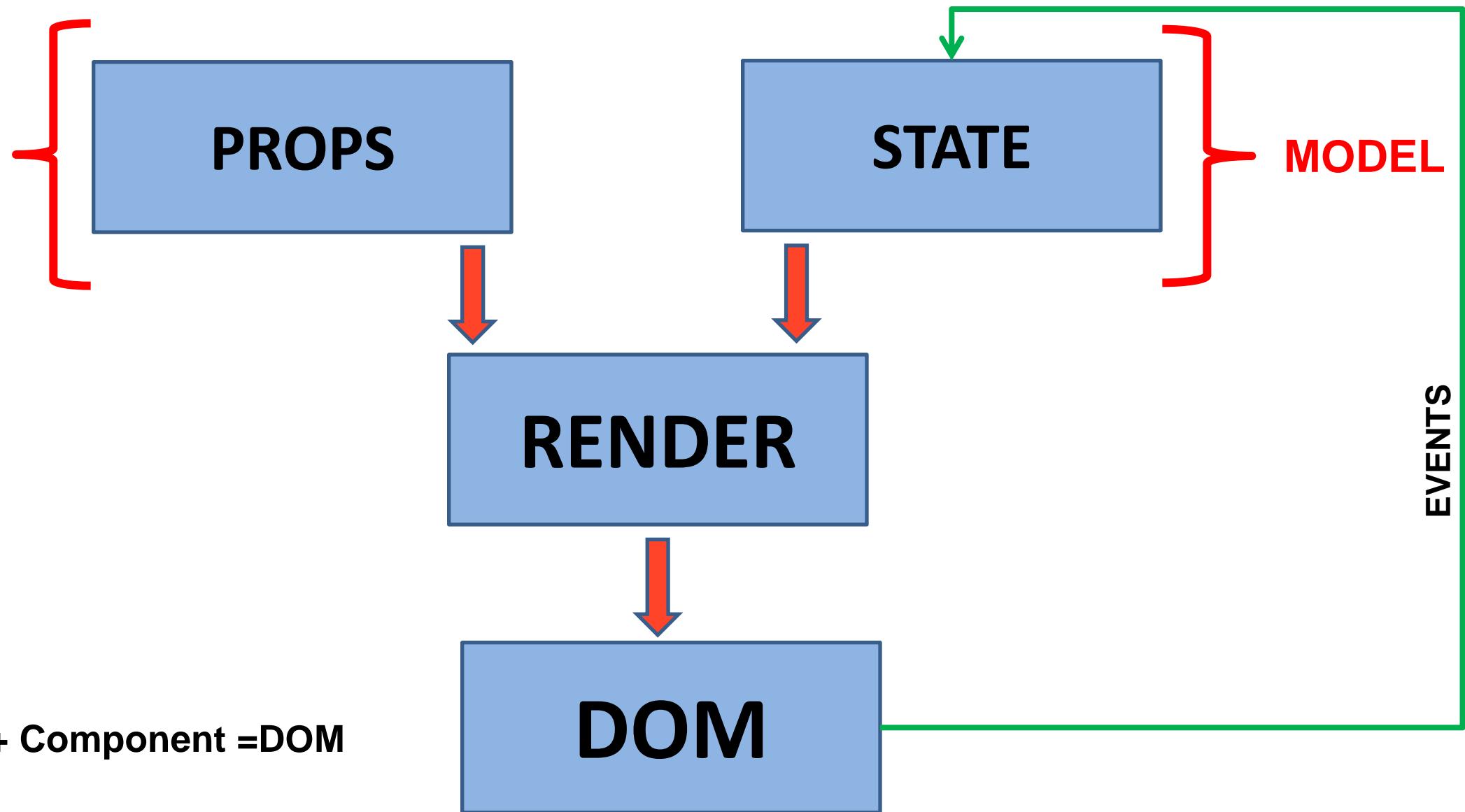
React.js Component



```
var Component = React.createClass({  
  render:function(){  
    return (  
      )  
  }  
});
```

The output of the code block is represented by an orange-outlined arrow containing the word 'HTML'.

State ? : A property on every react component that is used to record interactions with the user.



React Component without JSX

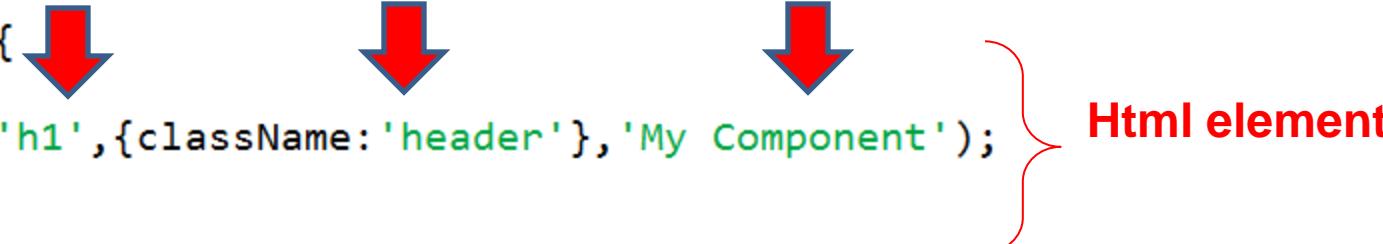
<http://magic.reactjs.net/htmltojsx.htm>

```
(function(){
  var ReactClass = React.createClass({
    render:function(){
      return React.createElement('h1',{className:'header'},'My Component');
    }
  });
}

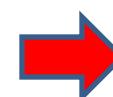
var reactComponentElement = React.createElement(ReactClass);

var reactComponent =ReactDOM.render(reactComponentElement, document.getElementById('react-app'));

})());
```



Output



```
▼ <ReactClass>
  <h1 className="header">My Component</h1>
</ReactClass>
```

React with child components

```
(function(){
  var rootElement = React.createElement('div', {}, 
    React.createElement('h1', {}, "Contacts"),
    React.createElement('ul', {}, 
      React.createElement('li', {}, 
        React.createElement('h2', {}, "Syed Awase"),
        React.createElement('a', {href: 'mailto:sak@sycliq.com'}, 'sak@sycliq.com')
      ),
      React.createElement('li', {}, 
        React.createElement('h2', {}, "Syed Azeez"),
        React.createElement('a', {href: 'mailto:azeez@sycliq.com'}, 'azeez@sycliq.com')
      )
    )
  )
  ReactDOM.render(rootElement, document.getElementById('react-app'));
})();
```



binding

My First React Example using JSX

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>ReactJS first Example</title>
    <!-- react js core files -->
    <script src="lib/react-15.3.1/build/react.js"></script>
    <script src="lib/react-15.3.1/build/react-dom.js"></script>
    <script src="https://npmcdn.com/babel-core@5.8.38/browser.min.js"></script>
  </head>
  <body>
    <h1>First Example with ReactJS 15x Version</h1>
    <div id="container">
      </div>
      <script type="text/babel">
        var APP = React.createClass({
          render:function(){
            return(
              <h1>Hello SYED AWASE!</h1>
            )
          }
        });
        ReactDOM.render(<APP/>, document.getElementById('container'))
      </script>
    </body>
  </html>
```

Core dependencies for ReactJS

React Component in ES5 with JSX

Bind it to the HTML View

My First React Example without JSX

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>ReactJS 15.x without JSX</title>
    <!-- react js core files -->
    <script src="lib/react-15.3.1/build/react.js"></script>
    <script src="lib/react-15.3.1/build/react-dom.js"></script>
    <script src="https://nmpcdn.com/babel-core@5.8.38/browser.min.js"></script>
</head>
<body>
    <h1>ReactJS 15x Example without JSX</h1>
    <div id="container">
    </div>
    <script type="text/babel">
        var DemoTwo = React.createClass({
            displayName: 'DemoTwo',
            render: function(){
                return React.createElement("div", null, "Hello ", this.props.name);
            }
        });
        ReactDOM.render(
            React.createElement(DemoTwo, {name: "Syed Awase"}),
            document.getElementById('container')
        );
    </script>
</body>
</html>
```

Core dependencies for ReactJS

React Component in ES5

Bind it to the HTML View

MountNode/MountPoint

```
▼<div id="container">
  ▼<div data-reactroot>
    <!-- react-text: 2 -->
    "Hello "
    <!-- /react-text -->
    <!-- react-text: 3 -->
    "Syed Awase"
    <!-- /react-text --> == $0
  </div>
</div>
```

My First React Example with State

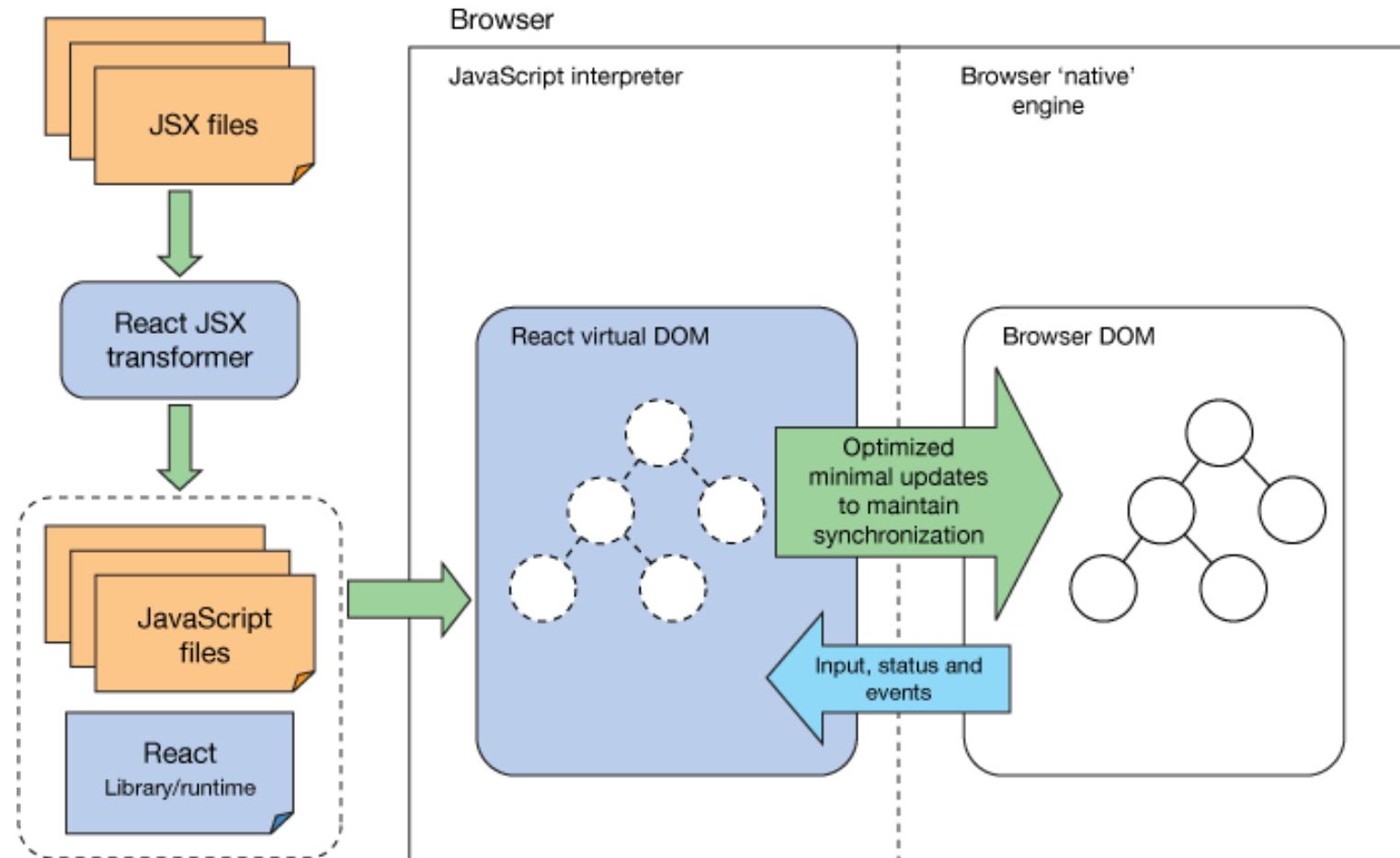
```
(function(){

  var Component = React.createClass({
    getInitialState:function(){
      return{counter:0};           ← Initial State Defined here
    },
    changeValue:function(){
      this.setState({counter:this.state.counter+1}) ← State Updated
    },
    render:function(){
      return (
        <button onClick={this.changeValue}>{this.state.counter}</button>
      )
    }
  });

  ReactDOM.render(<Component/>, document.getElementById('react-app'))]; } Bind it to the HTML View
})();
```

React API

- Top level React API
 - `React.createClass`
 - `React.createElement`
 - `React.render`



React operations source: <https://www.ibm.com/developerworks/library/wa-react-intro/>

Component

Fundamental Unit of
React Application

COMPONENT

**NESTED
COMPONENT**

Corresponds
to an
element in
the DOM

DOM

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>ReactJS 15.x without JSX</title>
  <!-- react js core files -->
  <script src="lib/react-15.3.1/build/react.js"></script>
  <script src="lib/react-15.3.1/build/react-dom.js"></script>
  <script src="https://npmcdn.com/babel-core@5.8.38/browser.min.js"></script>
</head>
<body>
  <h1>ReactJS 15x Example without JSX</h1>
  <div id="container">
  </div>
  <script type="text/babel">
var DemoTwo = React.createClass({
  displayName: 'DemoTwo',
  render:function(){
    return React.createElement("div", null, "Hello ", this.props.name);
  }
});
ReactDOM.render(
  React.createElement(DemoTwo, {name:"Syed Awase"}),
  document.getElementById('container')
);
</script>
</body>
</html>
```

Defining Components

- React is the top-level namespace
- We can create a component using *createClass*

```
<script type="text/babel">
  var APP = React.createClass({}
    render:function(){
      return(
        <h1>Hello SYED AWASE!</h1>
      )
    }
  );
}
```

MUST RETURN A SINGLE ELEMENT NODE

Rendering Components

- Rendering a component means linking it to a DOM element and populating that DOM element.
- Components can be nested inside other components
- Any components can be nested because components are completely self-contained.

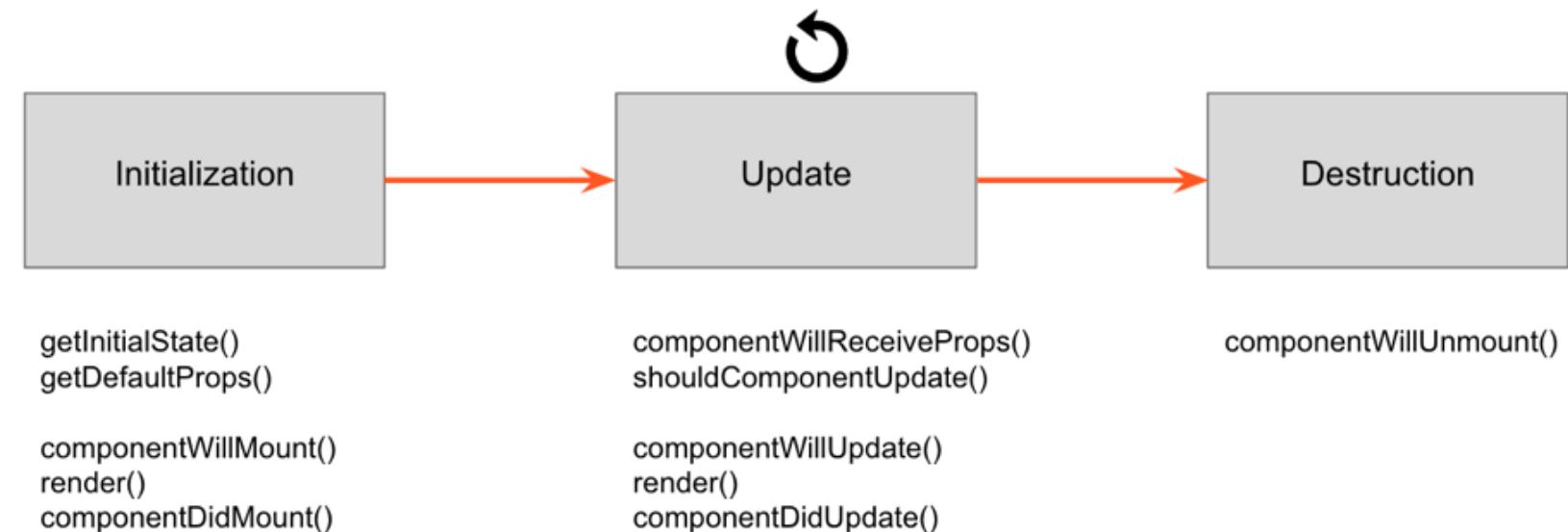
```
ReactDOM.render(<APP/>, document.getElementById('container'))
```

Component Structure

- A component is a React “**class**” instance
- Components have a “**default lifecycle methods**” : hooks for actions at various points in time.
- Components have “**State**” and “**Props**”: **props are immutable and inherited, state is mutable and changes trigger re-render.**
- **JSX is how you compose the view, combine child elements.**
- Extend with your own methods, or augment existing ones.
- Design a component to be **responsible for only one thing** – **Single Responsibility Principle**

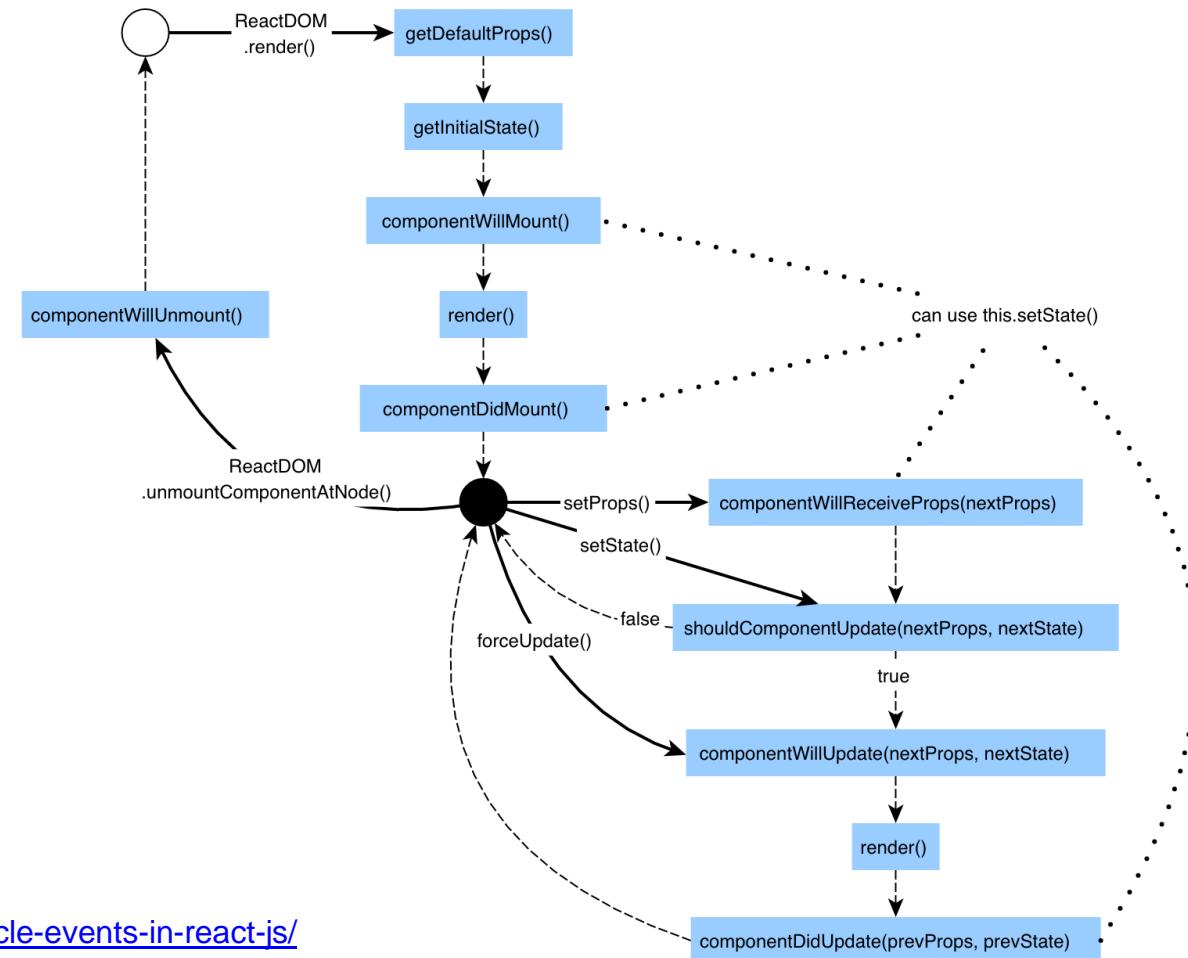
High-level Component Lifecycle

- React Components have lifecycle events that fall into 3 general categories
 - Initialization
 - State/Property Updates
 - Destruction



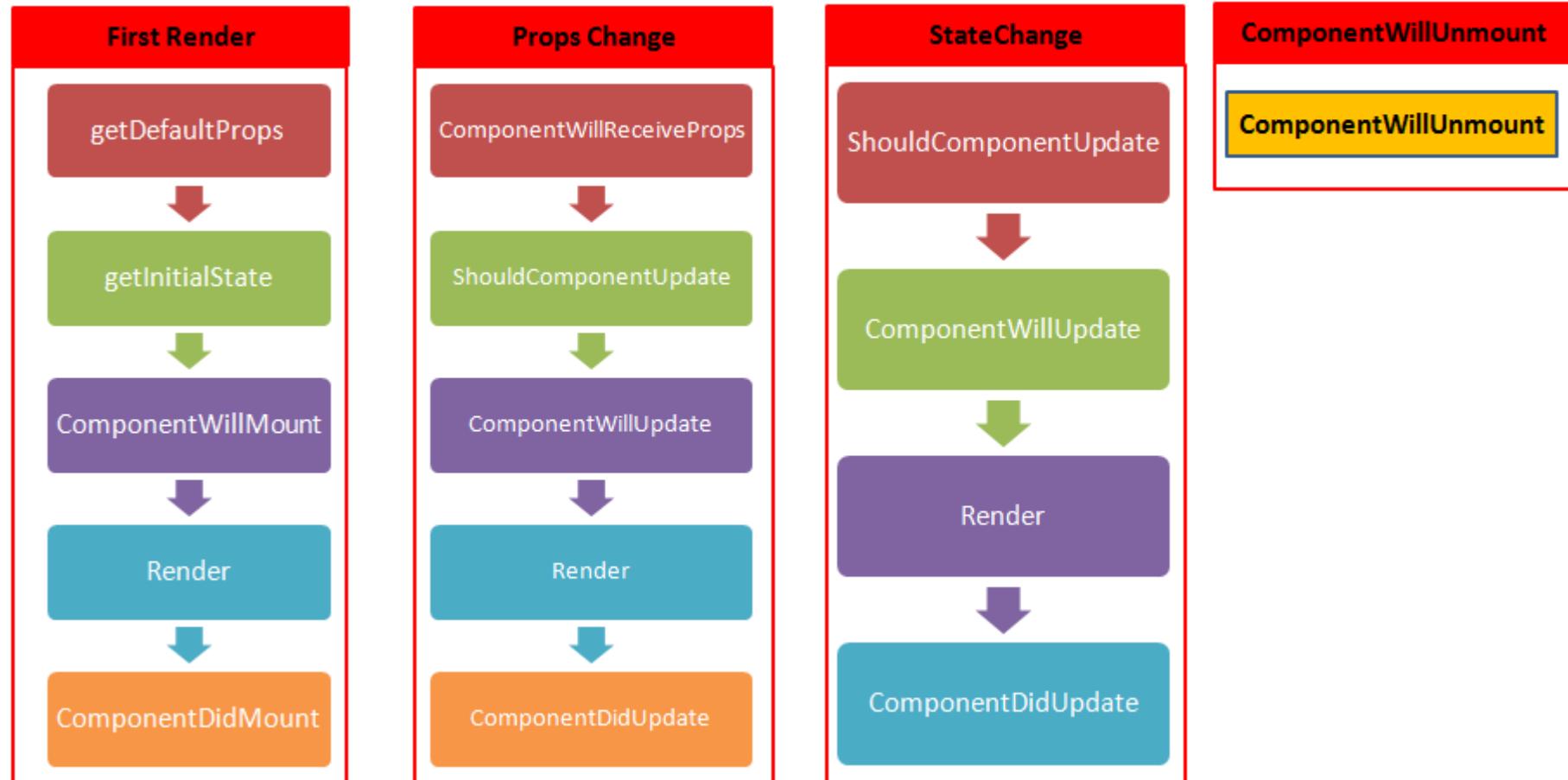
Component Life Cycle

React components are state machines. It's a good practice to keep as many React components stateless as possible.

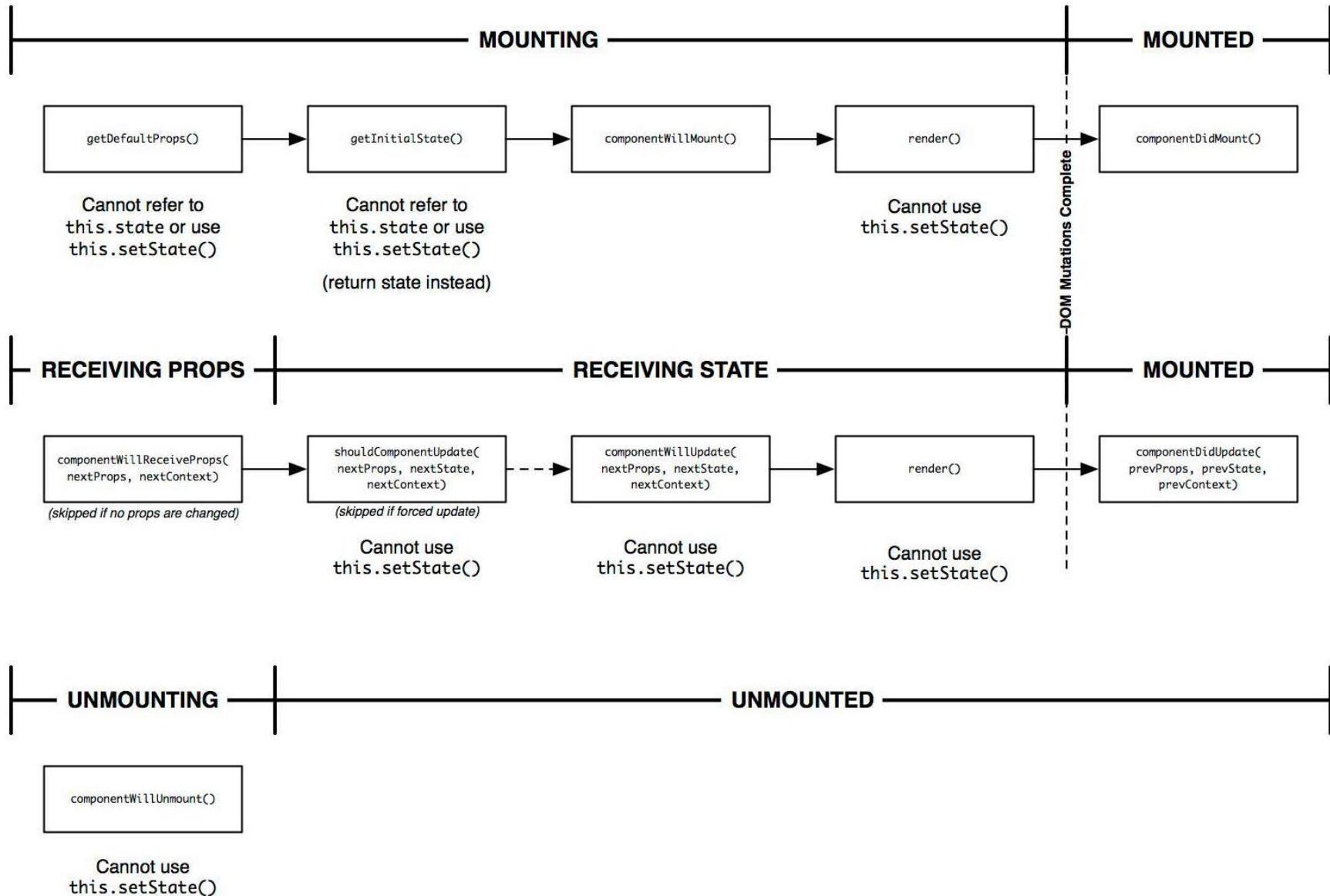


Source: <https://tylermcginnis.com/an-introduction-to-life-cycle-events-in-react-js/>

Component Life Cycle Stages



Remember ASP.NET Page Life Cycle Methods ???



State

For interactivity in the component

Mutable data (changing data)

Is updated by calling **setState()**

Every call to `setState()` triggers a re-render

Props

For data passed to the component

Should be treated as immutable.

Props

- For data passed to the component.
- Props are treated as attributes to the component
- Props can be validate props with **propTypes**, It supports validation of existence, data type or a custom condition
- Look like HTML attributes, but immutable
 - `This.props.username`
- **get defaultProps** : specifies property values to use if they are not explicitly supplied.

PropTypes

- They define **type** and which **props** are required.
- Typechecking is helpful feature, that is available with extensions like **flow** and **TypeScript**.

optionalArray: React.PropTypes.array,
optionalBool: React.PropTypes.bool,
optionalFunc: React.PropTypes.func,
optionalNumber: React.PropTypes.number,
optionalObject: React.PropTypes.object,
optionalString: React.PropTypes.string,

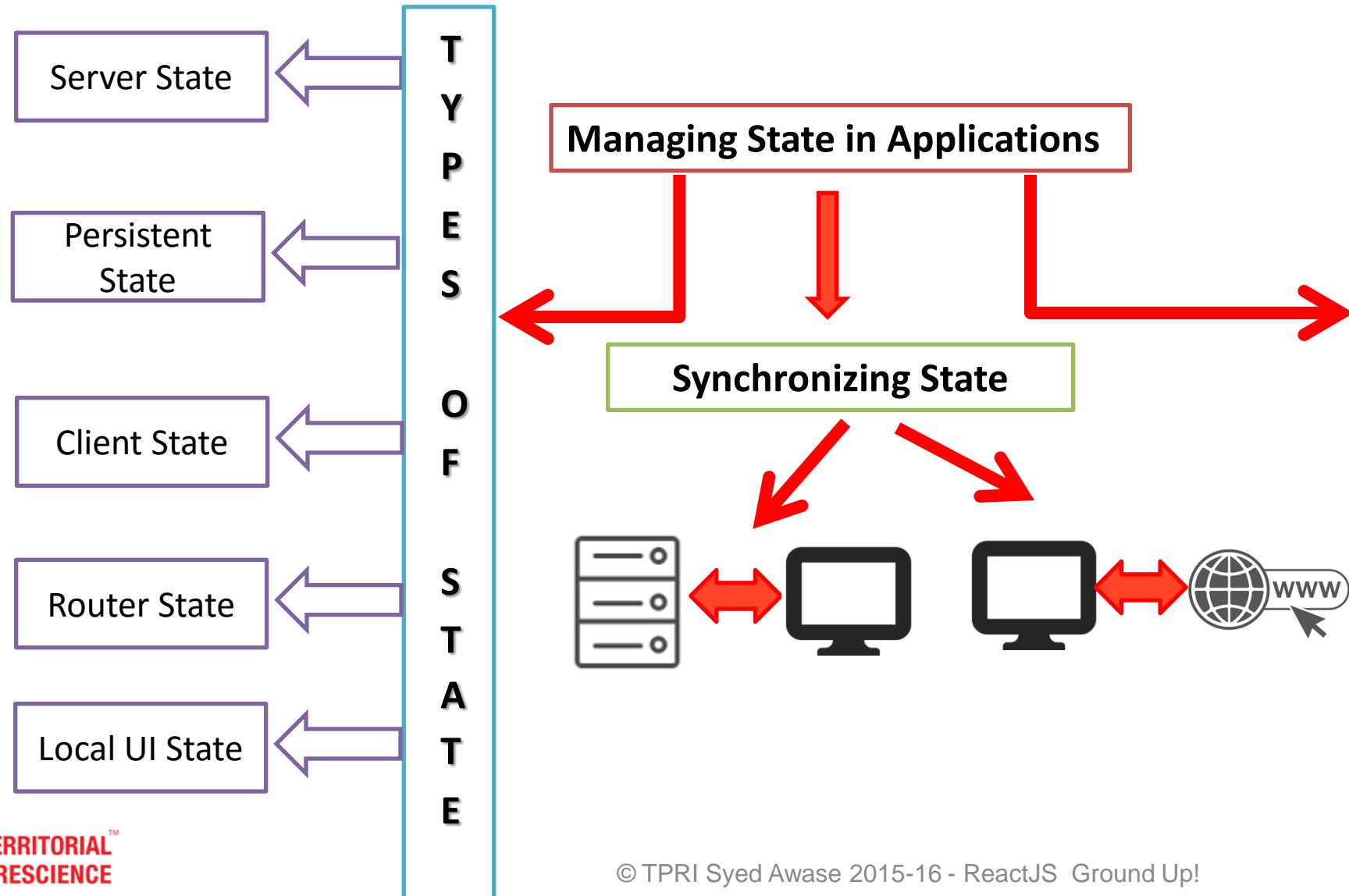
```
propTypes:{  
    size:React.PropTypes.number,  
    position : React.PropTypes.string.isRequired,  
    layout : React.PropTypes.oneOf(['fixed','absolute']),  
    email: function(props, propName, componentName) {  
        if (!/emailRegex/.test(props[email])) {  
            return new Error('Give me a real email!');  
        }  
    }  
}
```

Prop Validation is not run in the production (minified) version. Runs only in Development version.

State

- Holds mutable state
 - This.state.username
- Used when a component needs to change independently of its parent
- Components with state have more complexity
- State causes the component to re-render.
- **getInitialState:** allows a component to populate its initial state
- **setState:** the function used to update the state of a component.
 - It merges the new state with the old state
- **getDefaultProps:** specifies property values to use if they are not explicitly supplied.

Managing State in Application



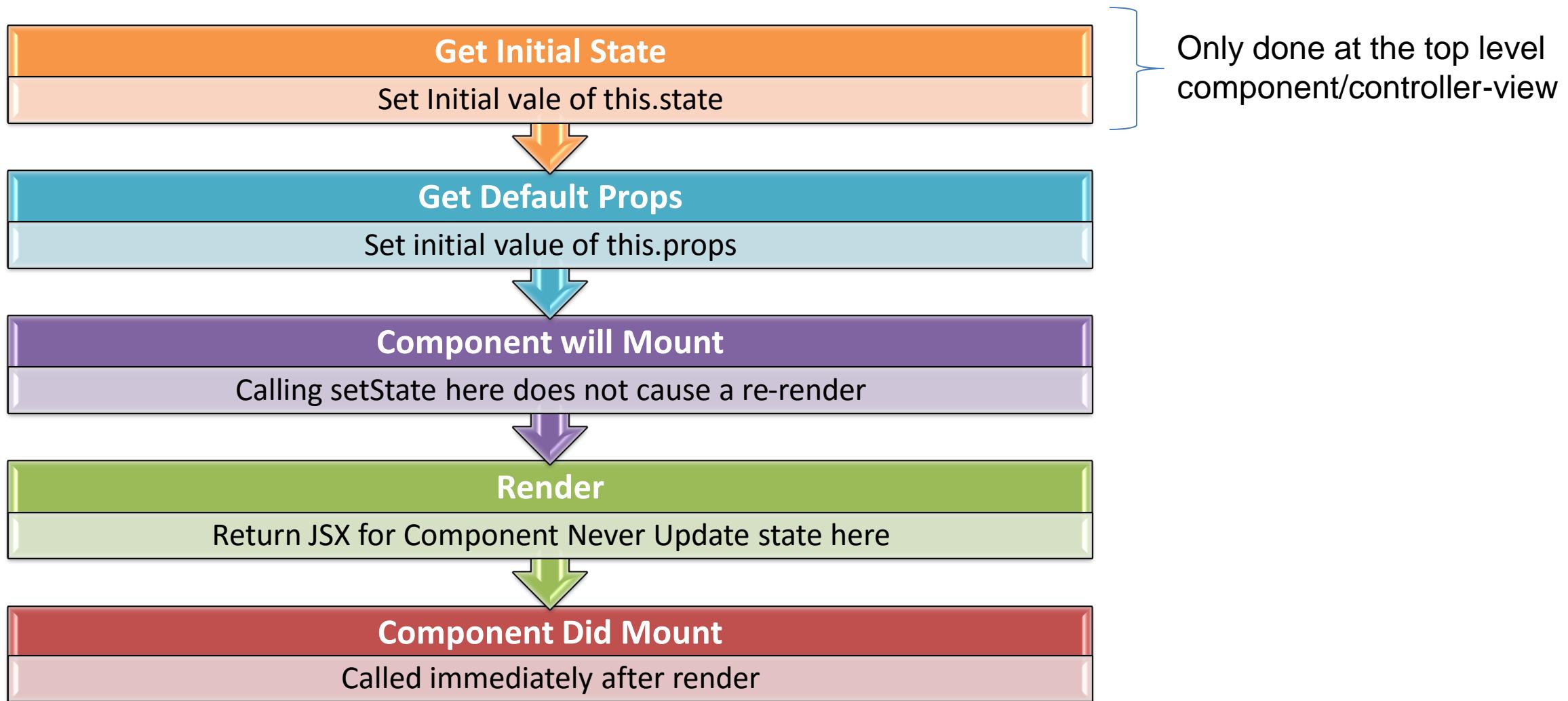
LifeCycle Methods When and Why?

	Lifecycle Method	When	Why
1	componentWillMount	Runs before initial render, both client and server	Good spot to set initial state
2	componentDidMount	Runs immediately after render	Access DOM, integrate with frameworks, set timers, AJAX calls
3	componentWillReceiveProps	Runs when receiving new props. Not called on initial render	Set state before a render.
4	shouldComponentUpdate	Runs before render when new props or state are being received. Not called on initial render	Useful for performance. Returns false to void unnecessary re-render

LifeCycle Methods When and Why?

5	componentWillUpdate	Runs immediately before rendering when new props or state are being received. Not called on initial render	Useful place to prepare for an update. We cannot call setState in this Function.
6	componentDidUpdate	Invoked immediately after component's updates are flushed to the DOM. Not called for the initial render	Can be used to work with the DOM after an update.
7	componentWillUnmount	Runs immediately before component is unmounted/removed from the DOM	Can be used for cleanup of resources etc.

Component Life Cycle: Initial Render



Initialization/Birth/Mounting

1. Initialize/Construction

2. GetDefaultProps() => React.createClass or MyComponent.defaultProps(ES6Class)

3. GetInitialState() => React.createClass or this.state=....(ES6 Constructor)

4. ComponentWillMount()

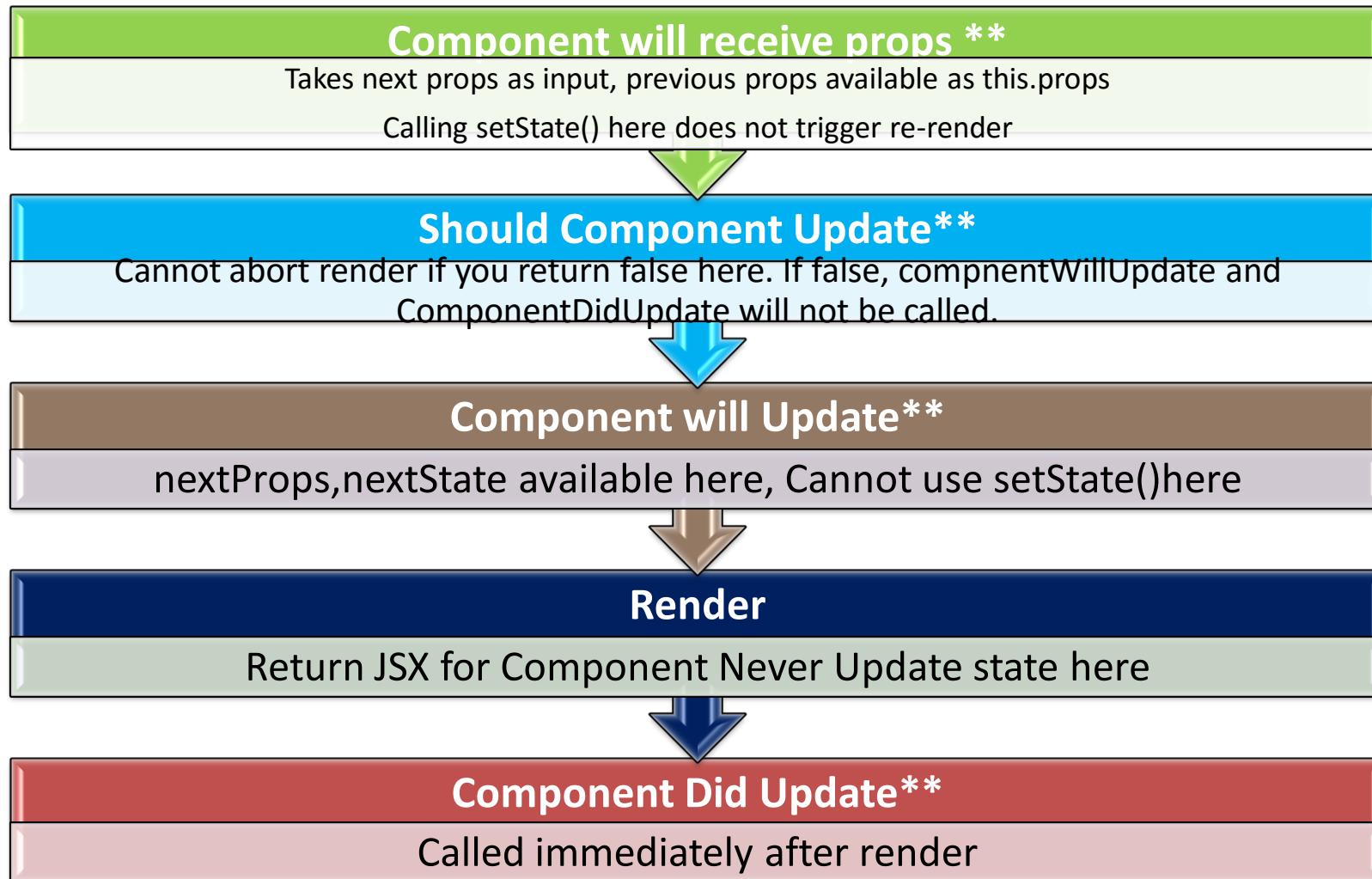
5. Render()

6. Children Initialization and Life Cycle kickoff

7. ComponentDidMount()

Component Life Cycle: Props Change

** - NOT CALLED FOR INITIAL RENDER



Validating props

- Existence and Data Type Validations

```
var Emp = React.createClass({
  propTypes: {
    empName: React.PropTypes.string.isRequired,
    empId: React.PropTypes.number.isRequired,
    designation: React.PropTypes.oneOf(['Software Engineer', 'Manager', 'Project Lead', 'Analyst'])
  }
})
```

- Custom Condition

```
var Threshold = React.createClass({
  propTypes: {
    calibrate: function(props, propName, componentName) {
      if(props[propName]>0.05){
        throw new Error(propName +
          "threshold confidence value should not be greater than 0.05");
      }
    }
  });
});
```

Update/Prop Changes

1. `componentWillReceiveProps`

2. `ShouldComponentUpdate()`

3. `ComponentWillUpdate()`

4. `Render()`

5. Children Life Cycle Methods

6. `ComponentDidUpdate()`

Props

- Props are supplied as attributes
- Props are meant to be passed as static functionality

```
var APP = React.createClass({  
    getDefaultProps: function() {  
        return ({ empName: 'James BOND',  
                 empId: 7 });  
    },  
    propTypes:{  
        empName:React.PropTypes.string.isRequired,  
        empId:React.PropTypes.number.isRequired  
    },  
    render:function(){  
        return(  
            <div>  
                <h1>Hello {this.props.empName}</h1>  
                <h2>Employee id: {this.props.empId}</h2>  
            </div>  
        )  
    }  
});  
ReactDOM.render(<APP empid={7} empName="James Bond"/>, document.getElementById('EmployeeApp'))
```

Setting the default values to the properties

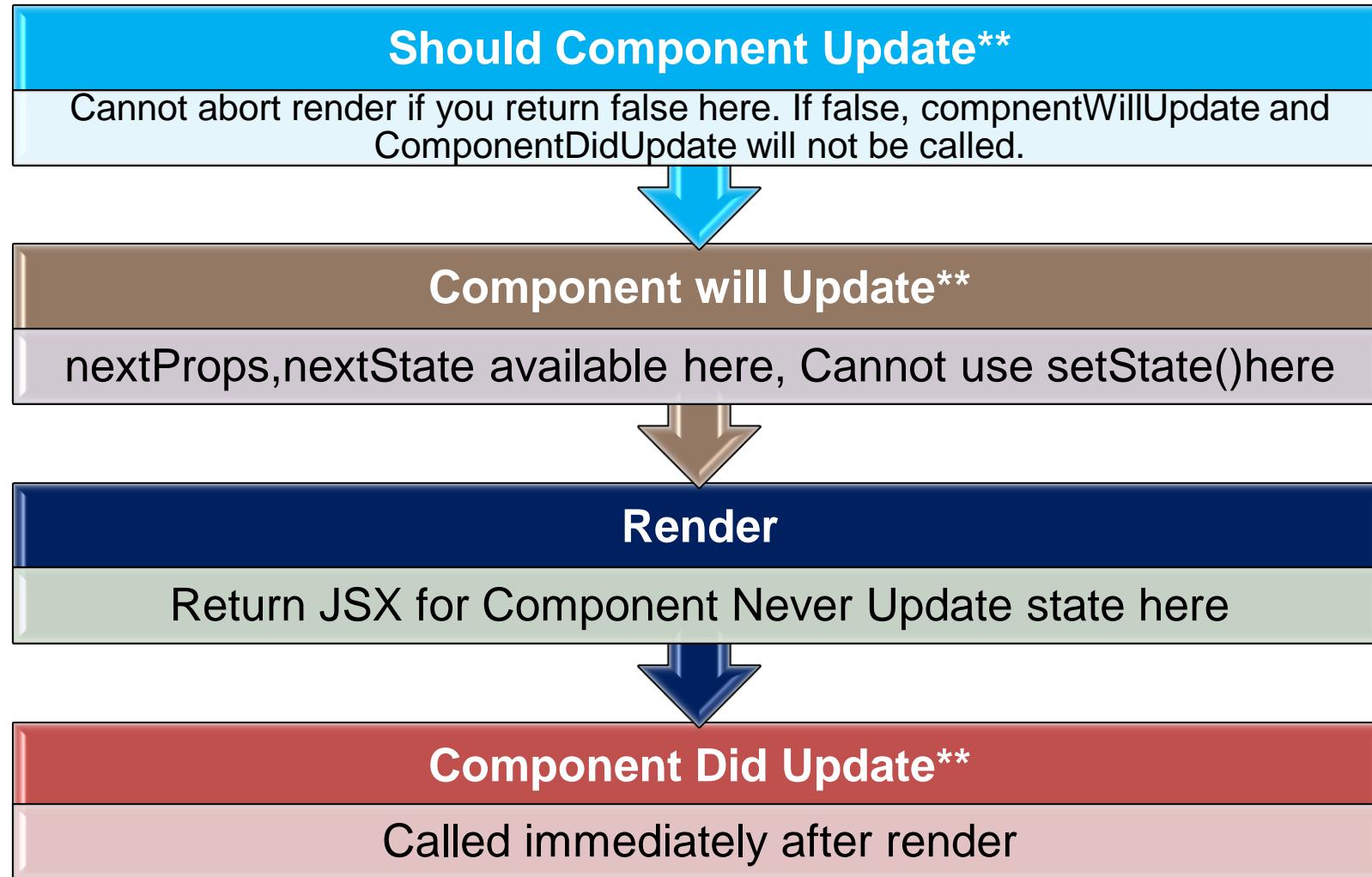
Defining PropTypes with Validation

Calling the PropTypes

Rendering it to DOM

Component Life Cycle: State Change

** - NOT CALLED FOR
INITIAL RENDER



State : Initial State

- States are used when the component is going to change.

```
<script type="text/babel">
  var APP = React.createClass({
    getInitialState:function(){
      return{empName:'James Bond',
      empId: '8'}
    },
    propTypes:{
      empName:React.PropTypes.string.isRequired,
      empId:React.PropTypes.number.isRequired
    },
    render:function(){
      return(
        <div>
          <h1>Hello {this.state.empName}</h1>
          <h2>Employee id: {this.state.empId}</h2>
        </div>
      )
    }
  );
  ReactDOM.render(<APP empid empName='Syed Awase' />, document.getElementById('EmployeeApp'))
</script>
```

Built-in method for all components

Validating properties

Reference to the state rather than props

Update/State Changes

1. Updating State

2. ShouldComponentUpdate()

3. ComponentWillUpdate()

4. *Render()*

5. Children Life Cycle Methods

6. ComponentDidUpdate()

State: Update State

```
<script type="text/babel">
  var APP = React.createClass({
    getInitialState:function(){
      return{empName:'James Bond',
            empId: '8'}
    },
    updateEmp:function(e){
      this.setState({
        empName:e.target.value
      })
    },
    propTypes:{
      empName:React.PropTypes.string.isRequired,
      empId:React.PropTypes.number.isRequired
    },
    render:function(){
      return(
        <div>
          <input type="text" onChange={this.updateEmp}/>
          <h1>Hello {this.state.empName}</h1>
          <h2>Employee id: {this.state.empId}</h2>
        </div>
      )
    }
  });
  ReactDOM.render(<APP empid empName='Syed Awase' />,
  | document.getElementById('EmployeeApp'))
</script>
```

Defining the initial state

Update method on some event

Prop Type validation and Required Check

Update method on change Event in the Input box

Binding

Nesting Components

```
<script type="text/babel">
  var voter ={
    title:'2019 Election!',
    content: 'Election Drama for the year 2019-worlds largest democracy',
    coverage: 'News Coverage by: TodayTamasha direction by Syed Awase',
    polparties:['aakiparty','rightwingrakshas','leftwingloafer','secularsants','regionalrogues']
  };
  var VoterAPP = React.createClass({
    render:function(){
      return(
        <div>
          <h1>Title: {this.props.data.title}</h1>
          <h3>{this.props.data.coverage}</h3>
          <p>{this.props.data.content}</p>
          <ol>
            <li><PoolParty context={this.props.data.polparties[0]}></PoolParty></li>
            <li><PoolParty context={this.props.data.polparties[1]}></PoolParty></li>
            <li><PoolParty context={this.props.data.polparties[2]}></PoolParty></li>
            <li><PoolParty context={this.props.data.polparties[3]}></PoolParty></li>
            <li><PoolParty context={this.props.data.polparties[4]}></PoolParty></li>
          </ol>
        </div>
      )
    }
  });
  var PoolParty=React.createClass({
    render:function(){
      return(
        <div>{this.props.context}</div>
      )
    }
  });
  ReactDOM.render(<VoterAPP data={voter}/>,
    document.getElementById('VoteComponent'))
</script>
```

JSON Object

Props

Inner Component Props

Outer Component

Inner Component

Render Component + passing json object

Nesting Components: map iterator

```
<script type="text/babel">
  var voter ={
    title:'2019 Election!',
    content: 'Election Drama for the year 2019-worlds largest democracy',
    coverage: 'News Coverage by: TodayTamasha direction by Syed Awase',
    polparties:['aakiparty','rightwingrakshas','leftwingloafer',
    'secularsants','regionalrogues','dalitdragons','minoritymachos']
  };
  var VoterAPP = React.createClass({
    render:function(){
      return(
        <div>
          <h1>Title: {this.props.data.title}</h1>
          <h3>{this.props.data.coverage}</h3>
          <p>{this.props.data.content}</p>
          <ol>
            {this.props.data.polparties.map(function(polparty){
              return <li><PoolParty context={polparty}></PoolParty></li>
            })}
          </ol>
        </div>
      )
    }
  });
  var PoolParty=React.createClass({
    render:function(){
      return(
        <div>{this.props.context}</div>
      )
    }
  });
  ReactDOM.render(<VoterAPP data={voter}/>,
  document.getElementById('VoteComponent'))
</script>
```

JSON Object

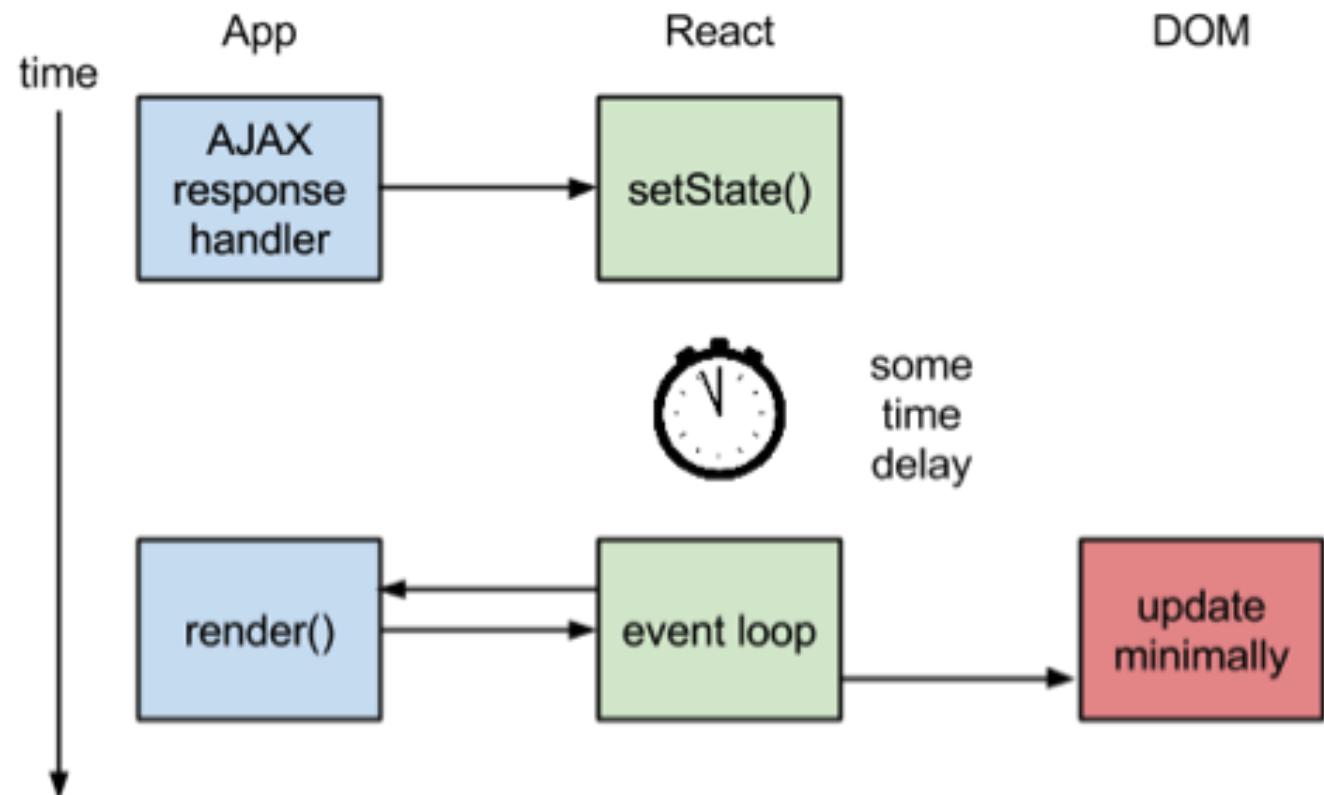
Map iterator to iterate over

Linking and populating the DOM element (rendering process)

ReactElement render()

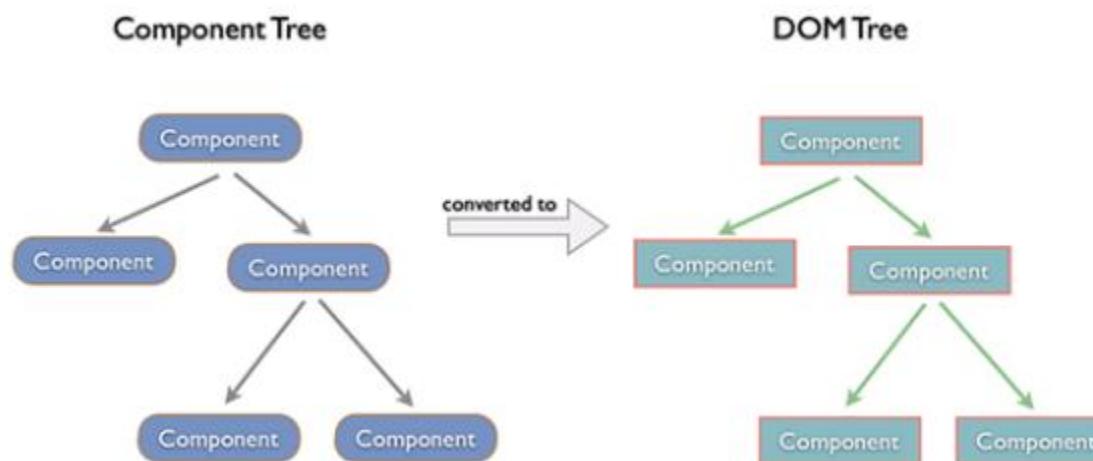
```
//Rendering  
ReactDOM.render(React.createElement(userApp), document.getElementById('app'));
```

- Rendering a component means linking it to a DOM element and populating that DOM element.



Structure of React Components

Components talk to each other via “**props**”



```
<great-grand-parent>
  <grand-parent>
    <parent>
      <child-one>
        </child-one>
        <child-two>
          </child-two>
        </parent>
      </grand-parent>
    </great-grand-parent>
```

Unmount

1. `ComponentWillUnmount()`

2. Children Life Cycle methods

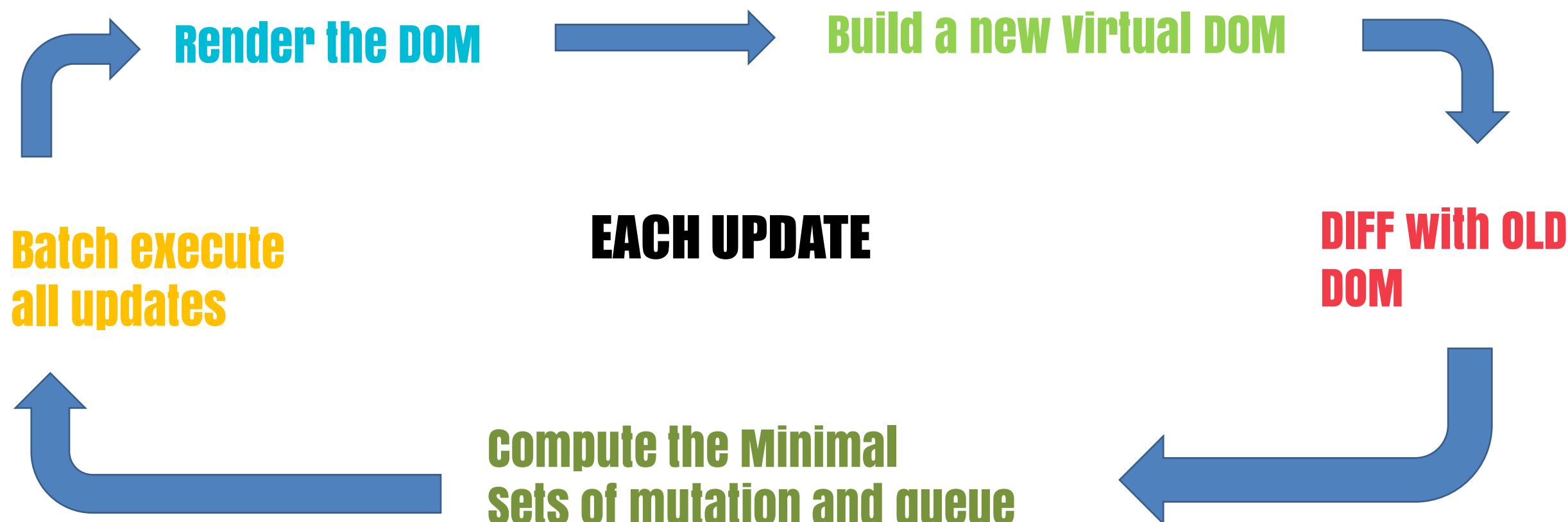
3. Instance destroyed for garbage collection

Component Life Cycle : Unmount

Component will unmount

- Invoked Immediately before component is unmounted.
- For cleanup, invalidating timers etc.

Virtual DOM



Mixins

- Easy way of transfer functionality to multiple components at one time

Component MIXINS

- Sometimes, you will have behavior that is required for many components.
- Mixins are handy bundles of reusable methods that you can apply to multiple component types
- Great for “glue” functions/boilerplate: service interaction, event listeners, dispatching actions, etc.
- Not available in ES6.
- Multiple mixins can class.

```
var RefreshMixin = {  
  componentDidUpdate:function(){  
    var node = $(this.getDOMNode());  
    node.slideUp();  
    node.slideDown();  
  }  
};  
var StockValue = React.createClass({  
  mixins:[RefreshMixin]  
});
```

Summary: Components

- Components are the building blocks of React application
- Components are composable
- Components map to equivalent DOM nodes that may have descendant nodes
- **createClass** defines a component (ES5)
- **ReactDOM.render** renders a component definition into the DOM
- **Props** provide the immutable data for a component
 - **propTypes** allow basic validation of props
 - Data type validation etc...
 - Custom validation
- **State** provide the mutable data for a component.
- **Mixins** allow reuse between components, without duplication of code
- **Components talk to each other via “props”.**

Controller Views

- Controller Views are top level react components , they control data flows for all of it's child components, by **setting props on child components.**
- **Controller views interact with FLUX stores**
- **Components are more portable**
- **Controller Views should be small and simple not deep component nesting**
- It is difficult to manage nested component's state, there by making it harder to reason.
- Easier Testing.
- **MovieListController**
 - Collects list of movies through ajax calls and passes to **movielistview**
- **MovieListView**
 - Prints movies as properties passed on from controller view.

JSX: JAVASCRIPT SYNTAX EXTENSION

A Javascript XML based extension that makes it easy to mix HTML with Javascript

An external domain specific language that is optimized for generating XML like documents.

JSX: XML-LIKE JAVASCRIPT

<https://babeljs.io/repl/>

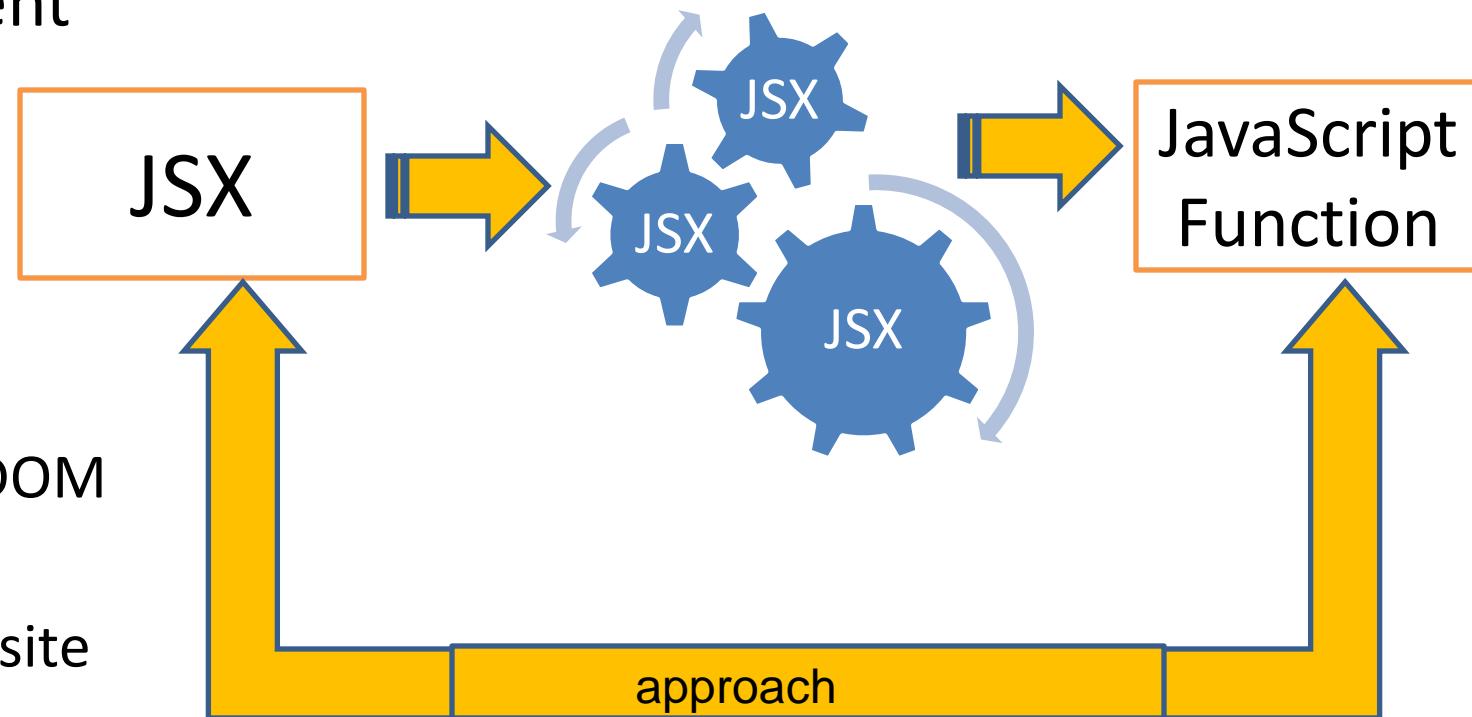
- It is an ECMASCIPT syntactic extension similar to HTML and XML
- JSX helps to write secure JavaScript with familiar HTML like syntax
- JSX transpilers help to compile JSX syntactic code into pure JavaScript for execution on browsers
- React JSX transpiler can be used to compile react JSX code into pure JavaScript for execution.
- Supports XML-like syntax inline in JavaScript
- Each element is transformed into a JavaScript function call

JSX: as language

- Strictly-typed OO programming language
- Syntax:
 - Class/function definition like Java
 - Function body is JavaScript
- Strict types lead to higher productivity/better quality than JavaScript
- Generated code runs faster than JS
 - By optimizing the generated code using type-info
 - JSX is designed so that there would be no overhead when compiled to JavaScript
- Generates source-map for debugging
- Compiler written in JS
- Easier to visualize the structure of the DOM
- Designers are more comfortable making changes
- Similar to MXML or XAML
- Cannot be minified/linted/processed in a browser.

React and JSX

- React and JSX are independent technologies, but JSX was primarily built with React in mind.
- JSX is used to
 - Construct instances of React DOM Components
 - Construct instances of Composite Components created with `React.createClass()`



Transforming JSX into JavaScript

- <https://babeljs.io/repl/>

The screenshot shows the Babel REPL interface. On the left, under 'JSX CODE', there is a code editor with the following JSX snippet:

```
1 var app = <App name="Syed Awase"/>
```

On the right, under 'Output JavaScript Code', there is another code editor with the generated JavaScript code:

```
1 "use strict";
2
3 var app = React.createElement(App, { name: "Syed Awase" });
```

Red arrows point from the 'JSX CODE' label to the JSX snippet and from the 'Output JavaScript Code' label to the generated JavaScript code.

JSX CODE

Output JavaScript Code



Transforming HTML to JSX

<http://magic.reactjs.net/htmltojsx.htm>

The screenshot shows a web browser window with the URL <http://magic.reactjs.net/htmltojsx.htm> in the address bar. The page title is "HTML to JSX Compiler". A checkbox labeled "Create class" is checked, with "Class name: NewComponent" entered. In the "Live HTML Editor" on the left, there is an HTML code block containing a form with a label and input field, and a placeholder text "Enter your HTML here". On the right, the corresponding JSX code is generated:

```
var NewComponent = React.createClass({
  render: function() {
    return (
      <div>
        /* Hello world */
        <div className="awesome" style={{border: '1px solid red'}}>
          <label htmlFor="name">Enter your name: </label>
          <input type="text" id="name" />
        </div>
        <p>Enter your HTML here</p>
      </div>
    );
  }
});
```

JSX

```
<hello/>  
var app=<Nav color="blue">  
    <Profile>click</Profile>  
    </Nav>
```

- Beyond the compilation step, JSX does not require any special tools
 - JSX neither provides nor requires a runtime library
 - JSX does not alter or add to the semantics of JavaScript.

Compiled JS

```
"use strict";  
  
React.createElement("hello", null);  
var app = React.createElement(  
    Nav,  
    { color: "blue" },  
    React.createElement(  
        Profile,  
        null,  
        "click"  
    )  
);
```

Just-in-time JSX Transformer

- Transform JSX to pure javascript immediately before execution
- Convenient for the developer
- Significant performance penalty
- other disadvantages related to not being valid javascript are
 - Cannot be minified
 - Cannot be linted
 - Cannot be debugged in the browser
 - Cannot be formatted with a javascript syntax highlighter.

```
<script src="JSXTransformer.js"></script>  
<script type="text/jsx">...</script>
```

Pre-process Transformer

- Transform JSX to pure JavaScript as a build step
- Much faster than the just-in-time transformer
- Works with javascript tools
- To install using node module react-tools
- `npm install -g react-tools`
- `jsx jsxdirectory/ compiledjs/`

JavaScript Expressions /JSX Attribute Expressions

- To use a javascript expression as an attribute value, wrap the expression in a pair of curly braces ({}) instead of quotes("")

```
var person = < Person name={window.isLoggedIn ? window.name: ''} />;  
<Hello now={new Date().toString()} />
```

```
'use strict';
```

```
var person = React.createElement(Person, { name: window.isLoggedIn ? window.name : '' });  
React.createElement(Hello, { now: new Date().toString() });
```

Child Expressions and Elements

- Any valid JSX expression
- A JSX component may have more than one child expression or element
- JavaScript expressions may be used to express children.

```
/**@jsx React.DOM */
//IIFE
(function(){
    var First = React.createClass({
        render:function(){
            return (<p>First Element</p>)
        }
    });
    var Second = React.createClass({
        render:function(){
            return (<p>Second Element</p>)
        }
    });
    var Parent = React.createClass({
        render: function(){
            return (
                <div className="parentName">
                    {this.props.children}
                </div>
            );
        }
    });
    ReactDOM.render(<Parent><First/><Second/> </Parent>, document.body);
})();
```

http://mdn.beonex.com/en/JavaScript/Reference/Reserved_Words.html

HTML Attributes -Style

- Style is a special react attribute that expects a javascript object with camelCase properties.

```
return(  
  <div style={{border:'1px solid'}}>  
    <table style={{border:'1px solid grey'}}>
```

```
var rowstyling ={  
  border: '1px solid blue',  
  background: 'yellow'  
};  
  
var users = userdata.map(function(user){  
  return (  
    <tr style={rowstyling}>  
      <td>{user.id}</td>  
      <td>{user.first_name}</td>  
      <td>{user.last_name}</td>  
      <td>{user.email}</td>  
      <td>{user.gender}</td>  
      <td>{user.ip_address}</td>  
    </tr>  
  );  
});
```



EVENTS

React Events

- React has its own event abstraction
 - Normalizes event behaviour across all supported browsers
 - Uniform event system for DOM events (button click, form submission) and component events(custom events raised by react components)
 - SyntheticEvent
 - Original event object is available via the nativeEvent property

DOM Events

- Events generated by Browser such as button click events, change events for text inputs and form submission events.
- DOM Events receive a react SyntheticEvent Object.

```
var confirmPassword = React.createClass({  
  render:function(){  
    return (<input type="text" onChange={this.handleChange}/>)  
  },  
  handleChange:function(e){  
    console.log(e.target.value);  
  }  
});  
  
ReactDOM.render(<confirmPassword/>, document.getElementById("react-app"));
```

Component Events

- Components can publish domain specific component events, which are typically at higher abstraction such as task added or priority change.

Touch Events

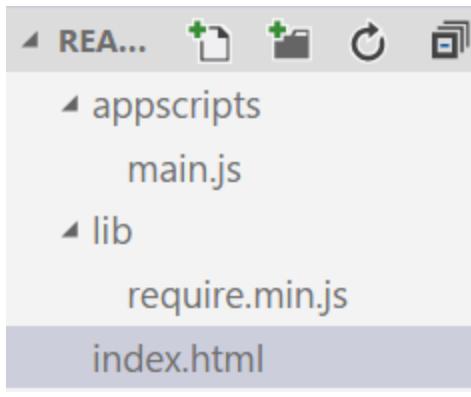
- Support for touch devices
- **React.initializeTouchEvents(true)**
 - Adds
 - onTouchStart
 - onTouchEnd
 - onTouchMove
 - onTouchCancel



React with RequireJS

Step I: Main.js

```
requirejs.config({
  paths: {
    'react': 'https://unpkg.com/react@15.3.2/dist/react',
    'react-dom': 'https://unpkg.com/react-dom@15.3.2/dist/react-dom'
  }
});
requirejs(['react', 'react-dom'], function(React, ReactDOM) {
  ReactDOM.render(
    React.createElement('h1', {}, 'Hello, Syed Awase !'),
    document.getElementById('react-app')
  );
});
```



Step II: Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="lib/require.min.js"></script>
</head>
<body>
<div class="container">
  <div id="react-app"></div>
</div>
<!--application specific scripts -->
<script src="appscripts/main.js"></script>
</body>
</html>
```

Naming Convention in React

- Different Naming Conventions
 - BotApp.react.js
 - BotApp.jsx
 - botApp.js
- Which extension/naming convention to choose?
 - .js
 - All IDEs will atleast highlight JS default applicaton
 - Require statements assume JS
 - May confuse IDE highlighting
 - .jsx
 - Some IDEs won't highlight any syntax
 - No default application
 - Must use.jsx extension to require
 - Clarifies content



SYED AWASE

Pattern for unidirectional data flow

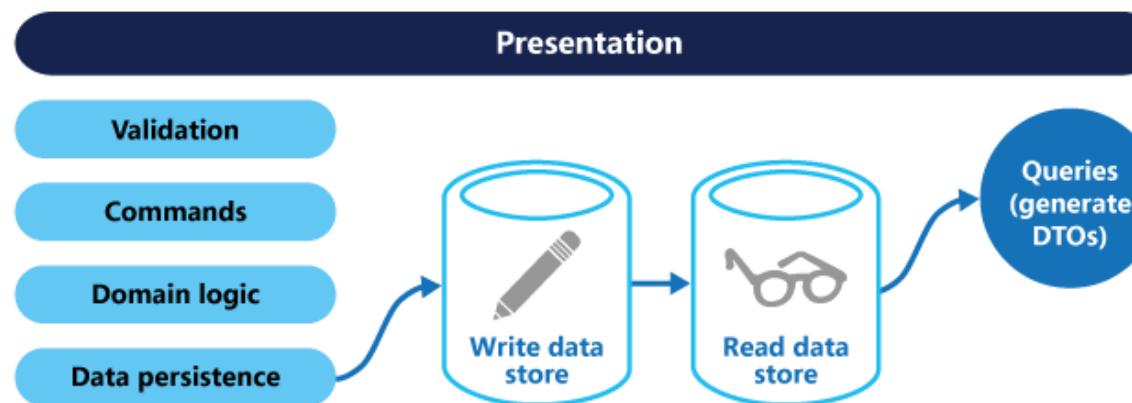
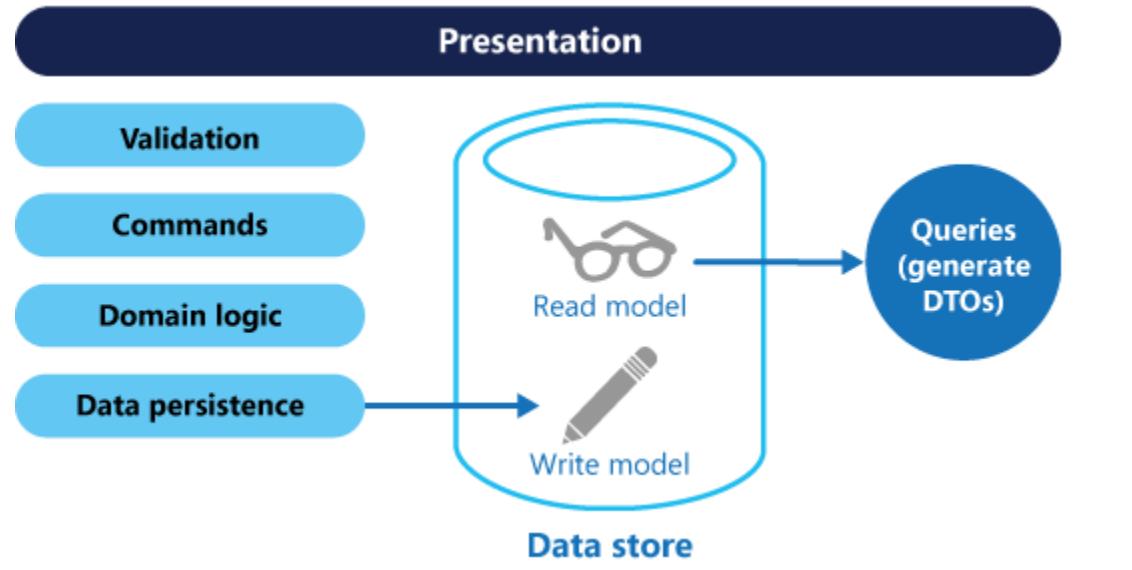


Data flows through out our application using flux

CQRS Pattern

- Command and Query Responsibility Segregation Pattern
 - Segregation operations that read data from operations that update data by using separate interfaces
 - Adopted for maximizing performance, scalability and security
 - Support evolution of the system over time through higher flexibility
 - Prevent update commands from causing merging conflicts at the domain level.
 - Flux is a similar to CQRS pattern.

CQRS Architecture



- Ideally suited for
 - Collaborative domains where multiple operations are performed in parallel on the same data
 - Task-based user interfaces, with complex domain models
 - Scenarios where performance for data reads must be fine-tuned separately from performance of data writes.
 - Scenarios where one team of developers can focus on the complex domain model that is part of the write model and another team can focus on read model and UI

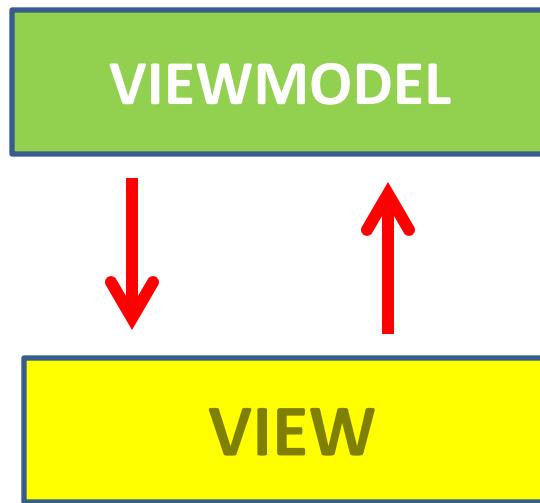
FLUX

Dictionary meaning of FLUX: the action or process of flowing or flowing out.

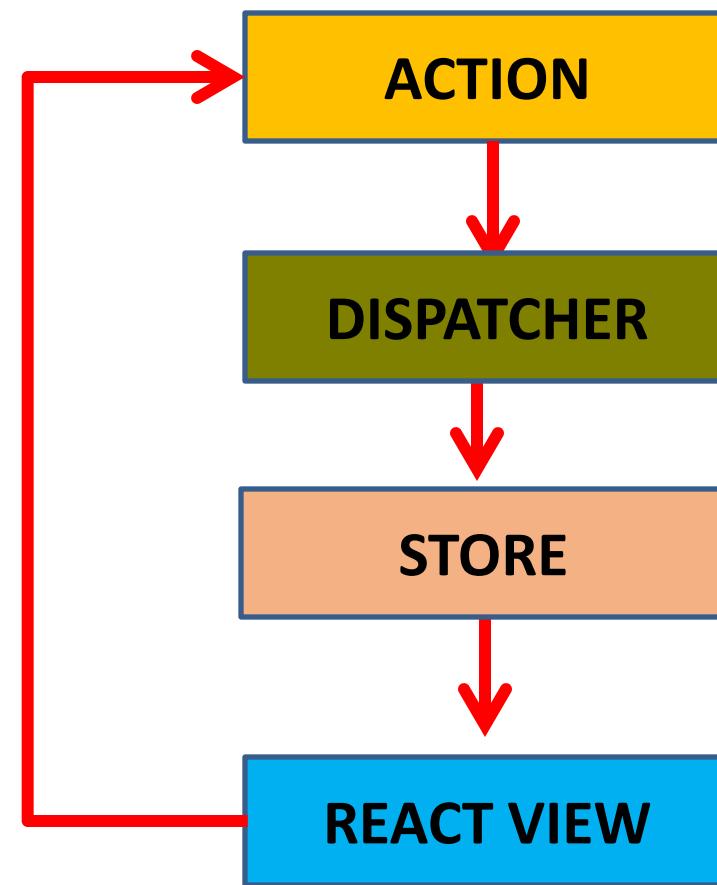
- More of a **pattern** rather than a formal framework
- A new kind of architecture that complements **React and the concept of Unidirectional data flow**
- **Glue for communicating between DUMB react components**
- **Fragmented /Evolving**
- **FLUX IS NOT ONLY FOR REACT**
- Implementations
 - Fluxxor (<http://fluxxor.com/>)
 - Marty.js (<http://martyjs.org/>)
 - Reflux (<https://github.com/spoike/refluxjs>)
 - Rx-flux (<https://github.com/fdecampredon/rx-flux>)
 - Omniscient (<http://omniscientjs.github.io/>)
 - Flummox (<http://acdlite.github.io/flummox>)
 - Fluxible (<http://fluxible.io/>)
 - NuclearJS (<https://optimizely.github.io/nuclear-js/>)
 - Delorean (<https://github.com/f/delorean>)
 - Alt (<http://alt.js.org/>)

Two-way Binding vs. Unidirectional flow

Two-way binding



Unidirectional

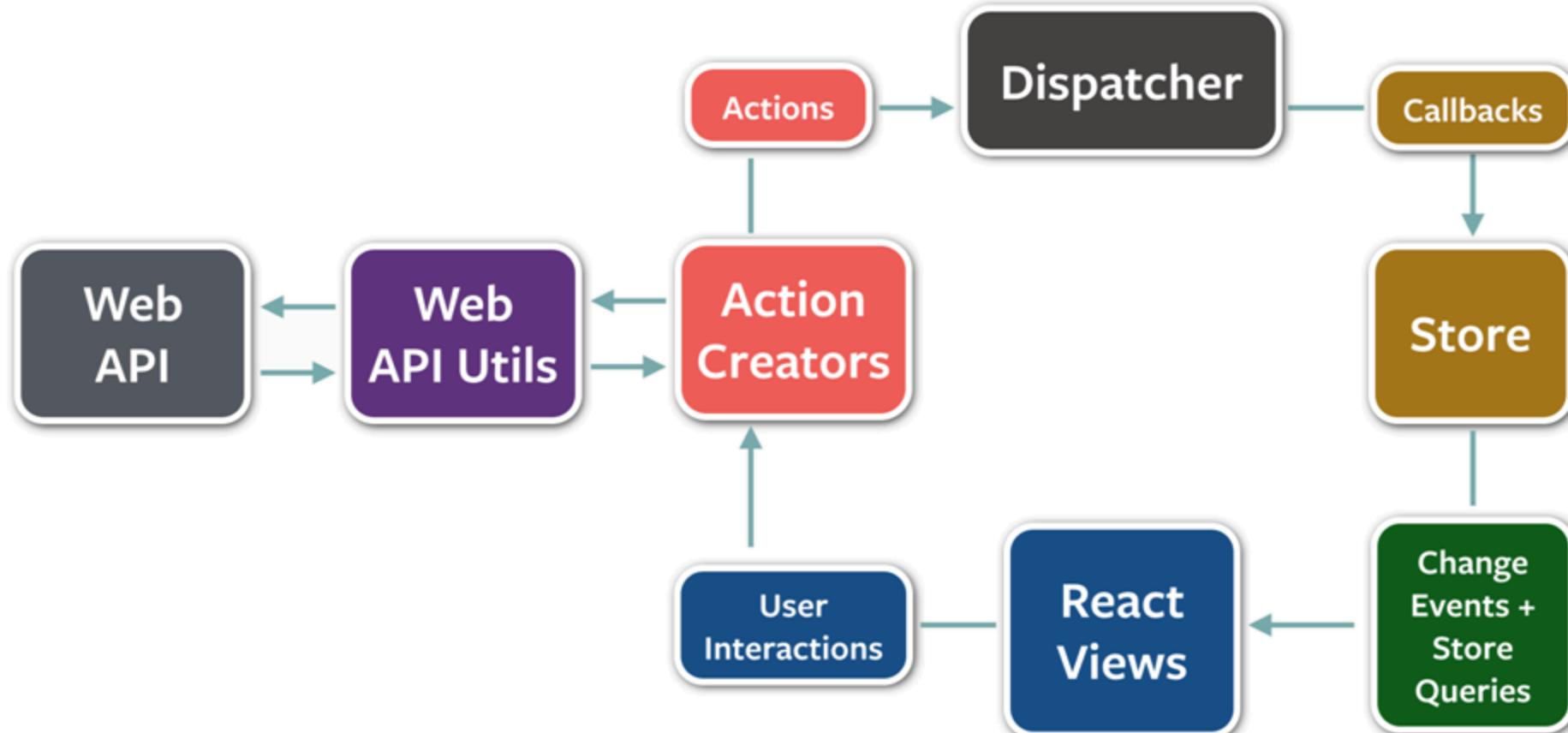


Flux : 3 CORE PARTS

- I. ACTIONS : user interaction that occur in a react component
- II. DISPATCHERS: singleton registry that notifies everyone who cares. A centralize list of call backs
- III. STORES: Hold applications data/state



FLUX



Source: <https://facebook.github.io/react/img/blog/flux-diagram.png>

FLUX vs MVC

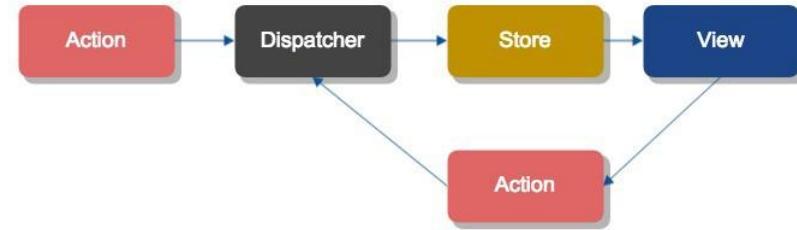
FLUX

- Flux pattern complex UIs no longer suffer from cascading updates; any given React component will be able to reconstruct its state based on the data provided by the store.
- Flux pattern also enforces data integrity by restricting direct access to the shared data.
- Redux, ALT are popular flux libraries.

MVC

- As the size of the application grows, MVC architectures encounter two main problems
 1. Poorly defined data flow: the cascading updates which occur across views often lead to a tangled web of events which is difficult to debug.
 2. Lack of data integrity: Model data can be mutated from anywhere, yielding unpredictable results across the UI

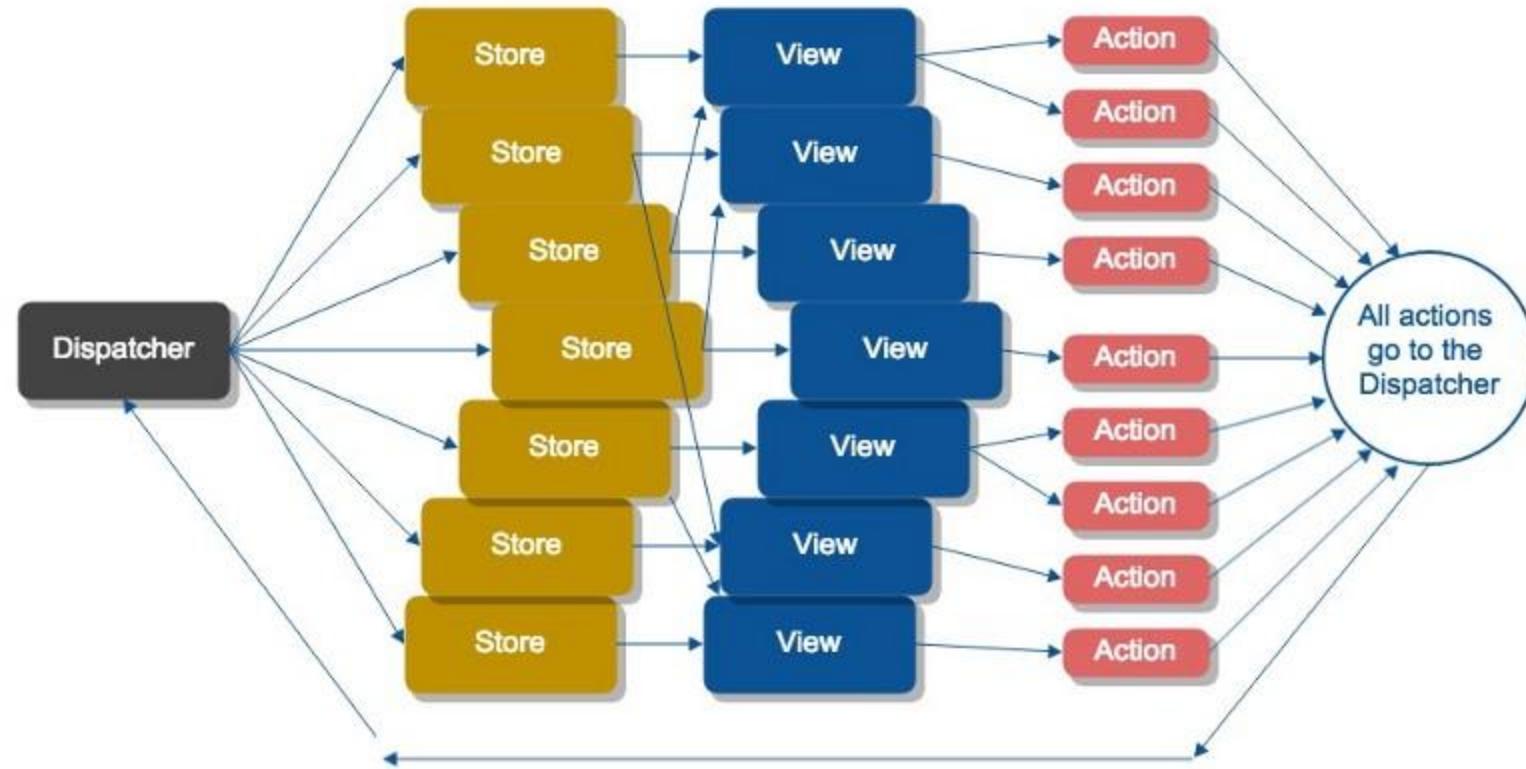
ADSV



- Actions : Users and Services DO things
- Dispatcher : NOTIFY everyone WHAT HAPPENED
- Stores – KEEP TRACK of the DATA
- Views- React Components



Complex Flux Data Flow



Source: <https://brigade.engineering/what-is-the-flux-application-architecture-b57ebca85b9e#.6n54yg7yp>

FLUX: DISPATCHER

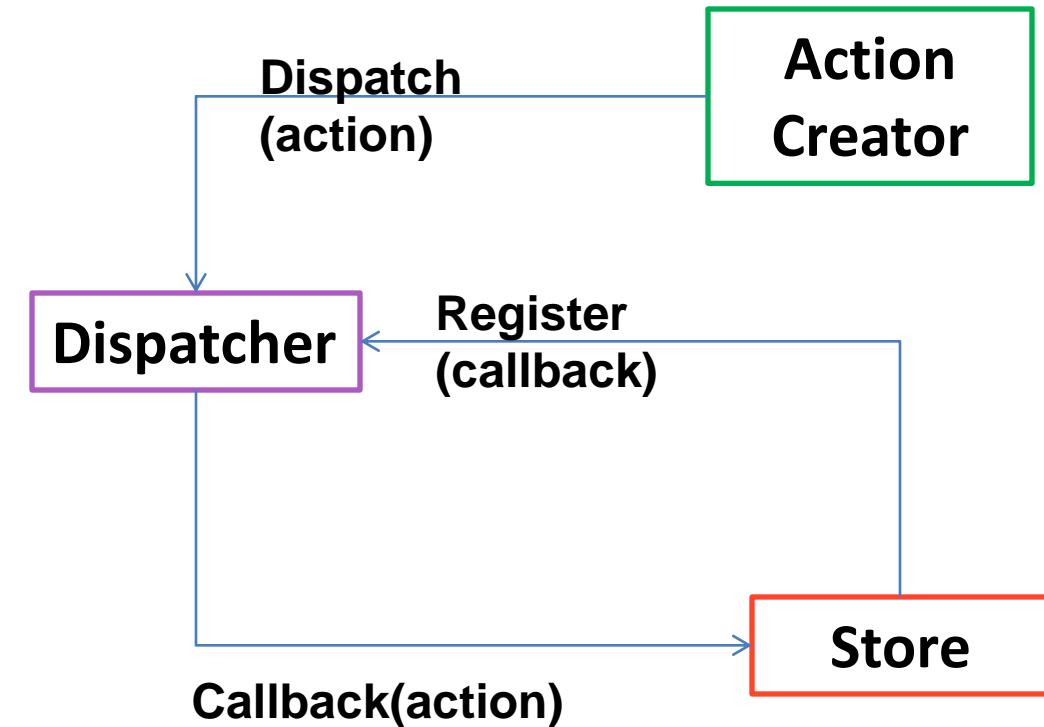
- All data flows through dispatcher as a **central hub**.
- **Dispatcher is a SINGLETON, so there is ONLY ONE dispatcher for application**
- **Dispatcher dispatches ACTIONS**
- **It holds a list of callbacks**
- **Broadcasts payload to all registered callbacks**
- **Predictable application data flow.**
- **Dispatcher sends actions to stores.**

Constants

- With flux, it would be helpful to create a **constants file**
 - To keep things organized by defining constants used throughout the application
- Provides high level view of what the application actually does.

FLUX: DISPATCHER

- A central hub that manages all data flow
- A registry of call backs
 - Stores register themselves by providing a callback
- When a new action arrives, all callbacks are invoked.
- Only one instance per app
- All Subscribers receive all events it dispatches
- Only clever trick: `waitFor()`

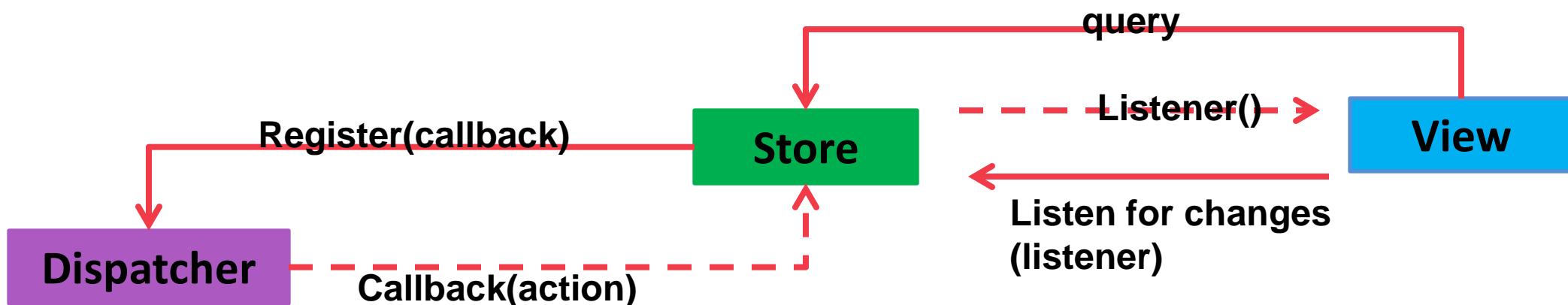


FLUX:STORES

- Holds app state, logic, data retrieval methods
- A store is not a model – it **contains models**
- An application can have single or multiple stores
- Related data can be grouped together
- Stores get updated because they have **callback which are registered with dispatcher**.
- Only stores are allowed to register dispatcher callbacks
- They use Node's EventEmitter.
- They have no direct setter methods instead they only accept updates via callbacks that are registered with dispatcher

FLUX:STORES

- Contains application **state** and **logic ,data retrieval**
 - Similar to repositories
- Manage a whole domain, rather than a single group of objects
- Register themselves to the dispatcher with a callback
 - The callback receives one argument – the action.



Structure of a Store

- Every store has these common traits (aka interface)
 - Extend EventEmitter
 - addChangeListener and removeChangeListener
 - emitChange
- Stores can declaratively wait for other stores to finish updating and then update themselves accordingly
 - **WAITFOR API** : allows users to specify the order stores are updated when a specific action has occurred.

FLUX:STORES

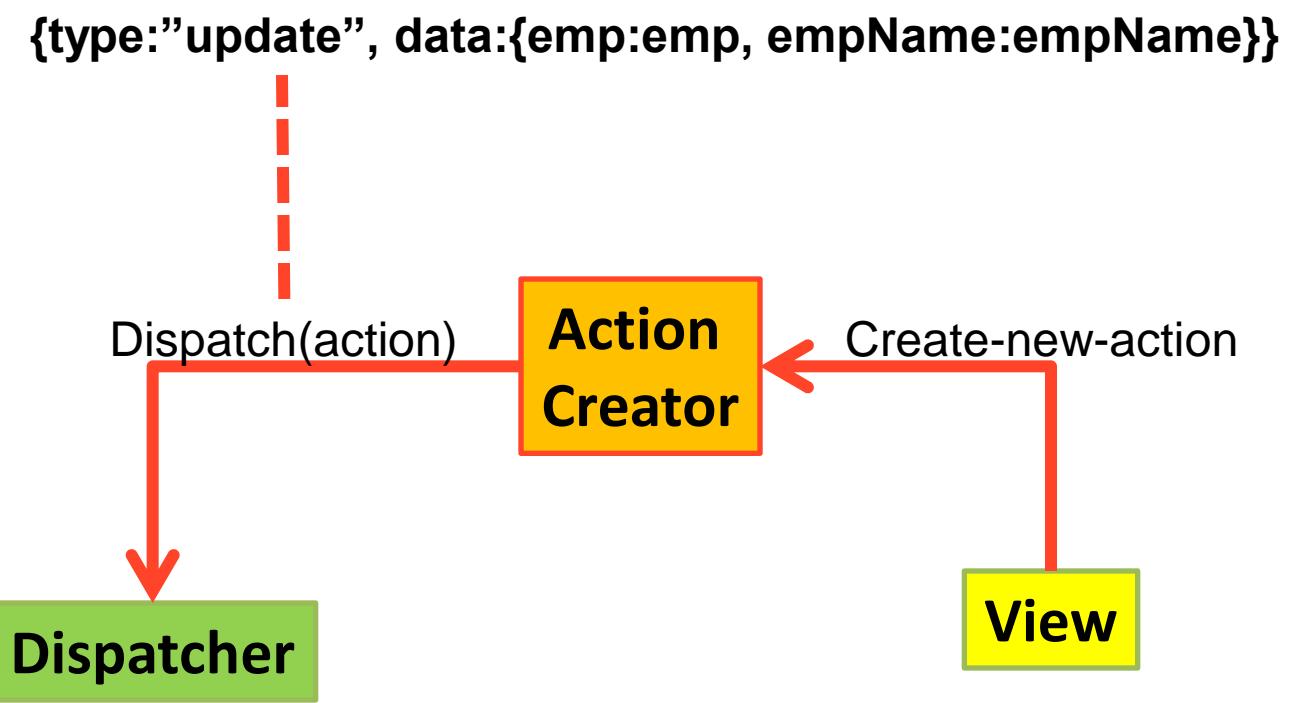
- How many? Approx.one store per entity type/domain
- Or, to model a relationship between entities.
- Simply accepts incoming changes, fires change events, lets the components worry about what happens next.
- Smart about data, dumb about interface and interaction
- Can also store app-wide user/interface state
- Flux is completely unopinionated about modes: POJOs seems most common
- Consider Immutable data structures

FLUX: ACTIONS

- Actions encapsulate specific events within the application
- Dispatcher exposes a method that allows us to trigger a dispatch to the stores and include a payload of data which is called an action
- Action creators are dispatcher helper methods, they describe all the actions that are possible in the application
- Actions are also triggered from the server.

FLUX:ACTION AND ACTION CREATORS

- Actions define an activity to be performed
- Typically have type and payload
- Stores use the action type to determine further processing
- Action creators are helper methods to create actions
 - They call also the dispatcher with the newly created action

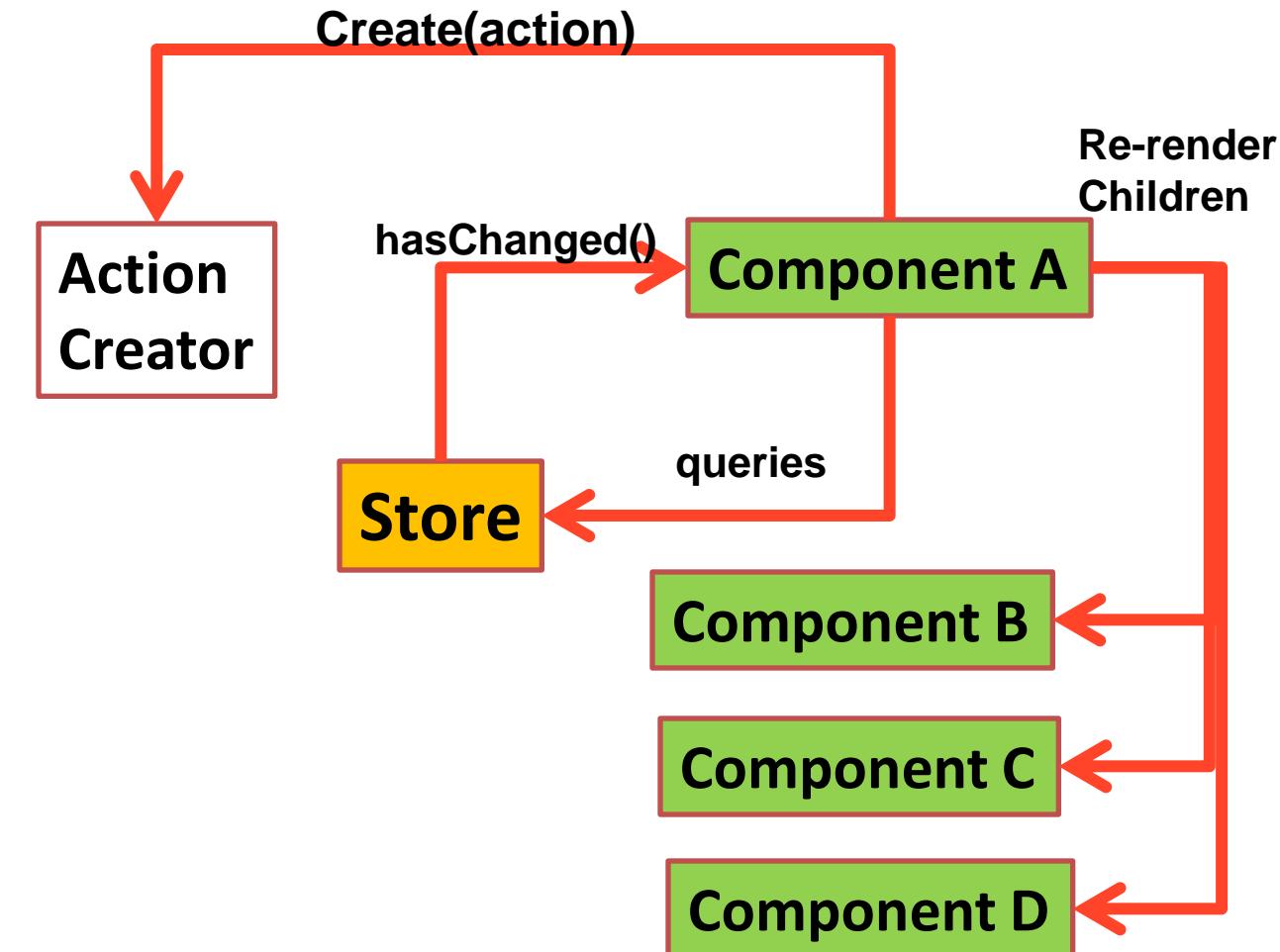


Controller Views

- They control the data flows down to the child components, they are top level components.
- They are components that interact with the stores.
- Controller Views hold data in state and sends data to children as props.
- It is possible to nest controller views, BUT NOT RECOMMENDED, doing us can cause multiple data update which can hamper performance.
- Recommendation, a single top level component per page.

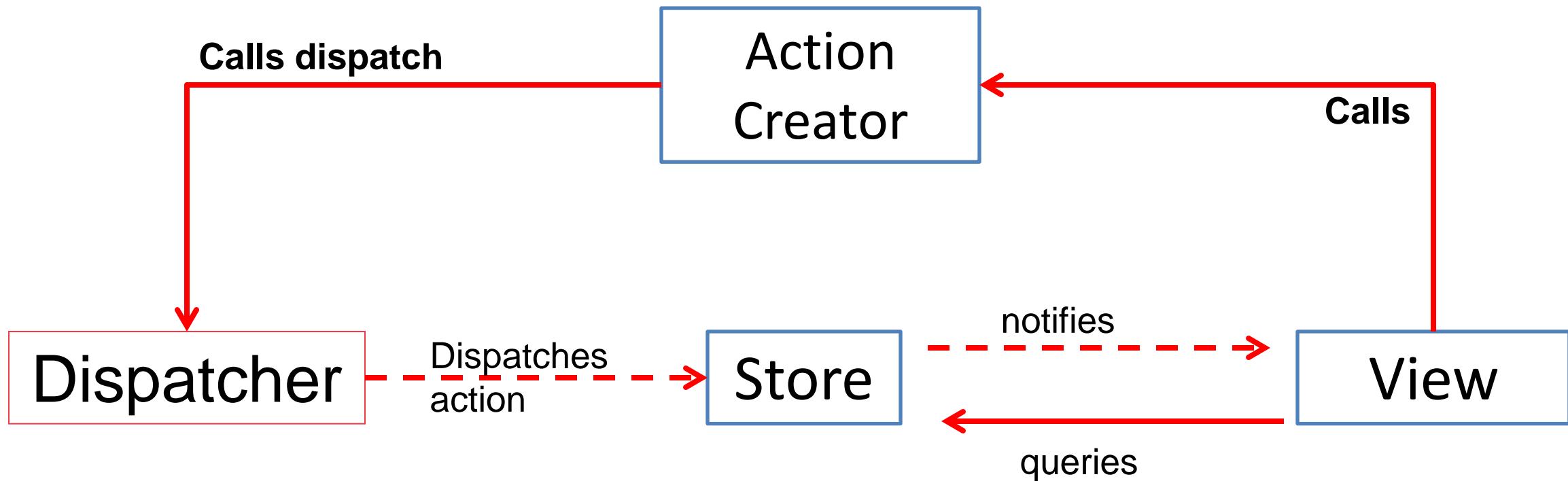
VIEWS

- Are react components
- Subscribe to stores and listen for changes
- Ensure one way of data flow
 - Only one view in a hierarchy listens for changes in the store
 - Updates all other views underneath in the hierarchy.



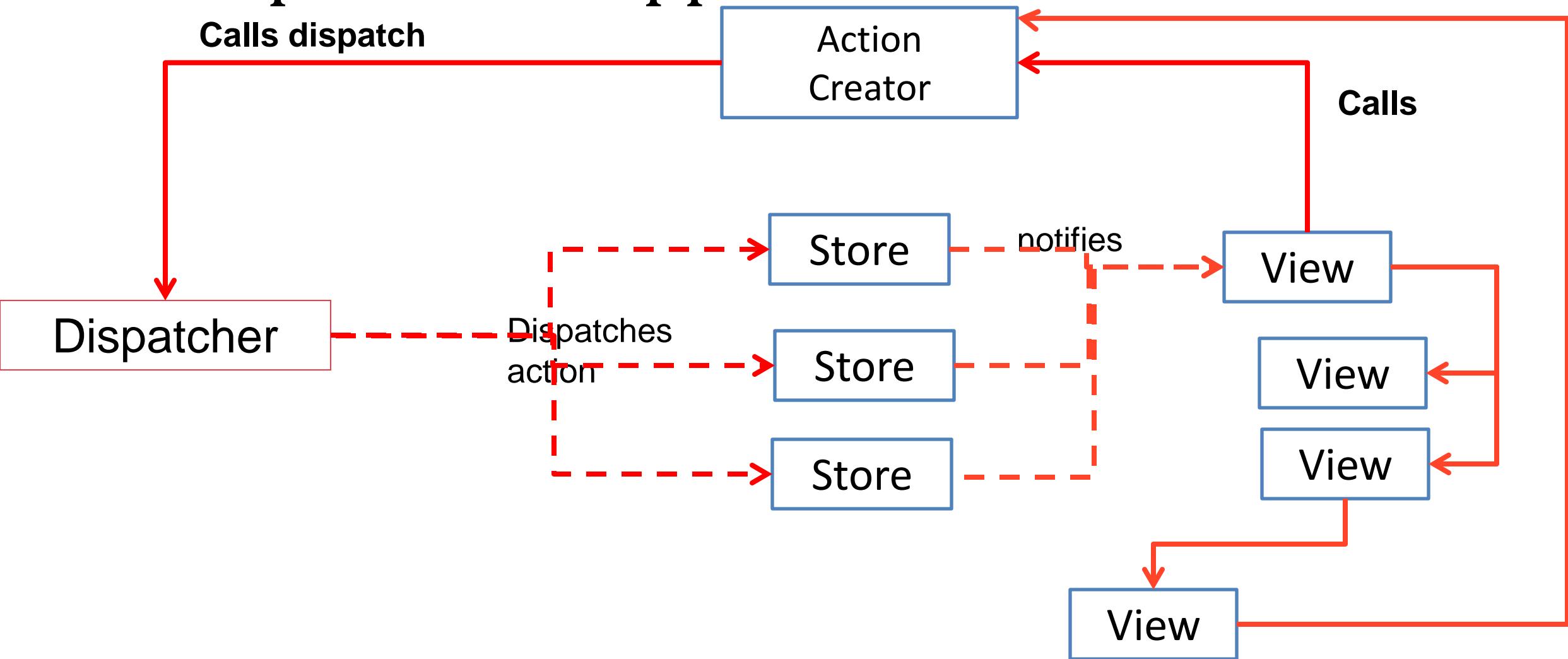
FLUX

Objects are highly decoupled (principle of least knowledge)



Empowers unidirectional flow

Enterprise Flux Application



FLUX API

Method	Description
Register(function callback)	Dispatcher run me when actions happen and call a registered store
Unregister(string id)	Dispatcher, unregister action to deregister a store
waitFor(array<string> ids)	Update this store first – store
Dispatch(object payload)	Dispatcher, tells the stores about this action- Action
isDispatching()	I am busy dispatching callbacks right now

Is Flux similar to Publish-Subscribe Model?

- To some extent BUT Differs in
 - Every payload is dispatched to all registered callbacks.
 - Callbacks can wait for other callbacks

FLUX PRACTICES

- Data Should not be changed in other places, but only in Store.
- Use **shouldComponentUpdate** wisely
 - Render correctly
 - Improve performance
- Keep ReactJS components thin and focused
- All behaviours are event-driven
- All logic is in the stores
- The only way to change something in the stores (and in the UI) is by publishing an event(no setters)

FLUX CHALLENGES

- A lot of boilerplate code
- Code is specific to the Flux framework used
- Store logic has dependency on its state.
- A change in the store would create new actions
- Migrating existing resource can be a big task
- Unit testing can be difficult without good structure.



SYED AWASE

AXIOS

H T T P R e q u e s t s / S e r v i c e s

Axios.get

```
var APP = React.createClass({
  getInitialState: function() {
    return {
      products: []
    }
  },
  componentDidMount:function(){
    var _this = this;
    this.serverRequest= axios.get("../data/ebaydata.json")
      .then(function(result){
        _this.setState({
          products:result.data
        });
      })
      .catch(function(error){
        console.log(error);
      })
  },
  ...
});
```

Defining the state to retrieve an array of objects

Axios.get http service to fetch data from web api

Displaying and Binding

```
render:function(){
  return(
    <div>
      <h1>Ebay Products List</h1>
      {this.state.products.map(function(product) {
        return (
          <div>
            <div>Product Name:{product.name}</div>
            <div>Product Image: <img src={product.image} alt="" /></div>
            <div>Product Description:{product.description}</div>
            <div>Product category:{product.category}</div>
            <div>Product price:{product.price}</div>
            <div>Product quantity:{product.quantity}</div>
            <div>Product shipping:{product.shipping}</div>
            <div>Product location:{product.location}</div>
          </div>
        );
      ))}
    </div>
  )
};

ReactDOM.render(<APP/>, document.getElementById('react-app'))
```

Displaying the data

Binding it to the view

Dependency Includes

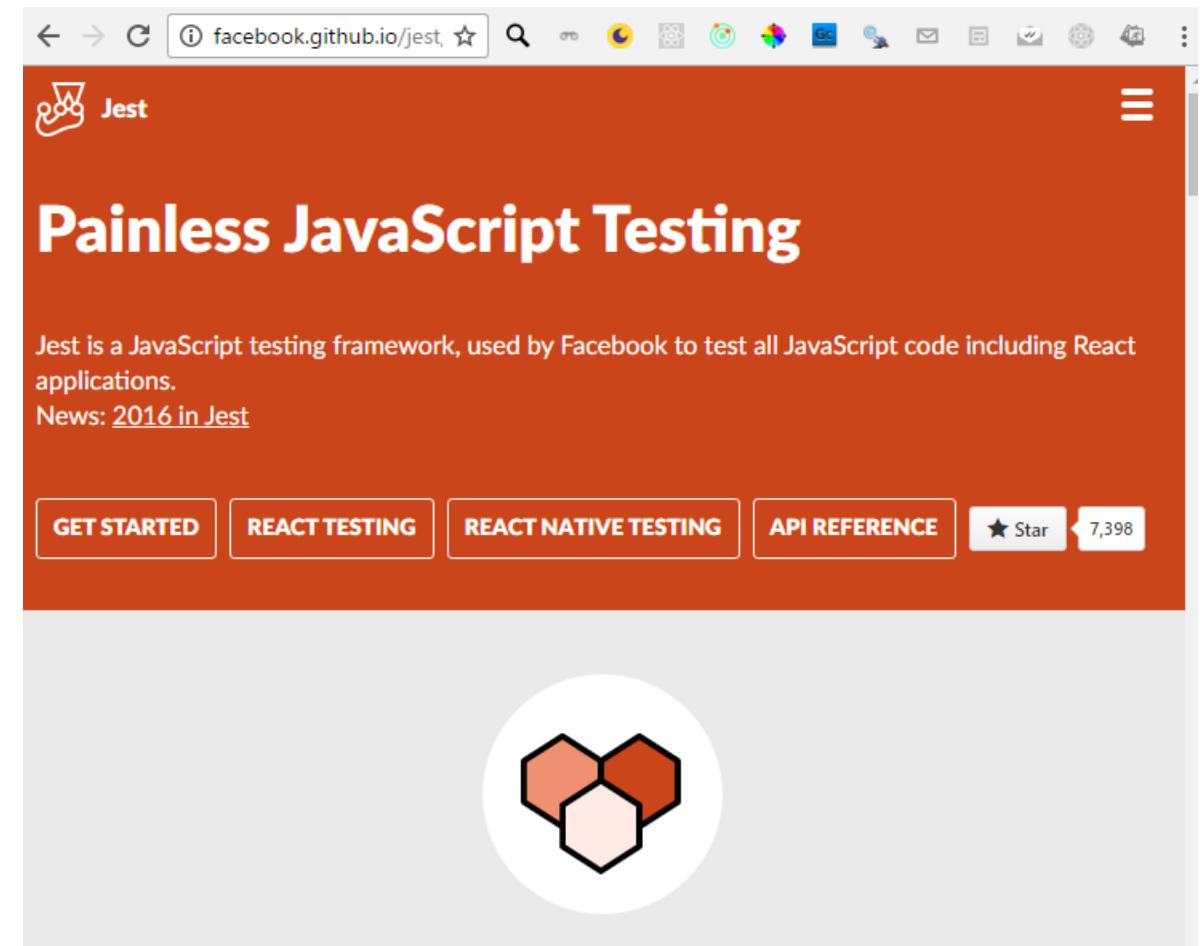
```
<!--react core libraries-->
<script src="lib/react-15.3.1/build/react.js"></script>
<script src="lib/react-15.3.1/build/react-dom.js"></script>
<script src="https://npmcdn.com/babel-core@5.8.38/browser.min.js"></script>
<!--end of react core libraries-->
<!--axios -->
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<!--end of axios -->
```



SYED AWASE

JEST:Testing

<http://facebook.github.io/jest/>



The screenshot shows the Jest homepage in a web browser. The URL in the address bar is facebook.github.io/jest/. The page has a red header with the title "Painless JavaScript Testing". Below the title, it says "Jest is a JavaScript testing framework, used by Facebook to test all JavaScript code including React applications." There is a link "News: [2016 in Jest](#)". At the bottom of the header, there are four buttons: "GET STARTED", "REACT TESTING", "REACT NATIVE TESTING", and "API REFERENCE". To the right of these buttons is a star icon with the number "7,398". Below the header is a large white circle containing three overlapping hexagons in orange, red, and light pink.

Testing Frameworks



Nightwatch.js

<http://nightwatchjs.org/>



Protractor

end to end testing for AngularJS



<https://karma-runner.github.io/1.0/index.html>

gremlins.js

<https://github.com/marmelab/gremlins.js/>



Chai Assertion Library

<http://chaijs.com/>



<https://github.com/testdouble/testdouble.js/>



<https://wallabyjs.com/>



<http://mochajs.org/>



<https://github.com/getify/LABjs>

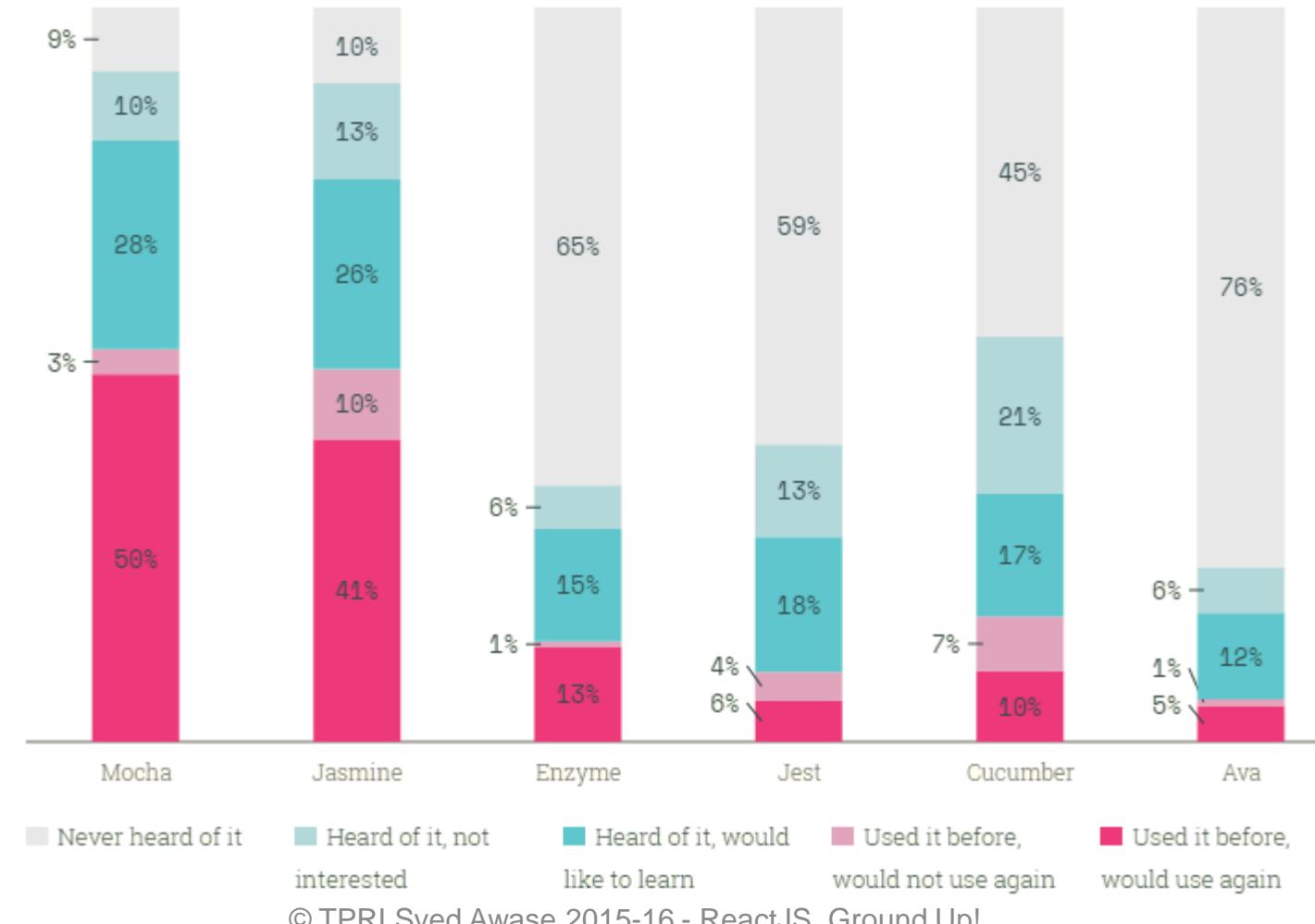


<https://chimp.readme.io/>

<http://www.nightmarejs.org/>

Testing Frameworks

Source: <http://stateofjs.com/2016/testing/>



JEST

- Built on top of the Jasmine Test Framework, using familiar `expect(value).toBe(other)` assertions
- Automatically finds tests to execute in your repo
- Automatically mocks dependencies for you when running your tests
- Allows you to test asynchronous code synchronously
- Runs your tests with a fake DOM implementation (via JSDOM) so that your tests can run on the command line

JEST?

- React's Virtual DOM : facilitates easy testing of your application
- A framework for testing component-based applications
- Cohesive framework to simplify testing and reduce errors.
- Istabbul(built-in to Jest) for coverage to enhance tests.
- Works with ES6 and JSX
- Most of test logic uses Jasmine interface, Jest API allows users to create, modify and analyze mocks.
- Node as the Foundation.

MOCKING

- It is the process of replacing a module with a generic and mostly empty object, called a “**mock**”, whose functionality is often defined inside the test.
- Mocks have more features than stubs.
- Most testing frameworks require that you manually specify mocks, while some do not support mocking at all.
- Jest automatically mocks all the modules in an application by default, and user needs to define **what not to mock**.
 - “**dontMock**” is an important way to define what not to mock.

ADVANTAGES OF MOCKING

- Isolates your code from its dependencies
- Write test specification with no knowledge of dependencies
- Decrease in false positives
- Less code refactoring to adapt to the changes made to dependencies.
- Test Specification are less fragile.

Criterion for Successful Unit Testing

- One assertion per test
- Tests are simple and understandable at a glance.
- Test output produces clear documentation
- Tests are treated like code: refactored, optimized and improved
- If there are **n** variables, then we need '**n!**' **test cases** + '**n**' **null cases**.