

2015

ES

Welcome

THE FUTURE
IS NOW

ECMAScript 2015 and beyond

ES 2015 Features

const & let

for...of & iterators

Template Strings

Arrow Functions

Default Parameters

Rest Parameters

Enhanced Object Literals

Destructuring

Spread

Classes

Modules

Generators

Promises

Set, Map, WeakSet, WeakMap

Proxies

Symbols

Reflect API

Language Feature (syntax)

Built-In Objects (runtime)

(list is not complete ...)

<https://babeljs.io/docs/learn-es2015/>

<https://github.com/lukehoban/es6features>

Classes

```
class Counter {  
  
    constructor(){  
        this.count = 0;  
    }  
  
    increase(){  
        this.count++;  
    }  
}  
  
let counter = new Counter();  
counter.increase()  
console.log(counter.count)
```

Classes are syntactic sugar on top of constructor functions and prototypal inheritance.

ES2015 introduced classes.
Every modern browser supports this today.

Decorators

```
@Component({
  selector: 'app-counter',
  template: 'counter.component.html'
})
class CounterComponent {
  constructor(){
    this.count = 0;
  }
  increase(){
    this.count++;
  }
}
```

Decorators can be used to attach metadata to classes.

Frameworks can then use this metadata.

The specification for decorators is not yet finished.

Browsers do not yet support decorators.

To use decorators today, they have to be compiled to JavaScript.

Modules

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-counter',
  template: 'counter.component.html'
})
export class CounterComponent {
  constructor(){
    this.count = 0;
  }
  increase(){
    this.count++;
  }
}
```

Modules have their own scope.
Sharing constructs is explicit with import & export.

The specification for modules is finished, but only most recent browsers support modules.

To use modules today a module bundler/loader is needed.

Class Fields

```
class Counter {  
  
    count = 0;  
  
    increase(){  
        this.count++;  
    }  
}  
  
let counter = new Counter();  
counter.increase()  
console.log(counter.count)
```

The specification class fields is not yet finished.

Browsers do not yet supports class fields.

To use class fields today, they have to be compiled to JavaScript.

Static Typing & Access Modifiers

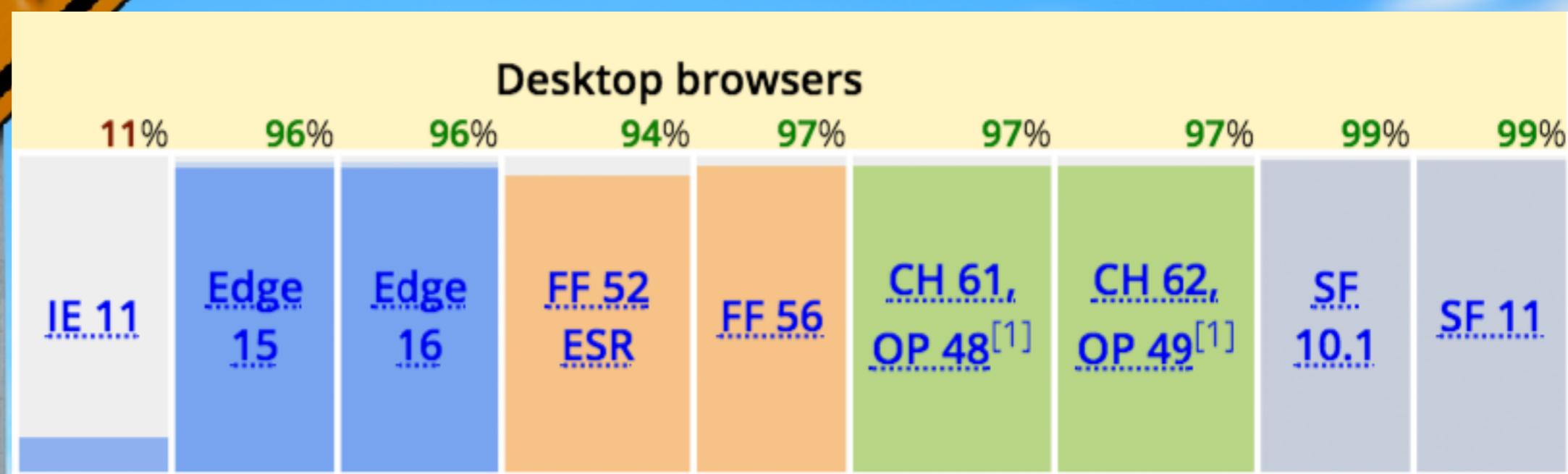
```
class Counter {  
  
    private count = 0;  
  
    increase(amount: number){  
        this.count += amount;  
    }  
    getValue(): number {  
        return this.count;  
    }  
}  
  
let counter = new Counter();  
counter.increase(2)  
console.log(counter.getValue())
```

Static typing and access modifiers are not part of JavaScript. The compiler removes this information at build time.



**REALITY
CHECK
AHEAD**

ES2015 browser support:



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



ES2015 Modules

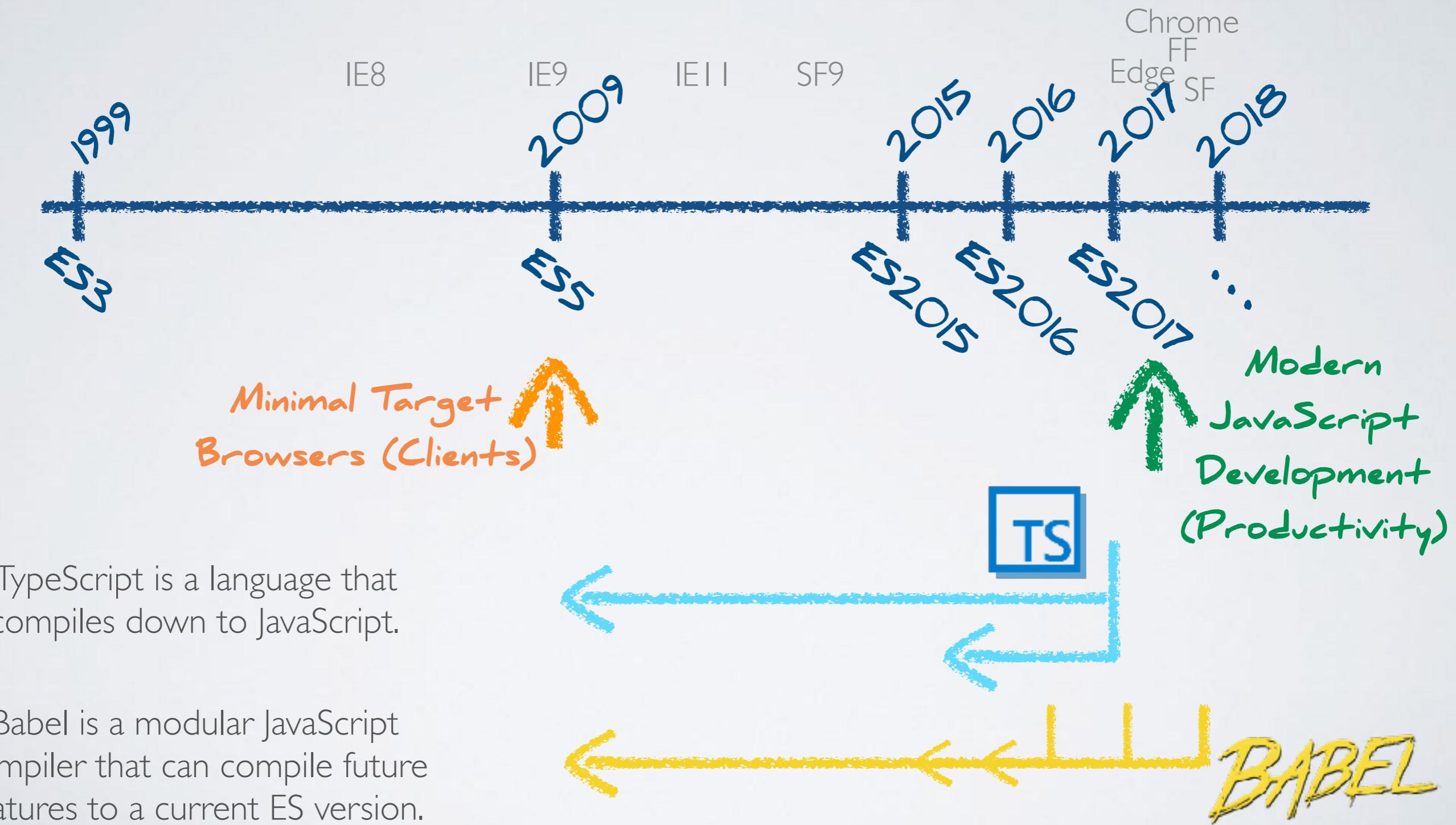
- New JavaScript syntax for `import` and `export` is part of the ES2015 spec.
- Module loading is not part of the ES2015 spec. There is a separate WhatWG loader spec.
- Native browser support for modules is still lacking
 - Official support in Safari 10.1, Chrome 61
 - Preview in Firefox 54, Edge 15

<http://caniuse.com/#feat=es6-module>

<https://medium.com/dev-channel/es6-modules-in-chrome-canary-m60-ba588dfb8ab7>

Addressing the Feature Gap

JavaScript has evolved rapidly in the past few years.



Transpiler

A transpiler is a source-code to source-code compiler.



TypeScript language features and most ES2015+ features can be written in plain ES5.

```
class Person {  
    private name: string;  
    constructor(name: string){  
        this.name = name;  
    }  
}  
  
const pers: Person = new  
Person('John');
```



```
var Person = (function () {  
    function Person(name) {  
        this.name = name;  
    }  
    return Person;  
}());  
var pers = new Person();
```



TypeScript is a superset of JavaScript

The goal of TypeScript



on top of ES5

new (future)
language features

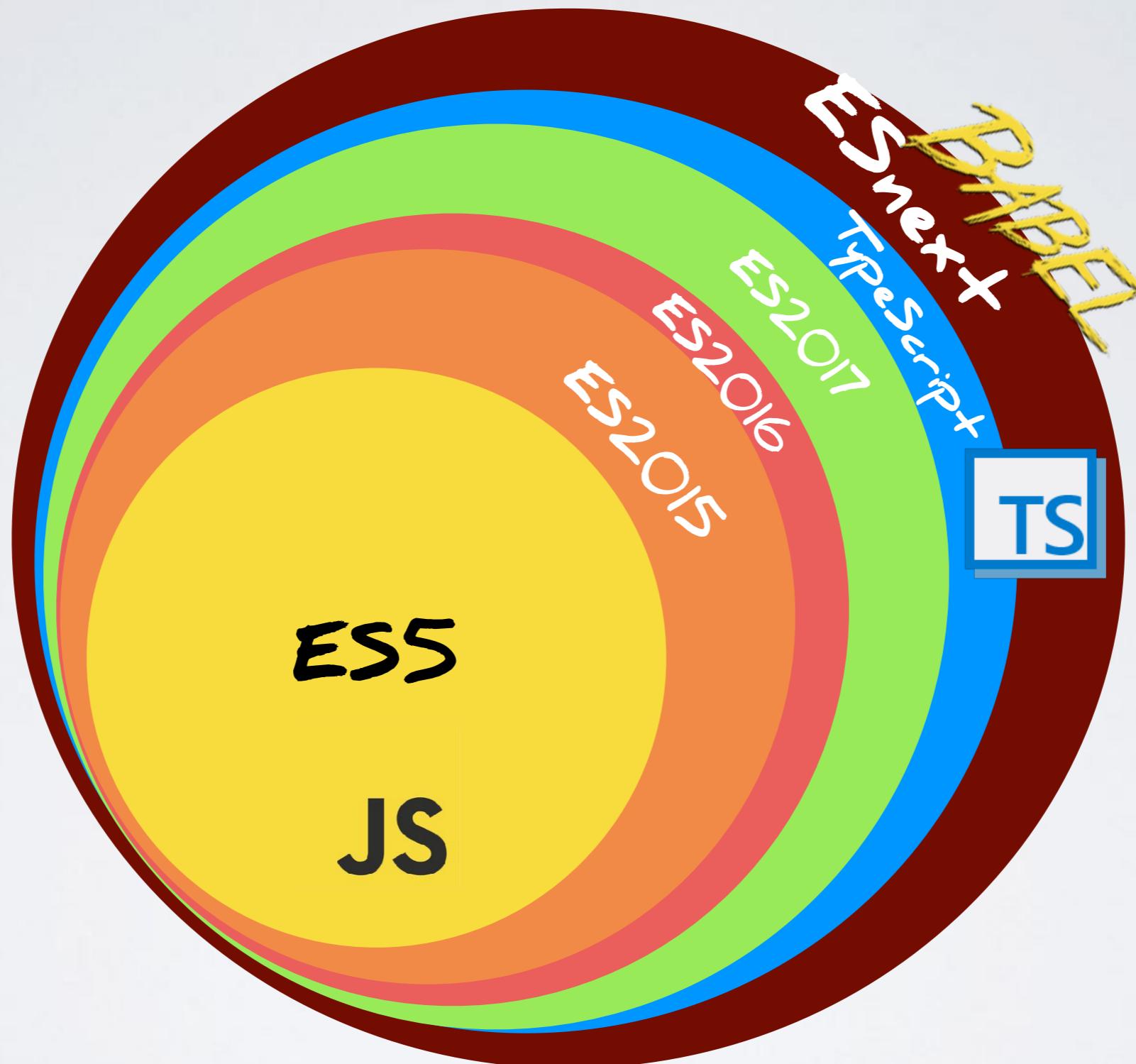
modules, classes, ...

static types

enables tools



TypeScript Today



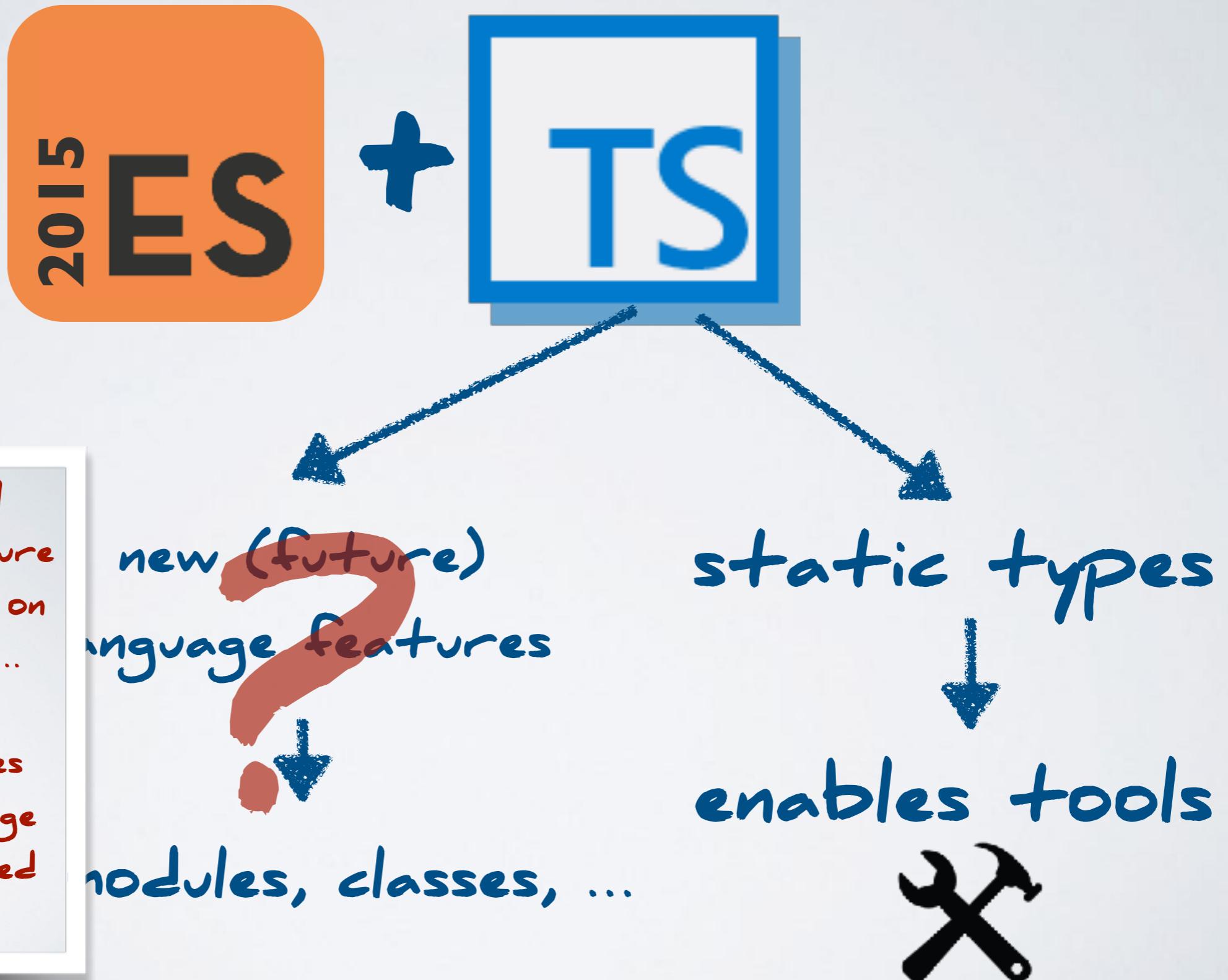
Additionally TypeScript has an optional compile-time static type system on top of ECMAScript and the browser APIs.

Static Typing & Access Modifiers

```
class Counter {  
  
    private count = 0;  
  
    increase(amount: number){  
        this.count += amount;  
    }  
    getValue(): number {  
        return this.count;  
    }  
}  
  
let counter = new Counter();  
counter.increase(2)  
console.log(counter.getValue())
```

Static typing and access modifiers are not part of JavaScript. The compiler removes this information at build time.

TypeScript Today



ES2015 Today: Transpiler & Polyfills

New Language Features (Syntax)



transpiling to ES5 for older browsers

TypeScript
BABEL

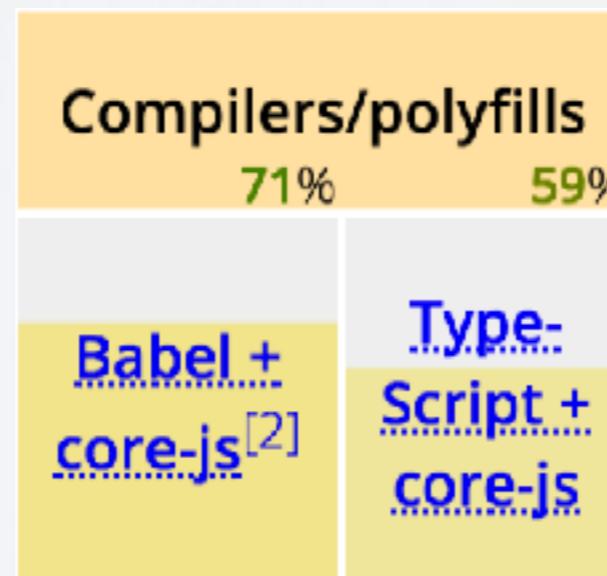
<https://kangax.github.io/compat-table/es6/>

New Built-In Functionality (Runtime Objects)



polyfills for older browsers

A polyfill is a JavaScript library that implements a missing browser feature.



core-js
babel-polyfill
polyfill.io

<https://github.com/zloirock/core-js>
<https://polyfill.io/v2/docs/examples>

```
    game._INIT_ACCEL = 0.05;
    game._INIT_TOTAL_PLATFORMS = 10;
    game._INIT_LEVEL_DIFF = [3,3,4,4,4,4,4,4,4,4];
}

function initialize() {
    console = document.getElementById("console");
    canvas = document.getElementById("game");
    context = canvas.getContext("2d");

    canvas.width = width;
    canvas.height = height;
}
```

JavaScript Basics

JavaScript

Language (Syntax)

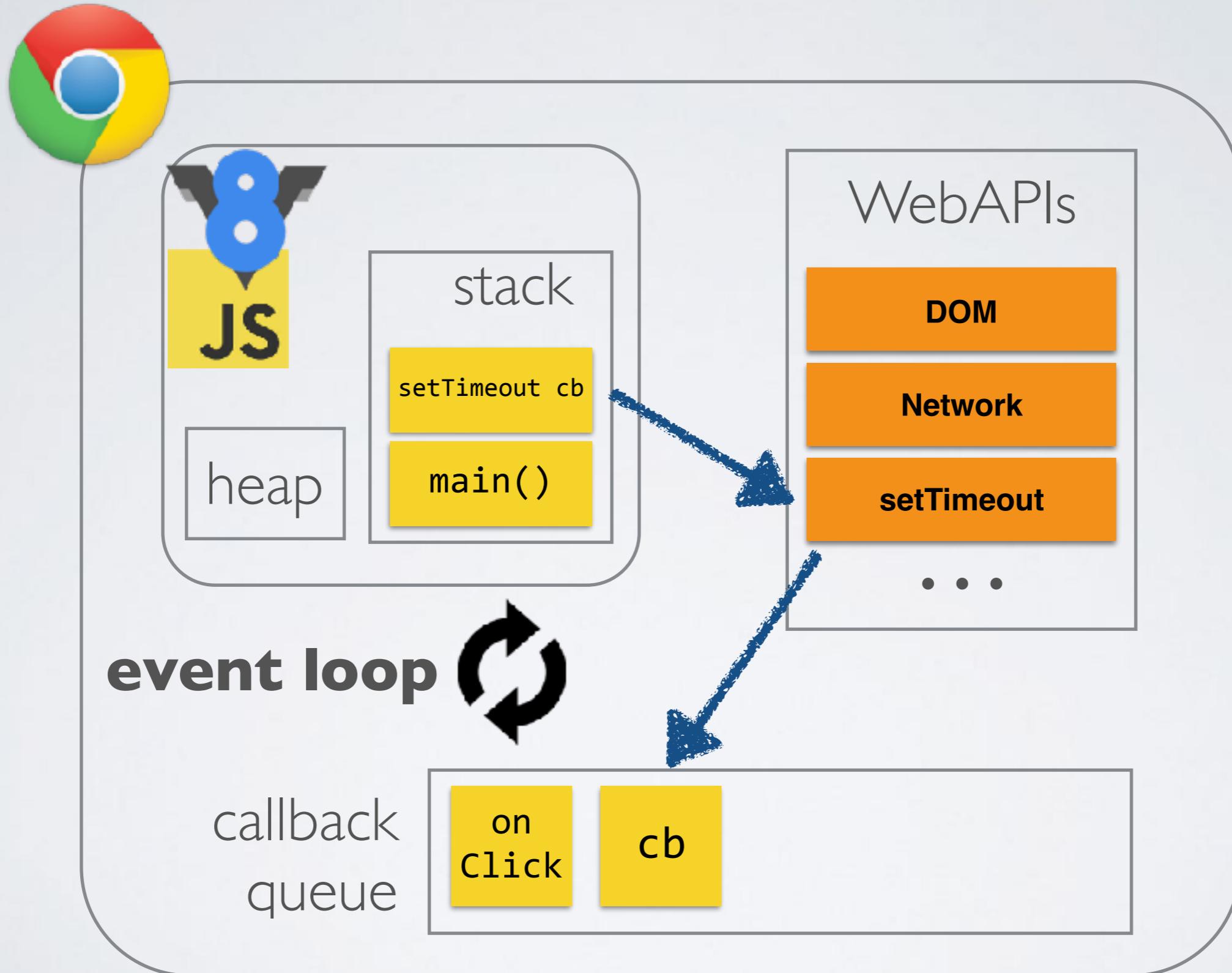
Runtime

**JavaScript Runtime
Built-In Objects**

**Browser Built-In
Objects & APIs
(DOM, AJAX ...)**



Event Loop



The Variable **this**

- **this** is the context of a function. The value of **this** is determined by *how a function is called*:
 1. If the function is called as a constructor with the **new** keyword, **this** is a new object
 2. If the method is called via `call()` or `apply()`, **this** is explicitly passed
 3. If the function is called as a method of an object, **this** points to that object
 4. If none of the above are used, **this** is the global object (or **undefined** in strict mode).
- Especially for callbacks it might be tricky to find out the value of **this**.

Higher Order Functions

- Higher order functions operate on functions
 - functions as parameters
 - functions as return values
- Callback Functions
- Partial Application

```
$( "#myBtn" ).on('click', function() {  
    alert("Button Clicked!");  
});
```

```
function arrayForEach(array, func) {  
    for (var i = 0; i < array.length; i++) {  
        func(array[i]);  
    }  
}  
arrayForEach([1,2,3,4,5],  
            function(msg){console.log(msg)});
```

```
var add = function(first, second) {  
    return first + second;  
};
```

```
var splitCall = function(first, func){  
    return function(second){  
        return func(first, second);  
    }  
}
```

```
var addOne = splitCall(1, add);  
addOne(22); // -> 23
```

ES5: bind()

```
function print(){console.log(this.message);}
var controller = {message: 'Hello!', greet: print};
setTimeout(controller.greet); // Does not work!
```

Manual explicit binding:

```
function bind(fn, object){
  return function(){
    fn.apply(object)
  }
}
```

```
setTimeout(bind(print, controller));
```

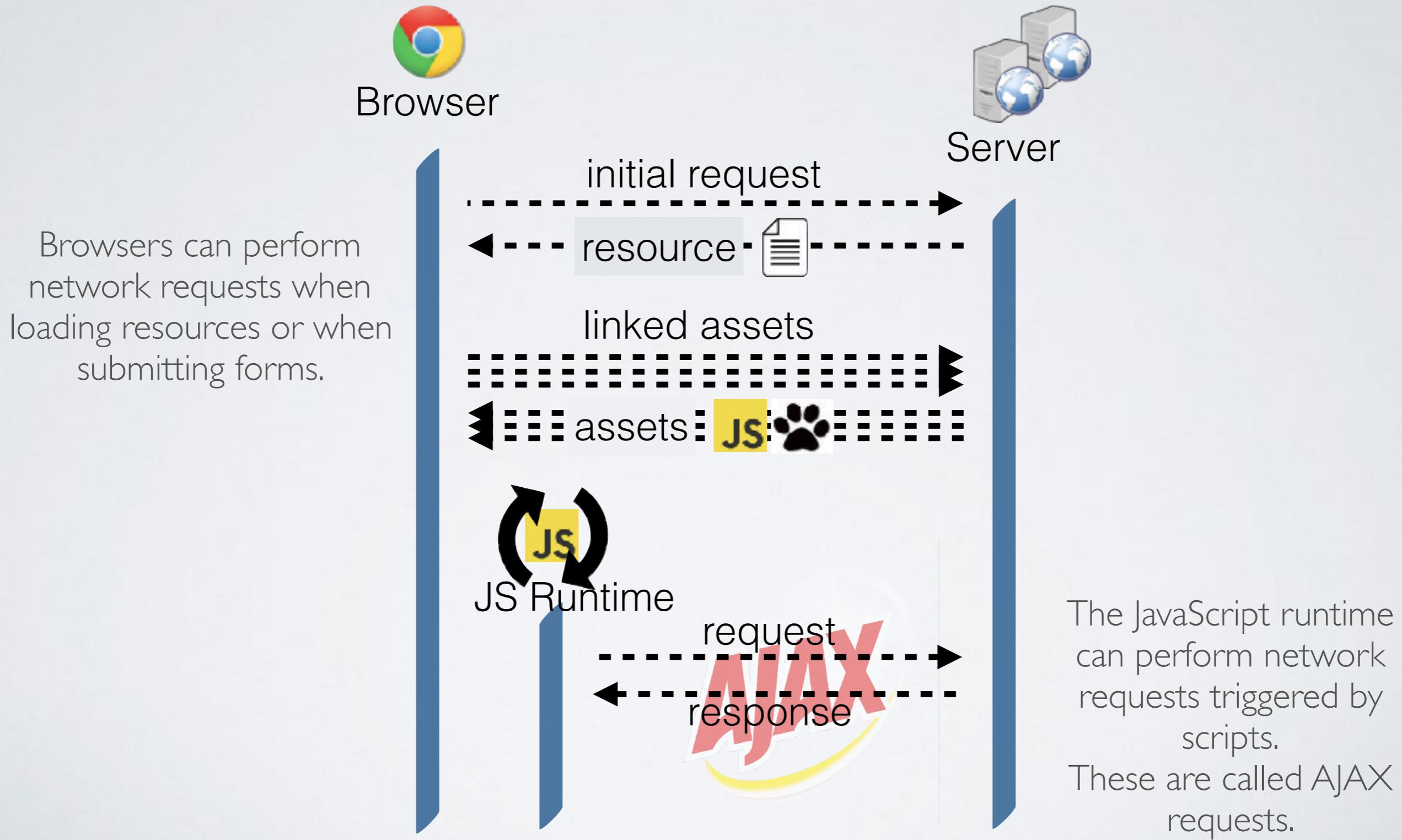
ES5 provides the **bind** method:

```
setTimeout(print.bind(controller));
```

AJAX & Async JavaScript



Browsers & Network Access



XMLHttpRequest

The underlying technology for AJAX.

```
var req = new XMLHttpRequest();
req.addEventListener("load", done);
req.addEventListener("error", failed); // only fired on network-level events
req.addEventListener("readystatechange", stateChanged);
req.open("GET", "http://localhost:3001/comments");
req.send();

function stateChanged(){
    if (this.readyState === 4) {
        if (this.status === 200) {
            console.log('success state', this.responseText)
        } else {
            console.log('error state', this.statusText);
        }
    }
}

function done() {
    console.log('DONE', this.responseText);
}

function failed() {
    console.log('ERROR', this.statusText);
}
```

The API is low-level, complicated and verbose.

AJAX with Callbacks

Using jQuery

```
$.get('http://localhost:3001/comments', function (data) {  
    console.log(data);  
});
```

```
$.post('http://localhost:3001/comments',  
    {text: 'test - ' + new Date()},  
    function () {  
        console.log('POST!');  
   });
```

fetch: a modern AJAX alternative

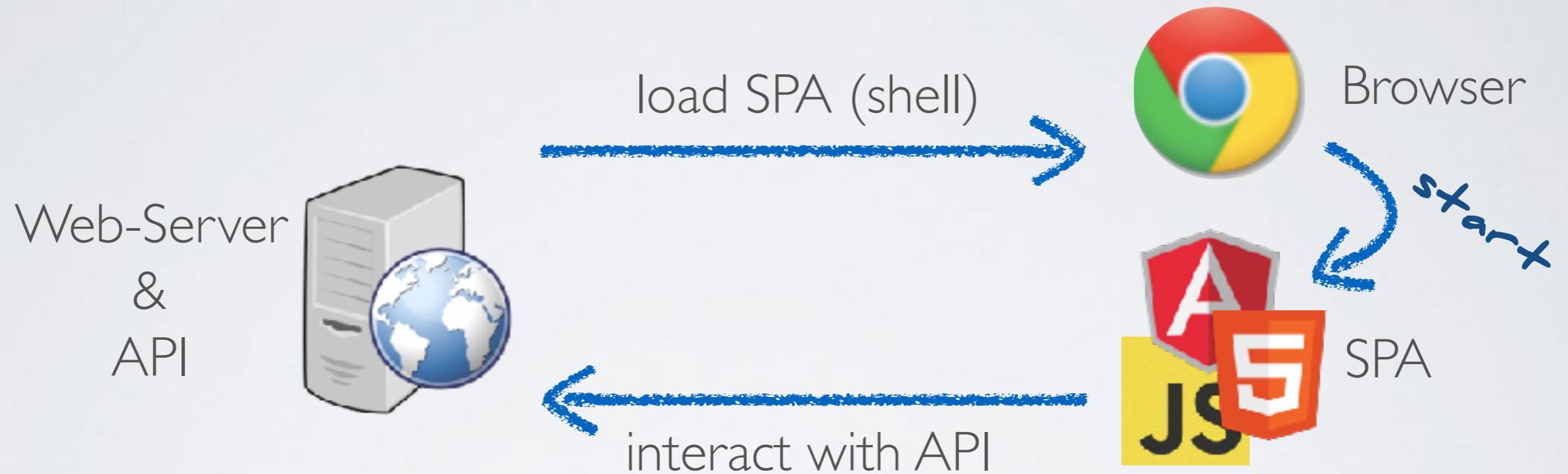
fetch is a new API available in modern browsers that is more elegant than XMLHttpRequest.

```
fetch('http://localhost:3001/comments2')
  .then(response => response.json())
  .then(data => console.log('SUCCESS', data))
  .catch(error => console.log('ERROR:', error));
```

Browser support:
<http://caniuse.com/#search=fetch>

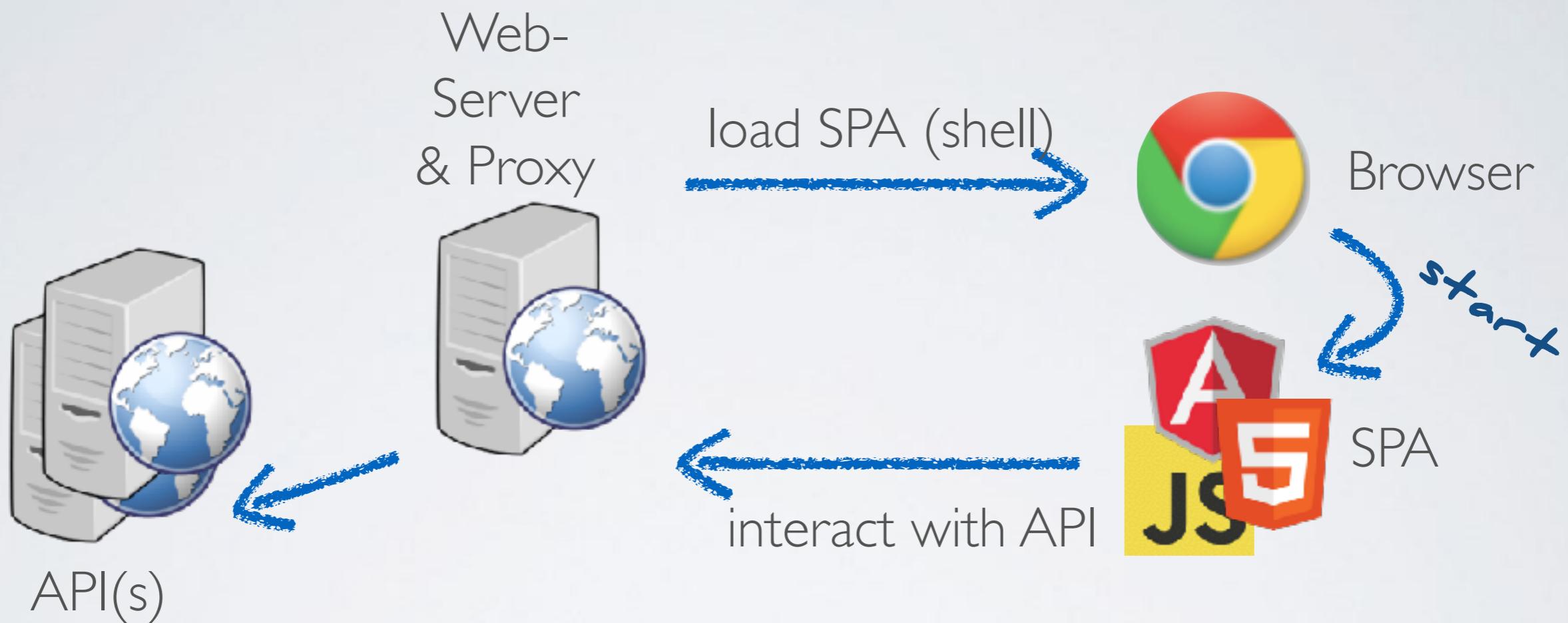
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
<https://fetch.spec.whatwg.org/#fetch-api>
<https://jakearchibald.com/2015/thats-so-fetch/>
<https://github.com/github/fetch>

"Traditional Web-App"



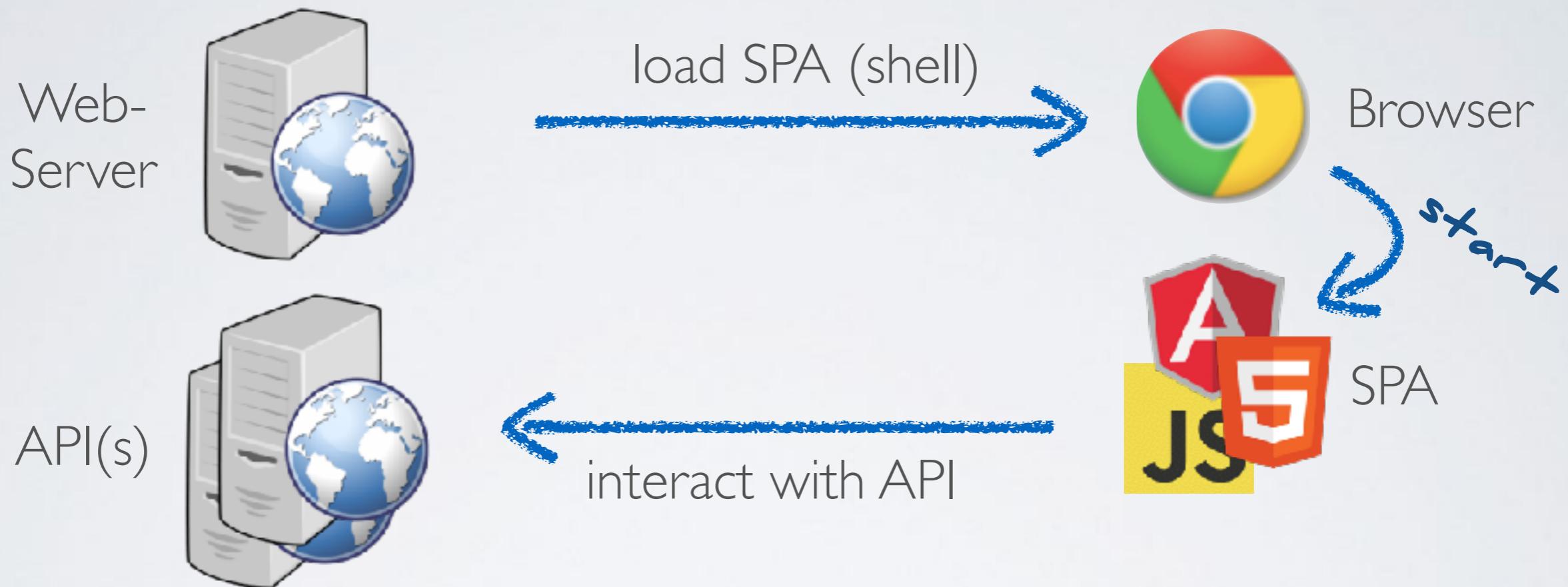
- Shell can be static or dynamic (rendered on the server)
- Same origin for all resources (shell & API)
- Often combined technologies on the Server (MVC & WebAPI, JSF & JAX-RS)
- "Monolithic", Advantages: Authentication/Authorization, Deployment ...

Web + Proxy + API-Backend



- One central entry point for the application
- Same origin for all resources (shell & API)
- Distributed / independent services ("SOA", "Microservices" ...)
- Advantages: logging, transactions, authentication

Web + API-Backend



- Shell is typically static (simple web-server or CDN)
- Different origin for shell & API resources
- Distributed / independent services ("SOA", "Microservices" ...)

Same Origin Policy (SOP)

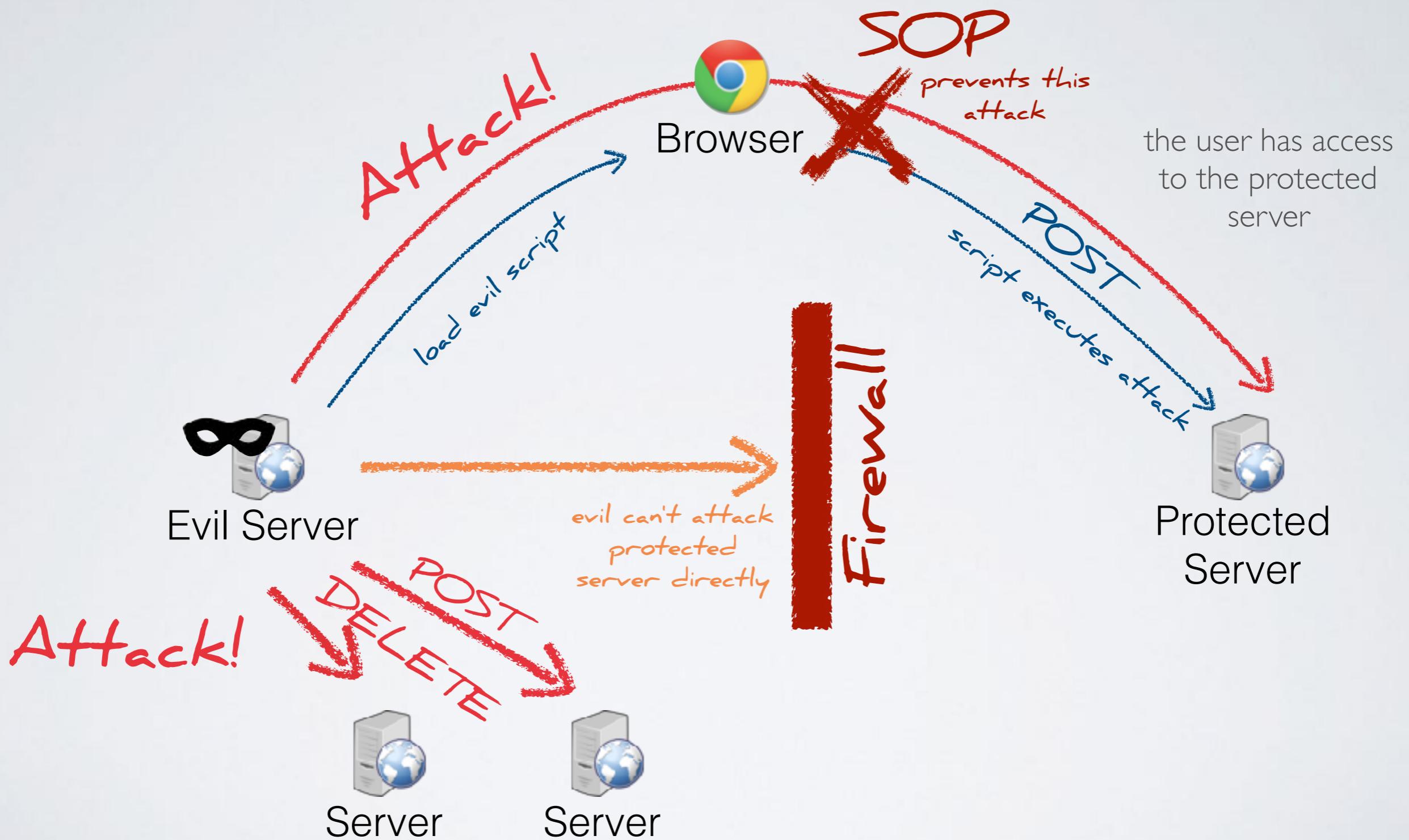
The same-origin policy restricts how a document or script loaded from one origin can interact with a resource from another origin.

- A page can't access data on another page
- A script can only make AJAX calls to resources from the origin it was loaded from
- Example: A script from "example.com" can't read or write your emails from "gmail.com"

The SOP protects the client not the server!

The SOP prevents "distributed APIs"!

SOP



Cross Origin Resource Sharing (CORS)

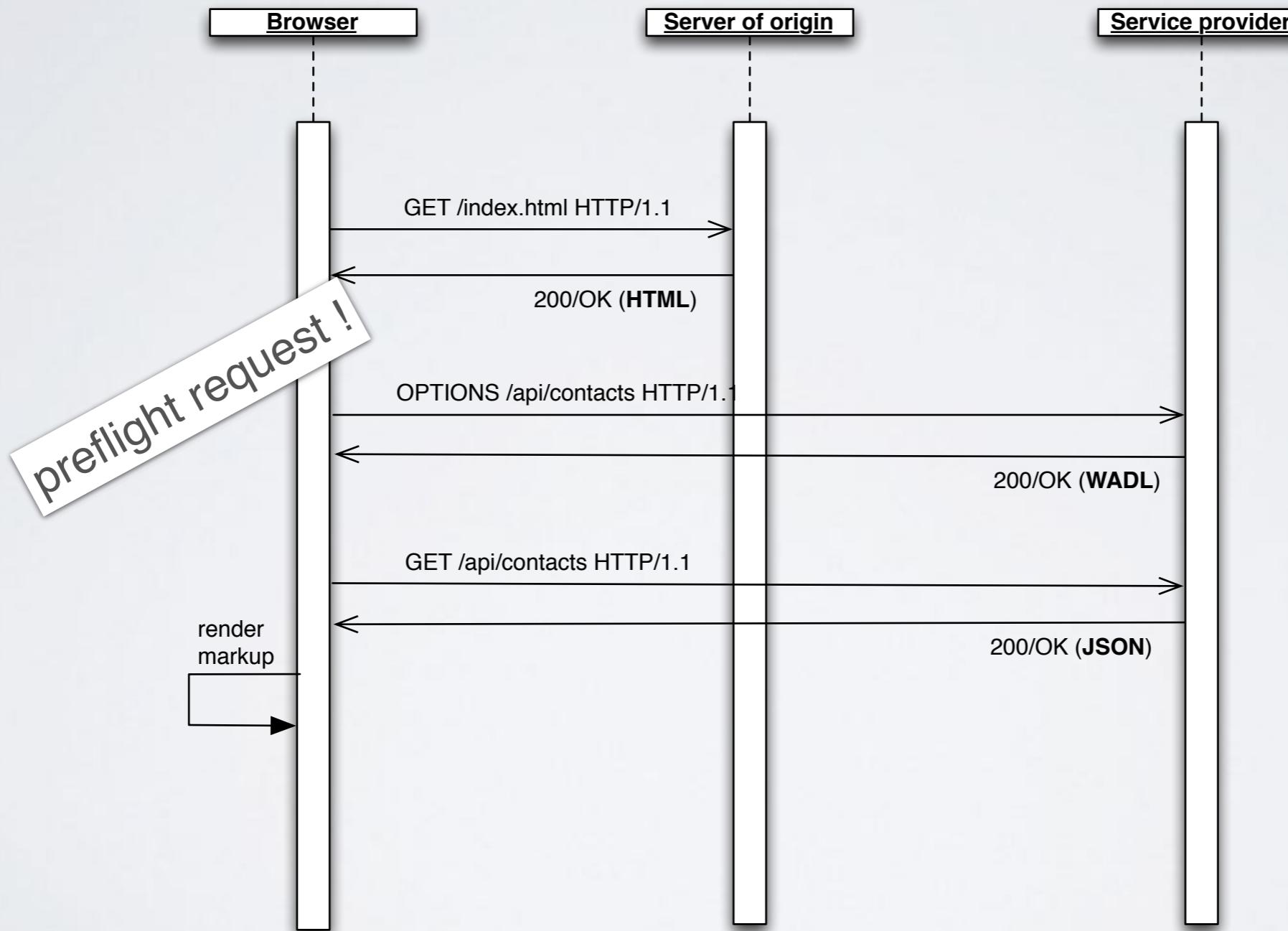
CORS allows to circumvent the SOP to implement "distributed APIs".

- A server API can define that its resources can be accessed from other origins.
- Browsers then allow scripts to make AJAX calls to that API.

CORS is implemented on the server but affects the behaviour of the browsers.

CORS only affects browser clients. Other clients (i.e. mobile) are not affected.

CORS: Preflight Request



Setting CORS Headers

JEE WebFilter

```
@WebFilter(filterName = "CorsFilter", urlPatterns = {"/rest/ratings-cors/*"})
public class CorsFilter implements Filter{

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
                         FilterChain filterChain) throws IOException, ServletException {
        final HttpServletRequest httpRequest = (HttpServletRequest) servletRequest;
        final HttpServletResponse httpResponse = (HttpServletResponse) servletResponse;

        httpResponse.addHeader("Access-Control-Allow-Origin", "*");
        httpResponse.addHeader("Access-Control-Allow-Methods",
                              "GET, POST, PUT, DELETE, OPTIONS");
        httpResponse.addHeader("Access-Control-Allow-Headers",
                              "content-type, x-requested-with, accept, origin, authorization, X-PINGOTHER");
        ...
    }
}
```

Promises



- A promise represents the result of an asynchronous operation.



Promises

- A promise represents the result of an asynchronous operation.
 - Counterparts: Future<> in Java or Task<> in .NET (TPL)
- Promises enable another programming model for asynchronous functions than the callback programming model.
 - Asynchronous functions can return a value without blocking (so they look much more like synchronous functions)
 - The return value is a promise that represents the final value that is possibly not yet known

Using Promises

Consume a promise:

```
var promise = startAsyncOperation();
promise.then(function (value) {
    // success handler
})
.catch(function (error) {
    // error handler
});
```

Provide a promise (ECMAScript 2015):

```
function startAsyncOperation(){
    var promise = new Promise(function (resolve, reject) {
        // resolve the promise later
        setTimeout(function () { resolve("Success!"); }, 1000);
    });
    return promise;
}
```

Advantages of Promises: Chaining Async Operations

Callback-Hell:

(aka Pyramid of Doom)

```
api(function(result){  
    api2(function(result2){  
        api3(function(result3){  
            // do work  
        });  
    });  
});
```

Promise Chaining:

If a "then-callback" returns another promise, the next "then" waits until it is settled.

```
api().then(function(result){  
    return api2();  
}).then(function(result2){  
    return api3();  
}).then(function(result3){  
    // do work  
}).catch(function(error) {  
    //handle any error  
});
```

```
api().then(api2)  
    .then(api3)  
    .then(/* do work */);
```

Advantages of Promises: Combining Multiple Actions

```
var firstData = null, secondData = null;

var responseCallback = function() {
  if (!firstData || !secondData) return;
  // do something
}

getData("http://url/first", function(data) {
  firstData = data;
  responseCallback();
});

getData("http://url/second", function(data) {
  secondData = data;
  responseCallback();
});
```

```
var promise1 = getData("http://url/first");
var promise2 = getData("http://url/second");
```

```
Promise.all([promise1, promise2])
  .then(function(arrayOfResults) {...})
  .catch(function(errOfPromise) {...});
```

```
Promise.race([promise1, promise2])
  .then(function(valOfPromise) {...})
  .catch(function(errOfPromise) {...});
```

Promise Chaining

→ t

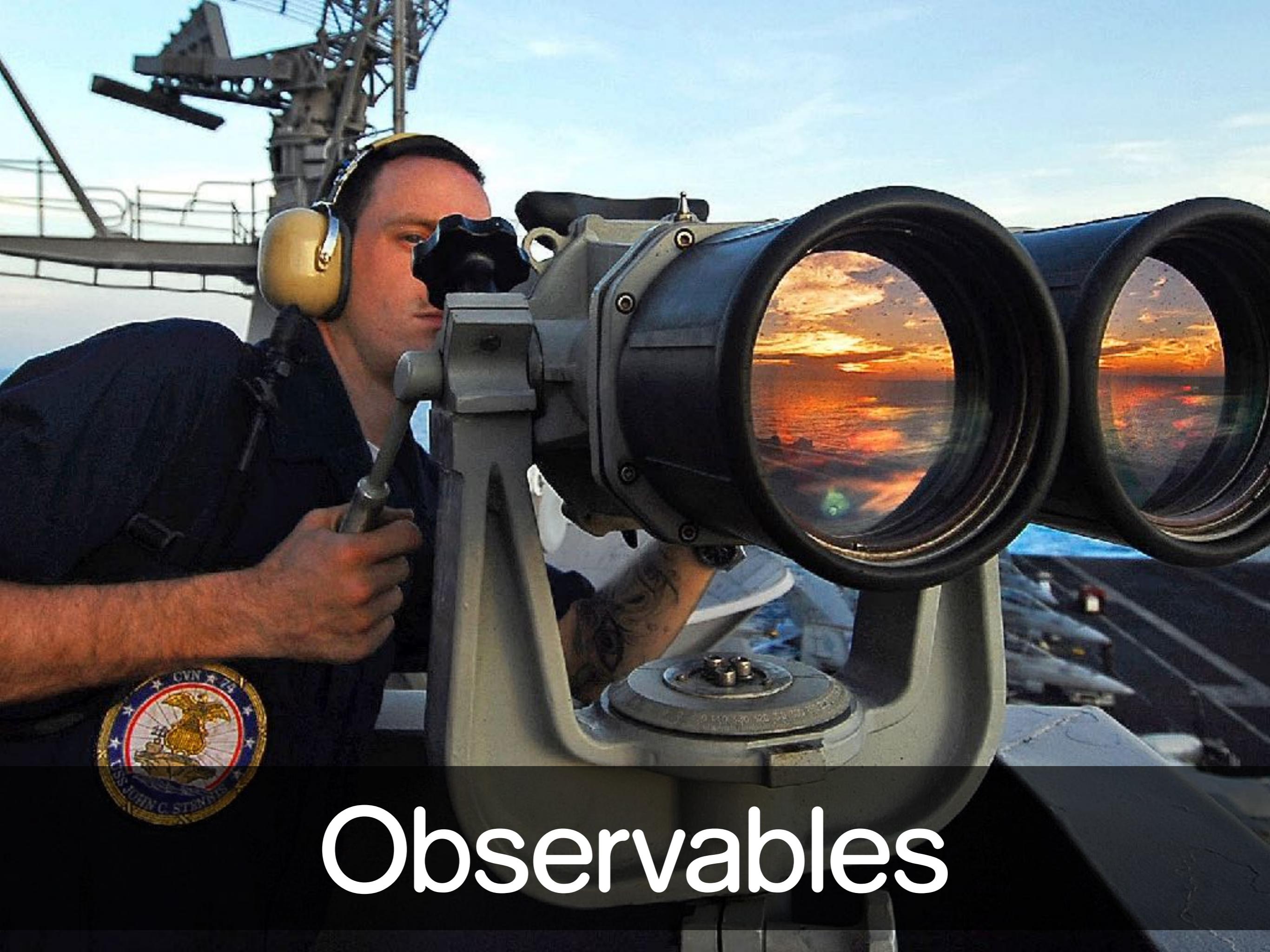
```
doSomething().then(function () {  
    return doSomethingElse();  
}).then(finalHandler);
```

```
doSomething().then(function () {  
    doSomethingElse();  
}).then(finalHandler);
```

```
doSomething()  
    .then(doSomethingElse())  
    .then(finalHandler);
```

```
doSomething()  
    .then(doSomethingElse)  
    .then(finalHandler);
```





Observables

Observables

An observable represents a multi-value push protocol:

	Single	Multiple
Pull	Function	Iterator
Push	Promise	Observable

Typical scenario: an asynchronous stream of events.

```
const observable =  
  Rx.Observable.fromEvent(  
    document.getElementById('tick'), 'click'  
  );  
  
observable.subscribe(x => console.log(x));
```

Using Observables

Consume an observable:

```
var stream = startAsyncOperation();
stream.subscribe(
  value => document.getElementById("content").innerText += value,
  error => console.log('Error: ' + error),
  () => console.log('Completed!')
);
```

Provide a observable (using RxJS):

```
function startAsyncOperation(){
  var stream = new Rx.Observable(
    observer => {
      setTimeout(() => observer.next("This is first value!"), 1000);
      setTimeout(() => observer.next("This is second value!"), 2000);
      setTimeout(() => observer.next("This is third value!"), 3000);
      setTimeout(() => observer.complete(), 3000);
    });
  return stream;
}
```

Observables vs. Promises

An observable can be an alternative to a promise:
a stream that pushes exactly one result.

```
const observable = new Rx.Observable(  
  observer => setTimeout(  
    () => {  
      observer.next(42);  
      observer.complete();  
    },  
    2000);  
  
observable.subscribe(x => console.log(x));
```

Angular exposes observables for backend access via http service.

Observables vs. Promises

Angular promotes observables over promises as the programming model for backend access.

Promise

returns a single value

not cancellable

Observable

works with multiple values over time

cancellable

supports powerful operators like map, filter & reduce (“lo-dash for async”)

For AJAX calls observables do not provide a clear advantage. But observables are used consistently throughout Angular.

Combining Observables

```
function api1(){
  return Rx.Observable.of(42).delay(2000);
}
function api2(){
  return Rx.Observable.of(43).delay(1000);
}
function api3(){
  return Rx.Observable.of(44).delay(500);
}

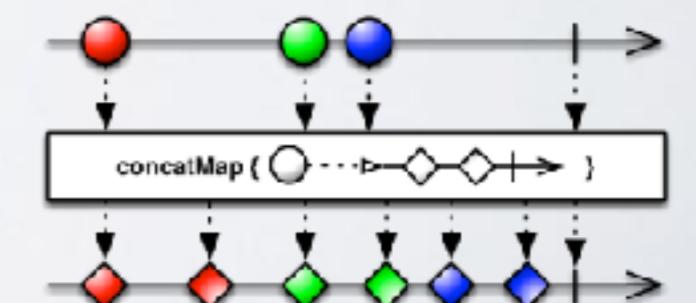
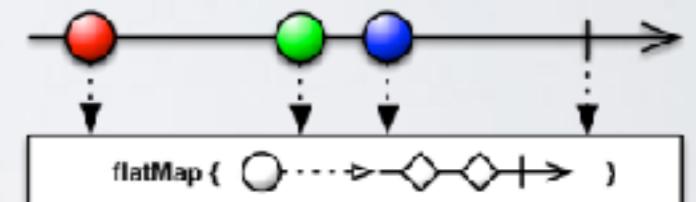
stream = Rx.Observable.from([api1, api2, api3]);
```

```
stream.flatMap(x => x())
  .subscribe(
    value => console.log(value)
  );
```

out:
44, 43, 42

```
stream.concatMap(x => x())
  .subscribe(
    value => console.log(value)
  );
```

out:
42, 43, 44



A collection of various hand tools including hammers, wrenches, pliers, and sockets.

Toolset

Dependency Management

Declare & Resolve project dependencies.
Including transitive dependencies.



Build Automation

Infrastructure to implement and run build steps.
Orchestrate other tools.



Module Bundling

Build one or several javascript files for deployment.
Resolve module dependencies.
Concatenate and minify JavaScript files.



Transpilation

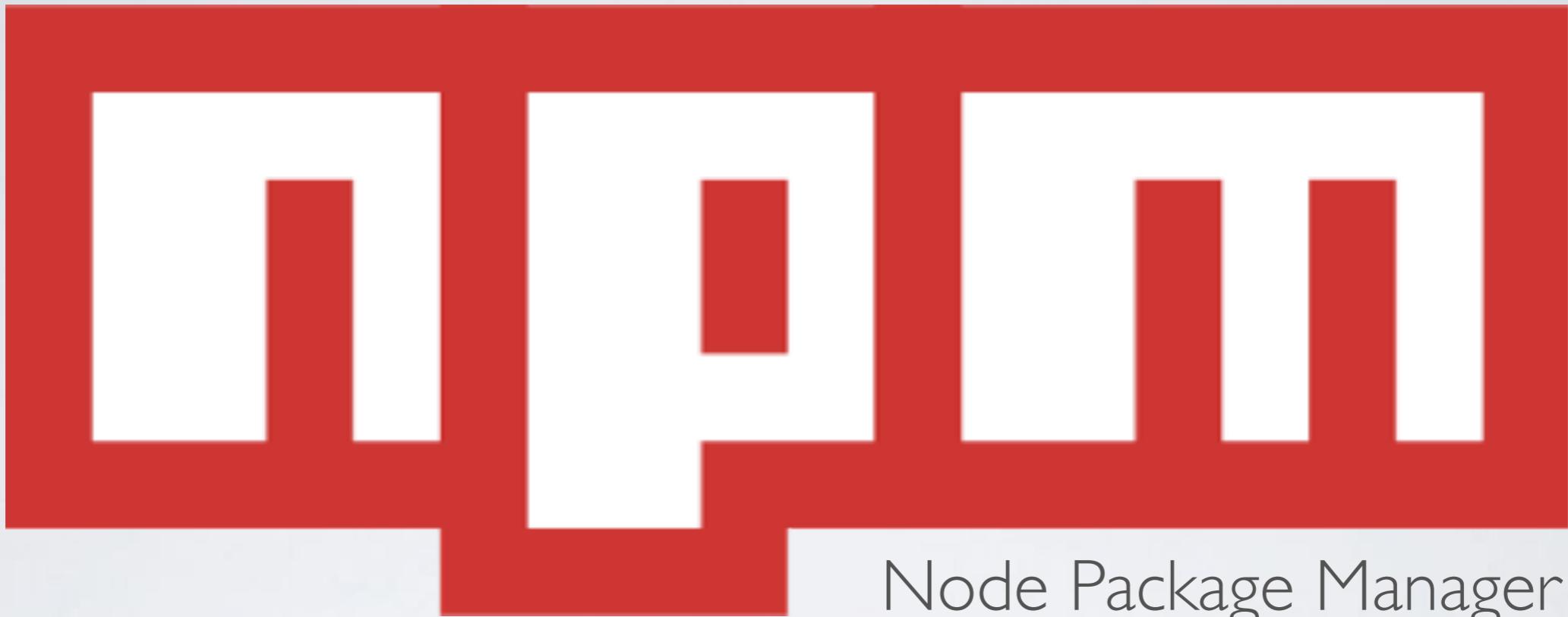
Transform development sources (ES2015+ / TS / JSX) into ES5.



Linting / Static Type-System

Static code analysis.
Compile-time checked static type-system





Node Package Manager

npm is installed together with Node

npm offers:

- Dependency Management (global and local packages)
- Simple Build Tool

Active Ecosystem



www.npmjs.org

Packages

585,507

Downloads · Last Day

184,548,673

Downloads · Last Week

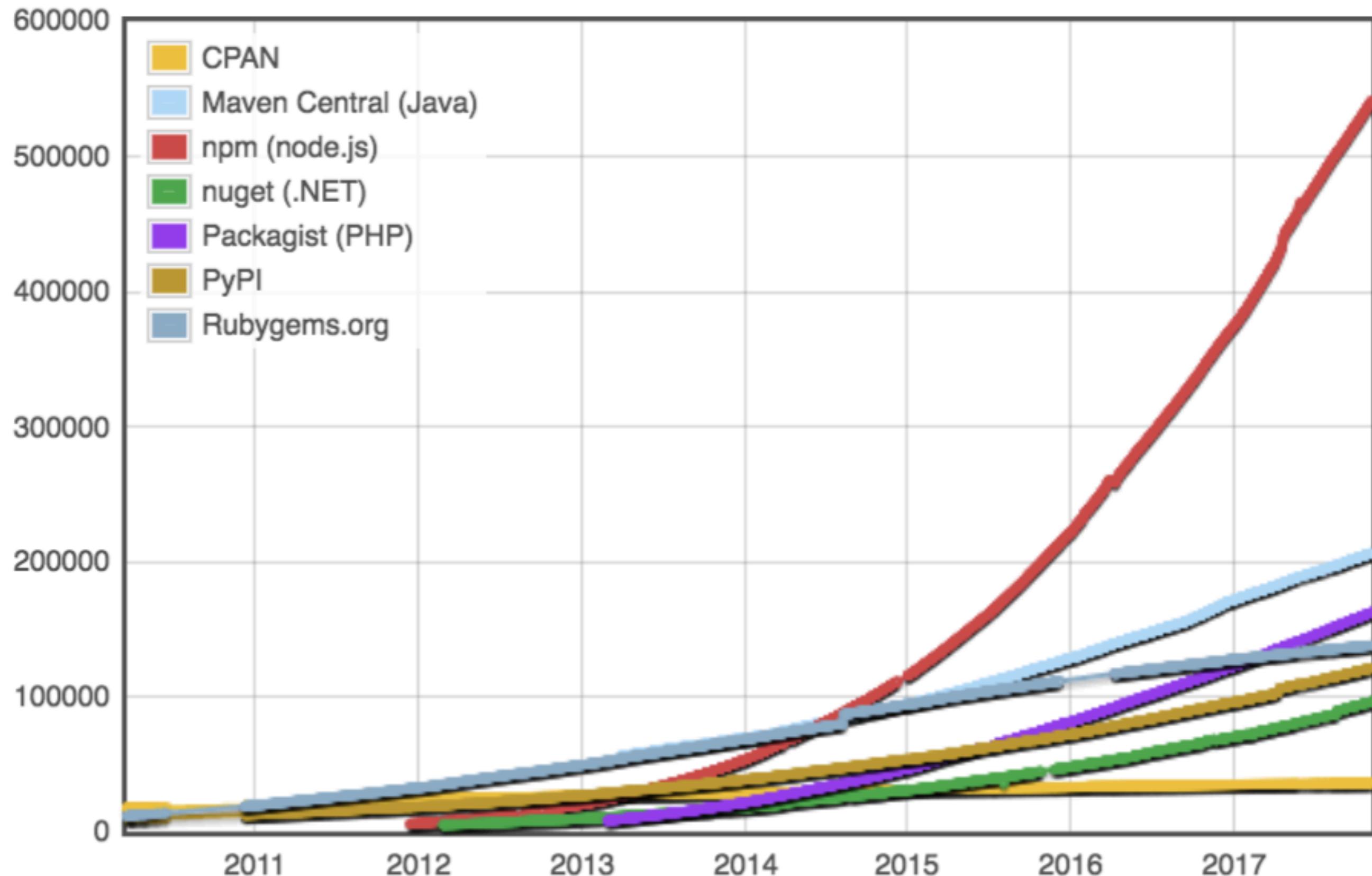
3,253,087,048

Downloads · Last Month

13,813,532,472

(Maven Central has
205,919 unique artifacts)

Module Counts





Node Package Manager

- Packages can be local (for the current project) or global
- **package.json** describes a package or project including it's dependencies
- packages are stored in **node_modules**
- hierarchical dependencies: dependencies can include their own dependencies
(you can have several versions of a package in your project)
- Dependencies are versioned according to semantic versioning (<https://semver.npmjs.com/>)
- Starting from npm 5, exact versions are listed in **package-lock.json**
- Repository: npmjs.org (or private)
- Config: **.npmrc**

Tip: `npm config set save-exact true`

See: <https://semver.npmjs.com/>

Typical commands:

`npm help`

`npm search`

`npm info`

`npm install`

`npm uninstall`

`npm list`

`npm update`

`npm init`

`npm root`

`npm config`

Flags:

`--global / -g`

`--help`

`--save / -S`

`--save-dev / -D`

Enterprise Concerns

There are options for a private npm registry:

- Nexus
<http://books.sonatype.com/nexus-book/2.10/reference/npm.html>
(Nexus 2 does not support scoped packages, i.e. @angular/core)
- Artifactory
<http://www.jfrog.com/confluence/display/RTF/Npm+Repositories>
- Visual Studio Team System:
<https://docs.microsoft.com/en-us/vsts/package/overview>
- NPM Enterprise: <https://www.npmjs.com/enterprise>
- Gemfury: <https://gemfury.com/>
- verdaccio: <http://www.verdaccio.org/>

```
npm config list
npm config set registry <registry url>
```

Config files:
<project>/.npmrc
\$HOME/.npmrc



yarn

...a better npm client.

<https://yarnpkg.com/>

npm@5 also introduced locking
and speed similar to yarn

Yarn is faster and provides a reliable dependency management by pinning all dependencies (including transitive dependencies) with a lockfile (`yarn.lock`).

Recommended installation via installer:

deprecated: `npm install --global yarn`

```
npm install
npm install --save [package]
npm install --save-dev [package]
npm run [script]
```



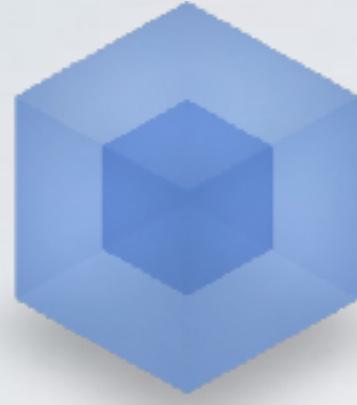
```
yarn
yarn add [package]
yarn add [package] --dev
yarn run [task]
```

Using yarn on CI

<https://yarnpkg.com/blog/2016/11/24/offline-mirror/>

<https://yarnpkg.com/en/docs/migrating-from-npm>

<https://shift.infinite.red/npm-vs-yarn-cheat-sheet-8755b092e5cc>



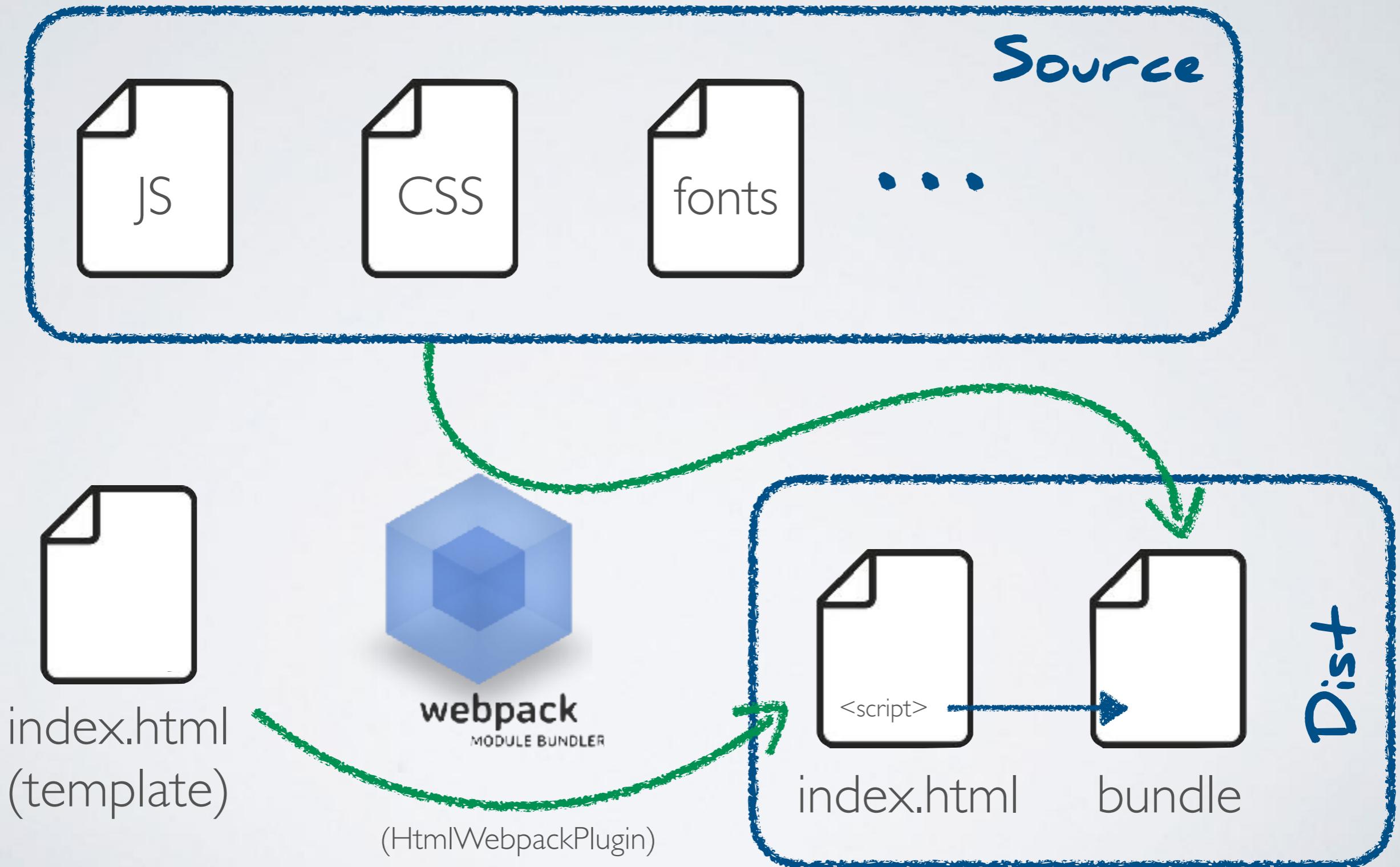
WebPack

Webpack is a bundler that bundles the fine grained assets of the application (primarily JavaScript and CSS) into one or several coarse grained bundles at build time.

The contents of a bundle are defined by building a dependency graph (following imports).

Bundles are JavaScript files that are loaded by the browser at runtime.

WebPack Build



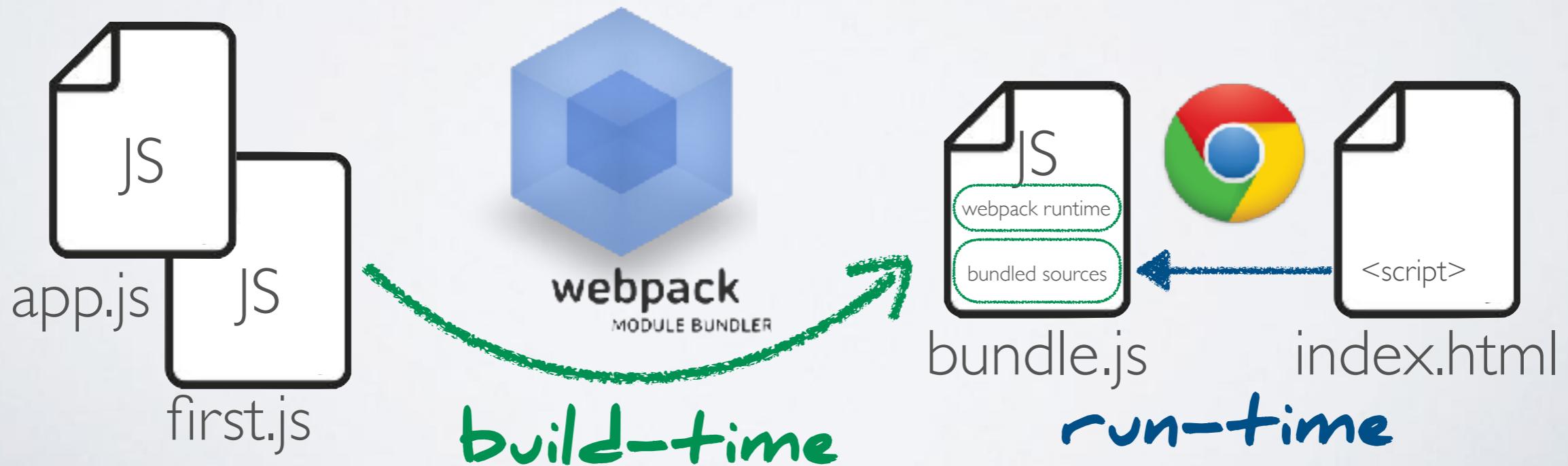
ES2015 Modules with WebPack

index.html

```
<script src="bundle.js"></script>
```

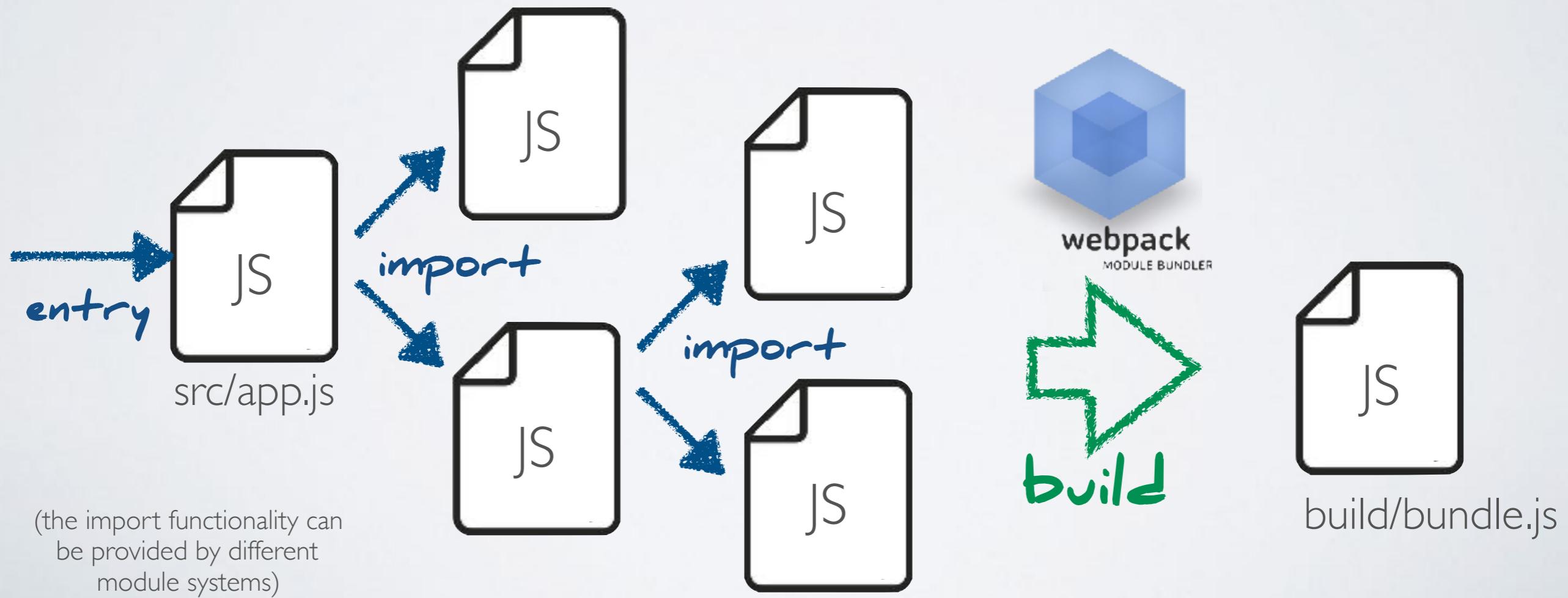
app.js

```
import './modules/first';
console.log("Hello from ES6 modules");
```

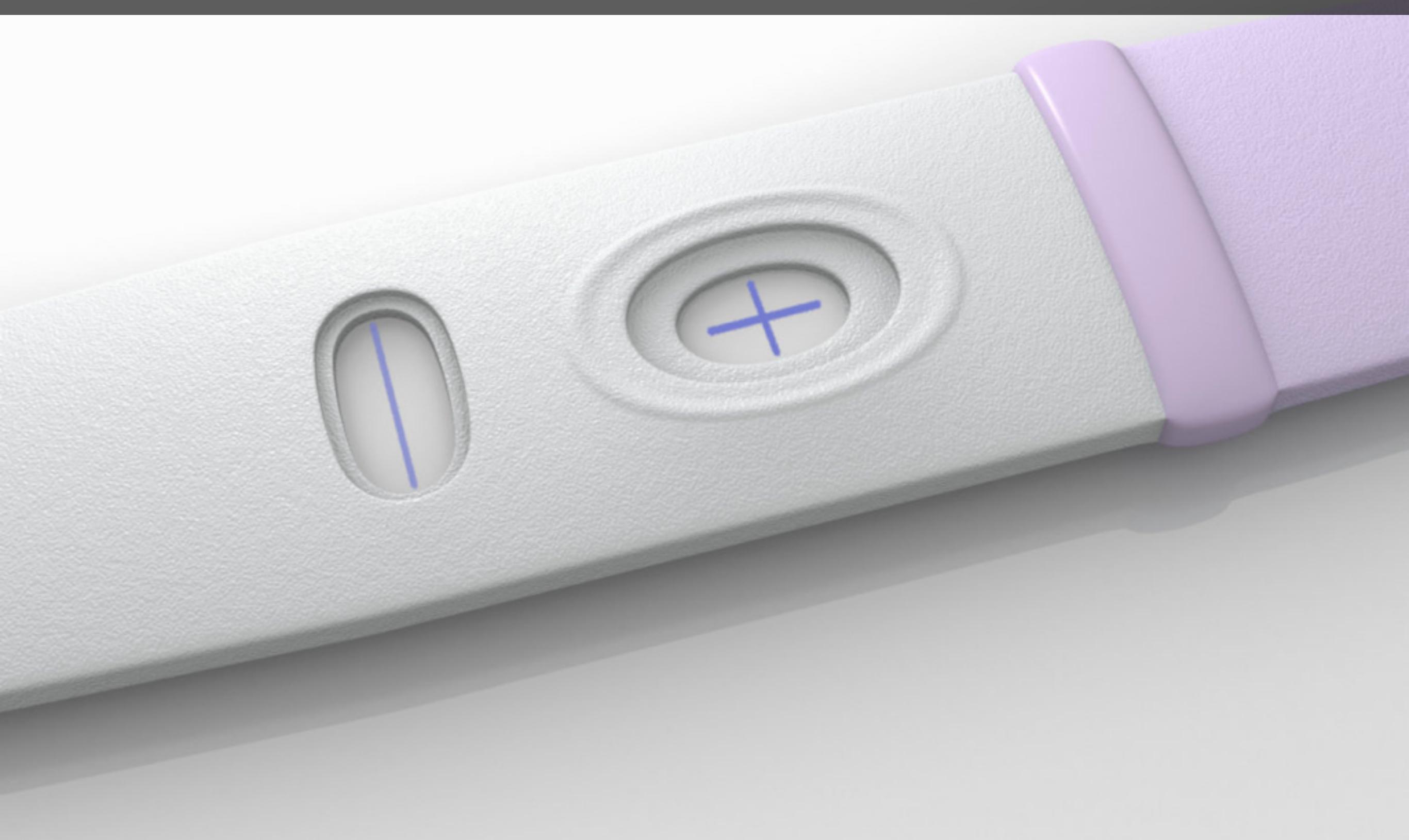


A minimal WebPack config

```
module.exports = {
  entry: './src/app/app.js',
  output: {
    filename: 'build/bundle.js'
  },
  devtool: 'source-map'
};
```



Testing



Test Framework

Formulate tests in code.
Run the tests in the browser.



tape



Jasmine

Node based test-runner

Run JavaScript tests in Node.js
No DOM is provided.
(use jsdom as a DOM implementation for node)



tape

Advanced Test Runner

Run tests in multiple browsers (including PhantomJS)
Provide different reporters (coverage, junit ...)
Run test continuously.
Automatically launch browsers.



Testem

End-to-End Testing

Automate browsers via Selenium



Protractor
end to end testing for AngularJS



Nightwatch.js

JS Testing Scenarios

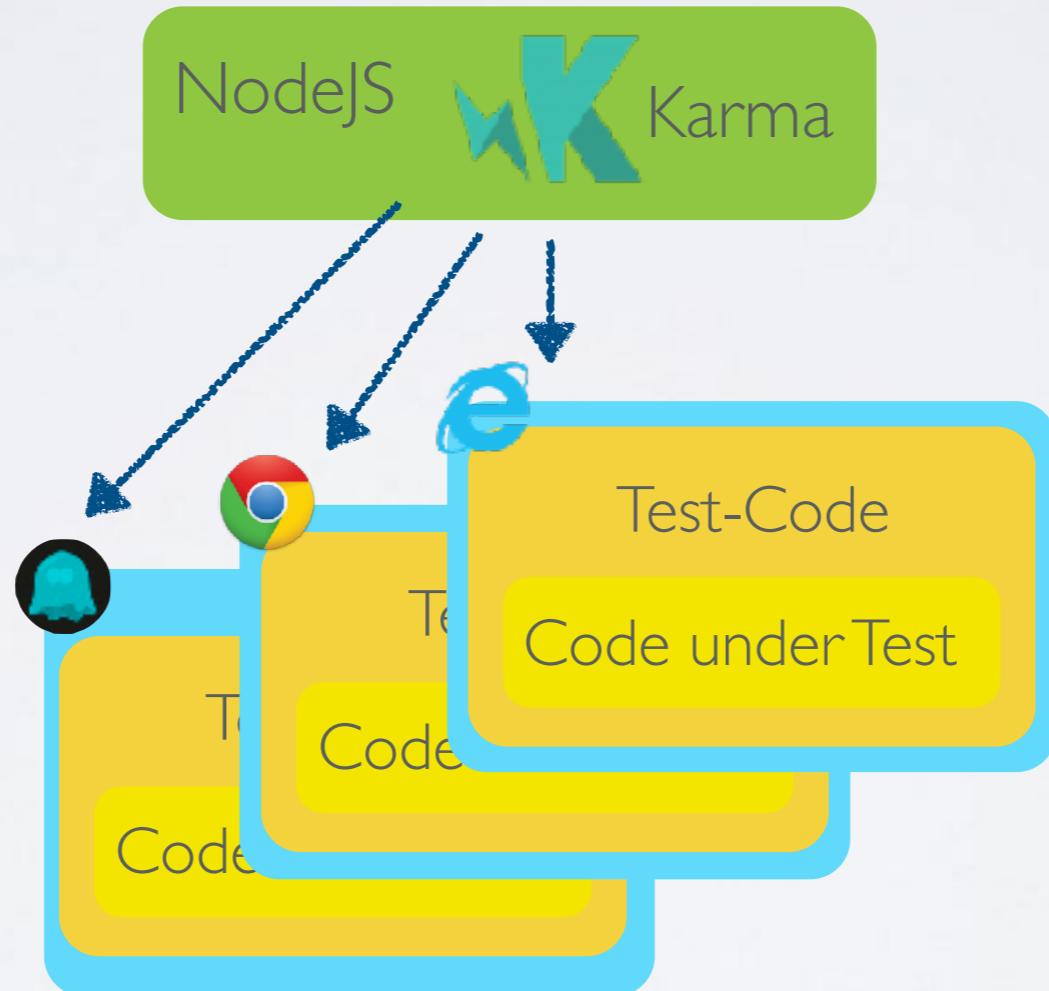
(where to run tests)

Pure Node-Testing
(unit-tests)



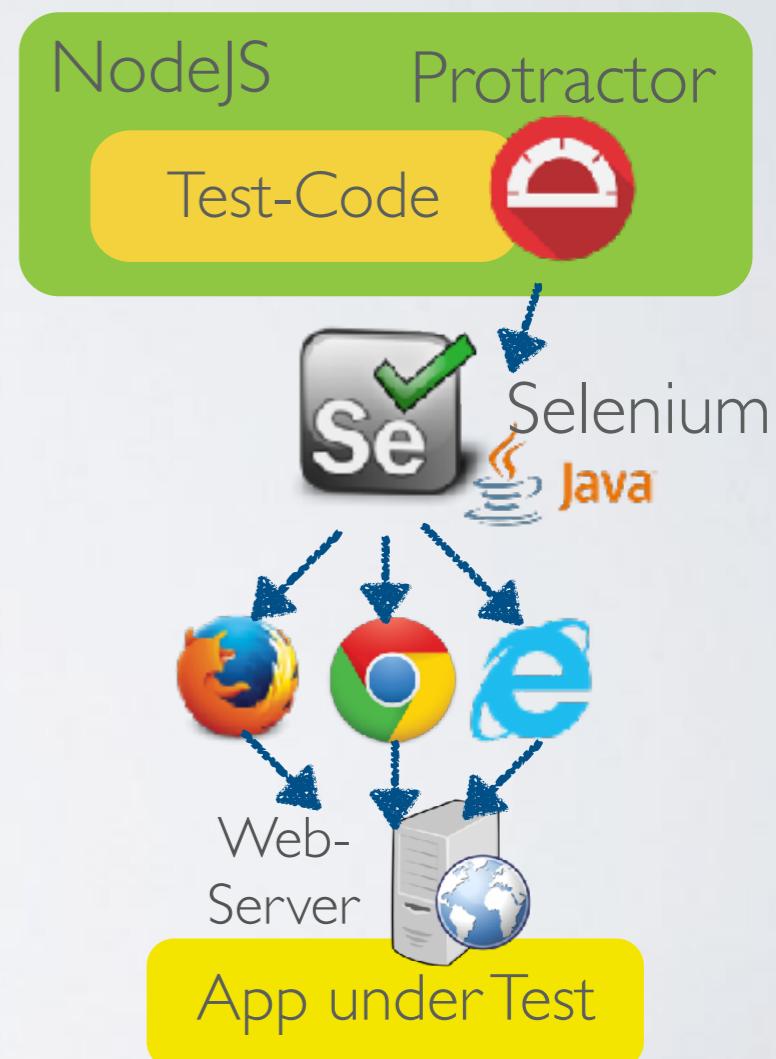
Run tests in node.
Optionally use jsdom
to fake DOM.

Karma
(unit-tests)



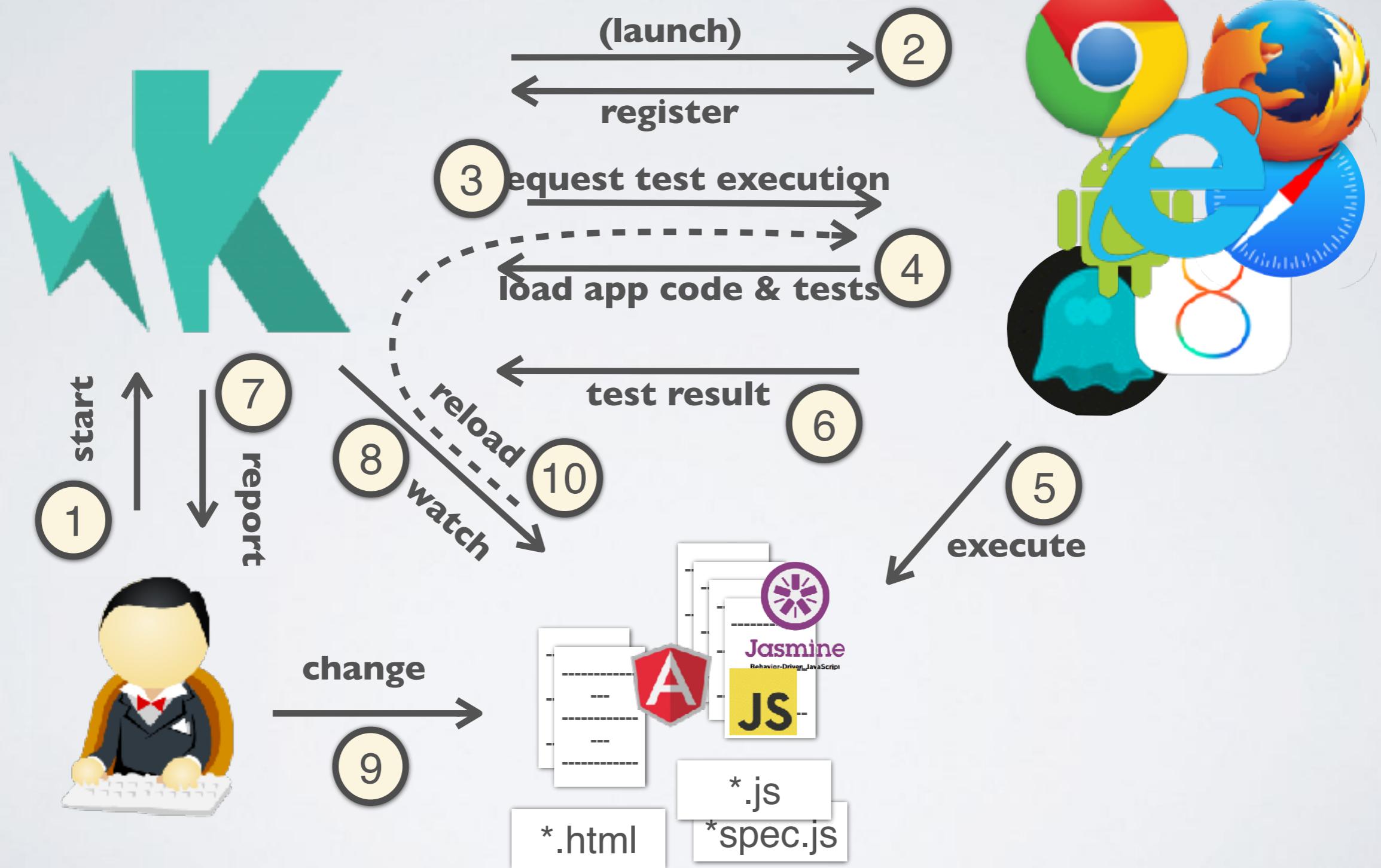
Run tests in the browser.
Optionally use as headless browser.

Protractor
(end-to-end tests, e2e)



Script a browser to interact
with a deployed app.

KARMA





Protractor

end to end testing for AngularJS

