

WEB ENGINEERING



PLAN

- Un peu d'histoire
- JavaScript : le langage d'action du Web (mais pas que ...)
- Outilage
- Nodejs
- TypeScript
- Utilisation de frameworks : AngularJS
- Utilisation de frameworks : Vue.js
- Utilisation de frameworks : AngularJS 2
- Utilisation de frameworks : React
- Conclusion

CONTEXTE HISTORIQUE

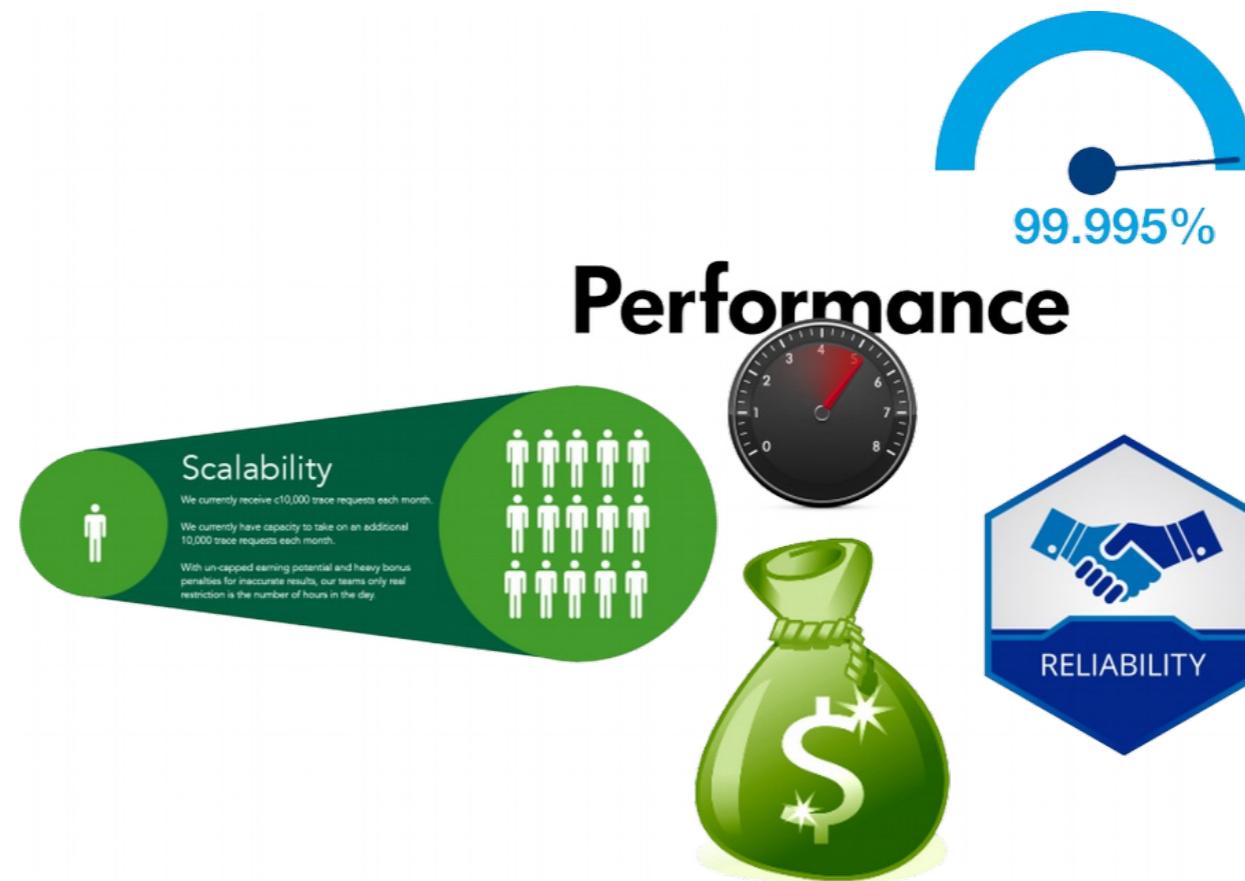
- Internet (~1980) et le Web (~1990)
 - Repose sur Internet, TCP/IP, modèle OSI
 - Un des services de l'Internet (port 80)
- 3 piliers : HTML, URI, HTTP
- Plusieurs évolutions
 - Séparation mise en forme (CSS), pages
 - Statiques vers interactions dynamiques
 - Cloud computing, Web sémantique, Web embarqué, systèmes pervasifs...
 - HTML5 (Conteneur d'applications complexes)

ÉVOLUTIONS DU WEB

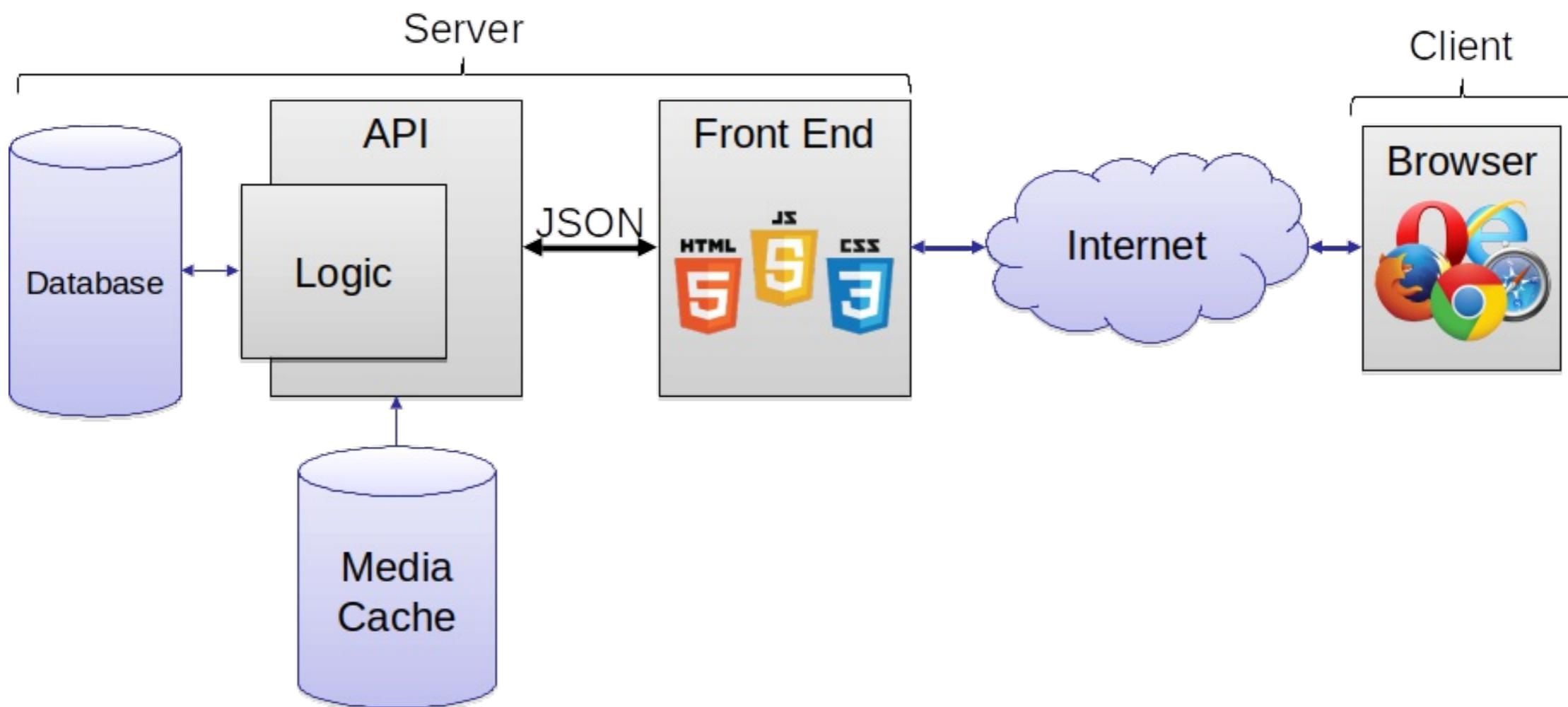
- Évolutions du Web => Web 2.0
 - Utilisateur proactif (wikipedia, blogs, etc.)
 - Interfaces riches, intégration d'applications
- Moyens techniques
 - Langage de scripts
 - côté serveur (PHP, ASP, C#...)
 - côté client (Javascript, flash...)
- Services Web (HTML => XML,JSON)
 - Échange de données entre applications

INGÉNIERIE WEB

- Availability
- Performance
- Reliability
- Scalability
- Manageability
- Cost



ELEMENT AU COEUR D'UNE APPLICATION WEB



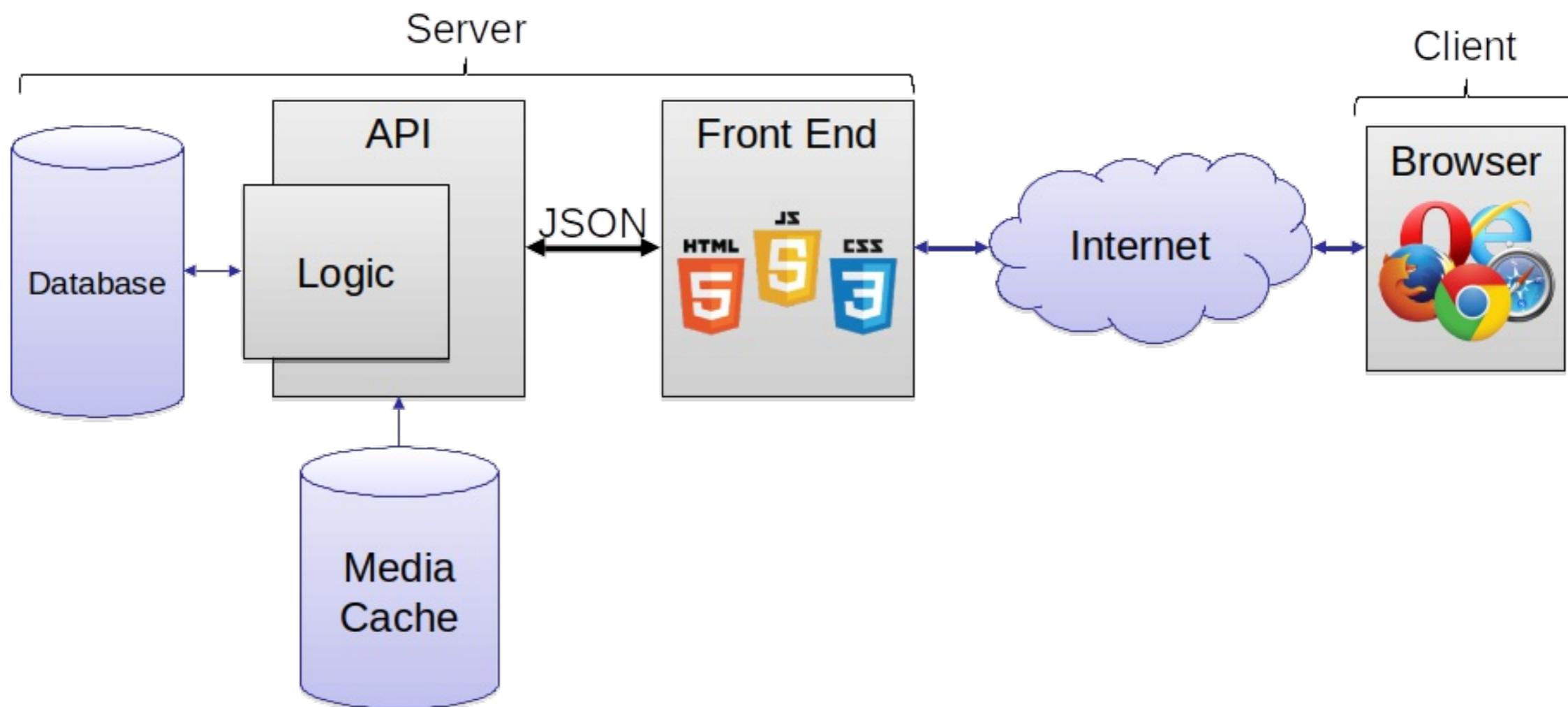
USE OF FRAMEWORKS?

WHAT IS A FRAMEWORK?

- Designed to reduce overhead in web development
- Types of Framework Architectures
 - Model-View-Controller (MVC)
 - Push vs Pull Based
 - Most MVC Frameworks user push-based architecture “action based” (Django, Ruby on Rails, Symfony)
 - Pull-based or “component based” (Vue, Angular2, React)
 - Three Tier Organization
 - Client (Browser running HTML/Javascript/CSS)
 - Application (Running the Business Logic) Database (Data Storage)

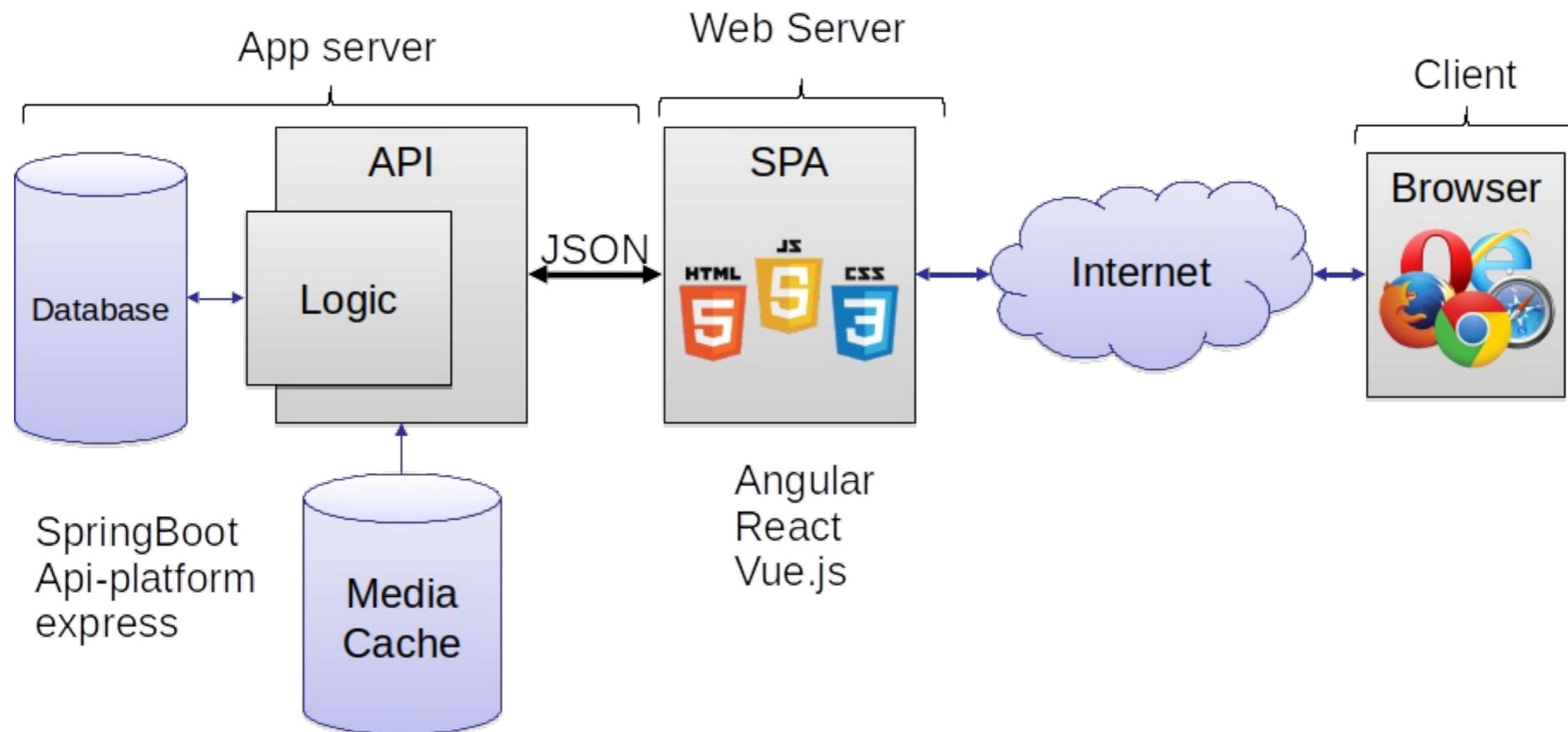
TYPES OF FRAMEWORKS

- Server Side: Django, symfony, Ruby on Rails (TAA)
- Client Side: Angular, React, Vue (CLI)



TYPES OF FRAMEWORKS AND SINGLE PAGE APPLICATION

- Server Side: Spring Boot, Express, api-platform (TAA)
- Client Side: Angular, React, Vue (CLI)



FRONTEND DEVELOPMENT

FRONT END RUNTIME LANGUAGES

- HTML/CSS
- Javascript



Javascript/HTML/CSS is the only real option for front-end native languages and is basically the standard. But there are many variations on JavaScript that are used.

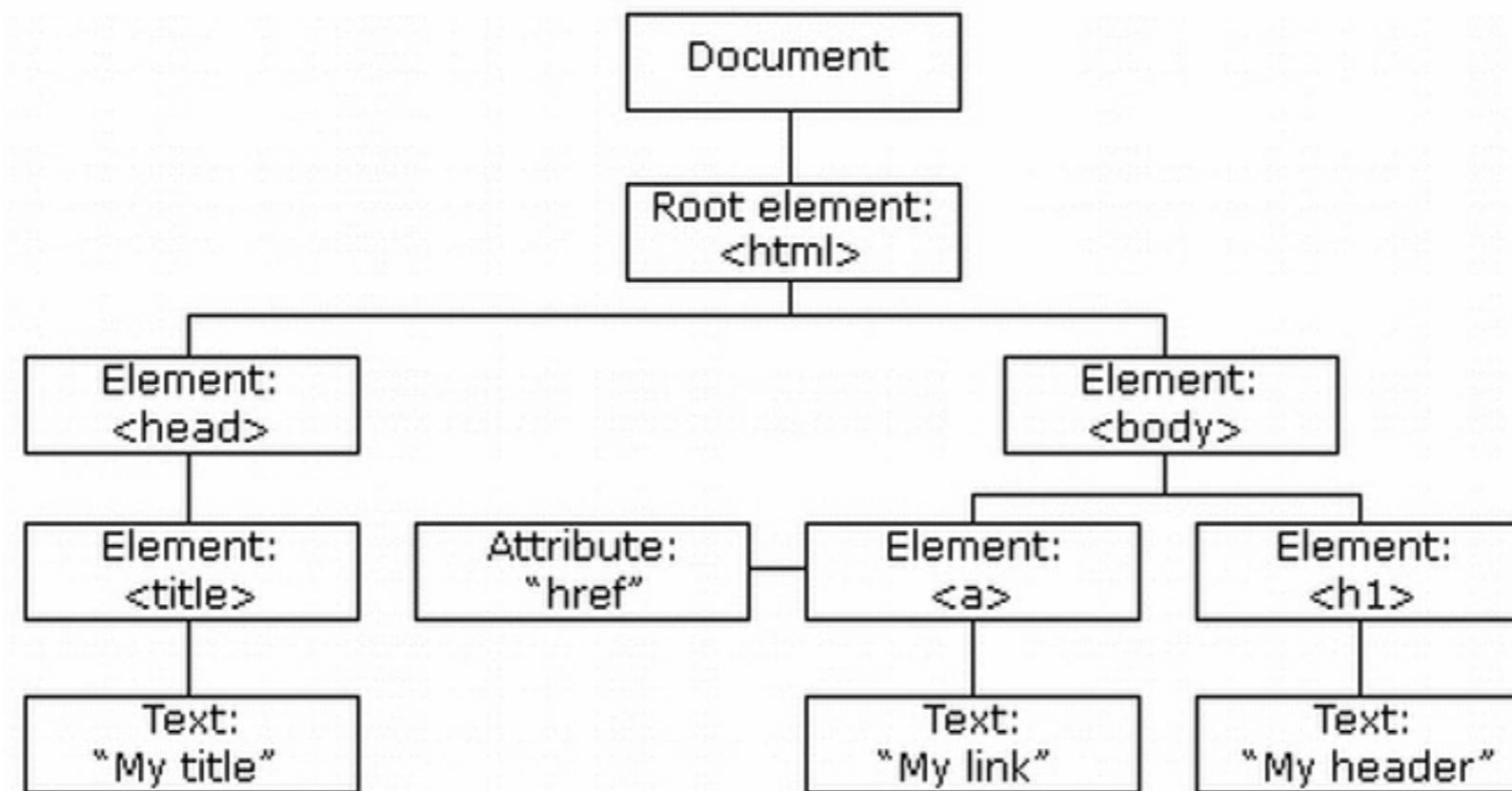
EMERGING SOLUTIONS

WebAssembly or wasm is a new portable, size- and load-time-efficient format suitable for compilation to the web.

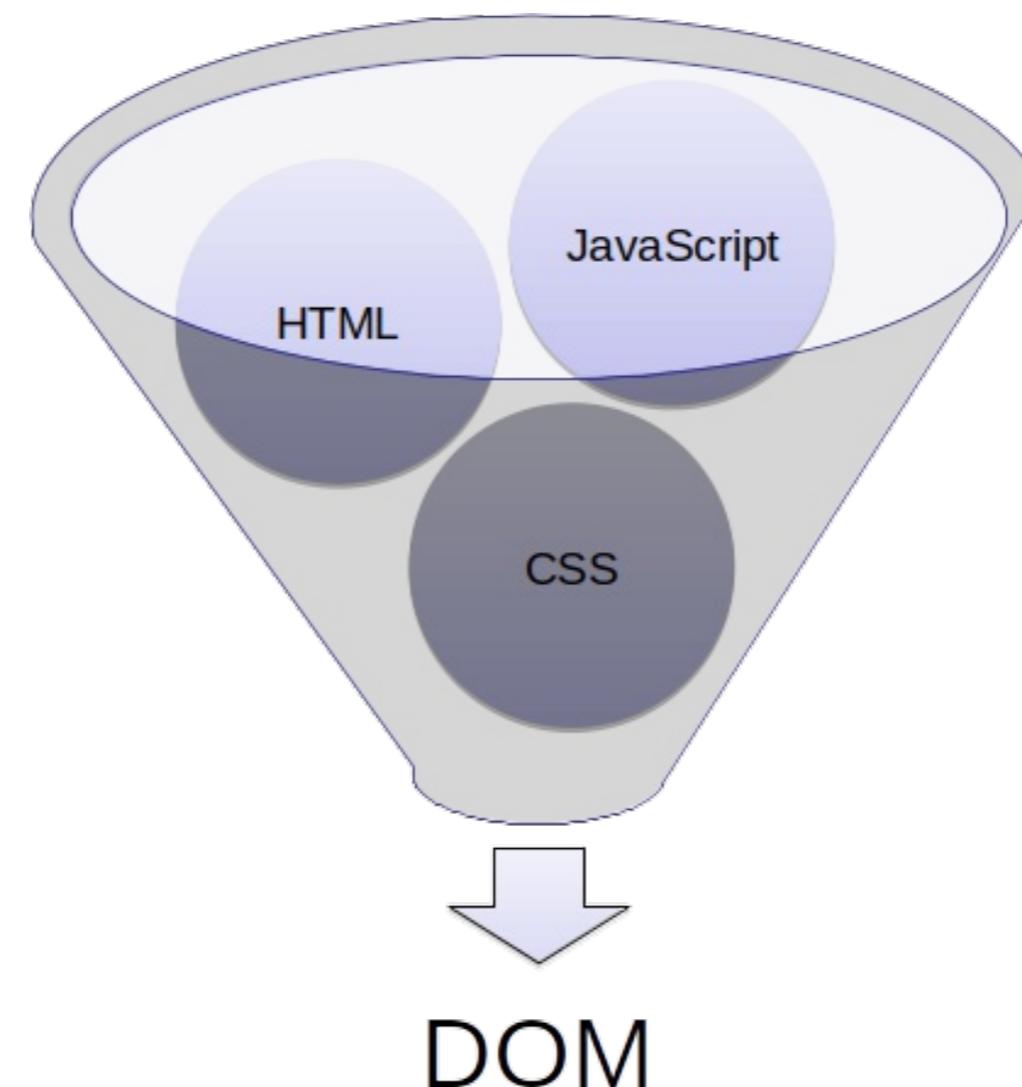
WebAssembly is currently being designed as an open standard by a W3C Community Group that includes representatives from all major browsers.

RUNTIME PLATFORM

- DOM (Document Object Model)
- Document Object Model makes every addressable item in a web application an Object that can be manipulated for color, transparency, position, sound and behaviors.
 - Every HTML Tag is a DOM object

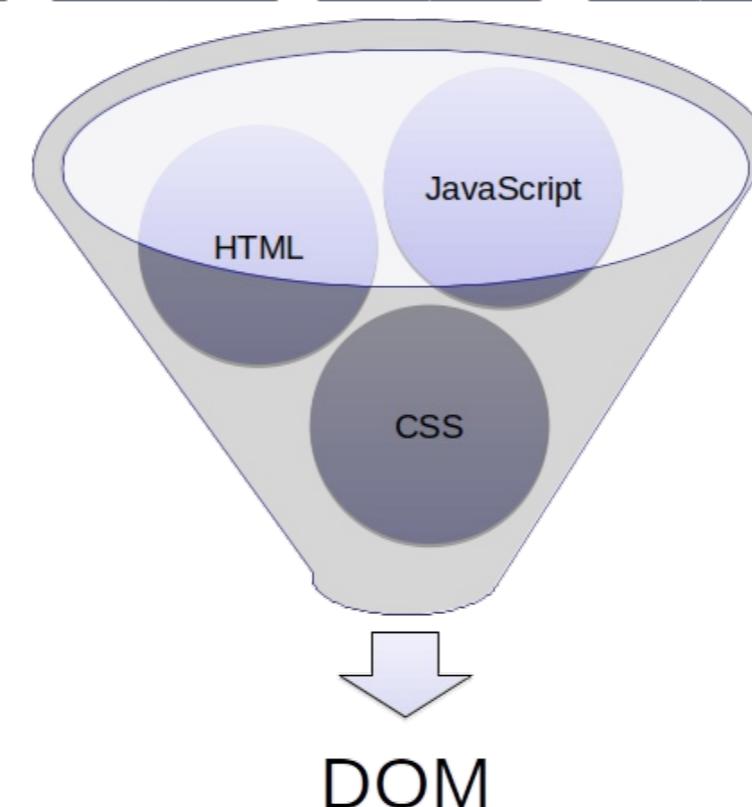
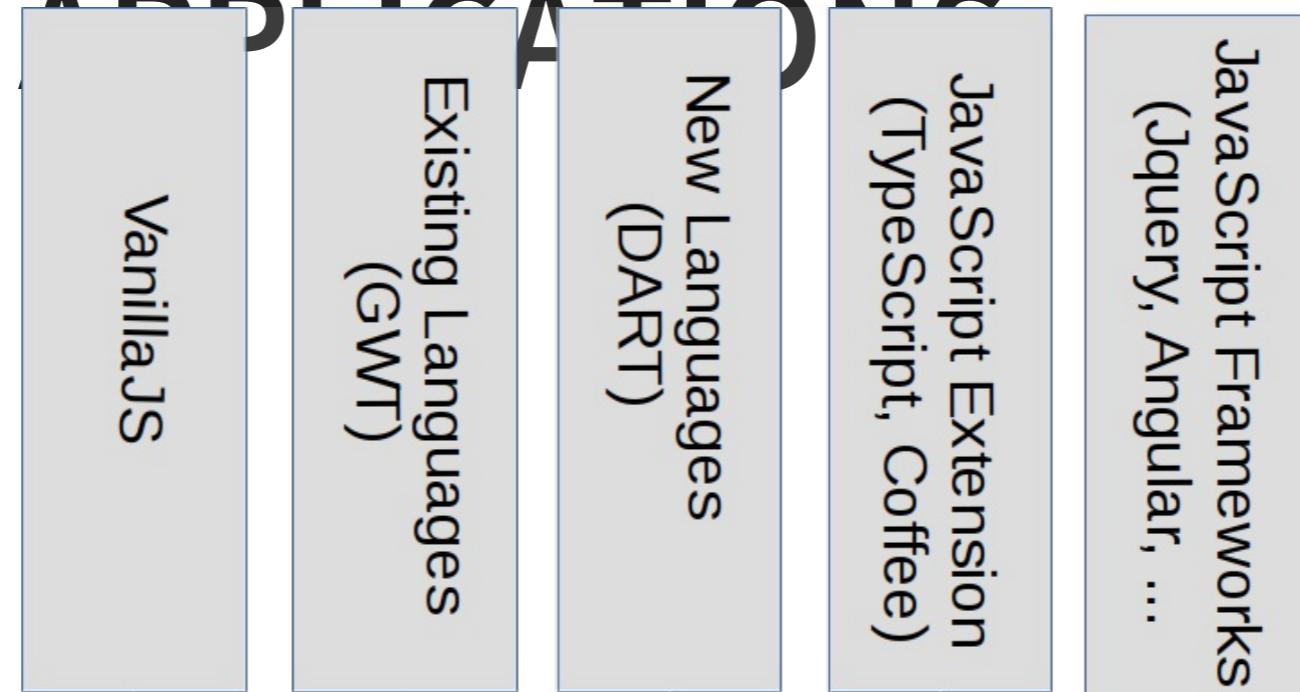


DOM (DOCUMENT OBJECT MODEL)



HOW TO CREATE WEB

APPLICATIONS



JAVASCRIPT

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

JAVASCRIPT

- Mais à quoi ça sert ?
 - Manipulation et animation dynamiques du contenu des pages Web
 - Ouverture, fermeture de fenêtres
 - Communication avec « le » serveur
 - Warning : cross-domain scripting
 - Enregistrement des actions de l'utilisateur (!)
 - Réaction aux évènements utilisateur
 - Sauvegarde de données...

DÉFINITION

JavaScript est un langage de programmation orienté objet développé par NETSCAPE dans les années 90, et s'appelait à l'origine LiveScript.

- Il fut adopté par la firme SUN (qui est à l'origine du langage JAVA) en 1995.
- Le JavaScript est une extension du langage HTML.

POPULARITÉ DE JAVASCRIPT (1)

- JavaScript, le langage de programmation populaire avec la plus importante progression (mais pas le plus grand nombre de lignes de code déployées)
- JavaScript langage de l'année 2014 et 2015 selon l'Index TIOBE (nombre de développeurs + nombre de tutoriaux + nombre de vendeurs tiers, sur la base des recherches dans les moteurs Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube et Baidu)



POPULARITÉ DE JAVASCRIPT (2)



TIOBE Index for January 2015

January Headline: JavaScript programming language of 2014!

After all these years, JavaScript has finally become TIOBE's language of the year. It was a close finish. Swift and R appeared to be the main candidates for the title but due to a deep fall of Objective-C this month, a lot of other languages took advantage of this and surpassed these two candidates at the last moment.

<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

POPULARITÉ DE JAVASCRIPT (3)

- Autres indicateurs de popularité :
 - GitHub (plus nombre de dépôts de source avec JavaScript et plus projets JavaScript les plus populaires)
 - <http://adambard.com/blog/top-github-languages-2014/>
 - <http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/>
 - <http://www.thoughtworks.com/radar/>
 - <http://pypl.github.io/PYPL.html>

GitHub Explore Features Enterprise Blog Sign up Sign in

Search stars:>1 Search

We've found 895,443 repository results Sort: Most stars ▾

Repositories	895,441
Code	
Issues	
Users	

Languages

JavaScript	173,591
Python	95,783
Ruby	91,121
Java	79,789
PHP	74,742
C	44,732
C++	36,032
Objective-C	33,058
Shell	24,476
C#	22,959

twbs/bootstrap CSS ★ 76,039 ⚡ 28,965
The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web.
Updated 3 hours ago

joyent/node JavaScript ★ 33,963 ⚡ 7,603
evented I/O for v8 javascript
Updated 6 days ago

angular/angular.js JavaScript ★ 33,544 ⚡ 13,261
HTML enhanced for web apps
Updated 15 hours ago

mbostock/d3 JavaScript ★ 32,977 ⚡ 8,068
A JavaScript visualization library for HTML and SVG.
Updated 7 days ago

jquery/jquery JavaScript ★ 32,950 ⚡ 7,748
jQuery JavaScript Library
Updated 4 days ago

[Advanced search](#) [Cheat sheet](#)

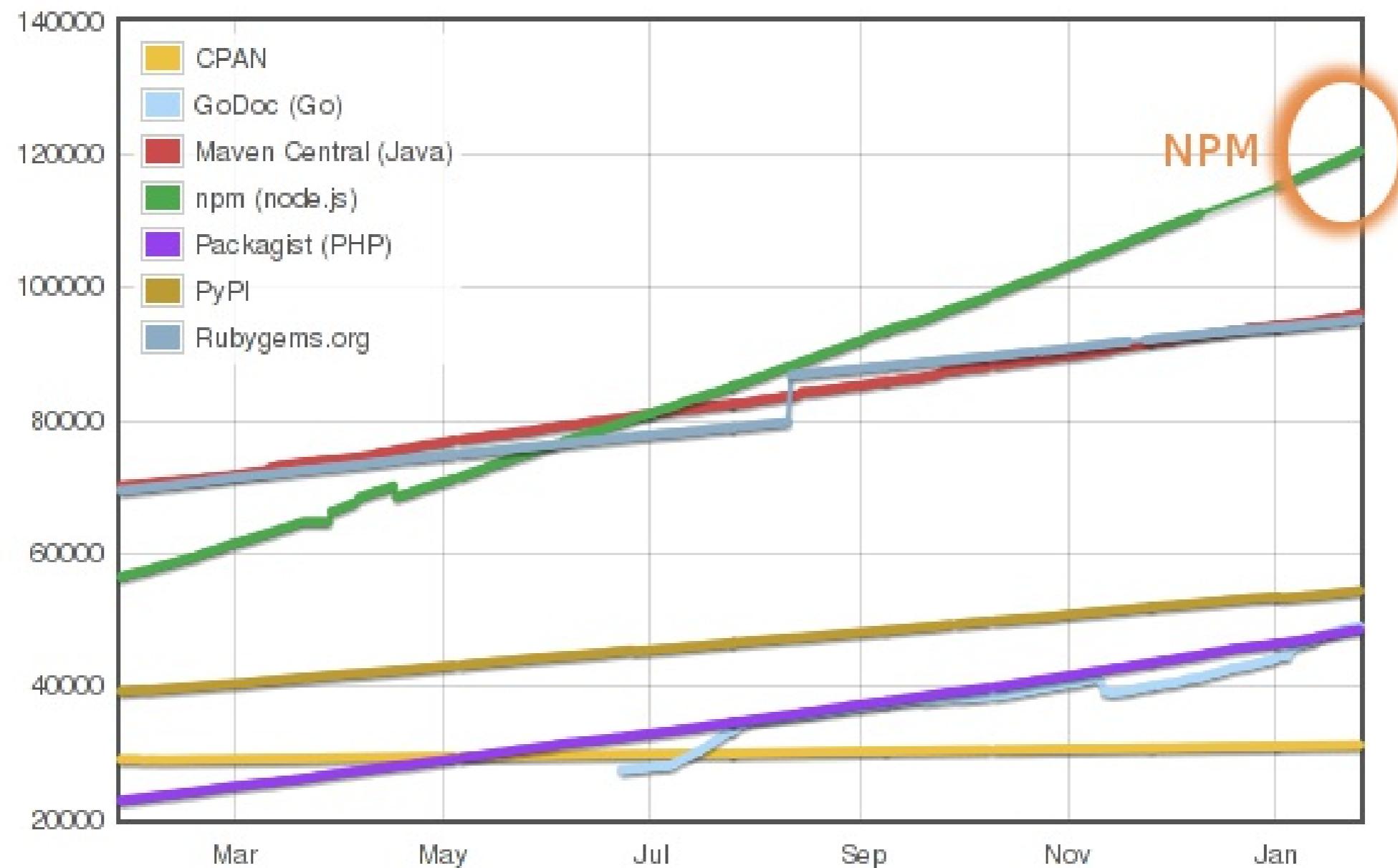
[https://github.com/search?
q=stars%3A%3E1&s=stars&type=Repositories](https://github.com/search?q=stars%3A%3E1&s=stars&type=Repositories)

POPULARITÉ DE JAVASCRIPT (4)

- JavaScript présent dans tous les navigateurs web avec maintenant des implémentations à jour
- ECMAScript 6 disponible et implémentation progressive dans les différents moteurs
- Node.js et NPM à la base du nouveau succès de JavaScript (après avoir été le langage le plus détesté)
- Le plus de modules tiers disponibles grâce à NPM et GitHub. Il existe un module JavaScript existant pour quasiment chaque besoin, ou bien il est en train d'être écrit !

POPULARITÉ DE JAVASCRIPT (5)

<http://modulecounts.com/>



NODE.JS CHOISI PAR LES ENTREPRISES DU WEB

- Plus de téléchargements des modules NPM durant la semaine que le week-end et les vacances
- Les modules JavaScript sont plus utilisés et développés dans le cadre des entreprises que dans le cadre des loisirs
- Node.js utilisé en production par Dow Jones, eBay, Walmart, PayPal, LinkedIn, Yahoo!, The New York Times, Über, Airbnb, Groupon, SAP, Rakuten, TF1, Google
- Node.js supporté officiellement par Oracle
(<http://nodejs.org/industry/>)

D'AUTRES CHOIX

- D'autres acteurs n'utilisent pas Node.js côté serveur :
 - Chez Twitter c'est du Java (après avoir débuté avec du Ruby On Rails)
 - Chez Facebook c'est du Hack (un nouveau PHP)
 - Chez GitHub c'est du Ruby On Rails et du Erlang
 - Google utilise d'autres langages que JavaScript et d'autres plateformes que Node.js : notamment Go (en remplacement de C++)

UBIQUITÉ

- On peut utiliser JavaScript :
 - Sur navigateur pour sites web et web apps
 - Sur serveur pour applications web
 - Sur navigateur + serveur pour applications isomorphiques
 - Sur serveur et poste développeur pour scripts/batches systèmes (export CSV, transformation/conversion de données, etc.)
 - Sur serveur et poste développeur pour scripts web (SlimerJS, PhantomJS, etc.)

JAVASCRIPT

- Spécification de référence: ECMAScript-262 (édition 6.0 – 2015)
- Fonctionnalités additionnelles incompatibles
- Caractéristiques :
 - Langage interprété
 - moteur intégré dans les navigateurs
 - Basé objet mais orienté prototype
 - Dynamique (typage, fonctions, code...)
 - Événementiel

ECMASCRIPT 6

- La spécification ECMAScript 6 est aussi appelée ES6, Harmony ou ES.next
- Finalisation de la spécification Juin 2015
- Les différents moteurs JavaScript implémentent déjà progressivement ECMAScript 6 depuis des années en évoluant en même temps que la spec
- Apporte ce qui manquait à JavaScript pour faciliter la vie des développeurs (et qu'on pouvait déjà trouver dans d'autres langages comme Python, Java, etc.) : modules, collections, template strings, iterateurs, générateurs, etc.

JAVASCRIPT

- Orienté “prototype” ???
 - Basé objet mais pas orienté objet
 - On déclare un objet générique, puis héritage
- Dynamique ???
 - Typage, fonctions, code...
 - Pratique mais dangereux...
- Événementiel ???
 - Paradigme de programmation
 - Attente” puis réaction aux actions “utilisateur”

DÉFINITION

Javascript

Langage interprété

Code intégré au HTML

Langage peu typé

Liaisons dynamiques: les références des objets sont vérifiées au chargement

Accessibilité du code

Sûr: ne peut pas écrire sur le disque dur

Java

Langage compilé

Code (applet)

Langage fortement typé

Liaisons statiques: Les objets doivent exister au chargement (compilation)

Confidentialité du code

Sûr: ne peut pas écrire sur le disque dur

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

SYNTAXE

- Intégration du script dans la page

```
*inline*
<script type="text/javascript">....</script>
```

Appel externe avec une réf. comme pour css

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

- Chargement du script
 - Importance du placement dans la page
 - Dans le header (délai)
 - Dans le body, ou en fin de body
 - Cache si appel externe (mieux), compression

SYNTAXE

- Bonnes pratiques
 - Fichier .js externe
- Lecture par événement “onload”
- “Separation of concerns” (Dijkstra)
 - Pour le script principal
 - Utilisation de fonctions
- chargement dynamique (cf plus tard)

SYNTAXE

- Le code JavaScript se trouve dans un fichier ayant pour extension .js
- L'inclusion de ce fichier se fait généralement au début des pages HTML grâce à la balise script:

```
<script type="text/javascript" src="url/nomdufichier.js"> </script>
```

SYNTAXE

- Les structures conditionnelles ou répétitives (boucles)ont la même syntaxe qu'en langage C.
- JavaScript est un langage faiblement typé.

SYNTAXE

PORTÉE DES VARIABLES:

- Une variable déclarée en début de script sera considérée comme étant globale.
- Les variables utilisées/déclarées dans une fonction restent locales à la fonction.

SYNTAX

- The JavaScript syntax is similar to C# and Java
 - Operators (+, *, =, !=, &&, ++, ...)
 - Variables (typeless)
 - Conditional statements (**if**, **else**)
 - Loops (**for**, **while**)
 - Arrays (**my_array[]**) and associative arrays (**my_array['abc']**)
 - Functions (can return value)
 - Function variables (like the C# delegates)

JAVASCRIPT DATA TYPES

- Numbers (integer, floating-point)
- Boolean (true / false)
- String type – string of characters

```
var myName = "You can use both single  
or double quotes for strings";
```

- Arrays

```
var my_array = [1, 5.3, "aaa"];
```

- Associative arrays (hash tables)

```
var my_hash = {a:2, b:3, c:"text"};
```

EVERYTHING IS OBJECT

- Every variable can be considered as object
- For example strings and arrays have member functions:

```
// file: objects.html
var test = "some string";
alert(test[7]);          // shows letter 'r'
alert(test.charAt(5));    // shows letter 's'
alert("test".charAt(1));  // shows letter 'e'
alert("test".substring(1,3)); // shows 'es'
var arr = [1,3,4];
alert (arr.length); // shows 3
arr.push(7); // appends 7 to end of array
alert (arr[3]); // shows 7
```

STRING OPERATIONS

The + operator joins strings (slow)

```
string1 = "lol ";
string2 = "cats";
alert(string1 + string2); // lol cats
```

What is "11" + 11?

```
alert("11" + 11); // 1111
```

Converting string to number:

```
alert(parseInt("11") + 11);    // 22
alert(parseInt("011",8) + 11);  // 20 ???
alert(parseInt("011", 10) + 11); // 22
```

ARRAYS OPERATIONS AND PROPERTIES

Declaring new empty array:

```
var arr1 = new Array(); var arr2 = [];
```

Declaring an array holding few elements:

```
var arr = [1, 2, 3, 4, 5];
```

Appending an element / getting the last element:

```
arr.push(3); var element = arr.pop();
```

Reading the number of elements (array length):

```
arr.length;
```

Finding element's index in the array:

```
arr.indexOf(1);
```

CONDITIONAL STATEMENT (IF)

```
unitPrice = 1.30;
if (quantity > 100) {
    unitPrice = 1.20;
}

/*
Greater than: >
Less than: <
Greater than or equal to: >=
Less than or equal to: <=
Equal: ==
Not equal: !=
*/
```

CONDITIONAL STATEMENT (IF)

The condition may be of Boolean or integer type:

```
var a = 0;
var b = true;
if (typeof(a)=="undefined" || typeof(b)=="undefined") {
    document.write("Variable a or b is undefined.");
}
else if (!a && b) {
    document.write("a==0; b==true;");
} else {
    document.write("a==" + a + "; b==" + b + ";");
}
```

SWITCH STATEMENT

The switch statement works like in C#:

```
switch (variable) {  
    case 1:  
        // do something  
        break;  
    case 'a':  
        // do something else  
        break;  
    case 3.14:  
        // another code  
        break;  
    default:  
        // something completely different  
}
```

LOOPS

Like in C# (for, while, do while)

```
/*
for loop
while loop
do ... while loop
*/
var counter;
for (counter=0; counter<4; counter++) {
    alert(counter);
}
for (var key in haystack);
while (counter < 5) {
    alert(++counter);
}
```

SYNTAXE

- Les fonctions:
 - La syntaxe de définition d'une fonction est:

```
function nomFonction(arg1,arg2,...)
```

- Contrairement au langage C, on ne donne pas le type des arguments ni celui de la valeur de retour éventuelle

SYNTAXE

- Les chaînes de caractères:
 - Une chaîne de caractères est délimitée par des guillemets simples ou doubles.
 - Exemple:

```
texte = "un beau texte"
```

Ou

```
texte = 'un beau texte'
```

SYNTAXE

- Les chaînes de caractères:
 - Pour manipuler des chaînes de caractères, il existe de nombreuses fonctions.
 - Contrairement au langage C, il est possible de comparer deux chaînes de caractères à l'aide de l'opérateur `==`.

FUNCTIONS

- Code structure – splitting code into parts
- Data comes in, processed, result returned

```
function average(a, b, c) {  
    var total;  
    total = a+b+c;  
    return total/3;  
}
```

- Anonymous functions

```
var average = function (a, b, c) { ... }  
var res1 = average(1,2,3);  
var myAvg = average;  
var res2 = myAvg(1,2,3);
```

FUNCTION ARGUMENTS AND RETURN VALUE

- Functions are not required to return a value
- When calling function it is not obligatory to specify all of its arguments
- The function has access to all the arguments passed via arguments array

```
function sum() {  
    var sum = 0;  
    for (var i = 0; i < arguments.length; i++)  
        sum += parseInt(arguments[i]);  
    return sum;  
}  
alert(sum(1, 2, 4));
```

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

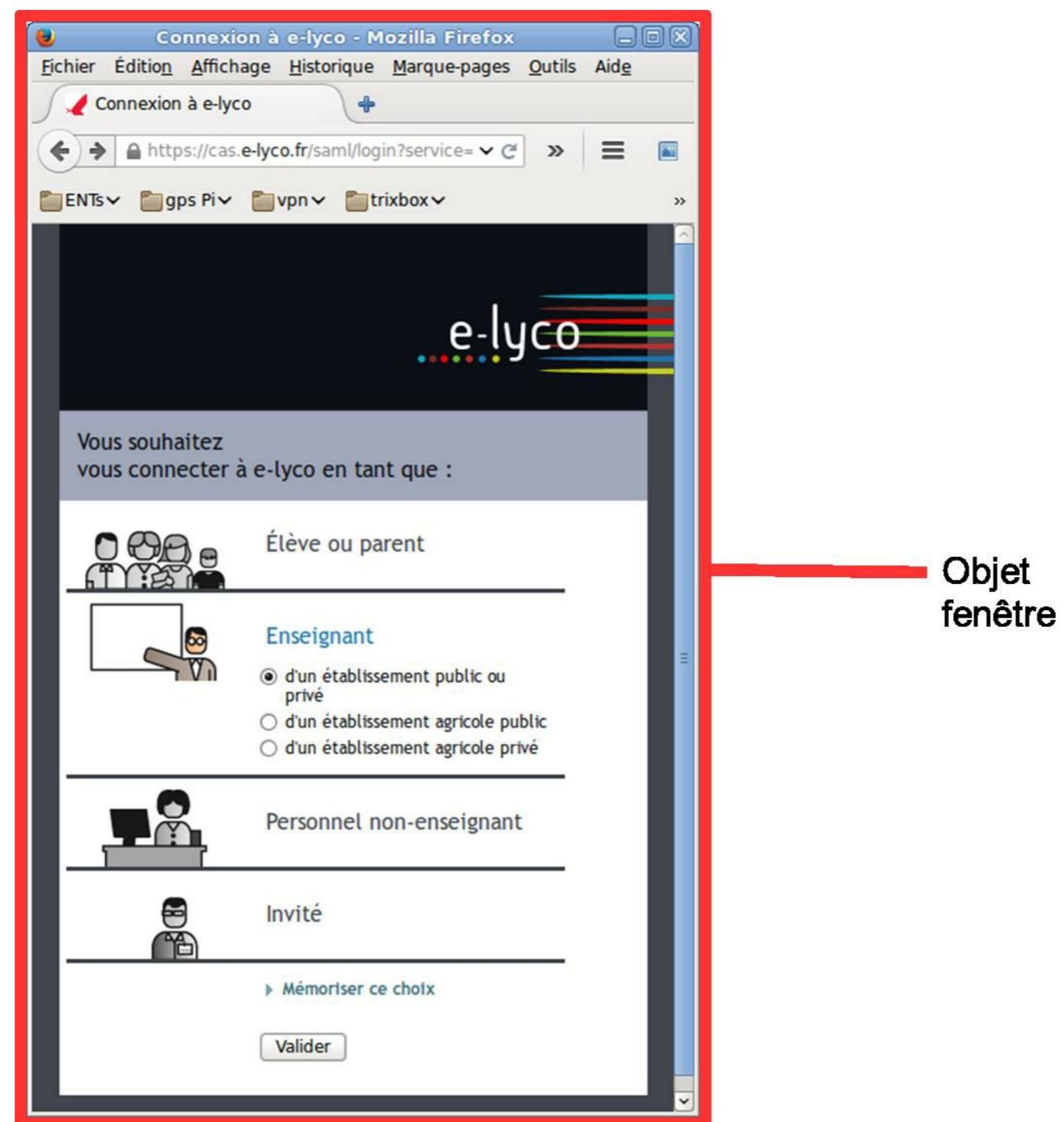
LES OBJETS

- L'utilisation principale de JavaScript est la manipulation des objets d'une page et plus particulièrement des objets des formulaires
- Il est possible d'interagir à deux niveaux:
 - Au niveau du navigateur internet
 - Au niveau de la page affichée dans le navigateur

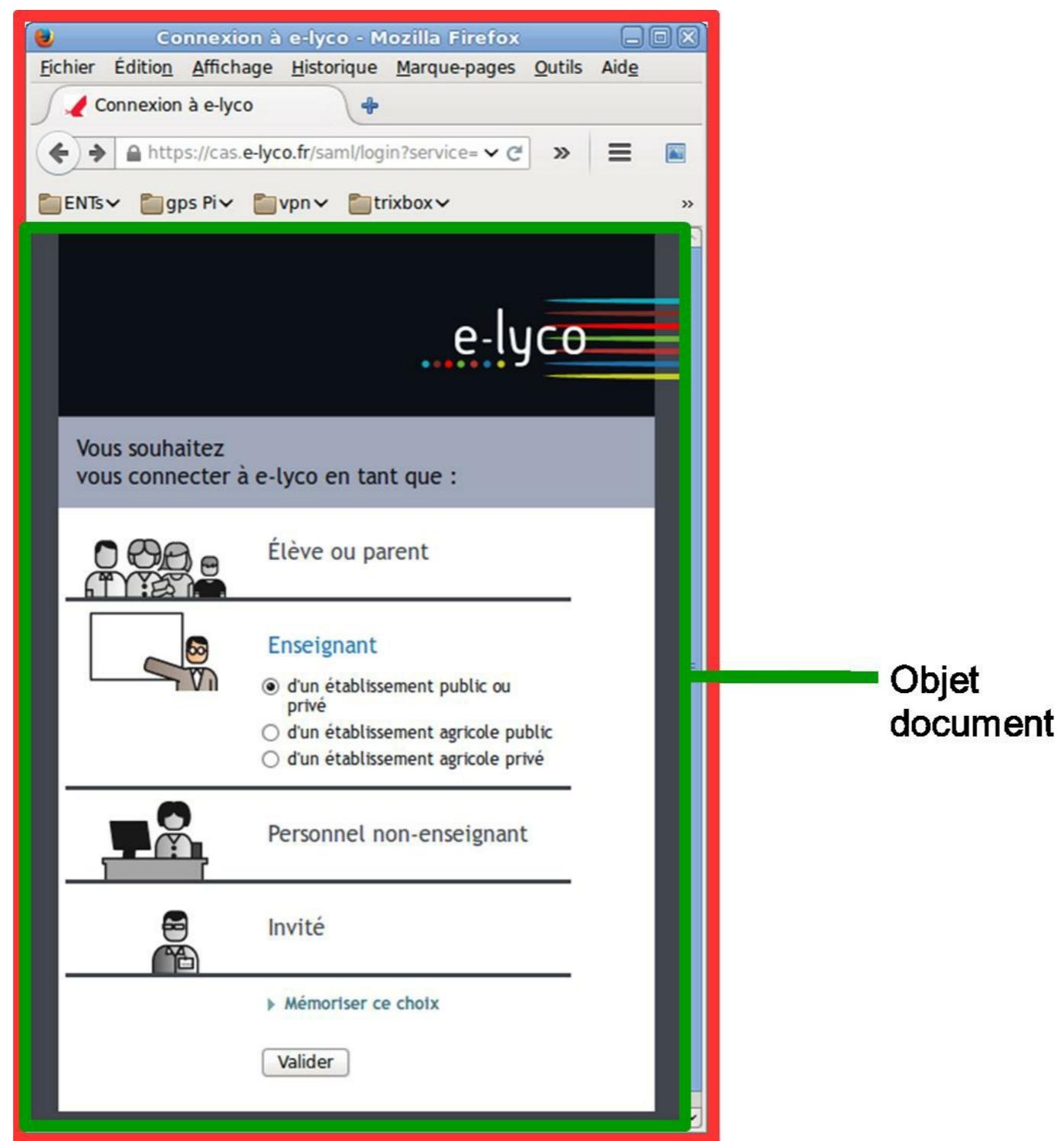
LES OBJETS

- Une page est composée de façon hiérarchique.

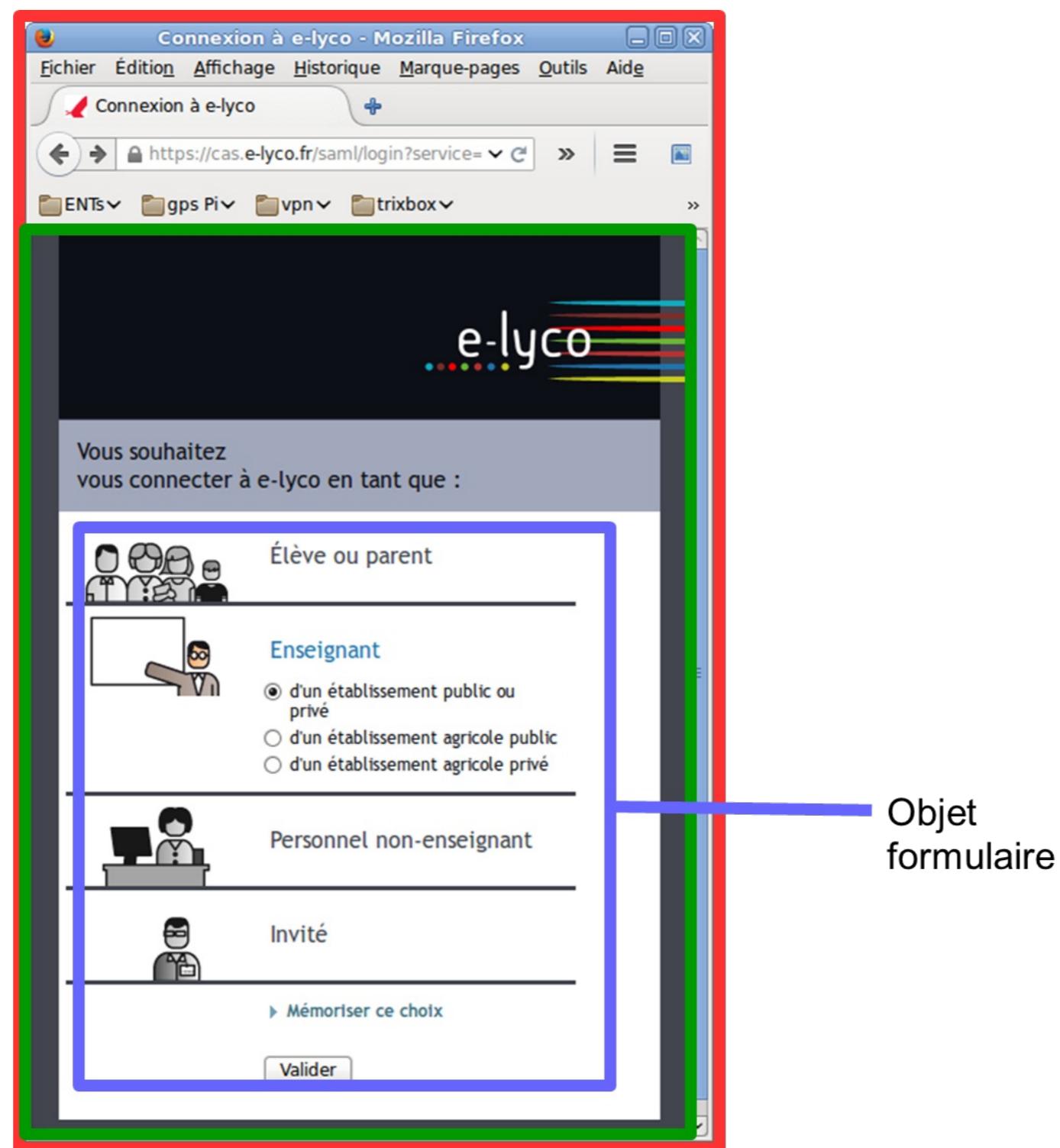
LES OBJETS



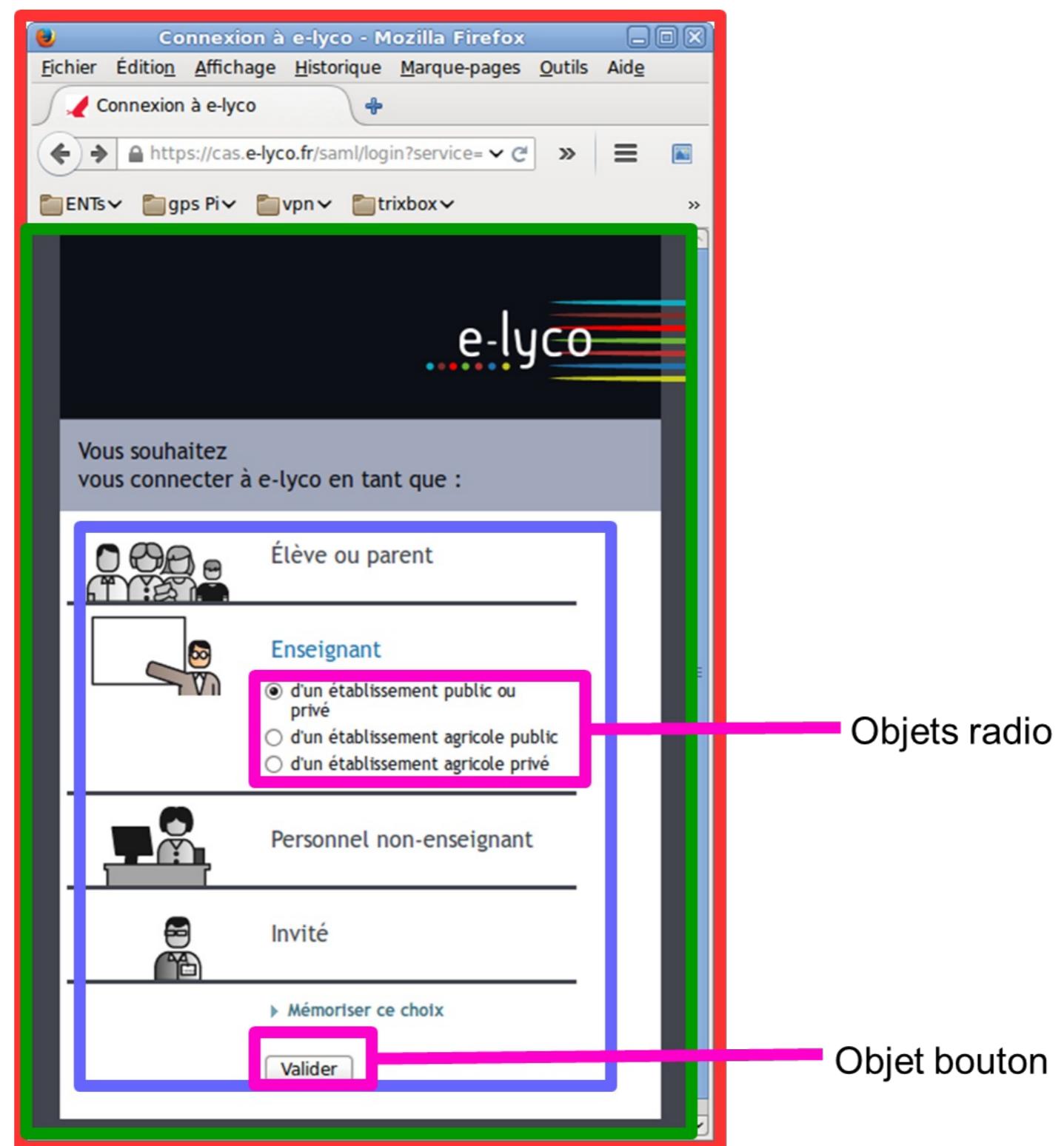
LES OBJETS



LES OBJETS



LES OBJETS



LES OBJETS

- L'accès se fait de façon hiérarchique.

```
window.document.forms["nomDuformulaire"].nomDeLélément
```

- Pour chaque niveau il existe des méthodes et des attributs. Exemple:

```
<html><body>
  <form name="monForm">
    <label for="login">Votre login :</label>
    <input type="text" name="login" id="login" />
  </form>
</body></html>
```

- Si je souhaite avoir la valeur contenue dans le champ login du formulaire, j'utiliserai le code JavaScript suivant:

```
leLogin=window.document.forms["monForm"].login.value;
```

LES OBJETS

- Classes are functions
- Objects are associative arrays
- **this** refers to the owning code
- **new** copies the prototype into new reference

```
var person = new Object();
person.first = "Tim";
person.last = "Scarfe";

person.sayHello = function() {
    console.log("Hello, " + this.first + " " + this.last);
};
person.sayHello();
```

LES OBJETS

```
function User(first, last) {  
    this.first = first;  
    this.last = last;  
  
    this.sayHello = function() {  
        console.log("Hello, " + this.first  
            + " " + this.last);  
    }  
}  
  
var svi = new User("Svi", "Ivanov");  
var alex = new User("Alex", "Ivanova");  
svi.sayHello();  
alex.sayHello();
```

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- **Les évènements**
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

LES ÉVÈNEMENTS

- L'action sur un élément de la page se fait lors d'un événement particulier (clique de souris, changement de la valeur d'un champ, etc)
- Voici quelques évènements possibles pour une page WEB
 - Click (onClick)
 - Load (onLoad)
 - Unload (onUnload)
 - MouseOver (onMouseOver)
 - MouseOut (onMouseOut)
 - Focus (onFocus)
 - Change (onChange)
 - Submit (onSubmit)

LES ÉVÈNEMENTS

- Pour qu'un objet réagisse à un événement, il faut lui associer une fonction de traitement.
- Exemple:

```
<html><body>
<script src="mesFonctions.js" type="text/javascript"></script>
<form name="monForm">
    <label for="login">Votre login :</label>
    <input type="text" name="login" id="login"
    onchange="afficheLogin();">
    <input type="text" name="loginBis" id="loginBis">
</form>
</body></html>
```

- Le fichier mesFonctions.js contient le code suivant:

```
function afficheLogin(){
    window.document.forms["monForm"].loginBis.value =
        window.document.forms["monForm"].login.value;
}
```

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- **Les boîtes de dialogue**
- Accès à un élément quelconque d'une page

LES BOÎTES DE DIALOGUE

- Il existe 3 types de boîtes de dialogue:
 - alert : affiche un message et un bouton ok



- confirm : affiche un message et un bouton ok et annuler



LES BOÎTES DE DIALOGUE

- prompt : affiche une zone de saisie et un bouton ok.



- confirm retournera true ou false selon la réponse (ok ou annuler).
- prompt retournera le message contenu dans la zone de saisie.

JAVASCRIPT

- Définition
- Syntaxe
- Les objets
- Les évènements
- Les boîtes de dialogue
- Accès à un élément quelconque d'une page

ACCÈS À UN ÉLÉMENT QUELCONQUE D'UNE PAGE

- La manière la plus simple pour avoir accès à un élément quelconque d'une page est d'utiliser son identifiant

```
var obj = document.getElementById("idelement") ;
```

- Il est ainsi possible d'avoir des informations et d'agir sur l'élément
 - **innerHTML** : va récupérer/modifier le contenu HTML de l'élément
 - **textContent** : va récupérer/modifier le contenu de l'élément
 - **nodeName** : va récupérer le nom de l'élément

AN EXAMPLE

```
<div id="div1"></div>
<input type="button" value="Vert" onclick="vert();"/>
<input type="button" value="Rouge" onclick="rouge();"/>
```

AN EXAMPLE

```
#div1{  
background-color:#ff0000;  
width:200px;  
height:200px;  
}
```

AN EXAMPLE

```
function vert(){
    document.getElementById('div1').style.backgroundColor='#00ff00';
}
function rouge(){
    document.getElementById('div1').style.backgroundColor='#ff0000';
}
```

JSON

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent *
- Key-value pairs

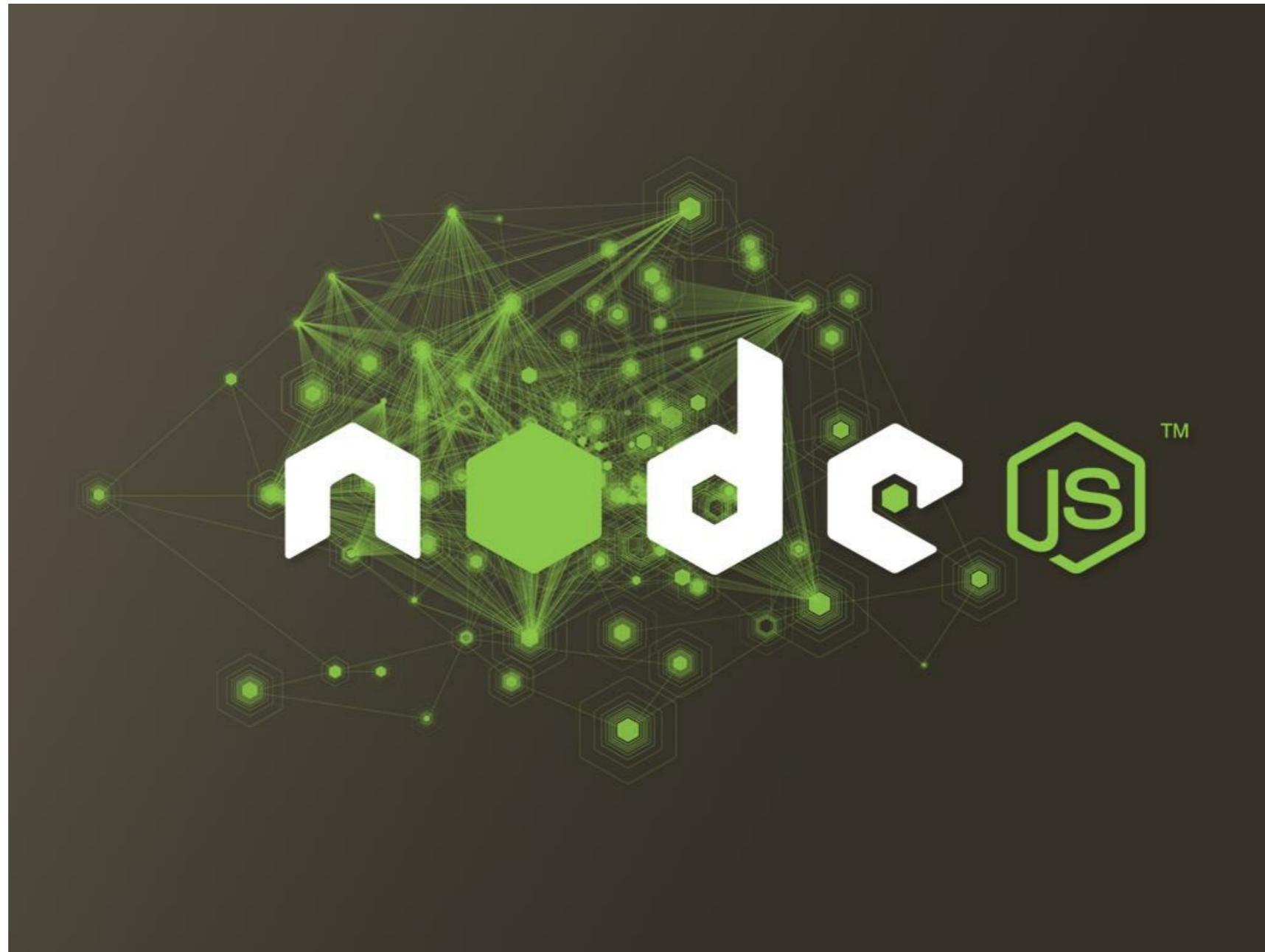
JSON

- JSON uses JavaScript syntax, but the JSON format is text only.
- Text can be read and used as a data format by any programming language.
- The JSON format was originally specified by Douglas Crockford.

JSON SAMPLE

```
{  
  "id": 1,  
  "name": "A green door",  
  "price": 12.50,  
  "tags": ["home", "green"]  
}
```

PRÉSENTATION DE NODEJS



PRÉSENTATION NODE.JS (1)

- Node.js, plateforme événementielle aux entrées-sorties non-bloquantes (asynchrone) pour développer des applications en JavaScript "côté serveur"
- Moteur léger et écologique !
 - (peu gourmand en ressources, CPU, RAM)
- Noyau en JavaScript et C++ par-dessus le moteur JavaScript V8 de Google Chrome
 - (moteur V8 régulièrement mis à jour dans Node.js)
- Logiciel libre à licence permissive (licence MIT) avec la quasi totalité des modules également à licence permissive (licence MIT ou BSD)

PRÉSENTATION NODE.JS (2)

- Les méthodes de l'API sont asynchrones par défaut
- Le streaming au cœur de l'API
- Manipule nativement le JSON
- Sait gérer nativement toutes les requêtes HTTP
 - (pour réaliser l'équivalent d'appels AJAX côté serveur)
- Développement aisément d'applications (grande agilité)
- Les applications réalisées sont extrêmement rapides
- Mettre frontal Nginx (basé événements) à la place de Apache (basé processus), sinon performances bridées

UTILISATIONS DE NODE.JS

- Node.js excellent pour :
 - API web (REST, services JSON)
 - Sites web (notamment streaming de bout en bout : par exemple streaming de fichiers depuis base de données puis streaming de ces fichiers dans archive ZIP puis streaming HTTP vers navigateur web)
 - Très nombreuses connexions et montée en charge (gère des milliers de connexions simultanées à faible charge sur un seul processus)
 - Communications temps réel (sockets, polling, etc.)

LIMITATIONS

- Node.js inadapté pour :
 - Usage intensif du CPU : 3D, transcodage vidéo, et en général tout calcul mathématique non fourni par l'API du cœur (les fonctions cryptographiques font, elles, partie du cœur et sont rapides, en C++)
 - Mise en mémoire de grande quantité de données (pas plus de 1Go augmentable à 1,7Go max). Mais de par programmation événementielle cette limite est moins gênante qu'elle le serait avec d'autres langages comme Java par exemple (qui consomme une très grande quantité de mémoire).

PHILOSOPHIE DE NODE.JS

- http://substack.net/node_aesthetic
- La philosophie de Node.js :
 - Bibliothèque centrale minimale
 - Programmabilité et composabilité maximale
 - Réutilisabilité radicale

Agilité maximale

Innovation rapide car décentralisée

CALLBACK NODE.JS NORMALISÉ

- Un callback normalisé doit être passé comme argument à toutes les méthodes asynchrones

```
function(err, res) {  
  if (err) {  
    console.error("Failed:", err);  
    return;  
  }  
  console.info("Success with produced result:", res);  
}
```

Mais on peut utiliser des promesses en « promisifiant » les méthodes de Node.js (cf. infra)

RECHERCHE DE MODULES

- <https://www.npmjs.com/>
- <https://github.com/>
 - Comparer la popularité grâce :
 - aux étoiles ($>10 \rightarrow$ c'est bien)
 - au nombre des contributeurs ($>3 \rightarrow$ c'est bien)

MODULES SERVEUR UTILES (1)

- **Tracer** : logs serveur
- **Commander** : Gestion de la ligne de commande
- **Node-convict** : lecture / enregistrement de configuration
- **Express** : cadriel web/REST (web framework)
- **Koa** : cadriel web/REST (web framework) utilisant les promesses et les générateurs

MODULES SERVEUR UTILES (2)

- **Bookshelf.js** : ORM fonctionnant avec PostgreSQL, MySQL, SQLite3, Oracle
- **Ldapjs** : Client et serveur LDAP
- **Nodemailer** : Envoi de courriels
- **Archiver** : Génération d'archives ZIP, TAR en streaming
- **Fast-csv** : création / lecture de CSV

MODULES CLIENT ET SERVEUR UTILES

- **Create-error** : création nouveaux types d'erreur
- **Bluebird** : Module pour promesses, ultra-rapide, avec plein d'utilitaires
- **Moment** : affichage, manipulations dates et durées
- **Validator.js** : Validation et nettoyage de chaînes de caractères
- **Nunjucks** : template engine proposant layouts et streaming (développé par Mozilla)
- **Base64url** : encode, decode, escape, unescape

MODULES CLIENT UTILES

- **Loglevel** : logs en console débraillables à l'exécution et compatible avec tout navigateur web (idéal pour assurer du débogage et du support)
- **Dexie** : API IndexedDB avec des promesses, permettant notamment la recherche plein texte

BONNES PRATIQUES AVEC MODULES NPM (1)

- Trouvez les modules pouvant être mis à jour

```
npm install -prefix ~/ -g npm-check-updates  
npm-check-updates
```

- Mettez à jour tous les modules (+ validez par tests) :

```
npm-check-updates -u
```

- Utilisez des numéros de version strictes de module (pas de ^ ou ~)

```
npm install loglevel --save --save-exact  
npm install jshint --save-dev --save-exact
```

BONNES PRATIQUES AVEC MODULES NPM (2)

- Factorisez les dépendances communes :

```
npm dedupe
```
- Définissez strictement l'arbre de dépendances :

```
npm shrinkwrap  
git add npm-shrinkwrap.json
```

BONNES PRATIQUES AVEC MODULES NPM (3)

- Vérifiez les licences de tous les modules utilisés (y compris les licences de toutes les dépendances)

```
npm install -prefix ~/ -g licence-checker  
license-checker  
license-checker | grep licences | sort | uniq
```

On trouve essentiellement des licences :

- MIT
- BSD
- Apache

FACTORIZER LE CODE AVEC MODULES NPM (1)

- Avec NPM on peut utiliser (installer) un module présent dans le registre [NPM](#) mais pas que ...
- Avec NPM on peut aussi utiliser (installer) un module présent sur un dépôt Git public, comme GitHub, ou privé



FACTORISER LE CODE AVEC MODULES NPM (2)

- Forme des URL de module sur dépôt Git :

```
git://github.com/user/project.git#commit-ish  
git+ssh://user@hostname:project.git#commit-ish  
git+ssh://user@hostname/project.git#commit-ish  
git+http://user@hostname/project/blah.git#commit-ish  
git+https://user@hostname/project/blah.git#commit-ish
```

- Exemple pour utiliser (installer) un module privé :

```
npm install git+ssh://git@git.entreprise.fr:module1.git --save-dev -save-exact
```

- Suppression du besoin d'utilisation de git-submodules dans beaucoup de cas

MODULES COMMONJS

- NPM et Node.js utilisent le standard CommonJS pour gérer les modules. La spec modules ES6 en est inspirée. Exemple de module fourni (repository.js) et de module importé (behavior.js) :

```
// repository.js
exports.getDocsCount = getDocsCount;
function getDocsCount() {
    return db.documents.count();
}
// behavior.js
var repository = require('./repository');
repository.getDocsCount();
```

MODULES COMMONJS POUR CODE CLIENT (NAVIGATEUR)

- Grâce à Browserify on peut utiliser les modules CommonJS avec tous les avantages que cela représente au niveau du code JavaScript déployé sur le client (navigateur)
- On peut ainsi disposer d'environ 80% des 120.000 modules NPM disponibles à ce jour côté client.
- Les développeurs peuvent utiliser les mêmes modules qu'ils connaissent côté client que côté serveur (et inversement :-))
- Pour mettre en pratique lire absolument :
<https://github.com/substack/browserify-handbook>

AUTRES GESTIONNAIRES DE MODULES (BOWER, YEOMAN)

- Il y a actuellement plusieurs gestionnaires de modules JavaScript :
 - NPM
 - Bower
 - Yarn
- Bower et Yeoman sont moins puissants que NPM et sont généralement utilisés par les développeurs connaissant peu (la puissance de) NPM et Browserify
- NPM a vocation à devenir l'unique gestionnaire de modules JavaScript. Mais Yarn est très populaire

OUTILS DE CONSTRUCTION D'APPLICATIONS

- **Grunt** : Outil très célèbre, basé sur la configuration, pour construire des applications web.
 - Facile à comprendre et à utiliser.
 - **Désavantage** : ne tire pas partie du streaming, fichiers lus et modifiés sur disque à chaque tâche.
- **Gulp** : Outil de construction basé sur le code, plutôt que la configuration, et utilisant le streaming
- **NPM** : NPM seul peut aussi servir à construire des applications (minification, etc.)

PROMESSES

- On peut gérer le code asynchrone avec des callbacks.
C'est la pratique par défaut dans Node.js
- Mais peut devenir très pénible quand plusieurs actions asynchrones s'enchaînent ou se déroulent en parallèle
 - La meilleure solution → les promesses (aussi appelées engagements ou futures)
 - En anglais on parle de promises ou de thenables

PROMESSES

Exemple de code asynchrone classique :

```
fs.readFile('file.json', function(err, val) {  
  if (err) {  
    console.error("Unable to read file");  
  } else {  
    try {  
      val = JSON.parse(val);  
      console.log(val.success);  
    } catch( e ) {  
      console.error("Invalid JSON in file");  
    }  
  }  
});
```

PROMESSES

```
var P = require('bluebird');
var fs = P.promisifyAll(require('fs'));

fs.readFileAsync('file.json').then(JSON.parse)
  .then(function(val) {
    console.log(val.success);
  })
  .catch(SyntaxError, function(err) {
    console.error("Invalid JSON in file", err);
  })
  .catch(function(err) {
    console.error("Unable to read file", err) ;
  });
});
```

VEILLE SÉCURITÉ ET CORRECTIFS

- Audit permanent de TOUS les modules JavaScript sur NPM et émissions d'alertes :
 - <http://nodesecurity.io/>
- Nouvelles versions de Node.js avec correctifs de sécurité :
 - <http://nodejs.org/>

NSP

```
npm install --prefix ~/ -g nsp
nsp audit-package
  Name   Installed  Patched  Vulnerable  Dependency
  connect  2.7.5  >=2.8.1  nodesecurity-jobs > kue > express
nsp audit-shrinkwrap
  Name   Installed  Patched  Vulnerable  Dependency
  connect  2.7.5  >=2.8.1  nodesecurity-jobs > kue > express
```

MON PREMIER PROGRAMME EN NODE.JS

```
// bonjour.js
console.log('Bonjour');
```

```
$ node bonjour.js
```

TOOLS FOR JAVASCRIPT DEVELOPMENT

**UNLEASH THE POWER OF JAVASCRIPT
TOOLING**

TABLE OF CONTENTS

- JavaScript Editors
 - JetBrains WebStorm
 - Sublime Text 2/3
 - Atom
 - Visual Code Editor
- Project tools
 - Package Management: NPM/Yarn & Bower
 - Scaffolding: Yeoman
 - Task Automation: Grunt && Gulp

JAVASCRIPT EDITORS

JETBRAINS WEBSTORM

- JavaScript IDE suitable for both client-side JavaScript and server (Node.js) development
 - Livereload
 - Build
 - Code highlighting/Intellisense for:
 - LESS/SASS/Stylus CSS preprocessors
 - CoffeeScript/TypeScript JavaScript preprocessors
 - AngularJS directives
 - Emmet Coding
 - Has only 30-days-long free trial

SUBLIME TEXT 2/3

- Editor for scripting languages
 - JavaScript, PHP, Python, Ruby, etc...
 - Basic code highlighting with vanilla installation
 - Endless number of configurable plugins
 - Free product
 - Paid only if used commercially
 - With plugins support for:
 - LESS/SASS/Stylus
 - CoffeeScript/TypeScript

ATOM: A HACKABLE TEXT EDITOR

- Cross-platform editing
- Built-in package manager
- Smart autocomplete
- File system browser
- Under the hood
 - Atom is a desktop application built with HTML, JavaScript, CSS, and Node.js integration. It runs on Electron, a framework for building cross platform apps using web technologies.
- Open source
 - support by Github

VISUAL CODE EDITOR

- Smart autocompletion with IntelliSense.
 - smart completions based on variable types, function definitions, and imported modules.
- Debugging support
- Git commands built-in
- Extensible and customizable
- great integration of TypeScript and Angular 2
- Open source
 - support by Microsoft

PROJECT TOOLS

NO MATTER THE EDITOR

PROJECT TOOLS

- NPM, Yarn & Bower
 - Install Node.js packages or client libraries
- Grunt & Gulp
 - Tasks runner
 - Create different tasks for build/development/test cases
- Yeoman
 - Scaffolding of applications
 - One-line-of-code to create a project template with views/routes/modules/etc...

PACKAGE MANAGEMENT

NPM, YARN & BOWER

PACKAGE MANAGEMENT: NPM

- Node.js Package Management (NPM)
- Package manager for Node.js modules

```
npm init #in CMD (Win) or Terminal (MAC/Linux)
```

- Initializes an empty Node.js project with package.json file

```
npm init
//enter package details
name: "NPM demos"
version: 0.0.1
description: "Demos for the NPM package management"
entry point: main.js
test command: test
git repository: http://github.com/user/repository-name
keywords: npm, package management
author: doncho.minkov@telerik.com
license: BSD-2-Clause
```

PACKAGE MANAGEMENT: NPM

- Installing modules

```
npm install package-name [--save][--save-dev][--save-optional]  
# Installs a package to the Node.js project
```

-S, --save: Package will appear in your dependencies in package.json

-D, --save-dev: Package will appear in your devDependencies

-O, --save-optional: Package will appear in your optionalDependencies.

```
npm install express --save-dev
```

Before running the project

```
npm install ## Installs all missing packages from package.json
```

PACKAGE MANAGEMENT: BOWER

- Bower is a package management tool for installing client-side JavaScript libraries
 - Like jQuery, KendoUI, AngularJS, etc...
 - It is a Node.js package and should be installed first

```
npm install -g bower  
bower init # in CMD (Win) or Terminal (Mac/Linux)
```

- Asks for pretty much the same details as *\$ npm init*
- Creates bower.json file to manage libraries

PACKAGE MANAGEMENT: BOWER

- Searching for libraries

```
bower search kendo
```

```
Search results:
kendo-ui git://github.co
angular-kendo-ui git://g
kendo-backbone git://git
knockout-kendo git://git
kendo-global git://githu
```

- Installing libraries

```
bower install kendo-ui
```

```
bower cached          git://gith
bower validate        2014.1.318
i.git#*
bower cached          git://gith
bower validate        2.0.3 again
.0.3
bower resolution     Removed un
bower install         kendo-ui#2
bower install         jquery#2.0
kendo-ui#2014.1.318 bower_com
└── jquery#2.0.3
```

GRUNT & GULP

TASKS RUNNER

GRUNT

- Grunt is a Node.js task runner
 - It can runs different tasks, based on configuration
 - Tasks can be:
 - Concat and minify JavaScript/CSS files
 - Compile SASS/LESS/Stylus
 - Run jshint, csshint
 - Run Unit Tests
 - Deploy to Git, Cloud, etc...
 - And many many more

GRUNT

- Why use a task runner?
 - Task runners gives us automation, even for different profiles:

DEVELOPMENT	TEST	BUILD
jshint	jshint	jshint
stylus	stylus	stylus
csshint	csshint	csshint
connect	mocha	concat
watch		uglify
		copy
		minify

CONFIGURING GRUNT

- To configure grunt, create a Gruntfile.js file in the root directory of your application
 - It is plain-old Node.js
 - Grunt is configured programmatically
 - Create an module that exports a single function with one parameter – the grunt object

```
module.exports = function (grunt) {  
  //configure grunt  
};
```

CONFIGURING GRUNT (2)

- All the configuration is done inside the module
- First execute the *grunt.initConfig()* method and pass it the configuration

```
module.exports = function (grunt) {  
  grunt.initConfig({  
    ...  
  });  
};
```

CONFIGURING GRUNT PLUGINS

- To use a plugin in grunt:
 - Install the plugin

```
npm install grunt-contrib-jshint --save-dev
```

- Load the plugin

```
//inside the grunt module
grunt.loadNpmTasks('grunt-contrib-jshint');
```

- Configure the plugin

```
//inside the grunt.initConfig()
grunt.initConfig({
  jshint: {
    app: ['Gruntfile.js',
          'path/to/scripts/**/*.*']
  }
});
```

GRUNT PLUGINS

GRUNT PLUGINS: BUILD

- *jshint* (*grunt-contrib-jshint*)
 - Runs jshint for specified files
- *csslint* (*grunt-contrib-csslint*)
 - Runs csslint for specified files
- *stylus* (*grunt-contrib-stylus*)
 - Compiles STYL files into CSS files
- *uglify* (*grunt-contrib-uglify*)
 - Minifies configured JavaScript files
- *concat* (*grunt-contrib-concat*)
 - Concats configured JavaScript files

GRUNT PLUGINS: DEVELOPMENT

- *connect (grunt-contrib-connect)*
 - Starts a Web server on a given port and host
- *watch (grunt-contrib-watch)*
 - Watches for changes to configured files
 - Can run other tasks on file changed

GULP: THE STREAMING BUILD SYSTEM

Streams come to us from the earliest days of unix and have proven themselves over the decades as a dependable way to compose large systems out of small components that do one thing well.

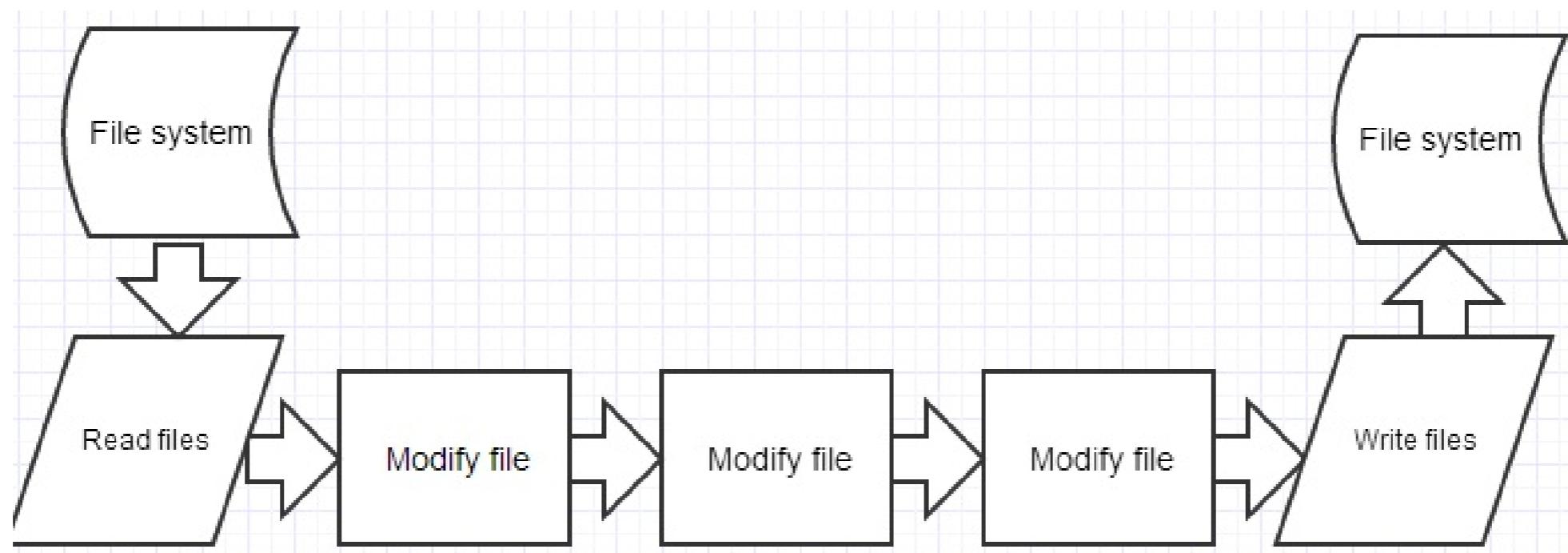
You can then plug the output of one stream to the input of another and use libraries that operate abstractly on streams to institute higher-level flow control.

WHY STREAMS?

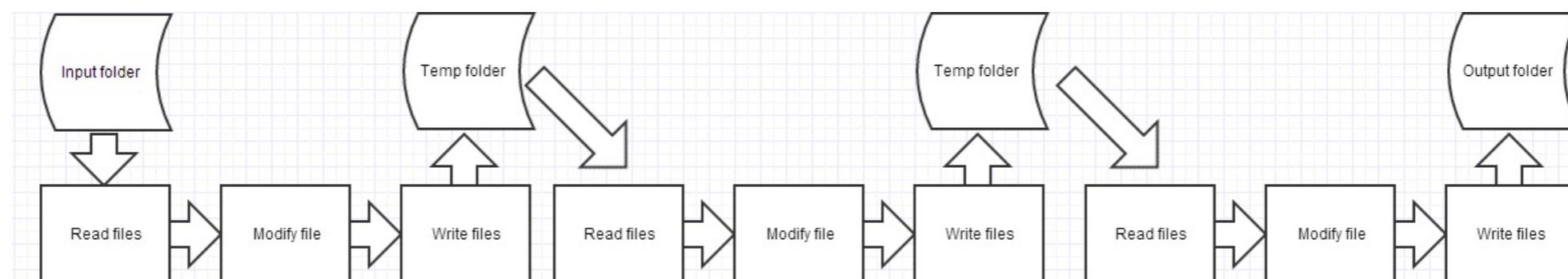
PICTURE A BUILD SYSTEM IN YOUR HEAD.

- It should take in files, modify them, and output the new ones

YOU PICTURED THIS



YOU DIDN'T PICTURED THIS



WHAT IS WRONG WITH GRUNT

- Plugins do multiple things
 - Want a banner? Use the javascript minifier
 - Plugins do things that don't need to be plugins
 - Need to run your tests? Use a plugin
- Grunt config format is a mess that tries to do everything
- Not idiomatic with "the node way"
- Headache of temp files/folders due to bad flow control

*Your build system should empower not
impede*

*It should only manipulate files - let other
libraries handle the rest.*

SAMPLE GRUNTFILE

```
module.exports = function(grunt) {  
  
  grunt.initConfig({  
    pkg: grunt.file.readJSON('package.json'),  
    concat: {  
      options: {  
        separator: ';'  
      },  
      dist: {  
        src: ['src/**/*.{js,html}'],  
        dest: 'dist/<%= pkg.name %>.js'  
      }  
    },  
    uglify: {  
      options: {  
        banner: '!*<%= pkg.name %> <%= grunt.template.today("dd-mm-yyyy")%>'  
      },  
      dist: {  
        files: {  
          'dist/<%= pkg.name %>.min.js': ['<%= concat.dist.dest %>']  
        }  
      }  
    },  
    qunit: {  
      options: {  
        // Options  
      },  
      files: {  
        // Files  
      }  
    }  
  });  
  
  // Load the plugin  
  grunt.loadNpmTasks('grunt-contrib-concat');  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  grunt.loadNpmTasks('grunt-qunit');  
  
  // Task configuration  
  concat.dist();  
  uglify.dist();  
  qunit.suites();  
};
```

WHAT'S THE DIFFERENCE?

- With Gulp your build file is code, not config
- You use standard libraries to do things
- Plugins are simple and do one thing - most are a ~20 line function
- Tasks are executed with maximum concurrency
- I/O works the way you picture it

Gulp does nothing but provide some streams and a basic task system Gulp has only 5 functions you need to learn

```
gulp.task(name, fn)
```

- It registers the function with a name.
- You can optionally specify some dependencies if other tasks need to run first.

```
gulp.run(tasks...)
```

- Runs all tasks with maximum concurrency

```
gulp.watch(glob, fn)
```

- Runs a function when a file that matches the glob changes
- Included in core for simplicity

```
gulp.src(glob)
```

- This returns a readable stream.
- Takes a file system glob (like grunt) and starts emitting files that match.
- This is piped to other streams

```
gulp.dest(folder)
```

- This returns a writable stream
- File objects piped to this are saved to the file system

YEOMAN

APPLICATION SCAFFOLDING

YEOMAN

- Yeoman is a Node.js package for application scaffolding
 - Uses bower & NPM to install the js package
 - Has lots of generators for many types of applications:
 - MEAN, AngularJS, Kendo-UI, WebApp, WordPress, Backbone, Express, etc...
 - Each generators install both needed Node.js packages and client-side JavaScript libraries
 - Generated Gruntfile.js for build/test/serve

YEOMAN

Install Yeoman:

```
npm install -g yo
```

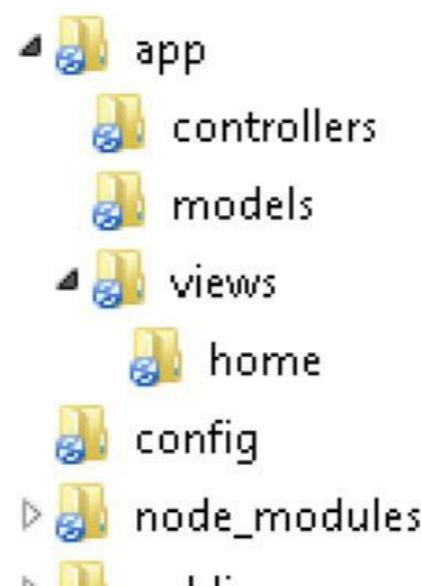
Install Yeoman generator:

```
npm install -g generator-jhipster
```

Scaffold Express application:

```
cd path/to/app/directory  
yo jhipster
```

Generates:



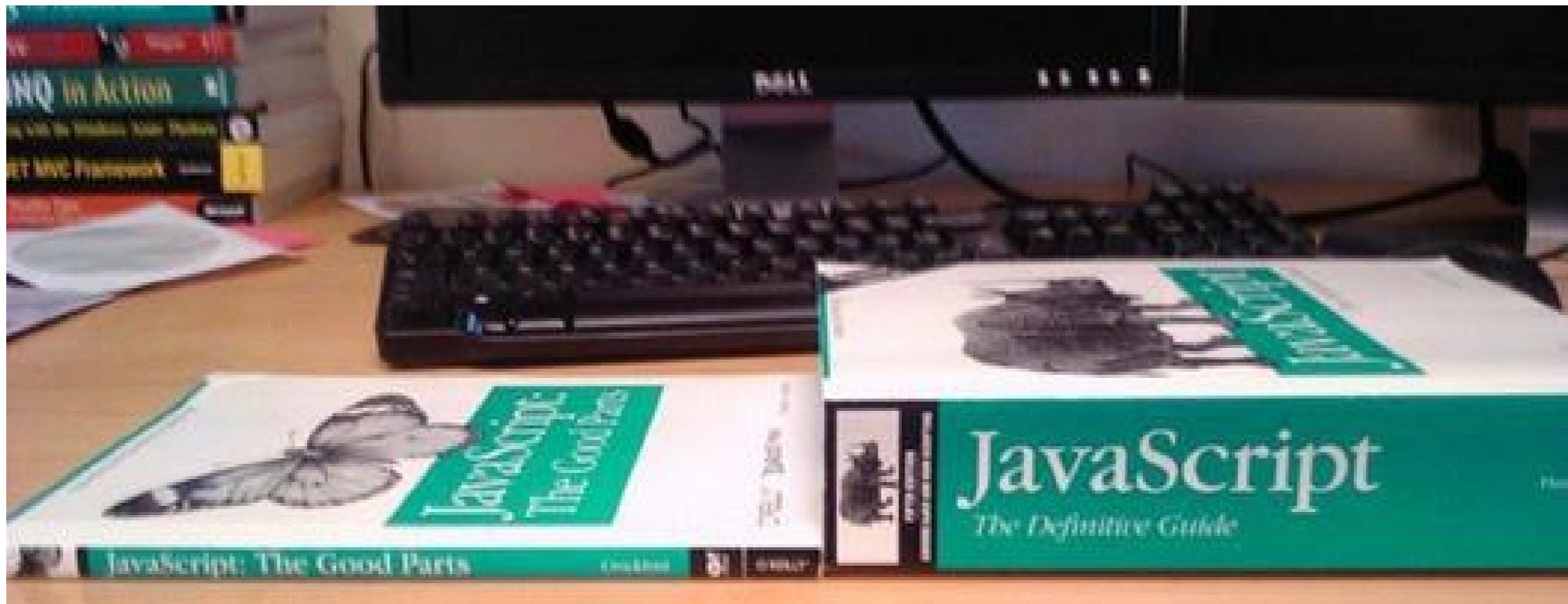
TYPESCRIPT

WHAT IS TYPESCRIPT?

- Free and open source, strongly supported by Microsoft
- Based on ecmascript 4 + ecmascript 6
- Created by the father of C# Anders Hejlsberg
- A superset of JavaScript
- To answer why we need JavaScript+, we need to understand what's wrong with vanilla JavaScript

WHAT IS THE PROBLEM ?

- Why do people hate working in JavaScript?



*Using state of the art software
engineering practices ;)*

WHAT IS THE PROBLEM ?

- JS is designed for small things
- We now use to do big things
- But JavaScript is not suited for building large applications

Your JavaScript code gets complex; it becomes extremely unwieldy

LET'S LOOK AT TYPESCRIPT

- To get started with TypeScript, grab it from
<http://typescriptlang.org>
- Let's look at TypeScript, first the core concept...

TYPESCRIPT - FIRST GLANCE - OPTIONAL STRONG TYPE CHECKING

```
// jsfunction
f(x, y) { return x * y;}
// tsfunction
f(x : number, y : number) : number { return x * y;}
// Type information is enforced in design and
// compile time, but removed at runtime
```

TYPESCRIPT FEATURES

- Static Type Checking
- Modules and Export
- Interface and Class for traditional Object Oriented Programming
- Works with all your existing JavaScript libraries
- Low learning overhead compared to similar JavaScript systems (CoffeeScript or Dart)
- Amazing Visual Studio, visual code studio, eclipse or IntelliJ tooling
- Outstanding team and refactoring scenarios

SUMMARY - WHY TYPESCRIPT ? (EXPECTED BENEFITS)(1)

- Have to learn one more thing - there is a learning curve, very easy if you already know JavaScript, or if you know C# or Java very well.
- You still have to learn JavaScript - Understanding how TypeScript converts to JavaScript will teach you better JavaScript
- Some definition files don't exist or incomplete, but I think this will vanish very quickly. These aren't hard to write if you really need something.

SUMMARY - WHY TYPESCRIPT ? (EXPECTED BENEFITS)(2)

- Modules and classes enable large projects and code reuse
- Compile-time checking prevents errors
- Definition files for common JavaScript libraries makes them very easy to work with, and provides strong type checking
- Source map debugging makes debug easy

INITIAL CONCLUSION - IF I HAVE TO MAKE A DECISION FOR YOU...

- If you see yourself using more JavaScript. You have to look at TypeScript.
- If you and your team has to work on JavaScript together, you have to look at TypeScript.
- Once you've done the initial hard work and converted a project. You can't stand going back.

INTRODUCTION TO



ANGULARJS
by Google

EARLY WEB

- Web designed for documents
- Server creates pages / browser displays
- Data input sent to and processed by the server
- Updated pages created on the server and resent

FIRST EXAMPLE - PHP

```
<!doctype html>
<html>
<head>
</head>
<body>
  <form method="post" action="hello.php">
    <label>Name:</label>
    <input type="text" id="yourName">
    <input type="submit" value="Say Hello" />
    <hr>
  <?php
    echo "<h1>Hello ".$HTTP_POST_VARS["yourName"]."</h1>";
?>
  </form>
</body>
</html>
```

WEB EVOLUTION - AJAX

- Interactive client-side web
 - Collect input from user
 - Update display
 - Communicate with server
- Client-side processing enabled by
 - JavaScript
 - DOM manipulation
 - HTTP server messaging

FIRST EXAMPLE - JQUERY

```
<html>
<head>
  <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
  <script type="text/javascript">
    $(function() {
      $("#yourName").keyup(function () {
        $("#helloName").text("Hello " + this.value + "!");
      });
    });
  </script>
</head>
<body>
  <div>
    <label>Name:</label>
    <input type="text" id="yourName">
    <hr>
    <h1 id="helloName"></h1>
  </div>
</body>
</html>
```

JQUERY

- Simplifies event binding and DOM manipulation
- Common API across multiple browsers
- Supports plug-in modules to extend functionality
- Requires writing JavaScript code to wire

TODAY'S WEB – CAN WE DO BETTER?

- Follow good programming practices
 - Separate: data / display / processing
 - Simplify connecting data to display
- Let us focus on the technologies of the web
 - HTML
 - CSS
 - JavaScript

FIRST EXAMPLE - ANGULARJS

```
<html ng-app>
<head>
  <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>
</head>
<body>
  <div>
    <label>Name:</label>
    <input type="text" ng-model="yourName">
    <hr>
    <h1>Hello {{yourName}}!</h1>
  </div>
</body>
</html>
```

IMPERATIVE VS. DECLARATIVE

```
<input type="text" id="yourName">
<h1 id="helloName"></h1>
<script type="text/javascript">
$(function() {
  $("#yourName").keyup(function () {
    $("#helloName").text("Hello " + this.value + "!");
  });
});
</script>
```

to AngularJS declarative relationships

```
<input type="text" ng-model="yourName">
<h1>Hello {{yourName}}!</h1>
```

ABSTRACTIONS

- jQuery abstracts browser functionality
 - e.g. DOM traversal, event binding
- AngularJS abstracts relationships (and more)

AngularJS, and all web apps, are built on browser functionalities

THE DOM ABSTRACTION

- HTML is a declarative document language
- Browser translates HTML into a Document Object Model (DOM)
- DOM is the browser's in-memory document representation
- JavaScript can manipulate the DOM

ANGULARJS "*COMPILES*" HTML

- Browsers send a document (i.e. DOM) ready event
- AngularJS can intercede and rewrite the DOM
- The rewrite is driven by markup in the DOM

Model-View-Controller (MVC)

- Software architectural pattern
 - Separates display from data
 - Originated with Smalltalk programmers
 - From work at [Xerox PARC](#) in the late 70's
- Models represent knowledge
- Views provide a (visual) representation of attached model data
- Controllers connect to and control views and models

MVC AND VARIATIONS

- Different variations of the pattern
- Model-View-ViewModel (MVVM)
- Model-View-Presenter (MVP)
- Variations differ on...
 - connectivity
 - cardinality
 - directionality

MODEL-VIEW-WHATEVER

" **MVC vs MVVM vs MVP**. What a controversial topic that many developers can spend hours and hours debating and arguing about. For several years AngularJS was closer to **MVC** (or rather one of its client-side variants), but over time and thanks to many refactorings and api improvements, it's now closer to **MVVM** – the **\$scope** object could be considered the **ViewModel** that is being decorated by a function that we call a **Controller**.

...

I hereby declare AngularJS to be **MVW** framework - Model-View-Whatever.

..."

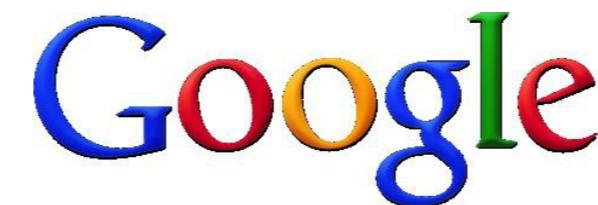
Igor Minar – Google AngularJS Development Team

SUMMARY WHAT IS ANGULARJS?

- An MVC framework for **efficiently** creating dynamic views in a web browser (using HTML and JavaScript)
- Some highlights/focuses:
 - Complete application framework
 - From *jQuery replacement* to a massive **enterprise Single Page Application (SPA)**
 - Fully dynamic MVVM with POJOs
 - Low level-DOM manipulation/markup invention with directives and templates
 - AJAX / REST API interaction
 - Code organization, dependency injection, testability
 - Comprehensive SPA and routing support

WHY SHOULD WE CARE?

- It's open source
- Actively developed by Google
 - Google is paying devs to actively develop Angular



- Actively developed by open source community (on GitHub)



ANGULAR.JS #1?

- Angular.js appears to be winning the JavaScript framework battle (and for good reason)

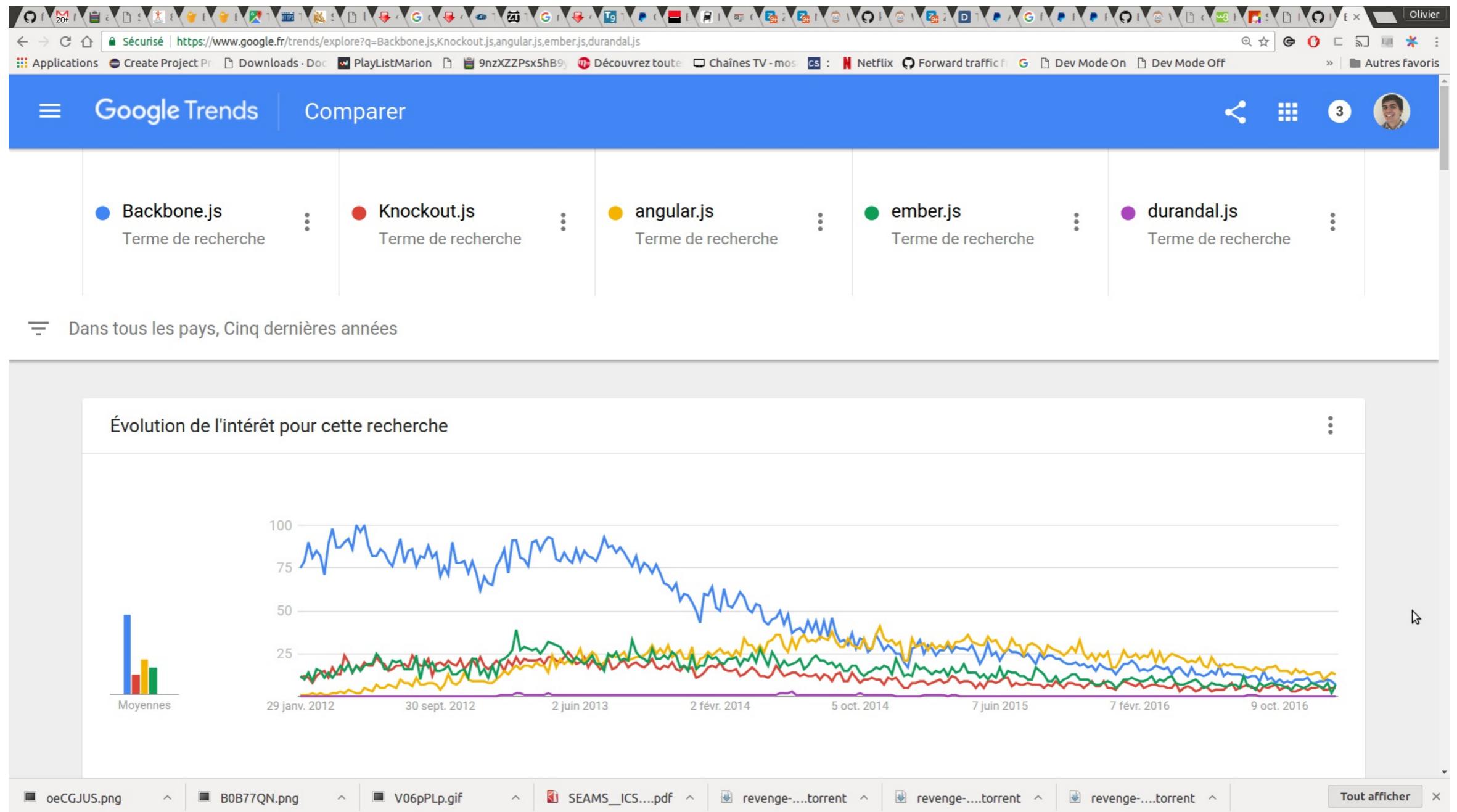


- Lets see some evidence...

WHY CARE? – GITHUB STATS

	Angular	Ember	Backbone	Knockout	Durandal
Stars	54,449	17,438	25,976	7,977	1,700
Watches	4,477	1,093	1,627	606	192
Forks	27,072	3,646	5,641	1,371	395
commit	8,329	13,969	3,337	1,484	1,181
release	185	232	30	38	6
contributors	1,562	638	292	62	70

WHY CARE? - GOOGLE TRENDS



AngularJS vs jQuery

- jQuery is a library meant for DOM manipulation, animations and an AJAX wrapper. NOT an application framework
- Pros
 - None. Angular has built in ‘basic’ jQuery.
 - If full-blown jQuery is added Angular will automatically use it. Generally full blown NOT needed.
- Cons
 - Horrible choice for creating dynamic UIs.
 - Verbose code, hard to maintain, not organized
 - Not MVVM or MVC

ANGULARJS VS BACKBONEJS

- Provides structure to web applications as well as model, view, templating, basic routing and RESTful interactions.
- Pros
 - Older and more mature, more people using it
- Cons
 - *Previous generation* web app framework
 - No MVVM w/o addons – use jQuery for DOM manip.
 - No DI, not as easily testable
 - Not a full-featured framework, meant to be ‘light’
 - Extremely verbose for what you get
 - Not actively developed

ANGULARJS VS DURANDAL.JS

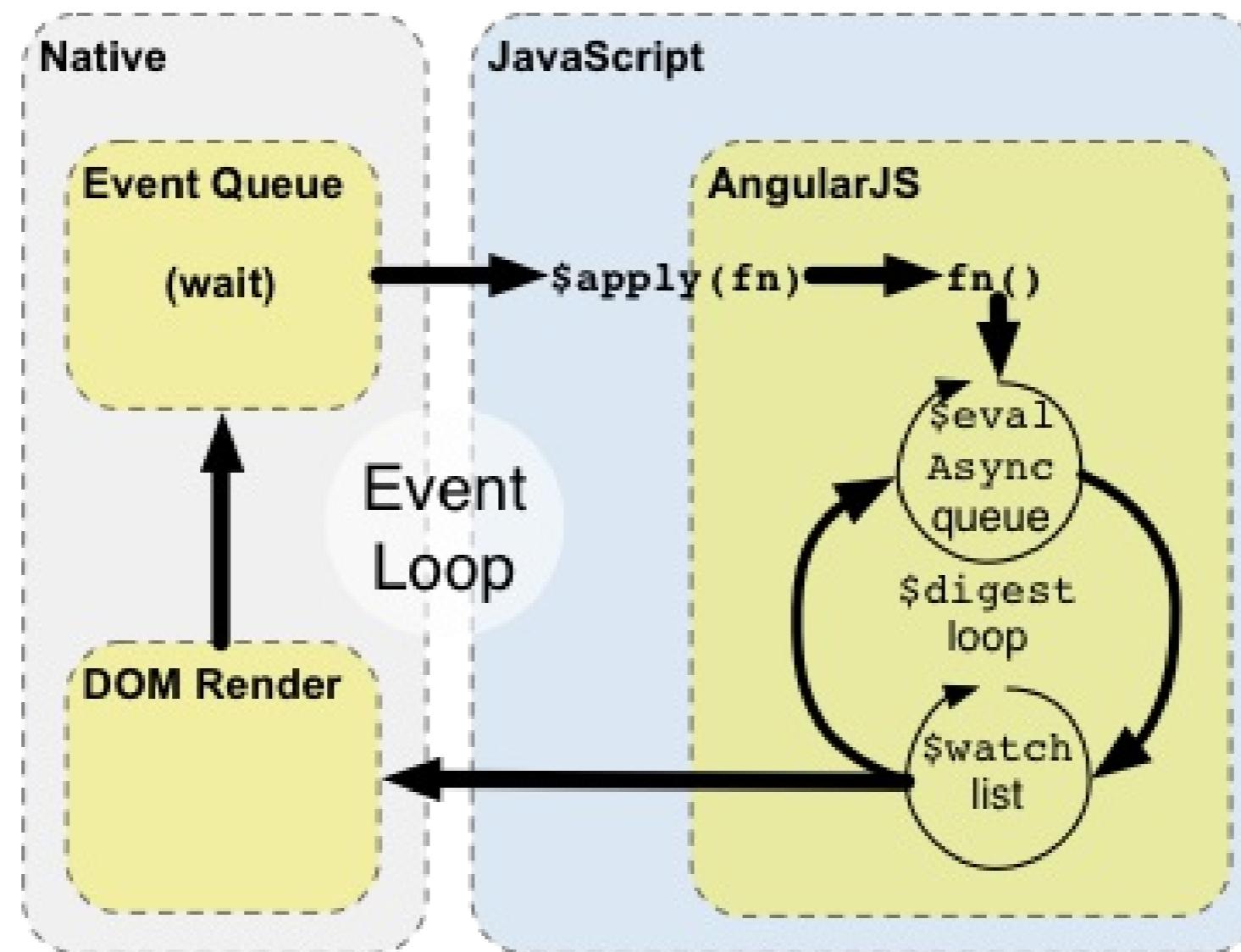
- Provides an app framework around Knockout
- Pros
 - Has a few features not baked into Angular (though readily available 3rd party from Angular community)
- Cons
 - Uses Knockout for data binding, suffers from same knockout issues
 - Lacking many Angular.js features
 - No one is using it
 - Is mostly a 1 man show (Rob Eisenberg)

WHY ANGULAR CAN BE GOOD (2)

1. FLEXIBLE! As big or small as you want it to be
 - Two line jQuery replacement to a MASSIVE enterprise app
2. POJOs make life so easy. No ‘observables’, wrappers etc.
Uses dirty checking for 2-way binding.
 - Fully embraces the dynamic nature of JavaScript
3. The community and popularity
4. DI, services, factories, providers offer flexibility and familiarity to traditionally server side paradigms
5. Directives offer DSL-like extension to HTML for your domain specific use cases
6. Scopes, although tricky, offer extreme flexibility

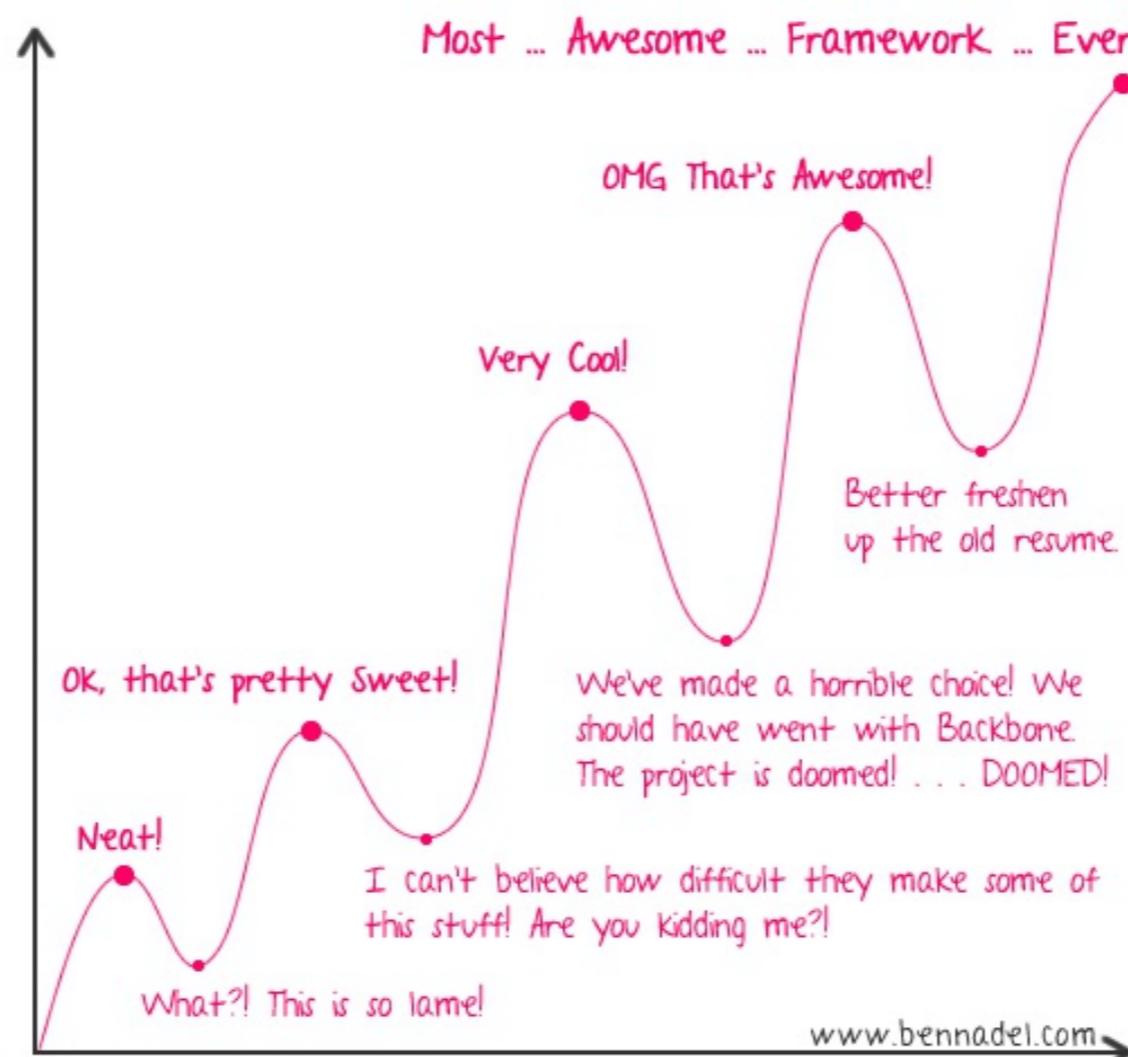
WHY ANGULAR CAN BE GOOD (2)

1. Efficient



WHY I DON'T DIDN'T LIKE ANGULAR

- Scopes are hard initially, but awesome
- Learning curve === eventual productivity



My Feelings About AngularJS Over Time

CODE TIME! A SIMPLE EXAMPLE

```
<html>
<head>
  <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>
</head>
<body ng-app="my-app">
  <div ng-controller="exempleCtrl">
    <label>Name:</label>
    <input type="text" ng-model="yourName">
    <hr>
    <h1>Hello {{yourName}}!</h1>
  </div>
</body>
</html>
```

CODE TIME! A SIMPLE EXAMPLE

```
var app = angular.module("my-app", []);  
  
app.controller("exempleCtrl", function($scope) {  
  $scope.yourName = "World"  
});
```

CODE TIME! A SIMPLE EXAMPLE

```
var app = angular.module("my-app", []);
app.controller("exempleCtrl", function($scope) {
  $scope.yourName = "World";
  var observeYourName = function(){
    console.log('new value' + $scope.yourName);
  };
  $scope.$watch('yourName', observeYourName);
});
```

SIMPLE EXAMPLE

- ng-app attribute causes Angular to scan children for recognized tokens
 - Creates the “root scope” \$rootScope
 - \$rootScope ≈ a ViewModel (Angular sees three “directives”)
 - `({{yourName}})`
- Evaluated against the current \$rootScope and updates the DOM on any change. "1 – way bound"
 - `ng-model="firstName"`
- An input to be 2-way bound against \$scope.yourName
 - `ng-model="lastName"`
- An input to be 2-way bound against \$scope.yourName

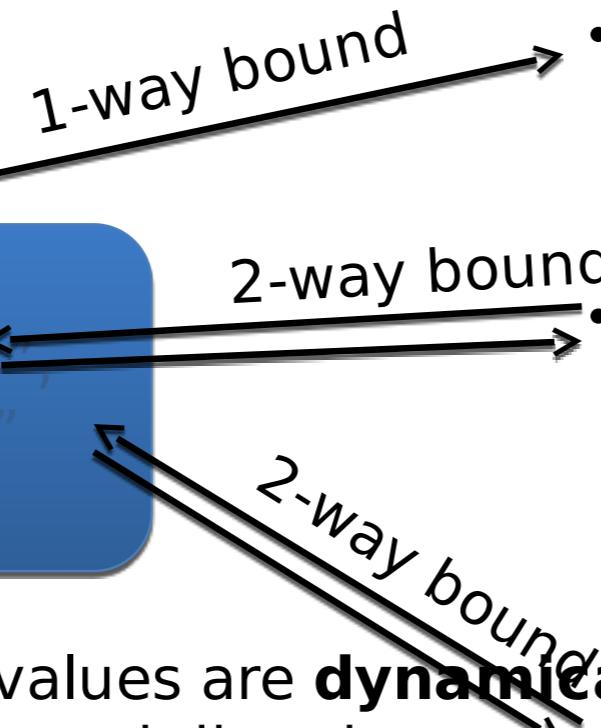
Original \$rootScope:

```
$rootScope = {};
```

After typing:

```
$rootScope = {  
  firstName: "Olivier",  
  lastName: "Barais"  
};
```

- Object fields and values are **dynamically** assigned by the bound directives.



Directives

Perform the 1 or 2 way binding between the DOM and the model (\$rootScope)

- {{firstName + " " + lastName}}
 - Watch for \$scope changes and reevaluate the expression
- ng-model="firstName"
 - Watch for \$scope.firstName changes, update the textbox
 - Watch for textbox changes, update \$scope.firstName
- ng-model="lastName"
 - Watch for \$scope.lastName changes, update the textbox
 - Watch for textbox changes, update \$scope.lastName

WHAT IS SCOPE?

- Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events. (from Angular website)
- Key points
 - Scope is like a ViewModel that allows communication between JavaScript code and Views
 - `{{yourName}}` is an expr executed against scope
 - Scope can be hierachal with DOM nesting of directives
 - Watches can be used to watch for changes to scope ex:

WHAT IS A DIRECTIVE?

- A reusable component for performing DOM interaction, templating and possibly two-way binding against \$scope
 - The ONLY place JS to DOM interaction should occur
- Angular offers a huge amount of built in directives for common UI tasks, ex:

```
<div ng-show="someBool">someBool is true!</div>
```

- 2 way binding inputs, setting classes, foreach loops of elements, clicking etc.

WHAT IS A DIRECTIVE?

- You can write your own directives for domain specific purposes (a ‘DSL’ for HTML). Ex:

```
<slideshow title="Shocked Cats">
<slide src="cat1.jpg"></slide>
<slide src="cat2.jpg"></slide>
...
</slideshow>
```

WHAT IS A DIRECTIVE?

- Or simply invoke an existing jQuery plugin

```
<datepicker ng-model="aDate"></datepicker>
```

- Or if you need <=IE8 support:

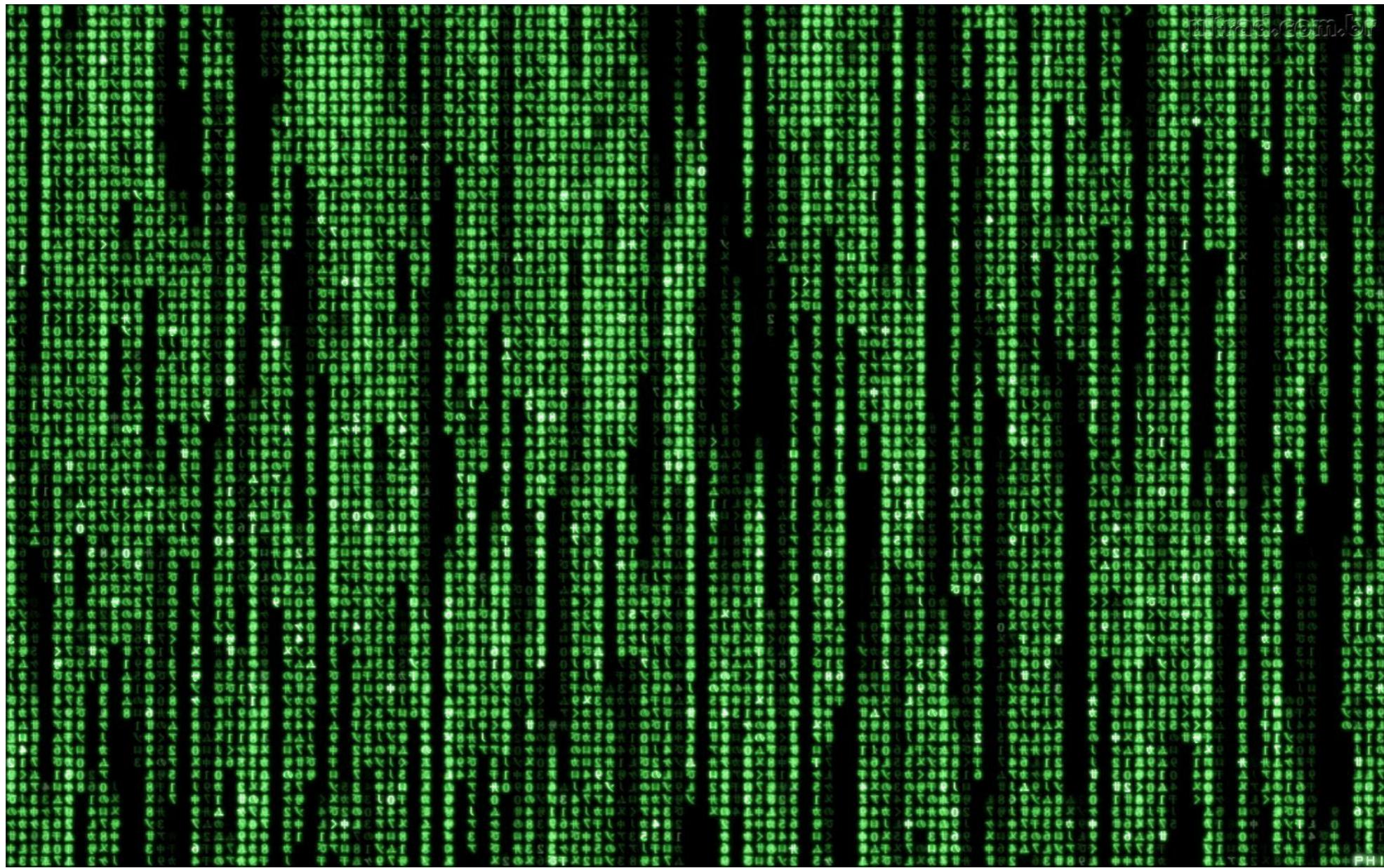
```
<input type="text" datepicker="" ng-model="aDate"/>
```

WHAT IS A DIRECTIVE?

- HUGE amount of directives out there due to Angular's popularity. Rarely have to write your own other than domain specific directives
 - EX: AngularUI
 - Twitter bootstrap wrappers
 - Select2
 - Sorting
 - Input masking
 - Enhanced router
 - Etc...
 - Various wrappers for jQuery UI components (ex: datepicker)

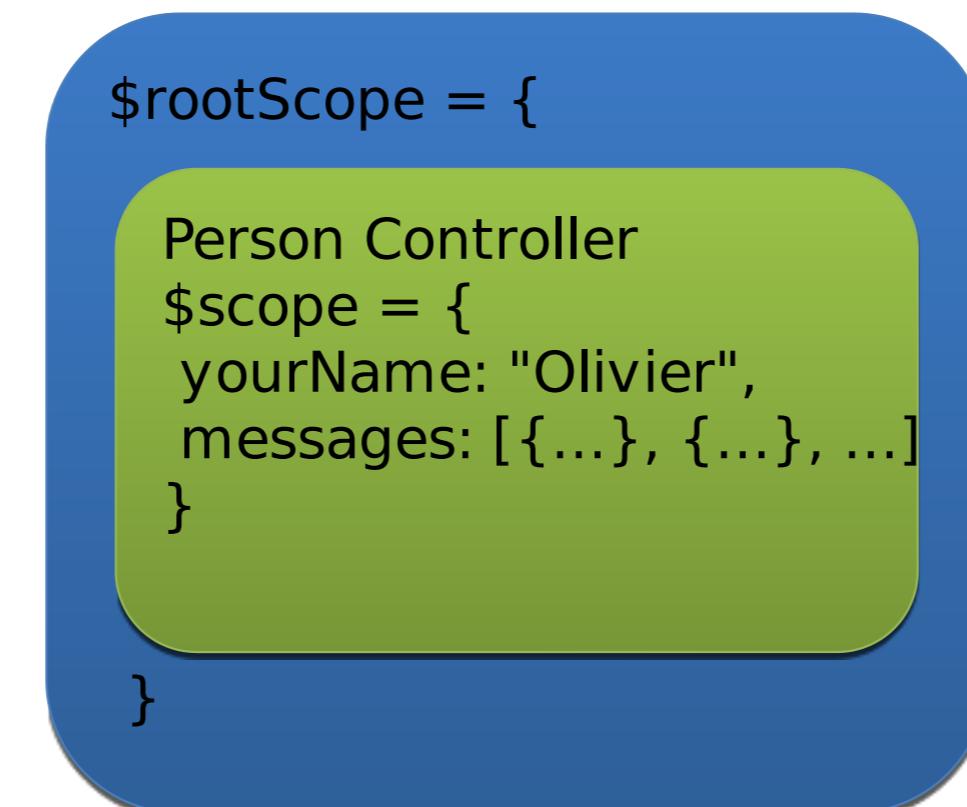


ADDING “MESSAGE UPDATES” WITH A CONTROLLER



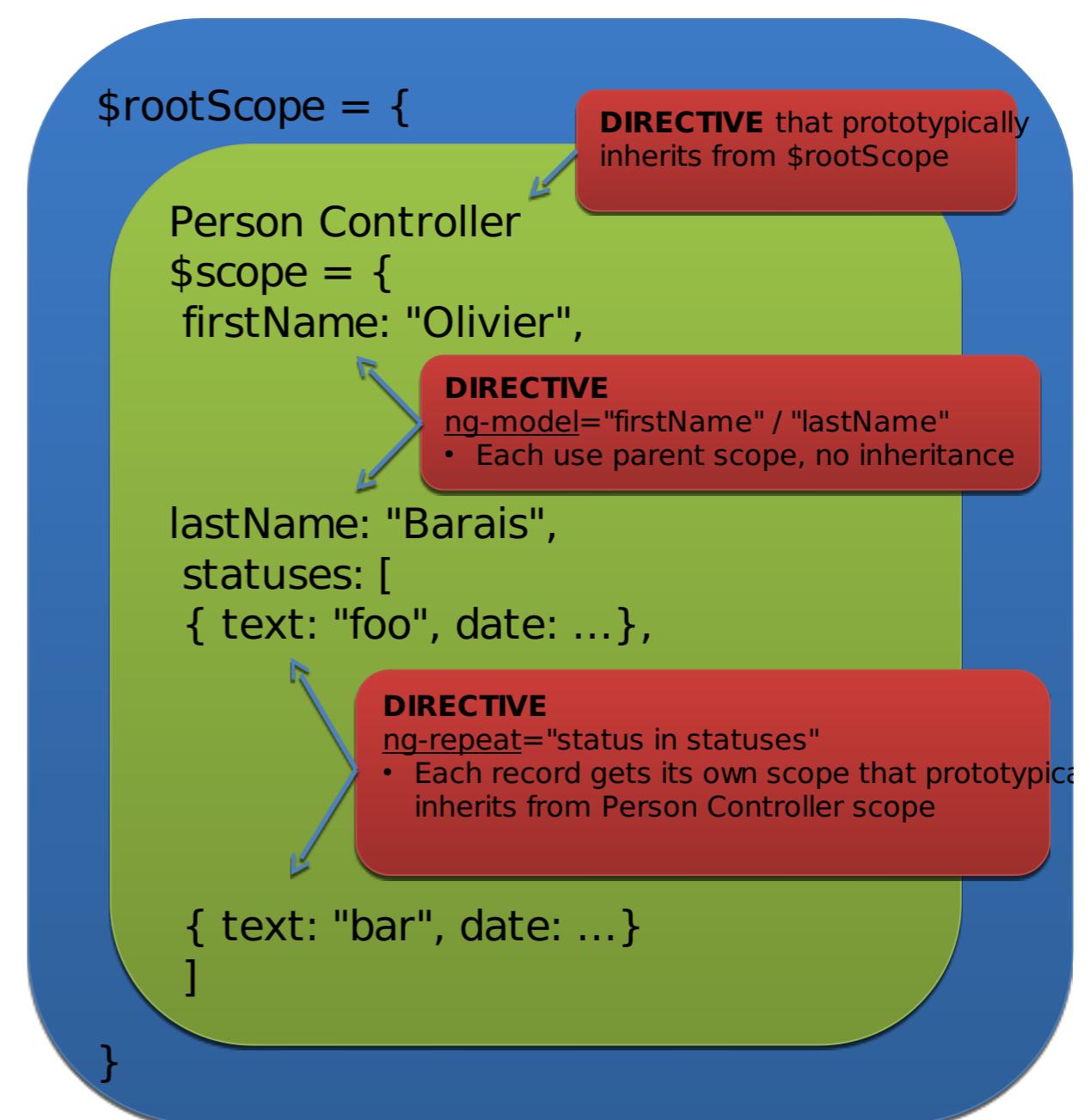
WHAT IS A CONTROLLER?

- A controller is really just a fancy name for a "scope container" that prototypically inherits from its parent scope container.
- A controller can interact with `$scope` (the 'view model') which the view can also interact with.



DIRECTIVES AND SCOPE

- A controller is really a directive that is configured to prototypically inherit from its parent
- Directives can be configured for what type of scope they create and parent access they have
Use "AngularJS Batarang" plugin for Chrome to explore scopes



SERVICES

- Software architectural components
- Services provide data and compute
- Exist across views
- Depend on other services
- AngularJS has 20+
- `$http` – service to communicate with servers
- `$log` – service to do pretty log messages

SERVER COMMUNICATION

- *\$http* service
 - Input config object
 - Returns promise
 - Communication is asynchronous

```
$http({method: 'GET', url: fetchUrl})
  .success(function(data, status) {
    // process the data here
  })
  .error(function(data, status) {
    // error handling
});
});
```

PROMISES

- **Promises** represent result of an action
- Particularly used with asynchronous actions
- They are either resolved or rejected

DEPENDENCY INJECTION (DI)

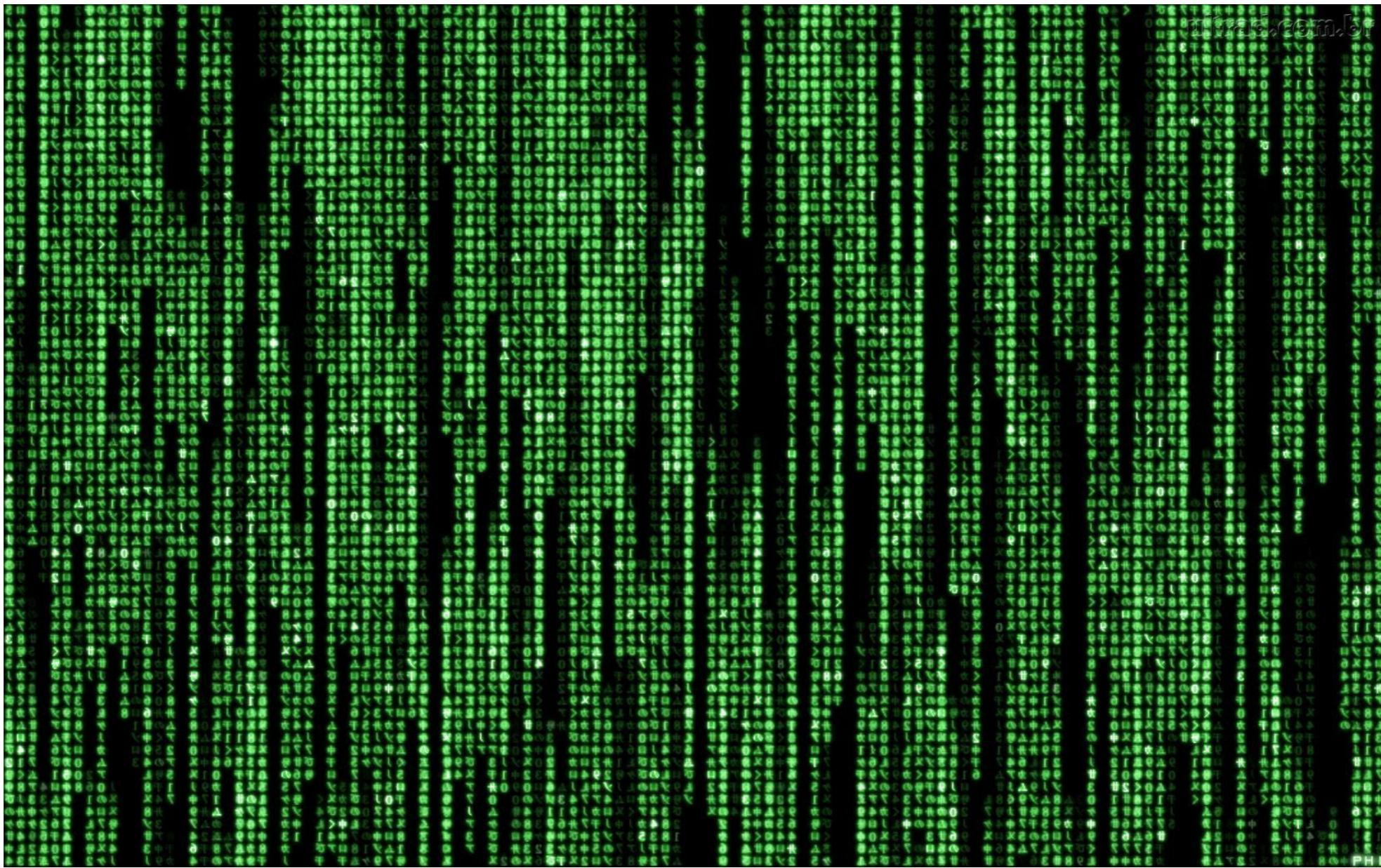
- DI is a software architectural pattern
- Separation of concerns
- Service creation independent from usage
- Good for
- Modular code
- Allows AngularJS to wire in correct order
- Supports substitution (for patching and testing)

DI AND JAVASCRIPT MINIFICATION

- Services identified by parameter name
- Minification obfuscates the name
- Pass names as strings in array

```
angular.module('GoaleryServices')
  .factory('StatusManager',
    [      'CloudLogin', '$q',
      function (cloudLogin,  $q) {
...
}]);
```

FIXING THE UGLY DATES AND ORDERING WITH FILTERS



WHAT IS A FILTER?



- A function that transforms an input to an output
 - Reminds me a lot of LINQ extension method lambdas in .NET
 - Can be "piped" UNIX style
 - Can create own
 - Angular has many built in filters:
 - currency, date, filter, json, limitTo, lowercase, number, orderBy, uppercase

VALIDATION WITH NG-FORM



WHAT IS NG-FORM?

- NOT a traditional HTML "form"
 - Requires a "name" and "ng-model" on each input you wish to validate
 - Angular will not push invalid values back into bound \$scope
- Allows for validation of collections of controls
 - Applies CSS classes to elements based on their validity
 - Lots of built in validator directives that work with ng-form:

```
required=""
ng-minlength="{number}"
ng-maxlength="{number}"
ng-pattern="{string}"
ng-change="{string}"
```

WHAT IS NG-FORM?

- Angular UI has some extensions
- AngularAgility - FormExtensions makes it easier
 - [demo](#)

ANGULARJS IS PURE JAVASCRIPT

- Prototype-based scripting language
- Dynamic, weakly typed, first-class functions
- Great JavaScript book:
 - Crockford (2008) *JavaScript: The Good Parts* – O'Reilly

TESTING ANGULARJS APPS

- JavaScript doesn't have a compiler
- Must execute code to test
- Testability was a fundamental objective of AngularJS
 - Miško Hevery (AngularJS creator)
 - Previously created JsTestDriver

ANGULARJS SUPPORTS TESTING

- Unit testing support
 - JsTestDriver
 - Jasmine
- DI allows substituting mocks for hard to test code
 - Server communication
 - Logging
- Angular Scenario Runner – E2E testing
 - Simulates user interactions

BUILDING WEB APPS

- Single web page
 - Loads the base HTML and included sources once
 - App changes views dynamically
- Server is called upon when needed
- Prefer using asynchronous server calls
 - Data changes
 - Fetch more data

BUILDING WEB APPS

- Declarative view specification
- HTML augmented with:
- Directives, Markup, Filter, Form Controls
- Loaded either
- with a simple single web page
- dynamically into a `view` as partials

URL ROUTING

- Define the mapping from URL to view
- Can also bind controller
- Define URL parameters

```
$routeProvider.when('/Book/:bookId', {  
  templateUrl: 'book.html',  
  controller: BookCtl  
});  
routeProvider.when('/Book/:bookId/ch/:chapterId', {  
  templateUrl: 'chapter.html',  
  controller: ChapterCtl  
});
```

DEEP LINKING

- AngularJS navigation updates the browser address bar
- Uses the HTML5 history API – fallback to hash-bang (#!) URL
- Users can link to pages in your app
- Server must recognize the link pattern and serve the application
- <https://github.com/angular-ui/ui-router/wiki/Quick-Reference>

CREATING DIRECTIVES

- Directives package reusable HTML
- Naming nuance: "myDir" becomes "my-dir"
- Conceptual compile and link phases
- Can specify: scope, binding, restrictions, etc
- Supports transclusion
 - In computer science, transclusion is the inclusion of part or all of an electronic document into one or more other documents by hypertext reference.
- Consider creating a custom DSL

MODULES

- Packaging of JavaScript code
- **Modules** declare dependencies
- AngularJS instantiates in correct order
- Provides separation of namespaces

LET US PRACTICE

- Let us use the yeoman generator
- Install yo, grunt-cli, bower, generator-angular and generator-karma:

```
npm install -g grunt-cli bower yo generator-karma generator-angular
```

If you are planning on using Sass, you will need to first install Ruby and Compass:

- Install Ruby by downloading from [here](#) or use Homebrew
- Install the compass gem:

```
gem install compass
```

LET US PRACTICE

- Make a new directory, and cd into it:

```
mkdir my-new-project && cd $_
```

- Run yo angular, optionally passing an app name:

```
yo angular [app-name]
```

Run grunt for building and grunt serve for preview

GENERATORS AND SUB

- angular (aka angular:app)
- angular:controller
- angular:directive
- angular:filter
- angular:route
- angular:service
- angular:provider
- angular:factory
- angular:value
- angular:constant
- angular:decorator
- angular:view

APP

- Sets up a new AngularJS app, generating all the boilerplate you need to get started. The app generator also optionally installs Bootstrap and additional AngularJS modules, such as angular-resource (installed by default).

Example:

```
yo angular
```

ROUTE

- Generates a controller and view, and configures a route in app/scripts/app.js connecting them.
- Example:

```
yo angular:route myroute
```

- Produces app/scripts/controllers/myroute.js:

```
angular.module('myMod').controller('MyrouteCtrl', function ($scope) {  
  // ...  
});
```

- Produces app/views/myroute.html:

```
<p>This is the myroute view</p>
```

- **Explicitly provide route URI**

- Example:

```
yo angular:route myRoute --uri=my/route
```

- Produces controller and view as above and adds a route to app/scripts/app.js with URI my/route

CONTROLLER

- Generates a controller in app/scripts/controllers.
- Example:

```
yo angular:controller user
```

- Produces app/scripts/controllers/user.js:

```
angular.module('myMod').controller('UserCtrl', function ($scope) {  
  // ...  
});
```

DIRECTIVE

- Generates a directive in app/scripts/directives.
- Example:

```
yo angular:directive myDirective
```

- Produces app/scripts/directives/myDirective.js:

```
angular.module('myMod').directive('myDirective', function () {  
  return {  
    template: '<div></div>',  
    restrict: 'E',  
    link: function postLink(scope, element, attrs) {  
      element.text('this is the myDirective directive');  
    }  
  };  
});
```

FILTER

- Generates a filter in app/scripts/filters.
- Example:

```
yo angular:filter myFilter
```

Produces app/scripts/filters/myFilter.js:

```
angular.module('myMod').filter('myFilter', function () {  
  return function (input) {  
    return 'myFilter filter:' + input;  
  };  
});
```

VIEW

- Generates an HTML view file in app/views.
- Example:

```
yo angular:view user
```

- Produces app/views/user.html:

```
<p>This is the user view</p>
```

SERVICE

- Generates an AngularJS service.
- Example:

```
yo angular:service myService
```

- Produces app/scripts/services/myService.js:

```
angular.module('myMod').service('myService', function () {  
  // ...  
});
```

- You can also do yo angular:factory, yo angular:provider, yo angular:value, and yo angular:constant for other types of services.

DECORATOR

- Generates an AngularJS service decorator.
- Example:

```
yo angular:decorator serviceName
```

- Produces
app/scripts/decorators/serviceNameDecorator.js:

```
angular.module('myMod').config(function ($provide) {
  $provide.decorator('serviceName', function ($delegate) {
    // ...
    return $delegate;
  });
});
```

OPTIONS

In general, these options can be applied to any generator, though they only affect generators that produce scripts.

COFFEEESCRIPT AND TYPESCRIPT

For generators that output scripts, the `--coffee` option will output CoffeeScript instead of JavaScript, and `--typescript` will output TypeScript instead of JavaScript.

- For example:

```
yo angular:controller user --coffee
```

- Produces app/scripts/controller/user.coffee:

```
angular.module('myMod')
.controller 'UserCtrl', ($scope) ->
```

- For example:

```
yo angular:controller user --typescript
```

- Produces *app/scripts/controller/user.ts*:

```
/// <reference path="../app.ts" />

'use strict';

module demoApp {
    export interface IUserScope extends ng.IScope {
        awesomeThings: any[];
    }
}

export class UserCtrl {

    constructor (private $scope:IUserScope) {
        $scope.awesomeThings = [
            'HTML5 Boilerplate',
            'AngularJS',
            'Karma'
        ];
    }
}
```

MINIFICATION SAFE

tl;dr: You don't need to write annotated code as the build step will handle it for you.

By default, generators produce unannotated code. Without annotations, AngularJS's DI system will break when minified.

Typically, these annotations that make minification safe are added automatically at build-time, after application files are concatenated, but before they are minified. The annotations are important because minified code will rename variables, making it impossible for AngularJS to infer module names based solely on function parameters.

The recommended build process uses [ng-annotate](#), a tool that automatically adds these annotations. However, if you'd rather not use it, you have to add these annotations manually yourself. Why would you do that though? If you find a bug in the annotated code, please file an issue at [ng-annotate](#).

ADD TO INDEX

- By default, new scripts are added to the index.html file. However, this may not always be suitable. Some use cases:
 - Manually added to the file
 - Auto-added by a 3rd party plugin
 - Using this generator as a subgenerator
- To skip adding them to the index, pass in the skip-add argument:

```
yo angular:service serviceName --skip-add
```

BOWER COMPONENTS

- The following packages are always installed by the [app](#) generator:
 - angular
 - angular-mocks

BOWER COMPONENTS

- The following additional modules are available as components on bower, and installable via bower install:
- angular-animate
- angular-aria
- angular-cookies
- angular-messages
- angular-resource
- angular-sanitize

All of these can be updated with `bower update` as new versions of AngularJS are released.

CONFIGURATION

Yeoman generated projects can be further tweaked according to your needs by modifying project files appropriately.

OUTPUT

- You can change the app directory by adding an `appPath` property to `bower.json`. For instance, if you wanted to easily integrate with Express.js, you could add the following:

```
{  
  "name": "yo-test",  
  "version": "0.0.0",  
  ...  
  "appPath": "public"  
}
```

- This will cause Yeoman-generated client-side files to be placed in `public`.

OUTPUT

- Note that you can also achieve the same results by adding an `--appPath` option when starting generator:

```
yo angular [app-name] --appPath=public
```

TESTING

Running

```
grunt test
```

will run the unit tests with karma.

THE PROGRESSIVE JAVASCRIPT FRAMEWORK



Vue.js

VUE IS EASY

REACTIVE WEB DEVELOPMENT CONCEPTS

TEMPLATES AND DATA BINDING

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" langage="Jav  
<div id = "app">  
{{message}}  
</div>
```

```
var app = new Vue({  
  el: '#app',  
  data:{  
    message : 'hello Vue!'  
  }  
})
```

PLAYING WITH THE DOM

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" language="Java  
  
<ul id="example-1">  
  <li v-for="item in items">  
    {{ item.message }}  
  </li>  
</ul>
```

```
var example1 = new Vue({  
  el: '#example-1',  
  data: {  
    items: [  
      { message: 'Foo' },  
      { message: 'Bar' }  
    ]  
  }  
})
```

TWO-WAY DATA BINDING

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" langage="Java  
<input v-model="message" placeholder="edit me">  
<p>Message is: {{ message }}</p>
```

TWO-WAY DATA BINDING

```
<select v-model="selected" multiple>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
<br>
<span>Selected: {{ selected }}</span>
```

TRANSITION EFFECTS

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" language="JavaScript">

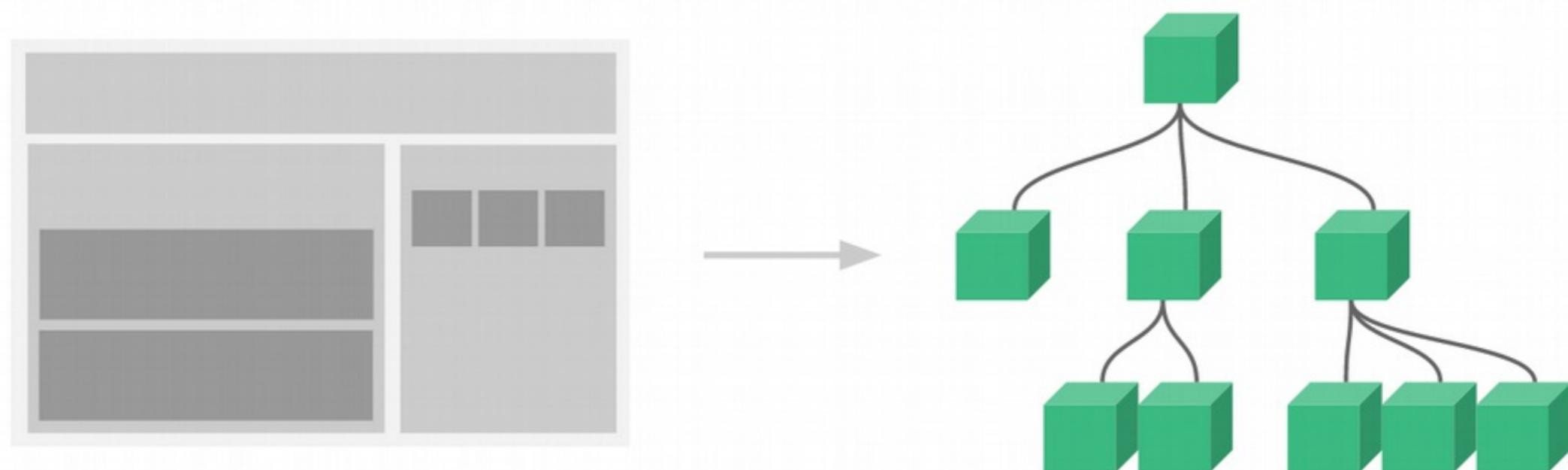
<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

```
new Vue({
  el: '#demo',
  data: {
    show: true
  }
})
```

```
.fade-enter-active, .fade-leave-active {
  transition: opacity .5s
}
.fade-enter, .fade-leave-to /* .fade-leave-active below version 2.1.8 */ {
  opacity: 0
}
```

**VUE CAN BE EASILY
INTEGRATED
LARGE OR SMALL**

COMPONENTS



COMPONENTS

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.2/vue.js" langage="Jav
<div id="example">
  <my-component></my-component>
</div>
```

```
// register
Vue.component('my-component', {
  template: '<div>A custom component!</div>'
})
// create a root instance
new Vue({
  el: '#example'
})
```

CLEAN AND SIMPLE

CLEAN AND SIMPLE

- Vue was created by Evan You after working for Google on AngularJS. You later summed up his thought process,

I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight without all the extra concepts involved?



A COMPLETE REWRITE

WHY COMPLETE REWRITE?

- JavaScript is more matured now
- Typescript is more mature now
- Browsers are more matured now
- Performance improvements
- Saying bye to some of old friends (scope, modules, DDO, controllers, jqLite)
- And several other reasons...

ANGULARJS

- Open source
- Maintained by google and community
- Frontend framework
- Adjusts and adapts traditional HTML in order to present dynamic content through the two way data biding which allows instantaneous synchronization of views and models.

ANGULARJS 2

- Release in September 2016
- Using TypeScript
- Many modern browsers support it (Even IE9)

TYPESCRIPT

- TypeScript is a typed superset of JavaScript that compiles to plain JavaScript
- Optional tipization and object oriented
- Supports ECMAScript 2015 features like async functions and decorators
- October 2012 (v0.8) Microsoft-a

TYPESCRIPT

```
class Student {  
    fullName: string;  
    constructor(public firstName, public middleInitial, public lastName) {  
        this.fullName = firstName + " " + middleInitial + " " + lastName;  
    }  
}  
  
interface Person {  
    firstName: string;  
    lastName: string;  
}  
  
function greeter(person : Person) {  
    return "Hello, " + person.firstName + " " + person.lastName;  
}  
  
var user = new Student("Jane", "M.", "User");  
  
document.body.innerHTML = greeter(user);
```

```
<html>  
    <head><title>TypeScript Greeter</title></head>  
    <body>  
        <script src="greeter.js"></script>  
    </body>
```

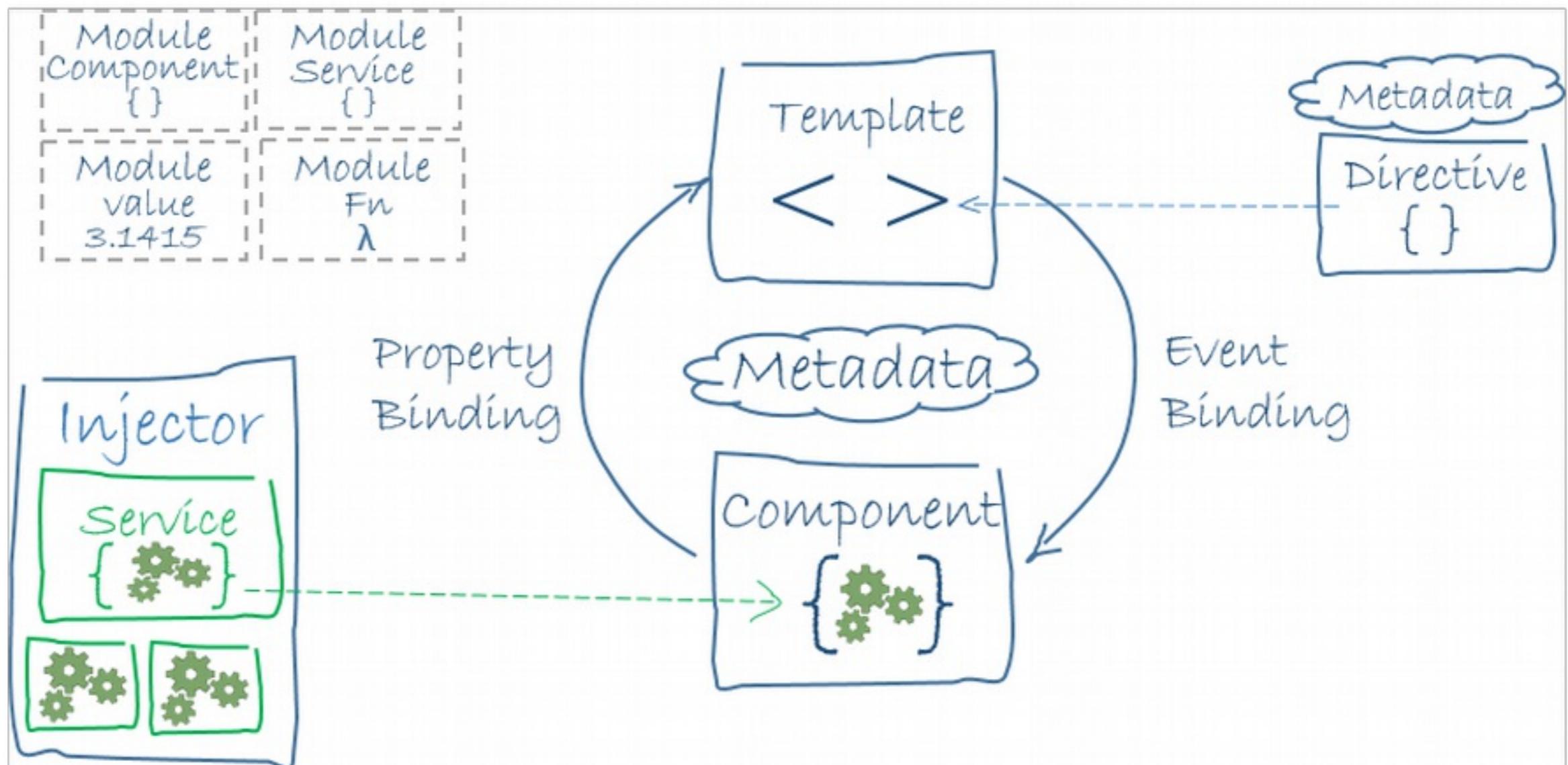
TYPESCRIPT

- Additional functionalities
 - Modules, classes, interfaces, “arrow” syntaks, namespaces, enumerated types
- Weakly or dynamically typed (:any)
- Strongly typed (:string, :number, :Class)

ANGULAR 2 CORE: CHANGE DETECTION

- Unidirectional tree based change detection
- Unidirectional flow is a win for performance
- Change detection works with:
 - Plain JS Objects
 - Immutable Objects
 - Observable objects

BIG PICTURE



ANGULARJS 2

- **Modules**
 - Contains parts of the application which we export
- **Components**
 - Application logic which controls parts of the user interface
- **Templates**
 - Renders the component on the page
- **Metadata** -- Information about the angular application parts

MODULES

app/app.component.ts:

```
export class AppComponent { }
```

app/boot.ts:

```
import {AppComponent} from './app.component';
```

Module

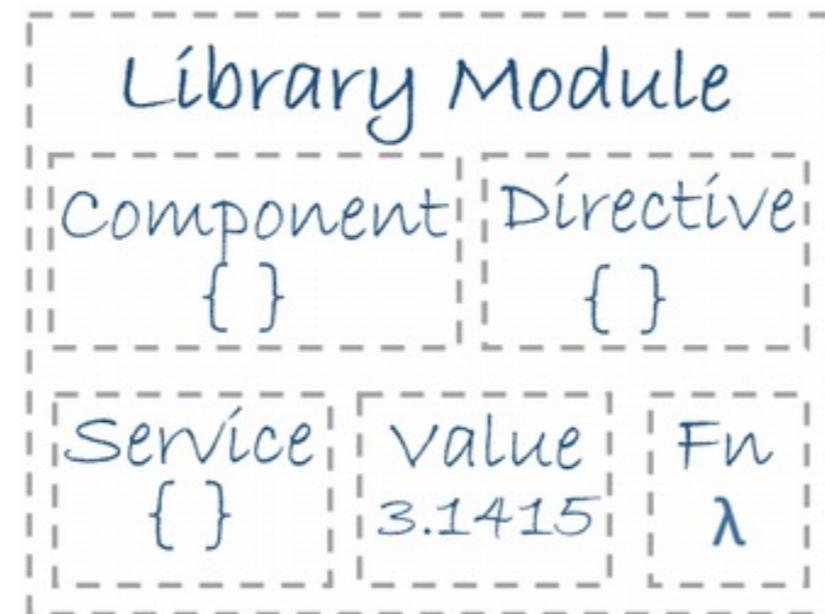
Component
{}
}

equivalent to decide which classes can be imported from a jar (Public Private for a module)

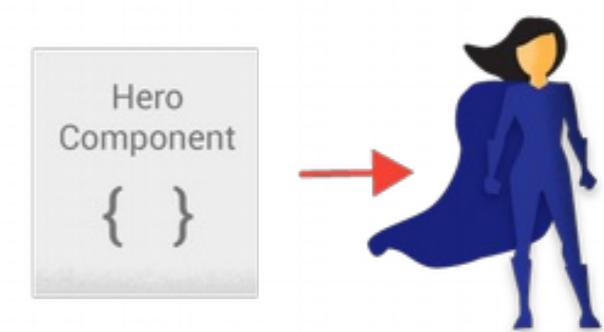
LIBRARY MODULES

```
import {Component} from 'angular2/core';
```

- Angular apps are composed of modules.
- Modules export things — classes, function, values — that other modules import.
- We prefer to write our application as a collection of modules, each module exporting one thing.



COMPONENT



```
export class HeroListComponent implements OnInit {  
  constructor(private _service: HeroService){ }  
  heroes: Hero[];  
  selectedHero: Hero;  
  ngOnInit(){  
    this.heroes = this._service.getHeroes();  
  }  
  selectHero(hero: Hero) { this.selectedHero = hero; }  
}
```

app/hero-list.component.ts

TEMPLATE



```
<h2>Hero List</h2>

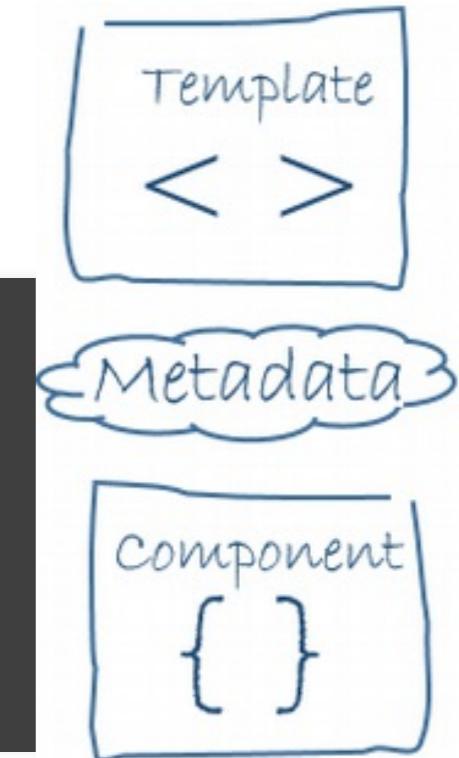
<p><i>Pick a hero from the list</i></p>
<div *ngFor="#hero of heroes" (click)="selectHero(hero)">
  {{hero.name}}
</div>

<hero-detail *ngIf="selectedHero" [hero]="selectedHero">
</hero-detail>
```

app/hero-list.component.html

METADATA

```
@Component({  
  selector: 'hero-list',  
  templateUrl: 'app/hero-list.component.html',  
  directives: [HeroDetailComponent],  
  providers: [HeroService]  
})  
export class HeroesComponent { ... }
```



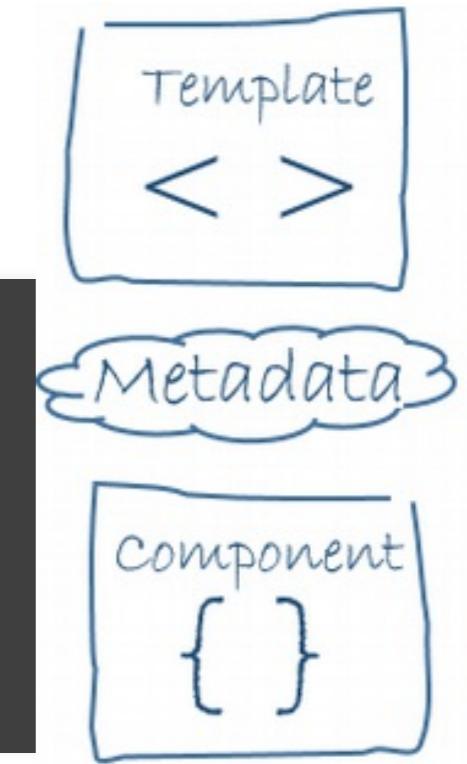
- **selector** : a css selector that tells Angular to create and insert an instance of this component where it finds a tag in parent HTML.

```
<hero-list></hero-list>
```

- Angular inserts an instance of the HeroListComponent view between tags.
- **templateUrl**: the address of this component's template

METADATA

```
@Component({  
  selector: 'hero-list',  
  templateUrl: 'app/hero-list.component.html',  
  directives: [HeroDetailComponent],  
  providers: [HeroService]  
})  
export class HeroesComponent { ... }
```

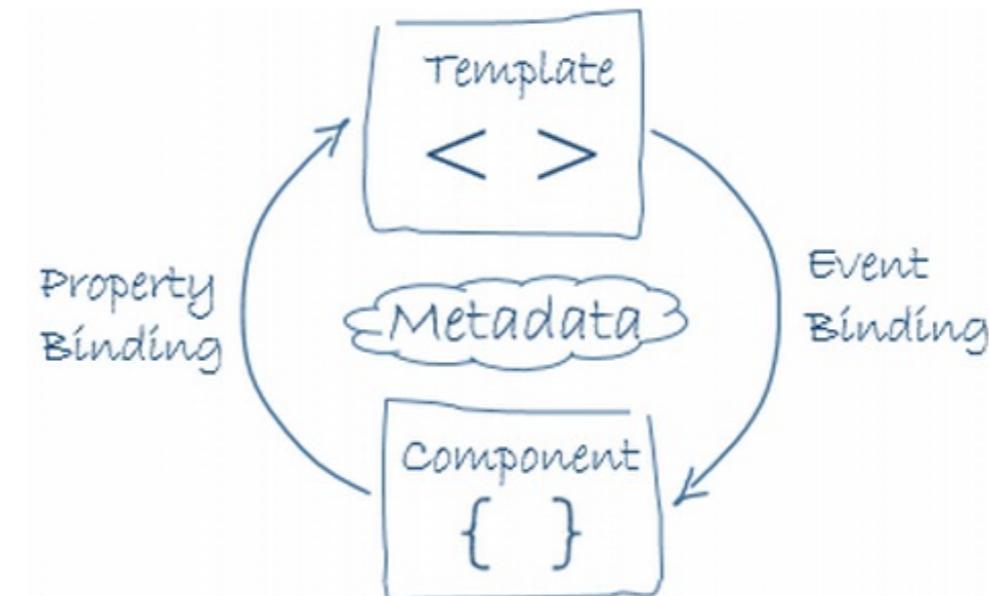
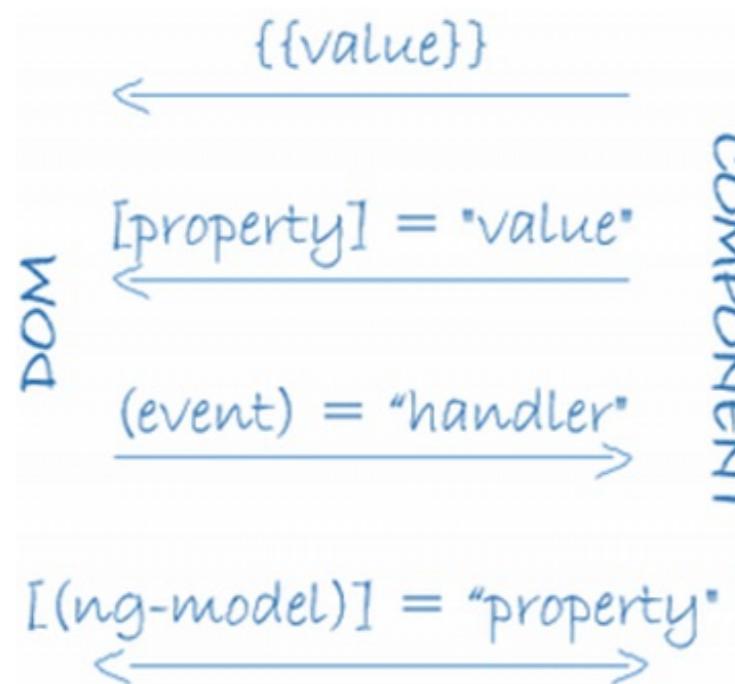


- **directives:** an array of the Components or Directives this template requires
- **providers:** an array of dependency injection providers for services that the component requires

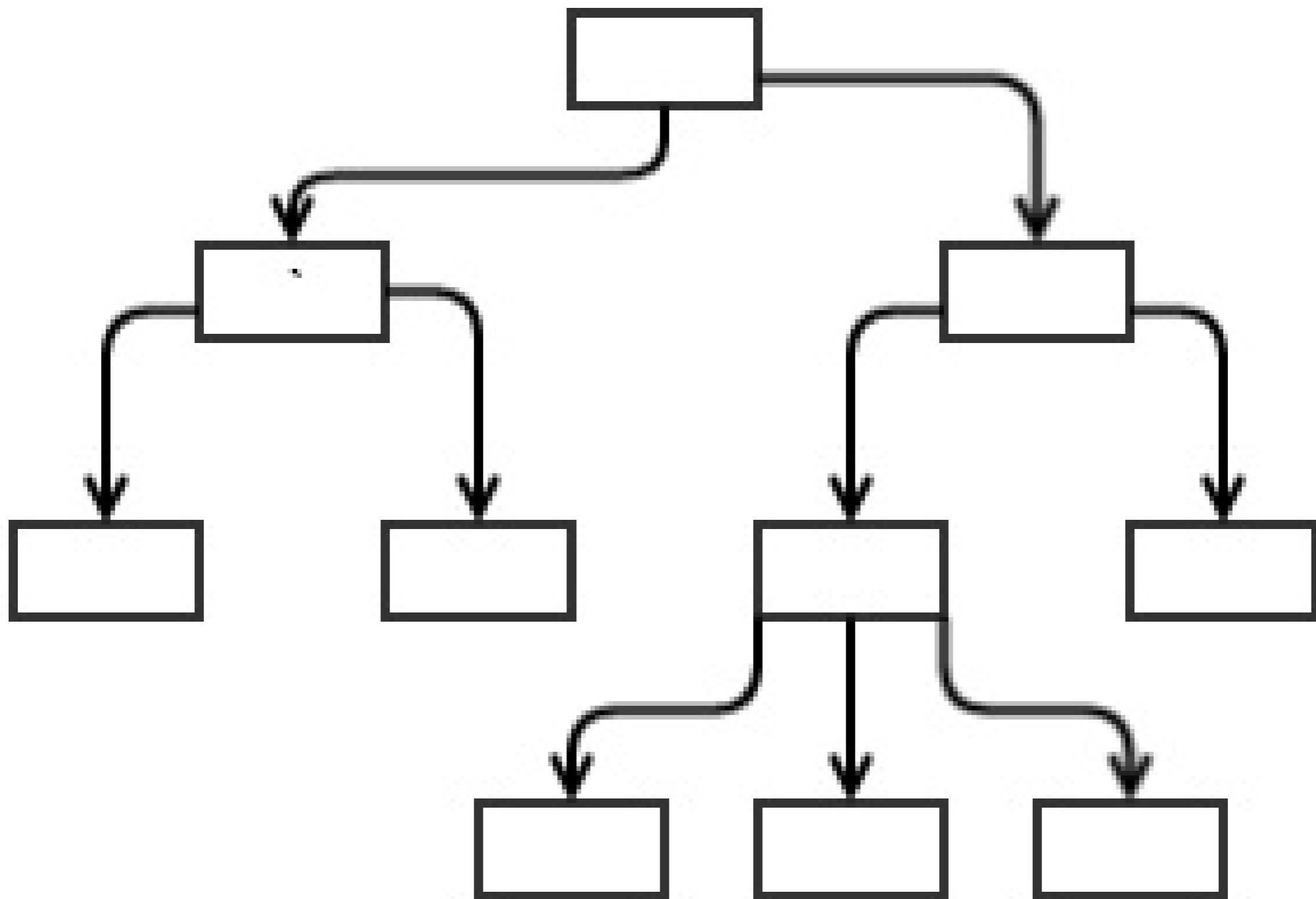
DATA BINDING

```
<div>{{hero.name}}</div>
<hero-detail [hero]="selectedHero"></hero-detail>
<div (click)="selectHero(hero)"></div>
```

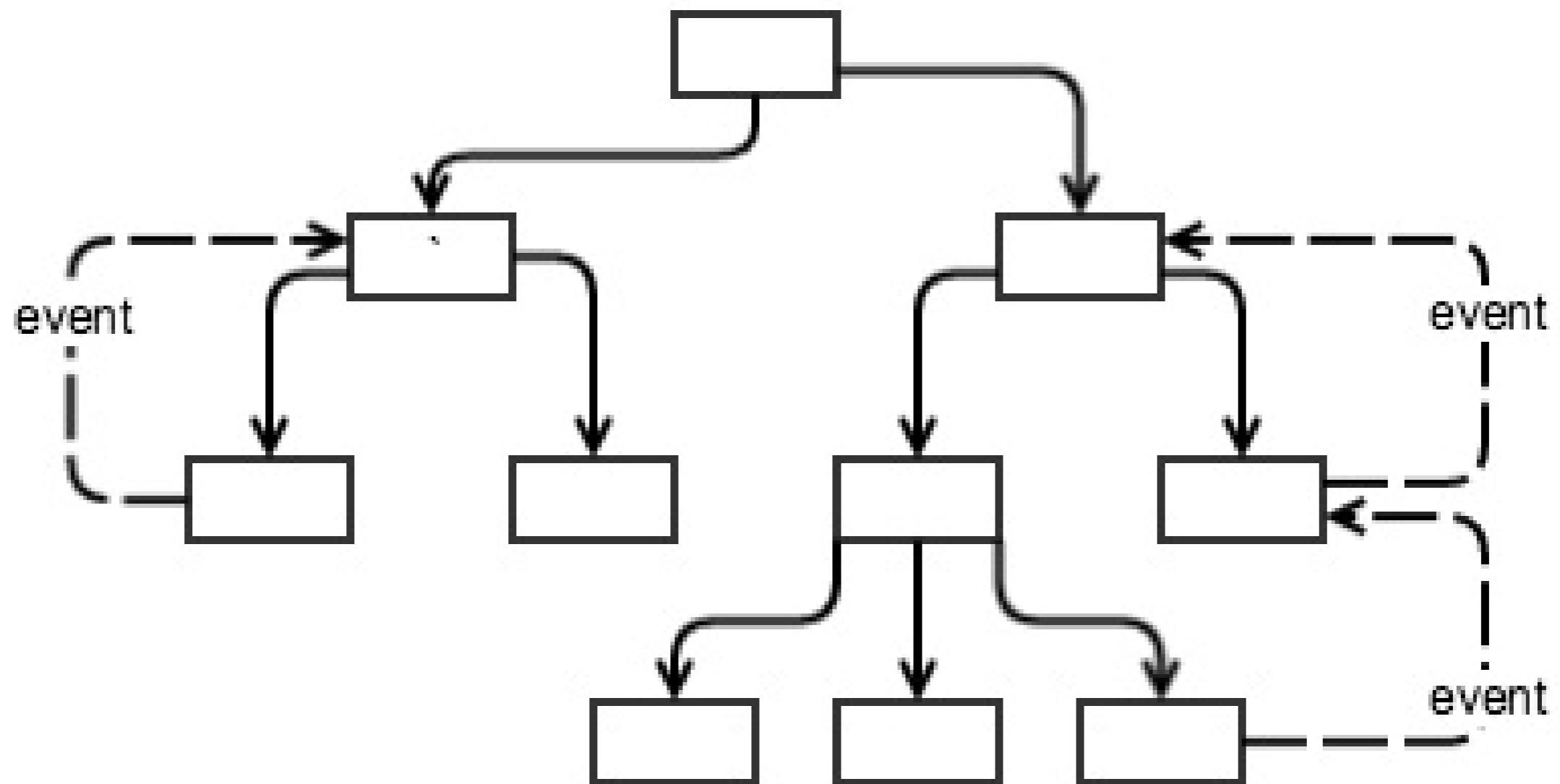
app/hero-list.component.ts



COMPONENT INTERACTION



COMPONENT INTERACTION



COMPONENT INTERACTION

“Inputs”, as you might guess from the hierarchy discussion above, specifies which properties you can set on a component whereas “outputs” identifies the events a component can fire to send information up the hierarchy to its parent.

EXAMPLE: COMPONENT

```
import {Component} from 'angular2/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
  `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

app.ts

```
<body>
  <my-app>Loading...</my-app>
</body>
```

index.html

EXAMPLE: SHOW THE LIST OF HEROES

```
export class AppComponent {  
  title = 'Tour of Heroes';  
  heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];  
  myHero = this.heroes[0];  
}
```

Component

```
<h1>{{title}}</h1>  
<h2>My favorite hero is: {{myHero.name}}</h2>  
<p>Heroes:</p>  
<ul>  
  <li *ngFor="#hero of heroes">  
    {{ hero.name }}  
  </li>  
</ul>  
<p *ngIf="heroes.length > 3">There are many heroes!</p>
```

template

EXAMPLE: CREATE CLASS FOR DATA AND FILL IT

```
export class Hero {  
  constructor(  
    public id:number,  
    public name:string) { }  
}
```

hero.ts

```
export class AppComponent {  
  title = 'Tour of Heroes';  
  heroes = [  
    new Hero(1, 'Windstorm'), new Hero(13, 'Bombasto'),  
    new Hero(15, 'Magneta'), new Hero(20, 'Tornado')  
  ];  
  myHero = this.heroes[0];  
}
```

app.component.ts:

EXAMPLE: WORK WITH THE EVENTS

```
@Component({
  selector: 'click-me',
  template: `
    <button (click)="onClickMe()">Click me!</button>
    {{clickMessage}}
  `})
export class ClickMeComponent {
  clickMessage = '';
  onClickMe() {
    this.clickMessage ='You are my hero!';
  }
}
```

EXAMPLE: ADDING HERO FORM

```
@Component({
  selector: 'little-tour',
  template: `
    <input #newHero
      (keyup.enter)="addHero(newHero.value)"
      (blur)="addHero(newHero.value); newHero.value=' '"
    <button (click)=addHero(newHero.value)>Add</button>
    <ul><li *ngFor="#hero of heroes">{{hero}}</li></ul>
  `
})
export class LittleTourComponent {
  heroes=['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
  addHero(newHero:string) {
    if (newHero) {
      this.heroes.push(newHero);
    }
  }
}
```

EXAMPLE: EXECUTE

```
import {bootstrap} from 'angular2/platform/browser'  
import {AppComponent} from './app.component'  
bootstrap(AppComponent);
```

boot.ts

```
name, version  
dependencies (SystemJS is used as module system)  
devDependencies
```

package.json

TypeScript configuration

tsconfig.json

npm start

BINDING IN TEMPLATES

Data Direction	Syntax	Binding Type
One way from data source to view target	<code>{{expression}}</code> <code>[target] = "expression"</code> <code>bind-target = "expr"</code>	Interpolation Property Attribute Class Style
One way from view target to data source	<code>(target) = "expression"</code> <code>on-target = "expr"</code>	Event
Two way	<code>[(target)] = "expr"</code> <code>bindon-target = "expr"</code>	Two-way

BINDING TARGETS

Binding Type	Target	Examples
Property	Element Property Component Property Directive property	<pre></pre> <pre><hero-detail [hero]="currentHero"></hero- detail></pre> <pre><div [ngClass] = "{selected: isSelected}"> </div></pre>
Event	Element Event Component Event Directive Event	<pre><button (click) = "onSave()">Save</button></pre> <pre><hero-detail (deleted)="onHeroDeleted()"> </hero-detail></pre> <pre><div myClick(myClick)="clicked=\$event">click me</div></pre>

BINDING TARGETS

Binding Type	Target	Examples
Two-way	Directive Event Property	<code><input [(ngModel)]="heroName"></code>
Attribute	Attribute (the exception)	<code><button [attr.aria-label]="help"> help</button></code>
Class	class Property	<code><div [class.special]="isSpecial"> Special</div></code>
Style	style Property	<code><button [style.color] = "isSpecial ? 'red' : 'green'"></code>

DEPENDENCY INJECTION

```
constructor(private _service: HeroService) { ... }
```



But how Injector get to know about service?

2 ways:

```
bootstrap(AppComponent, [HeroService, Logger]);
```

app/boot.ts

```
@Component({
  providers: [HeroService]
})
export class HeroesComponent { ... }
```

app/hero-list.component.ts

(we get a new instance of the service with each new instance of that component)

EXAMPLE: DEFINE MOCK HEROSERVICE

```
import {HEROES} from './mock-heroes';
import {Injectable} from 'angular2/core';

@Injectable()
export class HeroService {
  getHeroes() {
    return HEROES;
  }
}
```

hero.service.ts:

```
import {Hero} from './hero';
export var HEROES: Hero[] = [
  {"id": 11, "name": "Mr. Nice"},
  {"id": 12, "name": "Narco"}
]
```

mock-heroes.ts

EXAMPLE: DEFINE HEROSERVICE WITH PROMISES

```
import {Injectable} from 'angular2/core';
import {Hero} from './hero';
import {HEROES} from './mock-heroes';

@Injectable()
export class HeroService {
  getHeroes() {
    return Promise.resolve(HEROES);
  }
  // See the "Take it slow" appendix
  getHeroesSlowly() {
    return new Promise<Hero[]>(resolve =>
      setTimeout(()=>resolve(HEROES), 2000) // 2 seconds
    );
  }
}
```

EXAMPLE: USE HEROSERVICE IN COMPONENT

```
import {Component, OnInit} from 'angular2/core';
import {HeroService} from './hero.service';
@Component({
  selector: 'my-app',
  template: `...`,
  providers: [HeroService]
})
export class AppComponent implements OnInit {
  heroes:Hero[];
  constructor(private _heroService:HeroService) { }
  getHeroes() {
    this._heroService.getHeroes()
      .then(heroes => this.heroes = heroes);
  }
  ngOnInit() { this.getHeroes(); }
}
```

SERVICE

```
export class HeroService {  
  constructor(private _backend:BackendService,  
             private _logger:Logger) {}  
  private _heroes:Hero[] = [];  
  
  getHeroes() {  
    this._backend.getAll(Hero).then((heroes:Hero[]) => {  
      this._logger.log(`Fetched ${heroes.length} heroes.`);  
      this._heroes.push(...heroes); // fill cache  
  
    });  
    return this._heroes;  
  }  
}
```

app/hero.service.ts

FORMS

```
<div class="form-group">
  <label for="power">Hero Power</label>
  <select class="form-control" required>
    <option *ngFor="#p of powers" [value]="p">{{p}}</option>
  </select>
</div>
```

NgControl

```
<input type="text" class="form-control"
  required
  [(ngModel)]="model.name"
  ngControl="name" >
```

FORM VALIDATION

State	Class if true	Class if false
Control has been visited	ng-touched	ng- untouched
Control's value has changed	ng-dirty	ng-pristine
Control's value is valid	ng-valid	ng-invalid

```
.ng-valid[required] {  
  border-left: 5px solid #42A948; /* green */  
}  
.ng-invalid {  
  border-left: 5px solid #a94442; /* red */  
}
```

SHOW VALIDATION MESSAGES

```
<input type="text" class="form-control" required  
[(ngModel)]="model.name"  
    ngControl="name" #name="ngForm" >  
<div [hidden]="name.valid" class="alert alert-danger">  
    Name is required  
</div>
```

ROUTING

```
@Component({...})  
@RouteConfig([  
  { path: '/crisis-center',  
    name: 'CrisisCenter',  
    component: CrisisListComponent },  
  { path: '/heroes',  
    name: 'Heroes',  
    useAsDefault: true,  
    component: HeroListComponent },  
  { path: '/hero/:id',  
    name: 'HeroDetail',  
    component: HeroDetailComponent }  
])  
export class AppComponent {}
```

```
<!-- Routed views go here -->  
<router-outlet></router-outlet>
```

ROUTING: DEFINING IN MAIN APP COMPONENT

```
import { RouteConfig, ROUTER_DIRECTIVES, ROUTER_PROVIDERS }  
  from 'angular2/router';  
  
@Component({  
  selector: 'my-app',  
  template: `  
    <a [routerLink]=["['Heroes']">Heroes</a>  
    <a [routerLink]=["['CrisisCenter']">Crisis Center</a>  
    <router-outlet></router-outlet> `,  
  directives: [ROUTER_DIRECTIVES],  
  providers: [ROUTER_PROVIDERS, HeroService]  
})  
@RouteConfig([  
  { path: '/heroes', name: 'Heroes', component: HeroesComponent }  
])  
export class AppComponent {}
```

ROUTING: HEROES COMPONENT

```
@Component({
  selector: 'my-heroes',
  templateUrl: 'app/heroes.component.html',
  styleUrls: ['app/heroes.component.css'],
  directives: [HeroDetailComponent]
})
export class HeroesComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;
  constructor( private _router: Router,
    private _heroService: HeroService) {}
  getHeroes() {
    this._heroService.getHeroes().then(heroes =>
      this.heroes = heroes);
  }
  ngOnInit() { this.getHeroes(); }
  onSelect(hero: Hero) { this.selectedHero = hero; }
  gotoDetail() {
    this._router.navigate(['HeroDetail', { id:
      this.selectedHero.id }]);
  }
}
```

ROUTING: HEROES TEMPLATE

```
<h2>My Heroes</h2>
<ul class="heroes">
  <li *ngFor="#hero of heroes"
      [class.selected]="hero === selectedHero"
      (click)="onSelect(hero)">
    <span class="badge">{{hero.id}}</span> {{hero.name}}
  </li>
</ul>

<div *ngIf="selectedHero">
  <h2>
    {{selectedHero.name | uppercase}} is my hero
  </h2>
  <button (click)="gotoDetail()">View Details</button>
</div>
```

MR. NICE is my hero

[View Details](#)

ROUTING: HERO DETAILS COMPONENT

```
@Component({
  selector: 'my-hero-detail',
  templateUrl: 'app/hero-detail.component.html',
})
export class HeroDetailComponent implements OnInit {
  hero: Hero;
  constructor(
    private _heroService: HeroService,
    private _routeParams: RouteParams) {
  }
  ngOnInit() {
    let id = +this._routeParams.get('id');
    this._heroService.getHero(id)
      .then(hero => this.hero = hero);
  }
}
```

app/hero-detail.component.html

ROUTING: HERO DETAILS TEMPLATE

```
<div *ngIf="hero">
  <h2>{{hero.name}} details!</h2>
  <div>
    <label>id: </label>{{hero.id}}</div>
  <div>
    <label>name: </label>
    <input [(ngModel)]="hero.name" placeholder="name" />
  </div>
  <button (click)="goBack()">Back</button>
</div>
```

hero-detail.component.html

USING PIPES

```
import {Component} from 'angular2/core'

@Component({
  selector: 'hero-birthday',
  template: `<p>The hero's birthday is
    {{ birthday | date }}</p>`
})
export class HeroBirthday {
  birthday = new Date(1988,3,15); // April 15, 1988
}
```

DEFINE CUSTOM PIPE

```
import {Pipe} from 'angular2/core';

/* Raise the value exponentially
 * Takes an exponent argument that defaults to 1.
 * Usage: value | exponentialStrength:exponent
 * Example:
 *   {{ 2 | exponentialStrength:10}}
 * formats to: 1024
 */
@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe {
  transform(value:number, args:string[]) : any {
    return Math.pow(value, parseInt(args[0] || '1', 10));
  }
}
```

USE CUSTOM PIPE

```
import {Component} from 'angular2/core';
import {ExponentialStrengthPipe} from './exponential-strength.pipe';

@Component({
  selector: 'power-booster',
  template: `
    <h2>Power Booster</h2>
    <p>
      Super power boost: {{2 | exponentialStrength: 10}}
    </p>
    `,
  pipes: [ExponentialStrengthPipe]
})
export class PowerBooster { }
```

USE ASYNC PIPE

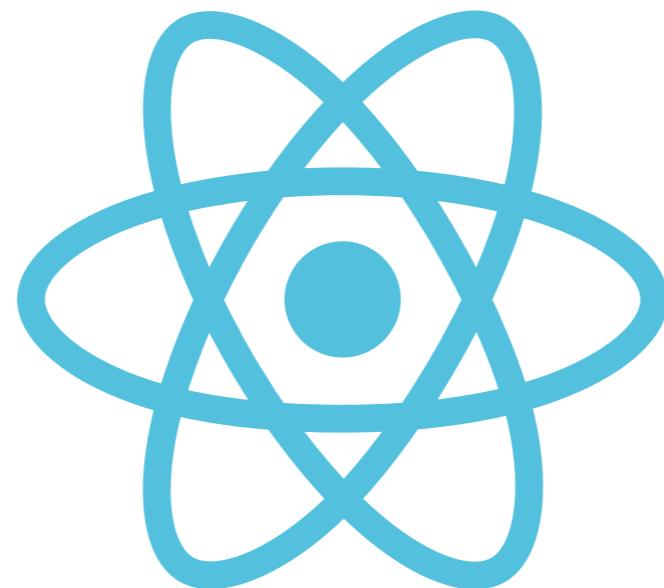
```
import {Component} from 'angular2/core';

// Initial view: "Message: "
// After 500ms: Message: You are my Hero!"

@Component({
  selector: 'hero-message',
  template: 'Message: {{delayedMessage | async}}',
})
export class HeroAsyncMessageComponent {
  delayedMessage:Promise<string> =
    new Promise((resolve, reject) => {
      setTimeout(() => resolve('You are my Hero!'), 500);
    });
}
```

REACT

A LIBRARY FOR CREATING USER INTERFACES.



WHAT IS REACT.JS

- React is a JavaScript library created by a collaboration of Facebook and Instagram
- It's not MVC, only the "V" in "MVC"
- Big Companies use React
 - Facebook, Instagram, Yahoo!, Airbnb, Sony, Netflix

WHY WAS REACT.JS MADE?

- React isn't an MVC framework
- React does not use templates
- React updates are dead simple (Reactive pattern)
- HTML is just the beginning

HOW IS REACT.JS USED?

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.6.1/react.js"> </script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.6.1/react-dom.js"> </script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.38/browser.min.js"> </script>
<script src="app.js" type="text/babel"> </script>

<div id="root"></div>
```

Root DOM node

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Render React element into root DOM node

HOW IS REACT.JS USED?

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.6.1/react.js"> </script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.6.1/react-dom.js"> </script>
<div id="root"></div>
```

Root DOM node

```
const element = React.createElement('h1', null, 'Hello react');
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Render React element into root DOM node

UPDATE THE RENDERED ELEMENT

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(
    element,
    document.getElementById('root')
  );
}
setInterval(tick, 1000);
```

HOW THE TECHNOLOGY WORKS

- Elements, Component, and Properties
- Component Life Cycle
- JSX
- States
- Virtual DOM
- One Way Data Flow

RULE 1: BUILD COMPONENTS, NOT TEMPLATES.

**WE ALL LIKE SEPARATION OF
CONCERNS, RIGHT?**

SEPARATION OF CONCERNS:

Reduce coupling, increase cohesion.

COUPLING IS:

The degree to which each program module relies on each of the other modules.

[wikipedia \(Coupling\)](#))

COHESION IS:

"The degree to which elements of a module belong together." [wikipedia \(Cohesion\)](#))

TEMPLATES ENCOURAGE A POOR SEPARATION OF CONCERNS.

So are Angular-style directives.

"VIEW MODEL" TIGHTLY COUPLES TEMPLATE TO DISPLAY LOGIC.

```
[{"price": "7.99", "product": "Back scratcher", "tableRowColor": "rgba(0, 0, 0, 0.5)"}]
```

DISPLAY LOGIC AND MARKUP ARE INEVITABLY TIGHTLY COUPLED.

How do you find DOM nodes?

DISPLAY LOGIC AND MARKUP ARE HIGHLY COHESIVE.

They both show the UI.

TEMPLATES SEPARATE TECHNOLOGIES, NOT CONCERNS.

And they do it by being deliberately underpowered.

SYMPTOMS THAT YOUR FRONT-END TECHNOLOGY IS UNDERPOWERED:

Reliance on primitive abstractions (like {{> }} and {{#each }}).

SYMPTOMS THAT YOUR FRONT-END TECHNOLOGY IS UNDERPOWERED:

Inventing lots of new concepts (that already exist in
JavaScript).

FROM THE ANGULAR DIRECTIVES DOCS:

However isolated scope creates a new problem: if a transcluded DOM is a child of the widget isolated scope then it will not be able to bind to anything. For this reason the transcluded scope is a child of the original scope, before the widget created an isolated scope for its local variables. This makes the transcluded and widget isolated scope siblings.

THE FRAMEWORK CANNOT KNOW HOW TO SEPARATE YOUR CONCERNS FOR YOU.

It should only provide **powerful, expressive tools** for the user to do it correctly.

THIS TOOL IS A REACT COMPONENT.

A highly cohesive building block for UIs loosely coupled
with other components.

AS AN ARCHITECT, USE COMPONENTS TO SEPARATE YOUR CONCERNS.

With the full power of JavaScript, not a crippled templating language.

COMPONENTS ARE REUSABLE.

COMPONENTS ARE COMPOSABLE.

COMPONENTS ARE UNIT TESTABLE.

They are, after all, units.

WHAT ABOUT SPAGHETTI CODE?

JUST DON'T WRITE SPAGHETTI CODE.

Keep your components small.

JUST DON'T WRITE SPAGHETTI CODE.

Only put *display logic* in your components.

JUST DON'T WRITE SPAGHETTI CODE.

With great power comes great responsibility – Uncle Ben in Spiderman

WHAT ABOUT WORKING WITH DESIGNERS?

JSX IS AN *OPTIONAL* PREPROCESSOR TO LET YOU USE HTML-LIKE SYNTAX.

```
<a href="http://olivier.barais.fr">  
  my personal web site  
</a>
```

JSX IS AN *OPTIONAL* PREPROCESSOR TO LET YOU USE HTML-LIKE SYNTAX.

```
React.DOM.a(  
  {href: 'http://olivier.barais.fr'},  
  'my personal web site'  
)
```

**WITH JSX, IT'S EASY FOR DESIGNERS
TO CONTRIBUTE CODE.**

THE ACCESSIBILITY OF TEMPLATES AND THE POWER OF JAVASCRIPT.

RULE 2. RE-RENDER THE WHOLE APP ON EVERY UPDATE

The key design decision that makes React awesome.

BUILDING UI'S IS HARD BECAUSE THERE IS SO MUCH STATE.

Lots of UI elements · Design iteration · Crazy environments ·
Mutable DOM · User input · etc etc

**DATA CHANGING OVER TIME IS THE
ROOT OF ALL EVIL.**

“Our intellectual powers are rather geared to master static relations and our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible” –

Dijkstra

IN THE 90S IT WAS EASIER.

Just refresh the page when the data changes.

DESIGN DECISION: WHEN THE DATA CHANGES, REACT RE-RENDERS THE ENTIRE COMPONENT.

THAT IS, REACT COMPONENTS ARE BASICALLY JUST IDEMPOTENT FUNCTIONS.

They describe your UI at any point in time, just like a server-rendered app.

RE-RENDERING ON EVERY CHANGE MAKES THINGS SIMPLE.

- Every place data is displayed is guaranteed to be up-to-date.
- No magical data binding.
- No model dirty checking.
- No more explicit DOM operations – **everything is declarative**

**"RE-RENDER ON EVERY CHANGE?
THAT SEEEMS EXPENSIVE."**

“And doesn't it mess up form fields and scroll position?”

RULE 3. VIRTUAL DOM

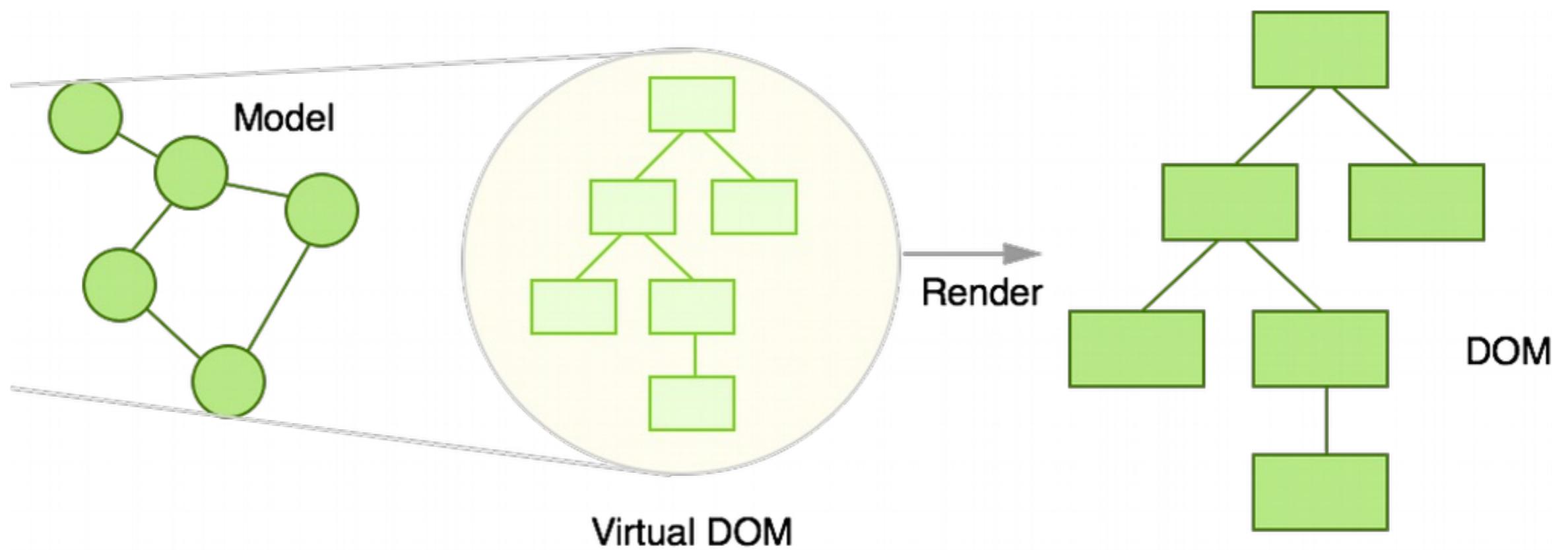
Makes re-rendering on every change fast.

**YOU CAN'T JUST THROW OUT THE DOM AND
REBUILD IT ON EACH UPDATE.**

It's too slow and you'll lose form state and scroll position.

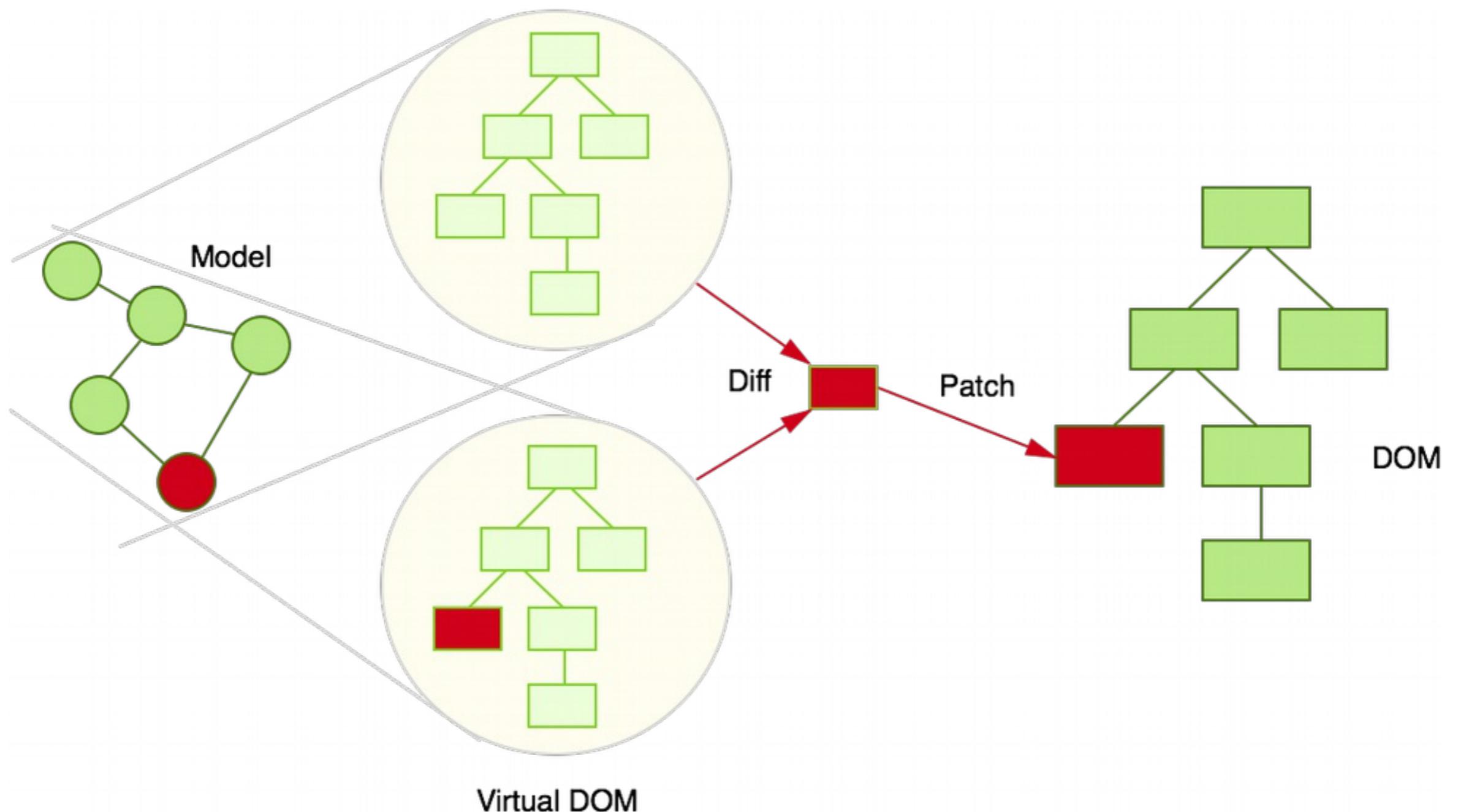
SO REACT BUILT A VIRTUAL DOM (AND EVENTS SYSTEM).

Optimized for performance and memory footprint



ON EVERY UPDATE...

- React builds a new virtual DOM subtree
- ... diffs it with the old one
- ... computes the minimal set of DOM mutations and puts them in a queue
- ... and batch executes all updates



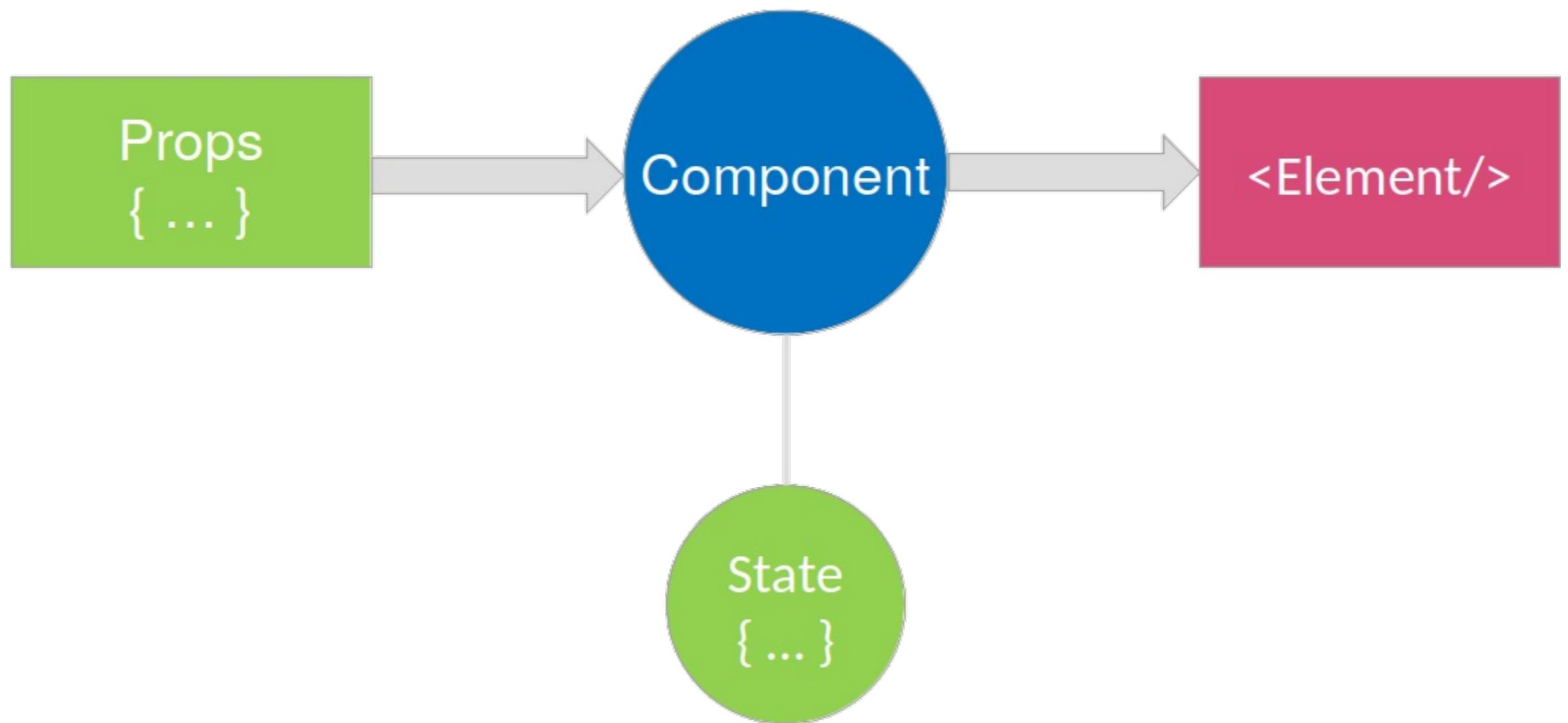
IT'S FAST!

Because the DOM is slow!

THE VIRTUAL DOM LETS US DO FUN THINGS.

- Testability for free
- SVG, VML and canvas support
- Running the whole app in a Web Worker

OTHER CONCEPTS



PROPS

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
const element = <Welcome name="Olivier" />;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

STATE

```
class Clock extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  
  }  
  componentDidMount() {  
    this.timerID = setInterval(() => this.tick(), 1000);  
  }  
  componentWillUnmount() { clearInterval(this.timerID);}  
  tick() {  
    this.setState({ date: new Date()});  
  }  
  render() {  
    return  
      <div>  
        <h1>Hello, world!</h1>  
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
      </div>  
  }  
}  
ReactDOM.render(  
  <Clock />,  
  document.getElementById('root')  
);
```

COMPONENT LIFECYCLE

- Mounting
- Updating
- Unmounting

MOUNTING

1. constructor()
2. componentWillMount()
3. render()
4. componentDidMount()

UPDATING

1. `componentWillReceiveProps()`
2. `shouldComponentUpdate()`
3. `componentWillUpdate()`
4. `render()`
5. `componentDidUpdate()`

UNMOUNTING

1. `componentWillUnmount()`

RULE 4: A FLOW IS UNIDIRECTIONAL.

All data flows down the component hierarchy

COMMUNICATION BETWEEN COMPONENTS

8 no-Flux strategies for React component communication

COMMUNICATION BETWEEN COMPONENTS

Use Flux or Redux

CONCLUSION ON REACT

- Components, not templates.
- Three main concepts:
 - Components
 - Props
 - State
- Re-render, don't mutate.
- Virtual DOM is **simple and fast**
- <https://facebook.github.io/react/>

CONCLUSION

- Commonalities between frameworks
 - usage of components

METHODOLOGY

- Start With A Mock
- Step 1: Break The UI Into A Component Hierarchy
- Step 2: Build A Static Version in React, Vue.js, Angular 2
- Step 3: Identify The Minimal (but complete)
Representation Of UI State
- Step 4: Identify Where Your State Should Live
- Step 5: Design your data-flow and the interactions
between components

PERSONAL RECOMMENDATION

- Use TypeScript for all frameworks, even on the server side

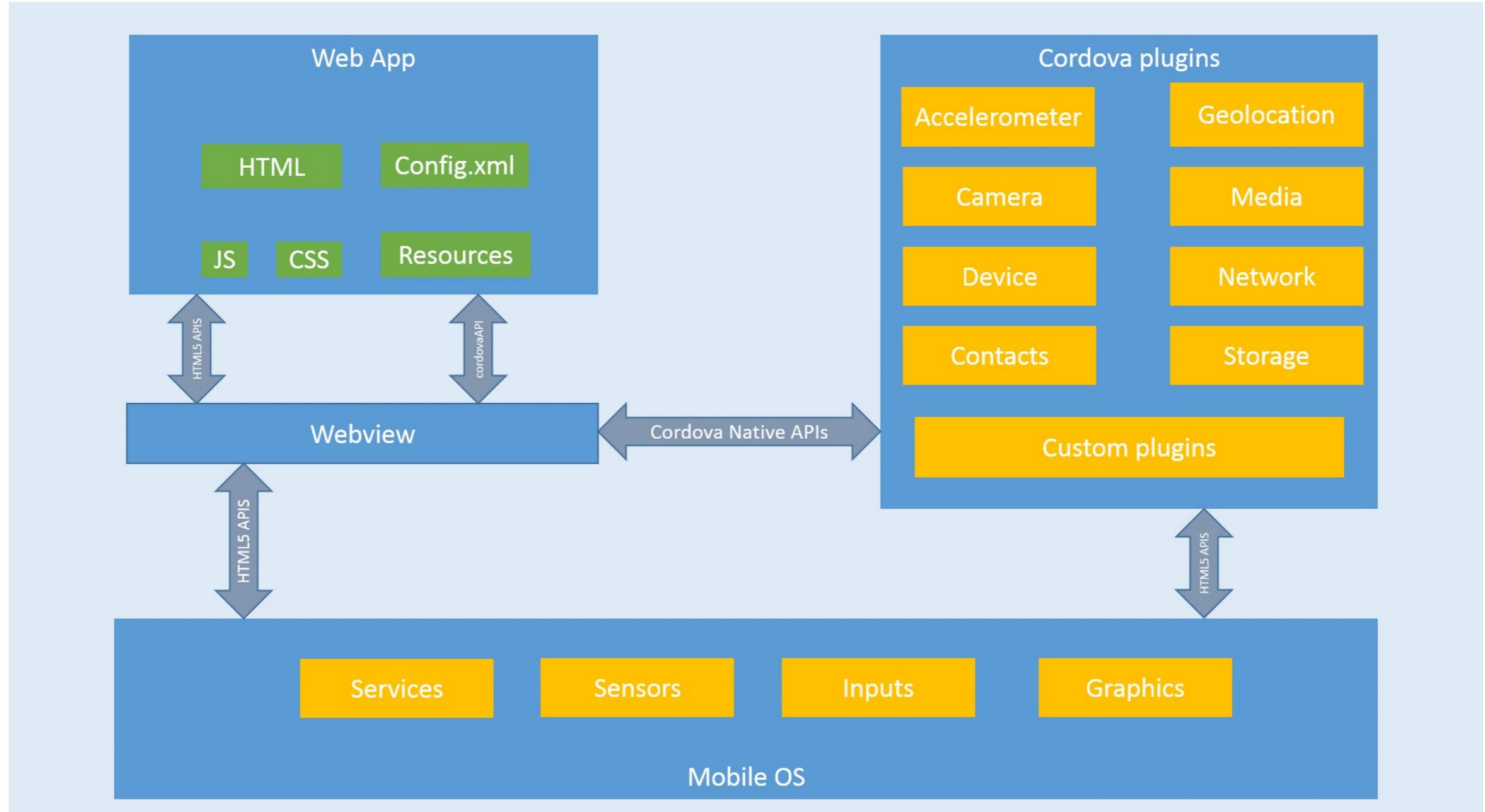
OTHER REMARKS

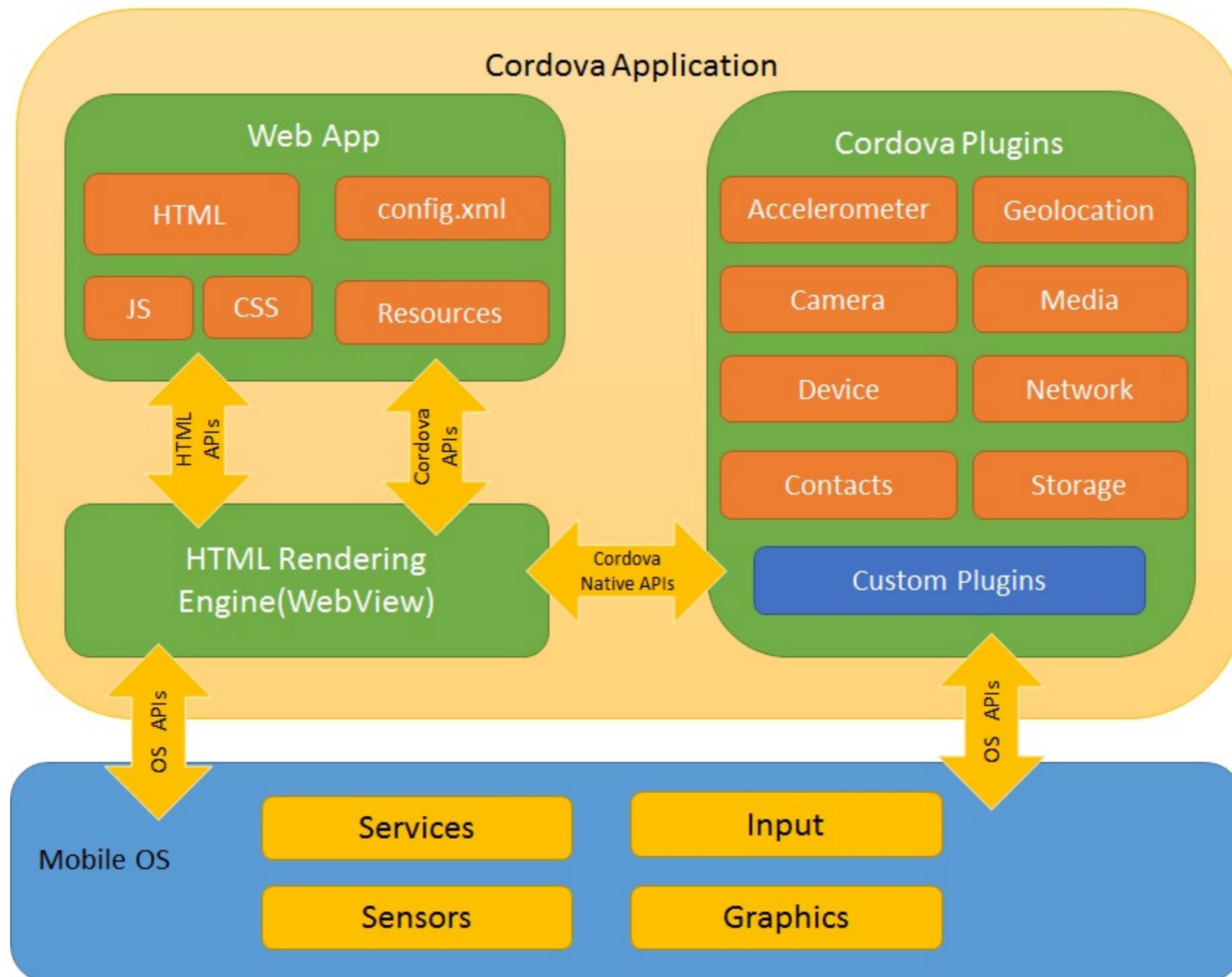
You can use [surge.sh](#) for deploying your SPA.

HYBRID DEVELOPMENT

- Build apps with web tech you know and love
- Ionic (AngularJS 2)
- ReactNative
- OnSen UI (Framework agnostic)
 - Written in pure JavaScript.
 - Support JQuery, AngularJS, AngularJS 2, React, Vue

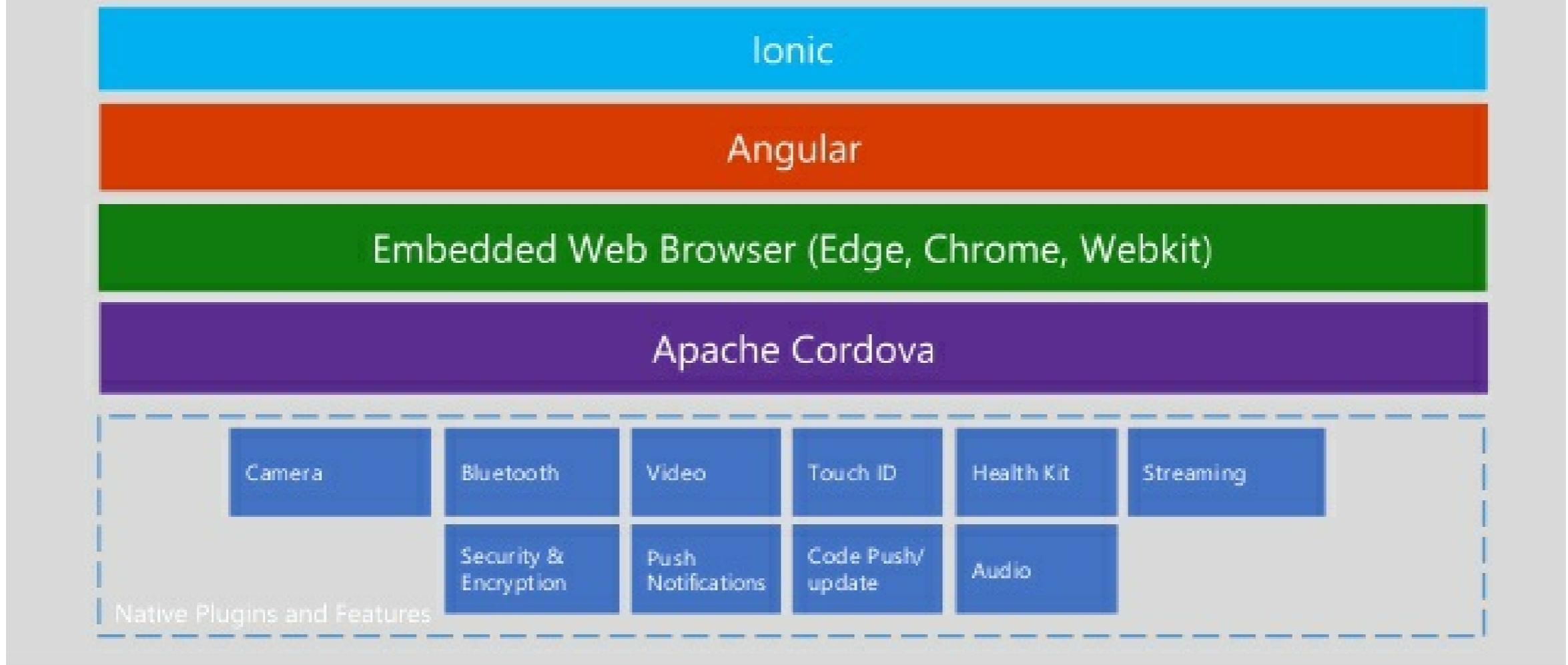
HYBRID DEVELOPMENT (APACHE CORDOVA)



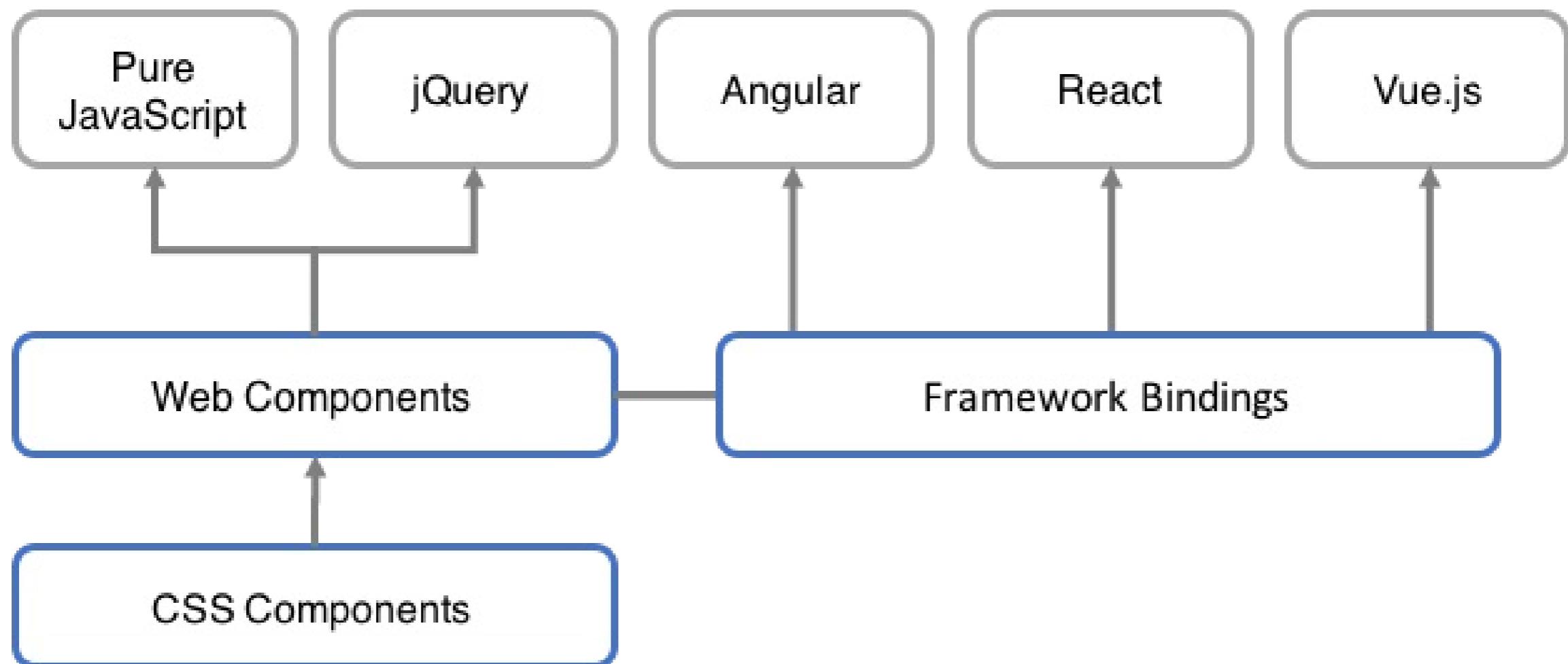


IONIC ARCHITECTURE

Full Runtime Architecture



ONSEN ARCHITECTURE



LINKS

- <https://vuejs.org/>
- <https://facebook.github.io/react/>
- <https://angular.io/>
- <https://angularjs.org/>
- <http://getbootstrap.com/>
- <https://jquery.com/>
- <https://facebook.github.io/react-native/>
- <https://ionicframework.com/>
- <https://cordova.apache.org/>
- <https://onsen.io/>
- <https://books.ninja-squad.com/>