ANGULAR 2 INTRO

LUXOFT

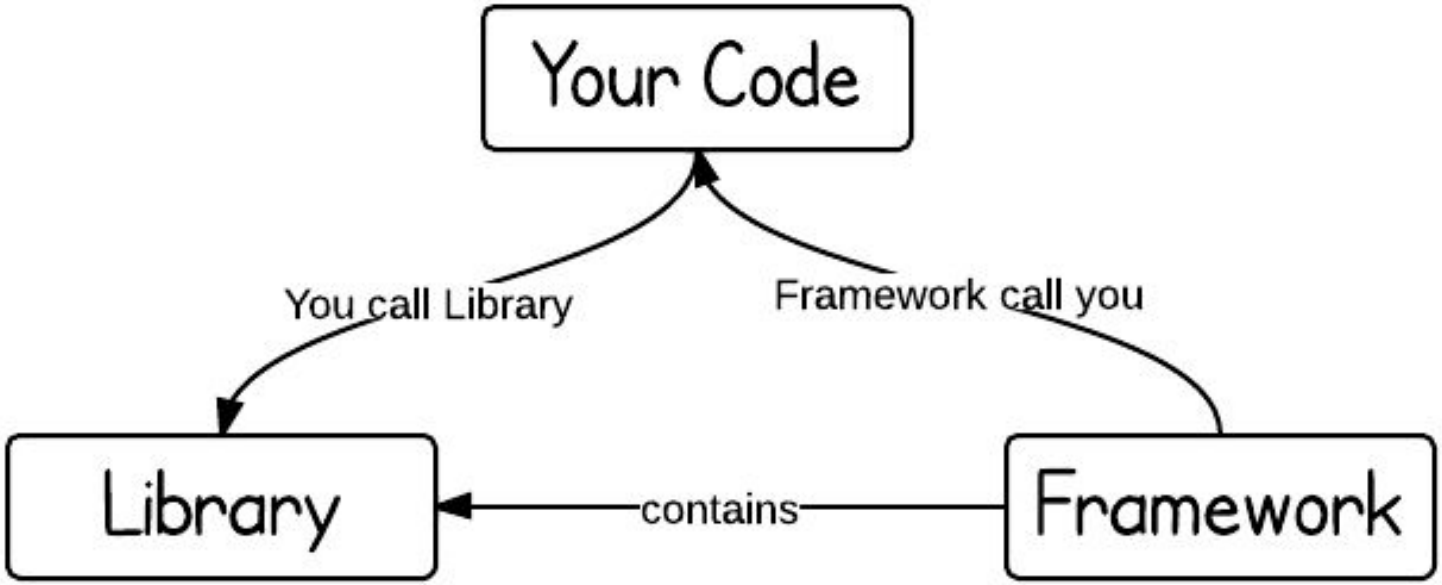
LXFT LISTED NYSE

WHY TO USE HTML5 FRAMEWORKS

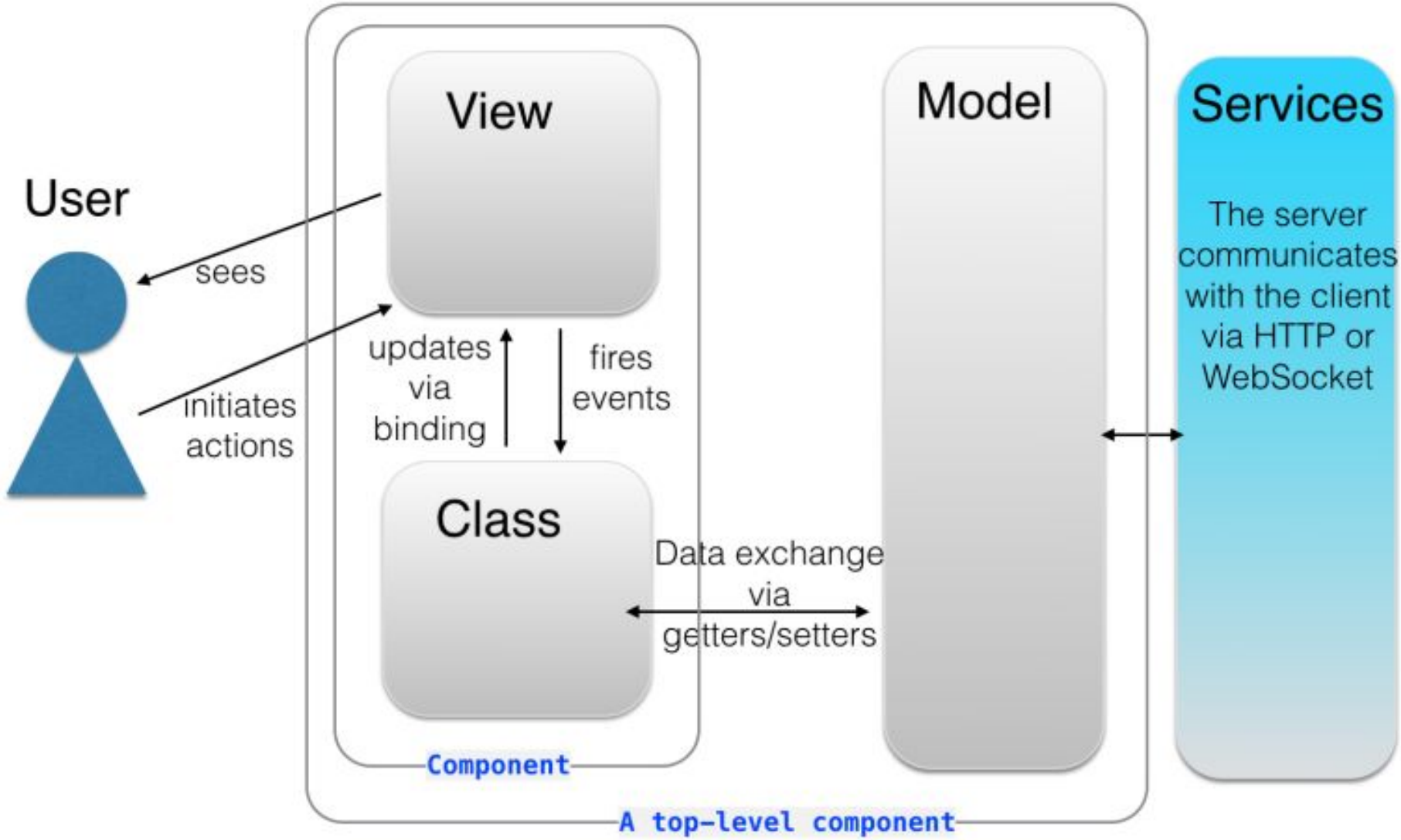
- Deal with cross-browser compatibility
- Make your application more structured
- May include reusable components
- Make programmers more productive
- Lowerthe amount of manually written code

FRAMEWORKS VS. LIBRARIES

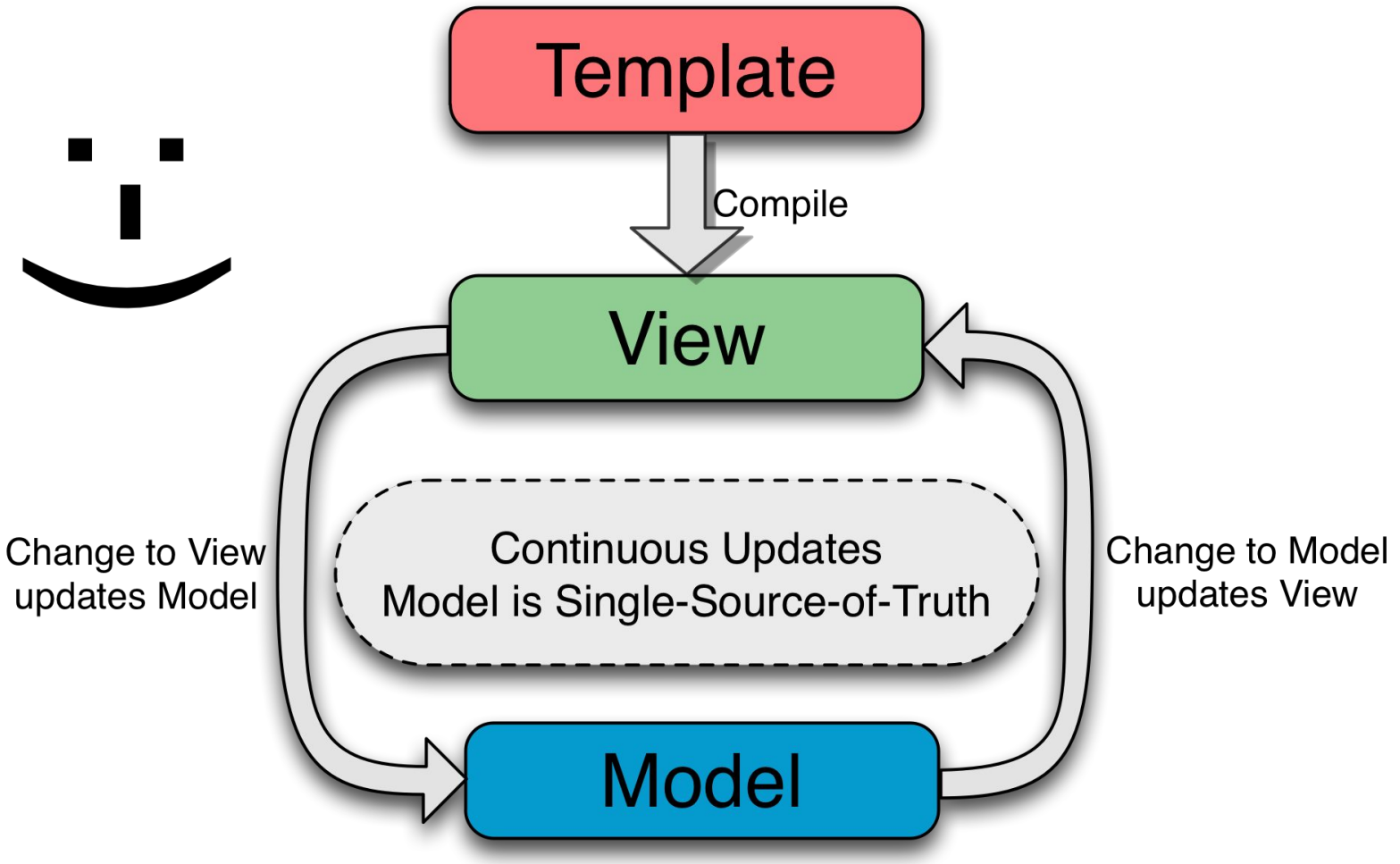
- Frameworks expect you to develop using well defined rules.
- Libraries just offer reusable components



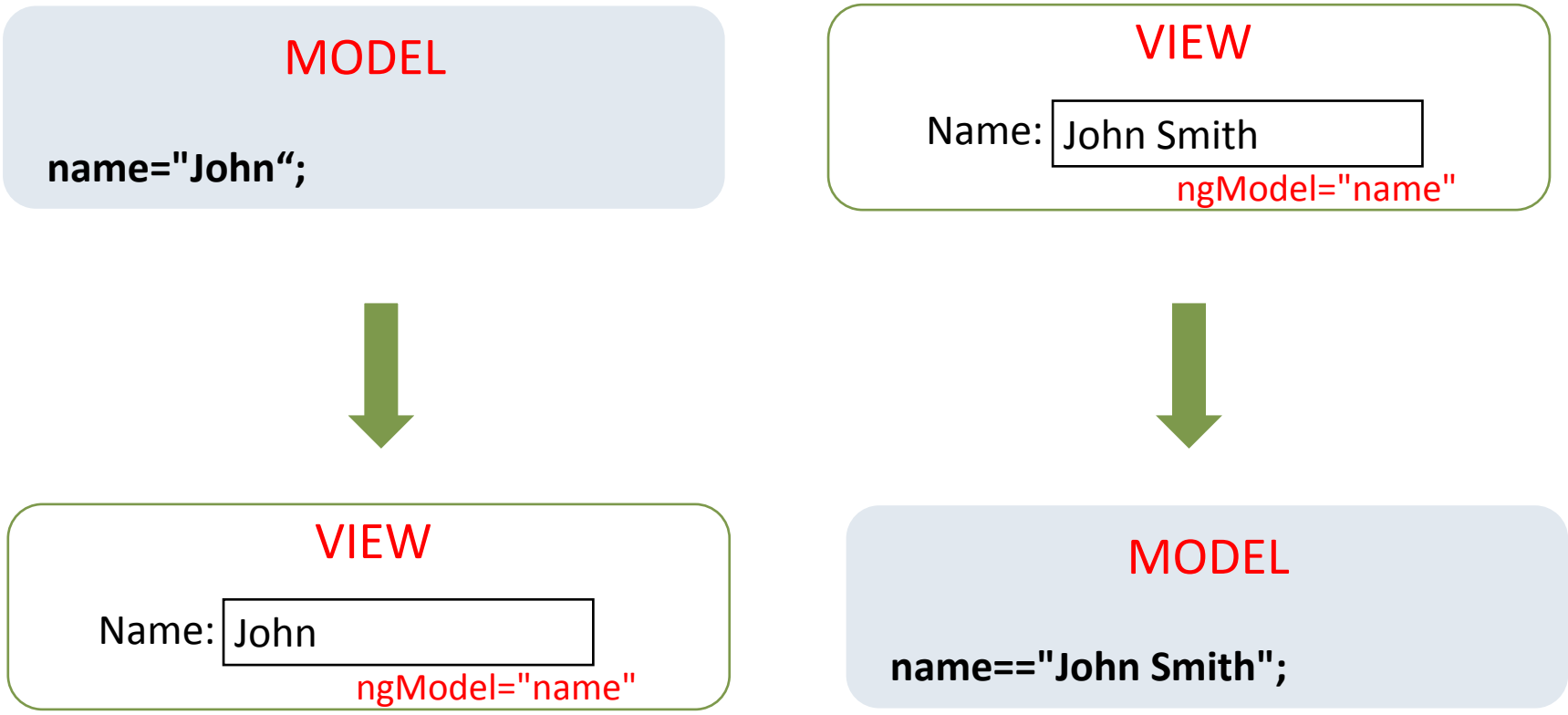
MVC model

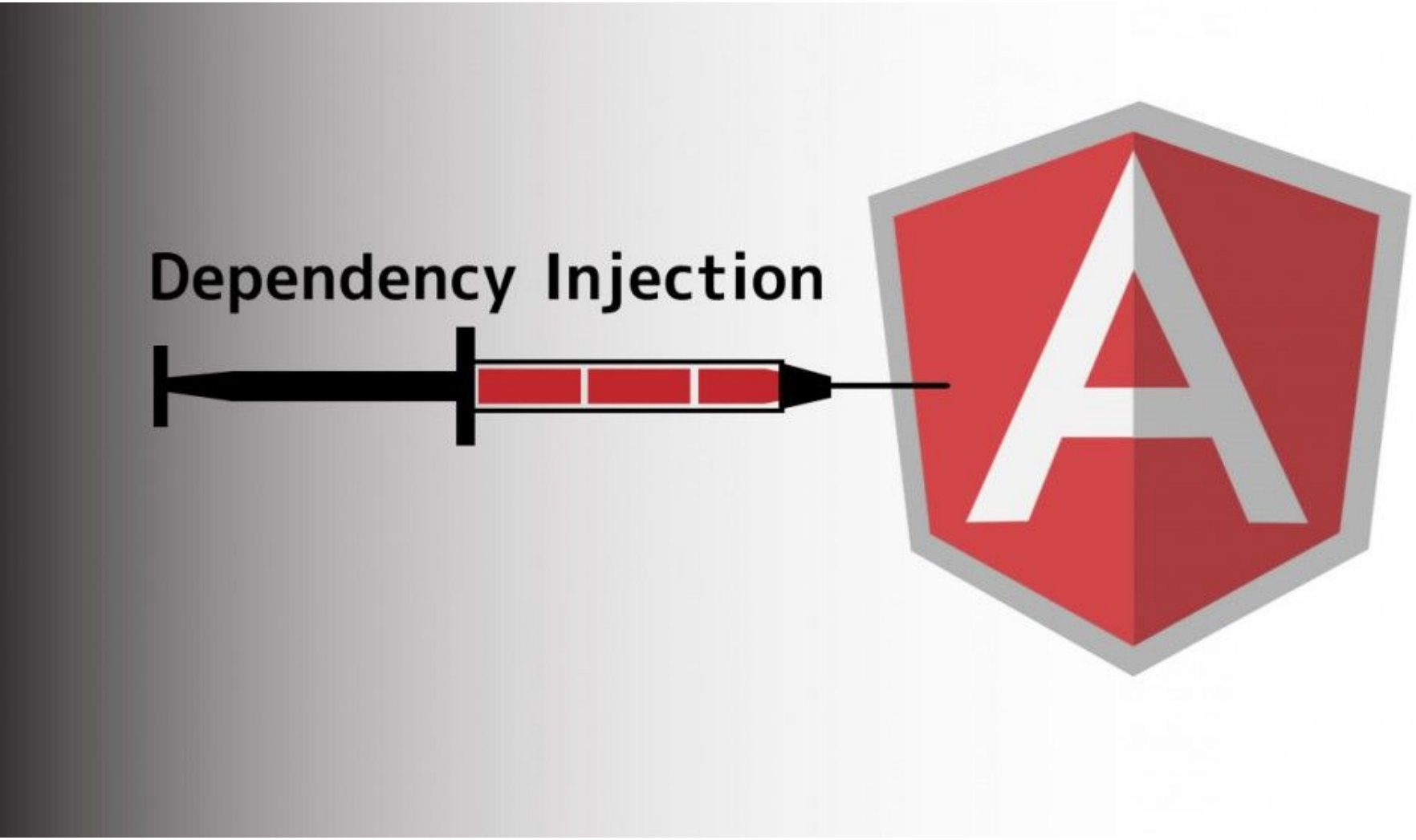

User
sees
initiates
actions
View
updates
via
binding
fires
events
Class
Component
Data exchange
via
getters/setters
Model
Services
The server
communicates
with the client
via HTTP or
WebSocket
A top-level component

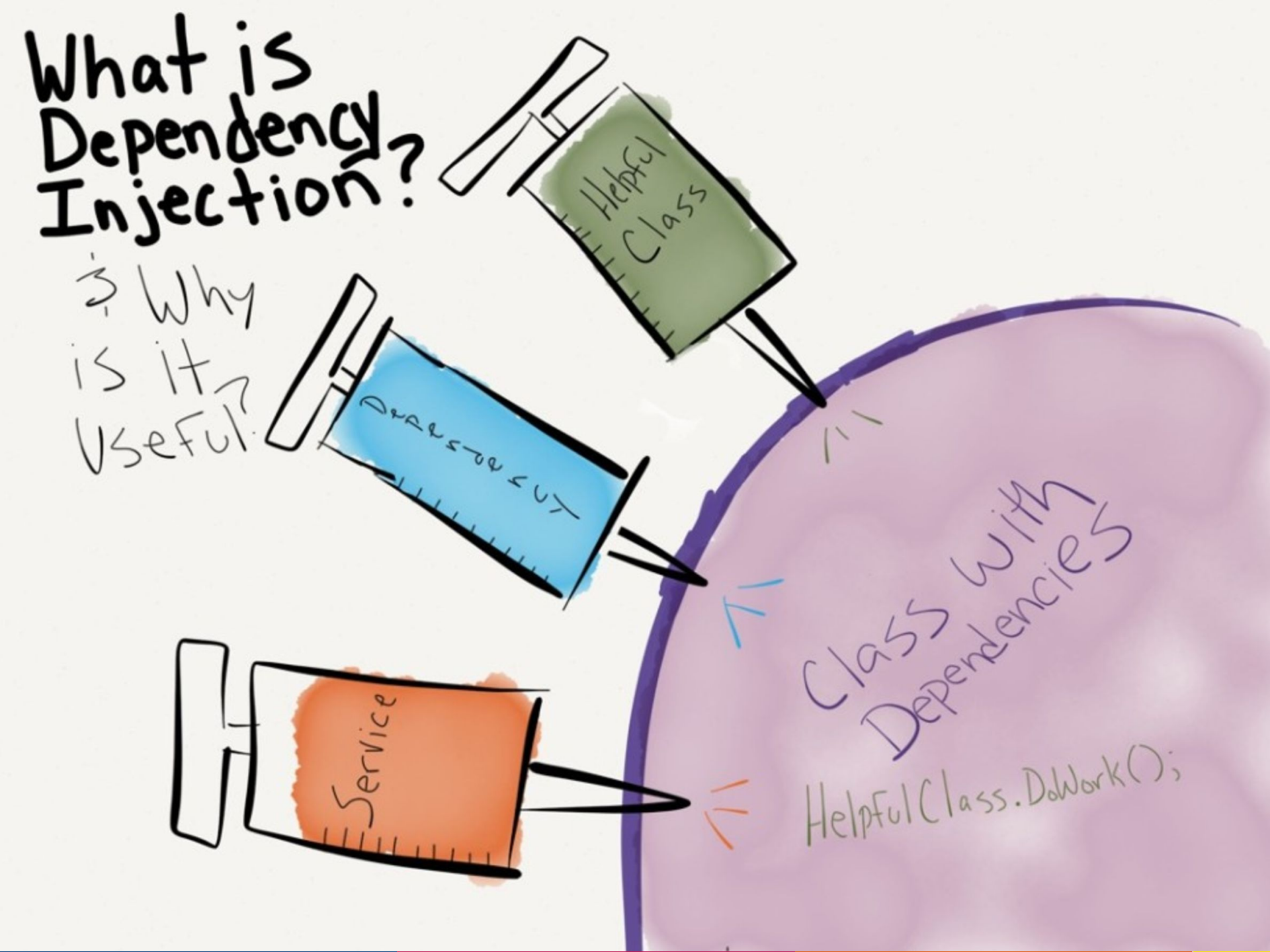
Two-way data binding



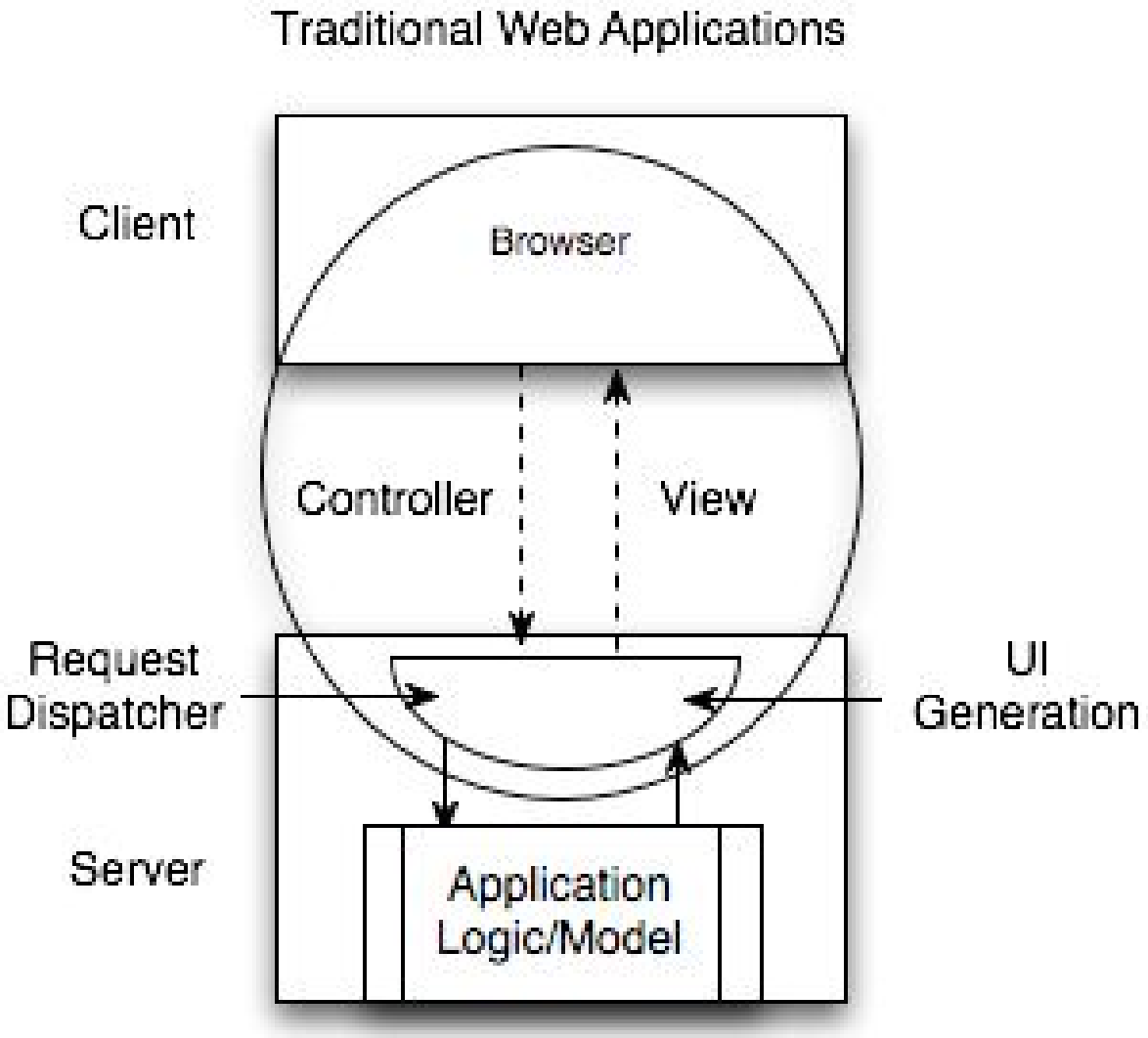
TWO WAY DATA BINDING EXAMPLE




Dependency Injection


What is Dependency Injection?
Why is it useful?
Helpful Class
Class with Dependencies
HelpfulClass.DoWork();

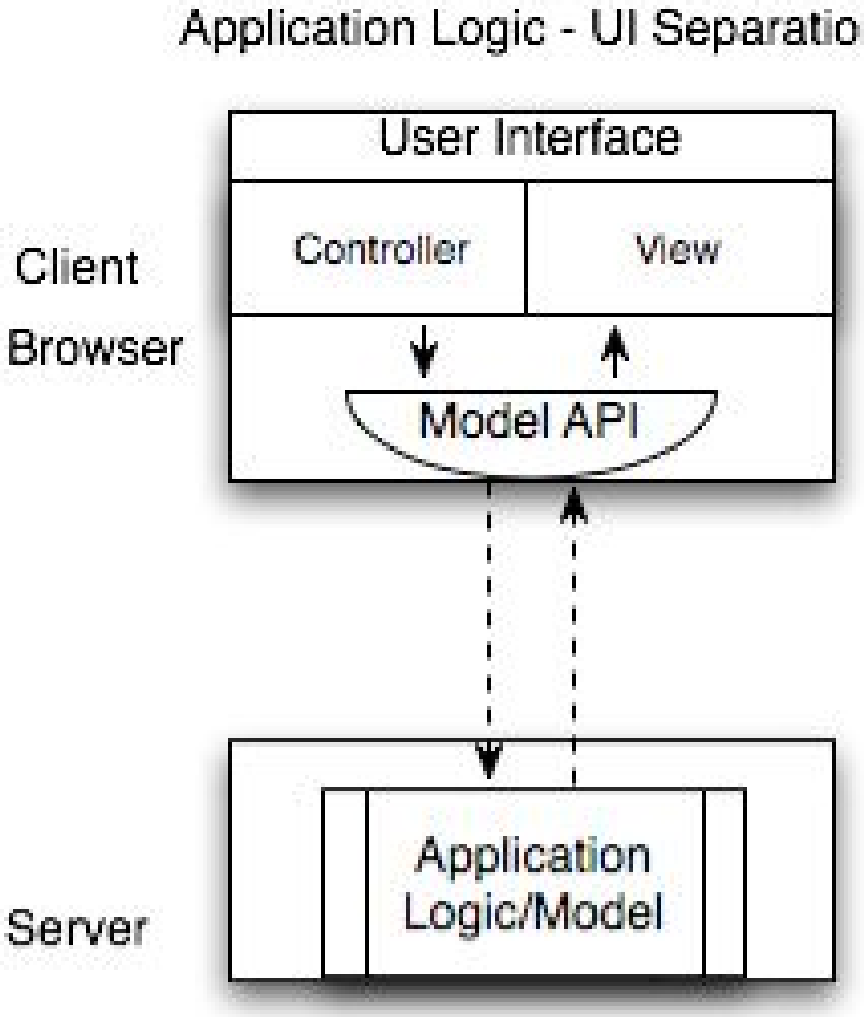
Why use a JS MVC framework



- **Poor distribution of processing** – With a large number of clients, doing all the processing on the server is inefficient.
- High user response latency
- Difficult programming model
- Increased vector of attack
- Heavy state management on the servers
- Offline Difficulties
- Reduced opportunity for interoperability

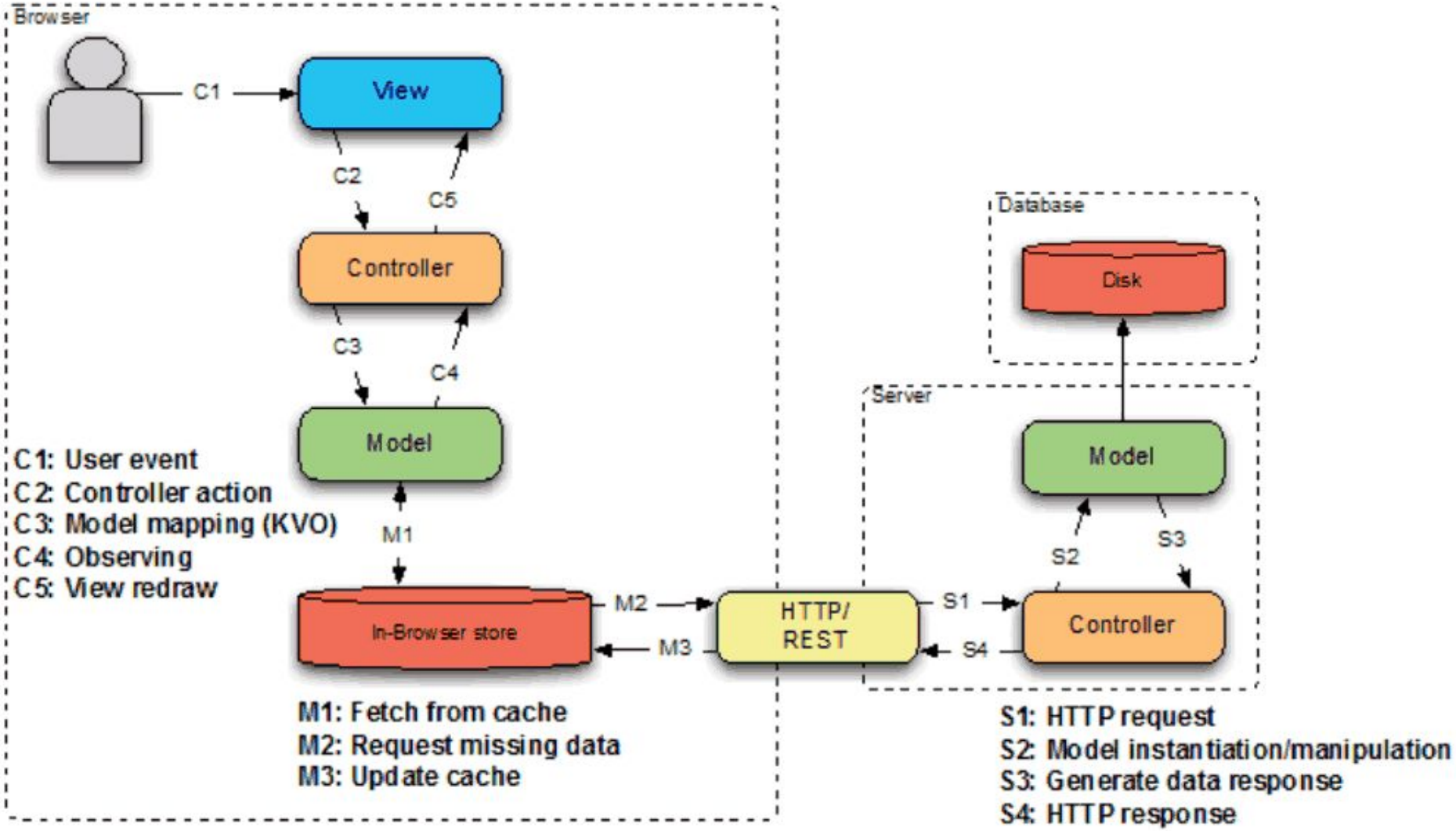
Why use a JS MVC framework

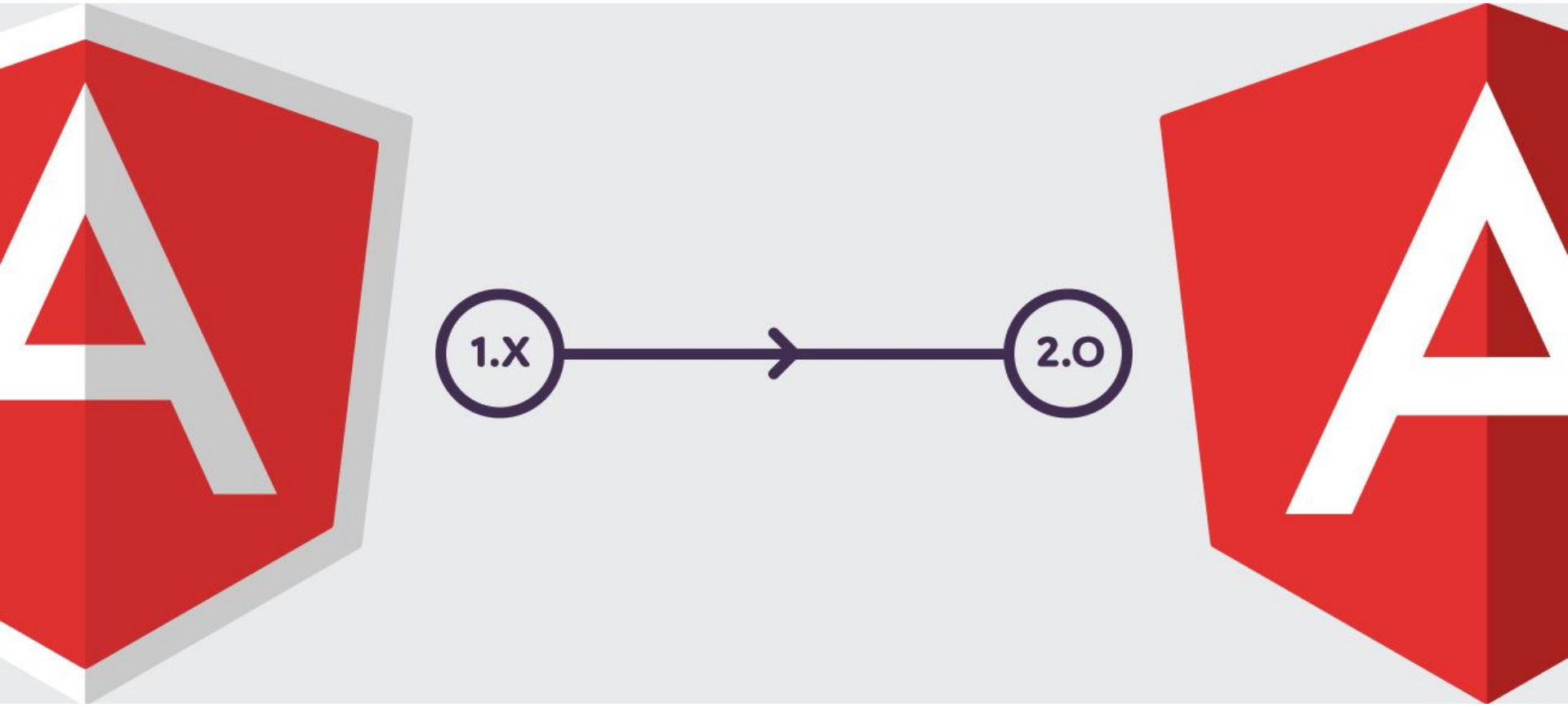
JSMVC Web Applications



- **Scalability** - The more clients that use an application, the more client machines that are available, whereas the server processing capabilities remain constant
- Immediate **user response**
- Organized **programming model**
- Client side **state management**
- **Offline** applications
- Interoperability

Data Flow for modern web development




1.X
2.0

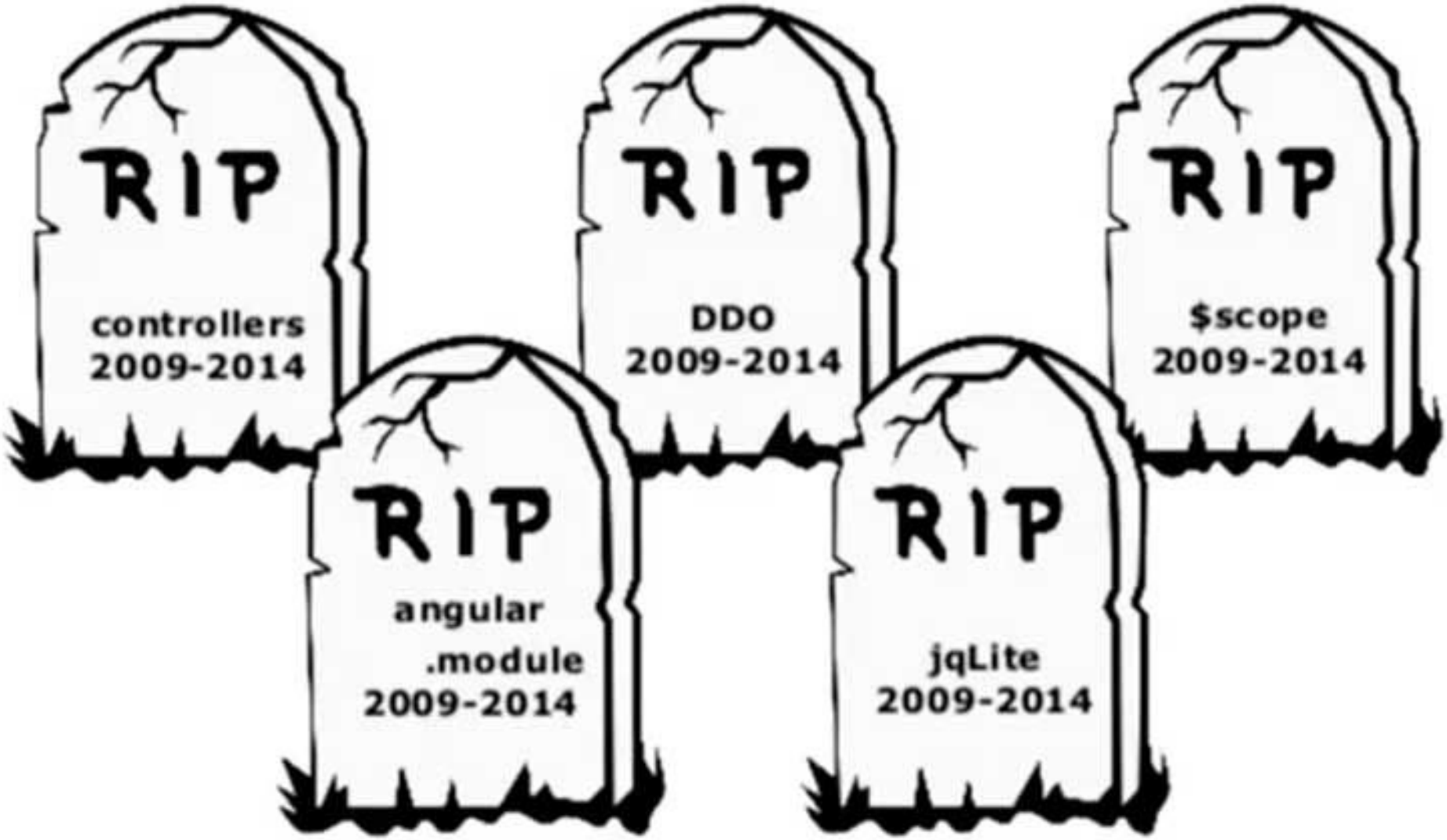

2.0

ANGULAR 2 **VS 1.X**

- Goodbye $scope
- No more controllers
- Component Based-UI
- 1-way data flow
- ES6 / Typescript
- New built-in directives

ANGULAR 2 **VS 1.X**

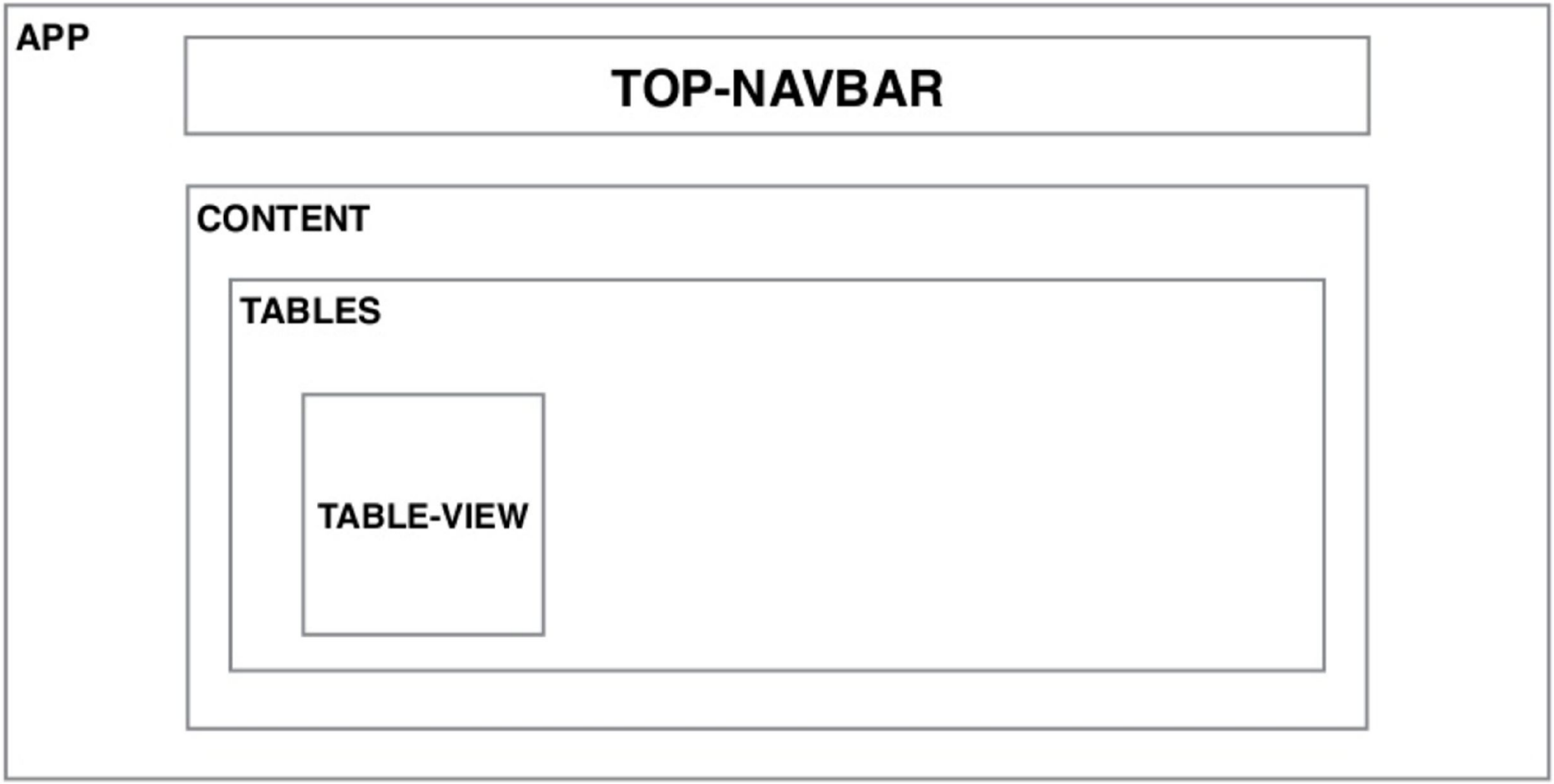
- New DI system
- Performance
- Better Mobile Support
- Server side render e Native Script
- Embrace Flux and RxJS
- Change Detection System


RIP
controllers
2009-2014
RIP
DDO
2009-2014
RIP
$scope
2009-2014
RIP
angular
.module
2009-2014
RIP
jqLite
2009-2014

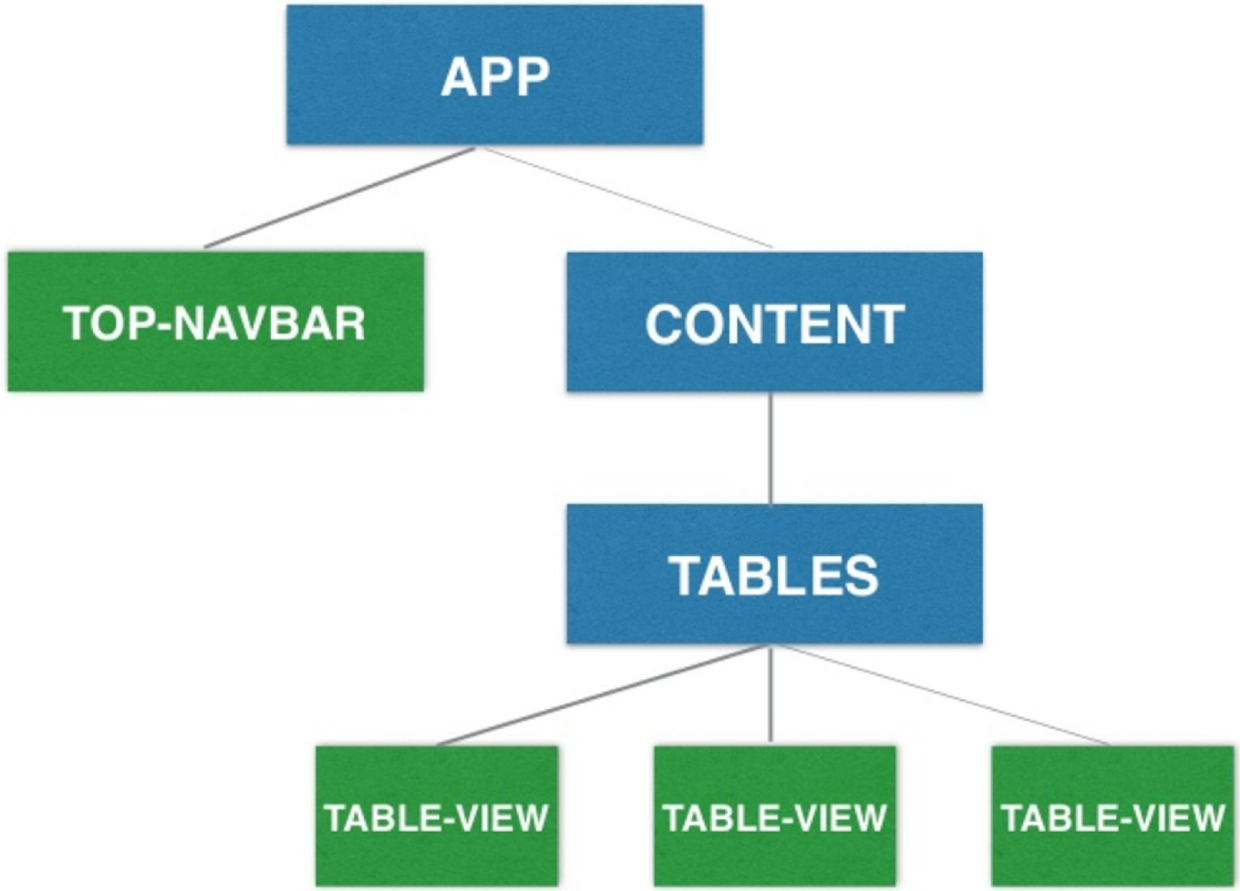
AN ANGULAR2 APPLICATION UI IS A COMPOSITION OF COMPONENTS

(reusable UI building blocks)

Thinking in components



Thinking in components



wrapper component

MOST BASIC COMPONENT

In JavaScript

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

@Component({
  selector: 'App',
  template: '<h1>Hello Component!</h1>'
})

class App {}
```

Use in HTML

```
<body>

  <App></App>

</body>
```

COMPONENT COMPOSITION

content.ts

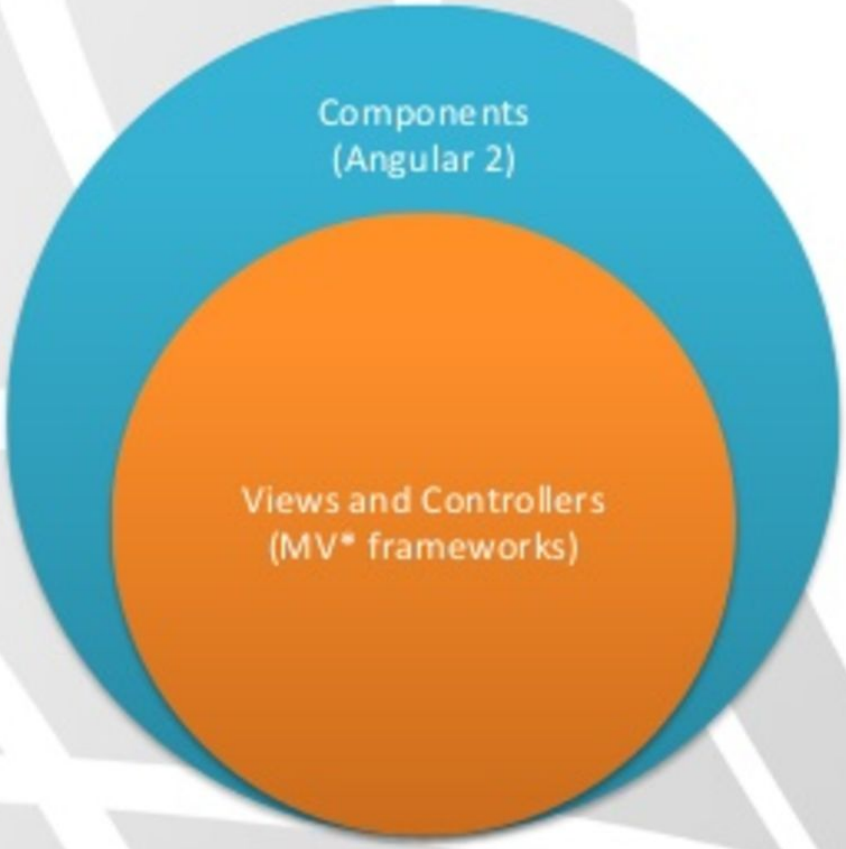
```
import {Component} from "angular2/core";

@Component({
  selector:'content',
  template:`<div class="container"></div>`
})

export class Content {}
```

app.ts

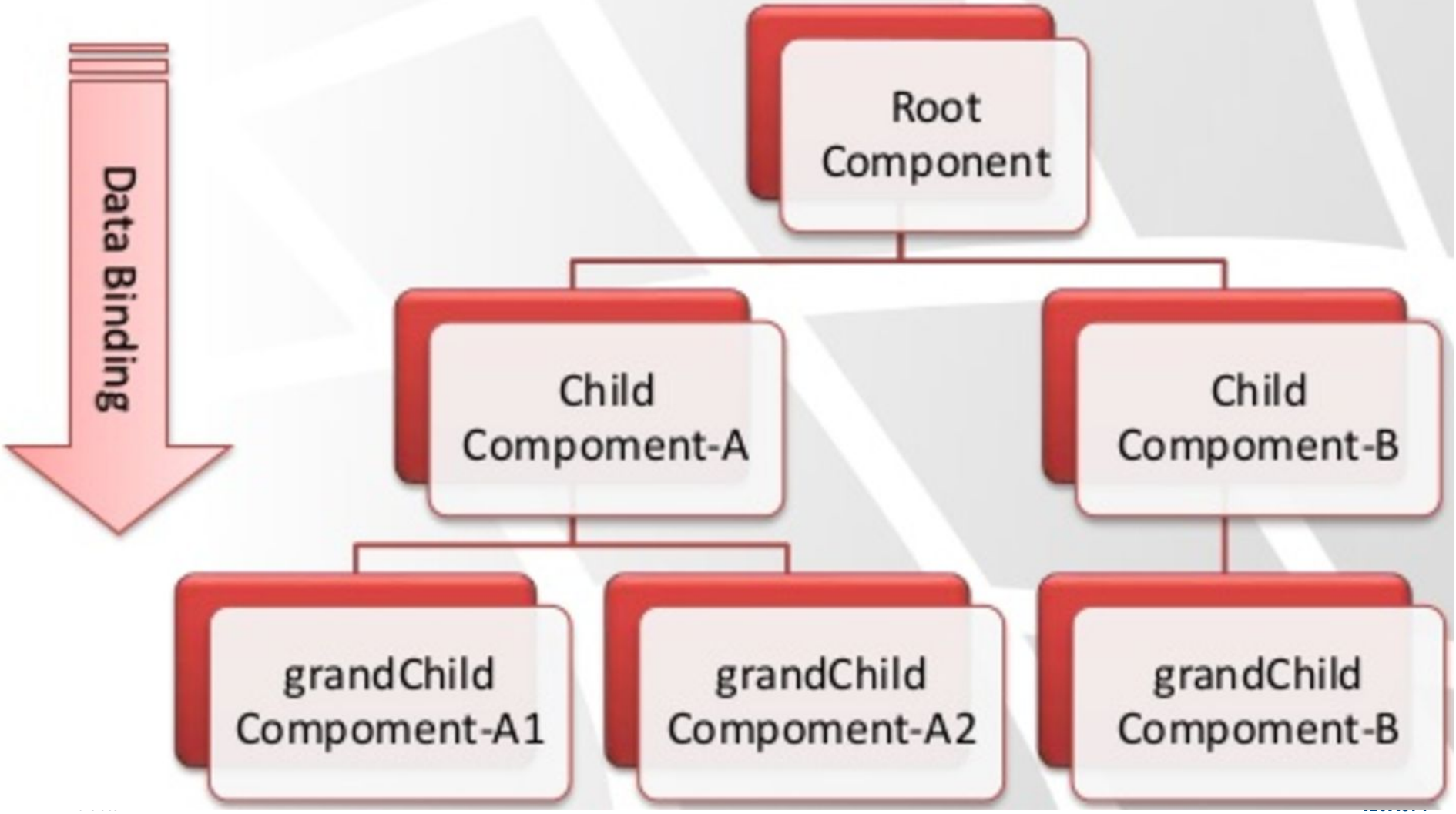
```
import {Component} from 'angular2/core';
import {TopNavBar} from './top-navbar';
import {Content}   from './content';

@Component({
  selector: 'app',
  directives: [Content, TopNavBar],
  template:
    `<top-navbar></top-navbar>
     <content></content>
    `
})

export class App {}
```

From MVC to Components

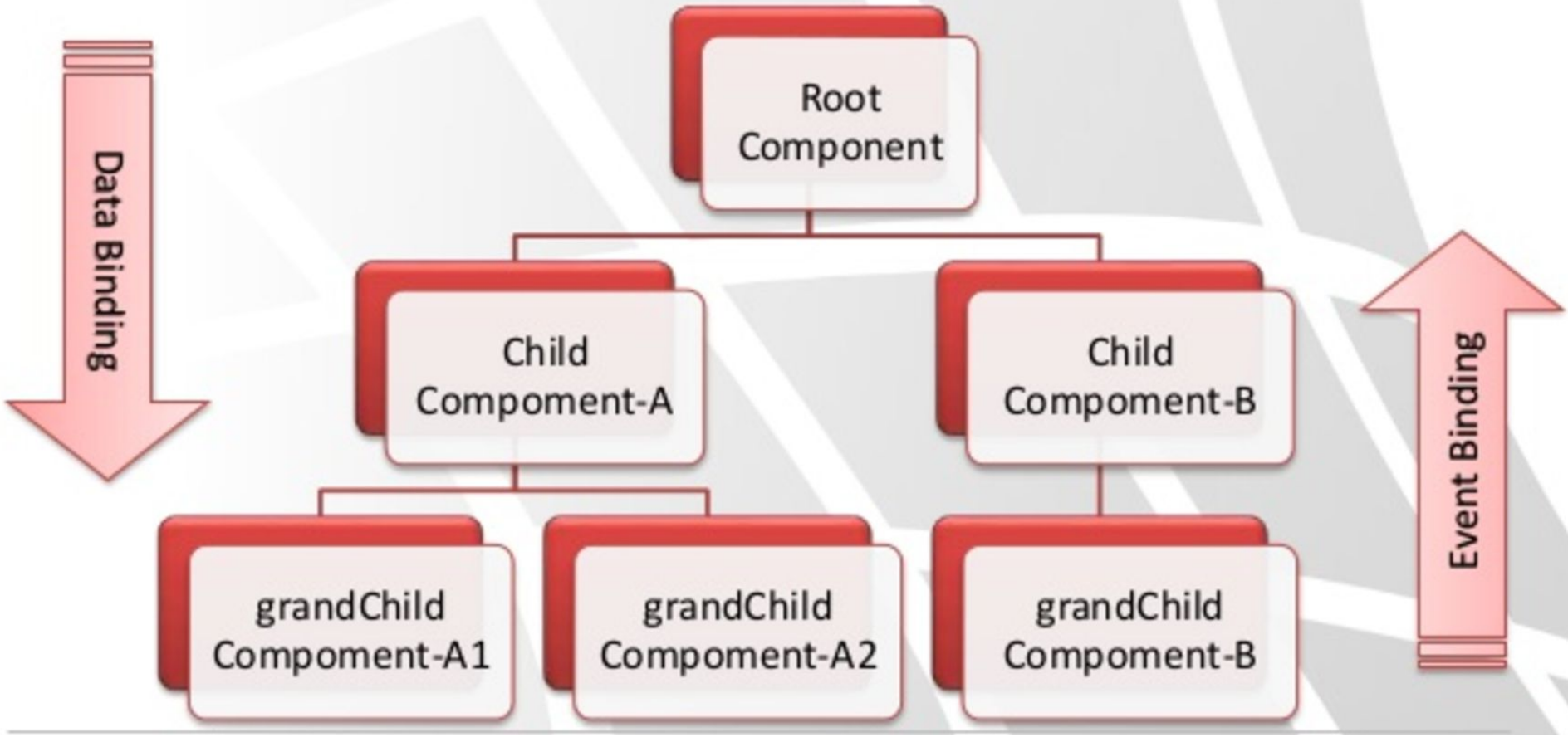
- Components are a higher-level abstraction than MVC
- MV* Frameworks focuses on controllers
- Angular 2 focuses on components
  - Component classes implement logic
  - Controllers and views are metadata



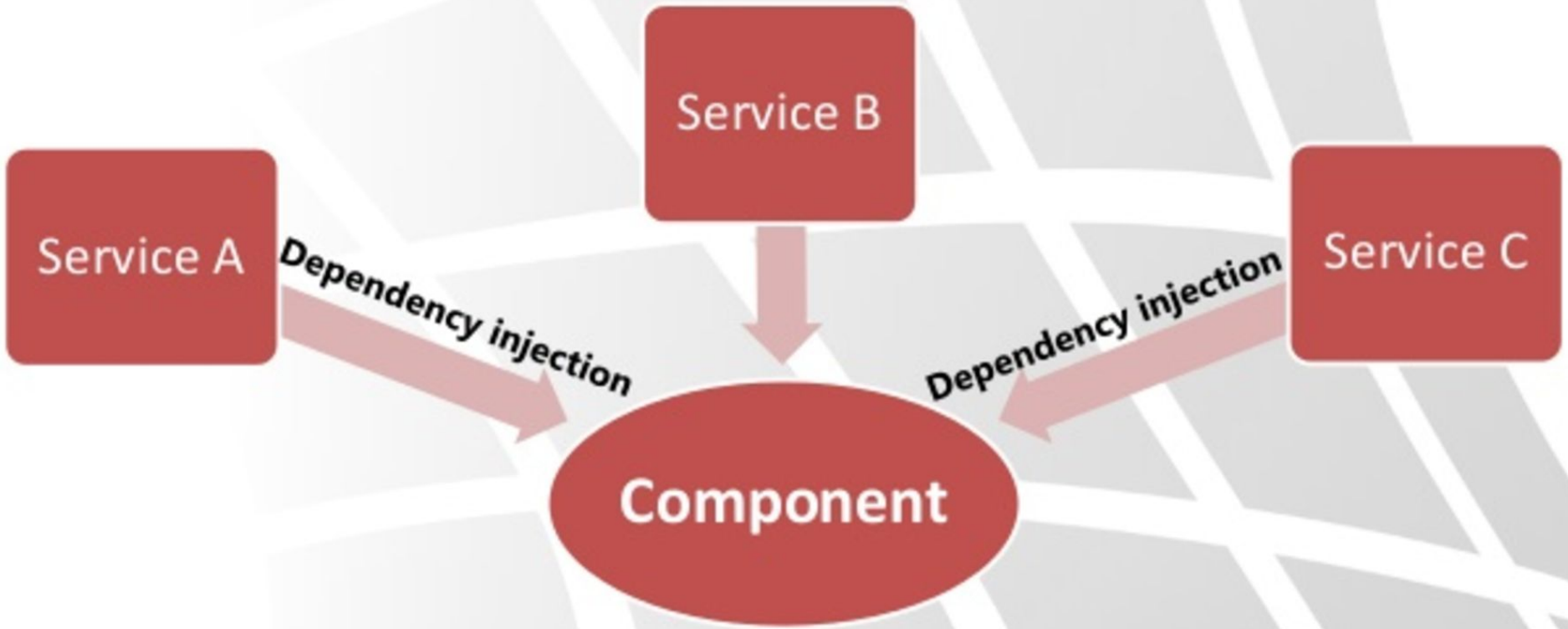
Data is Flowing downwards.



Events are Flowing upwards.

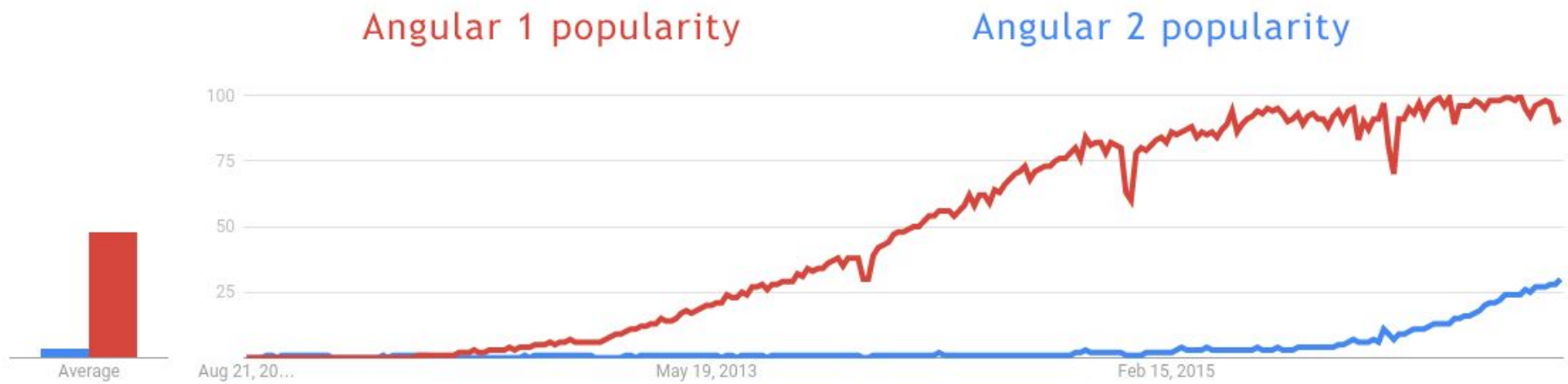


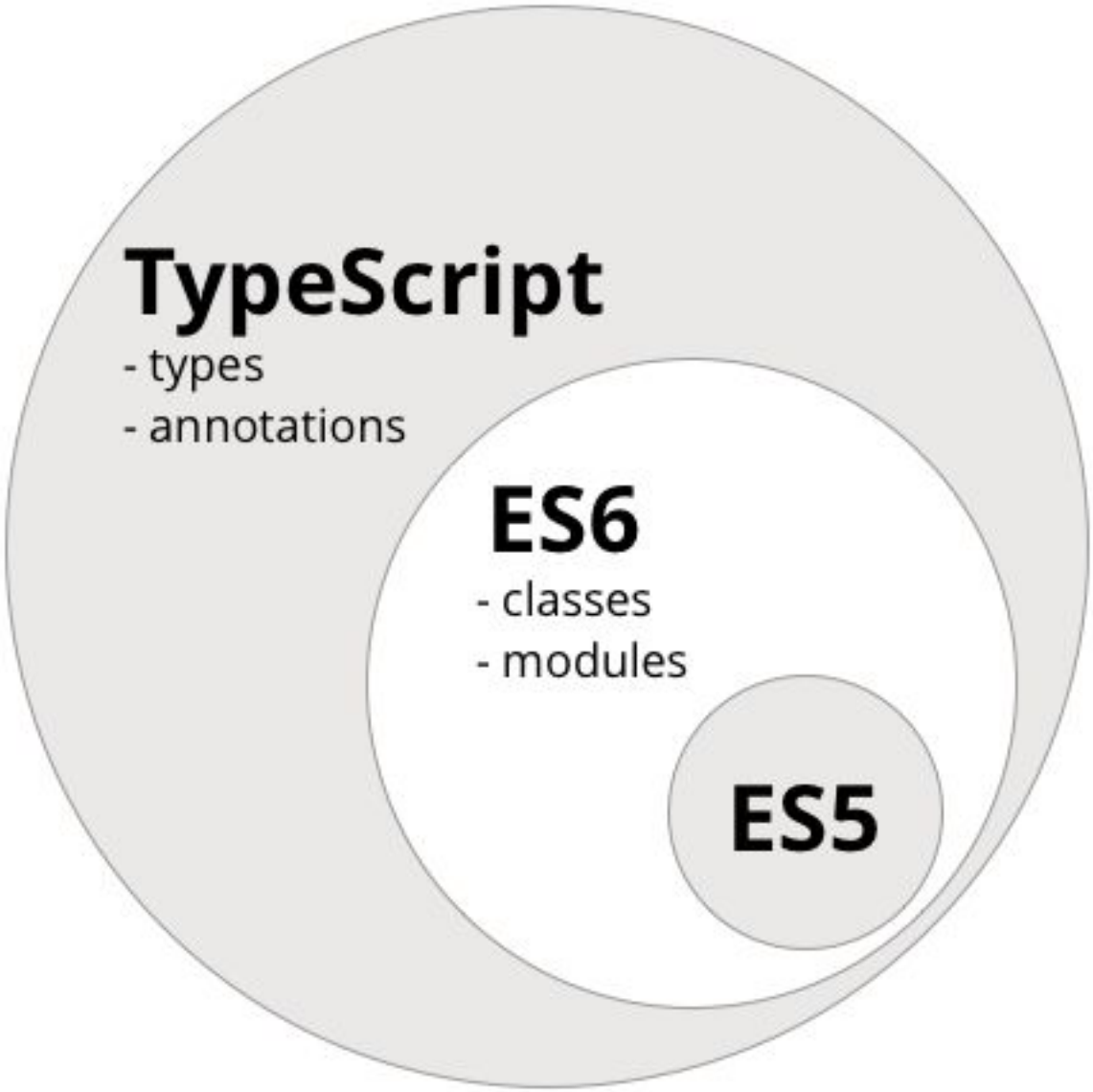
- Each Component Can consume injectable Services.



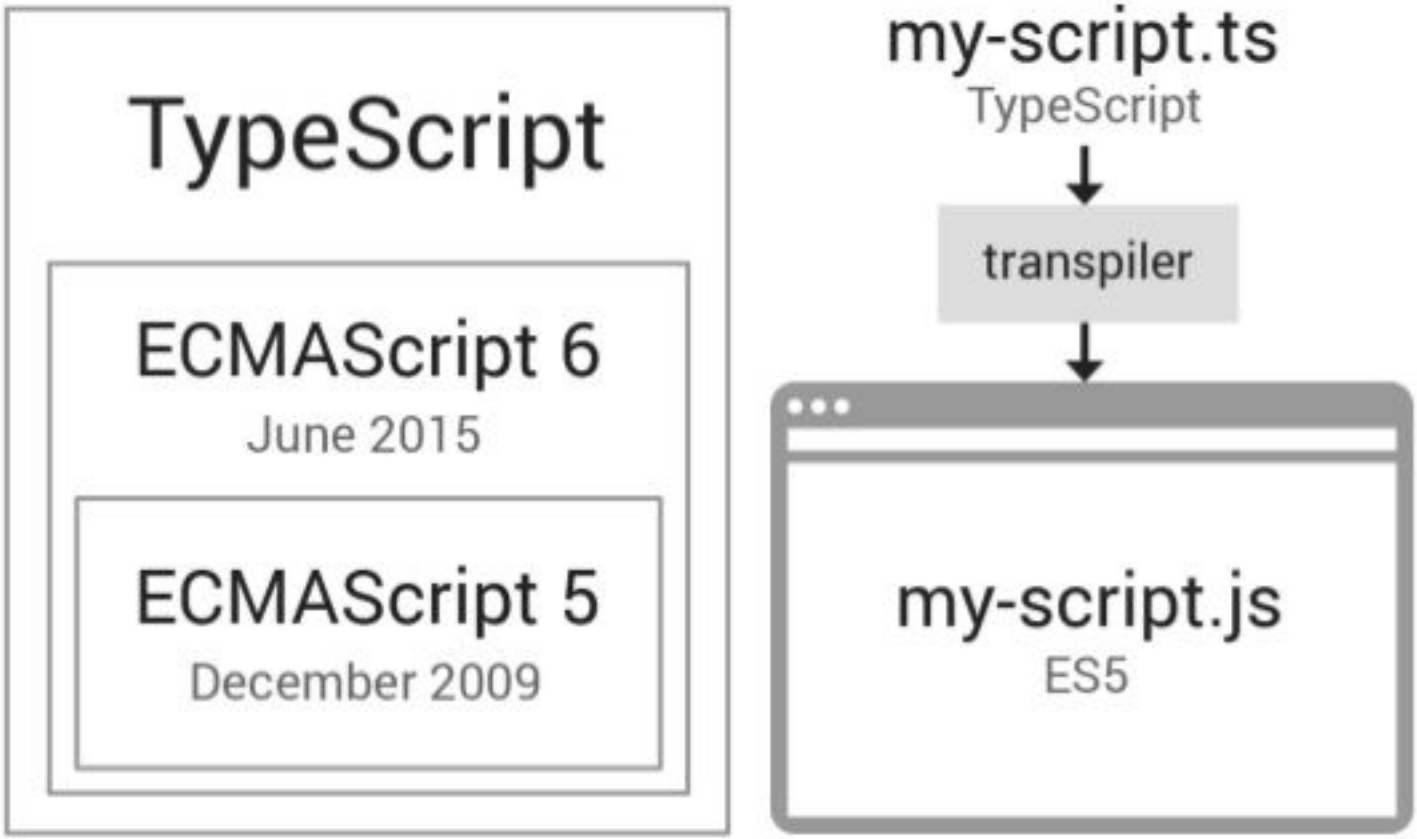
- Components, Services, Directives and Pipes are all defined inside Angular Modules

Angular 1 vs. 2 popularity




TypeScript
- types
- annotations
ES6
- classes
- modules
ES5

TypeScript


TypeScript
ECMAScript 6
June 2015
ECMAScript 5
December 2009
my-script.ts
TypeScript
transpiler
my-script.js
ES5

TSC - the TypeScript compiler

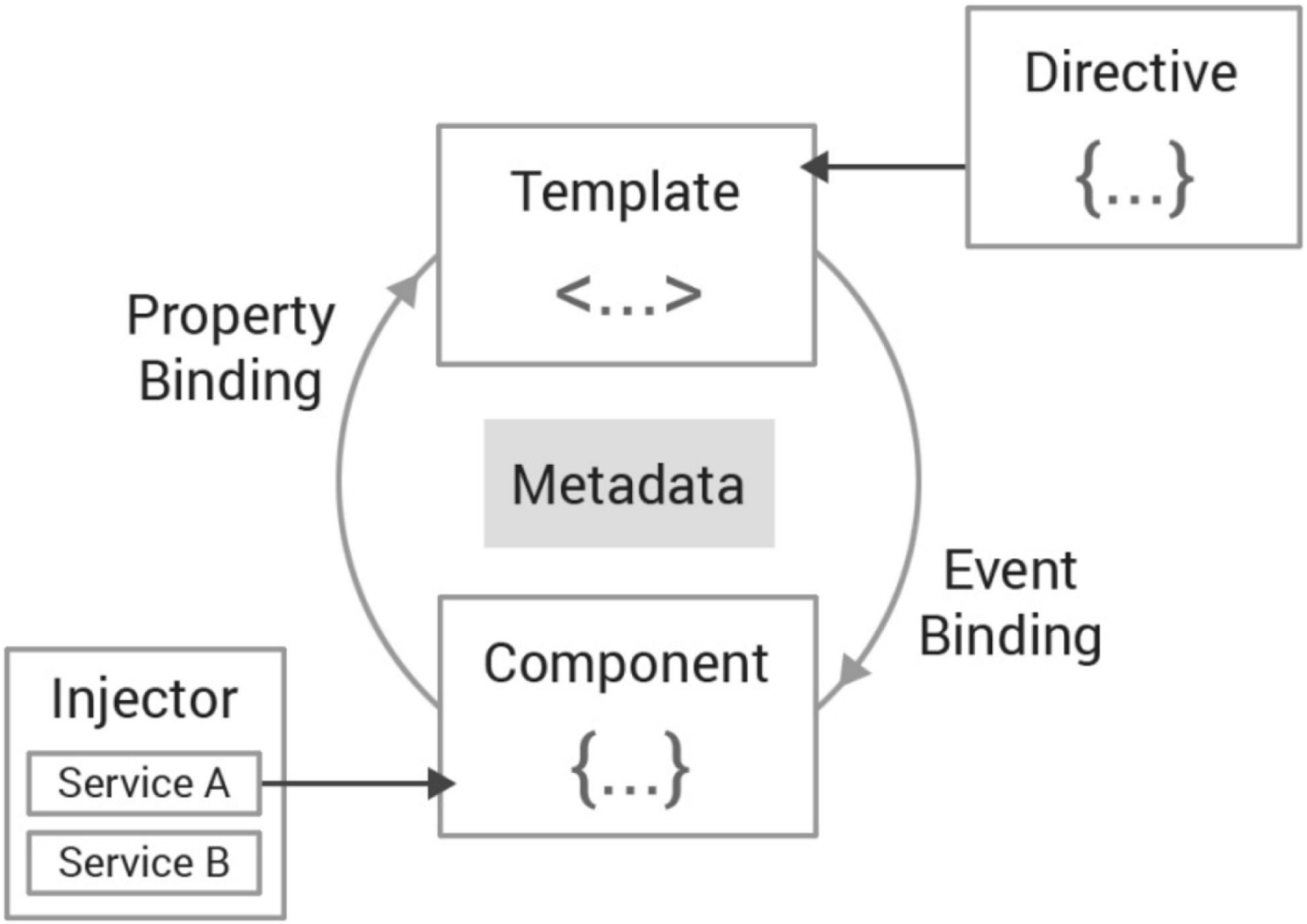
TSC is a source-to-source compiler (a transpiler).

There are lots of options that allow you to:

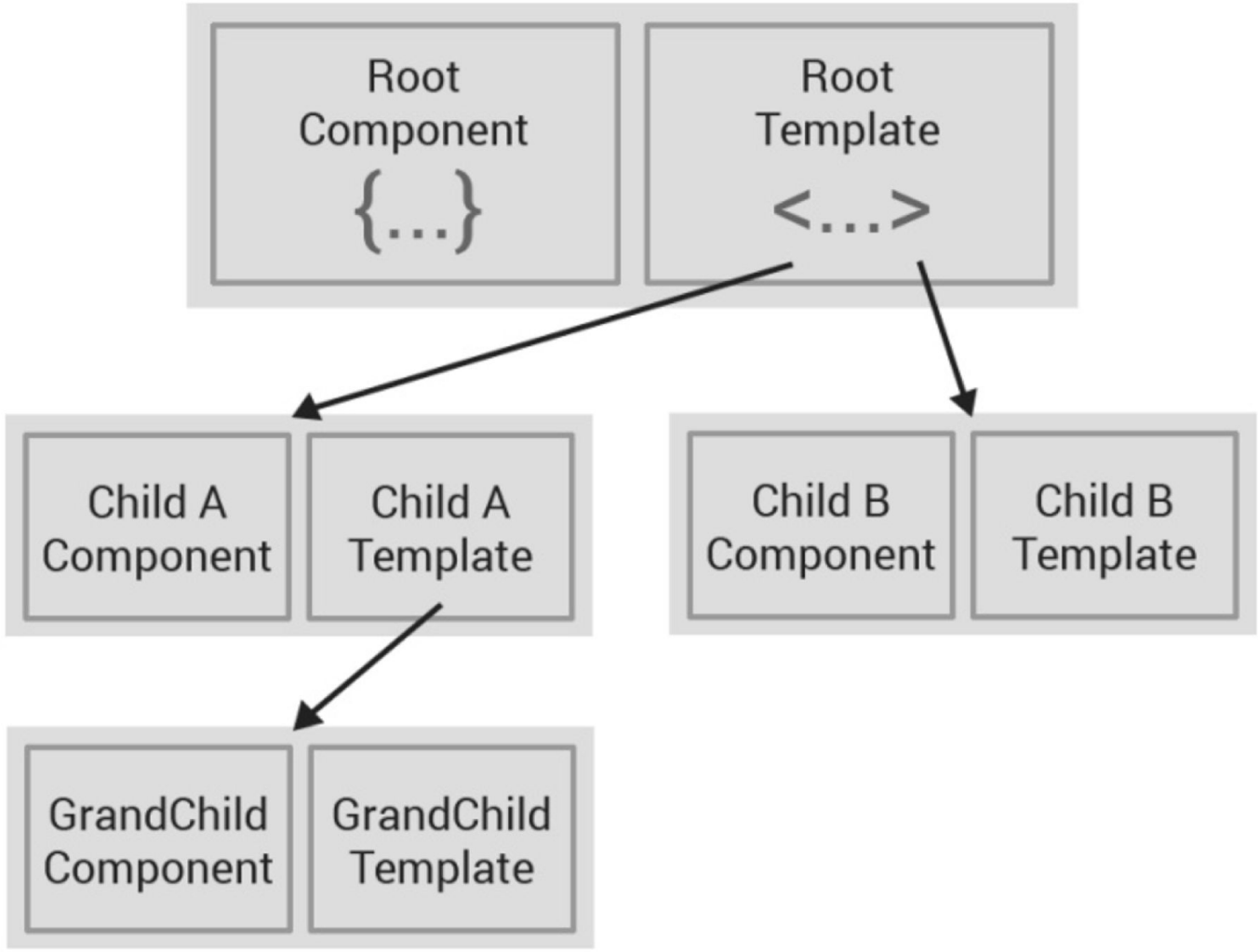
- concatenate different files in a single output file.
- generate sourcemaps.
- generate module loading code (node.js or require.js).

You can play with the TypeScript playground or setup your environment to see it in action.

Architecture



Architecture



Components

```
1 // my-app.component.ts
2 import {bootstrap} from 'angular2/platform/browser';
3 import {Component} from 'angular2/core';
4
5 @Component({
6     selector: 'my-app',
7     template: '<h1>My application</h1>'
8 })
9 export class MyAppComponent { }
10
11 bootstrap(MyAppComponent);
```

```
1 <!-- index.html -->
2 <body>
3     <my-app>Loading...</my-app>
4 </body>
```

Displaying data

```
1 import {Component} from 'angular2/core';
2
3 @Component({
4     selector: 'my-app',
5     template: '<h1>{{message}}</h1>'
6 })
7 export class MyAppComponent {
8     message: string;
9
10    constructor() {
11        this.message = 'My application';
12    }
13 }
```

Displaying data

```
1 import {Component} from 'angular2/core';
2
3 @Component({
4     selector: 'my-app',
5     template: `
6         <input value="{{message}}">
7         <input [value]="message">
8     `
9 })
10 export class MyAppComponent {
11     message: string = 'My application';
12 }
```

User Input

```
1 import {Component} from 'angular2/core';
2
3 @Component({
4     selector: 'my-app',
5     template: `
6         <p>{{message}}</p>
7         <button (click)="onClick()">
8         <input #box (keyup)="onKey(box.value)">
9     `
10 })
11 export class MyAppComponent {
12     message: string = 'My application';
13     onClick() { }
14     onKey(value: string) { }
15 }
```

Two-way data binding

```
1 import {Component} from 'angular2/core';
2
3 @Component({
4     selector: 'my-app',
5     template: `
6         <input
7             [ngModel]="message"
8             (ngModelChange)="message=$event">
9         <input [(ngModel)]="message">
10    `
11 })
12 export class MyAppComponent {
13     message: string = 'My application';
14 }
```

Parent and child components

```
1 // parent.component.ts
2 import {Component} from 'angular2/core';
3 import {ChildComponent} from './child.component';
4
5 @Component({
6     selector: 'parent',
7     template: `
8         <child [childMsg]="parentMsg"></child>
9     `,
10    directives: [ChildComponent]
11 })
12 export class ParentComponent {
13    parentMsg: string = 'My application';
14 }
```

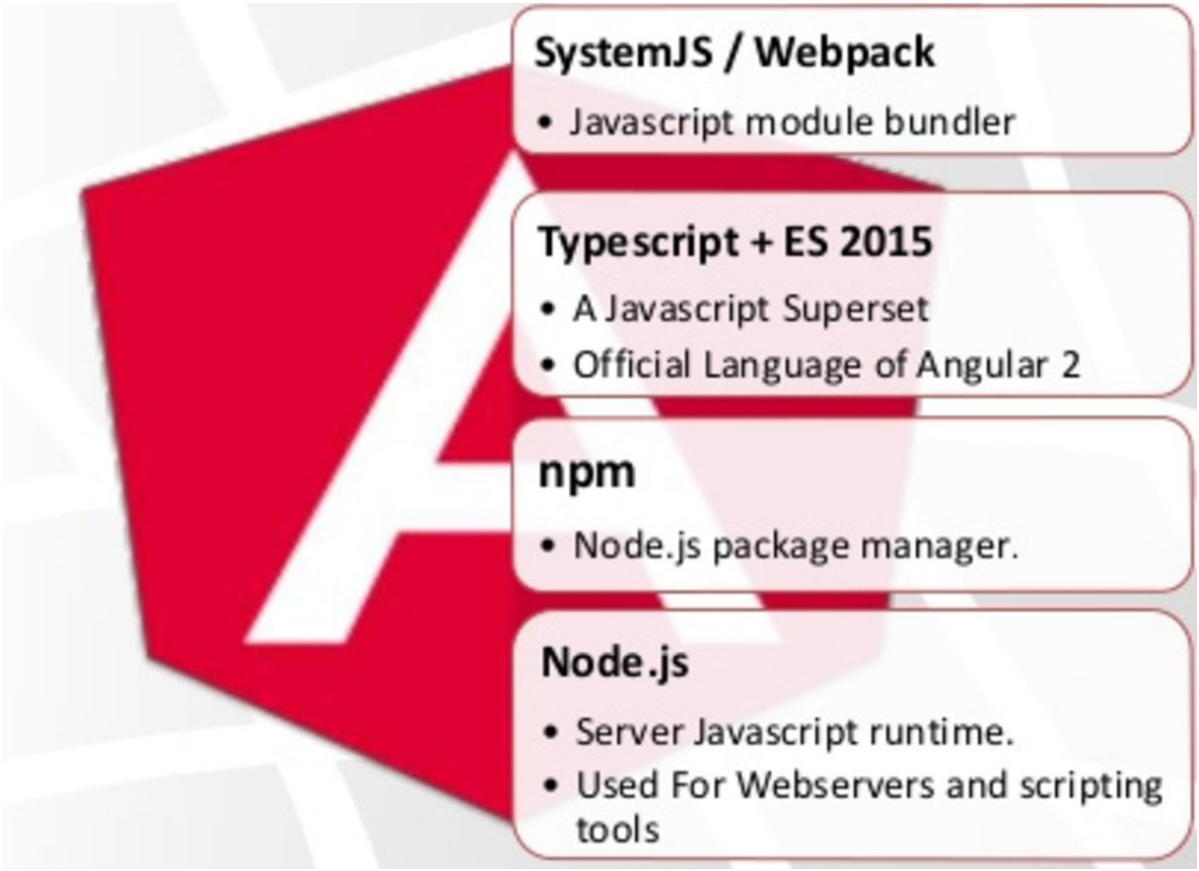
Dependency Injection

```
1 // my-app.service.ts
2 import {Injectable} from 'angular2/core';
3
4 @Injectable()
5 export class MyAppService {
6     doSomething() { }
7 }
```

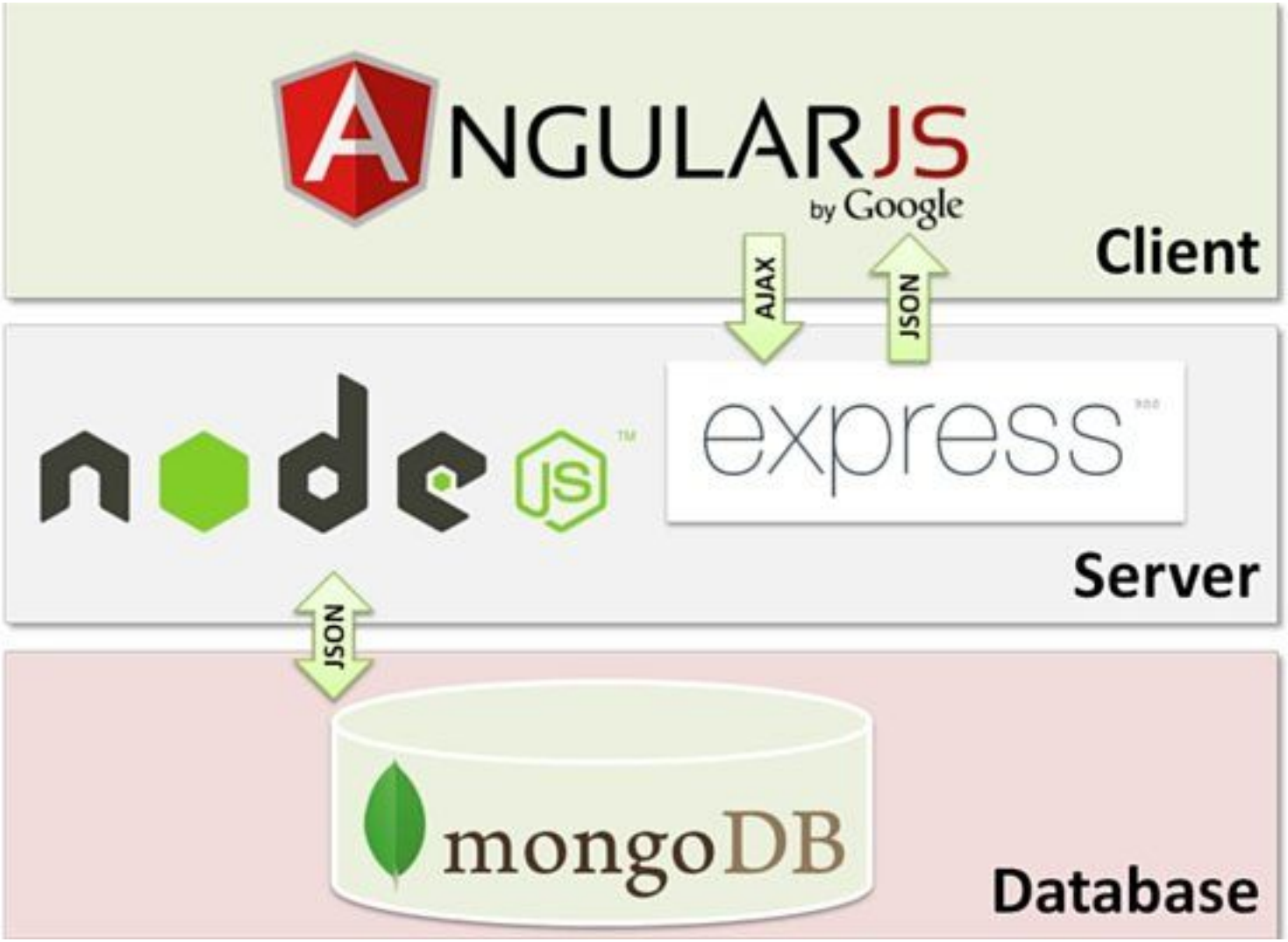
Dependency Injection

```
1 // my-app.component.ts
2 import {Component} from 'angular2/core';
3 import {MyAppService} from './my-app.service';
4
5 @Component({
6     selector: 'my-app',
7     template: ''
8 })
9 export class MyAppComponent {
10     constructor(private _myAppService: MyAppService) { }
11 }
```

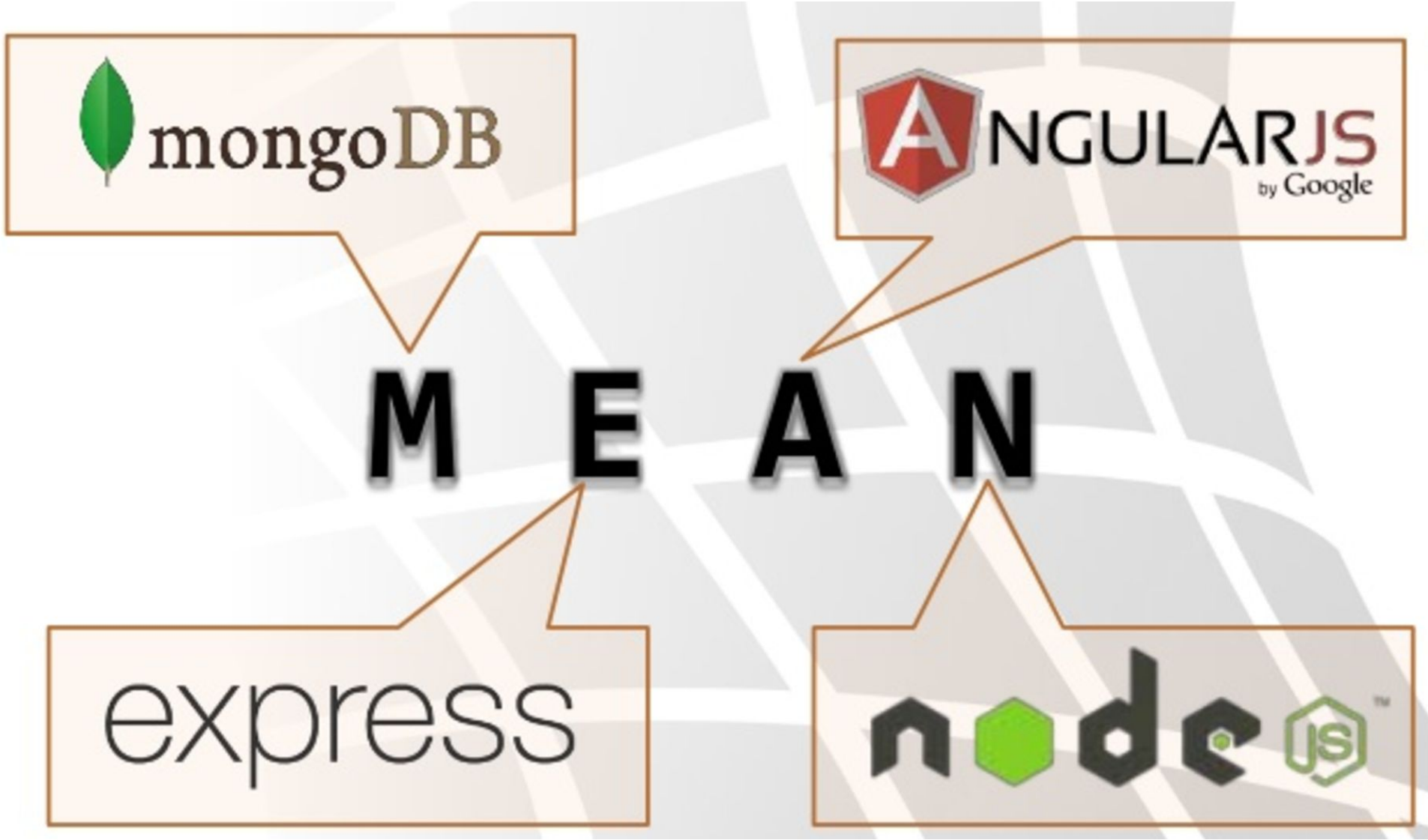
ANGULAR 2 DEVELOPMENT ENVIRONMENT



MEAN STACK


ANGULARJS
by Google
Client
AJAX
JSON
node
JS
express
Server
JSON
mongoDB
Database

MEAN STACK



THANK YOU
AND HAVE A GOOD TRAINING!