



# Modern Web Architectures

# The Early Web

- ◆ Focus on finding and sharing static content
- ◆ Mainstream application architecture was client/server – not much thought was given to moving applications to the browser
- ◆ Early attempts at dynamic content were provided for limited use cases
- ◆ Some browsers supported “plugging in” applications into the browser

# Web 1.0 Architecture



# Web 1.0 Architecture

## ◆ Advantages

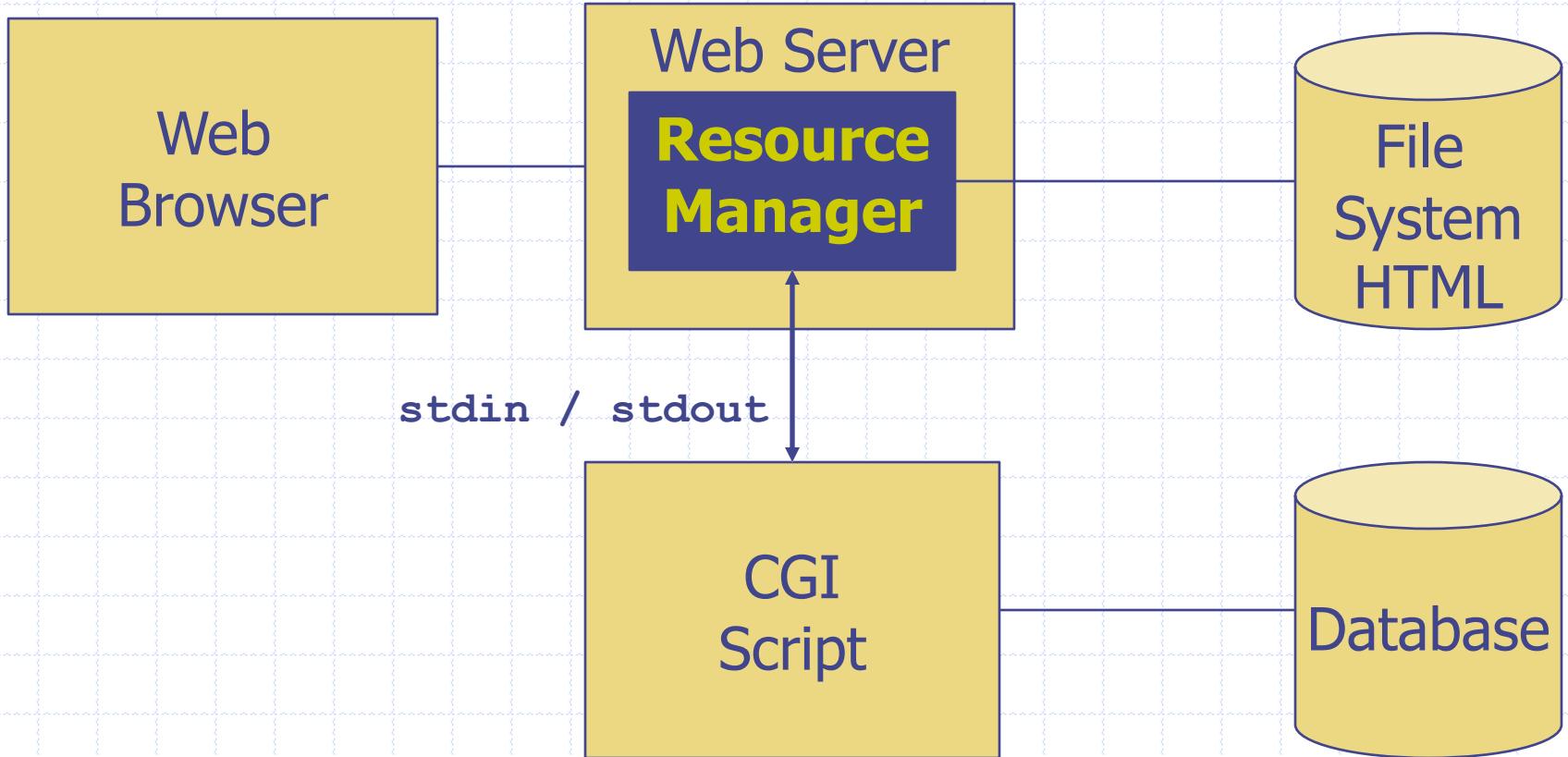
- Simple
- Cachable
- Indexable

## ◆ Disadvantages

- No dynamic content
- No interaction with user – every request must go back to web server

**Example : Netscape 1.0**

# Web 1.1 Architecture



# Web 1.1 Architecture

## ◆ Advantages

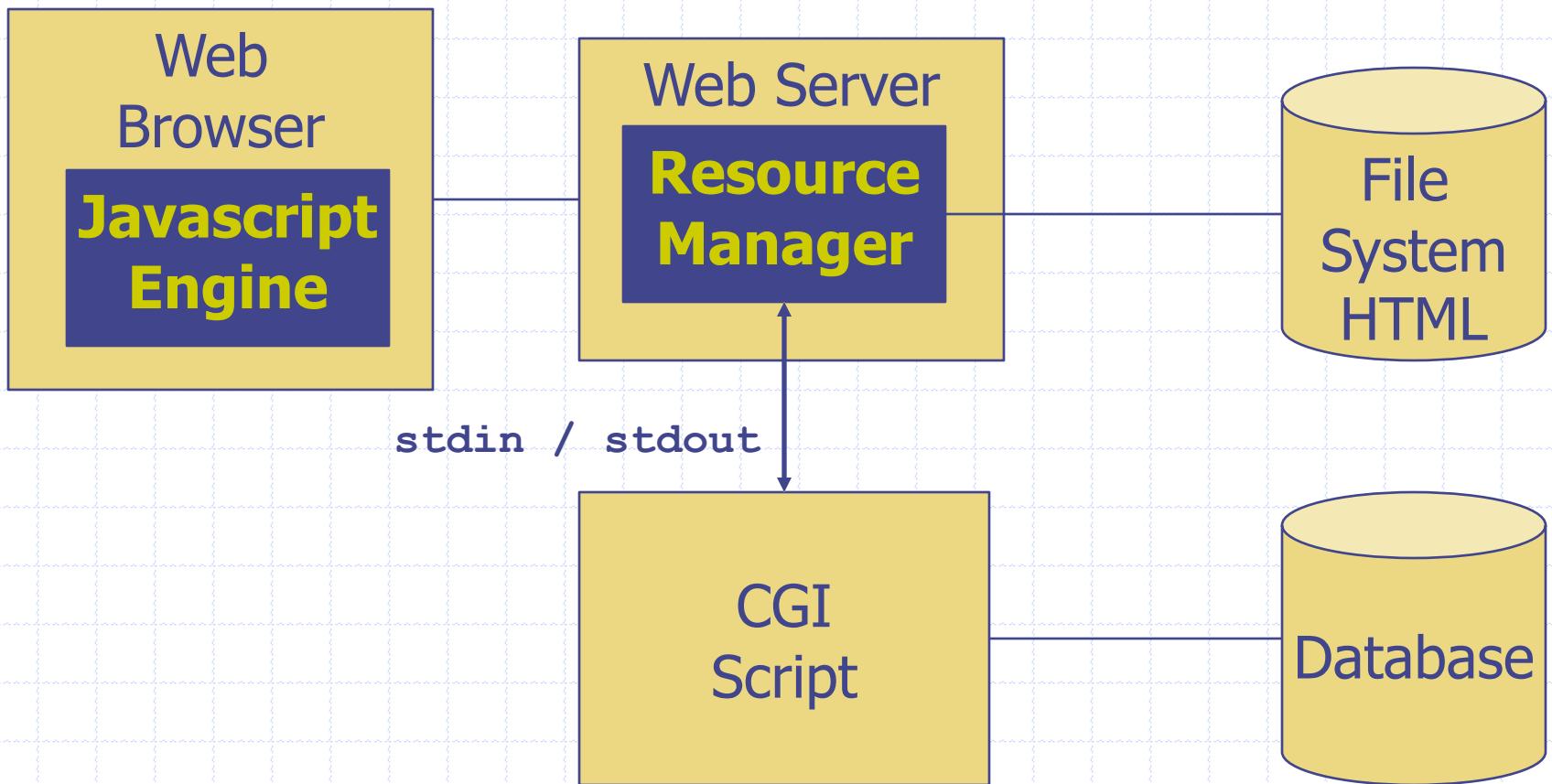
- Somewhat Cachable
- Some ability for Dynamic Content / Personalization

## ◆ Disadvantages

- High compute overhead – need to run a separate process for each request
- All I/O via anonymous pipes – must parse HTML in, and generate HTML out
- Although introduces dynamic content, all updates require a round-trip to the server

## Example : Perl

# Web 1.2 Architecture



# Web 1.2 Architecture

## ◆ Advantages

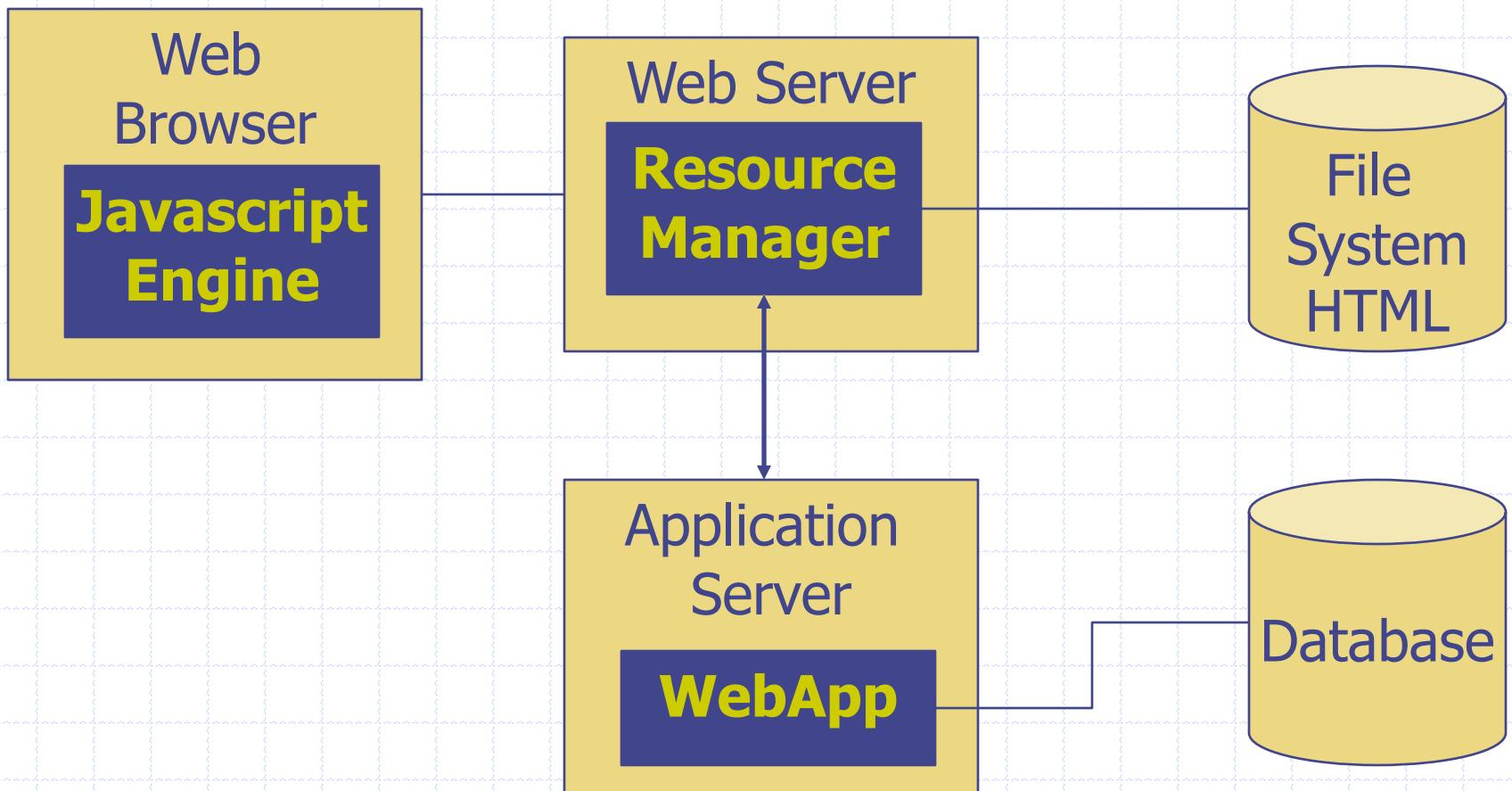
- Improved user experience – user input can be validated client side
- Some dynamic DOM manipulation to avoid round-trips to the server

## ◆ Disadvantages

- Same issues as architecture 1.1
- Code redundancy – same code needed on client and server side
- Javascript engines not very compatible

**Example : Netscape, IE3**

# Web 1.5 Architecture



# Web 1.5 Architecture

## ◆ Advantages

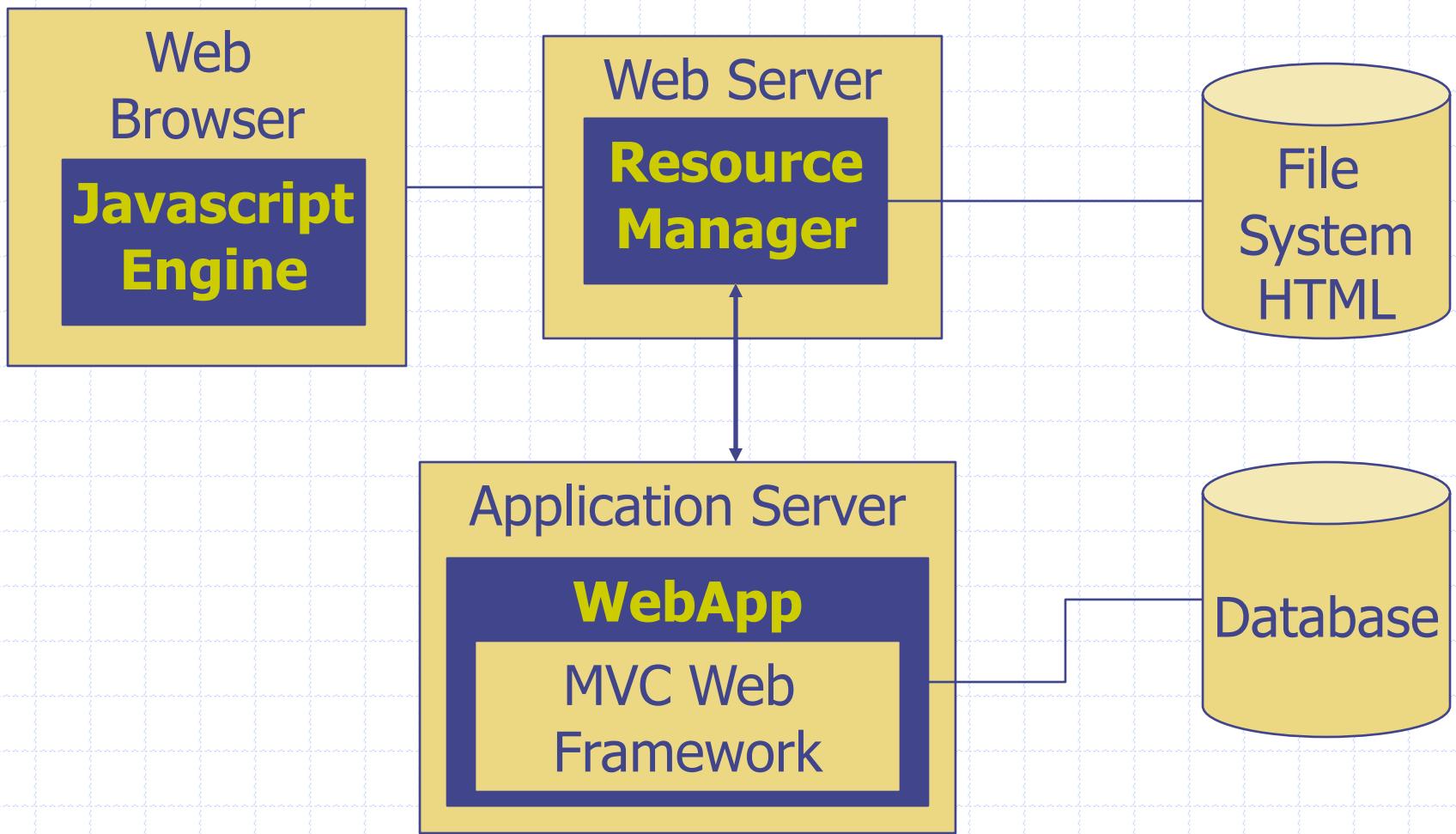
- Introduction of an application server component to provide lifecycle management for web applications
- Move to per-threads versus per-process for requests
- Wrappers for request and response objects, simplifying the need to parse and generate HTML

## ◆ Disadvantages

- Still need to work on a HTTP request and HTTP response level
- Common to generate

## Example : Tomcat 1.0

# Web 1.5+ Architecture



# Web 1.5+ Architecture

## ◆ Advantages

- Introduces the MVC framework to modularize server code
- Views can be pre-compiled to improve speed
- Views can be created in markup instead of code

## ◆ Disadvantages

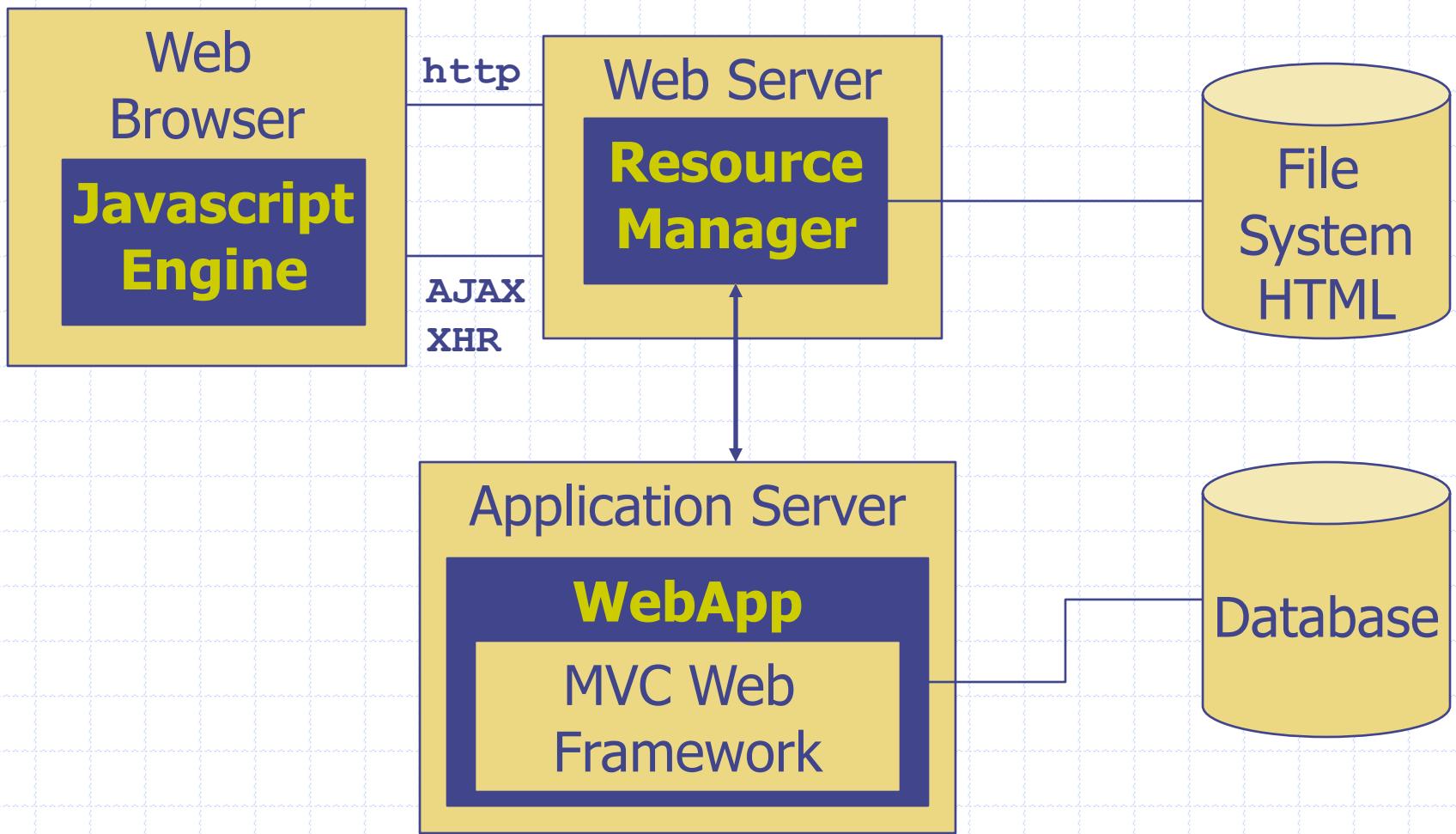
- Same as web 1.5
- Introduces additional complexity due to the discipline needed to work with, and configure a MVC framework
- Introduces non-standard markup to bind the view and controller

**Example : Spring MVC, Struts, ASP.Net**

# The Web 2.0 Architecture

- ◆ Demand for robust user experience delivered to the browser starts to emerge
- ◆ Javascript becomes a platform by accident – drives explosion of Javascript frameworks.
- ◆ XMLHttpRequest (XHR), which found its way into early browsers for limited use cases, is exploited by Google to create some amazing applications

# Web 2.0 Architecture



# Web 2.0 Architecture

## ◆ Advantages

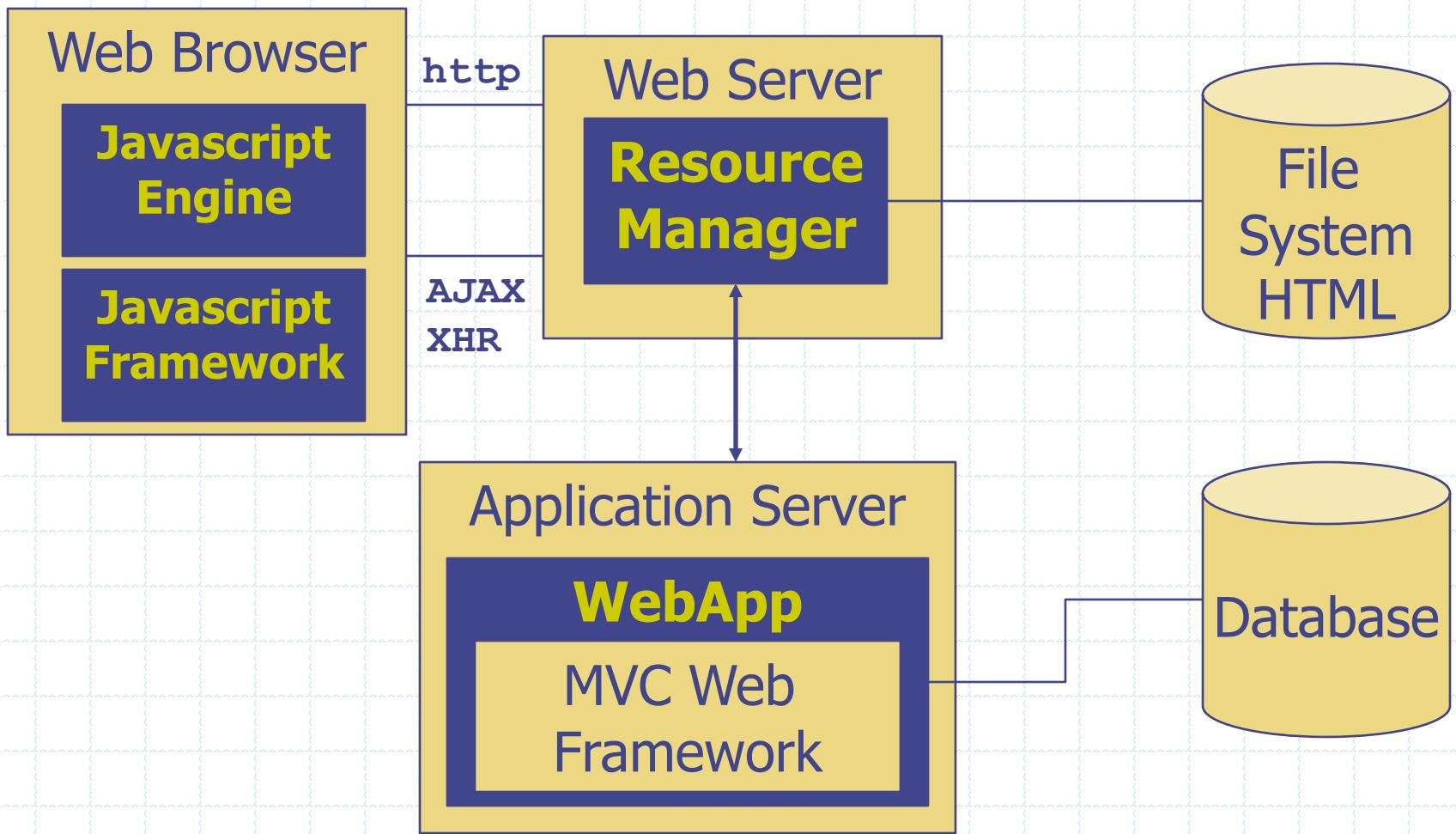
- First time ever having desktop like experience on browser-based application – amazing experience
- Can alter the UI without having to refresh the entire page

## ◆ Disadvantages

- Significant programming complexity
- Significant testing required due to browser differences in handling Ajax/XHTT
- Server-side resources must be returned from the same domain as the place that loaded the code (no cross-domain is allowed)
- Javascript libraries are required to deal with complexity - jQuery

**Example : Google Maps**

# Web 2.0+ Architecture



# Web 2.0+ Architecture

## ◆ Advantages

- Robust frameworks are developed to address Javascript complexity and Javascript compatibility
- Application functionality and richness continues to grow

## ◆ Disadvantages

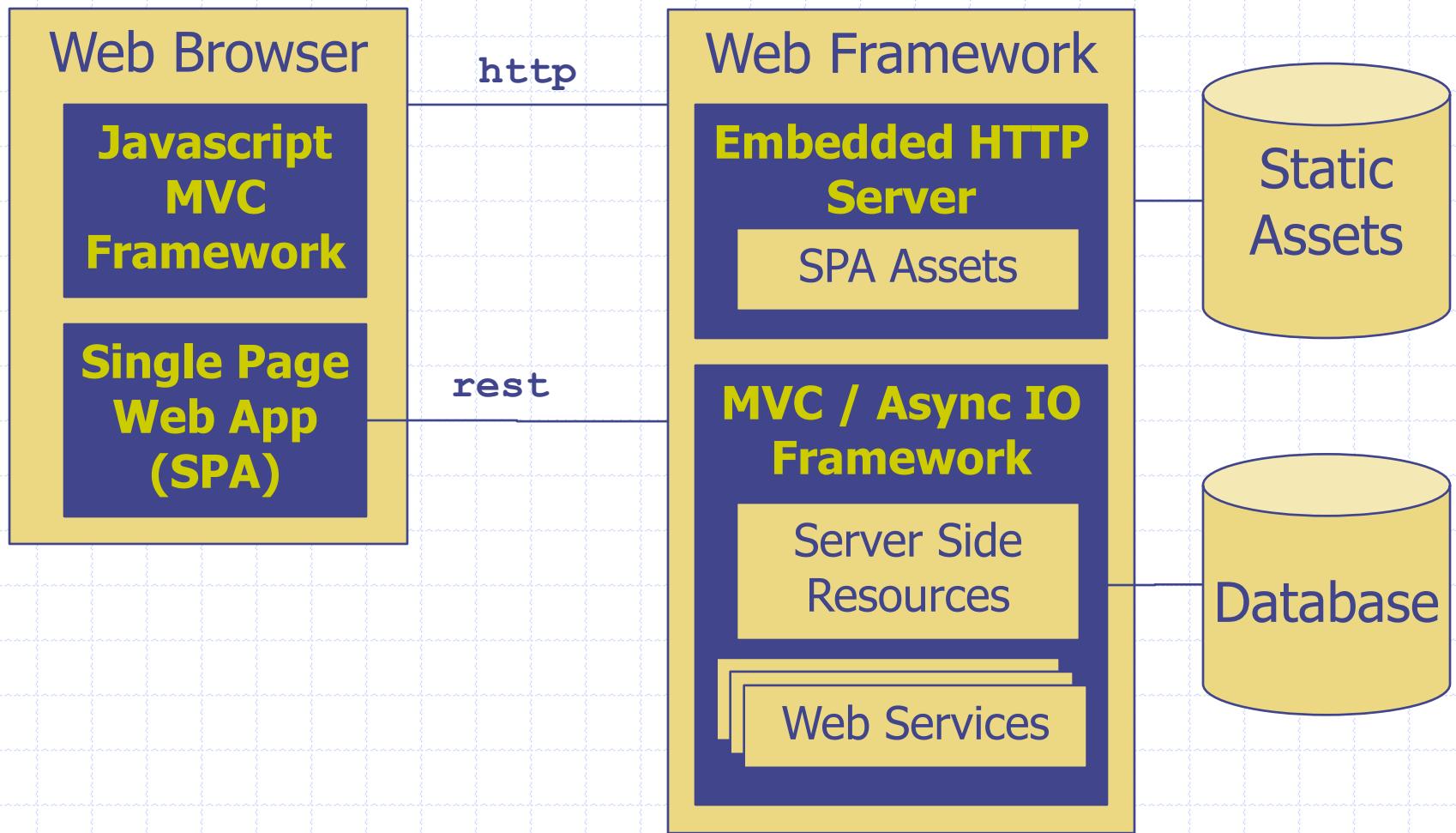
- Javascript codebases explode in size, programming model is based on pattern-matching and callbacks gets difficult to support
- New frameworks come and go almost on a weekly basis making it difficult to gain stability in the web 2.0 space

**Example : Gmail, mustache, handlebars, underscore**

# The Web 3.0 Architecture

- ◆ Application moves to the browser
  - Leverages frameworks typically found on the server to simplify Javascript
  - Enables application to be delivered from a CDN
  - Allows for the application to run disconnected using local storage
- ◆ Server primarily used to deliver data to the client
- ◆ Server used to protect secrets such as API keys and manage security given that the code on the client can be viewed from any browser

# Web 3.0 (rev 1) Architecture



# Web 3.0 (rev 1) Architecture

## ◆ Advantages

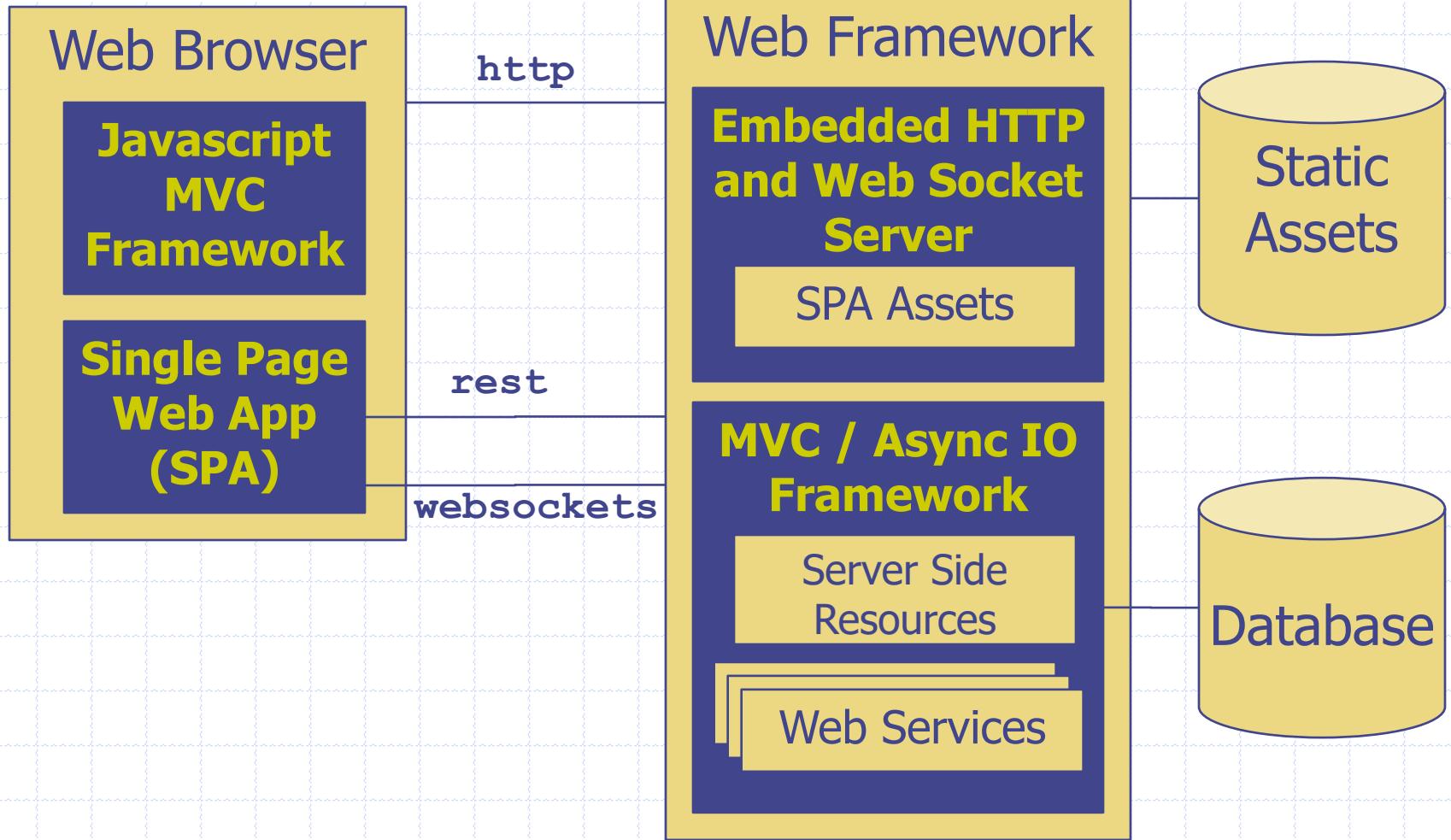
- Extremely robust applications are possible
- Frameworks finally enable Javascript as a platform– modularization, scope management, dependency injection, promise/futures, etc
- Frameworks enable full lifecycle debugging, and robust build/configuration management
- Cacheable and CDN friendly – only data needs to flow back to browser app
- Enables “reactive” programming model and async-IO to improve scalability

## ◆ Disadvantages

- Still need to master Javascript
- Polyglot (is advantage and disadvantage)
- Security might be challenging – can't maintain secrets in client app (e.g., “View Code”)
- Traditional application servers still request/reply based, limits scalability in some cases

**Example : AngularJS, BackboneJS, Node.js**

# Web 3.0 (rev 2) Architecture



# Web 3.0 (rev 2) Architecture

## ◆ Advantages

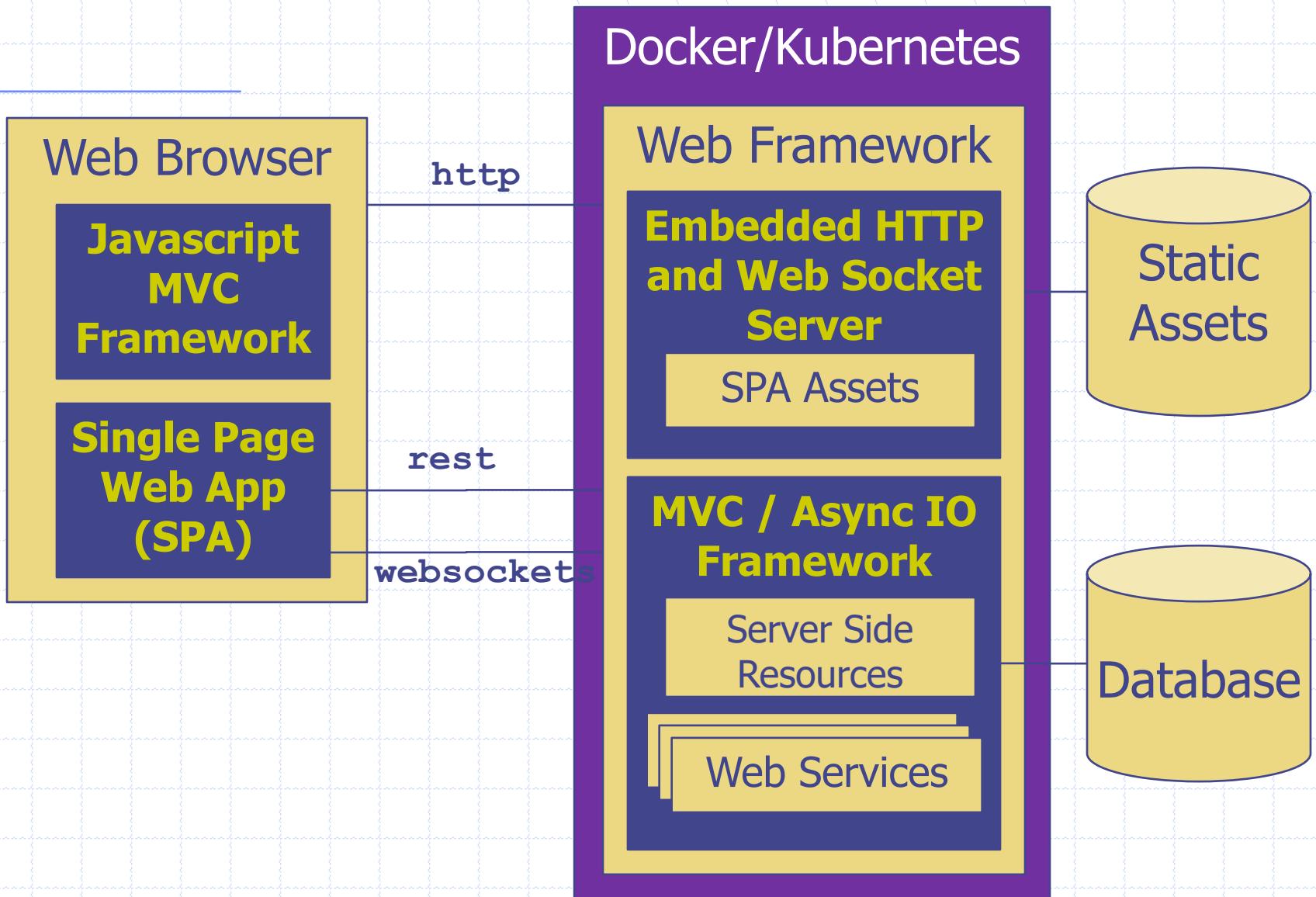
- Able to support for the first time real time architectures for applications like trading, chat, gaming etc
- Eliminate overhead of using the HTTP protocol
- Messages can be initiated from either the client (browser) or server

## ◆ Disadvantages

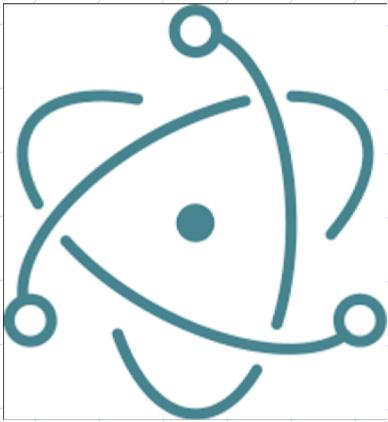
- Only modern browsers support, although web sockets can be simulated with XHR long polling
- Some proxies and HTTP infrastructure don't know how to deal with web sockets (this will change)

**Example : SockJS, socket.io, Node.js, Play!**

# Web 3.0 (rev 3) Architecture



# So Why Are Modern Web Architectures Useful

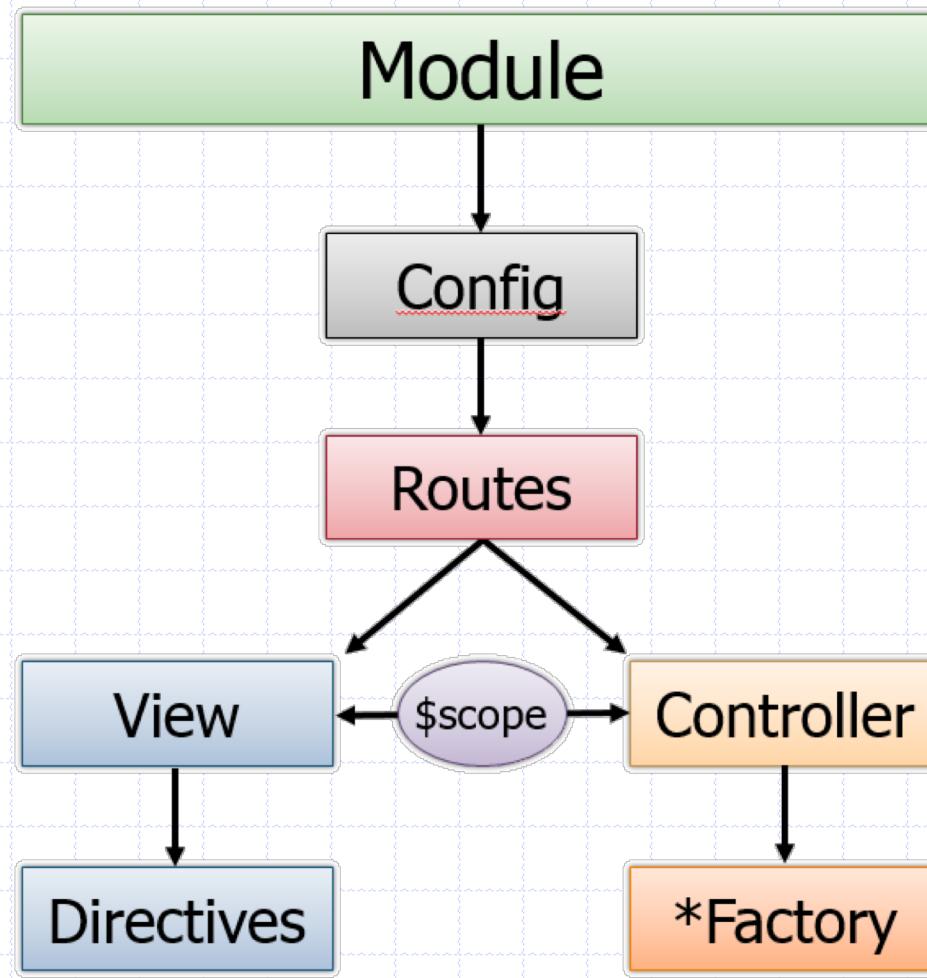


Desktop Apps via  
Electron

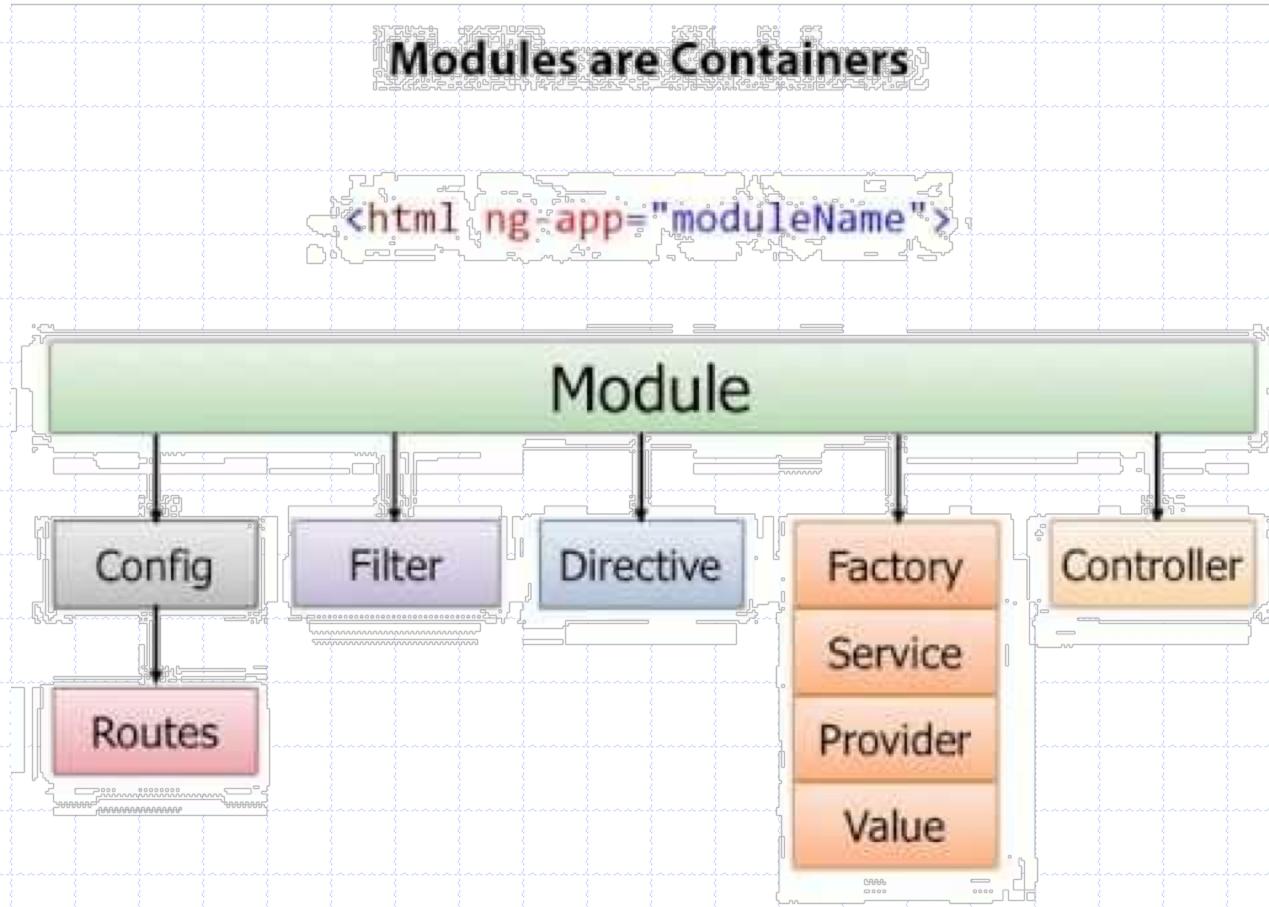
Engaging Web Apps

Native Mobile Apps

# Modern Web Architectures: Front End MVC - AngularJS Example



# Example: AngularJS Architecture



# Example: Lets Look at Angular via Example

We will look at some examples from here:

<https://curran.github.io/screencasts/introToAngular/exampleViewer/#/>

# Angular 1.x

The good and the bad discussion, what do you think?

# Javascript vs Typescript

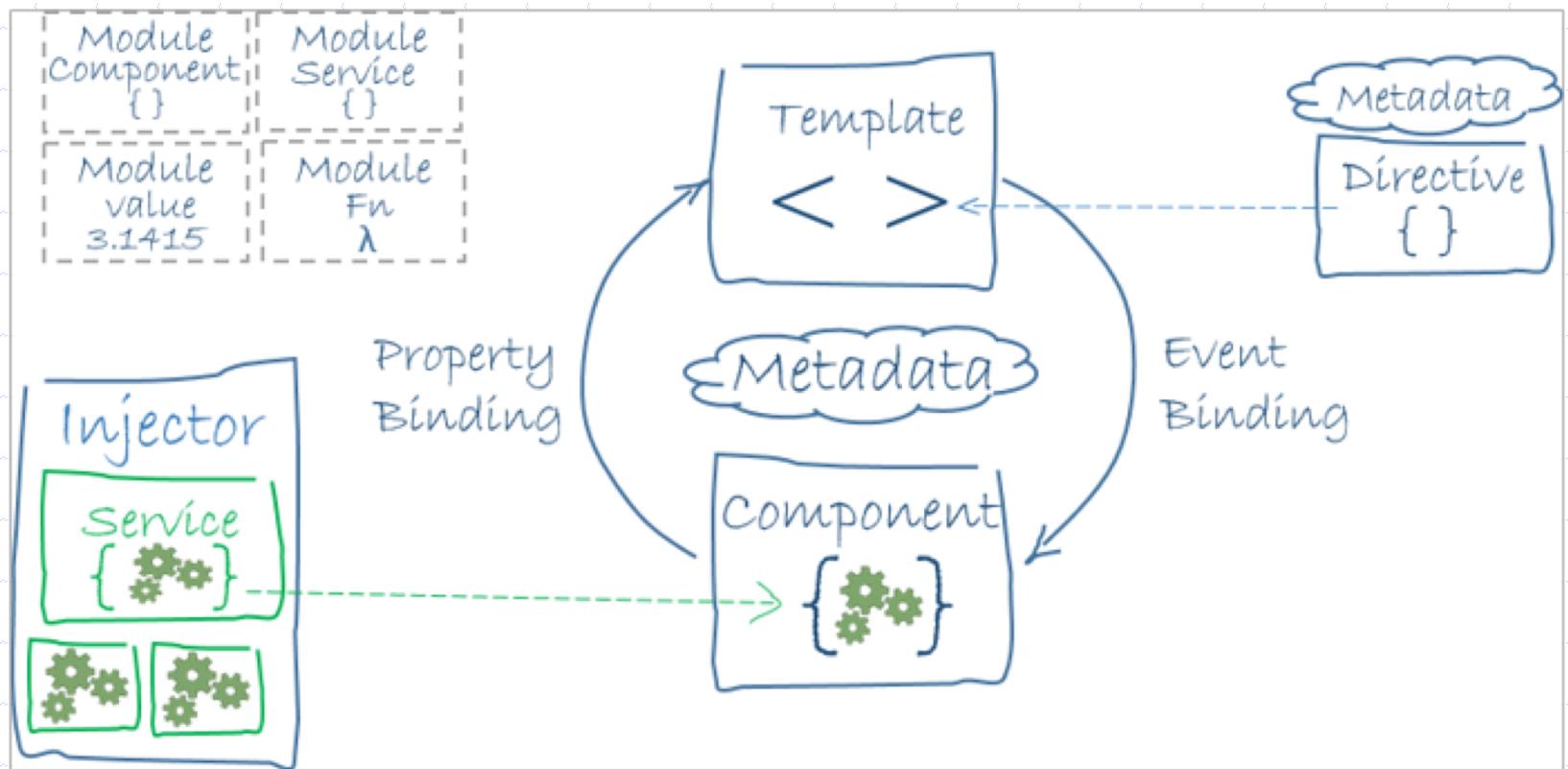
---

JavaScript has grown up a good bit, but we will actually be using TypeScript instead because it has many advantages

Lets take a look at the Typescript Playground:  
<https://www.typescriptlang.org/>

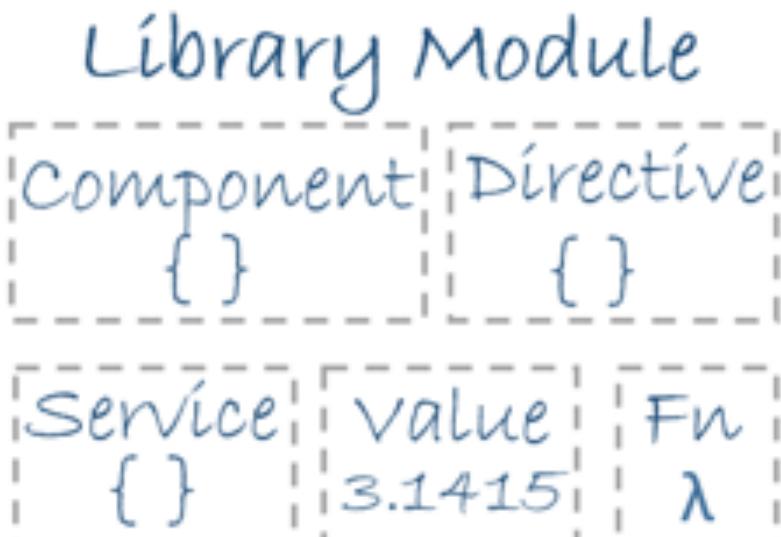
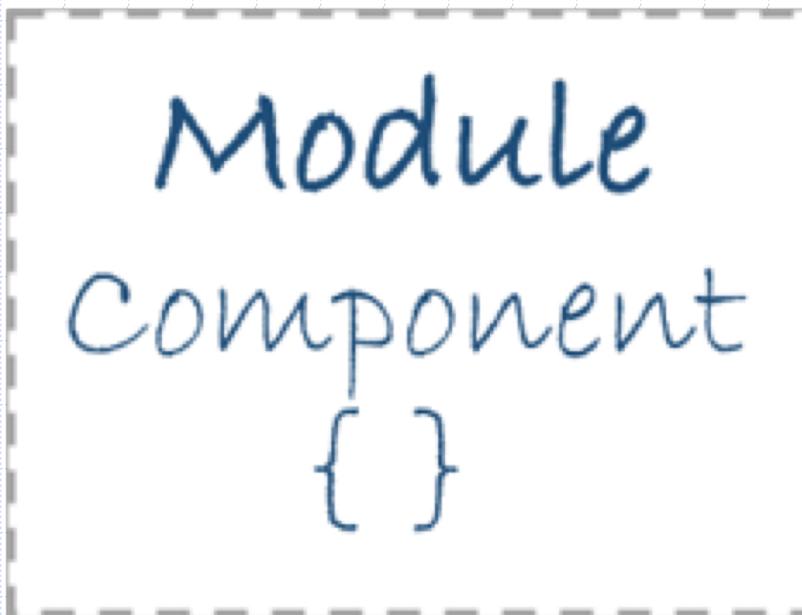
# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



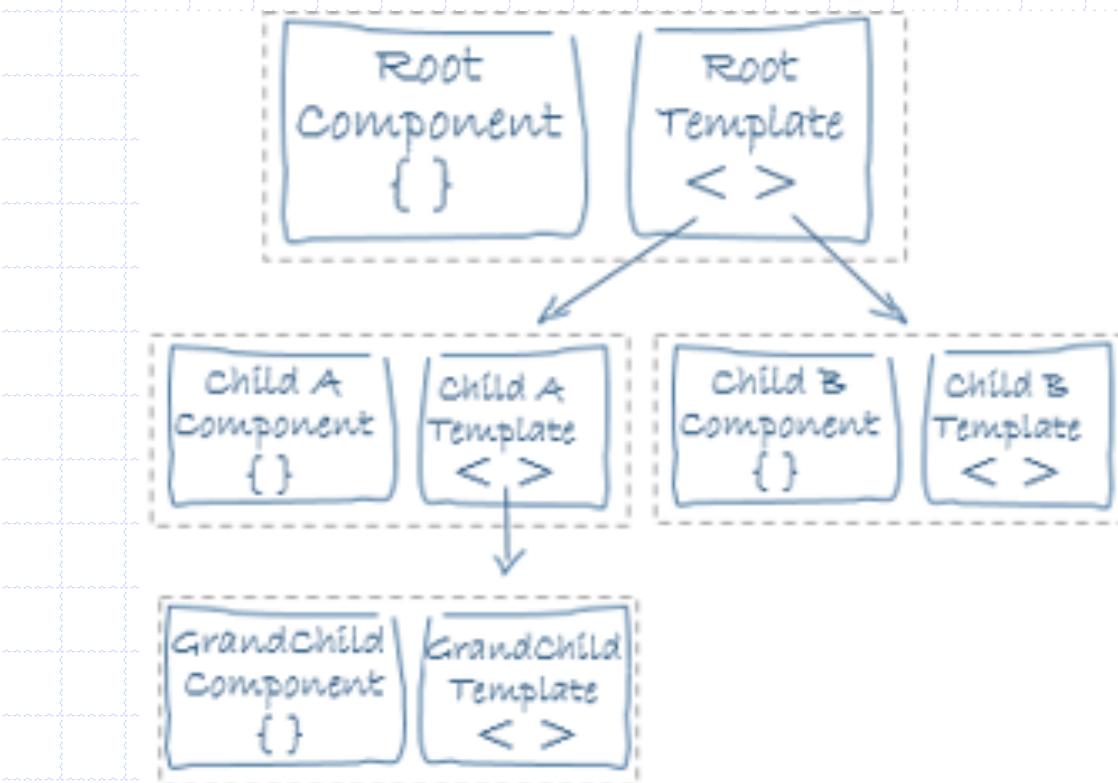
# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



src/app/hero-list.component.ts (metadata)

```
@Component({
  selector:    'hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
})
export class HeroListComponent implements OnInit {
  /* . . . */
}
```

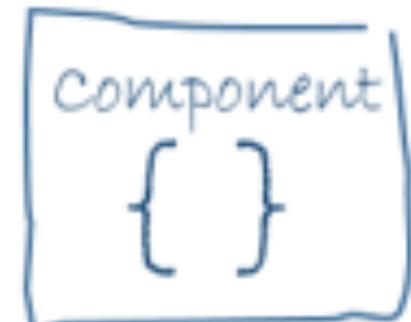
# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



src/app/hero-list.component.ts (metadata)

```
@Component({
  selector:    'hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
})
export class HeroListComponent implements OnInit {
  /* . . . */
}
```



# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>

DOM

  {{value}}

  [property] = "value"

  (event) = "handler"

  [(ng-model)] = "property"

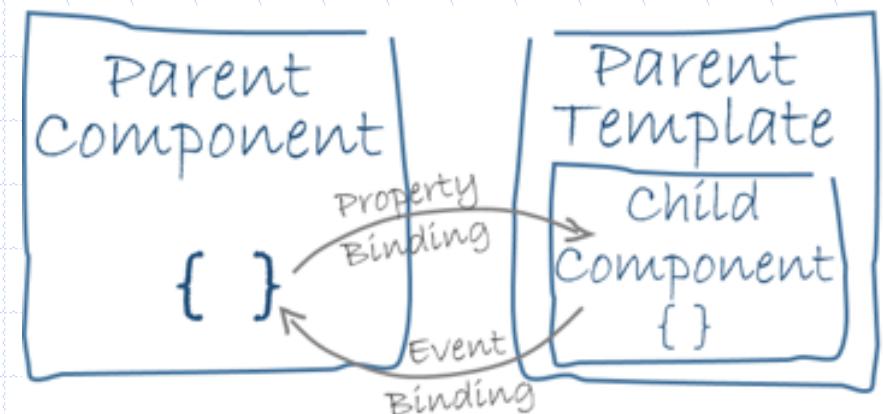
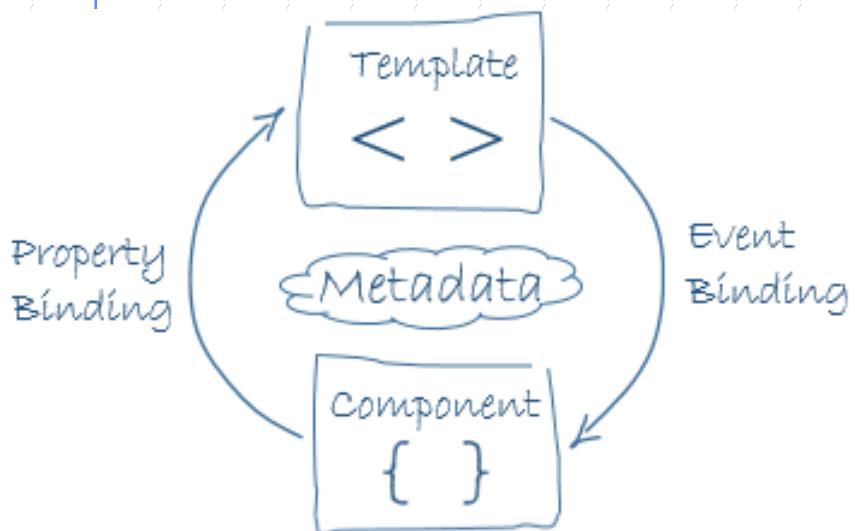
COMPONENT

src/app/hero-list.component.html (binding)

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

# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



src/app/hero-list.component.html (structural)

```
<li *ngFor="let hero of heroes"></li>
<hero-detail *ngIf="selectedHero"></hero-detail>
```

# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



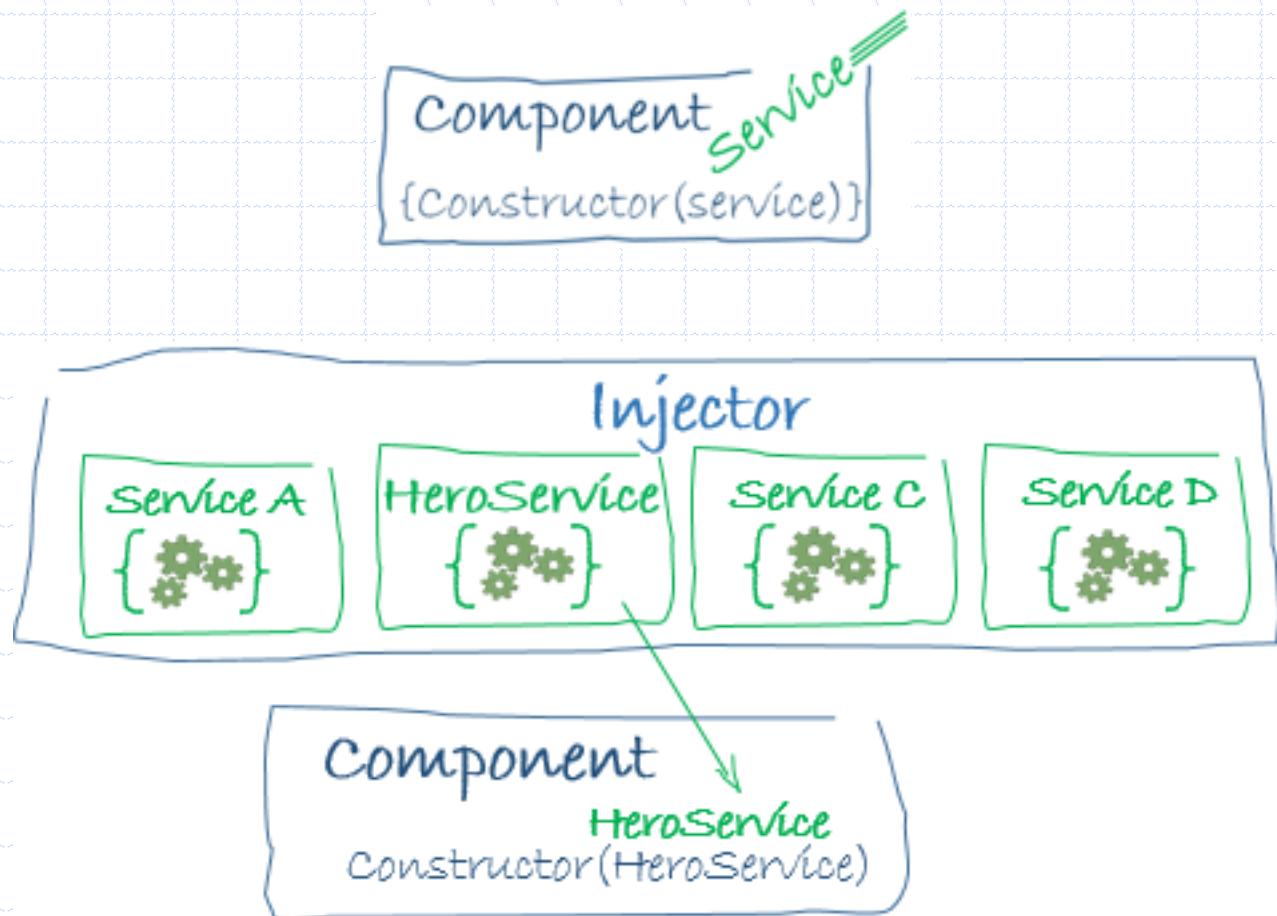
src/app/logger.service.ts (class)

```
export class Logger {  
  log(msg: any) { console.log(msg); }  
  error(msg: any) { console.error(msg); }  
  warn(msg: any) { console.warn(msg); }  
}
```

A service is just a class, recommend decorating with  
`@Injectable()`

# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>



# Angular.io (Angular 4 currently)

SEE: <https://angular.io/guide/architecture>

src/app/app.module.ts (module providers)

```
providers: [
  BackendService,
  HeroService,
  Logger
```

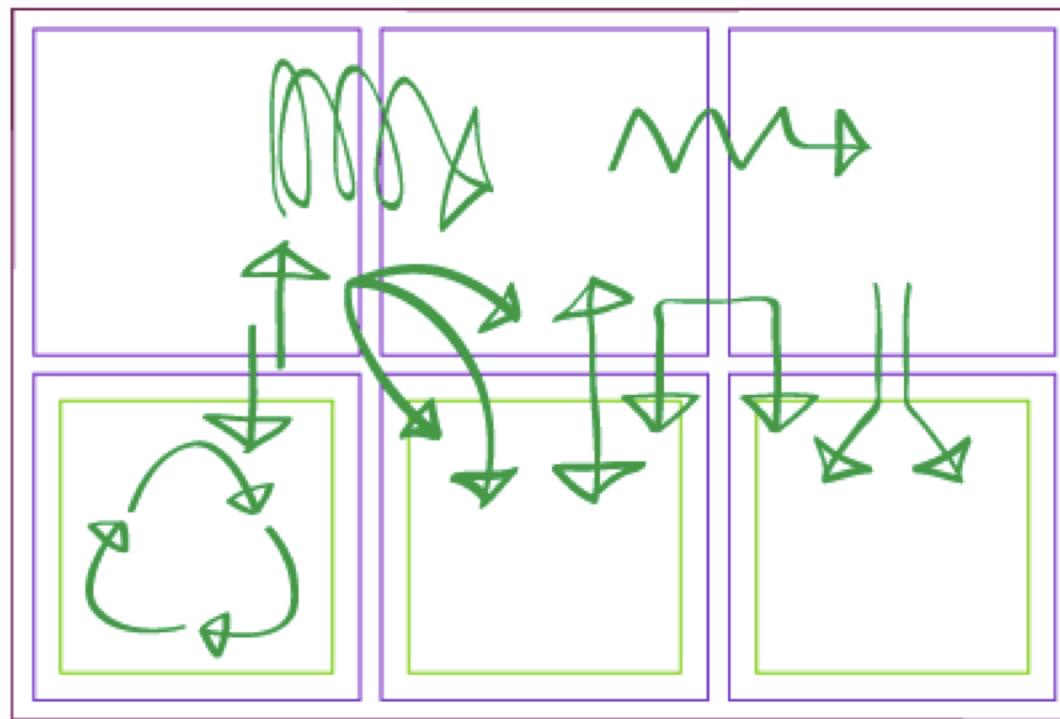
Alternatively, register at a component level in the `providers` property of the

src/app/hero-list.component.ts (component providers)

```
@Component({
  selector: 'hero-list',
  templateUrl: './hero-list.component.html',
  providers: [ HeroService ]
```

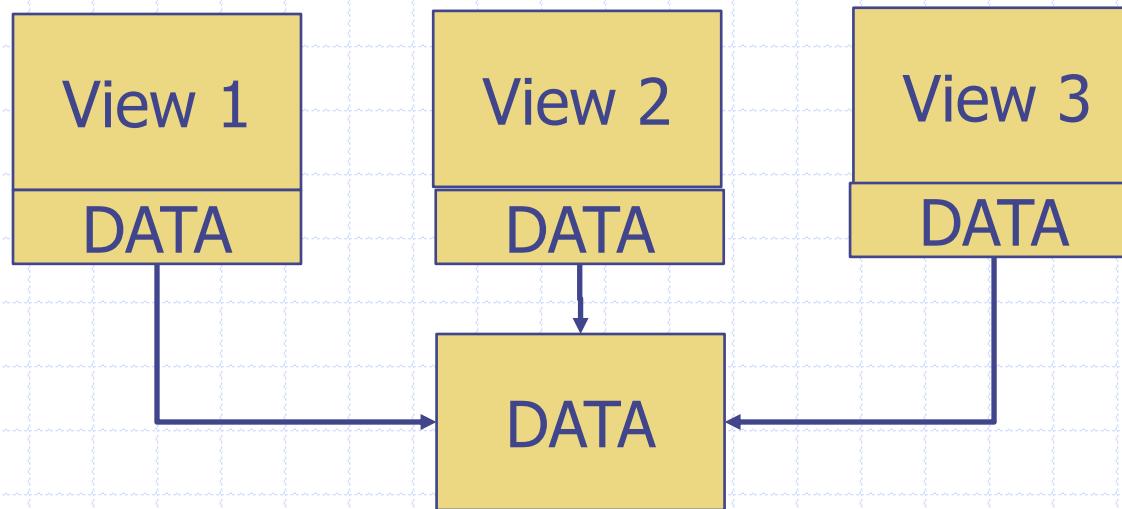
# State Management

- One of the main problems with web component frameworks like angular is related to managing application state



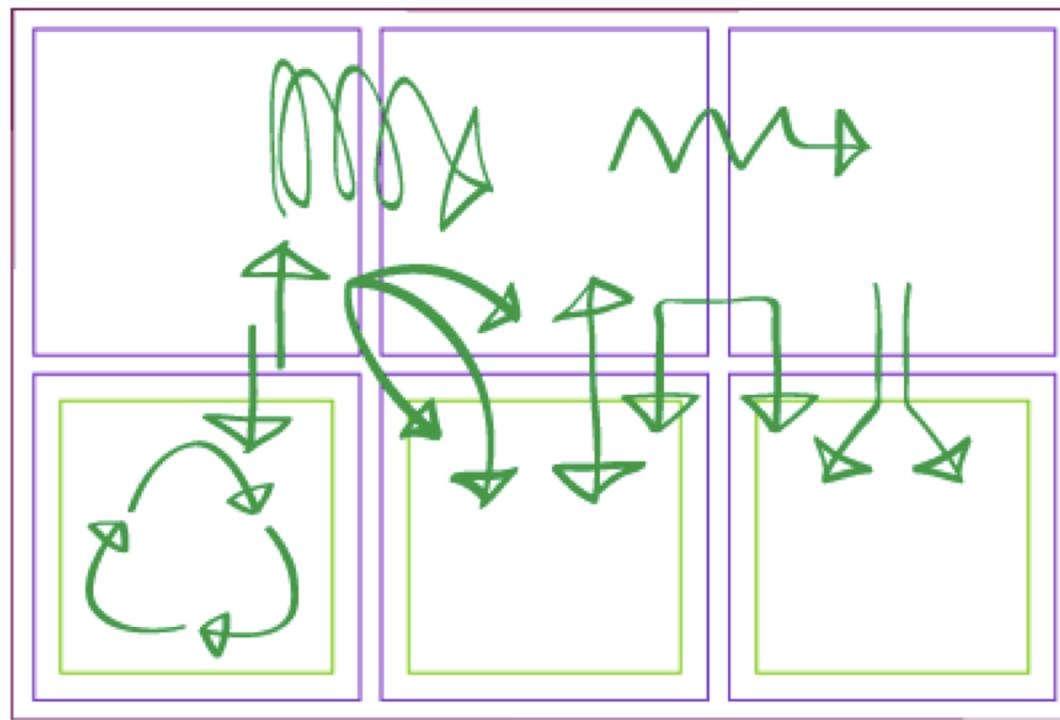
# State Management

- Commonly views cache copies of the same data, how do things stay consistant



# State Management

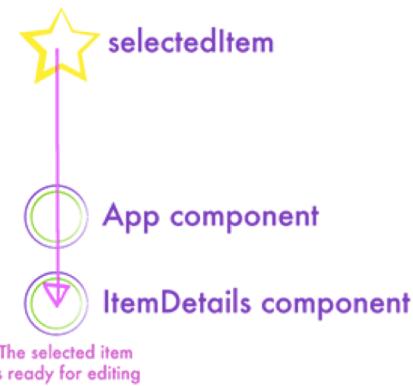
- To keep state manageable, a common best practice is to push state down via properties, and use events to pass state changes up



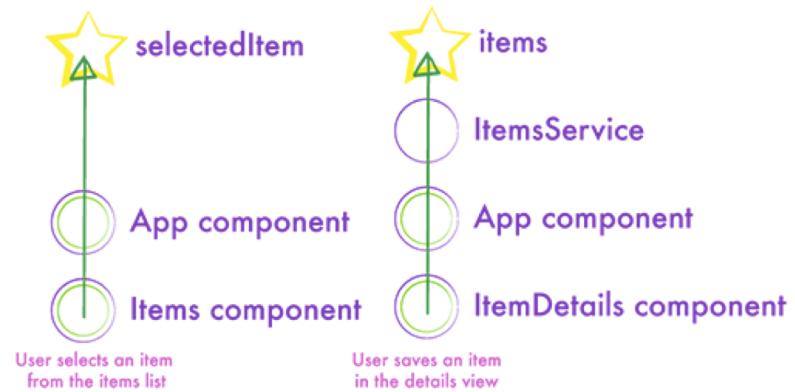
# State Management

Pushing State Changes Down, and Reacting to Events Up works well if the components fall into hierarchies

State Down



Events Up



# State Management

With more global event management, Angular supports propagating state via, services (singletons), event emitters and observables

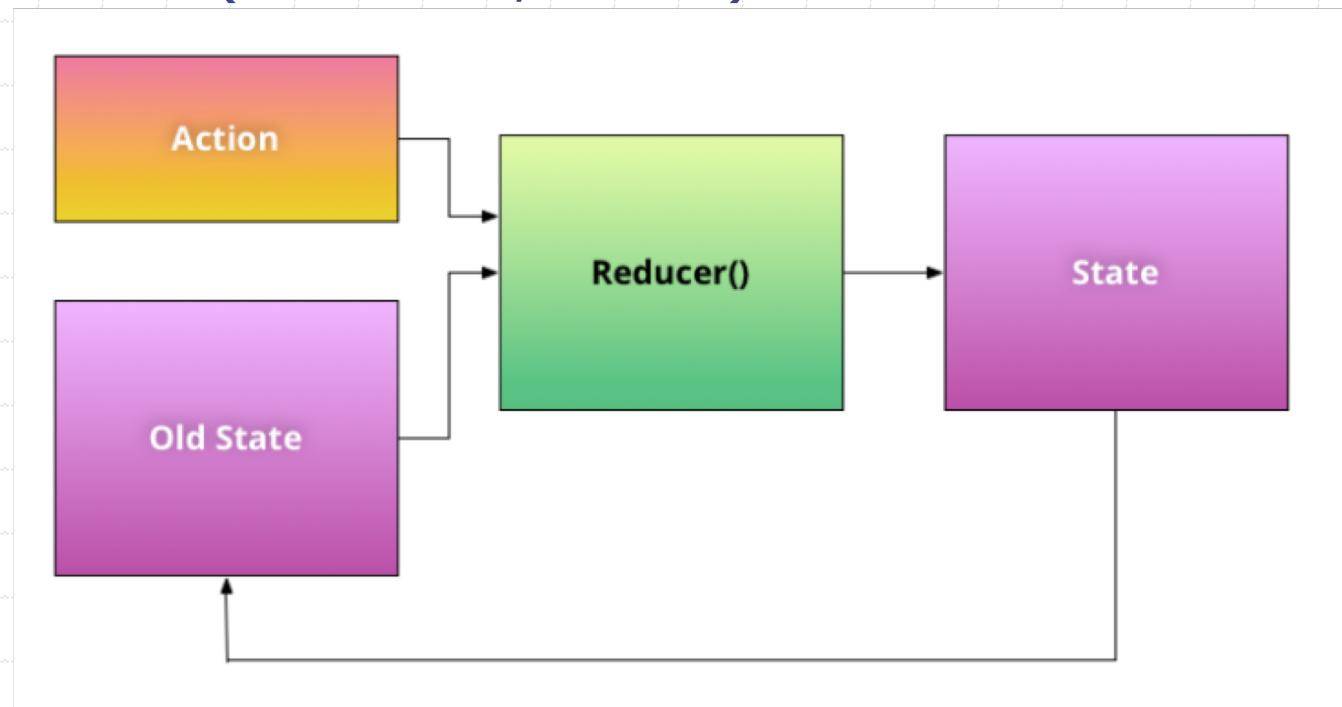
Lets take a look at this, but it gets complex fast...

# State Management – Flux / Redux

SEE: <http://blog.ng-book.com/introduction-to-redux-with-typescript-and-angular-2/>

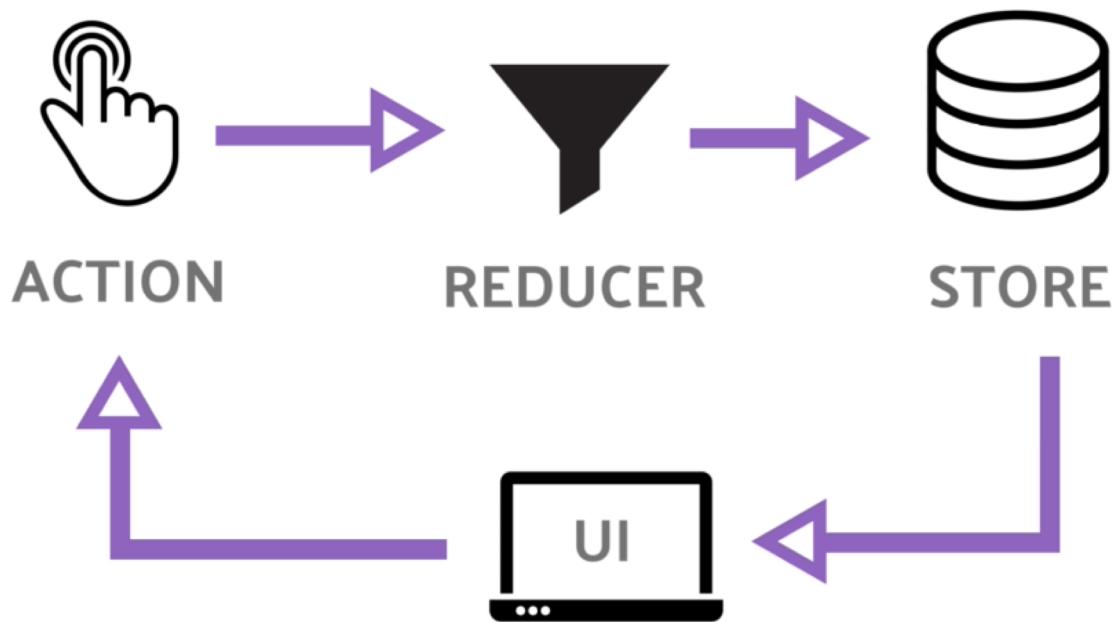
Embrace immutability – functional programming techniques

$(\text{Old\_State}, \text{Action}) \Rightarrow \text{New\_State}$

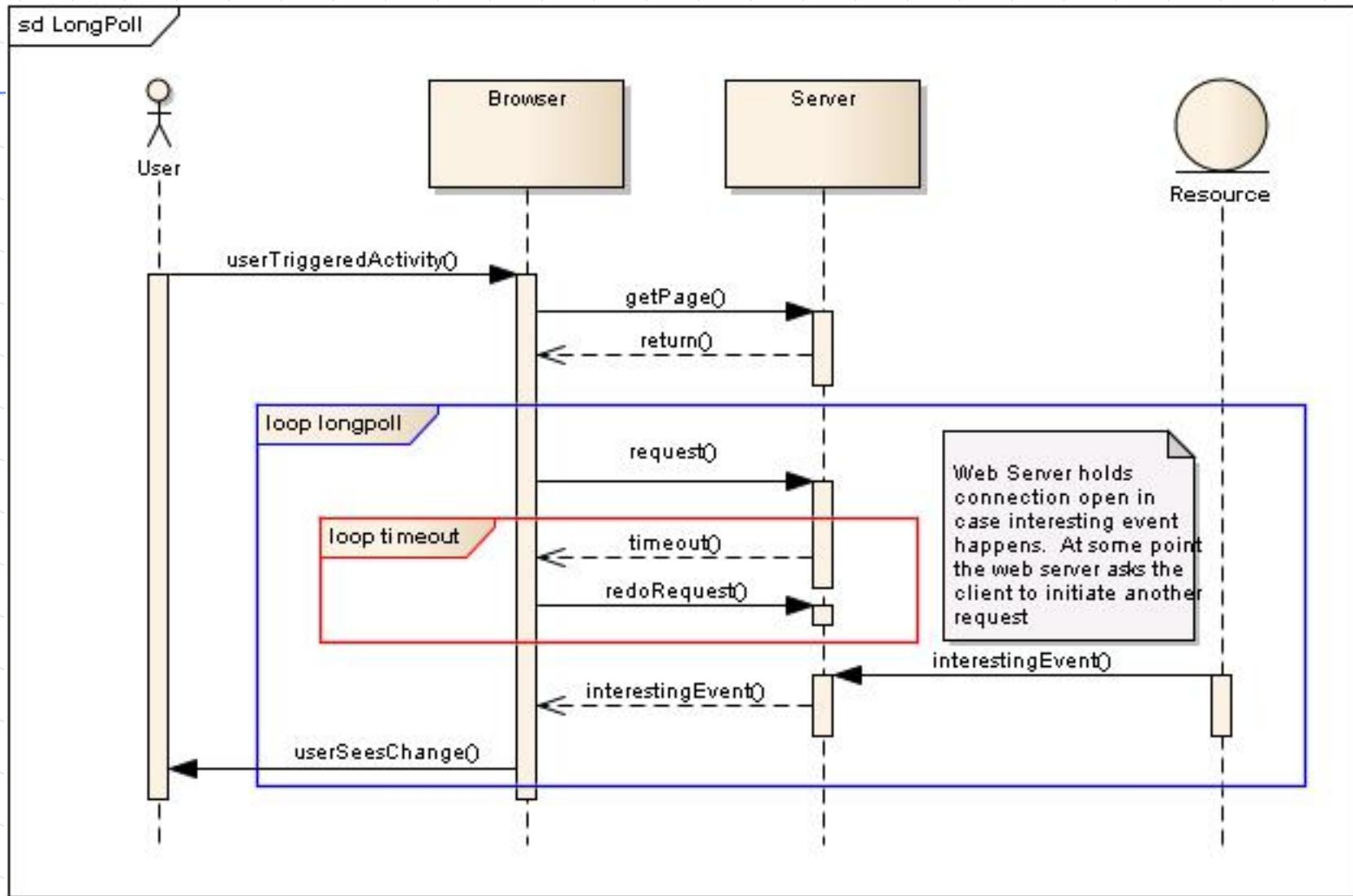


# State Management – Flux / Redux

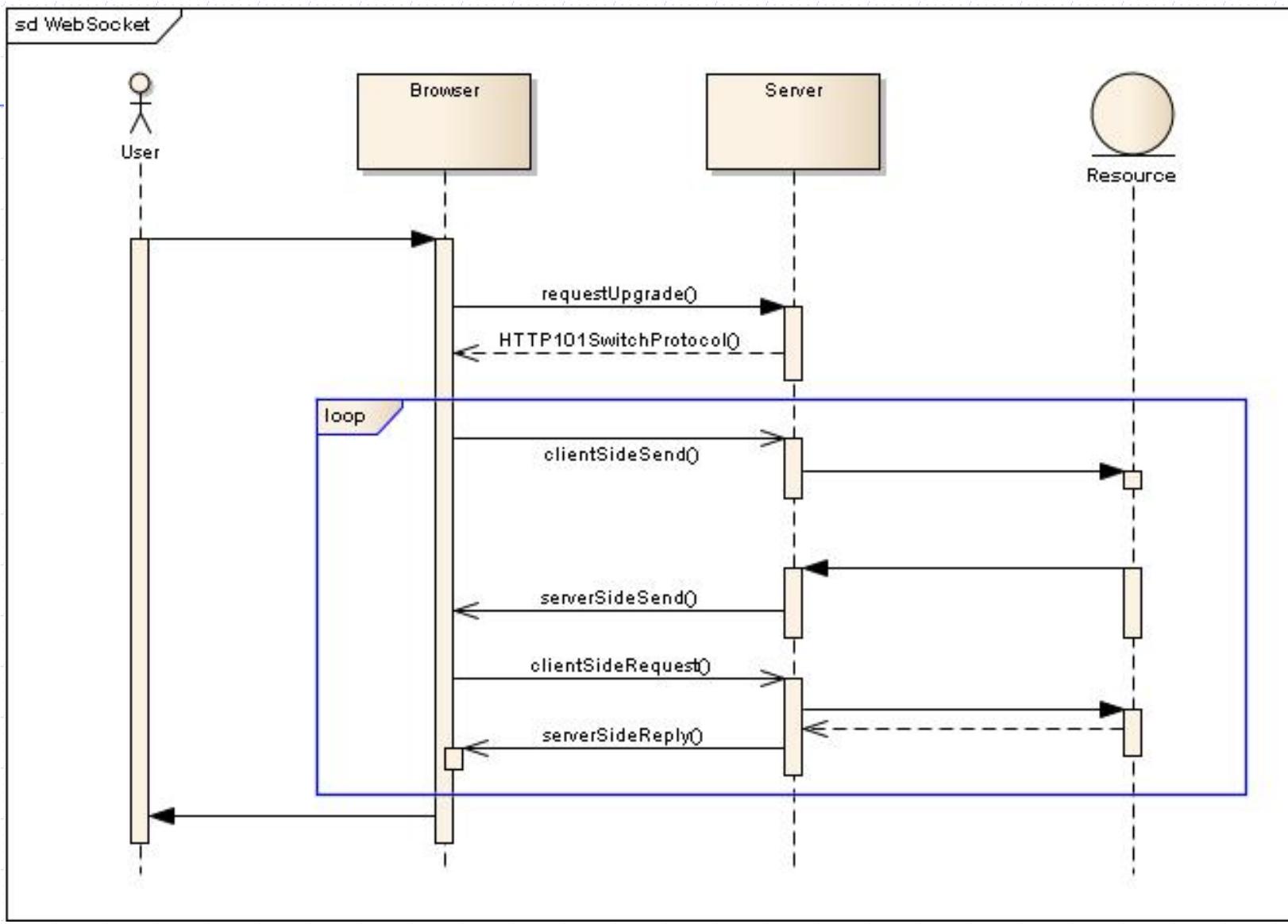
SEE: <https://angularfirebase.com/lessons/angular-ngrx-redux-starter-guide/>



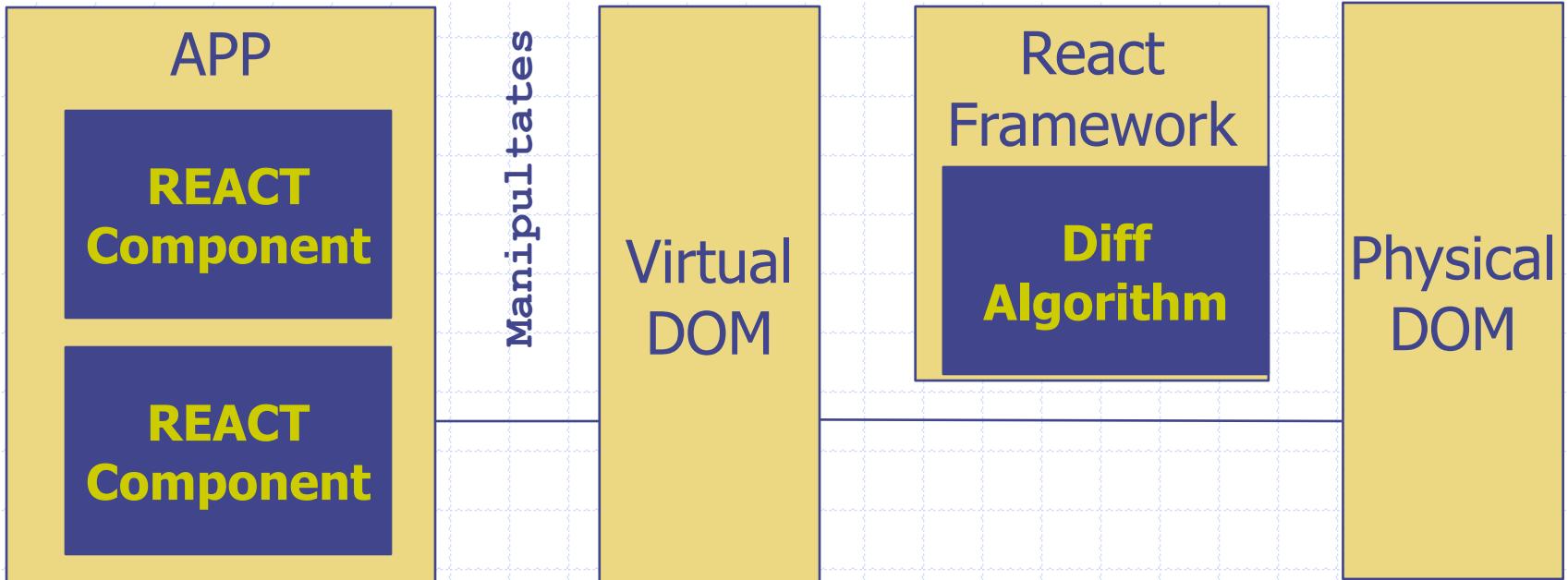
# How it works: XHR Long Poll



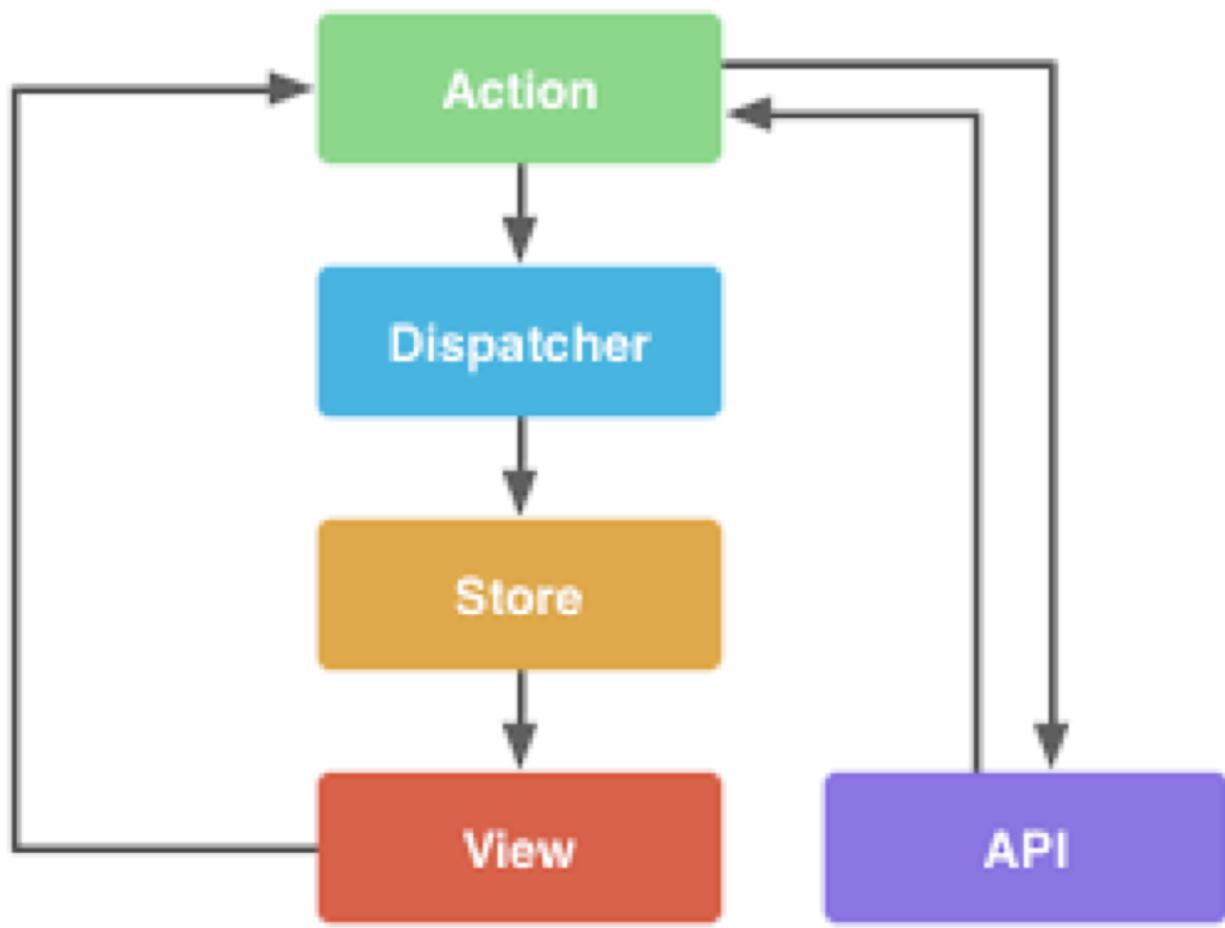
# How it works: Web Sockets



# React Architecture



# Flux Architecture – Data Travels One Way



# References

- ◆ AngularJS: <http://angularjs.org/>
- ◆ The WebSockets protocol: <http://tools.ietf.org/html/rfc6455>
- ◆ Node.js: <http://www.nodejs.org/>
- ◆ Play! Framework: <http://www.playframework.com/>