Introduction to MVW

Minko Gechev
github.com/mgechev
twitter.com/mgechev

Topics for discussion

- Single-Page Applications
- What is MVW?
- Why I need MVW?
- MVC
- Current landscape
- Backbone
  - Event
  - View
  - Model
  - Collection
  - Router

Single-Page Applications

Rich Internet Application

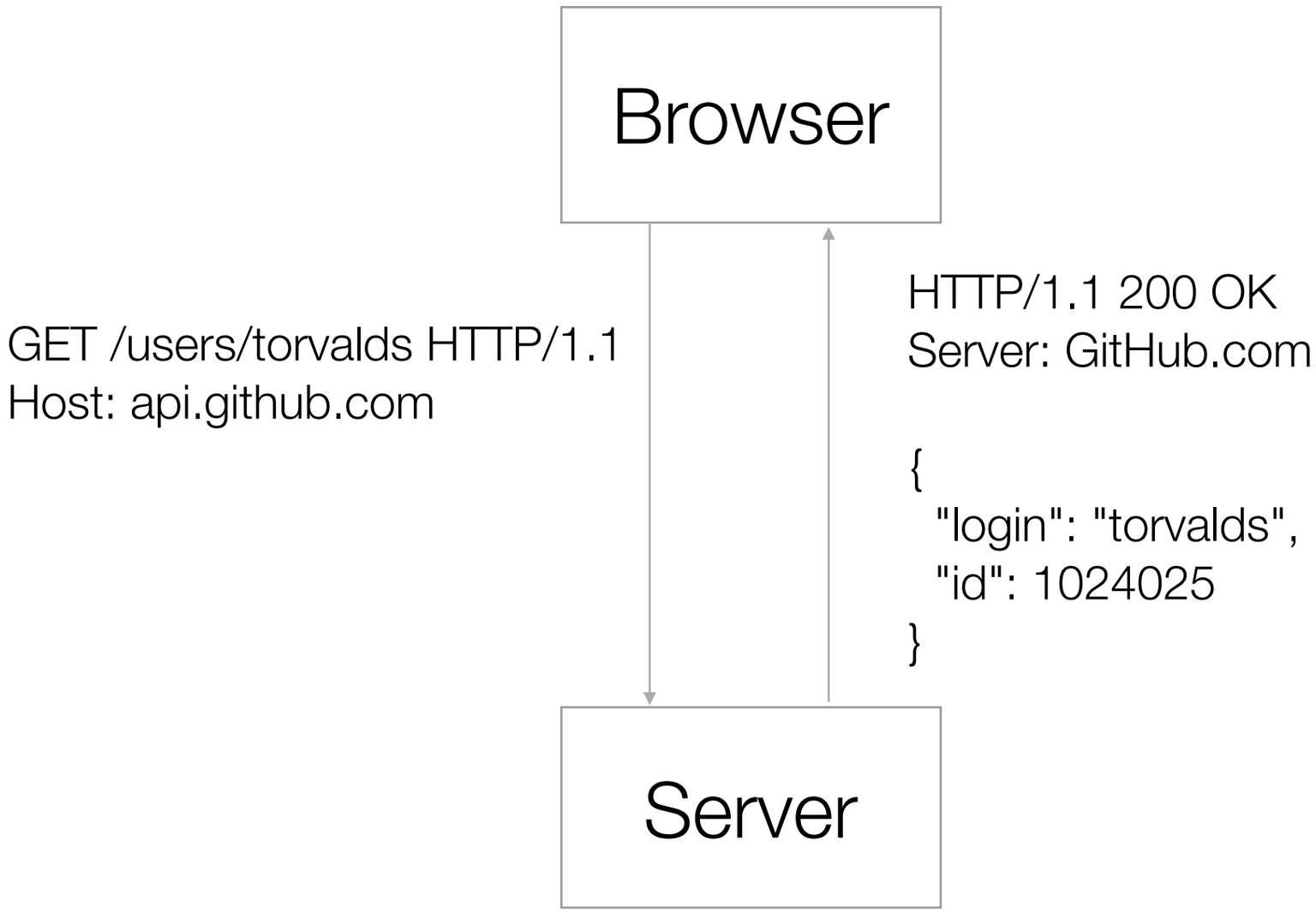
"A rich Internet application (RIA) is a Web application that has many of the characteristics of desktop application software, typically delivered by way of a site-specific browser, a browser plug-in, an independent sandbox, extensive use of JavaScript, or a virtual machine."

Single-Page Application

"A single-page application (SPA), also known as single-page interface (SPI), is a web application or web site that fits on a single web page with the goal of providing a more fluid user experience akin to a desktop application."

- Loads all the necessary content with the initial page load
- Loads the content lazily based on events

Nothing more than…



Problems with large-scale JavaScript applications

- JavaScript is weakly typed
- JavaScript has no classes and will never have (yes, even after ES6)
- Testability
- Spaghetti code
- A lot of boilerplates

MVW

Model View Controller
Model View Presenter
Model View View-Model
Model View **Whatever…**


Holy shit
I'm Batman

Why MVW?

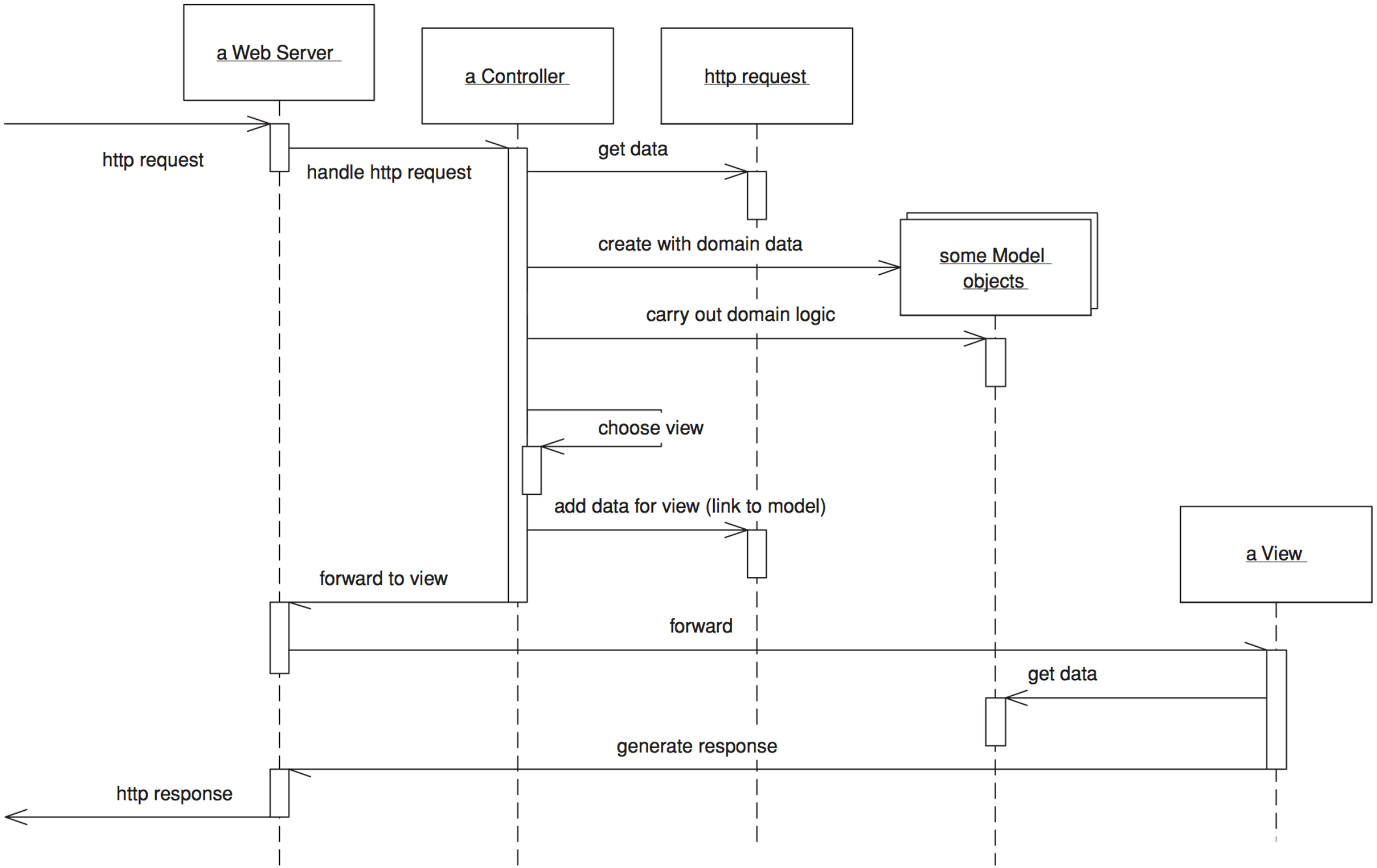
Separation of Concerns

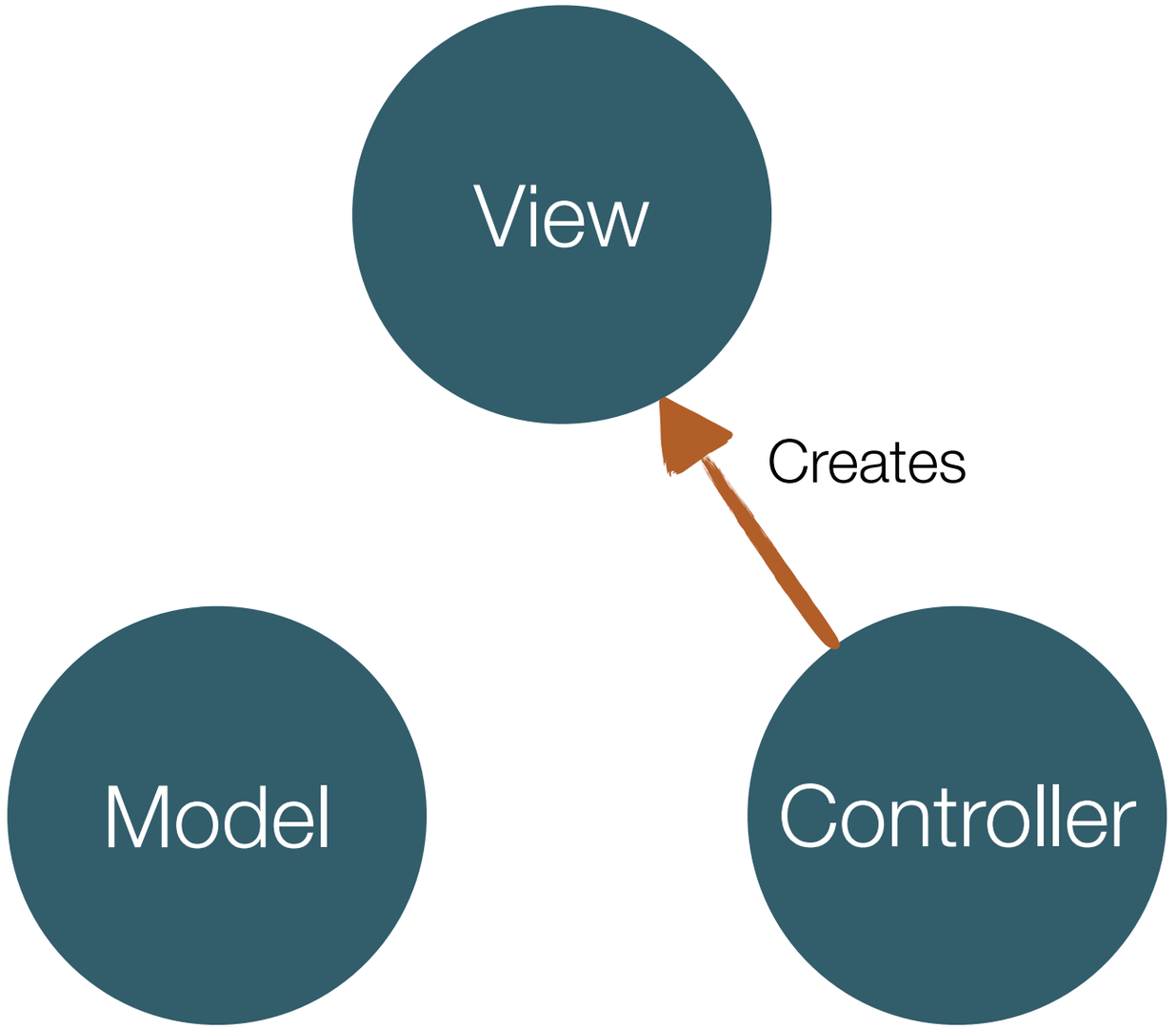
MVC

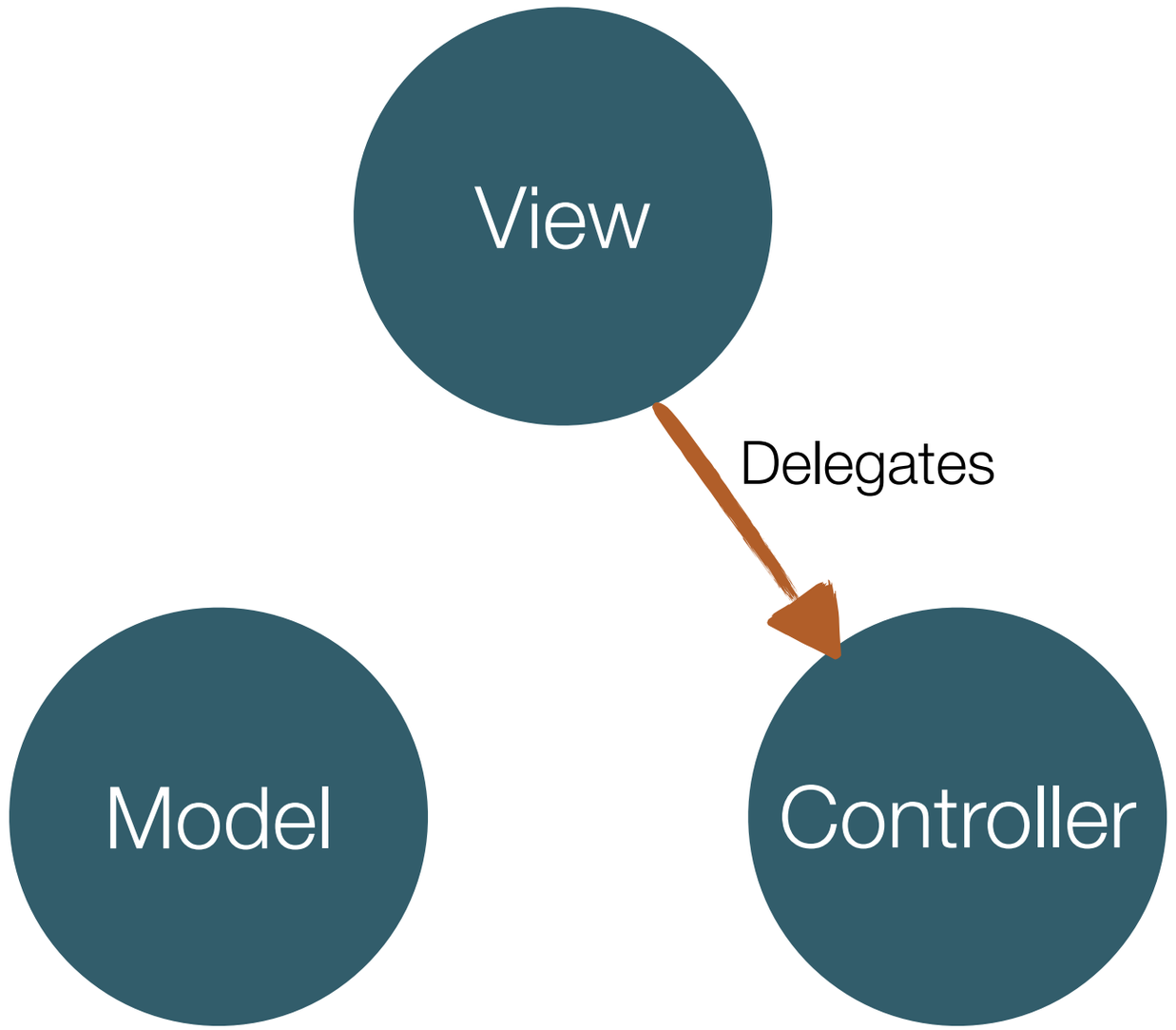
Model View Controller

- Initially introduced in SmallTalk-80
- Used in:
  - Ruby on Rails
  - ASP.NET MVC
  - Django
  - Backbone?
  - etc…

MVC (server-side)




View
Creates
Model
Controller

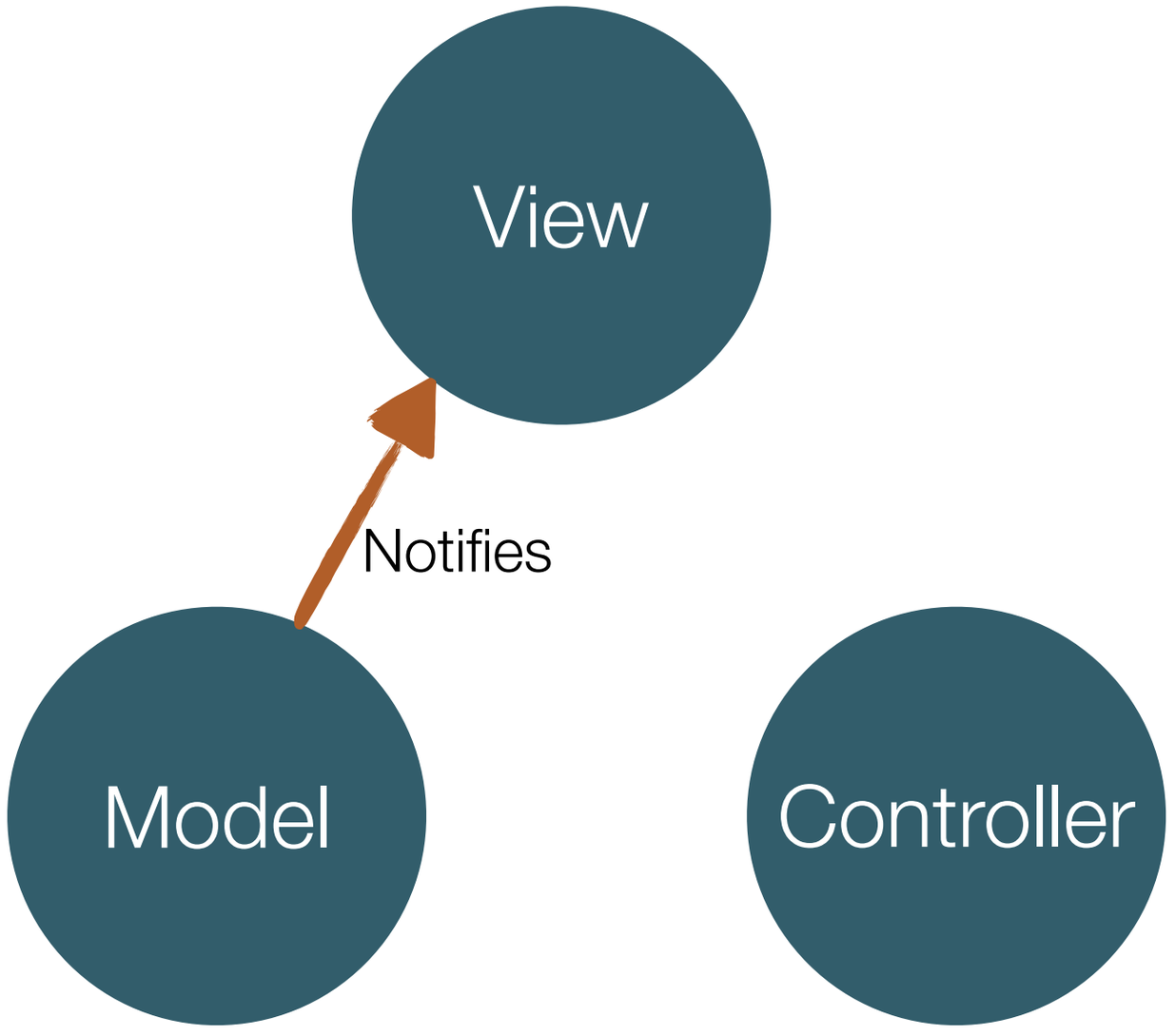

View
Delegates
Model
Controller

View

Model

Manipulates

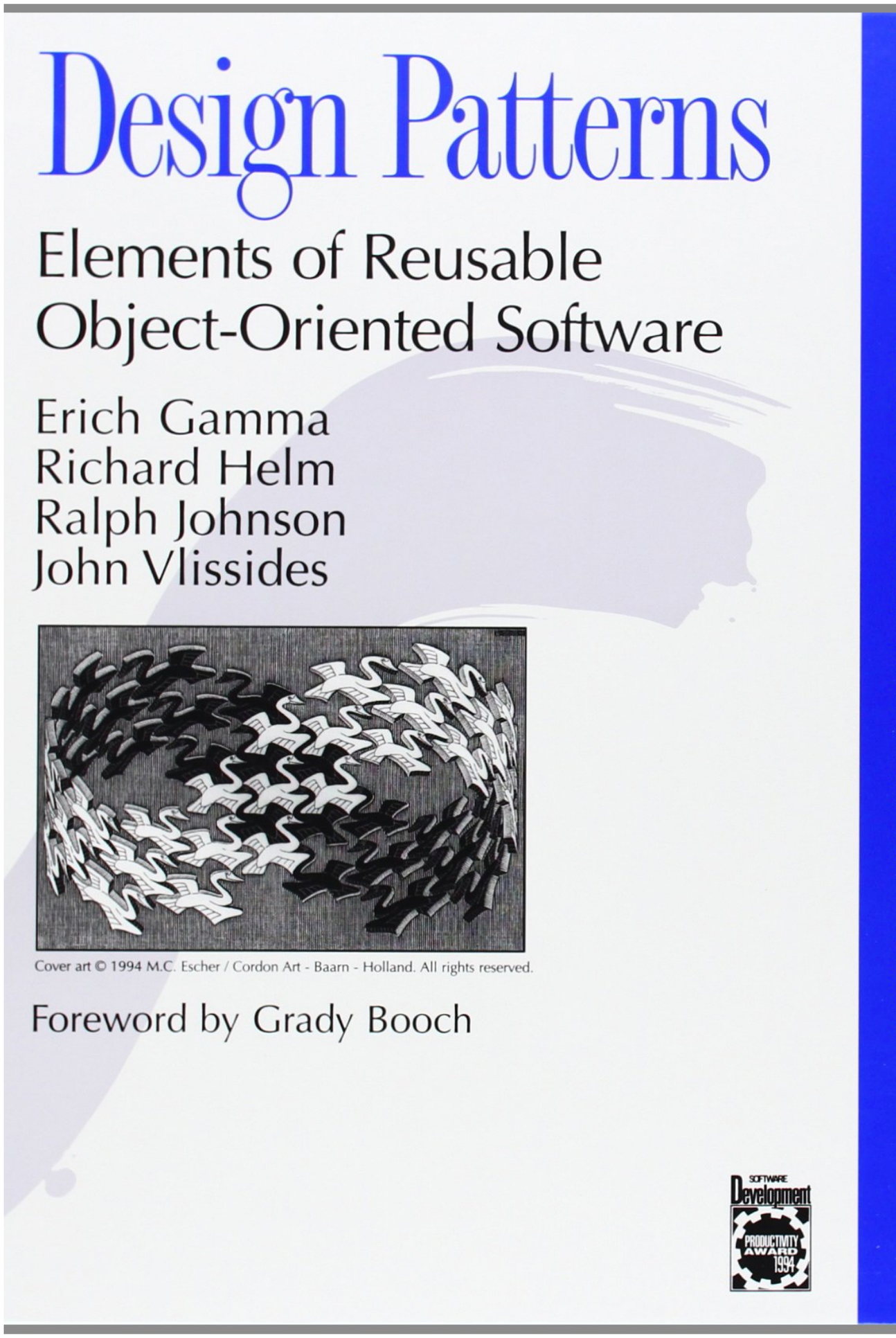
Controller


View
Notifies
Model
Controller

MVC in Patterns

According to “Gang of Four” MVC is: “…set of classes to build a user interface…”

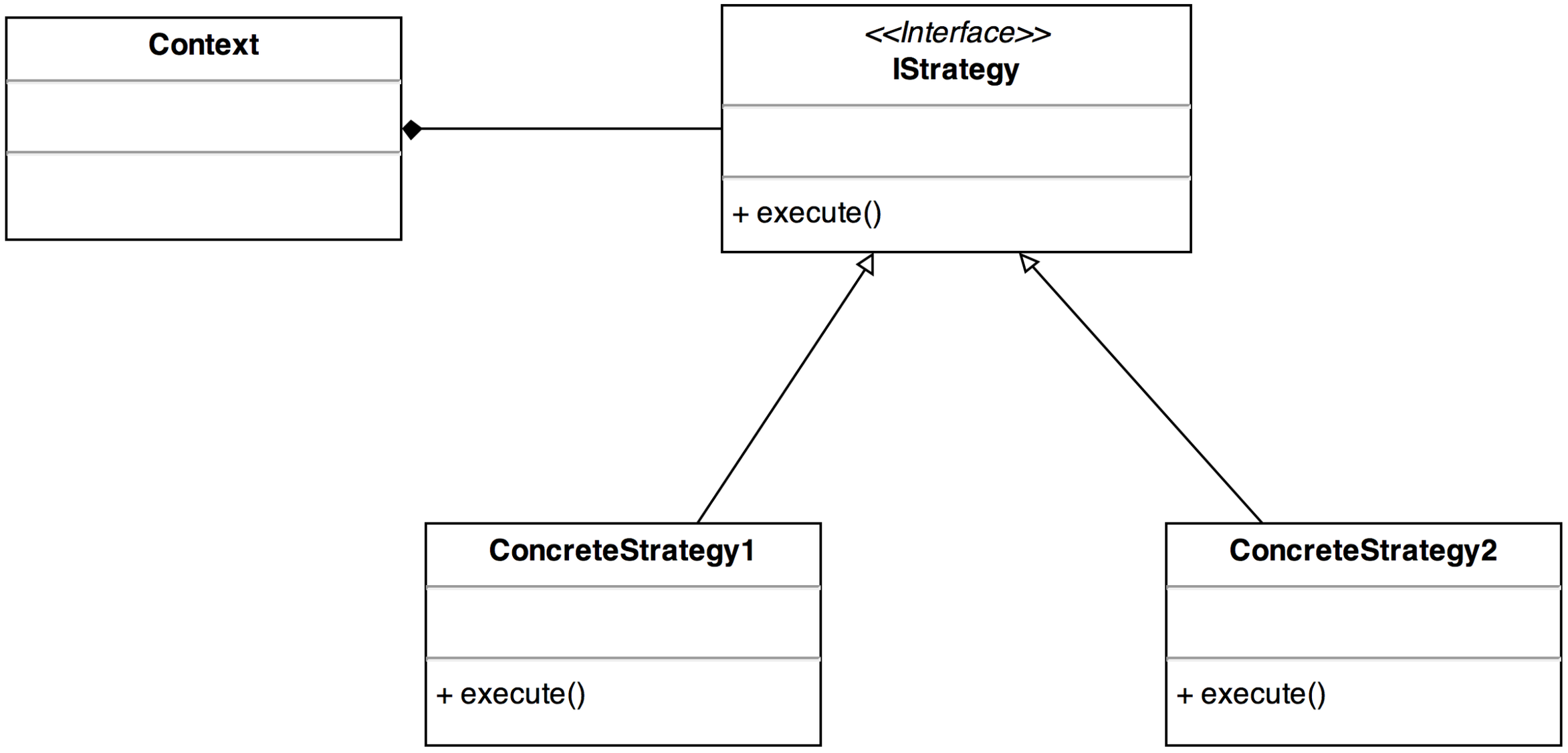
- Observer
- Strategy
- Composite



Strategy

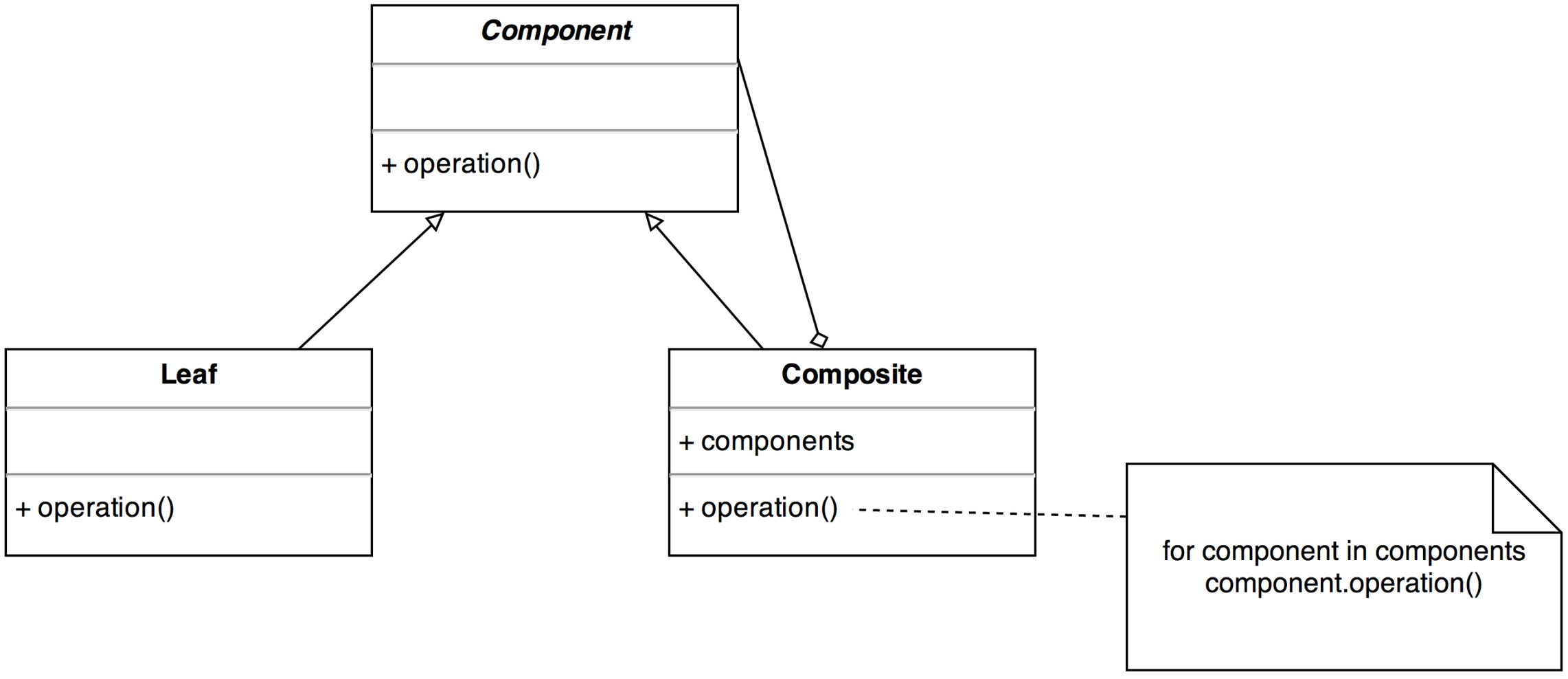
In computer programming, the strategy pattern (also known as the policy pattern) is a software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern:

- defines a family of algorithms,
- encapsulates each algorithm, and
- makes the algorithms interchangeable within that family.


Context
<<Interface>>
IStrategy
+ execute()
ConcreteStrategy1
+ execute()
ConcreteStrategy2
+ execute()

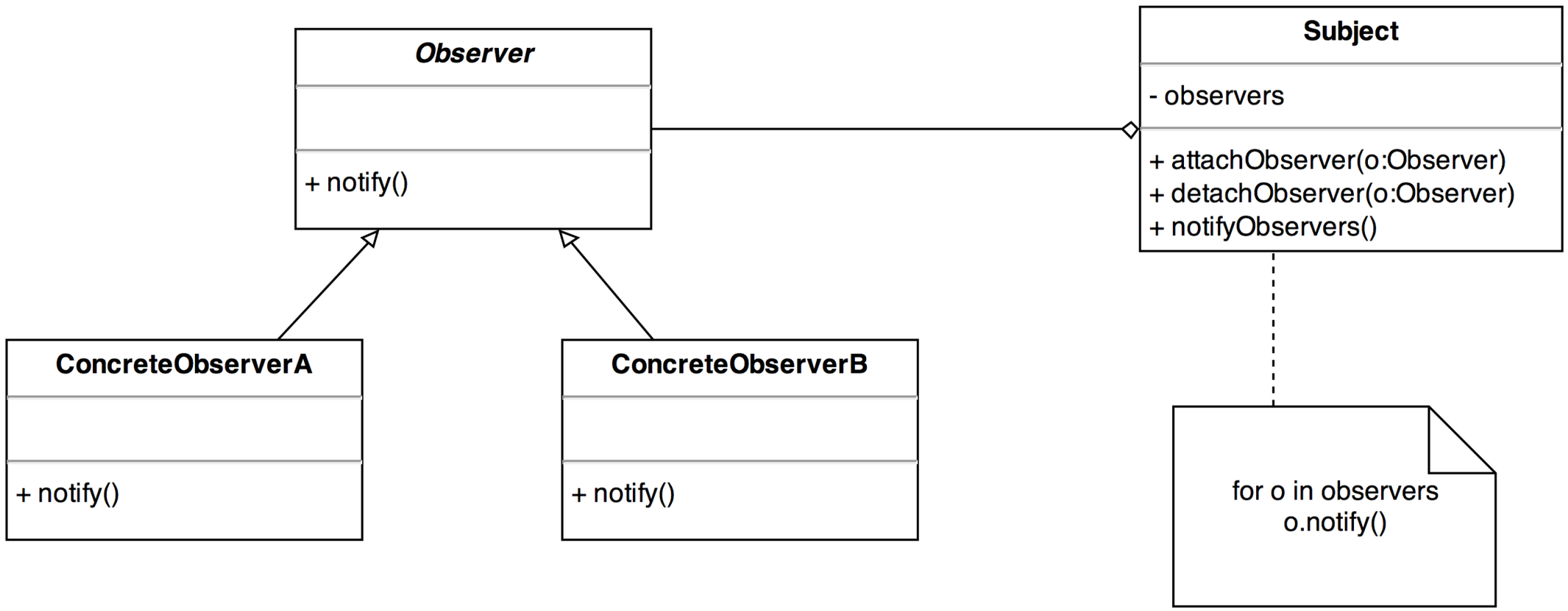
Composite

“The composite pattern is a structural design pattern. The composite pattern describes that a group of objects are to be treated in the same way as a single instance of an object. The intent of a composite is to "compose" objects into tree structures to represent part-whole hierarchies.”


Component
+ operation()
Leaf
+ operation()
Composite
+ components
+ operation()
for component in components
component.operation()

Observer

"The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems."


Observer
+ notify()
Subject
- observers
+ attachObserver(o:Observer)
+ detachObserver(o:Observer)
+ notifyObservers()
ConcreteObserverA
+ notify()
ConcreteObserverB
+ notify()
for o in observers
o.notify()

MVW frameworks

Framework vs Library

- A library provides a set of functions. We can reuse the functionality provided by the library in our application (jQuery, Underscore).

- A framework provides a base for given family of applications. We can build our application on top of the *solid* base provided by the framework (AngularJS, Backbone.js)

  - Reuse of functionality

  - Reuse of micro-architecture

How frameworks help?

- Helps with boilerplates
  - Base “classes” out of the box
  - Communication bus out of the box
- Easier reuse of custom components across applications
- Level of abstraction (we don’t need to think about details)
- Testability
- Implicit conventions (extremely useful in teams)

Current MVW landscape in the JavaScript world…


KEEP
CALM
AND
WELCOME TO
THE JUNGLE

A few MVW frameworks…

- **Backbone.js**
- **AngularJS**
- **Ember.js**
- **KnockoutJS**
- Dojo
- YUI
- Agility.js
- Knockback.js
- CanJS
- **Maria**
- **Polymer**

A few MVW frameworks…

- **React**
- cujoJS
- Montage
- Sammy.js
- Stapes
- Epitome
- soma.js
- DUEL
- Kendo UI
- PureMVC
- Olives


And wait for it...

A few MVW frameworks…

- PlastronJS
- Dijon
- rAppid.js
- Aria Templates
- SAPUI5
- Exoskeleton
- Atma.js
- Ractive.js
- ComponentJS
- Vue.js

Backbone.js

Backbone.js

"Backbone.js gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface."

Backbone.js

- Backbone.js is minimalistic *(6.5kb minified and gzipped)*
- Backbone.js depends on
  - Underscore
  - jQuery/Zepto

Backbone.js main components

- Events
- Views
- Models
- Collections
- Router

Backbone.Events

Backbone.Events

```
var obj = $.extend({},
Backbone.Events);

obj.on('event', function () {
  console.log(42);
});

obj.on('event', function () {
  console.log(1.618);
});

obj.trigger('event');
```

Backbone.View

Backbone.View

```
/* global Backbone, $ */

var GitHubApp = GitHubApp || {};

GitHubApp.Views = GitHubApp.Views || {};

GitHubApp.Views.Home = Backbone.View.extend({
  events: {
    'click #add-btn'    : 'addUser',
    'click .delete-btn': 'removeUser'
  },

  initialize: function () {
    'use strict';
    this.model.on('change', this.render, this);
  },

  addUser: function () {},

  removeUser: function (e) {},

  render: function () {}
});
```

Backbone.View

```
/* global Backbone, $ */

var GitHubApp = GitHubApp || {};

GitHubApp.Views = GitHubApp.Views || {};

GitHubApp.Views.Home = Backbone.View.extend({
  events: {
    'click #add-btn'    : 'addUser',
    'click .delete-btn': 'removeUser'
  },

  initialize: function () {
    'use strict';
    this.model.on('change', this.render, this);
  },

  addUser: function () {},

  removeUser: function (e) {},

  render: function () {}
});
```

Backbone.View

```
var View = Backbone.View.extend({
  el: '#parent',
  template: _.template('<%= name %>'),
  render: function () {
    this.$el.html(this.template(this.model));
    return this;
  }
});

var v = new View({
  model: {
    name: 'foo'
  }
});
v.render();
```

Backbone.Model

Backbone.Model

```
var Developer = Backbone.Model.extend({
  defaults: {
    name      : 'foo',
    languages: ['JavaScript', 'Ruby', 'Perl'],
    age       : 42
  },
  initialize: function () {
    console.log('Do some initialisation stuff');
  },
  incrementAge: function () {
    this.set('age', this.get('age') + 1);
  }
});
```

Backbone.Model

```
var dev = new Developer({
  name: 'bar'
});

dev.on('change', function (e) {
  console.log(Object.keys(e.changed)
    .toString(), 'changed');
});

dev.incrementAge();
```

Backbone.Model

```
var Developer = Backbone.Model.extend({
  url: function () {
    return 'https://api.github.com/users/' +
           this.get('name');
  }
});

var dev = new Developer({
  name: 'mgechev'
});

dev.fetch();
//GET https://api.github.com/users/mgechev
```

Backbone.Collection

```
/* global Backbone */

var GitHubApp = GitHubApp || {};

GitHubApp.Models = GitHubApp.Models || {};

GitHubApp.Models.UserCollection = Backbone.Collection.extend({
  model: GitHubApp.Models.User
});
```

```
var collection = new GitHubApp.Models.UserCollection();

collection.on('add', function () {
  console.log('User added');
});

collection.on('remove', function () {
  console.log('User removed');
});

var user = new User();

collection.add(user);
collection.remove(user);
```

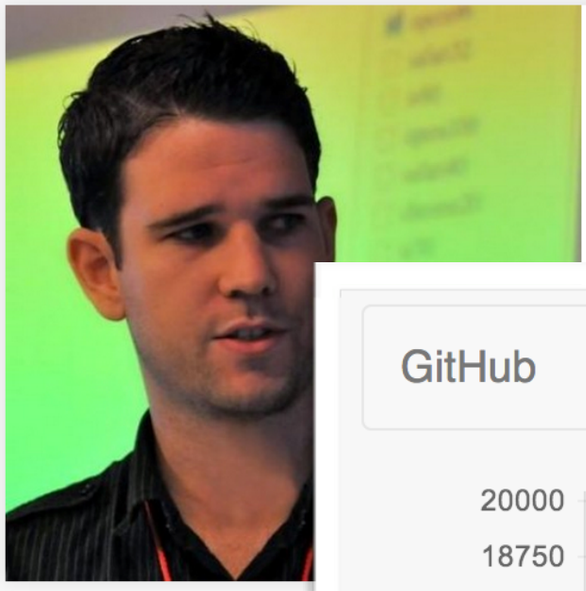
Backbone.Router

Backbone.Router

```
/* global Backbone, $ */

var GitHubApp = GitHubApp || {};

var GitHubAppRouter = Backbone.Router.extend({
  routes: {
    ''                  : 'home',
    'user/:username': 'user',
    'statistics'    : 'stats'
  },
  initialize: function () {},
  home: function () {},
  user: function (login) {},
  stats: function () {}
});

GitHubApp.router = new GitHubAppRouter();

Backbone.history.start();
```

Our Application

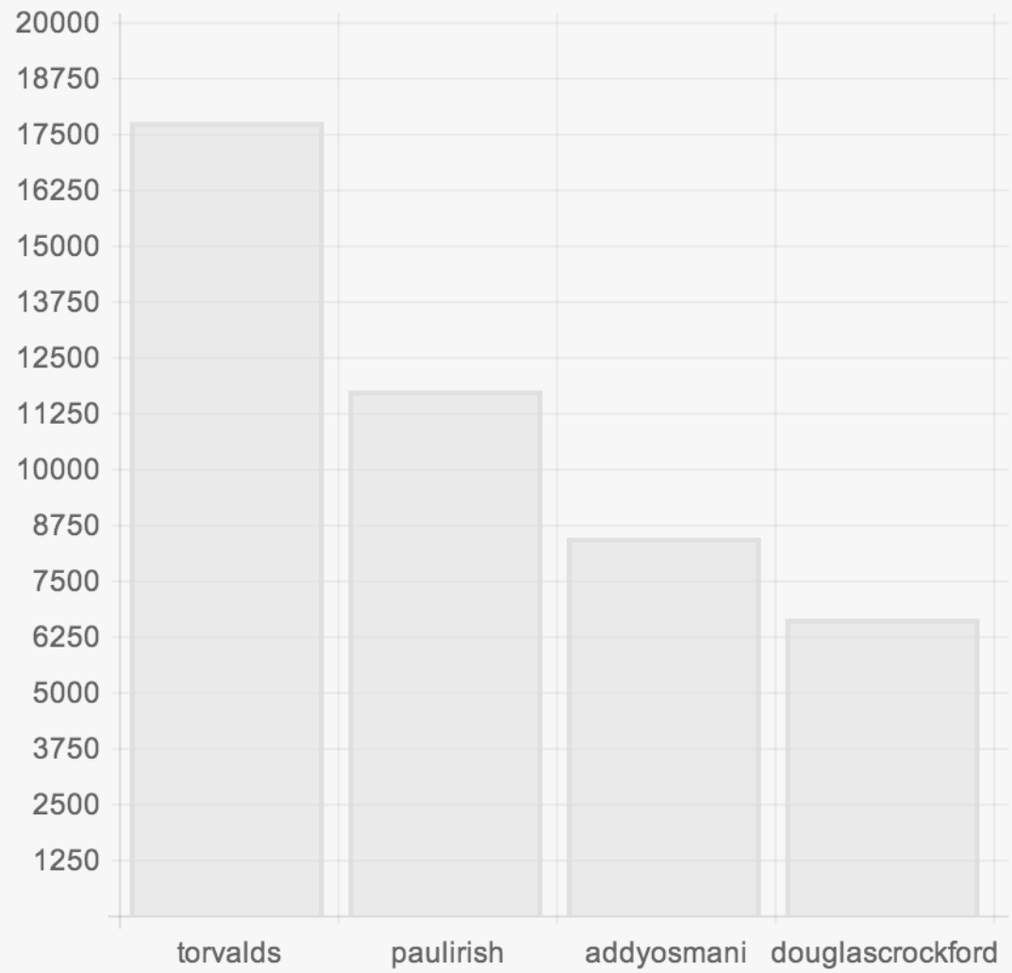
GitHub User stats



User paulirish

Followers : **11774**
Following : **185**

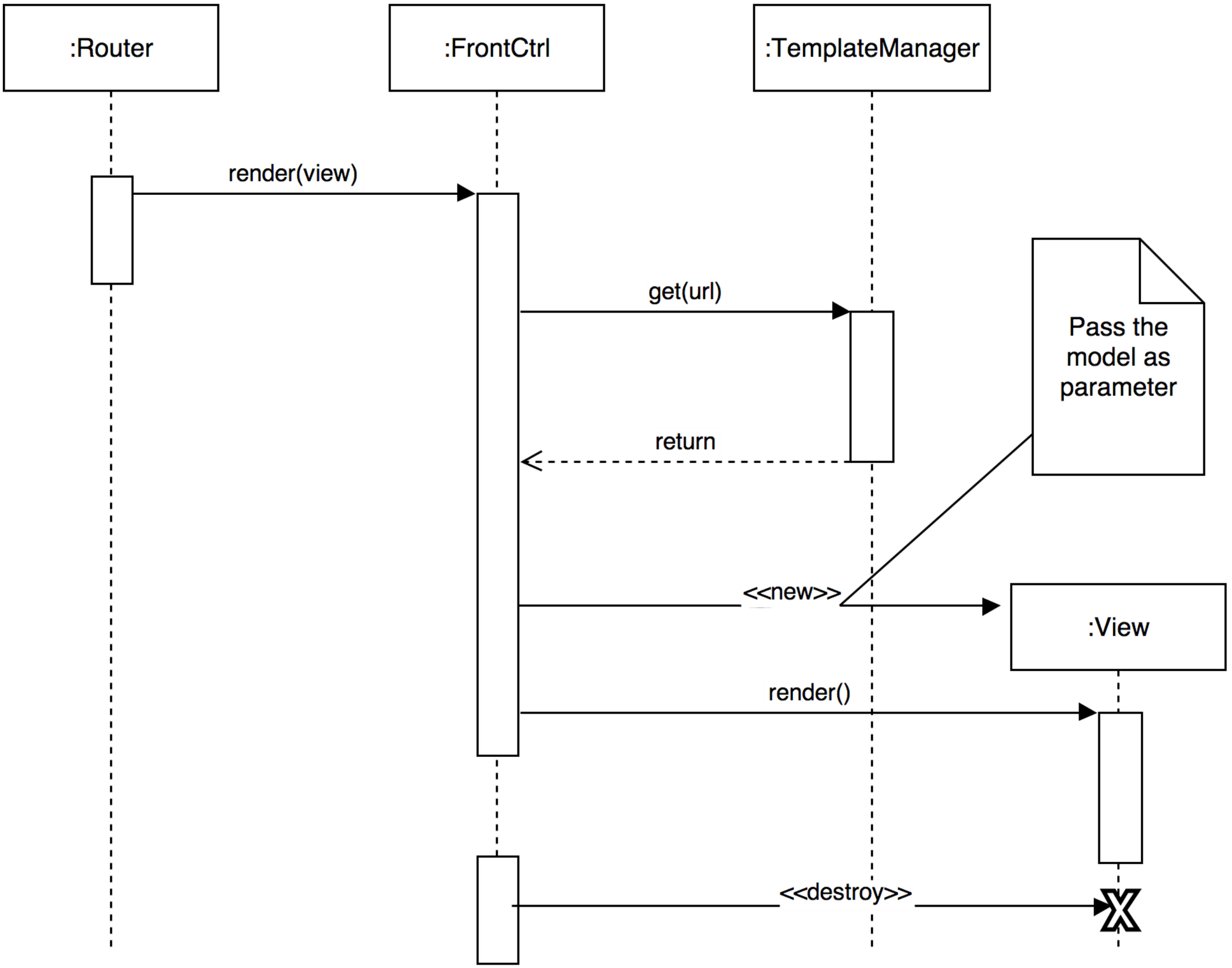
GitHub User stats



User stats

Add

[X]


:Router
:FrontCtrl
:TemplateManager
render(view)
get(url)
return
Pass the model as parameter
<<new>>
:View
render()
<<destroy>>

Thank you!