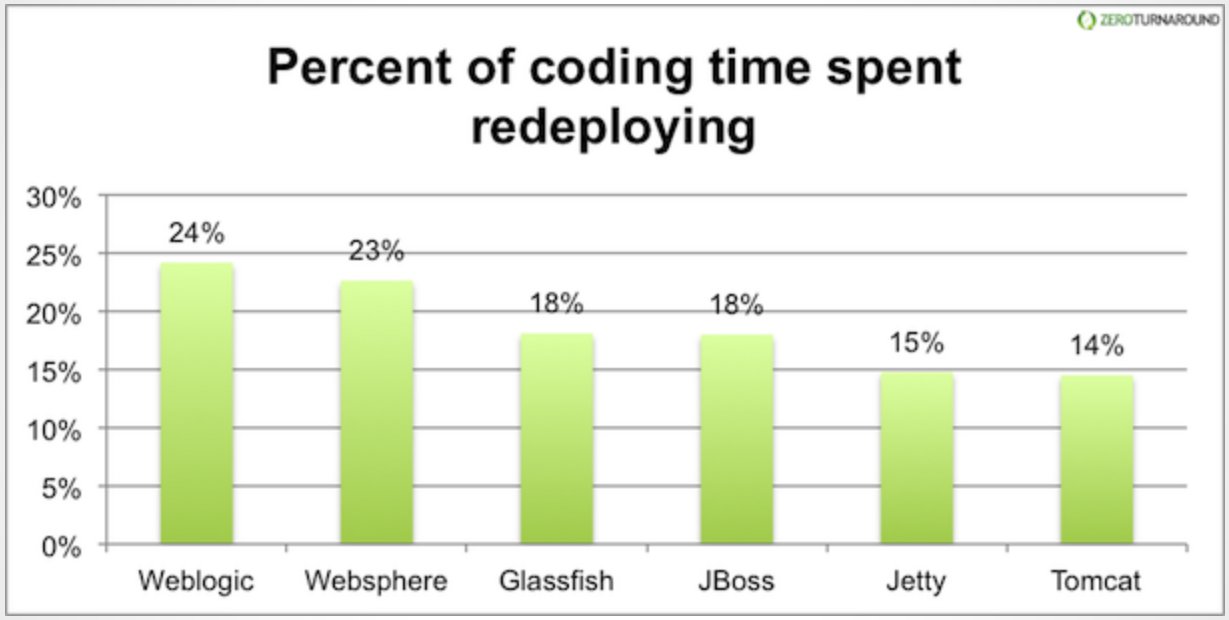
at

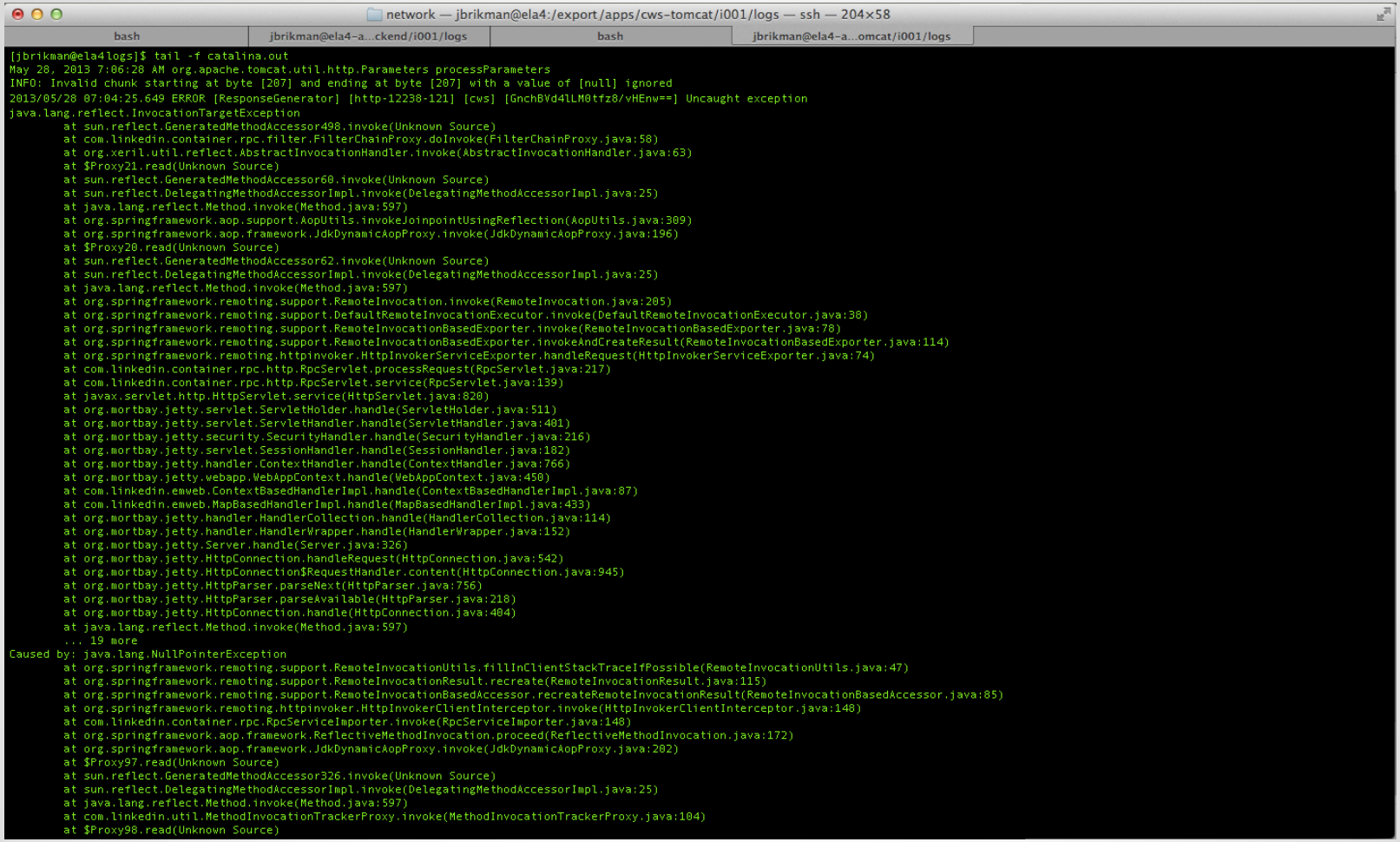
Productivity and Performance at Scale

You are a developer using Java to build web services.

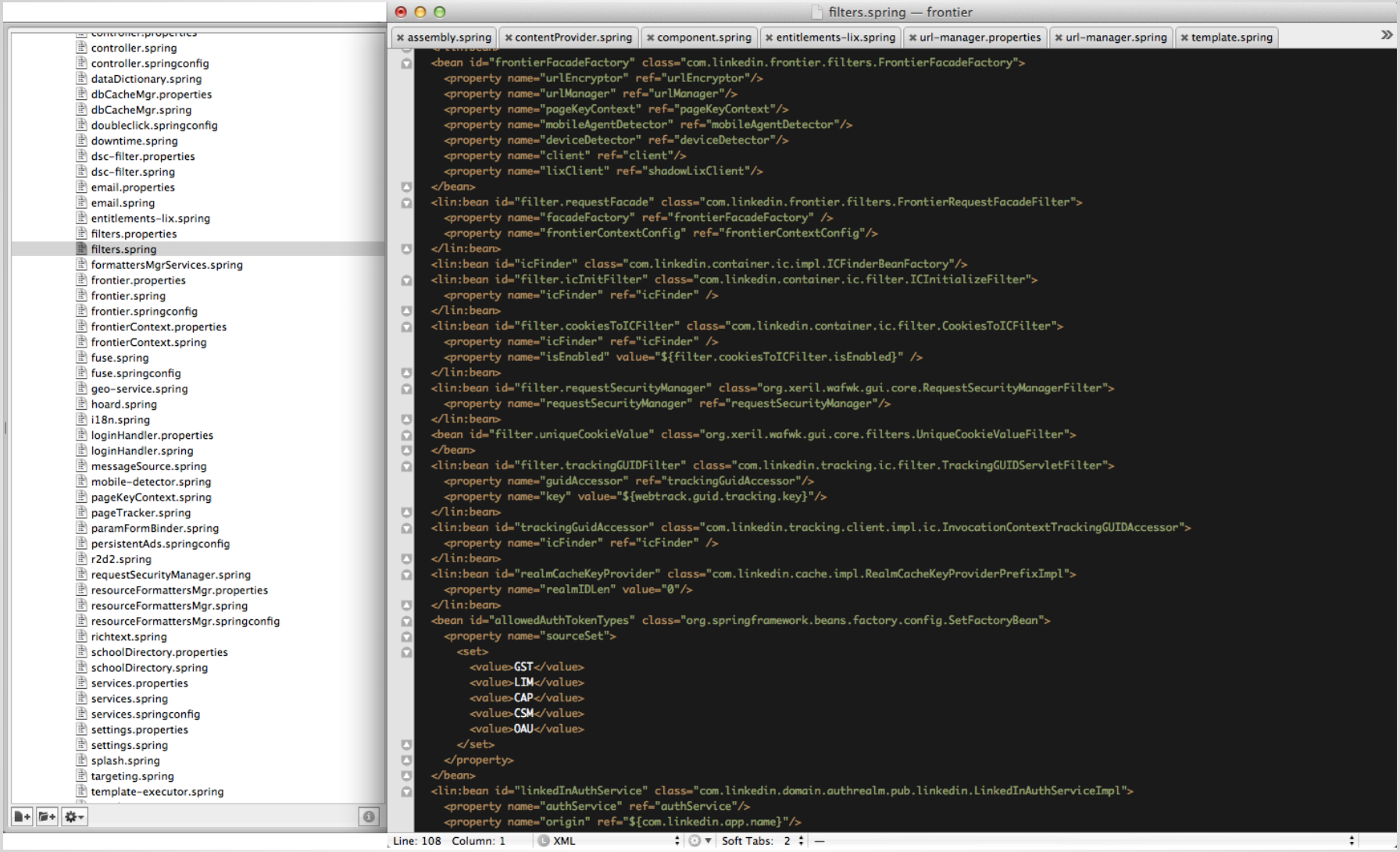
This is your life:


ZEROTURNAROUND
Percent of coding time spent redeploying
30%
25%
20%
15%
10%
5%
0%
24%
23%
18%
18%
15%
14%
Weblogic
Websphere
Glassfish
JBoss
Jetty
Tomcat

Waiting. [1]



tail -f logs/catalina.out


filters.spring — frontier
assembly.spring
contentProvider.spring
component.spring
entitlements-lix.spring
url-manager.properties
url-manager.spring
template.spring
controller.spring
controller.springconfig
dataDictionary.spring
dbCacheMgr.properties
dbCacheMgr.spring
doubleclick.springconfig
downtime.spring
dsc-filter.properties
dsc-filter.spring
email.properties
email.spring
entitlements-lix.spring
filters.properties
filters.spring
formattersMgrServices.spring
frontier.properties
frontier.spring
frontier.springconfig
frontierContext.properties
frontierContext.spring
fuse.spring
fuse.springconfig
geo-service.spring
hoard.spring
i18n.spring
loginHandler.properties
loginHandler.spring
messageSource.spring
mobile-detector.spring
pageKeyContext.spring
pageTracker.spring
paramFormBinder.spring
persistentAds.springconfig
r2d2.spring
requestSecurityManager.spring
resourceFormattersMgr.properties
resourceFormattersMgr.spring
resourceFormattersMgr.springconfig
richtext.spring
schoolDirectory.properties
schoolDirectory.spring
services.properties
services.spring
services.springconfig
settings.properties
settings.spring
splash.spring
targeting.spring
template-executor.spring
<bean id="frontierFacadeFactory" class="com.linkedin.frontier.filters.FrontierFacadeFactory">
<property name="urlEncryptor" ref="urlEncryptor"/>
<property name="urlManager" ref="urlManager"/>
<property name="pageKeyContext" ref="pageKeyContext"/>
<property name="mobileAgentDetector" ref="mobileAgentDetector"/>
<property name="deviceDetector" ref="deviceDetector"/>
<property name="client" ref="client"/>
<property name="lixClient" ref="shadowLixClient"/>
</bean>
<lin:bean id="filter.requestFacade" class="com.linkedin.frontier.filters.FrontierRequestFacadeFilter">
<property name="facadeFactory" ref="frontierFacadeFactory" />
<property name="frontierContextConfig" ref="frontierContextConfig"/>
</lin:bean>
<lin:bean id="icFinder" class="com.linkedin.container.ic.impl.ICFinderBeanFactory"/>
<lin:bean id="filter.icInitFilter" class="com.linkedin.container.ic.filter.ICInitializeFilter">
<property name="icFinder" ref="icFinder" />
</lin:bean>
<lin:bean id="filter.cookiesToICFilter" class="com.linkedin.container.ic.filter.CookiesToICFilter">
<property name="icFinder" ref="icFinder" />
<property name="isEnabled" value="${filter.cookiesToICFilter.isEnabled}" />
</lin:bean>
<lin:bean id="filter.requestSecurityManager" class="org.xeril.wafwk.gui.core.RequestSecurityManagerFilter">
<property name="requestSecurityManager" ref="requestSecurityManager"/>
</lin:bean>
<bean id="filter.uniqueCookieValue" class="org.xeril.wafwk.gui.core.filters.UniqueCookieValueFilter">
</bean>
<lin:bean id="filter.trackingGUIDFilter" class="com.linkedin.tracking.ic.filter.TrackingGUIDServletFilter">
<property name="guidAccessor" ref="trackingGuidAccessor"/>
<property name="key" value="${webtrack.guid.tracking.key}"/>
</lin:bean>
<lin:bean id="trackingGuidAccessor" class="com.linkedin.tracking.client.impl.ic.InvocationContextTrackingGUIDAccessor">
<property name="icFinder" ref="icFinder" />
</lin:bean>
<lin:bean id="realmCacheKeyProvider" class="com.linkedin.cache.impl.RealmCacheKeyProviderPrefixImpl">
<property name="realmIDLen" value="0"/>
</lin:bean>
<bean id="allowedAuthTokenTypes" class="org.springframework.beans.factory.config.SetFactoryBean">
<property name="sourceSet">
<set>
<value>GST</value>
<value>LIM</value>
<value>CAP</value>
<value>CSM</value>
<value>OAU</value>
</set>
</property>
</bean>
<lin:bean id="linkedInAuthService" class="com.linkedin.domain.authrealm.pub.linkedin.LinkedInAuthServiceImpl">
<property name="authService" ref="authService"/>
<property name="origin" ref="${com.linkedin.app.name}"/>
Line: 108 Column: 1
XML
Soft Tabs: 2

XML soup

org.springframework.aop.framework

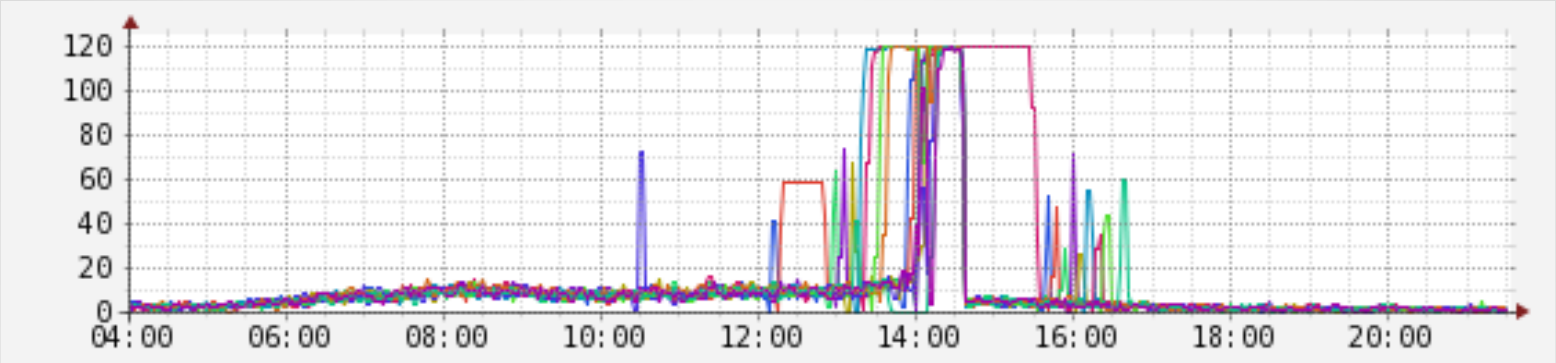
Class AbstractSingletonProxyFactoryBean

java.lang.Object
  └ org.springframework.aop.framework.ProxyConfig
    └ **org.springframework.aop.framework.AbstractSingletonProxyFactoryBean**

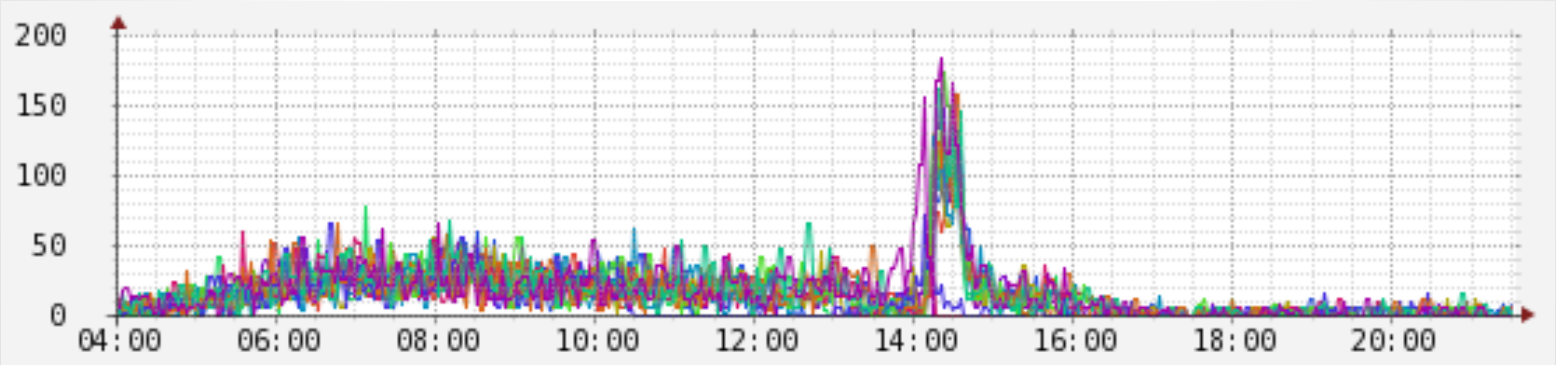
```
public abstract class AbstractSingletonProxyFactoryBean
extends ProxyConfig
implements FactoryBean, BeanClassLoaderAware, InitializingBean
```

Convenient proxy factory bean superclass for proxy factory beans that create only singletons.

"Convenient proxy factory bean superclass for proxy factory beans that create only singletons." [2]

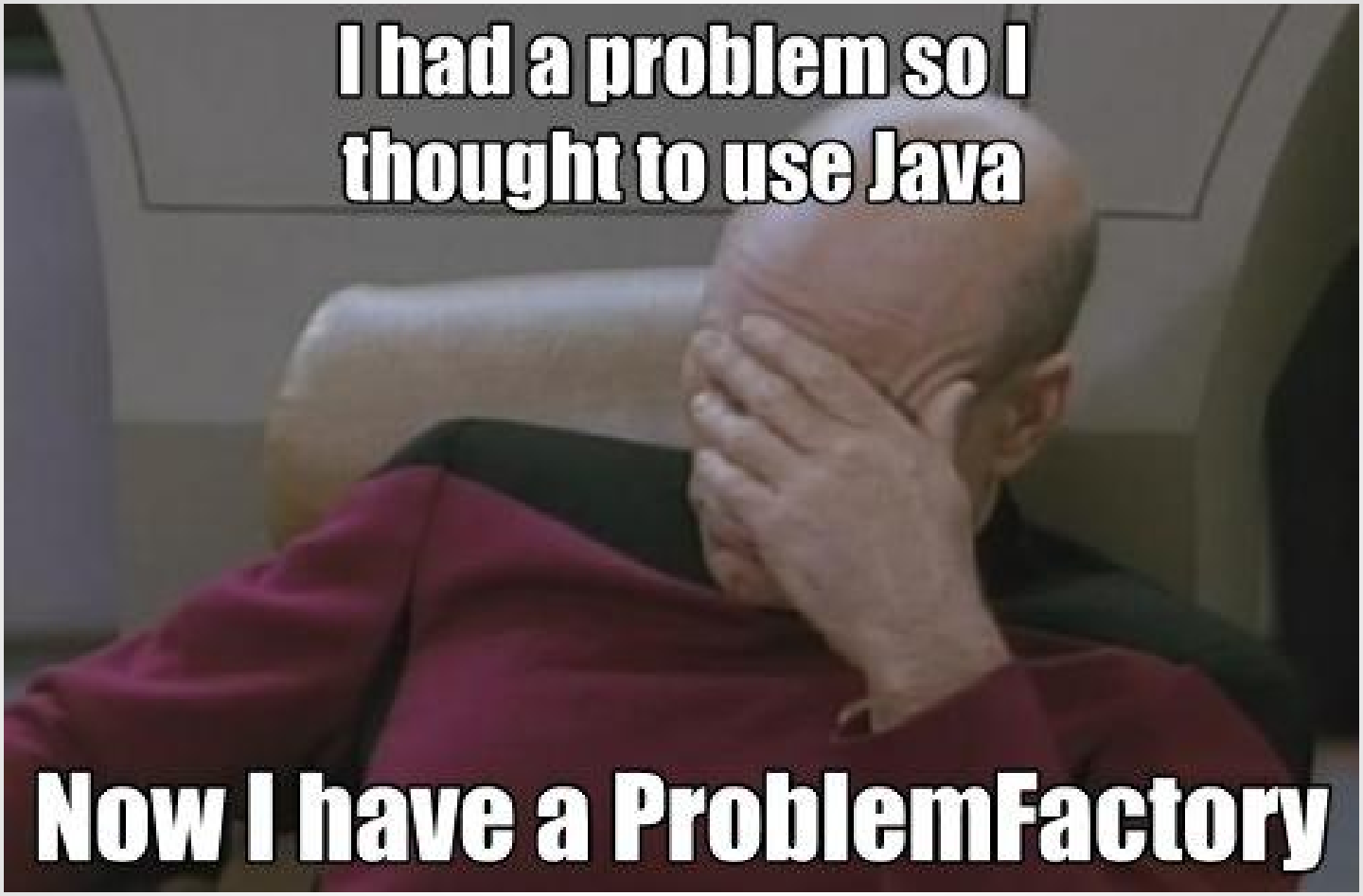


Thread pool usage



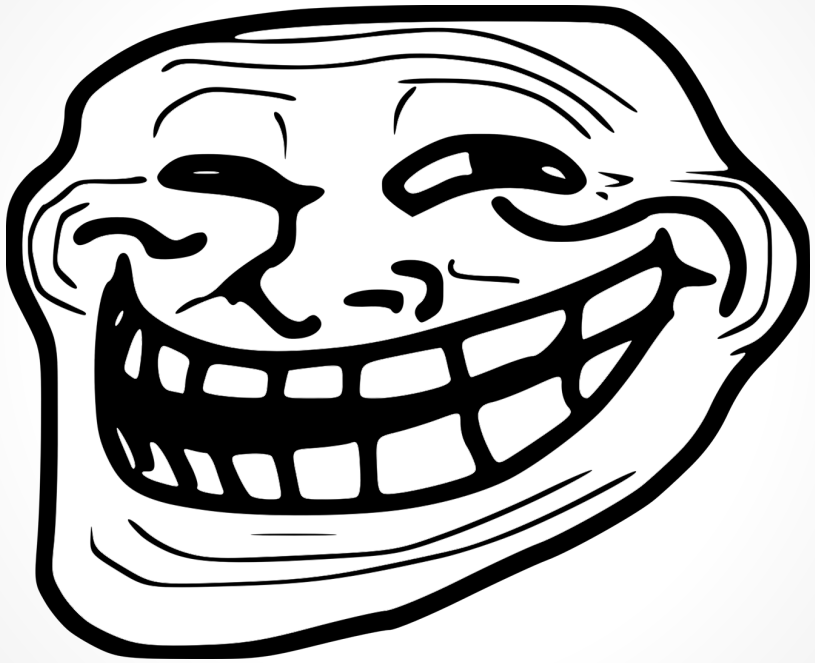
Latency

Thread pool hell


I had a problem so I
thought to use Java
Now I have a ProblemFactory

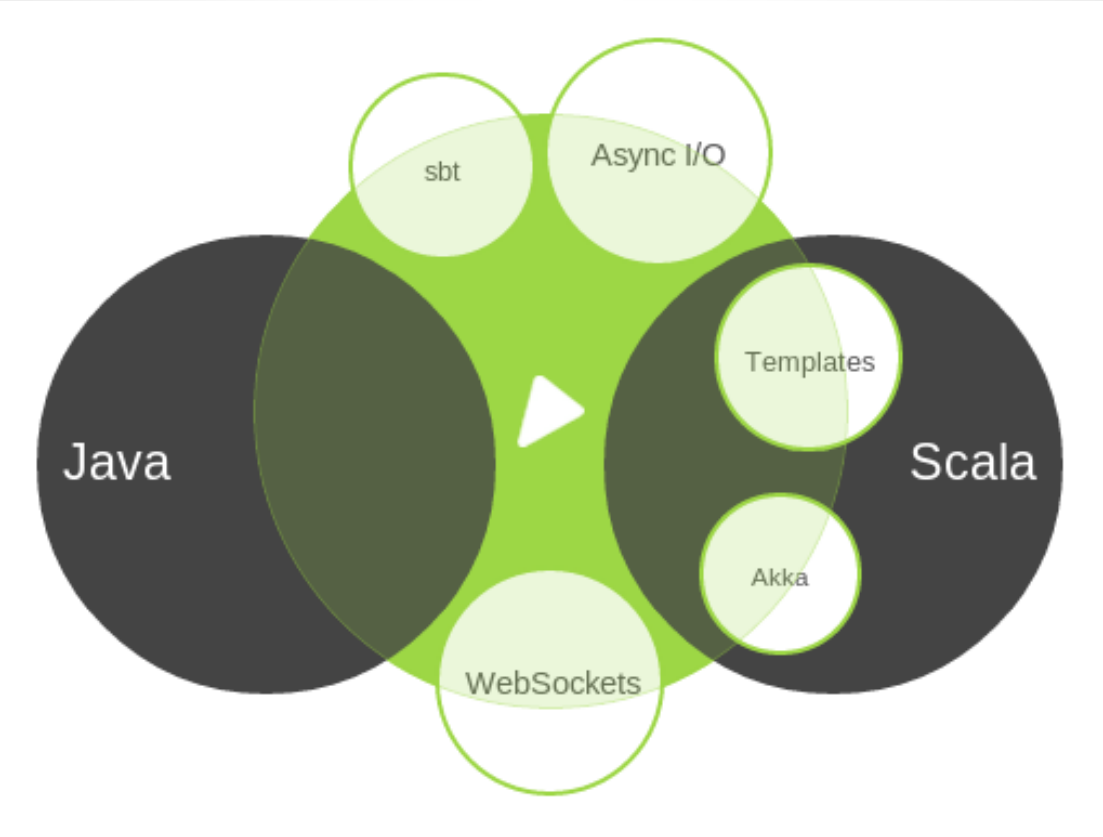
There is a better way.

Ruby on Rails!



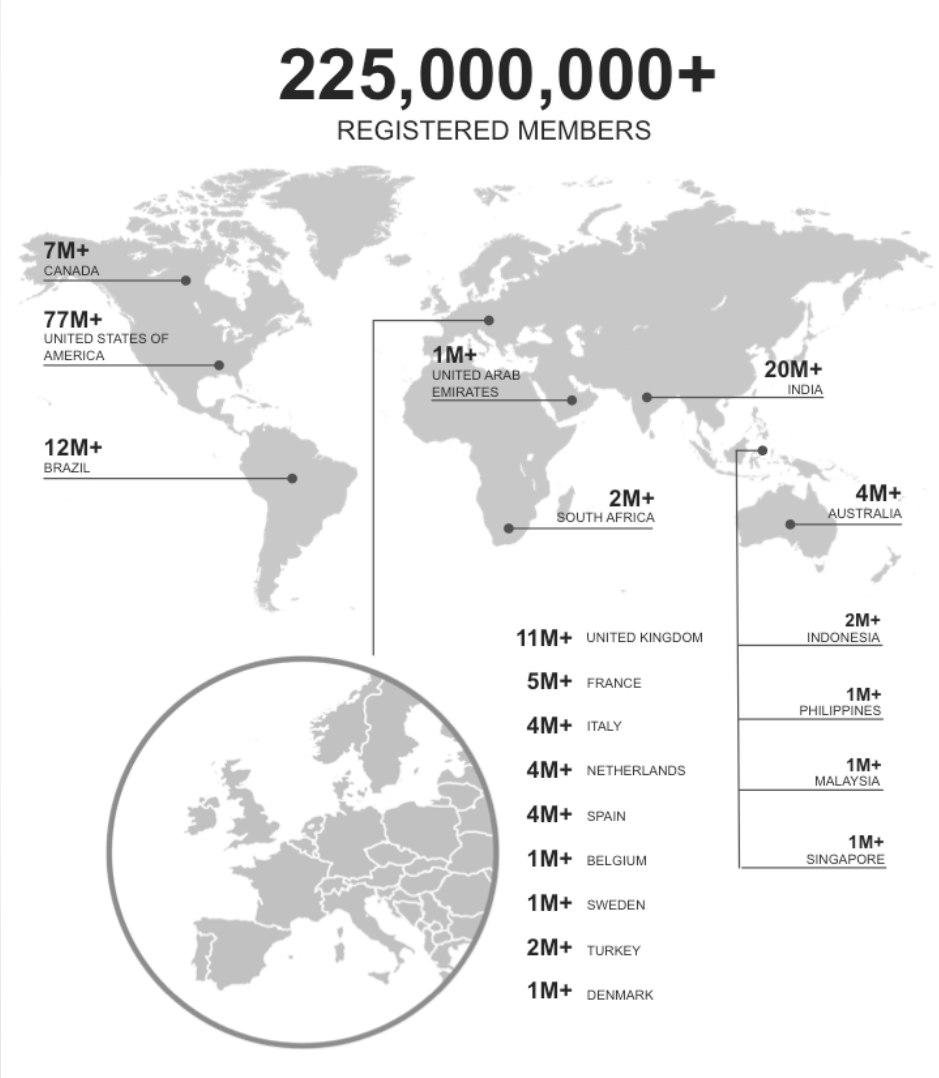
Nah, just kidding. We have real work to do.

play



A modern web framework for Java and Scala

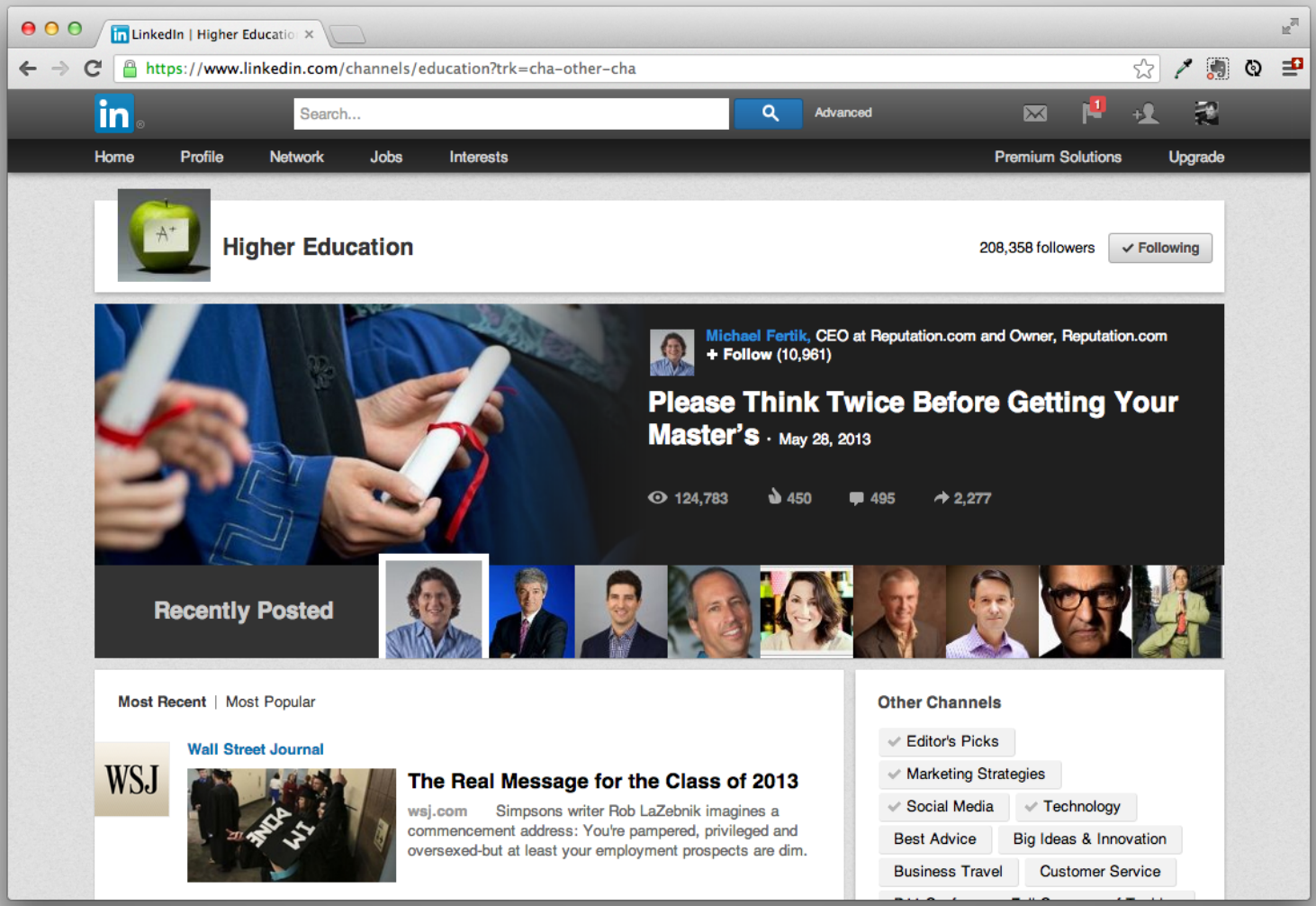
LinkedIn™ The world's largest professional network



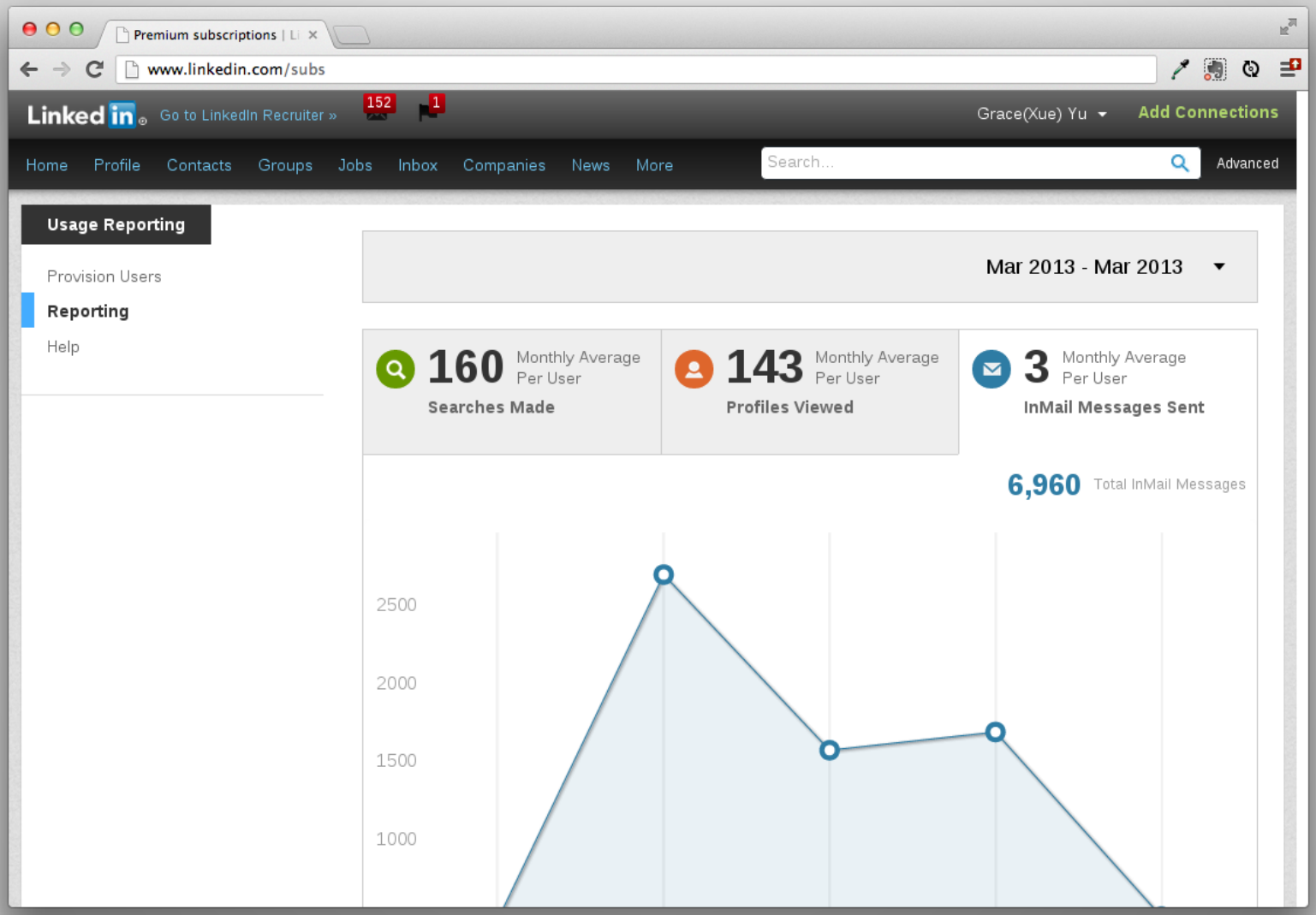
at

We've been using Play in production for more than 6 months [3]

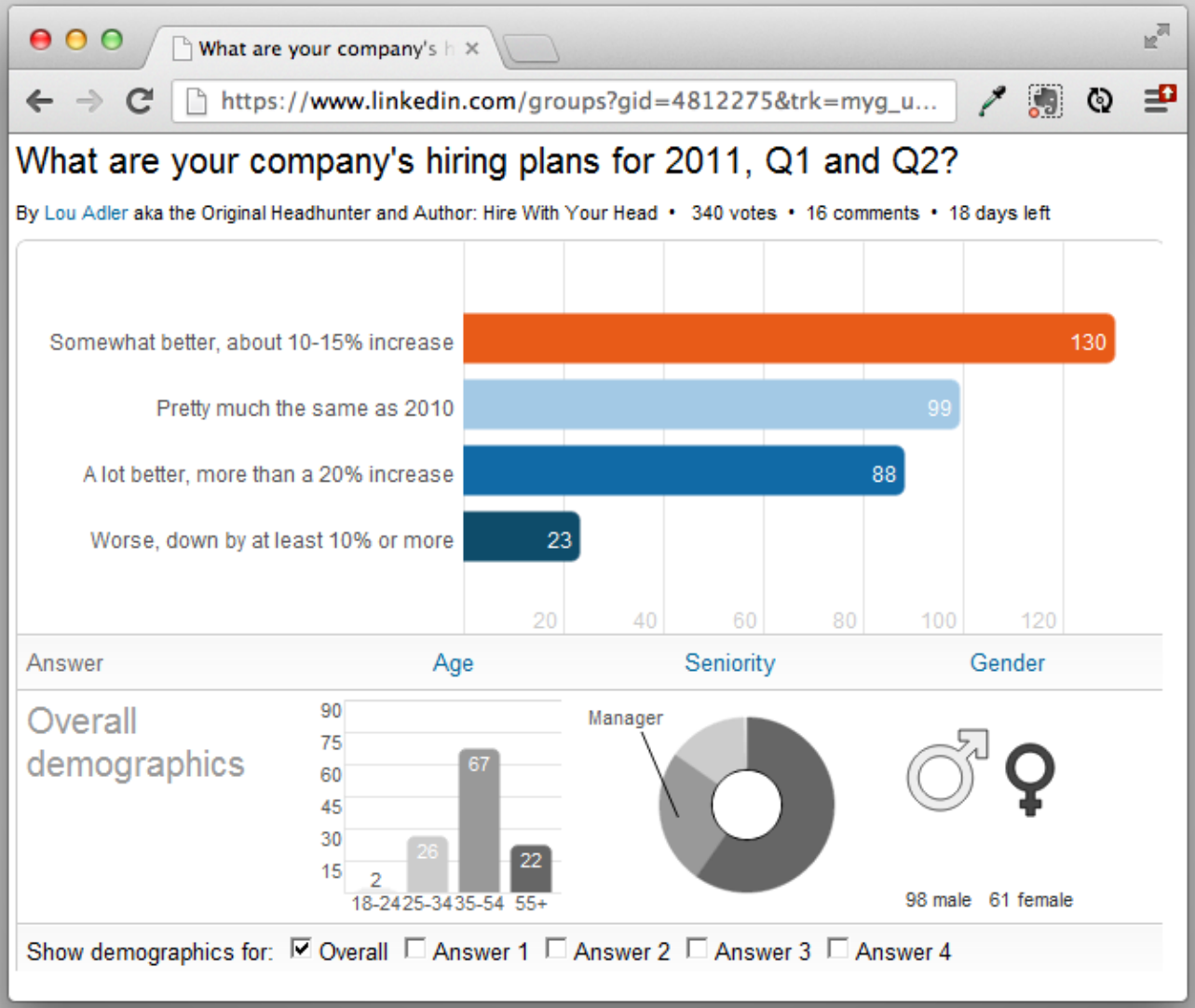
A few apps built on Play



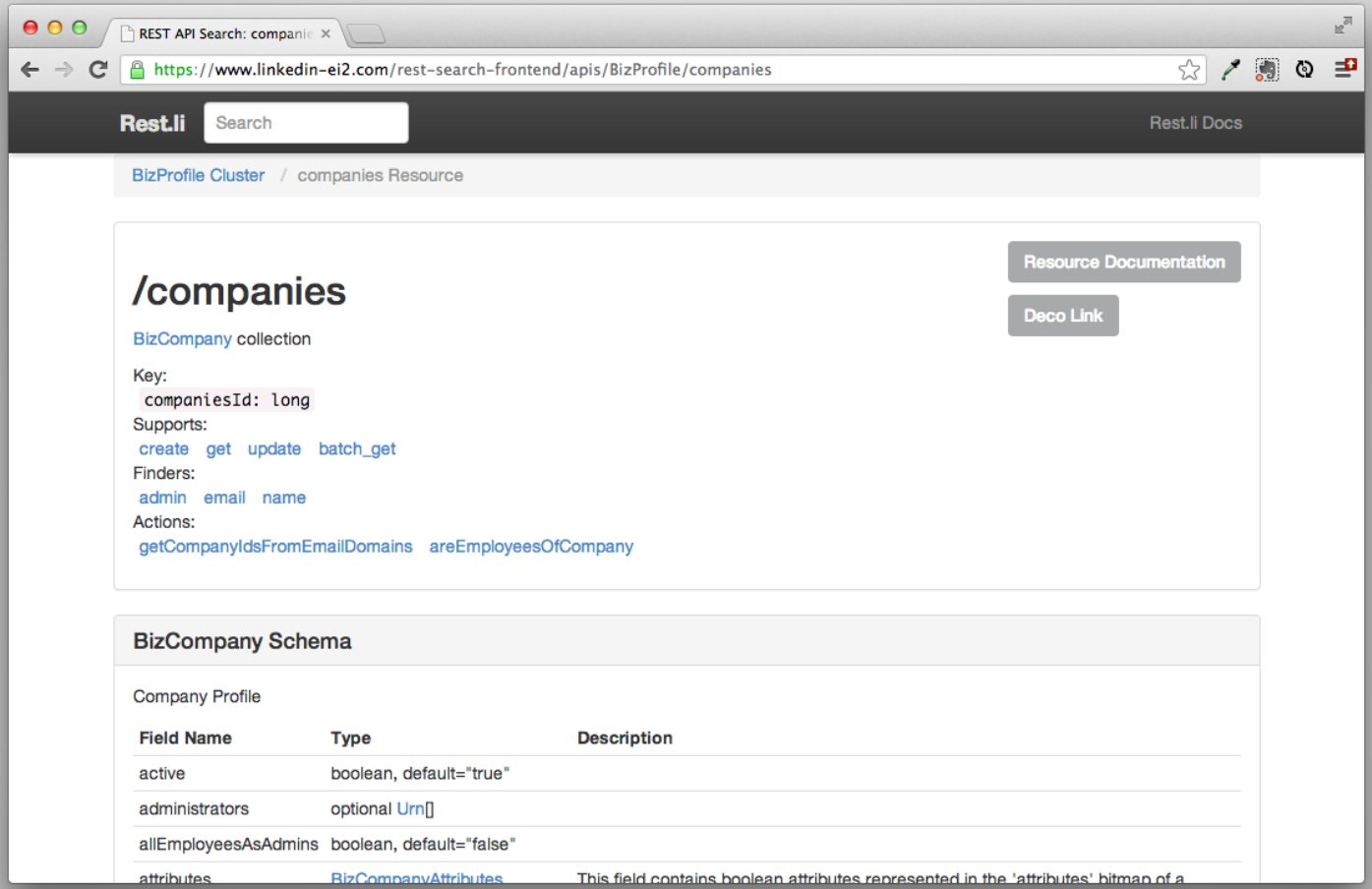
Channels (frontend)



Premium Subscriptions (frontend)

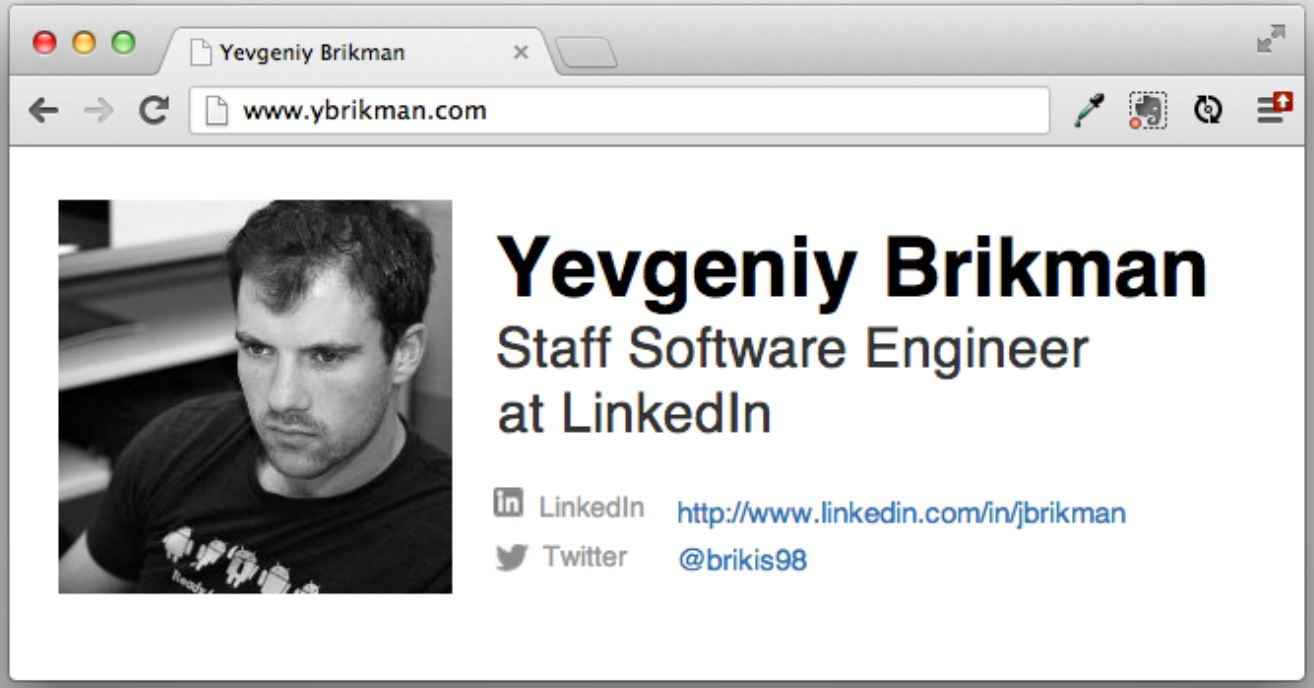


Polls (frontend + backend)



REST search (internal tool)

About me



Leading the Play project as part of LinkedIn's Service Infrastructure Team.
Also: hackdays, engineering blog, incubator, open source.

This talk is the story of building web services at massive scale with Java and Scala...

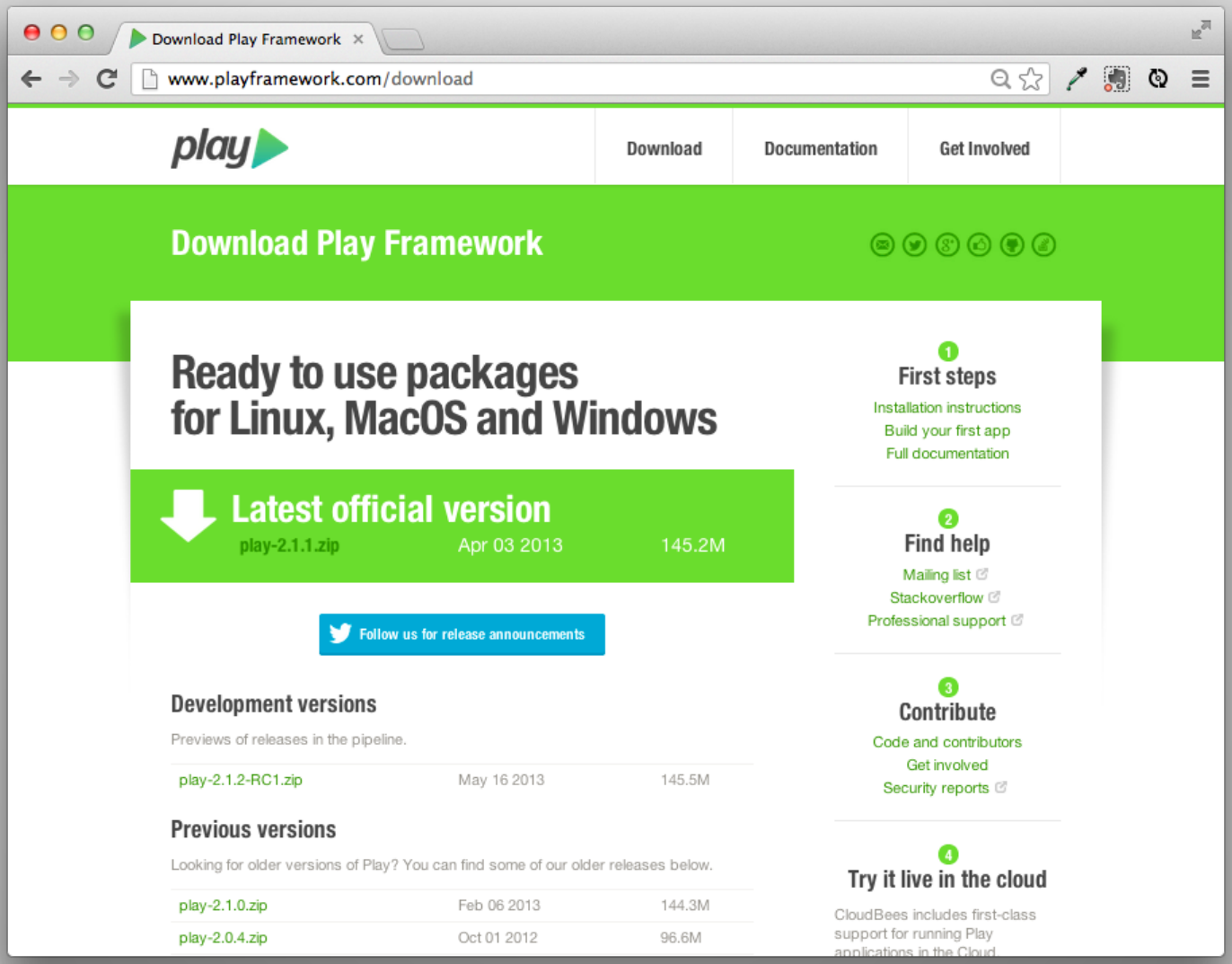
... while maintaining performance, reliability, and developer productivity.

Outline

1. Getting started with Play
2. Make a change and reload
3. Error handling
4. Threaded vs. evented
5. Non-blocking I/O
6. Scala
7. Performance
8. Community

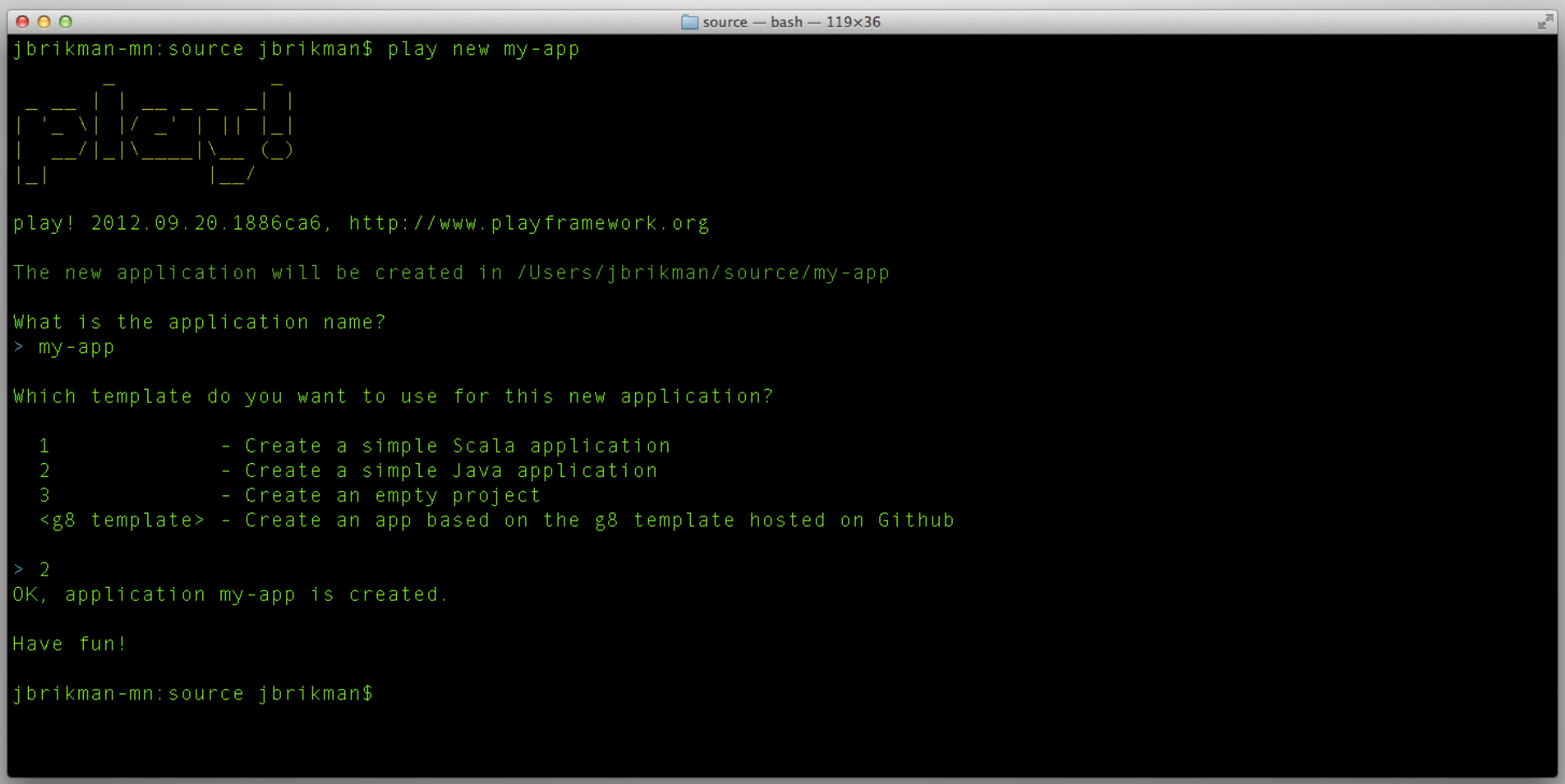
Outline

1. **Getting started with Play**
2. Make a change and reload
3. Error handling
4. Threaded vs. evented
5. Non-blocking I/O
6. Scala
7. Performance
8. Community

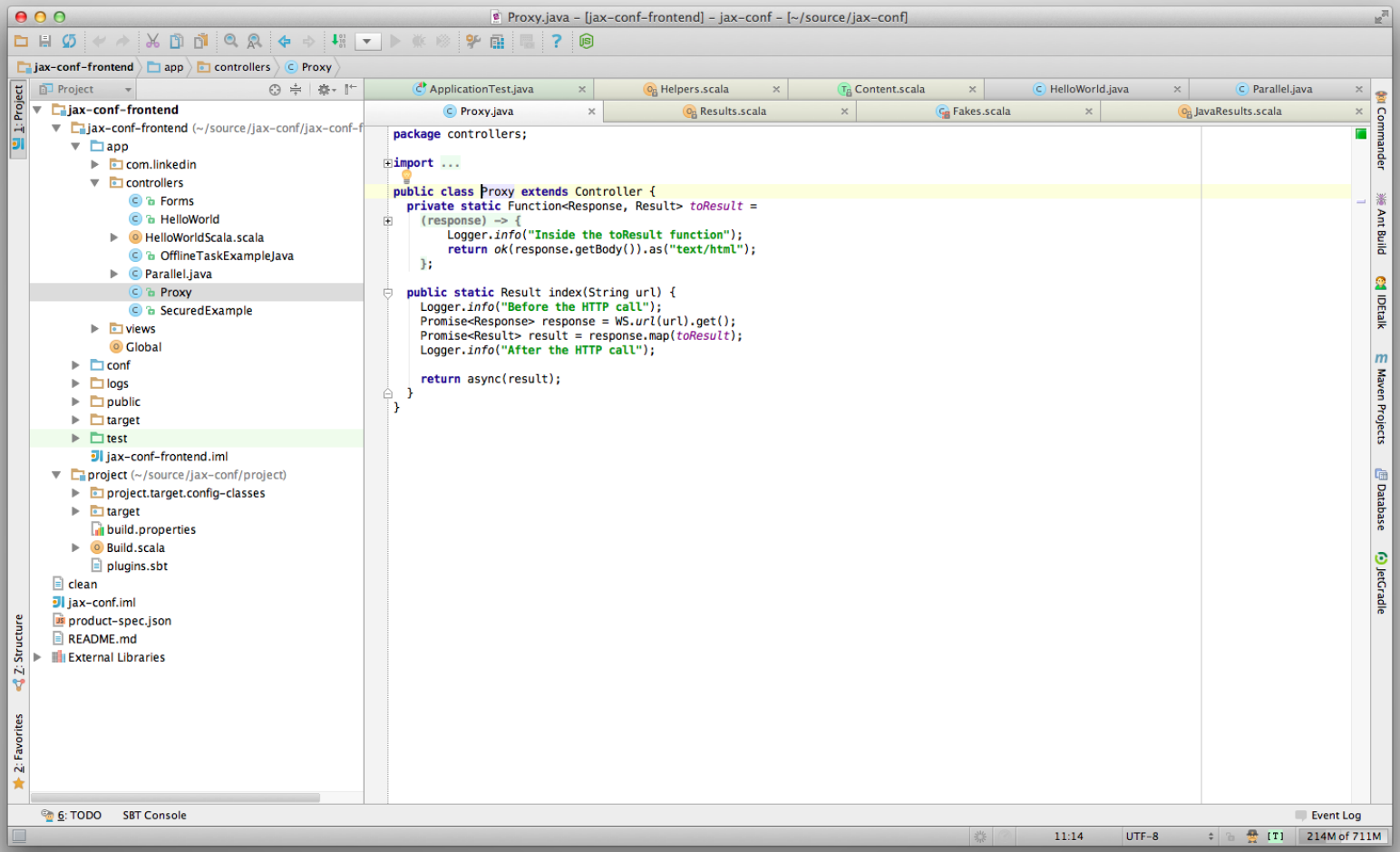


Download and install Play from

http://www.playframework.com

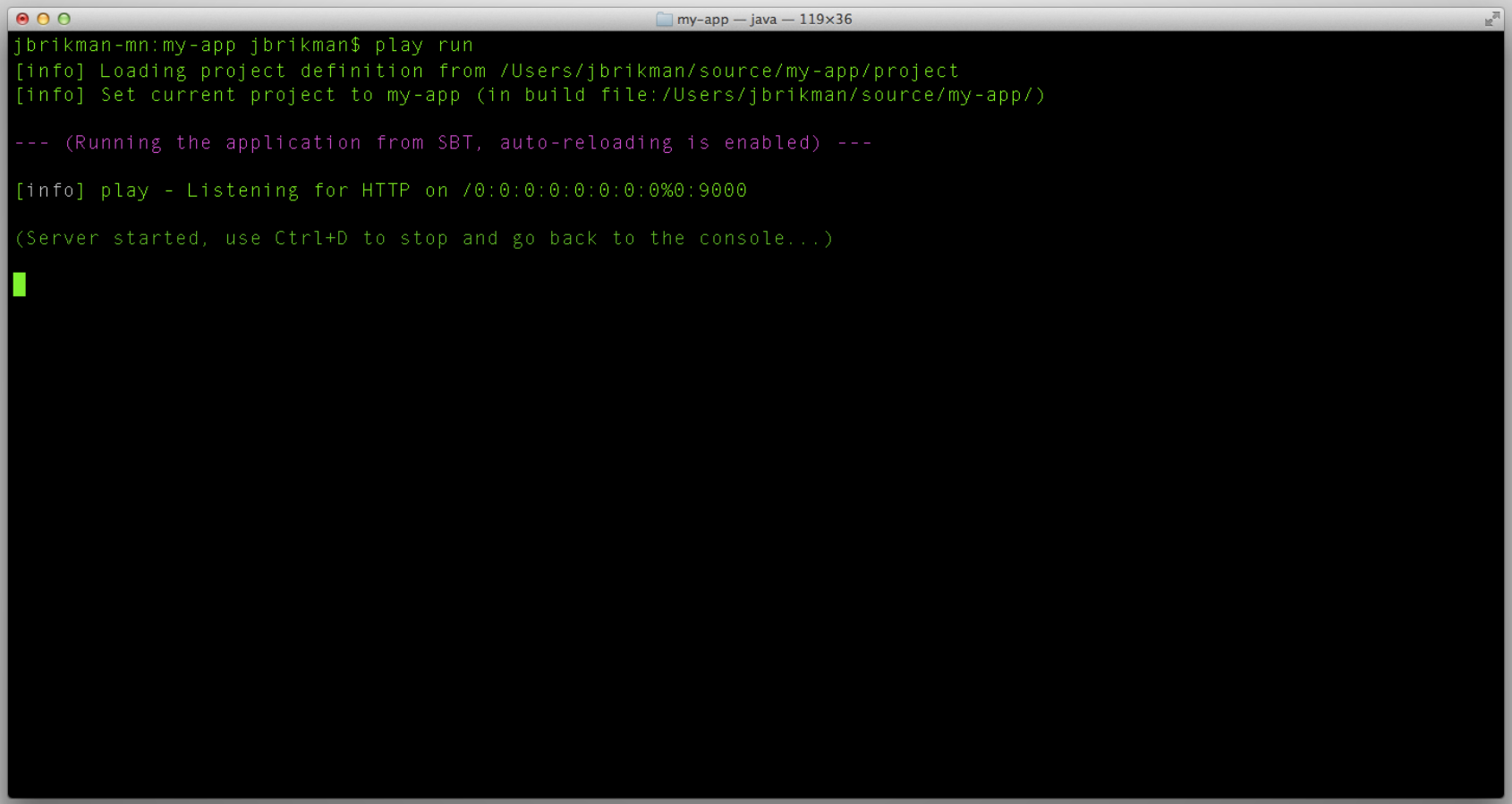


> play new my-app

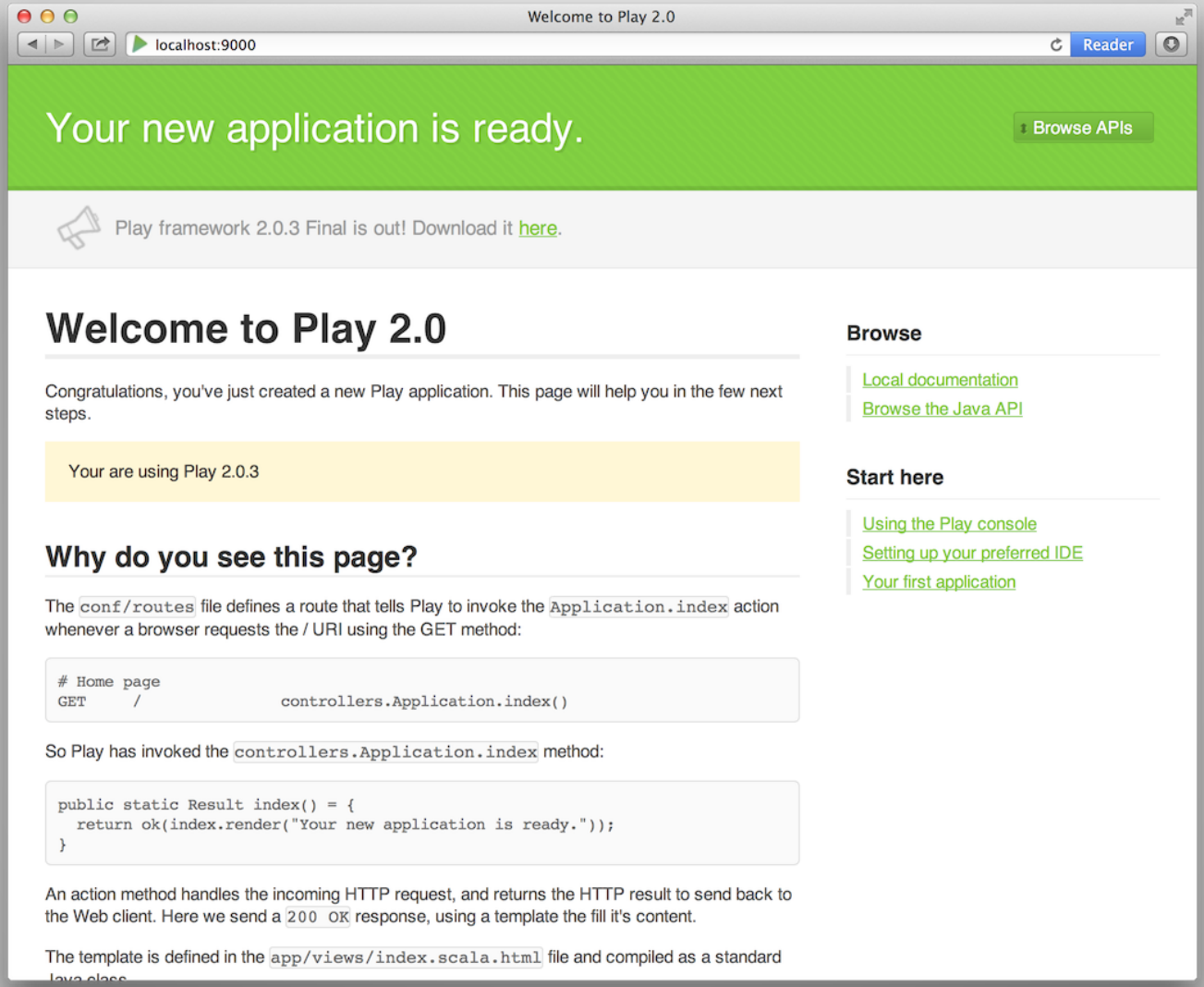


> play idea

> play eclipse


my-app — java — 119×36
jbrikman-mn:my-app jbrikman$ play run
[info] Loading project definition from /Users/jbrikman/source/my-app/project
[info] Set current project to my-app (in build file:/Users/jbrikman/source/my-app/)
--- (Running the application from SBT, auto-reloading is enabled) ---
[info] play - Listening for HTTP on /0:0:0:0:0:0:0:0%0:9000
(Server started, use Ctrl+D to stop and go back to the console...)

> play run



http://localhost:9000

```
app                         → Application sources
 └ assets                   → Compiled asset sources
 └ controllers              → Application controllers
 └ models                   → Application business layer
 └ views                    → Templates
conf                        → Configurations files
 └ application.conf         → Main configuration file
 └ routes                   → Routes definition
public                      → Public assets
 └ stylesheets              → CSS files
 └ javascripts              → Javascript files
 └ images                   → Image files
project                     → sbt configuration files
 └ Build.scala              → Application build script
 └ plugins.sbt              → sbt plugins
lib                         → Unmanaged libraries
dependencies
logs                        → Standard logs folder
target                      → Generated stuff
test                        → Unit or functional tests
```

Application layout

Outline

1. Getting started with Play
2. **Make a change and reload**
3. Error handling
4. Threaded vs. evented
5. Non-blocking I/O
6. Scala
7. Performance
8. Community

```java
public class HelloWorld extends Controller {

  public static Result index() {
    return ok("Hello World");
  }
}
```

app/controllers/HelloWorld.java

Create a new controller and action

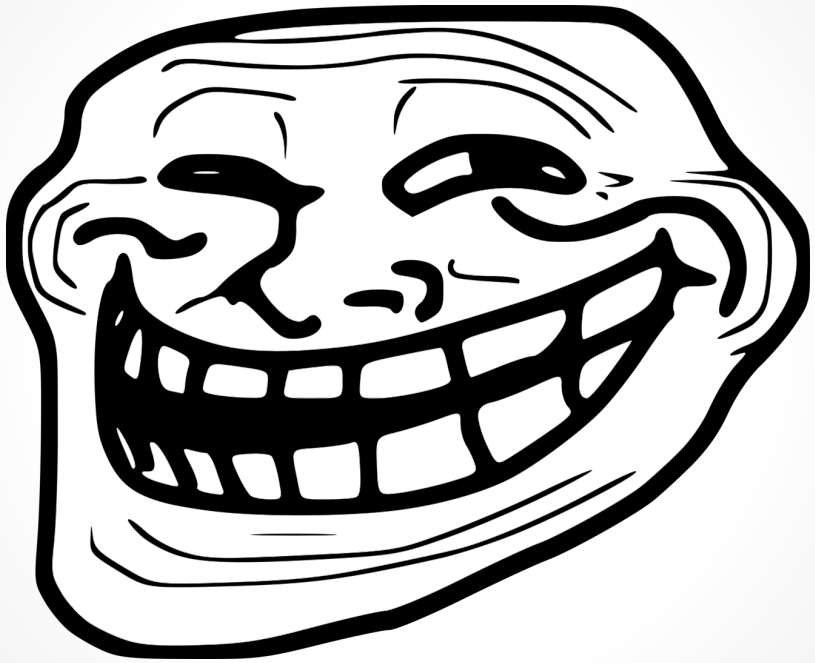
Don't worry about the use of `static`. Yes, Play supports IOC. Using `static` (and other shortcuts) lets me keep the examples simple.

```
GET     /hello    controllers.HelloWorld.index()
```

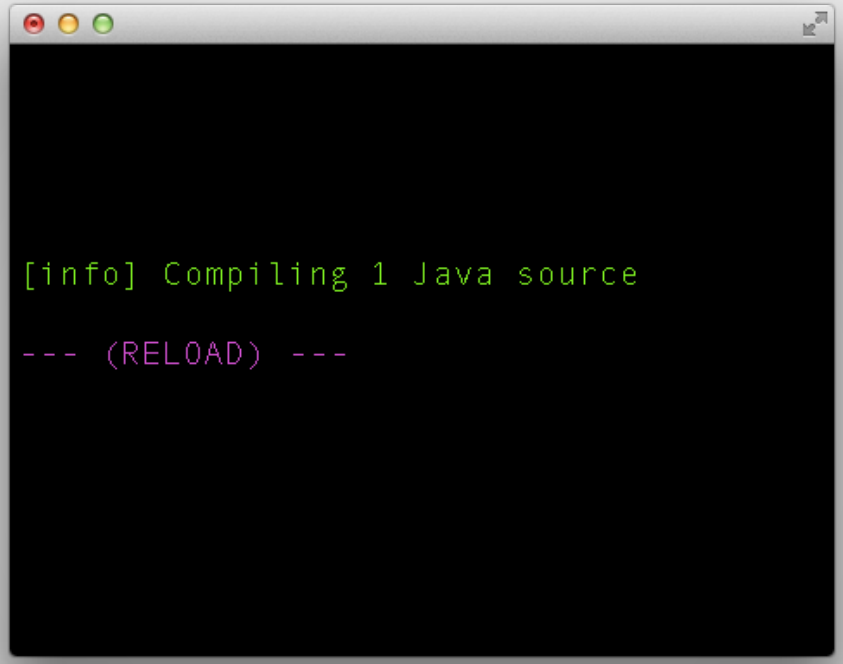
conf/routes

Expose the controller/action at a URL

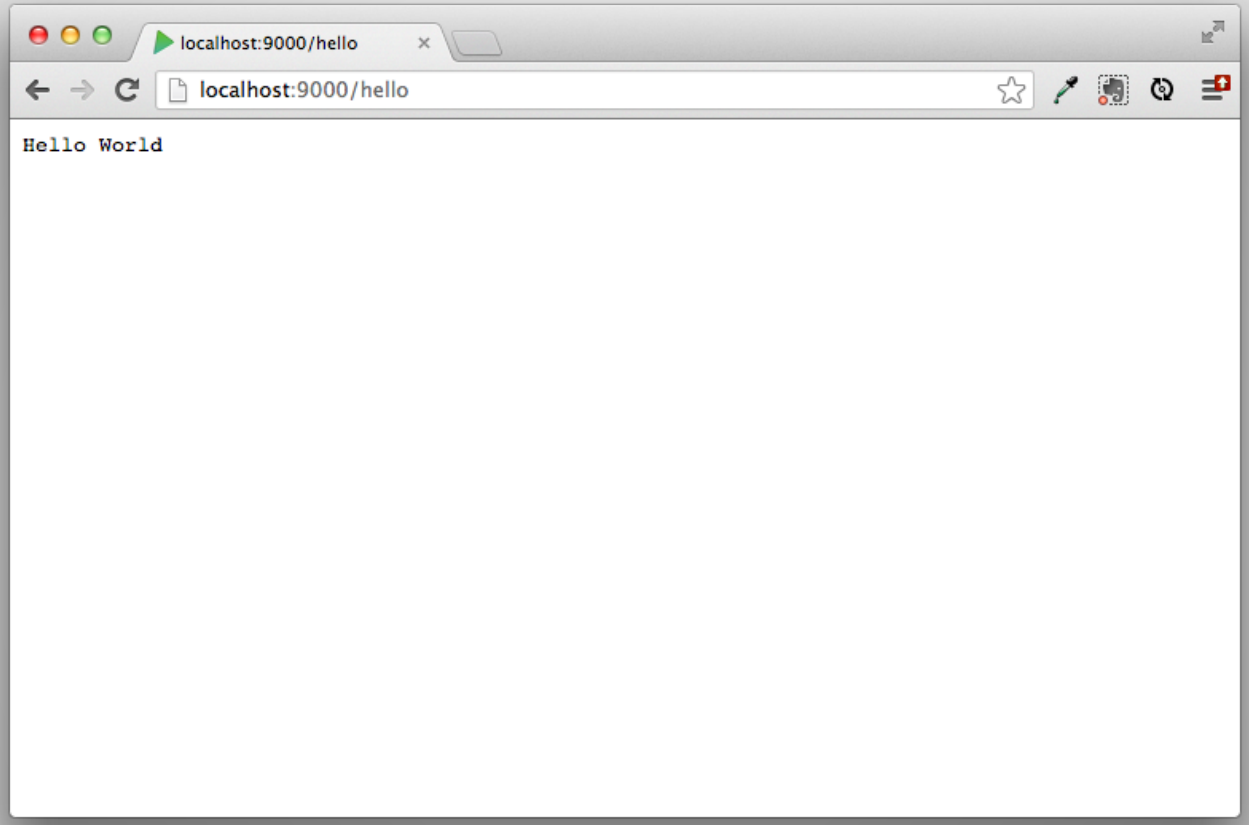
Now, restart the server.



Nah, just kidding. Refresh the page.



Woohoo, hot reload!



http://localhost:9000/hello

```java
public class HelloWorld extends Controller {

  public static Result index(String name) {
    return ok("Hello " + name);
  }
}
```

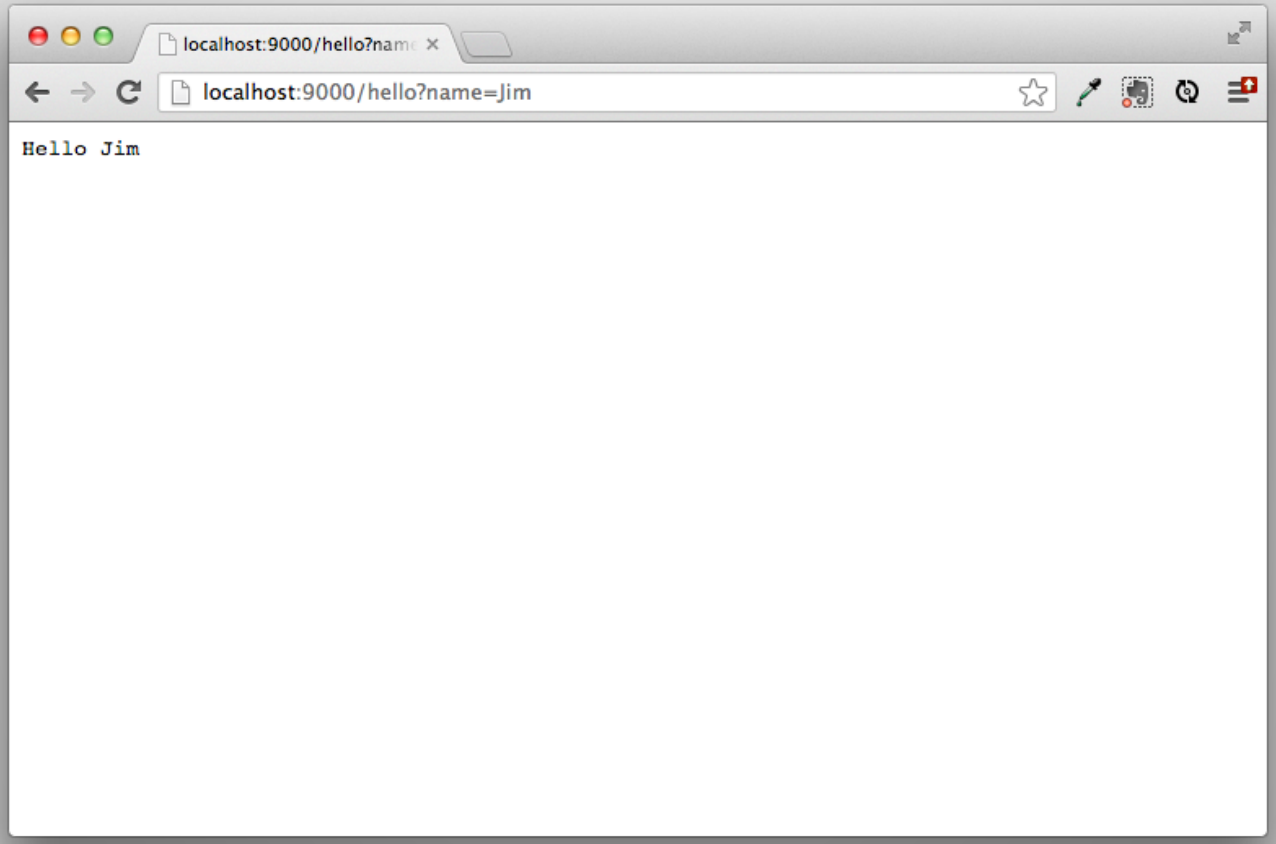
app/controllers/HelloWorld.java

Add a parameter

```
GET    /hello  controllers.HelloWorld.index( name)
```

conf/routes

Read the parameter from the query string

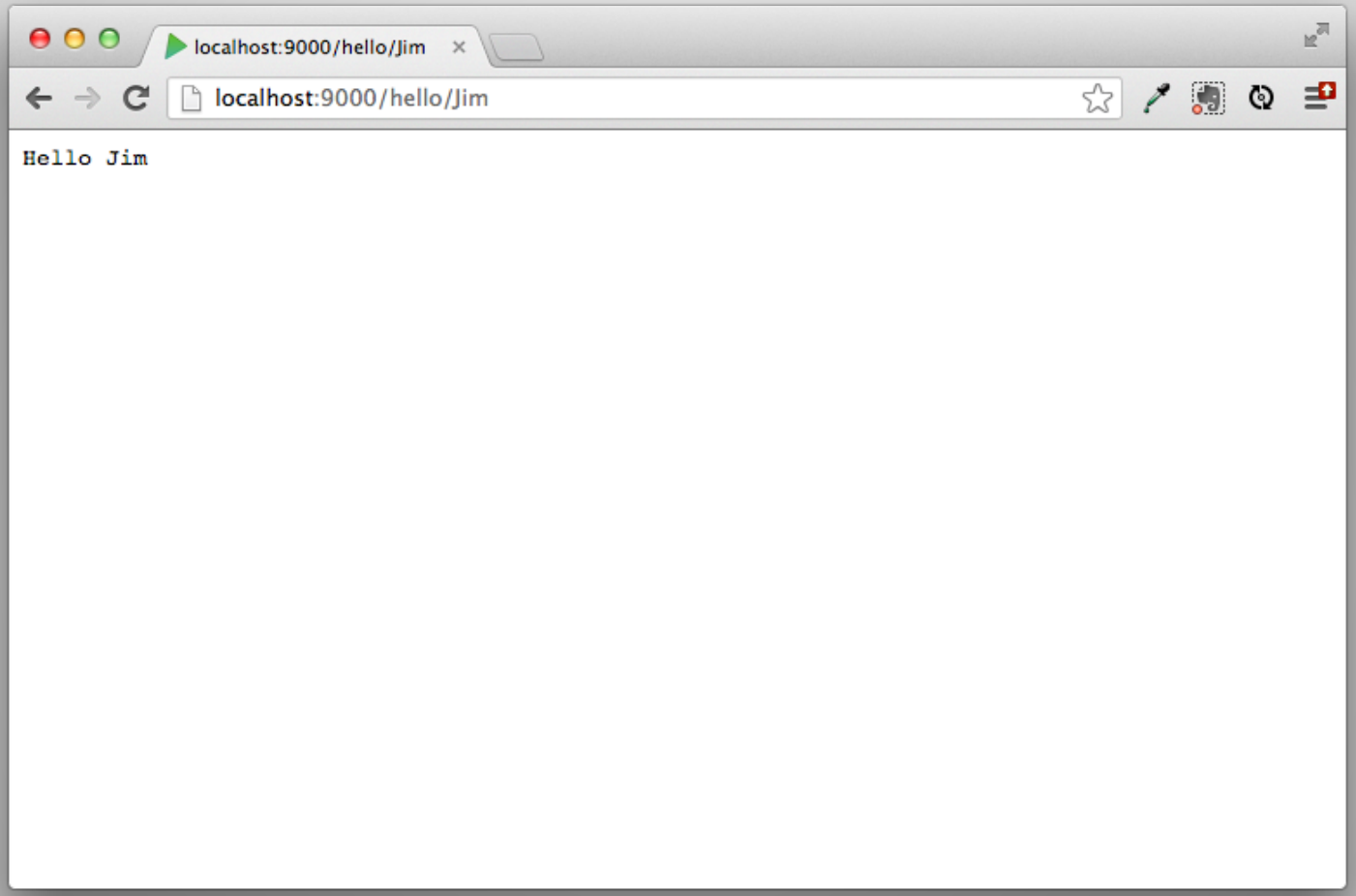


http://localhost:9000/hello?name=Jim

```
GET    /hello/:name   controllers.HelloWorld.index(name)
```

conf/routes

Read the parameter from the URL instead



http://localhost:9000/hello/Jim

```
public class HelloWorld extends Controller {

  public static Result index(String name, int age) {
    return ok("Hello " + name + " you are " + age +
              " years old");
  }
}
```

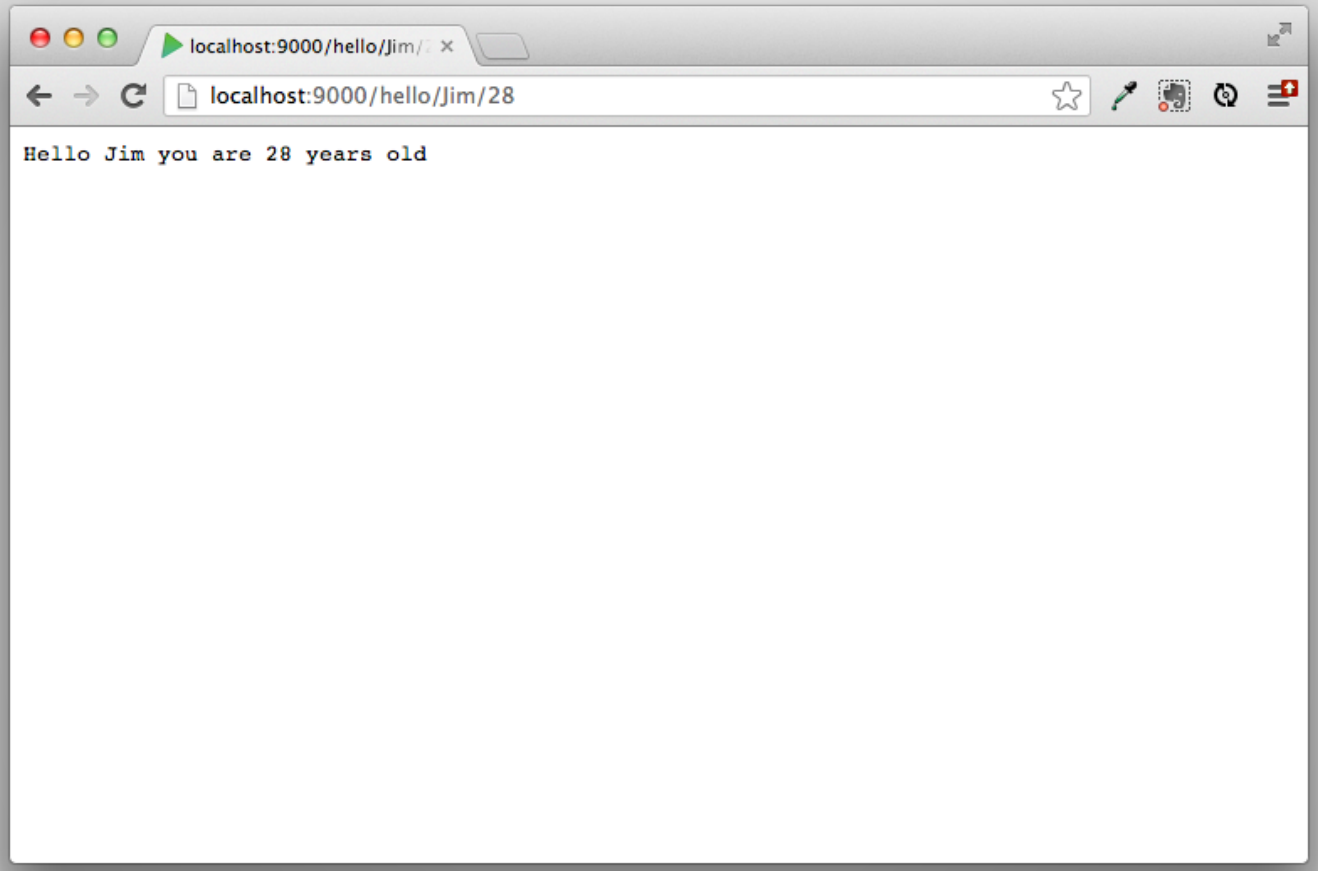
app/controllers/HelloWorld.java

Add another parameter, this time an `int`

```
GET    /hello/:name/:age   controllers.HelloWorld.index(name: String,  age: Int)
```

conf/routes

Add the parameter. Note the type checking!



http://localhost:9000/hello/Jim/28

```
@(name: String, age: Int)

<html>
  <head></head>
  <body>
    <img src="/assets/images/play-logo.png"/>
    <p>
      Hello <b>@name</b>, you are <b>@age</b> years old
    </p>
  </body>
</html>
```

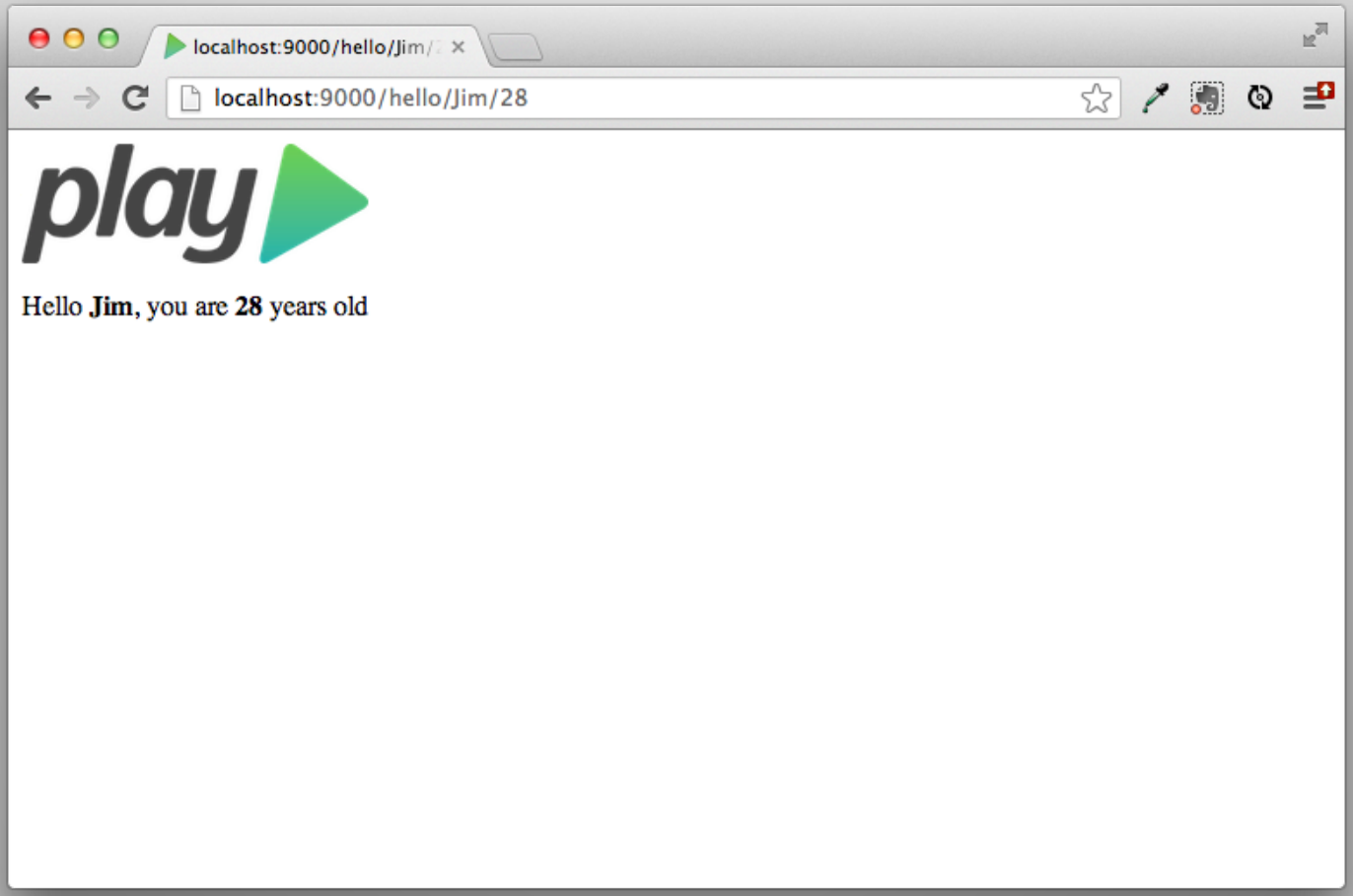
app/views/hello.scala.html

Add a view

```java
public class HelloWorld extends Controller {

  public static Result index(String name, int age) {
    return ok(views.html.hello.render(name, age));
  }
}
```

app/controllers/HelloWorld.java

Render the view from the controller



http://localhost:9000/hello/Jim/28

```
location = "JaxConf"
```

app/conf/application.conf

How about some config?

```java
public class HelloWorld extends Controller {

  public static Result index(String name, int age) {
    String location = getConfig().getString("location");
    return ok(views.html.hello.render(name, age,
location));
  }

  private static Configuration getConfig() {
    return Play.application().configuration();
  }
}
```

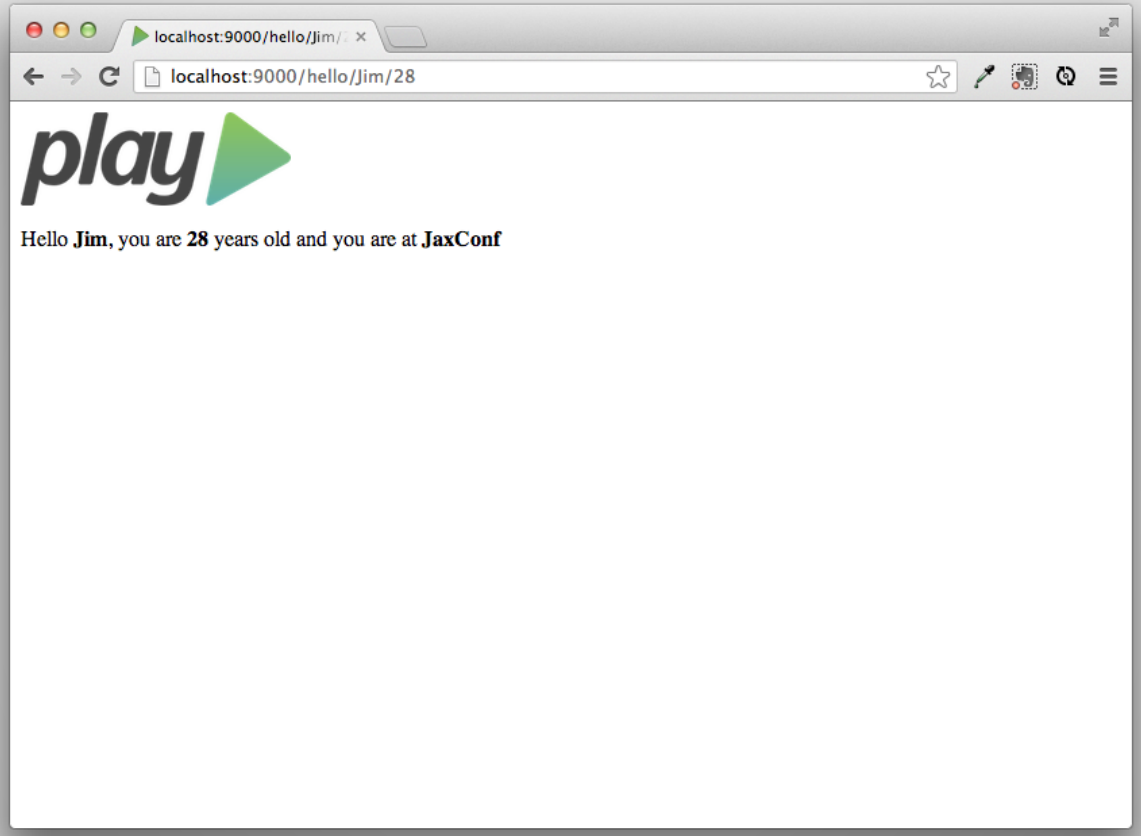
app/controllers/HelloWorld.java

Read the config and pass it to the view

```
@(name: String, age: Int, location: String)

<html>
  <head></head>
  <body>
    <img src="/assets/images/play-logo.png"/>
    <p>
      Hello <b>@name</b>, you are <b>@age</b> years old
      and you are at <b>@location</b>
    </p>
  </body>
</html>
```

app/views/hello.scala.html

Update the view

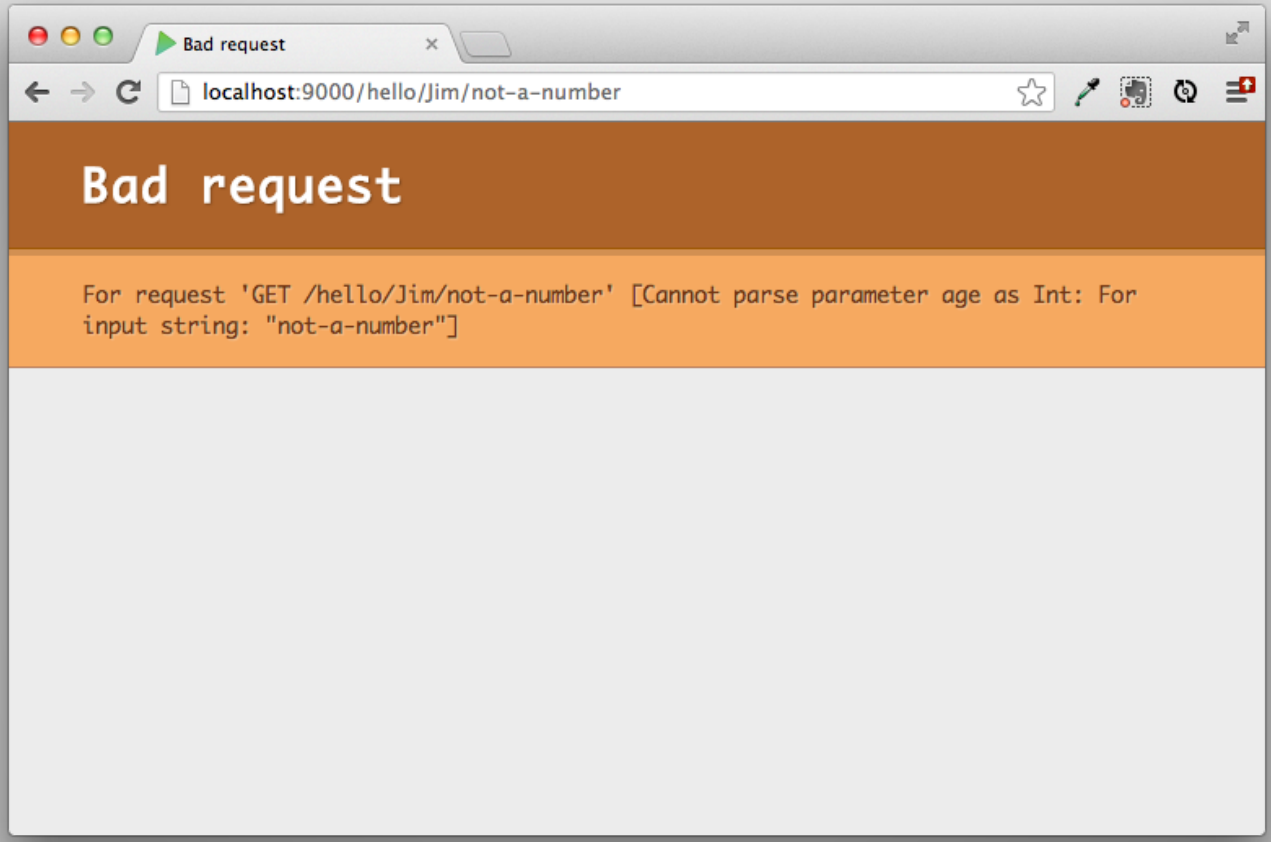


Refresh the page. Config hot reloads too!

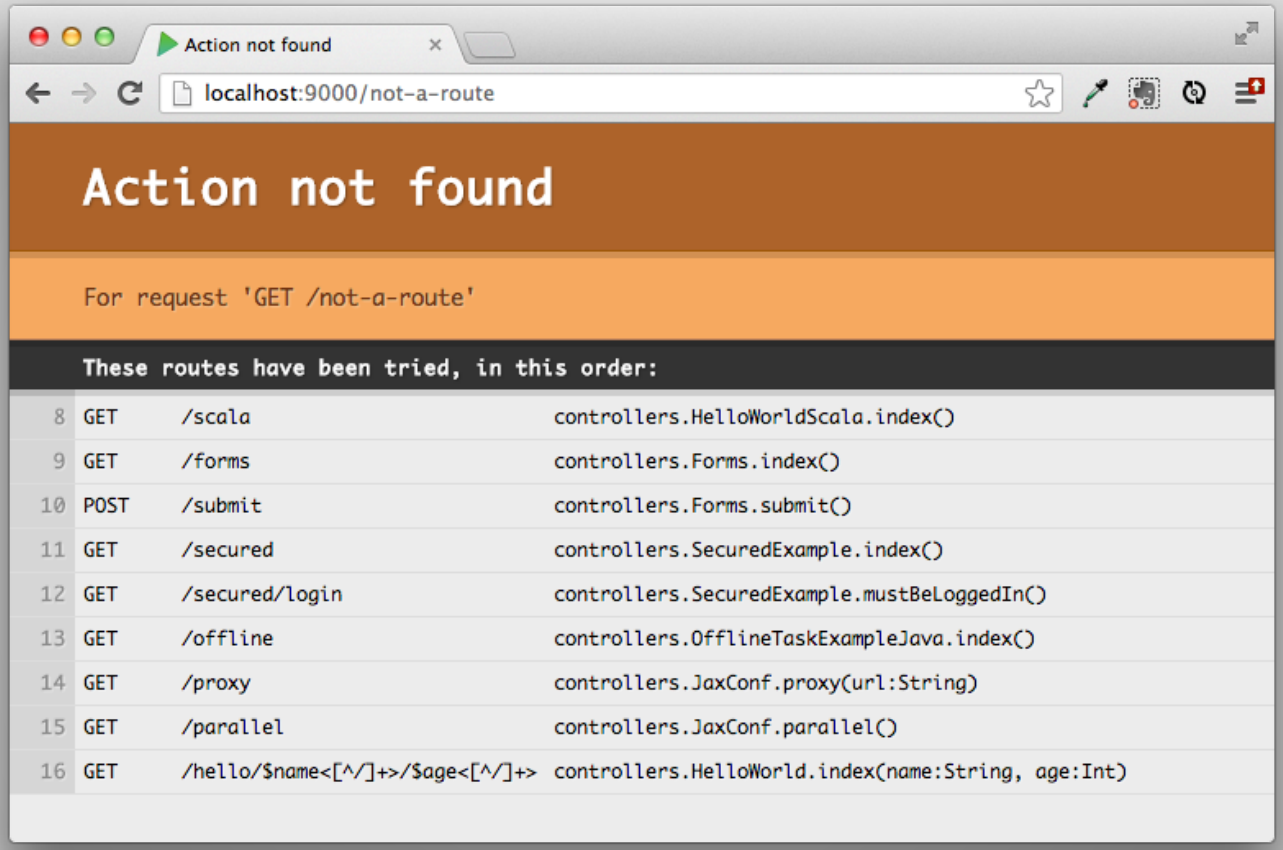
Outline

1. Getting started with Play
2. Make a change and reload
3. **Error handling**
4. Threaded vs. evented
5. Non-blocking I/O
6. Scala
7. Performance
8. Community

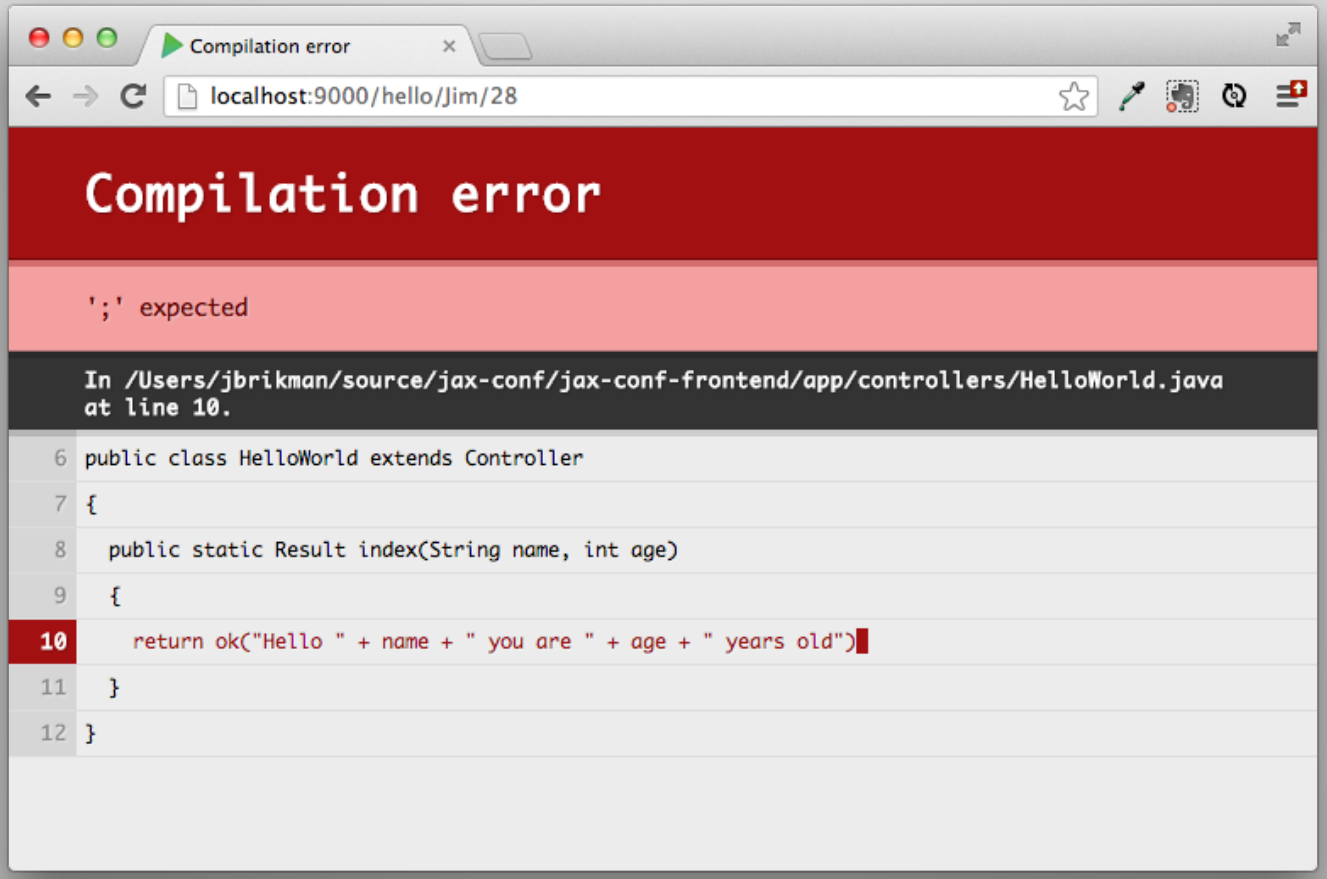
In dev mode, Play shows error messages *right in the browser*.



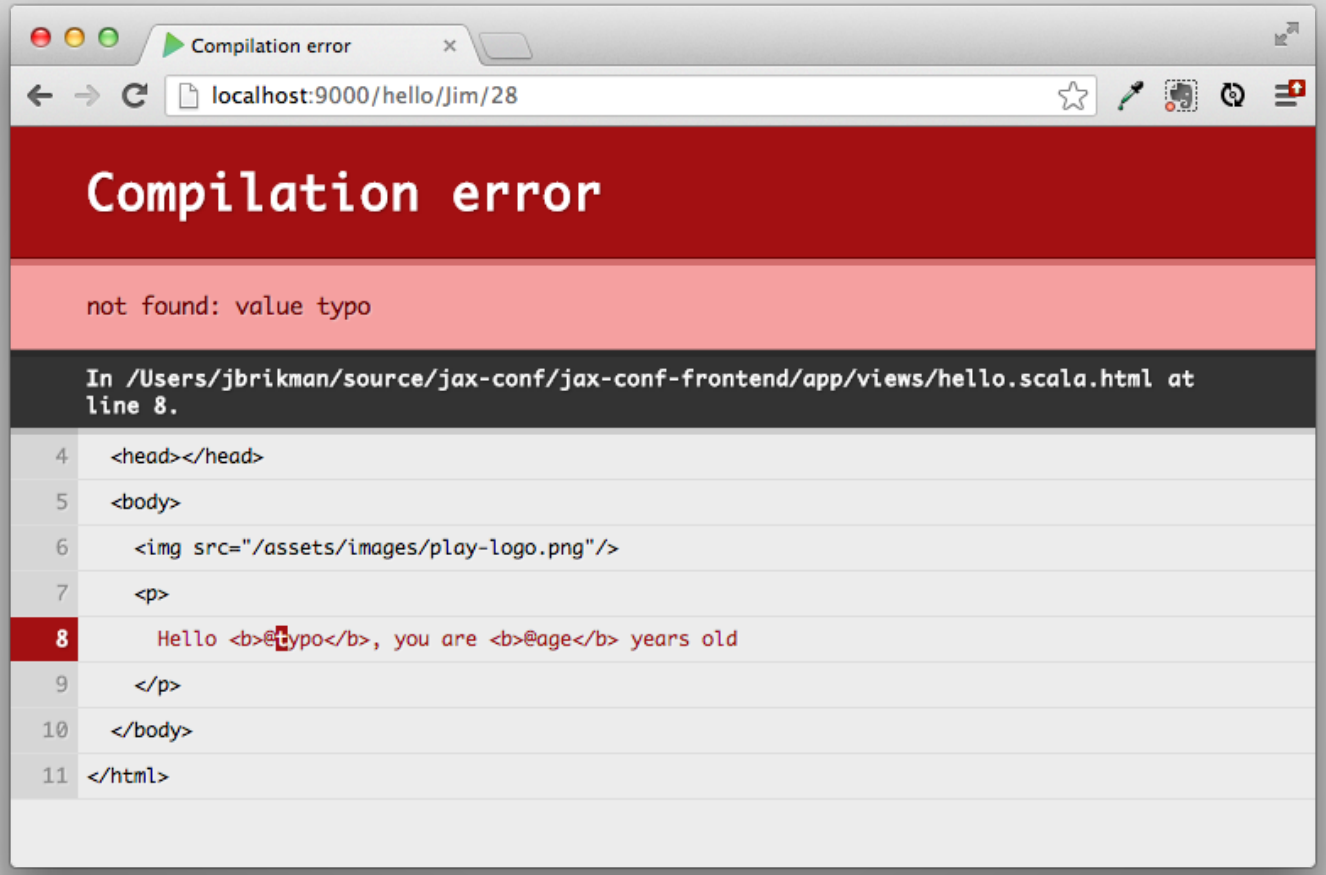
Parameter type checking



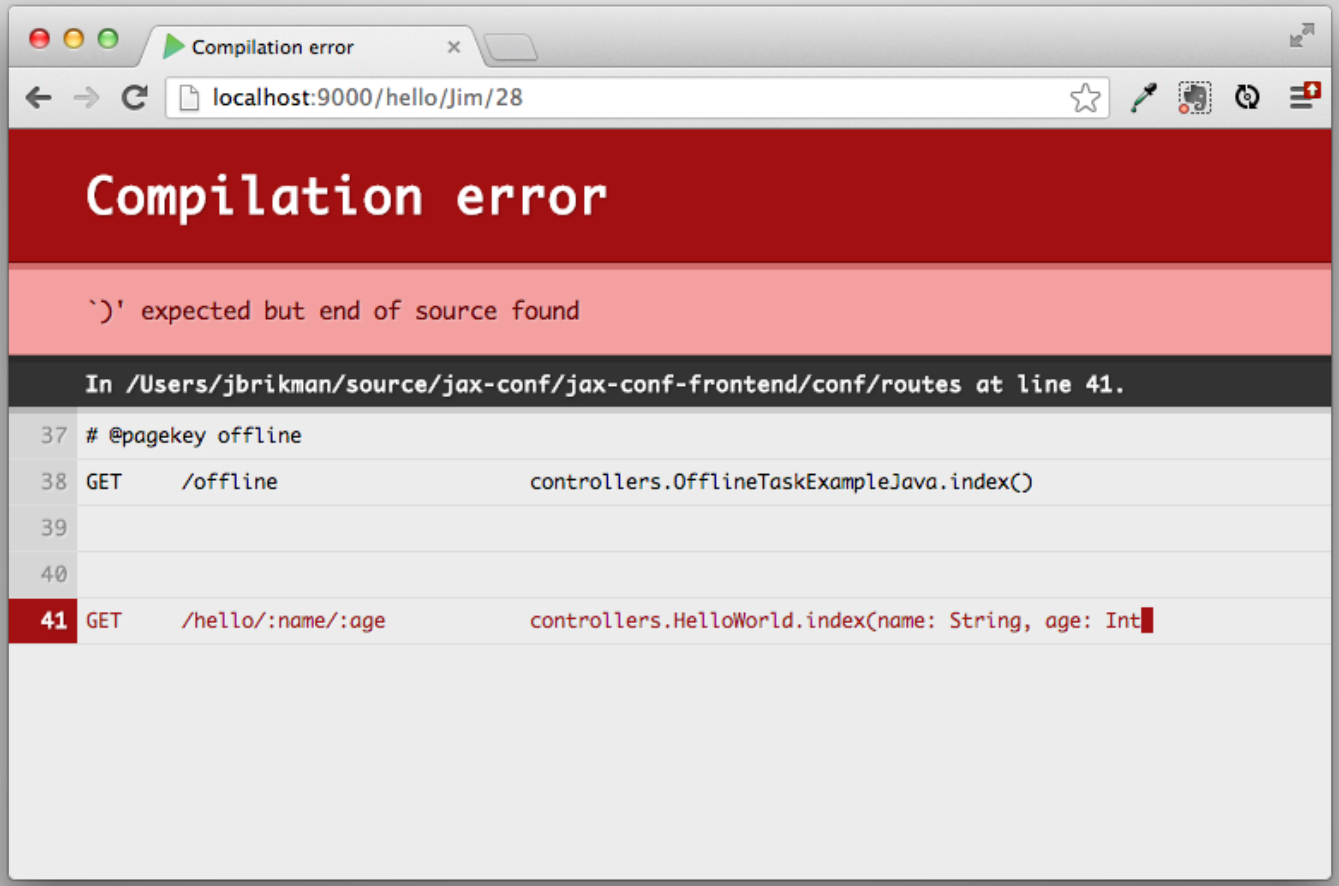
A helpful 404 page



Compile errors show problematic source code in the browser



Views are compiled as well



The routes file too

Outline

1. Getting started with Play
2. Make a change and reload
3. Error handling
4. **Threaded vs. evented**
5. Non-blocking I/O
6. Scala
7. Performance
8. Community

Most people are used to threaded servers

```
void doGet(HttpServletRequest req, HttpServletResponse res) {
  // Apache HttpClient
  HttpClient client = new HttpClient();
  GetMethod method = new GetMethod("www.example.com/");

  // executeMethod is a blocking, synchronous call
  int statusCode = client.executeMethod(method);
  System.out.println("Response " + statusCode);
}
```

MyServlet.java

Threaded servers assign one thread per request and use **blocking** I/O

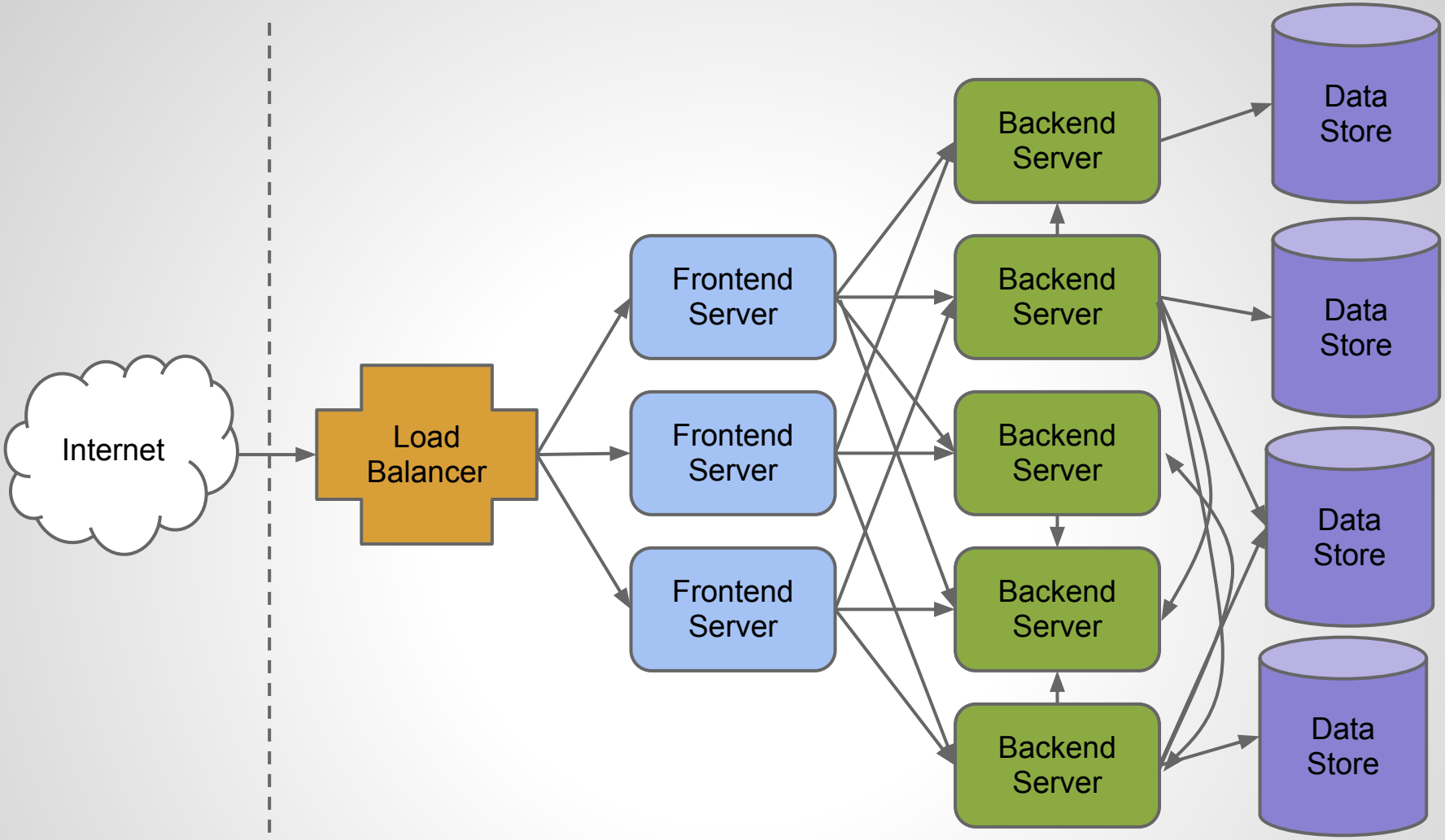
Evented servers are gaining popularity

```
var callback = function(data) {
  console.log("Response: " + data);
};

var options = {
  hostname: 'www.google.com',
  path: '/upload'
};

// Non-blocking HTTP call
http.request(options, callback);

console.log('This line may execute before the callback!');
```

MyNodeApp.js

Evented servers have one thread/process per CPU core and use **non-blocking** I/O

Why threaded vs. evented matters for LinkedIn

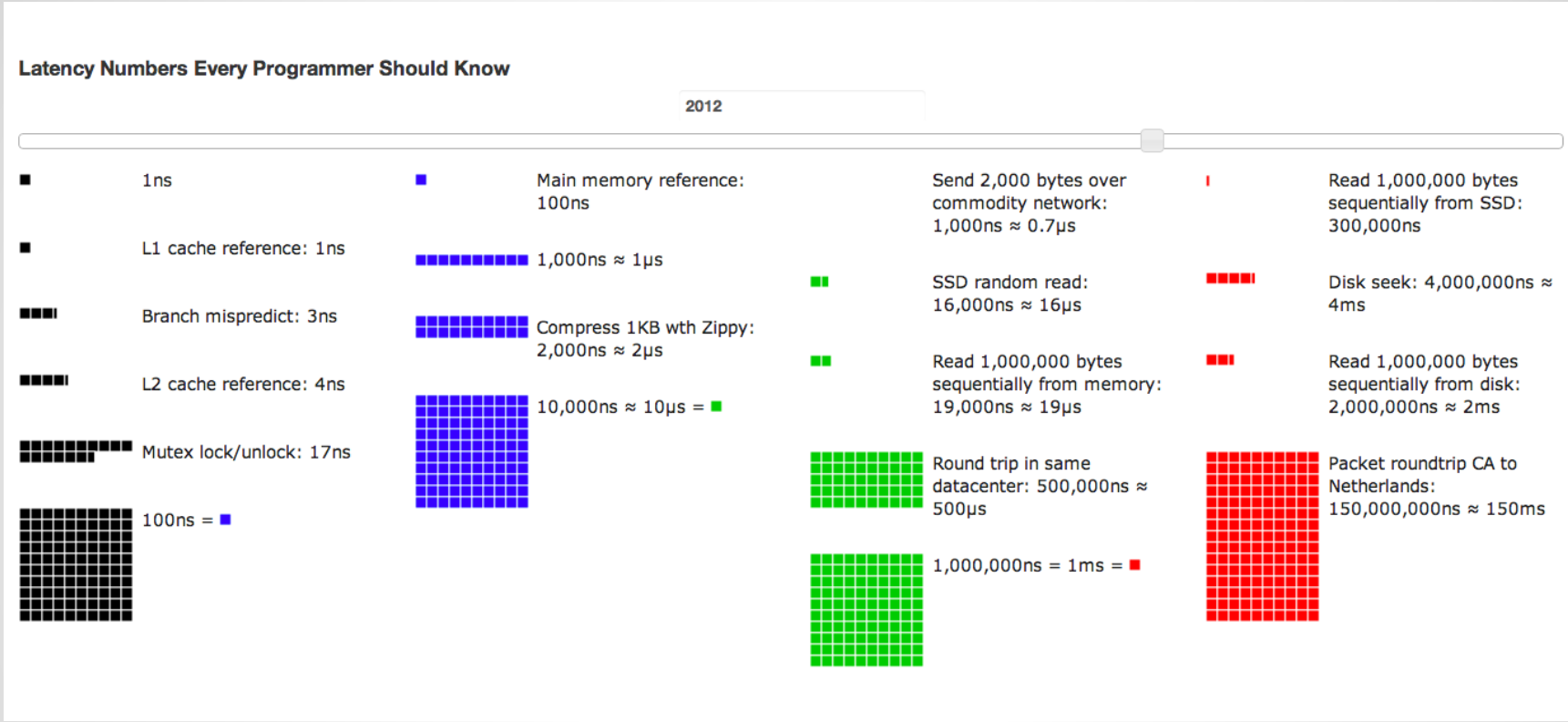


LinkedIn uses a Service Oriented Architecture

```
void doGet(HttpServletRequest req, HttpServletResponse res) {
  // Call a number of backend services to get data
  Profile profile = profileSvc.getProfile();
  Company company = companySvc.getCompany();
  Skills skills = skillsSvc.getSkills();
}
```

MyServlet.java

Our services spend most of their time **waiting** for data from other services and data stores



I/O is very expensive [4]

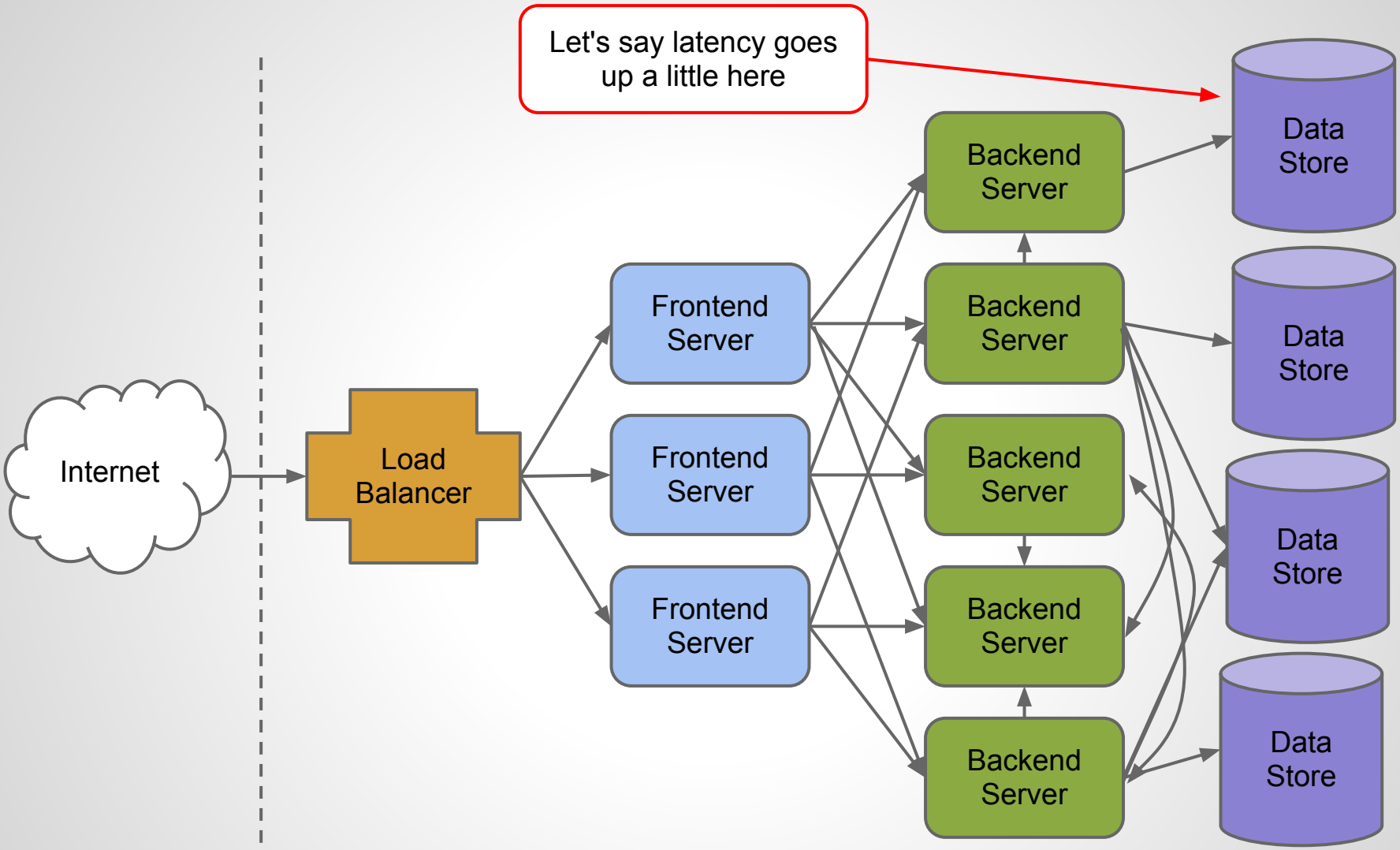
In a threaded server, threads spend most of the time **idle**, waiting on I/O

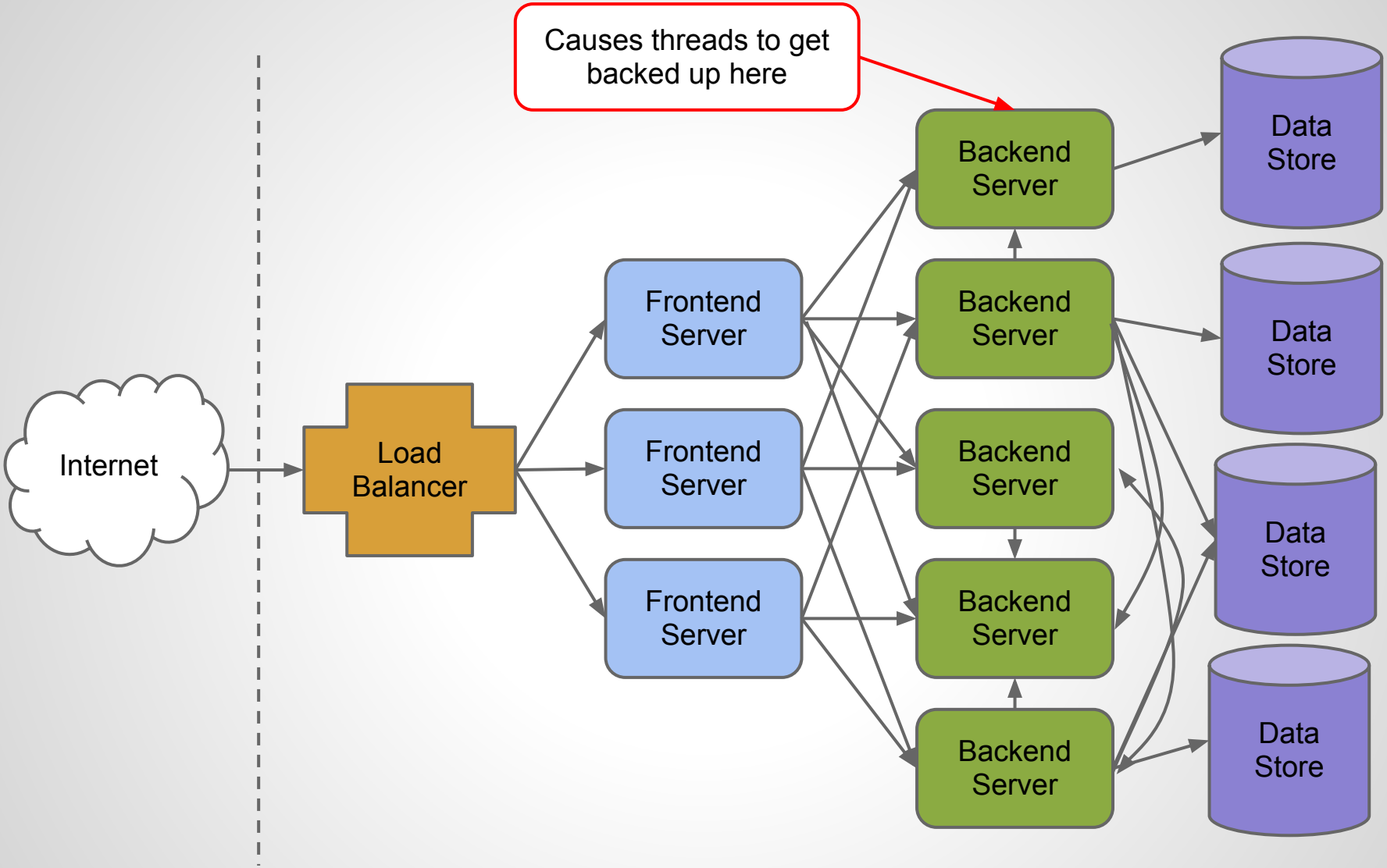
Threading dilemma

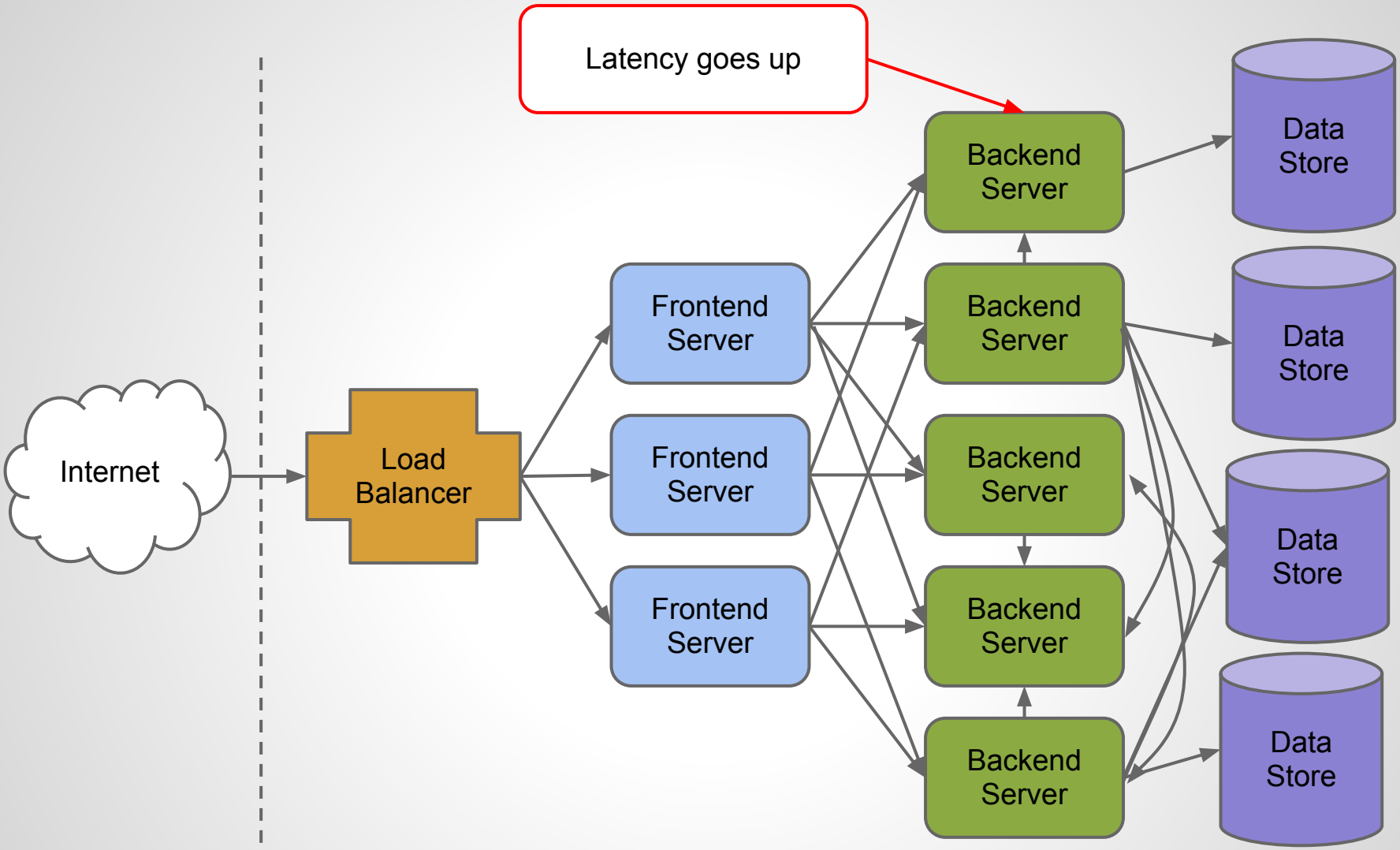
1. Creating new threads on the fly is expensive:
   a. Use a thread pool

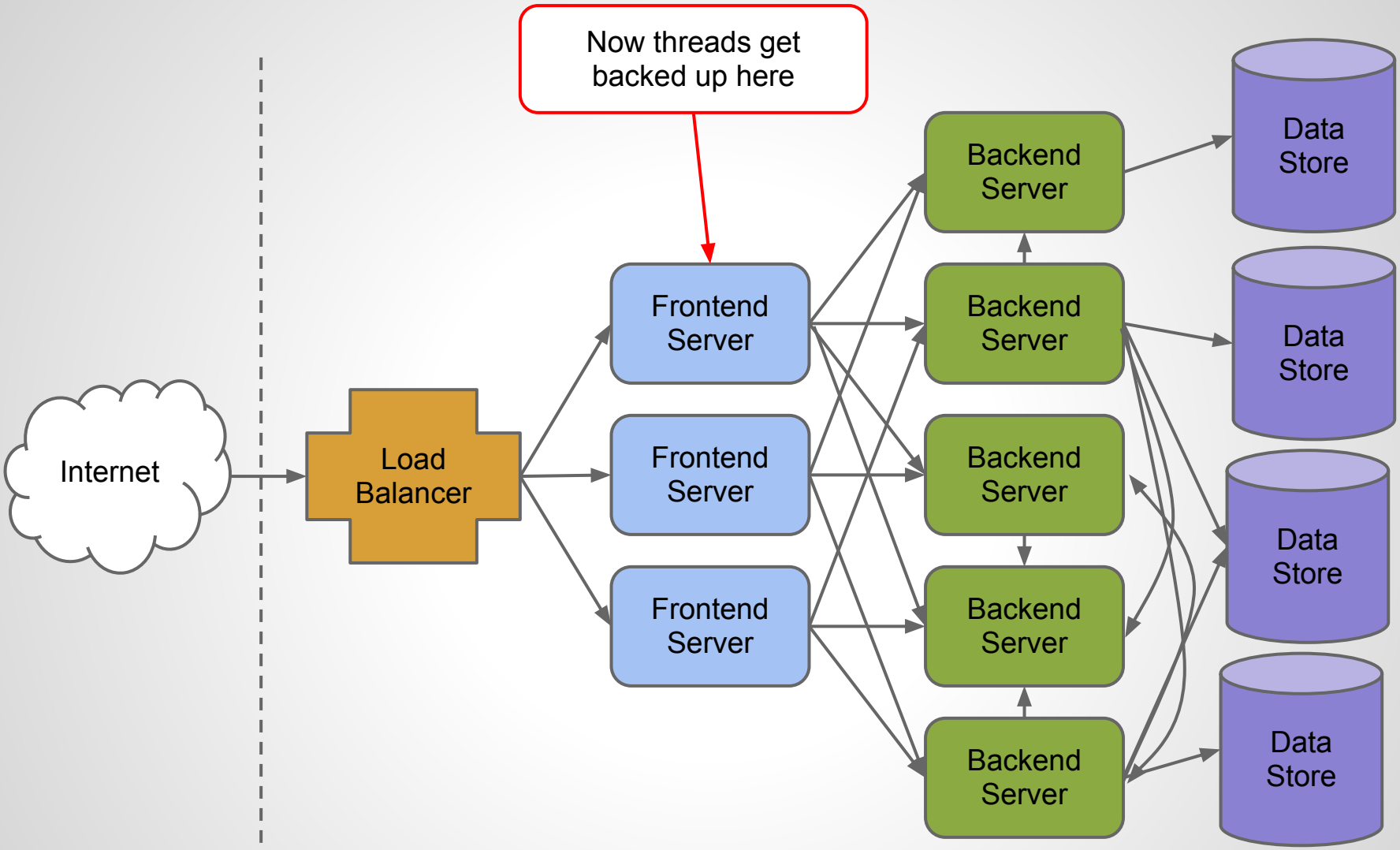
2. Too many threads in the thread pool:
   a. Memory overhead
   b. Context switching overhead

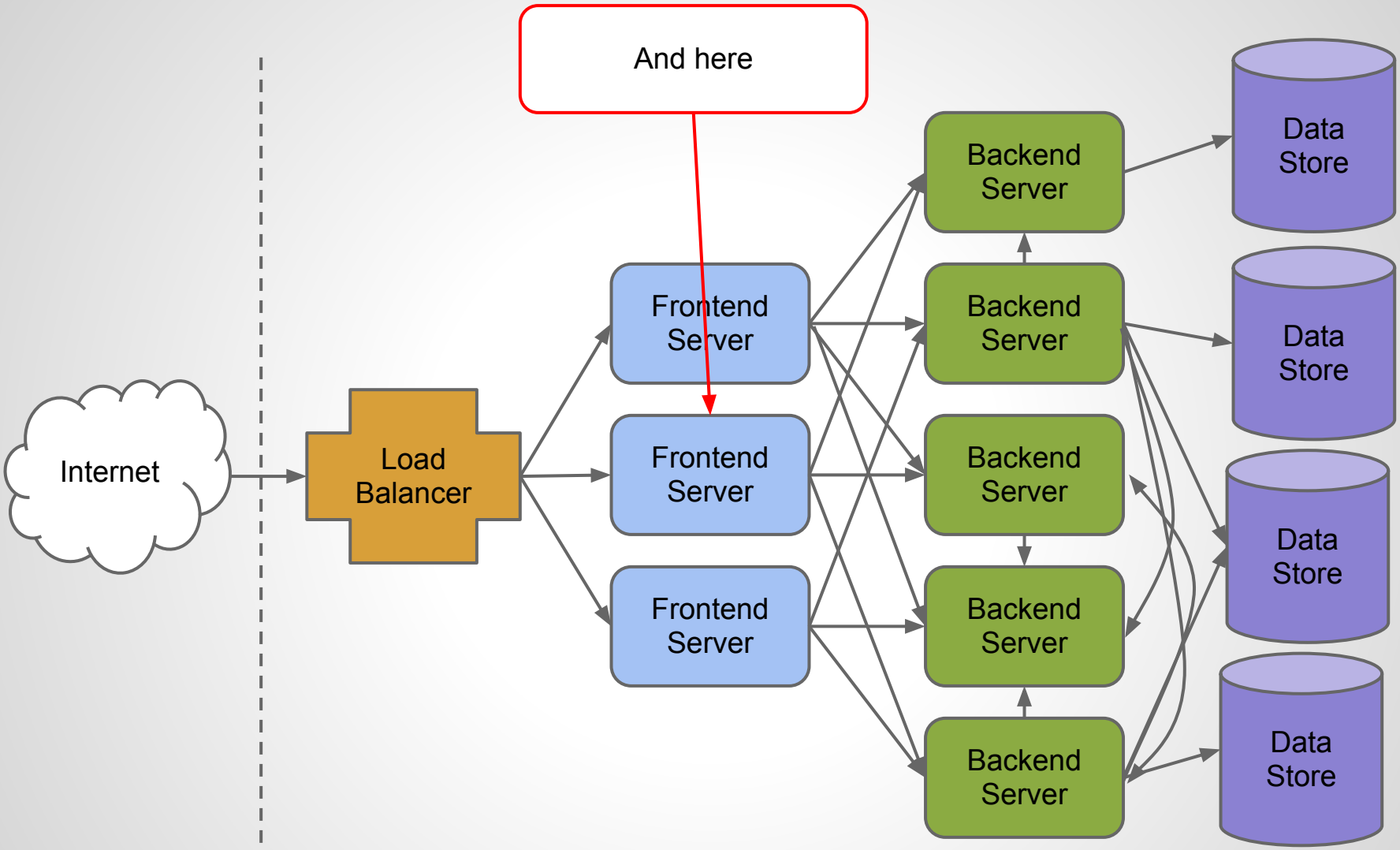
3. Too few threads in the thread pool:
   a. Run out of threads, latency goes up
   b. Sensitive to downstream latency!

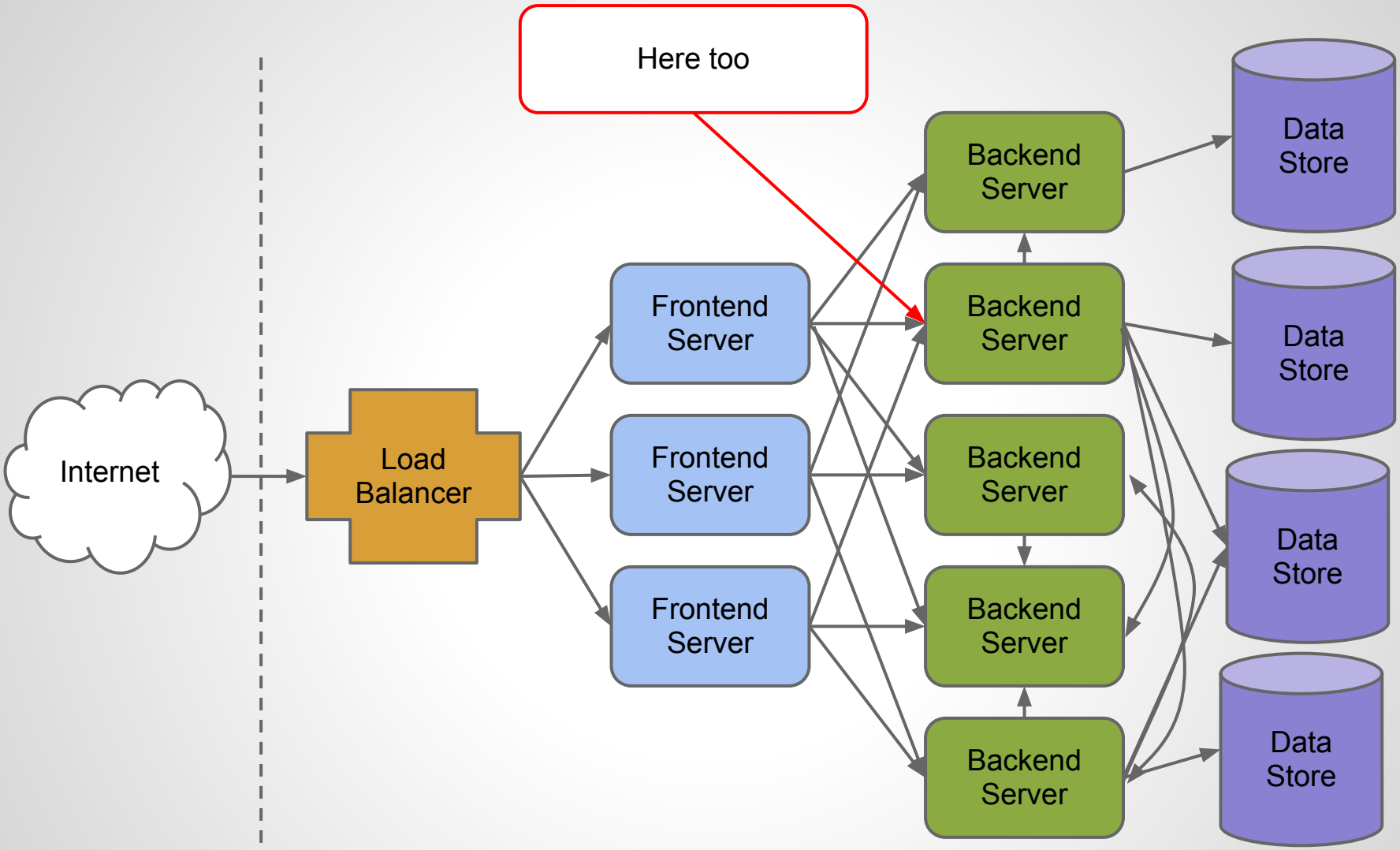

Let's say latency goes up a little here
Internet
Load Balancer
Frontend Server
Frontend Server
Frontend Server
Backend Server
Backend Server
Backend Server
Backend Server
Backend Server
Data Store
Data Store
Data Store
Data Store

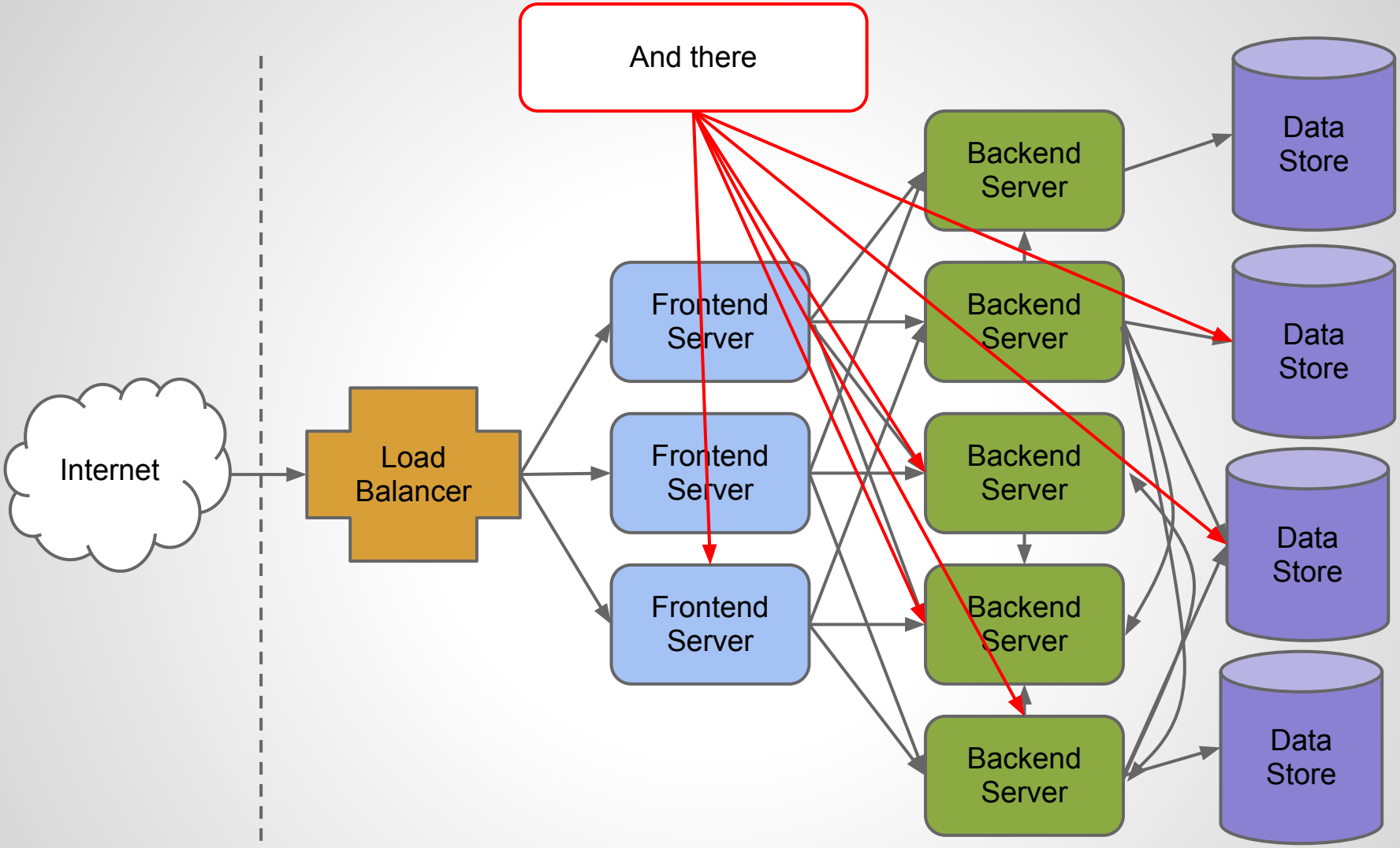

Causes threads to get backed up here
Internet
Load Balancer
Frontend Server
Frontend Server
Frontend Server
Backend Server
Backend Server
Backend Server
Backend Server
Backend Server
Data Store
Data Store
Data Store
Data Store


Latency goes up
Internet
Load Balancer
Frontend Server
Frontend Server
Frontend Server
Backend Server
Backend Server
Backend Server
Backend Server
Backend Server
Data Store
Data Store
Data Store
Data Store


Now threads get backed up here
Internet
Load Balancer
Frontend Server
Frontend Server
Frontend Server
Backend Server
Backend Server
Backend Server
Backend Server
Backend Server
Data Store
Data Store
Data Store
Data Store


And here
Internet
Load Balancer
Frontend Server
Frontend Server
Frontend Server
Backend Server
Backend Server
Backend Server
Backend Server
Backend Server
Data Store
Data Store
Data Store
Data Store


Here too
Internet
Load Balancer
Frontend Server
Frontend Server
Frontend Server
Backend Server
Backend Server
Backend Server
Backend Server
Backend Server
Data Store
Data Store
Data Store
Data Store


And there
Internet
Load Balancer
Frontend Server
Frontend Server
Frontend Server
Backend Server
Backend Server
Backend Server
Backend Server
Backend Server
Data Store
Data Store
Data Store
Data Store

And... the site is down.

This is thread pool hell [5]

Play is built on top of Netty, so it supports non-blocking I/O [6]

NIO benefits

1. No sensitivity to downstream slowness
2. Easy to parallelize I/O
3. Supports many concurrent and long-running connections, enabling:
    a. WebSockets
    b. Comet
    c. Server-Sent Events

Outline

1. Getting started with Play
2. Make a change and reload
3. Error handling
4. Threaded vs. evented
5. **Non-blocking I/O**
6. Scala
7. Performance
8. Community

```java
public class Proxy extends Controller {

  public static Result index(String url) {
    // Non blocking HTTP call
    Promise<Response> responsePromise = WS.url(url).get();

    // How do we turn a Promise into a Play Result?
  }
}
```

app/controllers/Proxy.java

Use Play's built in `WS` library to make a non-blocking HTTP call

A `Promise<T>` will *eventually* contain the value `T` (or an error)

```
public static class AsyncResult implements Result {
  private final Promise<Result> promise;

  public AsyncResult(Promise<Result> promise) {
    this.promise = promise;
  }
}
```

(Play Framework source code)

Play has a built-in subclass of `Result` called `AsyncResult` that takes a `Promise<Result>`

```java
public class Proxy extends Controller {
  public static Result index(String url) {
    // Non blocking HTTP call
    Promise<Response> response = WS.url(url).get();

    // Transform asynchronously into a Play Result
    Promise<Result> result = response.map(toResult);

    return async(result);
  }

  // A function that can transform a Response into a Result
  private static Function<Response, Result> toResult =
    new Function<Response, Result>() {
      public Result apply(Response response) {
        return ok(response.getBody()).as("text/html");
      }
    };
}
```

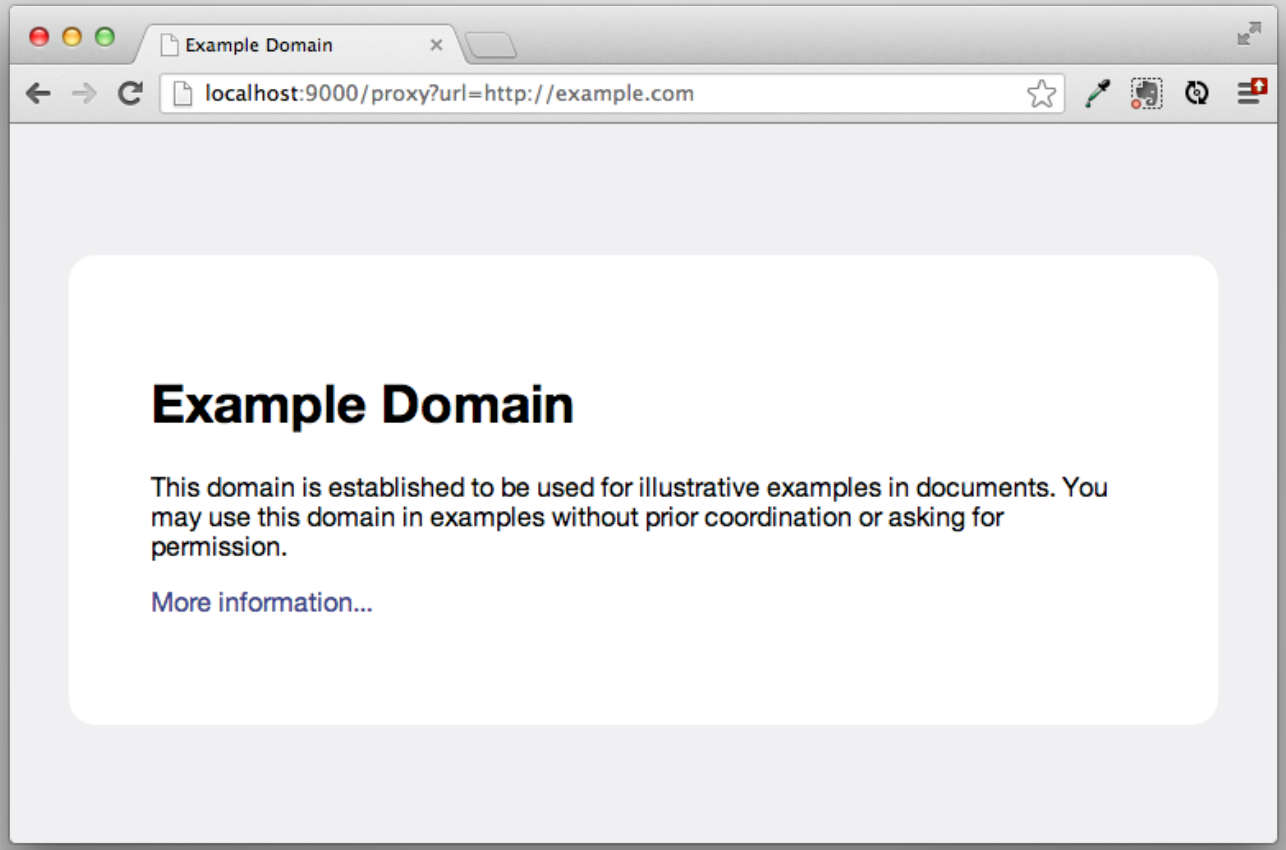
app/controllers/Proxy.java

We can use the `map` method to turn a `Promise<Response>` into a `Promise<Result>`

```
GET     /proxy    controllers.Proxy.index(url)
```

conf/routes

Let's add this endpoint to the routes file



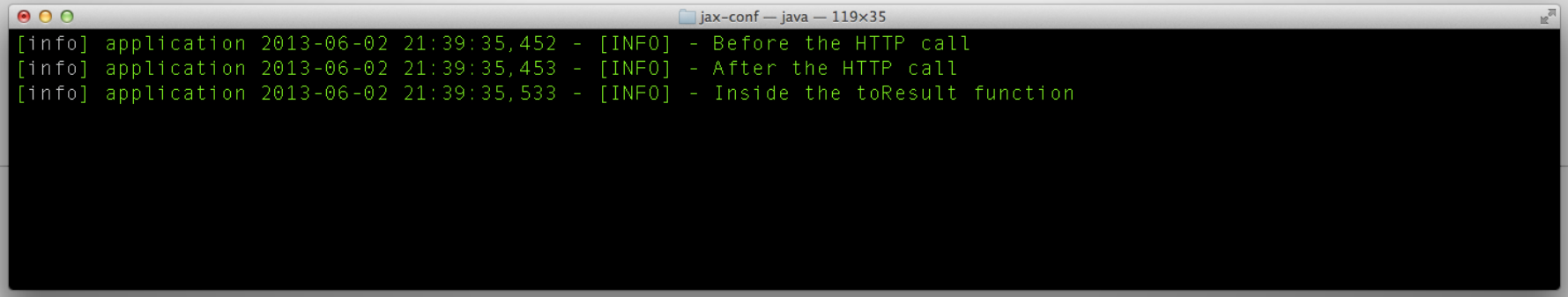
http://localhost:9000/proxy?url=http://example.com

We just built a completely non-blocking proxy!

```java
public class Proxy extends Controller {
  public static Result index(String url) {
    Logger.info("Before the HTTP call");
    Promise<Response> response = WS.url(url).get();
    Promise<Result> result = response.map(toResult);
    Logger.info("After the HTTP call");

    return async(result);
  }

  private static Function<Response, Result> toResult =
    new Function<Response, Result>() {
      public Result apply(Response response) {
        Logger.info("Inside the toResult function");
        return ok(response.getBody()).as("text/html");
      }
    };
}
```

app/controllers/Proxy.java

To see that it's non-blocking, let's add logging



Refresh the page and the logs show the HTTP call really is non-blocking

NIO makes parallel requests easy.
Let's try it.

```java
class Timing {
  public String url;
  public long latency;

  public Timing(String url, long latency) {
    this.url = url;
    this.latency = latency;
  }

  // Fire an HTTP request and record how long it took
  public static Promise<Timing> timedRequest(final String url) {
    final long start = System.currentTimeMillis();
    Promise<Response> res = WS.url(url).get();

    return res.map(new Function<Response, Timing>() {
      public Timing apply(Response response) {
        long latency = System.currentTimeMillis() - start;
        return new Timing(url, latency);
      }
    });
  }
}
```

app/models/Timing.java

First, define a class that times an HTTP request

```java
public class Parallel extends Controller {
  public static Result index() {
    // A Promise that will redeem when the 3 parallel
    // HTTP requests are done
    Promise<List<Timing>> all = Promise.waitAll(
      Timing.timedRequest("http://www.yahoo.com"),
      Timing.timedRequest("http://www.google.com"),
      Timing.timedRequest("http://www.bing.com")
    );

    // Asynchronously transform timings into a JSON response
    return async(all.map(new Function<List<Timing>, Result>() {
      public Result apply(List<Timing> timings) {
        return ok(Json.toJson(timings));
      }
    }));
  }
}
```

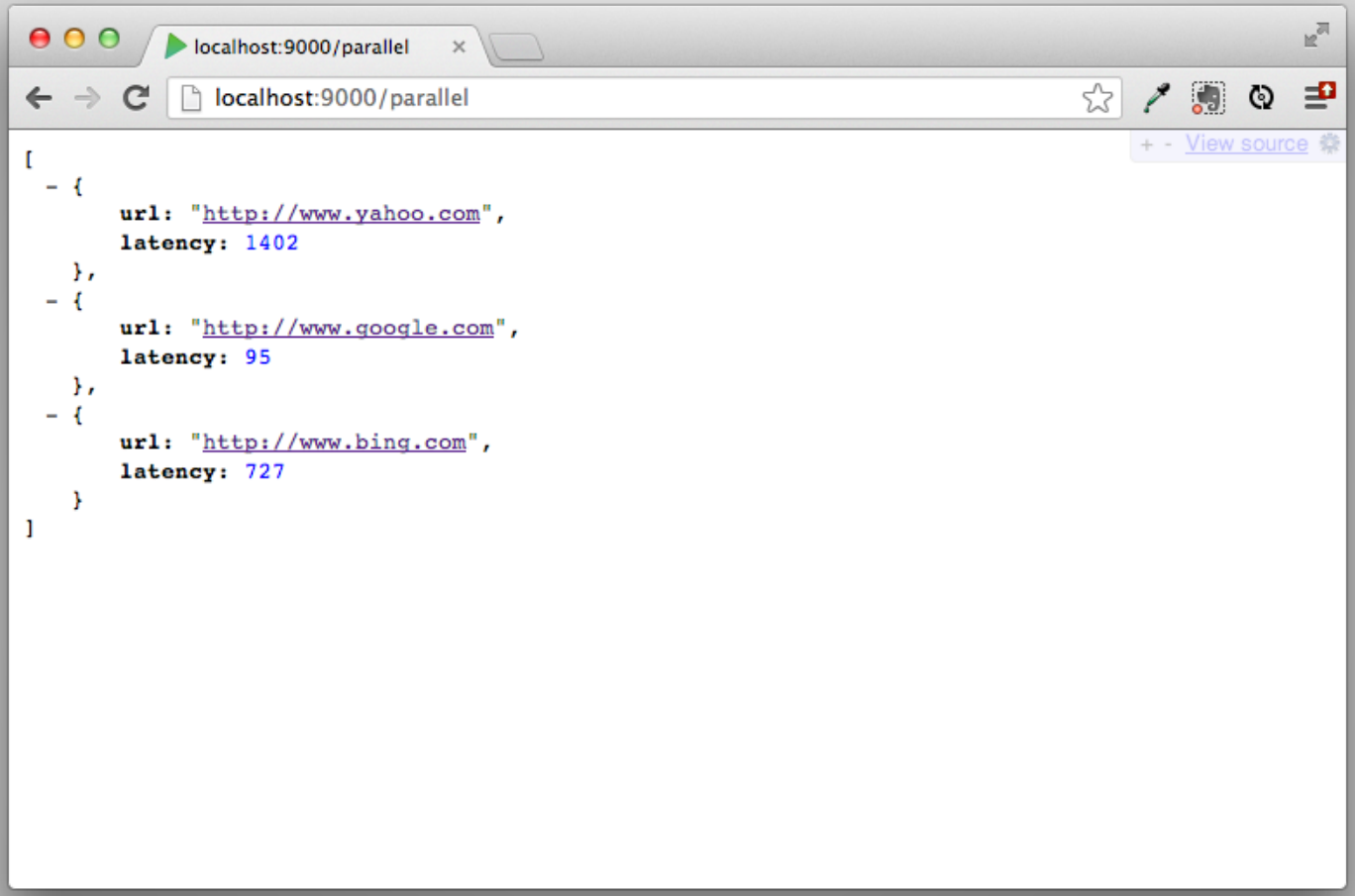
app/controllers/Parallel.java

Next, add a controller that fetches 3 URLs in parallel and returns the timings as JSON

```
GET     /parallel     controllers.Parallel.index()
```

conf/routes

Add it to the routes file



http://localhost:9000/parallel

Outline

1. Getting started with Play
2. Make a change and reload
3. Error handling
4. Threaded vs. evented
5. Non-blocking I/O
6. **Scala**
7. Performance
8. Community

Play has native support for Scala

```scala
object HelloWorldScala extends Controller {

  def index = Action {
    Ok("Hello World Scala")
  }
}
```

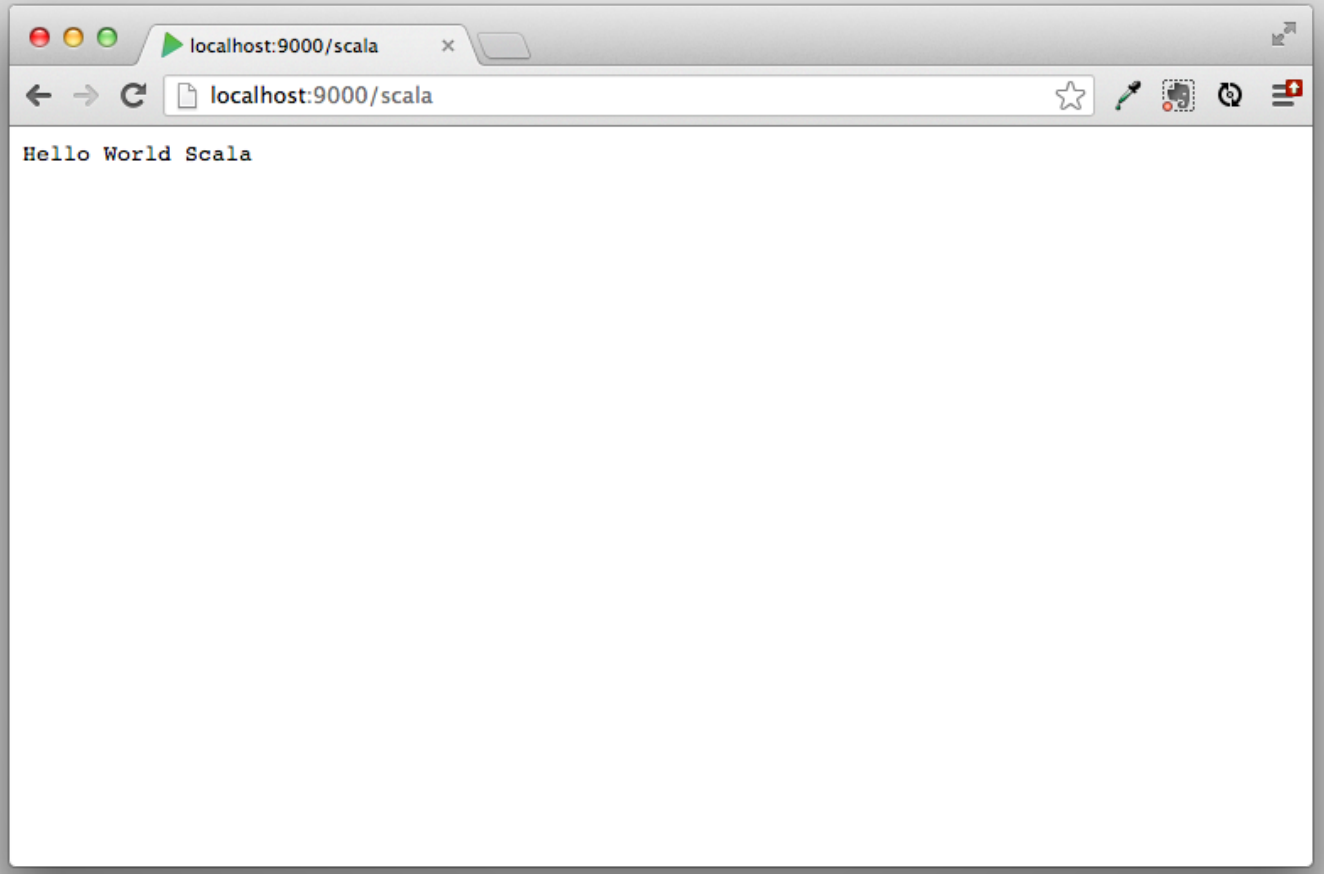
app/controllers/HelloWorldScala.scala

Just add a `.scala` file under `/app` and Play will compile it

```
GET     /scala    controllers.HelloWorldScala.index()
```

conf/routes

Add it to the routes file as usual



http://localhost:9000/scala

Play/Scala: the good parts

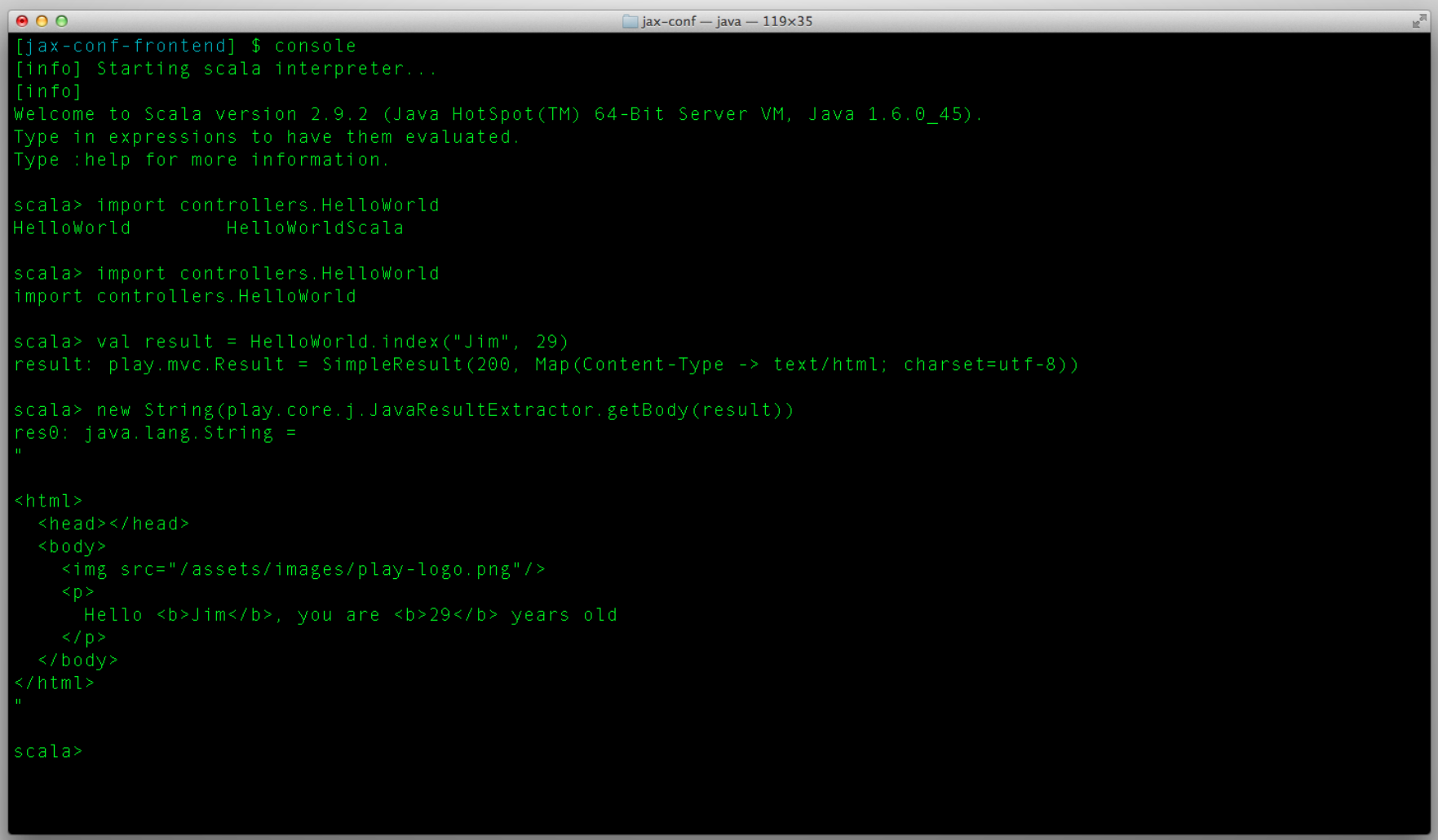
```
// 2 parallel async calls
val fooFuture = WS.url(url1).get()
val barFuture = WS.url(url2).get()

for {
  foo <- fooFuture
  bar <- barFuture
} yield {
  Ok(...)
}
```

Sequential async calls

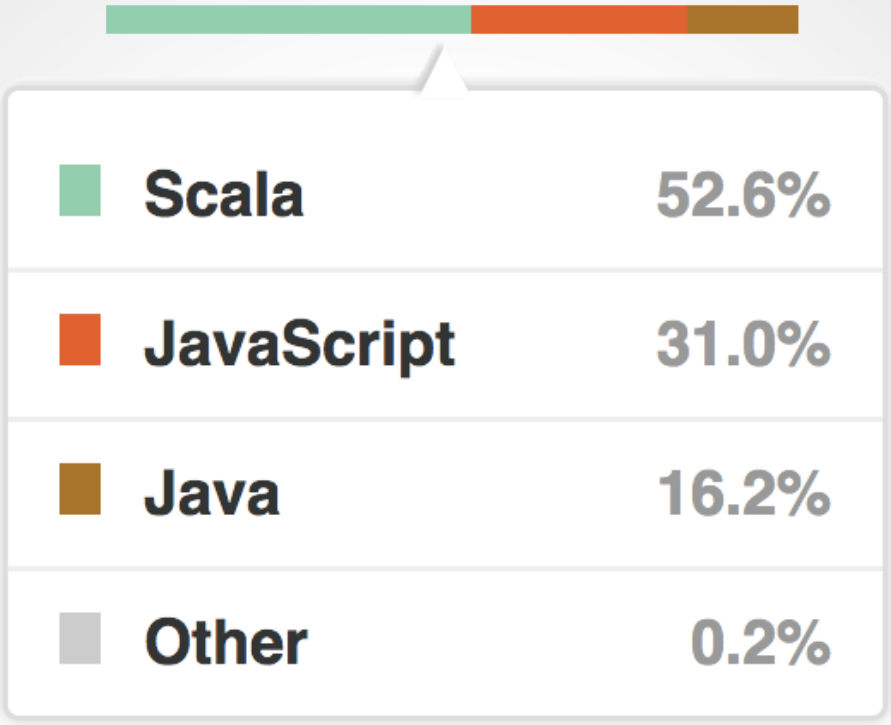
```
// 2 sequential async calls
for {
  foo <- WS.url(url1).get()
  bar <- WS.url(buildUrl(foo)).get()
} yield {
  Ok(...)
}
```

Sequential async calls

API is more concise, expressive, & composable, especially for async code.



Interactive console with full classpath of your app



Play is Scala at its core: main libraries, routes, templates, SBT, etc [7]

Play/Scala: the less good parts

Simple Build Tool: Play's build system. Very powerful, but *very* steep learning curve.

```
val settings = Seq(
  bootPath <<= target( _ / "boot" ),
  buildDir <<= baseDirectory(_ / ".." / "build"),
  srcDir <<= baseDirectory(_/ ".." / "src"),
  enableConfigCompile := true,
  fabrics <++= Seq("EI", "EI2")
)
```

project/Build.scala

Abandon all hope, ye who enter here without knowing Scala.

```
fabrics <++= Seq("EI", "EI2")
```

project/Build.scala

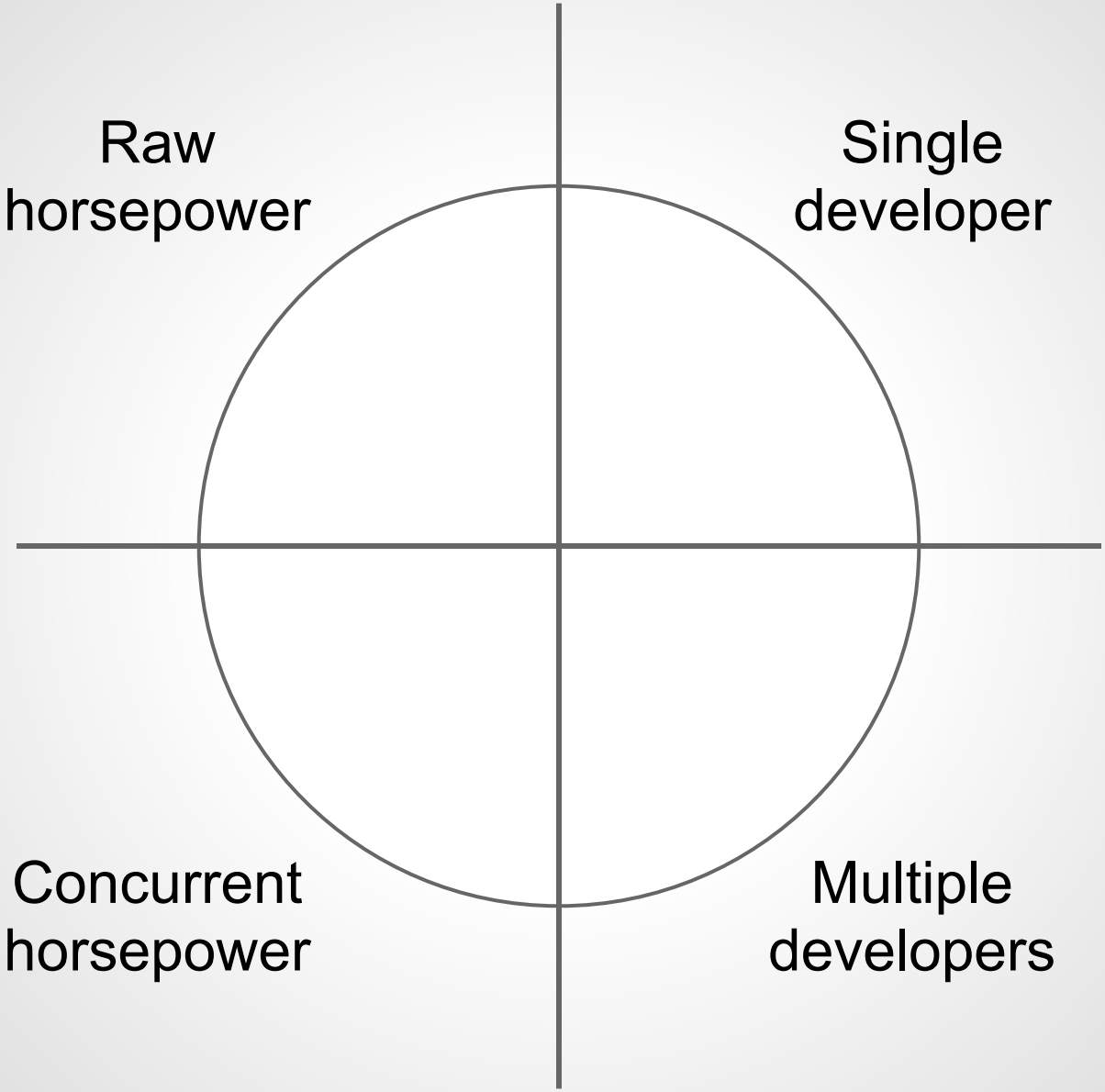
WTF is <++=? Some kind of fish bones?
How do I google that?

Outline

1. Getting started with Play
2. Make a change and reload
3. Error handling
4. Threaded vs. evented
5. Non-blocking I/O
6. Scala
7. **Performance**
8. Community

Is Play web scale? [8]

Scalability can be measured along many dimensions


Raw
horsepower
Single
developer
Concurrent
horsepower
Multiple
developers

Raw horsepower: theoretical maximum performance for the server in ideal conditions. A measure of language and framework overhead.

Concurrent horsepower: performance with many users, I/O, and more real-world scenarios. A measure of the framework's approach to concurrency.

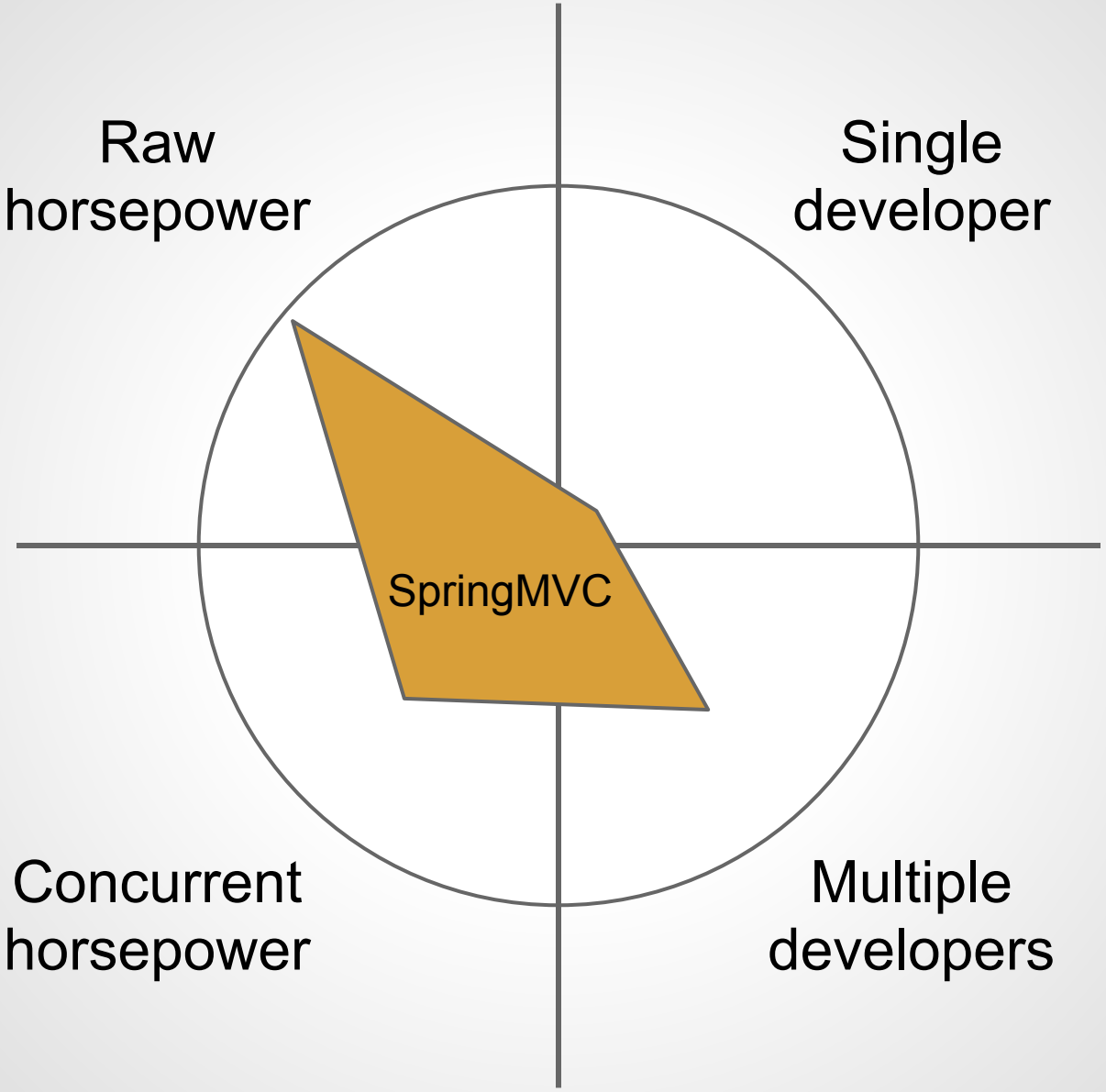
Single developer: how easy it is to get started, how quickly a single developer can build things. A measure of the framework's raw productivity and tooling.

Multiple developers: how the framework tolerates many developers working concurrently over many years. A measure of code rot and maintainability.

Here is how I'd rate some of the frameworks I've used. YMMV.

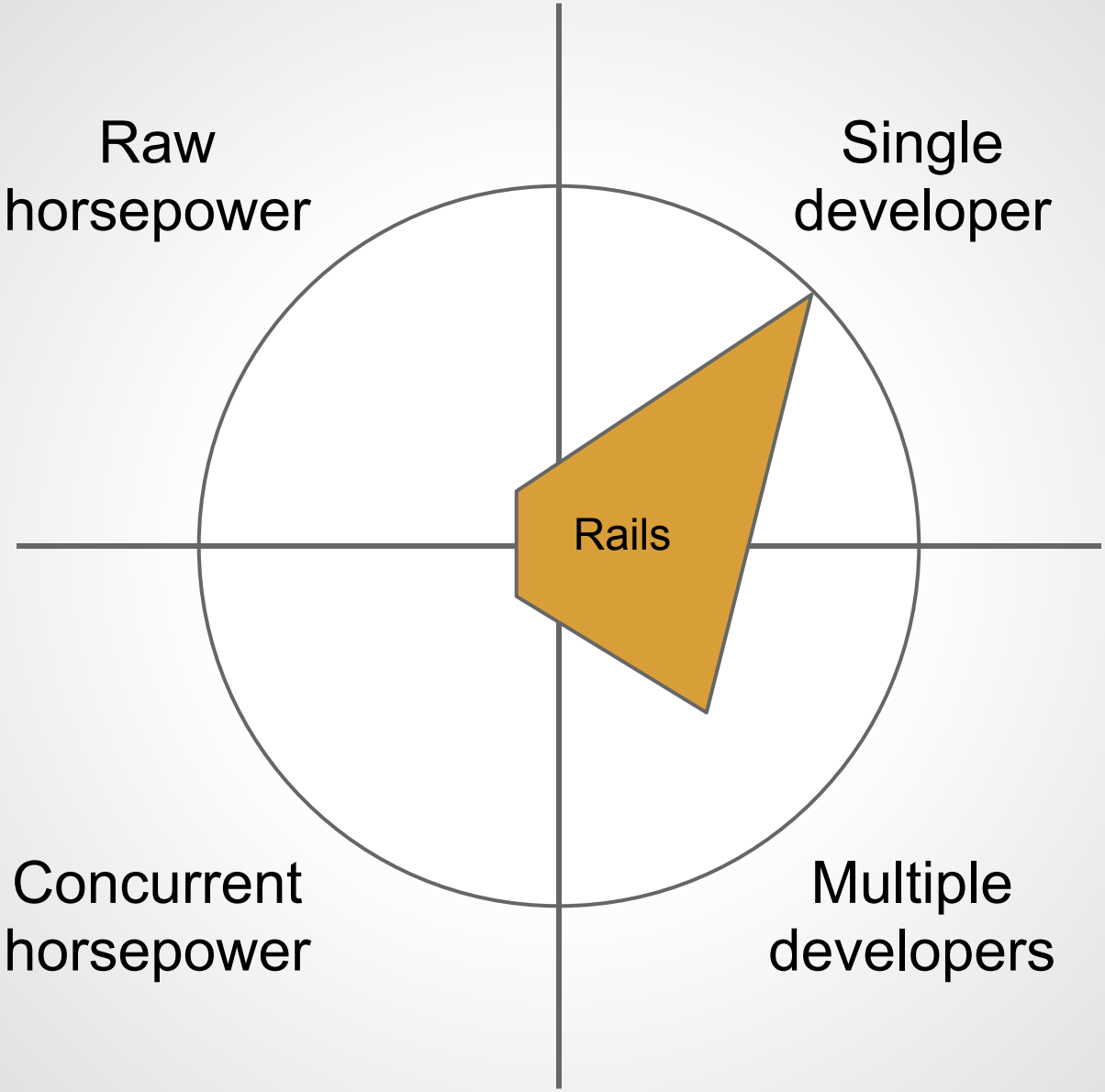
- **Pros**
  - Good raw throughput (qps, latency)
  - Type safety reduces code rot

- **Cons**
  - Not developer friendly. Getting things done takes forever.
  - Threaded, synchronous approach difficult to scale for lots of I/O in a SOA environment.


Raw
horsepower
Single
developer
SpringMVC
Concurrent
horsepower
Multiple
developers

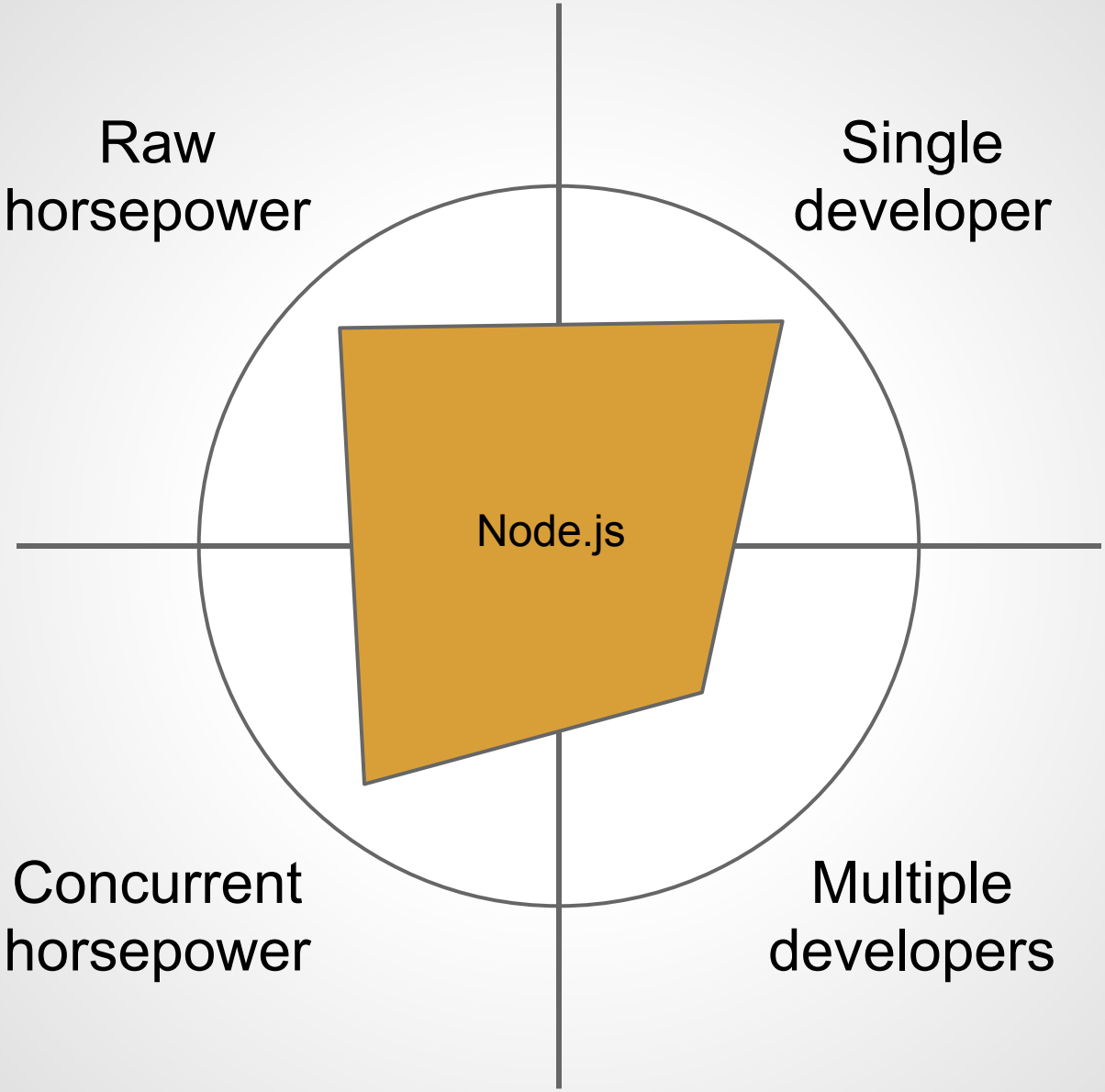
- **Pros**
  - Set the bar for developer productivity; all other frameworks are still trying to catch up.

- **Cons**
  - Ruby is slow
  - Ruby doesn't have real multithreading nor a great evented framework
  - Dynamic language makes it tougher to maintain a large codebase


Raw
horsepower
Single
developer
Rails
Concurrent
horsepower
Multiple
developers

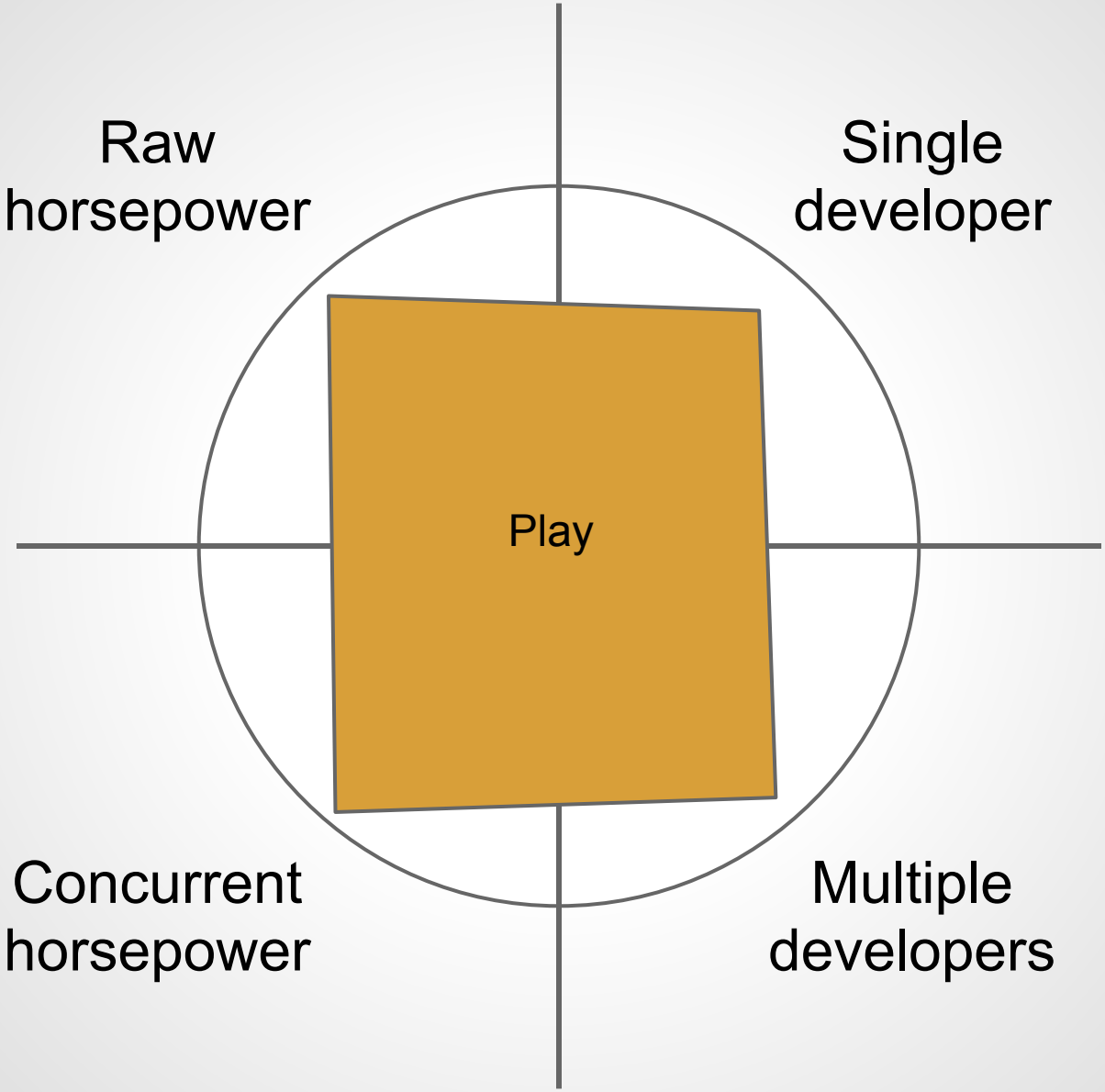
- **Pros**
  - v8 engine is pretty fast for raw throughput
  - Non-blocking I/O at the core makes concurrency easy
  - Strong open source community and lightning fast startup time makes it easy to get things done quickly

- **Cons**
  - Dynamic language makes it tougher to maintain a large codebase
  - Lots of immature libraries that constantly make backwards incompatible changes


Raw
horsepower
Single
developer
Node.js
Concurrent
horsepower
Multiple
developers

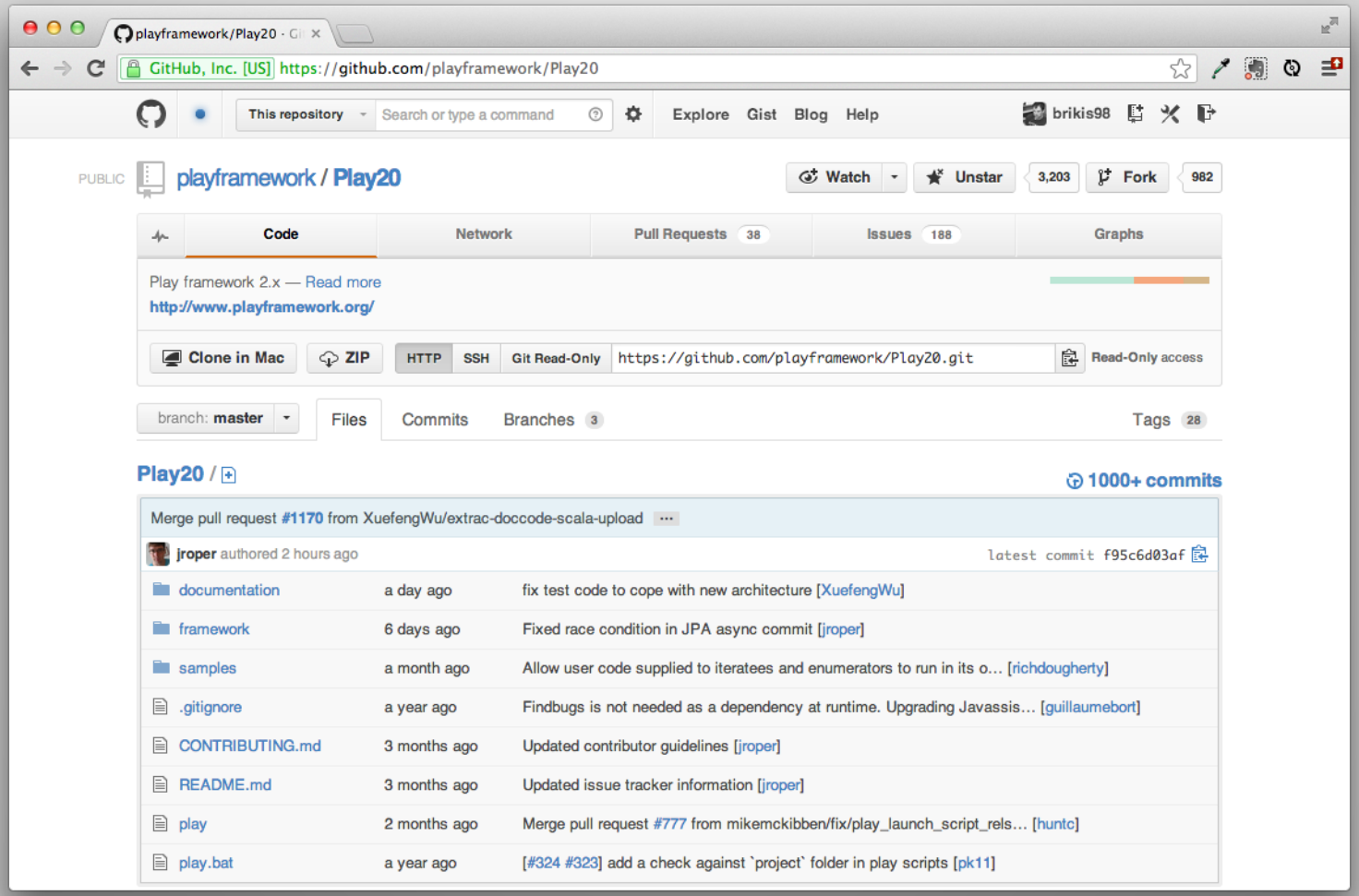
- **Pros**
  - Fast for raw throughput
  - Non-blocking I/O at the core makes concurrency easy
  - Hot reload makes it possible to get things done quickly
  - Strong type safety throughout reduces code rot

- **Cons**
  - Even with hot reload, a compiled statically typed language isn't *quite* as fast as an interpreted dynamically typed language

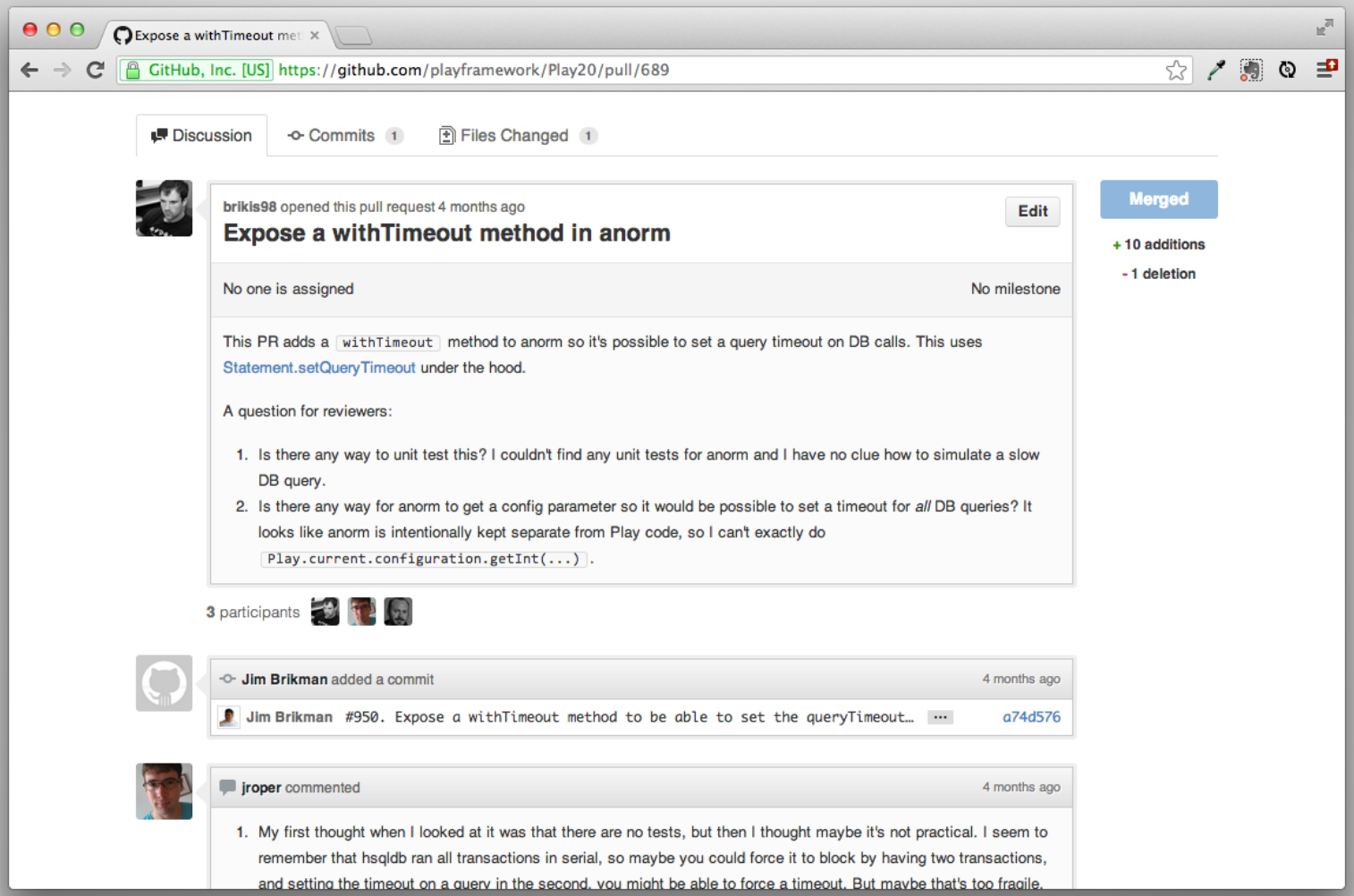

Raw
horsepower
Single
developer
Play
Concurrent
horsepower
Multiple
developers

Outline

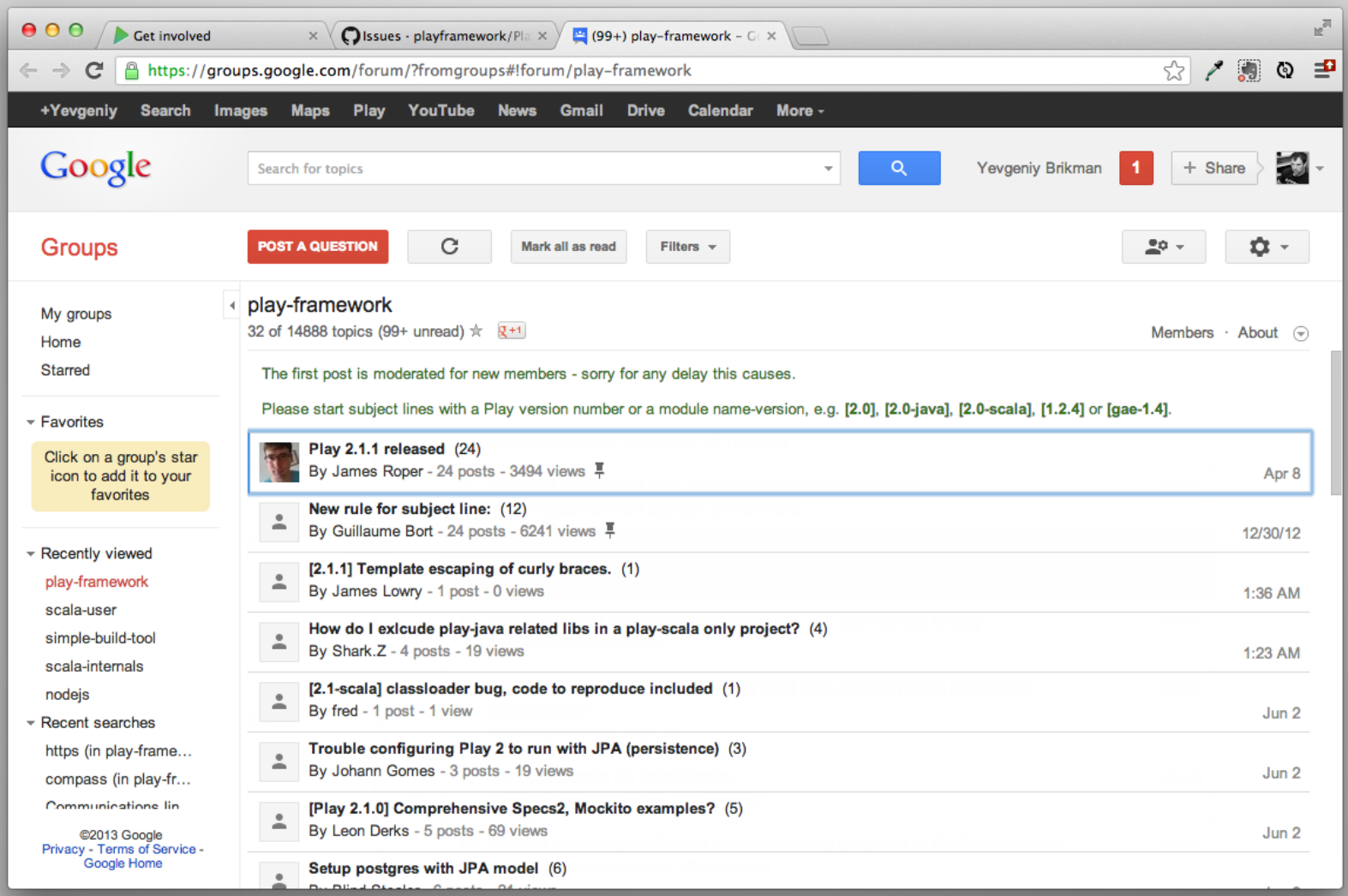
1. Getting started with Play
2. Make a change and reload
3. Error handling
4. Threaded vs. evented
5. Non-blocking I/O
6. Scala
7. Performance
8. **Community**



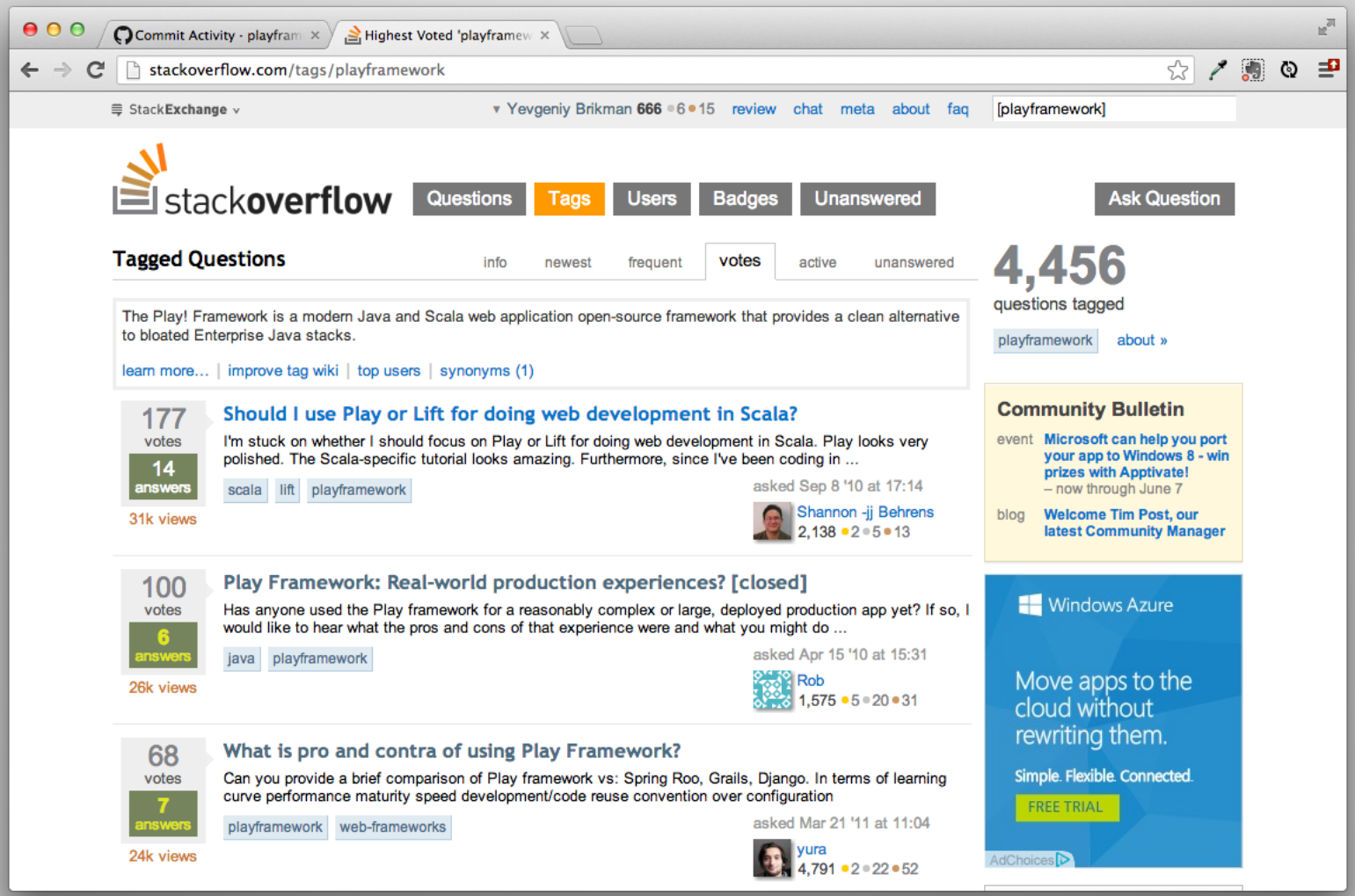
Play is open source [7]



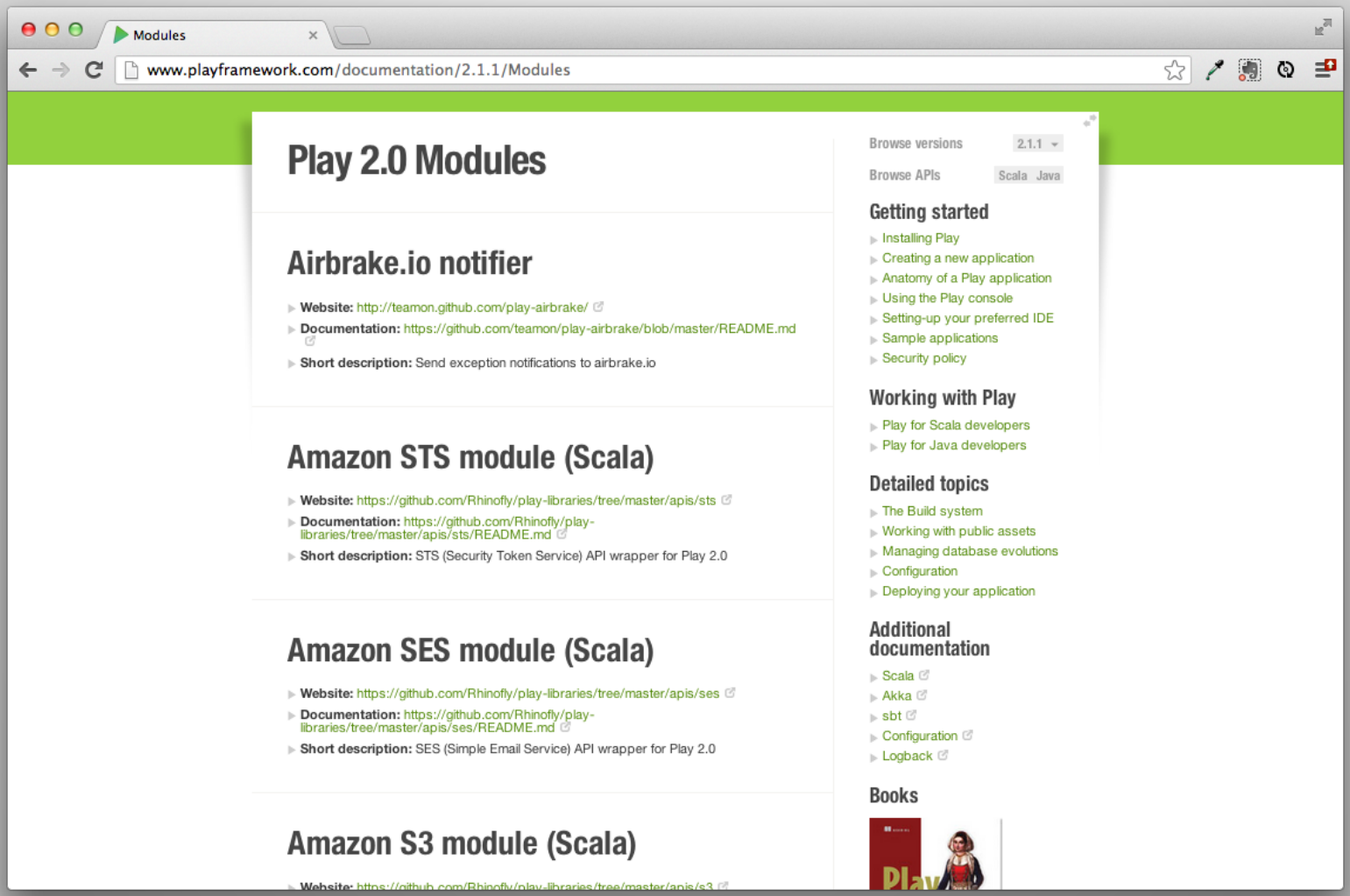
LinkedIn is contributing



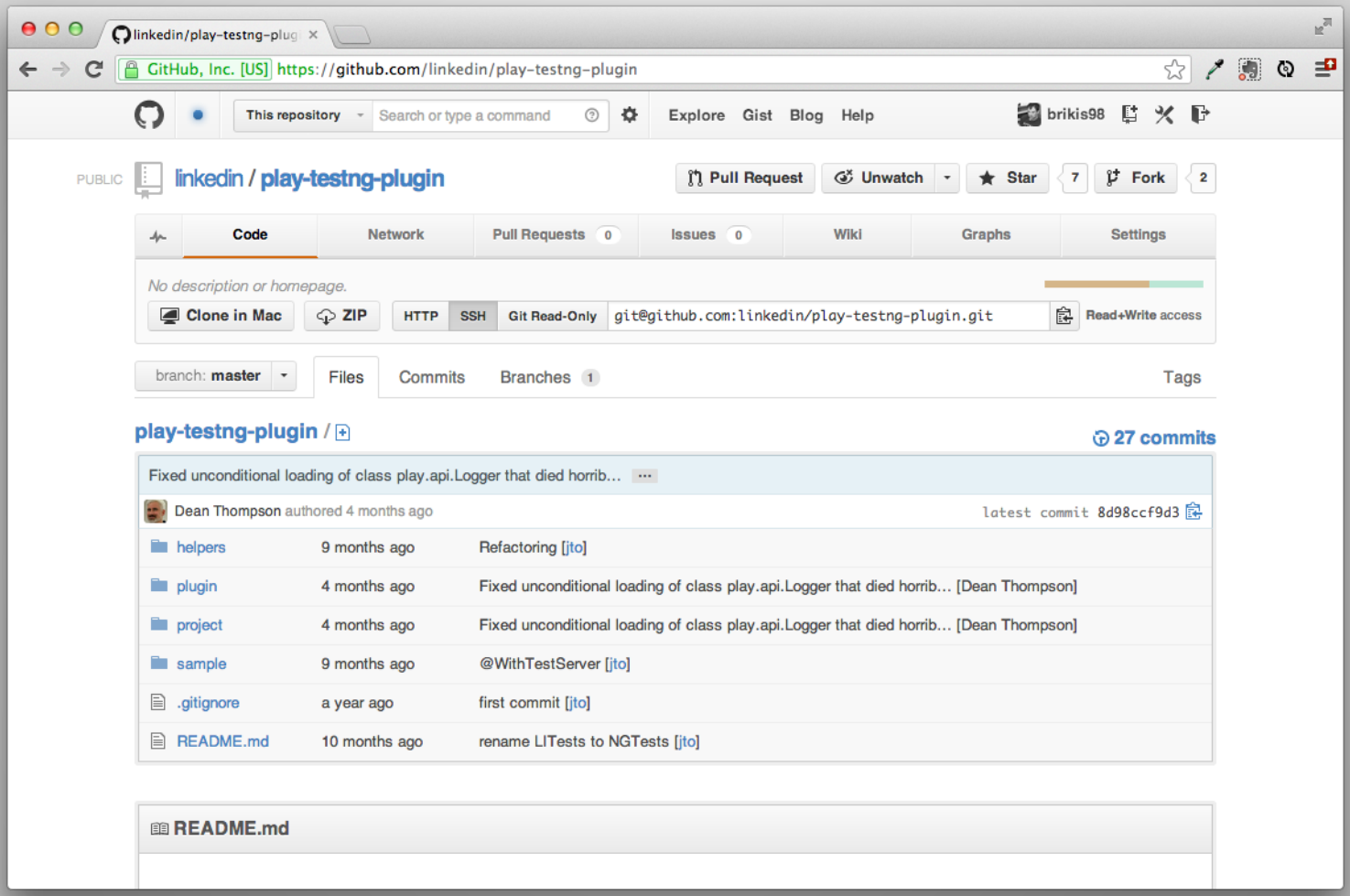
Very active google group (mailing list) [9]



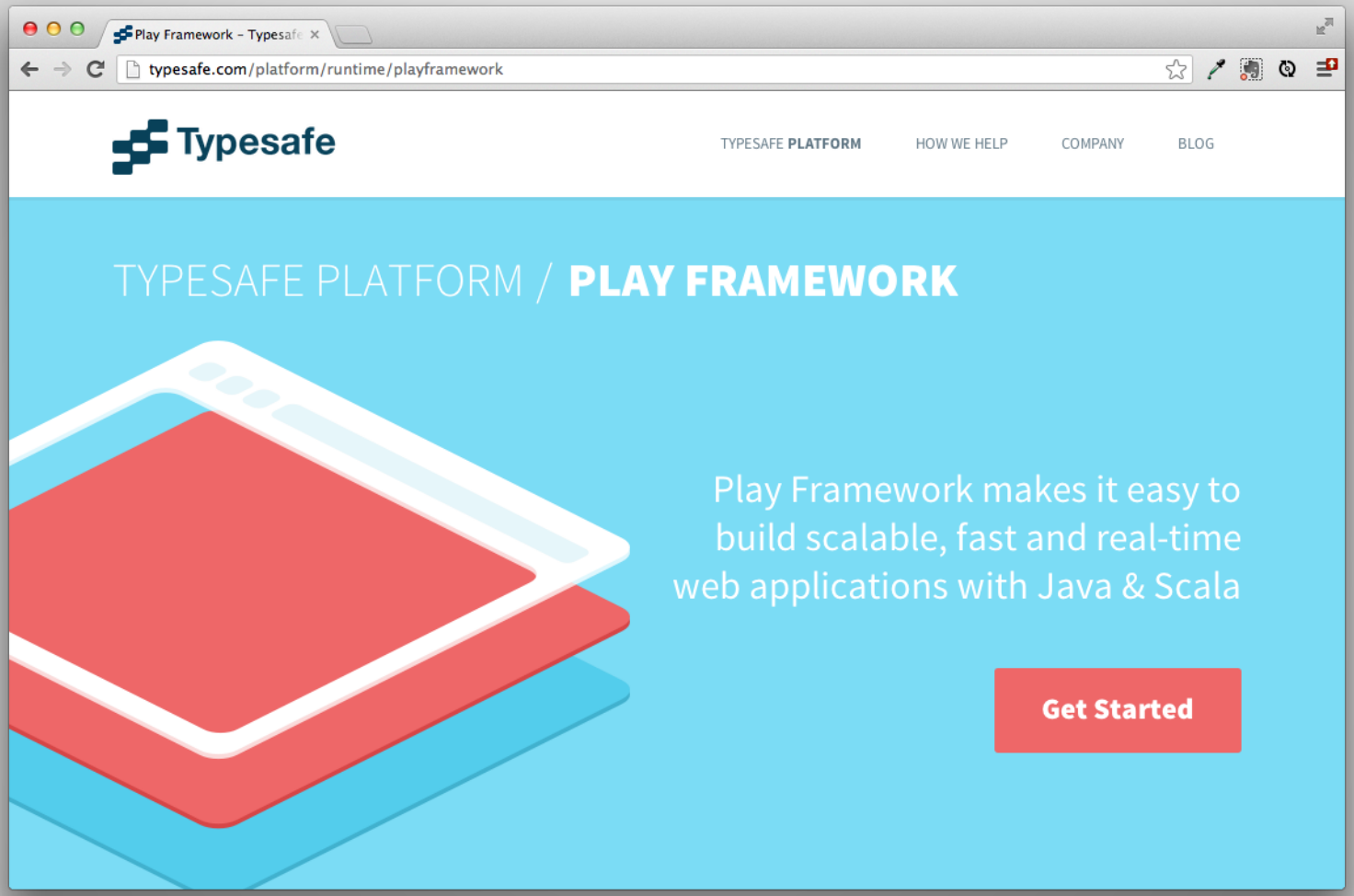
StackOverflow tag [10]



Open source modules and plugins [11]



We've open sourced a few of our plugins, with many more on the way [12]



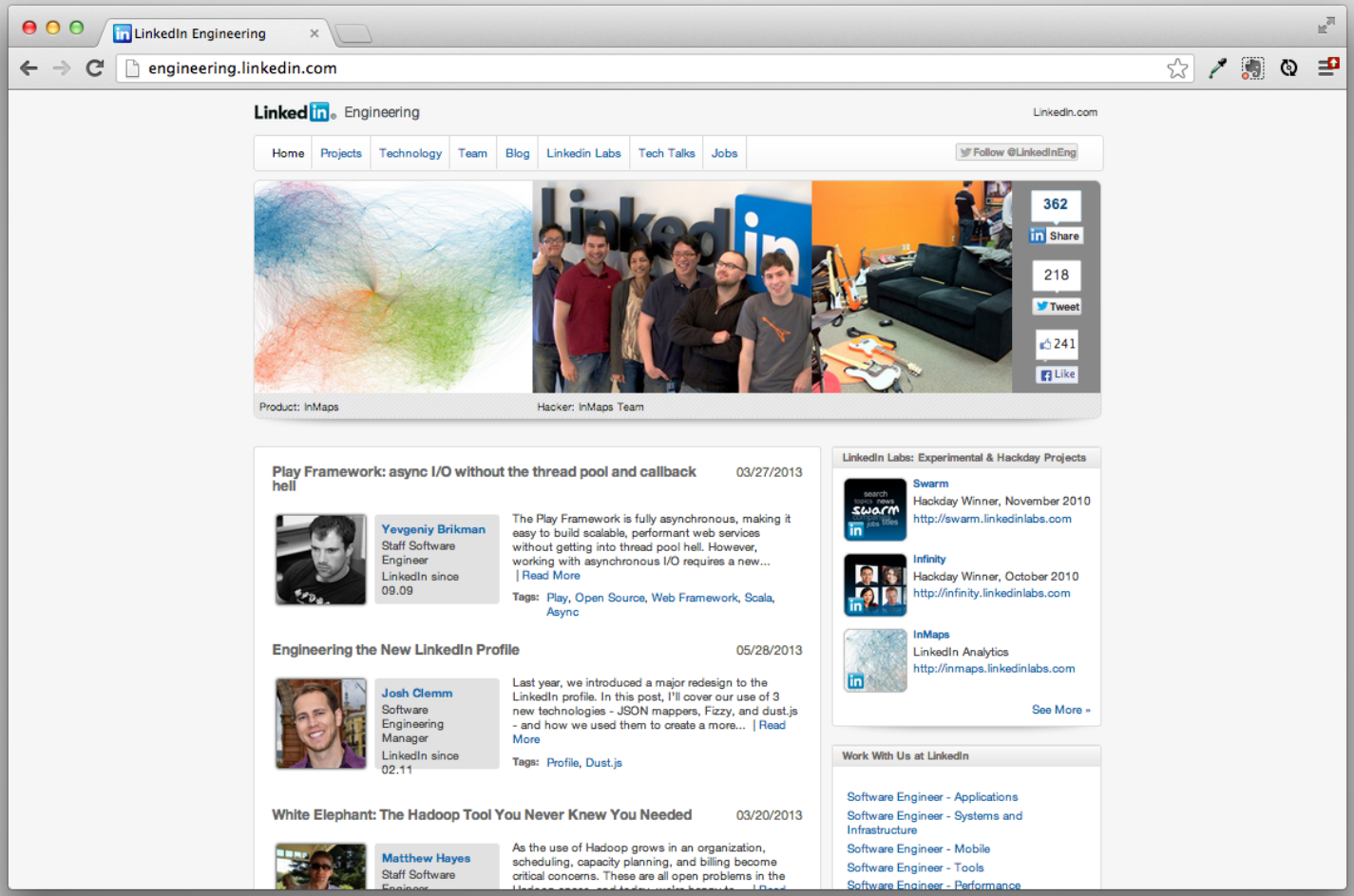
Commercial support from Typesafe [13]

Recap

1. Getting started with Play
2. Make a change and reload
3. Error handling
4. Threaded vs. evented
5. Non-blocking I/O
6. Scala
7. Performance
8. Community

That's the story of Play at LinkedIn so far...

But we're just getting started.



LinkedIn Engineering Blog [14]

@LinkedInEng on twitter [15]

Thank you!

References

1. http://zeroturnaround.com/java-ee-productivity-report-2011/
2. http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/...
3. http://engineering.linkedin.com/play/play-framework-linkedin
4. http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html
5. http://engineering.linkedin.com/play/play-framework-async-io-without...
6. http://netty.io/
7. https://github.com/playframework/Play20
8. http://mongodb-is-web-scale.com/
9. https://groups.google.com/forum/?fromgroups#!forum/play-framework
10. http://stackoverflow.com/tags/playframework
11. http://www.playframework.com/documentation/2.1.1/Modules
12. https://github.com/linkedin
13. http://typesafe.com/platform/runtime/playframework
14. http://engineering.linkedin.com/
15. https://twitter.com/LinkedInEng