

**ORACLE®**

ORACLE®

# Server-Side JavaScript auf der JVM

Peter Doschkinow  
Senior Java Architect

MAKE THE  
FUTURE  
JAVA

ORACLE®

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

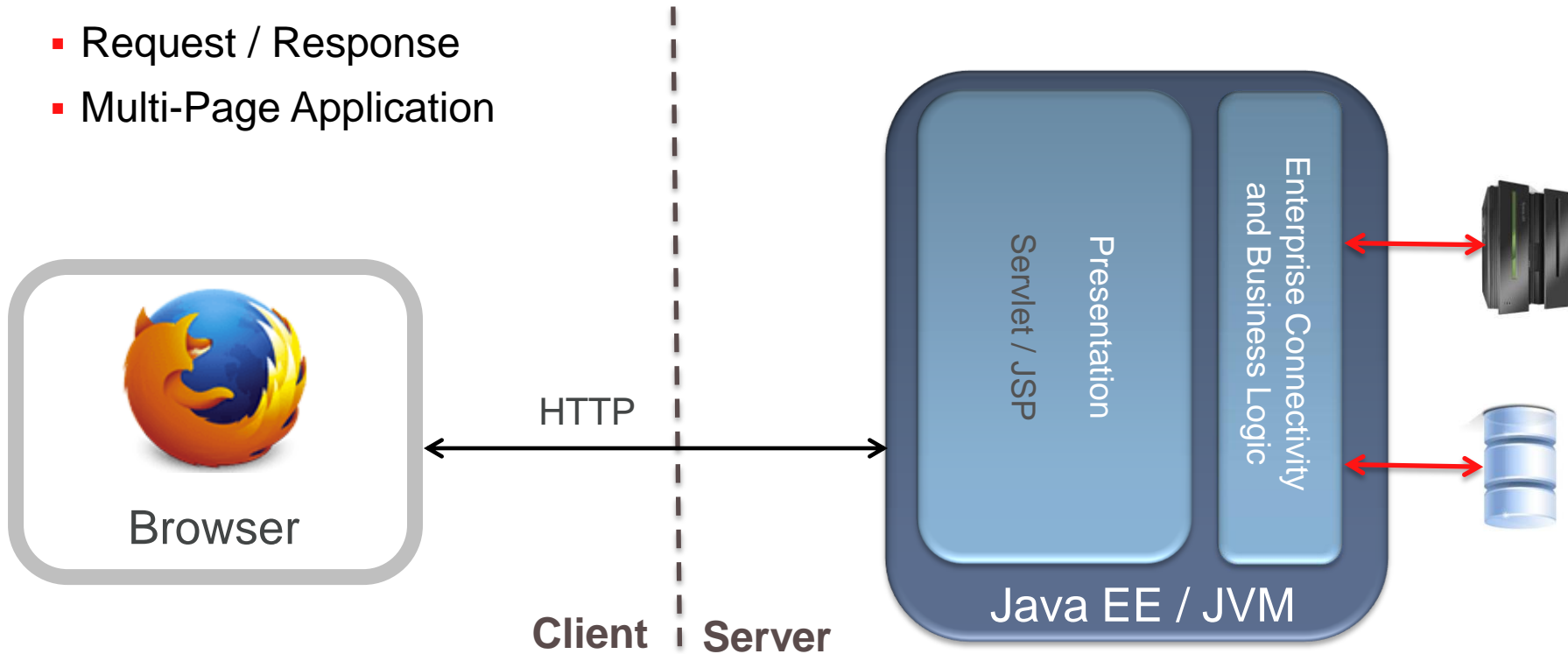
# Agenda

- Web Application Architecture
- JavaScript and Node.js on the JVM
- Project Avatar – Advanced JavaScript Services
- Avatar Client Framework
- Summary

# Evolution of Web Application Architecture

## A Java EE Perspective

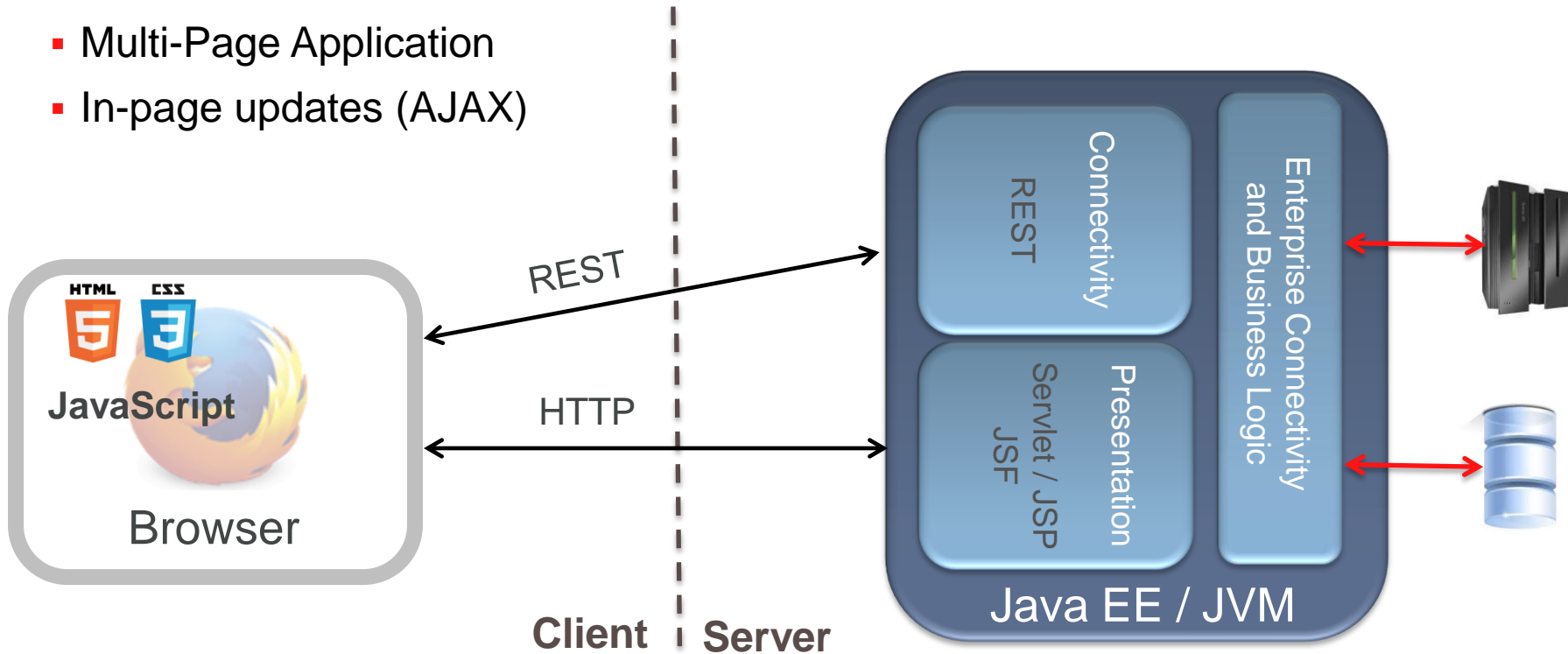
- Request / Response
- Multi-Page Application



# Evolution of Web Application Architecture

## A Java EE Perspective

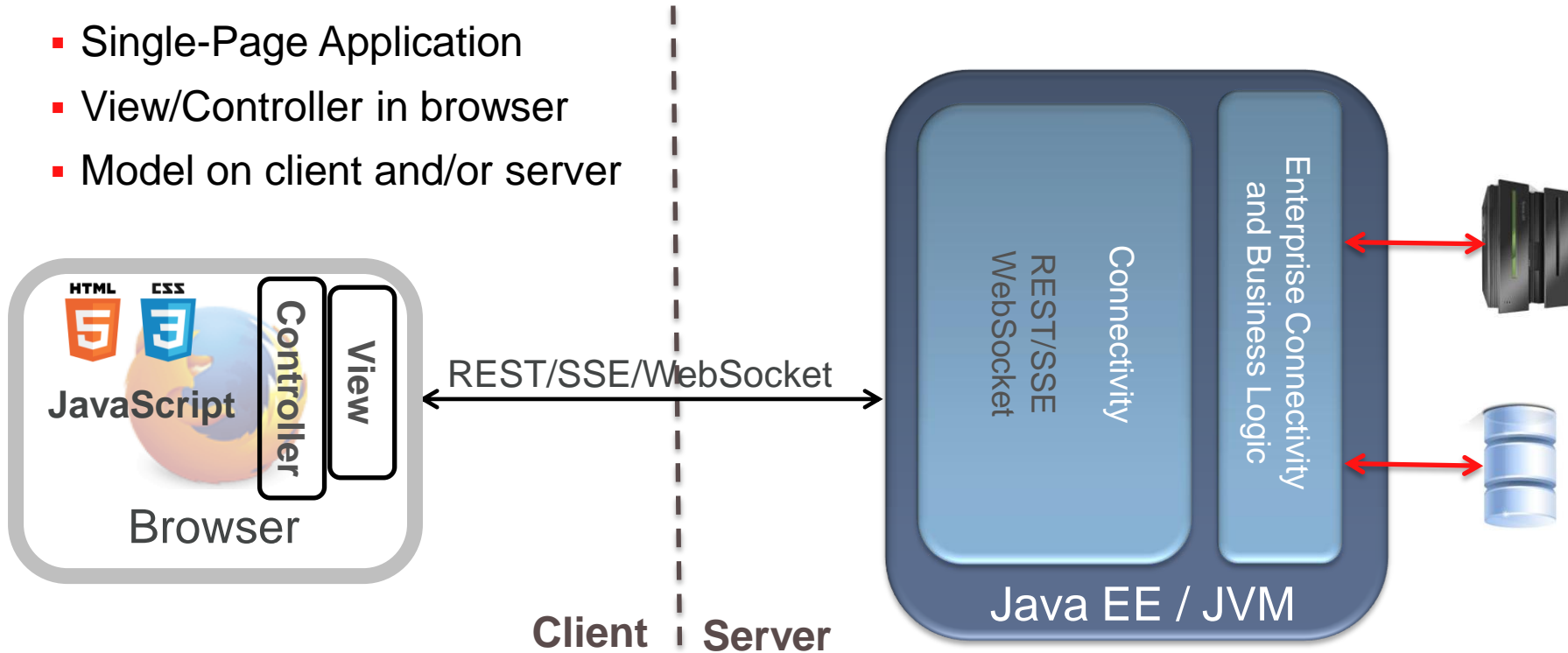
- Multi-Page Application
- In-page updates (AJAX)



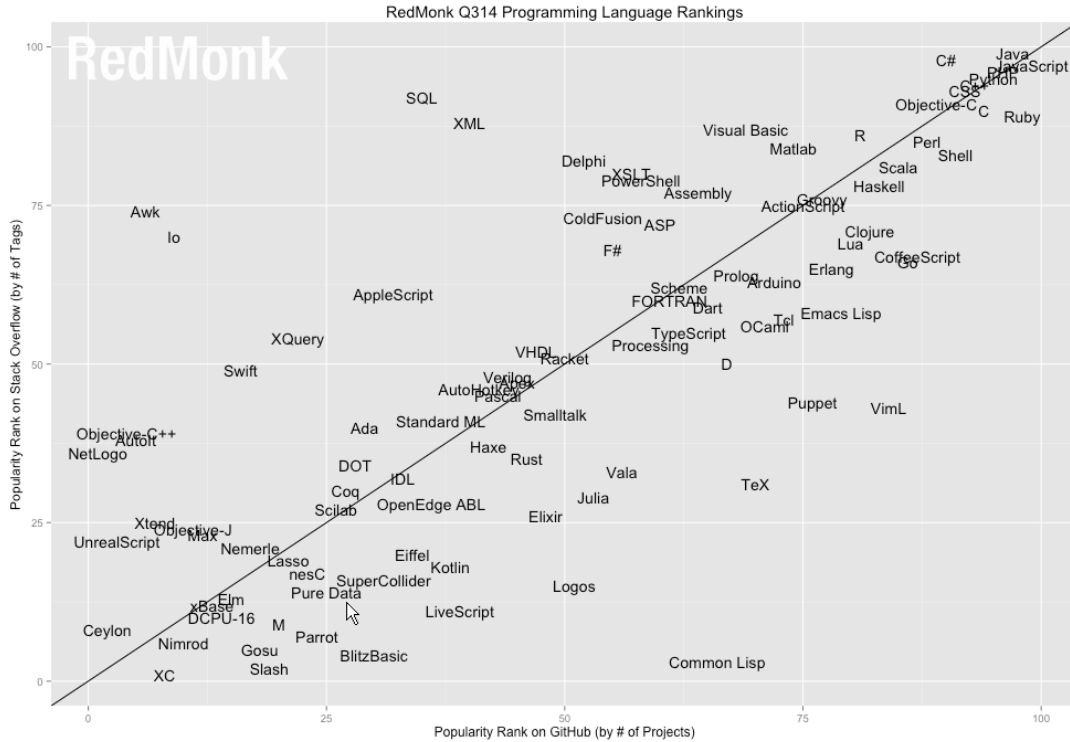
# Modern Web Application Architecture

## A Java EE Perspective

- Single-Page Application
- View/Controller in browser
- Model on client and/or server



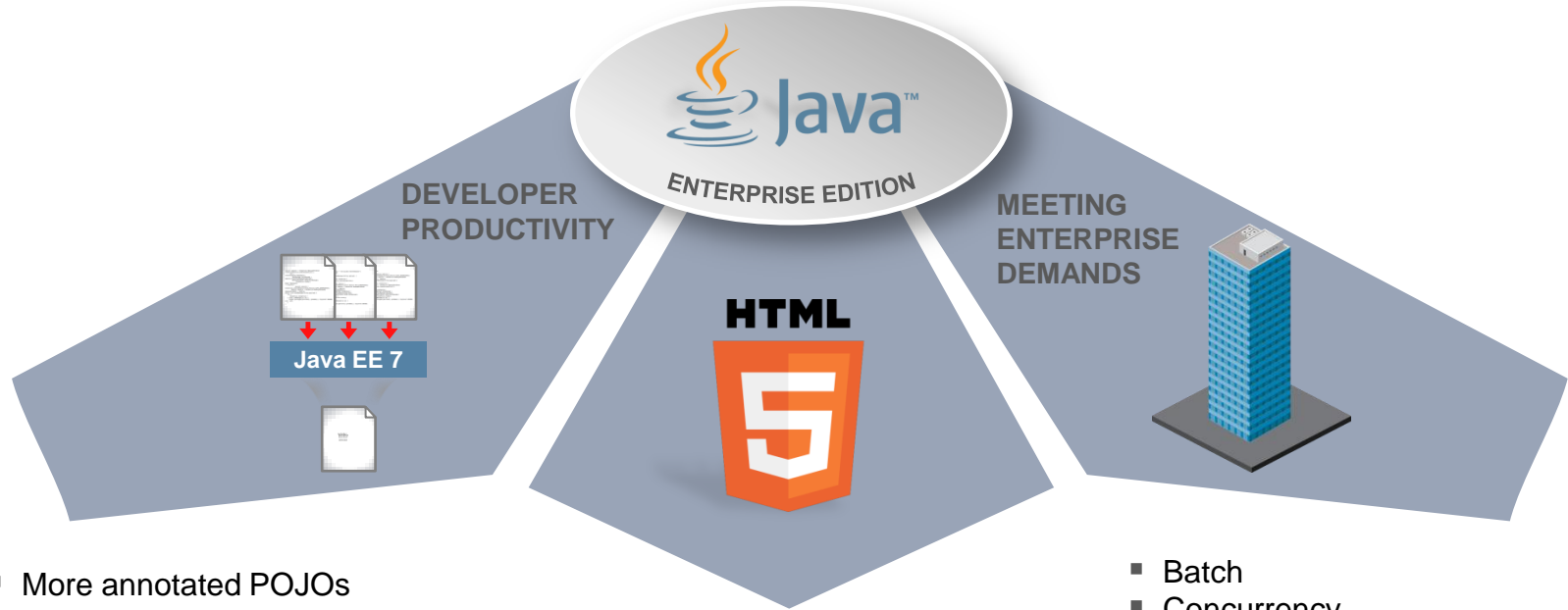
# The Rise of JavaScript



<http://redmonk.com/sograzy/2014/06/13/language-rankings-6-14/>



# Java EE 7 – The Latest in Enterprise Java



- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

- Batch
- Concurrency
- Simplified JMS

# Node.js

<http://www.nodejs.org>



- Platform built on Chrome's JavaScript runtime V8 for easily building fast, scalable network applications (Ryan Dahl , 2009)
  - perfect for DIRTy(Data Intensive Real-Time) apps
- Uses event-driven non-blocking I/O model
  - The async programming model is harder to develop to, but it allows scalability and high levels of concurrency
- Melting pot community
  - Java, .NET, Browser, PHP, etc ...
  - Very successful, second-most-watched project on GitHub with 60,000+ modules

# Node.js Programming Model

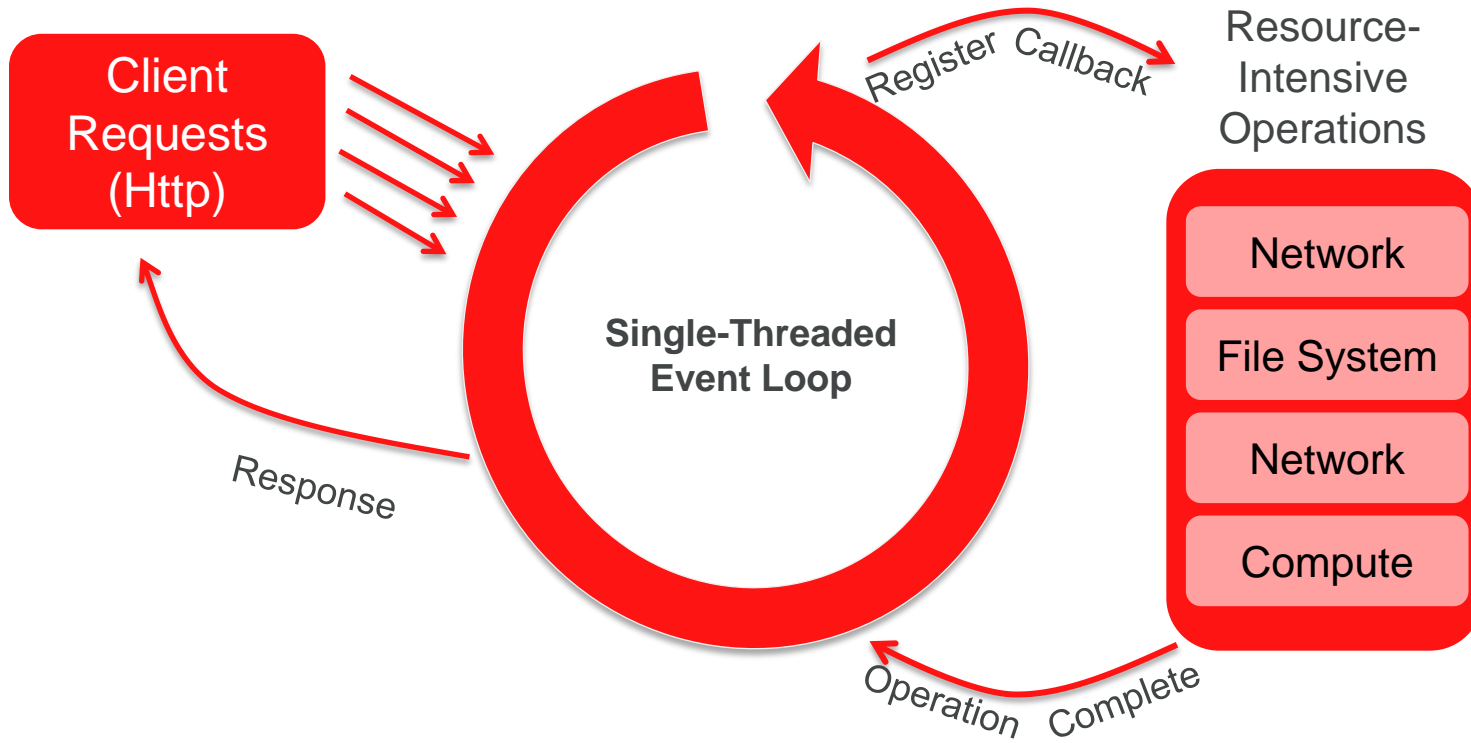
- Multi-threading is hard
  - Thousands of concurrent connections
  - Deal with deadlocks and race conditions
- Blocking on I/O is bad
- Single threaded Event-loop
  - Callback model
  - Non-blocking I/O calls
  - Heavily parallelized

## Minimal Web Server Example :

```
var http = require("http");

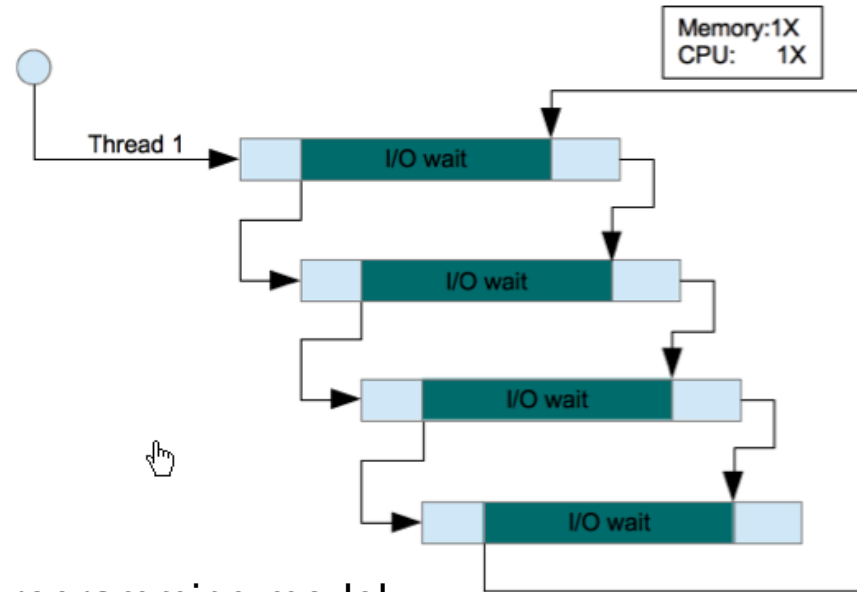
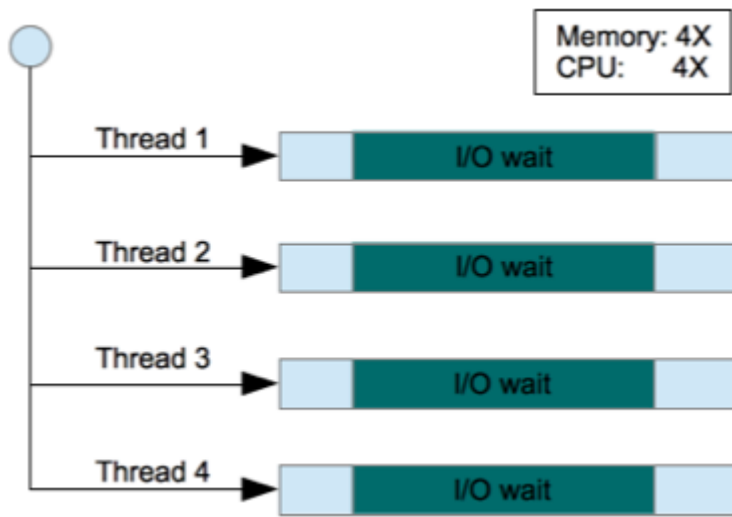
http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8080);
```

# Node.js Event Loop



# Ressource Utilization: sync vs. async I/O

<http://bijoor.me/2013/06/09/java-ee-threads-vs-node-js-which-is-better-for-concurrent-data-processing-operations/>

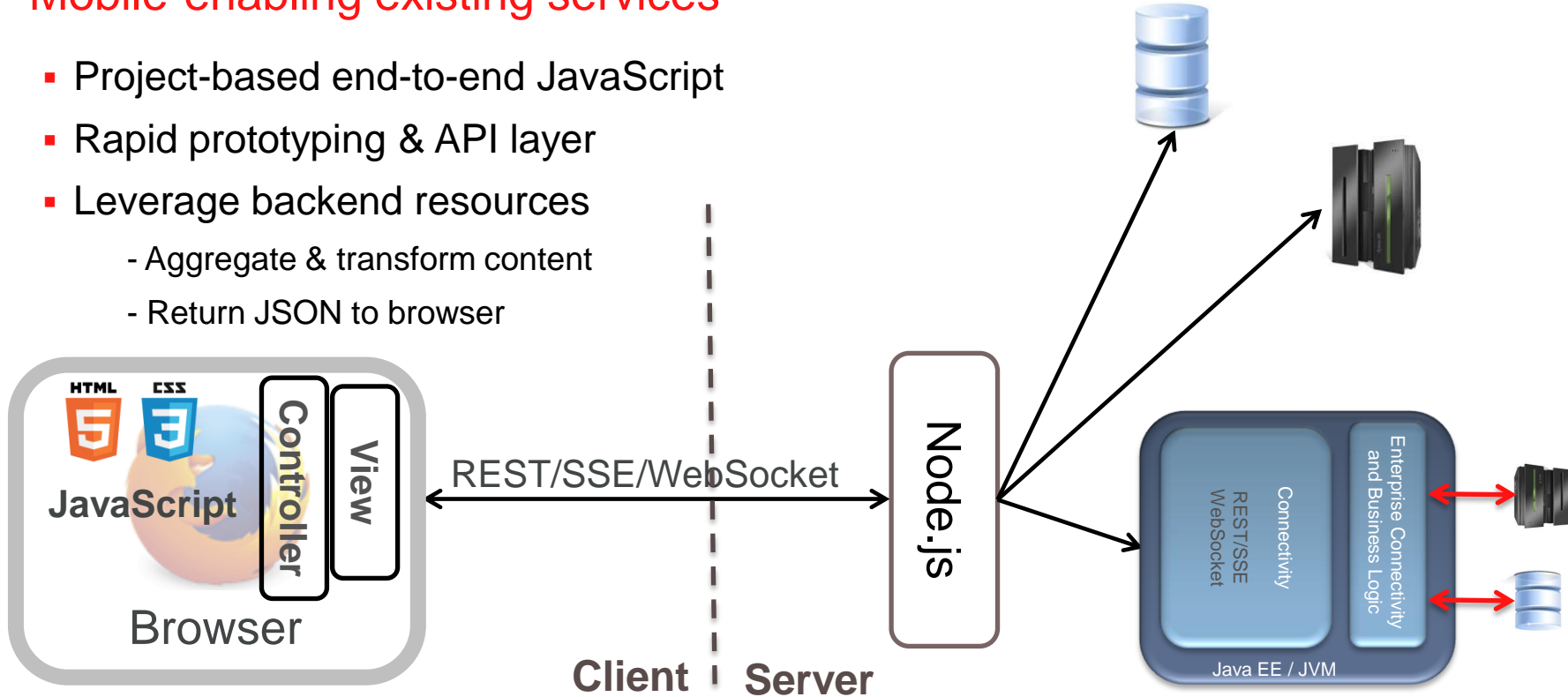


- Node.js, Vert.x are based on an async programming model
- Java EE introduces many new async API
  - Servlet, EJB, JAX-RS, Concurrency for Java EE, ...

# Evolution of Web Application Architecture

## Mobile-enabling existing services

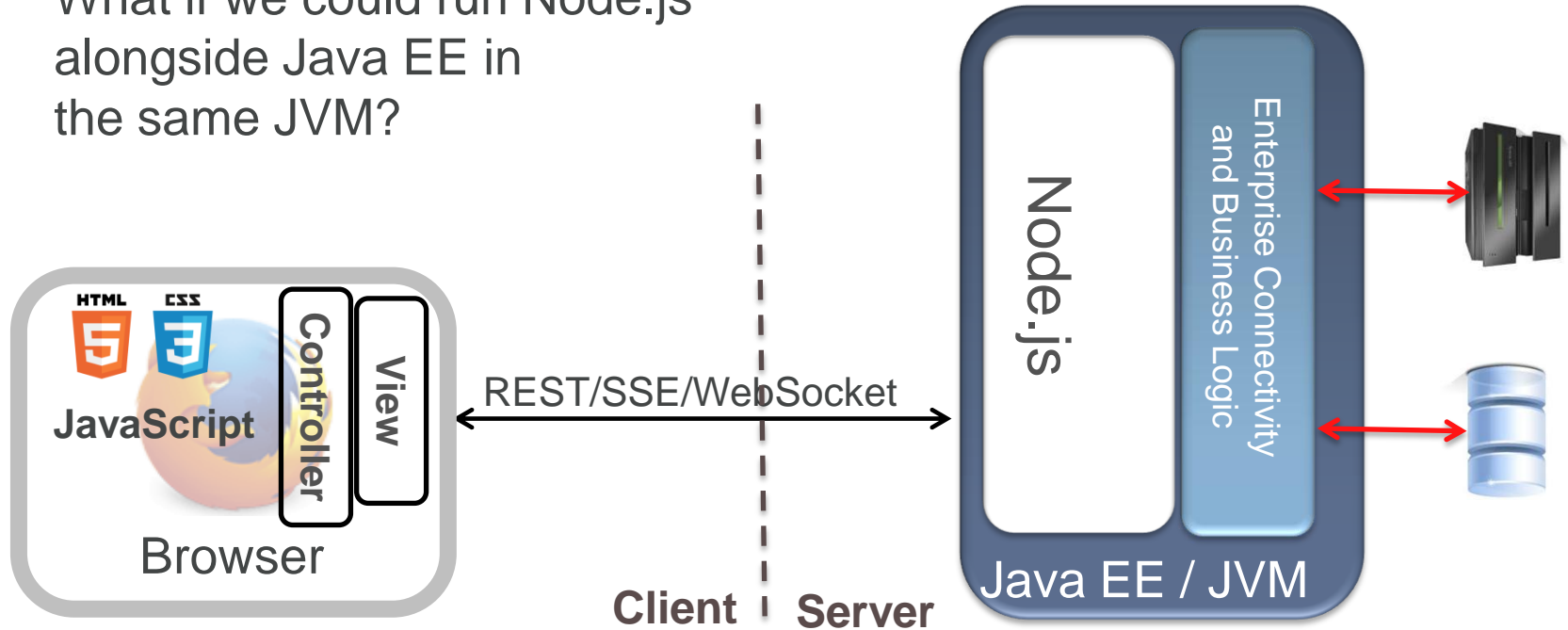
- Project-based end-to-end JavaScript
- Rapid prototyping & API layer
- Leverage backend resources
  - Aggregate & transform content
  - Return JSON to browser



# Evolution of Web Application Architecture

Mobile-enabling existing services

What if we could run Node.js alongside Java EE in the same JVM?

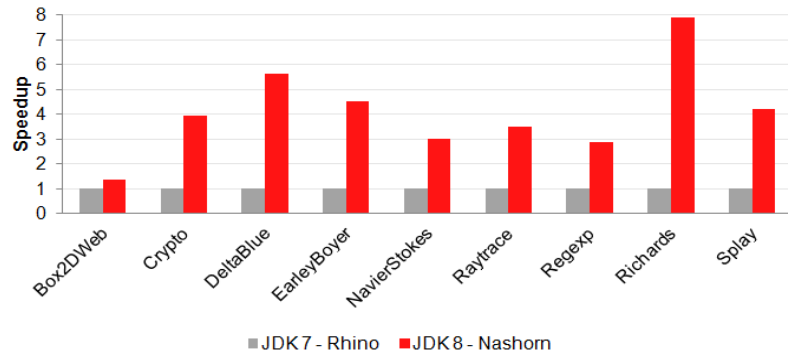


# Project Nashorn

## JavaScript on the JVM

- ECMAScript 5.1 compliant
- Bundled with JDK 8
  - Replaces Rhino in earlier JVMs
  - Faster (2x – 10x)
- New command-line tool `jjs` to run JavaScript
- Seamless Java  $\leftrightarrow$  JavaScript interoperability

<http://download.java.net/jdk8/docs/technotes/guides/scripting/nashorn/index.html>



```
var Button = javafx.scene.control.Button;  
  
var button = new Button();  
button.text = "Say 'Hello World'";  
button.onAction = function() {  
    print("Hello World!");  
}
```



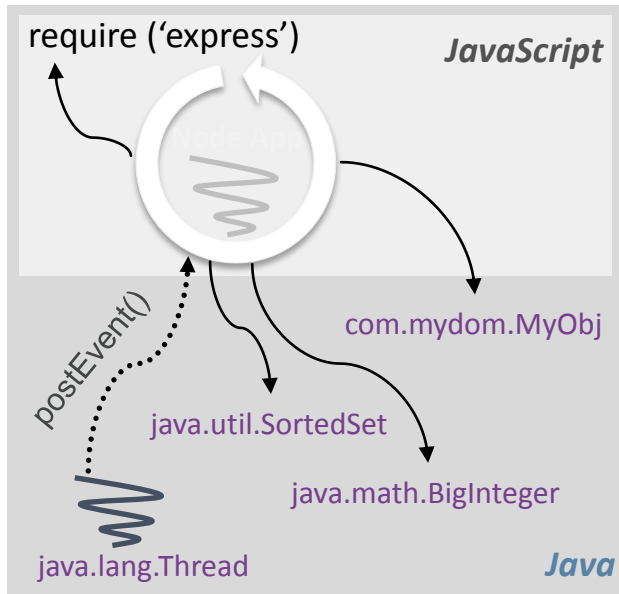
# Avatar.js

## Node.js on the JVM

- Platform for server side JavaScript applications
- Requires Nashorn (JDK 8)
- 95% Node.js compatibility
  - Use popular packages (Express, async, commander, etc)
  - Uses same portability libraries as Node.js
    - Java bindings for libuv and http-parser
  - Limitation: No Chrome v8 native APIs
- Avatar.js Advantages
  - Leverage JVM, Java frameworks and libraries, Security manager

# Avatar.js = Node.js + Java

Leverage Java, including Threads



- Node.js Programming Model
  - Code in JavaScript
  - Single event loop / thread
  - Require (import) Node modules
- Invoke Java code
  - Java types and libraries
  - `new java.lang.Thread();`
  - `new com.mydom.MyObj();`

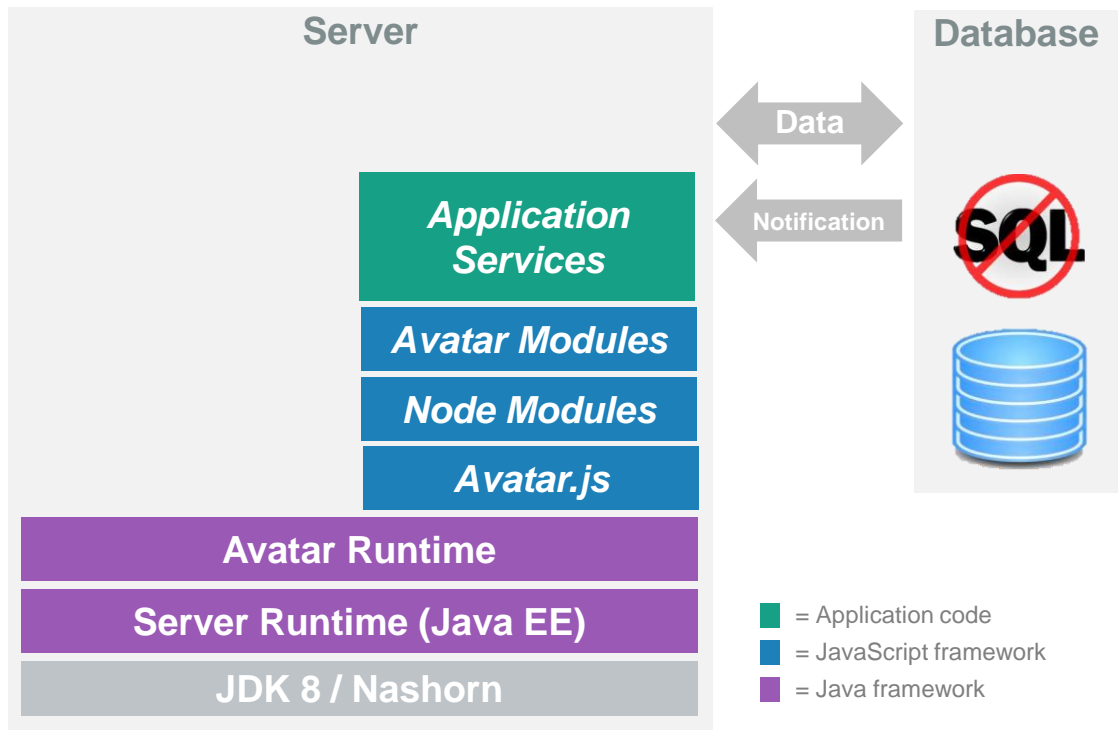
# Project Avatar – the Backend

## A Server Side JavaScript Services Framework

- Similar in spirit to Servlets, but focused on REST, WebSocket, Server Sent Event (SSE) endpoints
- Use familiar Node.js event-driven programming model and modules
- Layers on Avatar.js NodeJS-compatible runtime
- Adds integrated enterprise features

# Avatar Architecture - Server

Server side



# Project Avatar – Backend Features

Leveraging the JVM and Java EE in the Node.js programming model

- Out-of-the-box support for REST, WebSocket, SSE communications
- Multi-threading, lightweight message passing, shared state
- HTTP listener / load-balancer is managed by framework (unlike Node)
- Model Store – Object Relational Mapping
- DataProvider API
  - Simple key-value based collection abstraction
  - FileDataProvider, JPADDataProvider, NoSqlDataProvider
- Messaging integration with JMS on Java EE container
  - Through configuration of SSE- and WebSocket communication types

# WebSocket Service Example

```
// Load avatar module
var avatar = require('org/glassfish/avatar');

// Register service instance
avatar.registerSocketService(
  {url: 'websocket/chat'},
  function() {
    this.data = {transcript : ""};

    this.onMessage = function (peer, message) {
      this.data.transcript += message;
      this.data.transcript += '\n';
      peer.getContext().sendAll(this.data);
    };
  });
```

# WebSocket Service Example

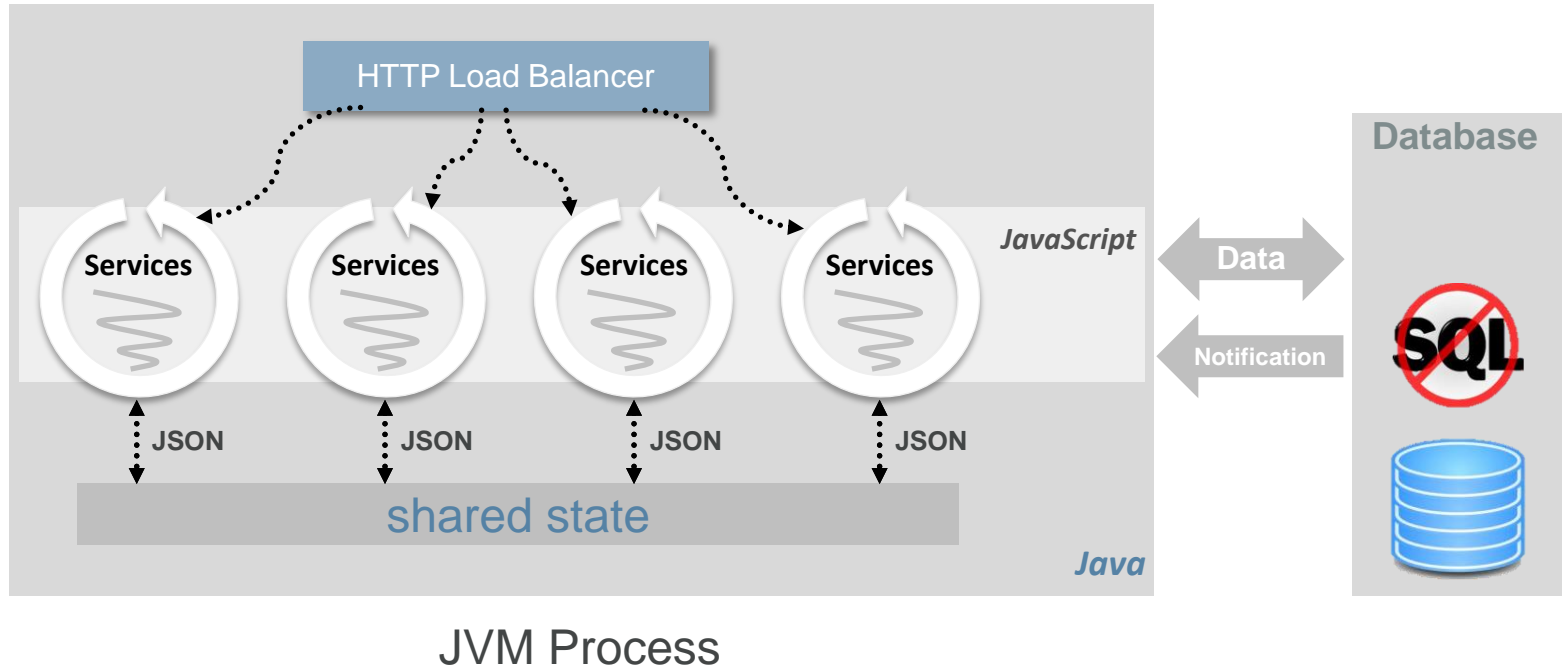
## With JMS integration

```
// Load avatar module
var avatar = require('org/glassfish/avatar');

// Register service instance
avatar.registerSocketService({
  url: "/websockets/jmschat/{chatroom}",
  jms: {
    connectionFactoryName: "jms/AvatarConnectionFactory",
    destinationName: "jms/AvatarTopic",
    messageSelector: "chatroom=#{this.chatroom}",
    messageProperties: {
      chatroom: "#{this.chatroom}"
    }
  }
},
function() { this.onMessage(peer, message) { ... };
```

# Avatar Services Scalability

Multi-core, state sharing, data storage





# Shared State

## Lightweight inter-thread communication

- Two Models
  - MessageBus
    - Publish/subscribe message passing
  - Shared State
    - Simple map API
    - Application-scoped instance
    - Session-scoped instance
      - Named
      - Leased, with configurable timeout
- Provide required serialization, concurrency, and caching

# State Sharing Example

```
var avatar = require('org/glassfish/avatar');
var threads = require('org/glassfish/avatar/threads');
var appname = avatar.application.name;
var bus = avatar.application.bus;

// Listen for messages on the 'hello' topic
bus.on('echo', function(msg) {
  print(appname + ' got ' + msg);
});

// Start a background thread which publishes to the 'echo' topic
new threads.Thread('background', 'monitor.js').start();

// or publish to the same topic in this thread
setTimeout(function() bus.publish('echo', { x : 'x', y : 'y' }), 3000);
```

# Model-Store Framework

- JavaScript ORM library
- Pure JavaScript API that
  - Supports relational and non-relational databases
  - Integration with other Avatar services
- Similar to pure Node.js libraries
  - Sequelize, JugglingDB, Mongoose

# Model-Store API

## Model and Database setup

```
var Product = avatar.newModel({
  "name": {
    type: "string",
    primary: true
  },
  "price": "number",
  "quantity": "integer"
});
```

```
var store = avatar.newStore('mysql', {
  host: 'localhost',
  port: 3306,
  database: 'test',
  username: 'root',
  password: 'gu3ssl1'
  createDb: true,
  dropTables: true
});
```

# Model-Store Example

## Creating and Storing an Object

```
// Binds Product model with store
Product.bind(store);

// Insert a new product into the db
store.connect(function() {
  Product.create({
    name: 'Widget',
    price: 1.00,
    quantity: 2
  }, function(err, w1) {
    console.log(JSON.stringify(w1));
    store.disconnect(function() {
      // done
    });
  });
});
```

- Bind model to data store
- Connect to store
  - Creates Product table if required
  - Callback adds product to table

# Model-Store API

- Models can have relationships with other models
  - 1:1, 1:n, M,N
- Data Stores
  - Relational
    - Tested: Oracle DB, MySQL, Derby (Embedded, Network)
    - Non-tested: Any other JDBC driver
  - Non-relational
    - Oracle NoSQL, MongoDB (in progress)

# Avatar Client

- View
  - Extensible component views
  - Pre-defined Widget Sets: jQuery UI (default), jQuery Mobile, Dijit
  - Declarative UI components
- Model
  - Models (WS, SSE, REST, local) in JavaScript
  - Easily connects to Java and JavaScript services
  - Model library usable as standalone JavaScript file
- Other Highlights
  - Familiar syntax in HTML with “data-” tags
  - Bidirectional Data binding using EL (Expression Language)
  - CSS support
  - Support for AMD modules for code partitioning

# Hello World Example

```
<script data-model="local" data-instance="name">
  var NameModel = function() {
    this.first = "Planet";
    this.last = "Earth";
    this.clear = function() { this.first = this.last = ""; };
  };
</script>
```

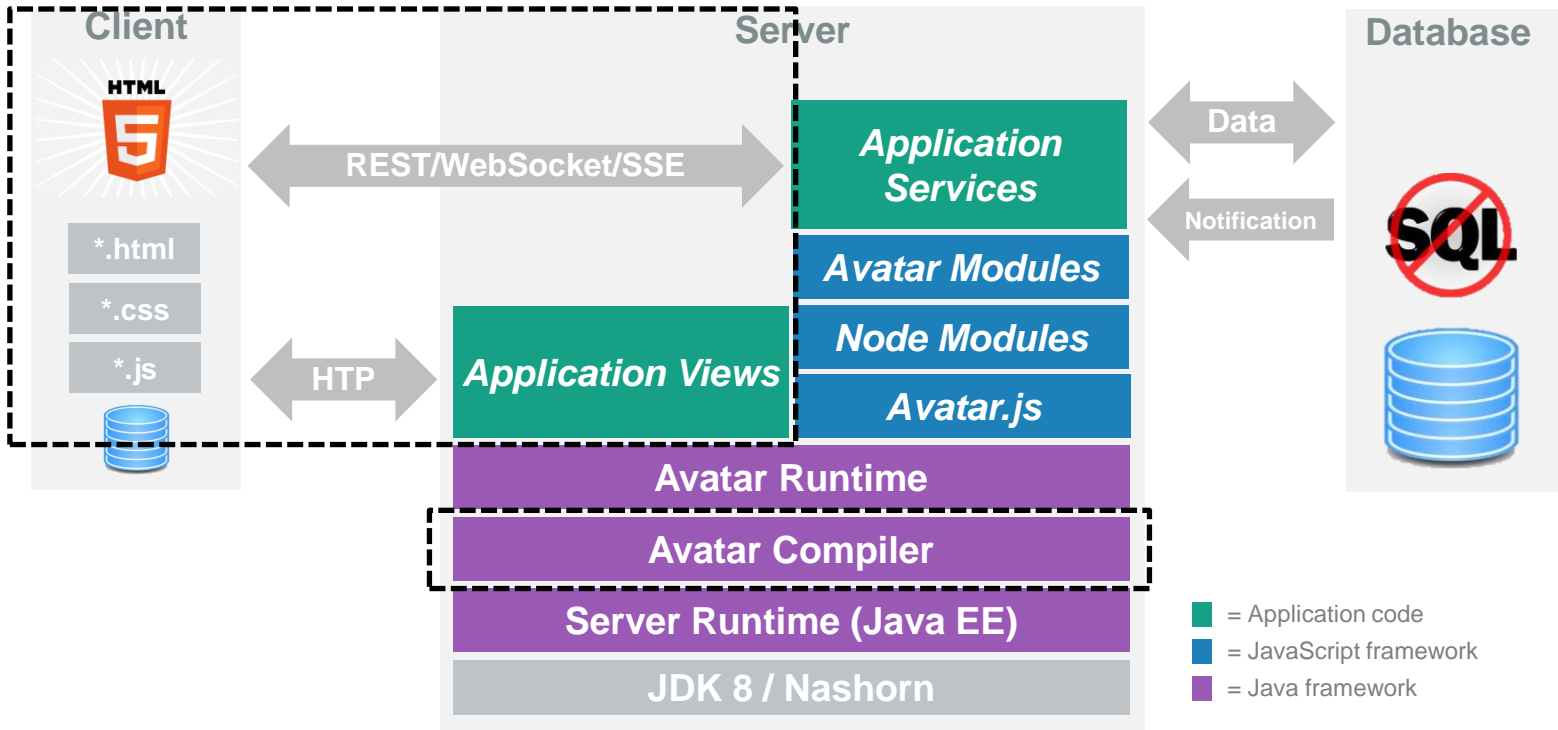
Model

```
<form>
  <label for="first">First Name</label>
  <input id="first" type="text" data-value="#{name.first}"/>
  <label for="last">Last Name</label>
  <input id="last" type="text" data-value="#{name.last}"/>
  Hello #{name.first} #{name.last}
  <button onclick="#{name.clear()}">Clear</button>
</form>
```

View



# Avatar Architecture – Server and Client



# Demo

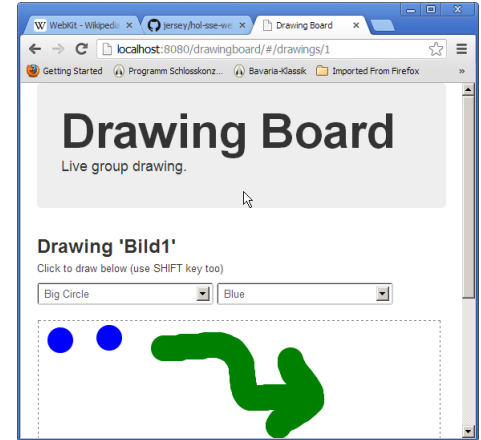
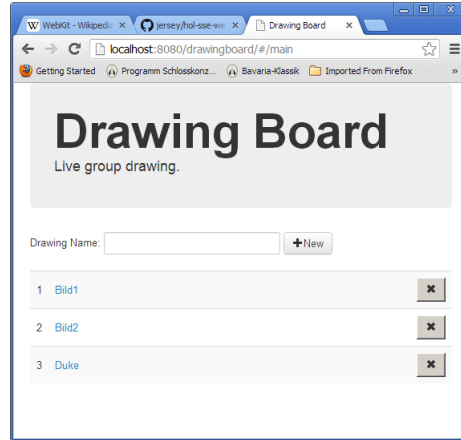
## Porting of a HTML5 Application to Avatar

- Client implementation in AngularJS
- Server implemented with Java EE 7, then ported to use Avatar services
- Focus on the server side
- Demonstrate usage of Avatar Services
  - built-in support for REST/WebSocket/SSE communication patterns
  - Shared state
  - Message bus
- Running on GlassFish 4.x or WebLogic 12.1.3

# Drawing Board HTML5 Demo

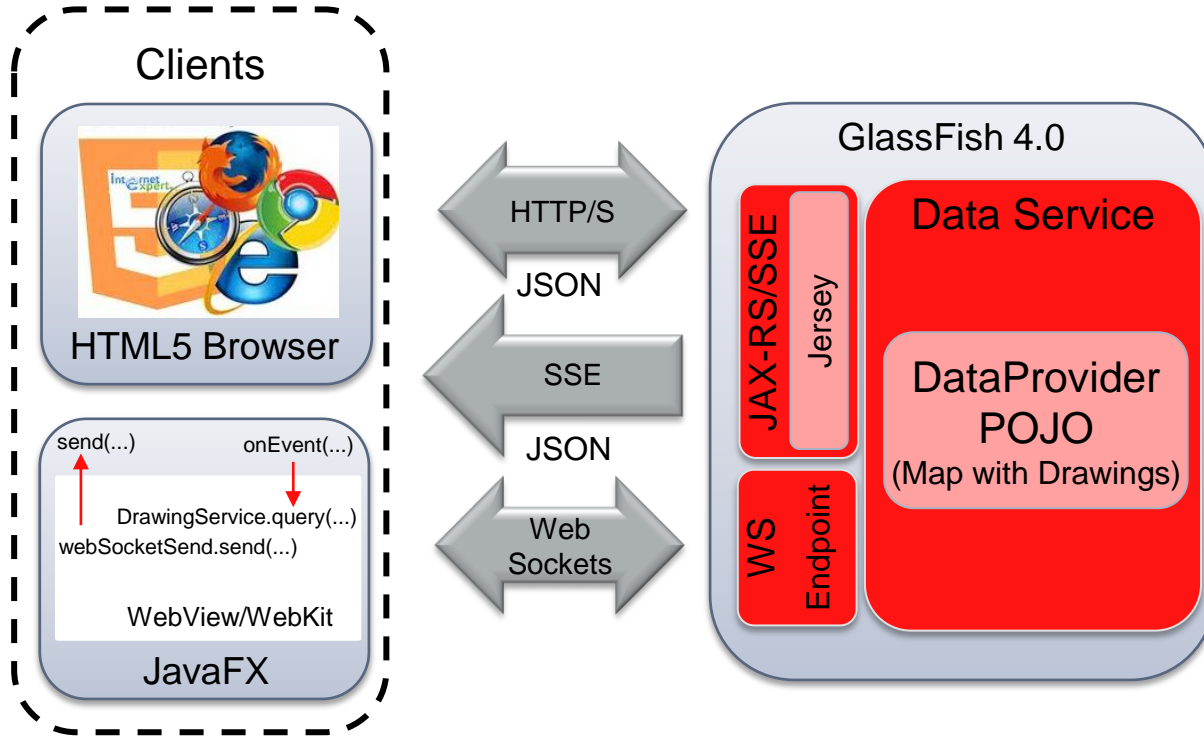
<http://github.com/doschkinow/hol-sse-websocket/solutions/exercise5>

- Collaborative drawing
- Two-page application
  - List of drawings
  - Drawing
- Demonstrating
  - Server-side: JAX-RS, JSON, WebSocket, SSE Java API
  - Client-side: JAX-RS, WebSocket, SSE Java and JavaScript API
  - JavaFX **hybrid** Java/HTML5 application



# Drawing Board HTML5 Demo

## Thin Server Architecture



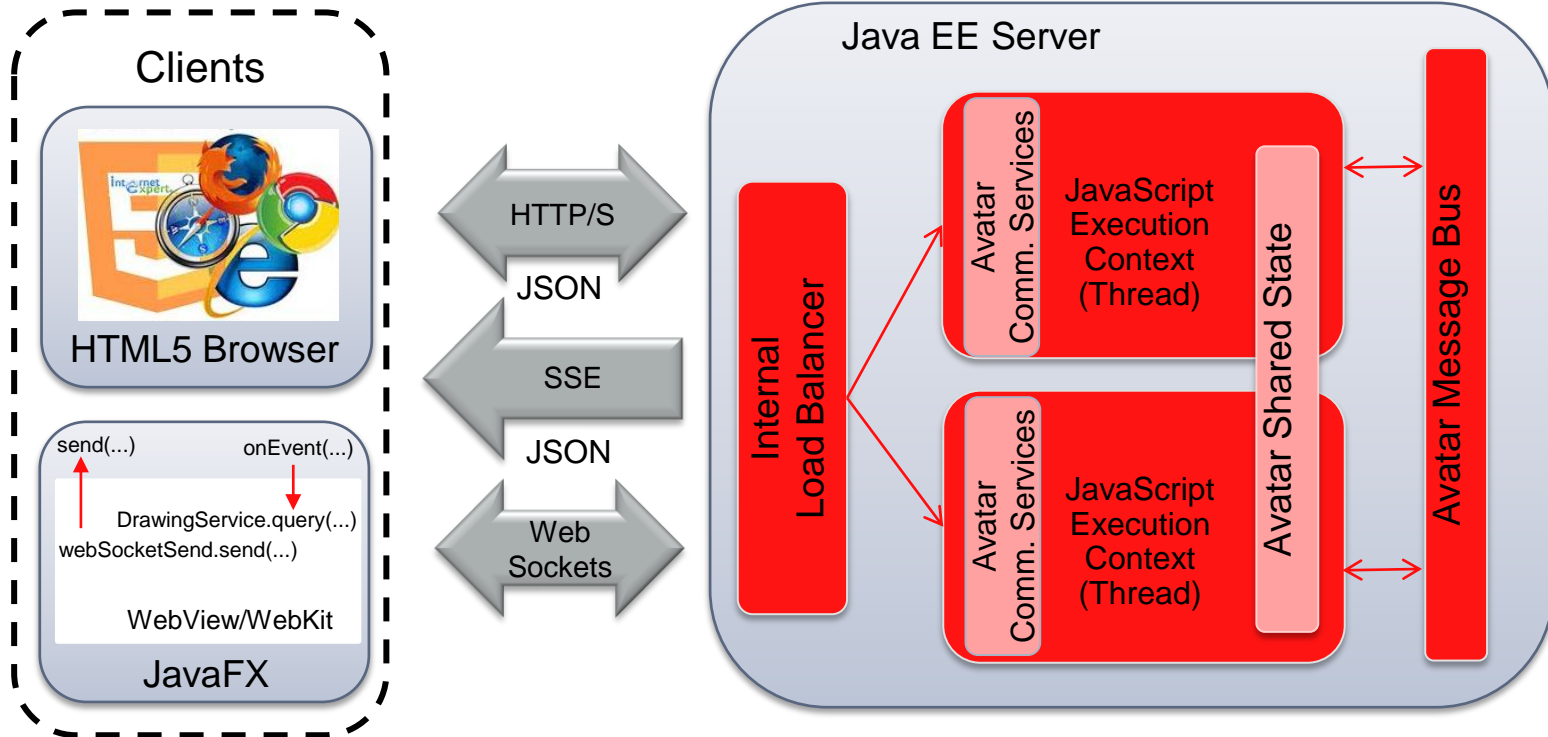
# Drawing Board HTML5 Demo

## Technology usage

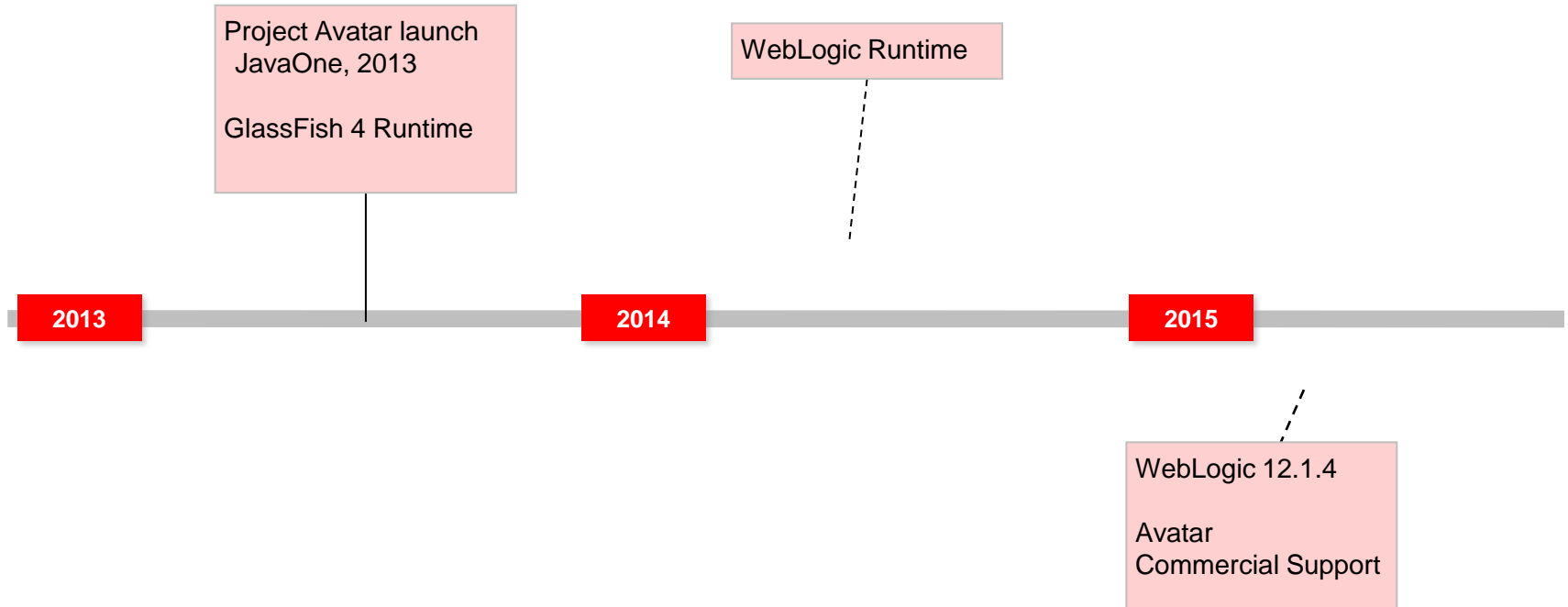
- JAX-RS: CRUD for drawings
- SSE: distributing the list of drawings to all connected clients
- WebSocket: distributing the updates of a drawing to all connected clients
- JSON: implementing of encoder/decoder of the WebSocket server endpoint
- Java – JavaScript bridge(WebEngine): modifying the AngularJS client by replacing the WebSocket/SSE JavaScript client communication with a Java implementation in the JavaFX client

# Drawing Board HTML5 Demo

Using Avatar Services (<http://github.com/doschkinow/hol-sse-websocket/solutions/exercise7>)



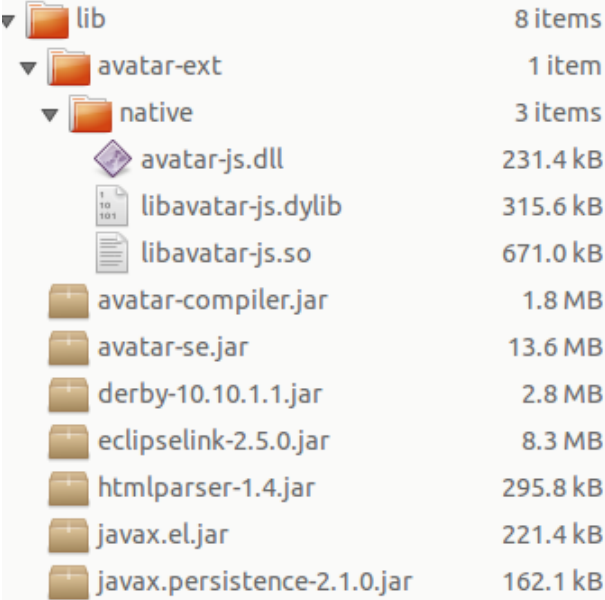
# Avatar Roadmap



# Avatar-SE

## Lightweight implementation on Java SE

- Internal project
  - Seeking to deliver a very lightweight implementation
- Zip-distribution, based on Grizzly as protocol engine
  - Includes JPA and JavaDB
- Running the Avatar examples application
  - `java -jar lib/avatar-se.jar start avatar-se-1.0-  
ea/Project-Avatar-examples/hangman`



lib	8 items
avatar-ext	1 item
native	3 items
avatar-js.dll	231.4 kB
libavatar-js.dylib	315.6 kB
libavatar-js.so	671.0 kB
avatar-compiler.jar	1.8 MB
avatar-se.jar	13.6 MB
derby-10.10.1.1.jar	2.8 MB
eclipselink-2.5.0.jar	8.3 MB
htmlparser-1.4.jar	295.8 kB
javax.el.jar	221.4 kB
javax.persistence-2.1.0.jar	162.1 kB



# Java Community Questions on Avatar 1/2

<http://blog.n-k.de/2014/07/avatarjs-project-avatar-feedback-from.html?m=1>

- Why did you start Avatar.js and Avatar
  - To exploit new JVM capabilities and Nashorn
  - Synergy effects when running Java EE and Node.js apps on same JVM
- Who is your target group
  - Node.js developers wishing to access existing Java apps/libs or to take advantage of a rich Java appserver infrastructure
  - Java/JVM-based language developers wishing to use/integrate Node.js modules or Node.js single threaded non-blocking programming model
  - Java/JavaEE platform provider wishing to extend their offerings

# Java Community Questions on Avatar 2/2

<http://blog.n-k.de/2014/07/avatarjs-project-avatar-feedback-from.html?m=1>

- Do you plan to invest more in Avatar
  - we are evaluating different scenarios and are going to invest in Avatar
- What about (more) documentation and more promotion
  - more to come at JavaOne 2014
- Why are there no real developing activities/commits since end of March
  - <https://java.net/projects/avatar/sources/git/history>? reveals a last modification on June 19 (but indeed a minor one)
  - We often evaluate/develop our products in closed source first

# Next Steps

- Go to [avatar.java.net](https://avatar.java.net)
  - <https://avatar.java.net>
- Download it
- Try it out
- Give us feedback
  - <https://avatar.java.net/mailing.html>

# Summary

## Server Side JavaScript on the JVM

- Invoke Java code
- Multi-threading optimizations
  - Share state across threads, JVMs
  - Built-in load balancing across threads
- Leverage Java EE services
- Deploy on existing Java EE infrastructure
  - Leverage appserver features (clustering, lifecycle management)

**Hardware and Software**

**ORACLE®**

**Engineered to Work Together**

**ORACLE®**