

ORACLE®

**Project Avatar:
Server Side JavaScript on the JVM
QCon London 2014**


David Delabasse

@delabasse

Software Evangelist - 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.



*"The French and the British are such good enemies
that they can't resist being friends."
(Peter Ustinov)*

Please evaluate
my talk via the
mobile app!





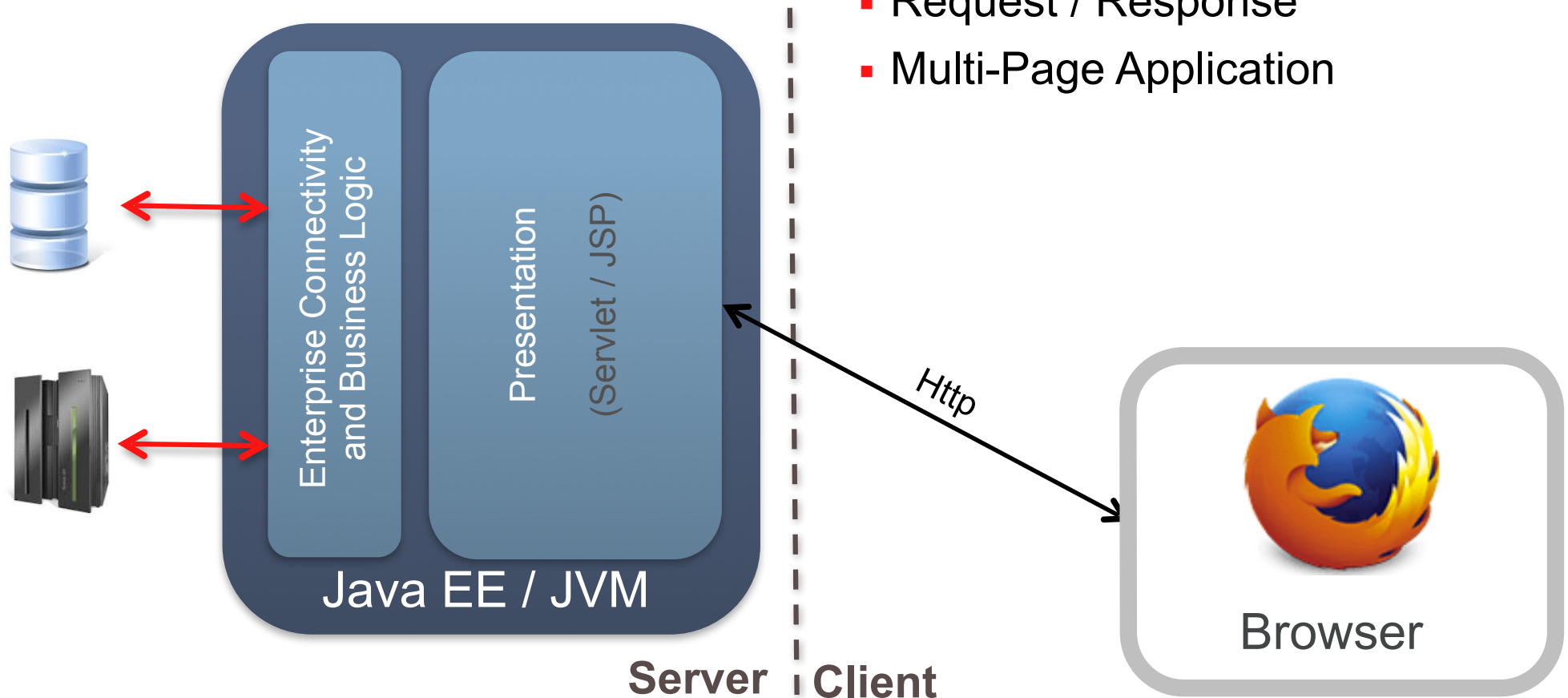
Agenda

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

Evolution of Web Application Architecture

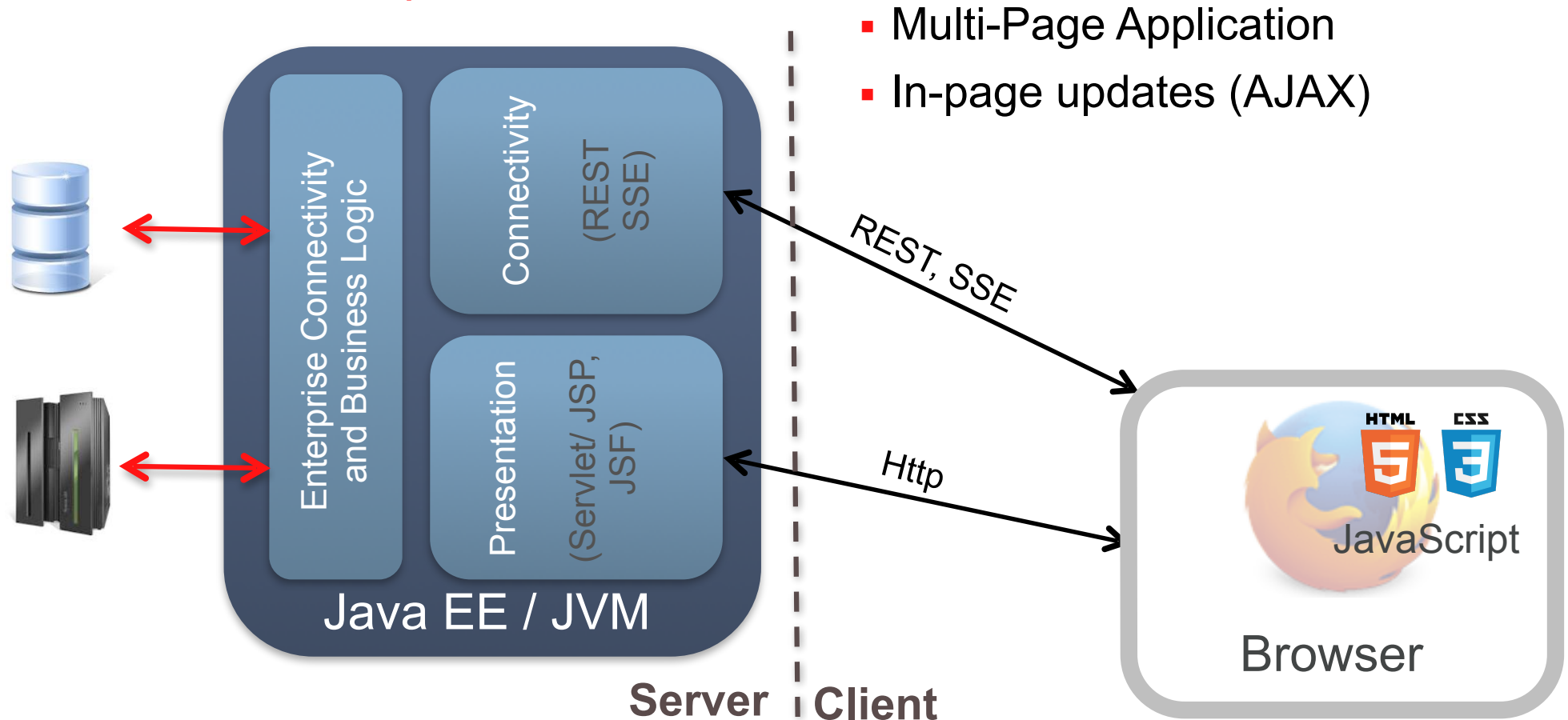
A Java EE Perspective

- Request / Response
- Multi-Page Application



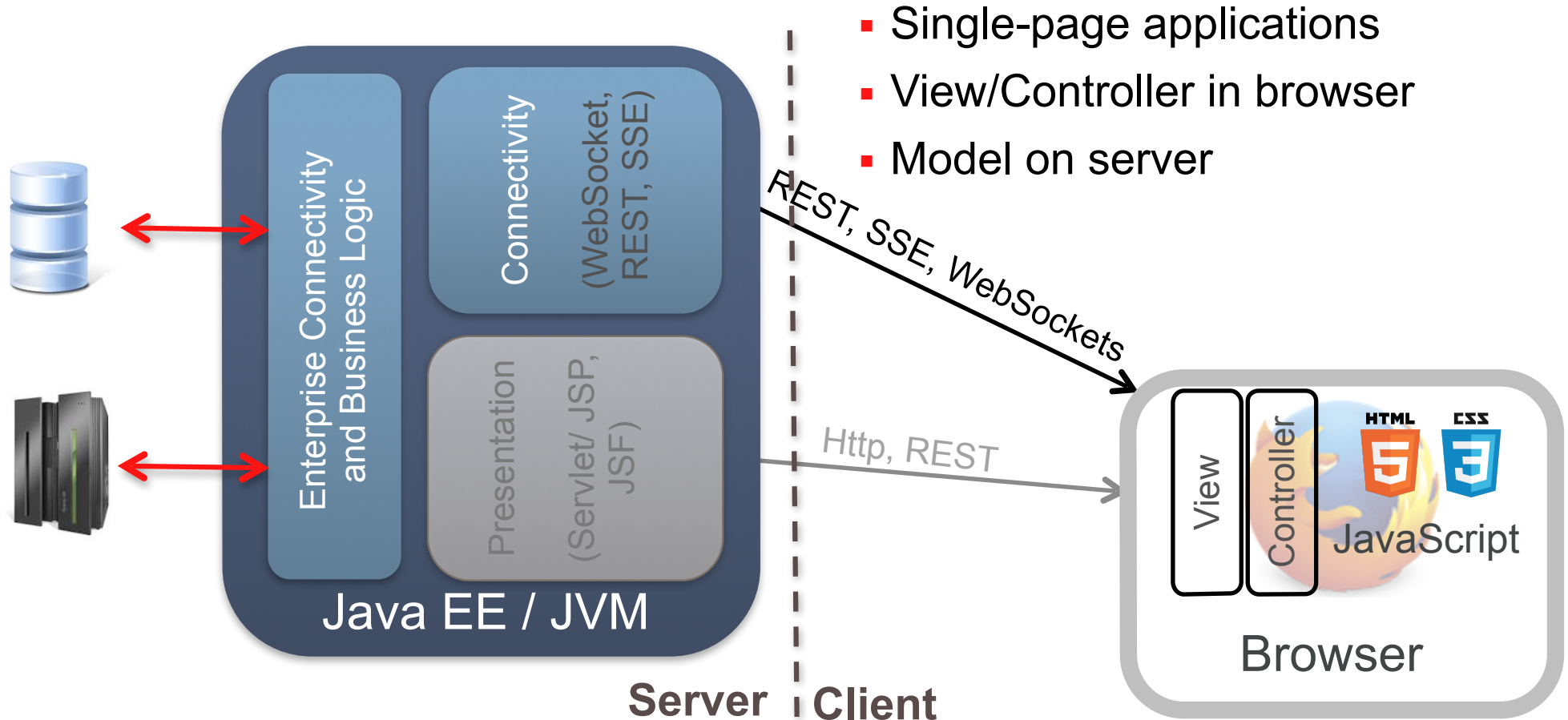
Evolution of Web Application Architecture

A Java EE Perspective



Modern Web Application Architecture

A Java EE Perspective



- Single-page applications
- View/Controller in browser
- Model on server

Java EE – 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

Server Side JavaScript

- Server-side JavaScript based on Chrome v8 engine
- Created by Ryan Dahl (2009), Open Source
- Designed for fast, scalable network applications
- Event-driven, non-blocking I/O model
- “Melting pot community”
 - Java, .NET, Browser, PHP, etc ...
 - Very active, with 60,000+ modules (YMMV)



Node.js Programming Model

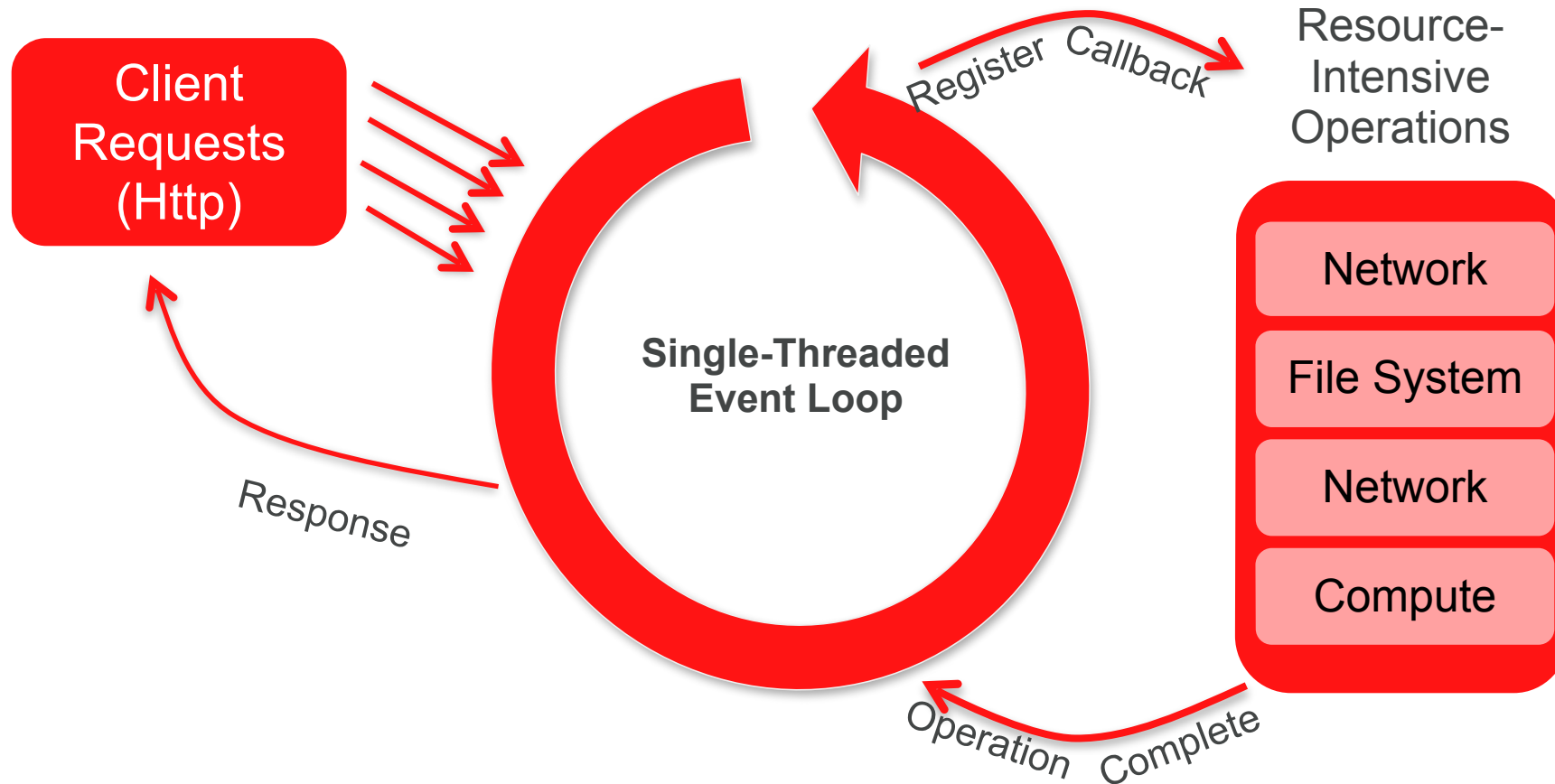
- Multi-threading is hard
 - Thousands of concurrent connections
 - Deal with deadlocks and race conditions
- Blocking I/O is bad
- Single threaded
- Event-loop
 - Callback model
 - Non-blocking I/O calls
 - Heavily parallelized
 - User code: sequential

HTTP Callback Example:

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

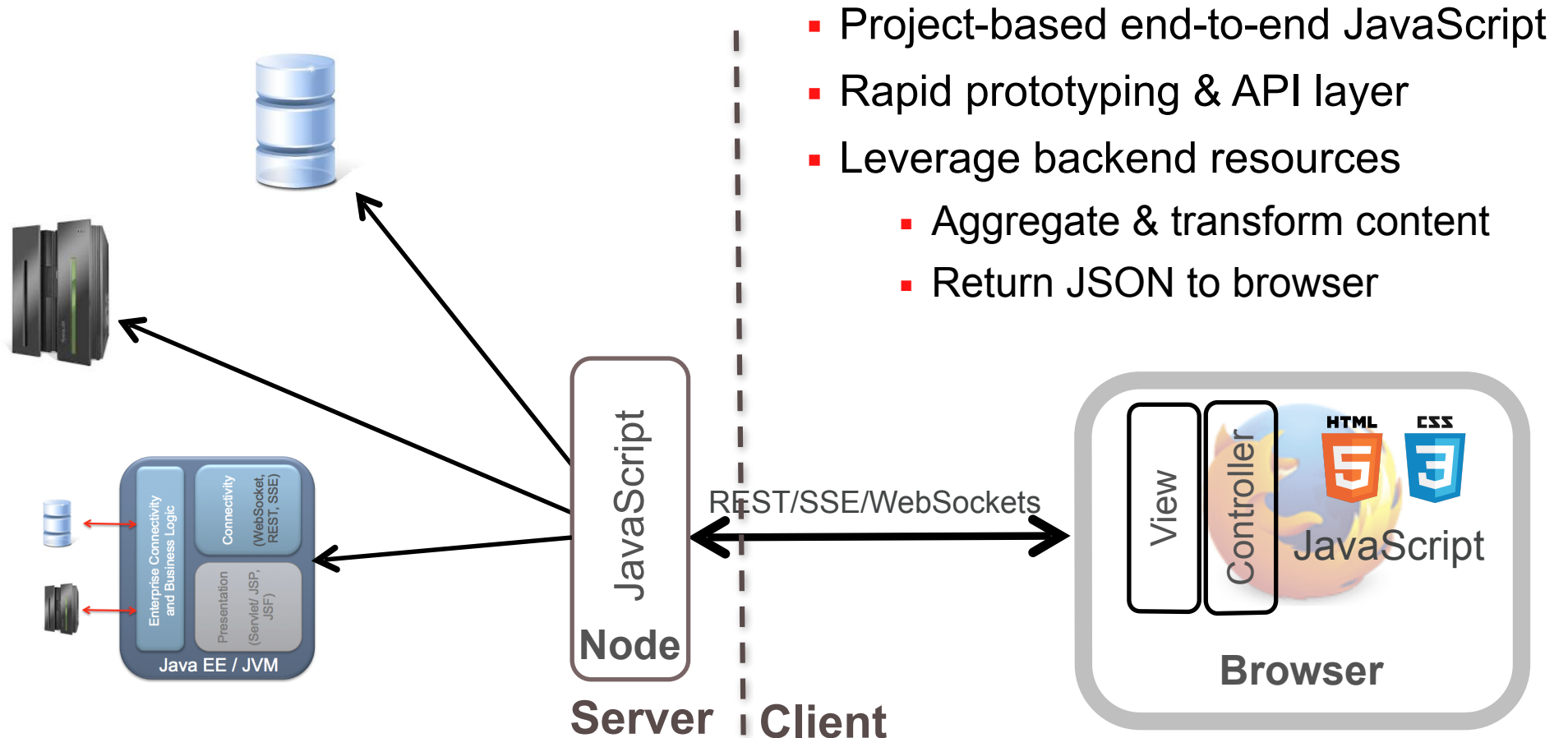
http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8080);
```

Node.js Event Loop



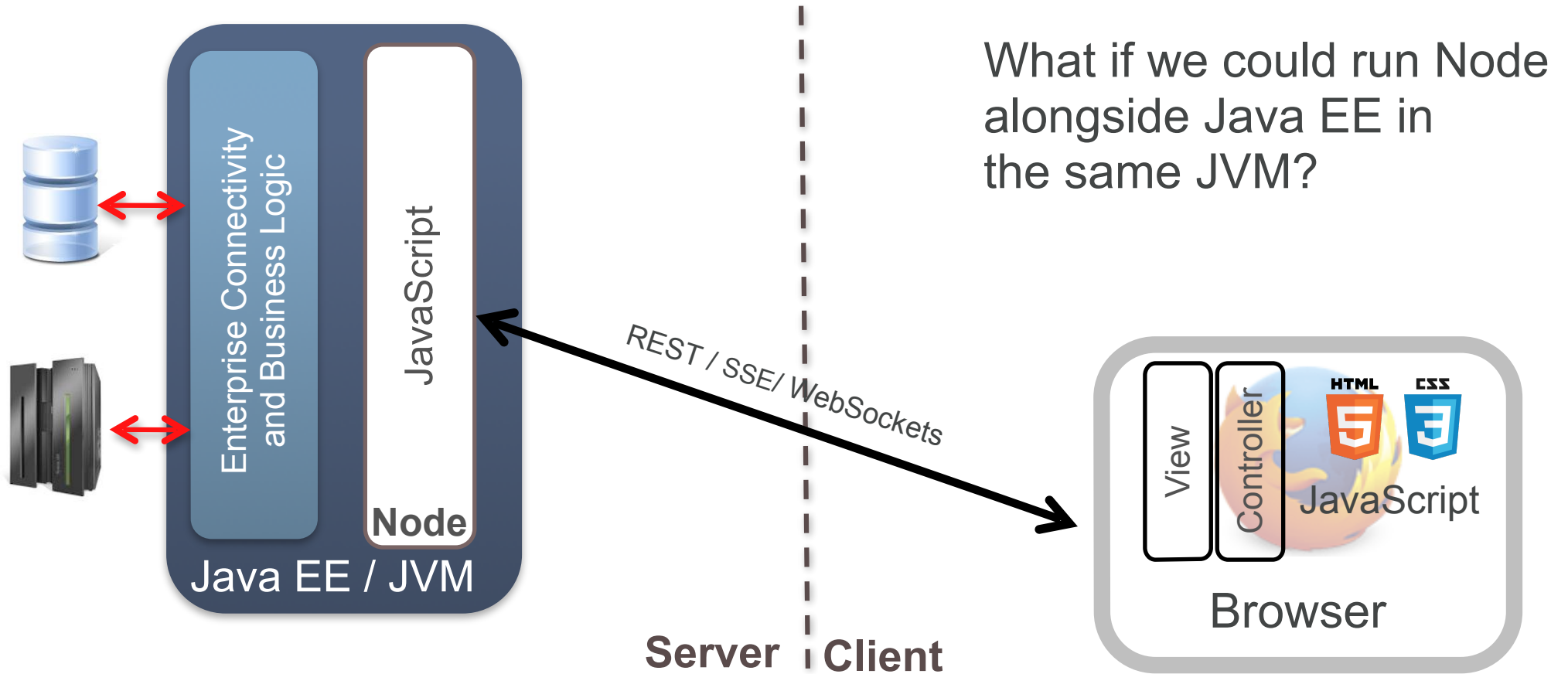
Evolution of Web Application Architecture

Mobile-enabling existing services



Evolution of Web Application Architecture

Mobile-enabling existing services





JavaScript and Node on the JVM

Project Nashorn

JavaScript on the JVM



- ECMAScript 5.1 compliant
- Bundled with JDK 8
 - Replaces Rhino
 - Faster (2x – 10x)
 - More secure
- Seamless Java \leftrightarrow JavaScript interoperability

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

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



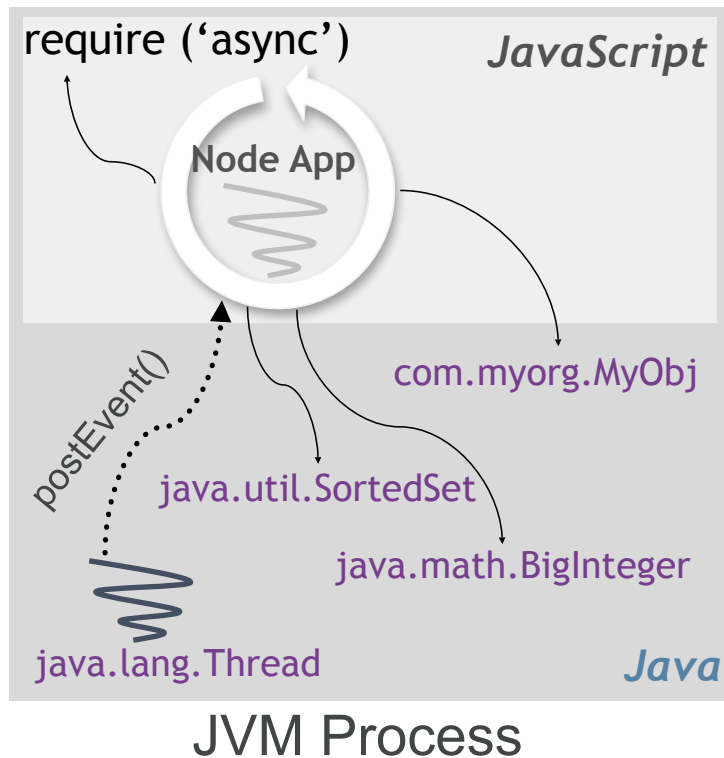
Avatar.js

Node on the JVM

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

Avatar.js = Node + Java

Leverage Java, including Threads



- Node 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.myorg.MyObj();`



Project Avatar: Advanced JavaScript Services Leveraging Java EE



Project Avatar

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 Node-compatible runtime
- Adds integrated enterprise features

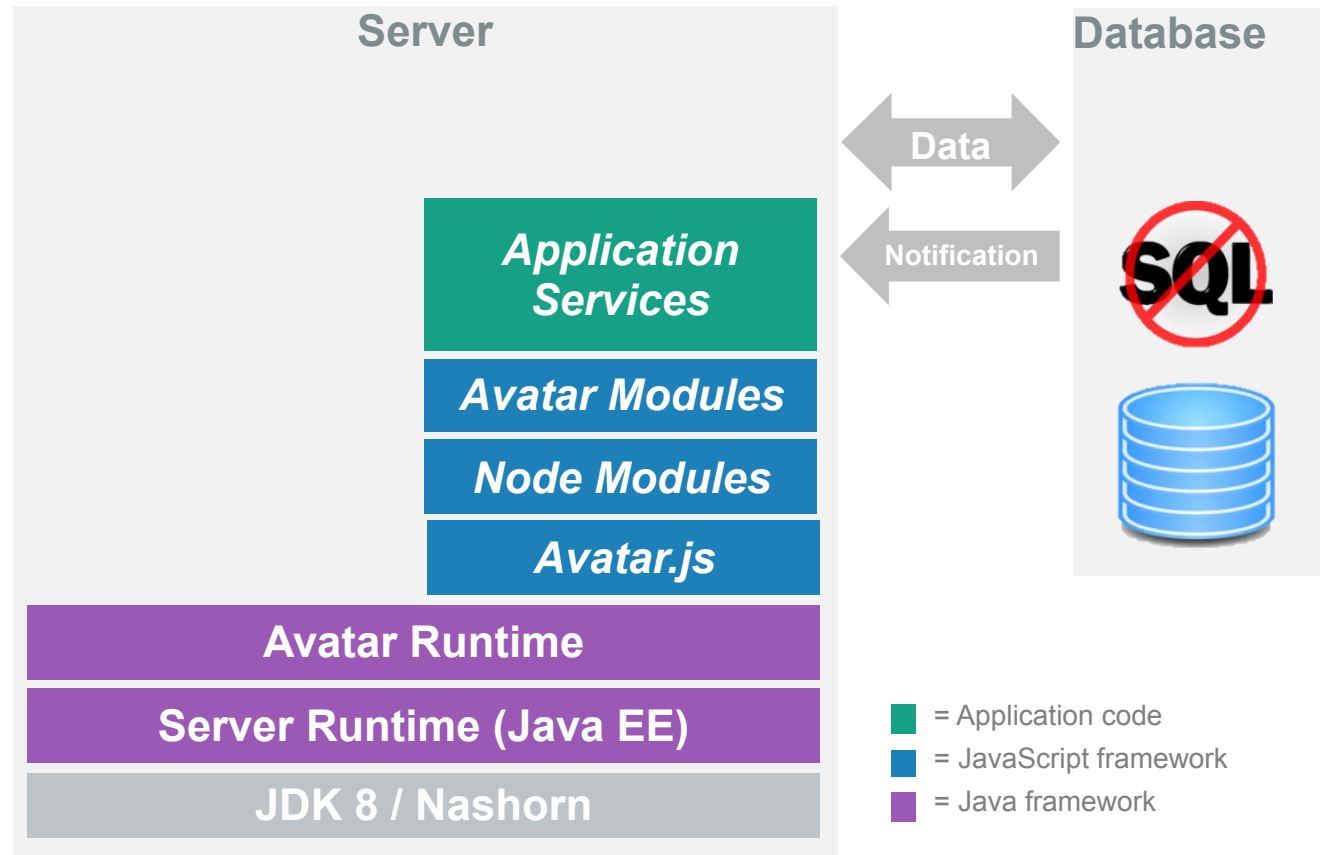


Project Avatar

Leveraging the JVM and Java EE

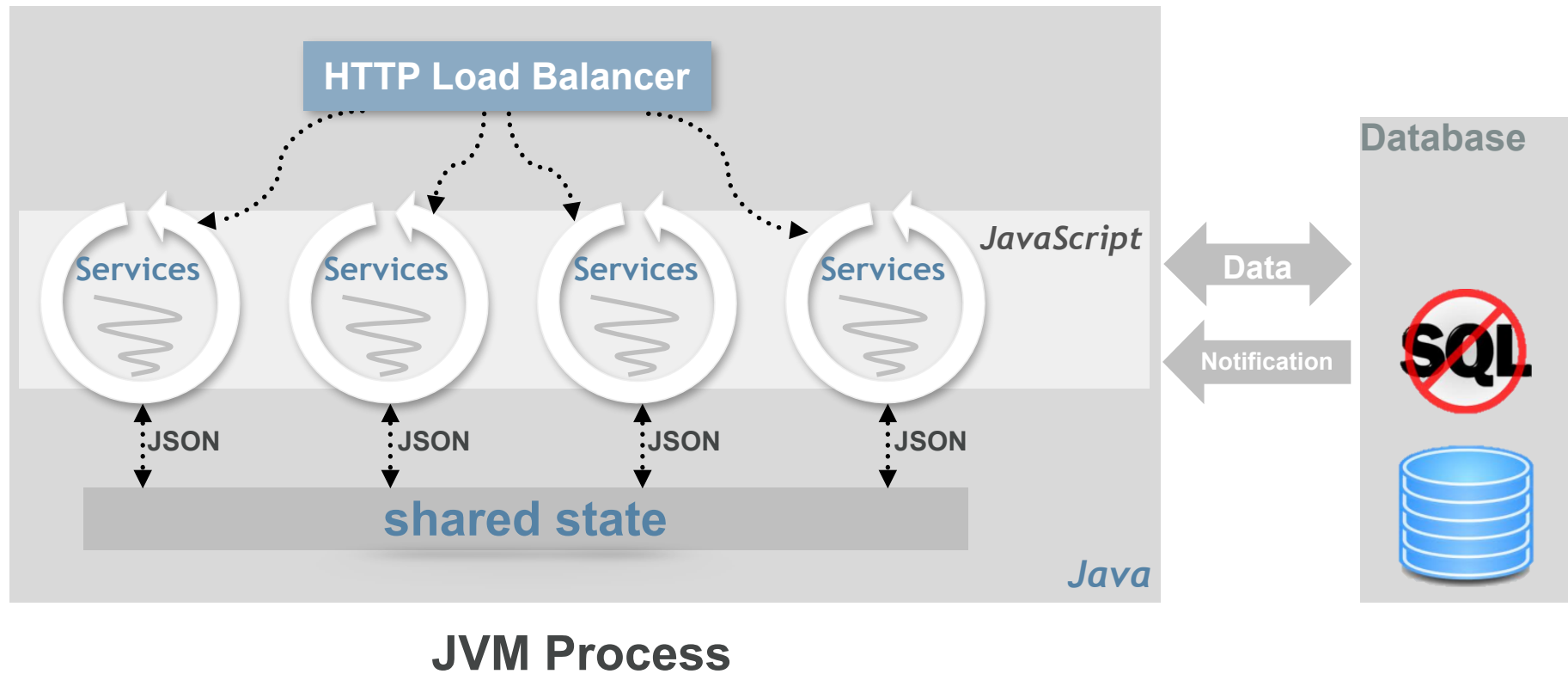
- Multi-threading, lightweight message passing, no mutable shared state
- Model Store – Object Relational Mapping
- HTTP listener / load-balancer managed by framework
- Messaging – JMS on Java EE container

Avatar Architecture - Server



Avatar Services

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 app = avatar.application;
var name = app.name;
var bus = app.bus;

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

// Start a background thread which publishes to the 'hello' topic
// Background threads do not participate in request processing
threads.newBackgroundThread('background', 'hello.js').start();
```

Push Service Example

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

// Register a push service that forwards background messages
avatar.registerPushService({url: 'push/message'}, function () {
  this.onOpen = function (context) {
    bus.on('example', function(msg) {
      context.sendMessage({body: msg});
    });
  };
});

// Create and start a background thread that publishes messages
threads.newBackgroundThread('background', 'background.js').start();
```

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);
    };
});
```



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, db

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',
  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 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
 - Testing: Oracle DB, MySQL, Derby (Embedded, Network)
 - Non-tested: Any other JDBC driver
 - Non-relational
 - Oracle NoSQL, MongoDB (in progress)

Model-Store API

Opportunities to leverage JPA features

- Lots of possibilities
 - Configure JPA provider using properties
 - Generate JavaScript model from database schema
 - User transactions
 - 2nd level JPA cache, TopLink Grid
 - Oracle RAC Support
 - More ...
- Maintain pure JavaScript API
- We're looking for YOUR feedback!

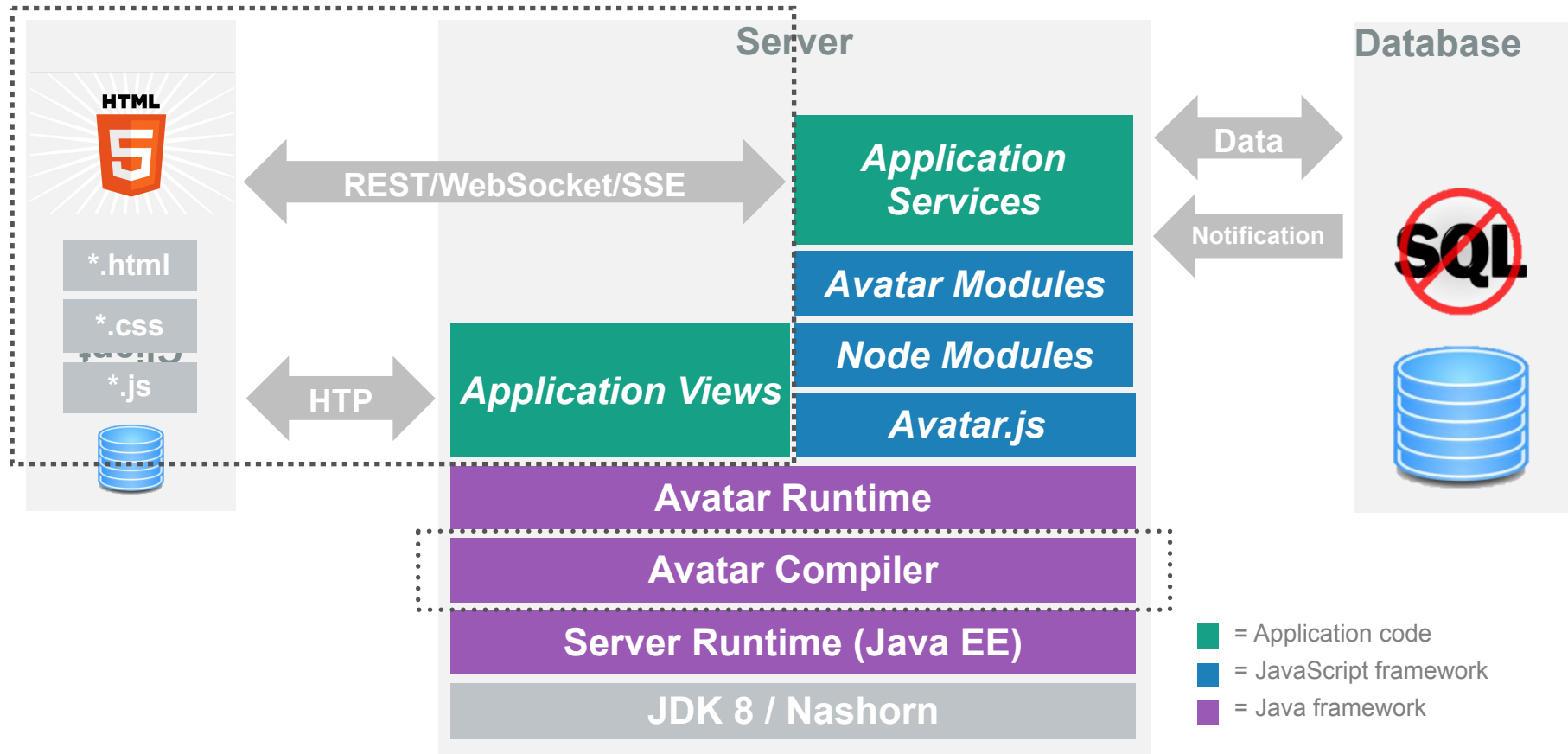
Avatar Client Framework



Avatar Client Model

- 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
 - AMD modules for code partitioning

Avatar Architecture – Server and Client



Hello World Example

Model

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

View

```
<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>
```

Chat Example: WebSockets

```
<script data-model="socket">
  var ChatModel = function() {
    ...
    this.sendMsg = function() {
      this.send(this.user + ":" + this.message);
      this.message = "";
    };
  };
</script>

<script data-type="ChatModel" data-instance="chat"
  data-url="websocket/chat">
</script>
```

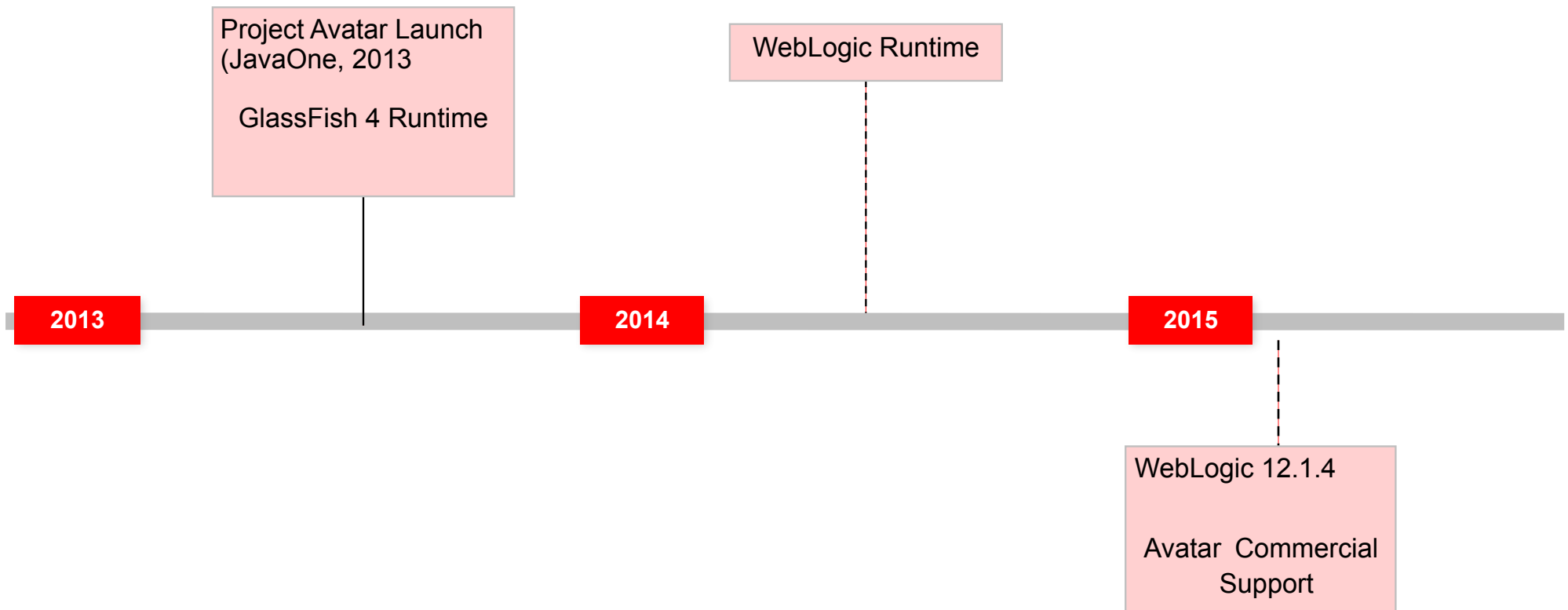


Summary

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)

Avatar Roadmap





Next Steps

1) Download

<https://avatar.java.net/>

2) Try it out

3) Give us feedback

<https://avatar.java.net/mailing.html>



Thanks!

Project Avatar

<https://avatar.java.net/>

Java EE 8 Survey

<http://glassfish.org/survey>

ORACLE®