4 Technology Evolution for Web Applications

Literature:

N. Chapman, J. Chapman: JavaScript on the Server Using Node.js and Express, MacAvon Media 2014 (Ebook €6)

J.R. Wilson: Node.js the Right Way, Practical, Server-Side JavaScript that Scales, Pragmatic Bookshelf 2014 (Ebook €8.50)

JavaScript: Full-Fledged Programming Language

- 1995 (Brendan Eich, Netscape): Mocha/LiveScript/JavaScript/ECMAScript
  - Java for demanding Web programs
  - JavaScript for simple browser interactivity
  - Standardized as ECMA-262 since 1997
- Hybrid language (procedural/functional/object-oriented):
  - Expressions, statements inherited from *C*
  - First class functions inherited from *Scheme*
  - Object prototypes (instead of classes) inherited from *Self*
- Growing attention on JavaScript since approx. 2005
  - DOM tree manipulation since 2000
  - Simplified DOM manipulations by JavaScript Libraries (e.g. JQuery 2006)
  - AJAX applications through XMLHttpRequest object (2004)
  - High-speed execution engines
- Continuing trend
  - Client-side Web frameworks, see later

Current standard:
http://www.ecma-international.org/ecma-262/6.0/

Node.js

- Stand-alone JavaScript interpreter and runtime system
  - Ryan Dahl 2009
  - Based on **V8** execution engine (developed for Chrome)
- Key feature: Asynchronous, non-blocking I/O operations
  - Single-threaded fast event loop
  - I/O handling based on *callback functions* only
- Targeted for distributed programs
  - Handling requests from many clients
  - Exchanging data between programs
  - Exchanging data between servers
  - Basis for fast Web server software
- Supports modular JavaScript software
- Easily extensible using Node Package Manager (NPM)

http://www.nodejs.org/

I/O Blocking And Restaurant Counters

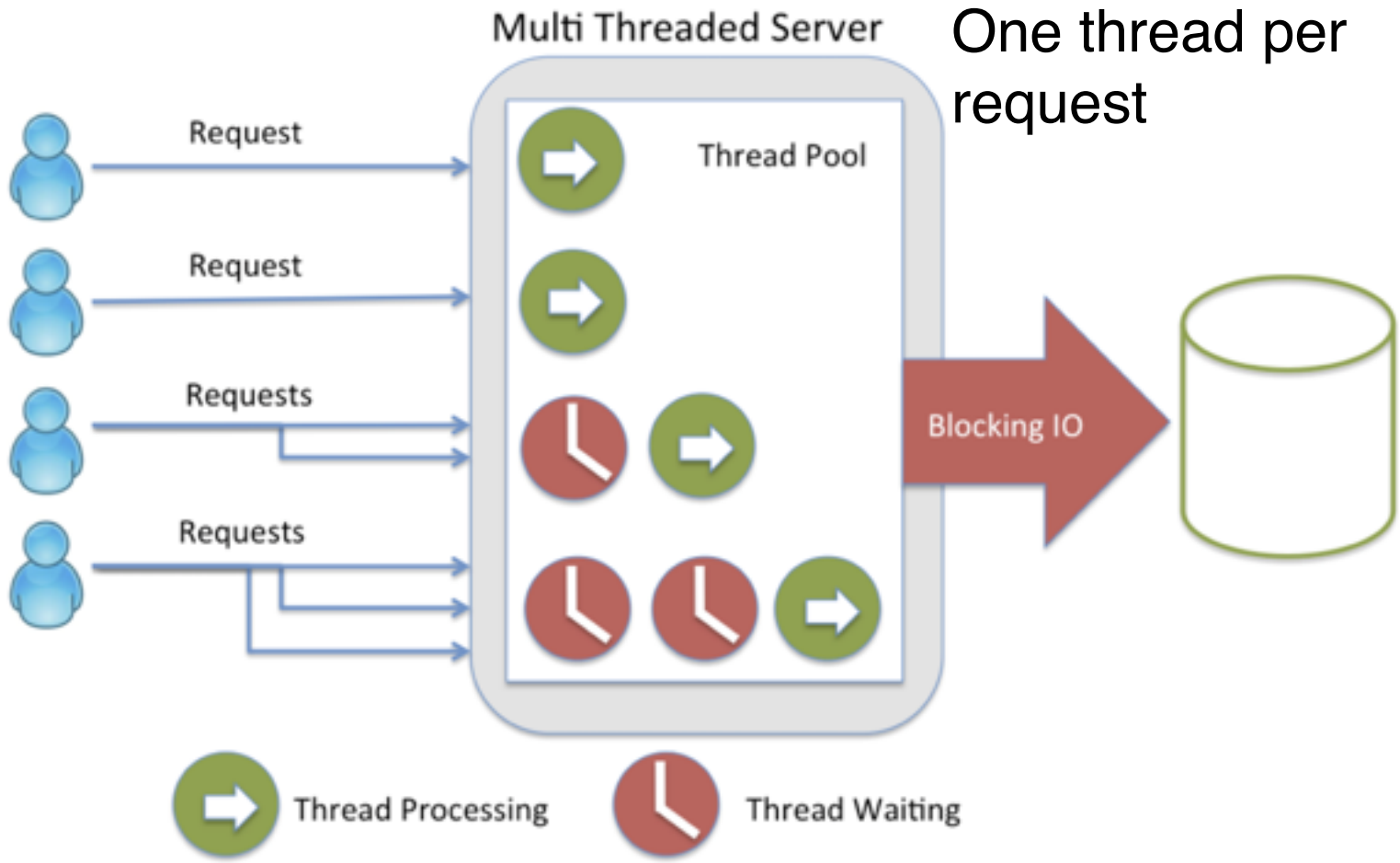
http://code.danyork.com/2011/01/25/node-js-doctors-offices-and-fast-food-restaurants-understanding-event-driven-programming/

Picture: Gerry Balding, Flickr

Question

- When do you feel on such a restaurant counter (or in any other queue) that the process could be sped up?
  - Assuming we do not add resources, so we just want to optimize…

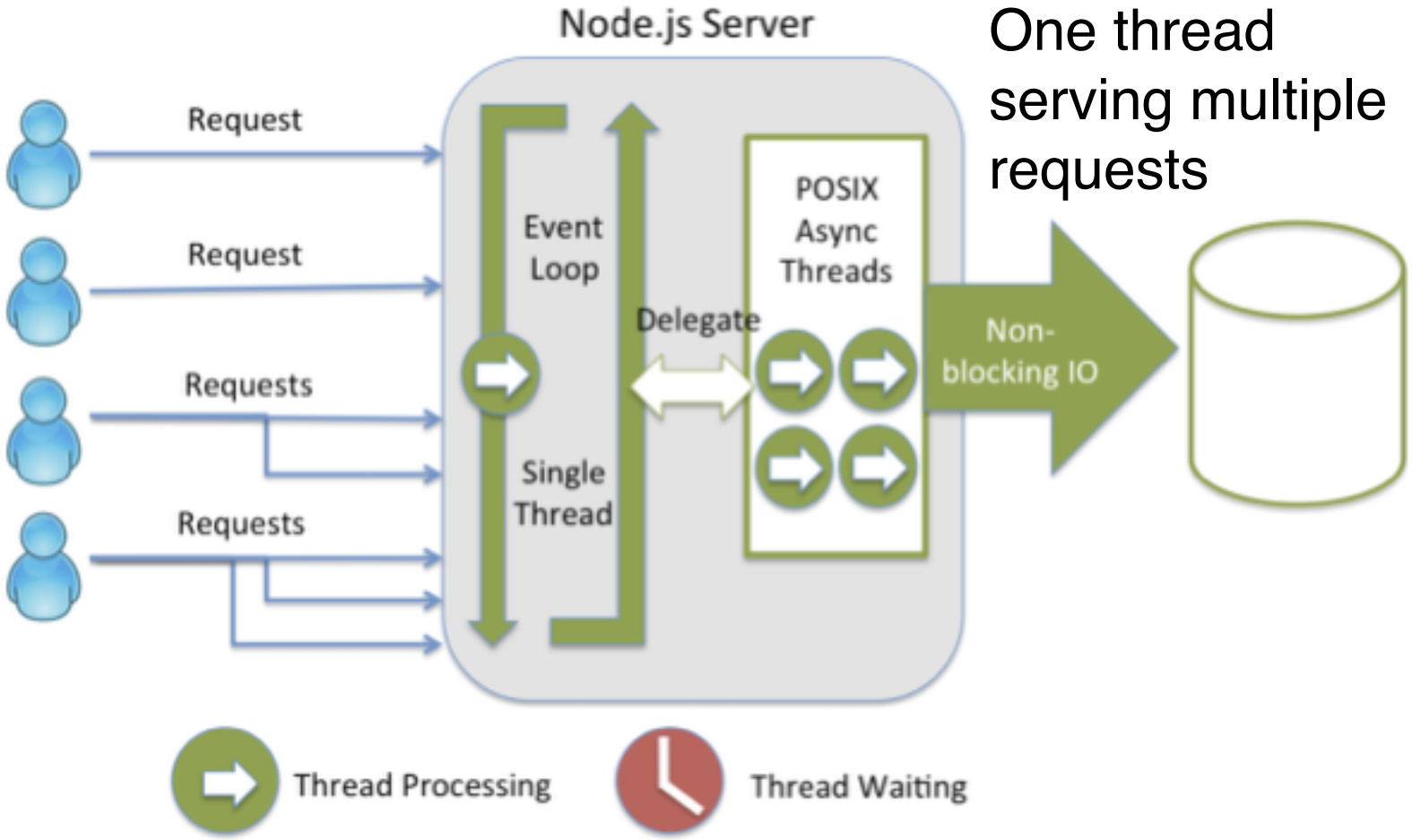
Blocking (Synchronous) I/O Model



Example:
Apache with PHP

http://strongloop.com/strongblog/node-js-is-faster-than-java/

Non-Blocking (Asynchronous) I/O Model



Example: Node.js

http://strongloop.com/strongblog/node-js-is-faster-than-java/

JavaScript Detour: Modules

- Module: Separate piece of code, to be used in other code
- JavaScript modules:
  - Not existent in currently mainly used version (ECMAScript 5)
  - Existent in ECMAScript 2015 (ES6, "Harmony")
- Node.js provides a simple proprietary module system
  - Modules are JavaScript (.js) files
  - Module code assigns functions to a special variable (**`export`**)
  - Using a module: **`require(modulename)`** function

```
var PI = Math.PI;

exports.area = function (r) {
  return PI * r * r;
};

exports.circumference = function (r) {
  return 2 * PI * r;
};
```

```
var circle = require('./circle.js');
console.log(
  'The area of a circle of radius 4 is '
          + circle.area(4));
```

node/circle.js
node/circleuse.js

Network I/O in Node: Echo Server

```
var net = require('net');

var server = net.createServer(function(c){
  console.log('server connected');
  c.on('end', function() {
    console.log('server disconnected');
  });
  c.write('hello, here is the echo server\r\n');
  c.on('data', function(data) {
      c.write('echoserver-> '+data);
  // shorter: c.pipe(c)
  });
});

server.listen(8124, function() {
  console.log('server bound');
});
```

emitter.on(event, listener)
Adds a listener to the end of the listeners array for the specified event.

Echo Server Demo

```
Heinrichs-MacBook-Pro:node hussmann$ node echoserver.js
server bound
server connected
server disconnected
^CHeinrichs-MacBook-Pro:node hussmann$
```

```
Heinrichs-MacBook-Pro:~ hussmann$ telnet localhost 8124
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello, here is the echo server
Test input
echoserver-> Test input
^]
telnet> ^C
Heinrichs-MacBook-Pro:~ hussmann$
```

4 Technology Evolution for Web Applications

Literature:

N. Chapman, J. Chapman: JavaScript on the Server Using Node.js and Express, MacAvon Media 2014 (Ebook €6)

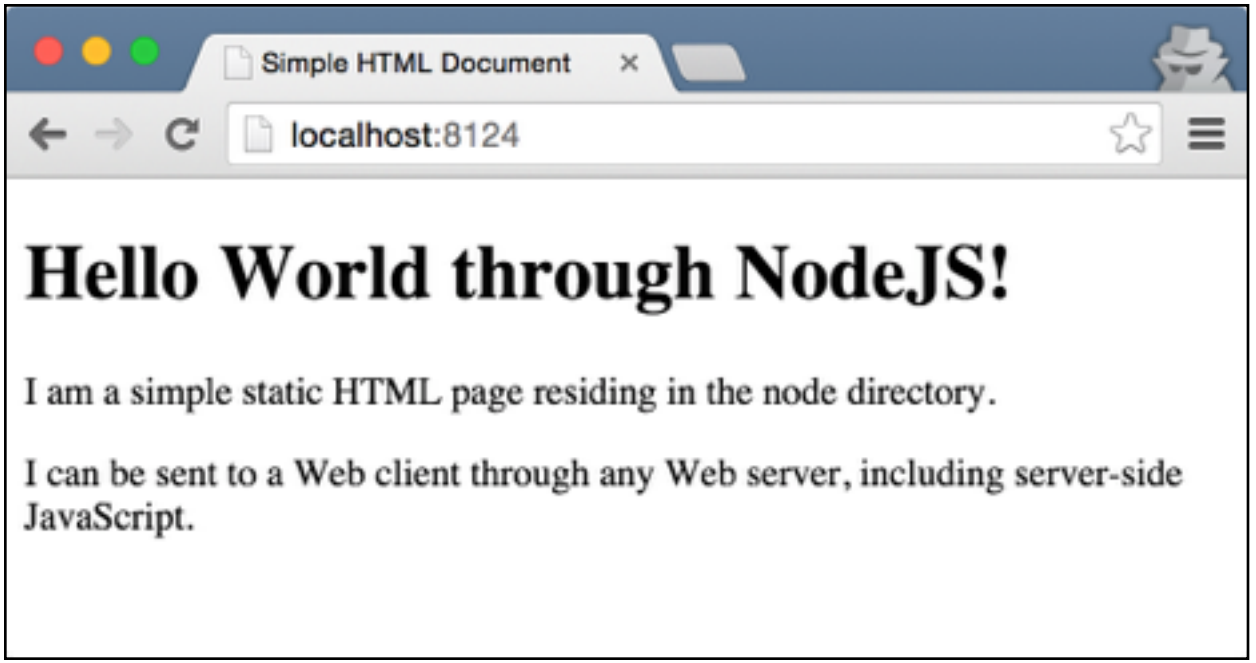
Trivial Web Server Based on Node

```
var http = require('http');
var fs = require('fs');
var filename = 'simple.html';

http.createServer(function (request, response) {
  response.writeHead(200, {'Content-Type': 'text/html'} );
  var fileStream =
    fs.createReadStream(filename);
  fileStream.pipe(response);
})
.listen(8124);

console.log('Server running at http://localhost:8124/');
```

webserver0.js

Trivial Web Server Demo

```
Heinrichs-MacBook-Pro:node hussmann$ node webserver0.js
Server running at http://localhost:8124/
```



Question

- What about the port numbers?
  - Why are we using strange large numbers like 8124?
  - Shouldn't we be using 80 for a Web server?

Express: Server-Side Web Application Framework

- "Full-Stack" frameworks for Web applications
  (in terms of the Model-View-Controller [MVC] paradigm):
  - Model: Manage data
  - View: Present data in HTML
  - Controllers: Co-ordinate input and output
  - Full-stack frameworks exist for other languages,
    e.g. Rails for Ruby, Django for Python, CodeIgniter for PHP
- Express:
  - Currently most popular framework for the Node platform
  - Limited in the "model" aspect of MVC
- Using Express:
  - "Generators": produce file structures and source code templates
  - Here: Avoiding generators for the first step, to understand the principles

http://www.marcusoft.net/2014/02/mnb-express.html

Trivial Web Server Based on Express

```
var express = require('express');
var app = express();

app.get('/', function (request, response) {
  response.send('Hello World from first Express example!')
})
.listen(3000);

console.log('Application created');
```

node/exapp0/app.js

- Steps required:
  - Create directory for application
  - Create dependency file "package.json" (using utility **`npm init`**)
  - Install app using **`npm install express –save`**
  - Save application code in file "app.js" (in resp. directory)
  - Execute **`node app.js`**

Express-Based Web Server for Static Files

```
var express = require('express');
var app = express();

app.use(express.static('public'));
app.listen(3000);

console.log('Web server (static files) created');
```

node/exstatic/app.js

- Replacement for traditional Web server (e.g. Apache)
- Directory (here: **`public`**) for HTML files

Express: Middleware, Routing

- Middleware: "Middleware is a function with access to the request object *(req)*, the response object *(res)*, and the next middleware in the application's request-response cycle." (expressjs.com)
  - Method **`use`**: execute middleware for any request
- Route handling:
  - *Path* against which request is matched
  - Callback to be called if match exists *(route handler)*
  - Methods to establish route for various request types (e.g. **`get`**)
  - Special object **`Router`** for sophisticated out handling
- Chaining of routes:
  - **`next()`** function as parameter
  - Calling **`next()`**: Next matching route handler (in declaration order)
  - Alternatively, a route handler can end the request processing
- Built-in middleware: Currently only **`static`**
- Third-party middleware: E.g. for cookie handling, logging etc.

Routing/Middleware: Simple Example

```
var express = require('express');
var app = express();

app.use(function (request, response, next) {
  console.log('Request received with query
   '+JSON.stringify(request.query));
  next();
});

app.get('/', function (request, response, next) {
  response.write('Hello World');
  next();
});

app.get('/', function (request, response) {
  response.write(' - from routing demo example!');
  response.end();
});

app.listen(3000);
```

node/exroute/app.js

Accessing a (NoSQL) Database from Node

```
var MongoClient = require('mongodb').MongoClient

var URL = 'mongodb://localhost:27017/music'

MongoClient.connect(URL, function(err, db) {

  if (err) return;

  var collection = db.collection('mysongs');
  collection.find({title: 'One'}).toArray(function(err, docs) {
      console.log(docs[0])
      db.close()
    })
})
```

node/index.js

Database & Express

- Basic idea:
  - Use a framework to access MongoDB:
    ```
    var monk = require('monk');
    ```
  - Connect to database through framework:
    ```
    var db = monk('localhost:27017/music');
    ```
  - Make database available to further request handlers (middleware):
    ```
    app.use(function(req, res, next){
        req.db = db;
        next();
    });
    ```
  - In route handler:
    ```
    router.get('/songs', function(req, res) {
        var db = req.db;
        var collection = db.get('mysongs');
        collection.find({…},{…},function(e,docs){
            …
        });
    });
    ```

Model-View-Controller in Node/Express

- Model:
  - Not directly supported
  - Indirectly through access to databases
- Controller:
  - Route handlers
  - Middleware
- View:
  - Templates for HTML code
  - Typical languages: Jade, Embedded JavaScript (EJS), Handlebars.js
  - Options:
    - » (Yet another) new language (e.g. Jade)
    - » HTML-style markup with extensions (EJS, Handlebars)
- Alternative (modern?) style:
  - Do not create view on server but just supply data to client apps
  - No "view" at all, but just some JSON/XML/… string

Example: View Using Embedded JavaScript

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <% include partials/head %>
  </head>
  <body>

    <% include partials/nav %>

    <h1><%= title %></h1>

    <ul>
      <% users.forEach(function(user) { %>
        <li><%= user.username %> is a <%= user.animal %>.</li>
      <% }); %>
    </ul>

  </body>
</html>
```

From: http://www.korenlc.com/node-js-tutorial-getting-started-with-node-express-and-mongodb/

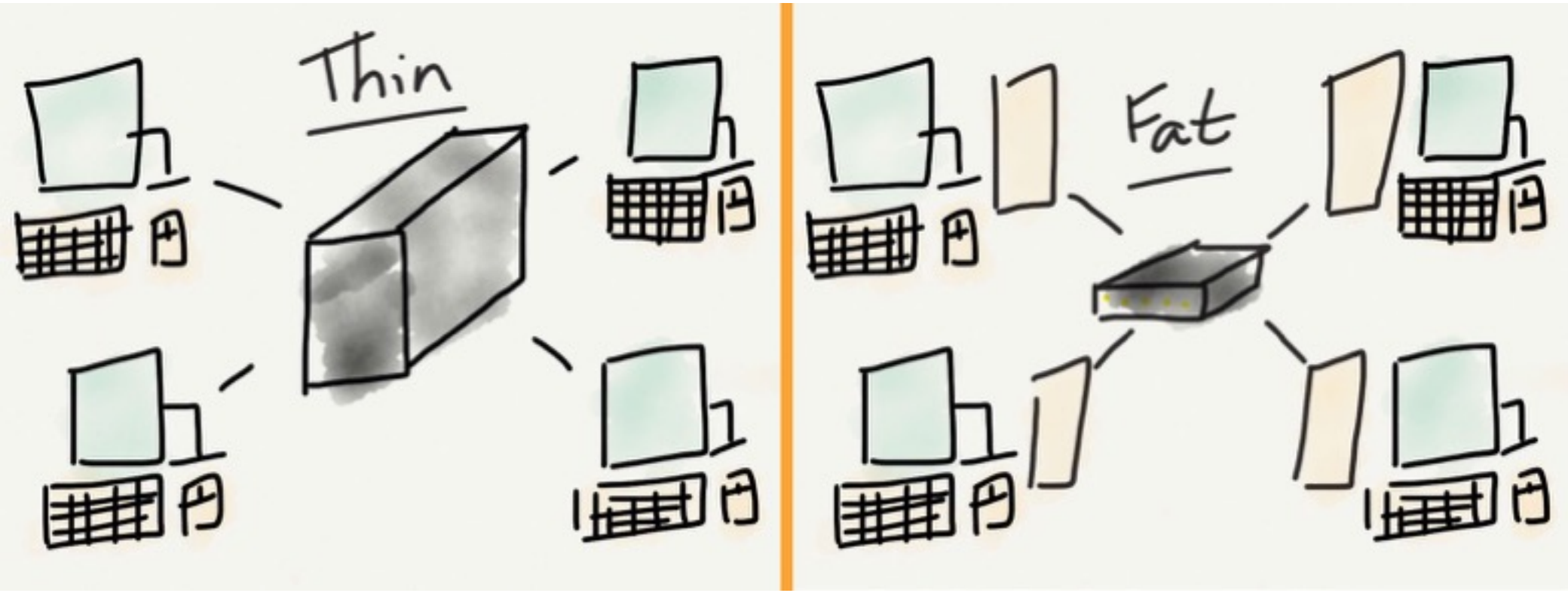
4 Technology Evolution for Web Applications

Literature:

J. Dickey: Write Modern Web Apps With the MEAN Stack (Mongo, Express, AngularJS, and NodeJS). Peachpit Press 2015

A. Grant: Beginning AngularJS. Apress 2014

Distributed System Architecture: Thin Clients vs. Fat Clients



https://stratechery.com/2013/the-alleged-13-inch-ipad-and-the-triumph-of-thin-clients-finally/

Thin vs. Fat – Round One (1980s-90s)

IBM
Mainframe 4381

https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP4381.html

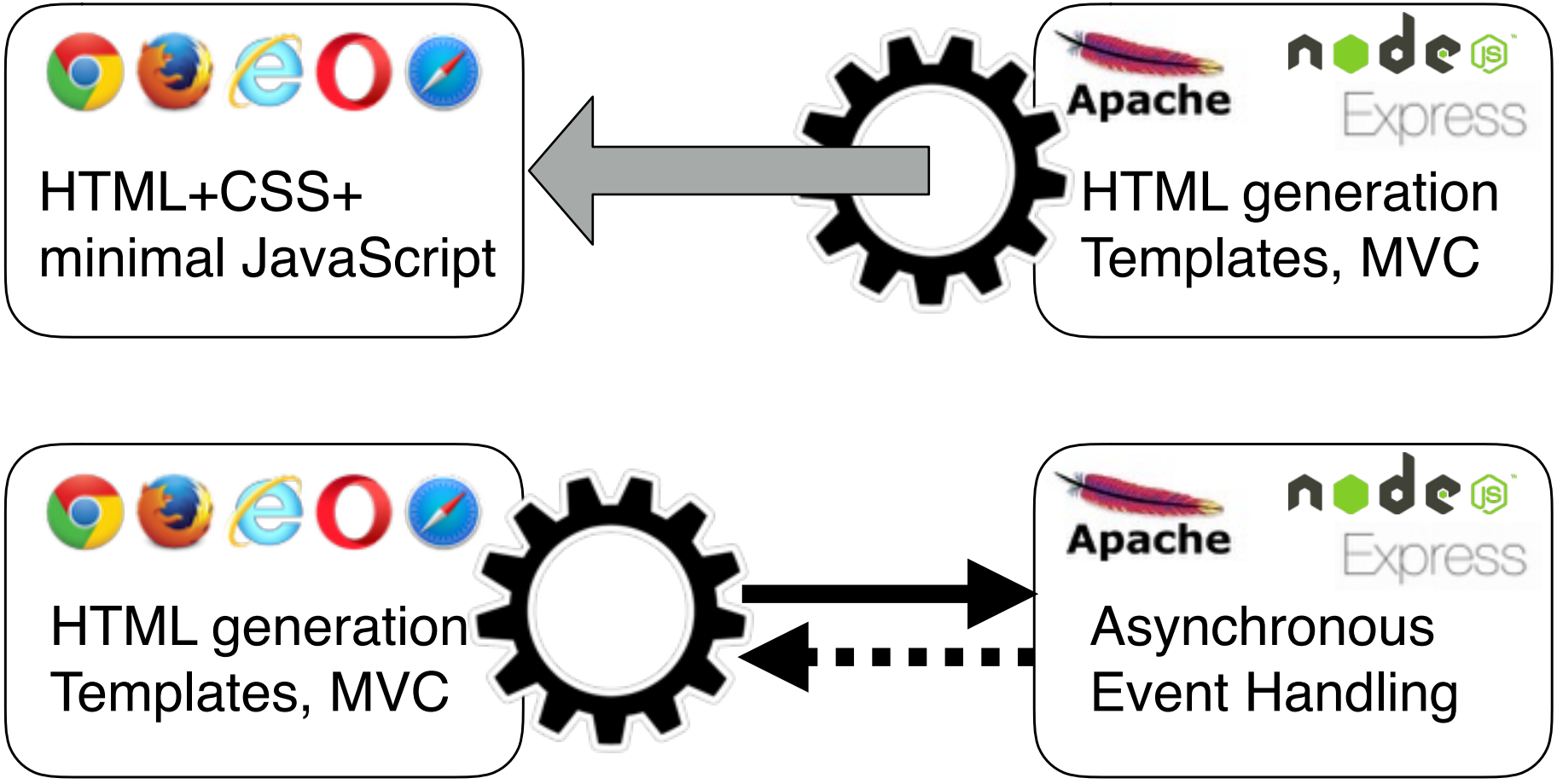
IBM
PC 5150

Source: Wikimedia

Thin vs. Fat – Round Two (1990s-today)

Source: Wikipedia

Thin vs. Fat – Round Three???

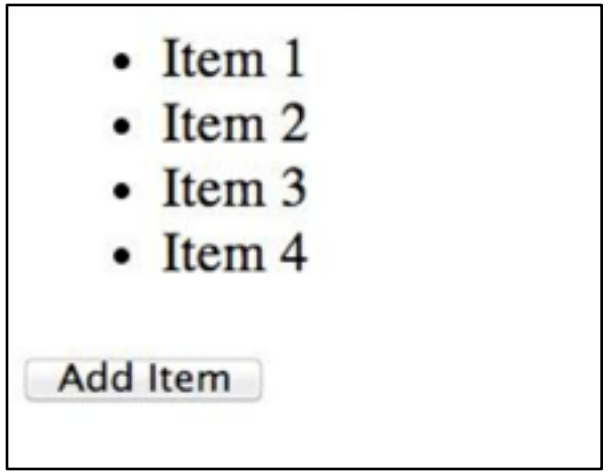


Client-Side JavaScript Frameworks

- Architecture based on Model-View-Controller ("MVx architecture")
- Event-driven logic
- Popular examples:
  - AngularJS (2009):
    - » Two-way data bindings
    - » Dependency injection
    - » Google sponsored
  - Backbone.js (2010):
    - » Lightweight framework, still having essential features
  - Ember (2007):
    - » Originally SproutCore
    - » Partially developed by Apple

jQuery vs. Angular, jQuery Version

- jQuery: Mainly procedural
  - Apply JavaScript to manipulate DOM tree
  - "Progressive-enhancement" Web paradigm: Augmenting static HTML with dynamic features
- Angular: Mainly declarative
  - Page structure integrates static and dynamic aspects



jQuery:

```
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
<button id='foo'>Add Item</button>

$('#foo').click(function(){
  $('#ul').append('<li>Item 4</li>');
});
```

jQuery vs. Angular, Angular Version

View (HTML-like):

```
<ul ng-controller='List-Ctrl'>
  <li ng-repeat='item in list_items'>{{item}}</li>
</ul>
<button ng-click='addListItem()'>Add Item</button>
```

Controller:

```
angular.module('app').controller('List-Ctrl',
function($scope) {
  $scope.list_items = [
    'Item 1',
    'Item 2',
    'Item 3'
  ];
  $scope.addListItem = function() {
    $scope.list_items.push('Item 4');
  });
});
```

Music Organizer with AngularJS

http://localhost/~hussmann/angular_songs/music.html

Music Organizer with AngularJS: HTML (1)

```
<!DOCTYPE html>
<html lang="en" ng-app="MusicApp">
<head> …
    <script src="js/angular.min.js"></script>
    <script src="js/app.js"></script>
</head>
<body ng-controller="MusicController">

<h1>Angular Music Organization</h1>

<div class="order">Order by:
<select ng-model="orderProperty" title="orderProperty">
    <option value="code">Code</option>
    <option value="title">Title</option>
    <option value="artist">Artist</option>
    <option value="album">Album</option>
    <option value="runtime">Runtime</option>
</select>
</div> (contd.)
```

Music Organizer with AngularJS: HTML (2)

```
(contd.)
<table class="songs">
    <thead>
    <tr>
        <th>…</th> …
    </tr>
    </thead>
    <tbody>
       <tr ng-repeat="song in songs | filter:searchTerm |
             orderBy:orderProperty">
          <td>{{song.code}}</td>
          <td>{{song.title}}</td>
          <td>{{song.artist}}</td>
          <td>{{song.album}}</td>
          <td>{{song.runtime}}</td>
       </tr>
    </tbody>
</table>
```

Music Organizer with AngularJS: HTML (3)

(contd.)

```
<input type="search" ng-model="searchTerm"
       placeholder="Filter..."/>

</body>
</html>
```

Music Organizer with AngularJS: app.js

```
var app = angular.module('MusicApp',[]);

app.controller('MusicController',
   function($scope,$http){
     $scope.orderProperty = 'code';
     $scope.searchTerm = '';
     $http.get('data/songsFromDB.php').
        success(function(data){
          $scope.songs = data;
        });
});
```

Most Recent Trend: Web Components

- Basic idea:
  - Hide DOM, behavior (JavaScript) and even style from site developer
  - Make HTML extensible: New elements in well-known syntax
  - Provide mechanism to create new custom components
- Characteristics:
  - Defined in html files
  - Imported with
    ```
    <link rel="import" href="somecomponent.html">
    ```
- Different libraries and flavors exist (e.g. polymer (Google), x-tag (Mozilla), Bosonic (community project)

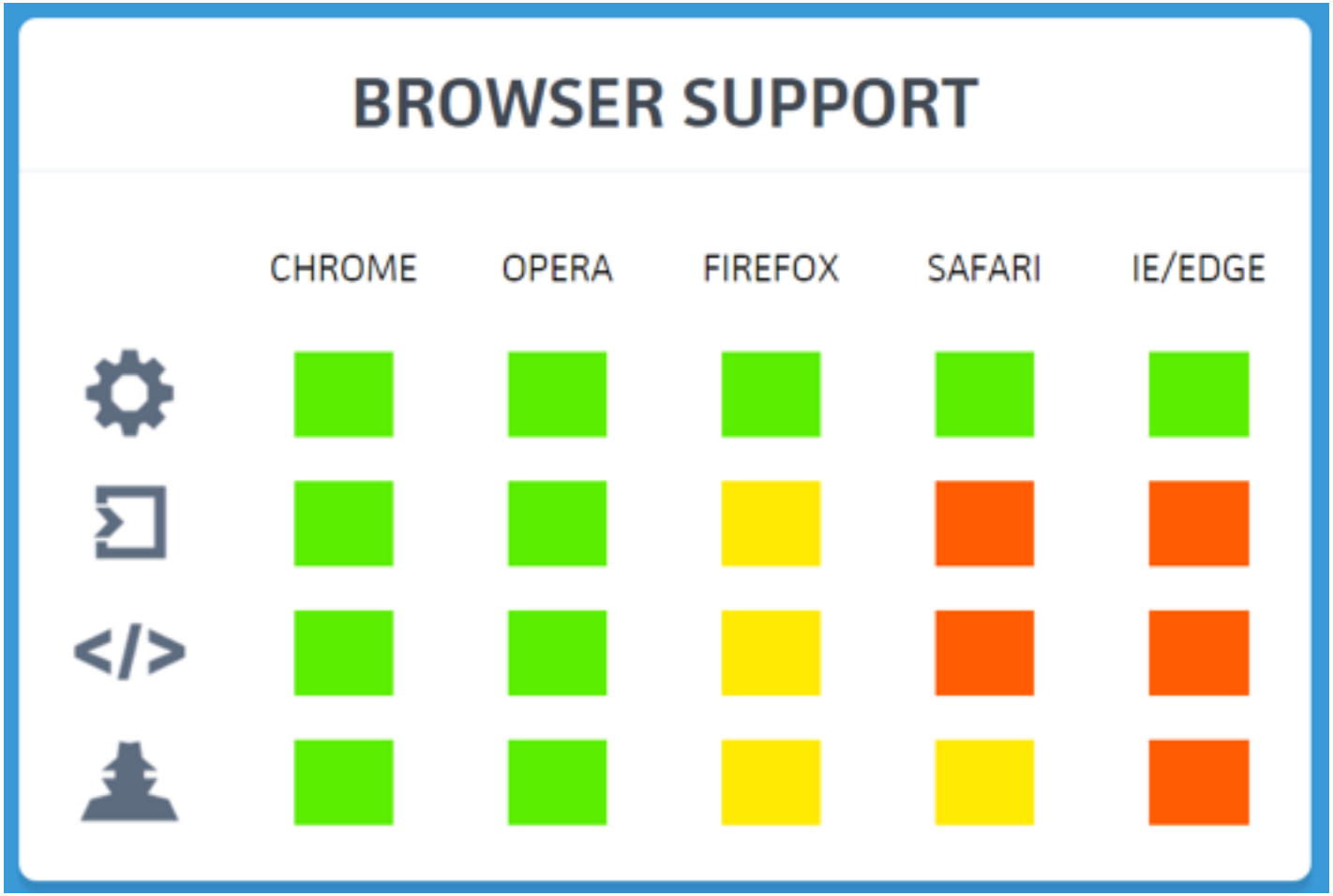
- Web components library, see https://www.polymer-project.org/1.0/
- Large catalogue of elements based on “Material Design” : see https://elements.polymer-project.org/
- Features:
  - Declarative:
    ```
    <iron-ajax url="data/songs.json" auto></iron-ajax>
    ```
  - Fast and light-weight
  - Built for responsive web apps
  - Two-way data binding (similar to AngularJS)
  - Presumably easier to learn than AngularJS

Example: Showing a Map

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Using GoogleMap Component</title>
    <script
      src="components/webcomponentsjs/webcomponents-lite.min.js">
    </script>

    <link rel="import" href="components/google-map/google-map.html">
    <style>
        google-map{
            height: 800px;
            width: 100%;
        }
    </style>
</head>
<body>

<google-map
        latitude="48.1470508"
        longitude="11.5759354"
        zoom="17"></google-map>
</body>
</html>
```

http://localhost/~hussmann/polymer/googleMapExample.html

Web Components Across Browsers

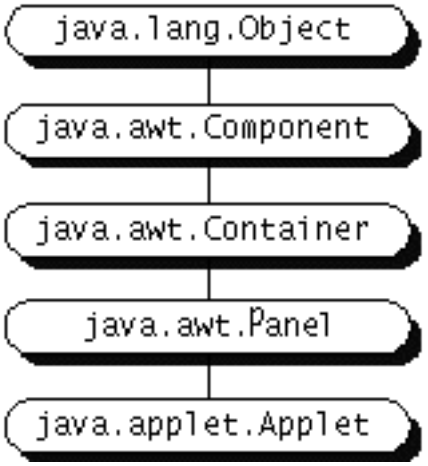


http://webcomponents.org/ visited Nov. 18th 2015

4 Technology Evolution for Web Applications

Example: Hello-World Applet (1)

- Applet = “small application”
  - Here: Java program, embedded in HTML page
- Class for applet derived from **`Applet`**
  - Calls **`paint`** method
  - Redefining the **`paint`** method = executed when painting display




```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
    public void paint(Graphics g) {
            g.setFont(new Font("SansSerif", Font.PLAIN, 48));
            g.drawString("Hello world!", 50, 50);
    }
}
```

Example: Hello-World Applet – HTML5

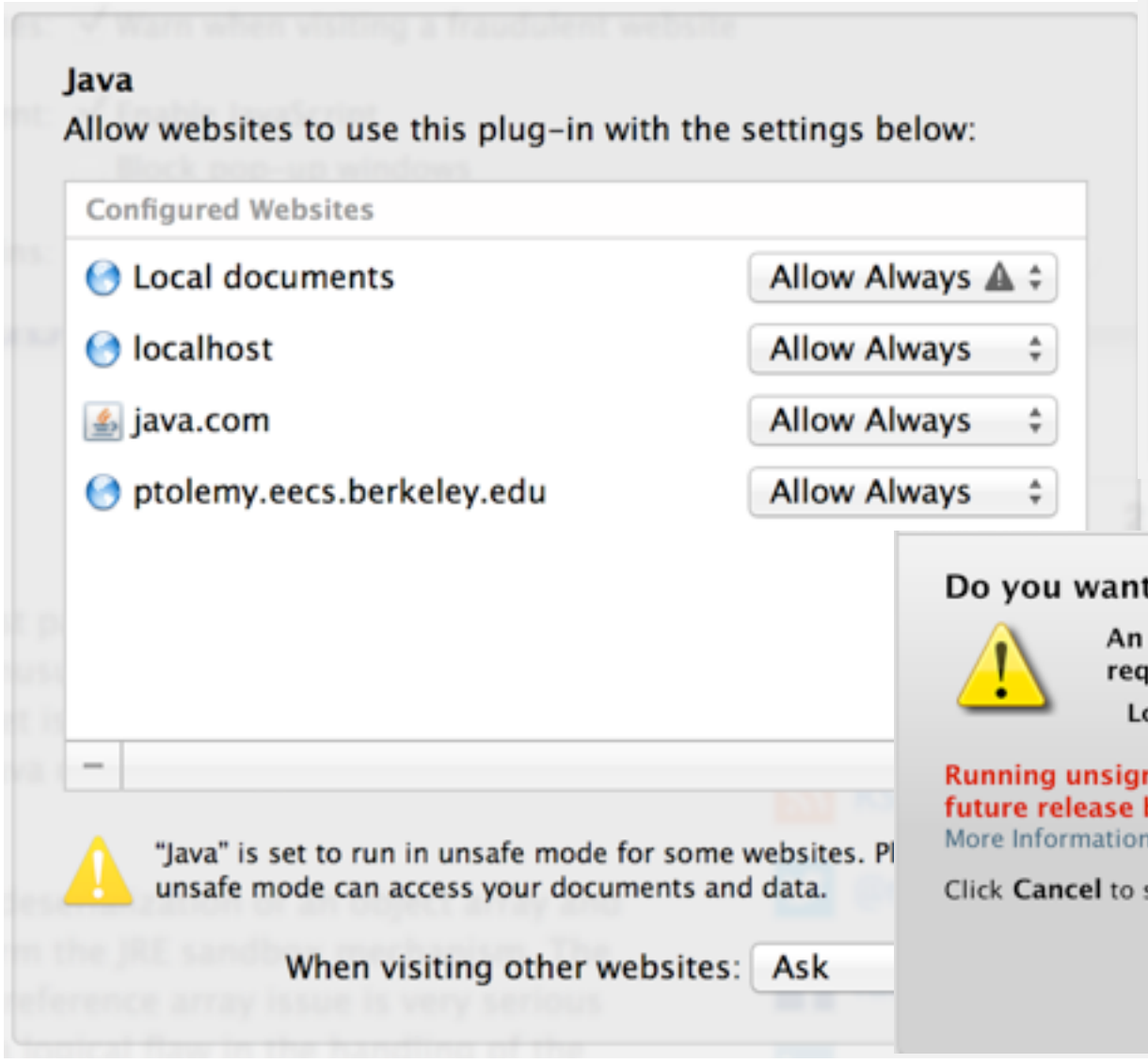
```
<html>
  <head>
     <title> Hello World </title>
  </head>
  <body>

     The Hello-World example applet is called: <br>
     <object type="application/x-java-applet"
      height="100" width="400">
          <param name="code" value="HelloWorldApplet" /
  >
     </object>
  </body>
</html>
```

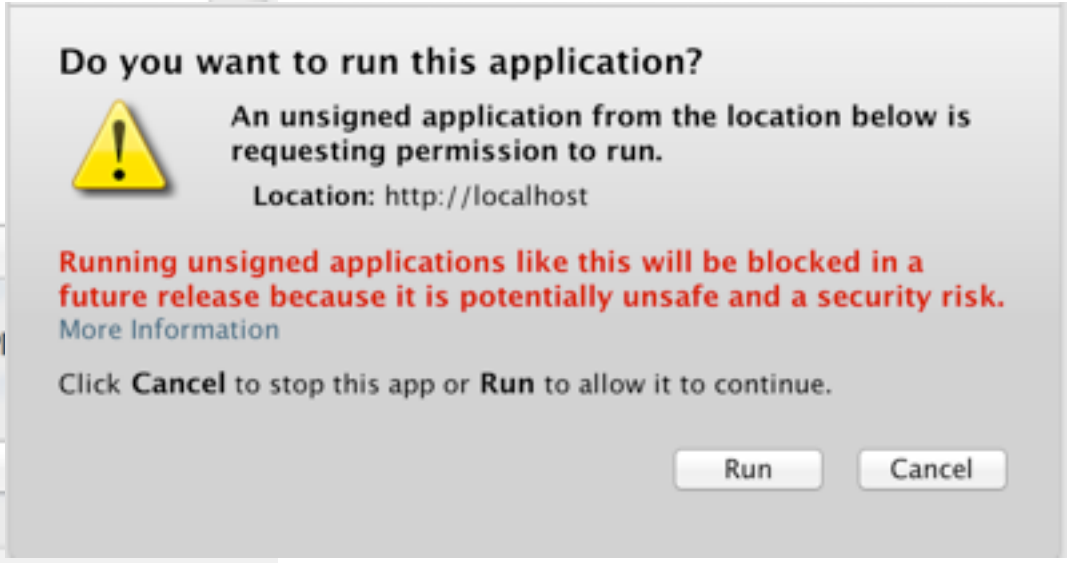
Deprecated HTML:
<applet>

Executes "HelloWorldApplet.class"

java/Applets/HelloWorldNew.html

Typical Security Precautions



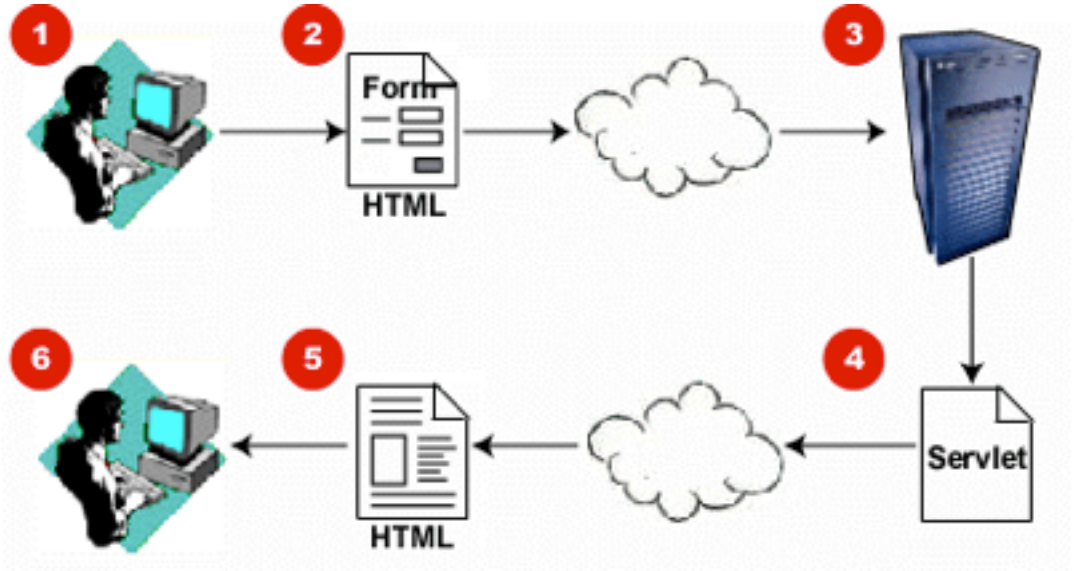


4 Technology Evolution for Web Applications

Literature:

http://java.sun.com/products/servlet/docs.html
http://glassfish.java.net/

Basic Principle: Server-Side Execution



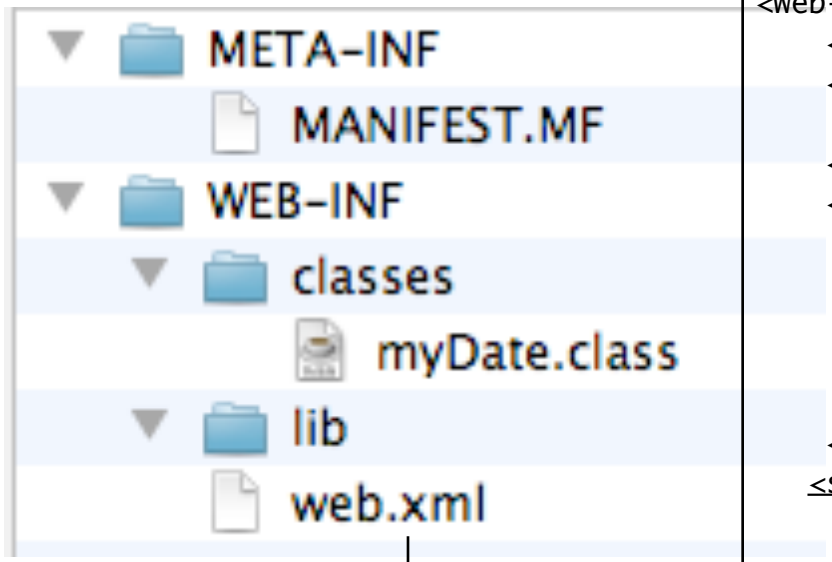
Servlet container needed on server: e.g. Glassfish, Tomcat

1. User fills form
2. Form is sent as HTTP request to server
3. Server determines servlet program and executes it
4. ***Servlet*** computes response as HTML text
5. Response is sent to browser
6. Response, as generated by servlet, is displayed in browser

Example: Hello-World Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
      throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

File Structure for Deployment



Meta information

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>My little Date Application</display-name>
    <description>
      Small demo example, by Heinrich Hussmann, LMU.
    </description>
    <context-param>
      <param-name>webmaster</param-name>
      <param-value>husmann@ifi.lmu.de</param-value>
      <description>
        The EMAIL address of the administrator.
      </description>
    </context-param>
   <servlet>
      <servlet-name>myDate</servlet-name>
      <description>
            Example servlet for lecture
      </description>
       <servlet-class>myDate</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>myDate</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>
    <session-config>
      <session-timeout>30</session-timeout>    <!-- 30 minutes -->
    </session-config>
</web-app>
```

Web application archive:

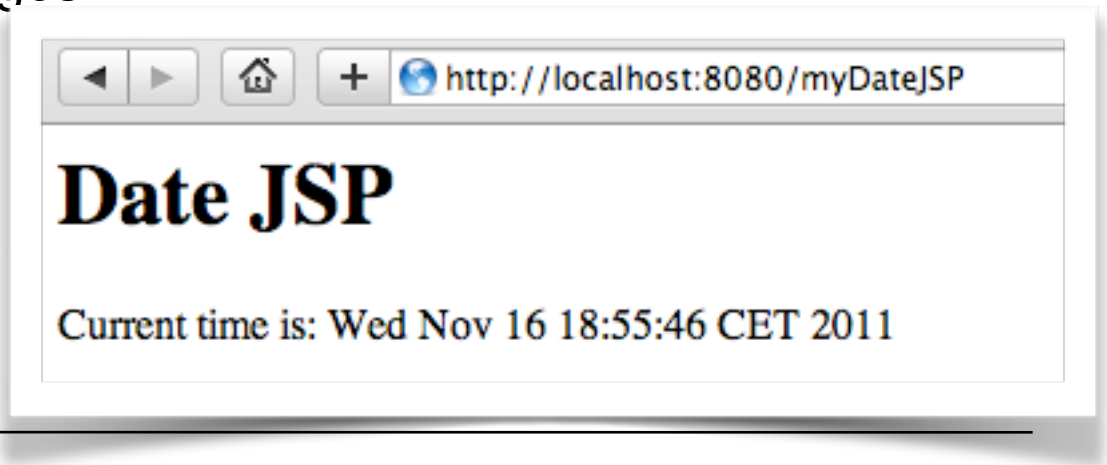
- single archive file with ".war" extension ()

Markup for Servlet Generation: Java Server Pages (JSP)

HTML page with current date/time

```
<html>
<%! String title = "Date JSP"; %>
<head><title> <%=title%> </title></head>
<body>
<h1> <%=title%> </h1>
<p>Current time is:
<% java.util.Date now = new GregorianCalendar().getTime(); %>
<%=now%></p>
</body></html>
```

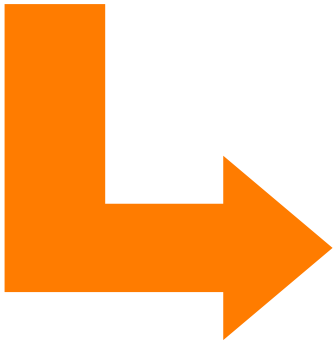
- Basic idea for Java Server Pages:
  - Scripts embedded in HTML ("*Scriptlets*")
  - Automatic translation into Java Servlet code



Java HTML

Generated Servlet Code (Excerpt)

```
<html>
  <%! String title = "Date JSP"; %>
  <head>
     <title> <%=title%> </title>
  </head>
  <body>
     <h1> <%=title%> </h1>
     <p>Current time is:
          <% java.util.Date now = new
  GregorianCalendar().getTime(); %>
          <%=now%>
  </body>
</html>
```

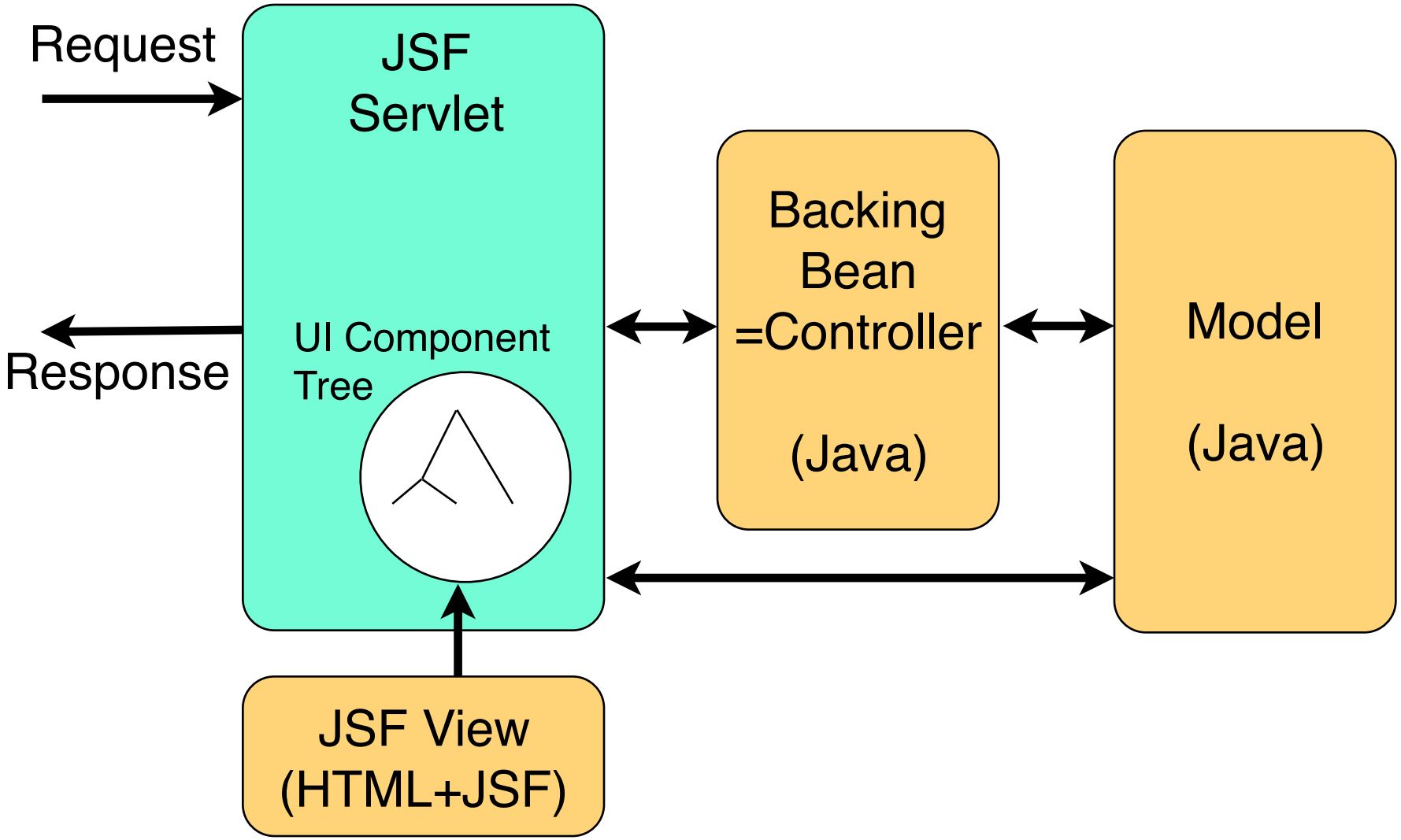


```
...
out.write("\r\n");
out.write("\t<body>\n");
out.write("\t\t<h1> ");
out.print(title);
out.write(" </h1>\n");
out.write("\t\t<p>Current time is:\n");
out.write("\t\t\t");
java.util.Date now = new GregorianCalendar().getTime();
out.write("\n");
out.write("\t\t\t");
out.print(now);
out.write("\n");
```

Java Server Faces (JSF)

- Java framework for building Web applications
  - Latest version 2.2 (2013)
    - » 2.0 was a heavy update to previous version
- JSF can be used together with JSP (and other technologies), but also as a separate tool for creating dynamic Web applications
  - JSF is likely to replace JSP in the future
- One single servlet: FacesServlet
  - loads view template
  - builds component tree mirroring UI components
  - processes events
  - renders response to client (mostly HTML)
- JSF follows a strict Model-View-Controller (MVC) architecture

Counter with JSF's MVC Architecture



Example: View for Counter

```
<!DOCTYPE html>
<html lang="en"
      xmlns:f="http://java.sun.com/jsf/core
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets">
  <h:head>
    <title>Counter with Java Server Faces</title>
  </h:head>
  <h:body>
    <h2>Counter with JSF</h2>
    <h:form>
      <h:panelGrid columns="2">
        <h:outputLabel for="ctrvalue">Counter value = </h:outputLabel>
        <h:outputText id="ctrvalue" value="#{counterController.counterBean.count}"/>
        <h:commandButton value="Count" action="#{counterController.submitCount}" />
        <h:commandButton value="Reset" action="#{counterController.submitReset}" />
      </h:panelGrid>
    </h:form>
  </h:body>
</html>
```

JSF Expression Language
(Extension of JSP-EL)

counter.jsf (or .xhtml)

4 Technology Evolution for Web Applications

Execution Architecture

- Performance and security are most important
  - Asynchronous, non-blocking request handling
  - Pooling resources for re-use
- Client vs. server / Thin vs. Fat
  - Web applications support any style
  - Recent tendency towards fat (JavScript) code on generic clients (browsers)
  - Hardware and context dependencies: Mobile devices, IoT
- Scalable, distributed software architectures
  - Examples: NodeJS, Non-SQL databases
  - But also: Java Enterprise!

Language / Development Environment

- Universal trend from procedural to declarative
  - See Express with templates, AngularJS, Web components, JSP/JSF
- Reduction of number of different languages
  - SQL, server-side scripts replaced by JavaScript
  - HTML style markup (with components) to encapsulate complex constructs (Web components, but also JSP/JSF)
- Structured applications
  - Many files, directory structure, conventions, "manifest", generator tools (Express, Angular, Enterprise Java)
  - Model-View-Controller (and similar patterns): separation of concerns
- Outlook:
  - Pretty unclear
  - Speed of development is even increasing…