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
  - Java for demanding Web programs
  - JavaScript for simple browser interactivity
- 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

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/

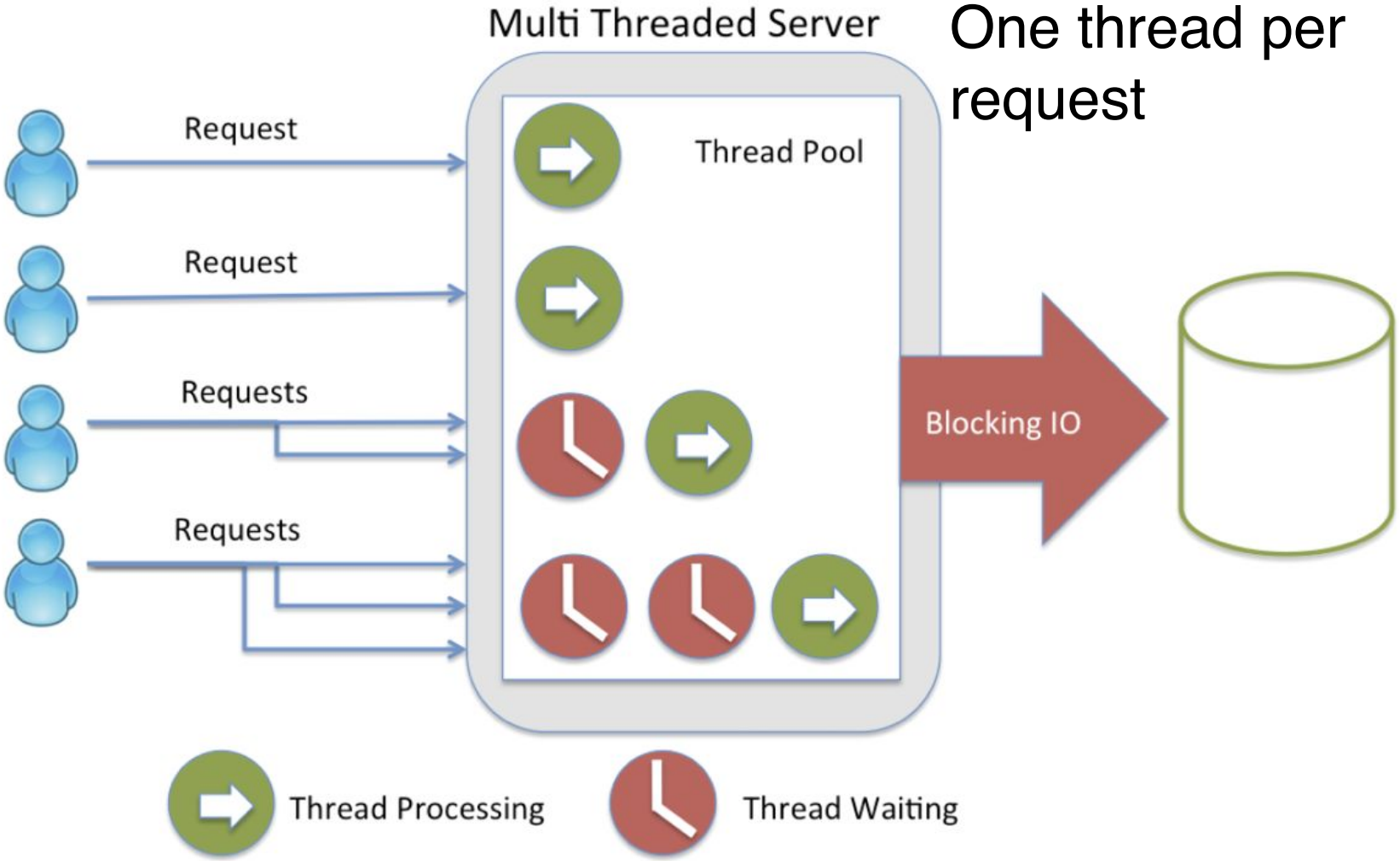
I/O Blocking And Restaurant Counters

http://www.theheroofcanton.com/local-restaurants/papa-gyros-canton-ohio/

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…

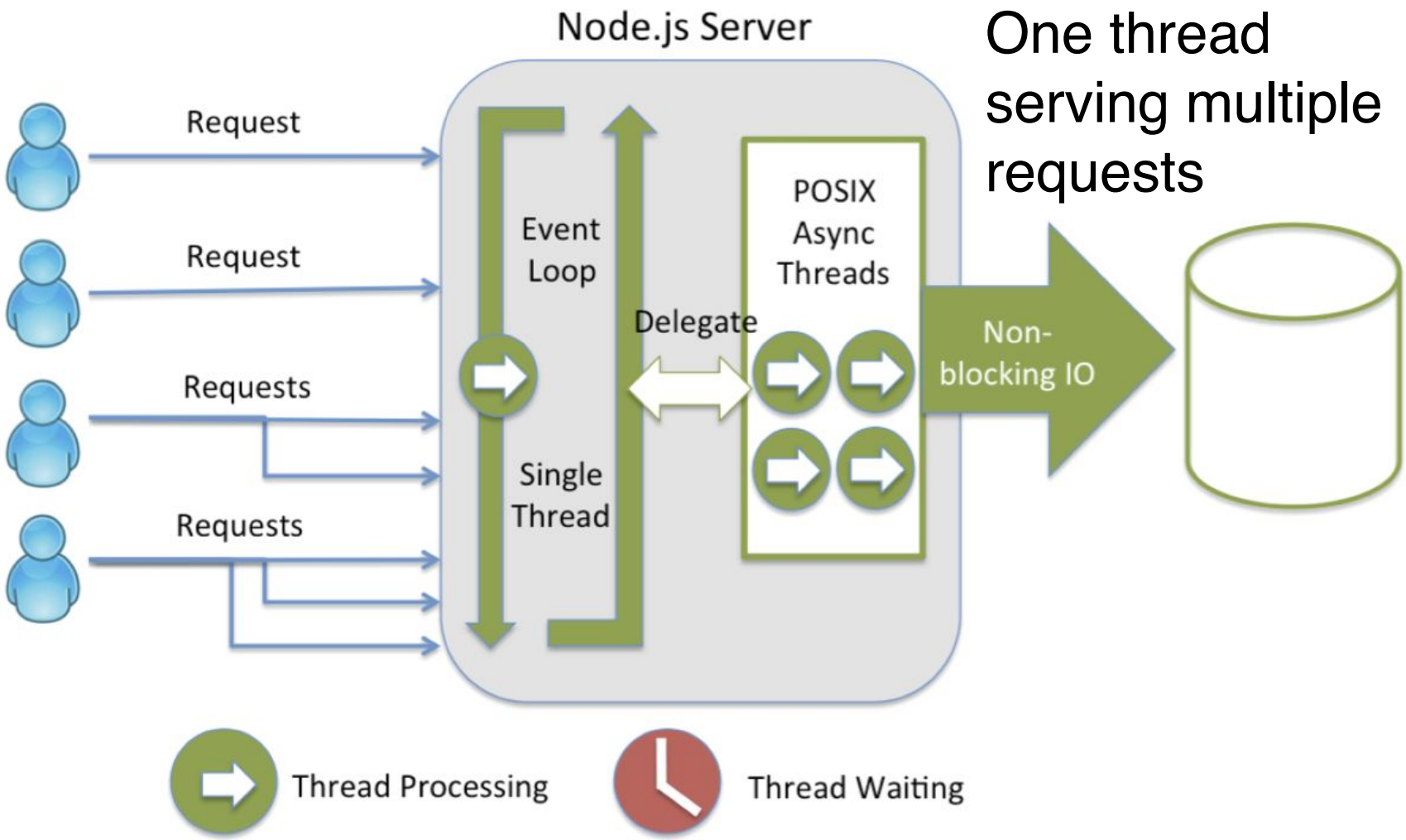
Blocking (Synchronous) I/O Model



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 used version (ECMAScript 5)
  - To be added in ECMAScript 6 ("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/');
```

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`**

Accessing a (NoSQL) Database from Node

```
var express = require('express');
var mongo = require('mongodb').MongoClient;

var app = express();

var dburl = 'mongodb://localhost:27017/music';

app.get('/', function (request, response) {
  var reqTitle = request.query.title;
  comsole.log("The title given was: "+reqTitle);
  mongo.connect(dburl, function(err, db) {
       var collection = db.collection('mysongs');
       var resCursor = collection.find({title: reqTitle});
       … Processing query results …
       db.close();
  });
})
.listen(3000);
```

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

Template Engines

- How to create HTML code from a pure program running on the server?
- Support needed for
  - HTML low-level syntax (angle brackets etc.)
  - HTML structure (tag level)
- Traditional approach:
  - HTML page with embedded server-side scripts
  - Supported e.g. by PHP
  - Also supported by variants of server-side JavaScript (e.g. Embedded Java Script style of Node.js)
- More radical approach:
  - Make HTML syntax tree a data structure on the server
  - Current fashion: "Jade templating engine"

Examples of Jade Templates

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

layout.jade

songlist.jade

```
extends layout

block content
    h1.
        Song List
    ul
        each song, i in songlist
            li
                song.title
```

Thin and Thick Clients

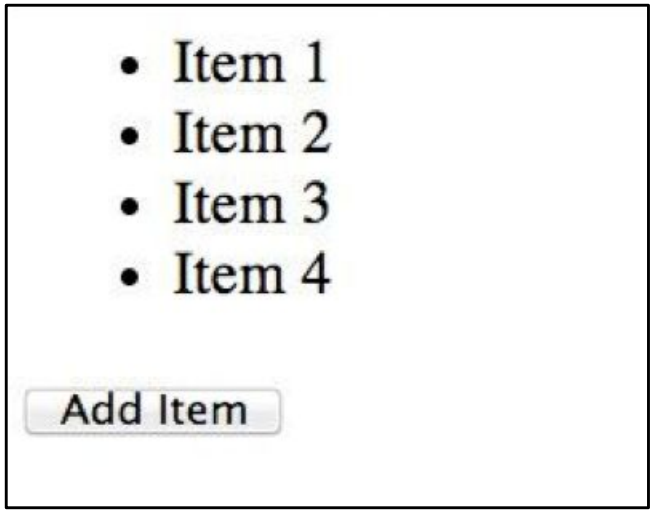
- Thick clients (old style):
  - Traditional client.side programs
  - Need to be installed, updated, maintained
- Thin clients (old style):
  - Client contains basic virtual machine (e.g. based on Java Byte Code)
  - Client loads all software from server
- Thin clients (modern style) – e.g. NodeJS/Express:
  - Client runs Web browser (with JavaScript)
  - Logic is distributed between client and server
  - Client gets HTML code with embedded JavaScript from server
- Thick clients (modern style):
  - Client runs JavaScript engine (in Web browser or OS – e.g. ChromeOS)
  - Client loads large JavaScript code when starting application
  - Client communicates with server asynchronously
    - » AJAX style on client, Node style on server

Client-Side JavaScript Frameworks

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

jQuery vs. Angular, jQuery Version

- jQuery: Mainly declarative
  - 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>
```

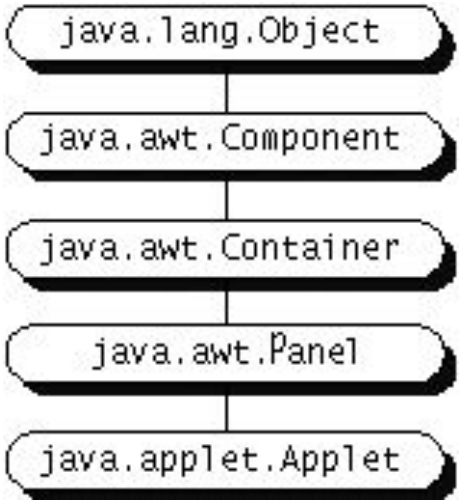
Controller:

```
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');
  });
});
```

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

Parameter Passing in HTML

Applet:

```
public class HelloWorldAppletParam extends Applet {

    public void paint(Graphics g) {
      String it = getParameter("insertedtext");
      g.setFont(new Font("SansSerif", Font.PLAIN, 48));
        g.drawString("Hello "+it+" world!", 50, 50);
    }
}
```

HTML:

This is modern HTML5.

```
<html>
  ...
      <br>
      <object type="application/x-java-applet"
        height="100" width="800">
          <param name="code" value="HelloWorldAppletParam" />
          <param name="insertedtext" value="wonderful" />
          Java Applets not supported.
      </object>
  ...
</html>
```

Java/Applets/HelloWorldParamNew.html

User Interaction in Applets

- Applets are able to react to user input
  - Define an event handler
  - Register during applet initialization (init())
- Applets are executed locally, and therefore have full access to local input
  - Mouse movements, key press, …
  - This is not possible with server-side code!
- Applets can make use of graphics libraries
  - For instance Java 2D
  - This is not easily possible with server-side code!

Java/Applets/ClickMe.html

Organisation of Bytecode Files

- **`<object>`** and **`<applet>`** tags allow
  - Declaration of a "codebase" directory (attribute **`codebase`**)
  - Declaration of a Java archive (JAR) file (attribute **`archive`**)
- Advantages of codebase:
  - Java bytecode concentrated at one location
  - Fits with Java file conventions
- Advantages of archives:
  - Less files, less HTTP connections, better performance
  - Lower bandwidth requirements due to (LZW) compression

Cubic.html

Applets and Security

- "Sandbox security":
  An applet is not allowed to
  - Open network connections (except of the host from which it was loaded)
  - Start a program on the client
  - Read or write files locally on the client
  - Load libraries
  - Call "native" methods (e.g. developed in C)
- "Trusted" Applets
  - Installed locally on the client, or
  - Digitally signed and verified
  - Such applets may get higher permissions, e.g. for reading/writing files
- Execution of applets from locally loaded files is restricted
  - Recent addition
  - Therefore, avoid local loading for tests!

Advantages and Disadvantages of Java Applets

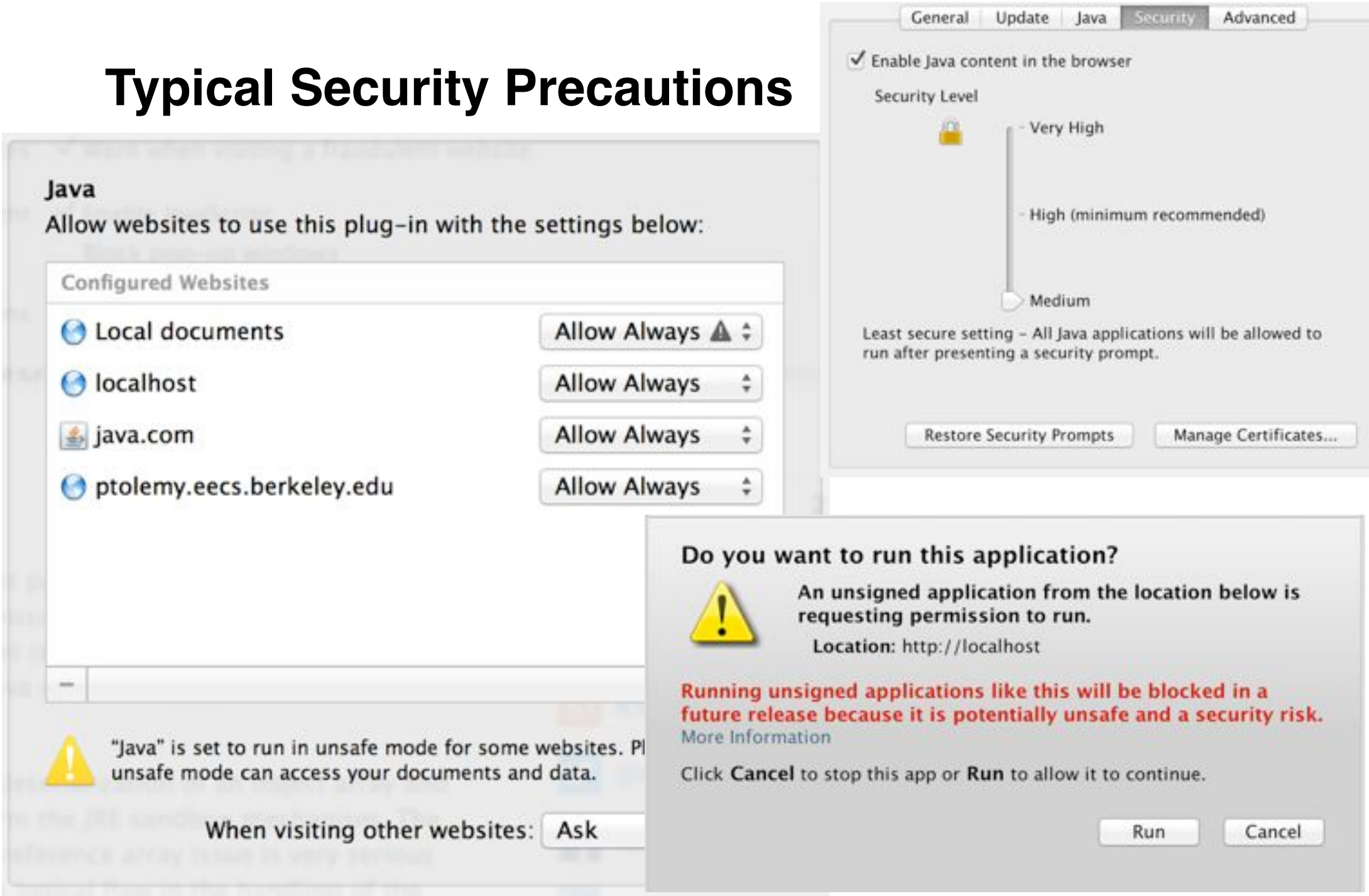
- Advantages:
  - Interaction
  - Graphics programming
  - No network load created during local interactions
  - Executed decentrally – good scalability
- Disadvantages:
  - Dependencies on browser type, browser version, Java version
    - » Persisting problem, leading to many incompatibilities, including recent Java 7 problems
  - Debugging is problematic
  - Java-related security problems (sandbox breaches)

Flashback Trojan botnet infects 600,000 Macs

05.04.2012

siliconrepublic.com

Typical Security Precautions

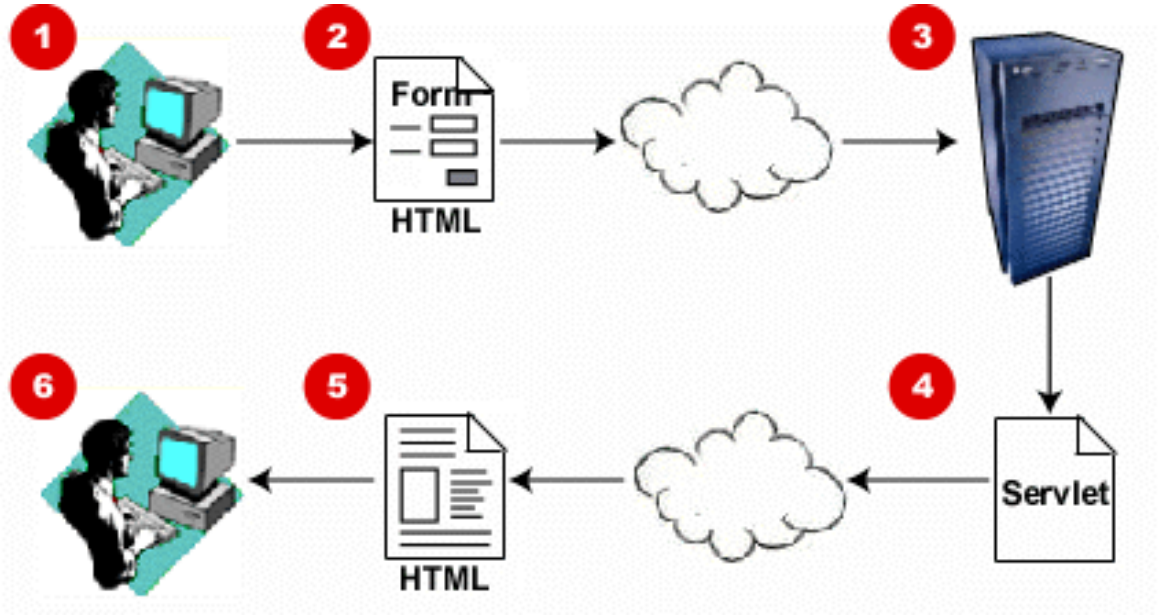


4 Technology Evolution for Web Applications

Literature:

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

Basic Principle: Server-Side Execution



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

Java Servlets

- Java Servlet Specification (JSS):
  - Part of Java Enterprise Edition (EE)
  - First version: 1996 (Java: 1995)
  - Current version: 3.1 (May 2013, with Java EE 7)
  - Java Server Pages: 1997–1999
  - Java Server Faces: 2001 – … (Version 2.2: May 2013)
- Reference implementation for a “servlet container”:
  - “GlassFish” (Oracle)

- Other well-known Java EE servers:
  - Apache Tomcat (Catalina), BEA Weblogic (now Oracle), JBoss, jetty
- Basic principle very similar to PHP:
  - Web server calls (Java) servlet code on request from client
  - Servlet computes text response to client
    - » HTML page or data to be used in AJAX requests

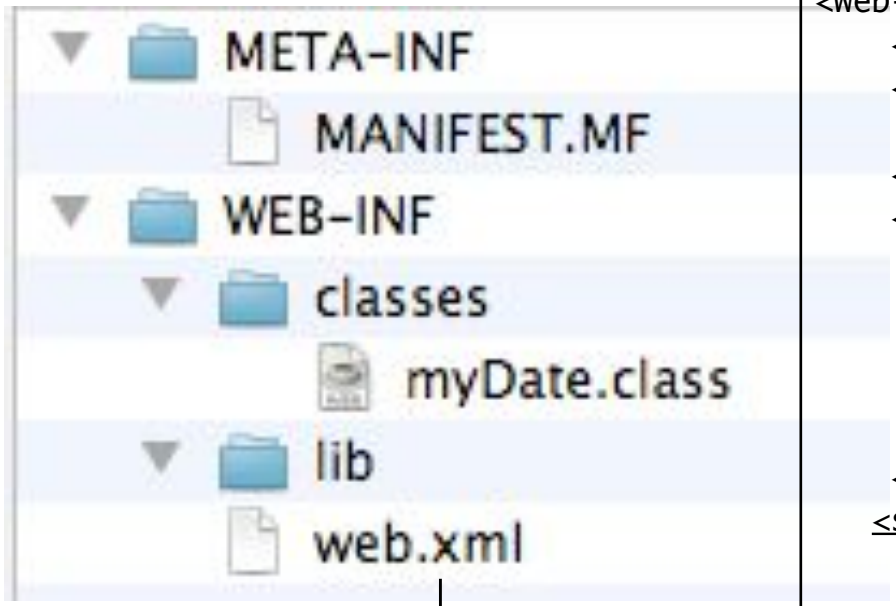
Example: Hello-World Servlet

```
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 ()

4 Technology Evolution for Web Applications

Literature:

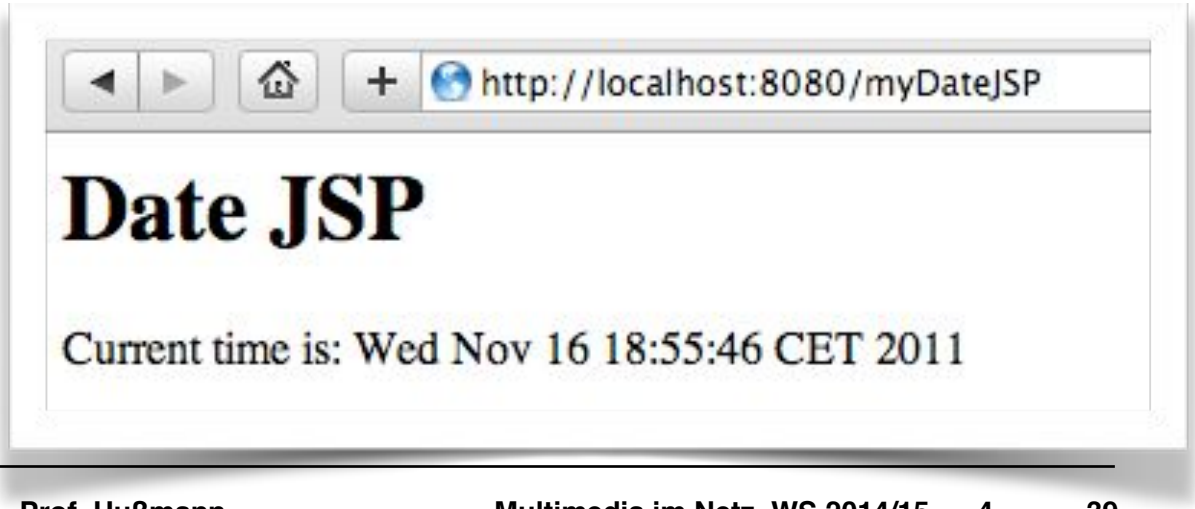
http://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html
https://javaserverfaces.java.net
M. Kurz, M. Marinschek: Java Server Faces 2.2, dpunkt, 3. Auflage 2013

Introductory Example: Java Server Page (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>
```

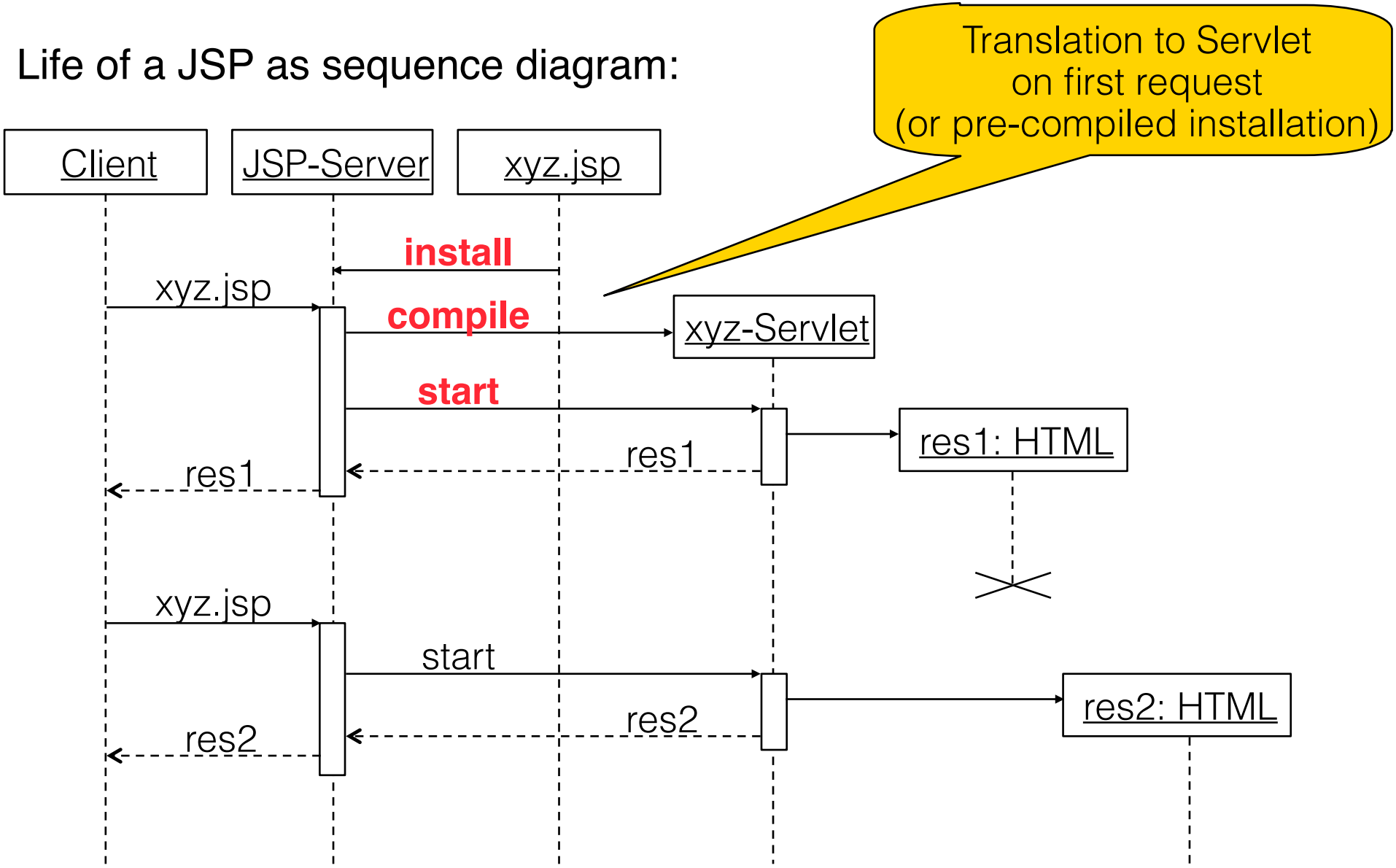
- Basic idea for Java Server Pages:
  - Scripts embedded in HTML ("*Scriptlets*")
  - Automatic translation into Java Servlet code



Java HTML

Java Server Pages und Servlets

Life of a JSP as sequence diagram:

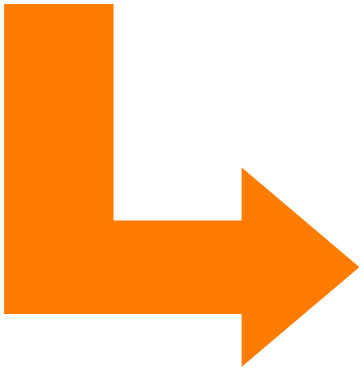


JSP Script Elements

- Declarations
  - Syntax: `<%! declarations %>`
    `<jsp:declaration> declarations </jsp:declaration>`
  - Example: `<%! String title = "Date JSP"; %>`
  - Is translated into instance variable of generated class, i.e. visible in all methods of the class.
- Anweisungen (*Scriptlets*)
  - Syntax: `<% commands %>`
    `<jsp:scriptlet> commands </jsp:scriptlet>`
  - Example:
    ```
    <% java.util.Date now = new
    GregorianCalendar().getTime(); %>
    ```
  - Local variables are not visible in other methods.
- Expressions
  - Syntax: `<%= expression %>`
    `<jsp:expression> expression </jsp:expression>`
  - Example: `<%= now %>`
  - Equivalent to `<% out.print(now); %>`

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>
```

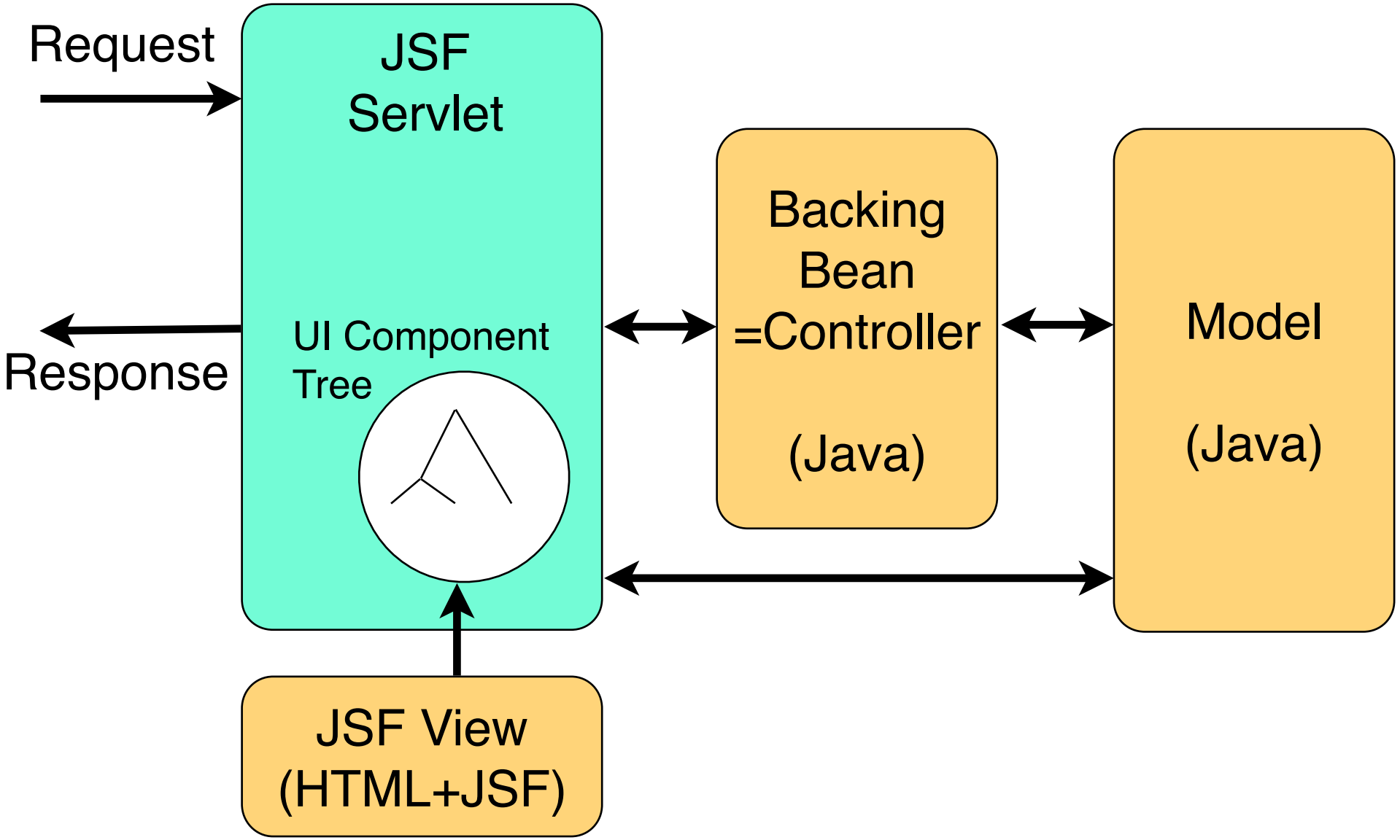


```
...
out.write("\r\n");
out.write("\t<body>\n");
out.write("\t\t<h1> ");
out.print(title);
out.write(" </h1>\n");
out.write("\t\t<p>Current time is:\n");
out.write("\t\t\t");
java.util.Date now = new GregorianCalendar().getTime();
out.write("\n");
out.write("\t\t\t");
out.print(now);
out.write("\n");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

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



What Is a JavaBean?

- JavaBeans is a *software component* model for Java
  - Not to be confused with Enterprise Java Beans (EJBs)!
- Software components:
  - Units of software which can be stored, transmitted, deployed, configured, executed without knowing the internal implementation
  - Main usage: Tools for composing components
- Driver for JavaBeans technology: User Interfaces
  - AWT and Swing components are JavaBeans
  - GUI editing tools instantiate and configure JavaBeans
- Main properties of a JavaBean:
  - Has a simple constructor without parameters
  - Provides public getter and setter methods for its properties: **`getProp, setProp`** (no setter = read-only)
  - Is serializable
  - Supports listener mechanism for property changes
    - » *Bound* properties: provide listener for changes
    - » *Constrained* properties: allow listeners to *veto* on changes

JavaBeans in JSP: Action useBean

- Syntax of useBean Aktion:

```
<jsp:useBean id=localName class=className
             scope=scopeDefn />
```

  scope: "page“ (current page), "request" (current request); "session" (current session), "application" (full application)

- Reading properties:

```
<jsp:getProperty   name=localName
                   property=propertyName/>
```

- Writing properties:

```
<jsp:setProperty   name=localName
                   property=propertyName/>
                   value=valueAsString
```

```
<jsp:getProperty   name="counter" property="current"/>
```

is equivalent to:

```
<%=counter.getCurrent();%>
```

Counter as JavaBean (1)


```
package counter;
import java.beans.*;

public class Counter extends Object implements java.io.Serializable {

    private static final String PROP_CURRENT = "current";
    private static final String PROP_START_VALUE = "start value";
    private static final String PROP_INCR_VALUE = "incr value";
    private static final String PROP_ENABLED = "enabled";
    private int count;
    private int startValue;
    private int incrValue;
    private boolean enabled;
    private PropertyChangeSupport propertySupport;

    public Counter() {
        propertySupport = new PropertyChangeSupport ( this );
        startValue = 0;
        incrValue = 1;
        reset();
        enabled = true;
    }
```

Counter as JavaBean (2)

```
public int getCurrent () {
    return count;
}

public int getStartValue () {
    return startValue;
}

public void setStartValue (int value) {
    int oldStartValue = startValue;
    startValue = value;
    propertySupport.firePropertyChange
      (PROP_START_VALUE, oldStartValue, startValue);
}

public int getIncrValue () {
    return incrValue;
}

public void setIncrValue (int value) {
    int oldIncrValue = incrValue;
    incrValue = value;
    propertySupport.firePropertyChange
      (PROP_INCR_VALUE, oldIncrValue, incrValue);
}
```

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)

Example: Controller for Counter (1)

```
package counter;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;

@ManagedBean
@ViewScoped
public class CounterController implements java.io.Serializable {

    private static final long serialVersionUID = 11L;

    private CounterBean counter;

    public CounterController() {
        counter = new CounterBean();
    }

  ...
```

Bean instance name is automatically created from class name (by making lowercase the first letter)

Example: Controller for Counter (2)

```
...
    public void submitCount() {
        counter.count();
    }

    public void submitReset() {
        counter.reset();
    }

    public CounterBean getCounterBean() {
        return counter;
    }

}
```

Advantages of JSF

- Clean separation of concerns
  - View (JSF/HTML) just specifies appearance
    - » View contains markup only (+references to beans)
  - Model is clearly separated
    - » Usually, access to persistence layers, databases etc.
  - Usage of controller is enforced
    - » Controller is simple Java Bean
- JSF is fully integrated into Java EE architecture
  - Supports load sharing, transactions etc.
- JSF Tag Libraries
  - Extensibility, variations in authoring style

AJAX Support in JSF

- Special tag in JSF Core Library:

```
<f:ajax>
```

- Ajax tag modifies reaction of components in which it is embedded
  - XMLHttpRequest instead of browser-global request updating the view
- Example:

```
<h:inputText ...>
    <f:ajax event="valueChange"
      execute="handler"
      render= … />
</h:inputText>
```

- On value change event (input into text field), specified handler is called
  - on the server, of course (= asynchronous handling of input)
- The *render* attribute can be used to specify the components to be updated after event processing
  - For instance by specifying document parts using “id”s