openHPI Web Technologies 2015
Week 5

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

openHPI

Server-Side Web Programming

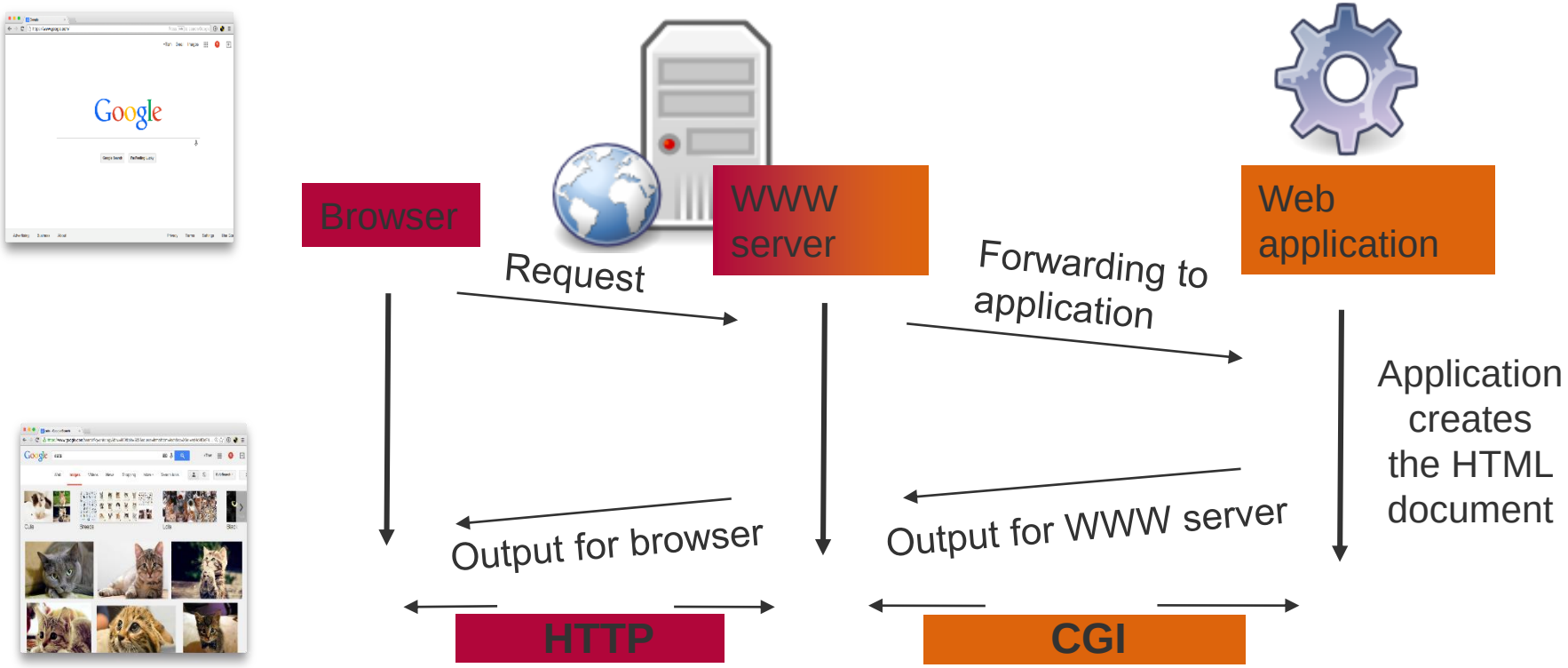
Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

Server-Side Web Programming

- Client activates via its Web browser a web application on server-side
- HTTP server initiates the execution on the server machine
- Results are passed to the client / browser as dynamically generated HTML document
- There is a standardized interface between HTTP server and web application at server-side: **CGI interface**
- Web applications at the server-side can be written in any programming language , e.g.
  - Java (Servlets, Server Pages, Beans, ...)
  - Scripting languages (ASP, PHP, Perl, Python, Ruby, ...)
  - Today: Web frameworks (Ruby on Rails, Spring, Django, …)

CGI Interface (1/7)

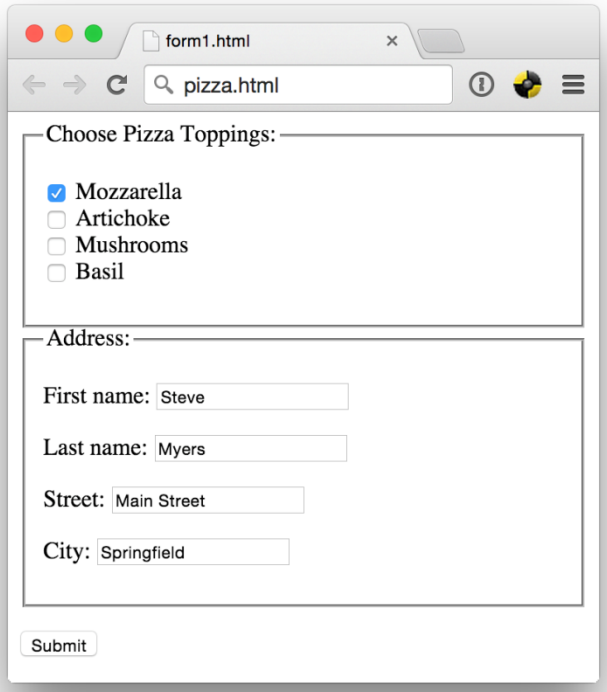
HTTP server provides a standard interface for server-side programs that enables the dynamic generation of the HTML document:
CGI - Common Gateway Interface



CGI Interface (2/7)

The HTTP server controls the web application by passing parameters

- **Example**: Completion and submission of an HTML form




```
<form name="PizzaPizza" action="order.php"
 method="GET">
  <fieldset>
    <legend>Choose Pizza Toppings:</legend>
    <p>
    <input type="checkbox"
     name="topping" value="mozzarella"/>
     Mozzarella<br>
     …
  </fieldset>
  …
</form>
```

CGI Interface (3/7)

Parameters are transferred in different ways to the server:

GET

- Form data is appended to the URL (specified in the `action` attribute) behind a "?" as key value pairs, e.g.

  → https://pizza.com/order.php?topping=mozzarella&firstname=Steve..

- HTTP server transfers data to the CGI program which is invoked over the given URL in the environment variable `QUERY_STRING`

POST

- An HTTP request to the URL in the `action` attribute is initiated
- Form data is transferred in the body of the HTTP request
- Body of the HTTP request is passed to the CGI program which is invoked by calling the given URL via the standard input

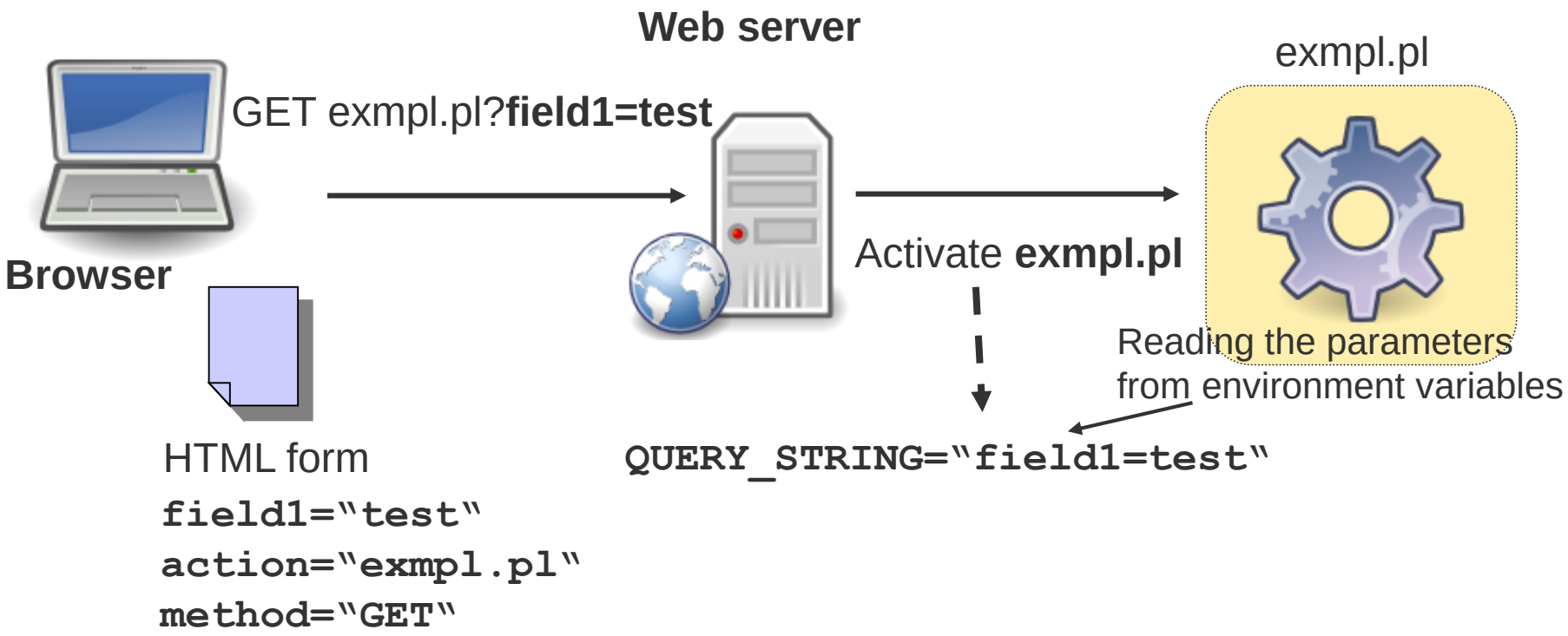
CGI Interface (4/7)

Environment variables of the HTTP server

- Server sets certain environment variables before calling the CGI program, e.g.
  - `CONTENT_LENGTH`, `SERVER_NAME`, `REQUEST_METHOD`
  - `PATH_INFO`, `SCRIPT_NAME`, `QUERY_STRING` ...
- CGI can access and evaluate the environment variables
  - Examples – display the port number:
    - Perl: `print "SERVER_PORT = $ENV{SERVER_PORT}\n";`
    - Ruby: `<%= "SERVER_PORT =" + request.env["SERVER_PORT"] %>`
    - PHP: `<?php echo "SERVER_PORT =".$_SERVER["SERVER_PORT"] ?>`
- CGI can access the data sent by the client via GET or POST
  - Example PHP: `<?php print_r($_POST) ?>`

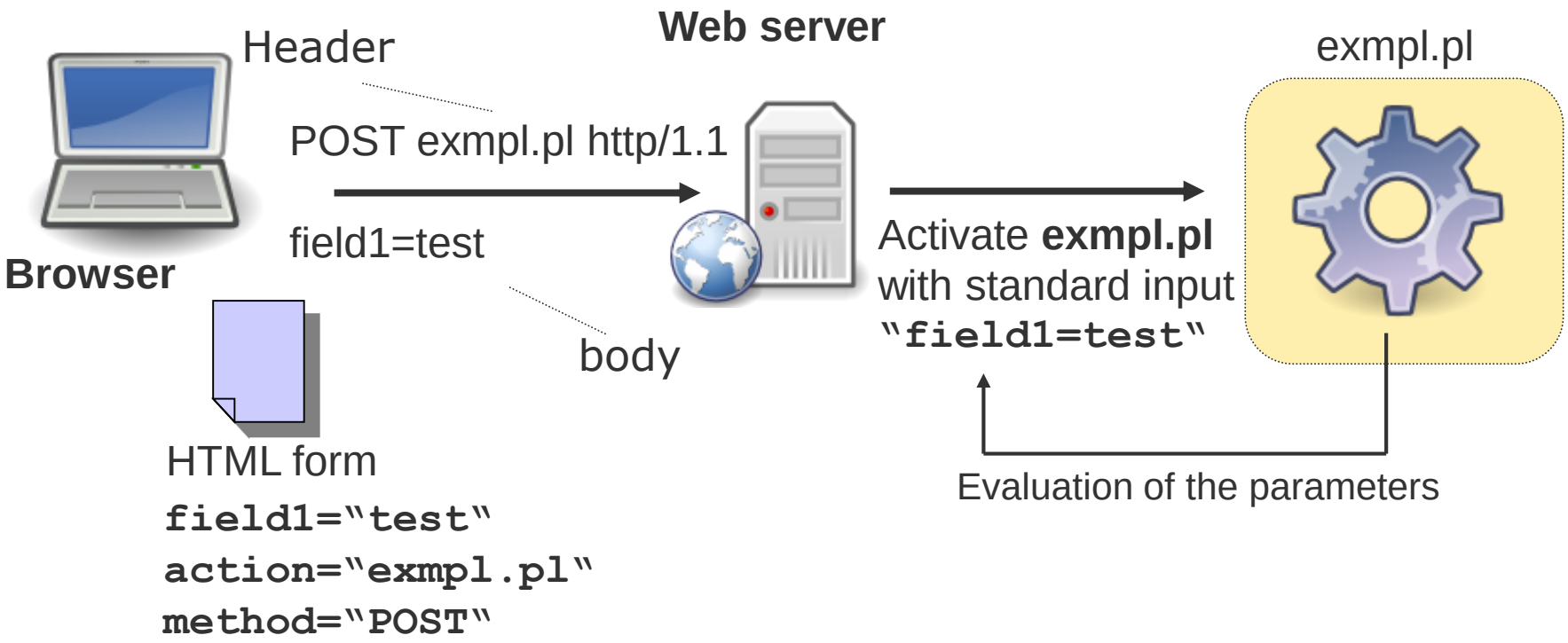
CGI Interface (5/7)

Handover parameters with GET



CGI Interface (6/7)

Handover of parameters with POST



CGI Interface (7/7)

Historical Development

- CGI was initially only runtime environment of the web server for external programs (e.g. C, C++)
- Later use of Perl scripts, activation via command line interpreter
- For fast execution:
  - Development of script interpreters as modules in the web server, e.g.
    - Apache: mod_perl, mod_python, mod_php
    - Internet Information Server: ASP 3.0, later ASP.NET
- Parallel: Launch of Java application servers

Today: Besides interpreter modules also new CGI standards, e.g.

- FastCGI, SCGI, WSGI (only for Python)

Server-Side Scripting Languages

Given that the corresponding language module is installed on server

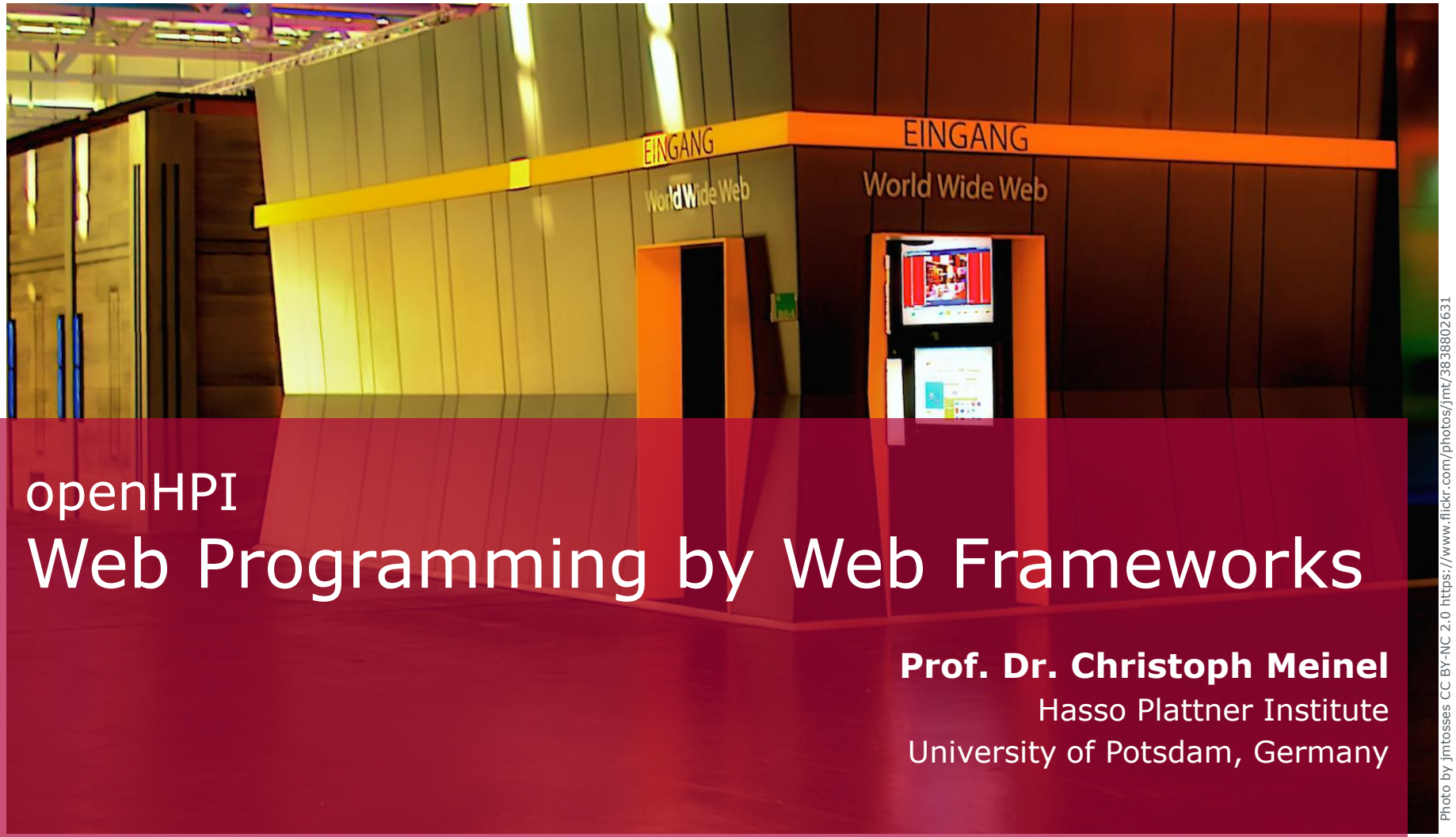
- Code can be integrated directly into HTML via special tags
  - Examples
    - PHP: `<?php echo "Hello PHP!" ?>`
    - Ruby: `<%= "Hello Ruby!" %>`
- If there is no module installed at the server, source code is not executed but displayed as it is

Advantage:

- Fast development through direct manipulation of the HTML/CSS/JS elements

Disadvantage:

- Display and application logic are mixed



openHPI

Web Programming by Web Frameworks

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

Web Programming Today: Web Frameworks (1/4)

Development of Web Programming

- **Pure CGI**: HTML markup is generated by the CGI program, e.g.
  `print("<h1>Headline</h1>");`
- **Server-side scripting**: Special markup for script code within HTML files

  → Example with PHP:

```
<ul>
<?php foreach($itemList as $item) { ?>
    <li><?php echo $item->description; ?></li>
<?php } ?>
</ul>
```

- **Template engines** (e.g. Smarty for PHP): Separation of program / script code and HTML display
- **Modern web frameworks** — often with MVC architecture

Web Programming Today: Web Frameworks (2/4)

Web frameworks offer …

- **Libraries** for (enormously) simplified web programming
- Possibility to **define the data model as an object** with an automatic transfer into a relational database (**ORM**, Object Relational Mapping) → Persistence framework
- Extensive use of **taglibs**, i.e. libraries with prefinished complex HTML control elements, e.g.
  - forms, light boxes, modal boxes, slideshow templates, AJAX requests, …
- …

Web Programming Today: Web Frameworks (3/4)

Web frameworks offer …

- …
- **Scaffolding**: Automatic generation of views and controllers to manipulate the defined model
  - Display of data + CRUD (Create, Read, Update, Delete) operations
- Ready-to-go developer packet
  - Framework, web server, script collection for management, …
- Seamless and transparent integration of JavaScript libraries

… and many others things that make the life of a web programmer easier

Web Programming Today: Web Frameworks (4/4)

Web frameworks follow a number of key concepts:

- Don‘t Repeat Yourself (DRY) – use as little handwritten boilerplate code as possible, better – make use of shared libraries
- Convention over Configuration (e.g. Ruby on Rails, Grails) – as little configuration as possible, instead conventions on directories, file and class names, …
- Keep It Simple, Stupid! (KISS) – simple installation, get started quickly, produce usable results faster

Web frameworks are available for many programming languages, e.g.

- Ruby, Groovy, Java, PHP, Python

Most current frameworks implement **Model-View-Controller – MVC** as architectural pattern

openHPI

MVC – Model-View-Controller

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

MVC – Model-View-Controler Architecture

Most current web frameworks are based on one of the many manifestations of the **Model-View-Controller** architectural pattern

→ Separation of data model (Model), display (View) and control logic (Controller)

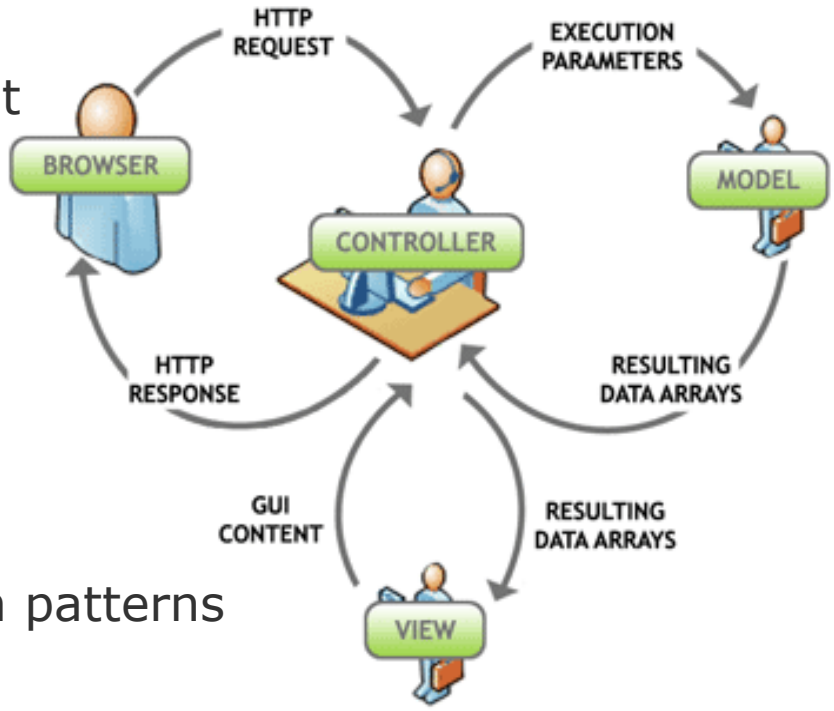
Goals:

- Re-usability and easy replacement of individual components
- Separation of tasks
- Maintainability

Predecessor: Template engines

- Separate view but model and controller are still meshed

MVC – aggregation of several design patterns



MVC – Design Patterns (1/3)

The general concept of design patterns originates in the architecture

- Introduced in the late 1970's – "A pattern language" (C. Alexander)
- Application in software engineering became popular with the book "Design Patterns: Elements of Reusable Object-Oriented Software" by the Gang of Four (GoF) in the 1990's

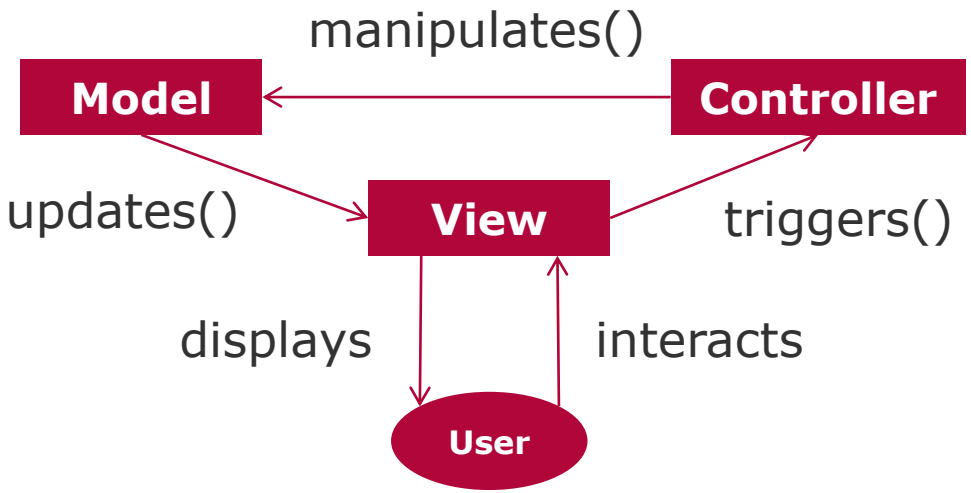
- Describe simple, elegant, reusable solutions to common problems in software design, in general, for object-oriented languages
- Origin in classic software engineering, more recently adopted in the web context
- One of the major goals is the encapsulation of responsibilities

→ Reusability of elements

→ Easy replacement of elements

MVC – Design Patterns (2/3)

A simple implementation of the MVC pattern is shown here:



MVC architectural pattern aggregates multiple **design patterns**

- One of the key elements is the Observer pattern
  - Defines the relation between model and view
  - The views register as observers to the model
  - Whenever the model is updated, all the views are notified about the changes and update themselves accordingly

MVC – Design Patterns (3/3)

Observer Pattern

Subject

state
observers

attach()
detach()
notify()

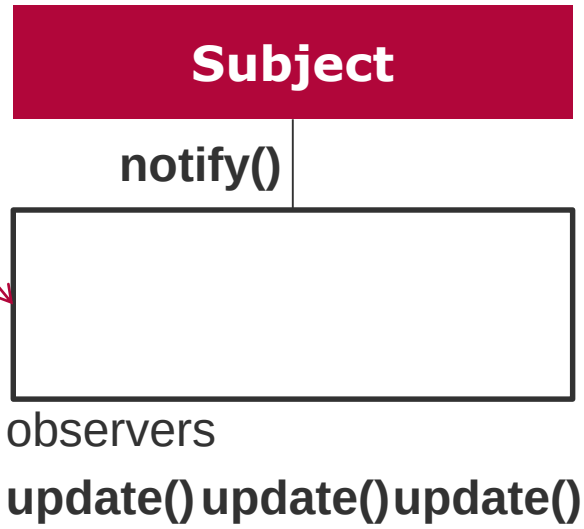
E.g. color

List of attached observers

Add observer to list

Remove observer from list

Call update method on all attached observers



attach()

Observer

state

update()

Set the observer's state to the subject's state

Frameworks Implementing MVC Architecture

- **Traditionally**: Thin client
  - Java ServerFaces, Spring (Java)
  - Cake, Symfony (PHP)
  - Django (Python)
  - Sinatra, Ruby on Rails (Ruby)
- **More Recently**: Rich client
  - AngularJS
  - Ember.js

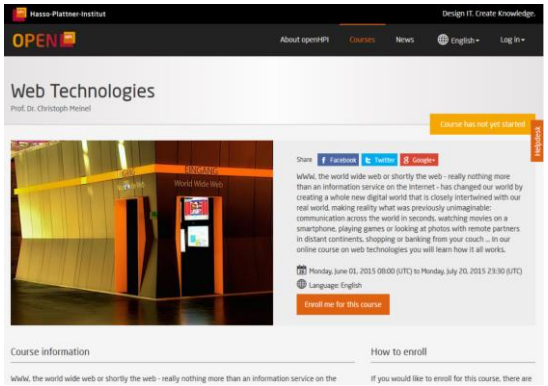
openHPI

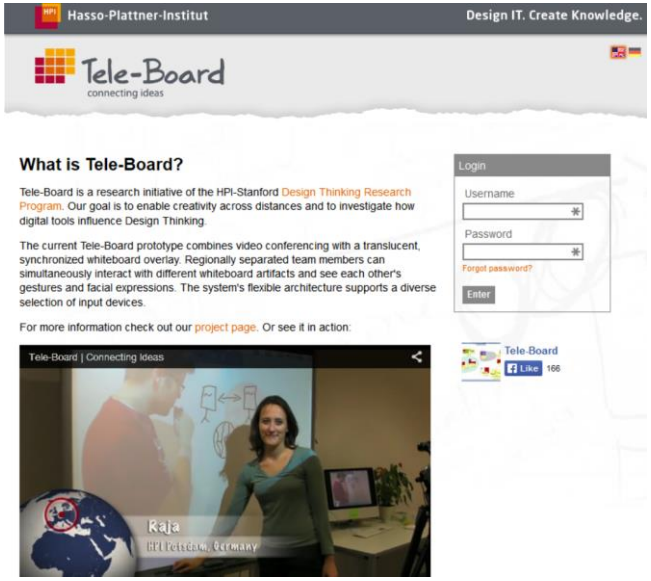
Selected Web Frameworks

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

Selected Web Frameworks

- **Django** (Python) → **tele-TASK**
- **Ruby** on Rails (Ruby) → **openHPI**
- PHP: Zend, Symfony and **CakePHP** → **Tele-Board**
- Grails (Groovy, based on Java platform)
- JavaServer Faces (JSF)
- ASP.NET (all .NET languages)
- …





Django (1/3)

Django - https://www.djangoproject.com

- Started in 2005, current version 1.7.4
- Open source, BSD license
- Based on Python, HTML
- Can be run on a variety of servers
  - Apache
  - nginx
  - Gunicorn
  - Cherokee
  - …

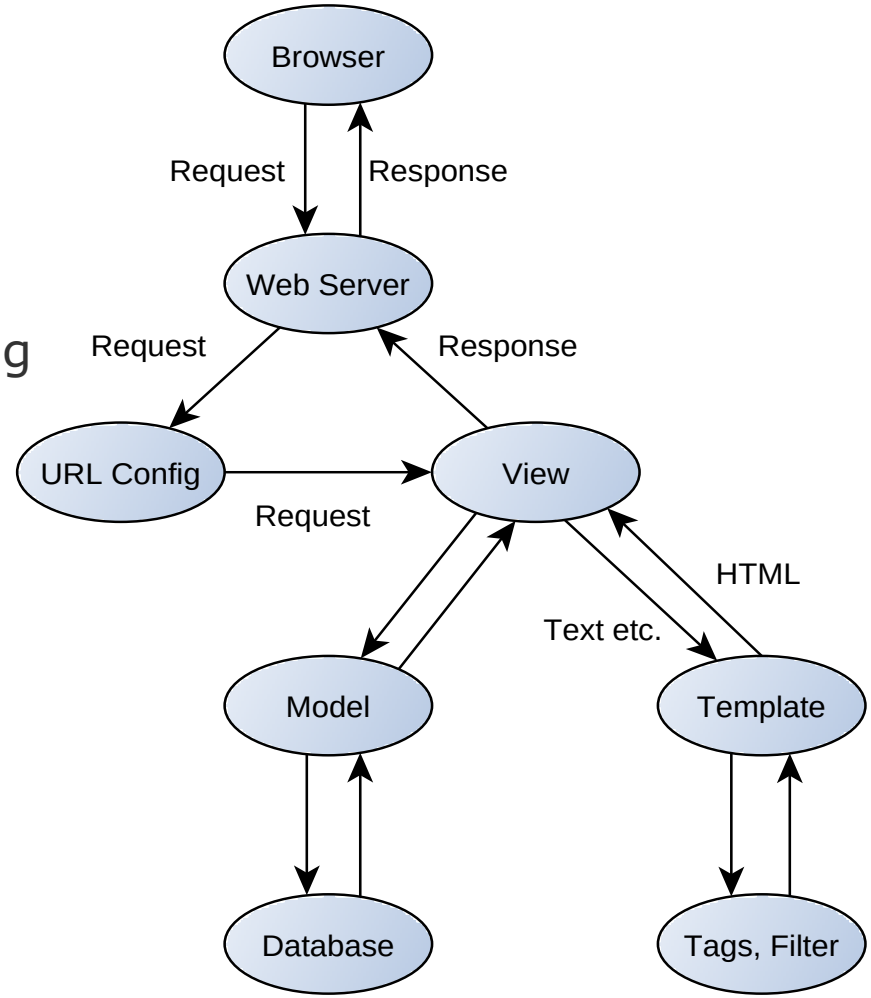
Django (2/3)

Basic philosophy

- DRY – don't repeat yourself
- Explicit is better than implicit
- MVC - model-view-controller architecture with different naming
  - Controller = View
  - View = Template "model-template-view"

Built-in features

- Cache framework
- Security mechanisms
- Forms framework
- Internationalization (i18n)



Django (3/3)

Create a new project:

```
$ django-admin.py startproject mywebsite
```

Files are created automatically:

```
mywebsite/
    manage.py
    mywebsite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

Manage.py – management commands

Settings.py – configuration of paths, database, installed apps, middleware, …

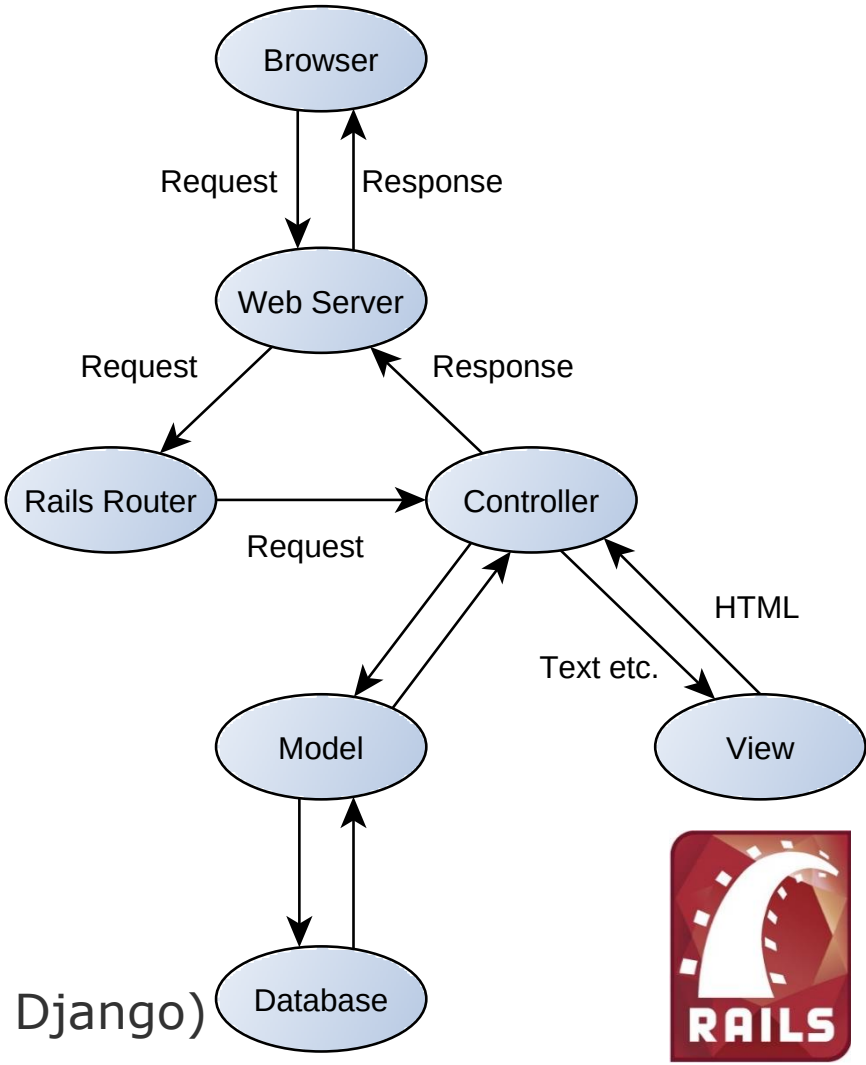
Ruby on Rails (1/3)

Ruby on Rails - http://rubyonrails.org

- Started in 2005, current version 4.2
- Open source, MIT license
- Based on Ruby
- MVC architecture
- Runs on many servers, e.g. Apache, nginx, WEBrick (default for local development), …

Basic philosophy

- DRY – don't repeat yourself
- Fat models, skinny controllers
- Scaffolding
- Convention over configuration (unlike Django)



Ruby on Rails (2/3)

Convention over configuration is a software design paradigm

- Instead of variably configuring things, developers shall stick to conventions for naming objects
- **Examples**:
  - The primary key in a database table should be "ID" with the data type Integer
  - A class "customer" in the model automatically refers to a table "customer" in the database
- **Advantages**:
  - Faster development
  - Code is a bit shorter and simpler
  - Only unconventional parts of the app have to be declared explicitly

Ruby on Rails (3/3)

The DRY principle – Don't Repeat Yourself

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

– Andy Hunt and Dave Thomas: "The Pragmatic Programmer"

- Avoid to duplicate code for doing the same in different parts of the software

Advantages – if DRY is properly applied:

- Necessary changes in the code only have to be done once and do not have to be repeated in several parts of the code
- All related parts are also changed and can not be forgotten
- Code maintenance becomes easier

Violations of the DRY principle are called "**WET** solution" ;-)
- "write everything twice" -

openHPI
Session Management

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

Session Management

HTTP is a stateless protocol, only simple request / response cycles

Problem: Many applications need a session management

- **Session** = Dialogue of multiple requests and responses between client and server → assumes collection and caching of information …
- **Example**: Online shop system with a shopping cart

How can session management be implemented over a stateless HTTP?

(1) Hidden fields

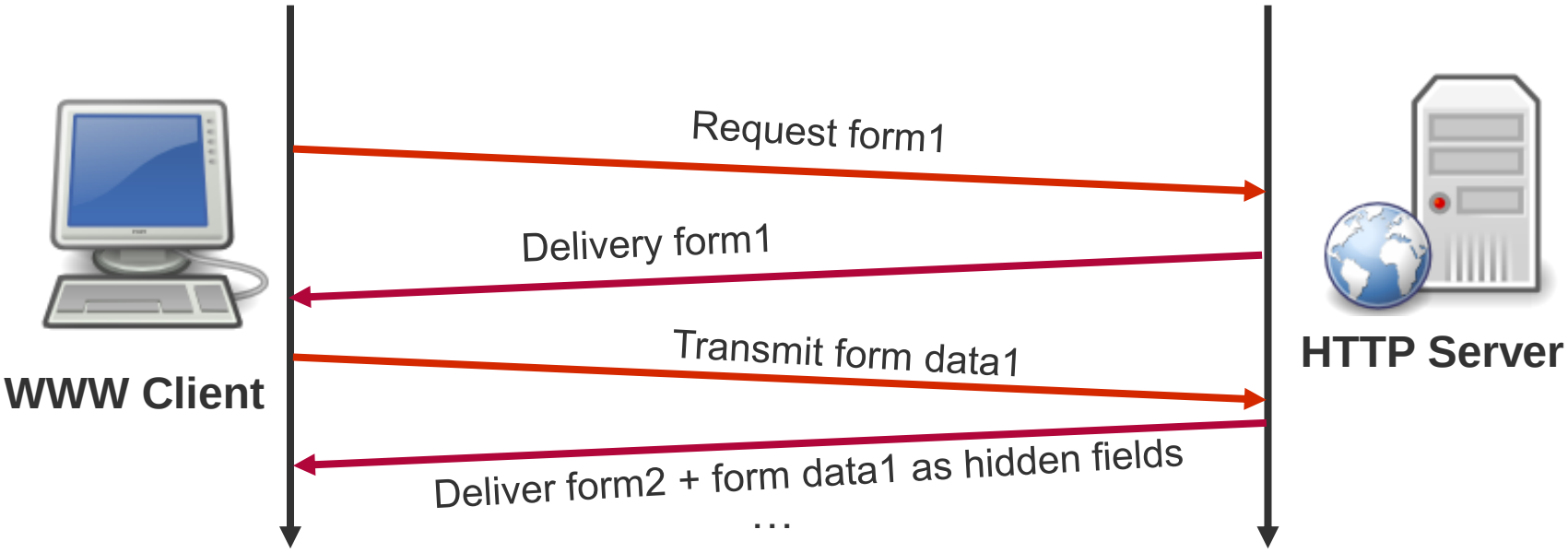
(2) URL rewriting

(3) Cookies

Session Management by Hidden Fields (1/3)

Hidden Fields

- HTML form element, comparable to text fields, selections, checkboxes
- Invisible for user
- Transmitted as name=value pair to the server



Session Management by Hidden Fields (2/3)

Example:

```
<form method="post" action="onlineshop">
   …

   <input type="hidden"
    name="shoppingcart" value="item1" />
   <input type="hidden"
    name="shoppingcart" value="item2" />

   …
</form>
```

Session Management by Hidden Fields (3/3)

- **Pros**:
  - Generally supported by browsers
  - No special provisions at client- or server-side necessary
  - Client remains completely anonymous beyond dialogue
  - …
- **Cons**:
  - Forced continuity - successive forms must be dynamically generated in succession at server-side
  - Problems when dialogue interrupted, e.g. to read terms and conditions
  - …

Session Management by URL Rewriting

- Session information is transmitted as part of the URL
- URL integration is done via form and GET parameter

http://www.youronlineshop.com/store.php?shoppingcart[ ]=item1

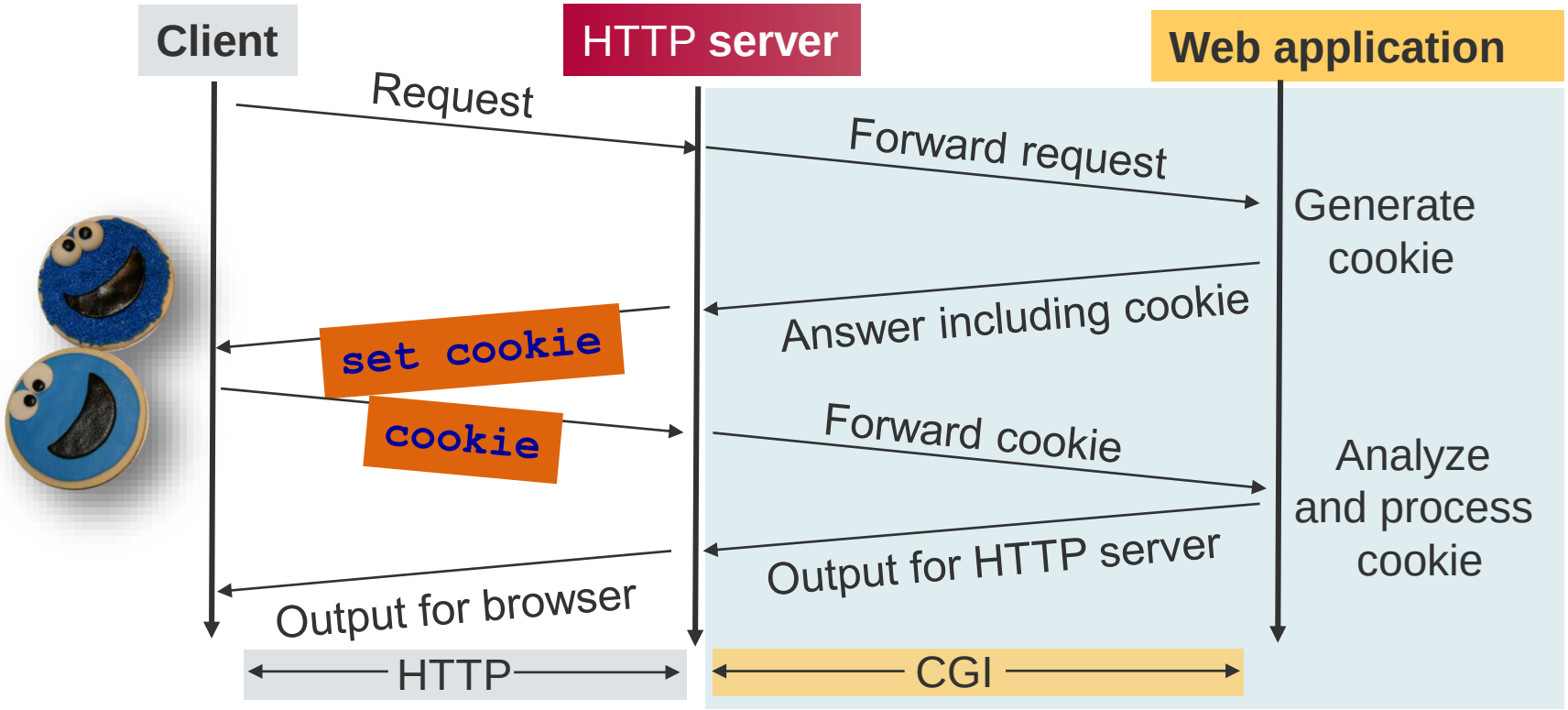
- When forms are re-delivered, server must supplement all hyperlinks dynamically with current session information
- Special characters need to be URL-encoded

Session Management by Cookies (1/4)

- Cookies are true extension of the HTTP protocol (RFC 2109)
  - HTTP header cookie
- Allow storage of session data at the client
  - Cookies are set via HTTP response header set-cookie: name=value
- Server determines lifetime of a cookie
- Cookie is transmitted in all subsequent HTTP requests to the server, regardless of server response
- User can suppress automatic transmission
- Cookie information is independent of current “surfing history”

https://flic.kr/p/6DvWd5

Session Management by Cookies (2/4)



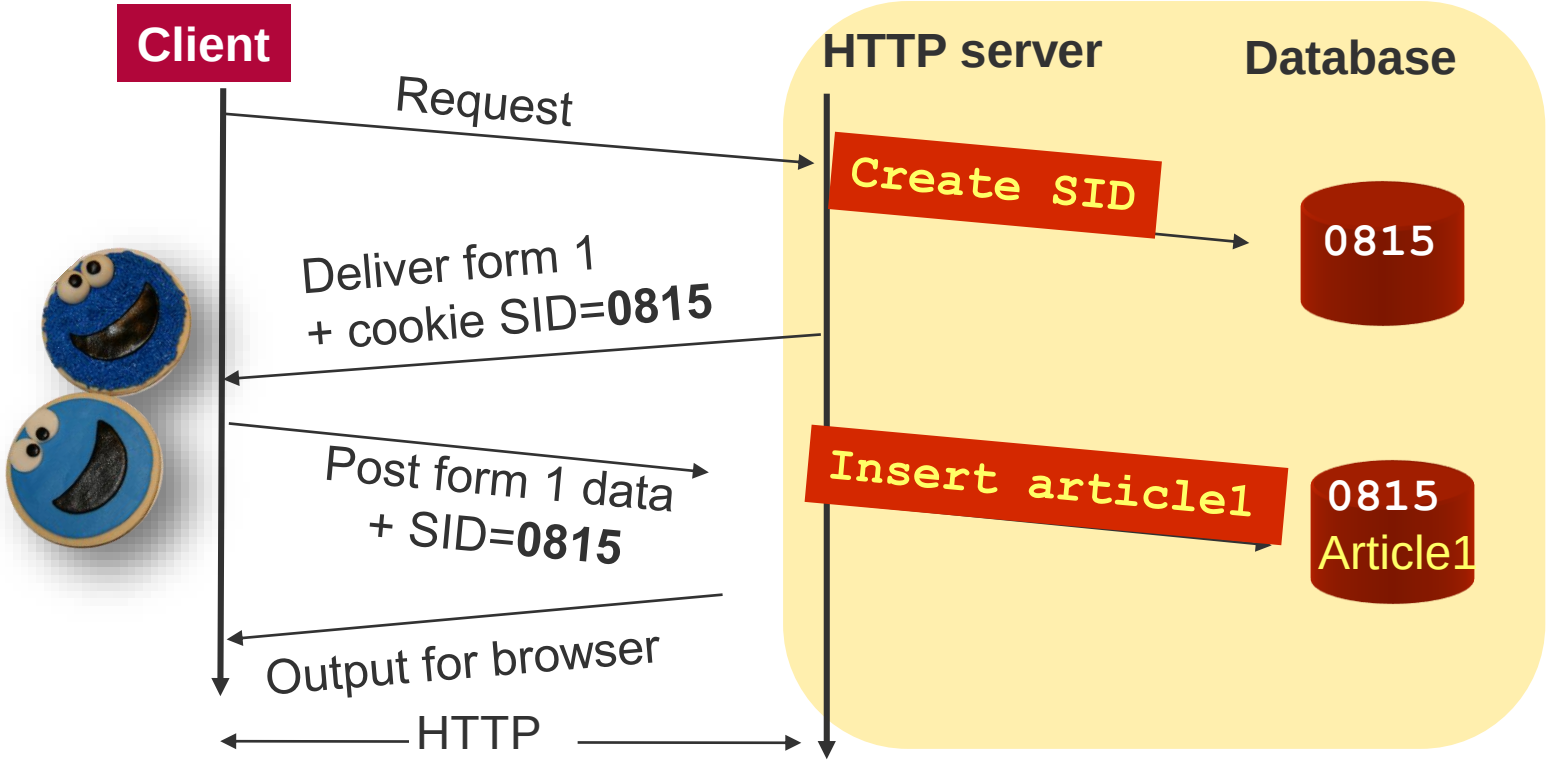
Session Management by Cookies (3/4)

Storing status information at the server

- RFC 6265 requires that a browser can store at least 50 cookies per domain (4096 bytes minimum size each - upper limit is not defined!)
- All stored data must be transmitted at every request to the server → Performance!
- **Idea**:
  - Store status information at the server side
  - Identify client / session beforehand via a session ID
  - Only session ID must be provided in the client's reply or stored in a cookie
  - Session must be terminated by client or server timeout

Session Management by Cookies (4/4)

Storing status information at the server



openHPI

Persistence and Database Layer

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

Persistence and Database Layer

How is data of a web application accessed and stored on the Web server?

Web applications are designed in different layers:

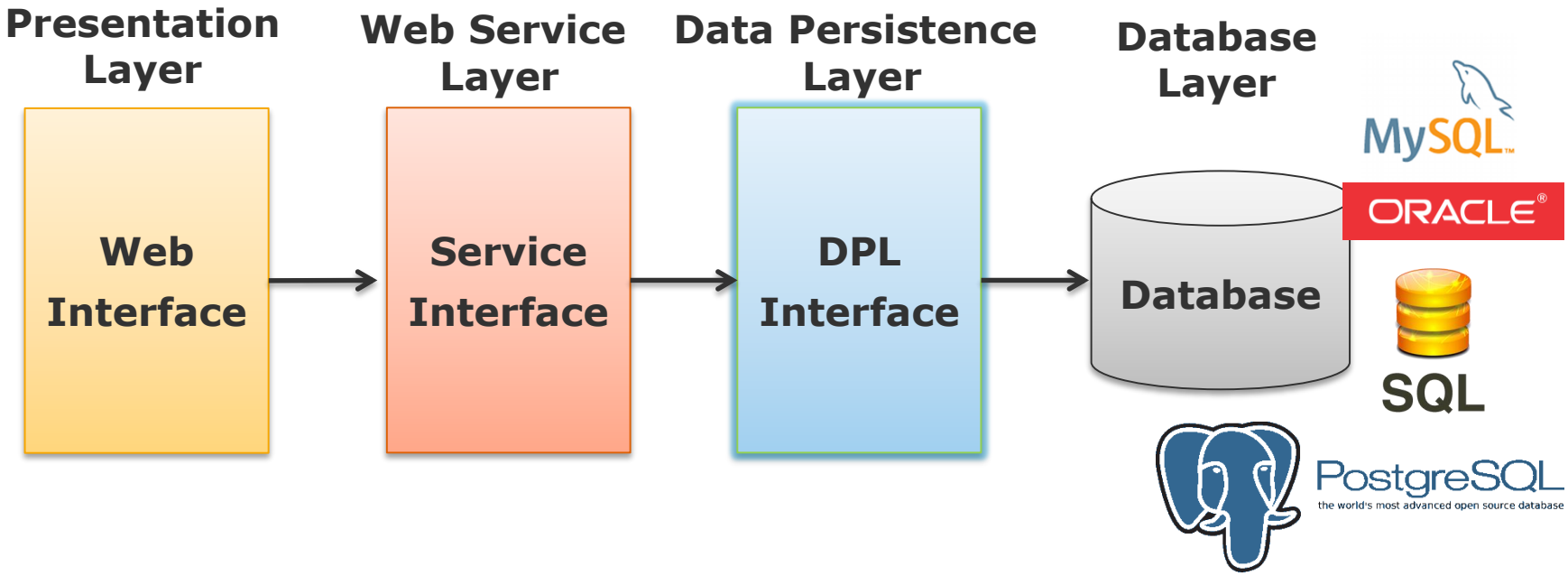
- Presentation Layer
- Web Service Layer with the application logic
- Data Persistence Layer
- Database Layer

Idea of the Data Persistence Layer

- Increasing the efficiency of the database access

Data Persistence Layer (1/3)

- **Data Persistence Layer** (or Data Access Layer) is an intermediate layer between the Application Logic Layer and the Database Layer
- It reduces coupling by isolating the rest of the application layers from the database layer



Data Persistence Layer (2/3)

- Must be able to support the complex data requirements of an application
- Handles scalability and performance issues of web data access
- A well-designed Data Persistence Layer can drastically improve performance of the entire application
- Success factors for building powerful Data Persistence Layer are
  - Built-in runtime performance and scalability
  - Developer productivity and code maintainability
  - High availability features for deployment

Data Persistence Layer (3/3)

Object-oriented Data Persistence Layer

- Uses persistent objects for data access and storage
- Improves usability and maintainability
- Treats persistent objects the same as any other type of object
- Applies the object-relational (O-R) mapping approach to take advantages from object-oriented (OO) programming
  - OO concepts such as encapsulation, abstraction and reusability make it easier to manipulate the Data Persistence Layer
  - Obtaining information in related tables becomes as simple as accessing an object attribute
- O-R mapping tools are available for almost all mainstream programming languages, e.g., Hibernate (Java), Nhibernate (.NET), Django (Python) etc.

Caching for Performance and Scalability

- Server-side caching is the most common and effective solution for performance enhancement of web applications
- It works by moving data closer to the application logic layer and, hence, reducing the number of database requests
- Factors for designing a caching solution:
  - How much data can be cached?
  - What kind of caching method should be used (static or dynamic)?
  - Caching relationships
  - Will the cached data remain consistent and accurate if it is distributed across multiple applications?

Caching Strategies: Static Caching

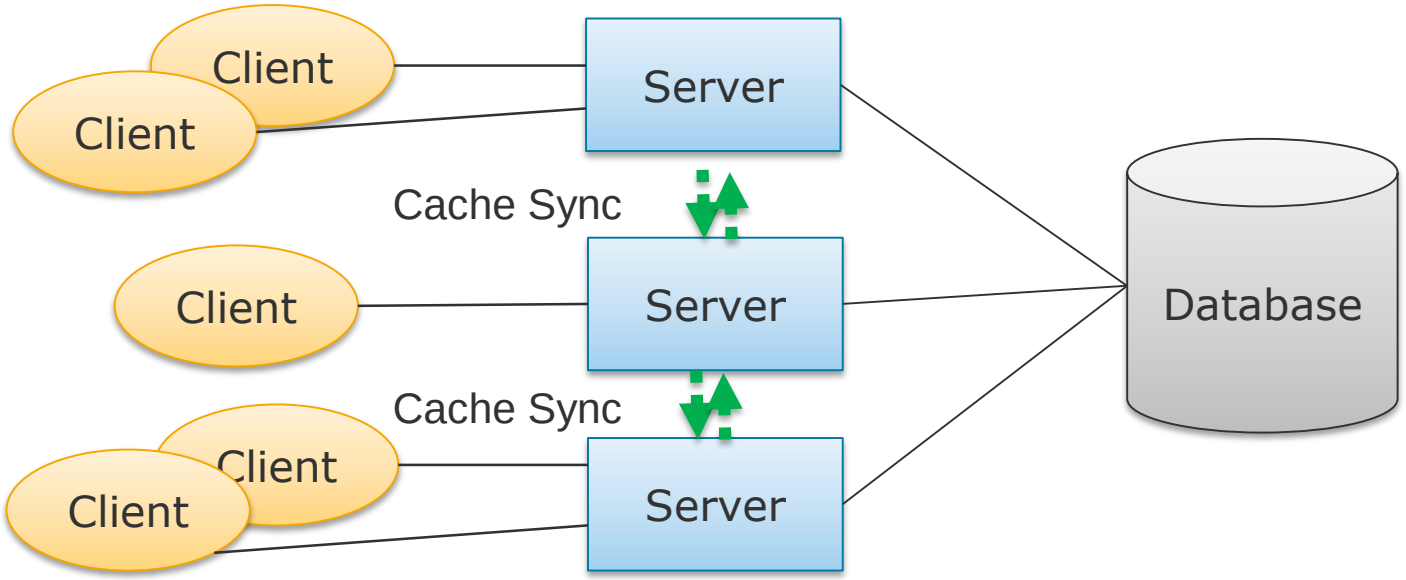
- A **static cache** (or read-only cache) is suited for data that remains unchanged during its lifecycle
- Instead of retrieving requested data from the database every time, client can simply query the server-side cache which in return reduces the overhead of database calls
- It only solves performance problems for read-only data which may be a portion of the entire data model of an application

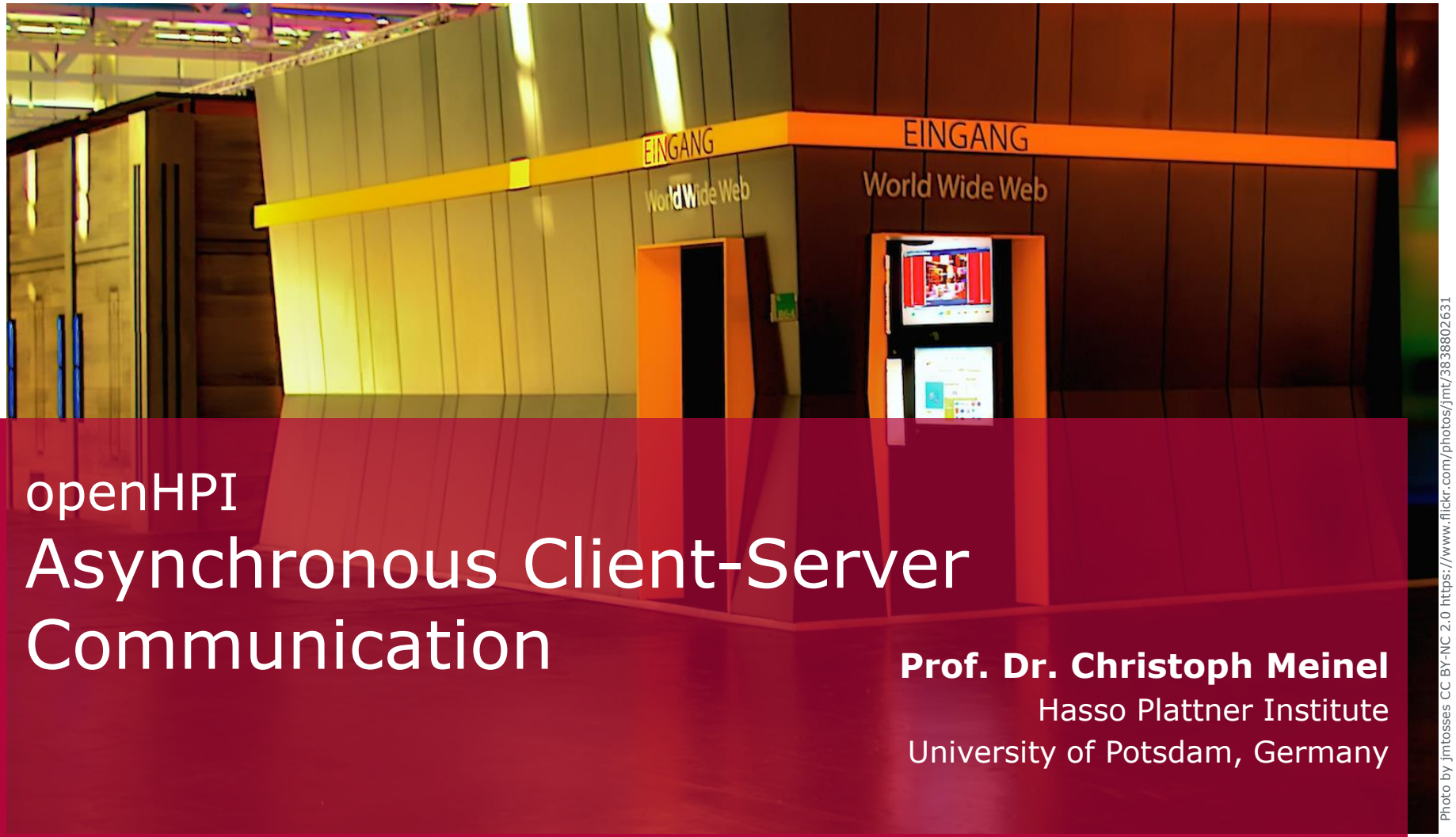
Caching Strategies: Dynamic Caching

- A **dynamic cache** allows to read and **update** data in the cache
- It provides increased performance benefits as it immediately impacts the response of updated data
- Versioning mechanisms are applied for solving the multiple access problem
  - Assigning version numbers to different revisions of the data
  - Let the cache know if a thread is attempting to update stale data
  - Improves performance since data in the cache will not be locked

Caching Strategies: Distributed Caching

- **Distributed caches** are simply a cluster of dynamic caches
- They address scalability and high availability of enterprise applications
- All application caches need to be synchronized
- When **one** single server goes down, the other servers would easily be able to satisfy the additional failover requests





openHPI

Asynchronous Client-Server Communication

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

AJAX (1/5)

Asynchronous JavaScript and XML

- Term coined by Jesse James Garett:
  "Ajax: A New Approach to Web Applications" in February 2005
- Often considered as a new technology enabling Web 2.0 applications, however, it was already introduced by Microsoft in 2001 under the name **XMLHttpRequest** in Internet Explorer 5

Basic idea:

- Data is exchanged between client and server using JavaScript. Thereby, only certain parts of the page are updated, i.e. the page does not have to be completely reloaded

AJAX (2/5)

General Procedure:

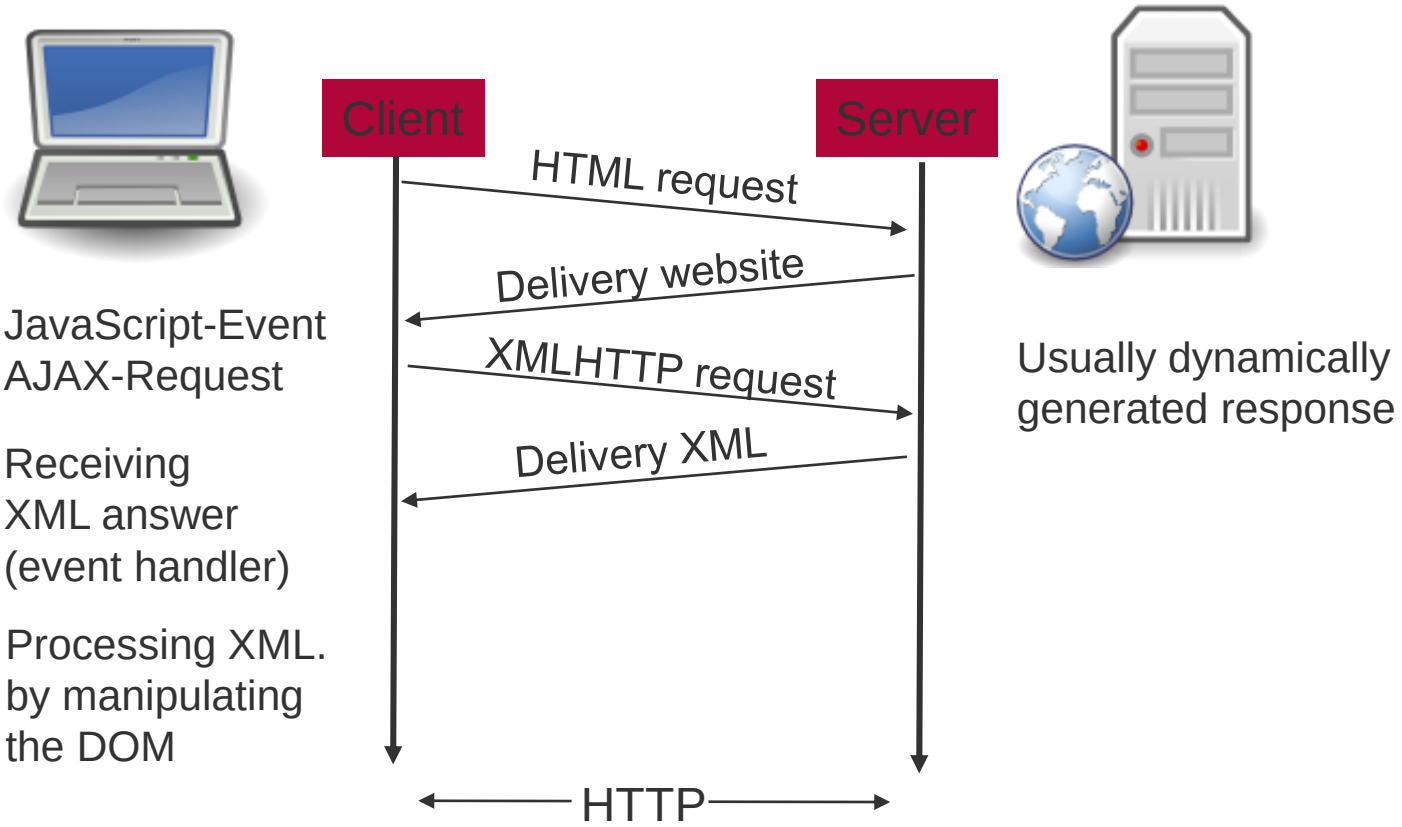
- By means of JavaScript a XMLHttpRequest is executed
- Requests can take place asynchronously in the background

  → Data is downloaded via HTTP request and does not interfere with user interaction within the page

  → When issuing the XMLHttpRequests, the function is linked with an event that is triggered at the end of the request
- Downloaded data is usually XML or JSON
- XML data is parsed by means of JavaScript and integrated using DOM manipulation in HTML source text

AJAX (3/5)

AJAX Request



AJAX (4/5)

Advantages

- Server load and bandwidth requirement is reduced as pages do not have to be completely reloaded; only certain page content is changed
- Faster feedback because changes can be displayed faster

  → improved usability
- Creation of GUIs is possible that blur the borders between web applications and desktop software
- Fosters creation of web applications
- Numerous frameworks / libraries are available for fast and effective Ajax programming, e.g.
  - jQuery or prototype

AJAX (5/5)

Disadvantages and limitations:

- JavaScript must be activated
- Potential increase in programming effort; often additional fallback solutions are required for important parts of a page
- Blurs separation of structure and layout
- Decreased accessibility as JavaScript is required
- Extensive browser testing is necessary for compatibility reasons
- Web crawlers do not execute JavaScript code

  → poor indexibility
- AJAX requests do not appear in browser history (function of the back button is suspended)

JSON (1/2)

JSON – JavaScript Object Notation

- Lightweight data-interchange format, alternative to XML
- Easy to read for humans, easy to parse for machines
- Supports universal data structures, available in most programming languages
  - Arrays: Ordered list of values
    - aka vector, list
  - Objects: Collection of name / value pairs
    - aka struct, dictionary, associative array

Compared to XML

- Advantage: More compact
- Disadvantage: No means to validate the data format

JSON (2/2)

JSON

```
{
 "course": {
  "title": "Web Technologies",
  "sections": {
   "section": {
    "title": "Week1"
    "description": ”Intro",
     "items": {
      "item": {
       "type" : "quiz",
       ...
      }
     ...
     }
  }
}
```

XML

```
<?xml version='1.0'?>
<course>
 <title>Web Technologies</title>
 <sections>
   <section>
     <title>Week 1</title>
     <description>Intro</description>
        <items>
         <item type="quiz">
           <title>Quiz1</title>
         </item>
        </items>
      </section>
   </sections>
</course>
```

Web 2.0 (1/3)

- **Web 2.0** is not a specific technology, but rather a bold generic term for new web standards and technologies making the creation of web content easier
- Term was created by Tim O'Reilly with the intent of describing a changed perception and use of the WWW
- Many of the Web 2.0 technologies had already been developed at the end of the 1990s, e.g.
  - AJAX
  - Web Service API
  - RSS
  - …

Web 2.0 (2/3)

Since about 2005 there has been a change in the perception and use of the Internet:

- Increased use of the Web as a platform - boundaries blur between web applications and desktop programs, e.g.
  - Google Docs instead of a local word processor

  → Only a browser is required
- Participatory principle: Users become editors, e.g. blogs and wikis

  → Community feeling
- Personalization of GUIs and web pages
- Complex roles and rights systems:
  - Structuring of users into groups who, based on rights and roles, can view or edit contributions

Web 2.0 (3/3)

Examples of Web 2.0 services:

- Flickr.com – online photo albums
- Writely.com, Google Docs – word processing via the Web
- Feedblendr.com – news feed service
- Blogspot.com – blog service
- Blog.sina.com.cn – blog service
- Google Maps – map service
- Delicious.com – online bookmark management
- Facebook, Google+, … social web platforms
- Renren.com – Chinese social web platform
- Mendeley – collaborative research paper management
- ...

openHPI

Lightweight Webservices: XML-RPC & REST

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

Lightweight Web Services

Lightweight alternatives to WS* Standards for large distributed systems on the Web at enterprise scale: **XML-RPC** and **REST**

XML-RPC

- Historical predecessor of SOAP –protocol specification for web services
- Very simple XML protocol for remote method invocation
- Uses only limited HTTP vocabulary

REST – REpresentational State Transfer

- No technology – it is an architecture
- ROA – Resource Oriented Architecture
- Uses the complete HTTP vocabulary
- It has prevailed over XML-RPC during the last years

XML-RPC

XML-RPC example:

```
POST /shoppingcart HTTP/1.1
Host: myServer.org
Content-Type: application/xml


<?xml version="1.0"?>
<methodCall>
  <methodName> deleteItem </ methodName >
  <params>
    <param><value><int>244</int></value></param>
  </params>
</methodCall>
```

Always uses the Post method

The actual action to be performed is specified in the body of the request

Also the subject on which the action has to be performed

REST (1/4)

REST - REpresentational State Transfer

- Web is resource-oriented and REST shows a way to process these resources exclusively with the methods of HTTP
- HTTP path determines which resources are processed
- HTTP methods determine how the resources are processed
  - POST – resource is created (Create)
  - GET – resource is read (Read)
  - PUT – resource is updated (Update)
  - DELETE – resource is deleted (Delete)
- HTTP body transports only the data to be transferred

REST (2/4)

Elements of the HTTP Protocol (RFC 2616)

- Request
  - Methods: GET, POST, PUT, DELETE
  - Path: Part of the URL https://openhpi.de**/courses/1/settings**
  - Request header
  - Body
- Response
  - Response code: 200 OK, 403 Forbidden, 404 Not Found ...
  - Response header
  - Body

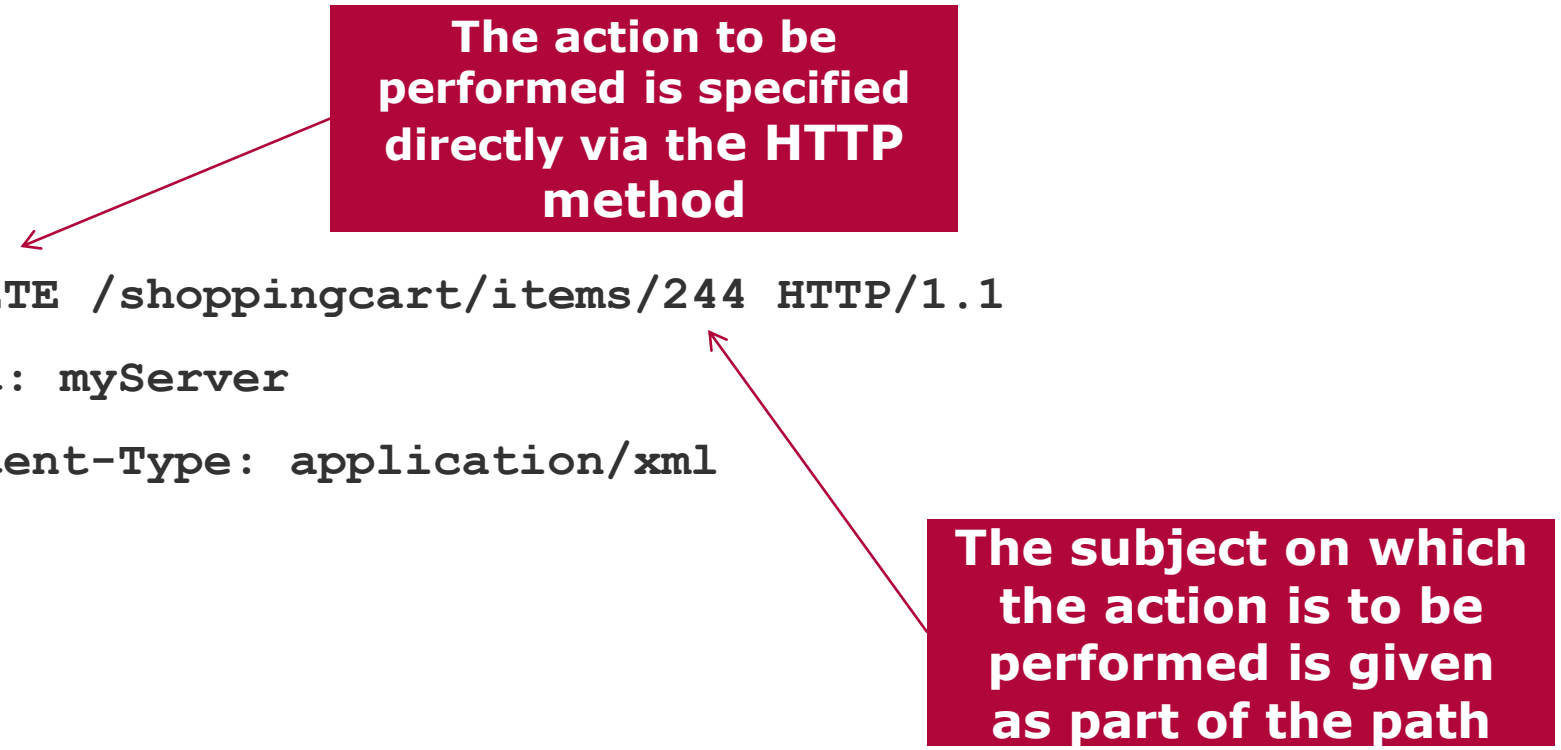
REST (3/4)

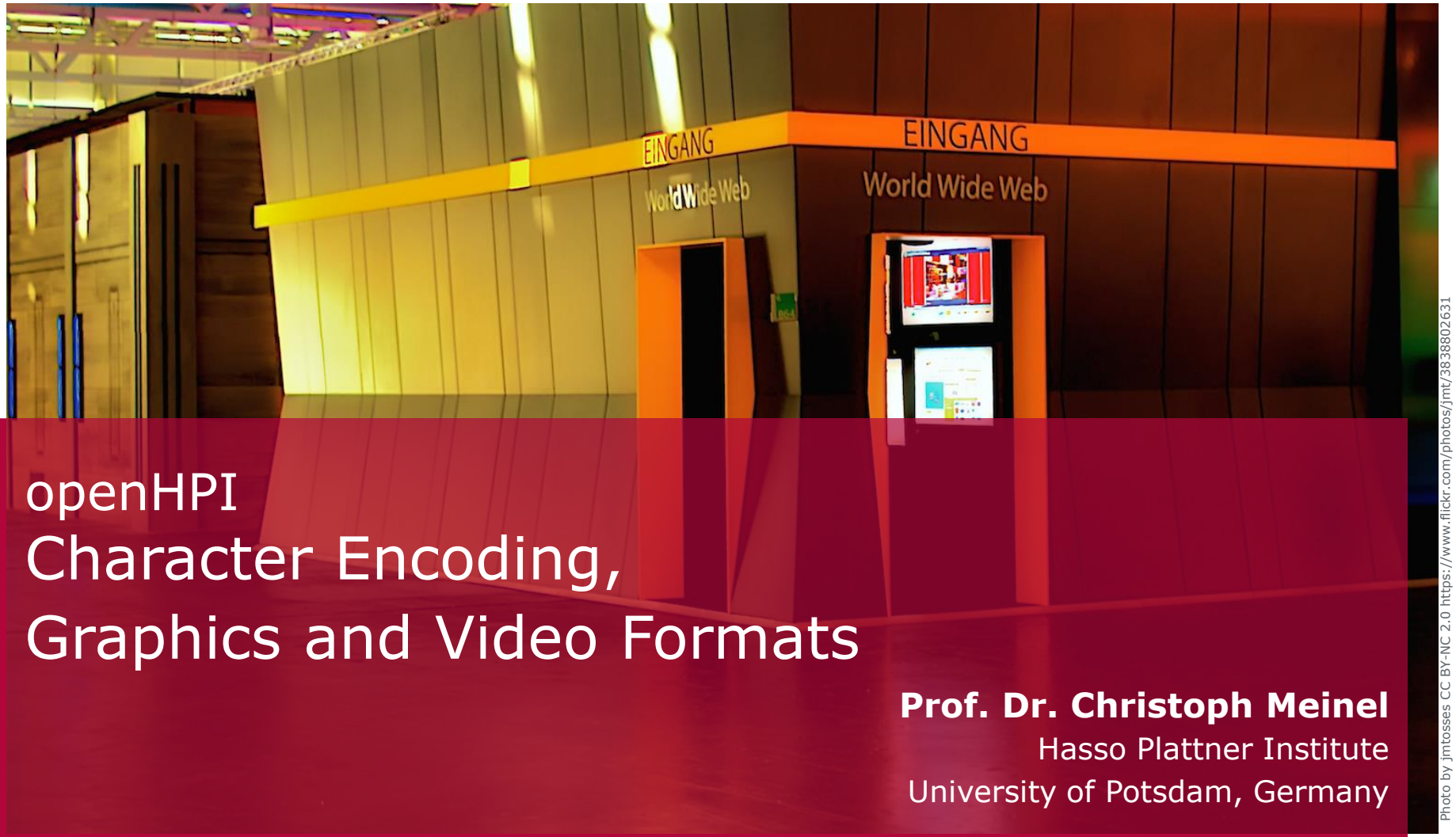
Four design principles:

- Only the HTTP methods GET, POST, PUT, and DELETE are used
- Statelessness – all necessary data is transferred upon request
- URLs are oriented on directory structures and should be intuitive to the point that when in doubt they can be guessed
- Data is transferred as XML or JSON
- The service notes the MIME type of the request and sends back the response in the appropriate MIME format

REST (4/4)

REST example:





openHPI

Character Encoding, Graphics and Video Formats

Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

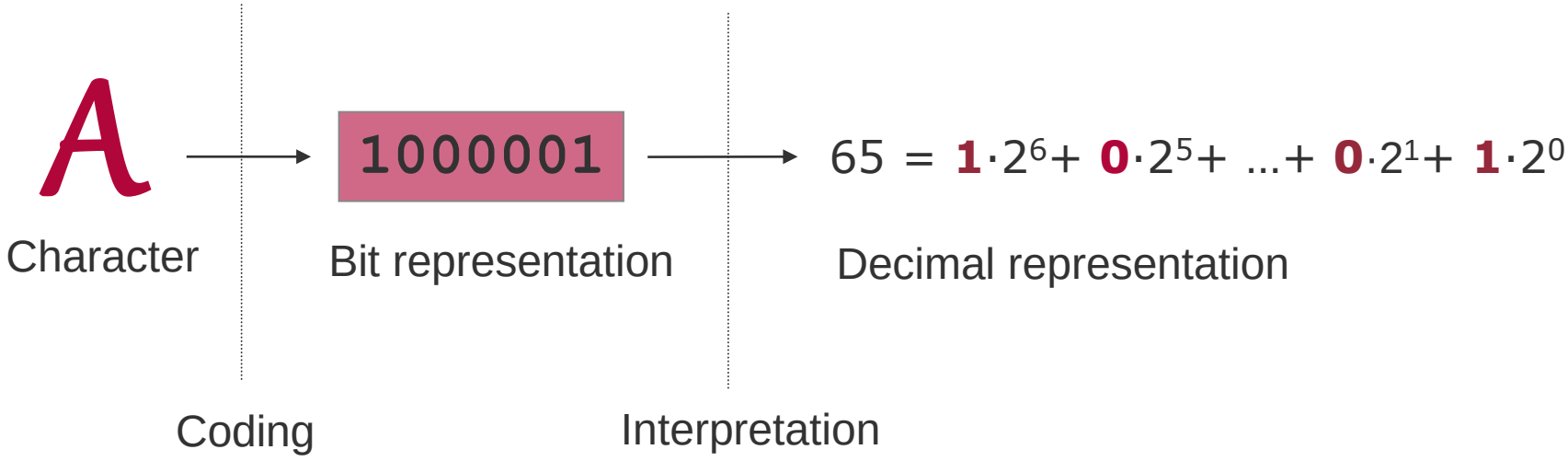
Bandwidth and Efficiency of Data Transfer

In the Web lots of data has to be transferred (typically) from server to client

- In order to be able to communicate text information, its characters need to be encoded binarily first
- Encoding needs to be standardized otherwise web browsers are not able to “understand” transmitted characters correctly, i.e. to decode and display the received text in the right way
- Today, more and more information does not come in form of text but as graphics, audio or video
- Since users expect fast and efficient communication, it is crucial to provide this data in formats that allow an efficient transfer

Character Encoding (1/3)

Characters (letters and others) have to be encoded to a format computers can handle



Historical first approach:

ASCII – American Standard Code for Information Interchange

→ Not international …

Character Encoding (2/3)

ASCII – 7 Bit encoding with 128 Characters:

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

Character Encoding (3/3)

- With 7 Bit ASCII only $2^7 = 128$ different characters can be encoded
- To encode more characters, more bits are necessary:
  - 8-Bit ASCII encoding with ISO 8859
    - 0-127 identical to ASCII
    - 128-159 rare control characters
    - 160-255 (multi)national extensions (e.g. ¿ i ñ ç ä ö ü ß ญ …)

ISO 8859-1 – Western Europe
ISO 8859-2 – Central Europe
ISO 8859-8 – Hebrew
…

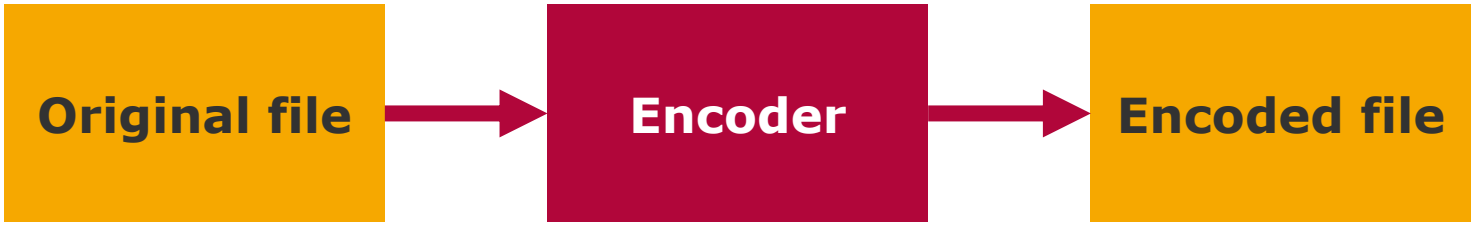
- **Still**: What about Chinese and others … having thousands of characters?
  - **Unicode – UTF** (Universal Transformation Formats)
    - UTF-8: 8-Bit Unicode coding, compatible with ASCII
    - UTF-16: 16-Bit Unicode (characters consume 16 or 32 bits)
    - UTF-32: 32-Bit Unicode (characters always consume 32 bits)

Graphics and Video Formats

Today, more and more information comes in form of multimedia data

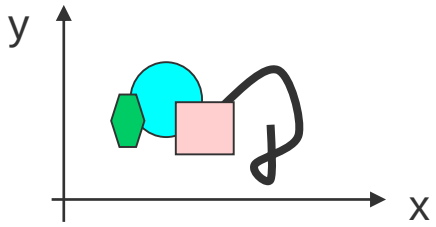
- While encoded text does not require lots of bandwidth, graphics, audio and video do
- In order to increase the browsing speed and reduce traffic for routers, it is important to reduce the size of the transferred files

→ **Encoding** – **compress** and **encode** data to a standardized format resulting in a smaller file size with an **information loss** as little as possible



Graphics Encoding: Vector Graphics (1/2)

Vector graphics is a mathematical format using geometrical primitives (points, lines, curves, shapes and polygons)



- Described by vectors with a starting point (x,y) and a direction

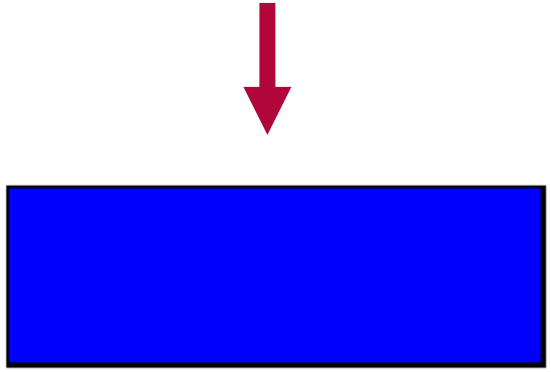
Most popular format: **SVG** – **Scalable Vector Graphics**

- Advantage:
  - **Scalable** at any zoom level, low space consumption, XML-based format (transformations possible)
- Disadvantage:
  - Does not work for **photos** or other complex pictures

Graphics Encoding: Vector Graphics (2/2)

SVG – A small example:

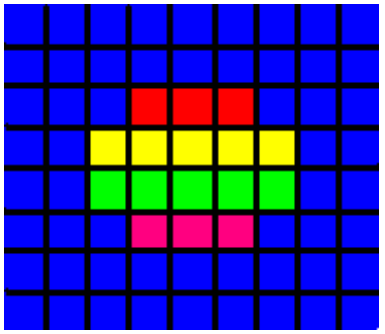
```
<svg width="400" height="110">
  <rect width="300" height="100"
    style="fill:rgb(0,0,255);
    stroke-width:3;stroke:rgb(0,0,0)" />
</svg>
```



Graphics Encoding: Bitmap Graphics

Bitmap Graphics

- Historically:
  - Save all the information for every single pixel
  - For example BMP format
  - Not suitable for the Web
- Better:
  - Compressed bitmap formats which omit (unnecessary) data and save space and bandwidth, e.g. **JPEG**, **PNG**, **GIF**, **TIFF**, …



Example:

BMP, 88 KB

JPEG, 27 KB

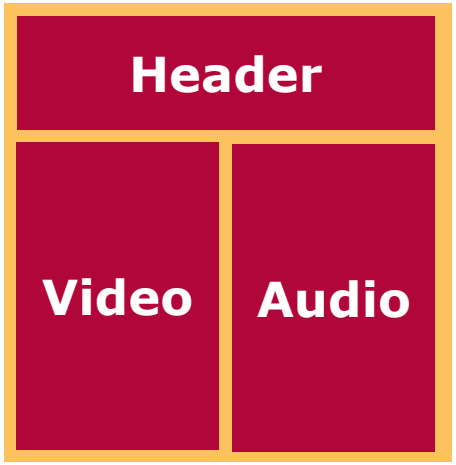
JPEG, 2 KB

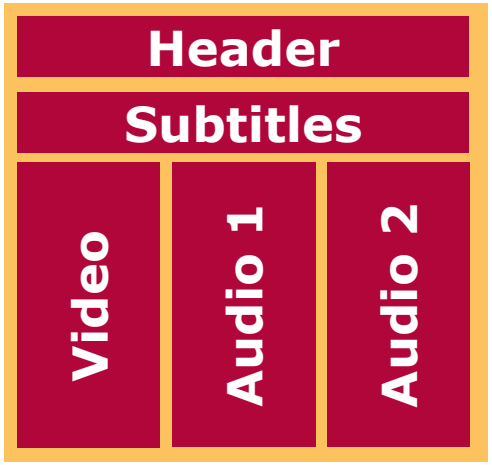
Video Encoding

Usually, a video consists of a **video stream** and an **audio stream**, both encoded with a **codec** combined in one file **container**

- **Video codecs**: H.264, Ogg Theora, WebM, …
- **Audio codecs**: mp3, wma, AAC, Ogg Vorbis, …
- **Containers**: mp4, WebM, OGV, …

Examples:





Problem: Not all browsers support all formats, HTML 5 …