openHPI Web Technologies 2015
Week 4

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

openHPI

Introduction Web Programming

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

Introduction to the Next Two Weeks

We want to introduce "**Web programming**"

Under this term different things are subsumed :

- Client- and server-side programming
- Middleware architectures and
- Web services

In the next two weeks we introduce in more detail **client-** and **server-side programming**

Introduction to the Next Two Weeks

In week 4 we introduce **client-side programming:**

- Dynamic HTML, DOM, JavaScript
- JavaScript libraries and frameworks
- CoffeeScript and Dart

In week 5 we introduce **server-side programming:**

- Server-side web-frameworks and MVC
- Session management, persistence and database layer
- Asynchronous client / server communication (AJAX)
- Web-based APIs and web-services (REST)

Introduction to the Next Two Weeks: Client-Side Programming

- Web server passes the executable / interpretable program code with the requested HTML document to the WWW browser (client), and it is then executed at the client side
- WWW client needs an interpreter (as a browser plugin if appropriate) which together with the client can interpret and execute the transferred program

  → Java Applets

  → JavaScript Programs (or scripts)

  → Microsoft Silverlight and Adobe Flash

Introduction to the Next Two Weeks: Server-Side Programming

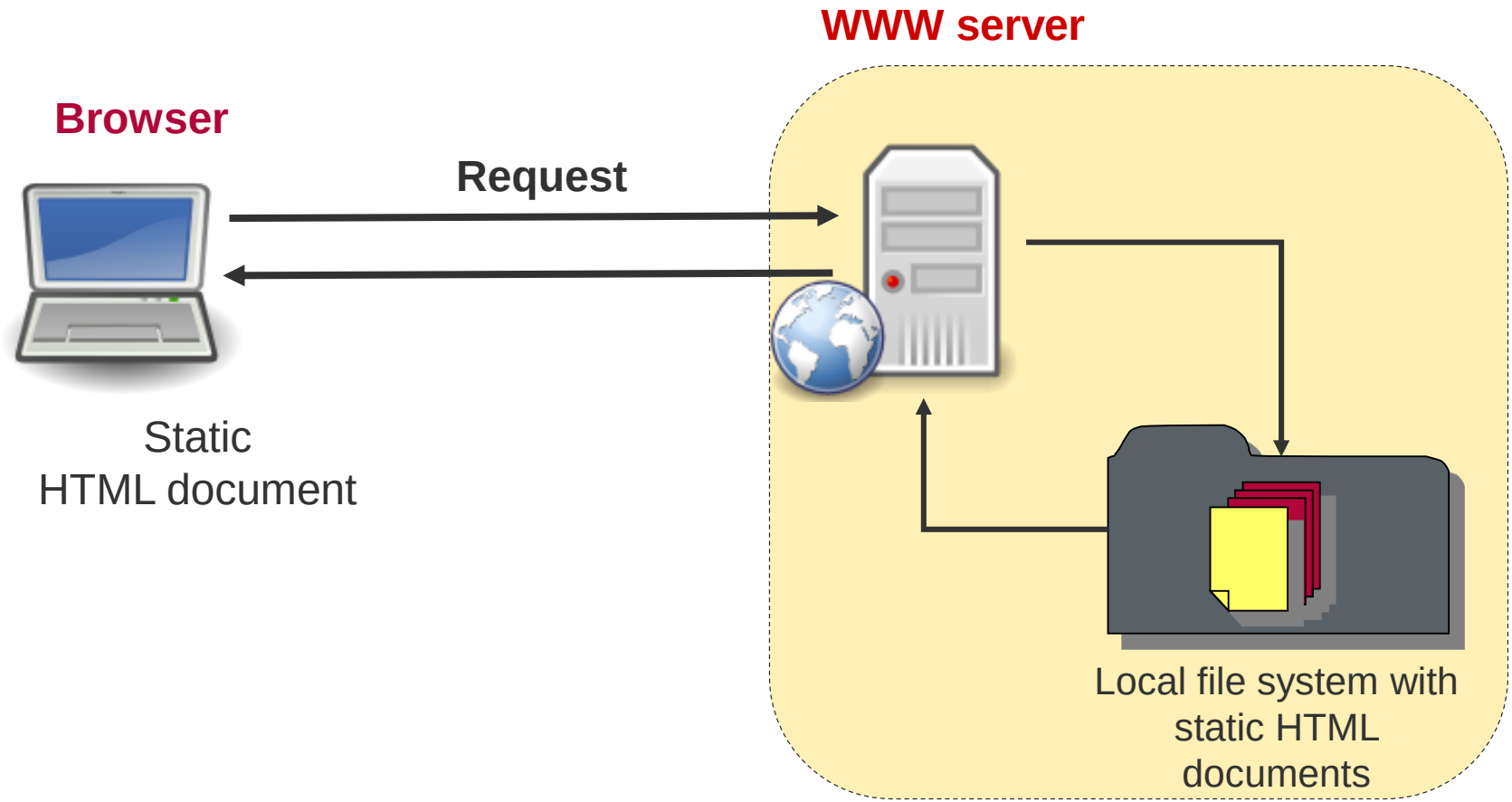
- Web application is activated – from WWW browser (client) over the WWW server – at the server machine and executed there
- Results are generated dynamically on the server side and returned to the client as HTML document
- **CGI interface** is the standardized interface on server side between web server and web application
- Web applications on 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, …

Static and Dynamic Web Documents

Browsers can request HTML documents that

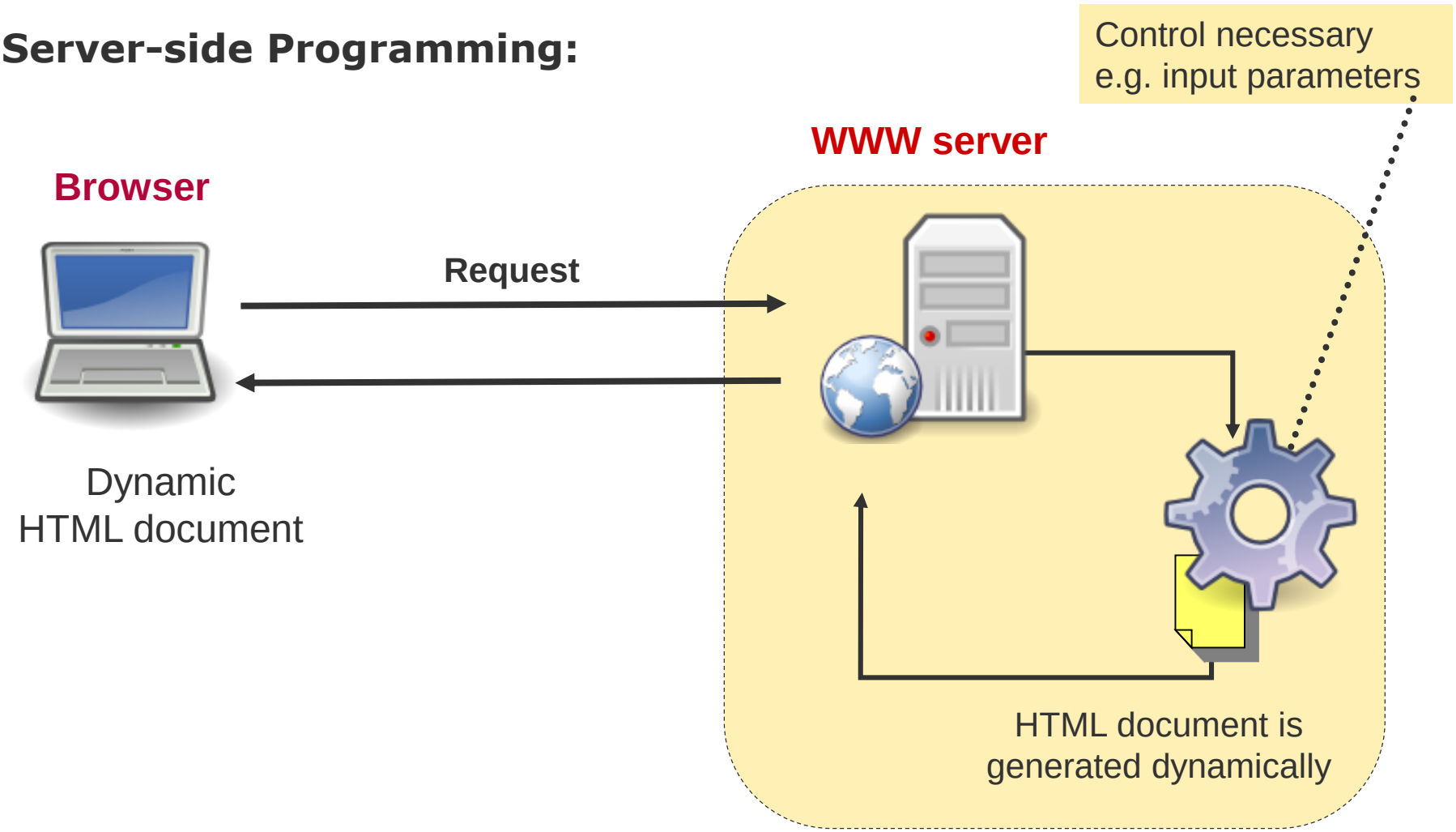
- exist **statically** and are stored for retrieval at the server's file system, or
- are **generated dynamically** by a web application according to parameters passed from the browser to the WWW server
  - practical if, e.g., queries are to be answered from a database as in the case of
    - product catalogues, e.g. amazon.com
    - search engines, e.g. google.com
    - newspapers, e.g. nytimes.com
    - …

Static Web Documents



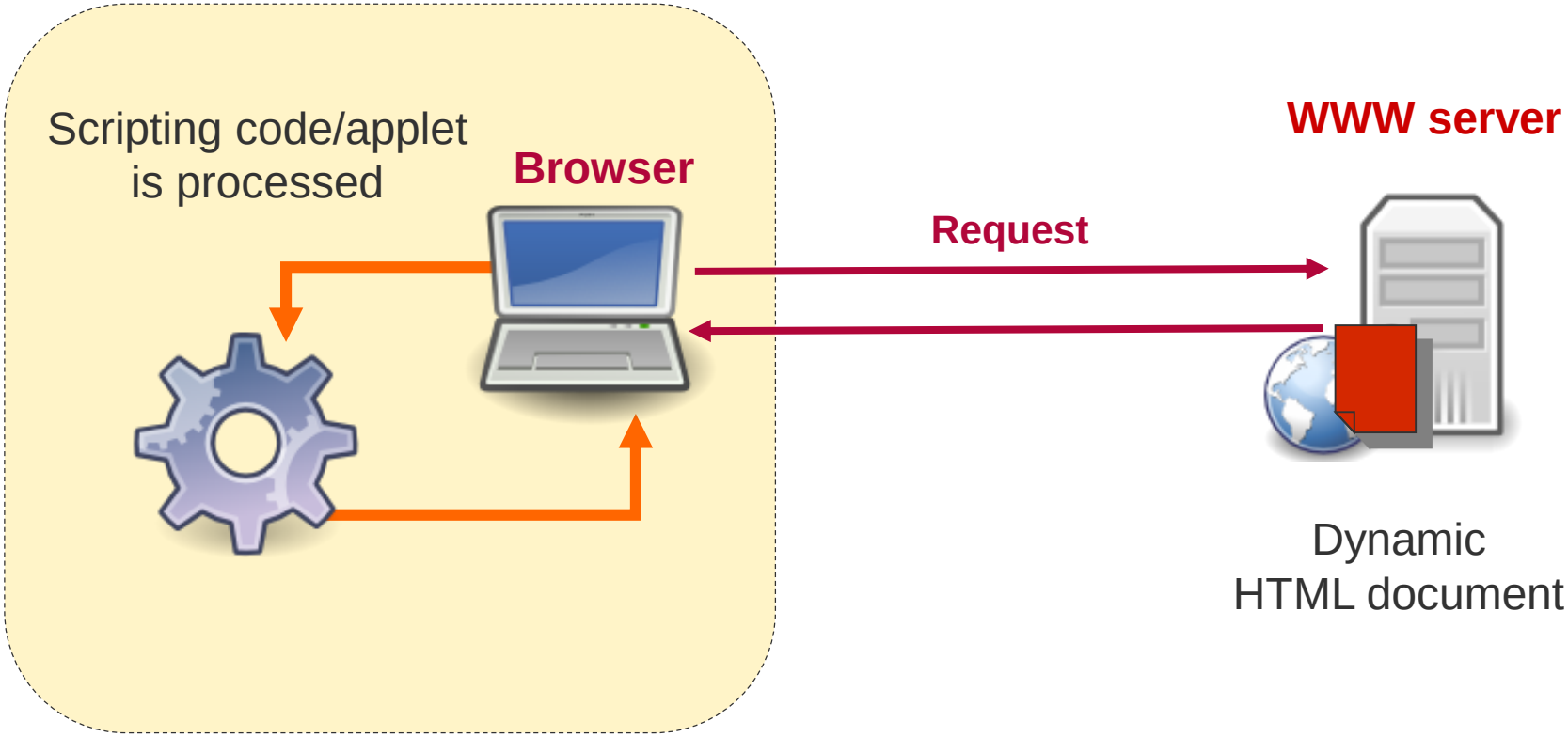
Dynamic Web Documents Generated on Server-Side

Server-side Programming:



Dynamic Web Documents Generated on Client-Side

Client-side Programming:



openHPI

Client-Side Web Programming

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

Client-Side Web Programming

- Executable / interpretable program code is passed from the web server within the requested HTML document to the web browser (client) and executed on the client machine
- Web client needs interpreter (as plugin if appropriate) that can interpret and execute the transferred program code
  - → Java Applets
  - → JavaScript programs (or scripts)
  - → Microsoft Silverlight and Adobe Flash

Java Applets (1/2)

- Applets are interactive GUI elements that are executed in a secure browser environment
- More precisely:
  - Applets are Java classes that are embedded in HTML documents
  - Inheriting from java.applet.Applet

```
public class HelloJavaApplet extends java.applet.Applet {
   public void paint(java.awt.Graphics gc) {
      gc.drawString("Hello Java (Applet)", 90, 95);
   }
}
```

Java Applets (2/2)

HTML document

```
<HTML>
…
<APPLET CODE="HelloWorld.class"
 CODEBASE="http://www....."
 WIDTH=300 HEIGHT=500>
<PARAM NAME="width" VALUE="200">
…
</APPLET>
…
```

Today applets are considered obsolete …

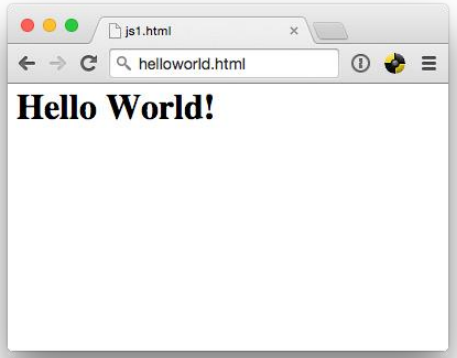
- often clearly recognizable as applets
- frequently "feel" too slow/sluggish
- are being replaced by e.g. DHTML or Flash

JavaScript / ECMAScript (1/2)

HTML document

```
<html>
...
<script language="JavaScript">
document.write("<h1>Hello
World!</h1>");
</script>
...
</html>
```

Browser



- Originally introduced by Netscape
- Later standardized as ECMAScript
- Classical application: client-side scripting
- Allows access to document contents (DOM)
- Allows response to user action (EventHandler)
- Note: No connection to Java except for the name

JavaScript / ECMAScript (2/2)

JavaScript is a dynamically typed language and offers:

- Primary data types: String, Number, Boolean, Function, Object
- Control structures: Conditions and loops
- Functions
- Objects and inheritance (via "prototypes")
- Pre-defined objects: e.g. Array, Date, Math, window, document as well as objects from the ➔ **DOM** (Document Object Model)

The use of "pure" JavaScript is rare today

Instead use of:

- Powerful libraries and frameworks:
  - jQuery, React.js, AngularJS, Ember.js, Raphael.js, etc.
- Languages that compile to JavaScript, e.g. CoffeeScript, Dart

Safety Mechanisms

Running code from others in your browser involves inherent safety risks !

- Therefore: JavaScript interpreter runs in a **sandbox**
  ➔ JavaScript only has access to browser objects, not to local hardware or file system
- Web applications / websites are separate in the bowser so they can only access resources that come from the same domain
  ➔ **Same Origin Policy (SOP)**
- Similar restrictions apply to Flash, Java, etc.
  (but here, access to local resources can be given)
- **Caution**:
  - SOP can be bypassed in Internet Explorer with ActiveX components
  - Recent developments, such as access to webcams, local storage, etc., might break the sandbox principle

Adobe Flash

- Originally an animation tool, scripting support added later
- Proprietary authoring environment and browser plugin required
- Scripting language ActionScript
  - ActionScript 1: close to JavaScript syntax and concepts, e.g. objects using prototyping
  - ActionScript 2 + 3: syntax and concepts closer to Java
- Authoring tool exports .swf files containing animations and scripts
- `<embed>` and `<object>` tag to embed .swf in HTML pages
- Better: JavaScript SWFObject (https://code.google.com/p/swfobject/)
  - Open source JavaScript library
  - Compatible with most browsers
  - Flash player version detection

Microsoft Silverlight

- Equivalent to Adobe Flash
- Proprietary authoring environment and browser plugin required
- First version released in 2007, five versions until 2013
- Silverlight and Flash have been very popular on the web
  → Particularly for video streaming, e.g. Netflix, Amazon Prime

- **Drastically reduced significance of Silverlight and Flash**:
  - Rise of JavaScript frameworks and libraries
  - HTML 5 video
  - 2010 no support for Flash and Flash Video on Apple's iPhone
  - Chrome said they will stop support for Silverlight in 2015

openHPI

DOM & DHTML

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

DOM – Document Object Model

How are hypermedia documents and their content accessed by means of a programming language?

Document Object Model – DOM

- Platform and language - independent interface for accessing any document / parts
- Standardized by W3C
  - **DOM Core Model** – compact low level description for any structured documents
  - **DOM HTML Model** – objects and methods for easy manipulation of HTML documents
  - **DOM XML Model** – objects and methods for easy manipulation of XML documents

DOM – Design

DOM Design

- DOM provides standardized set of objects and methods
- **DOM Level 1**
  - Core / HTML / XML models
  - Manipulation of documents and document navigation
- **DOM Level 2**
  - Stylesheet object model / event model and support of XML namespaces
- **DOM Level 3**
  - Load / store operation / validation of XML documents

DOM Object Model (1/4)

- DOM document modeled according to its structure as a tree
- DOM defines objects with associated attributes and methods to manipulate objects
- DOM defines
  - interfaces and objects for representation / manipulation of documents
  - semantics of the defined objects / interfaces for describing attributes and behavior
  - relationships and compatibility of individual objects / interfaces

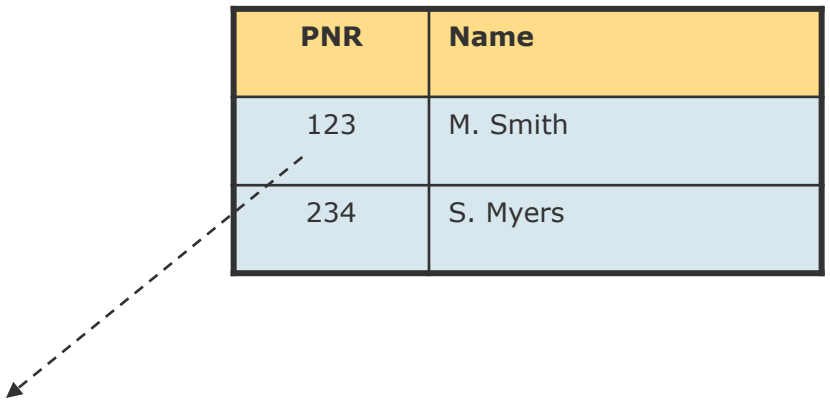
DOM Object Model (2/4)

- Each document includes:
  - root node
  - element nodes
  - attribute nodes
  - text nodes
  - (commentary nodes)
- Binding to a certain programming language through **language specific bindings**
  - e.g. in Java (Dom4J, JDOM, ...) or JavaScript

DOM Object Model (3/4)

HTML code and its representation

```
<table>
…
  <tbody>
    <tr>
      <td>123</td>
      <td>M. Smith</td>
    </tr>
    <tr>
      <td>234</td>
      <td>S. Myers</td>
    </tr>
  </tbody>
</table>
```

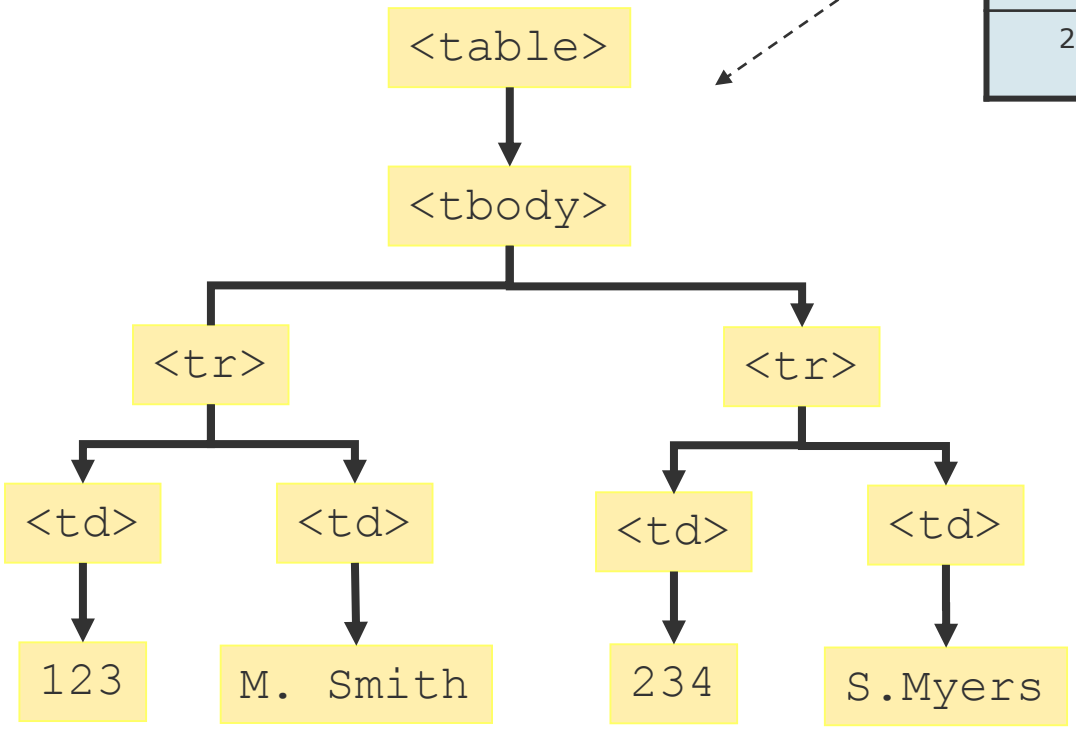


PNR	Name
123	M. Smith
234	S. Myers

DOM Object Model (4/4)

Shown in a tree structure:

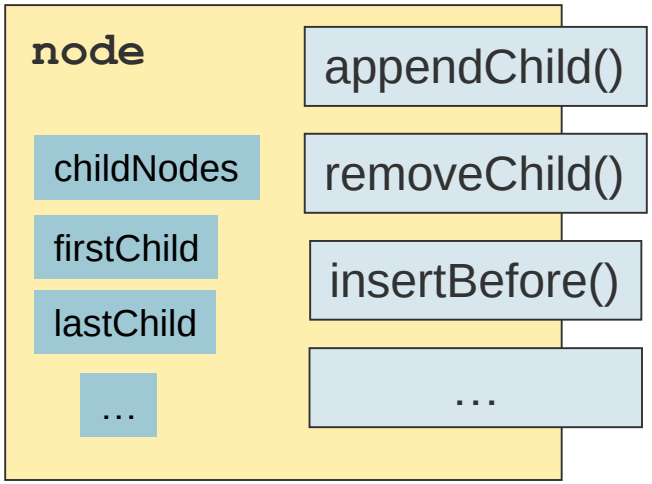
PNR	Name
123	M. Smith
234	S. Myers



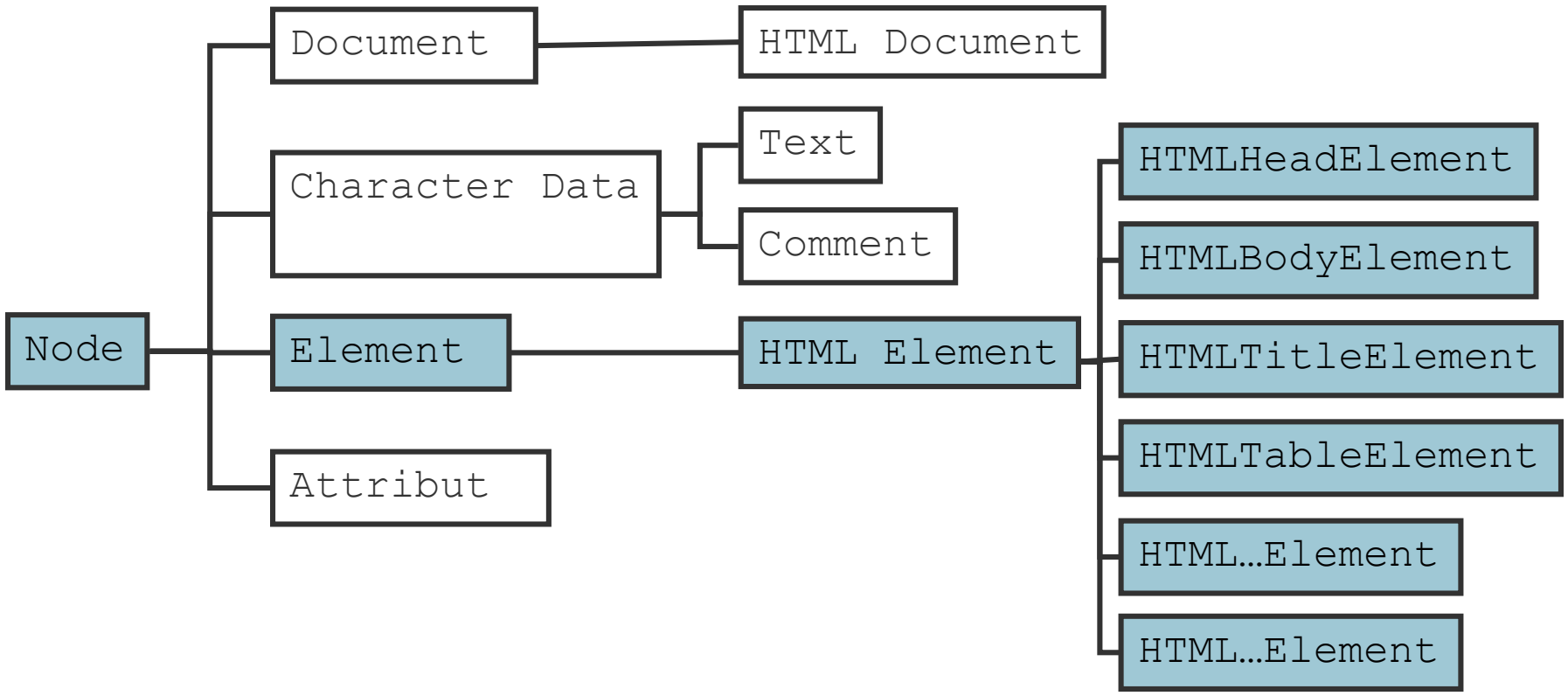
DOM Object Model – Reference

Central class: node

- Attribute:
  - node.childNodes
  - node.firstChild
  - node.lastChild
  - node.parentNode
  - …
- Methods:
  - node.appendChild()
  - node.removeChild()
  - node.insertBefore()
  - node.cloneNode()
  - …



DOM Object Model – Class Hierarchy



DHTML – Dynamic HTML (1/2)

DHTML = JavaScript + DOM (also: DOM Scripting)

- JavaScript is used to manipulate HTML pages in terms of "maturity" – meaning during display in the browser

Example: Changing text color in all <p> elements on page on click

```
<script language="JavaScript">
   function changeColor(newColor) {
      // provides array of all <p>-elements
      var paragraphs = document.getElementsByTagname("p");

      for(i=0; i<paragraphs.length; i++) {
         paragraphs[i].style.color = newColor;
      }
   }
</script>

<input type="button" value="Blue" onClick="changeColor(blue)" />
```

Event-Handler

DHTML – Dynamic HTML (2/2)

DHTML = JavaScript + DOM (also: DOM Scripting)

- DHTML allows displaying / hiding whole parts of a website (Reminder: CSS properties visibility / display)

  ➔ complex dynamic user interface possible without constantly re-loading the entire page

But: All possible contents must be loaded directly as well (even if they are not displayed)

➔ Performance disadvantage

Better: Dynamic reloading of partial contents

➔ AJAX

openHPI
JavaScript

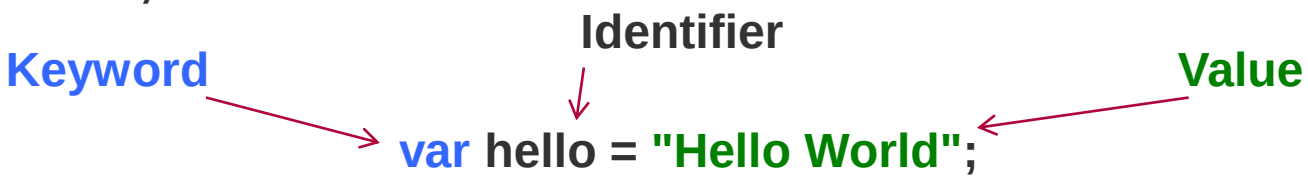
Prof. Dr. Christoph Meinel
Hasso Plattner Institute
University of Potsdam, Germany

JavaScript – Language Basics

- JavaScript is a dynamically typed language
- Primary data types: String, Number, Boolean, Function, Object
- Control structures: Conditions and Loops
- Functions
- Objects and inheritance (via "prototypes")
- Pre-defined objects: e.g. Array, Date, Math, window, document as well as objects from the ➔ **DOM** (Document Object Model)
- EventHandlers to interact with DOM Elements
- JavaScript references on the web:
  - http://www.w3schools.com/jsref/default.asp
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript

JavaScript – Variables and Types

- Variables allow to store **values** for a program and to access them later on by their identifier

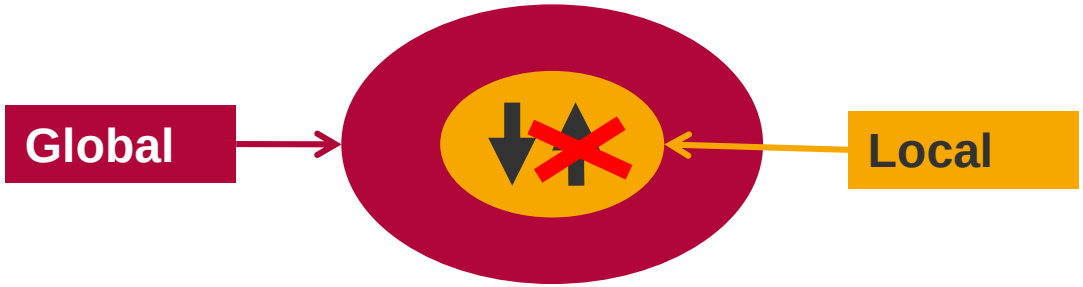


- Types define what sort of values can be stored in a variable, e.g. Strings, Numbers, etc.
- Dynamic typing (e.g. JavaScript) vs. static typing (e.g. Java)

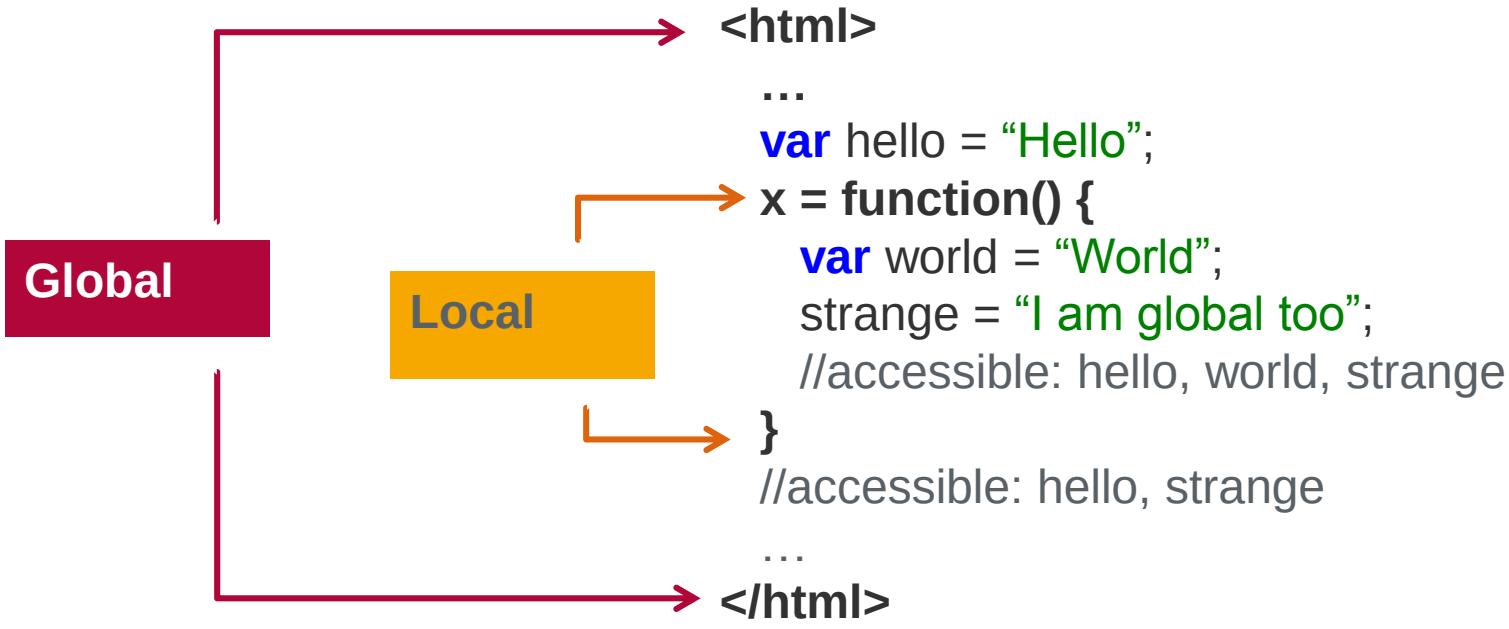
Dynamic Typing	Static Typing
Variables can change their type over time	Variables can only store values of the declared type
Functions can return values of any type	Functions can only return values of the declared type
Functions can take parameters of any type	Functions can only take parameters of the declared type

JavaScript – Variables and Scope (1/2)

- Variables are only accessible within a certain area of the code ➔ Scope
- In JavaScript in general there are two kinds of scopes
  - **Global scope**
  - **Local scope**
- Generally, the availability of a variable trickles down but not up
  - a variable declared in the global scope is available in the local scope
  - a variable declared in the local scope is not available in the global scope



JavaScript – Variables and Scope (2/2)



- In JavaScript the **local** scope is defined by **functions**
- It is limited by curly brackets **{}**
- **Control structures** do not define a scope
- The **global** scope is defined by the **document**

JavaScript – Functions

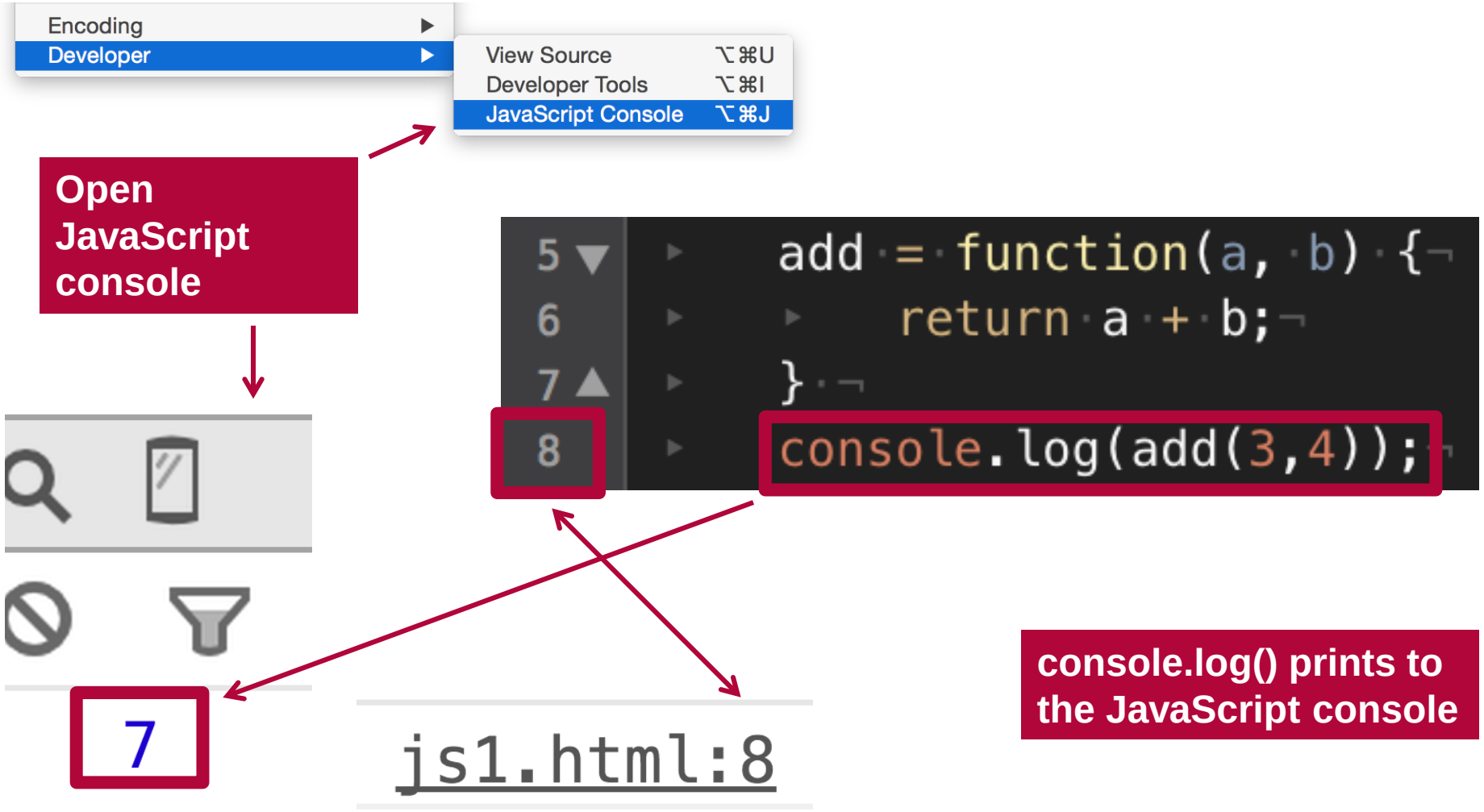
- Functions are small reusable blocks of code to store behavior …
  - … are customizable with parameters
  - … can return a value
  - … should be designed to execute exactly one action
  - … should be given an identifier that reflects this action
  - … should not have side effects
- In JavaScript functions can be stored in variables

```
add = function(a, b) {
    return a + b;
}
```

Definition

Call → `add(3, 4)`

JavaScript – Debug Console



JavaScript – Control Structures

- Conditions
  - if (x) { do y } else { do z }
    - Shorthand → ternary operator: x **?** y **:** z
  - multiple conditions are possible with else if
- Switch / case statements for more complex decisions
- Loops
  - while (x) { do sth. as long as x is true }
  - for (i = 0; i<10; i++) { do sth. 10 times}
  - for (x in array) { do sth. for each element in array }

JavaScript – Data Structures

- Arrays
  - Allow to store multiple elements
  - Allow to iterate over the elements
  - Allow random access of the elements
  - JavaScript only supports numerical arrays, not associative arrays
  - `var birds = [”Eagle", ”Parrot", ”Sparrow"];`
- Objects
  - Can serve as associative arrays
  - `var sparrow = {weightGrams:25, maxLifetimeYears:9};`

JavaScript – Objects

- In JavaScript the keyword **function** is used to define objects
- State is added in the constructor

```
function Ball () {
    this.x = 10;
    this.y = 10;
}
```

- Behavior is added via the prototype keyword

```
Ball.prototype.move = function () {
  this.x += this.deltaX;
  this.y += this.deltaY;
}
```

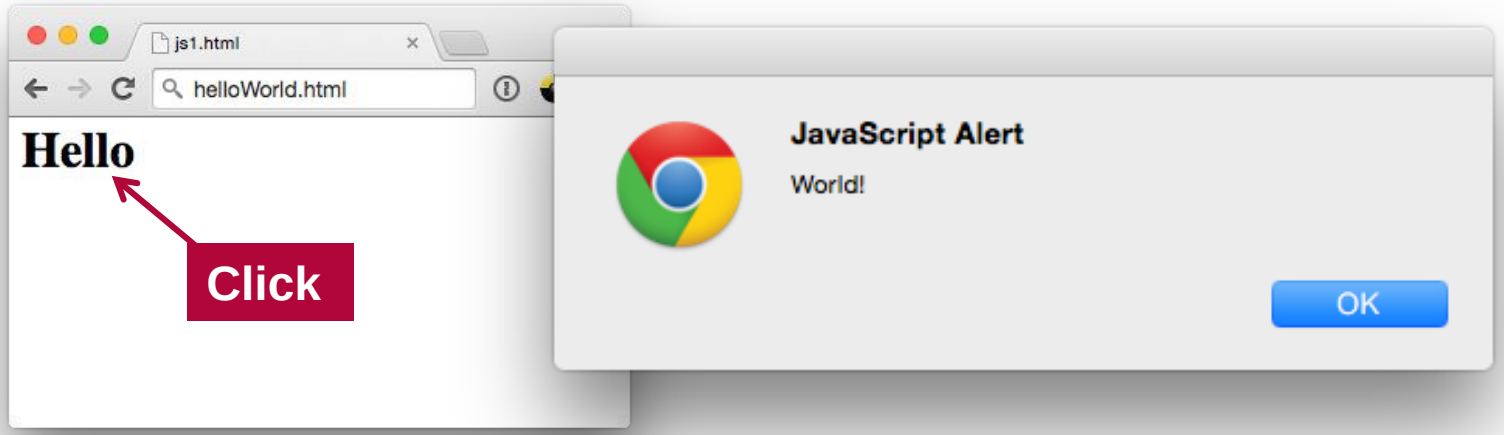
JavaScript – Addressing HTML Elements

- Root Element of DOM: document
- Starting with this, all other HTML elements can be addressed by …
  - … their Id: getElementById(“myID”) ➔ returns element
  - … their Name: getElementsByName(“myName”) ➔ returns array
  - … their HTML-Tag: : getElementsByTagName(“div”) ➔ returns array
- Shortcuts to some elements:
  - document.forms ➔ returns all forms in a document
  - document.images ➔ returns all images in a document
- Frameworks such as jQuery simplify accessing HTML elements

JavaScript – Event Handlers

- Supply HTML elements with additional interactivity
- User interaction with an HTML element, calls a JavaScript function

```
<h1 onclick="alert('World!');">Hello</h1>
```



openHPI

JavaScript Frameworks and Libraries

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

JavaScript Frameworks and Libraries

In (web) software development projects …

- many functions, requirements and tasks are similar or even identical even if the projects are different
  - e.g. an e-learning website as well as an online shop both need user management
- development can be accelerated and hence made more cost-efficient by reusing parts of software or by using provided libraries that have already been programmed by someone else (**libraries**)
- development can be supported by an environment (**framework**) which constitutes a kind of skeleton or structure which only has to be extended / filled in certain places

Frameworks and Libraries – What is the Difference?

What is a **library**?

- A library is a kind of collection of functionality which provides and performs specific, well-defined operations
- **Examples** of libraries: string / data manipulation, visualization, compression, regular expression evaluation

What is a **framework**?

- A framework is a skeleton that has to be filled out with the application
- It is more an environment with functionality and paradigms (for example for the architecture) defining a concept
- The fundamental functionality is still done by the application
- **Examples**: Frameworks for web applications, tests, GUIs

However, exact definitions differ and these terms are often mixed up …

JavaScript Frameworks – Popular Examples (1/3)

- **AngularJS** - https://angularjs.org
  - Maintained by Google
  - Model-View-Controller (MVC) architecture
- **Google Web Toolkit** (GWT) - http://www.gwtproject.org
  - Good for people who come from Java because one can develop in Java which will then be compiled to JavaScript
  - Many tools and good reusability
- **KnockoutJS** - http://knockoutjs.com
  - A Model-View-ViewModel (MVVM) framework
  - Good for dynamic JavaScript UIs

JavaScript Frameworks – Popular Examples (2/3)

- **Backbone.js** - http://backbonejs.org
  - Based on MVC design pattern
  - Very small size
  - Great for single-page applications

- **Cappuccino** - http://www.cappuccino-project.org
  - Look and feel like desktop applications on Mac OS X
  - No knowledge required about DOM or CSS

- **Ember.js** - http://emberjs.com
  - Based on MVC pattern
  - has big templating system

JavaScript Frameworks – Popular Examples (3/3)

- **Aurelia** - http://aurelia.io
  - Still in development, maybe one of the next popular frameworks?
  - Written in ECMAScript 6 and 7 by the makers of Durandal
  - Simple coding conventions, focus on developer experience

All these frameworks offer a lot of functionality and also rules or paradigms of software development or its architecture

Small Reminder:

- You integrate a **library** to call existing functions to extend your code
- A **framework** tells you where to put or how to develop your own app

openHPI

Selected JavaScript Libraries

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

Selected Javascript Libraries

- A **JavaScript library** is a (pre-written) collection of functionality which provides and performs specific, well-defined operations
- They help make the development of JavaScript-based web applications faster and easier
- Some popular Javascript libraries are
  - jQuery
  - jQuery UI
  - DHTMLX
  - Dojo Toolkit
  - Raphaël
  - Three.js
  - D3.js
  - …

jQuery (1/3)

jQuery is the most popular JavaScript library

- Used by more than 60% of the 10,000 most visited websites
- Free and open-source
- Multi-browser support
- Provides
  - DOM navigation and manipulation
  - Extended event system
  - Helper functions such as inArray() and each()
  - AJAX functionality
  - Extendibility through plugins

jQuery (2/3)

Usage and integration are very simple

- The jQuery library is one JavaScript file containing all functions
- It can be included within a web page
  - by linking to a local copy

```
<script src="jquery.js"></script>
```

  - by linking to a copy available on public servers

```
<script src="//code.jquery.com/jquery-2.1.3.min.js"></script>
```

- File will be downloaded to the client and hence is minified (remove breaks and whitespaces) and gzipped → only 32 kB to be downloaded

jQuery (3/3)

- The **$** function (selector)
  - Select the <button> element with the class "overview" and change its HTML to "See the topic overview"

```
$("button.overview").html("See the topic overview")
```

- The **each()** function

```
var myArray = [1, 2, 3, 4, 5];
$.each(myArray, function(){
        document.write(this + 1);
});
```

- **Chaining** - jQuery commands can be chained:

```
$("div.test").add("p.quote").addClass("red").slideDown("slow");
```

Raphaël – A JavaScript Library for Visualization (1/2)

Raphaël - http://raphaeljs.com

- Small JavaScript library especially for working with vector graphics
- Every SVG object is also a DOM object which can be modified
- The file raphael.js has to be downloaded (91 kB) and included in the HTML page
- Easy to learn and good choice for drawing not too complex things like charts etc.

- Extensible through plugins
- Free and open-source (MIT license)

Raphaël – A JavaScript Library for Visualization (2/2)

- Simple example with a canvas and a circle on it

```
// Creates a canvas 50 pixels from the top left corner
   with a size of 300 × 200 pixels
var background = Raphael(50, 50, 300, 200);

// Creates a circle at x = 60, y = 40, with a radius of 20
var circle = background.circle(60, 40, 20);

// Sets the fill attribute of the circle to red (#f00)
circle.attr("fill", "#f00");

// Sets the stroke attribute of the circle to white
circle.attr("stroke", "#fff");
```

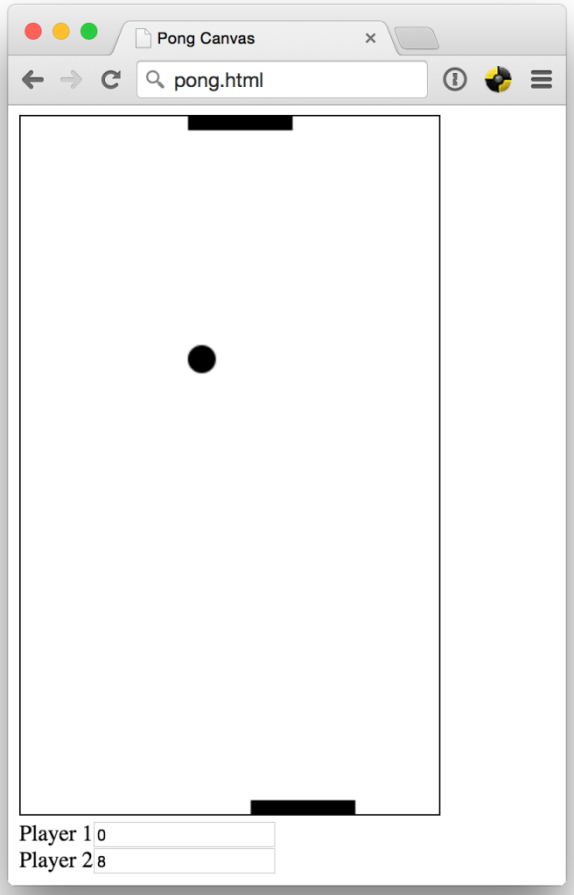
- You can find more examples and a good documentation on the project's website

openHPI

Example Application in JavaScript

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

Example: Application in JavaScript (1/6)

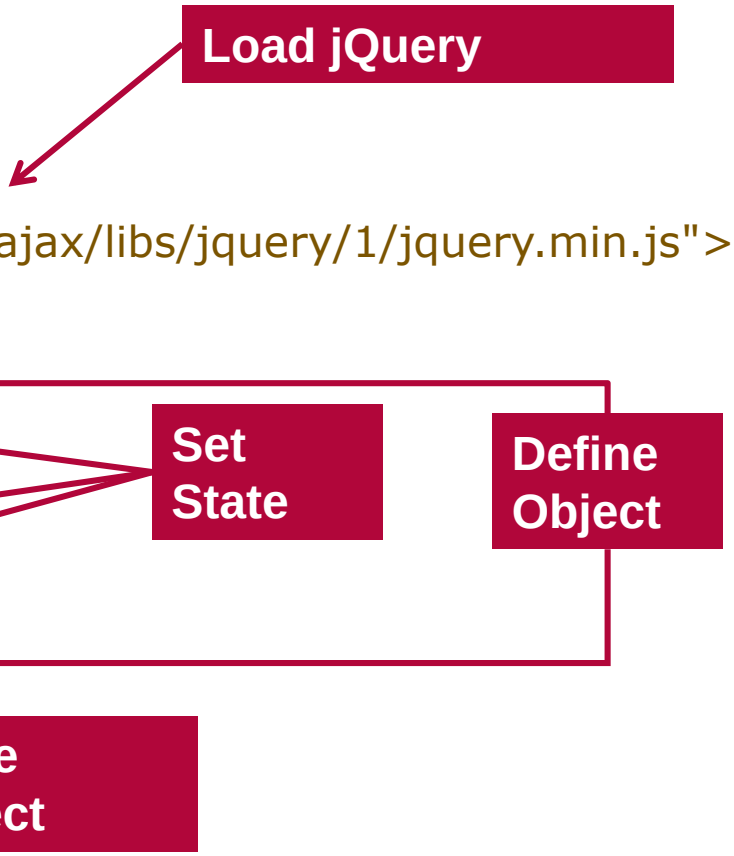


Pong – Game for two players and one ball

- Complete code:
  - see downloads section
- Task:
  - complete the missing elements in the code

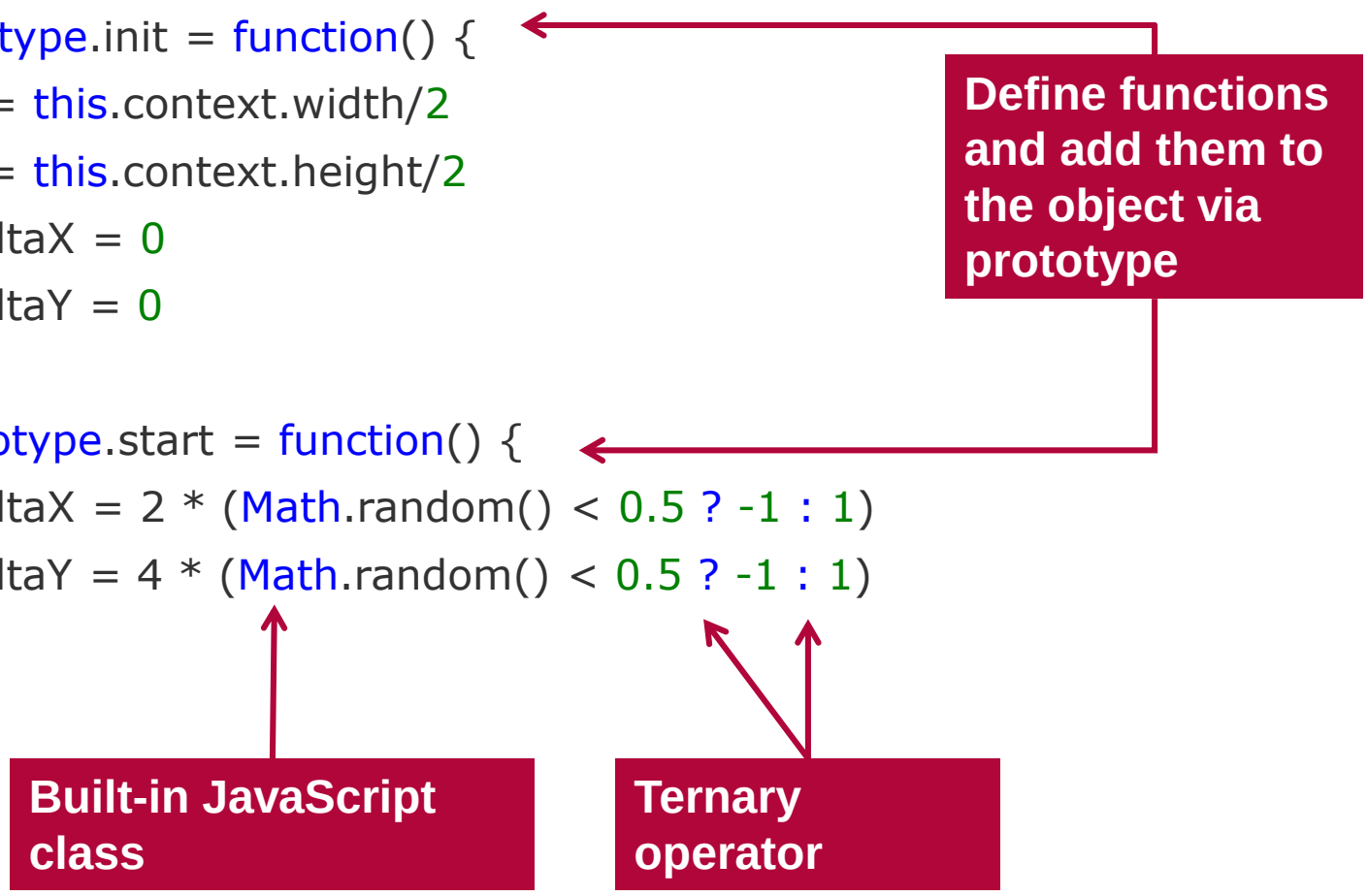
Example: Application in JavaScript (2/6)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Pong Canvas</title>
    <meta charset="utf-8">
    <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
    <script>
      function Ball (context) {
        this.context = context
        this.radius = 10
        this.init()
      }
      ...
```



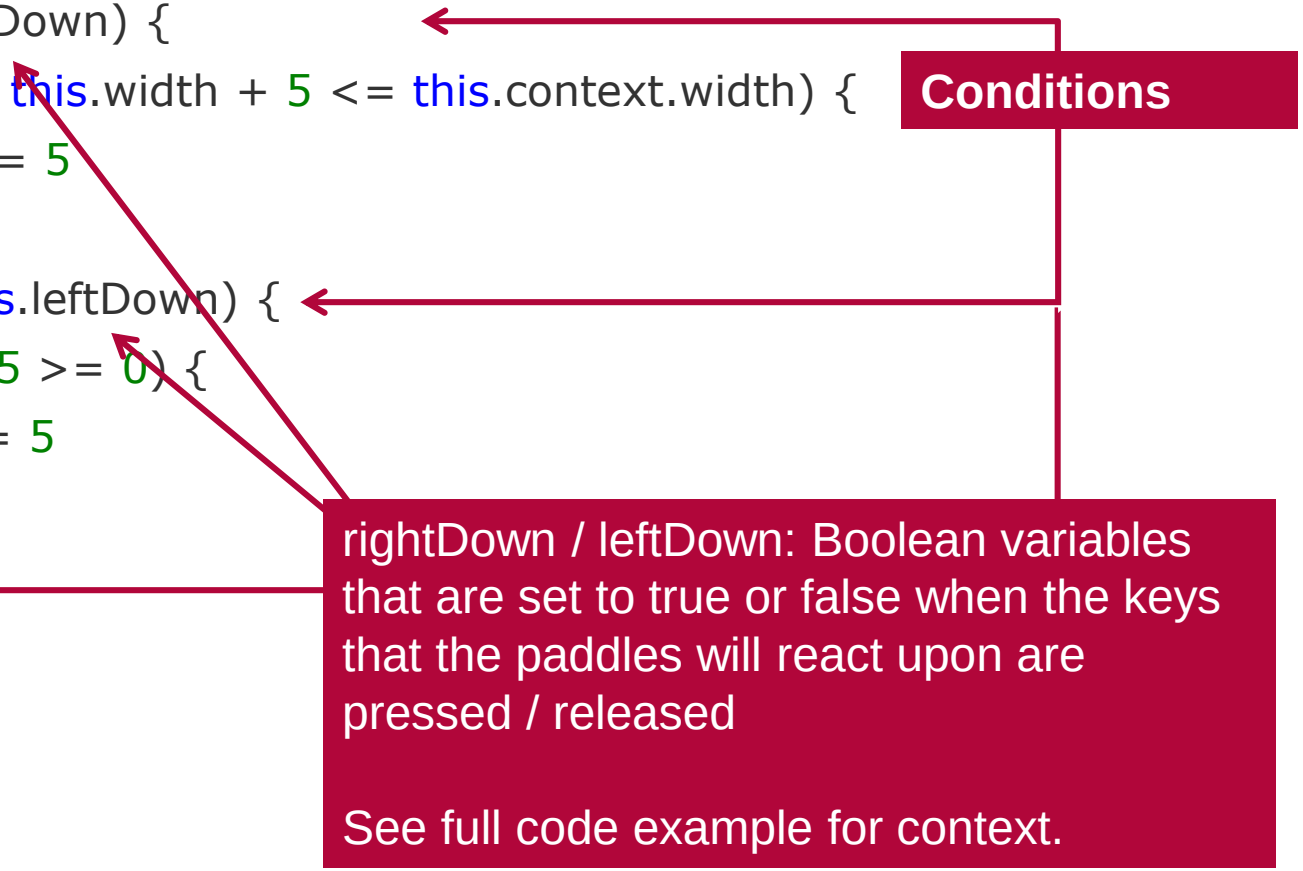
Example: Application in JavaScript (3/6)

```
...
Ball.prototype.init = function() {
    this.x = this.context.width/2
    this.y = this.context.height/2
    this.deltaX = 0
    this.deltaY = 0
}
Ball.prototype.start = function() {
    this.deltaX = 2 * (Math.random() < 0.5 ? -1 : 1)
    this.deltaY = 4 * (Math.random() < 0.5 ? -1 : 1)
}
...
```



Example: Application in JavaScript (4/6)

```
...
Paddle.prototype.move = function () {
    if (this.rightDown) {
        if(this.x + this.width + 5 <= this.context.width) {
            this.x += 5
        }
    } else if (this.leftDown) {
        if(this.x - 5 >= 0) {
            this.x -= 5
        }
    }
}
...
```



Example: Application in JavaScript (5/6)

```
...
  $(document).ready(function() {
      var context = new GameContext();
      var player1 = new Paddle(context, 1);
      var intervalId = setInterval(draw, 10)
      function onKeyDown(evt) {
        if (evt.keyCode == 32) ball.start()
        else {
          player1.onKeyEvent(evt, true);
      }
```

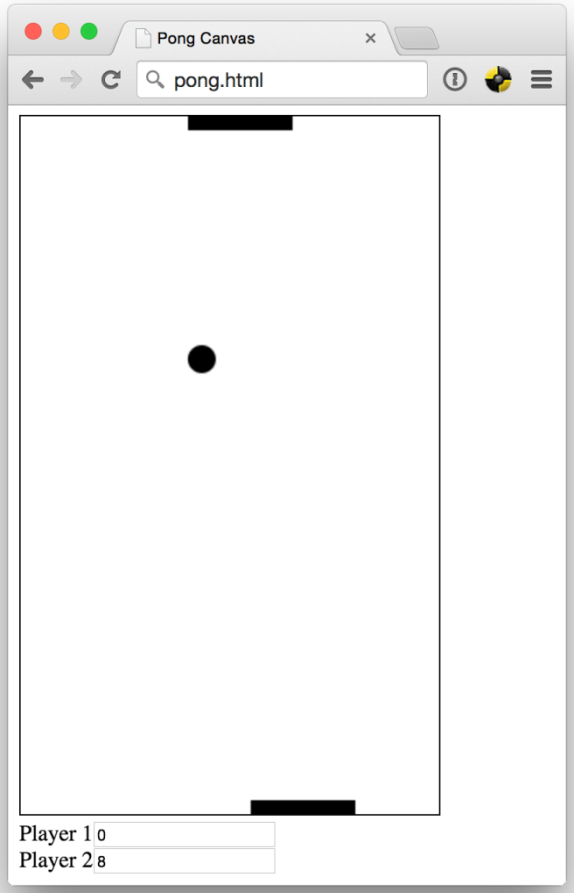
Use jQuery to make sure that all required elements have been loaded when the script is started

Built-in JavaScript function to repeatedly do something in the given interval

Define eventHandler

Create new objects according to their definition

Example: Application in JavaScript (6/6)



Pong

- Download section provides the complete code
- Some lines of code are missing
- Task:
  - complete the missing elements in the code
- Required tools:
  - Browser
  - Text editor e.g.
    - Windows: Notepad++;
    - Linux: gedit, nano, … ;
    - Mac: TextMate, Sublime
  - Do not use a word processor

openHPI

CoffeeScript and Dart

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

CoffeeScript and Dart

There are alternative web programming languages to JavaScript

- **CoffeeScript**, since 2009
  - Initial development by Jeremy Ashkenas
  - Code readability and brevity enhancement
  - Additional features, e.g. pattern matching
  - Sufficiently supported by Ruby community
- **Dart**, since 2011
  - Developed and supported by Google Inc.
  - **Goal**: To build a general-purpose programming language for web development on the open web platform

CoffeeScript – Overview

- Jeremy Ashkenas made the first commit of CoffeeScript on GitHub on December 13, 2009
- Inspired by the syntax of Ruby, Python and Haskell
- Retained the “good parts” of JavaScript
- Strongly improved the brevity and readability compared to JavaScript
- Additional features such as *list comprehension*, *pattern matching*
- Official page: http://coffeescript.org/

CoffeeScript – Compiling

- CoffeeScript needs to be transcompiled to JavaScript
- Due to the 1:1 transcompiling compatible with mainstream JavaScript libraries
- Can be run in any JavaScript environment

```
CoffeeScript vs. JavaScript: a „Hello world“ example

CoffeeScript:
foo = ->
  console.log „Hello world!“

JavaScript:
foo = function(){
    return console.log(„Hello world!“)
};
```

CoffeeScript – More Compact

- Typically 30% less lines of code compared to JavaScript, without changing the runtime performance
- Example: CoffeeScript *functions* are defined by an optional list of parameters in parentheses, an arrow, and the function body

```
CoffeeScript:

square = (x) -> x*x
cube = (x) -> square(x) * x
```

```
JavaScript:

var cube, square;
square = function(x) {
    return x * x;
};
cube = function(x) {
    return square(x) * x;
};
```

CoffeeScript – Issues

- Poor debugging support is a big issue
  - The compiler provides merely very rudimentary diagnostic information for error reporting
  - The compiled code is not always readable
- Increased maintenance effort
  - Sensitive to whitespace, counting a tab as a single space
  - A compile-to-JS language, the transcompiling step is always required

Dart – Overview

- Dart is an ECMA standard web programming language
- Developed and maintained by Google Inc.
- Designed as a general-purpose programming language intended to overcome the shortcomings of JavaScript
- Dart is a class-based, single inheritance, object-oriented language with **C-style** syntax
- Official page: https://www.dartlang.org/

Dart – Benefits and Tools

- A structured and flexible programming language for the Web
- Based on C-style syntax -> Easy to learn
- The language constructs enable high performance and fast program-startup
- Particularly suitable for web-communication devices, such as phones, tablets, laptops and web servers
- Dart can run on all major web browsers based on the support of Dart SDK (Software Development Kit), which includes:
  - a standalone-VM: Dart VM (virtual machine)
  - JavaScript transcompiler: *dart2js*
  - Other tools for coding, code analysis and formatting:
    - *Dart Editor*, *dartanalyzer, dartfmt, Observatory etc.*

Dart – Usage and Compiling

3 ways to run Dart code:

- Source-to-source pre-compiled as JavaScript using the *dart2js* tool
- In the Dartium browser
  - Using Dart SDK with modified Chromium browser (integrated with a Dart VM)
- The Dart SDK also ships with a standalone VM, allowing Dart code to run in a command-line environment.

Dart example for printing Fibonacci numbers:

```
int fib(int n) => (n > 1) ? (fib(n - 1) + fib(n - 2)) : 1;
void main() {
   print('fib(20) = ${fib(20)}');
}
```