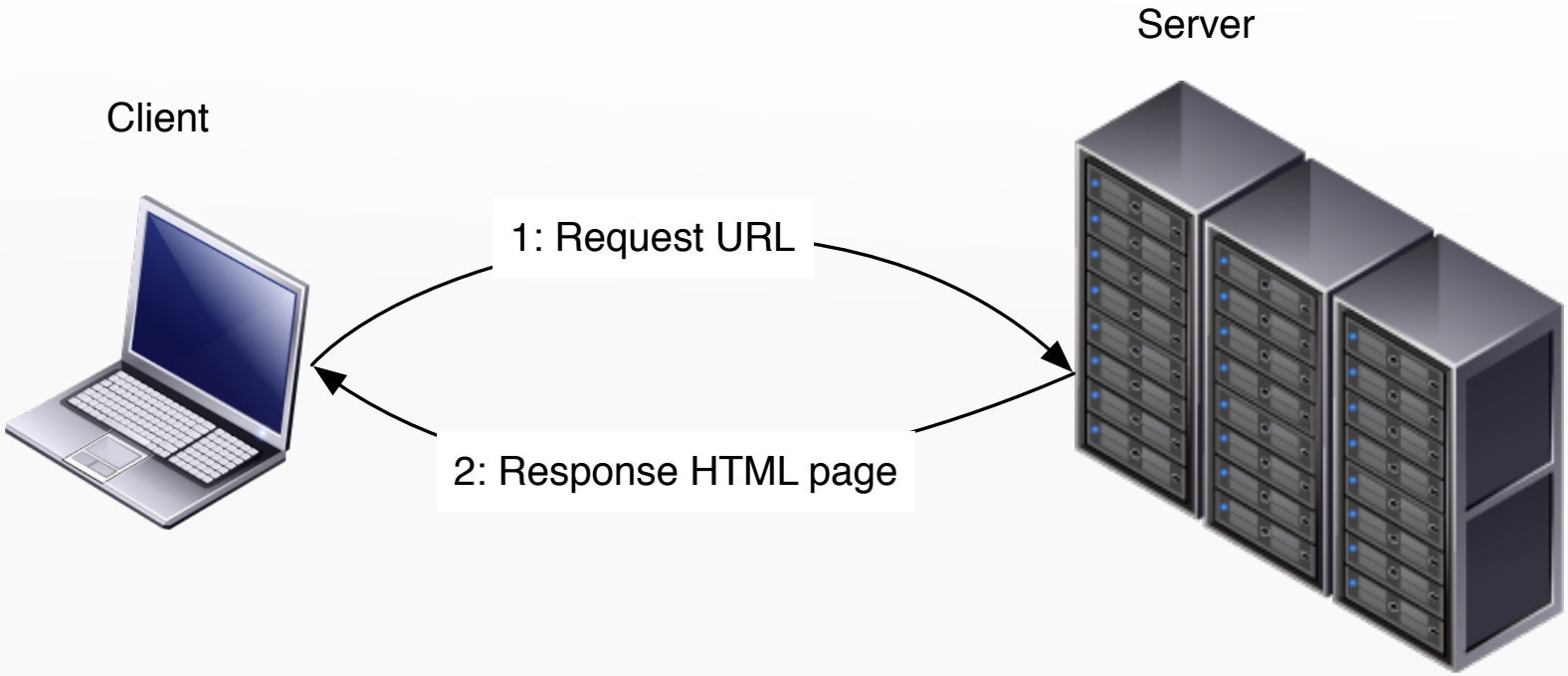

JavaScript

JavaScript in this course

- Learning JavaScript gives us
  - Perspective:
    - We will look at what we already know from a new point of view
    - Our Tomcat server now becomes a web service for JavaScript!
  - Reflection:
    - Is server side or client side programming right for my application?
    - What are the trade-offs compared to server programming?
  - Abstraction: What you already know is not Java specific
    - DOM follows the same model as JDOM
    - HttpUrlConnection and AJAX serve the same purpose!
    - HTML can be constructed on the client too!

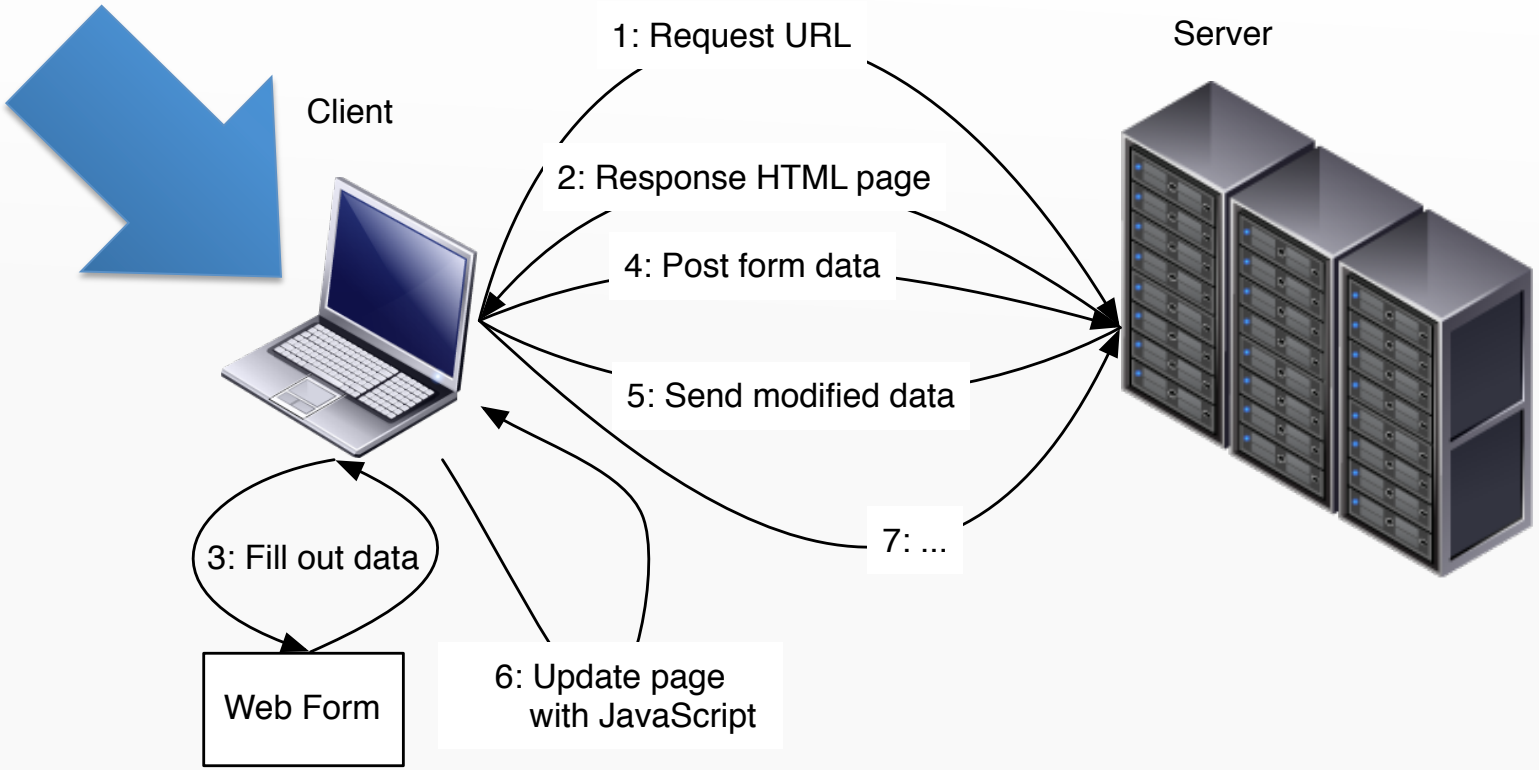
JavaScript: Client side

Simple web interaction

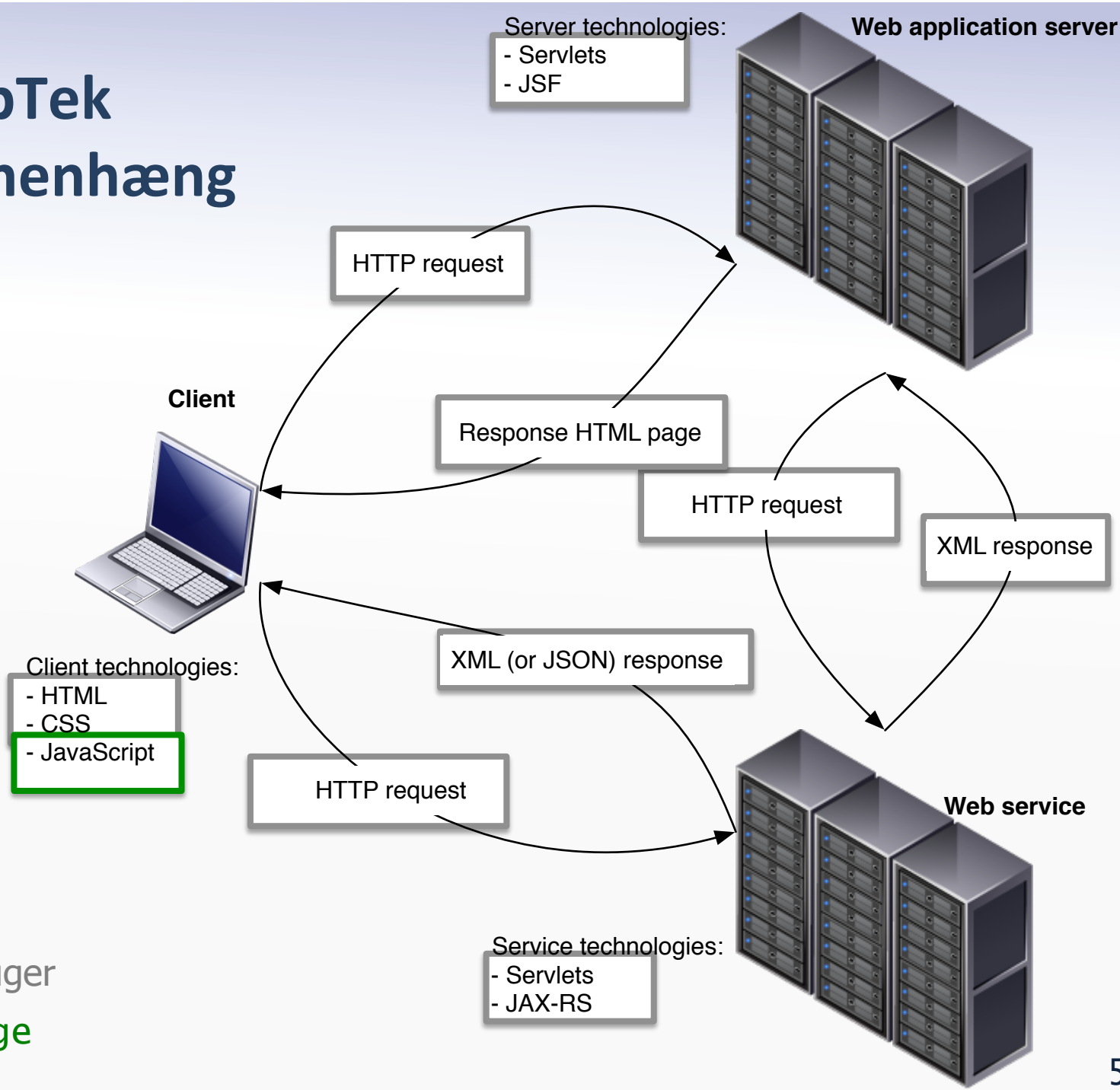


JavaScript: Client side

Modern web interaction



dWebTek sammenhæng



Tidligere uger

Denne uge

Overview

- **Crash course in JavaScript**
- Built-in types and objects
- JavaScript in the browser
- DOM programming
- Event registration
- AJAX
- JSON
- JAX RS and JavaScript example

JavaScript

- Similarity to Java
  - Name
  - Some of the syntax
  - Object orientation (sort of)
- In this course we will use a small but sufficient subset
  - Functions and methods
  - Objects and constructors
  - Control structures like `if`, `for` and `while`
- We will leave out:
  - Inheritance (which is weird)
  - Design patterns for encapsulation
  - …

Dynamic typing

- JavaScript has **no static type checking**

- Every value has a type at runtime
- Automatic **type conversions**,
  rather than runtime errors

- The type of any variable may change during execution!

Why not Java (applets)?

- Java works well on the server, not on the client:
  - Too slow VM start-up
  - Poor integration with the HTML page
  - Smartphones have poor Java support
- Even NemId are switching to JavaScript soon

JavaScript vs Java

- In Java but not in JavaScript
  - Static types:

```
var s = "foo";
s = 10;
```

  - Classes (but constructors do exist!)
  - The huge standard library
- In JavaScript (and similar in Java 8) :
  - Function objects and values:

```
var s = function() { return "foo"};
s();
```

- We will focus on the parts that look like Java!

Hello World!

```
<html>
 <head><title>Hello World!</title></head>
 <body>
  <script type="text/javascript">
   window.alert("Hello World!")
  </script>
 </body>
</html>
```

Add JavaScript to HTML with the `script` tag

Functions in JavaScript

- We declare functions using the `function` keyword:

```
function addNumbers(number1, number2) {
   return number1 + number2;
}
```

- We can declare **anonymous functions**:

```
var addNumbers = function(number1, number2) {
   return number1 + number2;
}
```

  Not unlike anonymous classes in Java

- Functions are objects and can be passed as arguments

More about anonymous functions

- **Callback**: Very common pattern in JavaScript :
  - Call a function with another function as an argument
  - The argument function is invoked in the called function
  - The argument function is typically anonymous:

```
foo(f) {
   return f(2,2); //Returns 4
}

foo(function (number1, number2) {
   return number1 + number2;
});
```

  - We will see this used for server communication

Closures in JavaScript

- **Closure**: a function plus bindings of non-local variables
- We can return function (like any other value)
- Functions have access to variables in outer scopes:

```
function addTo(number) {
   return function(number2) {
        return number + number2;
   }
}
```

- The anonymous, inner function stores `number`
  - A closure!
- This means: Callbacks can refer to outer variables!

Objects in JavaScript

- Objects are values in JavaScript (like in Java)
  - Objects have fields (called properties in JavaScript)
  - Objects have methods (properties that contain functions)
- Objects can be constructed with constructors:

```
var array = new Array(); //An ArrayList
array.push(42); //Add an element to the list
array[0]; //Gets a value, same as in Java
```

- Objects can be constructed as literals

```
var o = {name: "Mathias", age : 29};
var age = o.age;
```

- New fields can be added on the fly!

Person example in Java

This slide is Java, not JavaScript!

- In Java, we write **classes** to define structure of objects

```
class Person {
   String name;
   Person(String name) {
        this.name = name;
   }
}
....
   //in a method
   Person p = new Person("John Doe")
```

- In JavaScript, we don't have classes

Functions as constructors

- In JavaScript, we only have constructors
  - Any function can be a constructor!

```
function Person(n) {
  this.name = n;
}

var p = new Person("John Doe");
if (p instanceof Person) {
   ... //p is indeed a Person
}
```

Now p is a new object with a name property

- Even without classes, constructors work!
- Always use `this` when store fields
  - Is not added automatically like in Java

Don't forget this

```
var person = {
  age: 42,
  getAge: function() {return age;}
}

print(perspn.getAge());
```

uncaught JavaScript runtime exception: ReferenceError: "age" is not defined.

```
var person = {
  age: 42,
  getAge: function() {return this.age;}
}

print(person.getAge());
```

42

Methods in JavaScript

A **method** is a function that is associated with an object:

```
var counter = {
  value: 0,
  inc: function() { this.value++ }
};
counter.value; // is 0
counter.inc();
counter.inc();
counter.value; // is 2
```

`inc` is now a method

`this` is bound to the current object

`this.value++` is ***not*** the same as `value++`

Almost like Java methods

Writing function body code

- We have the usual constructs available:

```
if (b) {
   //do something
}
else {
   //do something else
}
```

```
while (b) {
   //do something until b is false
}
```

```
for (var i = 0; i < list.length; i++) {
   //iterate through the array
}
```

Clicker question

- What is the value of x in the following code

```
function addTo(number) {
   return function(number2) {
        return number + number2;
   }
}
var x = addTo(2)(2);
```

- Answers:
  - 2
  - 4
  - A syntax error

Overview

- Crash course in JavaScript
- **Built-in types and objects**
- JavaScript in the browser
- DOM programming
- Event registration
- AJAX
- JSON
- JAX RS and JavaScript example

Types

Built-in types:

- boolean (`true`, `false`)
- number (IEEE floating point)
- string (UTF-16)
- null (`null`)
- undefined (`undefined`)
- **object**

"primitive"

Note: *no integer type! no byte type!*

Predefined objects

- `Boolean`
- `Number`
- `String`

wrapper objects

- `Function`
- `Object`
- `Array`
- `Date`
- `Math`
- `RegExp`
- …

similar to the core classes in Java

+ "the global object" (=`window`, in browsers)

Some interesting operators

- Assignment: a=b
  - Like in Java
- Equality: a==b (allows coercion, e.g. "2" == 2)
  - Like o.equals(o2) in Java
- Identity: a===b
  - Like == in Java
- a || b
  - Like in Java but a and b do not have to be booleans!
- a && b
  - Like in Java but a and b do not have to be booleans!

Arrays

```
var x = [ 1.1, true, "a" ];
var y = new Array(1.1, true, "a");
y.push(2); //adds a value to the array

x[10000] = 42;  // All indices exist, so no error here!
```

- Arrays are similar to lists in Java
- In JavaScript arrays are a special kind of objects
  - **`Array`** has a lot of array-specific methods
  - the **`length`** property is automatically updated
  - setting **`length`** expands or truncates

Clicker question

- What is the value of the `length` property of `a`?

```
var a = [1, 2];
a.push(4);
a.push(5);

var l = a.length;
```

- Answers
  - 2
  - 4
  - 5

Overview

- Crash course in JavaScript
- Built-in types and objects
- **JavaScript in the browser**
- DOM programming
- Event registration
- AJAX
- JSON
- JAX RS and JavaScript example

JavaScript in a browser environment

– *JavaScript code embedded in HTML/XHTML documents*

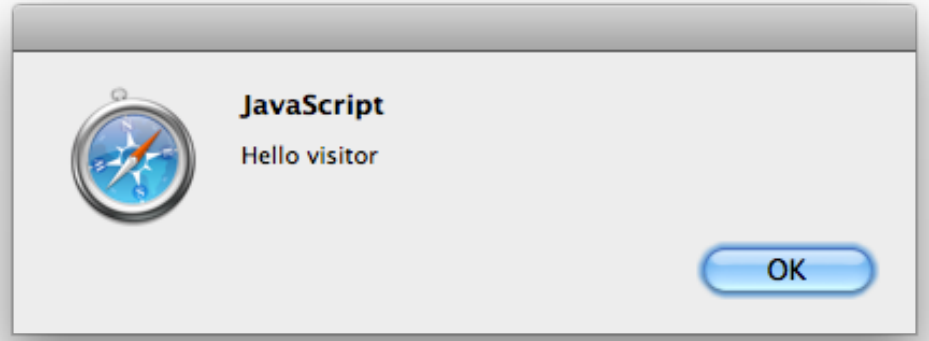
- Access to
  - the **DOM** (Document Object Model, W3C)
    - document tree structure (HTML / XML)
    - CSS properties
    - events
  - general browser state (windows, history buffer, ...)

- See Mozilla's DOM reference: RECOMMENDED
  http://developer.mozilla.org/en/docs/Gecko_DOM_Reference

A few browser functions

- The `window` object provides a basic browser API
- Example:

```
window.alert("Hello visitor!");
```



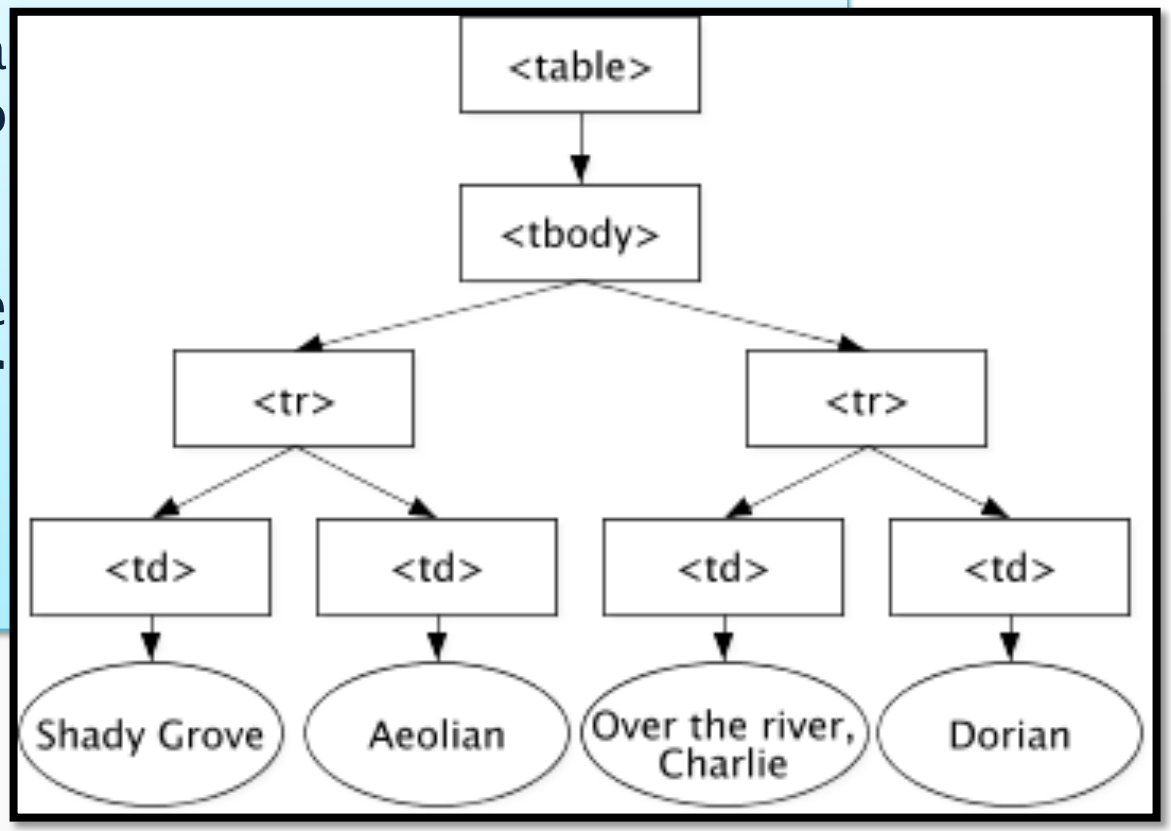
- Other window functions:
  - `window.prompt`
    takes input from the user
  - `window.close`
    closes the window
  - `window.open` opens a new window
  - `window.document` contains the page being shown
  - `window.location` for reading or changing the current URL
  - `window.setTimeout` for performing an operation later

Overview

- Crash course in JavaScript
- Built-in types and objects
- JavaScript in the browser
- **DOM programming**
- Event registration
- AJAX
- JSON
- JAX RS and JavaScript example

Example (from DOM spec)

```
<table>
  <tbody>
    <tr>
      <td>Sha
      <td>Aeo
    </tr>
    <tr>
      <td>Ove
      <td>Dor
    </tr>
  </tbody>
</table>
```



DOM: very much like JDOM

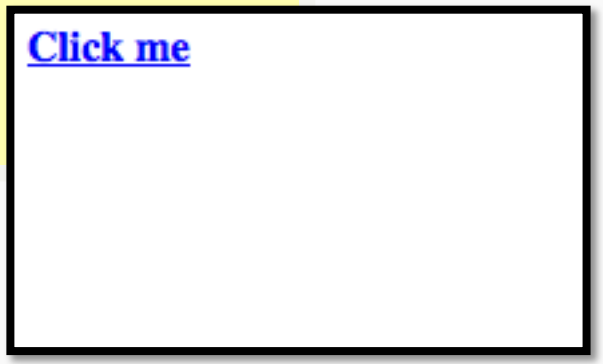
- DOM:
  - Tree-structure: DOM nodes (elements, text) are objects
  - Each node has an array (a list) of children
- Initial DOM structure:
  - The result of parsing an HTML document
- Updates to the DOM Structure:
  - Change the view in the browser!

DOM, actual model

- W3C standard
  - most of it supported uniformly across all browsers
- Each node has
  - a `childNodes` array
  - an `attributes` array
  - a `parentNode` field
  - …
  - `appendChild`, `removeChild`, `insertBefore`,… functions to update the tree
  - `getElementsByTagName` for querying descendants

DOM example

```
var links = document.getElementsByTagName("a");

for (var i = 0; i < links.length; i++) {
  var link = links[i];
  while (link.childNodes.length > 0) {
    link.removeChild(link.firstChild);
  }
  var click = document.createElement("b");
  var text = document.createTextNode("Click me");
  click.appendChild(text);
  link.appendChild(click);
}
```



Parsing HTML: innerHTML

Alternative to creating DOM elements directly

- Assign to `innerHTML` property
- Automatically parsed and inserted as content
- Replaces existing content (if any)

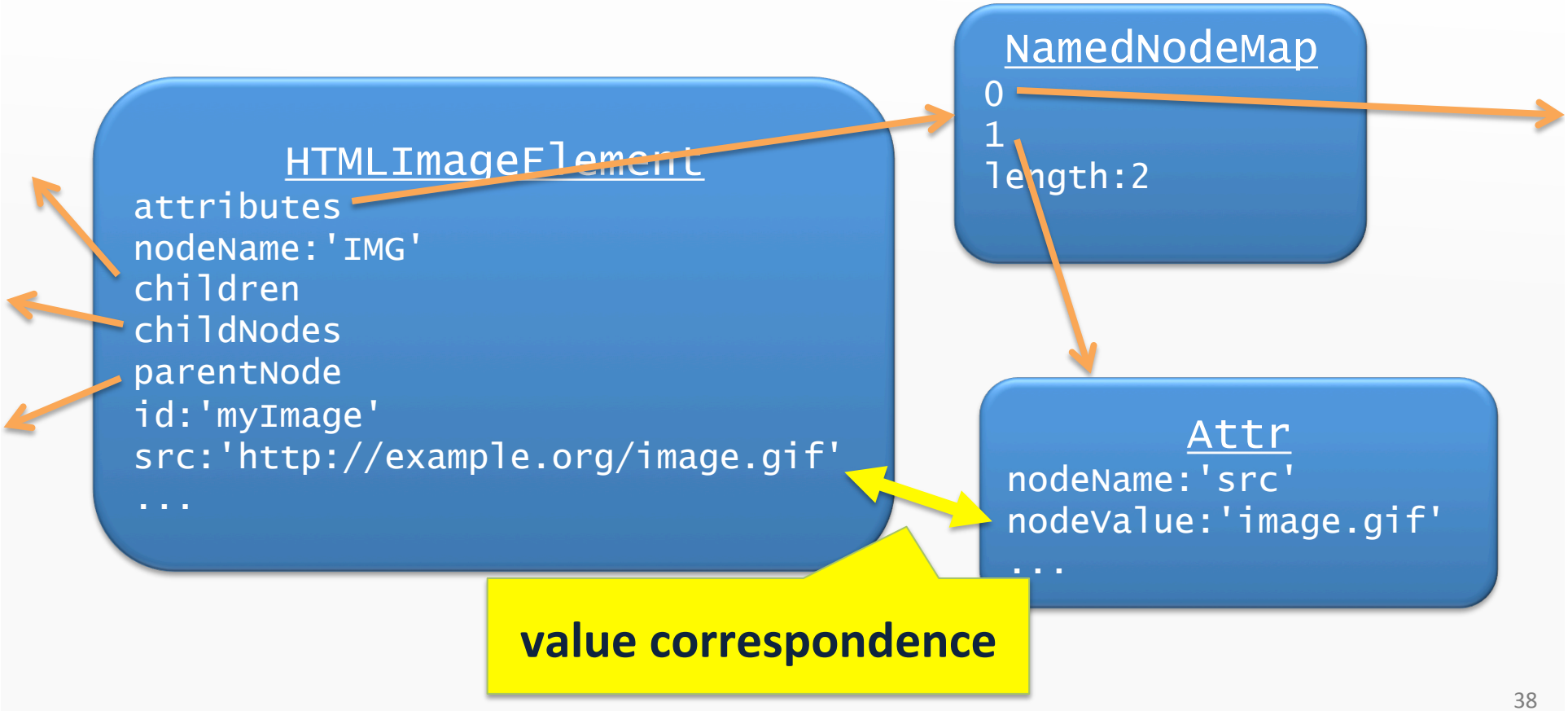
```
var links = document.getElementsByTagName("a");

for (var i = 0; i < links.length; i++) {
   var link = links[i];
   link.innerHTML = "<b>Click me</b>”
}
```

DOM HTML bindings

- In addition, DOM element objects have:
  - Properties corresponding to attributes
    - `img` element objects have a `src` property
    - `a` element objects have an `href` property
- Often more convenient than DOM alone
- The browser keeps properties in sync
  - Value correspondence

DOM HTML bindings

- Adds special properties to HTML objects
- The HTML code
  `<img id="myImage" src="image.gif"/>`
  creates a lot of JavaScript objects with a lot of properties:



DOM HTML example

```
var links = document.getElementsByTagName("a");

for (var i = 0; i < links.length; i++) {
  var link = links[i];
  link.href = "http://cs.au.dk/dWebTek";
}
```

changes the `href` attribute of the link,

the browser updates the view accordingly

Querying nodes in DOM

- Support for finding DOM nodes:
  - `document.getElementById(n)`
    - Returns the element with `id` attribute equal to `n`

```
var link = document.getElementById("foo");
link.href = "...";
```

  - `document.getElementsByTagName(name)`
    - Returns an array of elements with the given tag name
  - `document.querySelectorAll(css)`
    - Returns an array of elements that match the given css selector

- Only work well after the document is loaded fully

Changing CSS styles with DOM

- A `style` property on all elements
- All CSS properties are defined on the style object:
  - e.g. `style.visibility` holds the value of the CSS visibility property
  - naming conventions are slightly different, e.g. `backgroundColor` in DOM for `background-color` in CSS
- Changes to these properties change the view

- More about this with jQuery next week

DOM HTML example

```
var links = document.getElementsByTagName("a");

for (var i = 0; i < links.length; i++) {
  var link = links[i];
  link.style.visibility = "hidden";
}
```

changes the CSS property ‘visibility’ to hidden meaning that the link is no longer shown

Clicker question

- What does this code do?

```
var els = document.getElementsByTagName("img");

for (var i = 0; i < els.length; i++) {
  var el = els[i];
  el.src = "flag.gif";
}
```

- Sets `src` attribute of all images to `flag.gif`
- Sets `src` attribute of the first image to `flag.gif`
- Nothing until page is reloaded

Overview

- Crash course in JavaScript
- Built-in types and objects
- JavaScript in the browser
- DOM programming
- **Event registration**
- AJAX
- JSON
- JAX RS and JavaScript example

Event handling

- In JavaScript, **events** happen when
  - Something is clicked
  - The mouse is moved
  - A keyboard key is pressed
  - ...
- We can **listen** to these events
  - Register a function that is called when the event happens

Registering listeners

- IE:

```
myNode.attachEvent("onmousemove", myHandlerFunc);
myNode.detachEvent("onmousemove", myHandlerFunc);
```

- Firefox et al.:

```
myNode.addEventListener("mousemove", myHandlerFunc);
myNode.removeEventListener("mousemove", myHandlerFunc);
```

- Using feature detection (and a closure):

```
function addEventListener(myNode, eventType, myHandlerFunc) {
    if (myNode.addEventListener)
        myNode.addEventListener(eventType, myHandlerFunc);
    else
        myNode.attachEvent("on" + eventType, function (event) {
                myHandlerFunc.call(myNode, event);
            });
}
```

Event listener example

```
<html>
    <head>
        <script type="text/javascript">
            function show_alert() {
                alert("dWebTek is fun!");
            }
            window.onload = function() {
                var b = document.getElementById("button1")
                addEventListener(b, "click", show_alert);
            }
        </script>
    </head>
    <body>
        <input type="button" value="Click me!" id="button1" />
    </body>
</html>
```

When the button is clicked, we show the box

Overview

- Crash course in JavaScript
- Built-in types and objects
- JavaScript in the browser
- DOM programming
- Event registration
- **AJAX**
- JSON
- JAX RS and JavaScript example

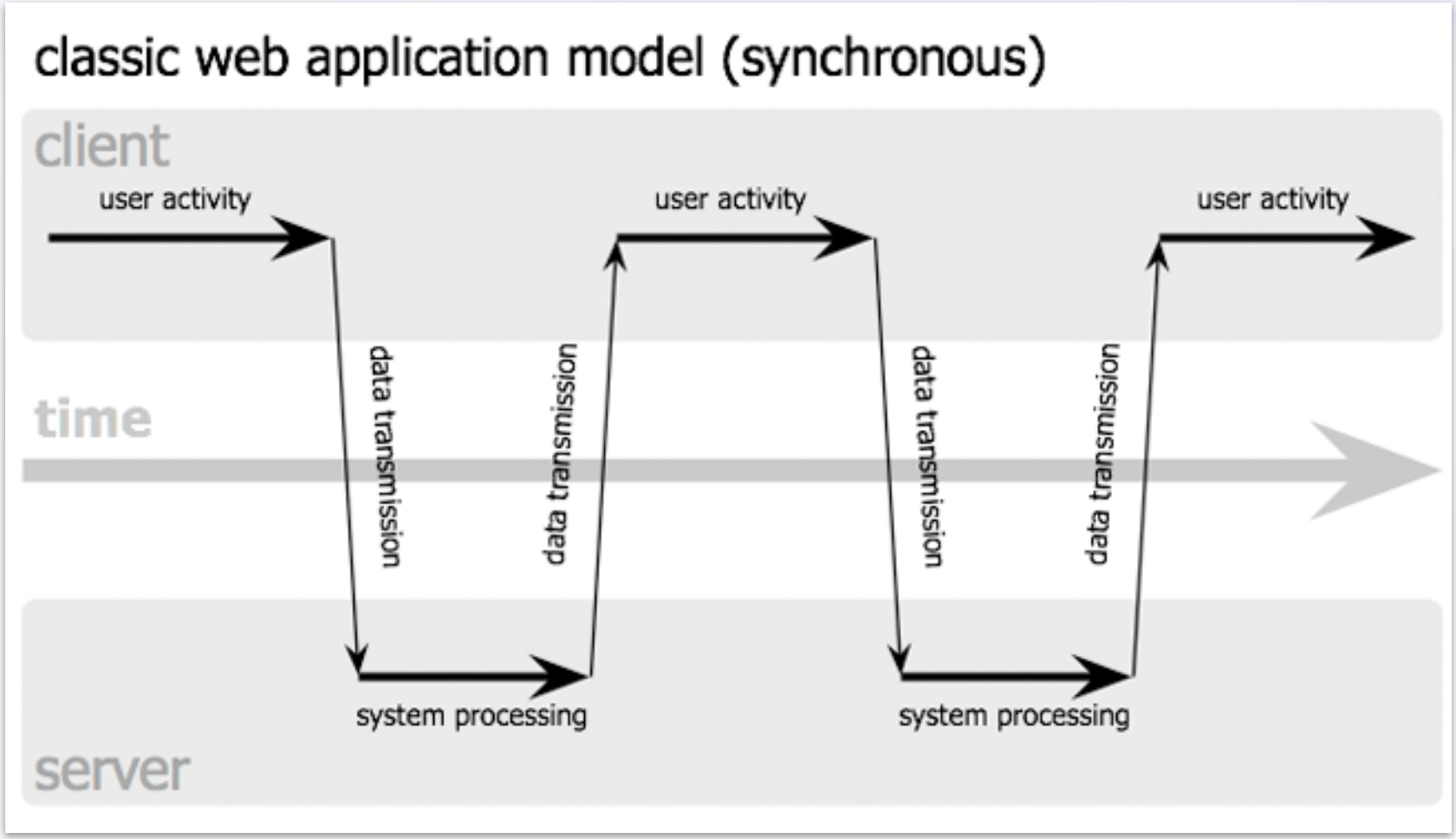
AJAX

– ***A****synchronous* ***J****avaScript* ***a****nd* ***X****ML*

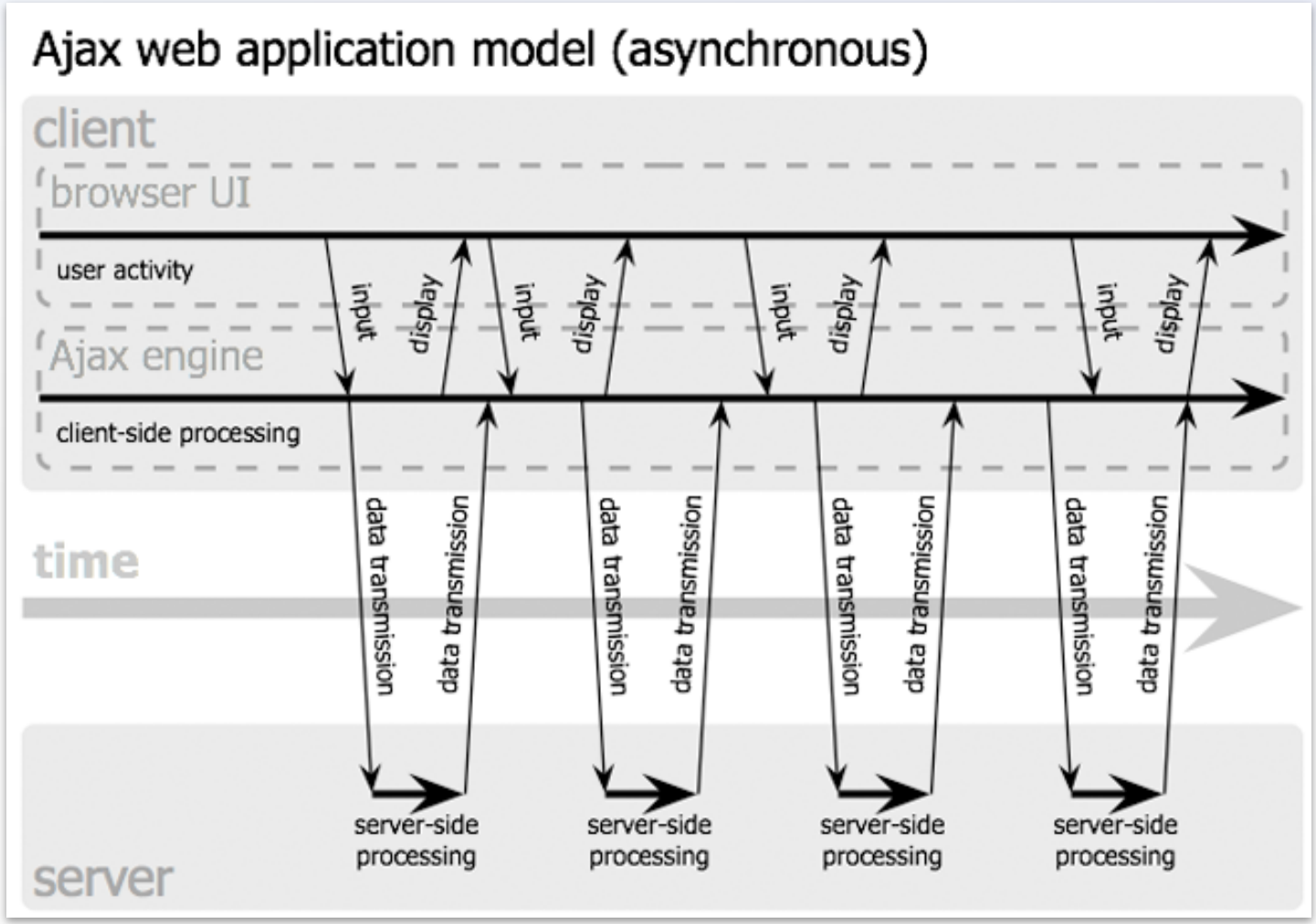
- Allows JavaScript code on the client to issue HTTP requests to the server
- We can use this to contact out JAX RS Service!

- Like HTTPUrlConnection in Java
  - Uses callbacks
  - Simpler design?

AJAX


classic web application model (synchronous)
client
user activity
user activity
user activity
data transmission
data transmission
data transmission
data transmission
time
system processing
system processing
server

AJAX


Ajax web application model (asynchronous)
client
browser UI
user activity
input
display
Ajax engine
client-side processing
data transmission
time
server-side processing
server

AJAX example

A primitive AJAX library:

```
var http;
if (!XMLHttpRequest)
    http = new ActiveXObject("Microsoft.XMLHTTP");
else
    http = new XMLHttpRequest();

function sendRequest(httpMethod, url, body, responseHandler) {
    http.open(httpMethod, url);
    http.onreadystatechange = function () {
        if (http.readyState == 4 && http.status == 200) {
            responseHandler(http.responseText);
        }};
    http.send(body);
}
```

AJAX examples

A primitive AJAX library, usage examples:

```
//Send a GET request to "/query" without an argument
sendRequest("GET", "/query", null, function(value) {
   //This code is run when GET data is received
});

//Include a "q" parameter
var foo = encodeURIComponent("Mathias");
sendRequest("GET", "/query?q=" + foo, null, function(value) {
   //This code is run when GET data is received
});

//Send a POST request with a parameter in request body
//Use this for updating values on the JAX RS service
sendRequest("POST", "/query", "q=" + foo, function(value) {
   //...
});
```

AJAX polling

- AJAX polling pattern for fetching new data
  - Request data at a fixed time interval
  - window.setTimeout allows us to run a method after a delay

```
var update = function() {
   sendRequest("GET", "/query", null, function(value) {
      //Update the DOM with new information
      //Run again after 2000 milliseconds
      window.setTimeout(update, 2000);
   }
}
window.setTimeout(update, 2000);
```

- Run the method and after each run, set timeout again

Clicker question

- We want to implement a chat application, and we want to fetch and show messages from other clients in our window while the client is viewing the page. How would we do this?
  - Using event handling
  - Using AJAX polling
  - Using the built-in Chat object in JavaScript

Client-side security

- No access to client file system etc.
- but the JavaScript code can mimic user events (clicking on links, submitting forms, …)

- **Same-origin policy:**
  - a script can read only the properties of windows and documents that have the *same origin* as the document containing the script
  - `XMLHttpRequest` can only contact the origin server
    - unlike dynamically generated `iframe` and `script` tags, images, and forms
    - `XMLHttpRequest` allows the JavaScript code to inspect the response, so the same-origin-policy can help preventing Cross-Site-Request-Forgery attacks

Overview

- Crash course in JavaScript
- Built-in types and objects
- JavaScript in the browser
- DOM programming
- Event registration
- AJAX
- **JSON**
- JAX RS and JavaScript example

JSON

– ***Java**Script **O**bject **N**otation*

- A light-weight alternative to XML for data interchange (in particular, of JavaScript values)
  – typically used with AJAX
- Same syntax that we use for writing literal objects
  - Easy to parse in JavaScript, easy to use
- See http://www.json.org/

JSON in Java and JavaScript

On the Java Server

```
JSONObject person = new JSONObject();
person.put("name", "Mathias");
person.put("age", 29);
return person.toString();
```

```
{
   "name": "Mathias",
   "age" : 29
}
```

On the JavaScript client

```
var person = JSON.parse(personString);
var name = person.name; //"Mathias"
var age = person.age; //29
```

Very convenient on the JavaScript side!

AJAX and JAX RS example

- We now write a JAX RS web service
  - We contact it from JavaScript
  - Our web service may contact other web services
    - Service oriented architecture
- This example becomes a part of your web shop code
  - Example can be downloaded from the course page
- The server holds items, the client gets and shows them
  - "View" part of application entirely on the client

JAX RS Server

- Create a JSON array of items

```
@GET
@Path("items")
public String getItems() {
   JSONArray array = new JSONArray();
   JSONObject jsonObject1 = new JSONObject();
   jsonObject1.put("id", 1);
   jsonObject1.put("name", "Stetson hat");
   jsonObject1.put("price", 200 + priceChange);
   array.put(jsonObject1);

   return array.toString();
}
```

- You will do this based on cloud server data!

Client JavaScript code 1/2

- Register a function to be called when document ready

```
window.onload = function () {
    sendRequest("GET", "rest/shop/items", null,
        function (itemsText) {
            var items = JSON.parse(itemsText);
            addItemsToTable(items);
    });
}
```

- Sends a request to the server
- Calls `addItemsToTable` with result parsed as JSON

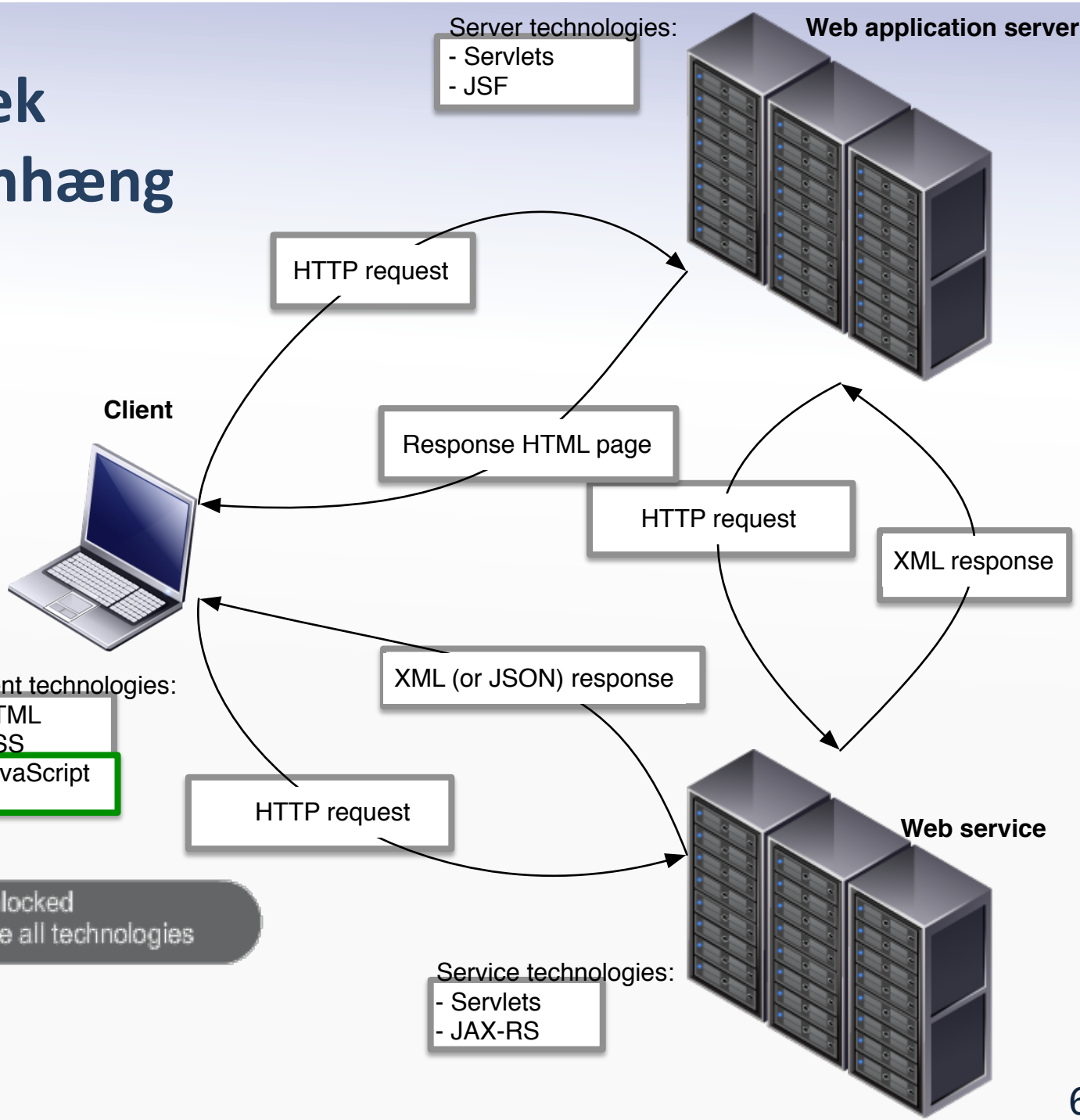
Client JavaScript code 2/2

```
function addItemsToTable(items) {
   var tableBody =
   document.getElementById("itemtablebody");
   tableBody.innerHTML = "";
   for (var i = 0; i < items.length; i++) {
        var item = items[i];
        var tr = document.createElement("tr");

        var nameCell = document.createElement("td");
        nameCell.textContent = item.name;
        tr.appendChild(nameCell);
        var priceCell = document.createElement("td");
        priceCell.textContent = item.price;
        tr.appendChild(priceCell);

        tableBody.appendChild(tr);
    }
}
```

Summary

- Crash course in JavaScript
- Built-in types and objects
- JavaScript in the browser
- DOM programming
- Event registration
- AJAX
- JSON
- JAX RS and JavaScript example

dWebTek sammenhæng



Tidligere uger

Denne uge

Links

- JavaScript guide and DOM reference:
  http://developer.mozilla.org/en/docs/JavaScript

- Technical(!) specification of ECMAScript (ECMA-262):
  http://www.ecma-international.org/publications/standards/Ecma-262-arch.htm

- W3C's DOM recommendations:
  http://www.w3.org/DOM/DOMTR