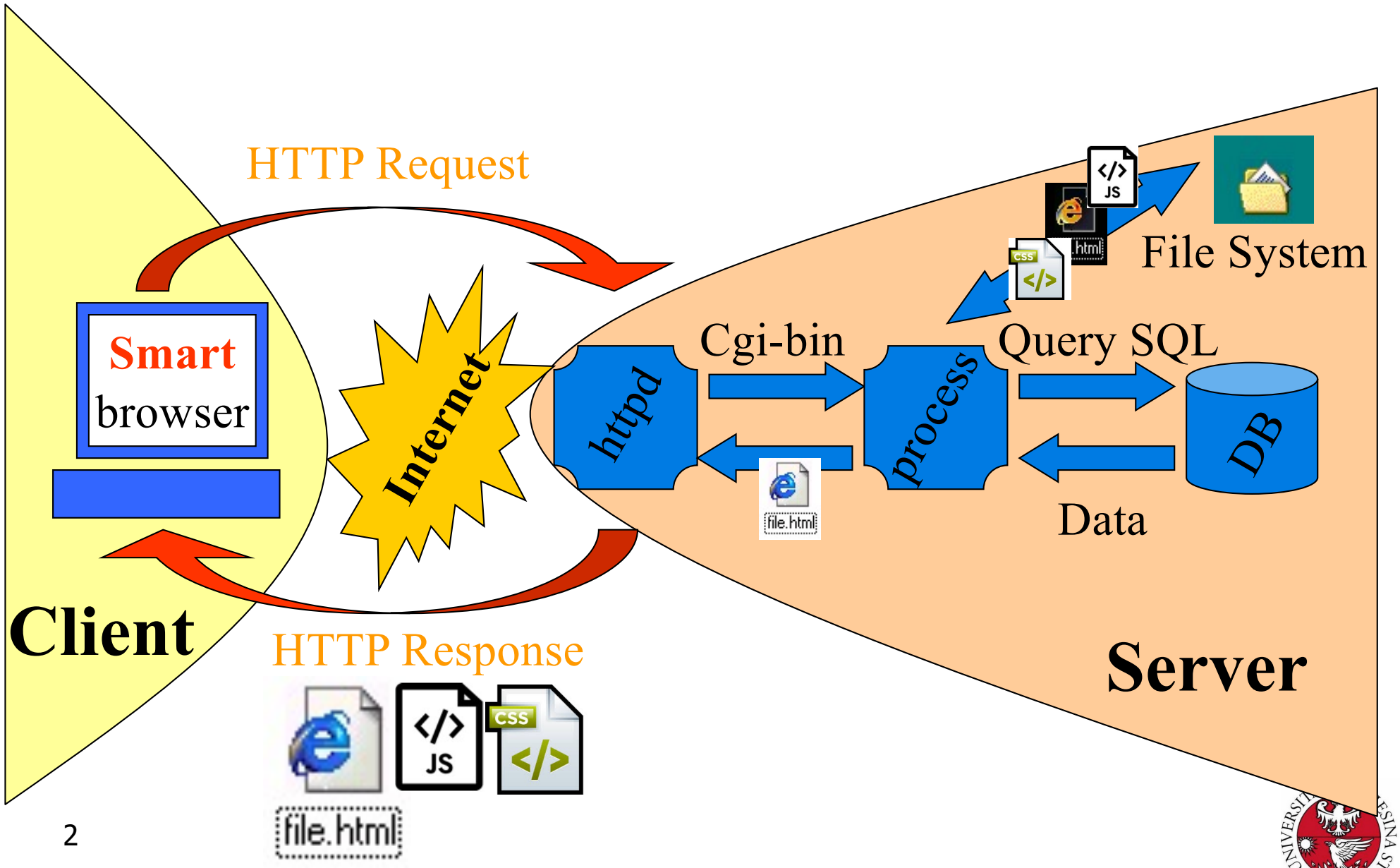
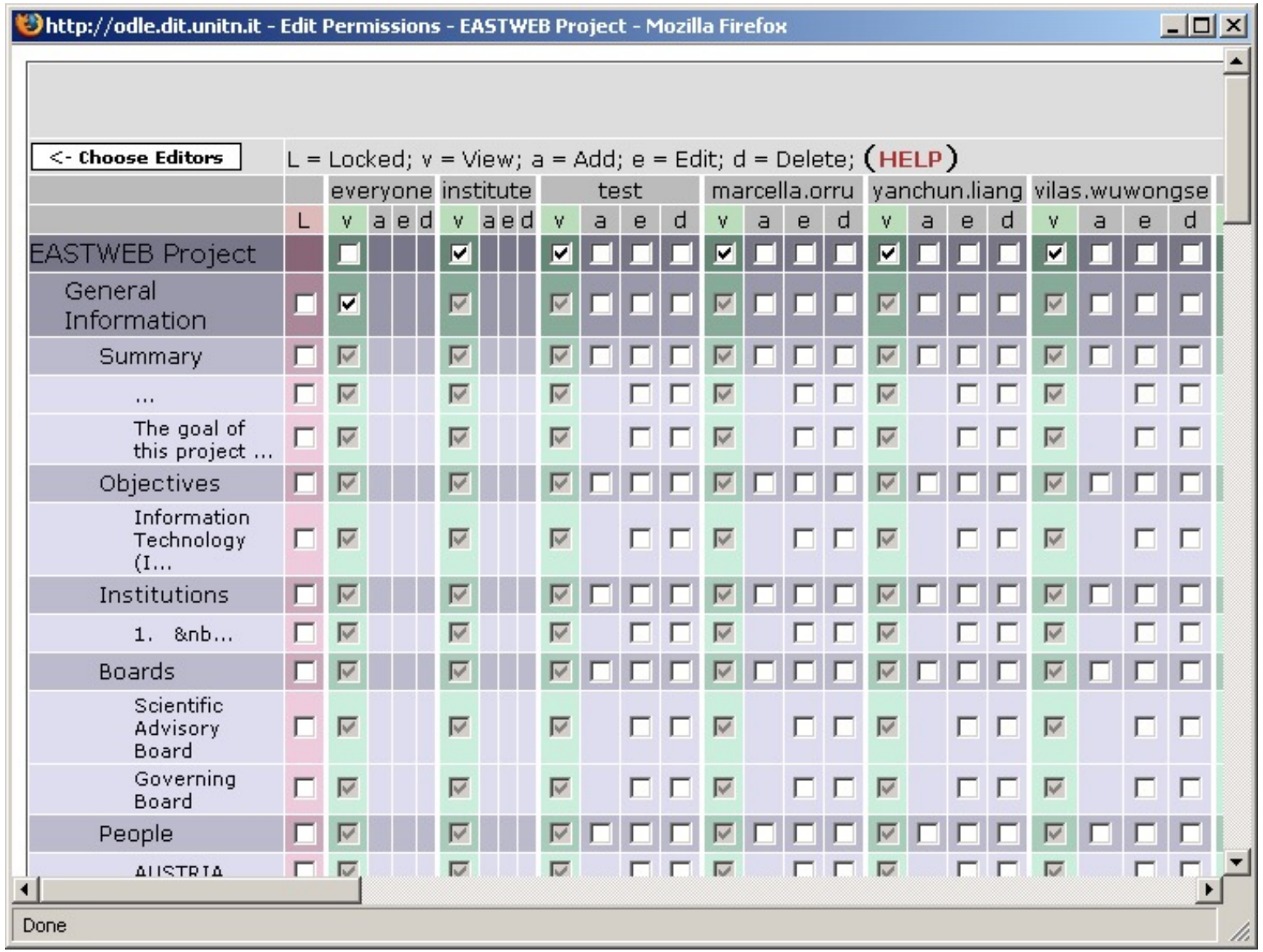
Q

How can we avoid unresponsiveness of pages ?


HTTP Request
Smart
browser
Internet
httpd
Cgi-bin
process
Query SQL
DB
File System
Data
file.html
Client
HTTP Response
Server
JS
CSS
file.html

The form nightmare…



Ajax !

- not a technology in itself: it is a term coined in 2005 by Jesse James Garrett: “Asynchronous JavaScript + XML”.
- new development technique
- blur the line between web-based and desktop applications.
- rich, highly responsive and interactive interfaces

Ajax was born as:

- dynamic presentation based on XHTML + CSS;
- dynamic display and interaction using Document Object Model;
- data exchange and manipulation using XML e XSLT;
- asynchrounous data fetching using XMLHttpRequest;
- JavaScript as glue.

How does Ajax work?

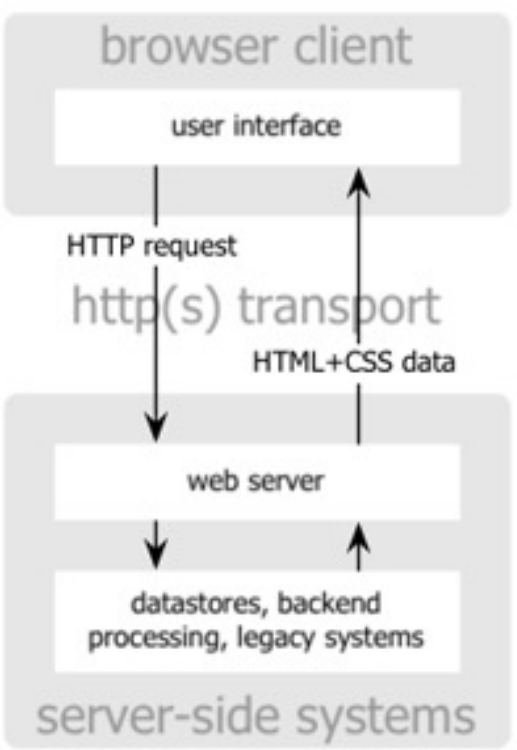
- The core idea behind AJAX is to make the communication with the server asynchronous, so that data is transferred and processed in the background.

- As a result the user can continue working on the other parts of the page without interruption.

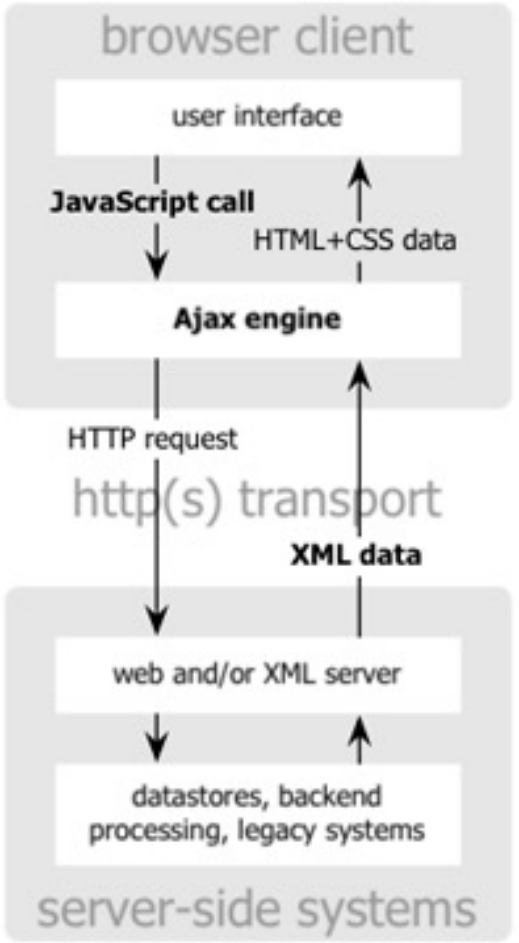
- In an AJAX-enabled application only the relevant page elements are updated, only when this is necessary.

The paradigms

1.0



classic
web application model

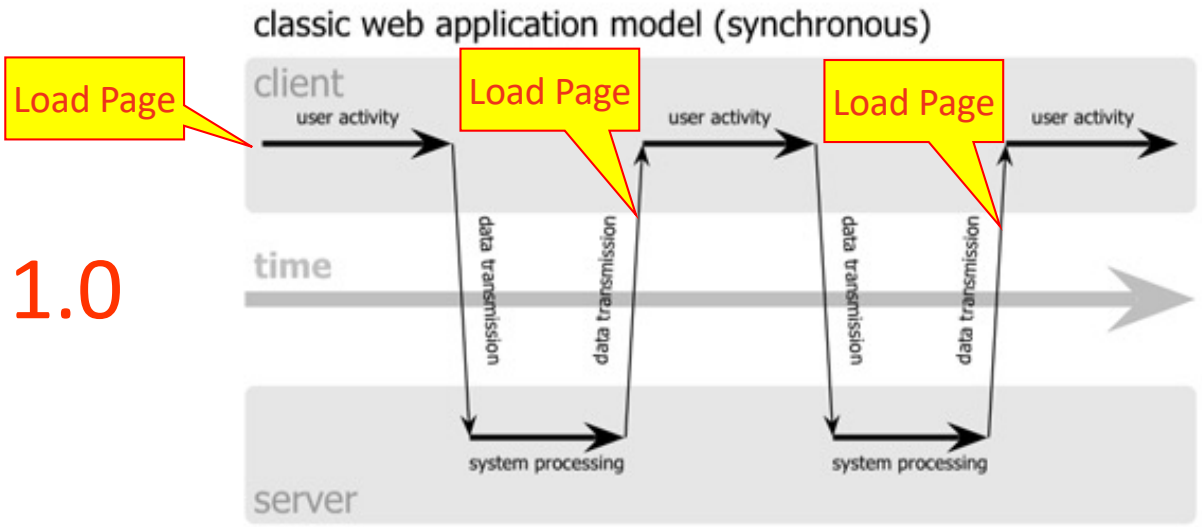


Ajax
web application model

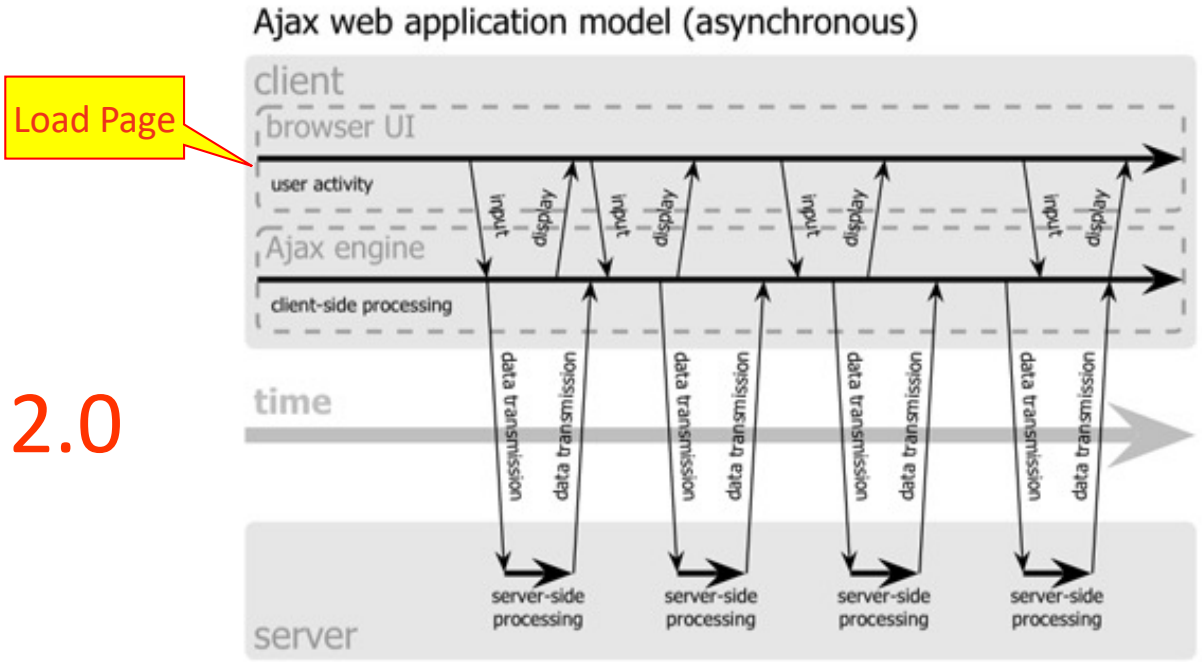
2.0

Pictures after
Jesse James Garrett

The models



1.0



2.0

Pictures after
Jesse James Garrett

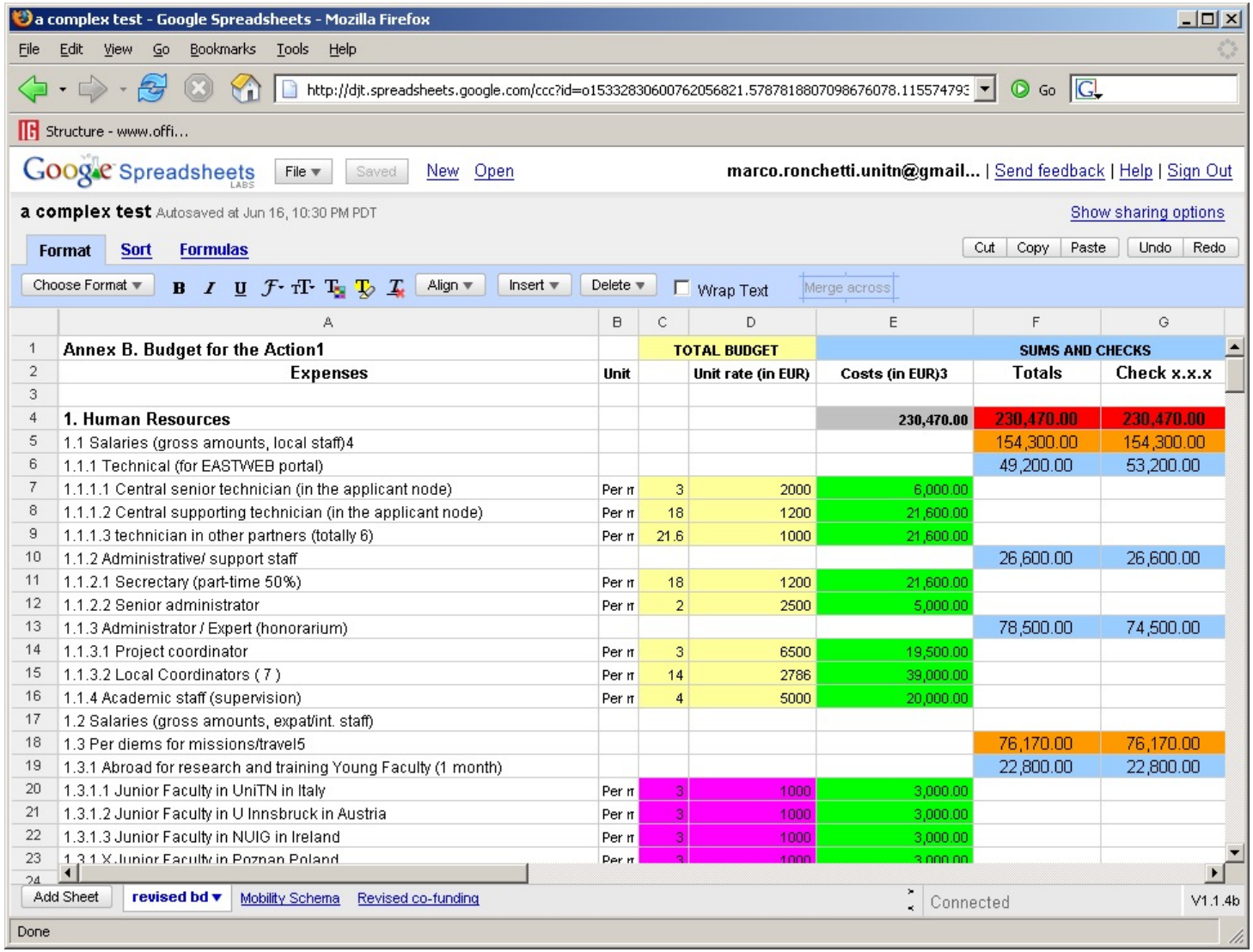
The heart and history of Ajax

- First used after Microsoft implemented Microsoft *XMLHTTP* COM object that was part of The Microsoft® XML Parser (IE 5.1)
- Similarly supported by a Mozilla Javascript object *XMLHttpRequest (Mozilla 1.0, Firefox, Safari 1.2 etc.)*
- Massively used by Google

```
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    http_request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
    http_request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Other labels for the same technology were Load on Demand, Asynchronous Requests, Callbacks, Out-of-band Calls, etc.

The (impressive!) result



a complex test - Google Spreadsheets - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://djt.spreadsheets.google.com/ccc?id=o15332830600762056821.5787818807098676078.115574793

Structure - www.offi...

Google Spreadsheets LABS File Saved New Open marco.ronchetti.unitn@gmail... | Send feedback | Help | Sign Out

a complex test Autosaved at Jun 16, 10:30 PM PDT Show sharing options

Format Sort Formulas Cut Copy Paste Undo Redo

Choose Format B I U Align Insert Delete Wrap Text Merge across

	A	B	C	D	E	F	G
1	**Annex B. Budget for the Action1**		**TOTAL BUDGET**			**SUMS AND CHECKS**	
2	**Expenses**	**Unit**		**Unit rate (in EUR)**	**Costs (in EUR)3**	**Totals**	**Check x.x.x**
3							
4	**1. Human Resources**				**230,470.00**	**230,470.00**	**230,470.00**
5	1.1 Salaries (gross amounts, local staff)4					154,300.00	154,300.00
6	1.1.1 Technical (for EASTWEB portal)					49,200.00	53,200.00
7	1.1.1.1 Central senior technician (in the applicant node)	Per m	3	2000	6,000.00		
8	1.1.1.2 Central supporting technician (in the applicant node)	Per m	18	1200	21,600.00		
9	1.1.1.3 technician in other partners (totally 6)	Per m	21.6	1000	21,600.00		
10	1.1.2 Administrative/ support staff					26,600.00	26,600.00
11	1.1.2.1 Secrectary (part-time 50%)	Per m	18	1200	21,600.00		
12	1.1.2.2 Senior administrator	Per m	2	2500	5,000.00		
13	1.1.3 Administrator / Expert (honorarium)					78,500.00	74,500.00
14	1.1.3.1 Project coordinator	Per m	3	6500	19,500.00		
15	1.1.3.2 Local Coordinators (7)	Per m	14	2786	39,000.00		
16	1.1.4 Academic staff (supervision)	Per m	4	5000	20,000.00		
17	1.2 Salaries (gross amounts, expat/int. staff)						
18	1.3 Per diems for missions/travel5					76,170.00	76,170.00
19	1.3.1 Abroad for research and training Young Faculty (1 month)					22,800.00	22,800.00
20	1.3.1.1 Junior Faculty in UniTN in Italy	Per m	3	1000	3,000.00		
21	1.3.1.2 Junior Faculty in U Innsbruck in Austria	Per m	3	1000	3,000.00		
22	1.3.1.3 Junior Faculty in NUIG in Ireland	Per m	3	1000	3,000.00		
23	1.3.1.X Junior Faculty in Poznan Poland	Per m	3	1000	3,000.00		
24							

Add Sheet revised bd Mobility Schema Revised co-funding Connected V1.1.4b

Done

Ajax - advantages

- Better Performance and Efficiency
  - small amount of data transferred from the server. Beneficial for data-intensive applications as well as for low-bandwidth networks.
- More Responsive Interfaces
  - the improved performance give the feeling that updates are happening instantly. AJAX web applications appear to behave much like their desktop counterparts.
- Reduced or Eliminated "Waiting" Time
  - only the relevant page elements are updates, with the rest of the page remaining unchanged. This decreases the idle waiting time.
- Increased Usability
  - Users can work with the rest of the page while data is being transferred in the background.

XMLHttpRequest (XHR)

see https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =
                this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

Asynchronous Programming!

XMLHttpRequest (XHR)

The **XMLHttpRequest.readyState** property returns the state an XMLHttpRequest client is in. An XHR client exists in one of the following states:

Value	State	Description
0	UNSENT	Client has been created. open() not called yet.
1	OPENED	open() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; responseText holds partial data.
4	DONE	The operation is complete.

XMLHttpRequest – Getting static resources

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =

            this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```


Getting dynamic resources with GET

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =
                this.responseText;
    }
  };
  xhttp.open("GET", "myservlet?param1=27", true);
  xhttp.send();
}
```

Note: if you want to avoid getting cashed results, add a fake parameter with the current time, e.g.

```
xhttp.open("GET", url + ((/\?/).test(url) ? "&" : "?") + (new Date()).getTime());
```

See https://www.w3schools.com/jsref/jsref_regexp_test.asp to understand the code above

Getting dynamic resources with POST

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
     document.getElementById("demo").innerHTML =
                this.responseText;
    }
  };
  xhttp.open("POST", "ajax_info.txt", true);
  xhttp.setRequestHeader("Content-type",
      "application/x-www-form-urlencoded");
  xhttp.send("nome=Dorothea&lname=Wierer");
}
```

1) add an HTTP header with setRequestHeader().
2) Specify the data you want to send in the send() method

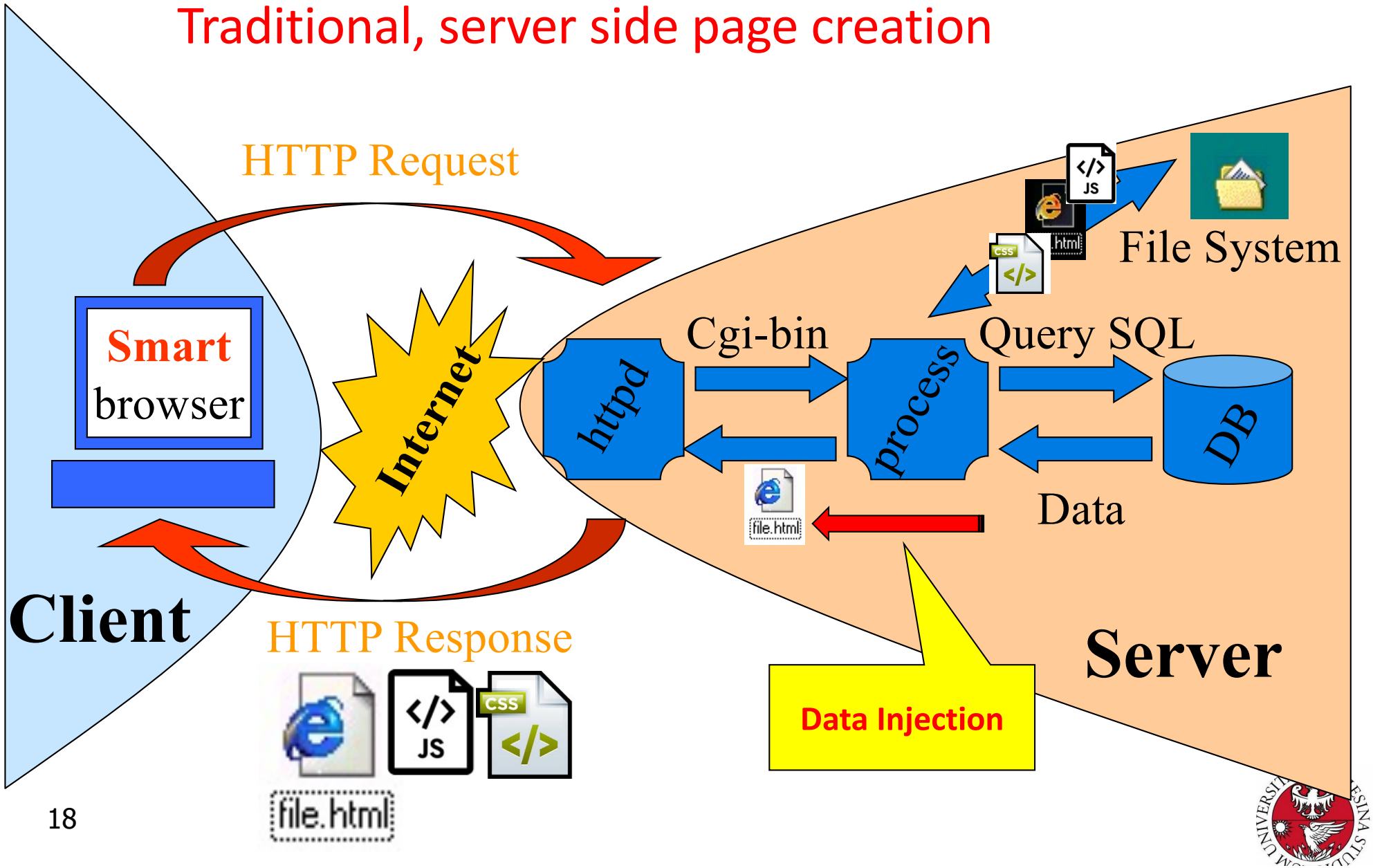
XMLHttpRequest methods

new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method, url, async, user, psw*)	Specifies the request *method*: the request type **GET** or **POST** *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

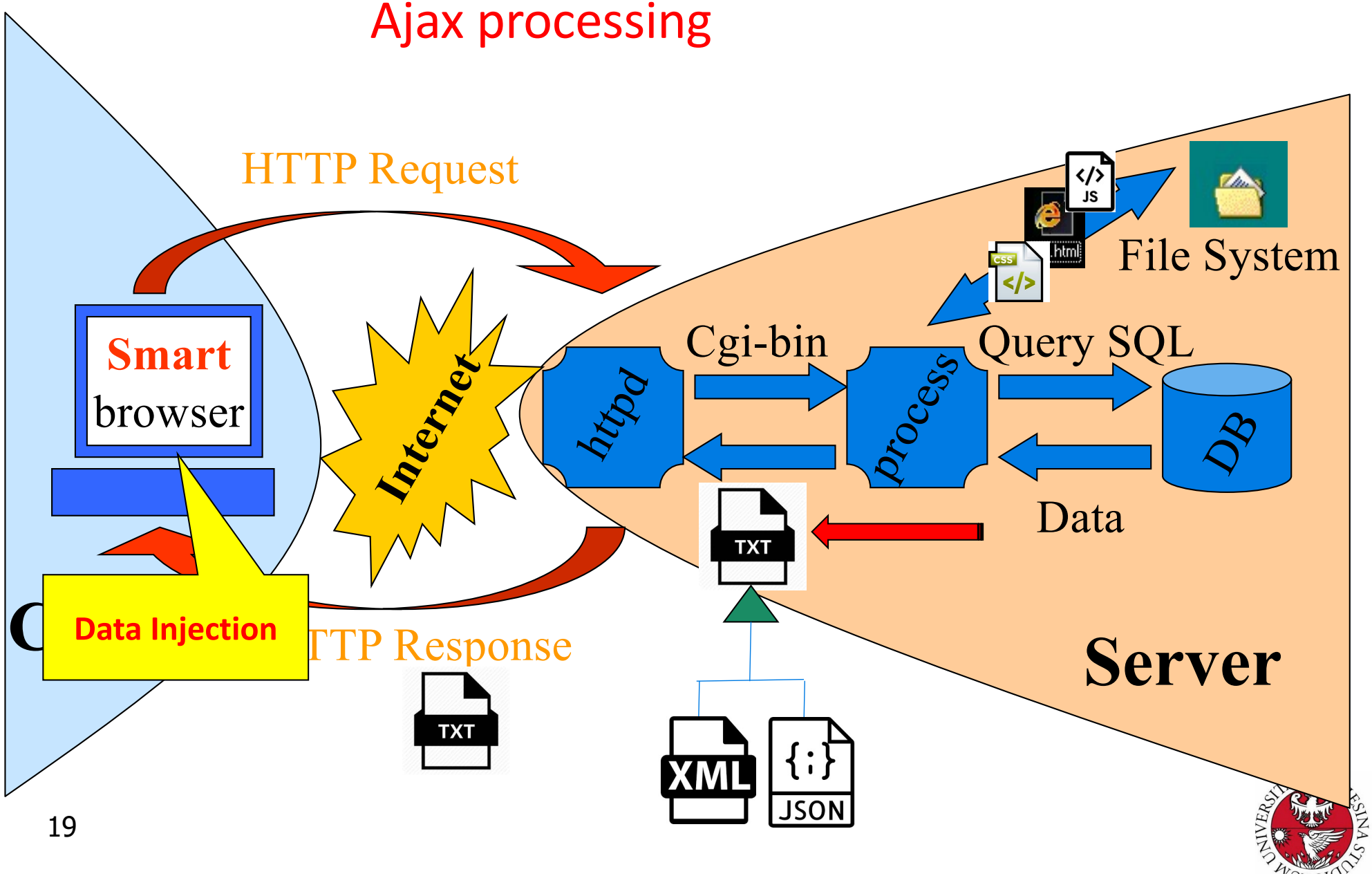
XMLHttpRequest properties

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the HTTP status-number of a request, e.g. 200: "OK" 403: "Forbidden" 404: "Not Found"
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Traditional, server side page creation



Ajax processing



And make sure that you…

- Preserve the Normal Page Lifecycle – as much as possible!
- Reflect Control State on the Server – in real-life scenarios there is no use of simply rendering controls on the page.
- Support Cross-Browser usage – there are different implementation of the XmlHttpRequest object. You should make sure that all AJAX components you choose operate properly on various browsers and platforms.
- Ensure proper Operation when Cookies are Disabled – support cookieless sessions.

And make sure that you…

- Give visual feedback - When a user clicks on something in the AJAX user interface, they need immediate visual feedback
- Keep the Back button – make sure that the Back button in your application functions on every page of the site.
- Use links for navigation – avoid the temptation to use links as an interface on your AJAX application to change the state of your application. Users have been trained over many years to expect a link to “take” them somewhere, so give them what they expect.
- Use human-readable links – people like to pass the addresses of useful web pages to each other. Make sure your application supports URLs that people can share easily, so not too long or complex.

Adapted from: www.telerik.com/documents/Telerik_and_AJAX.pdf

AJAX Tutorial and reference

https://www.w3schools.com/js/js_ajax_intro.asp

https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX

Q

What are the best ways to transfer data?

Two main forms of (structured) data transfer

XML

```
<employees>
 <employee>
  <firstName>John</firstName>
  <lastName>Doe</lastName>
 </employee>
 <employee>
  <firstName>Anna</firstName>
  <lastName>Smith</lastName>
 </employee>
 <employee>
  <firstName>Peter</firstName>
  <lastName>Jones</lastName>
 </employee>
</employees>
```

JSON

```
{"employees":[
  { "firstName":"John",
    "lastName":"Doe" },
  { "firstName":"Anna",
    "lastName":"Smith" },
  { "firstName":"Peter",
    "lastName":"Jones" }
]}
```

XML vs JSON

Both JSON and XML:

- are "self describing" (human readable)
- are hierarchical (values within values)
- can be parsed and used by lots of programming languages
- can be fetched with an XMLHttpRequest

For AJAX applications, JSON is faster and easier than XML:

XML

Fetch an XML document
Use the XML DOM to loop through the document
Extract values and store them in variables

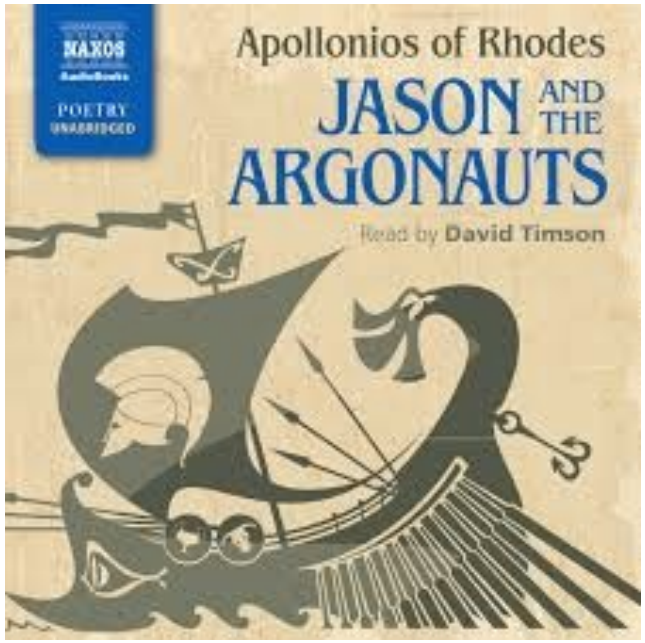
JSON

Fetch a JSON string
JSON.Parse the JSON string

JSON – JavaScript Object Notation

JSON is a language-independent data format.

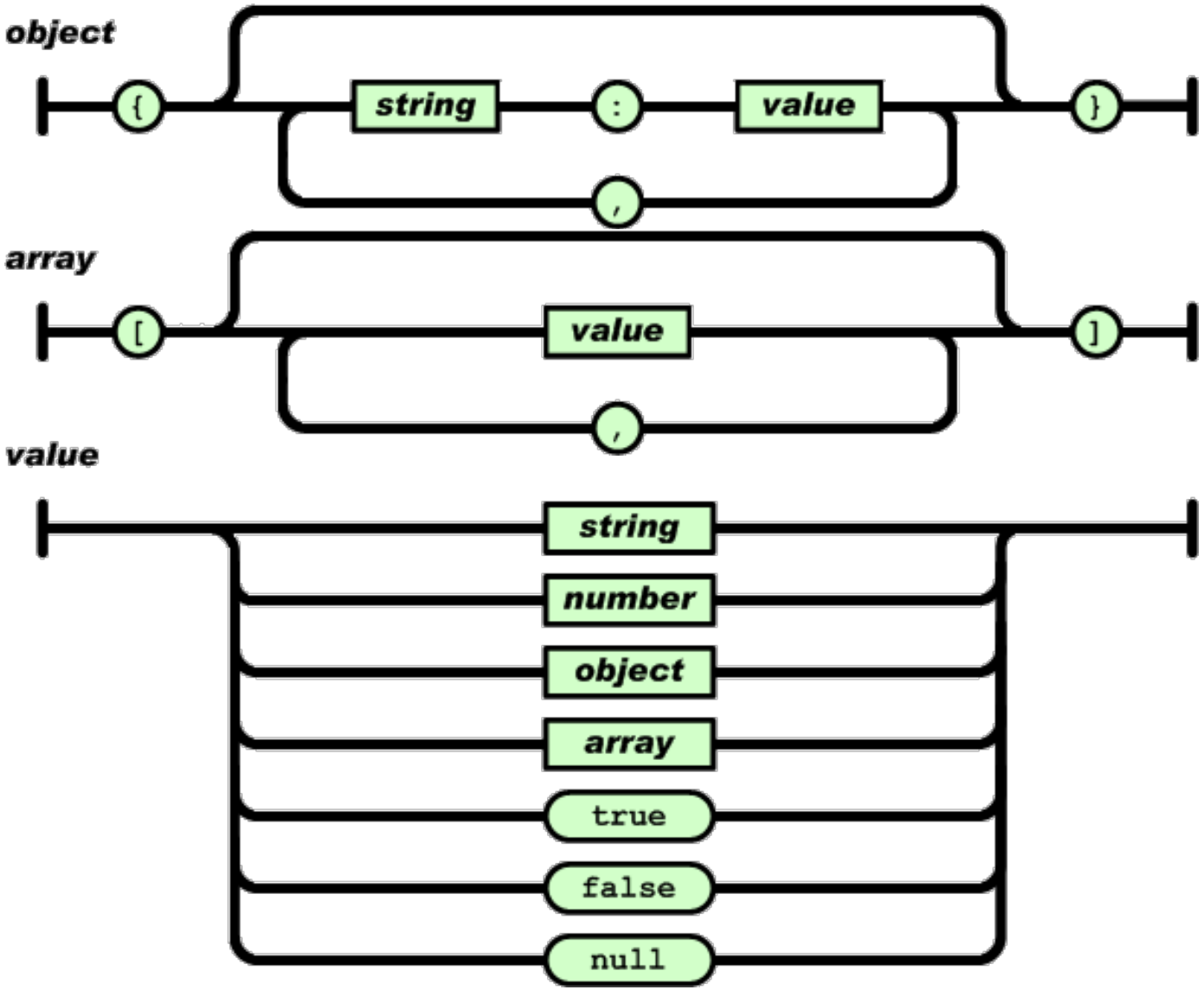
```
{ "name": "Mario",
        "surname": "Rossi",
        "active": true,
        "favoriteNumber": 42,
        "birthday": {
                "day": 1,
                "month": 1,
                "year": 2000
        },
        "languages": [ "it", "en" ]
}
```



Datatypes:

- int, float
- Boolean
- String
- Arrays []
- Associative Arrays {}
- null

JSON


object
{
string
:
value
}
,
array
[
value
]
,
value
string
number
object
array
true
false
null

Parsing JSON in JavaScript

```
var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';
var obj = JSON.parse(text);
obj.birth = new Date(obj.birth);

document.getElementById("demo").innerHTML = obj.name + ", " +
obj.birth;
```

Argo (Parsing JSON in Java)

```
{
   "name": "Black Lace",
   "sales": 110921,
   "totalRoyalties": 10223.82,
   "singles": [
      "Superman", "Agadoo"
   ]
}
```

```
String secondSingle = new JdomParser().parse(jsonText)
    .getStringValue("singles", 1);
```

http://argo.sourceforge.net/index.html

AJAJ

```
var my_JSON_object;
var url=" https://mdn.github.io/learning-
area/javascript/oojs/json/superheroes.json"
var xhttp = new XMLHttpRequest();
xhttp.open("GET", url, true);
xhttp.responseType = "json";
xhttp.onreadystatechange = function () {
  var done = 4, ok = 200;
  if (this.readyState === done && this.status === ok)
  {
    my_JSON_object = this.response;
  }
};
xhttp.send();}
```

An example

https://mdn.github.io/learning-area/javascript/oojs/json/superheroes.json


```
{
  "squadName" : "Super Hero Squad",
  "homeTown" : "Metro City",
  "formed" : 2016,
  "secretBase" : "Super tower",
  "active" : true,
  "members" : [
    {
      "name" : "Molecule Man",
      "age" : 29,
      "secretIdentity" : "Dan Jukes",
      "powers" : [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name" : "Madame Uppercut",
      "age" : 39,
      "secretIdentity" : "Jane Wilson",
      "powers" : [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name" : "Eternal Flame",
      "age" : 1000000,
      "secretIdentity" : "Unknown",
      "powers" : [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

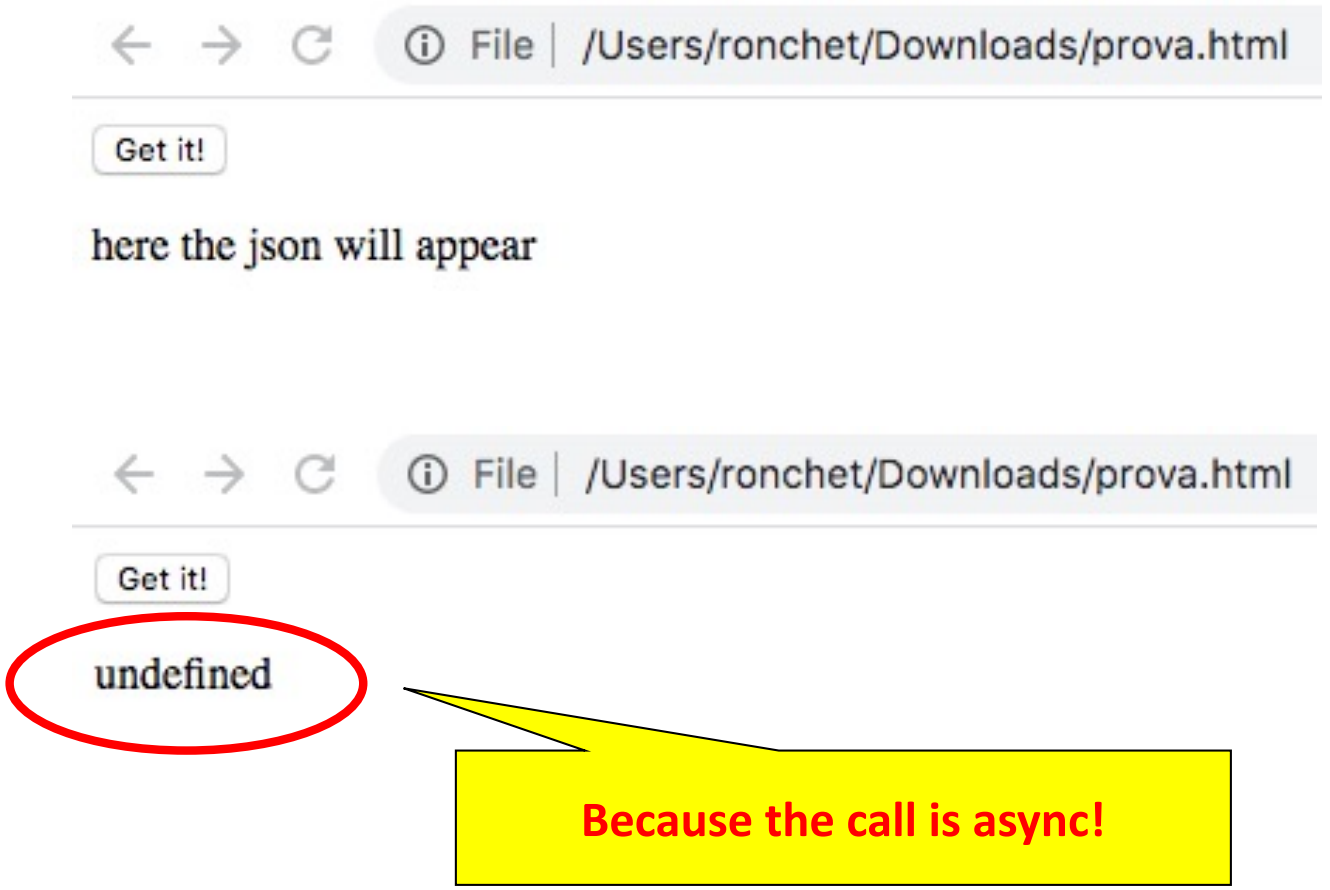
The traps of asynchronous computing 1A

```
<script>
  function getJson() {
    var my_JSON_object;
    var url="https://mdn.github.io/learning-
       area/javascript/oojs/json/superheroes.json"
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", url, true);
    xhttp.responseType = "json";
    xhttp.onreadystatechange = function () {
       var done = 4, ok = 200;
       if (this.readyState === done && this.status === ok){
          my_JSON_object = this.response;
       }
    };
    xhttp.send();
    return my_JSON_object;
  }
</script>
```


The traps of asynchronous computing 1B

```
<!DOCTYPE html>
<head>
    <title>AJAJ Demo</title>
    <script>…</script>
</head>
  <body>
    <form>
     <input type="BUTTON" onClick=
       'document.getElementById("myPar").innerHTML=getJson();'>
    </form>
    <p id="myPar">here the json will appear</p>
  </body>
</html>
```

The traps of asynchronous computing 1 out



The traps of asynchronous computing 2A

```
<script>
  function getJson() {
    var my_JSON_object;
    var url="https://mdn.github.io/learning-
       area/javascript/oojs/json/superheroes.json"
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", url, false);
    xhttp.responseType = "json";
    xhttp.onreadystatechange = function () {
       var done = 4, ok = 200;
       if (this.readyState === done && this.status === ok){
          my_JSON_object = this.response;
       }
    };
    xhttp.send();
    return my_JSON_object;
  }
</script>
```

Let us make it sync

The traps of asynchronous computing 2 out a

The traps of asynchronous computing 2 out b

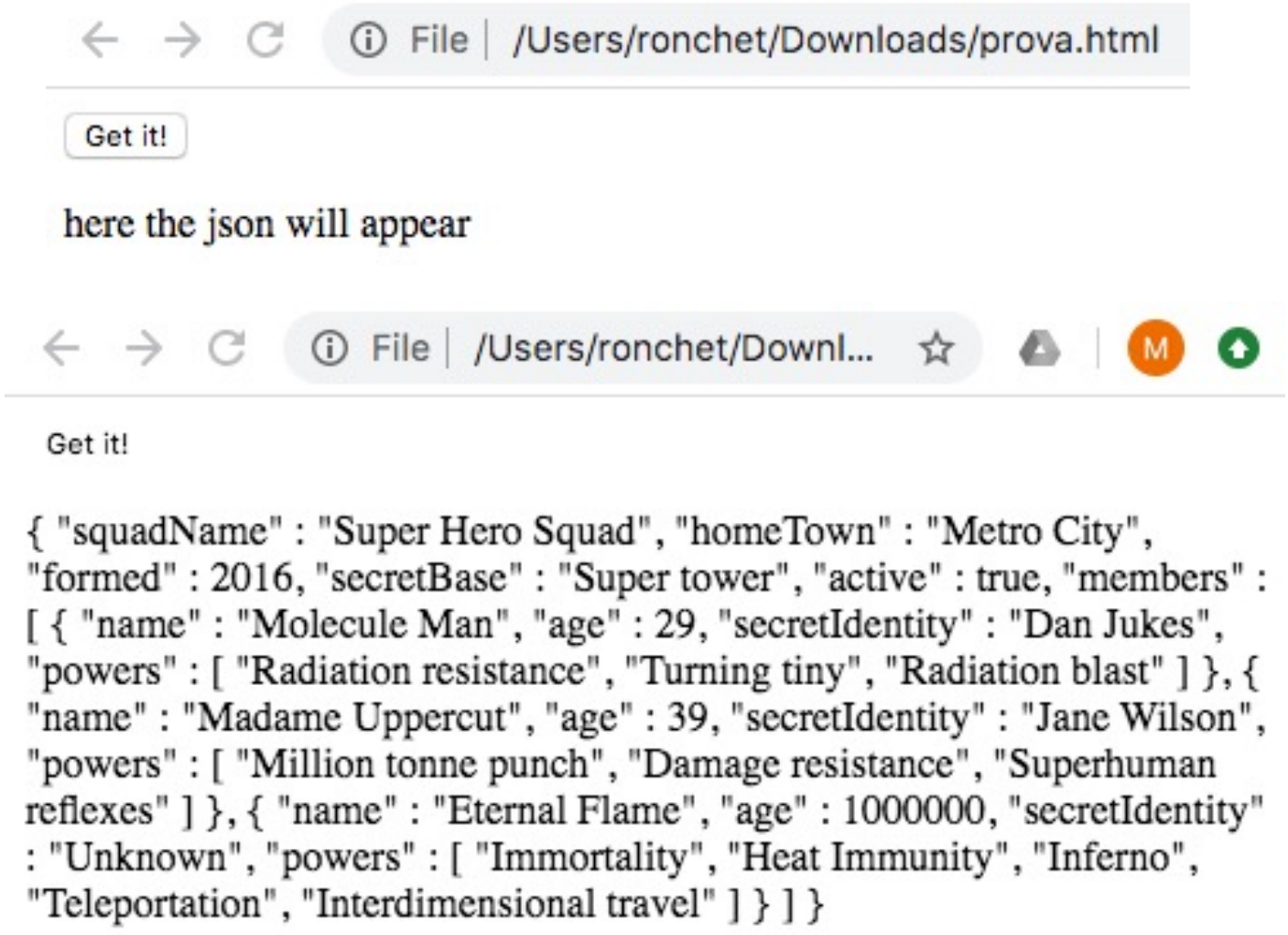

```
2 ▶Uncaught DOMException: Failed to set        prova.html:10
the 'responseType' property on 'XMLHttpRequest': The
response type cannot be changed for synchronous
requests made from a document.
    at getJson (file:///Users/ronchet/Downloads/prova.h
tml:10:28)
    at HTMLInputElement.onclick (file:///Users/ronchet/
Downloads/prova.html:27:112)
```

The traps of asynchronous computing 3A

```
<script>
  function getJson() {
    var my_JSON_object;
    var url="https://mdn.github.io/learning-
       area/javascript/oojs/json/superheroes.json"
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", url, false);
    //xhttp.responseType = "json";
    xhttp.onreadystatechange = function () {
       var done = 4, ok = 200;
       if (this.readyState === done && this.status === ok){
          my_JSON_object = this.response;
       }
    };
    xhttp.send();
    return my_JSON_object;
  }
</script>
```

Let's comment this line

The traps of asynchronous computing 3 out



The traps of asynchronous computing 4A

```
<script>
  function getJson() {
    var my_JSON_object;
    var url="https://mdn.github.io/learning-
       area/javascript/oojs/json/superheroes.json"
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", url, true);
    xhttp.responseType = "json";
    xhttp.onreadystatechange = function () {
       var done = 4, ok = 200;
       if (this.readyState === done && this.status === ok){
          my_JSON_object = this.response;
          document.getElementById("myPar".innerHTML=
             my_JSON_object.squadName;
       }
    };
    xhttp.send();
    return my_JSON_object;
  }
</script>
```

Let's make it async again

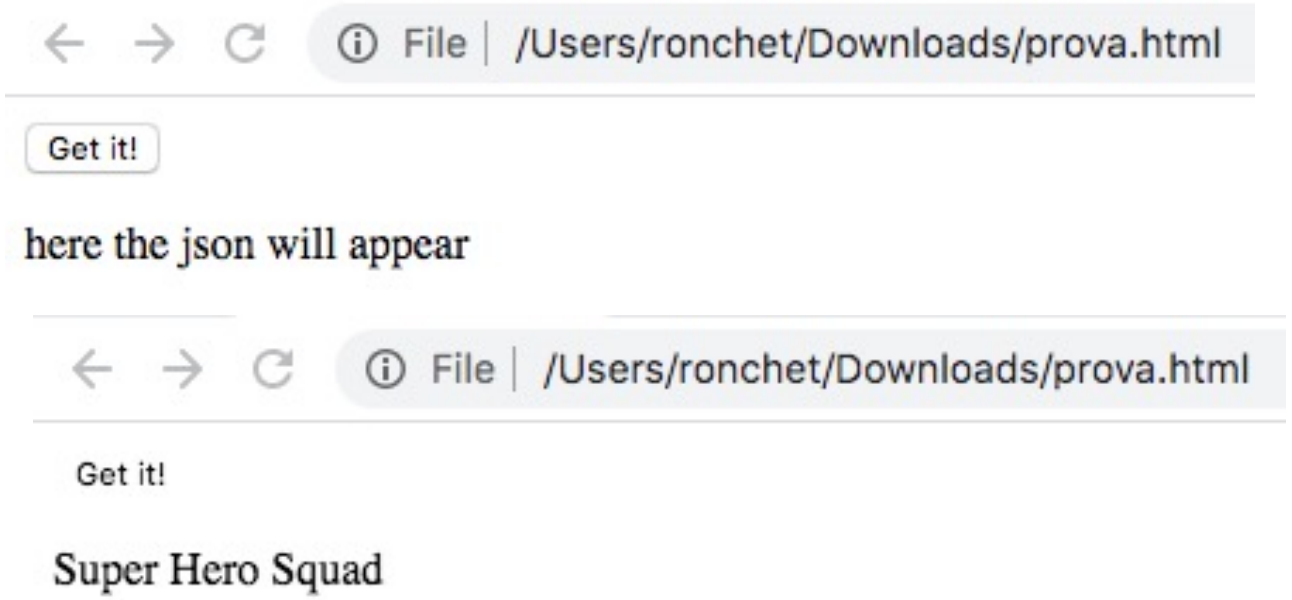
This is the right way of doing things!

The traps of asynchronous computing 4B

```
<!DOCTYPE html>
<head>
    <title>AJAJ Demo</title>
    <script>…</script>
</head>
  <body>
    <form>
     <input type="BUTTON" onClick=
       'innerHTML=getJson();'>
    </form>
    <p id="myPar">here the json will appear</p>
  </body>
</html>
```

This is the right way of doing things!

The traps of asynchronous computing 4 out



The traps of asynchronous computing 5A

```
<script>
  function getJson() {
    var my_JSON_object;
    var url="https://mdn.github.io/learning-
       area/javascript/oojs/json/superheroes.json"
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", url, true);
    //xhttp.responseType = "json";
    xhttp.onreadystatechange = function () {
       var done = 4, ok = 200;
       if (this.readyState === done && this.status === ok){
          my_JSON_object = this.response;
          document.getElementById("myPar".innerHTML=
             my_JSON_object.squadName;

       }
    };
    xhttp.send();
    return my_JSON_object;
  }
</script>
```

Let's comment this line

The traps of asynchronous computing 5 out

Json Tutorial and reference

https://www.w3schools.com/js/js_json_intro.asp

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

Q

What is the most modern way to write AJAX (AJAJ) queries?

JS Promises

A **Promise** is a placeholder (proxy) for a value not known when the promise is created.

It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.

In this way an asynchronous method returns a value like synchronous methods: it returns a *promise* to supply the value instead of returning the final value.

A very good tutorial: https://javascript.info/promise-basics

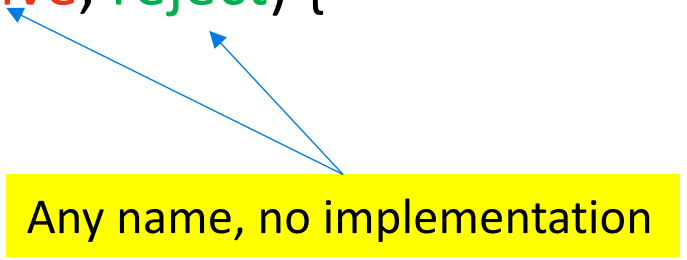
JS Promises

- A Promise is in one of these states:
- *pending*: initial state (neither fulfilled nor rejected).
- *fulfilled*: the operation was completed successfully.
- *rejected*: the operation failed.

JS Promises

Creating the Promise and evaluating it

```
let promise = new Promise(function(resolve, reject) {
    … some code…
    if (ok) {
        resolve(valueToBeReturned);
    } else {
        reject(new Error("message);
}
```



JS Request

```
promise.then(
  function(value) {doSomethingWithValue(value);}, //if fulfilled
  function(error) {doSomethingWithError(error);}  //if rejected
);
```


JS Promises: example

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Promise</h2>
<p id="demo"></p>

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}
let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;
// some code (try to change x to 5)
  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});
myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
</script>
</body>
</html>
```

shorter syntax

```
myPromise.then(
  myDisplayer,
  myDisplayer
);
```

try it here:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_promise2

JS Promises: example

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Promise</h2>
<p id="demo"></p>

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}
let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;
// some code (try to change x to 5)
  if (x == 0) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});
myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
</script>
</body>
</html>
```

```
  if (x == 0) {
    setTimeout(() => myResolve("OK"), 2000);
  } else {
    setTimeout(() => myReject("Error"), 1000);
  }
});
```

with delay

try it here:
https://www.w3schools.com/js/tryit.asp?filename=tryjs_promise2

JS fetch

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Fetch a file to change this text.</p>
<script>
let file = "fetch_info.txt"
fetch (file)
.then(x => x.text())
.then(y => document.getElementById("demo").innerHTML = y);
</script>
</body>
</html>
```

JS fetch

fetch takes a Request object (may have headers)

fetch returns a Promise that resolve to a Response object

Response has methods:

- text() returns a Promise that resolves to a string
- json() returns a Promise that resolves to a Json object
- blob() returns a Promise that resolves to a Blob object