

# JavaScript

## GUI

by:

Jan Sabbe &  
Wouter Groeneveld



1. The DOM Tree
2. Events
3. Concurrency model & Animations
4. Unit testing
5. AJAX

Today we will talk about JavaScript frameworks as tools to manipulate the **DOM**.

We will not talk about JavaScript as **language** – see other course!

# The DOM Tree



# HTML

```
<table>
```

```
<tbody>
```

```
<tr id="firstQuarter">
```

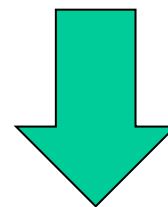
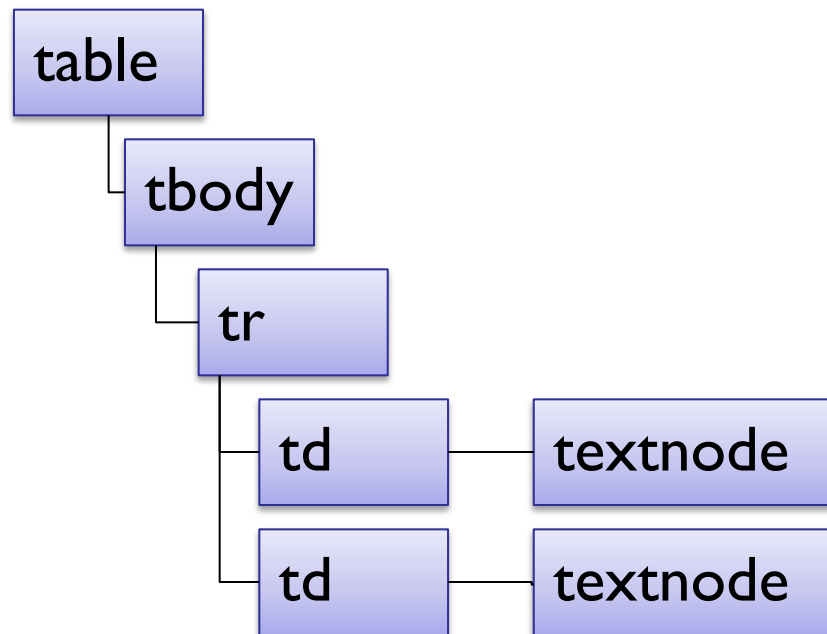
```
<td>Shady Grove</td>
```

```
<td>Aeolian</td>
```

```
</tr>
```

...

# DOM



The DOM is a tree you can manipulate with JavaScript.

# HTML

```
<table>
```

```
<tbody>
```

```
<tr id="firstQuarter">
```

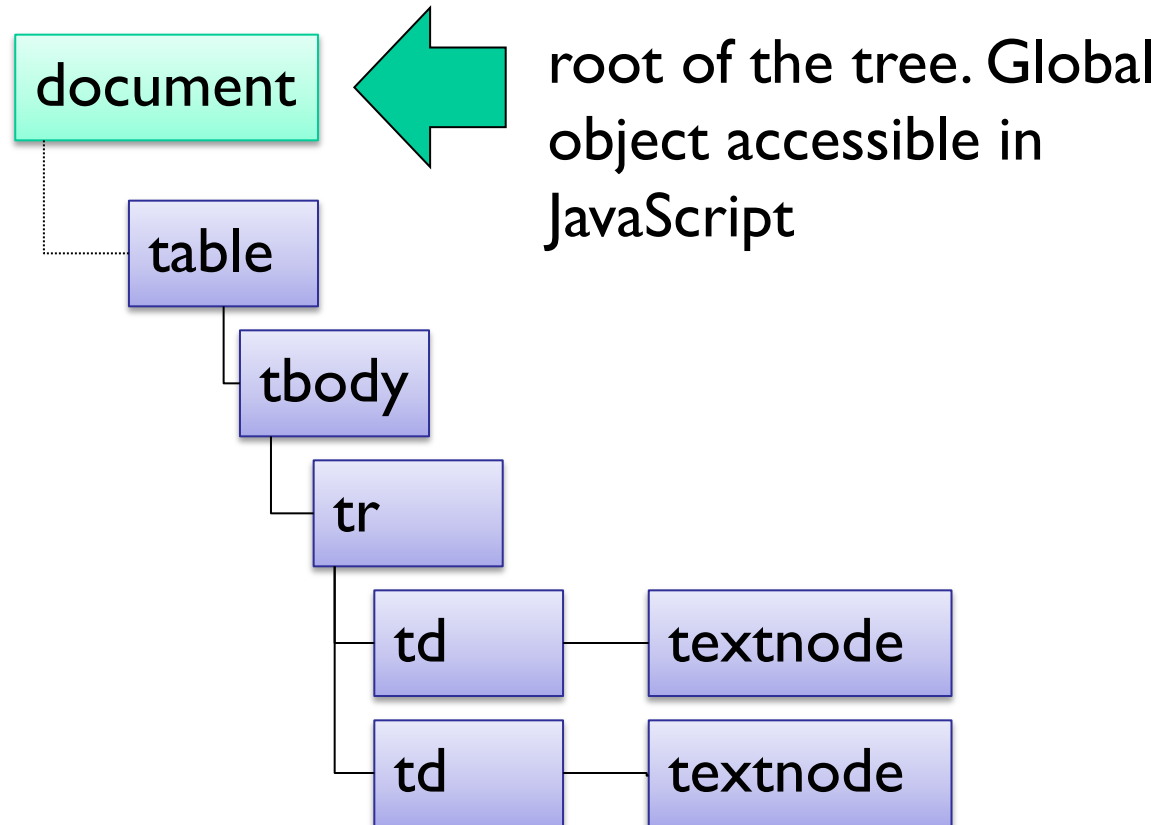
```
<td>Shady Grove</td>
```

```
<td>Aeolian</td>
```

```
</tr>
```

...

# DOM



td

Attributes:

- id
- children
- style
- ...

Methods:

- addEventListener
- appendChild
- ...

Each element in the DOM has attributes and methods to manipulate and query in JavaScript

# Problems with DOM

DOM methods/attributes are not easy to work with

DOM methods/attributes are not standard accros browsers



# Problems with DOM

DOM methods/attributes are  
not easy to work with

## **DOM:**

```
var newDiv = document.createElement("div");  
var newContent = document.createTextNode("Hi!");  
newDiv.appendChild(newContent);  
var my_div = document.getElementById("mainDiv");  
document.body.insertBefore(newDiv, my_div);
```

## **jQuery:**

```
jQuery( '<div>Hi!</div>' ).insertBefore( '# mainDiv' );
```

# Problems with DOM

DOM methods/attributes are  
not standard accros browsers

**IE:**

```
e1.attachEvent('onclick',function(){});
```

**Firefox:**

```
e1.addEventListener('click', function(){});
```

# Problems with DOM

Use a JavaScript framework that handles cross-browser issues and makes it easier to manipulate the DOM.



Most popular. We will use jQuery in this course.

**Warning:** jQuery and Prototype.js both provide a `$()` function.

Avoid collisions by calling `jQuery.noConflict()`

`$()` will be for Prototype.js

`jQuery()` will be for jQuery.

Use module pattern to still use `$` for jQuery in your code.

```
(function ($) {  
    //use $ here for jQuery code  
})(jQuery);
```

# jQuery introduction

<code>\$('#divWithId')</code>	Search for elements in DOM tree by using CSS selectors
<code>\$('&lt;div&gt;Hi!&lt;/div&gt;')</code>	Create a new element by typing literal HTML
<code>\$(domElement)</code>	Wraps a DOM element
<code>\$(function(){...})</code>	Calls function when DOM has been parsed and is ready for manipulation

## Search for elements in DOM tree by using standard CSS selectors

### basic

`$( '#header' )`

Search for element with id *header*

`$( '.warning' )`

Search for elements with class *warning*

`$( 'div' )`

Search for *div* elements

### can be combined

`$( '.warning li' )`

Search for *li* that are somewhere inside an element with class *warning*

`$( '.warning > li' )`

Search for *li* that are a direct child of an element with class *warning*

supports advanced CSS selectors

`$( '[href $= "png"]' )` Search for elements where href attribute ends with *png*

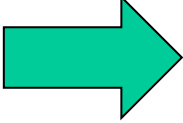
`$( '[href *= "www"]' )` Search for elements where href attribute contains *www*

`$( '[href ^= "http"]' )` Search for elements where href attribute starts with *http*

`$( 'tr:even' )` Search for all *even* rows

`$( 'input:checked' )` Search for all *inputs* that are checked (radiobutton, checkbox)

For more selectors see: <http://api.jquery.com/category/selectors/>

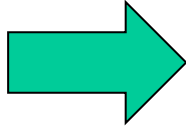
`$( '#myDiv' )`  Returns an object representing the search results.

Has methods and properties to query these results:

- `$( 'div' ).length`  
returns number of results
- `$( 'div' ).each(function(index ,element){...})`  
iterates over results, executes function for each element
- `$( 'div' ).find( 'li' )`  
find descendent *li* elements in set of *div* elements
- `$( '#checkbox' ).is( ':checked' )`  
check if one of the results matches given selector



`$( 'div' )`

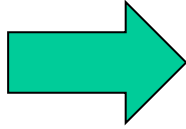


Returns an object representing the search results.

Has methods and properties to manipulate these results:

- `$( 'div' ).addClass( 'warning' )`  
adds *warning* class to all divs
- `$( 'div' ).removeClass( 'note' )`  
removes *note* class from all divs
- `$( 'ul' ).append( '<li>Hello</li>' )`  
appends new element as last child
- `$( 'span' ).css( 'font-size', '40px' )`  
set CSS property to given value

`$( 'div' )`



Returns an object representing the search results.

Even more interesting methods

- `$( 'div' ).html( '<span>Hello</span>' )`  
changes the html content inside the element
- `$( 'img' ).attr( 'src', 'http://cool.image.bro' )`  
changes the src attribute on all image elements
- `$( '#coolImage' ).attr( 'src' )`  
*returns* the value of the src attribute
- `$( '#warningMessage' ).html( )`  
*returns* the html content inside the element

# Lab 1

## Reference

<http://api.jquery.com/>

<https://developer.mozilla.org/en/JavaScript/Reference>

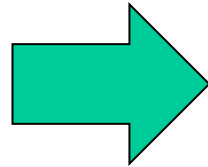
# Events

ul

li

a

li



When user interacts with element, an event is triggered.

- click
- hover
- change
- blur
- focus
- ...

ul

li

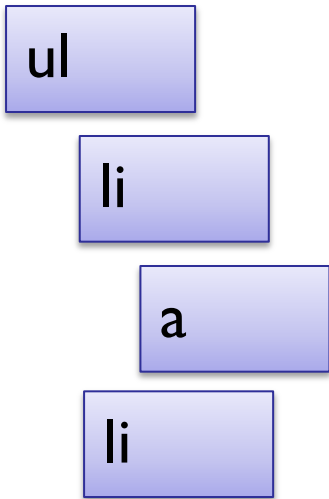
a

li

You can add an eventhandler for these types of events.  
The eventhandler will be executed when the event occurs.

```
<a onclick="console.log('clicked');">ok</a>
```

## Better way to add event handlers



### **HTML without JavaScript code**

```
<a id="linkje">ok</a>
```

### **Separate JavaScript file**

```
$( '#linkje' ).click(function (event) {  
    console.log( 'clicked!' );  
});
```

## **Specific methods**

```
$('#linkje').click(function (event) {});  
$('#linkje').hover(function (event) {});  
$('#input').change(function (event) {});  
$('#input').focus(function (event) {});  
$('#input').blur(function (event) {});
```

## **Generic method**

```
$('#linkje').on('click', function (event){});
```



## **Add multiple handlers on same element**

```
<input id="coolInput" type="text" />
```

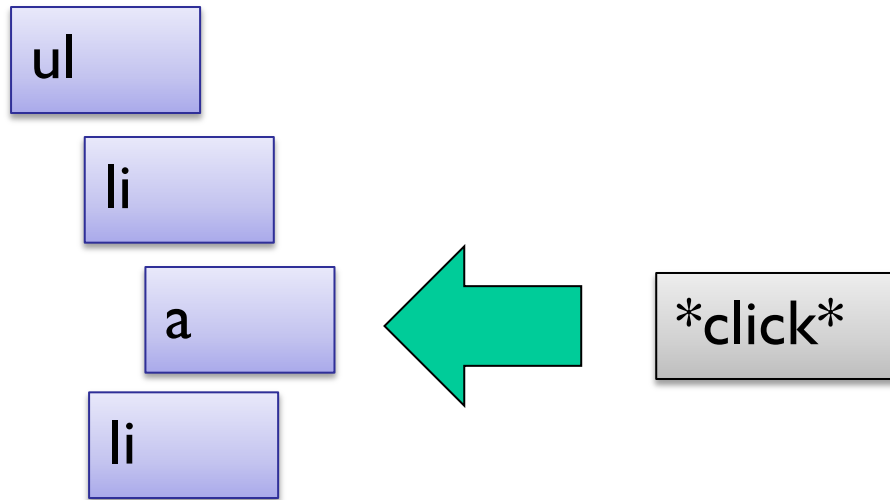
### Specific methods

```
$( '#coolInput' ).focus(function() {});  
$( '#coolInput' ).focusout(function() {});  
$( '#coolInput' ).keydown(function() {});
```

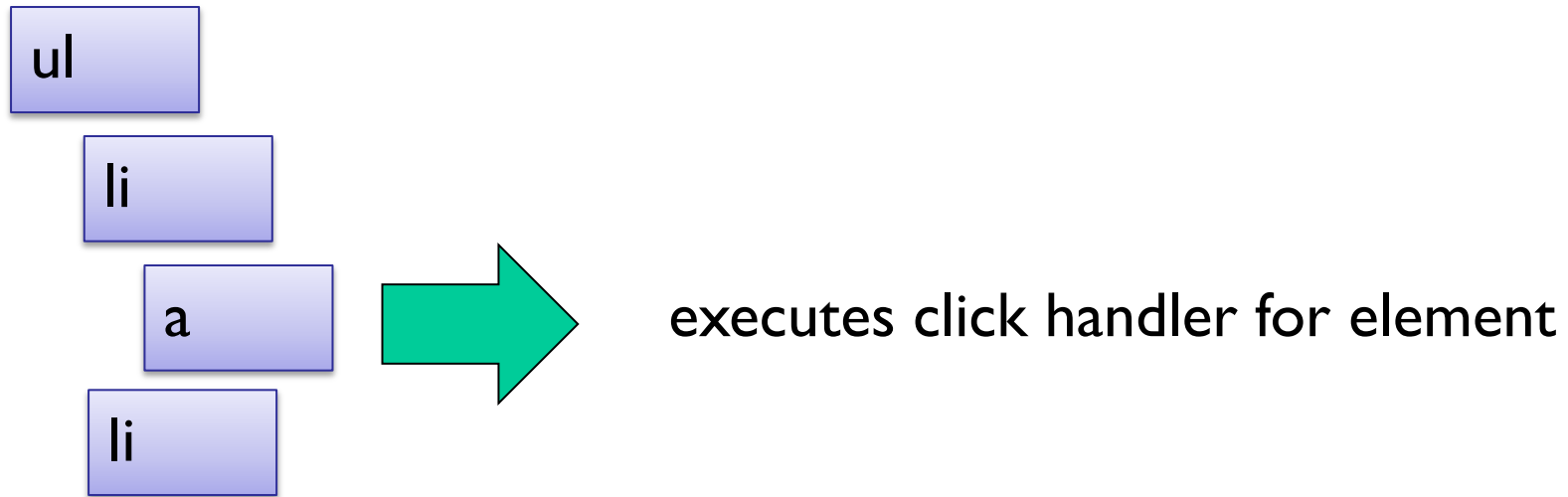
### Generic method

```
$( '#coolInput' ).on({  
    focus: function(){},  
    focusout: function(){},  
    keydown: function(){  
    }  
});
```

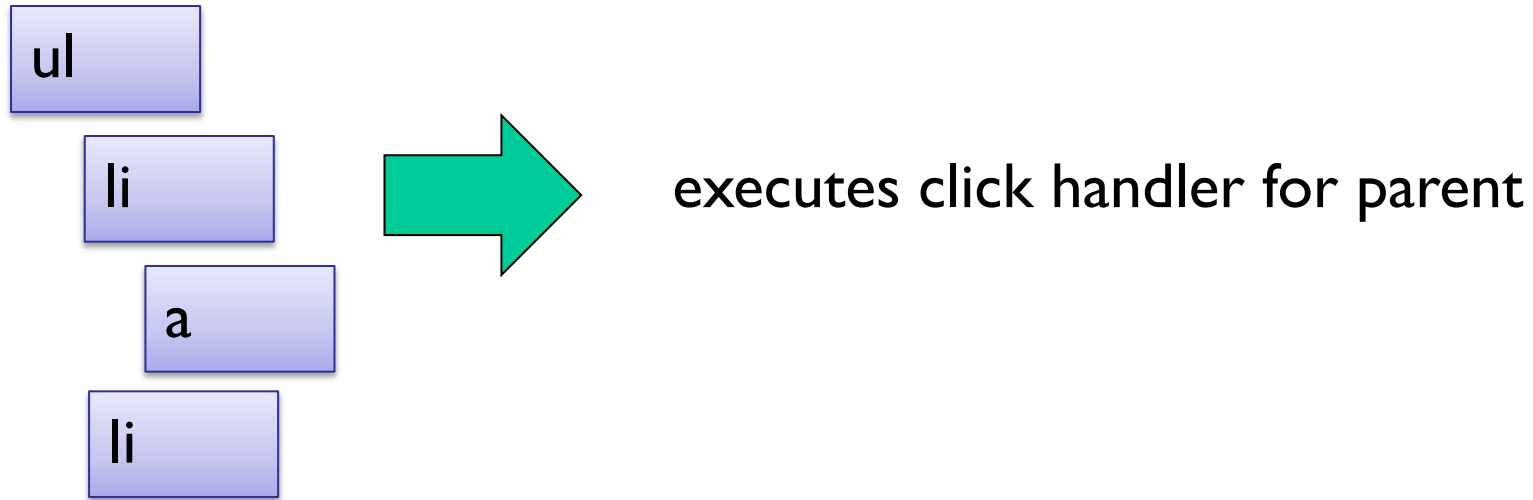
Events propagate up. Handlers of parent elements are executed as well.



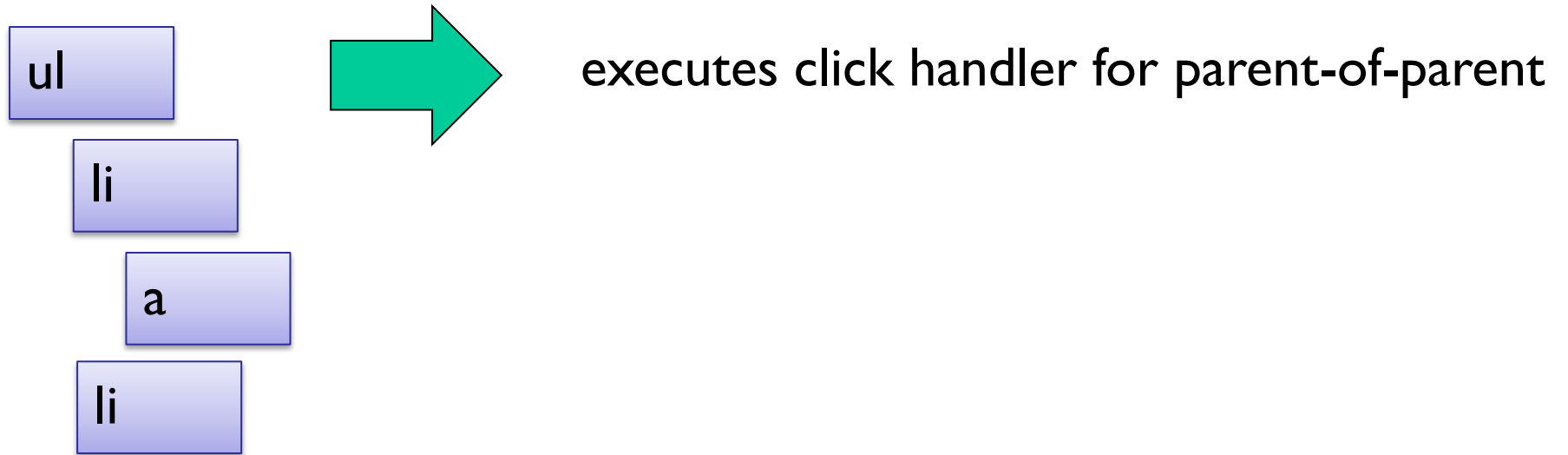
Events propagate up. Handlers of parent elements are executed as well.



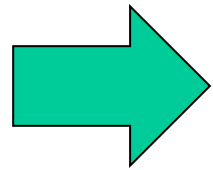
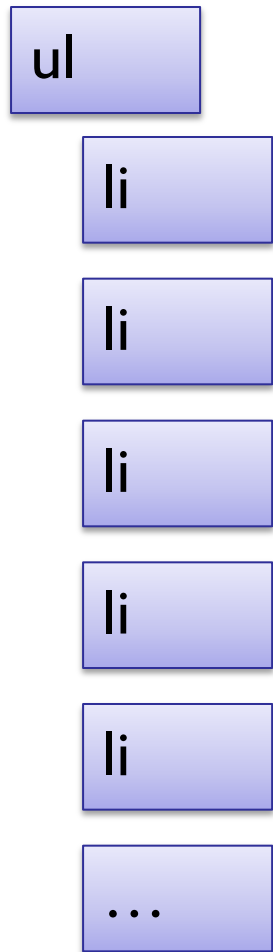
Events propagate up. Handlers of parent elements are executed as well.



Events propagate up. Handlers of parent elements are executed as well.



Why is this interesting?



instead of adding 100 clickhandlers on each *li* element

Why is this interesting?

ul

li

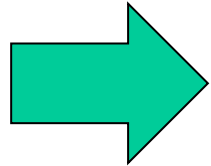
li

li

li

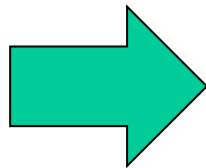
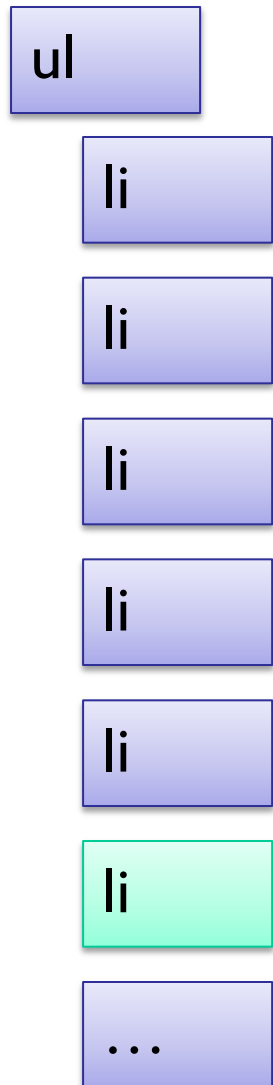
li

...



Add a single clickhandler on the parent

Why is this interesting?



When a new element is added, you don't need to manually add another click-handler



I want to handle clicks on red *li* elements

ul

li .red

li

li .red

li

li .red

li

li .red

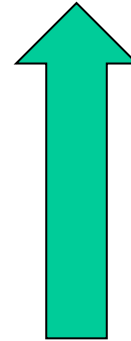
li

...

```
$( 'ul' ).on( 'click', 'li.red', function() {} );
```



Handler is added on *ul*



Handler will only be executed if  
clicked element matches selector

## Inside the handler

Event object contains extra information,  
and provides methods. Is optional



```
$( 'button' ).click(function(event) {  
    $(this).addClass( 'clicked' );  
});
```



this will match the DOM element that  
actually fired the event.

You need to wrap it with \$ function to use  
standard jQuery functions

## Inside the handler

```
event.stopPropagation();
```

Prevent an event from further propagating to the parents

```
event.preventDefault();
```

Prevents browser from executing default behaviour for event.

### **For example**

browser will by default follow link if you click on `<a href=..>`  
Using `preventDefault()` you can stop this from happening.

## Inside the handler

```
var ernie = {  
  name: "Ernie",  
  shout: function() {  
    console.log(this.name);  
  }  
}
```

```
$( 'button' ).click(ernie.shout);
```



Will not work.

- Why?
- How can we fix this?

## Special events

```
$(document).ready(function(){});  
$(function(){}); // shortcut
```

Function will be executed as soon as the DOM is ready to be manipulated. Images will not have been loaded yet

```
$(document).load(function(){});
```

Function will be executed when the DOM is ready **and** the browser has loaded everything. (Including images).

## Lab 2

### Reference

<http://api.jquery.com/category/events/>

## Programmatically triggering events

```
$( '#button' ).trigger( 'click' );
```

Manually trigger click handlers added to the element

This event will also propagate up to parents.

## Custom events

You can also bind to custom events

```
$(document).on('personDeleted', function() {  
    console.log('custom event caught');  
});
```

You need to trigger these custom events manually

```
$(document).trigger('personDeleted');
```



# Concurrency model

## Concurrency in JavaScript

JavaScript is single threaded. What happens if multiple events are fired at same time?



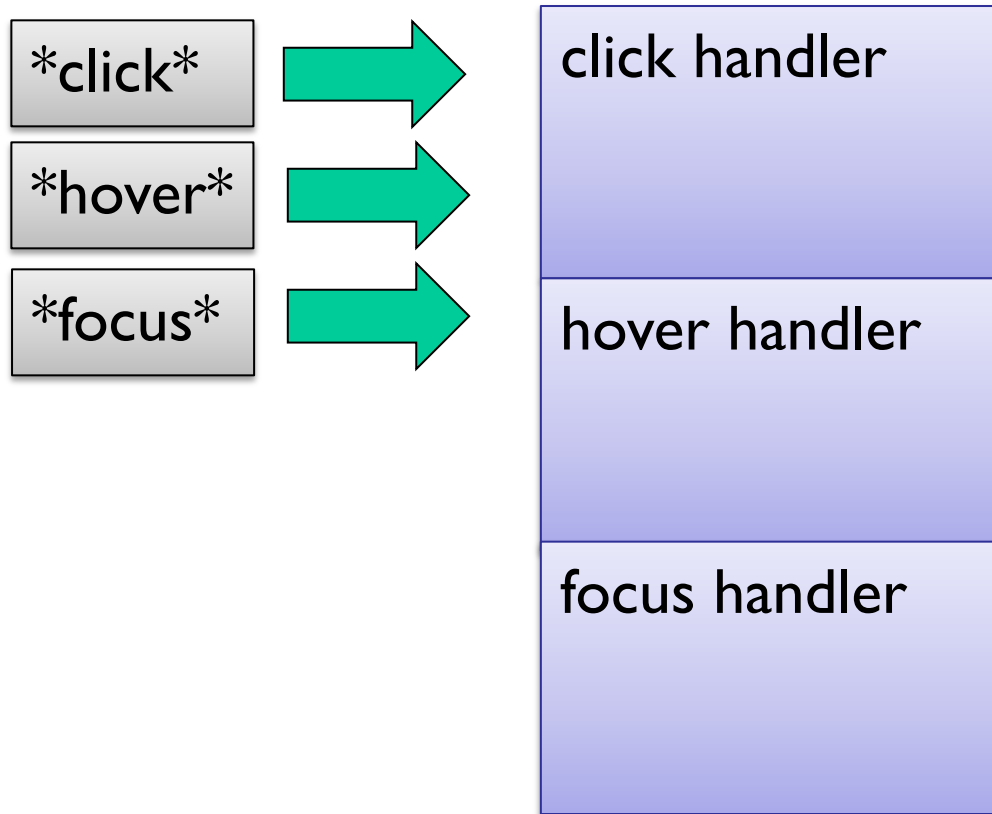
**\*click\***

**\*hover\***

**\*focus\***

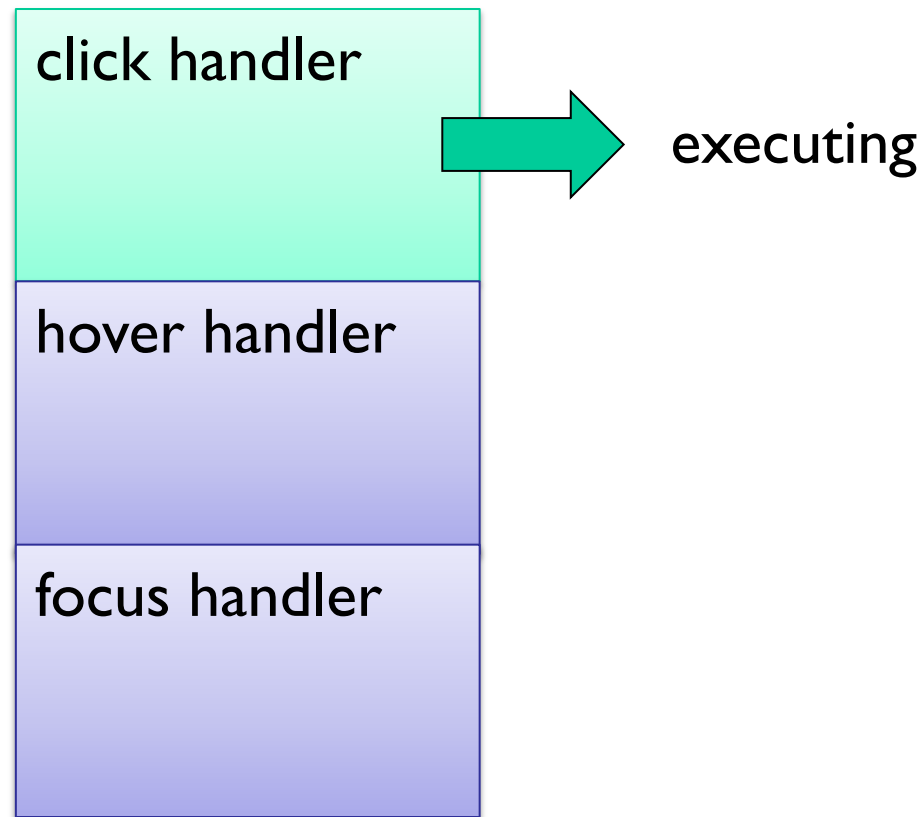
# Concurrency in JavaScript

The handlers that should be executed, are put on a queue



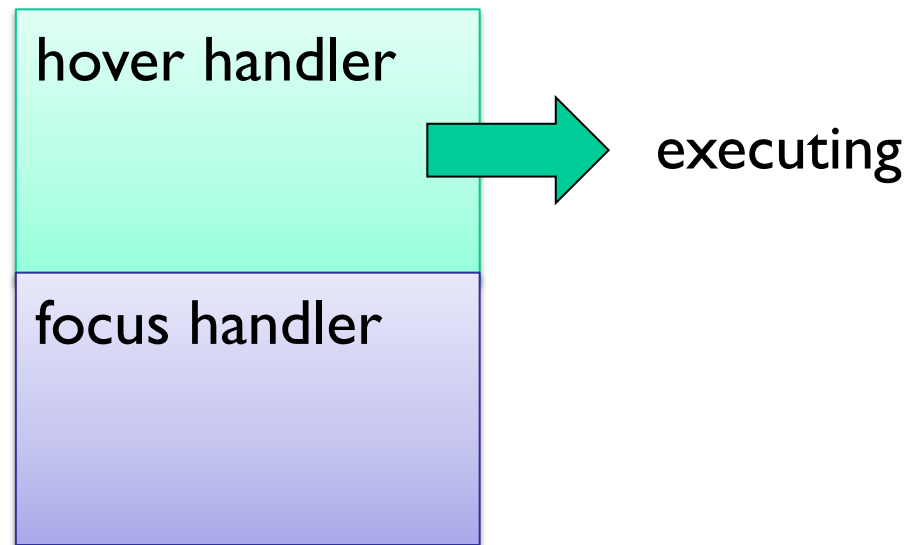
# Concurrency in JavaScript

These handlers are executed one by one



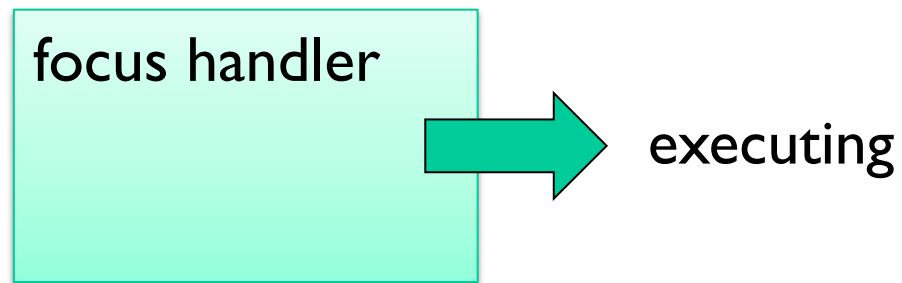
# Concurrency in JavaScript

These handlers are executed one by one



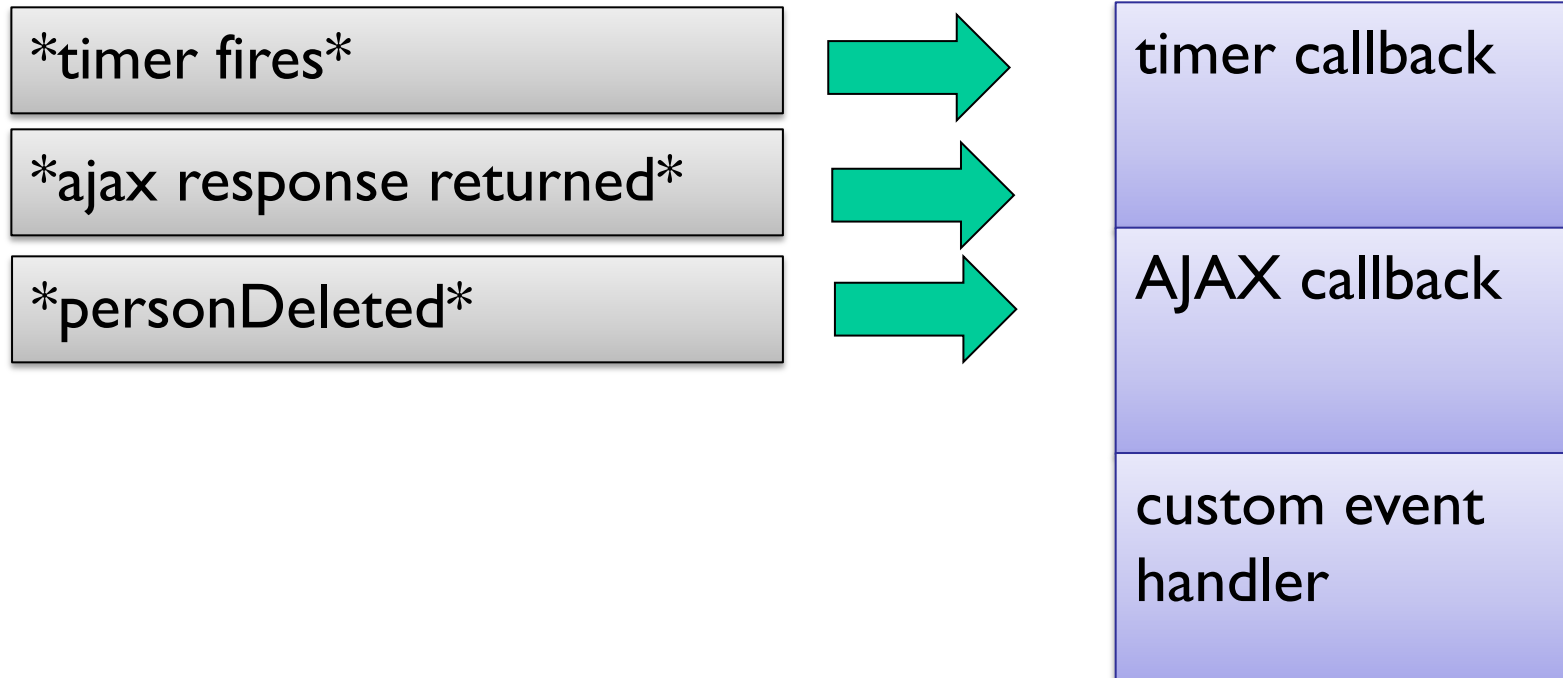
# Concurrency in JavaScript

These handlers are executed one by one



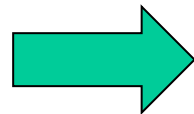
## Concurrency in JavaScript

This also applies to timers, AJAX responses, custom events...



## Concurrency in JavaScript

Make sure your event handlers and callbacks are fast. Otherwise the user interface won't feel responsive.



It will take some time before you will handle hover/focus events



Most functions in JavaScript and jQuery are asynchronous instead of blocking. This makes sure everything stays as responsive as possible.

### Instead of:

```
var response = sendBlockingNetworkRequest();  
alert(response);
```

### We do:

returns immediatly, before receiving response



```
sendAsyncNetworkRequest(function(response) {  
    alert(response);  
});
```



will be executed when we get response

# Animations

## **Animations with JavaScript**

Some easy jQuery functions provided

```
$( '#visible' ).fadeOut();  
$( '#hidden' ).fadeIn();
```

```
$( '#visible' ).slideUp();  
$( '#hidden' ).slideDown();
```

```
$( '#visible' ).hide();  
$( '#hidden' ).show();
```

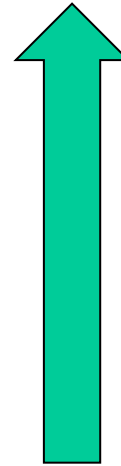
## Animations with JavaScript

Extra arguments

```
$('#visible').fadeOut(400, function(){});
```



Duration of animation in milliseconds.  
Can also be the string 'fast' or the string 'slow'



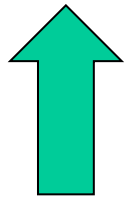
Callback is executed when animation has finished.

## Custom animations

```
$('#smallDiv').animate({  
  width: 400,  
  height: 400  
}, 500);
```



Target values. The width and height of the smallDiv will gradually increase to 400px



Duration of animation in milliseconds

## Multiple animations on same element are queued sequentially

```
$( '#smallDiv' )  
  .slideUp()  
  .slideDown()  
  .fadeOut()  
  .animate({ 'font-size':90 });
```

```
jQuery.fx.off = true;
```

Sets duration of all animations to 0.  
This is useful while writing unit-tests.

```
$( '#divThatIsCurrentlyAnimating' ).stop();
```

Stops animation that is still running.

# Lab 3

## Reference

<http://api.jquery.com/category/effects/>



# Unit testing

## Jasmine

```
describe('Person can shout', function() {  
  it('Person can shout loud', function() {  
    var person = PERSON.createPerson();  
    expect(person.shout()).toEqual('HELLO');  
  });  
});
```

Use `describe` to bundle a group of tests. (a suite of tests)

Use `it` to create a new test. Inside the test you can use `expect(actual).toEqual(expected)`.

## Matchers

`expect(..)`

`.toBe(..)`

Is object same (identity)

`.toEqual(..)`

Is object equal (compare properties)

`.toBeNull(..)`

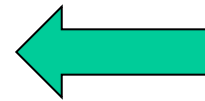
Is null

`.not.toBe(..)`

`.not.` negates matcher

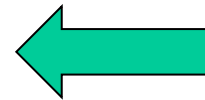
```
describe('Person can shout', function() {  
  var person;
```

```
  beforeEach(function() {  
    person = PERSON.createPerson();  
  });
```



run before each test

```
  afterEach(function() {  
    console.log('finished test');  
  });
```

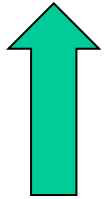


run after each test

```
  it('Person can shout loud', function(){  
    expect(person.shout()).toEqual('HELLO!');  
  });
```

```
});
```

```
describe('Person', function() {  
    describe('Person can shout', function(){..});  
    describe('Person can cry', function(){..});  
});
```



you can nest a suite in another suite

a beforeEach/afterEach in a suite, only applies  
to tests in that suite

## Custom matchers

You can write your own matchers

```
describe('People', function() {  
  beforeEach(function() {  
    this.addMatchers({ ← use this.addMatchers to  
      toBeDog: function(){ add custom matcher  
        return this.actual.isDog();  
      }  
    });  
  });  
});  
      ↑  
      use this.actual to get actual value
```

```
it('Best friend is a dog', function() {  
  expect(person.bestFriend).toBeDog();  
});  
});
```

## Spies

Replace a function on an object with a mocked function

```
it('dog bite will make person bleed', function() {  
    var dog = createDog(), person = createPerson();  
    spyOn(person, 'bleed');  
    dog.bite(person);  
    expect(person.bleed).toHaveBeenCalled();  
});
```

**Note:** In Jasmine you mock a function, you don't mock the entire object.

You can configure the mocked function:

```
spyOn(person, 'bleed').andReturn('lots of blood!');  
spyOn(person, 'bleed').andThrow('no more blood');
```

And you have some extra matchers:

```
expect(person.bleed).toHaveBeenCalled()  
expect(person.bleed).toHaveBeenCalledWith('lots')
```

<https://github.com/pivotal/jasmine/wiki/Spies>

After test ends, spies are replaced by original functions.



## jasmine-jquery

Extends jasmine with jQuery specific matchers

```
expect($(' #hiddenDiv' ).toBeHidden();
```

```
expect($(' input' ).toBeChecked();
```

```
expect($(' p' ).toHaveText('Who let the cow out');
```

```
expect($(' #messageDiv li' ).toHaveClass('warning');
```

...

<https://github.com/velesin/jasmine-jquery>

## jasmine-jquery

Use fixtures to load some test-html in your test

spec/javascripts/fixtures/test-welcome.html

```
<div id="welcome"></div><button id="#sayIt"/>
```

in jasmine test

```
it('clicking button changes text', function(){  
  loadFixtures('test-welcome.html');  
  
  $('#sayIt').trigger('click');  
  expect($('#welcome')).toHaveText('Hello');  
});
```

# Lab 4

## Reference

<https://github.com/pivotal/jasmine/wiki>

<https://github.com/velesin/jasmine-jquery>

## asynchronous tests

```
it('setTimeout calls function', function(){  
  var isCalled = false;  
  setTimeout(function(){isCalled =true;}, 1000);
```

`waits(1100);`  will run the next **runs** block after 1100 milliseconds

```
  runs(function() {  
    expect(isCalled).toBeTruthy();  
  });  
});
```

## asynchronous tests

```
it('setTimeout calls function', function(){  
  fetcher.fetch();
```

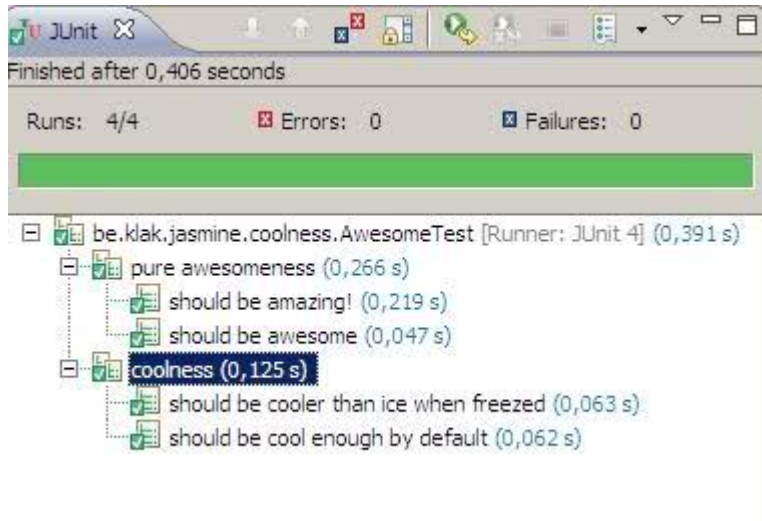
```
  waitFor(function(){return fetcher.isComplete()}));
```



will run the next **runs** block when given function returns true. You can pass a maximum timeout

```
    runs(function() {  
      expect(fetcher.fetchedData).toBe(...);  
    });  
  });
```

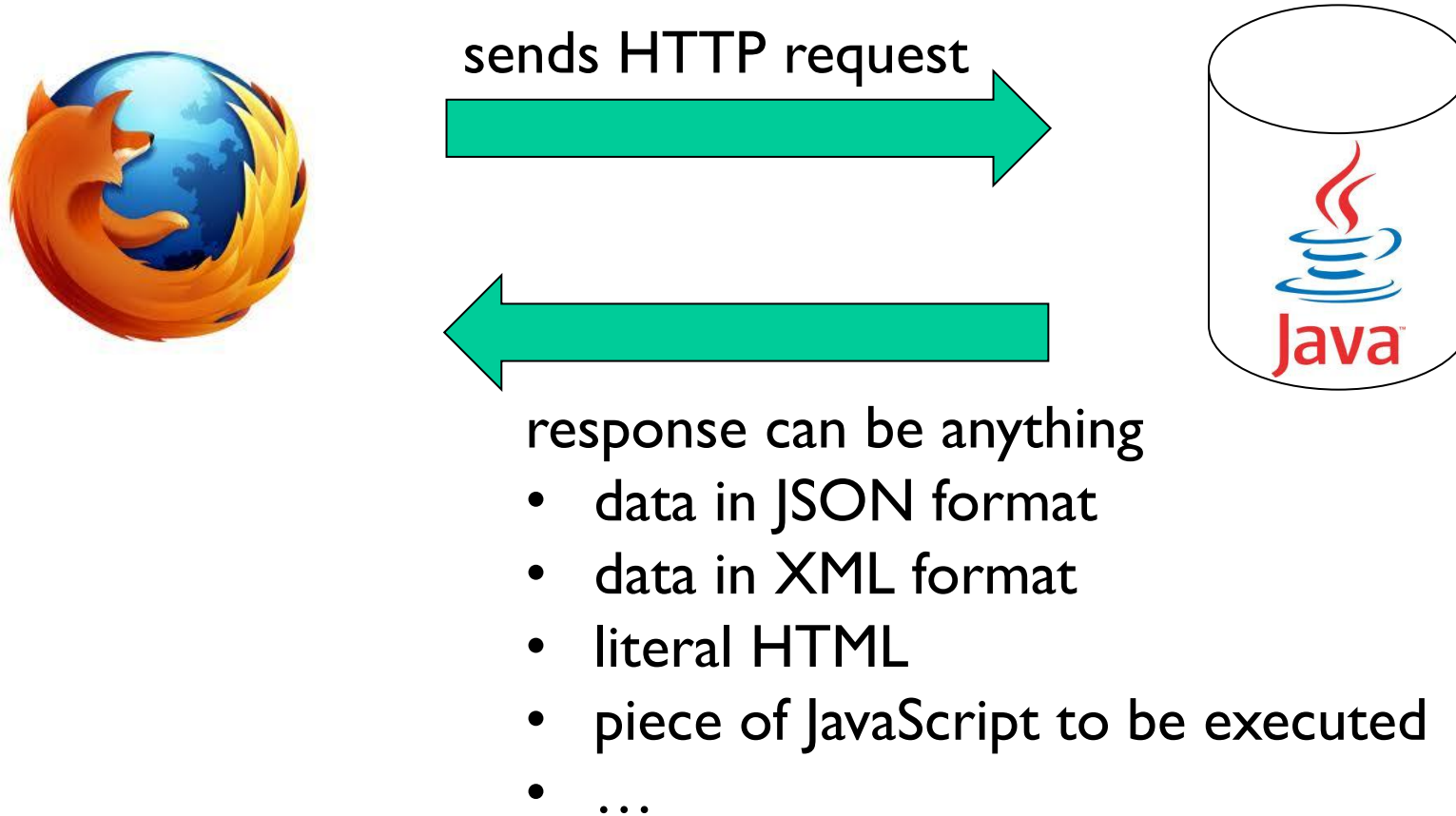
# run jasmine tests in your build, in eclipse,...



<https://github.com/jefklak/jasmine-junit-runner>

Ajax

# JavaScript apis to make HTTP calls to a server without page refreshes





# JSON?

serialization format for easily transferring javascript objects

```
var myObject = {  
  hello:[1,2,3],  
  tweets:2  
};
```

```
var myObjectAsString = JSON.stringify(myObject);
```



```
"{"hello":[1,2,3],"tweets":2}"
```

```
var parsedObject = JSON.parse(myObjAsString);
```

## load – using literal html

```
$( '#results' ).load( 'ajax/results.html' );
```

request *ajax/results.html* from server, and put complete response in element with id *results*

```
$( '#hole' ).load( 'ajax/results.html #peg' );
```

request *ajax/results.html* from server, retrieve element with id *peg* from response and put it in *#hole*

```
$( '#results' ).load( 'ajax/results.html', function() {  
    console.log( 'load completed' );  
});
```

when the response was received and put in *#results*, a callback function can be executed

## getScript – load and execute script

```
$.getScript('ajax/results.js');
```

request *ajax/results.js* from server, and executes the script in the global context

```
$.getScript('ajax/results.js', function() {  
    console.log('script executed');  
});
```

when script is loaded and executed, the callback will run

## getJSON

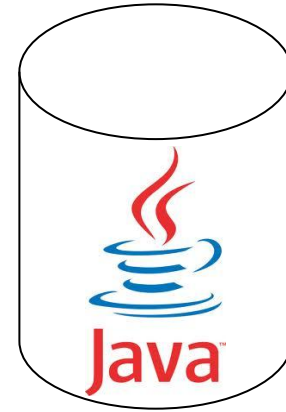
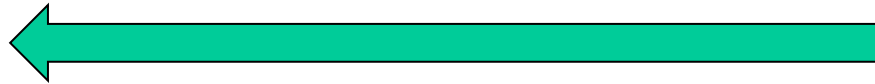
```
$.getJSON('ajax/results.json', function(data) {  
    console.log(data.naam);  
});
```

- request *ajax/results.json* from server
- parses the response as JSON data
- calls callback with parsed JavaScript object

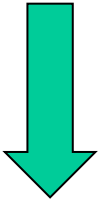
## Same origin policy



script loaded from tjoelaal.com

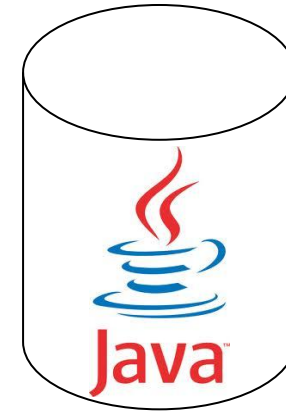


www.tjoelaal.com



script is **not** allowed to load  
JSON data or HTML from mail.google.com

script **is** allowed to load and run  
a JavaScript file from mail.google.com

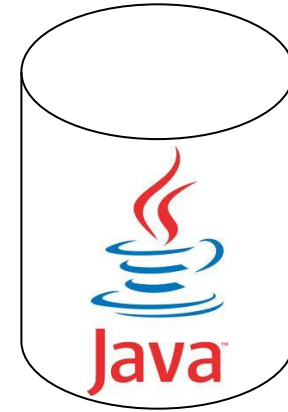
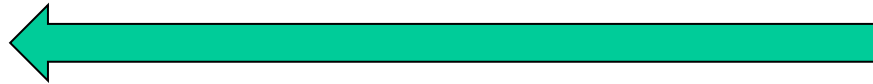


mail.google.com

# How to get around same origin policy

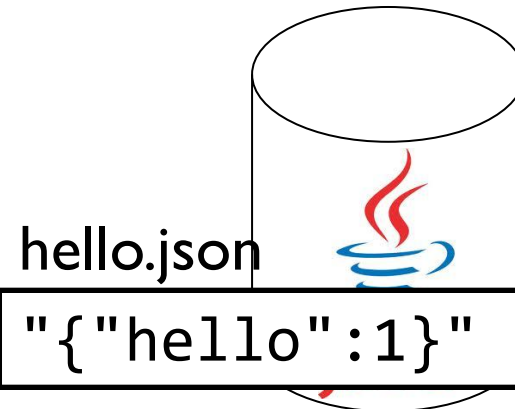


script loaded from tjoelaal.com



www.tjoelaal.com

how does script get JSON data?



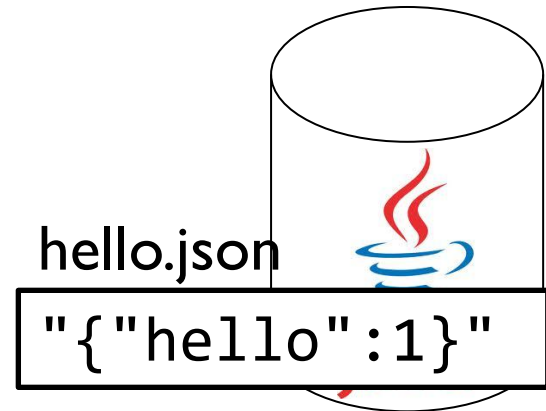
mail.google.com

# How to get around same origin policy



create a function parseData(jsonData)

```
function parseData(jsonData) {  
    var data = JSON.parse(jsonData);  
    $('#result').text(data.hello);  
}
```

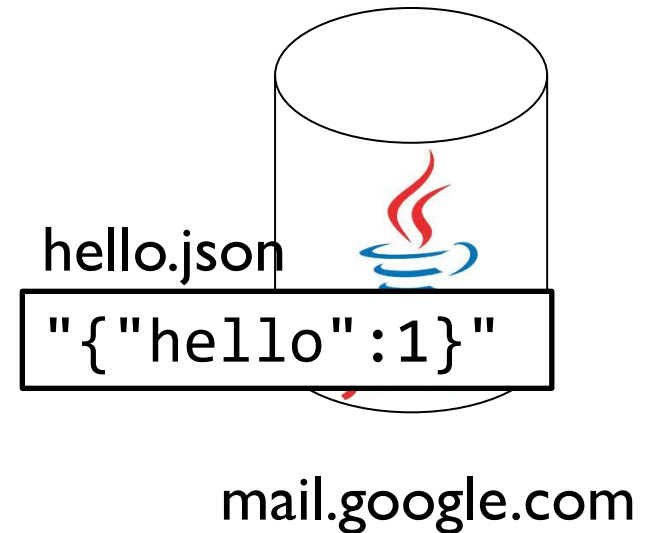


mail.google.com

# How to get around same origin policy



ask server for **script**  
`hello.json?callback=parseData`





# How to get around same origin policy



server generates script

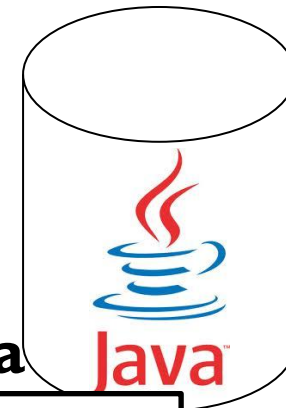
hello.json

```
"{"hello":1}"
```



hello.json?callback=parseData

```
parseData("{\"hello\":1}");
```



mail.google.com

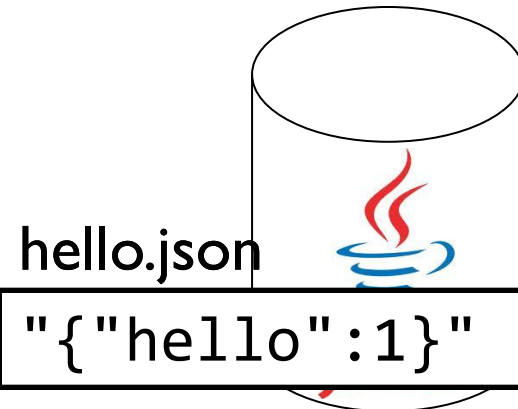
# How to get around same origin policy



hello.json?callback=parseData

```
parseData("{\"hello\":1}");
```

script will run and call our **parseData** function



mail.google.com

concept is called **JSONP**

## getJSON

```
$.getJSON('http://hello.json?callback=?', function(d) {  
    console.log(d.naam);  
});
```

- JQuery will automatically generate the callback function and put the name of this function instead of ?
- The generated function will call your given callback function with the parsed JSON data
- Server needs to support JSONP!

## **\$.ajax**

```
$.ajax();
```

Lower level function used by

- \$.getJSON()
- \$.getScript()
- \$(div).load()

allows more control over sending data to server

running callbacks before sending data, when things go wrong,...

allows configuring HTTP method (GET, POST, DELETE,..)

setting request headers

...

<http://api.jquery.com/jQuery.ajax/>

# Lab 5

## Reference

<http://api.jquery.com/category/ajax/>

