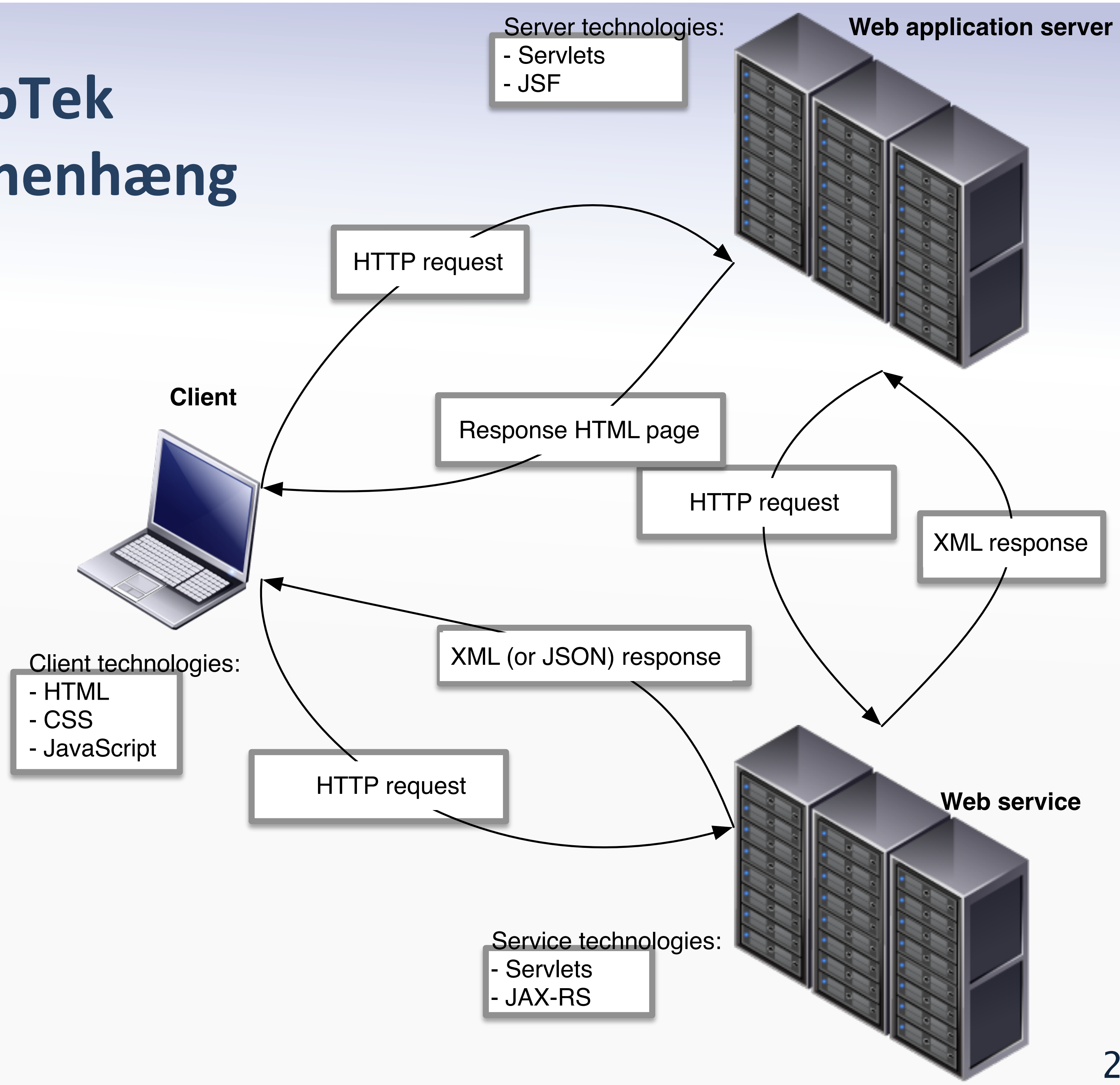
# Outro for dWebTek

Concepts revisited

# dWebTek sammenhæng

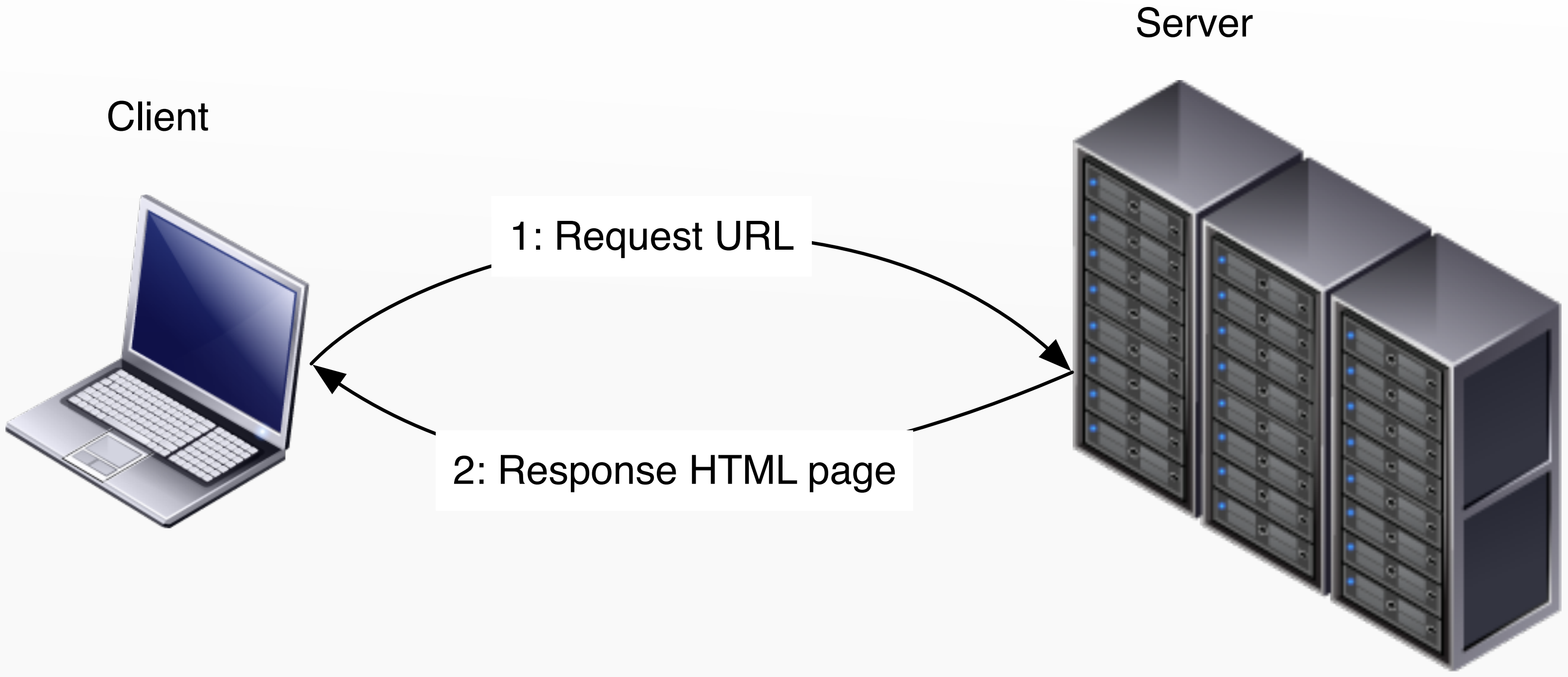


## Programming styles

- Server based web programming:
  - The server generates 'dead' HTML and sends it to the client
  - Clients posts a form
  - ... (back to first bullet)
- Client based web programming
  - The client runs a (JavaScript) program
  - Fetches data from the server
  - Sends data to the server
- Meaningful combinations?
  - The server generates programs that run on the client side?

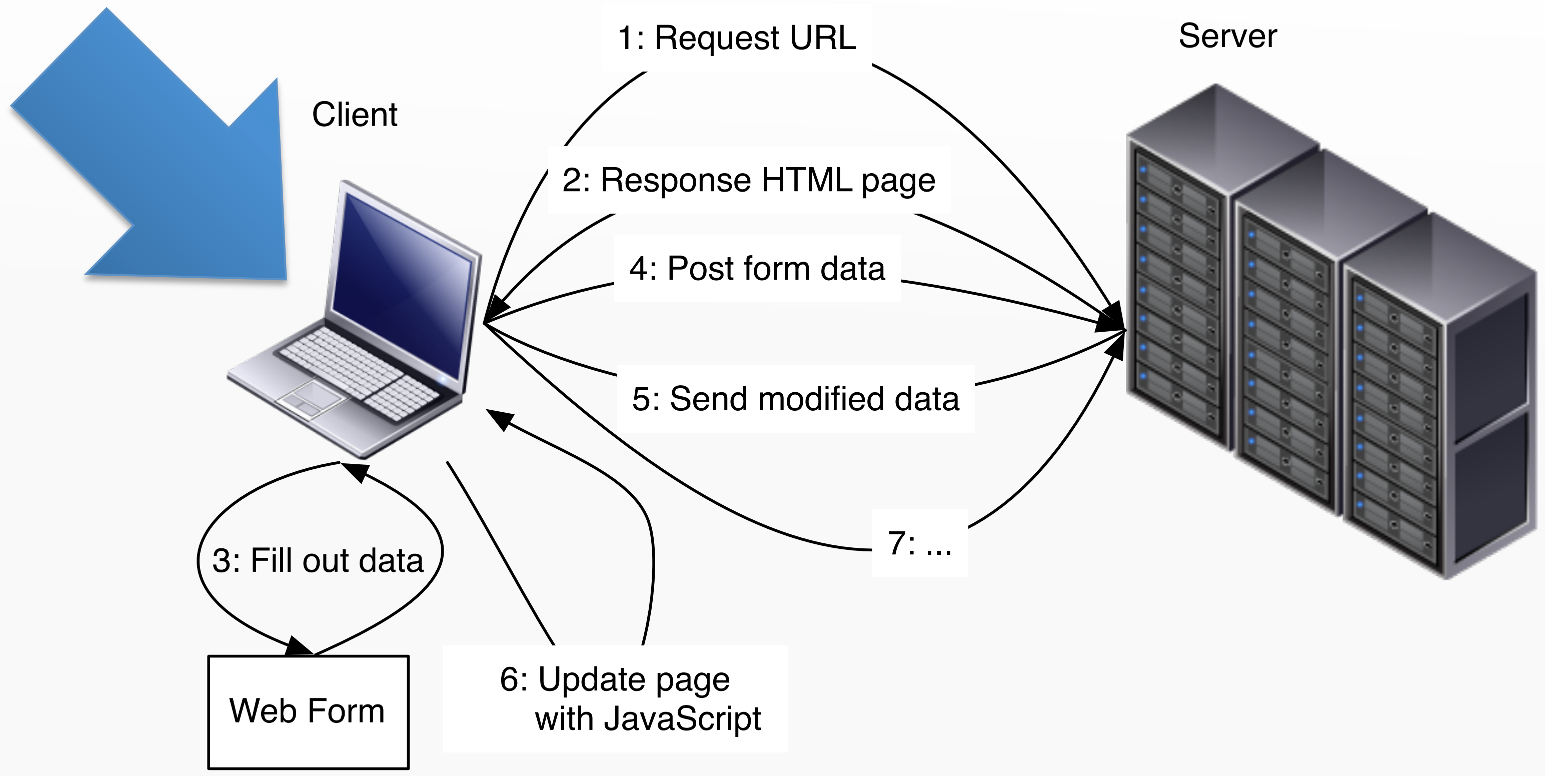
## Programming style: Server side

Simple web interaction



## Programming style: Client side

Modern web interaction



## Technologies and concepts

- Now, let's again revisit what you have learned:
  - XML, HTML, and CSS
  - Web services
  - DOM programming
  - Server state
    - Application, session, request
  - Model-view-controller
- None of this is Java, technology, or framework specific
  - We study these things in practice with concrete technologies

## Fundamental: XML, HTML, and CSS

- Omnipresent:
  - All web frameworks ultimately generate HTML and CSS
  - XML is used everywhere (maybe in too many places)

- We have seen:
  - Static HTML pages
  - HTML with JSF
  - Dynamic HTML pages with JavaScript
  - XML Schema for language syntax specification
- This knowledge is highly reusable:
  - No matter what framework, no matter what server language

## Concept: Web services

- Web services: platform-independent communication
  - Expose data via XML over HTTP
  - Very widely used on all platforms

- You have:
  - Used the cloud server as a web service client
  - Written your own web service in JAX RS
- Many web services out there
  - Google maps API
  - GeoIP
  - Many, many others...

## Concept: DOM Programming

- In DOM programming, we see HTML and XML as trees
  - Nodes (elements, text, etc.) are objects
  - Nodes have a list of children
  - We change documents my manipulating DOM objects

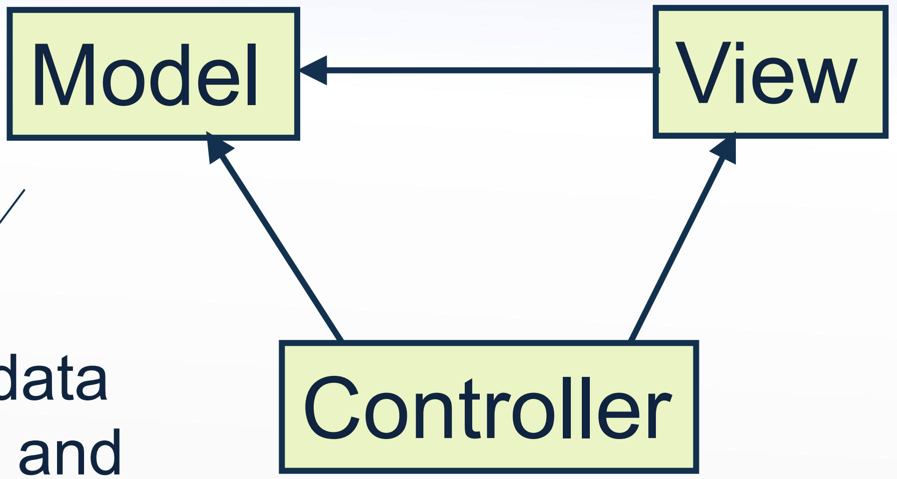
- We have seen:
  - JDOM: Java framework for manipulating XML trees
  - DOM in JavaScript: Framework for manipulating HTML/XML
- Our knowledge is re-usable beyond both!
  - DOM on .NET, DOM in C++, ...

## Concept: Server state scopes

- Server state:
  - Shared state: Shared for all requests for all clients
  - Session state: Shared for all requests for a single client
  - Transient state: Shared only during a single request
- We have seen:
  - Beans in JSF – Java objects as you know them!
  - Plain Java maps in Servlets – Type safety?
- Our knowledge is reusable:
  - Other frameworks and other languages have similar scopes
    - Some may not have all: PHP needs a database for application state
- When learning new frameworks: Find these scopes!

## Principle: The Model-View-Controller Pattern

In JSF:
Java beans



provides views to clients

In JSF:
XML templates

encapsulates data representation and business logic

handles interactions with clients

In JSF:
FacesController

## Concept: Model-view-controller

- MVC dominates the web framework landscape:
  - ASP.NET MVC (by Microsoft)
  - MonoRail for ASP.NET (by Castle Project)
  - JSF for Java (by Oracle)
  - Struts for Java (by Apache)
  - Joomla for PHP (by The Joomla Project Team)
  - Yii for PHP
  - And many, many more...

## Taking our knowledge further

- Chances are, you will program in other frameworks
  - Use your knowledge and relate!
  - Lets try that!

- Here is a framework that:
  - Is Client-side based (JavaScript based), like jQuery
  - Is Model-view-controller style, like JSF
  - Uses XML (like) templates, like JSF
- I give you: Google AngularJS
  - Quickly gaining popularity right now

## Remember MVC

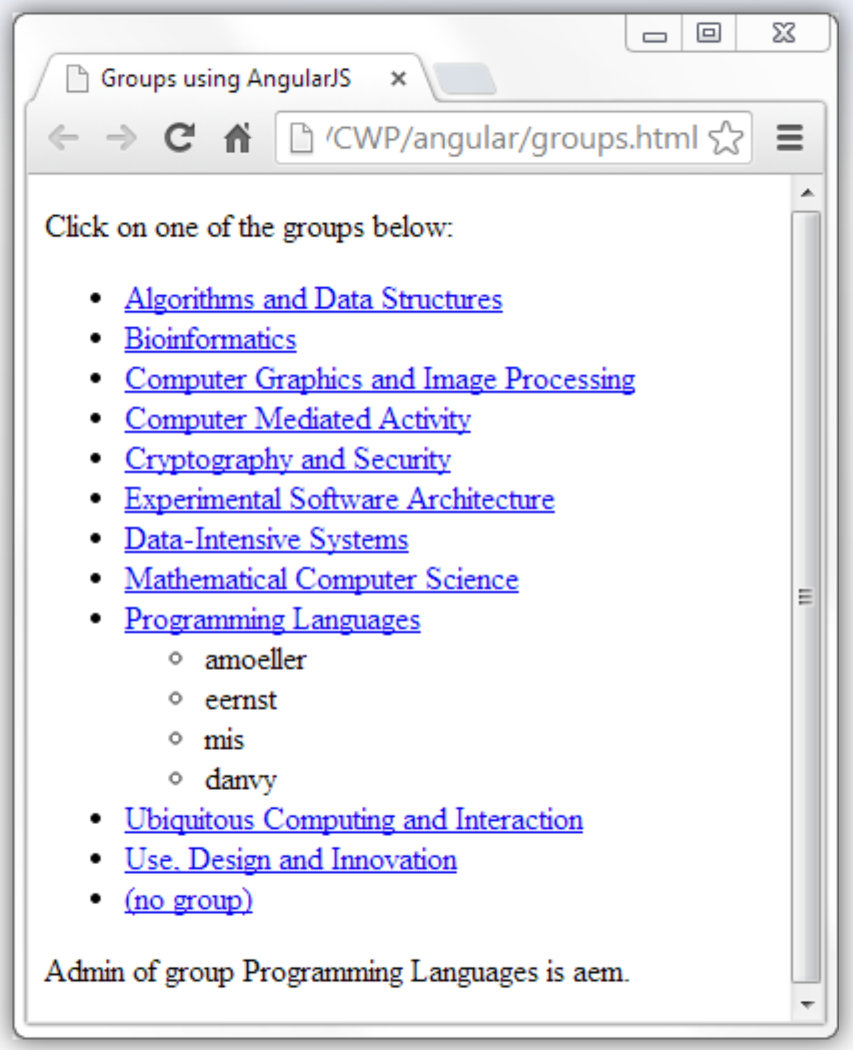
- To understand a model-view-controller framework:
  - Understand how the model is stored
  - Understand the language used for the view
  - Find out how the controller is handled
- Data scopes? Web services? DOM?
- Let us see how this looks in AngularJS!
  - (Don't worry, I won't ask about this at the exam)

## AngularJS

Client-side MVC framework:

- all HTML in the .html file
  (including the dynamically constructed HTML!)
- declarative “view” (reminiscent of EL in JSF)
  using {{...}} templating and custom tag attributes
- automatic re-computation of view

# Example: ResearchGroups



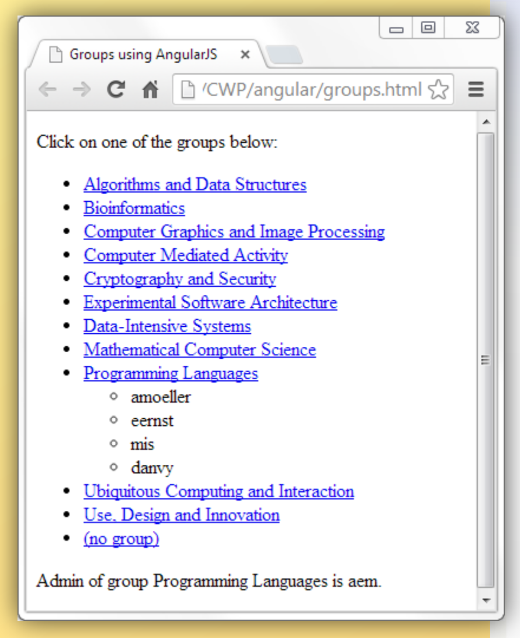
Click on one of the groups below:

- Algorithms and Data Structures
- Bioinformatics
- Computer Graphics and Image Processing
- Computer Mediated Activity
- Cryptography and Security
- Experimental Software Architecture
- Data-Intensive Systems
- Mathematical Computer Science
- Programming Languages
  - amoeller
  - eernst
  - mis
  - danvy
- Ubiquitous Computing and Interaction
- Use, Design and Innovation
- (no group)

Admin of group Programming Languages is aem.

# ResearchGroups in AngularJS (1/3)

```
<!doctype html>
<html ng-app="groupApp">
  <head>
    <meta charset="utf-8">
    <title>ResearchGroups using AngularJS</title>
    <style type="text/css">
      ul.visible { display: block; }
      ul.hidden { display: none; }
    </style>
    <script src="lib/angular/angular.js"></script>
    <script src="js/app.js"></script>
  </head>
  <body ng-controller="GroupsController">
    <p>{{beforeText()}}</p>
    <ul>
      <li ng-repeat="group in groups">
        <a href="" ng-click="setActiveGroup(group)">{{group.name}}</a>
        <ul ng-controller="GroupController" class="{{display()}}">
          <li ng-repeat="member in group.members">{{member}}</li>
        </ul>
      </li>
    </ul>
    <p>{{afterText()}}</p>
  </body>
</html>
```



Click on one of the groups below:

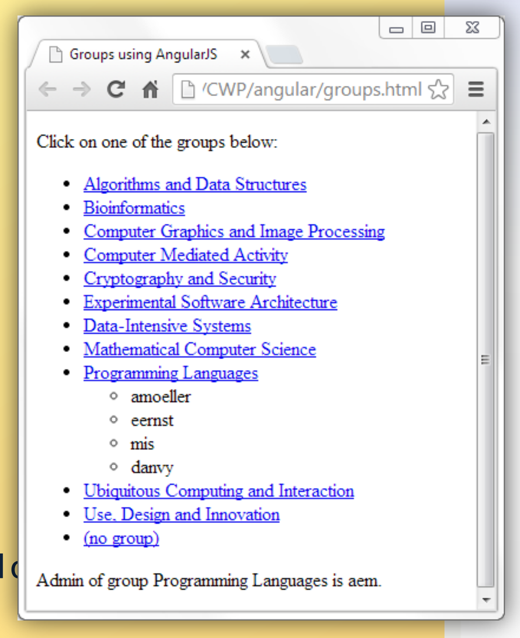
- Algorithms and Data Structures
- Bioinformatics
- Computer Graphics and Image Processing
- Computer Mediated Activity
- Cryptography and Security
- Experimental Software Architecture
- Data-Intensive Systems
- Mathematical Computer Science
- Programming Languages
  - amoeller
  - eernst
  - mis
  - danvy
- Ubiquitous Computing and Interaction
- Use, Design and Innovation
- (no group)

Admin of group Programming Languages is aem.

Much like JSF templates!

## ResearchGroups in AngularJS (2/3)

```
var groupApp = angular.module('groupApp', []);

groupApp.controller('GroupsController', function($scope, $http) {

  $http.get('data/groups.json').success(function(data) {
    $scope.groups = data;
  });

  $scope.activeGroup = null;
  $scope.setActiveGroup = function(group) {
    $scope.activeGroup = $scope.activeGroup == group ? null : group;
  }

  $scope.beforeText = function() {
    return $scope.groups.length > 0 ? "Click on one of the groups bel
  }

  $scope.afterText = function() {
    return $scope.activeGroup ? "Admin of group " + $scope.activeGroup.name + " is "
      + $scope.activeGroup.admin + "." : "";
  }
});
```



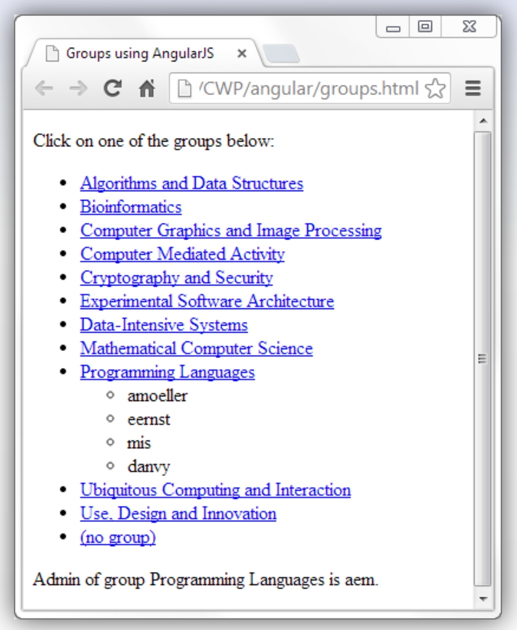
Click on one of the groups below:

- Algorithms and Data Structures
- Bioinformatics
- Computer Graphics and Image Processing
- Computer Mediated Activity
- Cryptography and Security
- Experimental Software Architecture
- Data-Intensive Systems
- Mathematical Computer Science
- Programming Languages
  - amoeller
  - eernst
  - mis
  - danvy
- Ubiquitous Computing and Interaction
- Use, Design and Innovation
- (no group)

Admin of group Programming Languages is aem.

# ResearchGroups in AngularJS (3/3)

```
groupApp.controller('GroupController', function($scope) {

  $scope.display = function() {
    return !$scope.activeGroup ||
      $scope.activeGroup.id != $scope.group.id
        ? "hidden" : "visible";
  };

});
```



Click on one of the groups below:

- Algorithms and Data Structures
- Bioinformatics
- Computer Graphics and Image Processing
- Computer Mediated Activity
- Cryptography and Security
- Experimental Software Architecture
- Data-Intensive Systems
- Mathematical Computer Science
- Programming Languages
  - amoeller
  - eernst
  - mis
  - danvy
- Ubiquitous Computing and Interaction
- Use, Design and Innovation
- (no group)

Admin of group Programming Languages is aem.

## More about JavaScript

- JavaScript is a complex language with many features
- Let us look at one of them: Prototypes
  - Essentially JavaScript's version of inheritance

- ... but how do we make inheritance without classes?
- ... and how do we encapsulate fields (make them private)?

## Recall: Functions as constructors

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

var x = new Person("John Doe");
```

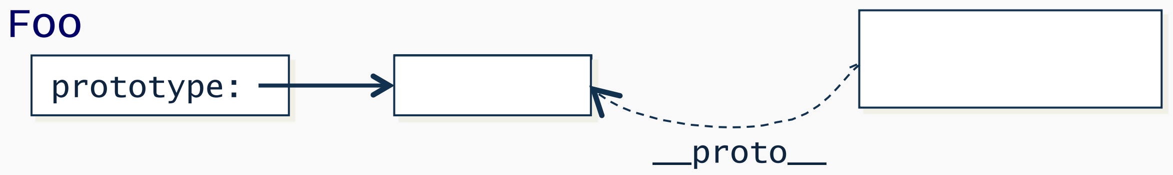
now x is a new object
with a name property

## Prototypes

- Every function object has a **`prototype`** property (aka. the *explicit prototype link*) for sharing information between instances

- `new Foo(...)` constructs a new empty object, sets the `__proto__` property (aka. the *internal prototype link*) to the prototype of `Foo`, and invokes `Foo` as a constructor

- Property lookup searches via the internal prototype link



- This is heavily used in the HTML DOM!

# Sharing with prototypes

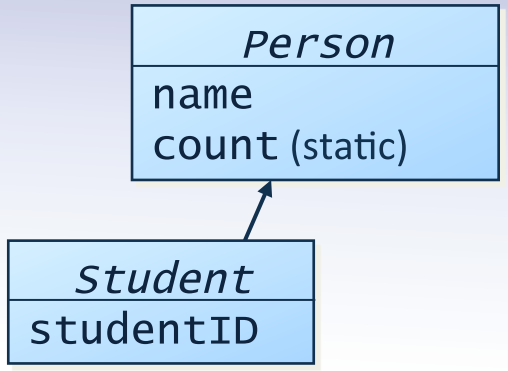
```
js> function Foo(n) { this.name = n; Foo.prototype.counter++; }
js> Foo.prototype.counter = 0;
0
js> var x = new Foo("hello");
js> var y = new Foo("world");
js> x.name
hello
js> y.name
world
js> x.counter
2
js> y.counter
2
```

# Another example

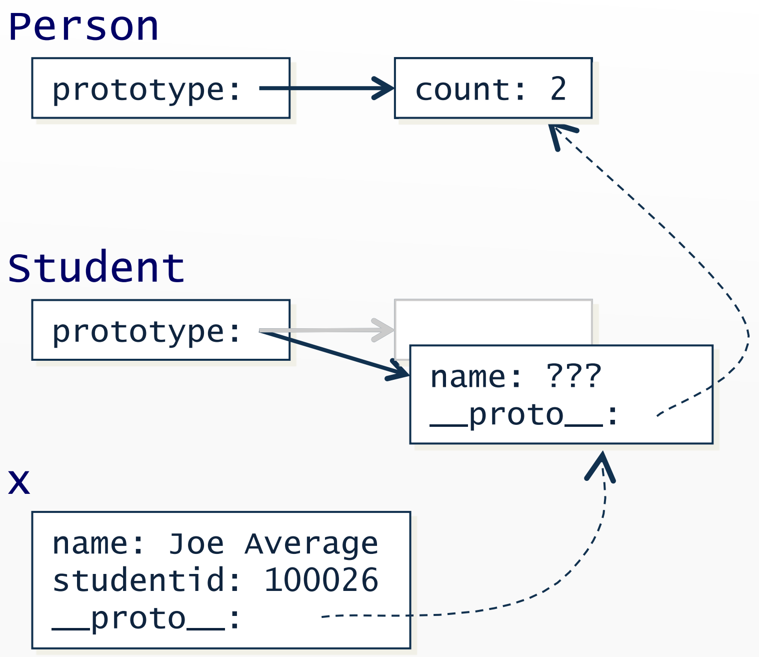
Java-style default `equals` and `hashCode` methods for all objects:

```
Object.prototype.equals = function(other) {
  this === other;
}

Object.prototype.hashCode = function() {
  if (!(myHashCode in this)) {
    this.myHashCode = Math.random();
  }
  return this.myHashCode;
}
```

## Prototype-based inheritance



```
function Person(n) {
  this.name = n || "???";
  Person.prototype.count++;
}
Person.prototype.count = 0;

function Student(n,s) {
  Person.call(this, n);
  this.studentid = s;
}
Student.prototype = new Person;
```



```
var x = new Student("Joe Average", "100026");
print(x.count); // returns 2
```

## Modules

- The global object gets crowded in large programs
  - Java has packages to structure programs
  - C# has namespaces
  - …
- In JavaScript we can use objects as modules:

```
var cwp = {
  addOne: function(n) { return n+1; }
  addTwo: function(n) { return n+2; }
}

var another_module = {
  addOne: function(n) { return n+"1"; }
}

var s = cwp.addOne(42); // yields 43
```

## Functions as modules

- Combine the tricks for a more fancy pattern:

```
var cwp = (function() {
  // internal stuff
  var next = 1;
  function getNext() { return counter++; }
  function reset() { counter = 1; }
  // expose the public interface
  return {
    getNext : getNext,
    reset : reset
  }
})();
var s = cwp.getNext(); // yields 1
```

- Recommended for large programs! (really!)

## But I want more!

- *Advanced Web Programming*, learn much more about:
  - The JavaScript language
  - Other client-side languages:
    - Dart, GWT, Flapjax
  - Frameworks and styles
    - What happens if we use JavaScript on the server side?
  - Implementing web languages
    - How could be implement languages like PHP and JavaScript?
    - What are the trade-offs?
  - Research in the web technologies area

## But I want more!

- Many interesting projects for your:
  - Master's thesis
  - PhD studies

- Come talk to us in the programming languages group!

## Exam

## When do we get the grade?

- Your TA will have graded your projects before 26/3
- We will grade your exam sets on 26/3

- Official registration:
  - Your grade needs to be entered into university IT systems
  - A lot of other exams take place right now
  - Expect it to take some weeks to get your grade!

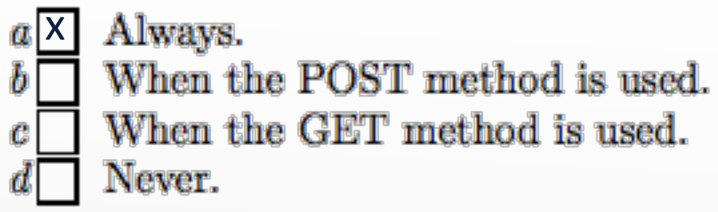
- There is nothing I can do to give you the grade faster

## Exam form

- Multiple choice exam
  - 120 minutes
  - Roughly 55 questions
    - One(!) correct answer per question
  - You may bring nothing but a pen
    - No notes
    - No cell phones
    - …

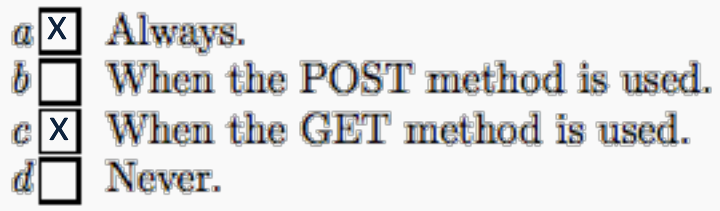
# Answering the exam questions

- There is exactly one correct answer per question
- If you know it, tick it off:



*a* [✓ Always.]
*b* ☐ When the POST method is used.
*c* ☐ When the GET method is used.
*d* ☐ Never.

- If you are in doubt, tick those off those you think it could be



*a* [✓ Always.]
*b* ☐ When the POST method is used.
*c* [✓ When the GET method is used.]
*d* ☐ Never.

## Grading

- Up to 100 points for the multiple choice test
  - Positive points for:
    - Correct answer
    - Multiple answers where one is correct
      - Less points than for a single, correct answer
  - Negative points for:
    - Wrong answer
    - Multiple answers where none are correct
      - More negative points than a single, wrong answer
- Statistically, random guessing gives 0 points

http://cs.au.dk/~mis/multiple.pdf

## Contents

- Same as what you needed for the project
  - 'Material' for each week
  - Knowledge about the project

- ... so you have already studied a lot for it!

## Preparation 1/2

- Old exam set on the front page:
  - Curriculum has changed a lot this year
- Clicker questions
  - Many of those could be exam questions

- Meaningful use of test questions:
  - Reflect over and revisit the technologies
- Meaningless use of test questions:
  - Learn answers to test questions
  - I am going to ask you about something else.

## Preparation 2/2

- Read the notes that you took during the lectures
- Spend time doing the weekly exercises
- Do the old exam sets
  - But expect the questions to be different
- Ask your TAs for help!

- ... maybe you should have a second look at XML Schema? ;-)

<blink>Good luck at the exam!</blink>