JavaScript Beyond jQuery

John Wilander

@johnwilander

shouldilearnjavascript.com

Part 1: Typing

Types

Java

Primitive:

`byte, short, int, long, float, double, boolean, char`

The rest is objects, so called reference types

JavaScript

Primitive:

`string, number, boolean, function, undefined`

The rest is `object`

Types

Java

Primitive:

`byte, short, int, long, float, double, boolean, char`

JavaScript

Primitive:

`string, number, boolean, function, undefined`

Types

Java

Primitive:

`byte, short, int, long, float, double, bool`

Strings are not general objects, they're a type

JavaScript

Primitive:

`string, number, boolean, function, undefined`

Types

Java

Primitiv
byte,
boole
long, float, double,

Only one type for numbers, i.e. no difference between integers or decimals

JavaScript

Primitive:

`string, number, boolean, function, undefined`

Types

Java

Primitive:

`byte, short, int, long, float, double, boolean, char`

Functions are full objects with their own type

JavaScript

Primitive:

`string, number, boolean, function, undefined`

Types

Java

Primitive:

byte, short, int, long, float, double, bool

JavaScript's version of **uninitialized is** `undefined`

JavaScript

Primitive:

string, number, boolean, function, undefined

Static vs Dynamic Typing

Java

Static typing:

```
String name;
name = "John";
name = 34;
```

Variables have types

Values have types

Variables cannot change type

JavaScript

Dynamic typing:

```
var name;
name = "John";
name = 34;
```

Variables have no types

Values have types

Variables change type dynamically

Dynamic Typing

JavaScript

```
var name = "John";
```

Variables have no types

Values have types

Strong vs Weak Typing

Strong vs Weak Typing

Strong !== good

Weak !== bad

Strong vs Weak Typing

Java

```
int intVar = 4;
"" + intVar;
intVar + 0.0;
intVar + (int)0.0;
intVar + null;
```

Pretty strong typing
Implicit type conversion
Implicit, widening type conv.
Explicit type conversion
~~Not allowed~~

JavaScript

```
var dynVar = 4;
dynVar + "";
dynVar + null;
dynVar + [];
dynVar + [0];
```

Very weak typing
Implicit type conversion
Implicit type conversion
Implicit type conversion
Implicit type conversion

Type Safe

≈

no type errors at runtime

John's Conclusion

Java is a statically, strongly typed language with decent type safety

JavaScript is a dynamically, weakly typed language with decent type safety

Object Orientation

@johnwilander

Java: Classes & Objects

Source

Class

Class

Executing code

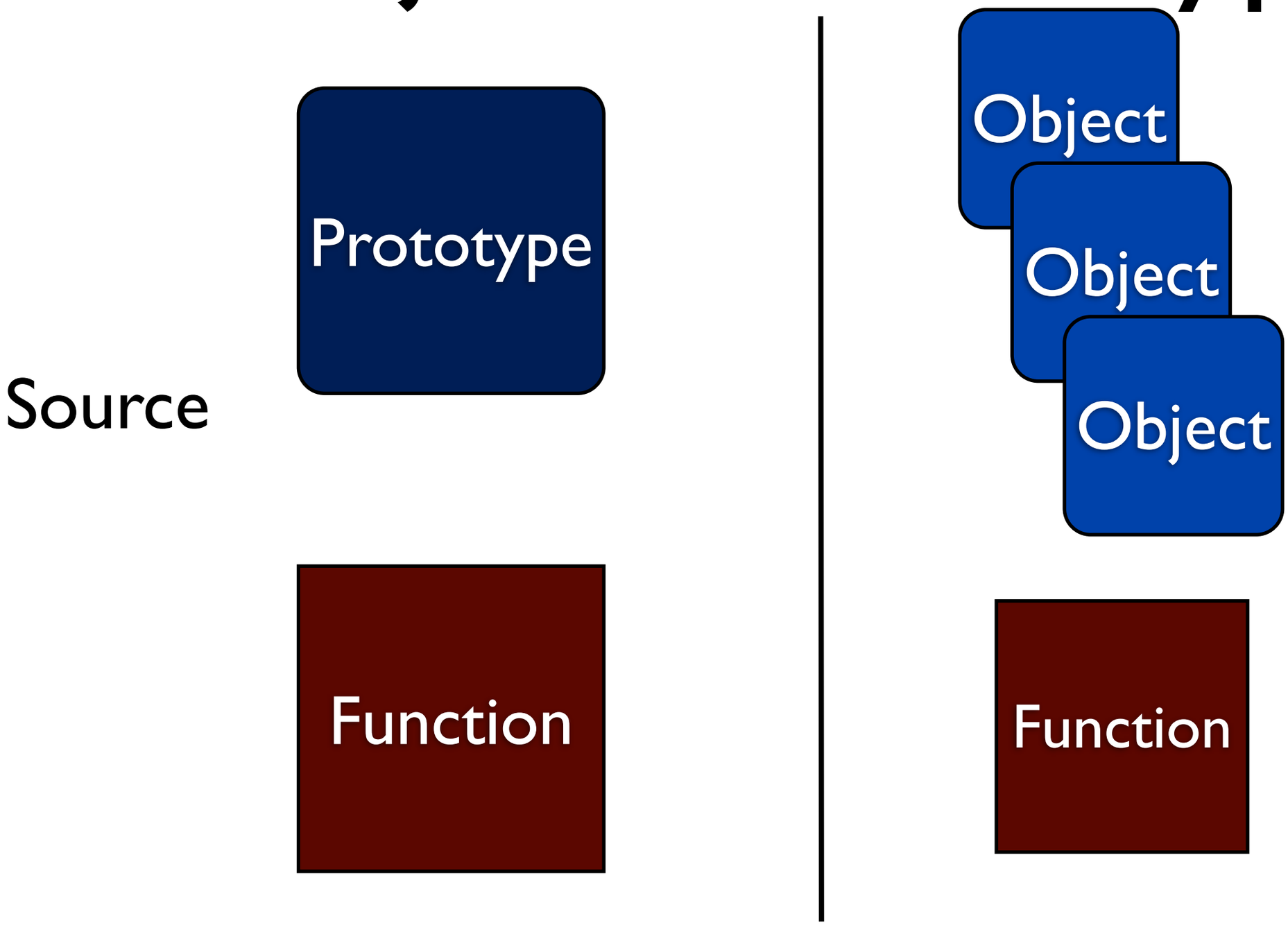
Object

Object

Object

Singleton

JavaScript: Functions, Object & Prototypes



JavaScript: Functions, Object & Prototypes

Prototype

Source

Function

Anonymous closures

Modules

Object

Object

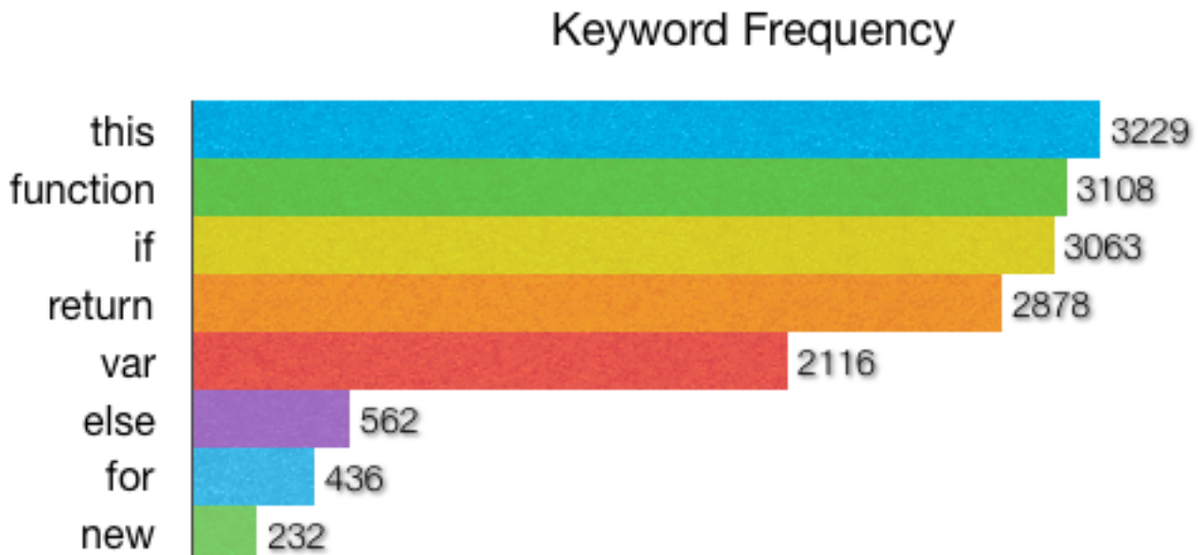
Object

Executing code

Function

@johnwilander

Keywords in JavaScript



Statistics from:

jQuery

jQuery.Mobile

Prototype

Ext JS

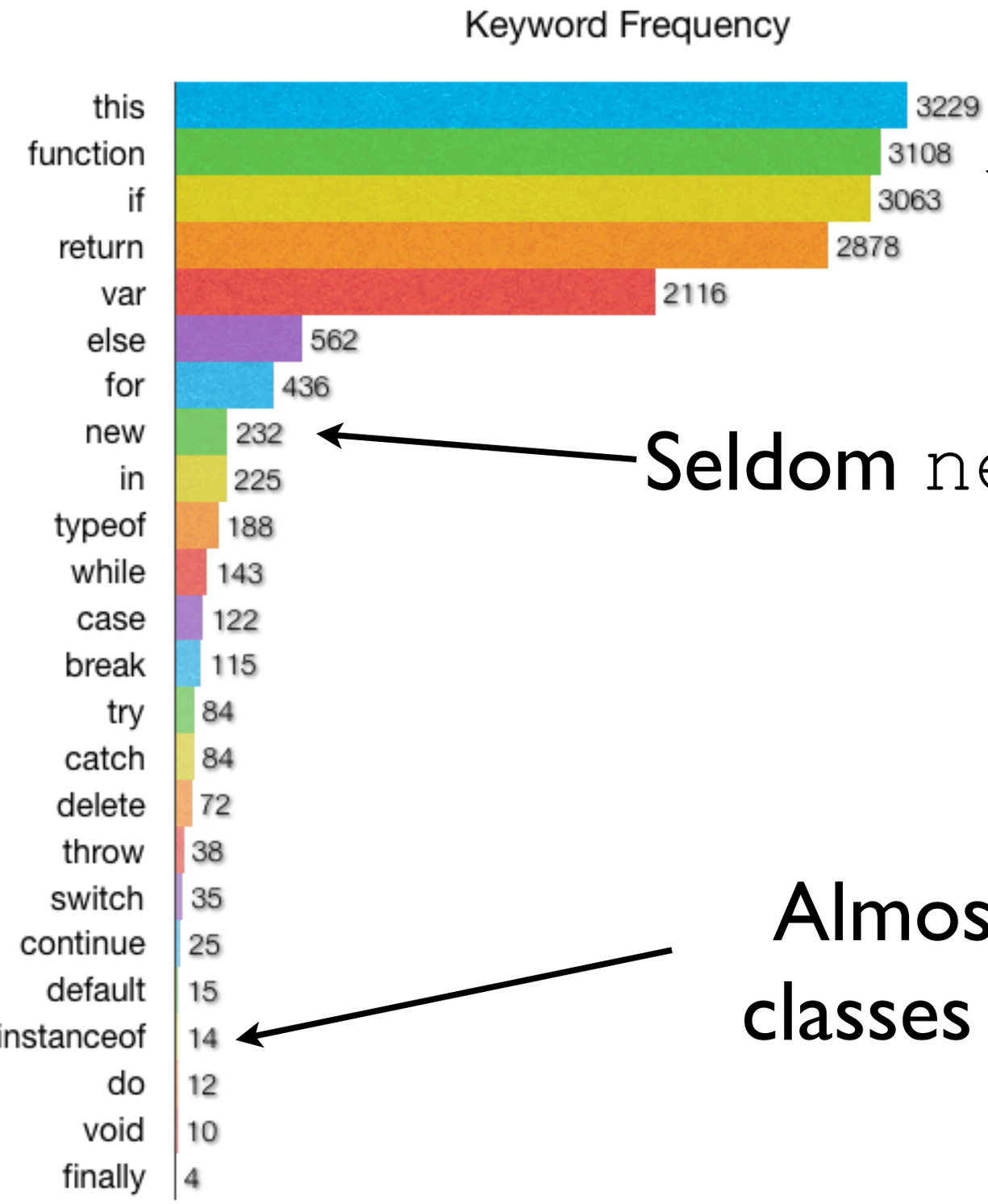
MooTools

Backbone

Underscore

http://ariya.ofilabs.com/2012/03/most-popular-javascript-keywords.html

Keywords in JavaScript

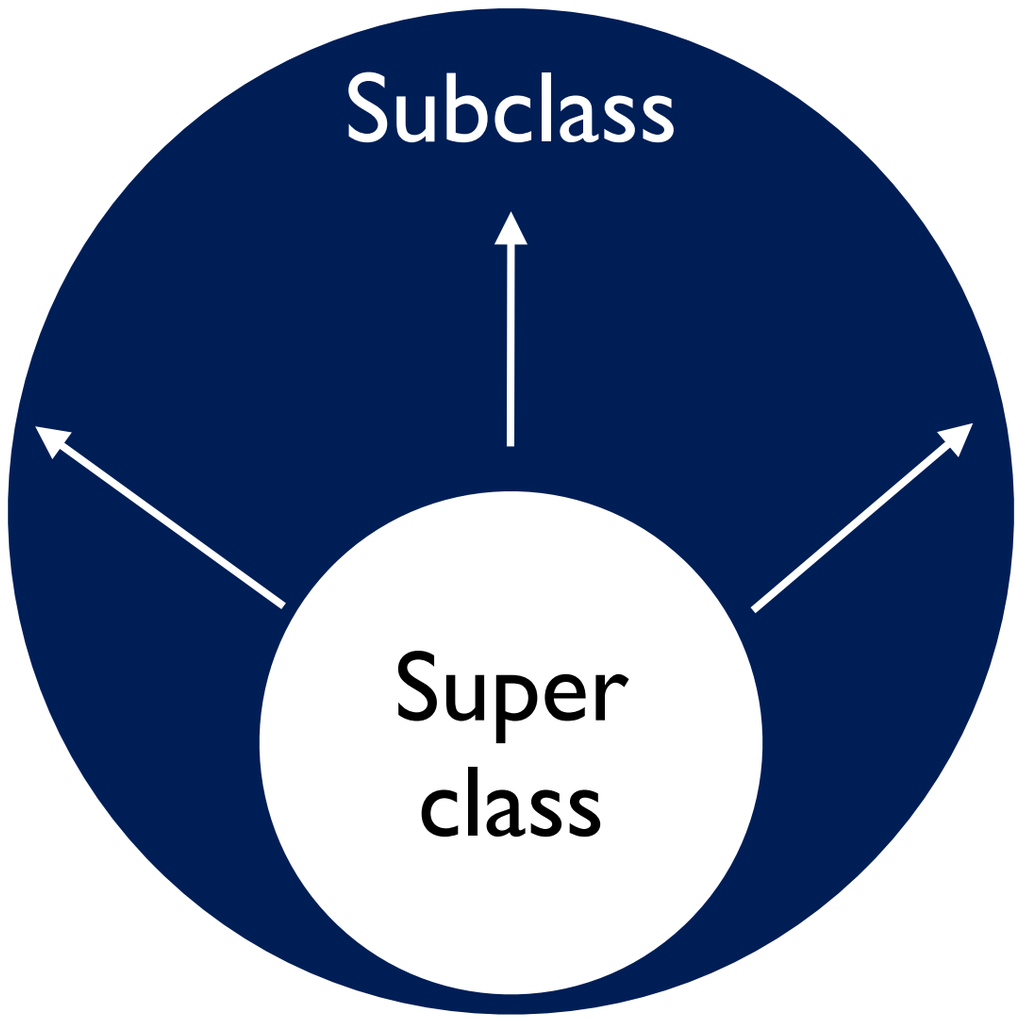


Functions,
functions,
functions,
functions

Seldom `new`

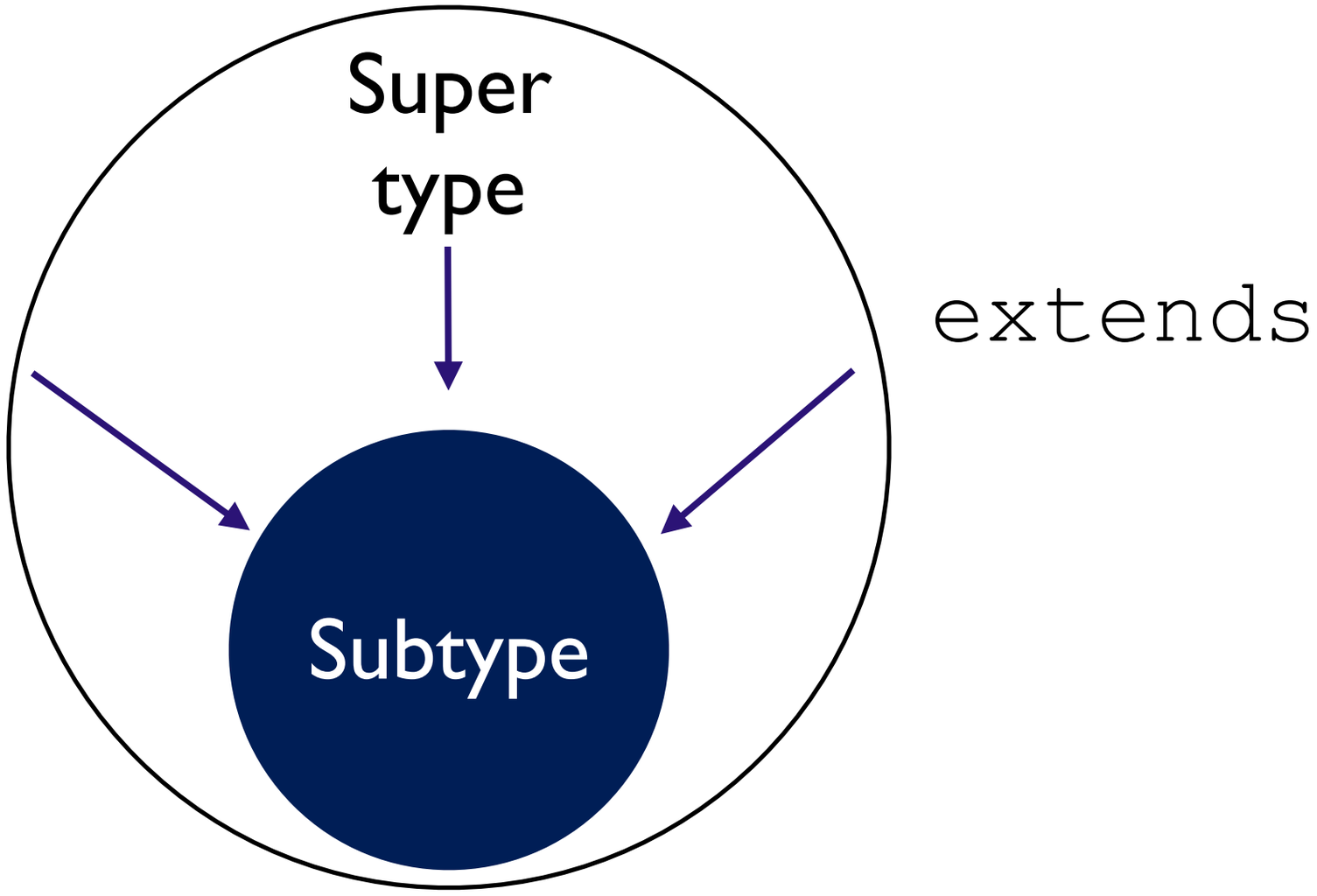
Almost never
classes as types

Java: Extension by Inheritance



`extends`

Java: Subtyping by Inheritance



Through inheritance
Java classes get larger
in terms of functions
and
smaller in terms of type
(more specific)

In JavaScript
extension
and
subtyping
are separate things

Why do we want to subtype?
Because we think in terms of static typing and *is-a*-relations

@johnwilander

Static Typing and Subtyping

Java

```
public void doPurchase(Item item)
```

Accepts `Item`
or subtypes to `Item`

Dynamic Typing

JavaScript

```
public void doPurchase(Item item)
```

Dynamic Typing

JavaScript

```
void doPurchase(Item item)
```

Dynamic Typing

JavaScript

```
doPurchase(Item item)
```

Dynamic Typing

JavaScript

```
doPurchase(      item)
```

Dynamic Typing

JavaScript

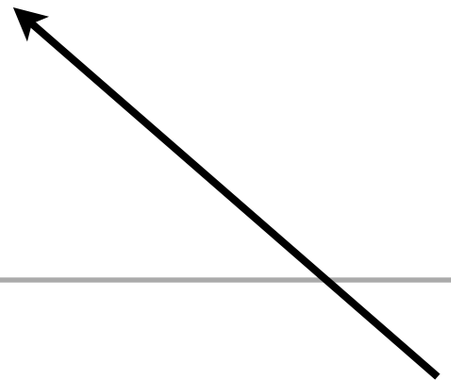
```
function doPurchase(item)
```

Accepts anything
... or nothing

Dynamic Typing

JavaScript

```
function doPurchase(item)
```



Can return anything
... or nothing

Part 2: To start a JavaScript program

JavaScript and Web

```
<html>
<head>
  <script src="js/main.js"></script>
  <script>
        var JW = {};
        // More code
  </script>
</head>
<body>
  <button onclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy text</button>
  <div>
    Some information
  </div>
  <script>
        JW.calcPrice = function(price, vat) { //… };
  </script>
  <script src="http://3rdparty.com/trackUsers.js"></script>
</body>
</html>
```

JavaScript and Web

```
<html>
<head>
  <script src="js/main.js"></script>
  <script>
        var JW = {};
        // More code
  </script>
</head>
<body>
  <button onclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy text</button>
  <div>
    Some information
  </div>
  <script>
       JW.calcPrice = function(price, vat) { //… };
  </script>
  <script src="http://3rdparty.com/trackUsers.js"></script>
</body>
</html>
```

Script from file in `<head>`
Synchronous loading and execution

JavaScript and Web

```
<html>
<head>
  <script src="js/main.js"></script>
  <script>
        var JW = {};
        // More code
  </script>
</head>
```

Inlined script in <head>
Synchronous execution
before DOM is loaded

```
<body>
  <button onclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy text</button>
  <div>
    Some information
  </div>
  <script>
       JW.calcPrice = function(price, vat) { //… };
  </script>
  <script src="http://3rdparty.com/trackUsers.js"></script>
</body>
</html>
```

JavaScript and Web

```
<html>
<head>
  <script src="js/main.js"></script>
  <script>
        var JW = {};
        // More code
  </script>
</head>
<body>
  <button onclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy text</button>
  <div>
    Some information
  </div>
  <script>
        JW.calcPrice = function(price, vat) { //… };
  </script>
  <script src="http://3rdparty.com/trackUsers.js"></script>
</body>
</html>
```

Script directly in HTML, executed on click event

JavaScript and Web

```
<html>
<head>
  <script src="js/main.js"></script>
  <script>
        var JW = {};
        // More code
  </script>
</head>
<body>
  <button onclick="document.getElementById('field2').value=
document.getElementById('field1').value
  <div>
    Some information
  </div>
  <script>
        JW.calcPrice = function(price, vat) { //… };
  </script>
  <script src="http://3rdparty.com/trackUsers.js"></script>
</body>
</html>
```

Inlined script in <body>
Executed when the parser gets here

JavaScript and Web

```
<html>
<head>
  <script src="js/main.js"></script>
  <script>
        var JW = {};
        // More code
  </script>
</head>
<body>
  <button onclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy text</button>
  <div>
    Some information
  </div>
  <script>
        JW.calcPrice = function(price
  </script>
  <script src="http://3rdparty.com/trackUsers.js"></script>
</body>
</html>
```

Script from file in <body>
Loaded and executed synchronously

Good Practice:

1. All JavaScript on file.

2. All statically loaded JavaScript in one block.

Dynamic Loading

```
(function(){
 var oldFirstScript =
   document.getElementsByTagName('script')[0];
     newScript =
   document.createElement('script'),

 newScript.async = true;
 newScript.src = 'js/script.js';

 oldFirstScript.parentNode.insertBefore(
   newScript, oldFirstScript);
}());
```

OK, that's loading.

But I want to *start*.

In the Beginning

Java

```
public static void main(String[] args)
```

JavaScript

```
(function(){}());

window.someGlobalFunction();
```

Self-Invoking Functions

Anonymous, self-invoking with parameter

```
(function myFunc(me) {
  // Code
}('John'));
```

Anonymous, self-invoking without parameter

```
(function myFunc() {
  // Code
}());
```

Part 4: Variables

Scope

JavaScript

```
var GLOB = {};

GLOB.func = function(array) {
   var local = …;

   …
   for (var i=0; i<array.length; i++){
      …
      var temp = array[i];
   }
};
```

Scope

JavaScript

```
var GLOB = {};

GLOB.func = function(array) {
  var local = …, i, temp;
  …
  for (i=0; i<array.length; i++){
    …
    temp = array[i];
  }
};
```

Scope

JavaScript

```
var GLOB = {};

GLOB.func = function(array) {
  var local = …, i, temp;
  …
  for (i=0;
    …
    temp =
  }
};
```

There are only two variable scopes: Global or local function scope.

All local variables are *hoisted* to the top of the function.

Closure

JavaScript

```
(function(array) {
  var i, temp;
  …
  for (i=0; i<array.length; i++){
    …
    temp = array[i];
  }
}([1, 2, 3]));
```

Closure

JavaScript

```
(function(array) {
  var i, temp;
  …
  for (i=0; i<array.length; i++){
    …
    temp =
  }
}([1, 2, 3
```

A closure keeps its scope and its context during its life span.

A reference ending up inside the closure (`array` above) will keep its value inside the closure.

Closure Pitfall

JavaScript

```
for(i=0; i<pages.length; i++) {
  var page = pages[i];
  pageElement.load(page.url,
    function() {
      launchForm(page.form);
    });
}
```

Closure Pitfall

JavaScript

```
for(i=0; i<pages.length; i++) {
  var page = pages[i];
  pageElement.load(page.url,
    function() {
      launchForm(page.form);
    });
}
```

Callback function since `pageElement.load()` is asynchronous

Closure Pitfall

JavaScript

```
for(i=0; i<pages.length; i++) {
  var page = pages[i];
  pageElement.load(page.url,
    function() {
      launchE
    });
}
```

`page` is hoisted and thus gets a new value for each iteration

Closure Pitfall

JavaScript

```
var page;
for(i=0; i<pages.length; i++) {
  page = pages[i];
  pageElement.load(page.url,
    function() {
      launchE
    });
}
```

`page` is hoisted and thus gets a new value for each iteration

Closure Pitfall

JavaScript

```
var page;
for(i=0; i<pages.length; i++) {
  page = pages[i];
  pageElement.load(page.url,
    function() {
      launchForm(page.form);
    });
}
```

When the callback is run `page` will refer to `pages[pages.length-1]` or at least another value than intended.

Closure Pitfall

JavaScript

```
var page;
for(i=0; i<pages.length; i++) {
  page = pages[i];
  pageElement.load(page.url,
    function() {
      launchForm(page.form);
    });
}
```

Making Use of Closures

JavaScript

```
var page;
for(i=0; i<pages.length; i++) {
  page = pages[i];
  (function(page) {
    pageElement.load(page.url,
      function() {
          launchForm(page.form);
      }
    });
  })(page);
}
```

Making Use of Closures

JavaScript

```
var page;
for(i=0; i<pages.length; i++) {
  page = pages[i];
  (function(page) {
    pageElement.load(page.url,
      function() {
          launchForm(page.form);
      }
    });
  })(page);
}
```

We bind `page` to a new closure so that the callback refers correctly

With self-invoking functions,
scope, and closures we are ready for
the most important design pattern:

Part 5: The Crockford Module or *Module Pattern*

JavaScript

```
JW.cache = (function(){}());
```

JavaScript

```
JW.cache = (function(){
  return {};
}());
```

JavaScript

```
JW.cache = (function(){
  return {
    getPatient:
      function(personnummer) {

      }
  };
}());
```

JavaScript

```
JW.cache = (function(){
  var cache = {};
  return {
    getPatient:
      function(personnummer) {

      }
  };
}());
```

JavaScript

```
JW.cache = (function(){
  var cache = {};
  return {
    getPatient:
      function(personnummer) {
        var res=cache[personnummer];
        // Check if fresh
        // Return
      }
  };
}());
```

JavaScript

```
JW.cache = (function(){
  var cache = {};
  return {
    getPatient:
      function(personnummer) {
        var res=cache[personnummer];
        if(_isValid(res)) {
          return res.patient;
        } else {
          // Handle cache miss
        }
      }
  };
}());
```

JavaScript

```
JW.cache = (function(){
  var cache = {},
      _isValid = function(res) {

      }
  return {
    getPatient:
      function(personnummer) {
        var res=cache[personnummer];
        if(_isValid(res)) {
          return res.patient;
        } else {
          // Handle cache miss
        }
      }
```

JavaScript

```
JW.cache = (function(){
  var cache = {},
      _isValid(res) {
        return !res ? false :
          (Date.now() - res.timestamp)
          <= 60000;  // One minute
      }
  return {
    getPatient:
      function(personnummer) {
        var res=cache[personnummer];
        if(_isValid(res)) {
          return res.patient;
        } else {
          // Handle cache miss
```

JavaScript

```
JW.cache = (function(){
  var cache = {},
      _isValid(res) {
        return !res ? false :
          (Date.now() - res.timestamp)
          <= 60000;  // One minute
      }
return {
  getPatie
    function(personnummer) {
      var res=cache[personnummer];
      if(_isValid(res)) {
        return res.patient;
      } else {
        // Handle cache miss
```

We can have (closure) private variables and functions

JavaScript

```
Crockford = (function(initParam) {
  var privVar1, privVar2,
      _privFunc1 = function() {

      },
      _privFunc2 = function() {

      };
  return {
    pubVar1: ”value”,
    pubFunc1: function() {

    }
  };
}(initParam));
```

Part 6:

Named Parameters

Method Signatures

Java

```
public void execPurchase(int, String,
String, boolean, User)
```

JavaScript

```
function execPurchase(price, item,
discountCode, wantsSpam, user)
```

Method Signatures

Java

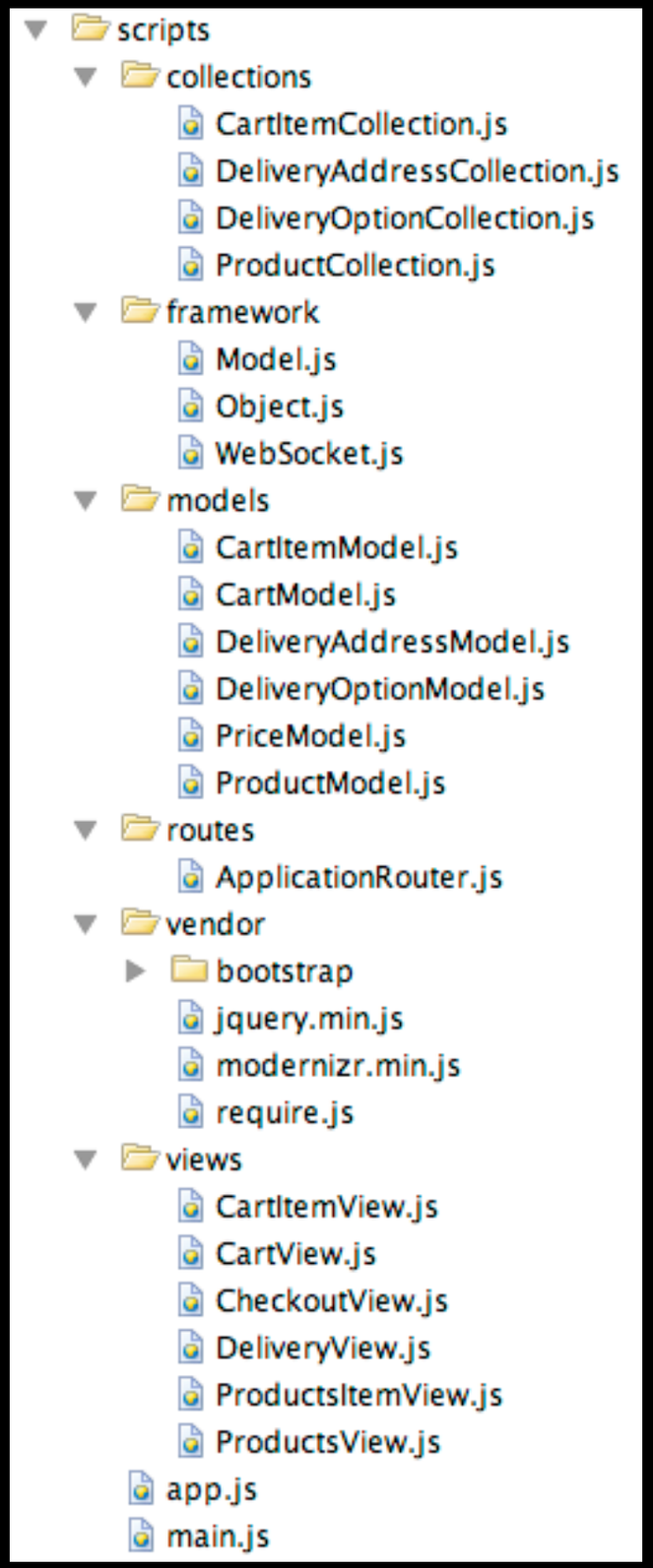
```
builder = new Purchase.Builder();
Purchase purchase =
      builder.setPrice(9900)
       .setItem(”cap”).wantsSpam(false)
       .setUser(user).build();
```

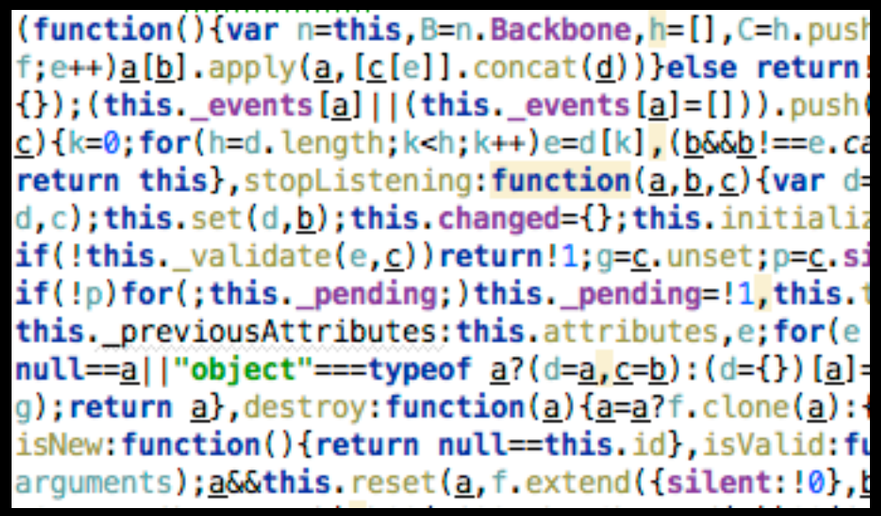
JavaScript

```
function execPurchase(purchase) {
  purchase.price …
  purchase.item …
  purchase.wantsSpam …
  purchase.user …
```

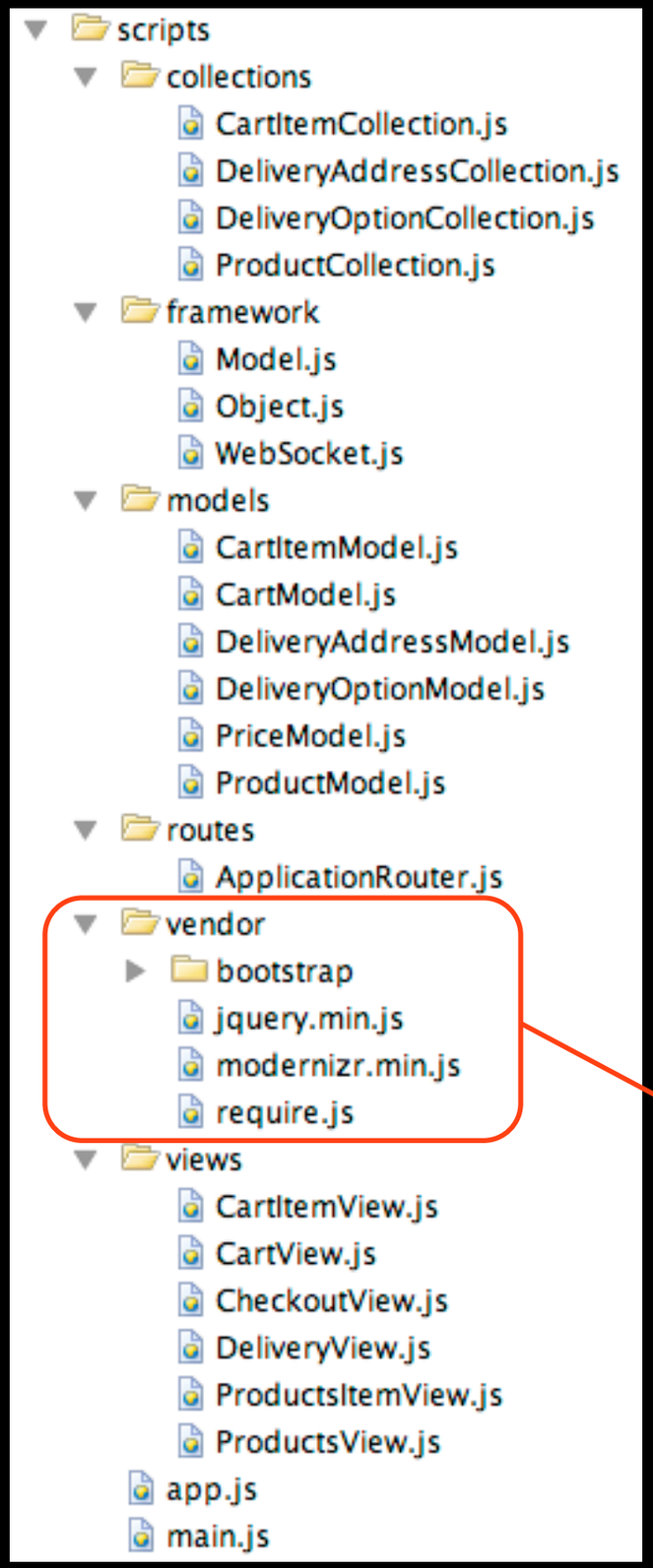
Part 7: Asynchronous Module Definition (AMD) with require.js

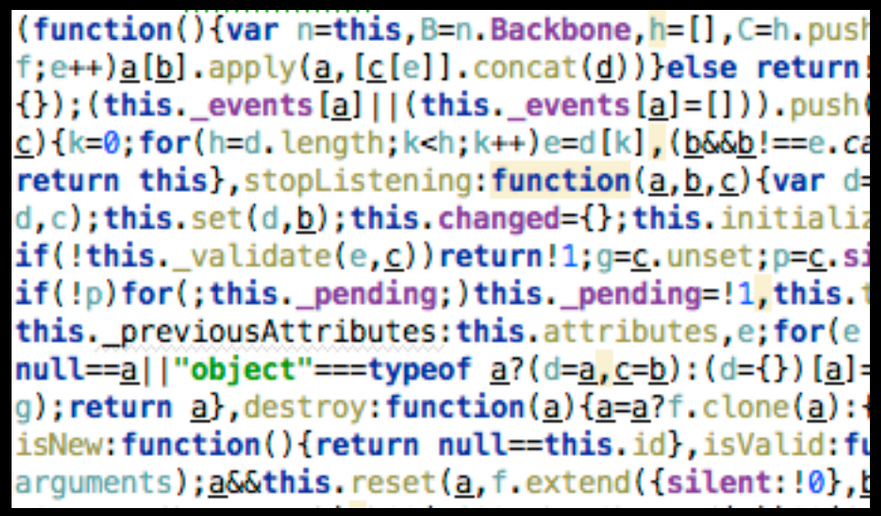
Concatenate and minify





Concatenate and minify





You may want to separate third party code to be able to cache it harder.

Concatenate and minify

- scripts
  - collections
    - CartItemCollection.js
    - DeliveryAddressCollection.js
    - DeliveryOptionCollection.js
    - ProductCollection.js
  - framework
    - Model.js
    - Object.js
    - WebSocket.js
  - models
    - CartItemModel.js
    - CartModel.js
    - DeliveryAddressModel.js
    - DeliveryOptionModel.js
    - PriceModel.js
    - ProductModel.js
  - routes
    - ApplicationRouter.js
  - vendor
    - bootstrap
    - jquery.min.js
    - modernizr.min.js
    - require.js
  - views
    - CartItemView.js
    - CartView.js
    - CheckoutView.js
    - DeliveryView.js
    - ProductsItemView.js
    - ProductsView.js
  - app.js
  - main.js

main.js

```
(function(){var n=this,B=n.Backbone,h=[],C=h.push
f;e++)a[b].apply(a,[c[e]].concat(d))}else return!
{});(this._events[a]||(this._events[a]=[])).push(
c){k=0;for(h=d.length;k<h;k++)e=d[k],(b&&b!==e.ca
return this},stopListening:function(a,b,c){var d=
d,c);this.set(d,b);this.changed={};this.initializ
if(!this._validate(e,c))return!1;g=c.unset;p=c.si
if(!p)for(;this._pending;)this._pending=!1,this.
this._previousAttributes:this.attributes,e;for(e
null==a||"object"===typeof a?(d=a,c=b):(d={})[a]=
g);return a},destroy:function(a){a=a?f.clone(a):
isNew:function(){return null==this.id},isValid:fu
arguments);a&&this.reset(a,f.extend({silent:!0},b
```

The goal is to mimimize the number of HTTP requests and the size of data over the wire.

Tools for Concatenation and Minification

- Google Closure Compiler
- UglifyJS
- require.js
  Concatenates and minifies with Closure Compiler or UglifyJS

require.js – import and namespace in JavaScript

@johnwilander

```
require([
    "backbone",
    "underscore",
    "models/CartModel",
    "text!templates/CartItem.html"],
    function (Backbone, _, CartModel,
              cartItemTemplate) {

    // Code dependent on Backbone etc

});
```

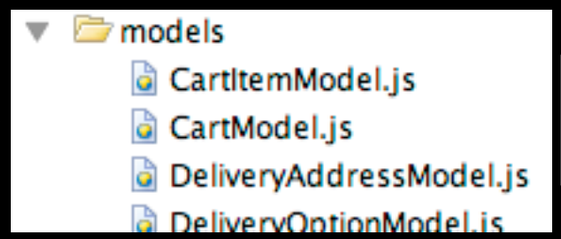
```
paths : {
  jquery : "vendor/jquery.min",
  backbone : "../components/backbone/backbone-min",
  underscore : "../components/underscore/underscore"
}
```

main.js

```
require([
  "backbone",
  "underscore",
  "models/CartModel",
  "text!templates/CartItem.html"],
  function (Backbone, _, CartModel,
            cartItemTemplate) {

  // Code dependent on Backbone etc

});
```



File system, like Java packages

```
require([
    "backbone",
    "underscore",
    "models/CartModel",
    "text!templates/CartItem.html"],
    function (Backbone, _, CartModel,
              cartItemTemplate) {

    // Code dependent on Backbone etc

});
```

```
require([
    "backbone",
    "underscore",
    "models/CartModel",
    "text!templates/CartItem.html"],
    function (Backbone, _, CartModel,
              cartItemTemplate) {

    var model = new CartModel();

});
```

Local name, like *dependency injection*

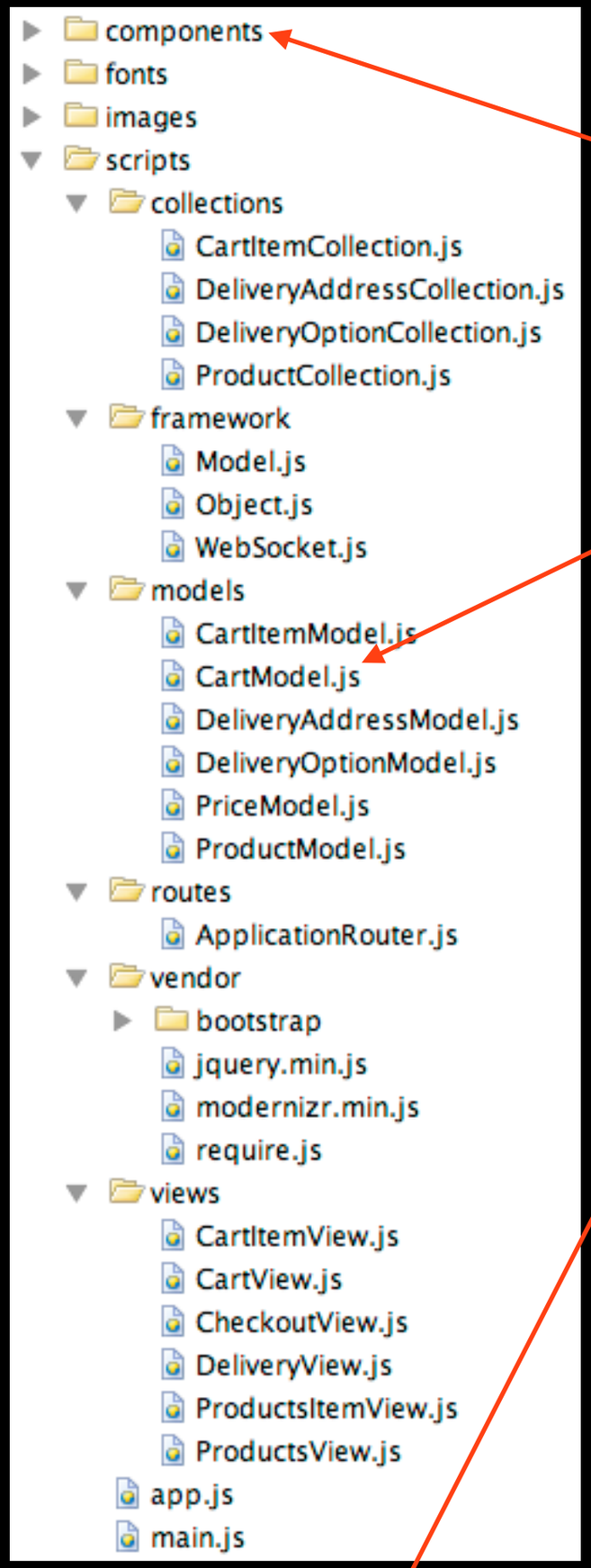
Define new modules

```
define([
    "backbone",
    "underscore",
    "models/CartModel",
    "text!templates/CartItem.html"],
    function (Backbone, _, CartModel,
              cartItemTemplate) {

    var CartItemView = …;

    …

    return CartItemView;
});
```



CartItemView.js

```
define([
    "backbone",
    "underscore",
    "models/CartModel",
    "text!templates/CartItem.html"
], function (Backbone, _, CartModel, cartItemTemplate) {
    "use strict";

    var CartItemView = Backbone.View.extend({

        tagName : "li",

        events : {
            "click .icon-trash" : "removeFromCart"
        },

        initialize : function () {
            this.template = _.template(cartItemTemplate);
            this.render();
        },

        render : function () {
            this.$el.empty();
            this.$el.html(this.template(this.model.toJSON()));
        },

        removeFromCart : function () {
            CartModel.globalCart.removeFromCart(this.model);
        }
    });

    return CartItemView;
});
```

index.html without require.js or the like

```
<head>
  <script src="js/lib/jquery-1.9.0.min.js"></script>
  <script src="js/lib/jquery-encoder-0.1.0.js"></script>
  <script src="js/base.js"></script>
  <script src="js/JW/util/util.js"></script>
  <script src="js/JW/personnummer/Personnummer.js"></script>
  <script src="js/JW/patient/Patient.js"></script>
  …
  <script src="js/JW/cache/cache.js"></script>
  <script src="js/JW/proxy/proxy.js"></script>
  <script src="js/JW/gui/gui.js"></script>
</head>
```

… manually, in the right order.

index.html with require.js

```
<head>
  <script data-main="scripts/main" src="scripts/require.js" >
  </script>
</head>
```

index.html with require.js

```
<head>
  <script data-main="scripts/main" src="scripts/require.js" >
  </script>
</head>
```

```
require.config({
  …
  paths : {
    jquery : "vendor/jquery.min",
    backbone : "../components/backbone/backbone-min",
    underscore : "../components/underscore/underscore"
  }
});

require(["app"], function (App) {
  App.start();
});
```

main.js

Reference Documentation

Always
developer.mozilla.org

Never
www.w3schools.com

Why?
w3fools.com

The Stream

http://javascriptweekly.com/

@javascript_news
@BrendanEich @littlecalculist
@addy_osmani @addyosmani
@paul_irish @badass_js
@rwaldron @slicknet
@kangax @unscriptable
@sthlmjs

1.8

Dojo Toolkit 1.8

Dojo saves you time and scales with your development process, using web standards as its platform. It's the toolkit
desktop and mobile web app

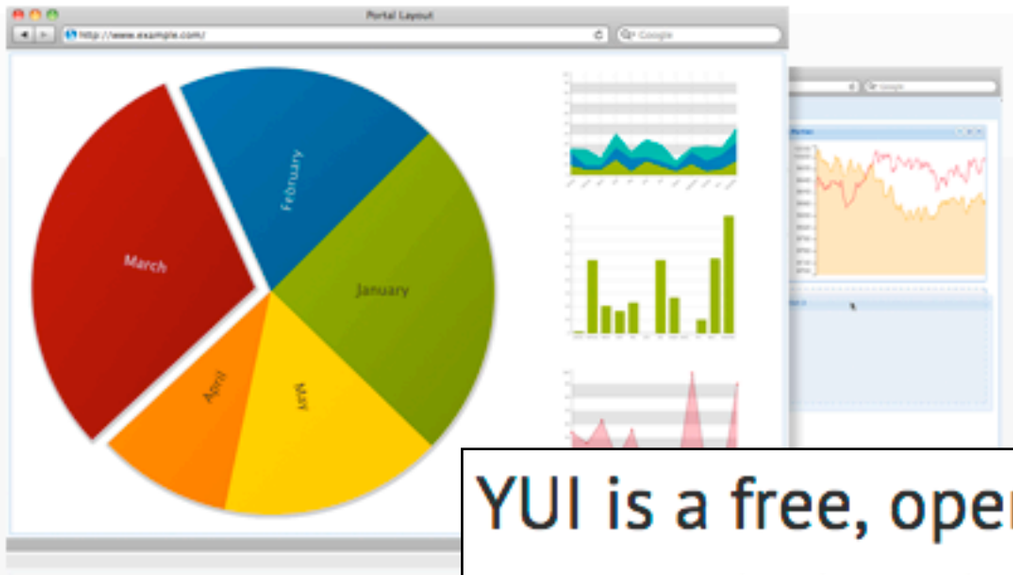
From simple websites to large
mobile, Dojo will meet your needs.

dojo-release-1.8.3.zip 11 Mb

Home / Products

Sencha Ext JS JavaScript Framework for Rich Apps in Every Browser

Buy Download



The
Web Application Development

With an advanced MVC architecture, plugin-free charting, and modern UI widgets, Sencha Ext JS is the industry's most powerful desktop application development platform. Sencha

ext-4.1.1a-gpl.zip 45,7 Mb
ext-all.js 1 Mb

YUI is a free, open source JavaScript and CSS library for building richly interactive web applications.

yui_3.8.1.zip 28,1 Mb

Get Started

Start using YUI 3.8.1 in two easy steps.

Simplified
Dom api
& shims

Templates

Widgets

MVC

@joakimkemeny & @johnwilander

underscore

Modernizr

jQuery

Simplified
Dom api
& shims

Templates

Widgets

MVC

@joakimkemeny & @johnwilander

Handlebars

Mustach

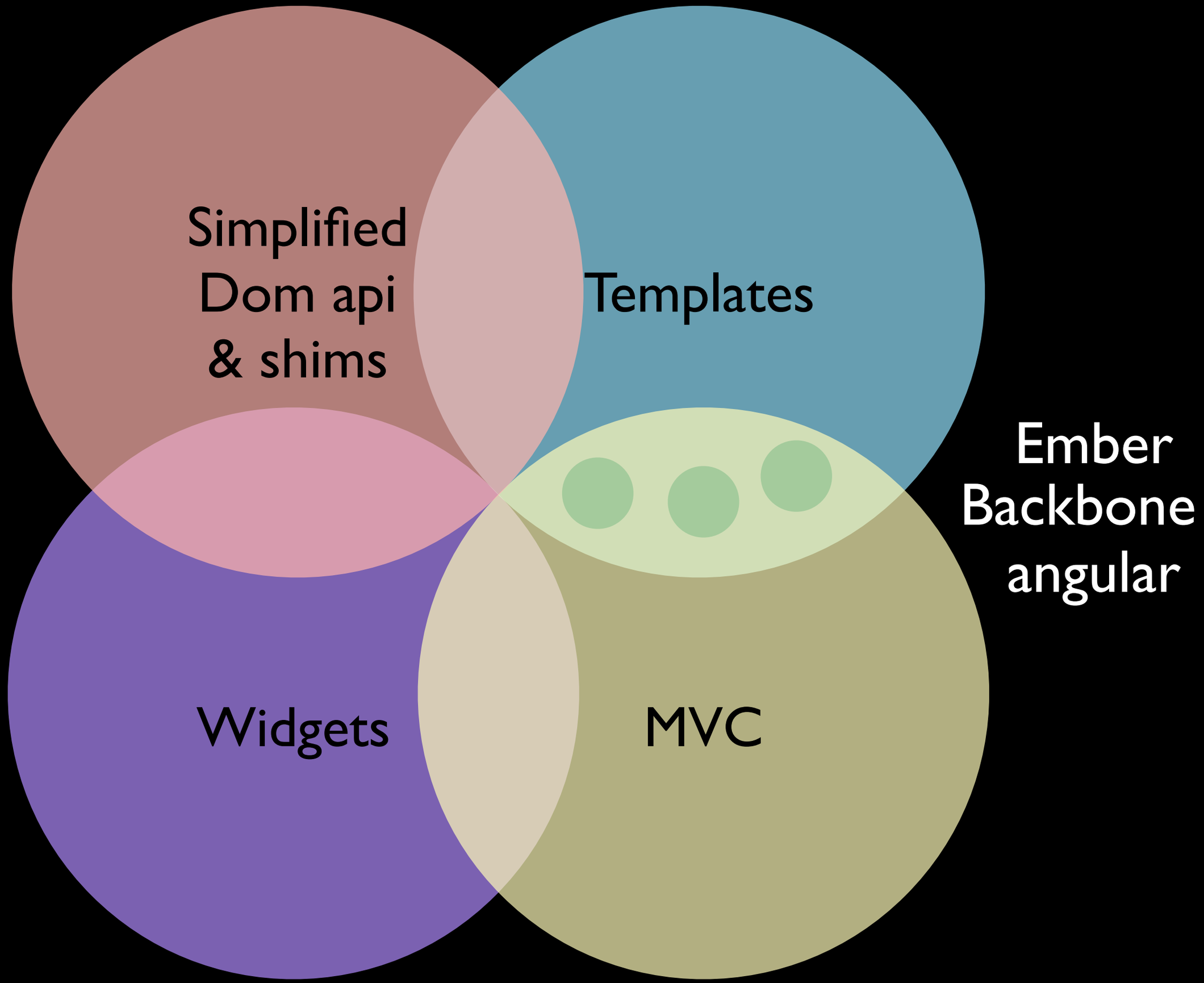
Simplified
Dom api
& shims

Templates

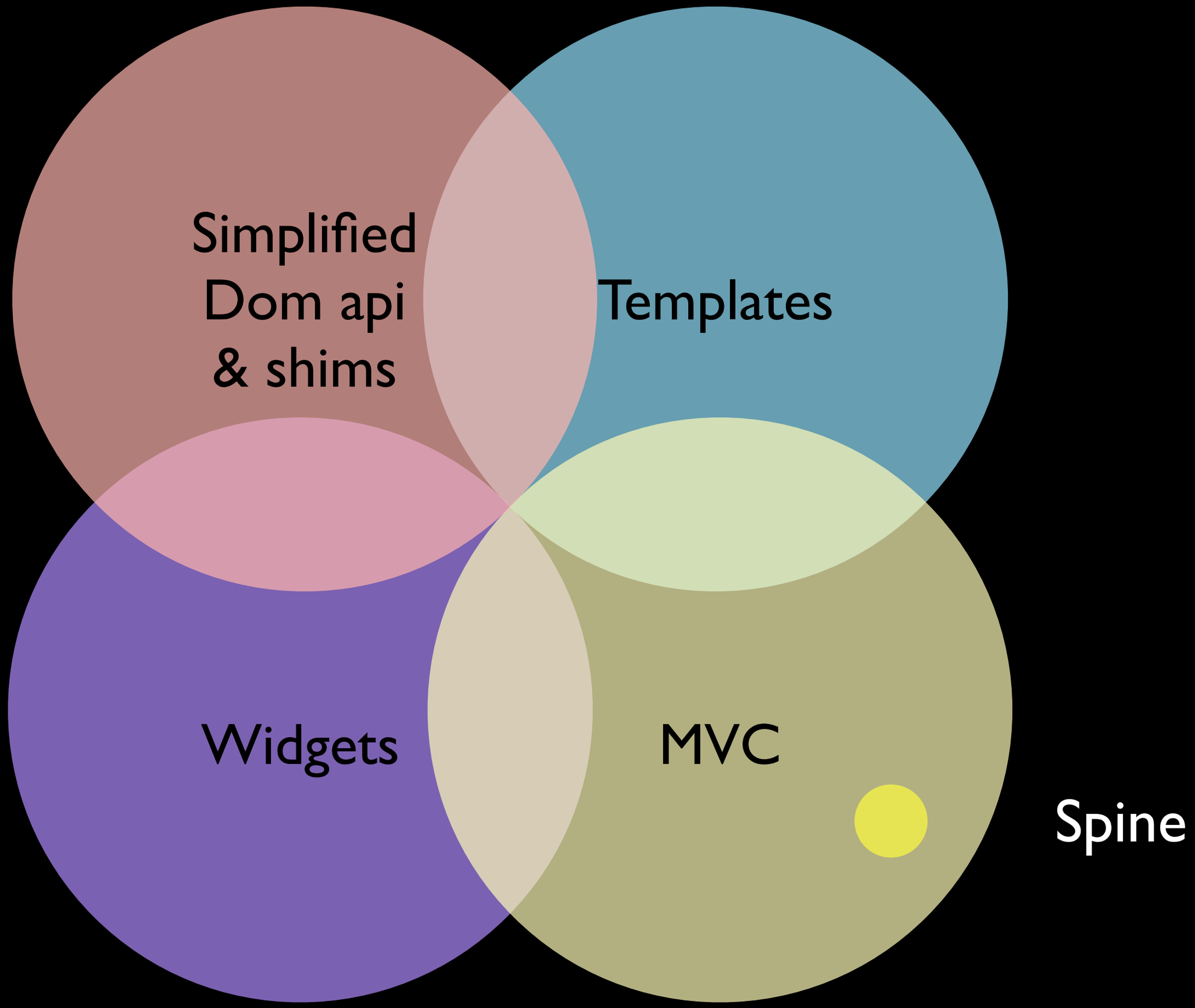
Widgets

MVC

@joakimkemeny & @johnwilander



@joakimkemeny & @johnwilander


Simplified
Dom api
& shims
Templates
Widgets
MVC
Spine

Simplified
Dom api
& shims

Templates

Widgets

MVC

Bootstrap

jQuery UI

Simplified
Dom api
& shims

Templates

Ext JS
Dojo
YUI

Widgets

MVC

@joakimkemeny & @johnwilander

All-in-one frameworks are best suited for newly written single-purpose applications without *mashup* behavior.

Micro frameworks are
better if you need to
integrate legacy code, some
nifty CMS product, or if you
have logic/content from
third parties.

Micro frameworks don't
”take over” you application.