# Creating Extensions, Plugins and Components



# Agenda

- Learn about the new class system
  - Ext.class.Base
  - Explore Mixins
  - Ext.Loader
- Discuss Component Life Cycle
- Create a simple extension
- Plugins



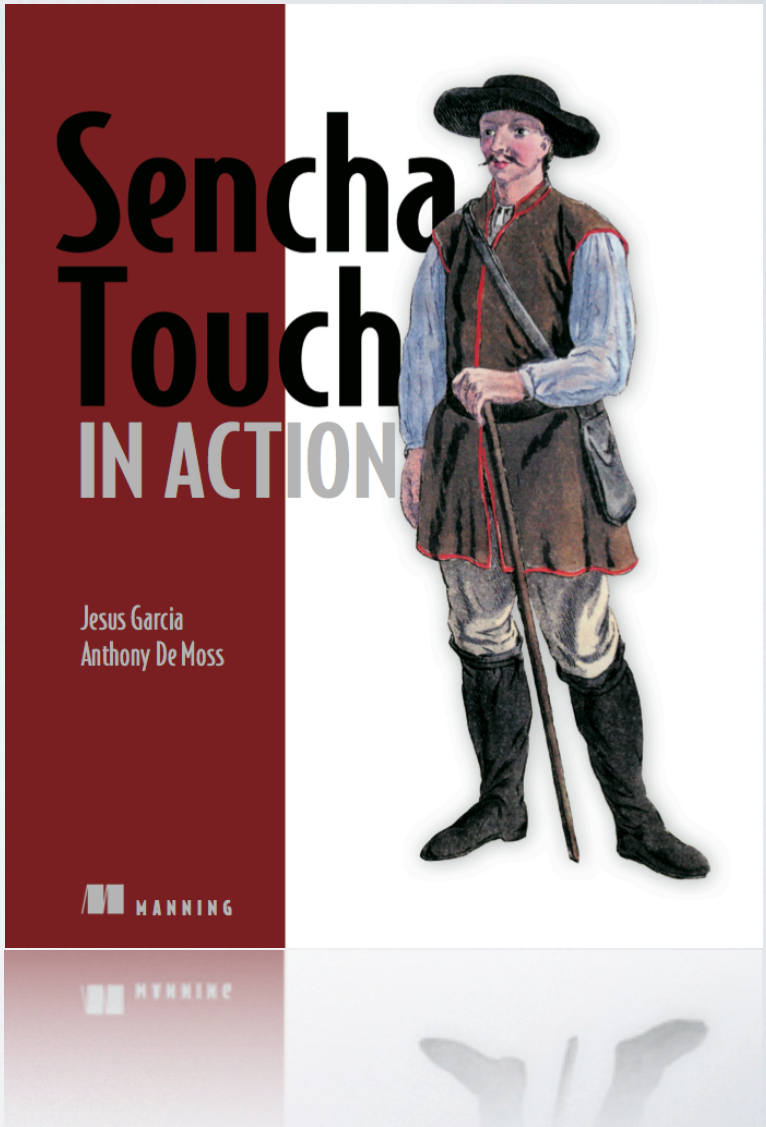
# What have I been up to?

- Bootstrapping a new company, Modus Create
  - Focusing on architecting and building **kick-ass** Ext JS and Sencha Touch apps!
  - Regional and open training on Advanced JavaScript, Ext JS 4 and Sencha Touch
  - ***Rapidly*** expanding and looking for experienced developers (Sencha Touch and Ext JS)
- http://moduscreate.com

Modus

# On the book front...

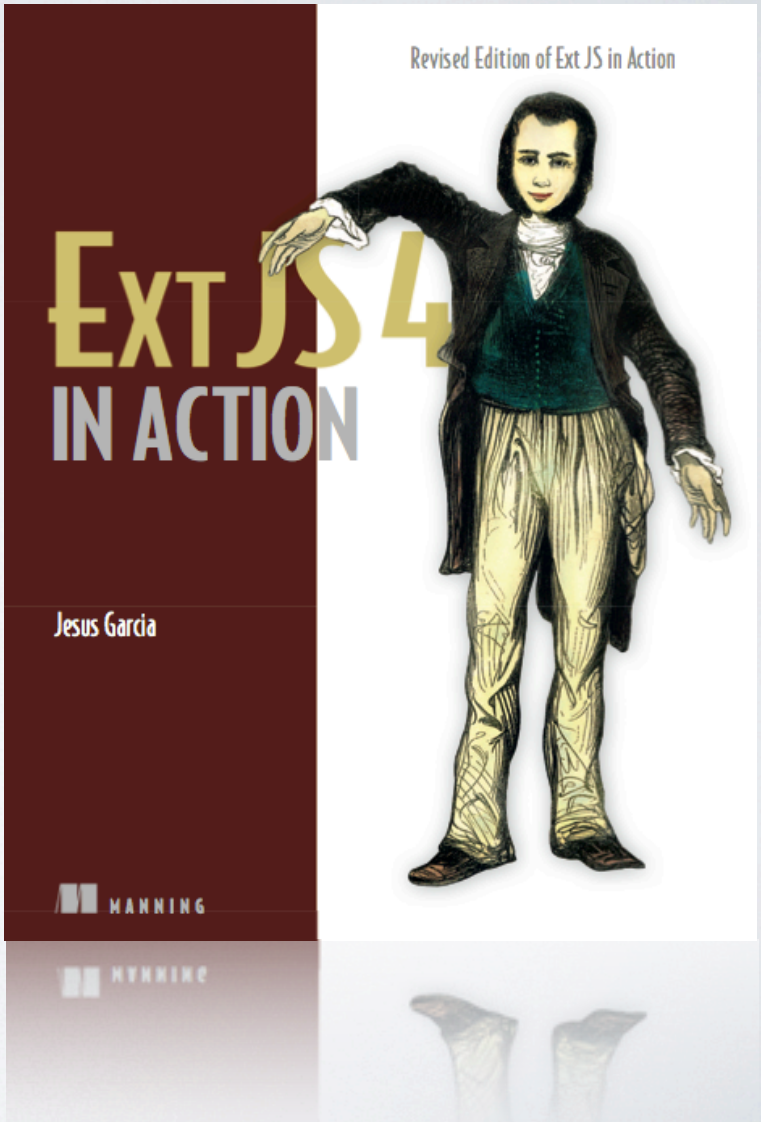
- Early access to Sencha Touch in Action available.
- Chapters 1-3 being distributed
- Chapter 7 is next to be published
- Chapter 4 will be published in about 2 weeks
- Chapters 5 and 6 are being worked on




Modus

# On the book front...

- Code for Chapter 1 is being worked on.
- Expect TOC revisions to cover the latest changes to Ext JS
- Anyone interested in working with me on this project?




Modus

# Ext JS 4.0 Class System

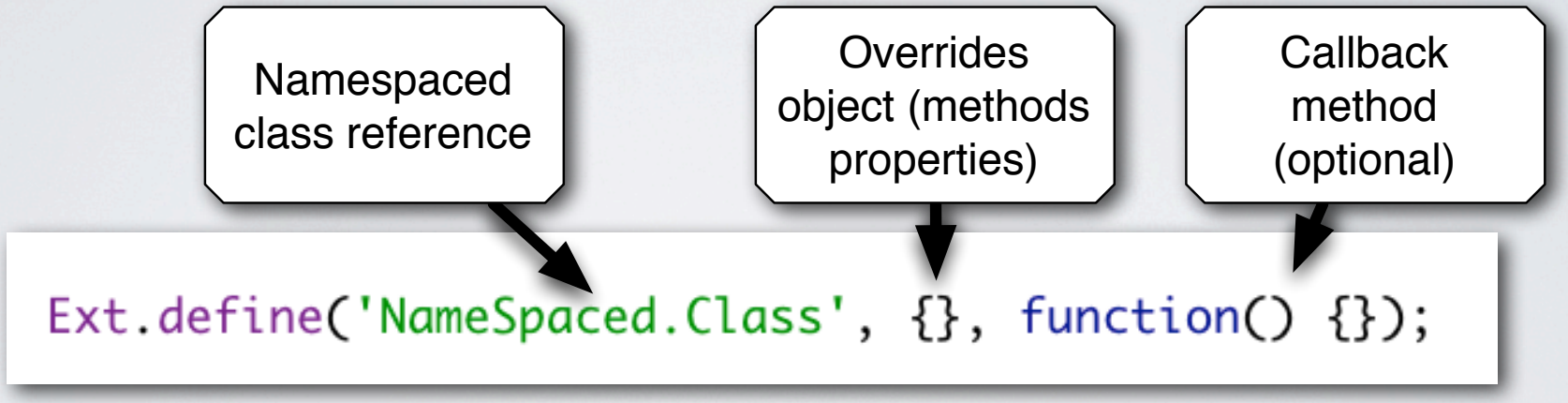
- Completely revised from 3.0

# Ext.Base features

- Automatic Namespace management
- Statics support
- Mixin Support
- Alias management (XTypes)
- Built-in override/extension methods
- Plays nicely with the dynamic class Loader system
- Alternate class name (good for deprecation management)



# Ext.define pattern



# Ext.define example

```
Ext.define('MyApp.MyClass', {
    sayHi : function() {
        console.info('Hi!!!');
    }
}, function(MyClass) {
    Ext.create('MyApp.MyClass').sayHi();
});
```

# Auto-generated setters/getters

```
Ext.define('MyApp.MyClass', {
    config : {
        name : 'Sam',
        age  : 21
    },
    constructor : function(cfg) {
        Ext.apply(this, cfg ||{});
        this.callParent();
    },
    sayHi : function() {
        console.info(this.name, 'says hi!!!')
    }
});

var person = Ext.create('MyApp.MyClass');

person.setName('Jay');
person.setAge(32);
person.sayHi();
```

Setters and getters auto-generated!

Use auto-generated setters!

# Ext JS 4 Class statics

- Provides scope-independent access to static members.
- Applied before constructor is called

# Statics example

```
Ext.define('MyApp.MyClass', {
    statics : {
        myName : 'person Garcia',
        age    : 32
    },

    constructor : function() {
        var statics = this.statics(),
            name    = statics.myName,
            age     = statics.age;

        console.log(name , 'says hi!!!');
        console.warn(name, 'is old (', age, ')!');

        this.callParent();
    }
});

Ext.create('MyApp.MyClass');
```

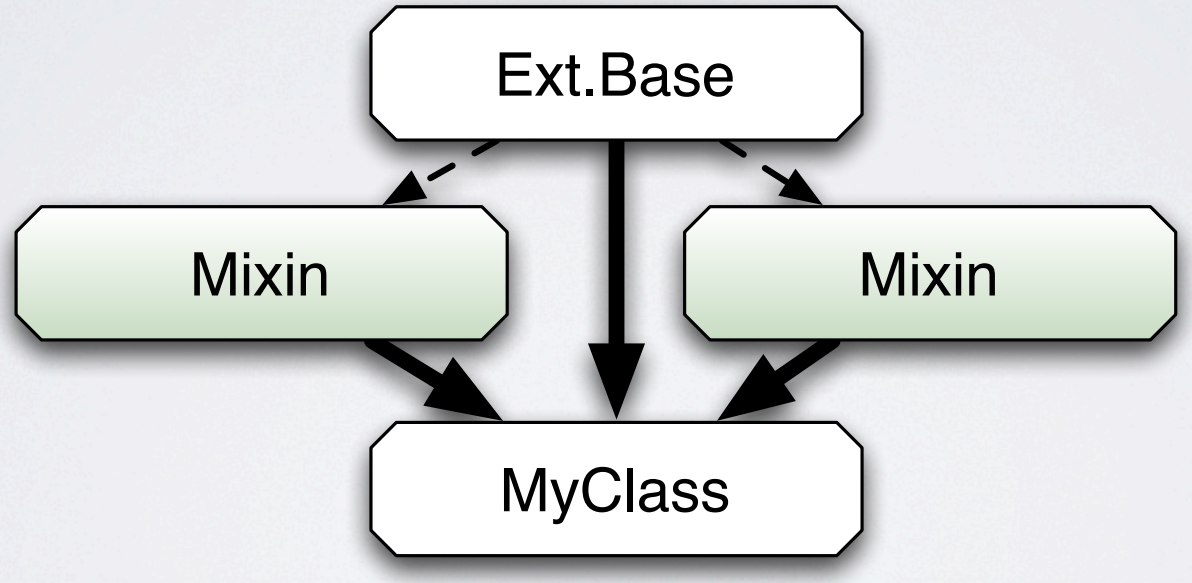
Configure statics

Access statics



# Ext JS 4 Class Mixin support

- Provides a means for extremely easy multiple inheritance



# Examples of Mixins inside of Ext JS 4

- General
  - Observable
  - State
  - Draggable
  - Floating
- chart
  - Tips
  - Label
  - Callout
- fx
  - Queue
  - Animate
- data
  - sortable
- form
  - Field Ancestor
  - Labelable



# Example Mixin class # 1

```
Ext.define('DrivingMixin', {
    constructor : function() {
        this.isDriver = true;

        this.callParent();
    },
    drive : function(){
        this.steer();
        this.shift();
        this.brake();
    },
    steer : function() {
        console.info(this.name, 'is steering!');
    },
    shift : function() {
        console.info(this.name, 'is shifting!');
    },
    brake : function() {
        console.warn(this.name, 'is BRAKING!');
    }
});
```



# Example Mixin class #2

```
Ext.define('PilotingMixin', {
    pilot : function() {
        this.rollLeft();
        this.doImmelman();
        this.land();
    },
    rollLeft : function() {
        console.info(this.name, 'is rolling left!');
    },
    doImmelman : function() {
        console.info(this.name, 'is doing an Immelman turn!');
    },
    land : function() {
        console.log(this.name, 'touched down!');
    }
});
```

# Implementing the Mixins

```
Ext.define('Person', {
    mixins : {
        driving  : 'DrivingMixin',
        piloting : 'PilotingMixin'
    },
    constructor : function(cfg) {
        Ext.apply(this, cfg || {});
        this.mixins.driving.constructor.call(this);

        this.callParent();
    },
    walk : function() {},
    talk : function() {},
    drive : function() {
        console.log(this.name, 'is driving');
        this.mixins.driving.drive.call(this);
    }
});
```

Include Mixins

Execute Mixin constructor

Execute Mixin method

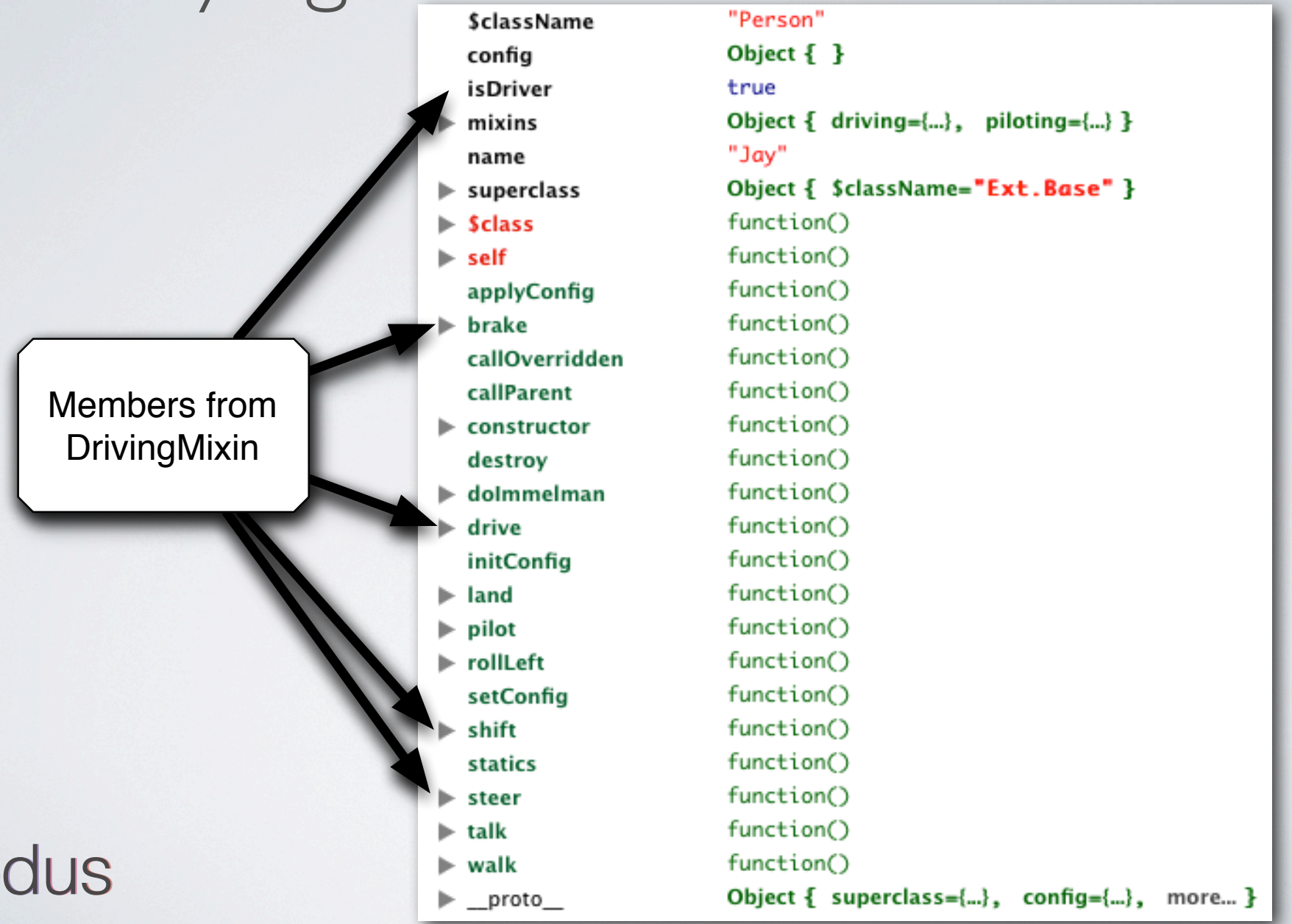
# Implementing the Mixins

Mixin instances


```
new Person({ name : 'Jay' });
```

Modus

# Identifying the cross inheritance



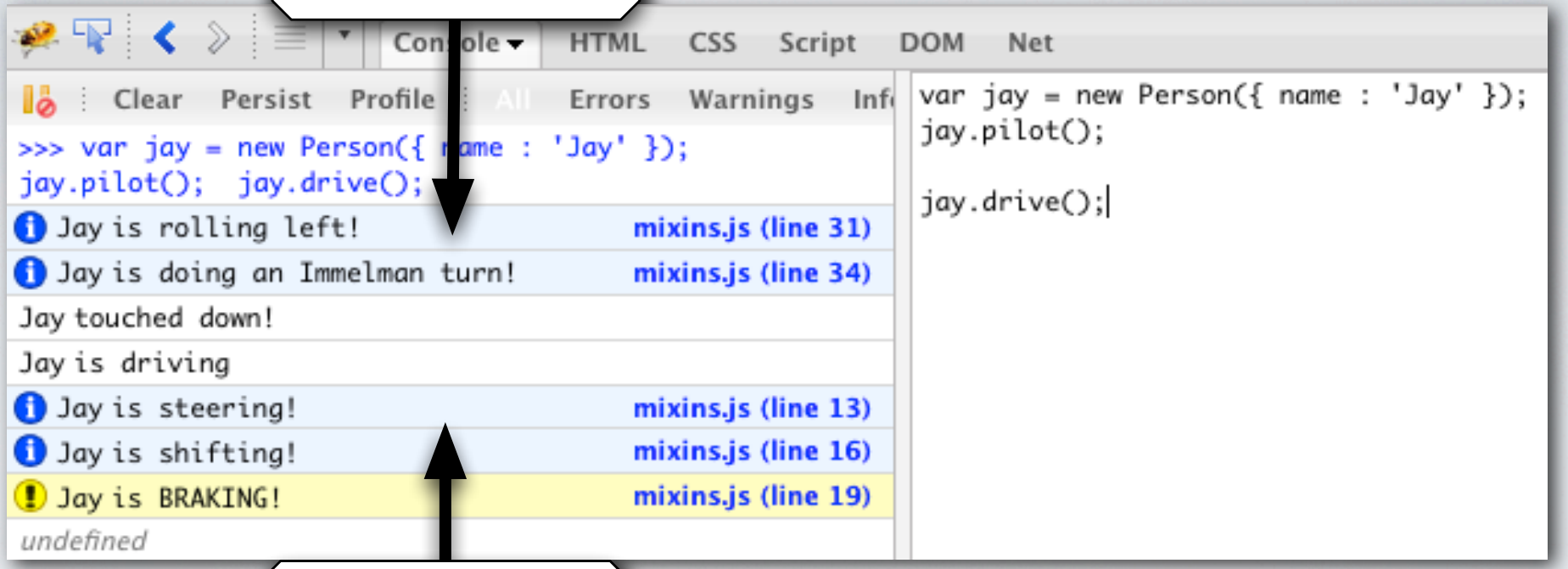

Modus

# Identifying the cross inheritance


Modus

# Exercising the mixins

Output from PilostingMixin




Output from DrivingMixin

Modus

# Ext JS 4 Class Loader

- Provide on-demand loading of classes
  - Synchronous and Asynchronous
- Built-in dependency management
- Can be used for 99% of your application!
- Ridiculously easy to use!
- Less HTML to write
- Has some pitfalls, which we'll pepper in our discussions



# First step

- Load **only** ext-all.css **and** ext.js or ext-debug.js

```
<title>Lazy Loading components</title>
<link rel="stylesheet" type="text/css" href="../../js/ext4/resources/css/ext-all.css"/>
<script type="text/javascript" src="../../js/ext4/ext-debug.js"></script>
<!--<script type="text/javascript" src="../../js/ext4/ext-all-debug.js"></script>-->
<script type="text/javascript" src="simple1.js"></script>
```

# Second step

- Stop using the ***new*** keyword!


- Instead use Ext.create

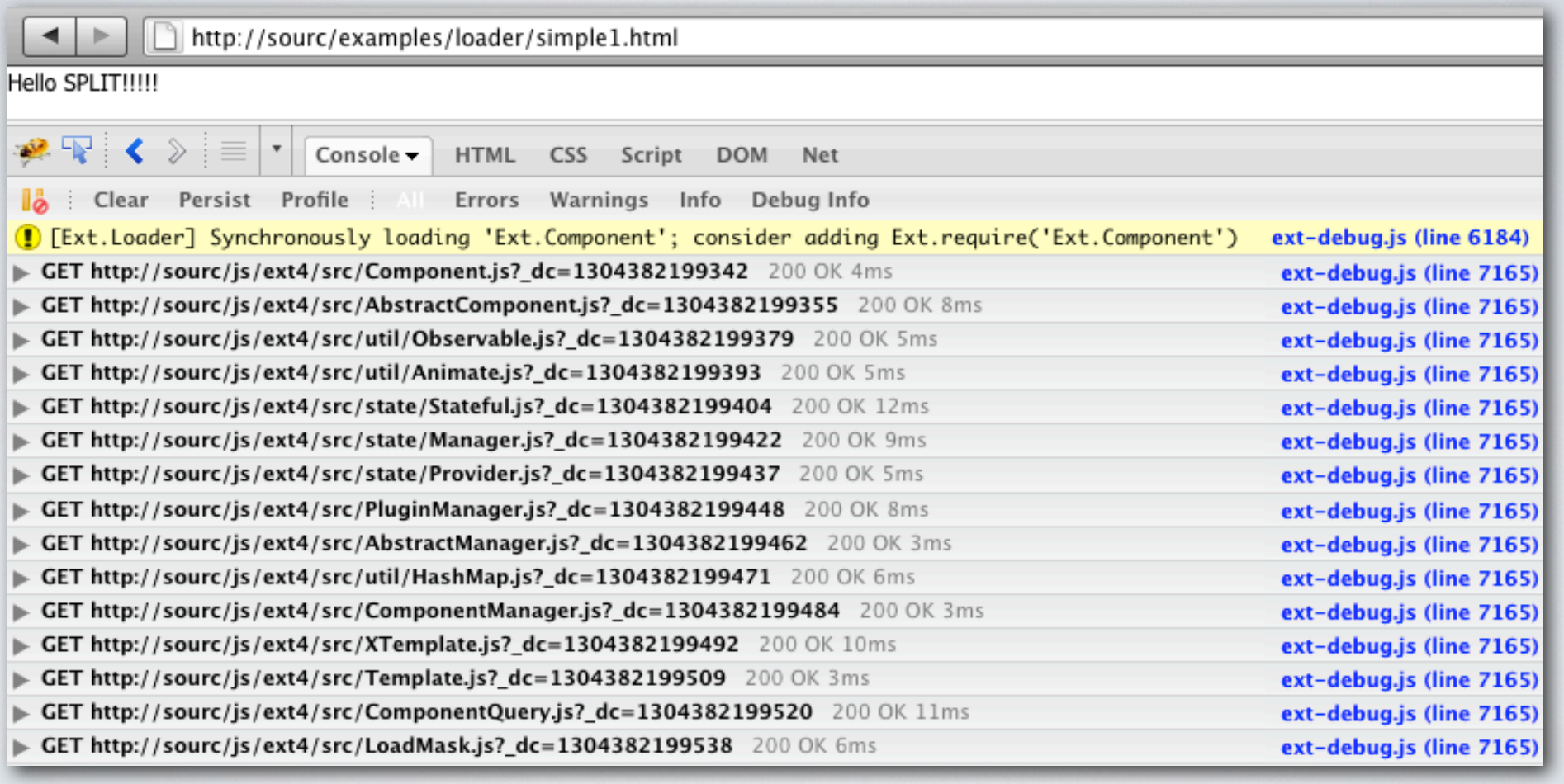
```
Ext.create('Ext.Component', {

});
```

# Simple example

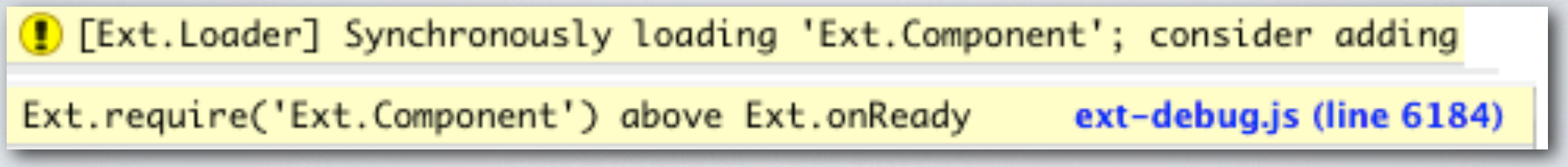
- Without the use of Ext.require, classes will load **synchronously!**

```
Ext.onReady(function() {
    Ext.create('Ext.Component', {
        width    : 100,
        html     : 'Hello SPLIT!!!!!',
        renderTo : Ext.getBody()
    });
});
```

# Synchronous loading results



# Why the Ext JS warning?!



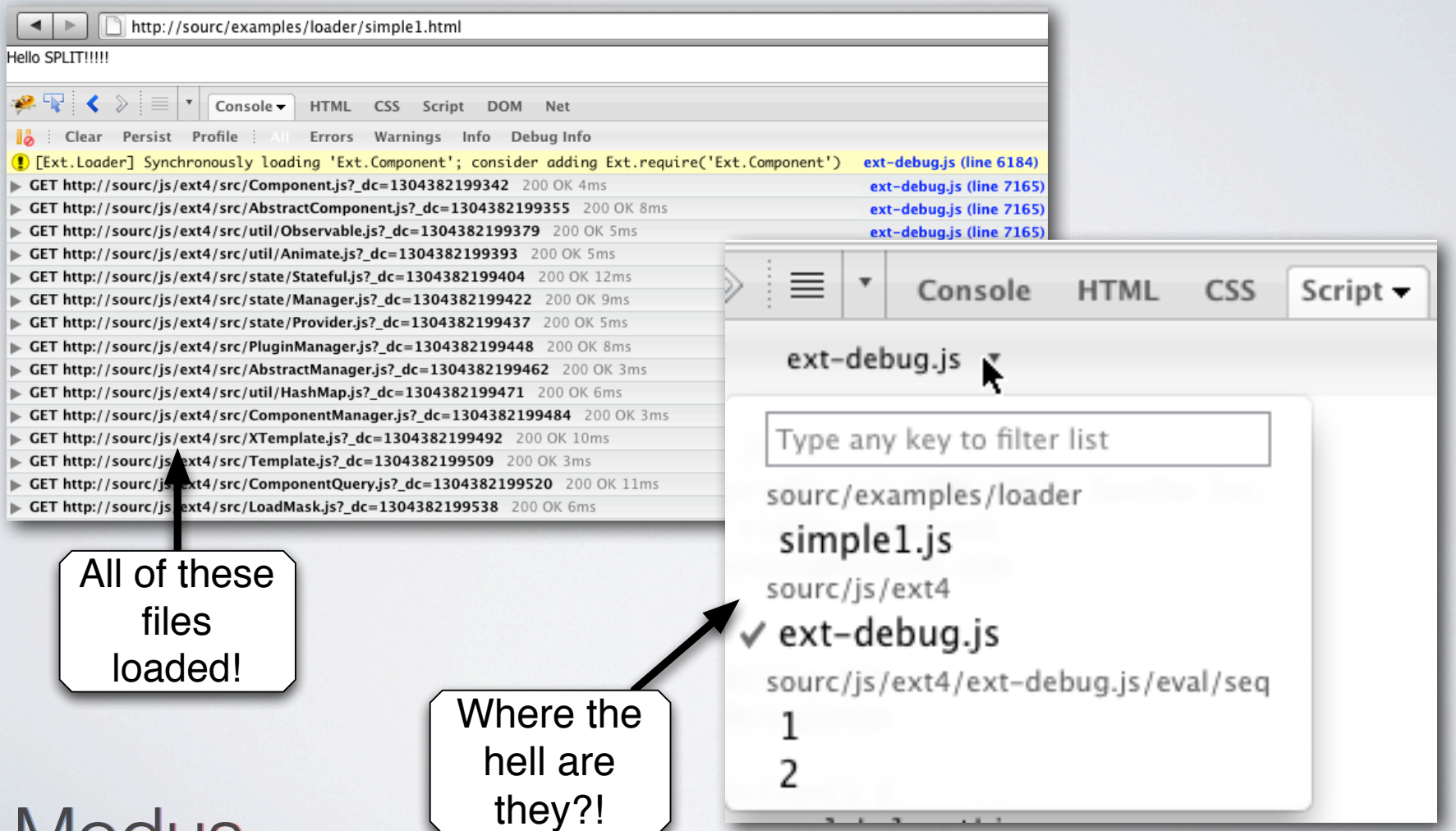

- Synchronous loading uses the XHR object
- Several disadvantages
  - (more than asynchronous loading!)

# Synchronous Loading disadvantages

- Uses the XHR object
  - Limited to same domain due to "same origin policy"
  - XHR requires a web server
  - Debugging is a nightmare!



# Debugging difficulty




Modus

# Why the ghost scripts?

- With synchronous loading, script tags are not added to the head, instead, they are *eval*'ed!

# Main advantage to synchronous loading is...

- Use of Ext.require is, well... not required.

```
Ext.onReady(function() {
    Ext.create('Ext.Component', {
        width    : 100,
        html     : 'Hello SPLIT!!!!!',
        renderTo : Ext.getBody()
    });
});
```

# Solution: use Ext.require

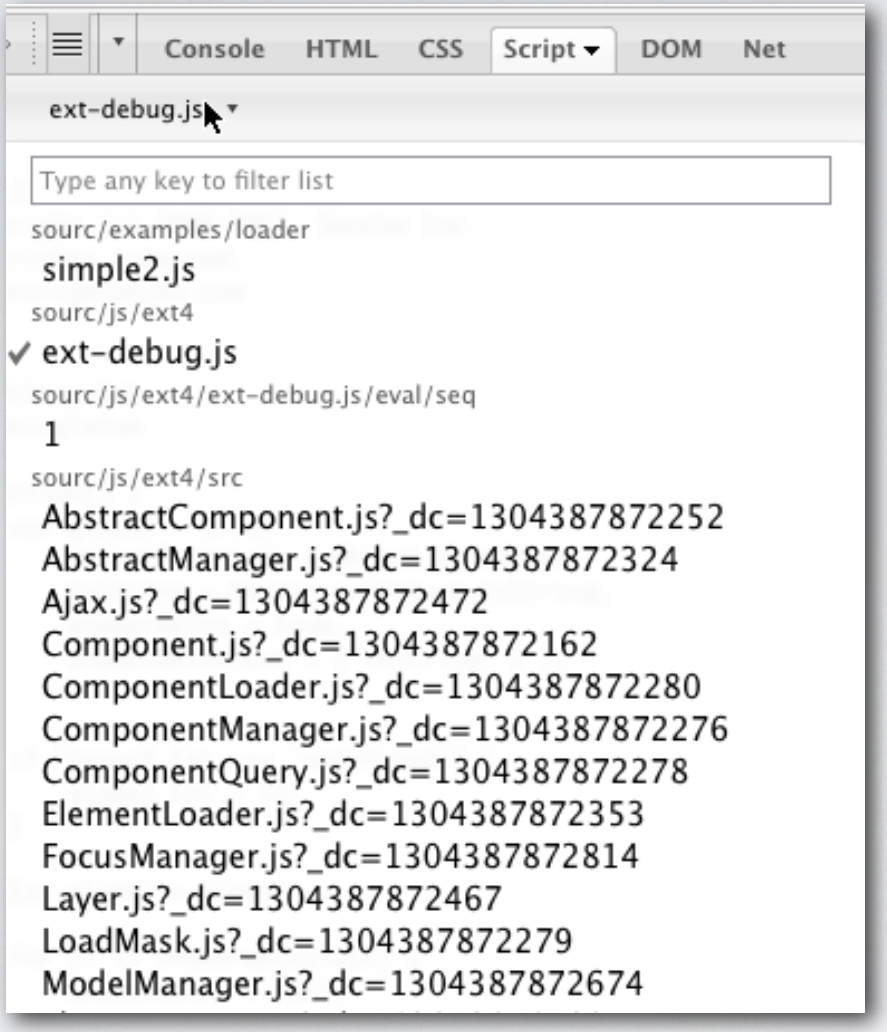
- Place Ext.require calls above Ext.onReady
- Ext.require will asynchronously load classes
  - Before "ready" state

```
Ext.require('Ext.Component');

Ext.onReady(function() {
    Ext.create('Ext.Component', {
        width    : 100,
        html     : 'Hello SPLIT!!!!!',
        renderTo : Ext.getBody()
    });
});
```

# Advantages of Asynchronous Loading

- Adds script tags to the HEAD of the document
- Debugging friendly
- Cross-domain compatible (does not use XHR)
- No web server required

# Using Ext.require makes Loader debugging friendly!




\:D/

Modus

{S}

# Main disadvantage for asynchronous loading

- Dependencies *must* be specified before the class is instantiated

```
Ext.require('Ext.util.Observable');

Ext.onReady(function() {
    var observable = Ext.create('Ext.util.Observable', {

    });

    console.log(observable)
});
```



# Overall disadvantages to using Ext Loader

- If at all costs, should not be used in production
  - Can drive your server NUTS with requests on each page load!
- Cache busting is either all on or all off
  - No way to configure cache busting on a per require or namespace basis



# Overall disadvantages to using Ext Loader cont.

- Can slow down page loads in firebug significantly for large slurps of classes (synchronous)
- No minified "package" loading.
- Unnecessary classes loaded at times
  - E.g. Ext.require('Ext.Component') slurps up Sprite from the draw package.
  - Requires that you exclude unnecessary items



# Overall advantages to using Ext Loader

- Lightning fast initial page loads relative to using ext-all.js!
- In ideal situations, the browser only consumes what is required
- Debugging is so much easier for the browser because each JS file is loaded separately
- No need to write HTML script tags

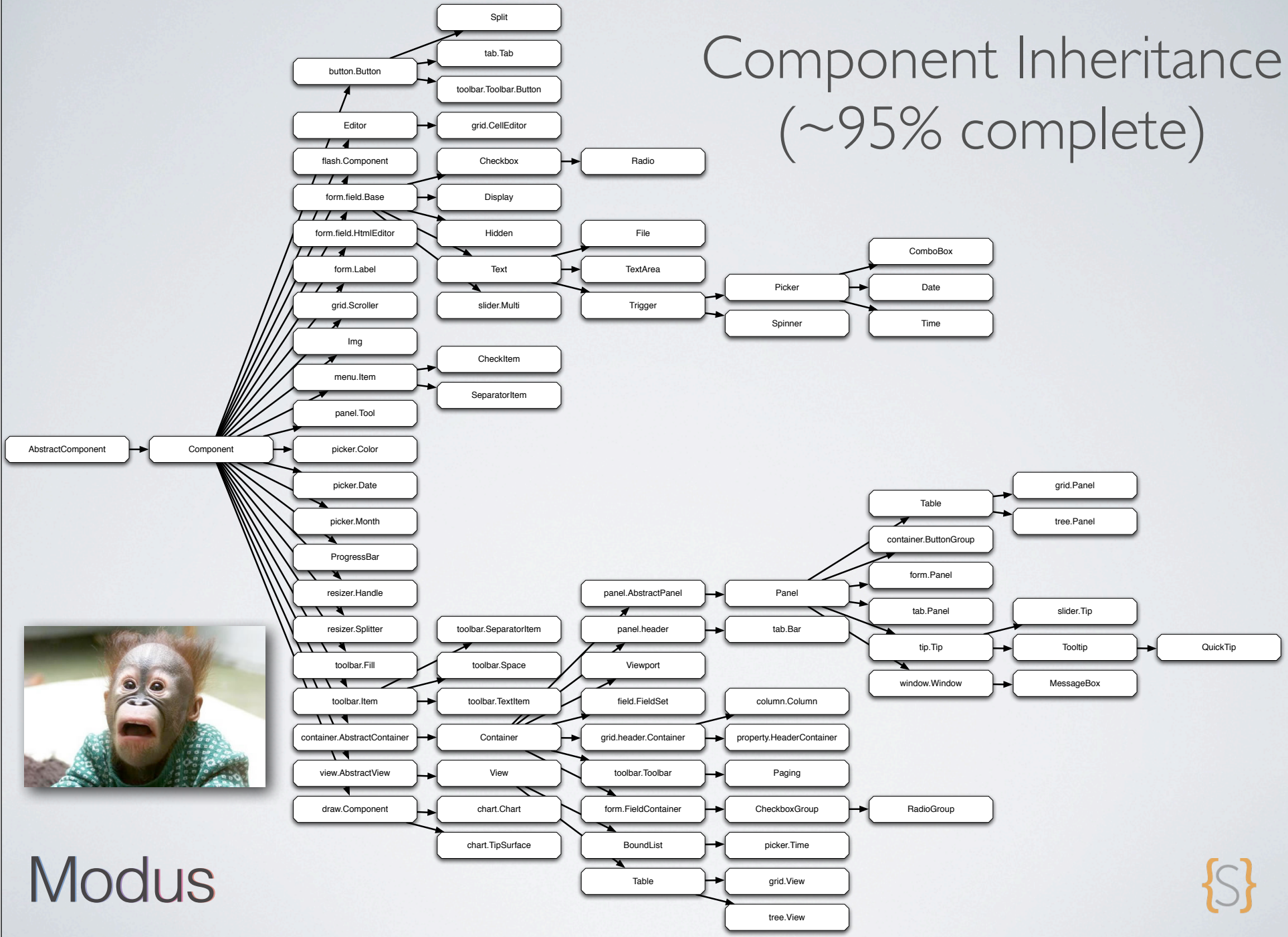


# Ext JS 4 Component Lifecycle

- Provided By Ext.component.AbstractComponent
- Abstracts out common behaviors to a central location
- Adds dependability and predictability to the framework
- ***Extremely important to know*** for both extending widgets and developing robust applications
- Is what ***makes Ext JS stand out*** from the rest of all other frameworks!



# Component Inheritance (~95% complete)

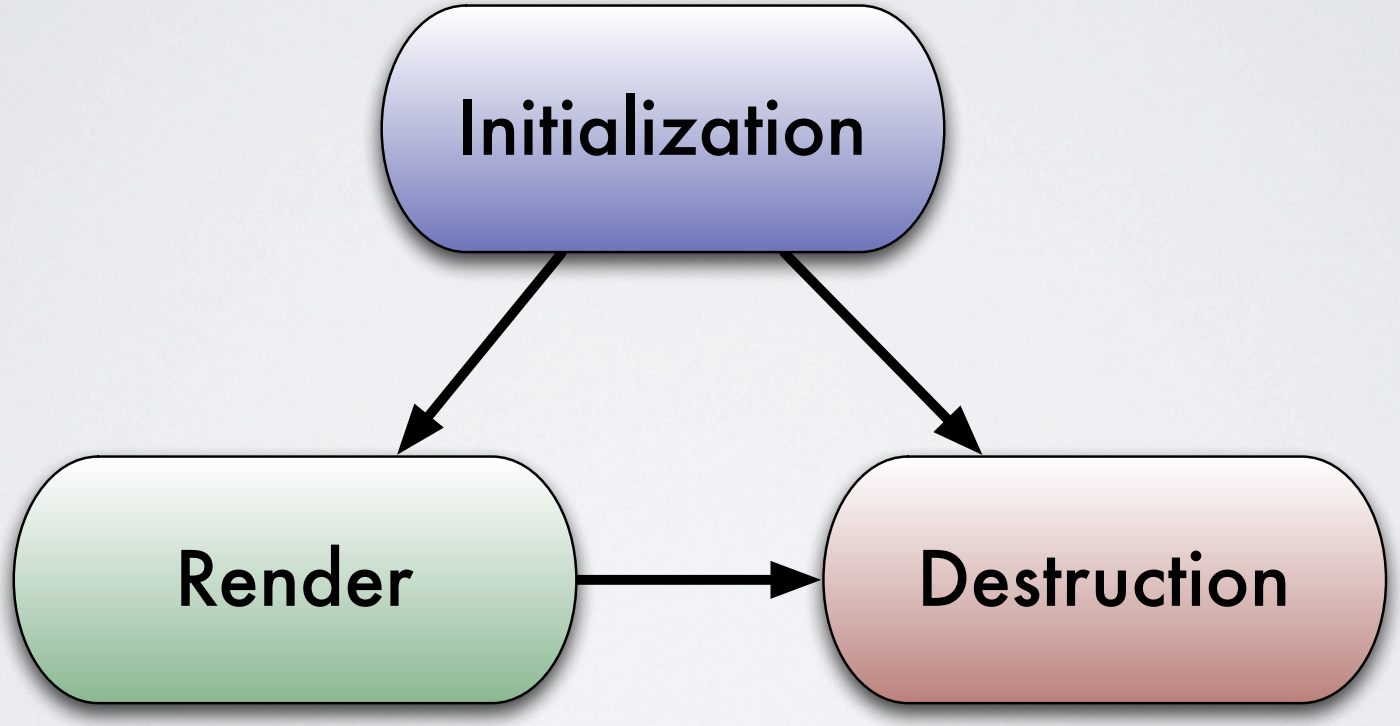





Modus

# Ext JS 4 Component Lifecycle

- Broken into three phases:



# Purposes of these steps

- Initialization
  - Bootstrap the Component (Create ID, register with ComponentMgr, etc).
- Render
  - Paint the Component on screen, hook element based events, use layouts to organize components
- Destruction
  - Wipe the Component from the screen, purge event listeners



# Initialization phase steps

1. Configuration object is applied and cached
2. Base events are added
    1. before activate, beforeshow, show, render, etc.
3. ID is assigned or auto generated
4. Plugins are constructed (think ptypes or aliases)



# Initialization phase steps (cont.)

5. initComponent is executed
    1. Custom listeners are applied
    2. ‘bubbleEvents’ are initialized
6. Component is registered with ComponentManager



# Initialization phase steps (cont.)

7. Base mixin constructors are executed
    1. Observable's constructor is called
    2. State's constructor is called
8. Plugins are initialized
9. Component Loader is initialized (not Ext.Loader!)
10. If configured, Component is rendered (renderTo, applyTo)



# Initialization phase steps (cont.)

11. If configured, Component is shown

12. Plugins are initialized

13. Component Loader is initialized (not Ext.Loader!)

14. If configured, Component is rendered (renderTo, applyTo)

15. If configured, Component is shown



# Render phase steps

1. 'beforerender' event is fired
2. Component's element is cached as the 'el' reference
3. If a floating Component, floating is enabled and registered with WindowManager
4. The Comonent's container element is initialized



# Render phase steps (cont.)

5. onRender is executed
    1. Component's element is injected into the DOM
    2. Scoped reset CSS is applied if configured to do so
    3. Base CSS classes and styles are applied
    4. "ui" is applied
    5. "frame" initialized



# Render phase steps (cont.)

5. onRender is executed (cont.)
    6. renderTpl is initialized
    7. renderData is initialized
    8. renderTpl is applied to Component's element using renderData
    9. Render selectors are applied
    10. "ui" styles are applied



# Render phase steps (cont.)

6. Element's visibility mode is set via the hideMode attribute
7. if overCls is set, mouseover/out events are hooked
8. render event is fired
9. Component's contents is initialized (html, contentEl, tpl/data)

# Render phase steps (cont.)

10. afterRender is executed
    1. Container Layout is initialized (AbstractContainer)
    2. ComponentLayout is initialized (AbstractComponent)
    3. Component's size is set
    4. If floating, component is moved in the XY Coordinate space



# Render phase steps (cont.)

11. afterrender event is fired
12. afterRender events are hooked into the cmp's Element.
13. Component is hidden if configured
14. Component is disabled if configured



# Destruction phase steps

1. beforedestroy event is fired
2. If floating, the component is deregistered from floating manager
3. Component is removed from its parent container
4. Element is removed from the DOM
    1. element listeners are purged



# Destruction phase steps (cont.)

6. onDestroy is called
7. Plugins are destroyed
8. Component is deregistered from ComponentManager
9. destroy event is fired
10. State mixin is destroyed
11. Component listeners are purged



# Creating extensions

- Ext JS is designed to be extended, overridden - period!
- Take advantage of this!
- Don't extend the wrong component/class. It may impact performance.
- How do we know what component to extend ***and*** which vector? (initComponent versus constructor)?



# Deciding when to extend Panel

- Need anything that panel provides?
  - Docking of components
  - Title bar
    - If ***yes***, extend Ext.panel.Panel
    - else, Extend Container or Component

# Deciding when to extend Container

- Need to render a Component that manages other Components using a layout?
  - If ***yes***, extend Ext.container.Container
  - else, Extend Ext.Component

# Deciding when to extend Component

- Need to render a widget with custom HTML?
- Don't need a panel's title bar, or to dock components?
- Don't need to render child widgets ***and*** manage them with a layout?
    - Extend Ext.Component



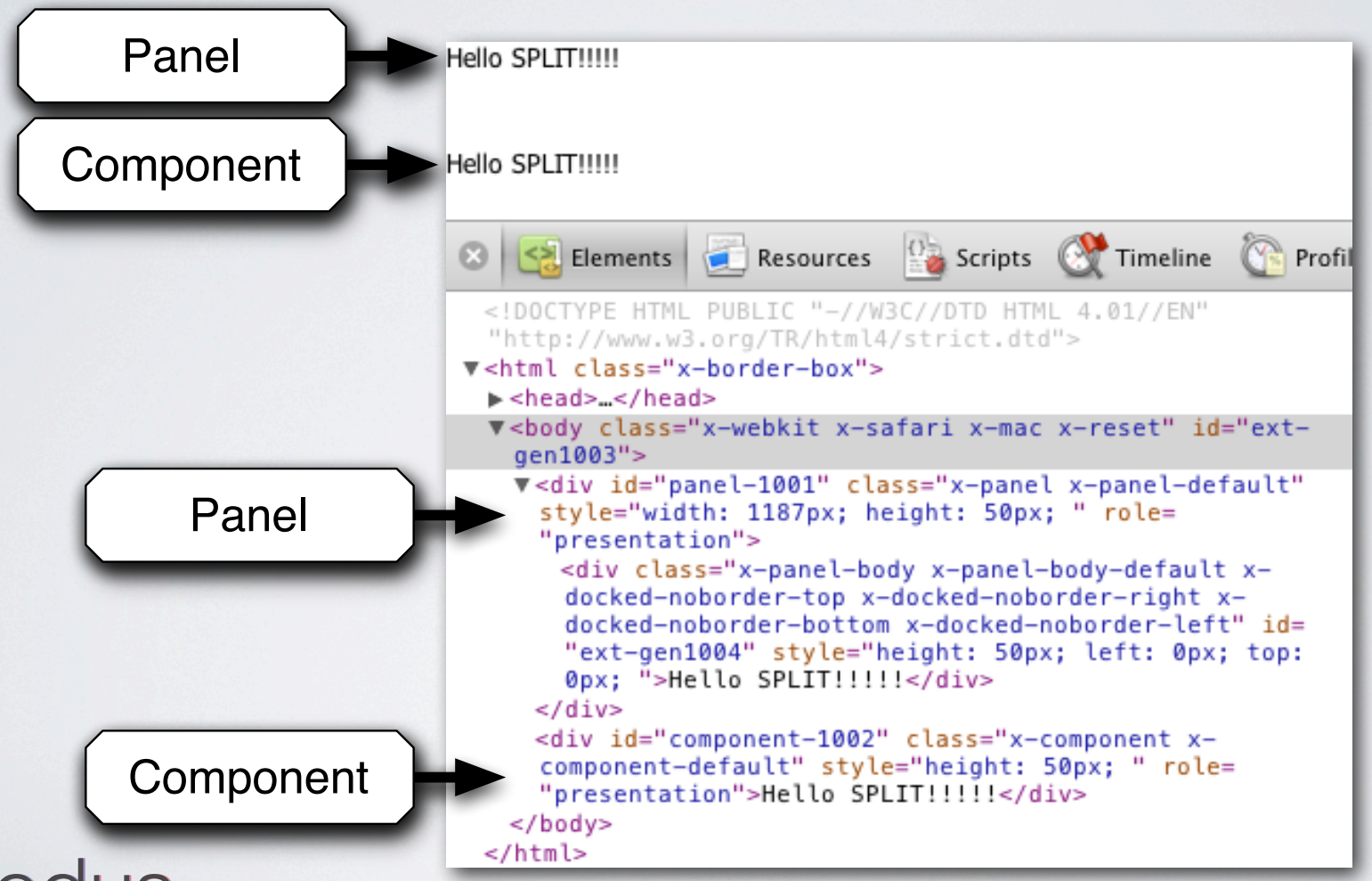
# Why does this matter?

- Simply put, choosing the wrong component to extend is wasteful!



# Panel vs. Component for simple HTML

# Panel vs. Component for simple HTML




Modus

# Component with tpl & data

```
Ext.define('MyApp.MyComponent', {
    extend   : 'Ext.Component',
    tpl : new Ext.XTemplate(
        '<tpl for=".">',
            '<div style="', '{[this.getStyle(values)]}', '">',
                '{name}, {age}',
            '</div>',
        '</tpl>',
        {
            getStyle : function(prsn) {
                var color = prsn.sex == 'M' ? '#E9E9F9' : '#F9E9E9';
                return 'background-color: ' + color + ';'
            }
        }
    ),
    data  : [
        { name : 'Jay Garcia', sex : 'M', age  : 32 },
        { name : 'Erika Garcia', sex : 'F', age  : 32 },
        { name : 'Takeshi Garcia', sex : 'M', age  : 5 },
        { name : 'Kenji Garcia', sex  : 'M', age  : 3 }
    ]
});
```

# Panel with child items

```
Ext.define('MyApp.MyPanel', {
    extend : 'Ext.panel.Panel',
    alias  : 'widget.MyApp.MyPanel',

    frame : true,
    title : 'Master Panel',

    layout : {
        type  : 'hbox',
        align : 'stretch'
    },
    defaults : {
        flex  : 1,
        frame : true
    },
    initComponent : function() {
        this.items = this.buildItems();
        this.callParent();
    },
    buildItems : function() {
        return [
            // child items
        ];
    }
});
```

Class being extended

Inline XType registration

Simple configs

Complex configs

Extend via initComponent

Call superclass initComponent

Use factory methods

Modus

{S}

# Plugins

- Instantiated & initialized during initialization phase of Component
- Typically used to bridge functionality between classes
- Introduced in Ext JS 2
- Ext JS 4 has an Abstract plugin (template)
- Still has its place in the world of Ext JS 4
- Now has a *pluginId* property



# Abstract Plugin

```
Ext.define('Ext.AbstractPlugin', {
    disabled: false,

    constructor: function(config) {
        Ext.apply(this, config);
    },

    getCmp: function() {
        return this.cmp;
    },

    init: Ext.emptyFn,
    destroy: Ext.emptyFn,

    enable: function() {
        this.disabled = false;
    },

    disable: function() {
        this.disabled = true;
    }
});
```

Modus

# Simple plugin pattern

```
Ext.define('MyApp.MyPlugin', {
    init : function(cmp) {
        this.cmp = cmp;
        Ext.apply(cmp, this.cmpOverrides);
    },
    cmpOverrides : {
        // Any overrides here to be
        // applied to parent.
    }
});
```

# Thank you!

Contact info:

jay@moduscreate.com

@_jdg (twitter)

Modus