User Interface Development with jQuery

Colin Clark, Fluid Project Technical Lead, Adaptive Technology Resource Centre

Justin Obara, Fluid Infusion Product Manager, Adaptive Technology Resource Centre

Topics We'll Cover

- What is jQuery?
- JavaScript 101: A refresher course
- The jQuery Way
- Finding things
- Attributes, classes, and more
- Events
- Accessibility and jQuery
- DOM manipulation
- AJAX
- Portal friendliness, application frameworks, and Fluid Infusion

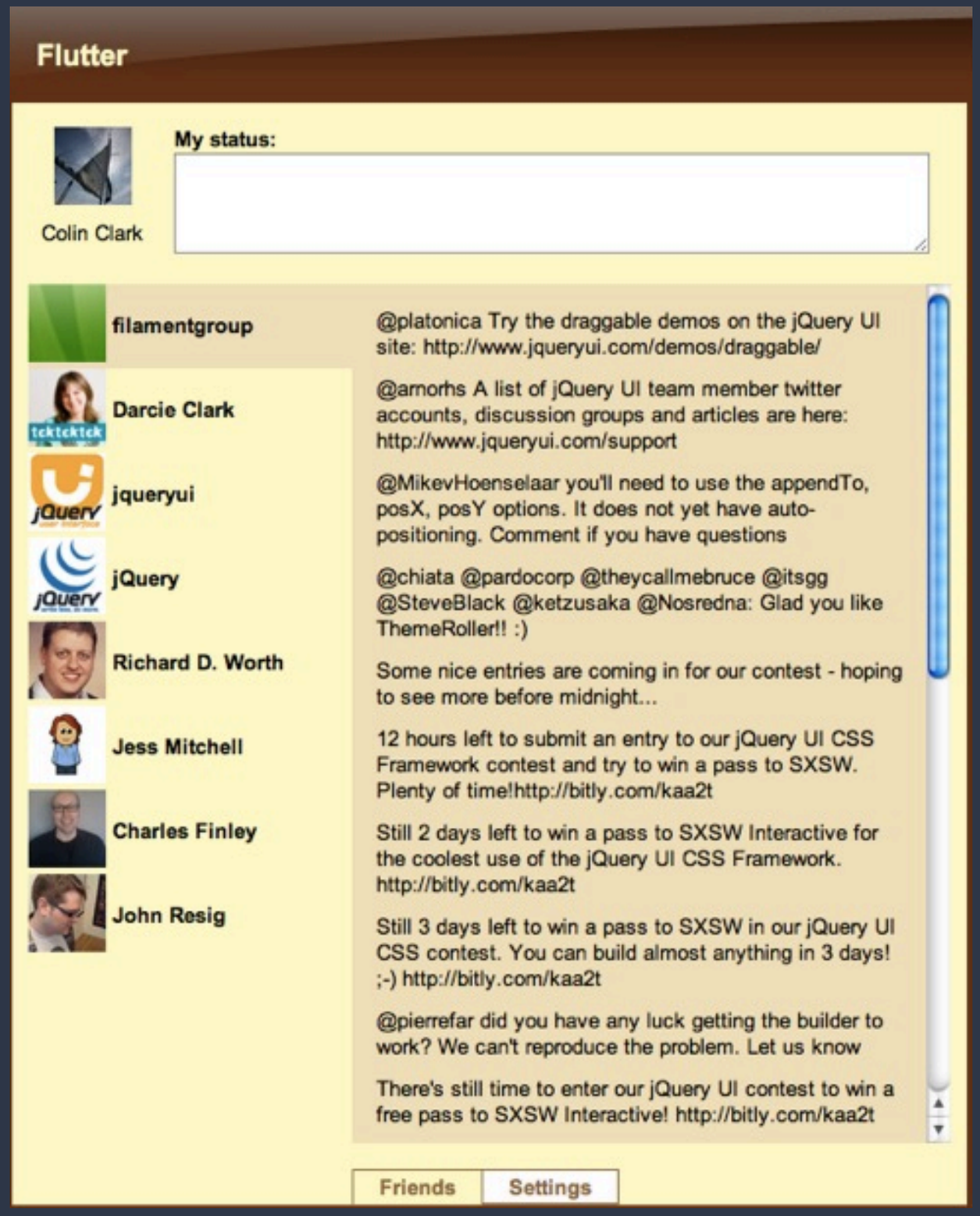
Who are we?

Who are you?

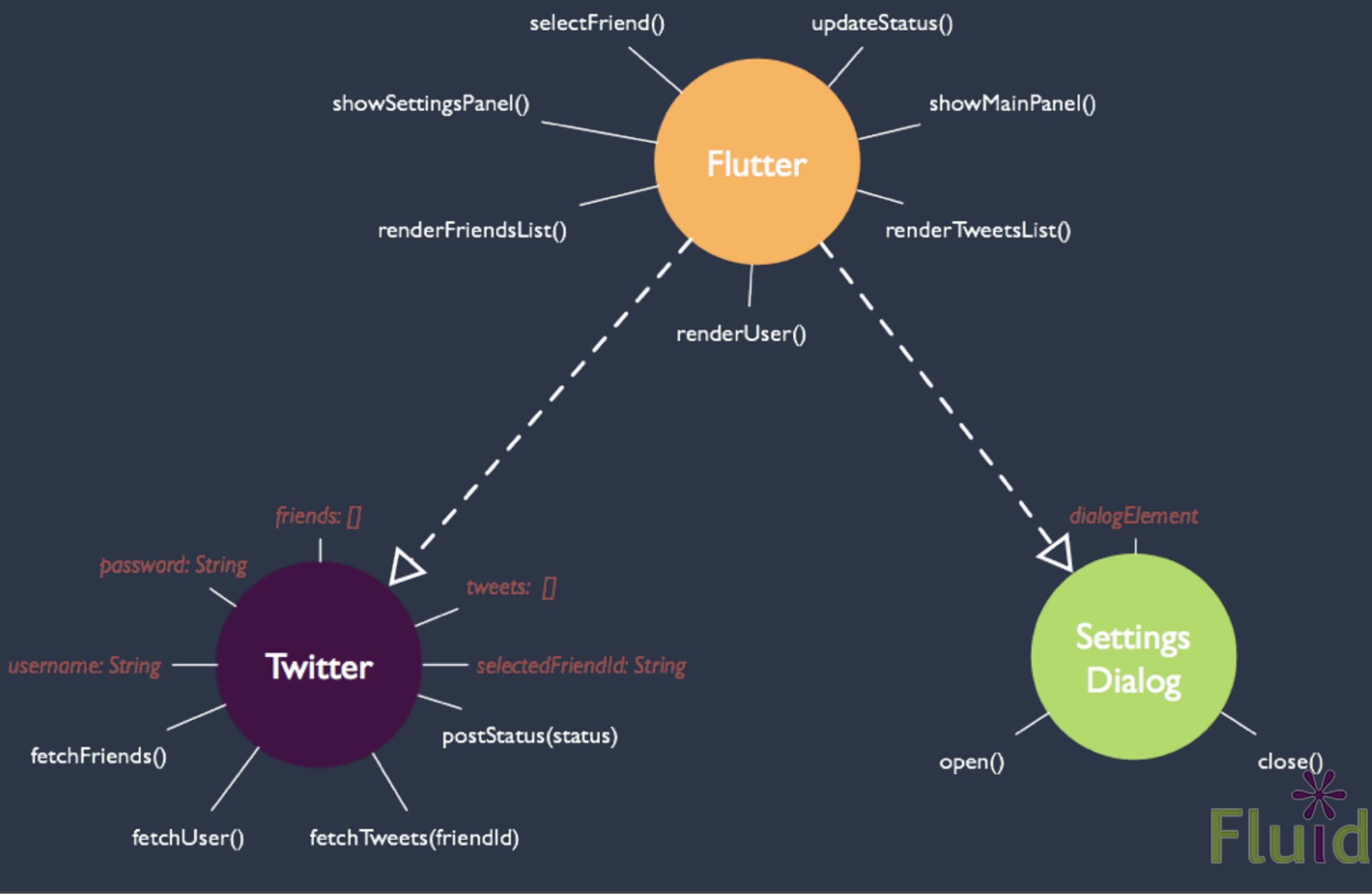
Check out the Sample Code

```
http://source.fluidproject.org/svn/scratchpad/jquery-workshop/trunk/
```

Example Code: Flutter



Flutter Architecture



Things You'll Need

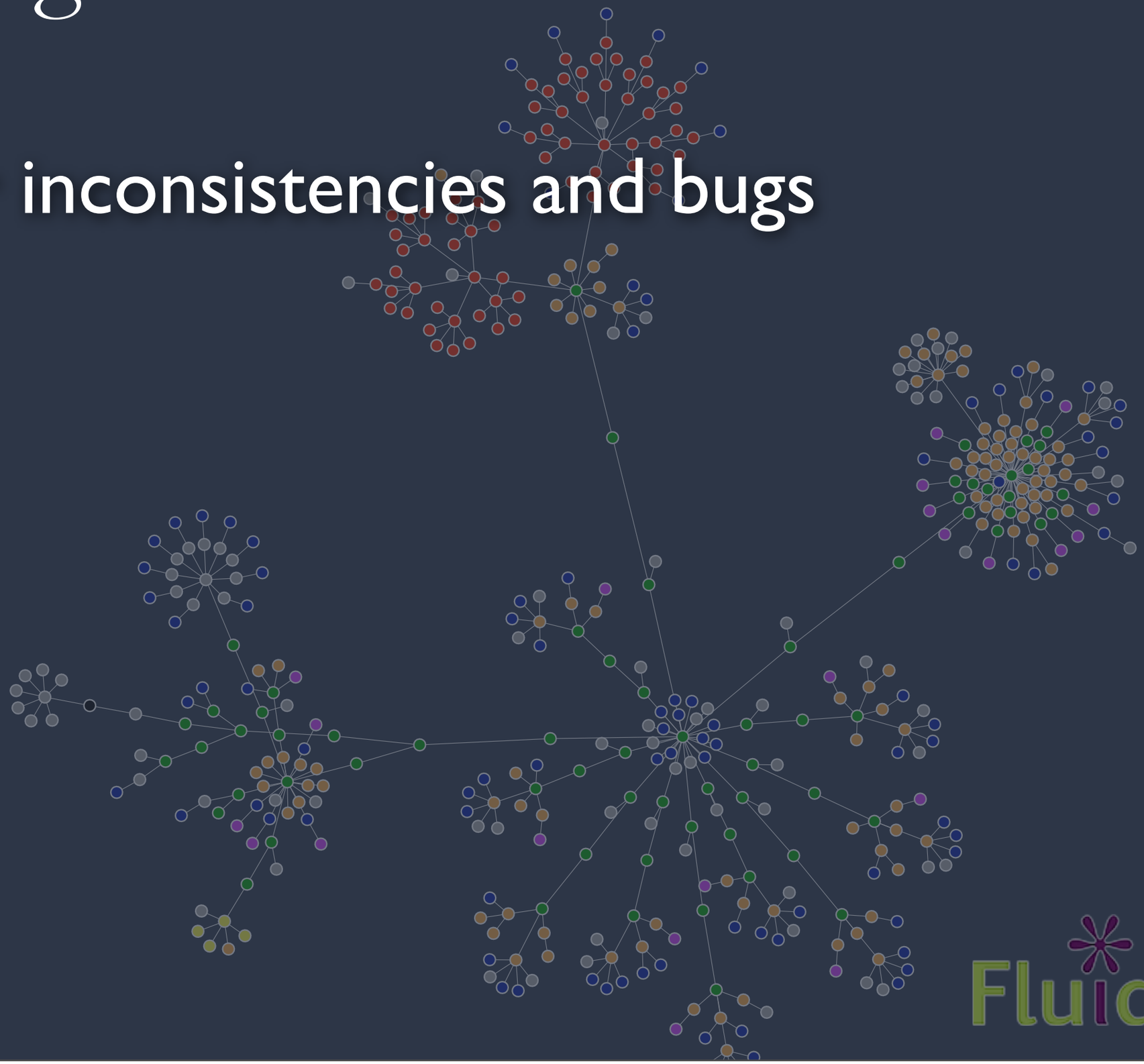
- Your favourite editor
  - Aptana or Eclipse?
- A sane browser with a good debugger
  - Firefox + Firebug
- A Servlet container
  - Tomcat or Jetty, if you want to try with live tweets

What is jQuery?

Challenges of the Client-Side

Challenges of the Client-Side

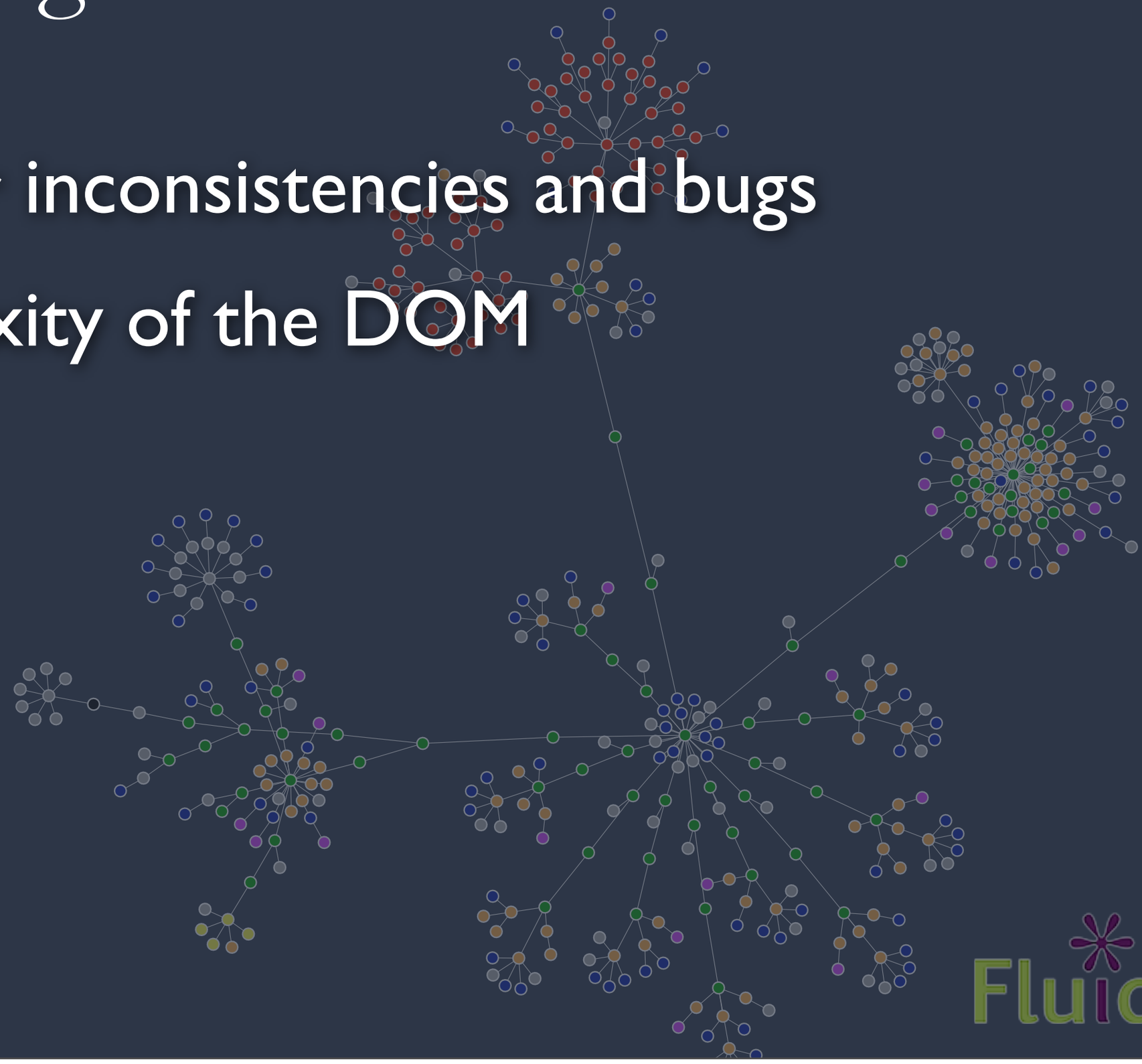
- Browser inconsistencies and bugs



Fluid

Challenges of the Client-Side

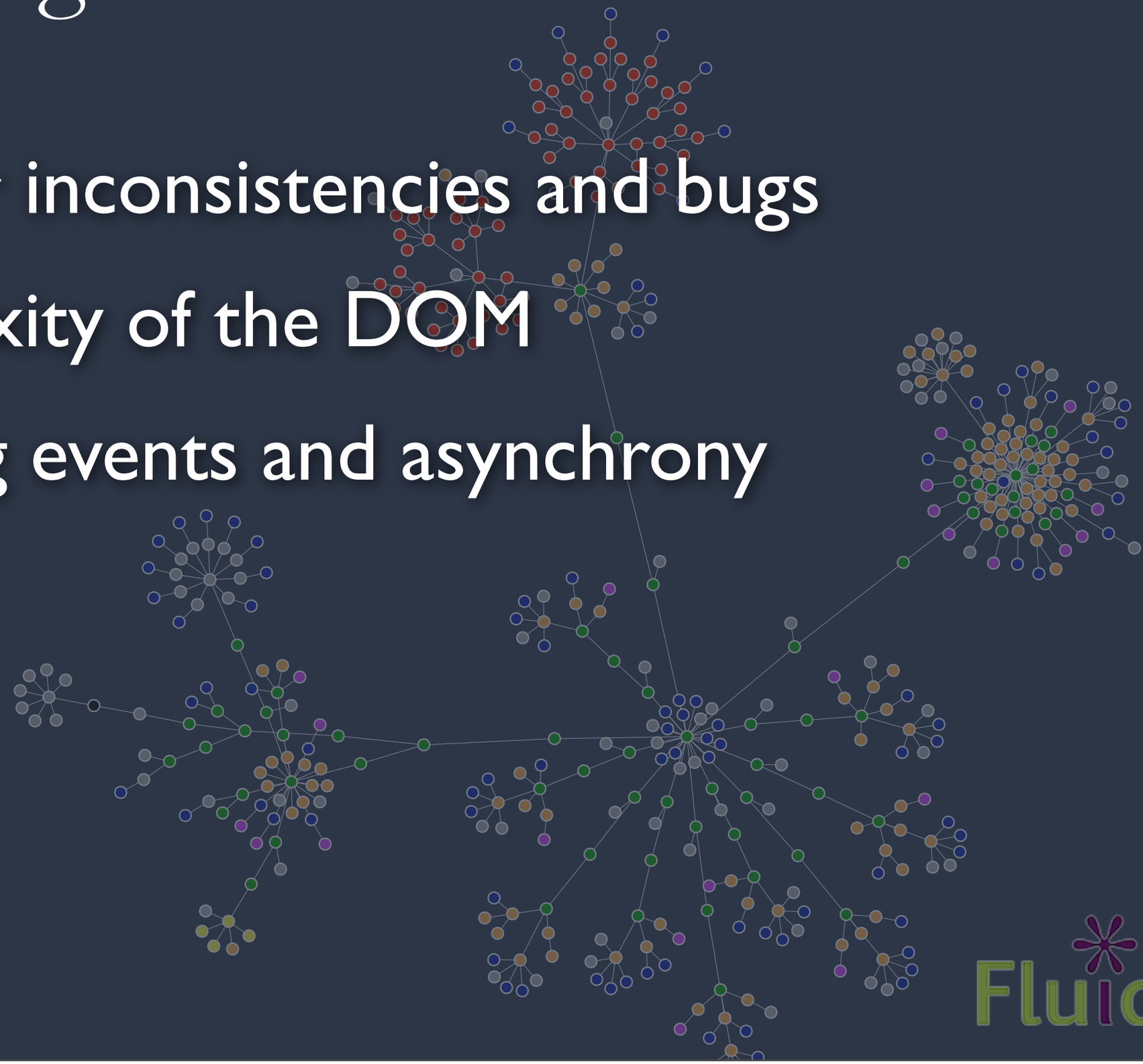
- Browser inconsistencies and bugs
- Complexity of the DOM



Fluid

Challenges of the Client-Side

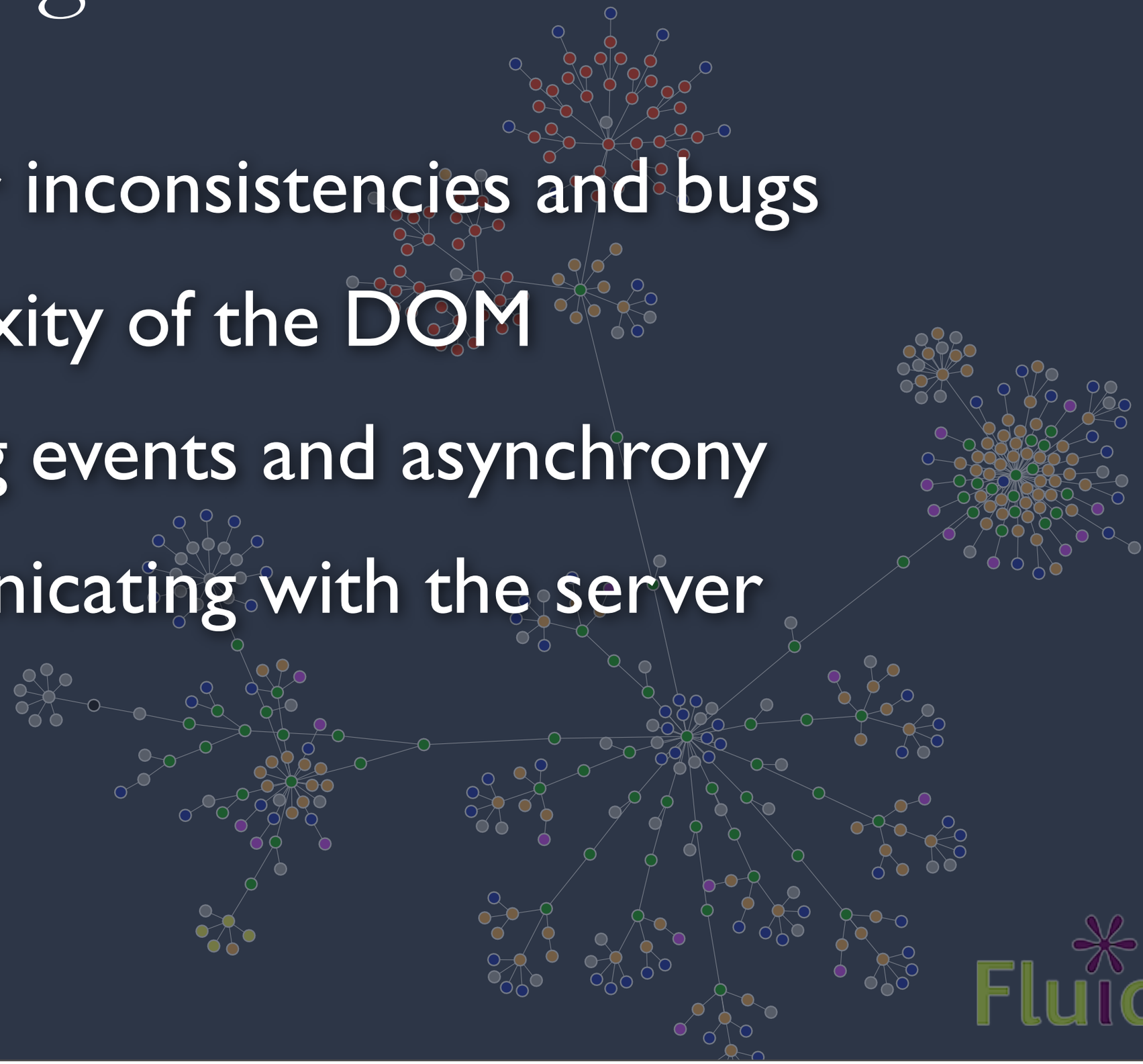
- Browser inconsistencies and bugs
- Complexity of the DOM
- Handling events and asynchrony



Fluid

Challenges of the Client-Side

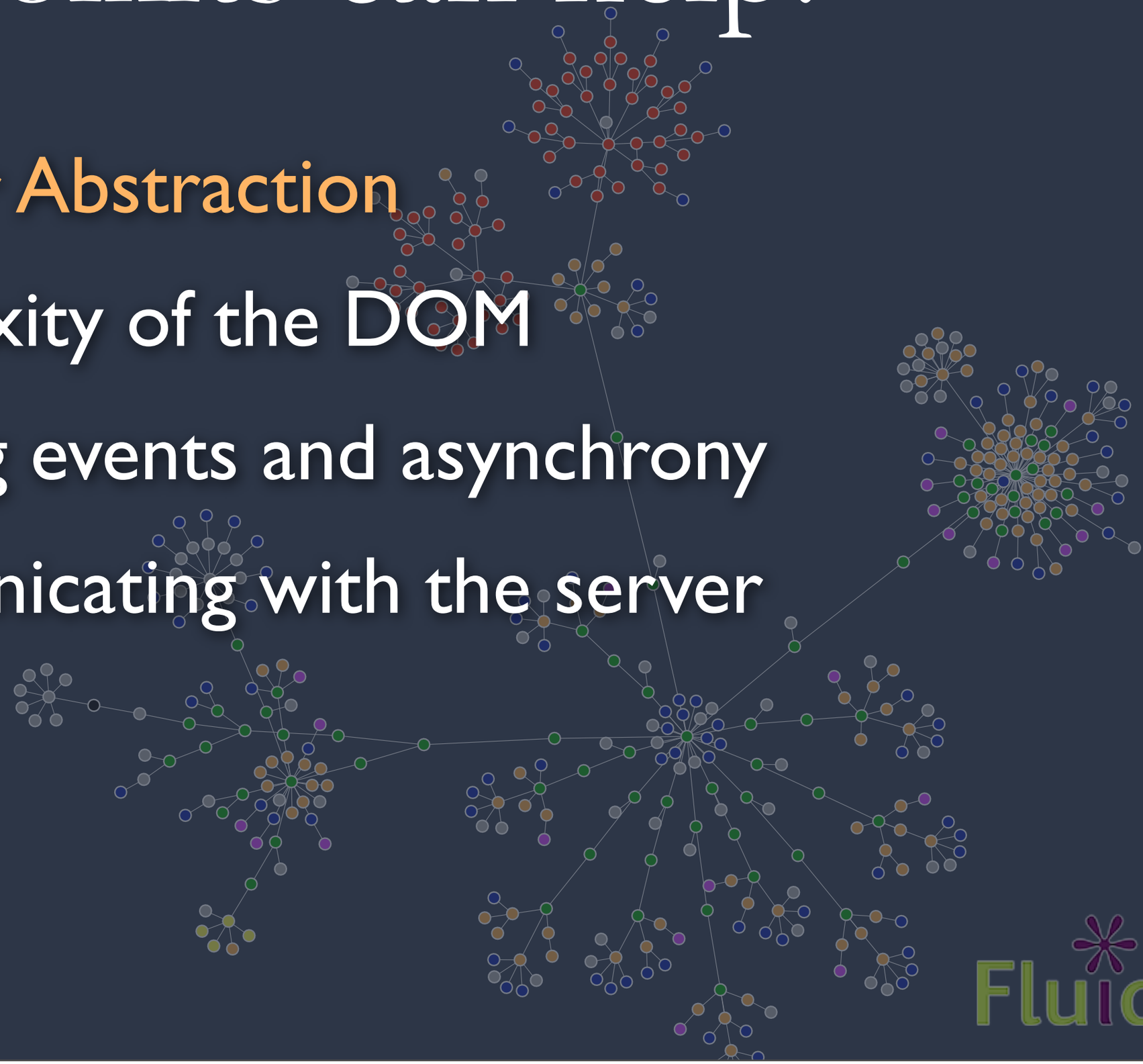
- Browser inconsistencies and bugs
- Complexity of the DOM
- Handling events and asynchrony
- Communicating with the server



Fluid

Toolkits can help!

- Browser Abstraction
- Complexity of the DOM
- Handling events and asynchrony
- Communicating with the server



Fluid

Toolkits can help!

- Browser abstraction
- A simple, unified API for the DOM
- Handling events and asynchrony
- Communicating with the server

Toolkits can help!

- Browser abstraction
- A simple, unified API for the DOM
- Easy, functional events system
- Communicating with the server

Fluid

Toolkits can help!

- Browser abstraction
- A simple, unified API for the DOM
- Easy, functional events system
- Built-in AJAX, XML, and JSON support

Find something...

Find something...

and do something with it

Fluid

doing something without a toolkit

```
function stripeListElements() {
    // get the items from the list
    var myItems = document.getElementsByTagName("li");
    // skip line 0 as it's the header row
    for(var i = 0; i < myItems.length; i++) {
        if ((i % 2) === 0) {
            myItems[i].className = "even";
        }
    }
}
```

Fluid

doing something with jQuery

```
jQuery("li");
```

doing something with jQuery

```
jQuery("li:even");
```

doing something with jQuery

```
jQuery("li:even").addClass("striped");
```

Types of JavaScript Tools

- Foundational toolkits
- Widget libraries
- Application frameworks

... compare and contrast

Foundational toolkits

- Totally presentation focused
- DOM manipulation
- Event binding
- Ajax

jQuery
Prototype
Dojo core

Widget Libraries

- Reusable user interface widgets
  - Drag & Drop
  - Tabs
  - Sliders
  - Accordions

jQuery UI
Ext
Scriptaculous

Fluid

Application frameworks

- Model notifications "something changed here"
- Views to help keep your presentational code clean
- Data binding to sync the display with your model

SproutCore

Dojo/Dijit/ Dojox

Cappuccino

Fluid

jQuery in a Nutshell

- Everything you need for:
  - Finding things
  - Styling things
  - Manipulating things
  - Attaching events
  - Making AJAX Requests
- Pretty low-level: you'll need more

The jQuery Way

jQuery Philosophy

- Unobtrusive
  - Separation of presentation, structure, logic
- Lightweight
  - Browser normalization, DOM, AJAX. That's it
- Functional, not object oriented

JavaScript 101 (quickly)

JavaScript is Different

- Everything is an object
- Has an extremely loose type system
- No classes
- Functions are first class
- Some annoying bugs

Defining Variables

```
var mango = "yum";
mango = 12345;
mango = false;
```

Defining Variables

- If you omit var, it will be defined as a global variable.
- This is extremely dangerous; JavaScript won't warn you!

```
rottenTomato = "gross!"; // This is global
```

Truthy and Falsey

- JavaScript automatically coerces types

```
if (cat) {
    cat.meow();
}
```

- Unlike static language: “shades” of true and false
- Use with care

Falsey Values

`false`

`null`

`undefined`

`""`

`0 (zero)`

`NaN`

- Everything else is truthy. Careful...

  `-1`,`"false"`,`"0"` are all true

Equivalent vs. Equal

Comparisons are coercive:

```
1 == "1" // true

0 == false // true
```

Non-coercive comparison:

```
0 === false // false

1 !== "1" // true

1 === Number("1") // true
```

Fluid

Objects Are Loose Containers

- Objects are just maps
- Keys can be any string, values can be anything
- Two different ways to access members:

```
basketOfFruit.kiwis; // dot notation
basketOfFruit["figs"]; // subscript notation
```

- You can add new members to any object at any time

No Classes

- JavaScript does not have a class system
- Methods are just properties in a container:
  - pass them around
  - modify them
  - delete them
- Inheritance is prototypal, but this is broken

{ }

Fluid

Objects Are Modifiable

```
var basketOfFruit = {
    pears: “bartlett”,
    oranges: “mandarin”
};
```

```
// New property
basketOfFruit.apples = "macintosh";

// New method
basketOfFruit.eat = function () {
    return “tasty”;
};
```

First Class Functions

1. Functions are real objects.
2. Functions are data: assign, pass, return them
3. Functions remember their scope and carry state

A Simple Closure

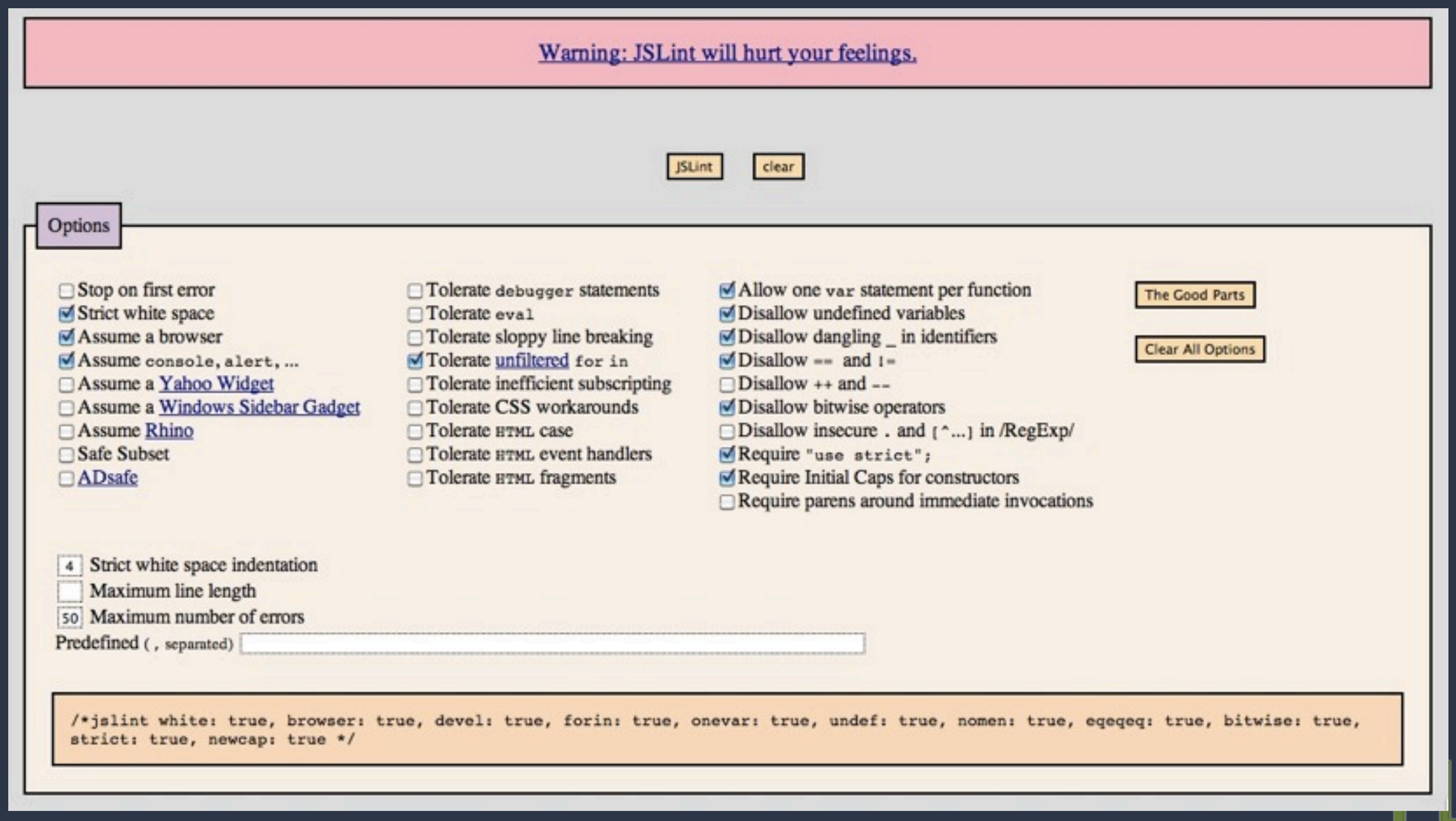
```
var addNumber = function (a) {
// This function will remember the value of a
    return function (b) {
        return a + b;
    };
};

var addOne = addNumber(1); // result is an “add 1” Function
addOne(5); // Result is 6
addOne(41); // Result is 42
```

A Realistic Example

```
var tweetBox = $(“#statusTextField”);

function createTweetHandler () {
    var user = {
        id: “12345”,
        lastTweet: null
    };

    return function (evt) {
        user.lastTweet = tweetBox.val();
    }
}

tweetBox.bind(createTweetHandler());
```

The Good Parts

JSLint: a static analyzer for JavaScript



Getting Started with jQuery

A shape for your code

```
// Your namespace is the only global variable.
var namespace = namespace || {};

// A private space, with a helpful alias to jQuery
(function ($) {

  // To make something available, add it to your namespace.
  namespace.myFunction = function () {

  };

})(jQuery);
```

jQuery === $

Constructor:

```
$(selectorString | Element | Array | jQuery);
```

Returns:

A jQuery instance.

Examples

```
// Selector
var allListItems = $(“li”);

// DOM Element
var theWholeDocument = $(document);
```

What's a jQuery?

- A wrapper for one or many elements
- A real object with useful methods
- A better API than the raw DOM
- Context-oriented and chainable:

```
$("li").addClass("selected").attr("tabindex", "-1").text("Hello!");
```

Basic jQuery Methods

```
var allListItems = $(“li”);

// Get the id attribute
allListItems.attr(“id”);

// Add a CSS class name.
allListItems.addClass(“stripey”);

// Get all the children
allListItems.children();

// Find items scoped within another jQuery
$(“a”, allListItems);
```

A Unified API for One or Many

- Most DOM code requires a lot of looping:

```
var myItems = document.getElementsByTagName("li");
for (var i = 0; i < myItems.length; i++) {
  myItems[i].className = “foo”;
}
```

- jQuery treats sets the same as single elements:

```
$("li").addClass(“foo”);
```

- Bottom line: no iteration means way less code (and it’s portable)

One or many?

```
// Returns the id attribute of the first element.
$(“li”).attr(“id”);

// Sets the tabindex attribute of all elements.
$(“li”).attr(“tabindex”, “-1”);
<li tabindex=”-1”>foo</li>

// Adds the class name to all elements.
$(“li”).addClass(“highlighted”);

// Returns true if at least one has this class
$(“li”).hasClass(“highlighted”);
```

Accessing Members

```
// Get the element at a specific position, as a jQuery
$(“li”).eq(0);

// Get the element at a specific position,
// as a pure DOM element
$(“li”)[0];

// Loop through each element in a jQuery
$(“li”).each(function (index, item) {
    $(item).addClass(“highlighted”);
});
```

Selectors

What's a Selector?

- Selectors are specified by a string
- A notation for identifying elements in the DOM
- The same thing you use when you're writing CSS

Types of Selectors

- Element selectors

  `“div”` `“span”` `“ul”` `“li”` `“body”`

- id selectors

  `“#flutter-friends”` `“#friends-error-dialog”`

- Class name selectors

  `“.invisible”` `“.flutter-status-panel”`

More Selectors

- Descendent selectors:

ancestor *descendent*

```
".flutter-status-panel textarea"
".flutter-status-panel.active"
```

- Child selectors:

ancestor *child* *child*

```
"#friends>li>img"
```

- Pseudo selectors:

```
":first"    ":even"    ":hidden"    ":contains('John Resig')"
":not(#flutter-friends-template)"
```

All the details...

http://api.jquery.com/category/selectors

Doing Stuff

Manipulating Attributes

```
var friends = $(“#friends li”);

// Get the id attribute
friends.attr(“id”);

// Set the id attribute
aFriend.attr(“id”, “123456789”);

// attr() also provides normalization
friends.attr(“tabindex”, -1);
```

Manipulating Classes

```
var friends = $(“#friends li”);

// Add a class name
friends.addClass(“flutter-hidden”);

// Remove a class name
friends.removeClass(“flutter-hidden”);

// Toggle a class name on or off
friends.toggleClass(“flutter-hidden”);
```

Directly Manipulating Styles

```
var friends = $(“#friends li”);

// Get the element’s computed border-color style
friends.css(“border-color”);

// Set the element’s style
friends.css(“border-color”, “red”);
friends.css(“border”, “5px”);

// Get and set height and width
settingsPanel.height();
settingsPanel.width(400);
```

Is the document ready?

- HTML gets parsed by the browser linearly
- Head first, then body, etc.
- So all your <head> scripts will execute immediately

```
<head>
  <script>
    console.log(“Number of <li>s on the page: “ +
                $(“li”).length);
  </script>
</head>
```

- How do you know when the page is loaded?

Script Blocks

```
<body>

  <ul>
    <li>Fish</li>
    <li>Penguins</li>
    <li>Kings</li>
  </ul>

  <script type=”text/javascript”>
    console.log(“Number of <li>s on the page: “ +
                $(“li”).length);
  </script>

</body>
```

$(document).ready(fn)

```
$(document).ready(function () {
        // This is the earliest point at which
        // the whole document is ready.
         console.log(“Number of <li>s on the page: “ +
                        $(“li”).length);
});
```

Exercise 1

Finding Things

- Find the following things:
  - The friends list
  - All list items in every list on the page
  - The list items inside the friends list
  - Everything with the class fl-centered
  - The first form element on the page
  - The last item in the friends list
  - The label for the username text field
- Give each thing a different background colour (there is a style sheet you can use if you're not hot on CSS)

Fluid

Exercise 1 Illustrated

Fluid

Events

Types of Browser Events

- Mouse
  - click() when the mouse button is clicked
  - mouseover() when the cursor is over an element
- Keyboard events:
  - keydown() as soon as a key is pressed down
  - keyup() when the key is released
  - keypress() can be buggy and inconsistent
  - focus() when an element is clicked or focused with the keyboard
  - blur() when focus leaves the event

jQuery and Events

- Events vary wildly across browsers
- Netscape vs. IE vs. W3C: they're all different
- jQuery normalizes all the standard browser events
- Also lets you define your own custom events

Event Bubbling

- Browser detects an event
- Starts at the immediate target element
- If a handler is not found, the parent is then checked
- And onwards up the tree
- If a handler is present, it is invoked
- Handler can stop bubbling; otherwise, the event propagates up the tree as above

Binding Events

```
// The generic way
$(“li”).bind(“click”, function (event) {
    alert(“You clicked me!”);
});

// Event binding shortcut
$(“li”).click(function (event) {
    alert(“You clicked me!”);
});
```

Event Handlers

- The `event` object provides more information about the event that occurred
- `this` points to the element on which the event listener was bound. Be careful!
- Event handlers always deal with pure elements, not jQuery instances (`this` and `event.target`)

Events: `this` vs. `target`


```
var friends = $("#friends");
friends.click(function (event) {
  // this === $("#friends");
  // evt.target === $("#friends li")[3]
});
```

Fluid

Default Actions

- When links are clicked, a new page loads
- When an arrow key is pressed, the browser scrolls
- When Enter is pressed in a form, it submits
- You can prevent the browser's default action:

```
$("a").bind("click", function (event) {
    event.preventDefault();
});
```

Stopping Propagation

- By default, events will propagate up the tree after your handler runs
- Stopping propagation:

```
$(“a”).click(function (event) {
    event.stopPropagation();
});
```

- To swallow propagation and the default action:

```
$(“a”).click(function (event) {
    return false;
});
```

The Event Object

```
{
  altKey: boolean,
  ctrlKey: boolean,
  metaKey: boolean,
  shiftKey: boolean, // Were these modifier keys depressed?
  keyCode: Number, // The numeric keycode for key events
  which: Number, // Keycode or mouse button code
  pageX: Number, // Horizontal coordinate relative to page
  pageY: Number, // Vertical coordinate relative to page
  relatedTarget: Element, // Element left or entered
  screenX: Number, // Horizontal coordinate relative to screen
  screenY: Number, // Vertical coordinate relative to screen
  target: Element, // The element for which the event was triggered
  type: String // The type of event that occurred (eg. “click”)
}
```

Event Delegation

- Often you've got a lot of elements on page (e.g. Twitter friends)
- Each shares the same event behaviour
- In this case, bind your event to their container:

```
<ul id="friends">
    <li id="19539154">
        <img />
        Darcie Clark
    </li>
    <li id="19539154">
        <img />
        John Resig
    </li>
    <li id="19539154">
        <img />
        Justin Obara
    </li>
</ul>
```

```
// This will be slow, because it binds an
// event on each item in the list.
$("#friends li").click(function (event) {
  showTweetsForFriend(this);
});

// Bind the event to the container, and let
// it bubble up. Way faster.
var friends = $("#friends");
friends.delegate("li", "click", function (evt) {
  showTweetsForFriend(this);
});
```

Fluid

Removing Events

```
// Remove all event listeners.
$(“li”).unbind();

// Remove all click event listeners.
$(“li”).unbind(“click”);

// Remove a specific listener.
var myListener = function (event) {...};
$(“li”).bind(myListener);
$(“li”).unbind(myListener);

// Remove a delegate-based event.
$(“#friends”).undelegate(“li”, “click”);
```

One-off Events

```
// This event will only ever fire once.
$(“li”).one(“click”, function (event) {
     alert(“You’ll only see me once.”);
});
```

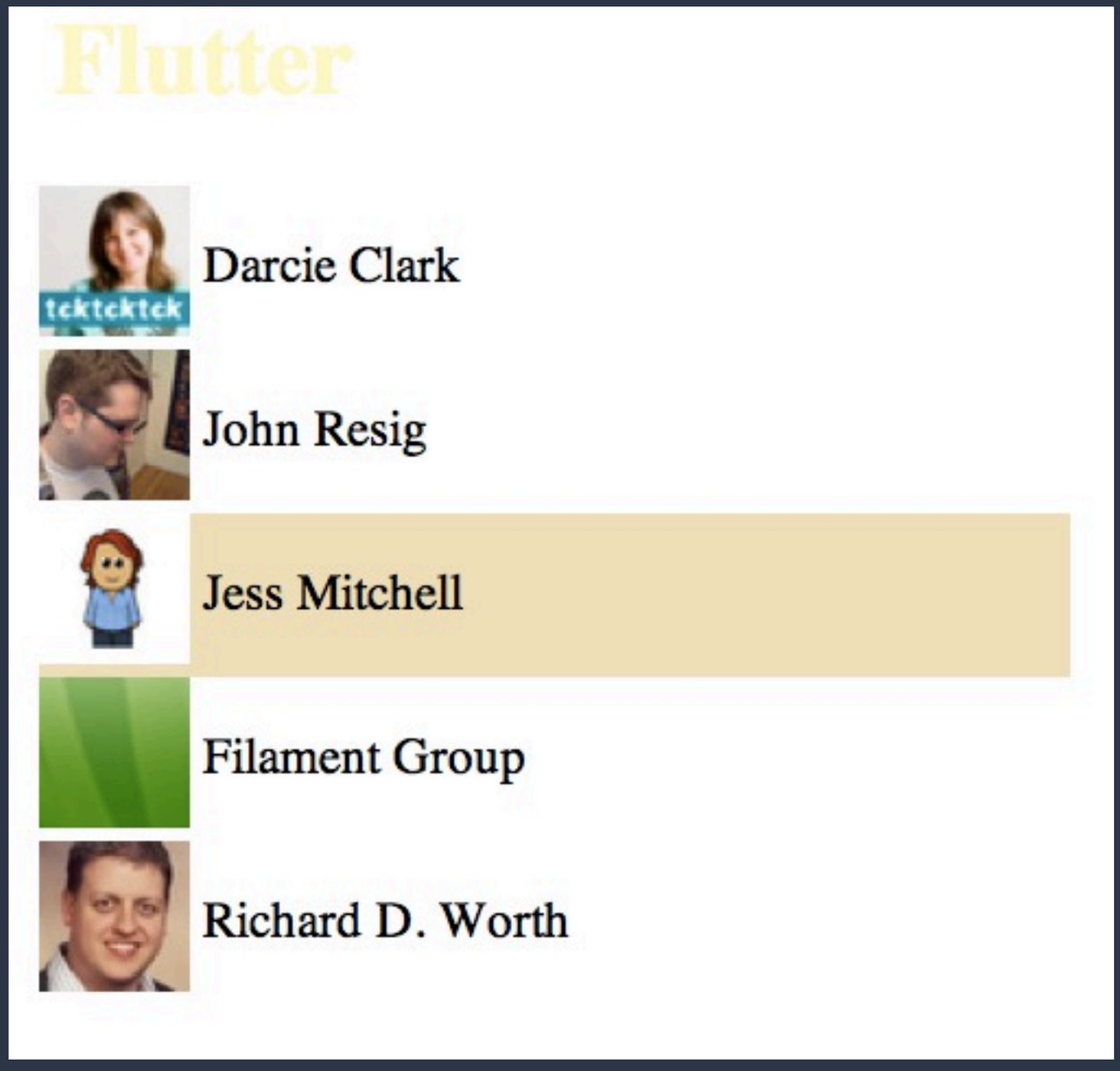
```
// A more awkward, verbose version:
var fireOnce = function (event) { ... };
$(“li”).bind(“click”, function (event) {
     fireOnce();
     $(“li”).unbind(fireOnce);
});
```

Exercise 2: Events

Binding Events

- Bind click handlers to each of the friend <li> elements.
- Your click handler should invoke a `selectFriend()` function with the friend that was clicked.
- The `selectFriend()` function should use jQuery to adjust the CSS classes of the friend elements so that the clicked element has the `flutter-active` style

Exercise 2 Illustrated



Fluid

DOM Manipulation

Getting/Setting Element Values

```
// Get a value from a form element.
$(“#status”).val();

// Set a value on a form element.
$(“#status”).val(“Giving a presentation a Jasig.”);

// Getting the text of an element.
$(“#status”).text();

// Setting the text of an element.
$(“#status”).text(“John Resig”);
```

DOM Manipulation

- The traditional DOM provides methods for creating new elements and adding them to existing elements
- Can also be quite slow

- IE implemented the now ad-hoc standard innerHTML, which was faster
- jQuery provides a great API for DOM manipulation, as well as cross-browser manipulation
- Ultimately, it still uses the DOM APIs underneath: still slow

Fluid

Manipulating the DOM

```
// Create a new element.
var myList = $(“<ul></ul>”);

// Appending elements to the end of a container.
var otherListItems = $(“li”);
myList.append(otherListItems);

// Same result.
otherListItems.appendTo(myList);

// Remove an element from the DOM entirely.
// Conveniently, this returns the item you just removed
$(“#flutter-friend-template).remove();

// Remove all children from a container.
myList.empty();
```

More manipulation: copying

```
// Clone an element
$(“#flutter-friend-template”).clone();
```

```
// Clone an element, along with all its event handlers
$(“#flutter-friend-template”).clone(true);
```

Injecting Lots of Stuff (slow)

```
// Create a new element.
var myList = $(“<ul></ul>”);
var john = myList.append(“<li></<li>”);
john.text(“John Resig”);
john.attr(“tabindex”, “-1”);

var paul = myList.append(“<li></li>”);
paul.text(“Paul McCartney”);
```

Injecting Lots of Elements at Once

```
var friends = [{
    id: “12345”,
    name: “John Resig”
}];

// The ugly, error-prone string concat way
$(“<li id=\”” + friends[0].id + “\” tabindex=\”-1\”>” +
friends[0].name + “</li>”).click(function () {
    showTweetsForFriends()
});
```

Quick Element Construction

```
var friends = [{
    id: “12345”,
    name: “John Resig”
}];
```

```
// The speedy, nice way.
$(“<li>”, {
   text: friends[0].name,
   id: friends[0].id,
   tabindex: “-1”,
   click: function () {
      showTweetsForFriend(this);
   }
});
```

DOM Manipulation Advice

- Try to use CSS instead of DOM manipulation where possible (e.g. hiding/showing elements, etc.)
- DOM manipulation can be very costly
- jQuery's API is great, but it isn't magic
- Avoid building up elements one at a time
- Injecting whole blocks of HTML at once:

```
myContainer.html("<ul><li>Colin</li><li>Antranig</li><li>Jess</li></ul>");
```

Exercise 3: Manipulation

Manipulating the DOM

- Bind a key handler to the entry field
- The key handler should:

i) listen for the Enter key ($.ui.keyCode.ENTER)

ii) clone the template node for a new tweet

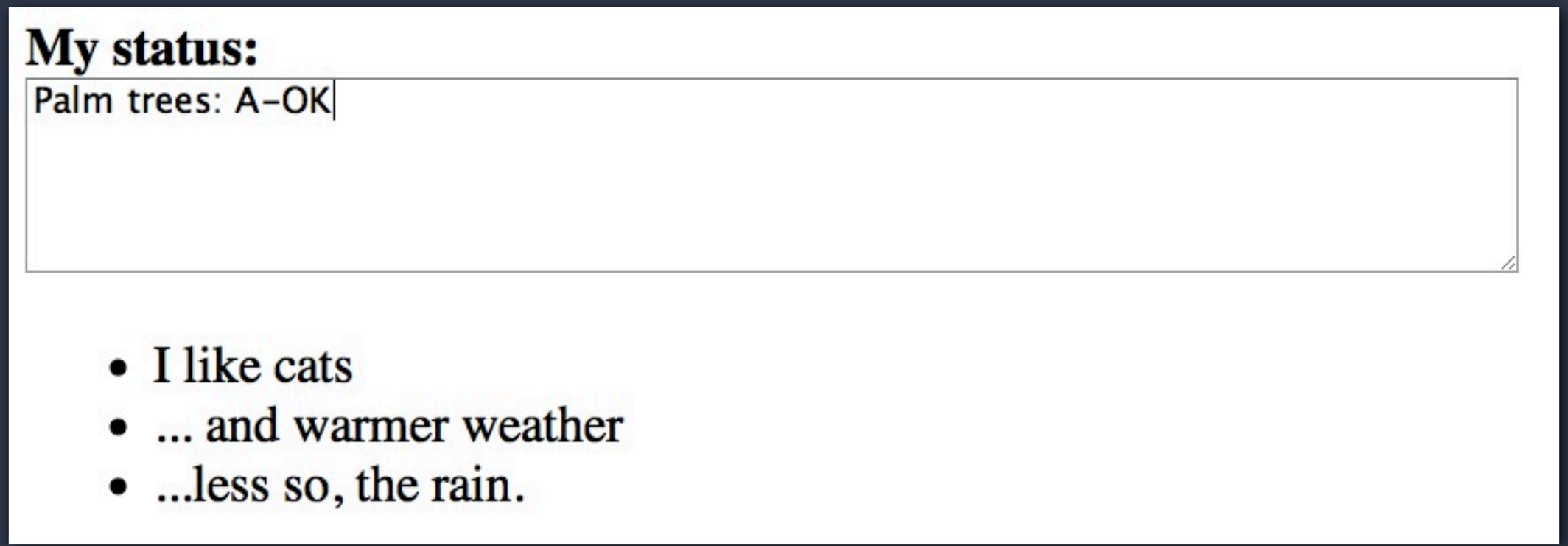
iii) fill in the node with the text entered by the user

iv) add the node to the twitter list

v) make the new node visible

vi) clear the entry field

Exercise 3 Illustrated



Accessibility

What is Accessibility?

A New Definition

- Accessibility is the ability of the system to accommodate the needs of the user
- Disability is the mismatch between the user and the interface provided
- We all experience disability
- Accessible software = better software

Assistive Technologies

- Present and control the user interface in different ways
- Not just screen readers!
- Use built-in operating system APIs to understand the user interface

Screen readers
Screen magnifiers
On-screen keyboards

DHTML: A New Can of Worms

- Shift from documents to applications
- Familiar a11y techniques aren't enough
- Most DHTML is completely inaccessible
- New techniques are still being figured out

The Problem

- Custom widgets often look, but don't act, like their counterparts on the desktop
- HTML provides only simple semantics
- Not enough information for ATs
- Dynamic updates require new design strategies to be accessible

The Solution

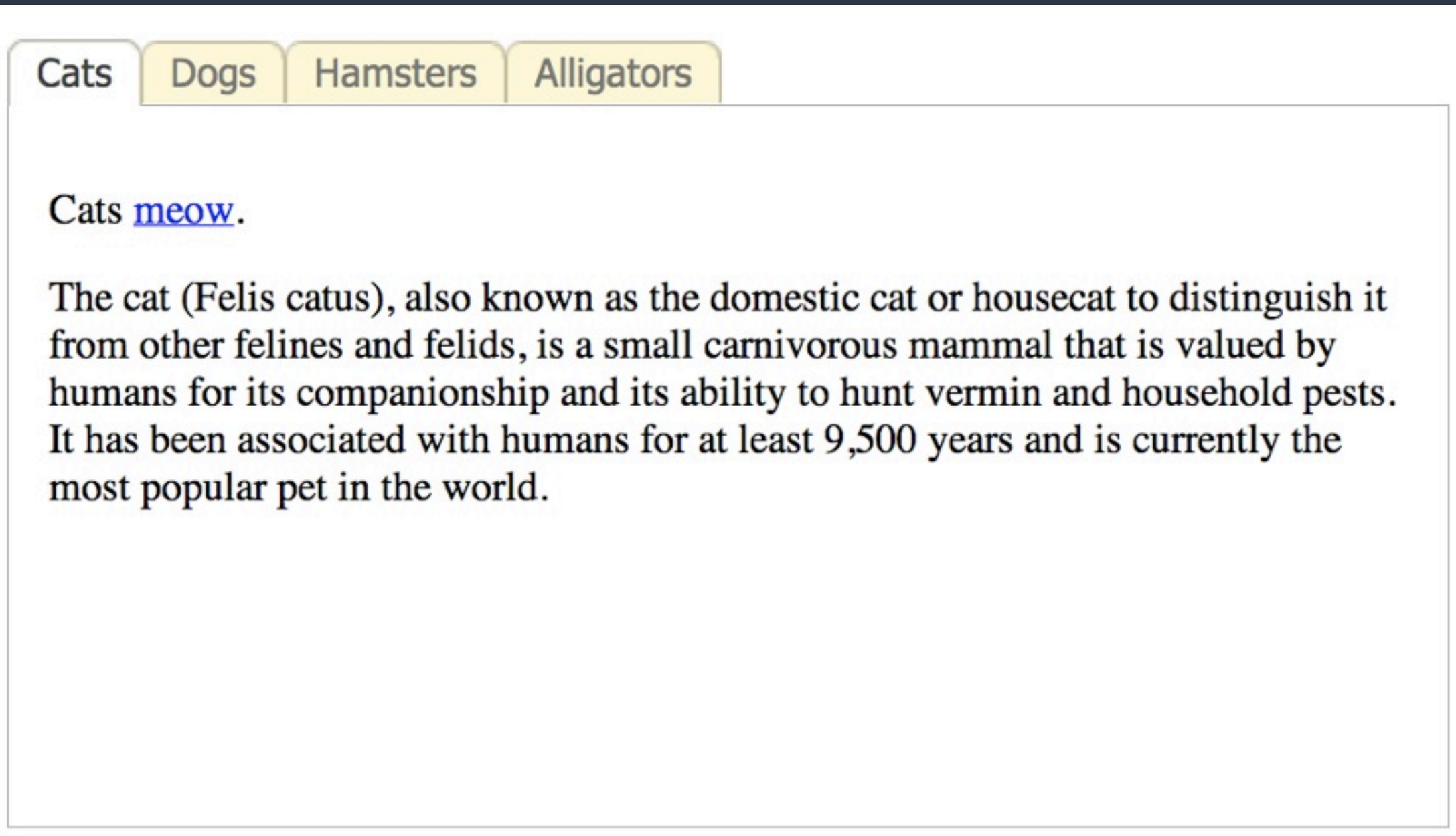
- Describe user interfaces with ARIA
- Add consistent keyboard controls
- Provide flexible styling and presentation

Supporting Assistive Technology

Opaque Markup

```
// These are tabs. How would you know?
<ol>
  <li><a href="#cats">Cats</a></li>
  <li><a href="#dogs">Dogs</a></li>
  <li><a href="#gators">Gators</a></li>
</ol>
<div>
  <div id="cats">Cats meow.</div>
  <div id="dogs">Dogs bark.</div>
  <div id="gators">Gators bite.</div>
</div>
```

Opaque Markup: Tabs



Fluid

ARIA

- Accessible Rich Internet Applications
- W3C specification in the works
- Fills the semantic gaps in HTML
- Roles, states, and properties
- Live regions

Roles, States, Properties

- Roles describe widgets not present in HTML 4
  - `slider, menubar, tab, dialog`
- Properties describe characteristics:
  - `draggable, hasPopup, required`
- States describe what's happening:
  - `busy, disabled, selected, hidden`

Using ARIA

```
// Now *these* are Tabs!
<ol id="animalTabs" role="tablist" tabindex="0">
  <!-- Individual Tabs shouldn't be focusable -->
  <!-- We'll focus them with JavaScript instead -->
  <li role="tab"><a href="#" tabindex="-1">Cats</a></li>
  <li role="tab"><a href="#" tabindex="-1">Dogs</a></li>
  <li role="tab"><a href="#" tabindex="-1">Gators</a></li>
</ol>
<div id="panels">
  <div role="tabpanel" aria-labelledby="cats">Cats meow.</div>
  <div role="tabpanel" aria-labelledby="dogs">Dogs bark.</div>
  <div role="tabpanel" aria-labelledby="gators">Gators bite.</div>
</div>
```

Adding ARIA in Code

```
// Identify the container as a list of tabs.
tabContainer.attr("role", "tablist");

// Give each tab the "tab" role.
tabs.attr("role", "tab");

// Give each panel the appropriate role,
panels.attr("role", "tabpanel");
panels.each(function (idx, panel) {
    var tabForPanel = that.tabs.eq(idx);
    // Relate the panel to the tab that labels it.
    $(panel).attr("aria-labelledby", tabForPanel[0].id);
});
```

Keyboard Accessibility

Keyboard Navigation

- Everything that works with the mouse should work with the keyboard
- ... but not always in the same way
- Support familiar conventions

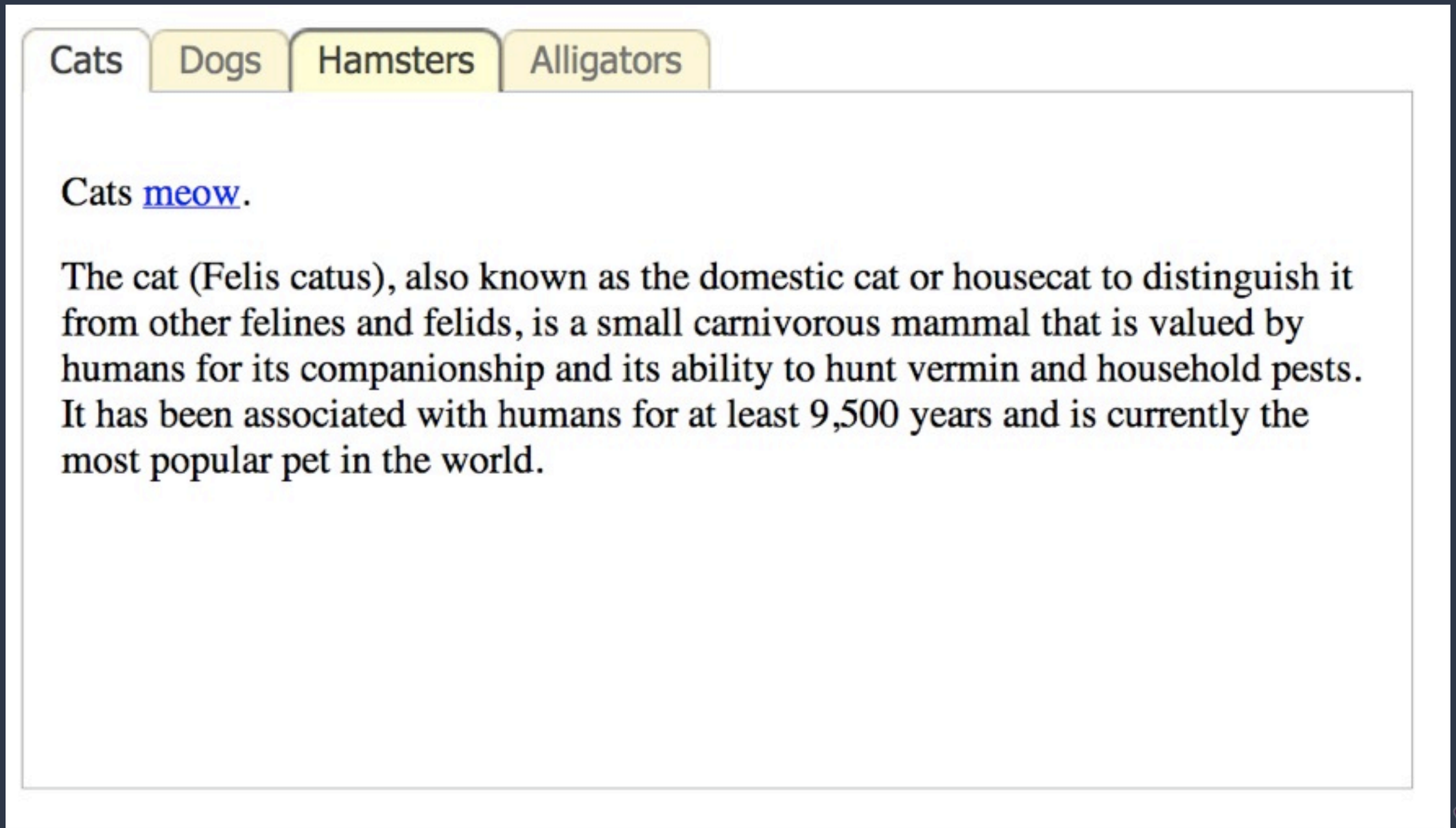
http://dev.aol.com/dhtml_style_guide

Keyboard Conventions

- Tab key focuses the control or widget
- Arrow keys select an item
- Enter or Spacebar activate an item

- Tab is handled by the browser. For the rest, you need to write code. A lot of code.

Keyboard a11y: Tabs



Fluid

Tabindex examples

```
<!-- Tab container should be focusable -->
<ol id="animalTabs" tabindex="0">
  <!-- Individual Tabs shouldn't be focusable -->
  <!-- We'll focus them with JavaScript instead -->
  <li id="tab1">
    <a href="#cats" tabindex="-1">Cats</a>
  </li>
  <li id="tab2">
    <a href="#cats" tabindex="-1">Dogs</a>
  </li>
  <li id="tab3">
    <a href="#cats" tabindex="-1">Alligators</a>
  </li>
</ol>
```

Making Things Tabbable

- Tabindex varies subtly across browsers
- jquery.attr() normalizes it as of 1.3
- For all the gory details:

  http://fluidproject.org/blog/2008/01/09/
  getting-setting-and-removing-tabindex-values-with-javascript/

```
// Make the tablist accessible with the Tab key.
tabContainer.attr("tabindex", "0");
// And take the anchors out of the Tab order.
$(“a”, tabs).attr("tabindex", "-1");
```

Adding the Arrow Keys

```
// Make each tab accessible with the left and right arrow keys.
tabContainer.fluid("selectable", {
    selectableSelector: that.options.selectors.tabs,
    direction: fluid.a11y.orientation.HORIZONTAL,
    onSelect: function (tab) {
        $(tab).addClass(that.options.styles.highlighted);
    },

    onUnselect: function (tab) {
        $(tab).removeClass(that.options.styles.highlighted);
    }
});
```

Making Them Activatable

```
// Make each tab activatable with Spacebar and Enter.
tabs.fluid("activatable", function (evt) {
    // Your handler code here. Maybe the same as .click()?
});
```

Documentation

- Tutorial:

  http://wiki.fluidproject.org/display/fluid/Keyboard+Accessibility+Tutorial

- API Reference:

  http://wiki.fluidproject.org/display/fluid/Keyboard+Accessibility+Plugin+API

Exercise 4: Accessibility

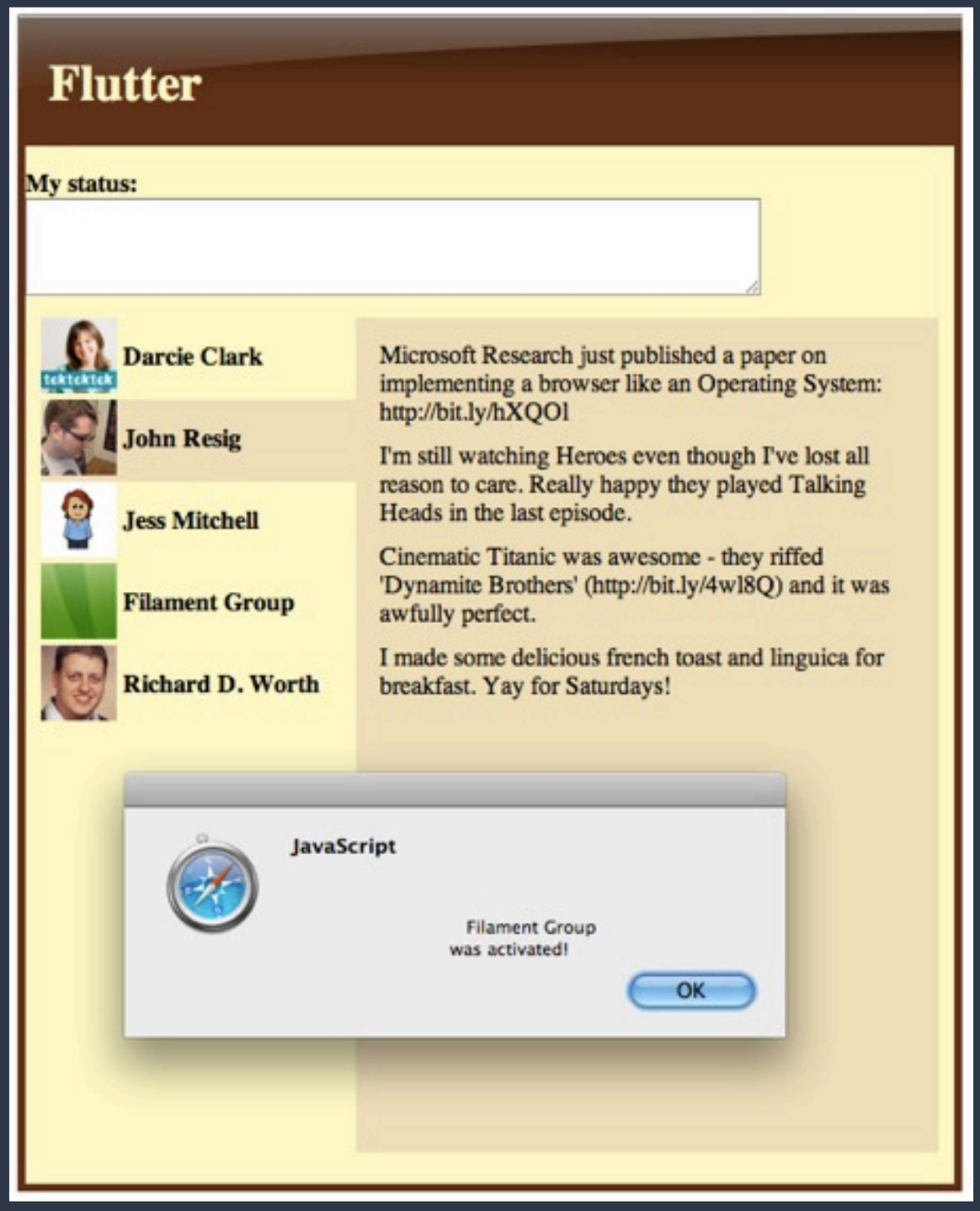
Make it More Accessible

The list of tweets is like a set of sideways tabs

- Put the friends list in the tab order
- Make each friend selectable with the arrow keys
- Make each friend activatable
    - show an alert when the friend is activated
- With ARIA, identify each friend as a tab
- Make the tweet list a tab panel

Fluid

Exercise 4 Illustrated



Fluid

AJAX

What is AJAX?

- A technique for making HTTP requests from JavaScript without reloading the entire page
- Asynchronous, meaning it doesn't block the UI
- The X stands for XML, but JSON is often more convenient
- The heart of Web 2.0: enables unprecedented dynamism on the Web

REST

- Just the way the Web works
- Resources are the nouns, referred to by URL
  - e.g. http://twitter.com/friends
- Representations define a format for a resource (eg. XML or JSON)
  - e.g. http://twitter.com/friends.json
- A small set of verbs:
  - GET: gets data from the server
  - POST: updates data on the server
  - DELETE, PUT

AJAX with jQuery

```
$.ajax({
  url: “http://twitter.com/friends”,
  type:”GET”,
  dataType: “json”, // “xml”, “json”, “html”, “script”
  data: { // Object containing query variables
    id: 123457
  },
  success: function (data) { }, // A callback upon success
  error: function () { } // Callback if an error occurs
});
```

jQuery UI & Plugins

jQuery Plugins

- Plugins extend jQuery's functionality
- No special API: they just mix themselves into the jQuery object directly
- Shared namespace: risky for portals
- Two types: community plugins and jQuery UI

Community Plugins

http://plugins.jquery.com/

- User-contributed
- Anything goes
- Some great stuff, some awful stuff

jQuery UI Plugins

http://ui.jquery.com/

- "Official" widgets for jQuery
- Generally more robust and reusable
- Developed by a community of experienced coders
- Accessibility
- Great themeing and skinning

jQuery UI Plugins

http://ui.jquery.com/

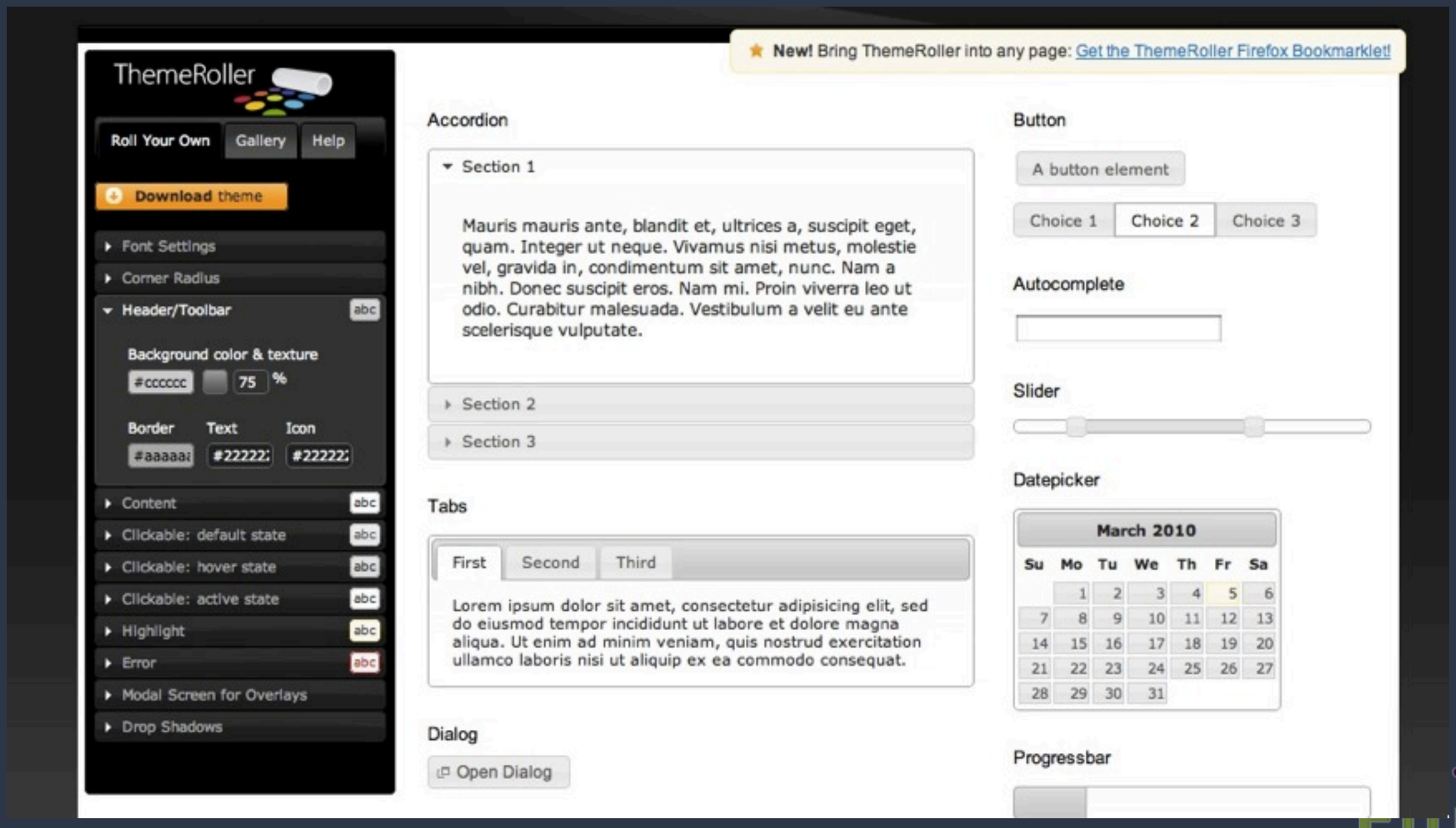
- Dialog
- Slider
- Tabs
- Accordion
- Date picker
- Autocomplete
- Progress bar
- Button

jQuery UI Dialog


```
var dialogRoot = $(“.settingsDialog”);
dialogRoot.dialog({
  modal: true
  buttons: {
    “Cancel”: function () {
       $(this).dialog(“close”);
    },
    “Settings”: function () {
        save();
        $(this).dialog(“close”);
    }
});

dialogRoot.dialog(“open”);
```

Theme Roller

http://jqueryui.com/themeroller/



Playing Nice With Others

Portals, Mashups, and CMS's

- These days, diverse code and markup coexists
- Most JavaScript is written as if it owns the whole browser
- As you combine stuff, things can break
- Namespacing and privacy is essential

Writing Collision-Free JavaScript

- Put code in a unique namespace
- Use closures for privacy
- Support more than one on the page
  - Scope all variables to an instance
  - Avoid hard-baking ID selectors
- Constrain selectors within a specific element

Questions?