jQuery: Getting the Ball Rolling

- How does a JavaScript or jQuery program know where to begin execution? Is there a main() or equivalent?
- no and yes
- As we've seen, JavaScript can be placed within the head or body element of an HTML document
- Code is executed as it is encountered by the browser
- However, we have to go behind the scenes to better appreciate why a "main()"-like method is needed …
- JavaScript or jQuery that interacts with the DOM can't run (properly) until the DOM is fully loaded, and that doesn't happen until some time after the entire document has downloaded from the server

jQuery: in the Beginning

- jQuery's solution to this timing issue is the ready() function, which is called after the DOM is loaded and ready for use:

  ```
  $(document).ready(function(){…})
  ```

  - which is often abbreviated as:

    ```
    $(function(){…})
    ```

  - in which we use jQuery ($) to wrap the (anonymous) function we want to run when the DOM is ready

- By putting your top-level jQuery function within a ready() or $() call, you ensure that the DOM will be loaded before your code tries to access it

jQuery: Collections

- The result of applying $() is always is a jQuery collection
- You can access a particular collection element with get(index), which returns the raw DOM element at that index. A shorthand notation for this is [index]
- Alternatively, you can index into a jQuery collection eq(index), which returns the collection element at that index, wrapped as a jQuery object
- This behavior has a tendency to trip people up, e.g.

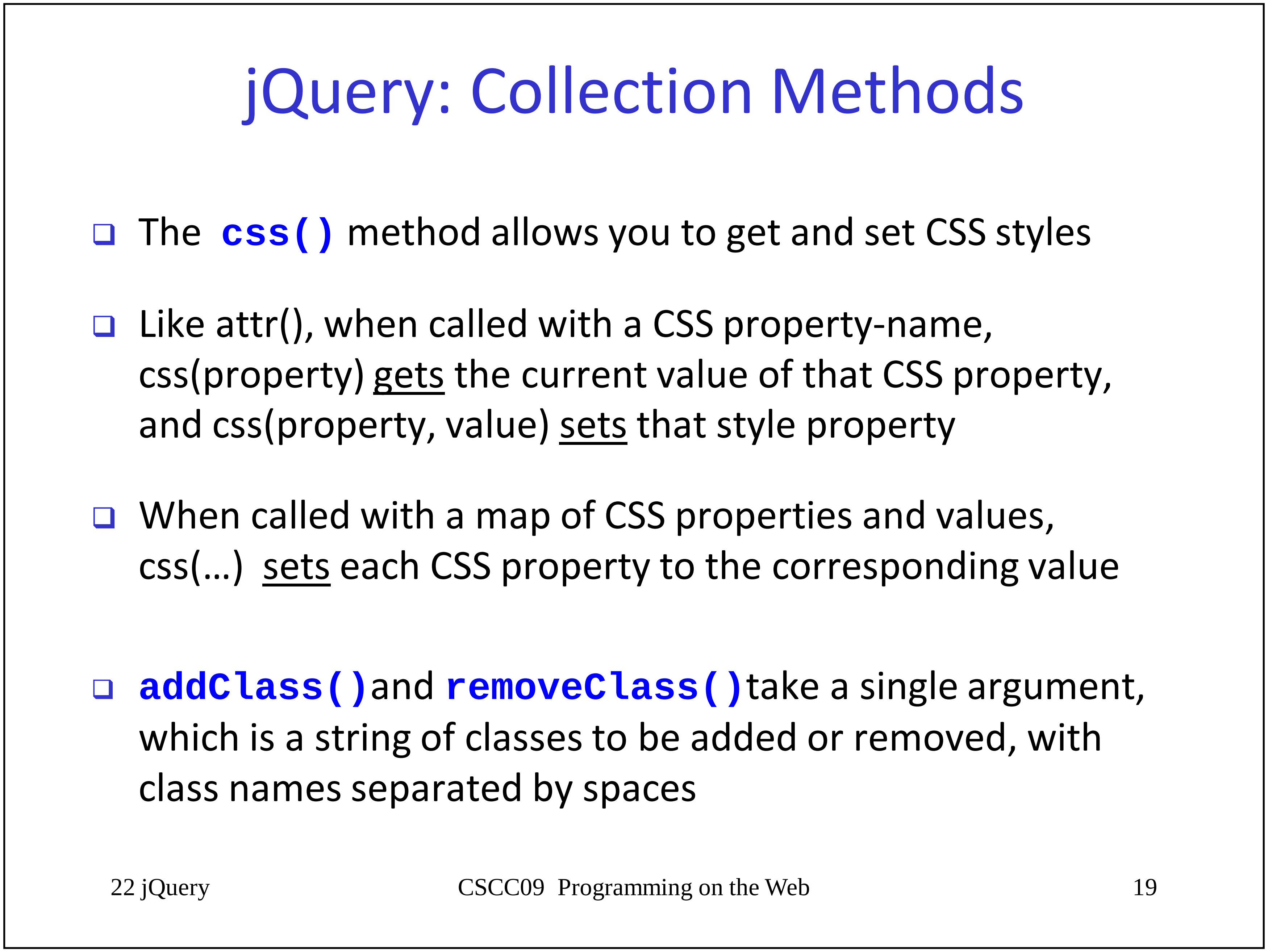
```
$(".thumbnail").get(0).fadeIn("slow");  // fails
$(".thumbnail").eq(0).fadeIn("slow");   // works
$(".thumbnail")[0].fadeIn("slow");   // fails
```

jQuery: Collection Methods

- The **`attr()`** method allows you to get and set HTML attributes, e.g.:

```
var title = $('#mydiv').attr("title")
```

- When called with an attribute name, the attr() method gets the current value of that attribute
- When called with an attribute name and a new value, attr() sets that attribute to the given value
- When called with a map of attribute names and new values, attr() sets each attribute to the corresponding new value

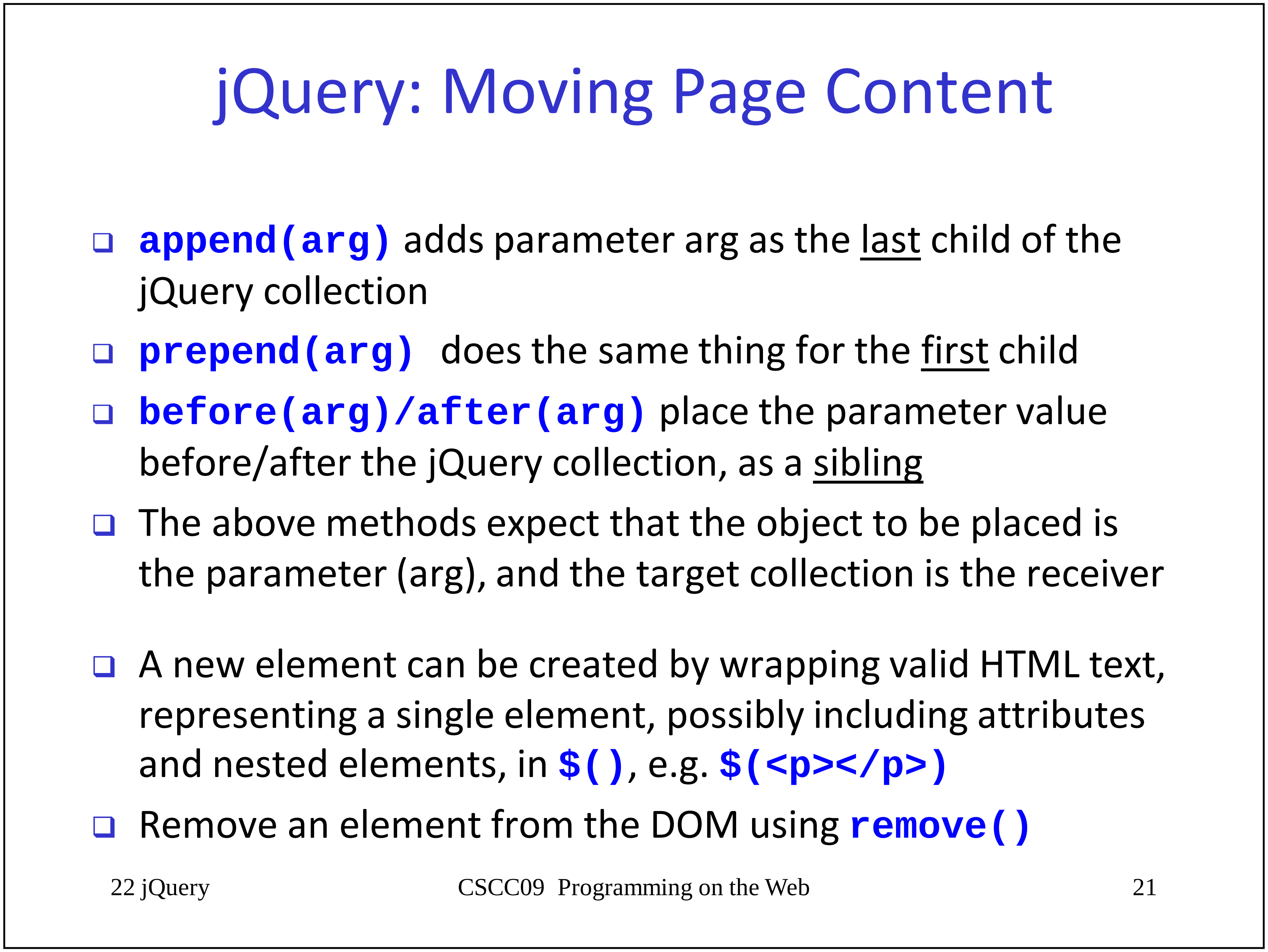


jQuery: Collection Methods

- The **`css()`** method allows you to get and set CSS styles
- Like attr(), when called with a CSS property-name, css(property) <u>gets</u> the current value of that CSS property, and css(property, value) <u>sets</u> that style property
- When called with a map of CSS properties and values, css(…) <u>sets</u> each CSS property to the corresponding value
- **`addClass()`** and **`removeClass()`** take a single argument, which is a string of classes to be added or removed, with class names separated by spaces

jQuery: Access/Modify Page Content

- To access/modify the content of a page element, use the **`html()`** or **`text()`** method
- As with attr() and css(), in the absence of a parameter, these methods get the current value of the element as a string, and with a parameter, they set the current value
  - html() treats its parameter as HTML
  - text() escapes its parameter (so, e.g. tag names become visible)
- Retrieve or set the value of an input element such as <input> or <select> using **`val()`** without or with args



jQuery: Moving Page Content

- **`append(arg)`** adds parameter arg as the last child of the jQuery collection
- **`prepend(arg)`** does the same thing for the first child
- **`before(arg)/after(arg)`** place the parameter value before/after the jQuery collection, as a sibling
- The above methods expect that the object to be placed is the parameter (arg), and the target collection is the receiver
- A new element can be created by wrapping valid HTML text, representing a single element, possibly including attributes and nested elements, in **`$()`**, e.g. **`$(<p></p>)`**
- Remove an element from the DOM using **`remove()`**

22 jQuery CSCC09 Programming on the Web 21

HTML Content Examples

- `object.html()`
  - gets (returns) jQuery object's innerHTML (the HTML content of the element)
- `object.html("<p>my html content</p>")`
  - sets jQuery object's html content to the parameter string
- `$(".important").before("<h1>Note</h1>");`
  - Inserts as a sibling node before all elements with a class of 'important' a new element "<h1>Note</h1>"

jQuery: Methods in Chains

- One of jQuery's powerful features is the ability to string together a sequence of actions as a chain, somewhat like a Unix command-pipeline (like ls –l | grep js | sort –r)
- When you call a method on a collection, it usually returns the collection itself, which enables you to call a series of successive methods on a collection. A pattern called chaining.
- Chaining cuts down on code size and improves readability, but it also improves performance, since jQuery does not have to repeat the sometimes-expensive action of selecting DOM elements that are involved in a sequence of actions

```
$(".note").css("background-color", "yellow").attr
      ("src", "help-url")  // can chain further ...
```

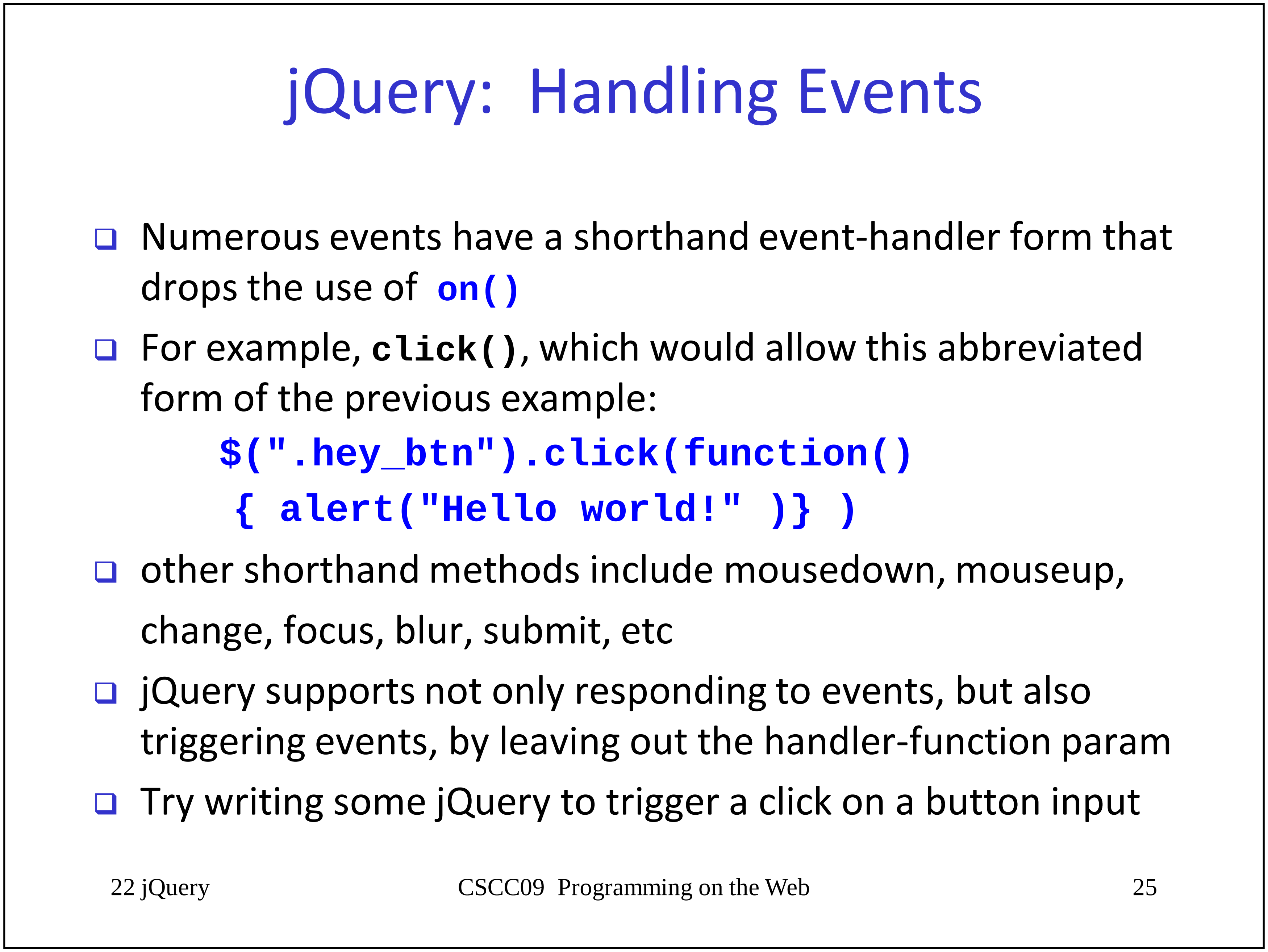
jQuery: Handling Events

- The jQuery's event listener function is called **`on()`**
- on() attaches a handler to an event for a jQuery collection of elements. If the collection has more than one element, the same listener is bound to each element in the collection

- For instance, bind an event-handler to the "click" event on all buttons with class "hey_btn":

```
$(".hey_btn").on("click", function()
    { alert("Hello world!" )} )
```

- The first parameter to on() is a string-named event, and the second parameter is a "callback" function.
- To remove an event handler use off() with the same param's



jQuery: Handling Events

- Numerous events have a shorthand event-handler form that drops the use of `on()`
- For example, **`click()`**, which would allow this abbreviated form of the previous example:

```
$(".hey_btn").click(function()
 { alert("Hello world!" )} )
```

- other shorthand methods include mousedown, mouseup, change, focus, blur, submit, etc
- jQuery supports not only responding to events, but also triggering events, by leaving out the handler-function param
- Try writing some jQuery to trigger a click on a button input

22 jQuery CSCC09 Programming on the Web 25

jQuery: Events on Future Elements

- One subtlety to event-handling thus far is that it applies only to elements that exist in the DOM at the time the event handler is set up
- Is it possible to bind an event handler to a DOM element that will exist in the future? Is this potentially useful?
- A variation on the on() method, supports dynamic binding to both current and future DOM elements; here's the previous example expressed to capture this behavior:

```
$("body").on("click", ".hey_btn",
   function() { alert("Hello world!" )} )
```

- This version of the on() method relies on propagation of events from "hey_btn" class elements to the body element

jQuery: Event Properties

- This dynamic version of the on() method relies on event propagation or “bubbling”; before we examine bubbling, we need a bit more background on the JS event object
- First every event has a type that is passed to the event-handler, for example, a keyboard key-press, or a mouse click, or a page load. The type of an event can be retrieved via the **`type`** property of the event object.
- Most events also have a <u>target</u>, an object representing the element that the event was triggered on.
- For example, a user typing into an input textbox triggers keyboard button-press events and that textbox is the <u>target</u> of the events; access with the event object’s <u>target</u> property

jQuery: Event Default Actions

- To respond to implied behavior many events have a default action.
- For example, clicking a hyperlink will cause the browser to navigate to the target page by default, and clicking a form submit button will submit the form to the server by default.

- If an event-listener is defined for an event with a default action, that listener will fire before the default action is run.
- This gives the event listener the opportunity to prevent the default action.
- Why might you want to prevent such useful behavior?

jQuery: Event Default Actions

- Why might you want to prevent such useful behavior?
- Sometimes, you want to handle things your own way, not the default way
- A situation that arises commonly in SPA's is that a user fills out a form, and when they submit the form, you want to grab the form data for an Ajax call whose response will then be used to effect a DOM update
- Great, except that the default action will then submit the form to a server-side handler, which will return a new page for display by the browser!
- That's how forms were meant to work, in the document-Web

jQuery: Event Default Actions

- OK, so how do we prevent this kind of default action?
- First, the event object that is passed as a parameter to an event-handler function has a method `preventDefault()`
- If the event handler calls this method, the default action will be cancelled
- Alternatively, if the handler-function returns value “false” the default action will be cancelled.

jQuery: Event Propagation

- OK, now back to explaining how **on( )** can support events on elements that are added during execution
- Suppose you define an input-button element that is within a div element that in turn is within a body element
- Furthermore, suppose that you have defined click handlers on the body, div, and input elements
- If you click on the button element, which click handler(s) should fire: the one for click the button, the div element, or the body element?
- Yes

jQuery: Event Propagation

- When you click the button, the button (target element) event-handler fires first, then the event-handler for the parent div element fires, and finally the event handler for the body element fires
- We refer to this as event propagation or event bubbling
- Each of the handlers in this sequence has the opportunity to stop the event from propagating to its parent by calling the **`stopPropagation()`** method on the event object
- In the case of events triggered on dynamically-added elements, those events propagate up to some higher-level static element, e.g. <body> where a static-handler is defined, and it can then filter for just the desired element(s)'s event

jQuery vs POJS

- Let’s consider a few simple examples to compare how things are done in jQuery vs plain old JavaScript (POJS) with the Document Object Model (DOM)
- Suppose we want to associate a click event with each link within a particular area of a page:
- using JavaScript and DOM:

```
// confirm user hypertext links upon click
var external_links =
      document.getElementById('external_links');
var links = external_links.getElementsByTagName('a');
for (var i=0;i < links.length;i++) {
  var link = links.item(i);
  link.onclick = function() {
   return confirm('You are going to visit: ' + this.href);
  };
}
```

jQuery vs POJS

- using JavaScript and DOM:

```
var external_links =
    document.getElementById('external_links');
var links = external_links.getElementsByTagName('a');
for (var i=0;i < links.length;i++) {
  var link = links.item(i);
  link.onclick = function() {
    return confirm('You are going to visit: ' +
       this.href);
  };
}
```

- using jQuery:

```
$('#external_links a').click(function() {
   return confirm('You are going to visit: ' +
              this.href);
});
```

jQuery vs POJS

- Suppose we want to create a new paragraph <p> and add it at the end of the existing page:
- using JavaScript DOM:

```
var new_p = document.createElement("p");
var new_text = document.createTextNode("Hello World");
new_p.appendChild(new_text);
var bodyRef =
   document.getElementsByTagName("body").item(0);
bodyRef.appendChild(new_p);
```

- Typical POJS pattern: incrementally build elements and glue (append) them together; only when glued to current document do the new elements become visible

jQuery vs POJS

- using JavaScript DOM:

```
 var newPP = document.createElement("p");
var newTxt = document.createTextNode("Hello World");
newPP.appendChild(newTxt);
var bodyRef =
    document.getElementsByTagName("body").item(0);
bodyRef.appendChild(newPP);
```

- using jQuery:

```
$('<p></p>').html('Hello World!').appendTo('body');
```

  - Create an element by quoting the HTML (‘<p></p>’)
  - Set it’s content by calling html() with parameter value content
  - Add it to the end of the document (appendTo body element)
  - jQuery excels at “chaining” together actions in a fluid way

jQuery vs POJS

- Suppose we want to bind a handler-function to an event associated with a particular element.
- using JavaScript DOM we use an on-event attribute to bind builtin function alert to a click event:

```
<a href="" onclick="alert('Hello world')">hey</a>
```

- using jQuery:

```
 $(document).ready(function() {
     $("a").click(
         function() { alert("Hello world!"); }
     );
  });
```

- wait a minute, how is that an improvement?!

jQuery vs POJS

```
<a href="" onclick="alert('Hello world')">hey</a>
```

- versus

```
$(document).ready(function() {
   $("a").click(
       function() { alert("Hello world!"); }
   );
 });
```

- The difference is that the jQuery version is handling every single **<a>** element, POJS just a single element
- Just as CSS separates presentation from structure, jQuery separates behavior from structure.
- This is one of the key insights that has made jQuery such a powerful & successful JavaScript library.

Using jQuery

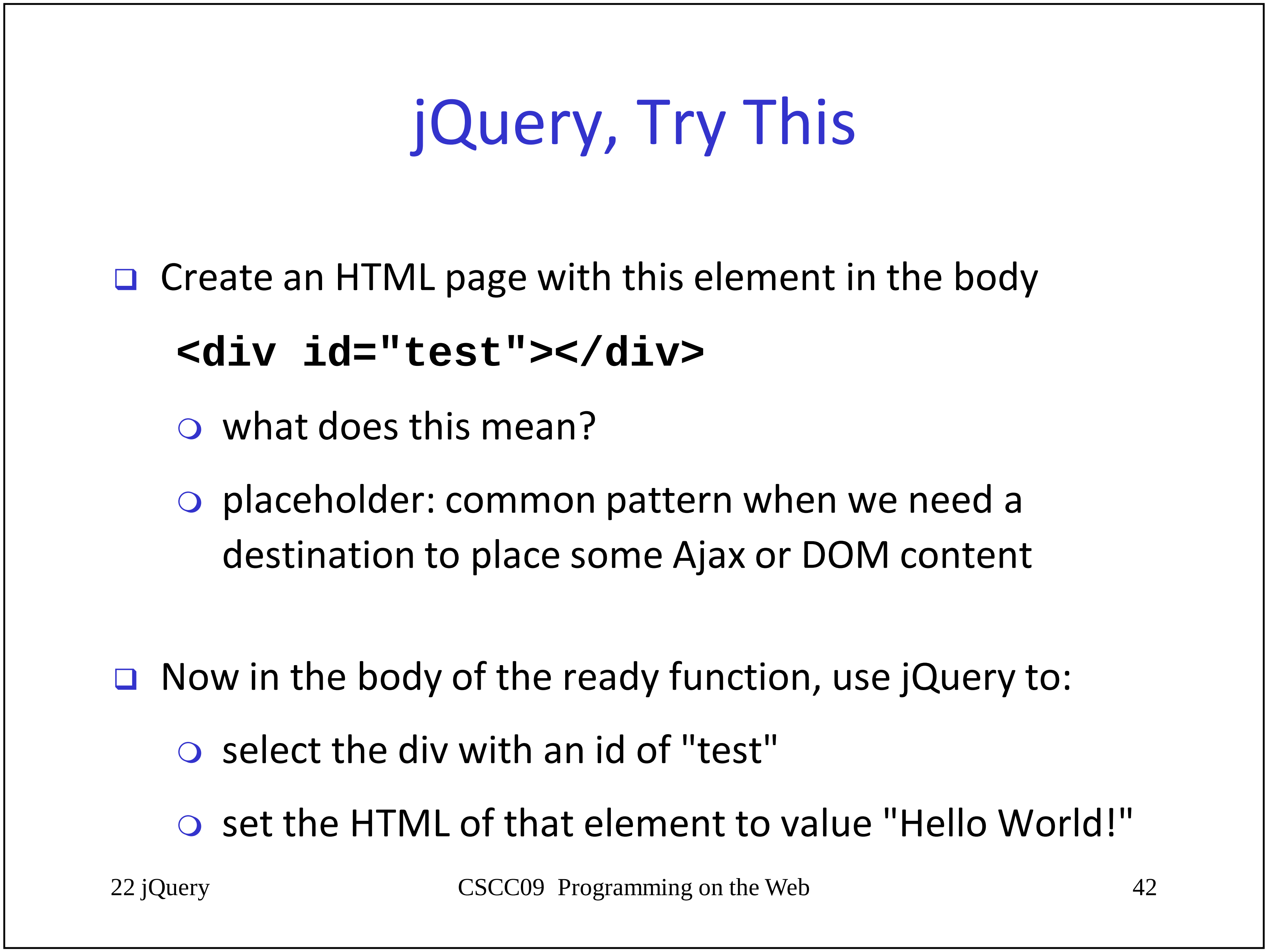
- Link to the jQuery source
  - local copy or remote master URL:

```
<script type ="text/javascript"
  src="http://code.jquery.com/jquery-latest.js">
</script>
```

(or http://code.jquery.com/jquery-latest.min.js "minified" version to reduce download time**)**

- To execute your jQuery code, put it in the ready function (inside a script element)

```
$(document).ready(function(){
    // jQuery code goes here
});
```



jQuery, Try This

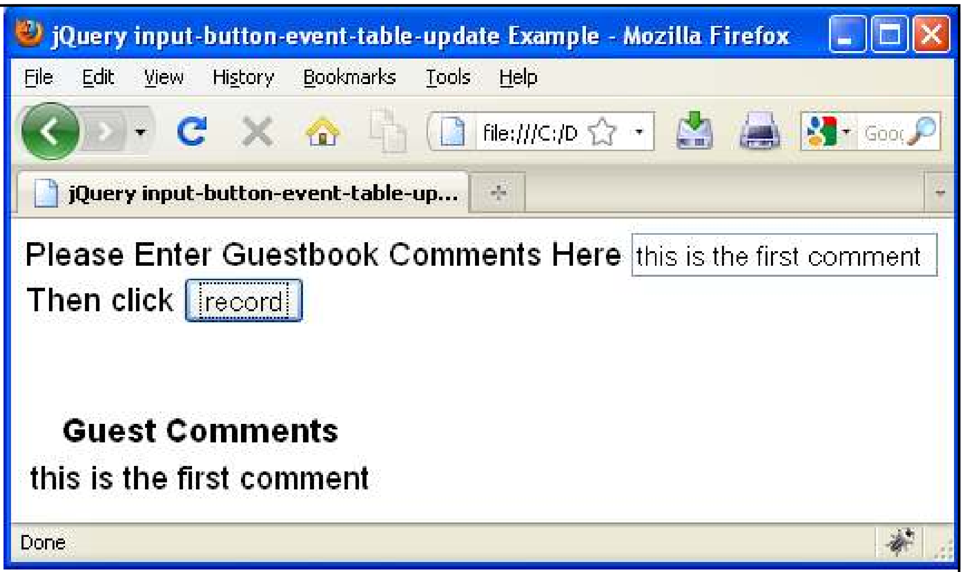
- Create an HTML page with this element in the body

  ```
  <div id="test"></div>
  ```

  - what does this mean?
  - placeholder: common pattern when we need a destination to place some Ajax or DOM content

- Now in the body of the ready function, use jQuery to:
  - select the div with an id of "test"
  - set the HTML of that element to value "Hello World!"

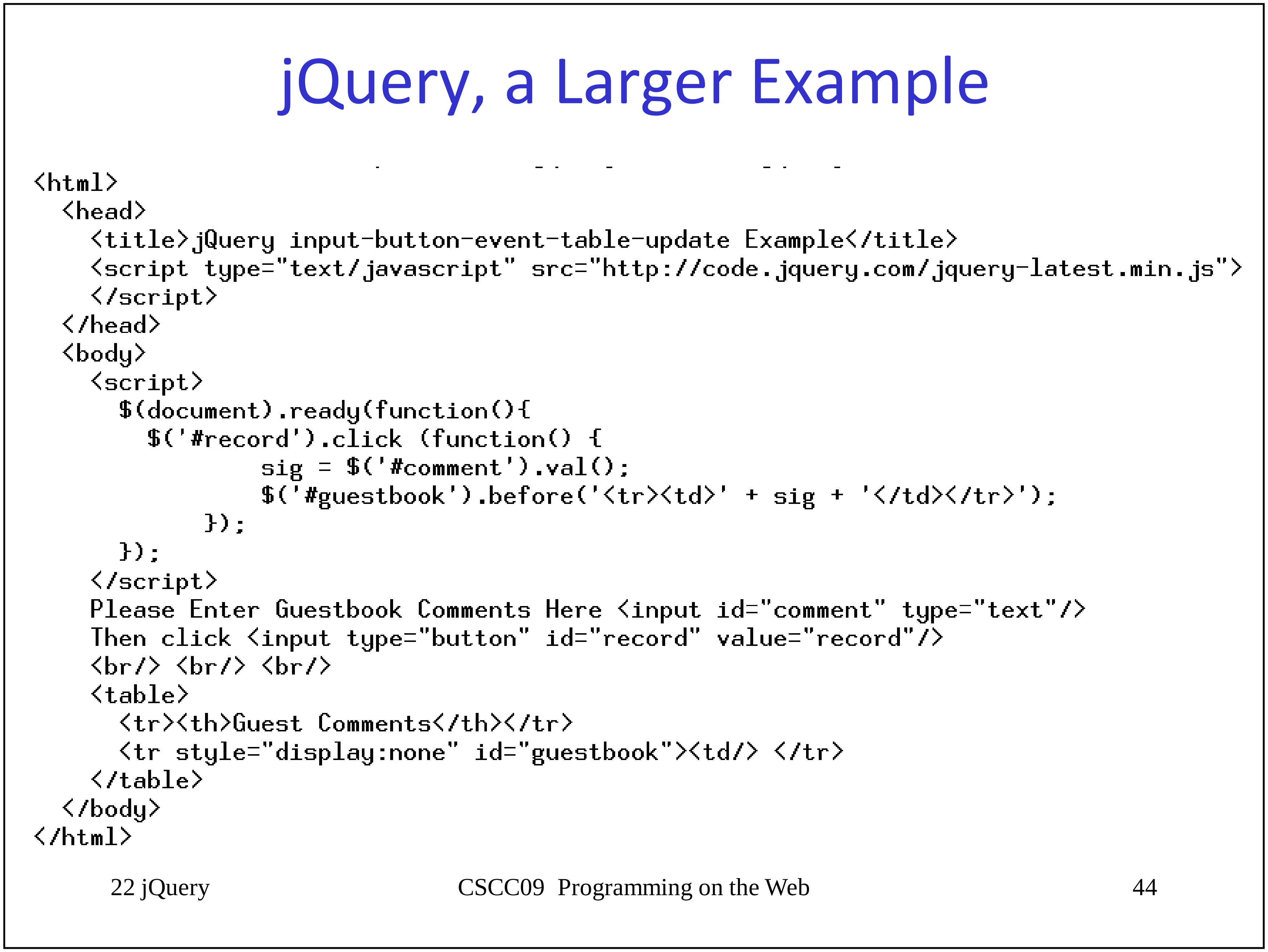
jQuery, a Larger Example: events with DOM manipulation


jQuery input-button-event-table-update Example - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/D
jQuery input-button-event-table-up...
Done

Please Enter Guestbook Comments Here this is the first comment
Then click record

Guest Comments
this is the first comment

- Let's raise the bar a bit by doing something practical
- Create a Web page with an input text field for users to enter brief guestbook comments
- Provide a button that when clicked will transfer the currently entered comment to a table where comments are collected.



jQuery, a Larger Example

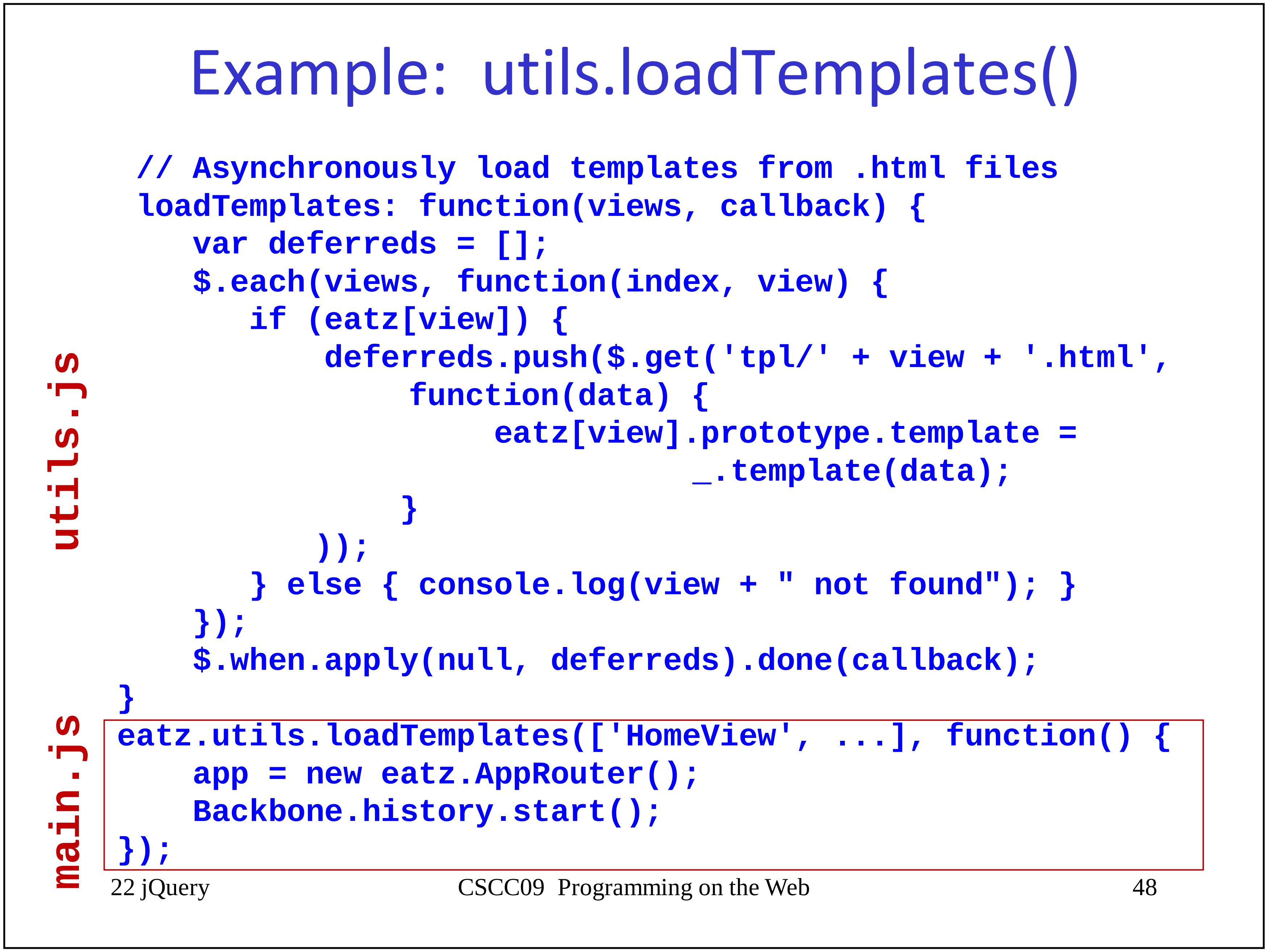
```
<html>
  <head>
    <title>jQuery input-button-event-table-update Example</title>
    <script type="text/javascript" src="http://code.jquery.com/jquery-latest.min.js">
    </script>
  </head>
  <body>
    <script>
      $(document).ready(function(){
        $('#record').click (function() {
                sig = $('#comment').val();
                $('#guestbook').before('<tr><td>' + sig + '</td></tr>');
            });
      });
    </script>
    Please Enter Guestbook Comments Here <input id="comment" type="text"/>
    Then click <input type="button" id="record" value="record"/>
    <br/> <br/> <br/>
    <table>
      <tr><th>Guest Comments</th></tr>
      <tr style="display:none" id="guestbook"><td/> </tr>
    </table>
  </body>
</html>
```

22 jQuery CSCC09 Programming on the Web 44

Ajax Requests

- A typical jQuery Ajax callback function binds the Ajax result value to the callback parameter, as in:
  - `$.getJSON("… URL …", function(data) { … });`
  - Within the anonymous callback-function, **`data`** refers to the value returned by the callback
- If the Ajax result is a dictionary, you can iterate over the items in this dictionary using:
  - `$.each(data, function(key, value) { … });`
  - Now, within the callback function, you can refer to each dictionary entry by its key and its value components

```
$.getJSON("… URL …", function(data) {
    …
    $.each(data, function(key, value) { … });
    …
});
```



Example: utils.loadTemplates()

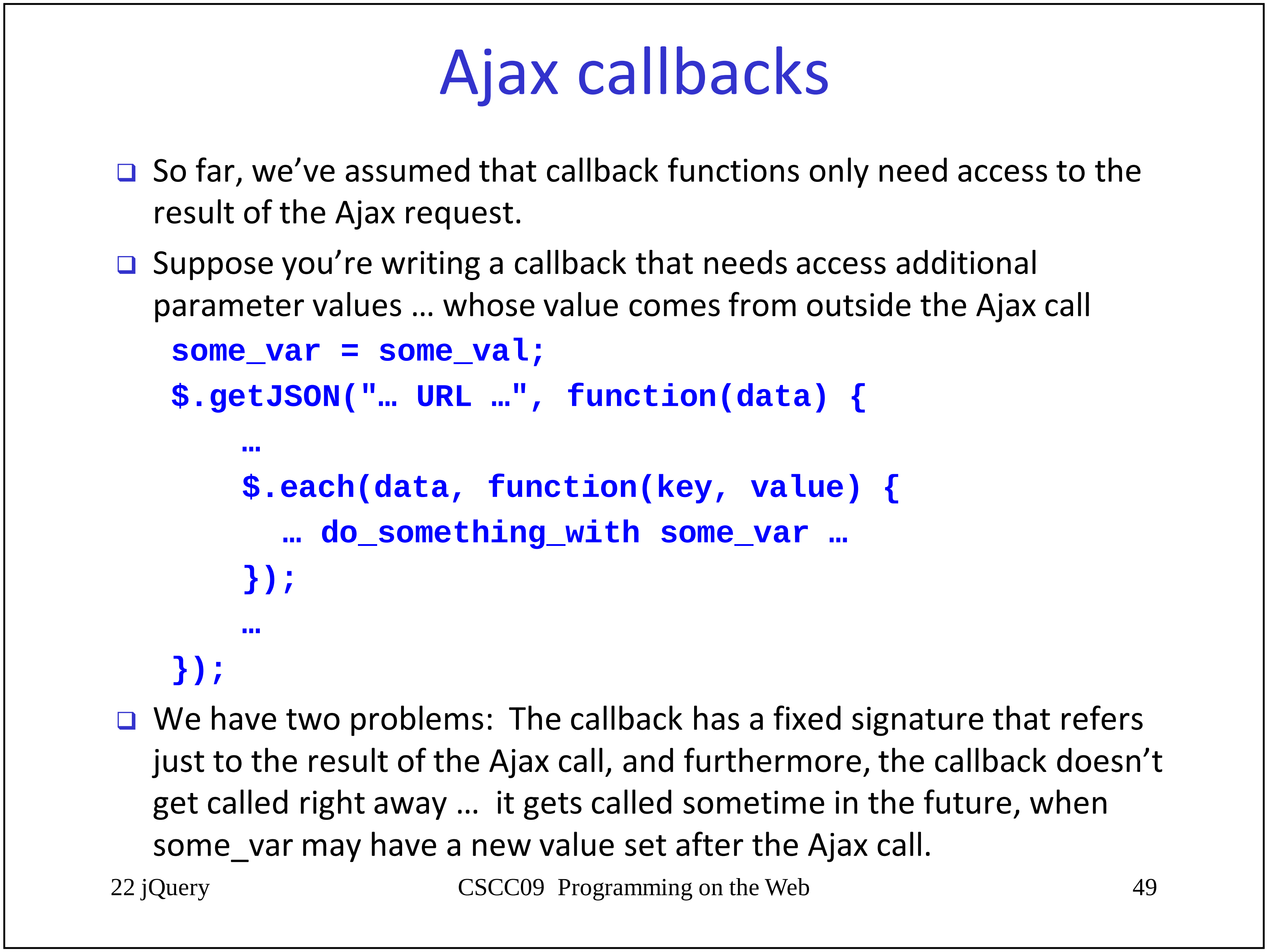
utils.js

```
// Asynchronously load templates from .html files
loadTemplates: function(views, callback) {
   var deferreds = [];
   $.each(views, function(index, view) {
      if (eatz[view]) {
          deferreds.push($.get('tpl/' + view + '.html',
               function(data) {
                   eatz[view].prototype.template =
                             _.template(data);
               }
          ));
      } else { console.log(view + " not found"); }
   });
   $.when.apply(null, deferreds).done(callback);
}
```

main.js

```
eatz.utils.loadTemplates(['HomeView', ...], function() {
    app = new eatz.AppRouter();
    Backbone.history.start();
});
```

22 jQuery CSCC09 Programming on the Web 48



Ajax callbacks

- So far, we've assumed that callback functions only need access to the result of the Ajax request.
- Suppose you're writing a callback that needs access additional parameter values … whose value comes from outside the Ajax call

```
some_var = some_val;
$.getJSON("… URL …", function(data) {
    …
    $.each(data, function(key, value) {
      … do_something_with some_var …
    });
    …
});
```

- We have two problems: The callback has a fixed signature that refers just to the result of the Ajax call, and furthermore, the callback doesn't get called right away … it gets called sometime in the future, when some_var may have a new value set after the Ajax call.

jQuery Ajax callbacks

- Does this mean callbacks can’t take extra parameters and can’t refer to data in existence at the time of the Ajax call?
- Fortunately not; we can use JavaScript’s first-class functions to bind a handler with extra parameters to the callback, and grab the value of an in-scope variable, for later reference.

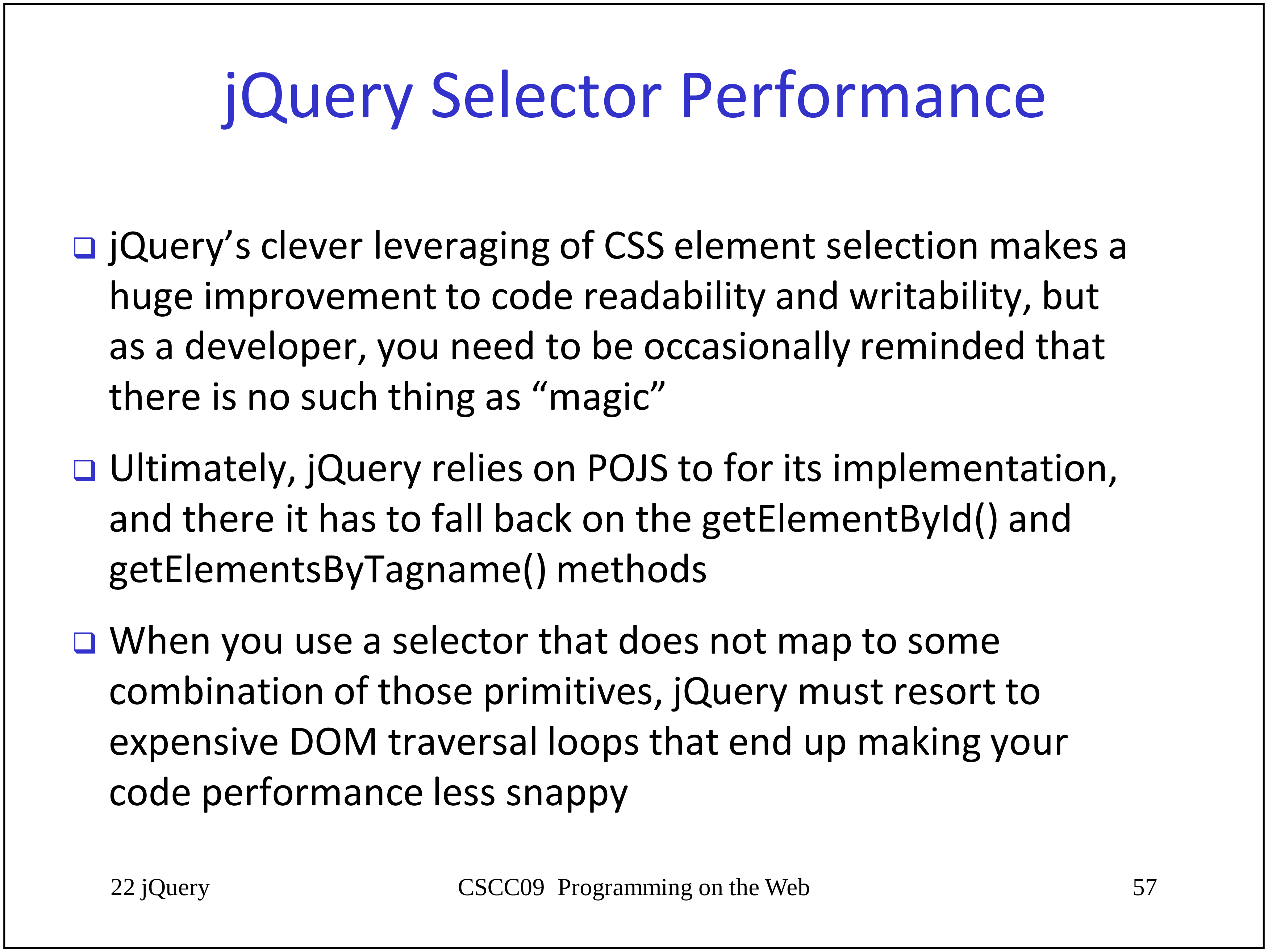
```
 id = some_val;
$.getJSON("… URL …", runCallback(id));
function runCallback(id) {
   return function(data) {
      processRoutes(data, id);
   };
}
function processRoutes(data, route_id) { ... }
```

jQuery Selector on DOM Subtree

- So far, we have looked at jQuery selectors that implicitly apply to the active document
- Sometimes you want to query on a subset of the active document or an independent DOM tree that isn't part of the active document (e.g. a Backbone.render().el result)
- Supply 2nd "context" parameter after the selector to identify the DOM tree to process: $(selector,context), e.g.

```
$(".myclass", DOMtree).jQueryMethod()…
```

- The context can be a DOM element, a document, or a jQuery object



jQuery Selector Performance

- jQuery's clever leveraging of CSS element selection makes a huge improvement to code readability and writability, but as a developer, you need to be occasionally reminded that there is no such thing as "magic"
- Ultimately, jQuery relies on POJS to for its implementation, and there it has to fall back on the getElementById() and getElementsByTagname() methods
- When you use a selector that does not map to some combination of those primitives, jQuery must resort to expensive DOM traversal loops that end up making your code performance less snappy

jQuery Selector Performance

- A few things to watch out for:
  - Id-selector’s with redundant qualifiers (e.g. ancestor or parent classes) – these achieve nothing and cost time
  - Trading off jQuery selectors against explicit document attributes – jQuery is not a substitute for appropriate placement of element attributes – use “id” and “class” attributes wisely to make selector searches more efficient
  - Avoid recurring selectors, since each gets wrapped in a jQuery object instance with a sizeable footprint. Chaining can help here – rather than repeating a selector, select once, then chain actions on it.