

MIDTERM REVIEW

NEXT MONDAY:
IN-CLASS MIDTERM

CANNOT MAKE IT?

If for some special circumstance, you CANNOT make the in-class midterm, please email me ASAP.

If we don't know about your special circumstance by 11.59pm Monday, Oct 24th, we won't be able to accommodate you.

THINGS YOU SHOULD KNOW ABOUT THE MIDTERM

Anything from lecture and MPs is fair game

One-sheet of handwritten notes (front and back)

Expect to write code: **Javascript**, HTML, CSS, SASS, JQuery, AngularJS, Mongo Query Language

Will test your ability to apply what you've learned in new situations -- NOT regurgitate memorized facts (i.e., history of HTML)

HOW TO STUDY FOR MIDTERM

Go through all the questions on slides

Go through all code examples on slides/CODE PEN

Review the challenging aspects of the MPs

*Be sure to review the
following topics...*

STRUCTURAL SEMANTIC TAGS

```
<body>
  <header>
    <h1>How to Get a PhD</h1>
    <nav>...</nav>
  </header>
  <article>
    <section>
      <figure></figure>
      <h3>Bribing your Committee</h3>
      <p>When blackmail fails...</p>
    </section>
    <aside>
      <h4>Useful Links</h4>
      <a href="www.bevmo.com">Research Supplies</a>
    </aside>
  </article>
</body>
```

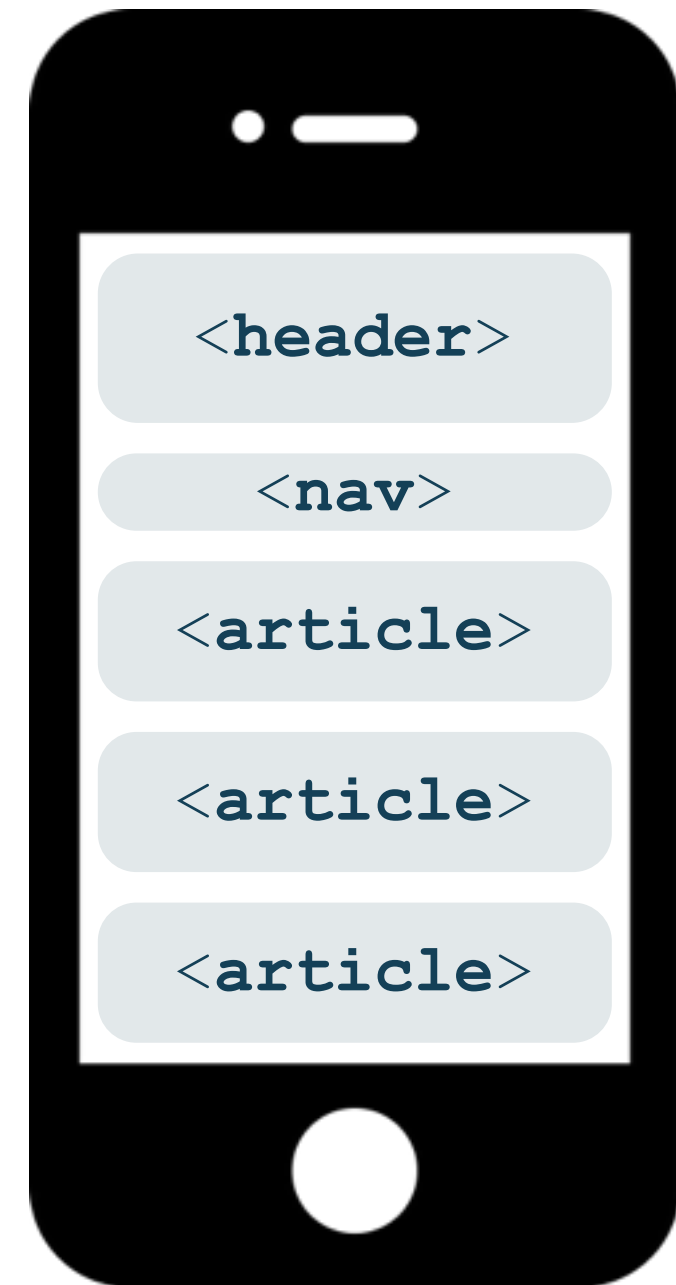
STRUCTURAL SEMANTIC APPLICATIONS?

STRUCTURAL SEMANTIC APPLICATIONS

Reuse stylesheets

Remix pages and applications

Retarget between form factors



CSS SELECTORS

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

```
<body>
```

```
  <div class="photo">
```

```
    <h3>My first photo</h3>
```

```
    
```

```
  </div>
```

```
...
```

```
</body>
```

```
</html>
```

```
.photo {  
  width:300px;  
}
```

```
.photo h3 {  
  font-weight:bold;  
}
```

```
img {  
  border:1px solid black;  
}
```

```
...
```

map HTML elements to CSS rules

Which selectors promote
the most *reuse*?

WHY CASCADING?

more than one rule can apply to an HTML element

priority rules for resolving conflicts

more *specific* = higher priority (class trumps element)

some properties (**font-size**) are inherited, while others aren't (**border, background**)

LINKING TO HTML

(1) `<link rel="stylesheet" href="gallery.css" type="text/css"/>`

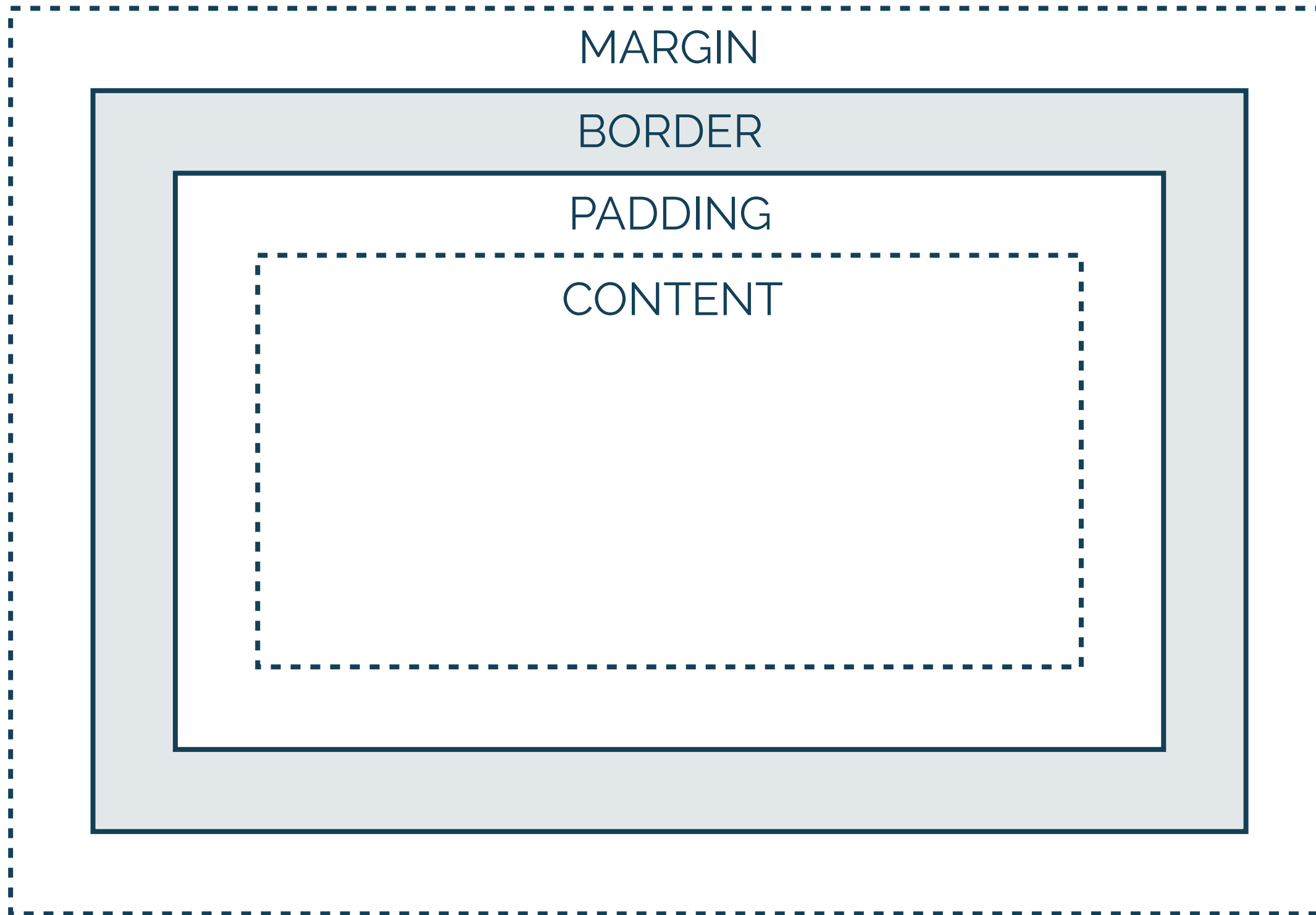
(2)

```
<html>
  <head>
    <style>
      h1 {color:red;}
      p {color:blue;}
    </style>
```

(3) `<div style="color:blue;text-align:center">`

higher priority

Box Model



control over
white space

position

	VALUE	DESCRIPTION
	static	default. positioned by the flow model; unaffected by top, bottom, left, right
<i>use with</i> top bottom left right	fixed	positioned relative to browser window; will not move when window is scrolled
	relative	positioned relative to its normal position
	absolute	positioned relative to the first ancestor where position!=static

Design Challenge:
vertically center a `<div>`
of unknown height

CODEPEN

SOLUTION

```
<div class="table-outer">  
  <div class="outer">  
    <div class="inner">  
    </div>  
  </div>  
</div>
```

```
.table-outer {  
  width: 100%;  
  display: table;  
}
```

```
.outer {  
  height: 200px;  
  background-color: #144057;  
  
  display: table-cell;  
  vertical-align: middle;  
}
```

```
.inner {  
  width: 100px;  
  height: 50%;  
  background-color: #B6C4C9;  
}
```

css tables!



Separation of CONTENT from PRESENTATION?

purely presentational html!

```
<div class="table-outer">  
  <div class="outer">  
    <div class="inner"></div>  
  </div>  
</div>
```

a lot of HTML suffers from presentational `div` bloat

CSS PREPROCESSORS

languages that extend CSS in meaningful ways

features: variables, nesting, mixins, inheritance

shrinks developer's codebase and compiles into CSS

popular CSS preprocessors: LESS and SASS

JAVA : JAVASCRIPT ::



Functions are first-class objects

FUNCTIONS ARE OBJECTS

that are callable!

reference by variables, properties of
objects

pass as arguments to functions

return as values from functions

can have properties and other functions

ANONYMOUS FUNCTIONS

create a function for later use

store it in a variable or method of an object

use it as a callback



see more examples next class

this

the other implicit parameter

a.k.a. **function context**

object that is implicitly associated
with a function's invocation

defined by how the function is
invoked (not like Java)

`apply()` *and* `call()`

two methods that exist for every function

explicitly define function context

`apply(functionContext, arrayOfArgs)`

`call(functionContext, arg1, arg2, ...)`

implemented in Javascript 1.6



```
function forEach(list, callback) {  
    for (var n = 0; n < list.length; n++) {  
        callback.call(list[n], n);  
    }  
}
```

```
var numbers = [5, 3, 2, 6];  
forEach(numbers, function(index) {  
    numbers[index] = this * 2; });  
console.log(numbers);
```

Classes are defined through functions

OBJECT-ORIENTED PROGRAMMING


new operator applied to a constructor function creates a new object

no traditional class definition

newly created object is passed to the constructor as this parameter, becoming the constructor's function context

constructor returns the new object

CONSTRUCTOR INVOCATION



```
function Llama() {  
  this.spitted = false;  
  this.spit = function() { this.spitted = true; }  
}
```

constructors are given the class name

```
var llama1 = new Llama();  
llama1.spit();  
console.log(llama1.spitted); true
```

```
var llama2 = new Llama();  
console.log(llama2.spitted); false
```

scopes are declared through
functions and not blocks { }

closure *scope created when a function is declared that allows the function to access and manipulate variables that are external to that function*

PRIVATE VARIABLES

```
var add = (function () {
```

```
    var counter = 0;
```

```
    return function () {return  
        counter += 1;}
```

```
})();
```

```
add();
```

self-invoking

PRIVATE VARIABLES

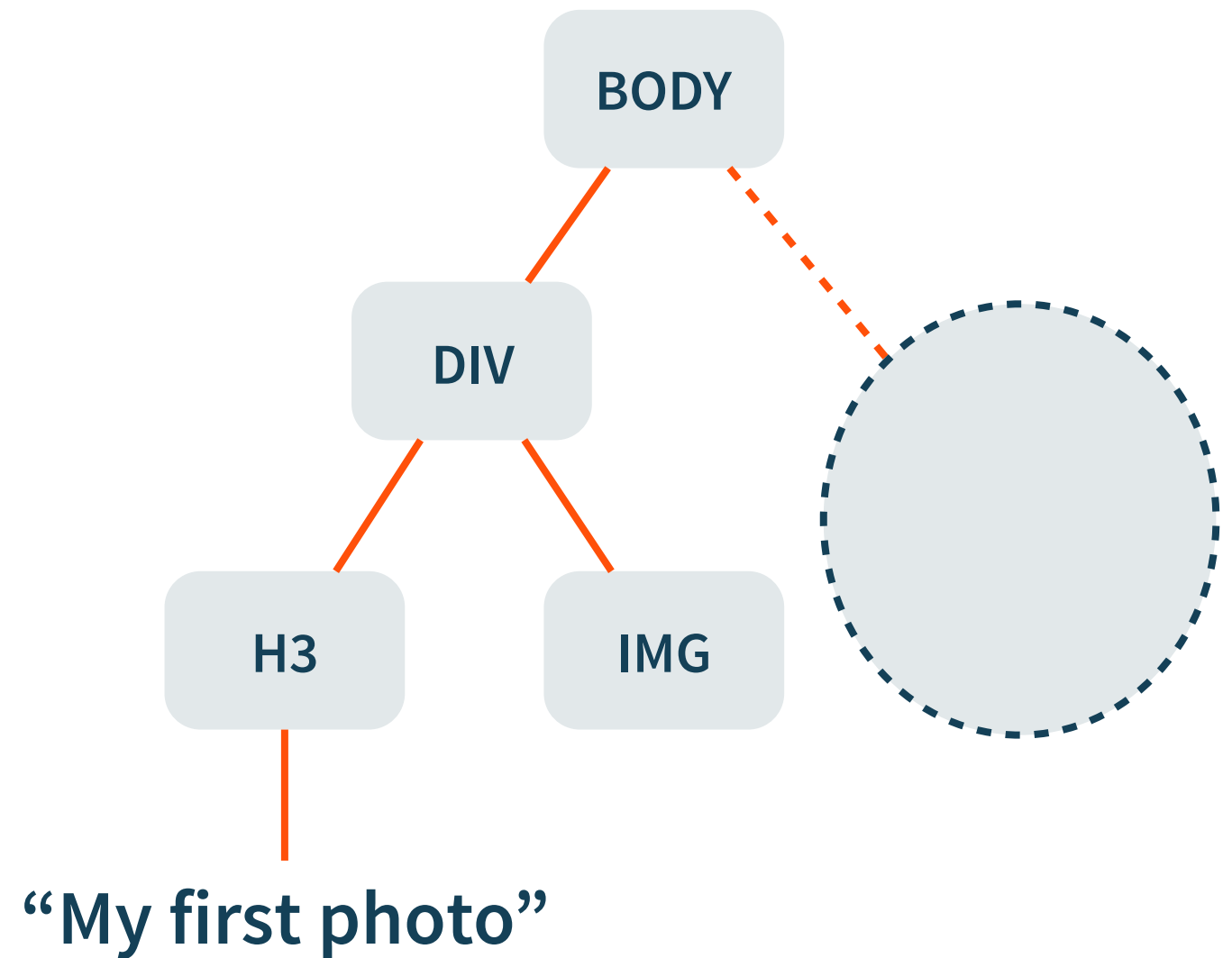
```
function Llama() {  
  var spitted = false;  
  this.spit = function() { spitted =  
    true; }  
  this.hasSpitted = function() { return  
    spitted; }  
}
```

private data member now!

DOCUMENT OBJECT MODEL

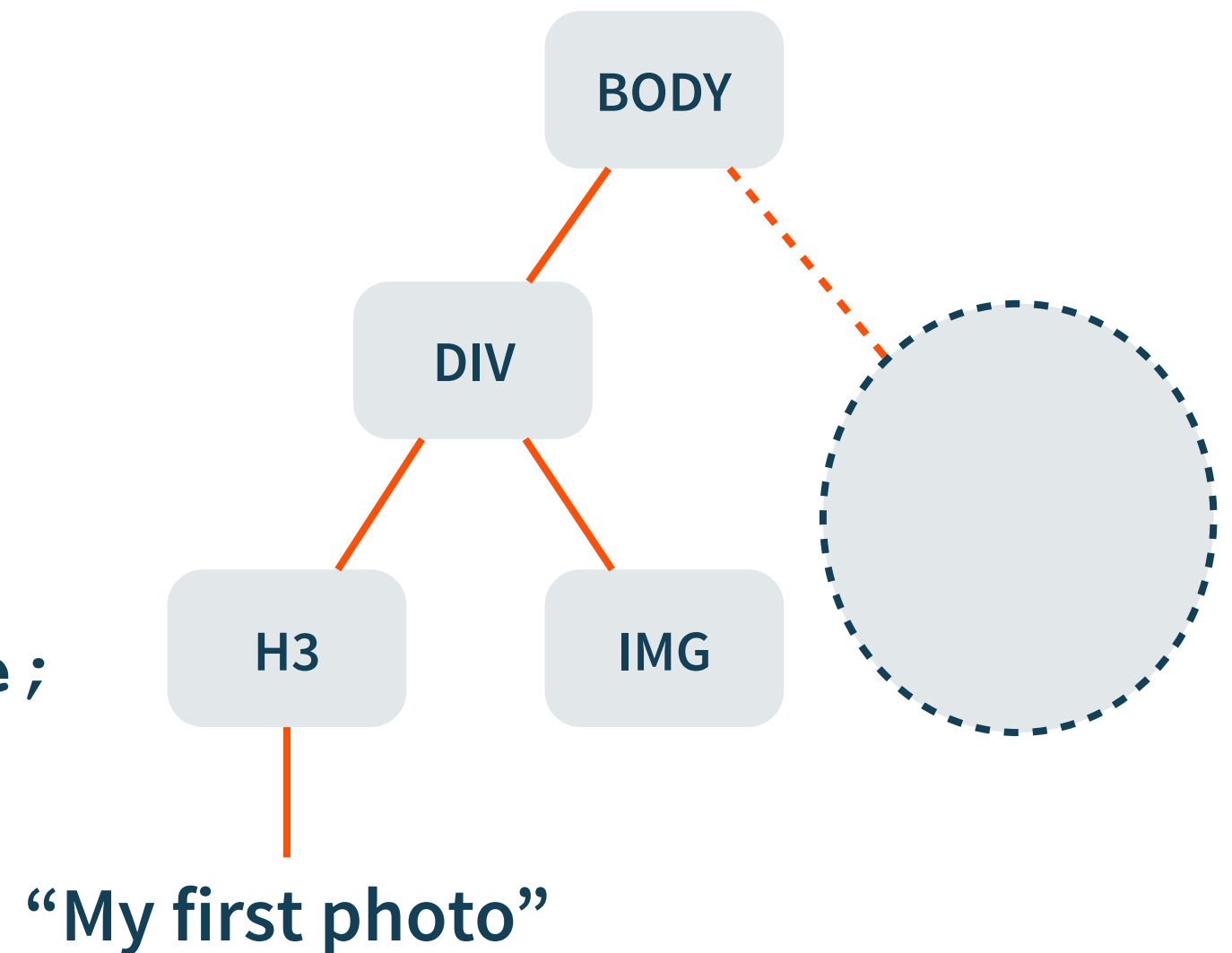
one-to-one correspondence between HTML elements and DOM nodes

```
<body>
  <div class="photo">
    <h3>My first photo</h3>
    
  </div>
  ...
</body>
```



TRAVERSING THE DOM

```
var body = document.body;  
var div = body.children[0];  
var h3 = div.children[0];  
var textNode = h3.childNodes[0];  
var textString = textNode.nodeValue;
```



DOM ELEMENT OBJECT

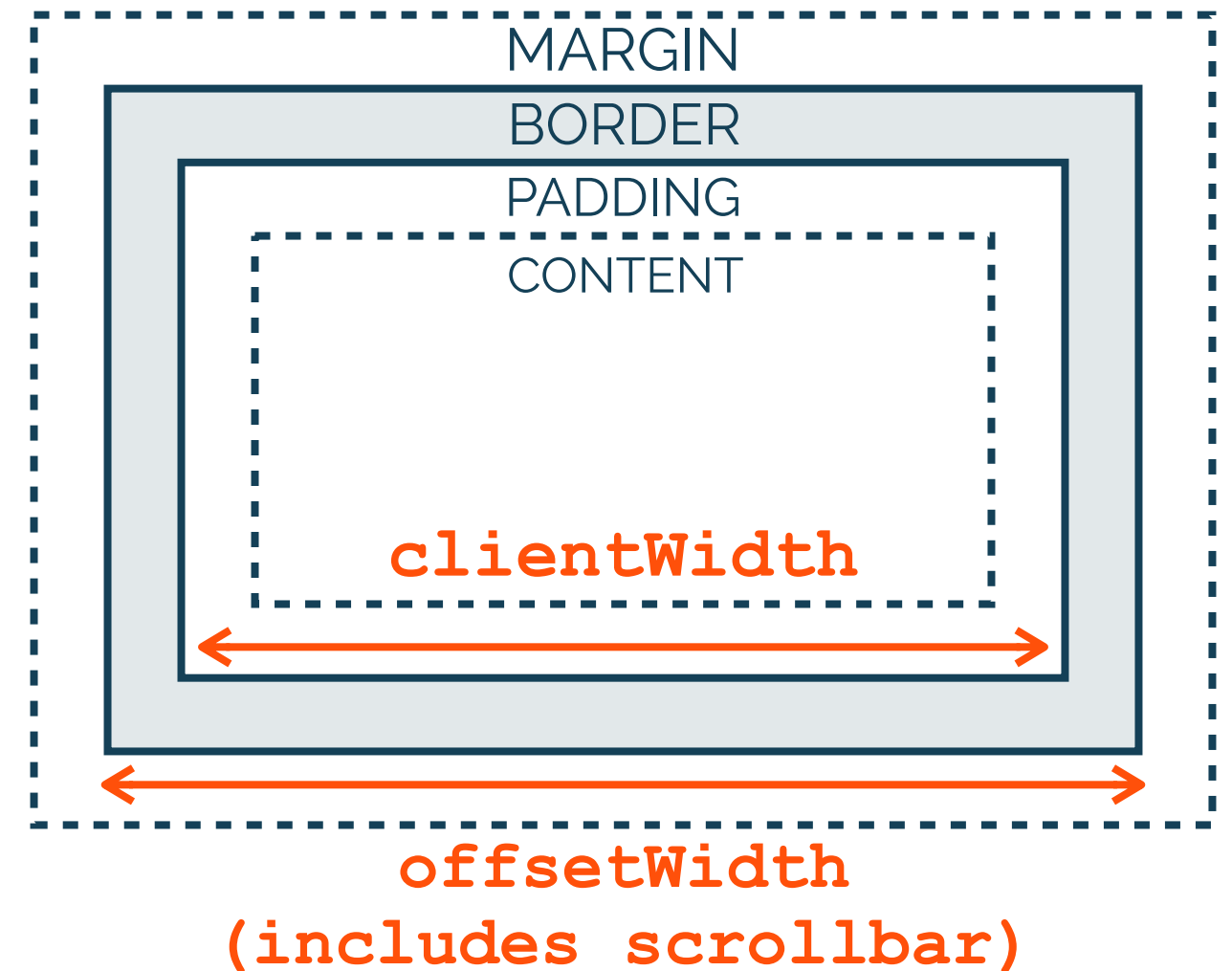
relative to
offsetParent



position: `element.offsetTop`,
`element.scrollTop`, ...

dimensions: `element.clientWidth`,
`element.offsetWidth`, ...

style: `element.style`



DOM MANIPULATION

programmatically change the structure and modify element properties

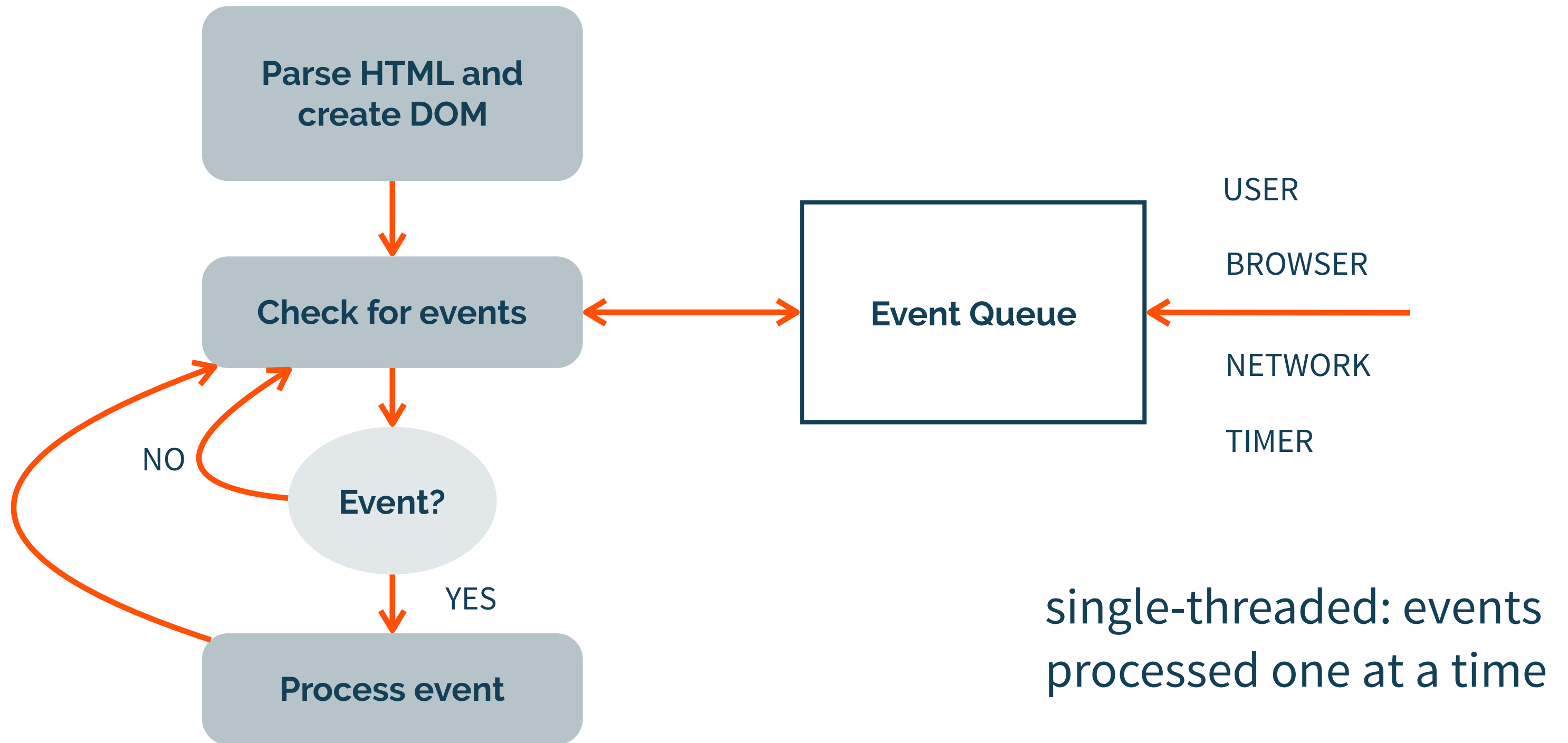
```
element.style.backgroundColor = "red";
```

```
element.innerHTML = "<div><h3>Llama!</h3>...</div>"
```

augment DOM structure:

```
element.appendChild(), element.removeChild(), ...
```

THE BROWSER EVENT LOOP



EVENT PROCESSING

events propagate in two phases

capture phase: root to innermost element

bubble phase: innermost element to root

DOM standard: *capture* then *bubble*

EVENT PROCESSING

```
element.addEventListener(event,  
function, useCapture)
```

 *set capture or bubble phase*

```
event.stopPropagation()
```

CODEPEN

JQUERY

cross-browser

use for all DOM manipulation:

(e.g., positioning relative to document and not offsetParent)

ANGULAR CONCEPTS

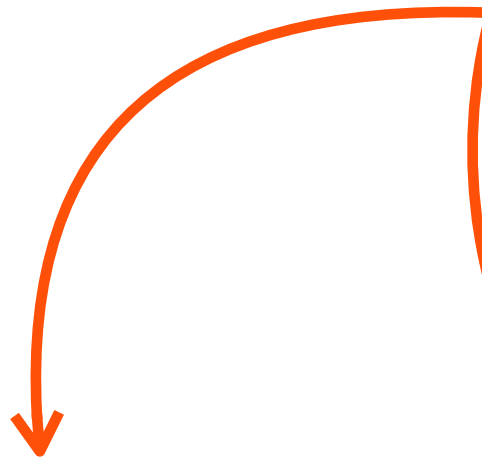
1.3



Controllers

\$scope object

Directives

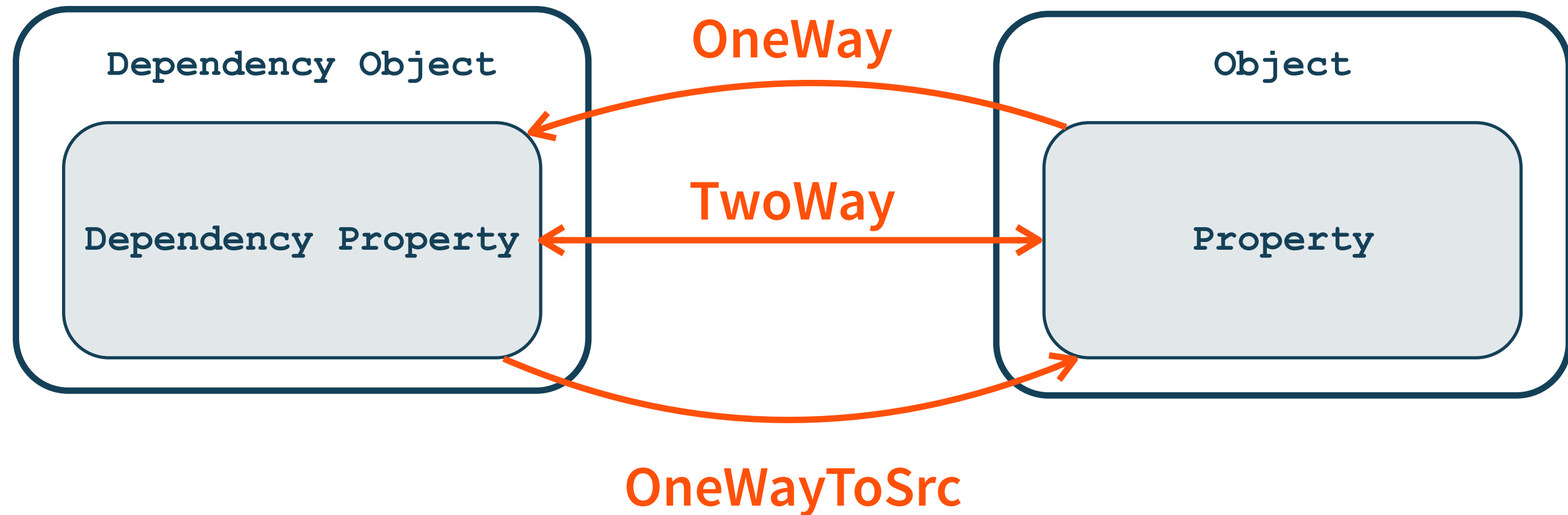


MVC

Client-side Templating

Data Binding

WAYS OF DATA BINDING



MADLIBS TEMPLATE

```
<div ng-app>
  <div ng-controller='MadlibsController'>
    <div>Hola
      <span class="madlib">{{madlibs.animal}}</span>,
    </div>
    <div>Se llama
      <span class="madlib" ng-bind="madlibs.name"></span>!
    </div>
    <form>
      <input ng-model="madlibs.name">
    </form>
  </div>
</div>
```

MONGO SCHEMA DESIGN

For “one-to-few”, you can use an array of embedded documents

For “one-to-many”, or on occasions when the “N” side must stand alone, you should use an array of references. You can also use a “parent-reference” on the “N” side if it optimizes your data access pattern

For “one-to-squillions”, you should use a “parent-reference” in the document storing the “N” side

RESTful API DESIGN

if a relation is usually requested alongside the resource, **embed the relation's representation** within the output representation of the resource

if a relation can exist independently, **include an identifier** for it within the output representation of the resource

GET Get a representation of resource

DELETE Destroy resource

POST Create a new resource based on the given representation

PUT Replace resource state with the one described in the given representation

HEAD Get the headers that would be sent with a representation, but not the representation itself

OPTIONS Discover which HTTP methods this resource responds to

PATCH Modify part of the state of this resource based on the given representation

COLLECTIONS

<VERB> `http://example.com/users`

GET Return all the objects in the collection

POST Create a new entry in the collection;
automatically assign new URI and return it

PUT and **DELETE** not generally used

ELEMENTS

<VERB> `http://example.com/users/12345`

GET Return the specific object in collection

PUT Replace object with another one

DELETE Delete element

POST not generally used

CALLBACK STYLE PROGRAMMING

```
fs.readdir(source, function(err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function(filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function(err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function(width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(destination + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }).bind(this))
        }
      })
    })
  }
})
```

Exam Practice Questions

1) What color will the string "Camelid" be?

```
<ul id="awesome">
  <li class="favorite">
    <span class="highlight">Camelid</span>
  </li>
</ul>
<style>
  #awesome .favorite .highlight {
    color: red;
  }
  #awesome .highlight {
    color: blue;
  }
</style>
```

A: Red
B: Blue

2) What will print?

```
var bar = 20;

var foo = (function() {
  var bar = 10;
  return function(llama) {
    return llama + bar;
  }
})();

var result = bar + foo(5);
console.log(result);
```

A: undefined

B: 25

C: 35

D: 45

3) What will print?

```
function foo() {  
  var result = '';  
  for (var i = 0; i < arguments.length; i++) {  
    result += arguments[i];  
  }  
  return result;  
}
```

```
function bar() {  
  return arguments.join('');  
}
```

```
console.log(foo('a', 'b', 'c'));  
console.log(bar('a', 'b', 'c'));
```

- A: foo and bar both print 'abc'
- B: foo prints 'abc', bar throws a TypeError
- C: foo throws a TypeError, bar prints 'abc'
- D: foo and bar both throw a TypeError

4) What does this evaluate to?

`0.1 + 0.2 == 0.3`

A: true
B: false

5) What will the following HTML code produce?

```
<div class="row">  
  <div class="medium-3 columns">3</div>  
  <div class="medium-3 columns end">3</div>  
  <div class="medium-3 columns end">3 end</div>  
</div>
```

A: 3 divs aligned to the left

B: 3 divs center aligned

C: 2 divs aligned to the left, 1 div aligned to the right

D: 1 div aligned to the left, 2 divs aligned to the right

6) Which of the following is NOT true of MongoDB?

- A: Mongo automatically generates an ID for each document.
- B: Only regular JSON types are allowed.
- C: Mongo does not support table joins.
- D: Documents in Mongo have a maximum size.

7) What will print?

```
var fun = (function outside() {  
  var name = "cat";  
  var cry = "meow";  
  return (function inside() {  
    return function() {  
      console.log(name + " goes " + cry);  
    }  
    var name = "dog";  
  })();  
  var cry = "woof";  
})();  
  
fun();
```

- A: cat goes meow
- B: dog goes meow
- C: undefined goes meow
- D: dog goes woof

Answers

- 1) A: Red
- 2) C: 35
- 3) B: foo prints 'abc', bar throws a TypeError
- 4) B: False
- 5) A: 3 divs aligned to the left
- 6) B: Only regular JSON types are allowed
- 7) C: undefined goes meow