

MIDTERM REVIEW

NEXT WEDNESDAY (10/20):
48-HOUR TAKE-HOME MIDTERM

FORMAT

5 questions with multiple parts

Coding and short answer

THINGS YOU SHOULD KNOW ABOUT THE MIDTERM

Anything from lecture and MPs is fair game

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

Will test your ability to apply what you've learned in new
situations

Open-book but you must cite your sources

You must work on it independently

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>
```

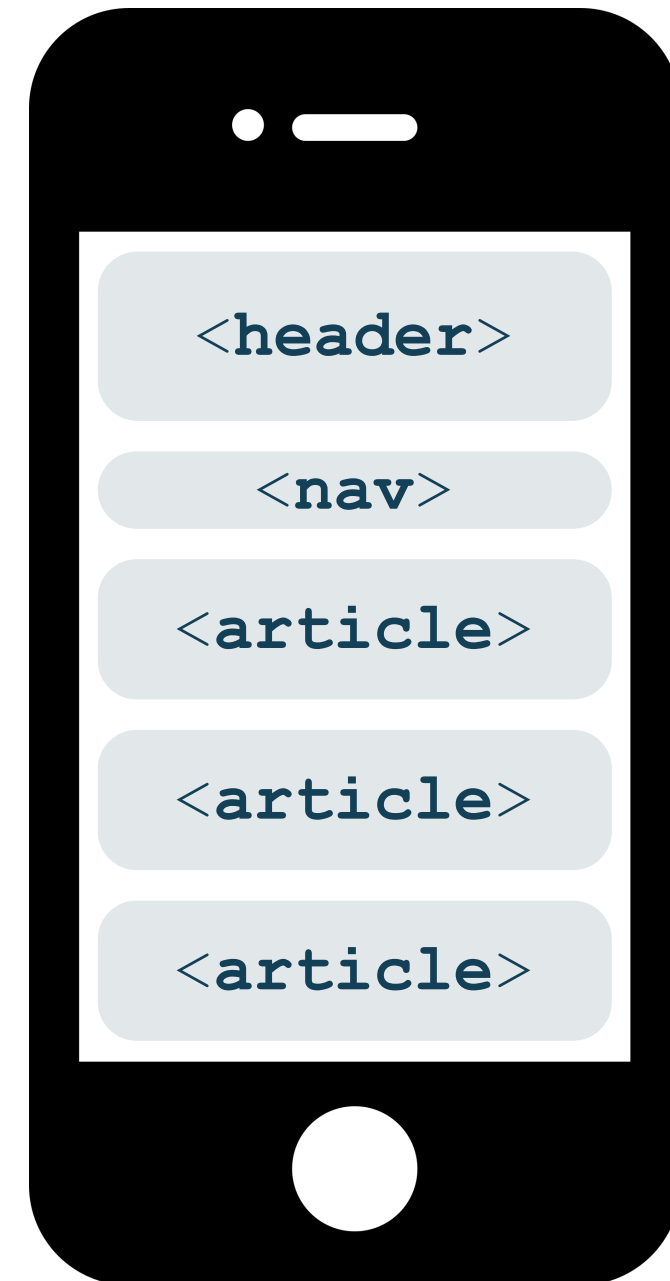
STRUCTURAL SEMANTIC APPLICATIONS?

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"/>`

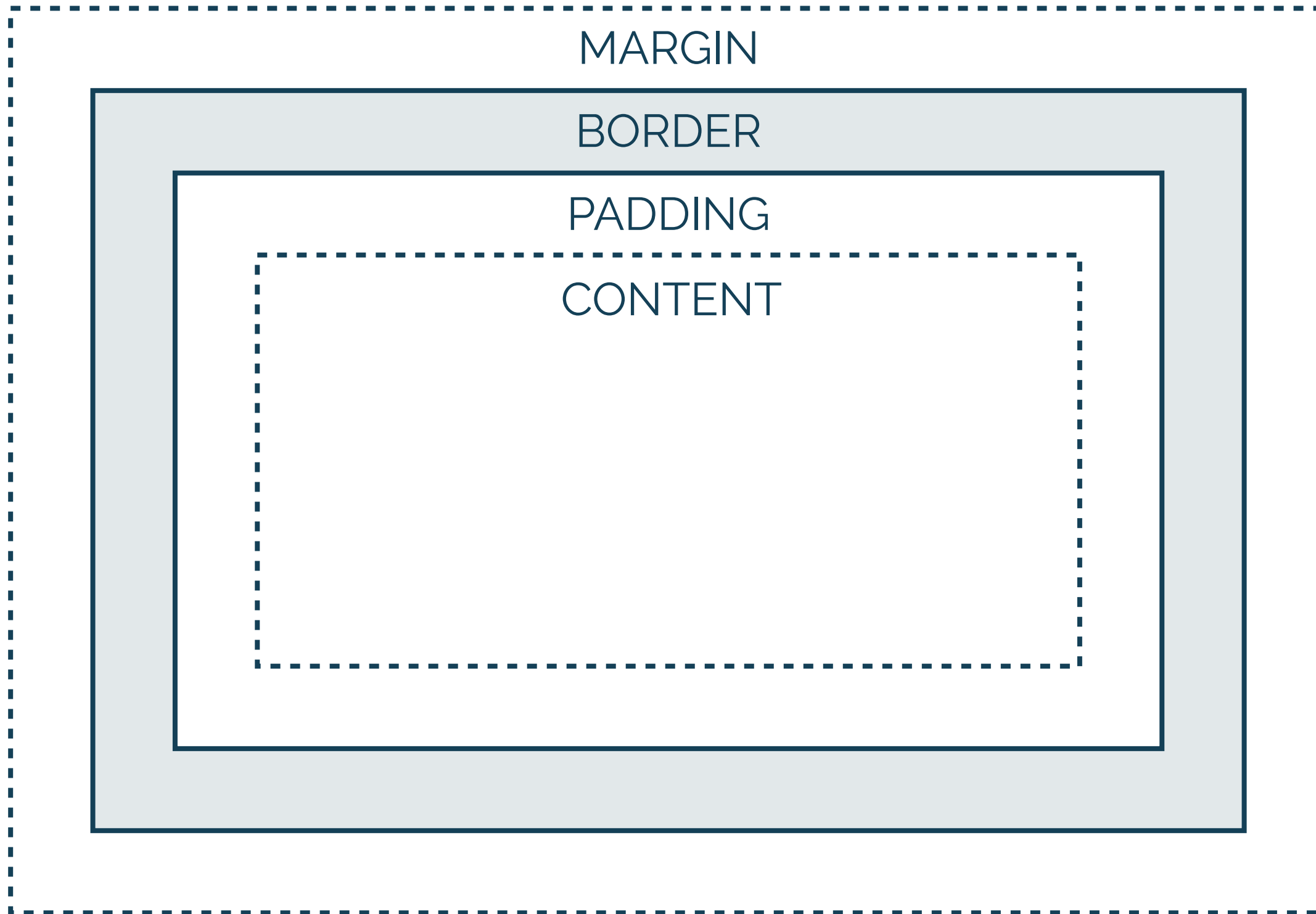
(2)

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

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

higher priority

Box Model

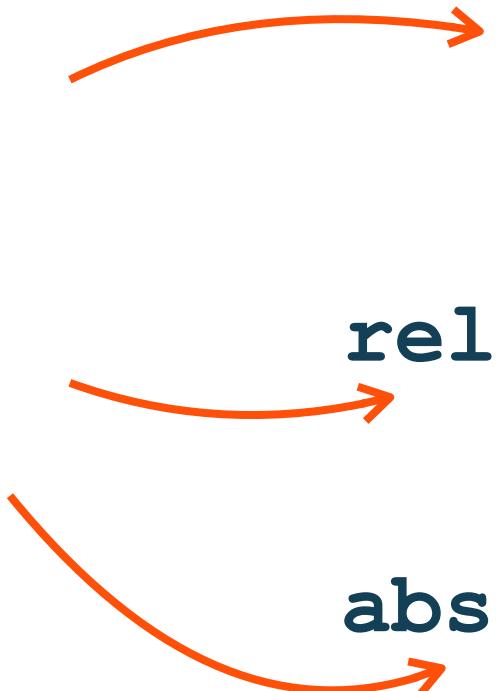


control over
white space

position

VALUE	DESCRIPTION
static	default. positioned by the flow model; unaffected by top, bottom, left, right
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

use with
top
bottom
left
right



```
graph LR; A[use with<br/>top<br/>bottom<br/>left<br/>right] --> B[fixed]; A --> C[relative]; A --> D[absolute];
```

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

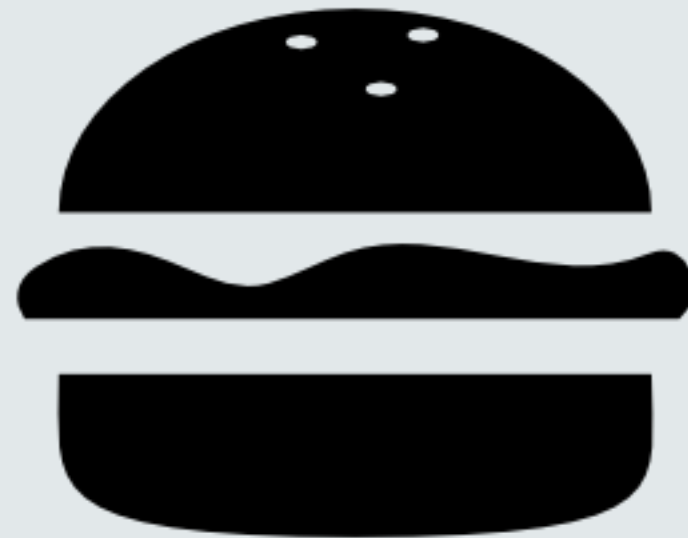
shrinks developer's codebase and compiles into CSS

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, ...)`

```
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

constructors are given the class name

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

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

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

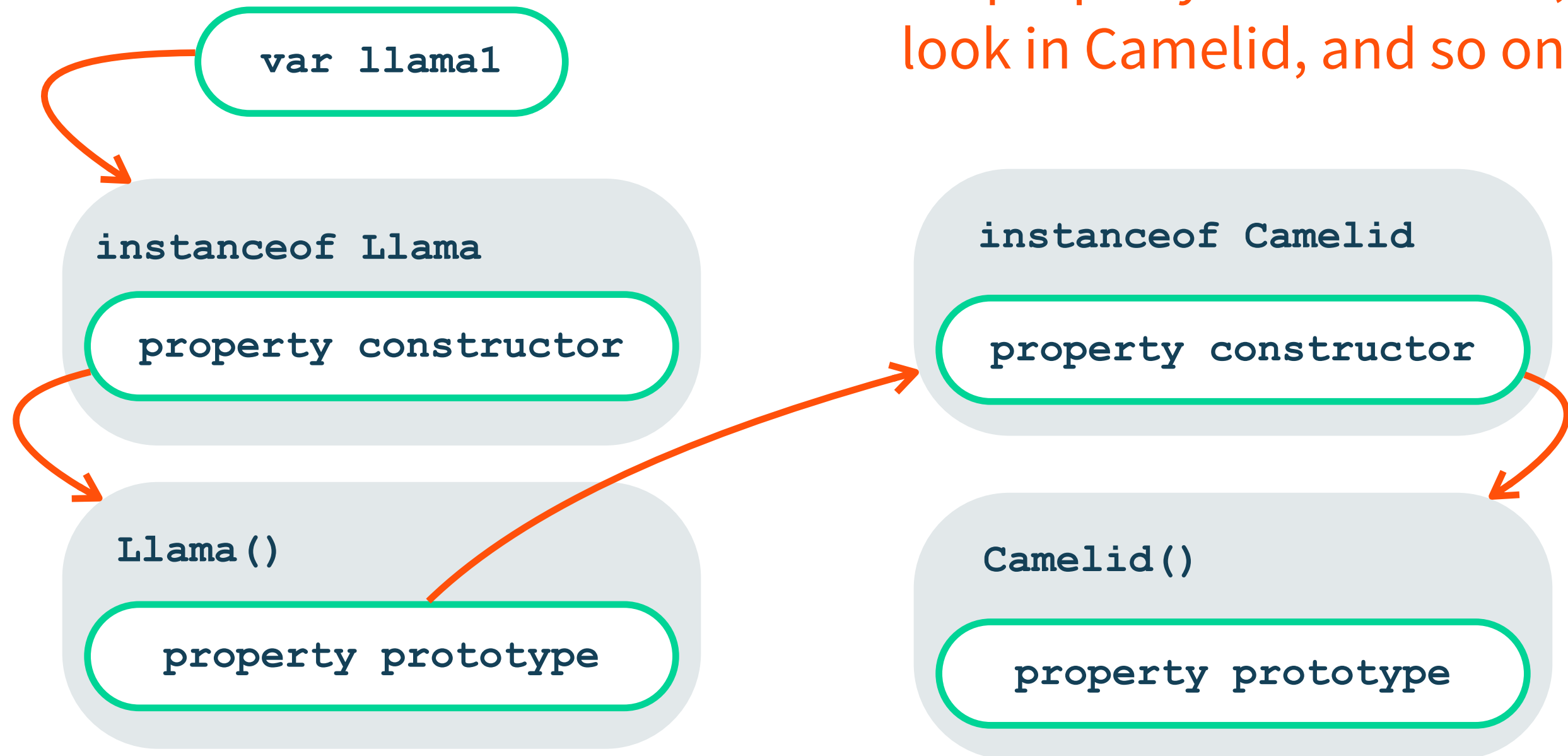
prototype

prototype is a property of the constructor
another way to add methods to objects

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

PROTOTYPE CHAINING

if a property isn't in Llama,
look in Camelid, and so on



scopes are declared through
functions and not blocks { }

HOISTING

Variables and functions are in scope within the entire function they are declared in

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; }
```

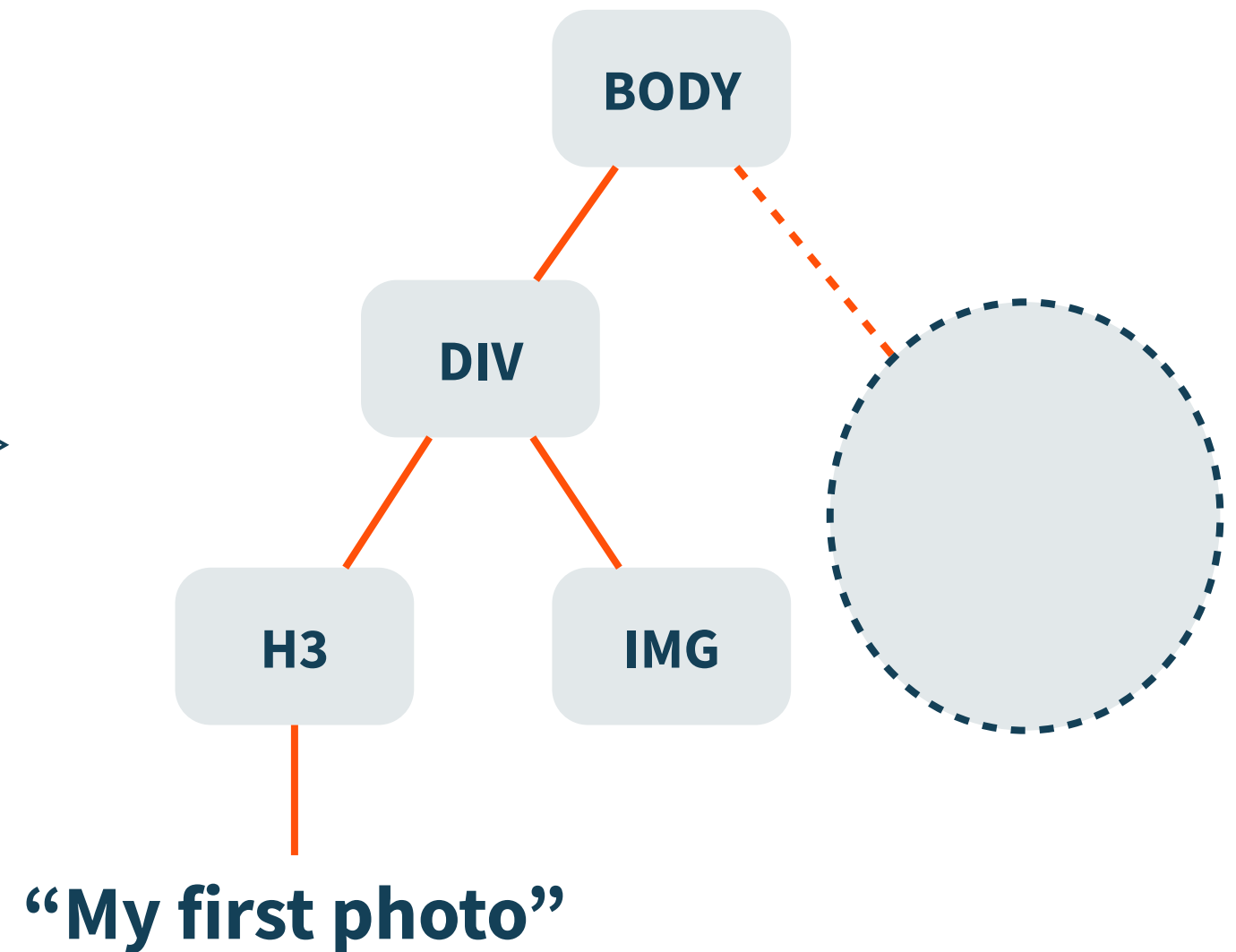
```
}
```

private data member now!

DOCUMENT OBJECT MODEL

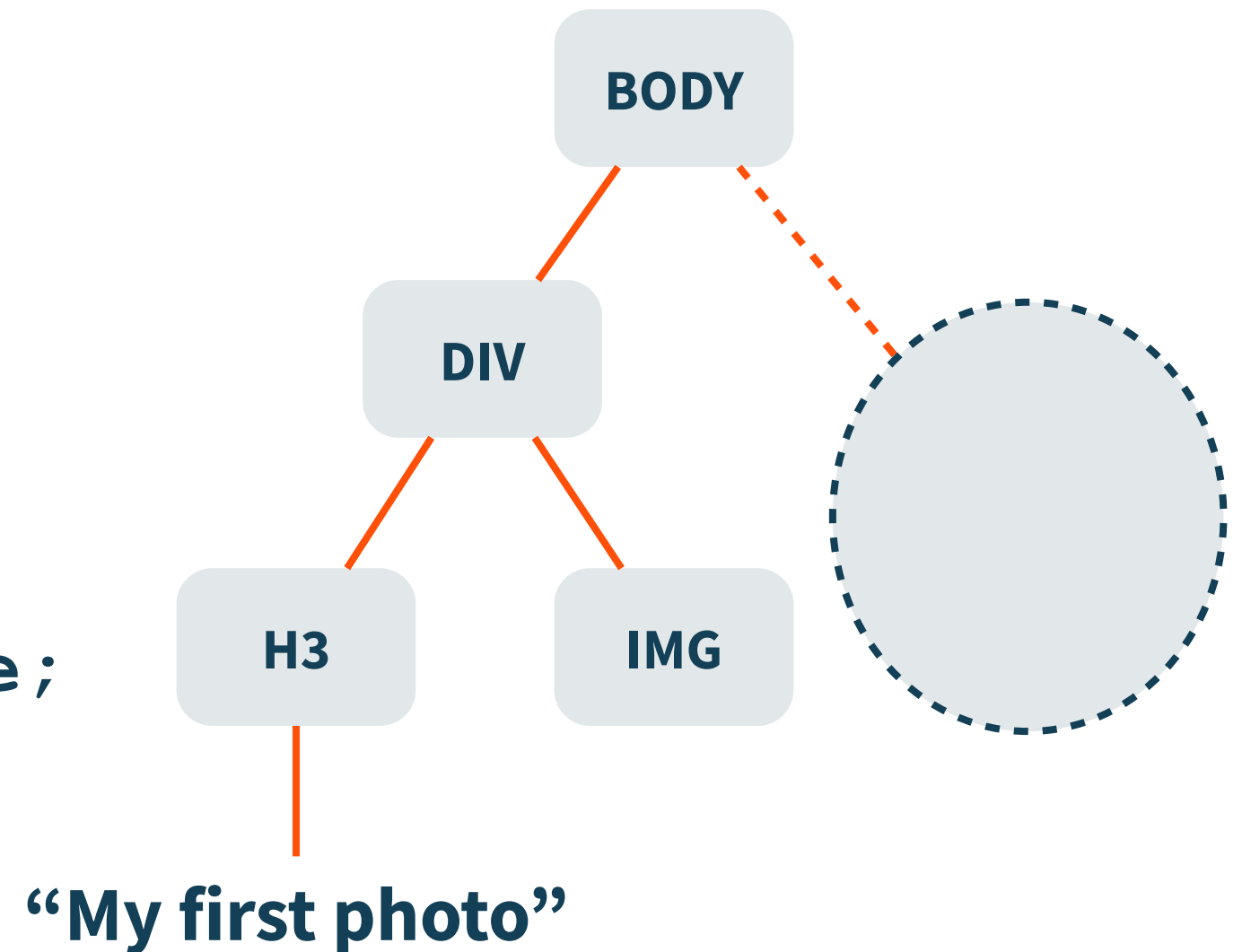
one-to-one correspondence between HTML elements and DOM nodes

```
<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

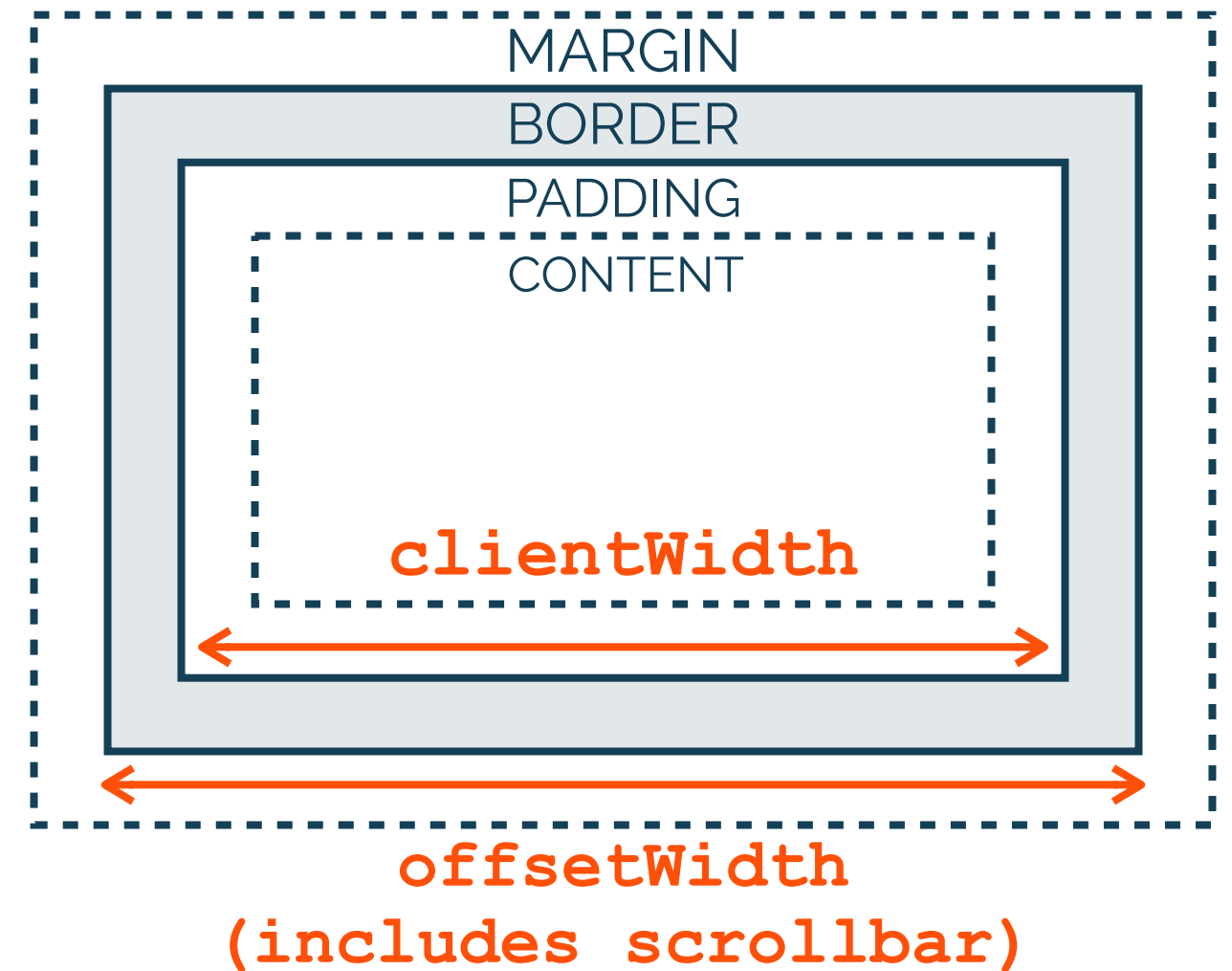
relative to
offsetParent



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

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

style: `element.style`



DOM MANIPULATION

programmatically change the structure and modify element properties

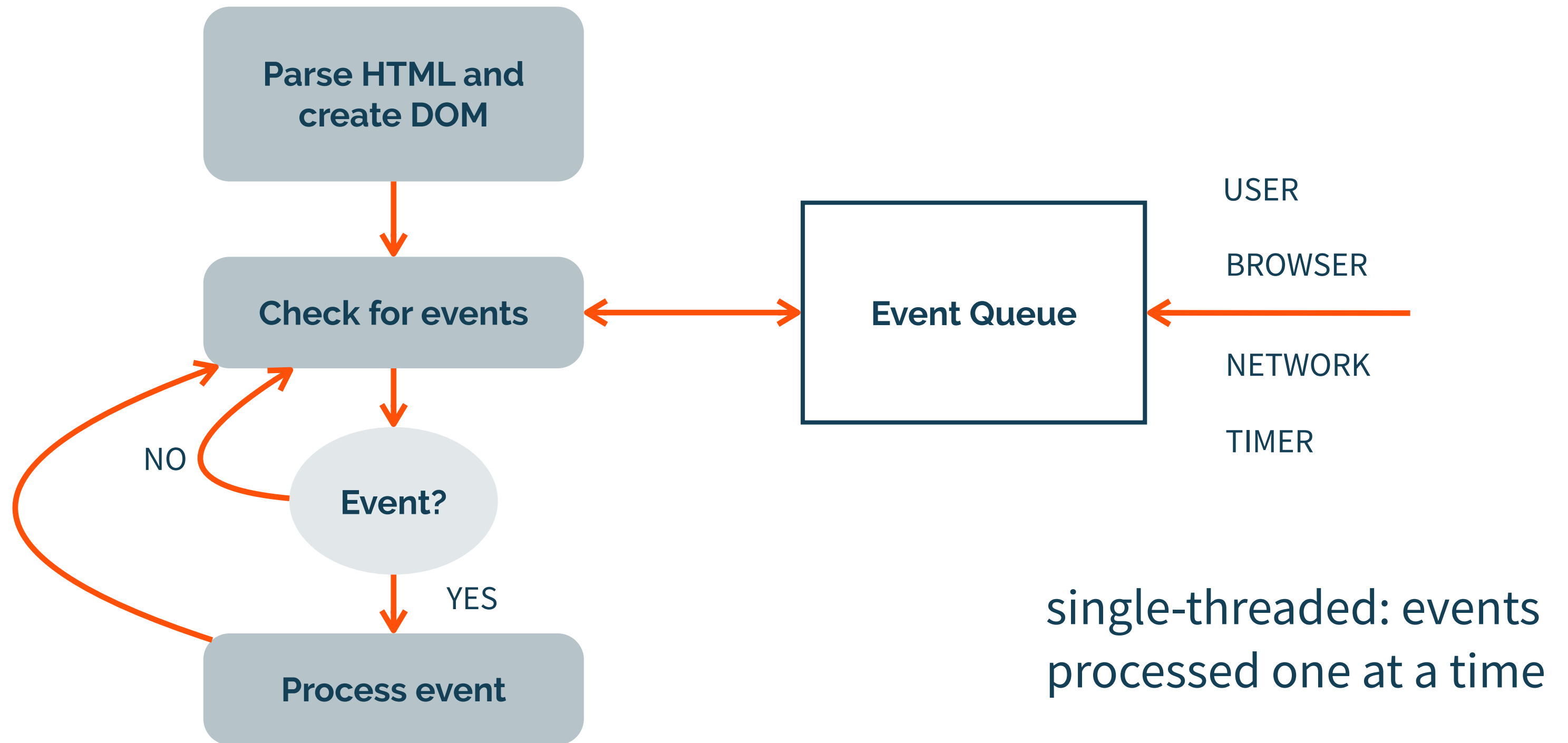
```
element.style.backgroundColor = "red";
```

```
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

REACT COMPONENTS

each module manages its
own data and views

how to write components
and compose them

REACT CONCEPTS

Unidirectional data flow

DOM Painting

Virtual DOM

How JSX works

REACT CONCEPTS

Props

State

Lifecycle Methods

render & setState

React Router

ES6

Fat arrow functions

let

const

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