In this lecture, we will discuss…

Development Environment Setup

Tools We Will Need

- Sign up for an account on GitHub.com
- Google Chrome Browser
  - Already comes with CDT (Chrome Developer Tools)
- Sublime Text 3 (or any code editor)
- Git
- Browser Sync
  - Will need to install NodeJs for it to work

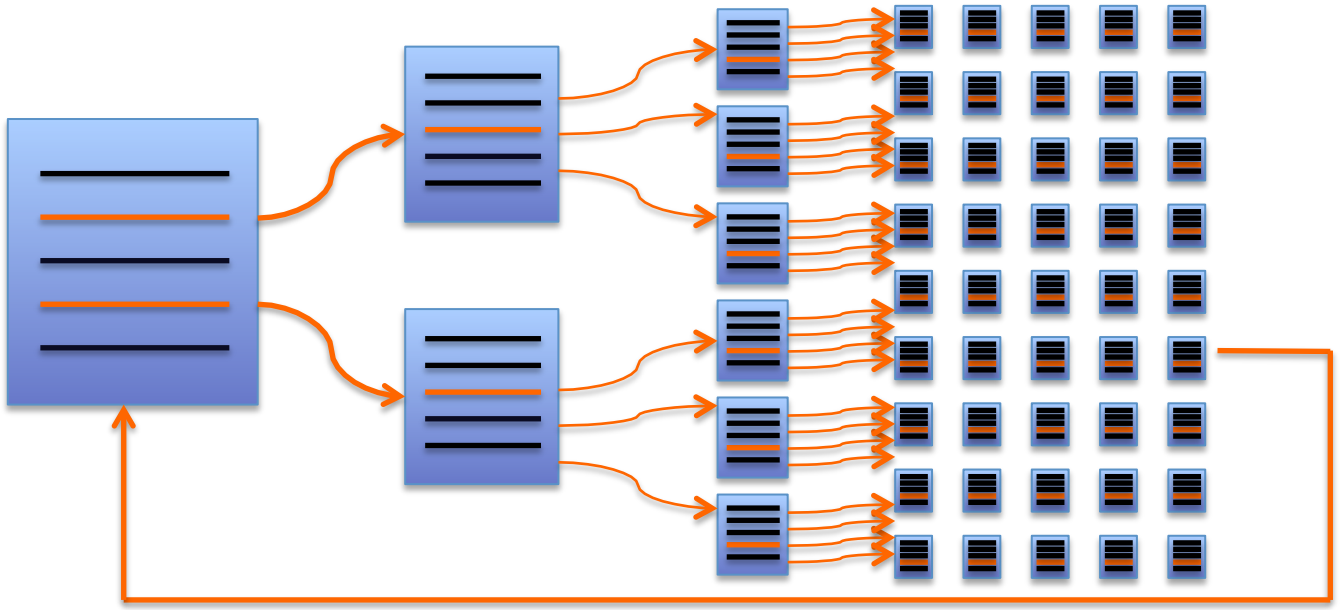
In this lecture, we will discuss…

What is HTML?

What Does HTML Stand For?

What Is HTML?

Hypertext Markup Language



What Is HTML?

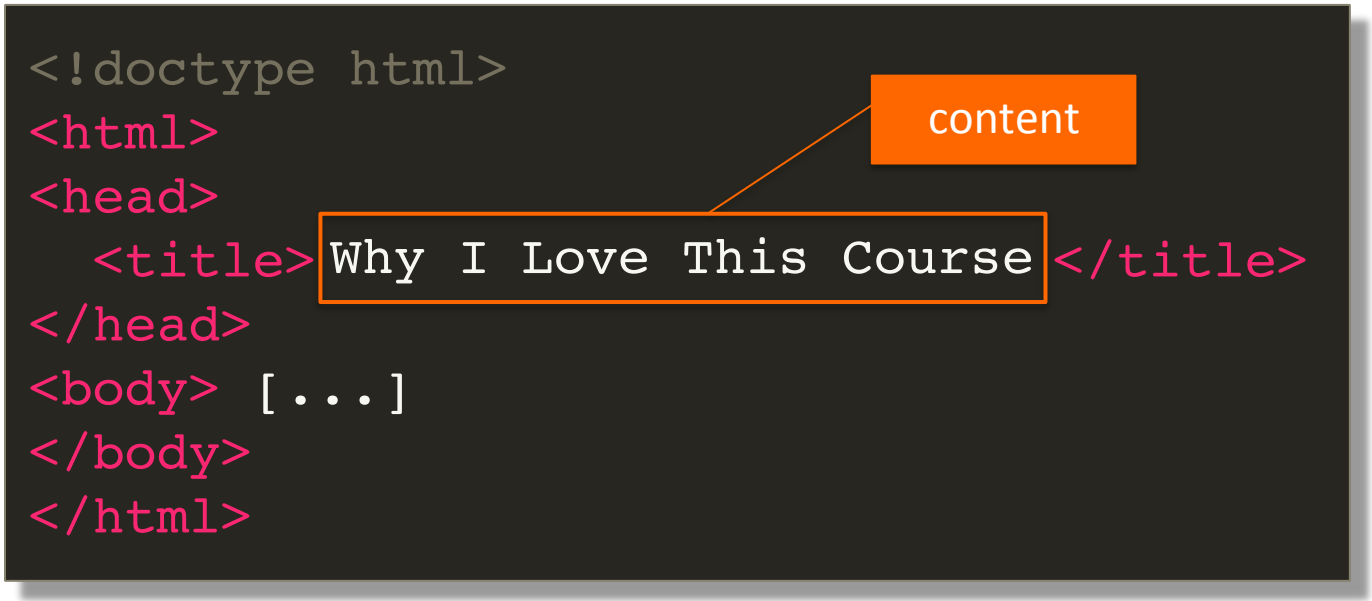
Hypertext Markup Language

Hypermedia

Hypertext

What Is HTML?

Hypertext **Markup** Language



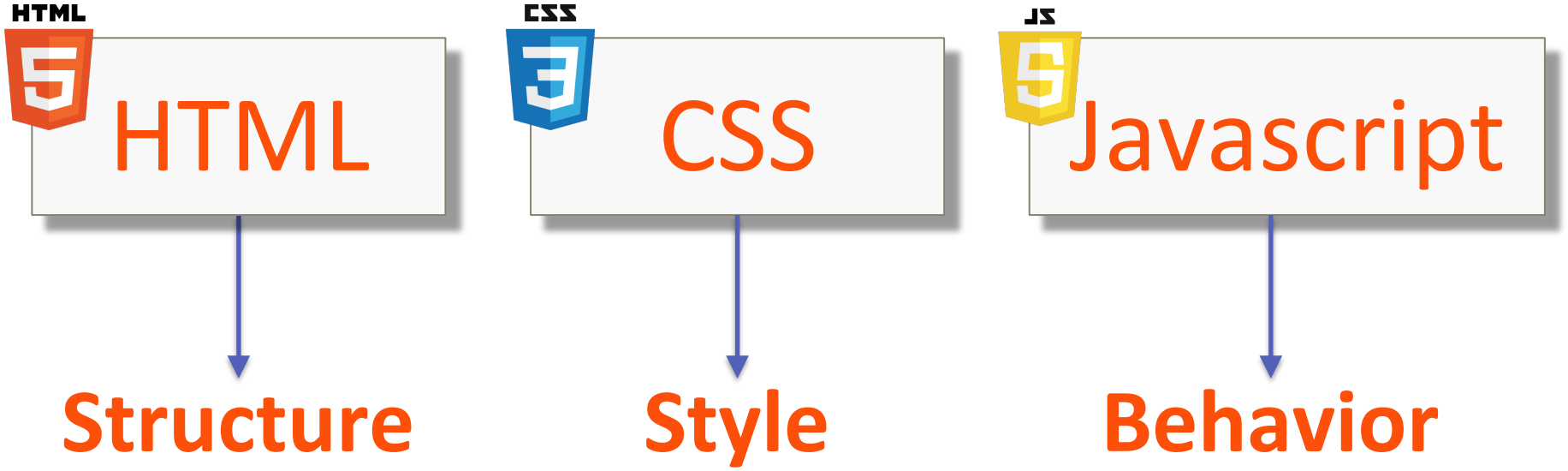
What Is HTML?

Hypertext Markup **Language**

```
<h1>
  <div>Hello World!</h1>
</div>
```

```
<h1>
  <div>Hello World!</div>
</h1>
```

Technologies That Drive The Web



Summary

- HTML
  - Annotates content
  - Defines document structure
- Right and wrong syntax
- 3 Core Web Technologies
  - HTML, CSS, Javascript

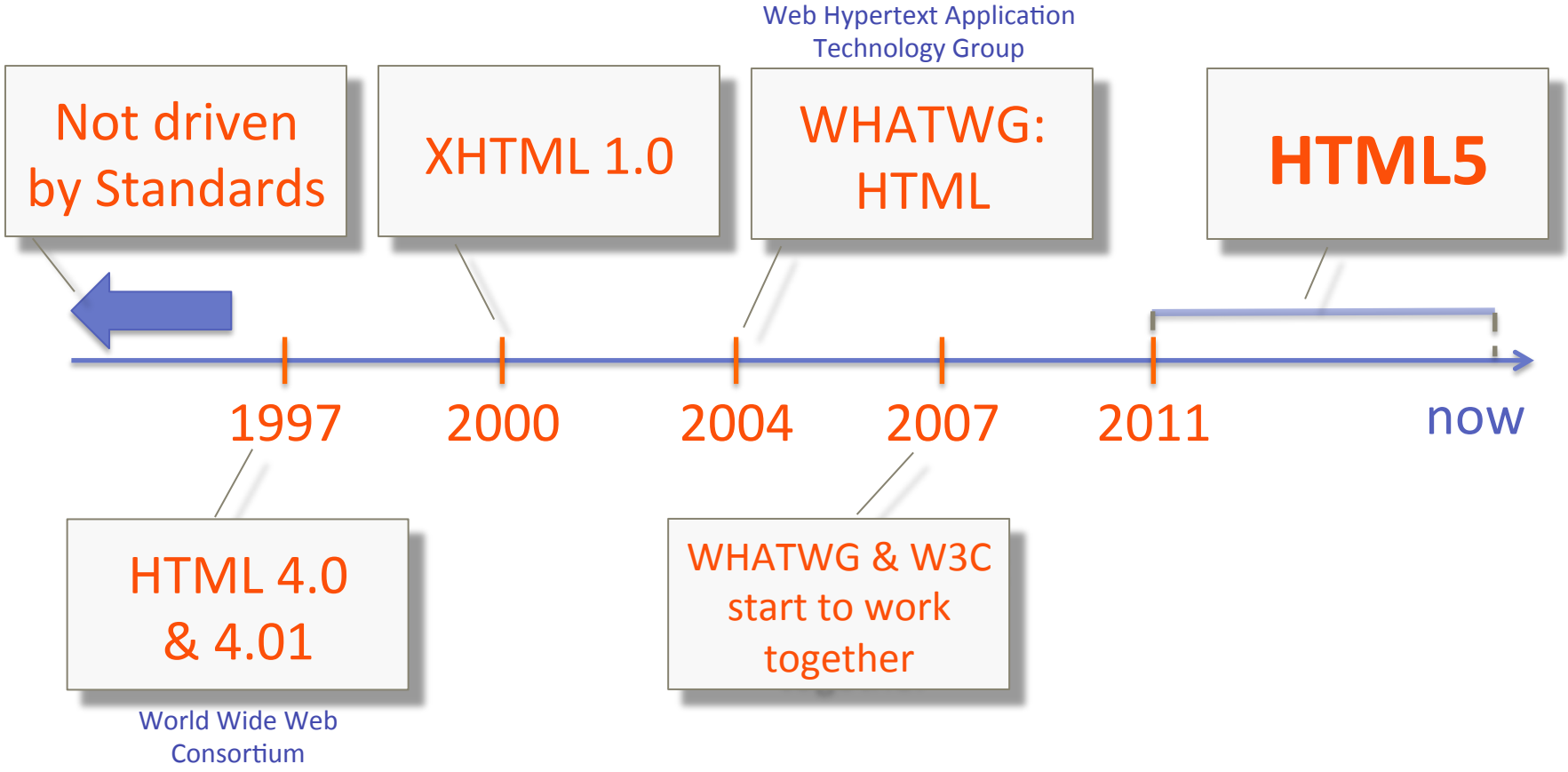
NEXT:

A Bit Of Relevant History Of The HTML Standard

In this lecture, we will discuss…

History of HTML

How We Got to HTML5



What Matters to You

HTML5

(standard)

HTML

(evolving)

Summary

- History of HTML and it affects us as developers
- Resources for validation
- Resources for feature investigation
- Browser stats resource

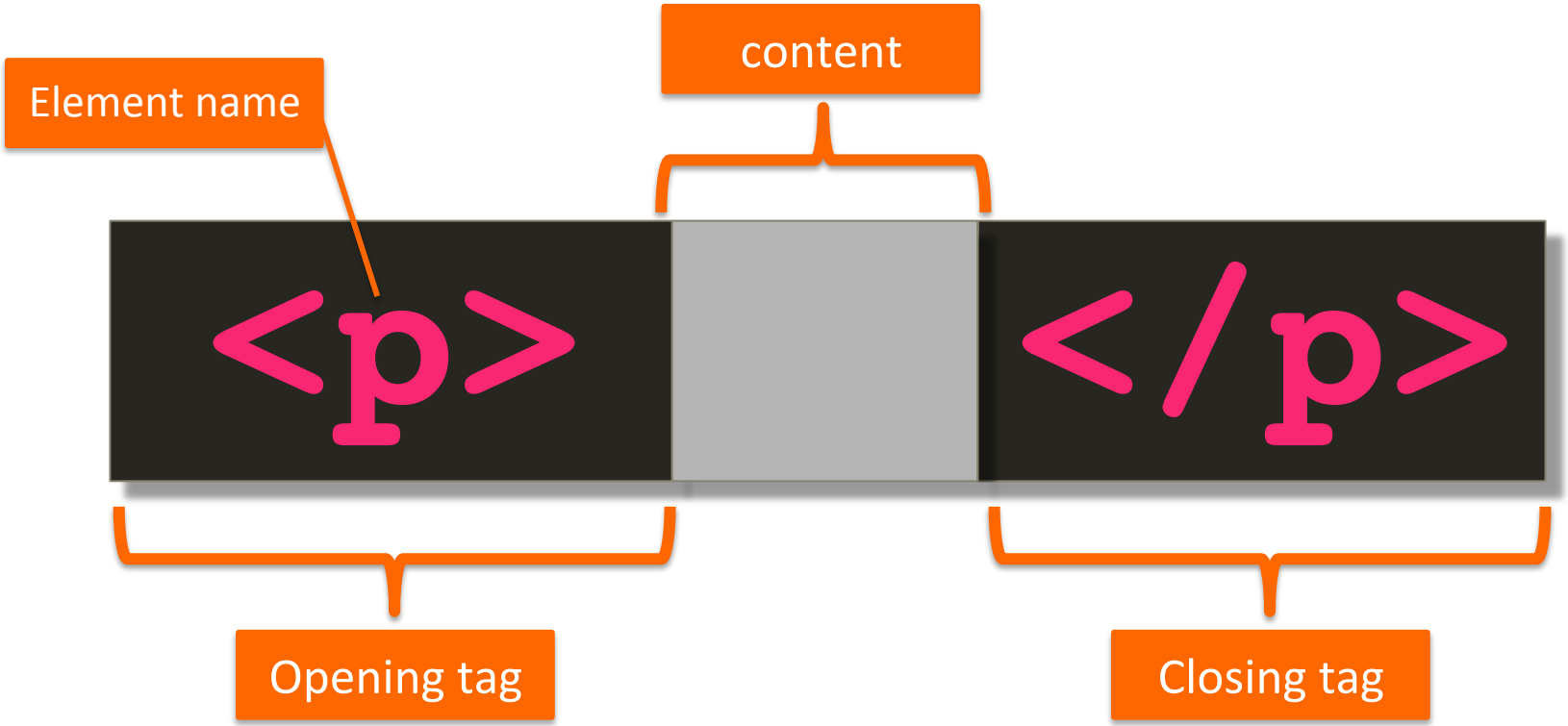
NEXT:

Anatomy of an HTML tag

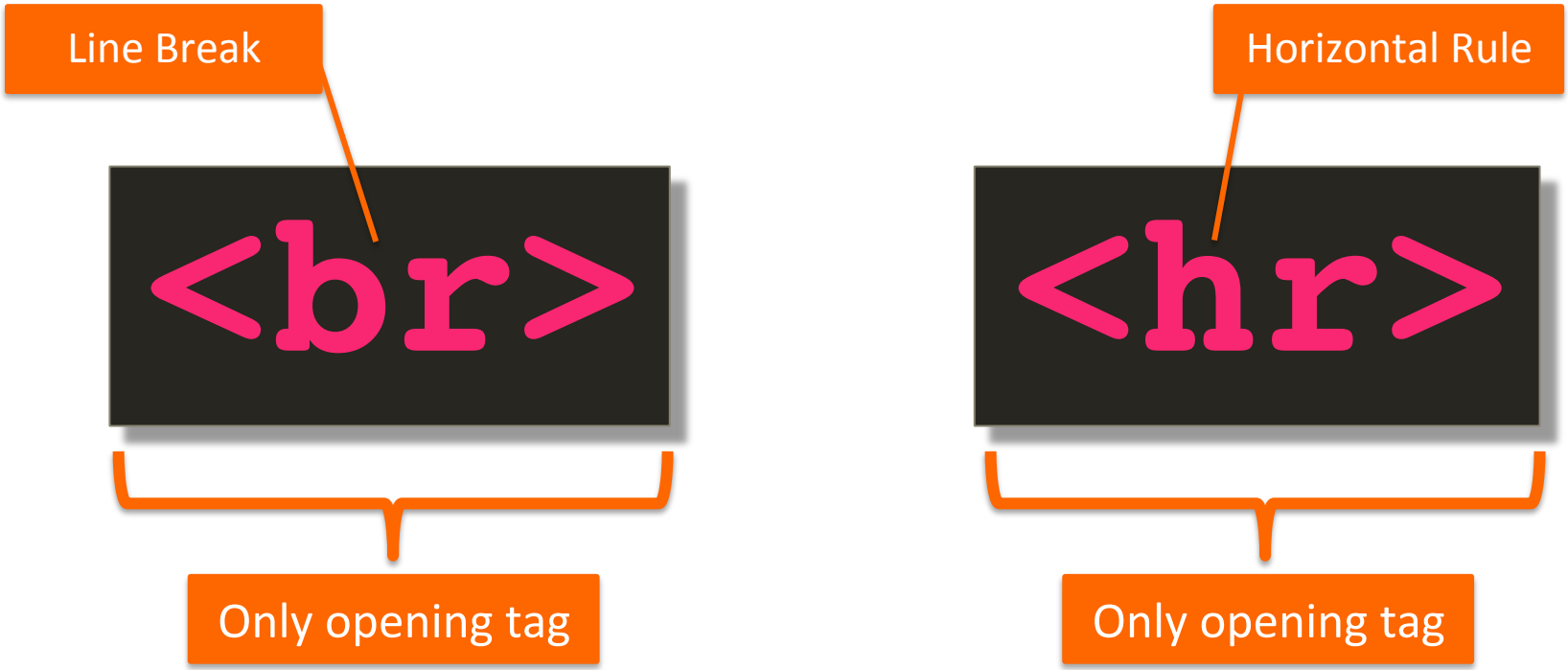
In this lecture, we will discuss…

Anatomy of an HTML Tag

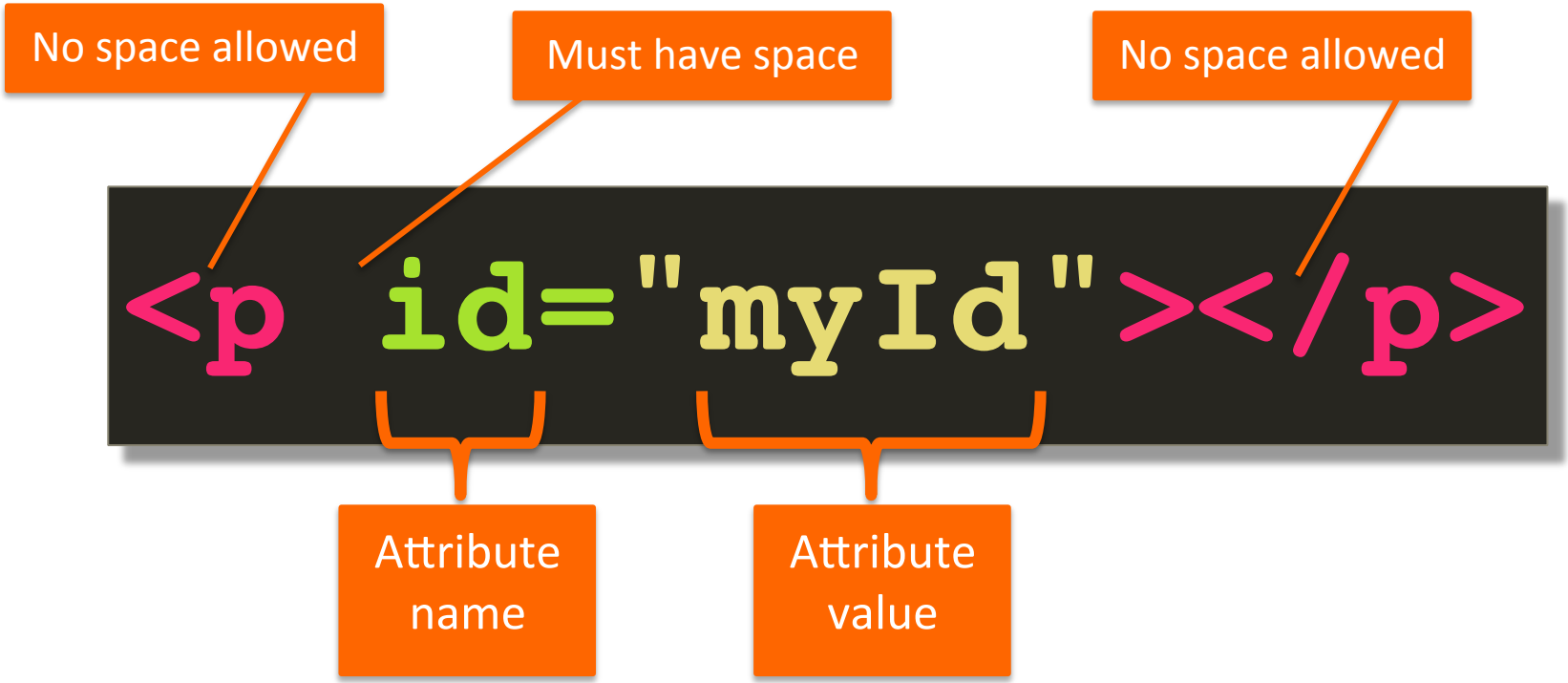
Anatomy of an HTML tag



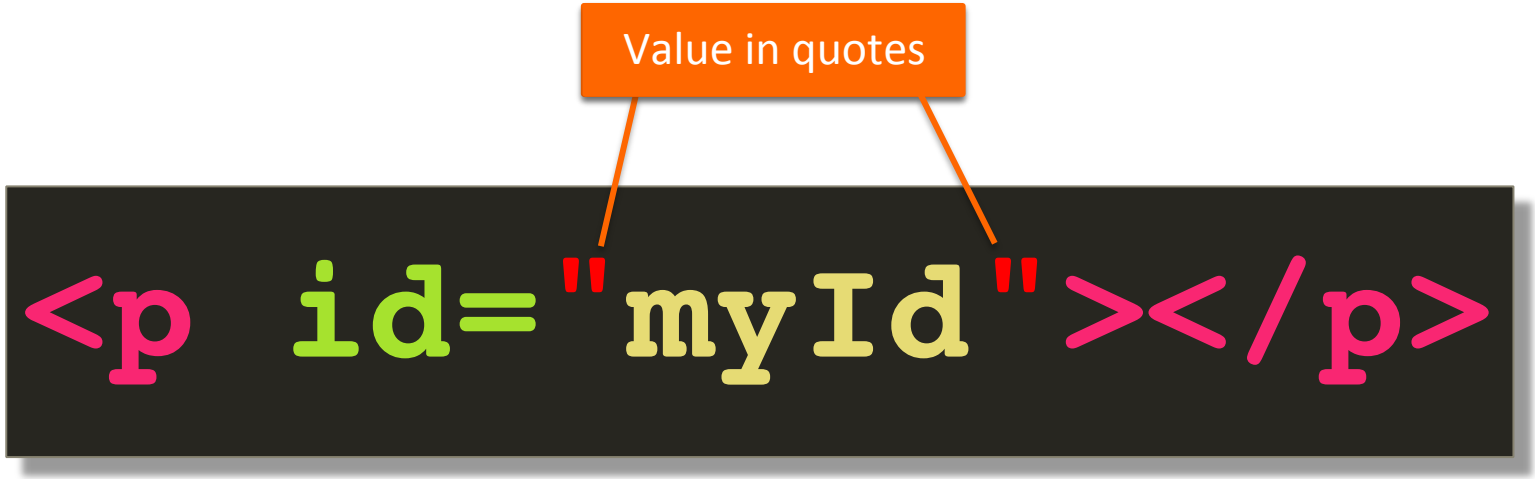
Anatomy of an HTML tag



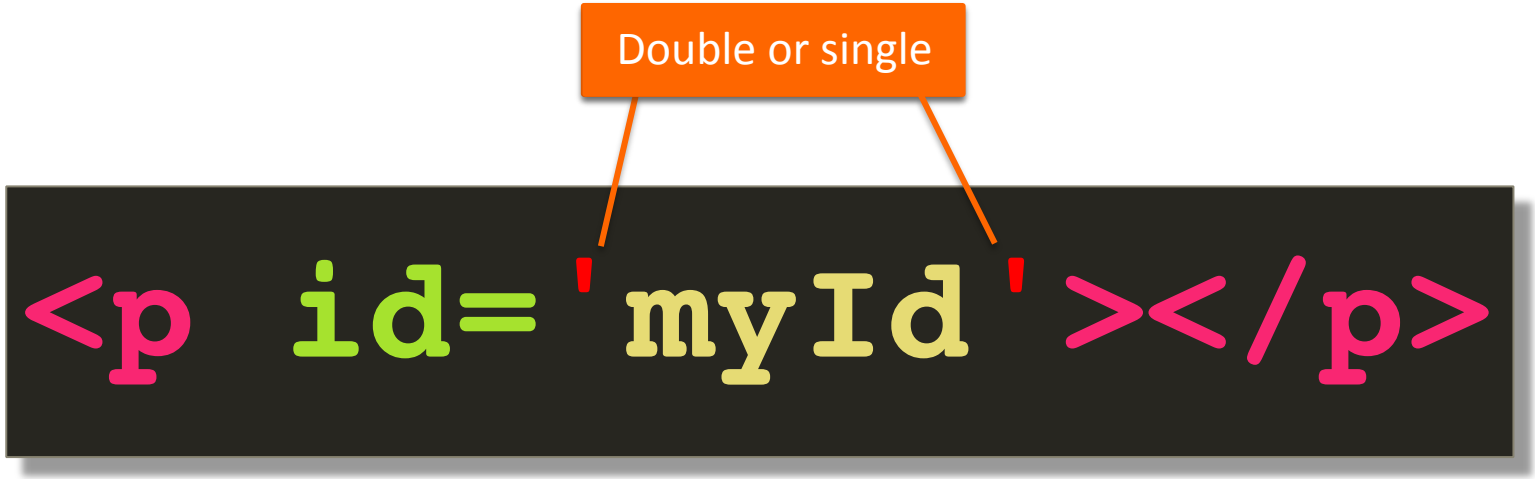
Anatomy of an HTML tag



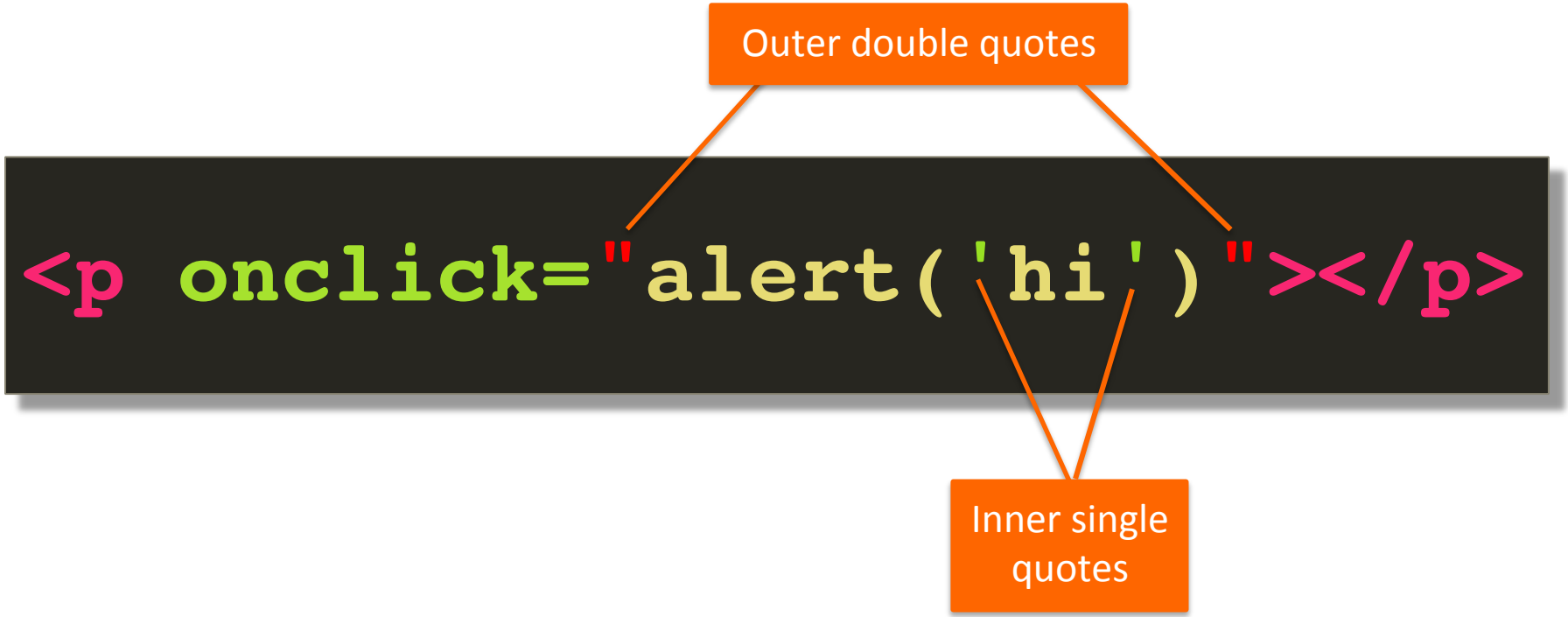
Anatomy of an HTML tag



Anatomy of an HTML tag



Anatomy of an HTML tag



Anatomy of an HTML tag

```
<p/>
```

✗

```
<p></p>
```

✓

Summary

- Anatomy of an HTML tag
  - Opening and closing tag
  - Attributes
  - Using double and single quotes
  - How to specify tag without any content

NEXT:

Basic HTML Document Structure

In this lecture, we will discuss…

Basic HTML Document Structure

Summary

- The bare minimum HTML document
- HTML version declaration
- Our first HTML tags!
  - `<html>` – `<meta>` – `<head>` – `<title>` – `<body>`
- Sequential (top to bottom) rendering

NEXT:

HTML Content Models

In this lecture, we will discuss…

HTML
Content Models

Block-Level Elements

- Render to begin on a new line (by default)
- May contain inline or other block-level elements
- Roughly Flow Content (HTML5 category)

Inline Elements

- Render on the same line (by default)
- May only contain other inline elements
- Roughly Phrasing Content (HTML5 category)

- **HTML5 replaces these definitions with more complex set of content categories.**
- **However, this distinction remains practical because it aligns well with existing CSS rules.**

Summary

- Compared block-level & inline content types
- Officially not part of HTML5, but still used
- Roughly equivalent to flow content & phrasing content

NEXT:

Headings & Some New Semantic Elements

In this lecture, we will discuss…

Heading Elements (and some new HTML5 semantic elements)

se·man·tic

Relating to meaning in language or logic

semantic html element

Element that implies some meaning to the content

- Human and/or machine can understand meaning of content surrounded by a semantic element better
- *May* help search engine ranking, i.e., SEO

Summary

- Well chosen content of H1 element is crucial to SEO
- Semantic elements allow for a more meaningful expression of the structure of our HTML page

NEXT:

Structuring Content with Lists

In this lecture, we will discuss…

Lists

Summary

- Lists provide a natural and commonly used grouping of content
- Very often, lists are used for structuring navigation portions of the web page

NEXT:

HTML Character Entity References

In this lecture, we will discuss…

HTML Character Entity References

3 Characters You Should Always Escape

Instead of: `<`

USE: `<`

Instead of: `>`

USE: `>`

Instead of: `&`

USE: `&`

Summary

- HTML Entities
  - Help avoid rendering issues
  - Safeguard against more limited character encoding
  - Provide characters not available on a keyboard

NEXT:

Making text hyper with Linking

In this lecture, we will discuss…

Creating Links

Summary

- Internal linking to other pages in the site
- External linking to other web sites
- Linking to sections of a document

NEXT:

Displaying Images

In this lecture, we will discuss…

Displaying Images

Summary

- Images can enhance your site
- Remember to specify width and height attributes whenever possible

Summary

- CSS is an incredibly powerful technology
- User experience of content matters

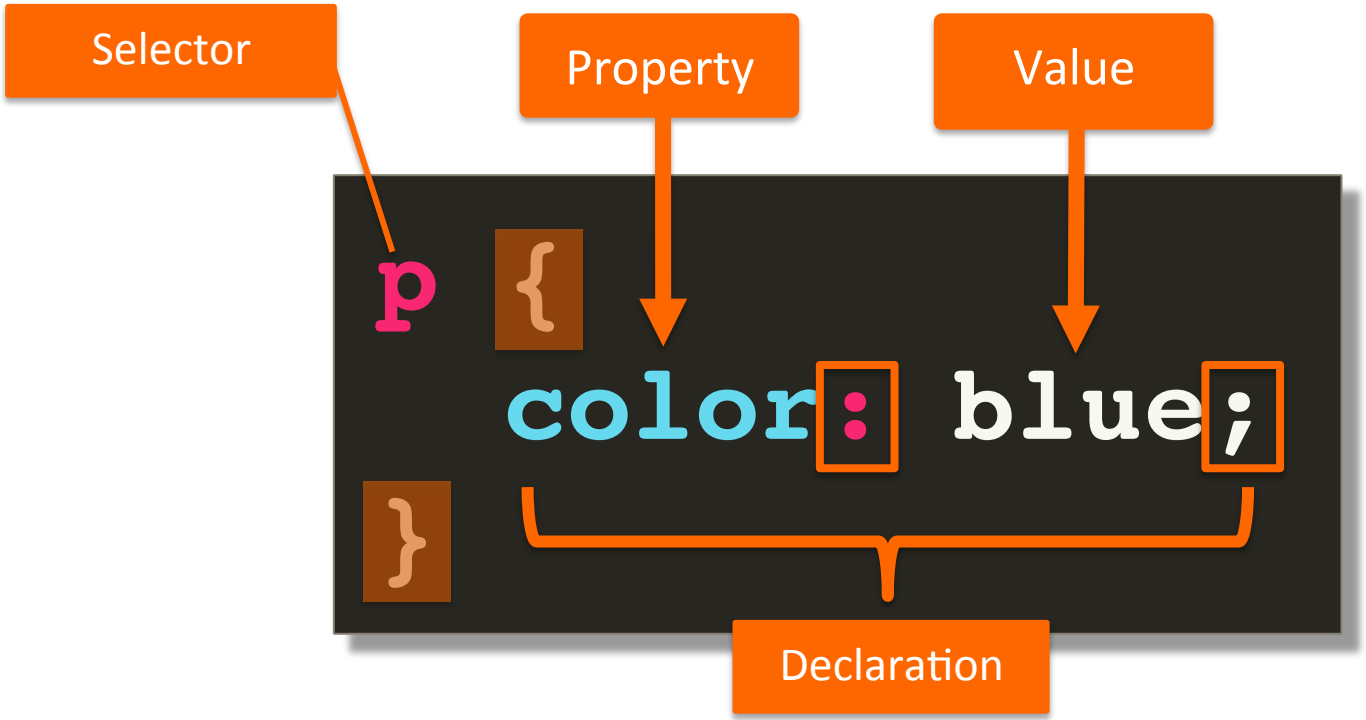
NEXT:

Anatomy of a CSS Rule

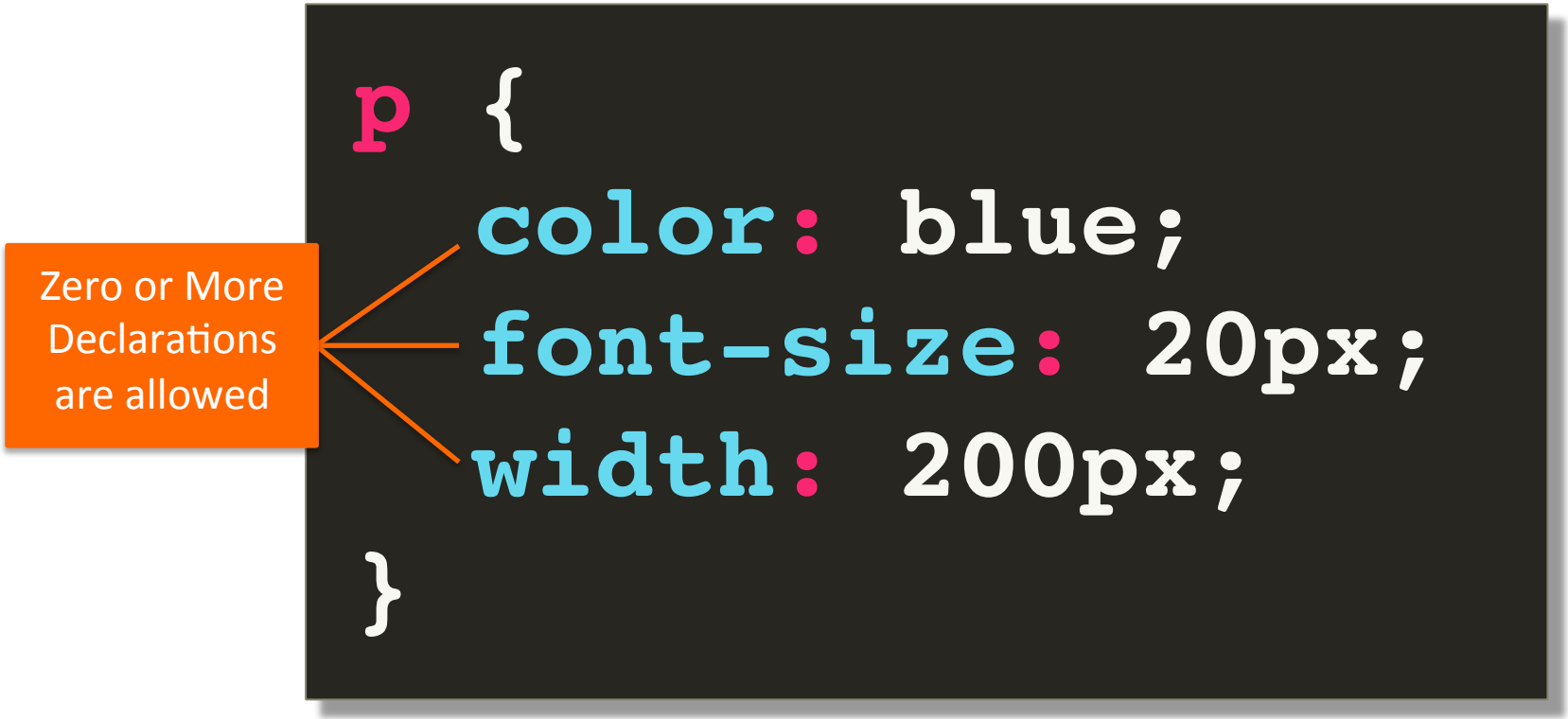
In this lecture, we will discuss…

Anatomy of a CSS Rule

Anatomy of a CSS Rule



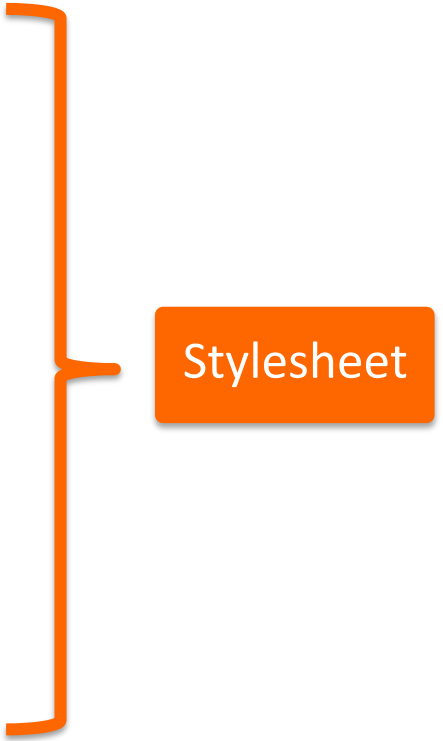
Anatomy of a CSS Rule



Anatomy of a CSS Rule

```
p {
  color: blue;
  font-size: 20px;
  width: 200px;
}

h1 {
  color: green;
  font-size: 36px;
  text-align: center;
}
...
```



Summary

- ✧ Syntax of a CSS rule
  - Selector
  - Declaration
  - Property/value pair
- ✧ Style sheet

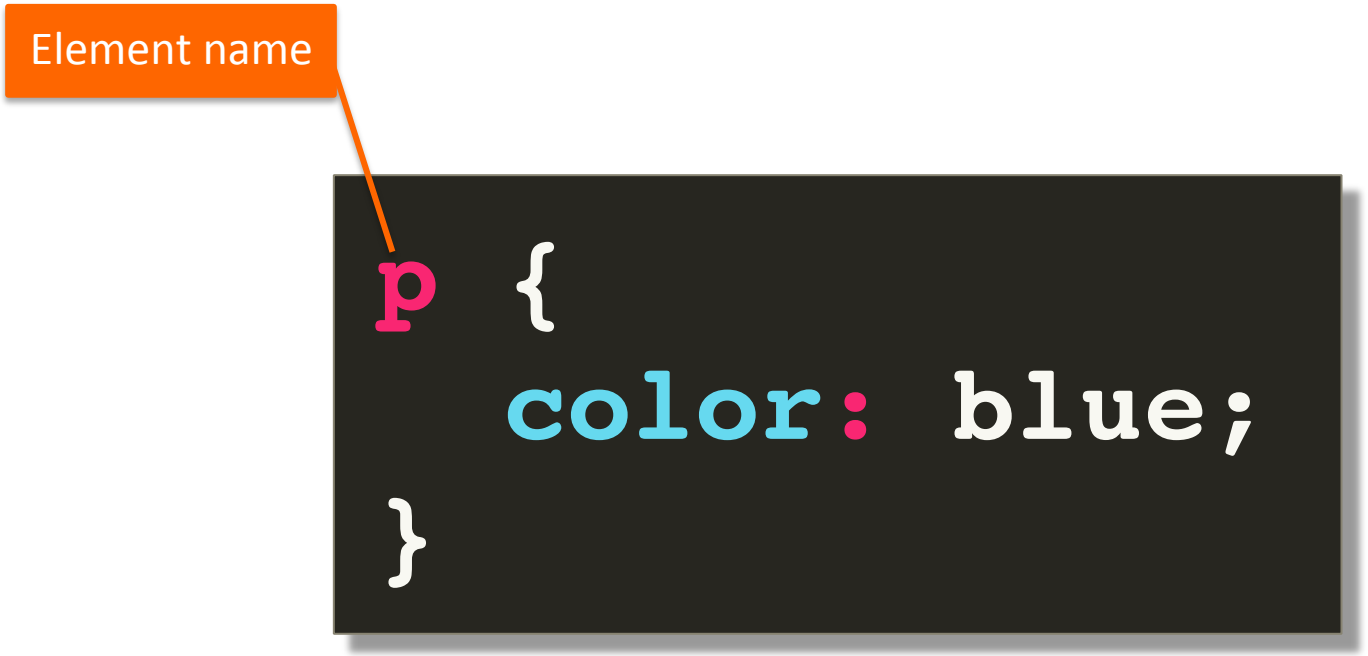
NEXT:

Element, `class`, & `id` Selectors

In this lecture, we will discuss…

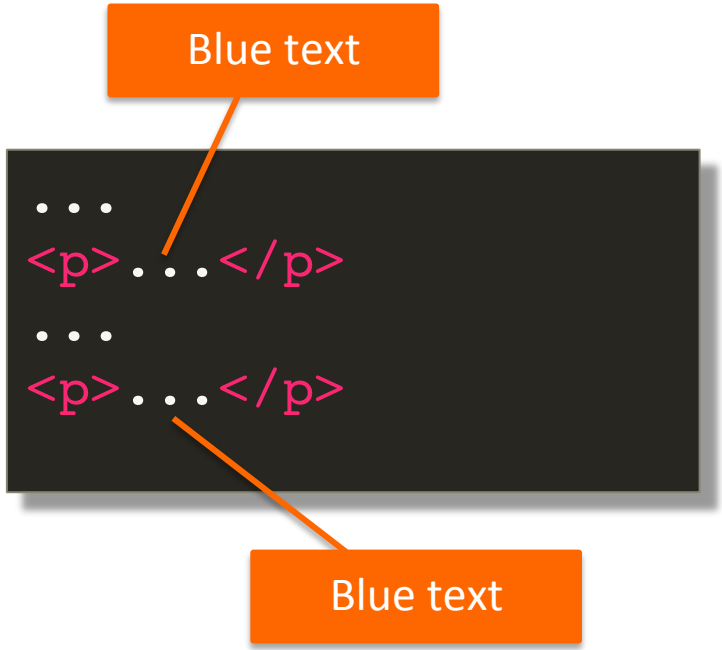
Element, `class`, & `id` Selectors

Element Selector

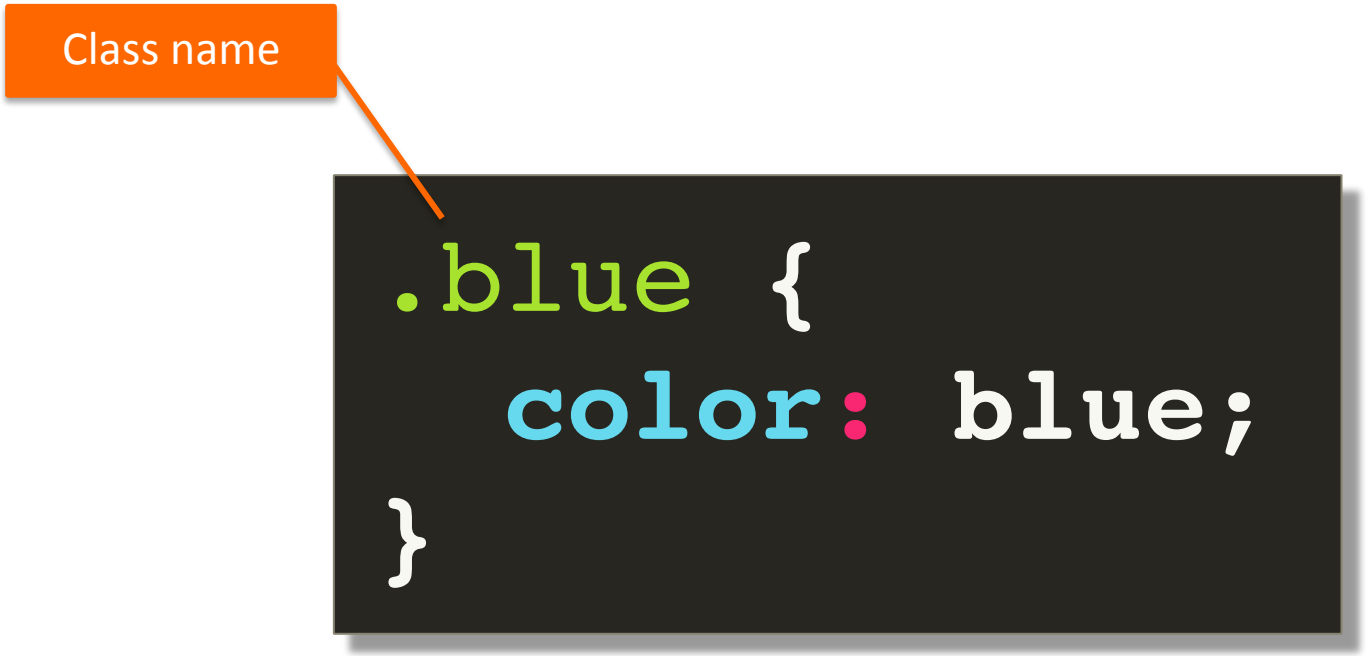


Element Selector

```
p {
  color: blue;
}
```



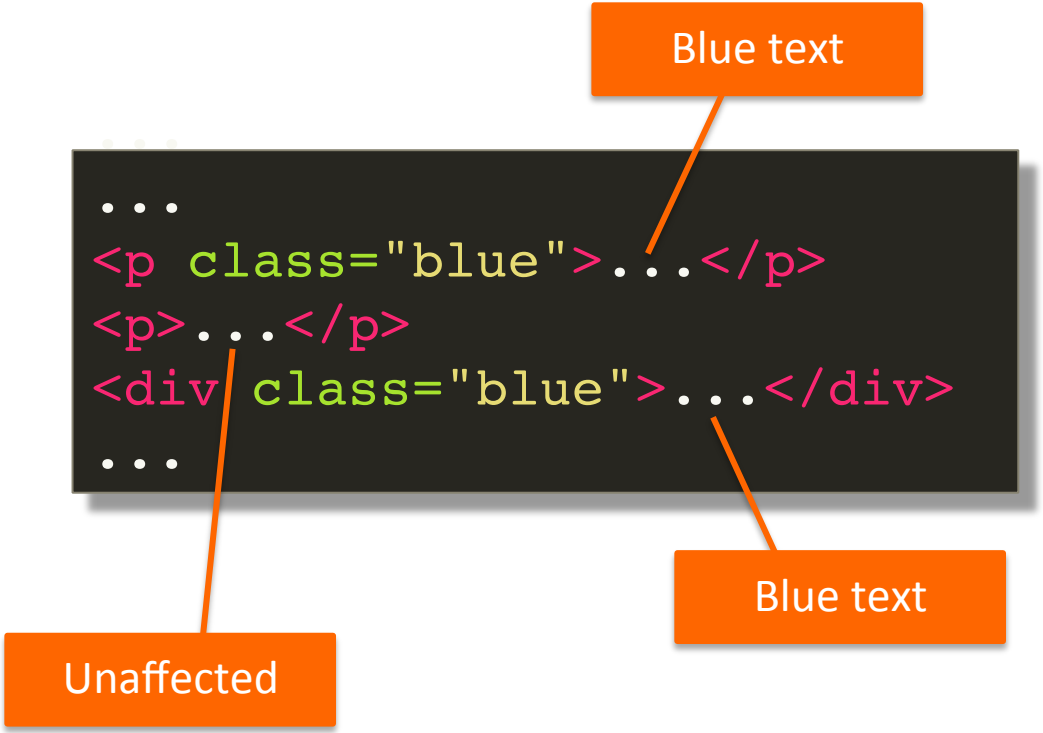
`class` Selector



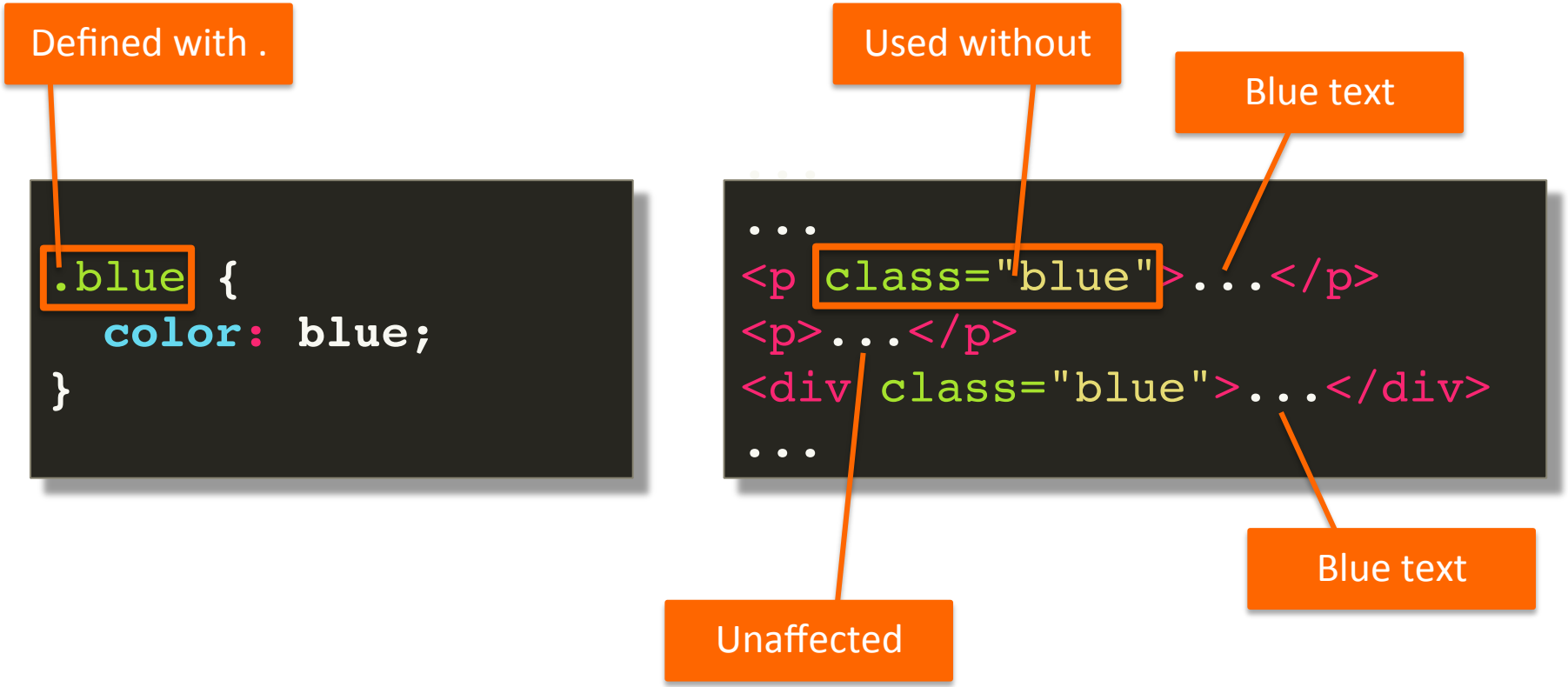
`class` Selector

```
.blue {
  color: blue;
}
```

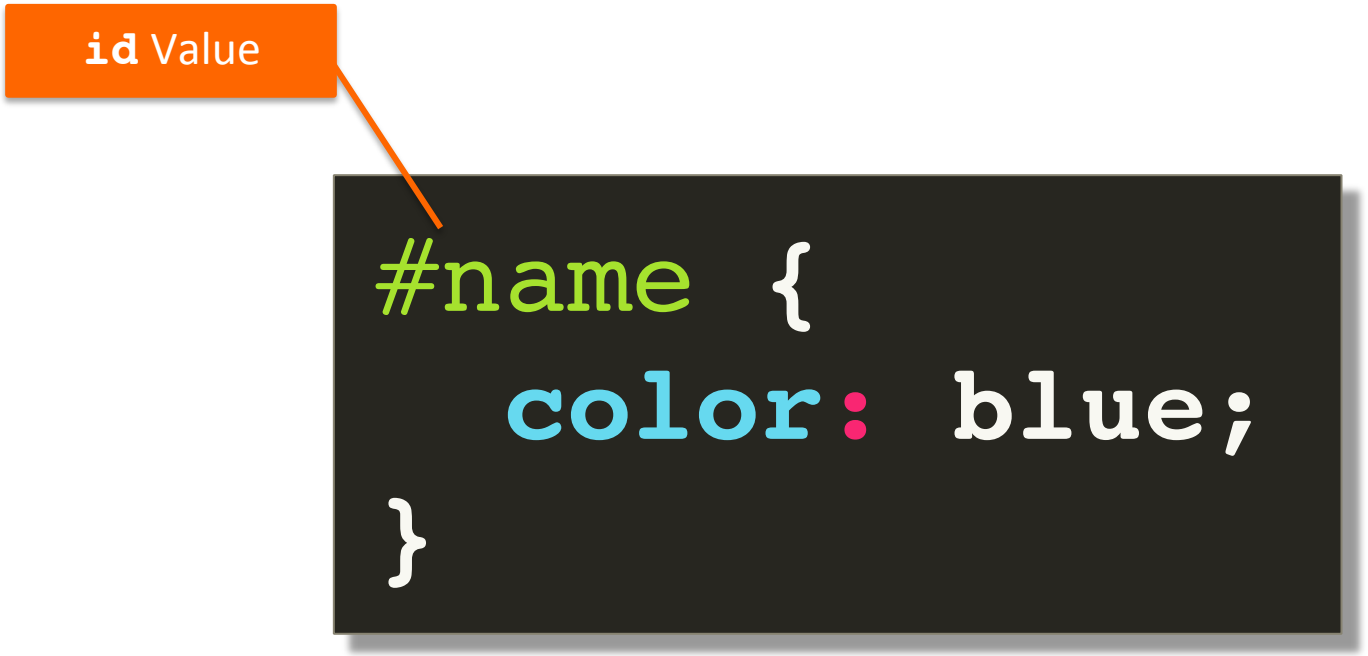
```
...
<p class="blue">...</p>
<p>...</p>
<div class="blue">...</div>
...
```



`class` Selector



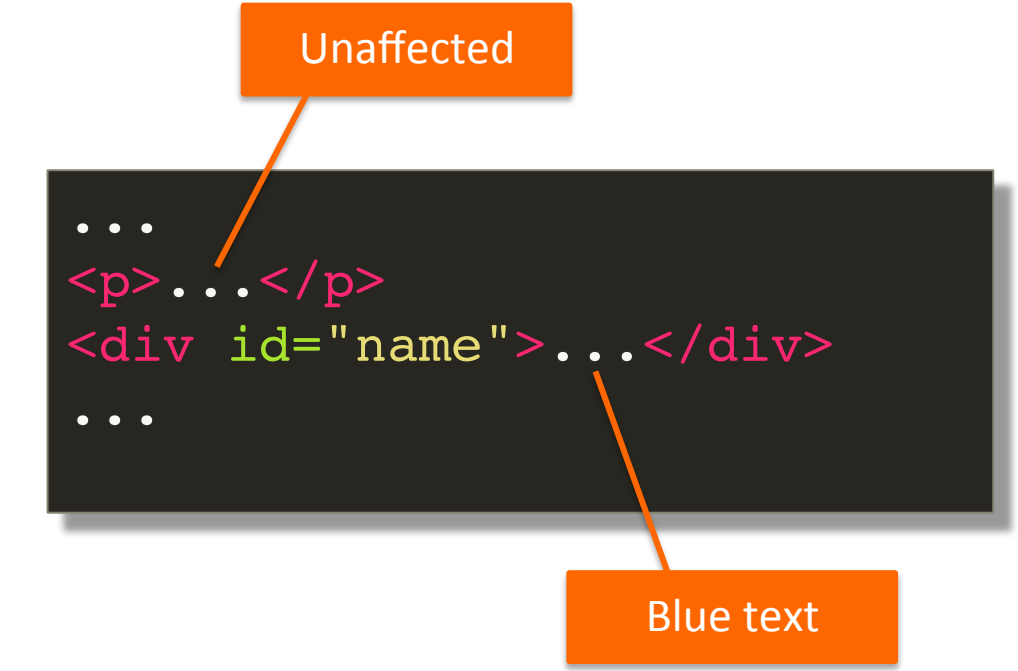
`id` Selector



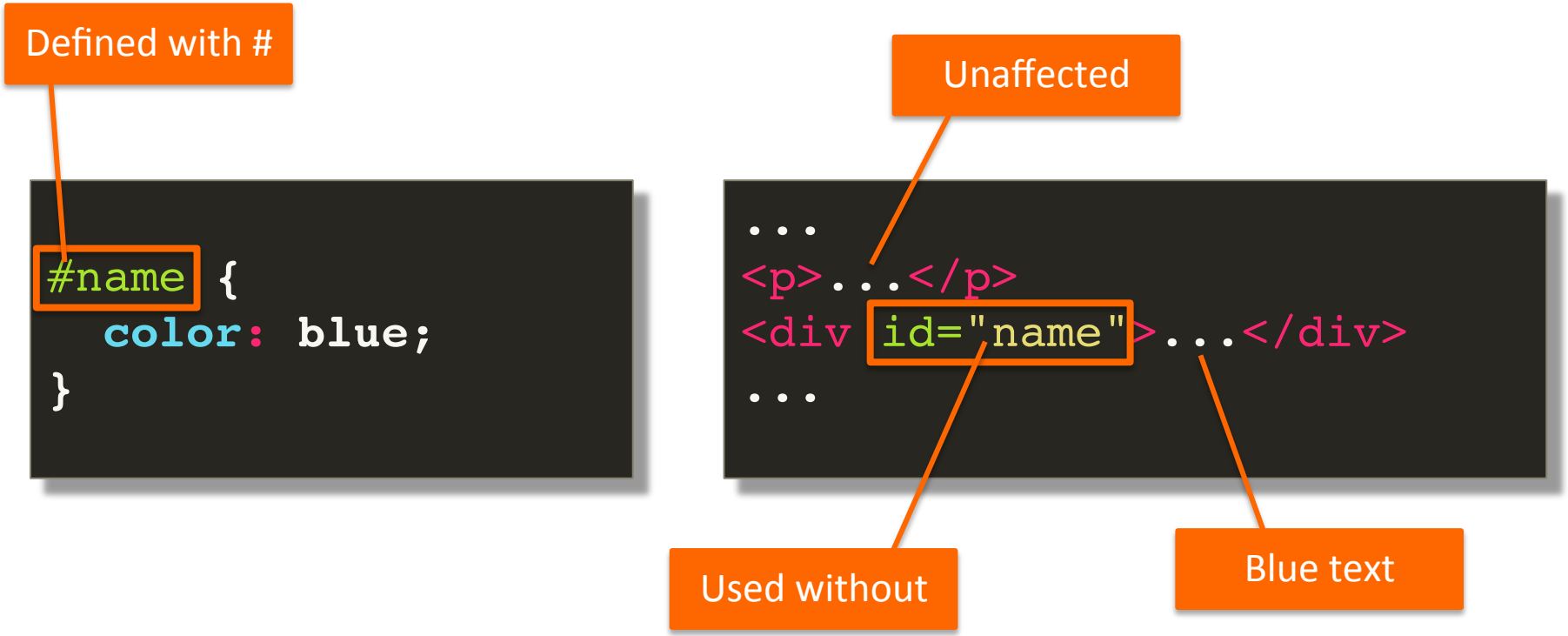
`id` Selector

```
#name {
  color: blue;
}
```

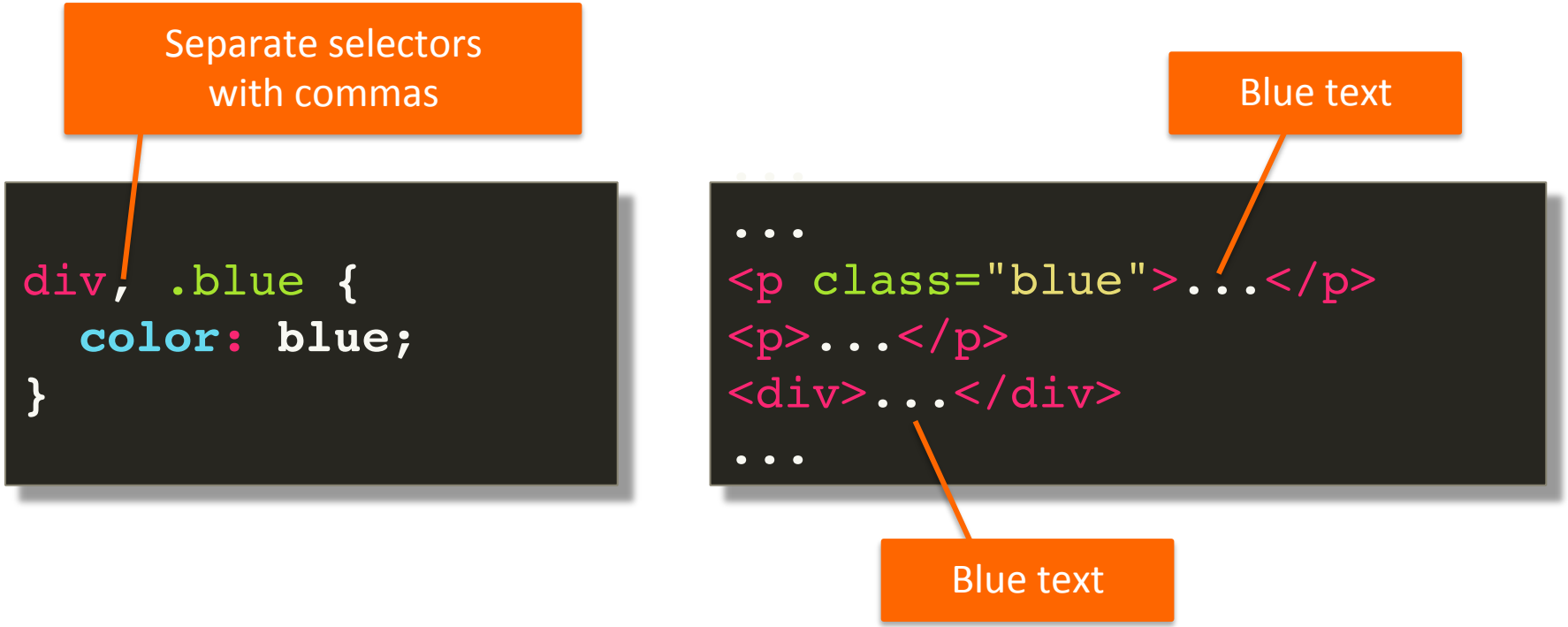
```
...
<p>...</p>
<div id="name">...</div>
...
```



`id` Selector


Defined with #
#name {
color: blue;
}
Unaffected
...
...
...
...
Used without
Blue text

Grouping Selectors



Summary

- ✧ Syntax simple CSS selectors
  - Element
  - `class` (define with .)
  - `id` (define with #)

NEXT:

Combining Selectors

In this lecture, we will discuss…

Combining Selectors

Element With Class Selector

Every **p** that has **class="big"**

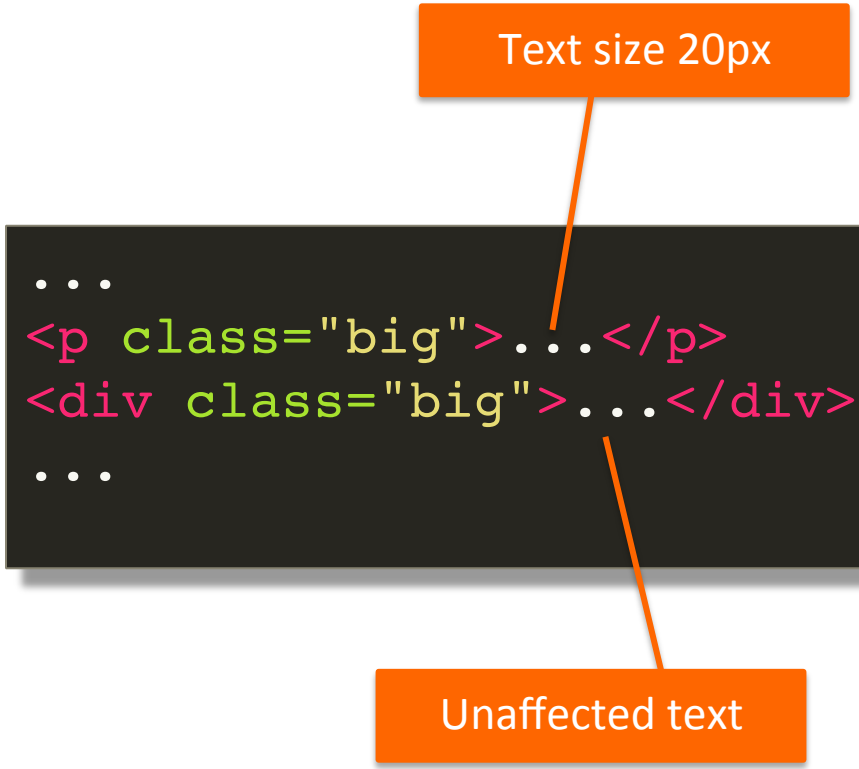
```
p.big {
    font-size: 20px;
}
```

NOTE lack of space between element and class definition

Element With Class Selector

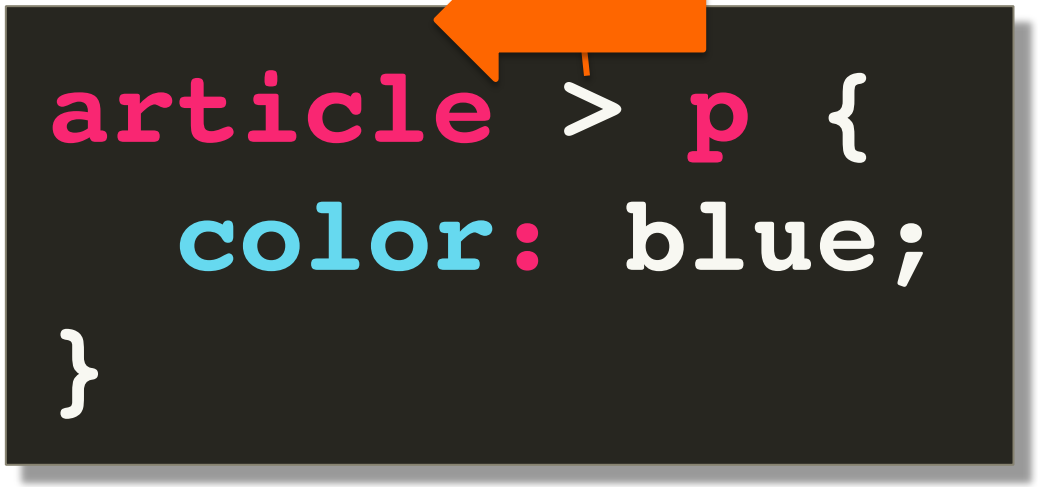
```
p.big {
   font-size: 20px;
}
```

```
...
<p class="big">...</p>
<div class="big">...</div>
...
```



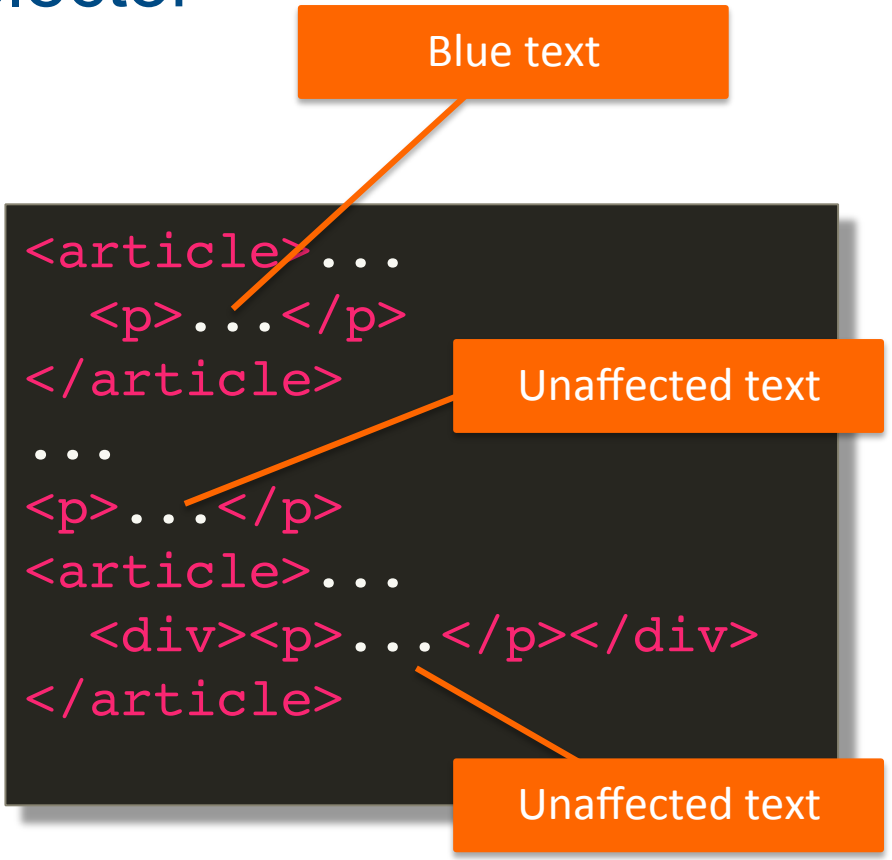
Child Selector

Every **p** that is a **direct** child of **article**



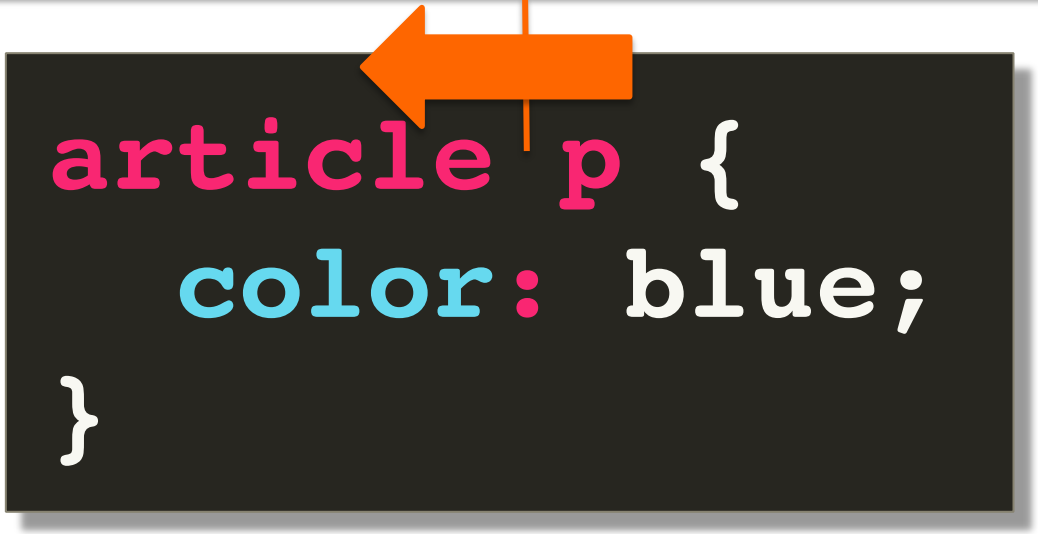
Child Selector

```
article > p {
  color: blue;
}
```



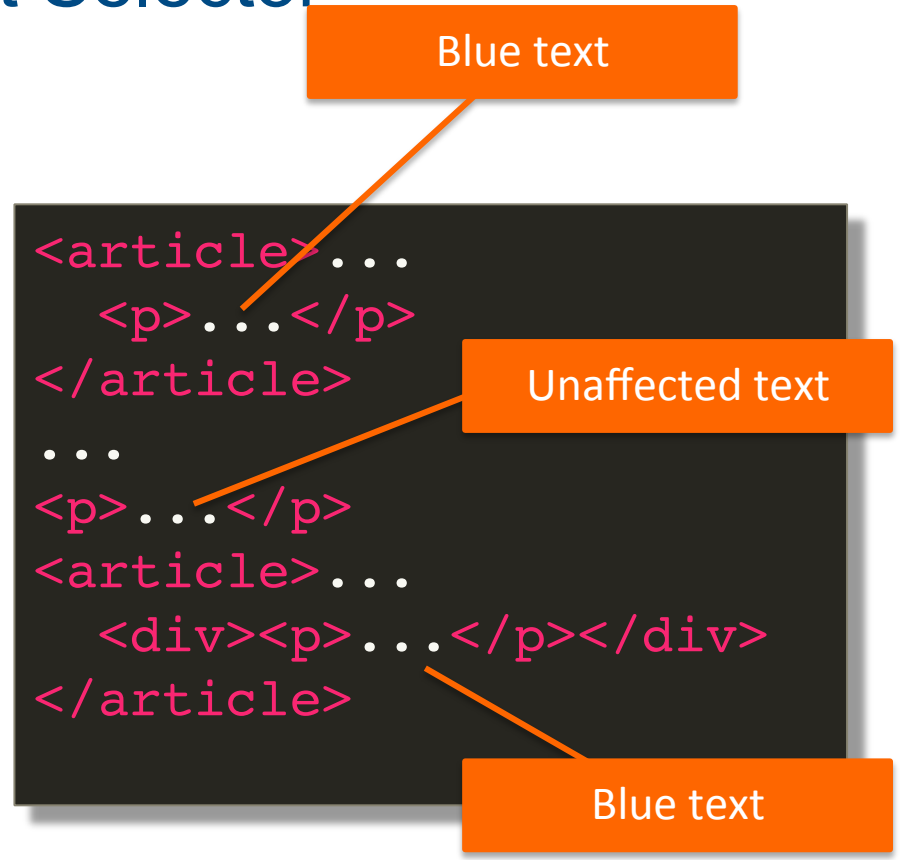
Descendant Selector

Every **p** that is inside (at any level) of **article**



Descendant Selector

```
article p {
  color: blue;
}
```



Not Limited To Element Selectors

```
.colored p {
  color: blue;
}
```

Every **p** that is inside (at any level) an element with **class="colored"**

```
article > .colored {
  color: blue;
}
```

Every element with **class="colored"** that is a **direct** child of **article** element

Summary

- Combining selectors
  - Element with `class` selector (`selector.class`)
  - Child (direct) selector (`selector > selector`)
  - Descendent selector (`selector selector`)
- Didn't cover
  - Adjacent sibling selector (`selector + selector`)
  - General sibling selector (`selector ~ selector`)

NEXT:

Pseudo Class Selectors

In this lecture, we will discuss…

Pseudo-Class Selectors

Pseudo-Class Selector

```
selector:pseudo-class {
    ...
}
```

Pseudo-Class Selector

- Many pseudo-class selectors exist
- We cover:
  - `:link`
  - `:visited`
  - `:hover`
  - `:active`
  - `:nth-child(…)`

Summary

- ✧ Pseudo-class selectors are very powerful
- ✧ Make sure your selector is still readable
  - Simple/Readable > Complicated/Tricky

NEXT:

Style Placement

In this lecture, we will discuss…

Conflict Resolution

Some Concepts

Origin Precedence (when in conflict)

simple rule

Last Declaration Wins

- Remember, HTML is processed sequentially, top to bottom.
- For precedence, think of external CSS as declared at the spot where it's linked to.

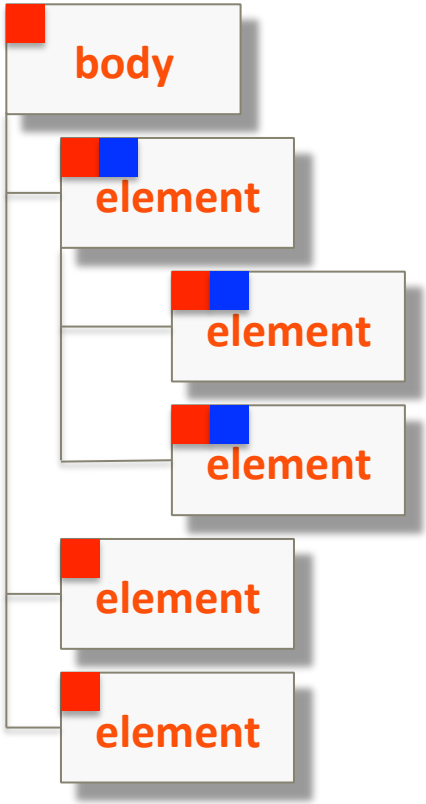
Origin Precedence (when no conflict)

even simpler rule

Declarations Merge

Inheritance

DOM Tree

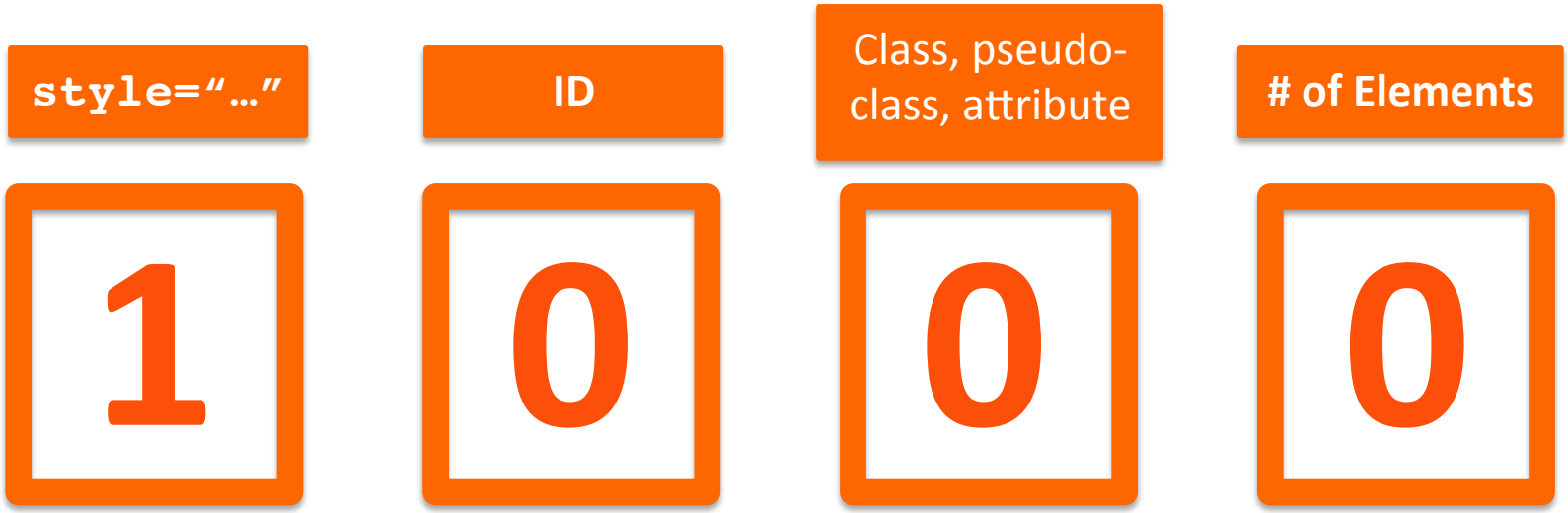


Specificity

simple rule

Most Specific Selector Combination Wins

Specificity (score)




```
<h2 style="color: green;">
```

Specificity (score)

style="…"	ID	Class, pseudo-class, attribute	# of Elements
0	0	0	2

```
div p { color: green; }
```

Specificity (score)

```
div #myParag {
   color: blue;
}
```

?

```
div.big p {
   color: green;
}
```

0	1	0	0

✔

0	0	1	2

✘

Summary

- Cascade: origin, merge; Inheritance, & Specificity
- Provide precise control over targeting content while allowing maximum re-use of styles across your website

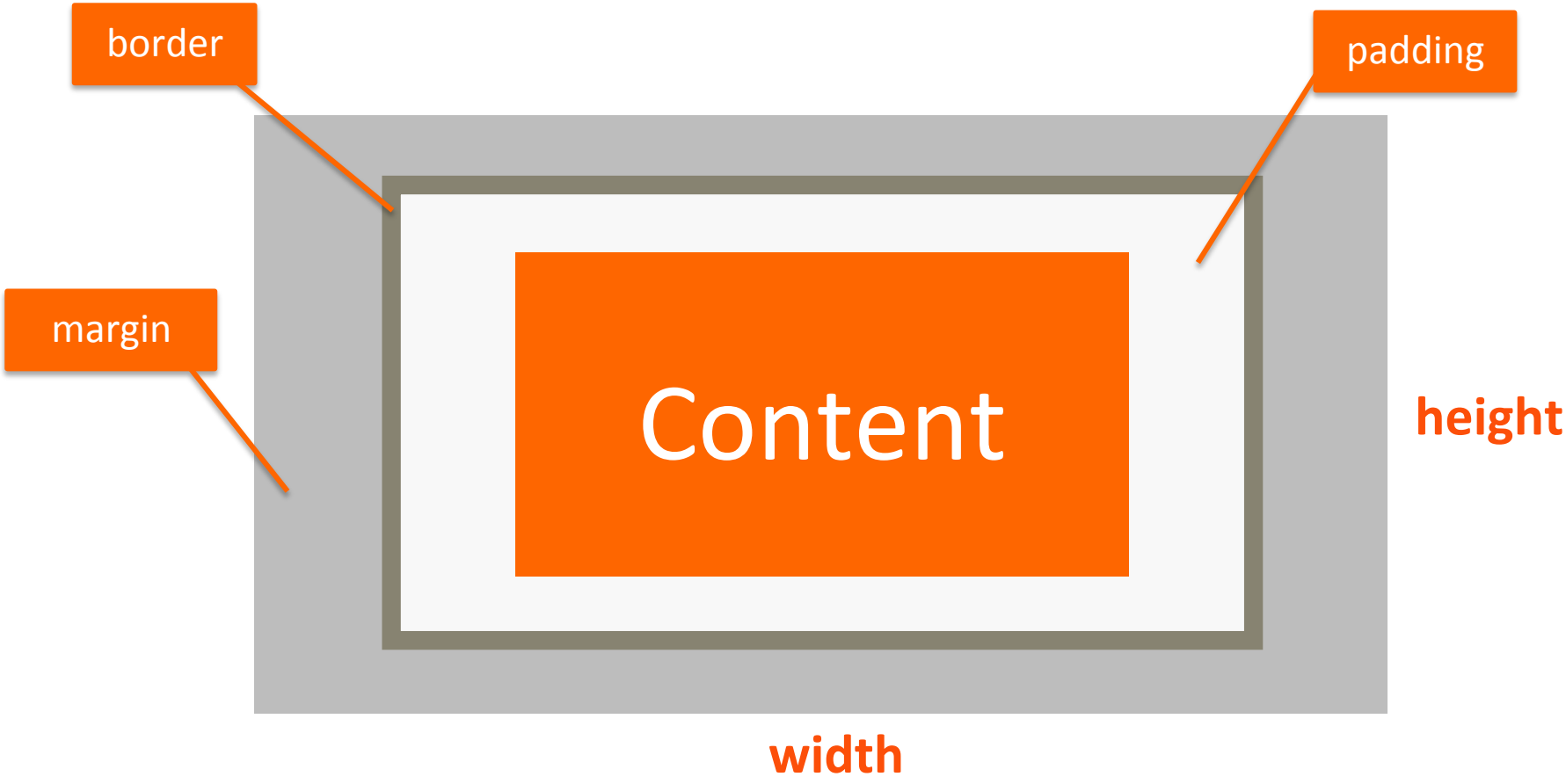
NEXT:

Styling Text

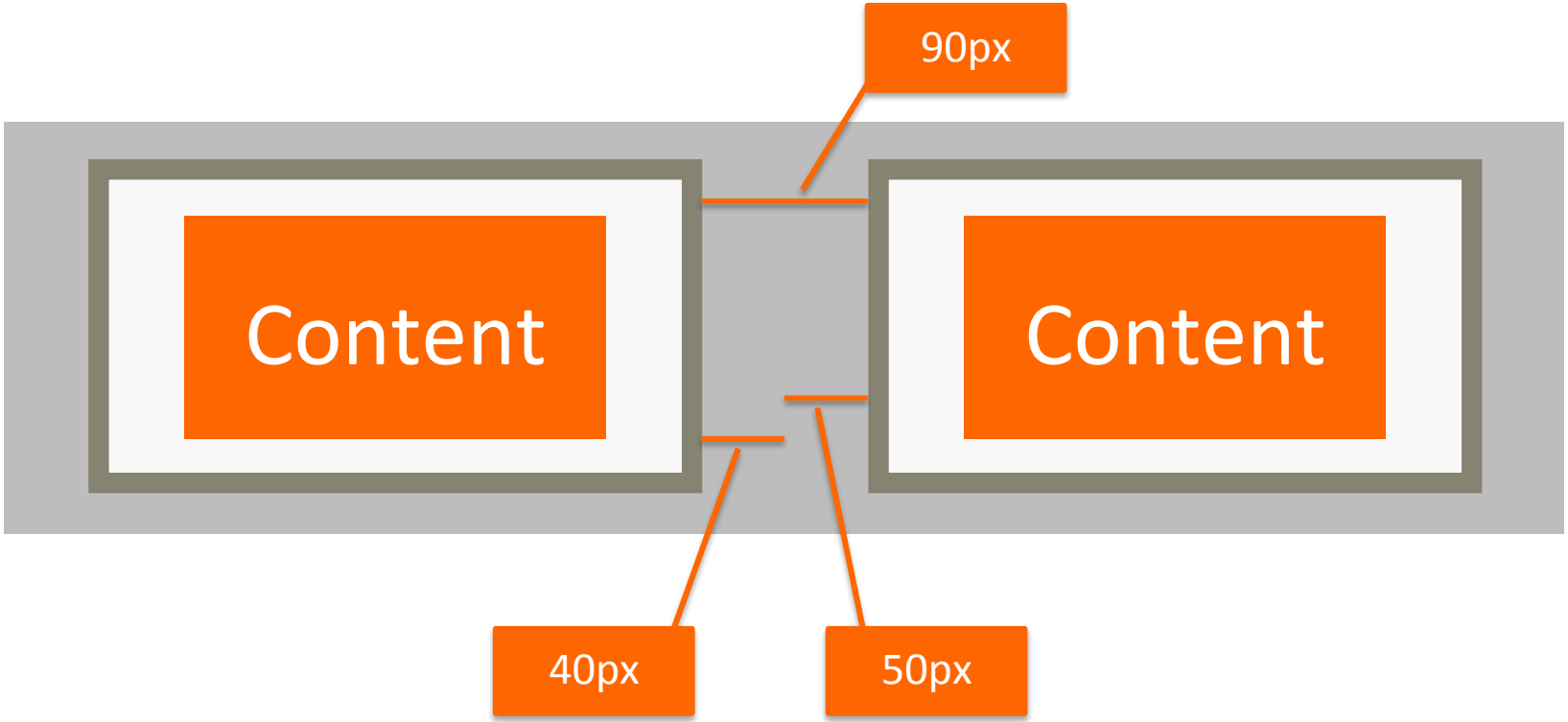
In this lecture, we will discuss…

The Box Model

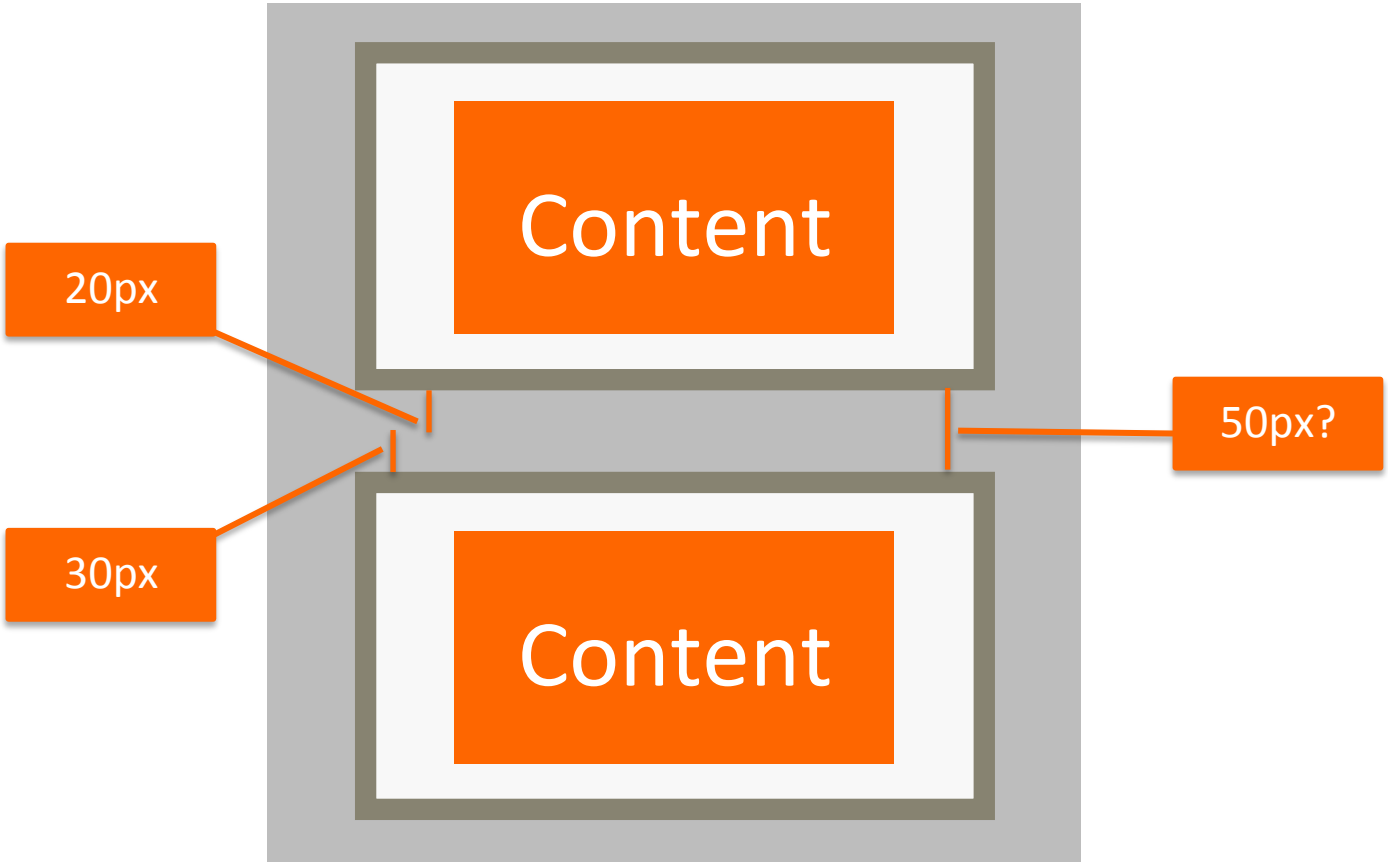
The Box Model



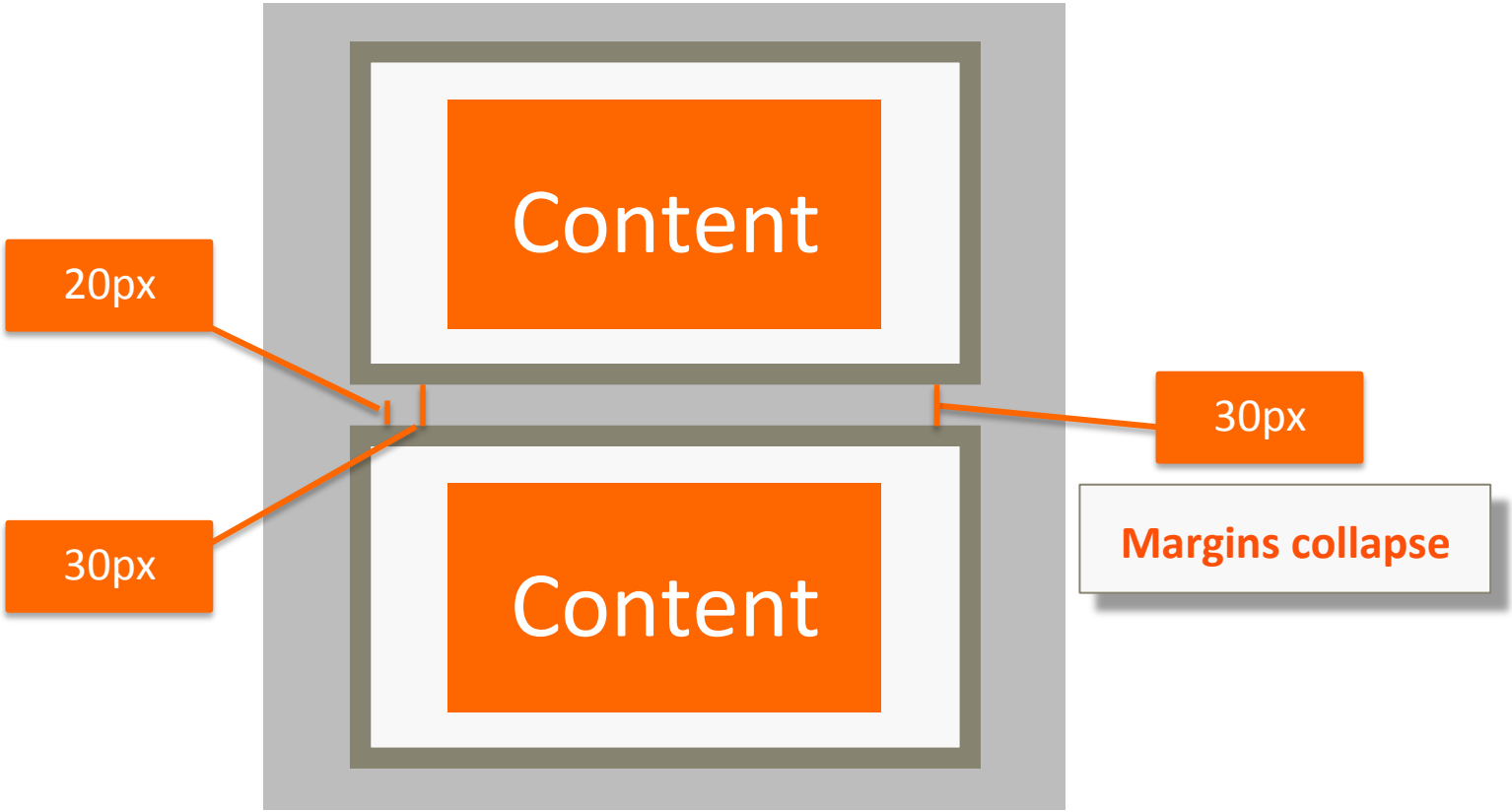
Cumulative Margins



Cumulative Margins?



Cumulative Margins



Summary

- ✧ Box model – essential to understand
  - Prefer **`box-sizing: border-box`**
- ✧ The `*` (universal) selector
- ✧ Cumulative and collapsing margins
- ✧ Content overflow

NEXT:

The `background` Property

In this lecture, we will discuss…

Relative and Absolute Element Positioning

Static Positioning

Normal document flow.
Default for all, except `html`.

- **Positioning offsets are ignored**

Relative Positioning

Element is positioned relative to its position in normal document flow.

- **Positioning CSS (offset) properties are:**
  - `top`, `bottom`, `left`, `right`
- **Element is NOT taken out of normal document flow**
  - **Even if moved, its original spot is preserved**

Relative Positioning

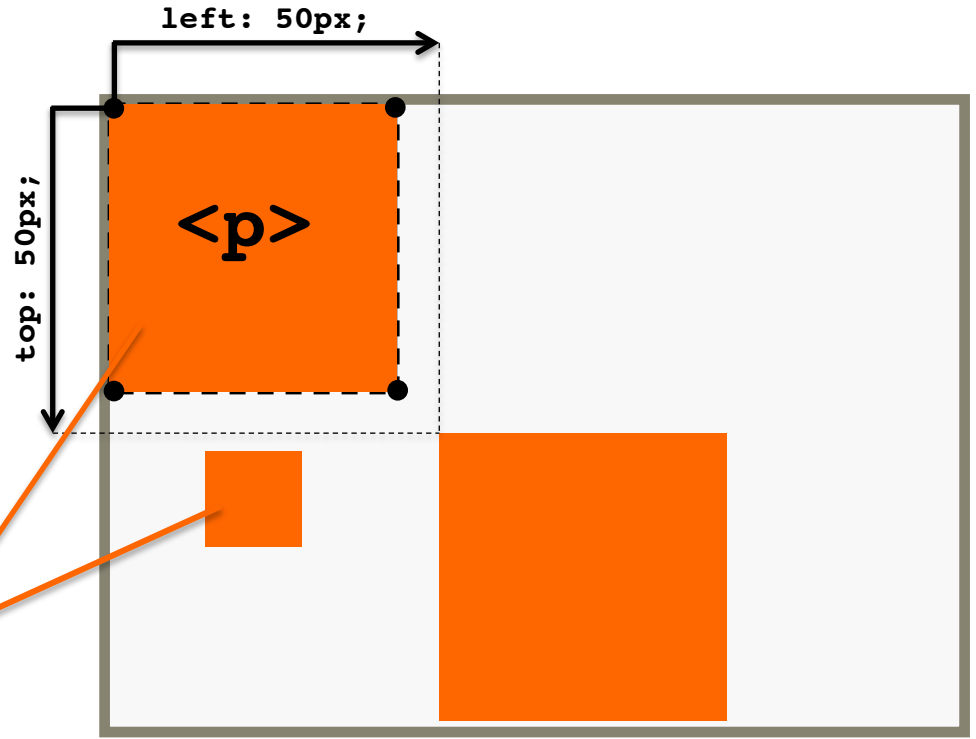
```
p {


}
```

Relative Positioning

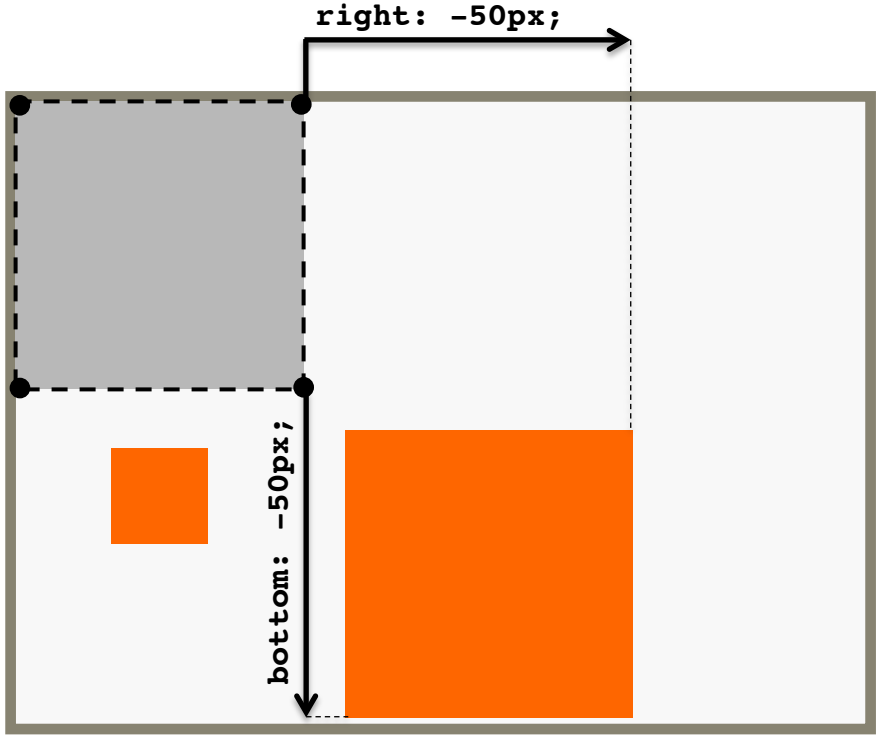
```
p {
    position: relative;
    top:  50px;
    left: 50px;
}
```

from
from



Relative Positioning

```
p {
    position: relative;
    bottom: -50px;
    right:  -50px;
}
```



Absolute Positioning

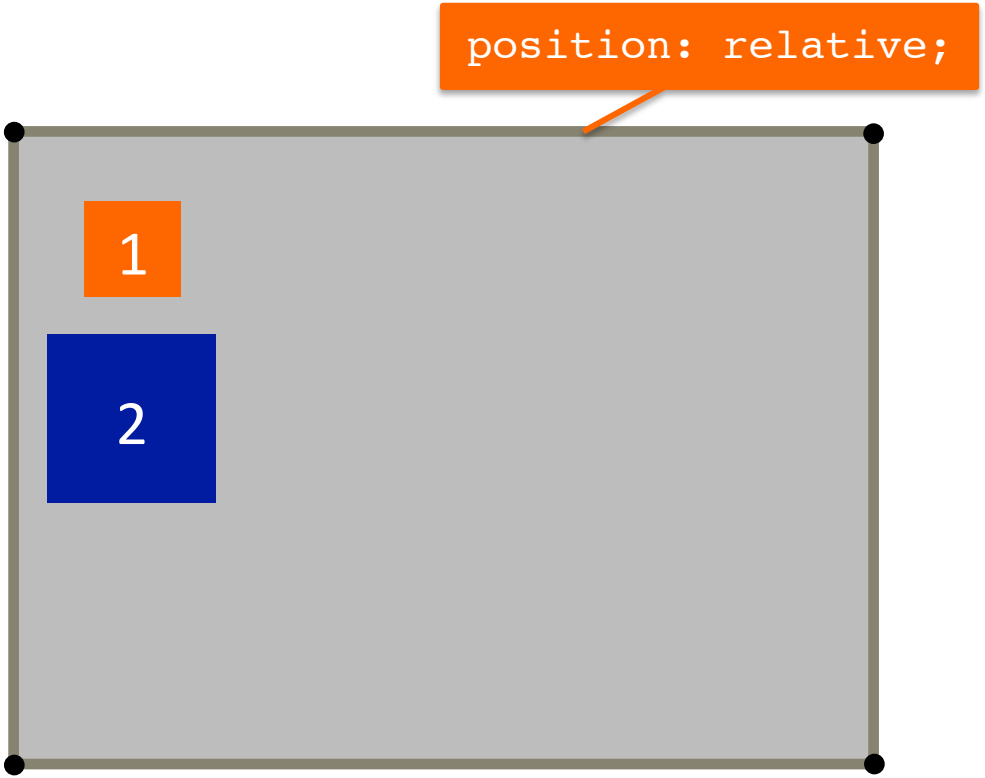
All offsets (top, bottom, left, right) are relative to the position of the nearest ancestor which has positioning set on it, other than static.

- By default, `html` is the only element that has non-static positioning set on it (relative).
- Element is taken out of normal document flow

Absolute Positioning

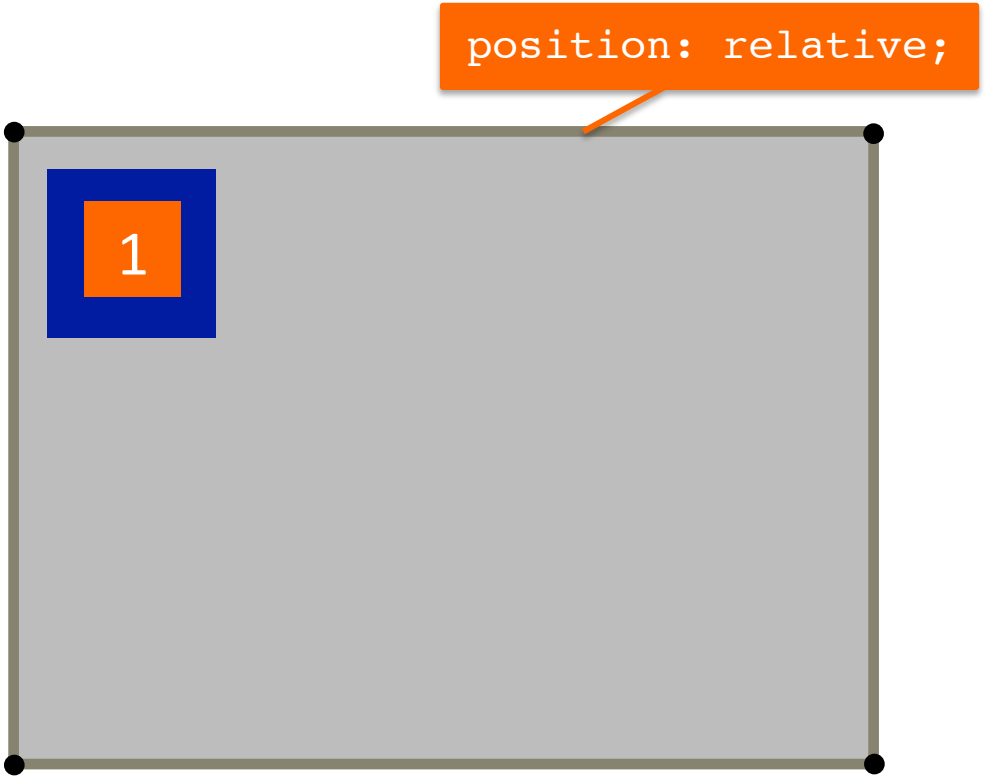
```
p { /* #1 */

}
```



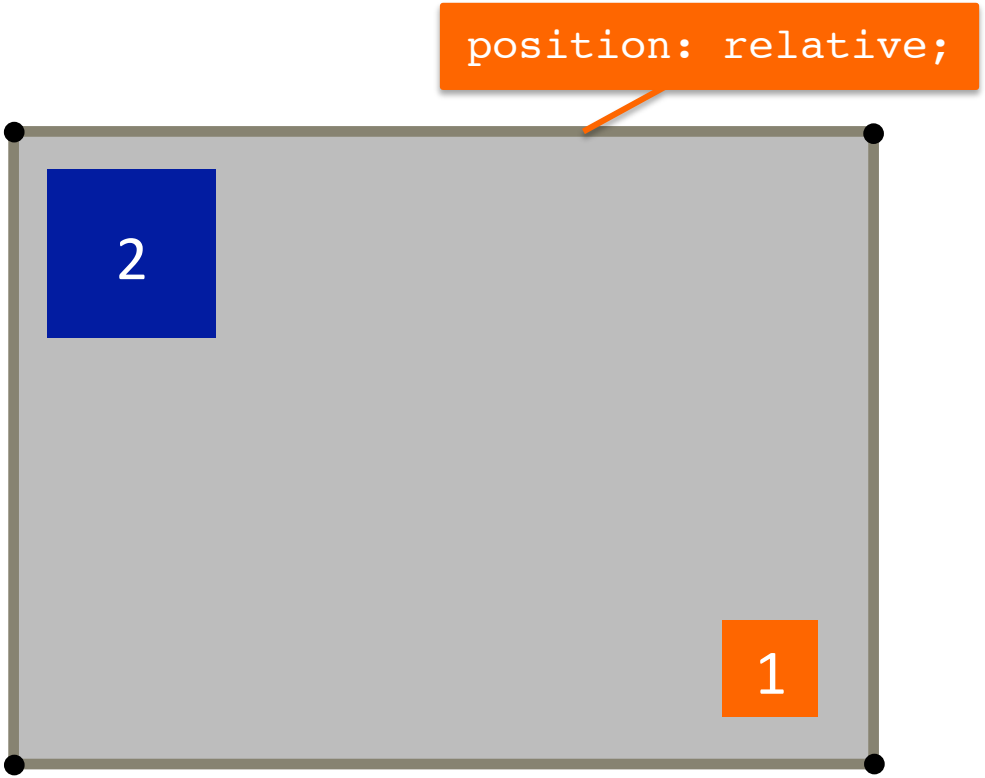
Absolute Positioning

```
p { /* #1 */
    position: absolute;
}
```



Absolute Positioning

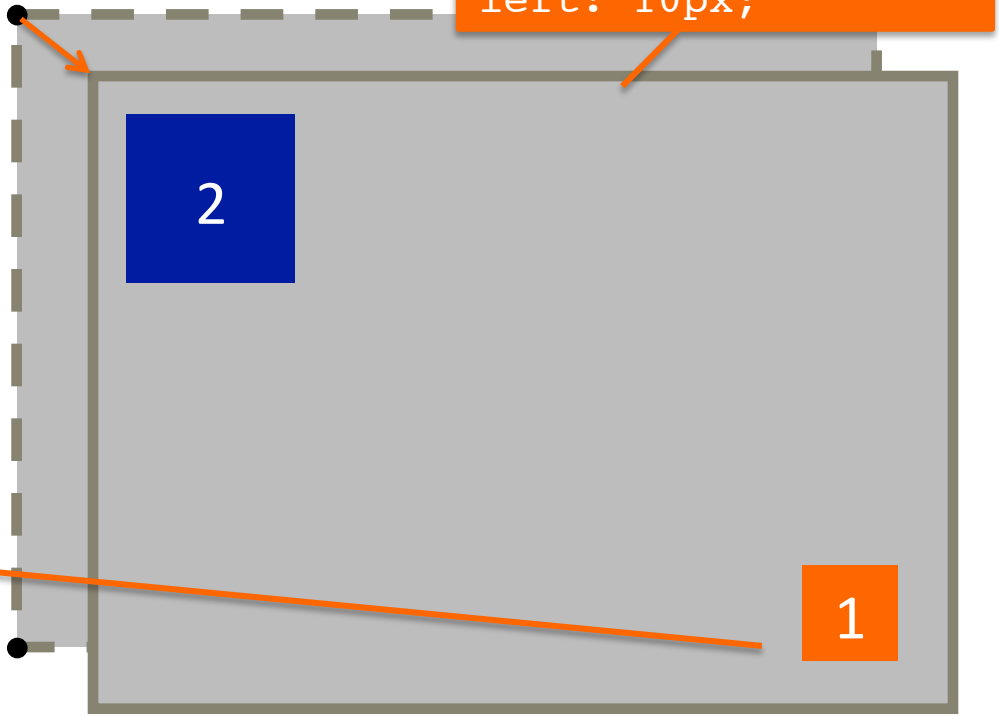
```
p { /* #1 */
    position: absolute;
    bottom: 10px;
    right:  10px;
}
```



Absolute Positioning

```
p { /* #1 */
    position: absolute;
    bottom: 10px;
    right:  10px;
}
```

```
position: relative;
top: 10px;
left: 10px;
```



If container element is offset, everything inside is offset with it.

Summary

- Static positioning is default for all elements, except html
- Relative positioning offsets the element relative to its normal document flow position
- Absolute positioning is relative to closest ancestor which has positioning set to non-static value
- Offsetting the relative container element offsets its contents as well

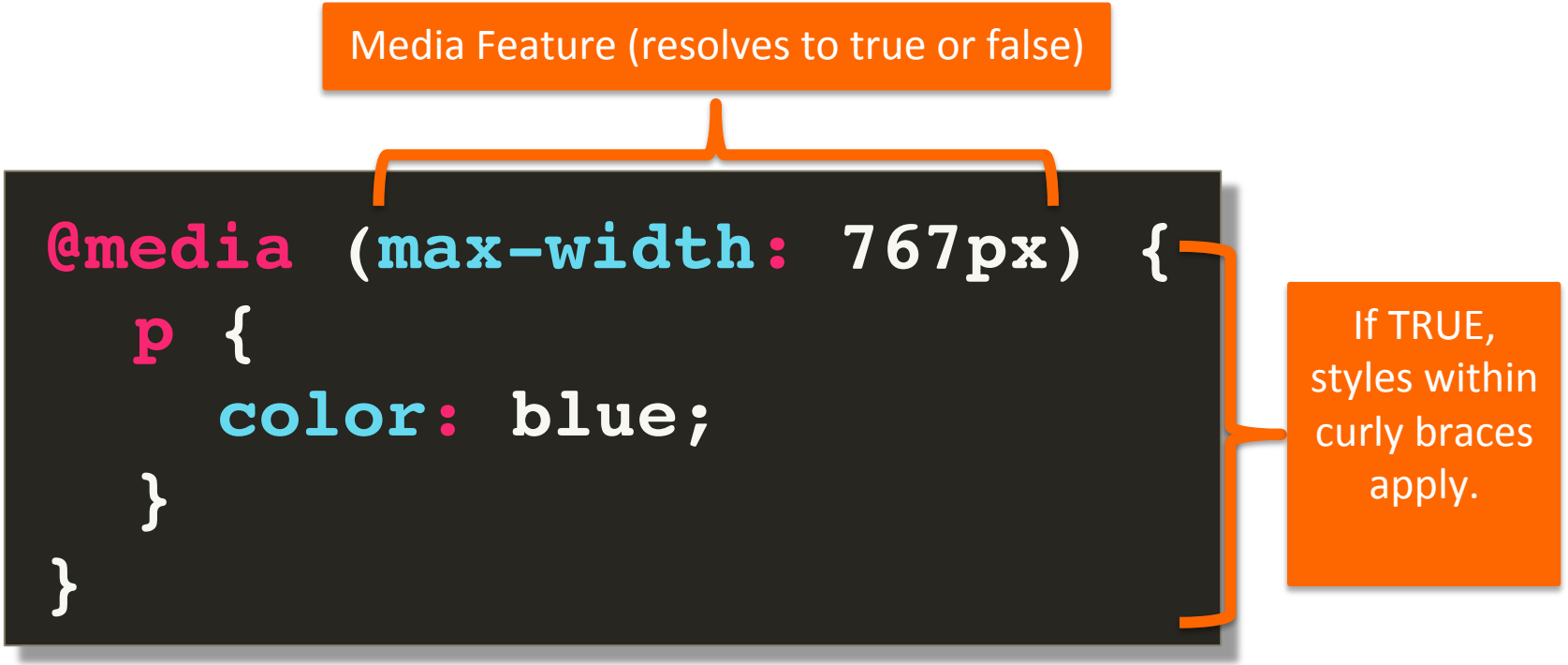
NEXT:

Using Media Queries

In this lecture, we will discuss…

Media Queries

Media Query Syntax



Media Query Common Features

```
@media (max-width: 800px) {…}
```

```
@media (min-width: 800px) {…}
```

```
@media (orientation: portrait) {…}
```

```
@media screen {…}
```

```
@media print {…}
```

• • •

Media Query Common Logical Operators

Devices with width within a range

```
@media (min-width: 768px) and (max-width: 991px) {…}
```

Comma is equivalent to OR:

```
@media (max-width: 767px) , (min-width: 992px) {…}
```

. . .

Media Query Common Approach

```
p { color: blue; } /* base styles */
…
@media (min-width: 1200px) {
…
}

@media (min-width: 992px) and (max-width: 1199px) {
…
}
…
```

Careful not to overlap range boundaries!

Summary

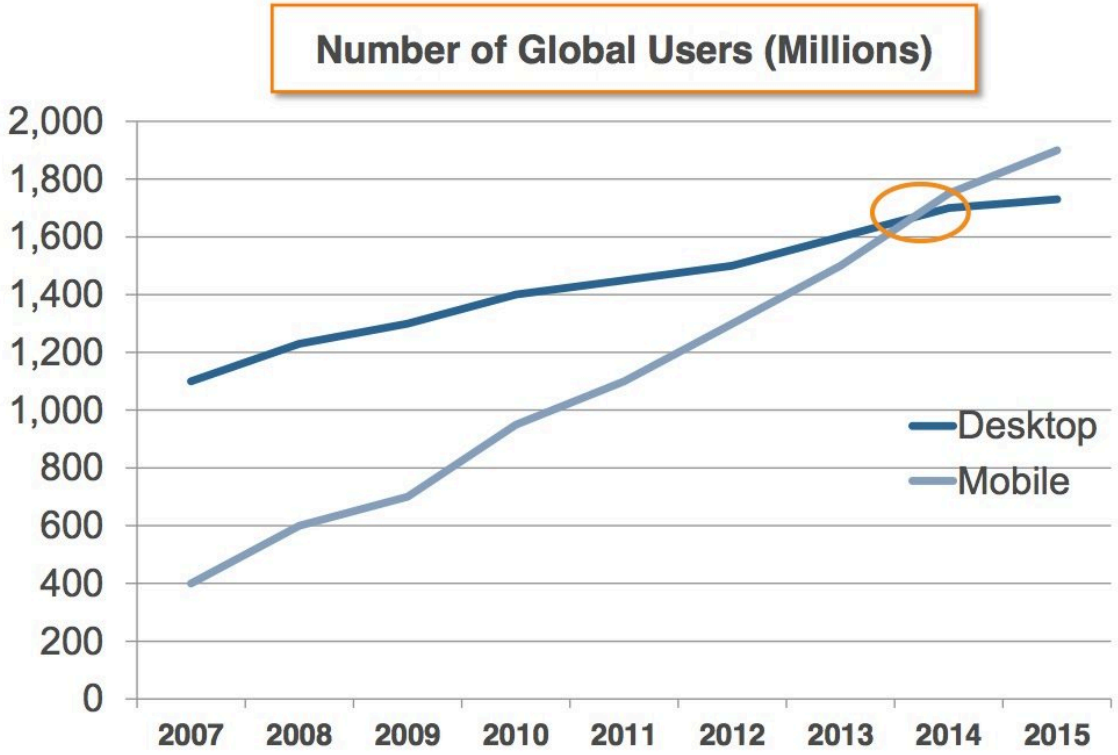
- Basic syntax of a media query
  - @media (media feature)
  - @media (media feature) logical operator (media feature)
- Remember not to overlap breakpoints
- Usually, you provide base styling
  - Then, change or add to them in each media query

NEXT:

Responsive Layout

In this lecture, we will discuss…

Responsive Design



comScore.

Source: Morgan Stanley Research

What device do we need to support?

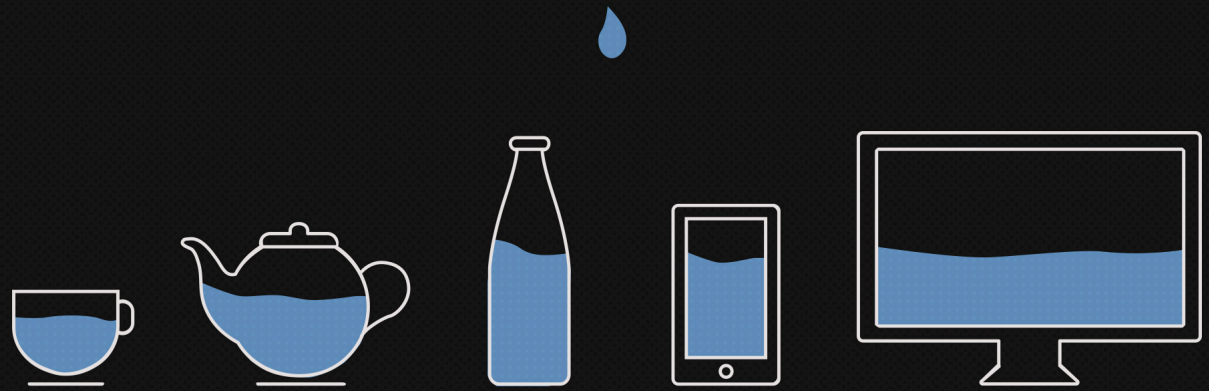
ALL OF THEM

What is a Responsive Web Site?

Site designed to adapt its layout to the viewing environment by using fluid, proportion-based grids, flexible images, and CSS3 media queries

Paraphrased from Wikipedia

CONTENT IS LIKE WATER



"You put water into a cup it becomes the cup.
You put water into a bottle it becomes the bottle.
You put it in a teapot, it becomes the teapot."

Josh Clark *(originally Bruce Lee)* - Seven deadly mobile myths

Illustration by Stéphanie Walter

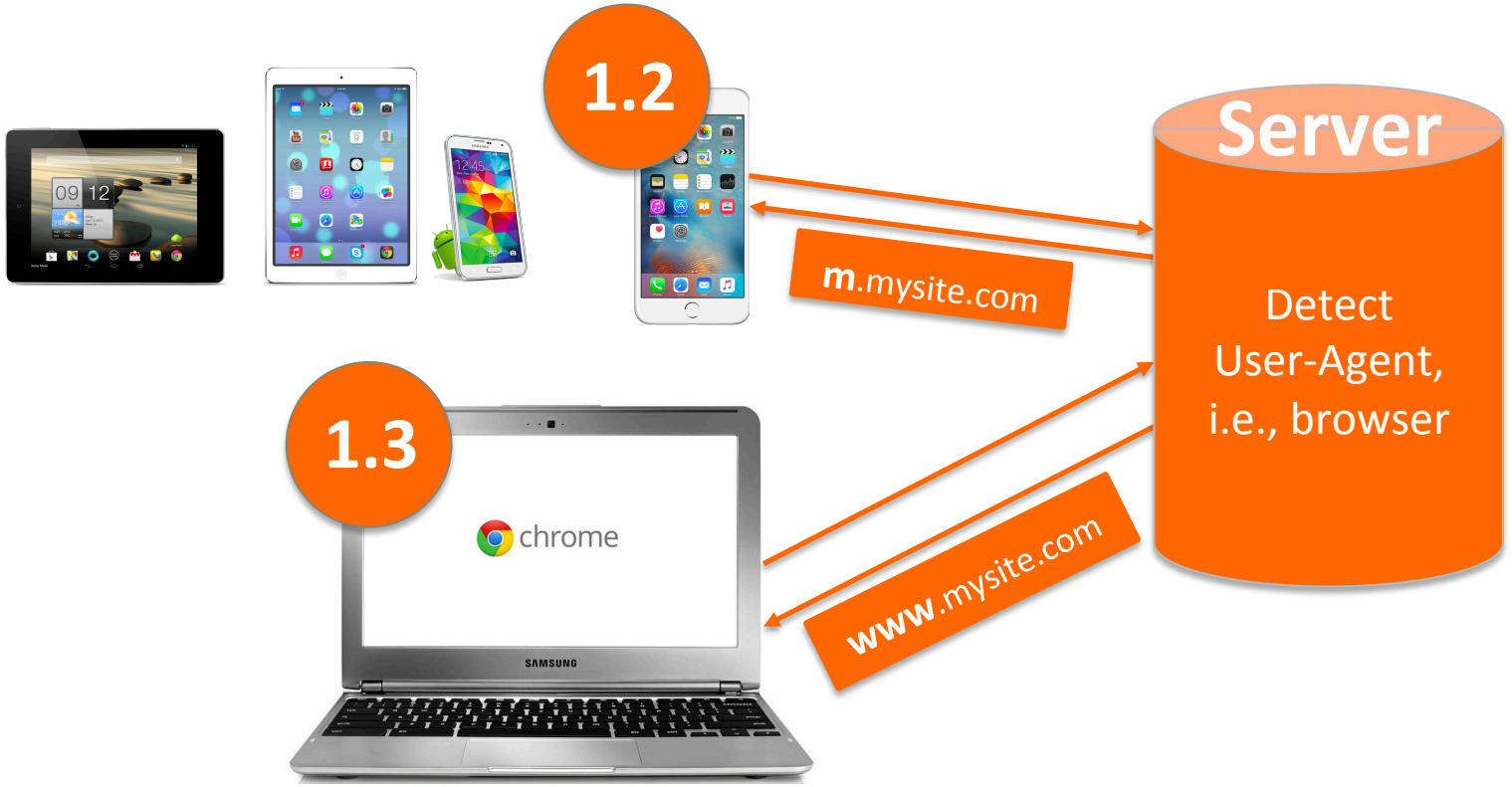
What is a Responsive Web Site?

Site designed to adapt its layout to the viewing environment by using fluid, proportion-based grids, flexible images, and CSS3 media queries

Paraphrased from Wikipedia

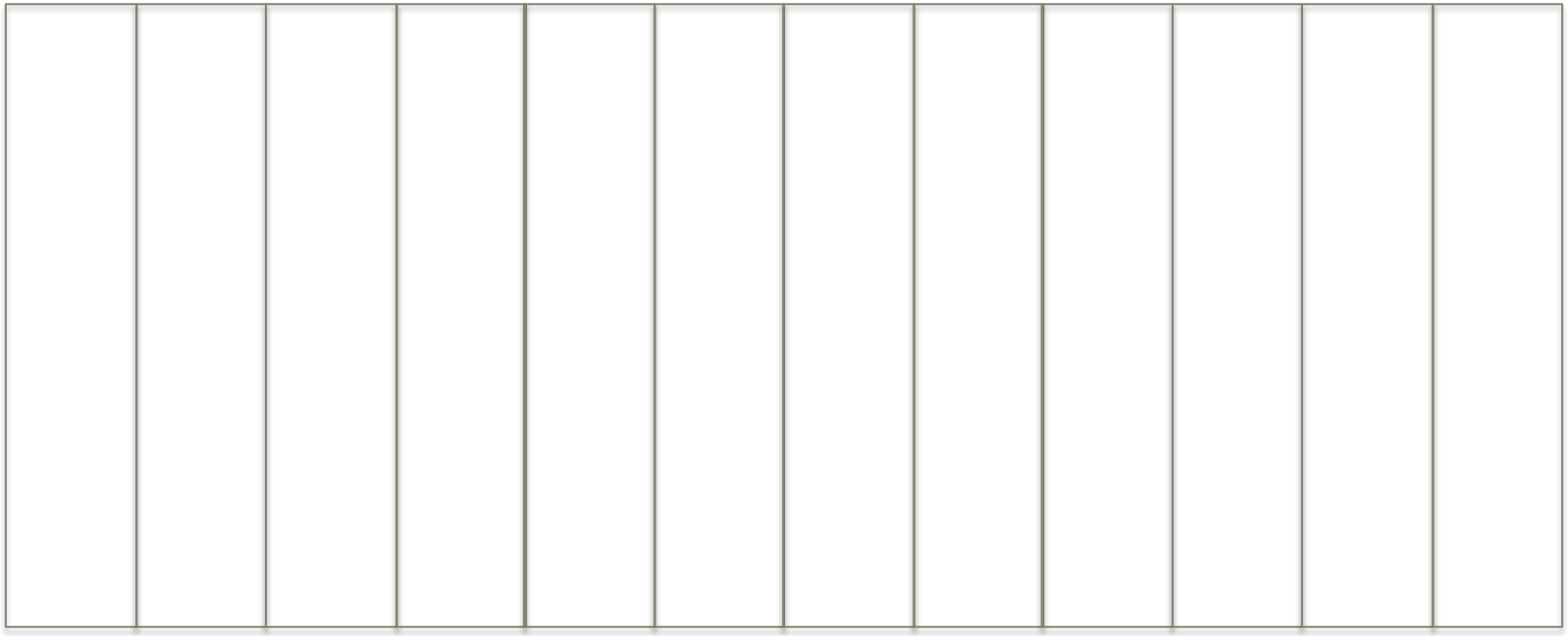
- **Site's layout adapts to the size of the device**
- **Content verbosity or its visual delivery may change**

Alternative to Responsive Design:

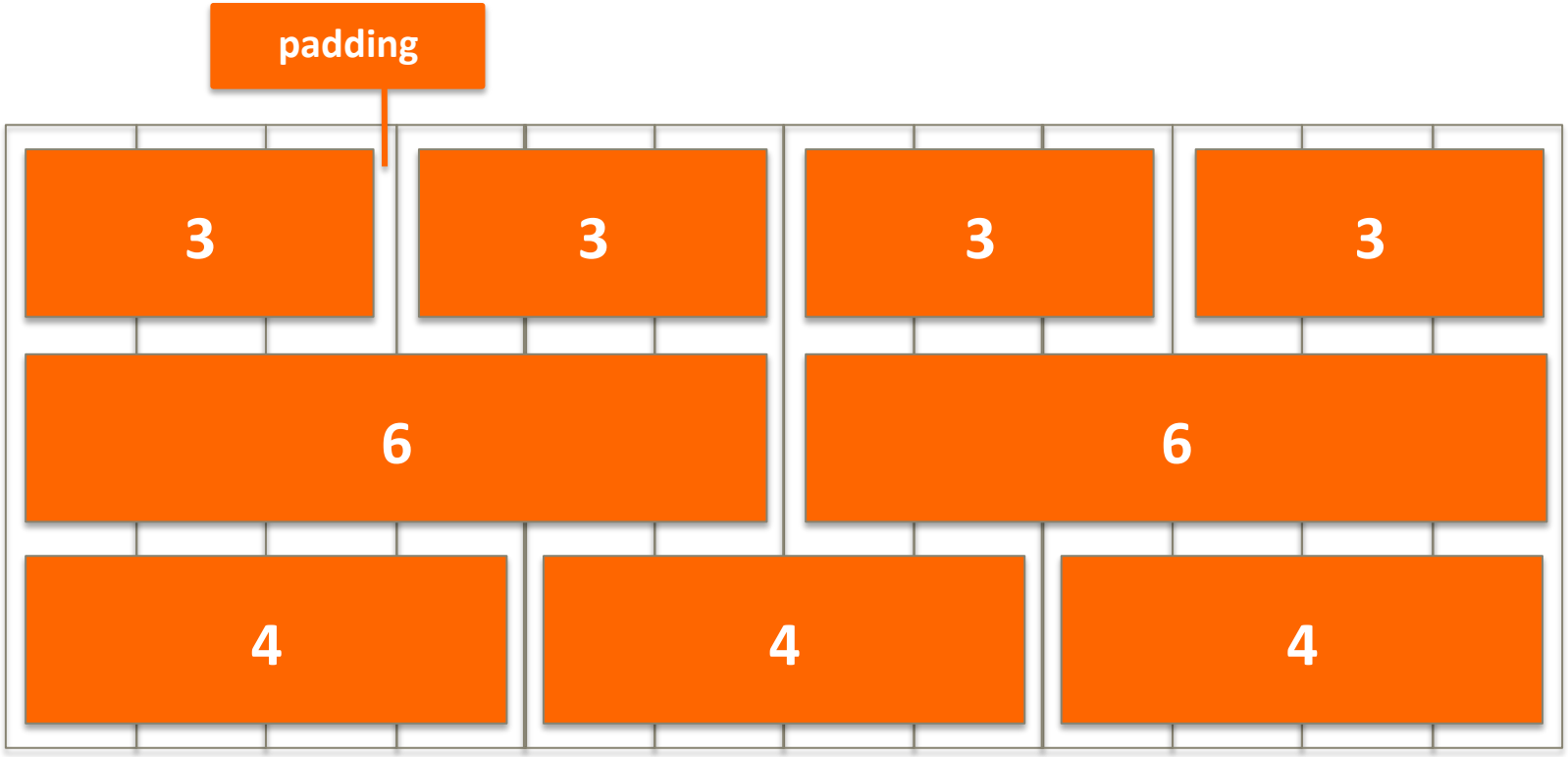


12-Column Grid Responsive Layout

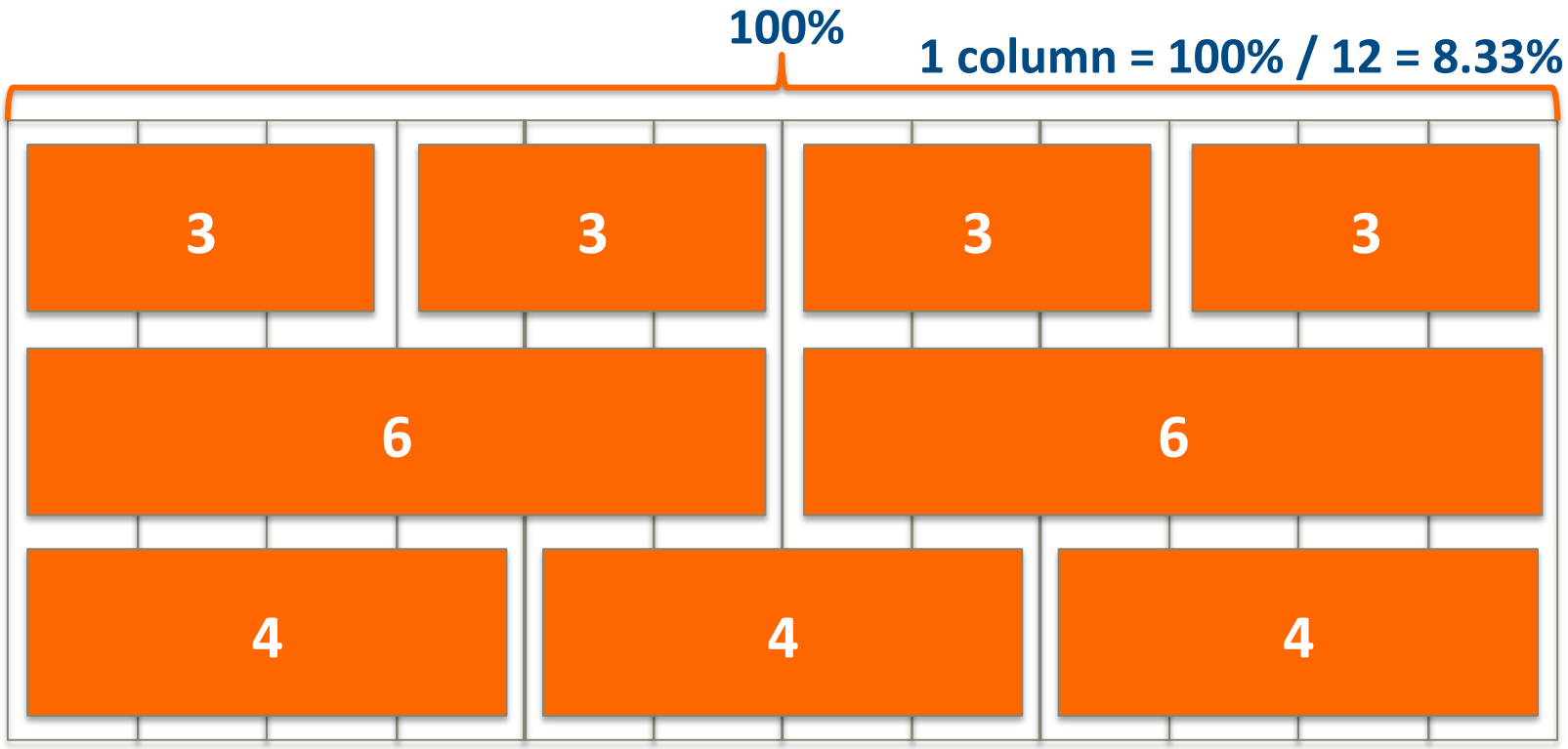
factors of 12: 1, 2, 3, 4, 6, 12



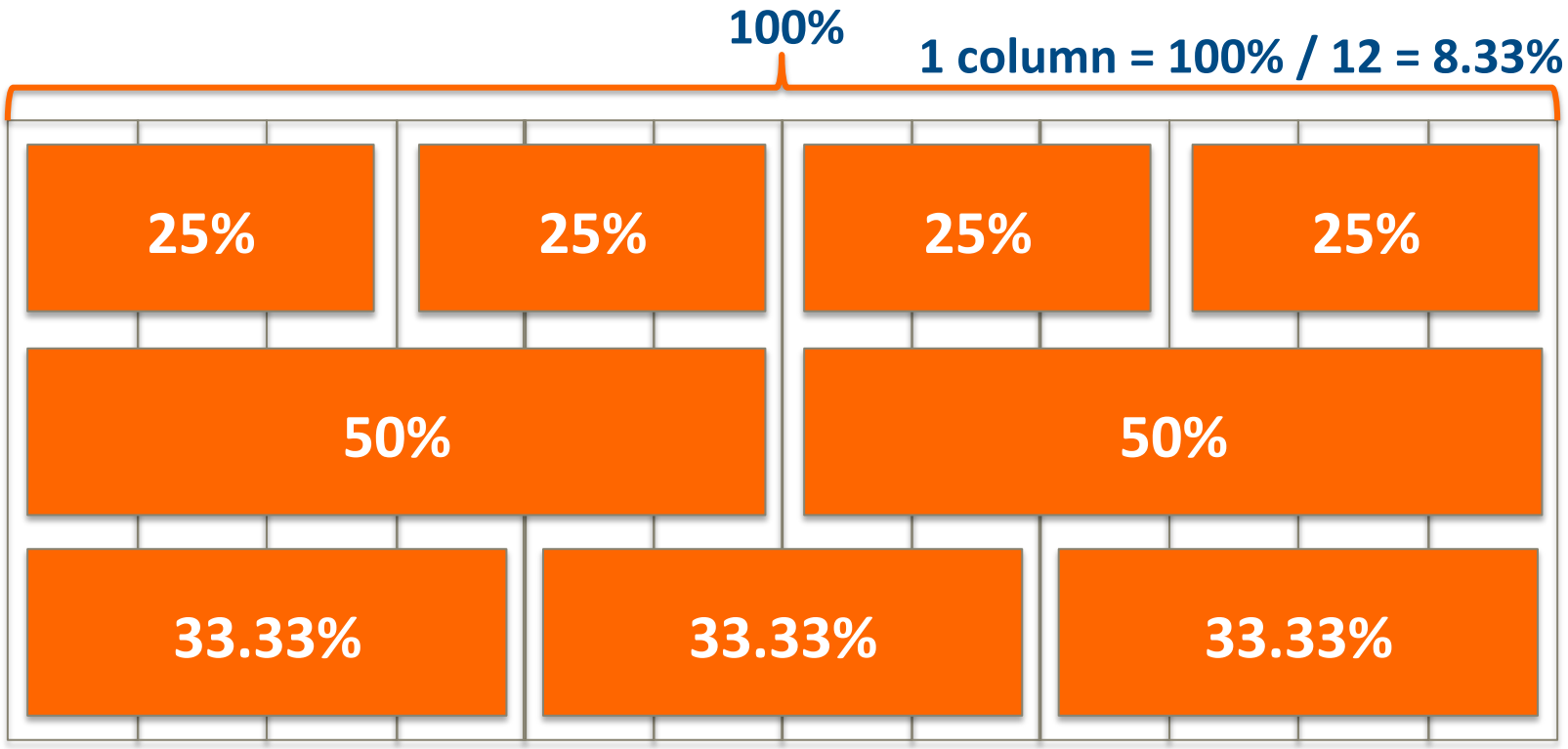
12-Column Grid Responsive Layout



12-Column Grid Responsive Layout

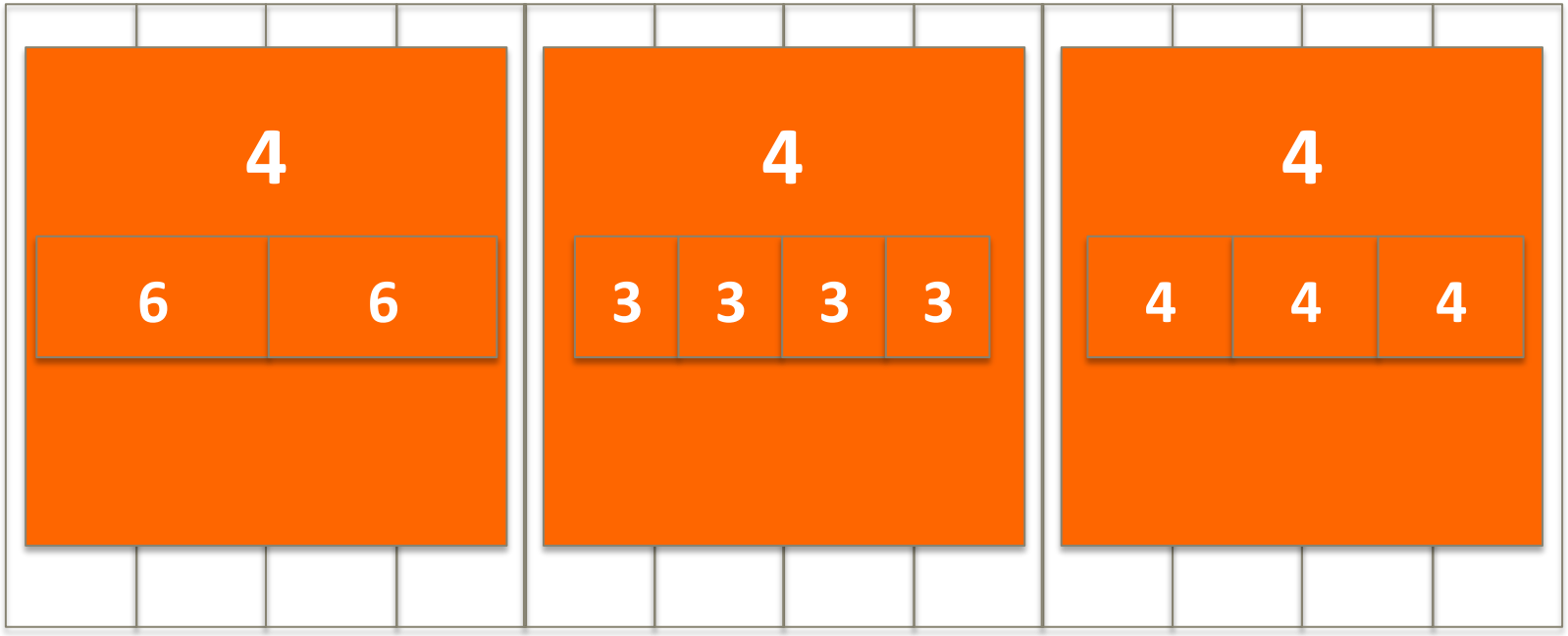


12-Column Grid Responsive Layout



12-Column Grid Responsive Layout

Nested grids



Summary

- Need and idea of responsive design
- 12-column grid layout
  - Use % to achieve fluid width (wrt browser width)
- Viewport meta tag to turn off default mobile zooming

NEXT:

Introduction to Twitter Bootstrap Framework

In this lecture, we will discuss…

Introduction to Twitter Bootstrap

What is Bootstrap?

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

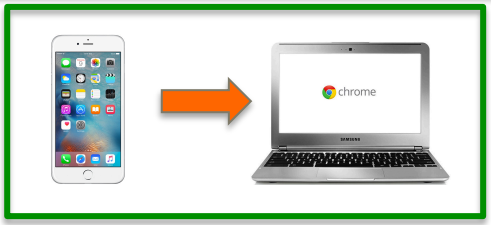
What is Bootstrap?

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

Mostly CSS classes

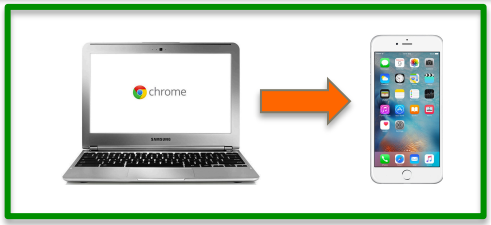
What is Bootstrap?

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.



✔

Which to code first?



✔

What is Bootstrap?

✔ **Mobile First == PLAN mobile from the start**

✔ **CSS Framework is mobile ready**

Disadvantages of Bootstrap (vs. Writing Your Own)?

#1 complaint

Too big, too bloated

- **A lot of features you will probably never use**
  - ***Use selective download***
- **You can write your own that's more targeted/smaller**
  - ***But it will take you a LOT longer to write it as well***

Summary

- Intro to the most popular project on GitHub
- Mobile First – importance of planning for mobile from the start
- Barebones Bootstrap web page

NEXT:

Bootstrap Grid System

In this lecture, we will discuss…

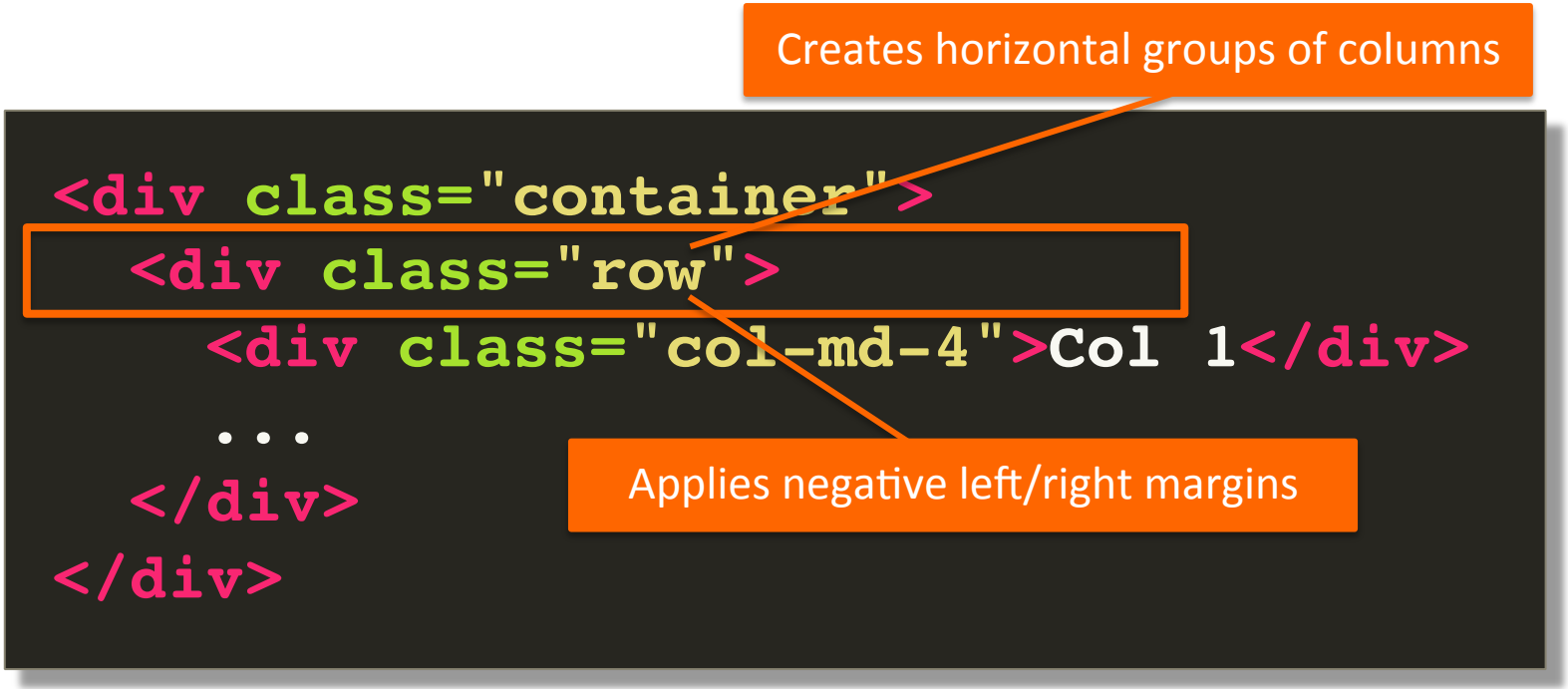
The Bootstrap Grid System

Bootstrap Grid System Basics

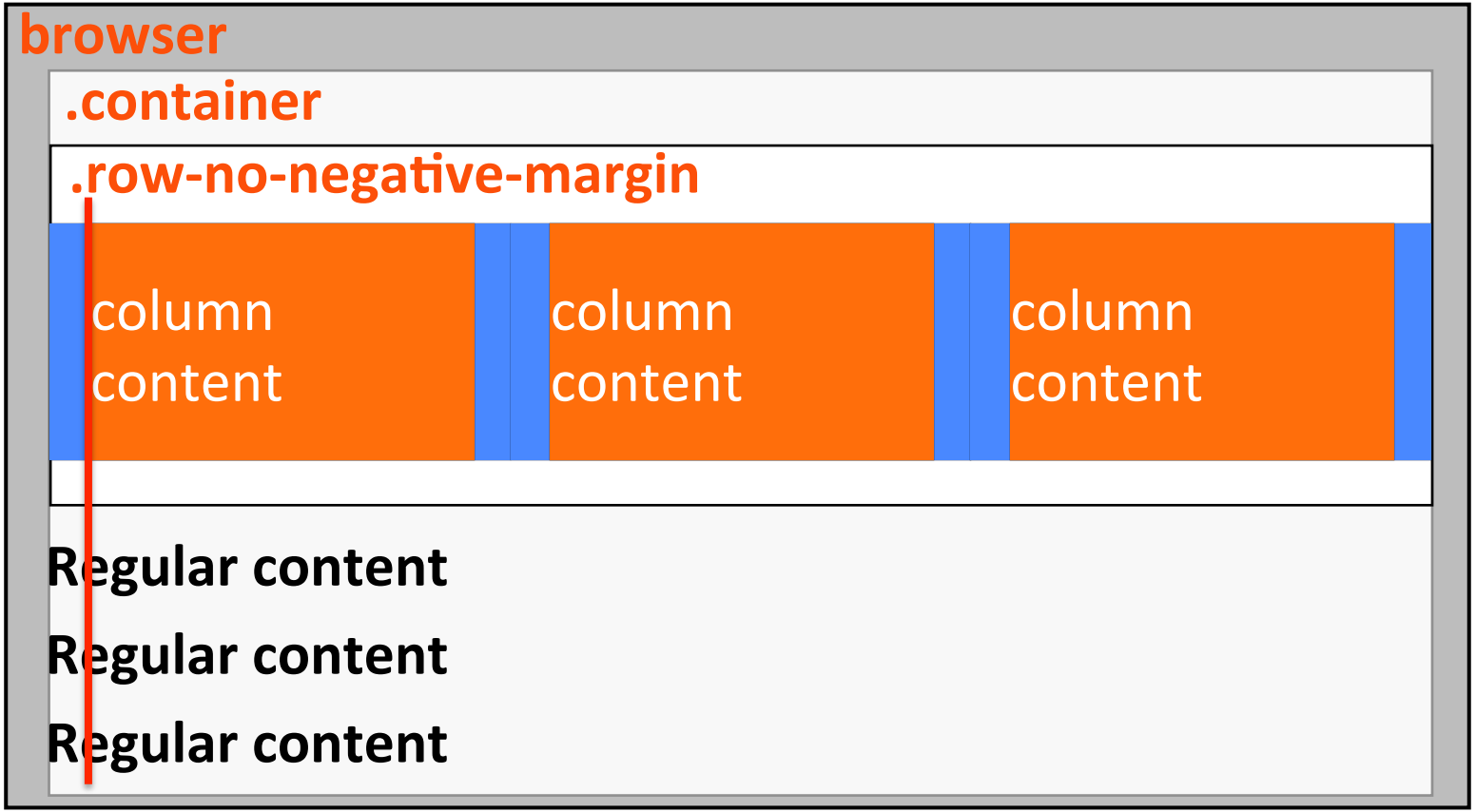
Must be inside container (or container-fluid)

```
<div class="container">
  <div class="row">
    <div class="col-md-4">Col 1</div>
    ...
  </div>
</div>
```

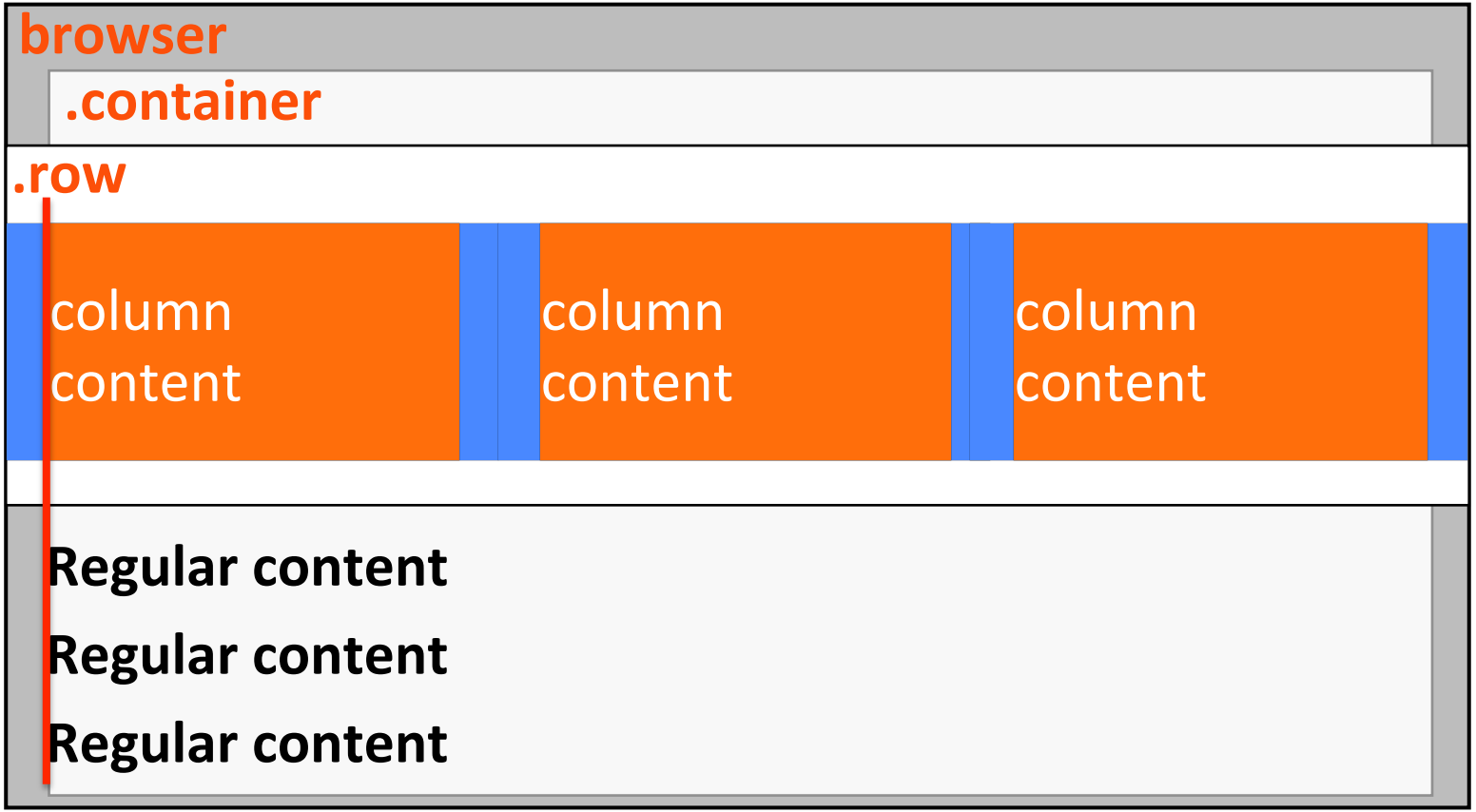
Bootstrap Grid System Basics



Why Negative Row Margin?

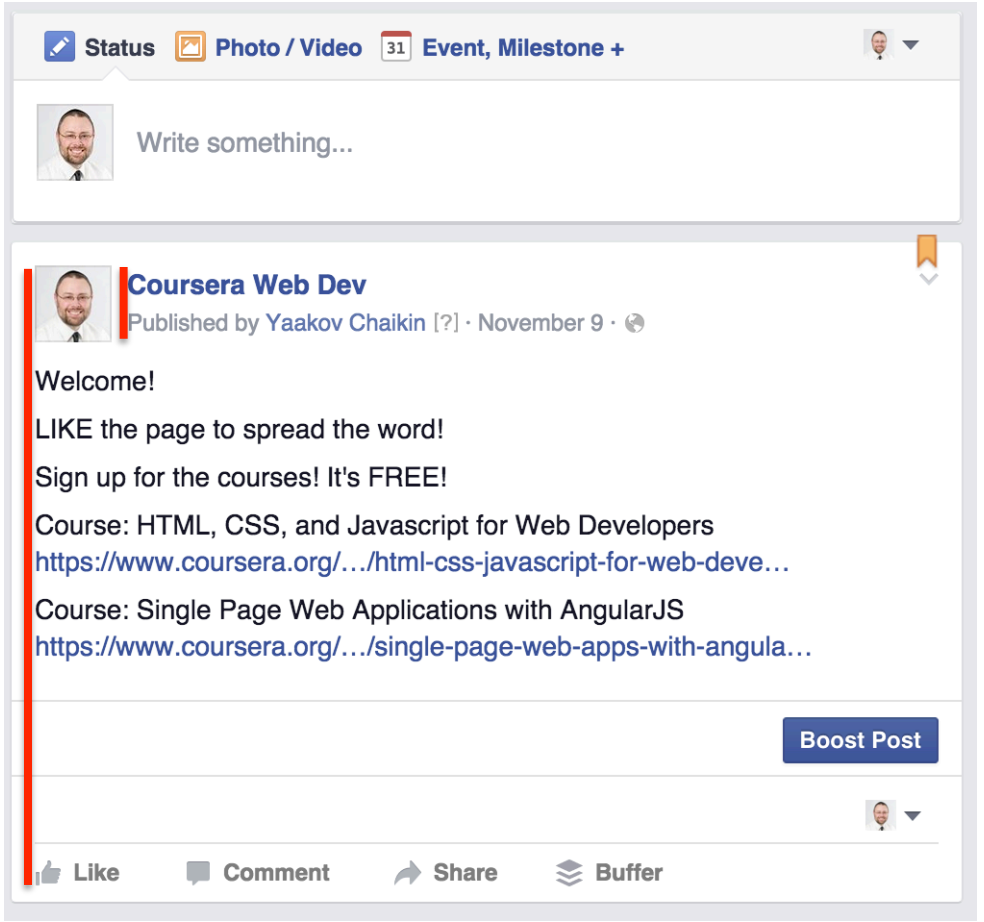


Why Negative Row Margin?



Like and Sign up

facebook.com/CourseraWebDev



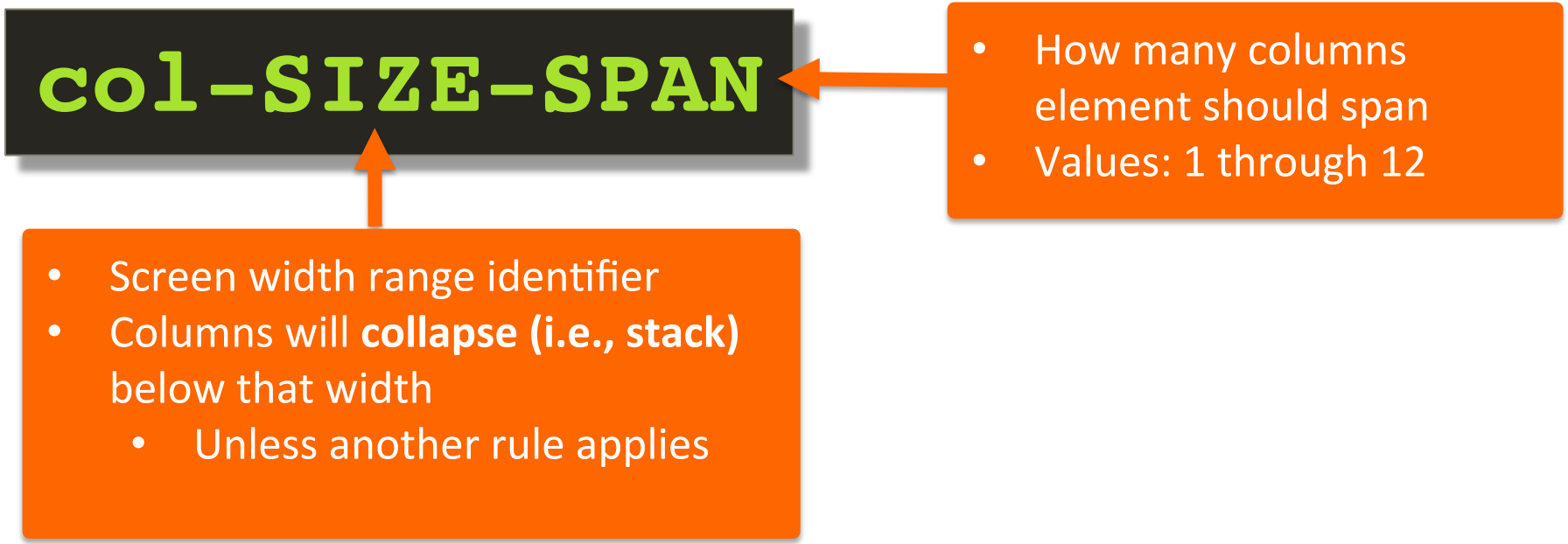
Bootstrap Grid System Basics

```
<div class="container">
  <div class="row">
    <div class="col-md-4">Col 1</div>
    ...
  </div>
</div>
```

Bootstrap Grid System Basics

```
<div class="container">
  <div class="row">
    <div class="col-md-4">Col 1</div>
    ...
  </div>
</div>
```

Bootstrap Grid System Basics



Bootstrap Grid System Basics

```
<header class="container">
  <nav class="row">
    <div class="col-md-4">Col 1</div>
    ...
  </nav>
</header>
```

Summary

- Structure Bootstrap expects for the grid-based layout
  - Needs to be inside .container (or .container-fluid)
  - All columns must be inside .row
- SIZE identifier identifies at which breakpoint specified column spans will be ignored and all elements will collapse (i.e., stack)
- If no other rules apply, specifying col-xs-… will keep that layout no matter what the size of the screen

Visit with the Client

In this lecture, we will discuss…

Visit with the Client

Visit with the Client

Most clients have NO IDEA what they want!

- ✧ **It's YOUR JOB to ask questions to figure it out**
- ✧ **Bring web site examples of similar businesses**

WELCOME!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod

MUST KNOW INFO HERE!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

THIS IS VERY IMPORTANT!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed

READ THIS! IT'S IMPORTANT

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed

REMINDER! IMPORTANT!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed

THIS IS VERY IMPORTANT!

Lorem ipsum dolor sit amet,

WELCOME!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do

MUST

Lorem i
consect
sed do e
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,

THIS IS VERY IMPORTANT!

Lorem ipsum dolor sit amet,

EVERYTHING IS IMPORTANT!

=

NOTHING IS IMPORTANT!

REMINDER! IMPORTANT!

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed

THIS IS VERY IMPORTANT!

Lorem ipsum dolor sit amet,

Visit with the Client

LESS (information) IS MORE!

- **Encourage your client not to cram information on the site**

Visit with the Client

Find a way for the client to INVEST in the project

- **Especially true if you're doing this for free just to build up your portfolio.**
  - **E.g., have them commit to pay for product photography**

Visit with the Client

Have client designate ONE PERSON responsible for decisions

Visit with the Client

Limit number of revisions UPFRONT

- **If it's a paying job, limit number of FREE revisions**

Visit with the Client

Google for
‘web development client questionnaire’

Visit with the Client

Involve others if needed

- Contact a local college and see if a graphic design student wants to join you by providing free design services
- Same for photography, if your site needs one

Visit with the Client

Get an idea of what the client has right now

Summary

- Bring examples of other sites to help client figure out what they want
- Encourage client to use less information
- Client should invest in the project
- Client should have one person as the decider
- Limit number of revisions
- Involve others to help you produce a great product

Field Trip!!!!

In this lecture, we will discuss…

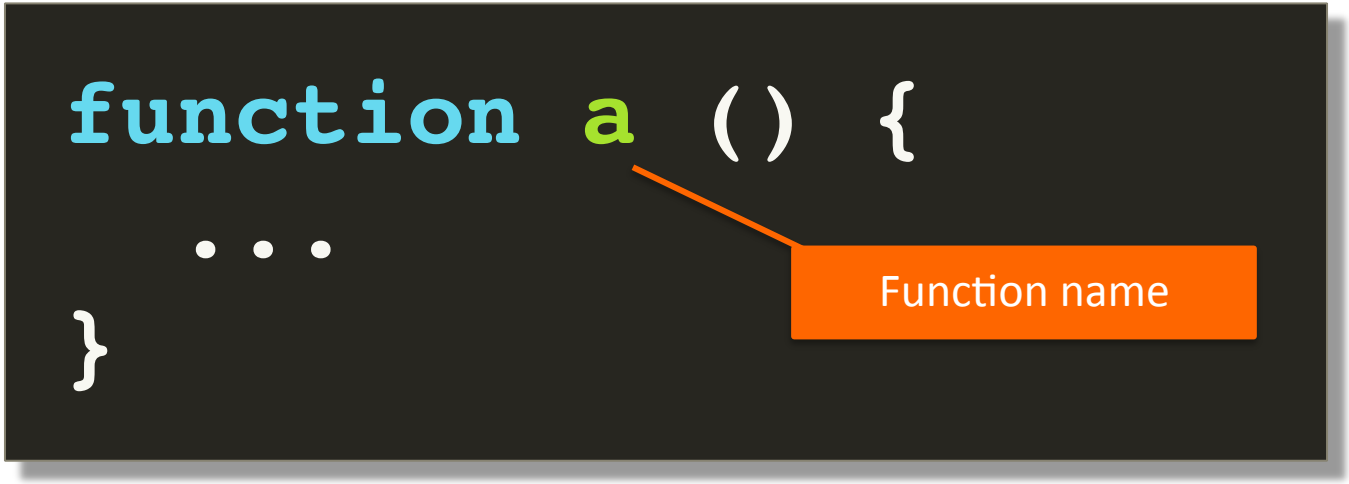
Defining Variables, Functions and Scope

Variables

```
var message = "hi";
```

- ✧ **Variable definition should always start with ‘var’**
- ✧ **No types are declared**
  - **JS is dynamically typed language**
  - **Same variable can hold different types during the life of the execution**

Functions



Functions

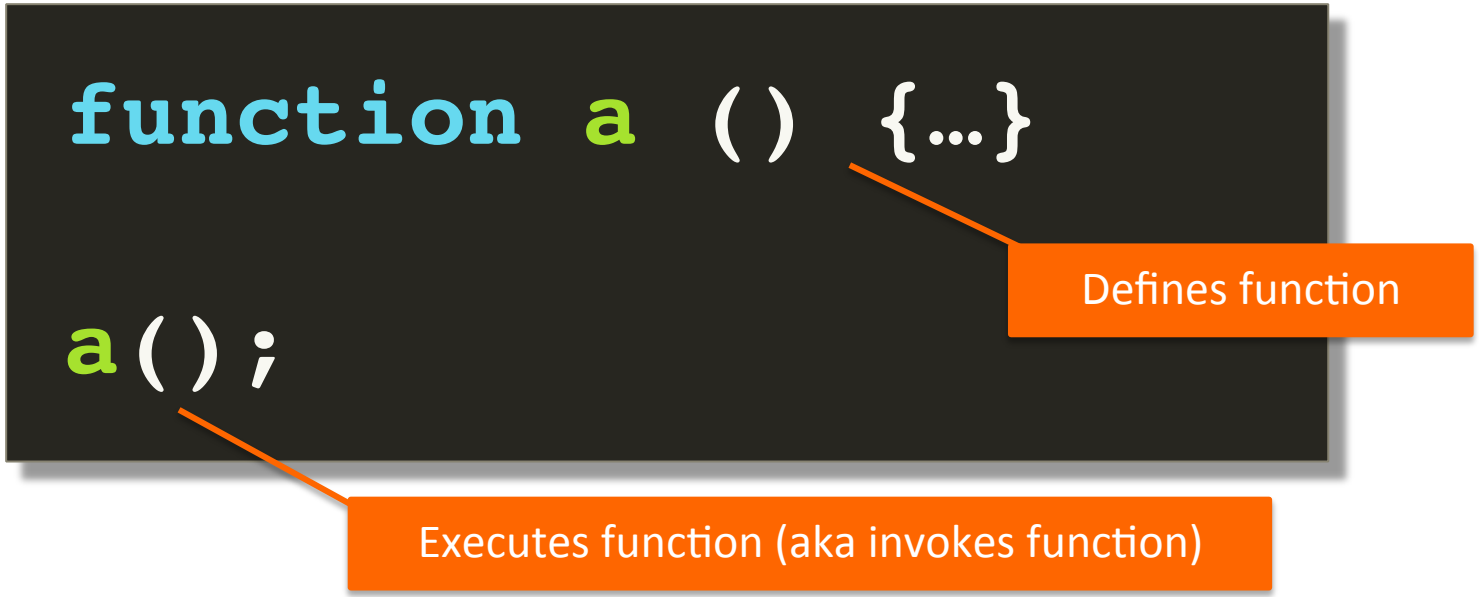
```
function a () {…}
```

```
var a = function () {…}
```

Value of function assigned, NOT the returned result!

No name defined

Functions


function a () {…}
a();
Defines function
Executes function (aka invokes function)

Functions

Arguments defined without ‘var’

```
function compare (x, y) {
  return x > y;
}
```

Functions

```
function compare (x, y) {…}
var a = compare(4, 5);
compare(4, "a");
compare();
```

ALL LEGAL

Scope

Global

- Variables and functions defined here are available everywhere

Function
aka lexical

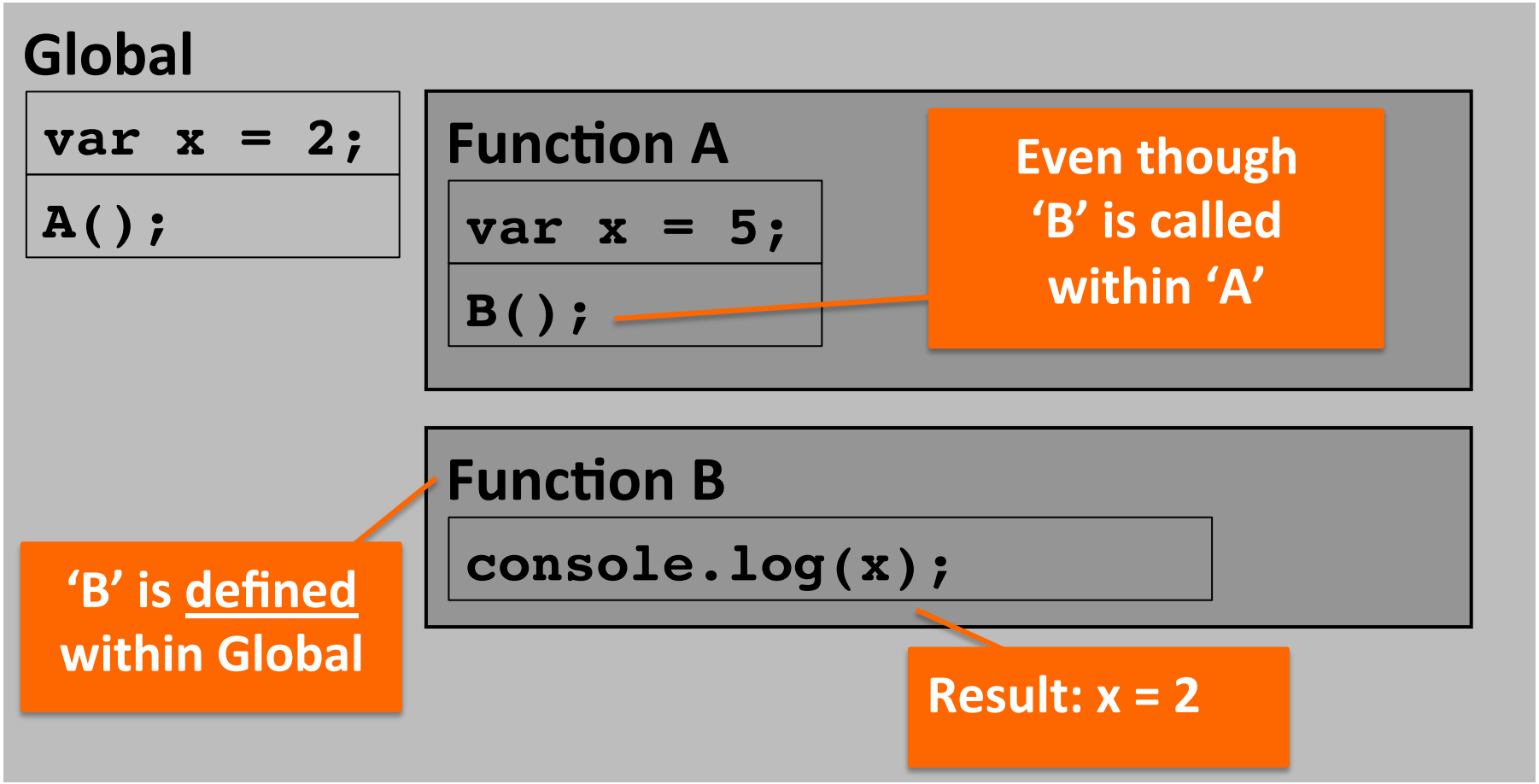
- Variables and functions defined here are available only within this function

Scope Chain

- Everything is executed in an *Execution Context*
- Function invocation creates a new *Execution Context*
- Each *Execution Context* has:
  - Its own *Variable Environment*
  - Special '`this`' object
  - Reference to its *Outer Environment*
- Global scope does not have an *Outer Environment* as it's the most outer there is

Scope Chain

Referenced (not defined) variable will be searched for in its current scope first. If not found, the Outer Reference will be searched.
If not found, the Outer Reference's Outer Reference will be searched, etc.
This will keep going until the Global scope.
If not found in Global scope, the variable is `undefined`.


Global
var x = 2;
A();
Function A
var x = 5;
B();
Even though 'B' is called within 'A'
Function B
console.log(x);
'B' is defined within Global
Result: x = 2

Summary

- Defining variables – dynamically typed
- Defining functions – creates its own scope (lexical)
- JS code runs within an Execution Context
- Scope chain is used to retrieve variables from Outer Variable Environments

In this lecture, we will discuss…

Javascript Types

Types

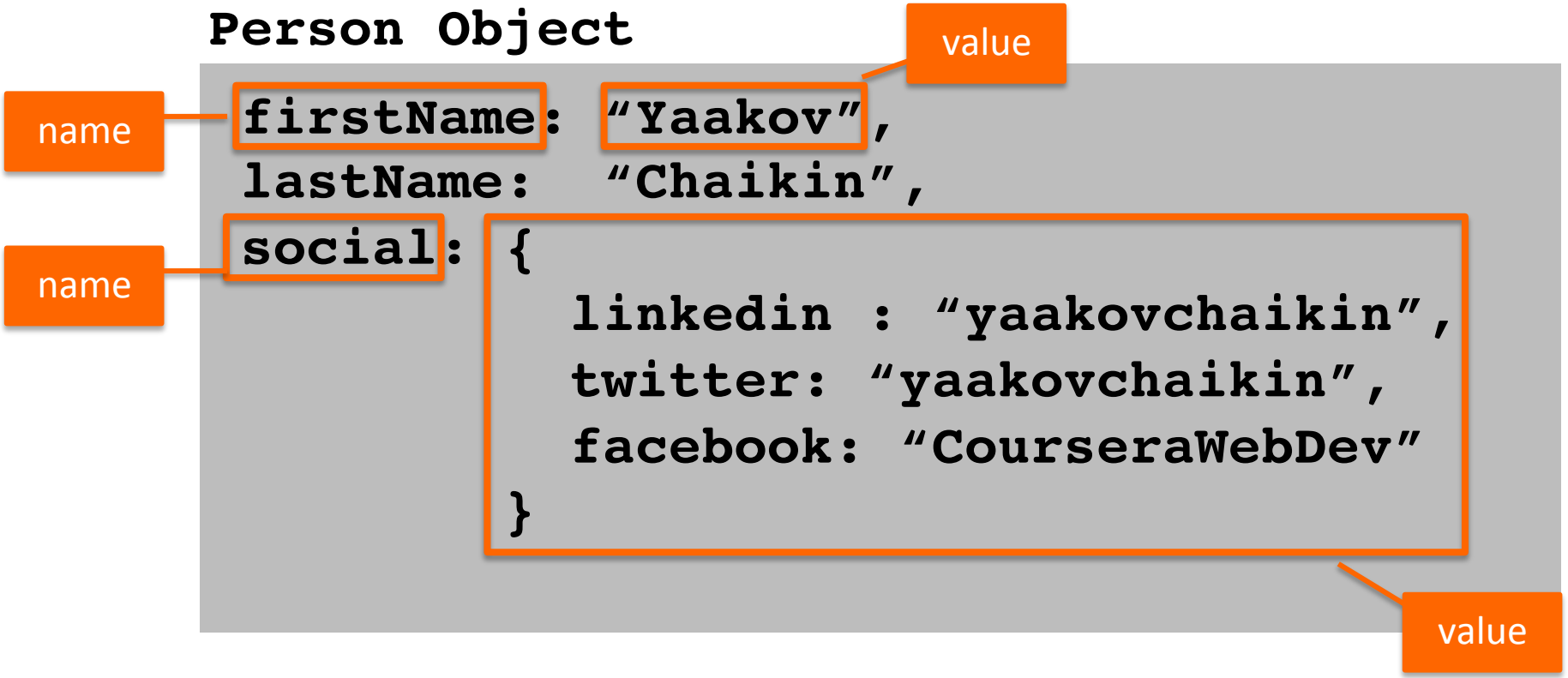
A type is a particular data structure.

- Each language defines some built-in types
- Built-in types can be used to build other data structures
- JS has 7 built-in types: 6 primitive and 1 Object type

Object Type

Object is a collection of name/value pairs

Object Type


Person Object
name
value
firstName: "Yaakov",
lastName: "Chaikin",
name
social: {
linkedin : "yaakovchaikin",
twitter: "yaakovchaikin",
facebook: "CourseraWebDev"
}
value

Primitive Types

Primitive type represents a *single, immutable* value

- **Single value, i.e., *not* an object**
- **Immutable means once it's set, it can't be changed**
  - **Value becomes read-only**
  - **You can create another value based on an existing one**

Primitive Type: Boolean

Boolean can only have 2 values: `true` or `false`

Primitive Type: Undefined

Undefined signifies that no value has ever been set

- Can only have one value: `undefined`
- You *can* set a variable to `undefined`, but you *should NEVER do it*
  - Its meaning is that it's never been defined, so defining it to `undefined` is counter to its core meaning

Primitive Type: Null

Null signifies lack of value

- As opposed to `undefined`, which is lack of definition
- Can only have one value: `null`
- It's ok to explicitly set a variable to `null`

Primitive Type: Number

Number is the only numeric type in Javascript

- **Always represented under the hood as double-precision 64-bit floating point**
- **JS does *not* have an integer type**
  - **Integers are a subset of doubles instead of a separate data type**

Primitive Type: String

String is sequence of characters used to represent text

- **Use either single or double quotes, i.e., ‘text’ or “text”**

Primitive Type: Symbol

Symbol is new to ES6

Not covered in this class

- **ES6 (released 2015) isn't widely supported or used yet**

Summary

- Javascript defines 7 built-in types
  - Object and 6 Primitives
- Object type is a collection of name/value pairs
- Primitive type can contain a single, immutable value
- Undefined means variable memory has been allocated but no value has ever been explicitly set yet

In this lecture, we will discuss…

Passing Variables by Value vs. by Reference

Passing (or Copying) By Value

Given `b=a`, passing/copying by value means changing copied value in `b` does not affect the value stored in `a` and vice versa

Passing (or Copying) By Reference

Given `b=a`, passing/copying by reference means changing copied value in `b` does affect the value stored in `a` and vice versa

In Javascript, primitives are passed by value, objects are passed by reference

- "Under the hood", everything is actually passed by value

primitives

```
var a = 7;
var b = a;
```

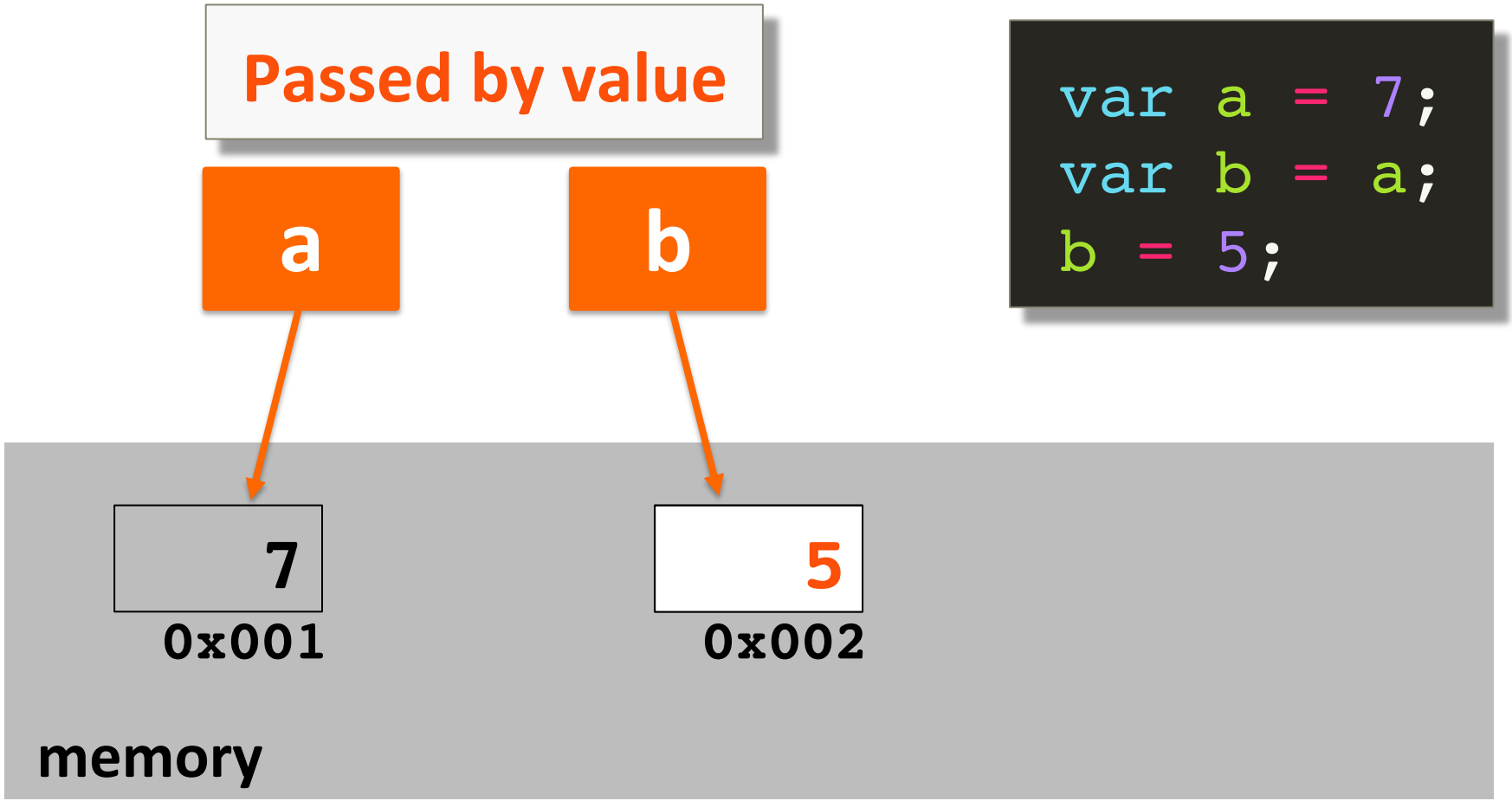
objects

```
var a = {x: 7};
var b = a;
```

- ‘b’ ends up with the same value as ‘a’

How does that work?

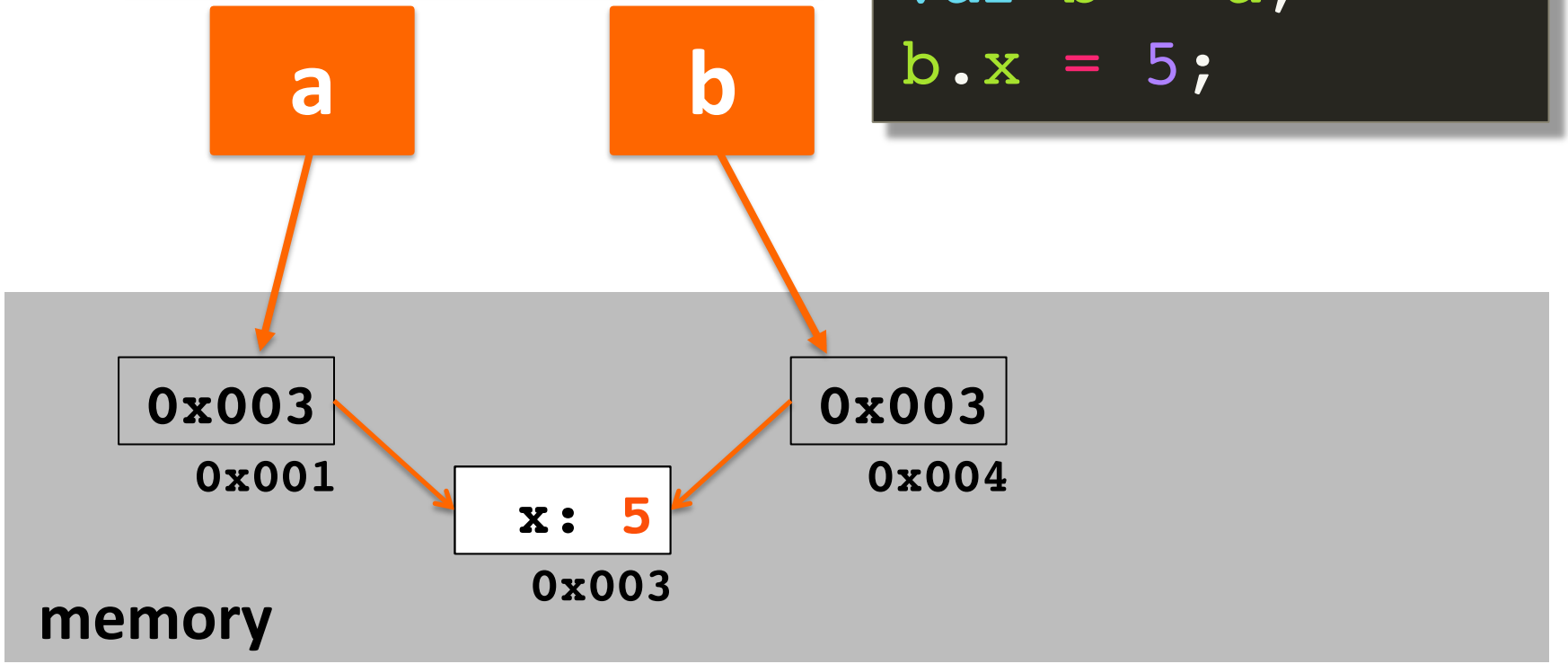
Passed by value




```
var a = 7;
var b = a;
b = 5;
```

Passed by reference

```
var a = {x: 7};
var b = a;
b.x = 5;
```



Summary

- Important! Passing by value vs. passing by reference
- Simple rule to remember
  - Primitives are passed/copied by value
  - Objects are passed/copied by reference

In this lecture, we will discuss…

HTTP Basics

What is HTTP?

HyperText Transfer Protocol

- **Based on request/response *stateless* protocol**
  - **Client opens connection to server**
  - **Client sends HTTP request for a resource**
  - **Server sends HTTP response to the client (w/resource)**
  - **Client closes connection to server**

Identifying Resources on the Web

URN: Uniform Resource Name

- Uniquely identifies resource or name of resource
- Does not tell us how to get the resource

example:

“HTML/CSS/Javascript/Web Developers/Yaakov/Chaikin”

Identifying Resources on the Web

URI: Uniform Resource Identifier

- Uniquely identifies resource or location of resource
- Does not necessarily tell us how to get the resource

example:

/official_web_site/index.html

Identifying Resources on the Web

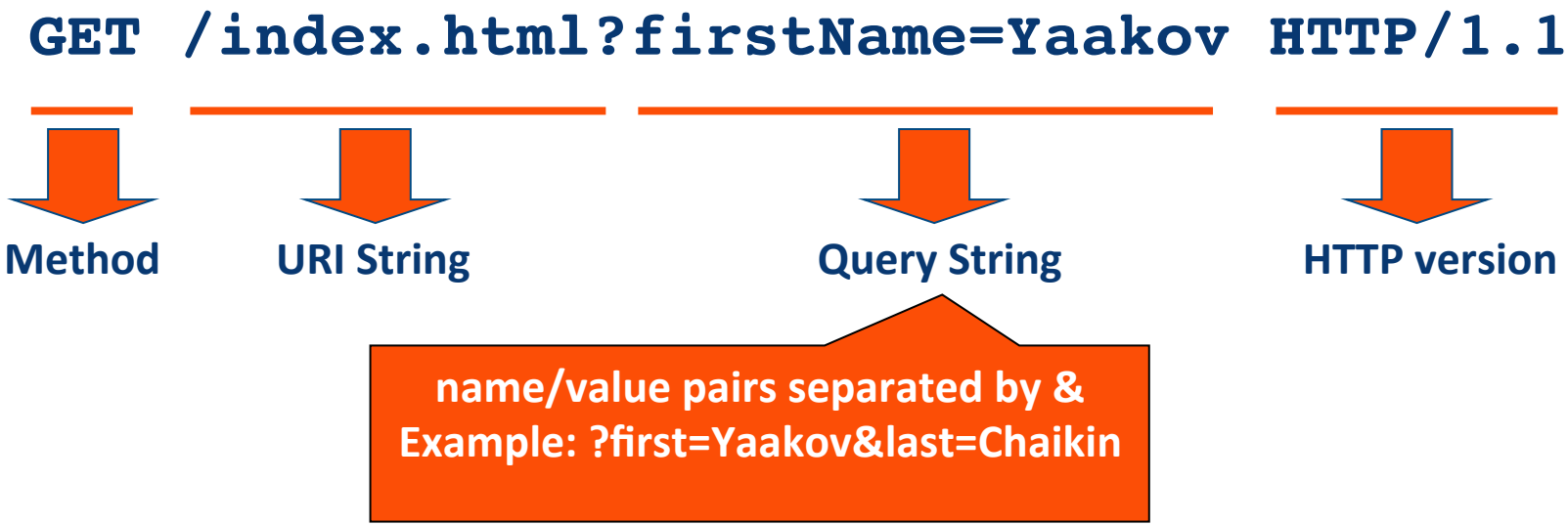
URL: Uniform Resource Locator

- **Form of URI that provides info on how to get resource**

example:

http://www.mysite.com/official_web_site/index.html

HTTP Request Structure (GET)



HTTP Methods

- **GET**
  - Retrieves the resource
  - Data is passed to server as part of the URI
    - I.e., query string
- **POST**
  - Sends data to server in order to be processed
  - Data is sent in the message body
- More methods not covered here

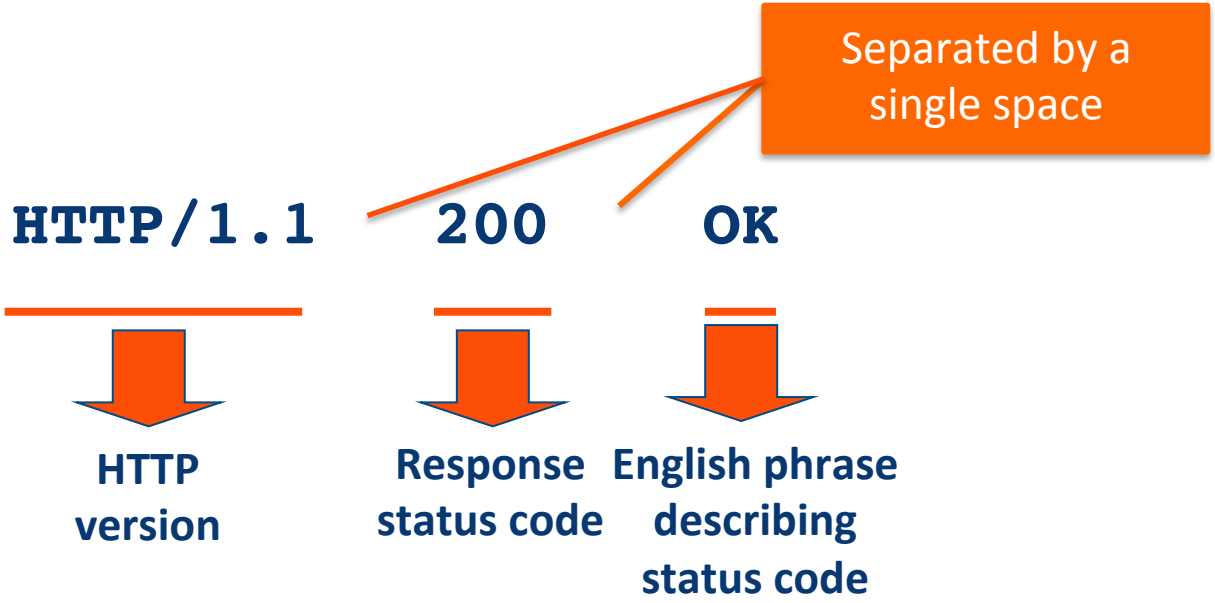
HTTP Request Structure (POST)

```
POST /index.html HTTP/1.1
Host: coursera.org
Accept-Charset: utf-8
firstName=Yaakov…
…
…
```

request headers

message body

HTTP Response Structure



Typical HTTP Response

```
HTTP/1.1 200 OK
Date: Tue, 11 Aug 2004 19:00:01 GMT
Content-Type: text/html
<html>
<body>
<h1>Secret to gaining weight REVEALED!</h1>
<p>Develop Coursera courses after work at night
while eating sweets to keep yourself awake!</p>
</body>
</html>
```

Some Response Status Codes

- **200 OK**
  - Ok, here is the content you requested
- **404 Not Found**
  - Server can’t find the resource requested
- **403 Forbidden**
  - Unauthenticated client tried to access a secure resource
- **500 Internal Server Error**
  - Some unhandled error was raised on the server

Summary

- Need to understand the basics of HTTP protocol
- GET method sends data to server as part of the URL
- POST method sends data as part of message body
- Response codes: 200, 404, 500, etc.

In this lecture, we will discuss…

Ajax Basics

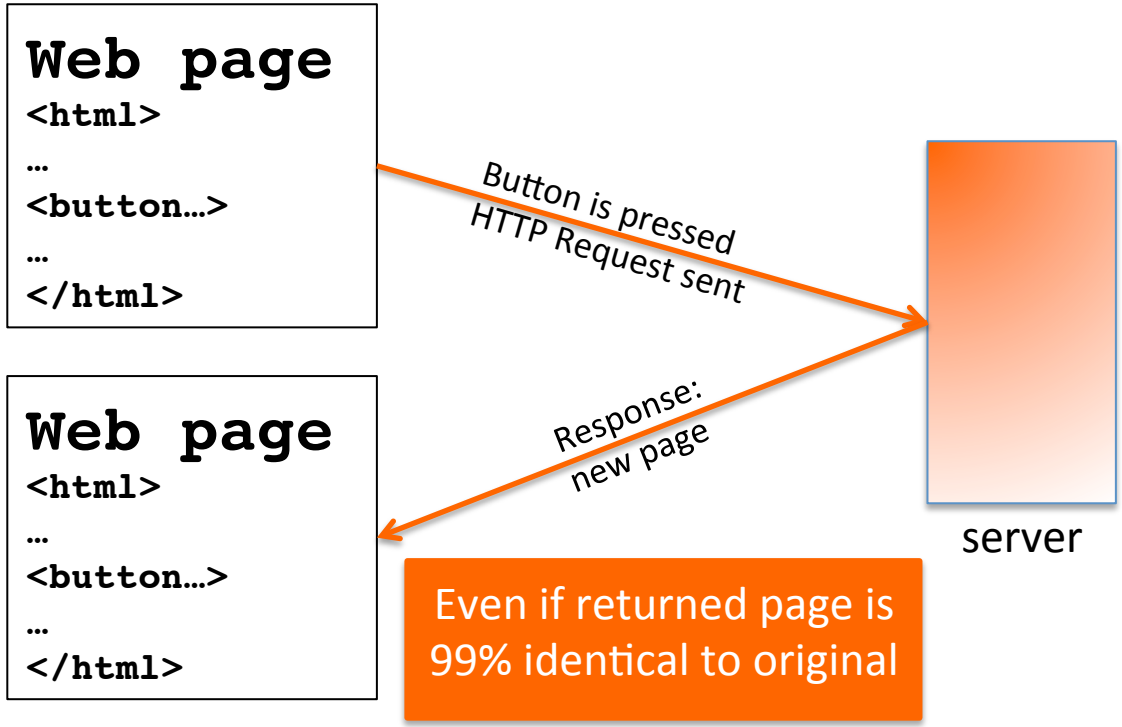
What is Ajax?

stands for

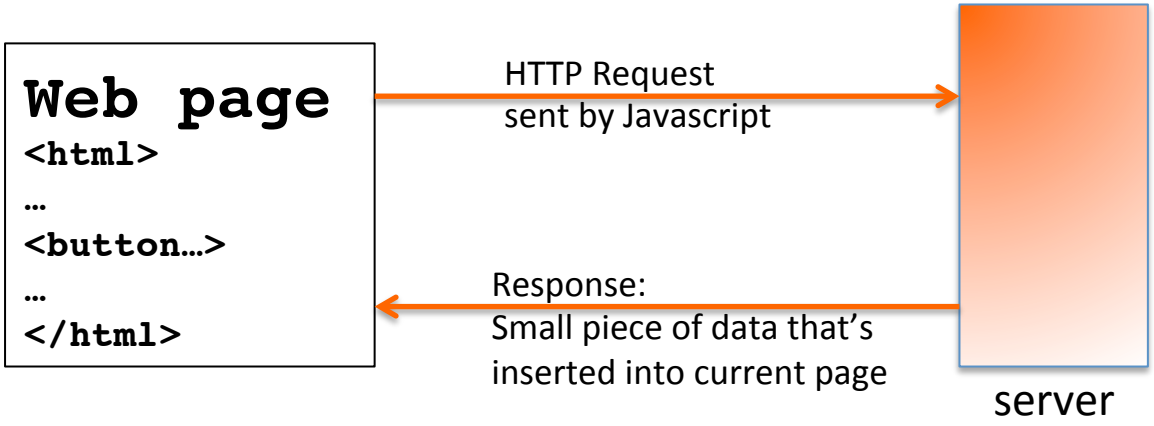
Asynchronous Javascript And XML

- **While Ajax started with XML, very few apps use it nowadays**
  - **Plain text (at times as html) and JSON is used instead**

Traditional Web App Flow



Ajax Web App Flow



- **Faster response**
  - **Less bandwidth, nicer experience for user**

Synchronous Execution

Execution of one instruction at a time

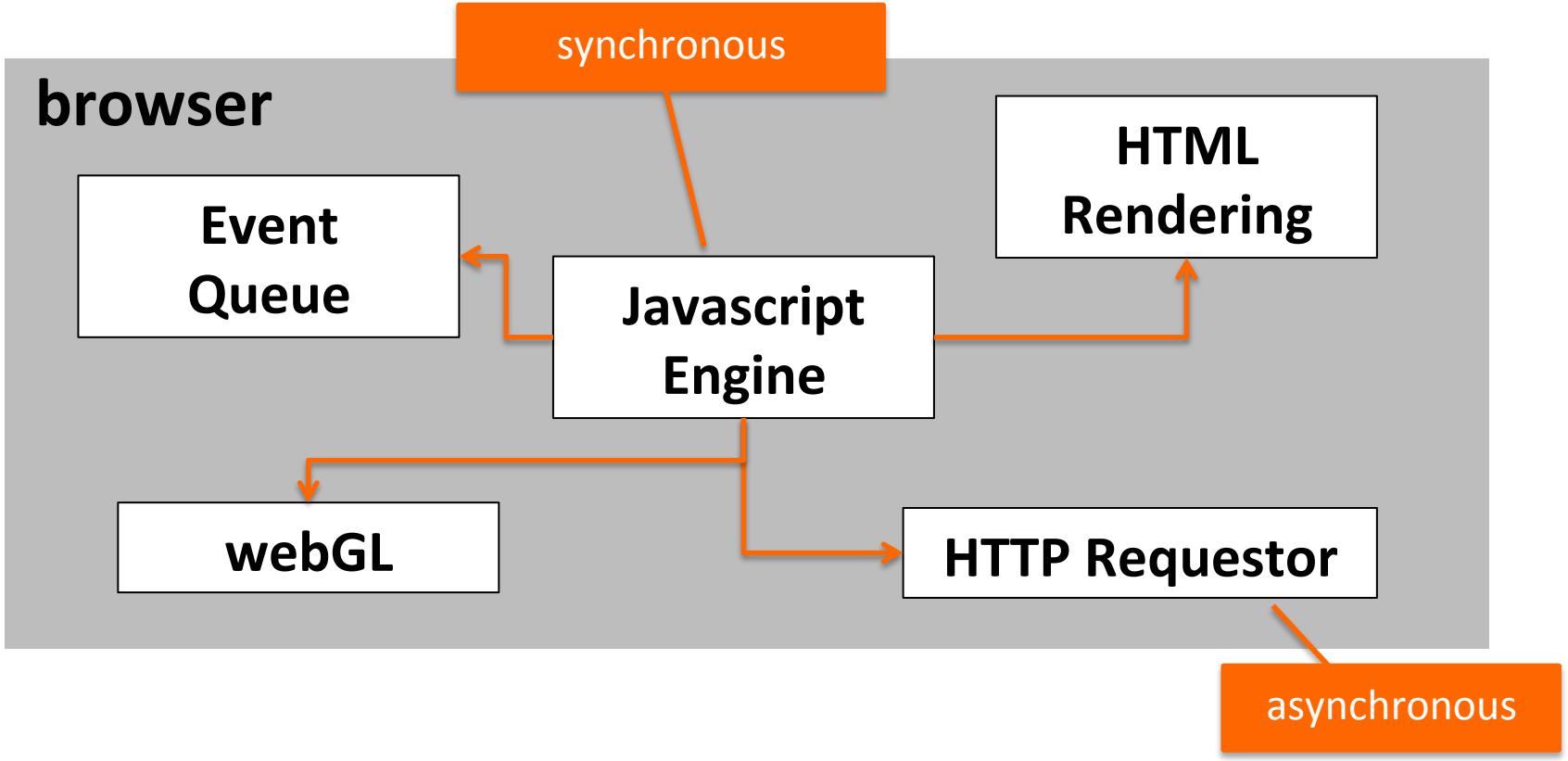
- **Can't start execution of another instruction until the first instruction finished its execution**

Asynchronous Execution

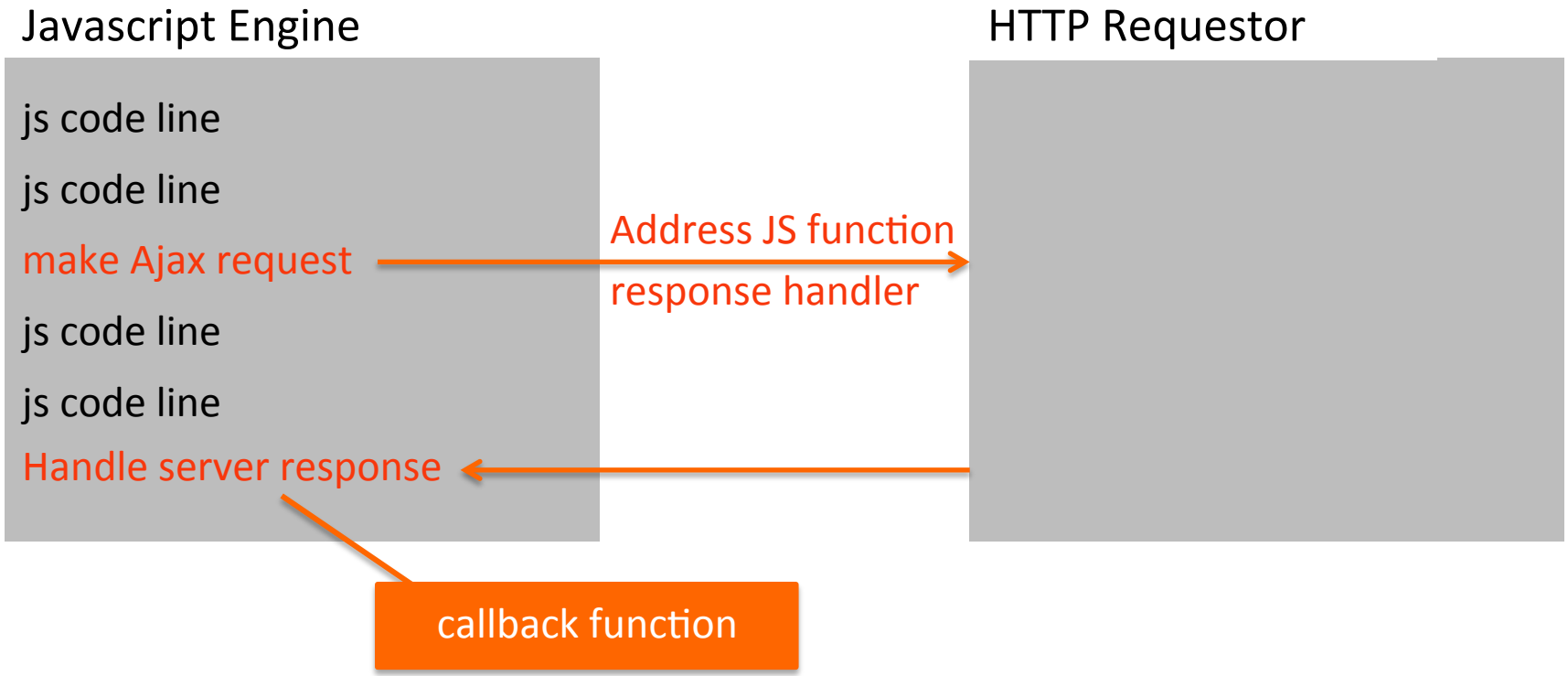
Execution of more than one instruction at a time

- Asynchronous instruction returns right away
- The actual execution is done in a separate thread or process

How Does Ajax Work Then?



Ajax Process



In this lecture, we will discuss…

Processing JSON

What is JSON?

stands for

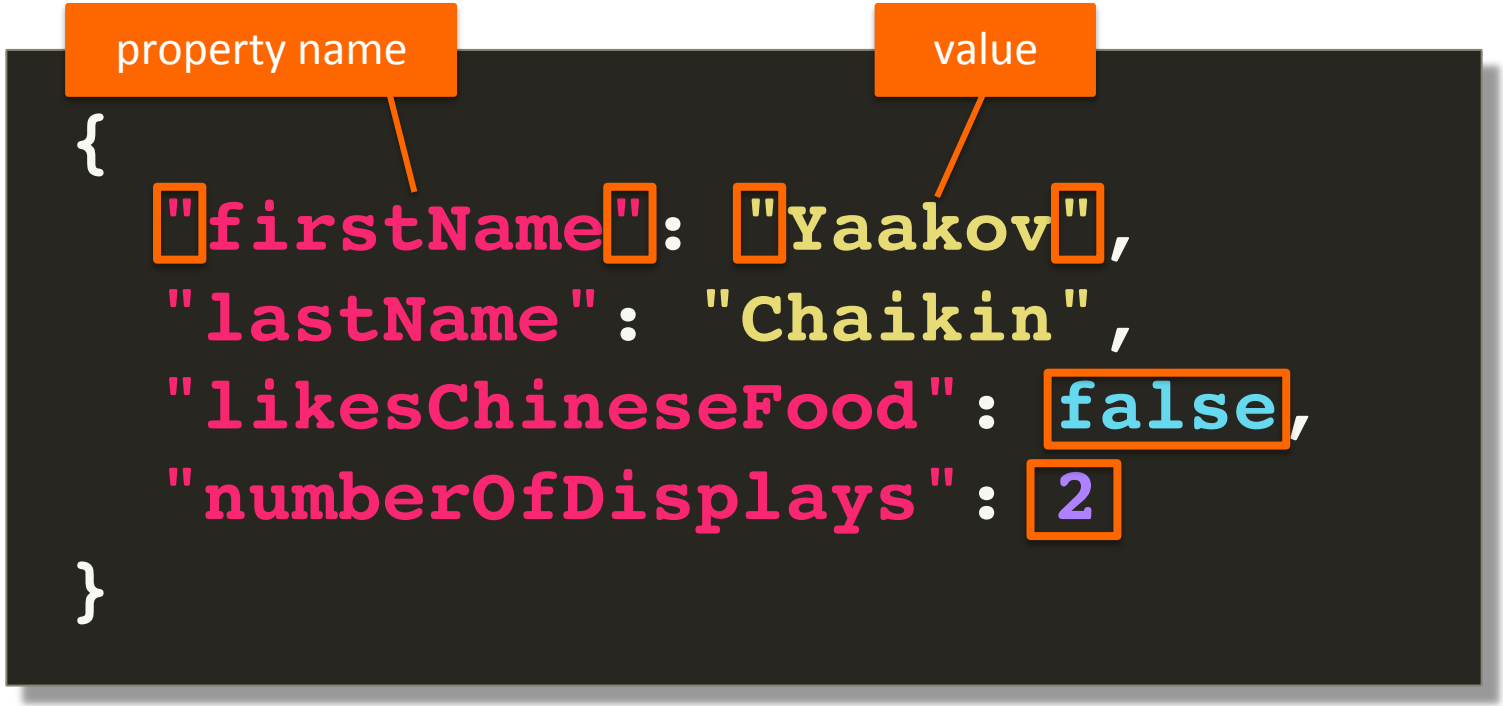
JavaScript Object Notation

- ✧ **Lightweight data-interchange format**
  - **Simple *textual* representation of data**
- ✧ ***Easy for humans* to read and write**
- ✧ ***Easy for machines* to parse and generate**
- ✧ **Completely independent of any language**

JSON Syntax Rules

- Subset of Javascript object literal syntax… but
  - Property names must be in *double* quotes
  - String values must be in *double* quotes
  - That's it!
- Syntax for everything else is *exactly* like for object literal

JSON Example



JSON Example

```
var jsonString =
'{
   "firstName": "Yaakov",
   "lastName": "Chaikin",
   "likesChineseFood": false,
   "numberOfDisplays": 2
 }';
```

Common Misconception

- **JSON is NOT a Javascript Object Literal**
- **JSON is just a string**
- **The syntax of JSON is based on object literal though**
- **Need to covert JSON into a JS object**

Converting JSON To String & Back to JSON

converts from json string to object

```
var obj = JSON.parse(jsonString);
```

converts from object to json string

```
var str = JSON.stringify(obj);
```

Summary

- JSON is a lightweight data representation
- Great format for passing data from server to client & back
- Syntax is based on Javascript object literal
  - But JSON is NOT JS object literal
- JSON.parse to convert from JSON string to object
- JSON.stringify to convert from object to JSON string