Internet Applications Design and Implementation

(Lecture 8: User-Centric Development)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Outline

- User-centric development vs Data-centric development
- Web: the underlying technology: HTML5 (the browser as a virtual machine)
- Client-side frameworks
- Client-side programming languages
- Front-end libraries and tools
- React and Redux

Internet Applications Design and Implementation

(Lecture 8- Part 1: User Centric Development)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

User-centric vs Data-centric development

- The decoupling between client applications and a data-based service architecture promotes that:
  - in user-centric development:
    - the main focus is on user interactions
    - the design method works around user stories, whose combination provides a full-fledged application.
  - in data-centric development:
    - the main focus is on the representation of state of a systems' information.
    - Interfaces and operations are based on data properties and state changes.

…It's much easier mentally to set up a context around "I am building an API for a database" and "I am building a front-end for users"… Jeff Escalante, carrot

Spectrum of client applications

- **website / static HTML pages**
  - no dynamically loaded data, poor behaviour expression, efficient response times (w/ caching), dynamic websites can be expanded for more efficiency.
- **website / dynamic server rendered HTML pages**
  - poor loading times, poor development features (distance between code and result). Poor modularity, high usage of templating languages.
- **website / ajax**
  - better loading times, development closer to the result (no templating needed). Better modularity and testing conditions.
- **web applications (SPA)**
  - app context, service based back-ends, richer behaviour and fluid UI
- **progressive web applications (PWA) - Local-first software**
  - device agnostic, and yet, closer to native client applications (storage, resources, responsive UI, notifications, etc.)
- **mobile applications**
  - more device resources, many times PWA in a shell, or produced with a platform agnostic framework (e.g. flutter)
- **other models** (Semantic web, Query-based — data science, AI, etc.)

THE NETFLIX TECH BLOG

Optimize Page Load

Netflix TechBlog
Learn about Netflix's world class engineering efforts, company culture, product developments and...

Follow

1.3K

The previous version of the home page rendered **all** rows of titles as its highest priority. This included fetching data from the server, creating all DOM nodes, and loading images. Our goal was to enable fast vertical scrolling through 30+ rows of titles.

For the new experience, we needed our page to load faster, minimize memory overhead, and allow for smooth playback. We knew these performance optimizations would come with a tradeoff against existing member behavior. To understand these tradeoffs, we began by measuring the existing home page load.

When the home page loads, we first render the billboard image and the top three rows on the server. Once this page has been delivered to the client, we make a call for the rest of the homepage, render the rest of the rows, and load all the images.

mic websites can be

, high usage of templating

l testing conditions.

fications, etc.)

Optimize Page Load

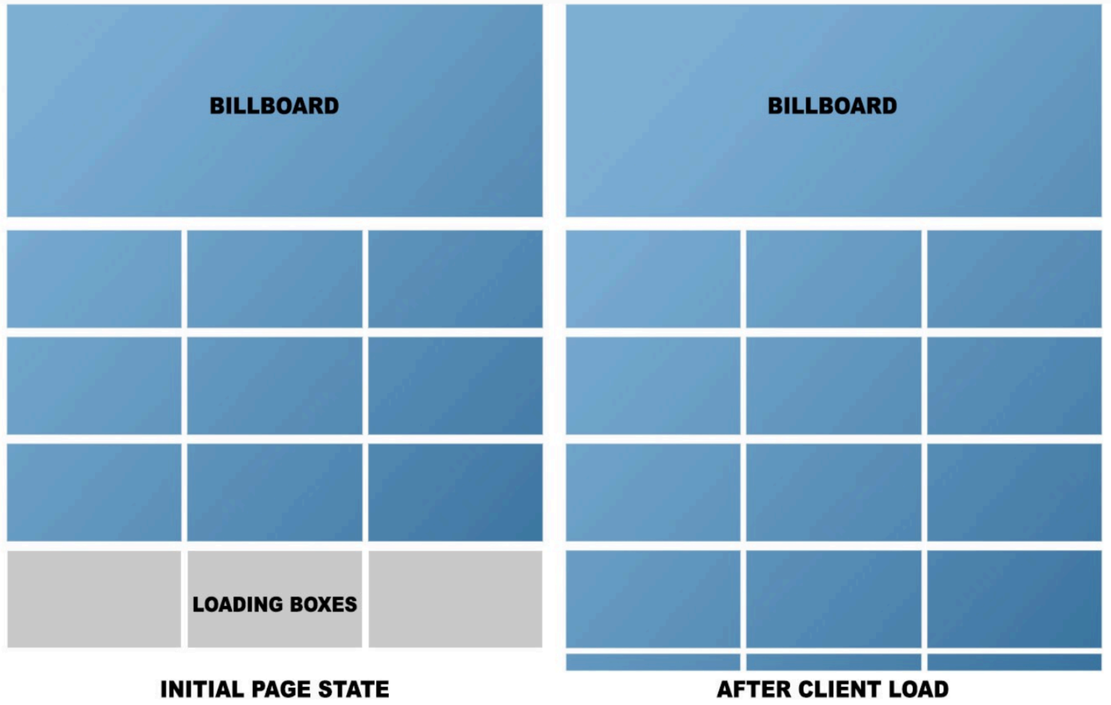
The previous version of the hom
highest priority. This included fe
DOM nodes, and loading image
scrolling through 30+ rows of ti

For the new experience, we nee
memory overhead, and allow fo
performance optimizations wou
member behavior. To understan
the existing home page load.

When the home page loads, we
three rows on the server. Once t
make a call for the rest of the ho
load all the images.

M | THE NETFLIX TECH BLOG

Netflix TechBlog
Learn about Netflix's world class engineering efforts, company culture, product developments and...
Follow
1.3K



Page load from the server vs. page load after auto-loading all rows

Here's what the page load looks like from a data perspective.

	Initial Page State	Full Page Load
DOM Nodes	650	11,744
Memory	8MB	45MB
# Images	46	298
Images (Total Size)	1.2MB	4.4MB

Client (web / mobile) Applications

- Running on a Browser
  - Cross-platform compatibility (HTML, CSS, Javascript)
  - Slow interpreted code
  - Limited capabilities (no camera, no gyro, no resources, sandboxed)

- Native code (maybe except Flutter generic code)
  - All device capabilities available
  - Fast and efficient applications
  - Proprietary languages and APIs (Java/Android ADT, Swift, C#/C++ UWP)

- Running on a shell (PhoneGap/Cordova, ReactNative, Electron, etc.)
  - Cross-platform compatibility (HTML, CSS, Javascript)
  - All device capabilities available
  - Precompiled and packaged code (fast)
  - Look and feel is almost right on latest approaches

Internet Applications Design and Implementation

(Lecture 8 - Part 2: HTML5)

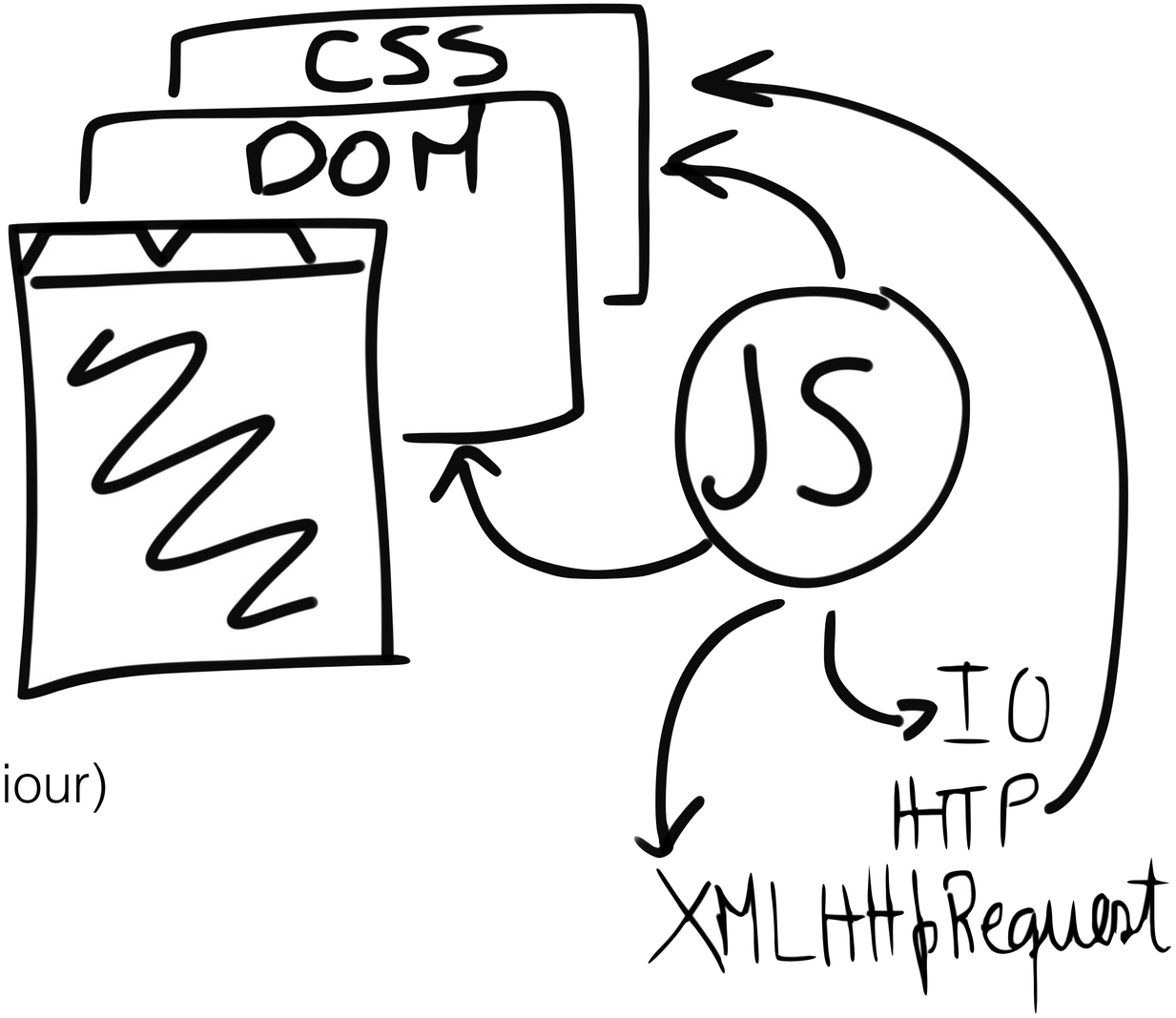
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

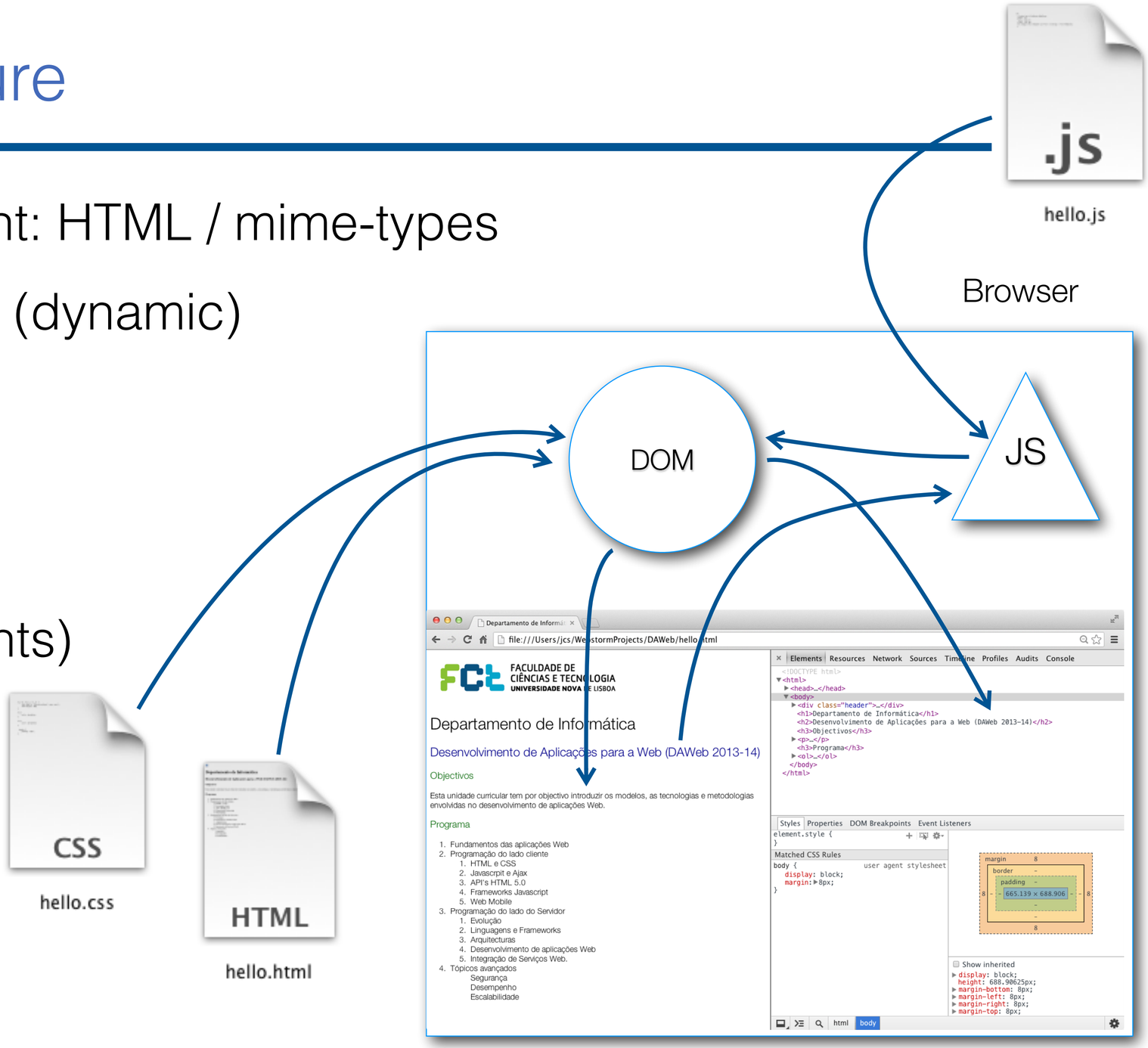
Web client architecture - Virtual Machine

- Browser (HTML5)
  - HTML (structure and semantics)
  - CSS (style and UI behaviour)
  - JS + AJAX,
    Socket interfaces (behaviour)
  - DOM
    (the supporting data structure)
  - UI Events & callbacks
    (mechanism for dynamic structuring of behaviour)



Browser logic architecture

- Source files with static content: HTML / mime-types
- DOM abstract representation (dynamic)
  - Elements
  - Style annotations
  - Event handlers
- User-interface (visible elements)
- JavaScript behaviour



What's HTML5

- Adds meaningful semantic markup
- Separates design from content
- Promotes accessibility and design responsiveness
- Reduces overlap between HTML, CSS, and JavaScript
- Supports rich media contents without the need for plugins (Flash, Java)

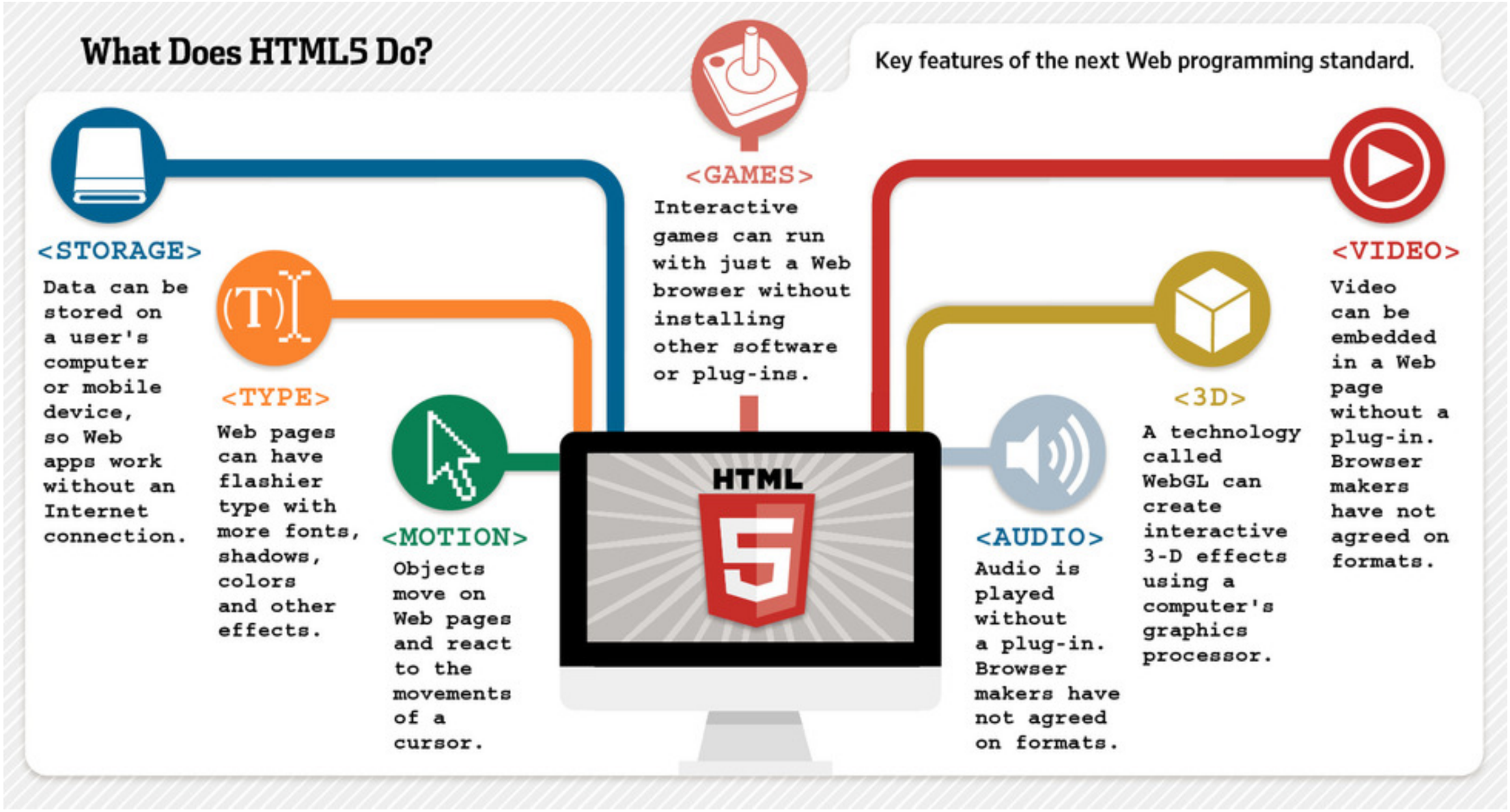
https://html.com/html5/

HTML5 comes as a package

- HTML5 is divided into:
  - HTML code to define structure and basic content to web pages
    - new tags like `<video>` `<audio>` `<svg>` `<header>`
  - CSS (Cascading Style Sheets) to define the appearance, layout, and some behaviour
  - JavaScript defines behaviour associated to the webpage elements (including dynamic construction of pages)
- The supporting structure for all these elements is the DOM

What's HTML5 - in more detail

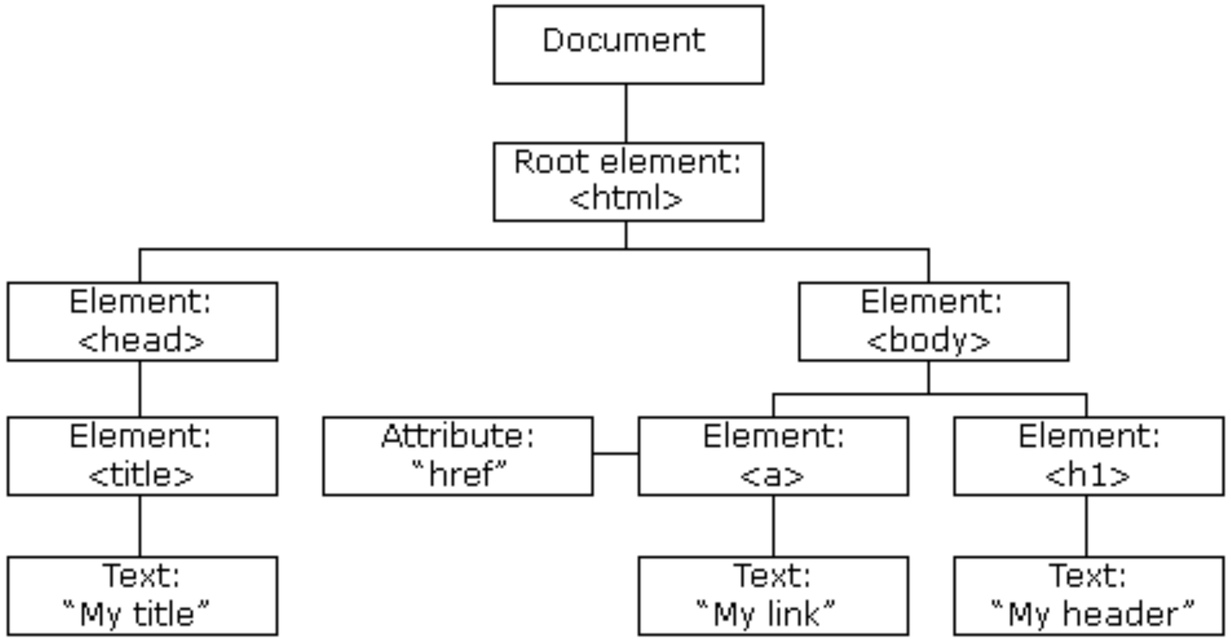
- Deprecated elements like **`center`**, **`font`**, and **`strike`** have been dropped
- Improved parsing rules allow for more flexible parsing and compatibility
- New elements including **`video`**, **`time`**, **`nav`**, **`section`**, **`progress`**, **`meter`**, **`aside`** and **`canvas`**
- New input attributes including email, URL, dates and times
- New attributes including charset, async and ping
- New APIs that offer offline caching, drag-and-drop support and more
- Support for vector graphics (SVG) without the aid of programs like Silverlight, Flash, or Java
- Support for MathML to allow better display of mathematical notations
- JavaScript can now run in the background thanks to the JS Web worker API
- Global attributes such as tabindex, repeat and id can be can be applied for all elements

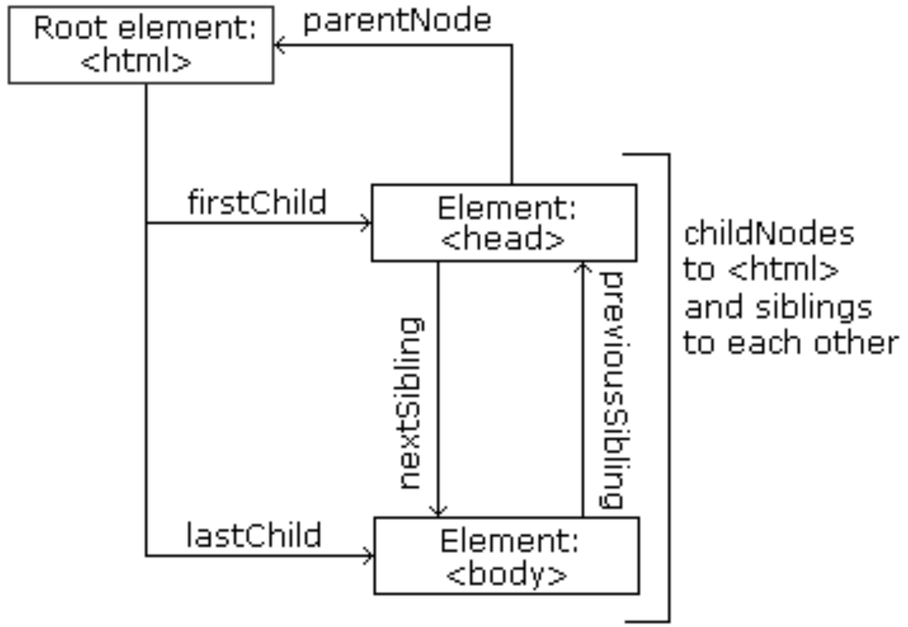


https://www.keycdn.com/blog/html-vs-html5

DOM - Document Object Model

- Standard (W3C) specification of the API of the data structure representing a structured document (HTML, XML, etc.).
- Defines objects and properties for all elements of a page and methods to access and modify them.
- Allows the dynamic retrieval, constructions, and modification of HTML elements in a web page.





http://www.w3schools.com/htmldom

DOM - Document Object Model

- DOM is a convention to represent HTML (XML) documents in a tree of objects.
- It's the **dynamic** supporting structure of a web page.
- Each node of the DOM has a particular structure (attributes) and associated behaviour (methods).
- DOM objects can be created, accessed, and modified using:
  - HTML: the static structure of DOM objects
  - CSS: using (rich) object selectors
  - JavaScript: by traversing the DOM tree using the DOM API.

HTML5 is an assembly-like technology

- The underlying technology is HTML, CSS, JS
- Developing tools should ensure:
  - A higher level of abstraction
  - Tools to develop faster, safer, and more secure

- A simple and robust base to build an UI
- Electron and ReactNative are building a (portable) bridge to native platforms.

HTML

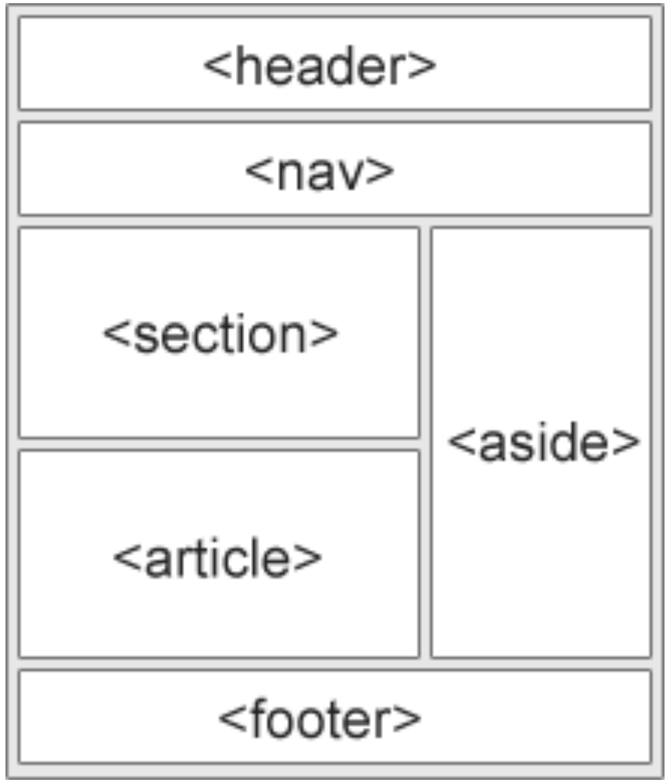
HTML

- Each HTML element has a tag
  - **<body>   ** …
- Content
  - **This is a paragraph**
- and a set of attributes:
  - Generic: set on any HTML element. (ex: **id**, **class**, **hidden**, …)
  - Particular: img/**src**, option/**value**, etc.
  - Events: dependent on the kind of element
    body / **onload**, button / **onclick**, etc.

HTML essentials in one slide

- <!DOCTYPE html>
- <html></html> root element
- <head></head> section to declare meta-information, stylesheets and scripts
- <body></body> section to define content
- <p> <h1>… block elements
-   <i> inline elements
- < > é escaped characters

HTML **5** semantic elements



HTML **5** semantic elements (part of)

Tag	Description
<article>	Defines an article in the document
<figcaption>	Defines a caption for a <figure> element
<figure>	Defines self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for the document or a section
<header>	Defines a header for the document or a section
<main>	Defines the main content of a document
<menuitem>	Defines a command/menu item that the user can invoke from a popup menu
<nav>	Defines navigation links in the document
<section>	Defines a section in the document

- What is structural?
  - …
- What is style?
  - …
- What is behaviour?
  - …

General structure - <html>


```
<!DOCTYPE html>
<html>
 <head>
     ...
 </head>

 <body>
     ...
 </body>
</html>
```

Invisible info - <head>

```html
<head>
    <!-- head defines invisible information about the page -->
    <!-- Meta information (machine parseable) can also be defined -->
    <meta charset="UTF-8">
    <meta name="description" content="My Home Library Web Site">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="João Costa Seco">
    <meta http-equiv="refresh" content="30">

    <!-- The title of the browser window (mandatory) -->
    <title>My home library</title>


    <!-- To define the base URL for all relative links -->
    <base href="ctp.di.fct.unl.pt/~jcs/daweb14-15">

    <!-- To define the style of the elements of the page -->
    <style>
        * { font-family: sans-serif;}
    </style>
    <!-- Or import an external file with the stylesheet -->
    <link href="screen.css" type="text/css" rel="stylesheet" media="screen"/>

    <!-- Custom behaviour can be defined using Javascript -->
    <!-- Directly in the HTML code -->
    <script>
        function Hello() { alert("Hello World."); }
    </script>
    <!-- Or by importing a script file -->
    <script src="books.js"></script>
</head>
```

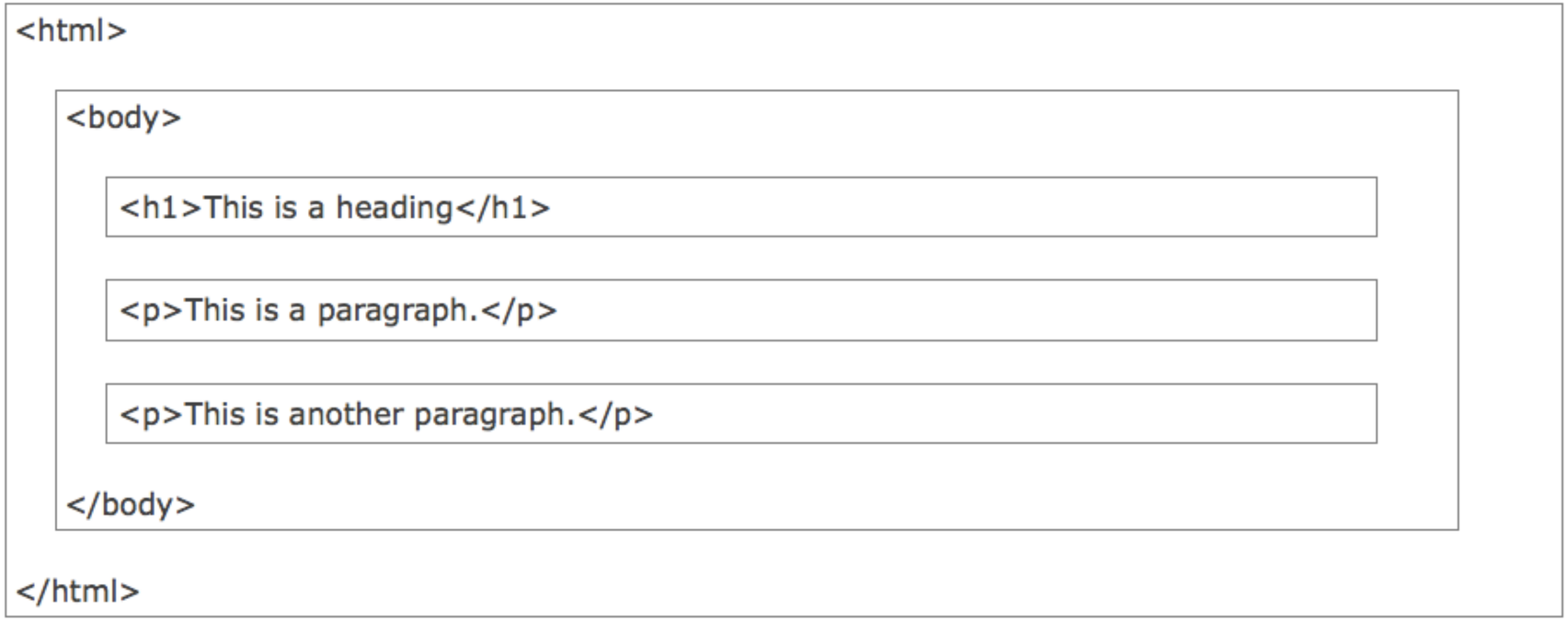
Visible info - <body> - **old style**

```
<div id="header">
  <h1>Monday Times</h1>
</div>

<div id="menu">
  <ul>
    <li>News</li>
    <li>Sports</li>
    <li>Weather</li>
  </ul>
</div>

<div id="content">
  <h2>News Section</h2>
  <div class="article">
    <h2>News Article</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in
porta lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
  </div>
  <div class="article">
    <h2>News Article</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in
porta lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
  </div>
</div>

<div id="footer">
  <p>&amp;amp;copy; 2016 Monday Times. All rights reserved.</p>
</div>
```

Visible info - <body> - **new style**

```
<header>
<h1>Monday Times</h1>
</header>

<nav>
<ul>
<li>News</li>
<li>Sports</li>
<li>Weather</li>
</ul>
</nav>

<section>
<h2>News Section</h2>
<article>
<h2>News Article</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in porta
lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
</article>
<article>
<h2>News Article</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in porta
lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
</article>
</section>

<footer>
<p>&copy; 2014 Monday Times. All rights reserved.</p>
</footer>
```

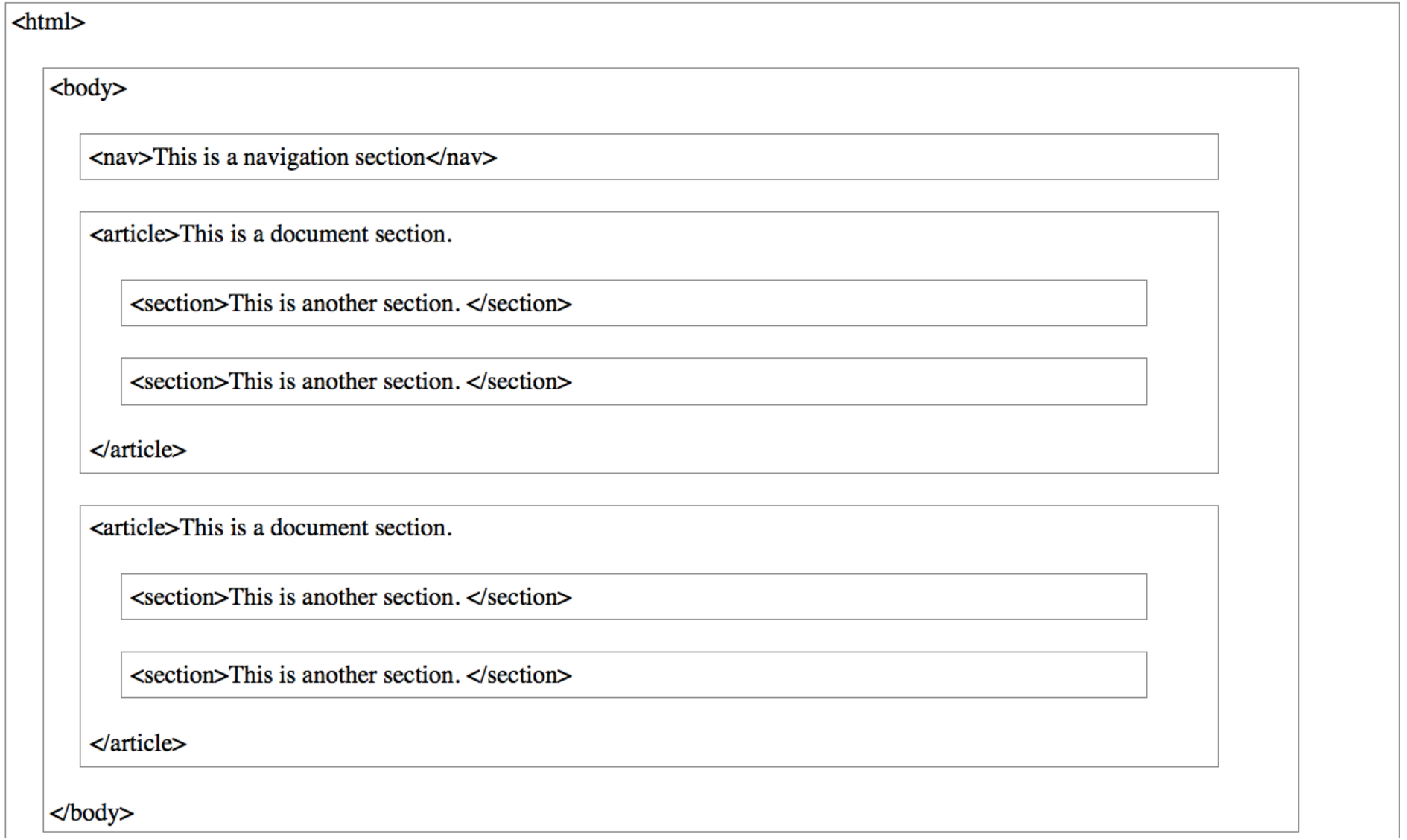
Textual block elements



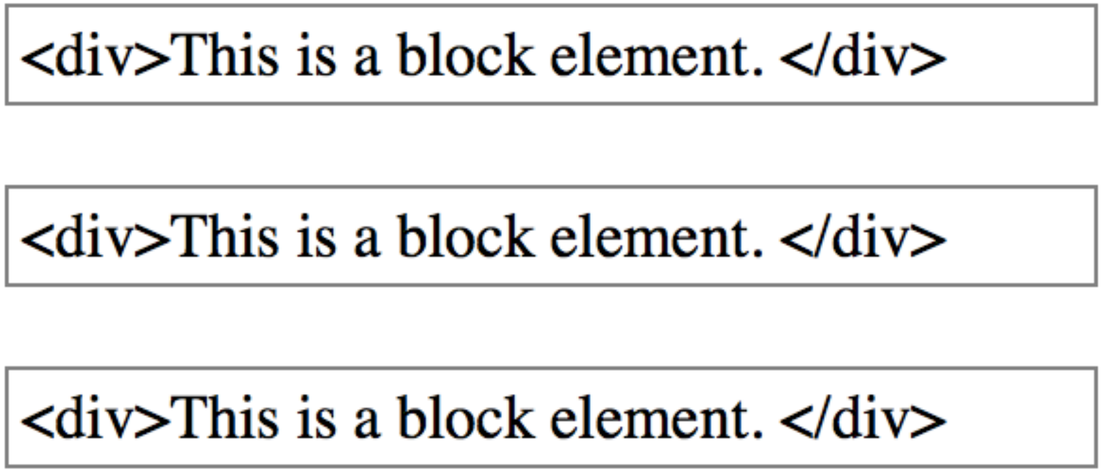
Textual inline elements

```
<html>

    <body>

        <h1>This is a heading</h1>

        <p>This is a paragraph. <span>This is a span</span> <b>This is a strong tag</b>
        <a>This is an anchor</a> </p>

        <p>This is another paragraph.</p>

    </body>

</html>
```

Structural block elements



Block elements



Inline elements

```
<span>This is </span> <span> an inline </span> <span> element.
</span> <span>This is </span> <span> an inline </span> <span>
element. </span> <span>This is </span> <span> an inline </span>
<span> element. </span>
```

Attributes

- HTML elements can be modified through attributes

```
<a href="http://www.w3schools.com">This is a link</a>
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
<div style="border:1px">This is a section.</div>
```

HTML <button> Tag

« Previous | Complete HTML Reference | Next »

Example

A clickable button is marked up as follows:

```
<button type="button">Click Me!</button>
```

Try it Yourself »

Attributes

= New in HTML5.

Attribute	Value	Description
autofocus	autofocus	Specifies that a button should automatically get focus when the page loads
disabled	disabled	Specifies that a button should be disabled
form	*form_id*	Specifies one or more forms the button belongs to

HTML - Basic Interaction

- Links
  - Produce a GET request to the given URL and replace the entire content of the DOM
- Forms
  - Pre-defined controls (buttons)
  - Produce a request and expects a response document (HTML)
  - replaces the entire content of the DOM (depends on _target)
  - method attribute defines the request type (GET/POST)
  - called URL format depends on the used method (body/query string)

HTML - FORMS

```
<form action="/my-handling-form-page" method="post">
    <div>
        <label for="name">Name:</label>
        <input type="text" id="name" />
    </div>
    <div>
        <label for="mail">E-mail:</label>
        <input type="email" id="mail" />
    </div>
    <div>
        <label for="msg">Message:</label>
        <textarea id="msg"></textarea>
    </div>

    <div class="button">
        <button type="submit">Send your message</button>
    </div>
</form>
```

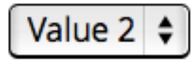
HTML

- Inputs

```
<input type="text" name="input" value="Type here">
```



```
<button name="button">Click me</button>
```




```
<select name="select">
  <option value="value1">Value 1</option>
  <option value="value2" selected>Value 2</option>
  <option value="value3">Value 3</option>
</select>
```

- Other interface elements (output)

```
<p>Heat the oven to <meter min="200" max="500" value="350">350 degrees</meter>.</p>
```

```
<progress value="70" max="100">70 %</progress>
```

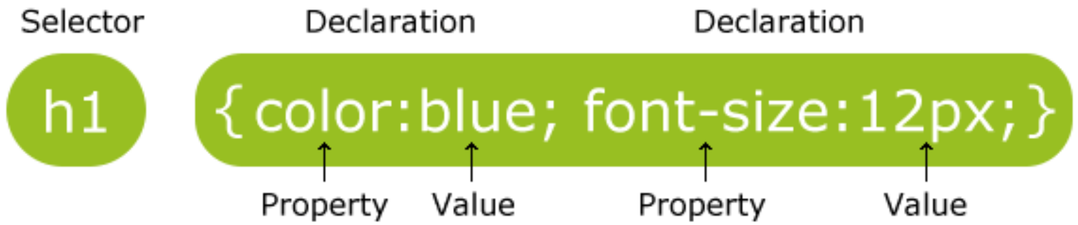

CSS
3

CSS - Cascading Style Sheets

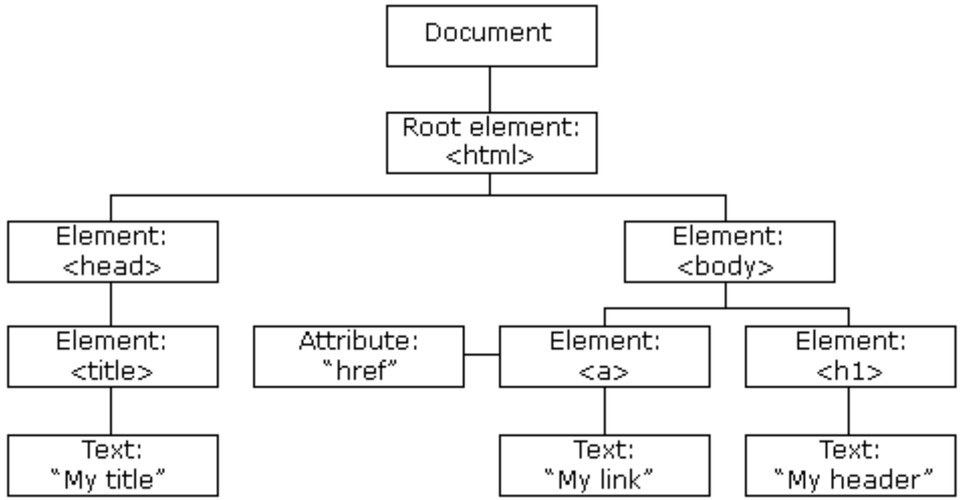
- Describes the presentation of a document
  defined using a markup language (HTML, XML)
- Decouples presentation from the document structure and behavior of a web application
- Based on declarative object descriptors and object attributes (and values) - Rules
- Based on general application priorities on conflicting cases
- Includes basic presentation behavior (animations)

Decorating the DOM

- A set of rules is applied to the DOM



- rule = selector + style definitions with the form **`property:value`**
- Rules are applied by the following order
  - Browser default
  - External and Internal Style Sheet (following the order in the section **`<head>`**)
  - Inline Style (HTML element)
- Propagate hierarchically through the DOM



CSS Selectors

from http://www.w3schools.com/cssref/css_selectors.asp

Selector	Example	Example description	CSS
.class	.intro	Selects all elements with class="intro"	1
#id	#firstname	Selects the element with id="firstname"	1
*	*	Selects all elements	2
element	p	Selects all <p> elements	1
element,element	div, p	Selects all elements and all <p> elements	1
element element	div p	Selects all <p> elements inside elements	1
element>element	div > p	Selects all <p> elements where the parent is a element	2
element+element	div + p	Selects all <p> elements that are placed immediately after elements	2
element1~element2	p ~ ul	Selects every element that are preceded by a <p> element	3
[*attribute*]	[target]	Selects all elements with a target attribute	2
[*attribute=value*]	[target=_blank]	Selects all elements with target="_blank"	2
[*attribute~=value*]	[title~=flower]	Selects all elements with a title attribute containing the word "flower"	2
[*attribute|=value*]	[lang|=en]	Selects all elements with a lang attribute value starting with "en"	2
[*attribute^=value*]	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"	3

CSS object selectors

- all elements:

```
* { font-family: sans-serif; }
```

- by HTML tag

```
h1, h2 { color: #f0f0f0; }
```

- by id

```
#form1 { border: solid 1px; }
```

- by class

```
.listitem { list-style: none; }
```

- composed selector

```
#booklist li { background-color: "rbg(100,80,10)"; }
```

- composed selector

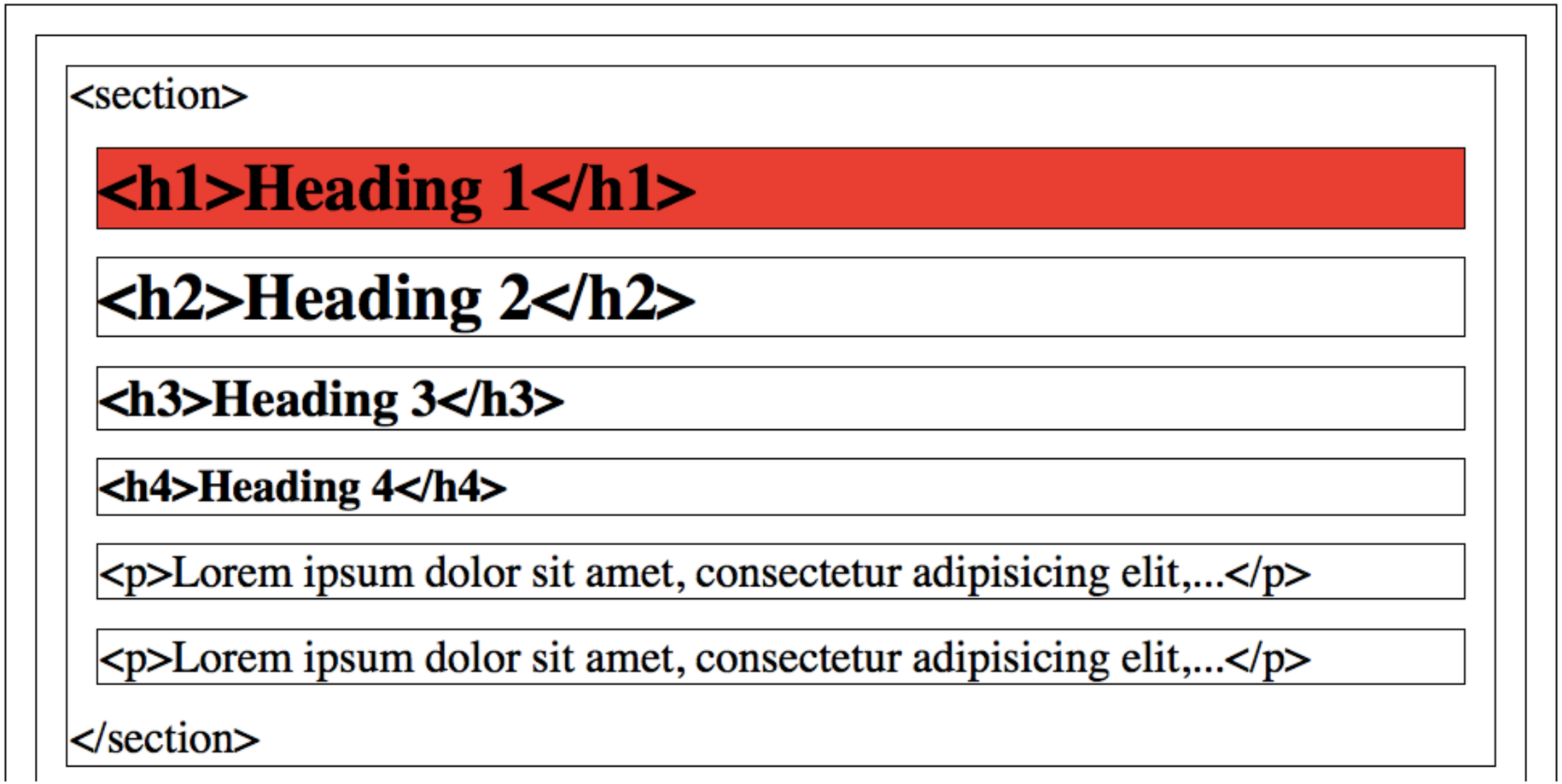
```
ul.menu > li { margin: 10px; padding: 5px;}
```

- pseudo-classes

```
.menuitem:hover { background-color: blue; }
```

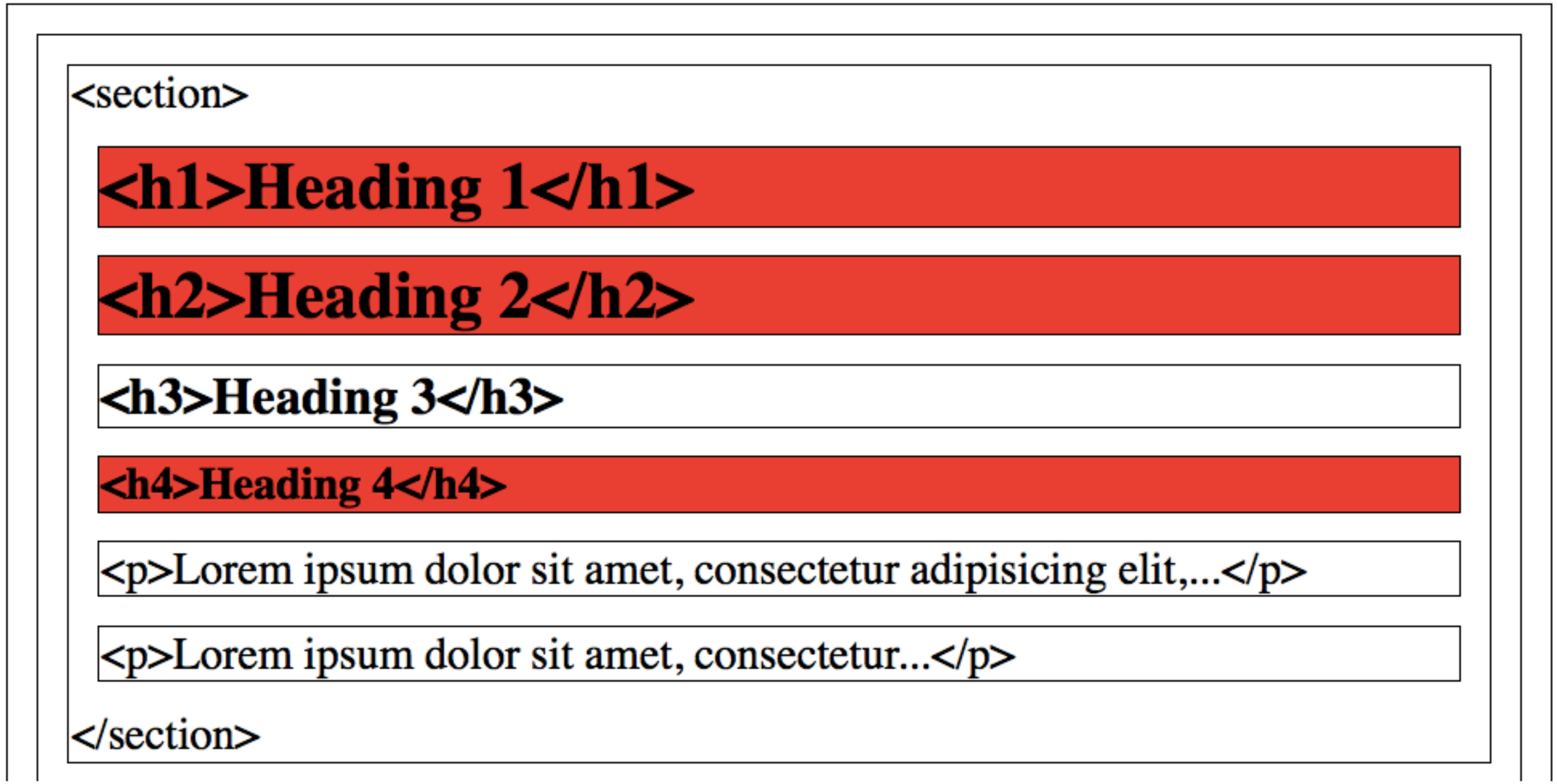
CSS Selectors

```
h1 {background-color: red;}
```



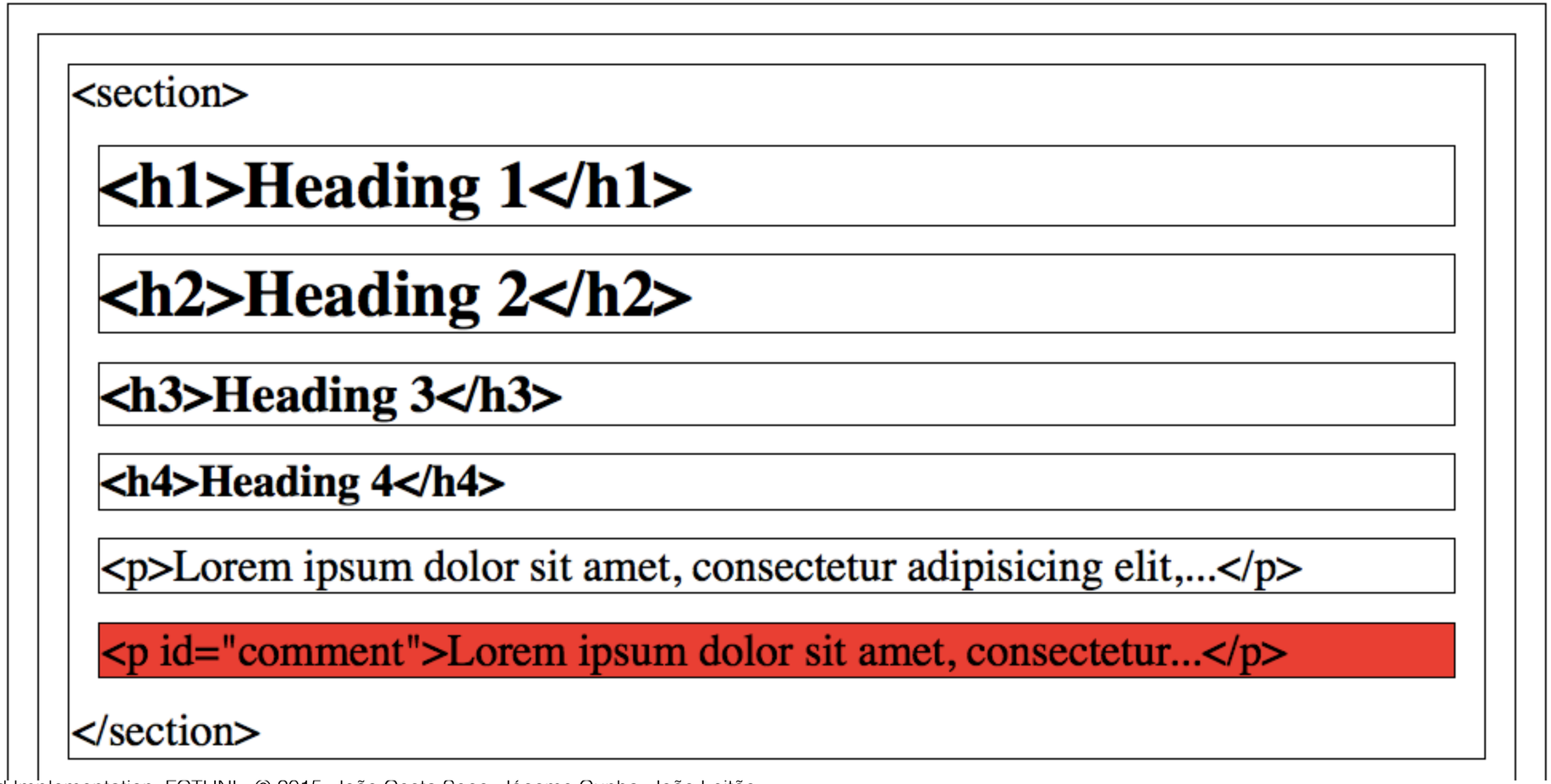
CSS Selectors

```
h1, h2, h4 {background-color: red;}
```



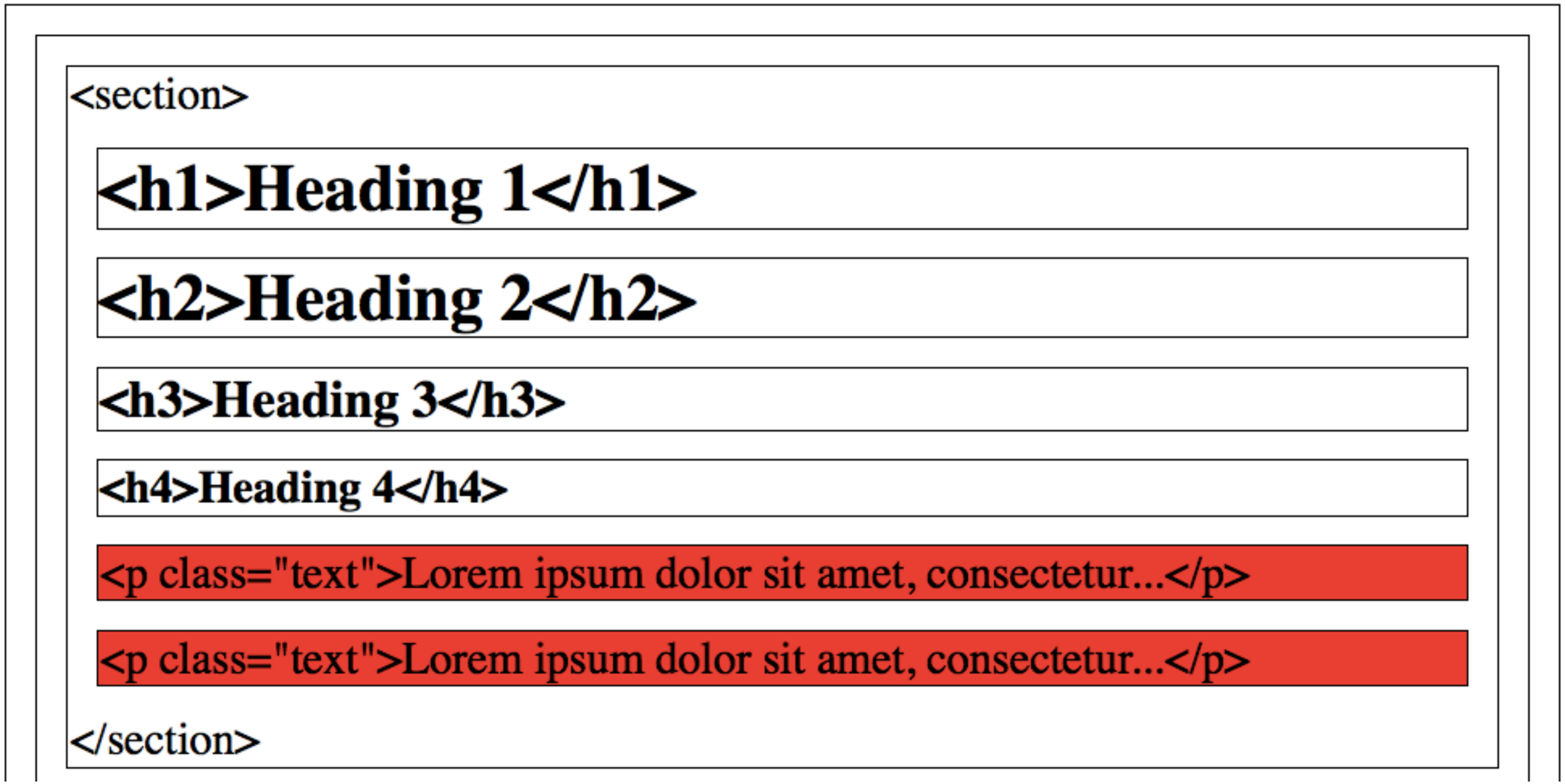
CSS Selectors

```
#comment {background-color: red;}
```



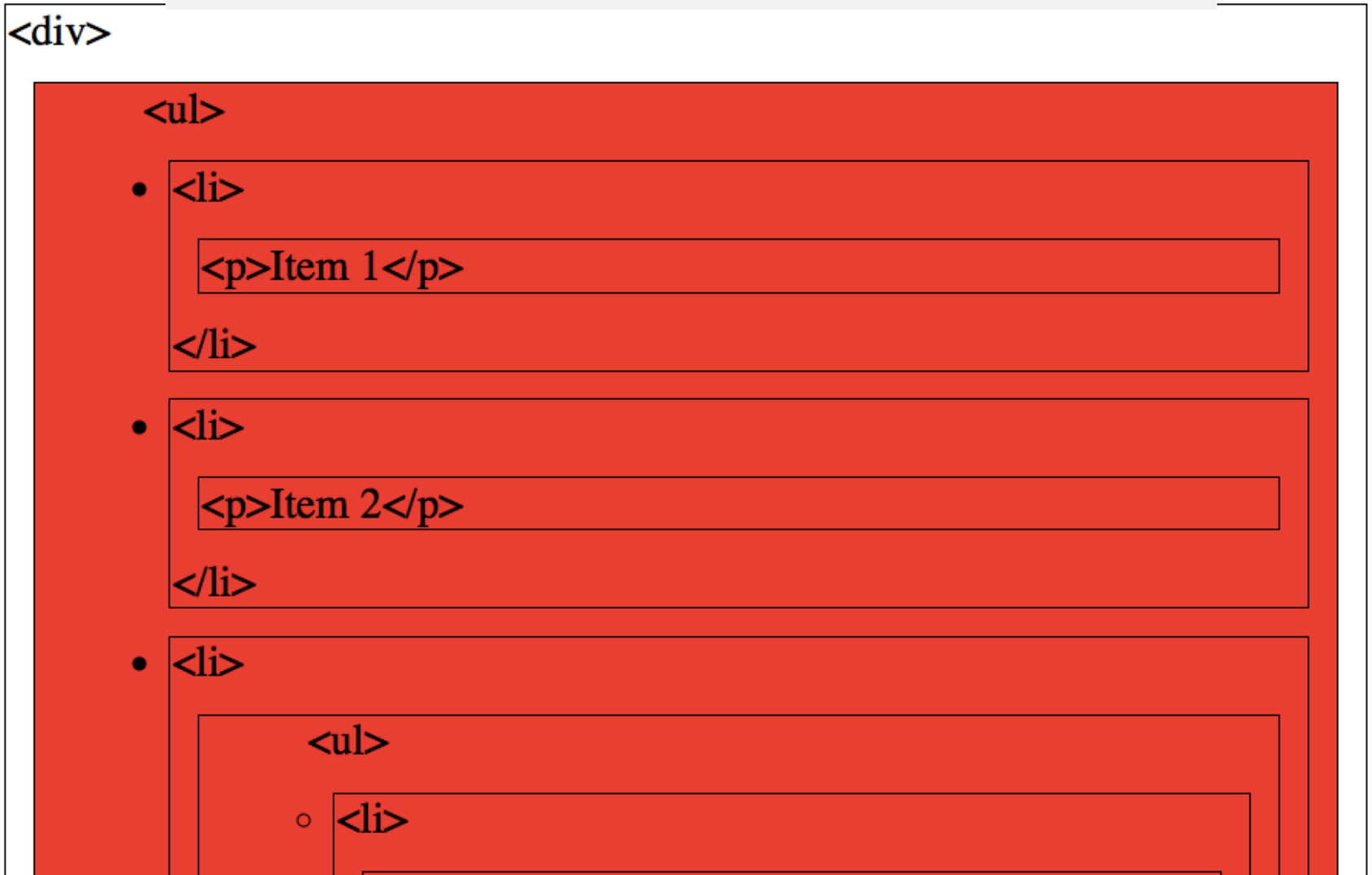
CSS Selectors

```
.text {background-color: red;}
```



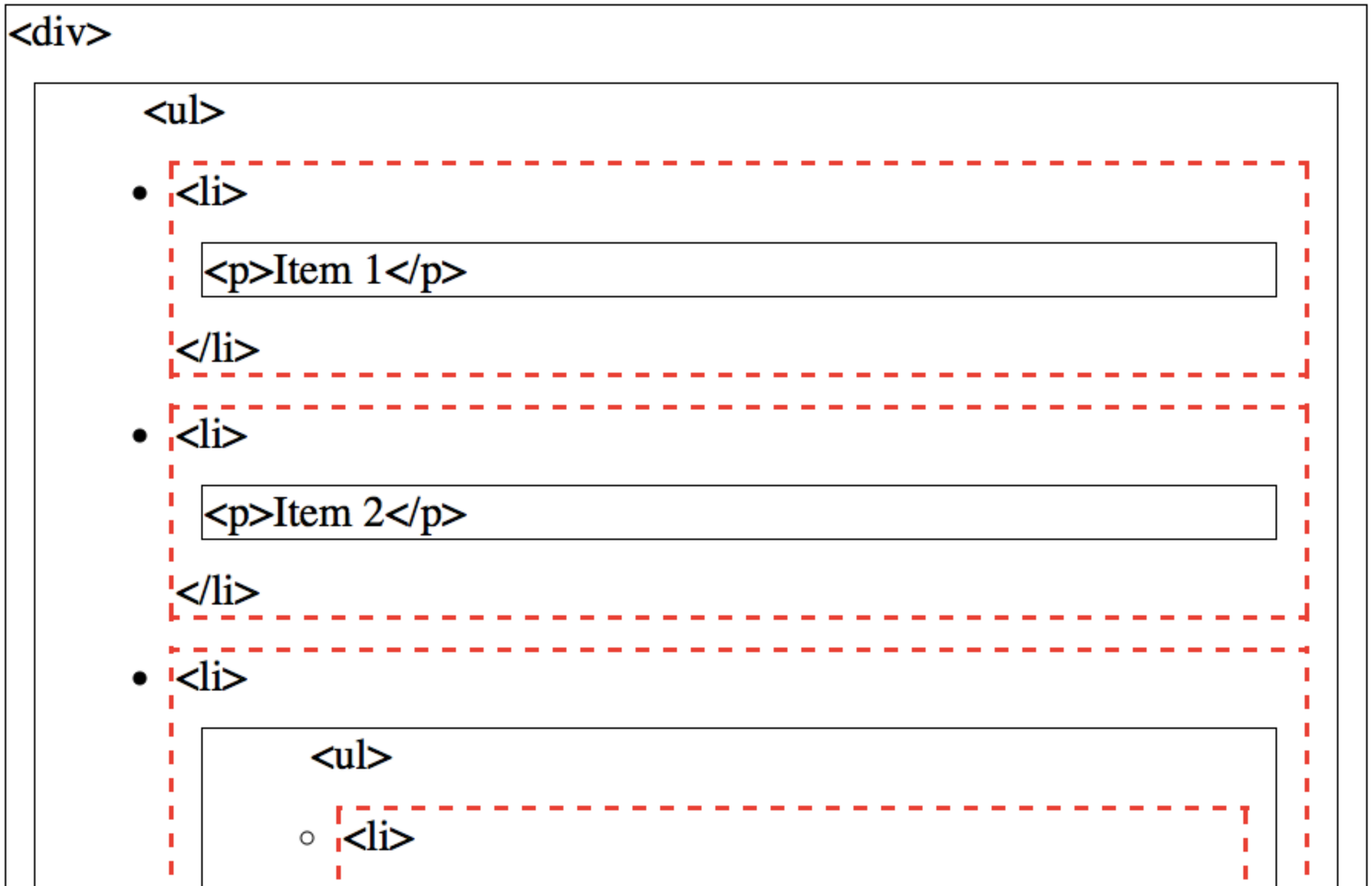
CSS Selectors

```
div ul {background-color: red;}
```



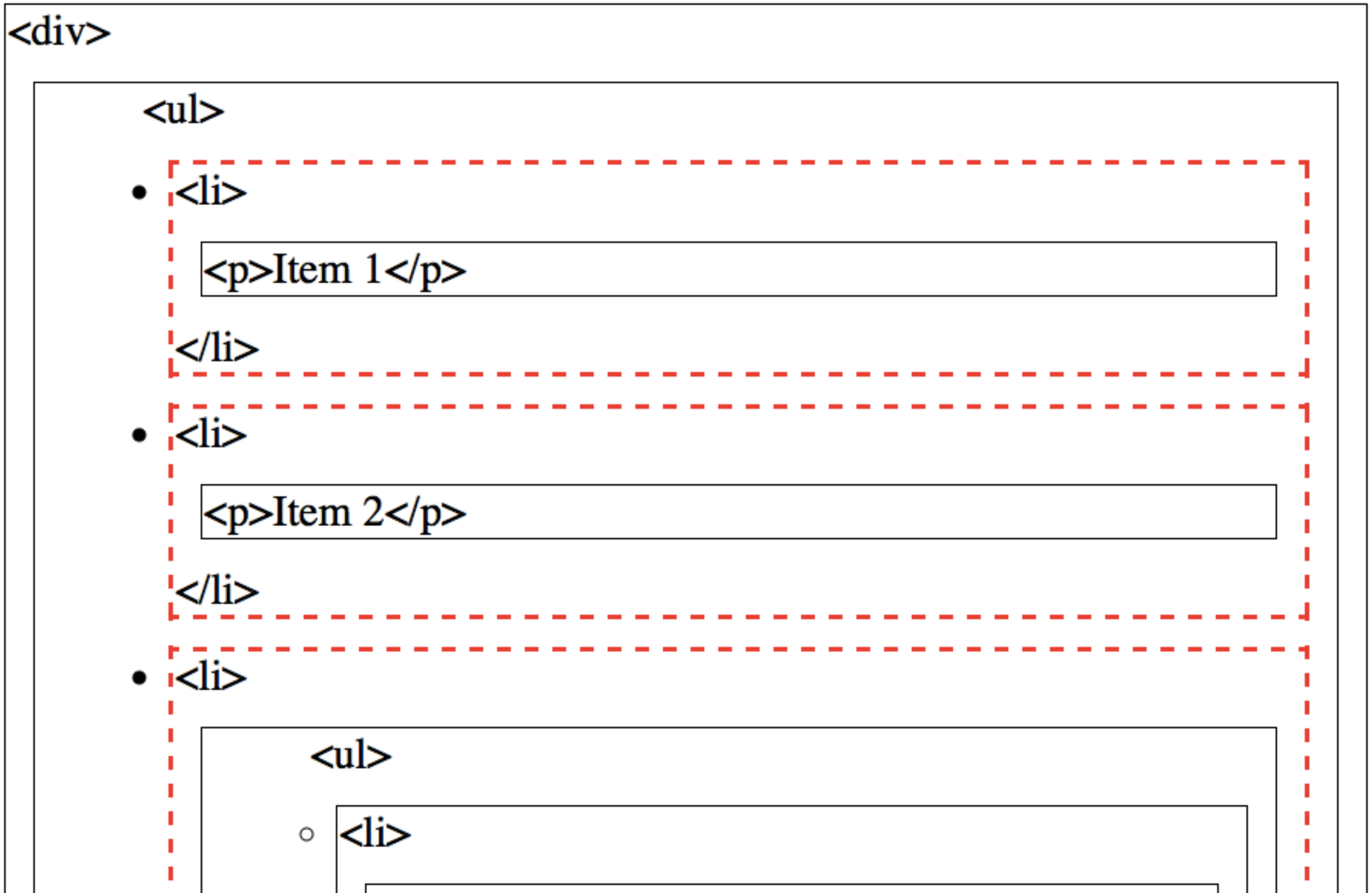
CSS Selectors

```
div li {border: dashed red 2px;}
```



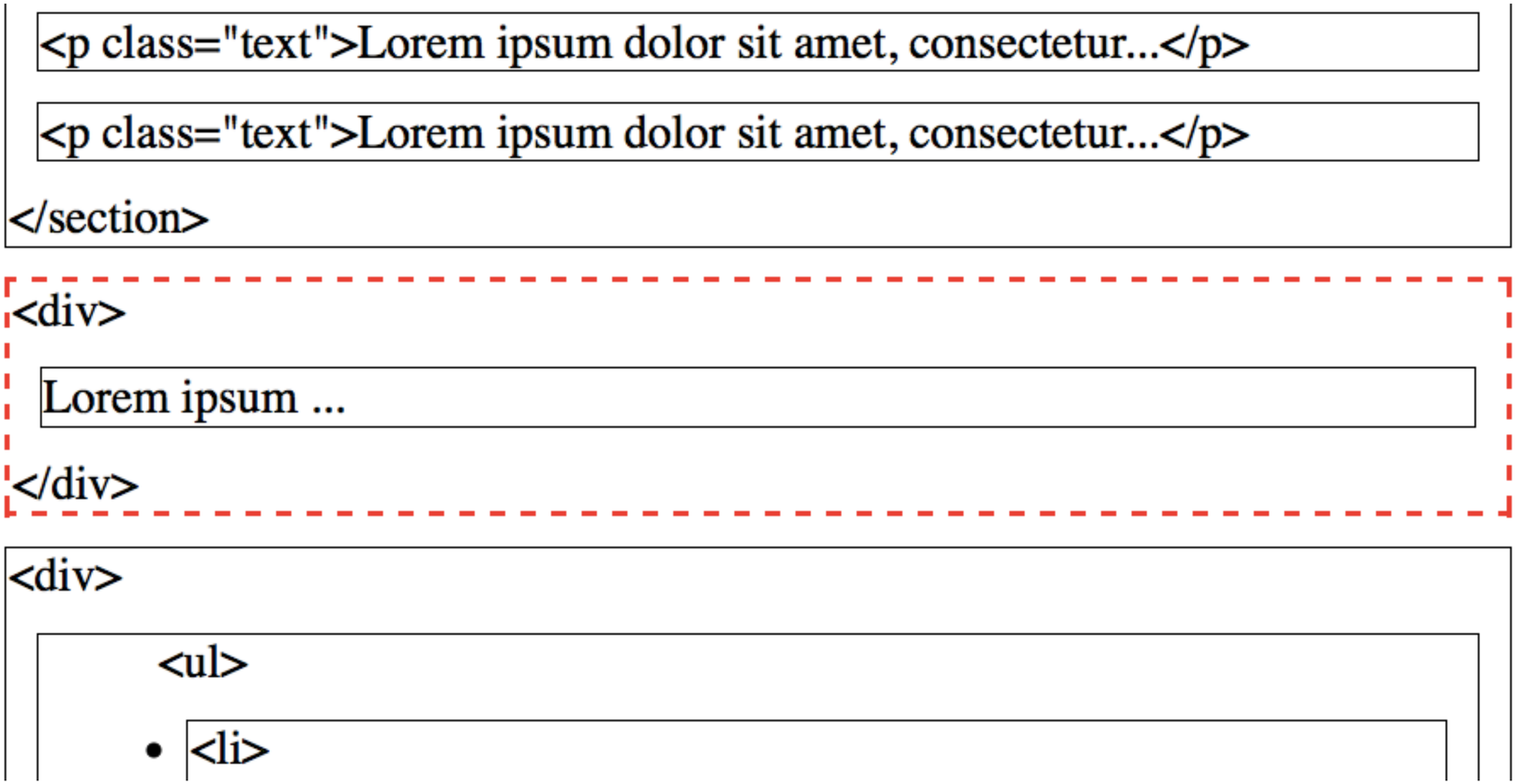
CSS Selectors

```
div > ul > li {border: dashed red 2px;}
```



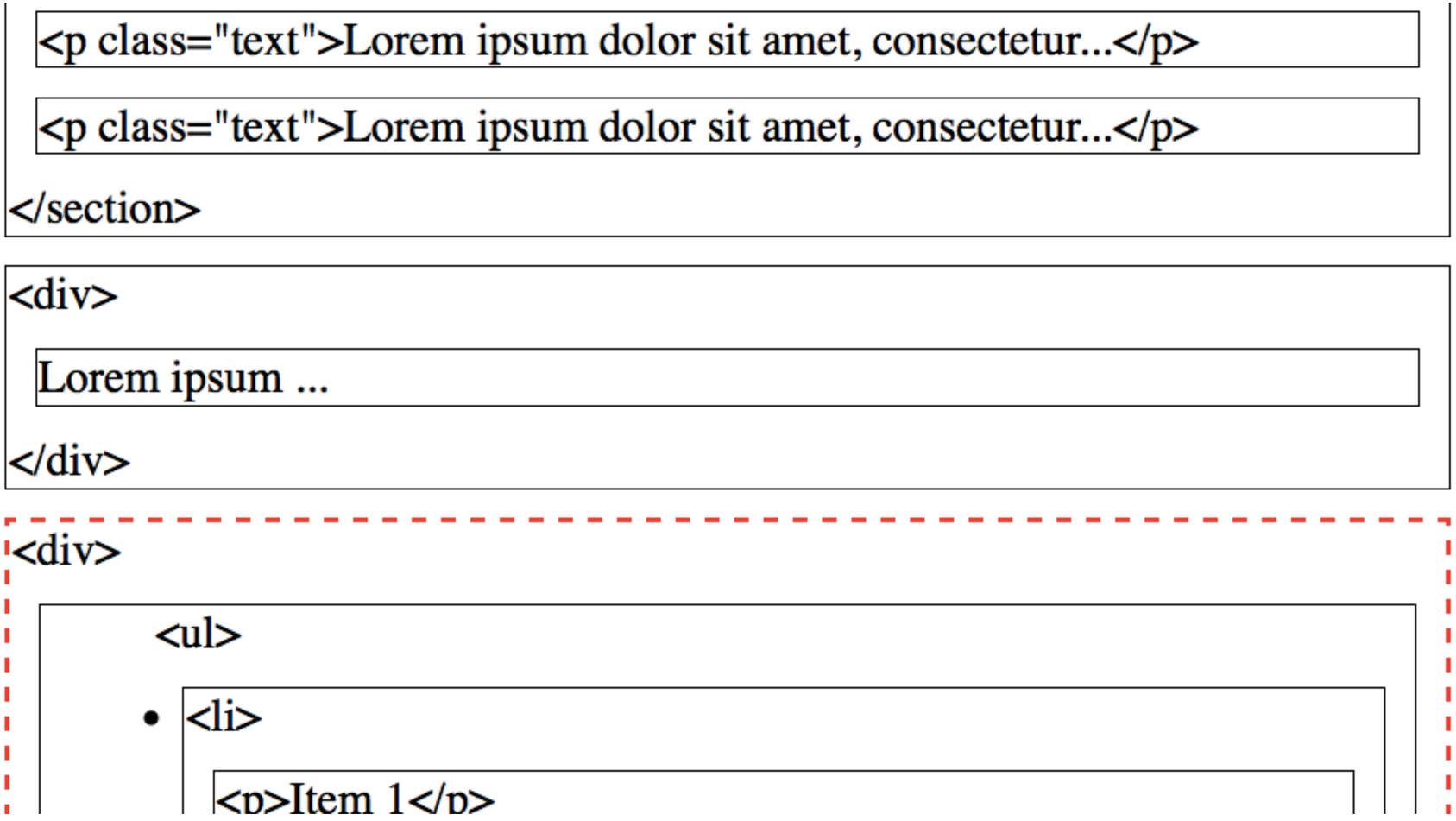
CSS Selectors

```
section+div {border: dashed red 2px;}
```



CSS Selectors

```
div~div {border: dashed red 2px;}
```



CSS performance

- CSS selectors are read from right to left

```
div#nav ul.menu > li.menuitem
```

  - Collect all menu items with CSS class .menuitem
  - Narrow down to all menu items that are actually list items (LI)
  - Narrow down to those that are under an element with CSS .menu
  - Narrow down to those which CSS .menu class belongs to a UL element
  - Narrow down to those items that fall under element of ID #nav
  - Finally, narrow the above to those elements with ID nav that are indeed DIV elements
- Better > ID > tag > class > composition > Worst

https://moduscreate.com/blog/efficient-dom-and-css/

CSS Properties

CSS Properties

CSS Property Groups

- Color
- Background and Borders
- Basic Box
- Flexible Box
- Text
- Text Decoration
- Fonts
- Writing Modes
- Table
- Lists and Counters
- Animation
- Transform
- Transition
- Basic User Interface
- Multi-column
- Paged Media
- Generated Content
- Filter Effects
- Image/Replaced Content
- Masking
- Speech
- Marquee

CSS Properties

Color Properties

Property	Description	CSS
color	Sets the color of text	1
opacity	Sets the opacity level for an element	3

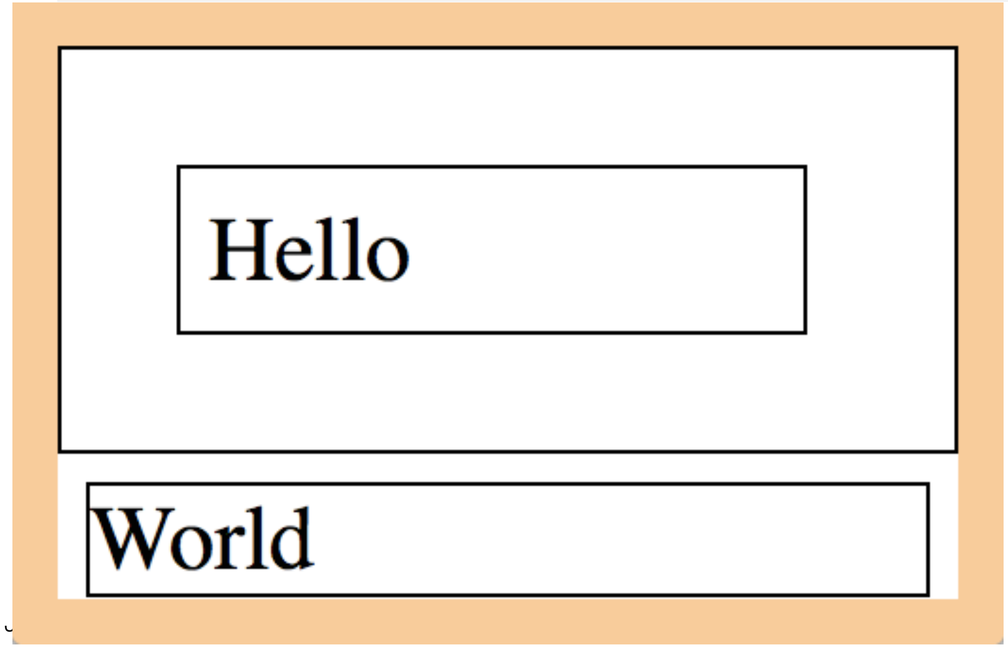
Background and Border Properties

Property	Description	CSS
background	Sets all the background properties in one declaration	1
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page	1
background-color	Sets the background color of an element	1
background-image	Sets the background image for an element	1
background-position	Sets the starting position of a background image	1
background-repeat	Sets how a background image will be repeated	1
background-clip	Specifies the painting area of the background	3
background-origin	Specifies the positioning area of the background images	3
background-size	Specifies the size of the background images	3
border	Sets all the border properties in one declaration	1

The Box Model

- The layout of web pages is based on the setting of box dimensions, using 4 different measures
  - Width and height; padding; border; margin

```
#a1 { width:100px; margin:20px; padding: 5px;}

#a2 { margin: 5px; }
```



The Box Model


```
div {
    width: 300px;
    padding: 25px;
    border: 25px solid navy;
    margin: 25px;
}
```

```
div {
    box-sizing: border-box;
    width: 50%;
    float: left;
}
```

http://css-tricks.com/box-sizing/

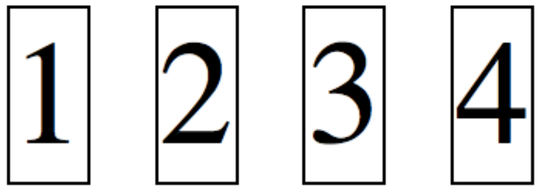
Display, Visible, Float, and Position

- There are three ways (properties) that control the positioning of an H element in a rendered page.
- **`display`**: inline, block, inline-block, none, etc.
- **`visibility`**: visible, hidden, collapse (tables)
- **`float`**: left, right…
- **`position`**: static, absolute, fixed, relative
  - associated properties: left, right, top, bottom

Display

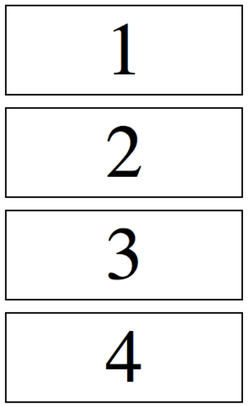
```
<div id="a1">1</div>
<div id="a2">2</div>
<div id="a3">3</div>
<div id="a4">4</div>
```

- inline



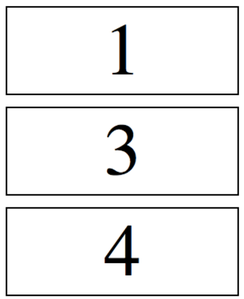

```
div { display: inline ; }
```

- block




```
div { display: block ; }
```

- none

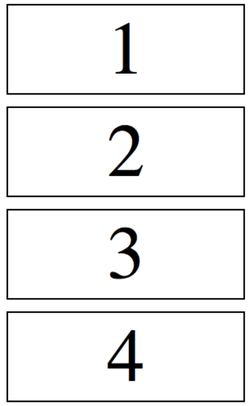



```
#a2 { display:none; }
```

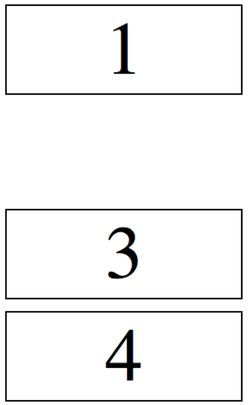
Visibility

```
<div id="a1">1</div>
<div id="a2">2</div>
<div id="a3">3</div>
<div id="a4">4</div>
```

- visible



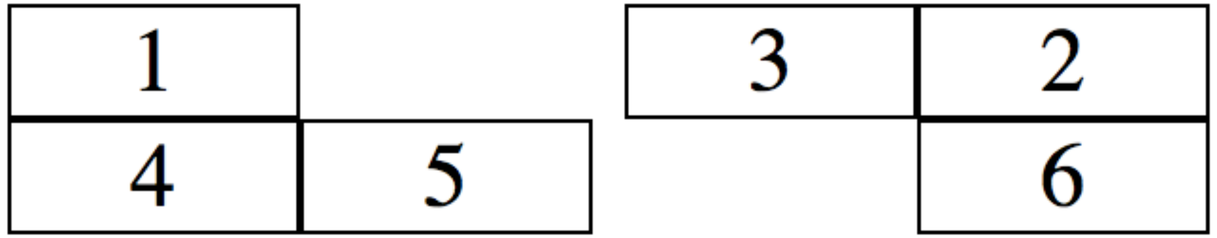
- hidden




```
#a2 { visibility:hidden;    }
```

Float & Clear

- Property **`float`** lets the elements to be arranged freely and other elements wrap around them (e.g. text)
- Possible values: left, right



- clear resets wrapping of elements
- overflow may make the container extend to the borders of the floating elements.

```
<div class="left">1</div>
<div class="right">2</div>
<div class="right">3</div>
<div class="left clear">4</div>
<div class="left">5</div>
<div class="right">6</div>
```

The Box Model

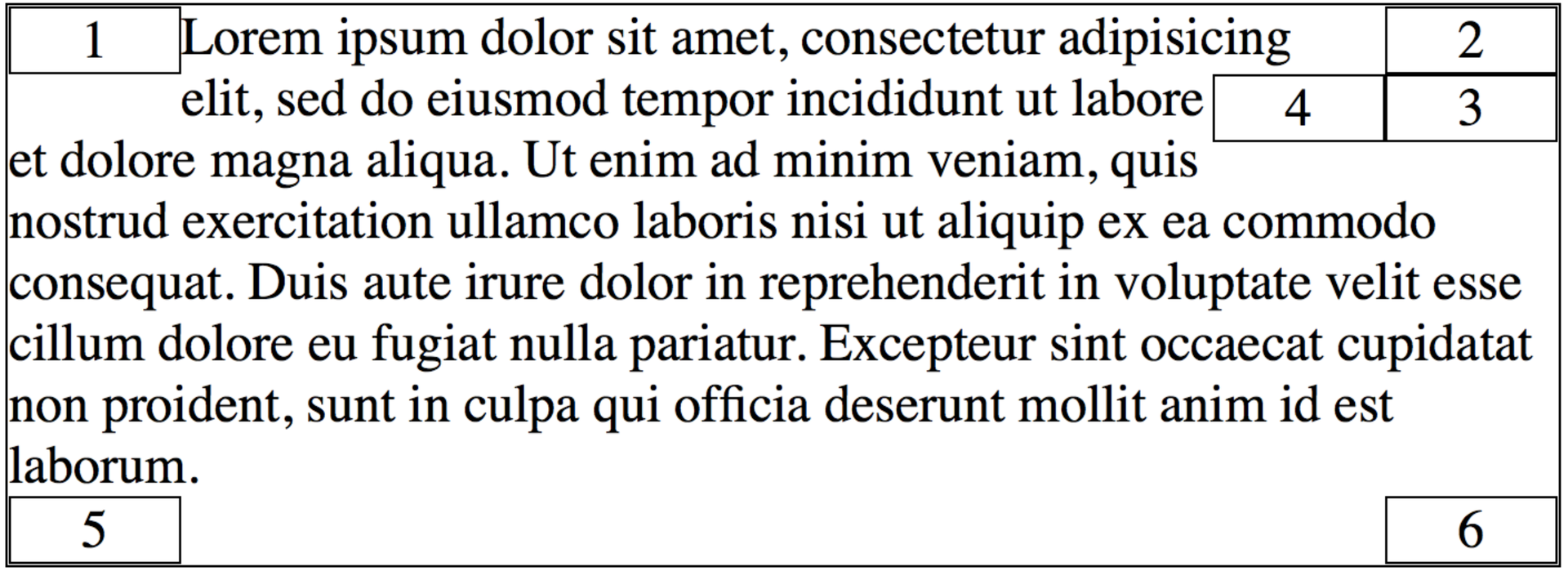
Basic Box Properties

Property	Description	CSS
bottom	Specifies the bottom position of a positioned element	2
clear	Specifies which sides of an element where other floating elements are not allowed	1
clip	Clips an absolutely positioned element	2
display	Specifies how a certain HTML element should be displayed	1
float	Specifies whether or not a box should float	1
height	Sets the height of an element	1
left	Specifies the left position of a positioned element	2
overflow	Specifies what happens if content overflows an element's box	2
overflow-x	Specifies whether or not to clip the left/right edges of the content, if it overflows the element's content area	3
overflow-y	Specifies whether or not to clip the top/bottom edges of the content, if it overflows the element's content area	3
padding	Sets all the padding properties in one declaration	1
padding-bottom	Sets the bottom padding of an element	1
padding-left	Sets the left padding of an element	1
padding-right	Sets the right padding of an element	1

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    div { border: solid 1px; width: 50px; text-align: center; }

    .left { float: left;}

    .right { float:right;}

    .clear { clear: left;}

    .container { width: 100%; }

    p { margin: 0px; text-align: left;}
</style>
</head>
<body>

<div class="container">
<div class="left">1</div>
<div class="right">2</div>
<div class="right">3</div>
<div class="right">4</div>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
<div class="left">5</div>
<div class="right">6</div>
</div>
</body>
</html>
```

1	Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.	4	3	2
5				6

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    div { border: solid 1px; width: 50px; text-align: center; }

    .left { float: left;}

    .right { float:right;}

    .clear { clear: left;}

    .container { width: 100%; overflow: auto;}

    p { margin: 0px; text-align: left;}
</style>
</head>
<body>

<div class="container">
<div class="left">1</div>
<div class="right">2</div>
<div class="right clear">3</div>
<div class="right">4</div>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
<div class="left">5</div>
<div class="right">6</div>
</div>
</body>
</html>
```



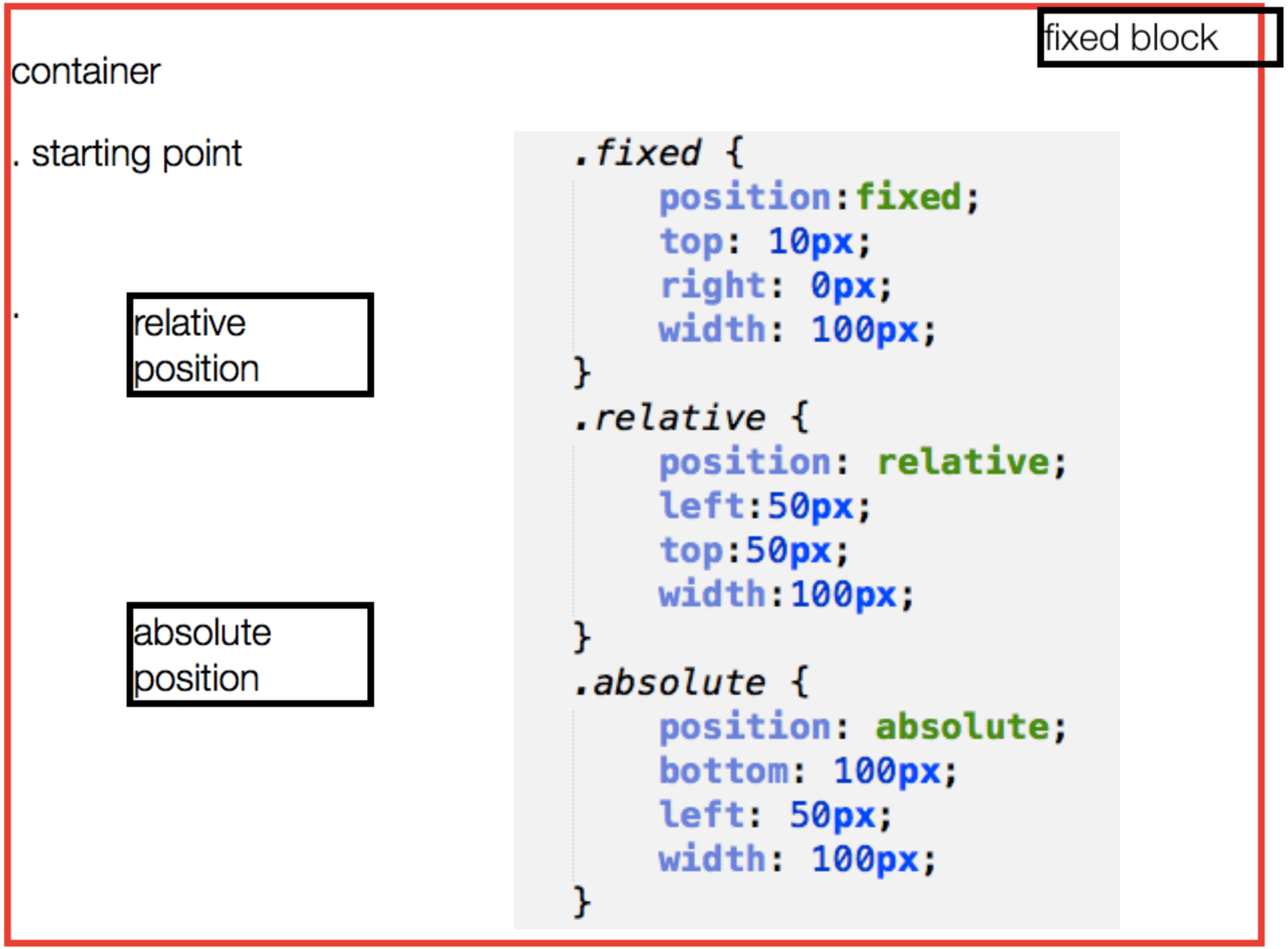
http://www.w3schools.com/css/css_float.asp

https://developer.mozilla.org/en-US/docs/Web/CSS/float

Position

- Position property defines the position of an element (independent of the flow layout).
- static (initial value, in the flow)
- absolute (with relation to the container element)
- fixed (with relation to the window)
- relative (with relation to the position in the flow)
- sticky (“normal” until the constraints are triggered, then changes to fixed position (see http://jsfiddle.net/daker/ecpTw/light/).

Position


fixed block
container
. starting point
.
relative position
absolute position
.fixed {
position:fixed;
top: 10px;
right: 0px;
width: 100px;
}
.relative {
position: relative;
left:50px;
top:50px;
width:100px;
}
.absolute {
position: absolute;
bottom: 100px;
left: 50px;
width: 100px;
}

Responsive design

- Sets of methods and techniques to optimize reading and navigation of a certain web design to a myriad of devices and displays.

- Usually achieved by gracious resizing, hiding, and rearrangement of page sections.
  - Media queries (CSS @media rule)
  - Flexible (flow and grid) layouts (e.g. bootstrap)
  - JavaScript reactive interfaces

Media queries

- The @media rule is used to define different style rules for different media types/devices.

```
<!-- CSS media query on a link element -->
<link rel="stylesheet" media="(max-width: 800px)" href="example.css" />

<!-- CSS media query within a stylesheet -->
<style>
@media (max-width: 600px) {
  .facet_sidebar {
    display: none;
  }
}
</style>
```

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

Media queries

- @media <media type> and <media features>

```
@media (min-width: 700px) { ... }
```

```
@media (min-width: 700px) and (orientation: landscape) { ... }
```

```
@media tv and (min-width: 700px) and (orientation: landscape) { ... }
```

```
@media (min-width: 700px), handheld and (orientation: landscape) { ... }
```

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

Media queries

Pseudo-BNF (for those of you that like that kind of thing)

```
media_query_list: <media_query> [, <media_query> ]*
media_query: [[only | not]? <media_type> [ and <expression> ]*]
  | <expression> [ and <expression> ]*
expression: ( <media_feature> [: <value>]? )
media_type: all | aural | braille | handheld | print |
  projection | screen | tty | tv | embossed
media_feature: width | min-width | max-width
  | height | min-height | max-height
  | device-width | min-device-width | max-device-width
  | device-height | min-device-height | max-device-height
  | aspect-ratio | min-aspect-ratio | max-aspect-ratio
  | device-aspect-ratio | min-device-aspect-ratio | max-device-aspect-ratio
  | color | min-color | max-color
  | color-index | min-color-index | max-color-index
  | monochrome | min-monochrome | max-monochrome
  | resolution | min-resolution | max-resolution
  | scan | grid
```

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

Bootstrap

Preprocessors

Bootstrap ships with vanilla CSS, but its source code utilizes the two most popular CSS preprocessors, Less and Sass. Quickly get started with precompiled CSS or build on the source.

One framework, every device.

Bootstrap easily and efficiently scales your websites and applications with a single code base, from phones to tablets to desktops with CSS media queries.

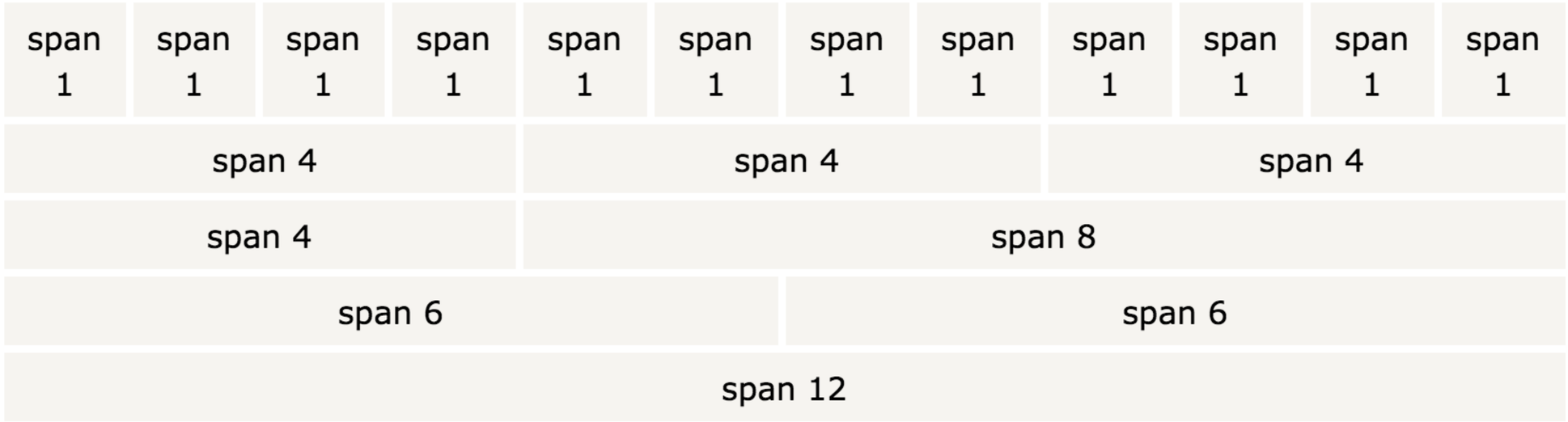
Full of features

With Bootstrap, you get extensive and beautiful documentation for common HTML elements, dozens of custom HTML and CSS components, and awesome jQuery plugins.

Bootstrap Grid System

- Bootstrap is a CSS/Javascript client framework that provides simpler and effective layout tools.
- Bootstrap divides a page into 12 columns and provides classes for rows and columns:
  - `container`, `row`, `col-X-Y`, etc.
  - X: **xs** (phones), **sm** (tablets), **md** (desktops), and **lg** (larger desktops)
  - Y: 1 .. 12
  - Explore bootstrap grids at: http://getbootstrap.com/examples/grid/

Bootstrap Grid System




WA

WebAssembly (Wasm)

- A binary instruction format designed for a stack-based virtual machine.
- High-performance execution of code in web browsers (also in servers).
- Security: The Sandbox environment ensures safe and secure execution.
- Efficiency: Small binary size leads to faster loading and execution.
- Portability, Performance, Interoperability (JavaScript, Rust, C/C++, and Go).
- Examples: Figma, AutoCAD Web App, Google Earth, Cloudflare Workers, Adobe Photoshop on Web, Unity and Unreal Engine

Internet Applications Design and Implementation

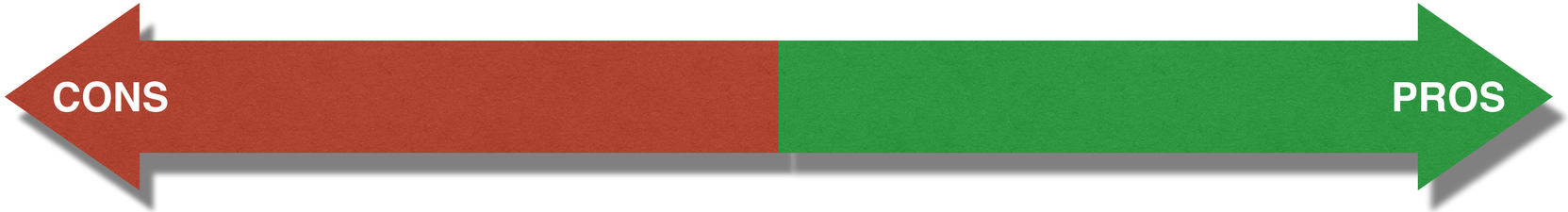
(Lecture 8 - Part 3: Client-side frameworks)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

Client-side Frameworks and Languages



Many cooperating languages

Weak binding mechanisms

Untyped development environment

Weak execution guaranties

Low-level DOM manipulation

Sequential programming language

A dynamic computational platform

Uniform across browsers (almost)

portable to mobile and desktop

Internet ready low-level interface

Low-level support for concurrency

Equipped with many libraries (node)

Client-side Frameworks and Languages

+ Abstractions + Patterns + Languages + Tools = Frameworks

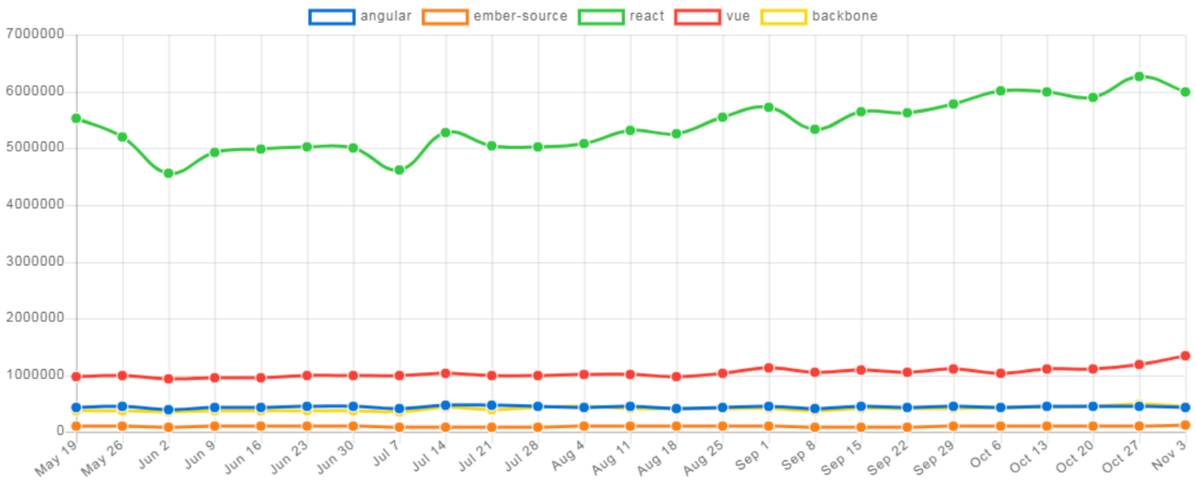
freeCodeCamp(🔥)

The Big 5 JavaScript Frameworks

The five JavaScript frameworks that currently dominate the market in terms of popularity and usage are:

- React
- Vue
- Angular
- Ember
- Backbone.js.

They each have large communities. If you are a front-end developer or are going to start your new project on front-end technologies, these five are your best bets. Here's a look at the npm trends over the last six months.



Sample Features

Often more than a client MVC

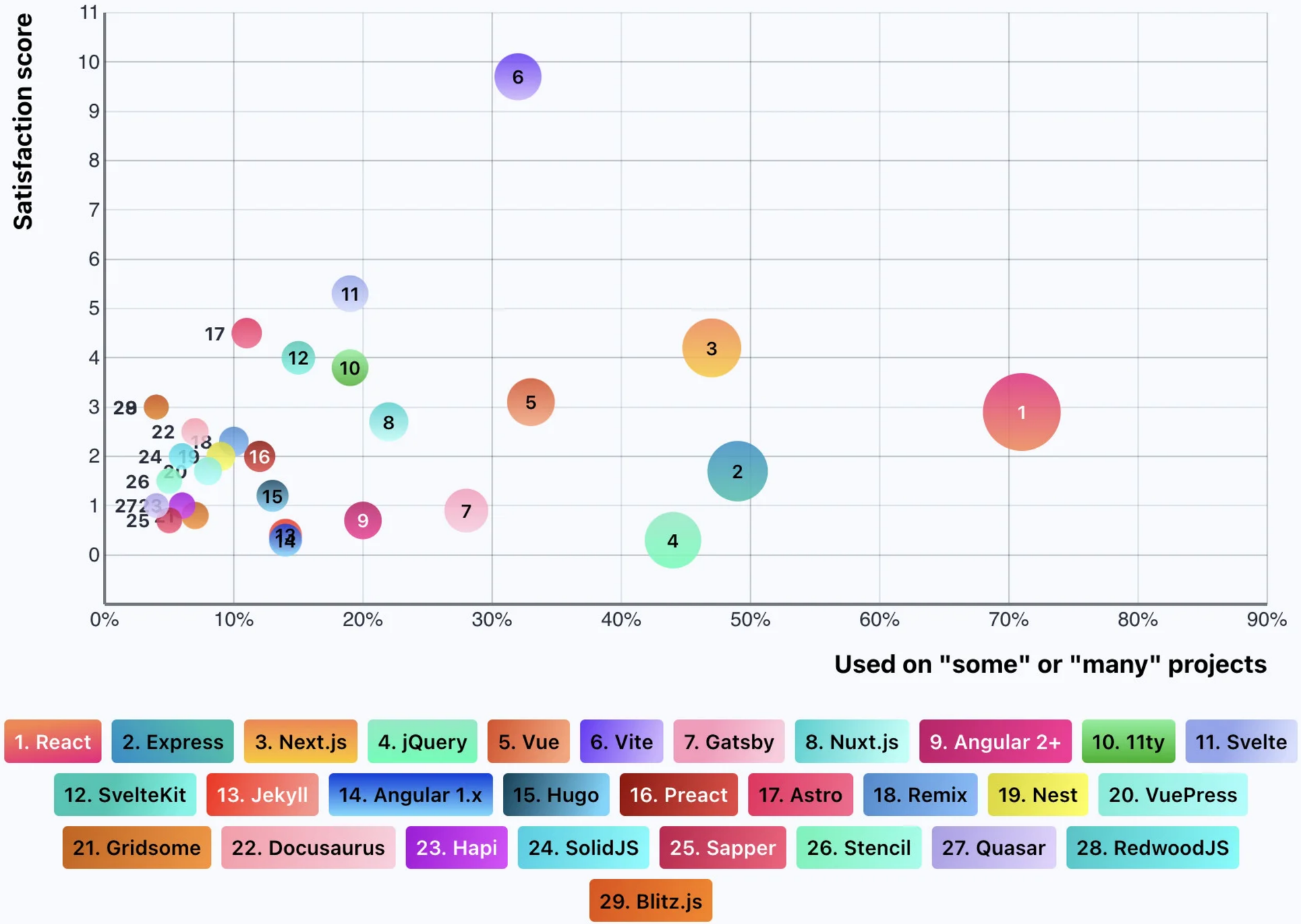
Join HTML/CSS/JS in one place

Offer abstractions of the DOM

Composition mechanisms

Out of the box reactivity

Transparency across network

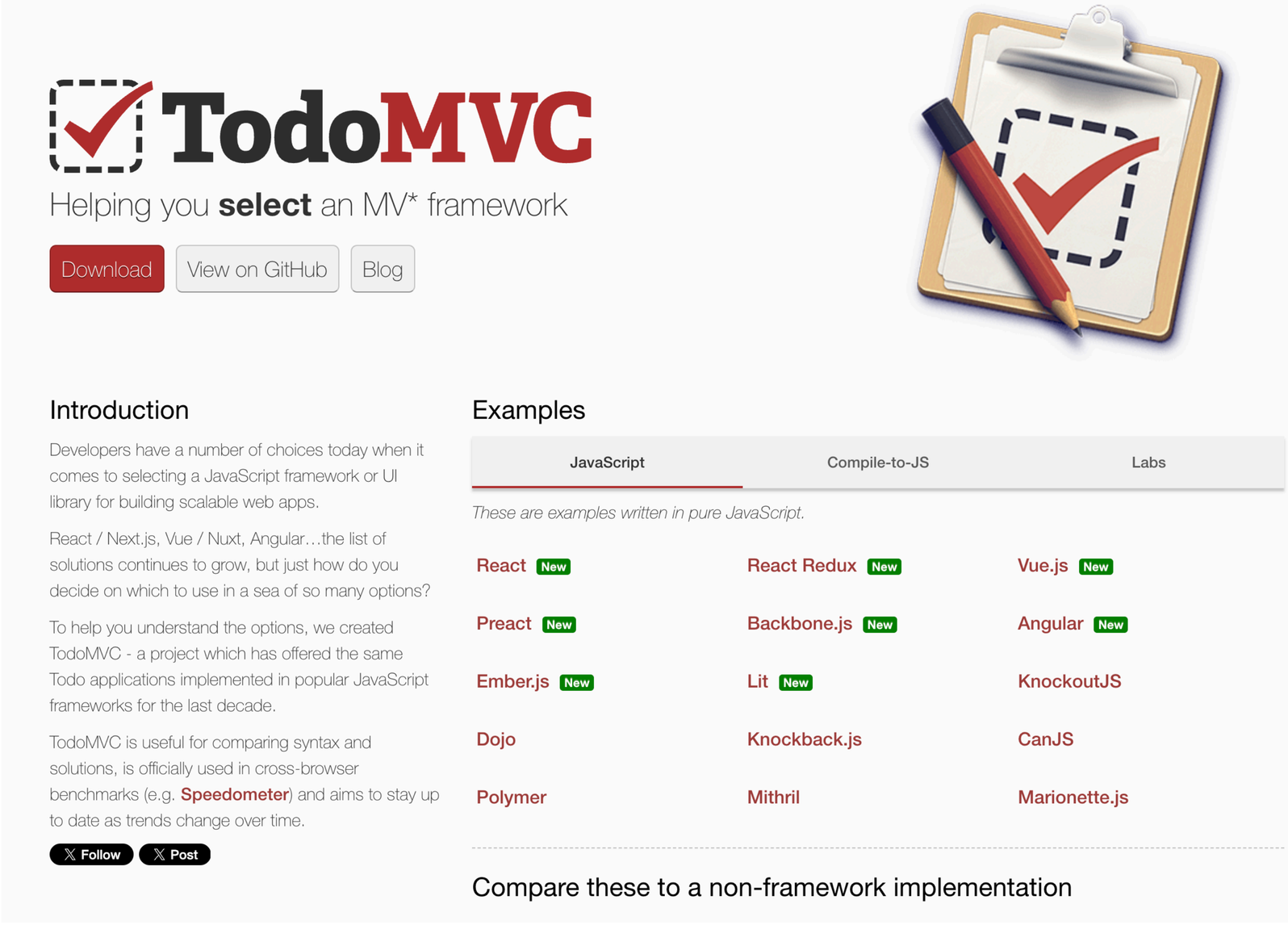


For 2025:

1. React
2. Angular
3. Vue.js
4. Svelte
5. Ember.js
6. Backbone.js
7. SolidJS
8. Preact
9. Alpine.js
10. Nuxt.js

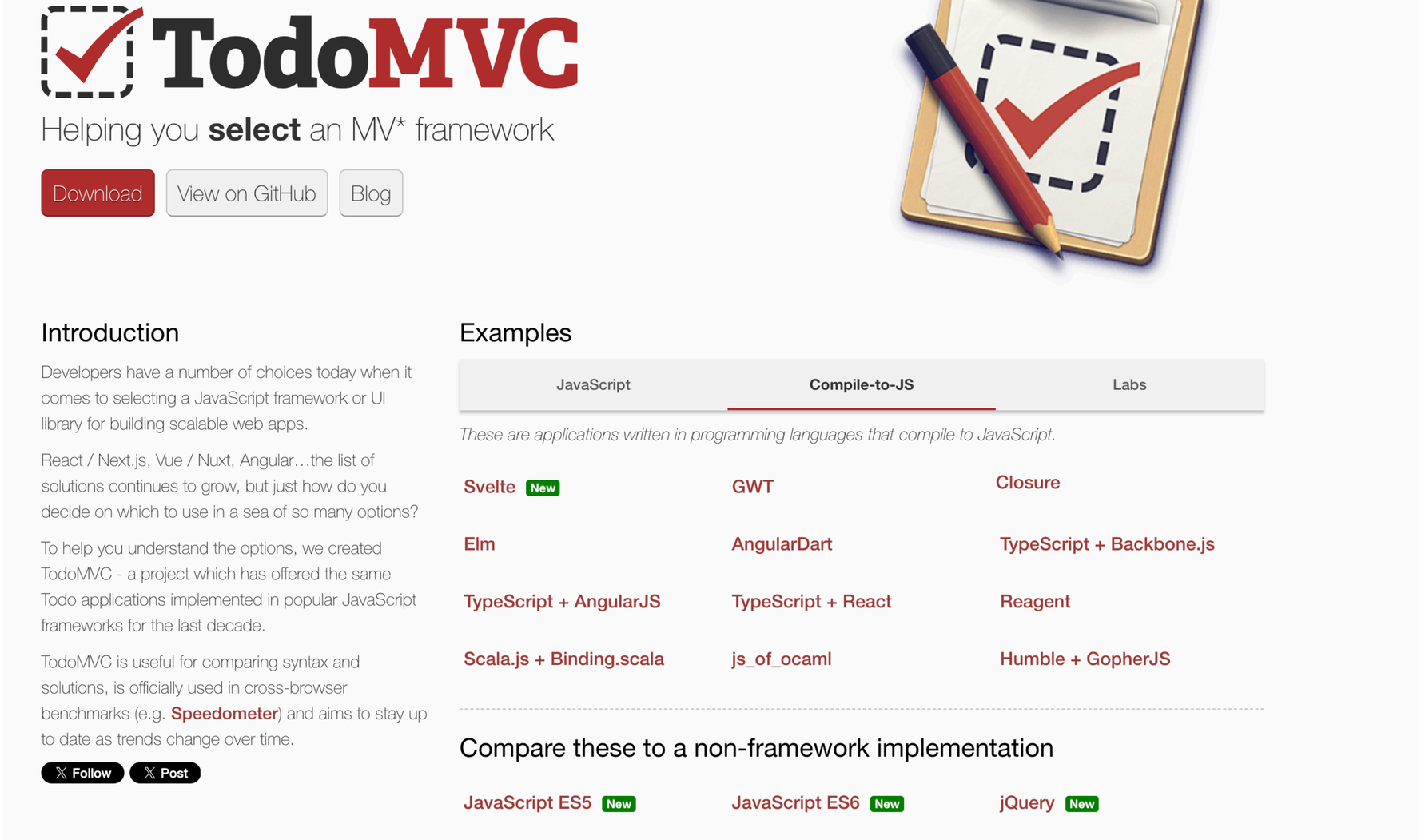
Compare Frameworks!

https://todomvc.com/



Compare Frameworks!

https://todomvc.com/



Compare Frameworks

https://todomvc.com/

flutter_architecture_samples

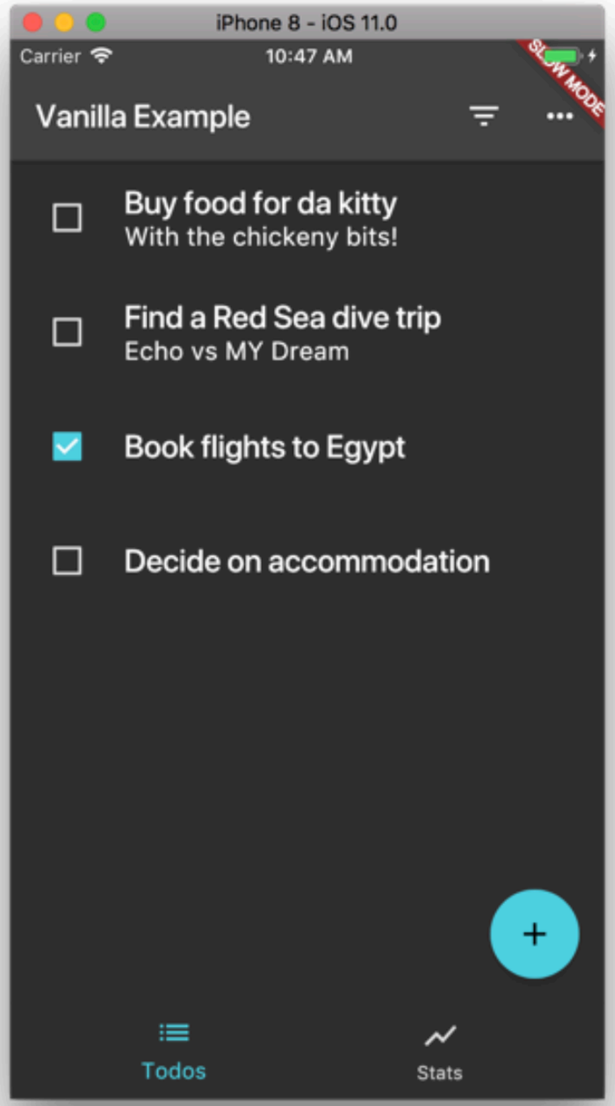
build unknown | build passing | codecov 67%

TodoMVC for Flutter!

Flutter provides a lot of flexibility in deciding how to organize and architect your apps. While this freedom is very valuable, it can also lead to apps with large classes, inconsistent naming schemes, as well as mismatching or missing architectures. These types of issues can make testing, maintaining and extending your apps difficult.

The Flutter Architecture Samples project demonstrates strategies to help solve or avoid these common problems. This project implements the same app using different architectural concepts and tools.

You can use the samples in this project as a learning reference, or as a starting point for creating your own apps. The focus of this project is on demonstrating how to structure your code, design your architecture, and the eventual impact of adopting these patterns on testing and maintaining your app. You can use the techniques demonstrated here in many different ways to build apps. Your own particular priorities will impact how you implement the concepts in these projects, so you should not consider these samples to be canonical examples. To ensure the focus is kept on the aims described above, the app uses a simple UI.



Angular

ines (75 sloc) | 3.27 KB

```
<!doctype html>
<html lang="en" data-framework="angularjs">
        <head>
                <meta charset="utf-8">
                <title>AngularJS • TodoMVC</title>
                <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
                <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
                <style>[ng-cloak] { display: none; }</style>
        </head>
        <body ng-app="todomvc">
                <ng-view></ng-view>

                <script type="text/ng-template" id="todomvc-index.html">
                        <section class="todoapp">
                                <header class="header">
                                        <h1>todos</h1>
                                        <form class="todo-form" ng-submit="addTodo()">
                                                <input class="new-todo" placeholder="What needs to be done?" ng-mod
                                        </form>
                                </header>
                                <section class="main" ng-show="todos.length" ng-cloak>
                                        <input id="toggle-all" class="toggle-all" type="checkbox" ng-model="allChec
                                        <label for="toggle-all">Mark all as complete</label>
                                        <ul class="todo-list">
                                                <li ng-repeat="todo in todos | filter:statusFilter track by $index"
                                                        <div class="view">
                                                                <input class="toggle" type="checkbox" ng-model="tod
                                                                <label ng-dblclick="editTodo(todo)">{{todo.title}}<
                                                                <button class="destroy" ng-click="removeTodo(todo)"
                                                        </div>
                                                        <form ng-submit="saveEdits(todo, 'submit')">
                                                                <input class="edit" ng-trim="false" ng-model="todo.
                                                        </form>
                                                </li>
                                        </ul>
                                </section>
```

```
8   angular.module('todomvc', ['ngRoute', 'ngResource'])
9           .config(function ($routeProvider) {
10                  'use strict';
11
12                  var routeConfig = {
13                          controller: 'TodoCtrl',
14                          templateUrl: 'todomvc-index.html',
15                          resolve: {
16                                  store: function (todoStorage) {
17                                          // Get the correct module (API or localStorage).
18                                          return todoStorage.then(function (module) {
19                                                  module.get(); // Fetch the todo records in the ba
20                                                  return module;
21                                          });
22                                  }
23                          }
24                  };
```

```
7   angular.module('todomvc')
8           .directive('todoFocus', function todoFocus($timeout) {
9                   'use strict';
10
11                  return function (scope, elem, attrs) {
12                          scope.$watch(attrs.todoFocus, function (newVal) {
13                                  if (newVal) {
14                                          $timeout(function () {
15                                                  elem[0].focus();
16                                          }, 0, false);
17                                  }
18                          });
19                  };
20          });
```

Vue

57 lines (56 sloc) | 2.43 KB

```
<!doctype html>
<html data-framework="vue">
	<head>
		<meta charset="utf-8">
		<title>Vue.js • TodoMVC</title>
		<link rel="stylesheet" href="node_modules/todomvc-common/base.css">
		<link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
		<style> [v-cloak] { display: none; } </style>
	</head>
	<body>
		<section class="todoapp" v-cloak>
			<header class="header">
				<h1>todos</h1>
				<input class="new-todo" autofocus autocomplete="off" p
			</header>
			<section class="main" v-show="todos.length">
				<input id="toggle-all" class="toggle-all" type="checkb
				<label for="toggle-all">Mark all as complete</label>
				<ul class="todo-list">
					<li class="todo" v-for="todo in filteredTodos"
						<div class="view">
							<input class="toggle" type="ch
							<label @dblclick="editTodo(tod
							<button class="destroy" @click
						</div>
						<input class="edit" type="text" v-mode
					</li>
				</ul>
			</section>
			<footer class="footer" v-show="todos.length">
```

```
	exports.app = new Vue({

		// the root element that will be compiled
		el: '.todoapp',

		// app initial state
		data: {
			todos: todoStorage.fetch(),
			newTodo: '',
			editedTodo: null,
			visibility: 'all'
		},

		// watch todos change for localStorage persistence
		watch: {
			todos: {
				deep: true,
				handler: todoStorage.save
			}
		},

		// computed properties
		// http://vuejs.org/guide/computed.html
		computed: {
			filteredTodos: function () {
				return filters[this.visibility](this.todos);
			},
			remaining: function () {
				return filters.active(this.todos).length;
			},
			allDone: {
				get: function () {
					return this.remaining === 0;
				},
				set: function (value) {
					this.todos.forEach(function (todo) {
						todo.completed = value;
```

React

33 lines (30 sloc) | 1.29 KB

```
<!doctype html>
<html lang="en" data-framework="react">
        <head>
                <meta charset="utf-8">
                <title>React • TodoMVC</title>
                <link rel="stylesheet" href="node_modules/todomvc-common/base.css">
                <link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
        </head>
        <body>
                <section class="todoapp"></section>
                <footer class="info">
                        <p>Double-click to edit a todo</p>
                        <p>Created by <a href="http://github.com/petehunt/">petehunt</a></p>
                        <p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
                </footer>

                <script src="node_modules/todomvc-common/base.js"></script>
                <script src="node_modules/react/dist/react-with-addons.js"></script>
                <script src="node_modules/classnames/index.js"></script>
                <script src="node_modules/react/dist/JSXTransformer.js"></script>
                <script src="node_modules/director/build/director.js"></script>

                <script src="js/utils.js"></script>
                <script src="js/todoModel.js"></script>
                <!-- jsx is an optional syntactic sugar that transforms methods in React's
                `render` into an HTML-looking format. Since the two models above are
                unrelated to React, we didn't need those transforms. -->
                <script type="text/jsx" src="js/todoItem.jsx"></script>
                <script type="text/jsx" src="js/footer.jsx"></script>
                <script type="text/jsx" src="js/app.jsx"></script>
        </body>
</html>
```

```
function render() {
        React.render(
                <TodoApp model={model}/>,
                document.getElementsByClassName('todoapp')[0]
        );
}
```

```
return (
        <div>
                <header className="header">
                        <h1>todos</h1>
                        <input
                                className="new-todo"
                                placeholder="What needs to be done?"
                                value={this.state.newTodo}
                                onKeyDown={this.handleNewTodoKeyDown}
                                onChange={this.handleChange}
                                autoFocus={true}
                        />
                </header>
                {main}
                {footer}
        </div>
);
```

React and Kotlin/JS

```kotlin
import kotlinx.html.*
import kotlinx.html.js.*
import org.jetbrains.common.*
import org.jetbrains.demo.thinkter.model.*
import react.*
import react.dom.*
import kotlin.browser.*
import kotlinx.coroutines.experimental.async

class LoginComponent : ReactDOMComponent<UserProps, LoginFormSta
    companion object : ReactComponentSpec<LoginComponent, UserPr

    init {
        state = LoginFormState("", "", false, "")
    }

    override fun ReactDOMBuilder.render() {
        div {
            form(classes = "pure-form pure-form-stacked") {
                legend { +"Login" }

                fieldSet(classes = "pure-group") {
                    input(type = InputType.text, name = "login")
                        value = state.login
                        placeholder = "Login"
                        disabled = state.disabled

                        onChangeFunction = {
                            setState {
                                login = it.inputValue
                            }
                        }
                    }
                    input(type = InputType.password, name = "pas
                        value = state.password
```

```kotlin
1  package org.jetbrains.demo.thinkter
2
3  import kotlinx.html.*
4  import kotlinx.html.js.*
5  import org.jetbrains.demo.thinkter.model.*
6  import react.*
7  import react.dom.*
8  import kotlin.browser.*
9  import kotlinx.coroutines.experimental.async
10
11 fun main(args: Array<String>) {
12     runtime.wrappers.require("pure-blog.css")
13
14     ReactDOM.render(document.getElementById("content")) {
15         div {
16             Application {}
17         }
18     }
19 }
20
21 class Application : ReactDOMComponent<ReactComponentNoProps, ApplicationPageState>() {
22     companion object : ReactComponentSpec<Application, ReactComponentNoProps, ApplicationPageState>
23     val polling = Polling()
24
25     init {
26         state = ApplicationPageState(MainView.Home)
27         checkUserSession()
28     }
29
30     override fun componentWillUnmount() {
31         polling.stop()
32         super.componentWillUnmount()
33     }
34
35     override fun ReactDOMBuilder.render() {
36         div("pure-g") {
```

Elm Language (elm-lang.org)

Why a *functional* language?

- No runtime errors in practice. No `null`. No `undefined` is not a function.
- Friendly error messages that help you add features more quickly.
- Well-architected code that *stays* well-architected as your app grows.
- Automatically enforced semantic versioning for all Elm packages.

https://ellie-app.com/new

```
module Main exposing (main)

import Browser
import Html exposing (Html, button, div, text)
import Html.Events exposing (onClick)

type alias Model = { count : Int }

initialModel : Model
initialModel = { count = 0 }

type Msg = Increment
         | Decrement

update : Msg -> Model -> Model
update msg model =
    case msg of
        Increment ->
            { model | count = model.count + 1 }

        Decrement ->
            { model | count = model.count - 1 }

view : Model -> Html Msg
view model =
    div []
        [ button [ onClick Increment ] [ text "+1" ]
        , div [] [ text <| String.fromInt model.count ]
        , button [ onClick Decrement ] [ text "-1" ]
        ]

main : Program () Model Msg
main =
    Browser.sandbox
        { init = initialModel
        , view = view
        , update = update
        }
```

Elm Lang

Why a *functi*

- No runtime errors
- Friendly error me
- Well-architected
- Automatically enf

https://ellie

BEST FUNCTIONAL LANGUAGES TO LEARN FOR WEB-FRONTEND DEVELOPMENT	PRICE	SITE	PRICE
90 Elm	-	http://elm-lang.org/	Open Sou
-- Reason ML	-	-	Free, Open s
71 ClojureScript	OPEN SOURCE	https://clojurescript.org/	Open Sou
-- OCaml	-	http://ocaml.org	Open source
-- F#	-	http://www.fsharp.org	-
-- Purescript	-	-	-
58 Scala	-	scala-lang.org	Open Source
-- JavaScript	-	-	-
-- Haste	-	-	-
-- Haskell	-	https://www.haskell.org/	-
-- Elixir	-	https://elixir-lang.org	Open source
-- Kotlin	-	http://kotlinlang.org	Free, Open S

```
kt)
1 }
1 }
kt "+1" ]
odel.count ]
kt "-1" ]
```

Loom Language

- By Nuno Castro Martins, MSc FCT UNL 2017

https://loom-lang.gitlab.io/loom-playground/ https://gitlab.com/loom-lang/loom

```
// Tasks counter
var id = 0

// Mutable list of tasks
var tasks = mut [createTask("Sleep"), createTask("Eat"), createTask("Play")]

// Creates a new task object (with a mutable "done" value)
function createTask(text)
  ({id: id++, text: text, done: mut false})

// Adds a task to the list of tasks when "Enter" is pressed
function addTaskOnEnter(e)
  if (e.key == "Enter") {
    *tasks = [...*tasks, createTask(e.target.value)]
    e.target.value = ""
  }

// CSS of a task
function taskCSS(task)
  css {
    |.text:if(task.done)| {
      textDecoration: "line-through"
    }
  }
```

Loom Language

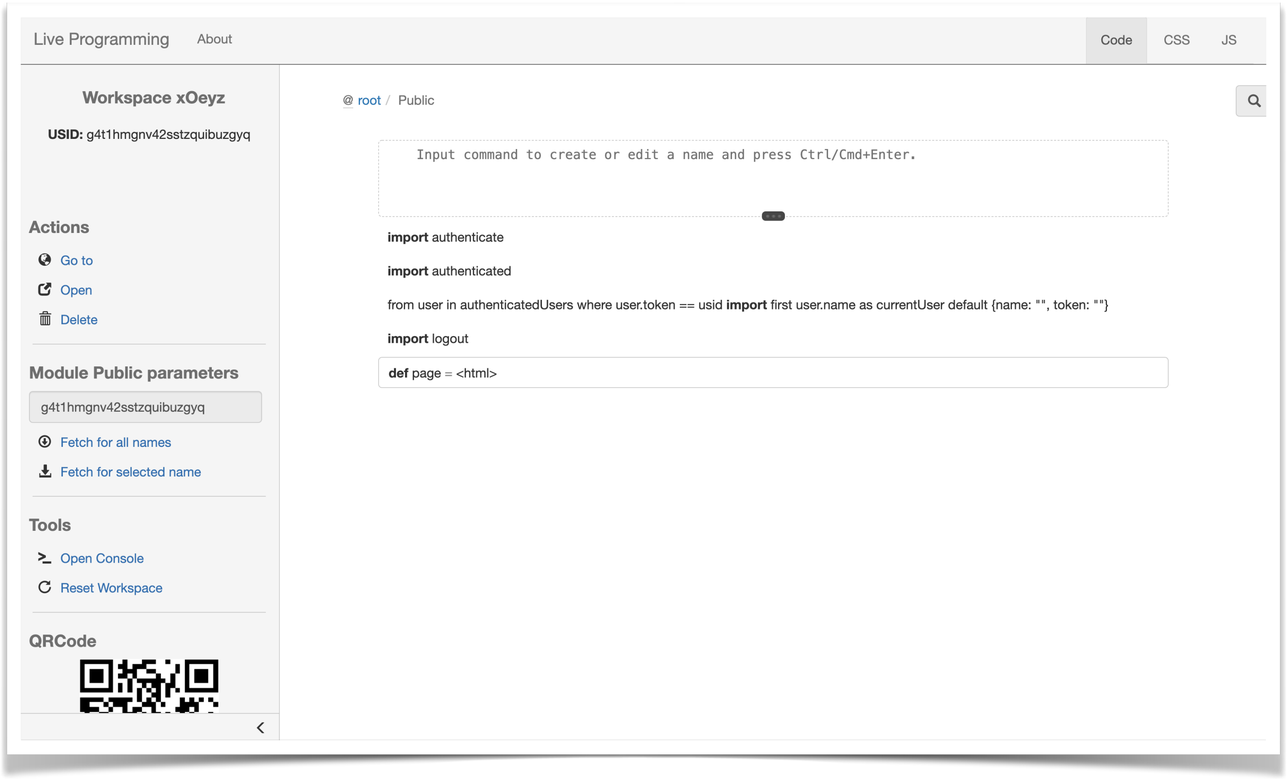
- By Nuno Castro Martins, MSc FCT UNL 2017

https://loom-lang.gitlab.io/loom-playground/ https://gitlab.com/loom-lang/loom

```
// HTML representation of a task
function taskHtml(task)
  <li.task {key: task.id, css: taskCSS(task)}> [
    <input.done {type: "checkbox", checked: task.done} />
    <span.text> task.text
  ]

// Number of tasks left to do
var nLeft = sig *tasks.reduce((n, task) => n + if (*(task.done)) 0 else 1, 0)

// Page to show
export default <div> [
  <input {onKeyDown: addTaskOnEnter} />
  <span> [nLeft, " left"]
  <ul>
    sig for (var task in *tasks) taskHtml(task)
]
```

Live Programming

- By João Costa Seco, Miguel Domingues, João Mateus, Tiago Lopes, Gil Alves



Interested in research?

Internet Applications Design and Implementation

(Lecture 8 - Part 4: Front-end languages)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

JavaScript in the browser

```
<h1>JavaScript Can Validate Input</h1>

<p>Please input a number between 1 and 10:</p>

<input id="numb" type="number">

<button type="button" onclick="myFunction()">Submit</button>

<p id="demo"></p>

<script>
function myFunction() {
    var x, text;

    // Get the value of the input field with id="numb"
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>
```

JavaScript - events

```
<some-HTML-element some-event='some JavaScript'>
```

```
<button onclick='document.getElementById("demo").innerHTML=Date()'>
The time is?
</button>
```

```
<button onclick="displayDate()">The time is?</button>
```

JavaScript + AJAX

- Asynchronous requests

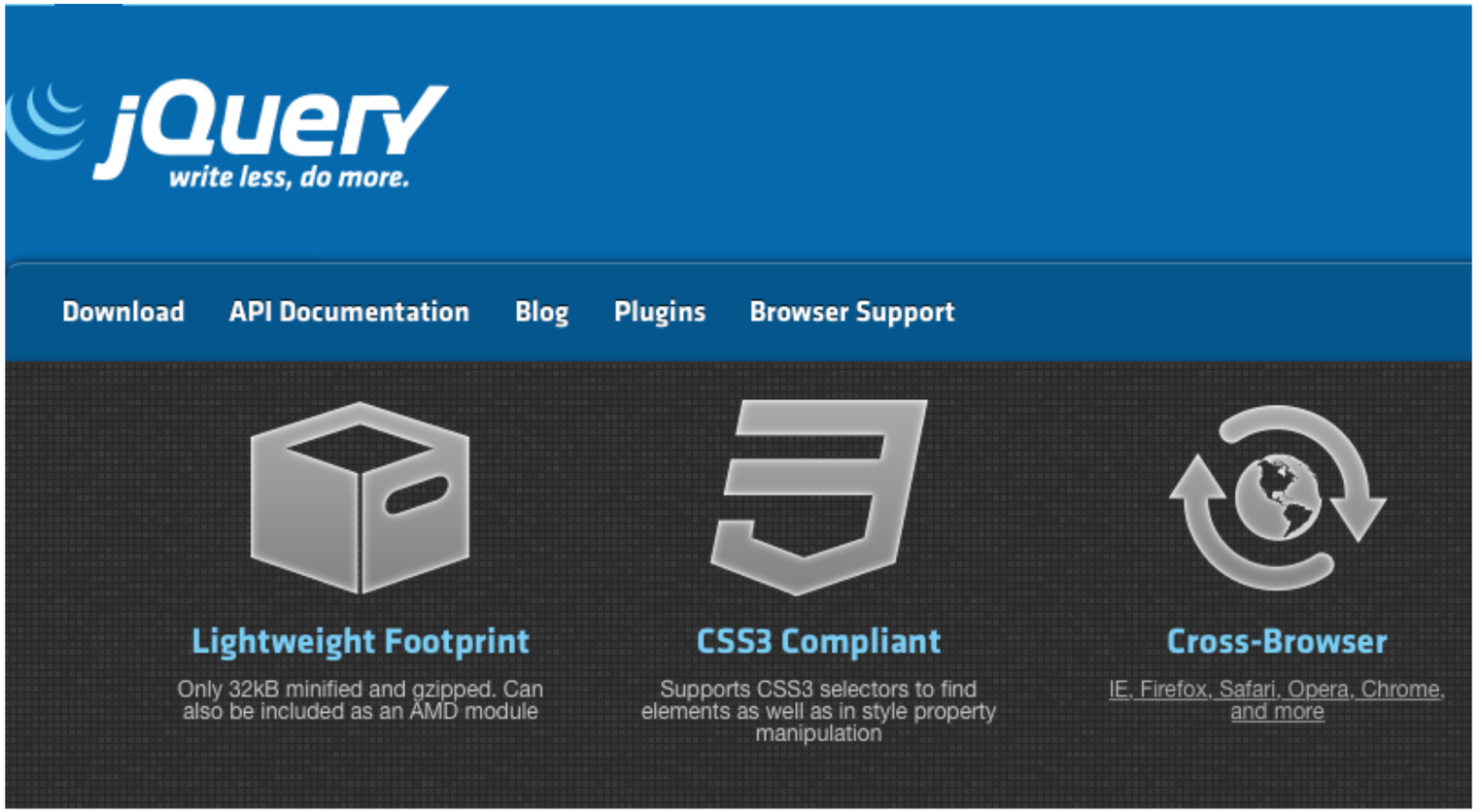
```
<div id="demo"><h2>Let AJAX change this text</h2></div>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
      document.getElementById("demo").innerHTML = xhttp.responseText;
    }
  }
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>
```

JavaScript Frameworks

JavaScript Frameworks

- Abstraction of browser related details
  (No, they are not the same as the standard!).
- Higher abstraction level in browser related operations.
  - Searching the DOM and iterating elements (jQuery)
  - Updating values of elements in the DOM (React)
- Inversion of control
  - Basic event model already provide it
  - Frameworks extend it further

jQuery



What is jQuery?

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

jQuery

$(*selector*).*action*()

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

jQuery - examples

$(this).hide() - hides the current element.

$("p").hide() - hides all <p> elements.

$(".test").hide() - hides all elements with class="test".

$("#test").hide() - hides the element with id="test".

```
$(document).ready(function(){
    // jQuery methods go here...
});
```

jQuery - selectors

Syntax	Description	Ex
$("*")	Selects all elements	Tr
$(this)	Selects the current HTML element	Tr
$("p.intro")	Selects all <p> elements with class="intro"	Tr
$("p:first")	Selects the first <p> element	Tr
$("ul li:first")	Selects the first element of the first	Tr
$("ul li:first-child")	Selects the first element of every	Tr
$("[href]")	Selects all elements with an href attribute	Tr
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"	Tr
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"	Tr
$(":button")	Selects all <button> elements and <input> elements of type="button"	Tr
$("tr:even")	Selects all even <tr> elements	Tr
$("tr:odd")	Selects all odd <tr> elements	Tr

jQuery - events

```
$("p").click(function(){
    $(this).hide();
});
```

```
$("#p1").hover(function(){
    alert("You entered p1!");
},
function(){
    alert("Bye! You now leave p1!");
});
```

jQuery - AJAX

```
$.ajax({
  method: "POST",
  url: "some.php",
  data: { name: "John", location: "Boston" }
})
  .done(function( msg ) {
    alert( "Data Saved: " + msg );
  });
```

```
$.ajax({
  url: "test.html",
  cache: false
})
  .done(function( html ) {
    $( "#results" ).append( html );
  });
```

Languages - TypeScript

- type annotations
- interfaces
- classes (now in ES6),
- Mixins
- any type, Generics
- Type inference,
- Namespaces and modules
- JSX

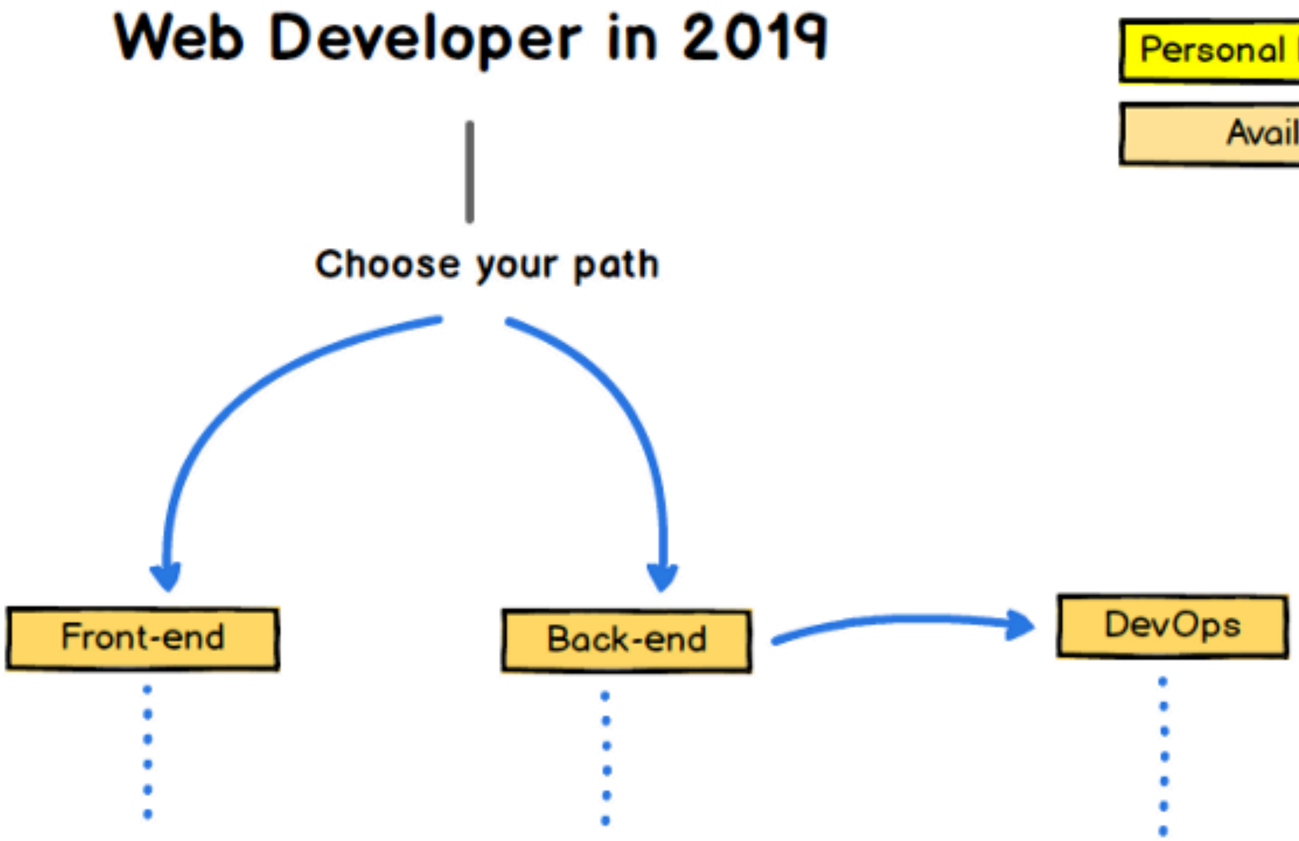
```
class Student {
    fullName: string;
    constructor(public firstName: string,
           public middleInitial: string,
           public lastName: string) {
        this.fullName = firstName + " " +
                        middleInitial + " " +
                        lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person : Person) {
    return "Hello, " +
           person.firstName + " " +
           person.lastName;
}

var user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);
```

Roadmap (again)

Tools for Internet Applications

Required for any path

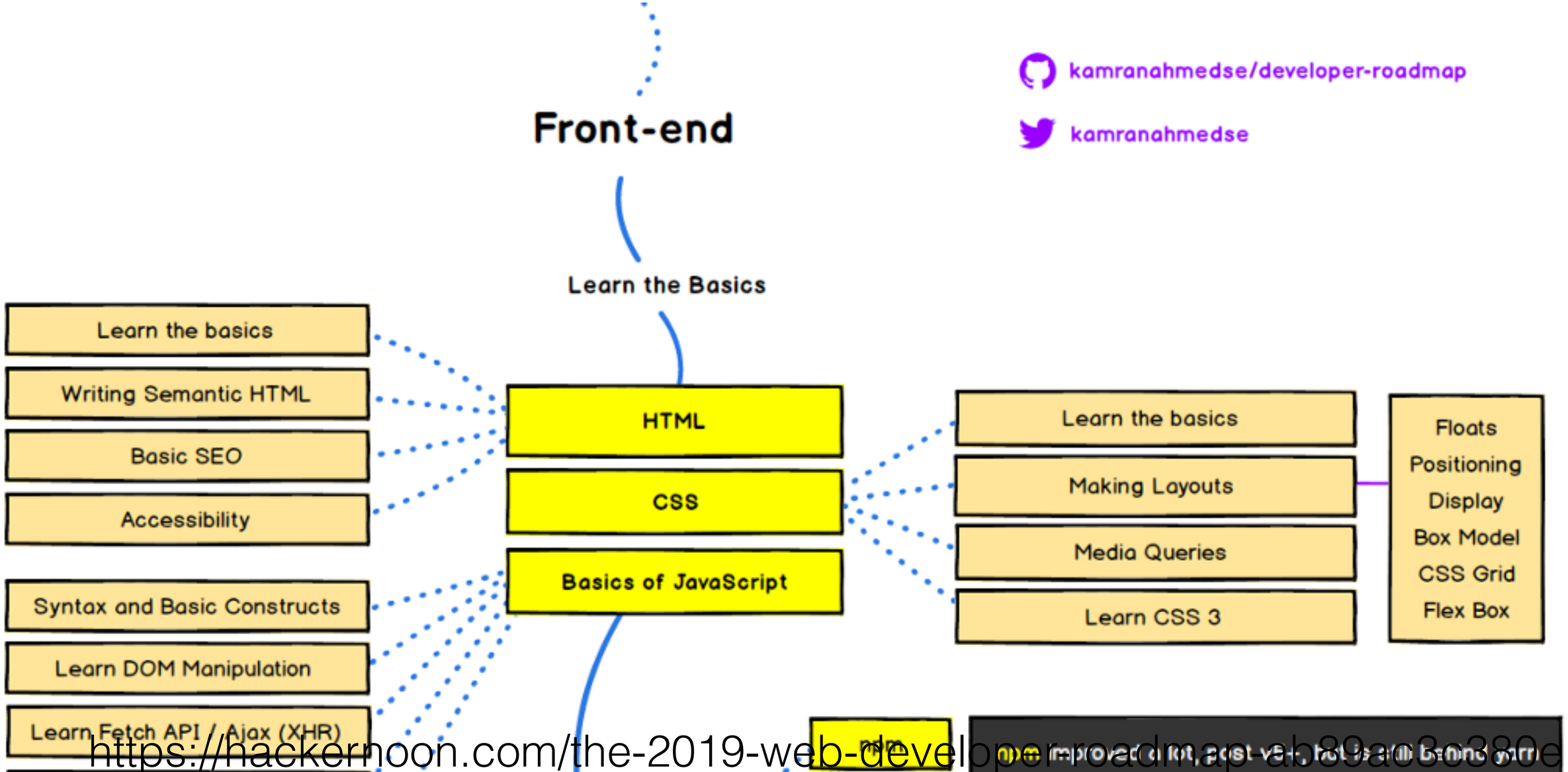
- Git - Version Control
- Basic Terminal Usage
- Data Structures & Algorithms
- SOLID, KISS, YAGNI
- GitHub
- Licenses
- Semantic Versioning
- SSH
- HTTP/HTTPS and APIs
- Design Patterns
- Character Encodings



Legends

- Personal Recommendation!
- Available Options

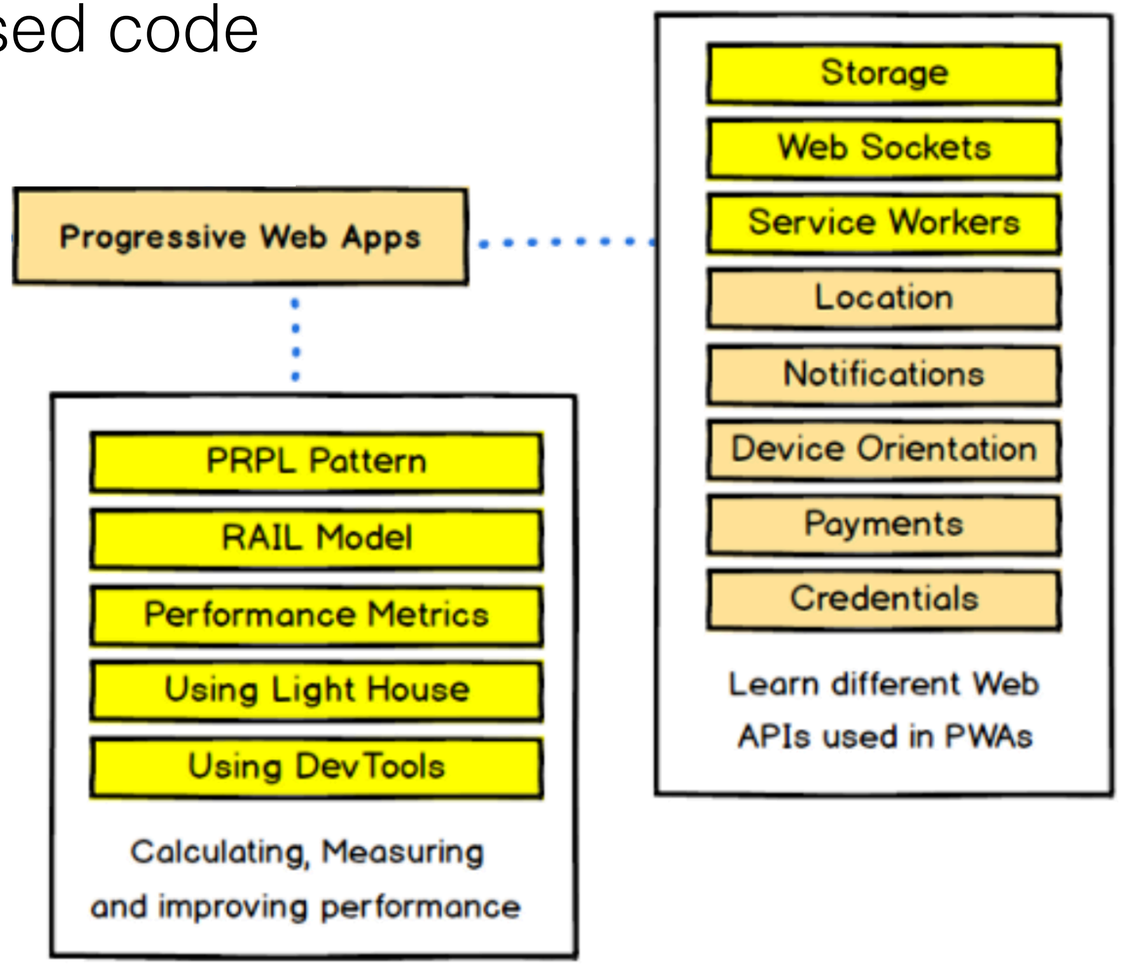
Programming Languages for Internet Applications



https://hackernoon.com/the-2019-web-developer-roadmap-ab89ac8c380e

Client-side frameworks (a case for React and Typescript)

- Client applications include many standardised code for many common situations
- Frameworks and libraries provide support for those patterns
  - separation of concerns, reuse of code and inversion of control
- Strongly typed languages provide a safe usage of libraries and frameworks



Internet Applications Design and Implementation

(Lecture 8 - Part 5 - Front-end Libraries)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Client Development (HTML5 - HTML, JS, CSS)

- Unobtrusive Javascript (and CSS)
  - Good and valid markup, easier to write without inline javascript
  - Loosely coupled structure, behaviour and presentation layers.
  - Name conventions and string based hooks hinder the development.
  - CSS lacks abstraction mechanisms, results in expanded and repetitive stylesheets
  - In practice, low code reuse and poor organisation.
  - Javascript is not friendly to manipulate DOM elements
  - Layouts are hard to get right and flexible in all devices and sizes

Development Tools

- **JQuery**

  Solves dynamic manipulation of DOM structures. Does not add much to the native event based model of javascript.

- **Bootstrap** (and many other CSS frameworks)

  Solves presentation and layout problems, still needs many string based conventions, tight coupling, which are hard to maintain.

- **Coffeescript**

  Simplifies the syntax of JavaScript. compiles down to Javascript.

- **Sass**

  Adds nested rules (less repetition), variables (abstraction, less literal repetition), mixins (composition/extension), file organization (imports). compiles to basic CSS.

Libraries - example: JQuery

- Provides shortcuts for the dynamic DOM manipulation
- Follows the *well-known* style of css selectors

```
$( selector ).action( arguments )
```

- Abstraction layer for DOM events

```
$(function() { do this or that })
```

- Functional style in event handlers and operations
- The "*de facto*" Javascript library

Libraries - example: JQuery

- Abstraction layer for AJAX requests

```
$.get(“/serverurl/path/?querystring”,
  function(data,status) {…})

$.post(“/serverurl/path/?querystring”,
  { arg1: val1, arg2: val2, … },
  function(data,status) {…})

$.ajax({type:”POST”, …
        success: function() {…},
        error: function() {…}})
```

Libraries - example: JQuery

```
$( selector ).action( arguments )
```

- Returns a list of (wrapped) elements matching `selector`
- Wraps all resulting elements with JQuery DOM objects.
- Chains operations on results (`fmap` functional style)

```
$("div p.cell").click(handleClick)
      .filter(".hidden")
      .addClass("done")
      .hide()

messages
  .map((msg)->$("ul.todo")
              .append($("li").text(msg))
```

Libraries - example: JQuery

- Still based on Javascript code patterns
- Chains of operations are ok, but wrappers are annoying…
- It's still "stringly typed" and error prone

Frameworks - NodeJS

- Cross-platform environment (server and client)
- Node provides a modular environment for Javascript (and a package manager, NPM)
- `require('fs')`,
- `exports.a = 1`
- `import` is part of ES2015, but… not really in browsers and node 6, but only using transpilers (babel) or bundlers (webpack).

Library - Bootstrap (CSS, HTML, and JS)

- Based on JQuery, developed using Sass
- Distributed in compiled and minified files
- Abstracts browser differences
- Responsive design from scratch, mobile centred
- Uniform look (the modern default)
- Grid system that makes layout construction easy and adaptable
- Javascript plugins (e.g. modals) that include behaviour and custom events.
- Configuration using Sass & NPM build scripts

Frameworks - Babel

- Babel is a javascript compiler
- Provides tools to build new languages and DSLs over javascript (and generate javascript target code)
- Solves the issues of ES6 and Node.JS on module loading time, by translating them to `require` operations.

Languages - Coffeescript

- compiles down to Javascript.
- based on npm and babel.
- Simplifies the syntax of JavaScript.

```
square = (x) -> x * x
cube   = (x) -> square(x) * x
```

```
var cube, square;

square = function(x) {
  return x * x;
};

cube = function(x) {
  return square(x) * x;
};
```

Languages - Coffeescript

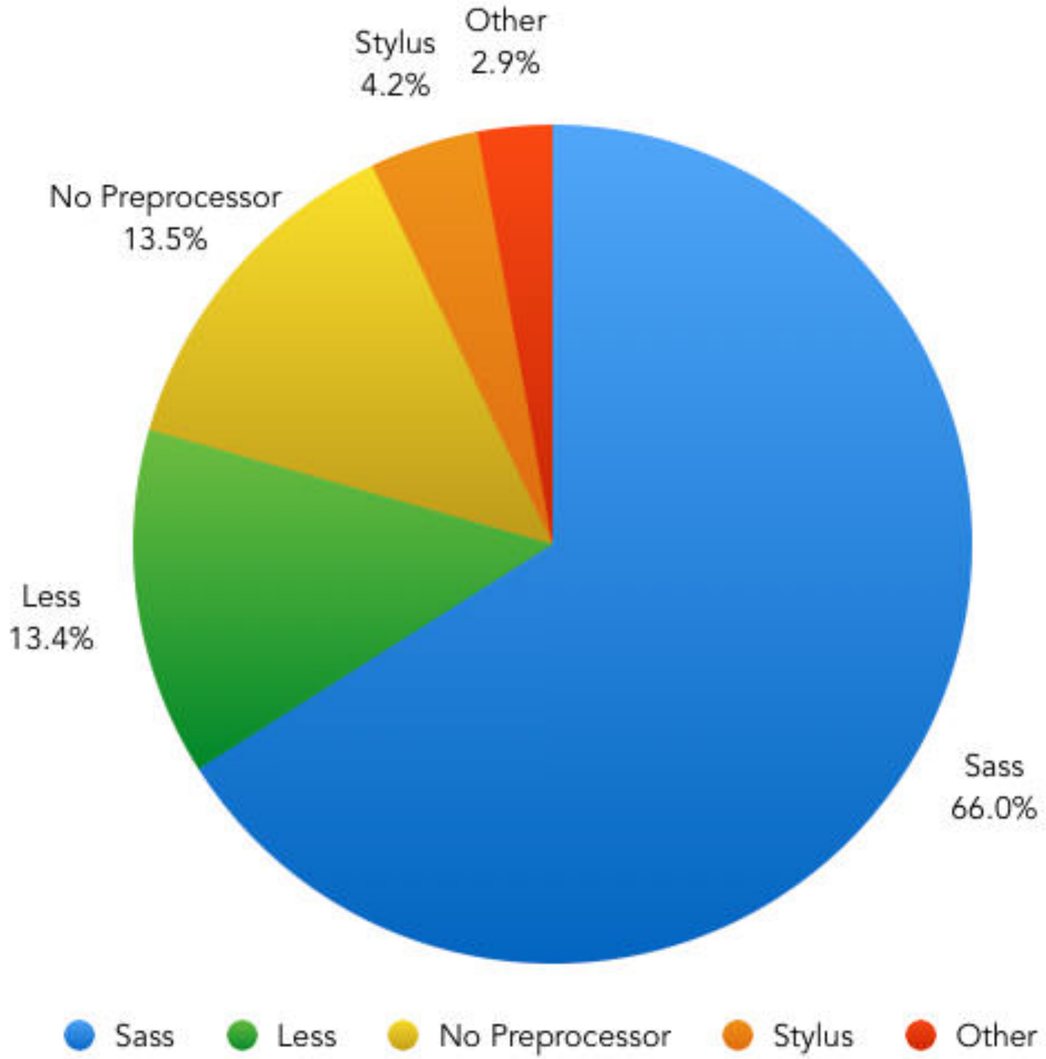
- compiles down to Javascript.
- based on npm and babel.
- Simplifies the syntax of JavaScript.

```
kids =
  brother:
    name: "Max"
    age:  11
  sister:
    name: "Ida"
    age:  9
```

```
kids = {
  brother: {
    name: "Max",
    age: 11
  },
  sister: {
    name: "Ida",
    age: 9
  }
};
```

Languages - Preprocessors: Sass, Less

- modularity, abstraction and extension mechanisms



https://www.keycdn.com/blog/sass-vs-less/

Client Development (HTML5 - HTML, JS, CSS)

- Unobtrusive Javascript (and CSS)
  - Good and valid markup, easier to write without inline javascript
  - Loosely coupled structure, behaviour and presentation layers.
  - Name conventions and string based hooks hinder the development.
  - ~~CSS lacks abstraction mechanisms, hence expanded and repetitive stylesheets are frequent.~~
  - ~~In practice, low code reuse and poor organisation.~~
  - ~~Javascript is not friendly to manipulate DOM elements~~
  - ~~Layouts are hard to get right and flexible in all devices and sizes~~

Frameworks - AngularJS

- Two way binding between view and model
- Reactive, valid HTML with custom attributes

```
<!doctype html>
<html ng-app="todoApp">
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>
    <script src="todo.js"></script>
    <link rel="stylesheet" href="todo.css">
  </head>
  <body>
    <h2>Todo</h2>
    <div ng-controller="TodoListController as todoList">
      <span>{{todoList.remaining()}} of {{todoList.todos.length}} remaining</span>
      [ <a href="" ng-click="todoList.archive()">archive</a> ]
      <ul class="unstyled">
        <li ng-repeat="todo in todoList.todos">
          <label class="checkbox">
            <input type="checkbox" ng-model="todo.done">
            <span class="done-{{todo.done}}">{{todo.text}}</span>
          </label>
        </li>
      </ul>
      <form ng-submit="todoList.addTodo()">
        <input type="text" ng-model="todoList.todoText"  size="30"
               placeholder="add new todo here">
        <input class="btn-primary" type="submit" value="add">
      </form>
    </div>
  </body>
</html>
```

Frameworks - React

- Structure building of UI
  - Components
  - Parameters
  - State
- Reactive refreshing
- JSX syntax
- Routing
- State management with Redux/MobX

```
class TodoApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = { items: [], text: '' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  render() {
    return (
      <div>
        <h3>TODO</h3>
        <TodoList items={this.state.items} />
        <form onSubmit={this.handleSubmit}>
          <input
            onChange={this.handleChange}
            value={this.state.text}
          />
          <button>
            Add #{this.state.items.length + 1}
          </button>
        </form>
      </div>
    );
  }
  handleChange(e) {
    this.setState({ text: e.target.value });
  }
  handleSubmit(e) {
    e.preventDefault();
    const newItem = {
      text: this.state.text,
      id: Date.now()
    };
    this.setState((prevState) => ({
      items: prevState.items.concat(newItem),
      text: ''
    }));
  }
}
class TodoList extends React.Component {
  render() {
    return (
      <ul>
        {this.props.items.map(item => (
          <li key={item.id}>{item.text}</li>
        ))}
      </ul>
    );
  }
}
ReactDOM.render(<TodoApp />, mountNode);
```

Languages - TypeScript

- type annotations
- interfaces
- classes (now in ES6),
- Mixins
- any type, Generics
- Type inference,
- Namespaces and modules
- JSX

```
class Student {
    fullName: string;
    constructor(public firstName: string,
           public middleInitial: string,
           public lastName: string) {
        this.fullName = firstName + " " +
                        middleInitial + " " +
                        lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person : Person) {
    return "Hello, " +
           person.firstName + " " +
           person.lastName;
}

var user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);
```

Internet Applications Design and Implementation

(Lecture 8 - Part 6 - React)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

https://facebook.github.io/react/

React is component-based

- The user interface is a composition of predefined building blocks
- It allows to build encapsulated components that manage their own state
- Components can be assembled/composed to make more elaborate UIs
- Hierarchical structures are easily instantiated in many different contexts

React is declarative, reactive and responsive

- Component logic is written in JavaScript instead of HTML templates
- Design simple views for each state in the app
- Combine and configure the active view using simple dynamic decisions
- React efficiently updates and renders components when data changes

The Key Ingredients

React components

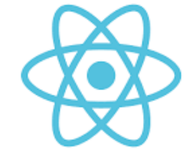
- React components are classes that implement a `render()` method that **takes input data** and **returns what to display**
- The next example uses an XML-like syntax called JSX to represent html elements
- Input data that is passed hierarchically to the component can be accessed by `render()` via `this.props`
- JSX is optional and is not required to be used in React.

A React application is a Node Application

npx create-react-app *clock* --template typescript

```
class HelloMessage extends React.Component<{name:String}> {
    render() {
        return <div>Hello {this.props.name}</div>;
    }
}

ReactDOM.render(<HelloMessage name="Jane" />,
                document.getElementById("root"));
```

A react component with state



- A component can maintain internal state
- Accessed via `this.state` and changed by `setState()`
- When a component's state data changes, the rendered markup will be updated by re-invoking `render()`

```
interface TimerInterface { secondsElapsed: number }

class Timer extends React.Component<{},TimerInterface> {
 …
    render() {
        return (
            <div>
               Seconds elapsed since you arrived: {this.state.secondsElapsed}
            </div>
        )
    }
}

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

```
interface TimerInterface { secondsElapsed: number }

class Timer extends React.Component<{},TimerInterface> {

    constructor(props:any) {
        super(props)
        this.state = { secondsElapsed: 0 };
    }
    …
    render() {
        return (<div>
                    Seconds elapsed since you arrived: {this.state.secondsElapsed}
                </div>)
    }
}

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

initial state

getter

```
class Timer extends React.Component<{},TimerInterface> {
    interval:any

    constructor(props:any) {
        super(props)
        this.state = { secondsElapsed: 0 };
        this.interval = null;
    }

    tick() { this.setState(({secondsElapsed}) => {secondsElapsed: secondsElapsed + 1}) }

    componentDidMount() { this.interval = setInterval(() => this.tick(), 1000); }

    componentWillUnmount() { clearInterval(this.interval); }

    render() {…}
}
```

```
interface TimerInterface { secondsElapsed: number }

class Timer extends React.Component<{},TimerInterface> {
    interval:any

    constructor(props:any) {
        super(props)
        this.state = { secondsElapsed: 0 };
        this.interval = null;
    }

    tick() { this.setState(({secondsElapsed}) => {secondsElapsed: secondsElapsed + 1}) }

    componentDidMount() { this.interval = setInterval(() => this.tick(), 1000); }

    componentWillUnmount() { clearInterval(this.interval); }

    render() {
        return (<div>
                    Seconds elapsed since you arrived: {this.state.secondsElapsed}
                </div>)
    }
}

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

React components

- React components can be defined as **stateless components**, functions that implement the `render()` function directly and **take input data** and **return what to display**

- Input data that is passed into the component can be accessed by `render()` via parameters for `props`

- Light-weight components are faster to render and take less memory

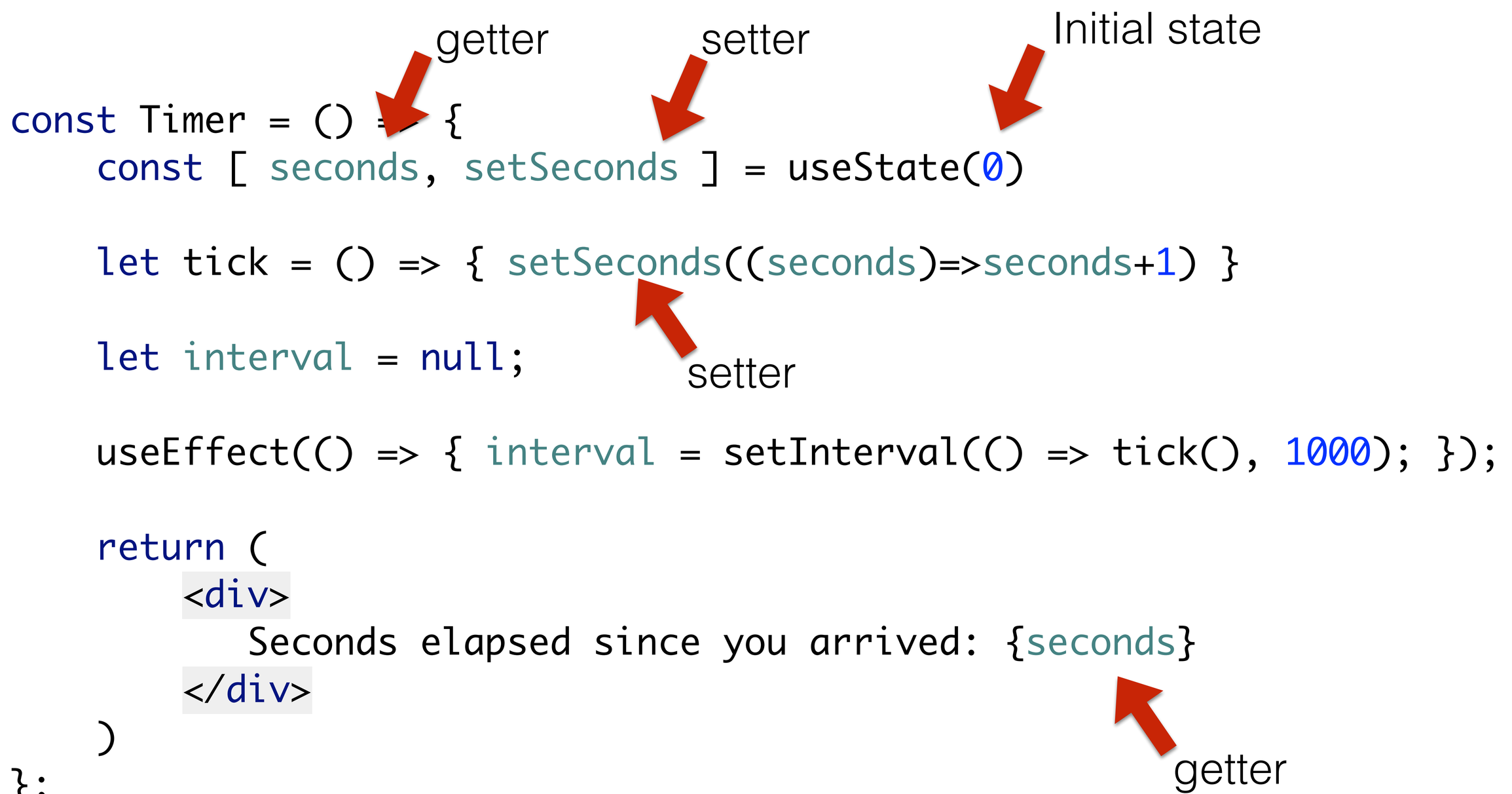
```
const HelloMessage =
  (props:{name:string}) => <div>Hello {props.name}</div>;

ReactDOM.render(<HelloMessage name="Jane" />,
                document.getElementById("root"));
```

A react component with state

- A component can maintain internal state data
- Accessed via “hooks” (`useState`)
- When a component's state data changes, the rendered markup will be updated by re-evaluating the stateless component function.
- Other effects are achieved with other (specialised) hooks (`useEffect`)

https://react-hooks-cheatsheet.com/useeffect

```
const Timer = () => {
    const [ seconds, setSeconds ] = useState(0)

    let tick = () => { setSeconds((seconds) => seconds+1) }

    let interval = null;

    useEffect(() => { interval = setInterval(() => tick(), 1000); });

    return (
        <div>
            Seconds elapsed since you arrived: {seconds}
        </div>
    )
};

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

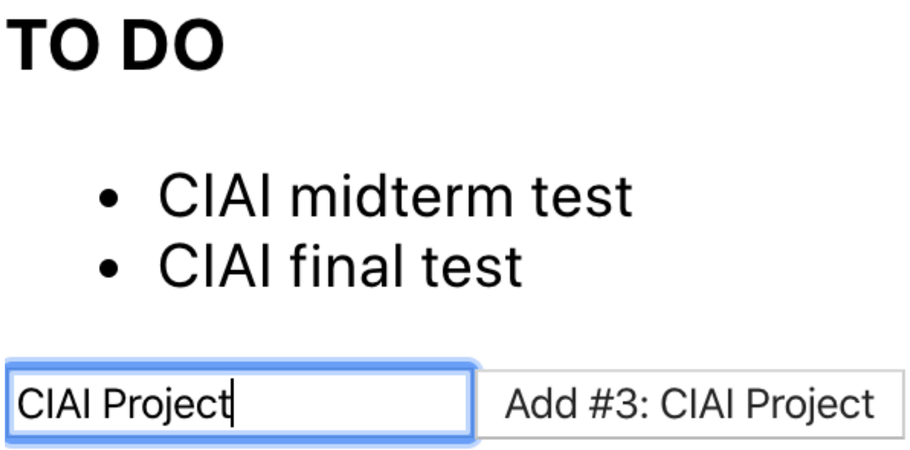

getter
setter
Initial state
const Timer = () => {
const [seconds, setSeconds] = useState(0)
let tick = () => { setSeconds((seconds)=>seconds+1) }
setter
let interval = null;
useEffect(() => { interval = setInterval(() => tick(), 1000); });
return (

Seconds elapsed since you arrived: {seconds}

)
};
getter
ReactDOM.render(<Timer />, document.getElementById("react_content"));

A complete example

Example: A complete reactive TODO app



Example: A complete reactive TODO app

- How to render a list of items?
- Use a stateless component that receives the list (as props) and renders it:

```
interface Item { id:number, text:string }

const TodoList = (props:{items:Item[]}) => {
    return (
        <ul>
            {props.items.map(item => (
                <li key={item.id}>{item.text}</li>
            ))}
        </ul>
    );
};
```

Example: A complete reactive TODO app

- Where to store the list of items? and the text of new tasks?

```
interface ToDoState { items:Item[], text:string }

class ToDo extends React.Component<{},ToDoState> {

    constructor(props:{}) {
        super(props);
        this.state = { items: [], text:"" }
    }
…
```

Example: A complete reactive TODO app

- How to render the list of tasks, and accept new ones? (with a list and a form)

```
class ToDo extends React.Component<{},ToDoState> {
  …
  render() {
        return (
            <div>
                <h3>TO DO</h3>
                <TodoList items={this.state.items}/>
                <form onSubmit={this.handleSubmit}>
                    <input onChange={this.handleChange} value={this.state.text}/>
                    <button> Add </button>
                </form>
            </div>
        );
    }
```

In this case, every time the item text changes, the button label also changes

Example: A complete reactive TODO app

- How to react to user interaction events? using event handlers that change the state…

```
class ToDo extends React.Component<{},ToDoState> {
  …
  handleSubmit = (e:any) => {
    e.preventDefault();
    var newItem = { text: this.state.text, id: Date.now() };
    this.setState((prevState) => ({
        items: [...prevState.items, newItem],
        text: “”
    }));
  };

  handleChange = (e:any) => { this.setState({text: e.target.value}); };
  …
```

In this case, every time the item text changes, the button label also changes

with hooks

```
const ToDo = () => {

    const [ state, setState ] = useState({ items: [] as Item[], text:"" });

    let handleSubmit = (e:any) => {
        e.preventDefault();
        let newItem = { text: state.text, id: Date.now() }
        setState({ items: [...state.items, newItem], text: '' });
    };

    let handleChange = (e:any) => { setState({...state, text: e.target.value}); };

    return (<div> <h3>TO DO</h3>
                <TodoList items={state.items}/>
               <form onSubmit={handleSubmit}>
                    <input onChange={handleChange} value={state.text}/>
                    <button>
                        {'Add #' + (state.items.length + 1) + ': ' + state.text}
                    </button>
               </form>
            </div>);
};
```

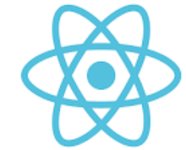
Ownership of components

- In React, an owner is the component that sets the **`props`** of other components
- More formally, if a component X is created in component Y's render() method, it is said that X is owned by Y
- Note that a component cannot mutate its **`props`**
- **`props`** are always consistent with what its owner sets them to
- This fundamental invariant leads to UIs that are guaranteed to be consistent
- In the todo example, TodoApp is the owner of TodoList

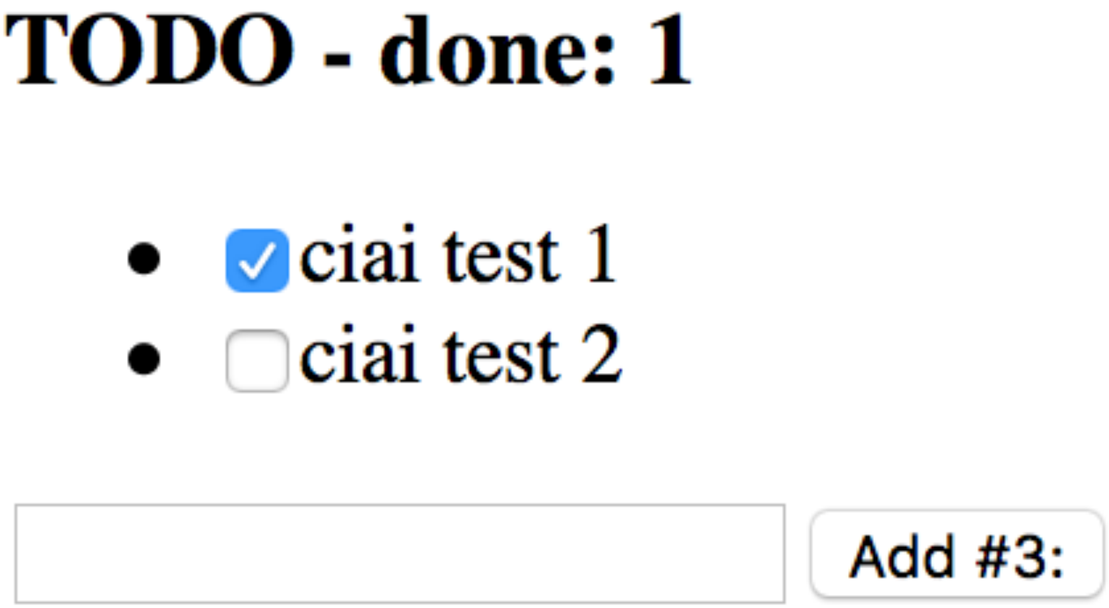
Ownership vs. Parenting

- There is a difference between the **owner-ownee** relationship and the **parent-child** relationship
- The owner-ownee relationship is specific to React
- The parent-child relationship is simply the one you know from the DOM
- In the todo example TodoApp is the **owner** of h3, and TodoList and form is the **parent** of input

Inverse data flow or Two-way binding



- Components can only update their own state
- Ownees can only call owners update functions
- So how to do this?



```
const TodoList = (props:{items:Item[], completeItem:(value:boolean)=>void }) => {

    let check = (e:any) => {
        if(e.target.checked)
            props.completeItem(true);
        else
            props.completeItem(false);
    };

    return (
        <ul>
            {props.items.map(item => (
                <li key={item.id}>
                    <input type="checkbox" key={"done"+item.id} onClick={check} />
                    {item.text}
                </li>
            ))}
        </ul>
    );
};
```

```
const ToDo = () => {
    const [ state, setState ] = useState({ items: [] as Item[], text:"", done:0 });

    let handleSubmit = …

    let handleChange = …

    let completeItem = (value:boolean) => {
        if(value) setState({...state, done:state.done+1 });
        else      setState({...state, done:state.done-1 });
    };

    return (<div>
                <h3>TO DO</h3>
                <TodoList items={state.items} completeItem={completeItem}/>
                <form onSubmit={handleSubmit}>
                    <input onChange={handleChange} value={state.text}/>
                    <button>
                        {'Add #' + (state.items.length + 1) + ': ' + state.text}
                    </button>
                    <p>{ "Tasks done "+ state.done }</p>
                </form>
            </div>);
};
```

Routing - setup

- In the React application:

```
var Link = ReactRouter.Link;
var Router = ReactRouter.Router;
var Route = ReactRouter.Route;
```

https://css-tricks.com/learning-react-router/

Routing

```
var HelloMessageRouted = React.createClass ({
   render: function() {
      return (<div>Hello {this.props.params.name}</div>)
   }});

var SomeComponent = React.createClass ({
   render: function() {
      return (…
     <Link to={`/timer`}>Click to go to timer app</Link>
   …)
   }});

ReactDOM.render((
      <Router>
         <Route path="/" component={TodoApp}/>
         <Route path="/timer" component={Timer}/>
         <Route path="/hello/:name" component={HelloMessage}/>
        </Router>
      ), document.getElementById('react_content'))
```

React Thinking

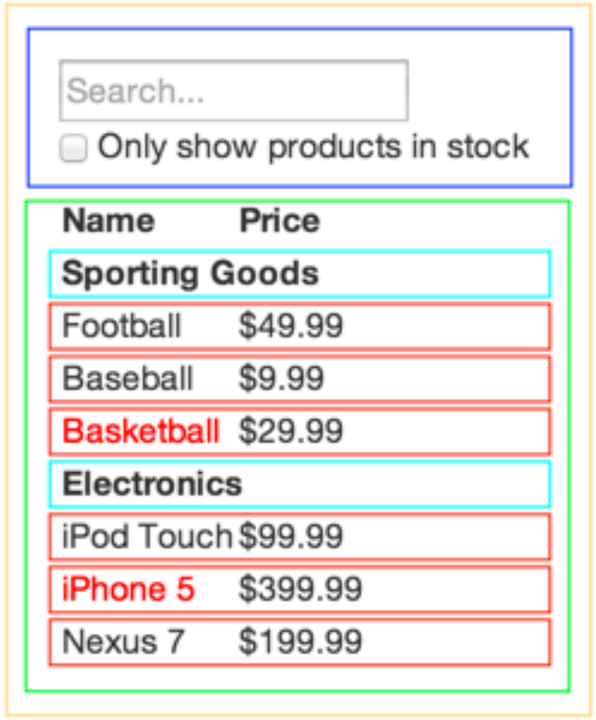
https://facebook.github.io/react/docs/thinking-in-react.html

Break the UI into a component hierarchy

Identify the UI minimal complete state

1. Is it passed in from a parent via props? If so, it probably isn't state.
2. Does it remain unchanged over time? If so, it probably isn't state.
3. Can you compute it based on any other state or props in your component? If so, it isn't state.

State example



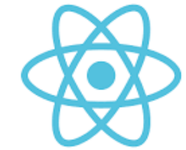
- Pieces of data:
  - The original list of products
  - The search text the user has entered
  - The value of the checkbox
  - The filtered list of products

- Actual state:
  - The search text the user has entered
  - The value of the checkbox

Where should the state live?

- Probably not in the place you think…
- Form component? Think again…
- React is all about **one-way** data flow **down the component hierarchy**
- It may not be immediately clear which component should own what state

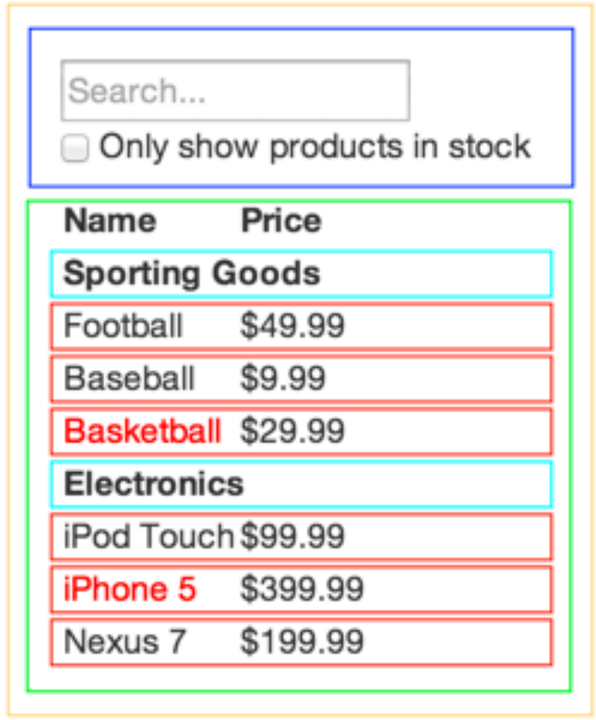
Find the state component by Q&A



For each piece of state in your application:

- Identify every component that renders something based on that state.
- Find a common owner component (a single component above all the components that need the state in the hierarchy).
- Either the common owner or another component higher up in the hierarchy should own the state.
- If you can't find a component where it makes sense to own the state, create a new component simply for holding the state and add it somewhere in the hierarchy above the common owner component.

So, where should the state be?



- Pieces of data:
  - The original list of products
  - The search text the user has entered
  - The value of the checkbox
  - The filtered list of products

- Actual state:
  - checkbox value: yellow component, complete app
  - search text: yellow component, complete app

Summary

- Create a component for each “part” of your app
- A component receives input (optional)
- And defines how it should be rendered (mandatory) in the **`render`** method
- Use **`this.props`** to access the inputs

Summary

- Optionally define a state to your component as a record (in `getInitialState`)
- Every time the component state changes the React runs the `render` method
- **Never** update the state directly!!
- Use `setState` to update it so Rect can properly propagate changes
- You can access the previous state just before updating it
- Just define as input for `setState` a function that receives the previous state and use it inside the function in any way you need

Summary

- Recall that React is all about one-way data flow
- Two make it two-way you need to explicitly state that in your code
- Owner components send a call-back function to ownees (through **`props`**) so they can update the owner state by calling the function

React and Bootstrap

Bootstrap and React

- Bootstrap provides css and javascript that can be linked to React output

```
import React from 'react';

const Example = () => {
    return (
        <div class="alert alert-danger alert-dismissible fade show" role="alert">
            <strong>Oh snap! You got an error!</strong>
            <p>
                Change this and that and try again.
            </p>
            <button type="button" class="close" data-dismiss="alert" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
    )
}
```

Bootstrap and React

- Bootstrap-react is a complete re-implementation of bootstrap to provide react components. Connection at the abstract level and not at the browser level.

```
import React, { Component } from 'react';
import Alert from 'react-bootstrap/Alert';

const Example = () => {
    return (
        <Alert dismissible variant="danger">
            <Alert.Heading>Oh snap! You got an error!</Alert.Heading>
            <p>
                Change this and that and try again.
            </p>
        </Alert>
    )
}
```

Bootstrap and React - also with state

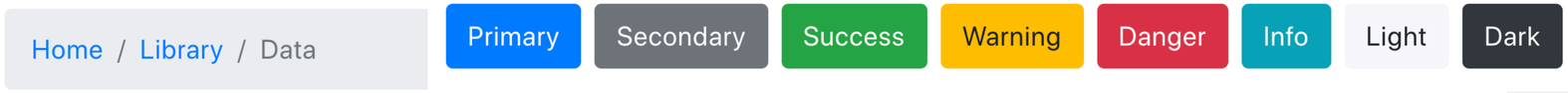
```
const AlertDismissible = () => {
    const [show, setShow] = useState(true);

    return (
        <> <Alert show={show} variant="success">
                <Alert.Heading>How's it going?!</Alert.Heading>
                <p>
                    Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget
                    lacinia odio sem nec elit. Cras mattis consectetur purus sit amet
                    fermentum.
                </p>
                <hr />
                <div className="d-flex justify-content-end">
                    <Button onClick={() => setShow(false)} variant="outline-success">
                        Close me ya'll!
                    </Button>
                </div>
            </Alert>
            {!show && <Button onClick={() => setShow(true)}>Show Alert</Button>}
        </>);}
```

Bootstrap and React

- Bootstrap-react provides the bootstrap grid system to be easily reused.

```
<Container>
    <Row>
        <Col>1 of 2</Col>
        <Col>2 of 2</Col>
    </Row>
    <Row>
        <Col>1 of 3</Col>
        <Col>2 of 3</Col>
        <Col>3 of 3</Col>
    </Row>
</Container>
```

- And all the components integrated in React




```
<Container>
    <Row>
        <Col xs={12} md={8}>
            xs=12 md=8
        </Col>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
    </Row>
    <Row>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
    </Row>
    <Row>
        <Col xs={6}>xs=6</Col>
        <Col xs={6}>xs=6</Col>
    </Row>
</Container>
```

Internet Applications Design and Implementation

(Lecture 8b - Modelling Interface & Interaction - React meets IFML)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

How do you specify interaction in a user interface?

How do you specify its views?

How do you specify navigation actions?

How do you specify the data needed?

From navigation? from server?

You don’t!

Most people use mocks only! or worse…

They just code ahead!

go to here

<form action=“jumptohere”…>…</form>

Outline

- Interaction Flow Modelling Language (IFML)
- IFML by example
- From IFML to React
- From User Stories to IFML to React

Internet Applications Design and Implementation

(Lecture 8b - Part 1 - IFML)

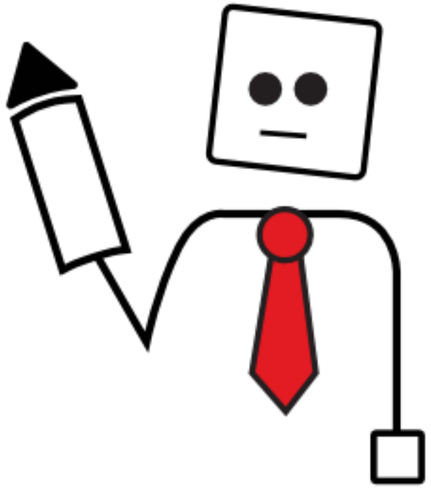
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

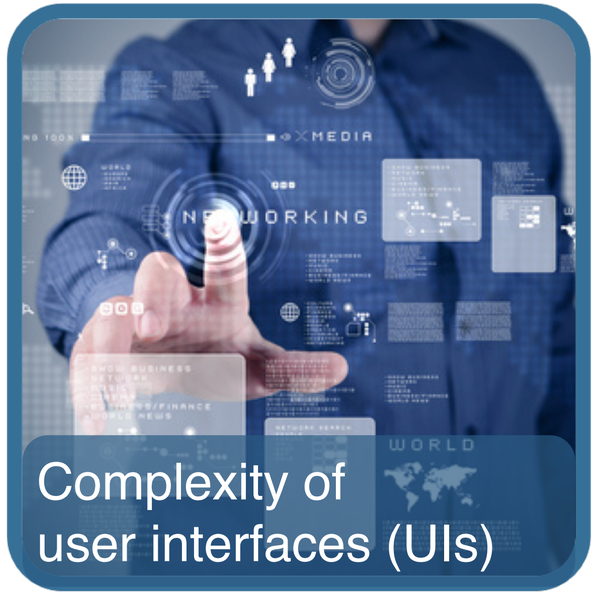
João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

The UI Design Problem

Costly and
Inefficient process





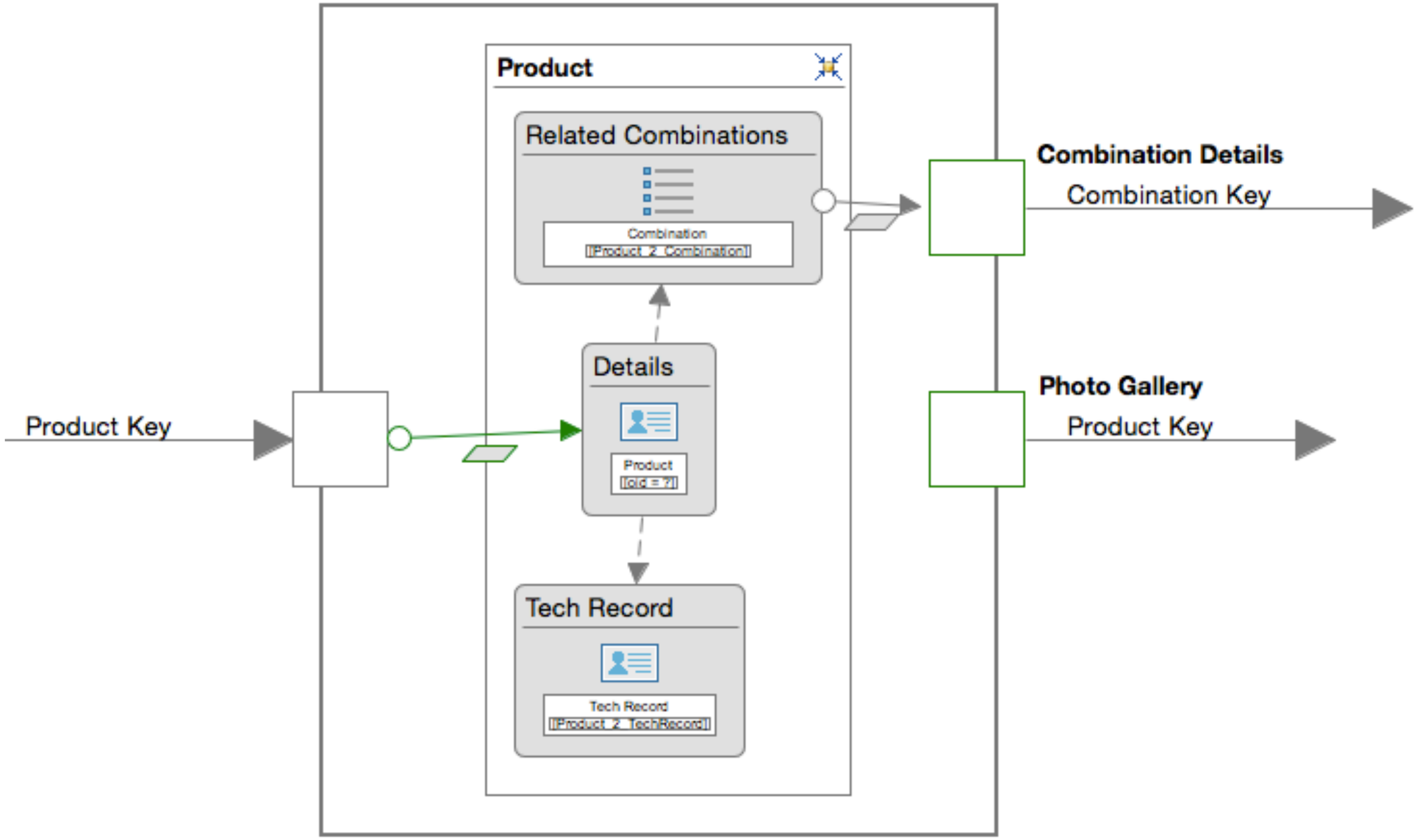
Complexity of
user interfaces (UIs)

Ineffective
tools

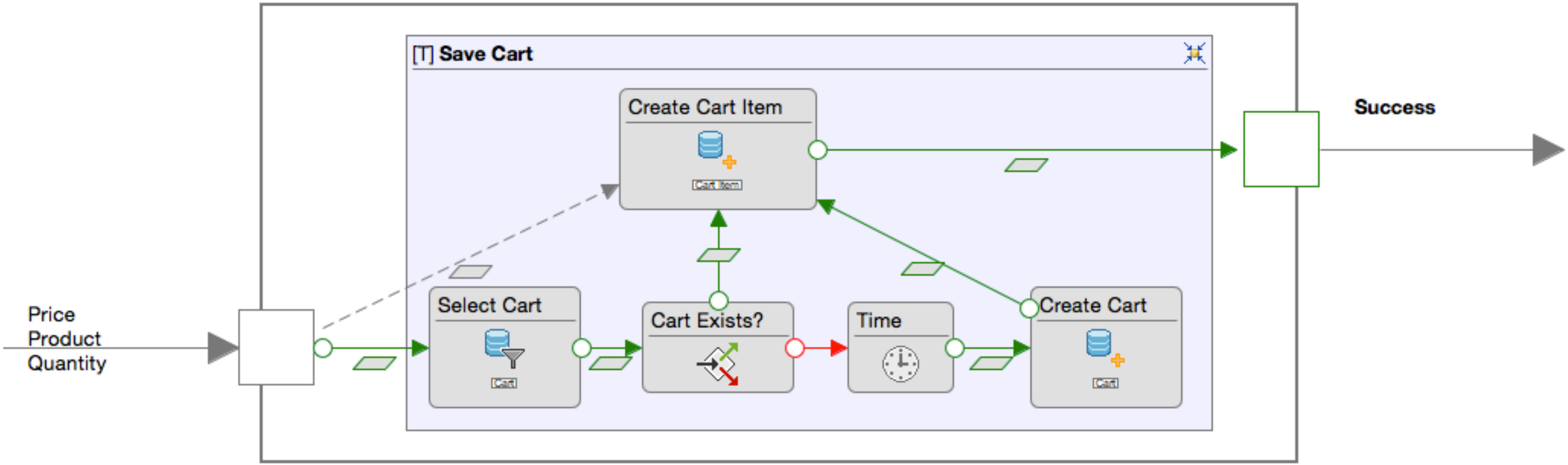
Manual
development

No MDE
technology

Solution: Abstraction of content and flow



Solution: Abstraction of content and flow



The Interaction Flow Modeling Language

The new OMG Standard for integrating the front-end design in your system and enterprise models

- OMG (Object Management Group) standard (since 03/2013)

- To express content, user interaction, and control behavior of front-end apps

Practical Results of Having a Standard

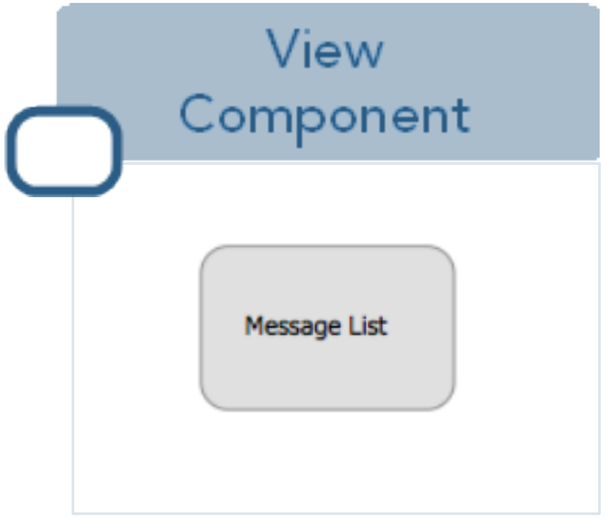
- An official metamodel/grammar of the language which describes the semantics of and relations between the modelling constructs
- A graphical concrete syntax for the interaction flow notation which provides an intuitive representation of the user interface composition, interaction and control logic for the front-end designer
- A UML Profile consistent to the metamodel
- An interchange format between tools using XMI
- All this, specified through standard notations themselves

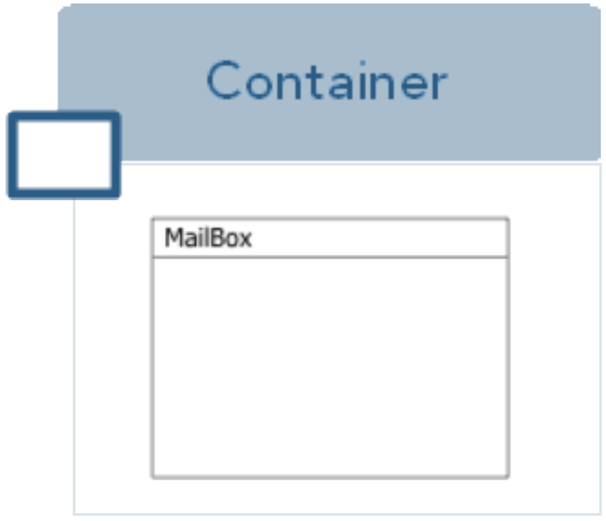
The UI Design solution: IFML

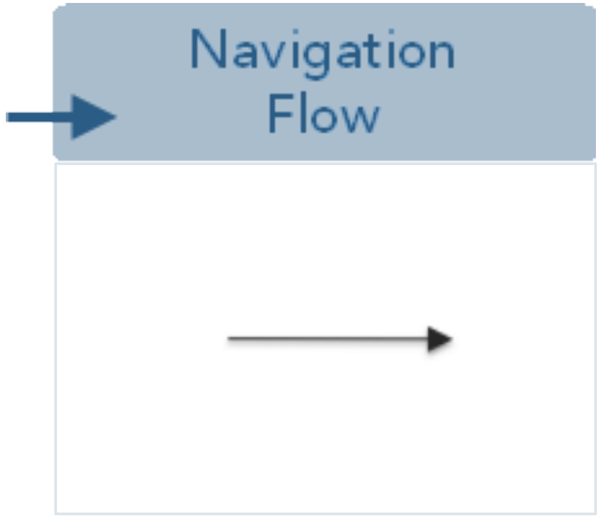
Covered aspects

- Multiple views for the same application
- Visualization and input of data, and production of events
- Components independent of concrete widgets and presentation
- Interaction flow, initiated by the user or by external events
- Modularization of the model (design-time containers for reuse purpose)
- User input validation and constraints, according to OCL or other existing constraint languages

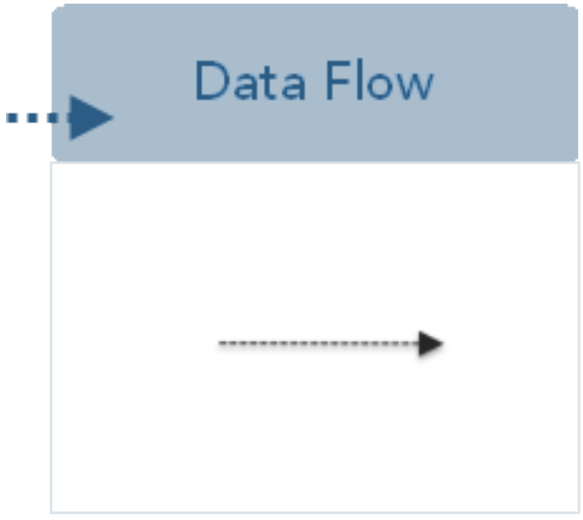
IFML Essentials

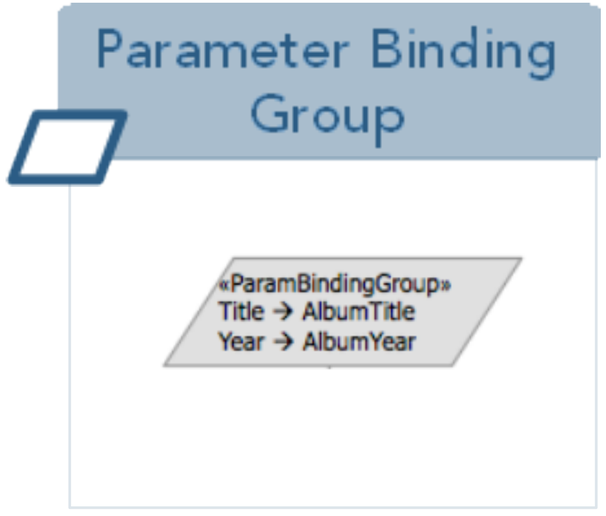


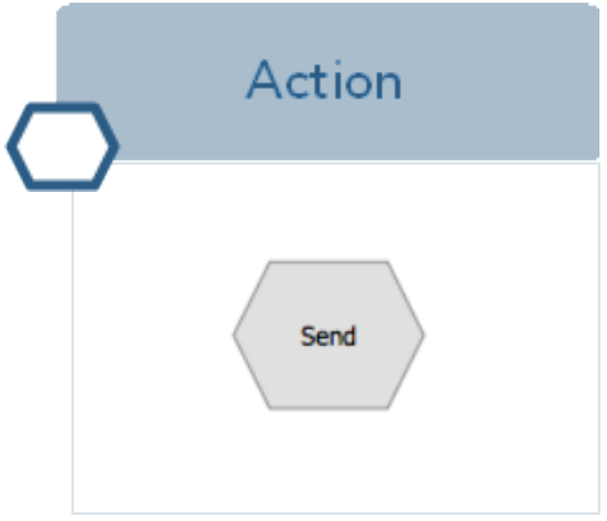




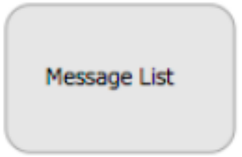
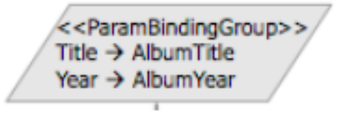
https://www.omg.org/spec/IFML/1.0/PDF

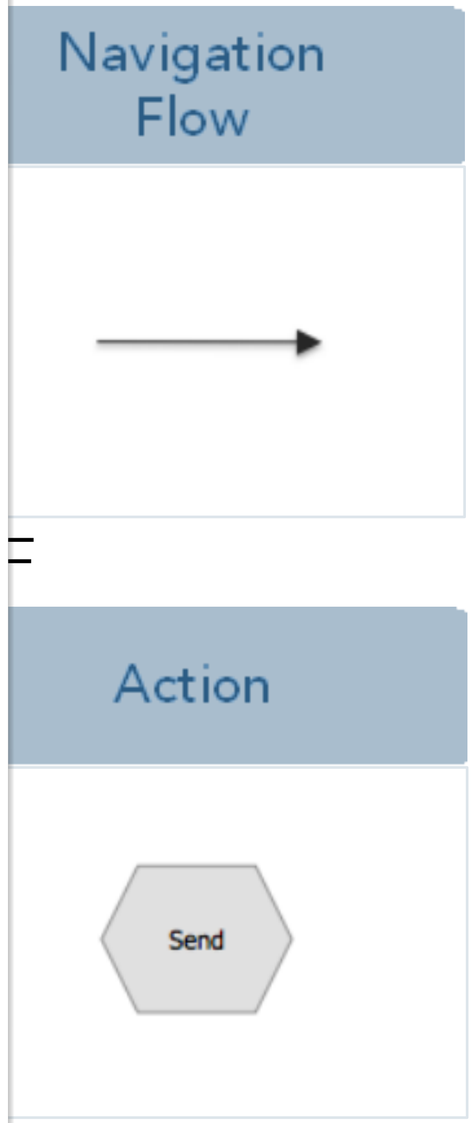






IFML Essenti

Concept	Meaning	IFML Notation	PSM Example
View Container	An element of the interface that comprises elements displaying content and supporting interaction and/or other ViewContainers.		Web page Window Pane.
View Component	An element of the interface that displays content or accepts input		An HTML list. A JavaScript image gallery. An input form.
Event	An occurrence that affects the state of the application		
Action	A piece of business logic triggered by an event		A database update. The sending of an email.
Navigation Flow	An input-output dependency. The source of the link has some output that is associated with the input of the target of the link		Sending and receiving of parameters in the HTTP request
Data Flow	Data passing between ViewComponents or Action as consequence of a previous user interaction.		
Parameter Binding Group	Set of ParameterBindings associated to an InteractionFlow (being it navigation or data flow)		



Internet Applications Design and Implementation

(Lecture 8b- Part 2 - IFML by Example)

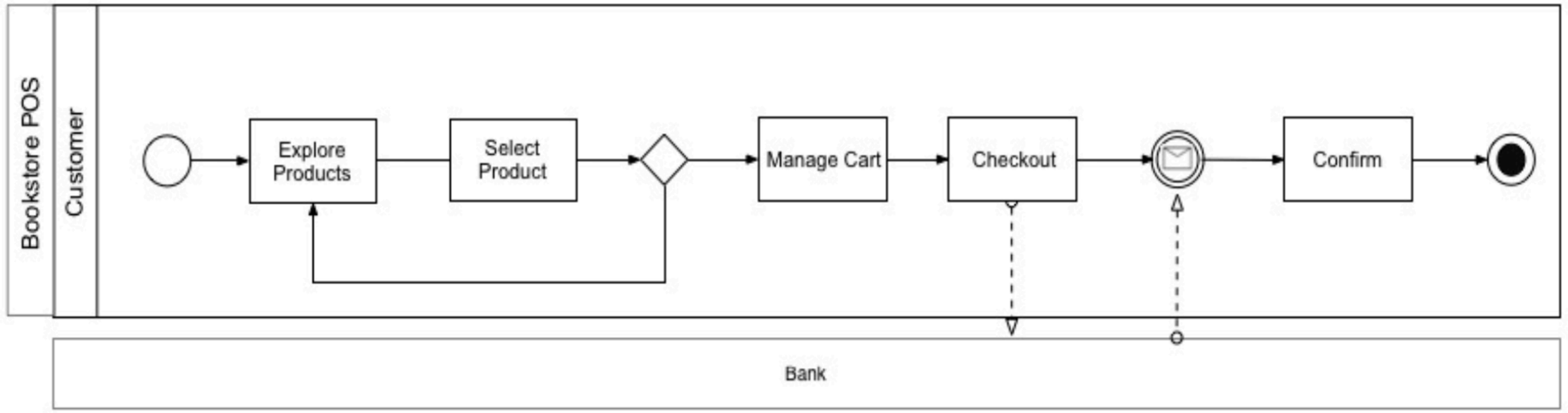
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

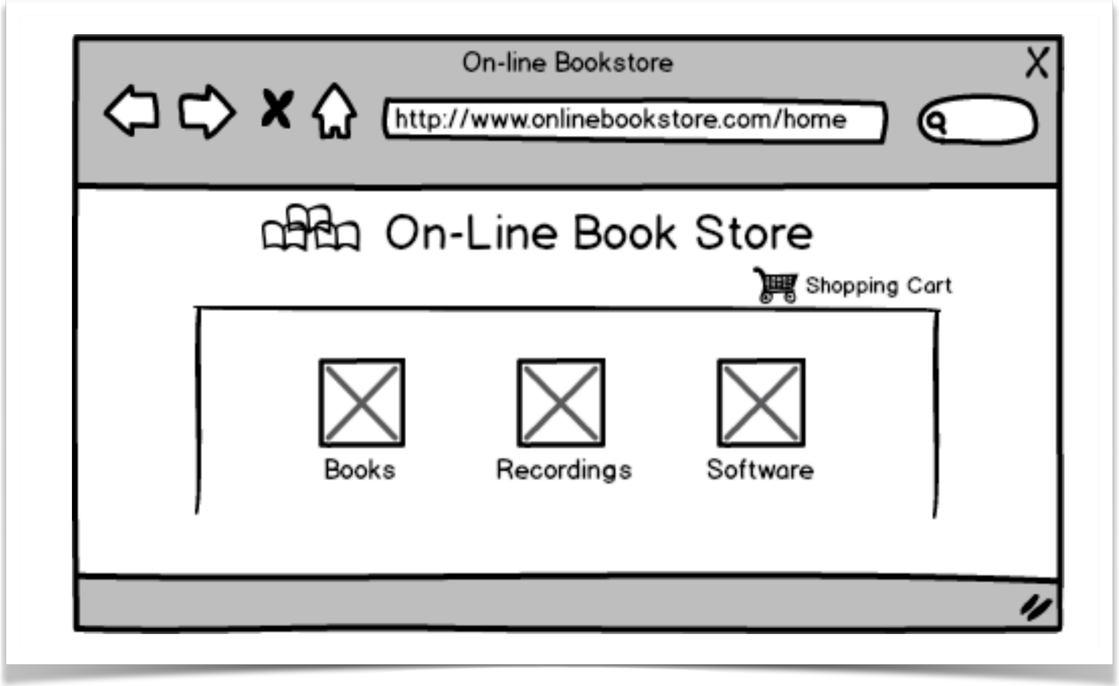
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

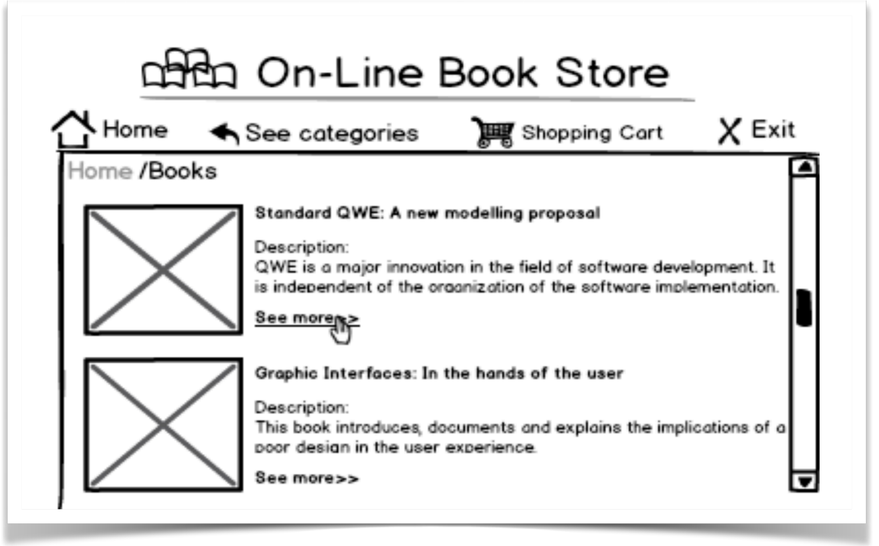
The design of UI usually starts with mockups (ifml.org)

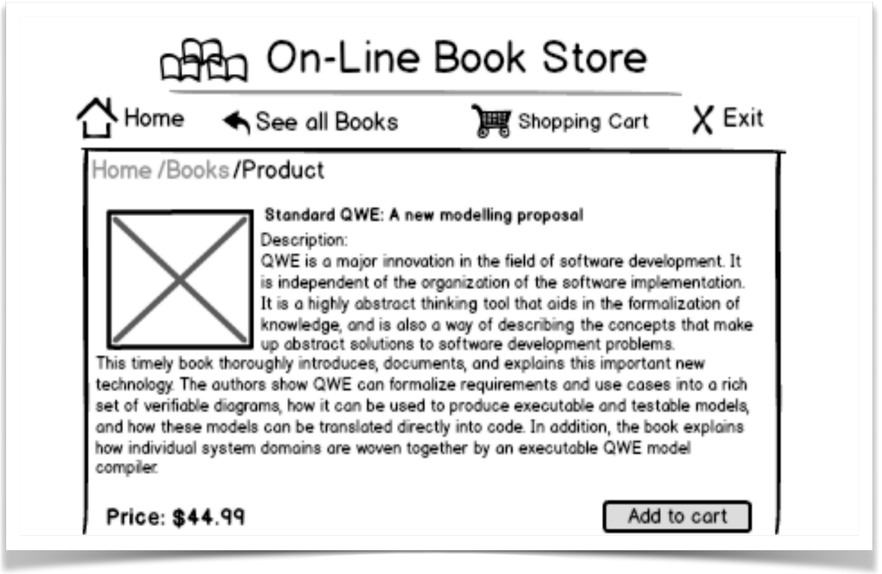
- IFML defines the behaviour and helps defining the components of UI parts.
- Start with a scenario of a book store and its shopping cart. The UI starts with a list of categories, products, and the corresponding details.
- The shopping process continues by asking user detailed information and connecting to the bank to process the payment.



IFML by Example



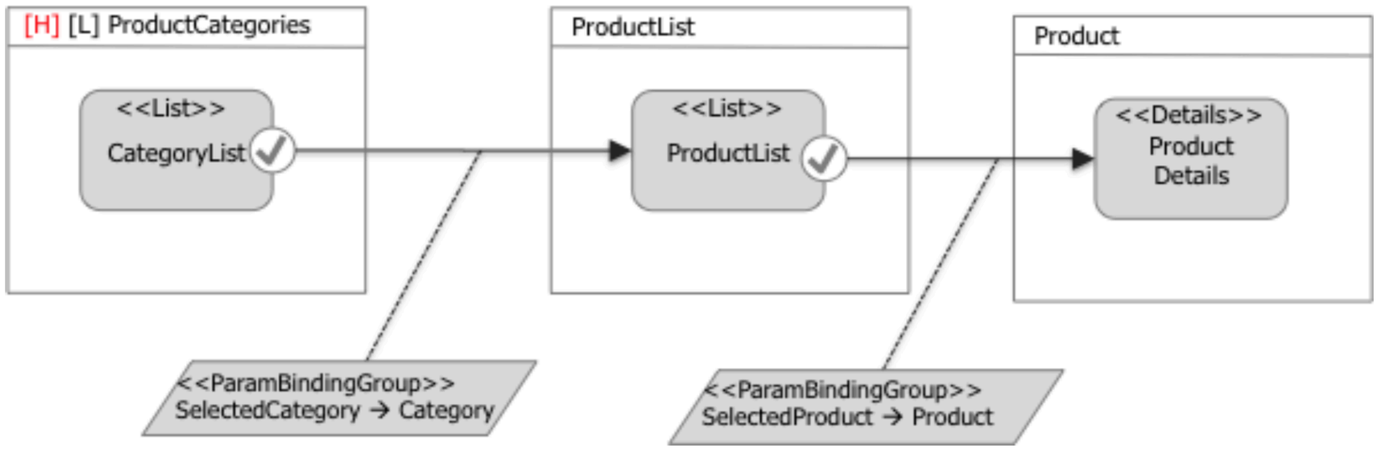




Consider the main screens of a sample application.
Only visual structure is defined.

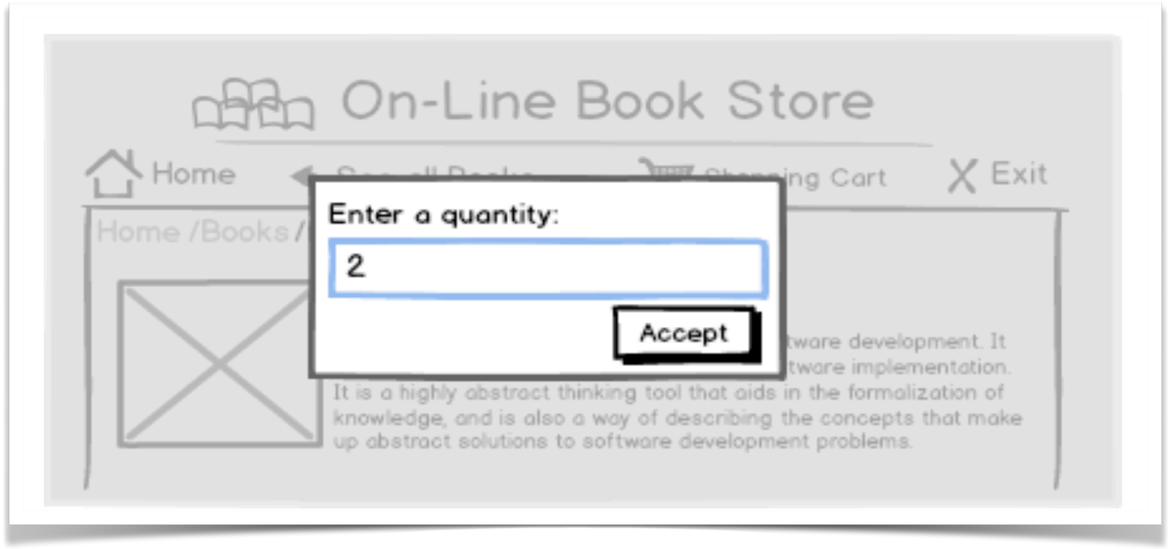
https://www.ifml.org/wp-content/uploads/IFML-Bookstore-Example.pdf

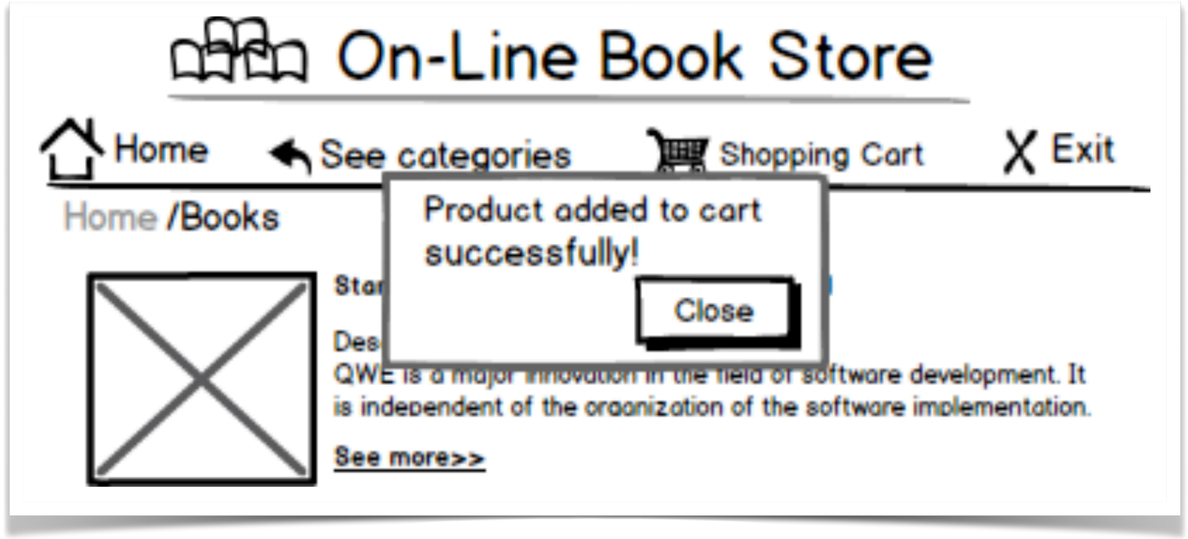
IFML navigation between pages



Flow between screens is defined using IFML. Data transmitted is also defined here.

Events, interactions, actions

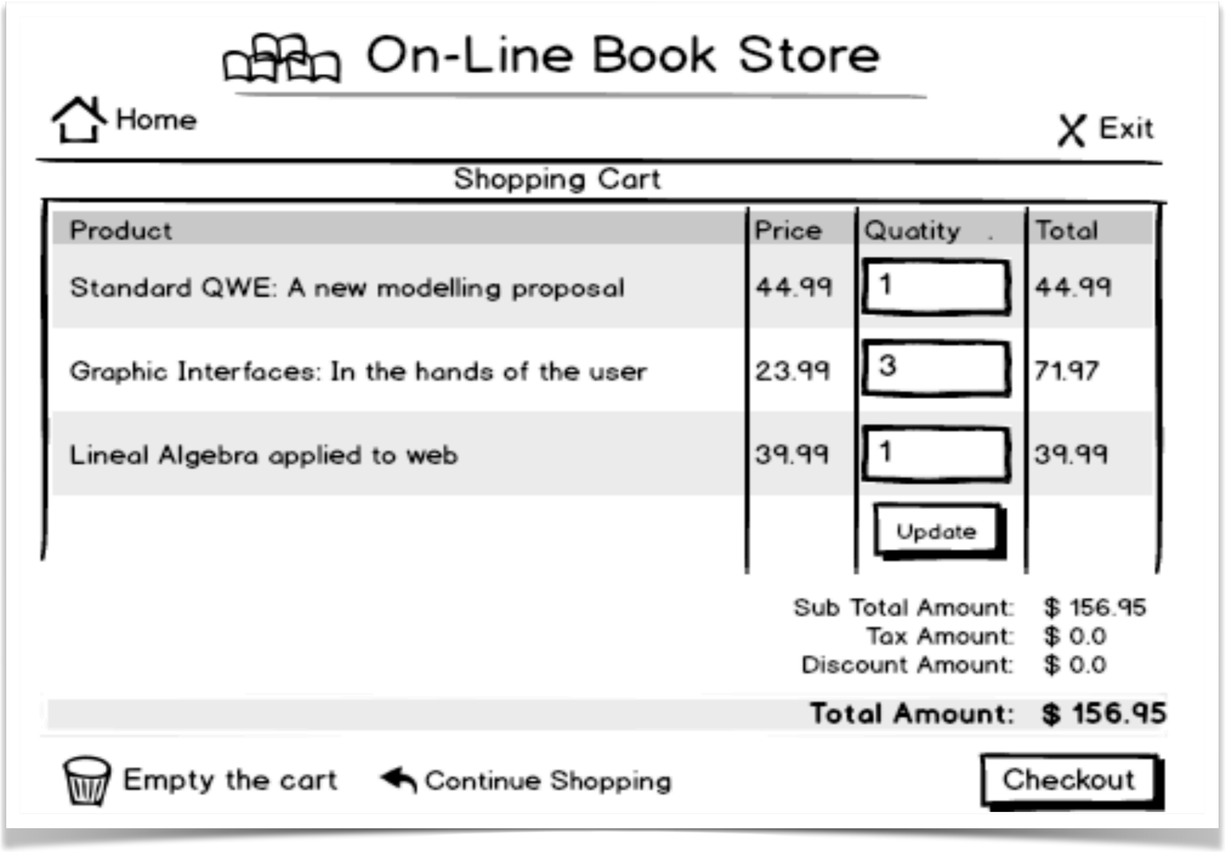




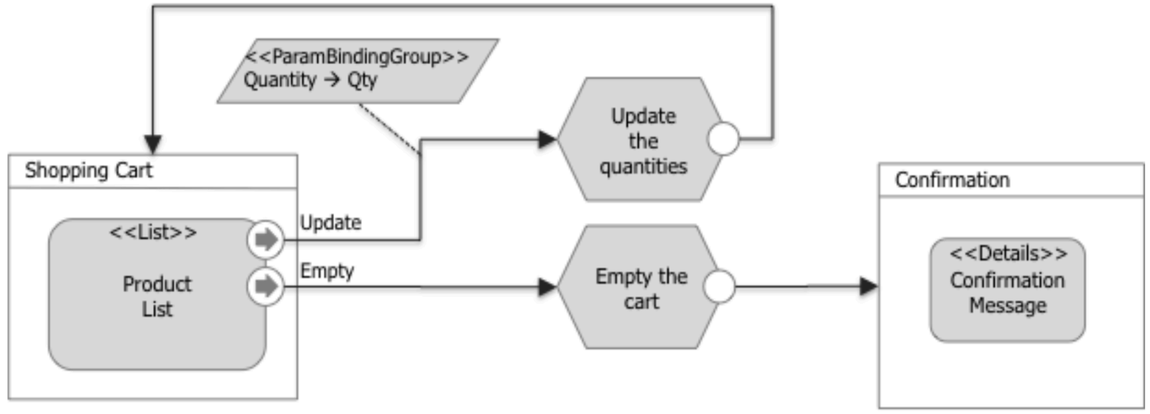
When a user “buys” a book

The event in the “Add to cart” button

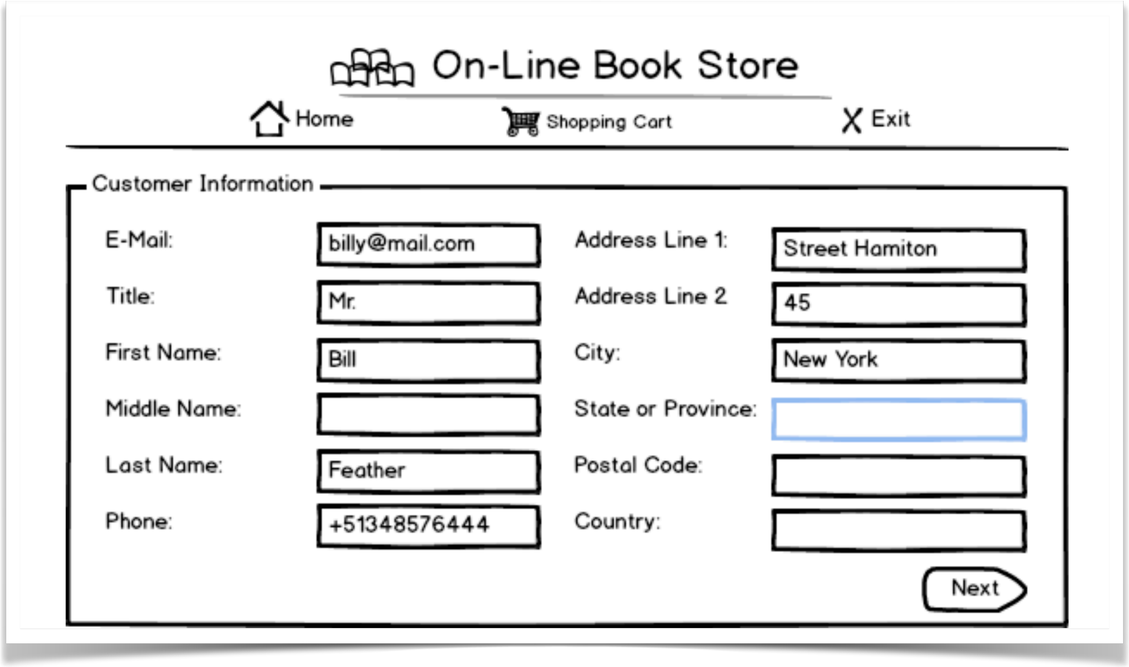
Events - Interactive interfaces

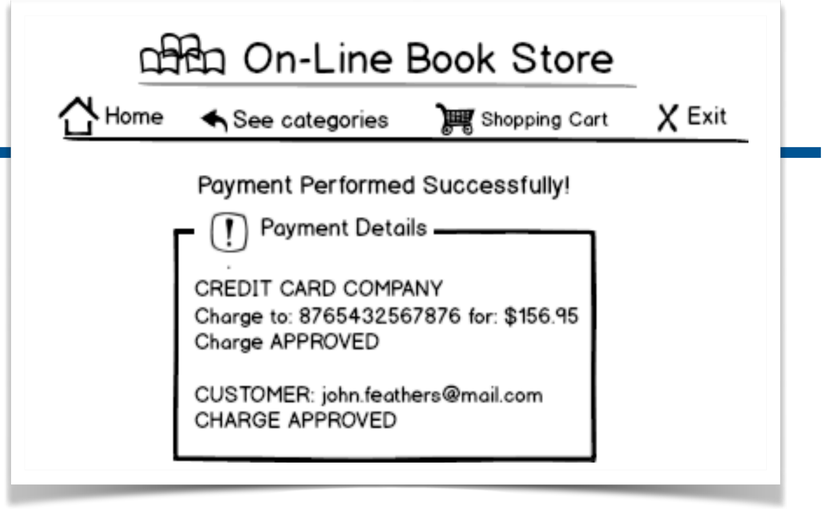


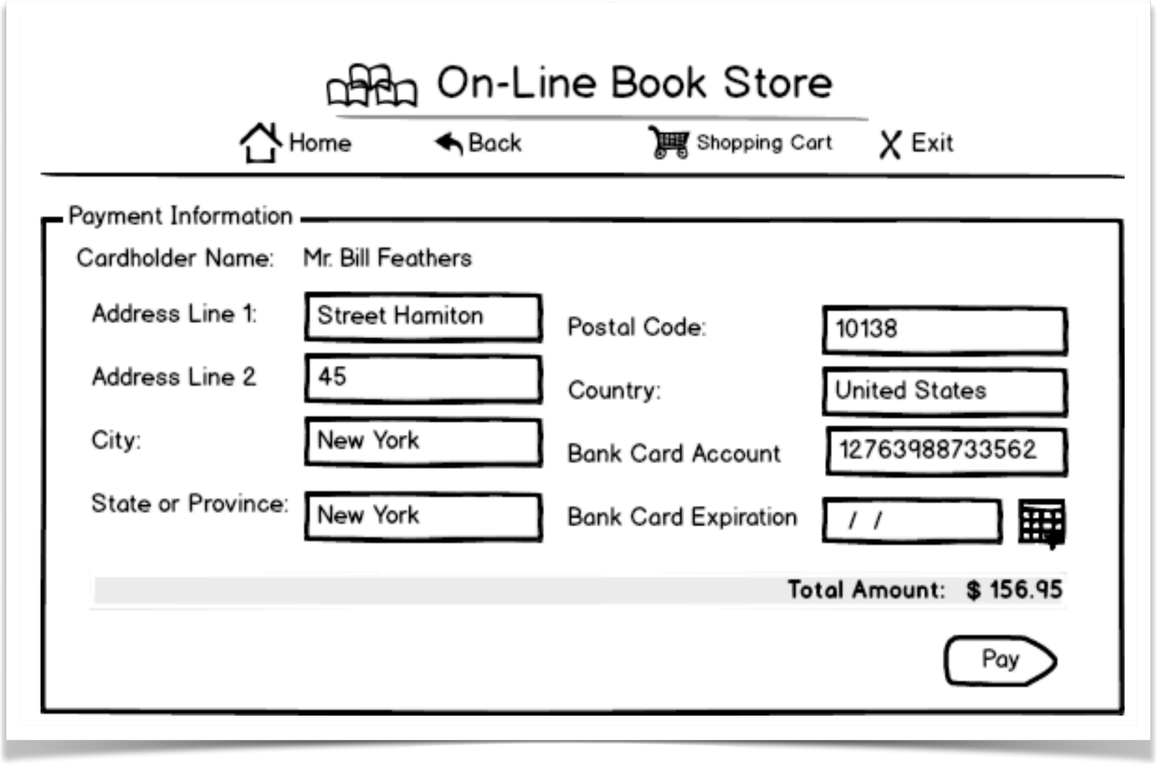
When landing on the shopping cart page



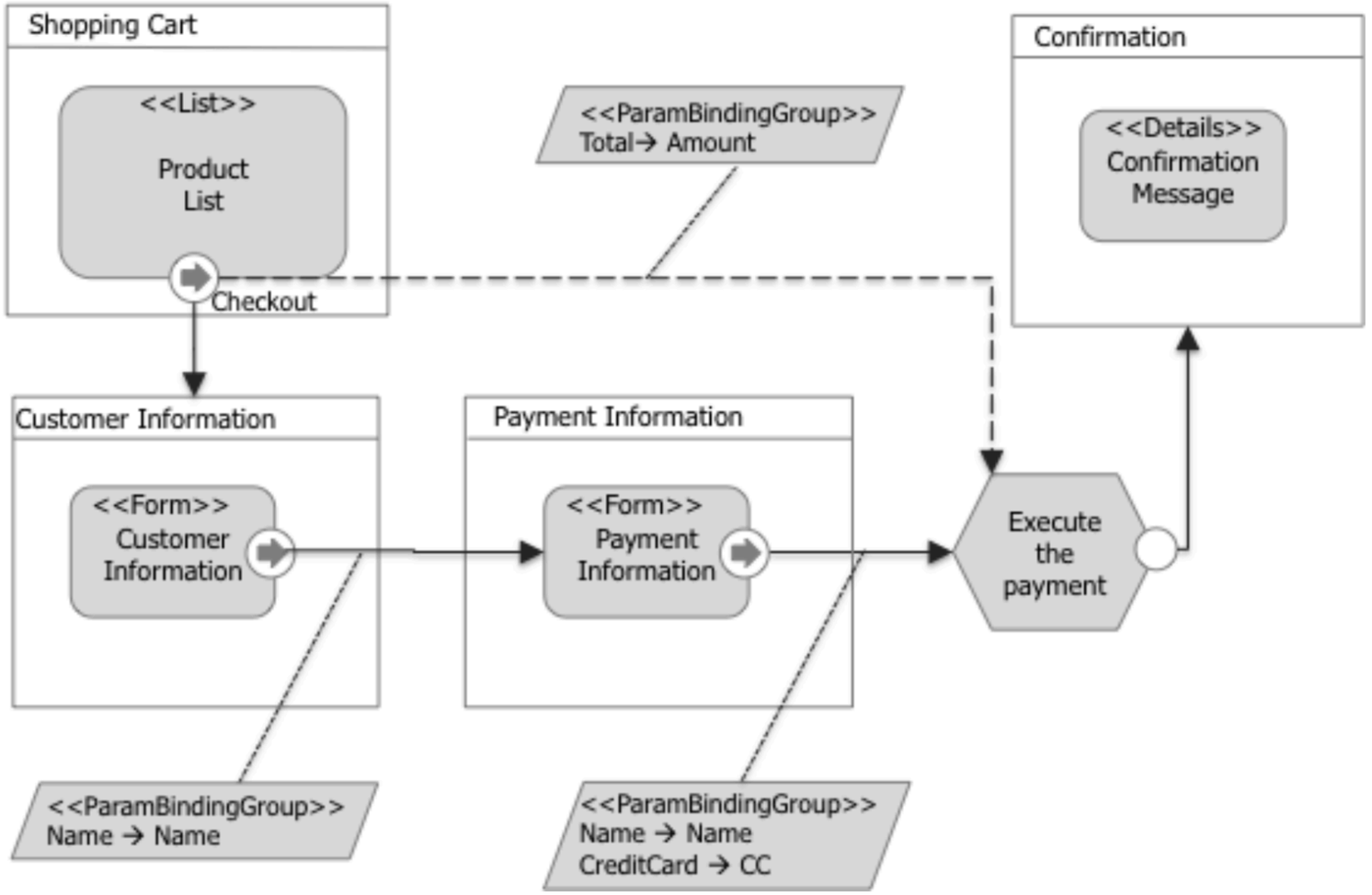
Events - Interactive interfaces



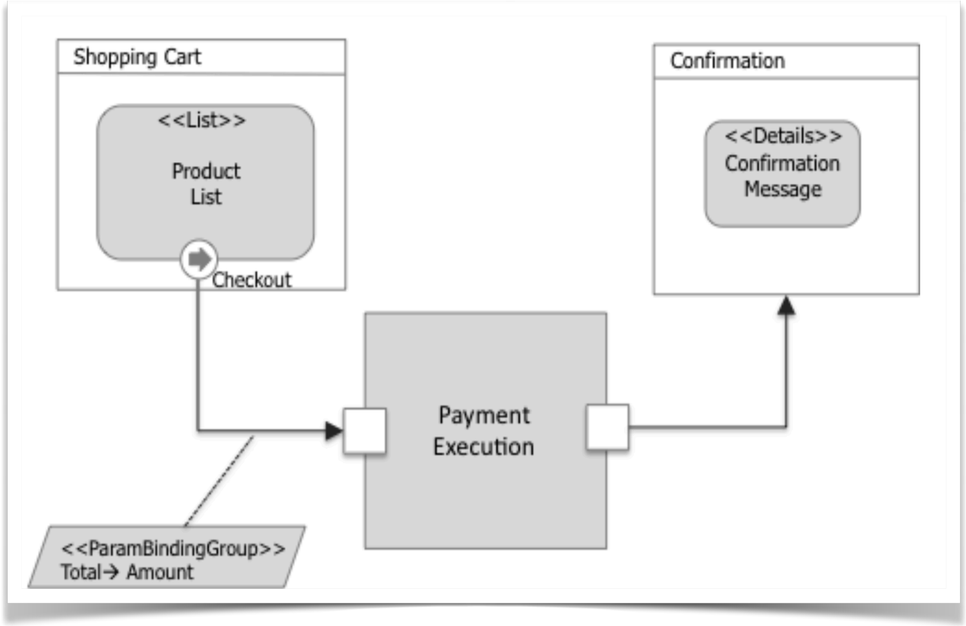


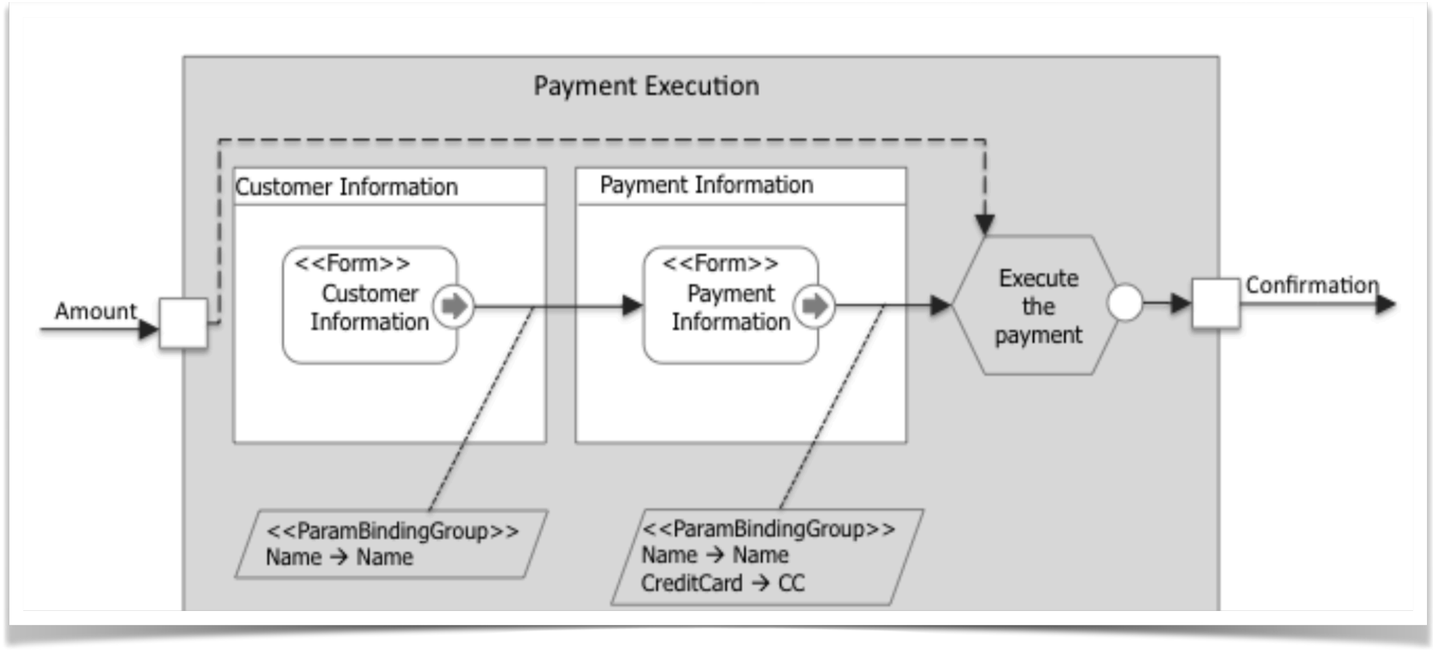


Events - Interactive interfaces



Composition and subcomponents





Summary

- Mock up screens and components are the main design tool for user interfaces
- IFML complements mock ups with behaviour: data passing, event handling, action triggering
- IFML is hierarchical and compositional