Modern Web Development & DevOps

20 October 2021 - Joris Herbots

UHASSELT EDM

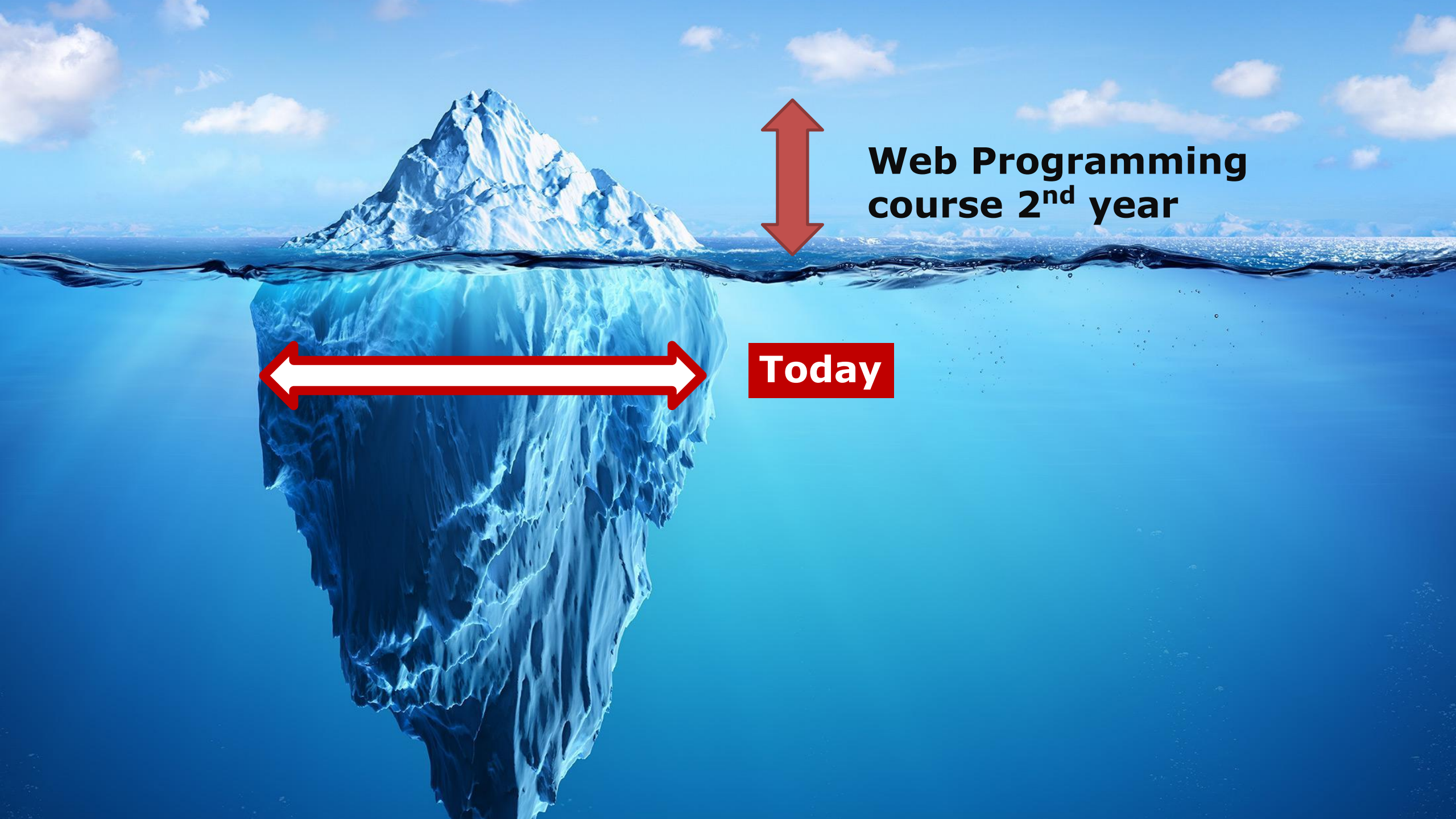
Introduction

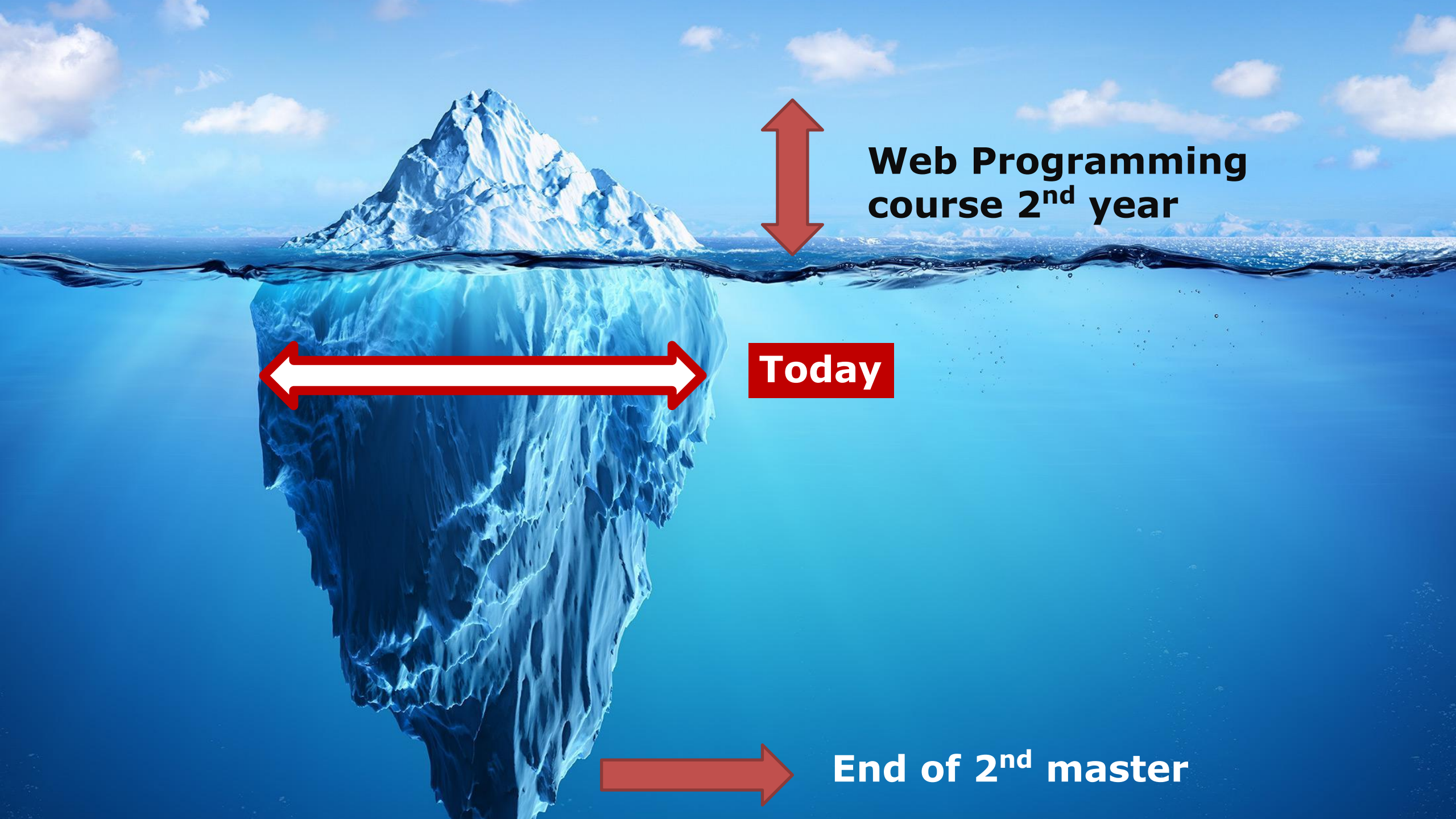
- Today we will discuss **modern web development** and **DevOps**
- **Difficult and complex field** with many topics
- Not easy to get into for new programmers

- Goal of today:
  - Get you an overview of what modern web development entails
  - Give you insights into how we got here
  - Teach you how you can do some DevOps of your own

- What is today not:
  - A coding tutorial
  - Mandatory subject matter for the exam
    - So relax and enjoy the ride for once, because it's going to be a wild one


Web Programming
course 2nd year


Web Programming
course 2nd year
Today


Web Programming course 2nd year
Today
End of 2nd master

List of topics we will handle today

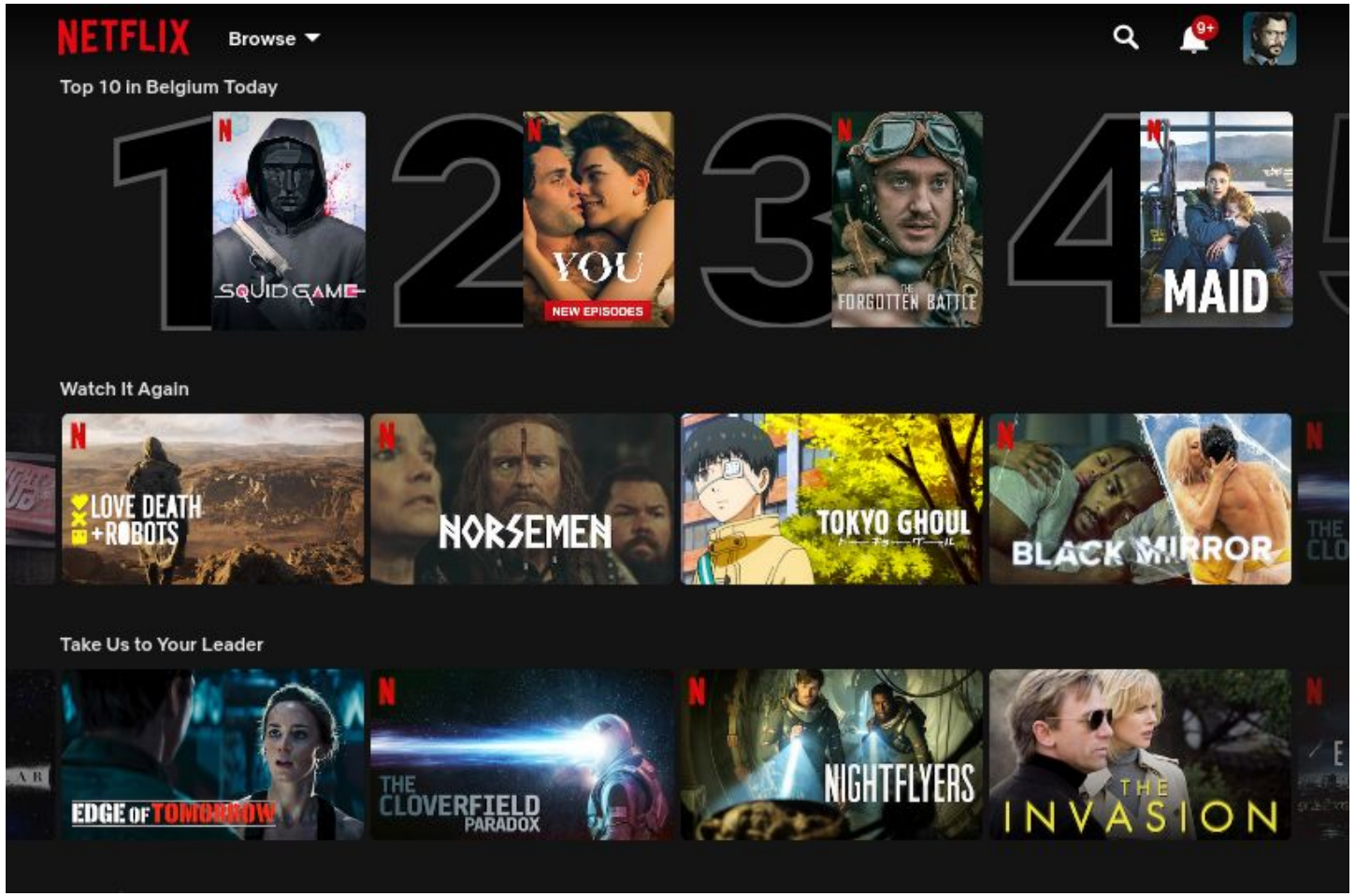
- History of the web
  - Part 1: The stuff you (should) already know
  - Part 2: The abyss of technology you (probably) don’t know
- DevOps
  - Linux
  - Virtual Machines
  - Containers
- Project requirements from this lesson

History of the web - Part 1: The stuff you (should) already know

https://xkcd.com/1144/

Question for the audience

Why is this so popular nowadays?



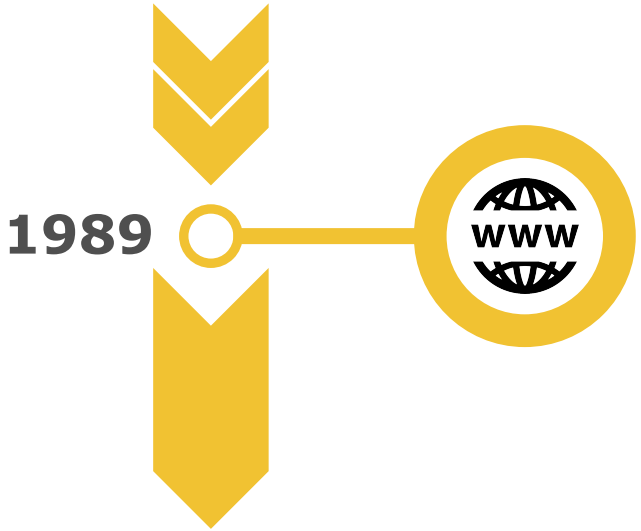
Getting our definitions straight

- **World Wide Web** (WWW) or **Web** is an information system
  - Consists of **resources** (e.g., web pages, images, video)
  - Resources are identified using **Uniform Resource Locators** (URLs)* (e.g., *http*://jherbots.info/)
    - **Hypertext Transfer Protocol** (HTTP) is the underlying protocol for transferring resources between a client and server
    - Resources can be interlinked using **hyperlinks** (e.g., http://jherbots.info/index.html)
- The Web is a **decentralized** given
  - Resources are (geographically) spread over multiple **web servers**
  - Resources under the same domain (e.g., jherbots.info) make up a **website**
  - A client connects to a website using a **browser**

- Backend vs. frontend *!Important: remember the difference, it'll get confusing real soon*
  - "Everything" that happens on a web server is considered the **backend**
  - "Everything" that happens in the browser (i.e., the client) is considered the **frontend**

* https://datatracker.ietf.org/doc/html/rfc3986

Timeline of key events in the development of web technologies



1989 Birth of the **World Wide Web**

Ye olde web

- In its most **simplistic form**, a web server handles **static resource requests**
  - Static resources do not change depending on user input
    - Examples: HTML pages, images, PDFs, music files, movie files, …
  - Nowadays, many web servers exist: Apache, Nginx, Lighttpd, Hiawatha, …

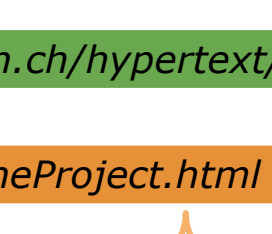
Ye olde web

Frontend

Backend

GET *http://info.cern.ch/hypertext/WWW/TheProject.html*

TheProject.html resource

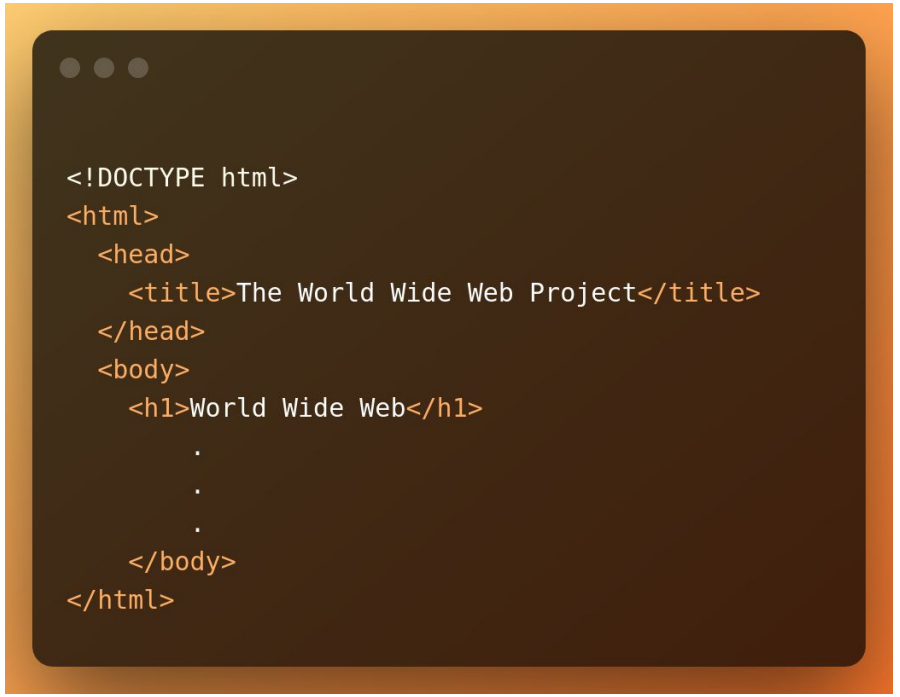


World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

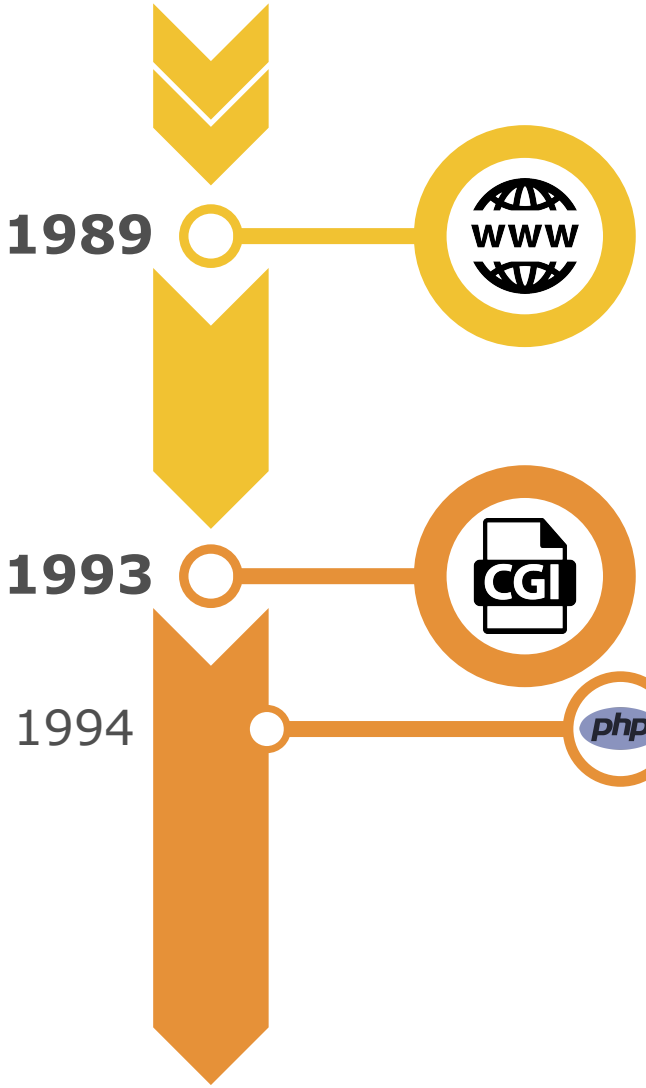


```
<!DOCTYPE html>
<html>
  <head>
    <title>The World Wide Web Project</title>
  </head>
  <body>
    <h1>World Wide Web</h1>
        .
        .
        .
    </body>
</html>
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

http://info.cern.ch/hypertext/WWW/TheProject.html

Timeline of key events in the development of web technologies



1989 Birth of the **World Wide Web**

1993 **Dynamic WWW** comes into existence with the arrival of the **Common Gateway Interface** (CGI)

1994 **Personal Home Page** (PHP) Tools V1.0 released

Birth of web applications: server side rendering

- People realised that **deferring HTTP requests to a script** in the backend that could generate HTML content **on-the-fly** could prove to be beneficial
  - The **Common Gateway Interface** (CGI) is the first effort into creating a unified way for exposing applications to the web
  - One could argue that these were the first cases of **web applications**
  - Mainly used to connect to a **database**
    - Examples: Contact forms, search boxes, surveys, … → **HTML forms**

- Scripts are run in the **backend**
  - Scripts are responsible for generating correct HTML and HTTP response → (i.e., **Server side rendering)** !Important: keep this definition in mind for when we get to SPAs
  - Users pass requests or submissions through **HTTP GET URL parameters** or **HTTP POST**
  - If something changes, a **full reload of the resource** from the server is required

Birth of web applications: server side rendering

- The first scripting language for web applications was a set of scripts created by Rasmus Lerdorf released as **Personal Home Page (PHP) Tools**

> "I really don't like programming. I built this tool to program less so that I could just reuse code."

```
<!--include /text/header.html-->

<!--getenv HTTP_USER_AGENT-->
<!--ifsubstr $exec_result Mozilla-->
  Hey, you are using Netscape!<p>
<!--endif-->

<!--sql database select * from table where user='$username'-->
<!--ifless $numentries 1-->
  Sorry, that record does not exist<p>
<!--endif exit-->
  Welcome <!--$user-->!<p>
  You have <!--$index:0--> credits left in your account.<p>

<!--include /text/footer.html-->
```

A lot changed in the past ~3 decades

```php
<?php
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
?>
```

https://www.php.net/manual/en/history.php.php

https://web.archive.org/web/20130728125152/http://itc.conversationsnetwork.org/shows/detail58.html

Birth of web applications: server side rendering

Frontend

Backend

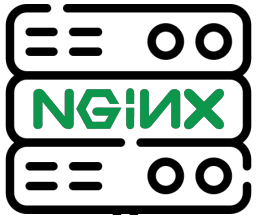
Browser

Server

Script

GET *index.php?firstname=Joris&lastname=Herbots*

dynamic resource



1

2

3

4

1. CGI* starts a PHP process
2. The PHP process fetches the requested script
3. Script processes with provided input parameters and renders HTML + HTTP response headers**
4. PHP process finishes and output is sent to the client

index.php

```
<!DOCTYPE html>
<html>
  <body>
    Hi Joris Herbots
  </body>
</html>
```

index.php

```
<!DOCTYPE html>
<html>
  <body>
    <?php
      $first = $_GET['firstname'];
      $last = $_GET['lastname'];
      echo "Hi " . $first . " " . $last;
    ?>
  </body>
</html>
```

* CGI is nowadays mostly replaced by alternatives like FastCGI and uWSGI

** HTTP response headers are typically abstracted away in modern web programming languages

https://www.php.net/manual/en/install.unix.commandline.php

Timeline of key events in the development of web technologies

Birth of web applications: client side scripting (JavaScript)

- JavaScript was created to act as a **glue language**: a not too difficult programming language so designers could automate interactions
  - Originally named **Mocha**, then **LiveScript** and finally **JavaScript**
  - First version was created in 10 days (and did not perform very well…)
  - Being the only in-browser language for dynamic interactions… it spread like wildfire

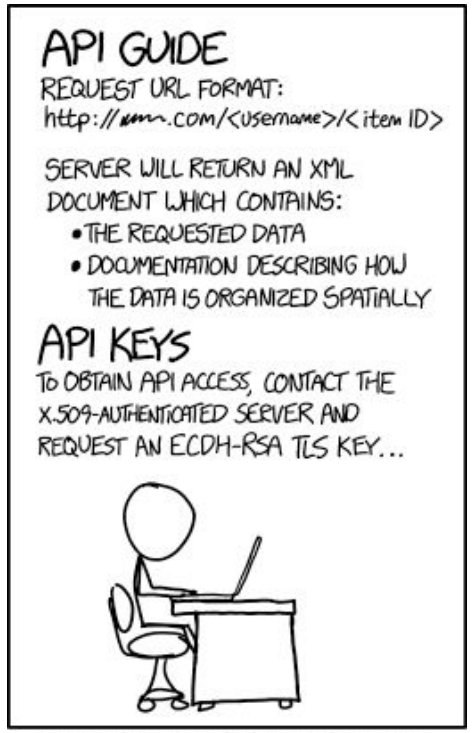
Birth of web applications: client side scripting (rich applications)

- Flash (Actionscript), Java Applets, ActiveX controls, … were the other side of the same coin
  - Provided **better performance** and enabled **rich web applications**
    - Examples: video streaming (YouTube), games (RuneScape, Minecraft)
  - But… were riddled with **many security issues**
  - Made use of the **browser plugin interfaces** !Important: remember this for when we arrive at HTML 5

```html
<!DOCTYPE html>
<html>
  <body>
    <h1 id="intro">Hello world!</h1>

    <script type="text/javascript">
        document.addEventListener("onload", function() {
          const intro = document.getElementById("intro");
          intro.innerText = intro.innerText.toUpperCase();
        });
    </script>
  </body>
</html>
```

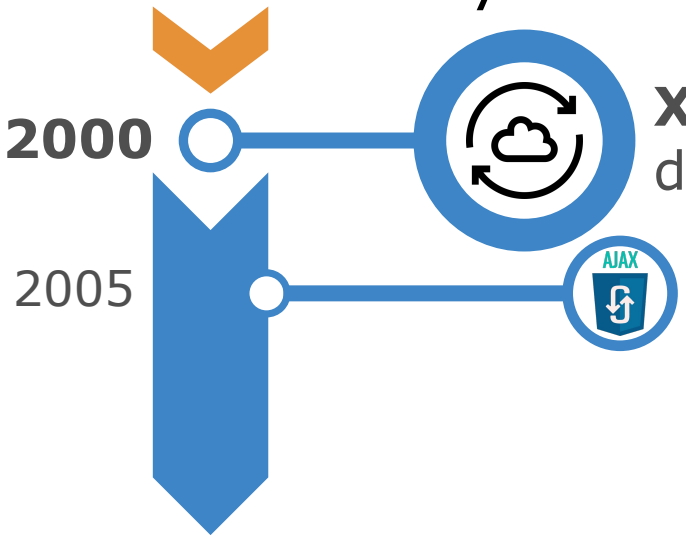
Little recap

- To recap:
  - We have a way to display resources in the browser with HTML
  - We have a mechanism to control how these resources look using CSS
  - We can interact with backend state by passing request parameters to scripts
  - We can make the website/web application dynamic with JavaScript
- What would be the next logical step?



IF YOU DO THINGS RIGHT, IT CAN TAKE PEOPLE A WHILE TO REALIZE THAT YOUR "API DOCUMENTATION" IS JUST INSTRUCTIONS FOR HOW TO LOOK AT YOUR WEBSITE.

Timeline of key events in the development of web technologies



XMLHttpRequest (XHR) brings forth second wave of dynamic WWW

Asynchronous JavaScript + XML (Ajax) coined

The rise of the JavaScript: XMLHttpRequest

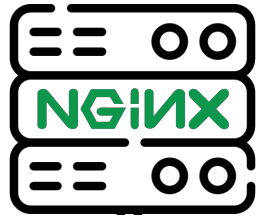
- **XMLHttpRequest** (XHR) allows communication with the backend **without reloading** the page
  - Came forth from the desire to create Win32-like applications in the browser
    - Original creators were the Outlook Web Access Developers
  - Almost all interactive implementations uses XHR nowadays.
    - Examples: Google Search, Google Maps, BlackBoard, this presentation slideset, …
      If it handles some data/state, you can be pretty sure it's interacting with the backend using XHR
- Typically confused with **Ajax**, which is a term used to describe **Asynchronous JS and XML***
  - XML is a bit misleading, whilst the most popular when XHR was created, we use **JSON** more now
- XMLHttpRequest is being phased out for the **Fetch API****

* https://immagic.com/eLibrary/ARCHIVES/GENERAL/ADTVPATH/A050218G.pdf
** https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

The rise of the JavaScript: XMLHttpRequest

Frontend

Backend



GET *index.php*

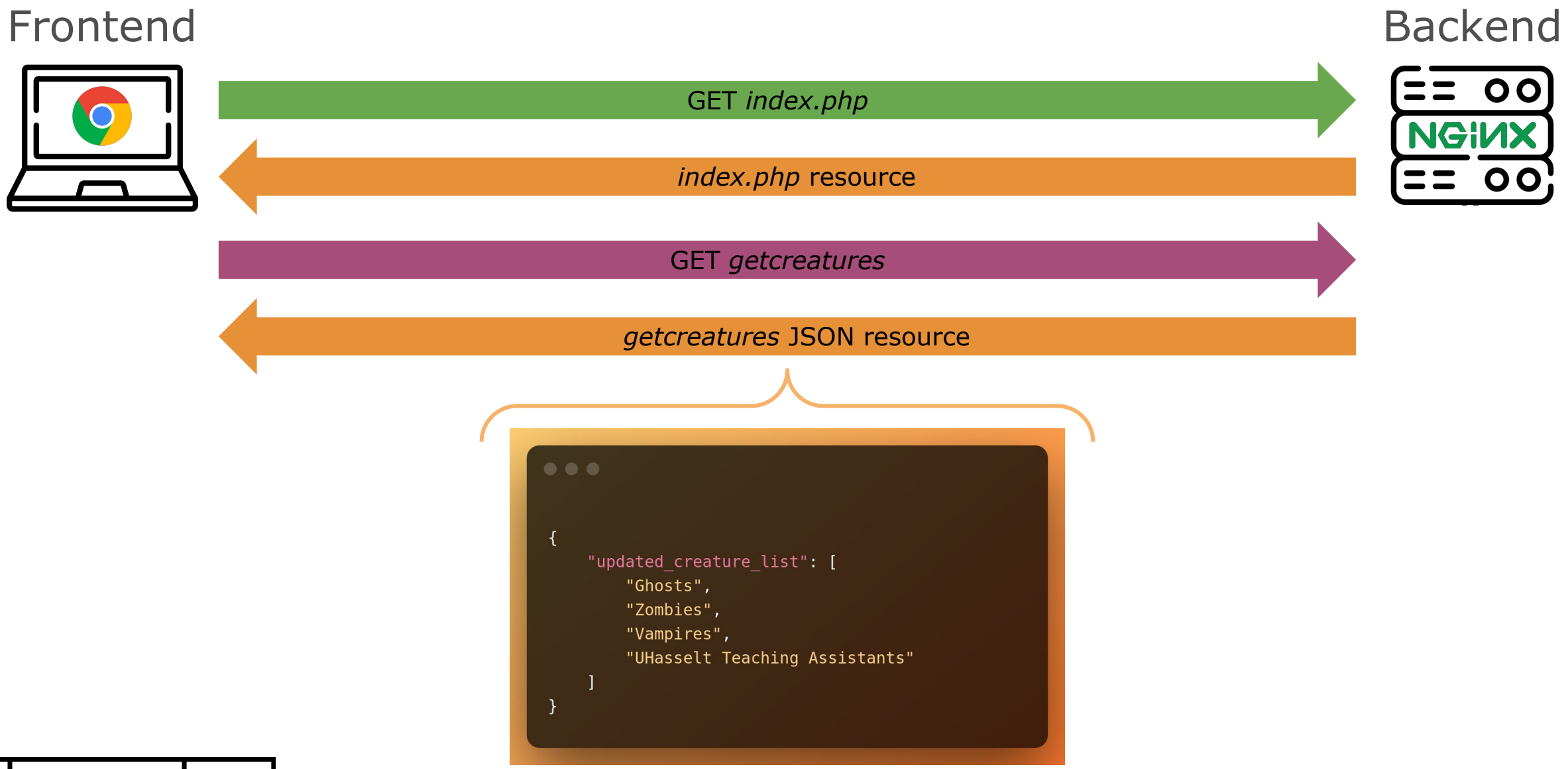
index.php resource

```html
<html>
<body>
    <ul id="halloween_creatures">
        <li>Ghosts</li>
        <li>Zombies</li>
    </ul>

    <script type="text/javascript">
        fetch('http://www.example.com/getcreatures')
          .then(response => response.json())
          .then(creatures => {
            // TODO: re-render our #halloween_creatures list in the DOM
          });
    </script>
</body>
</html>
```

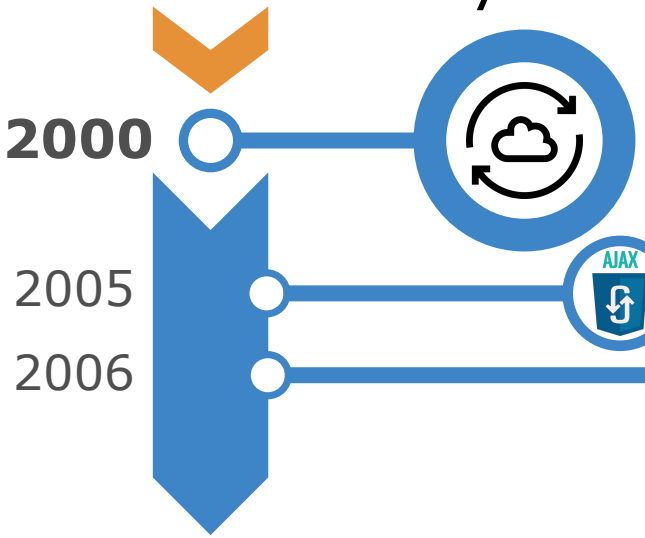
=

```html
<html>
<body>
    <ul id="halloween_creatures">
        <li>Ghosts</li>
        <li>Zombies</li>
    </ul>

    <script type="text/javascript">
        let req = new XMLHttpRequest();
        req.addEventListener('readystatechange', function() {
            if (req.readyState == 4 && req.status == 200) {
               creatures = JSON.parse(req.responseText);
               // TODO: re-render our #halloween_creatures list in the DOM
            }
        });
        req.open("GET", "http://www.example.com/getcreatures");
        req.send();
    </script>
</body>
</html>
```

The rise of the JavaScript: XMLHttpRequest



Timeline of key events in the development of web technologies



2000 — **XMLHttpRequest** (XHR) brings forth second wave of dynamic WWW

2005 — **Asynchronous JavaScript + XML** (Ajax) coined

2006 — **jQuery** JS library released

JavaScript is dead, long live jQuery!

- Manipulating the **Document Object Model** (DOM) is tedious
- (oldschool) JavaScript claims to be cross-platform but in reality differences exist which make it difficult for developers

- **jQuery** library to the rescue!
  - Easy querying and manipulation of DOM elements using **$***
  - Easier Ajax handling
  - Easy and intuitive event listeners

```
$("#halloween_creatures li").each( function (){
    let creature  = $(this);
    creature.text("🎃 " + creature.text());
});
```

- Modern JS (2021) has had a lot of updates
  - **HTML 5 APIs** make life easier
  - DOM traversal and manipulation is on-par with jQuery
    - **document.querySelector()** was added because of jQuery!!
  - jQuery is still popular, but not a necessity anymore

* *$()* is a shortcut for *jQuery()*

History of the web - Part 2: The abyss of technology you (probably) don’t know

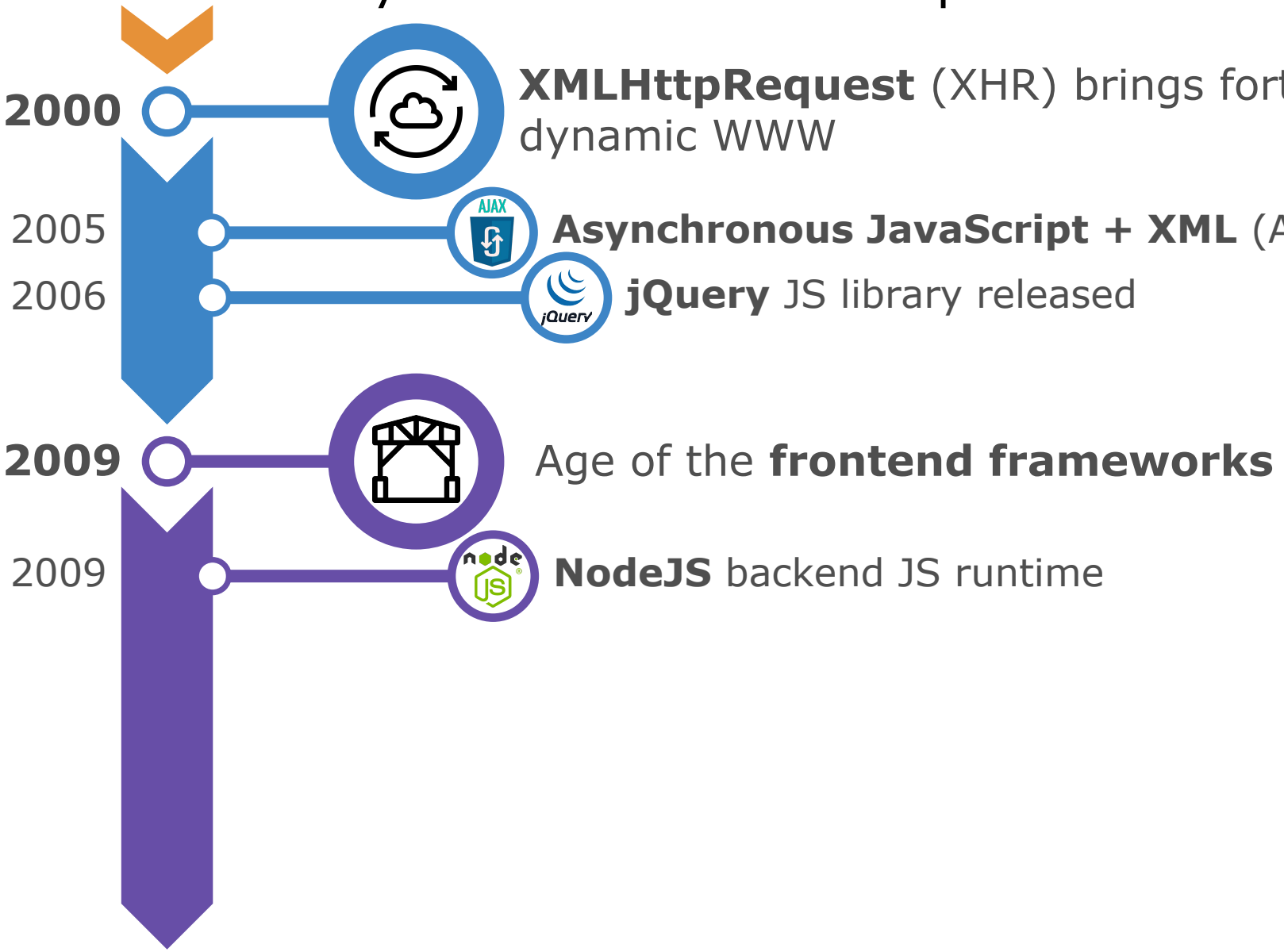
https://xkcd.com/1144/

Duality problem

1. Load a web page
2. Manipulate parts of the page using jQuery and Ajax
3. Browse to a new page || change page content using HTTP POST
4. Rinse and repeat

- **State is split** across page loading and Ajax endpoints
  - Backend needs to render HTML pages and provide Ajax endpoints (in multiple formats: XML, JSON, templates, …)
  - Non-trivial task to decide where to put what logic…
- Frontend code written in JS needs to handle the same data structures as the backend written in another language (PHP, Python, Ruby, …)
  - Similar to how we do some pre-checks on form fields in the frontend but also in the backend

Timeline of key events in the development of web technologies



NodeJS and NPM

- "What if we wrote JavaScript code on the backend?"
  - Cue **NodeJS**: a rip of the Google Chrome V8 engine with some added native APIs (e.g., filesystem, networking, …)
  - Now we can reuse our backend code in the frontend and vice versa!

    → **Isomorphic JavaScript**

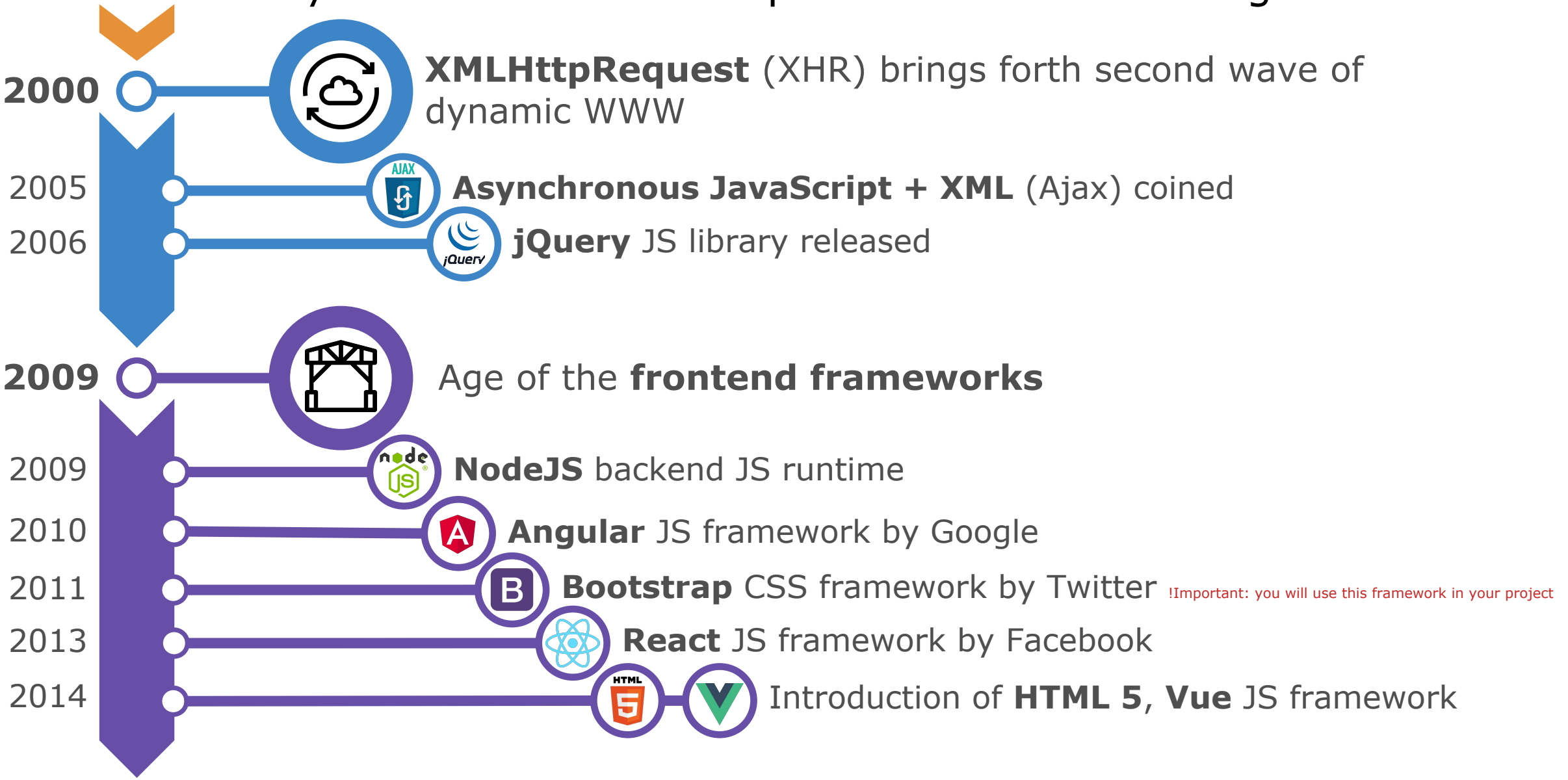
    !Important: We need tools and/or bundlers for this to work, these will not be discussed in this lecture (see PSOPV).

- "Let's make sharing our JavaScript codes with each other easier!"
  - Cue the **Node Package Manager** (NPM)
    - The largest ecosystem for sharing JavaScript modules !Important: NPM by itself is riddled with issues (see master networking and security for more info)
  - Need some common functionality? → *npm install <package>*

Single Page Application

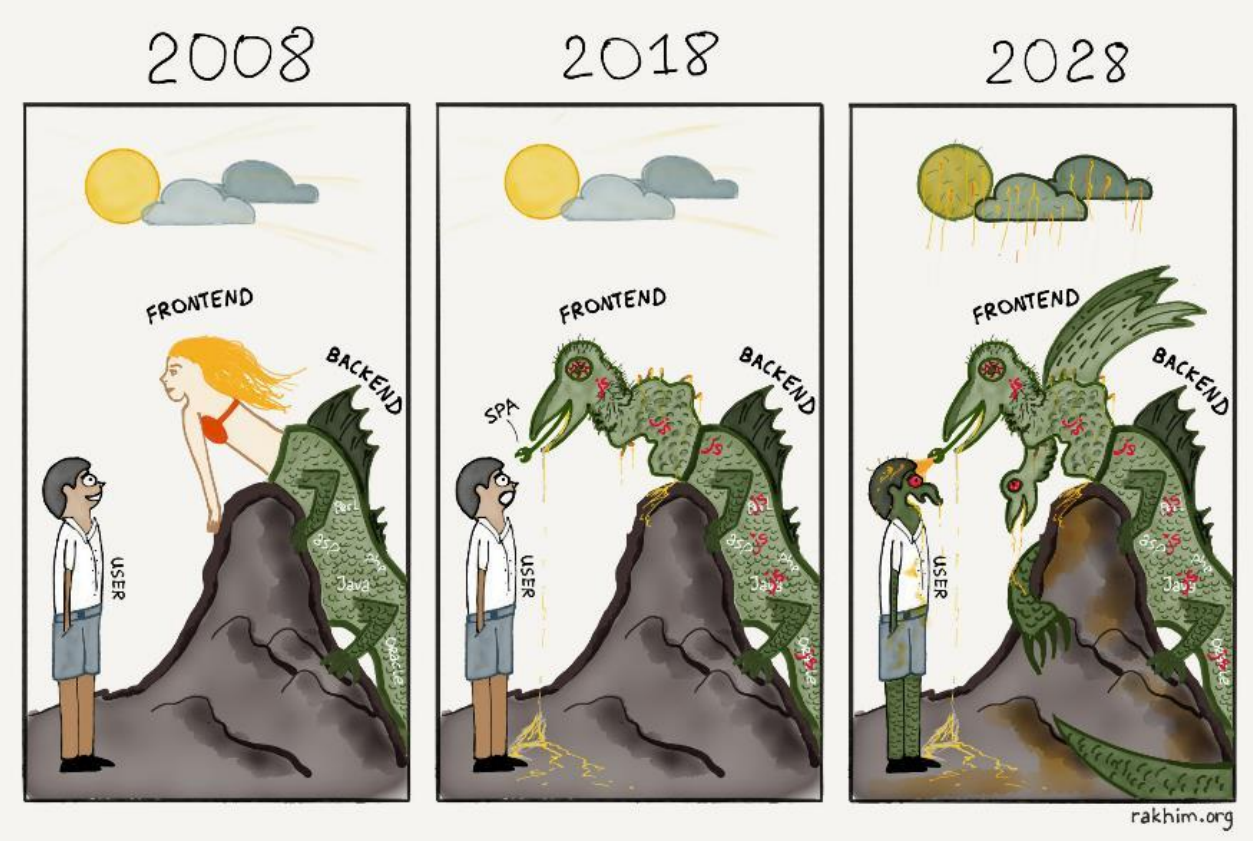
- So if we use Isomorphic JS… why still use server side rendering?
- Cue **Single Page Application** (SPA)
  - Frontend requests a web application
  - Frontend fetches **data and templates** from the backend using Ajax upon interacting with the state which gets updated in-place
  - Backend requires **less code** and can handle **more load**!
  - No more frontend page reloads required at all!
- Examples of SPAs: Facebook, Instagram, Reddit, Linkedin, …
  - If you don’t experience a page reload, you can be sure it’s a SPA

Timeline of key events in the development of web technologies



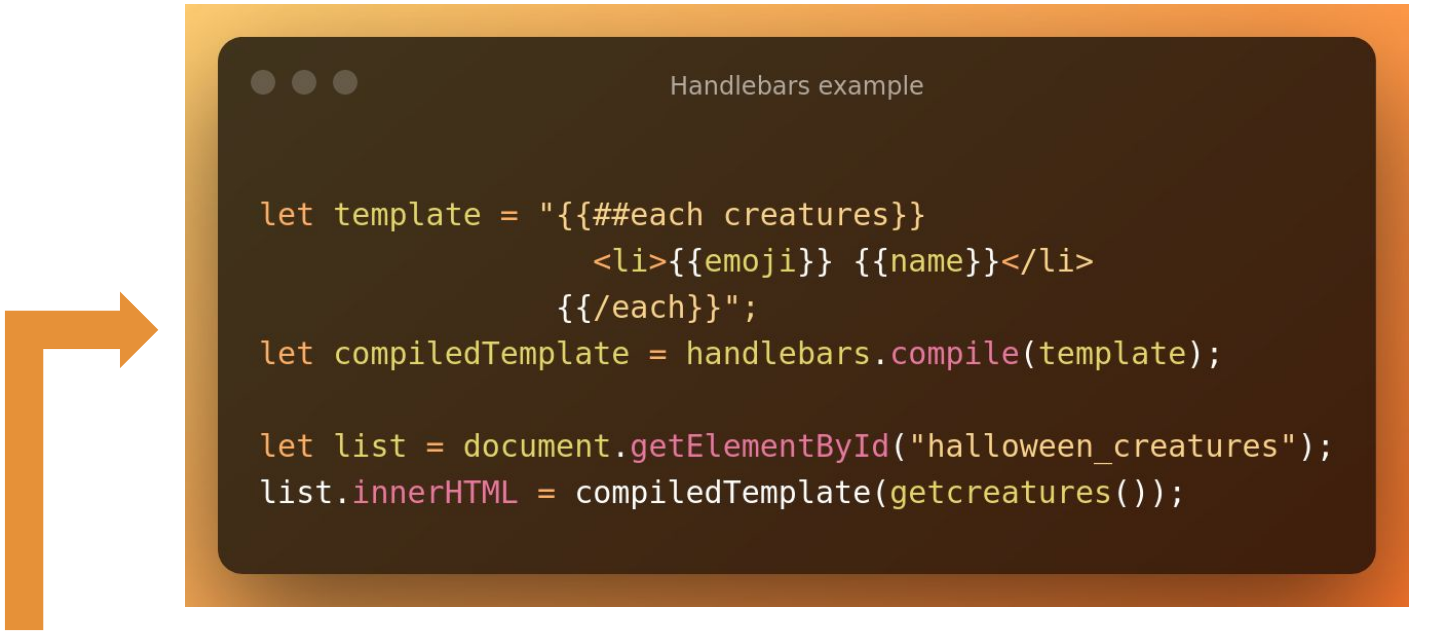
Modern frontend frameworks

- Modern JS frontend frameworks are a **complex given**
  - We will only handle what key technologies make up frontend frameworks, not how they work
  - A deep dive into this topic will be done during **PSOPV**



Modern frontend frameworks: Reactive data binding

- Rendering everything at the frontend requires **a lot of manual updating**
  - We need to retrieve state and update the DOM for every change
  - **Updating the DOM is very costly**! Should be avoided as much as possible.



- Doing it this* way, is expensive and not performant on big websites

*Example uses the HandleBars.js templating library. Many other similar libraries exist.

Modern frontend frameworks: Reactive data binding

- Solution to tedious DOM management: **Reactive data binding**
  - We bind some **state** to the **DOM** directly, when the state changes, the DOM also updates
  - Works by means of **getter and setter property accessors** combined with **observers**
    - We will see more on this during PSOPV, **for now assume this is magic**!
- Combination of a templates and JS code are called **Components**

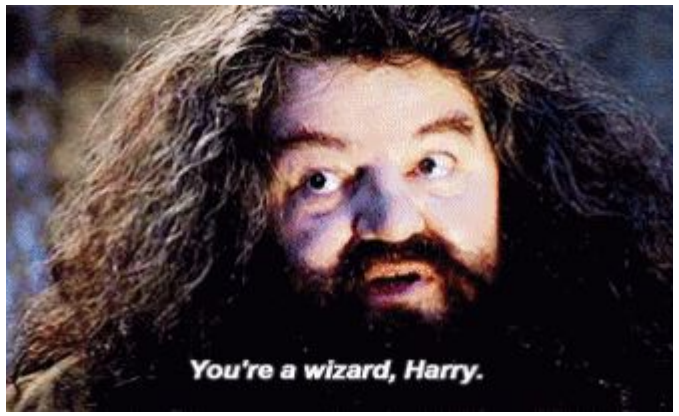
Magic

```
<template>
    <ul id="halloween_creatures">
        <li v-for="creature in halloweenCreatures">{{creature.emoji}} {{creature.name}}</li>
    </ul>
</template>

<script type="text/javascript">
    let state = {
        halloweenCreatures: [
            {emoji: "👻", name: "Ghosts"},
            {emoji: "🧟", name: "Zombies"}
        ];
    };

    function fakeEvent() {
        // Calling this function will automatically re-render our <template>
        state.halloweenCreaturs.push( {emoji: "🎃", name: "UHasselt Teaching Assistants"} );
    }
</script>
```

Modern frontend frameworks: Reactive data binding

- Solution to tedious DOM management: **Reactive data binding**
  - We bind some **state** to the **DOM** directly, when the state changes, the DOM also updates
  - Works by means of **getter and setter property accessors** combined with **observers**
    - We will see more on this during PSOPV, **for now assume this is magic**!
- Combination of a templates and JS code are called **Components**

- 👻 Ghosts
- 🧟 Zombies



- 👻 Ghosts
- 🧟 Zombies
- 🎃 UHasselt Teaching Assistants

Magic

```
<template>
    <ul id="halloween_creatures">
        <li v-for="creature in halloweenCreatures">{{creature.emoji}} {{creature.name}}</li>
    </ul>
</template>

<script type="text/javascript">
    let state = {
        halloweenCreatures: [
            {emoji: "👻", name: "Ghosts"},
            {emoji: "🧟", name: "Zombies"}
        ];
    };

    function fakeEvent() {
        // Calling this function will automatically re-render our <template>
        state.halloweenCreaturs.push( {emoji: "🎃", name: "UHasselt Teaching Assistants"} );
    }
</script>
```

Modern frontend frameworks: Virtual DOM

- But wait! Weren't DOM updates expensive?
- Cue the **Virtual DOM** (VDOM)
  - An abstraction of the real DOM
  - Instead of changing the real DOM, frameworks modify the VDOM
    - Works kind of like a blueprint containing all elements that need to be updated in the DOM
  - VDOM is **much faster** than the real DOM
  - Changes to the real DOM are done in a smart way by comparing the VDOM tree to the real DOM tree and **only rendering actual changes**
- Creating the VDOM requires an **upfront compilation of the template(s)**

Upsides and downsides of modern frontend frameworks

Upsides

+ Less manual DOM update code
+ Higher developer productivity and better code maintainability
+ Robuster state management

Downsides

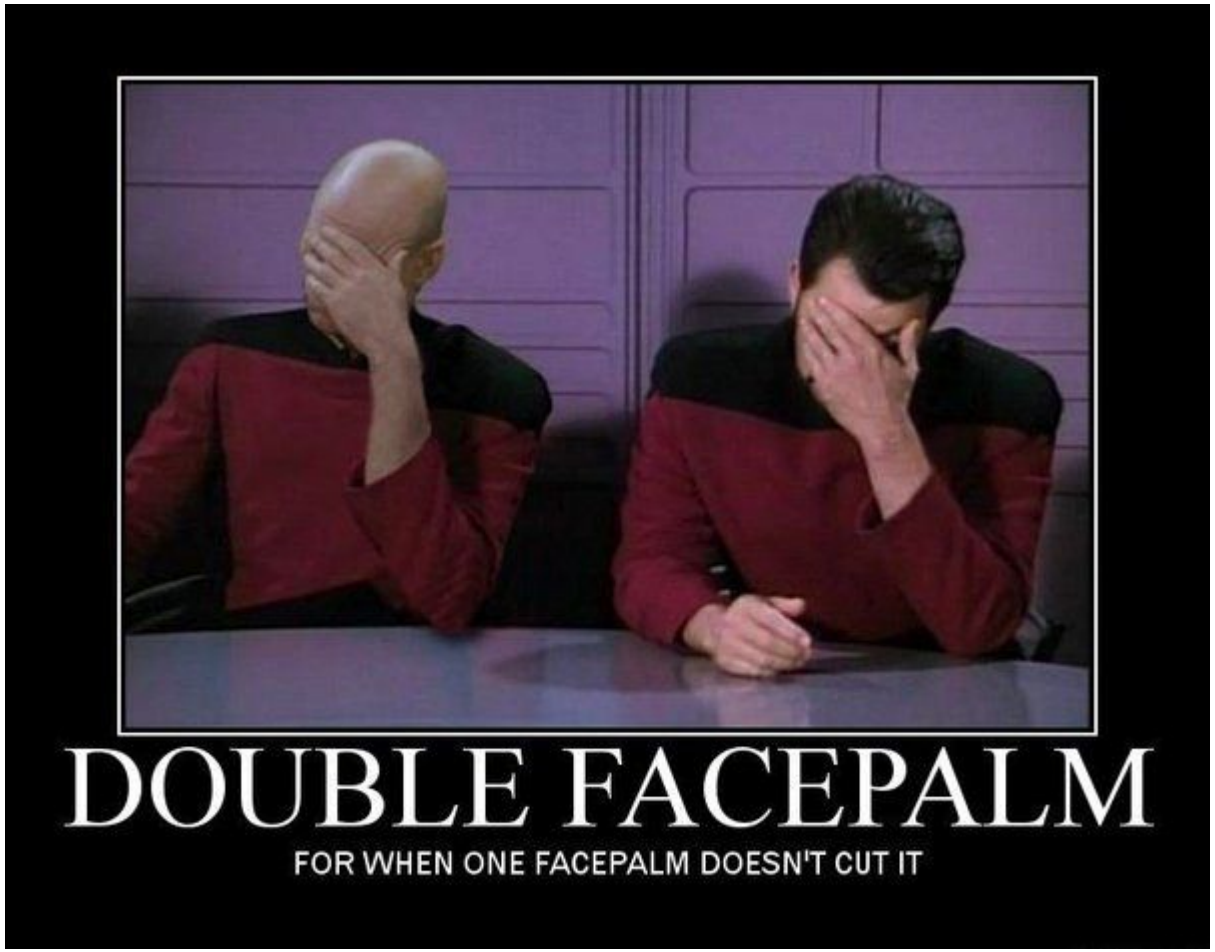
- Steep learning curve and difficult setup
- Vendor lock in, by only learning a framework you are never truly adaptive
- Web performance declines
  - First render of the website takes longer
  - Time to interactive suffers

History is doomed to repeat itself

- So… rendering the web page at the client is slow?
  - Maybe we can speed things up by rendering some stuff at the server!

History is doomed to repeat itself

- So… rendering the web page at the client is slow?
  - Maybe we can speed things up by rendering some stuff at the server!

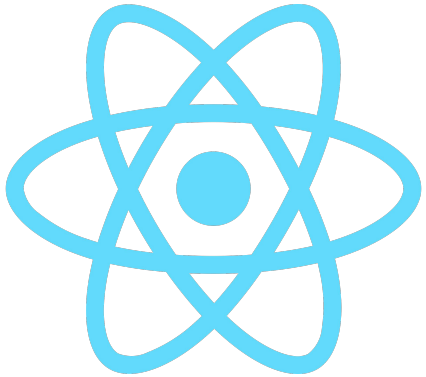


Reintroducing our old friend: Server side rendering

- **Server side rendering** is back!
  - But things have changed because of **isomorphic JS** and the **VDOM**
  - We can already pre-render the VDOM at the server!
- But… because it's rendered at the backend, we have no interactive bindings
  - Solution: Frontend **(re)hydration**
    - Show server side rendered HTML
    - Then refresh the whole HTML and attach event handlers to interactive elements
  - This allows a user to see the website faster, but elements remain non-responsive during the initial load
    - Ever loaded Facebook or Reddit but the buttons and links didn't seem to work?

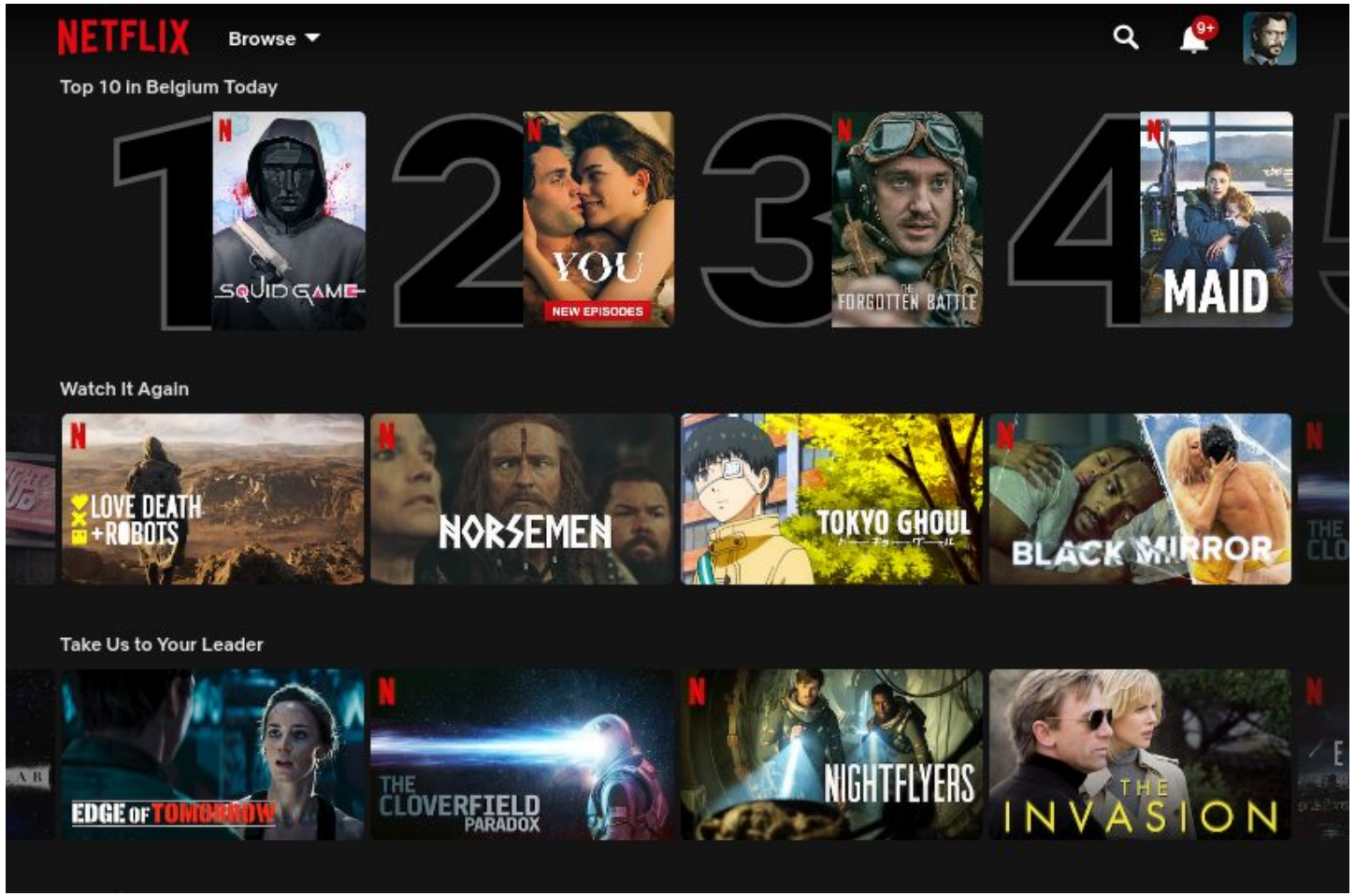
Overview of frontend frameworks

- There exists an assortment of frontend frameworks nowadays
  - Each has a design philosophy and doesn't necessarily do the same as another
  - A more deep dive into these will be undertaken during the start of PSOPV



Question for the audience

Let’s retake this question, why is this so popular nowadays?



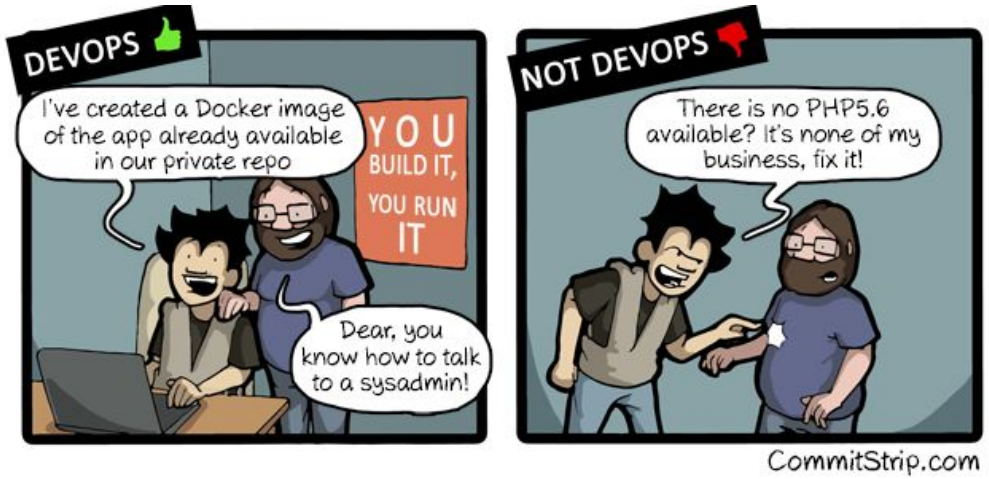
DevOps

https://xkcd.com/705/

DevOps

- **Developers set up their own infrastructure**
  - Very broad definition, we will focus on the deployment of web based projects today
- "But it works on my machine!" is not an excuse anymore
- Why do we need to setup our own infrastructure?
  - Because that's how it works in "real life"!
  - The web has become so complex, you can't do without anymore

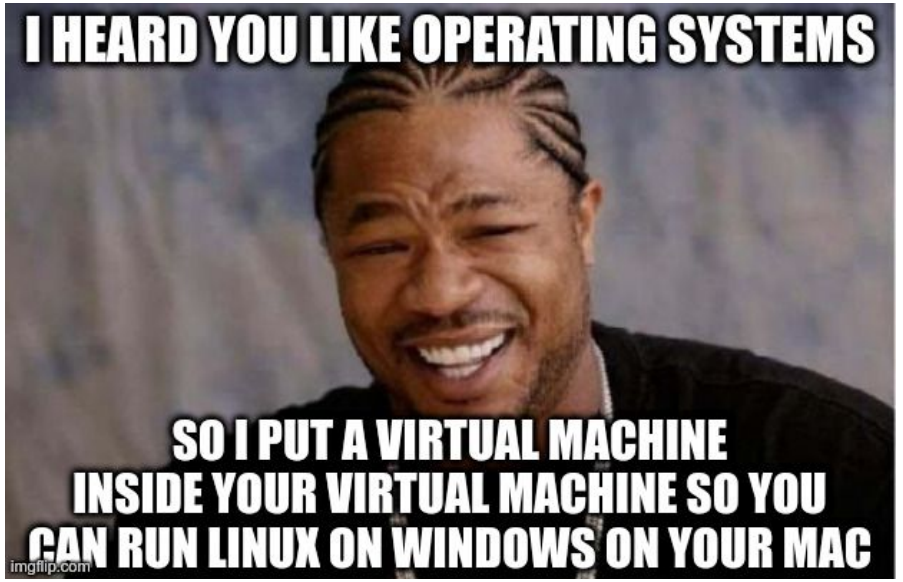
- Our focus today:
  - Linux
  - Virtual machines
  - Containers (with a focus on Docker)



Linux

- Hold on there! I like Windows more than Linux, can't we do Windows?
  - You better get used to it… you won't get far without a bit of Linux-foo*
  - The world is focussing on automation, **continuous delivery** (CD) and **continuous integration** (CI) are hot
  - Automation requires stability and thus avoids GUIs, command line tools are far from dead (e.g., sed, grep, cat, ps, ssh, nc, …)
    - Command line is **easy to automate**: we simply execute commands in the right order!

- Excuses we've heard before:
  - But I don't want to dual boot Windows and Linux
  - I run Arch, I don't want to install package X
- Introducing… **Virtual Machines**!
  - Install an operating system inside your operating system

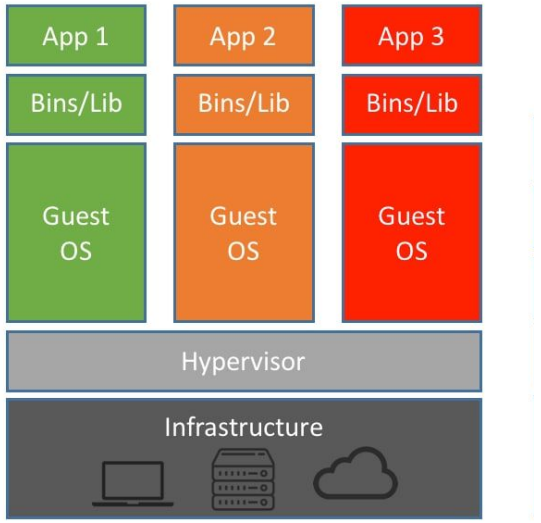


https://www.atlassian.com/devops/what-is-devops/devops-best-practices

* Even Microsoft had to bow before the superiority of Linux
https://www.zdnet.com/article/microsoft-developer-reveals-linux-is-now-more-used-on-azure-than-windows-server/

Virtual machines

- Virtual Machines (VM)
  - Easy to use with existings tools like VMware, Virtualbox, Parallels, KVM, …
  - Makes exchanging complex systems easier (especially handy in datacenter deployment scenarios)
  - But…
    - **Bulky**, a VM contains a full OS (you definitely do not want to push this to a GIT repo)
    - If something goes wrong, hope that someone wrote down how they created the VM (i.e., you have **no idea what and how software was installed/configured**)



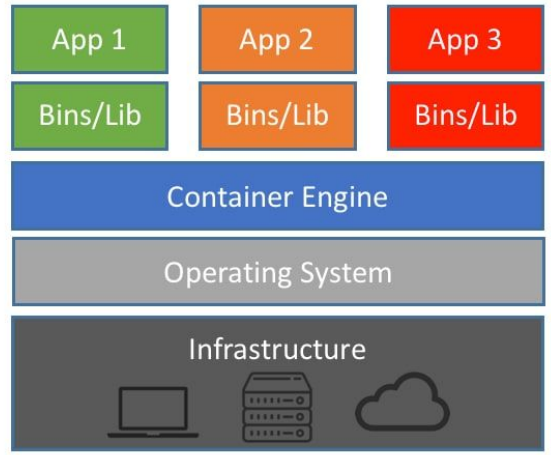
Machine Virtualization

Image: https://www.netapp.com/blog/containers-vs-vms/

Containers

- Containers
  - Exceptionally **lightweight** compared to VMs
  - Sit directly on top of the host OS
    - Shares host binaries
    - Shares host kernel
    - Processes within a container are self-contained and isolated from the host/other containers
  - Easy to distribute due to **small footprint** (MBs instead of GBs)
  - Make development and deployment easier in practice

- But what is a container?
  - A file with **instructions on how to setup/compile software** and run it with a **specific configuration**
  - There exists different container platforms: LXC, Docker, Hyper-V containers, …



Containers

Image: https://www.netapp.com/blog/containers-vs-vms/

Docker

- We will focus on docker during this lesson
  - Biggest container platform today
  - Large ecosystem of **images** provided via the **Docker Hub***
- Docker is not a container technology
  - Docker is a **tool to manage Linux containers**
  - It abstracts away the difficulties involved with packaging, sharing and running Linux containers
- https://docs.docker.com/ the documentation is your best friend!

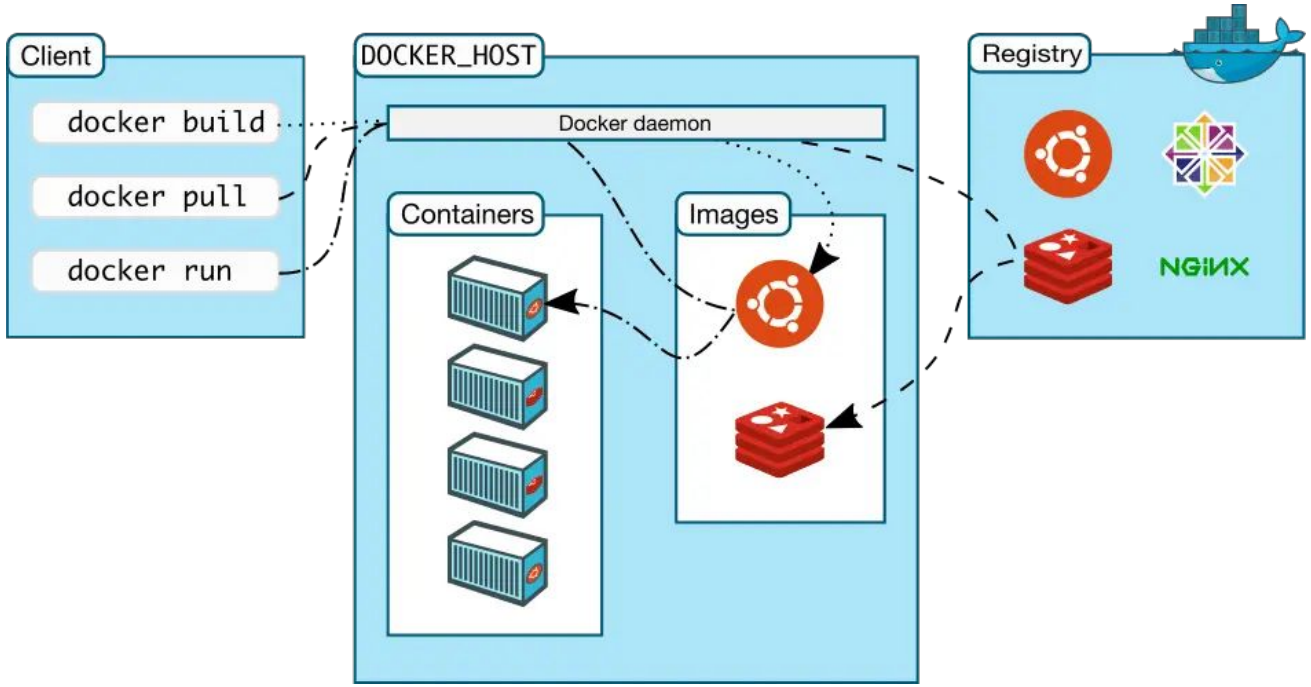
* https://hub.docker.com/

Dissecting docker commands: run

- docker run -p 8080:80 mattrayner/lamp:latest
  - docker is accessed through the *docker* command line program
  - *run* starts a new fully isolated container process https://docs.docker.com/engine/reference/run/
  - *-p 8080:80* is a port forwarding argument to expose internal network programs to the outside
    - Ports?? Programs accessible through a network are identified using port numbers.
    - Network processes containers are fully isolated and require port forwarding to be accessible
      - See course "Computernetwerken" in the third year
    - -p <localhost port>:<container port>
      - All traffic going from the adres localhost:8080 should go to container:80 and vice versa

Dissecting docker commands: run

- docker run -p 8080:80 mattrayner/lamp:latest
  - mattrayner/lamp:latest is the name of the image of which we want a container
    - Image?? A docker image is the basic building block of a container. It contains all the instructions on how to setup and start a container.
      - The mattrayner/lamp:latest image is one which is hosted on the docker hub, an online platform where we can find all sorts of images for different purposes (much like NPM for javascript)
      - https://hub.docker.com/



https://devopscube.com/what-is-docker/

Dissecting docker commands: run

- docker run -p 8080:80 mattrayner/lamp:latest
  - mattrayner/lamp:latest is an image which contains a full LAMP stack
  - Linux, Apache, MySQL and PHP (8.0)

```
jherbots … webprogramming guest_lecture_jherbots docker_lamp docker run -p 8080:80 mattrayner/lamp:latest
Unable to find image 'mattrayner/lamp:latest' locally
latest: Pulling from mattrayner/lamp
Digest: sha256:f7caeba589eaea7e3d20f4ae3e3bf88233e66e7df59eb5adb49161a5365f9f90
Status: Downloaded newer image for mattrayner/lamp:latest
Updating for PHP 8.0
Replacing CLI php.ini values
Editing APACHE_RUN_GROUP environment variable
Editing phpmyadmin config
Setting up MySQL directories
Allowing Apache/PHP to write to the app
Allowing Apache/PHP to write to MySQL
Editing MySQL config
=> An empty or uninitialized MySQL volume is detected in /var/lib/mysql
=> Installing MySQL ...
=> Done!
=> Waiting for confirmation of MySQL service startup
=> Creating MySQL admin user with random password
ERROR 1064 (42000) at line 1: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server
version for the right syntax to use near 'IDENTIFIED BY ''' at line 1
=> Done!
========================================================================
You can now connect to this MySQL Server with fs4MnezFpBp7

    mysql -uadmin -pfs4MnezFpBp7 -h<host> -P<port>

Please remember to change the above password as soon as possible!
MySQL user 'root' has no password but only allows local connections

enjoy!
========================================================================
Starting supervisord
2021-10-19 18:47:08,060 CRIT Supervisor is running as root.  Privileges were not dropped because no user is specified in th
e config file.  If you intend to run as root, you can set user=root in the config file to avoid this message.
2021-10-19 18:47:08,060 INFO Included extra file "/etc/supervisor/conf.d/supervisord-apache2.conf" during parsing
2021-10-19 18:47:08,060 INFO Included extra file "/etc/supervisor/conf.d/supervisord-mysqld.conf" during parsing
2021-10-19 18:47:08,063 INFO RPC interface 'supervisor' initialized
2021-10-19 18:47:08,063 CRIT Server 'unix_http_server' running without any HTTP authentication checking
2021-10-19 18:47:08,063 INFO supervisord started with pid 1
2021-10-19 18:47:09,068 INFO spawned: 'apache2' with pid 279
2021-10-19 18:47:09,072 INFO spawned: 'mysqld' with pid 280
2021-10-19 18:47:10,491 INFO success: apache2 entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
2021-10-19 18:47:10,492 INFO success: mysqld entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)
```

Dissecting docker commands: build

- docker build -t joris/website:latest /path/to/dockerfile
  - build creates a new image from a "dockerfile"
    - https://docs.docker.com/engine/reference/commandline/build/
  - -t joris/website:latest allows to tag the image so we can address it more easily later on
    - Naming convention: https://docs.docker.com/engine/reference/commandline/tag/
    - People typically do their github name or group name followed by a slash and the name of the project, the part after the semicolon is typically a version
      - Examples: uhasselt/website:2008, group3/psopv:latest, peter/webprogramming:version1

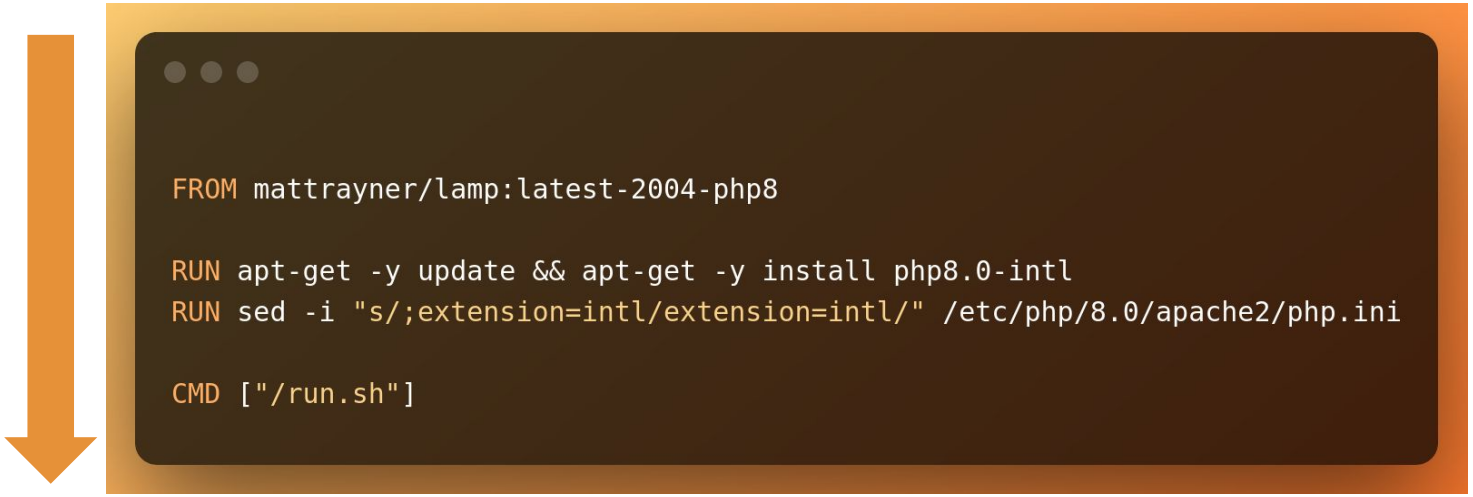
```
jherbots … webprogramming guest_lecture_jherbots docker_lamp docker build -t joris/lamp:latest .
Sending build context to Docker daemon  200.1MB
Step 1/4 : FROM mattrayner/lamp:latest-2004-php8
 ---> c15726116aef
Step 2/4 : RUN apt-get -y update && apt-get -y install php8.0-intl
 ---> Using cache
 ---> 368a8208fcef
Step 3/4 : RUN sed -i "s/;extension=intl/extension=intl/" /etc/php/8.0/apache2/php.ini
 ---> Using cache
 ---> f98793927162
Step 4/4 : CMD ["/run.sh"]
 ---> Using cache
 ---> ffb40be71800
Successfully built ffb40be71800
Successfully tagged joris/lamp:latest
```

- docker image ls
  - Shows a list of available images
  - Each image gets a unique ID
    - A tag is a simpler way to address the image rather than using its ID

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
joris/lamp	latest	ffb40be71800	9 hours ago	1.1GB
<none>	<none>	b185f0507e5b	10 hours ago	1.06GB

Dissecting docker commands: build

- docker build -t joris/website:latest /path/to/dockerfile
  - /path/to/dockerfile is the path to the **Dockerfile** file
  - Commands are executed from **top to bottom**
    - Uses a smart caching system to avoid rebuilding unchanged parts in the event of a rebuild
    - Order of commands is important to make optimal use of the caching system
      - We will go more in depth on this during PSOPV



```
FROM mattrayner/lamp:latest-2004-php8

RUN apt-get -y update && apt-get -y install php8.0-intl
RUN sed -i "s/;extension=intl/extension=intl/" /etc/php/8.0/apache2/php.ini

CMD ["/run.sh"]
```

  - Images are typically created **FROM** a **base layer**
    - Examples: scratch, ubuntu, debian, alpine
      - https://hub.docker.com/_/scratch
  - By adding more instructions we can tell our image what to configure and run
- docker run -p 8080:80 joris/lamp:latest
  - Runs the above image with our added changes

Dissecting docker commands: advanced run options

- docker run -p 8080:80 -v ./www:/app mattrayner/lamp:latest
  - -v /local/path:/remote/path allows for **mounting host directories** into the container
    - https://docs.docker.com/storage/volumes/
  - Each time a docker container starts, it starts **fresh** (i.e., containers are volatile)
  - -v ./www:/app
    - Mount the local ./www directory as the container /app directory in the container
  - By mounting volumes we **don't have to copy our data and rebuild our image** every time we make a change, the image stays static and simply makes use of mounted data in its container state

Dissecting docker commands: advanced run options

- docker run -p 8080:80 -v ./www:/app mattrayner/lamp:latest

```
jherbots … guest_lecture_jherbots docker_lamp www ls -al
total 8
drwxr-xr-x 2 jherbots jherbots 4096 Oct 19 22:11 .
drwxr-xr-x 5 jherbots jherbots 4096 Oct 19 22:10 ..
-rw-r--r-- 1 jherbots jherbots    0 Oct 19 22:11 hello_world.html
-rw-r--r-- 1 jherbots jherbots    0 Oct 19 22:11 index.php
```

=

```
root@19d6492c49ff:/# cd app/
root@19d6492c49ff:/app# ls -al
total 8
drwxr-xr-x 2 www-data staff 4096 Oct 19 20:11 .
drwxr-xr-x 1 root     root  4096 Oct 19 20:13 ..
-rw-r--r-- 1 www-data staff    0 Oct 19 20:11 hello_world.html
-rw-r--r-- 1 www-data staff    0 Oct 19 20:11 index.php
```

Dissecting docker commands

- docker ps
  - Shows an overview of **running containers**

```
jherbots … guest_lecture_jherbots docker_lamp www docker ps
CONTAINER ID   IMAGE               COMMAND     CREATED         STATUS         PORTS                                          NAMES
19d6492c49ff   joris/lamp:latest   "/run.sh"   5 minutes ago   Up 5 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp, 3306/tcp   adoring_blackwell
```

- docker exec -it 19d6492c49ff bash
  - Execute a command, **bash**, in this case in the running container
  - Allows us to open an interactive bash-shell on the container instance
- docker stop 19d6492c49ff
  - Halts the container
- docker rm 19d6492c49ff
  - Fully removes all remaining state after a stop
    - Docker keeps some state around in case you might want to persist a container for more advanced usecases. You can see these containers by executing *docker ps -a*

Project requirements from this lesson

There is always a catch my dear friends!

Backend framework: CodeIgniter

- **CodeIgniter 4** is a **PHP framework** that's been around for a while
  - Uses the **model-view-controller** design pattern
  - Lightweight, easy to setup and use and provides great out of the box functionality
  - Provides search engine optimized URLs by default
    - Example: http://example.com/yourcontroller/somepage/somevalue
  - The documentation is your friend: https://codeigniter.com/user_guide/index.html
- Does this mean all my previous PHP code is wasted effort?
  - No, you can perfectly copy and paste it in the MVC architecture of CodeIgniter
- Why should we waste all this time getting to know this framework?
  - No sane person reinvents the wheel… especially not when the wheel is a digital resource you access via the web

CodeIgniter

Frontend CSS framework: Bootstrap

- Twitter **Bootstrap 5.1** is a frontend CSS framework
  - We don't expect you to be a designer, but creating something pleasing to the eye is possible in 2021 with the help of a CSS framework like Bootstrap
  - Provides defaults for:
    - Responsive layouts (columns, grids, rows)
    - Form elements
    - Colors
    - …
  - Has optional JavaScript modules to create pretty things
  - Once again the documentation is your best friend: https://getbootstrap.com/docs/5.1/getting-started/introduction/

- Does this mean I have to throw away all my previous CSS?
  - No! Bootstrap is a framework that can perfectly work in harmony with your own CSS.
  - In fact, bootstrap only provides basics, you still have to design your own layout!
  - It is not allowed to use existing Bootstrap templates you find online!
    - This does not mean that you can't take inspiration from them

Docker LAMP configuration

- Finally the project will have to be developed and submitted using Docker
  - We will use the mattrayner/lamp:latest-2004-php8 image for this purpose
  - Because CodeIgniter and the above image don't play nicely out of the box, I made a new Dockerfile which you can use for this project
    - Will be provided for download or you can copy the next slide
- A few tips on mattrayner/lamp
  - Here is some helpful information in case you need it https://github.com/mattrayner/docker-lamp
  - You can can modify the database through a pre-installed phpmyadmin which you can access via http://localhost:8080/phpmyadmin
    - Username: admin
    - Password is generated during the first run of the container, look in your console for the key and save it somewhere!

```
================================================================================
You can now connect to this MySQL Server with uQrP5vV9vmwJ

    mysql -uadmin -puQrP5vV9vmwJ -h<host> -P<port>

Please remember to change the above password as soon as possible!
MySQL user 'root' has no password but only allows local connections

enjoy!
================================================================================
```

Docker LAMP configuration

- Username: admin
- Password is generated during the first run of the container, look in your console for the key and save it somewhere!

- You do not need a password for accessing the MySQL database from your php code, use:
  - Username: root
  - Password is empty

- In order to build the provides Dockerfile, use the following command in the directory of the Dockerfile:
  - docker build -t yourImageTag .
- To run your container, use:
  - docker run -p "8080:80" -v ${PWD}/codeigniter:/app -v ${PWD}/mysql:/var/lib/mysql yourImageTag
  - This will
    - Port forward the Apache webserver to http://localhost:8080
    - Mount the codeigniter directory in the container (the provided Dockerfile only works with CodeIgniter)
    - Mount the mysql directory in the container so your database is persistent
      - Empty the mysql directory to wipe the whole database OR change the host directory ${PWD}/mysql to something else
      - You can have multiple databases for testing this way!

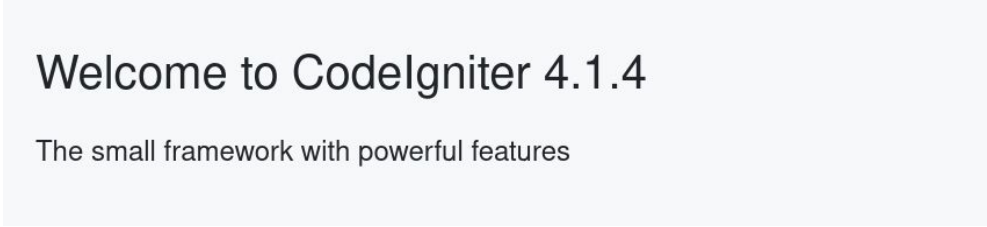
Dockerfile

Dockerfile > ...

```
1   # Web Programming project base from mattrayner/lamp
2   # Ubuntu 20.04 base-layer with PHP 8
3   FROM mattrayner/lamp:latest-2004-php8
4
5   # CodeIgniter requires intl module
6   RUN apt-get -y update && apt-get -y install php8.0-intl
7   RUN sed -i "s/;extension=intl/extension=intl/" /etc/php/8.0/apache2/php.ini
8
9   # Fix symlinks for CodeIgniter 4 security model
10  # .htaccess rewrite by default enables FollowSymlinks
11  RUN rm /var/www/html
12  RUN ln -s /app/public /var/www/html
13  RUN sed -i "s/<Directory \/>/<Directory \/var\/www>/" /etc/apache2/sites-available/000-default.conf
14
15  # mattrayner/lamp original boot code
16  CMD ["/run.sh"]
```

Exercises for today

1. Read the documentation on Twitter Bootstrap and try out some features to see if it works (I recommend playing around with layouting and buttons, those are fun)
2. Populate the empty *codeigniter* with a download of the framework, build your first Docker image using the provided Dockerfile and check if you can see the CodeIgniter welcome page



3. Follow the CodeIgniter guide on building your first application to get a feel for the framework

SO LONG
AND
THANKS
FOR ALL
THE FISH

Images

Icons from Freepik

- Flaticons
- xnimrodx

Logos from their respective websites.