UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

WEB TECHNOLOGIES — LECTURE 09

WELCOME TO THE SERVER-SIDE OF THE WEB

Luigi Libero Lucio Starace, PhD

luigiliberolucio.starace@unina.it

https://luistar.github.io

https://www.docenti.unina.it/luigiliberolucio.starace

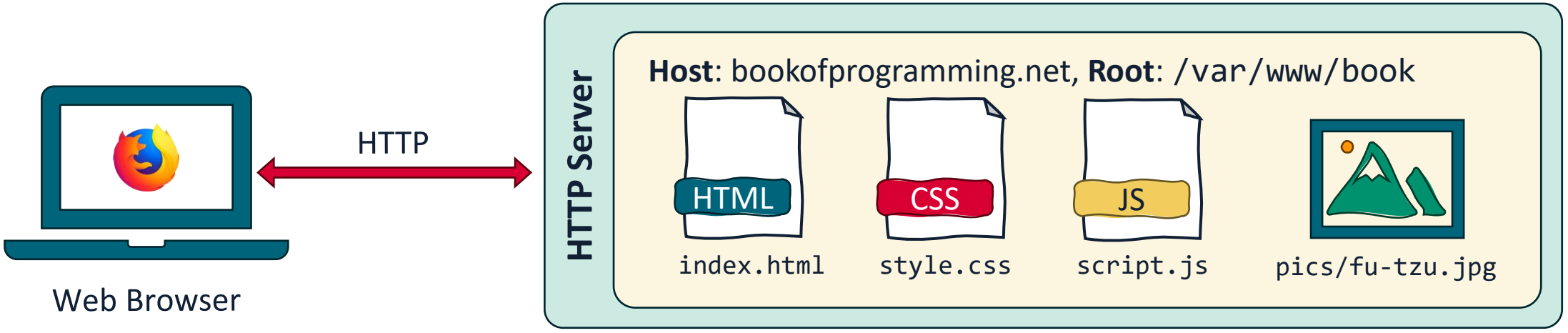
PREVIOUSLY, ON WEB TECHNOLOGIES

So far, we've learned a good deal about the **core web technologies**: **HTML**, **CSS**, and **JavaScript**. Let's contemplate our progress so far:

- We learned to design **modern**, **beautiful**, **responsive** web pages.
- We learned to make web pages **dynamic**, using JavaScript:
  - Handling events, dynamically changing the DOM
  - Making async HTTP network requests to fetch data to show in our pages

STATIC WEB APPS

- Our web pages can have a certain dynamism, thanks to JavaScript
- That dynamism happens **inside** the web browsers
- Nonetheless, in a way, they are still inherently **static**
- Everytime a Browser sends a request to one of our web pages, it will always be served the **same**, **identical** document



LIMITATIONS OF STATIC WEB APPS

- The «static» web approach we've seen so far is simple and effective
- It works great for web applications consisting of limited numbers of pages, that change quite rarely. It is affected by some limitations:

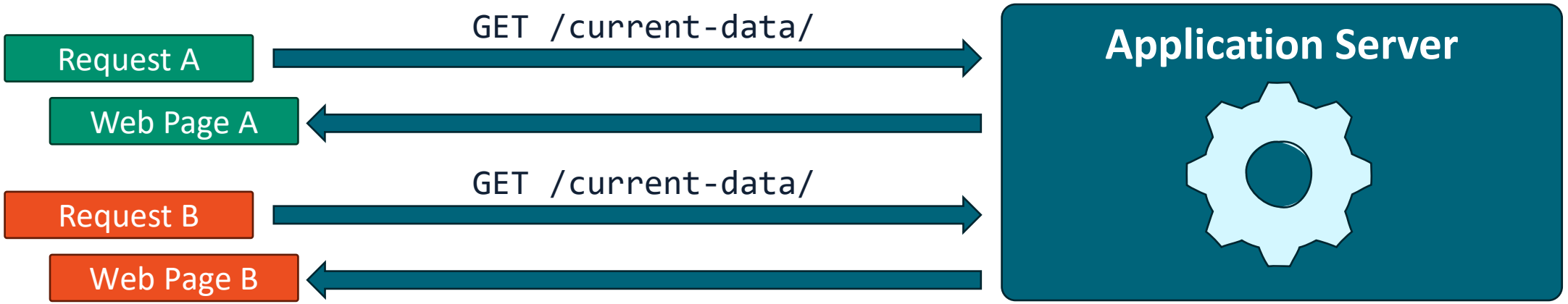
- What if we want to **personalize** a page for a specific user?
- What if the **content** of a page **changes** very frequently?
- What if our website consists of **millions** of pages?
- What if we need to **act** based on a request (e.g.: save an order, …)?

AN EXAMPLE: WIKIPEDIA

- Wikipedia features **~60 million** web pages
  - All with the same structure, same style…
  - …but different contents
- Let’s say the base structure in a Wikipedia page is **~150k chars**
- That’s **150kB** of data **repeated** in **every** page
- Multiply that for the 60 million web pages, you get **9 TeraBytes** of repeated data
- What if we need to change the navbar?

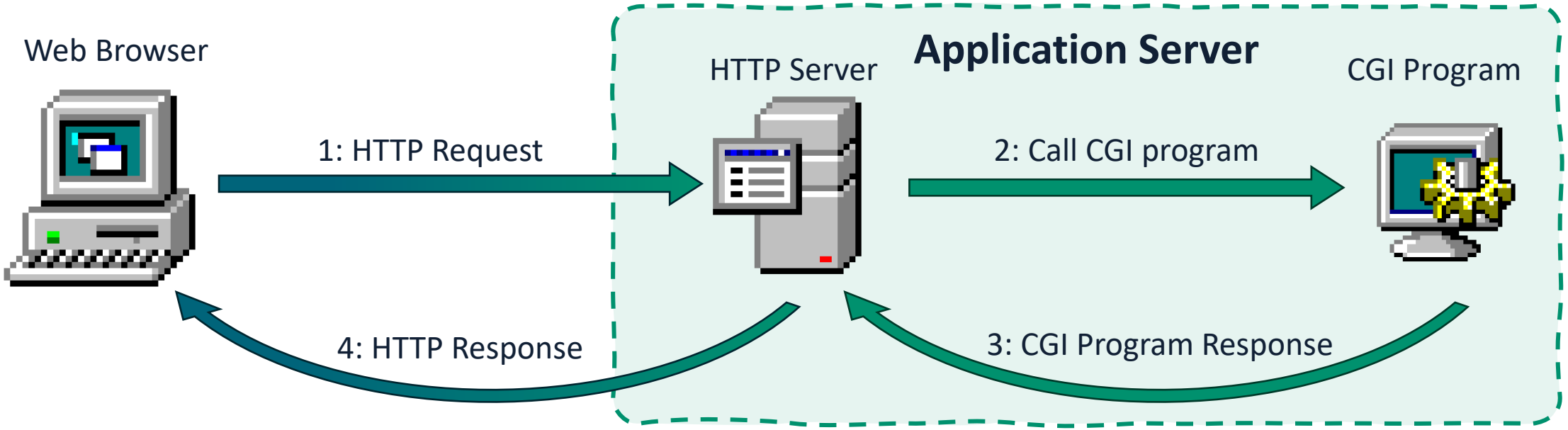
INTRODUCING SERVER–SIDE PROGRAMMING

- Until now, web servers just serve files from the **document root**
- A web server can do much more!
- For a starter, it can **generate** web pages **on-the-fly**
  - Typically based on parameters received in the HTTP request
- That's the key idea behind **server-side programming**



SERVER–SIDE PROGRAMMING: CGI

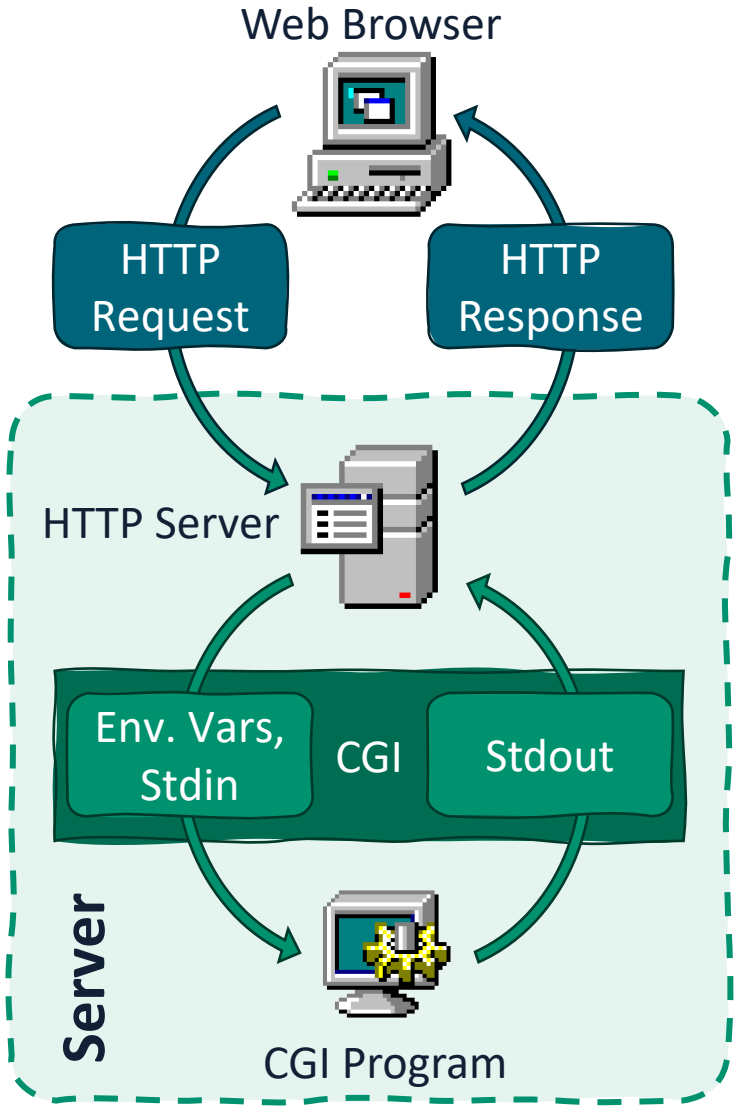
- In the early 1990, before JavaScript even existed, dynamic web pages could be achieved using the **Common Gateway Interface** (**CGI**)
- CGI is an interface specification allowing web servers to execute external programs to process an HTTP request



THE COMMON GATEWAY INTERFACE

Defines how web servers and CGI-compliant programs communicate

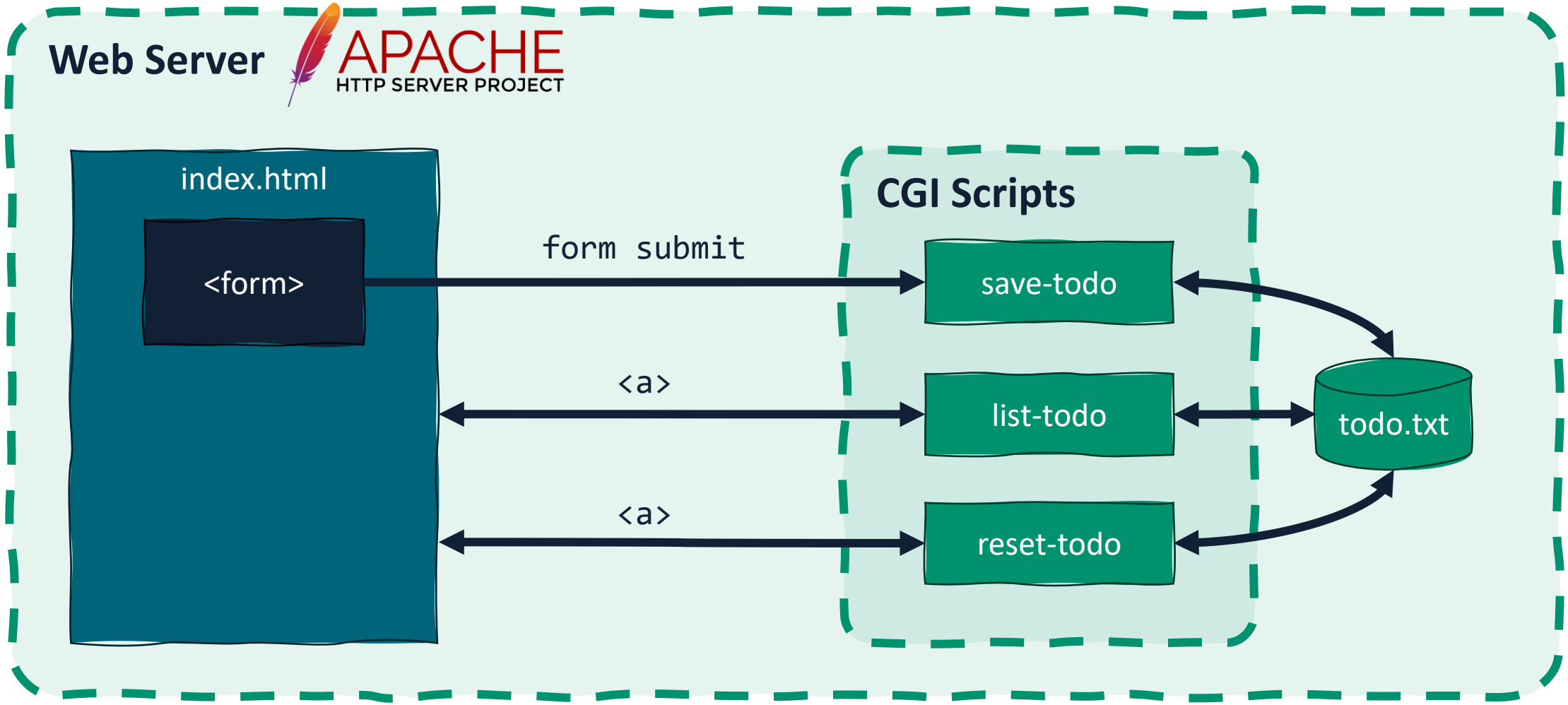
- How a CGI program should access its inputs
  - Request headers and Query String are available via **Environment Variables**
  - Request body is available to the program on the **stdin** stream
- How a web server should build a response
  - The program output on **stdout** is the body of the HTTP response



OUR VERY FIRST CGI SCRIPT

```
#!/bin/bash
echo "Content-type: text/html; charset=utf-8"
echo ""
echo "<!DOCTYPE html>"
echo "<html><head><title>Hello CGI!</title></head><body>"
echo "<h1>Hello CGI!</h1>"
echo "<p>This page was loaded on "
date +"%d/%m/%Y at %H:%M:%S. </p></body>"
```

EXAMPLE: CGI–BASED TODO LIST WEB APP



EXAMPLE: CGI-BASED TO-DO LIST WITH BASH

- Live demo time!
- Repo: https://github.com/luistar/cgi-to-do-list-with-bash
- Will the teacher make it through the first demo of the course?

SERVER–SIDE SCRIPTING

- Writing CGI programs is not a very pleasant experience
  - Writing entire HTML documents to Stdout is cumbersome
  - Manually parsing query strings and request bodies is not much fun
  - Spawning a new process for each request is not very resource effective
- Luckily, specialized programming languages and frameworks emerged
  - **PHP** (recursive acronym for PHP Hypertext Preprocessor), in 1995
  - **ASP** (Active Server Pages), Microsoft's server-side scripting language, in 1997
  - **JSP** (Java Server Pages), originally released by Sun in 1999
- The idea was to «**mix**» server-side code and HTML, with a special **interpreter** parsing the mix to produce «pure» HTML code

SERVER–SIDE SCRIPTING: PHP

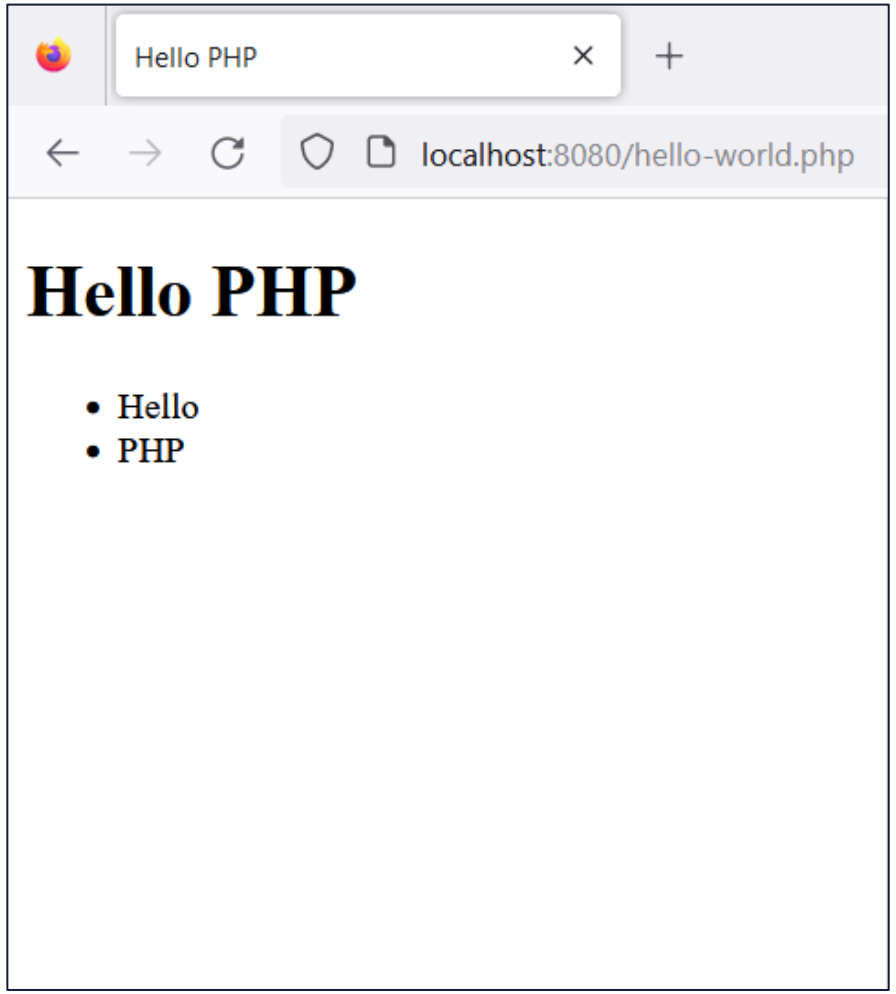
- Special **delimiters** `<?php ... ?>` are used to **separate** code and HTML

```
<!-- hello-world.php file -->
<!DOCTYPE html>
<html>
  <head>
    <title><?php echo "Hello PHP" ?></title>
  </head>
  <body>
    <h1>Hello PHP</h1>
    <?php $array = ["Hello", "PHP"] ?>
    <ul>
      <?php foreach($array as $item): ?>
        <li><?= $item ?></li>
      <?php endforeach; ?>
    </ul>
  </body>
</html>
```

- Text outside the delimiters remains **as-is** in the output
- Code within the delimiters is **interpreted**, and the resulting output is inserted in the html
- `<?=` is a shorthand for `<?php echo`

SERVER–SIDE SCRIPTING: PHP

```php
<!-- hello-world.php file -->
<!DOCTYPE html>
<html>
  <head>
    <title><?php echo "Hello PHP" ?></title>
  </head>
  <body>
    <h1>Hello PHP</h1>
    <?php $array = ["Hello", "PHP"] ?>
    <ul>
      <?php foreach($array as $item): ?>
        <li><?= $item ?></li>
      <?php endforeach; ?>
    </ul>
  </body>
</html>
```



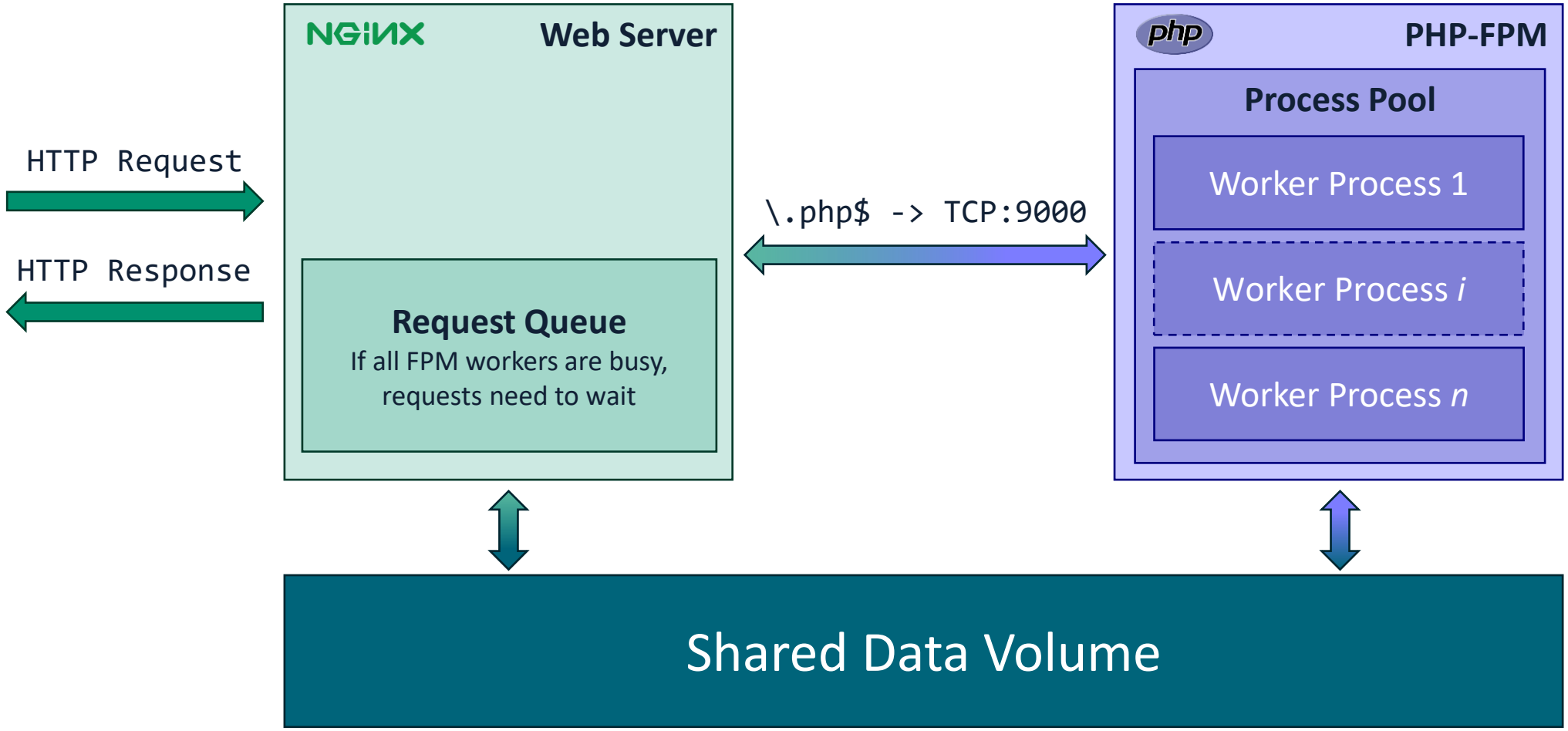
SERVER–SIDE SCRIPTING: PHP

- Server-side scripting solutions typically support developers in many tedious tasks typical of HTTP request handling
- Requests are automatically **pre-processed**
  - In PHP, request parameters are available in associative arrays (e.g.: `$_GET`, `$_POST`).
  - For example, `$_POST['todo']` can be used to access the `todo` request parameter.
  - In PHP, cookies are already parsed in the `$_COOKIES` associative array
- No need to explicitly output on stdout the entire response

EXAMPLE: PHP–BASED TO–DO LIST

- We will re-implement our To-do list web app using PHP 8
- Since we're making a leap forward towards modern web development, we'll make our app slightly more interesting
  - We'll manage persistency using an actual database (SQLite)
  - We'll use Bootstrap to make our pages slightly less ugly and responsive
  - We'll introduce a footer and a navbar, as template parts
- We'll use a modern architecture leveraging NGINX and PHP-FPM

EXAMPLE: PHP–BASED TO–DO LIST



EXAMPLE: PHP TO-DO LIST

- Live demo time! (again!)
- Repo: https://github.com/luistar/php-to-do-list

REFERENCES (1/2)

- **Server-side programming**
  MDN web docs
  https://developer.mozilla.org/en-US/docs/Learn/Server-side
  https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps
  https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction
  https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview
- **CGI**
  IBM Docs
  https://www.ibm.com/docs/en/i/7.5?topic=functionality-cgi
  RFC 3875: The Common Gateway Interface (CGI) Version 1.1 (2004)
  https://datatracker.ietf.org/doc/html/rfc3875
  Apache HTTPd documentation (in case you want to take a look at how to configure a web server for CGI)
  https://httpd.apache.org/docs/2.4/howto/cgi.html

REFERENCES (2/2)

- **PHP**
  Official website
  https://www.php.net/
  Getting started page
  https://www.php.net/manual/en/getting-started.php
  Official manual
  https://www.php.net/manual/en/

  ⚠️ You are **not** required to learn the PHP language for the Web Technologies course! All you need to know is what's included in the slides. In case you want to learn PHP on your own, the official manual is a great starting point.