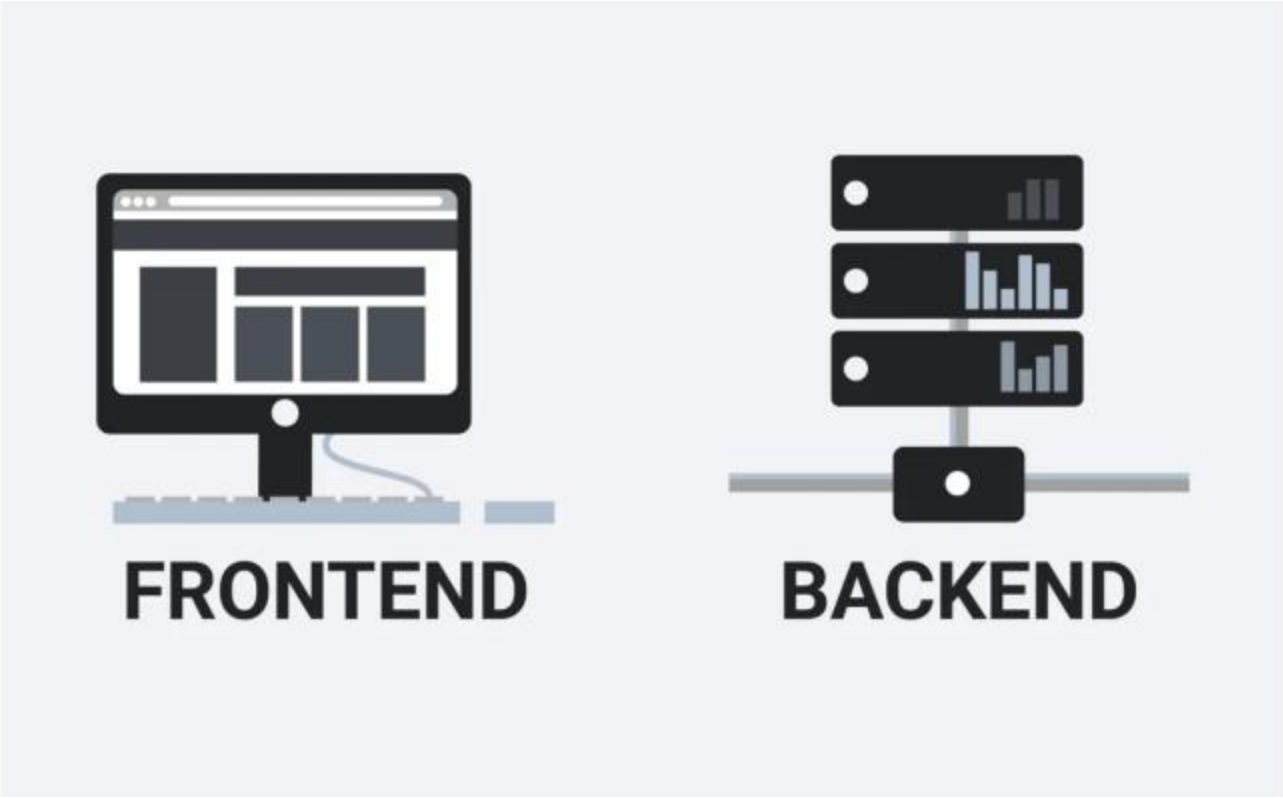
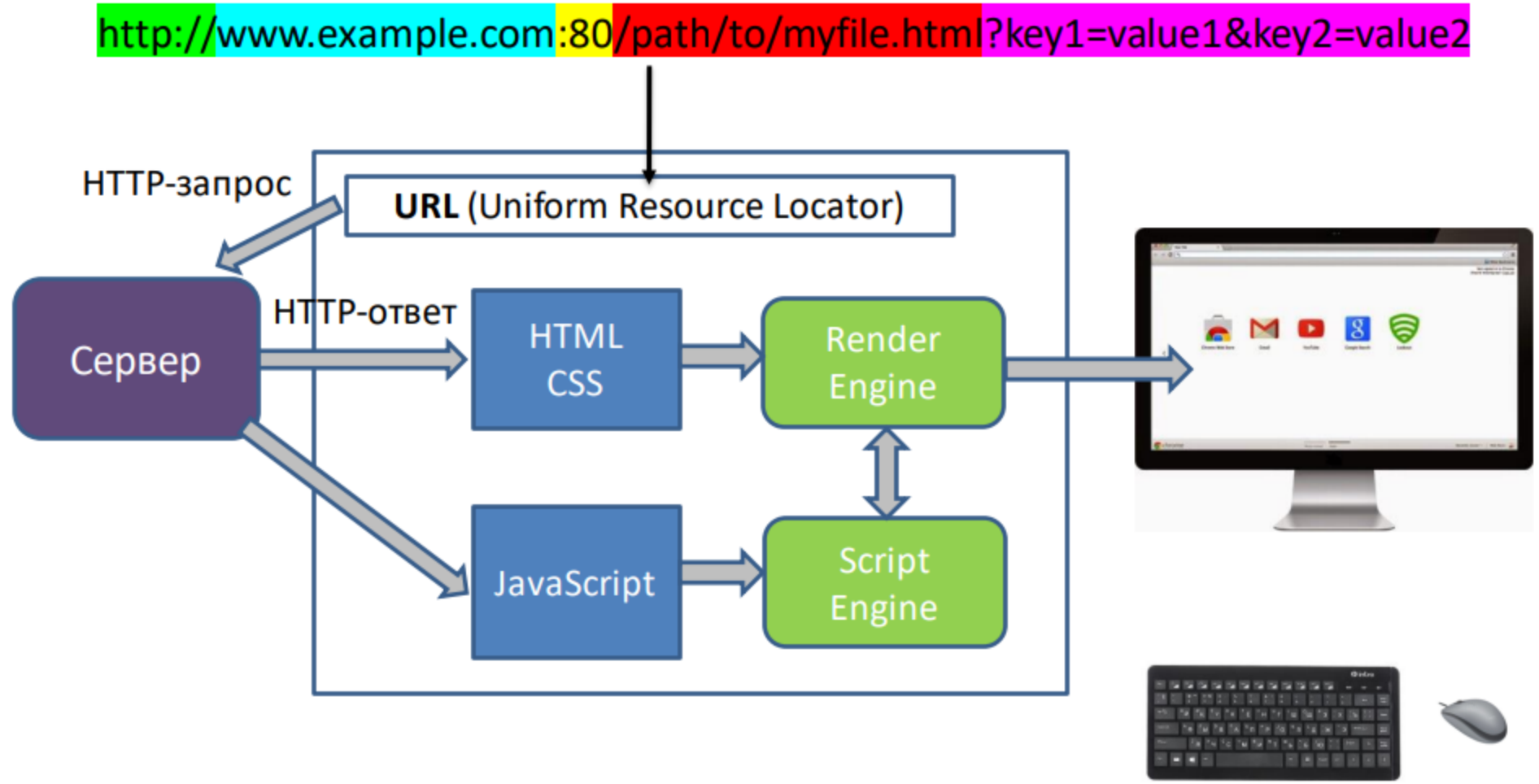
7. Подходы к созданию и запуску веб-приложений. Стандартный PHP как инструмент веб-разработки

Веб-приложение

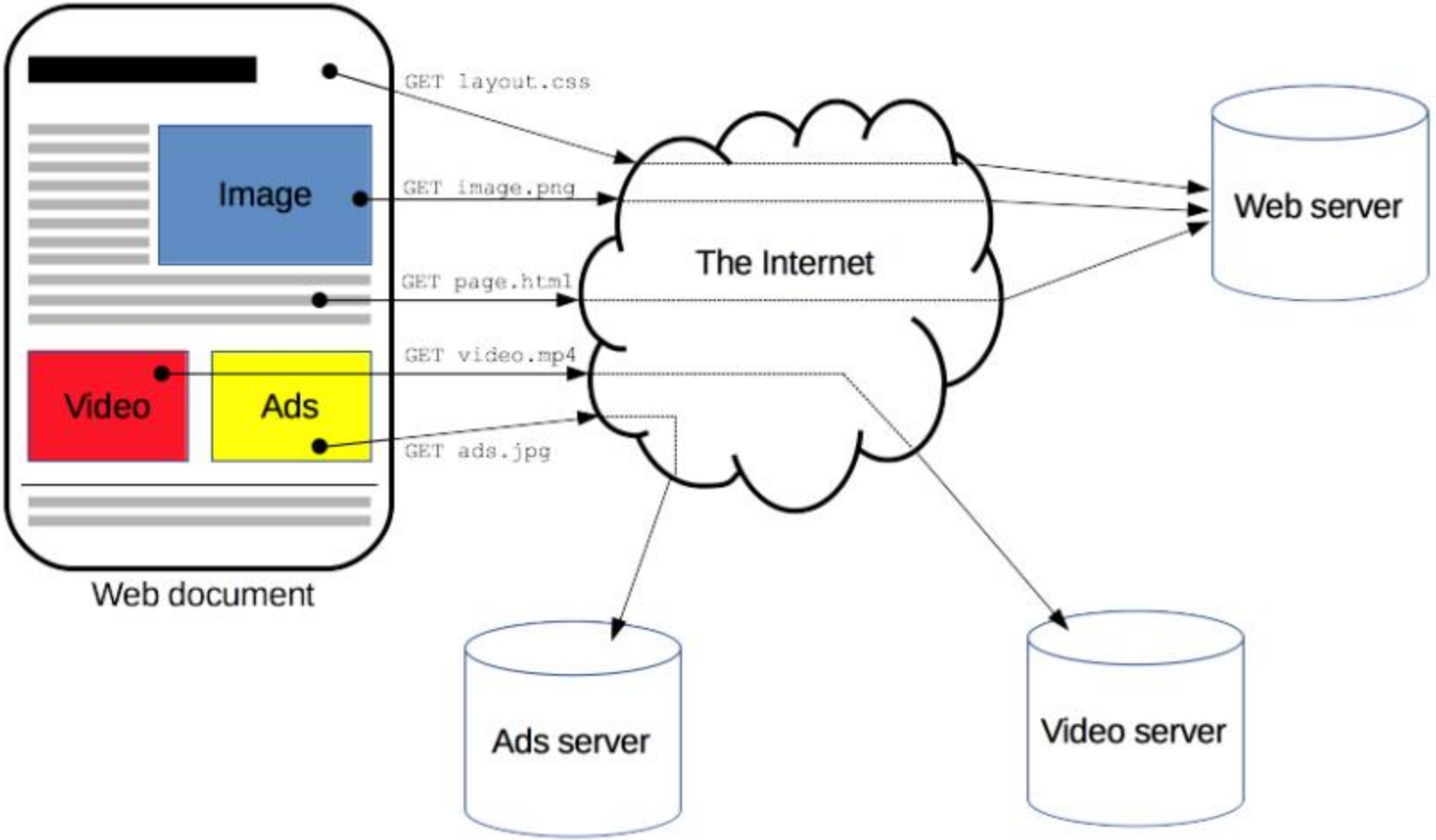


Frontend

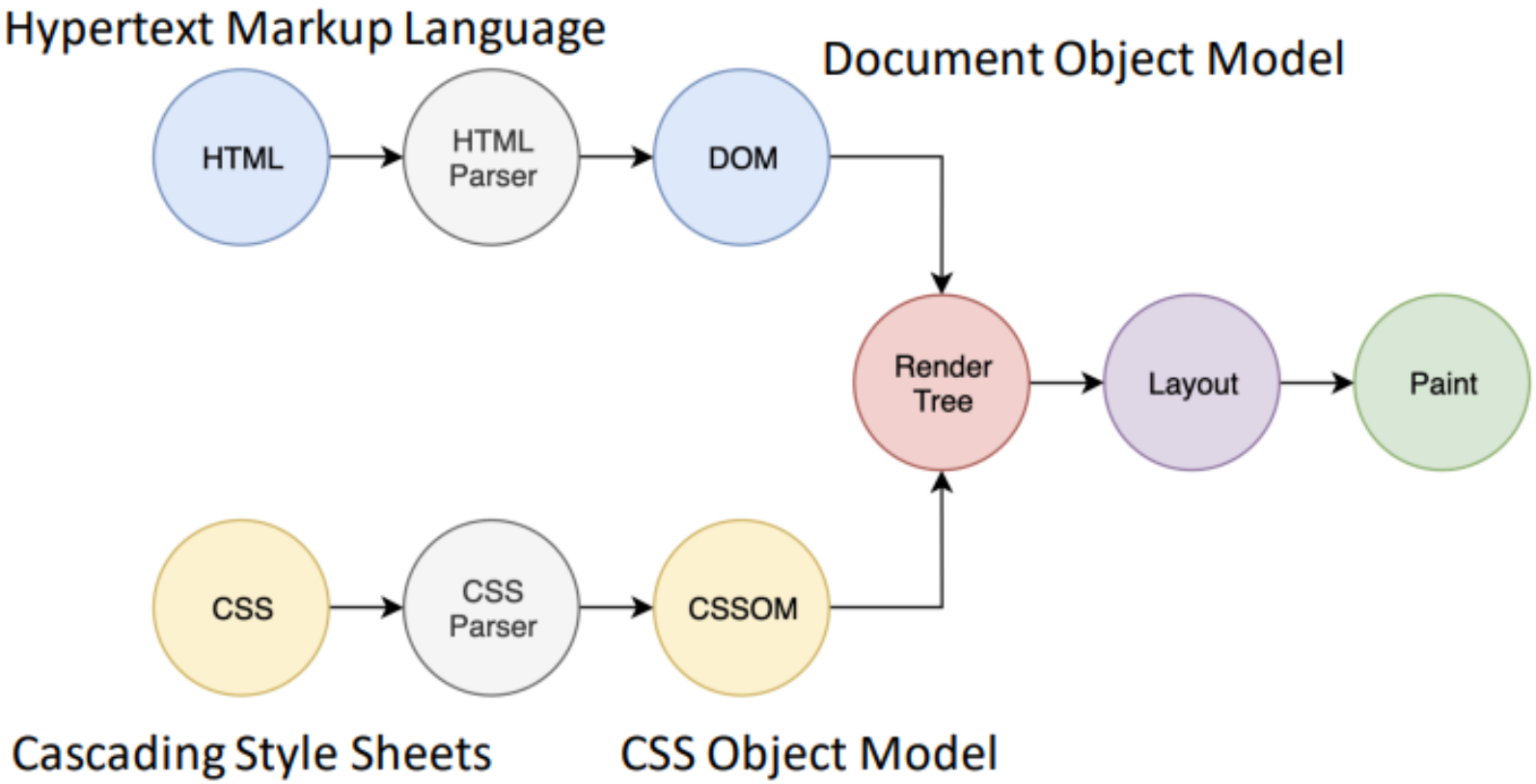
Frontend


http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2
HTTP-запрос
URL (Uniform Resource Locator)
Сервер
HTTP-ответ
HTML
CSS
Render
Engine
JavaScript
Script
Engine

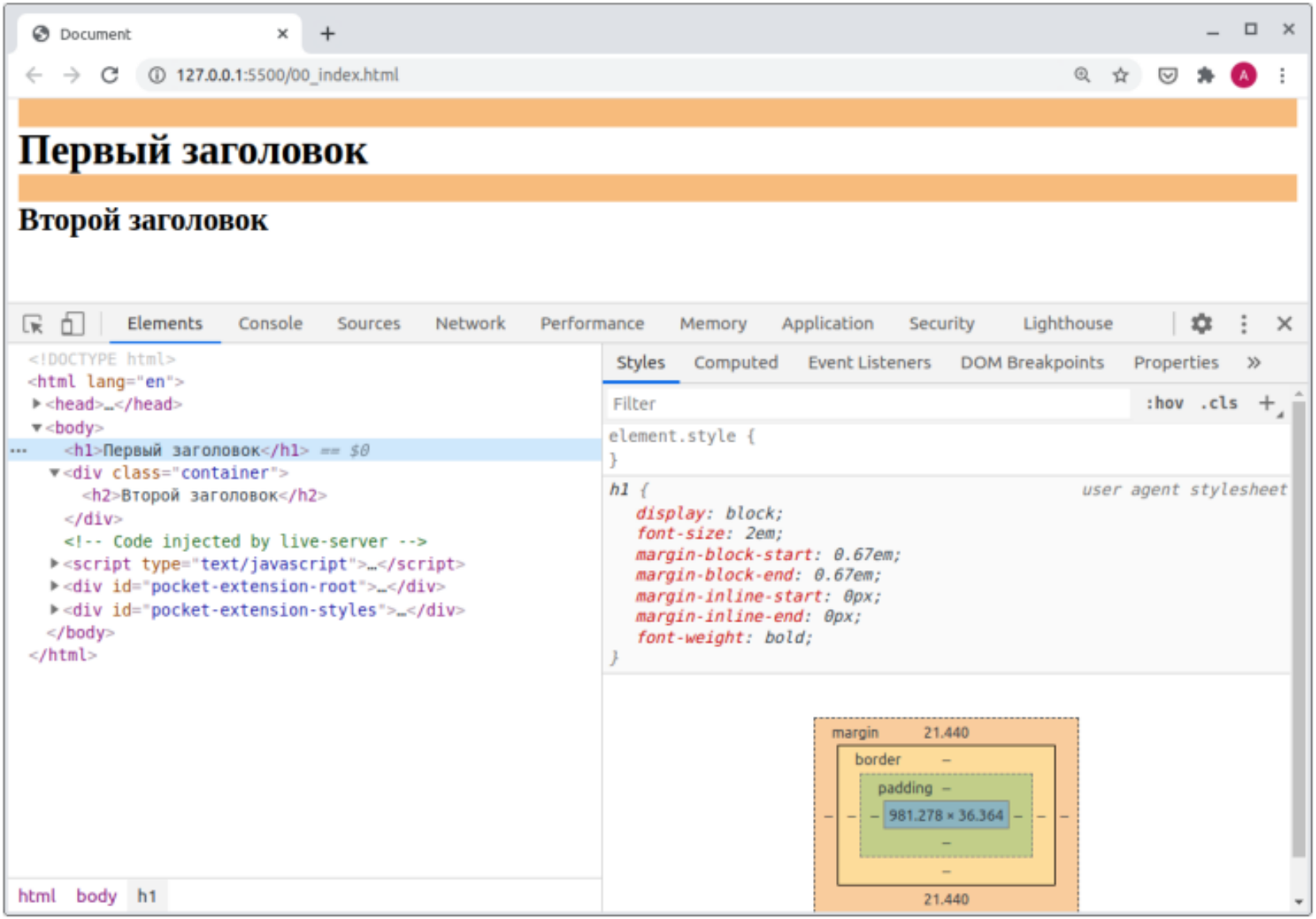
Документ в браузере



Рендеринг веб-страницы



DOM и стили


Document
127.0.0.1:5500/00_index.html
Первый заголовок
Второй заголовок
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
<!DOCTYPE html>
<html lang="en">
<head>…</head>
<body>
<h1>Первый заголовок</h1> == $0

<h2>Второй заголовок</h2>

<!-- Code injected by live-server -->
<script type="text/javascript">…</script>
…
…
</body>
</html>
Styles
Computed
Event Listeners
DOM Breakpoints
Properties
Filter
:hov .cls
element.style {
}
h1 {
user agent stylesheet
display: block;
font-size: 2em;
margin-block-start: 0.67em;
margin-block-end: 0.67em;
margin-inline-start: 0px;
margin-inline-end: 0px;
font-weight: bold;
}
margin 21.440
border –
padding –
981.278 × 36.364
21.440
html
body
h1

JavaScript в браузере

Может

- Изменять DOM, модифицировать стили
- Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш
- Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы
- Запоминать данные на стороне клиента
- Получать и устанавливать куки, показывать сообщения

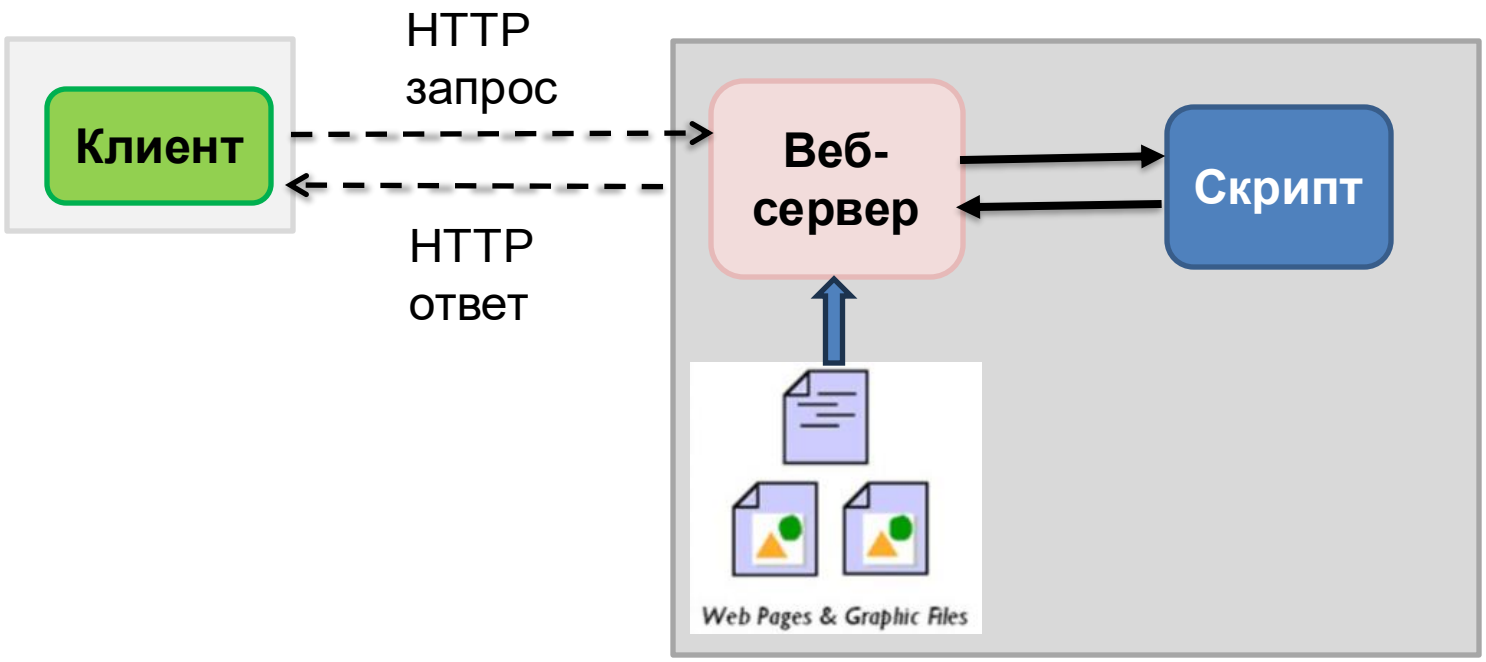
Не может

- Читать/записывать произвольные файлы на диске
- Напрямую работать с локальными СУБД

Backend

Статический и динамический контент

Статический и динамический контент



- **Статика** возвращается напрямую, если файл существует.
- **Динамика** запускается, если:
  - Файл имеет расширение скрипта (`.php` , `.py`).
  - Конфигурация сервера явно указывает на обработчик.
  - Используются правила маршрутизации (например, `index.php` во фреймворках).

MIME-типы ресурсов

MIME-типы (Multipurpose Internet Mail Extensions) — это стандарт, определяющий, как браузеры и серверы должны обрабатывать различные типы файлов. Они указывают, какой вид данных содержится в файле или передаётся по сети (например, текст, изображение, аудио, видео).

MIME-тип состоит из двух частей:

1. **Тип данных** (например, `text`, `image`, `application`).
2. **Подтип** (например, `plain`, `html`, `json`).

Примеры:

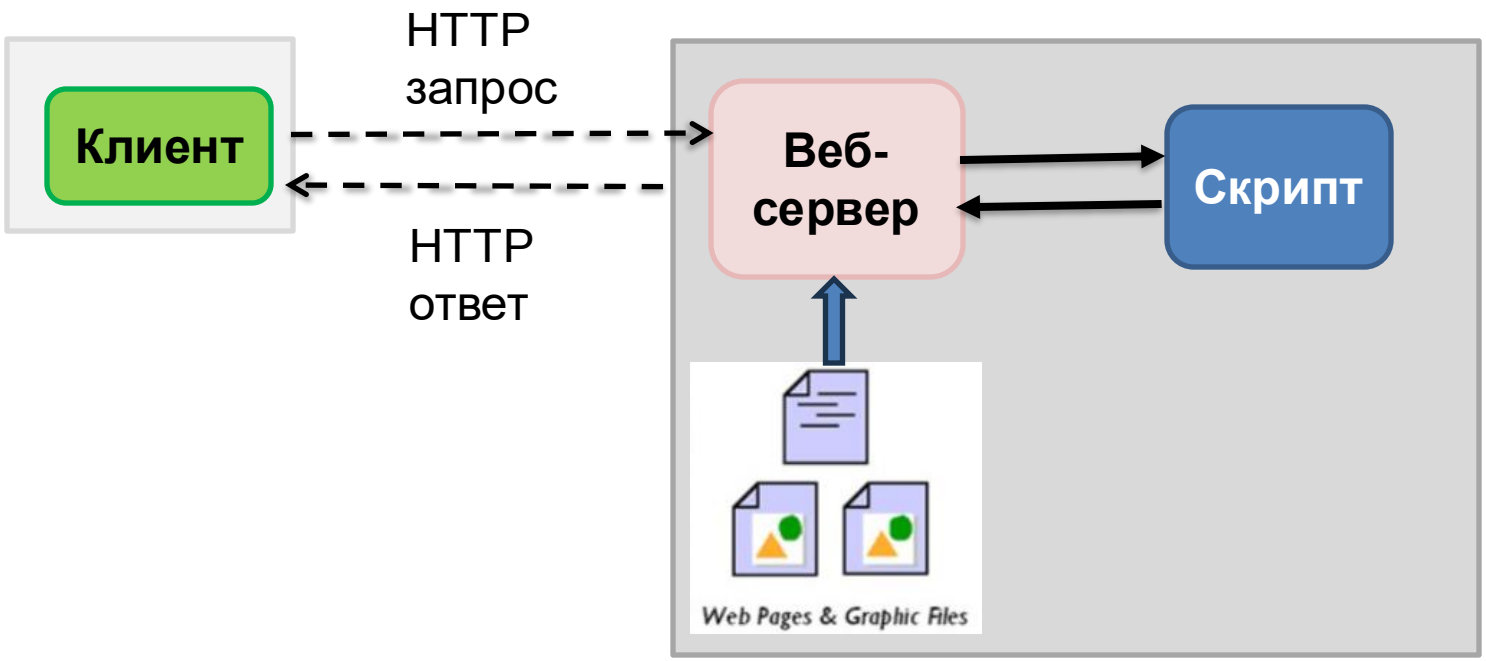
- `text/css` — CSS-файл.
- `application/javascript` — JavaScript-код.
- `video/mp4` — видео в формате MP4.

MIME-типы ресурсов

4. Как работают MIME-типы?

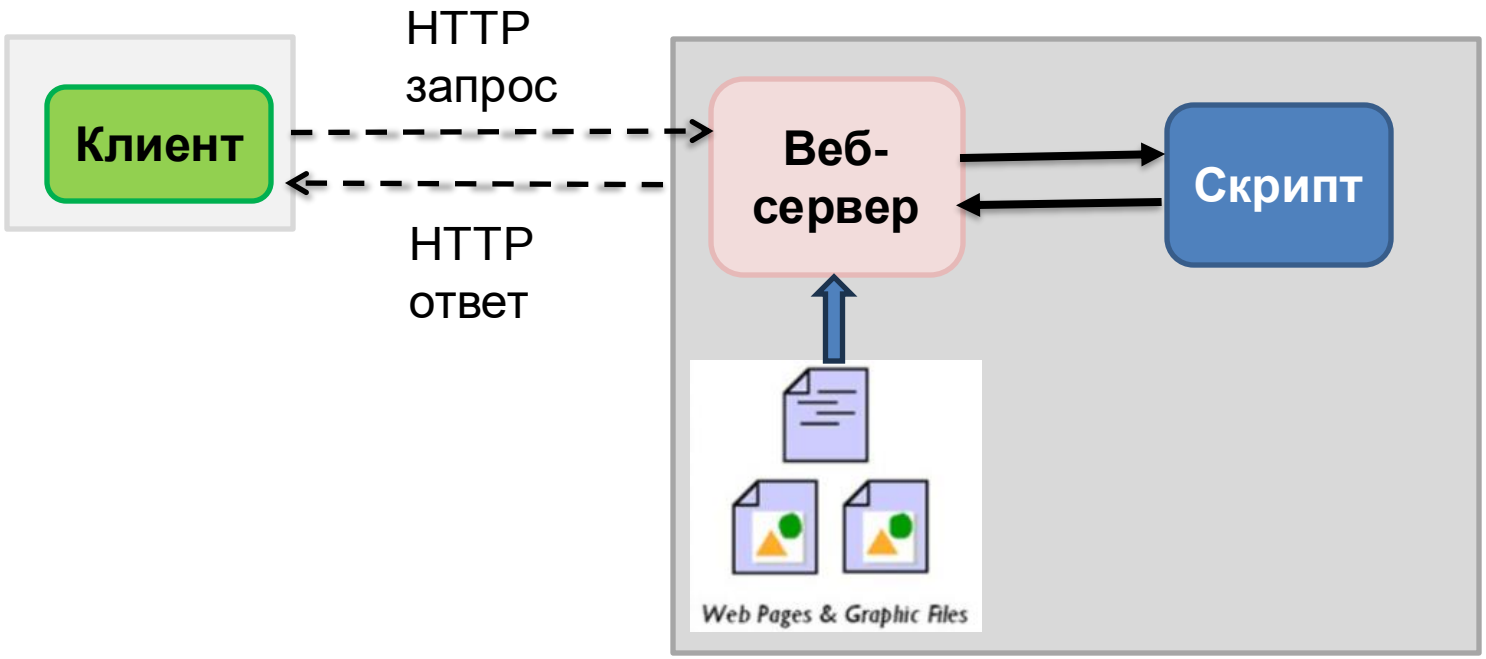
1. **Сервер** определяет MIME-тип файла по его расширению (например, `.html` → `text/html`).
2. **Браузер** использует MIME-тип для:
   - Отображения содержимого (например, рендеринг HTML).
   - Запуска плагинов (например, PDF-ридер для `application/pdf`).
   - Обработки загрузок (например, сохранение файла как `image/png`).

Статический и динамический контент



- **Веб-сервер**: прием HTTP-запроса от клиента, выбор действий на основании своих настроек и типа запрашиваемого ресурса.
- **Веб-сервер**: отдача статики клиенту статики по HTTP или запуск скрипта и передача ему параметров и тела HTTP-запроса.
- **Скрипт**: получение параметров и тела запроса от сервера, формирование HTTP-ответа, передача его серверу.
- **Веб-сервер**: прием результата от скрипта, передача его по HTTP клиенту с указанием MIME-типа.

Статический и динамический контент



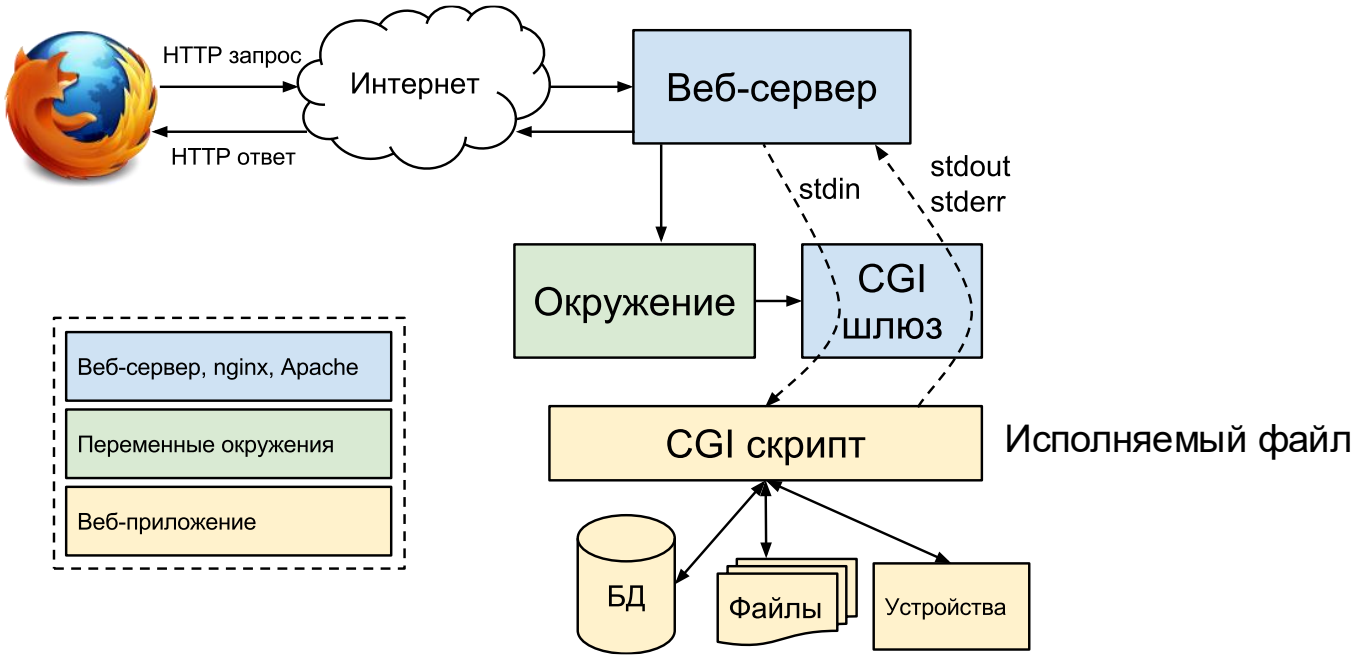
- Сервер настроен на выполнение PHP-скриптов.
- Пришел GET-запрос к существующему файлу /1.html.
- Пришел GET-запрос к существующему файлу /2.php.

Какие MIME-типы укажет сервер в HTTP-ответах на эти запросы?

Механизмы запуска веб-сервером скриптов

Common Gateway Interface (CGI)

Протокол CGI - запуск внешнего процесса



- Независимость от языка скрипта
- При каждом HTTP-запросе веб-сервер запускает новый процесс - работает медленно

Протокол CGI

Для каждого HTTP-запроса **веб-сервер** (Apache):

- Устанавливает переменные окружения, связанные с параметрами и заголовками этого запроса (QUERY_STRING, REQUEST METHOD, CONTENT_TYPE, CONTENT_LENGTH и т.д.)
- Помещает тело запроса в поток stdin
- Силами операционной системы запускает под своей учетной записью нужный исполняемый файл (может быть бинарным или интерпретируемым).
- Извлекает из stdout выполнившегося скрипта строки, формирующие HTTP-ответ.
- Передает HTTP-ответ клиенту.

Протокол CGI

Задачи скрипта:

- Прочитать переменные окружения, связанные с параметрами и заголовками обрабатываемого HTTP-запроса (QUERY_STRING, REQUEST METHOD, CONTENT_TYPE, CONTENT_LENGTH и т.д.).
- Прочитать тело полученного запроса из потока stdin.
- Распарсить Query String и тело запроса.
- Сформировать HTTP-ответ (статус, заголовки, тело ответа в формате HTML) и поместить его в stdout.

Протокол CGI

Задачи скрипта:

- Прочитать переменные окружения, связанные с параметрами и заголовками обрабатываемого HTTP-запроса (QUERY_STRING, REQUEST METHOD, CONTENT_TYPE, CONTENT_LENGTH и т.д.).
- Прочитать тело полученного запроса из потока stdin.
- Распарсить Query String и тело запроса.
- Сформировать HTTP-ответ (статус, заголовки, тело ответа в формате HTML) и поместить его в stdout.

PHP делает черновую работу за нас!

Поддержка HTTP в PHP (CGI)

Поддержка HTTP в PHP

```
→  ~ telnet localhost 3000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
POST /index.php?a=1&b=2 HTTP/1.1
Host: localhost:3000
Content-type: application/x-www-form-urlencoded
Content-length: 7

c=3&d=4
```

```
------ $_Server -------
Array
(
    [DOCUMENT_ROOT] => /home/andrey/projects/test
    [REMOTE_ADDR] => 127.0.0.1
    [REMOTE_PORT] => 53028
    [SERVER_SOFTWARE] => PHP 7.4.9 Development Server
    [SERVER_PROTOCOL] => HTTP/1.1
    [SERVER_NAME] => localhost
    [SERVER_PORT] => 3000
    [REQUEST_URI] => /?a=1&b=2
    [REQUEST_METHOD] => POST
    [SCRIPT_NAME] => /index.php
    [SCRIPT_FILENAME] => /home/andrey/projects/test/index.php
    [PHP_SELF] => /index.php
    [QUERY_STRING] => a=1&b=2
    [HTTP_HOST] => localhost:3000
    [CONTENT_TYPE] => application/x-www-form-urlencoded
    [HTTP_CONTENT_TYPE] => application/x-www-form-urlencoded
    [CONTENT_LENGTH] => 7
    [HTTP_CONTENT_LENGTH] => 7
    [REQUEST_TIME_FLOAT] => 1606400111.5033
    [REQUEST_TIME] => 1606400111
)
```

```
------ $_GET -------
Array
(
    [a] => 1
    [b] => 2
)
------ $_POST -------
Array
(
    [c] => 3
    [d] => 4
)
```

Поддержка HTTP в PHP - суперглобальные массивы

`$_GET`

- Содержит параметры из **URL-строки** (методы GET, HEAD, DELETE и др.).
- Пример URL: `http://example.com/?name=John&age=30`

```php
echo $_GET['name']; // "John"
```

`$_POST`

- Содержит данные из **тела запроса** (методы POST, PUT, PATCH), если:
  - Заголовок `Content-Type` равен `application/x-www-form-urlencoded` или `multipart/form-data`.

```php
// Для POST-запроса с данными name=John&age=30
echo $_POST['age']; // "30"
```

`$_REQUEST`

- Объединяет данные из `$_GET`, `$_POST` и `$_COOKIE`.
- Порядок объединения зависит от настройки `variables_order` в `php.ini`.

```php
echo $_REQUEST['name']; // Работает для GET или POST
```

Поддержка HTTP в PHP - суперглобальные массивы

`$_REQUEST`

- Объединяет данные из `$_GET`, `$_POST` и `$_COOKIE`.
- Порядок объединения зависит от настройки `variables_order` в `php.ini`.

```php
echo $_REQUEST['name']; // Работает для GET или POST
```

`$_SERVER`

- Содержит информацию о сервере и **заголовках HTTP-запроса**:

```php
echo $_SERVER['REQUEST_METHOD']; // "GET", "POST" и т.д.
echo $_SERVER['HTTP_USER_AGENT']; // User-Agent браузера
echo $_SERVER['REMOTE_ADDR']; // IP-адрес клиента
```

`$_FILES`

- Содержит данные о загруженных файлах (метод POST с `multipart/form-data`):

```php
// Для <input type="file" name="avatar">
echo $_FILES['avatar']['name']; // Имя файла
```

Поддержка HTTP в PHP - тело запроса

2. Чтение тела HTTP-запроса

Для доступа к **сырым данным** (например, JSON, XML, PUT/DELETE) используйте:

`php://input`

- Поток для чтения **сырого тела запроса** :

```php
$raw_data = file_get_contents('php://input');
// Для JSON:
$data = json_decode($raw_data, true);
```

Ограничения :

- Не работает для `multipart/form-data` (используйте `$_POST` и `$_FILES`).
- Можно читать только один раз (данные не кэшируются).

Поддержка HTTP в PHP - тело запроса

POST-запрос (JSON)

```php
// Тело запроса: {"login": "user", "password": "123"}
$raw = file_get_contents('php://input');
$data = json_decode($raw, true);
echo $data['login']; // "user"
```

PUT-запрос

```php
// Тело запроса: name=John&age=30
parse_str(file_get_contents('php://input'), $put_data);
echo $put_data['name']; // "John"
```

Создание тела ответа в PHP

```
<?PHP

$firstName = 'Arya';
$lastName = 'Stark';

?>

<html>
  <head>
  <title>Hello Word - First CGI Program</title>
  </head>
    <body>
      <h2>Hello <?= $first_name ?> <?= $last_name ?></h2>
    </body>
</html>
```

PHP сам по себе является шаблонизатором - в HTML можно вставлять данные из PHP-кода.

Явное разделение кода и HTML-блоков.

Альтернативный синтаксис PHP

Фигурные скобки ({}) заменяются на ключевые слова вроде `endif;` или `endforeach;` Это делает код визуально понятнее в шаблонах.

Обычный синтаксис

```php
<?php if ($condition) { ?>
  <p>Условие выполнено</p>
<?php } else { ?>
  <p>Условие не выполнено</p>
<?php } ?>
```

Альтернативный синтаксис

```php
<?php if ($condition): ?>
  <p>Условие выполнено</p>
<?php else: ?>
  <p>Условие не выполнено</p>
<?php endif; ?>
```

Альтернативный синтаксис PHP

```
<?php echo $string; ?>
<?= $string ?>
```

```
<?php foreach ($array as $item): ?>
<li><?=$item?></li>
<?php endforeach; ?>
```

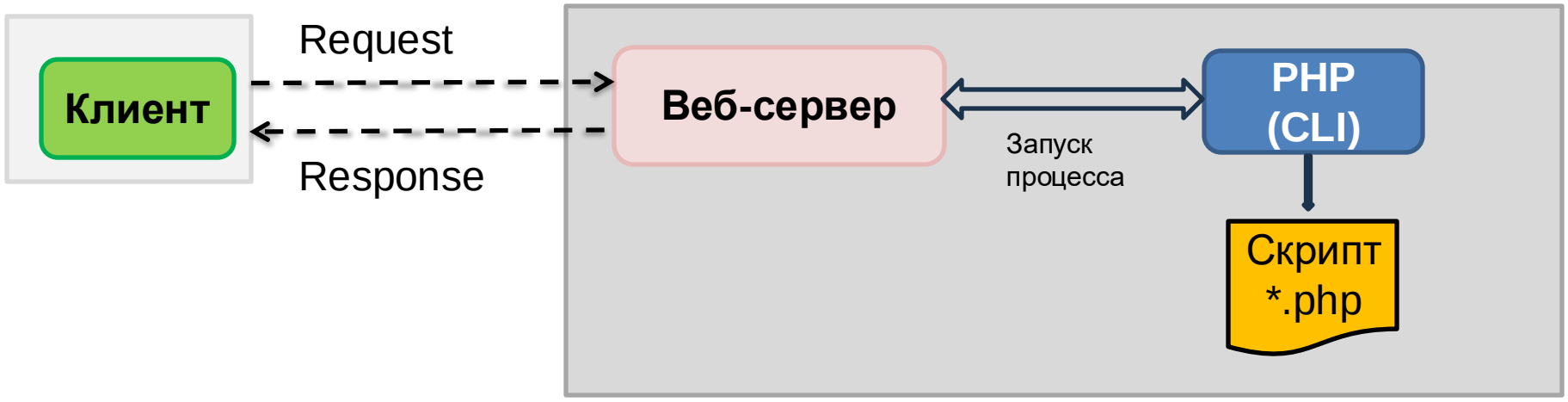
```
<?php if ($item > 10): ?>
Значение больше десяти.
<?php elseif ($item < 10): ?>
Значение меньше десяти.
<?php else: ?>
Значение равно десяти.
<?php endif; ?>
```

```
<?php while ($item < 10): ?>
<li><?= $item ?></li>
<?php endwhile; ?>
```

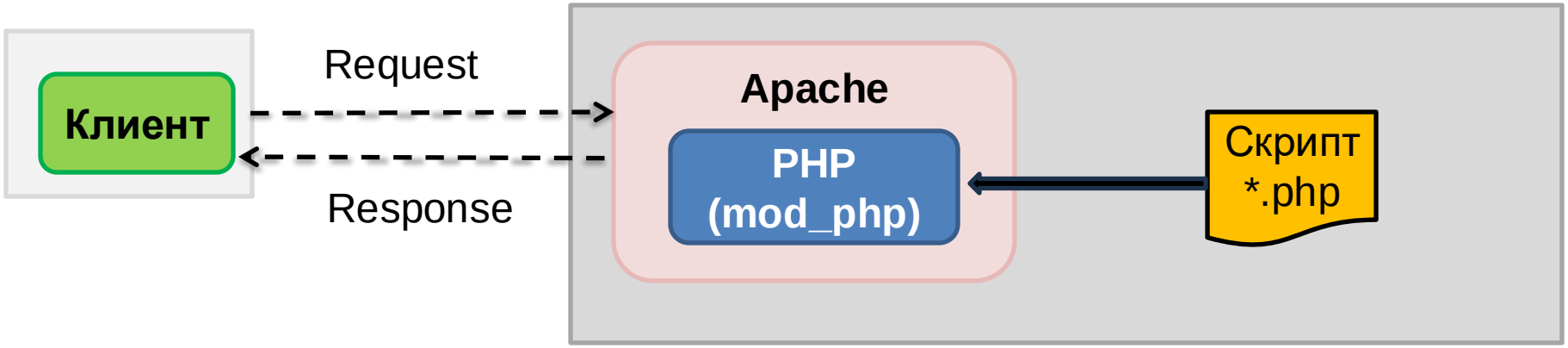
```
<?php for ($item = 1; $item <= 10; $item ++): ?>
<li><?= $item ?></li>
<?php endfor; ?>
```

Запуск PHP как модуля веб-сервера

Интерпретатор PHP - внешний процесс (CGI)



Интерпретатор PHP - модуль веб-сервера



mod_php vs CGI

- **PHP 3** (1998 год) стал переписан на C и получил поддержку модулей веб-серверов.
- `mod_php` **для Apache** :
  - Разработан **Зеевом Сураски** и **Анди Гутмансом** .
  - Встраивал PHP-интерпретатор непосредственно в Apache.
  - Появился в версии PHP 3.0 (июнь 1998), но стал стабильным в PHP 3.0.1 (1999).
  - **Преимущества** :
    - Высокая скорость (интерпретатор работал внутри Apache).
    - Снижение накладных расходов на запуск процессов.

- **PHP 4** (2000 год) сделал `mod_php` основным способом интеграции с Apache.
- **Особенности** :
  - Веб-сервер Apache с `mod_php` стал стандартом для хостинга.
  - Поддержка сессий, улучшенная работа с БД (MySQL).
  - Производительность была выше, чем в CGI, благодаря встроенному интерпретатору.

Поддержка HTTP в mod_PHP

Поддержка HTTP в mod_php

Суперглобальные массивы $_SERVER, $_GET, $_POST, … в CGI и mod_PHP выглядят одинаково, но формируются по-разному.

Как работает:

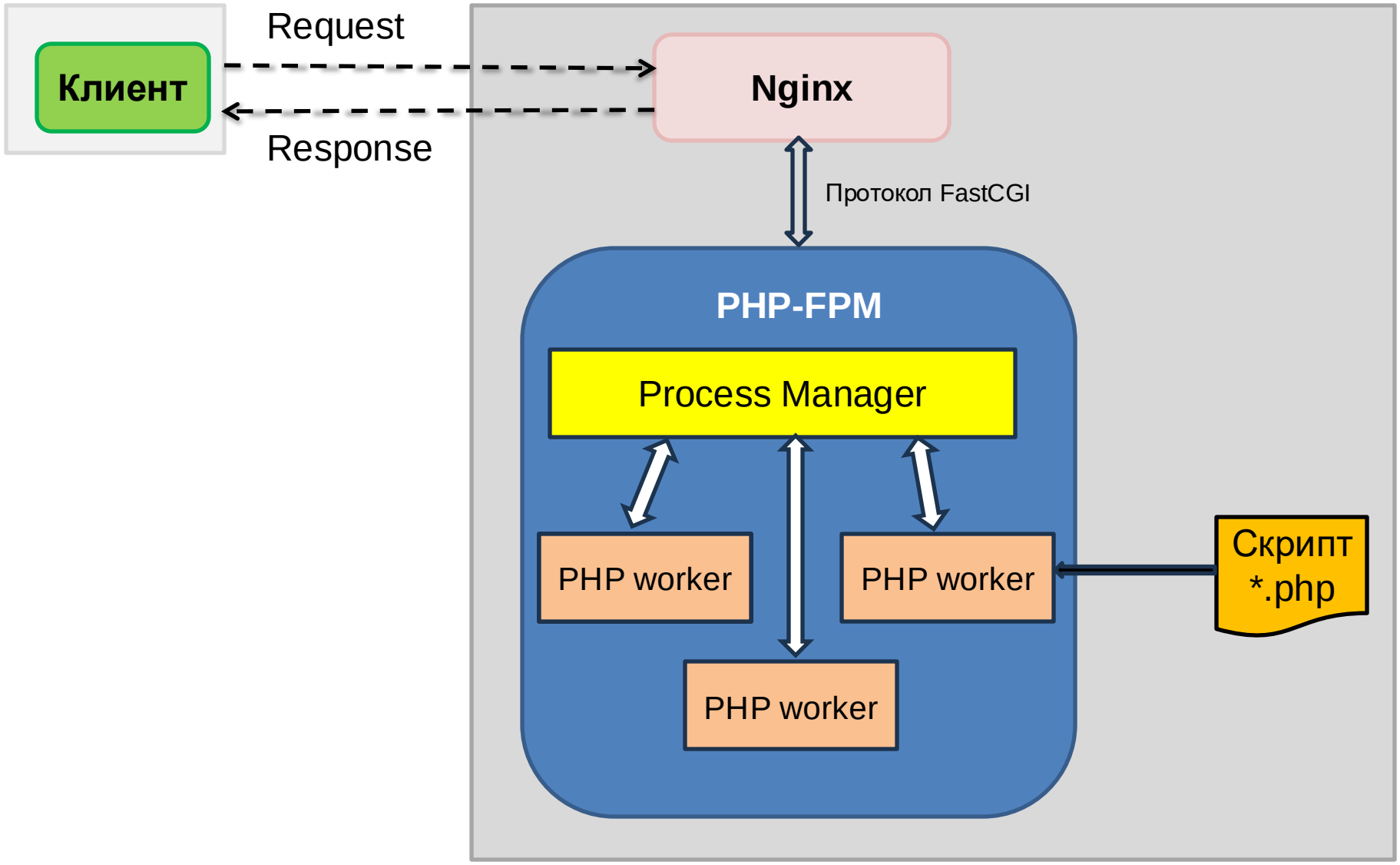
- **PHP встраивается в Apache** :
  - Интерпретатор PHP запускается внутри процесса Apache.
  - Данные запроса передаются через **Apache API** , а не через переменные окружения.
- **Формирование массивов** :
  - `$_SERVER` : Заполняется на основе данных Apache (например, `request_rec` структуры).
  - `$_GET` / `$_POST` : Парсятся из URL и тела запроса, используя Apache-модули (например, `mod_php` напрямую читает данные из буфера Apache).

Особенности:

- **Интеграция с Apache** :
  - Нет настоящих переменных окружения CGI (например, `QUERY_STRING` , `REQUEST_METHOD`).
  - PHP эмулирует CGI-переменные через Apache API.
- **Производительность** :
  - Нет накладных расходов на создание процессов.
  - Данные доступны мгновенно.

Запуск PHP через PHP-FPM (FastCGI)

Интерпретатор PHP управляется PHP-FPM (FastCGI)



PHP-FPM и FastCGI

- **Как работает** :
  - Процессы интерпретатора (например, PHP-FPM) запускаются **один раз** и остаются в памяти.
  - Веб-сервер (Apache, Nginx) передаёт запросы через **сокеты или TCP-порты** .
- **Преимущества** :
  - **Производительность** : Нет накладных расходов на создание процессов.
  - **Масштабируемость** : Можно настроить пул процессов для обработки множества запросов.
  - **Изоляция** : Ошибки в скриптах не влияют на работу веб-сервера.

FastCGI vs CGI vs mod_php

PHP-FPM :

- Получает параметры и тело через FastCGI.
- Заполняет суперглобальные массивы `$_SERVER` , `$_POST` , `$_GET` .
- Тело запроса доступно через `php://input` .

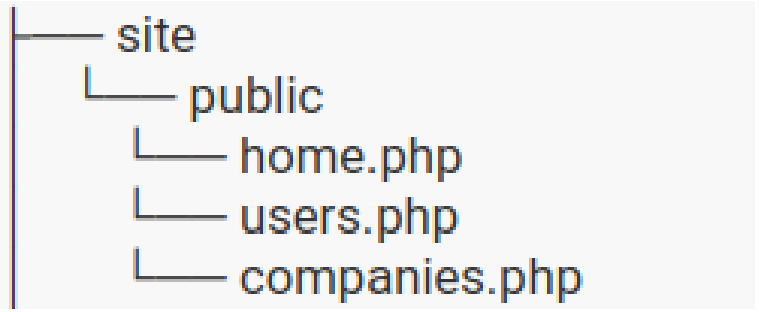
- **FastCGI** лучше **CGI** за счёт переиспользования процессов и масштабируемости.
- **FastCGI (PHP-FPM)** лучше **mod_php** , так как:
  - Экономит память.
  - Позволяет изолировать процессы и настраивать их под конкретные задачи.
  - Совместим с современными веб-серверами (Nginx).
  - Поддерживает асинхронную обработку запросов.

Современный стек : Nginx + PHP-FPM (FastCGI) стал стандартом для высоконагруженных проектов.

Два подхода к архитектуре веб-приложения

- **Page Controller**

example.com/home.php
example.com/users.php?action=new
example.com/companies.php?id=2



Запросы привязаны к конкретным файлам, каждая страница обрабатывается отдельным скриптом

- **Front Controller**

example.com/home
example.com/users/new
example.com/companies/2



Все запросы обрабатываются единой точкой входа (например, index.php)

Page Controller vs Front Controller

Page Controller

- Каждый URL соответствует конкретному файлу (например, /about.php).
- Общая логика (хедер, футер, подключение к БД) дублируется через require/include.
- Простота : Подходит для небольших сайтов с минимумом логики.

Front Controller

- Все запросы проходят через index.php.
- Маршруты определяются в одном месте (например, routes.php).
- Легко добавлять общую логику (аутентификация, логирование) в единую точку входа.

Настройка маршрутизации в веб-серверах

1. Apache

- Маршрутизация через `.htaccess` и `mod_rewrite` :
  - Используется для перенаправления всех запросов на центральный скрипт (например, `index.php`).
  - Пример конфигурации для фронт-контроллера (Laravel, Symfony):

apache Копир

```apache
# Включаем mod_rewrite
RewriteEngine On

# Если запрашиваемый файл/директория не существует
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

# Перенаправляем всё на index.php
RewriteRule ^(.*)$ index.php [QSA,L]
```

  - Особенности :
    - Гибкость (можно настраивать маршруты через регулярные выражения).
    - Поддержка `.htaccess` для локальных настроек (удобно в shared hosting).

Настройка маршрутизации в веб-серверах

2. Nginx

- Маршрутизация через `try_files` :
  - Перенаправляет запросы на `index.php` , если файл не существует.
  - Пример конфигурации:

```nginx
server {
    listen 80;
    server_name example.com;
    root /var/www/html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    # Обработка PHP через PHP-FPM
    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass unix:/var/run/php/php8.1-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }
}
```

  - Особенности :
    - Высокая скорость (асинхронная обработка).
    - Нет поддержки `.htaccess` , все настройки в конфигах Nginx.

Настройка маршрутизации в веб-серверах

4. LiteSpeed

- Аналогичен Apache :
  - Поддерживает `.htaccess` и `mod_rewrite`.
  - Используется в shared hosting для работы с PHP.
  - Пример конфигурации:

```apache
1 RewriteRule ^(.*)$ index.php [L,QSA]
```

Настройка маршрутизации в веб-серверах

5. Caddy

- **Автоматическая маршрутизация** :
  - Простой конфиг для фронт-контроллера:

caddyfile

```
example.com {
    root * /var/www/html
    php_fastcgi unix//var/run/php/php8.1-fpm.sock
    file_server
    try_files {path} {path}/ /index.php?{query}
}
```

  - **Особенности** :
    - Автоматическое HTTPS.
    - Минималистичный синтаксис конфигурации.

Локальный сервер, встроенный в PHP

Возможности встроенного веб-сервера

1. **Обработка статических файлов** :
   - Отдаёт HTML, CSS, JS, изображения и другие файлы.
2. **Выполнение PHP-скриптов** :
   - Автоматически обрабатывает `.php` файлы.
3. **Маршрутизация через роутер-скрипт** :

```bash
1 php -S localhost:8000 router.php
```

   - Если `router.php` существует, все запросы передаются в него.

Возможности встроенного веб-сервера

- Пример роутера:

```php
<?php
$path = $_SERVER['REQUEST_URI'];
if ($path === '/about') {
    require 'about.php';
} else {
    require 'index.php';
}
?>
```

4. **Поддержка HTTP-методов** (GET, POST, PUT, DELETE и др.).
5. **Логирование** :
    - Выводит логи в терминал (запросы, ошибки).

Преимущества встроенного веб-сервера

- **Простота** :
  - Нет необходимости настраивать Apache/Nginx.
  - Запуск одной командой.
- **Удобство разработки** :
  - Автоматически перечитывает PHP-файлы (без перезагрузки сервера).
  - Поддержка роутинга для фреймворков (Laravel, Symfony).
- **Отладка** :
  - Вывод ошибок в реальном времени (если включено в `php.ini`).

Только для разработки и тестирования!

Стандартный PHP как инструмент веб-разработки

Что есть в стандартном PHP

1. **HTTP-протокол** :
   - Суперглобальные массивы `$_GET` , `$_POST` , `$_SERVER` , `$_FILES` .
   - Функции вроде `header()` , `setcookie()` .
2. **Шаблонизация** :
   - Встроенный синтаксис `<?php ?>` и `<?= ?>` для смешивания HTML и PHP.
3. **Веб-сервер** :
   - `php -S localhost:8000` для быстрого запуска.
4. **Базовые функции** :
   - Работа с БД (`mysqli` , `PDO`).
   - Обработка форм, сессии, JSON.

Стандартный PHP идеален для:

- Простых скриптов.
- Быстрого прототипирования.
- Маленьких проектов.

Запуск PHP-скриптов из браузера

С перезагрузкой страницы (с новым или тем же URL)

- Адресная строка браузера
- Тег <a> в HTML-странице
- HTML-формы

Без перезагрузки страницы (URL остается прежним)

- Асинхронный метод fetch() языка JavaScript