



Основы HTTP

Лектор: Матвеева Светлана

- 1) Отмечаемся на портале
- 2) Проходим квиз по материалам прошлой лекции
презентации (ссылочка будет сейчас в чате)



Темы на сегодня

- Интернет и WWW
- Документы и их различия
- URL
- Клиент-серверная архитектура
- HTTP-протокол
- Трёхзвенная архитектура
- Веб-серверы
- Сервер приложения

Интернет и WWW



Интернет и WWW

Интернет — система объединённых компьютерных сетей для хранения и передачи информации.

World Wide Web — распределенная система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключённых к сети Интернет.

Интернет объединяет компьютеры в сеть

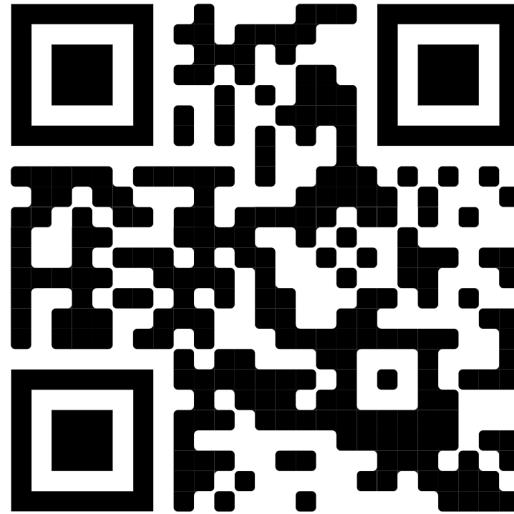
World Wide Web объединяют ресурсы (документы)

Интернет и WWW

Интернет состоит из следующих основных компонентов:

- **WWW** (World Wide Web) - система связанных гипертекстовых документов
- **DNS** (Domain Name System) - система, обеспечивающая возможность использовать доменные имена вместо IP-адресов
- **Email** - электронная почта
- **IRC** (Internet Relay Chat) - служба, предназначенная для поддержки текстового общения в реальном времени (чаты)
- **Телеконференции**, обеспечивающие возможность коллективной коммуникации
- **FTP** (File Transfer Protocol) - служба, обеспечивающая хранение и пересылку файлов различных типов
- **Telnet** (Teletype Network) - служба, предназначенная для управления удалёнными компьютерами
- **Потоковое мультимедиа**

Интернет и WWW



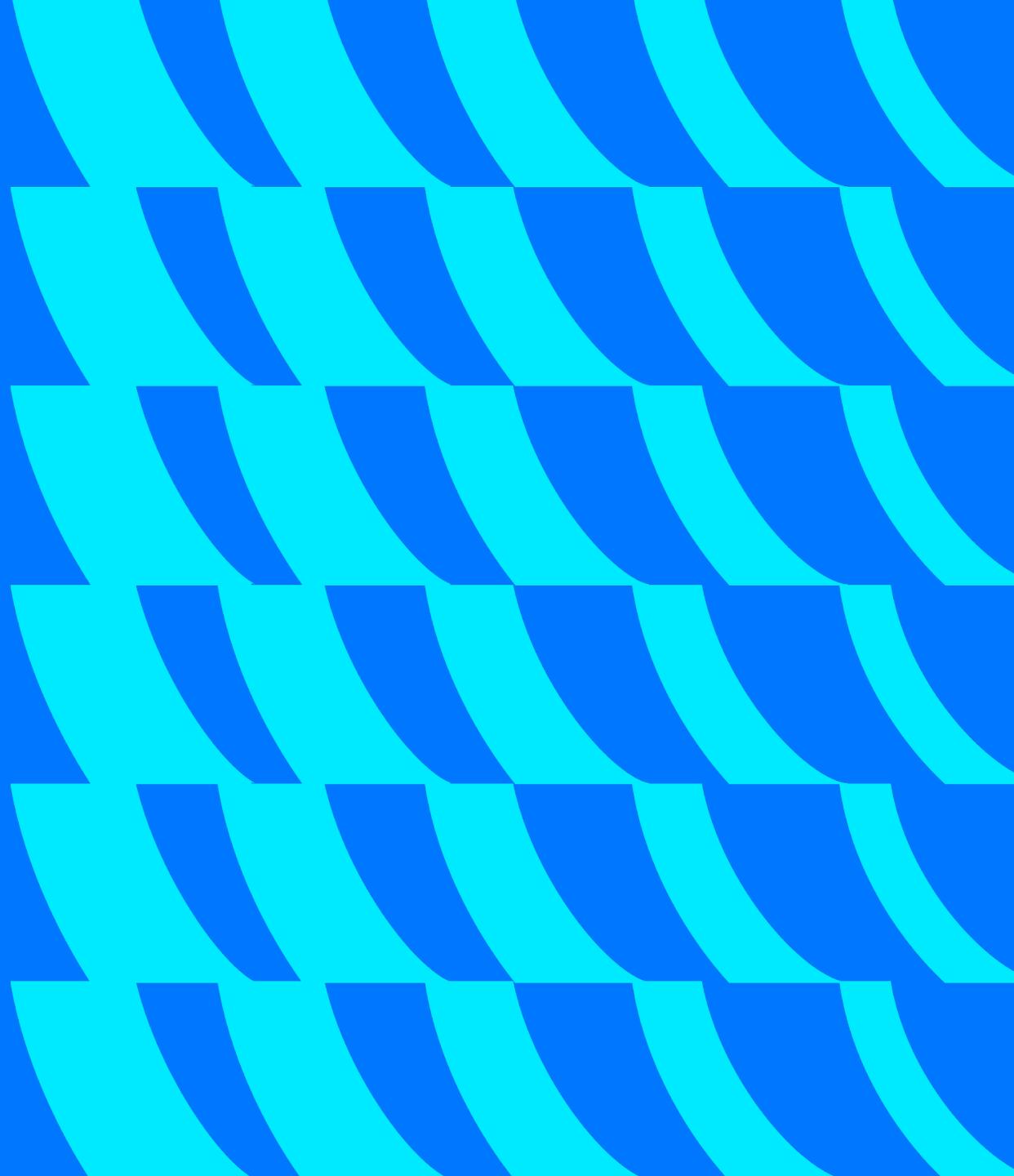
<http://internet-map.net/>

Тут можно посмотреть, насколько огромен WWW.

Чем больше у ресурса посещаемость, тем больше его круг на карте.

Чем ближе круги, тем теснее связаны ресурсы.

Документы



Документы

Web-документ/ресурс – это гипертекстовый документ, содержащий в себе ссылки на другие веб-документы.

Большая часть информации в WWW – это такие **web-документы** (в лекции будем называть их просто документами или ресурсами).

Документы

- **Статические**

Как правило, такие документы – это готовые файлы, лежащие на дисках сервера. Зачастую обладают постоянным адресом.

Примеры: сайт-визитка, PDF-ка с меню ресторана.

- **Динамические**

Изменяющиеся документы, имеющие разный контент для разных пользователей, или, например, для разного времени посещения. Генерируются заново на каждый запрос. Адрес может как быть постоянным, так и меняться.

Примеры: личный кабинет интернет-магазина.

Документы

Mimetype(Internet Media Type) - тип передаваемых данных.

Mimetype = базовый тип данных файла + / + расширение файла

Базовые типы: application, audio, example, image, message, model, multipart, text, video.

Например:

- text/html
- text/css
- image/png
- video/mp4
- application/json

URL



URL

URL (Uniform Resource Locator) - это адрес веб-ресурса.

Форма записи URL

<схема>:[//[<логин>[:<пароль>]@]<хост>[:<порт>]][/<URL-путь>][?<параметры>][#<якорь>]

Пример №1: https://ru.wikipedia.org/wiki/URL#Структура_URL

- *https* - протокол/схема
- *ru.wikipedia.org* - хост
- */wiki/URL* - URL-путь
- *Структура_URL* - якорь

URL

Форма записи URL

<схема>://[<логин>[:<пароль>]@]<хост>[:<порт>]][/<URL-путь>][?<параметры>][#<якорь>]

Пример №2: <https://education.vk.company/schedule/?course=fullstack&show=all>

- *https* - протокол/схема
- *education.vk.company* - хост
- *schedule* - URL-путь
- *course* и *show* - параметры

URL

Форма записи URL

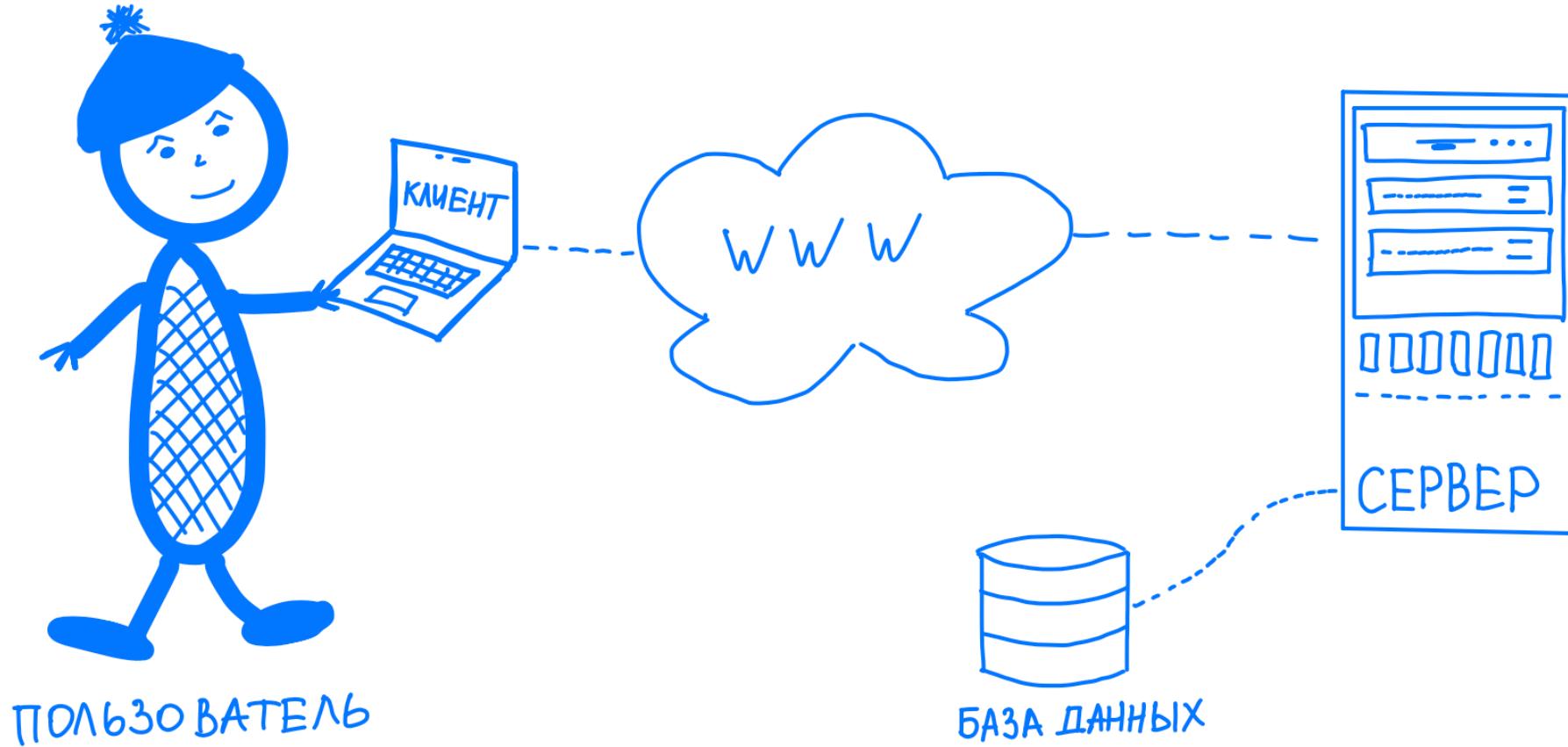
<схема>://[<логин>[:<пароль>]@]<хост>[:<порт>]][/<URL-путь>][?<параметры>][#<якорь>]

Относительный URL: как правило при разработке проекта нет смысла постоянно указывать схему и хост, и поэтому обычно URL-ом называется вся часть адреса на форме записи после хоста и порта.

Клиент-серверная архитектура



Клиент-серверная архитектура



Клиент-серверная архитектура

Веб-клиенты работают на компьютерах конечных пользователей. Задача веб-клиента - получать и отображать документы.

Веб-сервера как правило работают на серверах данных центров. Задача веб-сервера - сохранять, изменять и удалять данные, генерировать и отдавать документы на основе этих данных.

Клиент-серверная архитектура

Клиентами могут быть:

- Браузеры
- Библиотеки разных ЯП, которые могут открывать URL-адреса (в основном HTTP), например, `urllib`
- Поисковики, вредоносные скрипты
- Консольные утилиты, например `curl`

Клиент-серверная архитектура

Браузер - это программа с графическим интерфейсом, которая позволяет открывать html-документы и взаимодействовать с ними.

В современных браузерах есть очень много удобных функций для web-разработчиков.

Клиент-серверная архитектура

Как работает классическое веб приложение сегодня?

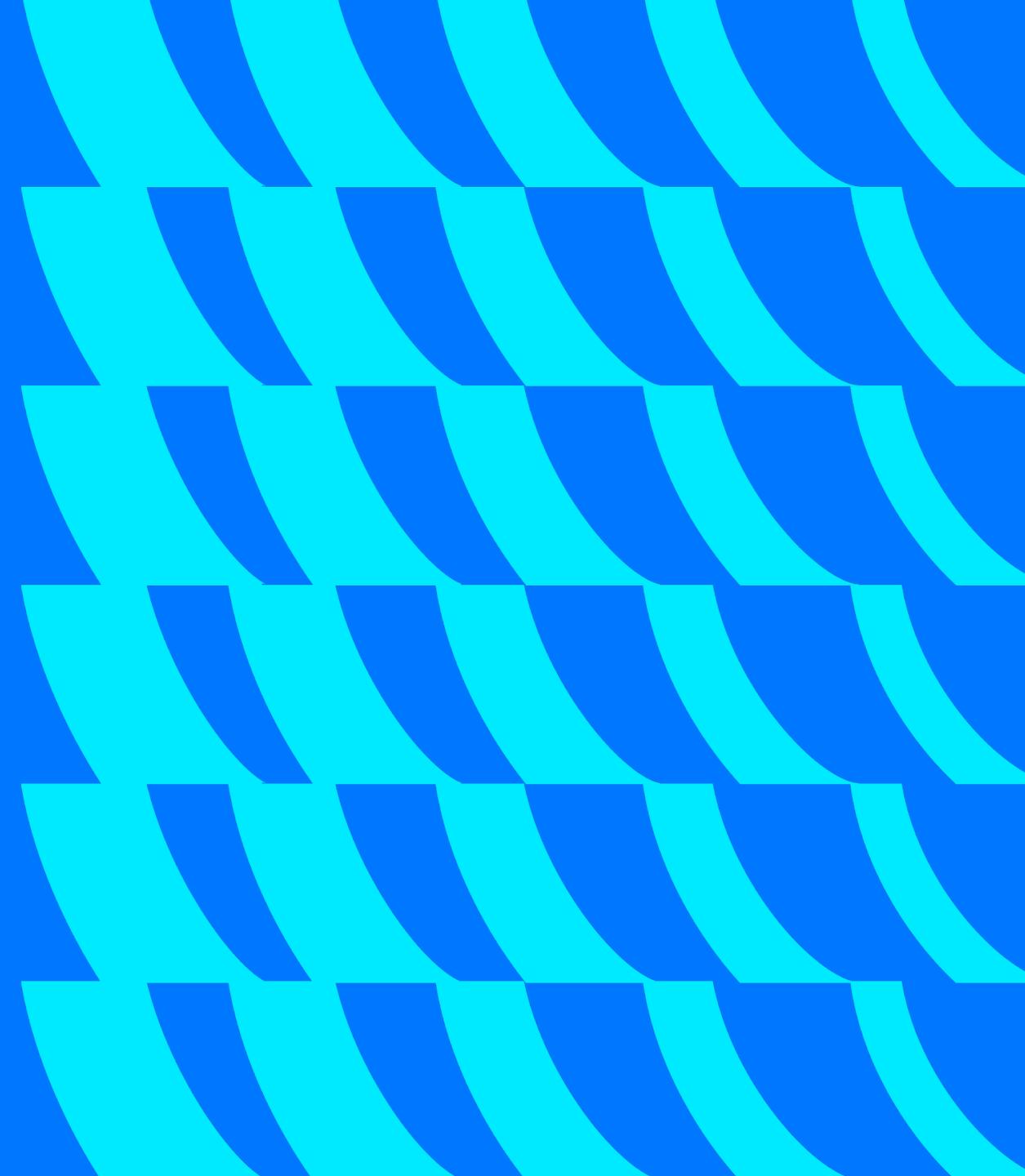
1. Пользователь вводит URL
2. Браузер загружает по этому URL некий HTML-документ
3. Браузер парсит этот документ, и дозагружает дополнительные ресурсы (стили CSS, JS-скрипты, json-файлики для контента)
4. Браузер рендерит готовую HTML страницу с контентом и UI
5. Пользователь взаимодействует элементами страницы, которые в свою очередь могут с помощью JS вызывать какие-либо действия на сервере (отправку/обновление/удаление данных)

Клиент-серверная архитектура

В чем особенность современных веб-приложений?

- Используется Single Page Application (SPA): используется единый html-шаблон для рендераинга всех страниц, информация для конкретной страницы дозагружается с помощью json-файлов
- UI полностью работает на стороне клиента, файлы для формирования UI (CSS, JS) загружаются отдельно от html шаблона
- Сервер работает с чистыми данными: данные не находятся внутри html, а приходят отдельно в json-файлах

HTTP-протокол

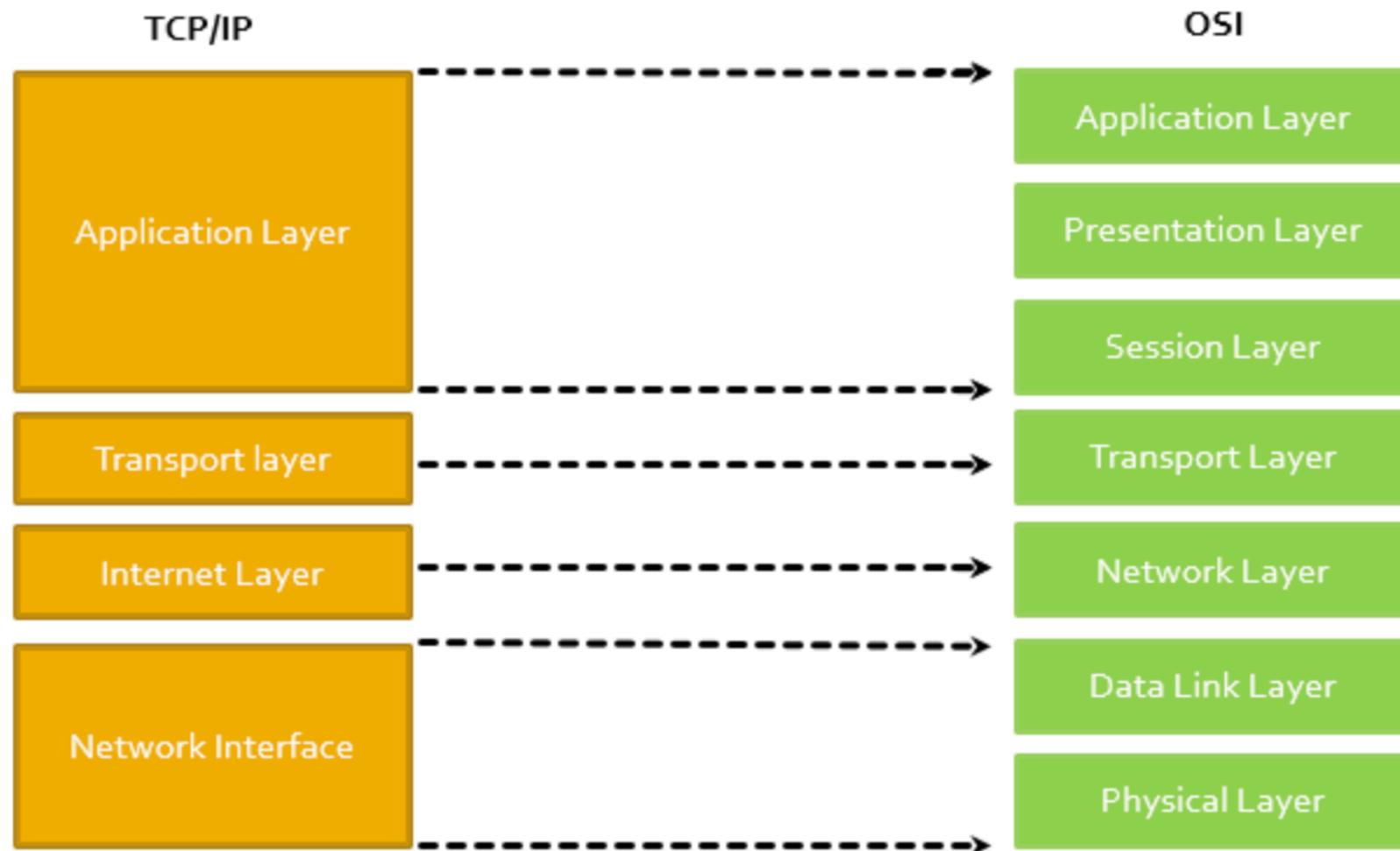


HTTP-протокол

Модель OSI

Данные	Прикладной доступ к сетевым службам
Данные	Представления представление и кодирование данных
Данные	Сеансовый Управление сеансом связи
Блоки	Транспортный безопасное и надёжное соединение точка-точка
Пакеты	Сетевой Определение пути и IP (логическая адресация)
Кадры	Канальный MAC и LLC (Физическая адресация)
Биты	Физический кабель, сигналы, бинарная передача данных

HTTP-протокол



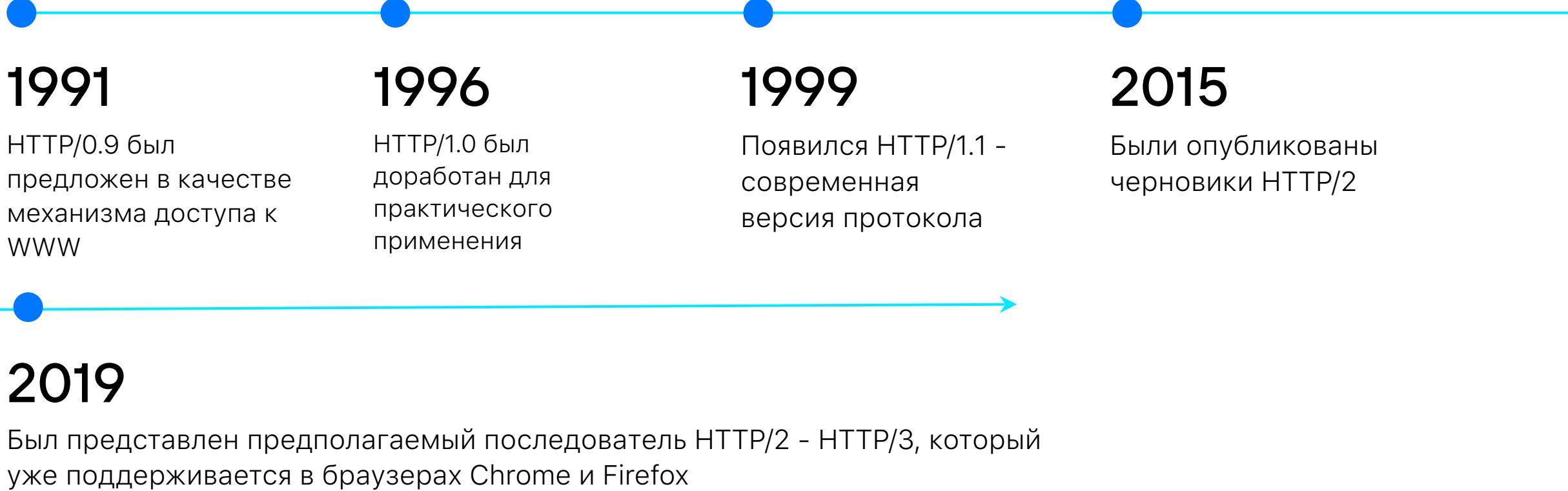
HTTP-протокол

HTTP (HyperText Transfer Protocol, протокол передачи гипертекста) — протокол прикладного уровня передачи данных, работающий на верхнем уровне модели TCP/IP, и осуществляющий передачу гипертекстовых страниц по сети.

Кроме HTTP, в рамках модели TCP/IP работают протоколы

- FTP (передача файлов)
- POP3 (почтовые соединения)
- SMTP (обмен почтой)
- TELNET (удаленный доступ)

HTTP-протокол



HTTP-протокол

Какие задачи решает HTTP-протокол?

- Передача документов
- Передача мета-информации
- Авторизация и аутентификация
- Поддержка сессий
- Кэширование документов
- Управление соединением

HTTP-протокол

Основные особенности HTTP протокола

- Работает поверх TCP/IP
- Работает по принципу «запрос-ответ»
- Не поддерживает постоянное соединение (stateless)

HTTP-протокол

HTTP запрос состоит из

- Стартовая строка - определяет тип запроса
- Заголовки - характеризуют тип данных, содержат разные аутентификационные данные (токены, идентификаторы сессий) и другое
- Тело запроса - содержит передаваемые данные

HTTP-протокол

Методы HTTP-запроса

- GET - получение данных
- HEAD - получение только заголовков документа
- POST - отправка данных на сервер
- PUT - обновление данных на сервере
- PATCH - обновление данных на сервере (частичное)
- DELETE - удаление данных
- OPTIONS - определение возможностей и параметров соединения с сервером

HTTP-протокол

Коды ответов на HTTP запросы

- 1 __ - информационные
- 2 __ - успешное выполнение запроса
- 3 __ - перенаправления (редиректы)
- 4 __ - ошибка на стороне клиента
- 5 __ - ошибка на стороне сервера

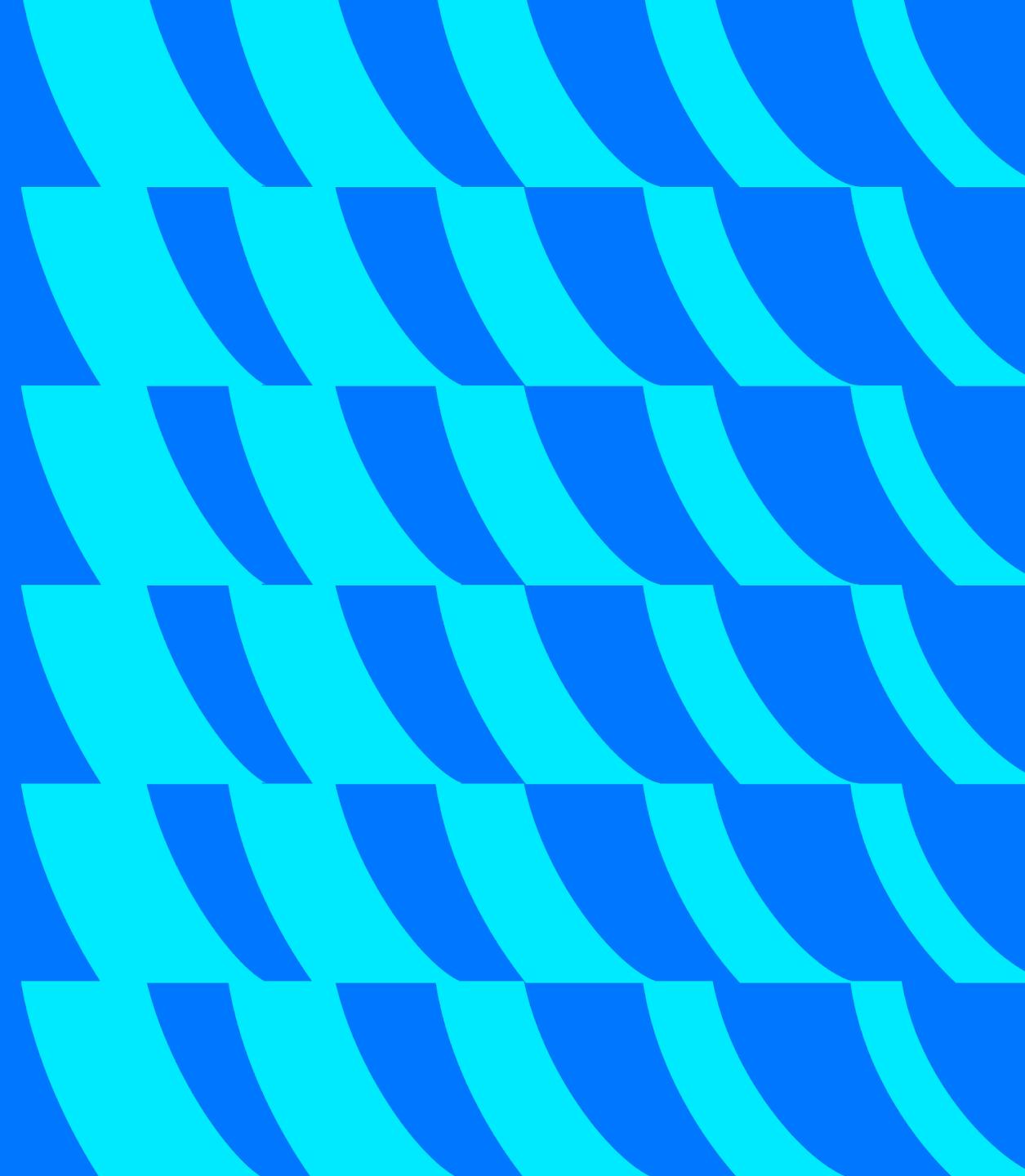
https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP

HTTP-протокол

Заголовки HTTP (самые нужные)

- Content-Type - тот самый mimetype + кодировка
- Content-Length - длина сообщения
- Allow - допустимые методы
- Authorization - данные авторизации
- Cookie - передача данных о сессии на сервер

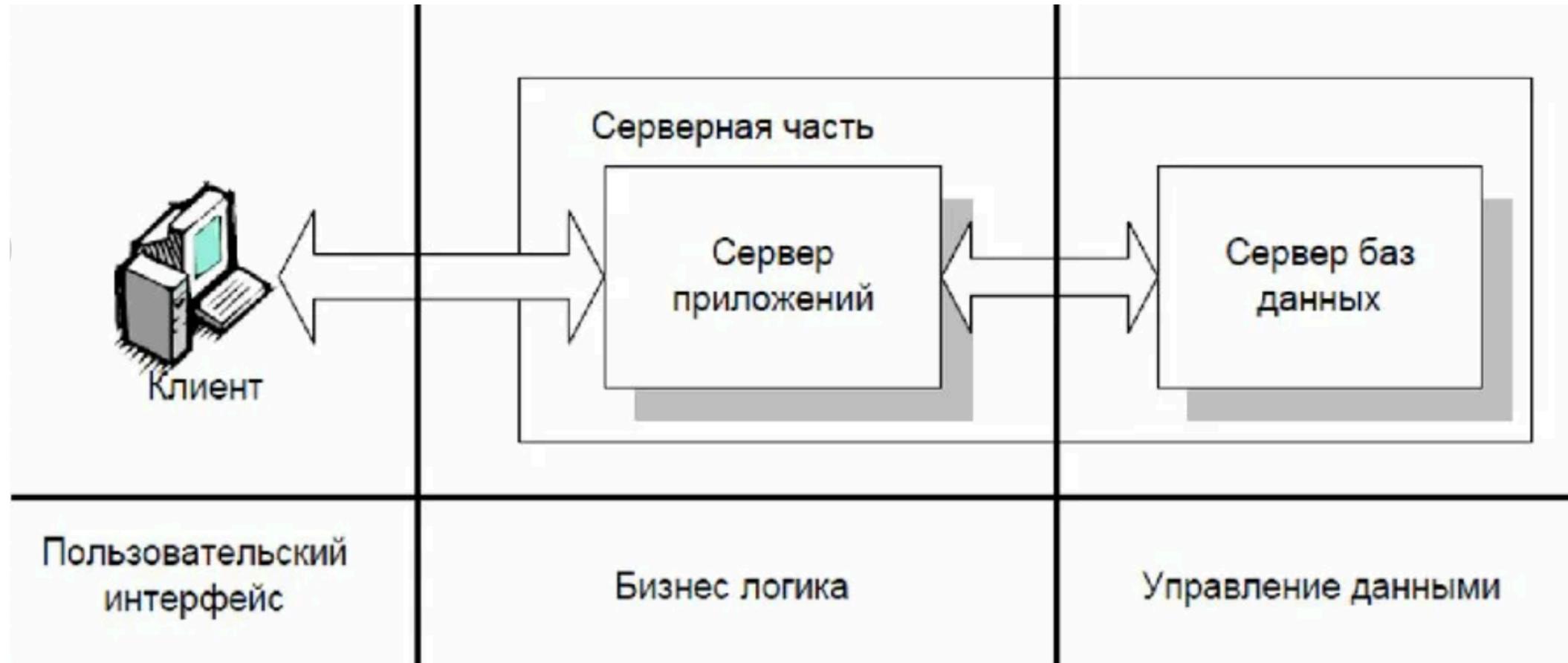
Трехуровневая архитектура



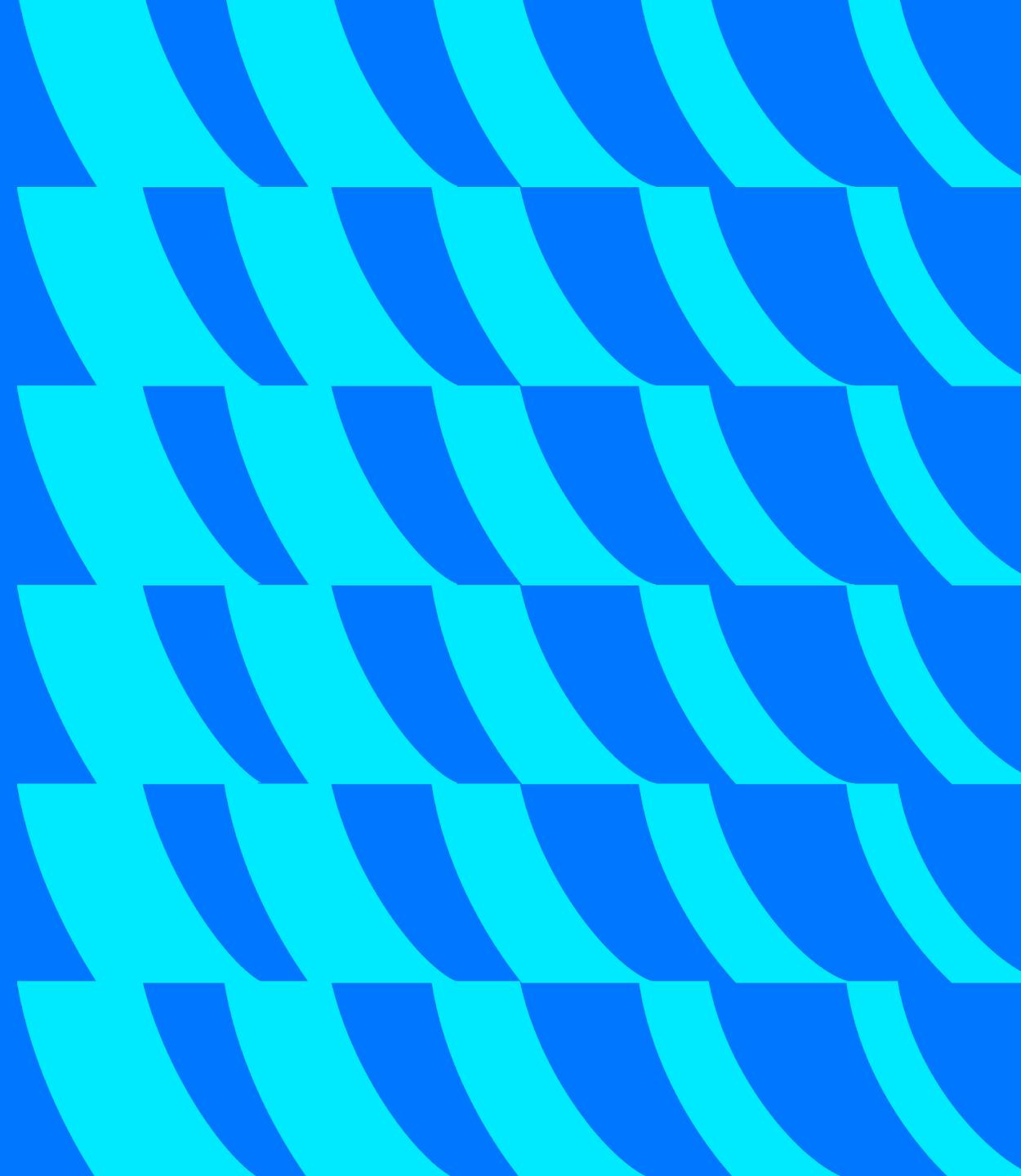
Трёхуровневая архитектура

- Уровень представления (интерфейс пользователя)
- Уровень бизнес-логики (логика работы, обработка запросов, формирование ответов, формирование html-документов, запросы к базе данных)
- Уровень хранения данных (база данных)

Трёхуровневая архитектура



Веб-сервер



Веб-сервер

Веб-сервер — некоторое ПО, развернутое на физическом сервере, принимающее HTTP-запросы от клиентов (браузеров чаще всего), и выдающее им HTTP-ответы (HTML-страницы, изображения, файлы медиа).

Сервер приложения - ПО, также развернутое на физическом сервере, отвечающее за бизнес-логику приложения и генерацию динамических документов.

Веб-сервер обращается с некоторыми запросами к **серверу приложения**.

Веб-сервер

Функции веб-сервера

- Хранение и отдача статических файлов (картинок, текста, html-страниц)
- Общение с сервером приложения (для передачи ему «сложных» запросов, для которых требуется динамическая генерация документа)
- Ведение логов запросов («журналов обращений»)
- Прием запросов
- Отправка ответов
- Поддержка запросов по другим протоколам (mailto, FTP)

Веб-сервер

Какие веб-сервера есть?

- [NGINX](#) — свободный веб-сервер, пользующийся большой популярностью на крупных сайтах
- [Apache](#) — свободный веб-сервер, наиболее часто используемый в UNIX-подобных ОС
- [IIS](#) — сервер от компании Microsoft, распространяемый с ОС семейства Windows
- [lighttpd](#) — свободный веб-сервер, разрабатываемый с расчётом на скорость и защищённость
- [Google Web Server](#) — веб-сервер, разработанный компанией Google
- [Go HTTP](#) — веб-сервер, встроенный в Go



Веб-сервер

NGINX

- Более 20% рынка
- Один из лучших по производительности
- Умеет проксировать запросы
- Хорошо пригоден для работы со статическими файлами
- Относительно(!) прост в настройке
- Поддерживает криптографические протоколы SSL, TLS для шифрования данных
- Прекрасная документация

Веб-сервер

Процессы NGINX

- Master-процесс (запускается в sudo, только один процесс)
 - Читает и валидирует конфиг
 - Открывает сокеты и заводит логи
 - Запускает и управляет рабочими процессами (workers)
- Worker-процесс (запускается без доп. прав, может быть много)
 - Обрабатывает входящий запрос
 - Подчиняется мастер-процессу

Веб-сервер

Установка NGINX

sudo apt install nginx - для Ubuntu

brew install nginx - для MacOS

Файлы NGINX

/etc/nginx/nginx.conf - конфиг для Ubuntu

/usr/local/etc/nginx/nginx.conf - конфиг для MacOS

/var/log/nginx/error.log - логи ошибок (неуспешных запросов)

/var/log/nginx/access.log - логи успешных запросов

/usr/local/nginx/logs/nginx.pid - файл с process id мастер-процесса nginx

Веб-сервер

Конфигурация NGINX

Конфиг Nginx состоит из модулей, которые настраиваются директивами, указанными в конфигурационном файле.

Директивы бывают:

- Простыми

worker_processes 2;

- Блочными

http {

server {

}

}

Веб-сервер

Что находится в конфиге NGINX?

- пользователь, от чьего имени будет запускаться Nginx
- расположение pid-файла, файлов с логами
- сколько рабочих процессов будет запущено
- ограничения на количество соединений
- условия сжатия контента

Веб-сервер

Проксируем в nginx

```
proxy_set_header Host      $host;
proxy_set_header X-Real-IP $remote_addr;
location / {
    proxy_pass http://backend;
}
location /partner/ {
    proxy_pass http://www.partner.com;
}
location ~ \.\w\w\w?\w?$ {
    root /www/static;
}
```

Веб-сервер

Отдача статических файлов в nginx

```
location ~* ^.+\.(jpg|jpeg|gif|png)$ {  
    root      /www/images;  
}
```

```
location /sitemap/ {  
    alias /home/www/generated/;  
}
```

```
/2015/10/ae2b5.png → /www/images/2015/10/ae2b5.png  
/sitemap/index.xml → /home/www/generated/index.xml
```

Сервер приложения (Application server)

• • • • • • •

Веб-сервер

Веб-сервер — некоторое ПО, развернутое на физическом сервере, принимающее HTTP-запросы от клиентов (браузеров чаще всего), и выдающее им HTTP-ответы (HTML-страницы, изображения, файлы медиа).

Сервер приложения - ПО, также развернутое на физическом сервере, отвечающее за бизнес-логику приложения и генерацию динамических документов.

На каждый HTTP запрос в Application сервере создается **некий обработчик**, который выполняет функцию, которая должна вызываться по URL-у запроса.

Веб-сервер

Что за обработчик?

Некоторое промежуточное ПО, обеспечивающий взаимодействие веб-сервера и сервера приложения.

Варианты протокола, по которым могут работать такие ПО:

- mod_perl, mod_python, mod_php
- CGI
- FastCGI
- SCGI
- PSGI, **WSGI**, Rack

Веб-сервер

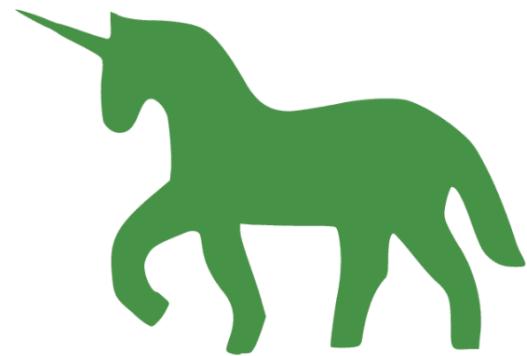
WSGI (Web-Server Gateway Interface)

Это стандарт/протокол взаимодействия между [Python](#)-программой, выполняющейся на стороне сервера, и самим [веб-сервером](#).

Реализован в таких серверах как:

- Bjoern
- uWSGI
- mod_wsgi
- Meinheld
- CherryPy
- Gunicorn - из коробки работает с Django, отлично ладит с Nginx

Веб-сервер



gunicorn

Веб-сервер

Gunicorn

```
$ pip install gunicorn
$ cat myapp.py
def app(environ, start_response):
    data = b"Hello, World!\n"
    start_response("200 OK", [
        ("Content-Type", "text/plain"),
        ("Content-Length", str(len(data)))
    ])
    return iter([data])
$ gunicorn -w 4 myapp:app
[2014-09-10 10:22:28 +0000] [30869] [INFO] Listening at: http://127.0.0.1:8000 (30869)
[2014-09-10 10:22:28 +0000] [30869] [INFO] Using worker: sync
[2014-09-10 10:22:28 +0000] [30874] [INFO] Booting worker with pid: 30874
[2014-09-10 10:22:28 +0000] [30875] [INFO] Booting worker with pid: 30875
[2014-09-10 10:22:28 +0000] [30876] [INFO] Booting worker with pid: 30876
[2014-09-10 10:22:28 +0000] [30877] [INFO] Booting worker with pid: 30877
```

Домашнее задание

- 0) Установить Nginx и Gunicorn
- 1) Создать папку public/ в корне директории с ДЗ, закинуть туда картиночку, настроить nginx для отдачи статических файлов из папки public/, показать, что картиночку получается открыть на <http://localhost:8080/>
- 2) Создать простейшее WSGI приложение (можно взять с руководства по gunicorn) и запустить его с помощью Gunicorn. Настроить проксирование запросов на nginx на запущенный gunicorn.
- 3) Измерить производительность Nginx и Gunicorn с помощью ab или wrk. Добиться отказа системы.

Домашнее задание (полезные ресурсы)

https://nginx.org/ru/docs/beginners_guide.html - руководство по Nginx

<https://gunicorn.org/> - Gunicorn

<https://docs.gunicorn.org/en/stable/> - Gunicorn документация

(Этих ресурсов достаточно для выполнения ДЗ в полном объеме)

Домашнее задание (как сдать)

Можно будет сдать либо лично, либо записать скринкаст (запись экрана), и выложить на него ссылку в гитхаб вместе с ДЗ.

Если выбрали второе, то в репозиторий обязательно:

- конфиг nginx (очищенный от лишних закомментированных строк)
- WSGI приложение
- ссылку на скринкаст, на котором видно:
 - Что localhost:8080 отдает статику
 - Что запросы проксируются на gunicorn
 - Как происходит тестирование производительности (попытаться зайти во время тестирования на ресурс и убедиться что он недоступен)

Спасибо!