



Основы HTTP. Сетевое взаимодействие.

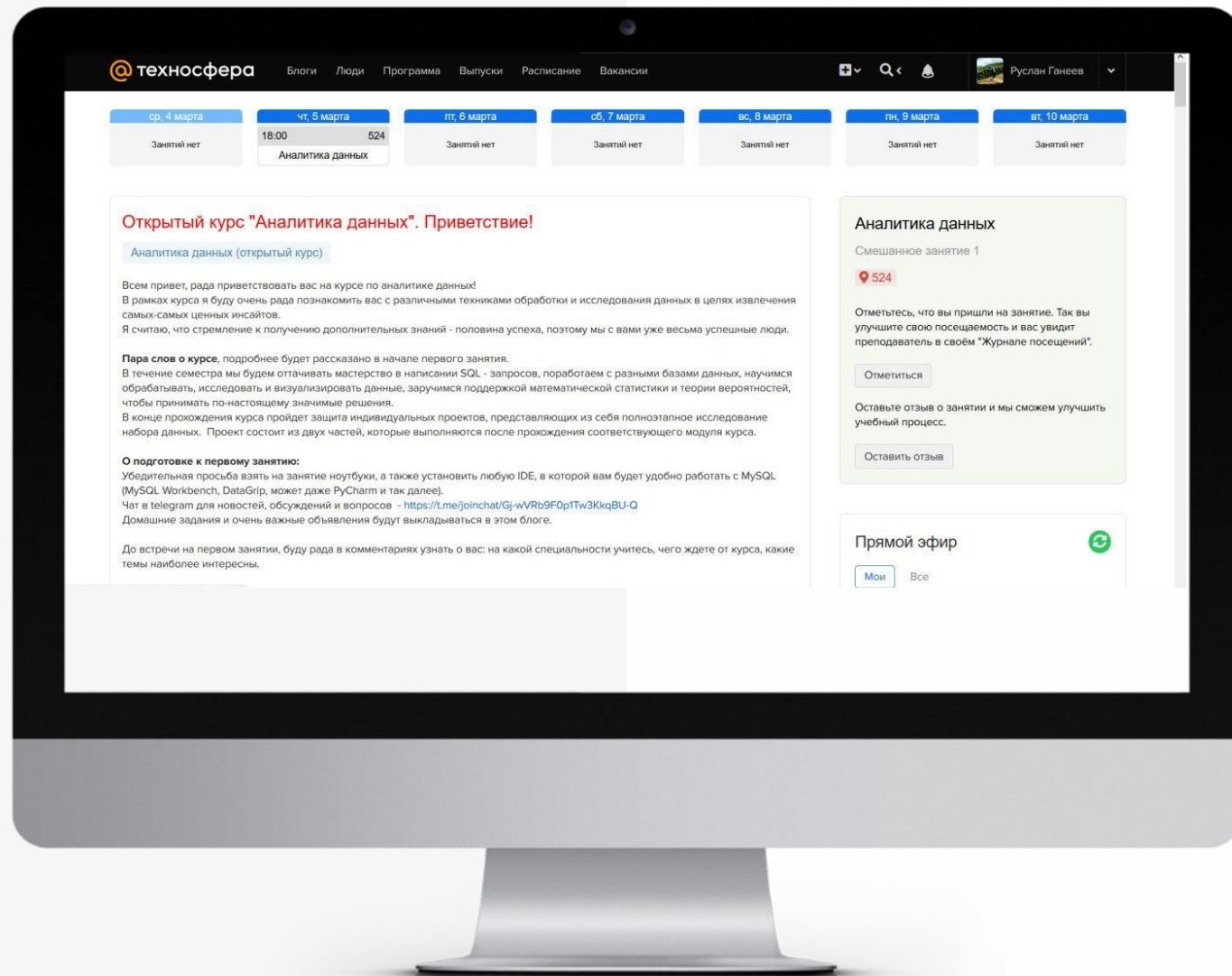
Антон Кухтичев





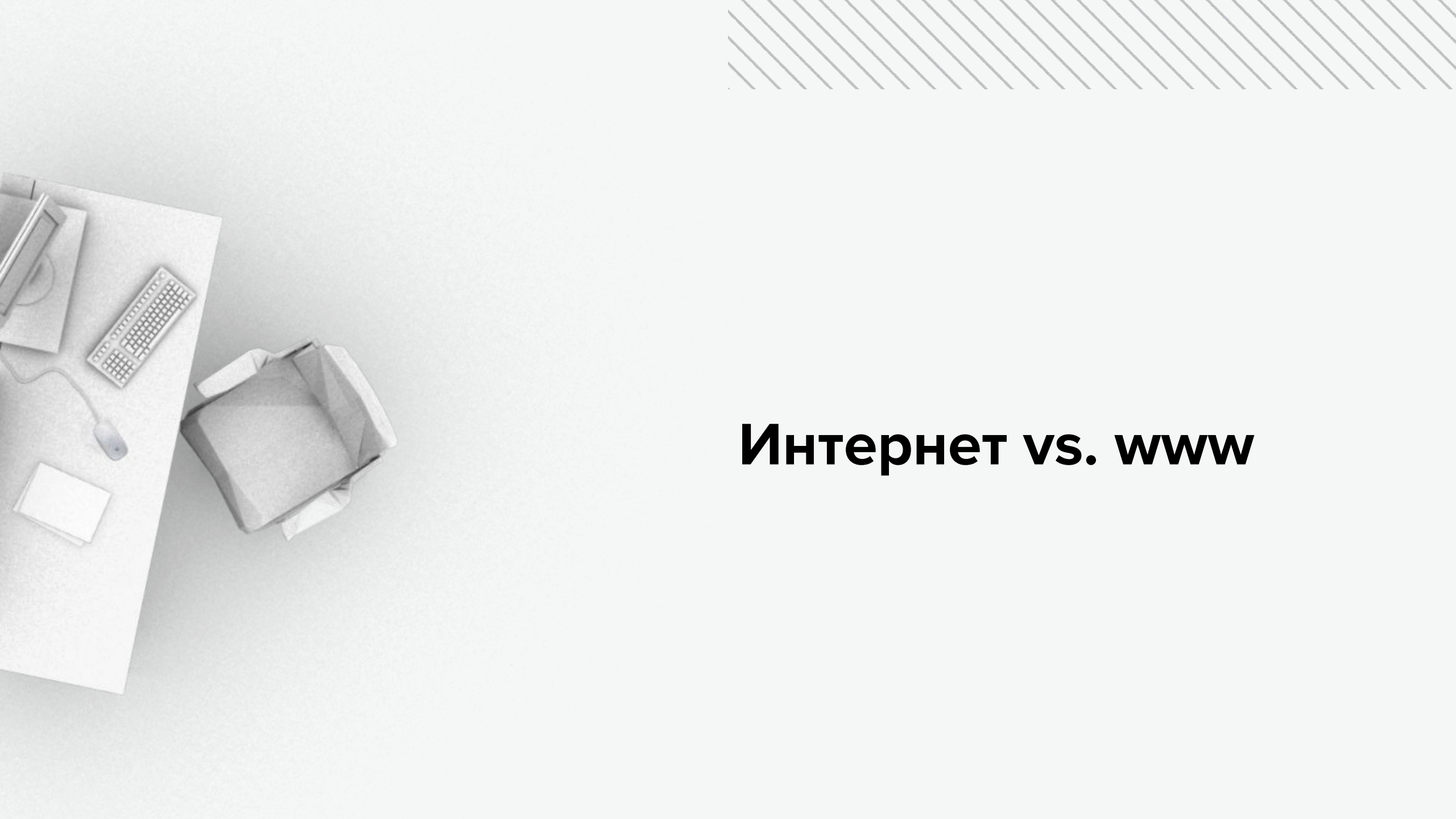
Содержание занятия

1. Интернет vs. www
2. Документы
3. Клиент-серверная архитектура
4. Веб-клиенты
5. HTTP-протокол
6. Трёхзвенная архитектура
7. Веб-сервер
8. Сервер приложений
9. Сокеты в Python'e



Напоминание отметиться на портале

Иначе плохо всё будет.



Интернет vs. www

Интересный факт

«**Правило:** слово *интернет* пишется с маленькой буквы и склоняется по падежам»

— Артемий Лебедев. «Ководство», §55. Как писать слово интернет.





Интернет vs. WWW

Интернет — глобальная сеть передачи данных.

Протоколы:

- **HTTP**, SSH, P2P — прикладные протоколы;
- DNS — система имён;
- TCP — надёжная последовательная передача данных;
- IP — глобальная адресация, передача в гетерогенной среде.

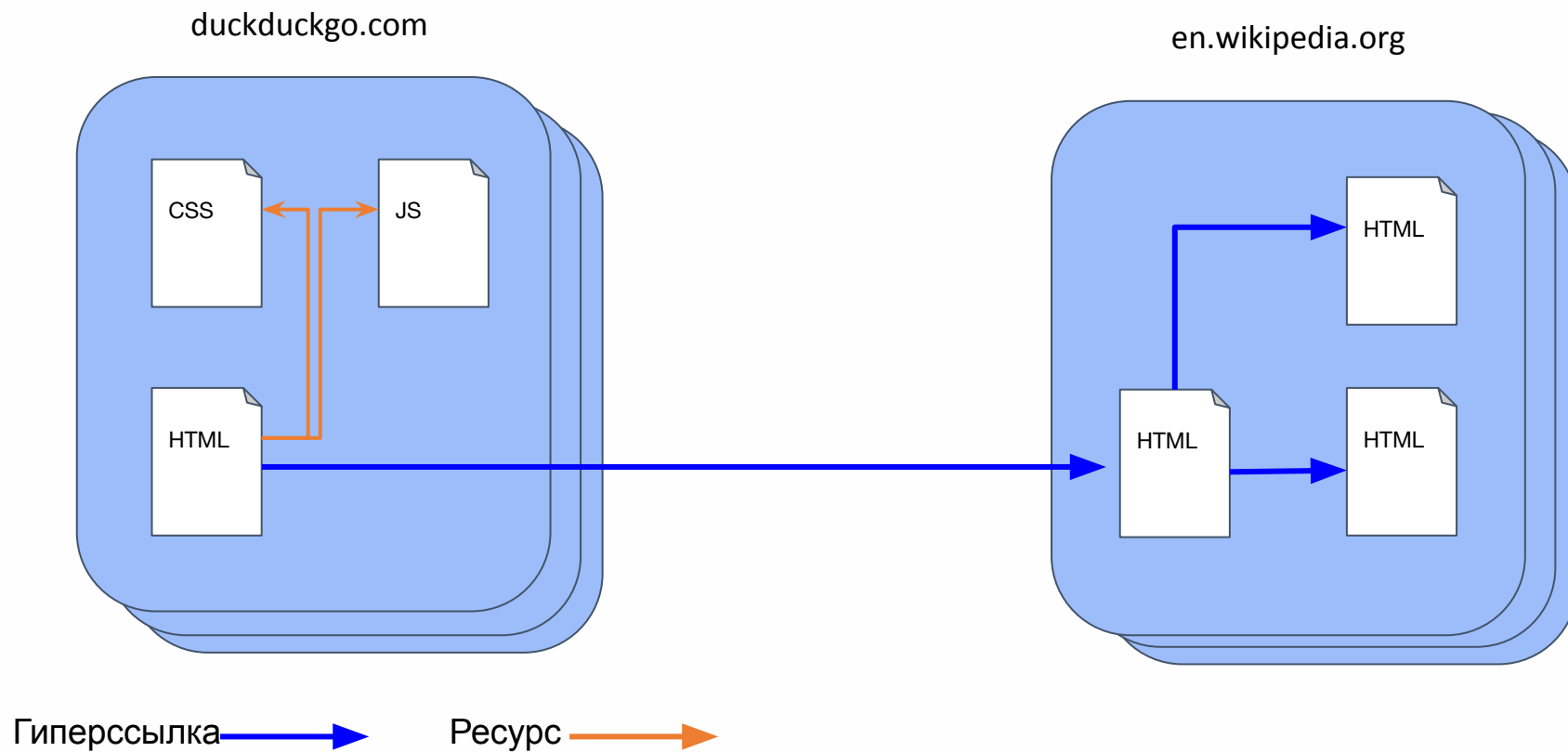


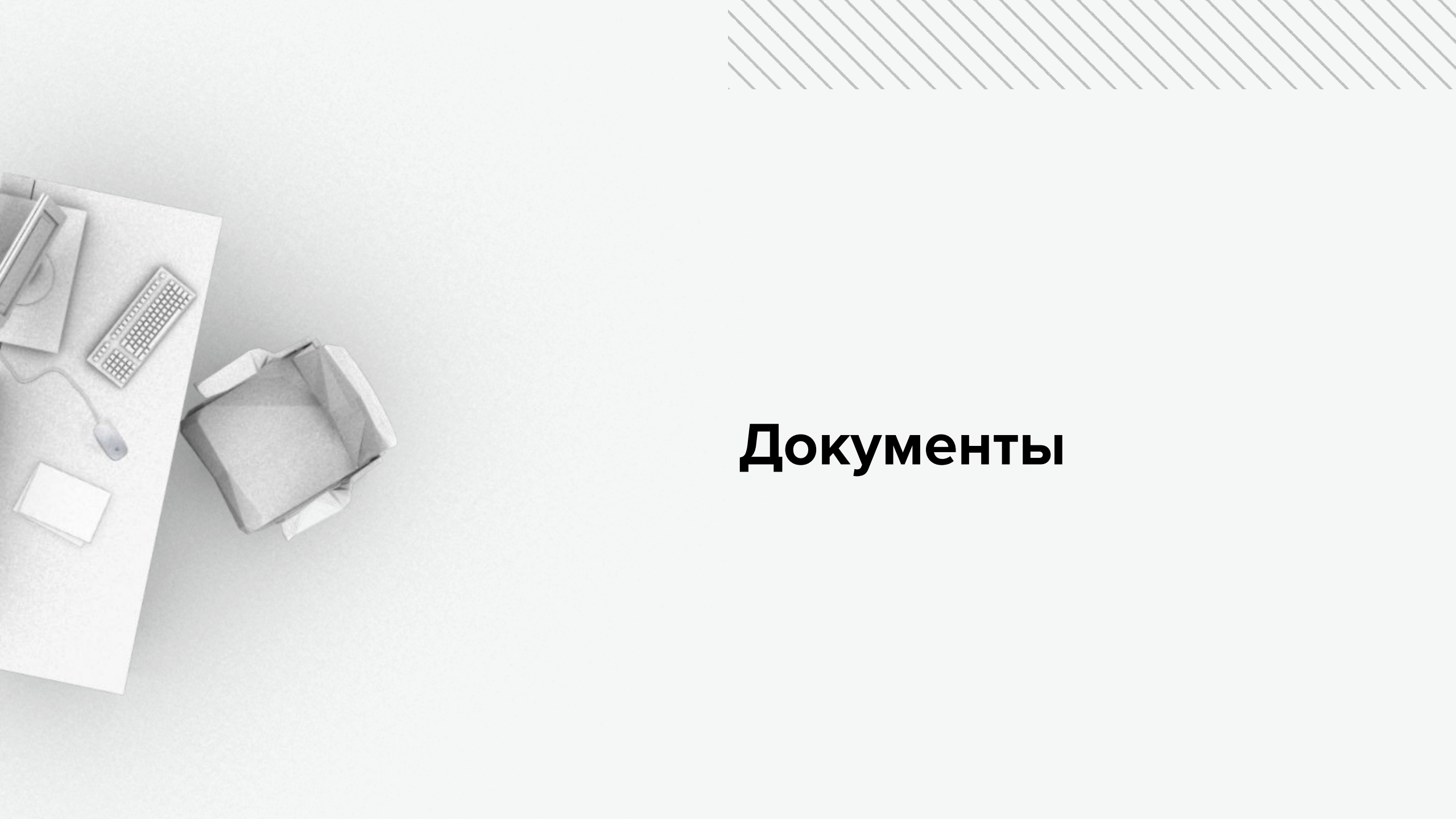
Интернет vs. WWW

WWW — множество взаимосвязанных документов, располагающихся на машинах, подключённых к интернету.

WWW — набор протоколов, серверного и клиентского ПО, позволяющих получать доступ к документам.

Интернет vs. WWW





Документы

Документы

Документы могут быть:

- Статические
 - Это файлы на дисках сервера;
 - Как правило, обладают постоянным адресом.
- Динамические
 - Создаются на каждый запрос;
 - Содержимое зависит от времени и пользователя;
 - Адрес может быть постоянным или меняться.



Типы документы (Mime-типы)

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

Расширения файлов играют второстепенную роль.



URL

URL — uniform resource locator

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

`http://server.org:8080/path/doc.html?a=1&b=2#part1`

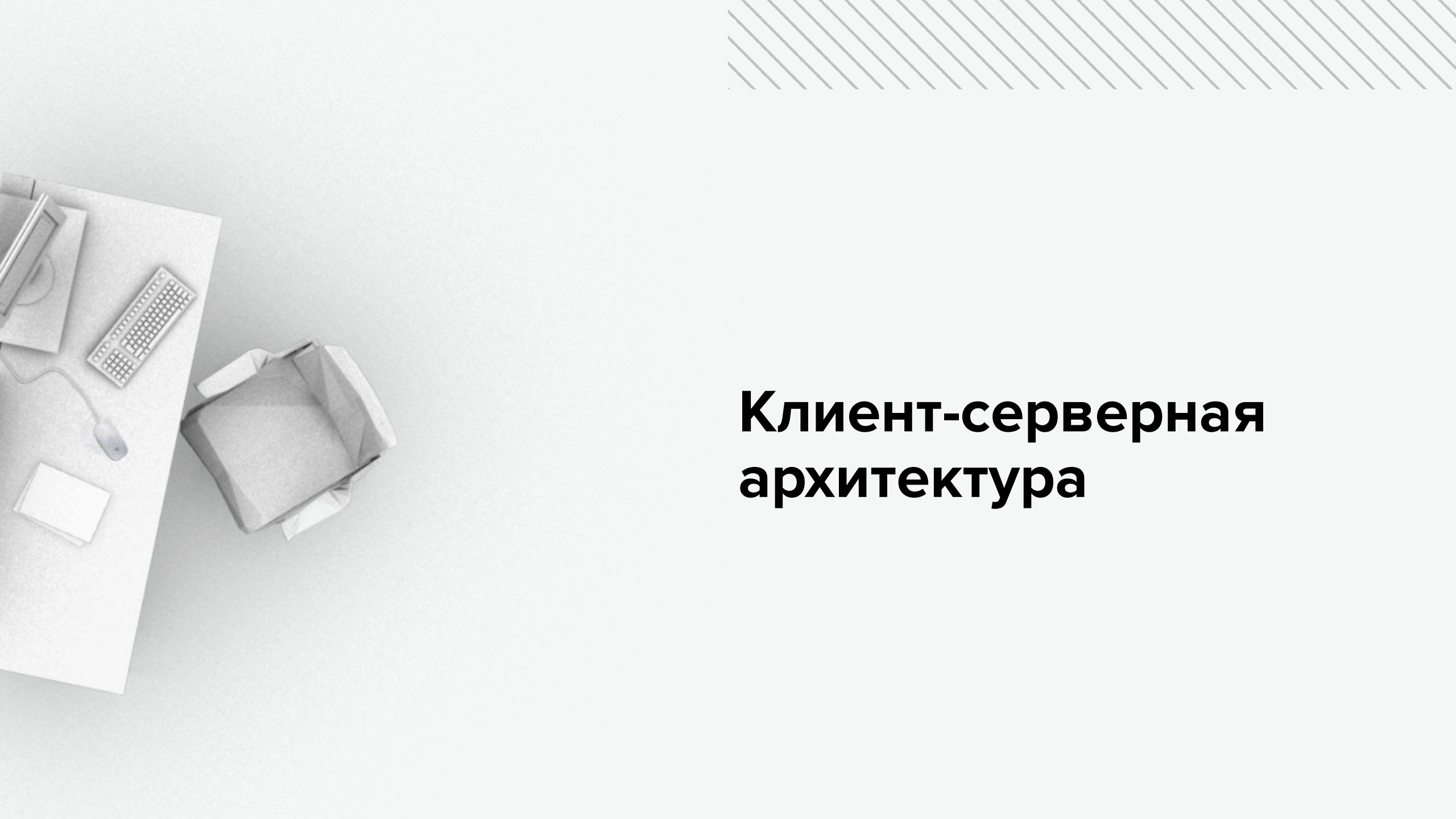
- `http` — протокол;
- `server.org` — DNS имя сервера (может указываться ip-адрес машины);
- `8080` — TCP порт;
- `/path/doc.html` — путь к файлу;
- `a=1&b=2` — параметры запроса;
- `part1` — якорь, положение на странице.

Абсолютные и относительные URL

- `http://server.org/1.html` — абсолютный;
- `//server.org/1.html` — абсолютный (schemeless);
- `/another/page.html?a=1` — относительный (в пределах домена);
- `pictures/cat.png` — относительный (от URL текущего документа);
- `?a=1&b=2` — относительный (от URL текущего документа);
- `#part2` — относительный (в пределах текущего документа);

Правила разрешения URL

- `https://site.com/path/page.html` — основной документ
- `http://wikipedia.org` = `http://wikipedia.org`
- `//cdn.org/jquery.js` = `https://cdn.org/jquery.js`
- `/admin/index.html` = `https://site.com/admin/index.html`
- `another.html` = `https://site.com/path/another.html`
- `?full=1` = `https://site.com/path/page.html?full=1`
- `#chapter2` = `https://site.com/path/page.html#chapter2`



Клиент-серверная архитектура



Клиент-серверная архитектура

Веб-клиенты работают на компьютерах конечных пользователей. Задача веб-клиентов состоит в получении и отображении документов.

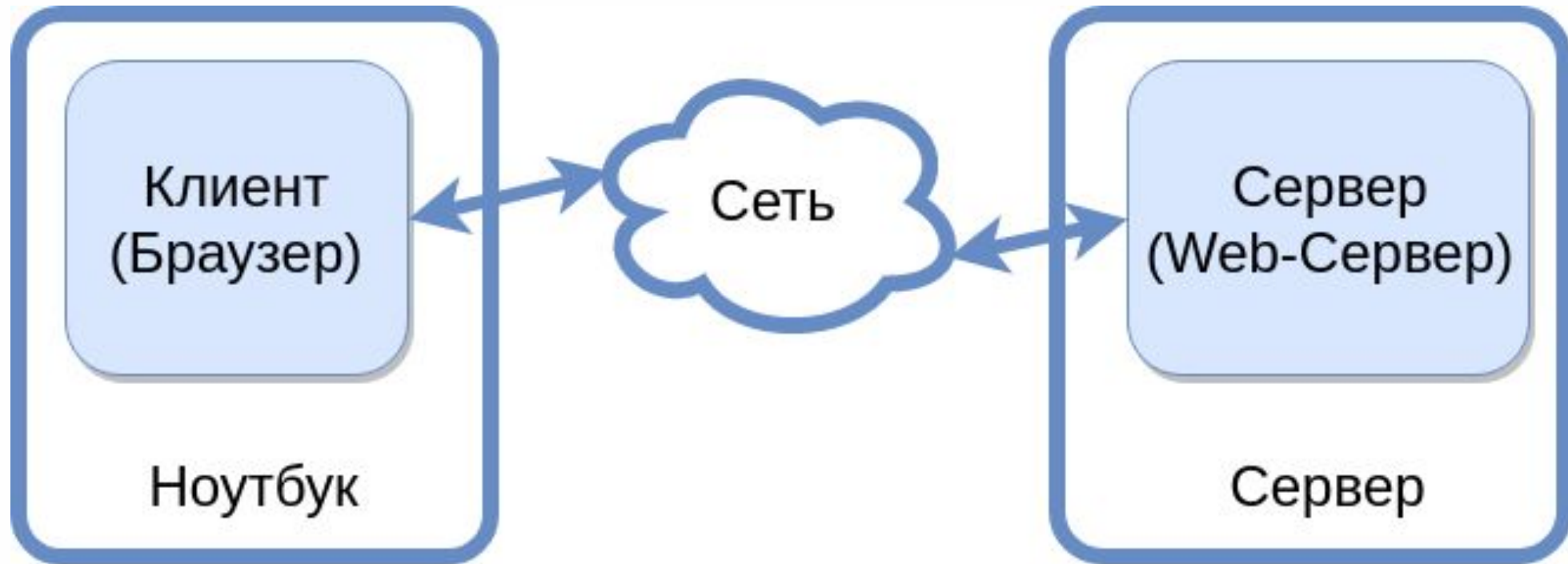
Веб-сервера работают (как правило) на серверах в датацентрах. Их задача заключается в хранении (или генерации) и отдаче документов.

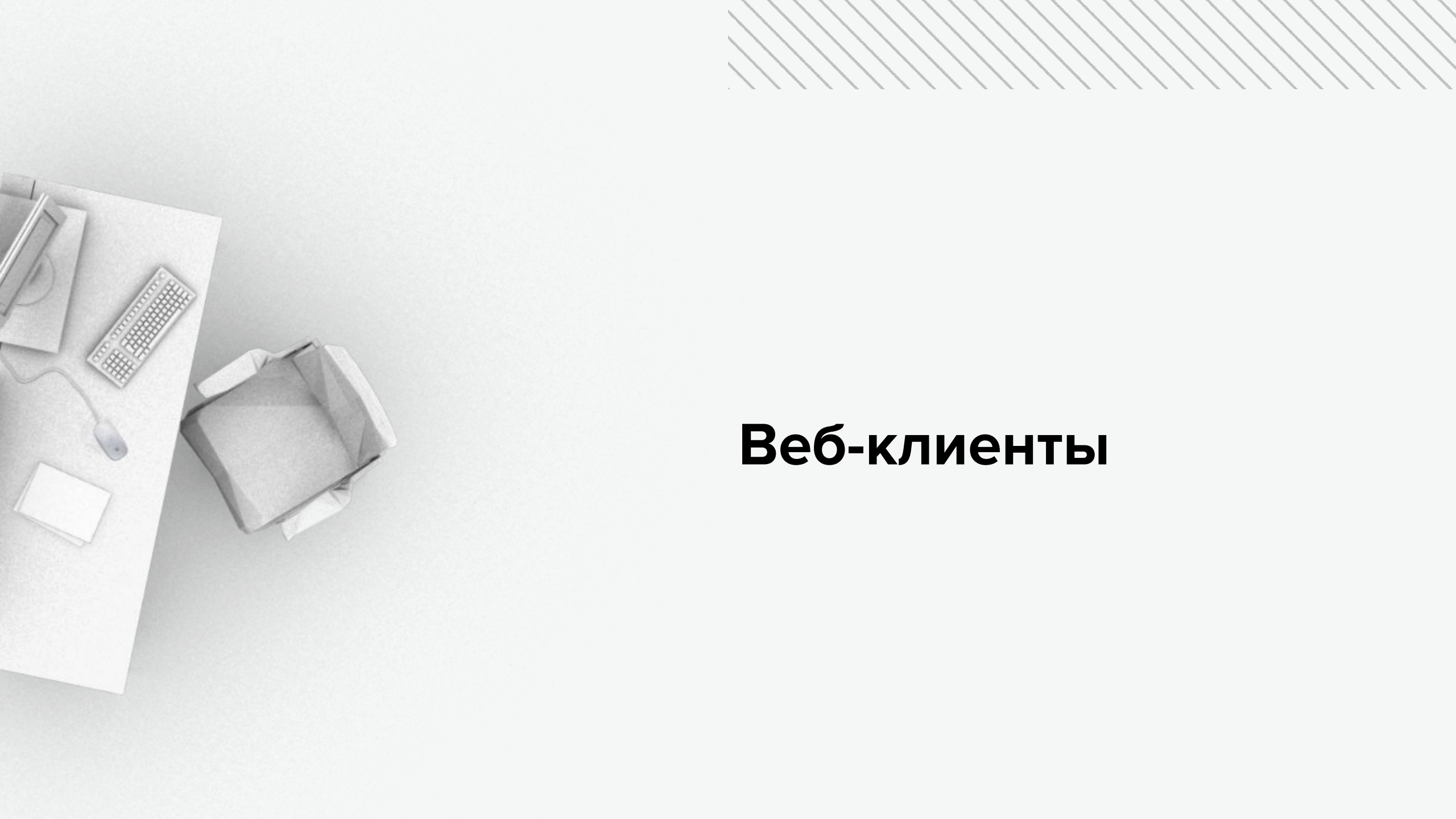


Преимущества подхода

- Открытый протокол;
- Стандартный клиент;
- Прозрачный способ взаимодействия веб-приложений между собой;
- Распределённая и масштабируемая система.

Клиент-серверная архитектура





Веб-клиенты



Разновидности веб-клиентов

- Библиотеки в ЯП: libcurl, urllib и т.д.;
- Консольные утилиты: wget, curl, telnet;
- Роботы: поисковики, вредоносные скрипты;
- Браузеры:
 - Полноценные: firefox, chrome и т.д.
 - Встроенные: web-view, webkit и т.д.




Особенности библиотек веб-клиентов

- Предоставляют максимум опций для работы с HTTP;
- Осуществляют кодирование/декодирование данных;
- Перенаправления, куки - опционально;

Назначение: используются внутри других программ для простоты работы с HTTP.



Назначение консольных клиентов

- 
- Автоматизация в shell-скриптах;
 - Создание статической копии сайта;
 - Отладка веб-приложений.

Пример отладки

Простейший GET-запрос:

```
curl -v 'https://python.org/'
```

POST-запрос:

```
curl -v -d POST -L -H 'User-agent: curl' 'https://python.org/'
```


Браузер

Браузер — это программа с графическим интерфейсом, которая позволяет html-документы.


Основное назначение браузера — отображение HTML-страниц.

Однако, возможности современных браузеров огромны.

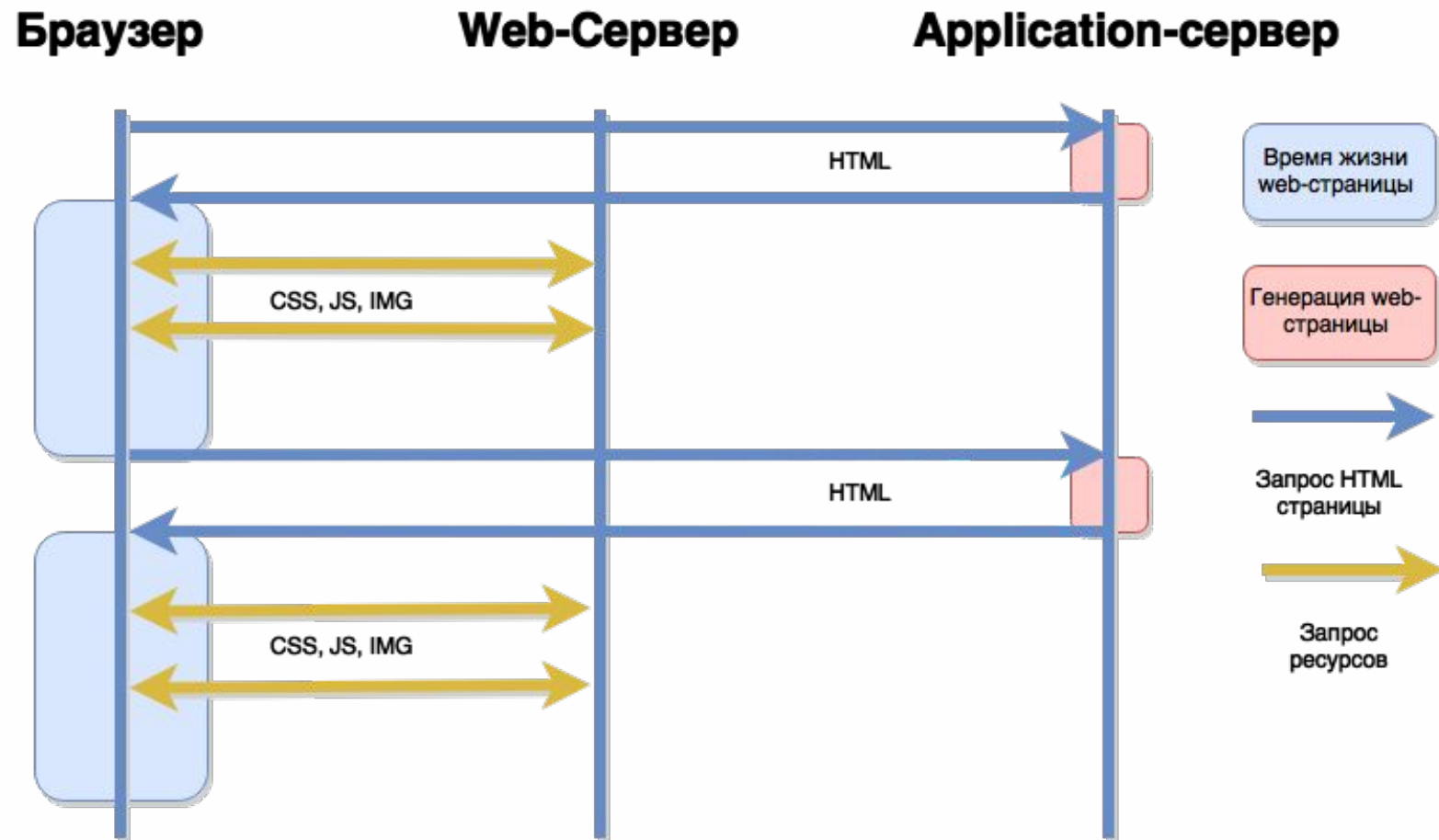
Существуют операционные системы и 3D-игры, работающие внутри браузеров!



Сценарий работы классического веб-приложения

- 
1. Пользователь вводит URL;
 2. Браузер загружает Web страницу — HTML документ;
 3. Браузер анализирует (parse) HTML и загружает доп. ресурсы;
 4. Браузер отображает (rendering) HTML страницу;
 5. Пользователь переходит по гиперссылке или отправляет форму;
 6. Цикл повторяется.

Сценарий работы классического веб-приложения

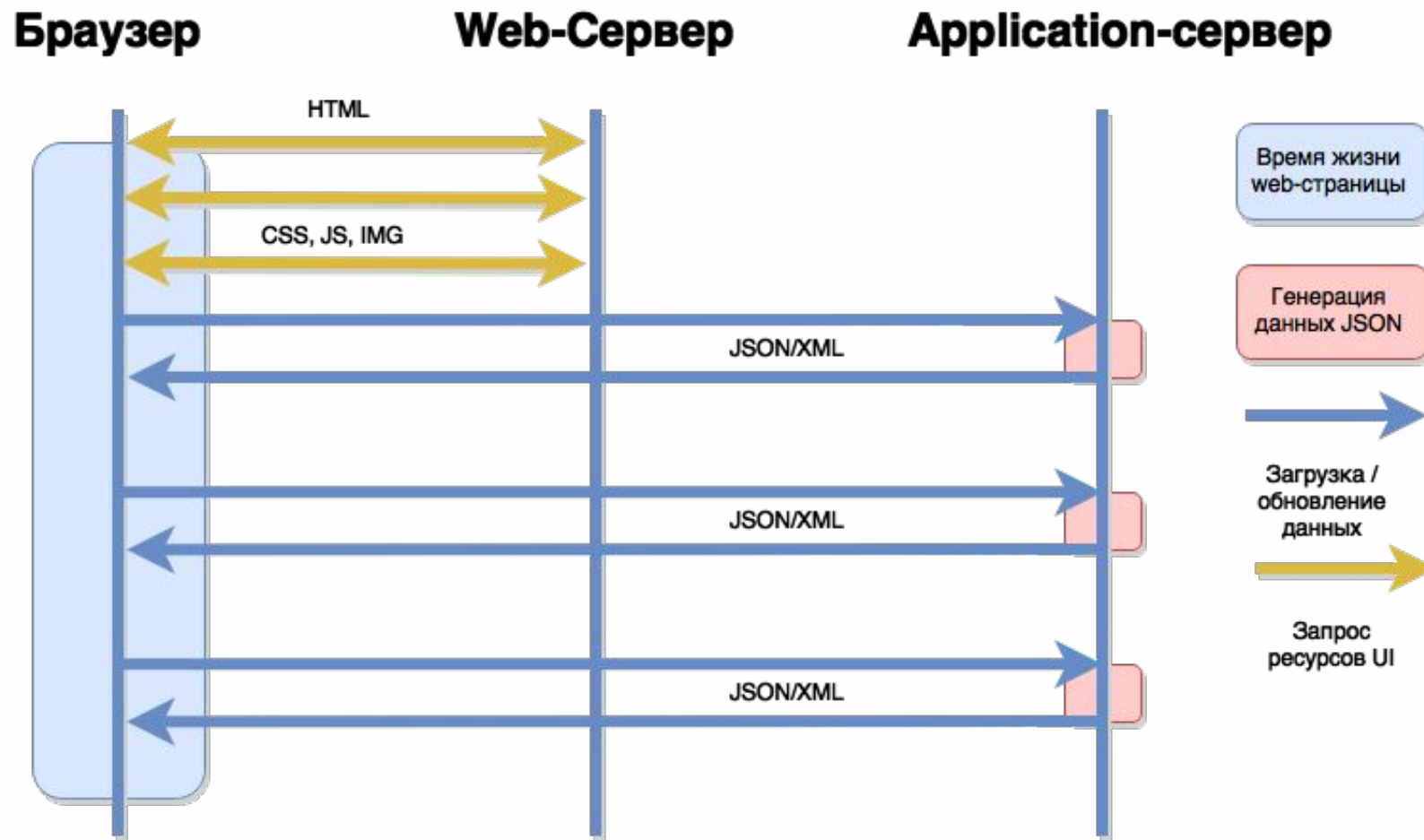




Сценарий работы современного веб-приложения

- Браузер загружает Web страницу, ресурсы и отображает её;
- JavaScript загружает данные с помощью AJAX запросов;
- JavaScript обеспечивает полноценный UI на странице;
- Пользователь взаимодействует с UI, что приводит к вызову JavaScript обработчиков;
- JavaScript обновляет данные на сервере или загружает новые данные, используя AJAX.

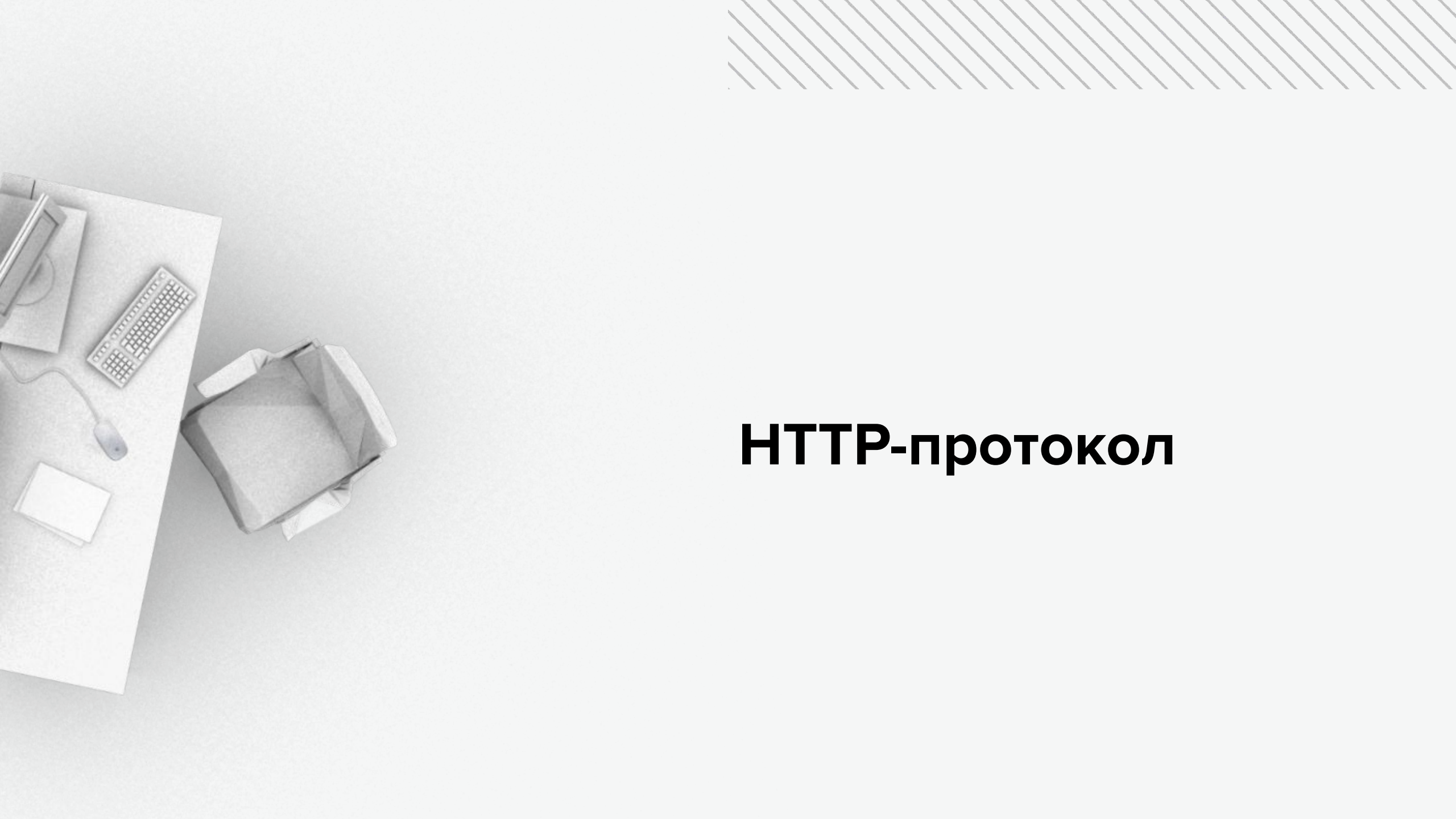
Сценарий работы современного веб-приложения





Особенности современных веб-приложений

- UI находится на одной или нескольких страницах (single page)
- UI полностью статичен: HTML, CSS, JS - статические файлы
- Логика UI полностью работает на стороне клиента
- Используется шаблонизация в JavaScript
- Application сервер возвращает чистые данные (JSON или XML, а не HTML)



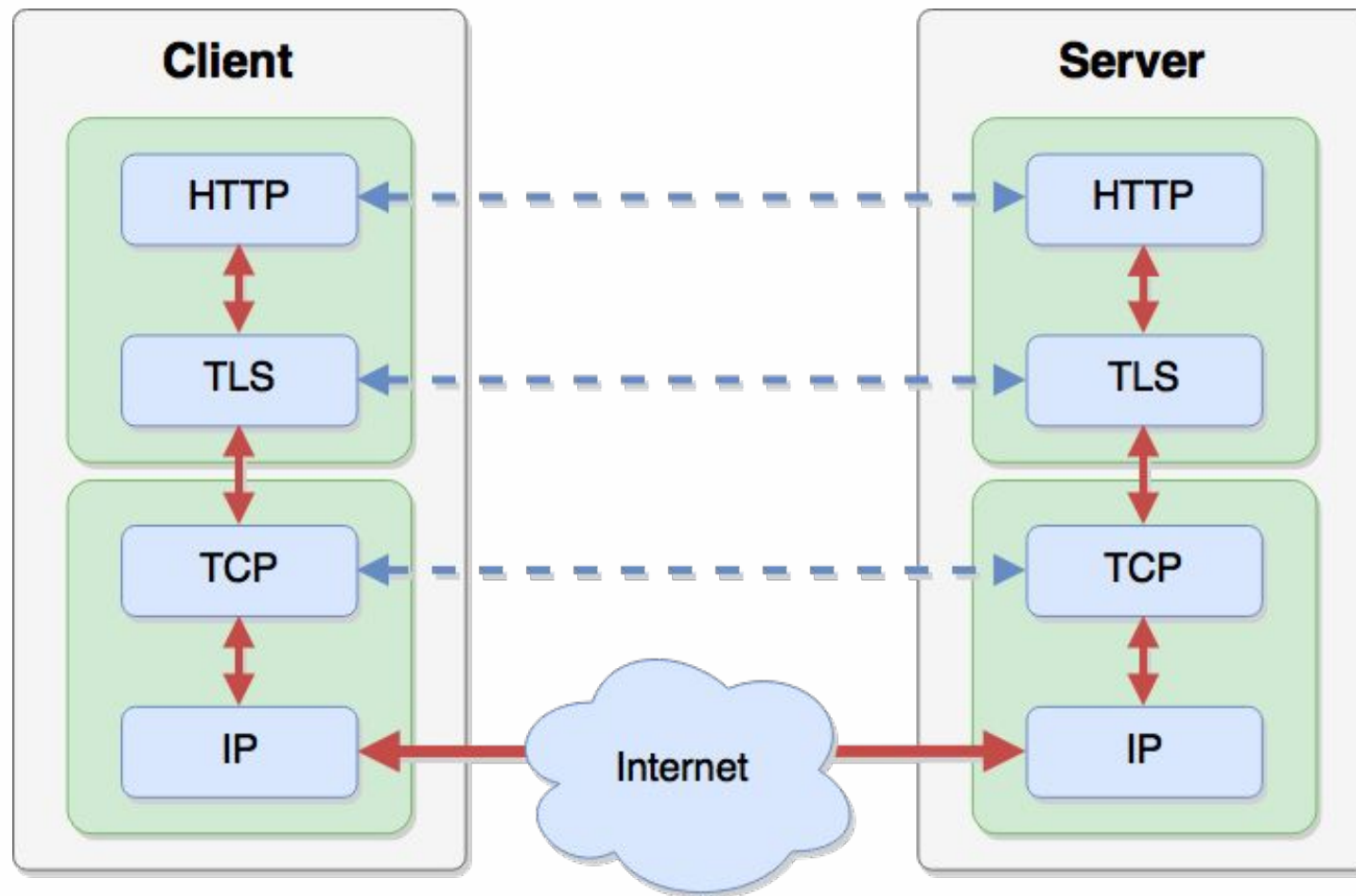
HTTP-протокол



Как задачи решает HTTP?

- Передача документов;
- Передача мета-информации;
- Авторизация;
- Поддержка сессий;
- Кеширование документов;
- Согласование содержимого (negotiation);
- Управление соединением.

Как происходит HTTP-запрос?





Ключевые особенности HTTP

- Работает поверх TCP/TLS;
- Протокол запрос-ответ;
- Не поддерживает состояние (соединение) — stateless;
- Текстовый протокол;
- Расширяемый протокол.



HTTP запрос состоит из

- строка запроса:
 - метод,
 - URL документа,
 - версия.
- заголовки;
- тело запроса;

HTTP/1.0 запрос

GET http://www.ru/robots.txt HTTP/1.0

Accept: text/html, text/plain

User-Agent: telnet/hands

If-Modified-Since: Fri, 24 Jul 2015 22:53:05 GMT

Перевод строки — \r\n

HTTP/1.0 запрос

GET /robots.txt HTTP/1.1

Accept: text/html,application/xhtml+xml

Accept-Encoding: gzip, deflate

Cache-Control: max-age=0

Connection: keep-alive

Host: www.ru

User-Agent: Mozilla/5.0 Gecko/20100101 Firefox/39.0

HTTP/1.1 ответ

HTTP/1.1 404 Not Found

Server: nginx/1.5.7

Date: Sat, 25 Jul 2015 09:58:17 GMT

Content-Type: text/html; charset=iso-8859-1

Connection: close

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<HTML><HEAD>...

Методы HTTP-запроса

- GET — получение документа;
- HEAD — получение только заголовков;
- POST — отправка данных на сервер;
- PUT — отправка документа на сервер;
- DELETE — удаление документа;
- CONNECT, TRACE, OPTIONS — используются редко;
- COPY, MOVE, MKCOL — расширения WebDAV.

HTTP-коды ответов

- 1xx — информационные;
- 2xx — успешное выполнение;
- 3xx — перенаправления;
- 4xx — ошибка на стороне клиента;
- 5xx — ошибка на стороне сервера.

HTTP-коды ответов (1)

200 OK — запрос успешно выполнен;

204 No Content — запрос успешно выполнен, но документ пуст;

301 Moved Permanently — документ сменил URL;

302 Found — повторить запрос по другому URL;

304 Not Modified — документ не изменился, использовать кеш.

HTTP-коды ответов (2)

400 Bad Request — неправильный синтаксис запроса;

401 Unauthorized — требуется авторизация;

403 Forbidden Moved Permanently — нет доступа (неверная авторизация);

404 Not Found — документ не найден;

500 Internal Server Error — неожиданная ошибка сервера;

502 Bad Gateway — проксируемый отвечает с ошибкой;

504 Gateway Timeout — проксируемый сервер не отвечает;

Заголовки HTTP (общие)

Для управления соединением и форматом сообщения (документа):

- `Content-Type` — mime-тип документа;
- `Content-Length` — длина сообщения;
- `Content-Encoding` — кодирование документа, например, gzip-сжатие;
- `Transfer-Encoding` — формат передачи, например, chunked;
- `Connection` — управление соединением;
- `Upgrade` — смена протокола.

Заголовки HTTP-запросов

- `Authorization` — авторизация, чаще всего логин/пароль;
- `Cookie` — передача состояния (сессии) на сервер;
- `Referer` — URL предыдущего документа, контекст запроса;
- `User-Agent` — описание web-клиента, версия браузера;
- `If-Modified-Since` — условный GET запрос;
- `Accept-*` — согласование (negotiation) содержимого.

Заголовки HTTP-ответов

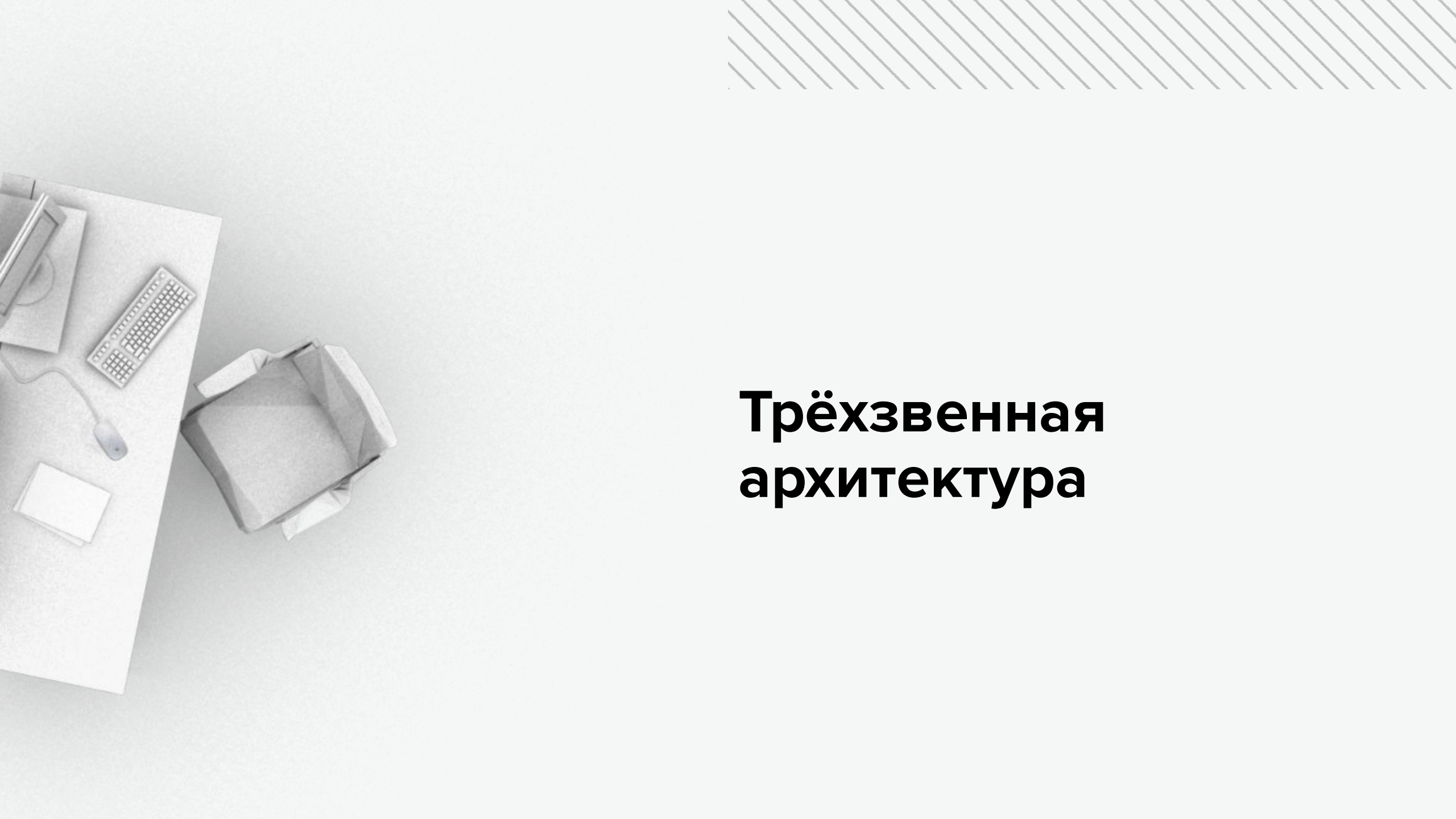
- `Location` — новый URL документа при перенаправлениях (коды 301, 302);
- `Set-Cookie` — установка состояния (сессии) в браузере;
- `Last-Modified` — дата последнего изменения документа;
- `Date` — Дата на сервере, для согласования кешей;
- `Server` — описание web-сервера, название и версия.

Логика управления в HTTP/1.1

Соединение должно быть закрыто, если:

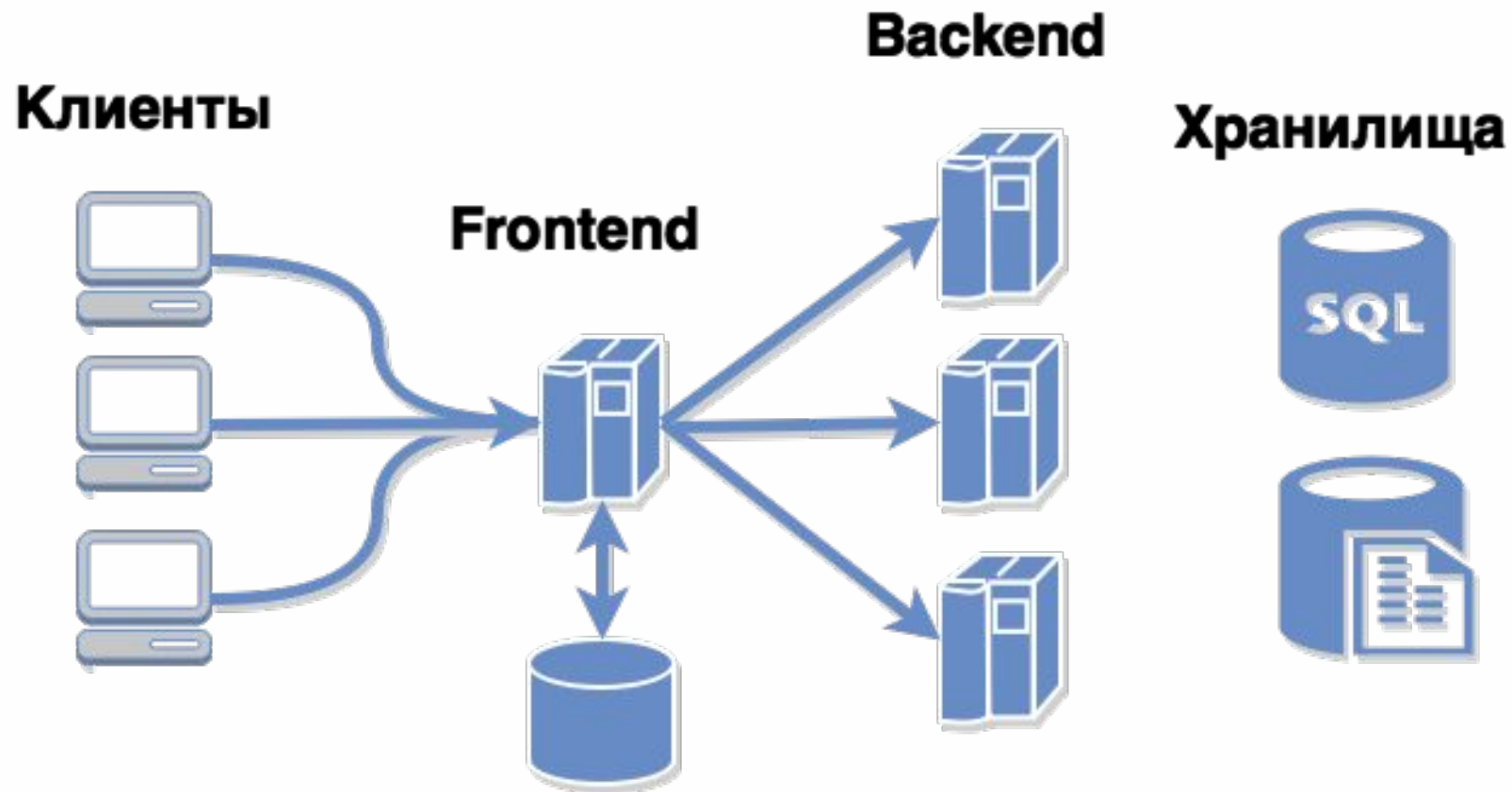
- сервер или клиент использует HTTP младше 1.1;
- сервер или клиент передал заголовок Connection: close;
- по истечении таймаута (обычно небольшой, около 10 с);

Иначе соединение остается открытым для последующих запросов.



Трёхзвенная архитектура

Общая архитектура





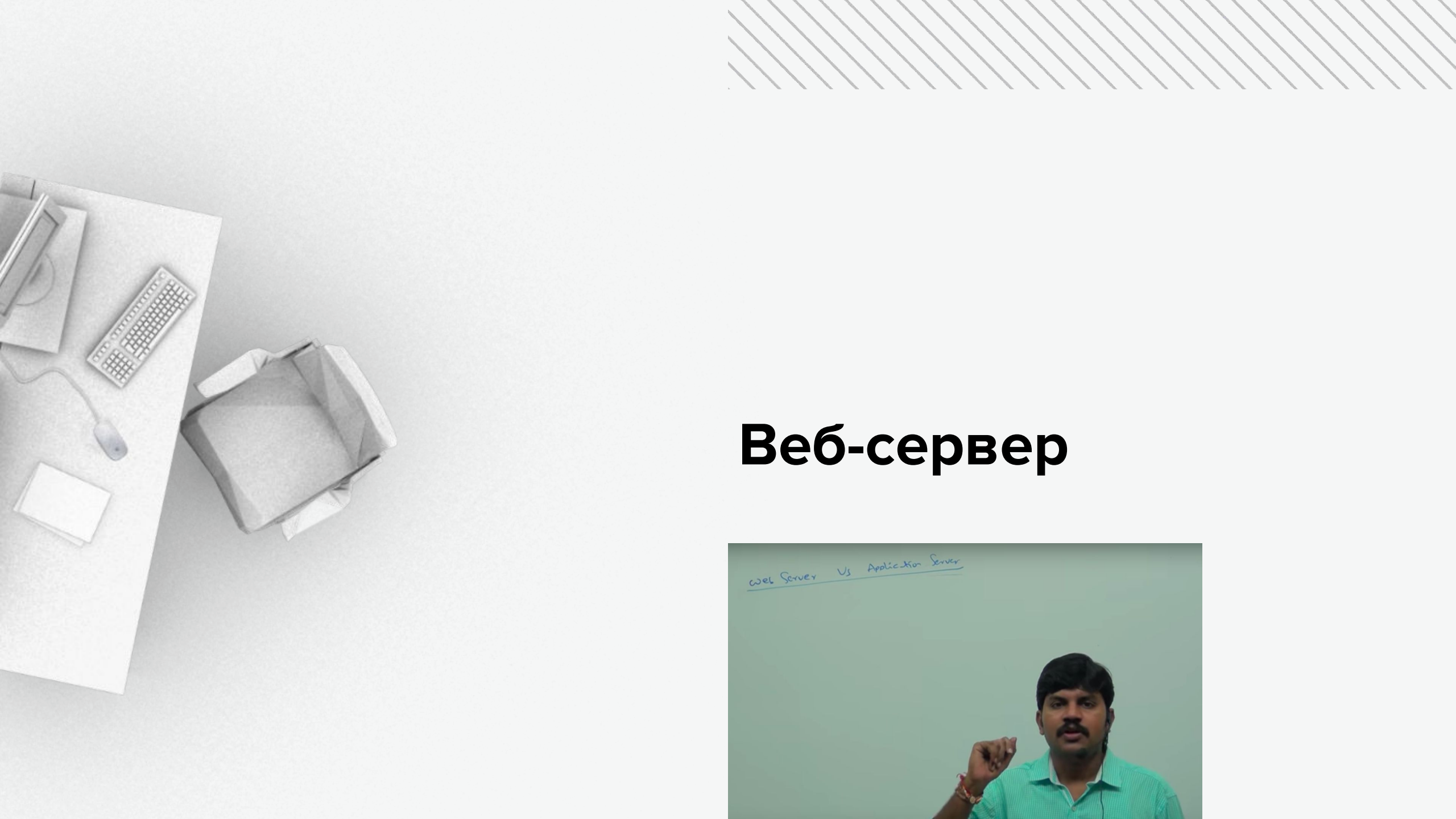
Задача Frontend (веб) сервера

- отдача статических документов;
- проксирование (reverse proxy);
- балансировка нагрузки;
- кеширование;
- сборка SSL;
- авторизация, SSL, нарезка картинок, gzip.

Reverse proxy

- frontend (медленно) читает запрос от клиента;
- frontend (быстро) передает запрос свободному backend;
- backend генерирует страницу;
- backend (быстро) возвращает ответ frontend серверу;
- frontend (медленно) возвращает ответ клиенту.

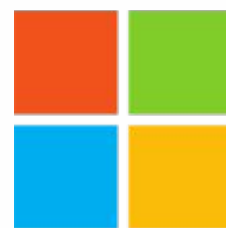
Результат: backend занят минимально возможное время.



Веб-сервер



Веб-сервера



Microsoft
IIS

Запуск веб-сервера

```
# Установка в Ubuntu
sudo apt install nginx
# Установка в MacOS
brew install nginx
```

- Команда на запуск;
`sudo /etc/init.d/nginx start`
- Чтение файла конфигураций;
- Получение порта 80;
- Открытие (создание) логов;
- Понижение привилегий;
- Запуск дочерних процессов/потоков;
- Готов к обработке запросов.

Файлы веб-сервера

Конфиг `/etc/nginx/nginx.conf` (для Ubuntu)

Конфиг `/usr/local/etc/nginx/nginx.conf` (для MacOS)

Init-скрипт `/etc/init.d/nginx` `[start|stop|restart]`

PID-файл `/var/run/nginx.pid`

Error-лог `/var/log/nginx/error.log`

Access-лог `/var/log/nginx/access.log`



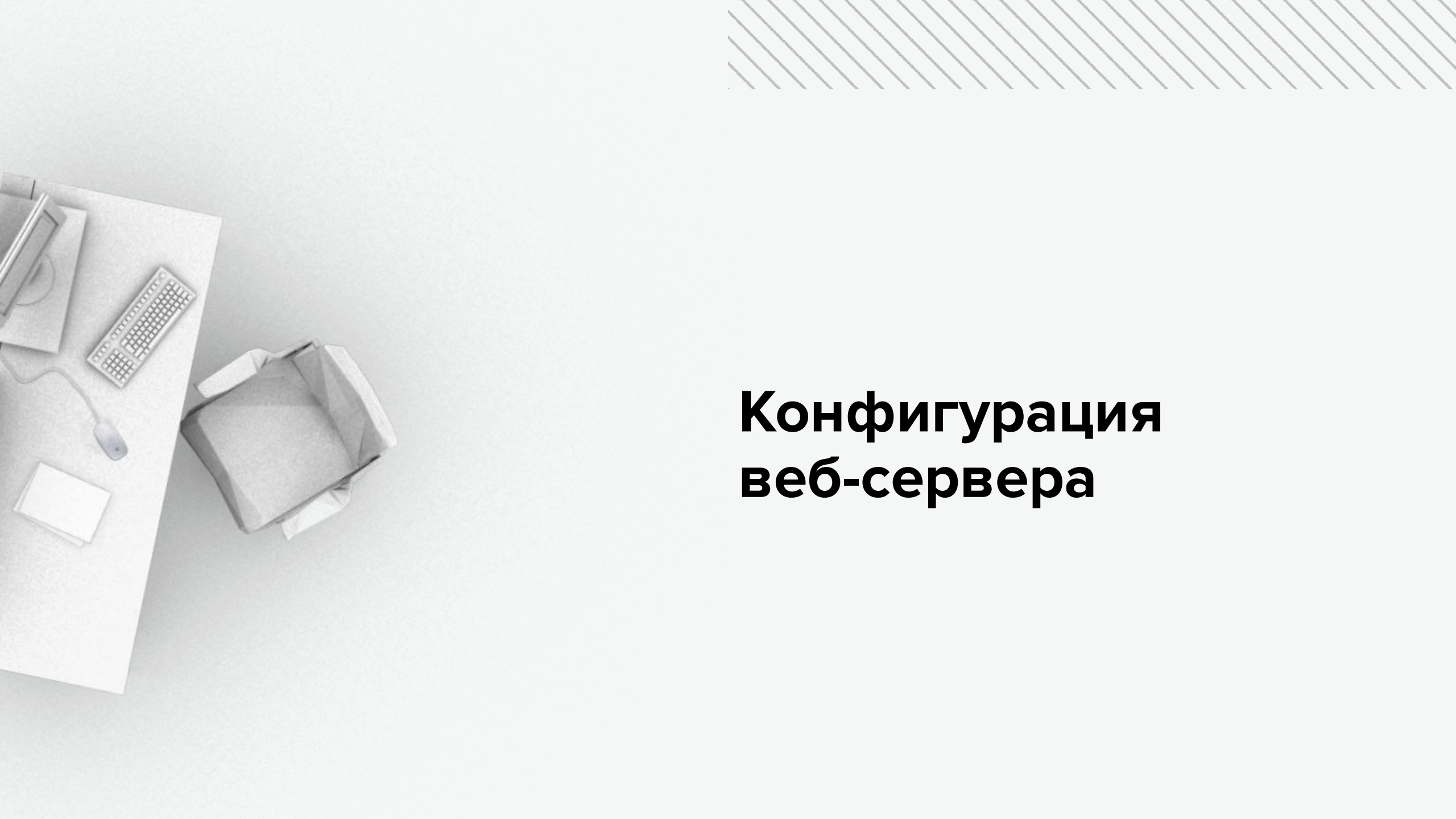
Процессы веб-сервера

- Master (root, 1 процесс)
 - Чтение и валидация конфига;
 - Открытие сокета(ов) и логов;
 - Запуск и управление дочерними процессами (worker);
 - Graceful restart, Binary updates.
- Worker (nobody, 1+процессов)
 - Обработка входящих запросов.



Процессы веб-сервера


- Master (root, 1 процесс)
 - Чтение и валидация конфига;
 - Открытие сокета(ов) и логов;
 - Запуск и управление дочерними процессами (worker);
 - Graceful restart, Binary updates.
- Worker (nobody, 1+процессов)
 - Обработка входящих запросов.



Конфигурация веб-сервера



Терминология



virtual host, вирт. хост — секция конфига web сервера, отвечающая за обслуживание определенного домена.

location — секция конфига, отвечающая за обслуживание определенной группы URL.

Структура конфига nginx

- nginx состоит из модулей, которые настраиваются директивами;
- директивы:
 - простые (`worker_processes 2;`)
 - блочные (`http{ server{} }`)
- `http`, `events` внутри `main`, `server` внутри `http`, `location` внутри `server`.

Основные директивы

- `user` — пользователь, от лица которого будут запущены;
- `worker_processes` — количество дочерних процессов;
- `error_log` — файл, в который записываются ошибки и уровень ошибок;
- `http` — конфиг веб-сервера;
- `include` — включает содержимое файла;
- `log_format` — формат записи в `access_log`;
- `server` — virtual host;

Приоритеты location

1. `location = /img/1.jpg`
2. `location ^~ /pic/`
3. `location ~* \.jpg$`
4. `location /img/`

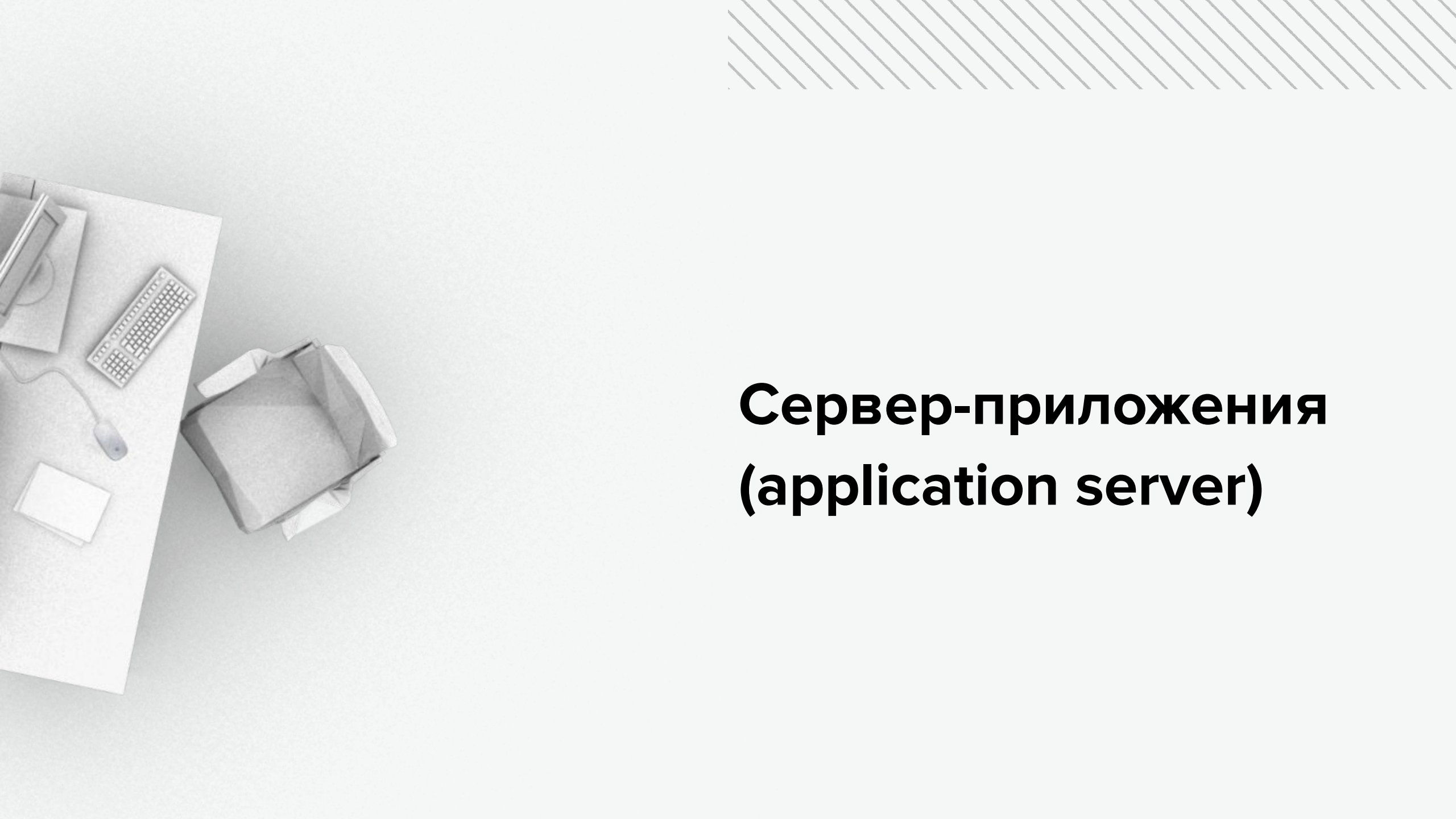
При одинаковом приоритете используется тот `location`, что находите **выше** в конфиге.

Отдача статических документов

```
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)



Backend (application) сервер

Роль application сервера заключается в исполнении бизнес-логики приложения и генерации динамических документов.


На каждый HTTP запрос application сервер запускает некоторый обработчик в приложении. Это может быть функция, класс или программа, в зависимости от технологии.

Подробнее про различие web server и application server:

<https://youtu.be/BcmUOmvI1N8>



Протоколы запуска приложений

- 
1. Servlets и др. специализированные API
 2. mod_perl, mod_python, mod_php
 3. CGI
 4. FastCGI
 5. SCGI
 6. PSGI, **WSGI**, Rack



CGI — Common Gateway Interface

- Метод, QueryString, заголовки запроса — через **переменные окружения**;
- Тело запроса передаётся через **STDIN**;
- Заголовок и тело ответа возвращаются через **STDOUT**;
- HTTP-код ответа передаётся через псевдозаголовок **Status**;
- Поток ошибок **STDERR** направляется в лог ошибок сервера.

Переменные окружения CGI


- `REQUEST_METHOD` — метод запроса,
- `PATH_INFO` — путь из URL,
- `QUERY_STRING` — фрагмент URL после `?`,
- `REMOTE_ADDR` — IP-адрес пользователя,
- `CONTENT_LENGTH` — длина тела запроса,
- `HTTP_COOKIE` — Заголовок `Cookie`,
- `HTTP_ANY_HEADER_NAME` — любой другой HTTP-заголовок.



WSGI



WSGI — актуальный протокол



WSGI, PSGI, Rack — протоколы вызова функции обработчика из application сервера. Сам application server при этом может выполняться в отдельном процессе или совпадать с web сервером.

Как правило, при использовании этих протоколов в качестве application сервера выступает отдельный легковесный процесс.

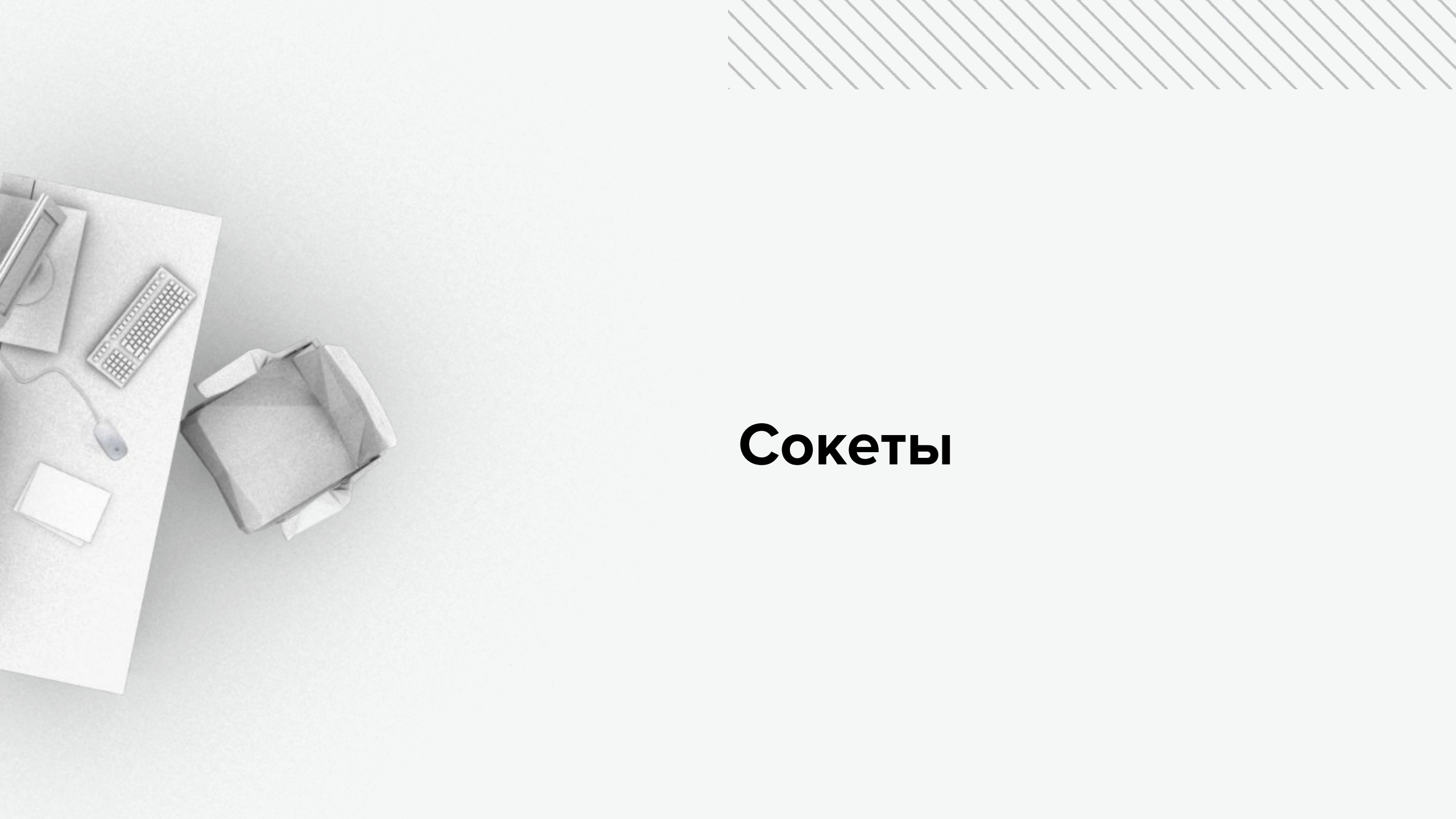
Простое WSGI-приложение

1. `pip install gunicorn`
2. `pip freeze > requirements.txt`
3. `cat myapp.py`
4.

```
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])
```
4. `gunicorn --workers 4 myapp:app`

Web Server Gateway Interface

- Обработчик — функция или класс (callable);
- Метод QueryString, заголовки запроса - через аргумент **environ**;
- Тело запроса передаётся через file-handle **wsgi.input**;
- HTTP-код ответа и заголовки ответа передаются через вызов функции **start_response**;
- Тело ответа возвращается в виде списка (*iterable*) из обработчика;
- Поток ошибок должен быть направлен в file-handle **wsgi.stderr**.



Сокеты



Сокет

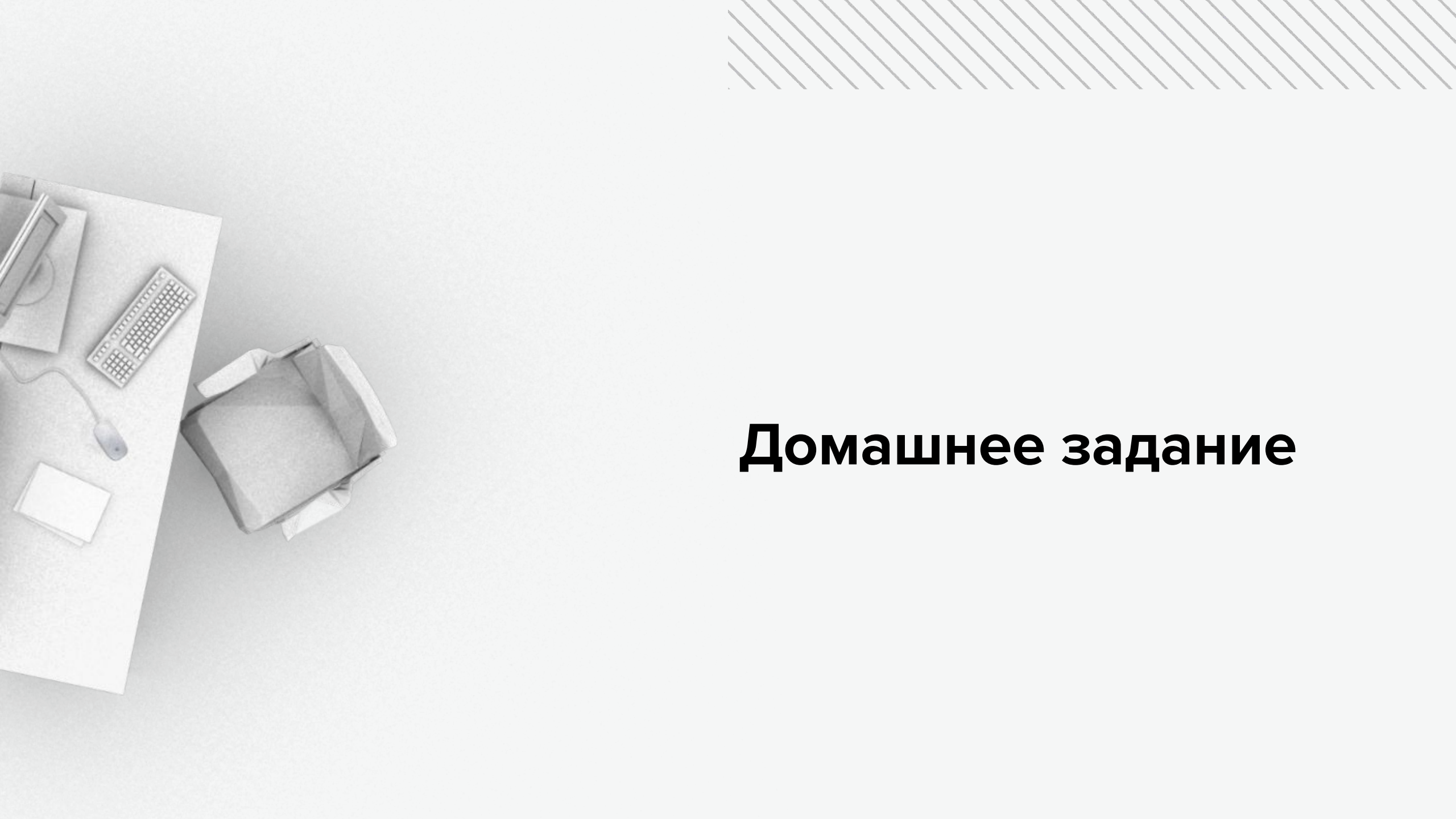
- Вид межпроцессного взаимодействия;
- Каждый процесс может создать слушающий сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы;
- Слушающий процесс обычно находится в цикле ожидания, просыпается при появлении нового соединения;
- Каждый сокет имеет свой адрес;

Создание сокета. Сервер

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('', 10001))
sock.listen()
conn, addr = sock.accept()
print('connected:', addr)
while True:
    data = conn.recv(1024)
    if not data:
        break
    conn.send(data.upper())
conn.close()
```

Создание сокета. Клиент

```
import socket
sock = socket.socket()
sock.connect(('127.0.0.1', 10001))
sock.sendall('hello, world!'.encode('utf-8'))
data = sock.recv(1024)
sock.close()
print(data.decode('utf-8'))
```



Домашнее задание

Домашнее задание #5

Разработать клиент-серверное приложение.

- Есть приложение клиента, есть приложение сервера;
- Сервер прослушивает открывает сетевой сокет по порту;
- Порт берётся из файла конфига;
- Клиентское приложение посылает запрос по порту сервера и ждёт ответ;
- Сервер получает запрос, парсит его, делает бизнес-логику (например ходит за информацией в сторонний сервис по API), формирует результат и возвращает клиенту.
- Клиент печатает в стандартный вывод ответ сервера, клиент завершает работу.



Домашнее задание по уроку #5

Домашнее задание №5

#078

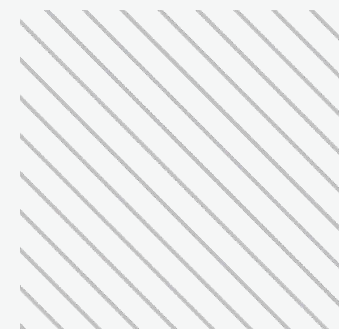
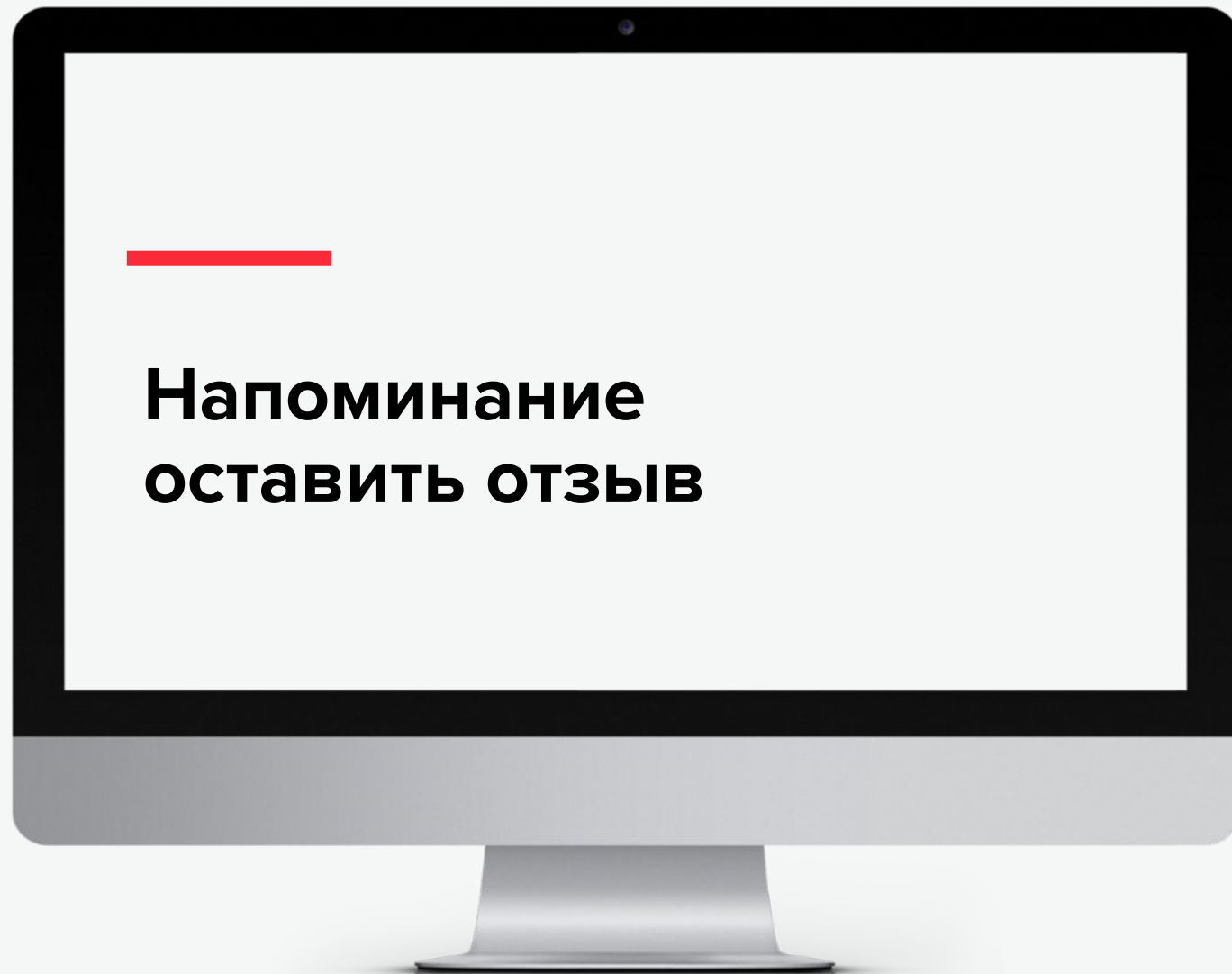
8

Баллов
за задание

Сроков нет, но
вы держитесь

Срок
сдачи

- Разработать клиент-серверное приложение.



**СПАСИБО
ЗА ВНИМАНИЕ**

