

6. Среда исполнения веб-приложения. Протоколы HTTP и CGI

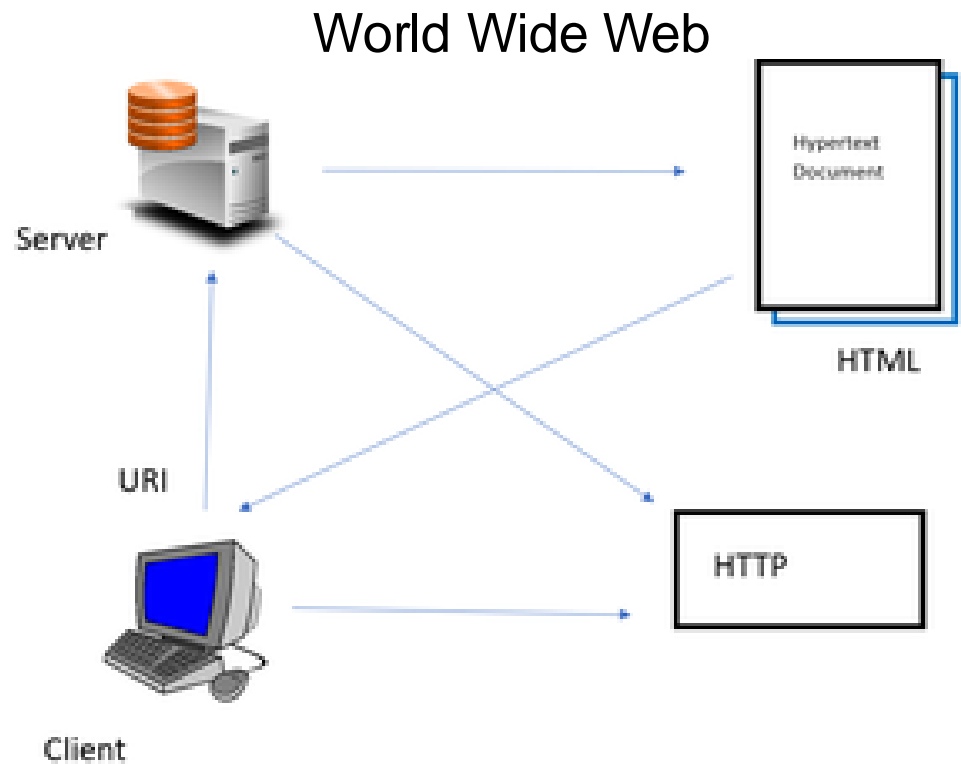
Среда исполнения веб-приложения

Веб приложение работает в вебе



Механизм работы веб-приложения

Интернет vs веб (World Wide Web)



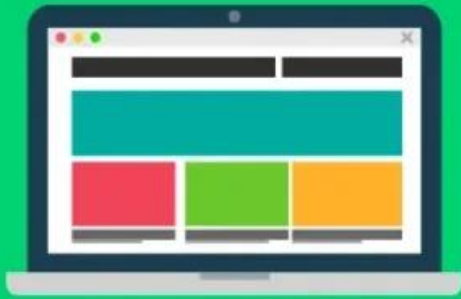
Интернет vs веб

	Internet	World Wide Web
Определение	Глобальная сеть сетей, объединяющая миллионы устройств	Служба, работающая поверх Интернета, предоставляющая доступ к гипертекстовым документам
Функция	Передача данных между устройствами	Просмотр и взаимодействие с гипертекстовыми документами через браузеры
Протоколы общения устройств	TCP/IP, UDP, ICMP, SMTP, FTP, SSH, DNS, и др.	HTTP, HTTPS
Инфраструктура	Серверы, маршрутизаторы, кабели, Wi-Fi, спутники	Веб-серверы, браузеры, HTTP-клиенты
Примеры использования	Электронная почта, VoIP, онлайн-игры, файловые передачи	Просмотр веб-сайтов, чтение новостей, просмотр видео
Создатель	Развитие началось в 1960-х годах (ARPANET)	Создан Тимом Бернерсом-Ли в 1989 году

Веб-приложения



Две части веб-приложения



FRONTEND



BACKEND

Влияние среды исполнения

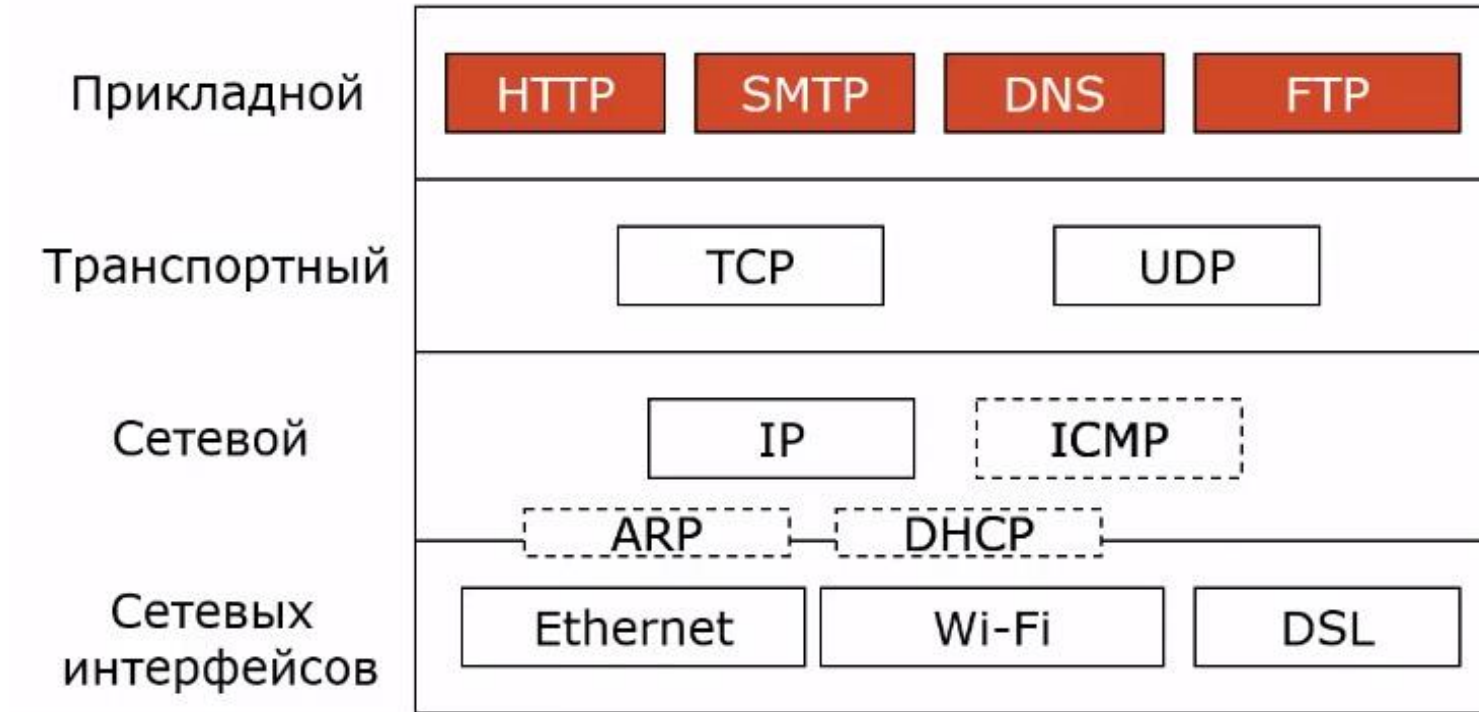
Среда исполнения определяет основную архитектуру приложения

- **Фронтенд.** Приложение с GUI, работающее в браузере. Архитектура - событийная модель
- **Бэкенд.** Консольное приложение на сервере, общающемся с клиентом

Наша задача - изучить механизм общения клиента с сервером и понять, как он влияет на архитектуру бекенд-приложения.

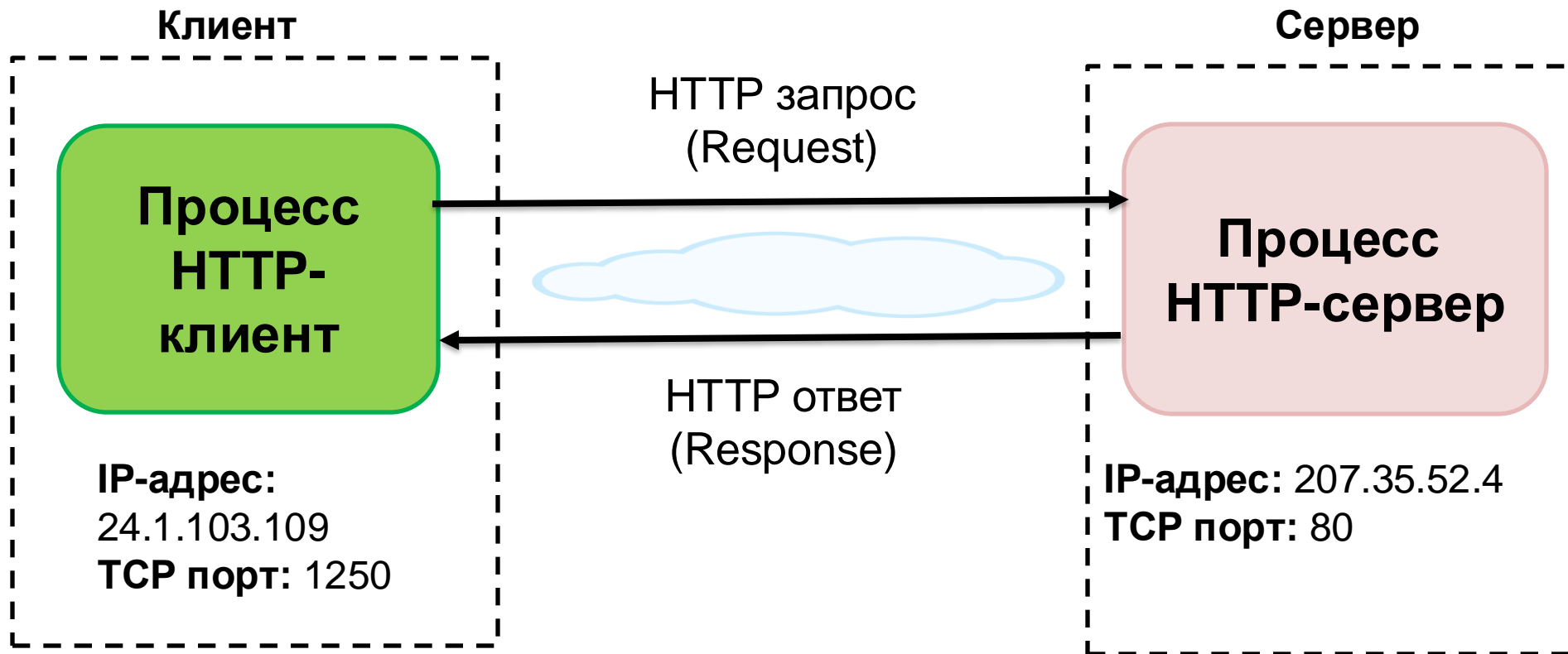
Протокол HTTP

Hypertext Transfer Protocol (HTTP)



Модель TCP/IP

Hypertext Transfer Protocol (HTTP)



1. Клиент устанавливает TCP соединение.
2. Клиент отправляет HTTP-запрос и ждёт ответа.
3. Сервер обрабатывает запрос и посылает HTTP-ответ.
4. Сервер разрывает соединение (HTTP 1.0).

Протокол HTTP

- HTTP - текстовый протокол прикладного уровня (поверх TCP)
- Процесс-клиент соединяется с процессом-сервером по протоколу TCP
- Подключение по IP-адресу и номеру порта (HTTP – 80, HTTPS – 443)

`telnet` : подключение по TCP к серверу, ввод HTTP-запроса, получение HTTP-ответа от сервера

Инструменты для работы с HTTP-запросами

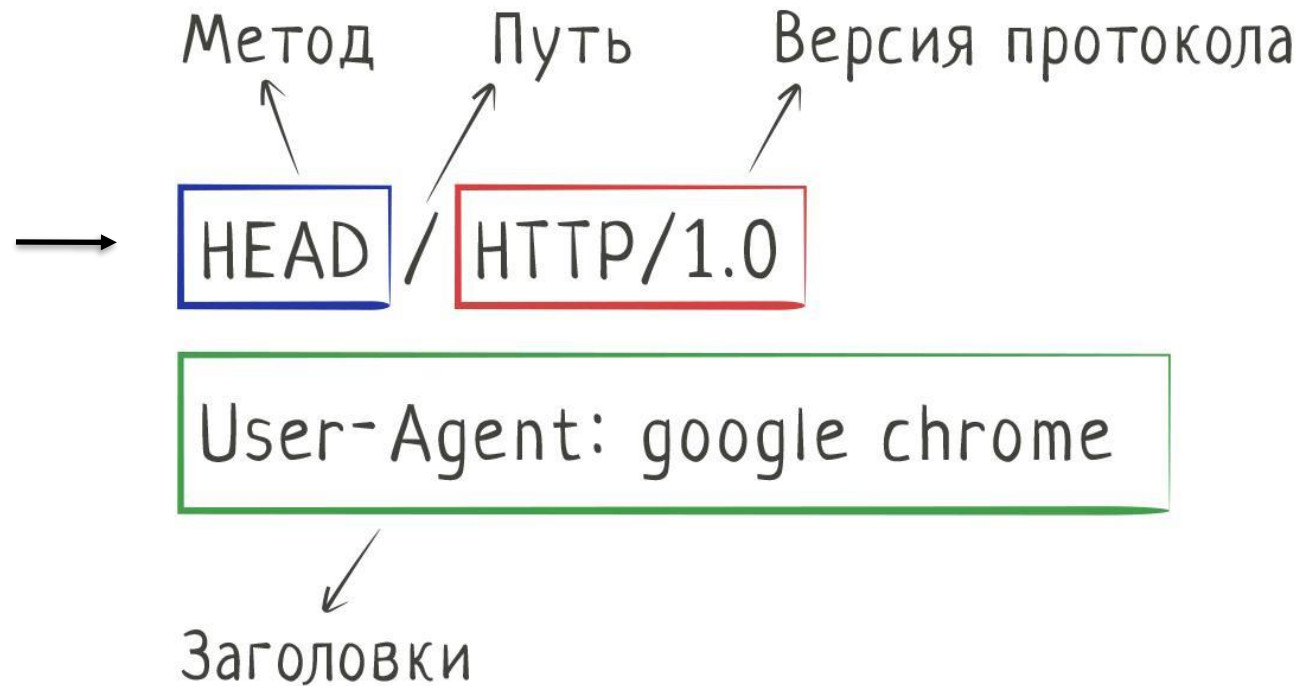
- Браузер
 - Вкладка Network в инструментах разработчика
- Консольные утилиты
 - `curl`, `wget`
 - `Invoke-WebRequest`, `Invoke-RestMethod` (PowerShell)
- Утилиты, встроенные в IDE
- Графические утилиты или веб-интерфейс
 - Postman

Базовая структура запроса и ответа

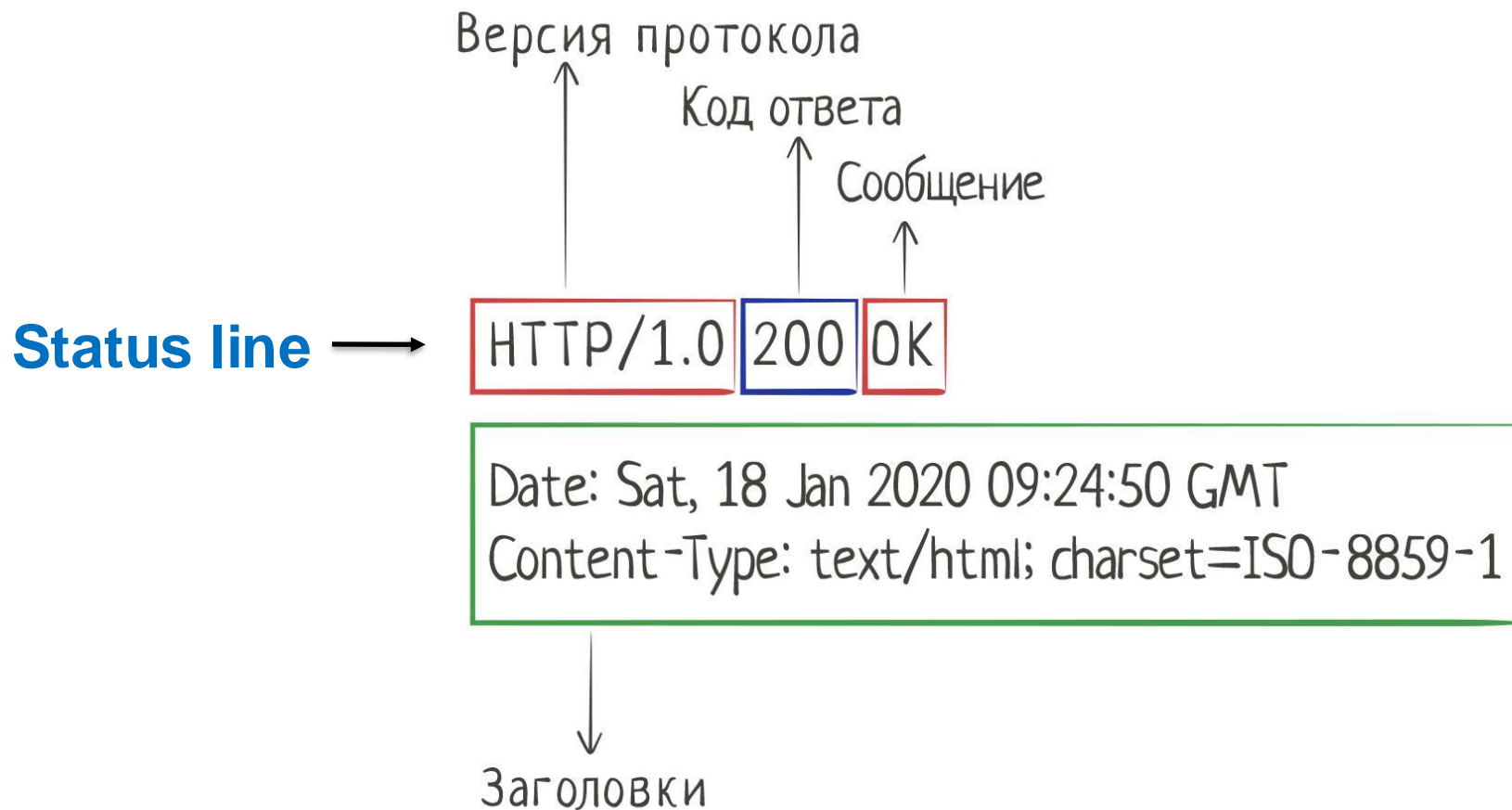
Структура HTTP запроса (request)

Протокол HTTP 1.0 - 1996 год

**Request line
(starting line)**



Структура HTTP ответа (response)



HTTP ответ (response)

```
andrey@mono:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
→ ~ telnet andpop.ru 80  
Trying 81.177.140.92...  
Connected to andpop.ru.  
Escape character is '^]'.  
GET / HTTP/1.0  
HTTP/1.1 403 Forbidden  
Date: Thu, 26 Nov 2020 13:37:17 GMT  
Content-Type: text/html  
Content-Length: 601  
Connection: close  
<!DOCTYPE html><html data-page="noservice" data-version="1.1.0">  
"content-type" content="text/html; charset=utf-8"><meta http-equi  
tent="IE=edge"><title>Сайт не обслуживается</title></head><body>  
бслуживается</h1><p>Запрошенный сайт не обслуживается на хостинг  
f="https://www.jino.ru/">Джино</a></p></noscript><div id="root">  
rking-static.jino.ru/static/main.js" charset="utf-8"></script></  
closed by foreign host.
```

Request line →

Status line →

Заголовки →

Пустая строка →

Тело →

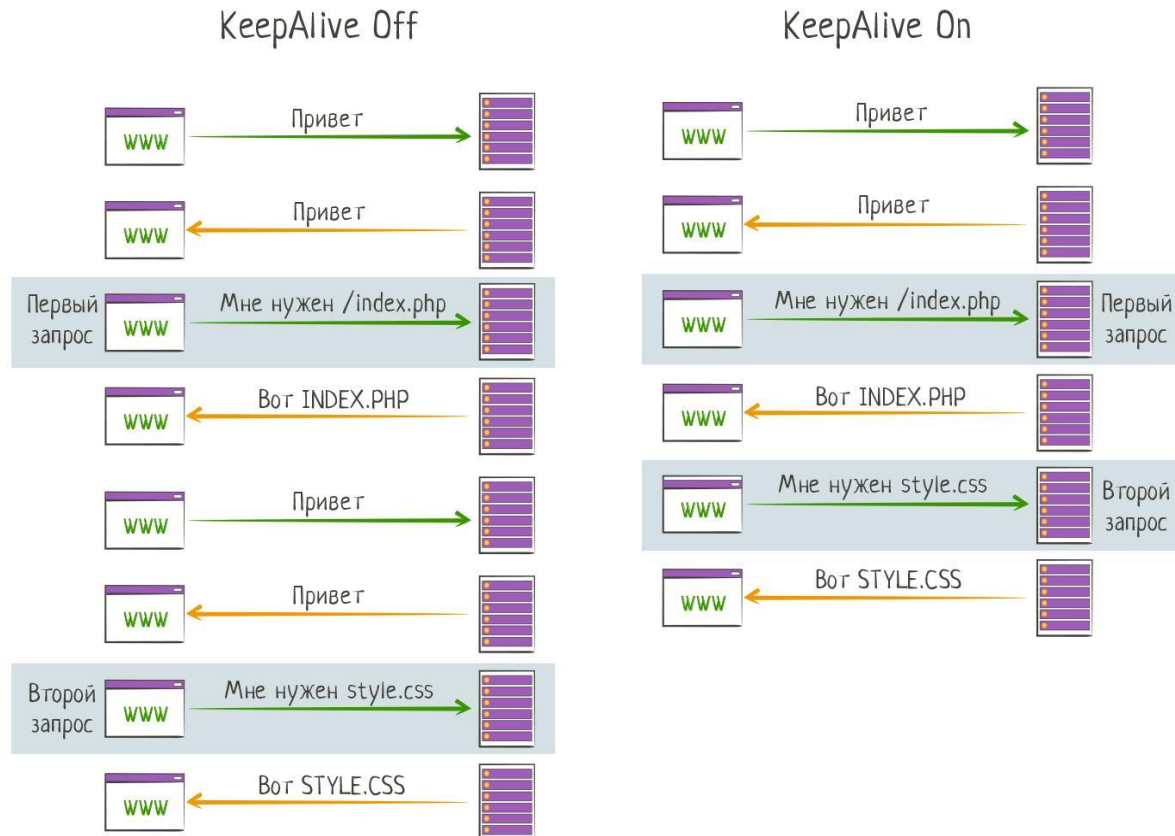
HTTP 1.1 (1999 год)

Более производительный, гибкий и подходящий для современного веба

- Поддержка постоянных соединений
- Поддержка виртуальных хостов (много доменов на одном IP-адресе)
- Новые методы
- Частичные запросы
- Механизм cookies
- Поддержка Unicode

HTTP 1.1

- Соединения остаются открытыми после завершения одного запроса-ответа, что позволяет отправлять несколько запросов через одно TCP-соединение.
- Для закрытия указывается заголовок `Connection: close`



HTTP 1.1

В самом начале появления веба на сервере размещался только один домен (один IP = один сайт)

Затем развивается хостинг, когда на одном IP-адресе размещается много сайтов (виртуальных хостов).

В протокол HTTP 1.1 (1999 г) добавлен обязательный для запроса заголовок Host.

```
HEAD / HTTP/1.1
```

```
Host: andpop.ru
```

```
User-Agent: google chrome
```

Хост в HTTP-запросе

```
andrey@mono:~  
Файл Правка Вид Поиск Терминал Справка  
→ ~ telnet andpop.ru 80  
INSERT --  
Trying 81.177.140.92...  
Connected to andpop.ru.  
Escape character is '^]'.  
GET / HTTP/1.1  
Host: andpop.ru  
HTTP/1.1 200 OK  
Date: Thu, 26 Nov 2020 13:56:56 GMT  
Content-Type: text/html  
Transfer-Encoding: chunked  
Connection: keep-alive  
Server: Jino.ru/mod_pizza  
Accept-Ranges: bytes  
Vary: Accept-Encoding  
f60  
<!DOCTYPE html>  
<html>  
<head>  
  <title>Андрей Попов - будни доцента</title>  
  <meta http-equiv="Content-Type" content="text/html; charset=  
  <meta name="description" content="Андрей Попов, доцент Мордо  
суниверситета. Материалы по учебным курсам, портфолио, резюме." />
```

Request line →
Заголовки →

Status line →

Заголовки →

Пустая строка →

Тело →

Методы запроса

HTTP-методы (глаголы)

Определяют тип операции, которую клиент хочет выполнить над ресурсом на сервере.

Каждый метод имеет свою семантику — то есть предназначение и поведение, которое он должен обеспечивать.

Виды запросов:

- Безопасный - не изменяет состояние сервера
- Идемпотентный - повторное выполнение дает тот же результат, что и первое выполнение.

HTTP-методы (глаголы)

1. GET

- **Семантика** : Запрос информации о ресурсе.
- **Особенности** :
 - Безопасный: не изменяет состояние сервера.
 - Идемпотентный: повторные вызовы дают тот же результат.
 - Может использоваться для получения данных (например, HTML-страниц, JSON, изображений).
- **Пример** :

```
http
```

```
1 GET /api/users/1 HTTP/1.1
```

```
2 Host: example.com
```

Этот запрос запрашивает информацию о пользователе с ID=1.

HTTP-методы (глаголы)

5. HEAD

- **Семантика** : Аналогичен `GET` , но без тела ответа.
- **Особенности** :
 - Безопасный: не изменяет состояние сервера.
 - Идемпотентный: повторные вызовы дают тот же результат.
 - Используется для проверки доступности ресурса или получения метаданных (например, заголовков).
- **Пример** :

```
http Копия  
1 HEAD /api/users/1 HTTP/1.1  
2 Host: example.com
```

Этот запрос проверяет, существует ли пользователь с ID=1, без получения содержимого.

HTTP-методы (глаголы)

2. POST

- **Семантика** : Создание нового ресурса или отправка данных на сервер.
- **Особенности** :
 - Не безопасный: может изменять состояние сервера.
 - Не идемпотентный: повторный запрос может создать дополнительные ресурсы.
 - Часто используется для создания новых записей или отправки форм.
- **Пример** :

```
http
1  POST /api/users HTTP/1.1
2  Host: example.com
3  Content-Type: application/json
4
5  {
6    "name": "John",
7    "age": 30
8  }
```

Этот запрос создает нового пользователя.

HTTP-методы (глаголы)

3. PUT

- **Семантика** : Обновление существующего ресурса или создание ресурса по указанному URI.
- **Особенности** :
 - Не безопасный: может изменять состояние сервера.
 - Идемпотентный: повторный запрос обновляет ресурс теми же данными.
 - В отличие от `POST` , `PUT` требует указания конкретного URI для обновления или создания ресурса.
- **Пример** :

```
http                                                                    Koi
1  PUT /api/users/1 HTTP/1.1
2  Host: example.com
3  Content-Type: application/json
4
5  {
6    "name": "John Updated",
7    "age": 31
8  }
```

Этот запрос обновляет данные пользователя с ID=1.

HTTP-методы (глаголы)

7. PATCH

- **Семантика** : Частичное обновление ресурса.
- **Особенности** :
 - Не безопасный: изменяет состояние сервера.
 - Не идемпотентный: повторный запрос может привести к другим изменениям.
 - Используется для обновления части данных ресурса, а не всего объекта целиком.
- **Пример** :

```
http
1  PATCH /api/users/1 HTTP/1.1
2  Host: example.com
3  Content-Type: application/json
4
5  {
6    "age": 32
7  }
```

Этот запрос обновляет только возраст пользователя с ID=1.

HTTP-методы (глаголы)

4. DELETE

- **Семантика** : Удаление ресурса.
- **Особенности** :
 - Не безопасный: удаляет ресурс.
 - Идемпотентный: повторный запрос к уже удаленному ресурсу не приводит к ошибке.
- **Пример** :

```
http
```

```
1 DELETE /api/users/1 HTTP/1.1
```

```
2 Host: example.com
```

Этот запрос удаляет пользователя с ID=1.

HTTP-методы и CRUD

Create (создание) - POST

Read (чтение) - GET

Update (обновление) - PUT и PATCH

Delete (удаление) - DELETE

Коды ответа сервера

HTTP-коды ответов

- **Автоматизация** Коды позволяют программам автоматически обрабатывать результаты запросов.
- **Отладка** Разработчики могут быстро определить причину ошибки.
- **Стандартизация** Обеспечивают единый язык взаимодействия между клиентами и серверами.

HTTP-коды ответов

1xx Информационный. Сервер принял запрос и продолжает его обработку.

Они редко используются в браузерах, но важны для других протоколов, типа WebSocket.

HTTP-коды ответов

2xx Успешный. Запрос был успешно принят, понят и обработан.

- **200 OK** Запрос успешен, сервер возвращает запрошенный ресурс.
- **201 Created** Ресурс создан (обычно после POST-запроса). В заголовке Location может быть указан URI нового ресурса.
- **204 No Content** Запрос выполнен успешно, но нет содержимого для возврата.

HTTP-коды ответов

3xx Перенаправление. Клиент должен выполнить дополнительное действие для завершения запроса (обычно повторный запрос к другому URI)

- **301 Moved Permanently** Ресурс перемещен на новый URI (постоянно).
- **302 Found** Ресурс временно доступен по другому URI (редирект).

HTTP-коды ответов

4xx Ошибки клиента. Неверный запрос со стороны клиента.

- **400 Bad Request** Неверный синтаксис запроса.
- **401 Unauthorized** Требуется аутентификация для доступа к ресурсу.
- **403 Forbidden** Доступ к ресурсу запрещен.
- **404 Not Found** Запрашиваемый ресурс не найден.
- **405 Method Not Allowed** Метод запроса запрещен для данного ресурса.

HTTP-коды ответов

5xx Ошибки сервера.

- **500 Internal Server Error** Произошла ошибка на сервере, подробности которой не уточняются.
- **501 Not Implemented** Сервер не поддерживает необходимую функциональность.
- **502 Bad Gateway** Сервер получил недопустимый ответ от upstream-сервера.
- **503 Service Unavailable** Сервер временно недоступен (например, из-за перегрузки).

**Передача данных от клиента
серверу**

Передача данных в HTTP-запросе

Два основных метода передачи данных

- **Query string (строка запроса).** Удобен для передачи небольших данных в URL, особенно при использовании метода GET.
- **Тело запроса.** Используется для передачи больших объемов данных или данных, которые не должны быть видны в URL, особенно при использовании методов POST, PUT или PATCH.

Оба метода могут сочетаться.

Передача данных в HTTP-запросе

1. Query String (Строка запроса)

Что это?

Query string — это часть URL, которая начинается после символа `?` и содержит параметры запроса в формате `ключ=значение`. Параметры разделяются амперсандом (`&`).

Формат:

```
1 https://example.com/path?key1=value1&key2=value2
```

Копировать

- `key1=value1` — первый параметр.
- `key2=value2` — второй параметр.

Когда используется?

- Обычно применяется с методом **GET**, так как параметры передаются прямо в URL.
- Подходит для передачи небольших объемов данных, которые не требуют конфиденциальности.

Пример:

```
http
1 GET /search?query=HTTP+protocol&page=2 HTTP/1.1
2 Host: example.com
```

Копировать

Передача данных в HTTP-запросе

2. Тело запроса (Request Body)

Что это?

Тело запроса — это часть HTTP-запроса, которая находится после заголовков и содержит данные, отправляемые на сервер. Эти данные могут быть любой формы, например, JSON, XML, текст или файлы.

Когда используется?

- Применяется с методами **POST**, **PUT**, **PATCH** и другими, которые предполагают передачу данных на сервер.
- Используется для отправки больших объемов данных или данных, которые не должны быть видны в URL (например, пароли или файлы).

Форматы тела запроса

1. `application/x-www-form-urlencoded` :

- Аналог query string, но данные передаются в теле запроса.
- Формат: `ключ=значение&ключ=значение` .
- Пример:

```
http
1 POST /api/login HTTP/1.1
2 Host: example.com
3 Content-Type: application/x-www-form-urlencoded
4
5 username=john&password=secret
```

Форматы тела запроса

2. `application/json` :

- JSON-формат для передачи структурированных данных.
- Пример:

```
http
1  POST /api/users HTTP/1.1
2  Host: example.com
3  Content-Type: application/json
4
5  {
6    "name": "John",
7    "age": 30
8  }
```

Форматы тела запроса

3. multipart/form-data :

- Используется для отправки файлов и данных различных типов.
- Например, при загрузке изображений через форму.
- Пример:

```
http
1  POST /upload HTTP/1.1
2  Host: example.com
3  Content-Type: multipart/form-data; boundary=boundary
4
5  --boundary
6  Content-Disposition: form-data; name="file"; filename="image.jpg"
7  Content-Type: image/jpeg
8
9  [содержимое файла]
10 --boundary--
```

Query String + Request Body

В некоторых случаях можно использовать оба способа одновременно. Например:

```
http
1  POST /api/upload?category=images HTTP/1.1
2  Host: example.com
3  Content-Type: multipart/form-data; boundary=boundary
4
5  --boundary
6  Content-Disposition: form-data; name="file"; filename="image.jpg"
7  Content-Type: image/jpeg
8
9  [содержимое файла]
10 --boundary--
```

Здесь:

- `category=images` — параметр в query string, указывающий категорию файла.
- Файл передается в теле запроса.

HTTP заголовки

Заголовки в HTTP протоколе

Играют ключевую роль в управлении взаимодействием между клиентом и сервером. Позволяют настроить поведение запросов и ответов, определить формат данных, настроить безопасность и многое другое.

В HTTP/1.1 заголовки разделяются на несколько категорий:

- заголовки запроса
- заголовки ответа
- заголовки сущности
- заголовки соединения

Основные заголовки

КАТЕГОРИЯ	ЗАГОЛОВОК	ОПИСАНИЕ
Запрос	Host	Имя хоста.
	User-Agent	Информация о клиенте.
	Accept	Поддерживаемые форматы контента.
	Content-Type	Тип данных в теле запроса.
Ответ	Date	Дата и время ответа.
	Server	Информация о сервере.
	Location	URI для перенаправления.
Сущность	Content-Disposition	Как обрабатывать контент.
	ETag	Идентификатор версии ресурса.
	Cache-Control	Правила кэширования.
Соединение	Connection	Параметры соединения.
	Transfer-Encoding	Метод кодировки передачи.

Заголовки запроса

b) User-Agent

- **Назначение** : Информировать сервер о программном обеспечении клиента (браузер, операционная система и т.д.).
- **Пример** :

```
http Копировать  
1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
```

c) Accept

- **Назначение** : Указывает форматы контента, которые клиент может принять.
- **Пример** :

```
http Копировать  
1 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

d) Accept-Language

- **Назначение** : Указывает предпочтительные языки клиента.
- **Пример** :

```
http Копировать  
1 Accept-Language: en-US,en;q=0.5
```

Заголовки запроса

e) Accept-Encoding

- **Назначение** : Указывает методы сжатия, поддерживаемые клиентом.
- **Пример** :

```
http
1 Accept-Encoding: gzip, deflate, br
```

f) Content-Type

- **Назначение** : Указывает тип данных в теле запроса.
- **Пример** :

```
http
1 Content-Type: application/json
```

g) Content-Length

- **Назначение** : Указывает длину тела запроса в байтах.
- **Пример** :

```
http
1 Content-Length: 128
```

Заголовки запроса

h) Authorization

- **Назначение** : Используется для передачи учетных данных для аутентификации.
- **Пример** :

```
http
```

```
1 Authorization: Bearer <токен>
```

i) Cookie

- **Назначение** : Передает cookies от клиента к серверу.
- **Пример** :

```
http
```

```
1 Cookie: session_id=abc123; user_lang=en
```

Cookies в HTTP

Куки (Cookies) — это небольшие текстовые файлы, которые сервер отправляет клиенту (обычно браузеру), и которые затем хранятся на устройстве клиента.

Используются для сохранения информации о состоянии или данных между запросами HTTP, позволяя реализовать такие функции, как аутентификация, сессии, настройки пользователя.

Куки добавляют состояние в работу протокола HTTP, который сам по себе является **без состояния (stateless)**, то есть каждый запрос к серверу рассматривается как отдельное взаимодействие, не связанное с предыдущими запросами.

Cookies в HTTP

Основные параметры куки

ПАРАМЕТР	ОПИСАНИЕ
Name=Value	Имя и значение куки.
Expires/Max-Age	Время жизни куки (до какой даты или через сколько секунд она будет действовать).
Domain	Домен, для которого действует кука.
Path	Путь на сайте, для которого действует кука.
Secure	Кука будет отправляться только через HTTPS.
HttpOnly	Защита от доступа к куке через JavaScript.
SameSite	Ограничивает отправку куки с внешних сайтов (защита от CSRF).

Заголовки ответа

a) Date

- **Назначение** : Указывает дату и время создания ответа.
- **Пример** :

```
http
1 Date: Mon, 20 Mar 2023 14:12:00 GMT
```

b) Server

- **Назначение** : Информировать о программном обеспечении сервера.
- **Пример** :

```
http
1 Server: Apache/2.4.41 (Ubuntu)
```

c) Content-Type

- **Назначение** : Указывает тип контента в теле ответа.
- **Пример** :

```
http
1 Content-Type: text/html; charset=UTF-8
```

Заголовки ответа

d) Content-Length

- **Назначение** : Указывает длину тела ответа в байтах.
- **Пример** :

```
http
1 Content-Length: 256
```

e) Location

- **Назначение** : Используется для указания URI ресурса после перенаправления.
- **Пример** :

```
http
1 Location: https://example.com/new-page
```

f) Set-Cookie

- **Назначение** : Используется для установки cookie на клиенте.
- **Пример** :

```
http
1 Set-Cookie: session_id=abc123; HttpOnly; Secure
```

Заголовки сущности

a) Content-Disposition

- **Назначение** : Указывает, как обрабатывать контент (например, скачивание файла).
- **Пример** :

```
http
```

```
1 Content-Disposition: attachment; filename="document.pdf"
```

b) ETag

- **Назначение** : Уникальный идентификатор версии ресурса для проверки актуальности.
- **Пример** :

```
http
```

```
1 ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"
```

c) Last-Modified

- **Назначение** : Указывает дату последнего изменения ресурса.
- **Пример** :

```
http
```

```
1 Last-Modified: Tue, 15 Nov 2022 12:45:26 GMT
```

Заголовки сущности

d) Cache-Control

- **Назначение** : Управляет кэшированием ресурсов.
- **Пример** :

```
http
```

```
1 Cache-Control: max-age=3600, must-revalidate
```

e) Expires

- **Назначение** : Указывает дату истечения срока действия кэшированного ресурса.
- **Пример** :

```
http
```

```
1 Expires: Thu, 01 Dec 2023 16:00:00 GMT
```

Заголовки соединения

a) Connection

- **Назначение** : Указывает, какое поведение соединения использовать.
- **Пример** :

```
http
1 Connection: keep-alive
```

b) Transfer-Encoding

- **Назначение** : Указывает метод кодировки передачи данных.
- **Пример** :

```
http
1 Transfer-Encoding: chunked
```

Инструменты для работы с HTTP

Инструменты для работы с HTTP-запросами

- Браузер
 - Вкладка Network в инструментах разработчика
- Консольные утилиты
 - `curl`, `wget`
 - `Invoke-WebRequest`, `Invoke-RestMethod` (PowerShell)
- Утилиты, встроенные в IDE
- Графические утилиты или веб-интерфейс
 - Postman

curl и wget

curl

1. Загрузка файла:

```
bash
```

```
1 curl -O https://example.com/file.zip
```

2. Отправка POST-запроса:

```
bash
```

```
1 curl -X POST -d "name=John&age=30" https://example.com/api
```

3. Отправка кастомных заголовков:

```
bash
```

```
1 curl -H "Authorization: Bearer token" https://example.com/secure
```

wget

1. Загрузка файла:

```
bash
```

```
1 wget https://example.com/file.zip
```

2. Рекурсивное скачивание сайта:

```
bash
```

```
1 wget --mirror https://example.com
```

3. Продолжение прерванной загрузки:

```
bash
```

```
1 wget -c https://example.com/largefile.iso
```

curl и wget

Когда использовать `curl` ?

- Когда вам нужно отправить сложные запросы (например, POST, PUT, DELETE).
- При работе с API.
- Если требуется полный контроль над заголовками и параметрами запроса.
- Для тестирования сетевых соединений.

Когда использовать `wget` ?

- Когда нужно просто скачать файл или сайт.
- Для рекурсивного скачивания содержимого.
- При необходимости продолжить прерванную загрузку.
- Для автоматизированного скачивания файлов без сложных манипуляций.

Invoke-WebRequest и Invoke-RestMethod

Invoke-WebRequest

1. Получение содержимого веб-страницы:

powershell

Копировать

```
1 $response = Invoke-WebRequest -Uri "https://example.com"
2 $response.Content # Выводит HTML-код страницы
```

2. Отправка POST-запроса:

powershell

Копировать

```
1 $body = @{name="John"; age=30}
2 $response = Invoke-WebRequest -Uri "https://example.com/api" -Method Post -Body $body
```

3. Парсинг HTML:

powershell

Копировать

```
1 $response = Invoke-WebRequest -Uri "https://example.com"
2 $response.ParsedHtml.title # Выводит заголовок страницы
```

Invoke-WebRequest и Invoke-RestMethod

Invoke-RestMethod

1. Получение данных из REST API:

powershell

Копировать

```
1 $data = Invoke-RestMethod -Uri "https://api.example.com/users"  
2 $data | Format-Table # Выводит список пользователей в виде таблицы
```

2. Отправка POST-запроса с JSON-данными:

powershell

Копировать

```
1 $body = @{  
2     name = "John"  
3     age = 30  
4 } | ConvertTo-Json  
5 $response = Invoke-RestMethod -Uri "https://api.example.com/create" -Method Post -Body $body
```

3. Авторизация через Bearer Token:

powershell

Копировать

```
1 $headers = @{  
2     Authorization = "Bearer your_token_here"  
3 }  
4 $data = Invoke-RestMethod -Uri "https://api.example.com/secure" -Headers $headers
```

Invoke-WebRequest и Invoke-RestMethod

Когда использовать `Invoke-WebRequest` ?

- Когда вам нужно работать с веб-страницами, HTML-кодом или XML-документами.
- Если требуется получить доступ к дополнительной информации о запросе/ответе (например, заголовкам или статус-коду).
- Для парсинга содержимого веб-страниц или скачивания файлов.

Когда использовать `Invoke-RestMethod` ?

- Когда ваша цель — работа с RESTful API.
- Если вы ожидаете ответ в формате JSON и хотите автоматически преобразовать его в объекты PowerShell.
- Для быстрой обработки данных без необходимости ручного парсинга.

Протокол CGI

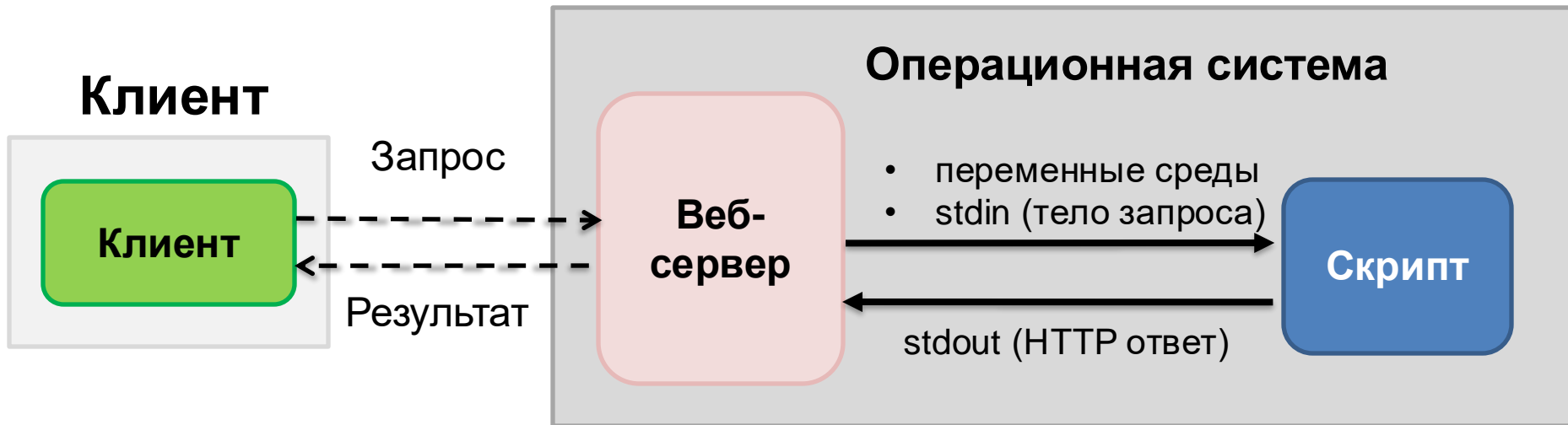
Протокол CGI

CGI (Common Gateway Interface) — стандарт, позволяющий веб-серверу передавать входящие запросы на обработку внешним скриптам, а затем возвращать результаты обратно клиенту.

- Обеспечивает динамическую генерацию веб-страниц и интерактивное взаимодействие с пользователем.
- Разработан в 1993 году сотрудниками NCSA (National Center for Supercomputing Applications), которые также создали один из первых популярных веб-серверов — NCSA HTTPd.

Базовая спецификация CGI

- **Механизм запроса.** Веб-сервер идентифицирует CGI-скрипты либо по специально настроенному каталогу (`/cgi-bin/`), либо по расширению файла (`.cgi`, `.pl`, ...).
- **Передача данных.** Данные из HTTP-запроса сервер передает скрипту через переменные среды и поток `stdin`. Вывод скрипта в `stdout` направляется веб-серверу, который отправляет его клиенту в качестве HTTP-ответа.



Переменные окружения:

- `QUERY_STRING`: строка запроса, переданная в URL после знака ?
- `REQUEST_METHOD`: метод HTTP запроса (GET, POST, ...)
- `CONTENT_TYPE` и `CONTENT_LENGTH`
- `SCRIPT_NAME` и `PATH_INFO`

Базовая спецификация CGI

Преимущество CGI - простота. Веб-сервер запускает внешнюю программу и отдает клиенту результат как есть, при этом не важно на каком языке программирования написан CGI скрипт.

Проблемы и ограничения

- **Медленная работа.** Каждый запрос требует запуска нового процесса
- **Безопасность.** Неправильно написанные скрипты могли стать уязвимыми для атак, таких как инъекции команд или переполнение буфера.
- **Сложность масштабирования.** При большом количестве одновременных запросов производительность CGI значительно снижается.