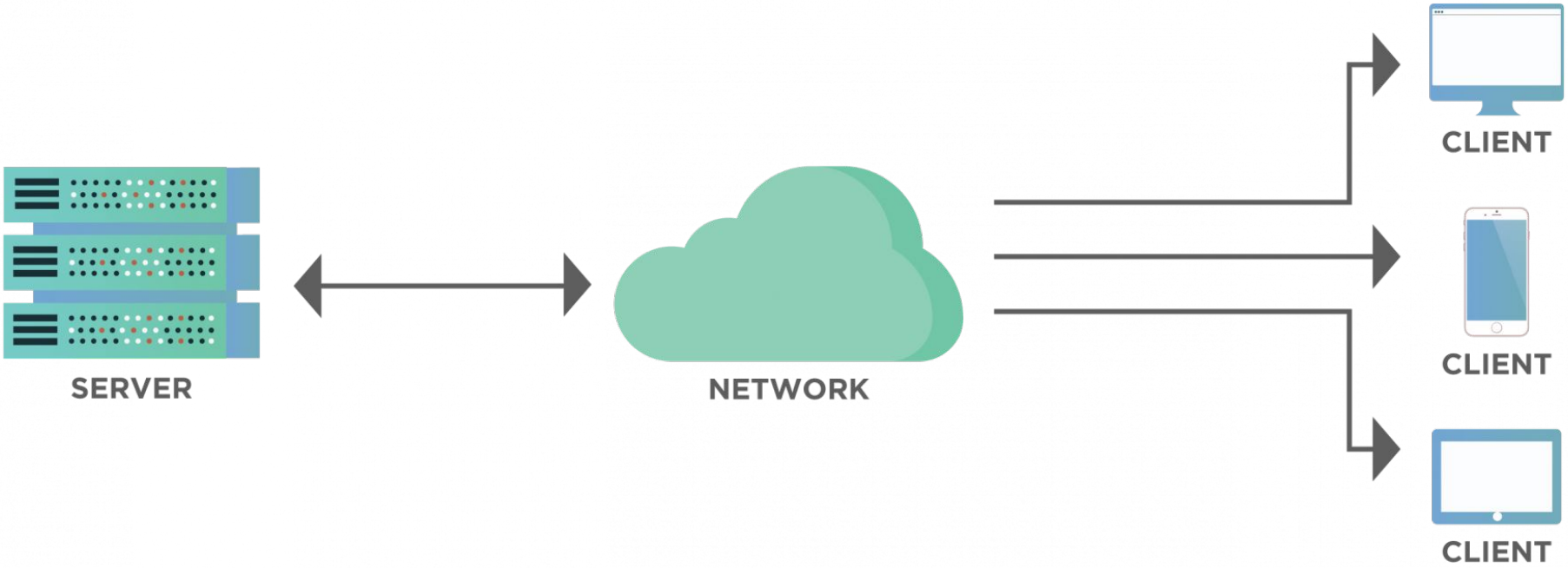
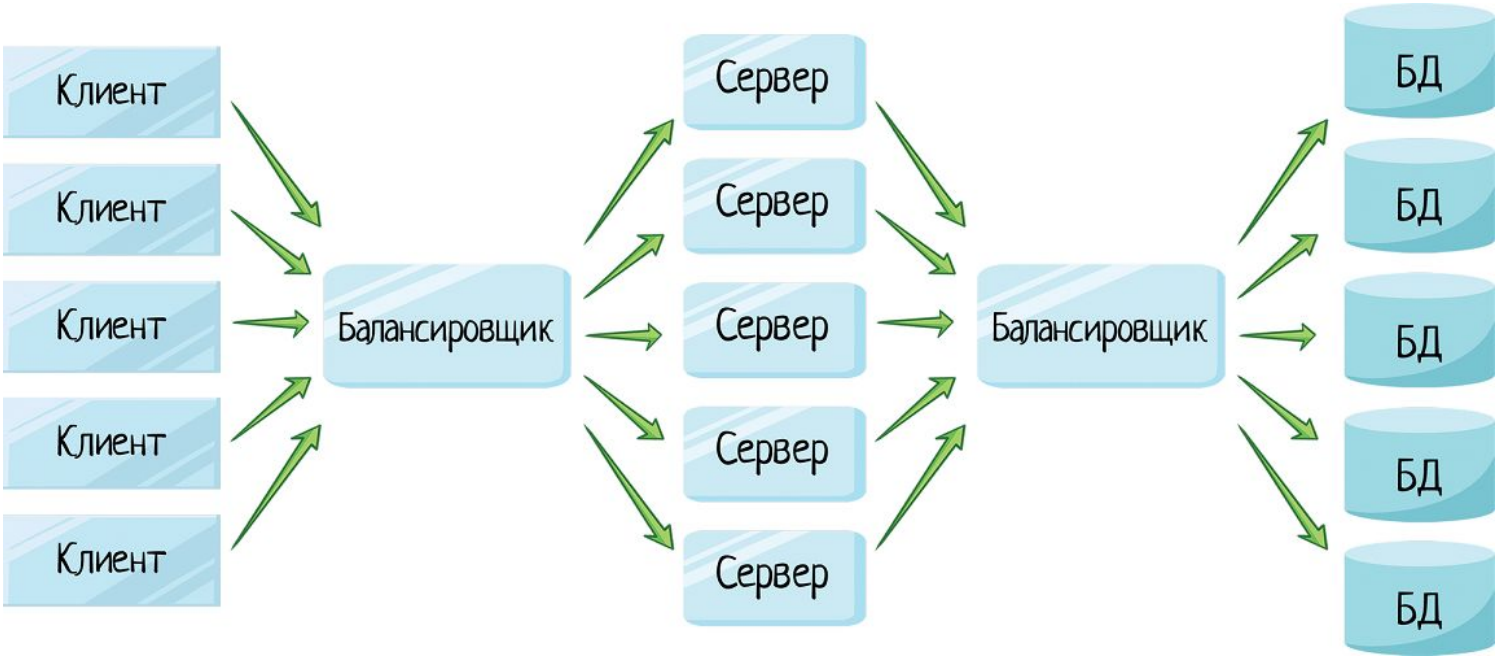
# Основы разработки веб-приложений

# Архитектура клиент-сервер



# Архитектура клиент-сервер



# Как работает Web?

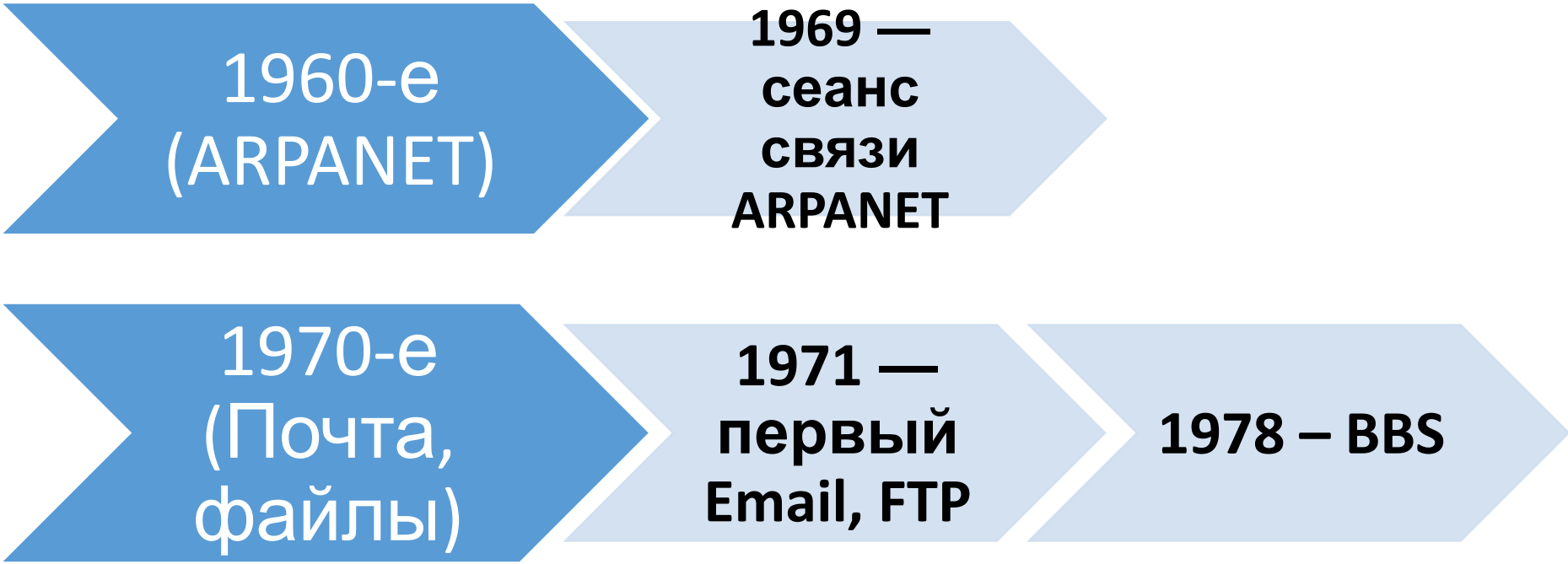
# А почему?

# Классический web-разработчик

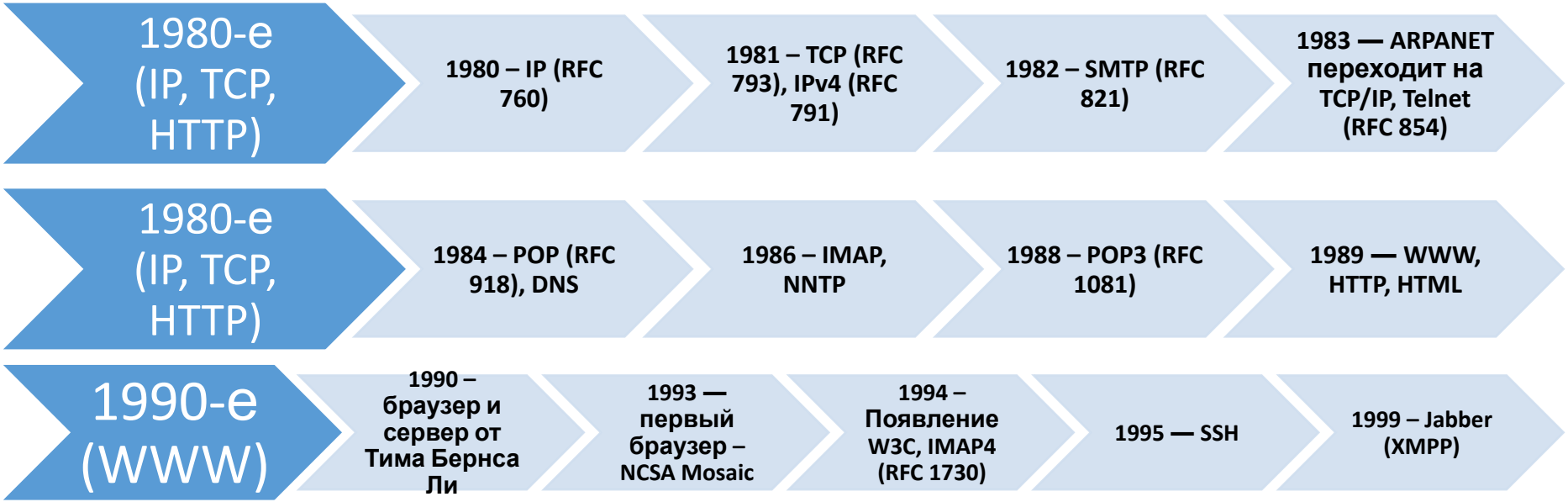
# Web-технологии
# Так почему же?

Так сложилось.

# Интернет

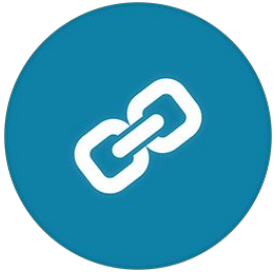

1960-е (ARPANET)
1969 — сеанс связи ARPANET
1970-е (Почта, файлы)
1971 — первый Email, FTP
1978 – BBS

# Интернет

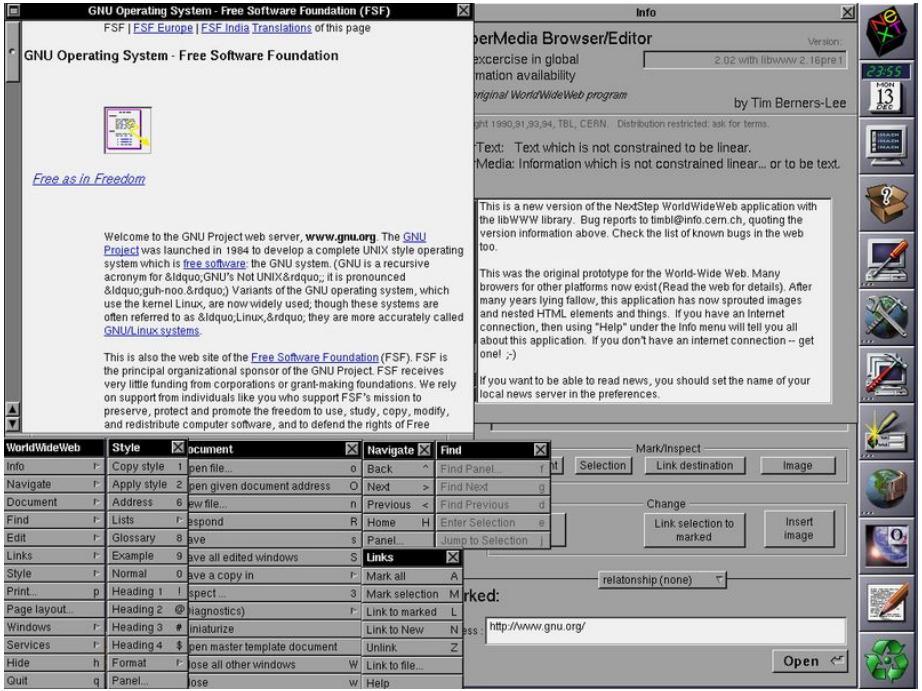

1980-е (IP, TCP, HTTP)
1980 – IP (RFC 760)
1981 – TCP (RFC 793), IPv4 (RFC 791)
1982 – SMTP (RFC 821)
1983 — ARPANET переходит на TCP/IP, Telnet (RFC 854)
1980-е (IP, TCP, HTTP)
1984 – POP (RFC 918), DNS
1986 – IMAP, NNTP
1988 – POP3 (RFC 1081)
1989 — WWW, HTTP, HTML
1990-е (WWW)
1990 – браузер и сервер от Тима Бернса Ли
1993 — первый браузер – NCSA Mosaic
1994 – Появление W3C, IMAP4 (RFC 1730)
1995 — SSH
1999 – Jabber (XMPP)

# Всемирная сеть

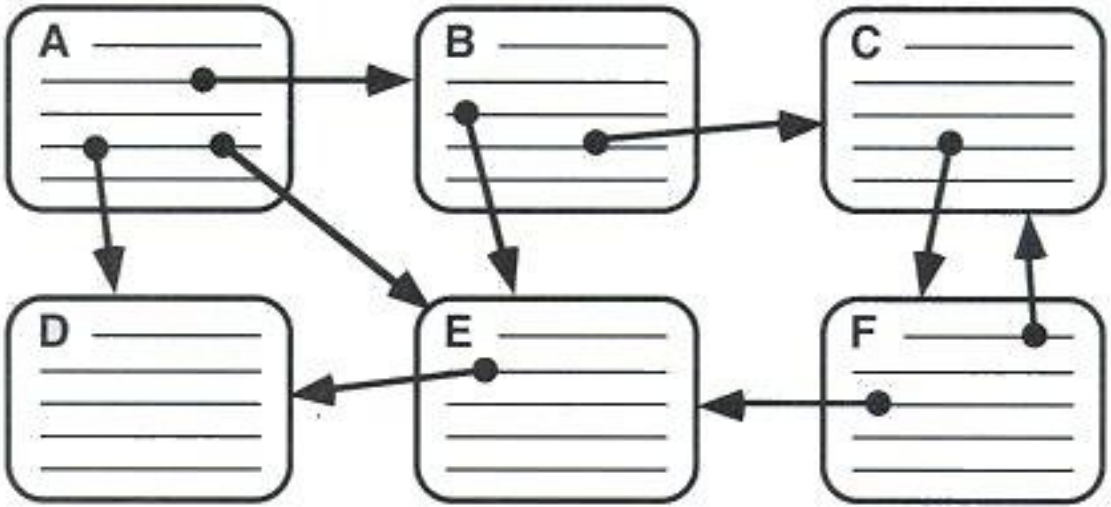


# WorldWideWeb



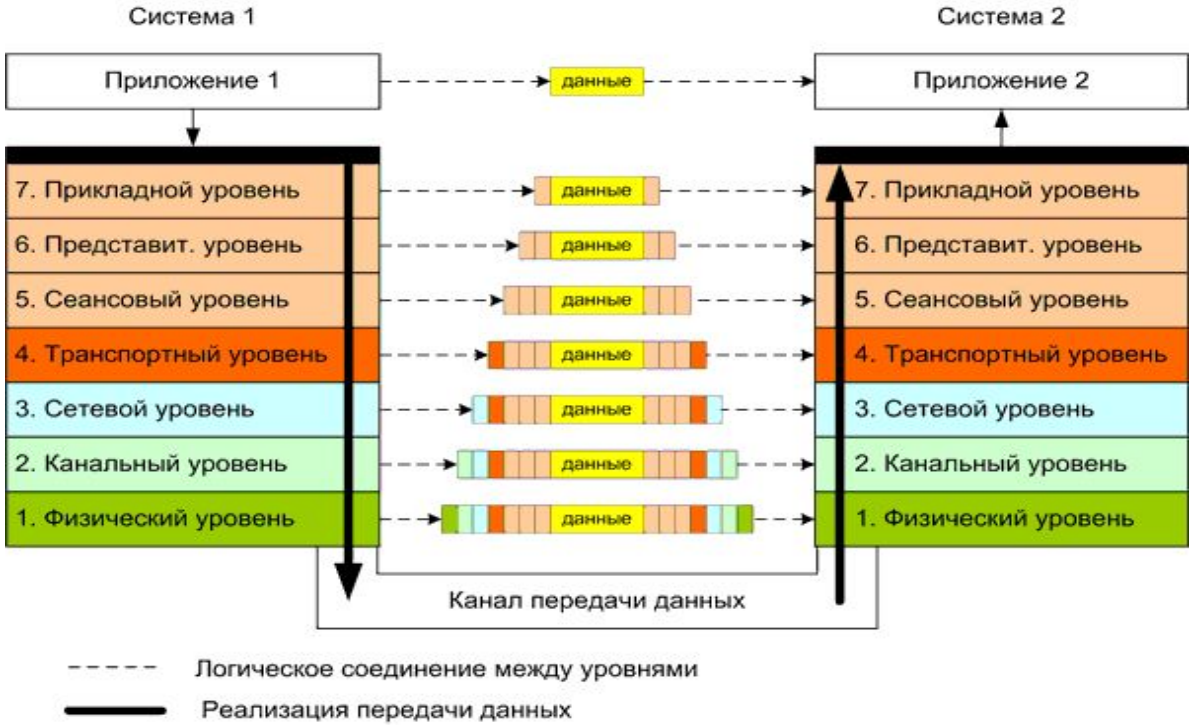
# Гипертекст



# Сеть интернет

# Базовые технологии WEB

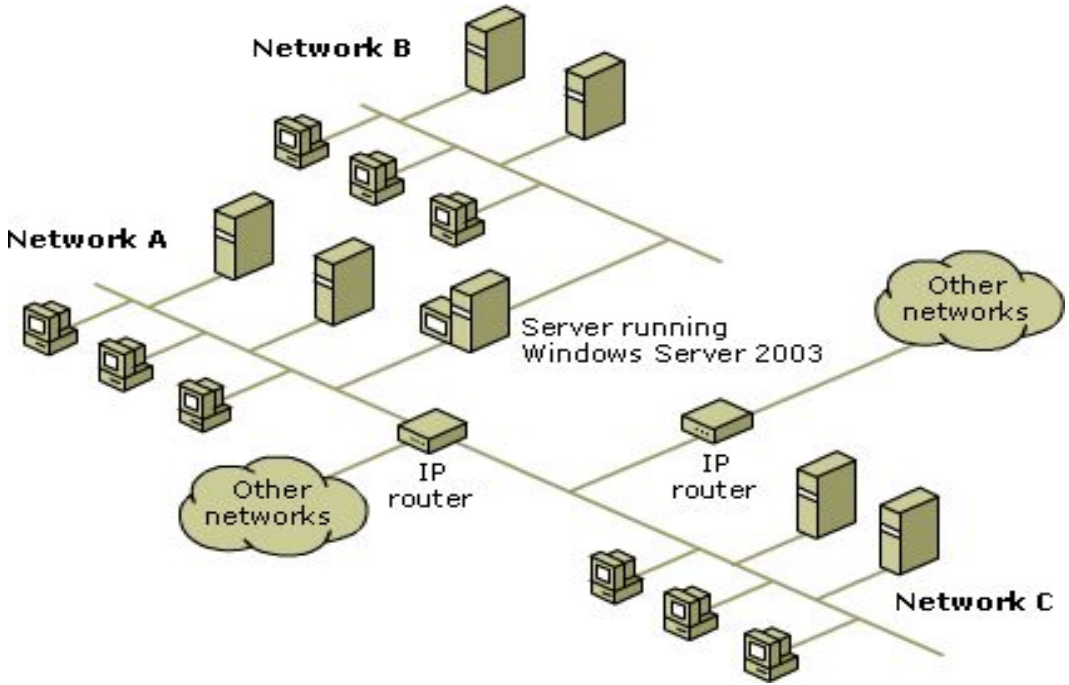
# OSI ISO


Система 1
Система 2
Приложение 1
данные
Приложение 2
7. Прикладной уровень
6. Представит. уровень
5. Сеансовый уровень
4. Транспортный уровень
3. Сетевой уровень
2. Канальный уровень
1. Физический уровень
Канал передачи данных
Логическое соединение между уровнями
Реализация передачи данных

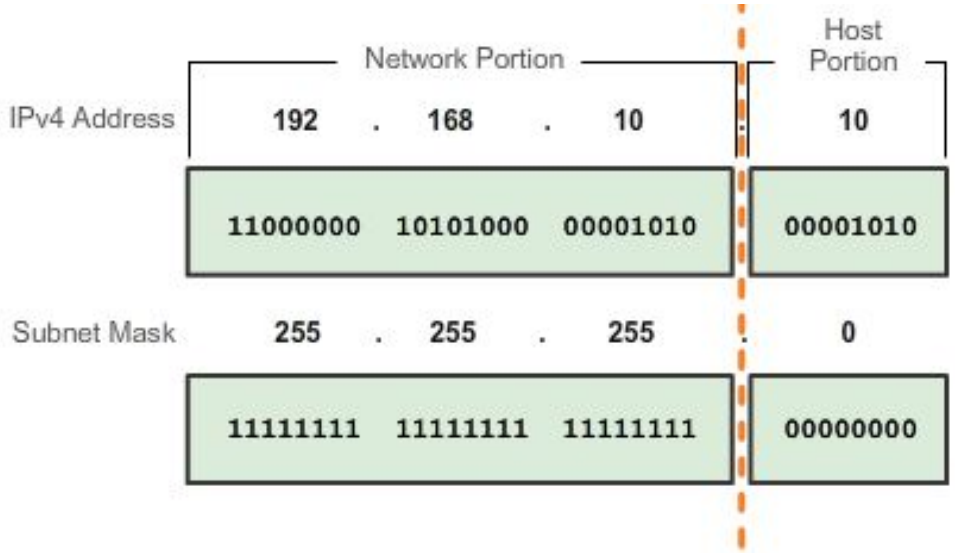
# IP: Internet Protocol

- Глобальная адресация
- Передача в гетерогенной сети (сегментация)
- Маршрутизация пакетов

# IP: Internet Protocol




# IP: Internet Protocol




# IP: Internet Protocol

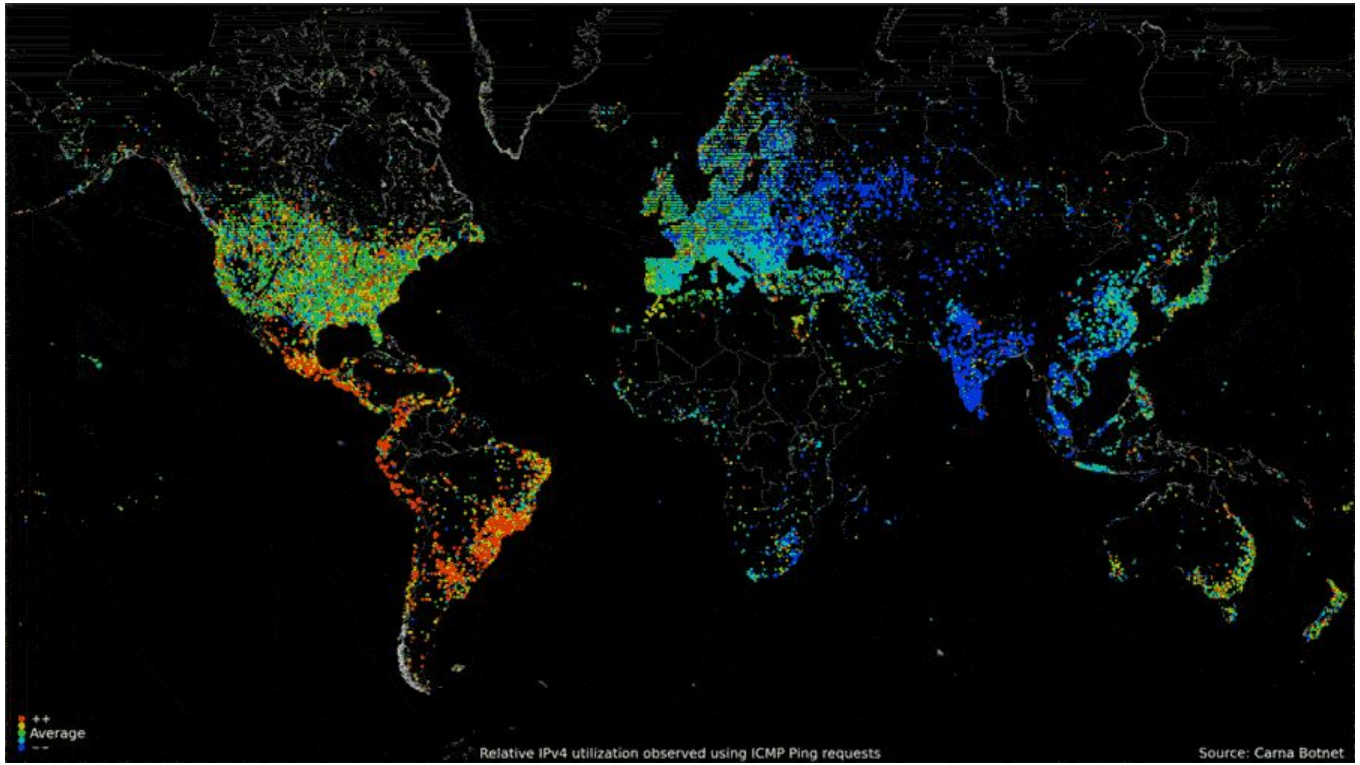
localhost == 127.0.0.1

# IP: Адресное пространство ipv4 vs ipv6

$2^{32}=4294967295$

$2^{128}=$ 340 282 366 920 938 463 463 374 607 431 768 211 456

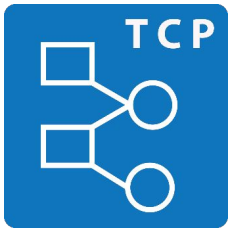
# IP: Суточная активность




IP

# TCP

- Адресация приложения в пределах хоста с помощью портов
- Последовательное двустороннее соединение
- Надежная доставка
- Управление потоком



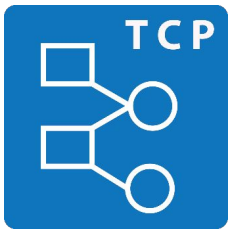
# TCP

## Порты – адресация приложения на хосте

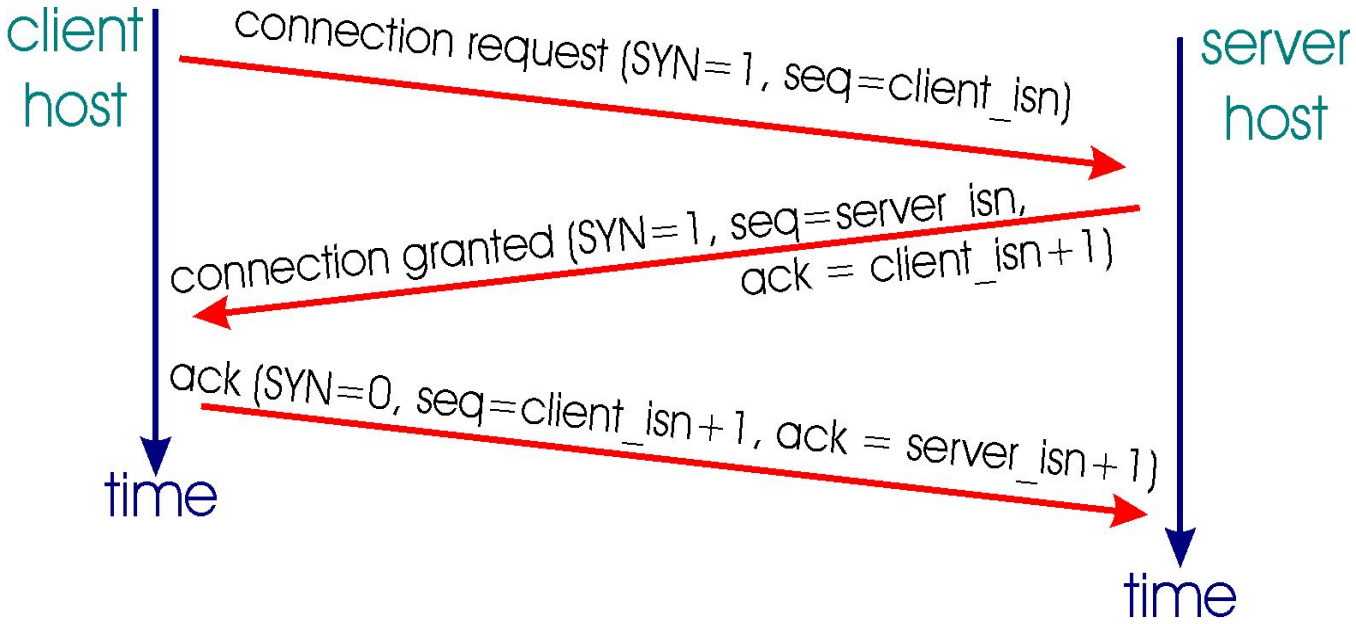
- Well-known: SSH=22, FTP=20,21, HTTP=80, SMTP=25, POP3=110
- Привилегированные (<1024)
- Остальные (>=1024)

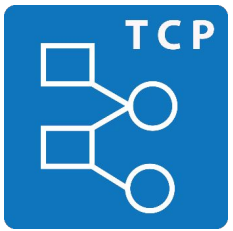
## Сокеты (sockets) – пара адрес-порт

- Серверные (bind, listen, accept)
- Клиентские (connect, send, recv)

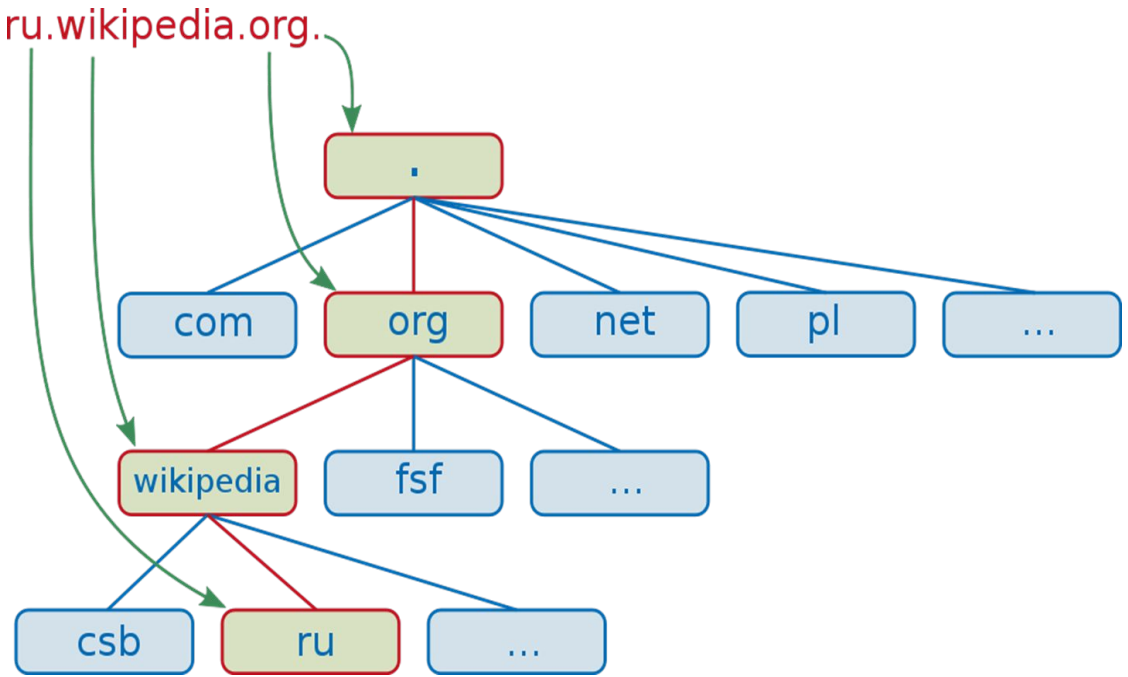


# TCP

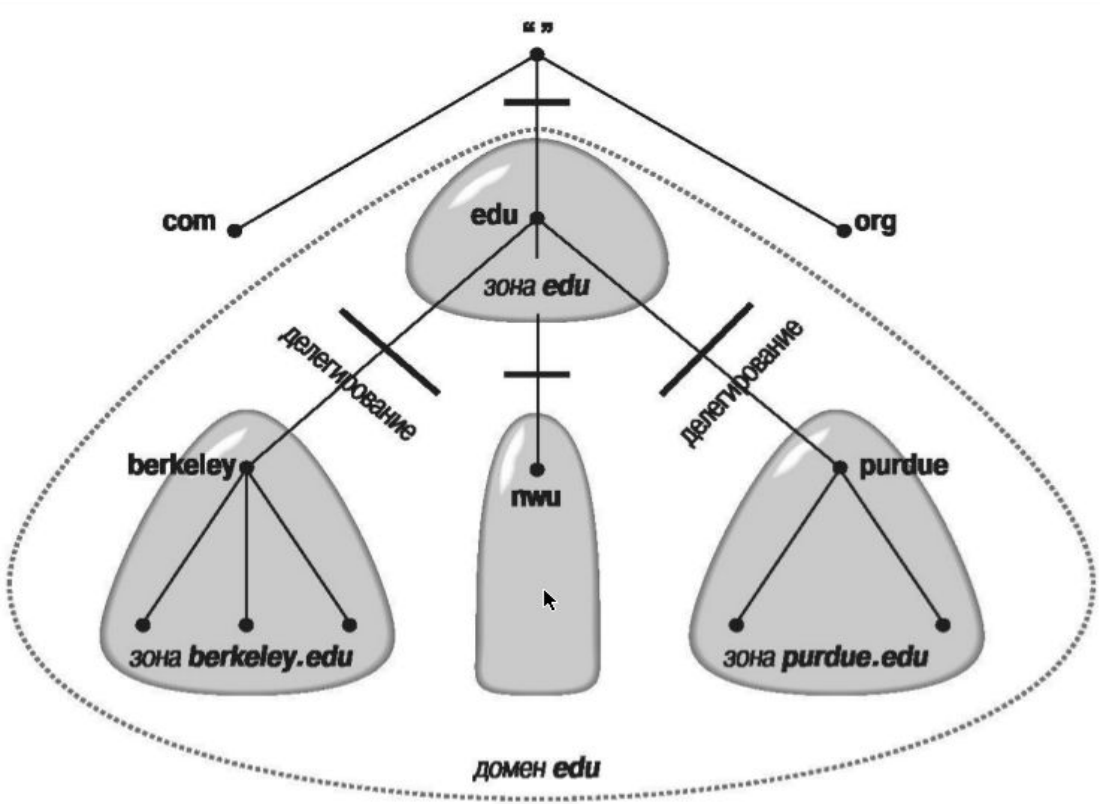

client
host
server
host
connection request (SYN=1, seq=client_isn)
connection granted (SYN=1, seq=server_isn,
ack = client_isn+1)
ack (SYN=0, seq=client_isn+1, ack = server_isn+1)
time
time


TCP

# DNS

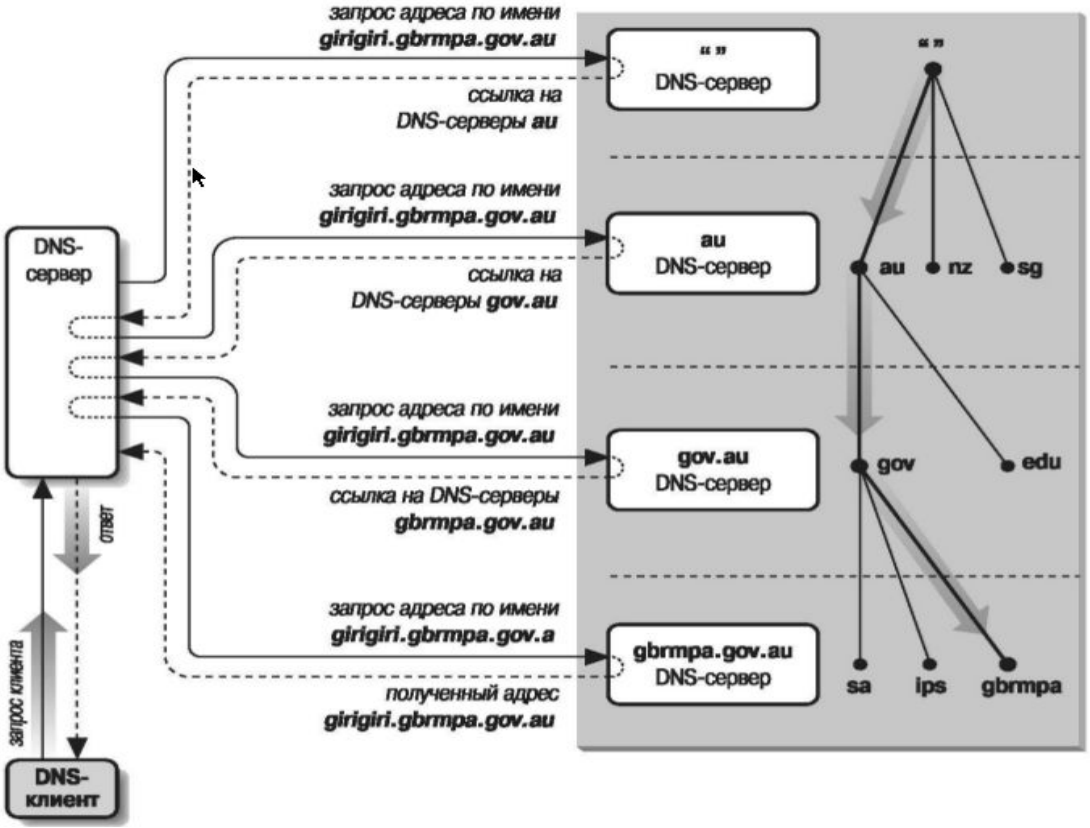



# DNS


" "
com
edu
org
зона edu
делегирование
делегирование
berkeley
nwu
purdue
зона berkeley.edu
зона purdue.edu
домен edu

DNS

# DNS


запрос адреса по имени
girigiri.gbrmpa.gov.au
“ ”
DNS-сервер
ссылка на
DNS-серверы au
запрос адреса по имени
girigiri.gbrmpa.gov.au
au
DNS-сервер
ссылка на
DNS-серверы gov.au
DNS-
сервер
запрос адреса по имени
girigiri.gbrmpa.gov.au
gov.au
DNS-сервер
ссылка на DNS-серверы
gbrmpa.gov.au
ответ
запрос адреса по имени
girigiri.gbrmpa.gov.a
gbrmpa.gov.au
DNS-сервер
полученный адрес
girigiri.gbrmpa.gov.au
запрос клиента
DNS-
клиент
“ ”
au
nz
sg
gov
edu
sa
ips
gbrmpa

DNS

# URI == URL || URN || URL + URN

**scheme://host:[port]/path/.../[;url-params][?query-string][#anchor]**

- scheme – протокол (http,https,ftp)
- host – ip-адрес или доменное имя
- port – порт, необязательно если стандартный
- path – путь в файловой системе корневого каталога сервера (или псевдоним, обрабатываемый сервером)
- url-params – необязательные пары ключ-значение. Используется для идентификаторов сессии пользователя.
- query-string – пары ключ-значение – параметры запроса
- anchor – ссылка на позиционный маркер в пределах документа

# HTTP: HyperText Transfer Protocol

Текстовый протокол без сохранения состояния

http://

# HTTP: Сессия

- Клиент устанавливает TCP-соединение с сервером (обычно, на 80 порт)
- Сервер принимает запрос на соединение и ожидает текст HTTP запроса.
- Клиент отправляет запрос со всей необходимой информацией (URI, тип запроса, заголовки, тело запроса).
- Сервер обрабатывает запрос и отдает ответ: код статуса, заголовки ответа и тело ответа.

http://

# HTTP: Структура

**Структура HTTP-запроса**

- Строка запроса.
- Заголовки.
- Пустая строка.
- Тело запроса.

**Структура HTTP-ответа**

- Строка статуса ответа.
- Заголовки ответа.
- Пустая строка.
- Тело ответа.

http://

# HTTP: Методы

- **OPTIONS** — запрос методов сервера (Allow)
- **GET** — запрос документа (Условный GET)
- **HEAD** — аналог GET, но без тела ответа
- **POST** — передача данных от клиента
- **PUT** — размещение файла по URI/изменение данных
- **DELETE** — удаление файла по URI/удаление данных
- **TRACE, LINK, UNLINK, CONNECT** — редко

http://

# HTTP: Коды ответов

- **1xx — Информационные**
- **2xx — Успешное выполнение**
  - 200 — OK
  - 204 — NoContent (только заголовки)
  - 206 — PartitialContent (часть ответа)
- **3xx — Перенаправления**
  - 301 — Moved Pemanently (SEO, кеширование)
  - 302 — Found (логика работы сайта)
  - 304 — Not Modified (при условном GET)
- **4xx — Ошибка клиента**
  - 400 — Bad Request (размер, формат..)
  - 401 — Unauthorized (запрос авторизации)
  - 403 — Forbidden (allow, deny)
  - 404 — Not Found
  - 408 — Request Timeout (на чтение)
  - 418 — I'm teapot
- **5xx — Ошибка сервера**
  - 500 — Internal Server Error
  - 502 — Bad Gateway (проксирование)
  - 503 — Service Unavailable
  - 504 — Gateway Timeout
  - 505 — HTTP version not supported
  - 507 — Insufficient Storage

# HTTP: Заголовки

- **Host** — указание домена, вирт. Хостинг
- **User-Agent** — описание клиента
- **Accept-*** — поддержка MIME типов, кодировок, языков и т.п.
- **Cookie** — куки для данной страницы
- **Referer** — текущая страница
- **If-Modified-Since** — условный GET
- **Connection** — управление соединением
- **Content-Type** — MIME тип документа
- **Content-Length** — размер документа
- **Content-Encoding** — кодирование документа
- **Date** — текущее время сервера
- **Expires** — время актуальности документа
- **Last-Modified** — время изменения файла
- **Set-Cookie** — установка кук для данного URI
- **Connection** — управление соединением

# HTTP REST

| Resource | GET | PUT | POST | DELETE |
|---|---|---|---|---|
| **Collection Uri, such as**<br><br>http://example.com<br>/resources | **List** the URIs and perhaps other details of the collection's members. | **Replace** the entire collection with another collection. | **Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation. | **Delete** the entire collection. |
| **Element Uri, such as**<br><br>http://example.com<br>/resources/item17 | **Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type. | **Replace** the addressed member of the collection, or if it doesn't exist, **create** it. | Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it. | **Delete** the addressed member of the collection. |

http://

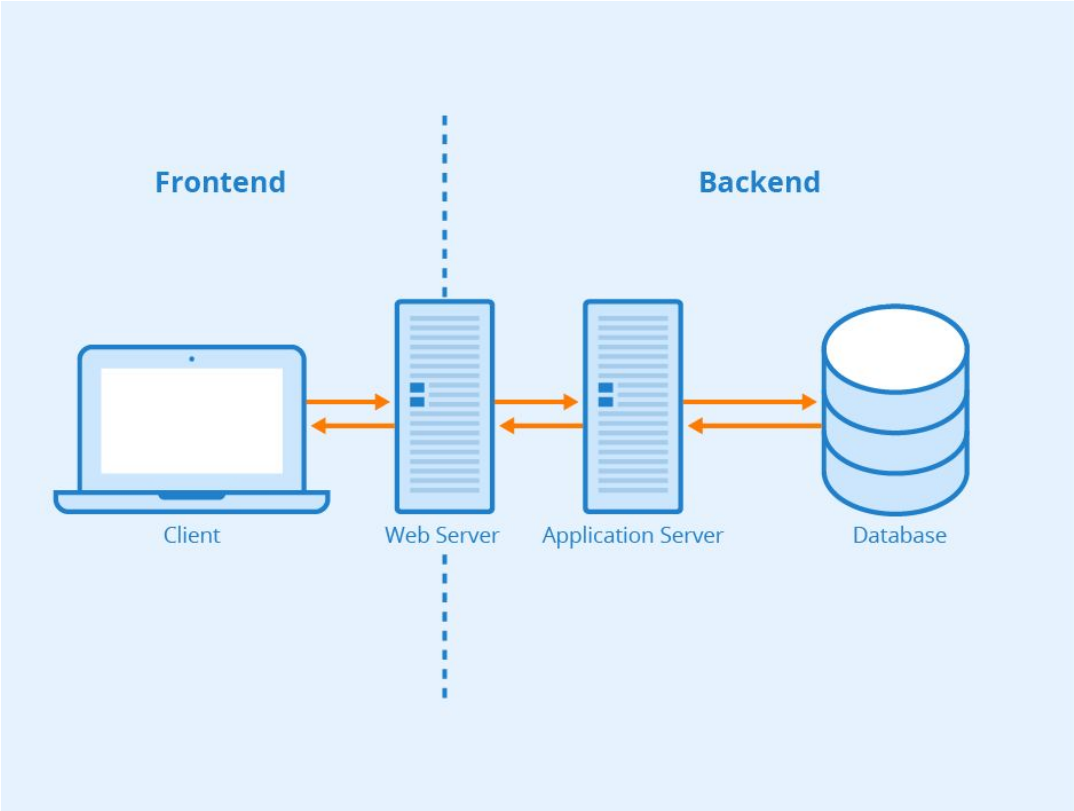
# HTTPS

# Другие протоколы

- GRPC
- GraphQL
- web-socket
- etc.

# Веб-приложение

# Frontend & Backend

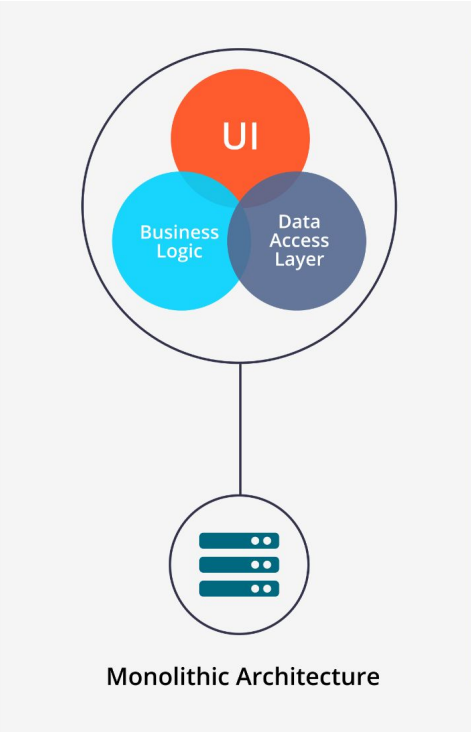


# Высокоуровневая архитектура клиента (MPA vs SPA)

# Монолит

+ Простота развертывания
+ Простота разработки
+ Отсутствие накладных расходов на взаимодействие
+ Простота запуска и дебага

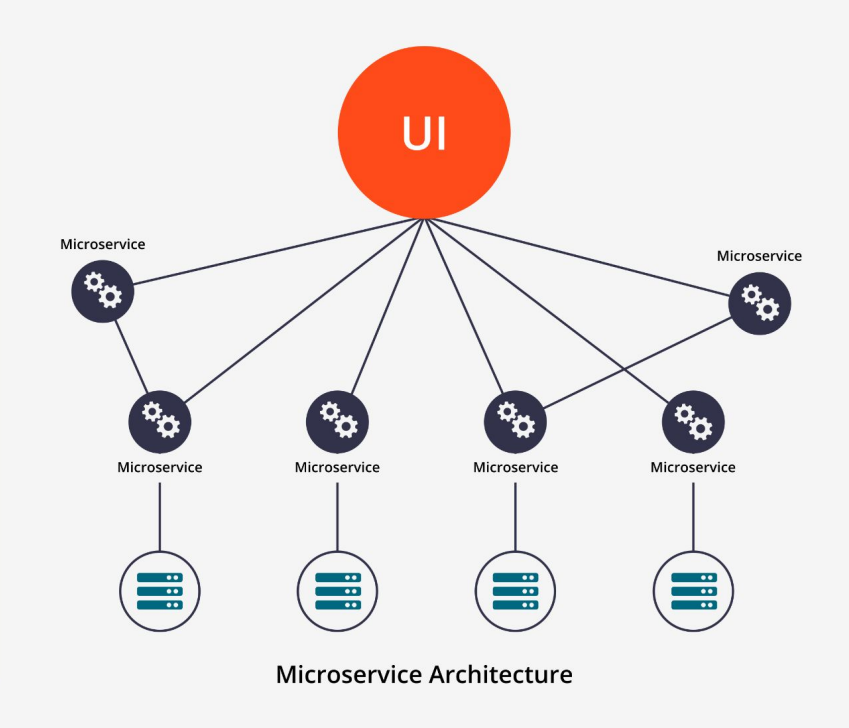
- Скорость разработки (на большом масштабе)
- Масштабируемость
- Доступность (если падает, то все)
- Любое изменение = редеплой всего монолита
- Сложнее внедрять новые технологии



# Микросервисы

+ Масштабируемость
+ Раздельный деплой
+ Доступность (частичная)
+ Скорость разработки (на большом масштабе)

- Сложность развертывания
- Сложность разработки
  - Синхронизация данных
  - Организация взаимодействия сервисов
  - Сложность отладки
- Накладные расходы на взаимодействие



# Микросервисы

- У каждого МКС - своя БД => нет простой транзакционности
- Взаимодействие может быть:
  - синхронным
  - асинхронным
- Всегда нужно помнить, что сеть и железо ненадежны
- При работе с микросервисами мониторинг и dev-ops критически важны