Focused forward

Netcracker

Лекция 2.

Клиент-серверная архитектура ПО. Протоколы передачи данных

Web System Design and Development

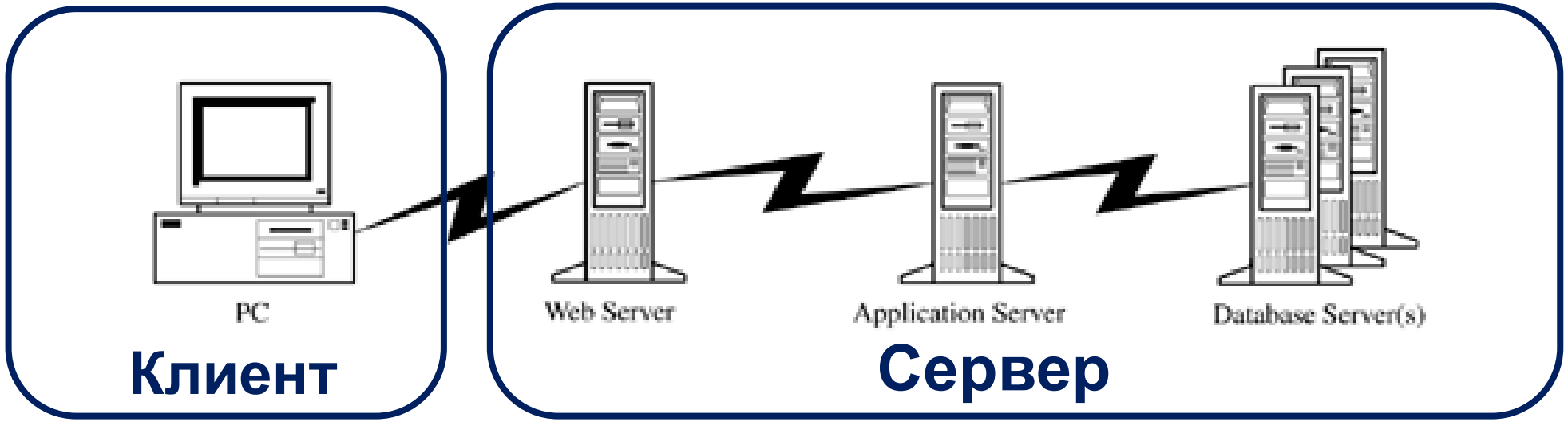
План лекции

- Клиент-серверная архитектура
- Клиент-серверные технологии
- Модель обмена информацией OSI
- Протоколы прикладного уровня: HTTP и HTTPS
- История развития протокола HTTP
- Почтовые протоколы: POP3/IMAP, SMTP
- Другие протоколы прикладного уровня: FTP, SSH, SFTP, RDP
- Основные протоколы транспортного уровня: TCP, UDP

Клиент-серверная архитектура

- **Архитектура клиент-сервер (Client-server)** определяет общие принципы организации взаимодействия в сети, где имеются ***серверы***, узлы-поставщики некоторых специфичных функций (сервисов) и ***клиенты***, потребители этих функций.

- **Клиентом (front end)** является запрашивающая машина (обычно ПК), **сервером (back end)** — машина, которая отвечает на запрос.



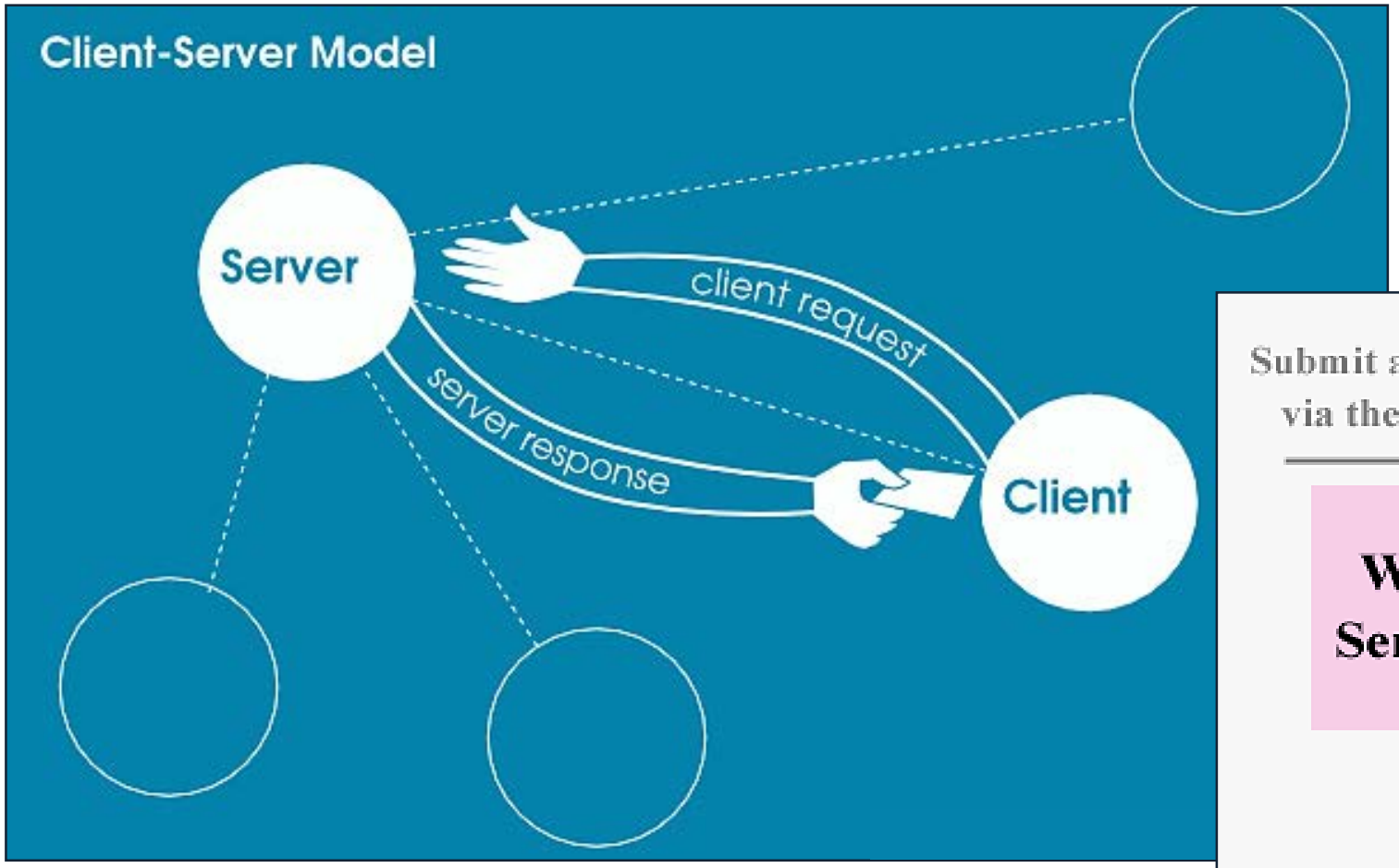
Клиент-серверная архитектура

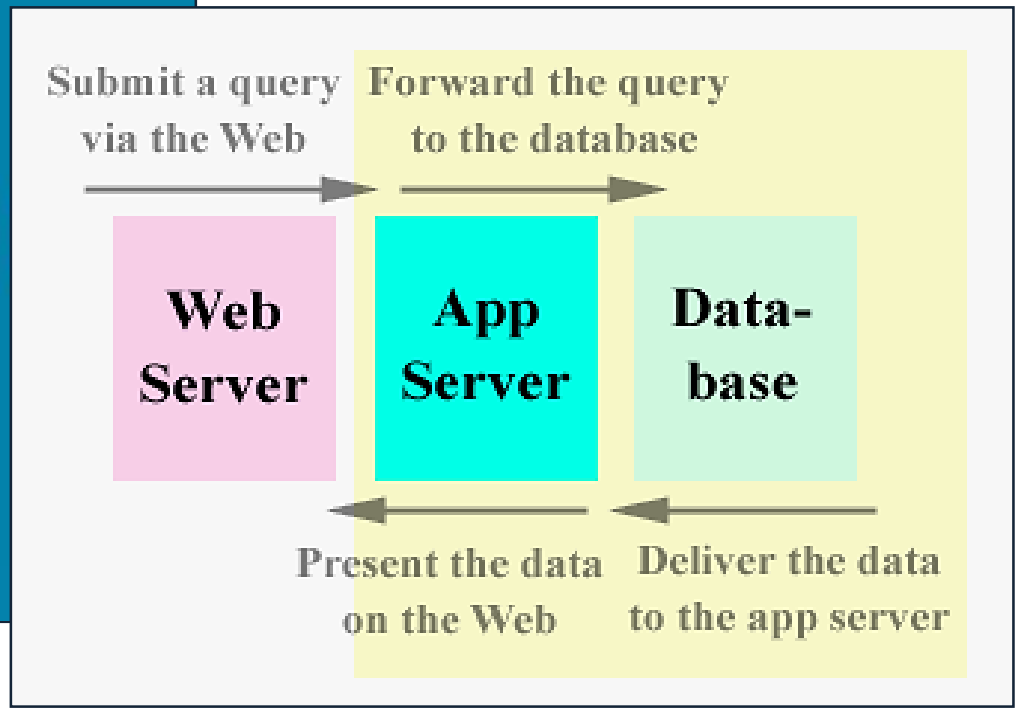
- **Веб-сервер (Web Server)** — это сервер, **принимающий HTTP-запросы от клиентов**, обычно веб-браузеров, и **выдающий им HTTP-ответы**, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. Обеспечивает взаимодействие клиента с сервером.

- **Сервер приложений (Application Server)** — это сервисная программа, которая обеспечивает доступ клиентов к прикладным программам, выполняющимся на сервере. Реализует бизнес-логику и позволяет вызывать на исполнение процедуры и функции ПО.

- **Сервер баз данных (Database Server)** выполняет обслуживание и управление базой **данных** и отвечает за целостность и сохранность **данных**, а также обеспечивает операции ввода-вывода при доступе к информации.

Клиент-серверная архитектура





Клиент-серверные технологии

Клиентские технологии

HTML, XHTML

CSS

ActiveX

JavaScript

Silverlight

Adobe Flash, Adobe Flex

Java

Серверные технологии

ASP.NET

SQL

PHP

Java

VBScript

© W3Techs.com	usage	change since 1 January 2017
1. PHP	82.5%	+0.1%
2. ASP.NET	15.3%	
3. Java	2.7%	
4. static files	1.5%	
5. ColdFusion	0.6%	

percentages of sites

Клиент-серверные технологии

Популярность клиентских технологий:

© W3Techs.com	usage	change since 1 January 2017
1. JavaScript	94.4%	
2. Flash	7.2%	-0.1%
3. Silverlight	0.1%	

percentages of sites

Популярность кодировок:

© W3Techs.com	usage	change since 1 January 2017
1. UTF-8	88.5%	+0.3%
2. ISO-8859-1	5.3%	-0.2%
3. Windows-1251	1.7%	
4. Shift JIS	1.0%	
5. Windows-1252	0.8%	-0.1%

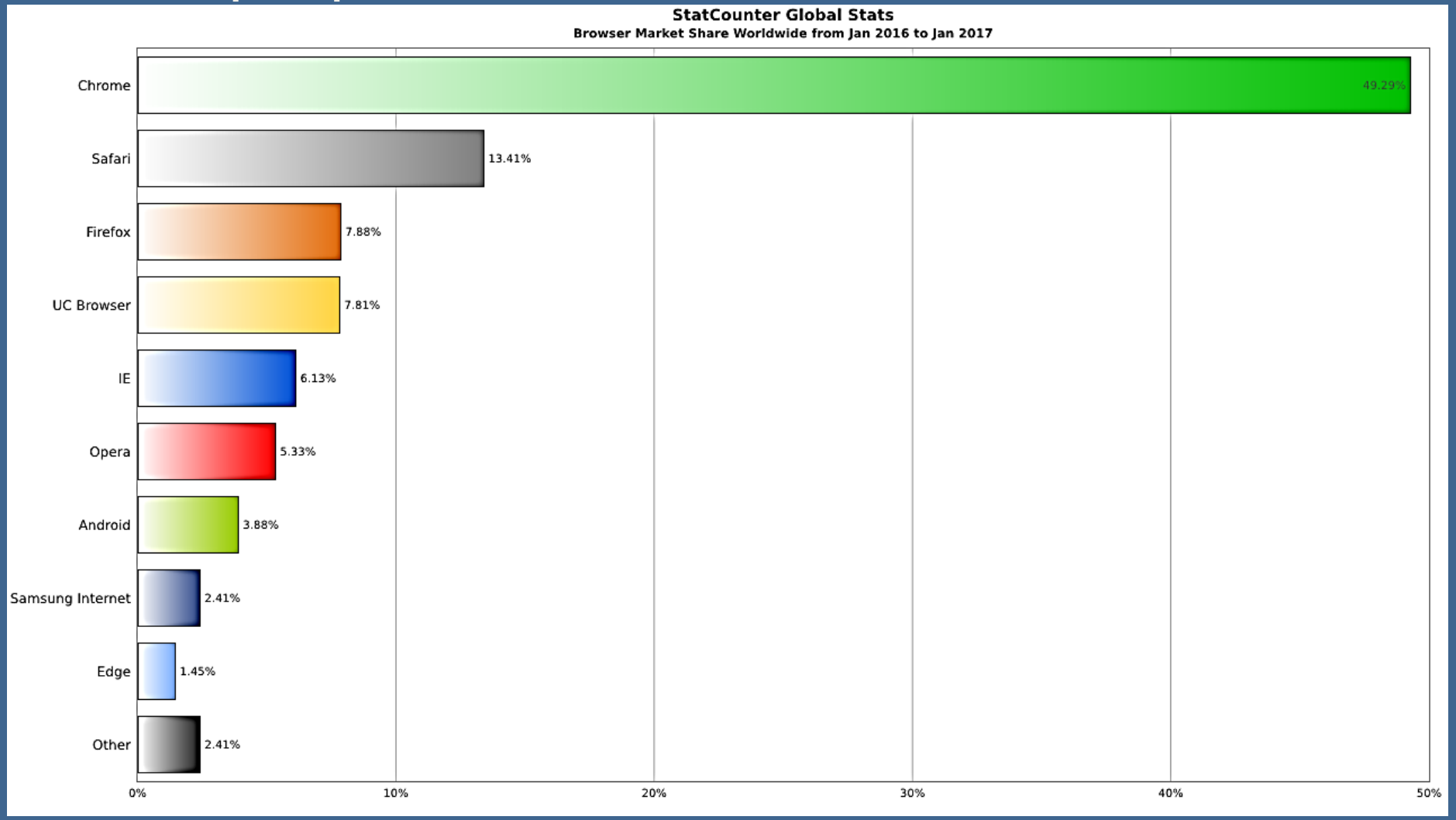
percentages of sites

Популярность библиотек JavaScript:

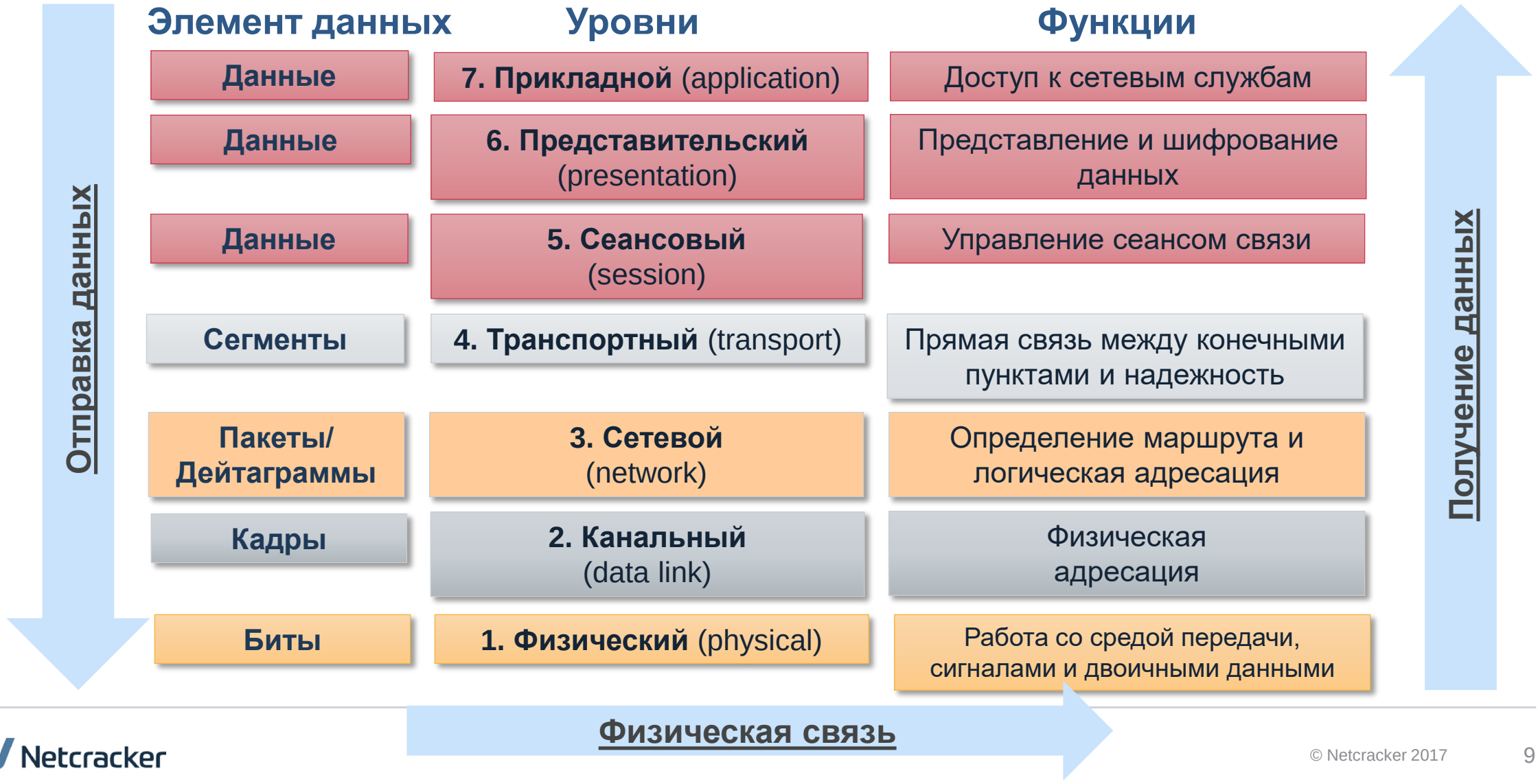
© W3Techs.com	usage	change since 1 January 2017	market share	change since 1 January 2017
1. jQuery	72.0%	+0.1%	96.5%	+0.1%
2. Bootstrap	14.4%	+0.2%	19.3%	+0.3%
3. Modernizr	10.8%		14.5%	
4. MooTools	3.1%	-0.1%	4.2%	-0.1%
5. ASP.NET Ajax	2.2%		2.9%	-0.1%

percentages of sites

Клиент-серверные технологии



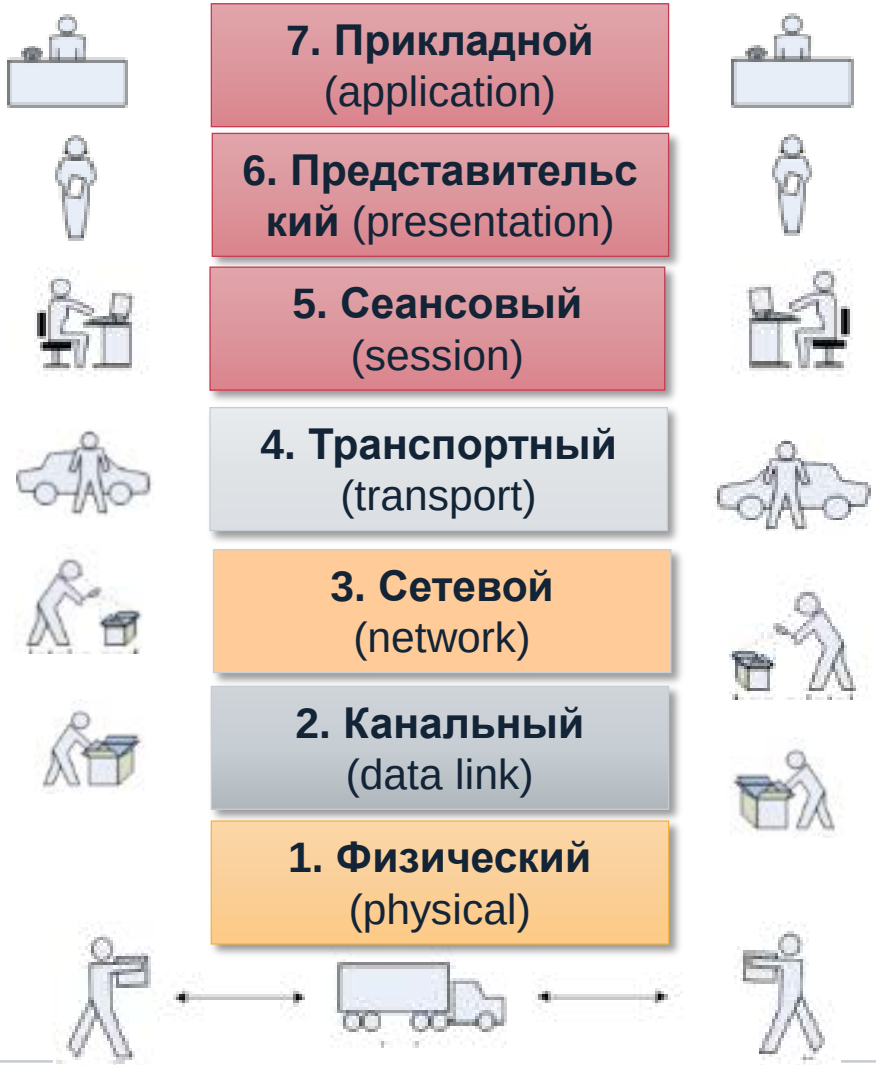
Модель обмена информацией OSI



Модель обмена информацией OSI

Отправка данных

Подготовка сообщения
↓
Форматирование, перевод, зашифровка, сжатие сообщения
↓
Предоставление адреса и запаковка сообщения
↓
Перевозка на почту
↓
Принятие и сортировка
↓
Упаковка
↓
Загрузка



Средство передачи сообщения

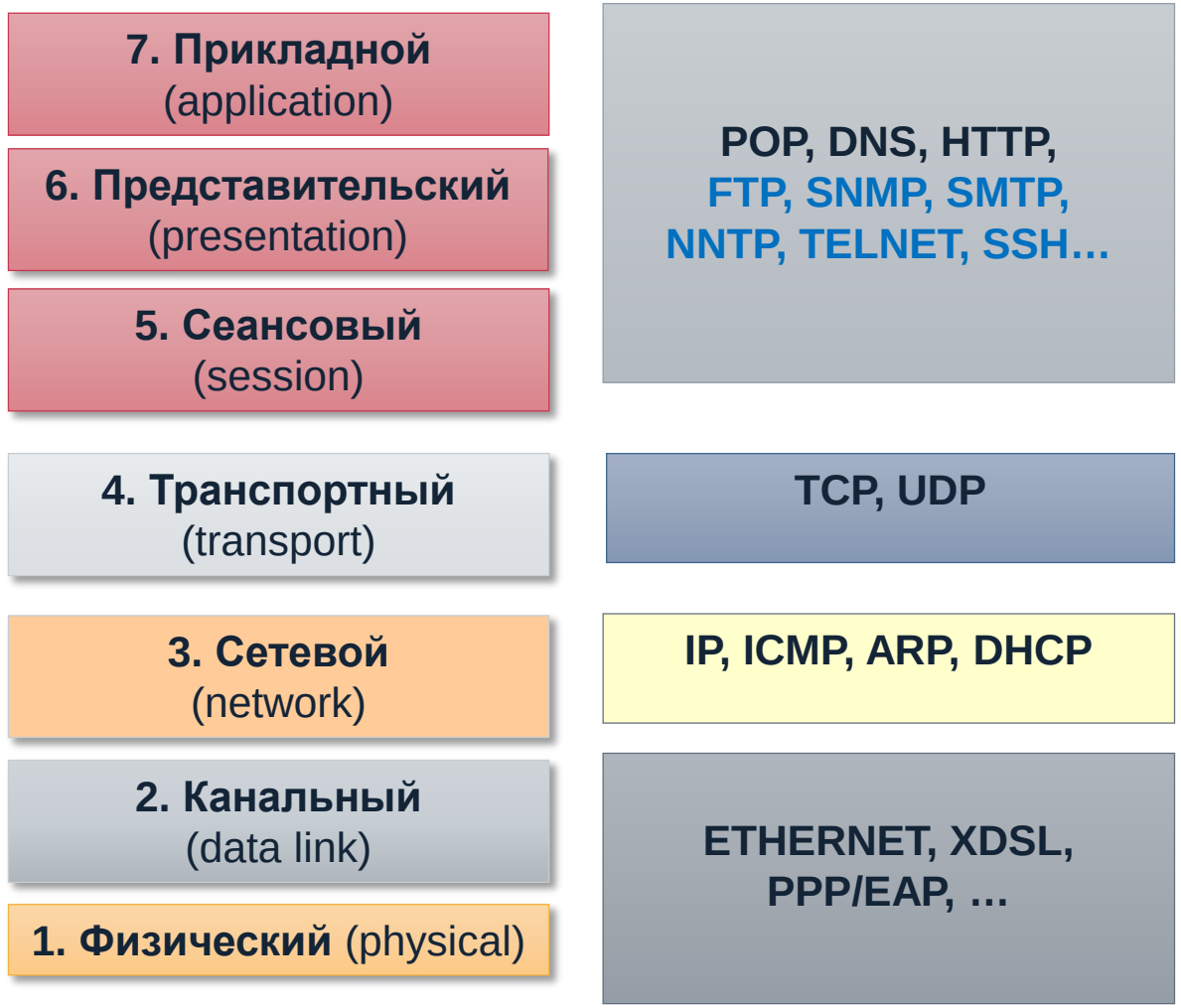
Чтение сообщения
↑
Форматирование, перевод, расшифровка, сжатие сообщения
↑
Проверка адреса и открытие сообщения
↑
Получение на почте
↑
Сортировка и распределение
↑
Распаковка
↑
Разгрузка

Получение данных

Протоколы передачи данных

Протокол – набор правил и соглашений, позволяющих провести обмен информацией между разнородными системами.

Стандартизированный протокол передачи данных также **позволяет разрабатывать интерфейсы** (уже на физическом уровне), **не привязанные к конкретной аппаратной платформе и производителю.**



Протоколы прикладного уровня: HTTP и HTTPS

HTTP (Hypertext Transfer Protocol):

- протокол **прикладного уровня**;
- **обмен контентом** между web-сервером и web-клиентом.
- обмен сообщениями по схеме «**запрос-ответ**».
- для идентификации ресурсов HTTP использует **глобальные URI**.
- работает **поверх TCP/IP**, обеспечивая обмен запросами/ответами между клиентом и сервером.

Порт: 80, 8080

Протоколы прикладного уровня: HTTP и HTTPS

Порт: 443

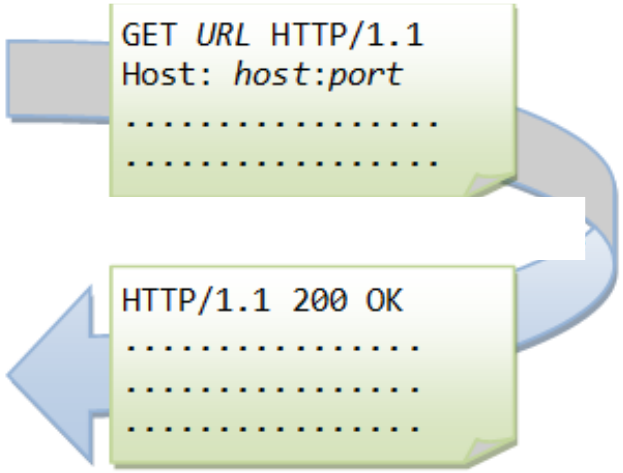
HTTPS (Secure HTTP):

- **расширение протокола HTTP**, поддерживающее **шифрование**;
- **безопасный обмен контентом**;
- **защита от атак;**
- используется для приложений, в которых **важна безопасность соединения**;
- **проприетарный протокол.**

Протоколы прикладного уровня: HTTP и HTTPS

(1) Пользователь запрашивает URL в браузере: http://host:port/path/file

(2) Браузер посылает запрос



(3) Сервер ищет по заданому в URL адресу нужный файл

(4) Сервер возвращает ответ

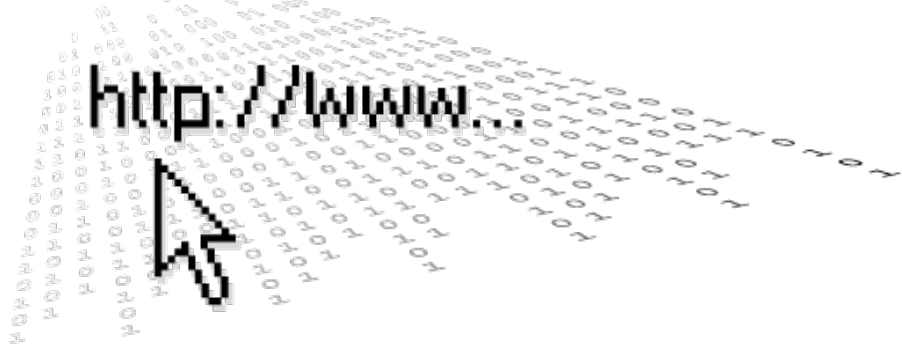
(5) Браузер формирует ответ и отображает его

Клиент (Браузер)

HTTP
(над TCP/IP)

Сервер (@ *host:port*)

Протокол HTTP



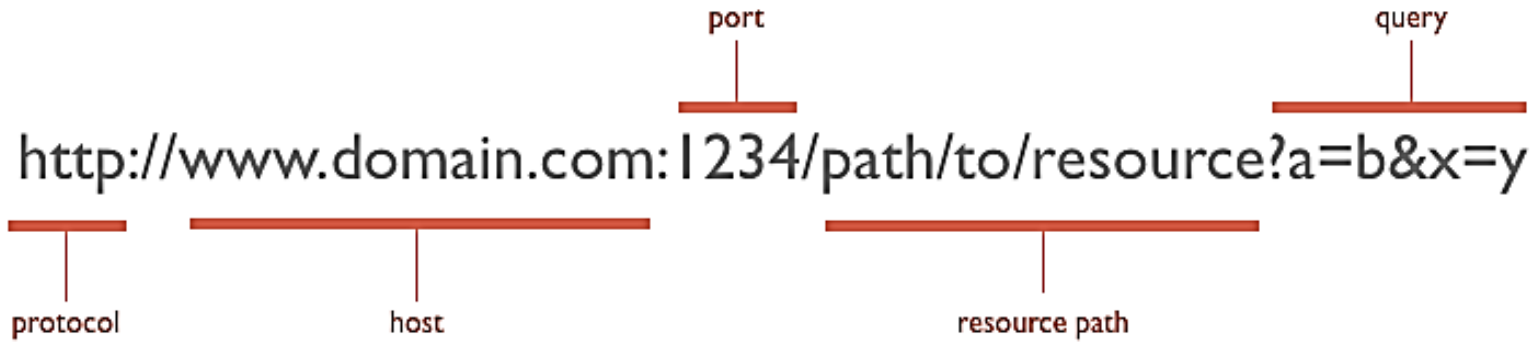
HTTP (Hypertext Transfer Protocol)

Почему он так важен?

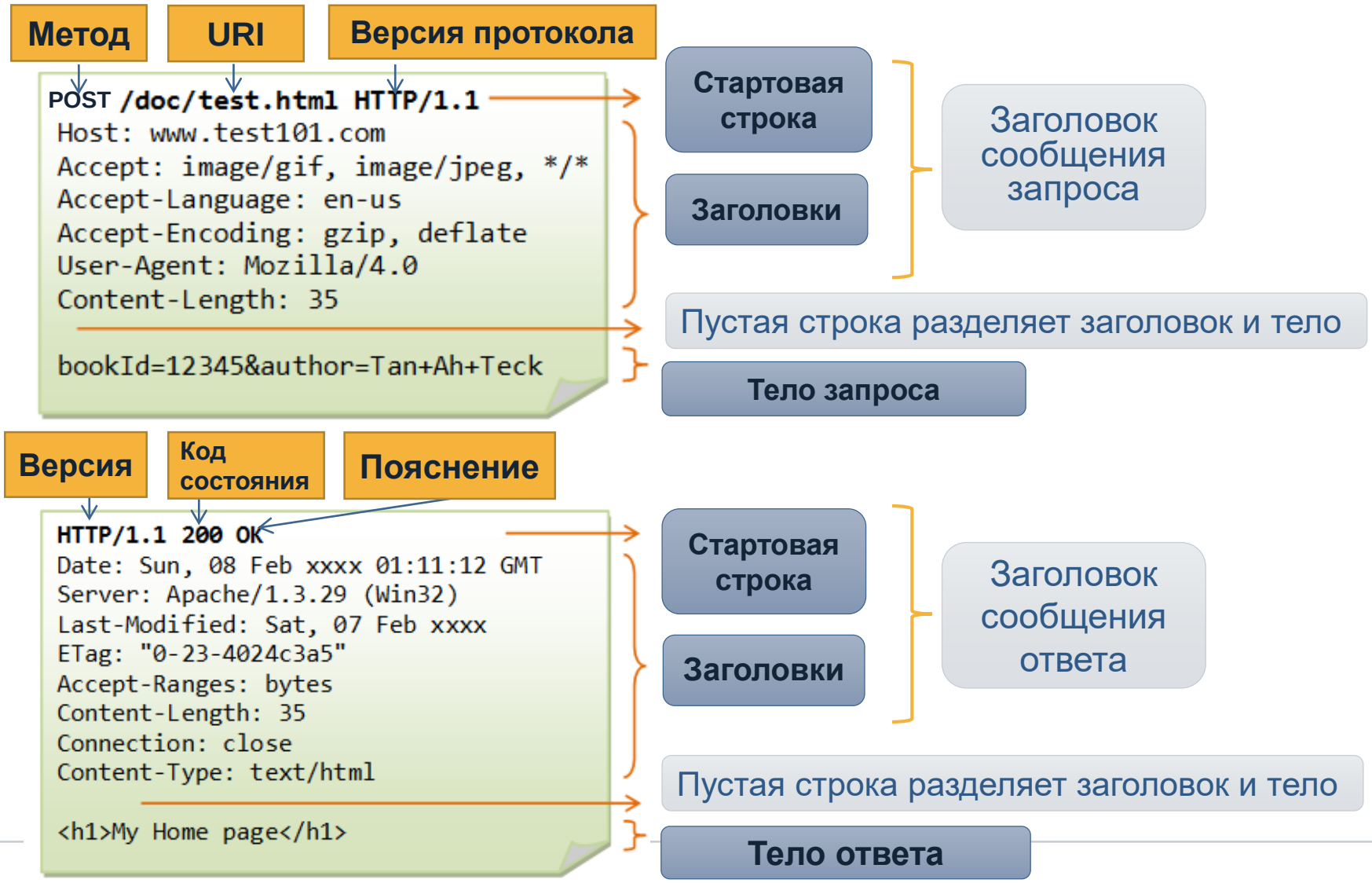
HTTP — протокол, пронизывающий веб.

Знать структуру и методы HTTP обязан каждый веб-разработчик.

Понимание принципов работы HTTP поможет вам делать более качественные веб-приложения.



Структура и методы HTTP запроса и ответа



Структура и методы HTTP запроса и ответа

1xx: Informational (информационные):

102 Processing («идёт обработка»).

2xx: Success (успешно):

200 OK («хорошо»).
206 Partial Content («частичное содержимое»).

3xx: Redirection (перенаправление):

302 Moved Temporarily («перемещено временно»).
304 Not Modified (не изменялось).

4xx: Client Error (ошибка клиента):

400 Bad Request («плохой, неверный запрос»).
401 Unauthorized («неавторизован»).
404 Not Found («не найдено»).

5xx: Server Error (ошибка сервера):

500 Internal Server Error («внутренняя ошибка сервера»).

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

Структура и методы HTTP запроса и ответа

Методы HTTP

POST
передача пользовательских данных заданному ресурсу

GET
запрос содержимого указанного ресурса

LINK
устанавливает связь указанного ресурса с другими

DELETE
удаляет указанный ресурс

PUT
загрузка содержимого запроса на указанный в запросе URI

PATCH
аналогичен PUT, но применяется только к фрагменту ресурса

UNLINK
убирает связь указанного ресурса с другими

CONNECT
преобразует соединение запроса в прозрачный TCP/IP туннель

HEAD
извлечение метаданных, проверки наличия ресурса, обновлений

OPTIONS
определение возможностей веб-сервера или параметров соединения

TRACE
возвращает полученный запрос так, что клиент может увидеть какую информацию промежуточные серверы добавляют или изменяют в запросе

Структура и методы HTTP запроса и ответа

GET

- **тело запроса пустое;**
- запрос обрабатываются **быстрее и с меньшим потреблением ресурсов**;
- передача переменных **в адресной строке** (данные не защищены);
- способен передать **небольшое количество данных**: есть ограничения на длину URL (1024 симв.);
- может передать **только ASCII символы**;
- запрос можно **скопировать, сохранить**;
- запрос может **кэшироваться**;
- доступны **условные и частичные запросы**;
- **не разрывает HTTP соединение** (при включенном на сервере Keep Alive).

?

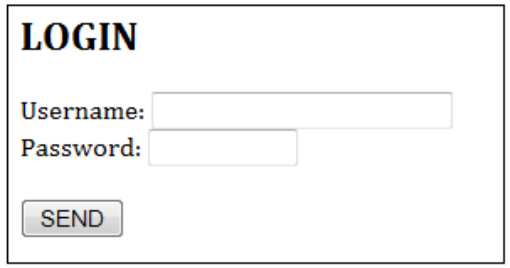
POST

- передача данных **в теле запроса**;
- обработка **медленнее и «тяжелее»**;
- способен передать **большие объемы данных (**лимит устанавливается веб-сервером);
- способен **передать файлы**;
- **нельзя сохранить** в закладки;
- **разрывает HTTP соединение**;
- для передачи информации браузер отправляет **минимум два TCP пакета**: заголовок, а потом тело запроса.

Структура и методы HTTP запроса и ответа

Примеры HTTP запросов

Web-страница:



Исходный код Web-страницы:

POST или GET?

```
<body>
  <h2>LOGIN</h2>
  <form method="    " action="/bin/login">
    Username: <input type="text" name="user" size="25" /><br />
    Password: <input type="password" name="pw" size="10" /><br /><br />
    <input type="hidden" name="action" value="login" />
    <input type="submit" value="SEND" />
  </form>
</body>
```

HTTP **GET** запрос

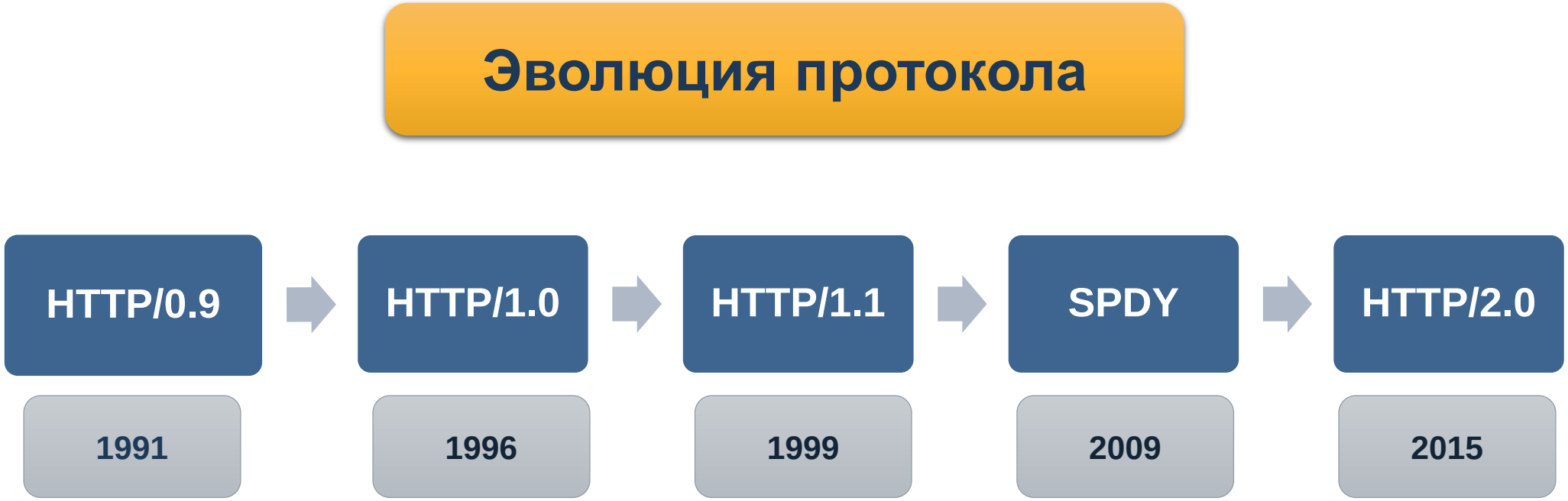
```
GET /bin/login?user=Peter+Lee&pw=123456&action=login HTTP/1.1
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
```

HTTP **POST** запрос

```
POST /bin/login HTTP/1.1
Host: 127.0.0.1:8000
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Length: 37
Connection: Keep-Alive
Cache-Control: no-cache

User=Peter+Lee&pw=123456&action=login
```

История развития протокола HTTP



История развития протокола HTTP HTTP

HTTP/0.9

- Единственный метод — **GET**.
- **Нет заголовков**.
- Спроектирован только для **HTML**-ответов

Если клиенту нужно было получить какую-либо страницу на сервере, он делал запрос:

GET /index.html

Ответ выглядел примерно так:

(response body)
(connection closed)

Сервер получает запрос, посылает HTML в ответ, и как только весь контент будет передан, закрывает соединение.

История развития протокола HTTP

HTTP/1.0

Теперь сервер мог послать любой тип контента клиенту, поэтому словосочетание «Hyper Text» в аббревиатуре HTTP стало искажением. HMTP, или Hypermedia Transfer Protocol, стало бы более уместным названием, но все к тому времени уже привыкли к HTTP.

- Может получать в качестве ответа, помимо HTML, другие форматы: **изображения, видео, текст и другие типы контента**.
- Добавлены новые методы (**POST** и **HEAD**)
- Изменился **формат запросов/ответов**:
  - к запросам и ответам добавились **HTTP-заголовки**;
  - добавлены **коды состояний**, чтобы различать разные ответы сервера.
- Введена **поддержка кодировок**.
- Добавлены составные типы данных (multi-part types), авторизация, кэширование, различные кодировки контента и ещё многое другое.

Один из недостатков HTTP/1.0 — невозможно отправить несколько запросов во время одного соединения.

Протокол HTTP

HTTP/1.1

Потоковая передача данных, при которой клиент может в рамках соединения посылать множественные запросы к серверу, не ожидая ответов, а сервер посылает ответы в той же последовательности, в которой получены запросы.

Новые HTTP-методы — PUT, PATCH, HEAD, OPTIONS, DELETE.

Идентификация хостов (обязательность заголовка HOST) . В HTTP/1.0 заголовок Host не был обязательным.

Постоянные соединения, т.е. соединения, которые по умолчанию не закрывались, оставаясь открытыми для нескольких последовательных запросов. **Connection: keep-alive.**
Чтобы закрыть соединение, нужно было при запросе добавить заголовок **Connection: close**.
Клиенты обычно посылали этот заголовок в последнем запросе к серверу, чтобы безопасно закрыть соединение.

История развития протокола HTTP

HTTP/1.1

HTTP/1.1 ввёл **chunked encoding** — механизм разбиения информации на небольшие части (chunks) и их передачу.

Но как же клиент узнает, когда закончится один ответ и начнётся другой? Для разрешения этой задачи устанавливается **заголовок Content-Length**, с помощью которого клиент определяет, где заканчивается один ответ и можно ожидать следующий.

Chunked Transfers. Если контент строится динамически и сервер в начале передачи не может определить *Content-Length*, он начинает отсылать контент частями, друг за другом, и добавлять *Content-Length* к каждой передаваемой части. Когда все части отправлены, посылается пустой пакет с заголовком *Content-Length*, установленным в 0, сигнализируя клиенту, что передача завершена. Чтобы сказать клиенту, что передача будет вестись по частям, сервер добавляет заголовок *Transfer-Encoding: chunked*.

История развития протокола HTTP

HTTP/1.1

HTTP/1.1 появился в 1999 и пробыл стандартом долгие годы.

В отличие от базовой аутентификации в HTTP/1.0, в HTTP/1.1 добавились:

- **Дайджест-аутентификация и прокси-аутентификация.**
- **Кэширование**.
- **Диапазоны байт** (byte ranges).
- **Кодировки.**
- **Согласование содержимого (content negotiation).**
- **Клиентские куки.**
- **Улучшенная поддержка сжатия.**
- И другие...

История развития протокола HTTP

SPDY

В 2015 в Google решили, что не должно быть двух конкурирующих стандартов, и объединили SPDY с HTTP, дав начало HTTP/2

Основная идея SPDY:
сделать веб быстрее и улучшить уровень безопасности за счёт уменьшения времени задержек веб-страниц.

SPDY включал в себя мультиплексирование, сжатие, приоритизацию, безопасность и т.д…

SPDY не старался заменить собой HTTP. Он был переходным уровнем над HTTP

История развития протокола HTTP

HTTP/2.0

HTTP/2 разрабатывался **для транспортировки контента с низким временем задержки**.

HTTP/2 уже здесь, и уже обошёл SPDY в поддержке

Главные отличия от HTTP/1.1:

- **бинарный вместо текстового.** Бинарные сообщения быстрее разбираются автоматически, но, в отличие от HTTP/1.x, не удобны для чтения человеком. Основные составляющие HTTP/2 — фреймы (Frames) и потоки (Streams).
- **мультиплексирование — передача нескольких асинхронных HTTP-запросов по одному TCP-соединению:** клиенту не придётся простаивать, ожидая обработки длинного запроса, ведь во время ожидания могут обрабатываться остальные запросы.
- **сжатие заголовков методом HPACK**
- **Server Push — несколько ответов на один запрос:** сервер, зная, что клиент собирается запросить определённый ресурс, может отправить его, не дожидаясь запроса.
- **приоритизация запросов**
- **безопасность**

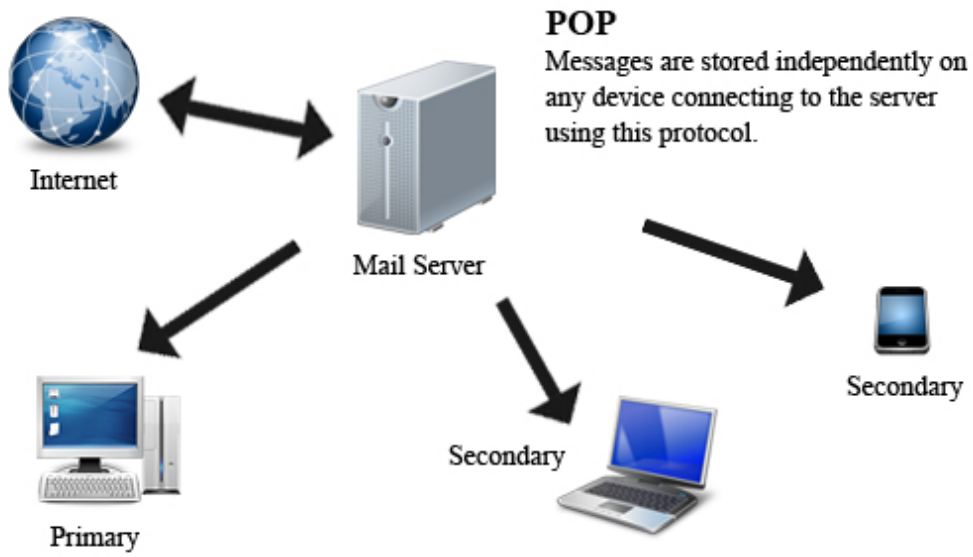
Почтовые протоколы: POP3/IMAP, SMTP

POP3 (Post Office Protocol),
IMAP (Internet Message Access Protocol)

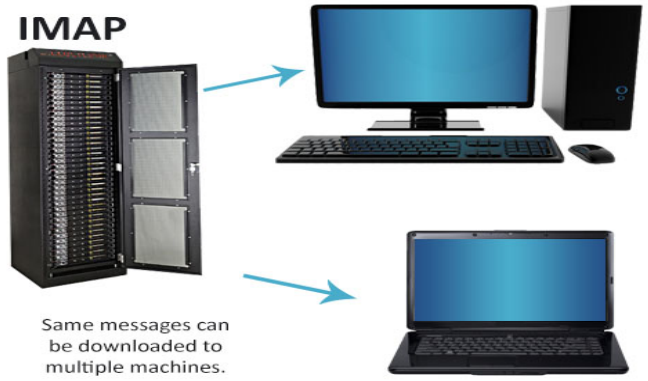
- используются клиентами электронной почты для извлечения электронного сообщения с удаленного сервера по TCP/IP-соединению.

POP3
Порты: 110, 995

IMAP
Порты: 143, 993



Недостаток POP3:
отсутствие возможностей по управлению перемещением и хранением сообщений на сервере.



Почтовые протоколы: POP3/IMAP, SMTP

SMTP (Simple Mail Transfer Protocol):

- предназначен для отправки клиентами электронной почты на сервер электронной почты в сетях TCP/IP.

Порты: 25, 587

Электронные почтовые серверы и другие агенты:
отправка и получение почтовых сообщений.

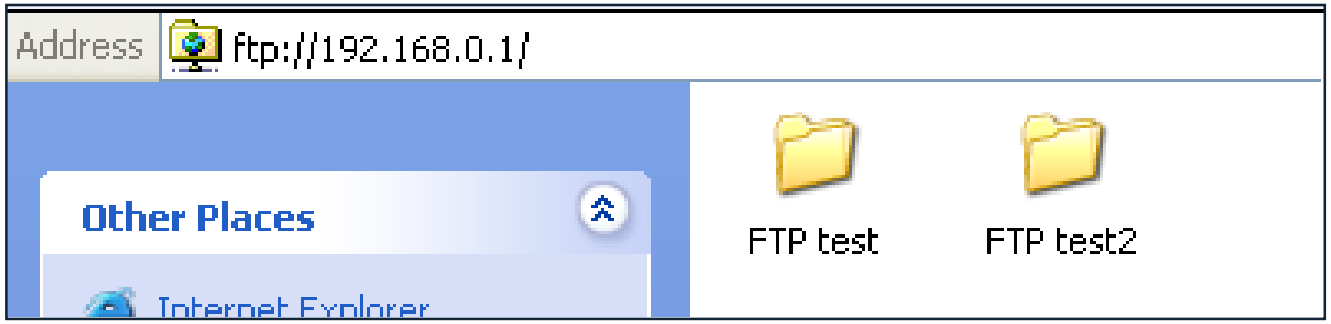
Клиентские почтовые приложения на пользовательском уровне:
только отправка сообщений на почтовый сервер для ретрансляции.

Другие протоколы прикладного уровня: FTP, SSH, SFTP, RDP

FTP (File Transfer Protocol)

- протокол прикладного уровня;
- для передачи файлов по TCP-сетям (например, Интернет);
- часто используется для загрузки сетевых страниц и других документов с частного устройства разработки на открытые сервера хостинга;
- широко используется для распространения ПО и доступа к удалённым хостам.

Порт: 21, 20



Другие протоколы прикладного уровня: FTP, SSH, SFTP, RDP

SSH (Secure Shell)

- сетевой протокол прикладного уровня;
- позволяет производить удалённое управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов);
- допускает выбор различных алгоритмов шифрования.
- позволяет безопасно передавать в незащищённой среде сетевые протоколы.
- позволяет не только удалённо работать на компьютере через командную оболочку, но и передавать по шифрованному каналу звуковой поток или видео (например, с веб-камеры).
- может использовать сжатие передаваемых данных для последующего их шифрования.

Порт: 22

Другие протоколы прикладного уровня: FTP, SSH, SFTP, RDP

SFTP (SSH File Transfer Protocol)

- протокол прикладного уровня, предназначенный для копирования и выполнения других операций с файлами поверх надёжного и безопасного соединения;
- использует SSH для передачи файлов;
- не связан с FTP, за исключением того, что он тоже передаёт файлы и имеет аналогичный набор команд для пользователей;
- шифрует и команды, и данные, предохраняя пароли и конфиденциальную информацию от открытой передачи через сеть;
- по функциональности похож на FTP, но так как он использует другой протокол, клиенты стандартного FTP не могут связаться с SFTP-сервером и наоборот.

Другие протоколы прикладного уровня: FTP, SSH, SFTP, RDP

RDP (Remote Desktop Protocol)

- **проприетарный протокол прикладного уровня;**
- **протокол удалённого рабочего стола;**
- **куплен Microsoft у Citrix;**
- **используется для обеспечения удалённой работы пользователя с сервером, на котором запущен сервис терминальных подключений.**

Порт: 3389

mstsc.exe – клиент в **Windows 2k/XP/2003/ Vista/2008/7/8**

Основные протоколы транспортного уровня: TCP, UDP

TCP (Transmission Control Protocol - протокол управления передачей) — один из основных протоколов передачи данных Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP.

- управление передачей данных на уровне приложений;
- подразумевает проверку соединения;
- проверка факта доставки и порядка доставки данных;
- широко используется в системах, не чувствительных ко времени.

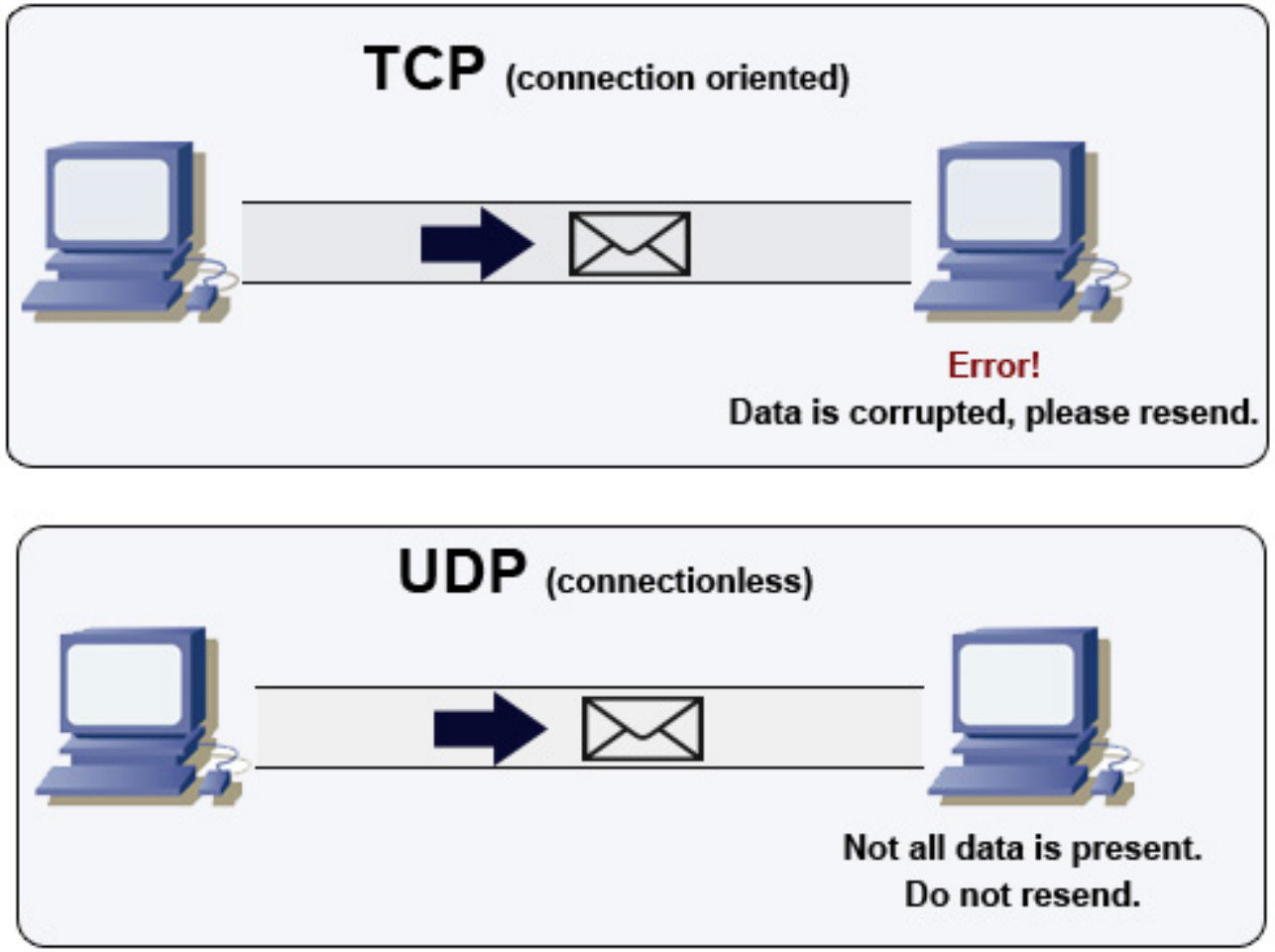
Основные протоколы транспортного уровня: TCP, UDP

UDP (User Datagram Protocol — протокол пользовательских датаграмм) — один из ключевых элементов TCP/IP, набора сетевых протоколов для Интернета.

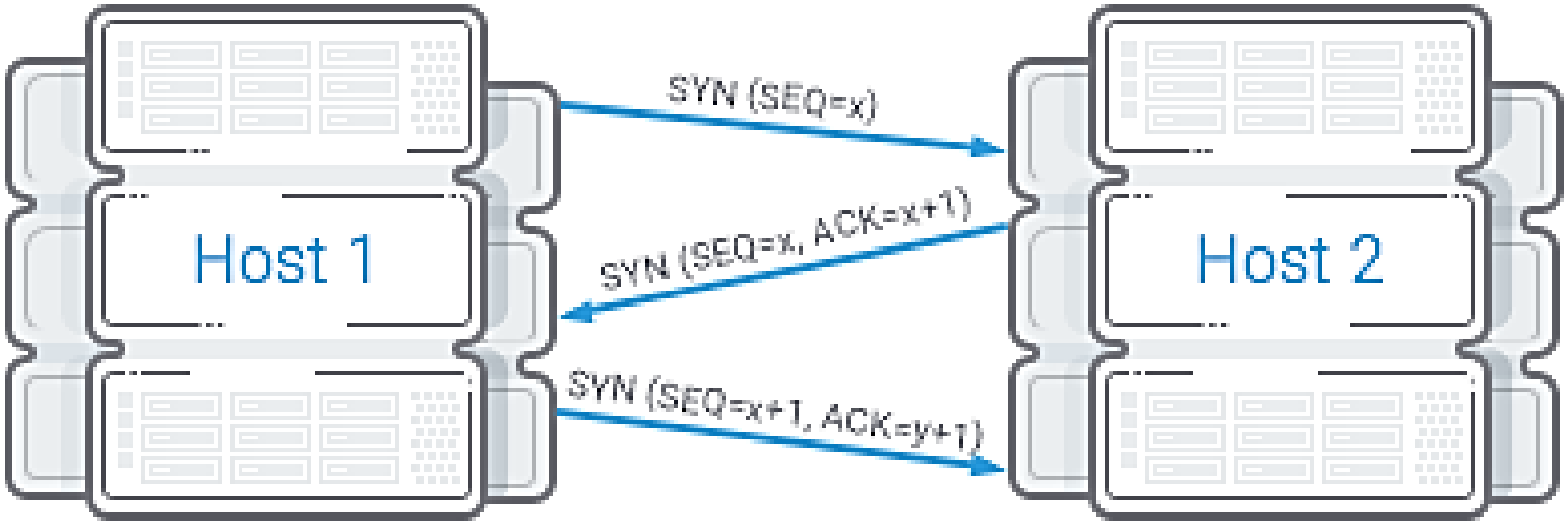
С UDP компьютерные приложения могут посылать сообщения (датаграммы) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

- **управление передачей данных на уровне приложений;**
- **нет проверки факта доставки и порядка доставки данных;**
- **не подразумевает проверку соединения;**
- **широко используется в системах реального времени.**

Основные протоколы транспортного уровня: TCP, UDP



Основные протоколы транспортного уровня: TCP, UDP



SYN — синхронизация номеров последовательности
ACK — поле *«Номер подтверждения»*

Материалы для самостоятельного изучения

- http:// www.w3schools.com/
- http://htmlbook.ru/
- http://www.codenet.ru/webmast/php/HTTP-POST.php
- http://ru.wikipedia.org/wiki/Протокол_передачи_данных
- https://ru.wikipedia.org/wiki/Microsoft_Exchange_Server
- www.google.com

Q&A

Thank You

Focused forward

Netcracker