OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование

Не забыть включить запись!

Меня хорошо видно && слышно?

Ставьте + , если все хорошо

Напишите в чат, если есть проблемы

Правила вебинара

Активно участвуем

Задаем вопрос в чат или голосом

Off-topic обсуждаем в Slack #канал группы или #general

Вопросы вижу в чате, могу ответить не сразу

Протокол HTTP (**часть** 1)

Викирюк Павел

Системный инженер

Маршрут вебинара

Протокол HTTP

↓

REST API

↓

HTTPS

↓

Веб-серверы

Цели занятия | После занятия вы сможете

1. Понять основные принципы работы протоколов HTTP и HTTPS

2. Познакомиться с основными требованиями архитектурного стиля REST

3. Познакомиться с веб-серверами Apache и NGINX

Смысл | Зачем вам это уметь

1. Чтобы понимать как работает протокол HTTP

2. Чтобы отлаживать и оптимизировать работу приложений, связанных с использованием протокола HTTP

3. Чтобы работать с сервисами, связанными с высокой нагрузкой

HTTP: **немного истории**

HTTP: **немного истории**

HTTP (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных

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

Факты:

- текстовый протокол одностороннего свойства
- в основе лежит технология “клиент-сервер”
- версия 0.9 появилась в 1992 г.
- версия 1.0 выпущена в 1996 г., описан в **RFC 1945**
- версия 1.1 выпущена в 1999 г.
- версия 2 выпущена в 2015 г.

HTTP/1.1: **особенности**

HTTP: **особенности**

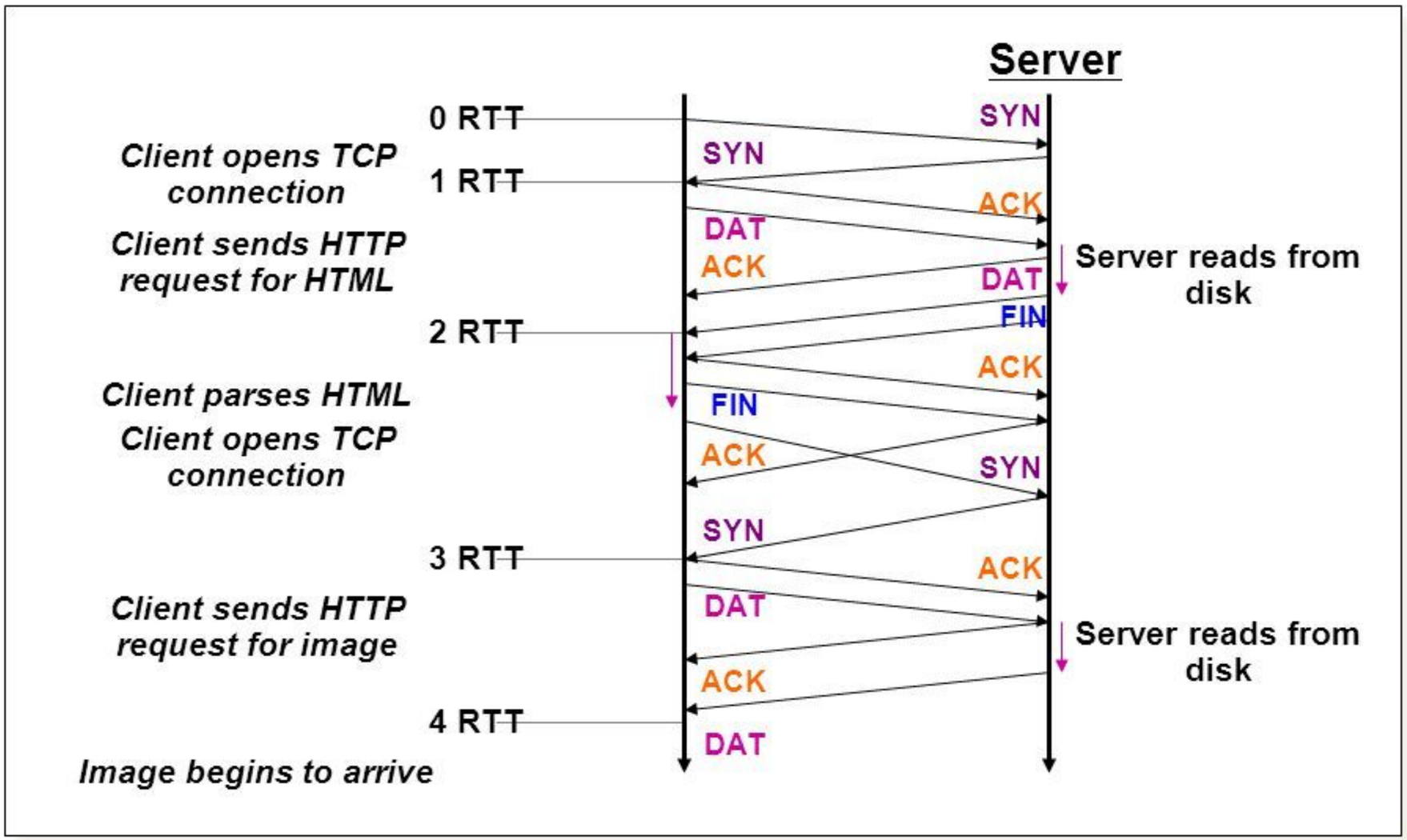
HTTP/1.1:

- режим постоянного соединения (несколько запросов через одно TCP-соединение)
- нет ожидания ответа на каждый запрос
- определяет, как взаимодействуют между собой клиент и сервер
- определяет как запрашивается и передает контент
- для идентификации ресурсов использует глобальные **URI**

HTTP: **особенности**

16

Single Transfer Example



HTTP: **особенности**

Термины:

URI (Uniform Resource Identifier) - унифицированный идентификатор ресурса:

- может указывать на имя, локацию или на то и другое

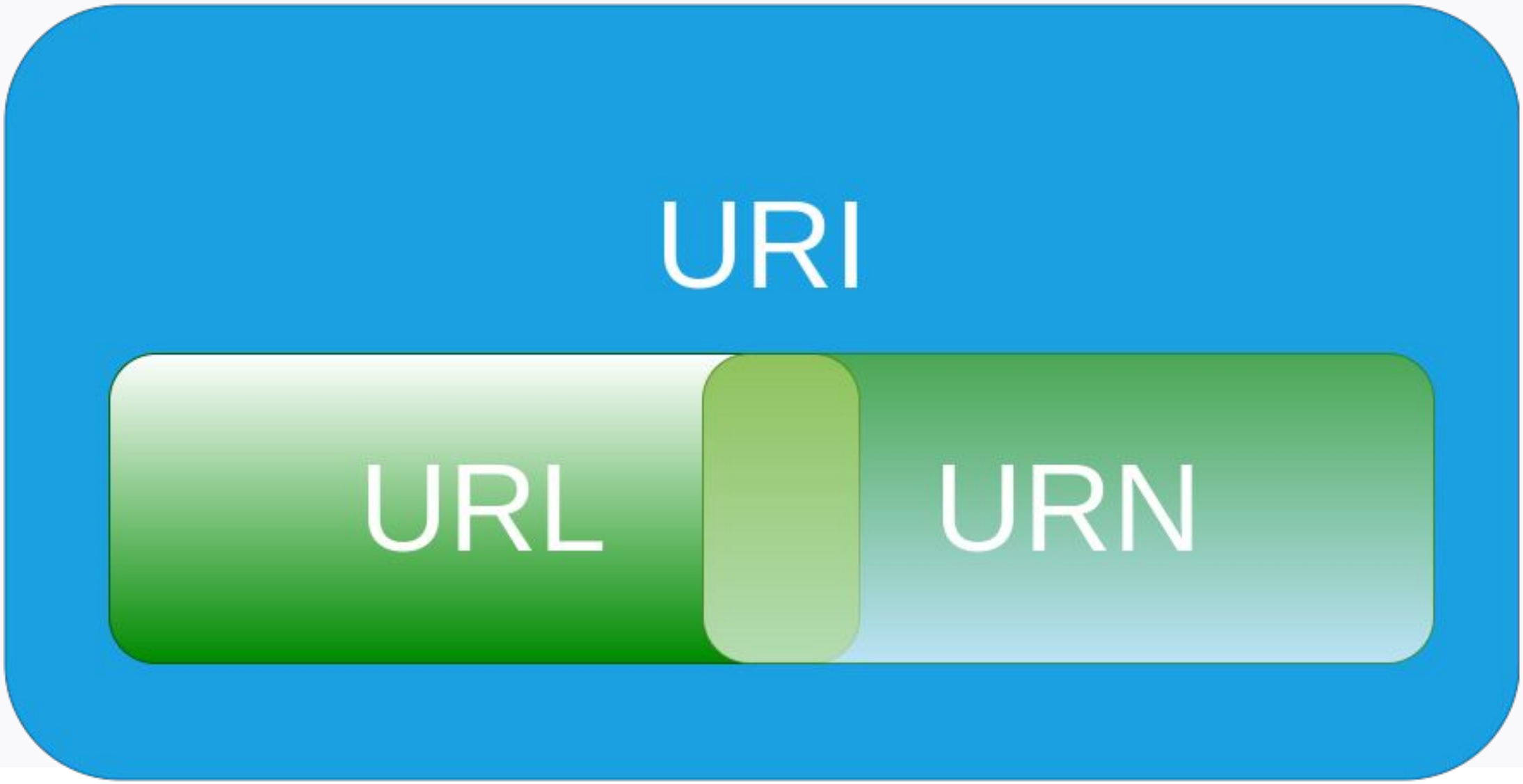
URL (Uniform Resource Locator) - унифицированный указатель ресурса

- определяет местонахождение ресурса
- используется как стандарт записи ссылок на объекты в Интернет
- стандарт URL закреплен в **RFC 3986**

URN (Uniform Resource Name) - единообразное название (имя) ресурса

- набор символов, описывающий ресурс
- не указывает на местонахождение и способ обращения

HTTP: **особенности**



HTTP: **особенности**

Примеры:

URI, URL = https://otus.ru/nest/post/632/

URL, URI = https://otus.ru

URN, URI = /nest/post/632/

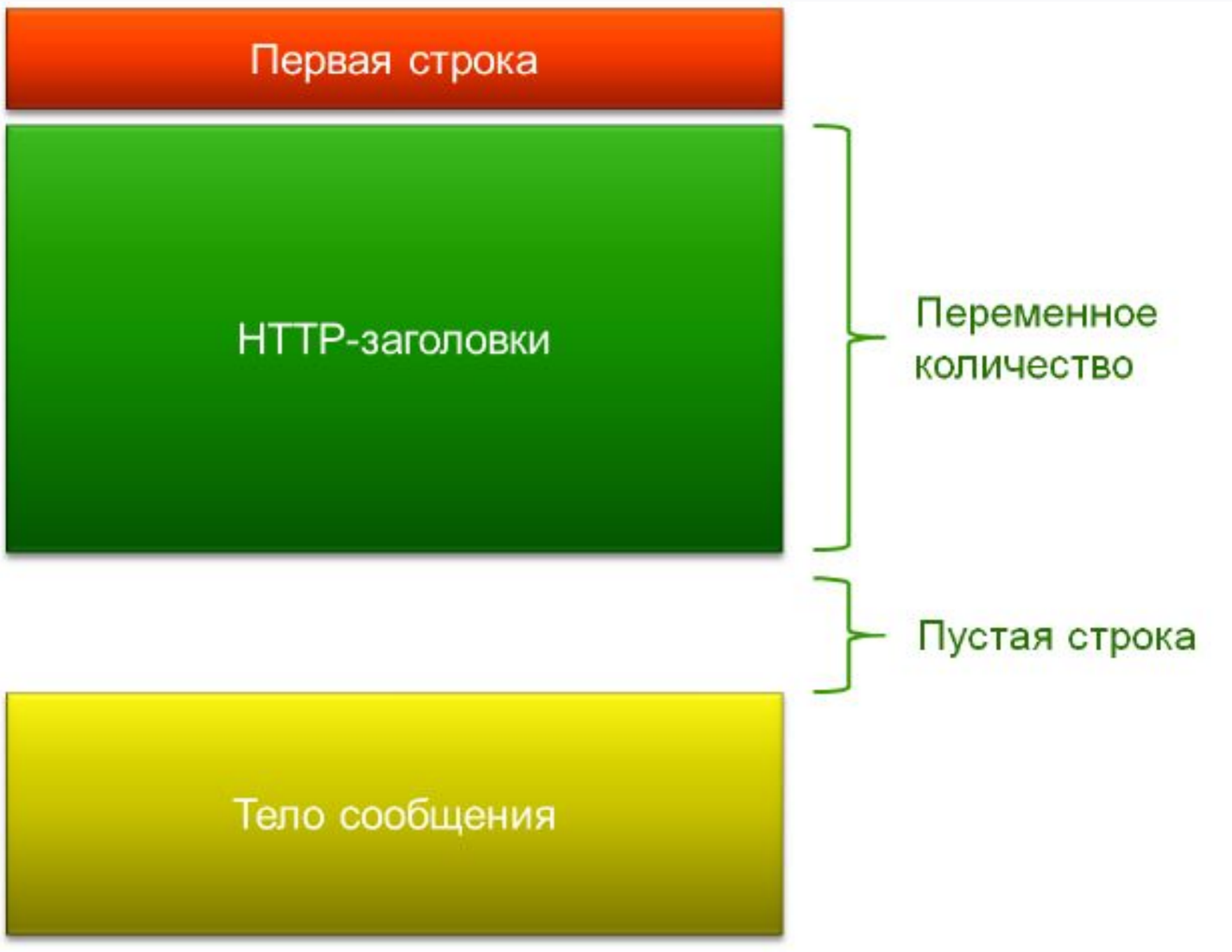
Синтаксис:

protocol://user:password@host:port/resource

http://www.site.ru/index.html
smtp://user@mail.com
imap://user:password@mail.com/inbox
mysql://user:password@dbms.com/database/table
ampq://rabbit.host/vhost/queue

HTTP: **структура и методы**

HTTP: **структура и методы**



HTTP: **структура и методы**

HTTP-сообщения состоят из трех частей, которые передаются в указанном порядке:

1. Стартовая строка - определяет тип сообщения
2. Заголовки - характеризуют тело сообщения, параметры передачи и прочие сведения
3. Тело сообщения - данные. Обязательно отделяется от заголовков пустой строкой

В HTTP/1.1

- стартовая строка и заголовок являются обязательными элементами
- обязательно содержит заголовок **Host**

HTTP: **структура и методы**

Методы - последовательность символов, указывающая на основную операцию над ресурсом

GET - получить содержимое указанного ресурса
HEAD - получить только заголовки
POST - отправить данные
DELETE - удалить указанный ресурс
OPTIONS - определить параметры сервера

HTTP: **заголовки**

HTTP: **заголовки**

Заголовки - строки в HTTP-сообщении, содержащие разделенную двоеточием пару **параметр-значение**.

Пример заголовков:

Server: nginx/1.16.1
Date: Tue, 10 Dec 2019 16:32:07 GMT
Content-Type: image/jpeg
Content-Length: 33519
Last-Modified: Mon, 19 Feb 2018 04:59:24 GMT
Connection: keep-alive
ETag: "1a7a93ac-34ef"
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: *
Access-Control-Allow-Headers: *
Access-Control-Max-Age: 86400
Accept-Ranges: bytes

HTTP: **заголовки**

Заголовки подразделяются на четыре основные группы:

General Headers (основные заголовки) - могут включаться в любое сообщение клиента и сервера

Request Headers (заголовки запроса) - используются только в запросах клиента

Response Headers (заголовки ответа) - только для ответов от сервера

Entity Headers (заголовки сущности) - сопровождают каждую сущность сообщения

HTTP: **заголовки**

Особенности:

- все необходимые заголовки описаны в основных RFC
- можно добавлять свои кастомные заголовки
- традиционно кастомные заголовки начинаются с префикса “**X-**”, чтобы избежать конфликта имен с существующими
- например заголовок **referer** позволяет серверу узнать откуда был осуществлен переход на запрашиваемую страницу. Сервер может анализировать эти данные, записывать их в логи или оптимизировать процесс кэширования
- заголовок **Location** с кодом ответа **301/302** осуществляет переадресацию клиента

HTTP: cookies

HTTP: cookies

Cookies - это набор данных, в формате ключ-значение, привязанный к домену. Используется для:

- аутентификации пользователя

HTTP: cookies

Cookies - это набор данных, в формате ключ-значение, привязанный к домену. Используется для:

- аутентификации пользователя
- хранения персональных предпочтений и настроек пользователя
- отслеживания состояния пользовательской сессии
- сведения статистики о пользователях

Особенности:

- отправляется клиенту в заголовке Set-Cookie
- браузер обязан отправить cookie обратно серверу при следующем запросе
- в теории в cookie можно хранить все, что угодно, например содержимое корзины в интернет-магазине

HTTP: **недостатки**

HTTP: **недостатки**

Недостатки:

- блокировка очереди (head-of-line blocking, или HOL)
- ограничения на количество одновременных TCP-соединений
- избыточность передаваемых данных
- больший размер сообщений

Ваши вопросы?

Механизмы HTTP

Chunked transfer encoding

Chunked transfer encoding - механизм передачи данных в HTTP, позволяющий надежно доставлять данные от сервера клиенту, без необходимости знать точный размер сообщения

- сервер добавляет заголовок **Transfer-Encoding: chunked**
- данные разбиваются на небольшие части
- каждая часть передается с указанием только ее размера
- окончание передачи определяется наличием пустого пакета с заголовком **Content-Length: 0**

Частичные GET

Частичные GET:

- HTTP позволяет запросить не сразу все содержимое ресурса, а только указанный фрагмент
- используются заголовки **Range**, которые содержат байтовые диапазоны
- механизм должен поддерживаться сервером

Способы передачи фрагментов сервером:

- в ответе передается заголовок **Content-Range** с указанием байтовых диапазонов в соответствии с которыми фрагменты последовательно помещаются в тело сообщения
- сервер указывает медиатип **multipart/byteranges** для основного содержимого и передаёт фрагменты, указывая соответствующий Content-Range для каждого элемента

Условные GET

Условные GET:

- GET является условным, если содержит поле заголовка **If-Modified-Since**

Условные GET

Условные GET:

- GET является условным, если содержит поле заголовка **If-Modified-Since**
- тело ресурса передается только если он изменялся **после** даты, указанной в заголовке

Условные GET

Условные GET:

- GET является условным, если содержит поле заголовка **If-Modified-Since**
- тело ресурса передается только если он изменялся **после** даты, указанной в заголовке
- разгружает сеть, так как сокращает объем избыточной информации

Content negotiation

Content negotiation (согласование содержимого) - механизм автоматического определения необходимого ресурса при наличии нескольких разнотипных версий документа

Согласование, управляемое клиентом:

- тип содержимого определяется на стороне клиента
- сервер возвращает ответ с кодом 300 (Multiple Choices) или 406 (Not Acceptable) и список вариантов, из которых можно выбрать

Кэширование

Кэширование

Кэширование - метод, заключающийся в сохранении копии полученного ресурса, чтобы вернуть ее по запросу

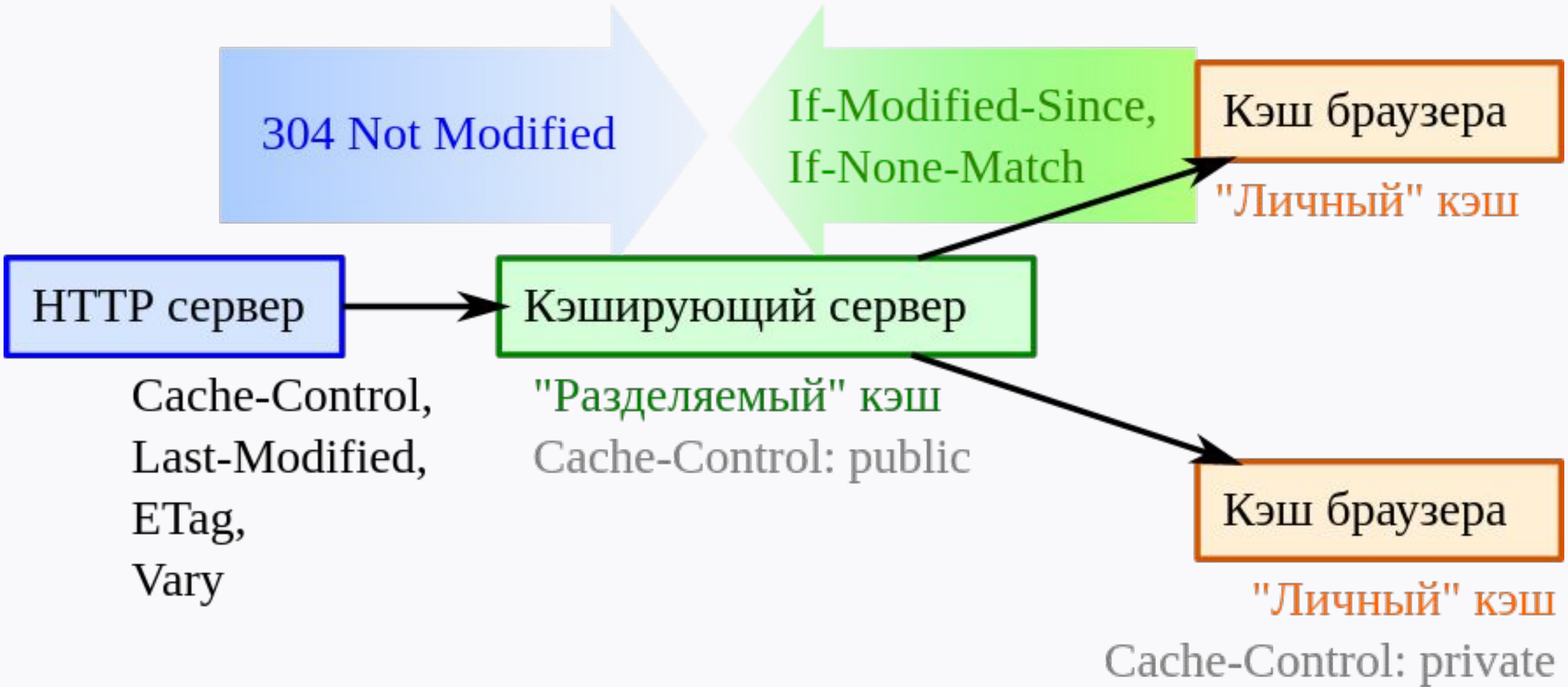
Особенности:

- кэшируются ответы на GET-запросы
- управление кэшированием осуществляется с помощью заголовков

Уровни кэширования:

- клиентский
- сетевой
- серверный
- уровень приложения

Кэширование



Кэширование

Заголовки ответа сервера:

Expires - время, после которого контент будет неактуальным

Etag - ID или метка контента

Cache-Control - передачи инструкций по механизму кэширования

Кэширование

Заголовки ответа сервера:

Expires - время, после которого контент будет неактуальным

Etag - ID или метка контента

Cache-Control - передачи инструкций по механизму кэширования

Заголовки запроса:

If-Modified-Since - отдать ресурс, если изменилось, возвращает 200 или 304

If-None-Match: Etag - отдать, если есть у ресурса такая метка

Ваши вопросы?

Маршрут вебинара

Протокол HTTP

REST API

HTTPS

Веб-серверы

REST API

REST API

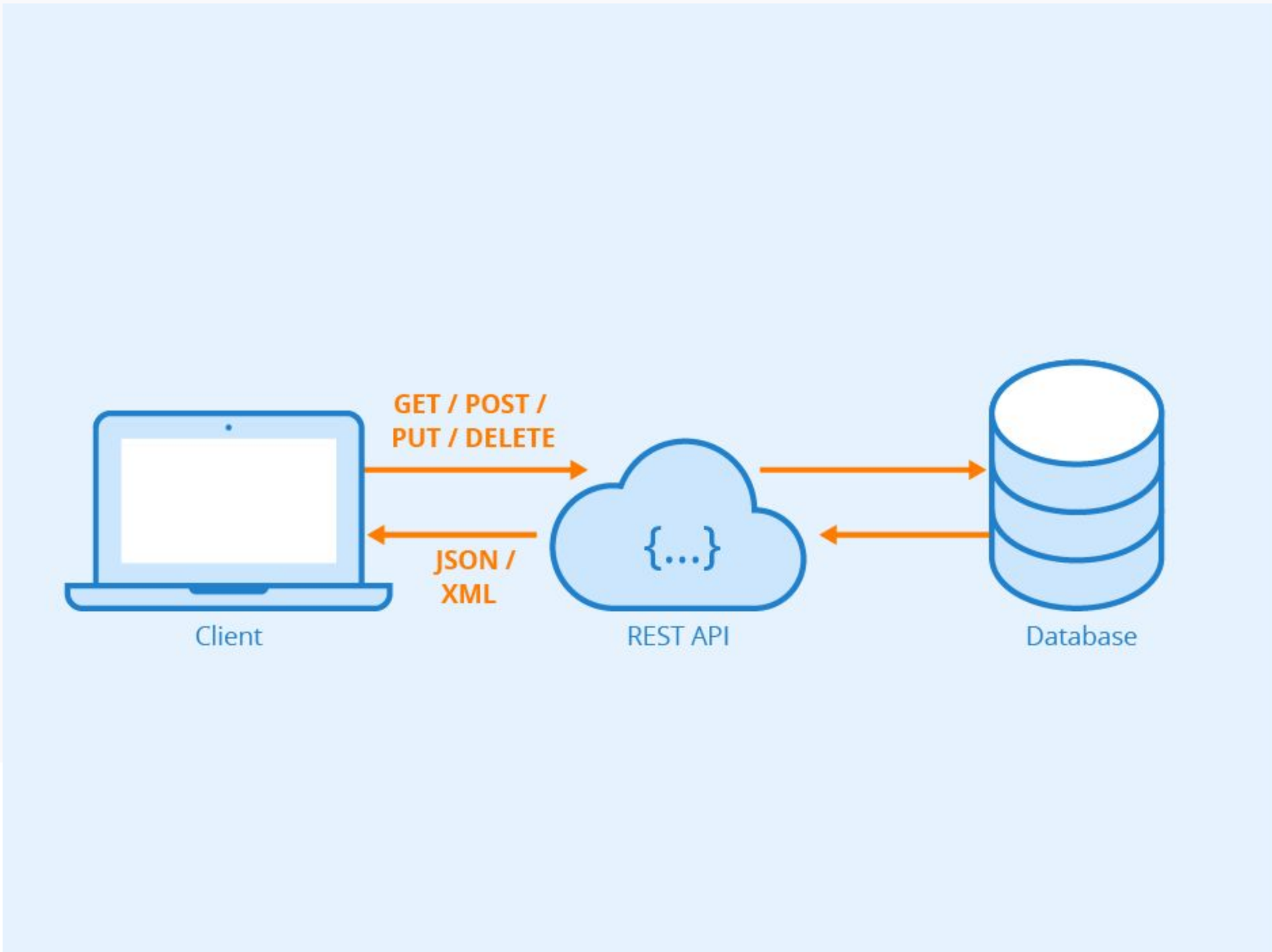
REST (сокращение от англ. Representational State Transfer — «передача состояния представления») — архитектурный стиль взаимодействия компонентов распределённого приложения в сети.

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

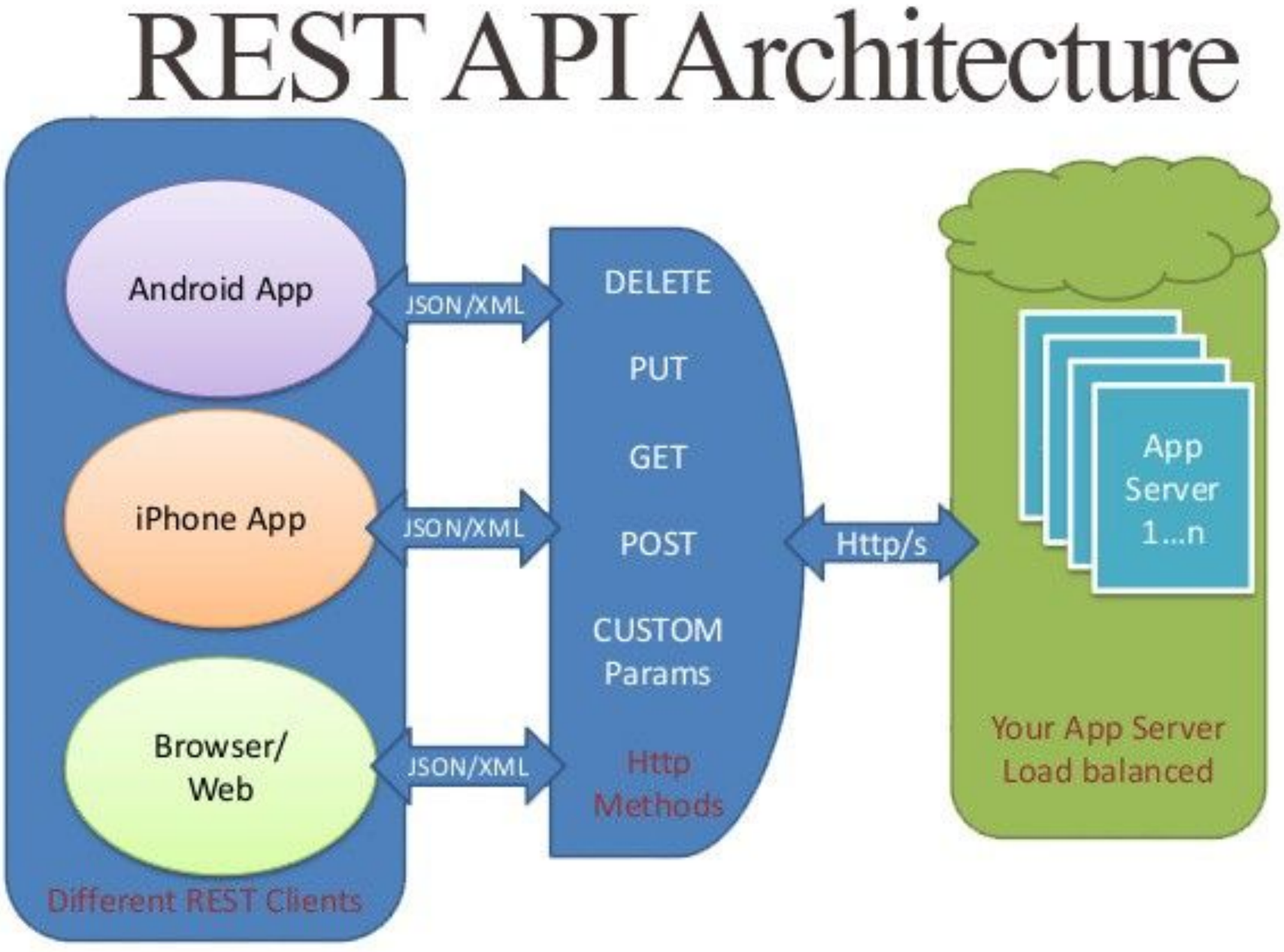
Особенности:

- автором идеи и термина является Рой Филдинг 2000 г.
- производительность
- масштабируемость взаимодействия компонентов системы
- независимое внедрение компонентов
- промежуточные компоненты, снижающие задержку, усиливающие безопасность

REST API



REST API


REST API Architecture
Android App
iPhone App
Browser/
Web
Different REST Clients
JSON/XML
JSON/XML
JSON/XML
DELETE
PUT
GET
POST
CUSTOM
Params
Http
Methods
Http/s
App
Server
1...n
Your App Server
Load balanced

REST API

Основные принципы (требования) к REST архитектуре:

1. **Модель клиент-сервер**
   - повышает переносимость кода клиентов на другие платформы
   - улучшает масштабируемость кода сервера
2. **Отсутствие состояния**
   - stateless protocol - на сервере не хранится информация о состоянии клиента
3. **Кэширование**
   - кэширование для улучшения производительности интерфейса
4. **Единообразие интерфейса**
   - идентификация ресурсов (идентификатором в REST является URI)
   - манипуляция ресурсами
   - самодокументируемые сообщения (в сообщении содержится вся необходимая информация)
   - HATEOAS (hypermedia as the engine of application state) - Статус ресурса передается через содержимое body, параметры строки запроса, заголовки запросов и запрашиваемый URI (имя ресурса)

Основные принципы (требования) к REST архитектуре:

5. Слои (Layered System)

- иерархическое деление системы
- конфиденциальность вызовов в рамках одного слоя системы

6. Код по требованию (необязательное ограничение)

- возможность загрузки и выполнения кода на стороне клиента

Ваши вопросы?

Маршрут вебинара

Протокол HTTP

REST API

HTTPS

Веб-серверы

HTTPS

HTTPS

HTTPS (аббр. от англ. HyperText Transfer Protocol Secure) - расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов SSL или TLS. В отличие от HTTP с TCP-портом 80, для HTTPS по умолчанию используется TCP-порт 443

Особенности:

- является все тем же обычным HTTP, работающим поверх транспортных механизмов TLS и SSL
- требует установки сертификатов открытого и закрытого ключа на сервер
- сертификат открытого ключа подтверждает принадлежность данного открытого ключа владельцу сайта
- сертификат открытого ключа и сам открытый ключ посылаются клиенту при установлении соединения
- закрытый ключ используется для расшифровки сообщений от клиента

HTTPS

SSL (англ. Secure Sockets Layer — уровень защищённых сокетов) — криптографический протокол, который подразумевает более безопасную связь.

- использует асимметричную криптографию для аутентификации ключей обмена
- симметричное шифрование для сохранения конфиденциальности
- коды аутентификации сообщений для целостности сообщений
- в 2014 году правительство США сообщило об уязвимости в текущей версии протокола
- SSL должен быть исключён из работы в пользу TLS (CVE-2014-3566)

HTTPS

TLS (англ. transport layer security — Протокол защиты транспортного уровня), также криптографический протокол, основанный на спецификации протокола SSL версии 3.0

Особенности:

- использует асимметричное шифрование для аутентификации
- симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений

Уровни защиты:

- шифрование данных - позволяет избежать их перехвата
- сохранность данных - любое изменение данных фиксируется
- аутентификация - защищает от перенаправления пользователя

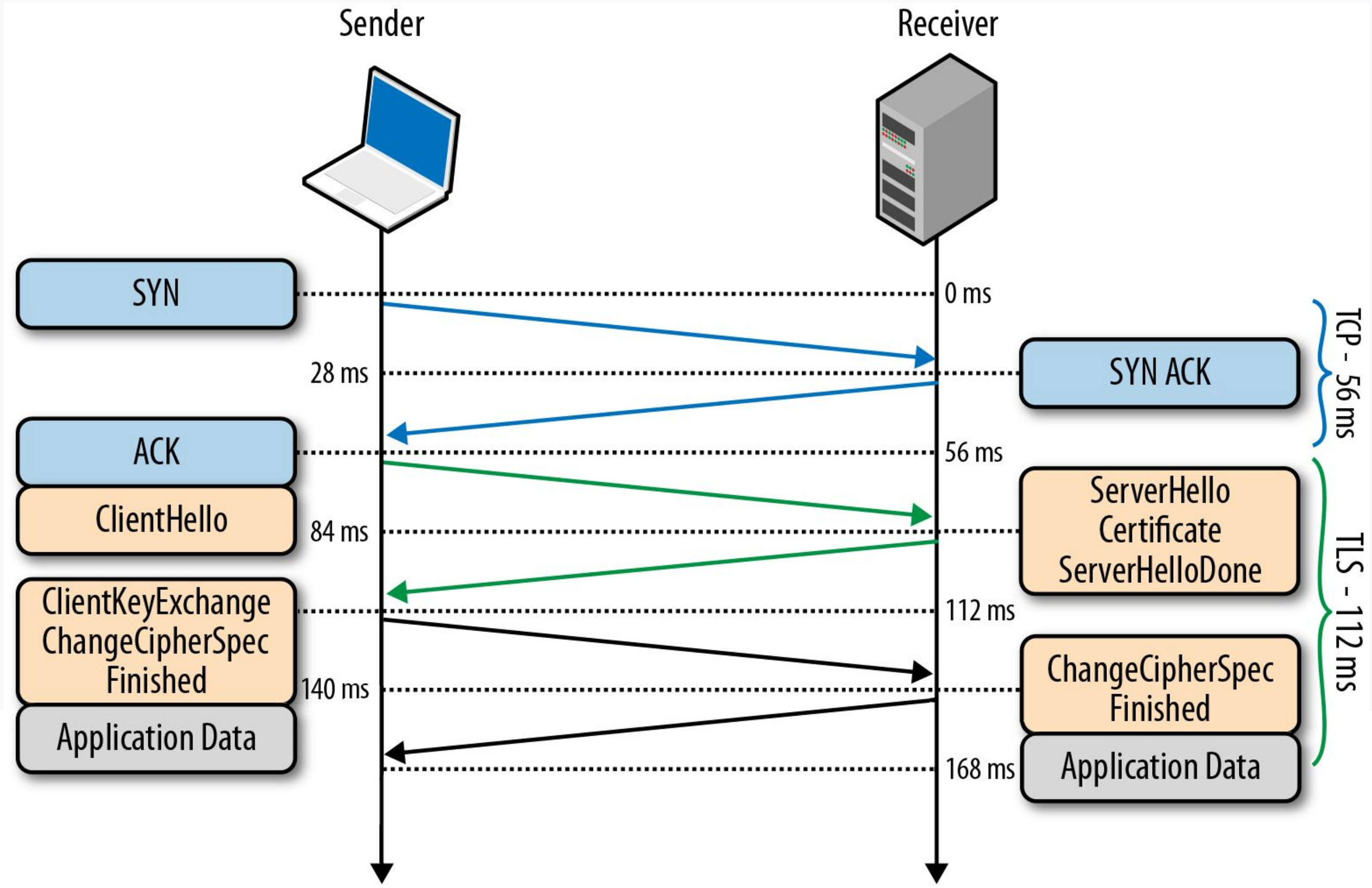
HTTPS

Step	Client	Direction	Message	Direction	Server
1			Client Hello	→	
2		←	Server Hello		
3		←	Certificate		
4		←	Server Key Exchange		
5		←	Server Hello Done		
6			Client Key Exchange	→	
7			Change Cipher Spec	→	
8			Finished	→	
9		←	Change Cipher Spec		
10		←	Finished		

HTTPS

Step	Client	Direction	Message	Direction	Server
1			Client Hello Supported Cipher Suites Guesses Key Agreement Protocol Key Share	→	
2		←	Server Hello Key Agreement Protocol Key Share Server Finished		
3			Checks Certificate Generates Keys Client Finished	→	

HTTPS


Sender
Receiver
SYN
0 ms
28 ms
SYN ACK
TCP - 56 ms
ACK
56 ms
ClientHello
84 ms
ServerHello
Certificate
ServerHelloDone
TLS - 112 ms
ClientKeyExchange
ChangeCipherSpec
Finished
112 ms
140 ms
ChangeCipherSpec
Finished
Application Data
168 ms
Application Data

Ваши вопросы?

Маршрут вебинара

Протокол HTTP

REST API

HTTPS

Веб-серверы

Apache

Apache

Apache HTTP-сервер (назван именем группы племён североамериканских индейцев апачей; кроме того, является искажённым сокращением от англ. a patchy server; среди русских пользователей общепринято переводное апа́ч) - свободный веб-сервер

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

Особенности:

- первый релиз в 1995 г.
- архитектурно выполнен из ядра и модулей
- ядро написано на C
- кроссплатформенный

Apache

Конфигурация:

Имеет три условных уровня конфигурации:

- конфигурация сервера (**httpd.conf**)
- конфигурация виртуального хоста (httpd.conf с версии 2.2, **extra/httpd-vhosts.conf**)
- конфигурация уровня каталога (**.htaccess**)

Большая часть модулей имеет свои собственные параметры

NGINX

NGINX

nginx (engine x — англ. Engine X) - веб-сервер и почтовый прокси-сервер, работающий на Unix-подобных операционных системах.

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

Особенности:

- первый релиз в 2004 г.
- архитектурно состоит из мастер-процесса, который вызывает дочерние процессы (воркеры, загрузчик кэша и кэш менеджер)
- использует бесконечный цикл для обработки запросов клиентов
- так как основное назначение - reverse-proxy, из коробки умеет разные алгоритмы балансировки между бэкендами

Apache VS NGINX

Apache VS NGINX

Основные отличия:

1. **Метод обработки соединений**
   - apache - создает на каждый запрос отдельный процесс или поток
   - nginx - мастер-процесс один, он создает дочерние процессы и удаляет их после закрытия соединения
2. **Контент**
   - apache отдает и статический и динамический контент
   - nginx отдает только статический контент
3. **Работа с модулями**
   - apache - модули подключаются динамически, без перезагрузки
   - nginx - модули необходимо собирать, загрузка на лету не поддерживается
4. **Интерпретация запросов**
   - apache умеет интерпретировать запросы как как ресурс в файловой системе или как URI
   - nginx в основном работает с URI и транслирует запросы при необходимости

Apache VS NGINX

Основные отличия:

5. Работа со скриптовыми языками

- apache имеет один модуль (например mod_php) и все хосты будут работать с одной и той же версием php с одним конфигом
- nginx позволяет использовать разные версии (например php) на разных виртуальных хостах

6. Скорость работы

- есть тесты, которые показывают, что статический контент примерно в 2.5 раза быстрее отдает nginx
- динамический контент примерно одинаково отдают оба сервера

7. Поддержка ОС

- apache полностью поддерживает и UNIX-системы и ОС Windows
- nginx в основном собран для UNIX-систем, но есть сборка и под Windows

Ускорение TLS handshake

Ускорение TLS handshake

Пример настройки SSL (TLS) в apache:

```
<VirtualHost 192.168.0.1:443>
DocumentRoot /var/www/site
ServerName www.site.com
SSLEngine on
SSLCertificateFile /etc/ssl/crt/cert.crt
SSLCertificateKeyFile /etc/ssl/crt/private.key
SSLCertificateChainFile /etc/ssl/crt/chain.crt
</VirtualHost>
```

Ускорение TLS handshake

Пример настройки SSL (TLS) в nginx:

```
server {
    listen                  443 ssl;
    server_name             example.com;
    ssl_certificate         example.com.crt;
    ssl_certificate_key     example.com.key;
    ssl_protocols           TLSv1.1 TLSv1.2;
    ssl_ciphers             AES128-SHA:AES256-SHA:RC4-SHA:DES-CBC3-SHA:RC4-MD5;
    ...
}
```

Ускорение TLS handshake

Варианты оптимизации сессии и ускорения TLS:

- **постоянные соединения (keepalive)** - дают возможность в условиях одного подключения обрабатывать сразу несколько запросов

```
server {

...
keepalive_timeout   70;

…

}
```

Ускорение TLS handshake

Варианты оптимизации сессии и ускорения TLS:

- **кэширование** - необходимо для повторного использования ключей, чтобы не повторять хэндшейк

```
http {
    ssl_session_cache   shared:SSL:100m;
    ssl_session_timeout 1h;
...
}
```

Ускорение TLS handshake

Варианты оптимизации сессии и ускорения TLS:

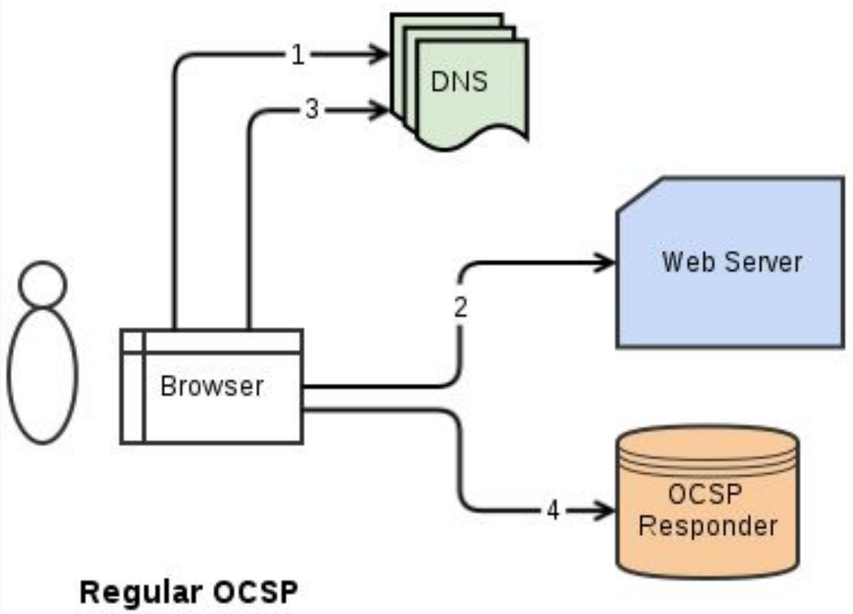
- **статические ключи (session tickets)** - возобновление сессии клиентом и сервером по зашифрованным ключам сессии

```
server {

...
ssl_session_ticket_key current.key;
ssl_session_ticket_key prev.key;
ssl_session_ticket_key prevprev.key;

...
}
```

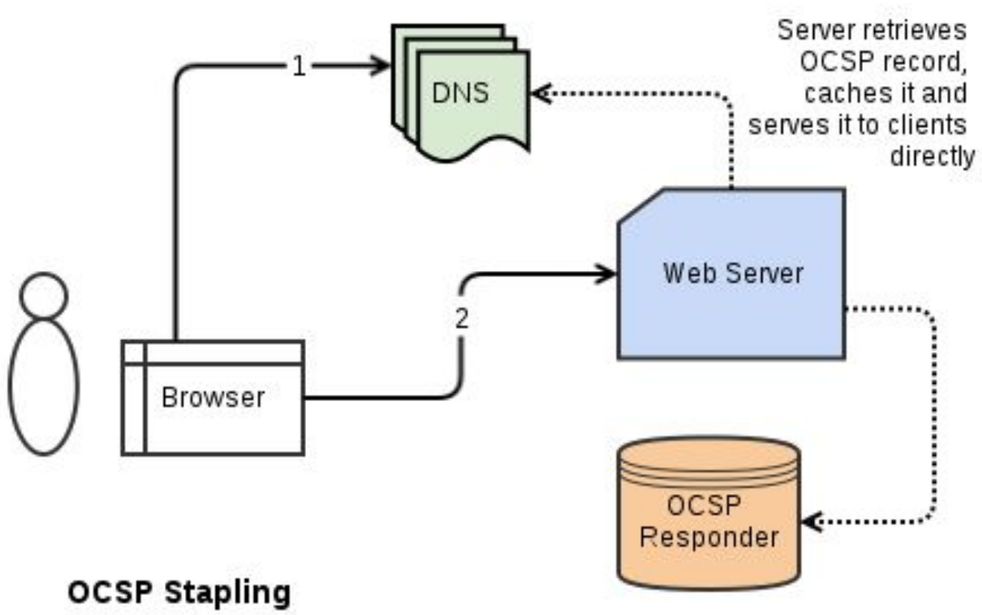
Ускорение TLS handshake

Варианты оптимизации сессии и ускорения TLS:

- **OCSP stapling (Online Certificate Status Protocol)** - механизм проверки актуальности сертификата SSL



Regular OCSP



OCSP Stapling

Ускорение TLS handshake

Варианты оптимизации сессии и ускорения TLS:

- **OCSP stapling (Online Certificate Status Protocol)** - опрос и кэширование ответа сервером, передача ответа клиенту в TLS handshake

```
server {

...
ssl_stapling on;
ssl_stapling_verify on;
ssl_trusted_certificate /etc/nginx/ssl/example.com.crtt;
...
}
```

Сжатие контента

Сжатие контента

Сжатие (архивирование) - вариант уменьшения объема трафика и увеличения скорости доставки веб-контента или веб-приложения

Различают три основные группы алгоритмов сжатия (архивирования):

1. **Поточные алгоритмы**. К этой группе относятся алгоритмы семейств RLE (run-length encoding), LZ* и др. Например, **gzip** (LZ77), **bzip** и **compress**.

2. **Алгоритмы статистического (энтропийного) сжатия**. Алгоритм сжатия данных **brotli** использует, в том числе, кодирование Хаффмана совместно с LZ-алгоритмами.

3. **Разностные алгоритмы**. В отдельную группу можно выделить алгоритмы преобразования информации (включая использование словарей).Алгоритм **SDCH** (VCDIFF) использует именно словари и разностное кодирование информации.

Сжатие контента

Какой контент лучше сжимать?

- HTML
- XML
- CSS
- Javascript
- txt
- другие текстовые файлы

Как понять, поддерживает ли сжатие клиент?

Посмотреть наличие заголовка **accept-encoding**:

accept-encoding: gzip, deflate, sdch, br

Сжатие контента

Включение сжатия в конфиге apache:

```
<ifmodule mod_deflate.c>
AddOutputFilterByType DEFLATE text/html text/css text/javascript application/javascript
DeflateCompressionLevel 7
</ifmodule>
```

Сжатие контента

Включение сжатия в конфиге nginx:

```
server {
    ....
    gzip on;
    gzip_comp_level 5;
    gzip_types text/plain text/css application/json application/x-javascript text/xml
application/xml application/xml+rss text/javascript application/javascript;
}
```

Ваши вопросы?

Рефлексия

Назовите 3 момента, которые вам запомнились в процессе занятия

Что вы будете применять в работе из сегодняшнего вебинара?

Заполните, пожалуйста,
опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Викирюк Павел

Системный инженер