

3. Протокол HTTP и веб-API

Сухорослов Олег Викторович

21.09.2026

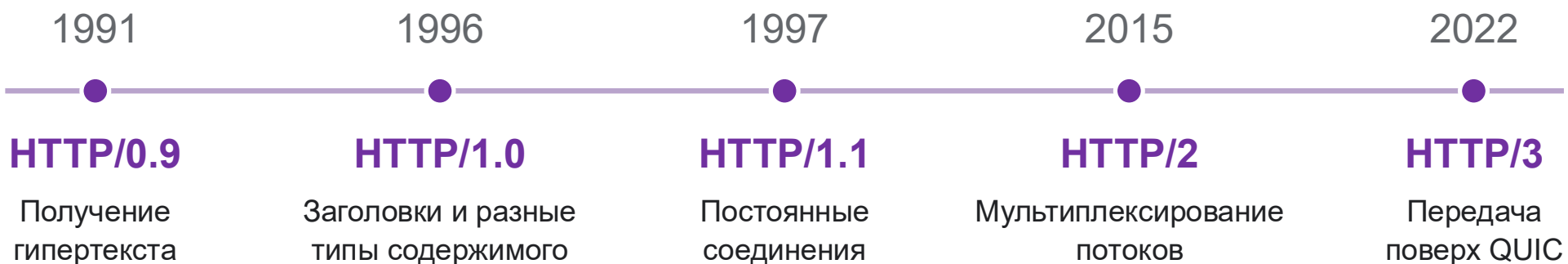
Взаимодействие процессов в РС

- **Завершение отправки** ещё не означает выполнение операции
- **Отсутствие ответа** может оставлять результат операции неизвестным
- **Повторы запросов** требуют учёта семантики операции и обработки дубликатов
- **Ожидание ответов** влияет на производительность взаимодействия

HTTP: назначение и развитие

HTTP – Hypertext Transfer Protocol, протокол передачи гипертекста.

- Изначально HTTP служил для получения гипертекстовых документов в WWW.
- Общий интерфейс ресурсов и разные форматы данных расширили его применение.
- Сегодня HTTP используют и для загрузки веб-страниц, и для взаимодействия приложений через API.



Создание заказа: RPC и HTTP

Клиент передаёт товар и количество, сервер создаёт заказ

RPC

Вызов:

```
orders.createOrder("book", 1)
```

Результат:

42

HTTP

Запрос:

```
POST /orders
```

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

```
{"item": "book", "quantity": 1}
```

Ответ:

```
201 Created
```

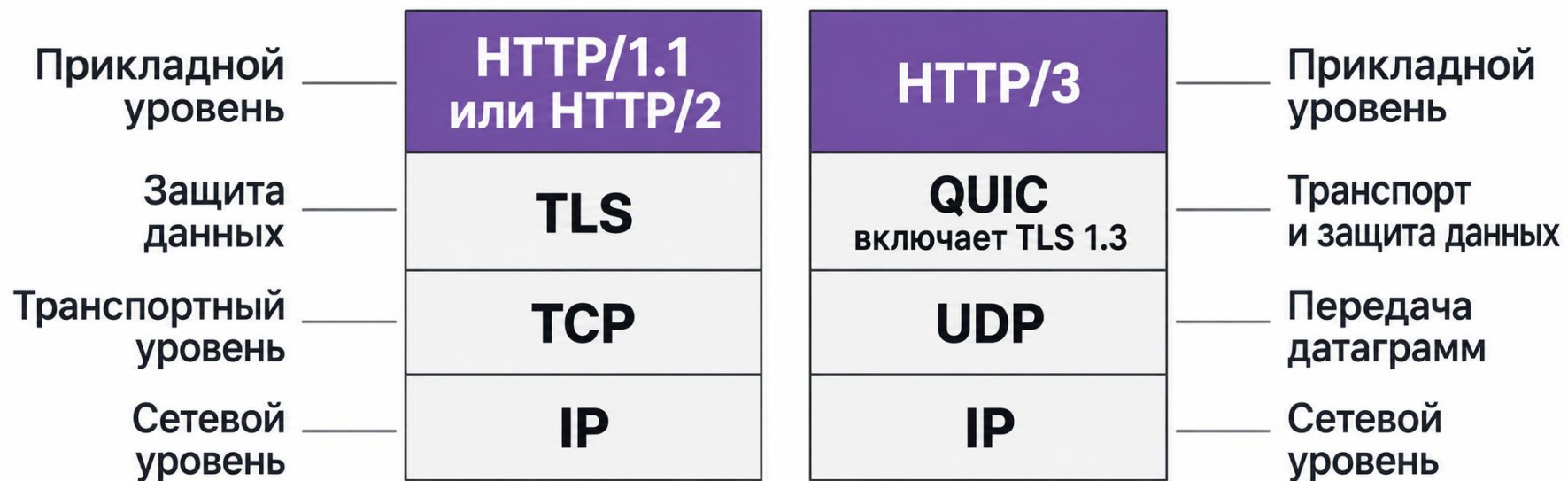
```
Location: /orders/42
```

План лекции

- **Модель и семантика HTTP:** запросы, ответы, методы и повторы
- **HTTP/1.1:** соединения, задержки и конвейерная передача
- **HTTP/2:** мультиплексирование запросов
- **HTTP/3 и QUIC:** взаимодействие HTTP и транспорта
- **Проектирование API:** REST и сравнение с RPC

HTTP в стеке протоколов

- **HTTP** – прикладной протокол взаимодействия «запрос–ответ».
- **HTTP/1.1** и **HTTP/2** обычно используют TCP, **HTTP/3** – QUIC поверх UDP.
- **Семантика HTTP сохраняется**, способы передачи сообщений меняются.



Ресурсы, URI и представления

- **Ресурс** – то, к чему адресован HTTP-запрос.
- **URI (Uniform Resource Identifier)** – идентификатор ресурса.
- **Представление** – описание состояния ресурса в определённом формате.

Ресурс: заказ №42

URI: <https://shop.example/orders/42>

JSON	HTML (вид в браузере)
<pre>{ "id": 42, "item": "book", "quantity": 1, "status": "created" }</pre>	Заказ №42 Товар: книга Количество: 1 Статус: создан

У одного ресурса могут быть разные представления.

Структура запроса HTTP/1.1

- **Строка запроса:** метод, адрес ресурса и версия протокола.
- **Заголовки:** параметры запроса и сведения о передаваемых данных.
- **Тело:** данные для обработки сервером; может отсутствовать.

```
POST /orders HTTP/1.1
```

```
Host: shop.example
```

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

```
Accept: application/json
```

```
Content-Length: 31
```

```
{"item": "book", "quantity": 1}
```

Строка
запроса

Заголовки

Конец
заголовков

Тело запроса

Структура ответа HTTP/1.1

- **Строка статуса:** версия протокола, код ответа и текстовое пояснение.
- **Заголовки:** параметры ответа и сведения о передаваемых данных.
- **Тело:** представление ресурса или другие данные; может отсутствовать.

```
HTTP/1.1 201 Created
```

```
Location: /orders/42
```

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

```
Content-Length: 31
```

```
{"id": 42, "status": "created"}
```

Строка
статуса

Заголовки

Конец
заголовков

Тело ответа

Коды ответа HTTP

Код ответа (*status code*) сообщает результат обработки запроса.

Первая цифра определяет класс ответа, конкретный код уточняет результат.

Класс	Значение	Примеры
1xx	Информационные ответы	100 Continue
2xx	Успешные ответы	200 OK, 201 Created, 202 Accepted
3xx	Перенаправление	301 Moved Permanently, 304 Not Modified
4xx	Ошибки клиента	400 Bad Request, 404 Not Found
5xx	Ошибки сервера	500 Internal Server Error, 503 Service Unavailable

- **202:** запрос принят в обработку, операция ещё не завершена.
- **304:** можно использовать сохранённое представление ресурса.

Методы HTTP

Метод задаёт **смысл (семантику) операции** над ресурсом.

Метод	Назначение	Пример
GET	Получить представление ресурса	GET /orders/42
POST	Передать данные ресурсу для обработки	POST /orders (создать заказ)
PUT	Создать или заменить состояние ресурса переданным представлением	PUT /orders/42
PATCH	Применить изменения к ресурсу	PATCH /orders/42 (изменить количество)
DELETE	Удалить ресурс	DELETE /orders/42

Один адрес ресурса может использоваться с разными методами.

Безопасные и идиempotentные методы

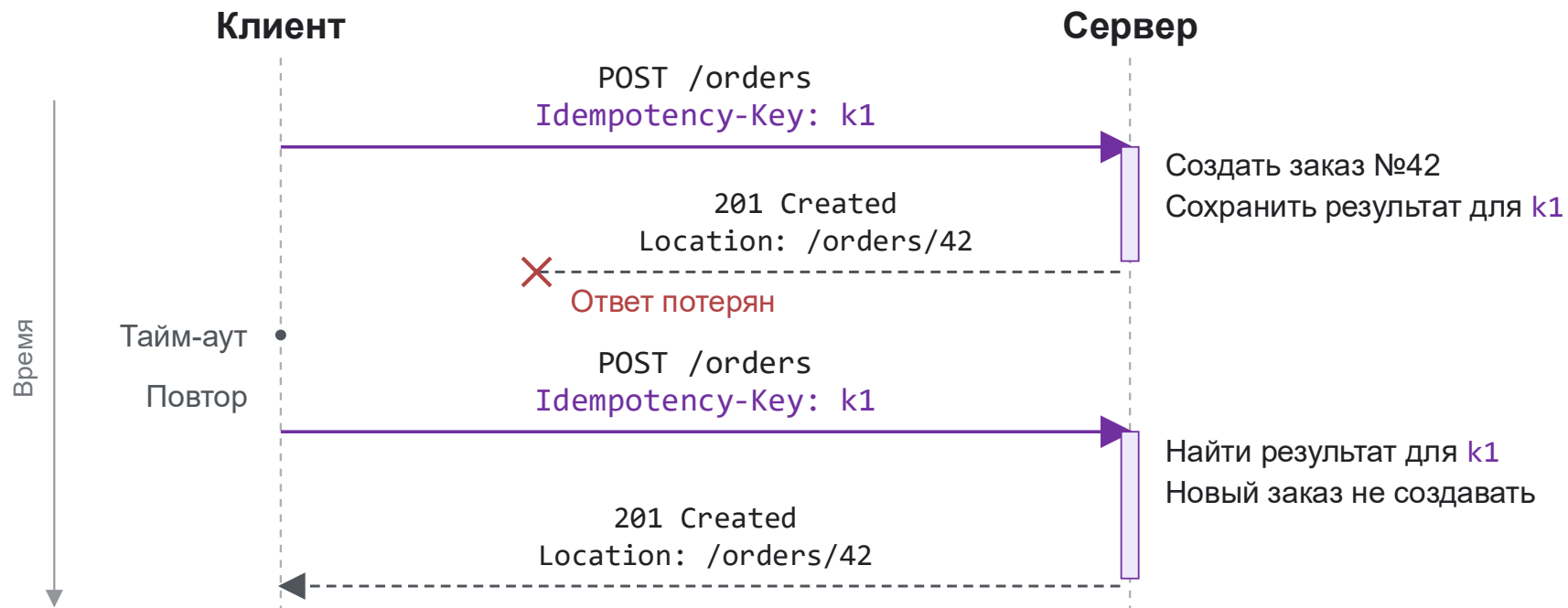
- **Безопасный метод (safe):** клиент не запрашивает изменение состояния сервера.
- **Идиempotentный метод (idempotent):** повтор одинакового запроса имеет тот же ожидаемый эффект, что и однократное выполнение.

Метод	Безопасный	Идиempotentный
GET	Да	Да
POST	Нет	Не гарантируется
PUT	Нет	Да
PATCH	Нет	Не гарантируется
DELETE	Нет	Да

Идиempotentность относится к эффекту операции. Ответы на повторы могут различаться.

Повторные запросы и обработка дубликатов

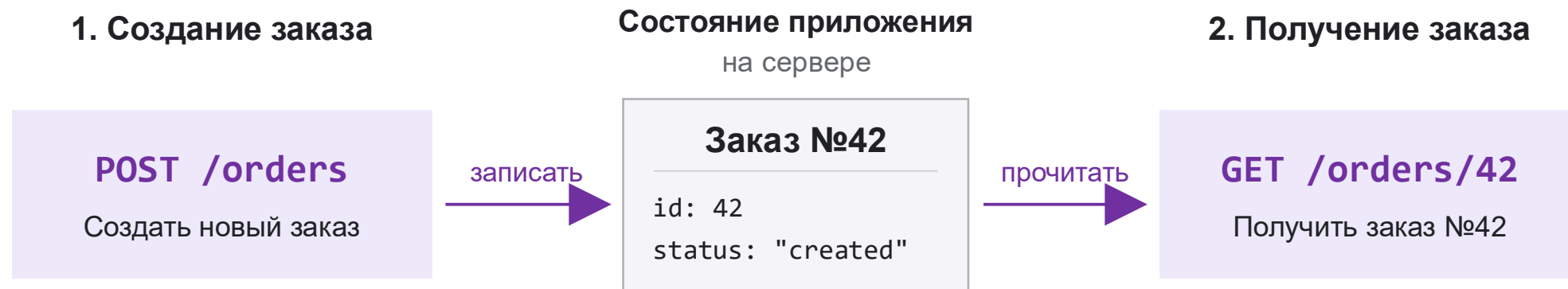
- **Тайм-аут** не позволяет определить, выполнена ли операция.
- **Идемпотентность** позволяет повторить запрос без дополнительного ожидаемого эффекта.
- **Обработка дубликатов** позволяет повторять операции создания: клиент сохраняет ключ запроса, сервер распознаёт повтор.



Пример API с поддержкой Idempotency-Key. Тело запроса при повторе не меняется.

HTTP – протокол без состояния

- **Каждый запрос имеет самостоятельный смысл:**
его интерпретация не зависит от истории сообщений в соединении.
- Соединение не является пользовательской сессией.
- Сервер может хранить состояние ресурсов и приложения.

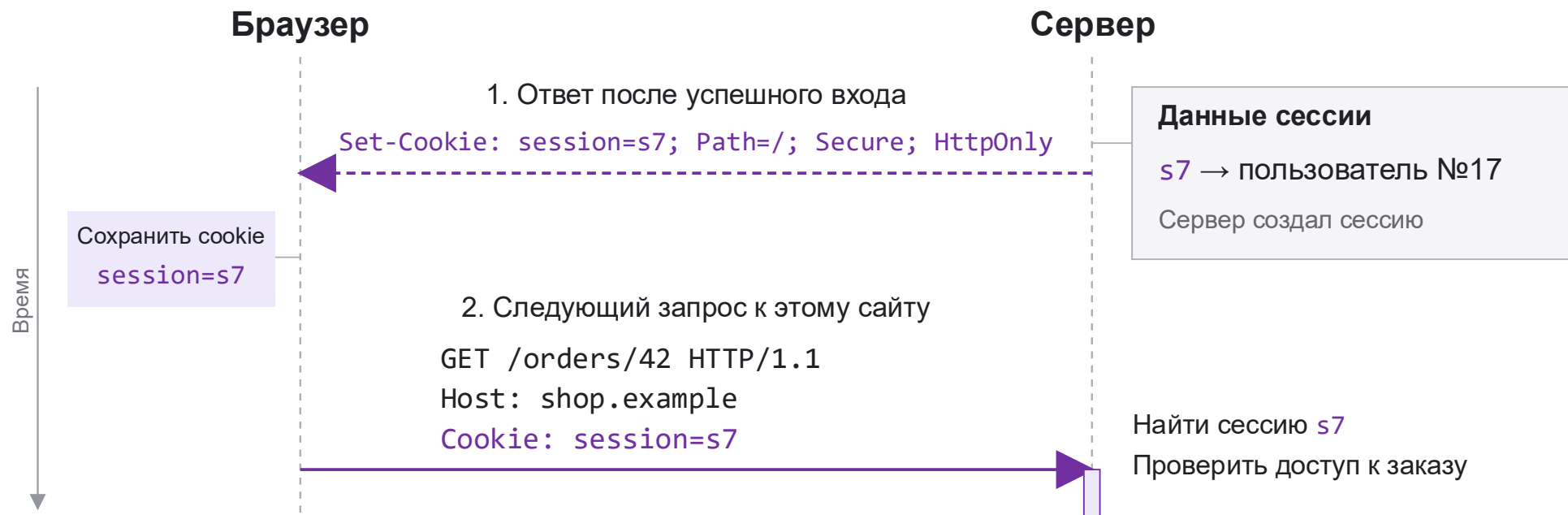


Без состояния протокола ≠ без состояния приложения

Запросы показаны сокращённо. Стрелки обозначают запись и чтение данных.

Cookies и состояние приложения

- **Cookie** – пара «имя–значение», которую браузер хранит и отправляет в подходящих запросах.
- **Set-Cookie в ответе** устанавливает cookie; **Cookie в запросе** передаёт её серверу.
- **Идентификатор сессии в cookie** позволяет связать запрос с состоянием приложения на сервере.



В этой схеме браузер хранит идентификатор, сервер — данные сессии

Сообщения показаны сокращённо. s7 — условный идентификатор. Обмен идёт по HTTPS.

Кеширование HTTP-ответов

- Кеш хранит ответы для повторного использования.
- Свежий ответ можно использовать без запроса к серверу.
- Устаревший ответ можно проверить условным запросом.

Первый ответ на GET /orders/42

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=60
ETag: "order42-v1"

{"id": 42, "status": "created"}
```

сохранить
→

Кеш браузера

Тело ответа
и заголовки

1. Ответ ещё свежий

Возраст ответа меньше 60 секунд

Использовать сохранённый ответ
без обращения к серверу

2. Срок свежести истёк

```
GET /orders/42 HTTP/1.1
If-None-Match: "order42-v1"
```

Если представление не изменилось:

```
HTTP/1.1 304 Not Modified
```

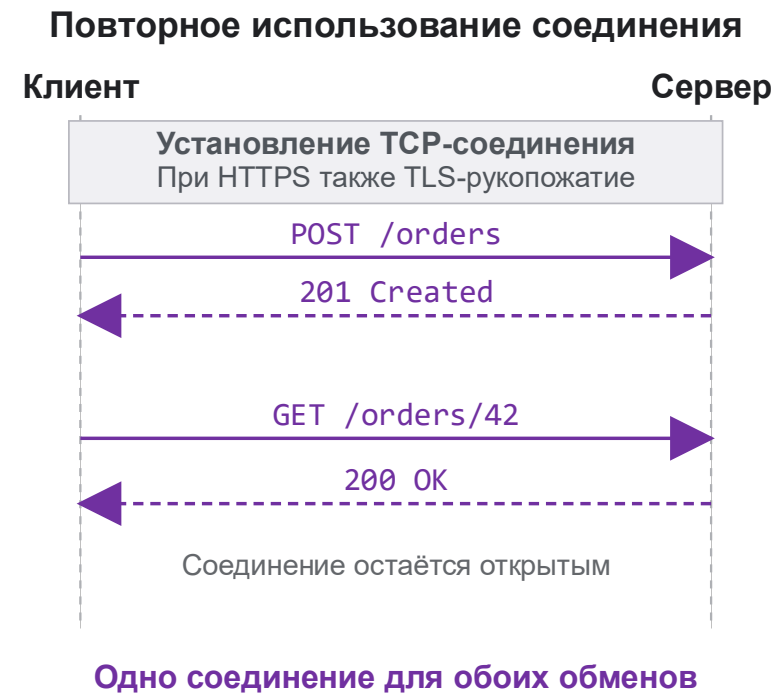
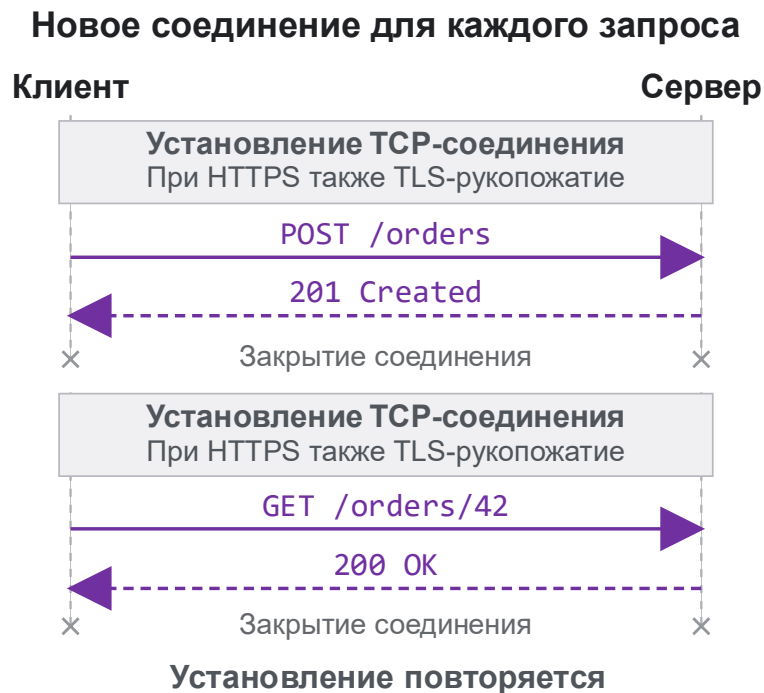
Без тела. Использовать сохранённое тело.

Свежесть экономит запросы. Проверка изменений экономит передачу тела.

Сообщения показаны сокращённо. Рассматривается обычное повторное обращение, допускающее использование кеша.

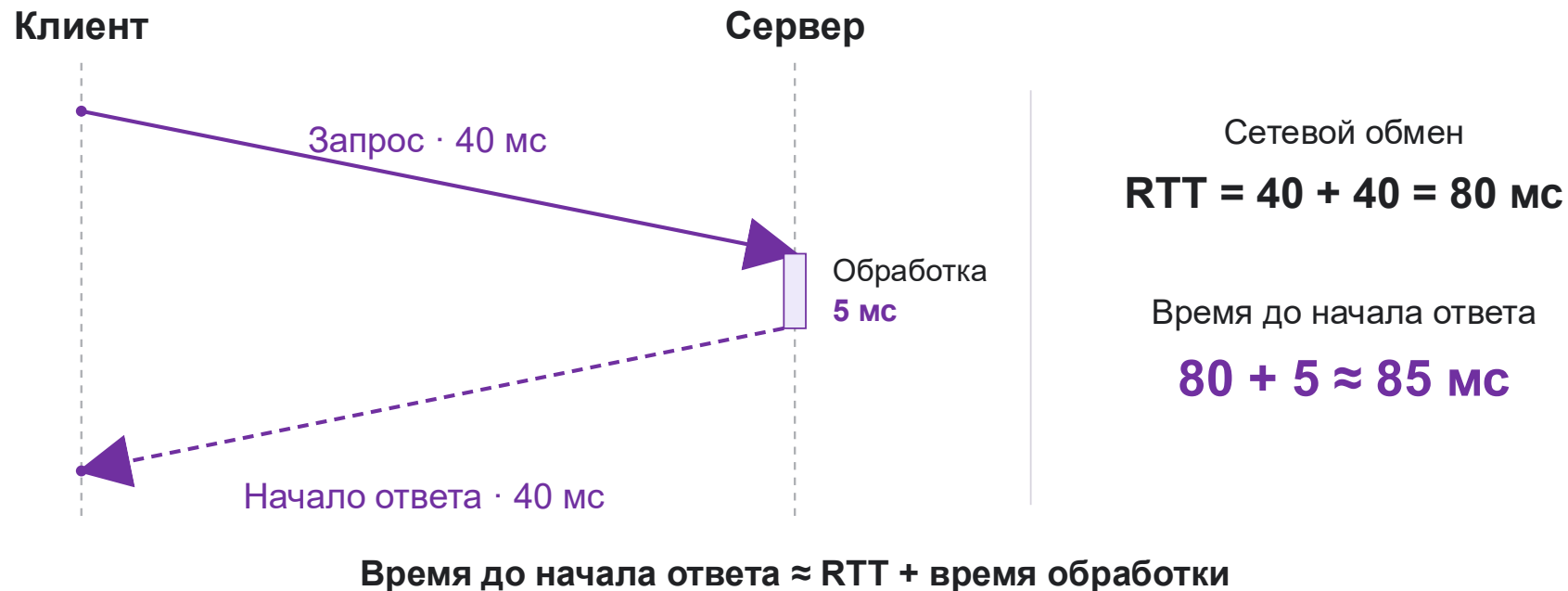
Соединения в HTTP/1.1

- Перед обменом создаётся **TCP-соединение**, а при HTTPS дополнительно выполняется **TLS-рукопожатие**.
- В HTTP/1.1 соединение **по умолчанию сохраняется** для следующих запросов и ответов.
- **Повторное использование соединения** сокращает затраты на его установление.



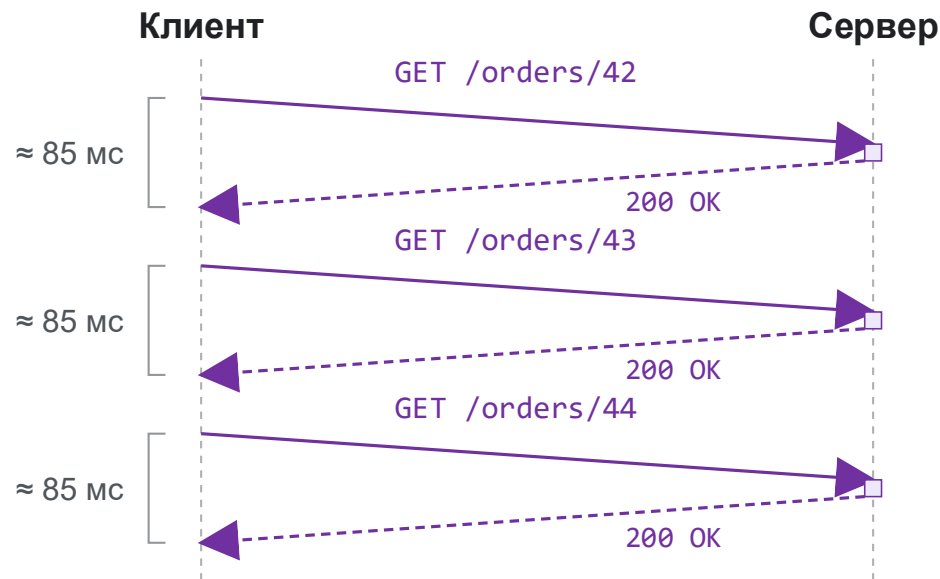
RTT и задержка HTTP-запросов

- **RTT (Round-Trip Time)** – время прохождения данных по сети до другой стороны и обратно.
- На установленном соединении **ожидание начала ответа** складывается из RTT и времени обработки на сервере.



Последовательные запросы

- При **последовательном обмене** следующий запрос отправляется после полного получения предыдущего ответа.
- Для каждого запроса повторяется сетевой обмен, поэтому **задержки складываются**.
- Постоянное соединение экономит время установления, но **не устраняет ожидание между запросами**.



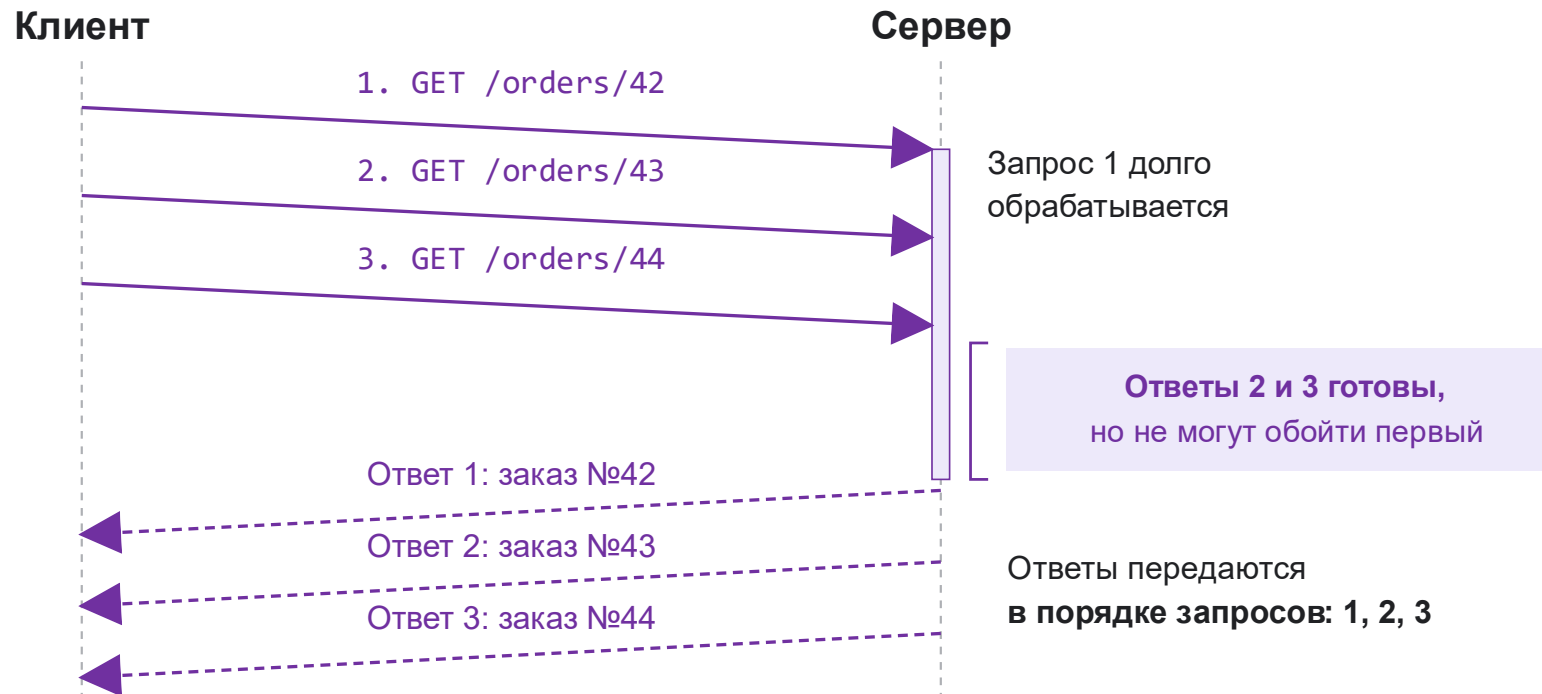
RTT = 80 мс
Обработка = 5 мс
на каждый запрос

Три последовательных запроса
 $3 \times (80 + 5) \approx 255 \text{ мс}$

Одно соединение, но ожидание ответа повторяется для каждого запроса

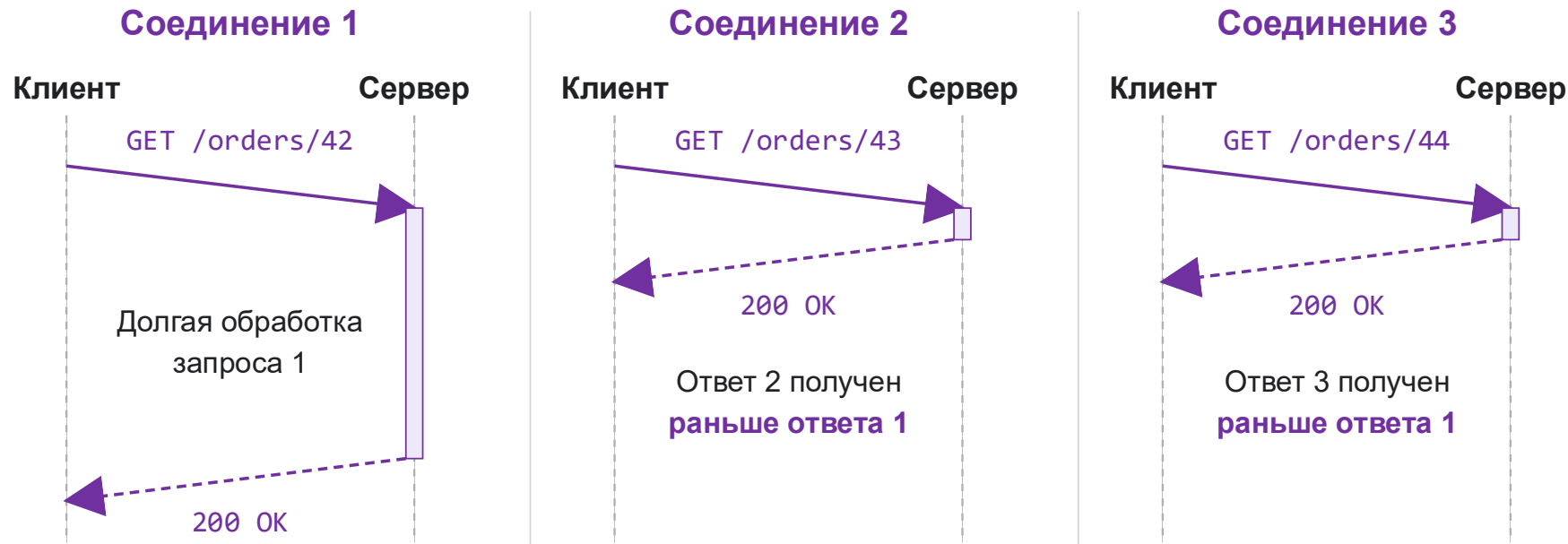
Конвейеризация и блокировка очереди

- **Конвейеризация (pipelining)** в HTTP/1.1 позволяет отправить несколько запросов по одному соединению, не дожидаясь ответов.
- Сервер должен передавать ответы **в порядке получения запросов**.
- Медленный первый ответ **задерживает следующие**, возникает т.н. **head-of-line blocking**.



Параллельные соединения

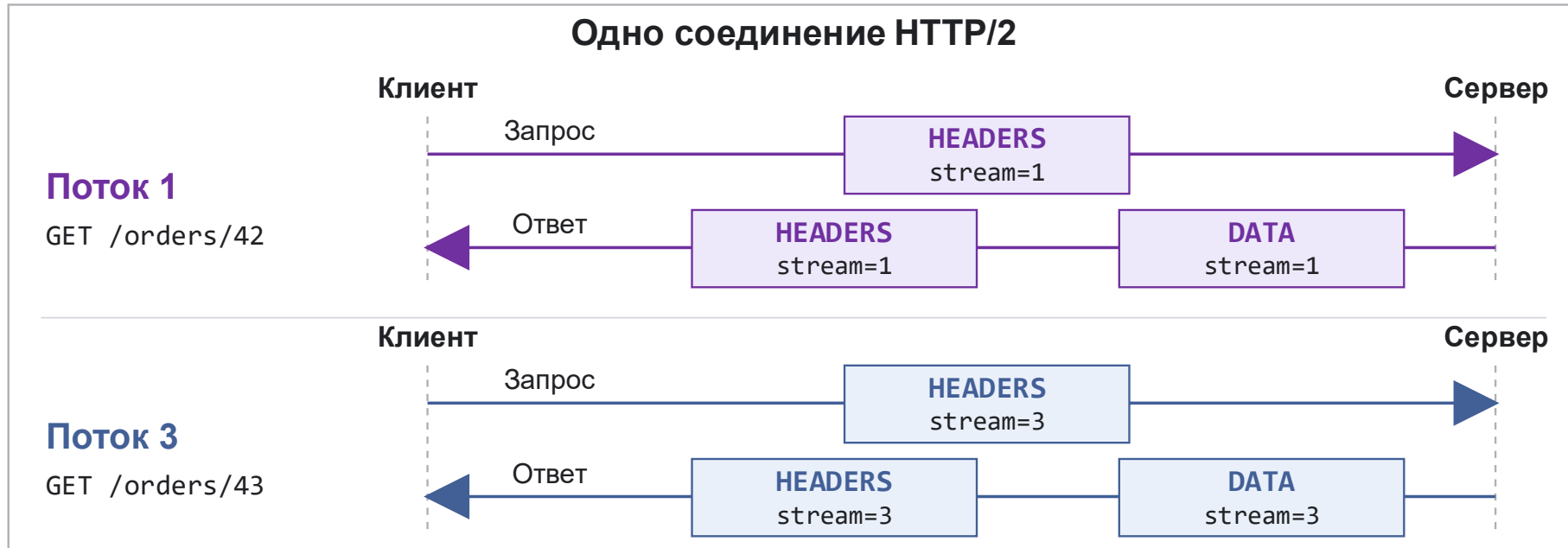
- **Несколько соединений** позволяют выполнять запросы параллельно: ответы на разных соединениях не обязаны ждать друг друга.
- **Каждое соединение требует ресурсов** и затрат на установление TCP и TLS при HTTPS.
- **Число соединений ограничивают**: при занятости всех соединений новые запросы ожидают.



На разных соединениях ответы 2 и 3 не обязаны ждать ответа 1

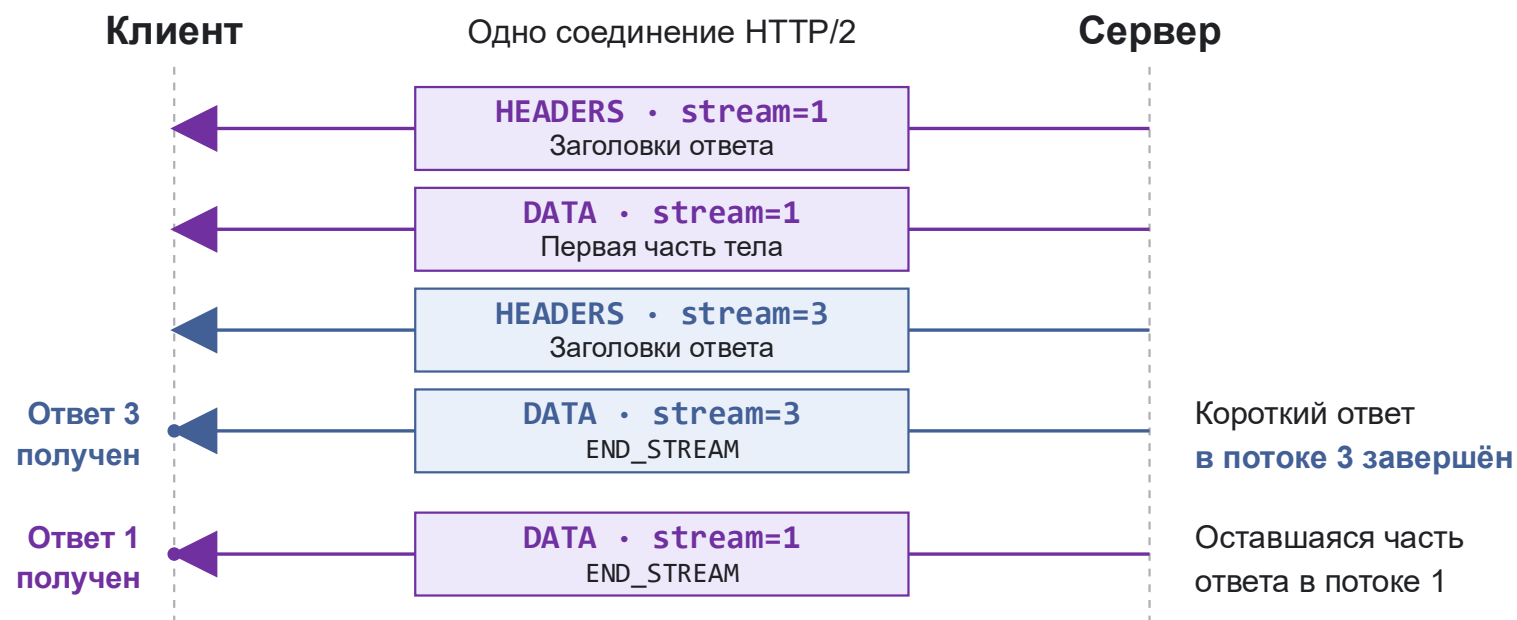
HTTP/2: потоки и кадры

- **Семантика HTTP сохраняется:** методы, коды ответа и ресурсы имеют прежний смысл.
- Каждый обмен «запрос–ответ» использует отдельный **поток (stream)** внутри соединения.
- Сообщения передаются **бинарными кадрами (frames)**: HEADERS – заголовки, DATA – тело. Кадры обмена содержат идентификатор потока.



Мультиплексирование запросов и ответов

- **Кадры разных потоков чередуются** в одном соединении.
- Внутри потока порядок сохраняется, но **ответы разных потоков могут завершаться в разном порядке**.
- Для передачи короткого ответа **не нужно ждать завершения длинного ответа** в другом потоке.



Ответ в потоке 3 завершился раньше ответа в потоке 1

Сжатие заголовков: HPACK

- HPACK сжимает заголовки HTTP/2, но не тело сообщения.
- Поля можно передавать **ссылками на записи таблиц**, вместо повторения имени и значения.
- Используются **статическая таблица** известных полей и **динамическая таблица**, пополняемая при обмене.

Статическая таблица

Заранее определена стандартом

16	accept-encoding: gzip, deflate
19	accept: пустое значение

Динамическая таблица

Пополняется при обмене

До первого запроса: пуста

После первого запроса:

62	accept: application/json
----	--------------------------

1. Первый запрос

Индекс 16

accept-encoding: gzip, deflate

Имя: индекс 19

Значение: application/json

Добавить пару в динамическую таблицу

2. Следующий запрос

Индекс 16

accept-encoding: gzip, deflate

Индекс 62

accept: application/json

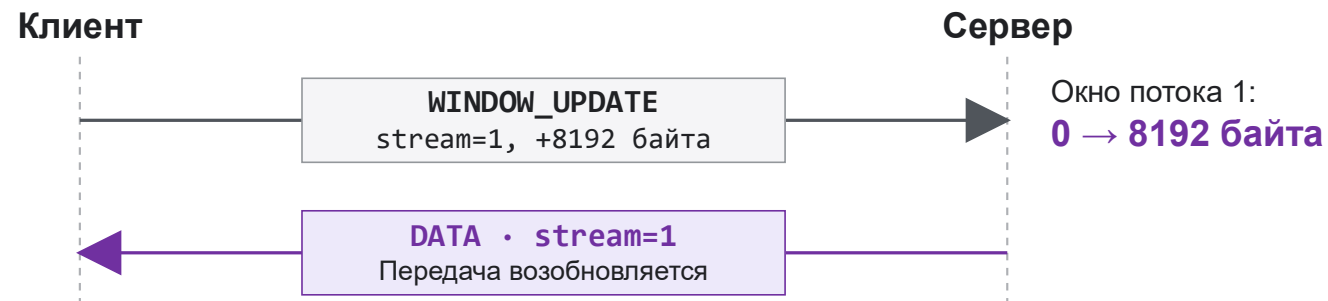
Во втором запросе обе пары передаются индексами

Управление потоком данных в HTTP/2

- **Получатель задаёт окна передачи** – сколько данных отправитель ещё может передать.
- Действуют два ограничения: **окно отдельного потока** и **общее окно соединения**. Передача DATA уменьшает оба.
- Кадр **WINDOW_UPDATE** увеличивает окно, разрешая передать дополнительные данные.

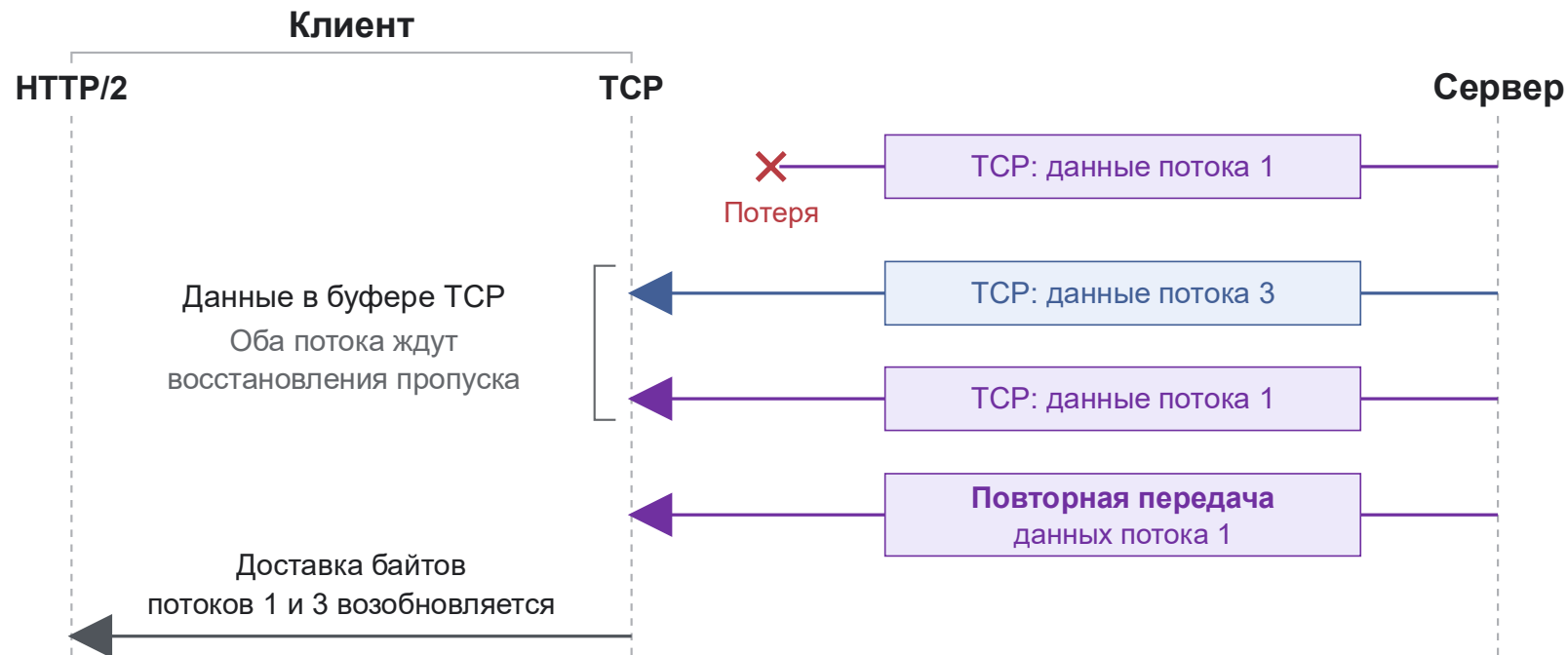
Окна для передачи данных от сервера к клиенту

До обновления	Доступно, байт	Передача данных
Соединение	16 384	Общий запас для потоков
Поток 1	0	Приостановлена
Поток 3	8192	Возможна



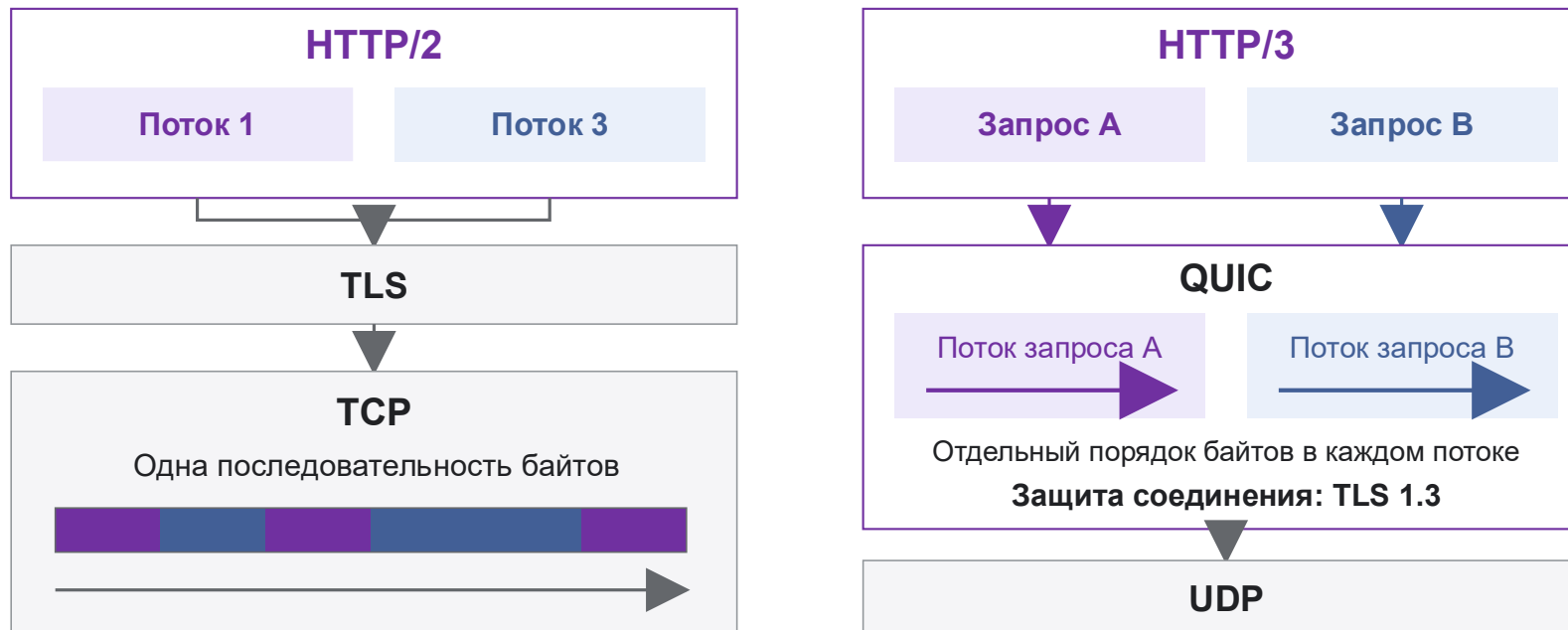
HTTP/2: блокировка на уровне TCP

- Потоки HTTP/2 используют **одно TCP-соединение** с упорядоченной доставкой байтов.
- При потере пакета TCP задерживает доставку последующих байтов до восстановления пропуска (тот же **head-of-line blocking**, но на уровне транспорта).
- Потеря внутри соединения может задержать несколько потоков HTTP/2, включая потоки, чьи данные дошли без потерь.



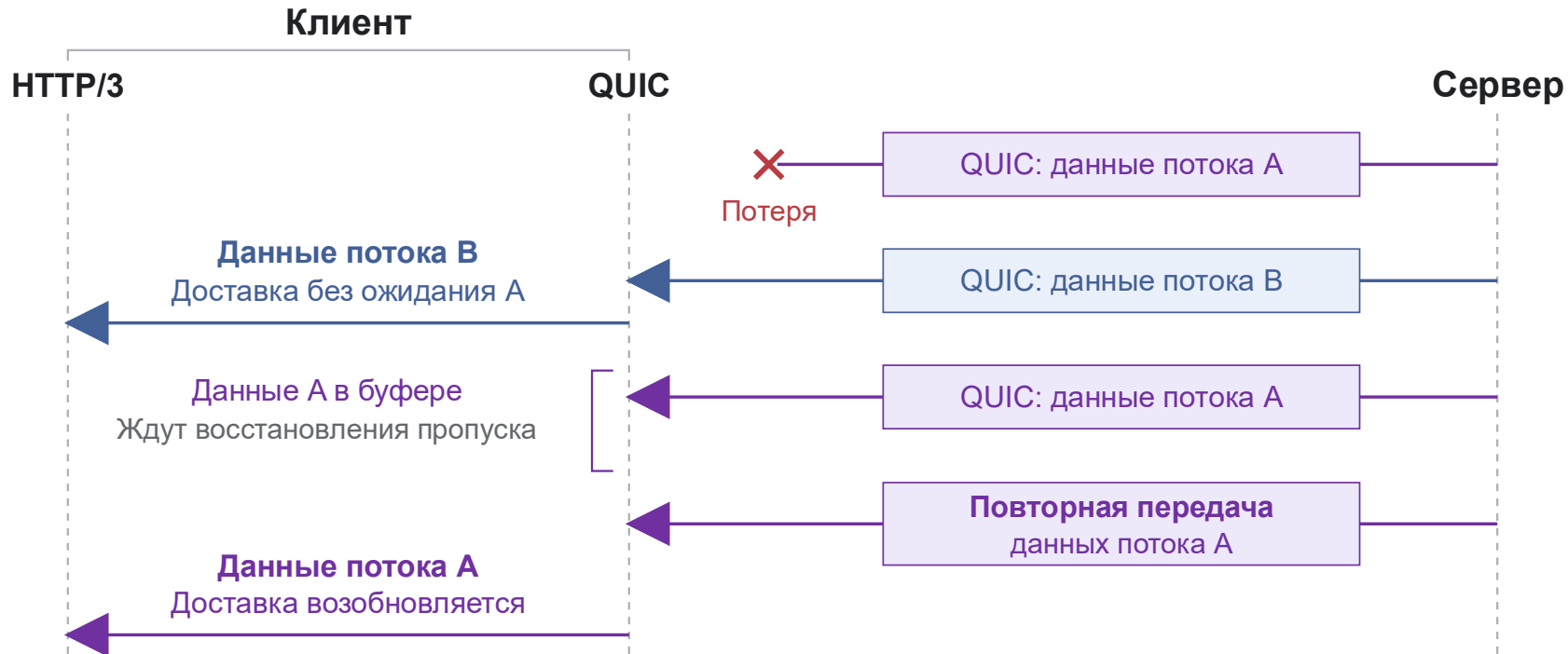
HTTP/3: транспорт QUIC вместо TCP

- HTTP/3 использует **QUIC поверх UDP** вместо TCP, сохраняя семантику HTTP.
- QUIC обеспечивает надёжную доставку и порядок байтов внутри каждого потока.
- Мультиплексирование потоков реализовано в QUIC, а защита соединения использует TLS 1.3.



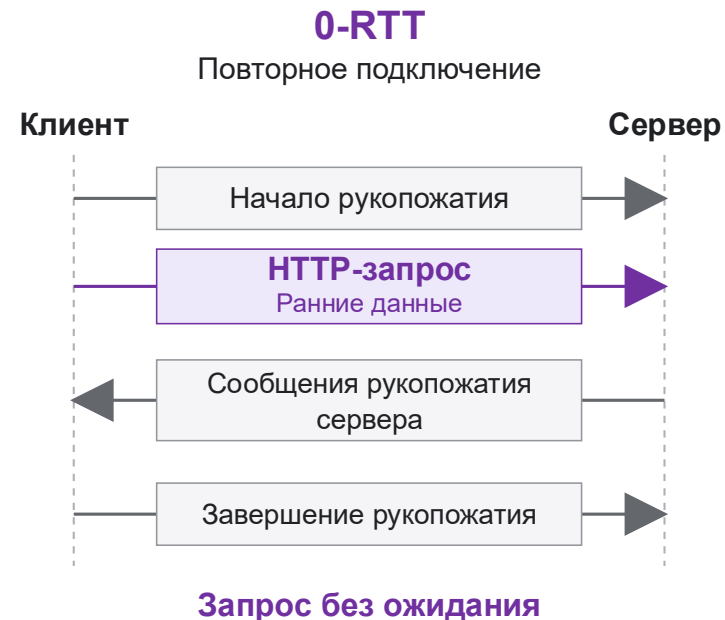
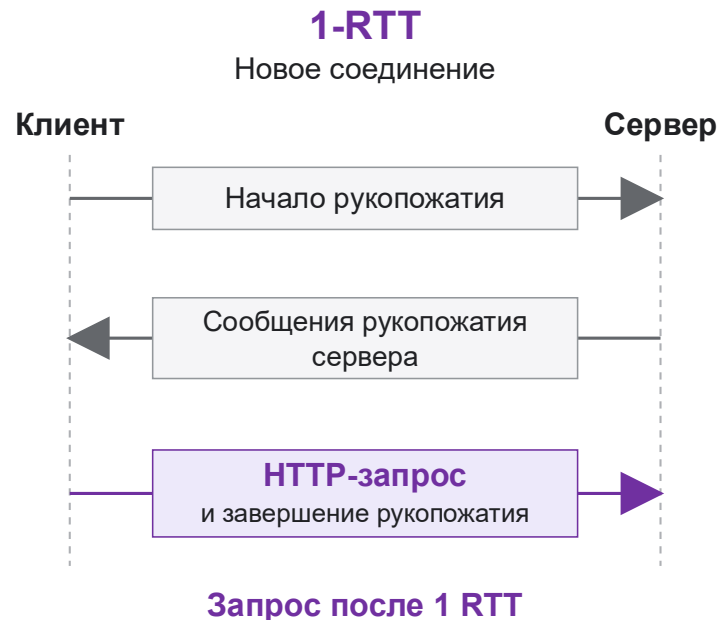
Независимая доставка данных в потоках QUIC

- QUIC восстанавливает порядок байтов **отдельно в каждом потоке**.
- Пропуск в одном потоке **не мешает** доставлять полученные данные другого.
- Внутри потока данные после пропуска ждут его восстановления.



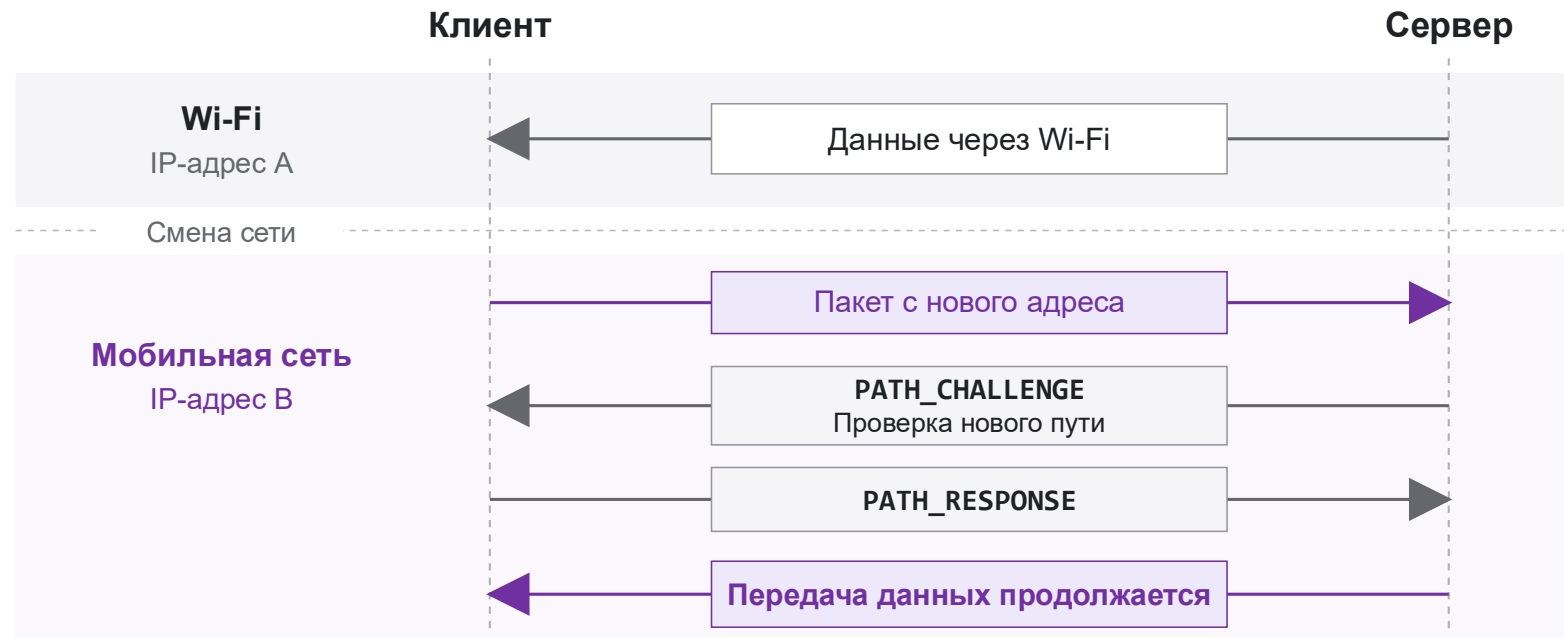
Установление соединения: 1-RTT и 0-RTT

- **1-RTT:** при новом соединении клиент обычно может отправить HTTP-запрос после одного обмена с сервером.
- **0-RTT:** при повторном подключении клиент может отправить запрос сразу, используя сохранённые параметры.
- **Злоумышленник может повторно отправить перехваченные 0-RTT-данные.** Поэтому ранняя отправка подходит только для запросов, повтор которых допустим.



Миграция соединения при смене сети

- QUIC позволяет сохранить соединение при смене IP-адреса или порта клиента.
- **Connection ID** связывает пакеты с соединением независимо от сетевого адреса.
- Новый сетевой путь проходит проверку, после чего потоки могут продолжить работу без нового рукопожатия.



То же соединение QUIC

HTTP/3: преимущества и ограничения

Преимущества:

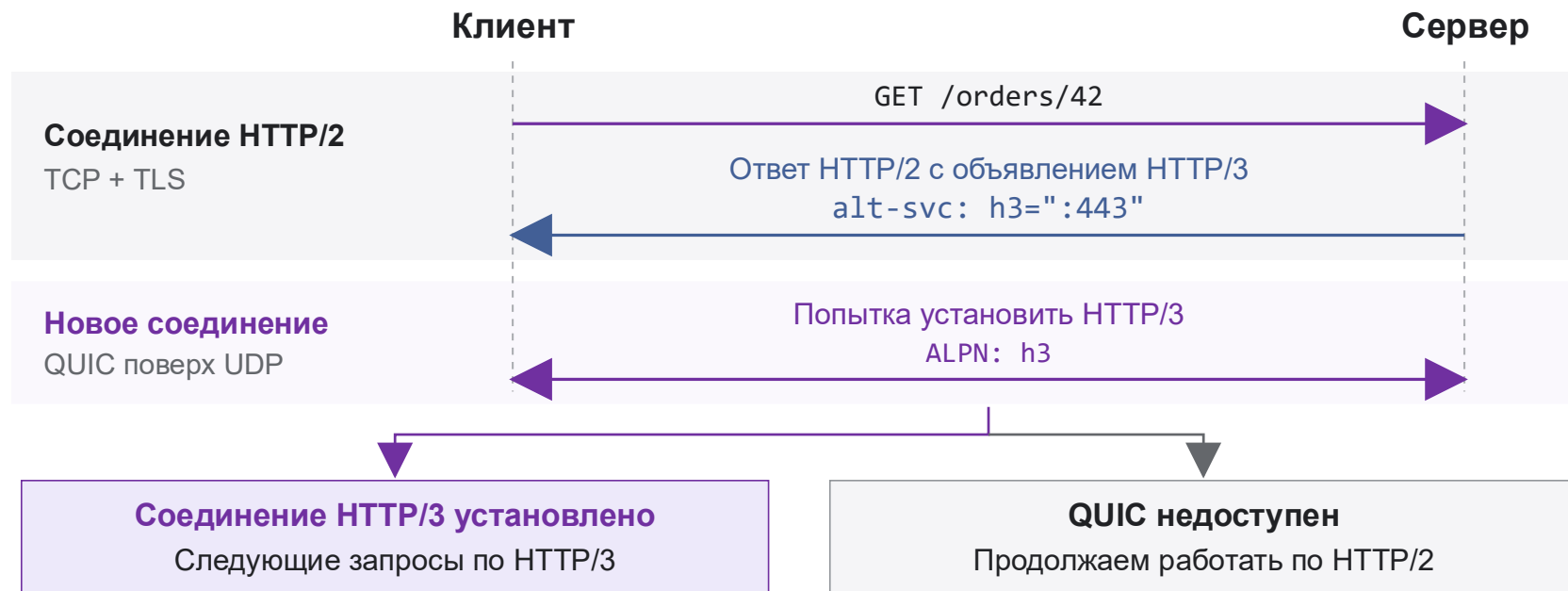
- Потеря данных одного потока **не блокирует доставку** других потоков.
- 1-RTT и 0-RTT **сокращают ожидание** перед отправкой запроса.
- Миграция позволяет **сохранить соединение** при смене сети.

Ограничения:

- Потоки разделяют пропускную способность и управление перегрузкой.
- 0-RTT требует учитывать последствия повторного выполнения запроса.
- При недоступности UDP нужен переход на HTTP/2 или HTTP/1.1.

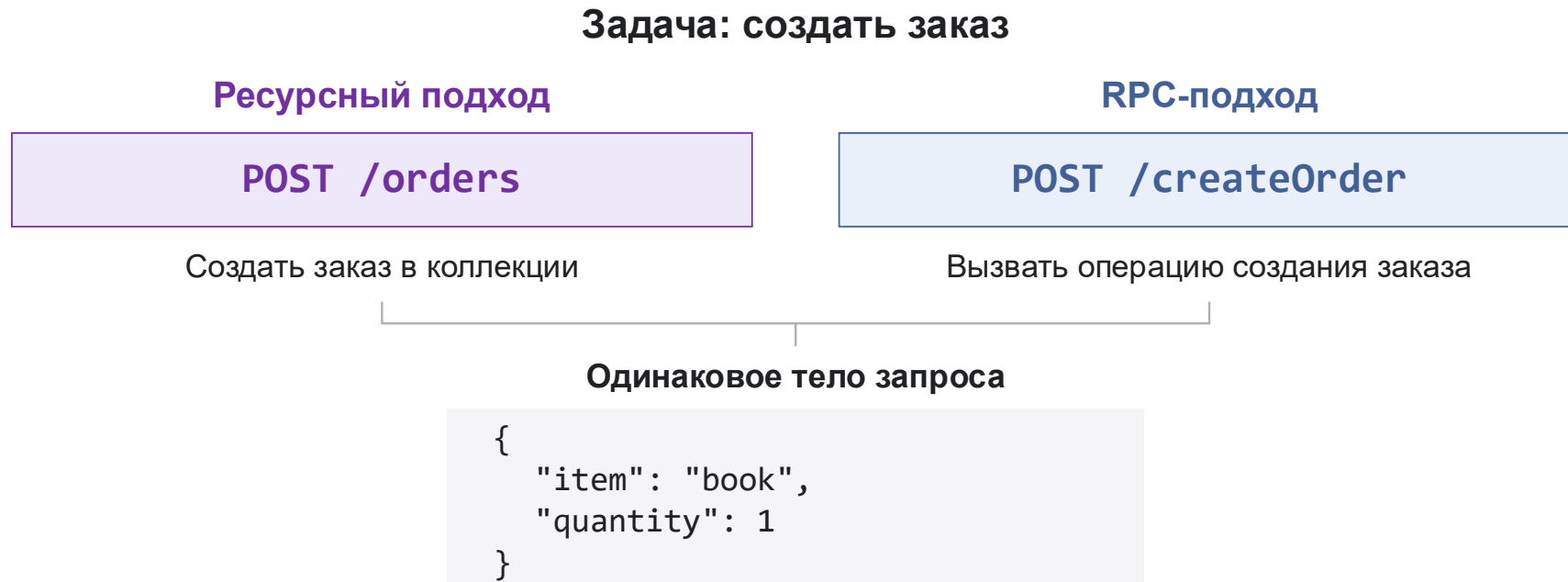
Как выбирается версия HTTP

- Для HTTPS версия HTTP/1.1 или HTTP/2 согласуется при установлении TLS-соединения через ALPN (Application-Layer Protocol Negotiation).
- О поддержке HTTP/3 сервер может сообщить заголовком Alt-Svc. Клиент пробует установить отдельное QUIC-соединение.
- Если QUIC недоступен, клиент может продолжить работу через HTTP/1.1 или HTTP/2.



Проектирование API: ресурсы и операции

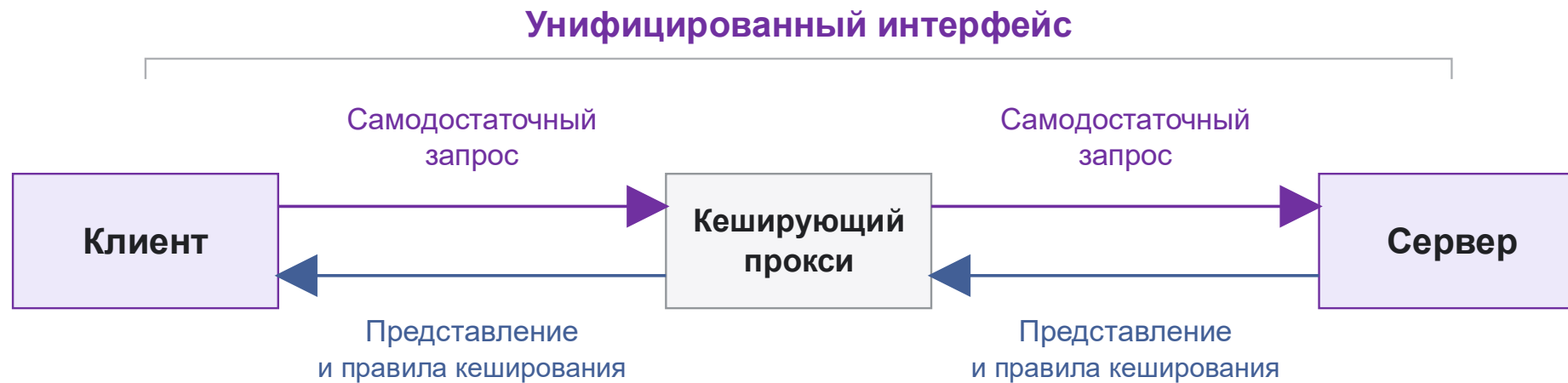
- **Ресурсный подход:** клиент обращается к ресурсам через унифицированный интерфейс.
- **RPC-подход:** клиент вызывает именованные операции с параметрами.
- Оба подхода можно реализовать поверх HTTP независимо от его версии.



REST: архитектурные ограничения

REST (Representational State Transfer) – архитектурный стиль взаимодействия распределённых компонентов.

- **Клиент-сервер:** разделение ответственности клиента и сервера.
- **Stateless:** каждый запрос самостоятелен и не требует серверного контекста сессии.
- **Кеширование:** для ответа определено, допустимо ли его повторное использование.
- **Унифицированный интерфейс:** общие правила взаимодействия с ресурсами.
- **Многослойная (layered) система:** взаимодействие через посредников без знания всей цепочки.



Унифицированный интерфейс REST

- **Идентификация ресурсов:** ресурсы идентифицируются с помощью URI.
- **Работа через представления:** клиент получает и передаёт данные о состоянии ресурсов.
- **Самоописываемые сообщения:** методы, коды ответа и метаданные определяют способ обработки сообщения.
- **Гипермедиа:** сервер сообщает доступные переходы через ссылки и элементы управления.

Запрос

GET `/orders/42`

URI ресурса

Ответ (фрагмент)

200 OK

Content-Type: application/json

Код ответа
и метаданные

```
{
  "id": 42,
  "status": "created",
  "links": {
    "collection": "/orders"
  }
}
```

Представление
состояния заказа

Ссылка
на коллекцию заказов

REST: преимущества и компромиссы

Что даёт REST	Что приходится учитывать
Самодостаточные запросы упрощают балансировку нагрузки	Контекст приходится передавать в каждом запросе
Кеширование снижает нагрузку и задержки	Нужно управлять актуальностью данных
Унифицированный интерфейс упрощает взаимодействие	Не каждую прикладную операцию удобно выразить через ресурсы

REST упрощает взаимодействие компонентов, но не гарантирует масштабируемость и совместимость: они зависят от проектирования API и реализации системы.

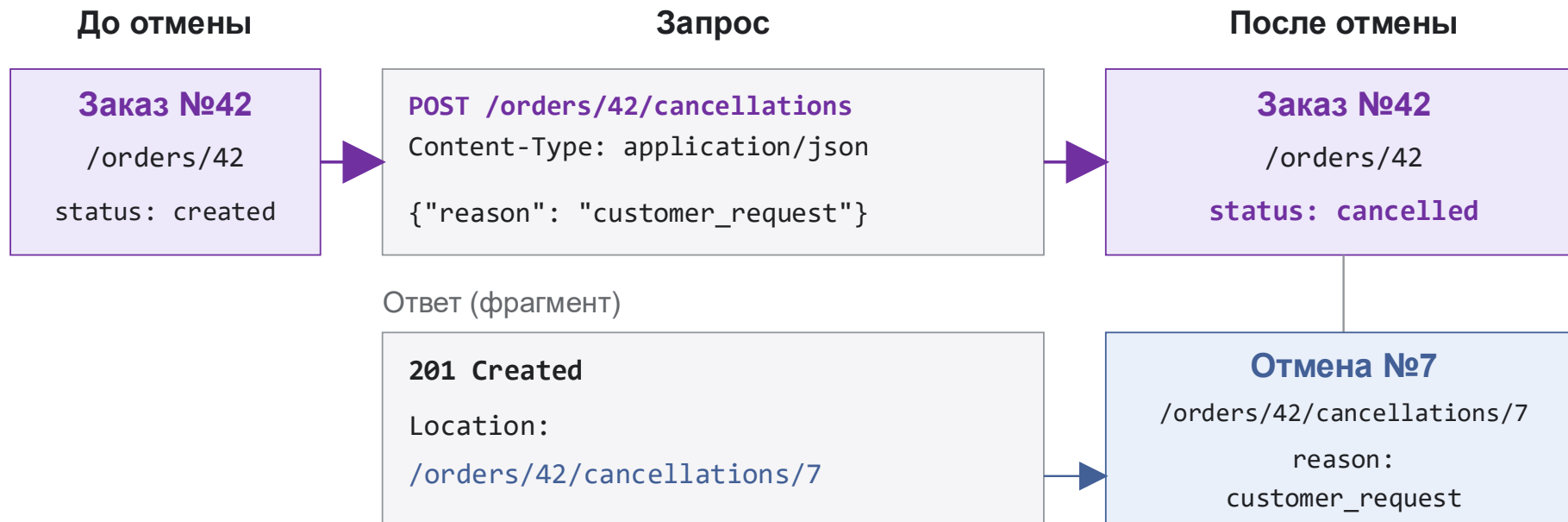
REST API: коллекции и отдельные ресурсы

- Коллекция и отдельный объект имеют разные URI.
- HTTP-метод определяет действие над выбранным ресурсом.
- Доступные операции зависят от предметной области и состояния ресурса.

Запрос	Назначение в нашем API
GET /orders	Получить список заказов
POST /orders	Создать заказ
GET /orders/42	Получить заказ №42
PATCH /orders/42	Частично изменить заказ
DELETE /orders/42	Удалить ресурс заказа, если это разрешено

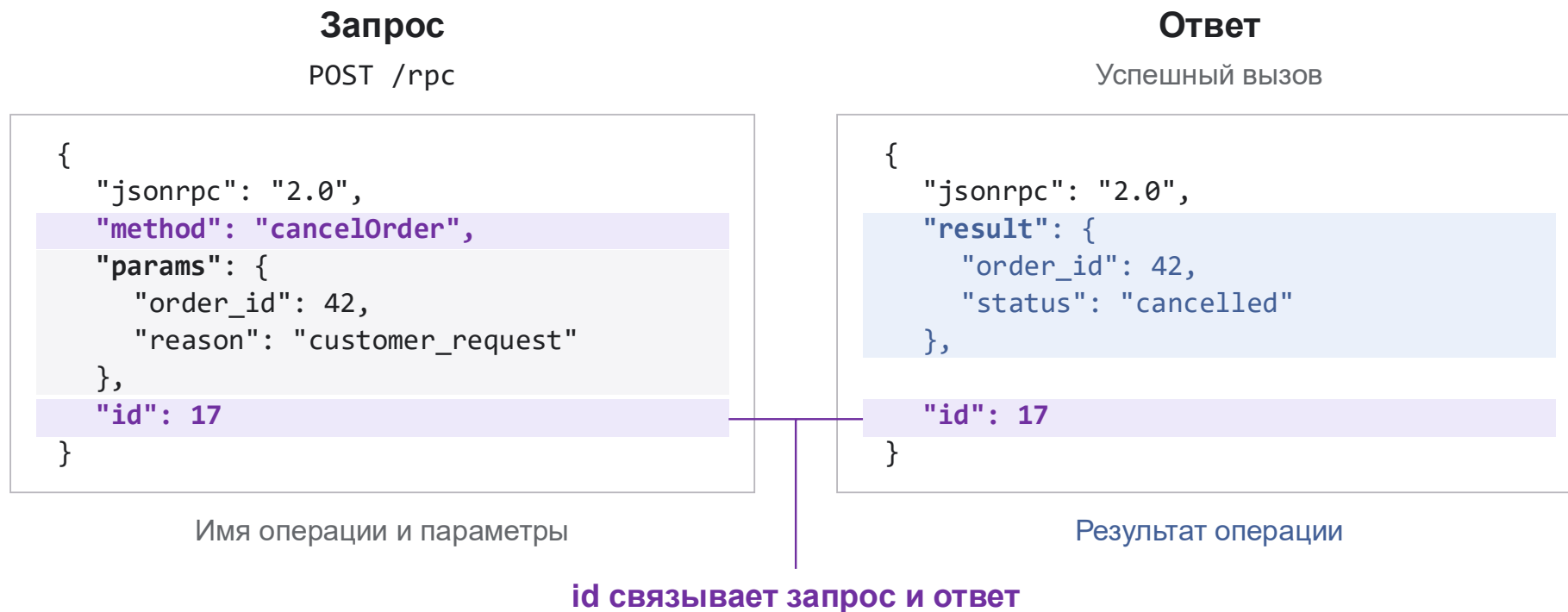
REST API: прикладные операции

- Прикладные действия не всегда сводятся к изменению или удалению объекта.
- **Операцию можно представить отдельным ресурсом** со своими данными и состоянием.
- Сервер проверяет допустимость действия и сохраняет результат.



RPC поверх HTTP

- Клиент передаёт в запросе имя операции и её параметры.
- Сервер выполняет операцию и возвращает результат или ошибку.
- HTTP передает сообщения RPC, а прикладные операции определяет API.



REST и RPC: сравнение подходов

Критерий	REST API поверх HTTP	RPC
Схема взаимодействия	Ресурсы, их представления и унифицированный интерфейс	Вызов именованных операций с параметрами и результатом
Формат сообщений	Часто JSON, но возможны другие форматы	Зависит от технологии: JSON-RPC использует JSON, gRPC обычно Protobuf
Описание интерфейса	Например, OpenAPI	Зависит от фреймворка: например, .proto в gRPC
Генерация кода	Возможна по описанию API	В gRPC и многих других фреймворках входит в основной процесс разработки
Совместимость версий	Нужно сохранять контракт ресурсов и представлений	Нужно сохранять контракт операций и сообщений
Типичное применение	Внешние API для независимо развиваемых клиентов	Взаимодействие внутренних сервисов при контроле над обеими сторонами

Области применения пересекаются. Выбор зависит от задачи, инструментов и контроля над клиентами.

HTTP: что важно запомнить

- **Семантика HTTP** определяет работу с ресурсами: методы, коды ответа, заголовки и представления.
- **HTTP/1.1, HTTP/2 и HTTP/3** сохраняют общую семантику, но различаются организацией передачи сообщений.
- **REST и RPC** предлагают разные способы организации API. Выбор зависит от задачи, инструментов и контроля над клиентами.

Материалы

- [High Performance Browser Networking](#) – главы 9, 11, 12
- [HTTP/2 in Action](#) – главы 1, 4; дополнительно §7.2 и глава 8
- [REST in Practice: Hypermedia and Systems Architecture](#) – главы 1, 4, 5
- [Roy Fielding: Representational State Transfer \(REST\)](#) – §5.1
- [Everything curl: HTTP/3](#)

Дополнительные материалы

- [RFC 9110: HTTP Semantics](#) – §1–3, §9, §15: основные понятия, методы и коды ответа
- [RFC 9113: HTTP/2](#) – согласование протокола, кадры и потоки
- [RFC 9114: HTTP/3](#) – §2–4: обзор, установление соединений и обмен сообщениями
- [RFC 9000: QUIC](#) – устройство транспортного протокола
- [The QUIC Transport Protocol: Design and Internet-Scale Deployment](#) – ранний Google QUIC и опыт его внедрения