

Дизайн систем машинного обучения

9. Поточковые данные

План курса

- 1) Практическое применение машинного обучения
- 2) Основы проектирования ML-систем
- 3) Обучающие данные
- 4) Подготовка и отбор признаков
- 5) Выбор модели, разработка и обучение модели
- 6) Оценка качества модели
- 7) Развертывание
- 8) Диагностика ошибок и отказов ML-систем
- 9) Поточковые данные — Вы находитесь здесь**
- 10) Жизненный цикл модели
- 11) Отслеживание экспериментов и версионирование моделей
- 12) Сложные модели: временные ряды, модели над графами
- 13) Непредвзятость, безопасность, управление моделями
- 14) ML инфраструктура и платформы
- 15) Интеграция ML-систем в бизнес-процессы

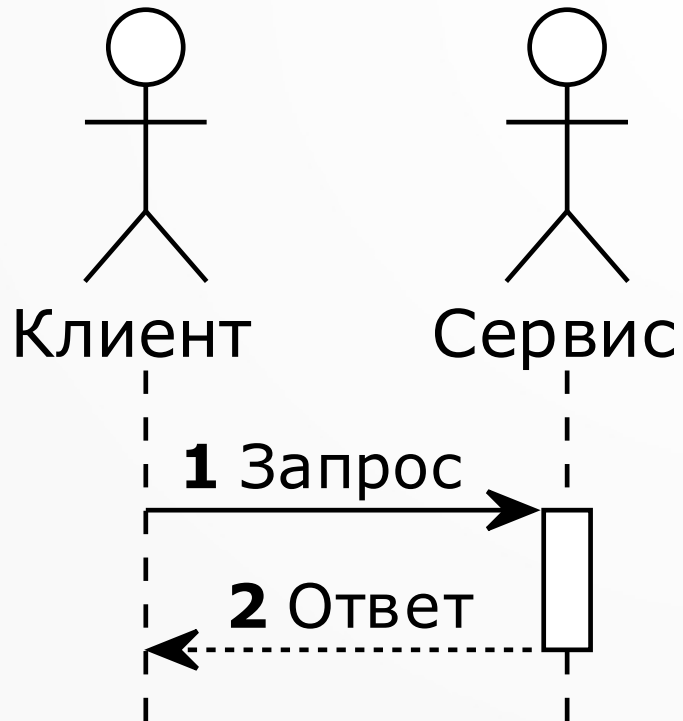
Disclaimer

- Это не учебник по распределенным системам
- Это не учебник по потоковым данным
- Это не учебник для датаинженеров
- Вам все равно придется столкнуться с потоковыми данными
- Задача лекции: показать, что бывает и как гуглить

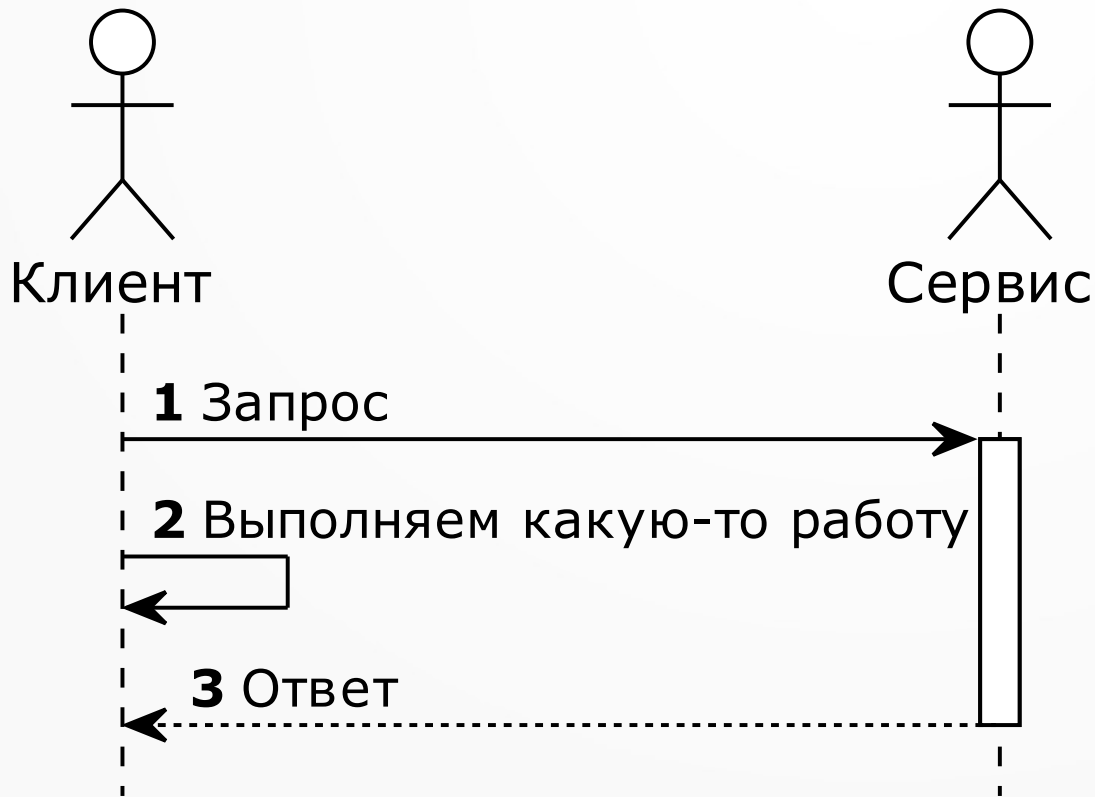
Как данные поступают в систему?

- Синхронный запрос-ответ
- Асинхронный ответ
- Полноасинхронный запрос-ответ
- Запрос-подтверждение
- Издатель-подписчик
- Одностороннее взаимодействие
- Длинные опросы
- Наблюдатель + обратный вызов

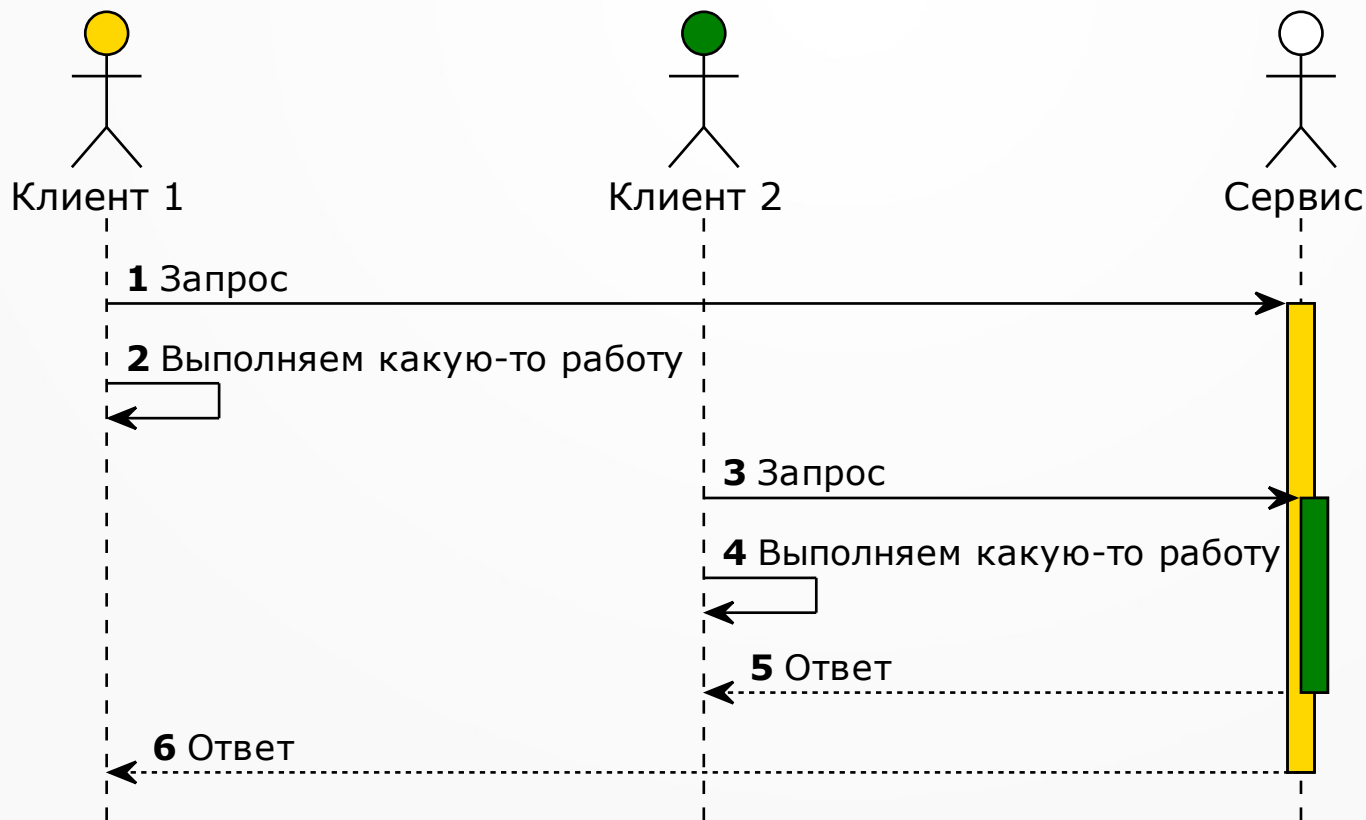
Синхронный запрос-ответ



Асинхронный ответ

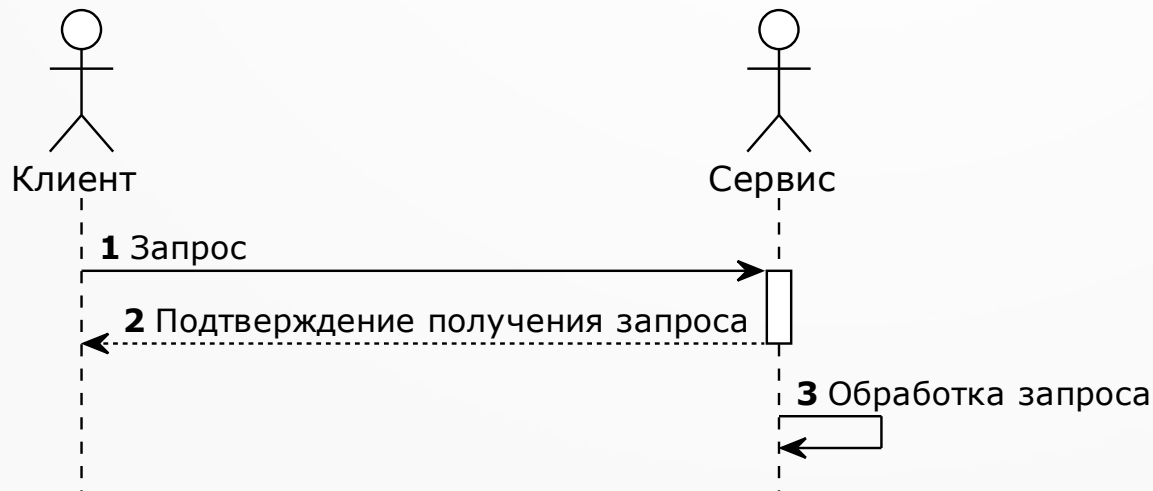


Полноасинхронный запрос-ответ

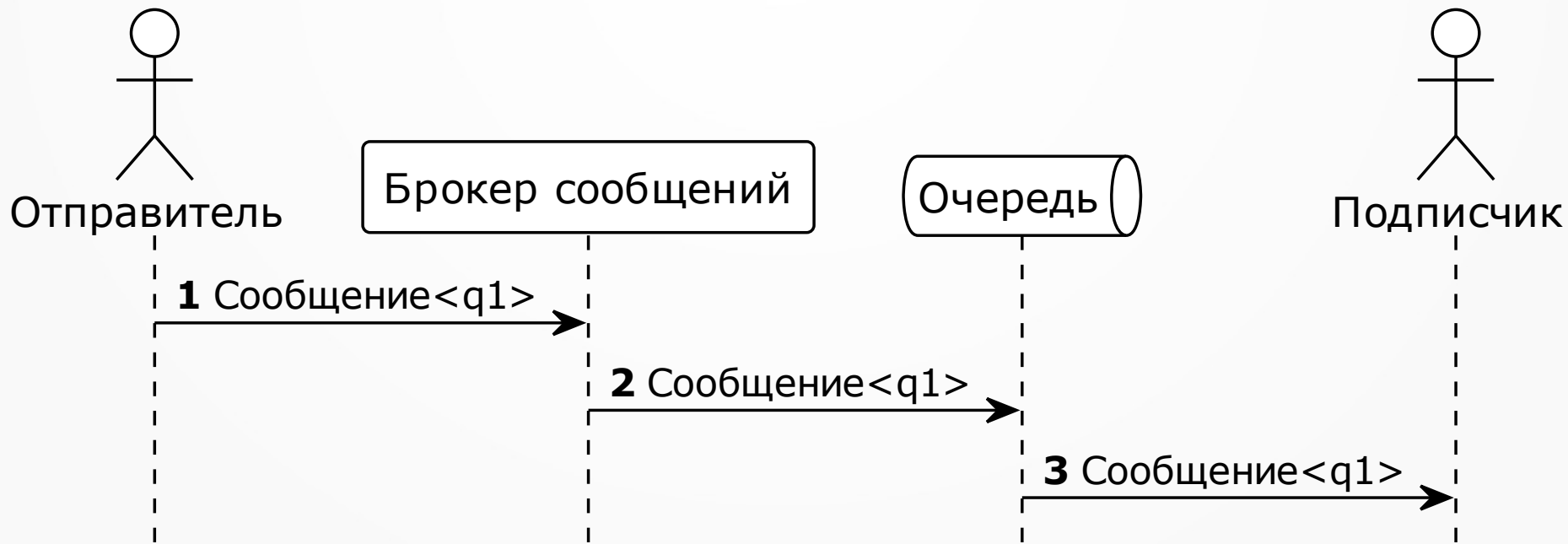


Запрос-подтверждение

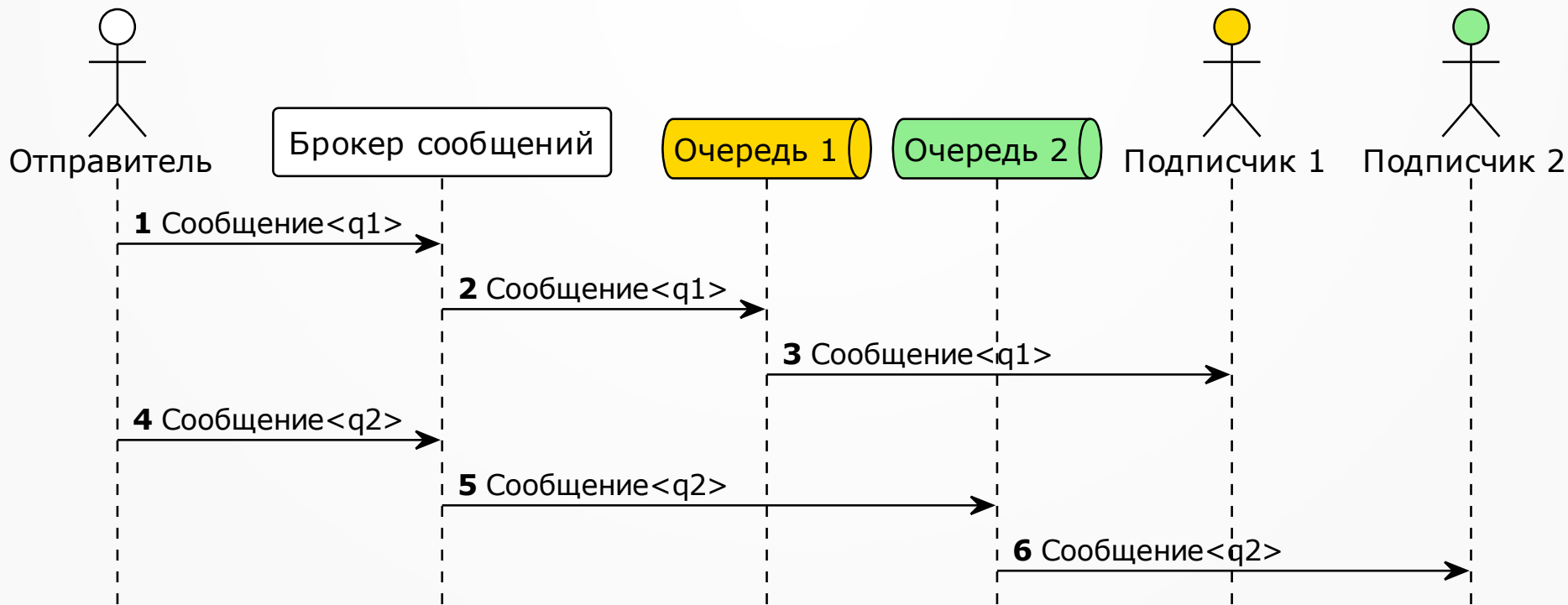
- Запрос передается и подтверждается получателем
- Результаты обработки не передаются
- Обработка запроса производится после получения



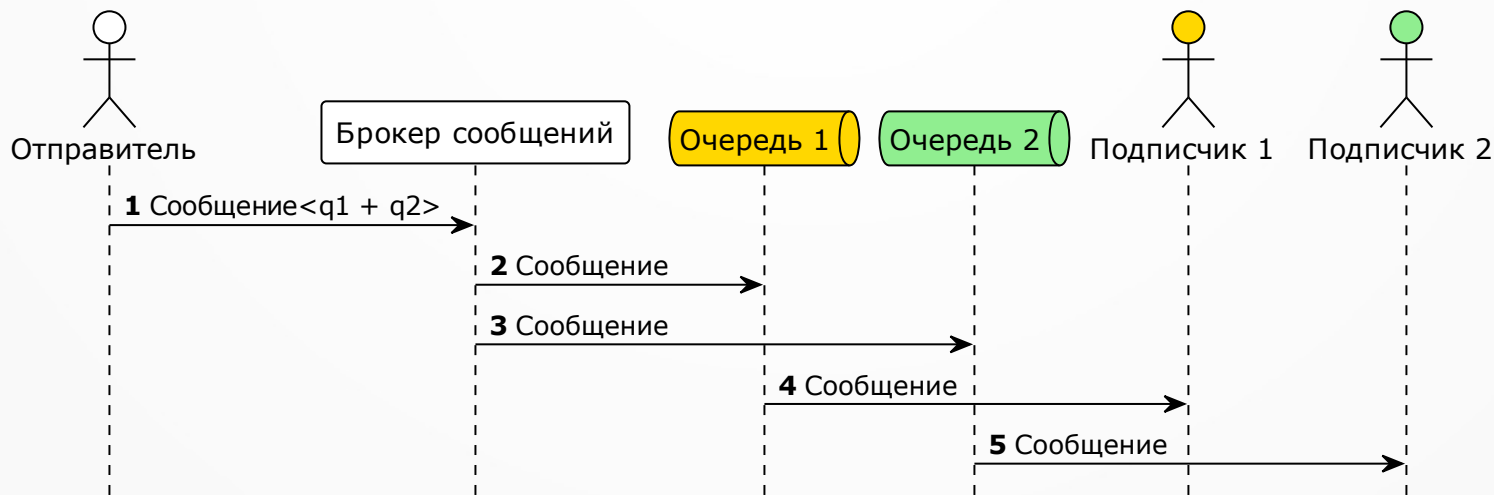
Отправитель-подписчик



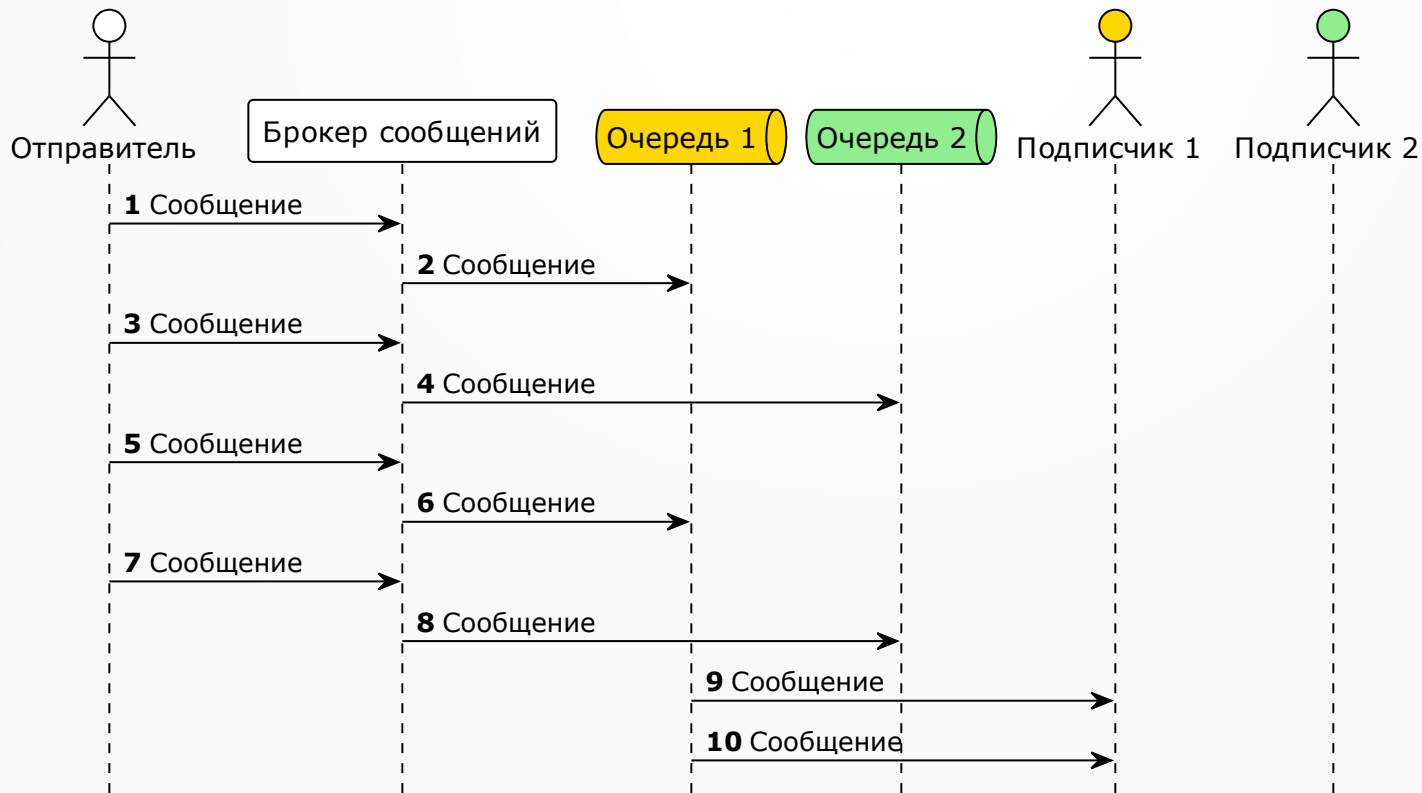
Отправитель-подписчик



Отправитель-подписчик

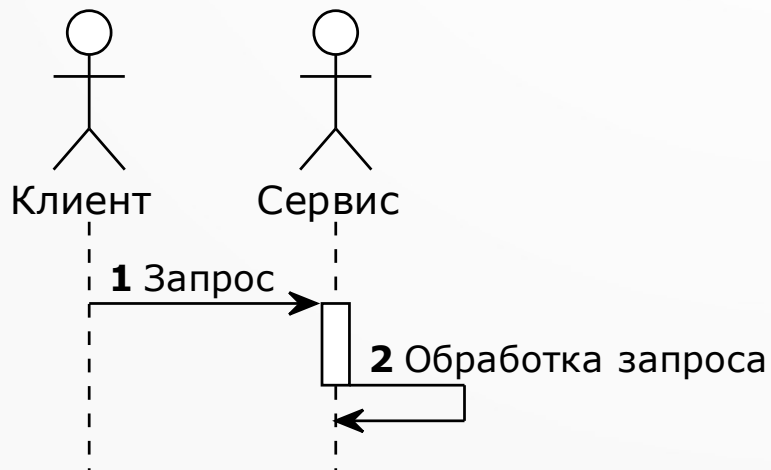


Отправитель-подписчик

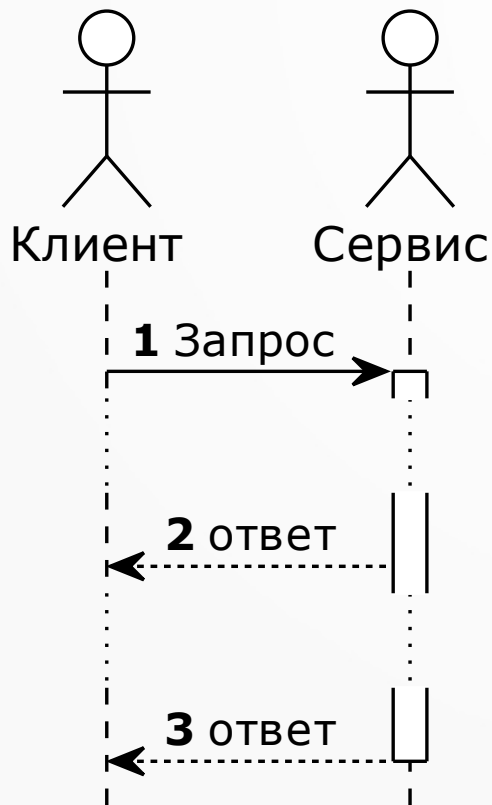


Одностороннее взаимодействие

- Источник передает данные
- Не ждет результатов и подтверждения получения
- Пример: **UDP** протокол



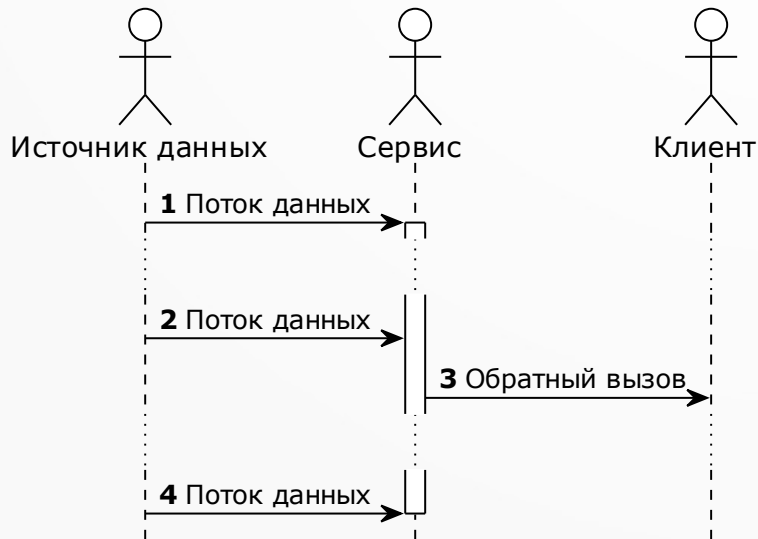
Длинные опросы Long Polling



- Socket.IO
- VK
- Yandex MQ

Наблюдатель + обратный вызов

- Поток событий (например, звук с микрофона)
- Обратный вызов клиента, когда что-то интересное случилось



Подробнее про потоковую обработку

- Stream Processing
- Streaming Data
- Терминология
- Брокеры сообщений
- Сценарии использования

Обработка потоков событий Stream Processing

- Способ проектировать системы:
 - Обмен информации организован событиями
 - В систему поступают события
 - Приложения подписываются на события
 - Приложения генерируют события
- Событие:
 - Что-то, что случилось в системе
 - Маленький независимый кусочек данных
- см. [Stream processing](#)

Потоковые данные Streaming Data

- Способ организации данных:
 - постоянный поток событий
 - генерирующихся в реальном времени
 - не имеющий определенного начала или конца
 - можно использовать, имея доступ только к части данных
 - обрабатываются асинхронно
 - сохранение и обработка данных разделены
- см. [Streaming Data: How it Works, Benefits, and Use Cases](#)

Примеры потоковых данных

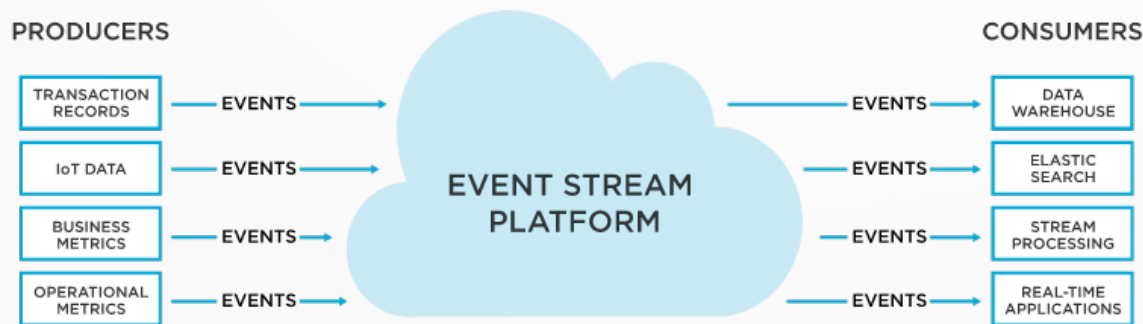
- Логи серверов
- Геолокация пользователя
- Данные с датчиков оборудования
- Like Dislike (соцсети, рекомендательные системы)
- Транзакции по кредитной карте (Fraud Detection)
- Запросы пользователей к сайту (Bot Detection)
- Ценовые сигналы для торгового робота (Stock Trading)

Особенности потоковых данных

- Структурированные короткие сообщения
- Отправитель не ждет ответа на каждое сообщение
- Отправитель не знает, кто обрабатывает его данные
- Данные могут поступать с задержкой
- Данные могут поступать с перерывами
- Неравномерный поток данных

Как обычно устроено

- Producer — генерирует события
- Message Broker — сохраняет события
- Consumer — потребляет события
- Queue, Topic — очередь событий



Producer

- Кто-то, кто генерирует событие
- Внешний источник данных
- Ваше приложение
- Подключается к Message Broker
- Может быть одновременно Consumer

Message Broker

- Получает сообщение
- Складывает в одну или несколько очередей
- По запросу отдает сообщение из очереди
- Удаляет сообщения из очередей по запросу или по правилам

Consumer

- Кто-то, кто получает сообщения
- Ваше приложение
- Внешний пользователь системы
- Подключается к Message Broker
- Может быть одновременно Producer

Торіс или Queue

- Очередь сообщений
- Много Producers
- Много Consumers
- Внутри Message Broker может быть много топиков
- Сообщения внутри одного топика обрабатываются одинаково

Message Broker — какие бывают

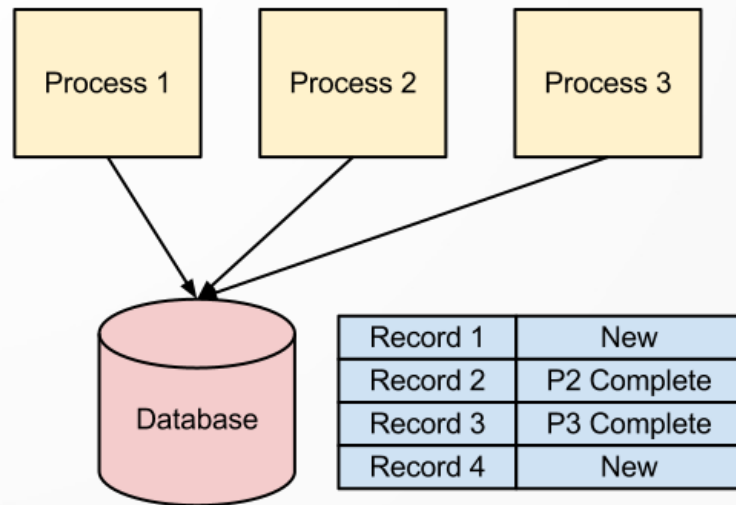
- Обычная база данных
- Google Pub/Sub
- Amazon Simple Queue Service (SQS) Yandex Message Queue (YMQ)
- RabbitMQ
- Apache Kafka
- Amazon Kinesis Yandex Data Streams
- CM Message Broker

Обычная база данных — брокер сообщений

- Producer записывает событие в базу данных
- Consumer раз в минуту, например, делает запрос к базе

```
SELECT * FROM MyWorkflow WHERE Status = 'New'
```

- Обработанные события удаляются или помечаются как обработанные
- Не рекомендуется так делать



<http://mikehadlow.blogspot.com/2012/04/database-as-queue-anti-pattern.html>

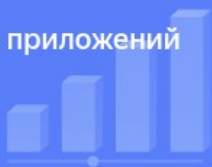
Обычная база данных — брокер сообщений

- Не рекомендуется так делать:
 - Реляционные базы придуманы для другого типа нагрузки
 - Прямой доступ Producer в базу данных — плохая практика
- Но так все равно делают!
 - В маленьких системах это очень удобно
 - В монолитных системах это очень удобно
 - База данных уже есть и хорошо масштабируется
- Dagster использует Postgress для обработки логов →
- Таких систем, на самом деле, много

Yandex MQ

- Стандартные и FIFO-очереди
- Интеграция с функциями и триггерами

Масштабирование приложений



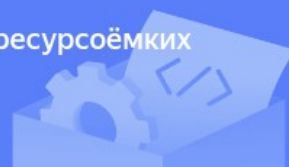
Используйте очередь сообщений, чтобы масштабировать систему из независимых приложений и сервисов. Когда налажен процесс отправки и обработки сообщений, можно поменять количество приложений, не затрагивая остальные части системы.

Повышение отказоустойчивости



Если приложение не может обработать сообщение, оно вернется в очередь и его сможет прочитать другой обработчик. Также можно настроить Dead Letter Queue (DLQ) — очередь, куда перенаправляются сообщения, которые не смогли обработать получатели в обычных очередях.

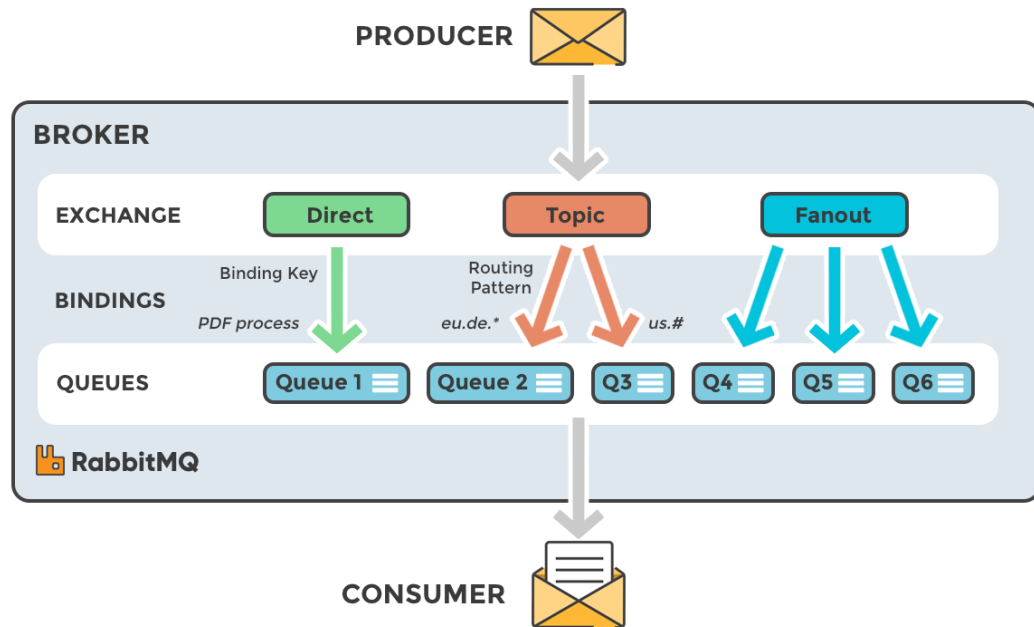
Выполнение ресурсоёмких задач



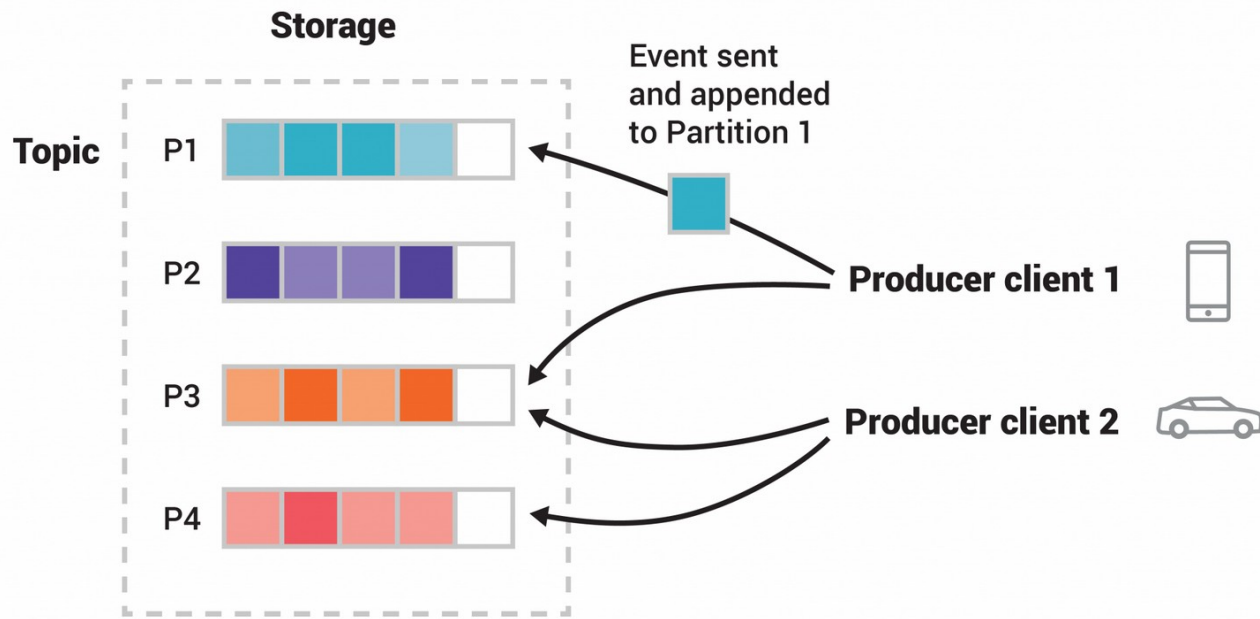
С помощью очереди сообщений можно вынести выполнение длительных задач в отдельные приложения, быстрее освободив ресурсы для новых запросов пользователей.

<https://cloud.yandex.ru/services/message-queue>

RabbitMQ



Apache Kafka



<https://kafka.apache.org/documentation/#gettingStarted>

Apache Kafka

- Каждый клиент помнит, докуда он дочитал
- Можно делать оконные запросы к потоку сообщений
- Обычно топики удаляют старые сообщения
- Может получиться, что удалили сообщения, которые вы не успели прочитать
- Самый сложный (и полезный) вариант

Что делать с потоковыми данными

- Материализованное представление Materialized View
 - По мере поступления данных обновляем насчитанные признаки
 - Используем их при запросе
- Оконные запросы →
- Триггеры при выходе оконной статистики за пределы
 - Отправка уведомления
 - Реакция на событие
 - Переобучение модели

Вводная 1

- Сервер может обработать 10 запросов в секунду
- Один запрос 1 Мбайт входящего трафика, 0.01 исходящего
- 1000 пользователей, каждый будет отправлять 10 запросов в день
- Запросы будут равномерно идти в рабочее время с 8 до 18
- Итого 10000 запросов, 10 часов, 1000 в час, 0,3 в сек
- Трафик входящий $0,3 \times 1 \text{ Мбайт} \times 10 \sim 3 \text{ Мбит/сек}$

О размере байта

- Почему умножаем не на 8?
- В теории 100-Мбит Ethernet пропускает ~96 мегабит (12 мегабайт)
- на практике еще по мелочи добавит http https.
Лучше чуть с запасом, поэтому байт у нас 10, а не 8
- Сколько весит картинка в 1 Мбайт
 - Base64: примерно 1,3 Мбайт, но это неточно
 - Protobuff: примерно 1 Мбайт

Вводная 2

- Сервер может обработать 10 запросов в секунду
- Один запрос 1 Мбайт входящего трафика, 0.01 исходящего
- 10000 пользователей, каждый будет отправлять 10 запросов в день
- Запросы будут равномерно идти в рабочее время с 8 до 18
- Итого 100 000 запросов, 10 часов, 10000 в час, 2.8 в сек
- Трафик входящий $3 \times 1\text{Мб} \times 10 \sim 30 \text{ Мбит/сек}$
- КПД системы 12% (ночью не загружена)

Вводная 3

- Сервер может обработать 10 запросов в секунду
- Один запрос 1 Мбайт входящего трафика, 0.01 исходящего
- 10000 пользователей, каждый будет отправлять 10 запросов в день
- Запросы будут идти в рабочее время с 8 до 18, но в первый час все отправят в среднем 5,5 запросов
- Итого 100000 запросов, 10 часов
 - 55000 запросов в первый час работы, ~15.3 запроса в секунду, трафик 153 Мбит/сек
 - 5000 в обычные часы, всего 45000, ~ 1.4 запроса в секунду, трафик 14 Мбит/сек
- 1000 в час, 2.8 в сек — должно хватить одного сервера, но нет

Ставим два сервера

- Прокси-сервер на входе
- Два сервера
- Пропускная способность 20 запросов в секунду 1728000 в сутки
- КПД системы > 6%
- Почти весь день используется 3
- Ночью сервера не используются
- Платим за два сервера и прокси-сервер

Обрабатываем с задержкой

- Сервер принимает задачи, ставит в очередь
 - Воркер выбирает задачи из очереди и обрабатывает
 - Результат клиенту приходит по электронной почте
-
- Платим за воркер и за сервер очередей
 - Жалуются, что долго ждать

Облачная функция

- Файл складывается в S3
- Как только файл создан, запускается облачная функция
- Готовый результат складывается в очередь сообщений
- Сервис выбирает из очереди сообщений и отдает клиентам
- Платим только за время работы облачной функции
- Цена за 1 запрос в 5 раз больше, но сумма примерно та же
- Можем обрабатывать пиковую нагрузку

Дополнительные материалы

- [Postgres: a better message queue than Kafka?](#)
- [RabbitMQ tutorials](#)
- [Introduction to Apache Kafka](#)
- [Example of using Yandex Message Queue in Python](#)