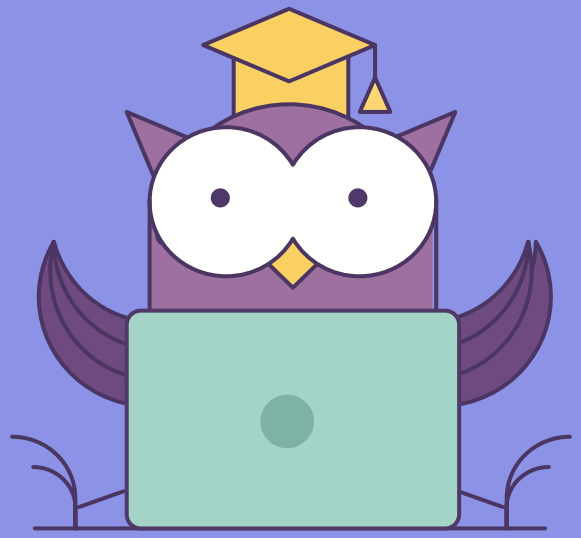
ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

Эволюция подходов работы с данными

Базовые принципы и понятия

Трусов Иван Алексеевич

Содержание

- Эволюция аналитических хранилищ данных
  - Монолиты
  - Микросервисы и ESB
  - Data Lake
  - Data-as-a-Service
- Форматы хранилищ данных
  - Реляционные (SQL)
  - Key-Value (NoSQL)
- Подходы к обработке данных
  - Batch
  - Stream
- Современные архитектуры
  - Lambda
  - Kappa

01

Эволюция аналитических хранилищ данных

Монолиты

Монолиты - 1980 - 200X (1/5)

Монолитный подход:

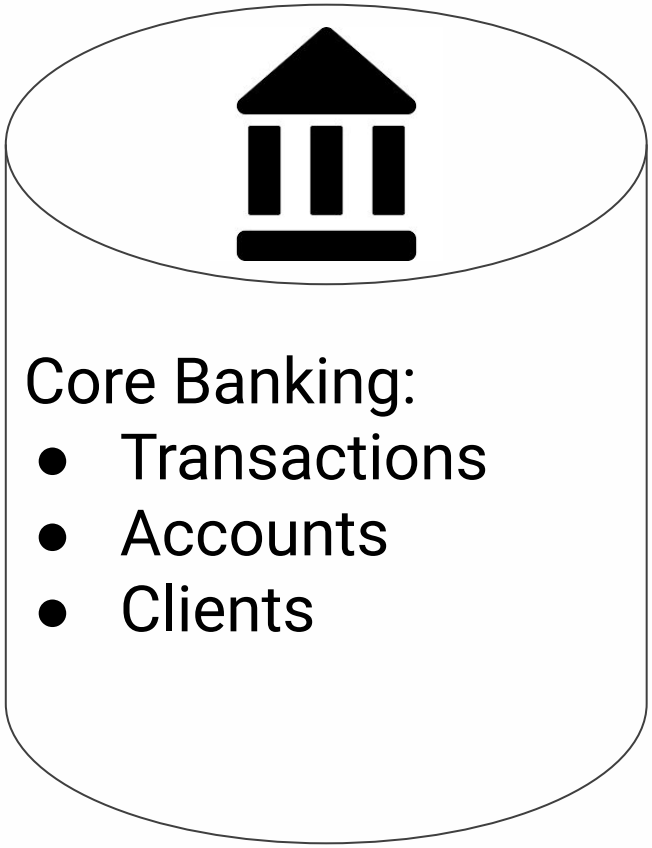
- Одна глобальная БД
- Одно большое приложение
- Только реляционные структуры
- Большой IO
- Stored procedures
- Предназначена для операционной нагрузки
- Вся бизнес-логика завязана на работу с данной БД

Монолиты - 1980 - 200X (2/5)

OTUS

Пример: банковская сфера

- Сверхнадежная БД
- Очень строгие SLA
- Страхование на случай отказов
- Географически распределенная система (разные ДЦ)
- Повышенные требования к безопасности (PCI/DSS)

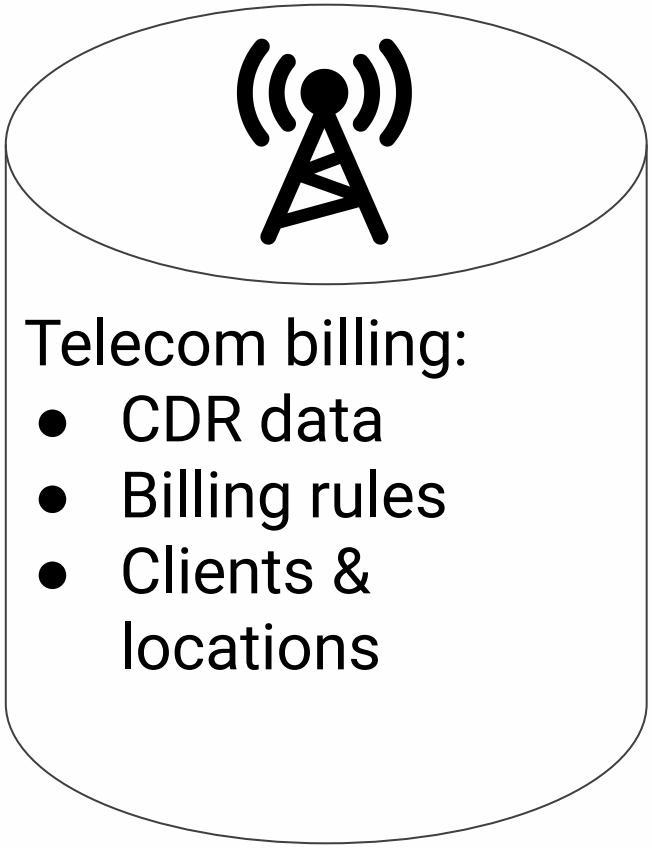


Монолиты - 1980 - 200X (3/5)

OTUS

Пример: биллинг в телекоме

- Обработка в реальном времени
- Огромный поток CDR
- Большое количество бизнес-логики
- Разный биллинг в зависимости от локации



Монолиты - 1980 - 200X (4/5)

OTUS

Пример: Ecommerce

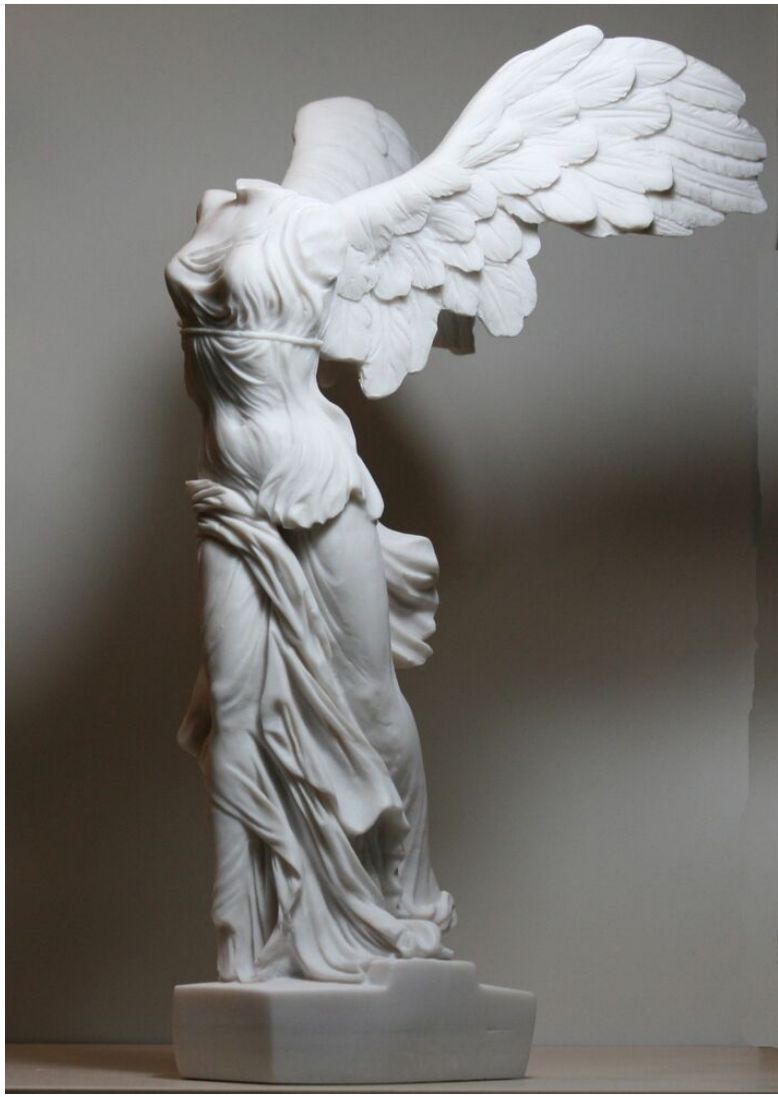
- Специфический формат и структура данных
- Only real time
- Множество интеграций с внешними системами:
  - Соцсети (авторизация)
  - Платежные системы
  - Системы доставки

Монолиты - 1980 - 200X (5/5)

OTUS

Проблемы монолитов

- Масштабируемость
- Связанность логики
- Высокая нагрузка
- Отсутствие аналитических инструментов
- Жесткая структура БД



01

Эволюция аналитических хранилищ данных

Микросервисы

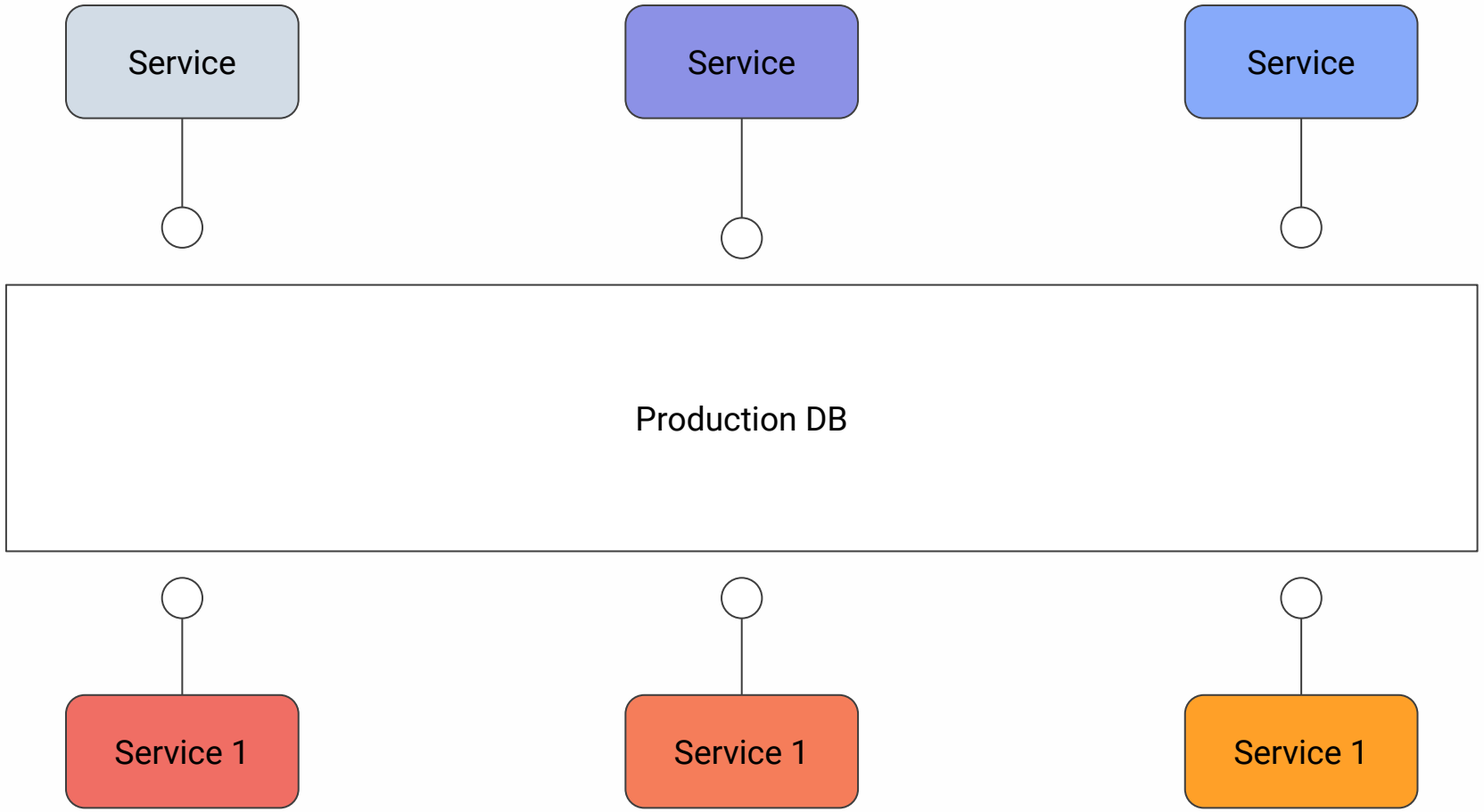
Микросервисы - 200X - 201X (1/6)

Микросервисный подход:

- Много слабо связанных БД
- Эксперименты с KV
- Ориентированы на бизнес-задачи
- Разделение логики
- Разделение CI/CD
- Integration overhead

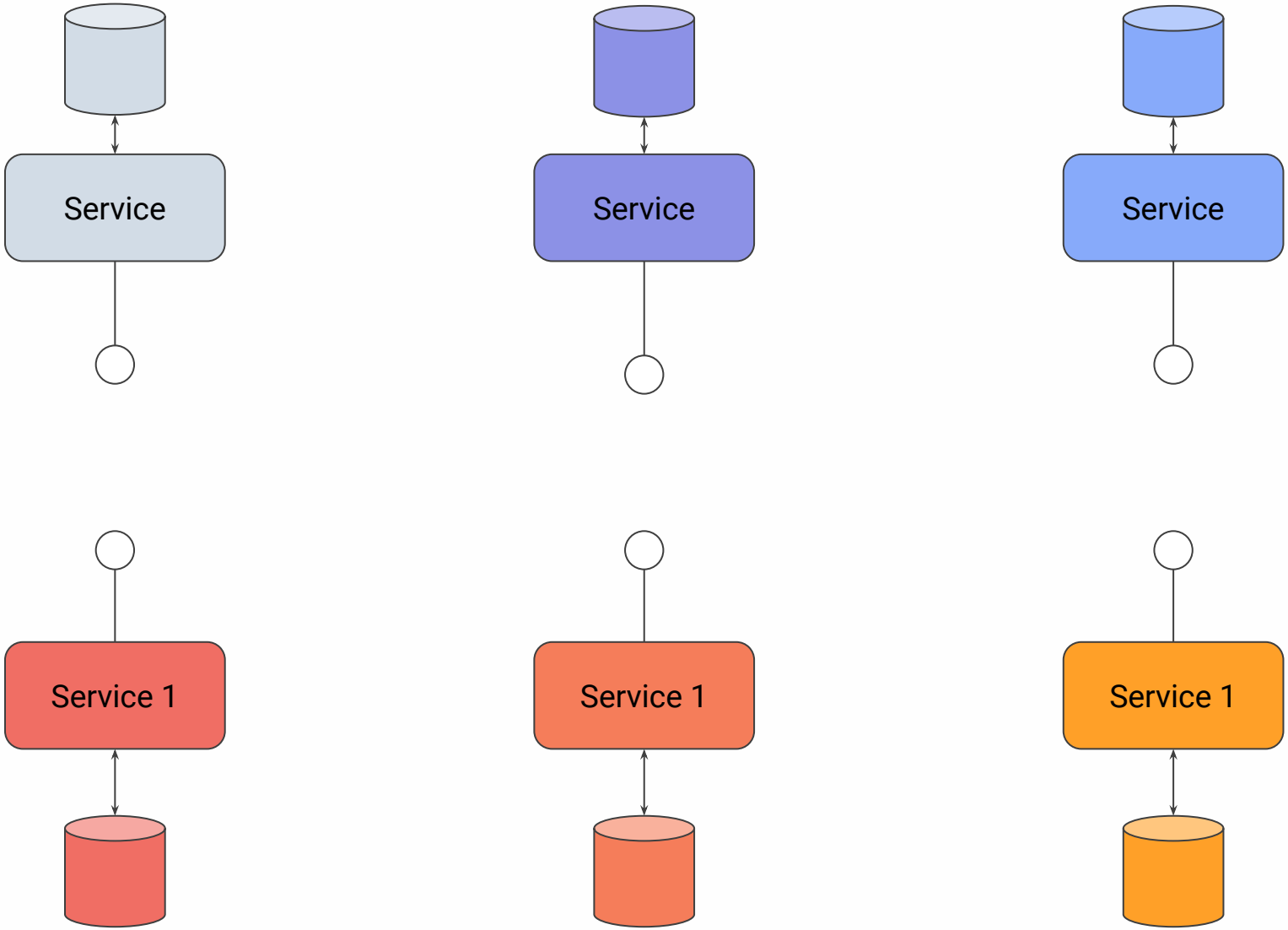
Микросервисы - 200X - 201X (2/6)

Шаг 1 - Перенос бизнес-логики в отдельные приложения



Микросервисы - 200X - 201X (3/6)

Шаг 2 - Перенос данных в отдельные БД



Микросервисы - 200X - 201X (4/6)

Проблема - потеря **связности** между сервисами

Решение - пусть **все** сервисы будут иметь **API**

Микросервисы - 200X - 201X (5/6)

Проблема - потеря **связности** между сервисами

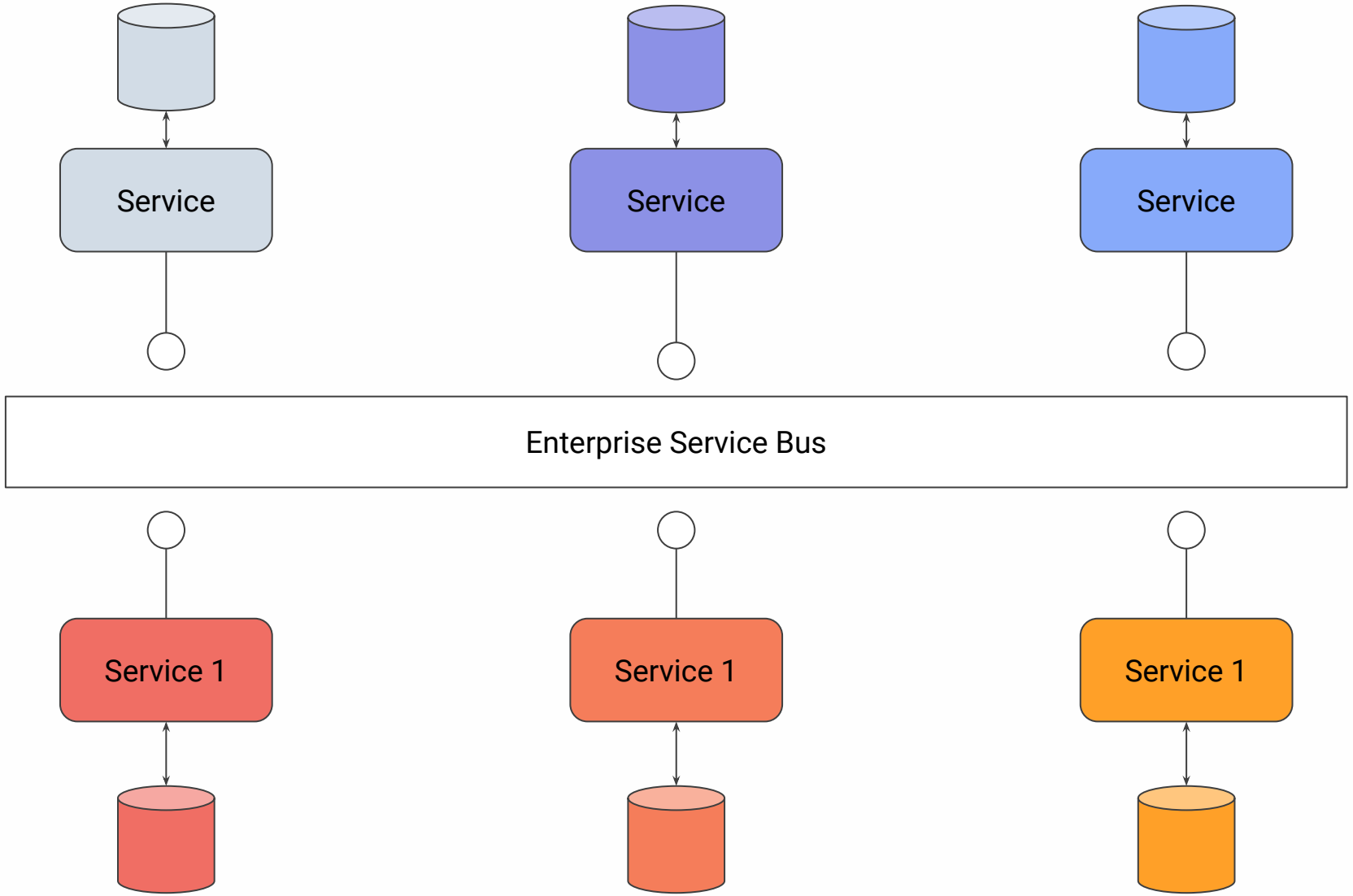
Решение - пусть **все** сервисы будут иметь **API**

Проблема - сервисы потребяют **одинаковые** данные

По разному.

Микросервисы - 200X - 201X (6/6)

Шаг 3 - Шина передачи данных



01

Эволюция аналитических хранилищ данных

Data Lake

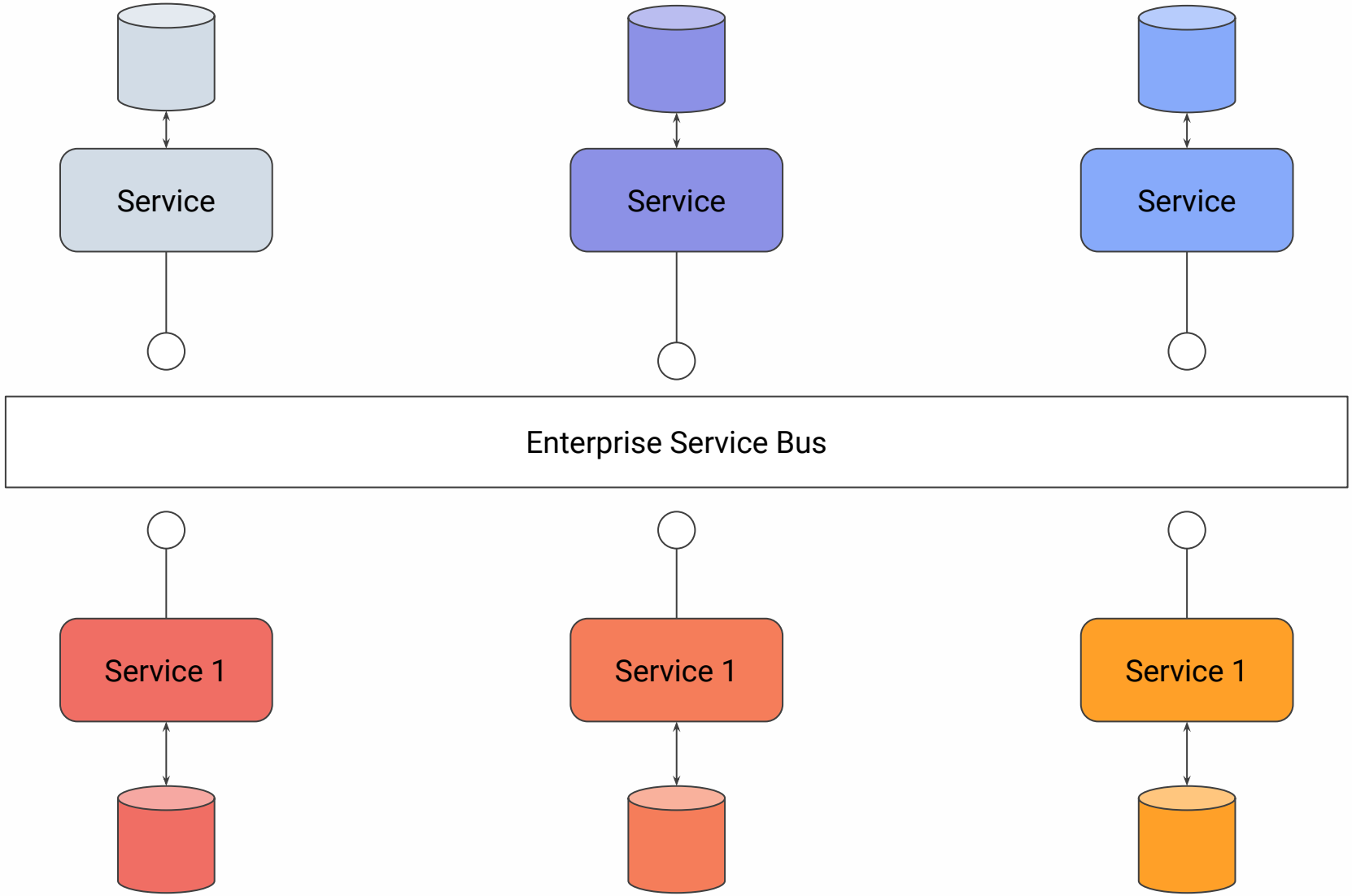
Data Lake - 201X - 201X (1/4)

Потребность в данных:

- Real-time
- Любые данные в компании
- Связанность
- Масштабируемость
- Скорость вычислений
- Высокая аналитическая нагрузка

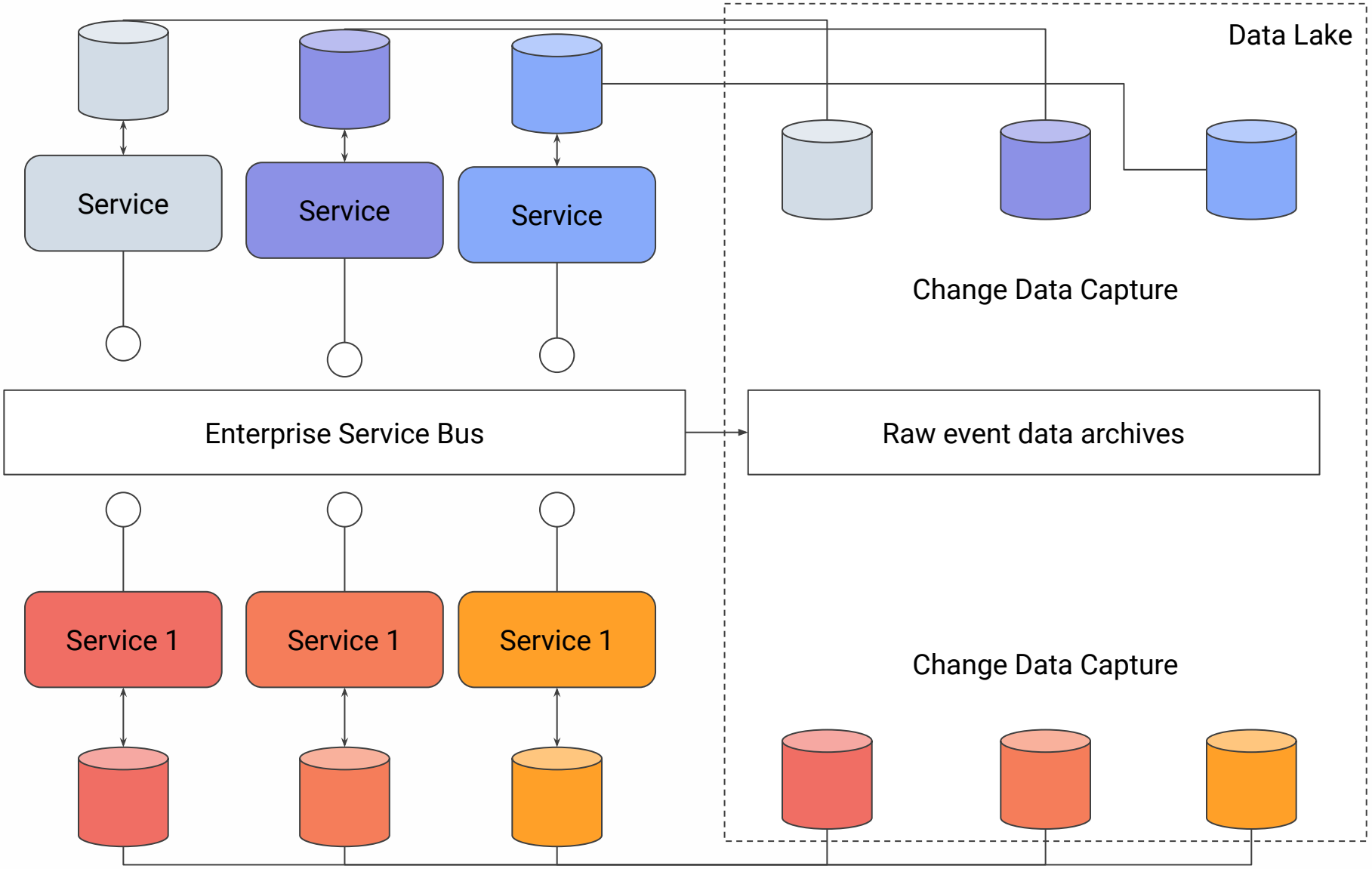
Data Lake - 201X - 201X (2/4)

Проблема - данные разрознены по нескольким БД



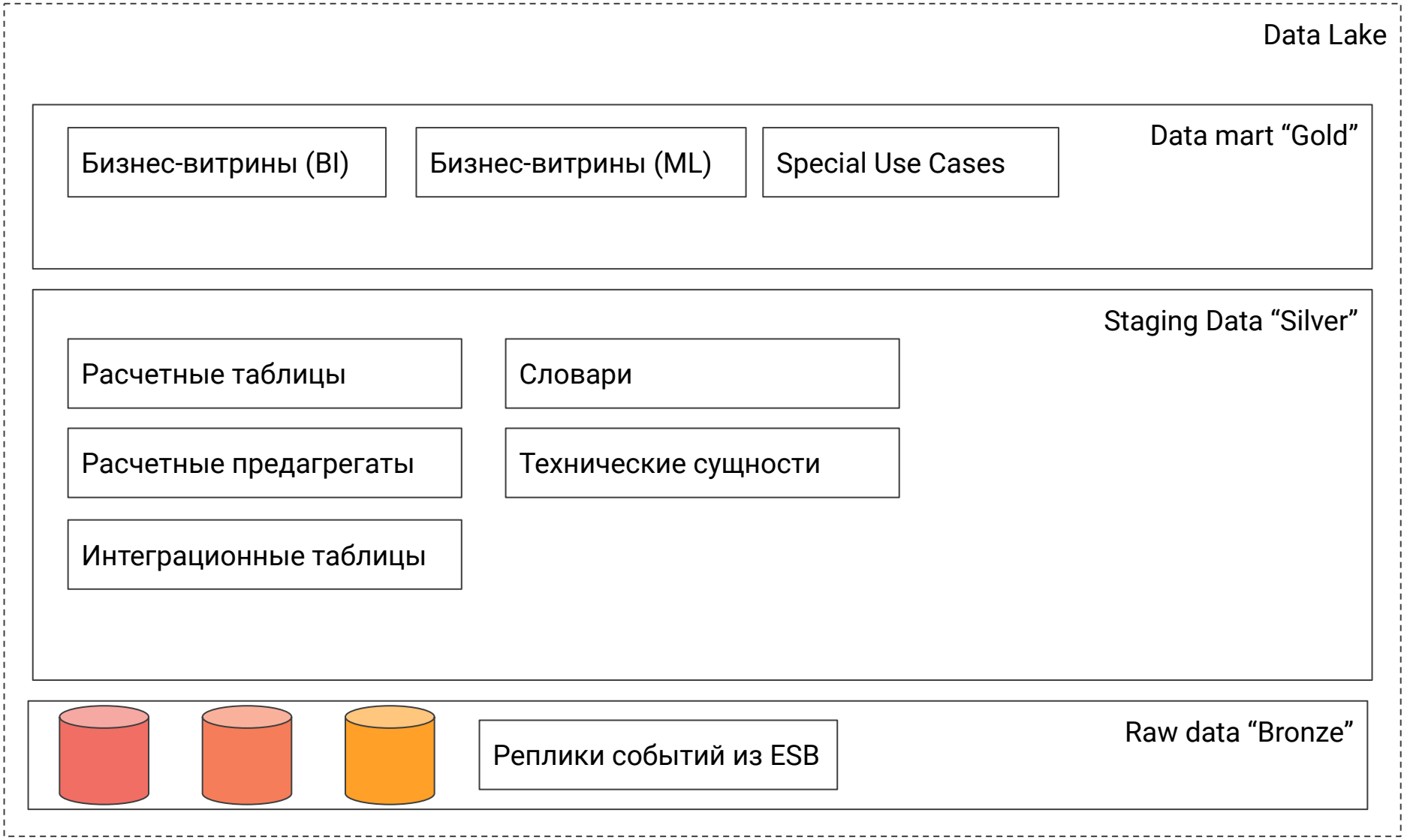
Data Lake - 201X - 201X (3/4)

Необходимо надежное и масштабируемое решение для их хранения



Data Lake - 201X - 201X (4/4)

Data Lake - это не просто **архив данных, а инфраструктурная** компонента



01

Эволюция аналитических хранилищ данных

Data-as-a-Service

Data-as-a Service - 201X - 201X (1/2)

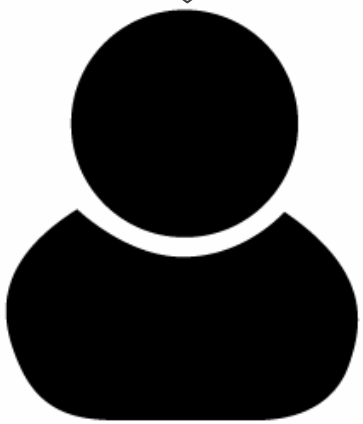
Данные как сервис:

- Любой формат потребления
- Интегрированные данные
- Доступное API
- Масштабируемость по запросам
- Легкость изменения структуры

Data-as-a-Service - 201X - 201X (2/2)

Data-as-a-Service - это **конструктор** датасетов и потоков

- Что:
  - датасет о покупках клиентов со скидками в регионе X, которые были доставлены в последние три часа
- Когда:
  - Каждые 10 минут с отставанием не более 30 секунд от реального времени
- Где:
  - В качестве .csv выгрузки в указанную папку



Data Service

- API
- UI
- Data catalog
- Most-used DS
- Data Governance

Data Lake

- Clients
- Discounts
- Orders
- Shipments
- Payments

02

Форматы хранилищ данных

Реляционные (SQL)

Реляционные БД (1/4)

Реляционные БД:

- предполагается ACID
  - Atomic
  - Consistent
  - Isolation
  - Durable
- SQL-compliant
- JDBC/ODBC-compliant
- Ключевые возможности:
  - insert/upsert/delete
  - партицирование
  - шардинг
  - индексы

ACID принципы

- **Atomicity (Атомарность)**
  - Транзакции могут состоять из **нескольких** операций. Атомарность гарантирует, что транзакция либо выполняется **полностью**, либо **не выполняется вообще**
- **Consistency (Консистентность)**
  - **Любая** запись в базу сохраняет ее **нормальное** состояние (корректно работают связанные объекты)
- **Isolation (Изоляция)**
  - Гарантирует, что **параллельные** транзакции выполняются так же, как если бы они выполнялись **последовательно***
- **Durability (Надежность)**
  - Если транзакция **подтверждена**, ее изменения **навсегда** останутся в базе (даже если сервер выключится)

Проблемы при параллельных транзакциях

- **Потерянное обновление**
  - Блок данных изменяется одновременно двумя транзакциями, применяется только последнее обновление
- **Грязное чтение**
  - Чтение данных, изменение которых не подтвердится
- **Неповторяющееся чтение**
  - Данные изменились во время чтения в одной транзакции
- **Фантомное чтение**
  - При повторном чтении внутри одной транзакции количество строк изменилось

Реляционные БД (4/4)

OTUS

Summary:

- Это не “**плохо**”
- Это не “**не модно**”

- **Это:**
  - хорошо развитая экосистема
  - понятное поведение БД
  - доступ из любого языка программирования

Всегда стоит подумать дважды, выбирая БД под высоконагруженный проект. Вполне возможно что обычная реляционная БД вполне способна справиться с нагрузкой.

02

Форматы хранилищ данных

Key Value Stores

Key Value Stores (1/4)

Key Value Stores:

- Не предполагается **ACID**
- Может предполагаться **EC**
- В большинстве случаев **не** SQL-compliant
- JDBC/ODBC-compliant
- Ключевые возможности:
  - индексы
  - insert/upsert/delete **BY KEY**
  - партицирование
  - шардинг
  - индексы

Key Value Stores (2/3)

Ключевые идеи:

- Данные хранятся в виде **коллекций из ключей и значений** (ассоциативный массив)
- Схема данных **легко** расширяется
- Основная задача - параллельные,сверхбыстрые **чтение и запись по ключу**

Key Value Stores (3/3)

Summary:

Да, это “модно”.

- **Это:**
  - очень быстрая запись и чтение по ключу
  - легко изменяемая структура хранения данных
  - зачастую требует много настройки и DevOps

Всегда стоит подумать трижды, выбирая KV под высоконагруженный проект. Различные KV дают различные возможности, нужно смотреть каждый отдельный use-case.

03

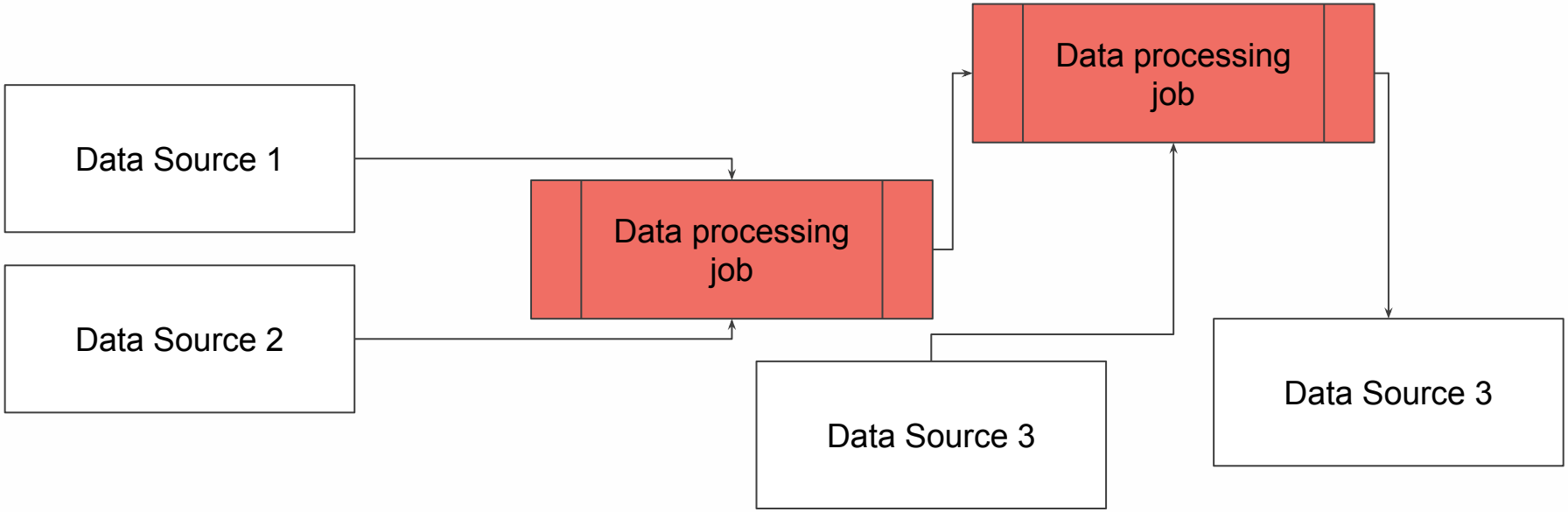
Подходы к обработке данных

Batch

Batch обработка данных (1/2)

Batch обработка:

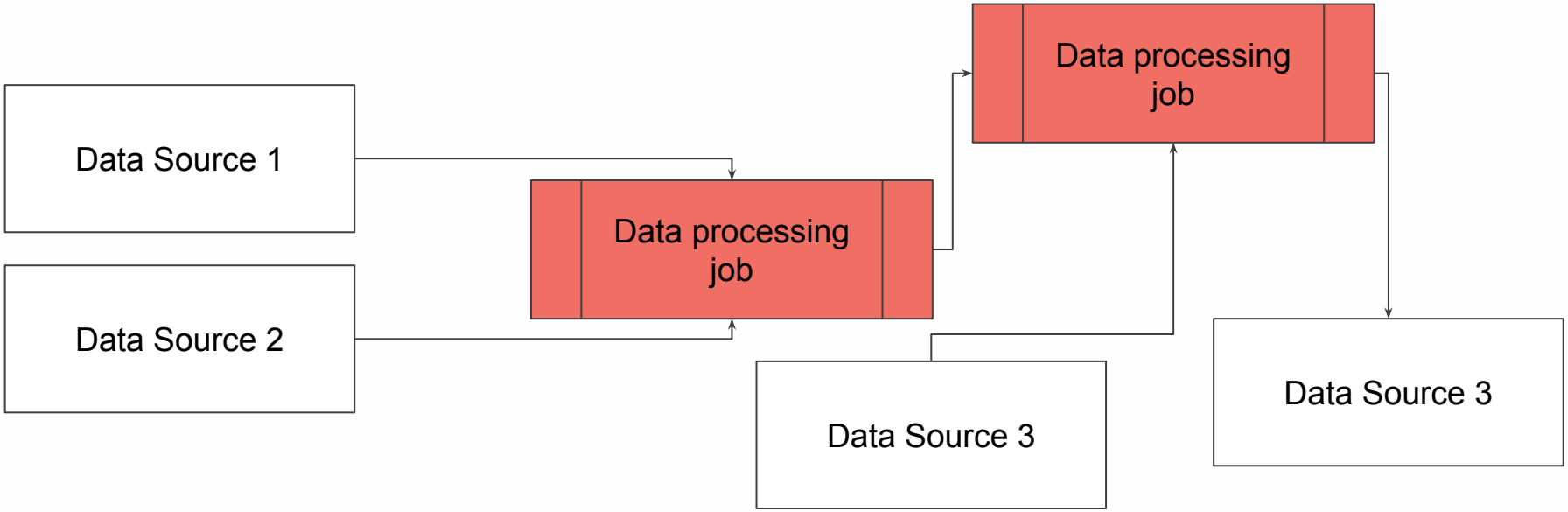
- Самый **распространенный** подход
- Идея:
  - Читаем данные блоком
  - Изменяем данные блоком
  - Записываем данные блоком
- Большинство ETL/ELT - это Batch



Batch обработка данных (1/2)

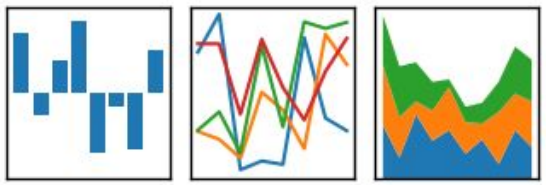
Batch обработка:

- Самый **распространенный** подход
- Идея:
  - Читаем данные блоком
  - Изменяем данные блоком
  - Записываем данные блоком
- Большинство ETL/ELT - это Batch



Batch обработка данных (1/2)

Инструменты для batch-обработки



04

Подходы к обработке данных

Stream

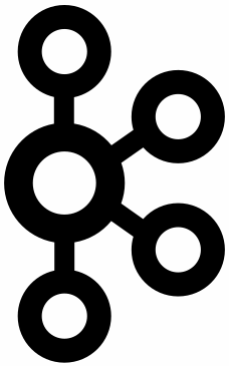
Stream обработка данных (1/2)

OTUS

Stream обработка:

- Обрабатываем данные “на лету”
- Различные процессы обрабатывают различные события
- Постоянно работающее streaming application
- Low overhead на обработку единичных событий
- Высокая масштабируемость - одно из ключевых требований
- Low-latency
- Различные delivery semantics

Инструменты для стриминговой обработки



confluent

APACHE Spark™

NATS

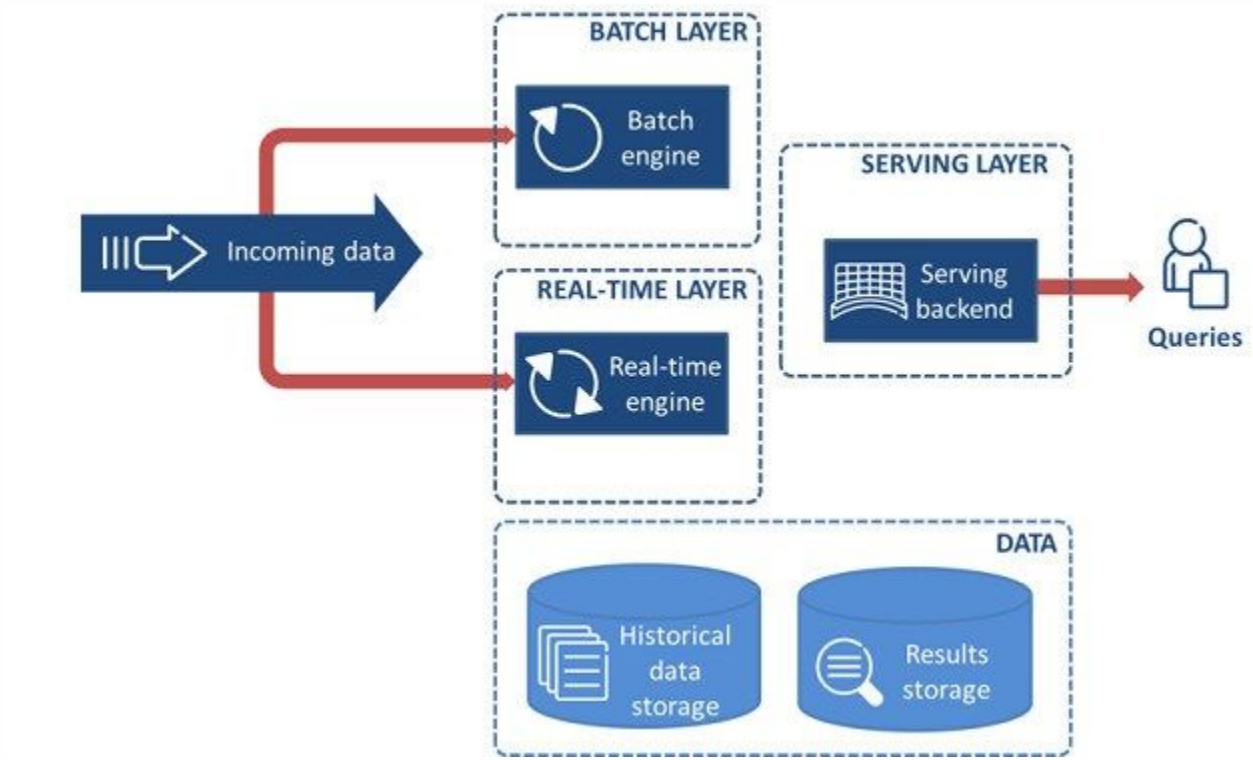
PULSAR

04

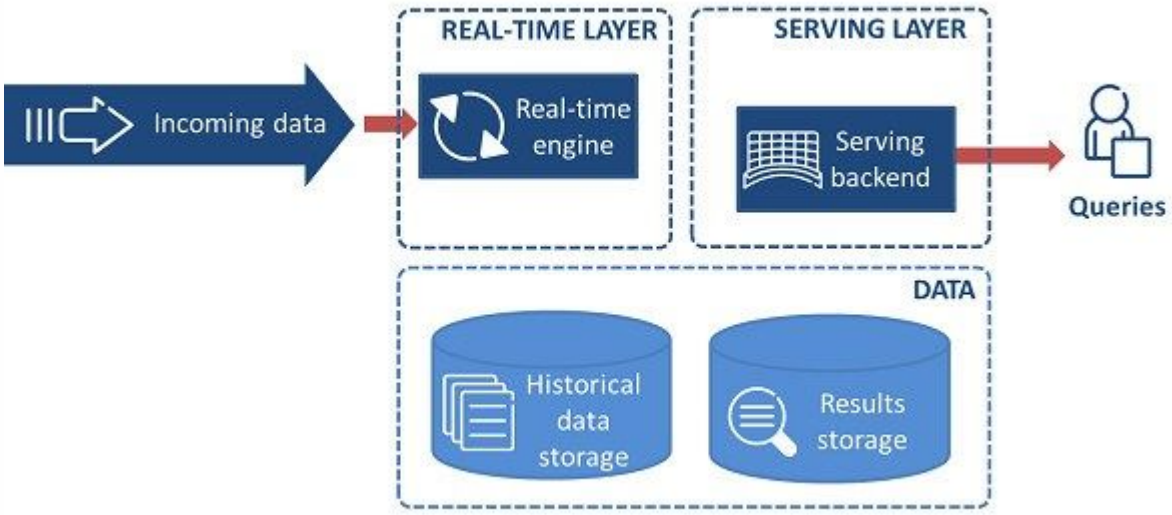
Современные архитектуры

Lambda & Kappa

Lambda architecture



Kappa architecture



Сравнение

- **Lambda** более универсальна
- **Lambda** архитектура требует двух кодовых баз:
  - одна для Batch Layer
  - другая для Realtime Layer
- **Kappa** архитектура предполагает что Batch обработка отсутствует

Трусов Иван

ivan.trusov.contact@gmail.com
https://renardeinside.github.io/

Спасибо
за внимание!