Architecture Concepts Based on Change Data Capture

Mateusz Dymiński

whoami

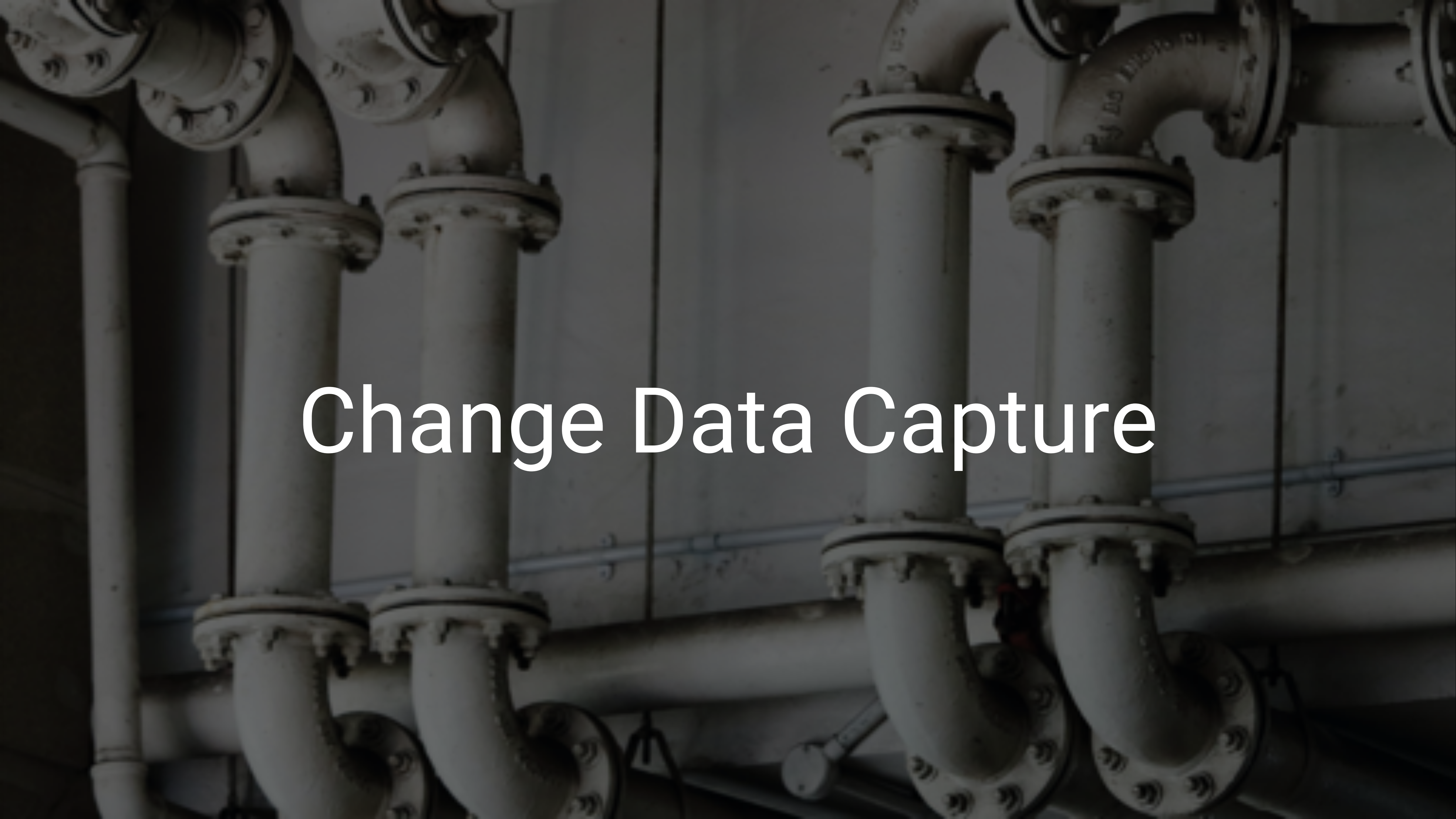
Mateusz Dymiński

- Software Developer at Nokia
- 8+ exp with Java
- 5+ exp with Go
- One of the organizer GoWroc - Golang Wroclaw Meetup
- Page: https://mateuszdyminski.com
- Github: github.com/mateuszdyminski
- Twitter: @m_dyminski
- LinkedIn: linkedin.com/in/mdyminski

Agenda

- Change Data Capture
- Turning on CDC in DB
- CDC Tooling
- Architecture Concepts
    - Streaming
    - Microservices Communication
    - Outbox Pattern
    - Strangler Pattern
- CDC challenges
- Summary

github.com/mateuszdyminski/cdc



Change Data Capture

Change Data Capture

change data capture (**CDC**) is a set of software design patterns used to determine and track the data that has changed so that action can be taken using the changed data.

Types of Change Data Capture

- Trigger-based CDC
- Query-based (polling) CDC
- Log-based (WAL, Redo logs, Binlog) CDC

Trigger-based Change Data Capture

- DBs provide trigger functions to performing user-defined actions once events, like insertions of data, occur
- Trigger could copy records which have changed in a separate table used as an event queue
- It requires recurring polling of the *event* table
- Vendor-specific code for implementing trigger

Query-based Change Data Capture

- Every X seconds we need to query DB to get changes
- Slows down the DB due to often heavy queries
- Requires recurrent polling of the table
- Determining the difference between two data sets is a compute-heavy operation that makes frequent executions impossible

Log-based Change Data Capture

- Based on WAL, Redo logs, Binlog
- All data changes are captured
- No polling or overhead
- Transparent – no need to touch any of old/legacy applications
- Event with changes is sent to subscribers
- Reactive approach

SQL

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    firstname TEXT NOT NULL,
    lastname TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);
```

Following INSERT

```sql
INSERT INTO users(firstname, lastname) VALUES('Johny', 'Rambo');
```

Produces following EVENT

```json
{
  "change": [
    {
      "kind": "insert",
      "schema": "public",
      "table": ”users",
      "columnnames": [”id”, ”firstname", ”lastname", "created_at"],
      "columntypes": [
         "integer",
         ”text",
         ”text",
         "timestamp without time zone"
      ],
      "columnvalues": [1, ”Johny", ”Rambo", "2020-09-15 11:58:28.988414"]
    }
  ]
}
```

Following UPDATE

```sql
UPDATE users SET lastname = 'Kowalski' WHERE id = 1;
```

Produces following EVENT

```json
{
  "change": [
    {
      "kind": "update",
      "schema": "public",
      "table": "users",
      "columnnames": ["id", "firstname", "lastname", "created_at"],
      "columntypes": ["integer", "text", "text", "timestamp without time zone"],
      "columnvalues": [1, "Johny", "Kowalski", "2020-09-15 11:58:28.988414"],
      "oldkeys": {
        "keynames": ["id"],
        "keytypes": ["integer"],
        "keyvalues": [1]
      }
    }
  ]
}
```

Following DELETE

```sql
DELETE FROM users WHERE id = '1';
```

Produces following EVENT

```json
{
  "change": [
    {
      "kind": "delete",
      "schema": "public",
      "table": "users",
      "oldkeys": {
        "keynames": ["id"],
        "keytypes": ["integer"],
        "keyvalues": [1]
      }
    }
  ]
}
```

Turning on CDC in DB

To allow CDC(log-based) in Postgres

- Set wal_level = logical
- Set max_replication_slots > 1
- Postgres version > 9.4

To allow CDC(log-based) in MySQL

- Run MySQL with binlog-format set to row

```
[mysqld]
log_bin             = /var/log/mysql/mysql-bin.log
max_binlog_size     = 100M
binlog-format       = row
```

CDC on Cloud Providers

- In most cases you can use CDC with DBs provided by cloud providers
- There are some problems with CDC on GCE

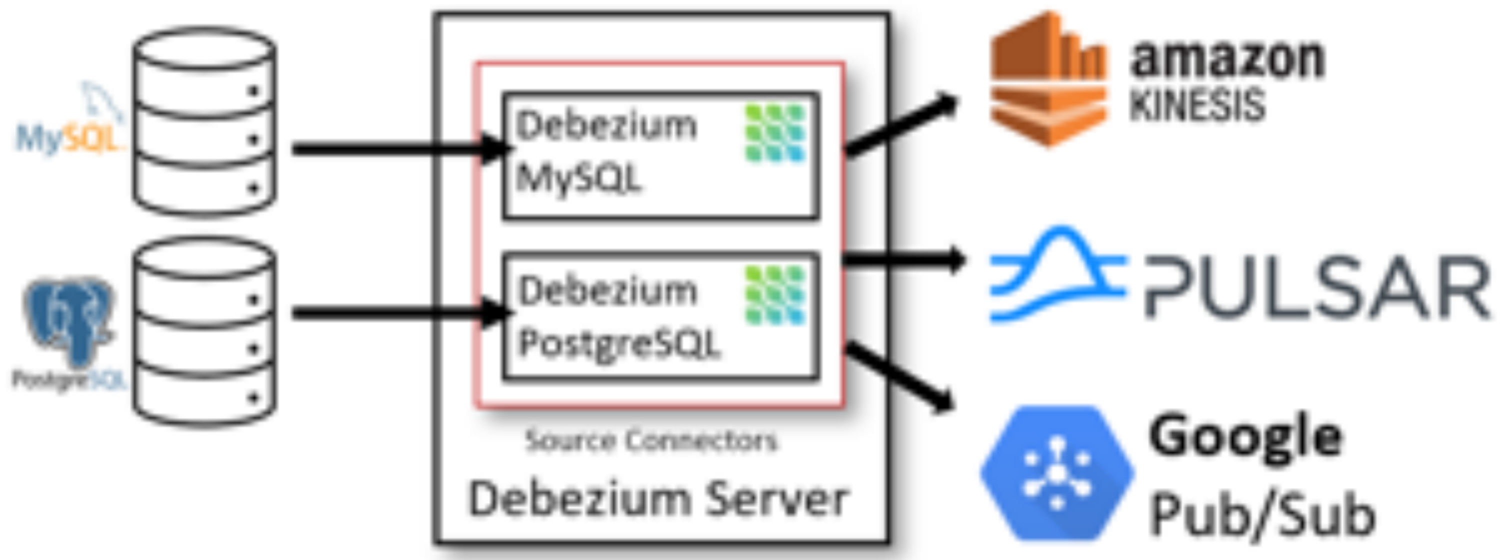
Demo

CDC Tooling

Debezium

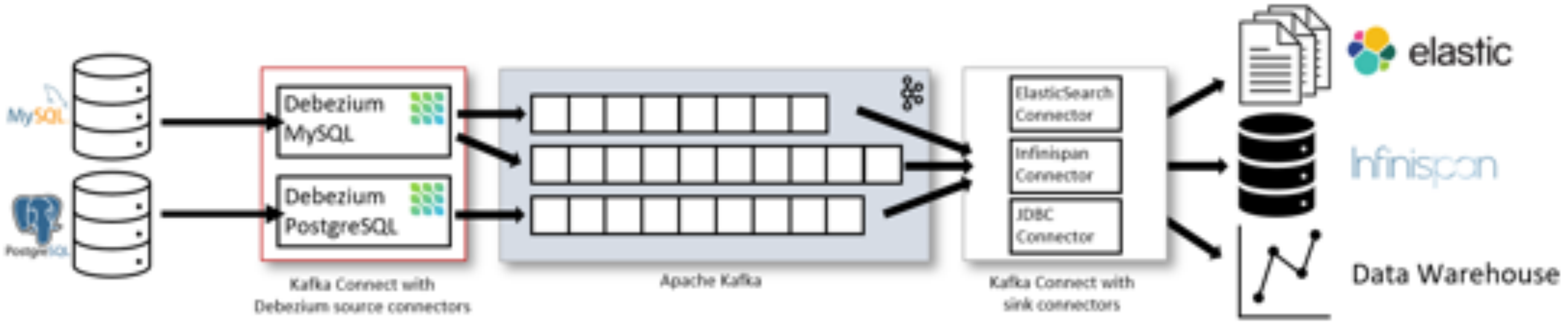
- The most popular CDC platform
- CDC with multiple databases support
  - Based on transactional logs
  - Snapshotting
- 3 modes of operation:
  - Kafka connect
  - Debezium server
  - Embedded
- Opensource: https://github.com/debezium/debezium
- Created by RedHat
- Battle tested
- But https://github.com/alibaba/canal has 4x stars on Github

Debezium – Server



Source: https://debezium.io/documentation/reference/1.2/architecture.html

Debezium – Kafka Connect



Source: https://debezium.io/documentation/reference/1.2/architecture.html

Debezium – Embedded

- Library embedded into your Java app
- Consuming change events within your application itself
- No need to deploy Kafka

Debezium – supported databases

- MongoDB
- MySQL
- PostgreSQL
- SQL Server
- Oracle (Incubating)
- Db2 (Incubating)
- Cassandra (Incubating)

APACHE kafka®

A distributed streaming platform

Kafka

Apache describes Kafka as a distributed streaming platform that lets us:

- Publish and subscribe to streams of records.
- Store streams of records in a fault-tolerant way.
- Process streams of records as they occur.

Debezium alternatives

- https://github.com/zendesk/maxwell
- https://github.com/airbnb/SpinalTap
- https://github.com/Yelp/mysql_streamer

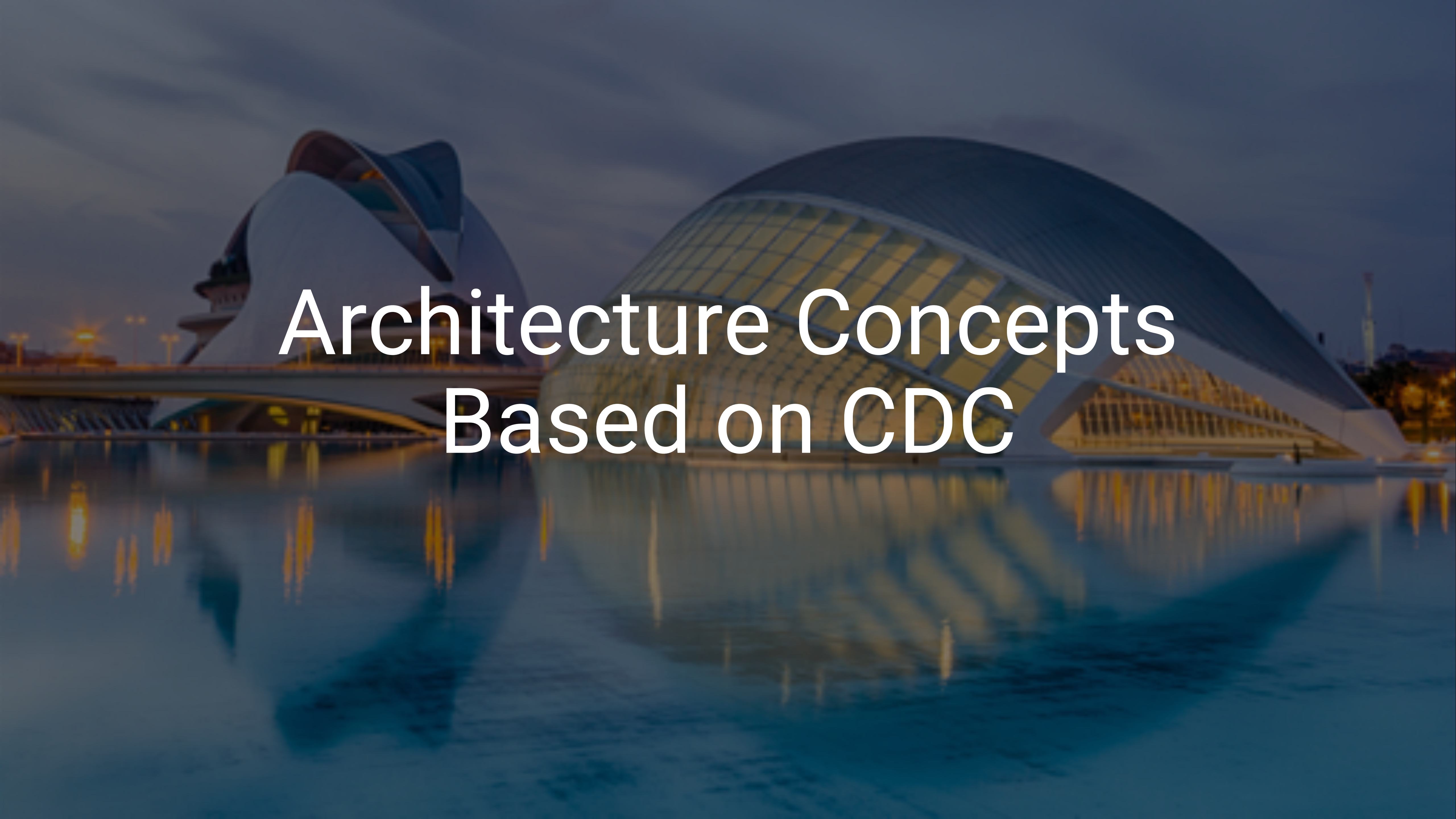
More alternatives - MySQL

- aesop https://github.com/Flipkart/aesop
- databus https://github.com/linkedin/databus
- FlexCDC http://github.com/greenlion/swanhart-tools/
- Lapidus https://github.com/JarvusInnovations/lapidus
- mypipe https://github.com/mardambey/mypipe
- MySqlCdc https://github.com/rusuly/MySqlCdc
- mysql-binlog-connector-java https://github.com/shyiko/mysql-binlog-connector-java
- oltp-cdc-olap https://github.com/xmlking/nifi-examples/tree/master/oltp-cdc-olap
- Open Replicator https://code.google.com/p/open-replicator/
- Canal https://github.com/alibaba/canal
- python-mysql-replication https://github.com/noplay/python-mysql-replication
- recordbus https://github.com/pyr/recordbus
- Tungsten Replicator https://github.com/continuent/tungsten-replicator
- wombat https://github.com/TiVo/wombat
- kafka-mysql-connector https://github.com/wushujames/kafka-mysql-connector
- php-mysql-replication https://github.com/krowinski/php-mysql-replication
- StreamSets Data Collector https://streamsets.com/products/sdc/

Source: https://github.com/wushujames/mysql-cdc-projects/wiki

New player on the market - DBLog

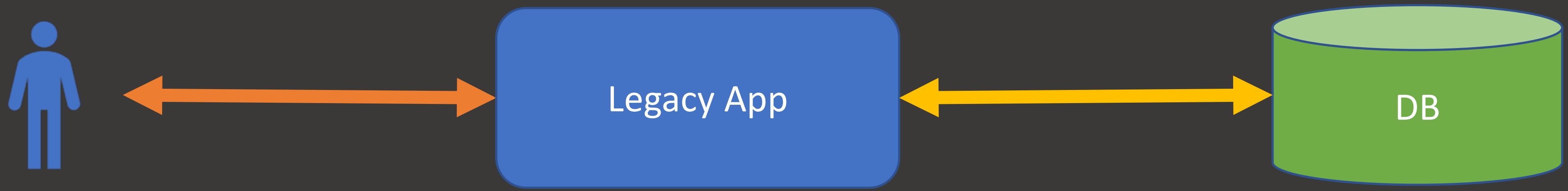
- New alternative for Debezium created by Netflix
- Not opensourced yet, but should be in 2020
- Some features:
  - Dumps can be taken anytime
  - No locks on tables
  - Designed to perform Snapshots quite often
- More info: https://netflixtechblog.com/dblog-a-generic-change-data-capture-framework-69351fb9099b



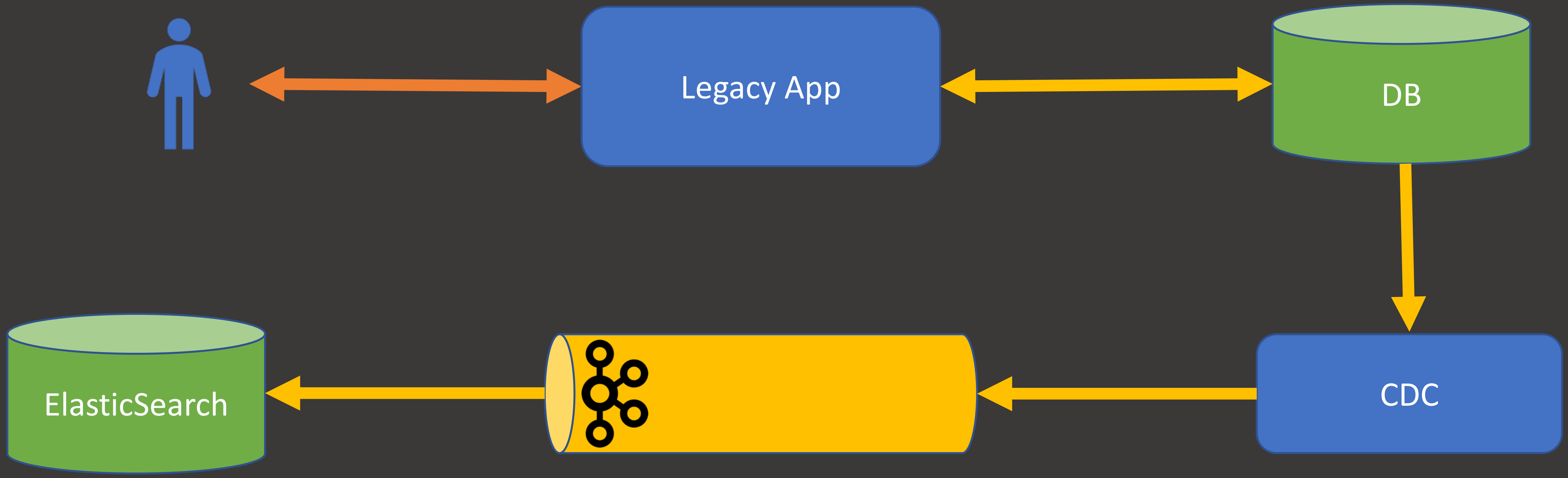
Architecture Concepts Based on CDC

Streaming

Streaming

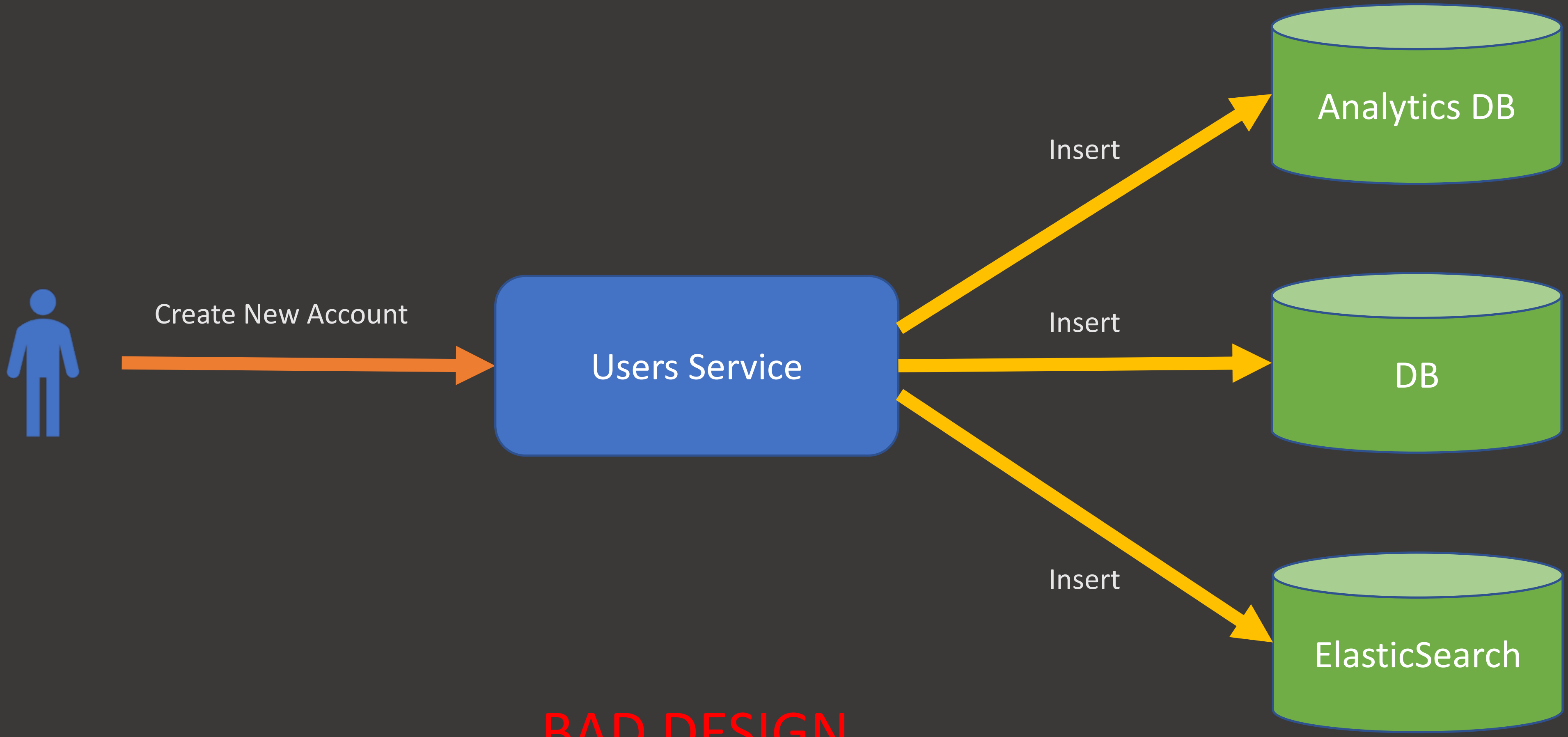


Streaming



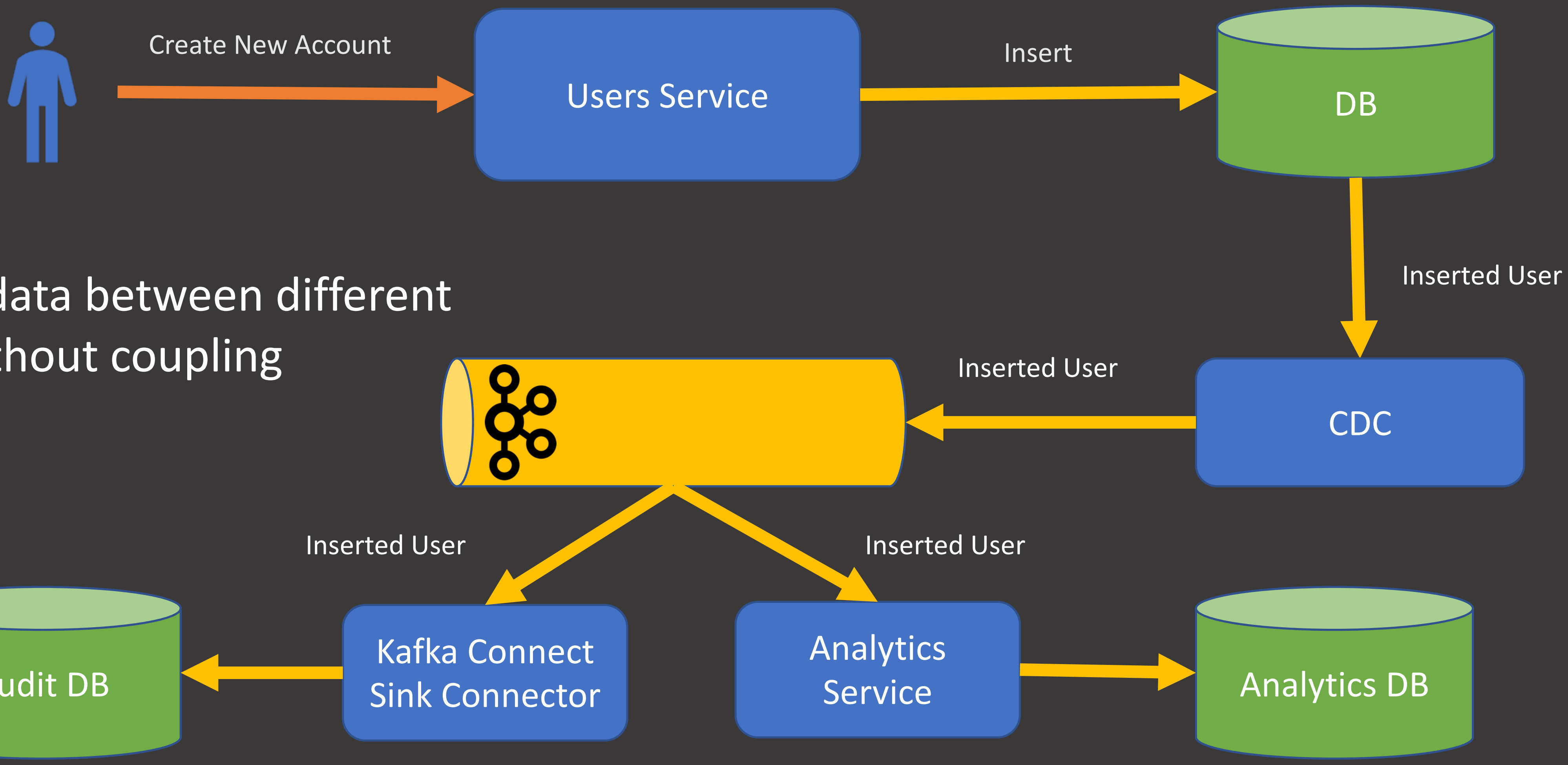
Microservices Communication

Data Synchronization

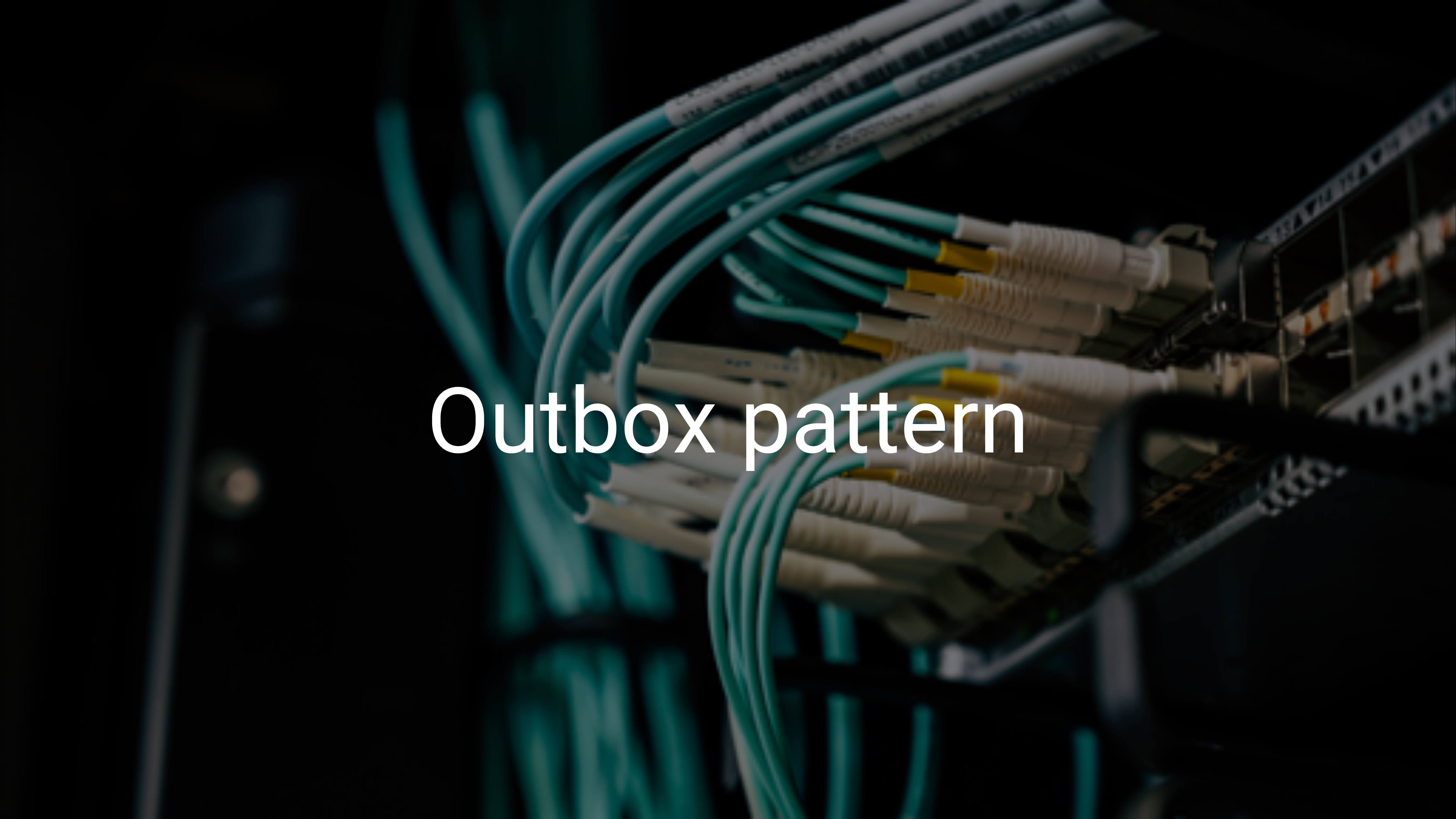


BAD DESIGN

Data Synchronization



Propagate data between different services without coupling

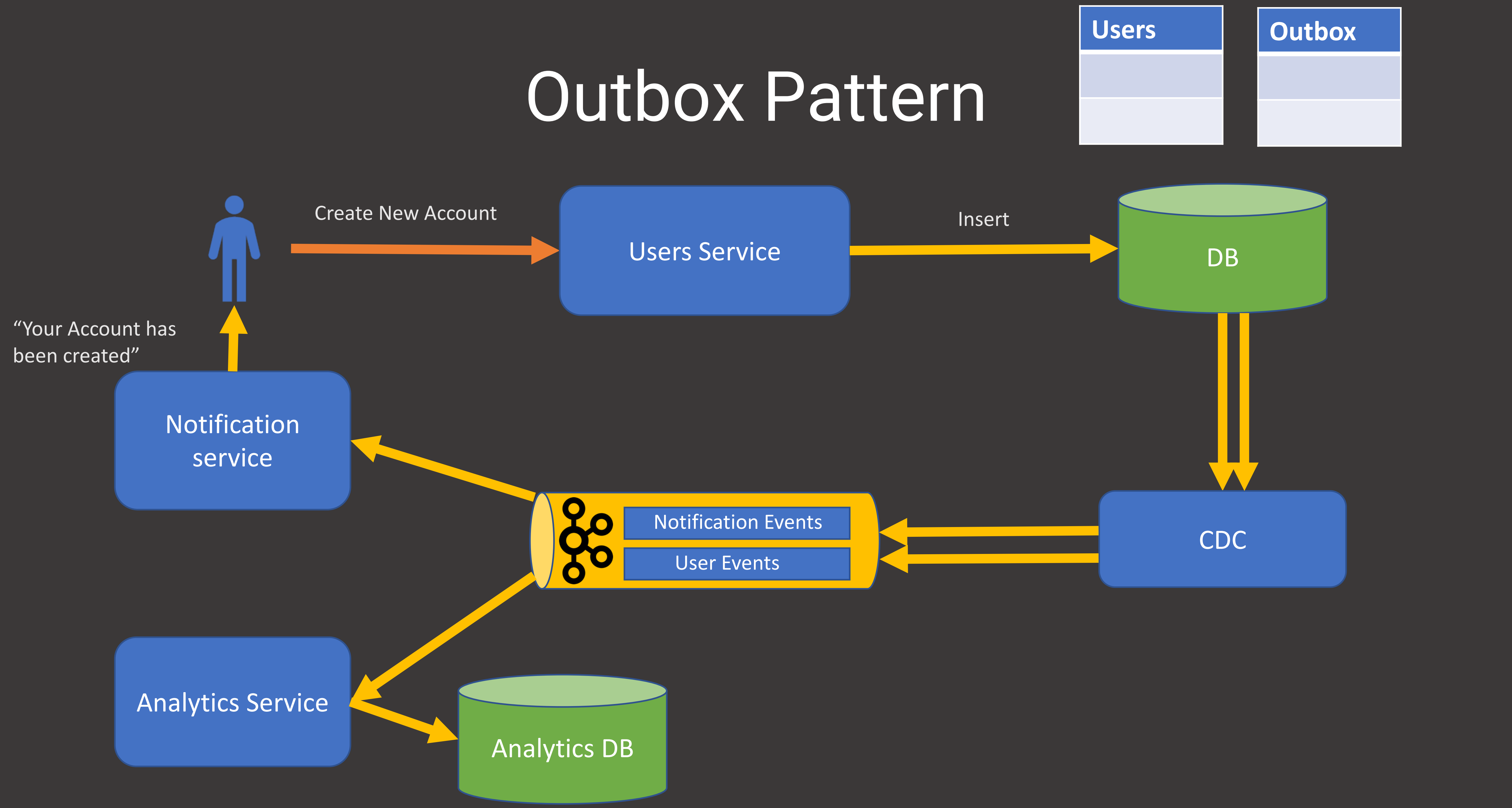


Outbox pattern

Outbox pattern

- Problem: service needs to update the database **and** send messages/events and preserve the consistency of data
- Extra table in DB is created - Outbox
- Insert into both tables – Outbox and desired is in single transaction
- The outbox pattern is a great way for propagating data amongst different microservices.
- By only modifying a single resource

Outbox Pattern

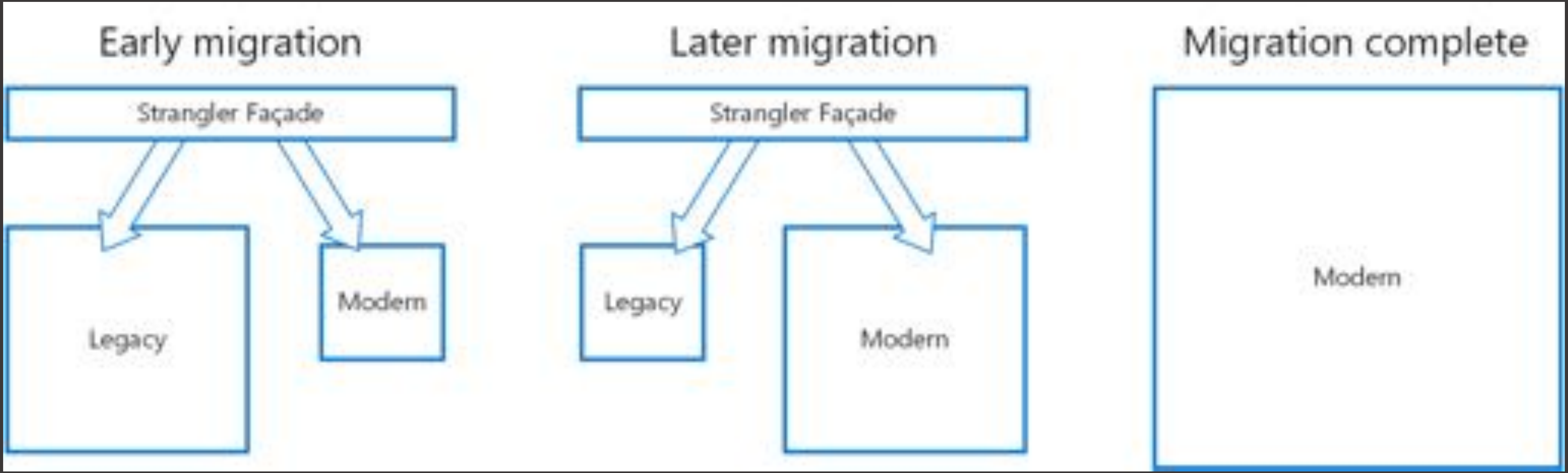


Strangler pattern

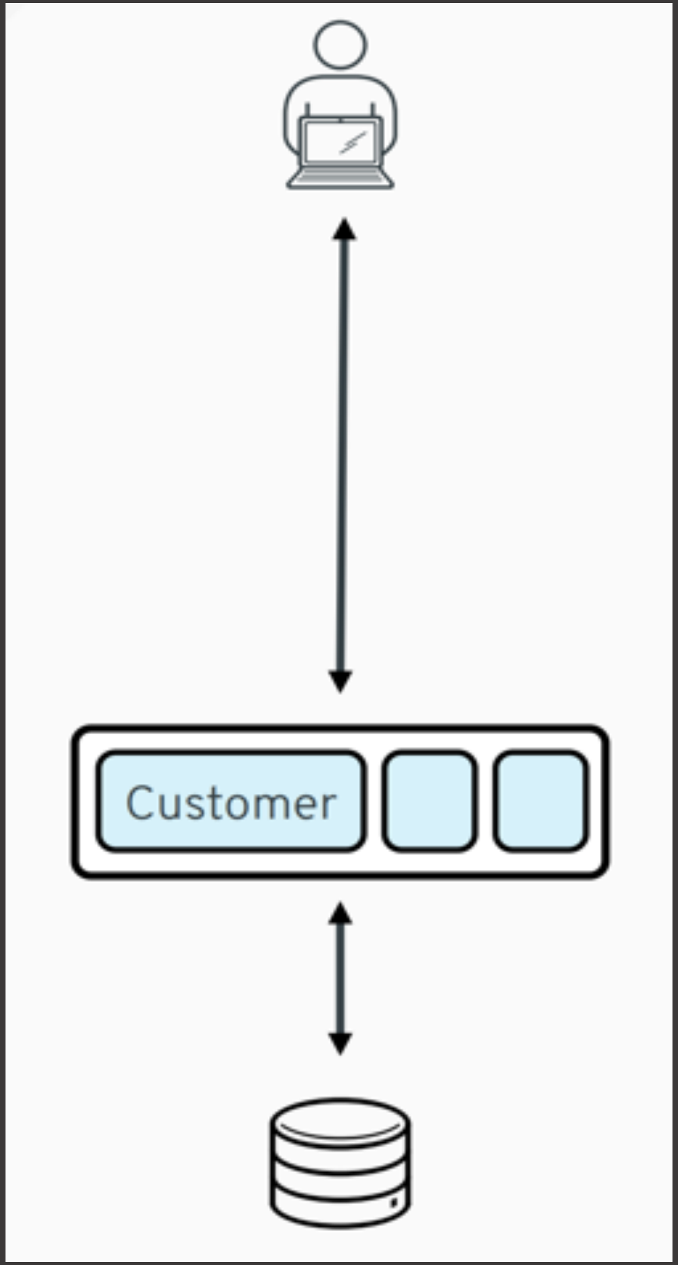
Strangler Pattern

- Way of migrating a legacy system incrementally
- API Gateway as the entry to the system – router
- Step by Step (like strangler fig) remove old components to new architecture
- Great pattern for huge legacy projects
- The legacy and microservices have to run side by side
- The same data might be modified by both systems

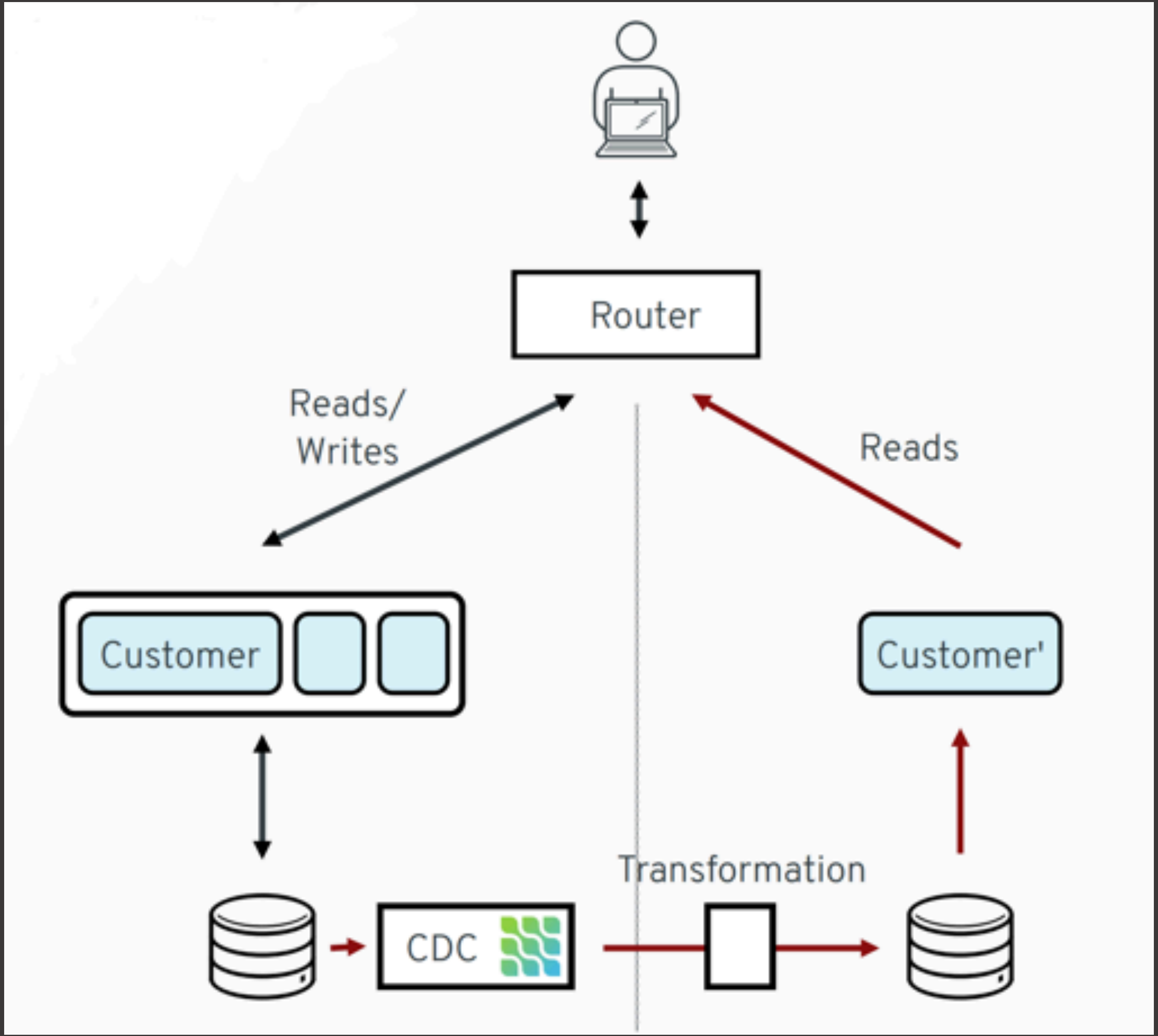
Strangler Pattern



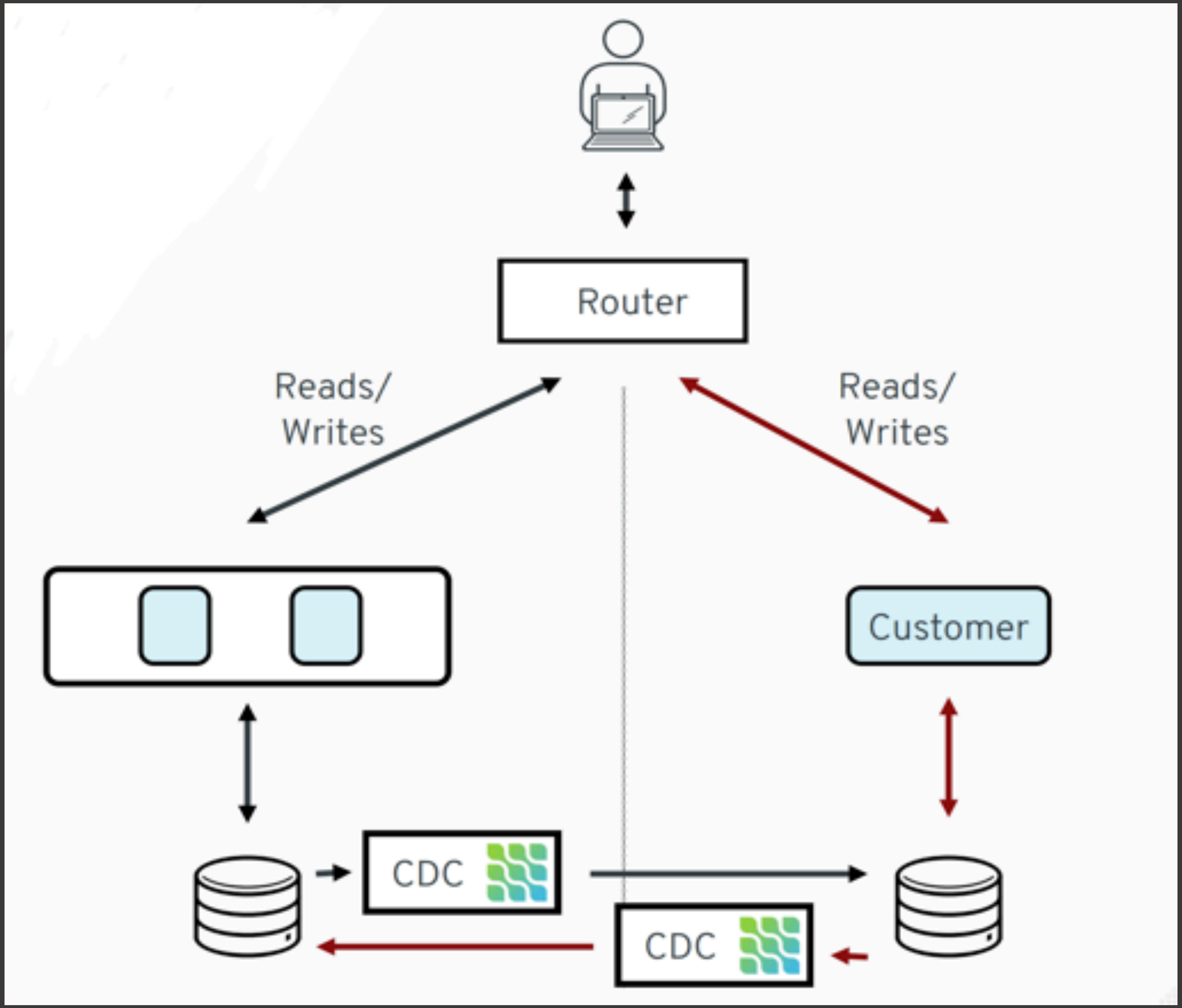
Source: https://docs.microsoft.com/pl-pl/azure/architecture/patterns/strangler



Source: https://speakerdeck.com/gunnarmorling/practical-change-data-streaming-use-cases-with-apache-kafka-and-debezium-qcon-san-francisco-2019



Source: https://speakerdeck.com/gunnarmorling/practical-change-data-streaming-use-cases-with-apache-kafka-and-debezium-qcon-san-francisco-2019



Source: https://speakerdeck.com/gunnarmorling/practical-change-data-streaming-use-cases-with-apache-kafka-and-debezium-qcon-san-francisco-2019

CDC Challenges

Challenges

- MySQL – not all DELETE events are visible in CDC – link
- LogCompaction on Kafka
- Blocking write traffic by locking tables.
- Missing ability to trigger dumps on demand.
- Stopping log event processing while processing a dump.

Summary

Takeaways

- CDC concept is very easy to grasp
- CDC enables features like:
  - Replication
  - Streaming
  - Auditing
  - Decouples services
- Debezium is most mature framework on the market, but have a look at DBLog

Q&A