wepay
a CHASE company

The Why's and How's of Database Streaming

Joy Gao
Senior Software Engineer @ WePay

Agenda

1. The Beauty of Change Data Capture

2. Real-World Example: Streaming MySQL

3. Future Challenge: Streaming Cassandra

Agenda

1. The Beauty of Change Data Capture

2. Real-World Example: Streaming MySQL

3. Future Challenge: Streaming Cassandra

BigQuery Overview

BigQuery

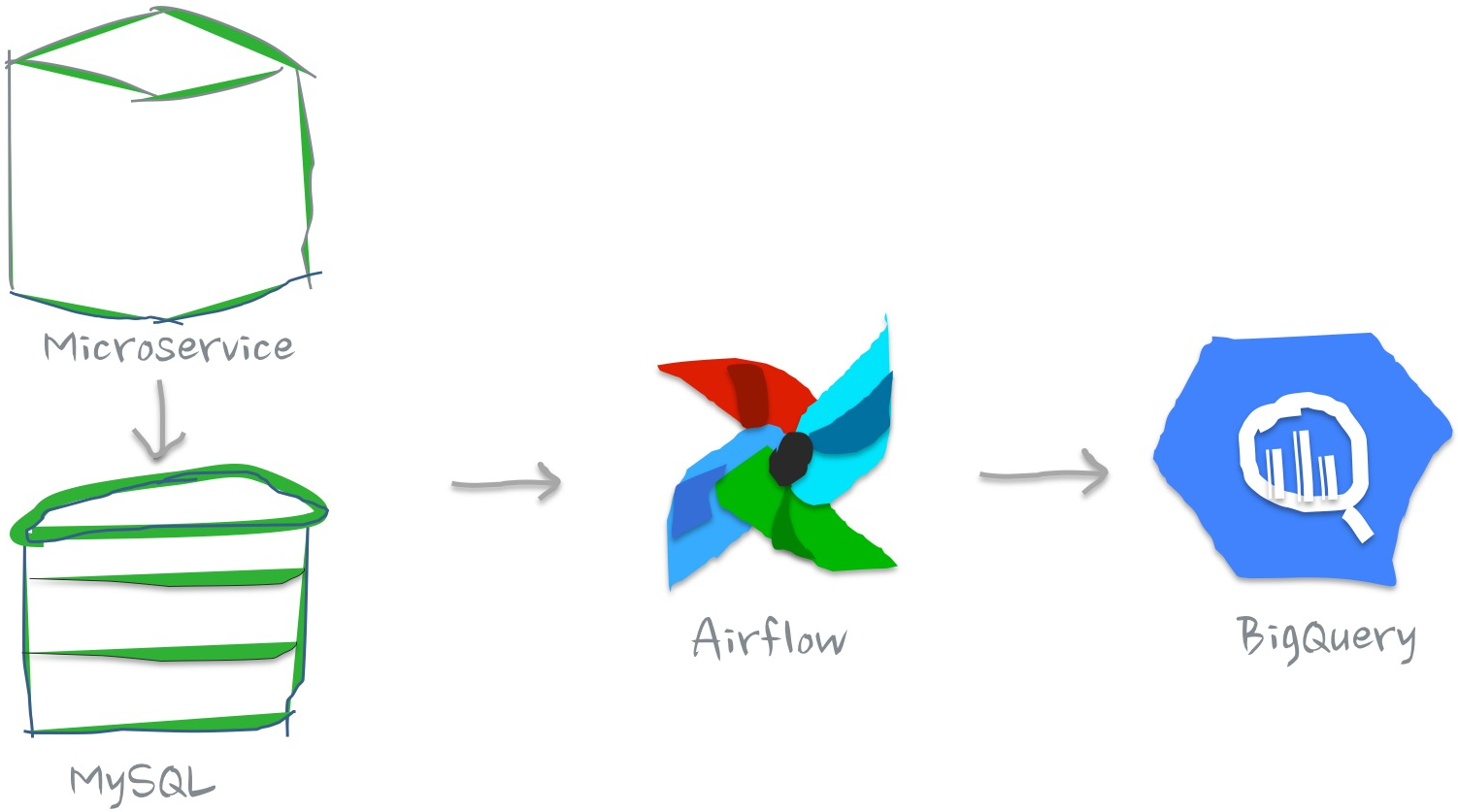
Google's Serverless DWH

ANSI-Compliant SQL

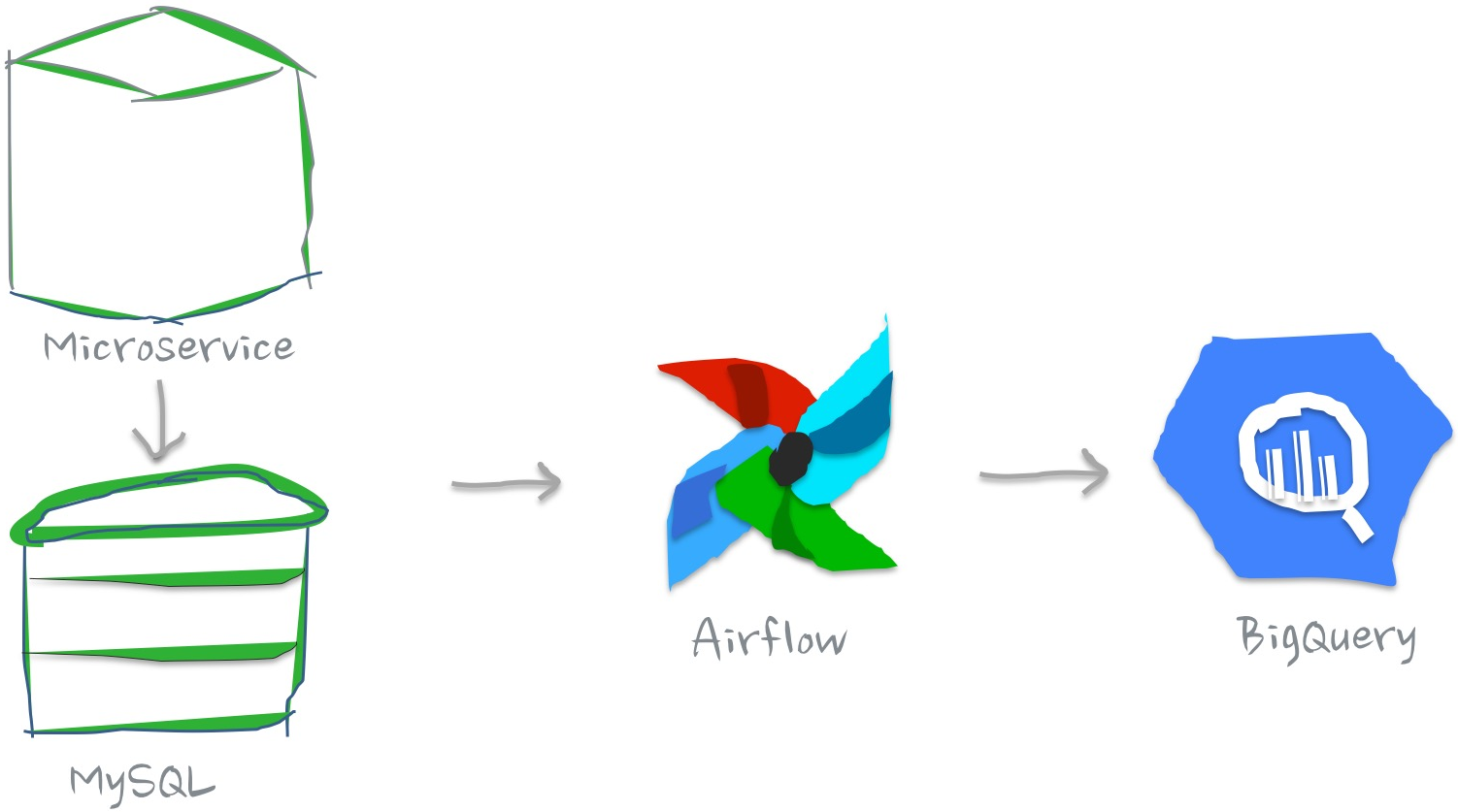
Nested & Repeated Structures

Virtual Views

WePay's Traditional Batch ETL Pipeline

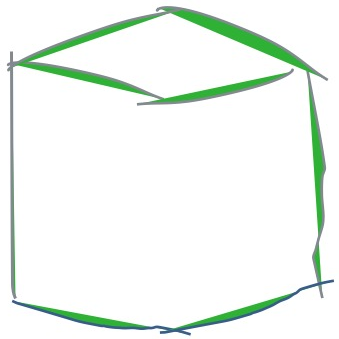


WePay's Traditional Batch ETL Pipeline

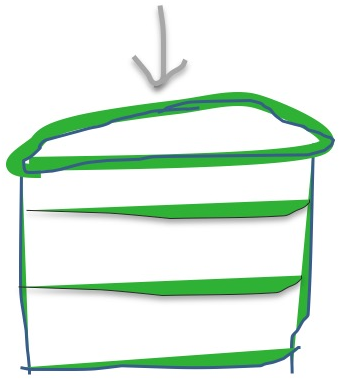


- High latency
- Huge number of jobs
- No hard deletes
- Error-prone
- Manual schema update

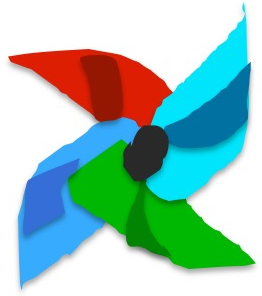
WePay's Data Ecosystem



Microservice

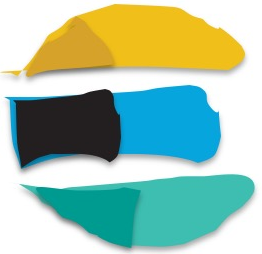


MySQL



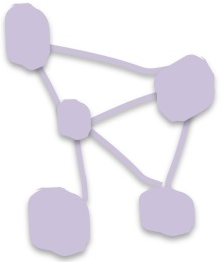
Airflow

Redis

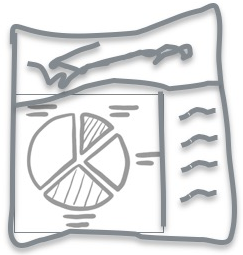


Elasticsearch

BigQuery



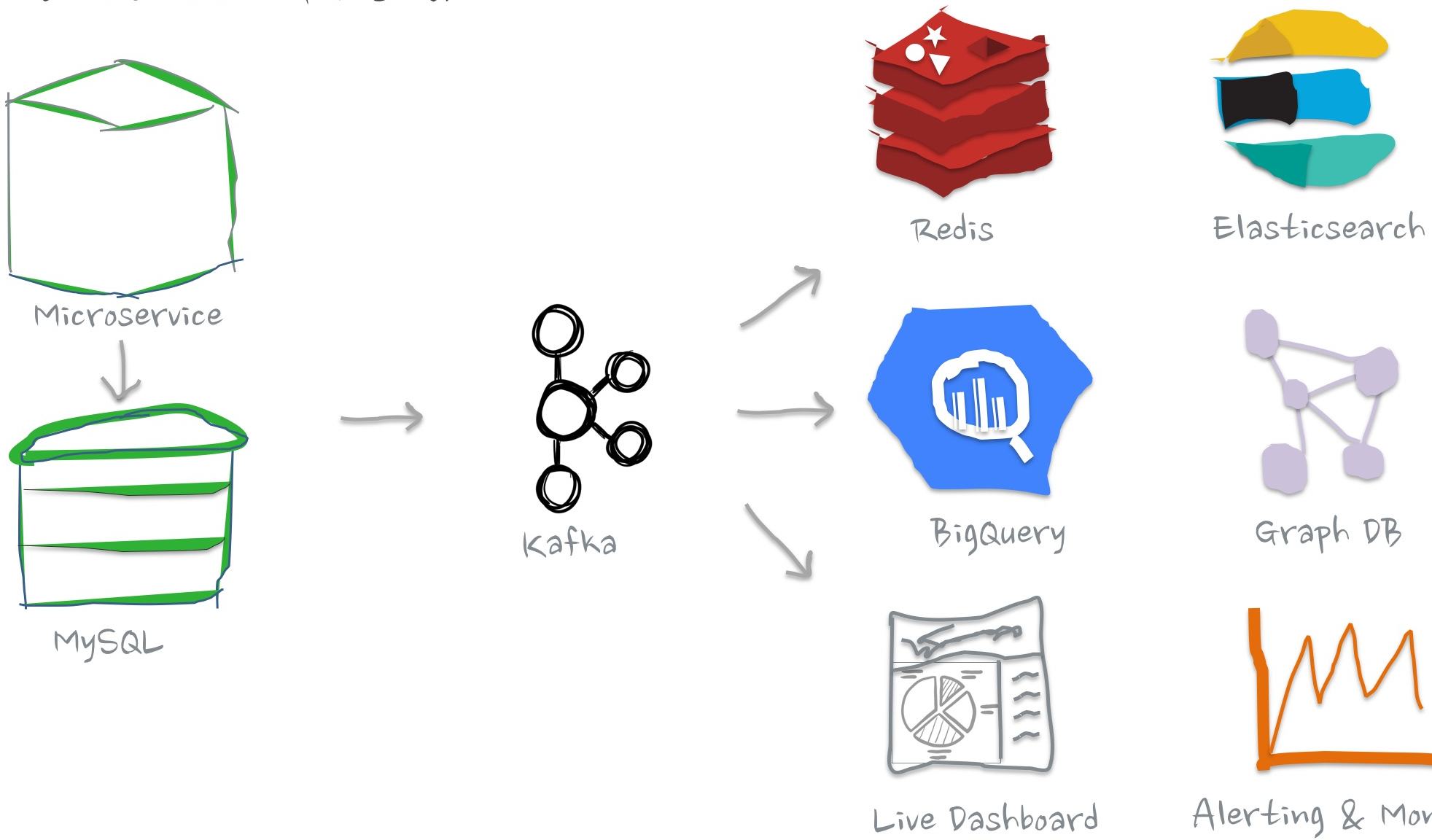
Graph DB



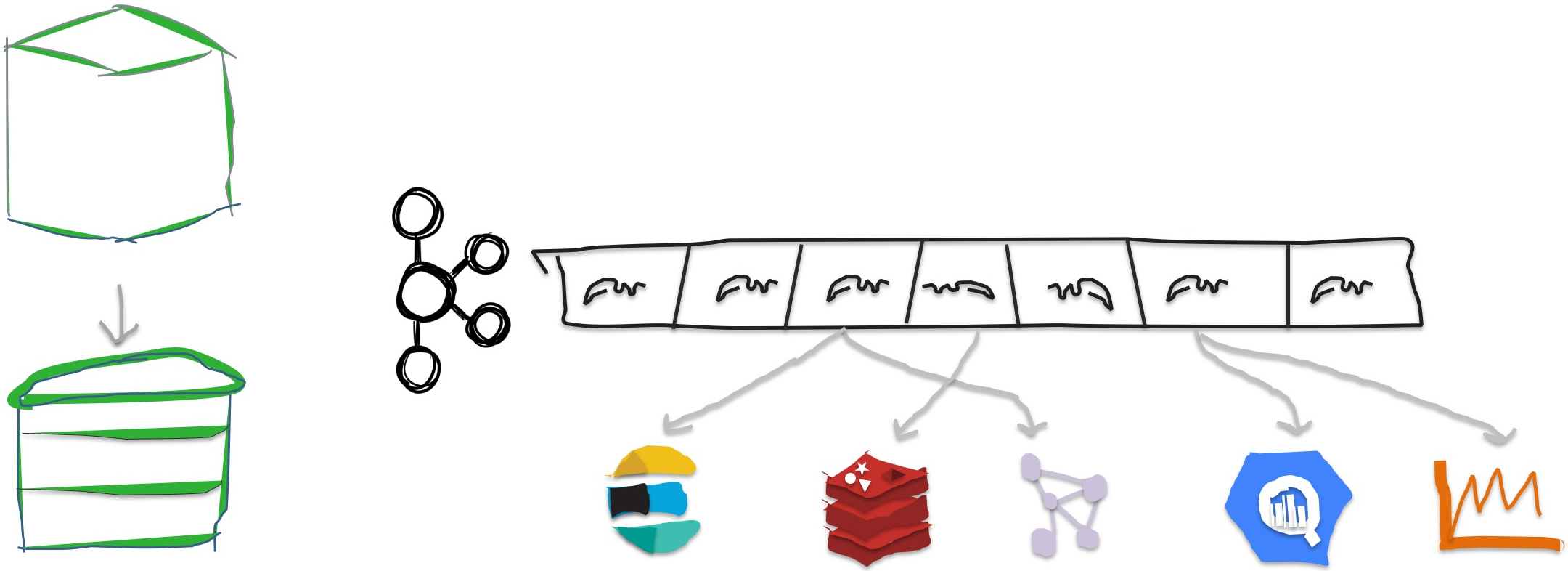
Live Dashboard

Alerting & Monitoring

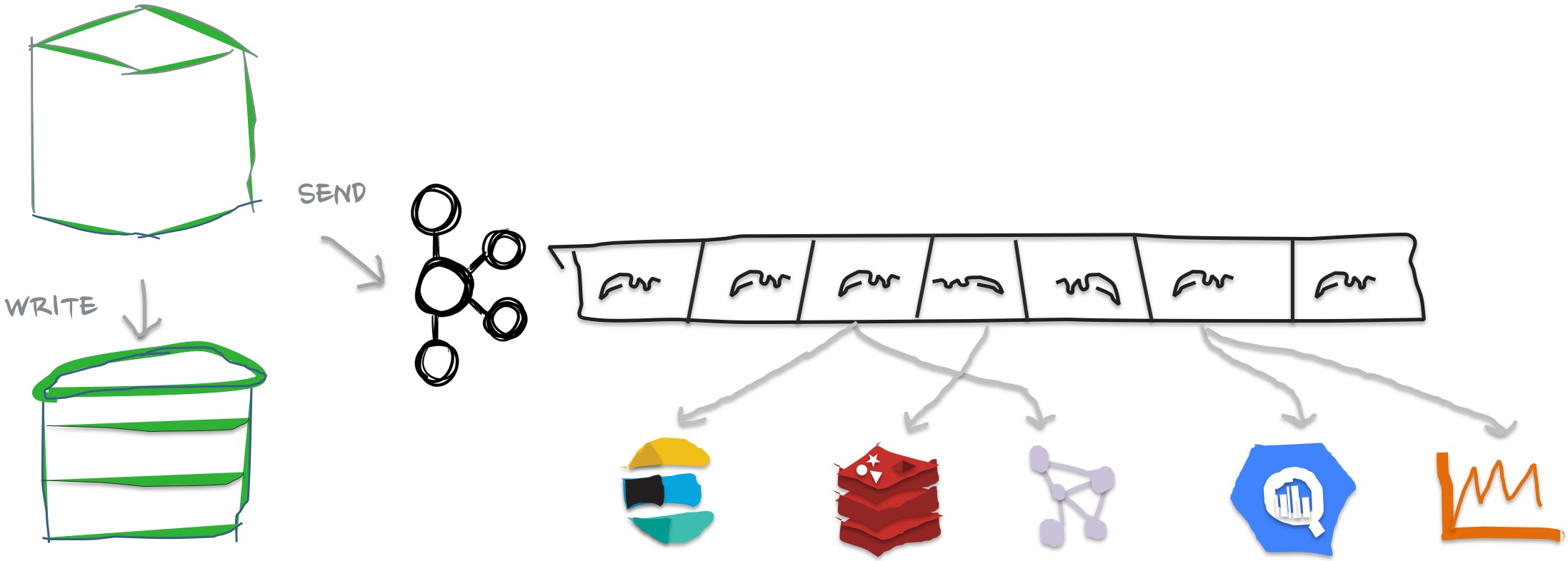
Kafka to the Rescue



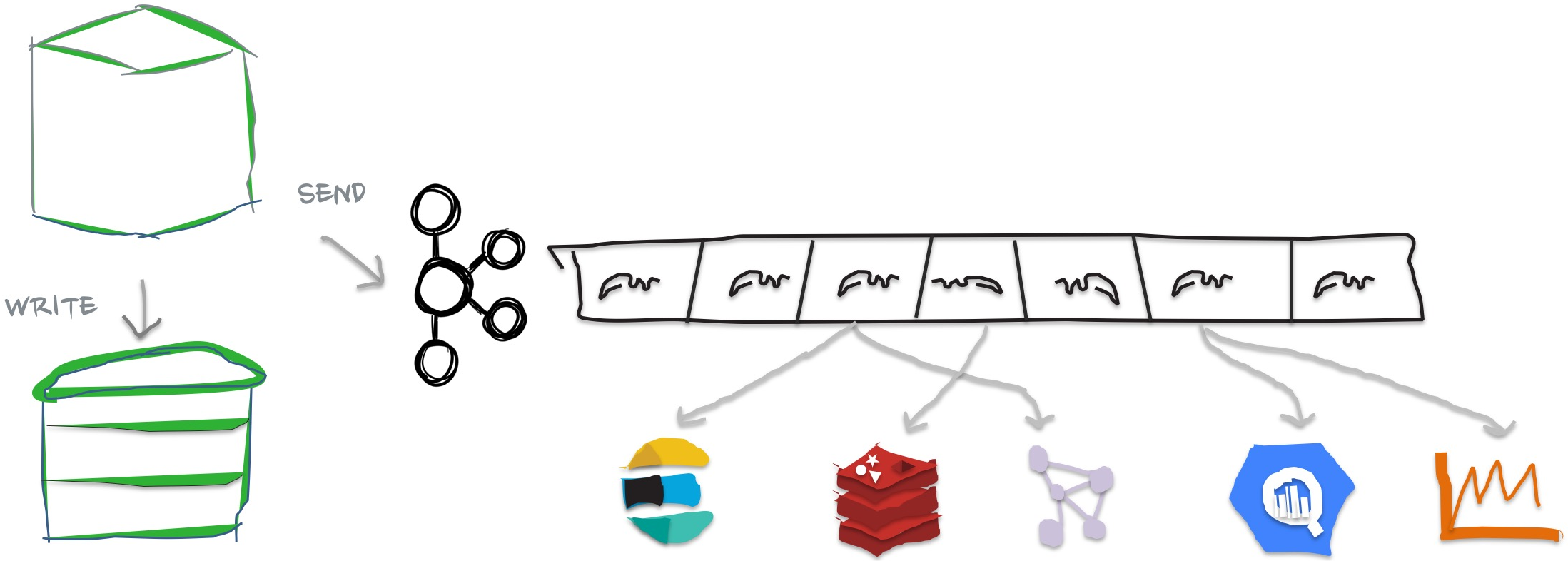
Getting Data to Kafka



Option One: Double-Write

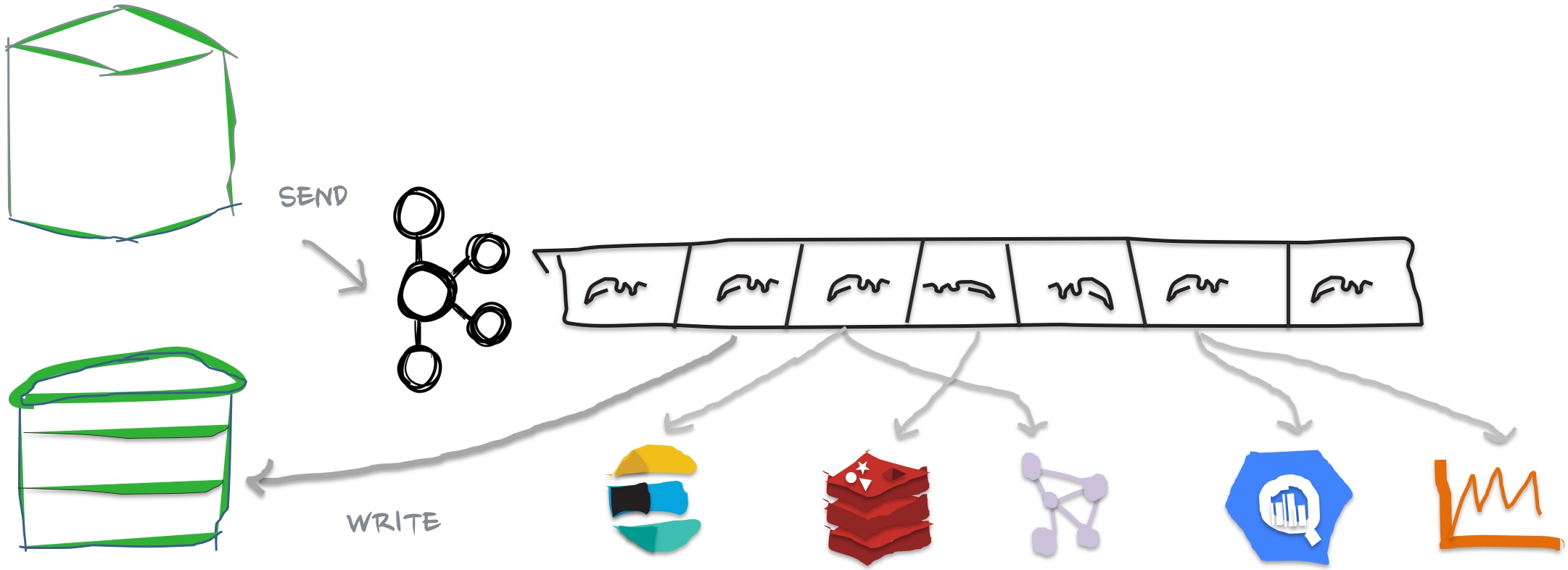


Option One: Double-Write

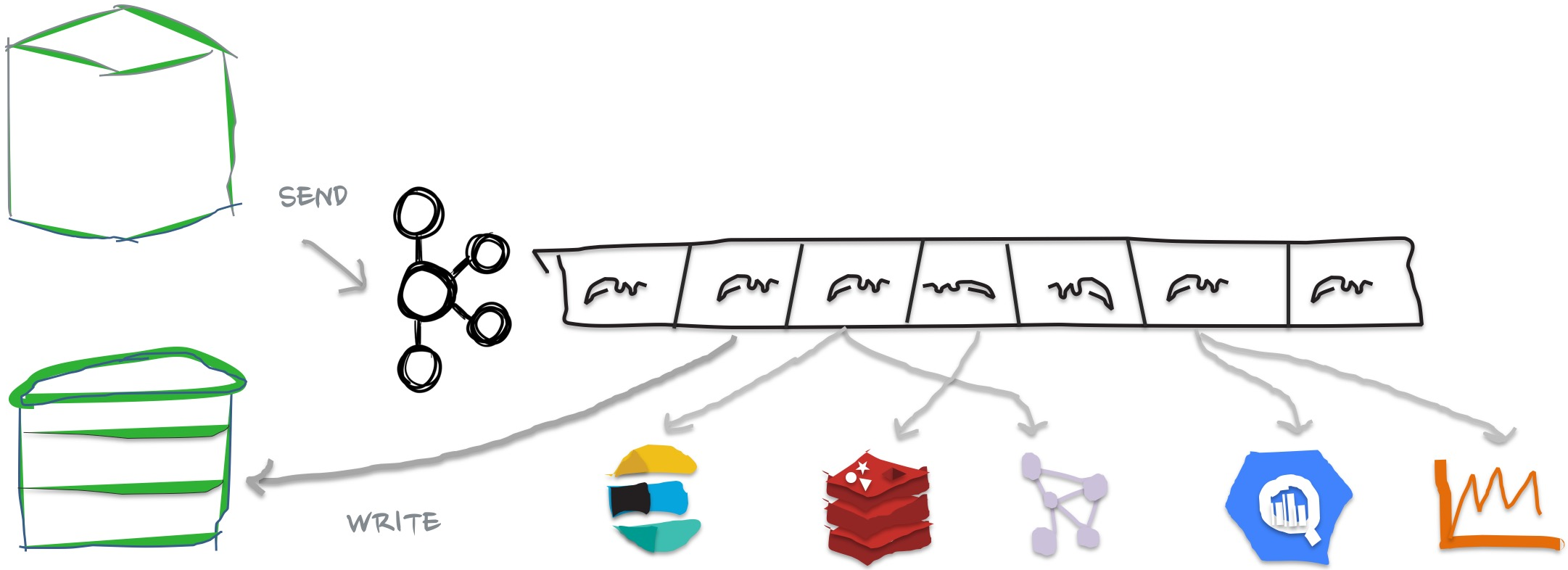


Distributed transaction tho ☹

Option Two: Event-Sourcing

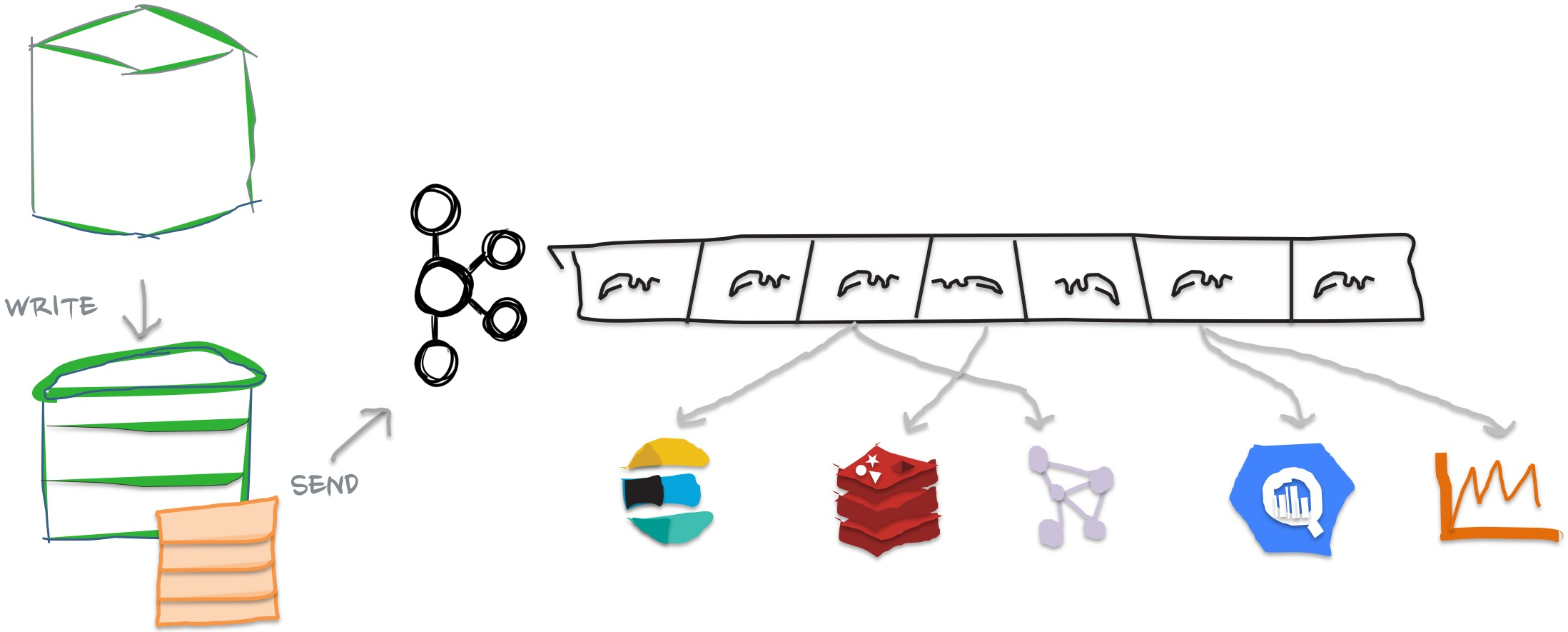


Option Two: Event-Sourcing



Violates read-your-writes consistency ☹

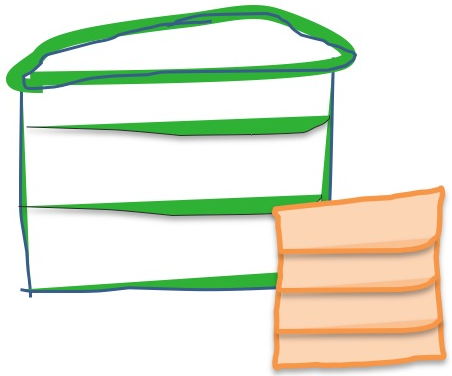
Option Three: Change Data Capture with WAL



Change Data Capture (CDC)

In databases, a design pattern that captures individual database changes into a stream of change events

Write-Ahead Logging (WAL)



Implemented in almost every database

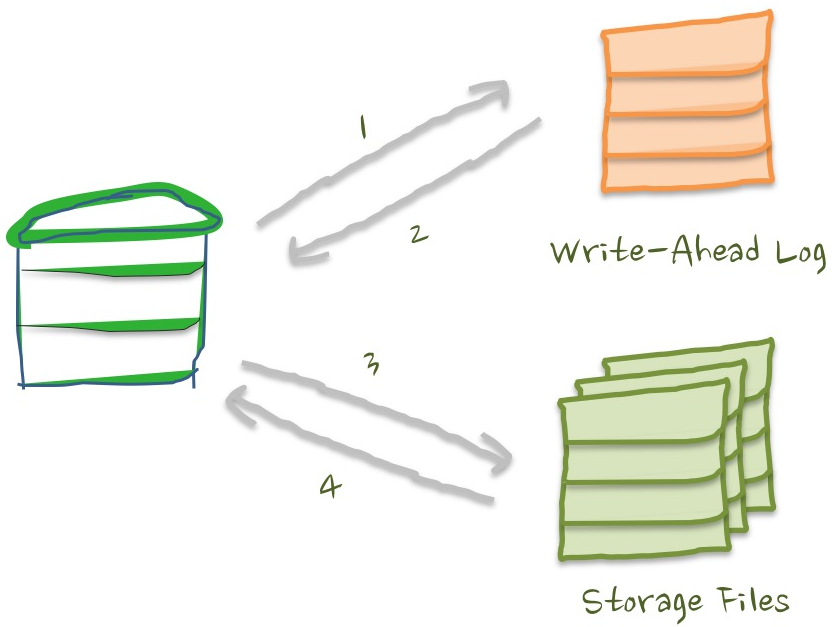
Writes are first recorded to a log file

Used for:

- crash recovery
- write performance
- streaming replication

Statement-based vs Row-Based Logging

Write-Ahead Logging (WAL)



Implemented in almost every database

Writes are first recorded to a log file

Used for:

- crash recovery
- write performance
- streaming replication

Statement-based vs Row-Based Logging

Write-Ahead Logging (WAL)

```
UPDATE customers SET name = "Alice"
    WHERE id = 1;

INSERT INTO customers (name, email)
    VALUES ("Bob", "bob@noreply.org);
```

```
1, "Alice", "alice@noreply.org"

2, "Bob", "bob@noreply.org"
```

Implemented in almost every database

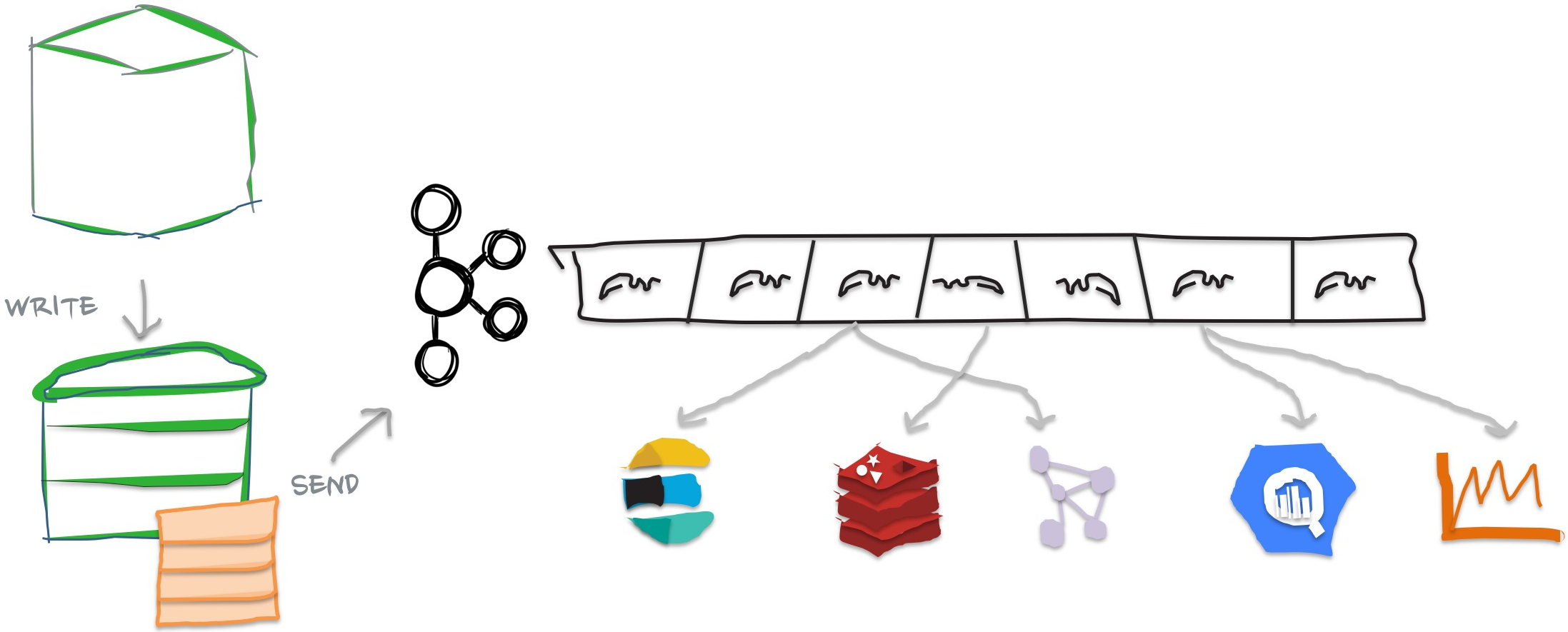
Writes are first recorded to a log file

Used for:

- crash recovery
- write performance
- streaming replication

Statement-based vs Row-Based Logging

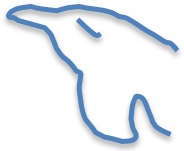
Option Three: Change Data Capture with WAL



Heck yeah! ☺

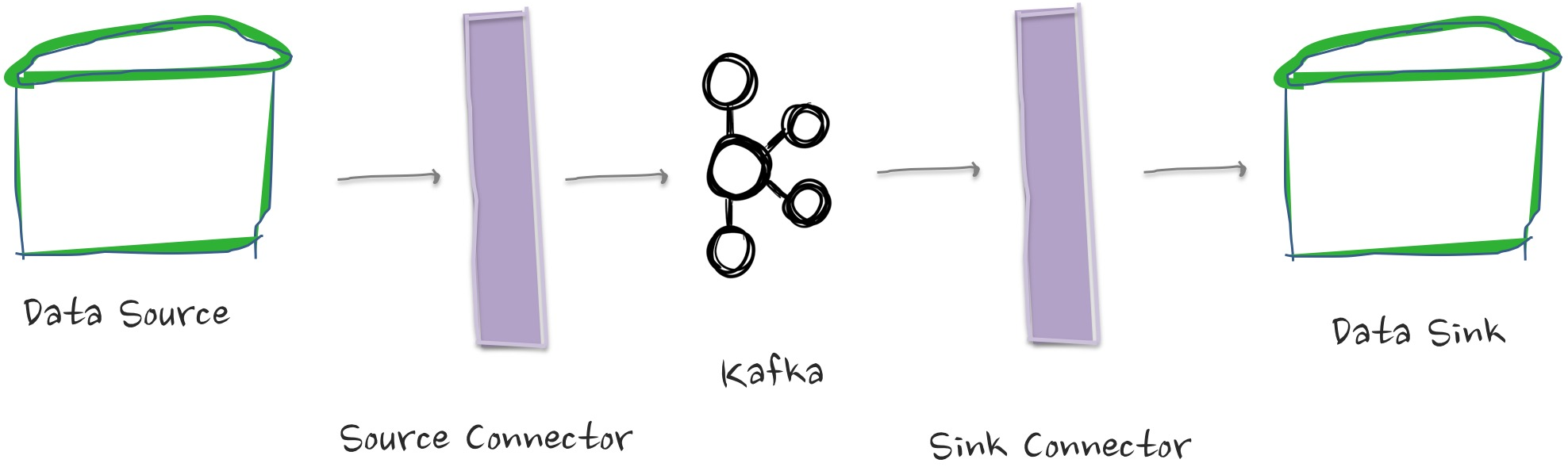
Agenda

1. The Beauty of Change Data Capture

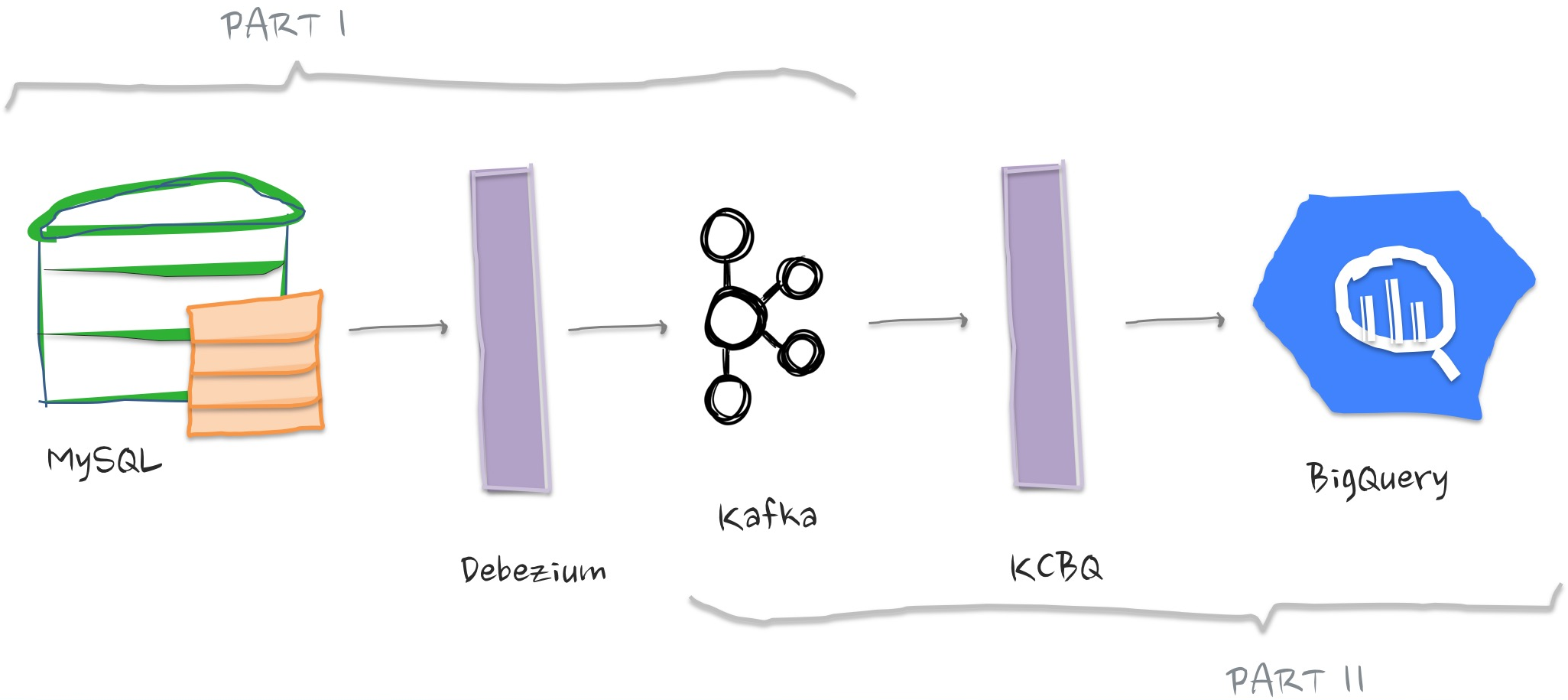
2. Real-World Example: Streaming MySQL 

3. Future Challenge: Streaming Cassandra

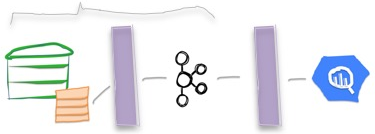
Kafka Connect Framework

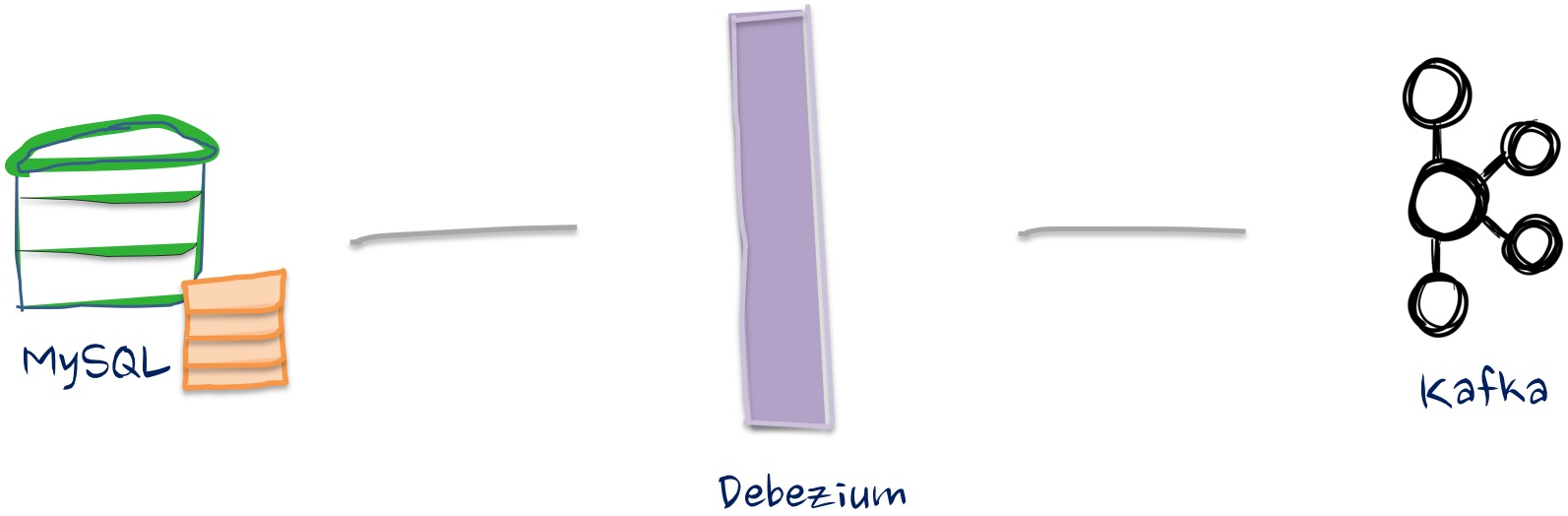


MySQL -> BigQuery (Bird's-Eye View)

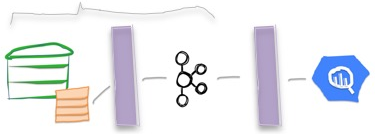


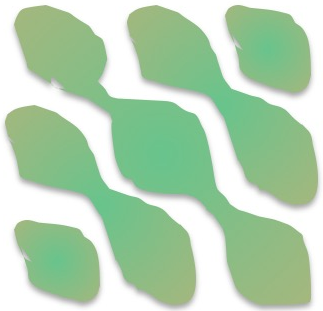
MySQL -> Kafka





Debezium Overview





Debezium

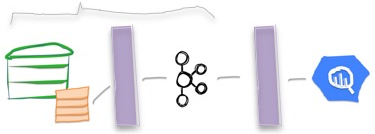
Open-Source distributed platform for CDC

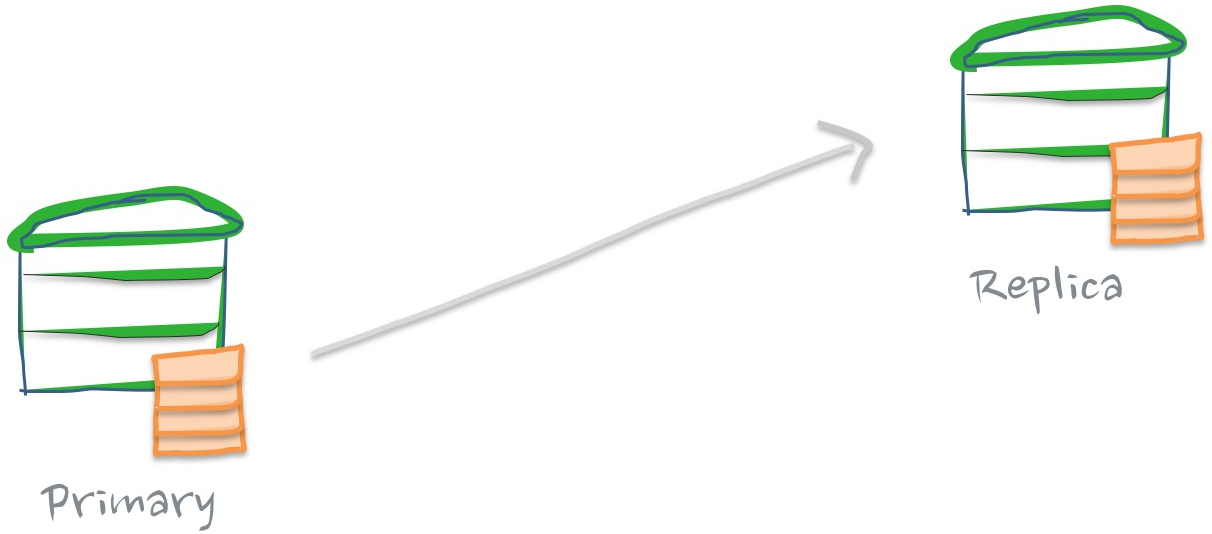
Records row-level changes via WAL

Guarantees at-least-once semantics

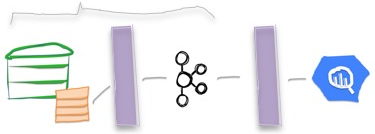
Supports MySQL, MongoDB, PostgreSQL, Oracle, SQL Server

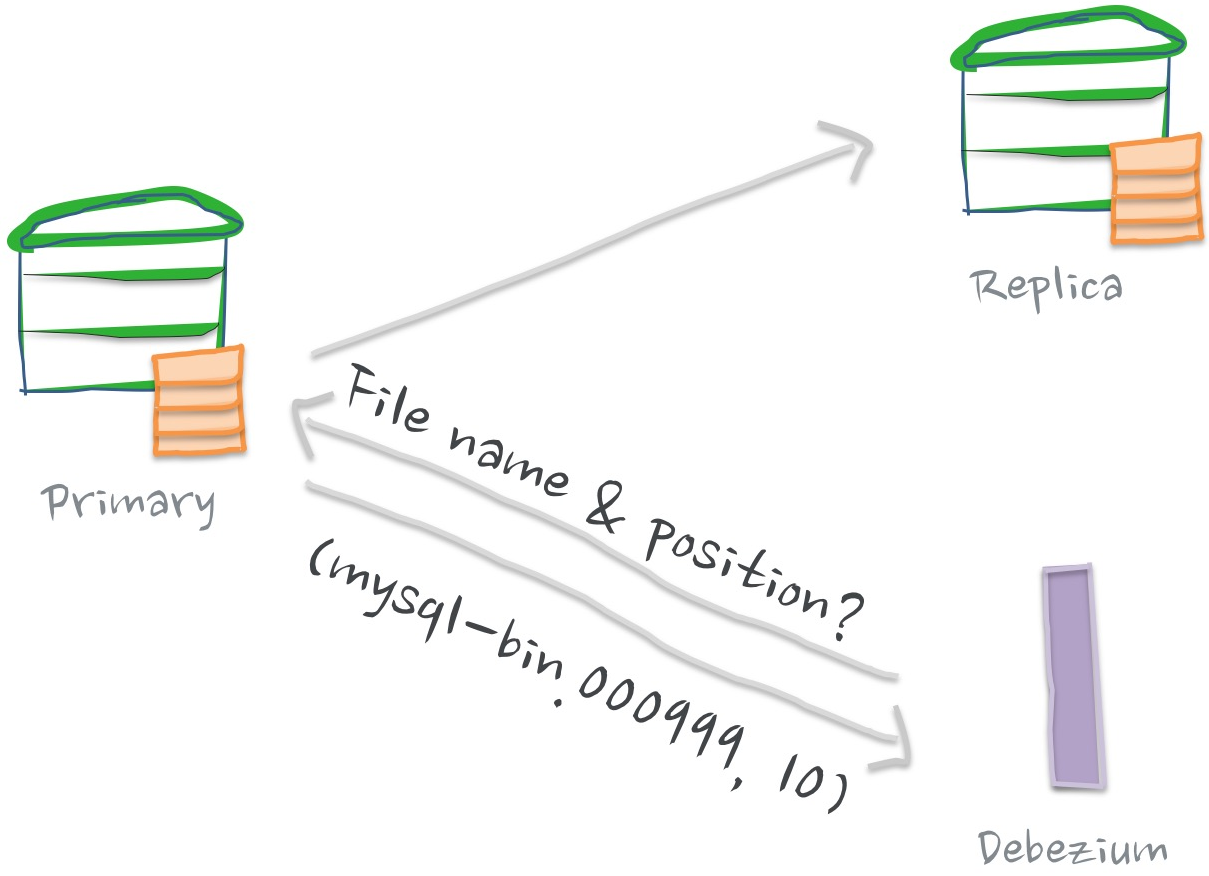
Debezium In Action





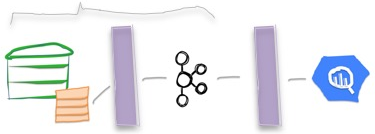
Debezium In Action

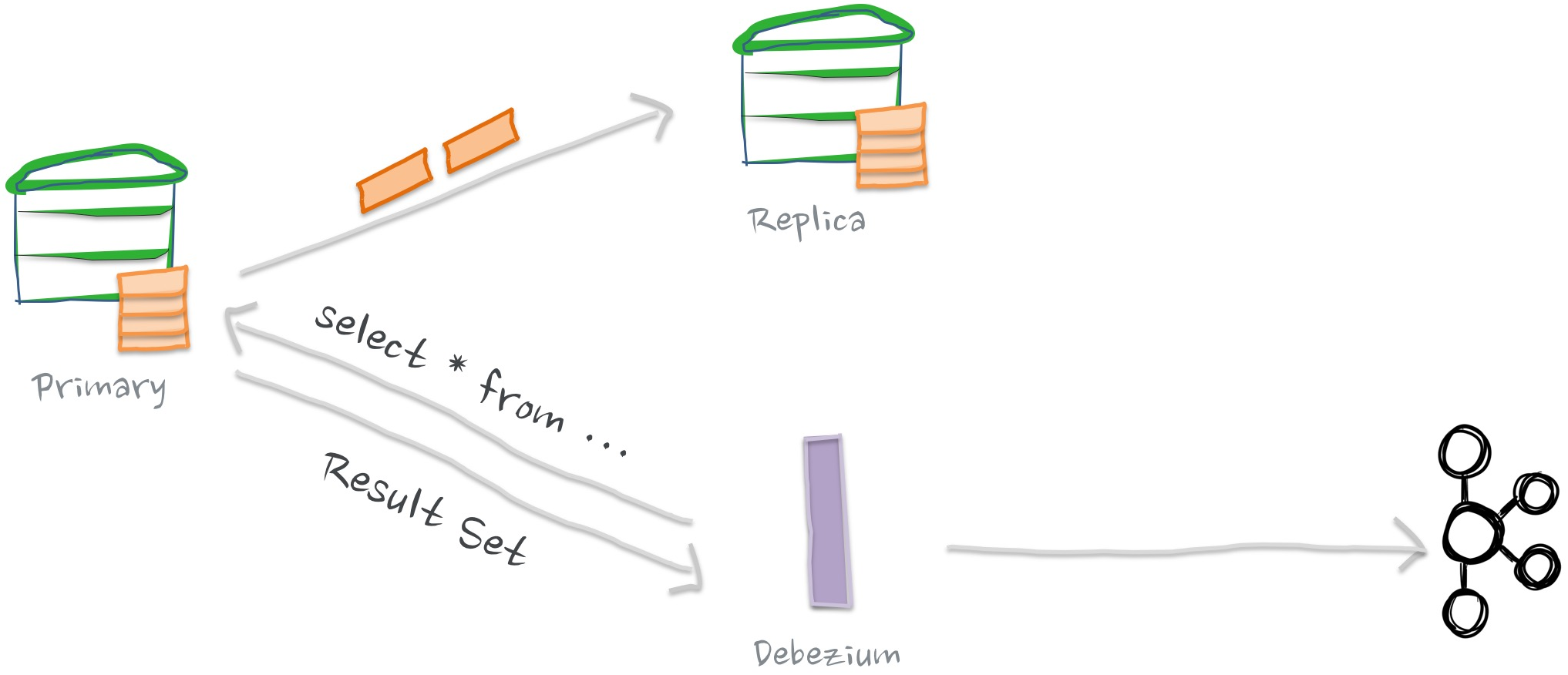




(1) Record latest position

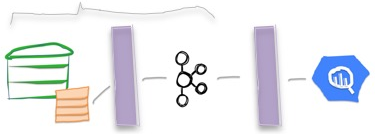
Debezium In Action

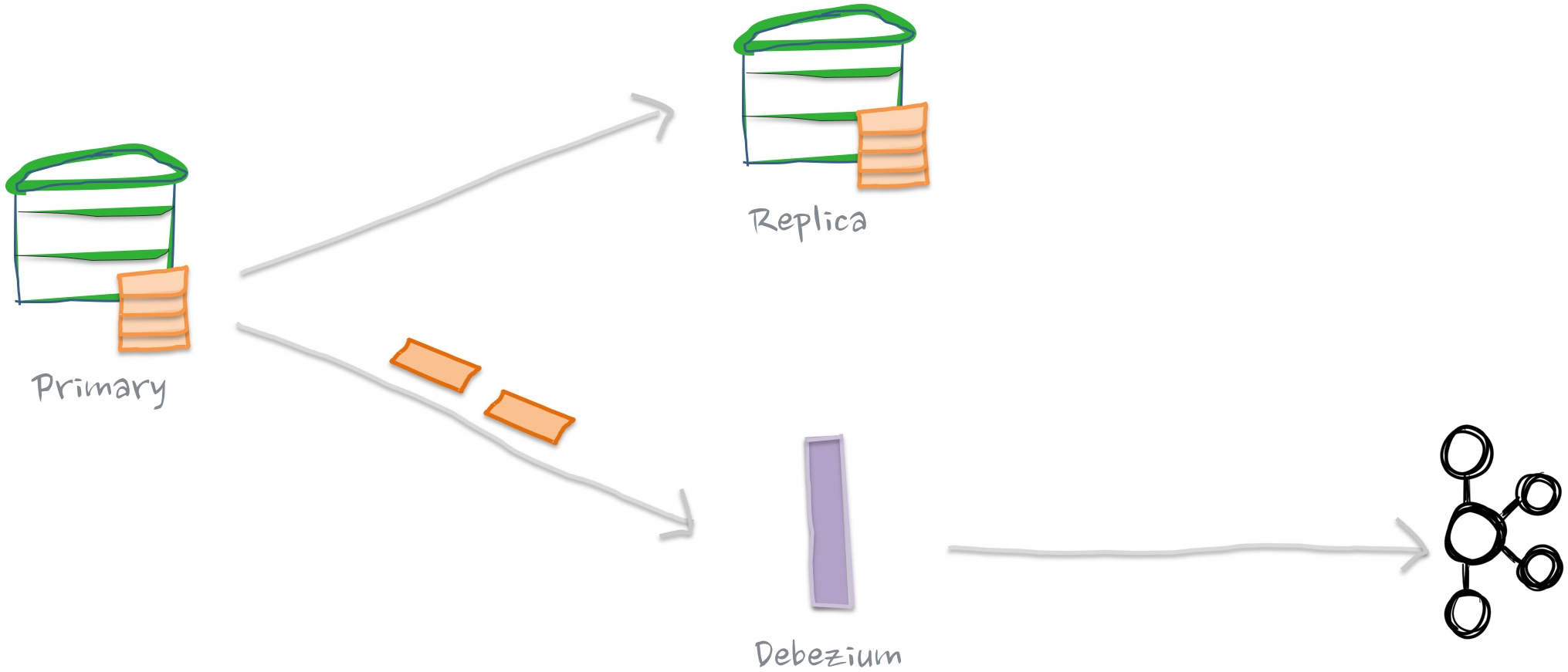




(2) Take a consistent snapshot

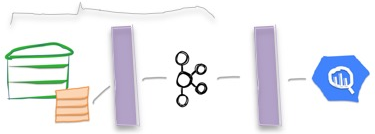
Debezium In Action

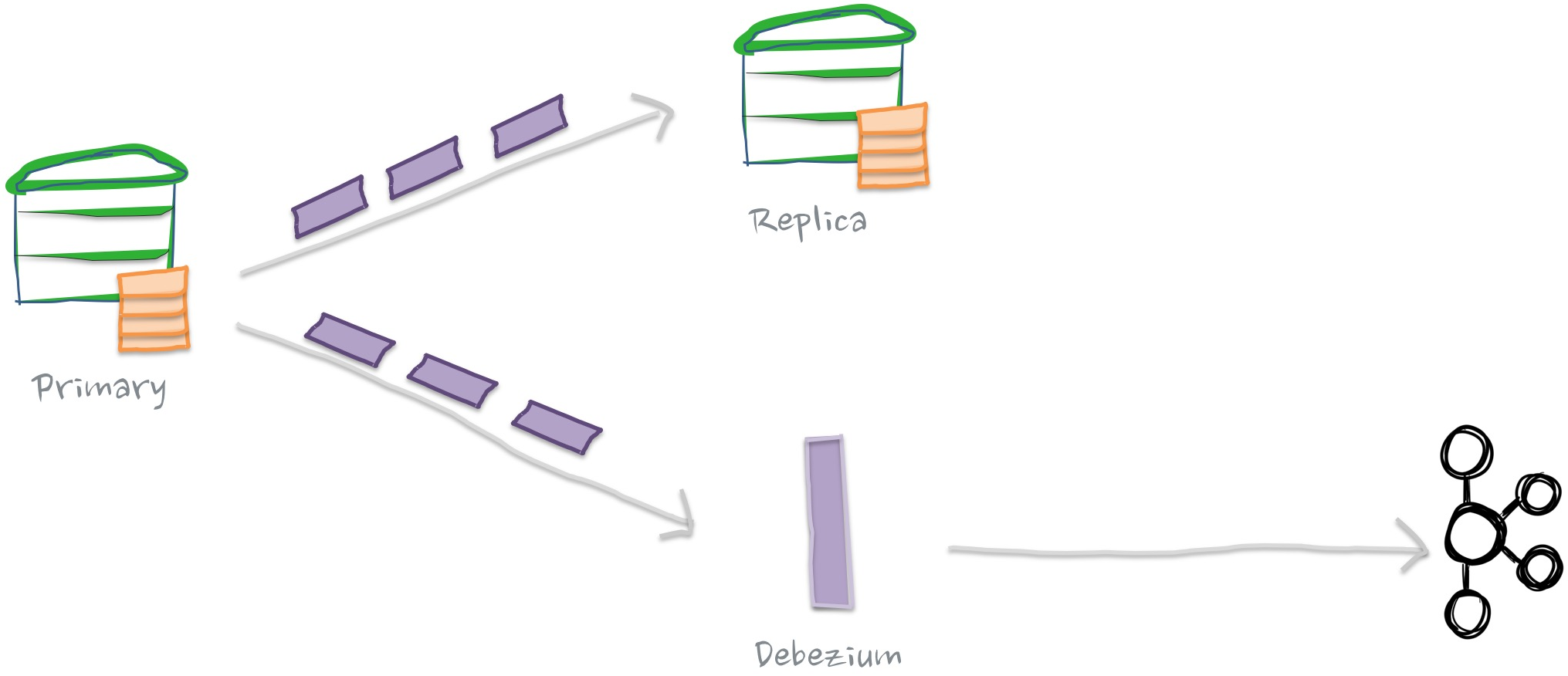




(3) Catch up

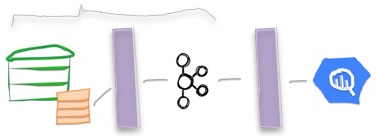
Debezium In Action





(3) Stream changes in real-time

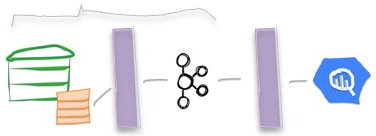
Debezium Event (Update)



```
UPDATE customers SET fname = “Anne Marie” where id = 1004;
```

```
{
  "before": {
      "id": 1004,
      ”fname": "Anne",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "after": {
      "id": 1004,
      ”fname": "Anne Marie",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  …
  …
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      "gtid": null,
      “file": "mysql-bin.000003",
      “pos": 484,
      "row": 0,
      “snapshot": false,
      “db”: "inventory",
      "table":"customers"
  },
  "op": "u",
  "ts_ms": 1465581029523
}
```

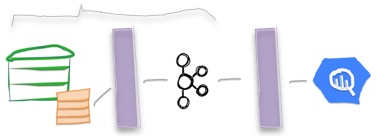
Debezium Event (Update)



```
UPDATE customers SET fname = “Anne Marie” where id = 1004;
```

```
{
  "before": {
      "id": 1004,
      ”fname": "Anne",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "after": {
      "id": 1004,
      ”fname": "Anne Marie",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  …
  …
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      "gtid": null,
      “file": "mysql-bin.000003",
      “pos": 484,
      "row": 0,
      “snapshot": false,
      “db”: "inventory",
      "table":"customers"
  },
  "op": "u",
  "ts_ms": 1465581029523
}
```

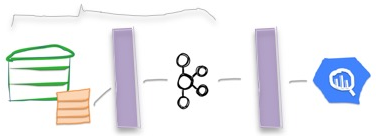
Debezium Event (Update)



```
UPDATE customers SET fname = “Anne Marie” where id = 1004;
```

```
{
  "before": {
      "id": 1004,
      ”fname": "Anne",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "after": {
      "id": 1004,
      ”fname": "Anne Marie",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  …
  …
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      “gtid": null,
      “file": "mysql-bin.000003",
      “pos": 484,
      "row": 0,
      “snapshot": false,
      “db”: "inventory",
      "table":"customers"
  },
  "op": "u",
  "ts_ms": 1465581029523
}
```

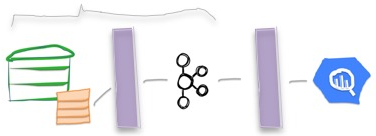
Debezium Event (Update)



```
UPDATE customers SET fname = "Anne Marie" where id = 1004;
```

```
{
  "before": {
      "id": 1004,
      "fname": "Anne",
      "lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "after": {
      "id": 1004,
      "fname": "Anne Marie",
      "lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  …
  …
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      "gtid": null,
      "file": "mysql-bin.000003",
      "pos": 484,
      "row": 0,
      "snapshot": false,
      "db": "inventory",
      "table":"customers"
  },
  "op": "u",
  "ts_ms": 1465581029523
}
```

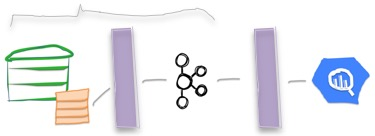
Debezium Event (Update)



```
UPDATE customers SET fname = "Anne Marie" where id = 1004;
```

```
{
  "before": {
      "id": 1004,
      "fname": "Anne",
      "lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "after": {
      "id": 1004,
      "fname": "Anne Marie",
      "lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  …
  …
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      "gtid": null,
      "file": "mysql-bin.000003",
      "pos": 484,
      "row": 0,
      "snapshot": false,
      "db": "inventory",
      "table":"customers"
  },
  "op": "u",
  "ts_ms": 1465581029523
}
```

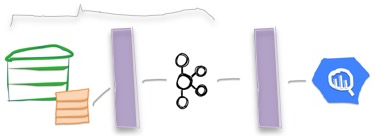
Debezium Event (Create)



```
INSERT INTO customers (id, fname, lname, email)
  VALUES ( 1004, “Anne”, “Kretchmar”, “annek@noanswer.org”);
```

```
{
  "before": null,
  "after": {
      "id": 1004,
      ”fname": "Anne Marie",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  …
  …
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      "gtid": null,
      “file": "mysql-bin.000003",
      “pos": 484,
      "row": 0,
      “snapshot": false,
      “db”: "inventory",
      "table":"customers"
  },
  "op": ”c",
  "ts_ms": 1465581029523
}
```

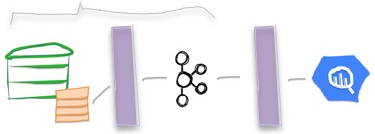
Debezium Event (Delete)

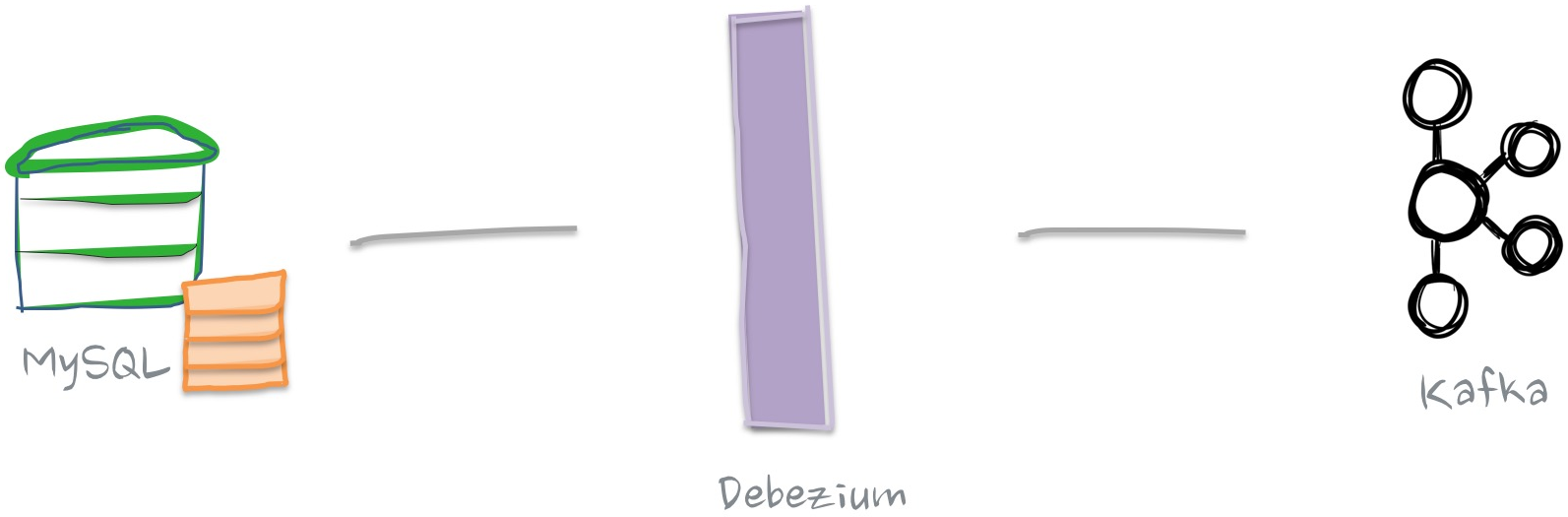


```
DELETE FROM customers WHERE id = 1004;
```

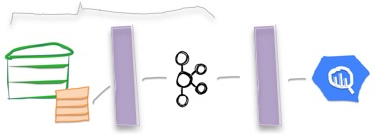
```
{
  "before": {
      "id": 1004,
      ”fname": "Anne",
      ”lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "after": null,
  …
  …
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      "gtid": null,
      “file": "mysql-bin.000003",
      “pos": 484,
      "row": 0,
      “snapshot": false,
      “db”: "inventory",
      "table":"customers"
  },
  "op": ”d",
  "ts_ms": 1465581029523
}
```

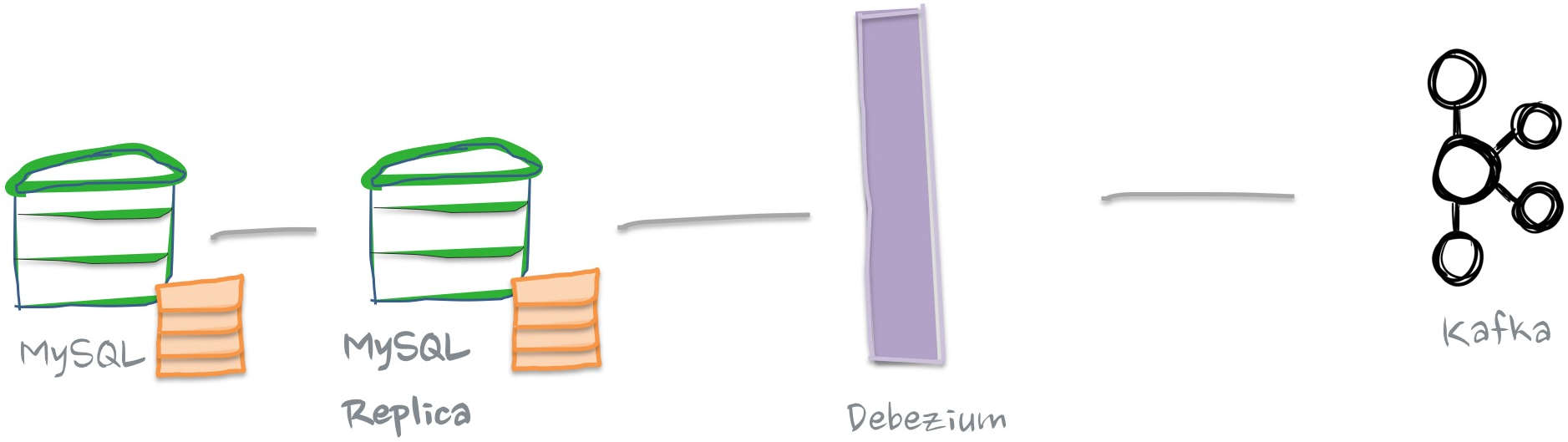
Resilient Debezium Pipeline



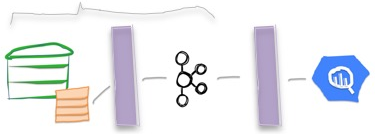


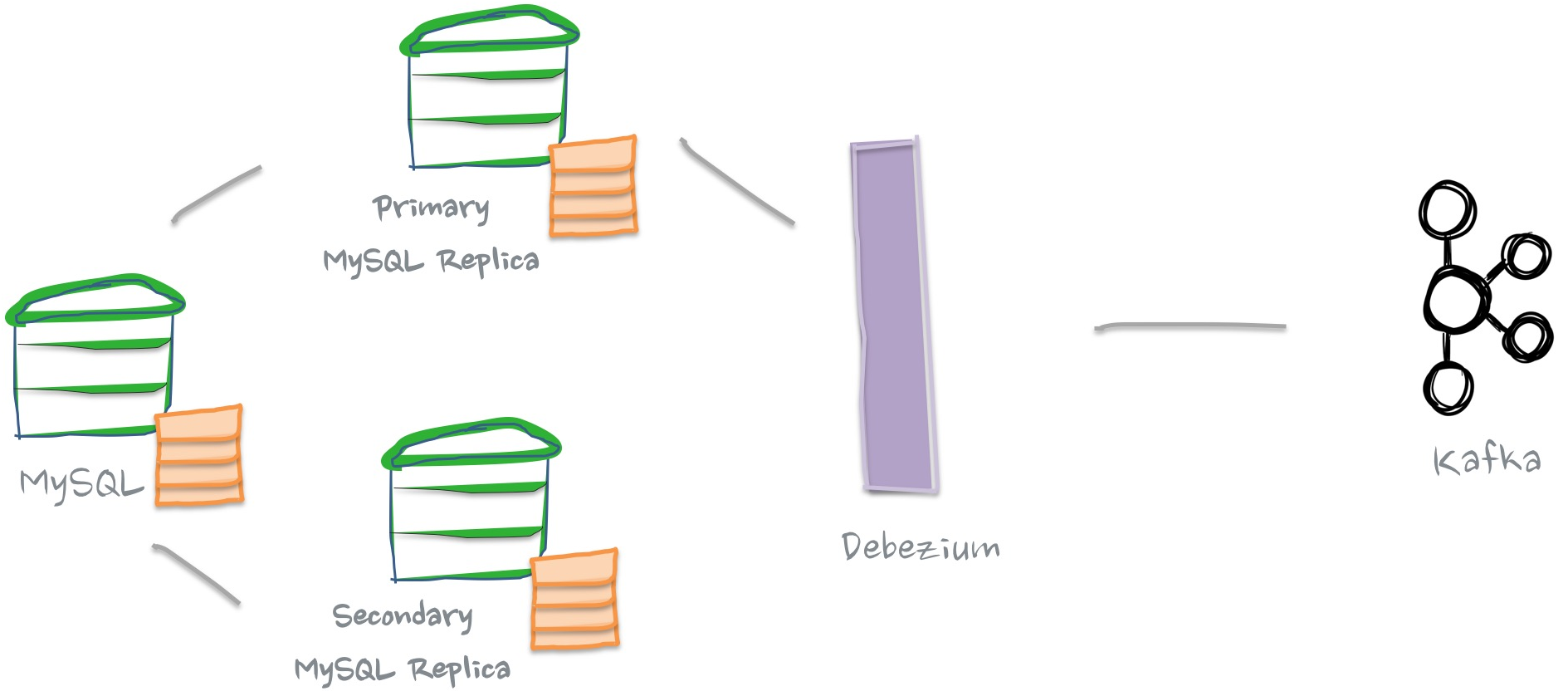
Resilient Debezium Pipeline



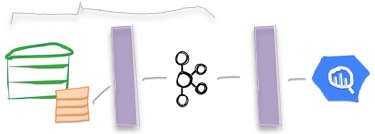


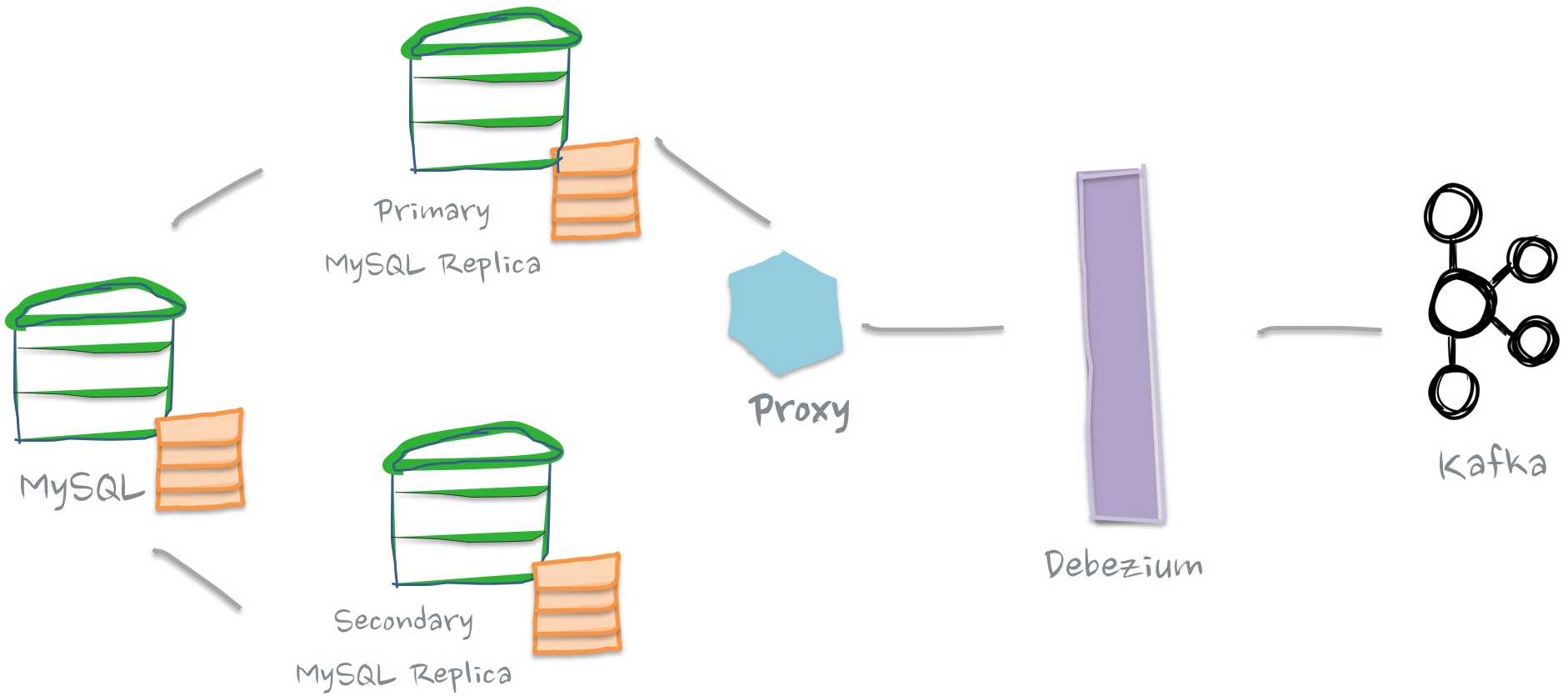
Resilient Debezium Pipeline



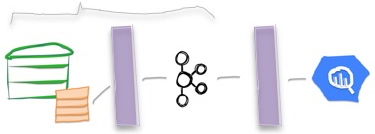


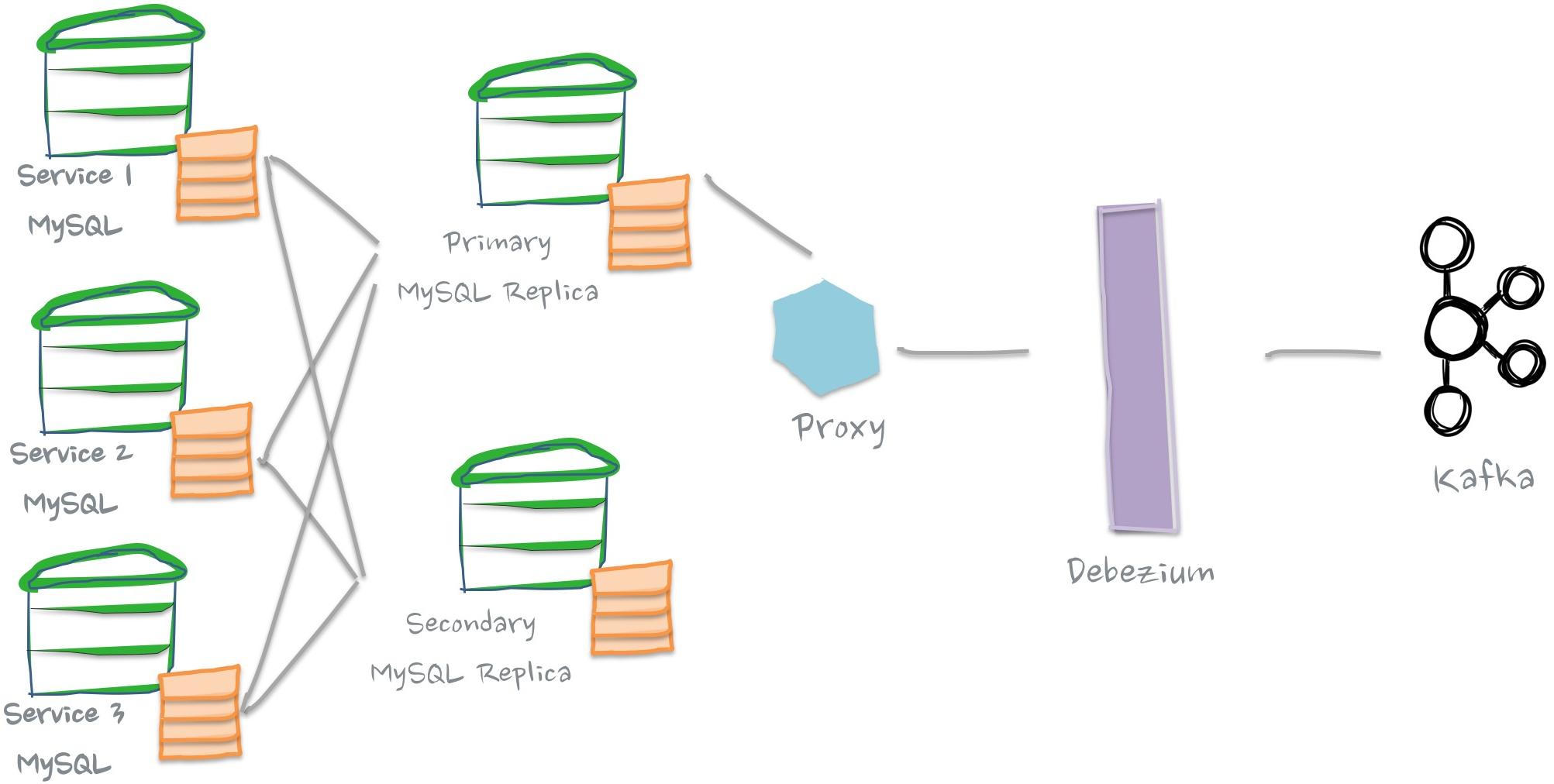
Resilient Debezium Pipeline



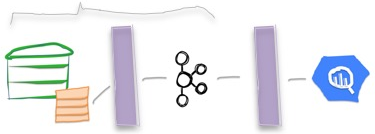


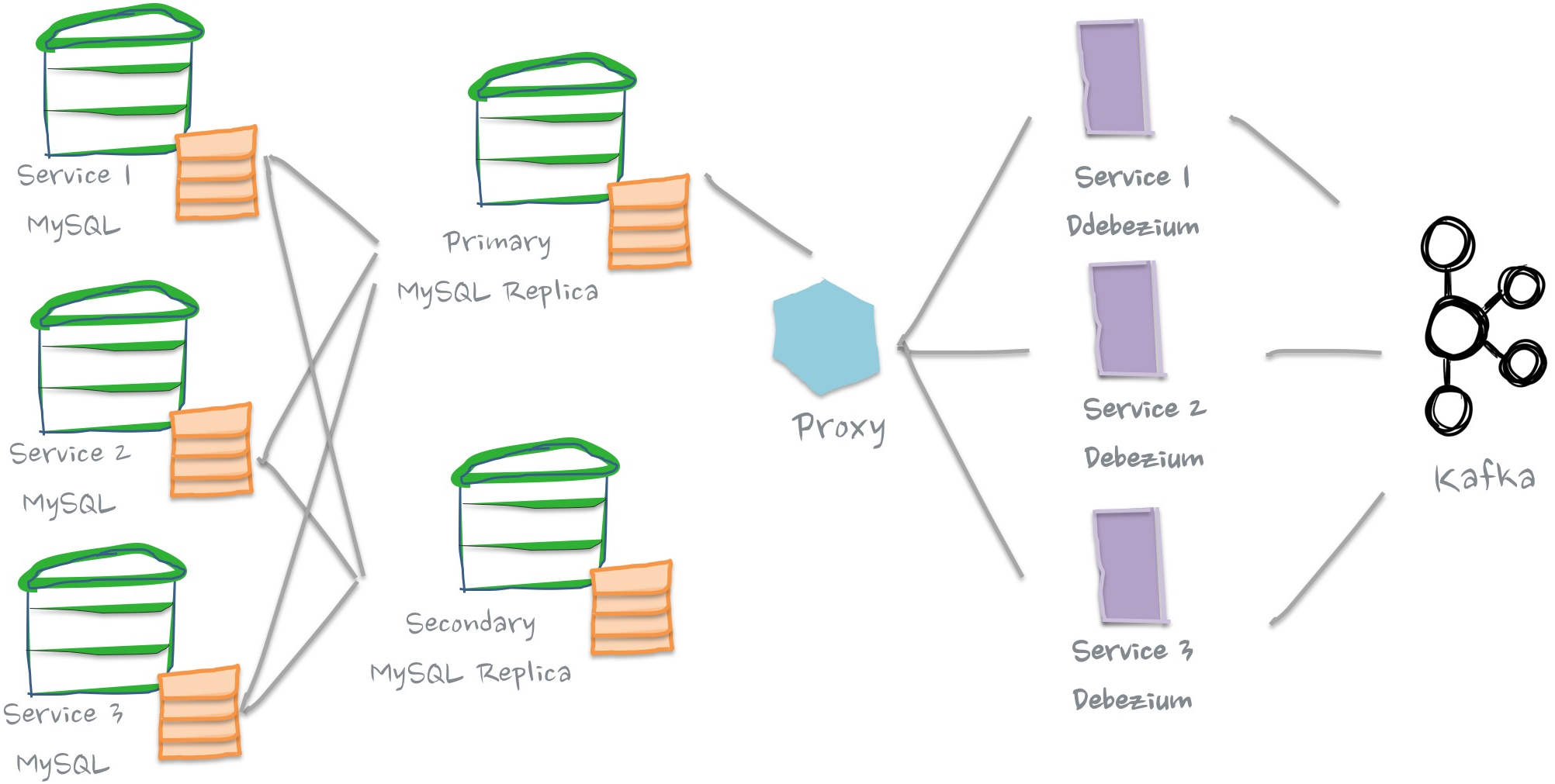
Resilient Debezium Pipeline



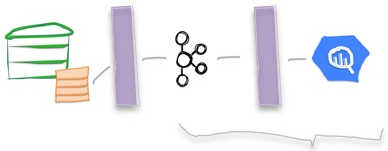


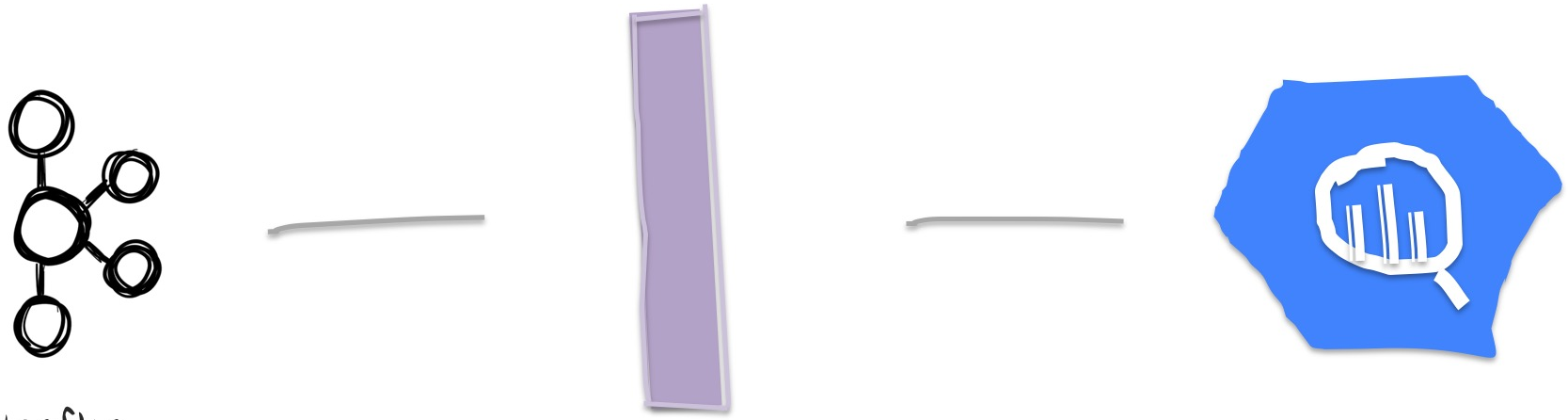
Resilient Debezium Pipeline



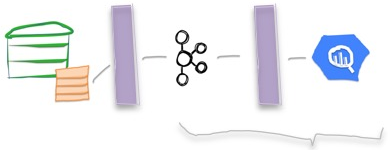


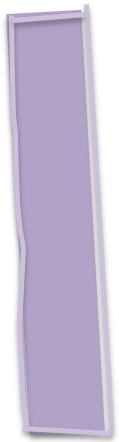
Kafka -> BigQuery





KCBQ Overview

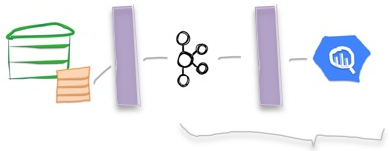




KCBQ

- Open-sourced
- Configurable retry logic
- Lazily update BigQuery schema
- Batch & streaming inserts

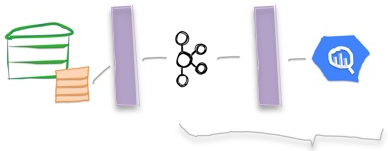
KCBQ Event



UPDATE customers SET fname = "Anne Marie" where id = 1004;

```
{
  "before": {
      "id": 1004,
      "fname": "Anne",
      "lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "after": {
      "id": 1004,
      "fname": "Anne Marie",
      "lname": "Kretchmar",
      "email": "annek@wepay.com"
  },
  "kafka": {
     "offset" : 12345
  },
  ...
  ...
  "source": {
      "name": "mysql-server-1",
      "server_id": 223344,
      "ts_sec": 1465581,
      "gtid": null,
      "file": "mysql-bin.000003",
      "pos": 484,
      "row": 0,
      "snapshot": false,
      "db": "inventory",
      "table":"customers"
  },
  "op": "u",
  "ts_ms": 1465581029523
}
```

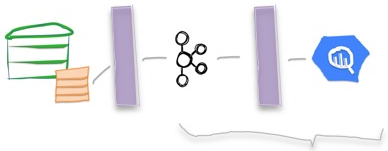
BigQuery View Deduplication & Compression



```
$ SELECT after.id, after.fname, after.lname, after.email, kafka.offset
  FROM customers WHERE after.id = 1004;
```

after.id	after.fname	after.lname	after.email	kafka.offset
1004	Anne	Kretchmar	annek@wepay.com	12300
1004	Anne Marie	Kretchmar	annek@wepay.com	12345

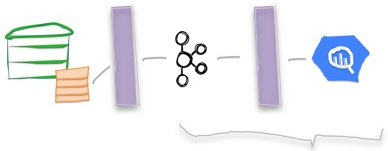
BigQuery View Deduplication & Compression



```
$ SELECT after.id, after.fname, after.lname, after.email, kafka.offset
  FROM customers WHERE after.id = 1004;
```

after.id	after.fname	after.lname	after.email	kafka.offset
1004	Anne	Kretchmar	annek@wepay.com	12300
1004	Anne Marie	Kretchmar	annek@wepay.com	12345

BigQuery View Deduplication & Compression



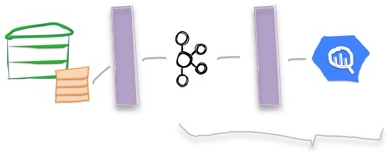
```
$ SELECT after.id, after.fname, after.lname, after.email, kafka.offset
  FROM customers WHERE after.id = 1004;
```

after.id	after.fname	after.lname	after.email	kafka.offset
1004	Anne	Kretchmar	annek@wepay.com	12300
1004	Anne Marie	Kretchmar	annek@wepay.com	12345

```
$ SELECT * FROM customers__full_view WHERE id = 1004;
```

id	fname	lname	email
1004	Anne Marie	Kretchmar	annek@wepay.com

BigQuery View Masking



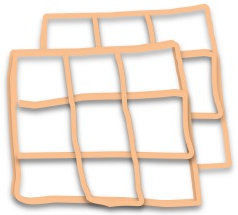
```
$ SELECT after.id, after.fname, after.lname, after.email, kafka.offset
  FROM customers WHERE after.id = 1004;
```

after.id	after.fname	after.lname	after.email	kafka.offset
1004	Anne	Kretchmar	annek@wepay.com	12300
1004	Anne Marie	Kretchmar	annek@wepay.com	12345

```
$ SELECT * FROM customers__clean_view WHERE id = 1004;
```

id	fname	lname
1004	Anne Marie	Kretchmar

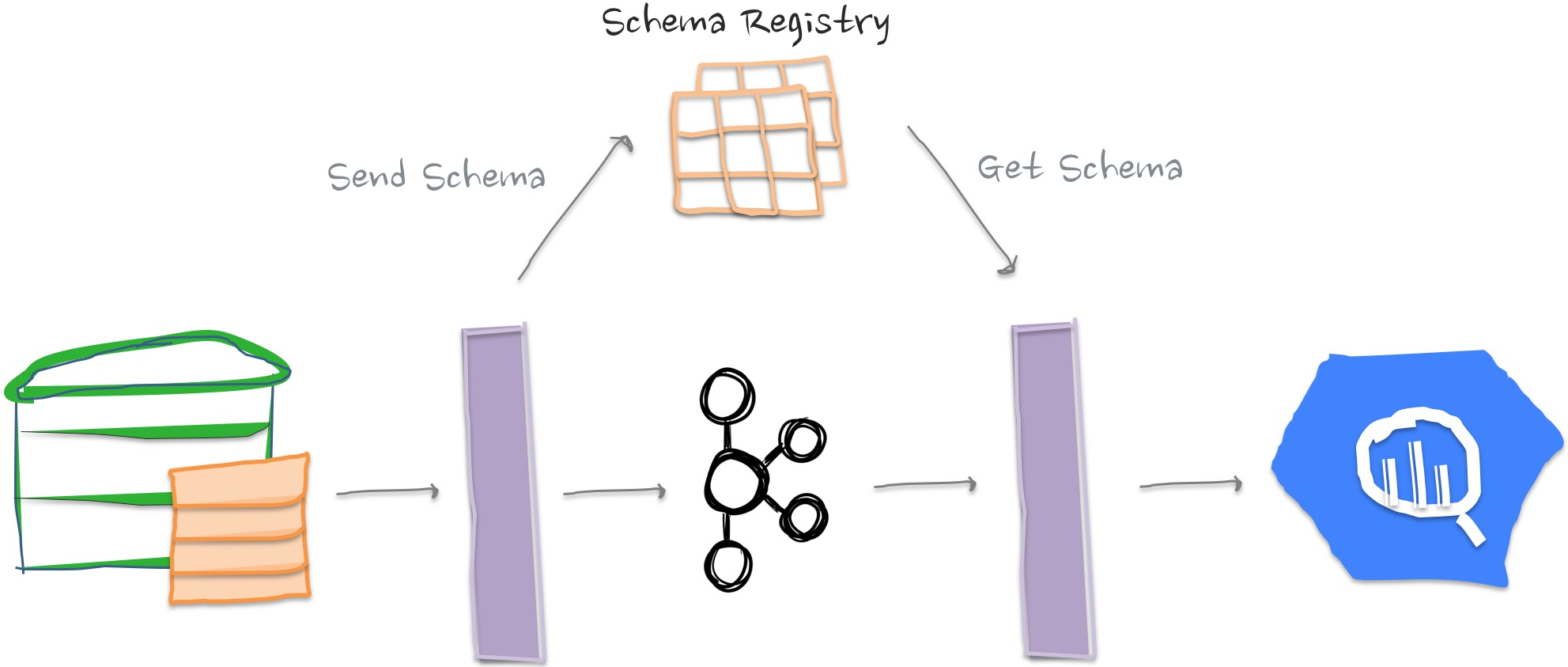
Schema Registry Overview



Schema Registry

- Registry of data schemas
- Kafka as the underlying storage layer
- Apache Avro as data serialization format
- Distributed, single-master architecture

Schema Evolution

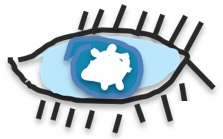


Agenda

1. The Beauty of Change Data Capture

2. Real-World Example: Streaming MySQL

3. Future Challenge: Streaming Cassandra



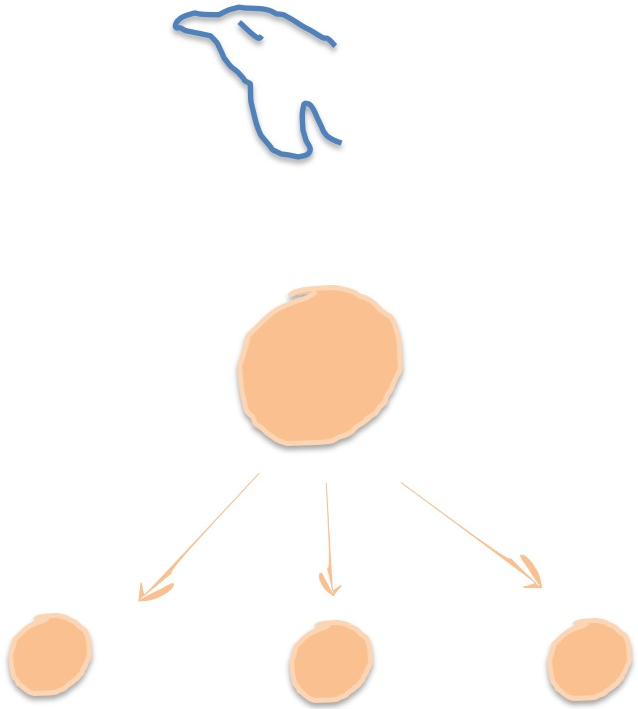
Cassandra Overview

High write throughput

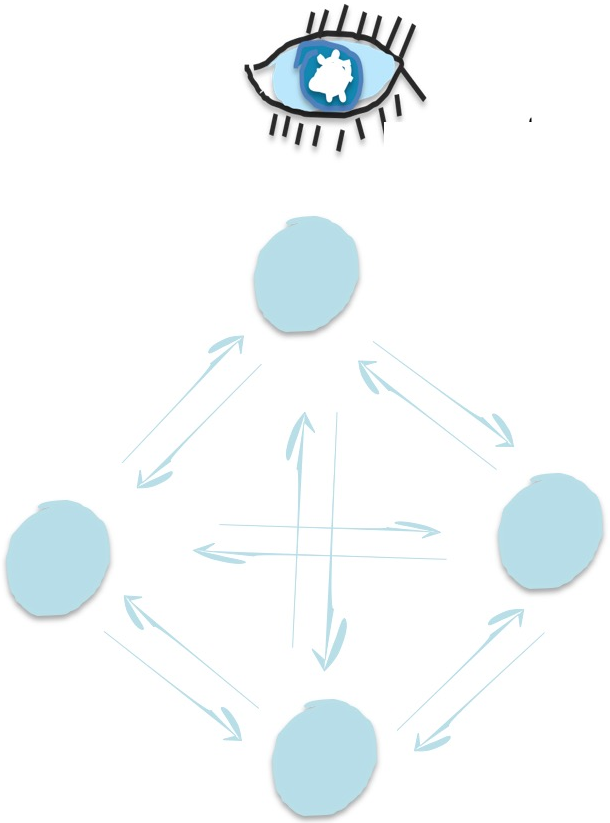
Horizontally scalable

Highly available

Peer-to-Peer Replication Model

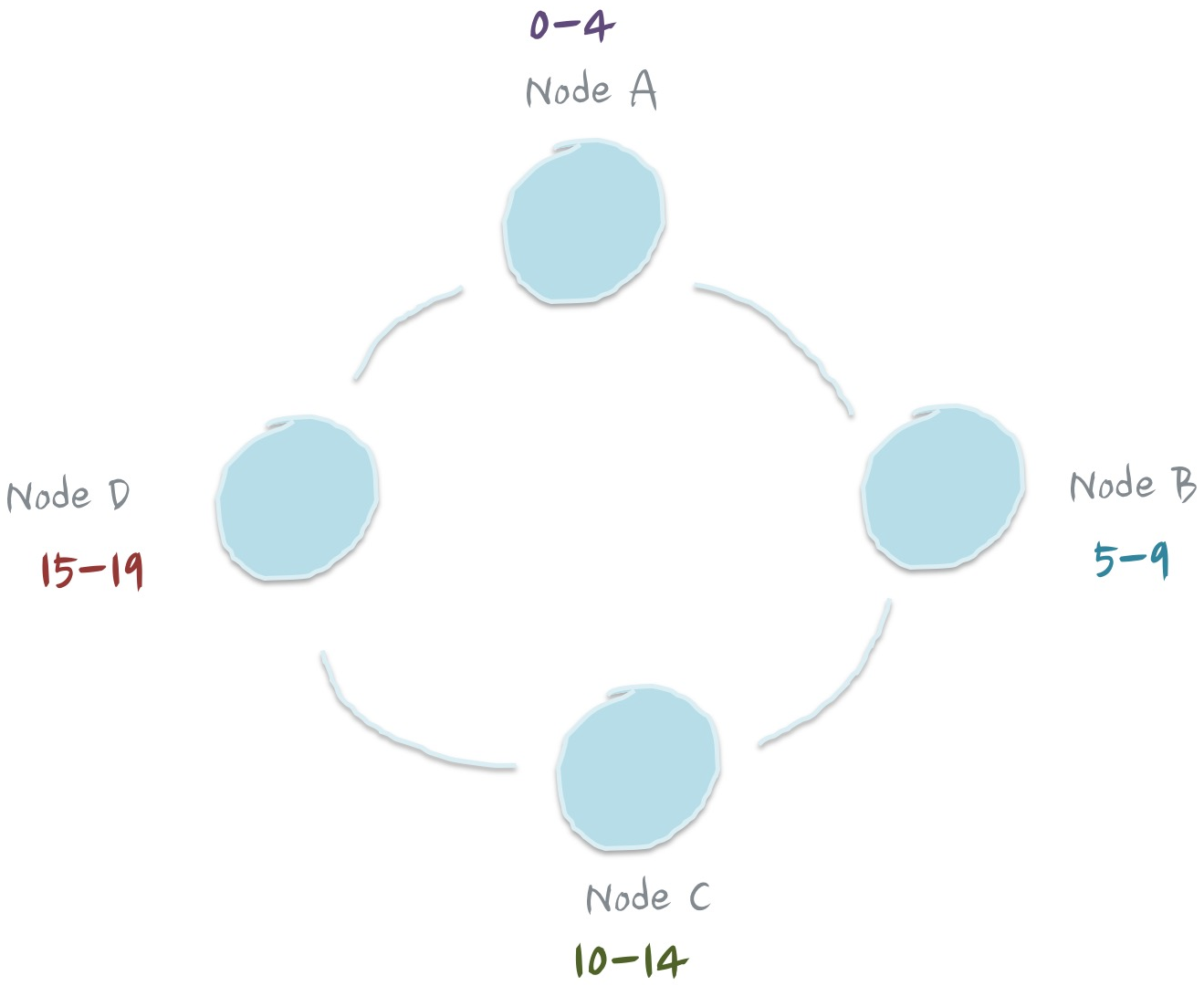


Primary-Replica

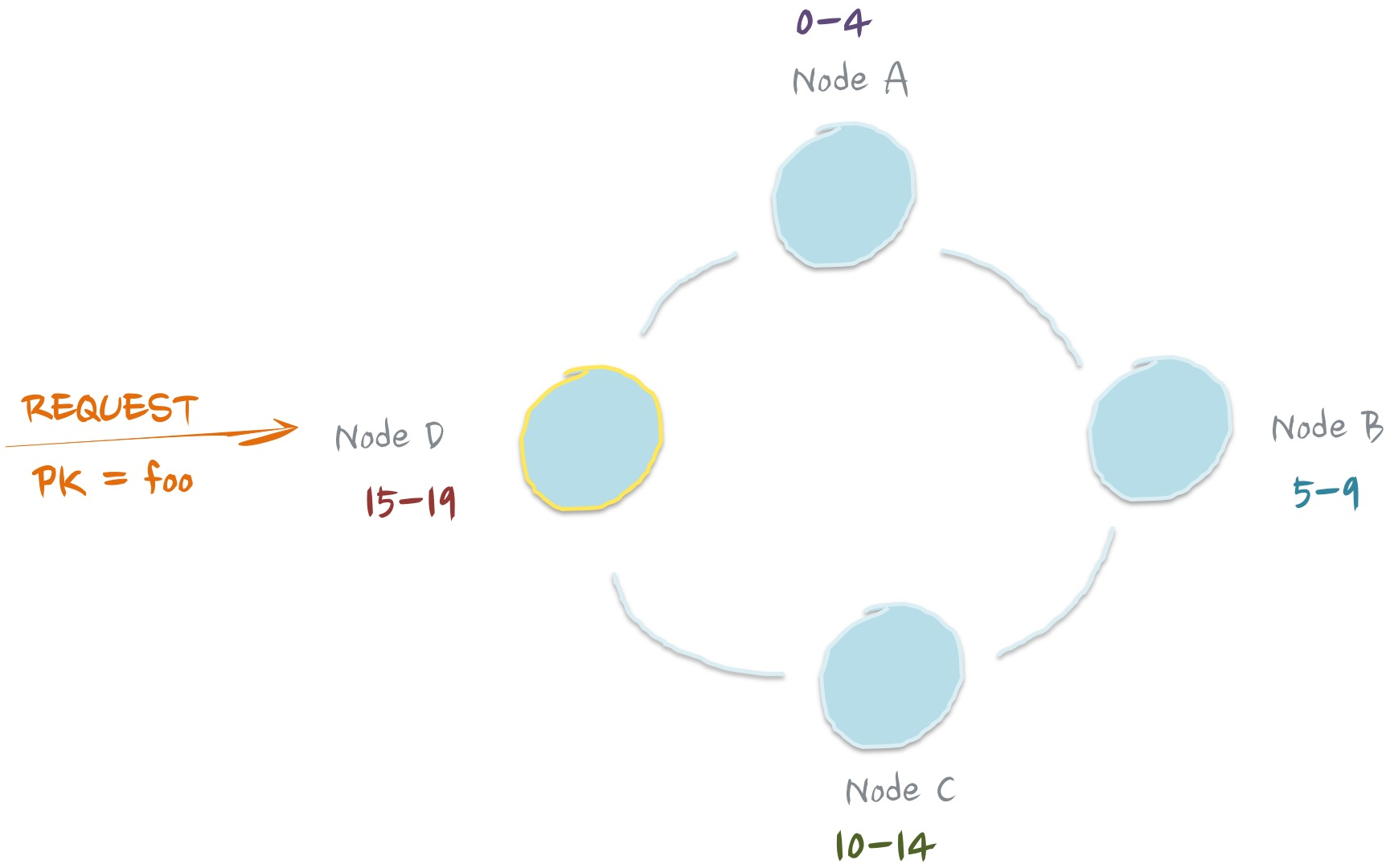


Peer-To-Peer

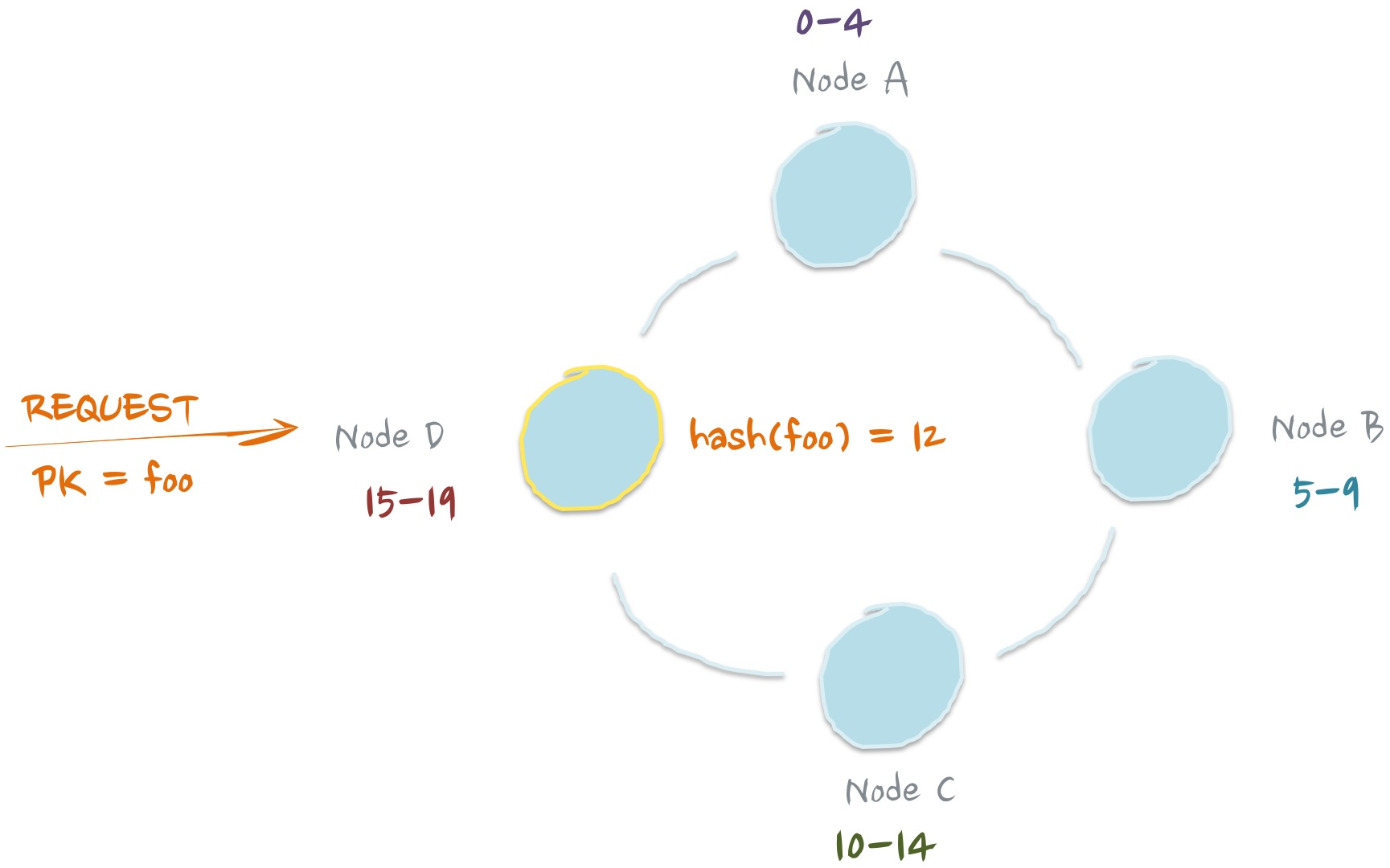
Token Ring (RF = 1)



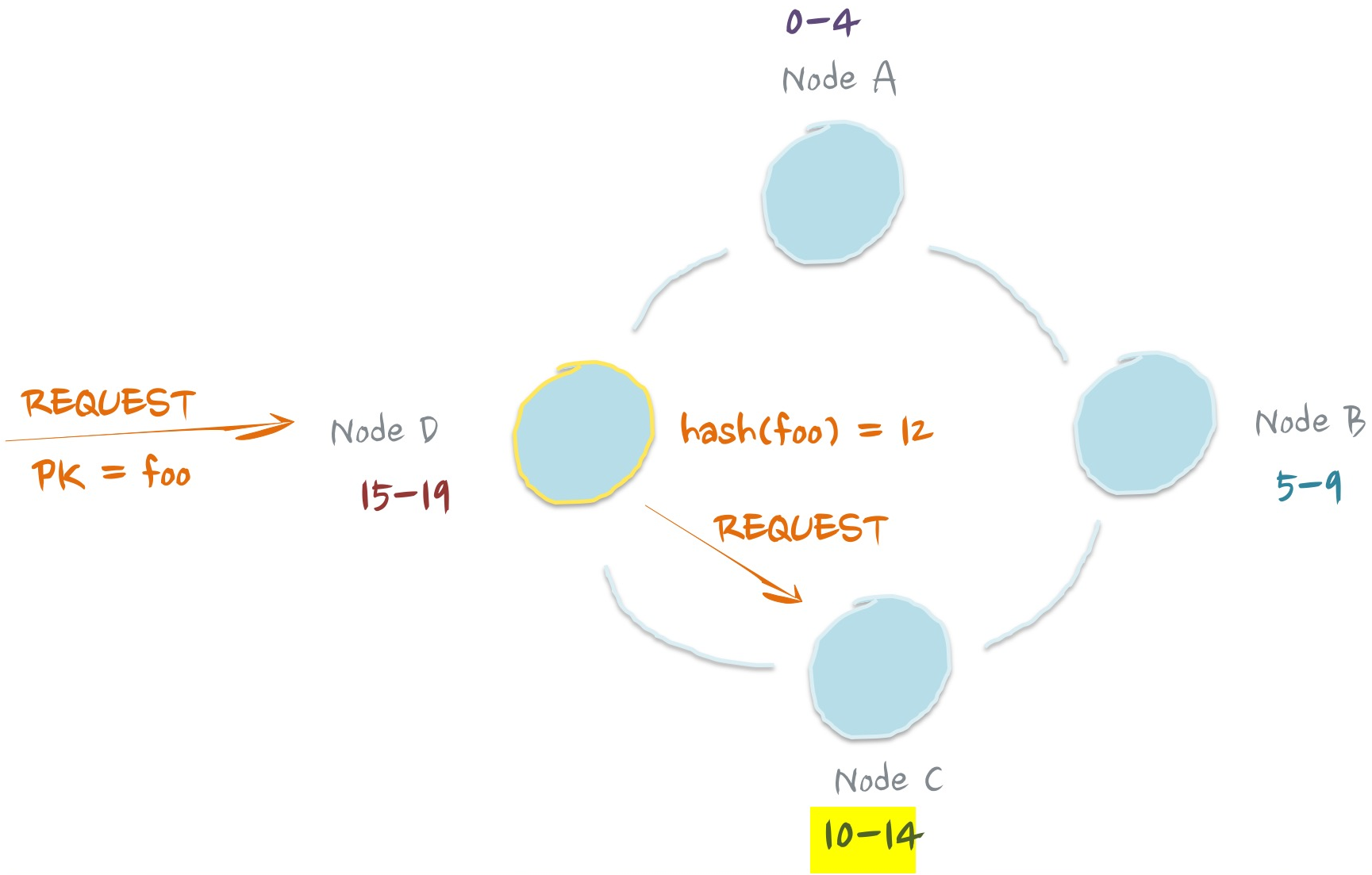
Token Ring (RF = 1)



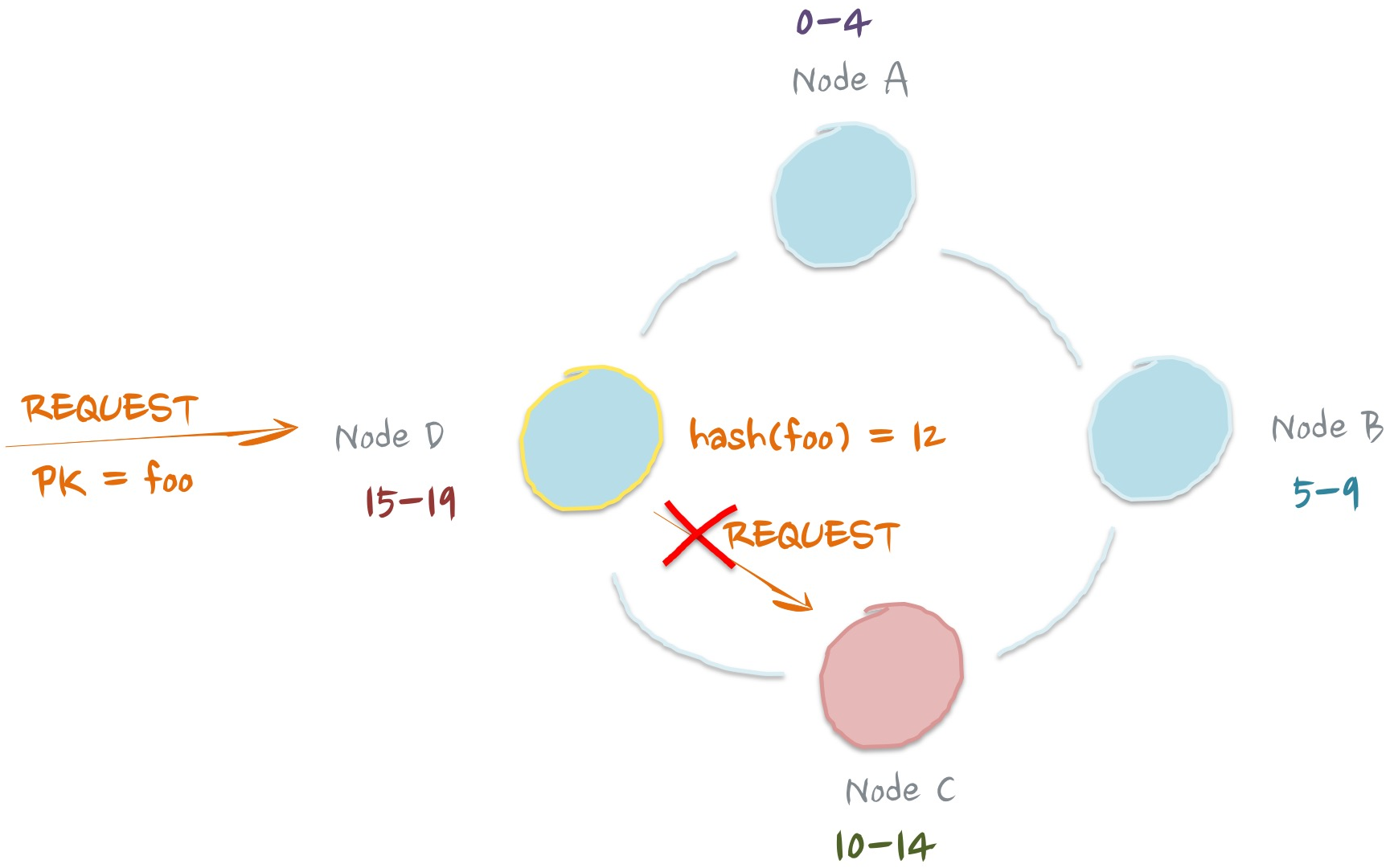
Token Ring (RF = 1)



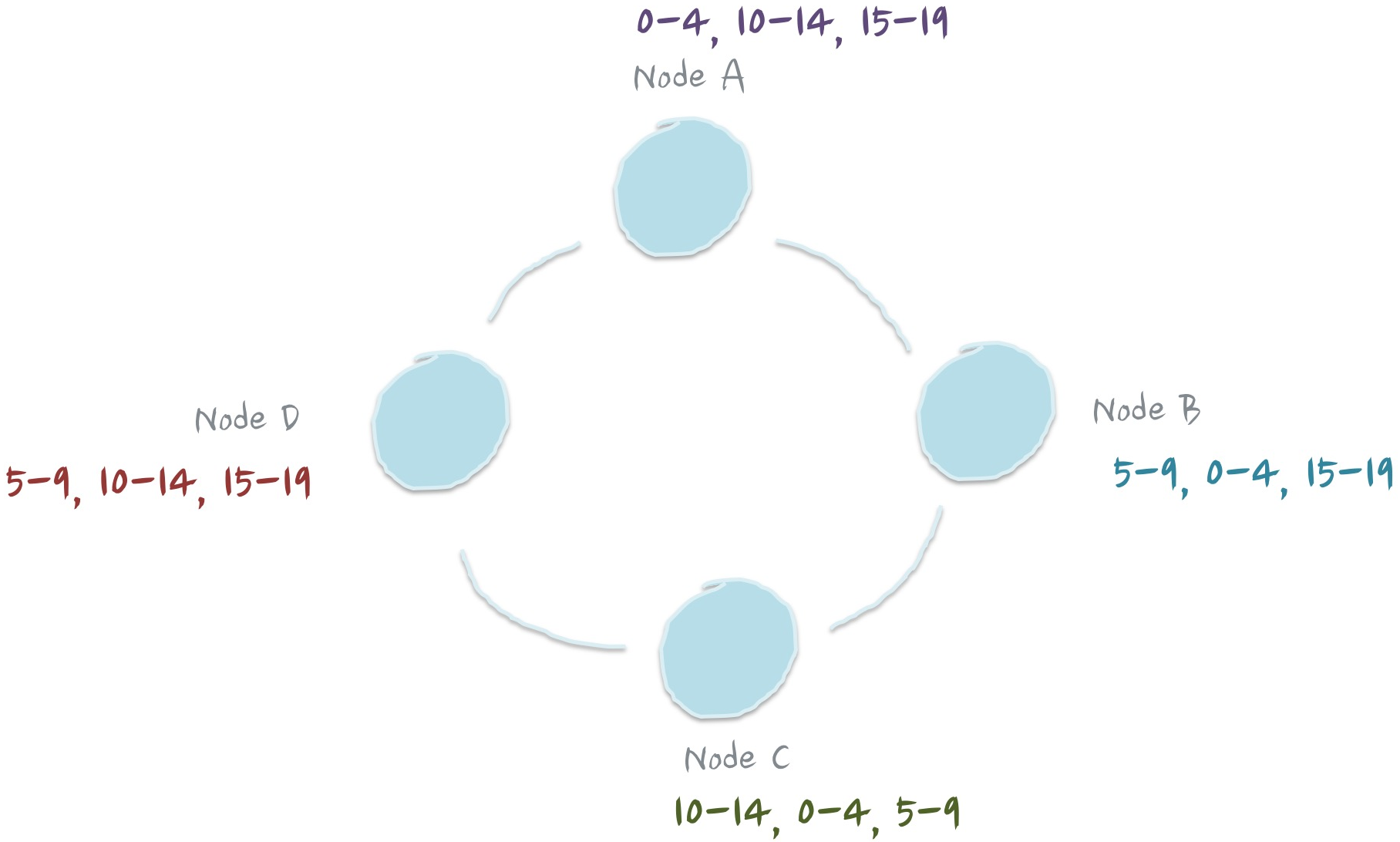
Token Ring (RF = 1)



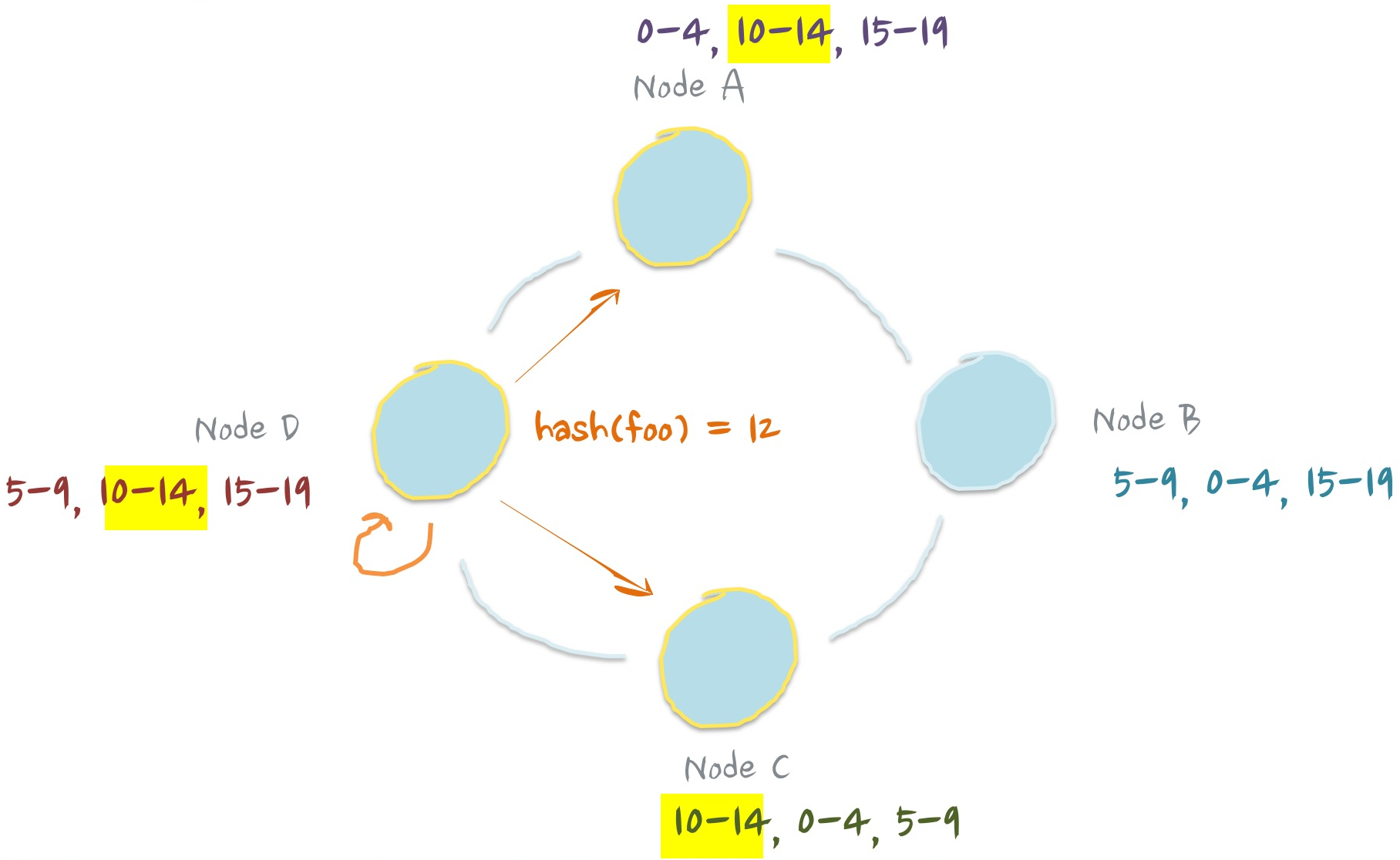
Token Ring (RF = 1)



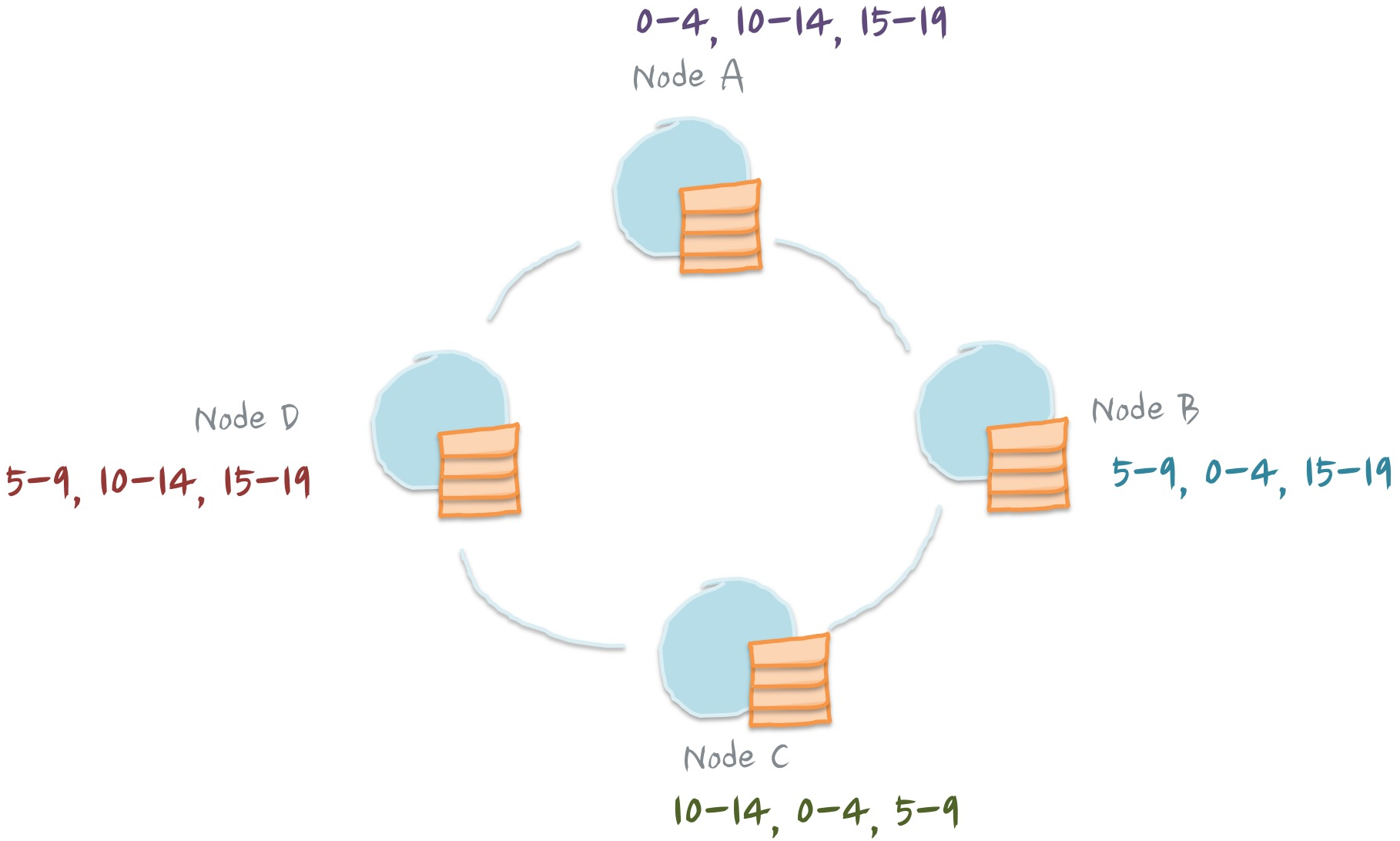
Token Ring (RF = 3)



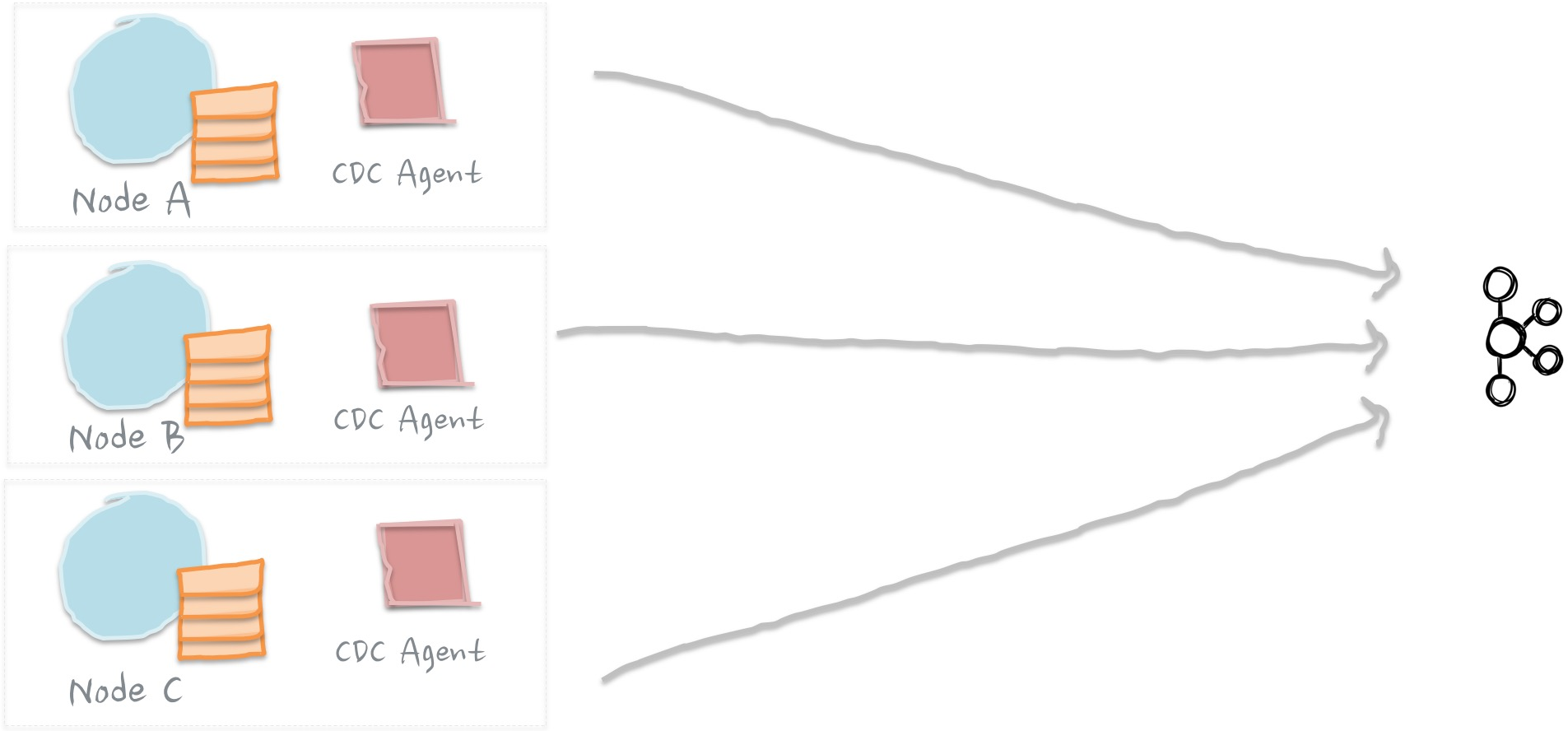
Token Ring (RF = 3)



Token Ring (RF = 3)



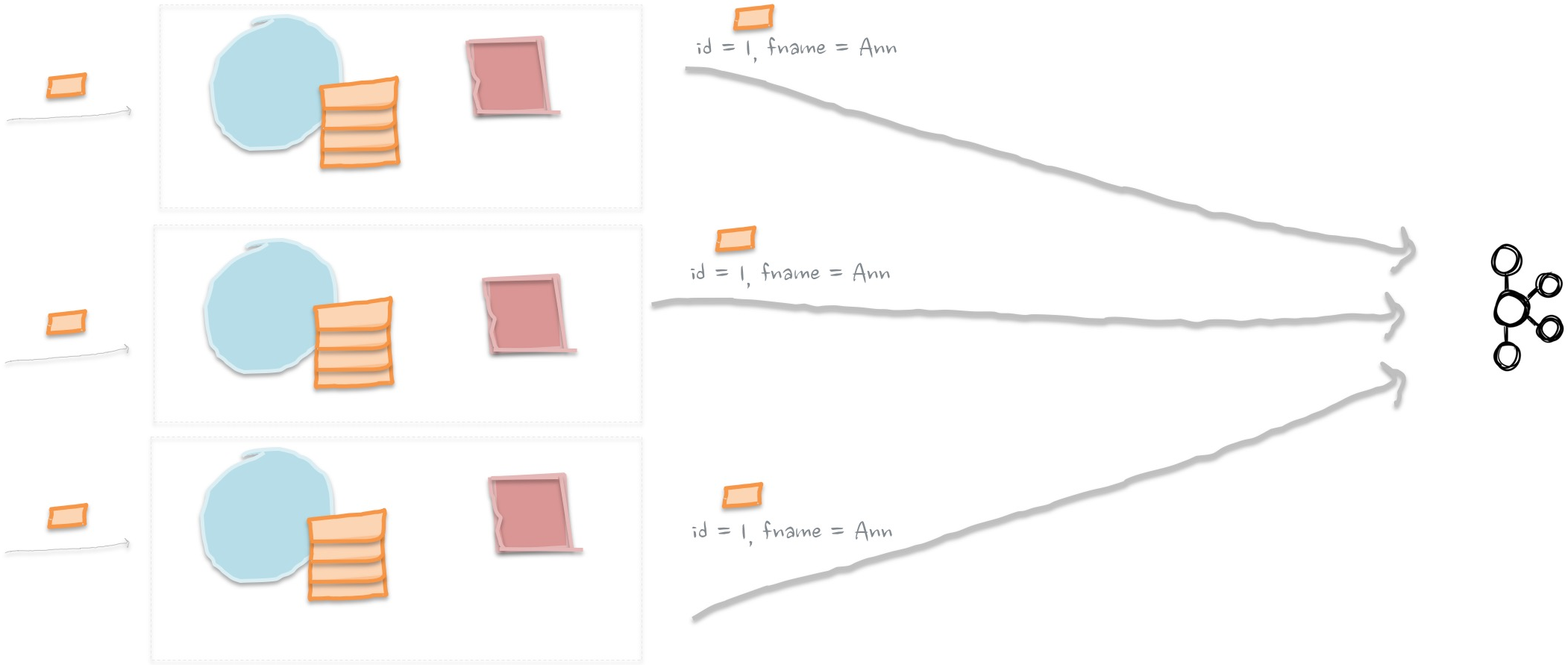
Streaming Cassandra Commit Logs



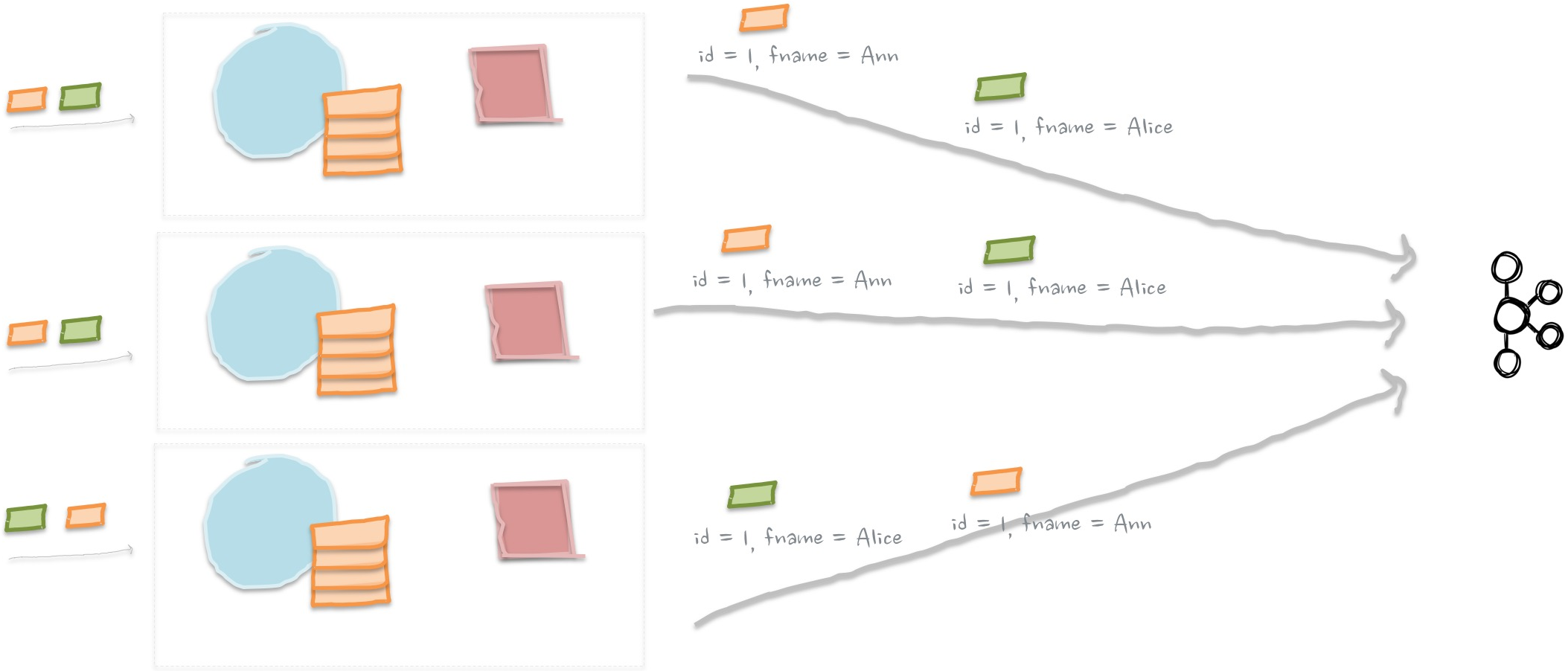
Streaming Cassandra Commit Logs

```
public interface CommitLogReadHandler {

    // Process a deserialized mutation
    void handleMutation(Mutation m,
                        int size,
                        int entryLocation,
                        CommitLogDescriptor desc);

}
```

Challenge 1: Duplicated Change Events

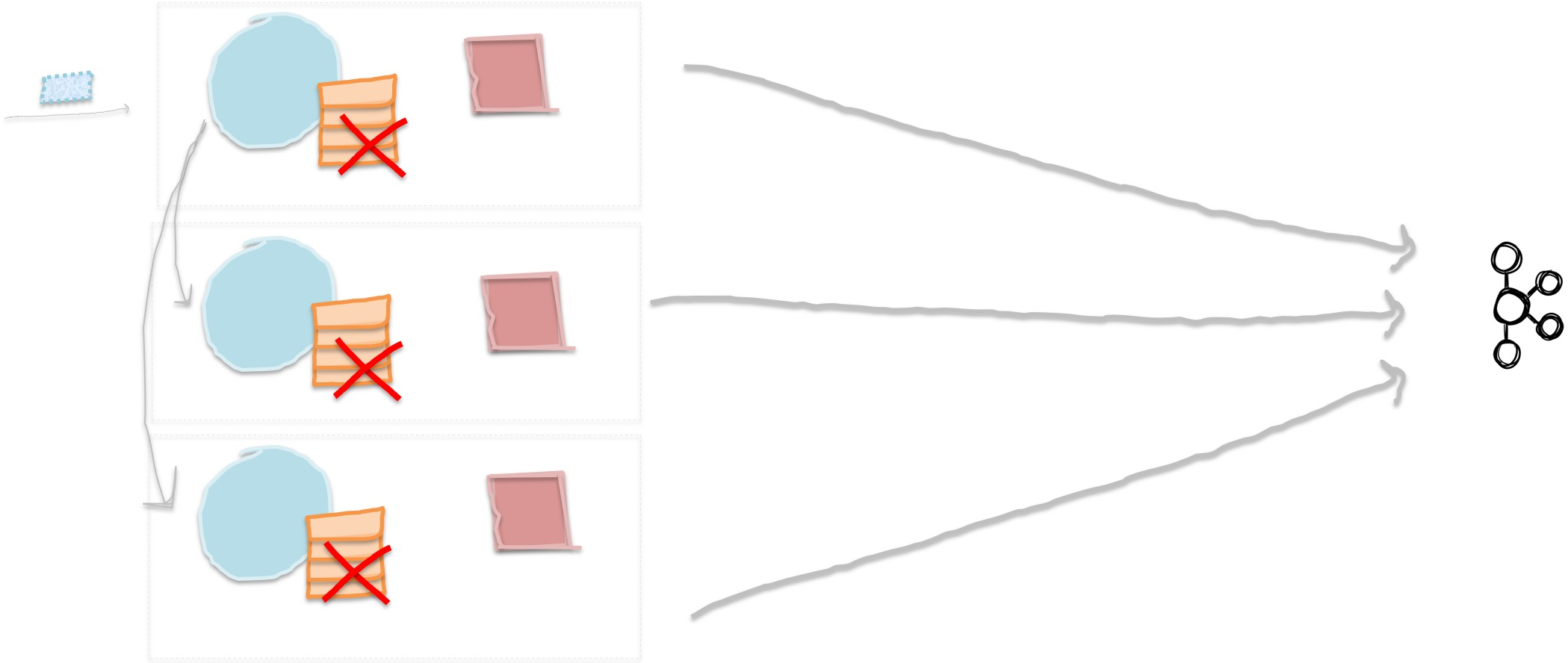


Challenge 2: Out-of-order Change Events

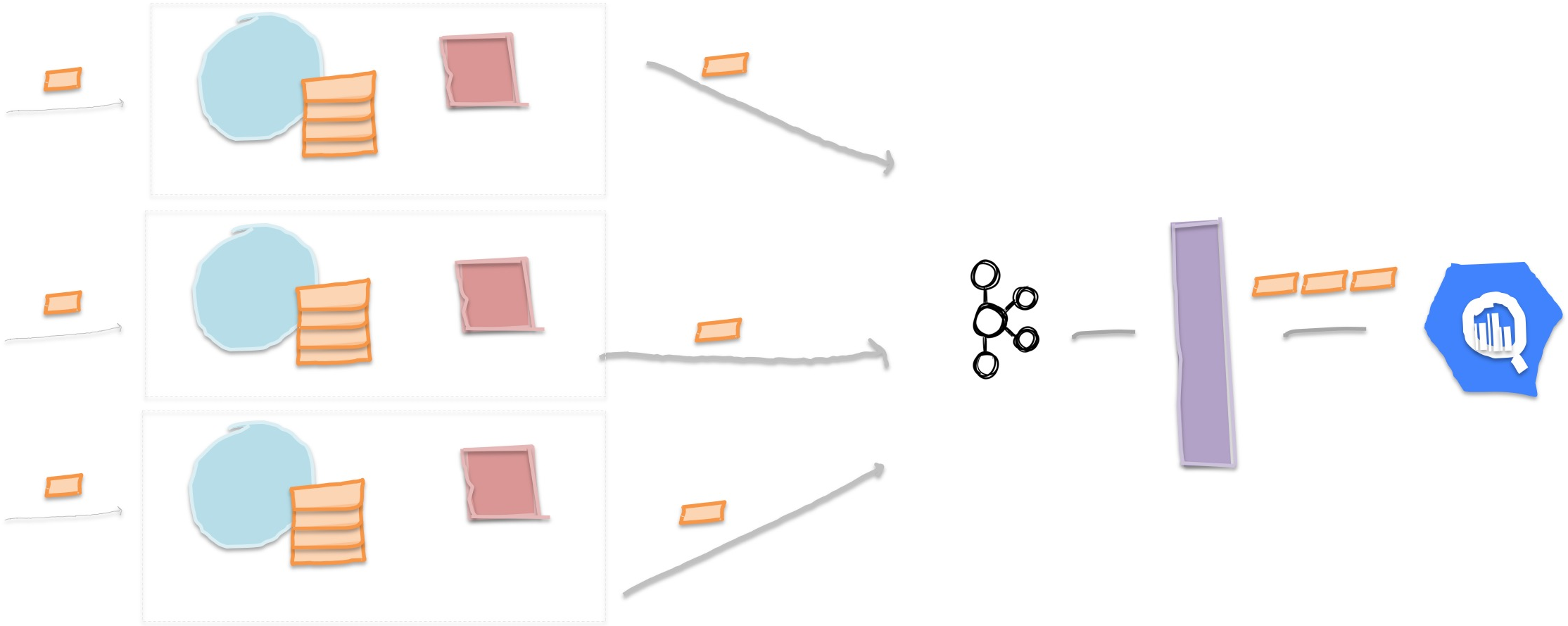


Challenge 3: Incomplete Change Events

Challenge 4: Unlogged Schema Change



Current Bridging-The-Gap Solution



A Much More Comprehensive Change Event

```
...
"after": {
    "id": {
      "value": 1004,
      "timestamp" : 1541230518081,
      "deletion_timestamp": -1,
      "is_primary": true
   },
   "fname": {
      "value": "Anne Marie",
      "timestamp" : 1541230518081,
      "deletion_timestamp": -1,
      "is_primary": false
   },
}

...
```

BigQuery View Deduplication & Compression

```
$ SELECT after.fname.value, after.fname.ts, after.fname.deletion_ts, after.fname.is_primary
  FROM customers WHERE after.id.value = 1004;

 ------------------------------------------------------------------------------------------
 after.fname.value    | after.fname.ts     | after.fname.deletion_ts  | after.fname.is_primary
 ------------------------------------------------------------------------------------------
 Anne                 | 1541230518080      | -1                       | false
 ------------------------------------------------------------------------------------------
 Anne Marie           | 1541230518081      | -1                       | false
 ------------------------------------------------------------------------------------------
```

BigQuery View Deduplication & Compression

```
$ SELECT after.fname.value, after.fname.ts, after.fname.deletion_ts, after.fname.is_primary
  FROM customers WHERE after.id.value = 1004;
```

after.fname.value	after.fname.ts	after.fname.deletion_ts	after.fname.is_primary
Anne	1541230518080	-1	false
Anne Marie	1541230518081	-1	false

```
$ SELECT after.lname.value, after.lname.ts, after.lname.deletion_ts, after.lname.is_primary
  FROM customers WHERE after.id.value = 1004;
```

after.lname.value	after.lname.ts	after.lname.deletion_ts	after.lname.is_primary
Kretchmar	1541230518080	-1	false
null	null	null	null

BigQuery View Deduplication & Compression

```
$ SELECT after.fname.value, after.fname.ts, after.fname.deletion_ts, after.fname.is_primary
  FROM customers WHERE after.id.value = 1004;
```

after.fname.value	after.fname.ts	after.fname.deletion_ts	after.fname.is_primary
Anne	1541230518080	-1	false
Anne Marie	1541230518081	-1	false

```
$ SELECT after.lname.value, after.lname.ts, after.lname.deletion_ts, after.lname.is_primary
  FROM customers WHERE after.id.value = 1004;
```

after.lname.value	after.lname.ts	after.lname.deletion_ts	after.lname.is_primary
Kretchmar	1541230518080	-1	false
null	null	null	null

BigQuery View Deduplication & Compression

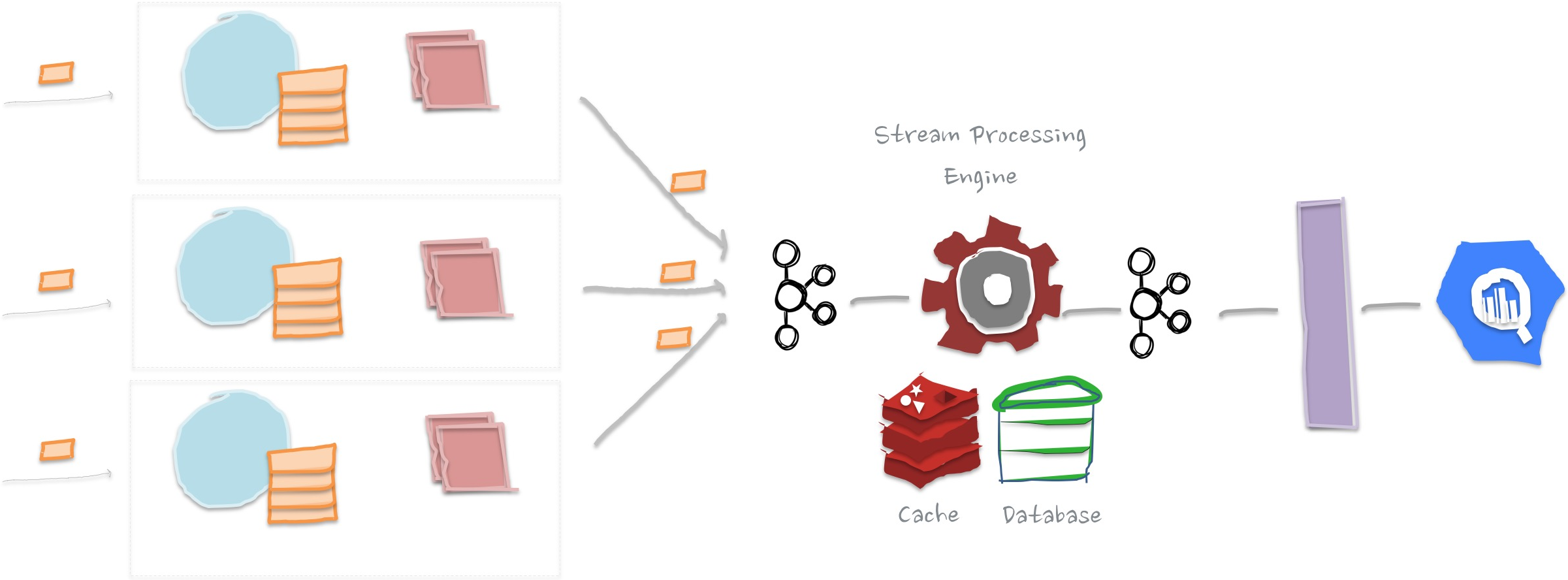
```
$ SELECT * from customers__full_view WHERE after.id.value = 1004;

 ----------------------------------------------------------------------
 id                | fname               | lname              | email
----------------------------------------------------------------------
 1004              | Anne Marie          | Kretchmar          | annek@wepay.com
 ----------------------------------------------------------------------
```

Advantages & Disadvantages

+ Quick iteration
+ Few operational overhead
+ Doesn't impact production

- Expensive computation on every read
- Regular compaction required
- Custom pipeline for BigQuery

Potential Future Solution



Summary

Database as a stream of events (CDC) is a natural & useful concept

Log-centric architecture is at the heart of streaming data pipelines

CDC for peer-to-peer distributed database is not trivial

Additional Info

Streaming databases in real-time with MySQL, Debezium, and Kafka

-> http://bit.ly/streaming-databases-in-real-time

KCBQ Github

-> http://bit.ly/kafka-connect-bigquery

Q&A

 @joygao  /joygao