Transaction Cascades

or how to build a transactional microservice architecture

Harald Wendel

TransferWise

About me

- Engineering Lead @ TransferWise
- 15 years of Java
- Spent the last 10 years building trading and risk management systems

What you’ll learn today is

How to build a transactional microservice architecture that scales

and a little bit about transactions and KAFKA :-)

You should listen to this if

- You’re stuck with this monolith that dies under the load
- You’re interested in building asynchronous systems
- You just want to hear what we are doing with KAFKA
- You don’t like the term ‘Enterprise” :-)

- Quick Recap: Transactions
- What problem are we trying to solve?
- Quick Recap: KAFKA
- Solution
- Performance
- Alternatives
- Q&A

- **Quick Recap: Transactions**
- What problem are we trying to solve?
- Quick Recap: KAFKA
- Solution
- Performance
- Alternatives
- Q&A

What is a transaction (in computer science)?

- An atomic unit of work
- Must either complete entirely or not at all
- Moves a system from one valid state to another
- Can be distributed or local
- ACID properties

ACID

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity

- Transactions are either completed entirely or not at all
- If one part fails then the whole transaction fails

Consistency

- Transactions move a system from one valid state to another

Isolation

- Concurrent transactions leave the system in a state as if they were serialized

Durability

- Changes are stored permanently

Distributed Transactions

- Involves multiple network hosts
- Common implementations use 2-Phase-Commit (2PC) to guarantee ACID properties
- 2PC requires a transaction coordinator

Distributed Transactions - Java

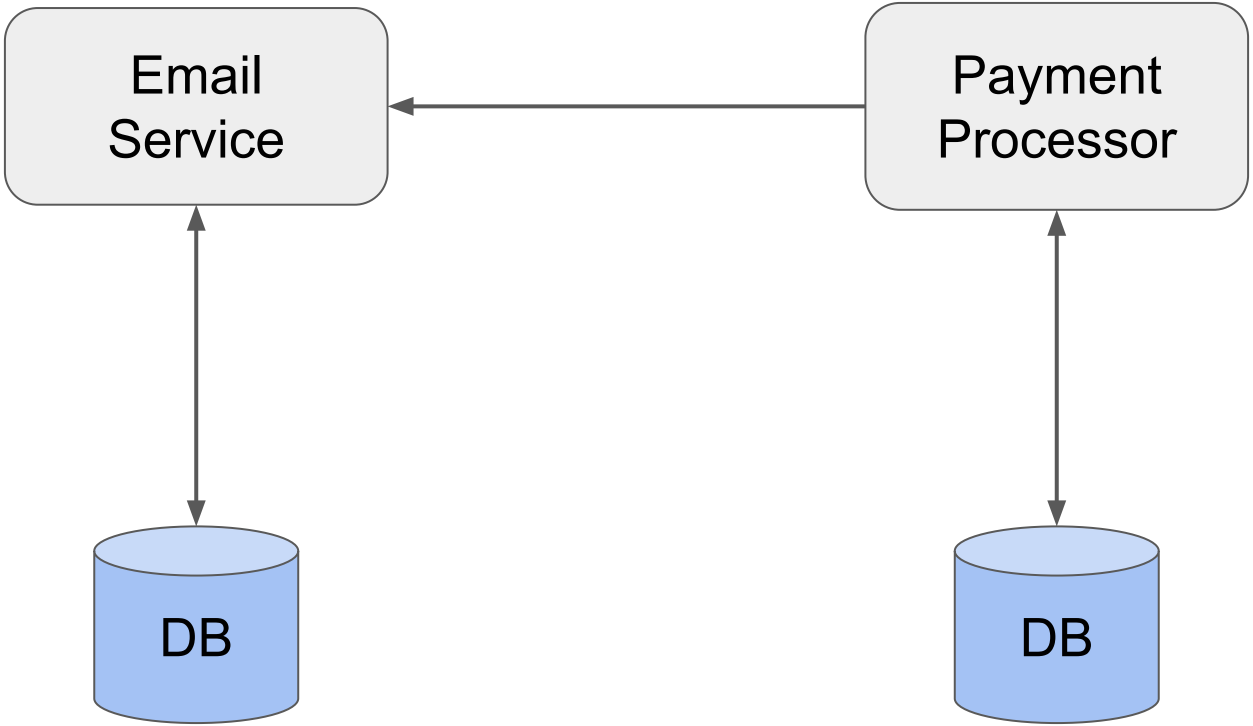
- Java Transaction API (JTA) to implement transactional resources
- EJB containers provide JTA support out-of-the-box
- Standalone transaction manager (Atomikos, Bitronix, etc)

- Quick Recap: Transactions
- **What problem are we trying to solve?**
- Quick Recap: KAFKA
- Solution
- Performance
- Alternatives
- Q&A

The Monolith

```
class PaymentProcessor {

   @Transactional // local database transaction
   void processPayment(Payment payment) {

      payoutToRecipient(payment);

      notifyCustomer(payment);
   }
}
```

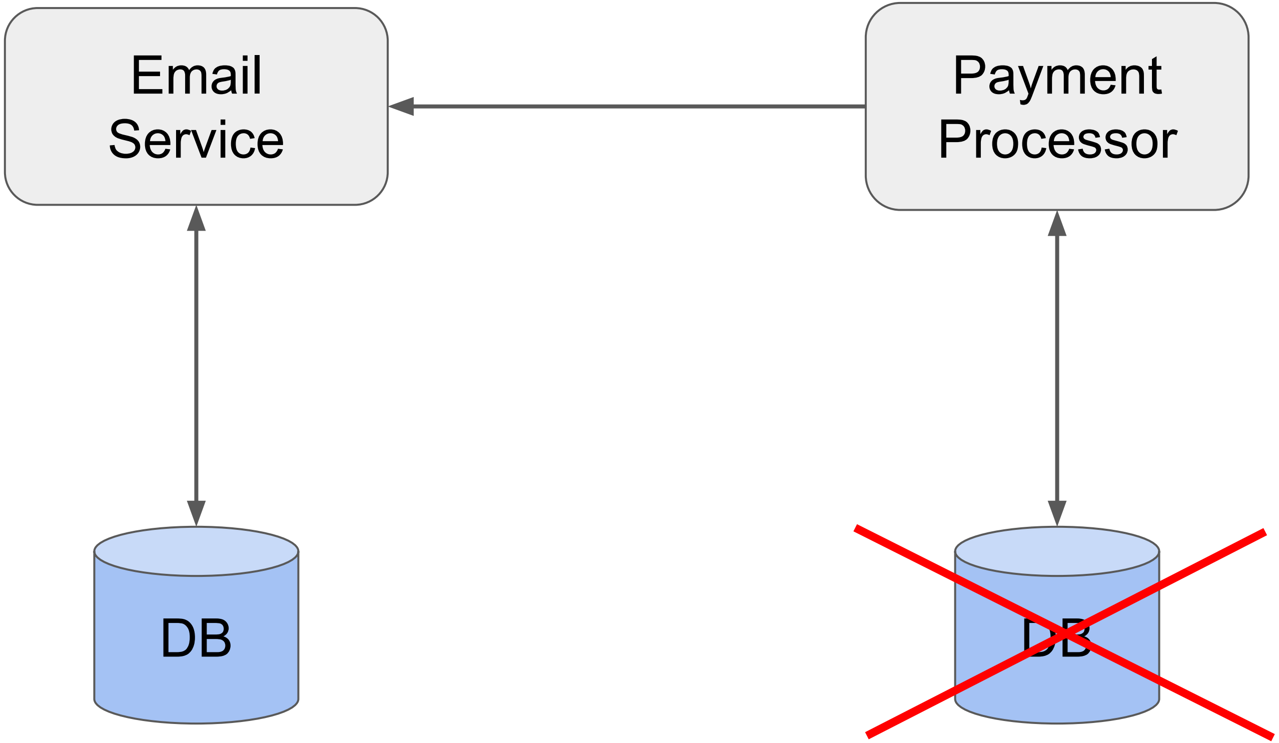
Multiple services and databases



Adding Microservice Calls - Happy Flow

```
class PaymentProcessor {

    @Transactional // now what does that mean?
    void processPayment(Payment payment) {

        payoutToRecipient(payment);

        emailClient.notifyCustomer(payment);
    }
}
```

Multiple Services and Databases



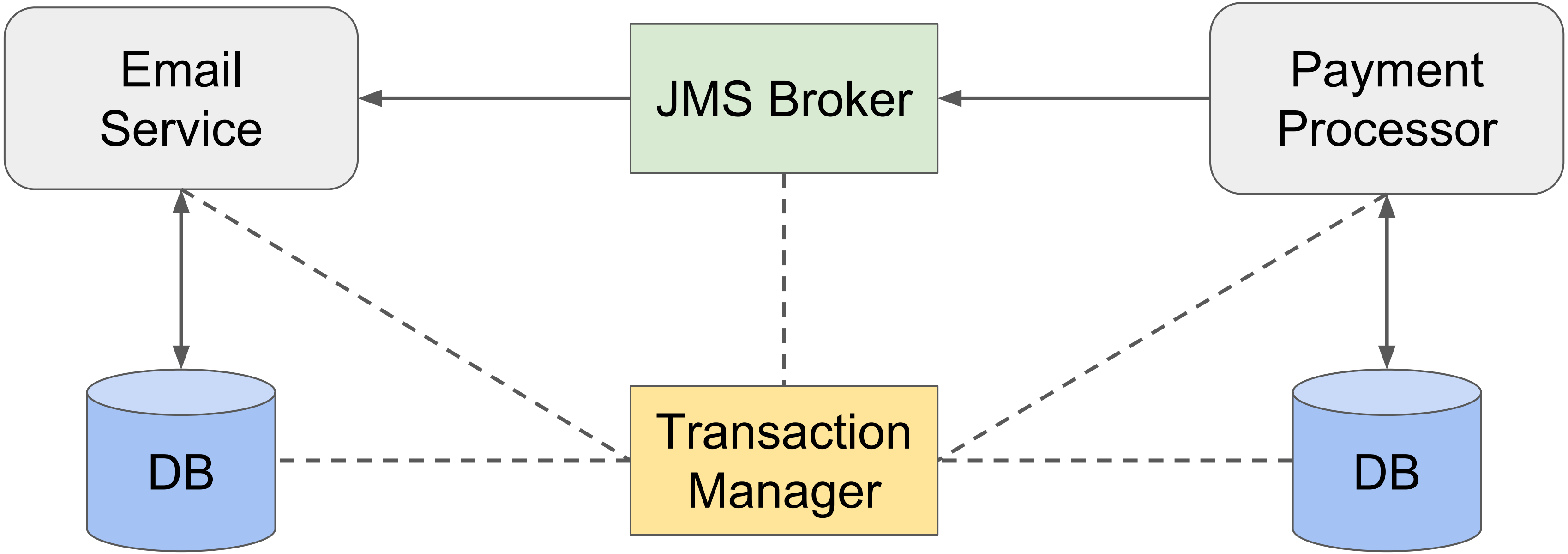
Adding Microservice Calls - Unhappy Flow

```
class PaymentProcessor {

    @Transactional // still just a local transaction
    void processPayment(Payment payment) {
        payoutToRecipient(payment);
        emailClient.notifyCustomer(payment);

        transactionManager.onRollback(() -> {
            emailClient.unnotifyCustomer(payment); ?????
        });
    }
}
```

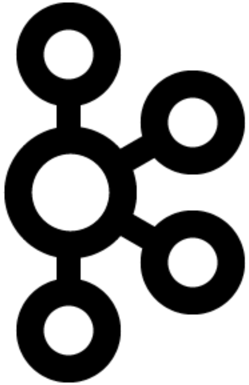
Async Processing

```
class PaymentProcessor {

   @Transactional(transactionManager = “jta”)
   void processPayment(Payment payment) {

      payoutToRecipient(payment);

      sendToJmsBroker(new CustomerNotification(...));
   }
}
```

Multiple Services and Databases + JMS Broker



- Quick Recap: Transactions
- What problem are we trying to solve?
- **Quick Recap: KAFKA**
- Solution
- Performance
- Alternatives
- Q&A



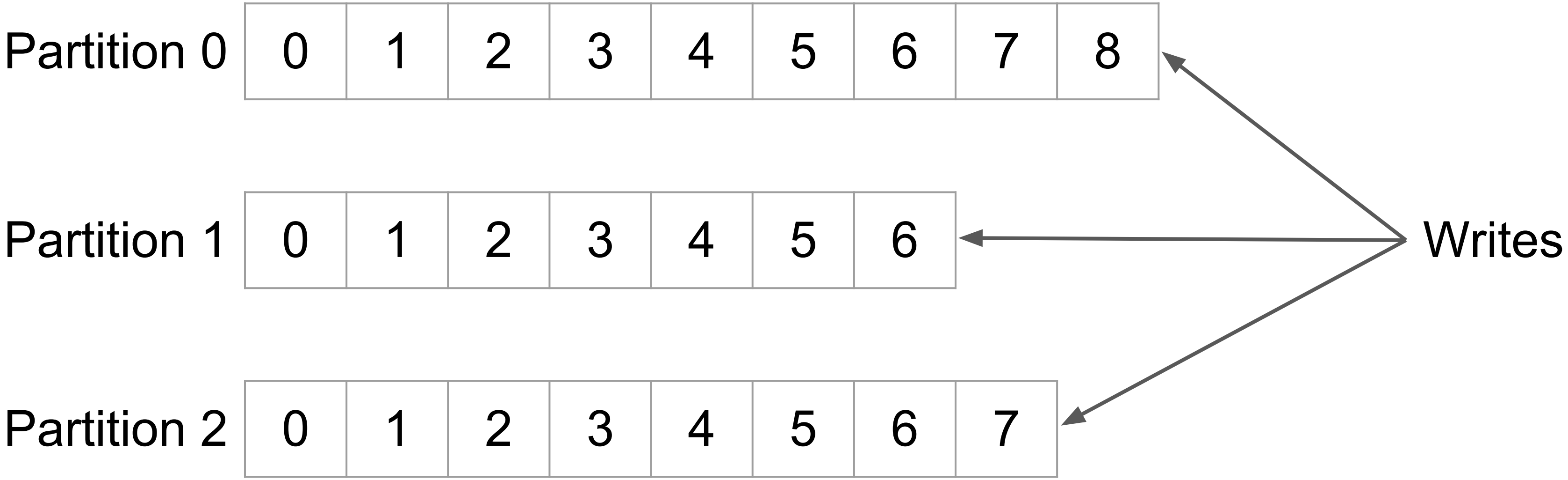
APACHE
kafka®

A distributed streaming platform

Why KAFKA?

- High availability
- High throughput
- (Eventually) persistent

KAFKA - Topics and Logs



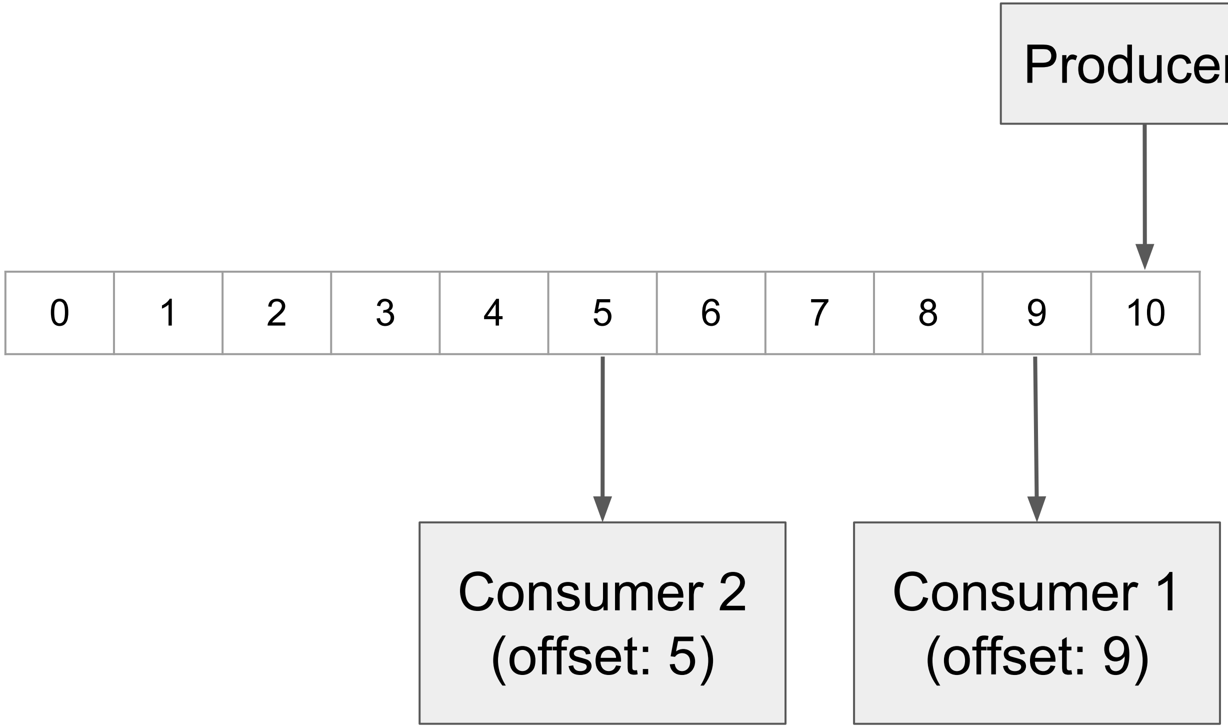
Partition 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

Partition 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6

Partition 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

Writes

KAFKA - Producer / Consumer



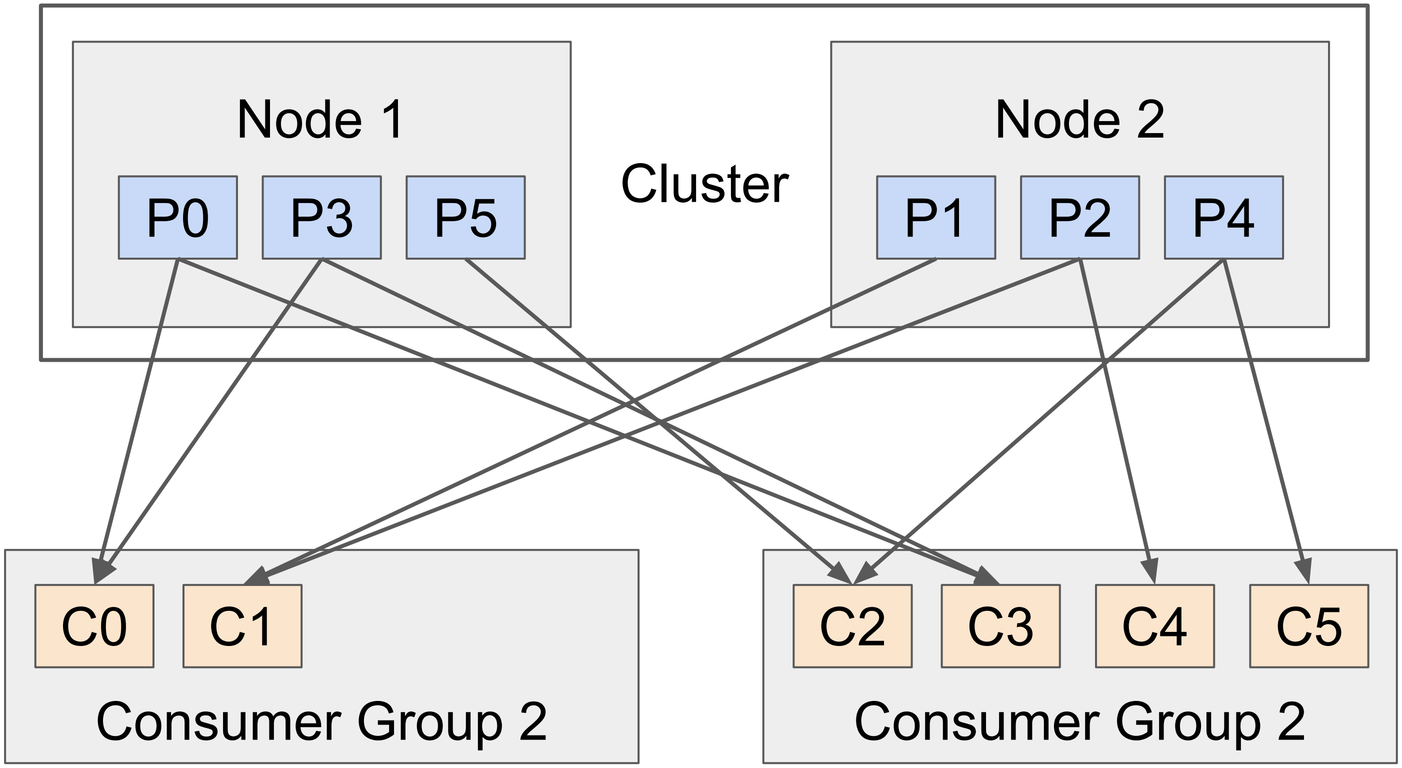
Producers

0	1	2	3	4	5	6	7	8	9	10

Consumer 2
(offset: 5)

Consumer 1
(offset: 9)

KAFKA - Nodes and Consumer Groups



KAFKA - Design Notes

- At least once delivery
- Uses pagecache instead of heap (by default no fsync)
- Uses linear reads and writes for throughput (X00MB/sec)

Our Cluster

- 5 Nodes
- 4 vCPUs, 16G Memory, 500G sdd, 1G NIC per node
- ISR = 2, replication factor = 3
- Runs on virtualized hardware
- 3 zookeepers

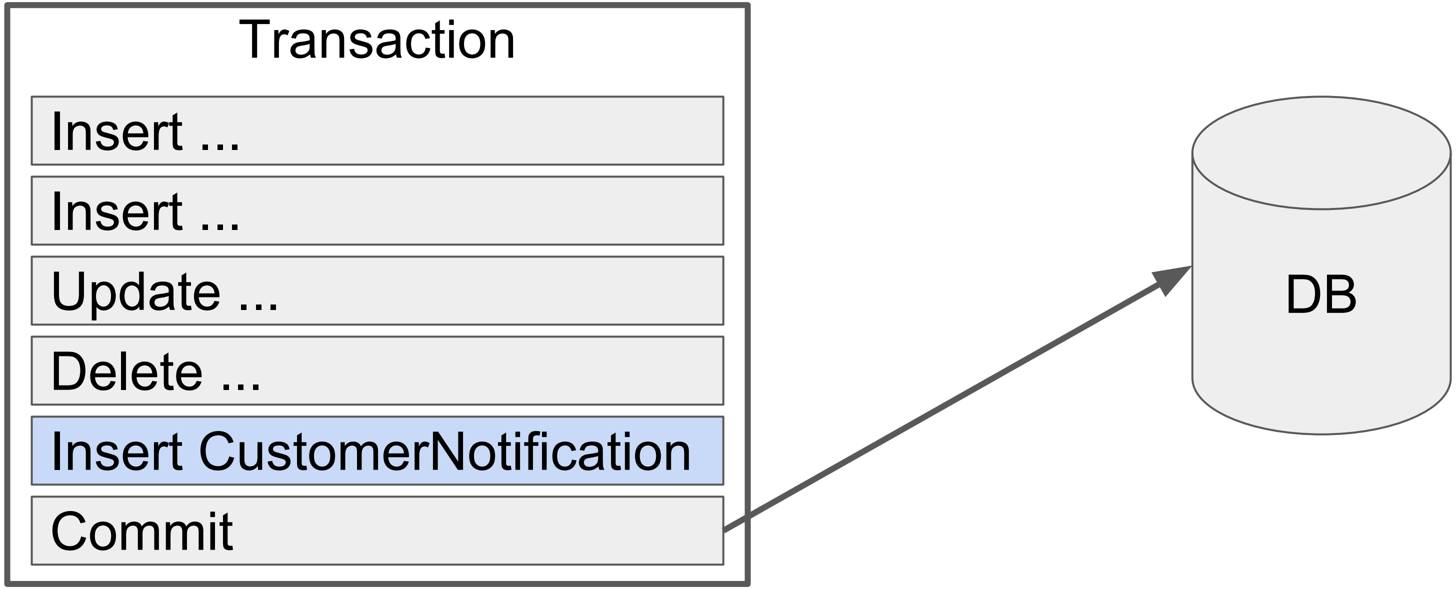
- Quick Recap: Transactions
- What problem are we trying to solve?
- Quick Recap: KAFKA
- **Solution**
- Performance
- Alternatives
- Q&A

Max input: 230 V
© ColiN00B

Solution - Overview

- Split transactions into small and fast blocks
- Use only local transactions
- Store and forward notifications to the next service
- Use KAFKA for high throughput

Store Message



Payment Processor

```
class PaymentProcessor {

   @Transactional // local database transaction
   void processPayment(Payment payment) {

      payoutToRecipient(payment);

      saveToDatabase(new CustomerNotification(...));
   }
}
```

Message Implementation - 1/2

```
abstract class Message {

    private String uuid = UUID.randomUUID().toString();

    abstract String getDestination();
}
```

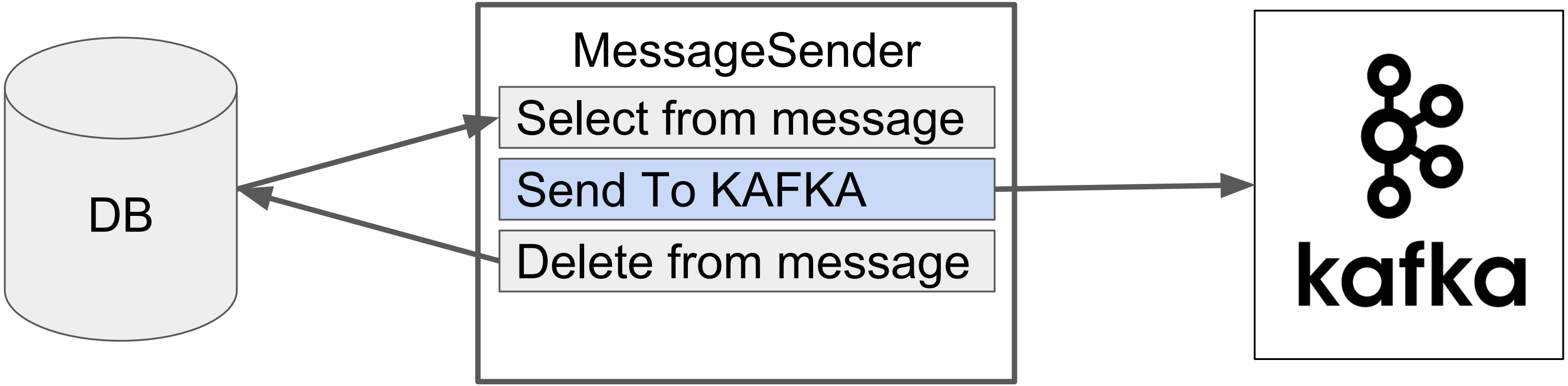
Message Implementation - 2/2

```
class CustomerNotification extends Message {

   ...

   String getDestination() {
      return “topic.CustomerNotification”;
   }
}
```

Message table

```
create table message (

   destination varchar(255) not null,

   payload text not null // json or any other format
)
```

Poll Message and Send



DB

MessageSender

Select from message

Send To KAFKA

Delete from message

kafka

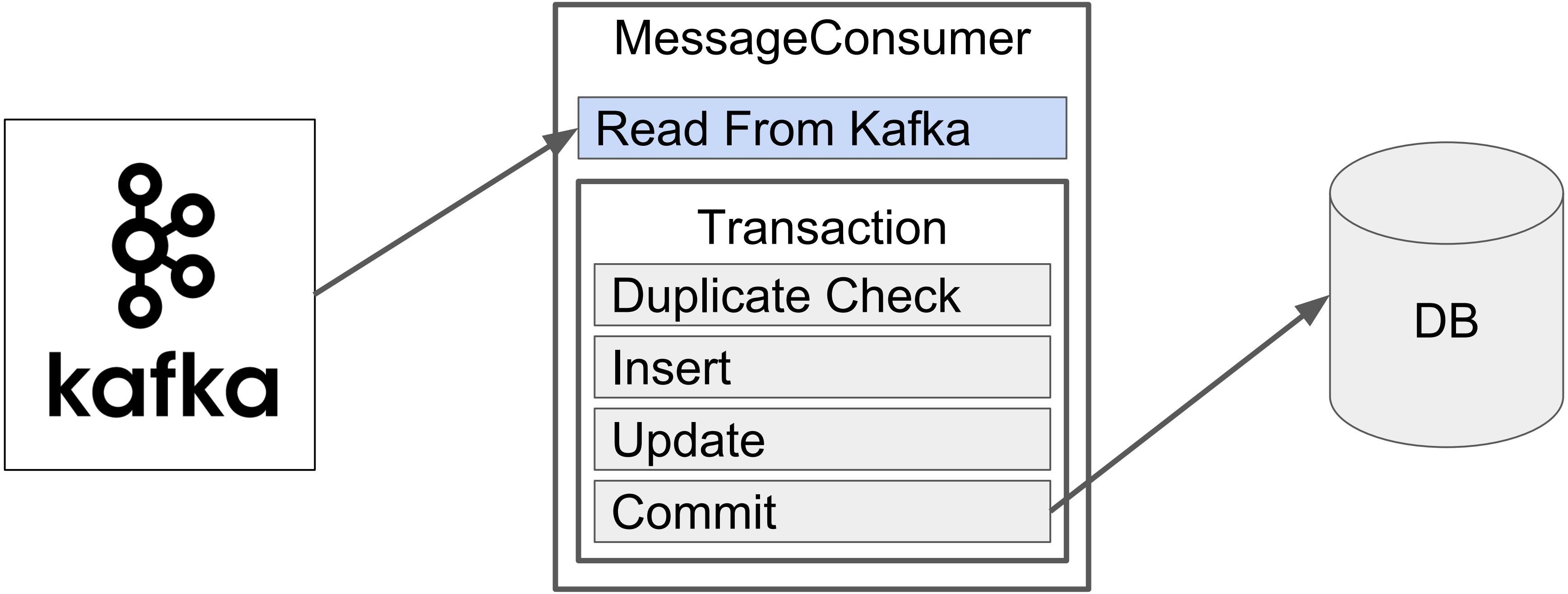
Sending Messages - 1/2

```
class MessageSender implements Runnable {

    void run() {
        while (running) {
            Message message = pollFromDatabase();
            sendToKafka(message);
        }
    }
}
```

Sending Messages - 2/2

```
sendToKafka(Message message) {

   String topic = message.getDestination();
   String value = serialize(message);

   producer.send(new ProducerRecord(topic,value),
      (...) -> removeFromDatabase(message)
   );
}
```

Consume and De-duplicate



kafka

MessageConsumer

Read From Kafka

Transaction

Duplicate Check

Insert

Update

Commit

DB

Consuming Messages - 1/3

```
class MessageProcessor implements Runnable {
    void run() {
        while (true) {
            for (ConsumerRecord r : consumer.poll(...)) {
                Message message = parse(r);
                processMessage(message);
            }
            consumer.commitAsync();
        }
    }
}
```

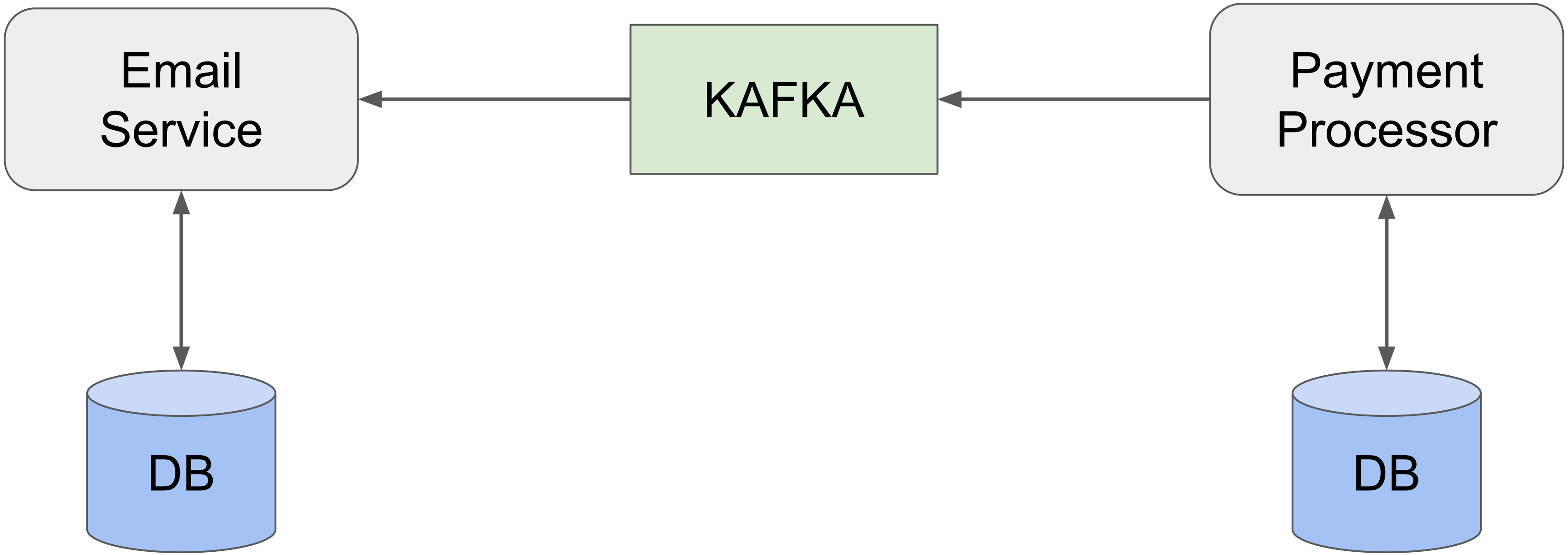
Consuming Messages - 2/3

```
@Transactional // local database transaction
void processMessage(Message message) {
   if (!isDuplicate(message)) {
      ...
   }
}
```

Consuming Messages - 3/3

```
boolean isDuplicate(Message message) {
   try {
      saveMessageUuidToDatabase(message.getUuid());
      return false;
   } catch (DuplicateKeyException e) {
      return true;
   }
}
```

Multiple Services and Databases + KAFKA



- Quick Recap: Transactions
- What problem are we trying to solve?
- Quick Recap: KAFKA
- Solution
- **Performance**
- Alternatives
- Q&A

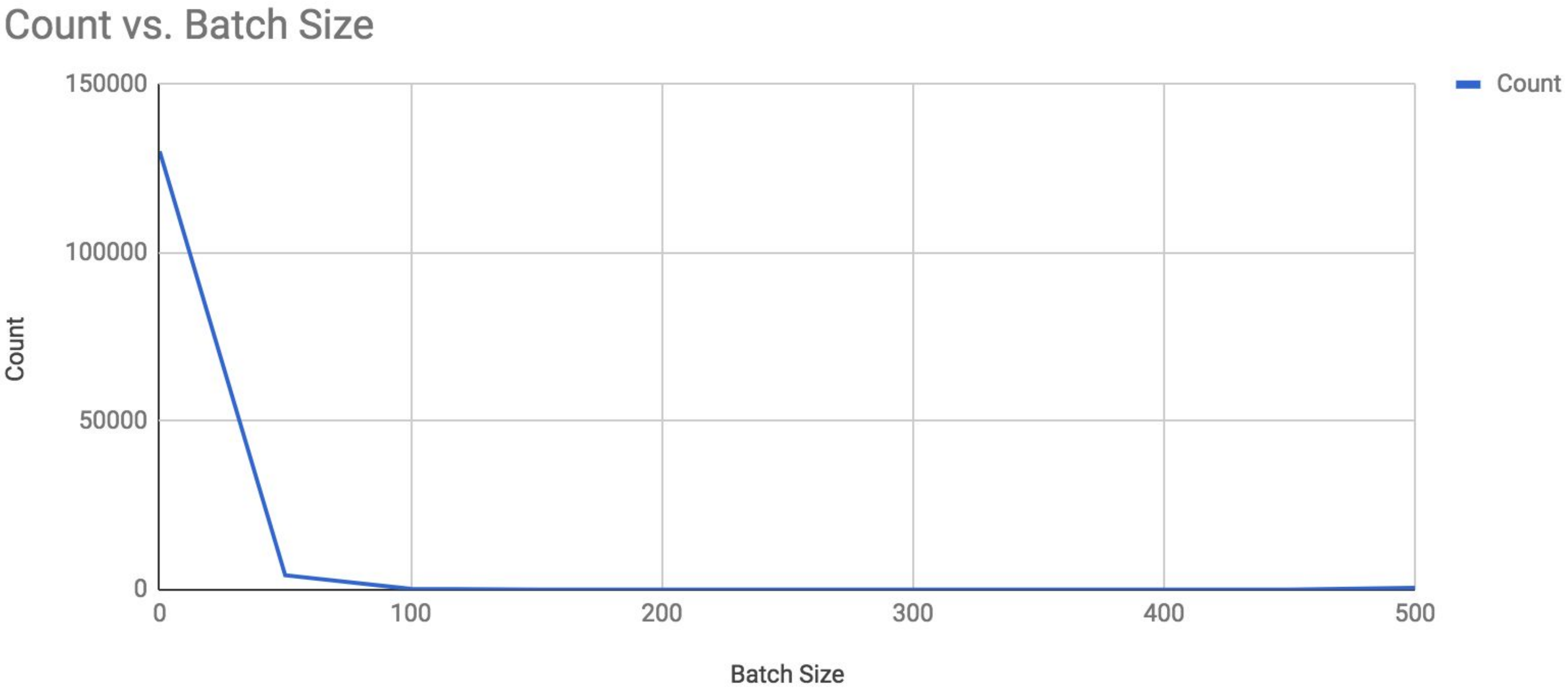
Which components are we monitoring?

- Message loader
- Message sender
- Message consumer (incl. dedup)
- End-to-End

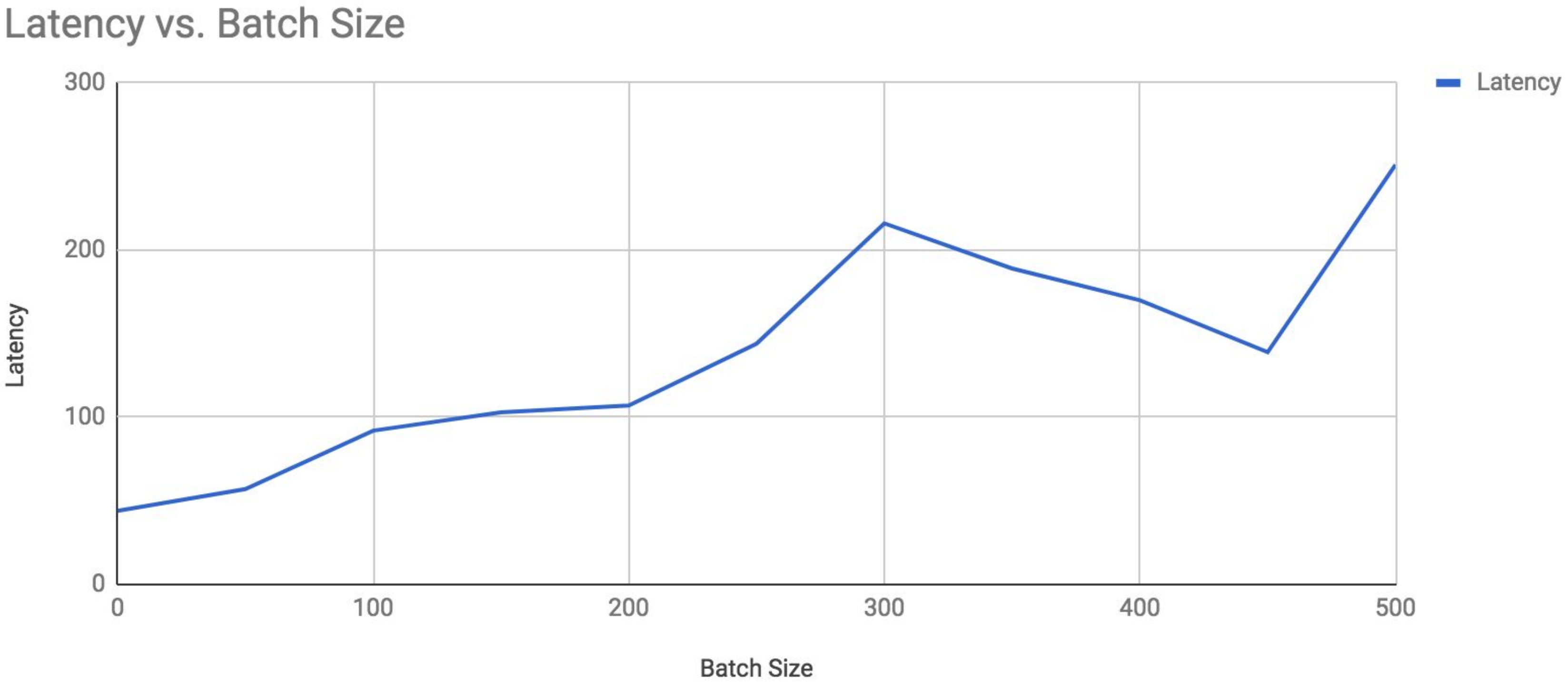
Message Loader

- Avg batch size: 20 (max 500)
- 500ms sleep if no new messages available
- Average 90 ms/batch
- MySQL 5.7 innodb

Message Loader - Count vs Batch Size



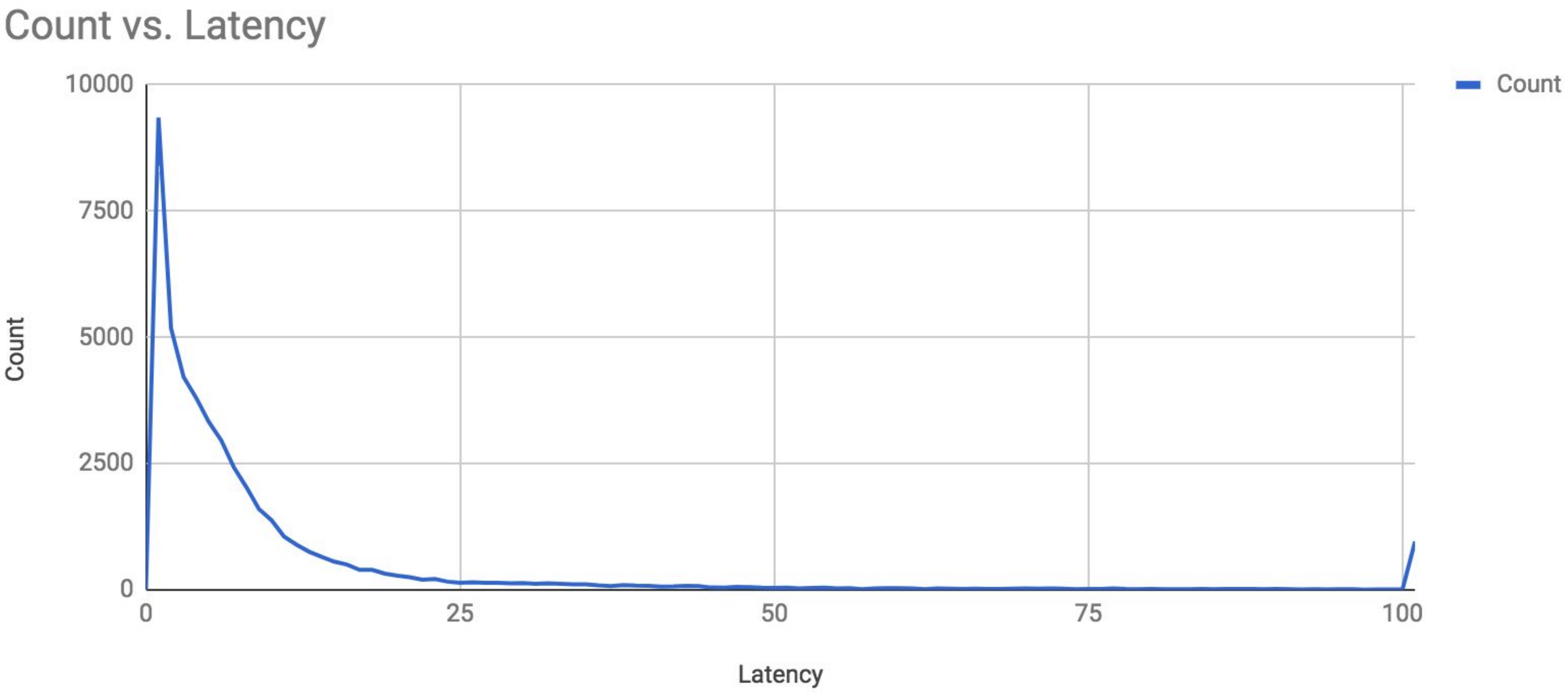
Message Loader - Latency vs Batch Size



Message Sender

- 4 publisher threads
- Throughput is up to 700 msg/sec on a busy day

Message Sender - Count vs Latency



Message Consumer

- One thread per partition
- ~ 350 msg/sec per partition
- Dedup time: 3 ms/msg using MySQL
- Fast dedup is key to high throughput

End-to-end

- We care more about throughput than latency
- We don’t have millisecond latency data :-(
- But we measure it in seconds!
- On average our latency is < 1 sec

- Quick Recap: Transactions
- What problem are we trying to solve?
- Quick Recap: KAFKA
- Solution
- Performance
- **Alternatives**
- Q&A

Alternative Solutions

- Using a traditional JMS broker with transaction support

  (Artemis, ActiveMQ, TIBCO, etc.)

- JBoss REST-AT (still a draft, supported by WildFly)
- Try to write your own XA stuff?

Modify our solution to your liking!

- Choose a different broker or messaging platform
- Choose a different database
- Replace the broker with direct service calls
- Add commit hooks for low latency

Thank You!

Harald Wendel

harald.wendel@transferwise.com