Exactly-once Delivery

PYGMALIOS

Ján Antala
@janantala / j.antala@pygmalios.com

Mathias Verraes
@mathiasverraes

Follow

There are only two hard problems in distributed systems: 2. Exactly-once delivery 1. Guaranteed order of messages 2. Exactly-once delivery

8:40 PM - 14 Aug 2015

7,001 Retweets **4,996** Likes

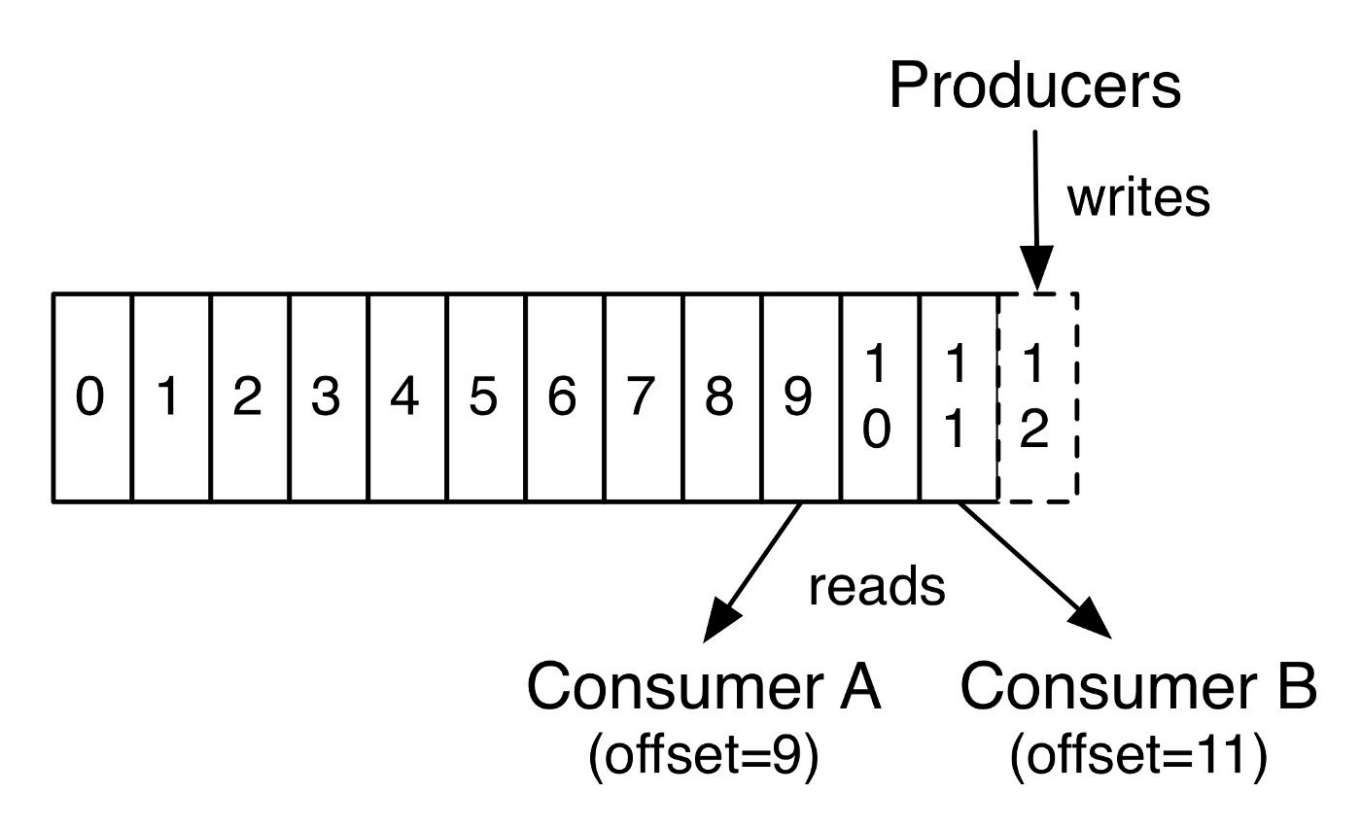
68 7.0K 5.0K

Kafka: on-disk circular buffer

distributed, fast, resilient

- Publish & subscribe, like MQ
- Real time data streaming
- Distributed replicated cluster

Originally producer & consumer client



Everyone  Kafka

Monitoring Kafka on Docker Cloud

https://sematext.com/blog/2016/04/19/monitoring-kafka-on-docker-cloud/

What can fail?

Broker

producer-to-broker RPC

client

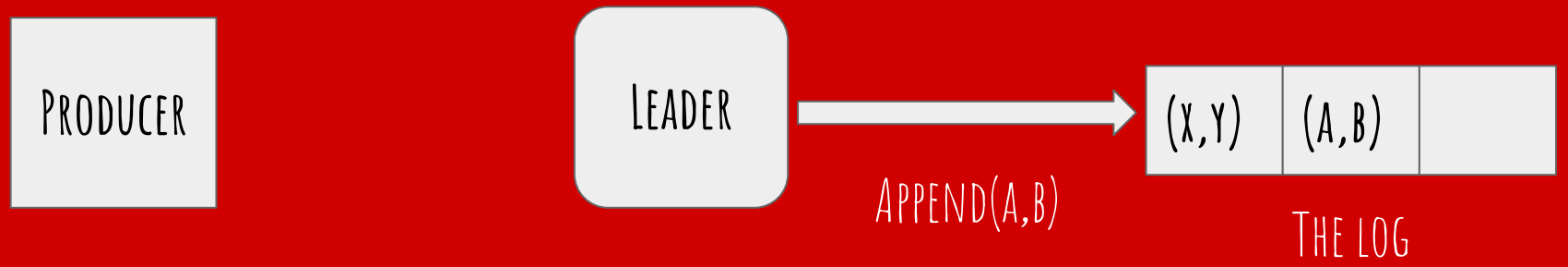
At least once delivery

At least once delivery

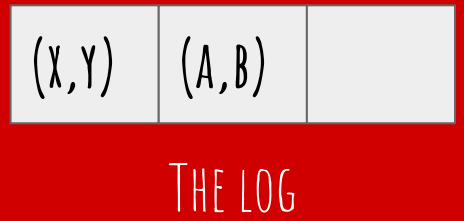
At least once delivery

At least once delivery

At least once delivery



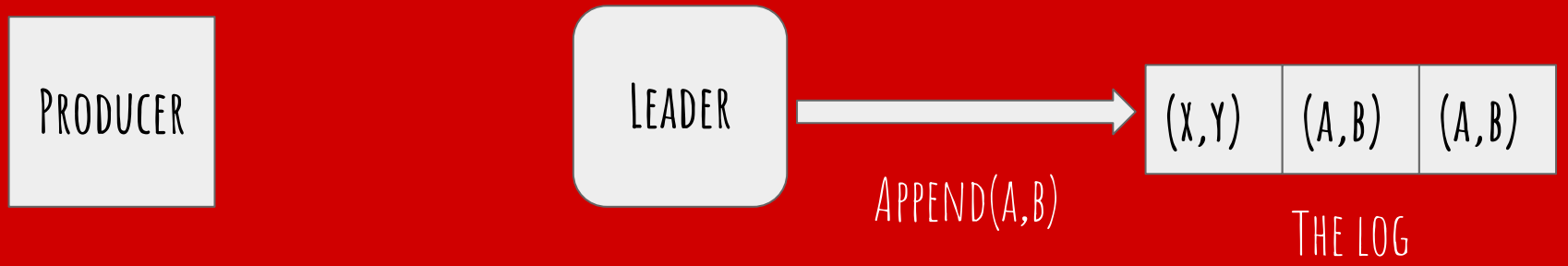
At least once delivery



At least once delivery

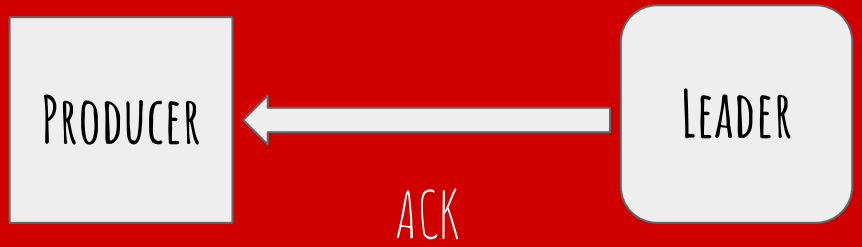
Producer retries can introduce duplicates

At least once delivery



Producer retries can introduce duplicates

At least once delivery



(x,y)	(a,b)	(a,b)

The log

Producer retries can introduce duplicates

At most once delivery

On failure restart at last saved offset

messages are lost

Pygmalios

Customer Experience Analytics For Physical Stores

Problem:

Tracking the customer behavior and managing the customer experience online is a norm.

In the physical retail it is unavailable.

How it Works?

BEHAVIOR DATA SOURCES

TRAFFIC

WiFi Routers, Monocular Video Sensors

BROWSING

Radio Based Positioning System, 3D Stereo Video Sensors

QUEUE

Monocular Video Sensors

DEMOGRAPHICS

Monocular Video Sensors

BUYING

POS Data Integration

PYGMALIOS CLOUD

All the sensors and external factors' data is processed in a safe cloud environment.

EXTERNAL FACTORS

Weather, holidays, other external factors.

OUTPUTS

USER INTERFACE

Web and mobile friendly user interface for an easy access to reports.

CUSTOMER SUCESS

Dedicated service to help customers understand the data.

API

Programmable interface for an easy integration with other business services.

Outside Traffic Conversion

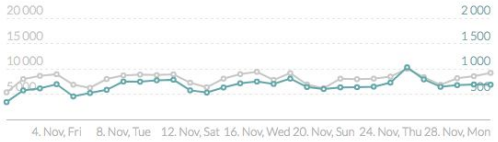
Data Source: WiFi

Capture Rate

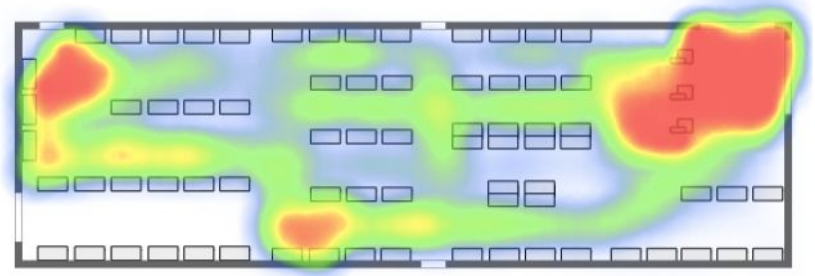
8,45 %

+17,0 %

Traffic Trends



Outside Traffic Intensity	240 808	+0,7 %
In-store Traffic Intensity	20 360	+17,8 %



Stay Time

Data Source: WiFi

Avg. Stay Time

20min 13s

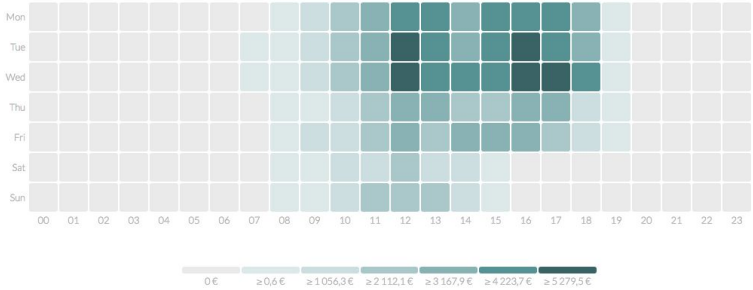
-0,5 %

Visits by Stay Time

Productivity by Hours in Week

Data Source: POS, People Counter

Conversion Rate | Visits | Transactions | Revenue | ATV | UPT



Bigger picture: lambda architecture

Or how we achieved unique messages

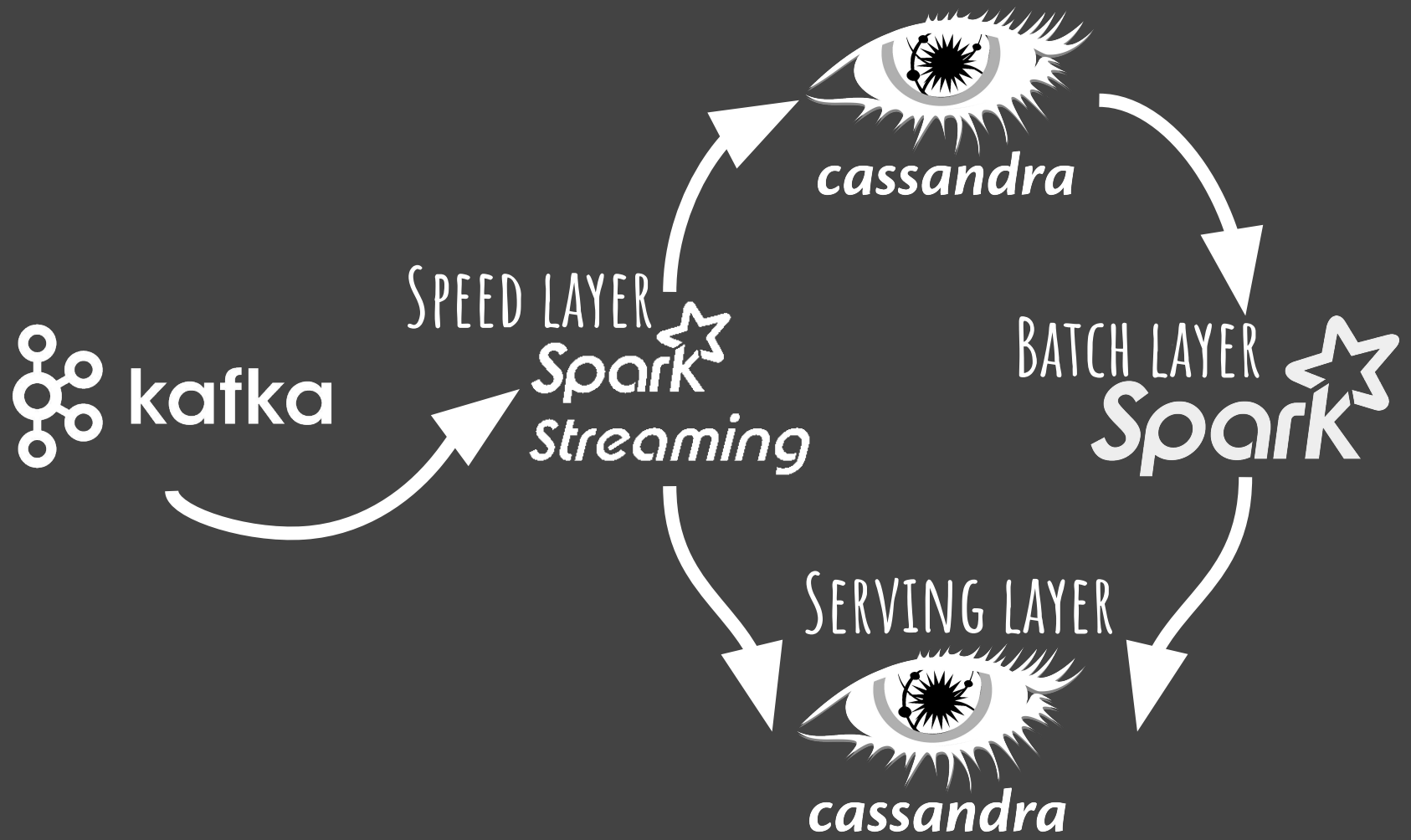
Lambda architecture

Immutable master dataset

Speed layer

Batch layer

Serving layer


kafka
Speed layer
Spark
Streaming
cassandra
Batch layer
Spark
Serving layer
cassandra

Master dataset grows forever

Contains everything, immutable source of truth

Batches generate views for queries

Scheduled jobs, slow, fix consistency

Speed layer compensates slowness

But can cause temporary inconsistency

- Linear scalabiliy
- High availability
- Super fast writing
- All nodes are equal: no SPoF

How we solved delivery issues?

Example - POS data stream:

- time delays
- At least once delivery (multiple messages)
- Time-series data!

```
// Point-of-Sale JSON message example:
{
    happened_at: 2017-09-19T22:48:03+00:00,
    operation_id: 1,
    transaction_id: 42,
    vendor_fiscal_module: 123,
    transactions: [ ... ] // Item title, price, amount, ...
}
```

```scala
import akka.actor.Actor

class PosTransactionSpeedActor(ssc: StreamingContext) extends Actor {
    ...
    val kafkaStream: DStream[RawPosTransaction] = KafkaUtils
      .createDirectStream(ssc, kafkaParams, "posTransactionTopic")
      .map(RawPosTransaction.fromKafka)

    // Save raw data
    kafkaStream
      .saveToCassandra(cassandraKeyspace, cassandraTableRawPosTransaction)

    // Save aggregated data (15-minute windows)
    kafkaStream
      .map(AggPosTransaction.fromRaw)
      .saveToCassandra(cassandraKeyspace, cassandraTableAggPosTransaction)

    ...
}
```

```
// Cassandra table definition - note the primary key:
CREATE TABLE raw_pos_transaction_byday (
    happened_at timestamp,
    operation_id text,
    transaction_id text,
    year_day text,
    transactions LIST<text>,
    vendor_fiscal_module text,
    PRIMARY KEY (( year_day, operation_id), happened_at)
) WITH CLUSTERING ORDER BY ( happened_at DESC )
AND compaction = {
    'class' : 'SizeTieredCompactionStrategy'
};
```

- Records with the same key are updated
- Data are sorted by timestamp in the partition

What if a message comes twice?

- Batch layer will fix it

Lessons learned

- Be careful about granularity
  (multiple transactions in the same second from single POS)
- But what if want to send <u>real-time</u> alerts?

Kafka exactly-once delivery to the rescue!

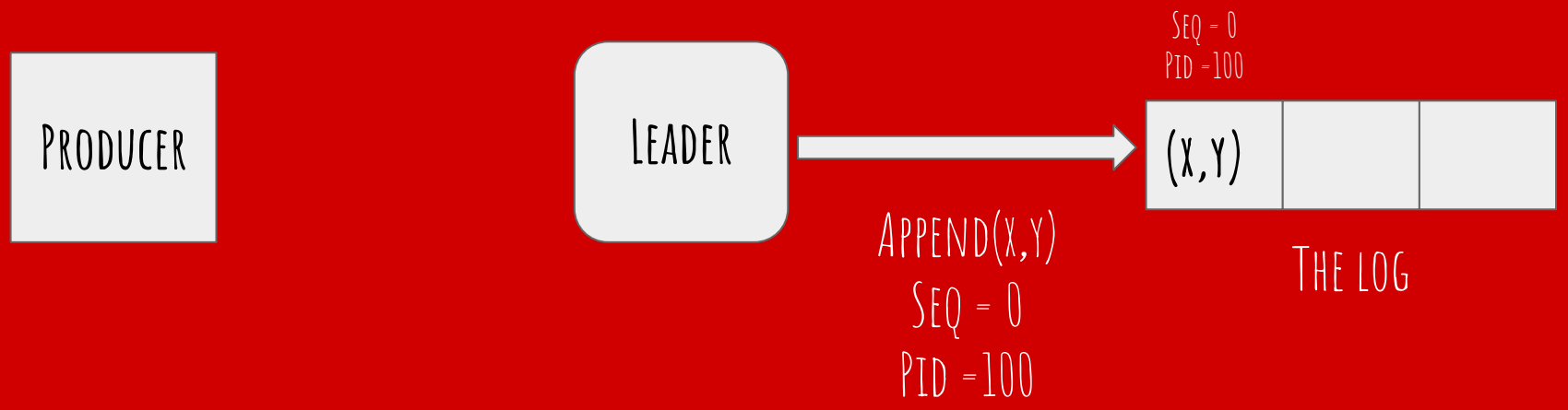
How to guide

What's new?

- Exactly once in order delivery per partition
- Atomic writes across multiple partitions
- Performance considerations

The idempotent producer

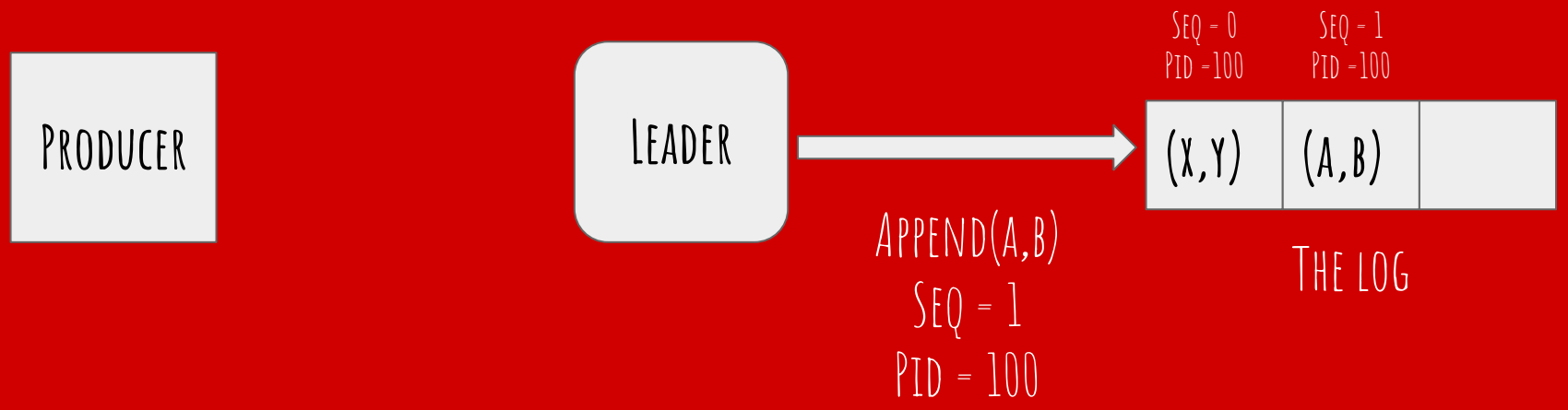
The idempotent producer



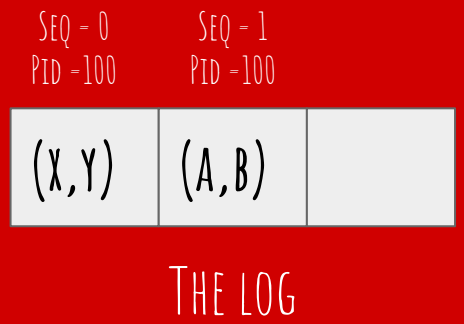
The idempotent producer

The idempotent producer

The idempotent producer



The idempotent producer



The idempotent producer

Messages are repeated, but we don't care

The idempotent producer

messages are repeated, but we don't care

No API changes

Per partition

enable.idempotence = true

Transactions

Atomic writes across multiple partitions

```
// The new transactions Producer API

producer.initTransactions();
try {
    producer.beginTransaction();
    producer.send(record1);
    producer.send(record2);
    producer.commitTransaction();
} catch(ProducerFencedException e) {
    producer.close();
} catch(KafkaException e) {
    producer.abortTransaction();
}
```

SEND A BATCH OF MESSAGES TO MULTIPLE PARTITIONS SUCH THAT EITHER ALL MESSAGES IN THE BATCH ARE EVENTUALLY VISIBLE TO ANY CONSUMER OR NONE ARE EVER VISIBLE TO CONSUMERS

Customer side

isolation.level

- READ_COMMITED

  After transaction is committed

- READ_UNCOMMITED

  Without waiting to commit

Performance boost!

The new message format - variable length encoding

Starting from batch size of 2

even if you don't use any of the exactly-once features

Exactly once stream processing

all of the processing to happen exactly once

processing_mode = exactly_once

OBSERVABLY EXACTLY-ONCE GUARANTEE

Links:

- Exactly-once Semantics are Possible / Confluent
  https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/
- Exactly Once Delivery and Transactional Messaging / Kafka Improvement Proposal
  https://cwiki.apache.org/confluence/display/KAFKA/KIP-98+-+Exactly+Once+Delivery+and+Transactional+Messaging
- Exactly Once Delivery and Transactional Messaging in Kafka / The definitive design
  https://docs.google.com/document/d/11Jqy_GjUGtdXJK94XGsEIK7CP1SnQGdp2eF0wSw9ra8/edit#heading=h.14ub5zyE01nh