Time & messages order in Kafka

In Kafka messages are ordered based on their offset (generated when message is written in partition) - nit based on their timestamp

Messages order can be guaranteed only at partition level (based on offset)

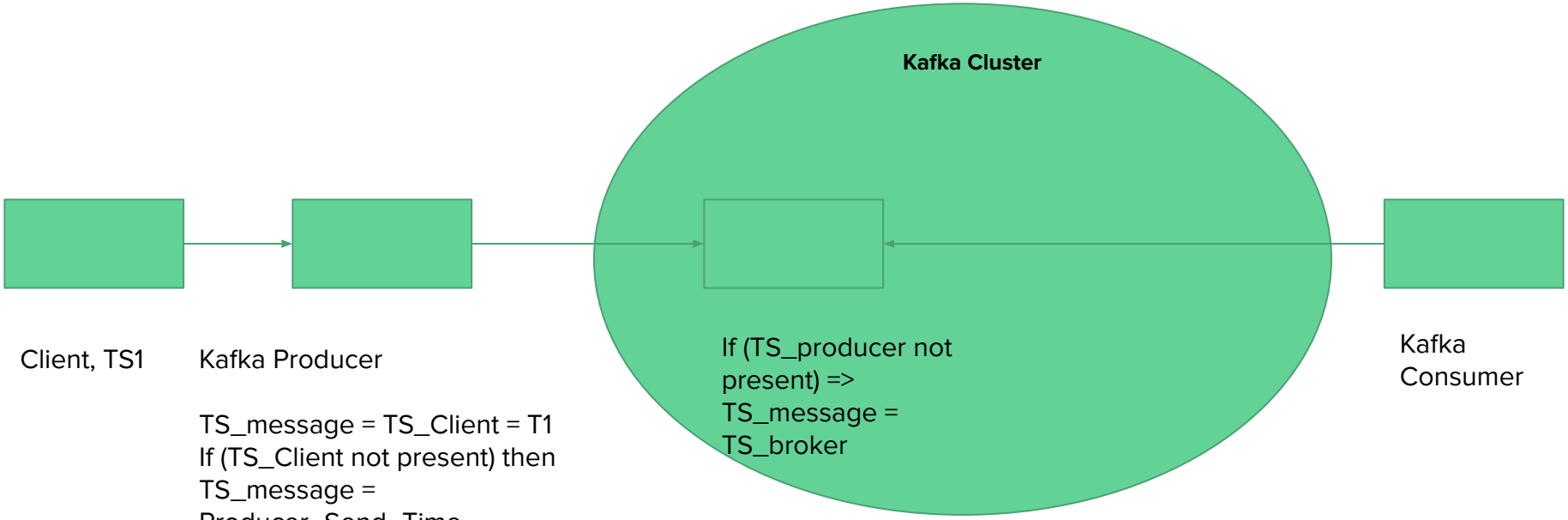
But even at partition level Kafka guarantees that the consumer will read the messages in the same order in which the messages reached Kafka.

Can we guarantee that the original order (producer side order) of the messages is preserved when reaching Kafka? Yes - idempotent producer

Is the timestamp of the message relevant in the order of the messages in the partition? No - timestamp is relevant for messages retention, but the order of the messages is given by their offsets.

-

Starting with Kafka 0.10: createTime OR LogAppendTime



Topic: Time of a message

Topic configuration:

- **Message.timestamp.type** // two possible values
  - createTime (default) // If producer sends the timestamp it will be used as the timestamp of the message.
  - LogAppendTime
- **max.message.time.difference.ms** - This configuration only works when message.timestamp.type=CreateTime. The broker will only accept messages whose timestamp differs no more than max.message.time.difference.ms from the broker local time.

Issues

If we use the timestamp sent by the client or producer as message timestamp, there are some issues to be kept under consideration:

- Can I guarantee that the producer's clock is synchronized
- I ingest sensor data and there is no local clock
- I use an older producer (< 0.10) and cannot see timestamp at all

Message.timestamp.type (topic level) and log.message.timestamp.type (broker level) - 2 possible values

- createTime - the time is sent by the producer
- LogAppendTime - broker overwrites the producer time

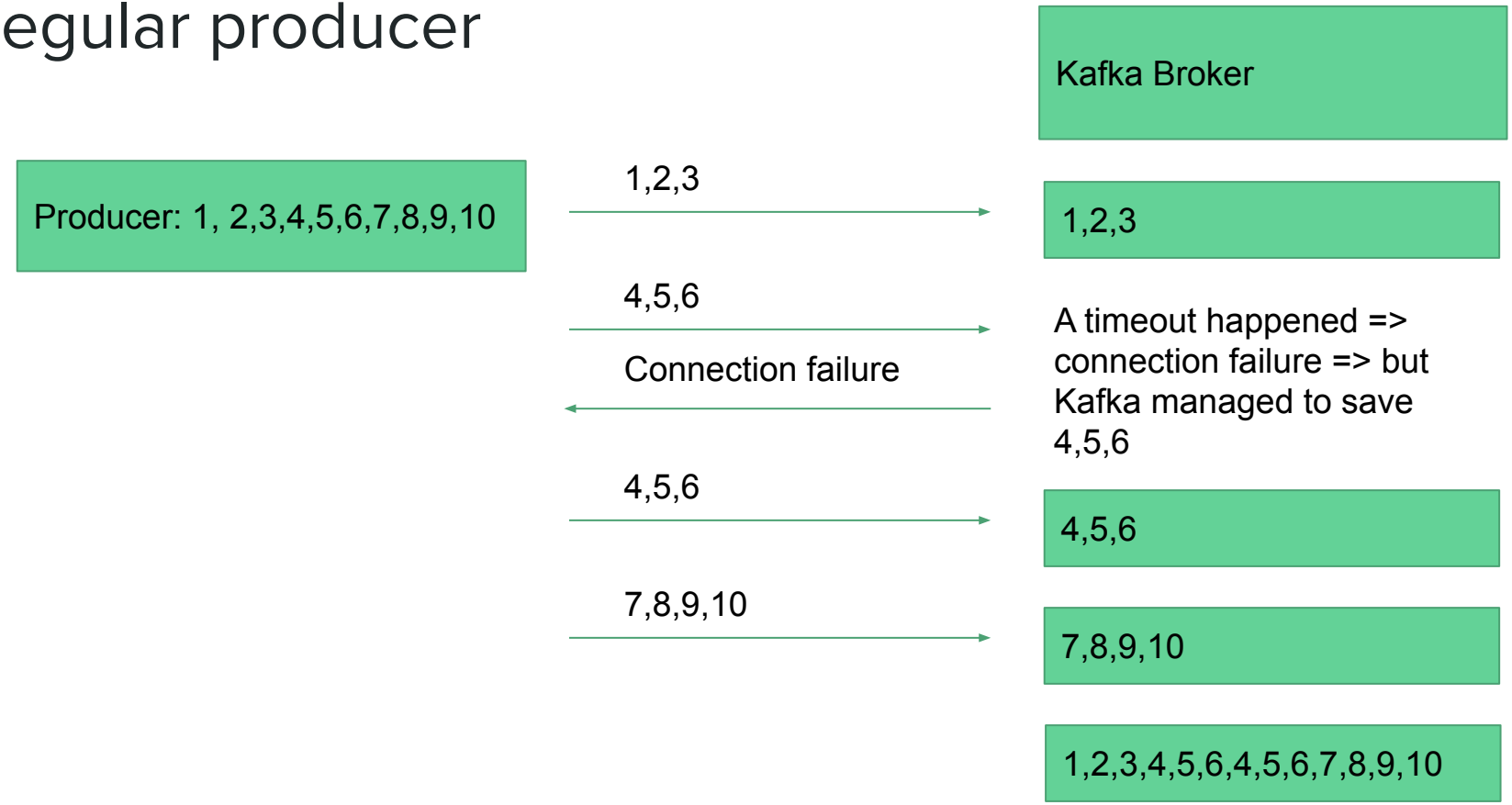
Recommendation: In case the producer cannot send the time, include the time as a column in the message/payload.

If you need strong ordering guarantees of the messages

Single writer principle (strict)

- **1 producer/partition**
- **enable.idempotence = true** // ensures that messages always get delivered, in the right order and without duplicates.

Regular producer



idempotent.producer = true

Each producer gets assigned a ProducerId and it includes its PID every time it sends messages to a broker. Additionally, **each message gets a monotonically increasing sequence number.** A separate sequence is maintained for each topic partition that a producer sends messages to. On the broker side, on a per partition basis, it keeps track of the largest PID-Sequence Number combination is has successfully written. When a lower sequence number is received, it is discarded.

M1 (PID: 1, SN: 1) - written to partition. For PID 1, Max SN=1
M2 (PID: 1, SN: 2) - written to partition. For PID 1, Max SN=2
M3 (PID: 1, SN: 3) - written to partition. For PID 1, Max SN=3
M4 (PID: 1, SN: 4) - written to partition. For PID 1, Max SN=4
M5 (PID: 1, SN: 5) - written to partition. For PID 1, Max SN=5
M6 (PID: 1, SN: 6) - written to partition. For PID 1, Max SN=6
M4 (PID: 1, SN: 4) - rejected, SN <= Max SN
M5 (PID: 1, SN: 5) - rejected, SN <= Max SN
M6 (PID: 1, SN: 6) - rejected, SN <= Max SN
M7 (PID: 1, SN: 7) - written to partition. For PID 1, Max SN=7
M8 (PID: 1, SN: 8) - written to partition. For PID 1, Max SN=8
M9 (PID: 1, SN: 9) - written to partition. For PID 1, Max SN=9
M10 (PID: 1, SN: 10) - written to partition. For PID 1, Max SN=10

idempotent.producer = true

Note: enabling idempotence requires max.in.flight.requests.per.connection to be less than or equal to 5, retries to be greater than 0 and acks must be 'all'.

M1 (PID: 1, SN: 1) - written to partition. For PID 1, Max SN=1
M2 (PID: 1, SN: 2) - written to partition. For PID 1, Max SN=2
M3 (PID: 1, SN: 3) - written to partition. For PID 1, Max SN=3
M4 (PID: 1, SN: 4) - written to partition. For PID 1, Max SN=4
M5 (PID: 1, SN: 5) - written to partition. For PID 1, Max SN=5
M6 (PID: 1, SN: 6) - written to partition. For PID 1, Max SN=6
M4 (PID: 1, SN: 4) - rejected, SN <= Max SN
M5 (PID: 1, SN: 5) - rejected, SN <= Max SN
M6 (PID: 1, SN: 6) - rejected, SN <= Max SN
M7 (PID: 1, SN: 7) - written to partition. For PID 1, Max SN=7
M8 (PID: 1, SN: 8) - written to partition. For PID 1, Max SN=8
M9 (PID: 1, SN: 9) - written to partition. For PID 1, Max SN=9
M10 (PID: 1, SN: 10) - written to partition. For PID 1, Max SN=10

When producer restarts, new PID gets assigned.

When to use idempotent=true

When data consistency is important: If acks=all then there is no reason not to enable this feature. It works flawlessly and without an additional complexity for the application developer.

If you currently use acks=0 or acks=1 for reasons of latency and throughput then you might consider staying away from this feature. If you already use acks=0 or acks=1 then you probably value the performance benefits over data consistency.

Important: Offset vs Timestamp order

- Kafka guarantees that all consumers see messages in the same **offset order** (per partition)
- There is no guarantee that messages are appended in timestamp order though

Processing order, position and time

Initial startup behaviour (no offsets are committed yet)

auto.offset.reset

- Possible values: earliest/latest
- Consumers: default is latest
- Kafka streams: default is earliest

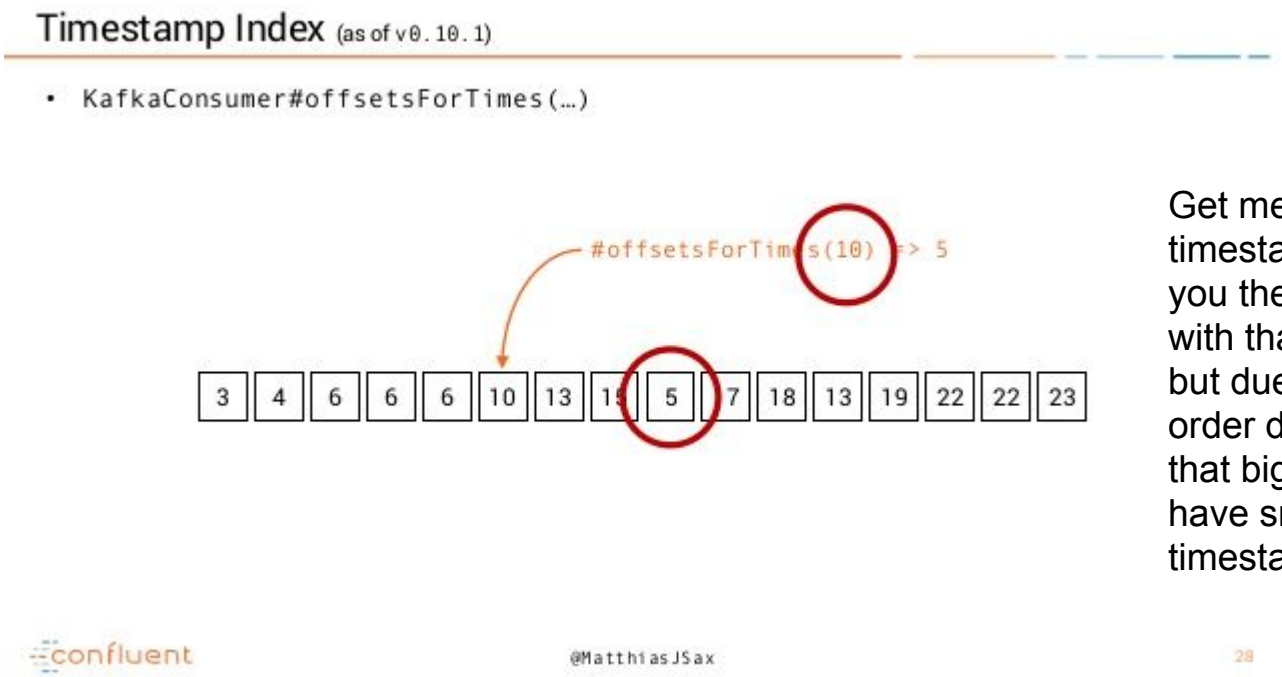
SeekToOffset (KafkaConsumer only)

- #seek, #seekToBeginning, #seekToEnd

SeekByTimestamp (KafkaConsumer only)

- #offsetForTimes + seek
- Uses message timestamp (but not in the payload)

Timestamp index but offsets not ordered by TS



Get messages by timestamp = gives you the lowest offset with that timestamp but due to out of order data, it may be that bigger offsets have smaller timestamps.

Source: https://www.confluent.io/kafka-summit-san-francisco-2019/whats-the-time-and-why