## Consumer Offsets and Commits

Any consumer instance in that consumer group should send its offset commits to the group's offset manager (i.e GroupCoordinator)

The offset manager sends a successful offset commit response to the consumer only after all the replicas of the offsets topic *__consumer_offsets* receive the offsets.

Auto Commit: The consumer can automatically commit offsets periodically

Consumer properties: `enable.auto.commit` , `auto.commit.interval.ms`

Commit SYNC: Manually commit the offsets and block until the offsets have been successfully committed

Commit ASYNC: Manually commit the offsets using non-blocking request and can trigger *OffsetCommitCallback* upon either successfully committed or fatally failed

## Kafka message delivery semantics

When *publishing* a message we have a notion of the message being "committed" to the log.

### At least once

Messages are never lost but may be redelivered (Kafka < 0.11.0.0 i.e Vanila kafka)
I.e Message may be written to the log again during retry
Eg: ?

### At most once (Commit / Abort)

Messages are never lost but may be redelivered (Kafka >= 0.11.0.0)
I.e Resending will not result in duplicate entries in the log
Eg: ?

### Exactly once / Idempotence

Each message is delivered once and only once (Kafka >= 0.11.0.0)
Eg: ?

An Idempotent producer retry of the same message will only be written to the Kafka log once.
Producer property: enable.idempotence

### How Exactly once is achieved

Producers can request the partition broker to generate a unique identifier for that producer. Now the producer can produce a message to that partition with this key along with a sequence number that increments for each message. The partition broker can now validate against this key pair for duplicates.

## Transactional Messaging

### Why Transactions?

1. Atomicity: A consumer's application should not be exposed to messages from uncommitted transactions.
2. Durability: The broker cannot lose any committed transactions.
3. Ordering: A transaction-aware consumer should see transactions in the original transaction-order within each partition.
4. Interleaving: Each partition should be able to accept messages from both transactional and non-transactional producers
5. There should be no ***duplicate*** messages within transactions.

### How transaction is achieved

Kafka uses Multi phase commit protocol for transaction management

#### Control Records

Simple solution is to maintain a log for all transaction events per transaction in a dedicated topic named __transaction_state.
These special log entries are called Control Records.

#### Transaction Coordinator

Idempotence guarantees exactly once delivery only within a session. I.e while writing a batch to a single topic partition on a broker.

Transaction API guarantees at-most-once semantics across multiple sessions i.e over multiple topics and also topic partition combinations.

For this semantics to work, Instead of propagating the Producer ID to partition broker for idempotence guarantee, We bypass these requests through a Transaction coordinator who

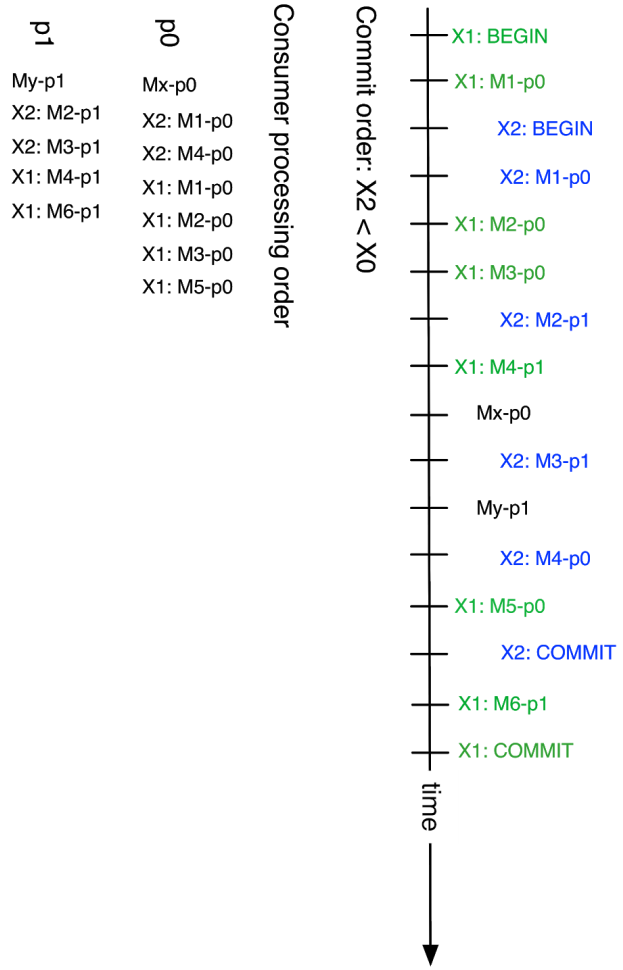
keeps the mapping of the combination key Transaction ID + Producer ID and only sends the Transaction ID  to all partition brokers participating in the transaction

Transaction ID is a User generated key and needs to be registered before producers start using.

### Transaction Groups

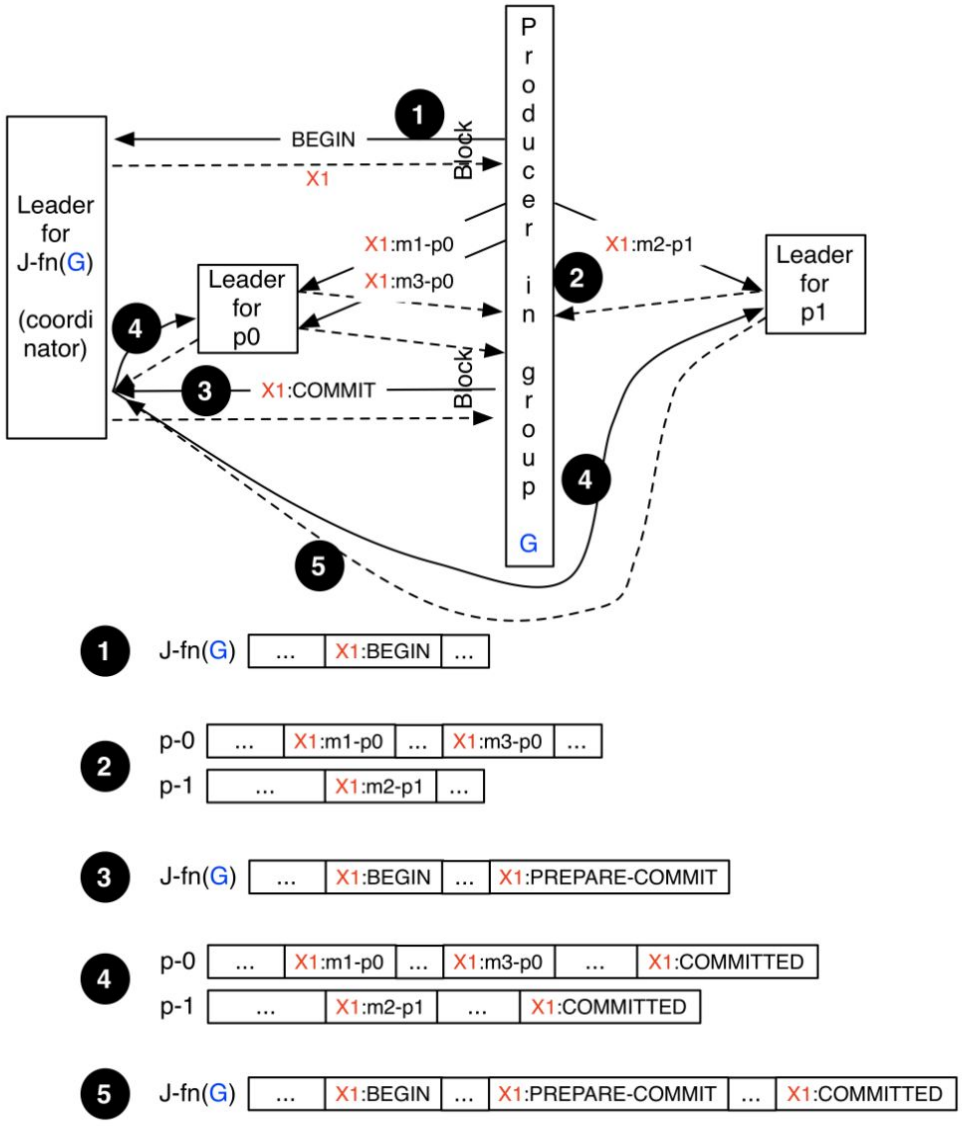
Group of transactional producers can be formed into groups and use the same Transaction ID. Only one producer ID at a time is allowed for any given transaction ID

## Example




Since X2 is committed first, each partition will expose messages from X2 before X1
Since the non-transactional messages arrived before the commits for X1 and X2, those messages will be exposed before messages from either transaction.

## Implementation




InitPhase (Step 1)

Producer sends BeginTransaction request to transaction Coordinator

Append Control Record: BEGIN

SendPhase (Step 2)

Send transaction payloads

EndPhase (Steps 3, 4, 5)

CommitTransaction

Append Control Record: PREPARE_COMMIT

If the Transaction Coordinator hears back from all brokers involved in the transaction.

Append Control Record: COMMITTED

Else

Append Control Record: PREPARE_ABORT

### Transaction aware consumer

Transactional producers can help consumers become transaction aware by writing transactional control messages(control records) to __consumer_offsets topics as well.

We can configure the consumer to read only committed messages as part of the transaction by setting *isolation level property* to “read_committed”.

## Exercise

### Other ConsumerAPI functions

Subscribe & Poll: to a list of topic using auto assignment of partitions (group management)

*Manual Partition Assignment*
Assign & Poll: Manually assign topic partitions

*Manual Offset Control*
commitSync: Commit the processed offsets synchronously

*Automatic Offset Committing*
commitAsync: Commit the processed offsets asynchronously

*Controlling The Consumer's Position*
Seek: Consume from given offset
SeekToBeginning: Consume from beginning offset of the topic partitions given (OffsetResetStrategy.*EARLIEST or —from-beginning flag in console consumer*)
SeekToEnd: Consume from last offset of the topic partitions given (OffsetResetStrategy.*LATEST*)

*Consumption Flow Control*
Pause: Pause consumption from topic partitions given
Resume: Resume consumption from topic partitions given

Read the below links:

Explore common.message folder in client jar for all request responses formats

https://cwiki.apache.org/confluence/display/KAFKA/Transactional+Messaging+in+Kafka
https://cwiki.apache.org/confluence/display/KAFKA/Idempotent+Producer
https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Client-side+Assignment+Proposal
https://cwiki.apache.org/confluence/display/KAFKA/Committing+and+fetching+consumer+offsets+in+Kafka

# Hands-On

## Set up the below project

https://github.com/swathi-kurella/kafka-workshop-series.git
Prerequisites:
Language: Java
Build: Gradle

## Create a sample topic

bin/kafka-topics.sh --create --topic kafka-workshop-eg --partitions 3 --bootstrap-server localhost:9092 --replication-factor 1

## Run the sample Consumer

Start ConsumerOffsets class

## Start console producer to produce messages

bin/kafka-console-producer.sh --topic kafka-workshop-eg --broker-list localhost:9092

## Observe console consumer console to view messages