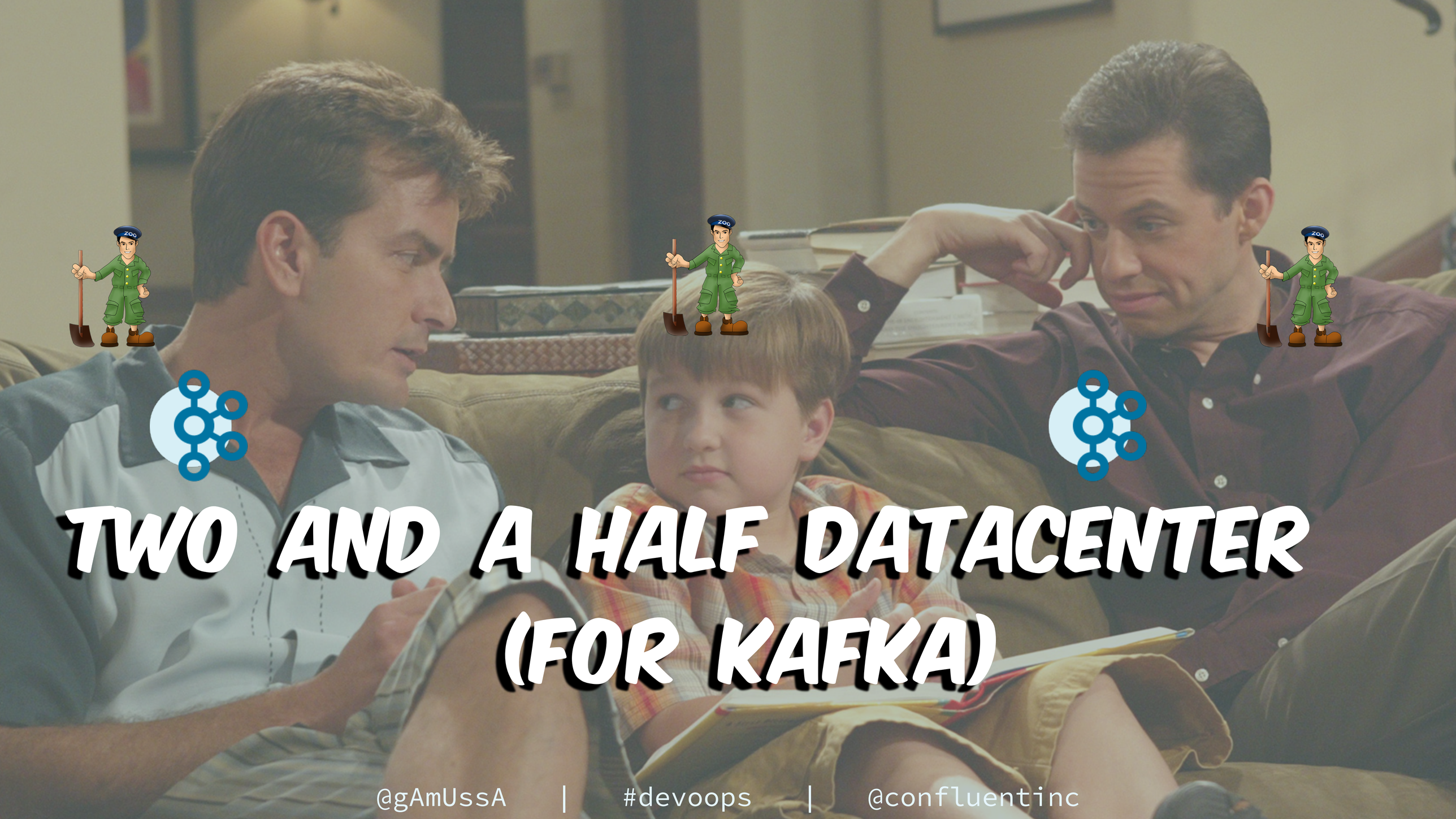
Your projector is working...

screenshare

Your ~~projector~~ is working...


TWO AND A HALF DATACENTER
(FOR KAFKA)
@gAmUssA | #devoops | @confluentinc

@gAmUssA | #devoops | @confluentinc

SPECIAL THANKS

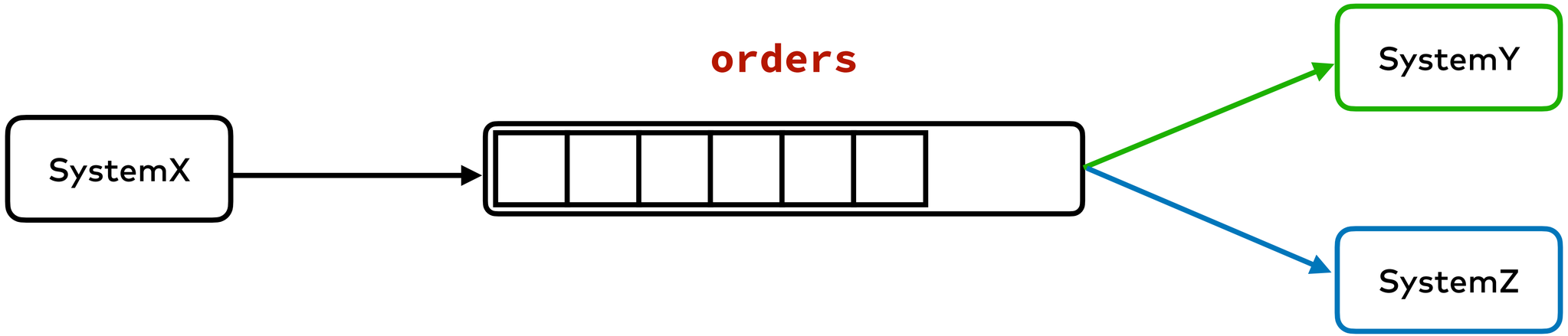
@jakekorab

@gAmUssA | #devoops | @confluentinc

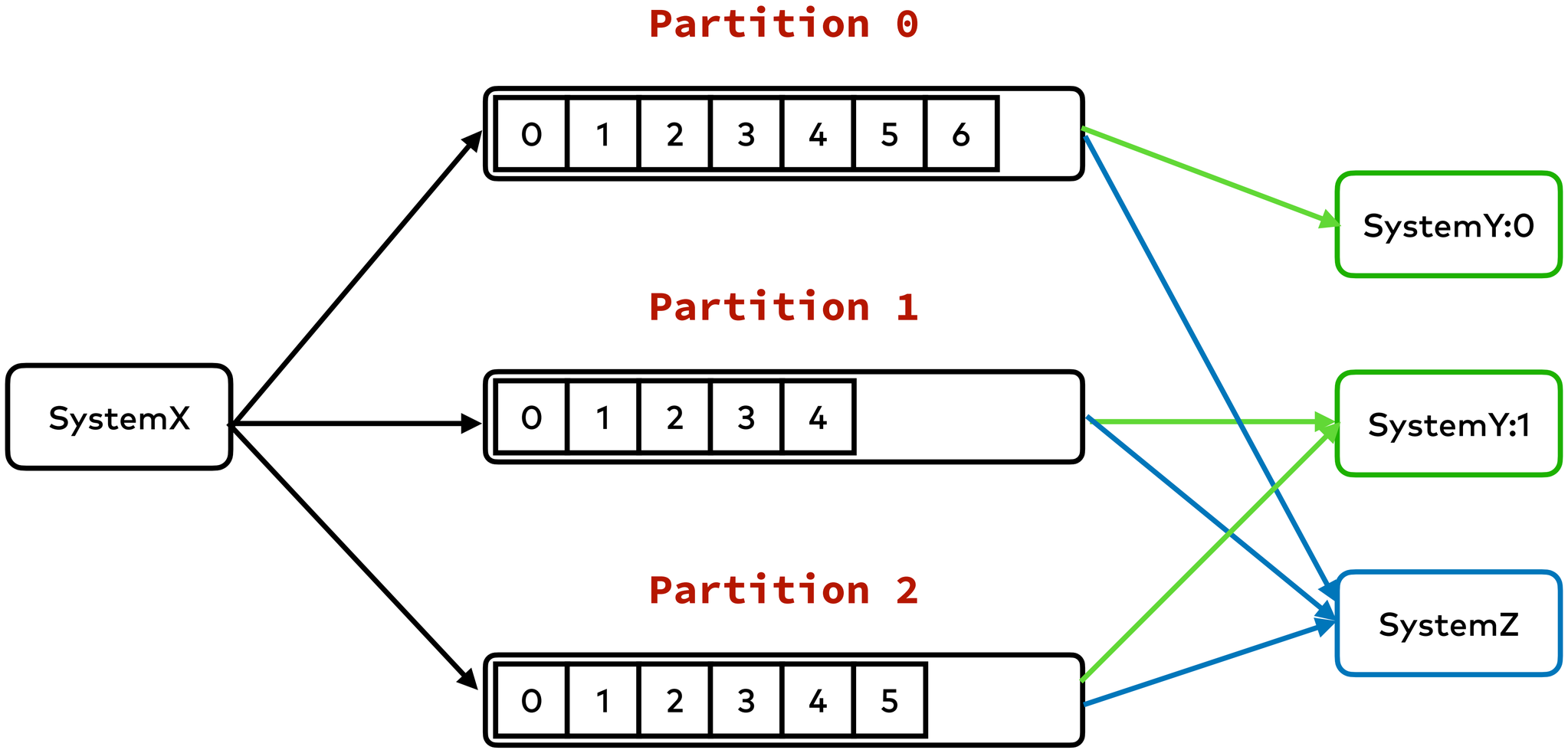
CONFLUENT

WHAT ARE WE TRYING TO DO HERE?

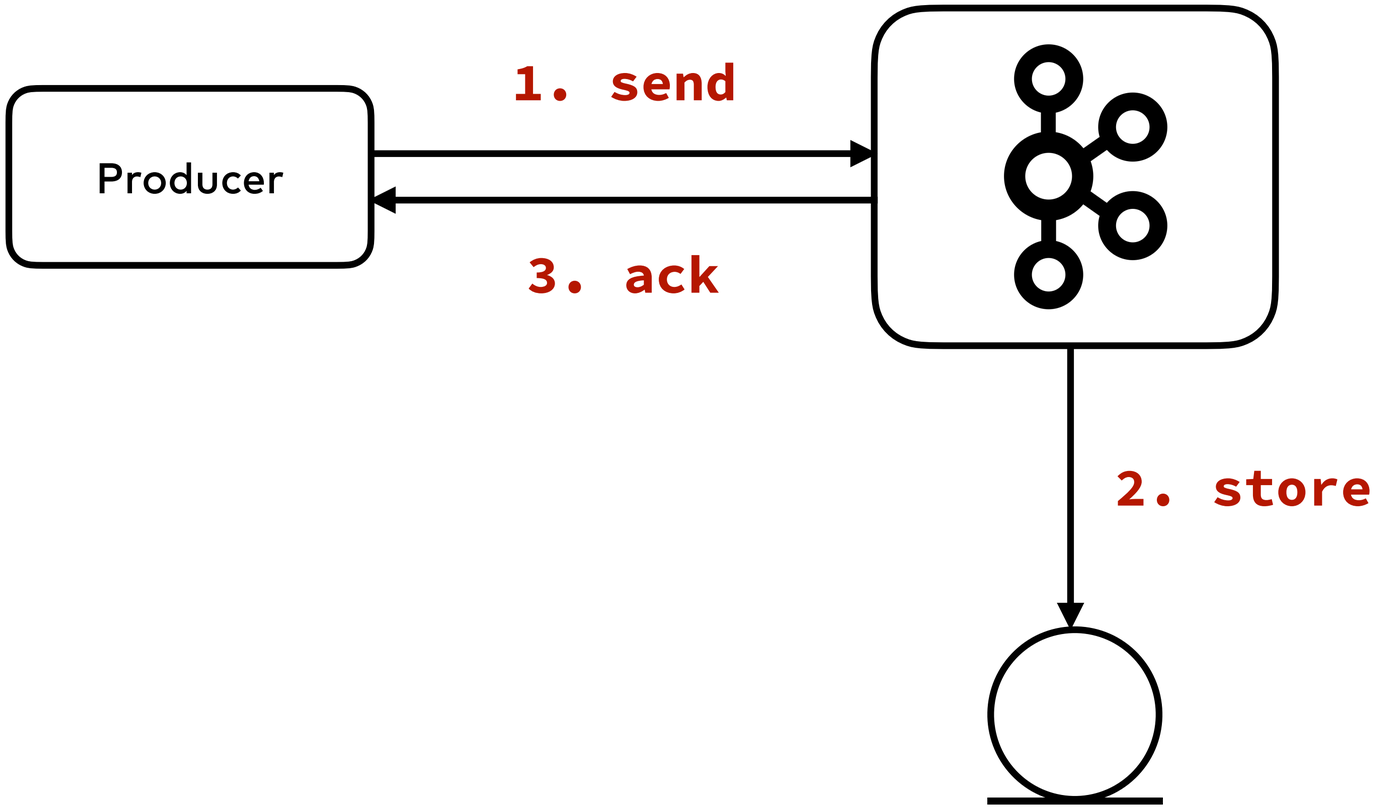
THE HIGH-LEVEL VIEW



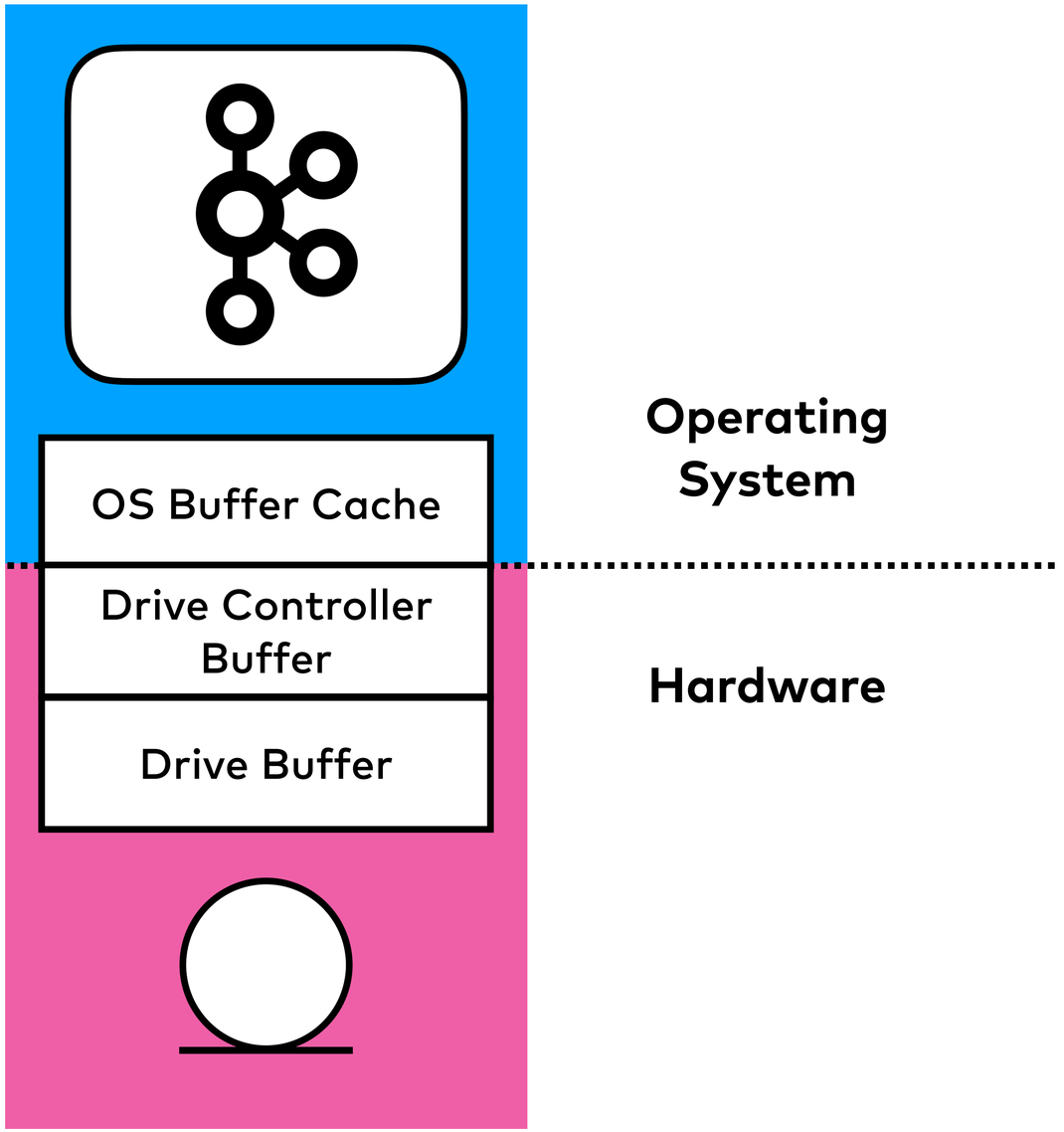
PARTITIONS AND CONSUMERS



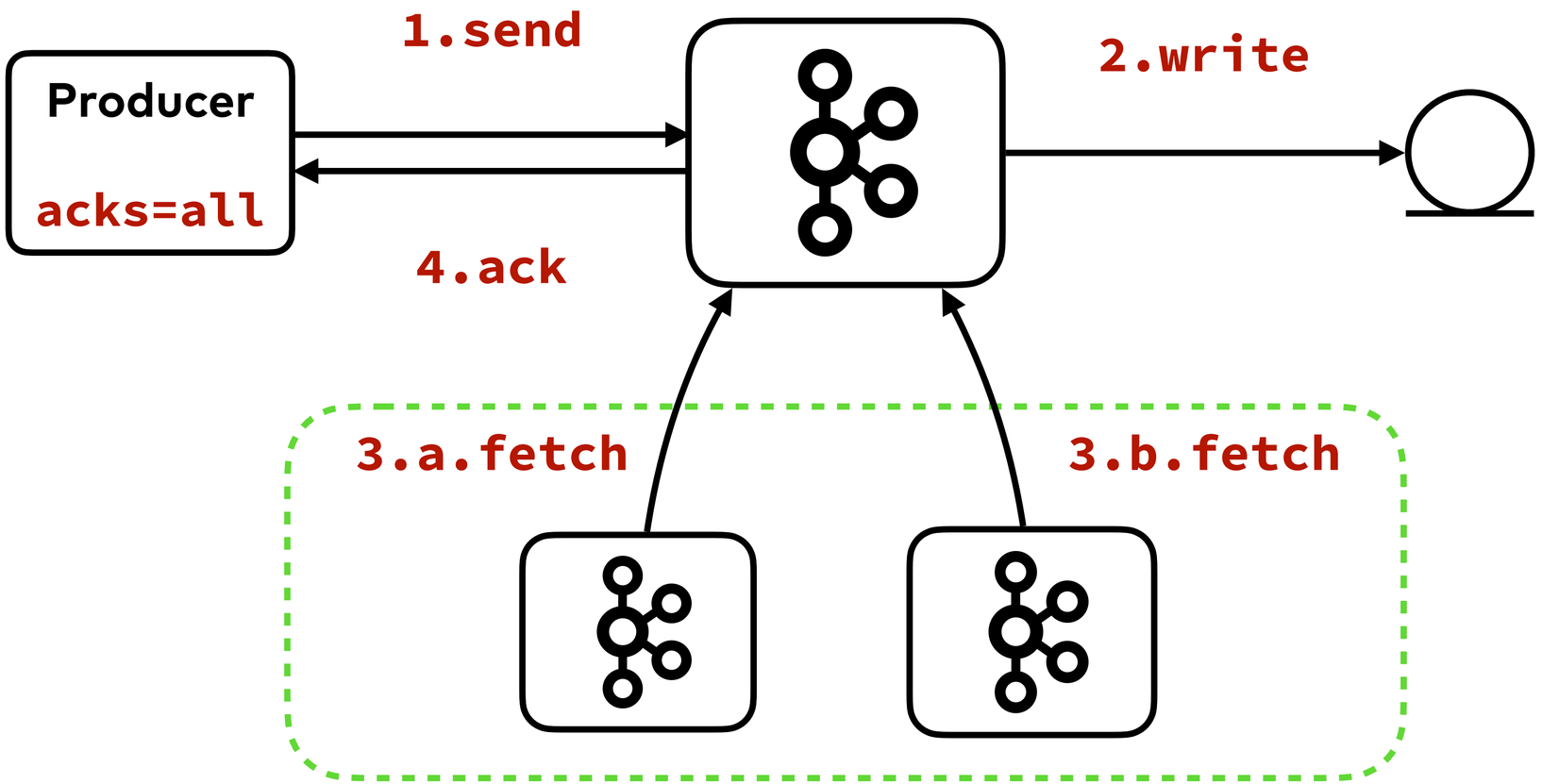
RELIABLE SENDS



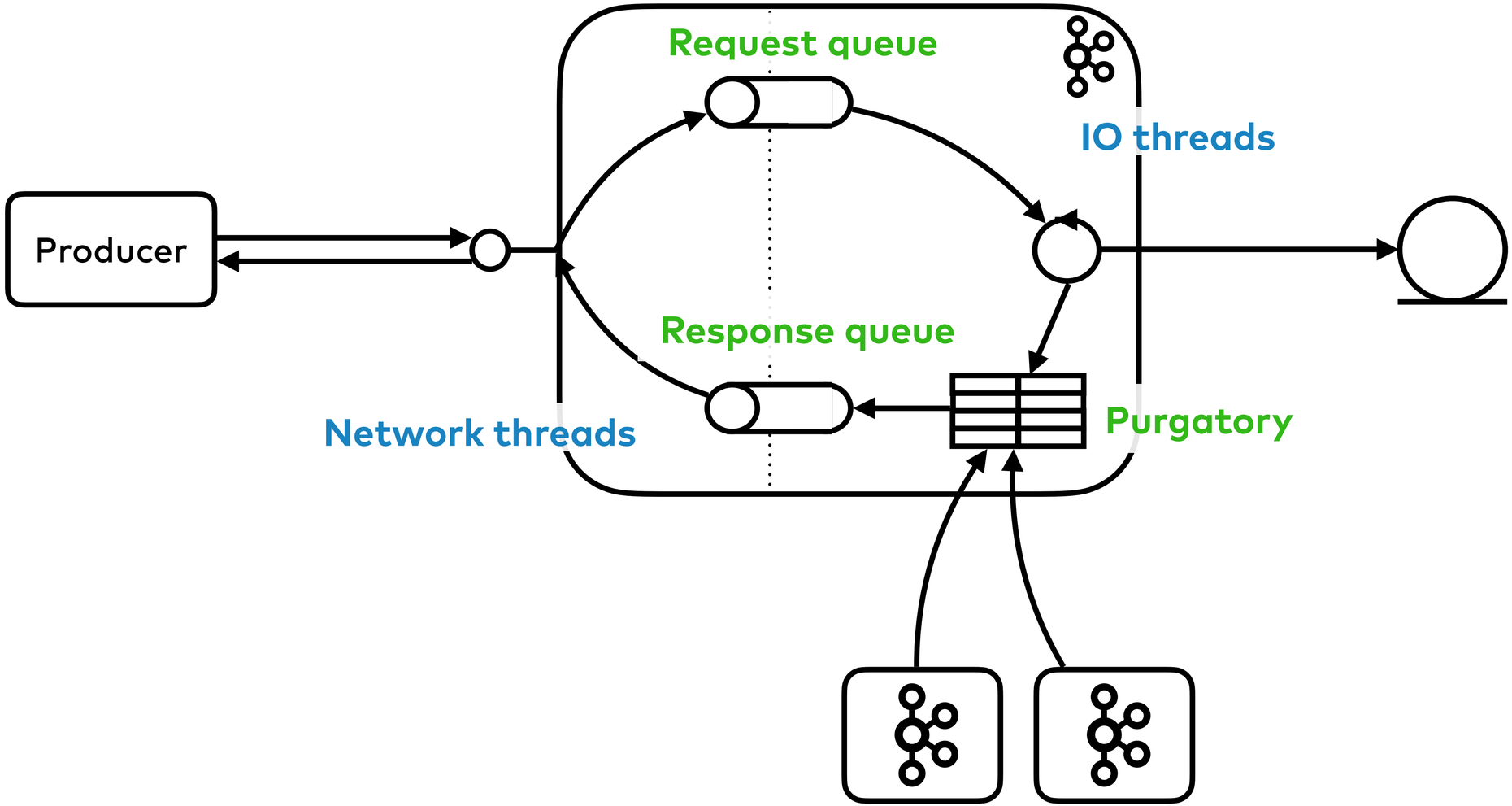
KAFKA, MEET STORAGE



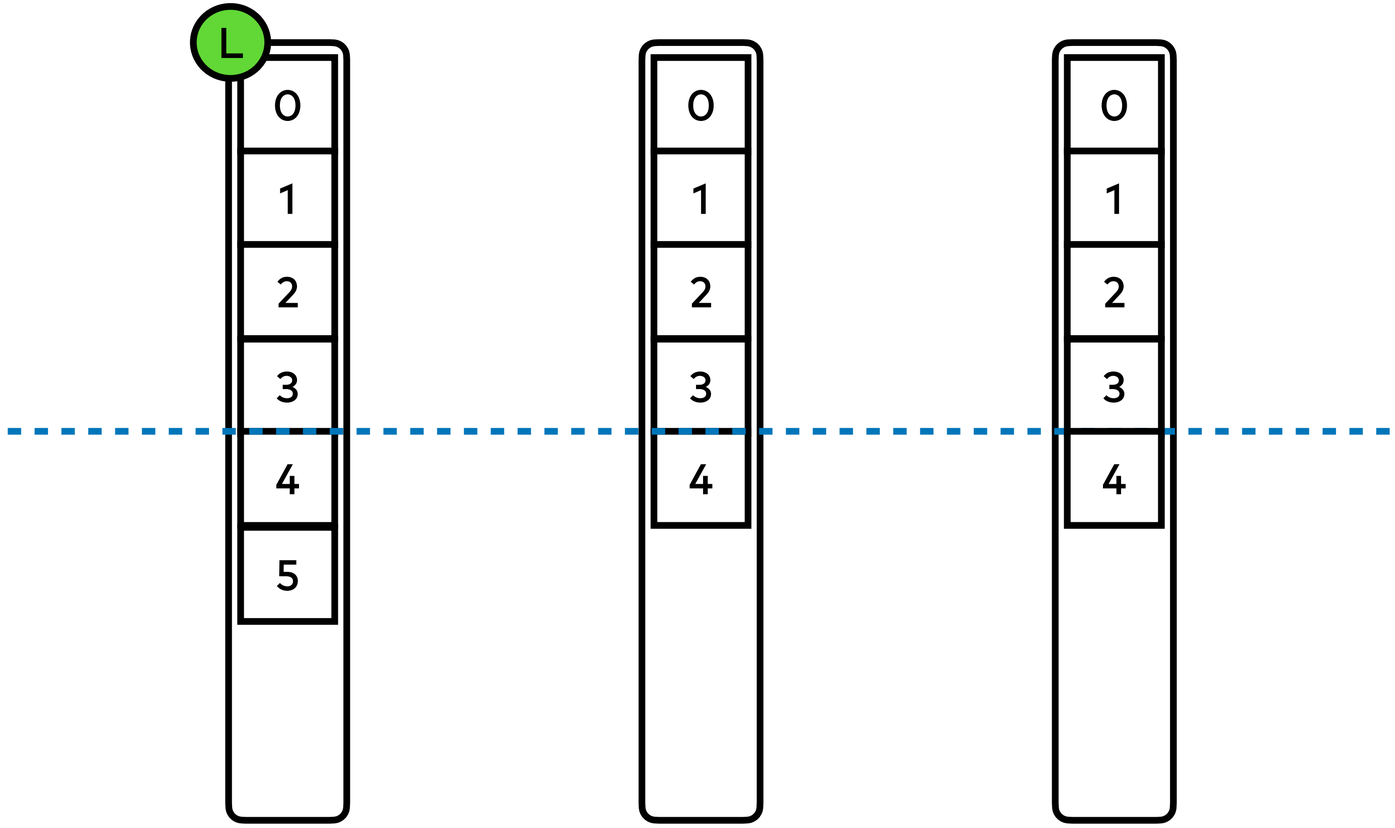
RELIABILITY THROUGH REPLICATION



MECHANICS OF SENDING

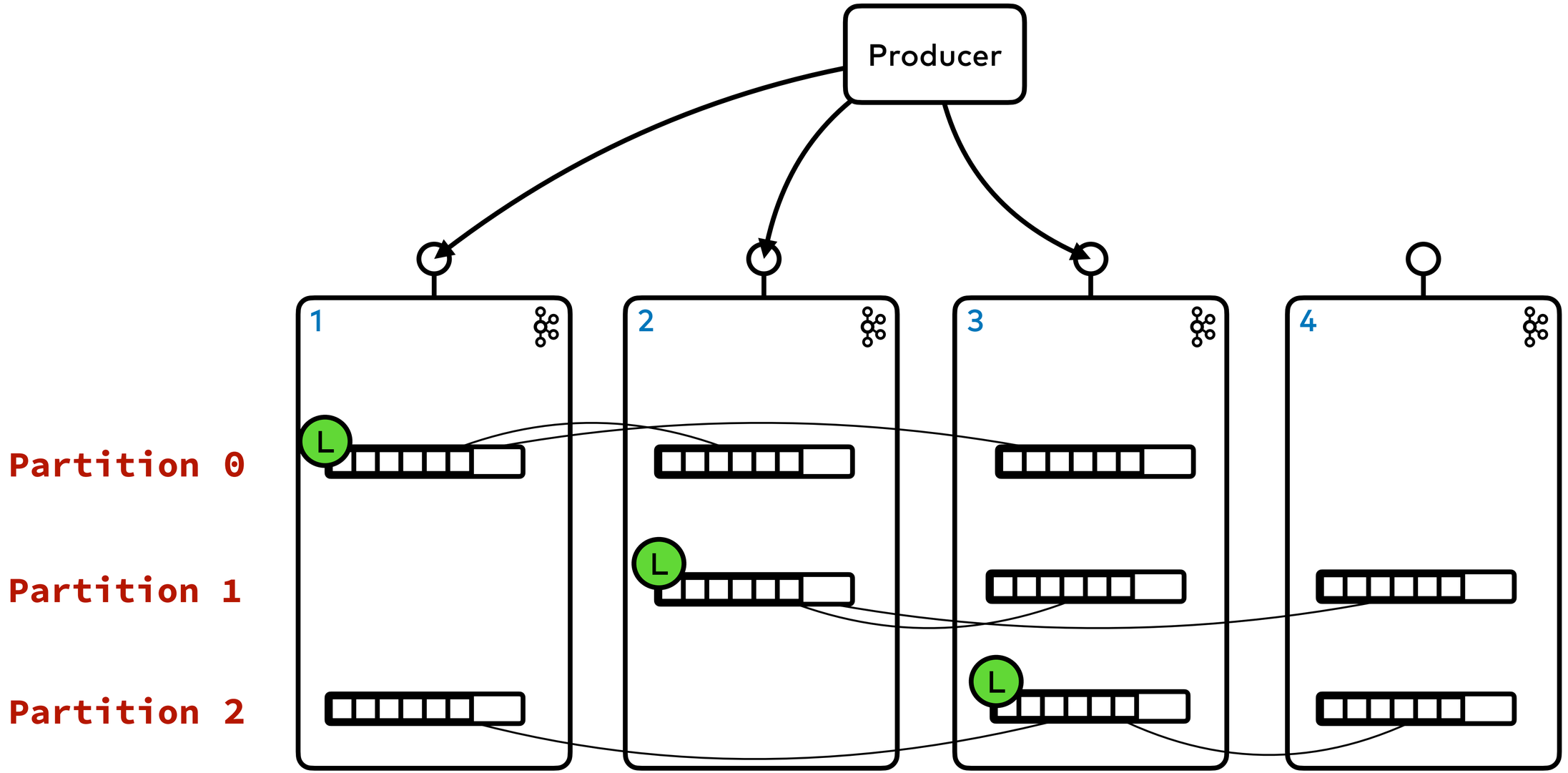


HIGH WATER MARK

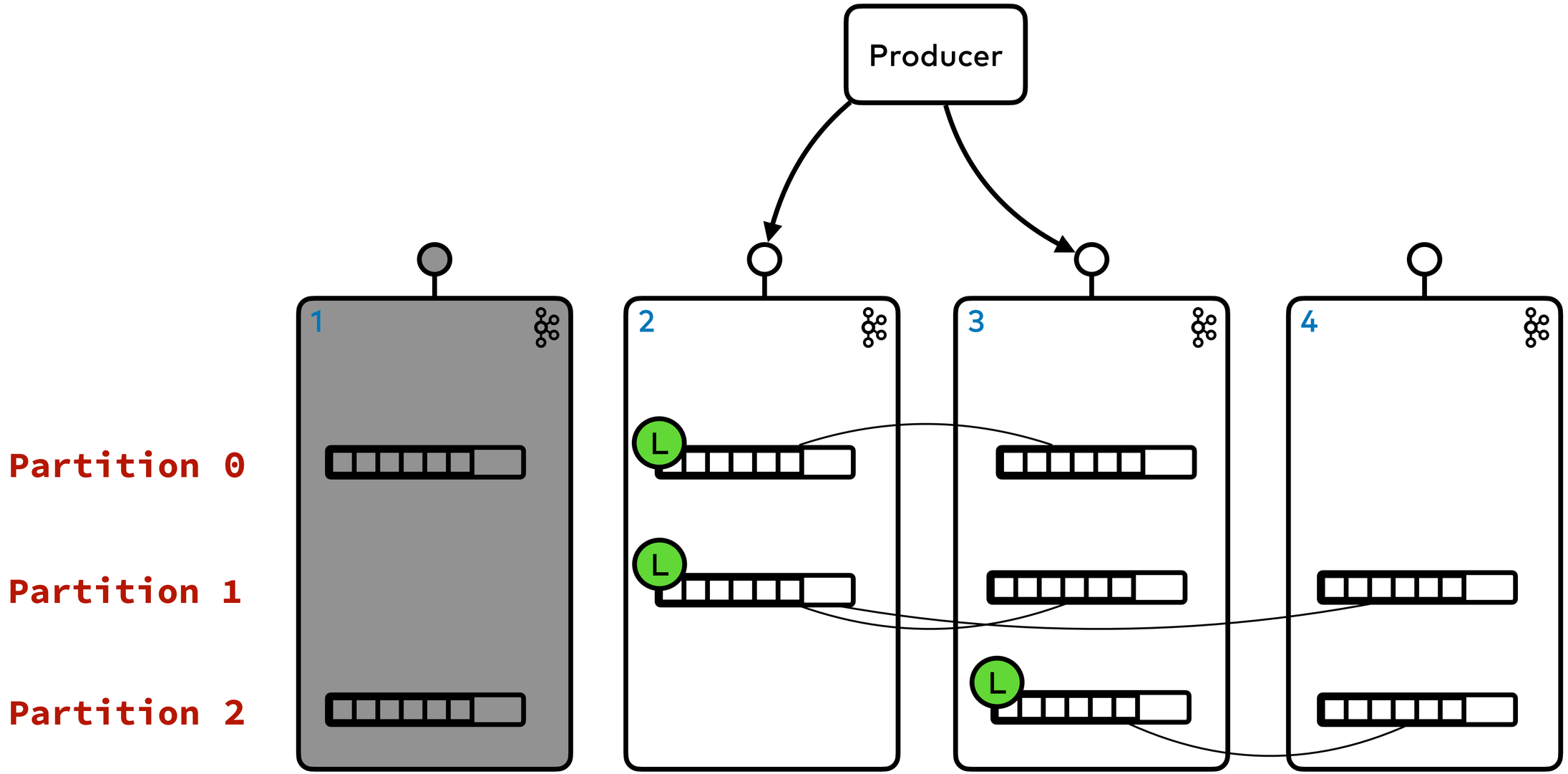


Replica 1 **Replica 2** **Replica 3**

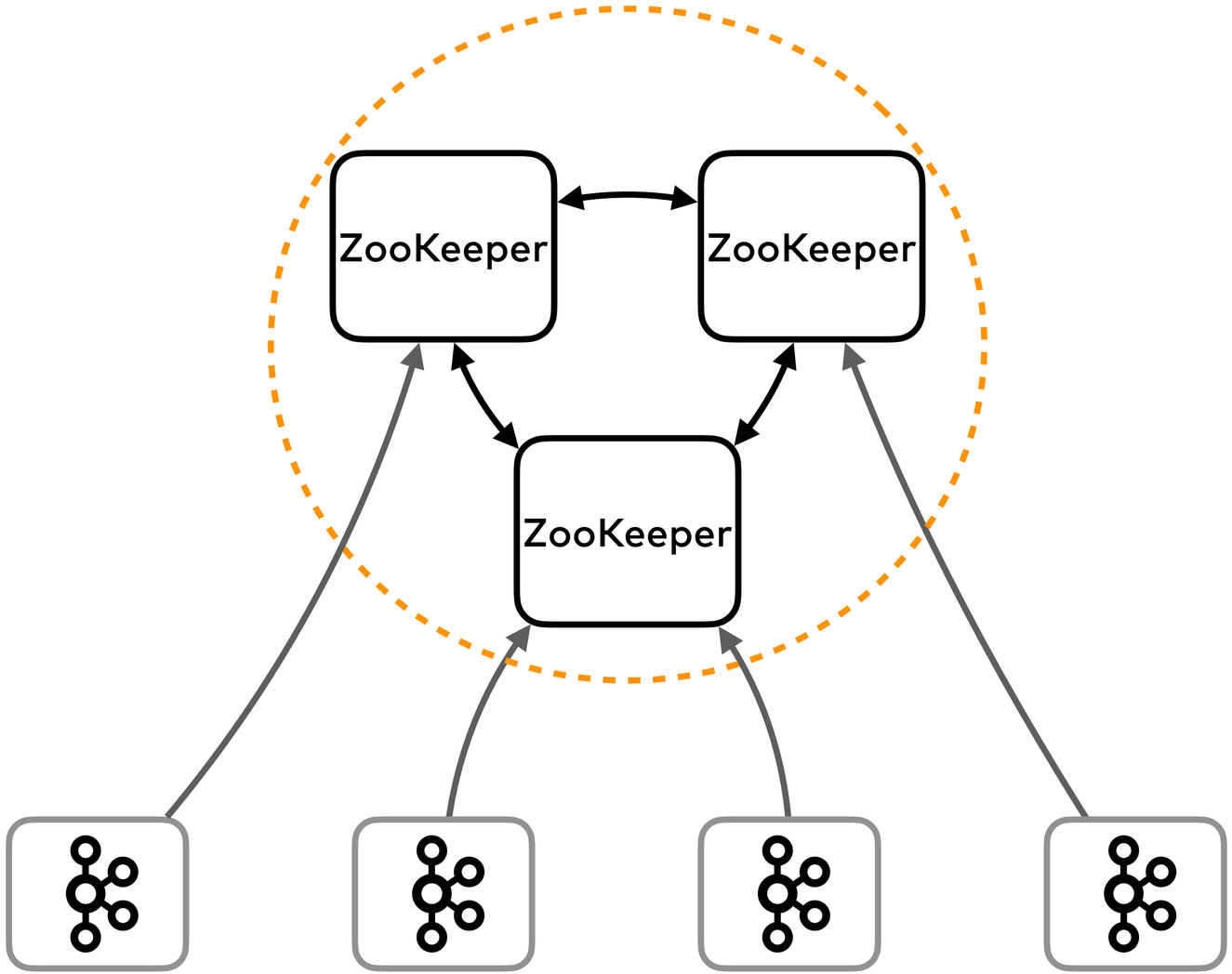
PARTITION DISTRIBUTION



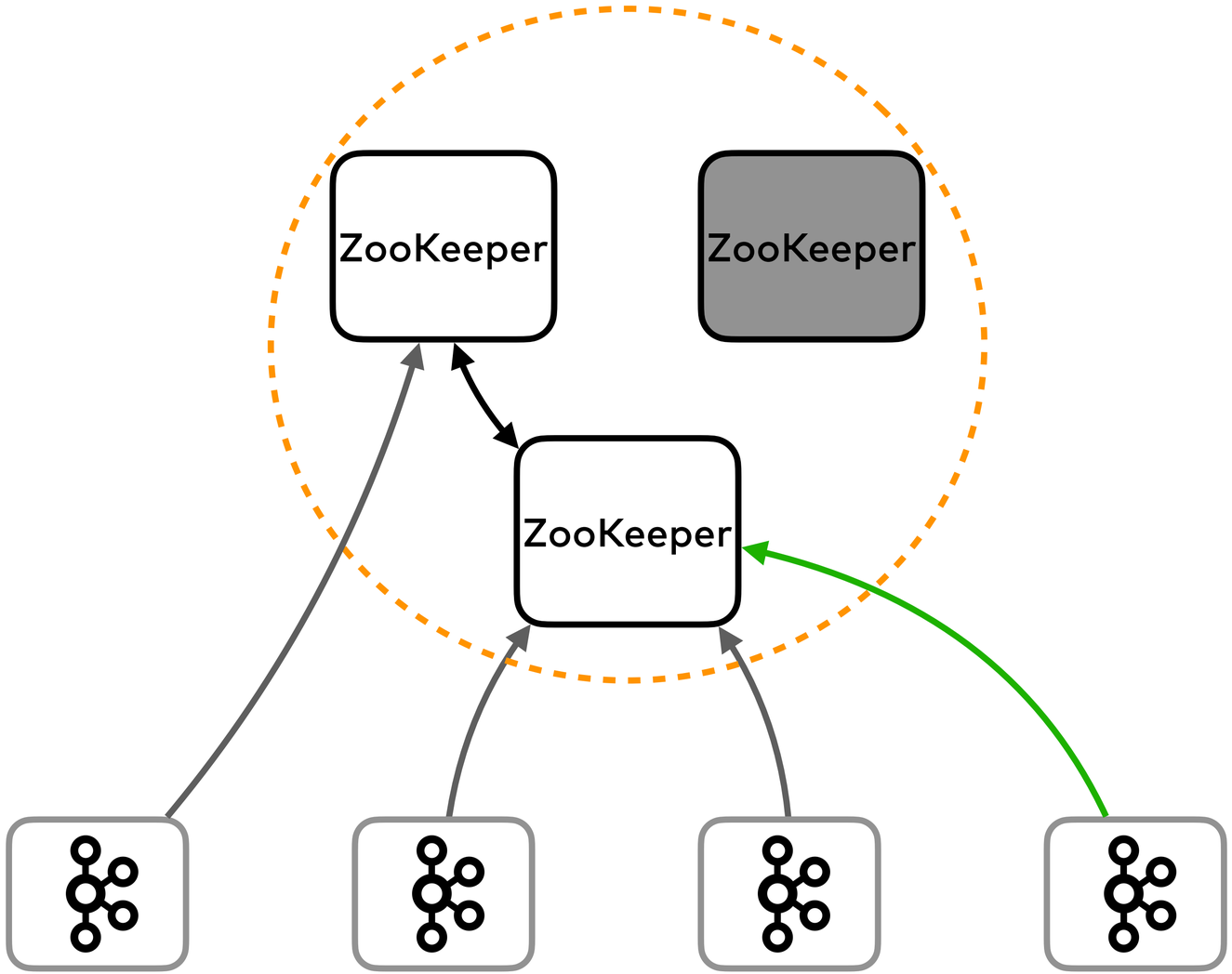
KAFKA NODE FAILURE



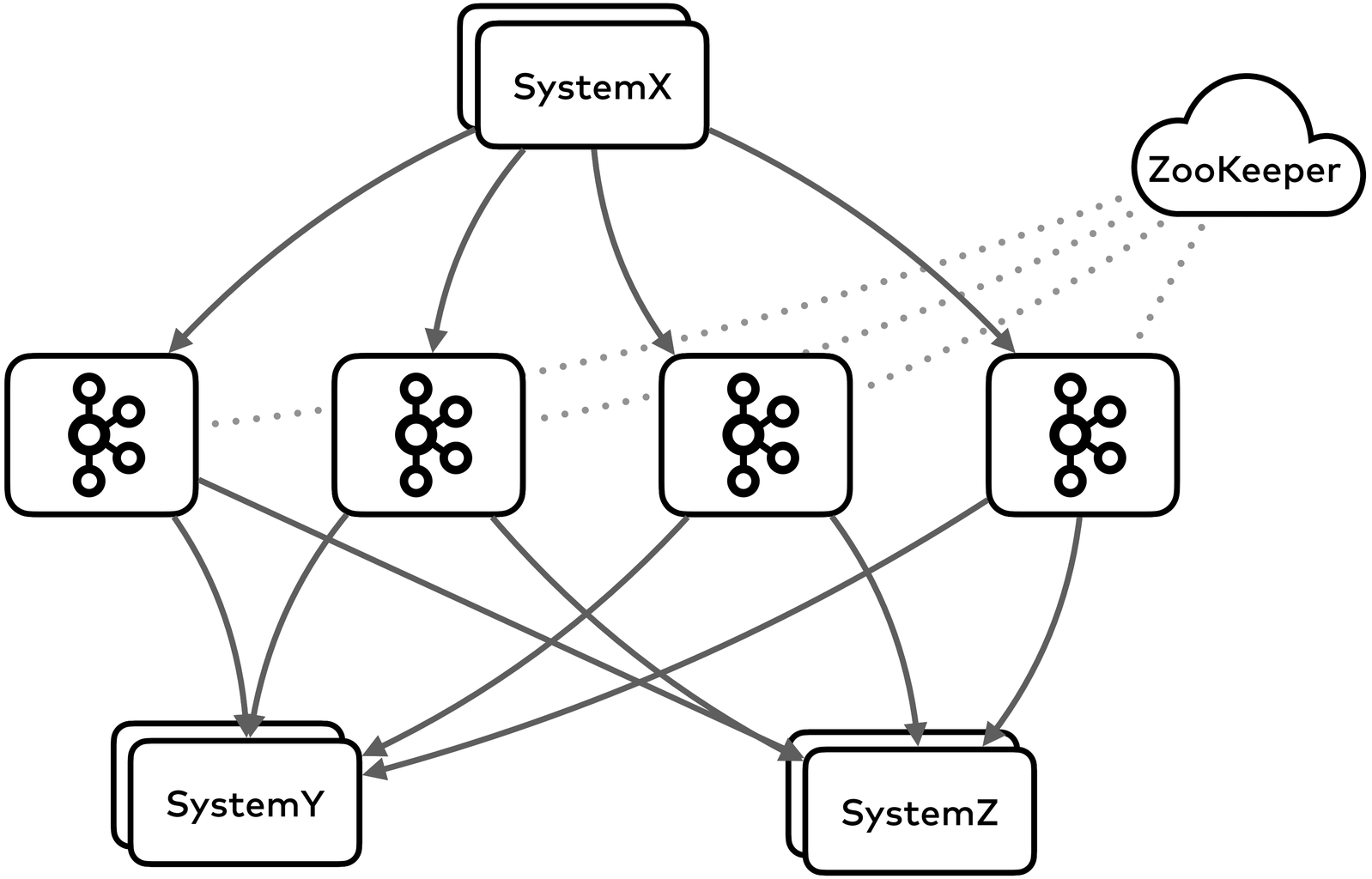
ZOOKEEPER ENSEMBLES



ZOOKEEPER NODE FAILURE



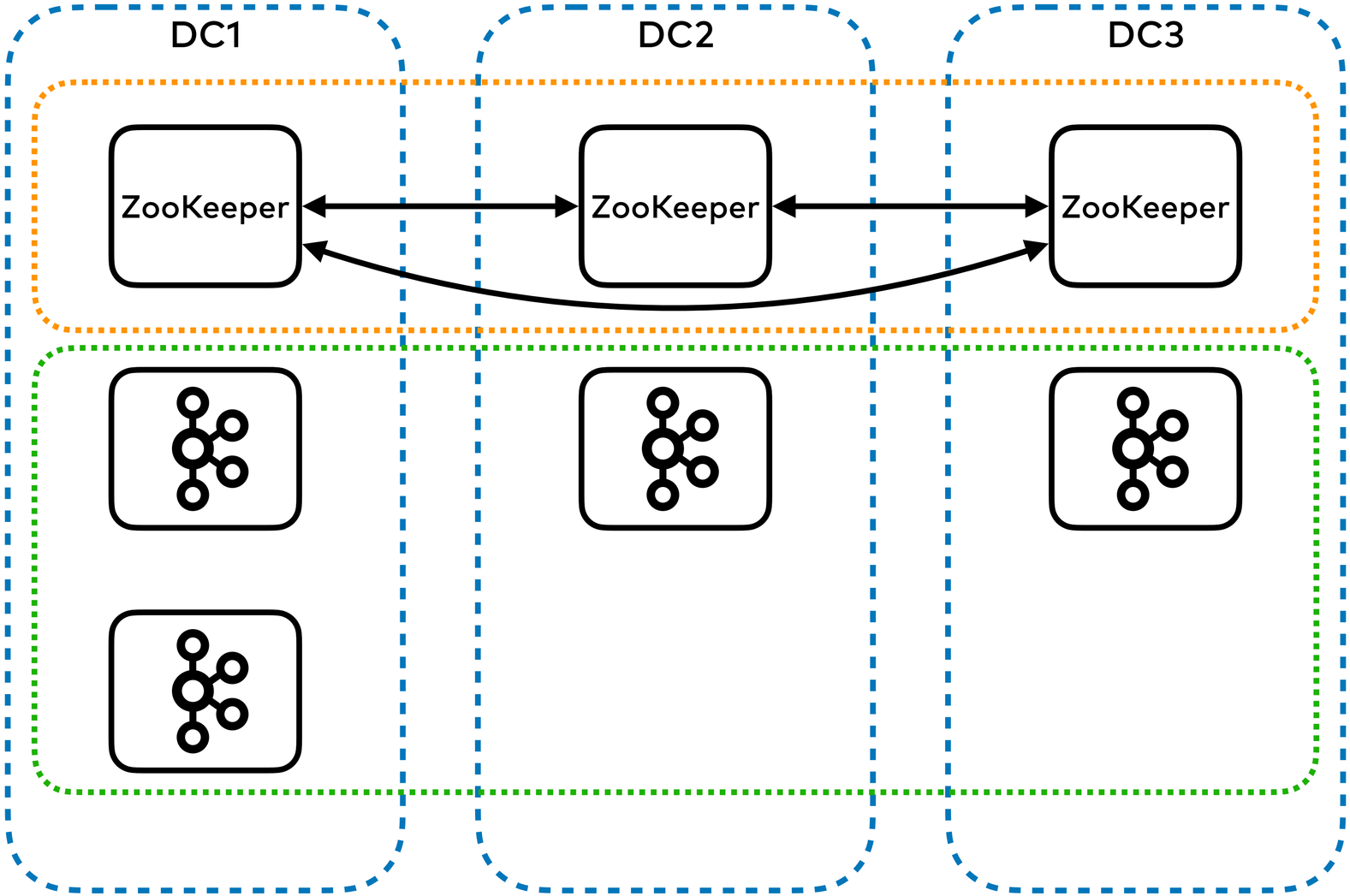
RUNTIME VIEW



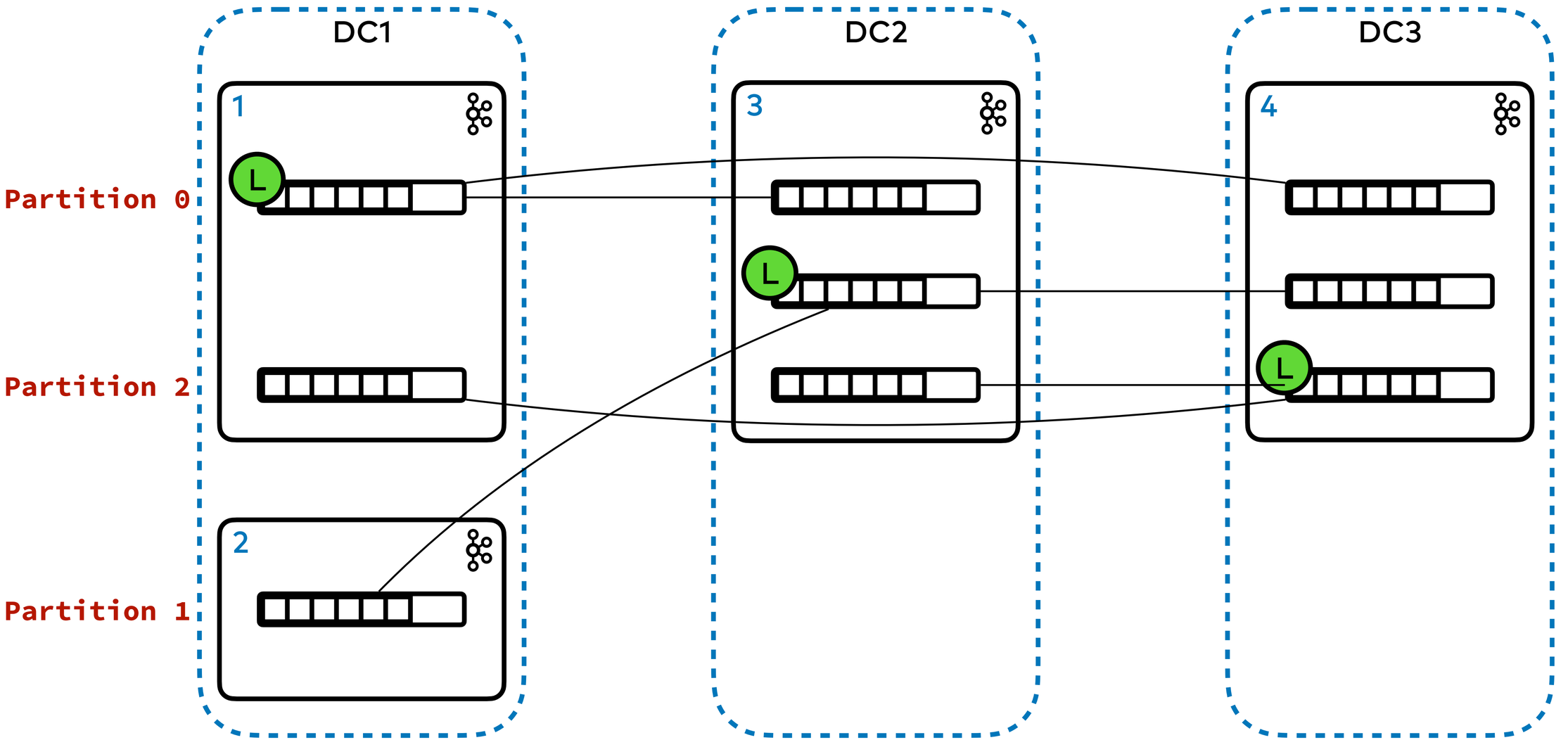
Replication


THREE DATA CENTERS

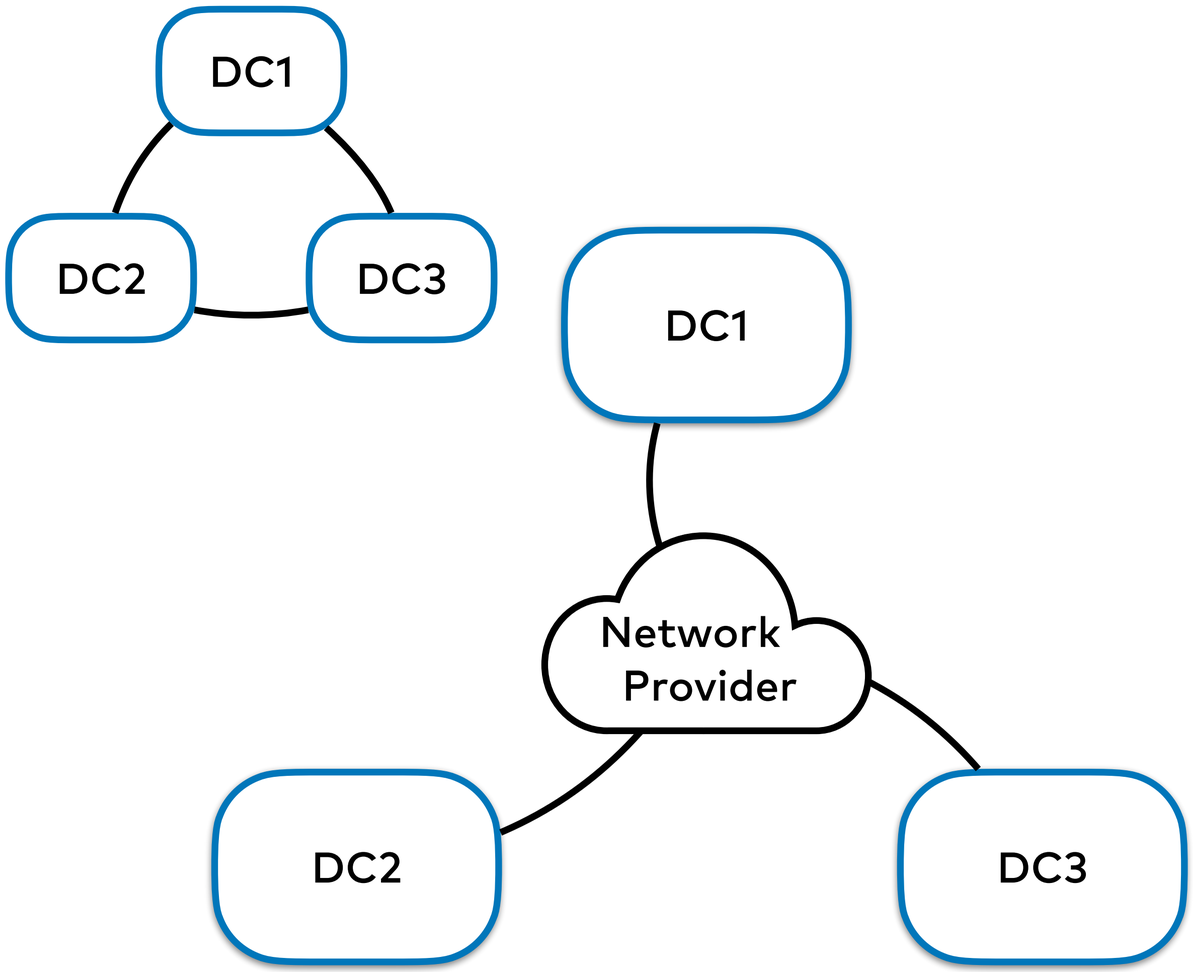
3 DATA CENTRES



RACK AWARE REPLICA ASSIGNMENT

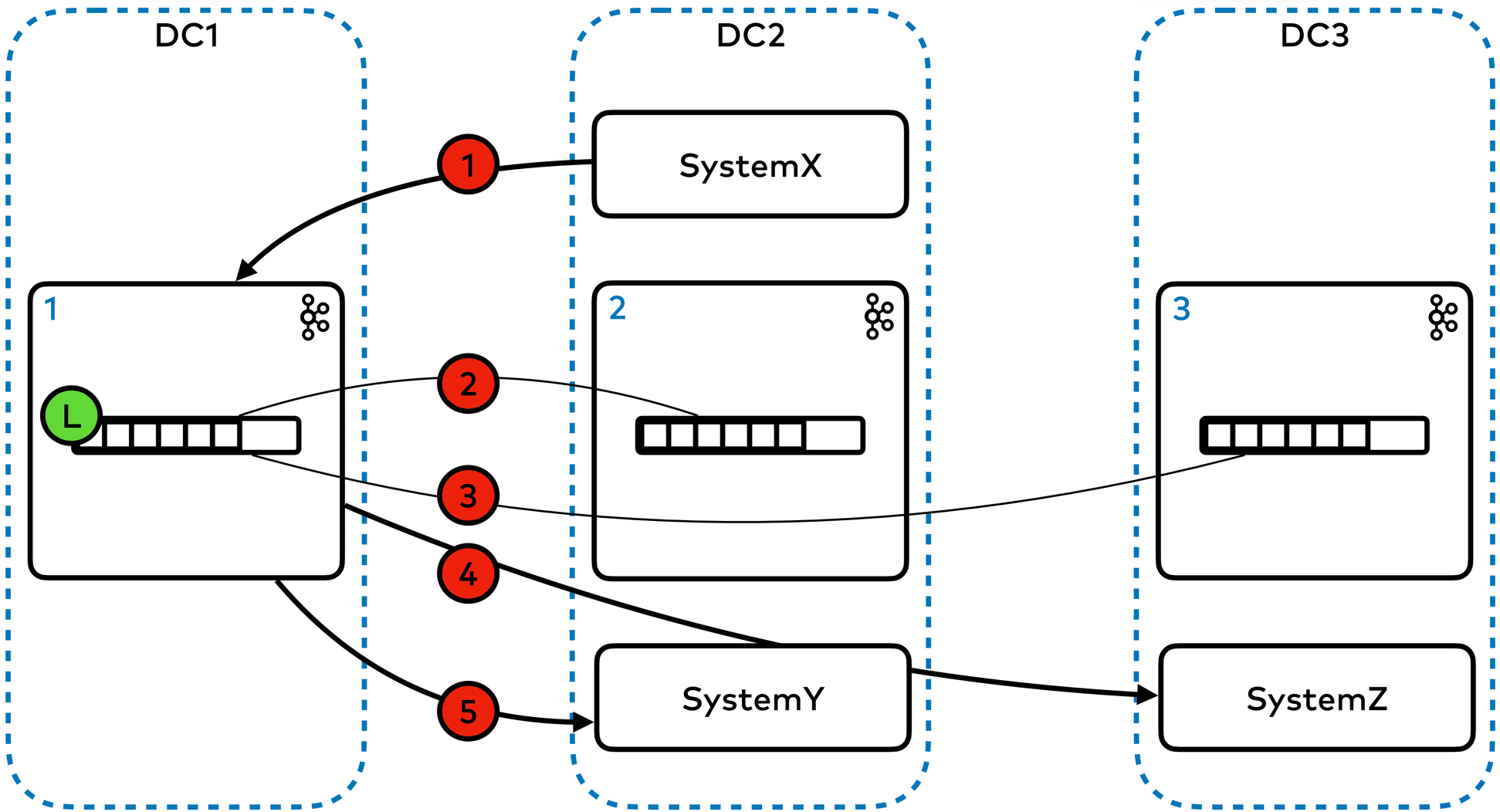


NETWORK CONSIDERATIONS



- What are the costs associated with data transfer?
- What is your latency? ~30ms OK
- Shared infrastructure that could cause contention?
- Single point of failure?

GitHub jkorab/kafka-cloud-calculator



KIP-392: ALLOW CONSUMERS TO FETCH FROM CLOSEST REPLICA

Client Config

`rack.id=<location>`

Broker Config

`replica.selector.class=`
`<ReplicaSelector impl>`

Out of the box:

- **LeaderSelector (default)**
- **RackAwareReplicaSelector**

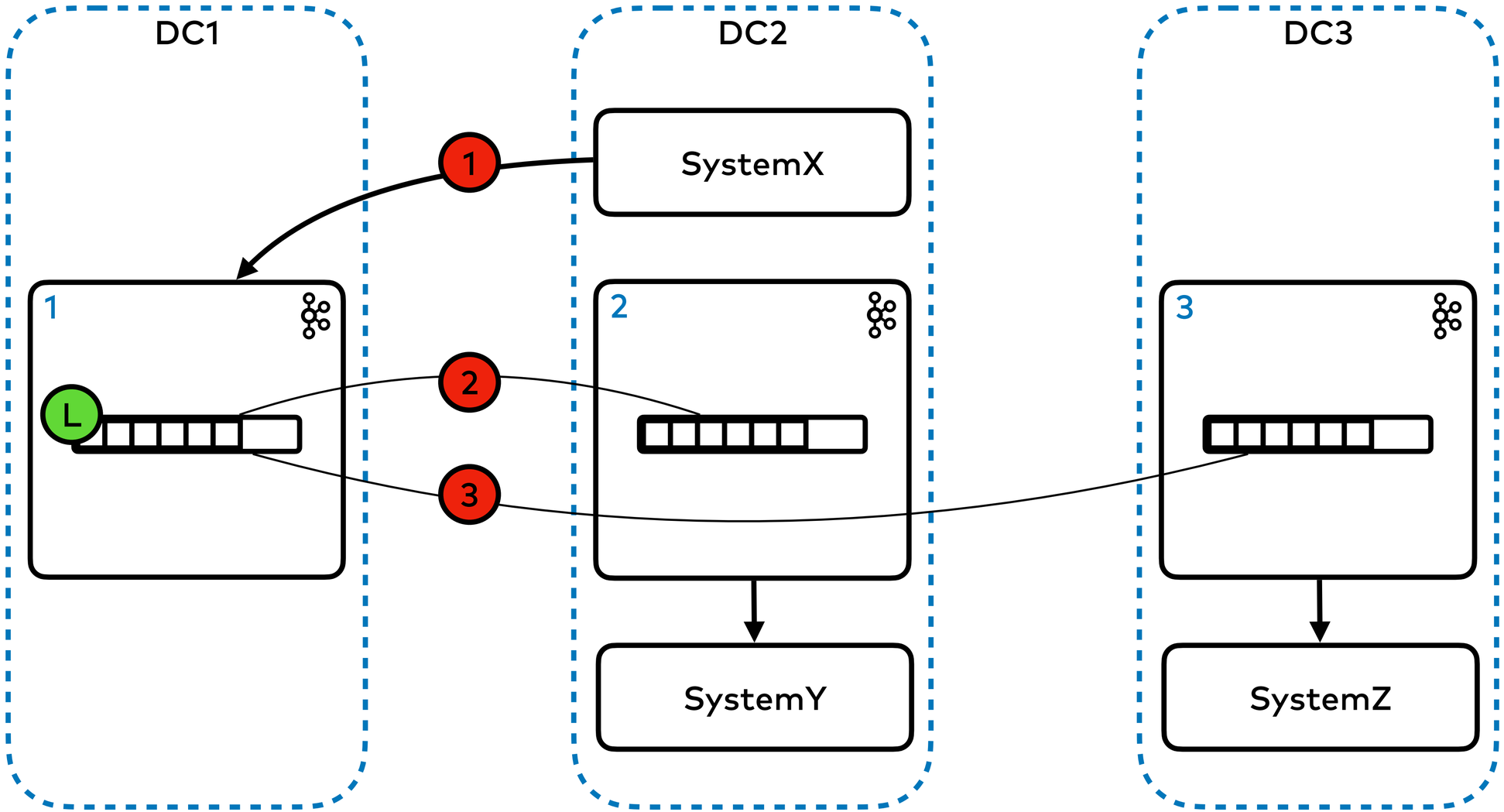
KIP-392: Allow consumers to fetch from closest replica

```
class ClientMetadata {

  final String rackId;
  final String clientId;
  final InetAddress address;
  final KafkaPrincipal principal;
}

interface ReplicaSelector extends Configurable, Closeable {
  /**
   * Select the preferred replica a client should use for fetching.
   * If no replica is available, this method should return null.
   */
  Node select(ClientMetadata metadata, PartitionInfo partitionInfo);
}
```

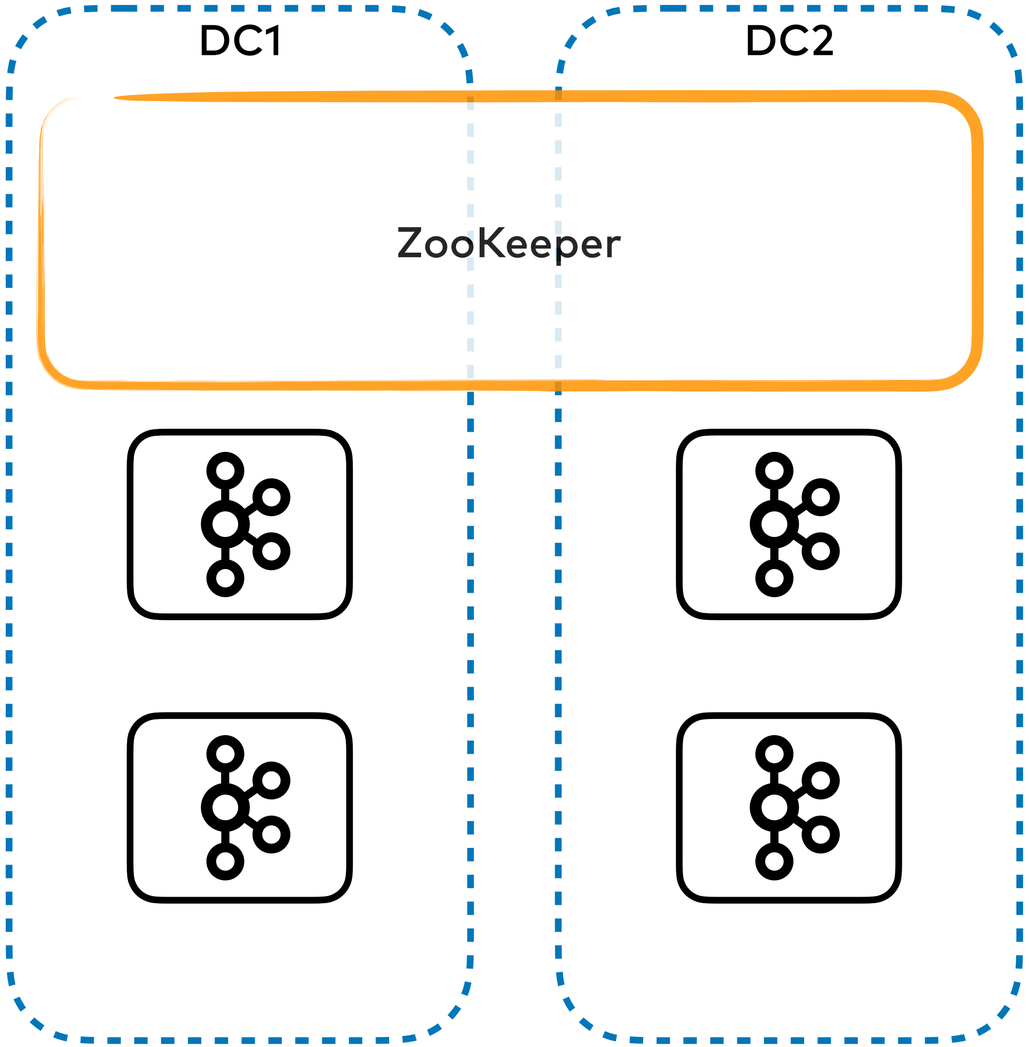
replica.selector.class=RackAwareReplicaSelector



rack.id=DC2 rack.id=DC3


TWO DATA CENTERS

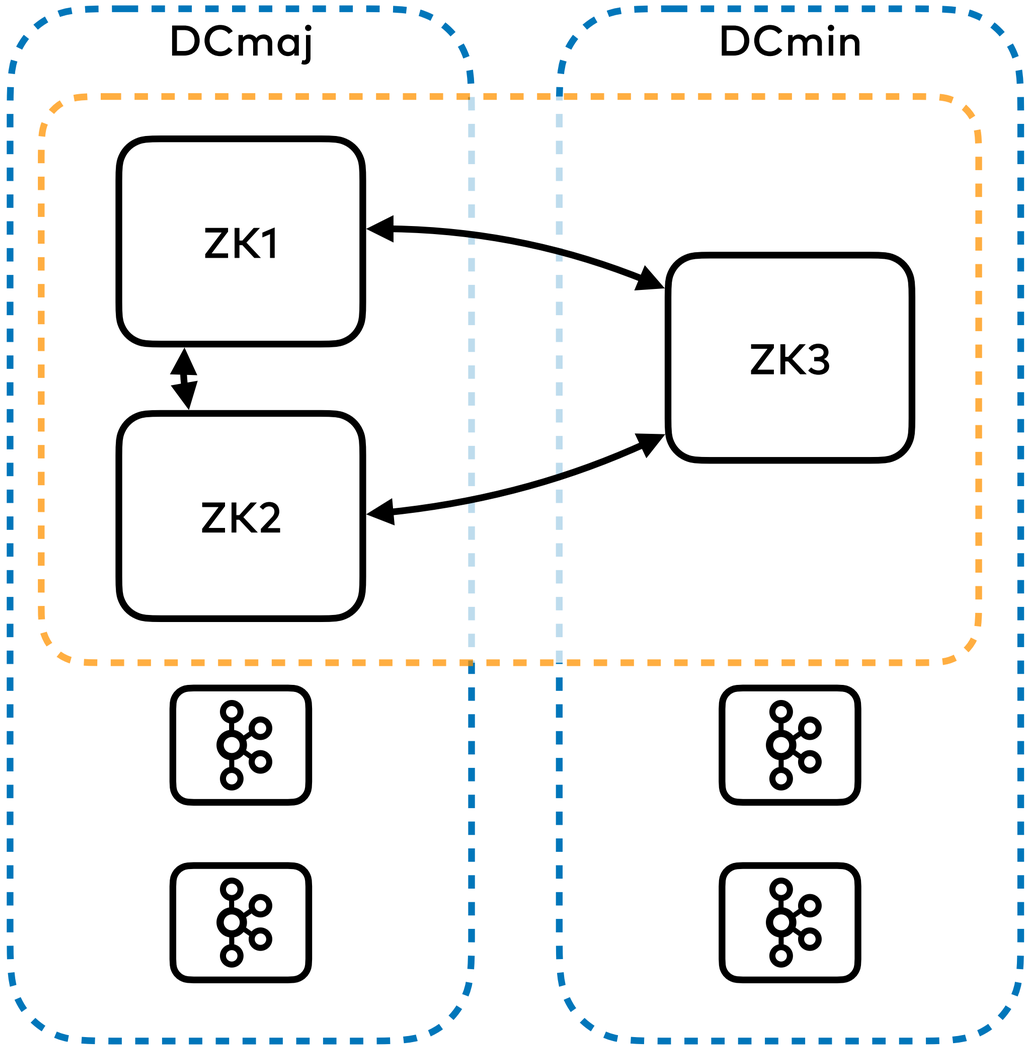
2 DATA CENTRES



Problems:

1. ZooKeeper
2. Topic Replication

2DC - NAIVE ZOOKEEPER SETUP

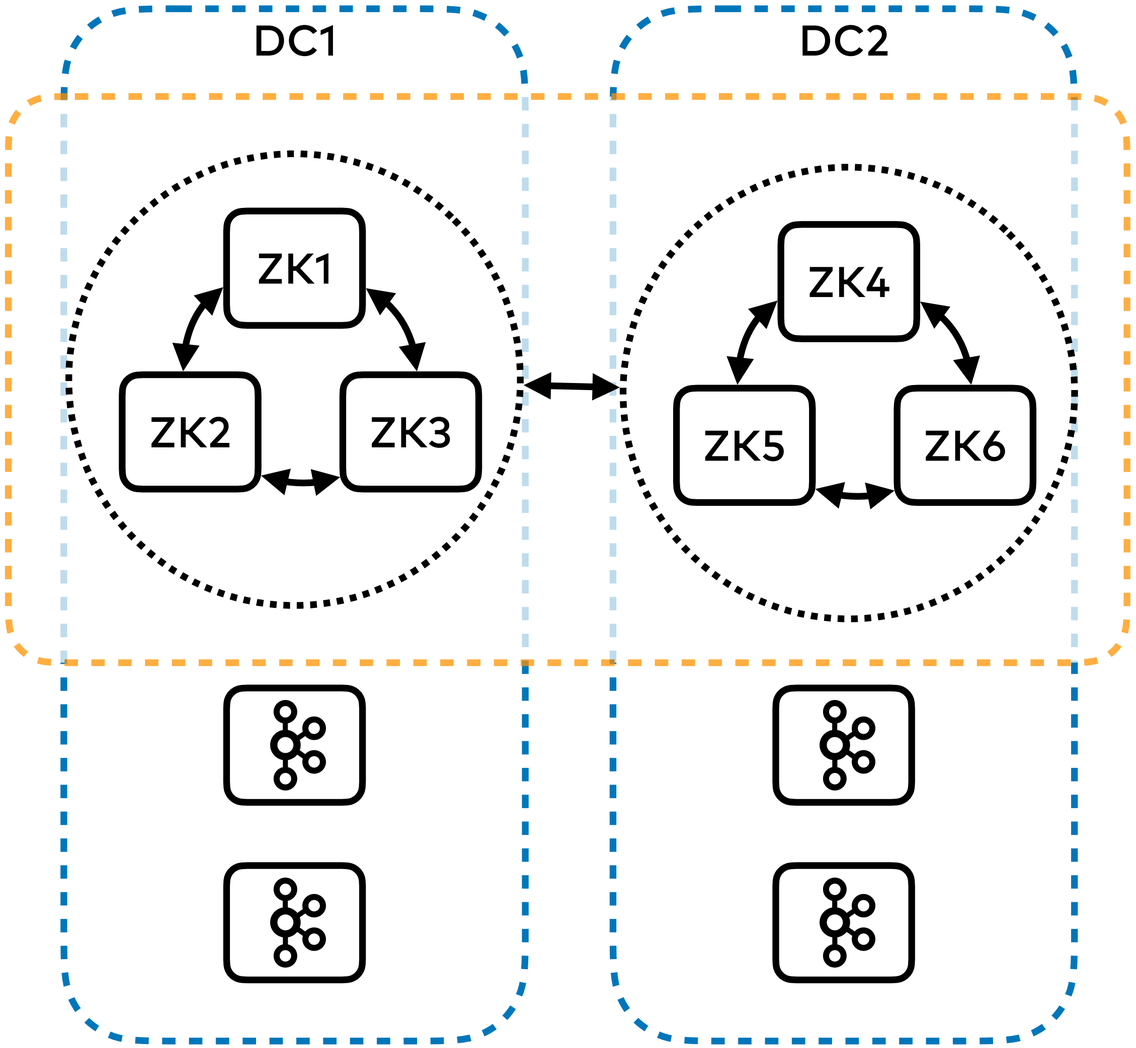


DC Failure Scenarios:

1. **DCmin** goes down - all OK
2. **DCmaj** goes down - ZK3 in quorum minority, shuts down. Outage.

Do not do this

2DC - HIERARCHICAL QUORUMS IN ZOOKEEPER

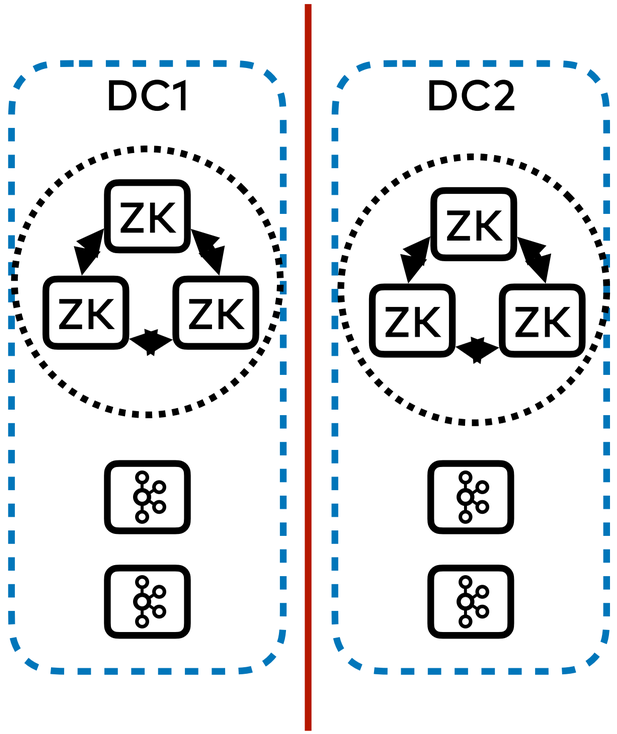


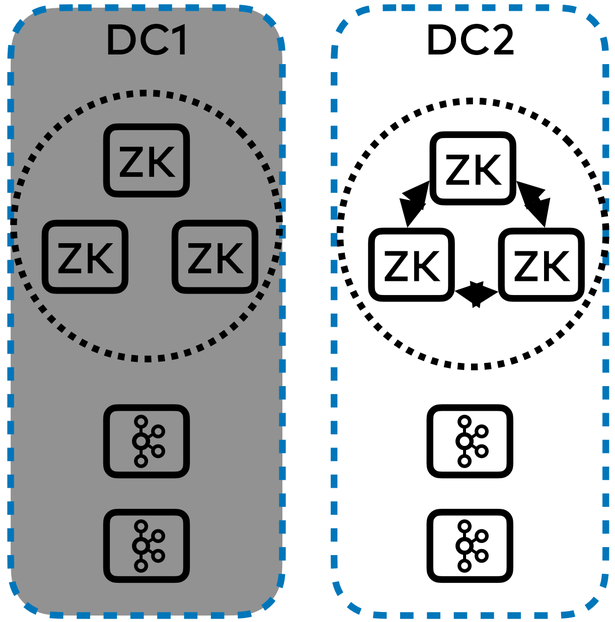
Brokers configured to talk to local ZKs.

Tolerates outage of one ZooKeeper per local cluster.

Trades off Availability for Consistency.

2DC - HIERARCHICAL QUORUM SETUP





Communication outage looks just like a DC outage.

- Clients lose visibility of partition leaders in other DC

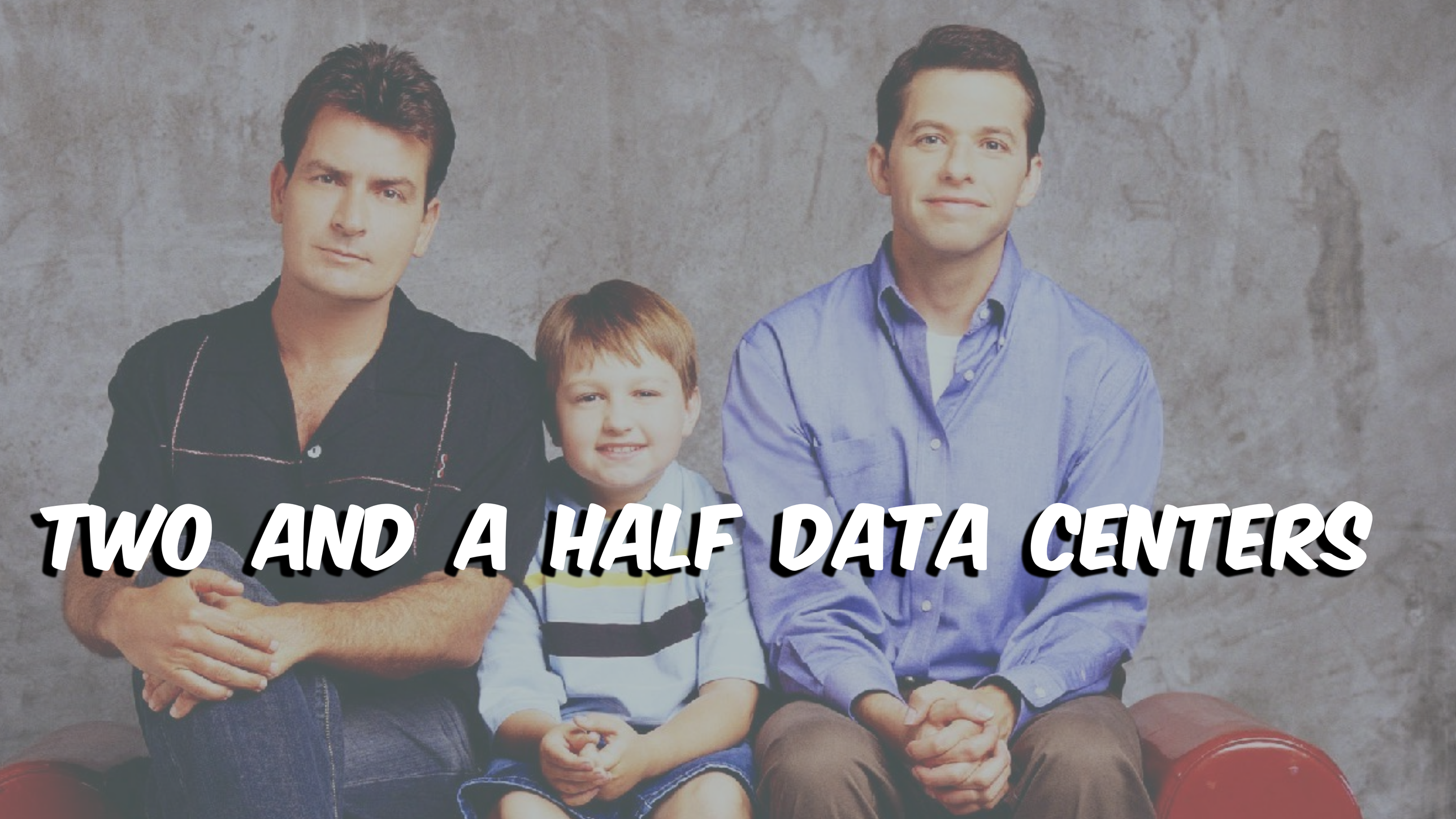
- Production either partially continues or blocks, depending on replication settings

Manual intervention required to resume processing

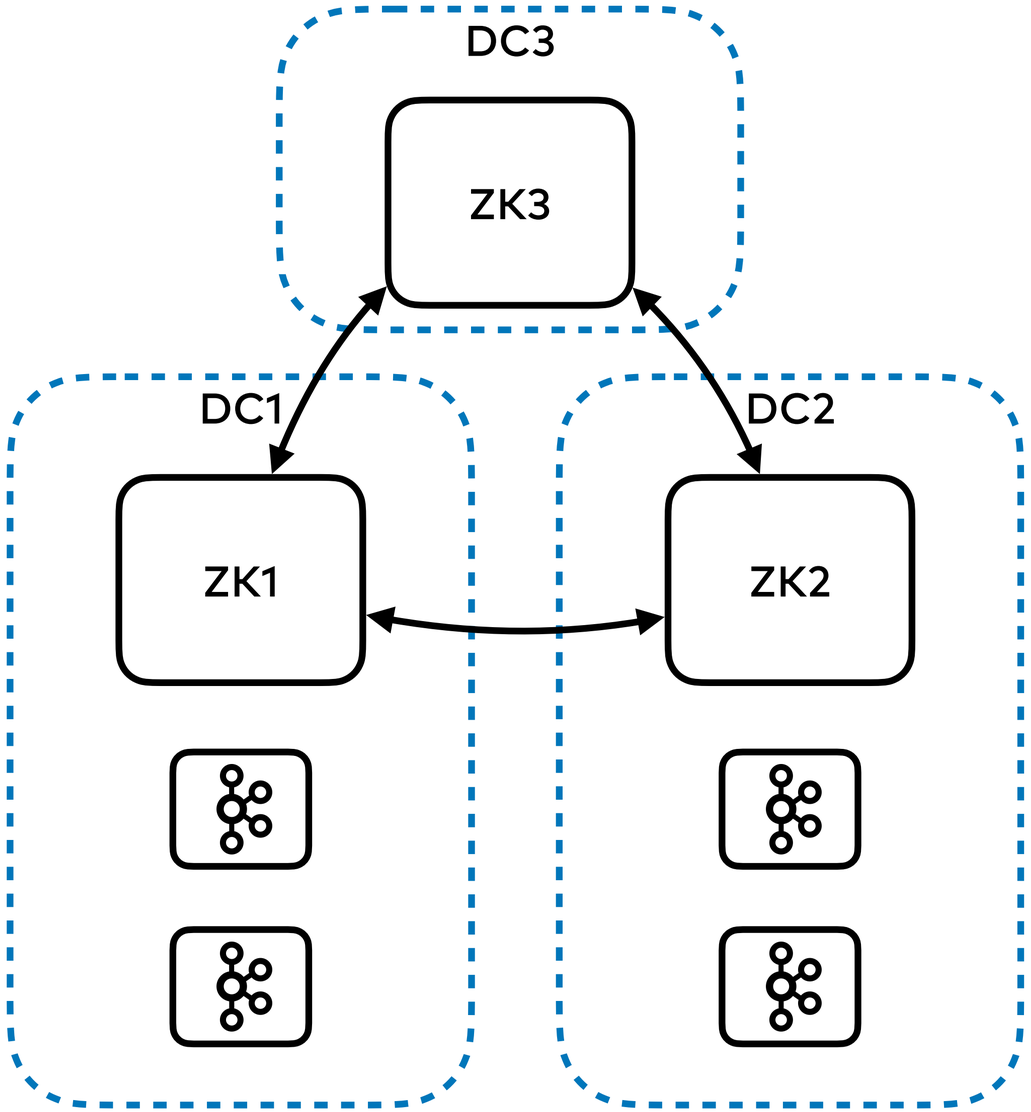
2DC - REPLICATION SETTINGS

replication-factor	min.insync.replicas	enable.unclean.leader.election	Behaviour
4	3	false	**Consistency over Availability** Guarantees that all data is replicated to both DCs. Topics need to be reconfigured during outage to resume flow.
4	2	true	**Availability over Consistency** Data not guaranteed to be replicated to both DCs under some conditions. No topics reconfiguration needed during outage to resume flow.

GitHub Dabz/kafka-boom-boom


TWO AND A HALF DATA CENTERS

2.5 DATA CENTERS



ZooKeeper behaviour same as 3DC setup.

Replication tradeoffs same as 2DC setup.

Mirroring

MULTI-DC VIA MIRRORING

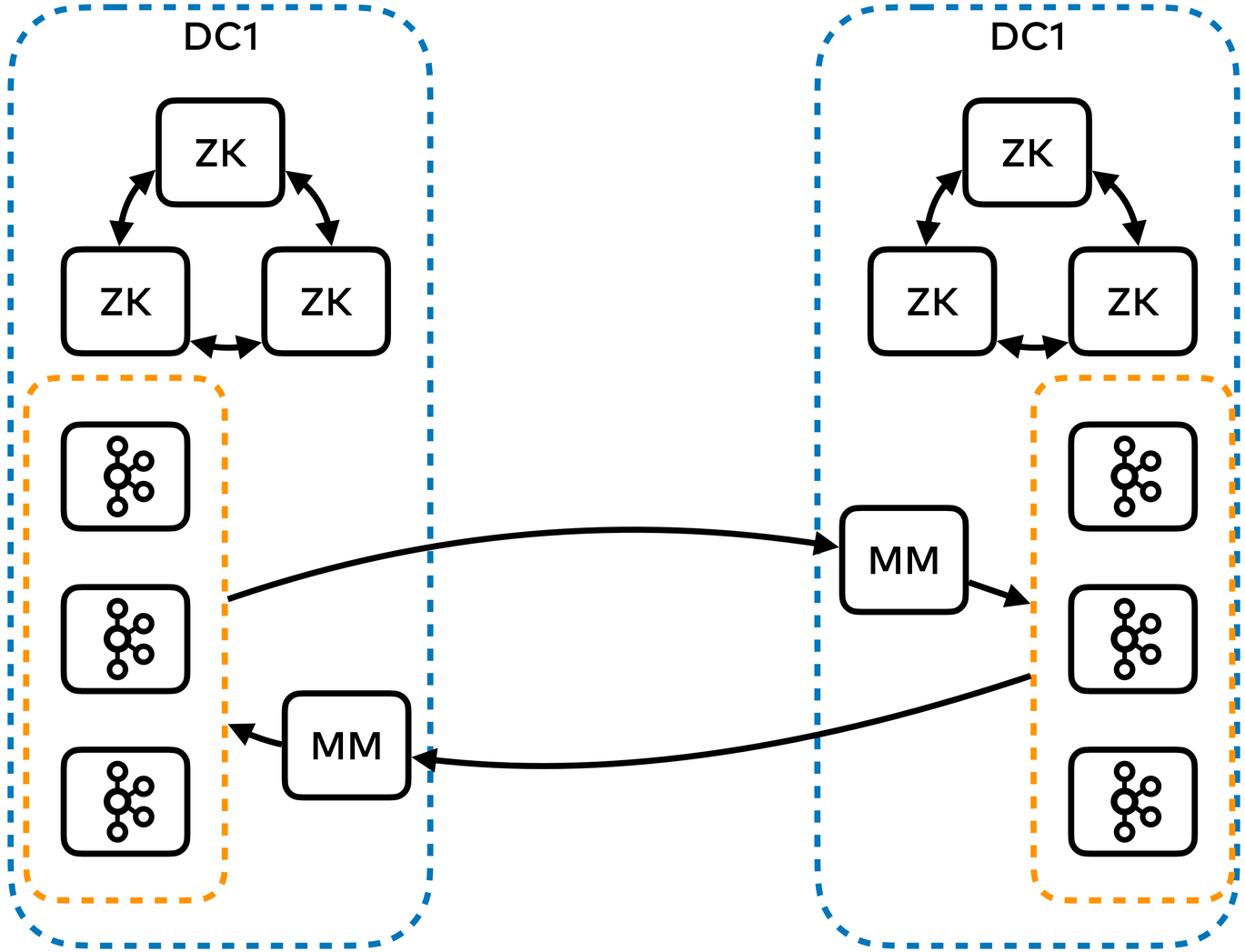
Drivers

- Don't have 3 Data Centers or inter-DC latency >30ms
- Can't accept data loss
- Can't accept stop the world

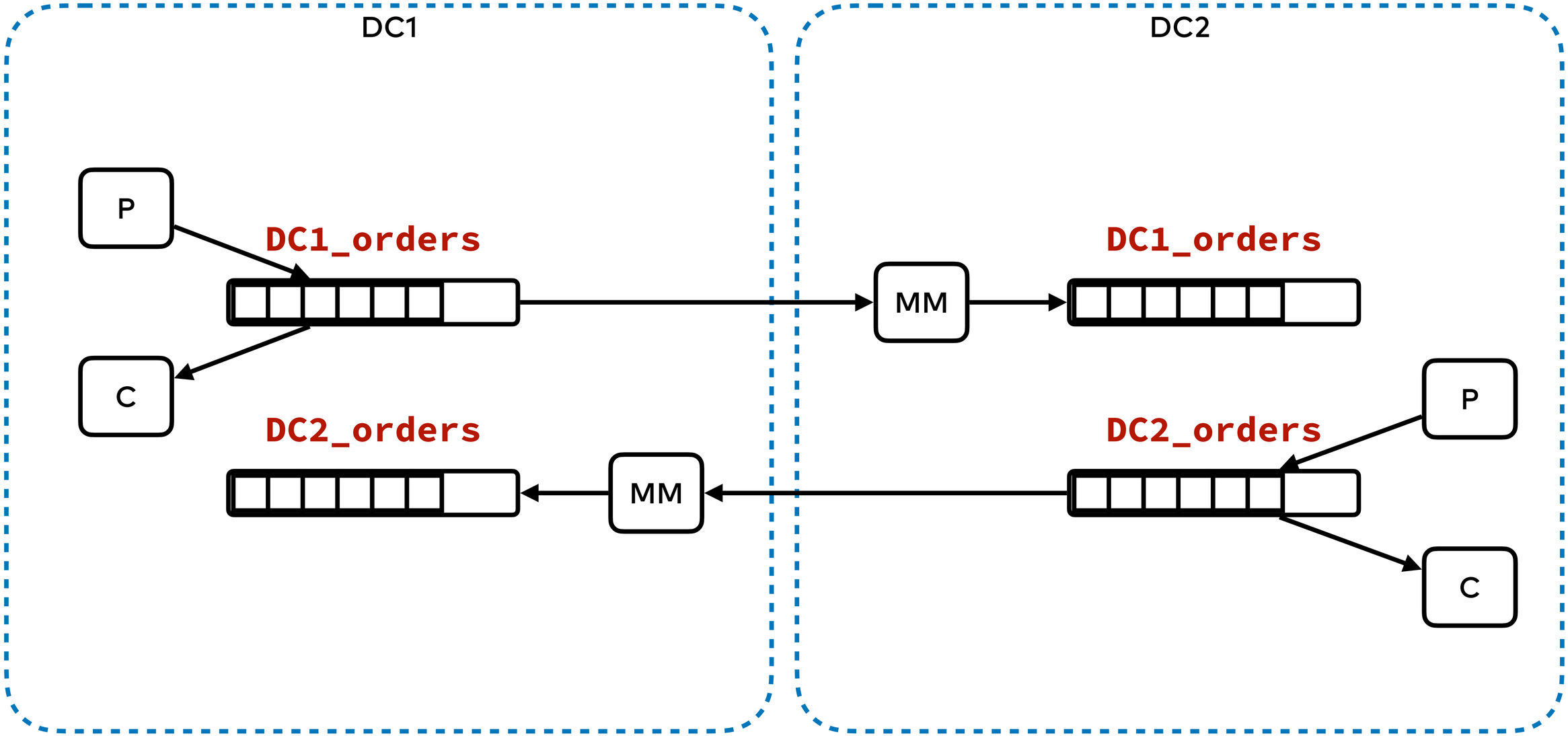
Characteristics

- Uses multiple clusters
- Asynchronous
- Typical uses are uni-directional
- Typically used for inter-region traffic

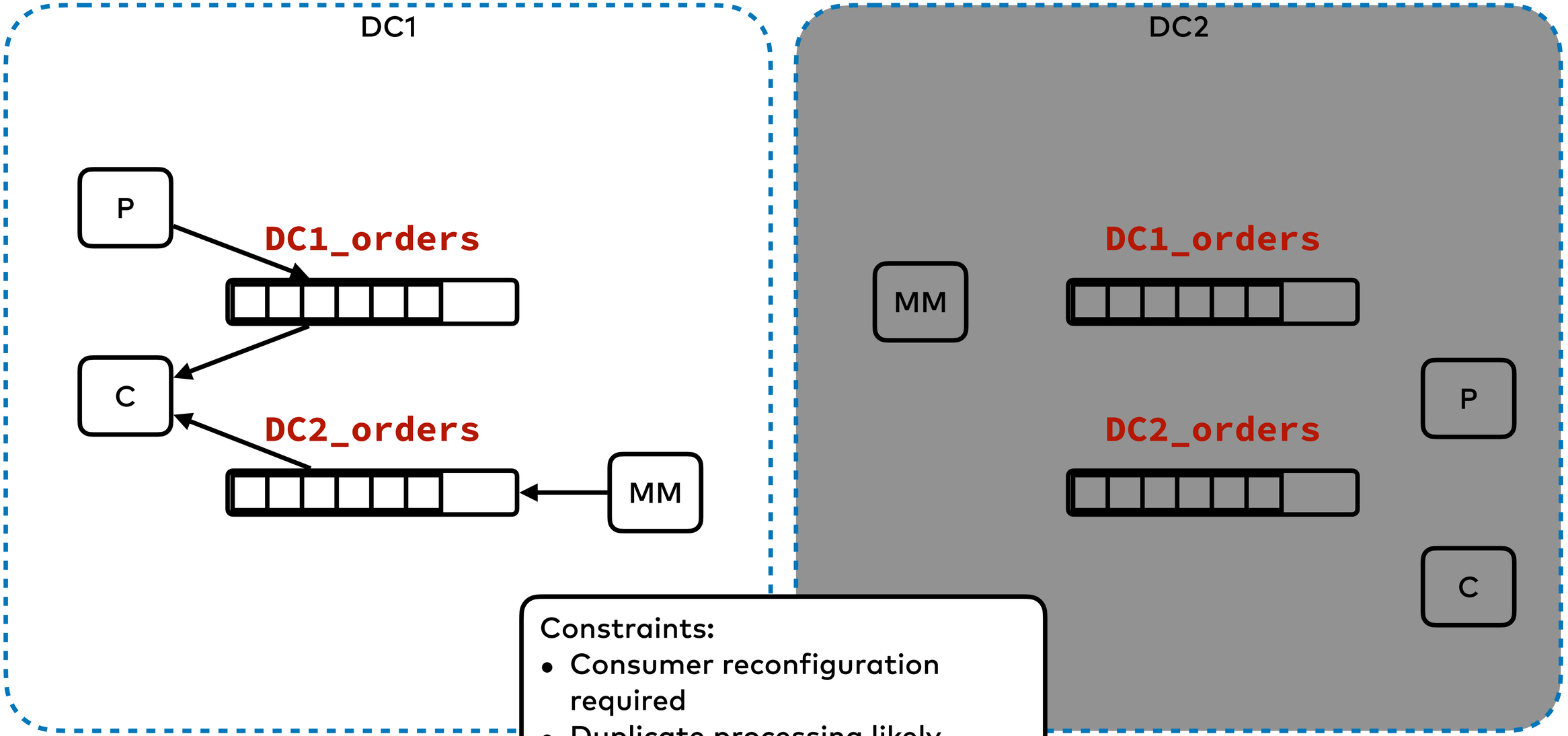
MIRRORING - GENERAL SETUP



LOCATION-PREFIXED TOPICS



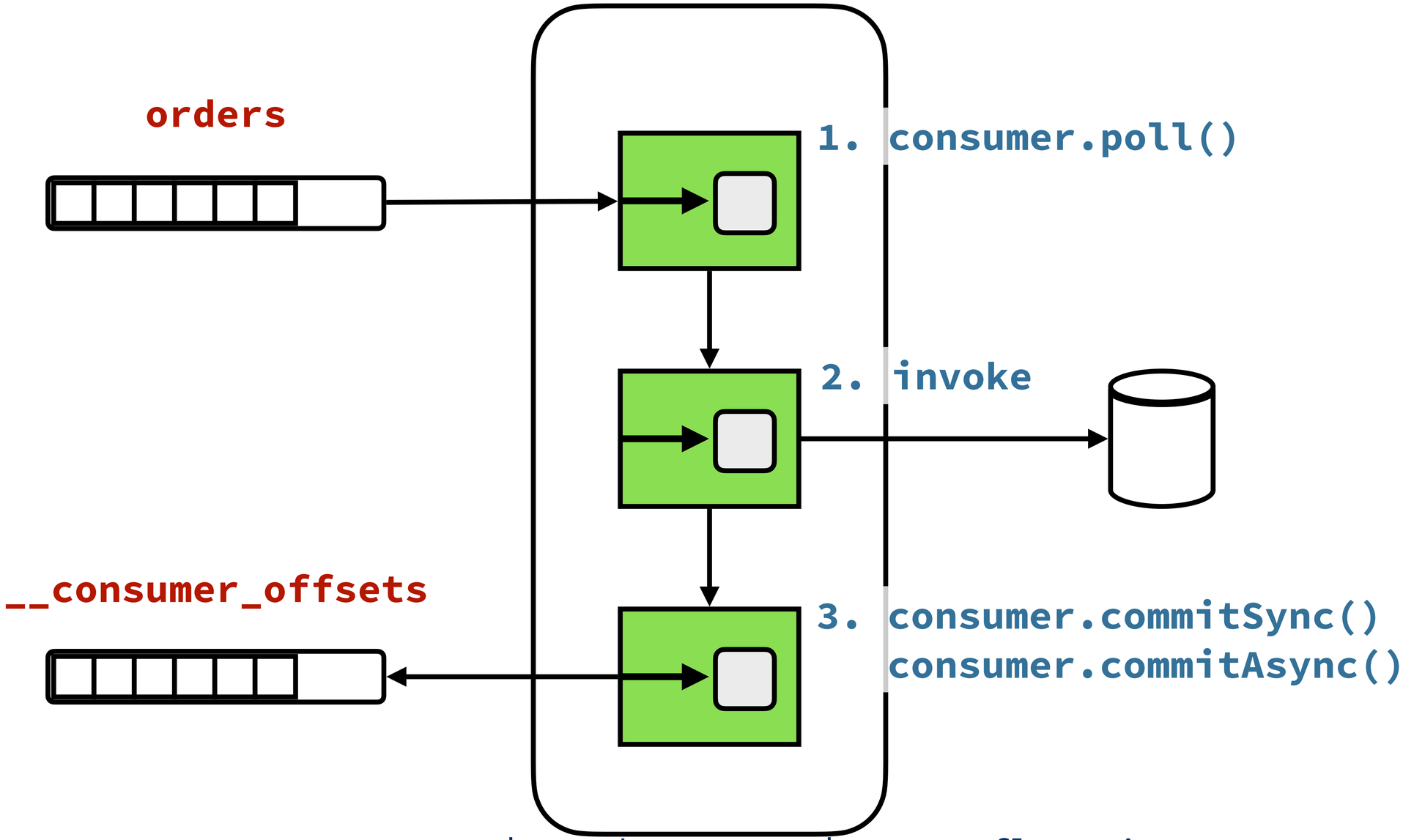
DEALING WITH DC FAILURE



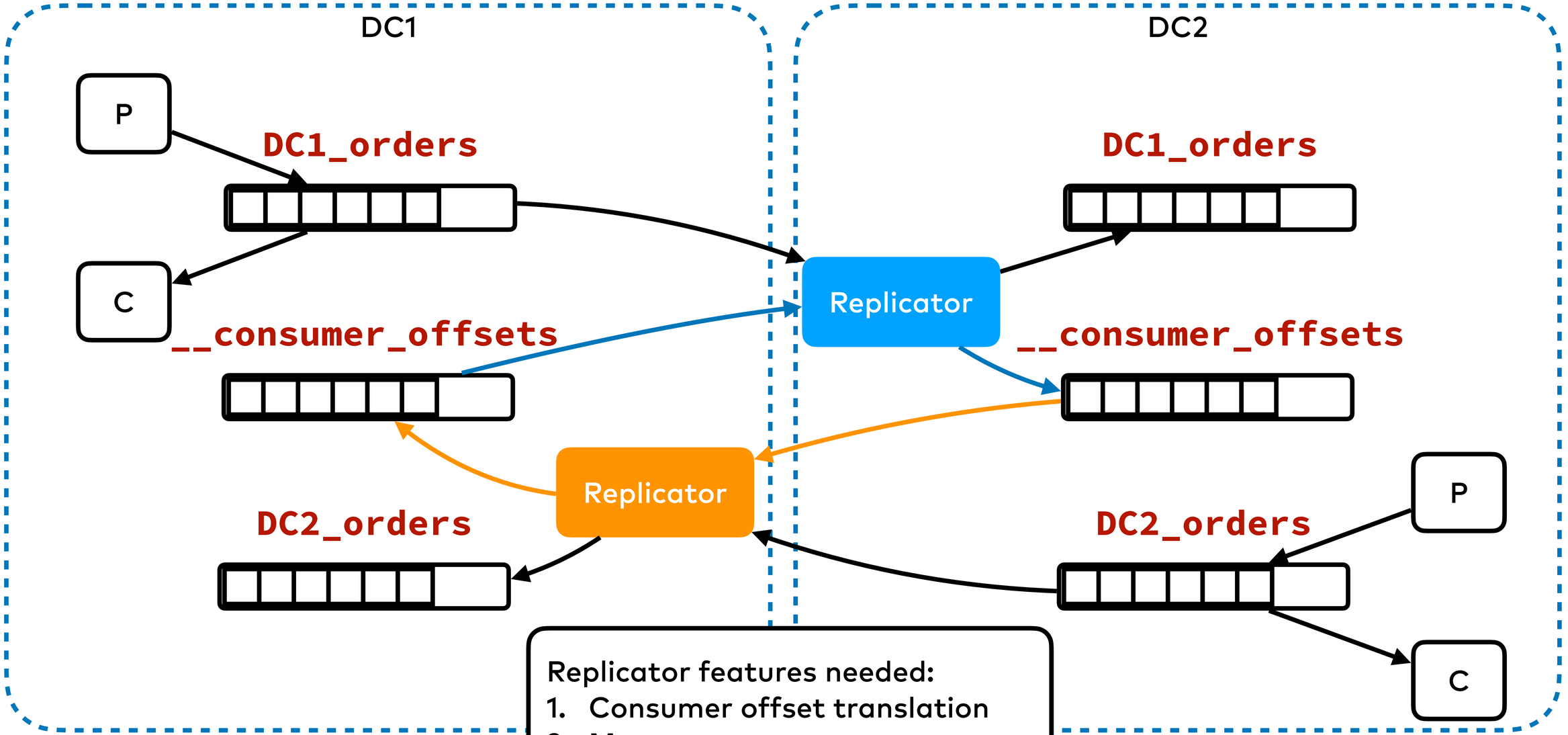
Constraints:

- Consumer reconfiguration required
- Duplicate processing likely

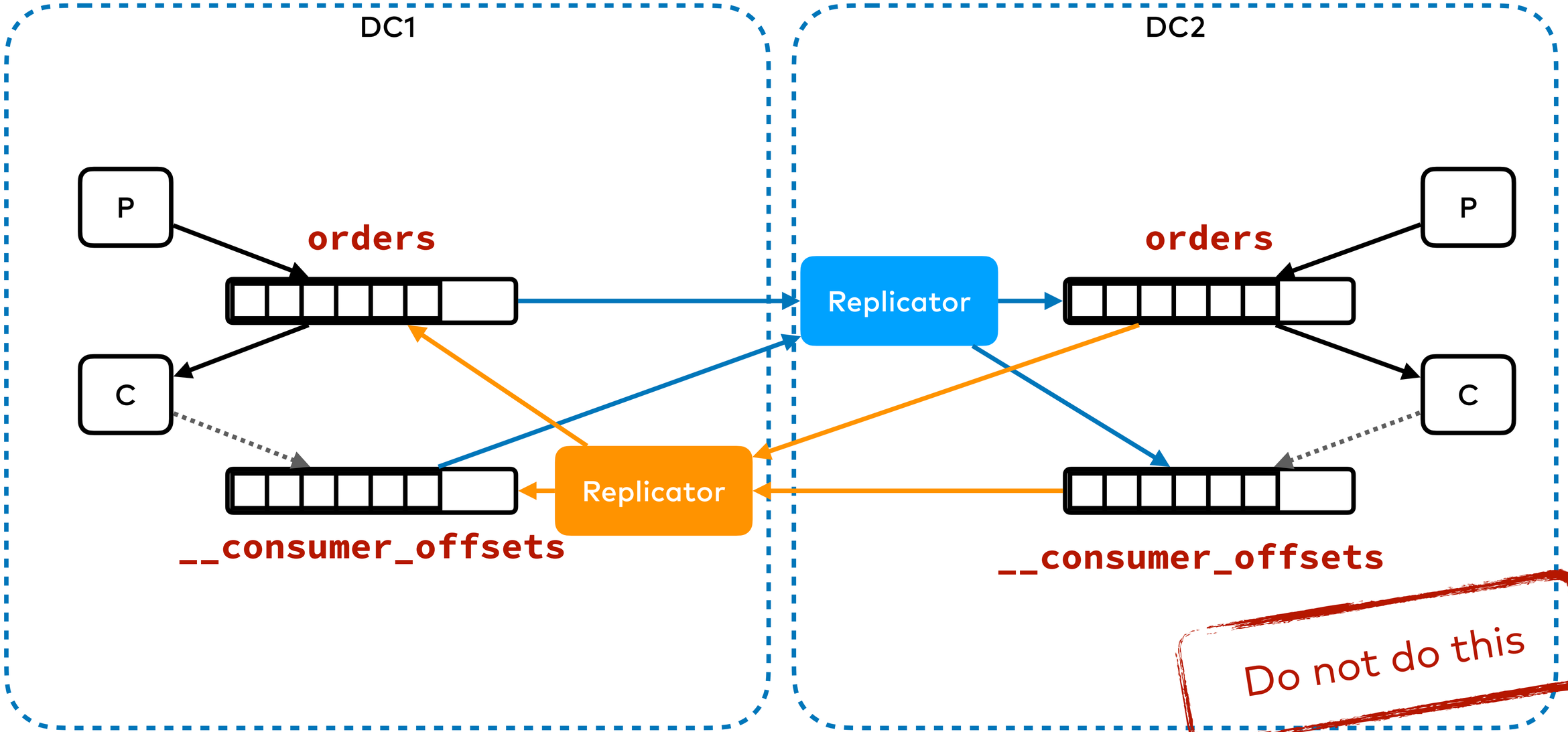
CONSUMER OFFSETS



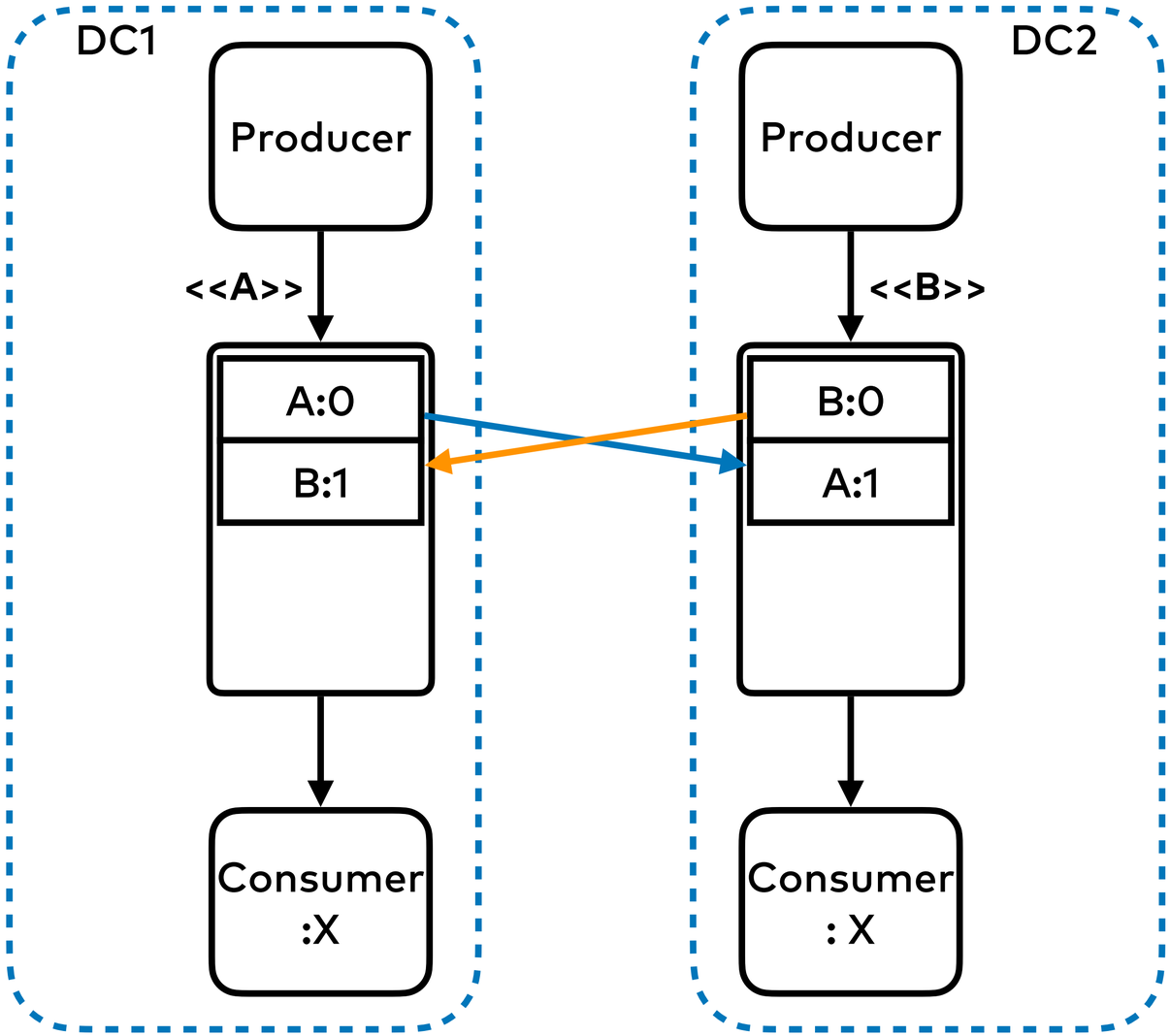
MIRRORING OFFSETS



SINGE TOPIC + MIRRORED OFFSETS

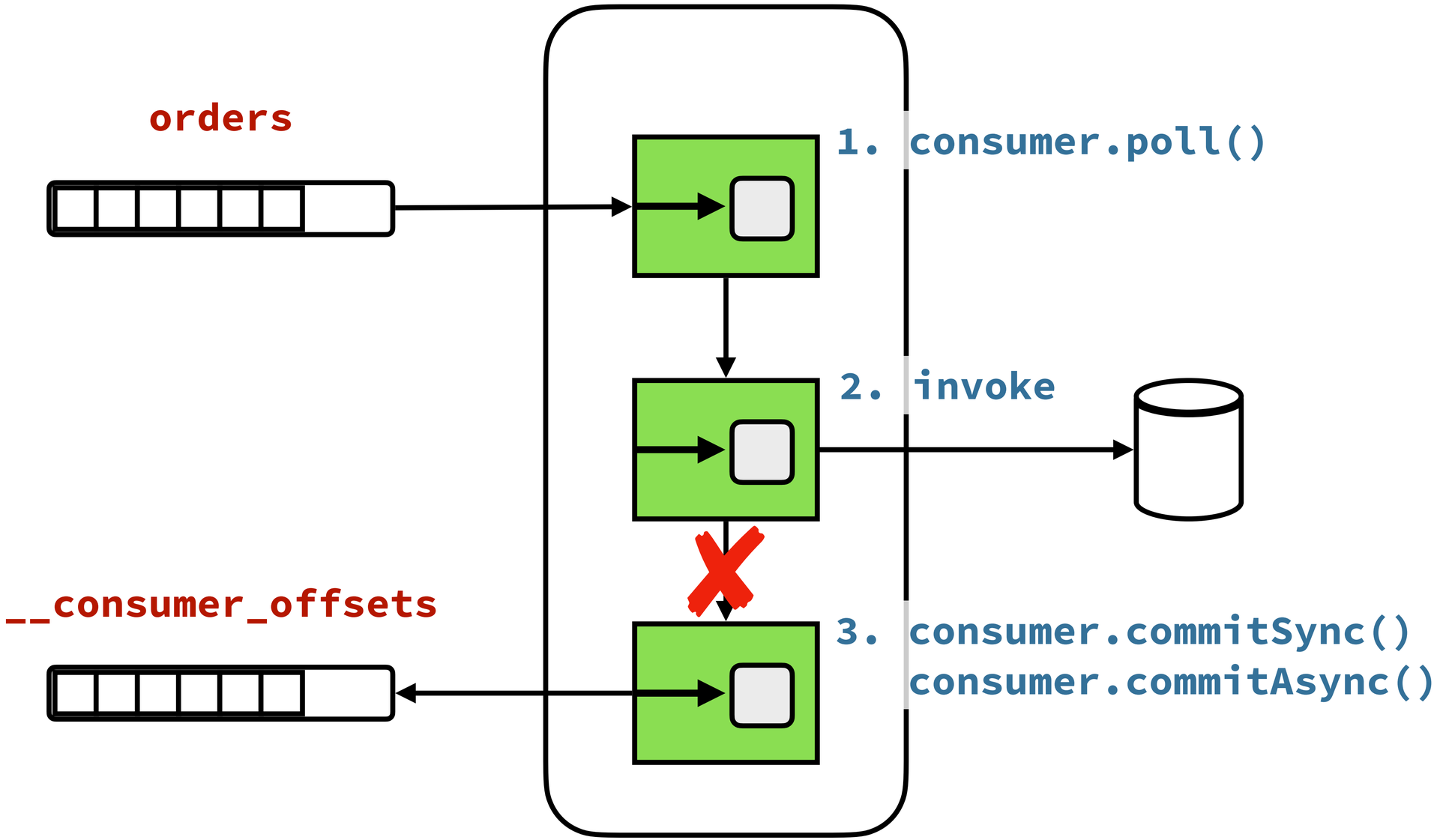


INCONSISTENT MESSAGE ORDERING

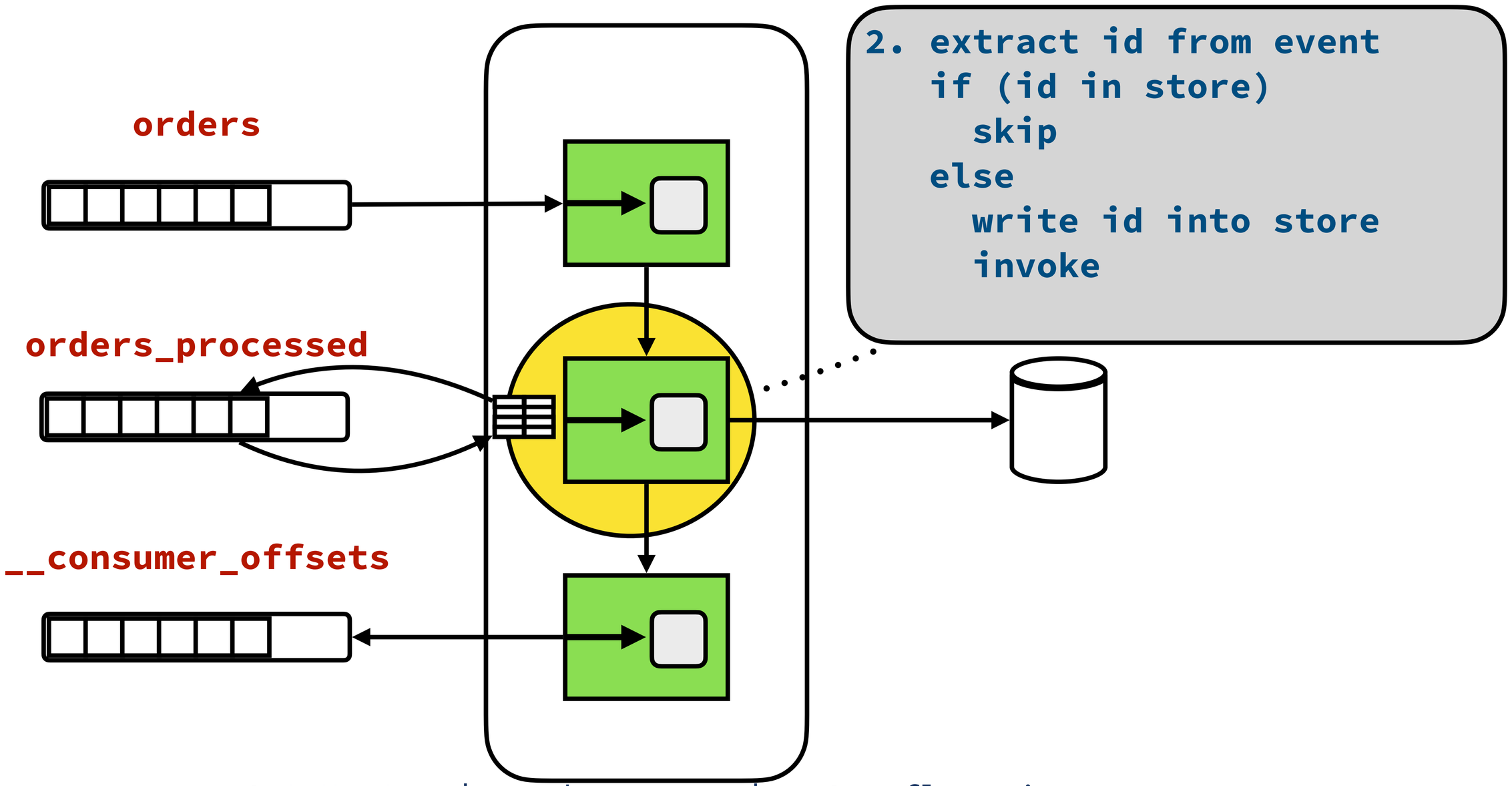


What does it mean for consumer group X to be up to offset 1?

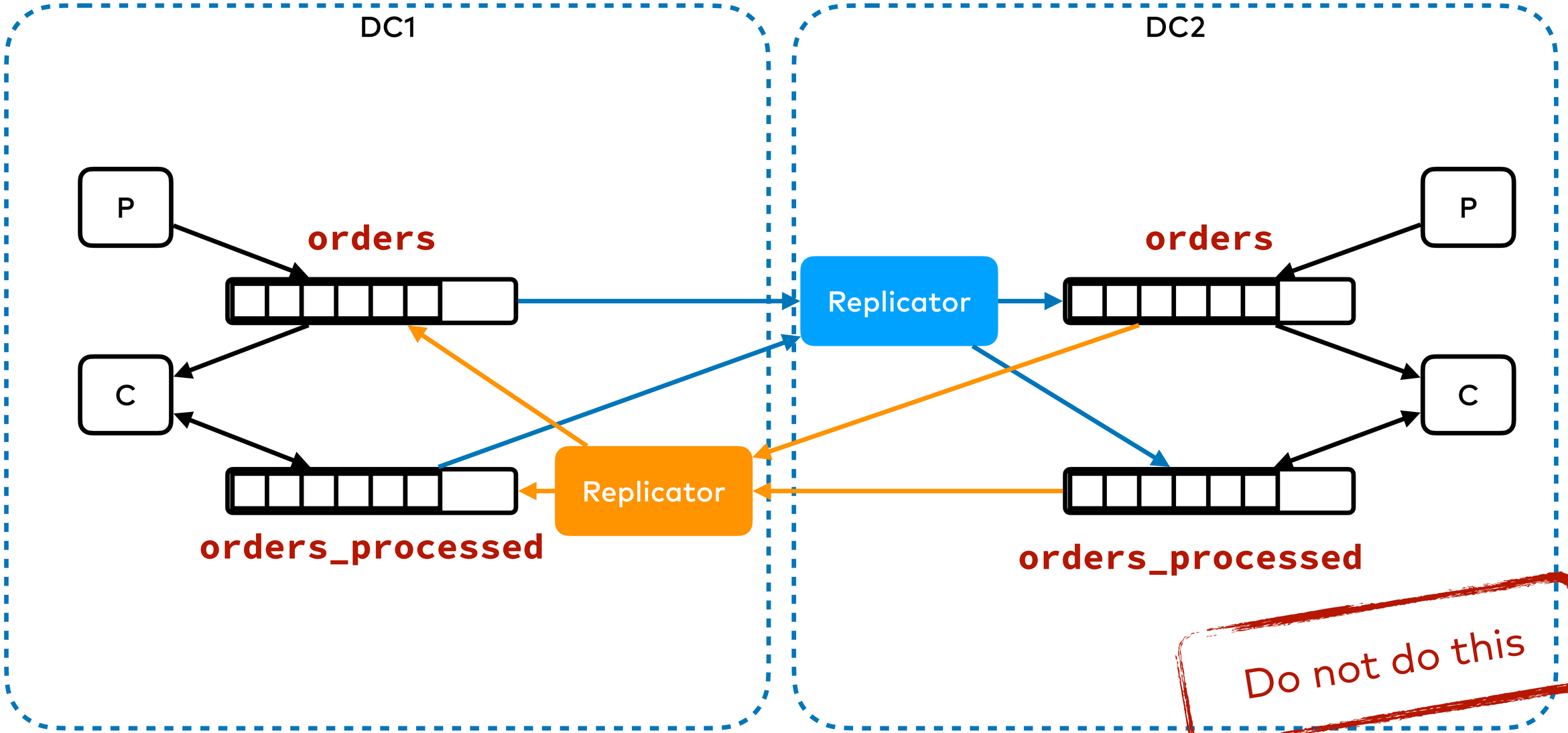
THE LIMITS OF OFFSETS



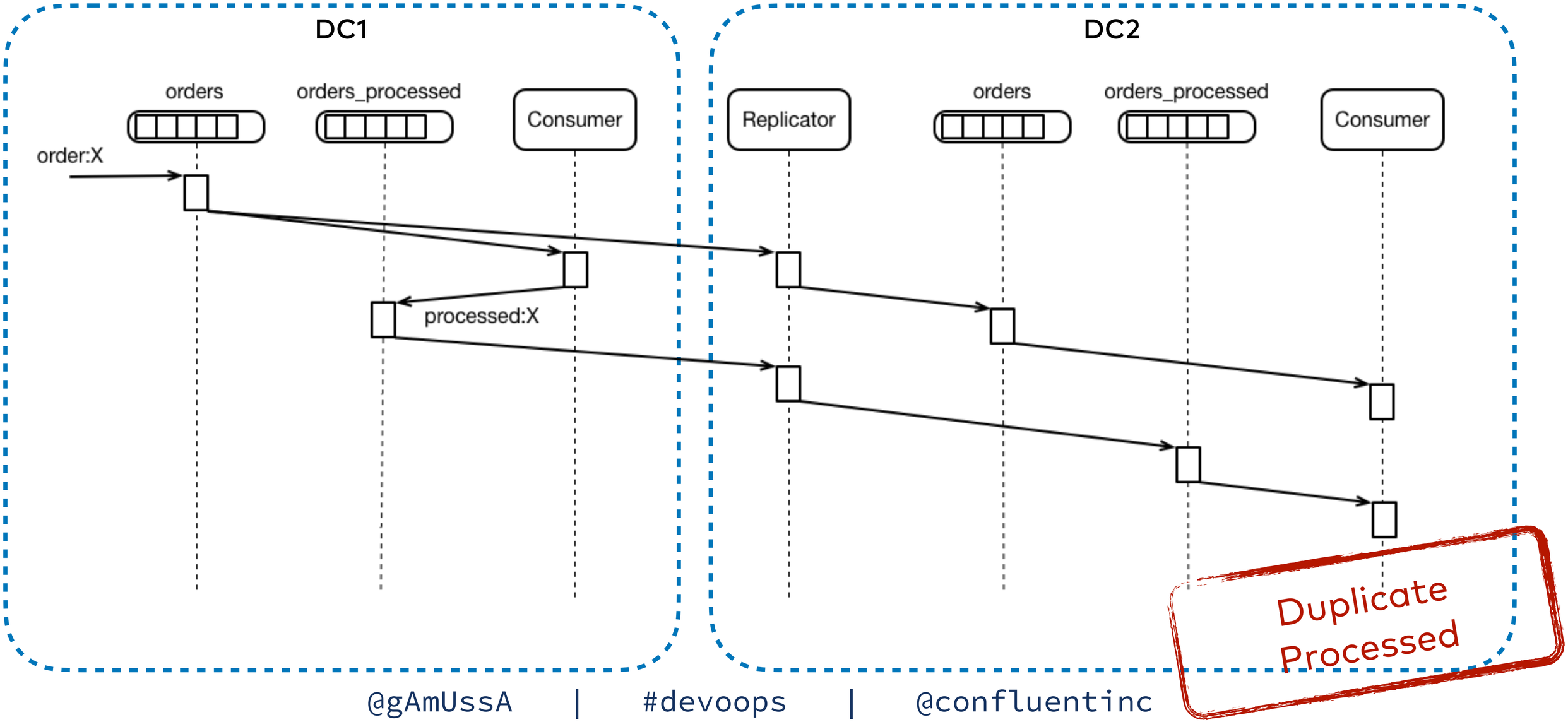
IDEMPOTENT CONSUMPTION



"WHAT IF WE DO THIS..."



MIRRORING PROCESSED STATE



ONE LAST THING...

MULTI-REGION REPLICATION - STRETCH CLUSTER DONE RIGHT

https://gamov.dev/mrc-demo

Following Fetching aka KIP-392

allows consumers to read from a replica other than the leader

Observers

Aka async replicas which are not part of ISR and can't be elected a leader

Replica Placement

JSON-based specification allows you to specify replica assignment as a set of matching constraints. For example, allows to keep the regular replicas in a single region and putting an observer in a different region

WHAT YOU NEED TO KNOW

- Stretched clusters are awesome; assuming <~30ms latency
  - **3DC > 2.5DC > 2DC**
- KIP-392 will make them even better
  - Confluent Server takes this further
- Mirroring is an alternative
  - Asynchronicity means it acts differently than a single cluster
  - Think about the impacts on operations and design of your code

CONFLUENT

@gAmUssA | #devoops | @confluentinc