April , 2021

RabbitMQ

Reliable messaging

- Message Delivery Guarantees
- Message Ordering Guarantees
- Message Durability
- High Availability

Message Acknowledgement Protocols

At-most-once
At-least-once

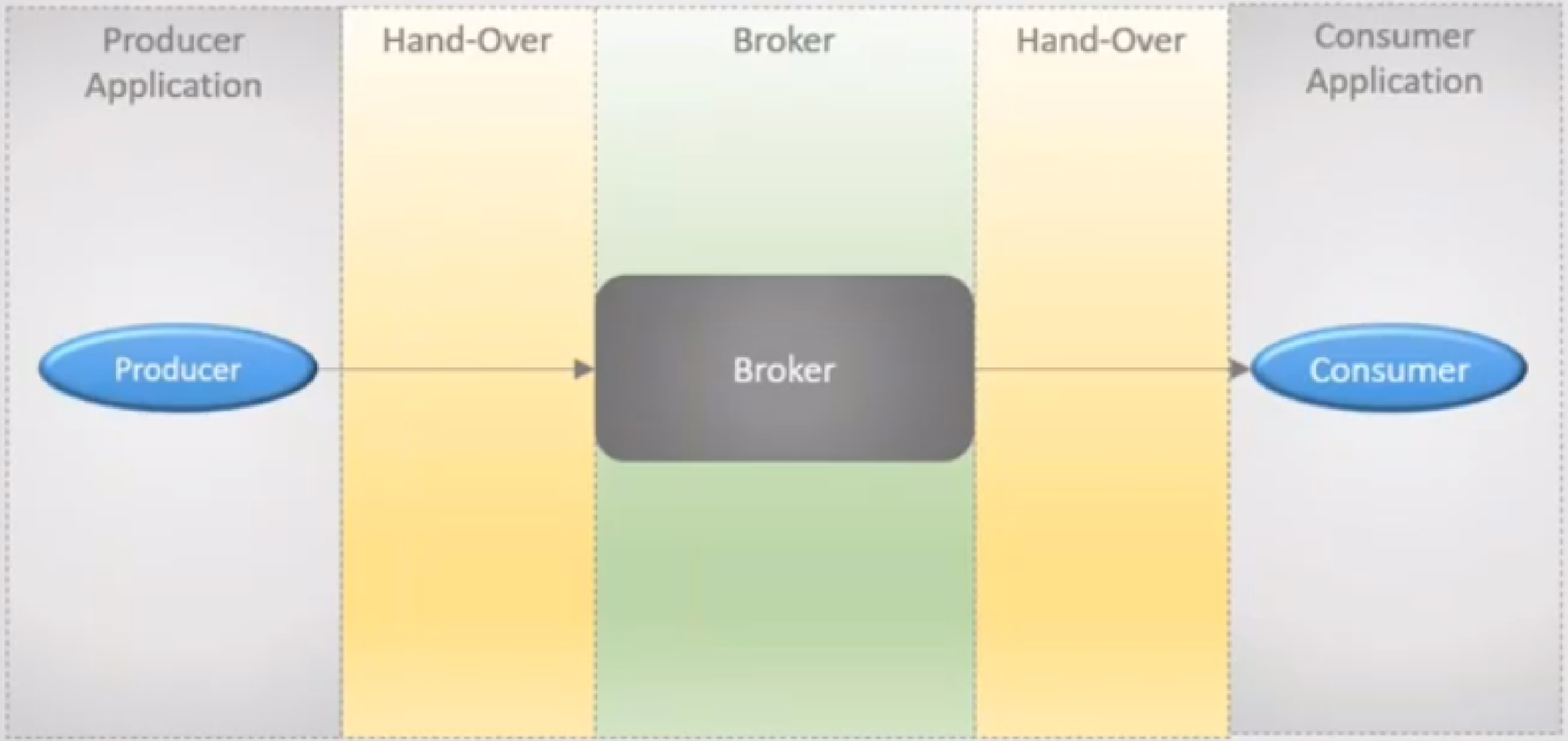
At-most-once.
This means that a message will never be delivered more than once but messages might be lost.

At-least-once.
This means that we'll never lose a message but a message might end up being delivered to a consumer more than once.

Exactly-once.
The holy grail of messaging. All messages will be delivered exactly one time.

Delivery vs Processing
Delivered twice to be processed once.

Chain of Responsibility



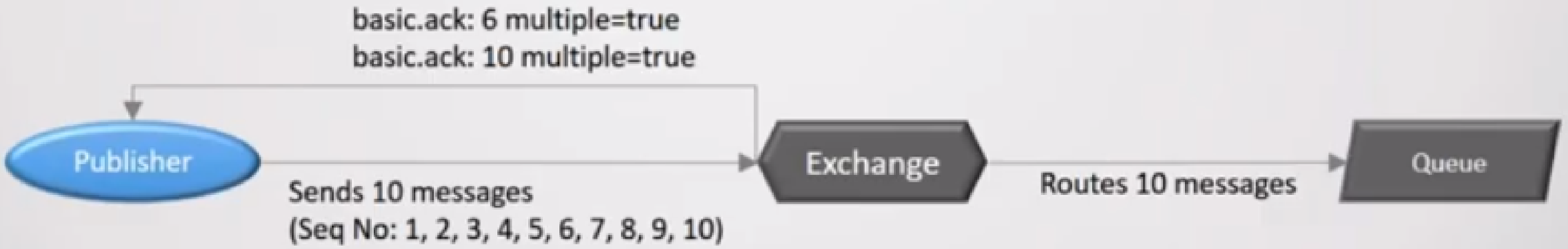
RabbitMQ - Producer Side Acknowledgements

Publisher Confirms

- **basic.ack** (all ok!)
- **basic.nack** (error!)
- **basic.return + basic.ack** (undeliverable!)

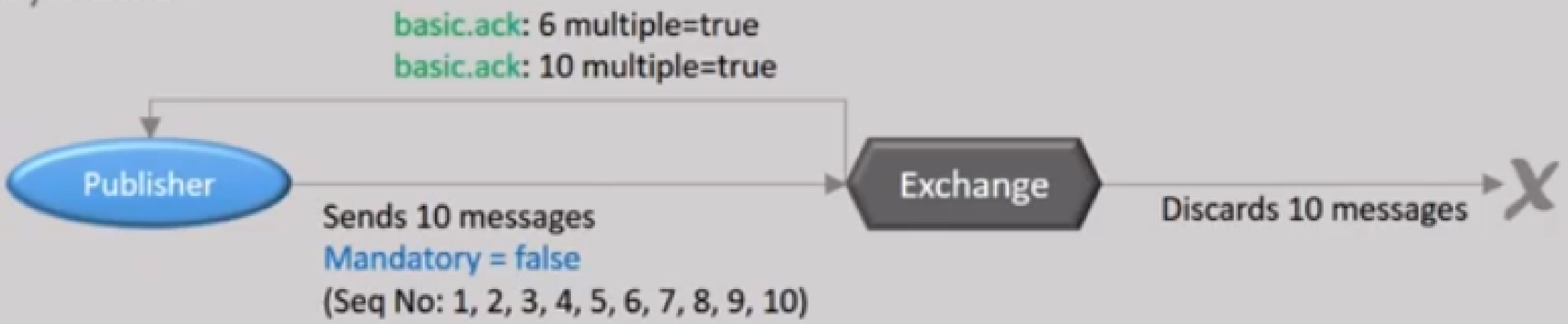
Flags

- **Multiple** (I am acknowledging multiple message deliveries)
- **Mandatory** (give me a basic.return if you can't deliver to any queues)

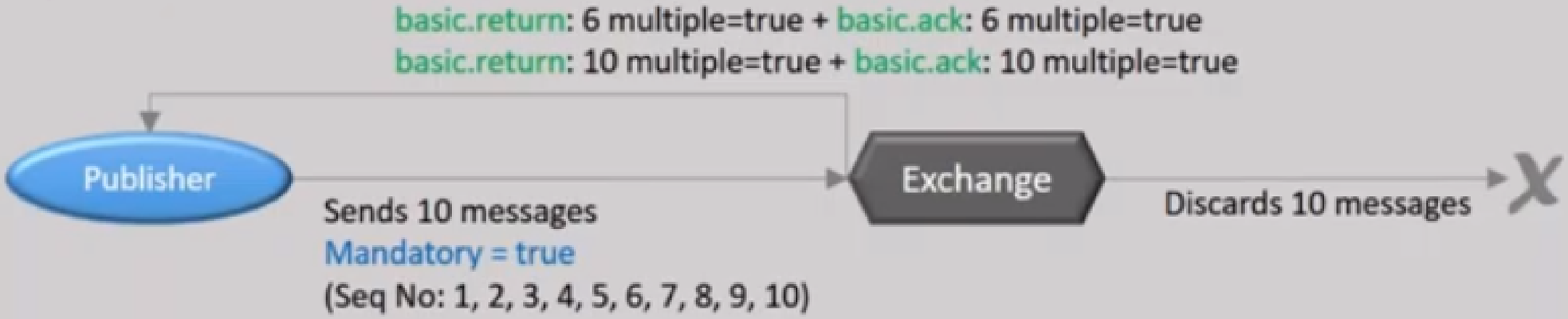


RabbitMQ - Producer Side Acknowledgements

Mandatory=false

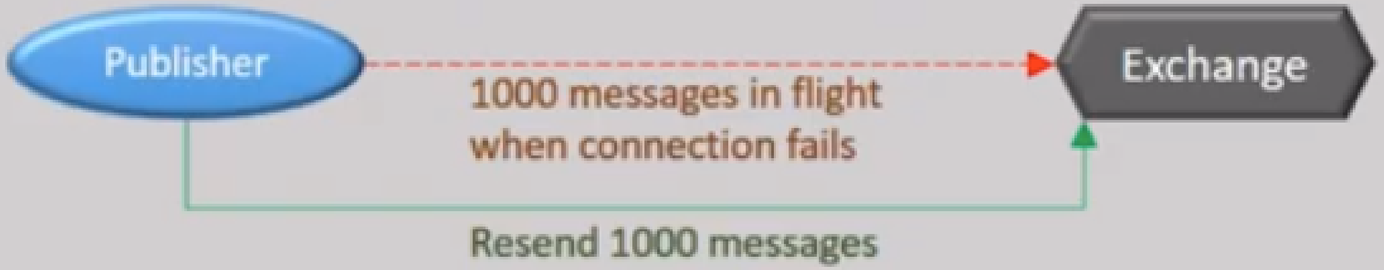


Mandatory=true



RabbitMQ - Producer Side Duplication

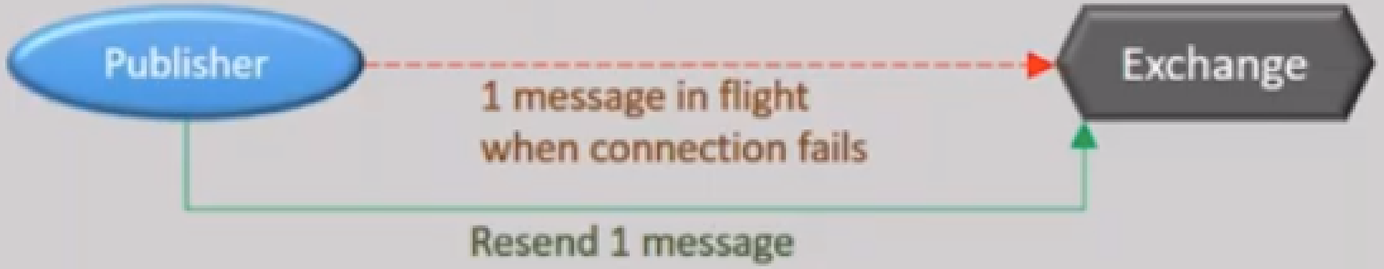
Large # of Messages in Flight = High Throughput, High Message Duplication on Failure



25% of the messages persisted to a queue

Queues ends up with 250 duplicates

Low # of Messages in Flight = Low Throughput, Low Message Duplication on Failure



Message was persisted to a queue but connection died before ack could be sent.

Queues ends up with 1 duplicate
(Resent custom header)

RabbitMQ - Consumer Side Acknowledgements

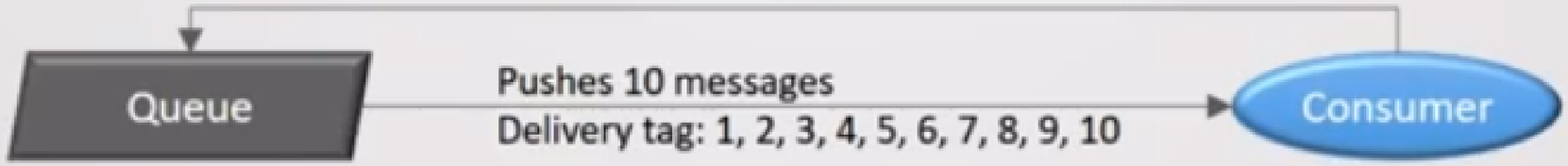
Acknowledgement Mode

- **Auto Ack** (Push me messages as fast as you can!)
- **Manual Ack** (I will explicitly tell you when a message can be removed from the queue)

Consumer Acknowledgements

- **basic.ack** (all ok, remove from the queue!)
- **basic.nack, redeliver=false** (error, but remove anyway)
- **basic.nack, redeliver=true** (error, please redeliver)
- **basic.reject** (same as basic.nack but without multiple flag support)

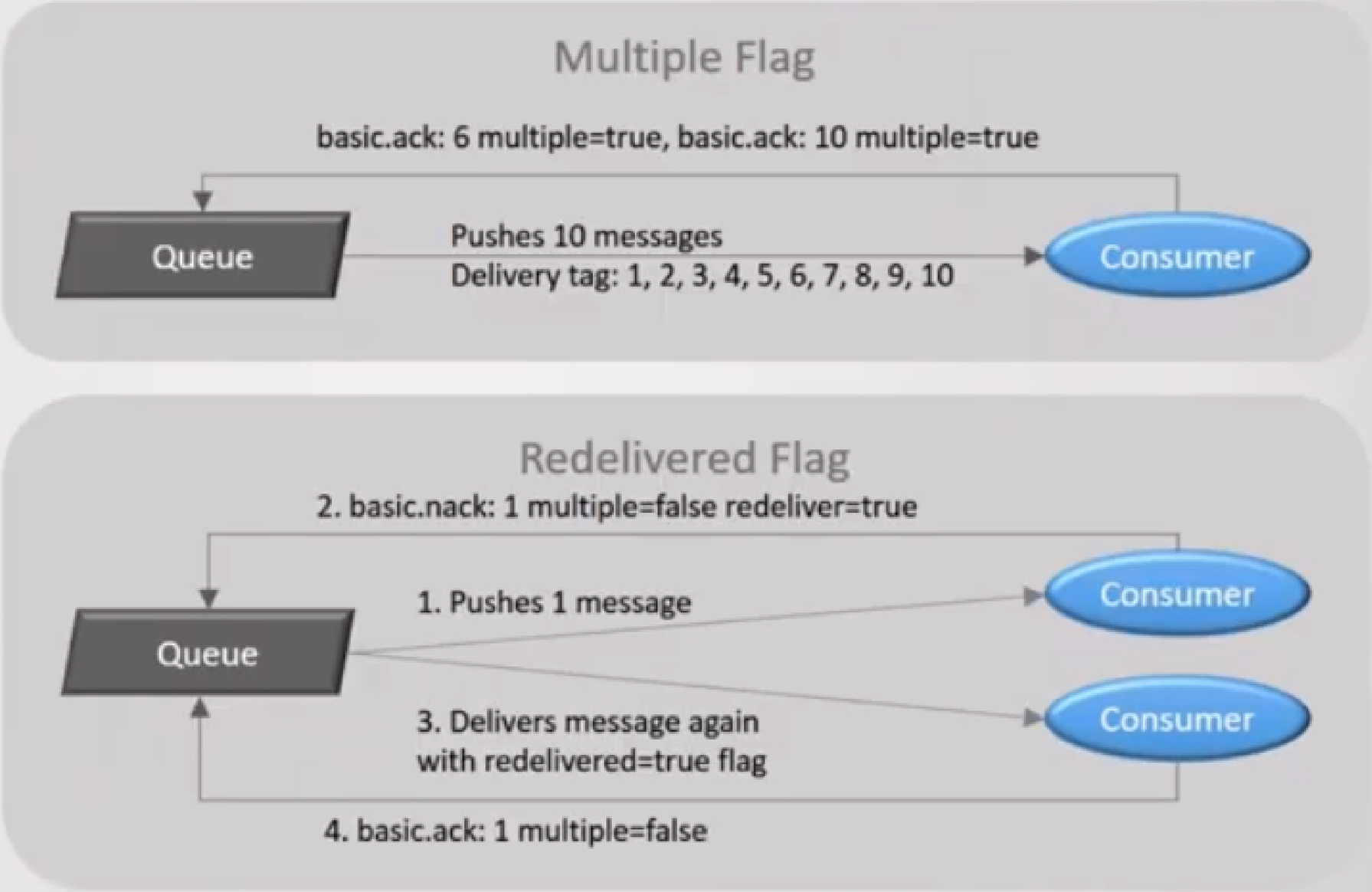
basic.ack: 1 multiple=false, basic.ack: 2 multiple=false
basic.ack: 3 multiple=false, basic.ack: 4 multiple=false
basic.ack: 5 multiple=false, basic.ack: 6 multiple=false
basic.ack: 7 multiple=false, basic.ack: 8 multiple=false
basic.ack: 9 multiple=false, basic.ack: 9 multiple=false



RabbitMQ - Consumer Side Acknowledgements

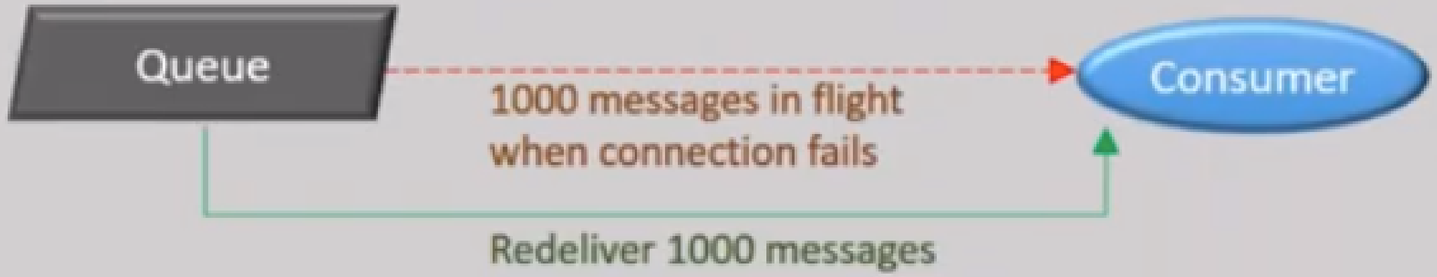
Flags

- **Multiple** (I am acknowledging multiple messages)
- **Redelivered** (This message is a redelivery)



RabbitMQ - Consumer Side Duplication

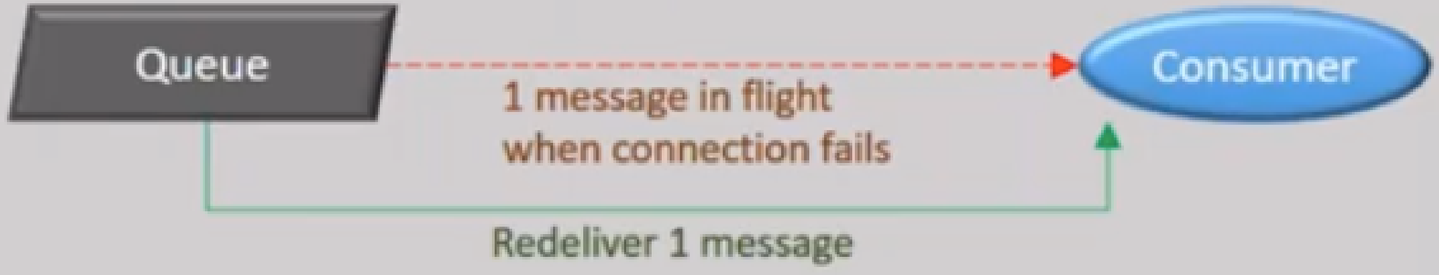
Large # of Messages in Flight = High Throughput, High Message Duplication on Failure



25% of the messages processed, but before ack could be sent when connection failed

250 messages get processed twice

Low # of Messages in Flight = Low Throughput, Low Message Duplication on Failure



Message was processed but connection died before ack could be sent.

1 message gets processed twice

RabbitMQ

Broker Durability

Durable Queues
Persistent Messages
Mirrored Queues

RabbitMQ - The Broker

Surviving Broker Restart

- Durable Queue
- Persistent Message

Surviving Broker Loss

- Queue Mirroring (Clustering)

	Broker Restart	Broker Loss
Non-Durable Queue Non-Persistent Message	Queue Message	Queue Message
Durable Queue Non-Persistent Message	Queue Message	Queue Message
Durable Queue Persistent Message	Queue Message	Queue Message
Mirrored Queue Persistent Message	Queue Message	Queue Message

RabbitMQ - The Broker

Surviving Broker Restart

- Durable Queue
- Persistent Message

Surviving Broker Loss

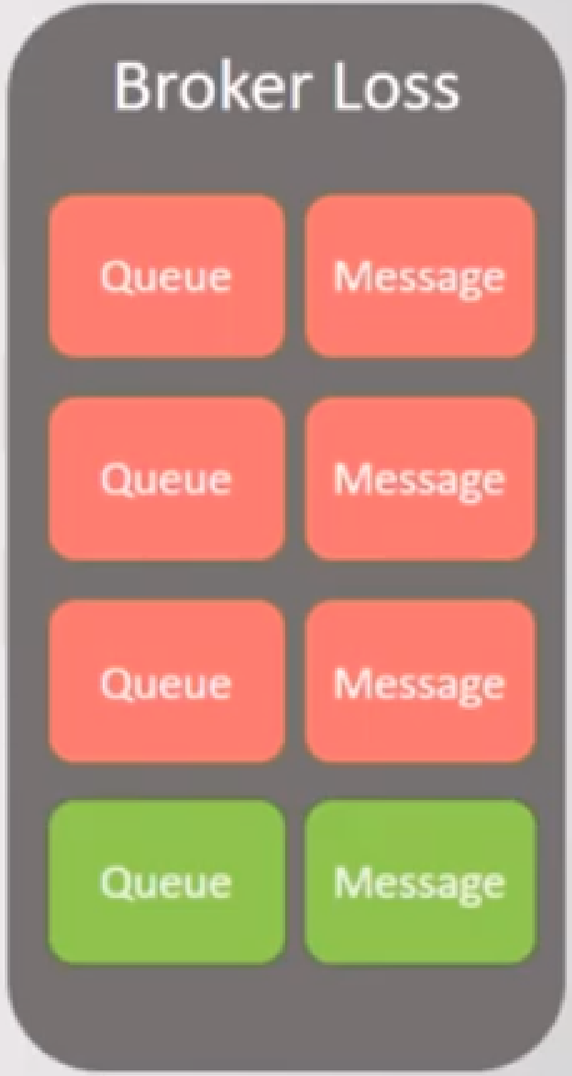
- Queue Mirroring (Clustering)

Non-Durable Queue
Non-Persistent Message

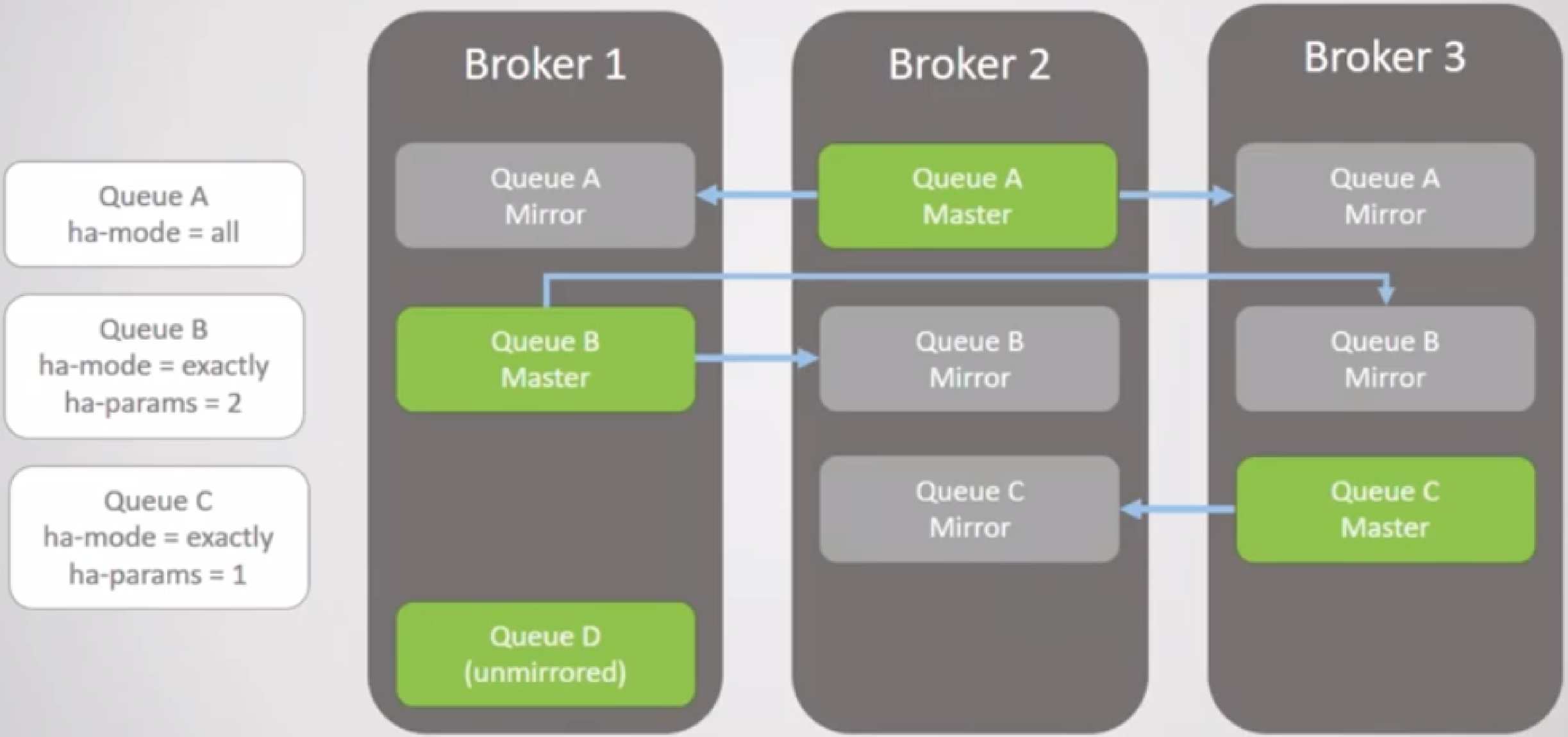
Durable Queue
Non-Persistent Message

Durable Queue
Persistent Message

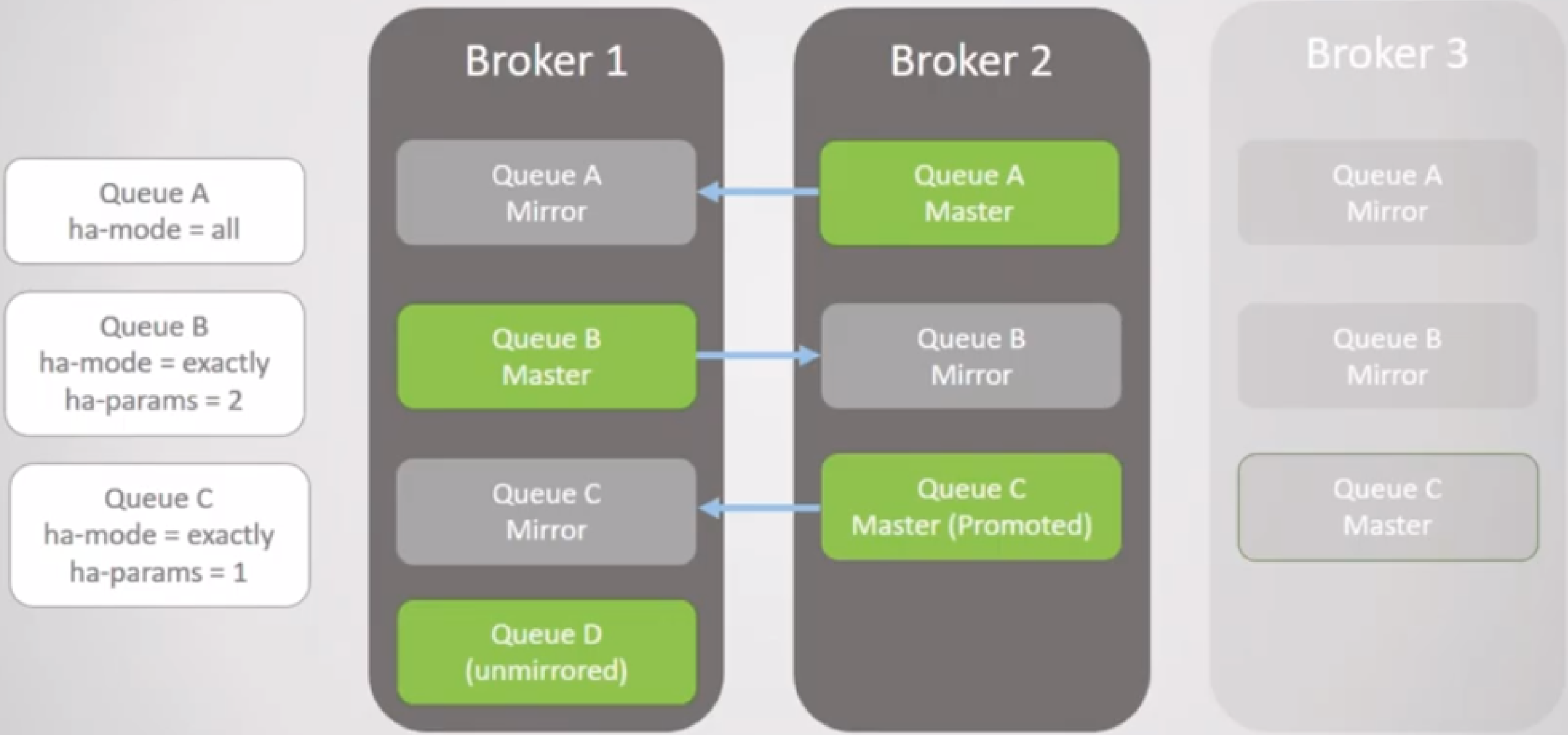
Mirrored Queue
Persistent Message



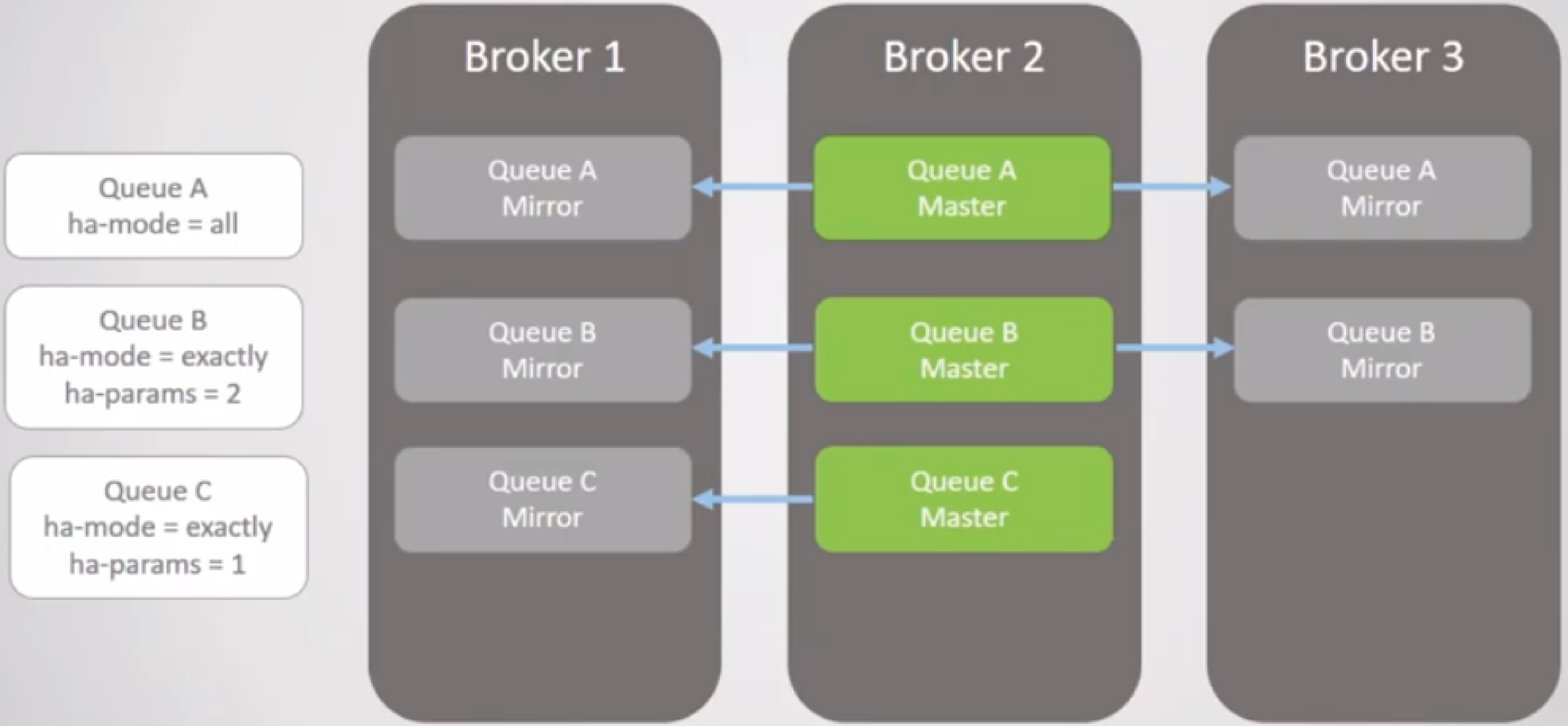
RabbitMQ - The Broker - Queue Mirrors



RabbitMQ - The Broker - Queue Mirrors



RabbitMQ - The Broker - Queue Mirrors



RabbitMQ

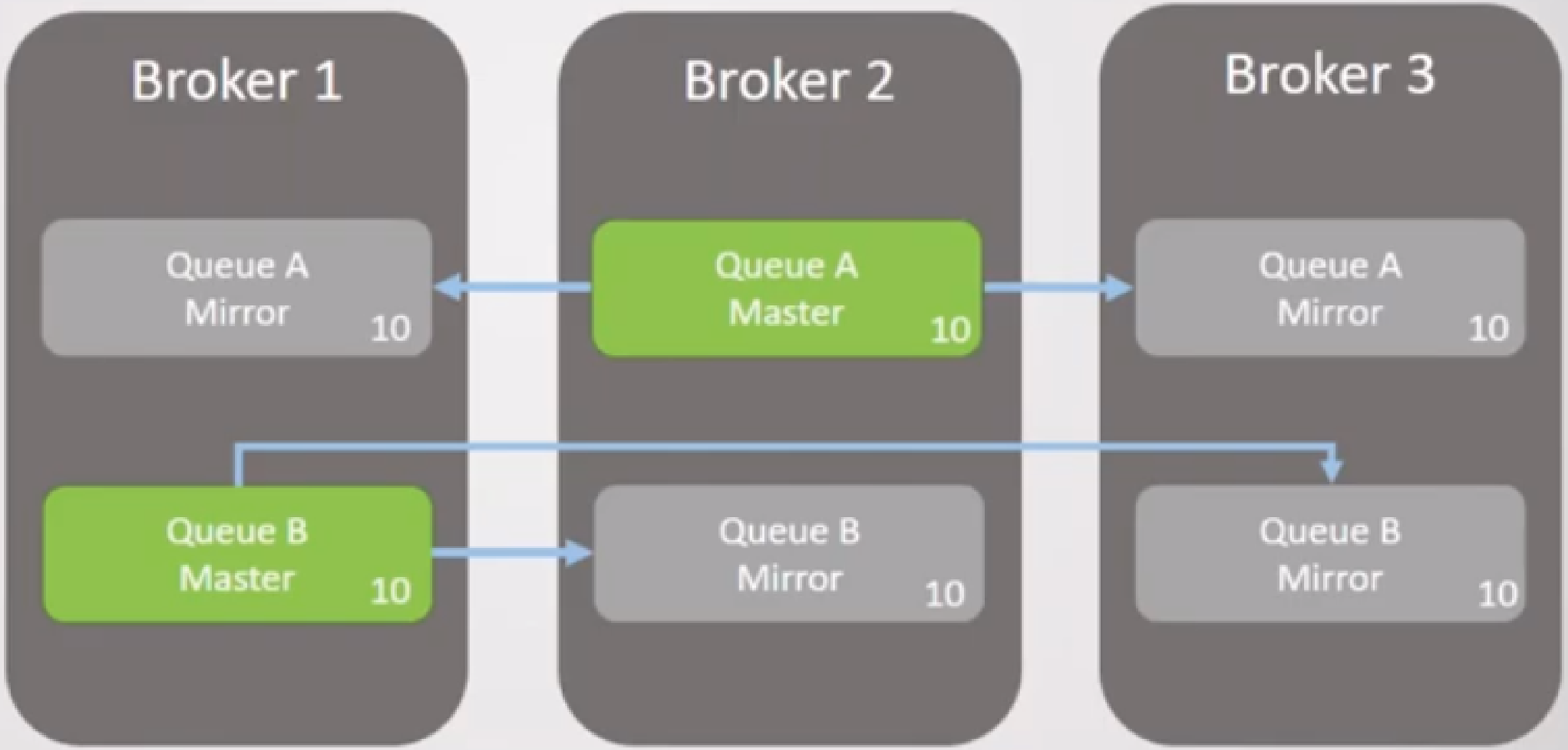
Queue Mirror
Synchronization
And
Queue Failover

RabbitMQ - Queue Mirrors - Synchronization

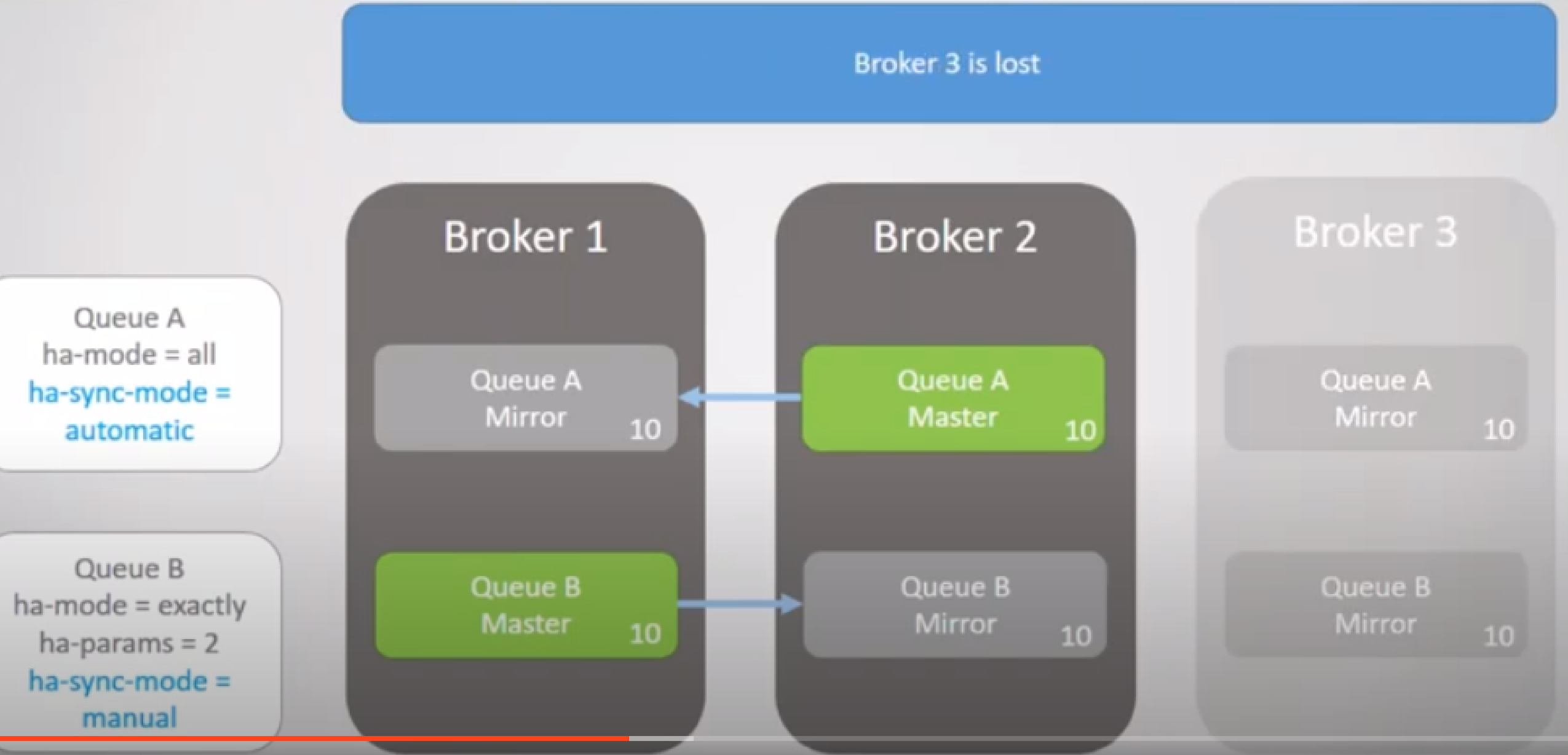
Three nodes, two mirrored queues each with 10 messages

Queue A
ha-mode = all
ha-sync-mode = automatic

Queue B
ha-mode = exactly
ha-params = 2
ha-sync-mode = manual

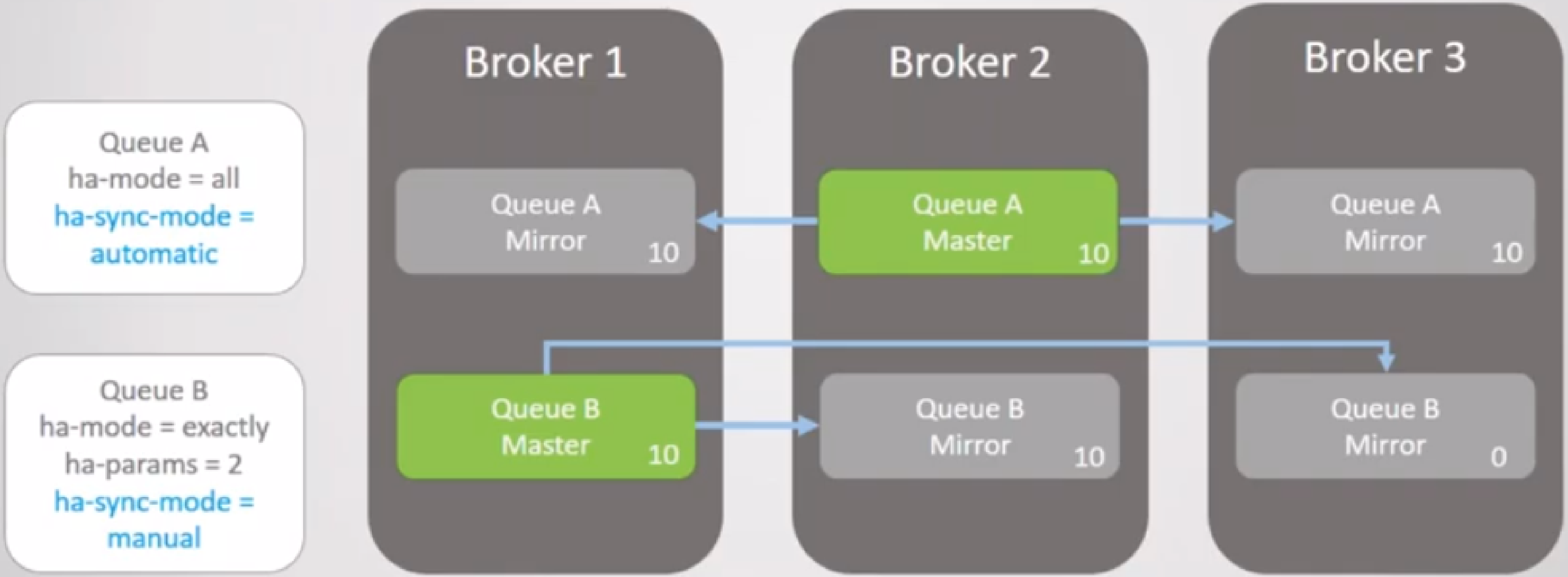


RabbitMQ - Queue Mirrors - Synchronization



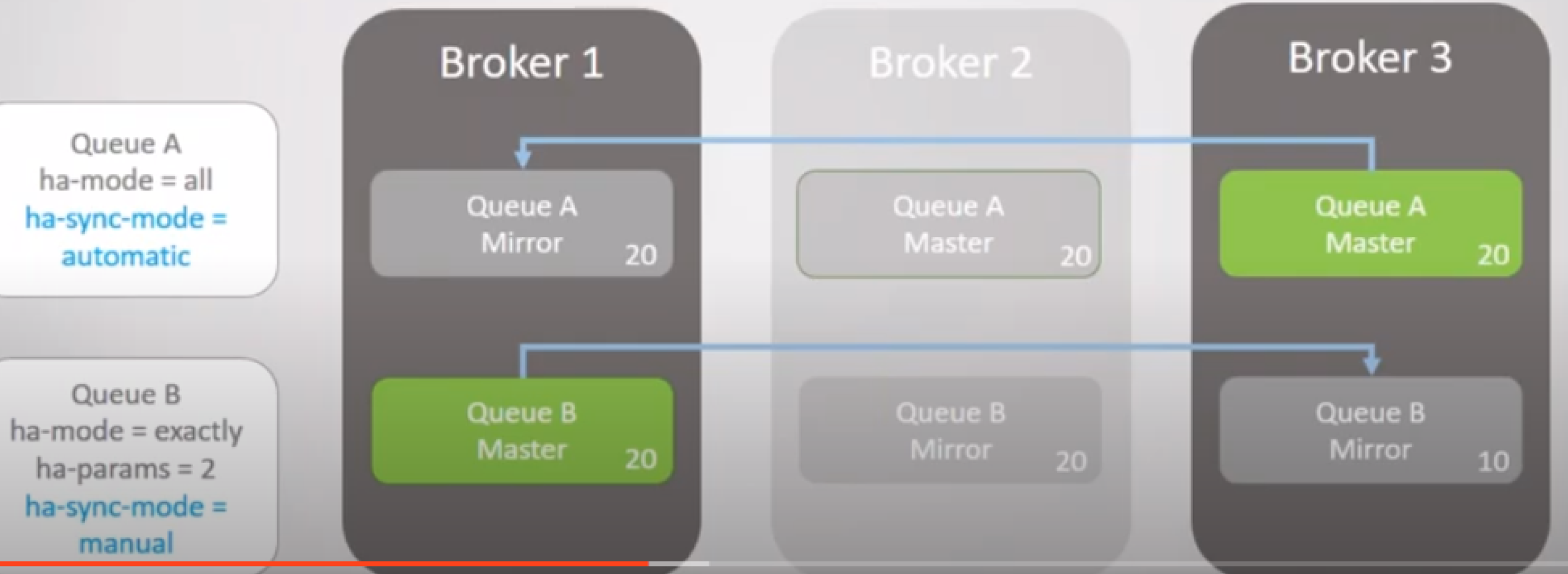
RabbitMQ - Queue Mirrors - Synchronization

Broker 3 comes back.
Mirror A is automatically synchronized. Mirror B remains at 0 messages.



RabbitMQ - Queue Mirrors - Synchronization

Each queue receives 10 more messages.
Broker 2 is lost. Queue A fails over to mirror 3 without data loss.

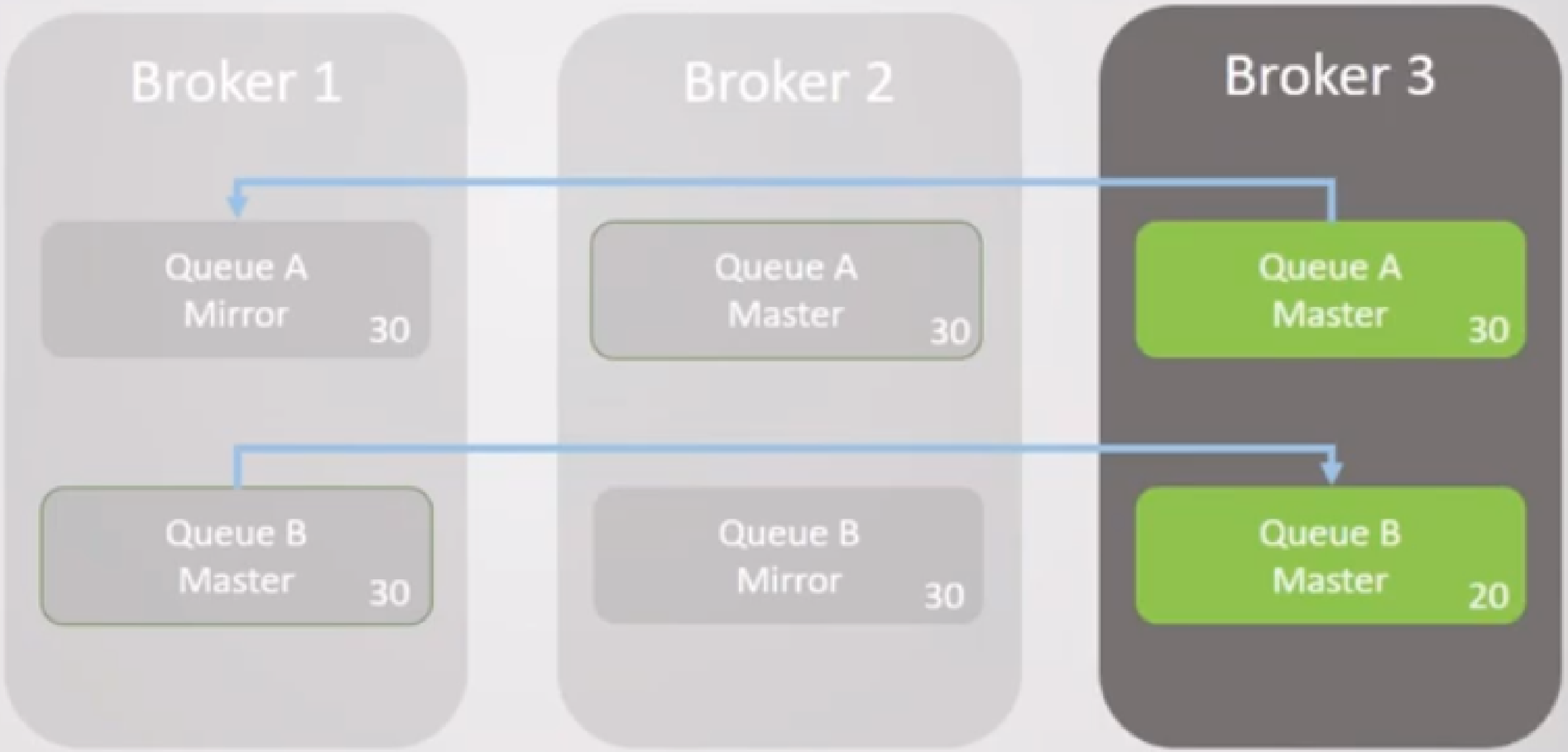


RabbitMQ – Queue Mirrors - Synchronization

Each queue receives 10 more messages.
Broker 1 is lost. Queue B fails over to mirror 3 and loses 10 messages.

Queue A
ha-mode = all
ha-sync-mode = automatic

Queue B
ha-mode = exactly
ha-params = 2
ha-sync-mode = manual
ha-promote-on-failure = always

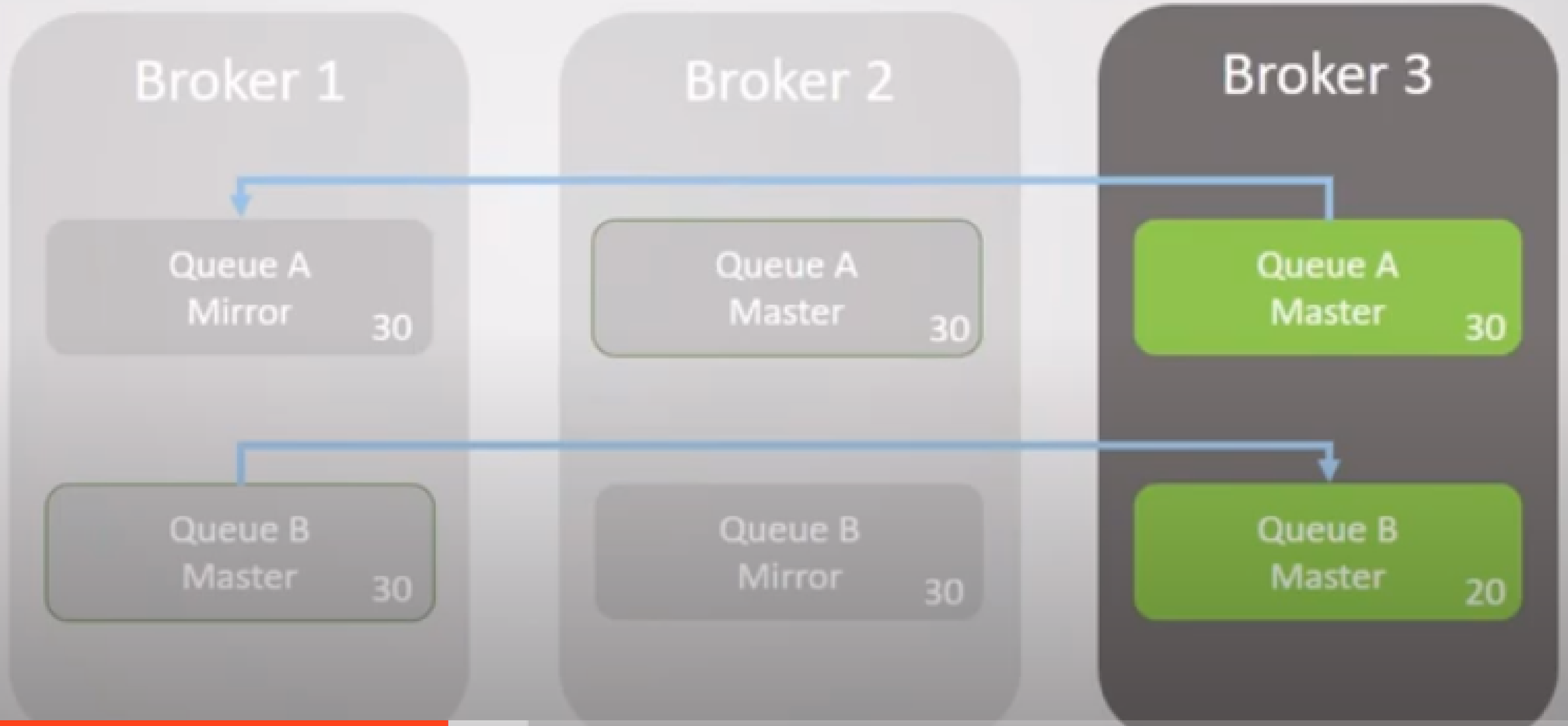


RabbitMQ - Queue Mirrors - Synchronization

Each queue receives 10 more messages.
Broker 1 is lost. Queue B fails over to mirror 3 and loses 10 messages.

Queue A
ha-mode = all
ha-sync-mode = automatic

Queue B
ha-mode = exactly
ha-params = 2
ha-sync-mode = manual
ha-promote-on-failure = always

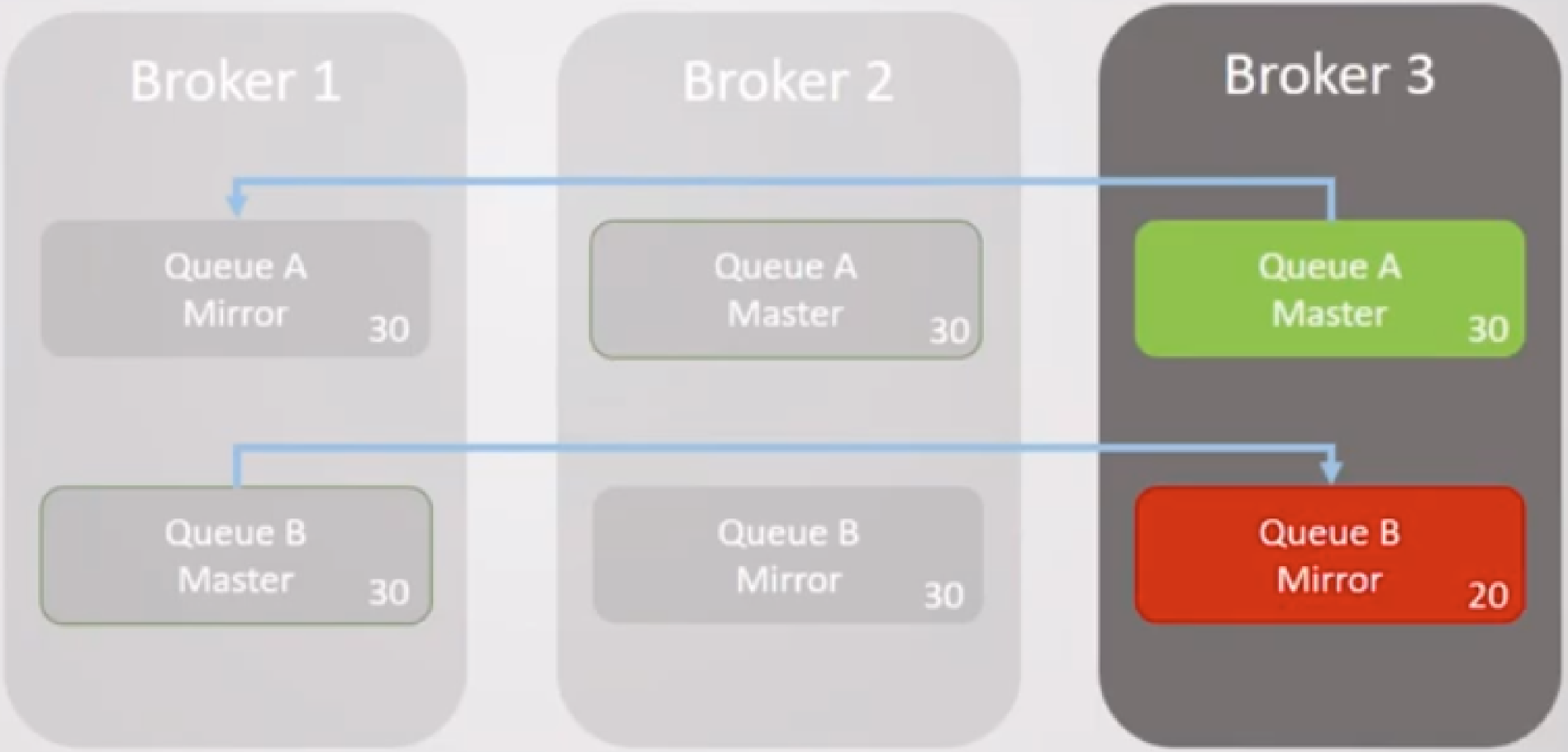


RabbitMQ - Queue Mirrors - Synchronization

Alternate scenario: ha-promote-on-failure = when-synced
Queue B does not fail over as mirror 3 is unsynchronized.

Queue A
ha-mode = all
ha-sync-mode = automatic

Queue B
ha-mode = exactly
ha-params = 2
ha-sync-mode = manual
ha-promote-on-failure = when-synced



RabbitMQ

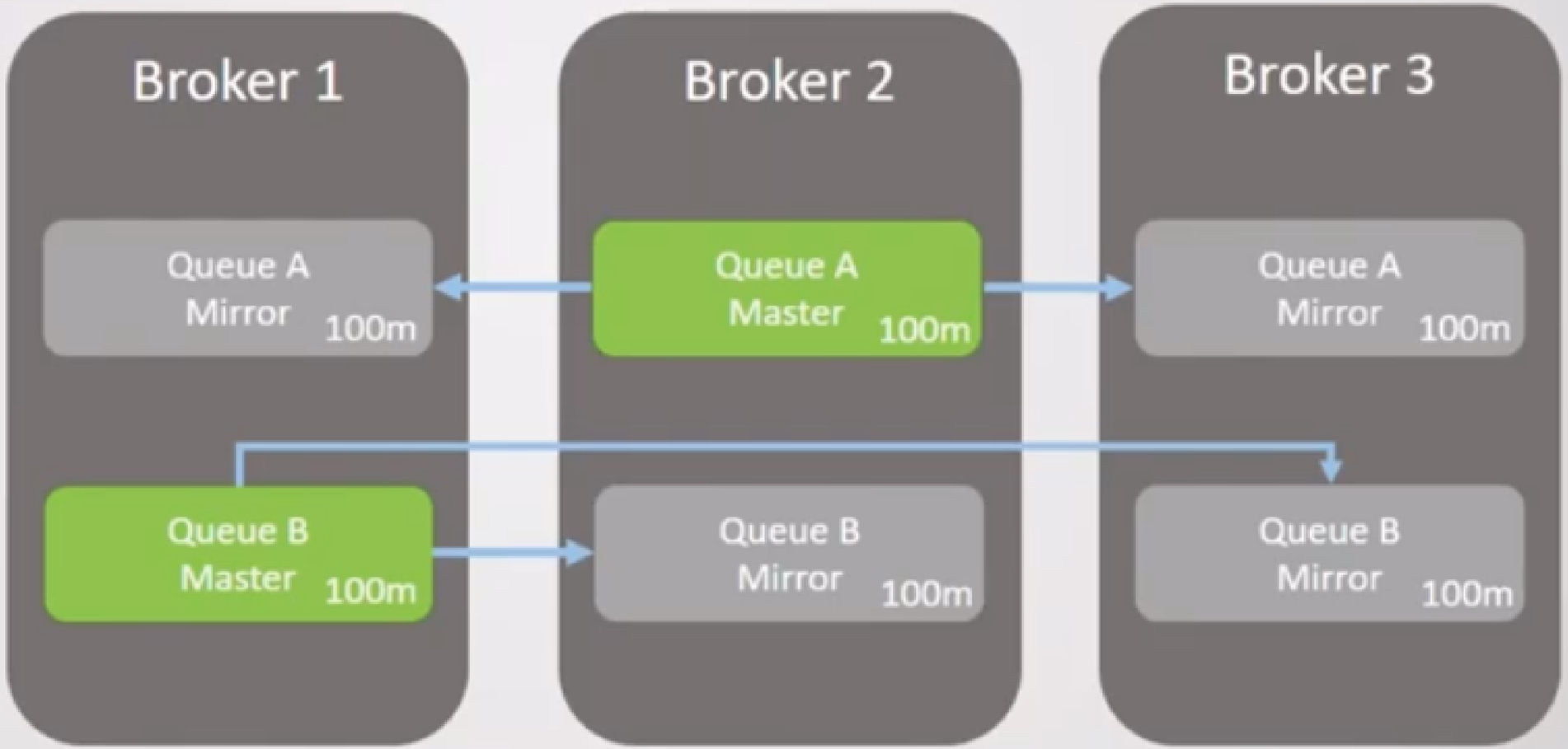
Queue Mirror
Synchronization
And
New Mirrors

RabbitMQ - Queue Mirrors - Synchronization

Three nodes, two mirrored queues each with 100 million messages

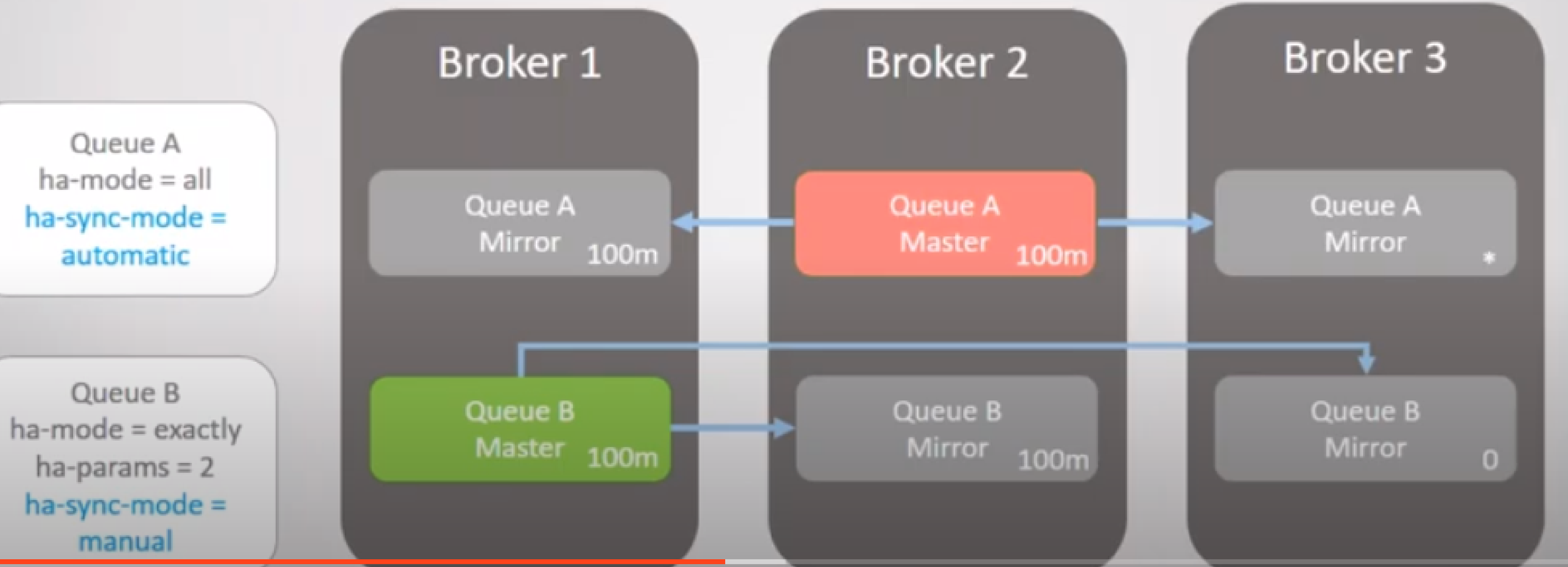
Queue A
ha-mode = all
ha-sync-mode = automatic

Queue B
ha-mode = exactly
ha-params = 2
ha-sync-mode = manual



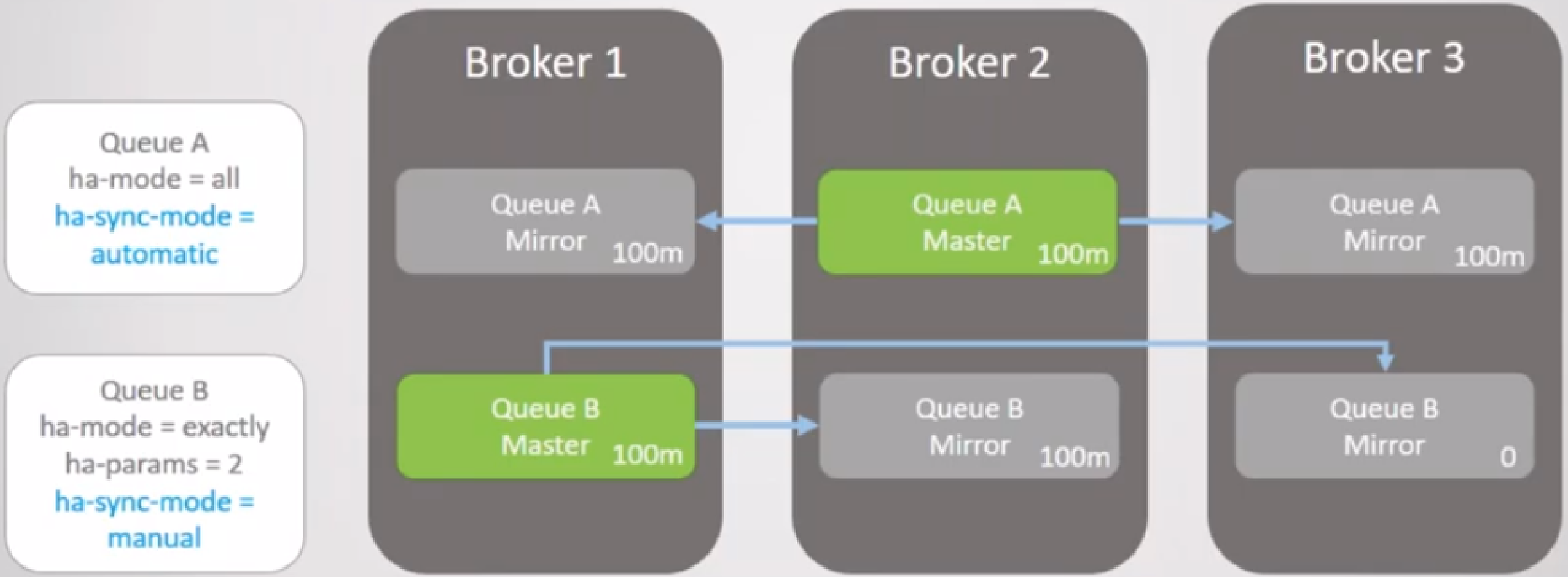
RabbitMQ - Queue Mirrors - Synchronization

Broker 3 comes back.
Queue A is unavailable due to synchronization.
Queue B is available but mirror on broker 3 remains at 0 messages.



RabbitMQ - Queue Mirrors - Synchronization

Queue A synchronization completes and the queue becomes available again.



RabbitMQ

Optimizing for High Throughput

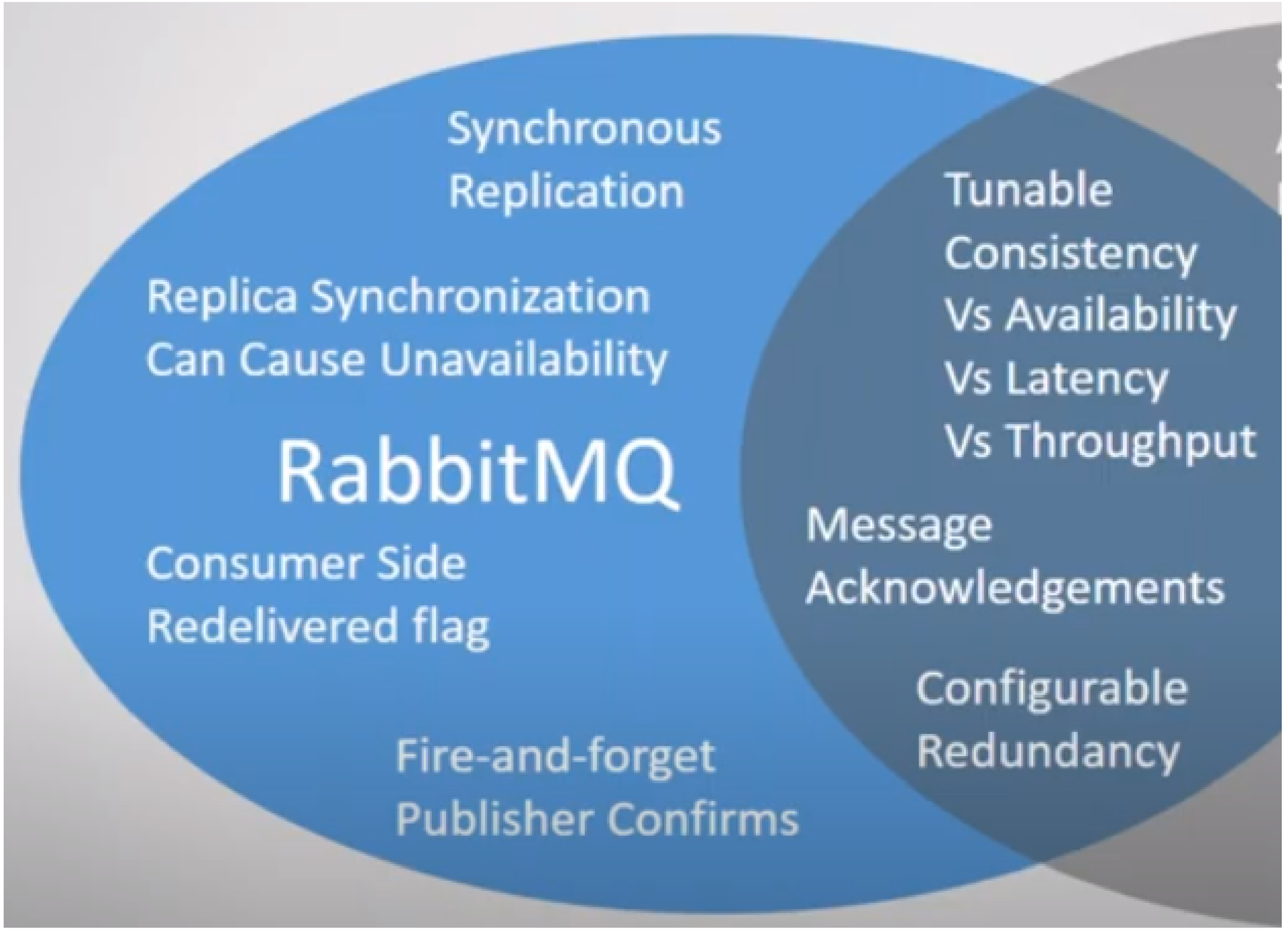
- Producers fire and forget
- Consumers use auto-ack mode
- Non-Persistent messages
- Non-Mirrored Queues
- Cluster for throughput

Balancing Data Safety with High Throughput

- Producers wait periodically for acknowledgements
- Consumers group acknowledgements with the Multiple flag
- Persistent messages
- Cluster
- Queue mirroring with 1 mirror*

Optimizing for Data Safety

- Producers wait for acknowledgements after each message or after small number of messages
- Consumers acknowledge each message individually
- Persistent messages
- Cluster for durability
- Queue mirroring with 2+ mirrors.
- promote-on-failure=when-synced
- ha-sync-mode=automatic for active queues


Synchronous
Replication
Replica Synchronization
Can Cause Unavailability
RabbitMQ
Consumer Side
Redelivered flag
Fire-and-forget
Publisher Confirms
Tunable
Consistency
Vs Availability
Vs Latency
Vs Throughput
Message
Acknowledgements
Configurable
Redundancy