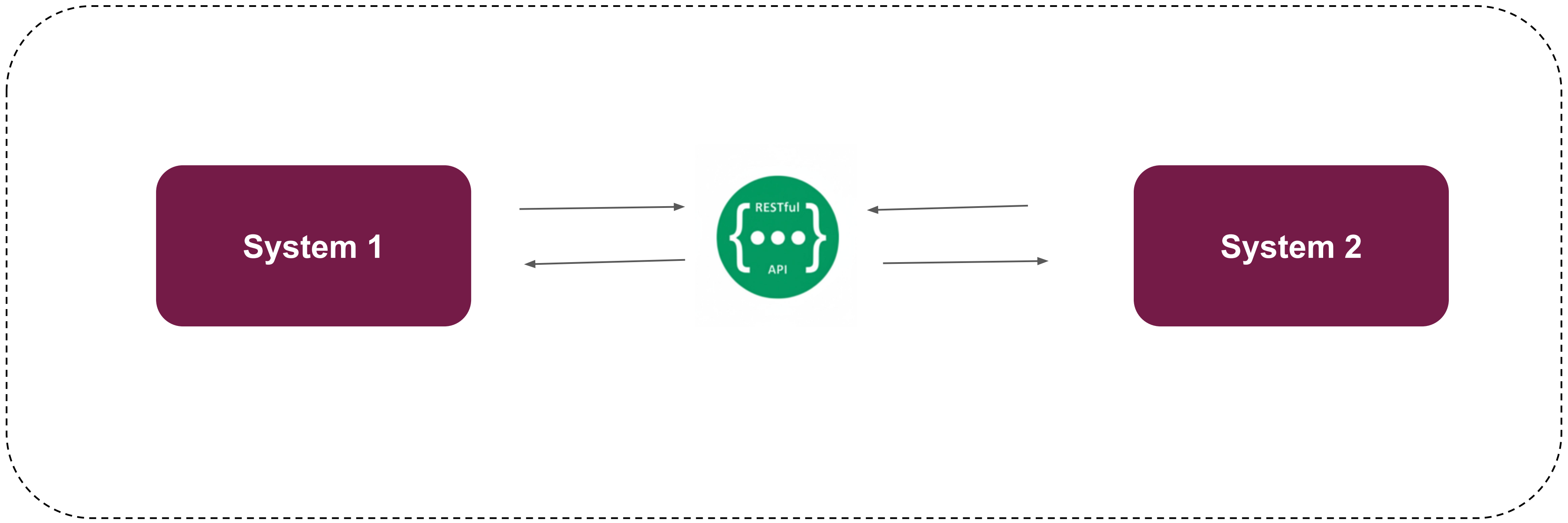
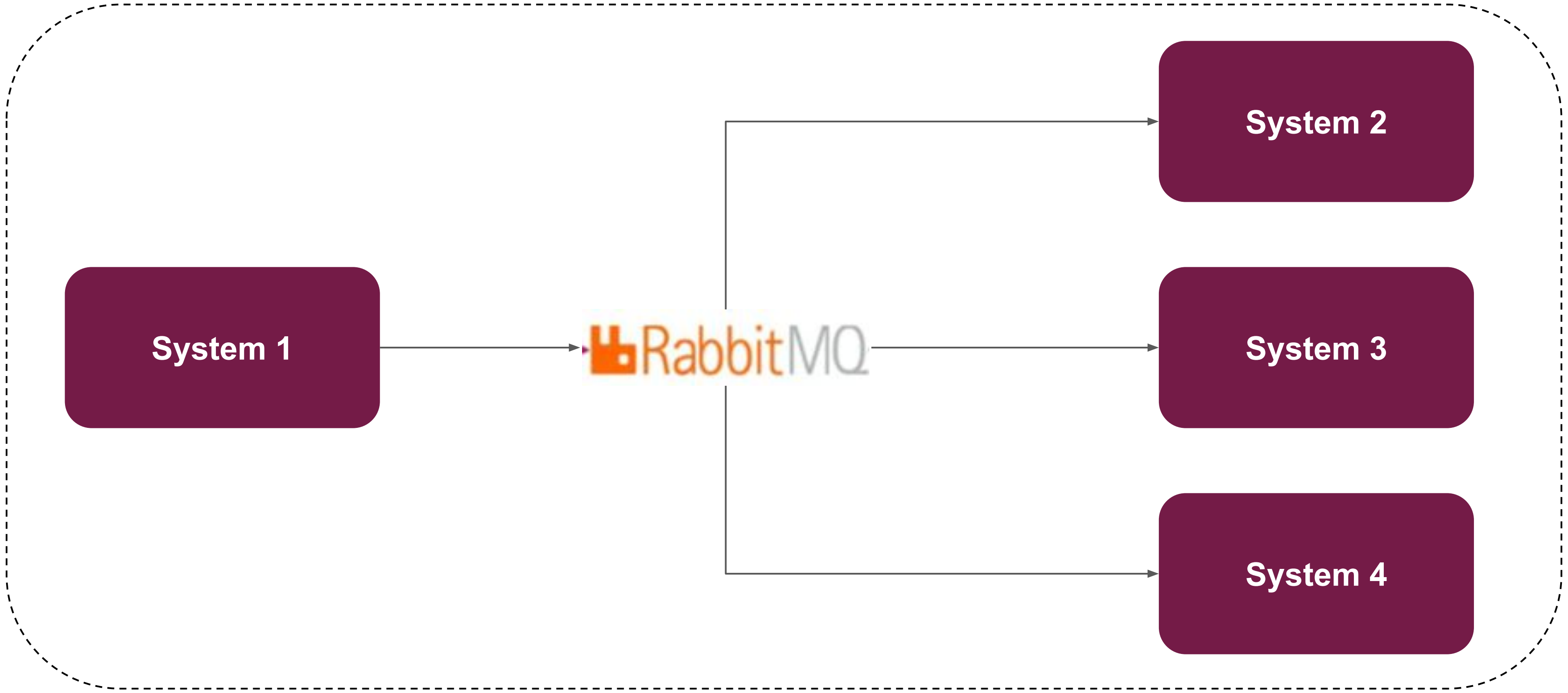
RabbitMQ

ENG. ISMAIL ANJRINI
ENG. Mouaz Alsayyad (writer)

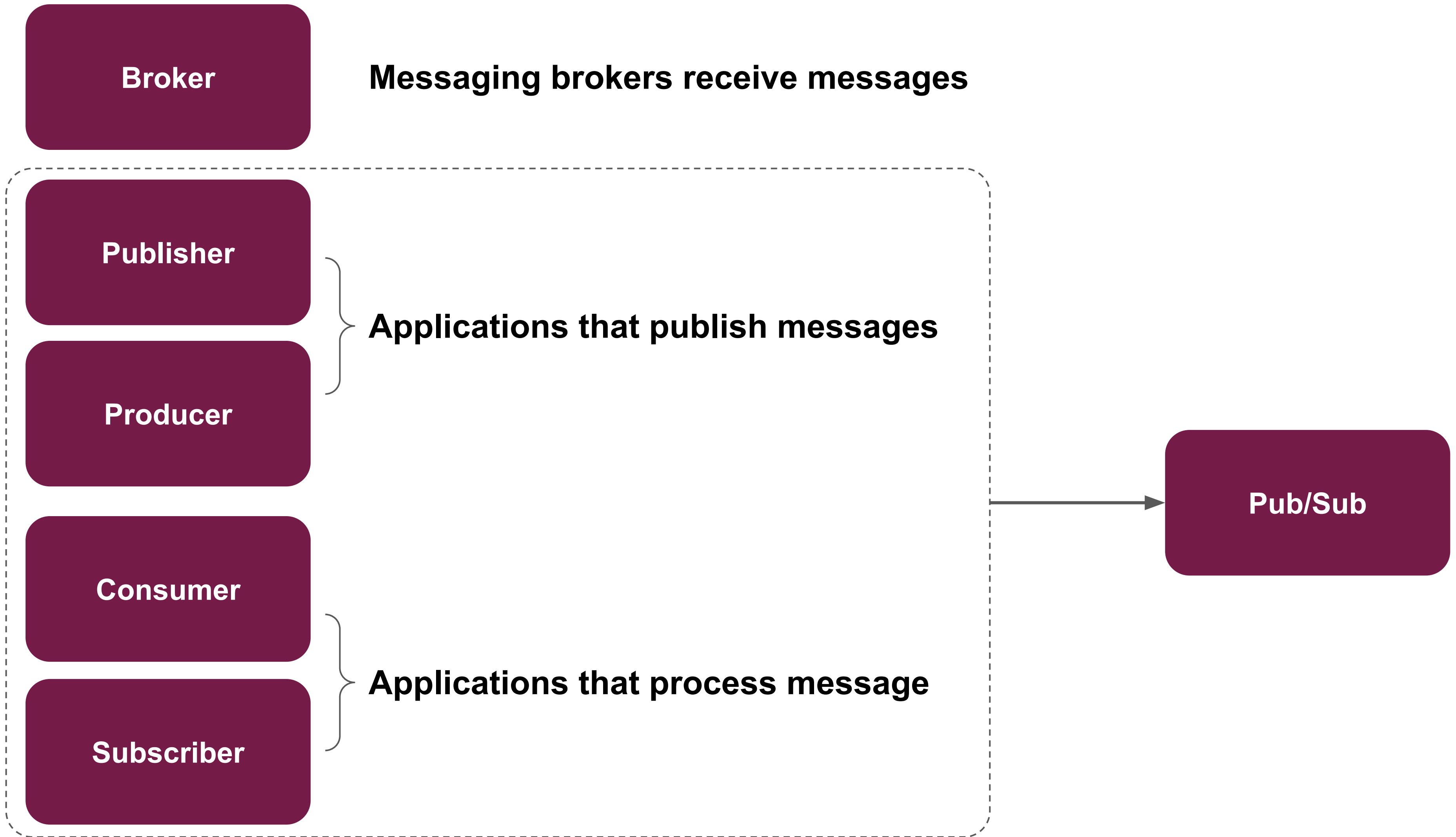
Sync

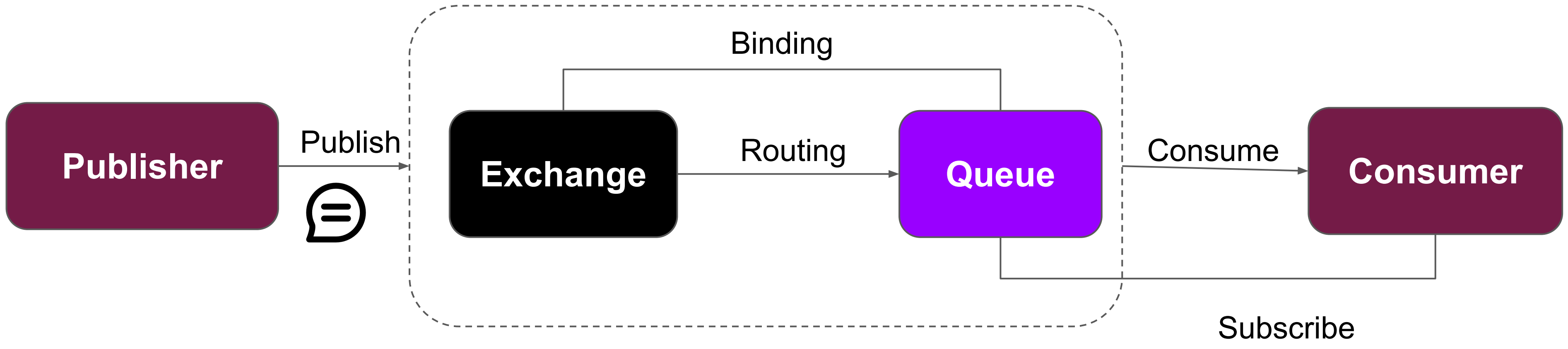


Async **Event Driven**



- RabbitMQ is the most widely deployed open source message broker
- RabbitMQ is lightweight and easy to deploy on premises and in the cloud
- RabbitMQ can bc deployed in distributed and fodcratod configurations to meet high-scale, high-availability requirements
- Asynchronous Messaging
- Develop cross-language messaging with favorite programming languages
- Management Monitoring
- OSS vs Commercial


Broker
Messaging brokers receive messages
Publisher
Producer
Applications that publish messages
Consumer
Subscriber
Applications that process message
Pub/Sub


Publisher
Publish
Binding
Exchange
Routing
Queue
Consume
Consumer
Subscribe

Create Exchange

Name

Type

Exchange Type

direct fanout headers topic

direct: The direct exchange will route the message to a queue whose binding key matches the routing key of the message exactly. So if you bind a queue to a direct exchange with the binding key “blue”, all messages published to that exchange with the routing key “blue” will end up in that queue. One queue can have multiple bindings to the same exchange with different binding keys.

topic: It is similar to the direct exchange in that it will route messages where the routing key matches the binding key from the queue binding. However, with a topic exchange, you can also use wildcards in the binding key. When using a topic exchange the routing key of the message must be a list of words separated by dots, like “metrics.server.cpu”. The reason is that topic exchange allows you to match on parts of the routing key and uses dots as separators.

fanout: The fanout exchange will route published messages to any queues bound to it without any conditions and no balancing between the queues.

headers: A headers exchange routes messages based on arguments containing headers and optional values. Headers exchanges are very similar to topic exchanges, but route messages based on header values instead of routing keys. A message matches if the value of the header equals the value specified upon binding.

Create Exchange

Durability	Durable Transient Durable: Survive restarts; auto-redeclared Transient: Lost on restart; must redeclare
Auto delete	Auto delete: Automatic cleanup, reduces orphaned exchanges
Internal	
Argurnenls	Argurnenls: alternale-exchange

Exchange Types

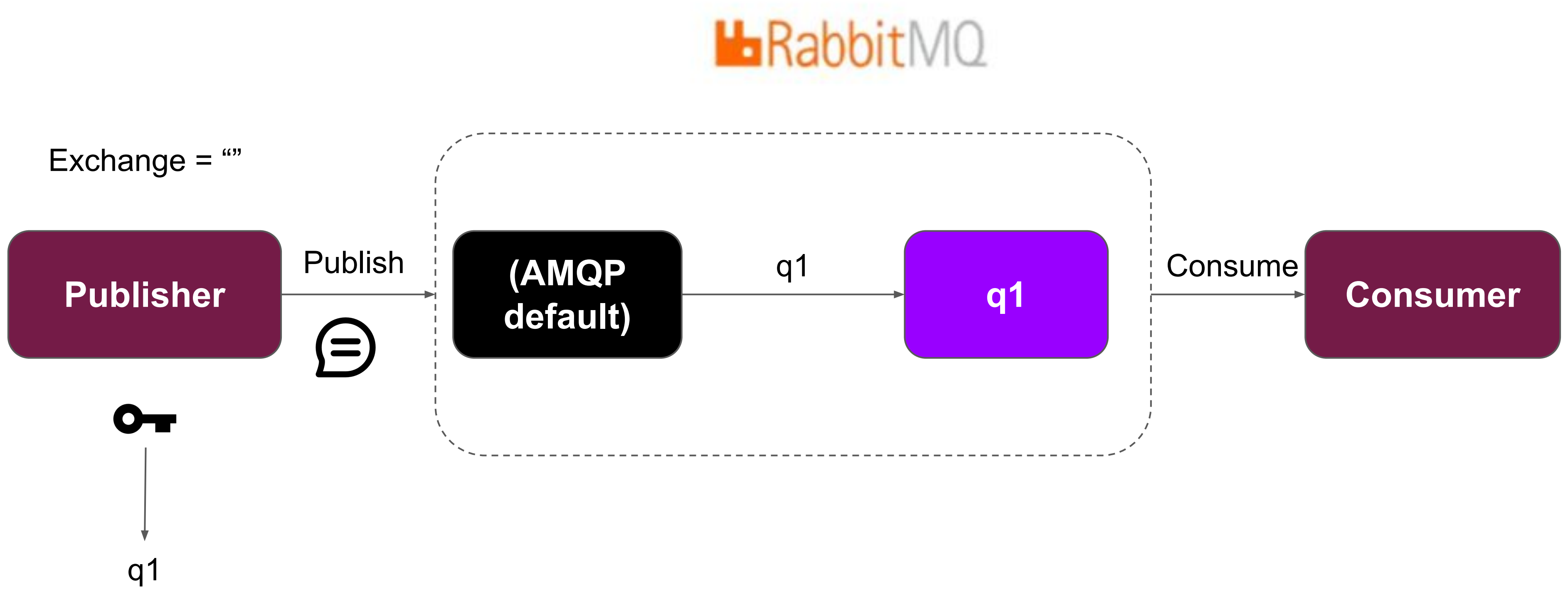
Default Exchange

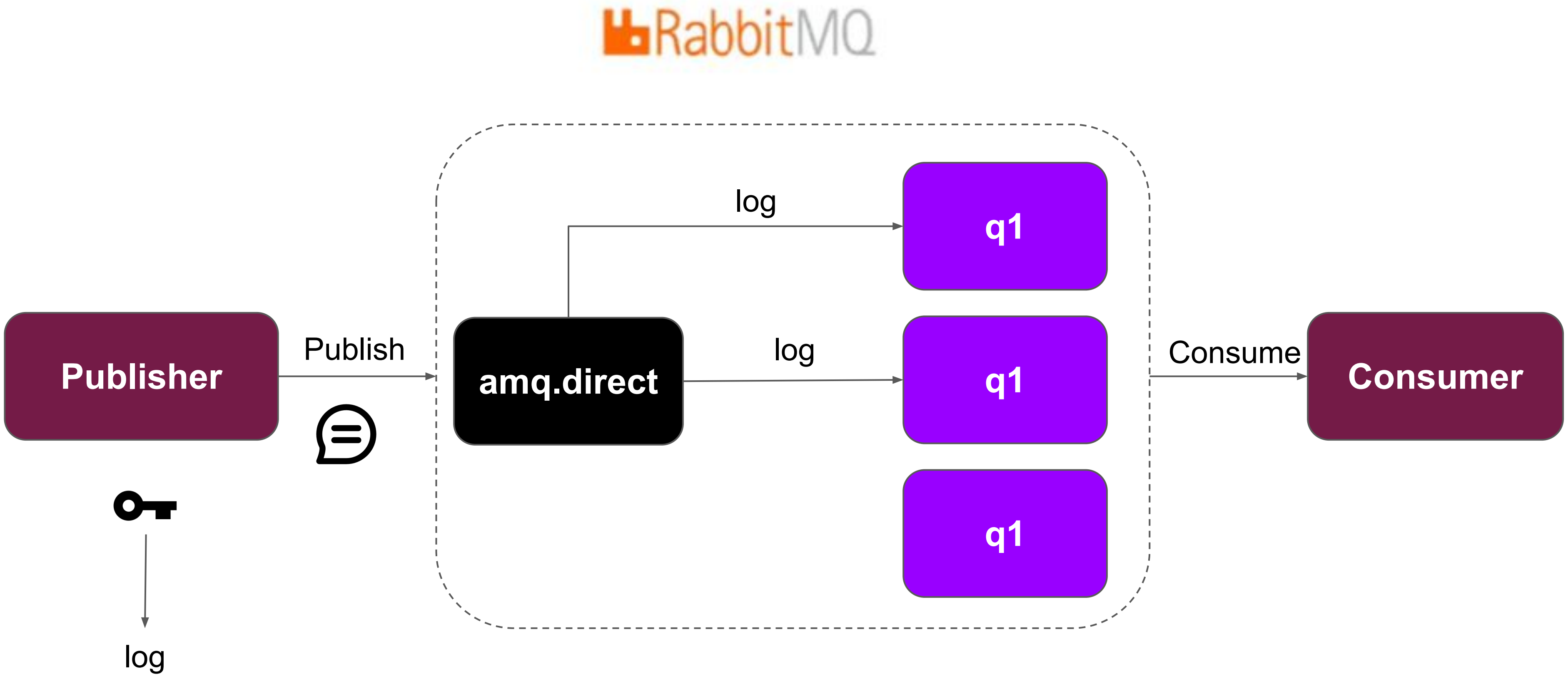
Direct Exchange

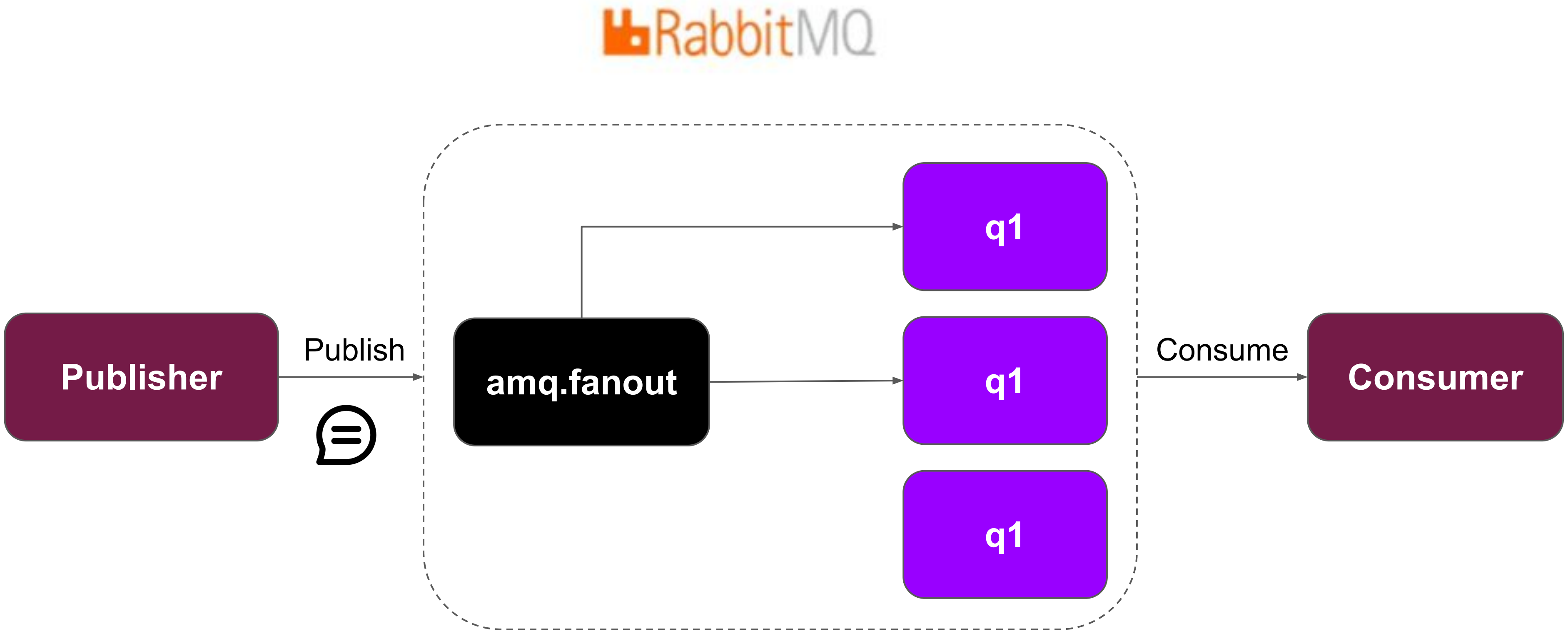
Fanout Exchange

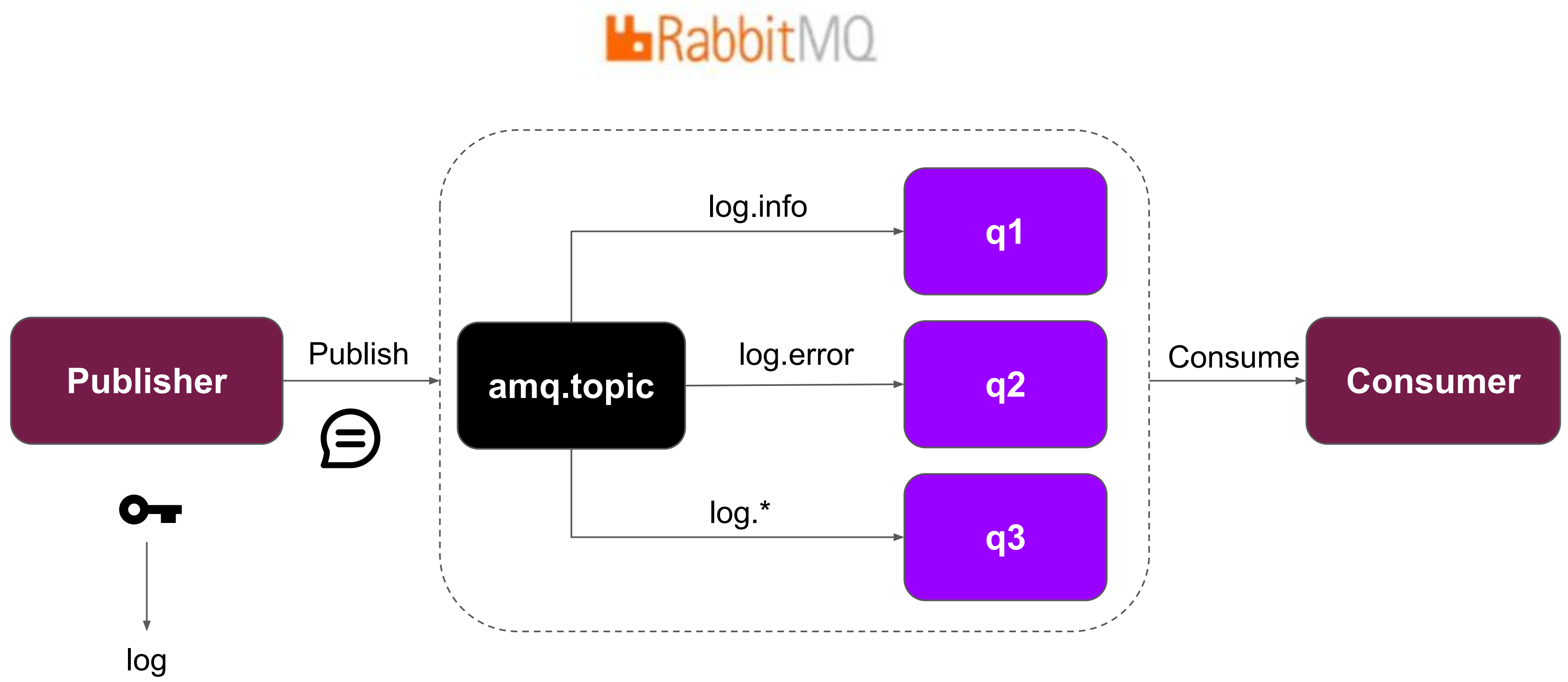
Topic Exchange

Headers Exchange


RabbitMQ
Exchange = “”
Publisher
Publish
(AMQP default)
q1
q1
Consume
Consumer
q1

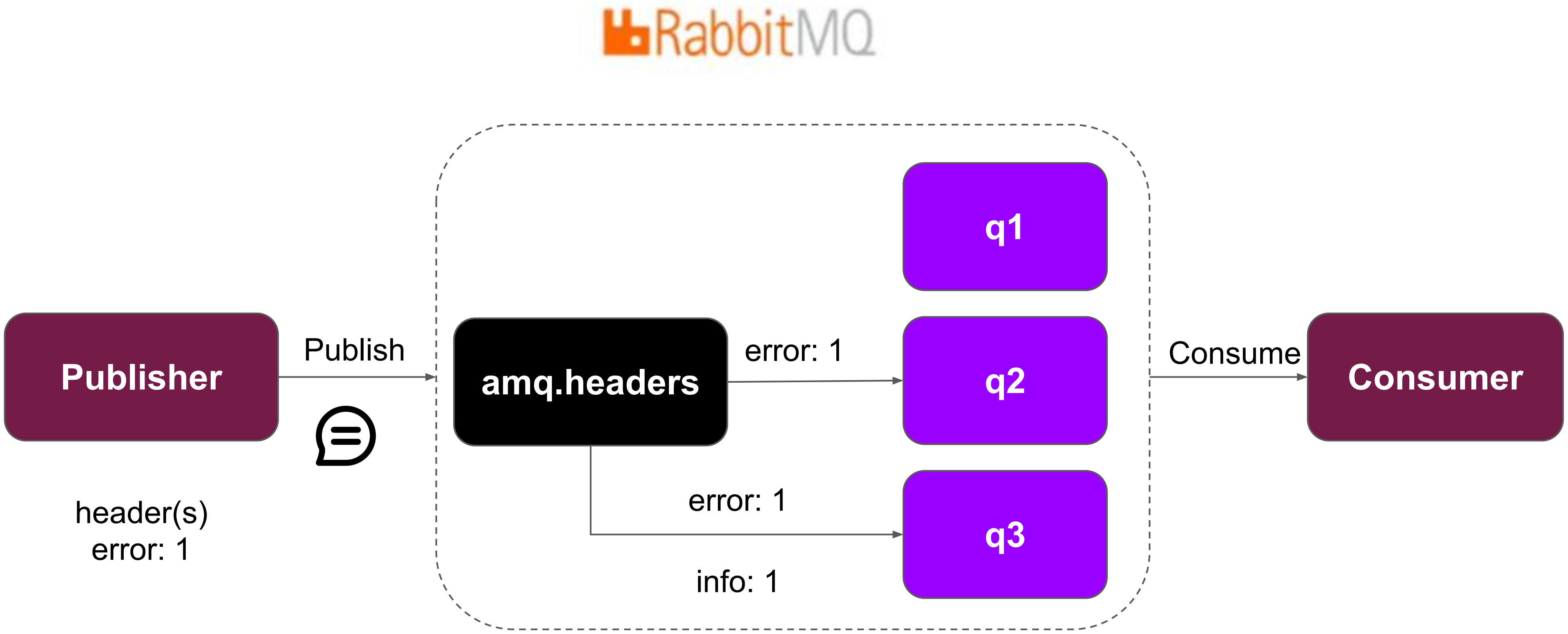

RabbitMQ
Publisher
Publish
amq.direct
log
q1
log
q1
q1
Consume
Consumer
log


RabbitMQ
Publisher
Publish
amq.fanout
q1
q1
q1
Consume
Consumer


RabbitMQ
Publisher
Publish
log
amq.topic
log.info
log.error
log.*
q1
q2
q3
Consume
Consumer

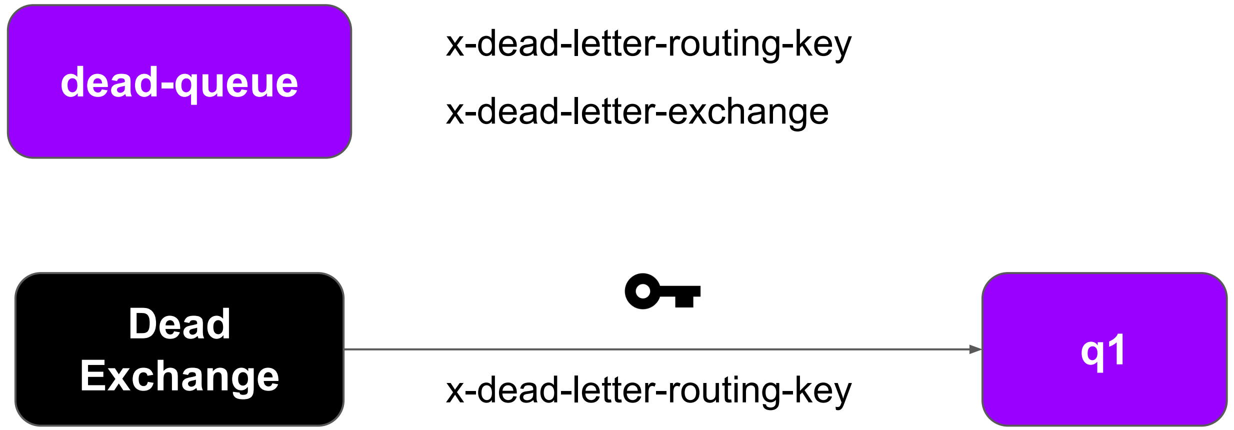
Header Exchange

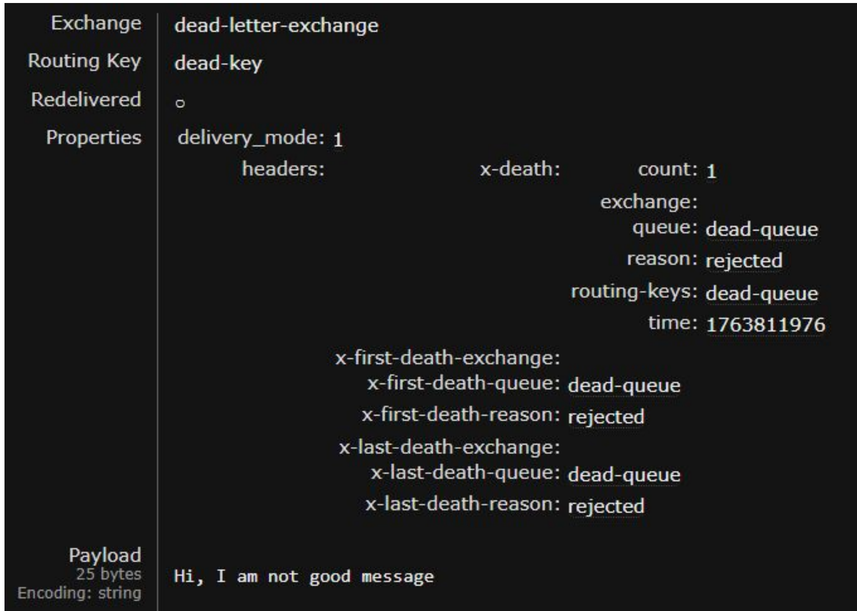
x-match	any	all
	any-with-x	all-with-x

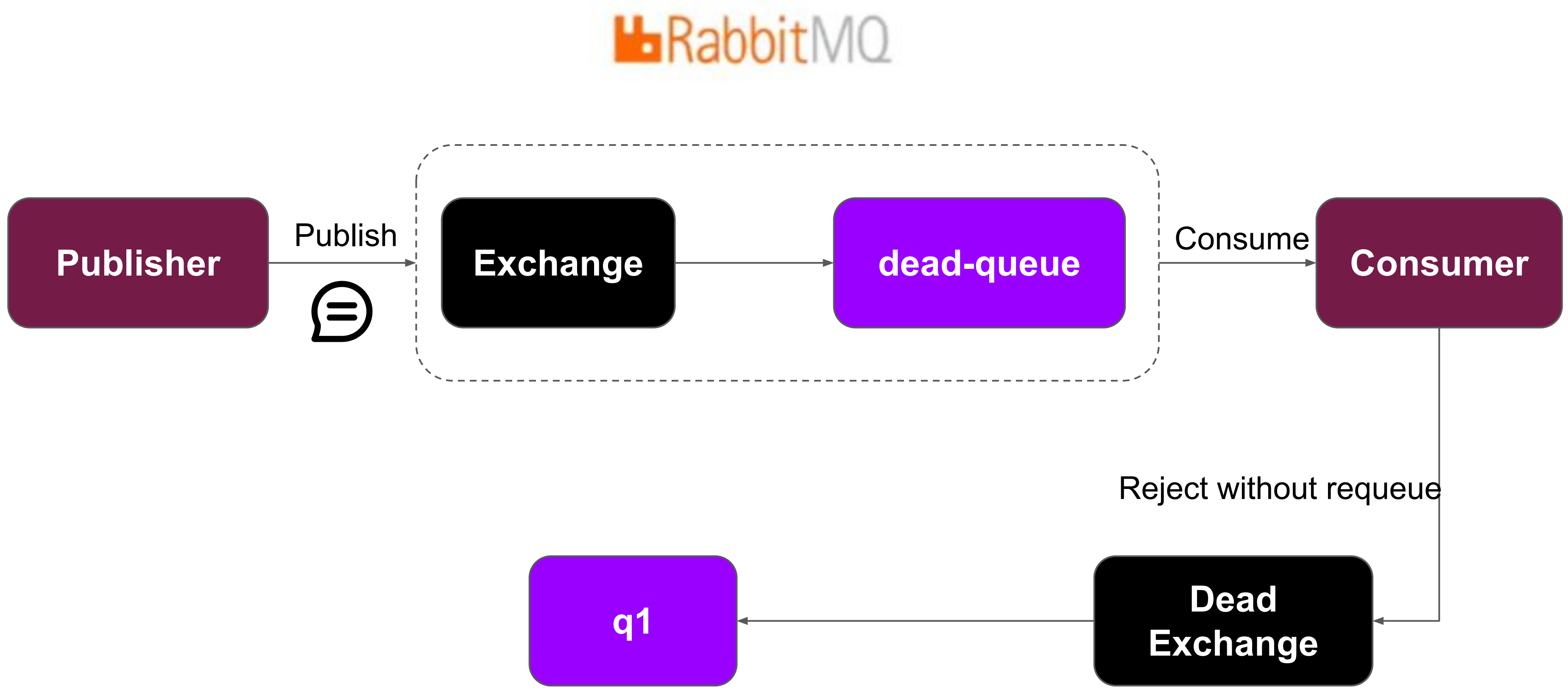

RabbitMQ
Publisher
Publish
amq.headers
error: 1
error: 1
info: 1
q1
q2
q3
Consume
Consumer
header(s)
error: 1

Create Queue

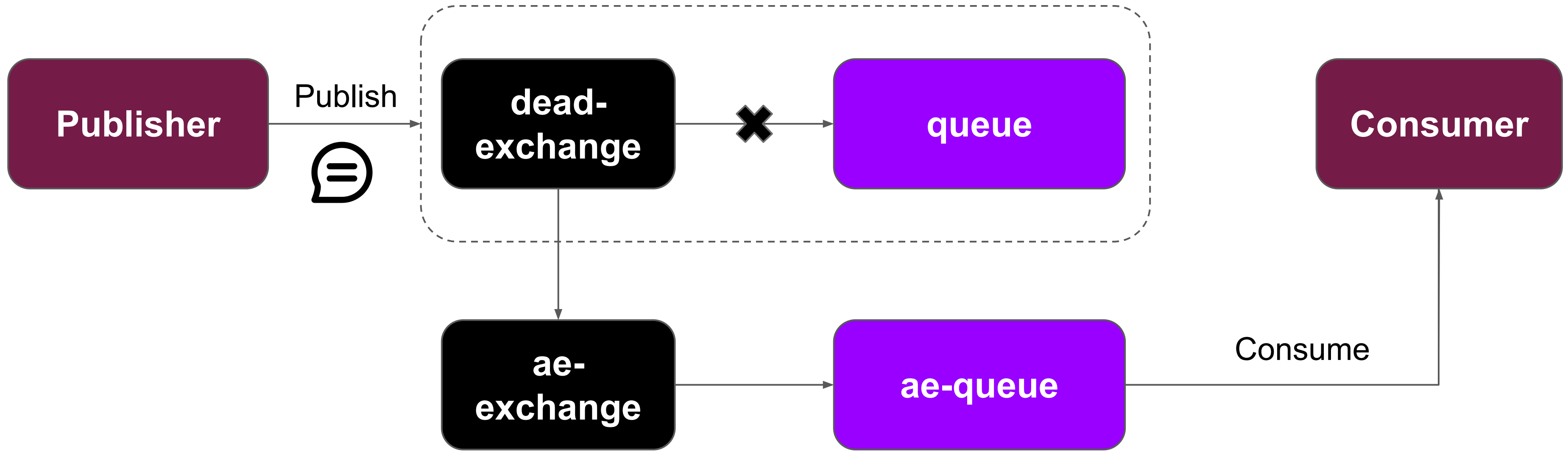
Name			
Exclusive	Only one connection can use it, and it auto-deletes when that connection ends.		
Durability	Durable	Transient	
Auto delete			
Arguments	x-expires	x-message-ttl	x-max-length
	x-dead-letter-routing-key	x-dead-letter-exchange	



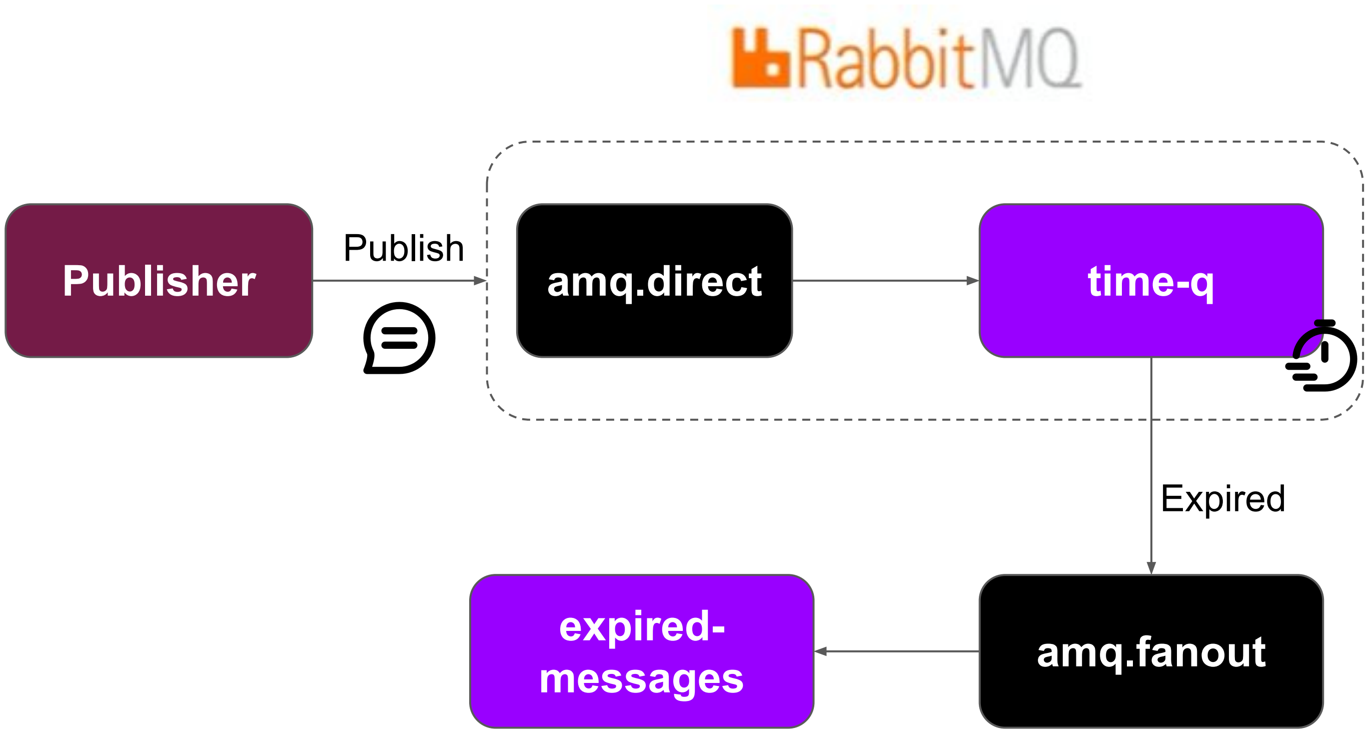

```
     Exchange   dead-letter-exchange
  Routing Key   dead-key
  Redelivered   ○
   Properties   delivery_mode: 1
                       headers:            x-death:         count: 1
                                                         exchange:
                                                            queue: dead-queue
                                                           reason: rejected
                                                     routing-keys: dead-queue
                                                             time: 1763811976
                              x-first-death-exchange:
                                 x-first-death-queue: dead-queue
                                x-first-death-reason: rejected
                               x-last-death-exchange:
                                  x-last-death-queue: dead-queue
                                 x-last-death-reason: rejected

      Payload
     25 bytes   Hi, I am not good message
Encoding: string
```


RabbitMQ
Publisher
Publish
Exchange
dead-queue
Consume
Consumer
Reject without requeue
Dead Exchange
q1

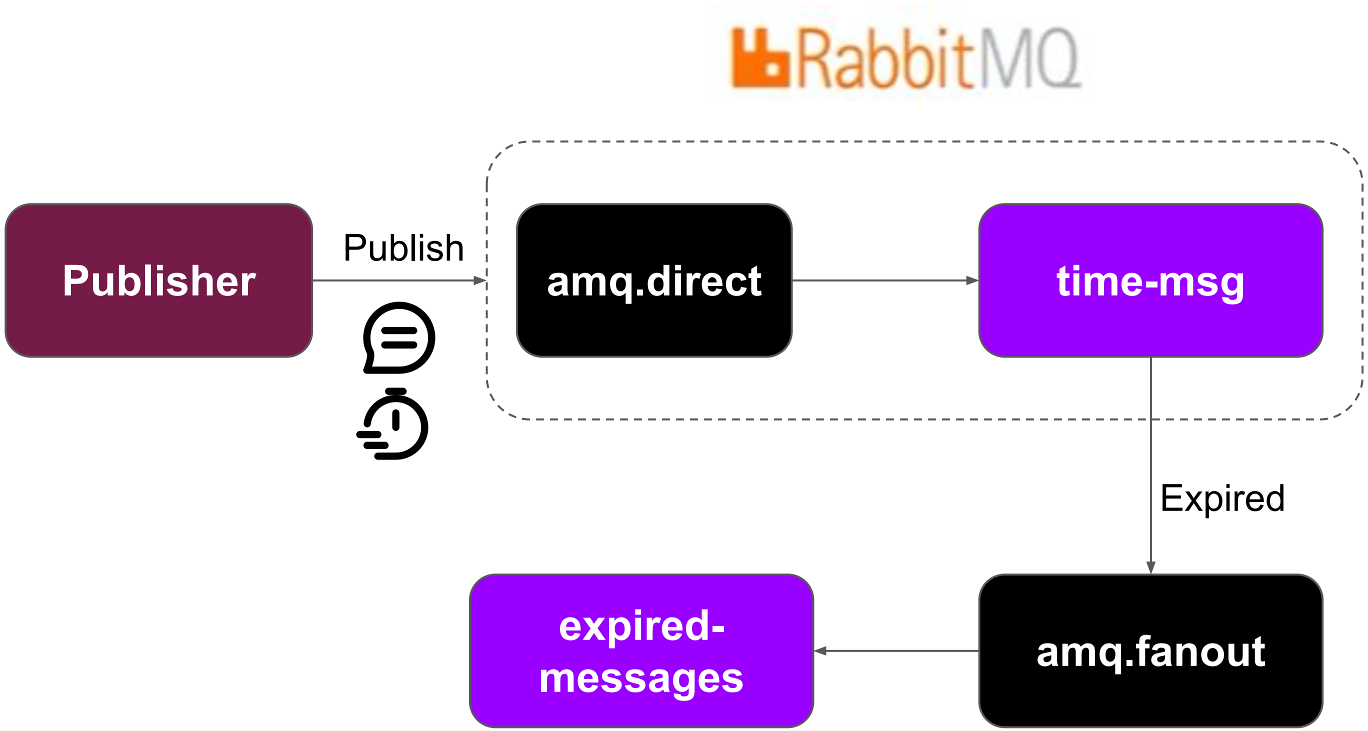
Alternate Exchange



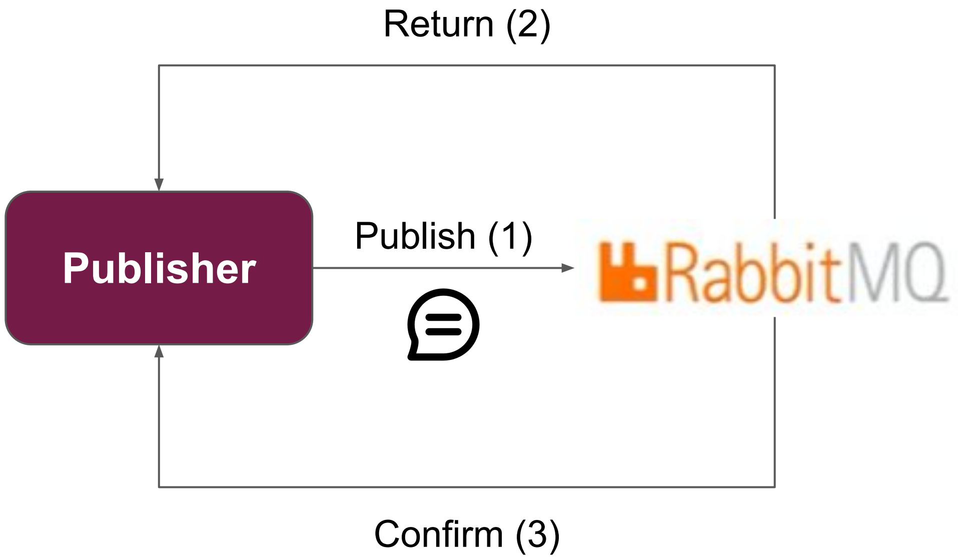
Message Expiry Time



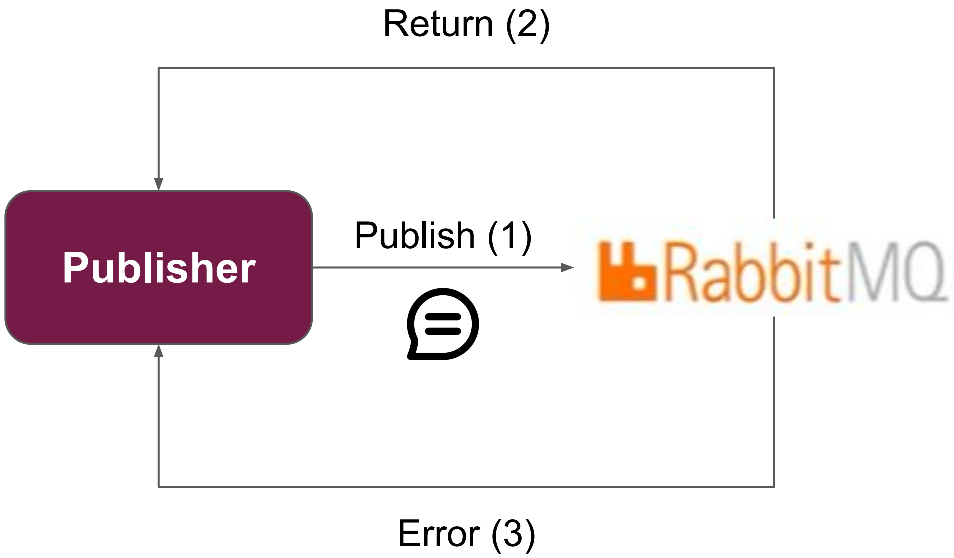
Message Expiry Time



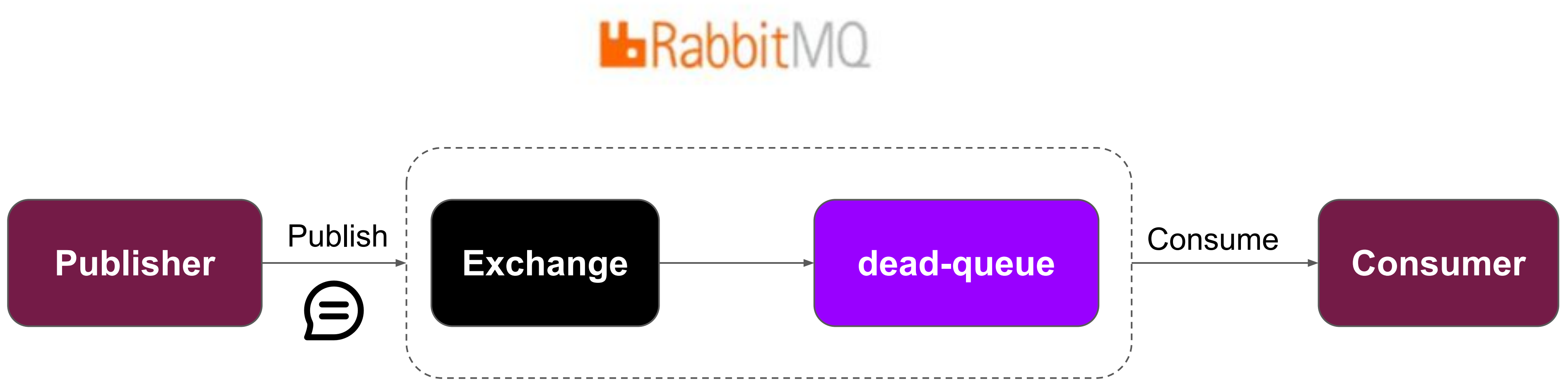
Reliable Publishing



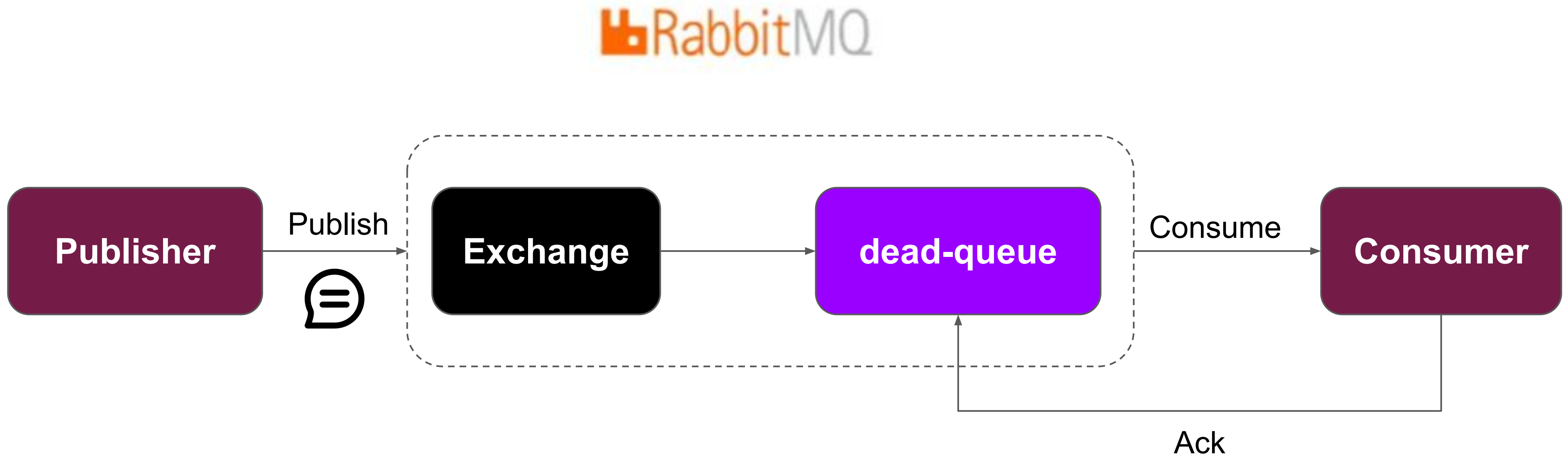
Reliable Publishing



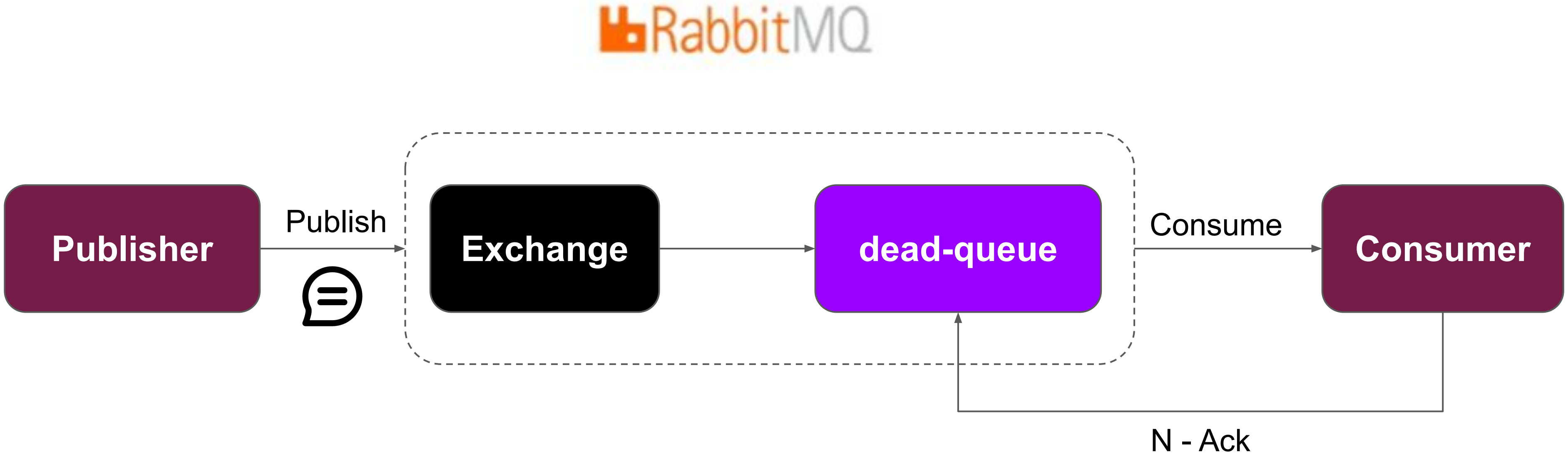
Auto Positive Ack



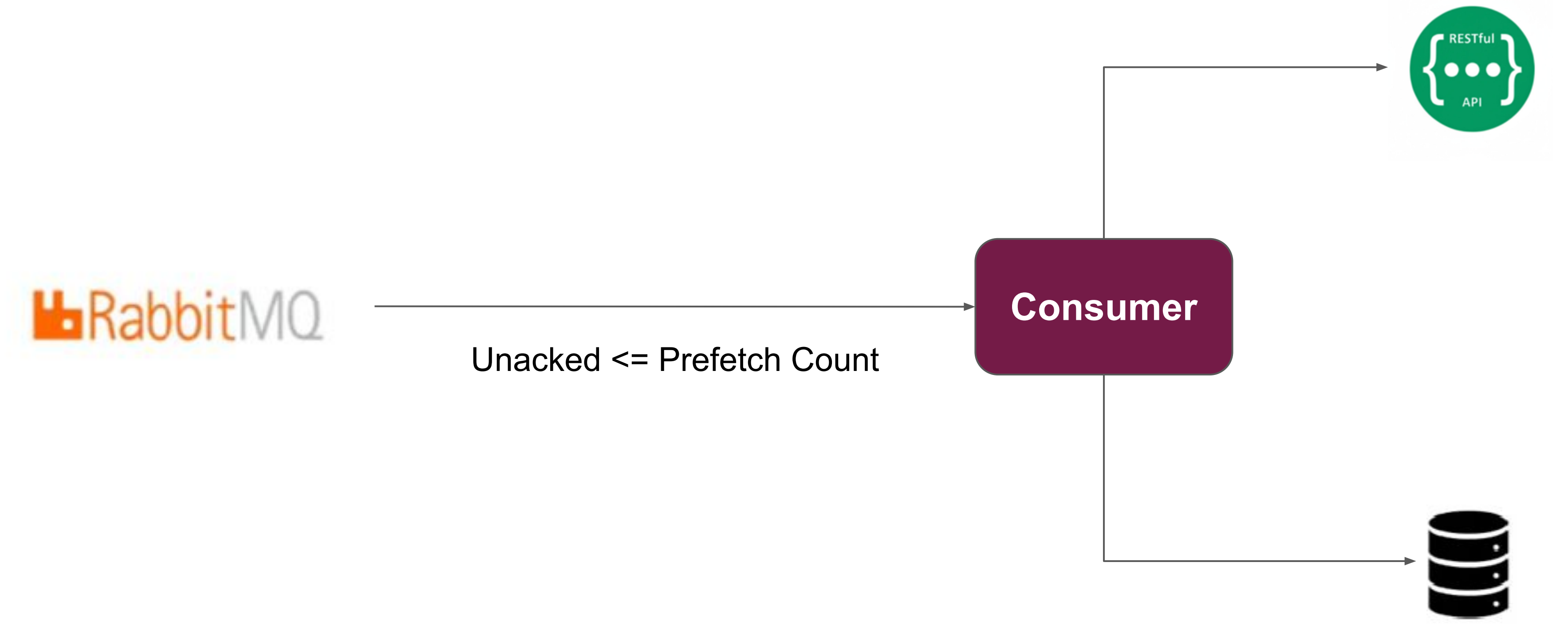
Manual Positive Ack



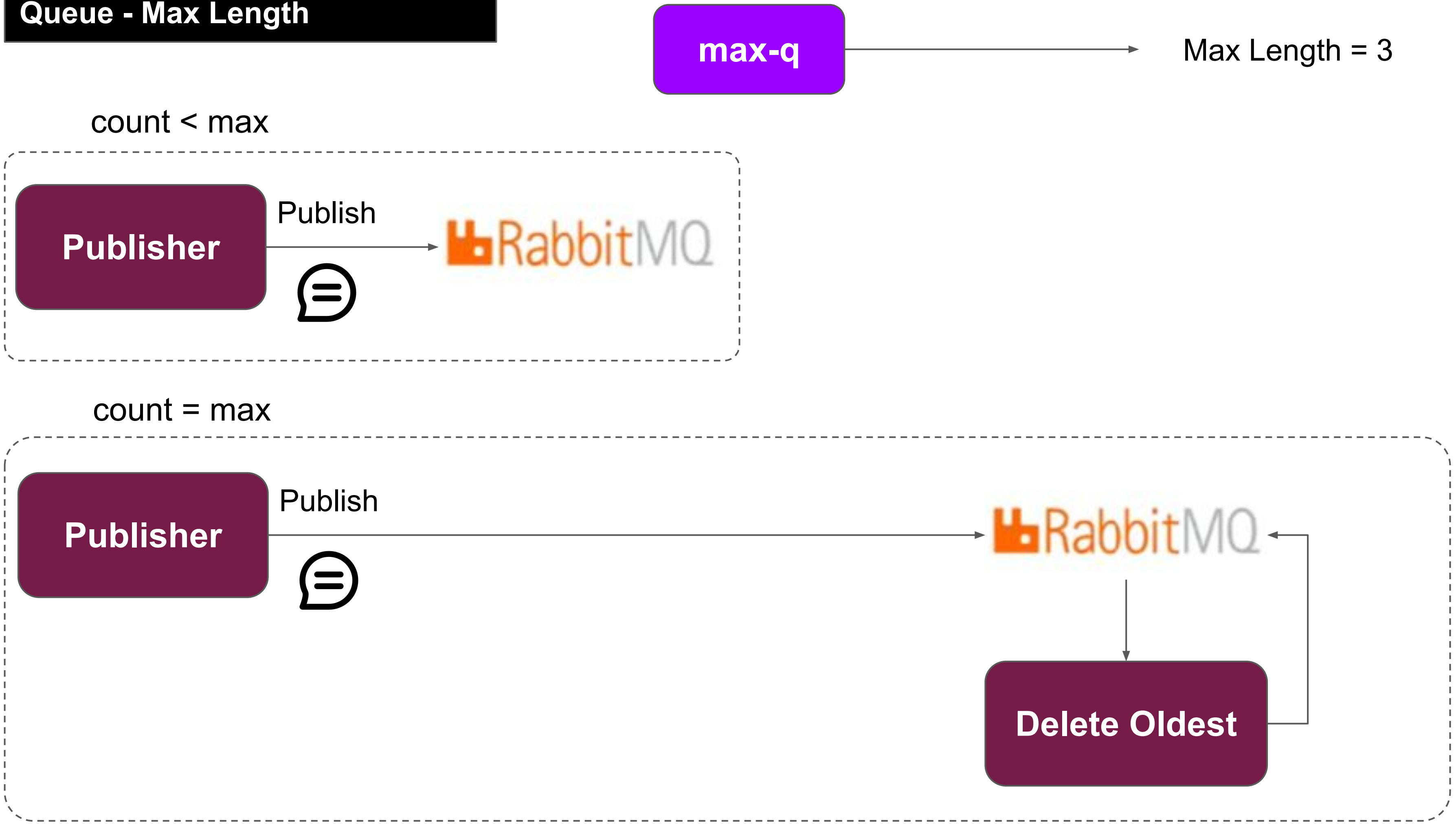
Negative Ack



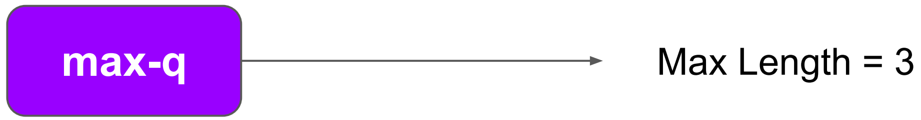
Consumer Prefetch

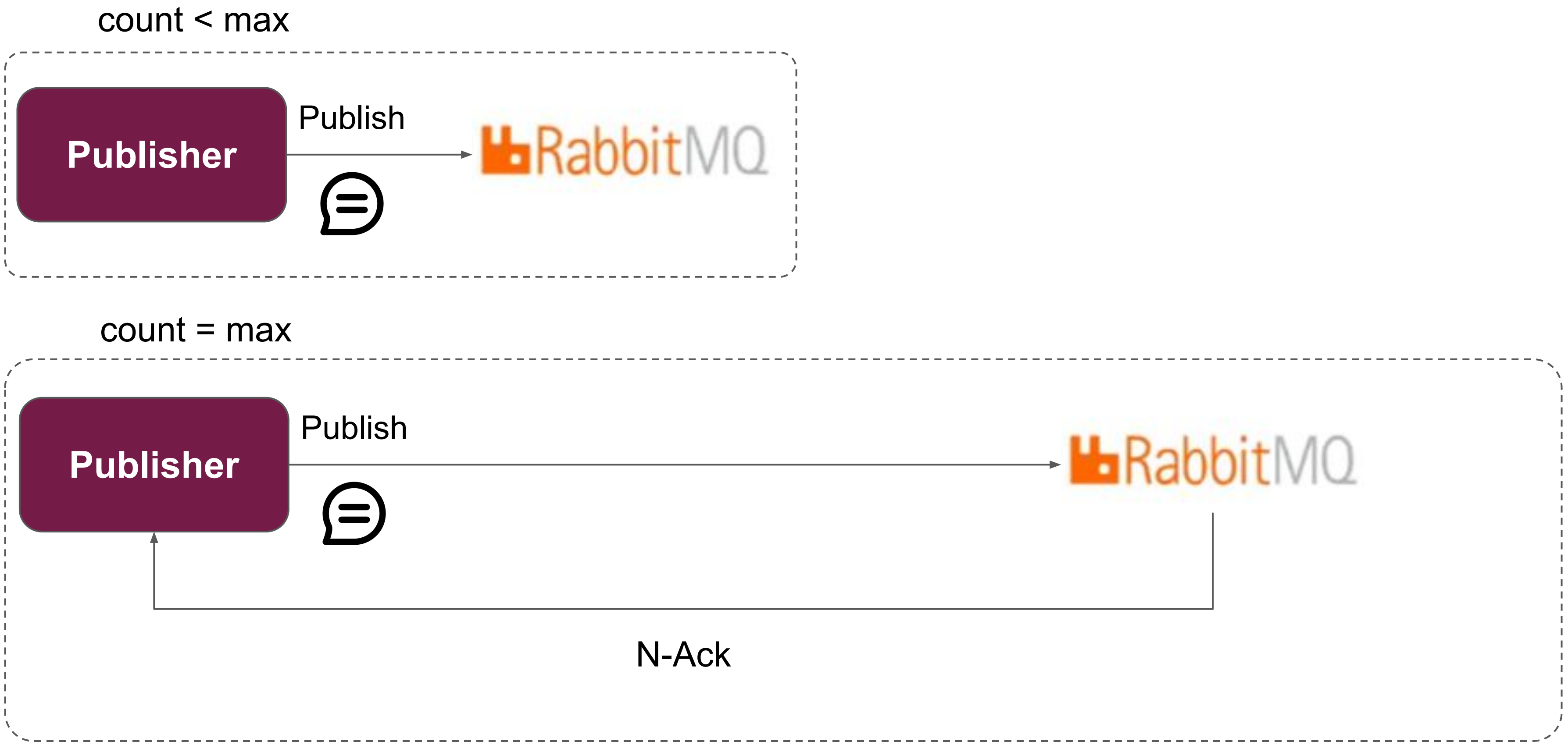


Queue - Max Length



Queue - Max Length - Overflow





Consumer Priorities

Priorities

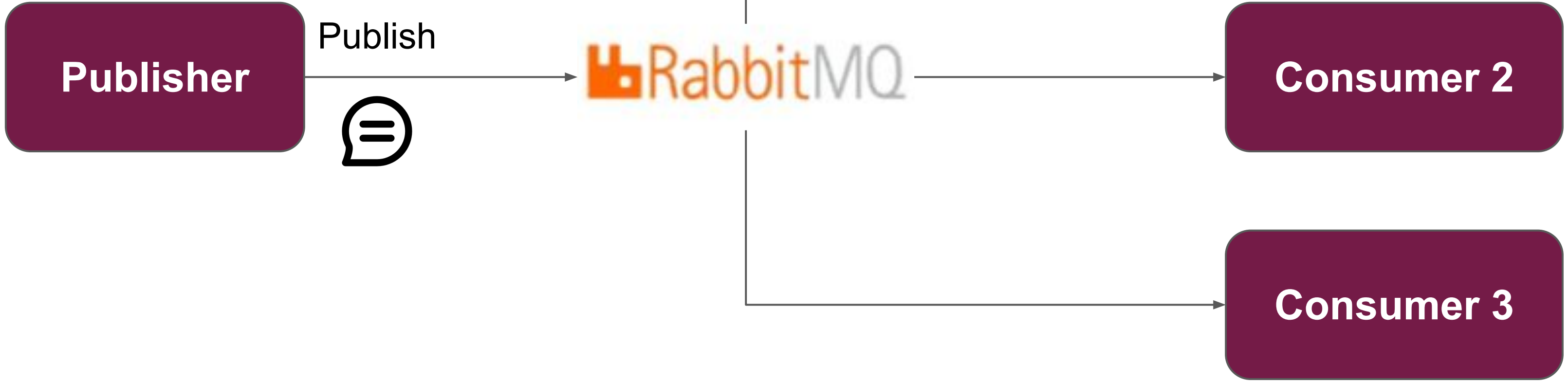
Consumer 1= 10

Consumer 2 = 5

Consumer 3 = 0

Qos

Prefetch Count = 3

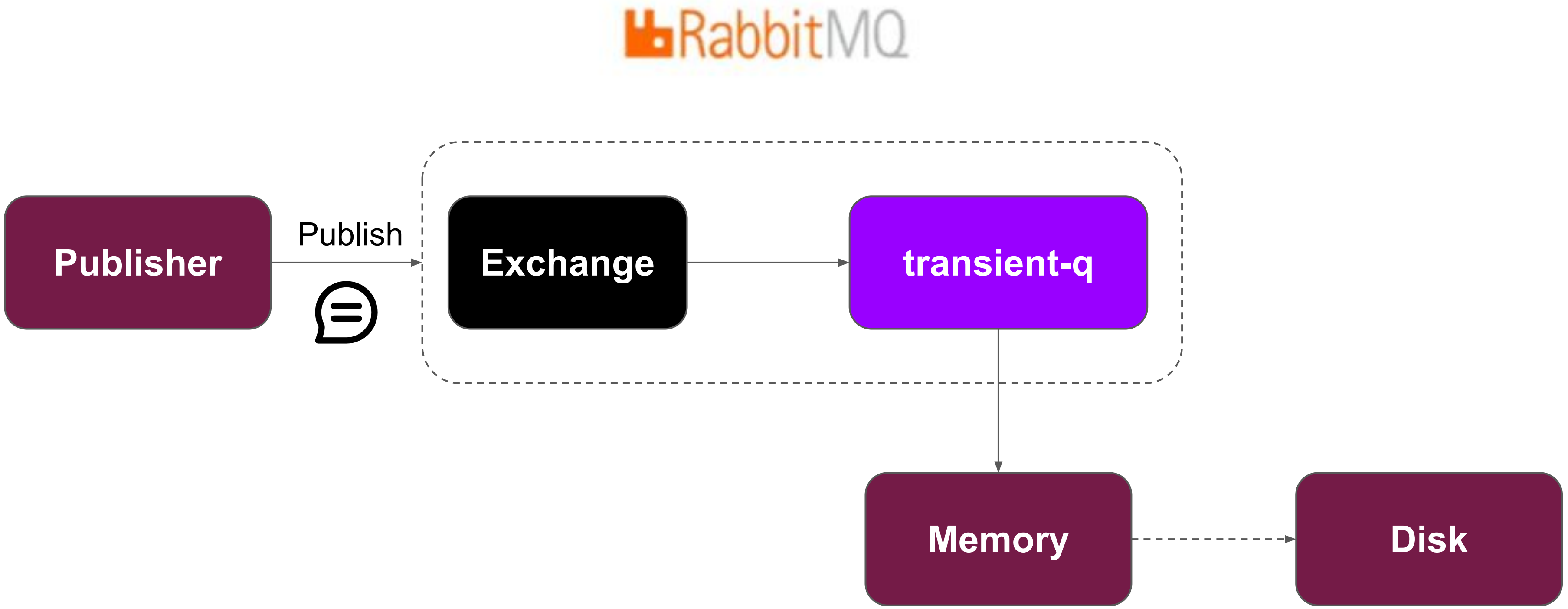


Message Persistence

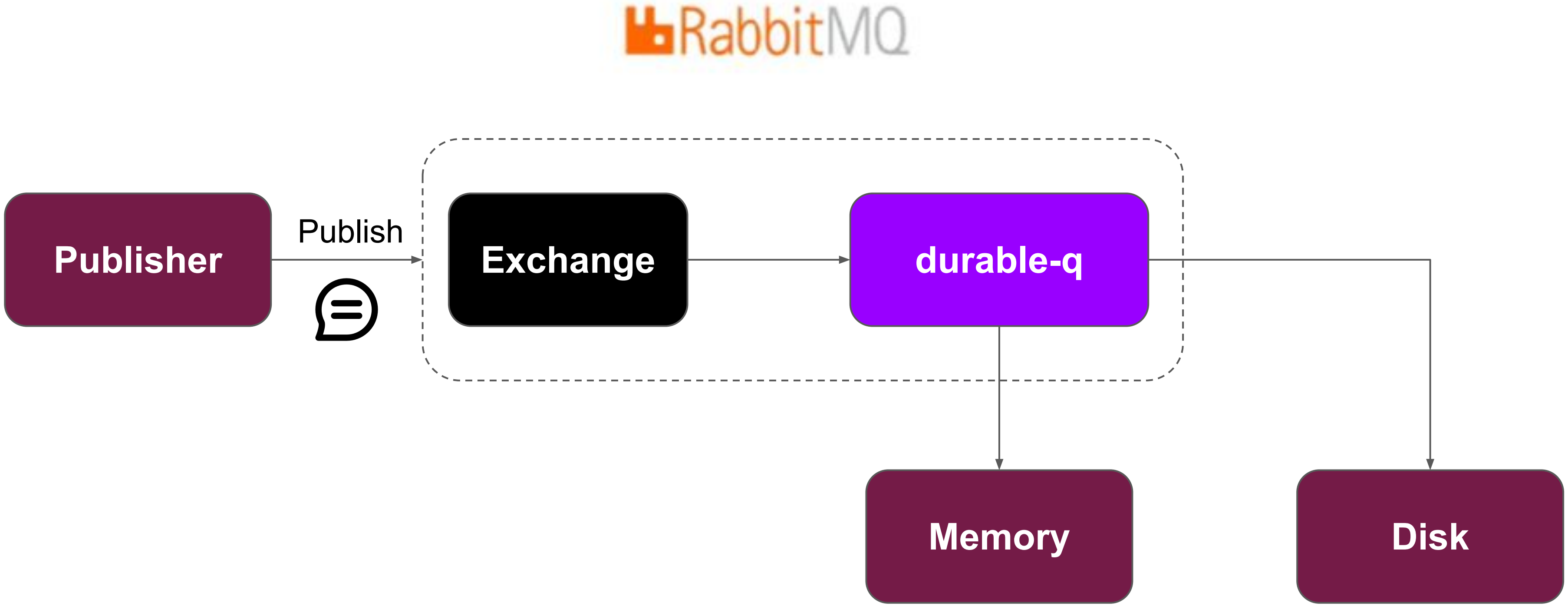
Queue - Durability	Message - Delivery mode
Durable (True)	1 - Non-persistent
Transient (False)	2 - Persistent

Queue	Delivery mode	Message
Durable	Non-persistent	Non-persistent
Durable	Persistent	Persistent
Transient	Non-persistent	Non-persistent
Transient	Persistent	Non-persistent

Non Persistent Message

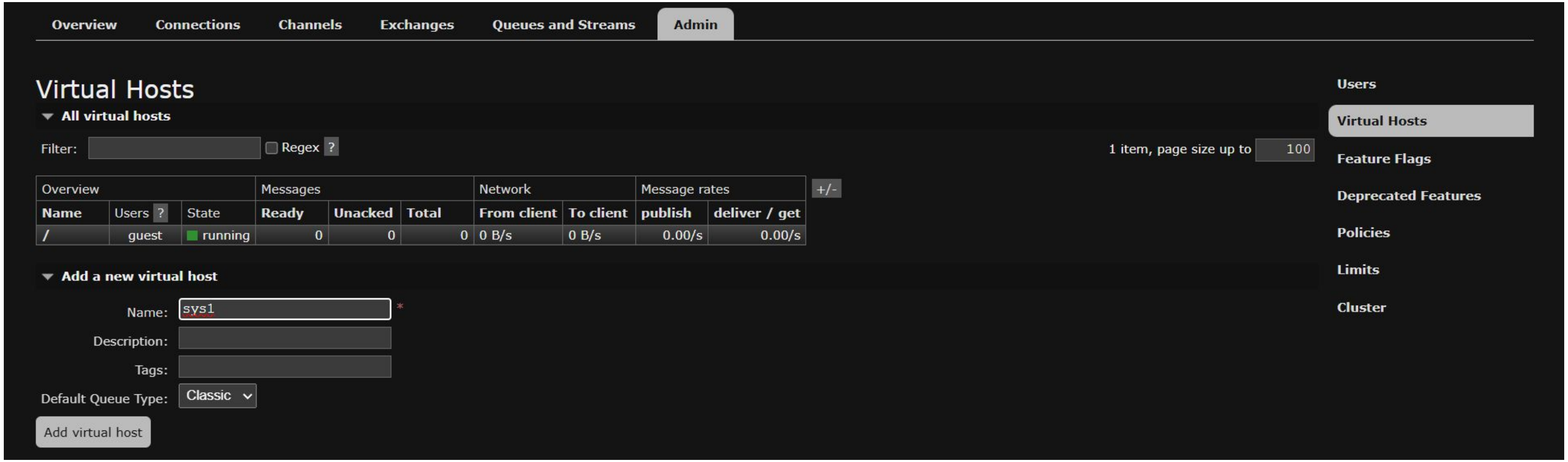


Persistent Message



Virtual Hosts (Multi Tenancy)

Client Connections	Delete
Exchanges	Limits
Queues	User Permissions
Bindings	
Messages	



Virtual Hosts

All virtual hosts

Filter: [ ] Regex ?

1 item, page size up to 100

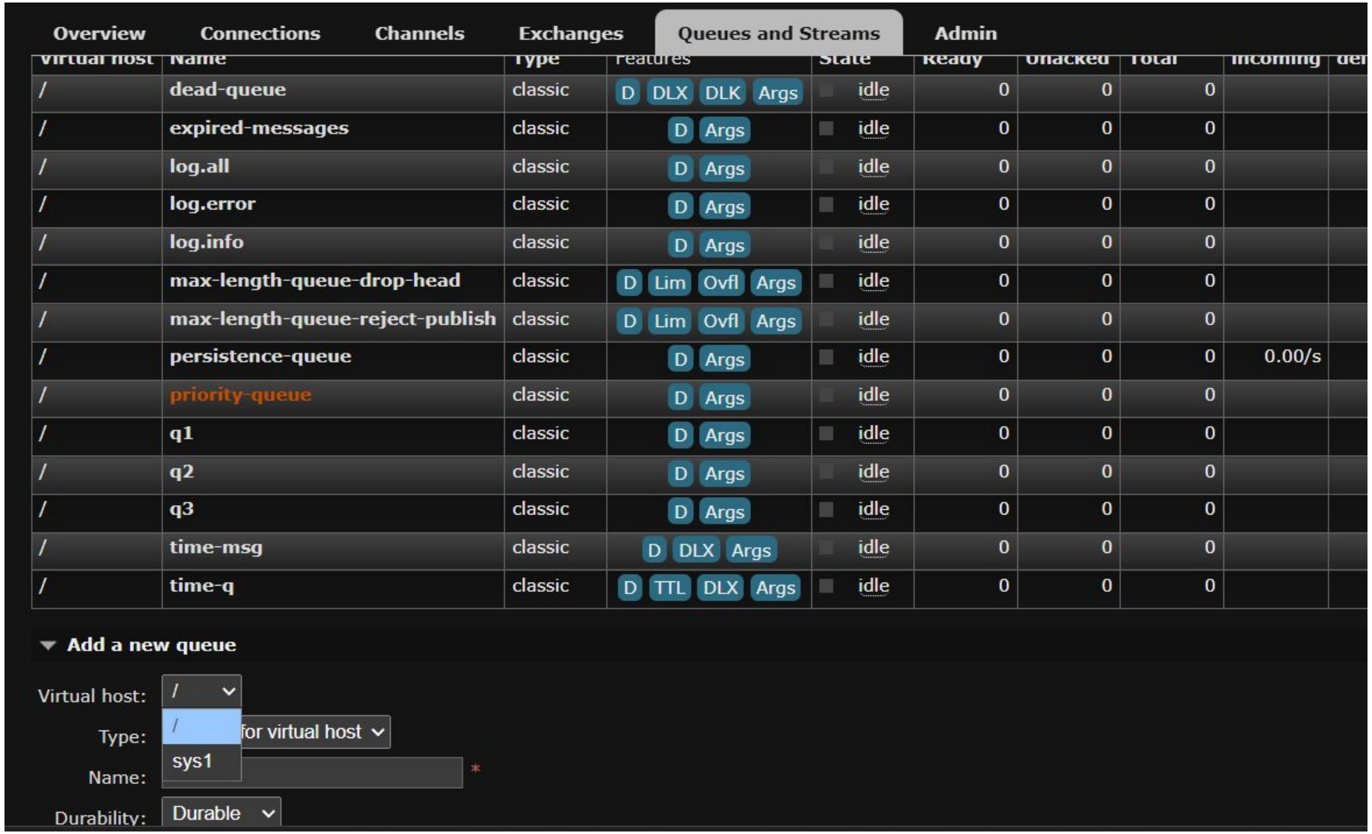
Overview			Messages			Network		Message rates	
Name	Users ?	State	Ready	Unacked	Total	From client	To client	publish	deliver / get
/	guest	🟩 running	0	0	0	0 B/s	0 B/s	0.00/s	0.00/s

Add a new virtual host

Name:	sys1 *
Description:	
Tags:	
Default Queue Type:	Classic

Add virtual host

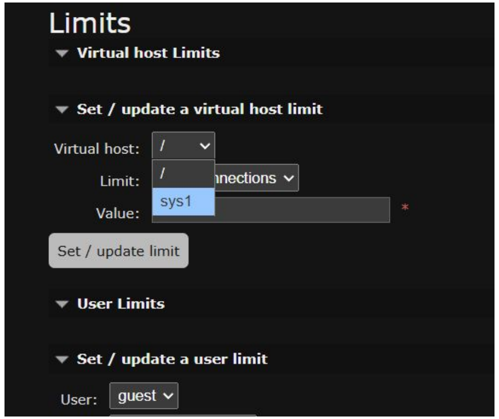
Virtual Hosts (Multi Tenancy)



Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	del
/	dead-queue	classic	D DLX DLK Args	idle	0	0	0		
/	expired-messages	classic	D Args	idle	0	0	0		
/	log.all	classic	D Args	idle	0	0	0		
/	log.error	classic	D Args	idle	0	0	0		
/	log.info	classic	D Args	idle	0	0	0		
/	max-length-queue-drop-head	classic	D Lim Ovfl Args	idle	0	0	0		
/	max-length-queue-reject-publish	classic	D Lim Ovfl Args	idle	0	0	0		
/	persistence-queue	classic	D Args	idle	0	0	0	0.00/s	
/	priority-queue	classic	D Args	idle	0	0	0		
/	q1	classic	D Args	idle	0	0	0		
/	q2	classic	D Args	idle	0	0	0		
/	q3	classic	D Args	idle	0	0	0		
/	time-msg	classic	D DLX Args	idle	0	0	0		
/	time-q	classic	D TTL DLX Args	idle	0	0	0		

▼ Add a new queue

Virtual host:	/ / sys1
Type:	for virtual host
Name:	*
Durability:	Durable



Limits

▼ Virtual host Limits

▼ Set / update a virtual host limit

Virtual host:	/ / sys1
Limit:	nnections
Value:	*

Set / update limit

▼ User Limits

▼ Set / update a user limit

User: guest

Virtual Hosts

guest user

Create user

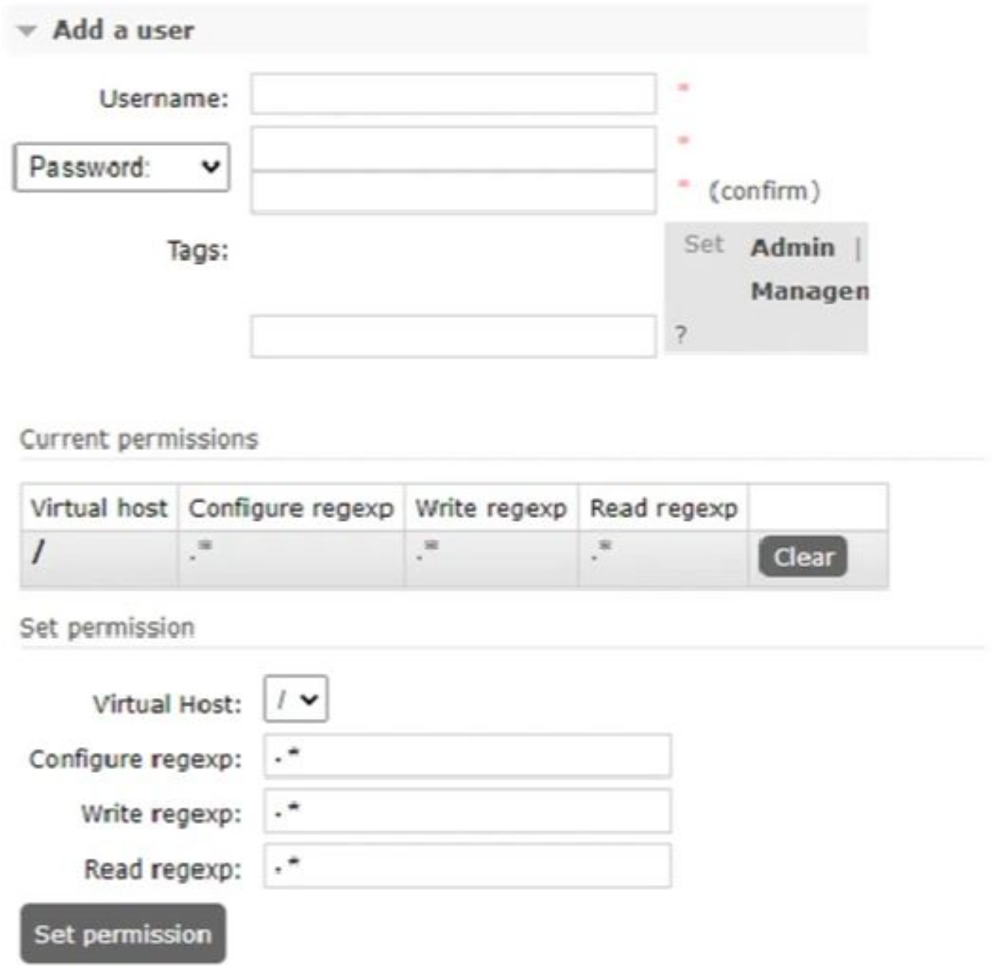
Permissions

- configure
- write
- read

^(amq.fanout)$

^(amq.*)$

^(amq.fanout|amq.direct)$



Add a user

Username:	*
Password:	*
	* (confirm)
Tags:	Set Admin | Managen ?

Current permissions

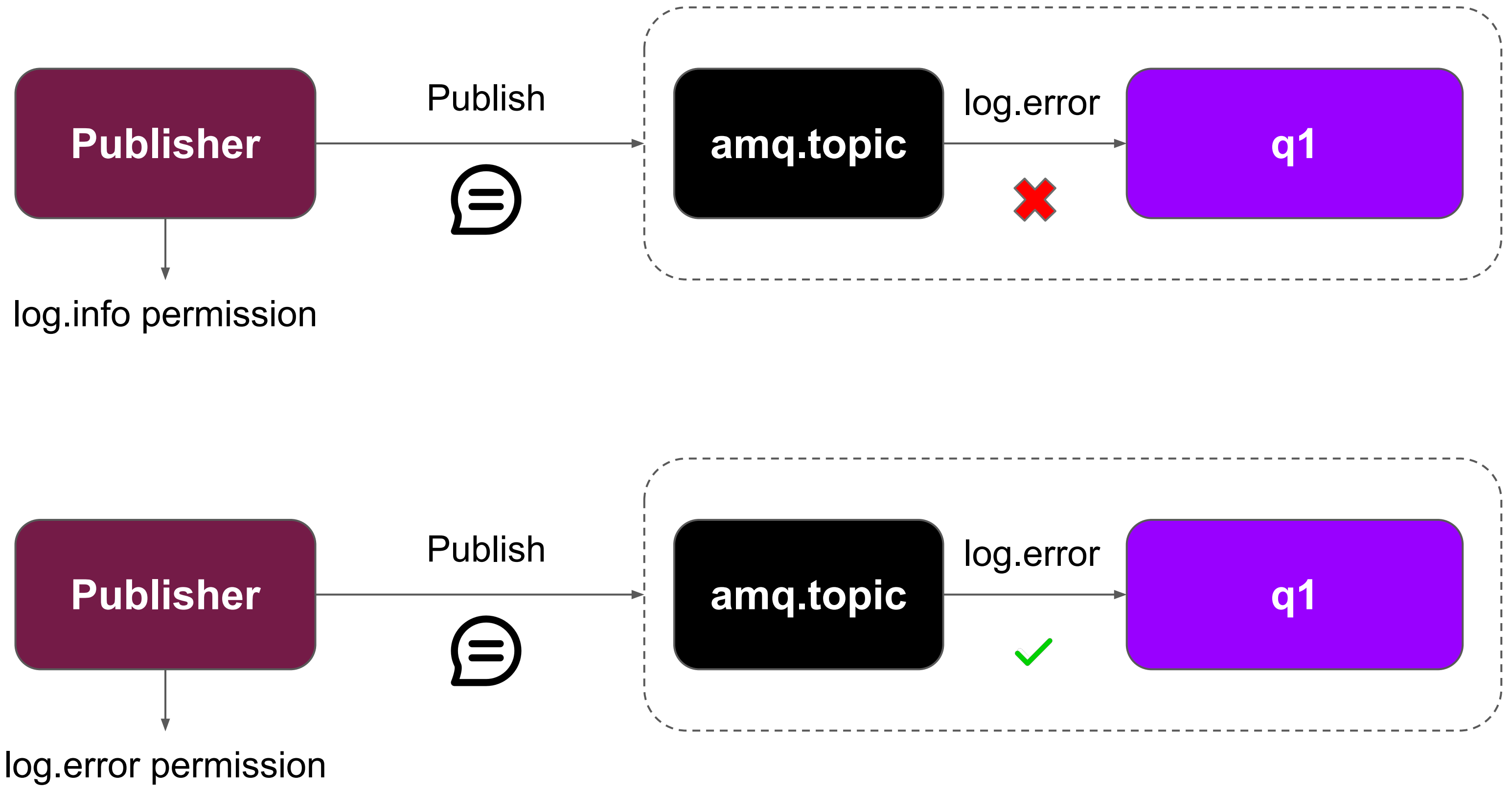
Virtual host	Configure regexp	Write regexp	Read regexp	
/	.*	.*	.*	Clear

Set permission

Virtual Host:	/
Configure regexp:	.*
Write regexp:	.*
Read regexp:	.*

Operation		Permission		
		configure	write	read
exchange.declare	(passive=false)	exchange		
exchange.declare	(passive=true)			
exchange.declare	(with AE)	exchange	exchange (AE)	exchange
exchange.delete		exchange		
queue.declare	(passive=false)	queue		
queue.declare	(passive=true)			
queue.declare	(with DLX)	queue	exchange (DLX)	queue
queue.delete		queue		
exchange.bind			exchange (destination)	exchange (source)
exchange.unbind			exchange (destination)	exchange (source)
queue.bind			queue	exchange
queue.unbind			queue	exchange
basic.publish			exchange	
basic.get				queue
basic.consume				queue
queue.purge				queue

Topic Write Permissions



Quorum Queue

Durable, replicated FIFO queue based on the Raft consensus algorithm

Quorum queues and streams replace durable mirrored queues

Quorum queues are optimized where data safety is a top priority

A client library that can use regular mirrored queues will be able to use quorum queues

Feature	Classic	Quorum
Non-durable queues	Yes	No
Exclusivity	Yes	No
Per message persistence	Yes	Always
Membership changes	Automatic	Manual
Message TTL	Yes	Yes
TTL	Yes	Yes
Queue length limits	Yes	Yes
Lazy behavior	Yes	Always
Message priority	Yes	No
Consumer priority	Yes	Yes
Dead letter exchanges	Yes	Yes
Adheres to policies	Yes	Yes
Poison message handling	No	Yes
Global QoS Prefetch	Yes	No

Quorum Queue

Streams are a new persistent and replicated data structure

They can be used via a RabbitMQ client library as if it was a queue

Or through a dedicated binary protocol plugin and associated client

Large fan-outs

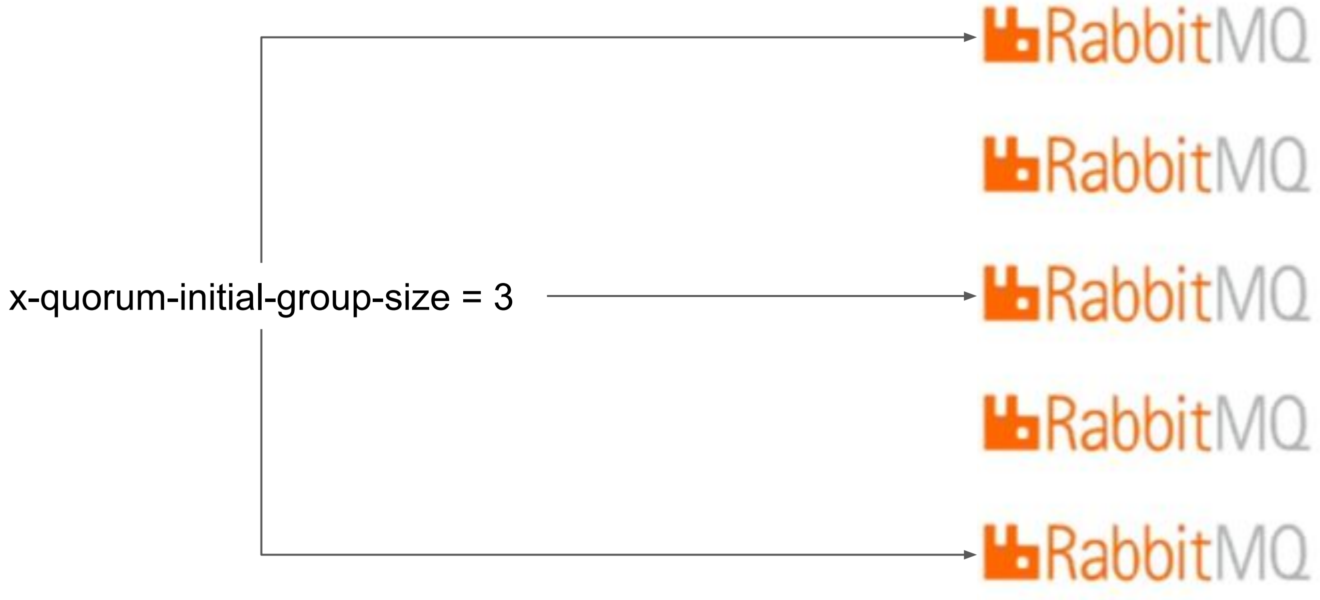
Replay / Time-travelling

Throughput Performance

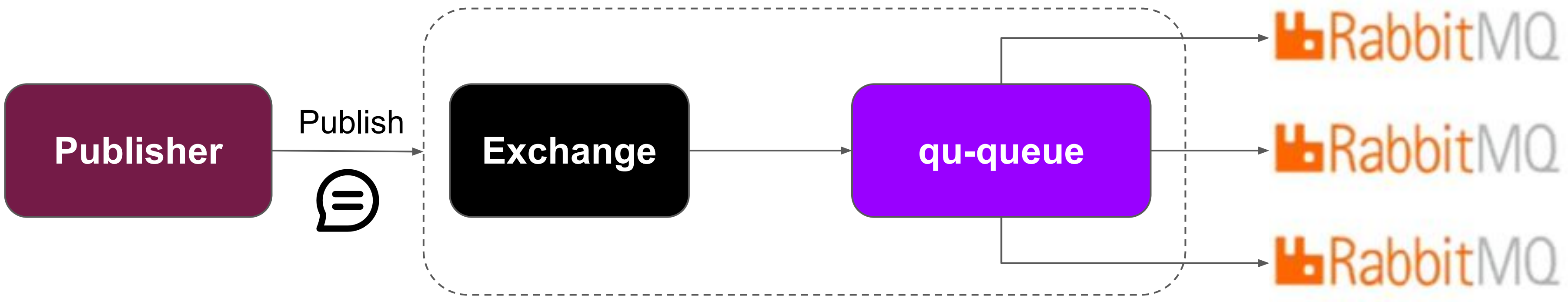
Large logs

Feature	Classic	Stream
Non-durable queues	Yes	No
Exclusivity	Yes	No
Per message persistence	Yes	Always
Membership changes	Automatic	Manual
Message TTL	Yes	No
Queue length limits	Yes	No
Lazy behavior	Yes	Inherent
Message priority	Yes	No
Consumer priority	Yes	No
Dead letter exchanges	Yes	No
Adheres to policies	Yes	Check retention
Reacts to memory alarms	Yes	No
Poison message handling	No	No
Global QoS Prefetch	Yes	No

Quorum Replicas



x-quorum-initial-group-size (cont.)



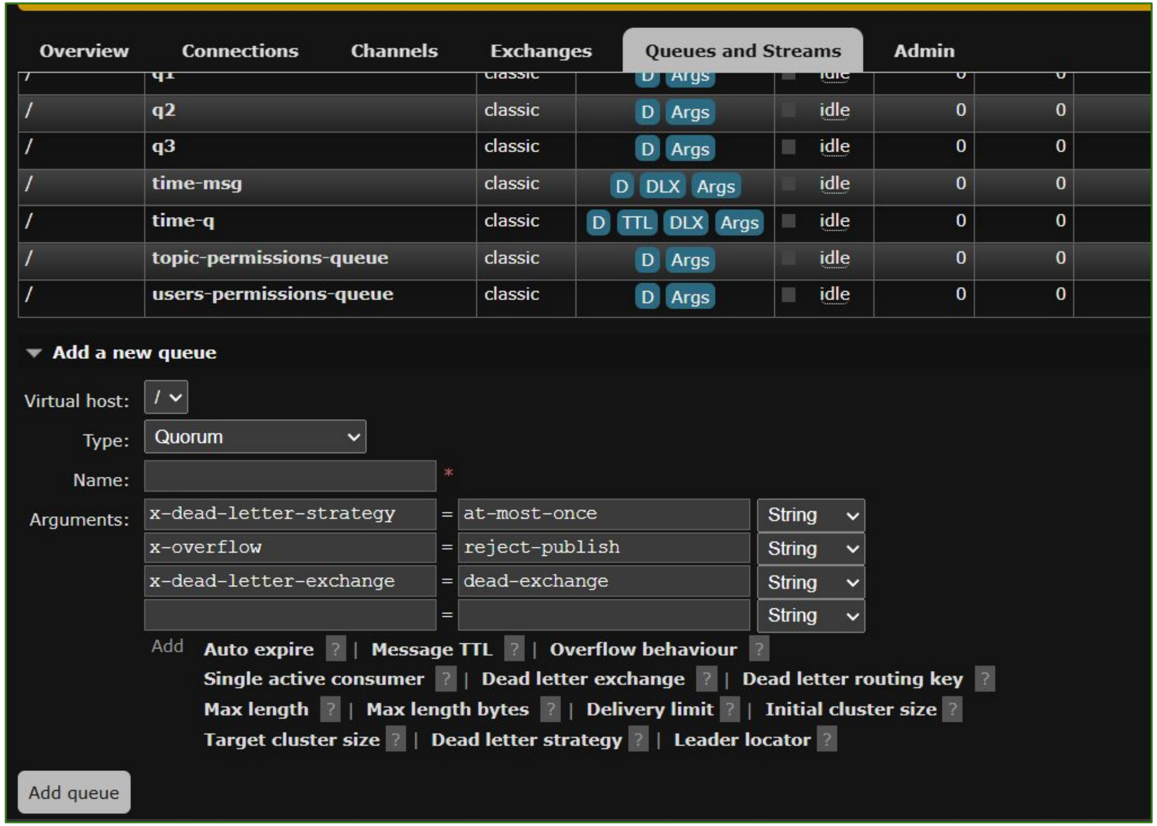
Dead Letter Strategy

x-dead-letter-strategy = at-least-once **default** = at-most-once

x-overflow = reject-publish

x-dead-letter-exchange = x

stream_queue is enabled



/	q1	classic	D Args	idle	0	0
/	q2	classic	D Args	idle	0	0
/	q3	classic	D Args	idle	0	0
/	time-msg	classic	D DLX Args	idle	0	0
/	time-q	classic	D TTL DLX Args	idle	0	0
/	topic-permissions-queue	classic	D Args	idle	0	0
/	users-permissions-queue	classic	D Args	idle	0	0

▾ Add a new queue

Virtual host: /

Type: Quorum

Name: *

Arguments:

x-dead-letter-strategy	=	at-most-once	String
x-overflow	=	reject-publish	String
x-dead-letter-exchange	=	dead-exchange	String
	=		String

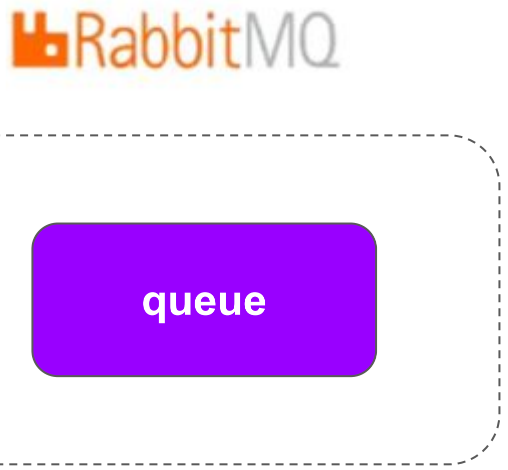
Add Auto expire ? | Message TTL ? | Overflow behaviour ?
Single active consumer ? | Dead letter exchange ? | Dead letter routing key ?
Max length ? | Max length bytes ? | Delivery limit ? | Initial cluster size ?
Target cluster size ? | Dead letter strategy ? | Leader locator ?

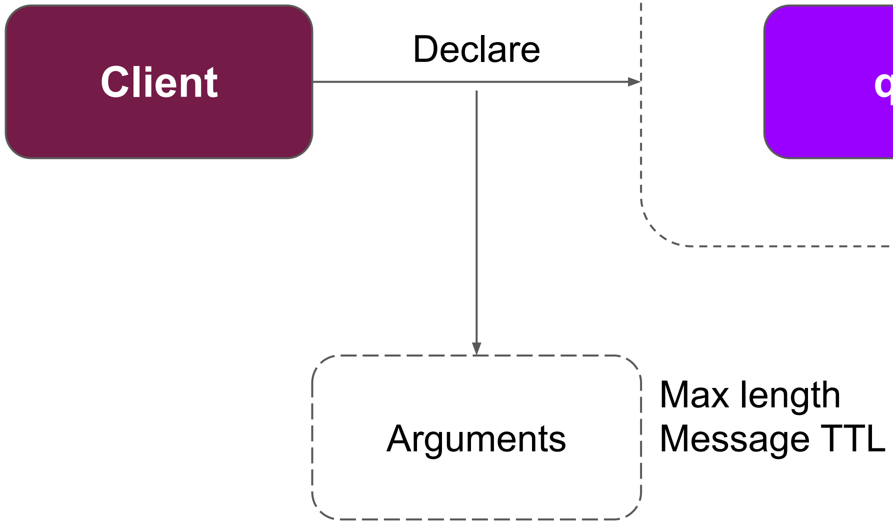
Add queue

Retention Arguments

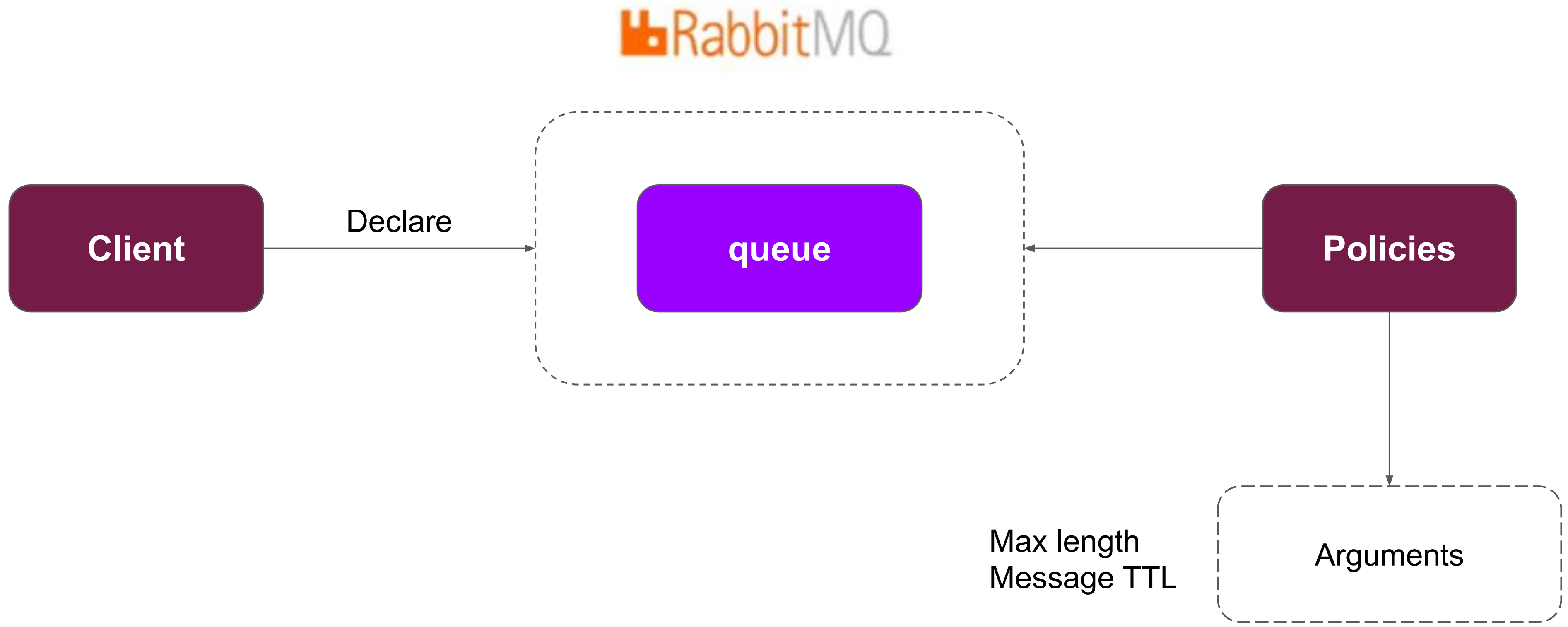
x-max-age	Y, M, D, h, m, s
x-max-length-bytes	
x-stream-max-segment-size-bytes	

Why Policies were introduced?

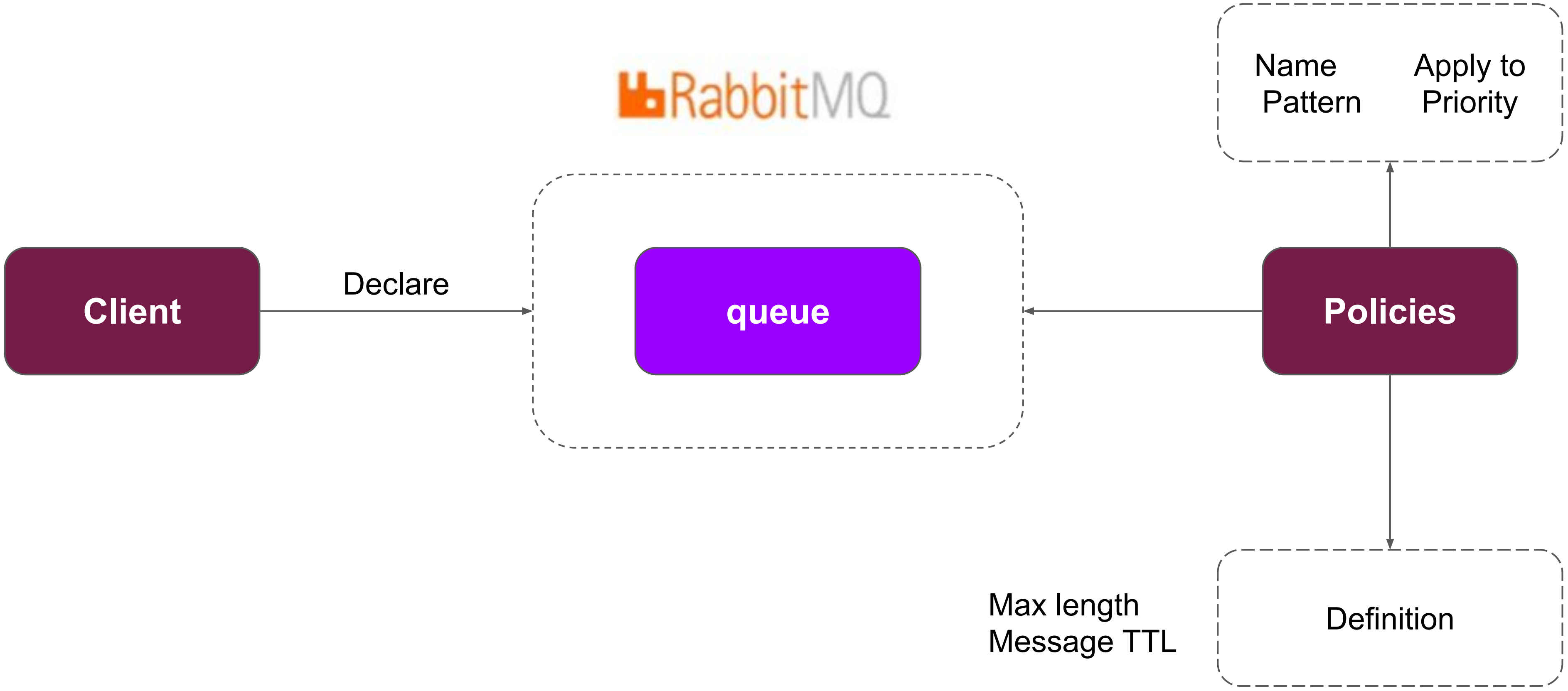




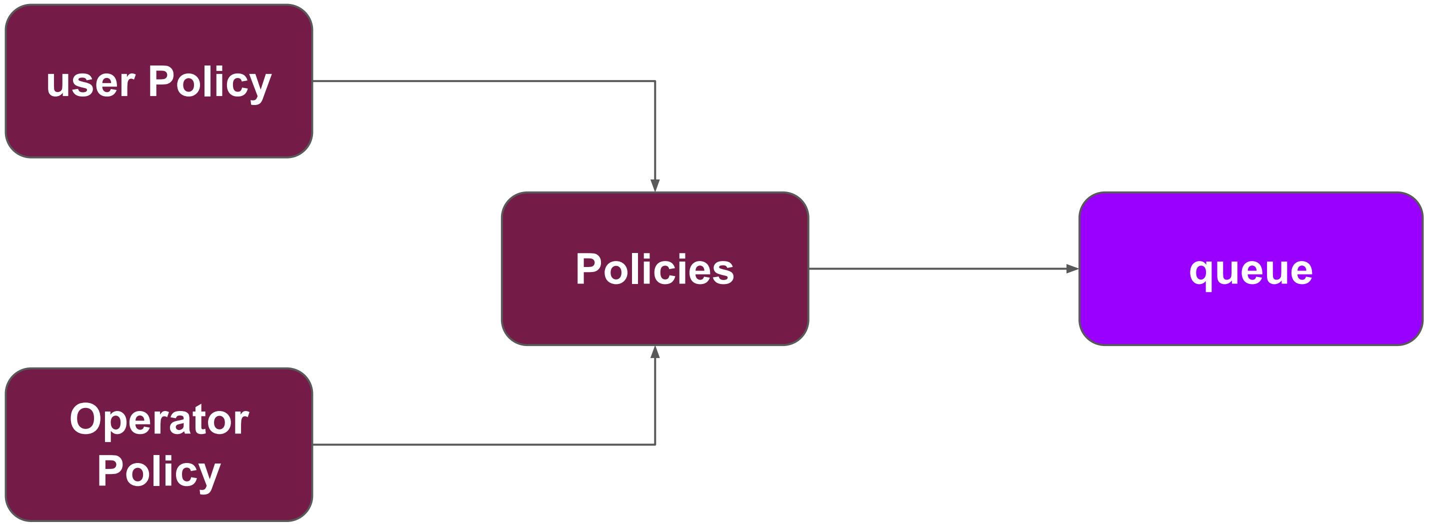
Why Policies were introduced? (cont.)



How Policies Work?



How Operator Policy Works?



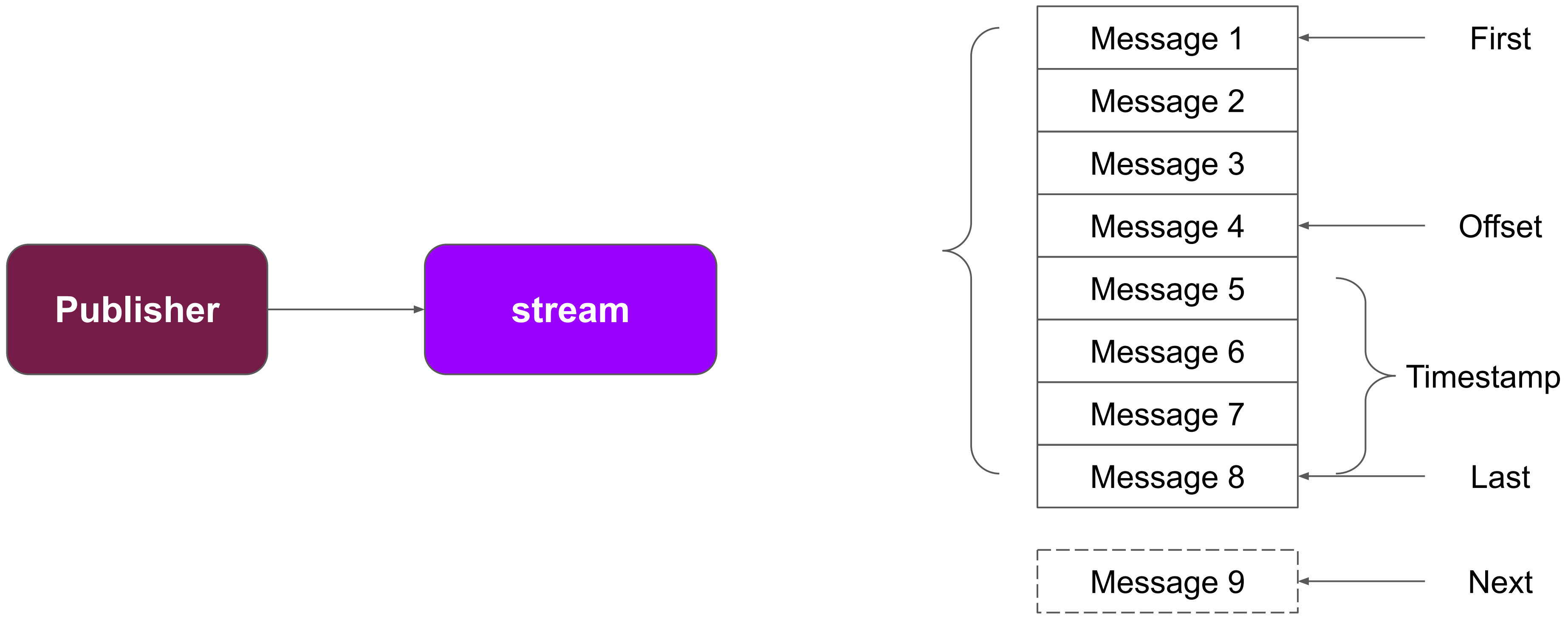
Consumer Stream by DOTNET

rabbitmq-plugins enable rabbitmq_stream

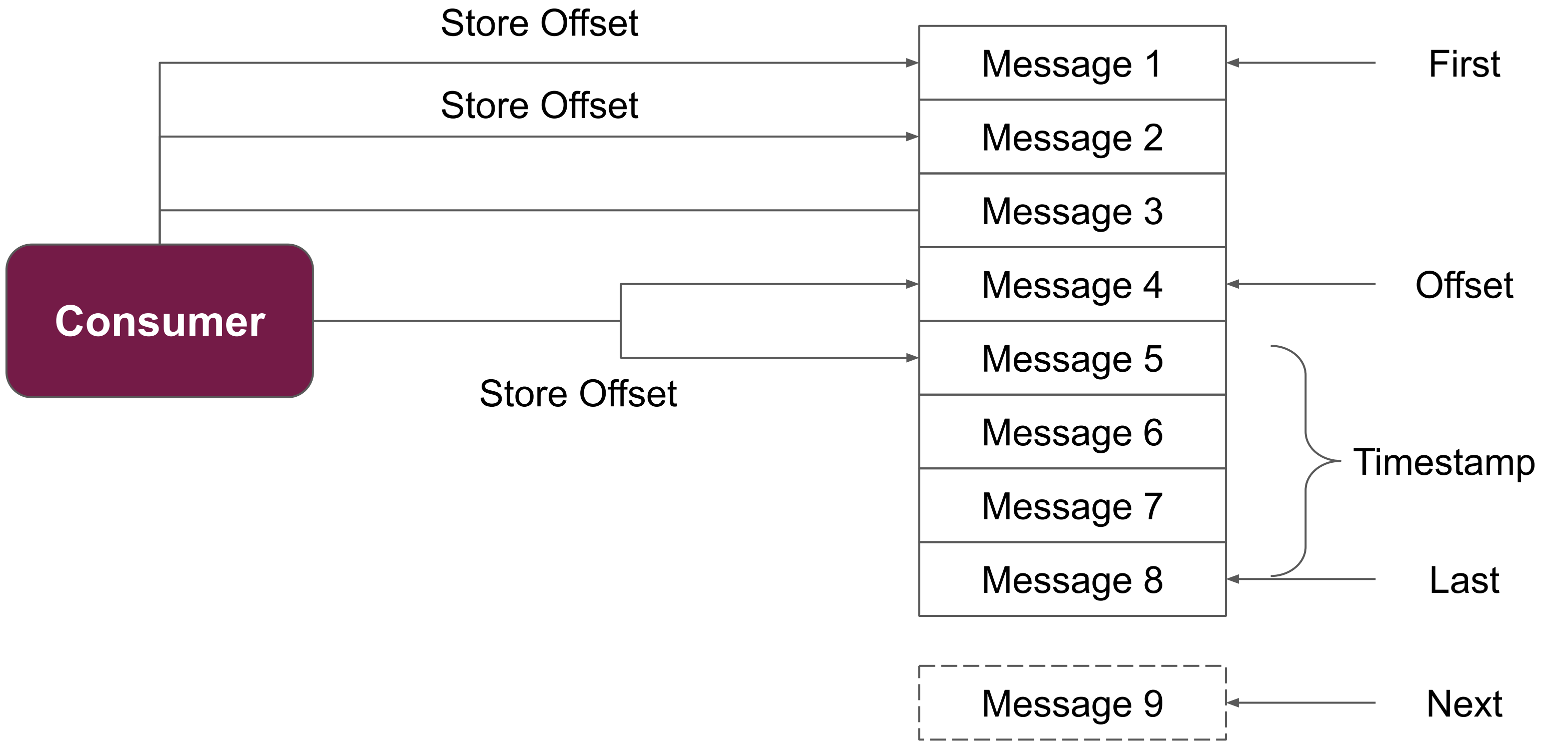
RabbitMQ.Stream.Client

Create user, guest user ?

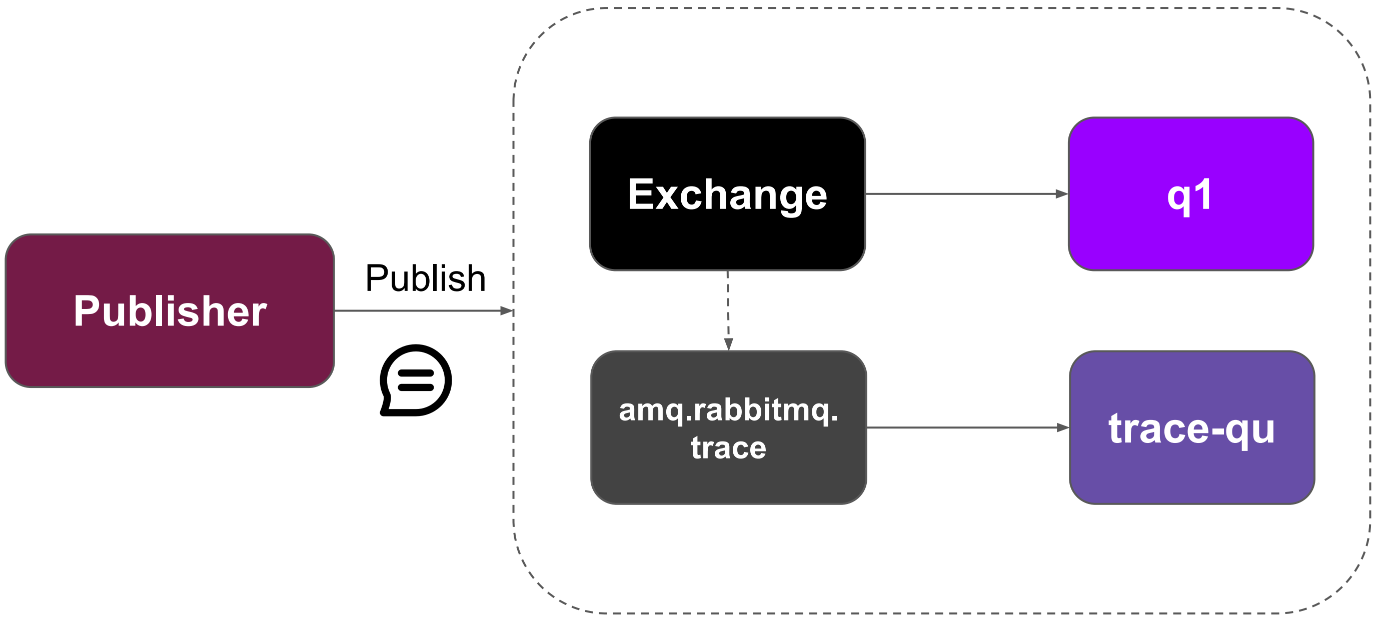
Consumer Stream - Offset



Manage Consumer Offset

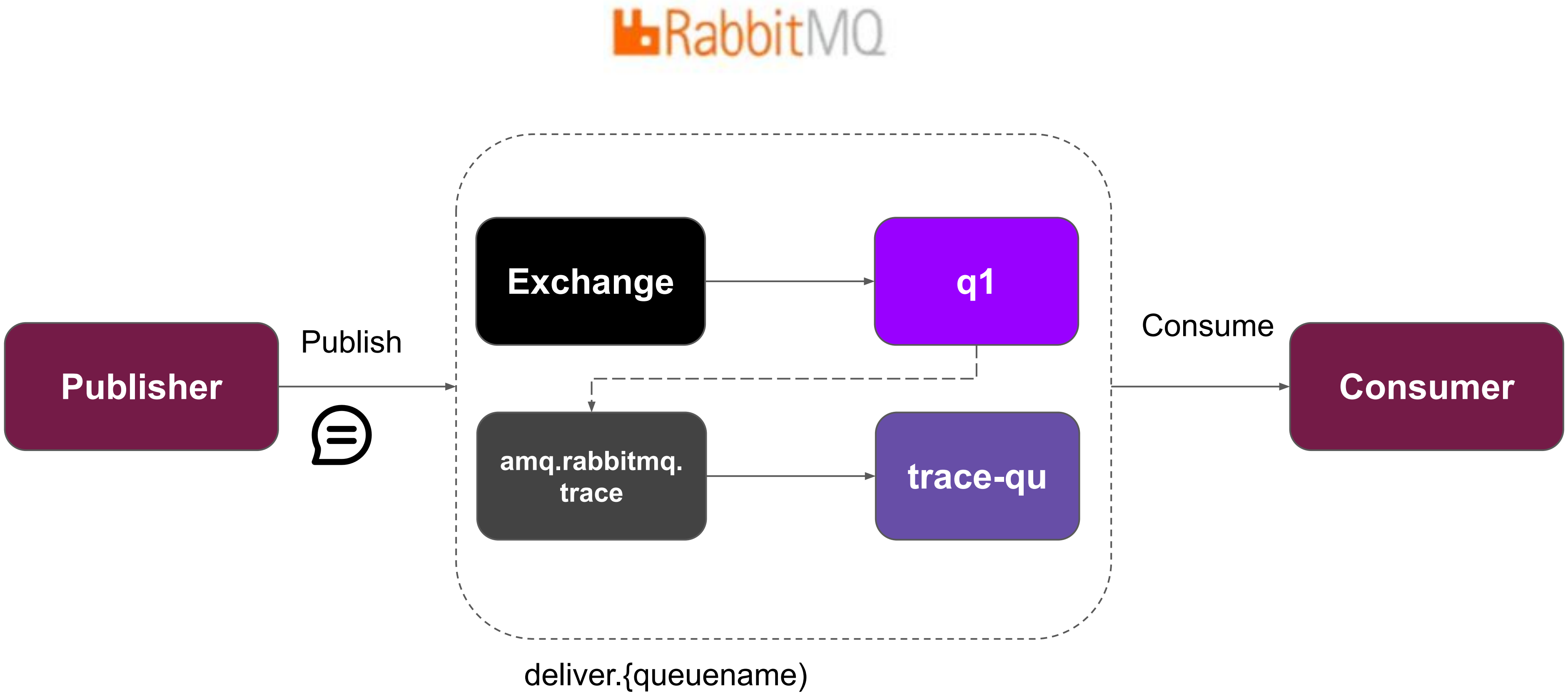


Firehose Plugin



publish.(exchangename}

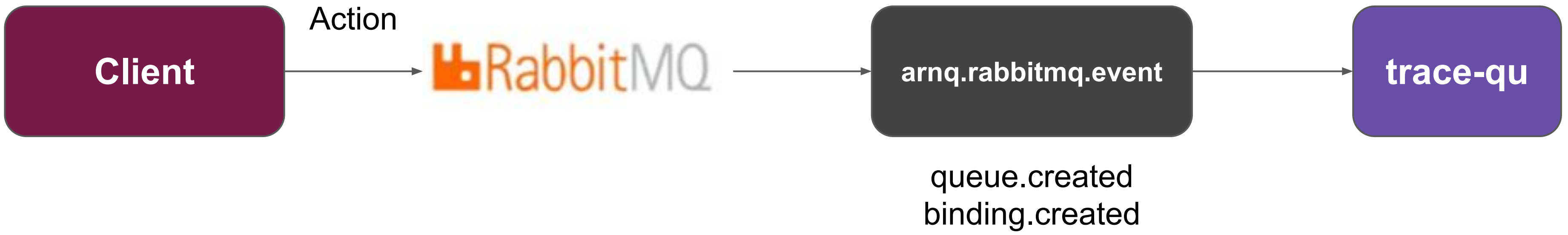
Firehose Plugin (cont.)



Firehose Plugin (cont.)

```
rabbitmq-plugins enable rabbitmq_tracing

rabbitmqctl trace_on

rabbitmqctl trace_off
```

Event Exchange Plugin



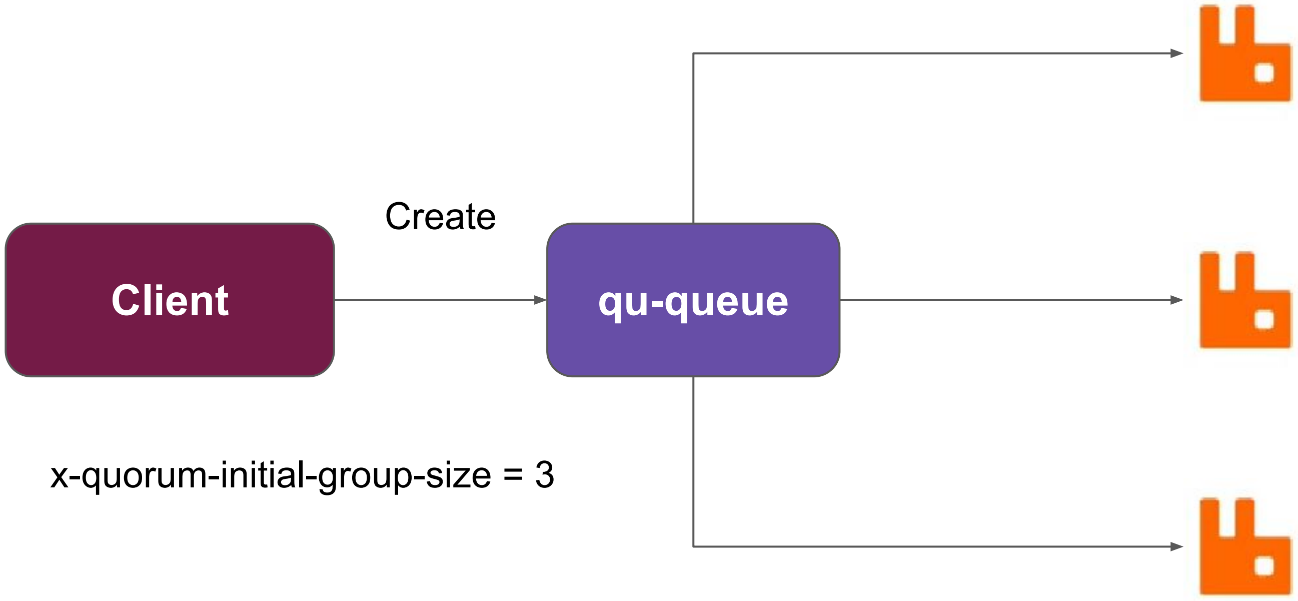
Event Exchange Plugin (cont.)

```
rabbitmq-plugins enable rabbitmq_event_exchange
```

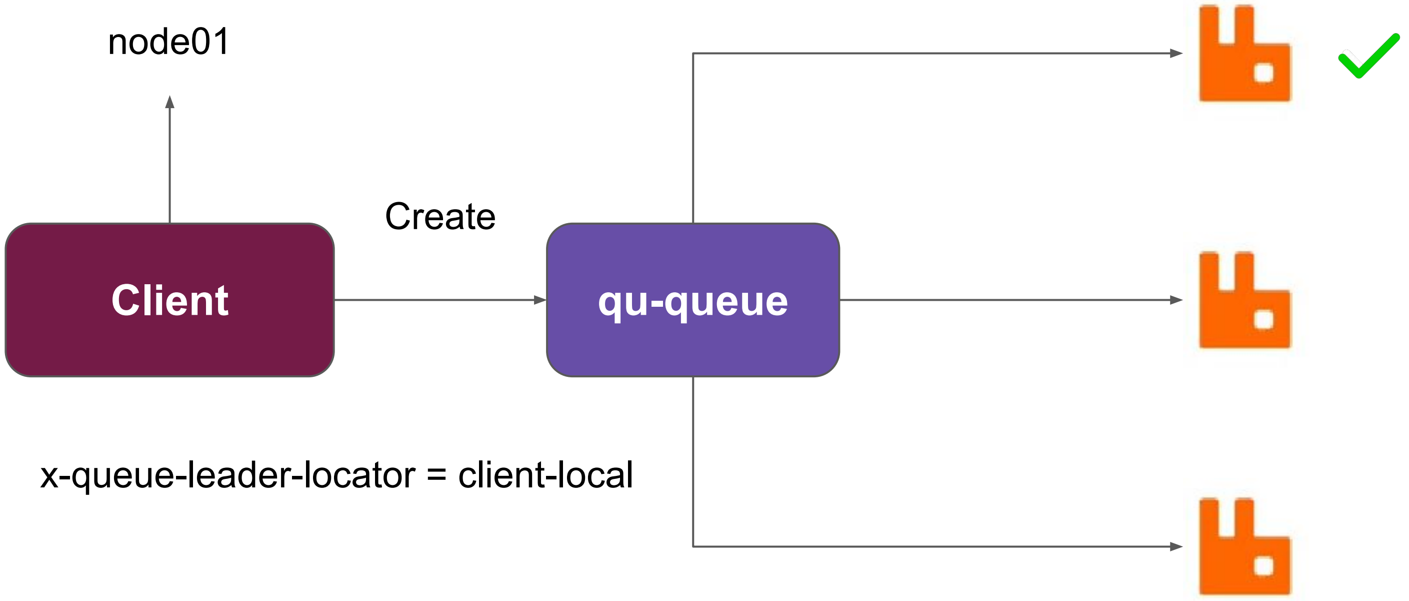
RabbitMQ Cluster

node01
node02
node03

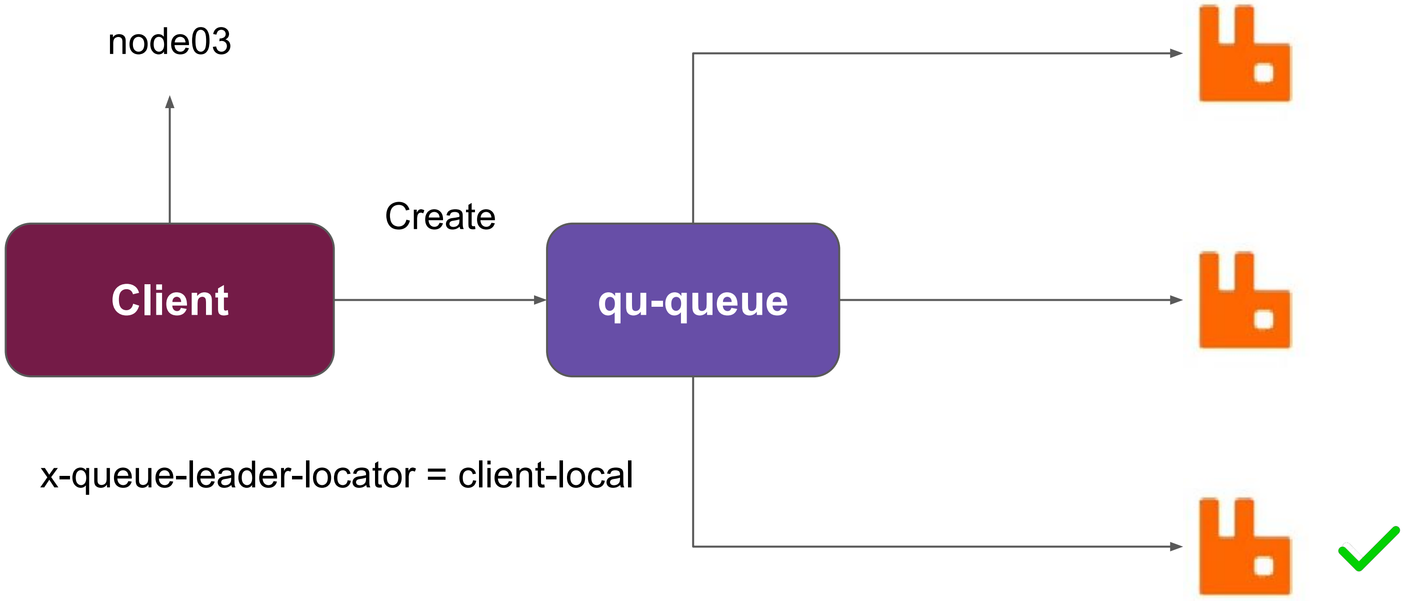
x-quorum-initial-group-size



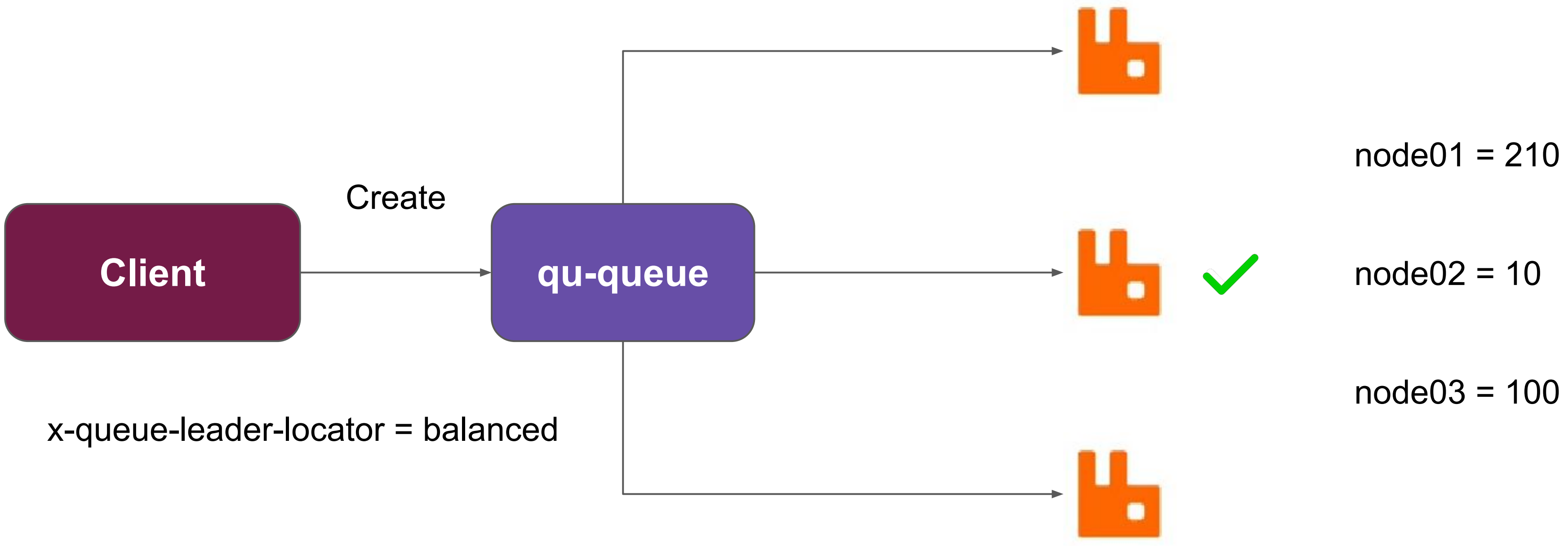
x-queue-leader-locator

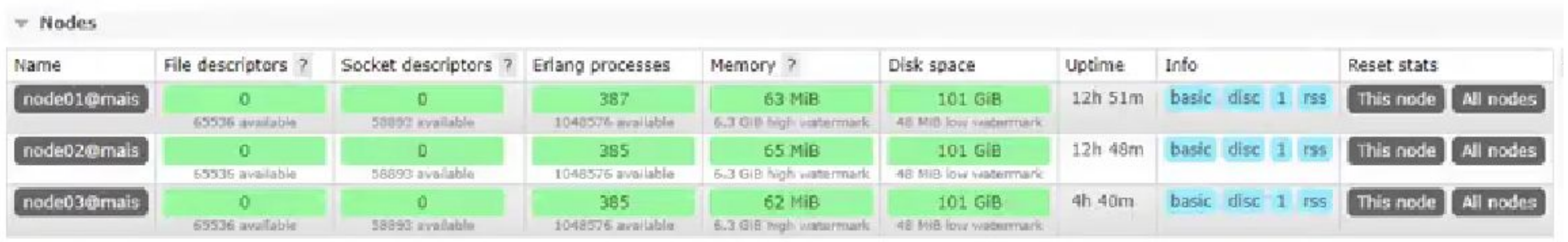


X-queue-leader-locator (cont.)



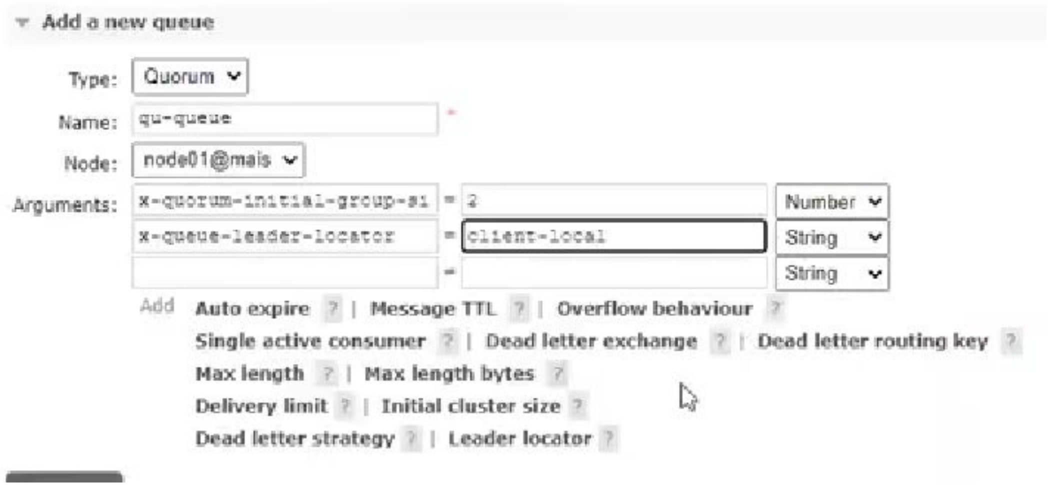
X-queue-leader-locator (cont.)





▾ Nodes

Name	File descriptors ?	Socket descriptors ?	Erlang processes	Memory ?	Disk space	Uptime	Info	Reset stats
node01@mais	0 65536 available	0 58893 available	387 1048576 available	63 MiB 6.3 GiB high watermark	101 GiB 48 MiB low watermark	12h 51m	basic disc 1 rss	This node All nodes
node02@mais	0 65536 available	0 58893 available	385 1048576 available	65 MiB 6.3 GiB high watermark	101 GiB 48 MiB low watermark	12h 48m	basic disc 1 rss	This node All nodes
node03@mais	0 65536 available	0 58893 available	385 1048576 available	62 MiB 6.3 GiB high watermark	101 GiB 48 MiB low watermark	4h 40m	basic disc 1 rss	This node All nodes

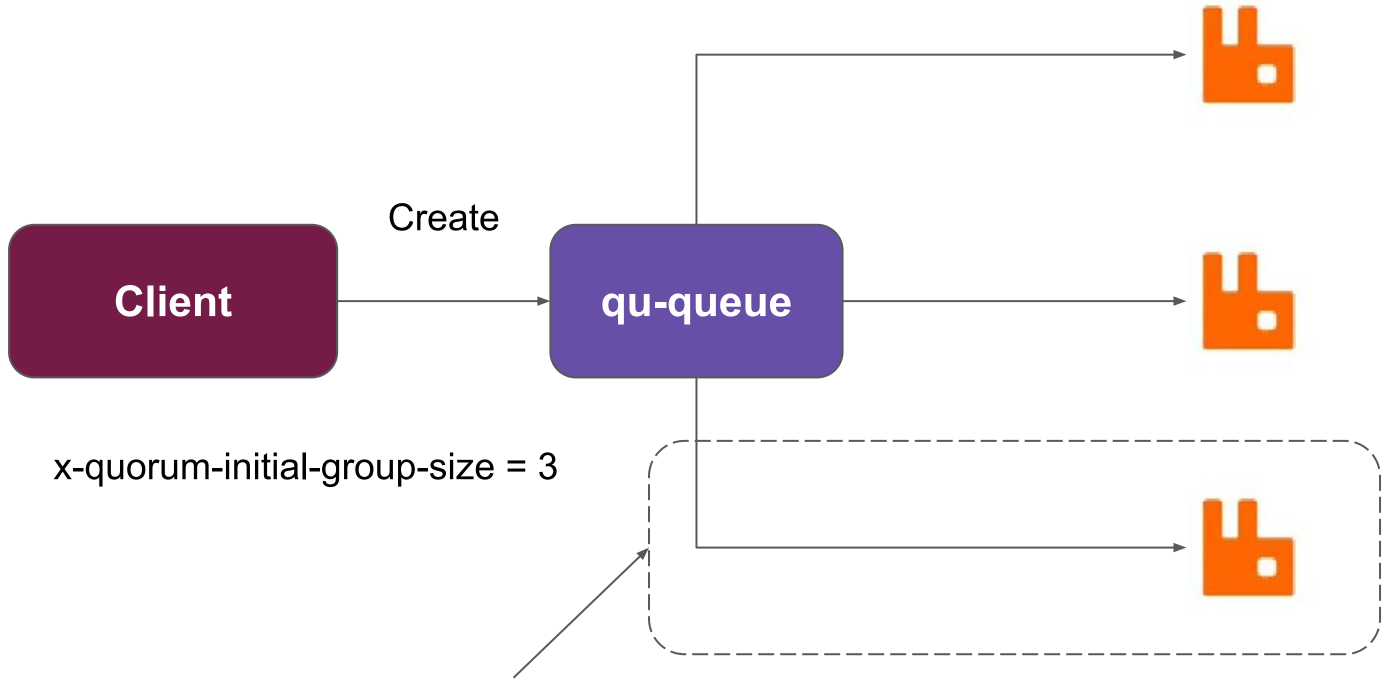


▾ Add a new queue

Type:	Quorum		
Name:	qu-queue *		
Node:	node01@mais		
Arguments:	x-quorum-initial-group-si =	2	Number
	x-queue-leader-locator =	client-local	String
	=		String

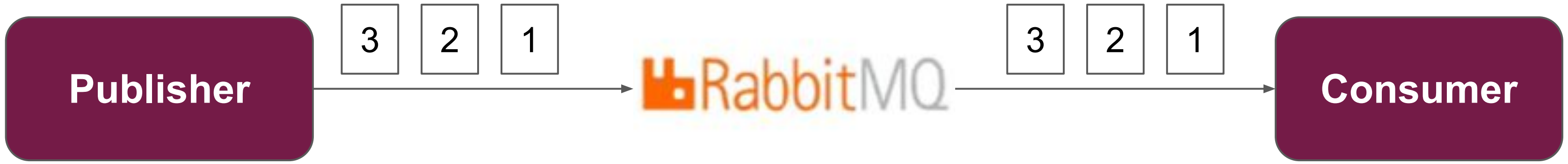
Add Auto expire ? | Message TTL ? | Overflow behaviour ?
Single active consumer ? | Dead letter exchange ? | Dead letter routing key ?
Max length ? | Max length bytes ?
Delivery limit ? | Initial cluster size ?
Dead letter strategy ? | Leader locator ?

x-quorum-initial-group-size

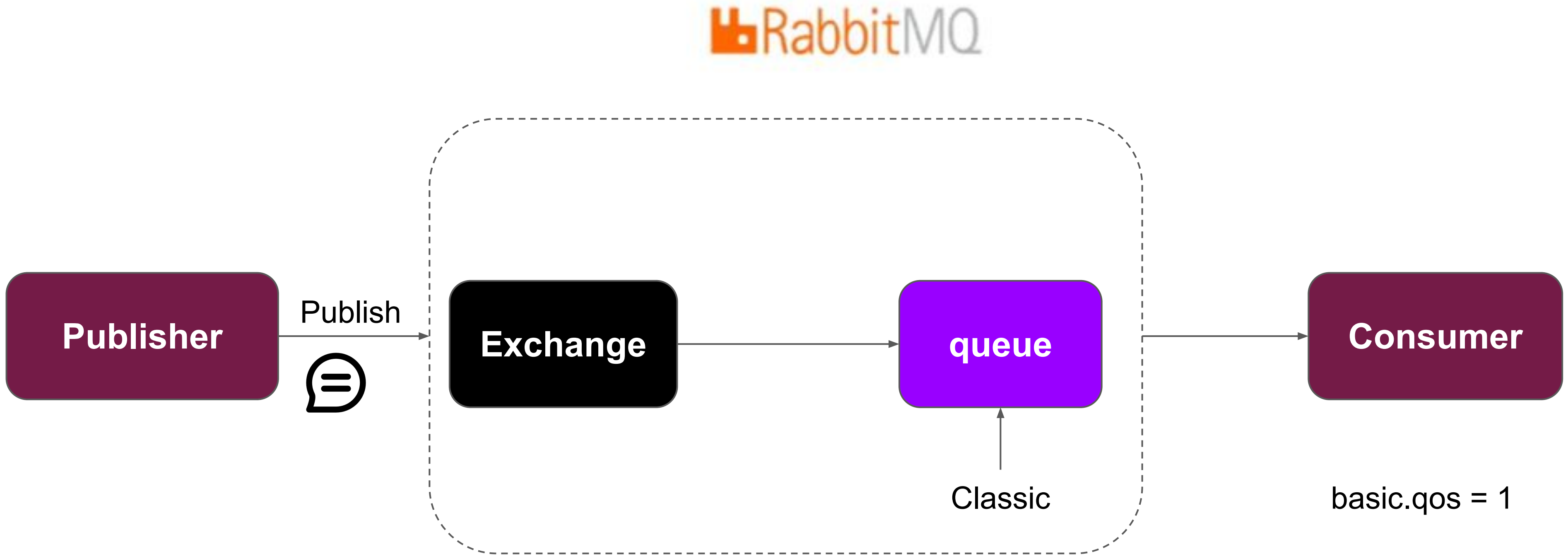


```
rabbitmq-queues add_member qu-queue node03@mais
```

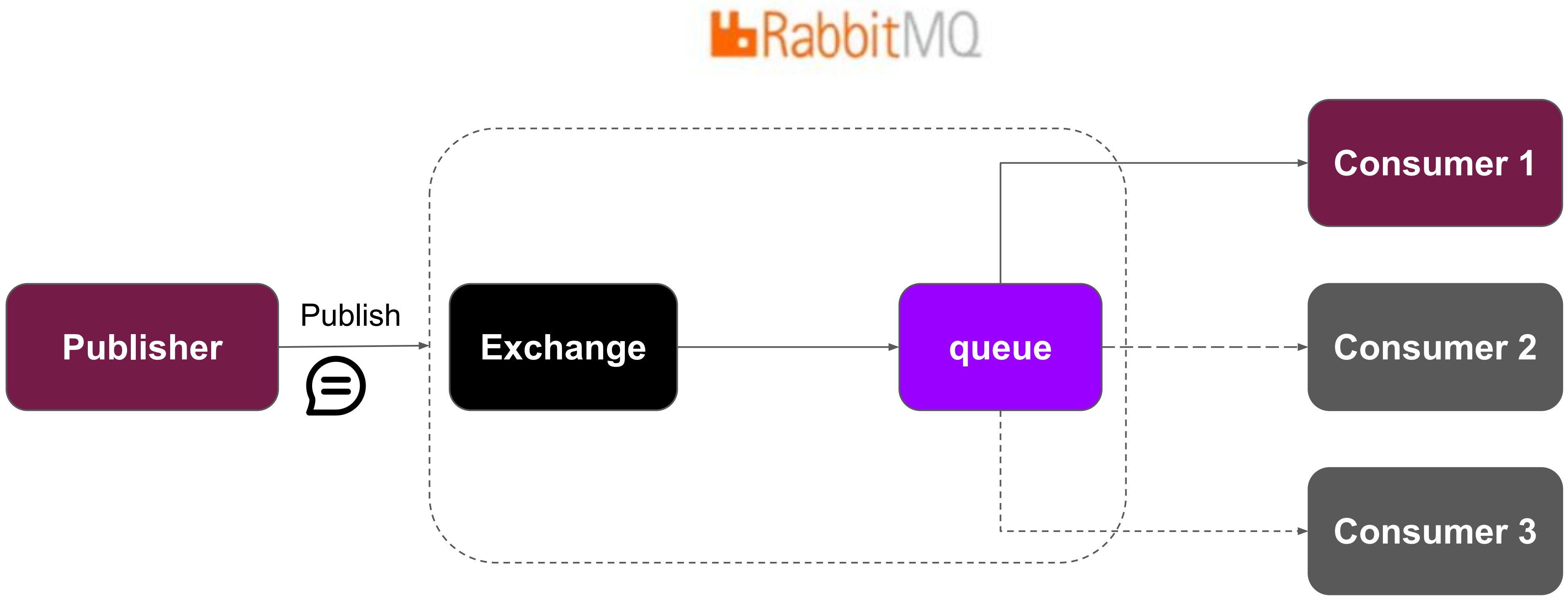
Message Ordering

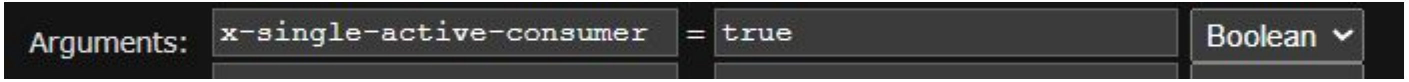


Prefetch Count = 1



Single Active Consumer



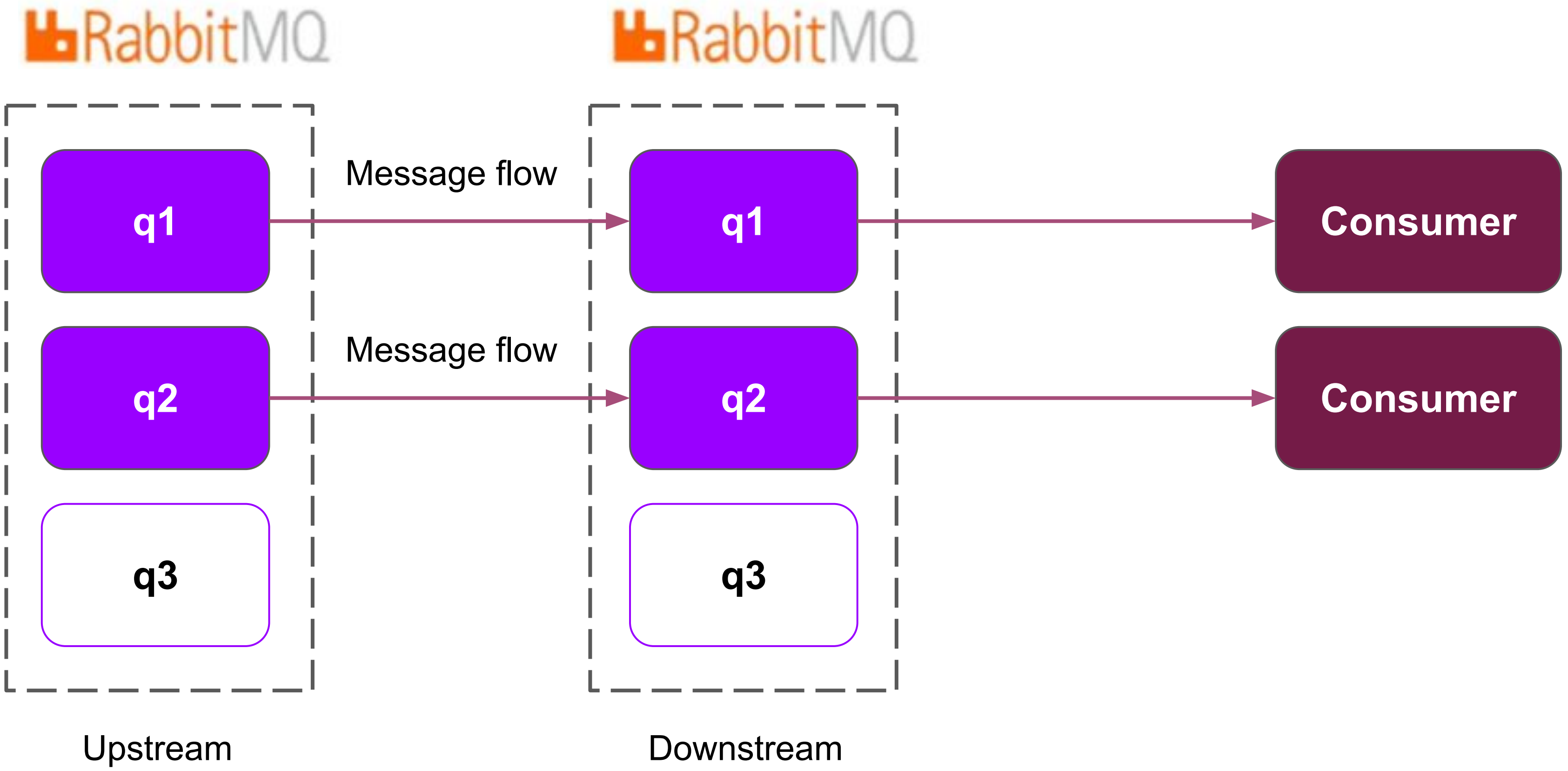

Arguments: x-single-active-consumer = true Boolean

Overview Federation

The goal of the Federation plugin is to transmit messages between brokers

The federation plugin is designed to tolerate intermittent connectivity

The federation plugin makes exchanges and queues federated

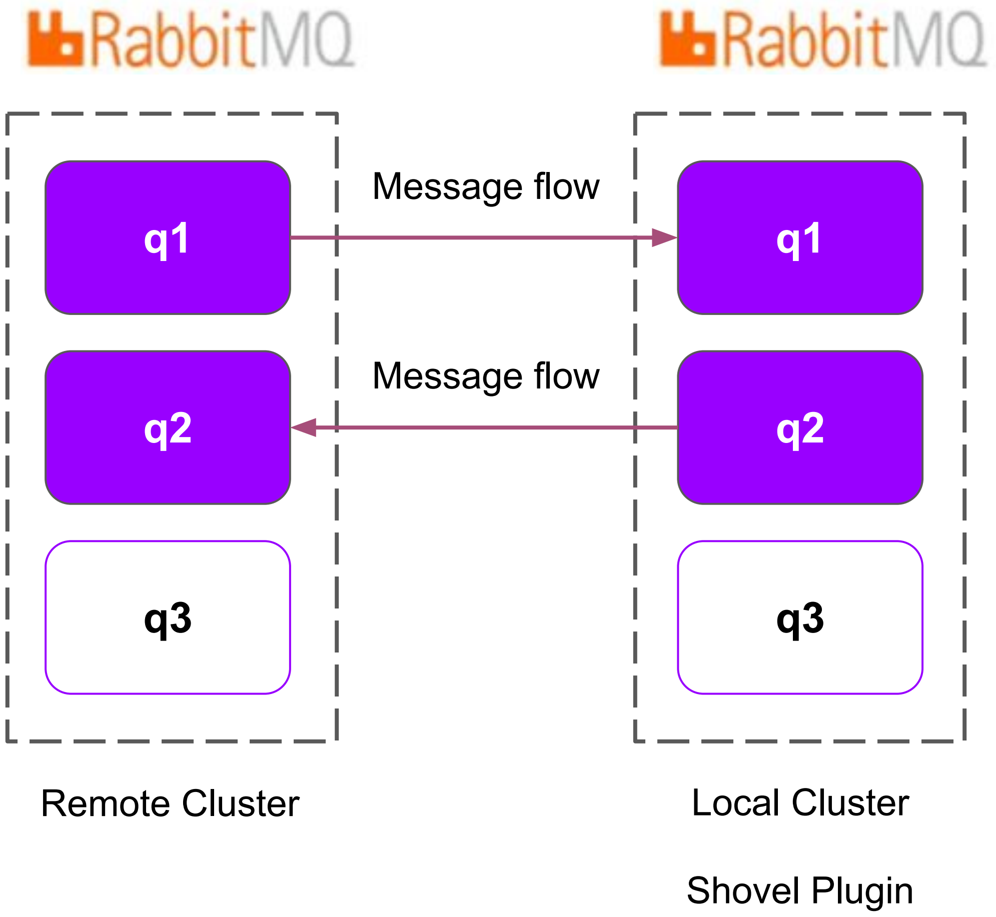

RabbitMQ
RabbitMQ
q1
q2
q3
Message flow
Message flow
q1
q2
q3
Consumer
Consumer
Upstream
Downstream

Shovel Overview

Moves messages from a source to a destination

Can move messages between brokers (or clusters) in different geographic or administrative domains

The shovel plugin is designed to tolerate intermittent connectivity


RabbitMQ
RabbitMQ
Message flow
q1
q1
Message flow
q2
q2
q3
q3
Remote Cluster
Local Cluster
Shovel Plugin

MQTT Overview

The Standard for IoT Messaging

It is designed as an extremely lightweight publish/subscribe messaging transport

It is ideal for connecting remote devices with a simple code and minimal network bandwidth

MQTT can scale to connect with millions of IoT devices

Support for Unreliable Networks

Reliable Message Delivery

MQTT Products

HIVEMQ

EMQ

RabbitMQ

mosquitto

mosquitto

MQTT Plugin

RabbitMQ supports MQTT 3.1.1 via a plugin that ships in the core distribution

The plugin must be enabled on all cluster nodes

Supported MQTT 3.1.1 features

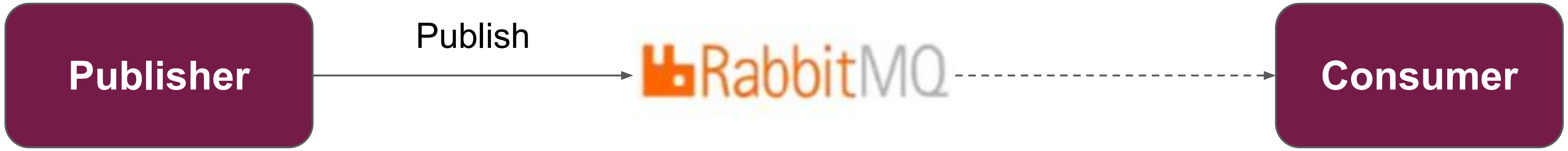
- QoS0 and QoS1 publish & consume
- QoS2 publish (downgraded to QoS1)
- Last Will and Testament (LWT)
- TLS
- Session stickiness
- Retained messages with pluggable storage backends

RabbitMQ does not support QoS2 subscriptions

MQTT QoS X

MQTT QoS0

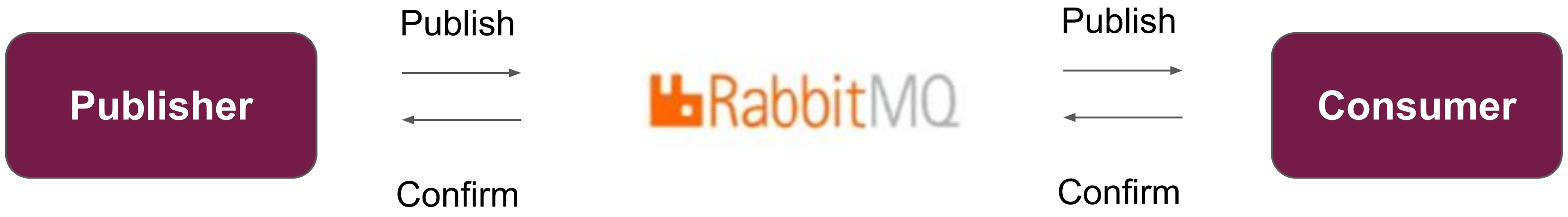
QoS0 ensures that a message is sent only once but doesn't guarantee delivery



MQTT QoS X

MQTT QoS1

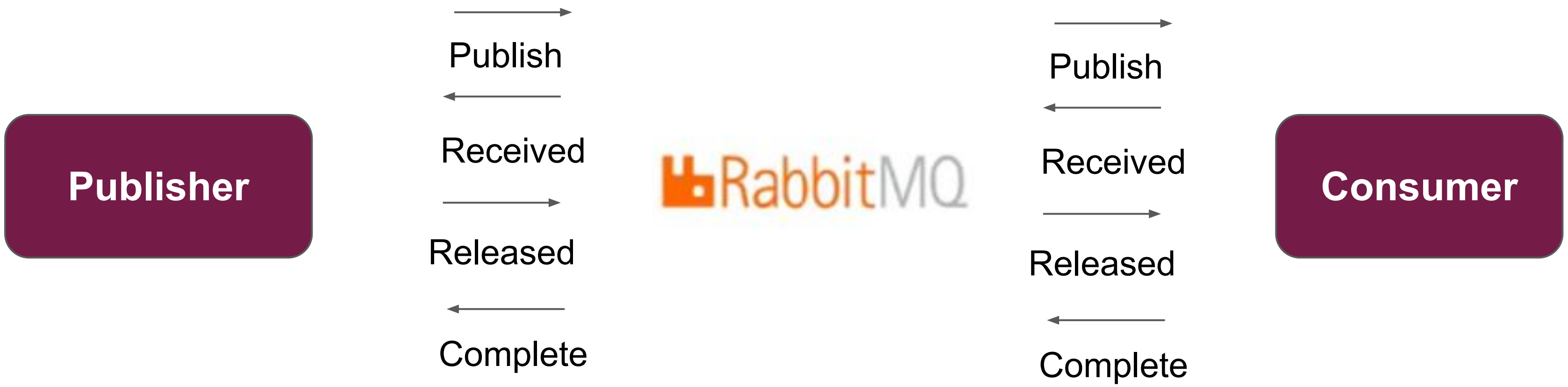
QoS1 ensures that a message will be delivered and received at least once



MQTT QoS X

MQTT QoS2

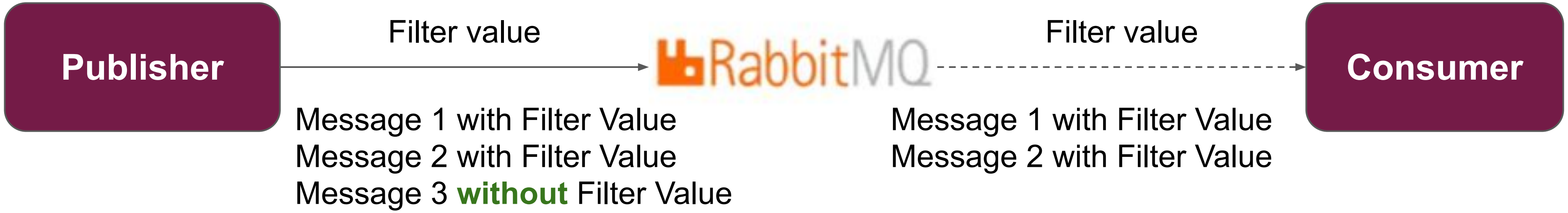
QoS 2 ensures that subscribers receive a message exactly once



MQTT Plugin

```
rabbitmq-plugins enable rabbitmq_mqtt
```

Stream Filtering


Publisher
Filter value
RabbitMQ
Filter value
Consumer
Message 1 with Filter Value
Message 2 with Filter Value
Message 3 **without** Filter Value
Message 1 with Filter Value
Message 2 with Filter Value

Stream Overview

RabbitMQ Streams is a persistent replicated data structure

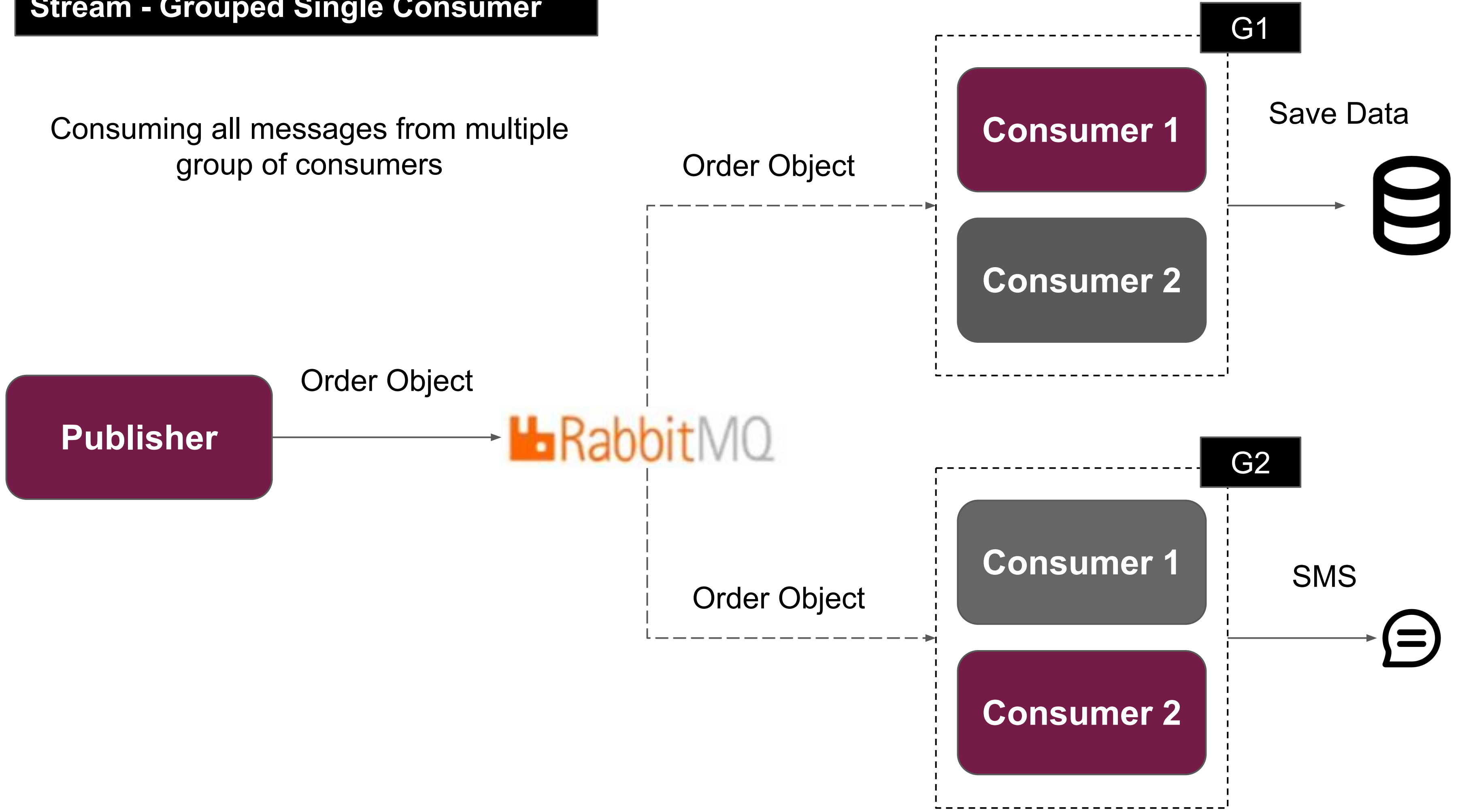
Streams model an append-only log of messages that can be repeatedly read until they expire

Streams Use Cases:

- Large fan-outs
- Replay (Time-travelling)
- Throughput Performance
- Large backlogs

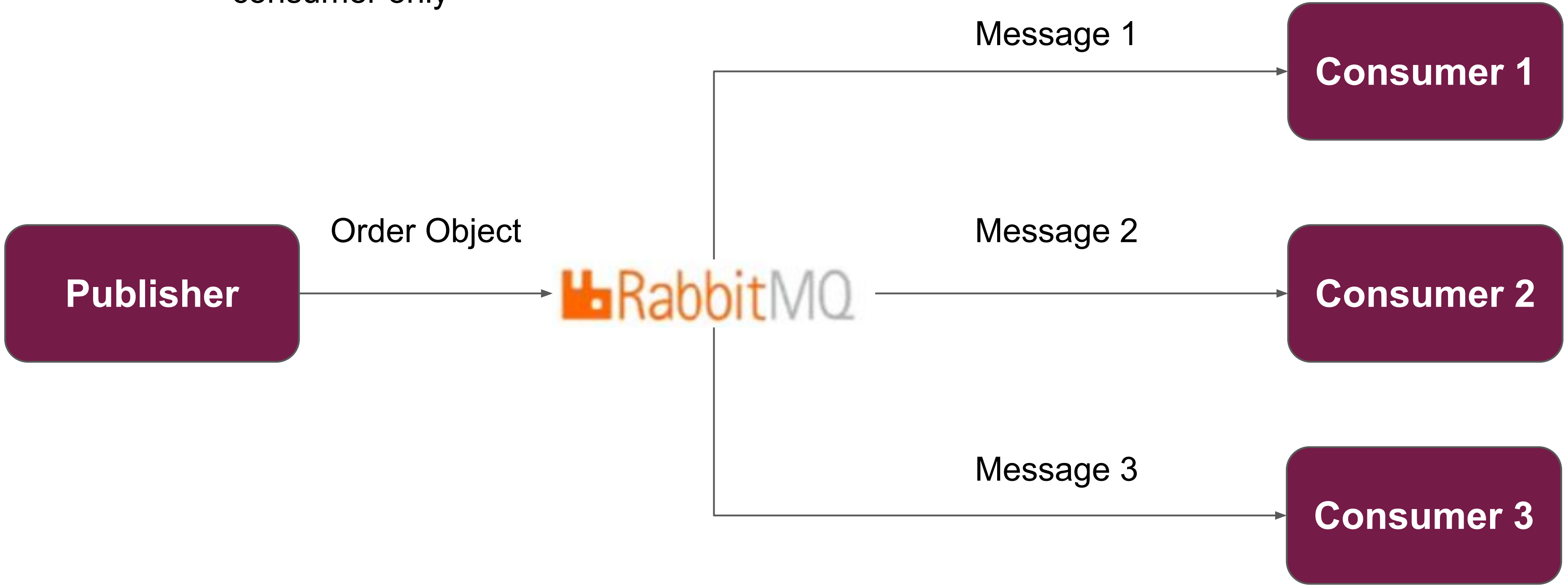
Stream - Grouped Single Consumer

Consuming all messages from multiple group of consumers



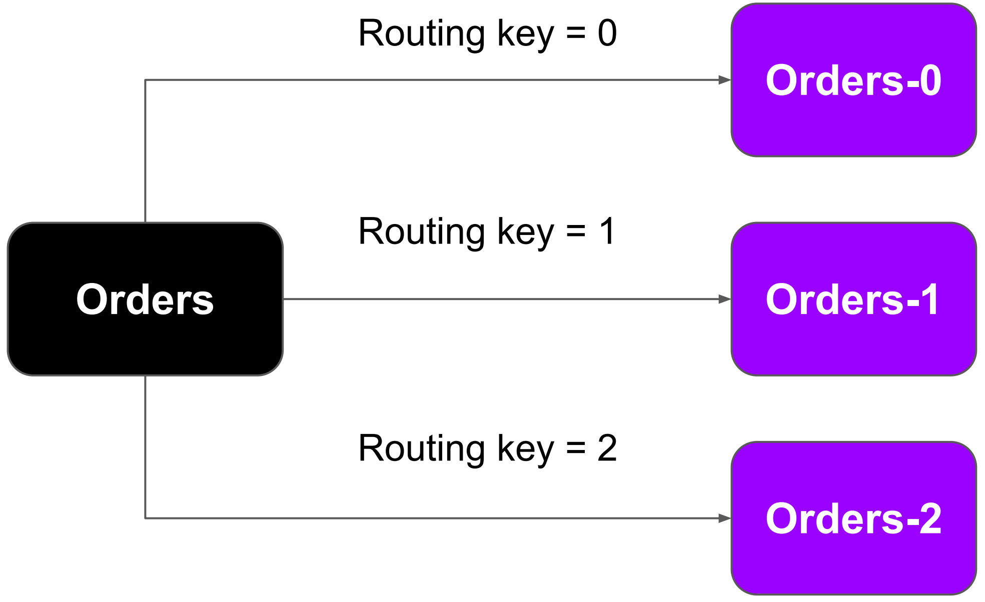
Super Streams

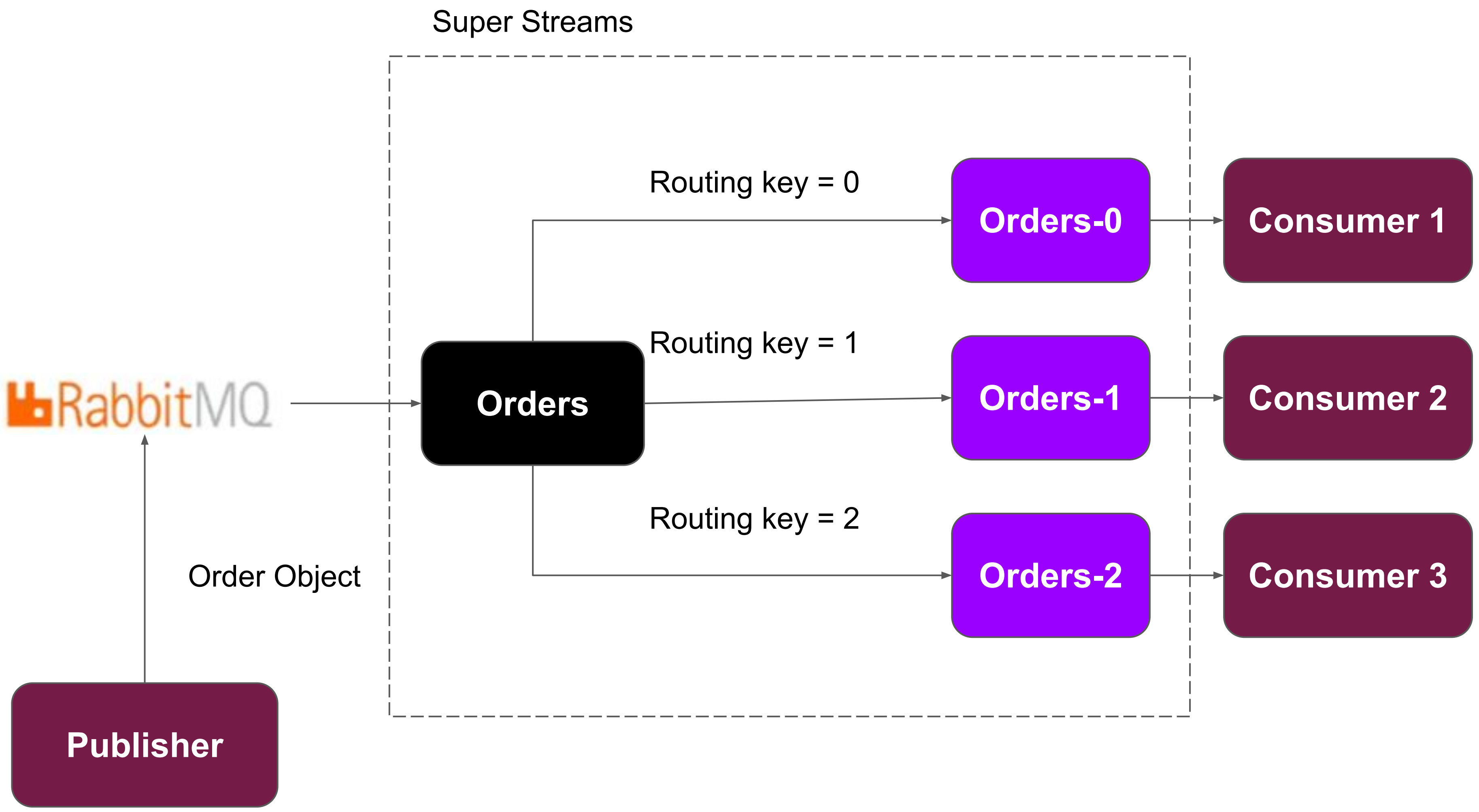
Consuming the same message from one consumer only



Super Streams

```
rabbitmq-streams add_super_stream orders –partitions 3

rabbitmq-streams delete_super_stream orders
```




Super Streams
Routing key = 0
Routing key = 1
Routing key = 2
Orders
Orders-0
Orders-1
Orders-2
Consumer 1
Consumer 2
Consumer 3
RabbitMQ
Order Object
Publisher