RabbitMQ™

A somewhat in-depth look

Overview

RabbitMQ History & Facts

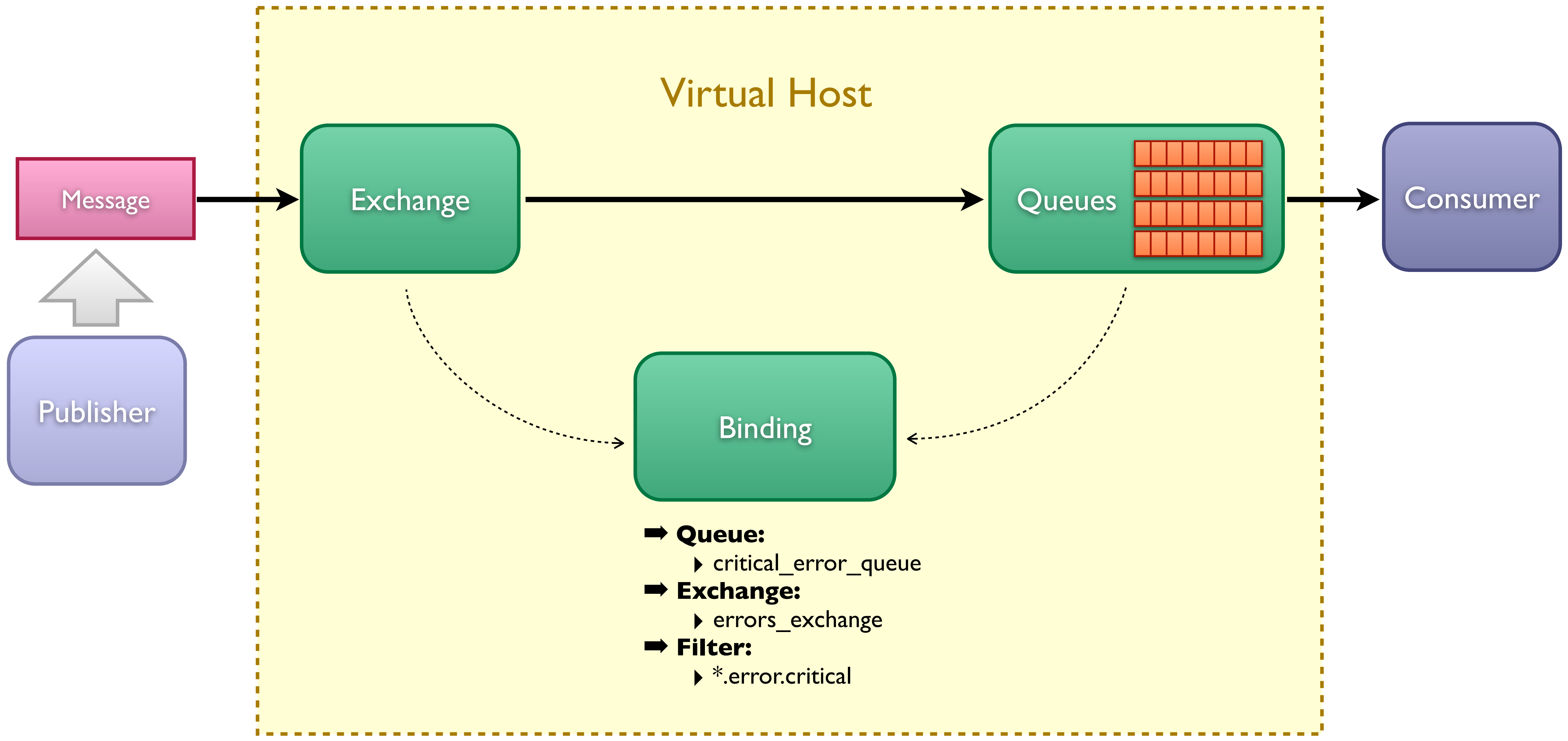
- Idea was born around the same time AMQP was first drafted by JP Morgan Chase
- First released in 2006
- Acquired by SpringSource in 2010 (now part of VMWare)
- First version built in under 3 months!
- Written in Erlang/OTP

RabbitMQ History & Facts

- Used by quite a few major projects including
  - Huffington Post (Huff Live)
  - Indeed (Job Aggregation service)
  - VMWare (vFabric)
  - SoundCloud (transcoding service)
  - Mozilla (Pulse)

The Basics

AMQ Model Recap



Final overview of what the AMQ Model is about – in a nutshell...
 * all within a virtual host (completely isolated)
 * this is protocol at the server's service level

AMQ Model Recap

Publisher

- can create exchanges and queues

Exchange

- routes messages based on criteria
- doesn’t store messages
- can inspect message content
- can be created at runtime consumers

Queues

- store messages
- named
- bound-able to exchange
- criteria
- can be created at runtime by consumers

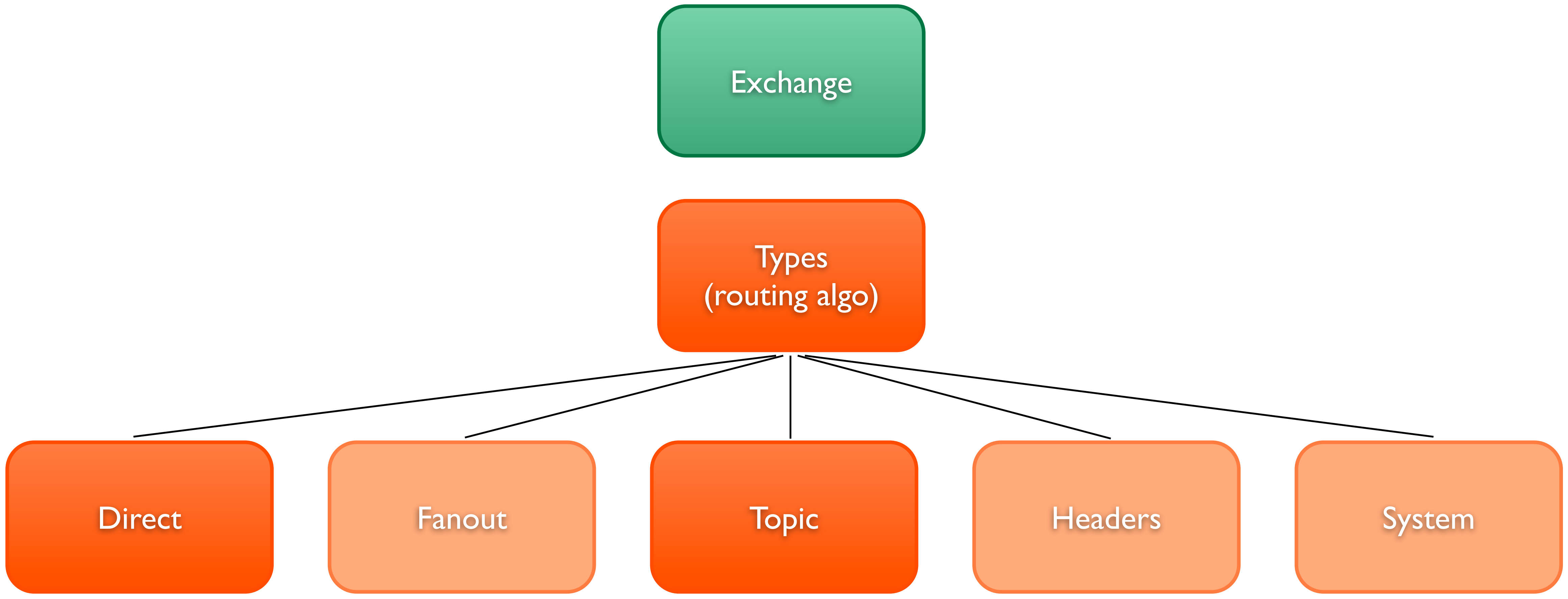
Binding

- creates a relationship between queues and exchanges
- contains criteria and properties
- can be created at runtime by consumers

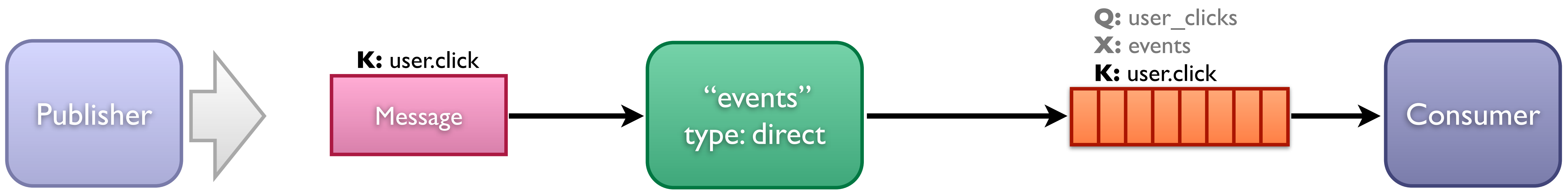
Consumer

- can create exchanges and queues

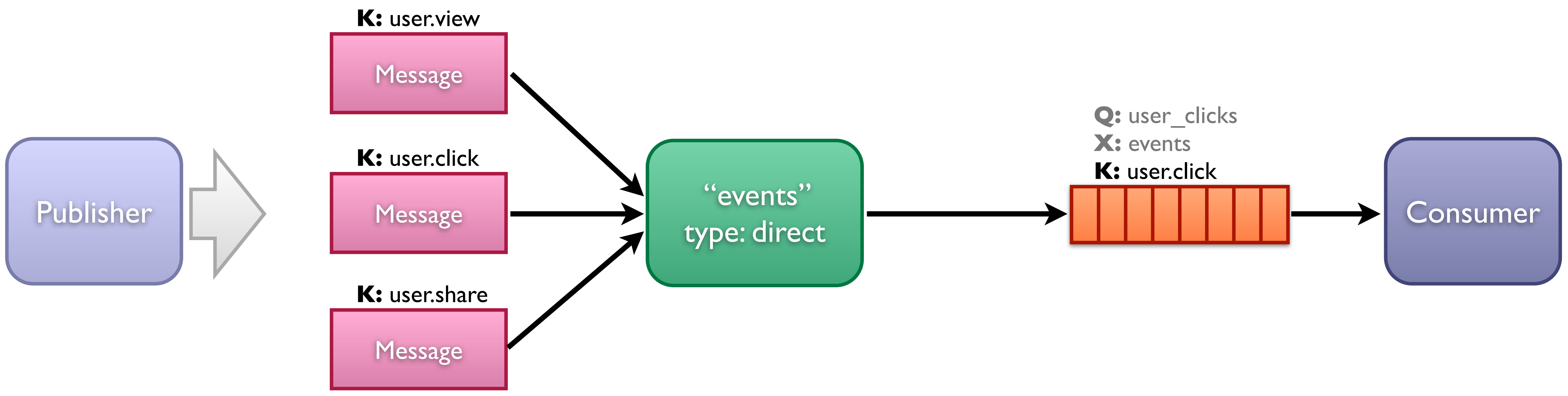
Exchange Types



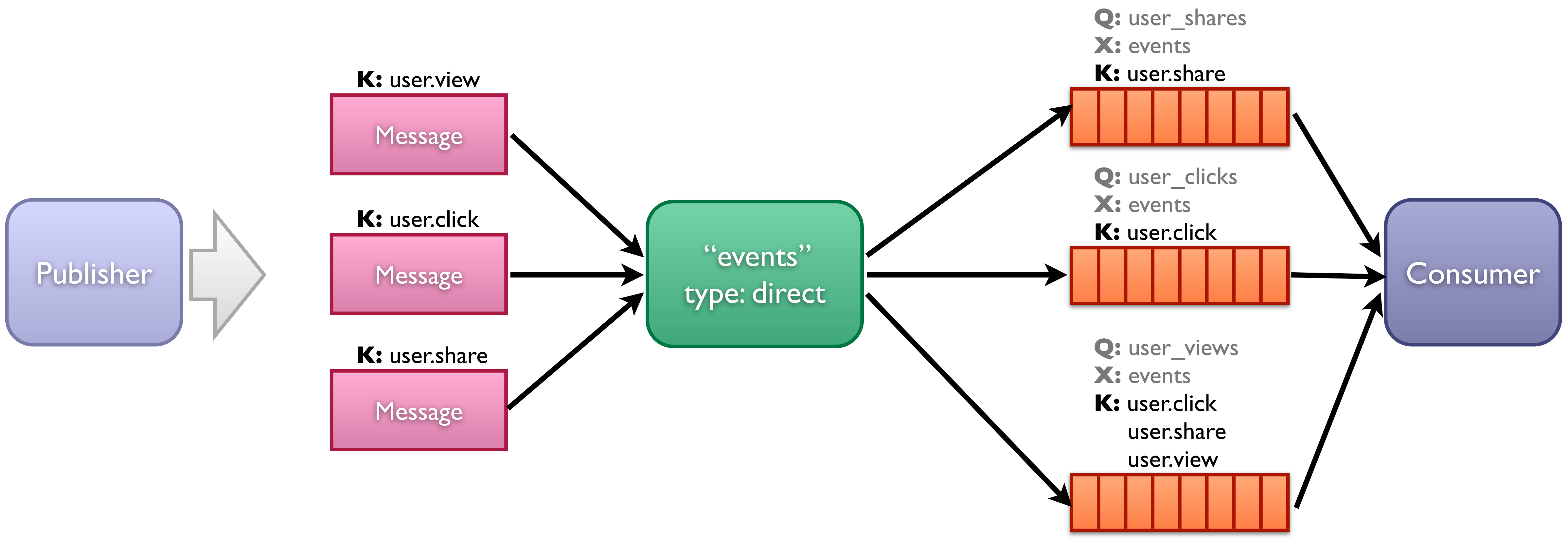
Direct Exchange



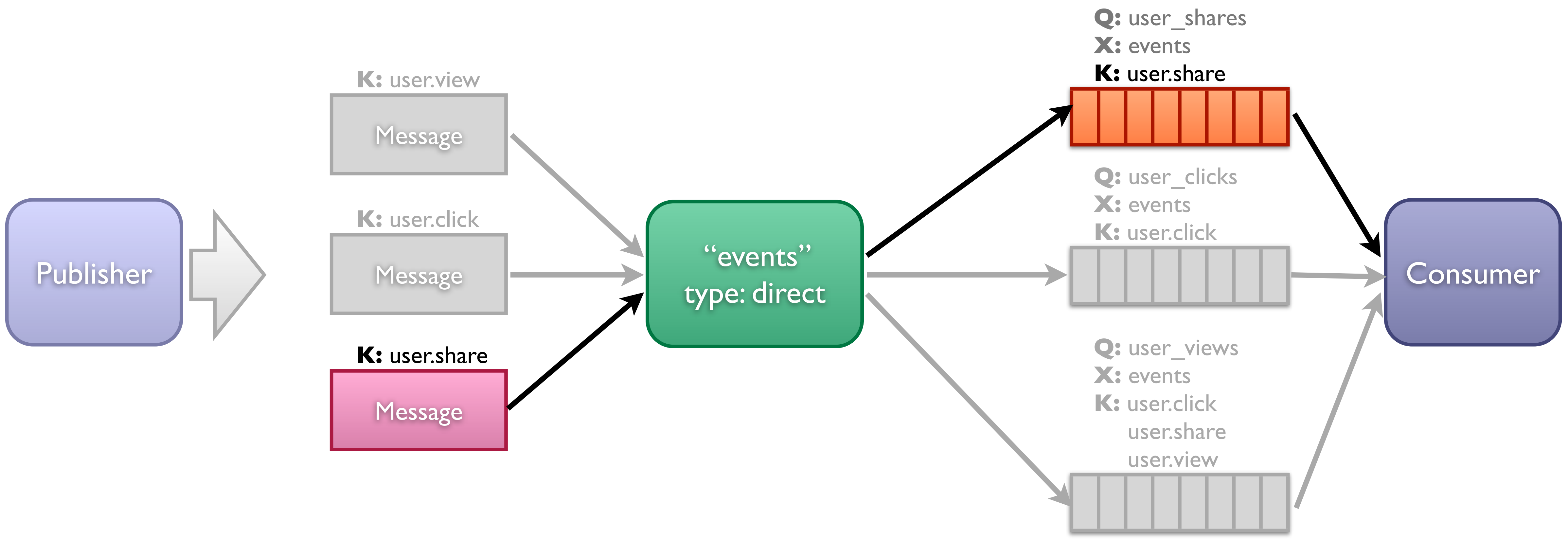
Direct Exchange



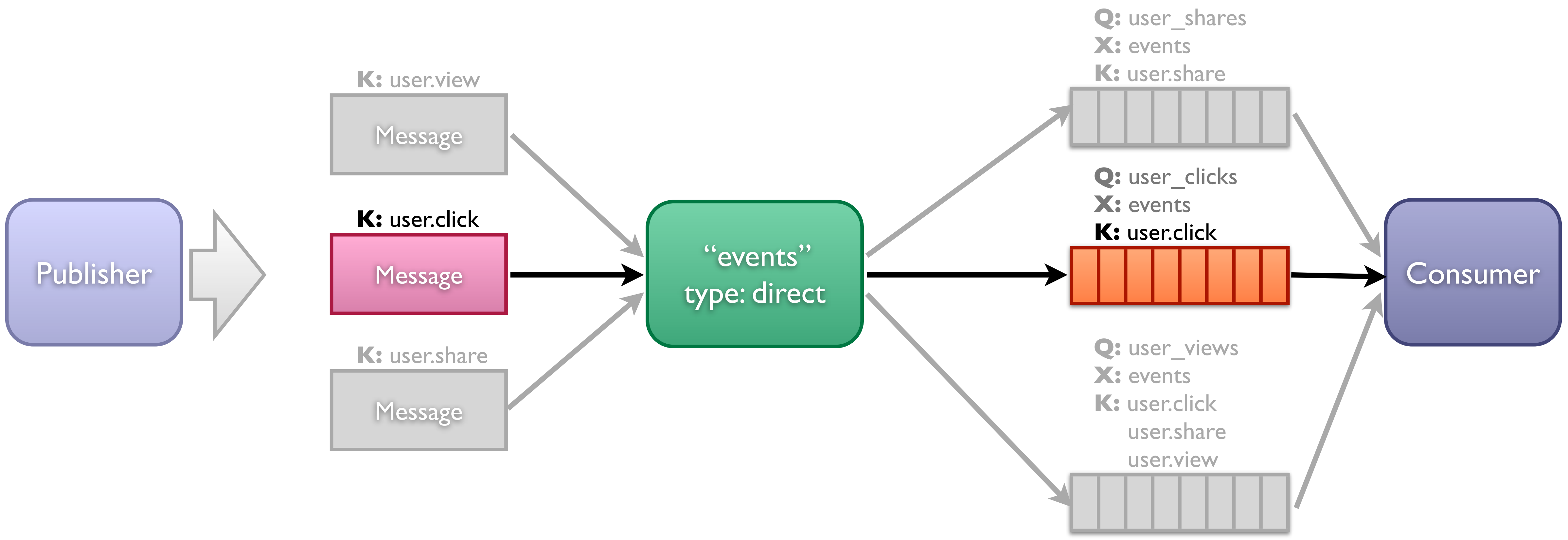
Direct Exchange



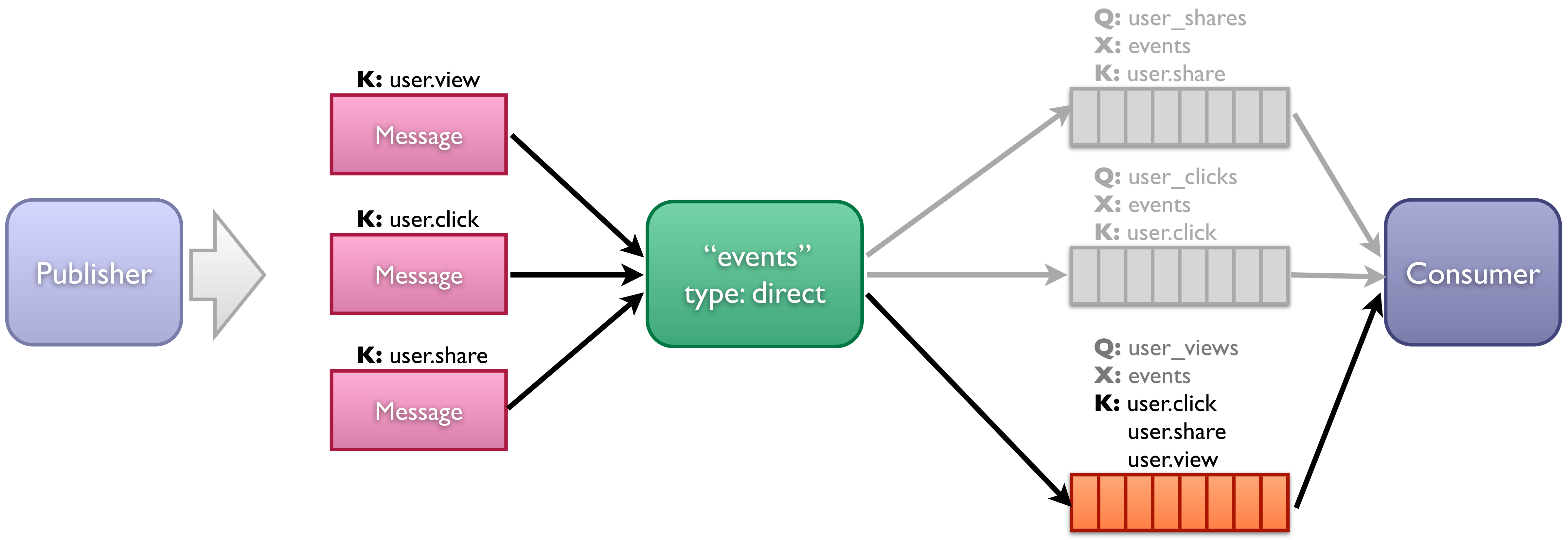
Direct Exchange



Direct Exchange



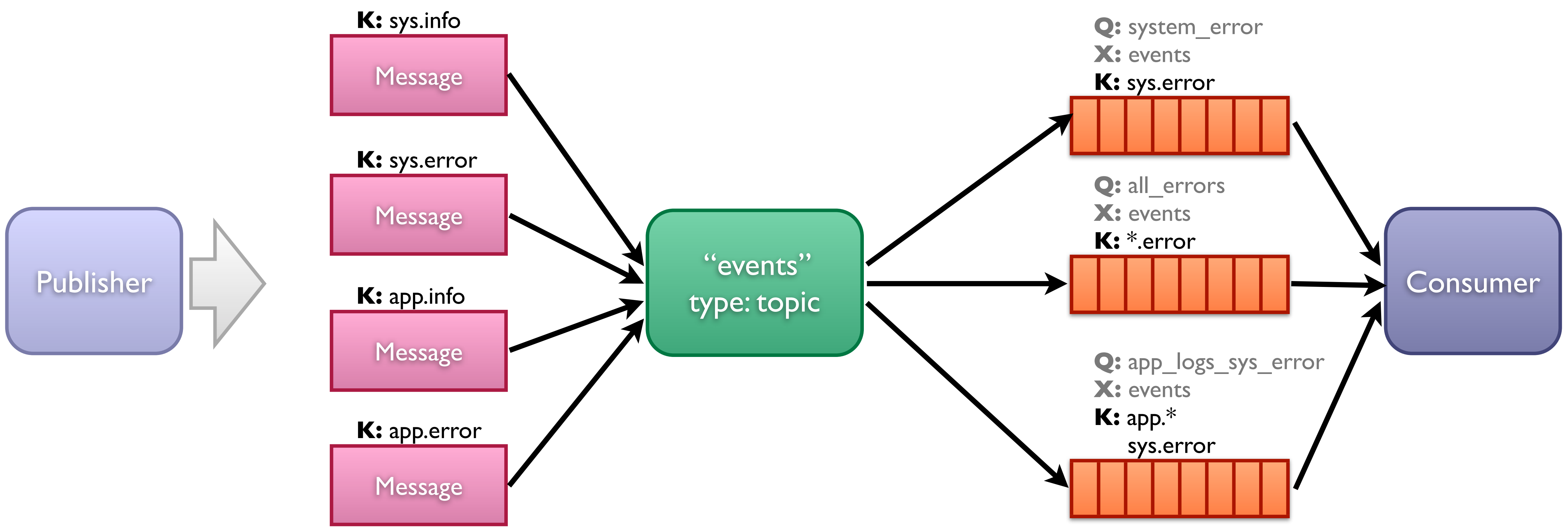
Direct Exchange



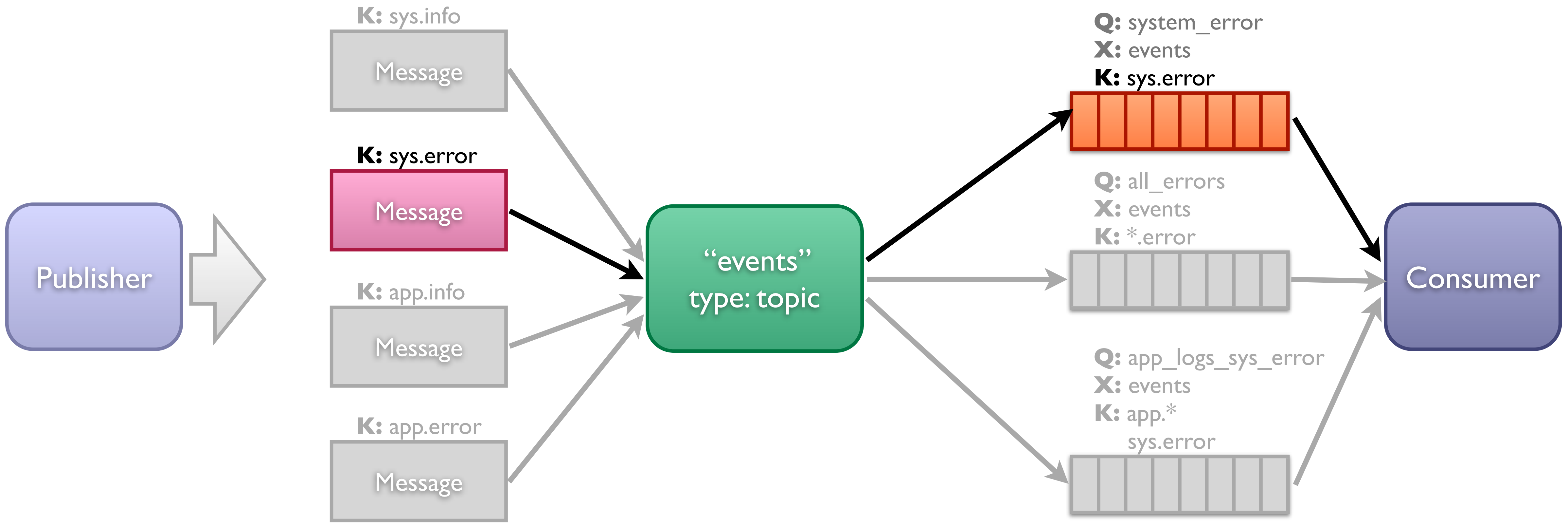
Direct Exchange

Walkthrough and demo

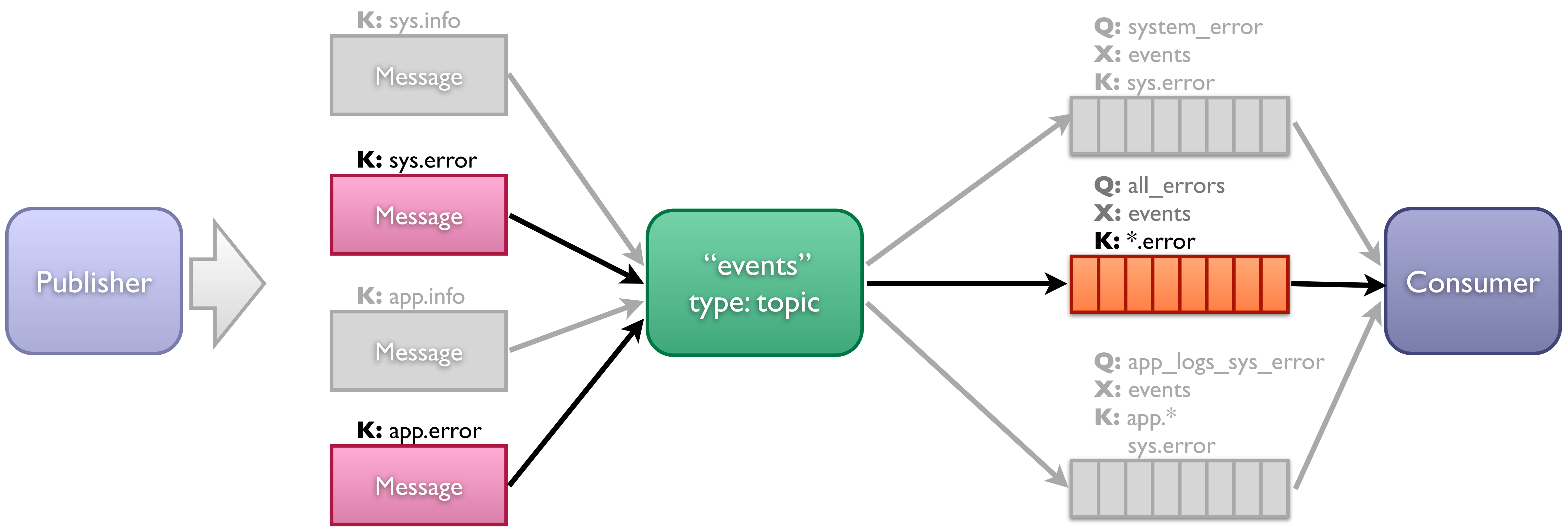
Topic Exchange



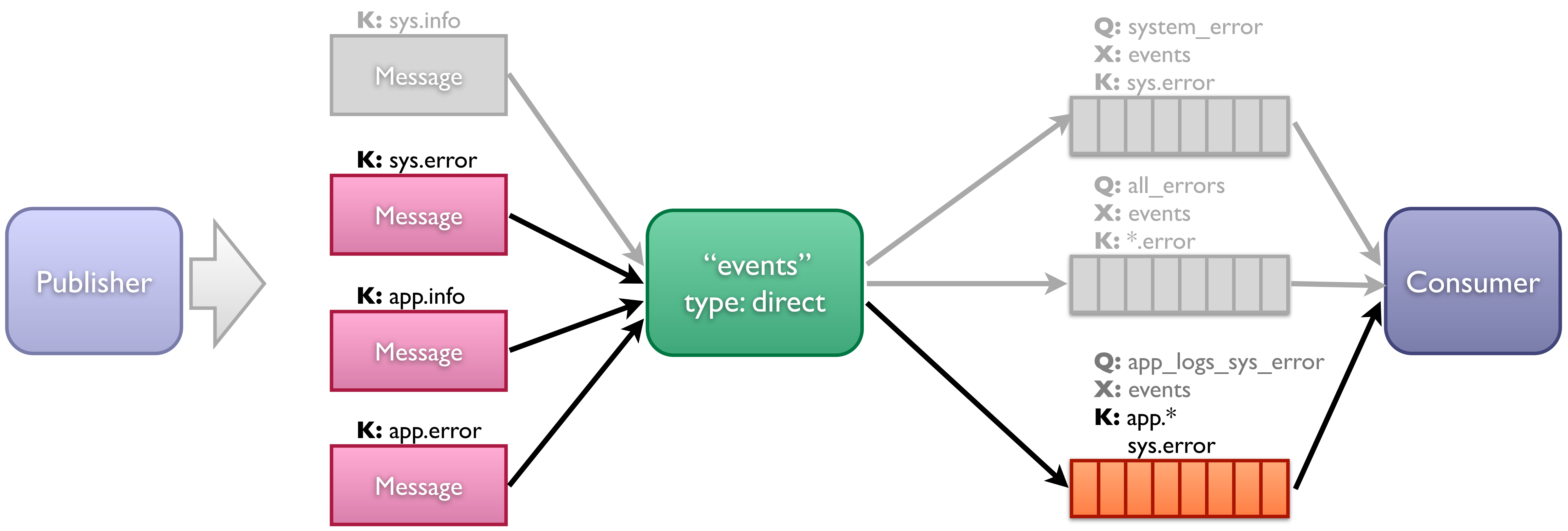
Topic Exchange



Topic Exchange



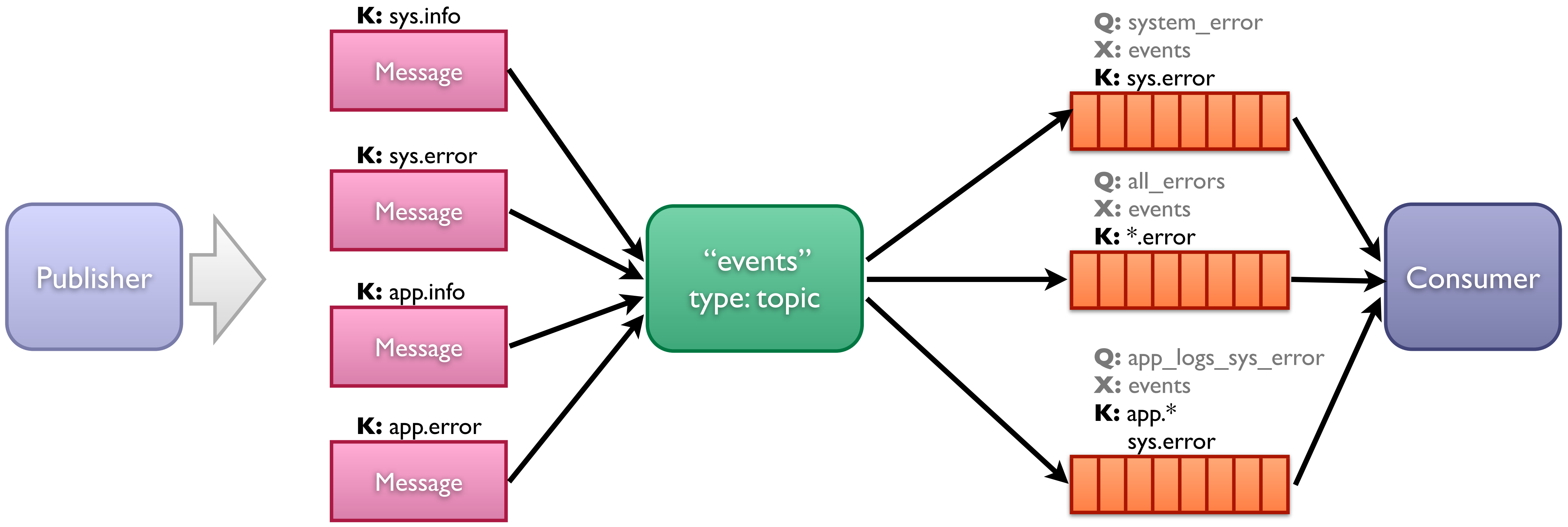
Topic Exchange



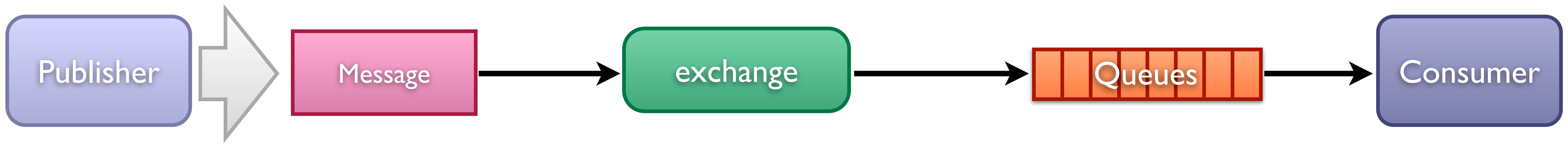
Topic Exchange

Walkthrough and demo

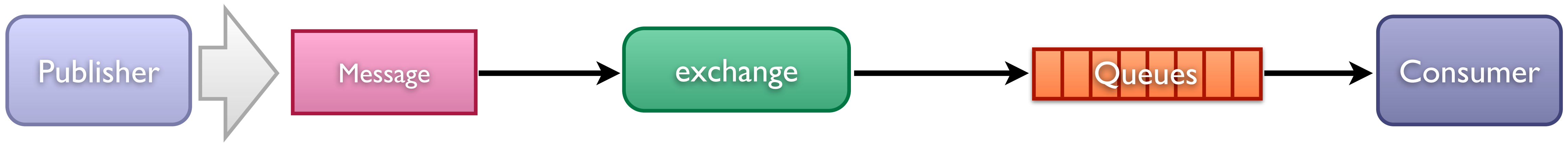
Topic Exchange



Persistence & Durability

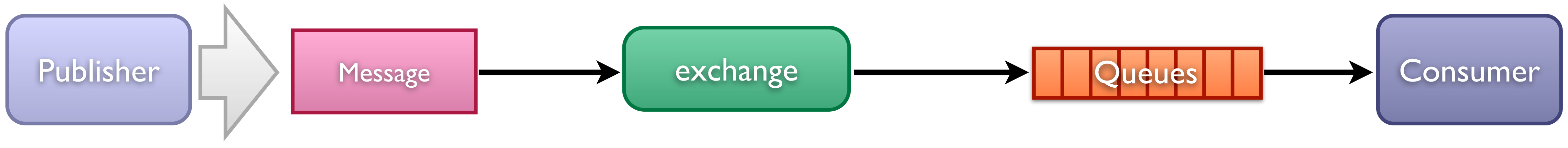


Persistence & Durability



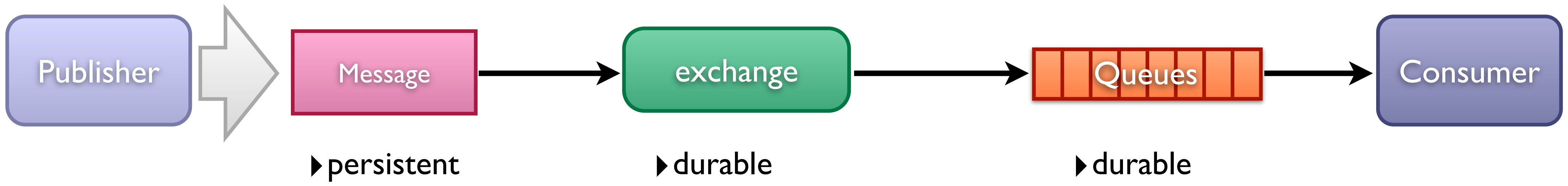
- persistent

Persistence & Durability

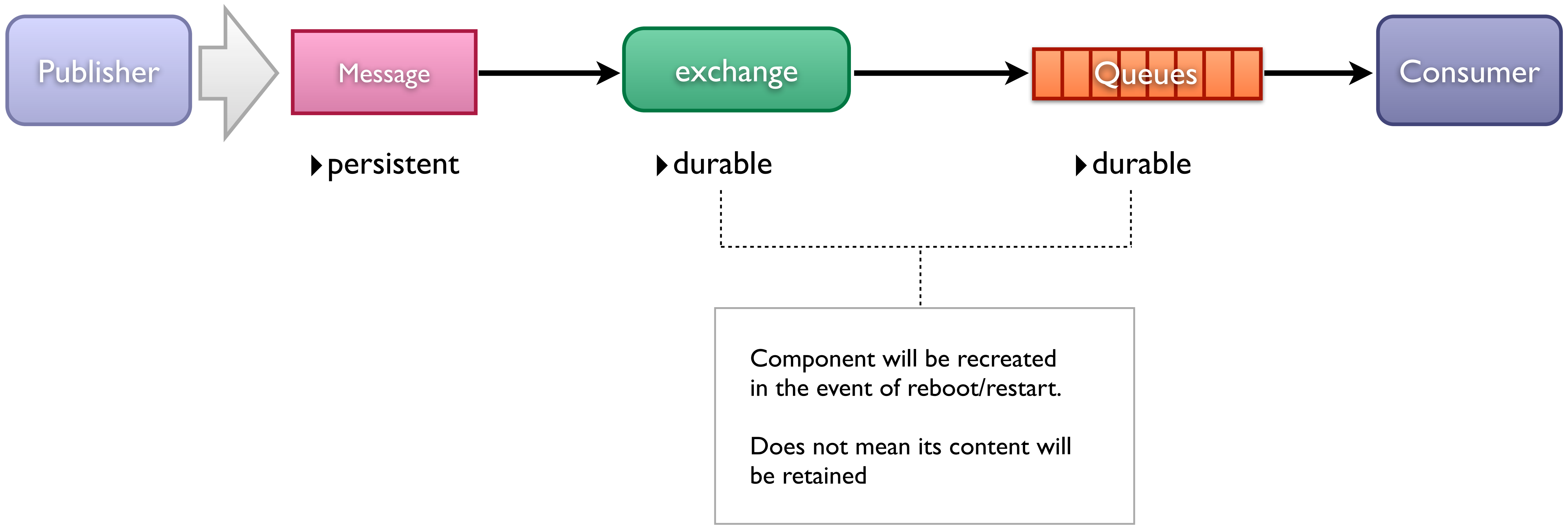


- persistent
- durable

Persistence & Durability



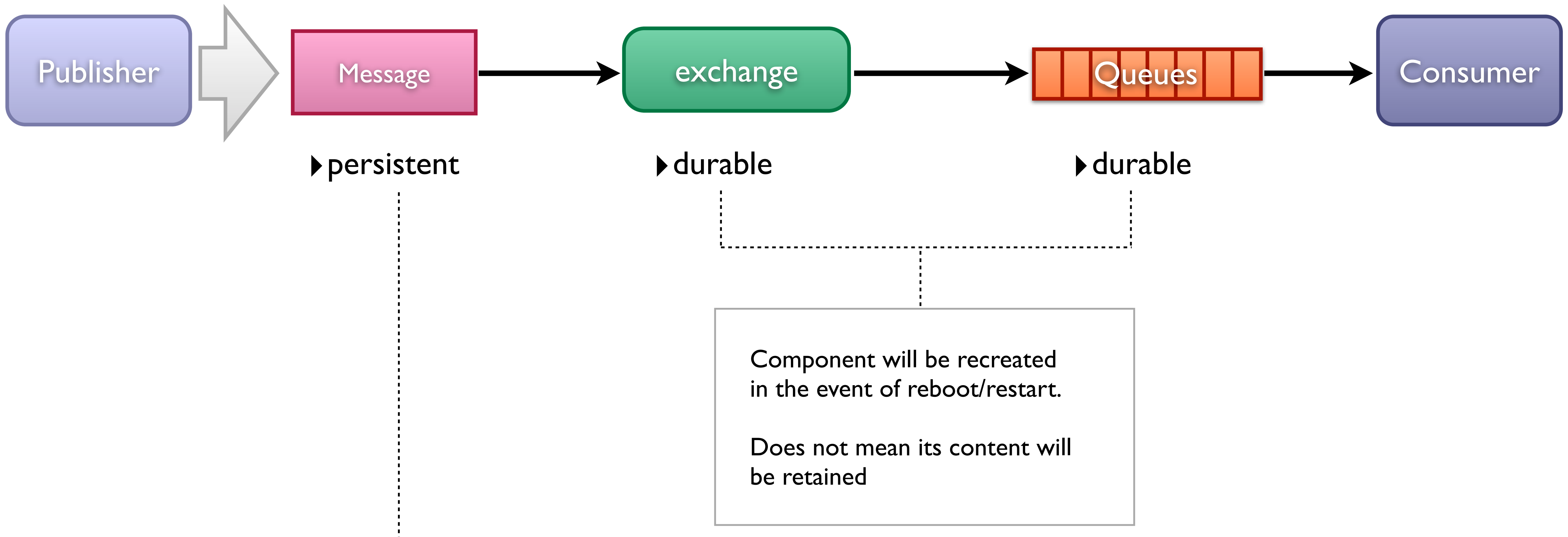
Persistence & Durability



Component will be recreated in the event of reboot/restart.

Does not mean its content will be retained

Persistence & Durability



Component will be recreated in the event of reboot/restart.

Does not mean its content will be retained

Messages will be persisted to disk.

In RabbitMQ, messages are actually queued in memory before written to disk and f'syncd

For recoverable messages, **queue(s) must be durable**

Administration

Administration

- Management tools via rabbitmq-plugins
- Command:
  `sudo rabbitmq-plugins enable rabbitmq_management`
- Gives you:
  - `rabbitmqctl`
  - MochiWeb:
    `http://localhost:15672`

Administration

- Important tools:
  - `rabbitmqctl` commands
    - users, perms, queues, bindings
    - status, reset
  - MochiWeb features

Distributed RabbitMQ

Clustering & Replication

Distributed RabbitMQ

Clustering

Replication

HA

Distributed RabbitMQ

Clustering

Replication

HA

➡ Acts as a single logical unit
➡ Consumer sees cluster as single node
➡ Nodes shares the same Erlang cookie
➡ Automatic metadata replication, etc
➡ Requires reliable LAN-like environment
➡ Emphasis on Consistency & Availability

Distributed RabbitMQ

Clustering

- ➡ Acts as a single logical unit
- ➡ Consumer sees cluster as single node
- ➡ Nodes shares the same Erlang cookie
- ➡ Automatic metadata replication, etc
- ➡ Requires reliable LAN-like environment
- ➡ Emphasis on Consistency & Availability

Replication

- ➡ Acts like a forwarding agent
- ➡ Nodes doesn't share any cookie and is standalone
- ➡ Uses AMQP protocol itself for forwarding
- ➡ Suitable for unreliable WAN connections
- ➡ Emphasis on Availability & Partition Tolerance

HA

Distributed RabbitMQ

Clustering

- ➡ Acts as a single logical unit
- ➡ Consumer sees cluster as single node
- ➡ Nodes shares the same Erlang cookie
- ➡ Automatic metadata replication, etc
- ➡ Requires reliable LAN-like environment
- ➡ Emphasis on Consistency & Availability

Replication

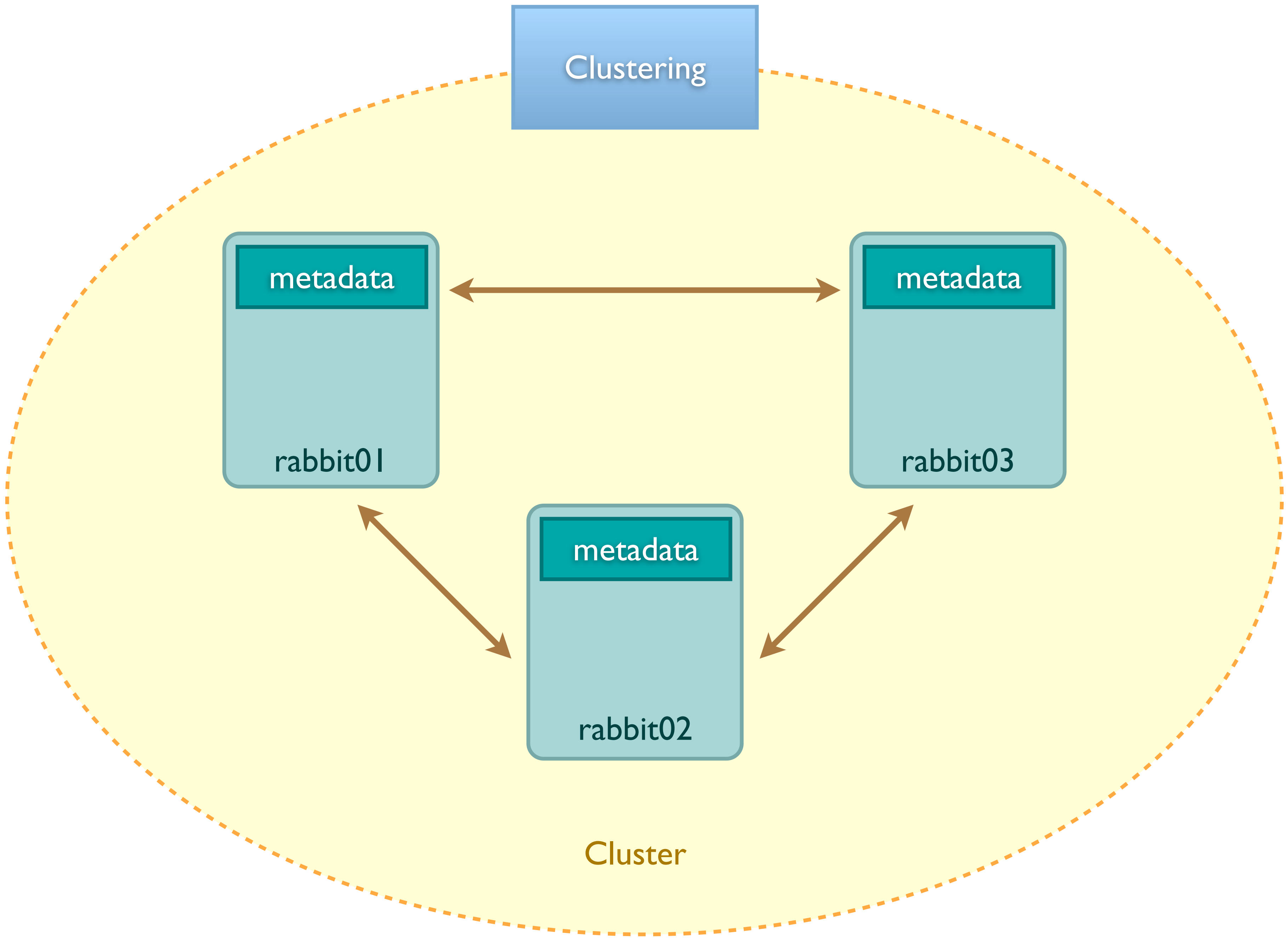
- ➡ Acts like a forwarding agent
- ➡ Nodes doesn't share any cookie and is standalone
- ➡ Uses AMQP protocol itself for forwarding
- ➡ Suitable for unreliable WAN connections
- ➡ Emphasis on Availability & Partition Tolerance

HA

- ➡ A combo of both
- ➡ It's cluster with replication
- ➡ Usually in form of master/ slave
- ➡ Emphasis on Consistency & Availability

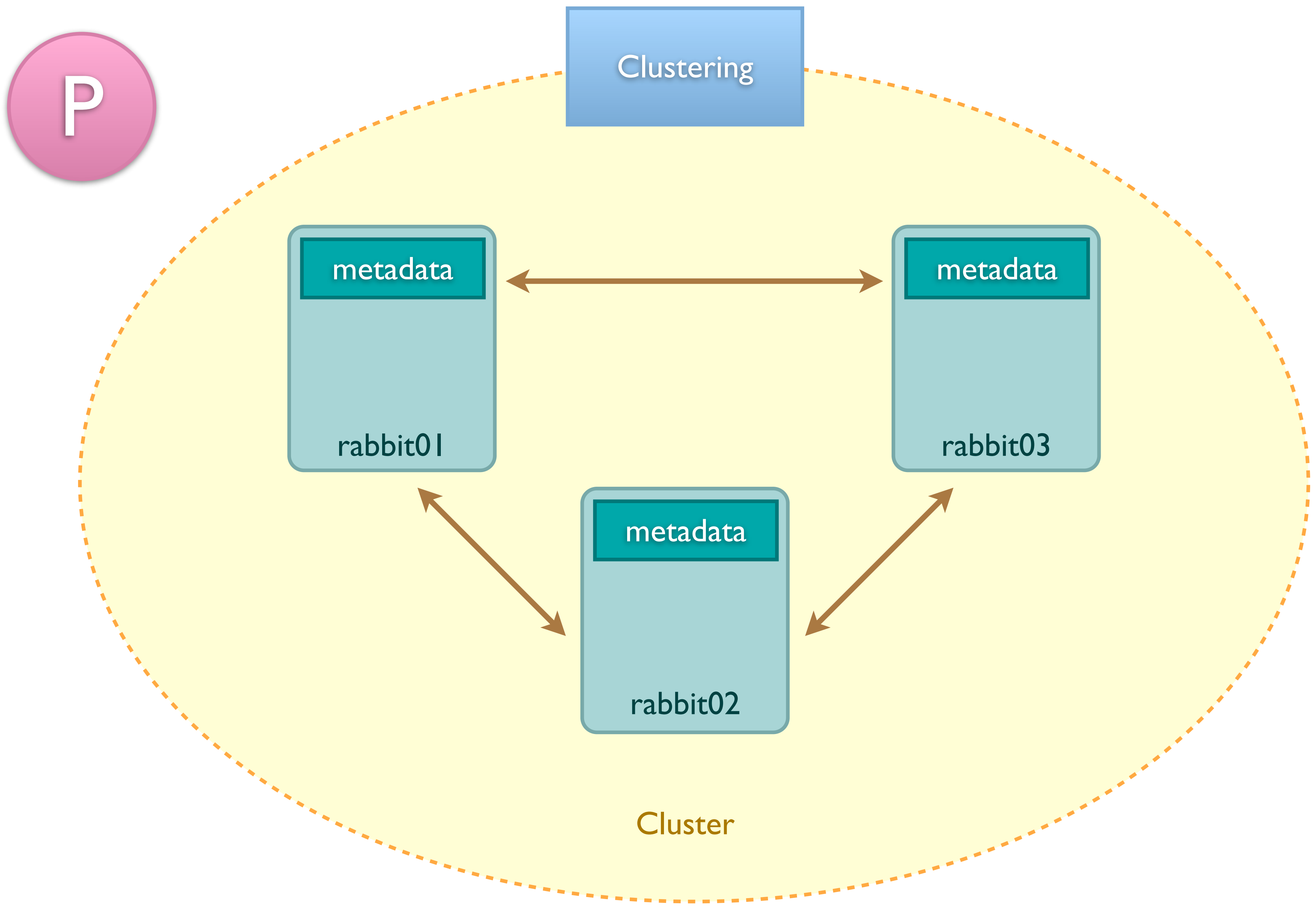
Clustering

Clustering



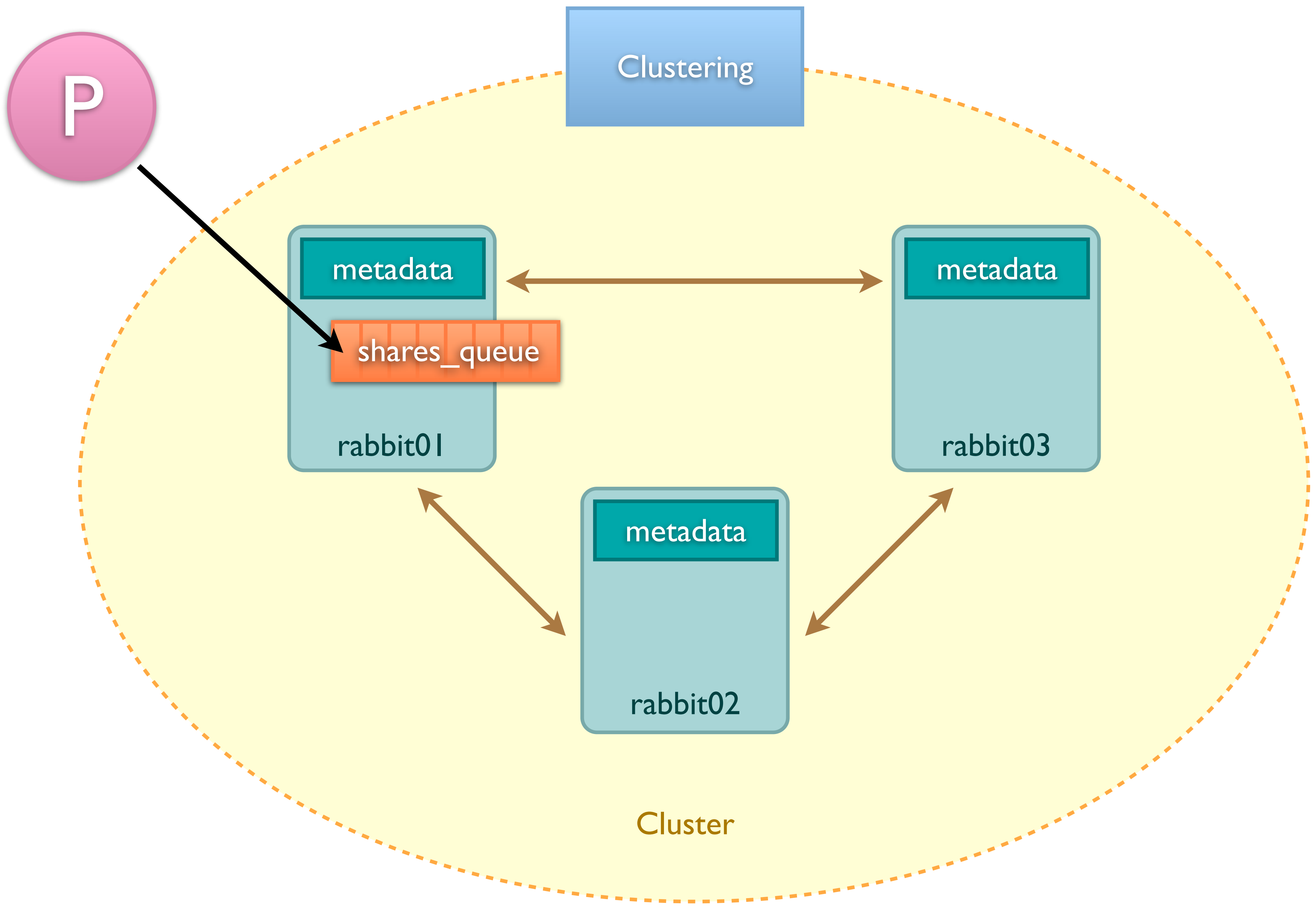
Typical clustering
Metadata is replicated across all nodes (operations, queues, exchange and bindings declaration, etc)
Let’s see what happens when a queue is created...
Let’s say we have a publisher...

Clustering



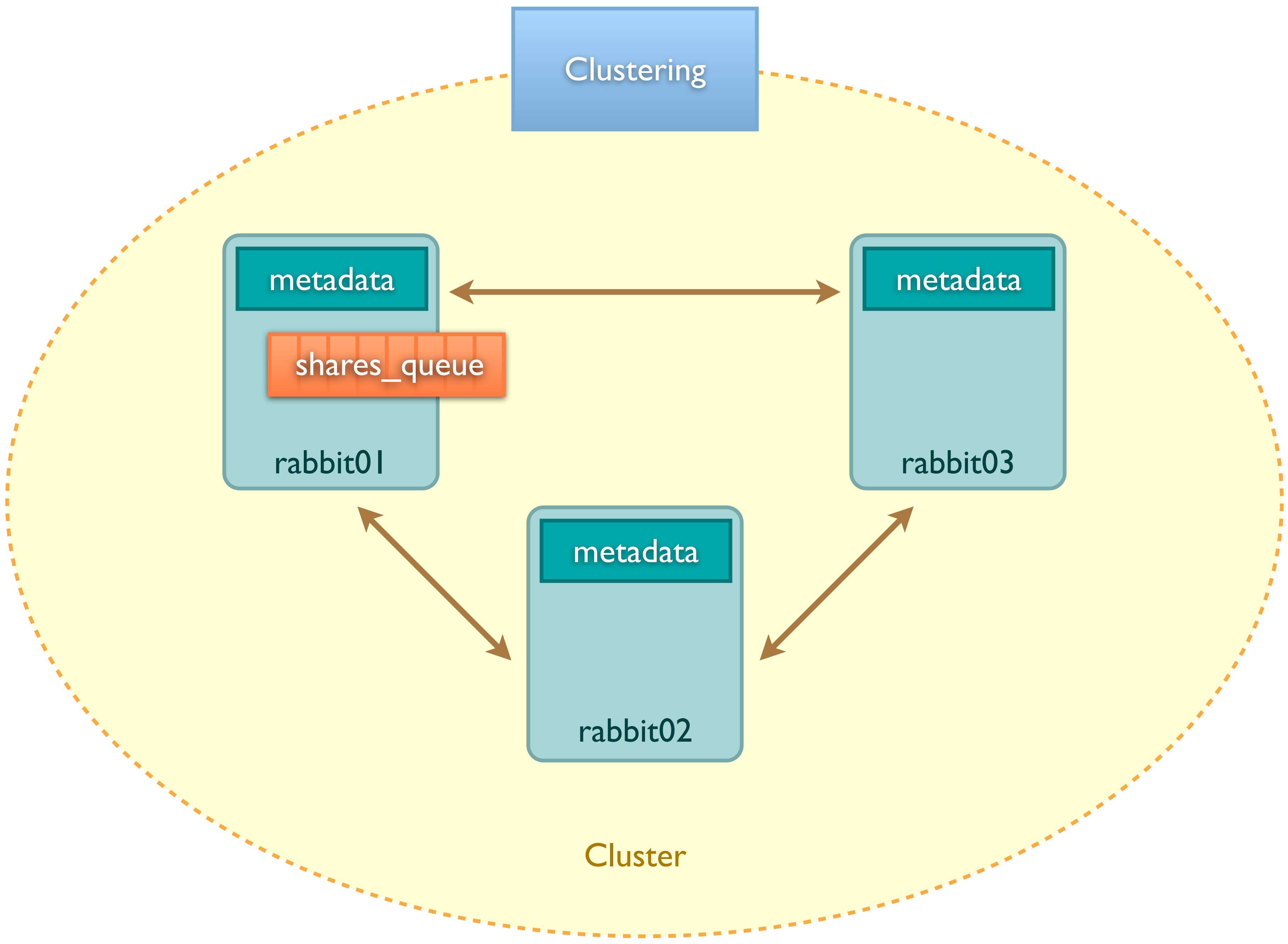
and this is the first time it's publishing message to a newly declared queue

Clustering



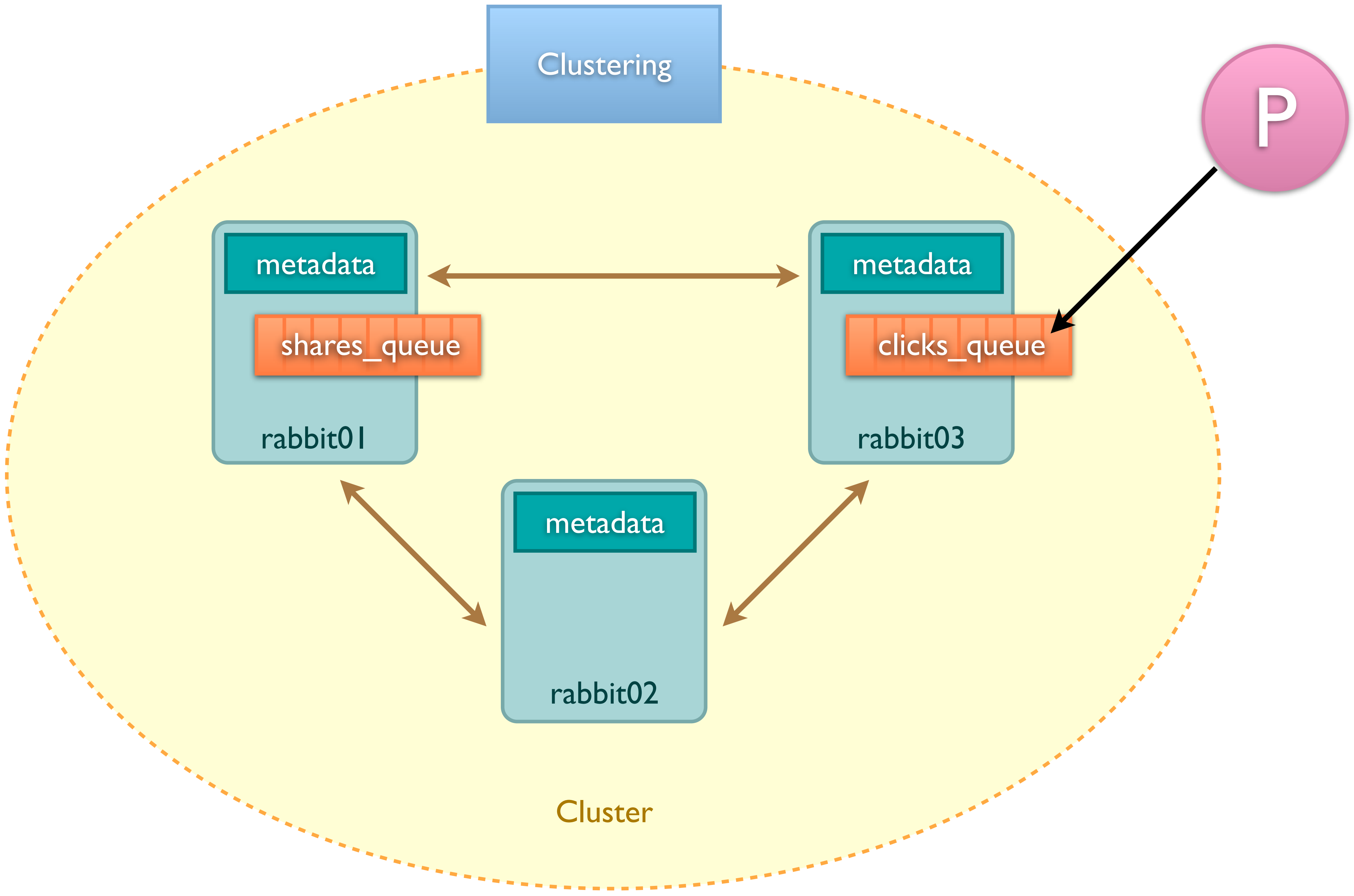
Through a LB, the connection is established with rabbit01.
Publish the message, which creates a queue and then message to it.
What should happen?

Clustering



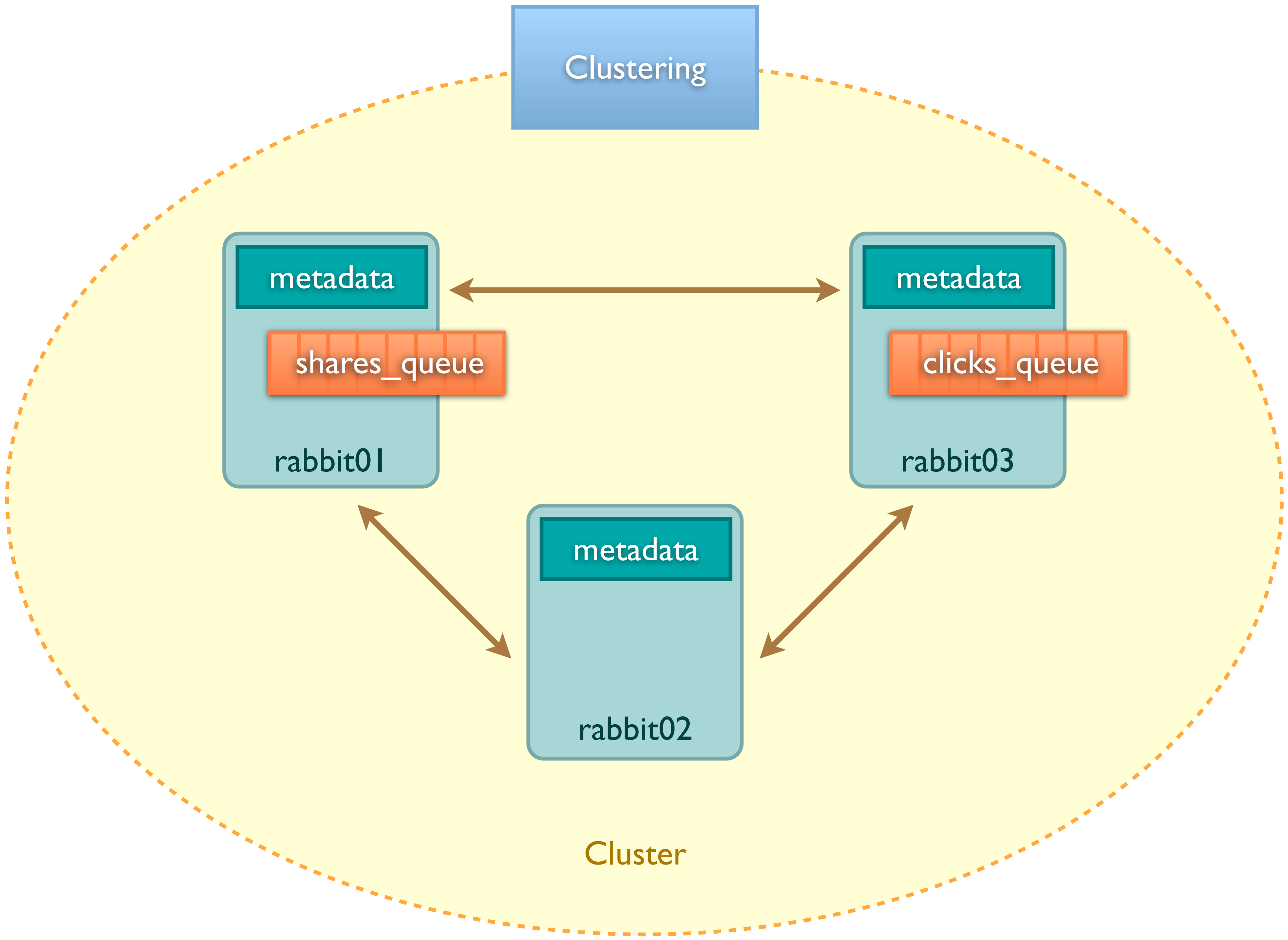
Queues are created on the node called by consumer/publisher and not replicated
BUT the metadata is replicated.

Clustering



Queues are created on the node called by consumer/publisher and not replicated

Clustering


Clustering
metadata
shares_queue
rabbit01
metadata
clicks_queue
rabbit03
metadata
rabbit02
Cluster

Clustering Notes

- There are three types of nodes:
    - Stat, Disk and Ram
- All nodes must share the same cookie
- Works like a shard by default, thus adding new node to cluster improves performance
- Upgrade is not automated
- Best have one stat-node with WebUI, rest with agents

Important

- RabbitMQ was not designed to handle network partition.
- Nodes in clusters should have LAN-like reliability

Clustering

Walkthrough and demo

Distributed RabbitMQ

Clustering

- ➡ Acts as a single logical unit
- ➡ Consumer sees cluster as single node
- ➡ Nodes shares the same Erlang cookie
- ➡ Automatic metadata replication, etc
- ➡ Requires reliable LAN-like environment
- ➡ Emphasis on Consistency & Availability

Replication

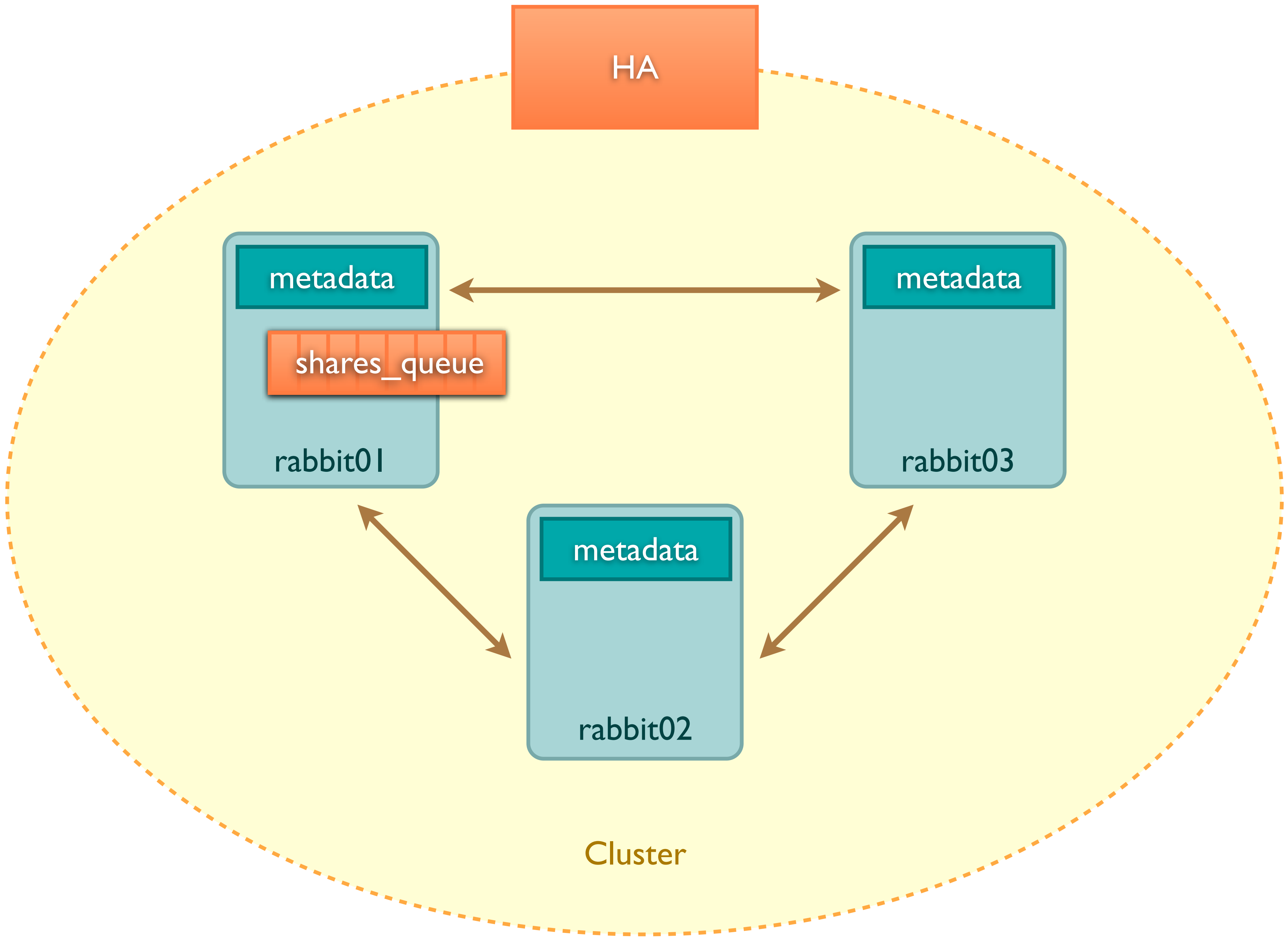
- ➡ Acts like a forwarding agent
- ➡ Nodes doesn't share any cookie and is standalone
- ➡ Uses AMQP protocol itself for forwarding
- ➡ Suitable for unreliable WAN connections
- ➡ Emphasis on Availability & Partition Tolerance

HA

- ➡ A combo of both
- ➡ It's cluster with replication
- ➡ Usually in form of master/ slave
- ➡ Emphasis on Consistency & Availability

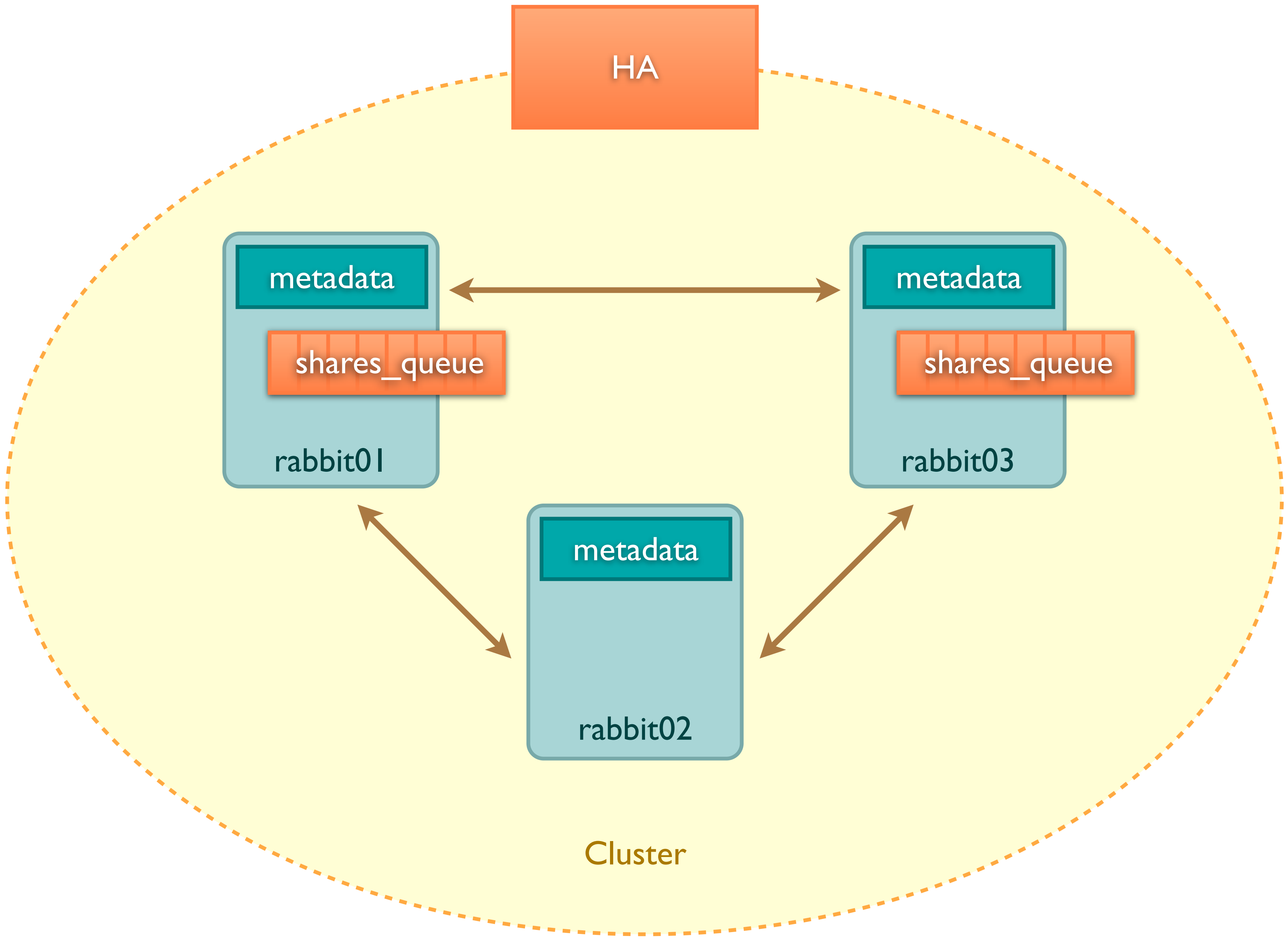
Naturally clustering doesn't really solve high availability problem as it is a separate issue. Let's look at HA next

HA with Mirrored Queues



Needs to be in a clustered environment before HA can be achieved

HA with Mirrored Queues

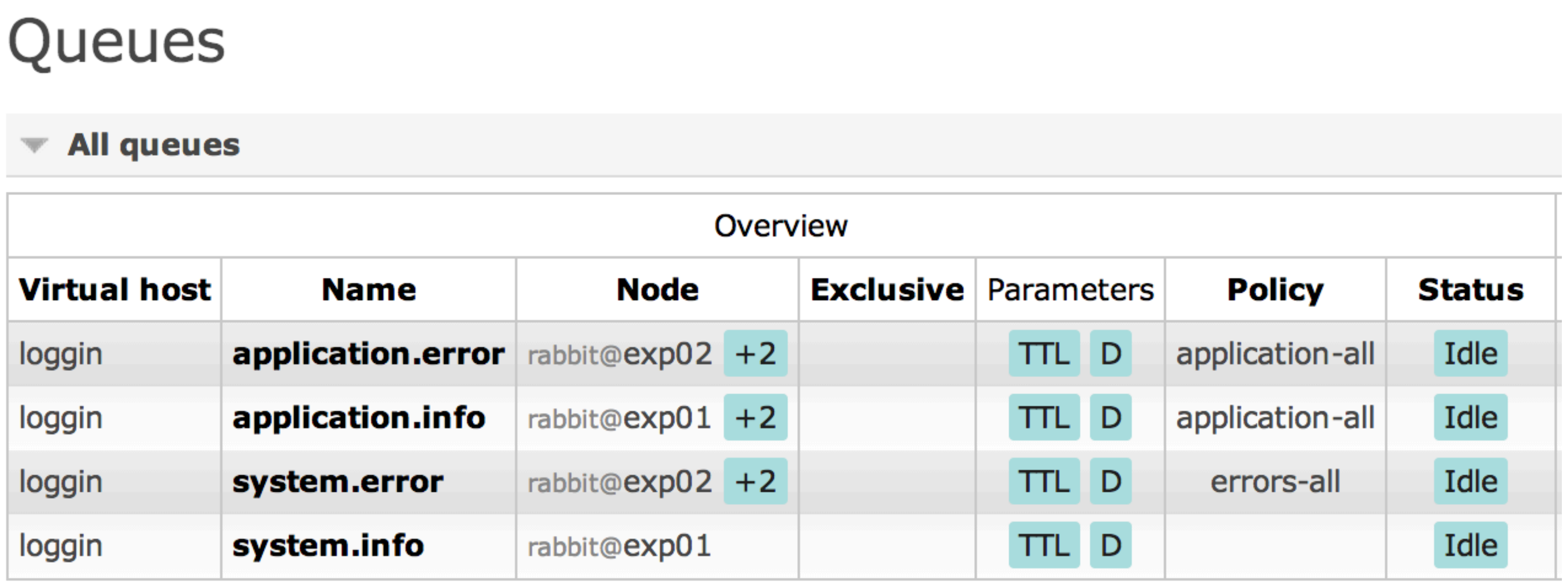


Needs to be in a clustered environment before HA can be achieved

HA with Mirrored Queues

- Set up as a queue policy in RabbitMQ
  - Create a new policy
  - Attach a set of *policy keys* to it to trigger mirroring
- Policy key:
  `ha-mode`
- Policy values:
  `all, exactly(N), nodes(str...)`

HA with Mirrored Queues

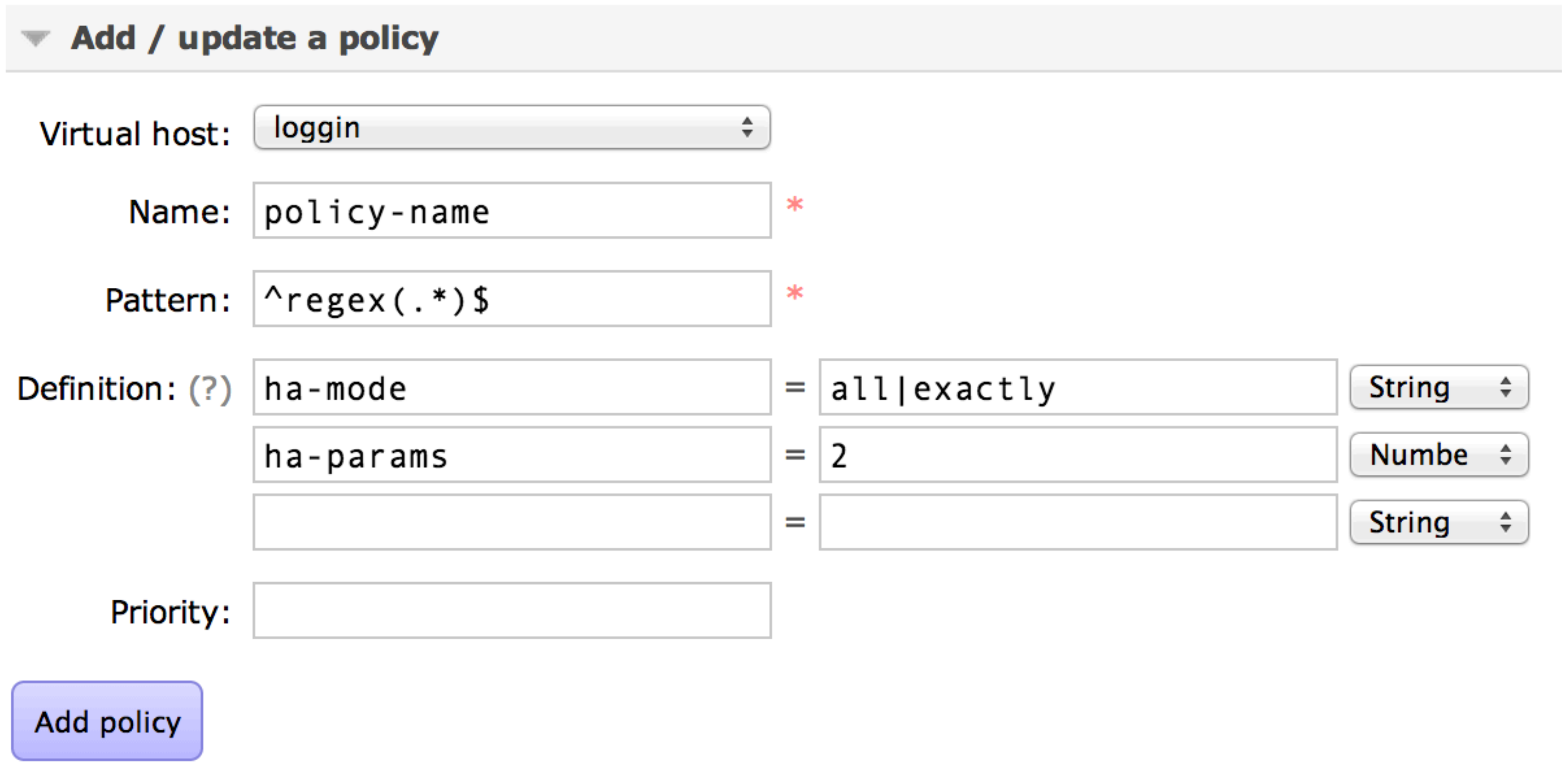

Queues

All queues

Overview						
Virtual host	Name	Node	Exclusive	Parameters	Policy	Status
loggin	**application.error**	rabbit@exp02 +2		TTL D	application-all	Idle
loggin	**application.info**	rabbit@exp01 +2		TTL D	application-all	Idle
loggin	**system.error**	rabbit@exp02 +2		TTL D	errors-all	Idle
loggin	**system.info**	rabbit@exp01		TTL D		Idle

Here is an example of what queues look like when it is replicated/mirrored across multiple nodes. “+2” means it’s on two other nodes.

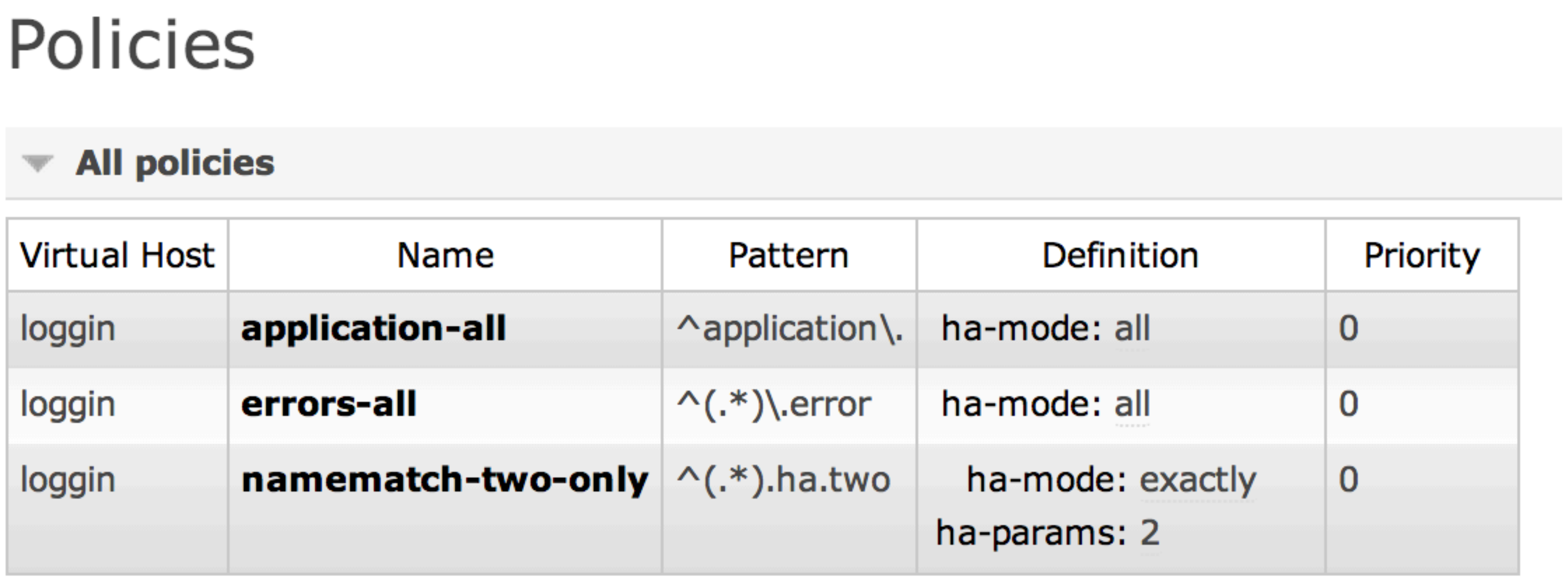
HA with Mirrored Queues


Add / update a policy

Virtual host:	loggin			
Name:	policy-name *			
Pattern:	^regex(.*)$ *			
Definition: (?)	ha-mode	=	all|exactly	String
	ha-params	=	2	Numbe
		=		String
Priority:				

The most powerful part of this approach is that the policy is applied to queues using a regex pattern. This makes it extremely powerful.

HA with Mirrored Queues



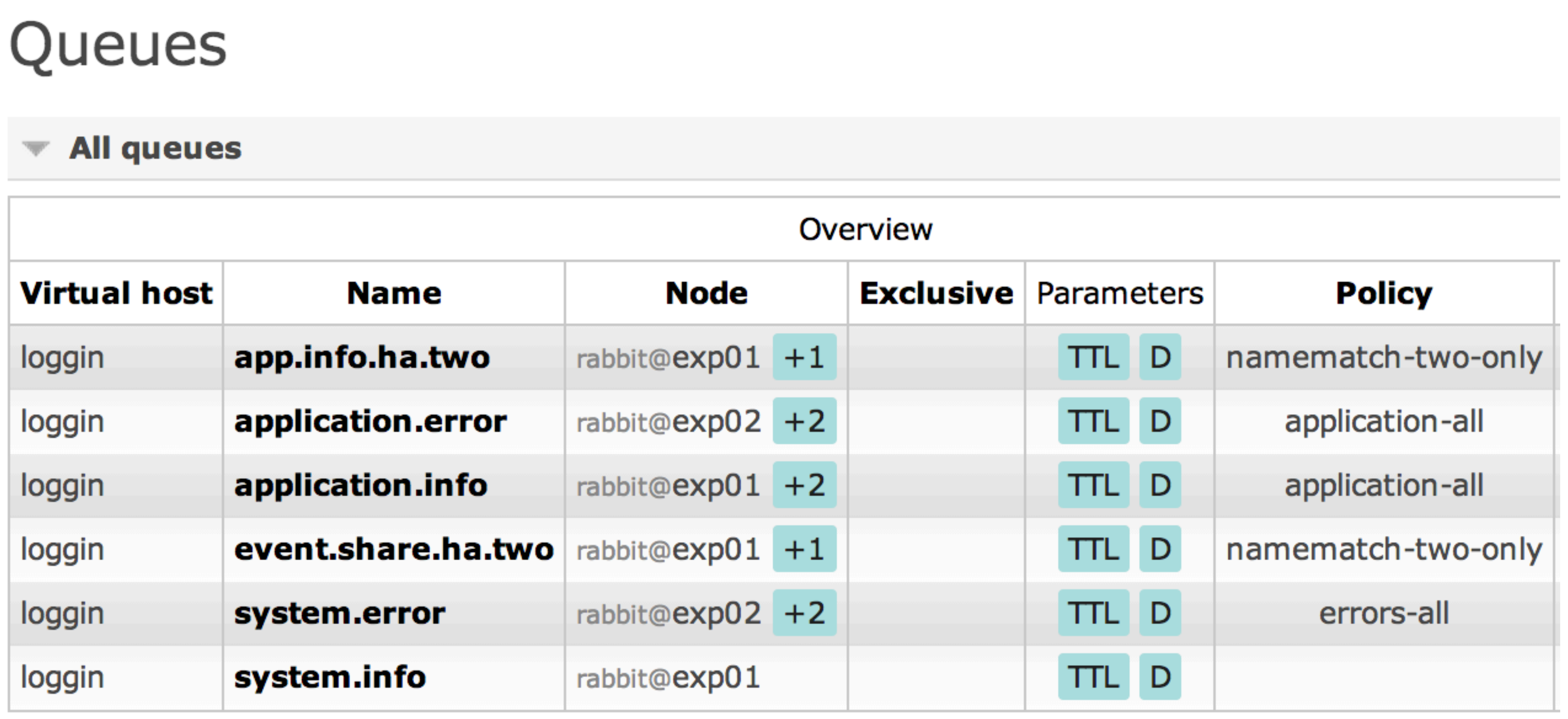
Policies

▼ All policies

Virtual Host	Name	Pattern	Definition	Priority
loggin	**application-all**	^application\.	ha-mode: all	0
loggin	**errors-all**	^(.*)\.error	ha-mode: all	0
loggin	**namematch-two-only**	^(.*).ha.two	ha-mode: exactly ha-params: 2	0

How it would typically look like. The "definition" is the name you gave the policy
Most powerful part is you can create something similar to namematch-two-only. Basically this means any future queues that ends with *.ha.two will be automatically mirrored across 2 nodes, like the next slide

HA with Mirrored Queues



Queues

All queues

Overview					
Virtual host	**Name**	**Node**	**Exclusive**	Parameters	**Policy**
loggin	**app.info.ha.two**	rabbit@exp01 +1		TTL D	namematch-two-only
loggin	**application.error**	rabbit@exp02 +2		TTL D	application-all
loggin	**application.info**	rabbit@exp01 +2		TTL D	application-all
loggin	**event.share.ha.two**	rabbit@exp01 +1		TTL D	namematch-two-only
loggin	**system.error**	rabbit@exp02 +2		TTL D	errors-all
loggin	**system.info**	rabbit@exp01		TTL D	

you can see here app.info.ha.two and event.share.ha.two is a “+1”

HA with Mirrored Queues

Walkthrough and demo

Distributed RabbitMQ

Clustering

- ➡ Acts as a single logical unit
- ➡ Consumer sees cluster as single node
- ➡ Nodes shares the same Erlang cookie
- ➡ Automatic metadata replication, etc
- ➡ Requires reliable LAN-like environment
- ➡ Emphasis on Consistency & Availability

Replication

- ➡ Acts like a forwarding agent
- ➡ Nodes doesn't share any cookie and is standalone
- ➡ Uses AMQP protocol itself for forwarding
- ➡ Suitable for unreliable WAN connections
- ➡ Emphasis on Availability & Partition Tolerance

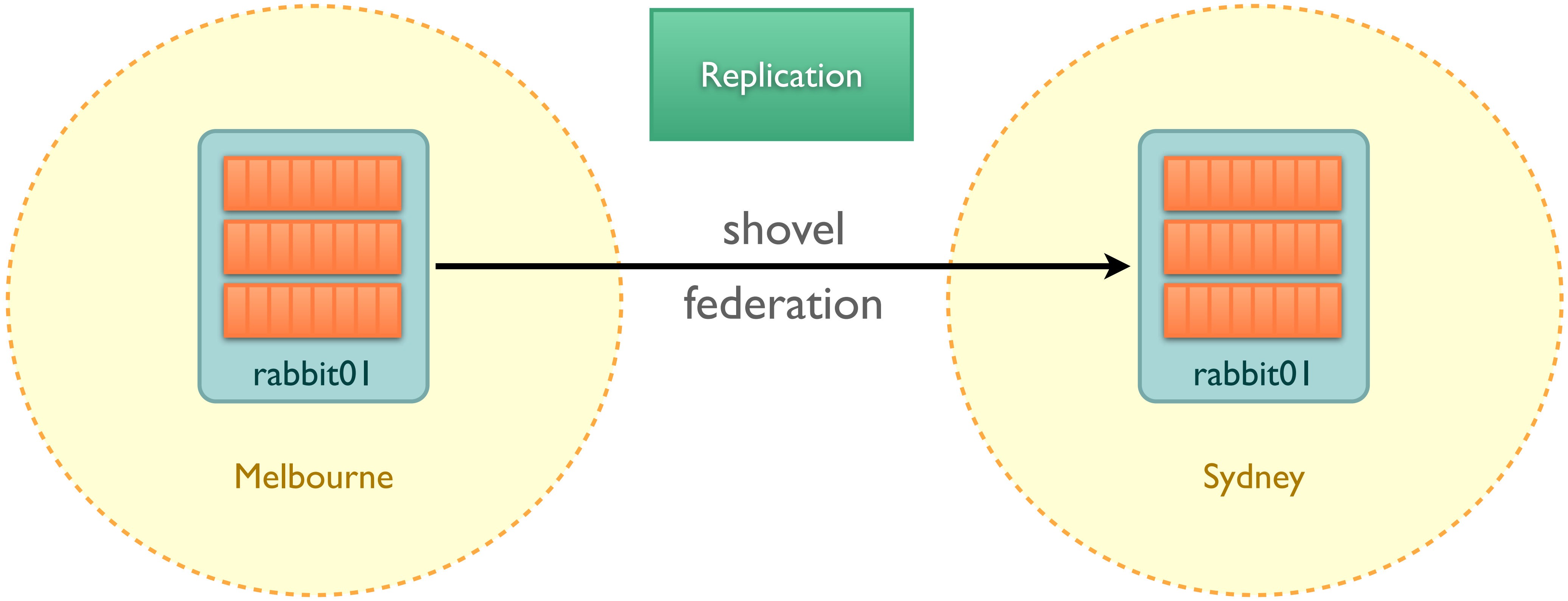
HA

- ➡ A combo of both
- ➡ It's cluster with replication
- ➡ Usually in form of master/slave
- ➡ Emphasis on Consistency & Availability

Distributed RabbitMQ

Replication

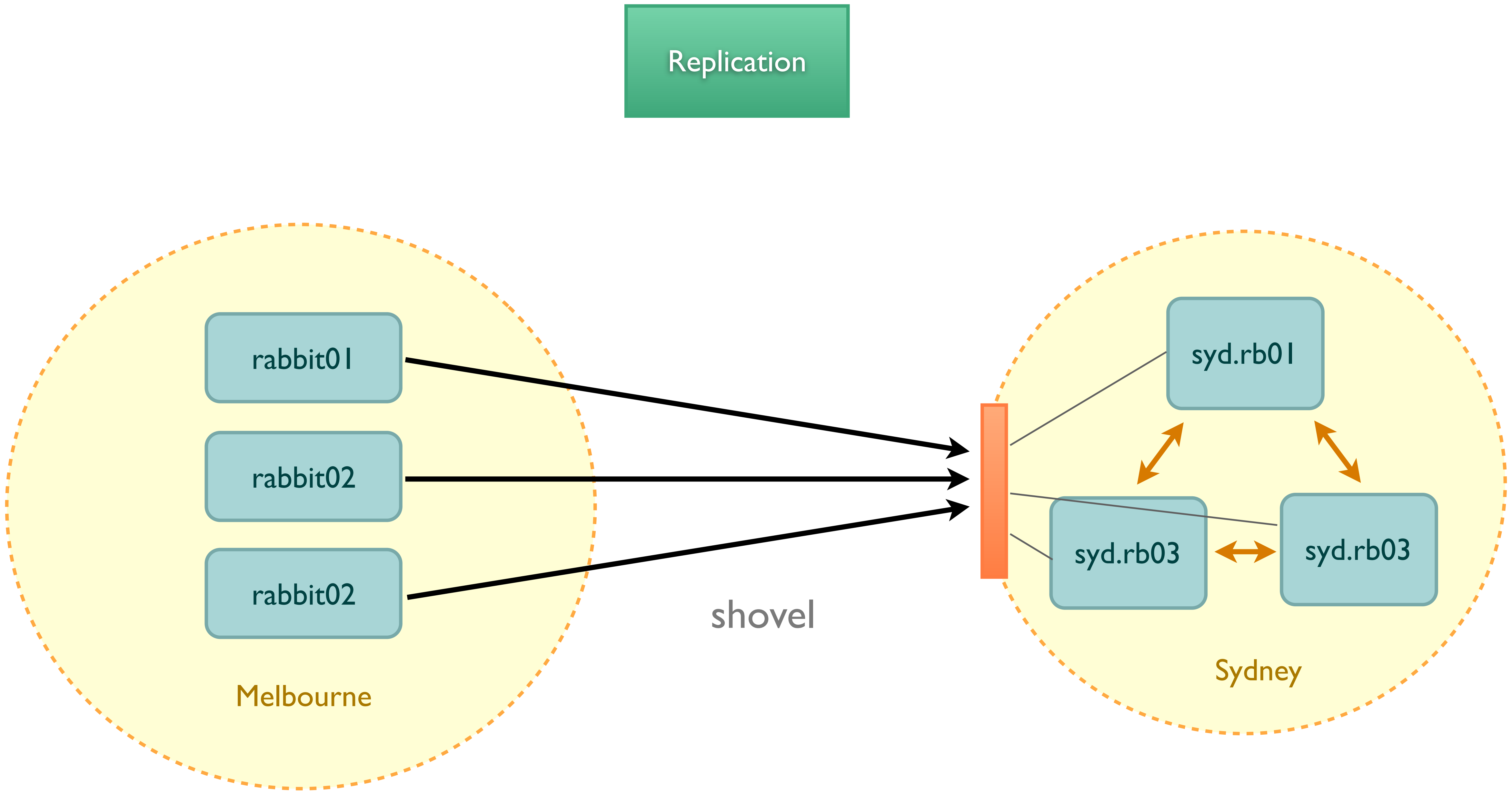
Distributed RabbitMQ



- consumes and republishes
- exchange to exchange
- works well across WAN
- two main difference:
  - shovel is lower level
  - shovel is more flexible

Two main plugins - shovel and federation. Both are very very similar but it seems there's more push for Shovel. Both does the same things listed here...

Distributed RabbitMQ



Idea is to have local rabbitmq on web nodes re-publishing messages across WAN to a cluster.

Replication with Shovel

Walkthrough and demo

Q&A