Messaging Standards and Systems

AMQP & RabbitMQ

Gavin M. Roy
VP of Architecture
AWeber Communications
Twitter: @Crad

All Things Open
October 22, 2014
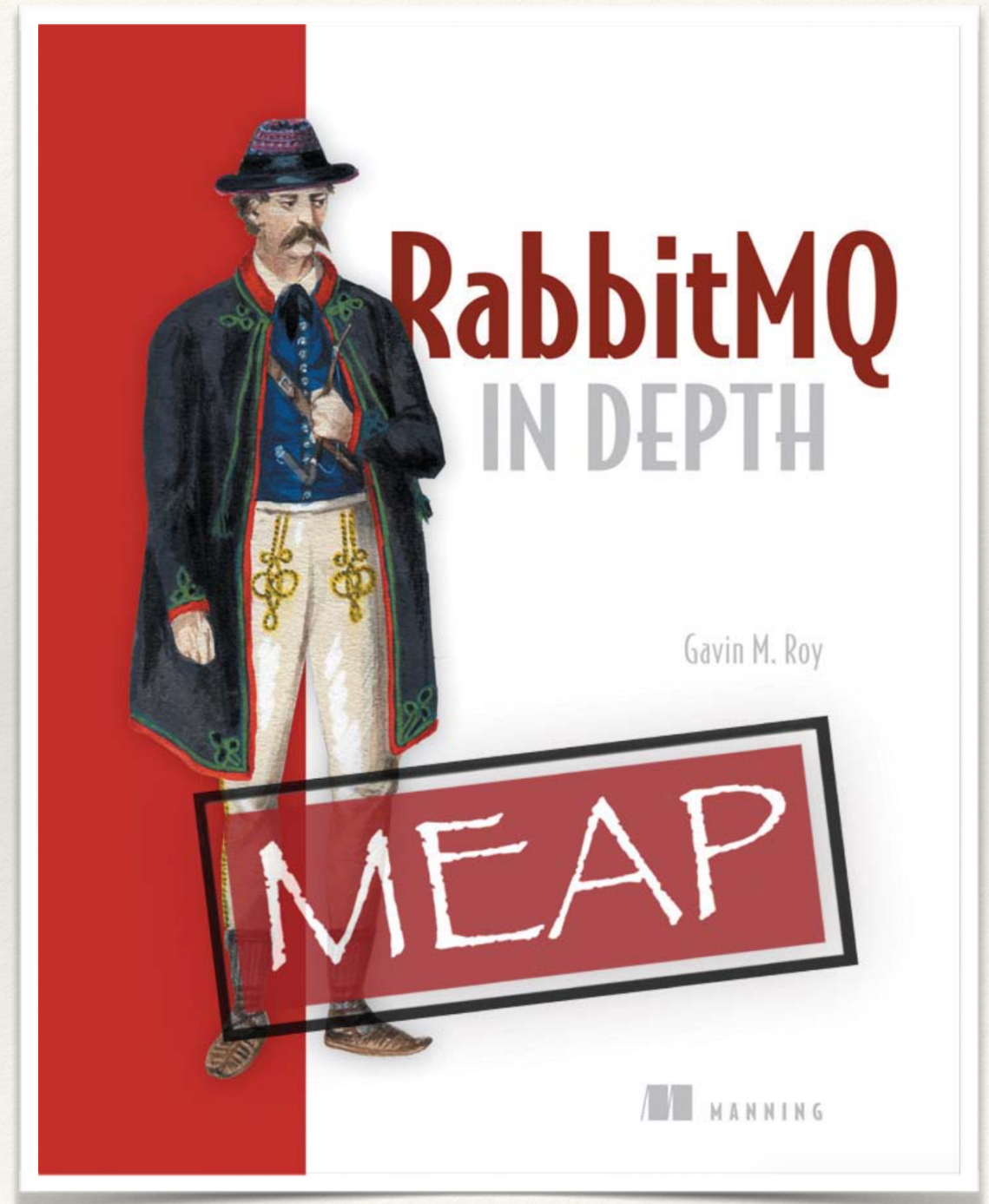
About Me

VP of Architecture
AWeber Communications

Blame me for pika, rabbitpy, pamqp, and a handful of RabbitMQ plugins

Blog: https://gavinroy.com

Github: https://github.com/gmr

Book: http://manning.com/roy
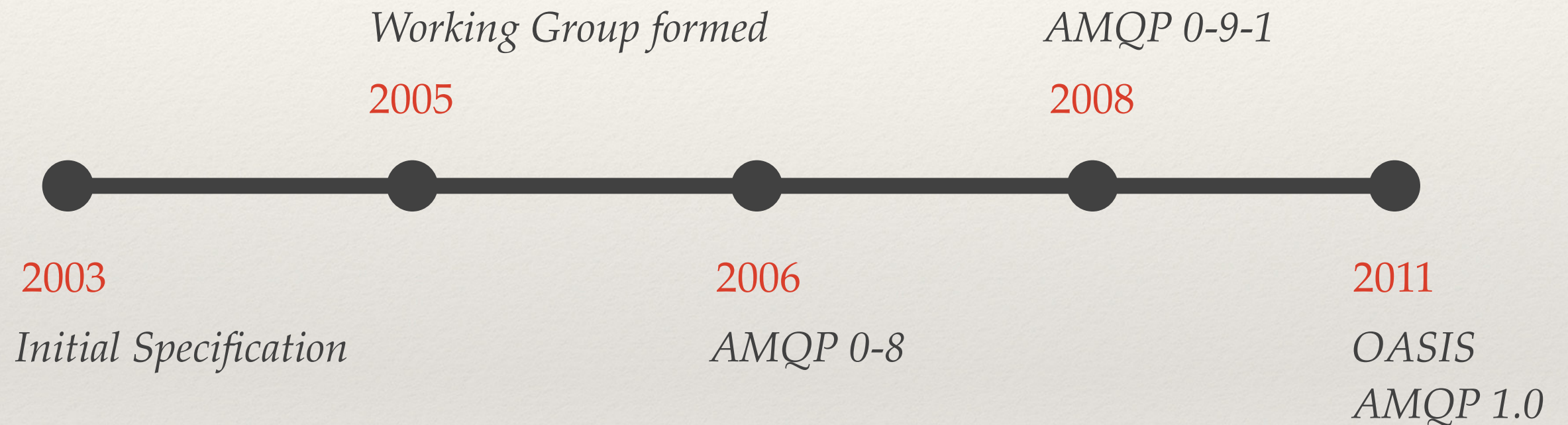

RabbitMQ IN DEPTH

Gavin M. Roy

MEAP

MANNING

Advanced Message Queueing Protocol

- ❖ Open Standard

- ❖ Platform and Vendor Neutral

- ❖ Multiple Versions

AMQP Timeline

Working Group formed

2005

AMQP 0-9-1

2008

2003

2006

2011

Initial Specification

AMQP 0-8

OASIS AMQP 1.0

AMQP Working Group

- JPMorgan Chase
- Cisco Systems
- IONA Technologies
- iMatrix
- RedHat
- TWIST
- Bank of America
- Barclays
- Credit Suisse
- Deutsche Börse Systems
- Goldman Sachs
- HCL Technologies
- Progress Software
- IIT Software
- INETCO Systems Ltd.
- Informatica Corporation
- Microsoft Corporation
- my-Channels
- Novell
- Solace Systems
- Tervala, Inc.
- VMWare
- WSO_2

Competing AMQP Standards

0-8
&
0-9-1
≠
1.0
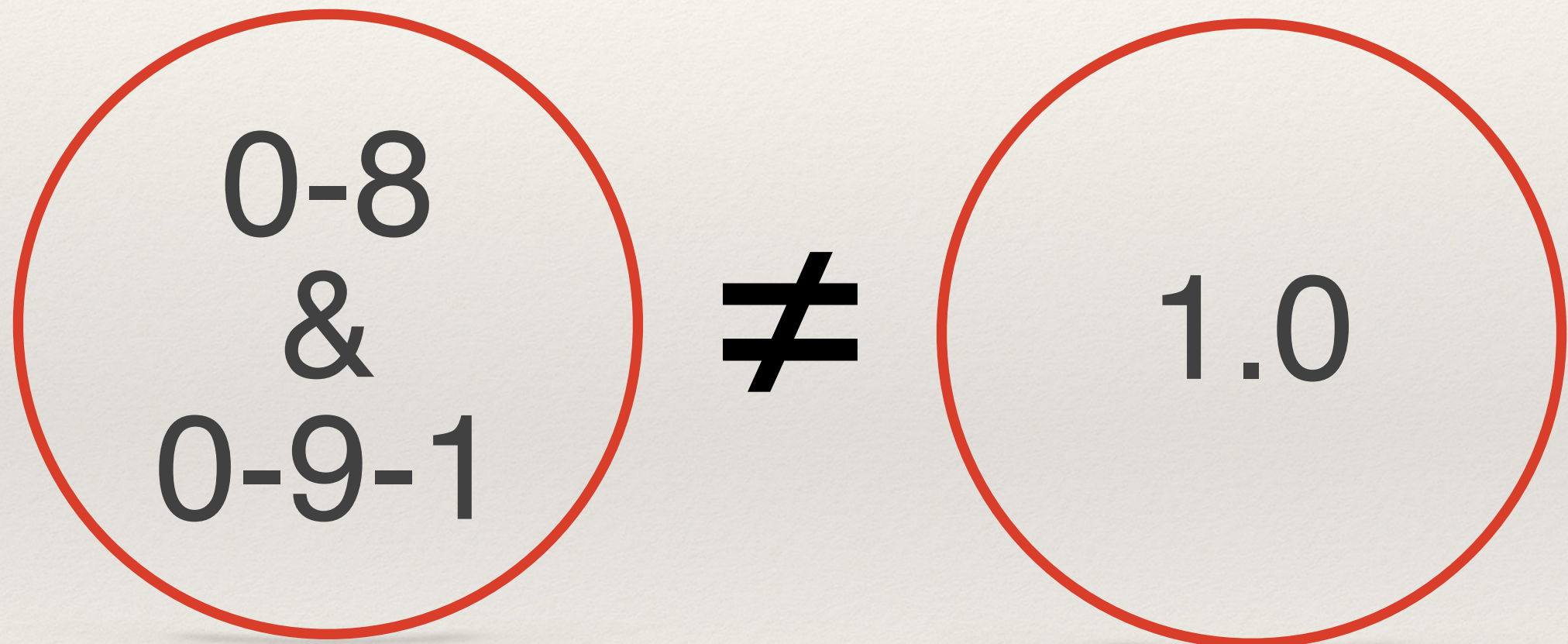
AMQP 0-9-1

- ❖ Currently has wider support than AMQP 1.0

- ❖ Multiple Broker Implementations:
  RabbitMQ, Apache Qpid, and SwiftMQ to name a few

- ❖ Specifies a Model and Protocol

Native AMQP 0-8/0-9-1 Clients

| C	Go	OCaml
Clojure	Groovy	Perl
Cobol	Haskell	PHP
Common Lisp	Java	Python
Delphi	JavaScript	Ruby
Erlang	.NET	Scala

Advanced Message Queueing Model

❖ **Exchange**
Receives and route messages

❖ **Message Queue**
Stores messages until they can be consumed

❖ **Binding**
Defines the relationship between an Exchange and Queue and provides routing criteria

Exchange

Binding

Message Queue

X
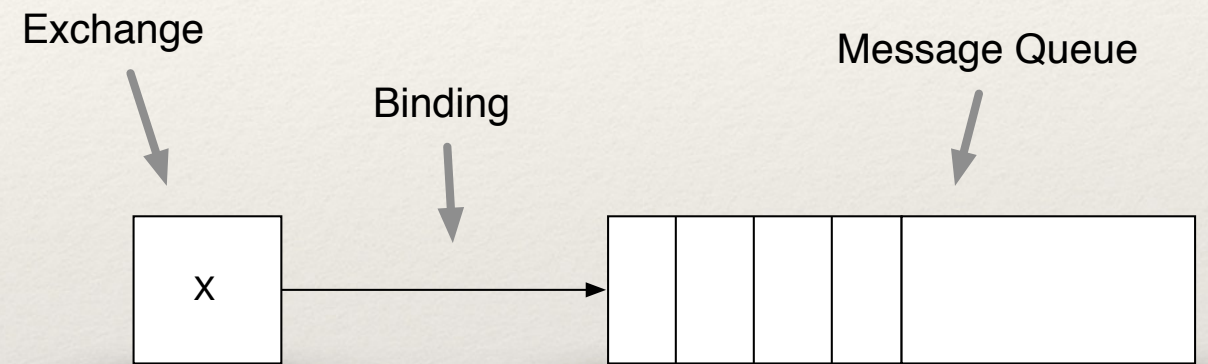
Routing Keys

❖ Provided when publishing a message

❖ Compared against binding keys by exchanges

❖ Ideally provide context to the message:

    ❖ *Connote the type of the message*

    ❖ *Categorize the content in the message*

    ❖ *Specify the type of consumer that should receive it*

Advanced Message Queueing Protocol

- ❖ Compact, binary frame format wire protocol

- ❖ Bi-directional RPC

- ❖ Commands consist of Classes and Methods:

  - ❖ *Example Request:*
    `Queue.Declare(name="foo")`

  - ❖ *Response:*
    `Queue.DeclareOk(messages=0, consumers=0)`

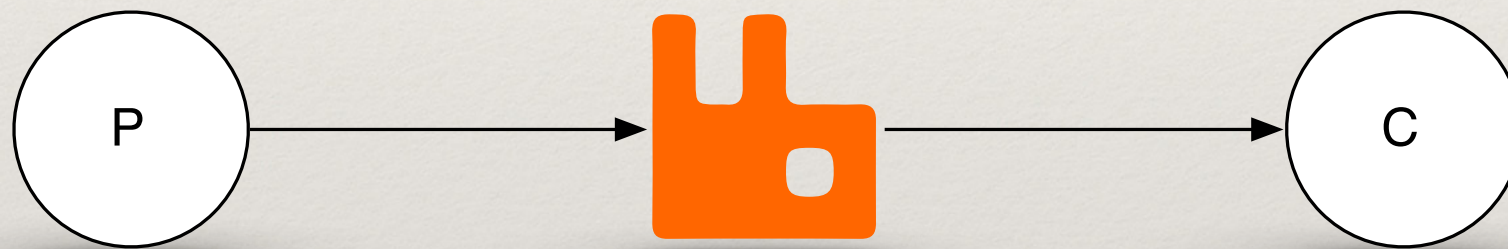# Common AMQP Terms

❖ Broker
A server that implements AMQP

❖ Producer or Publisher
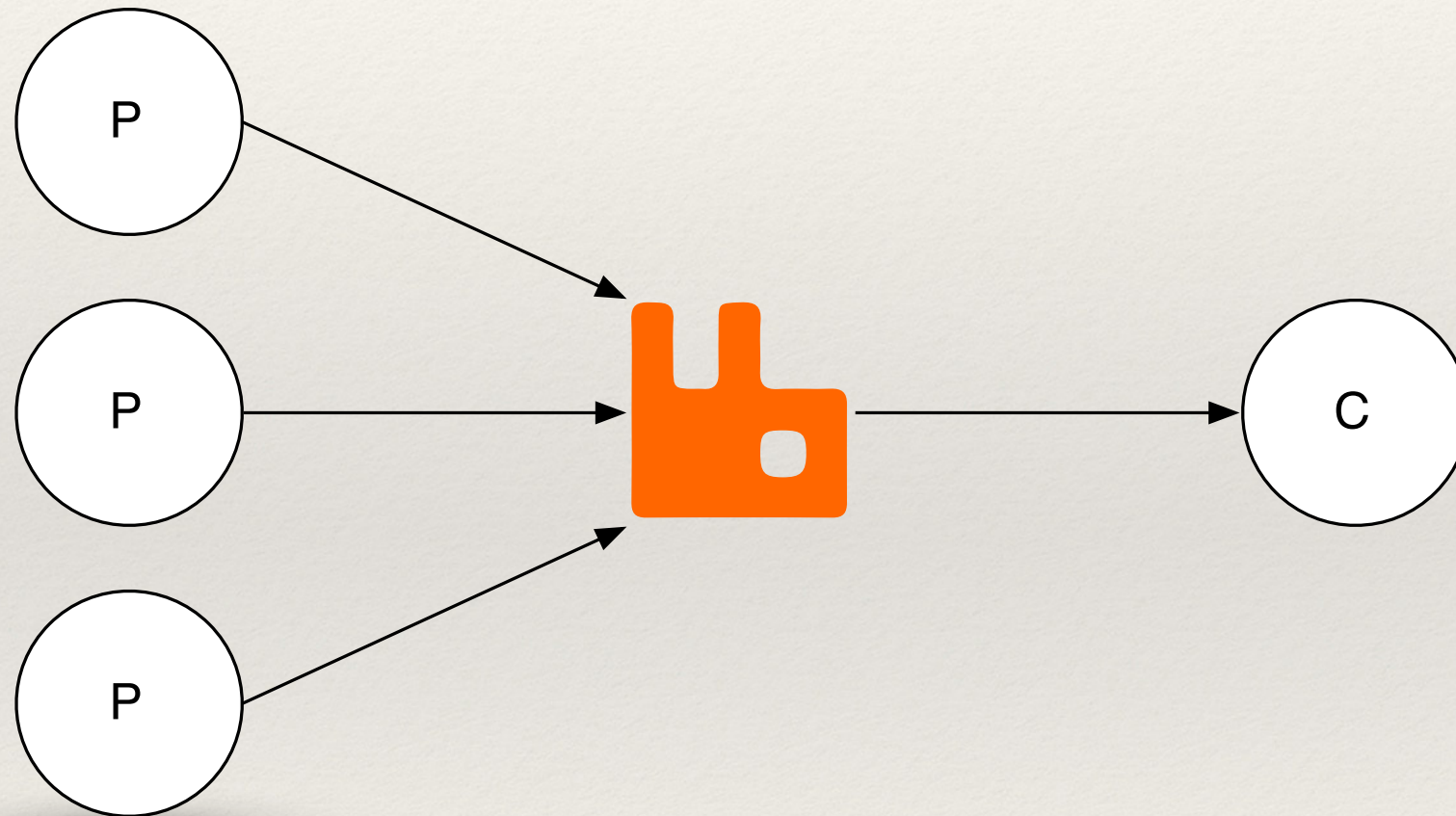A client application that sends messages to a broker

❖ Consumer
A client application that reads messages from a queue

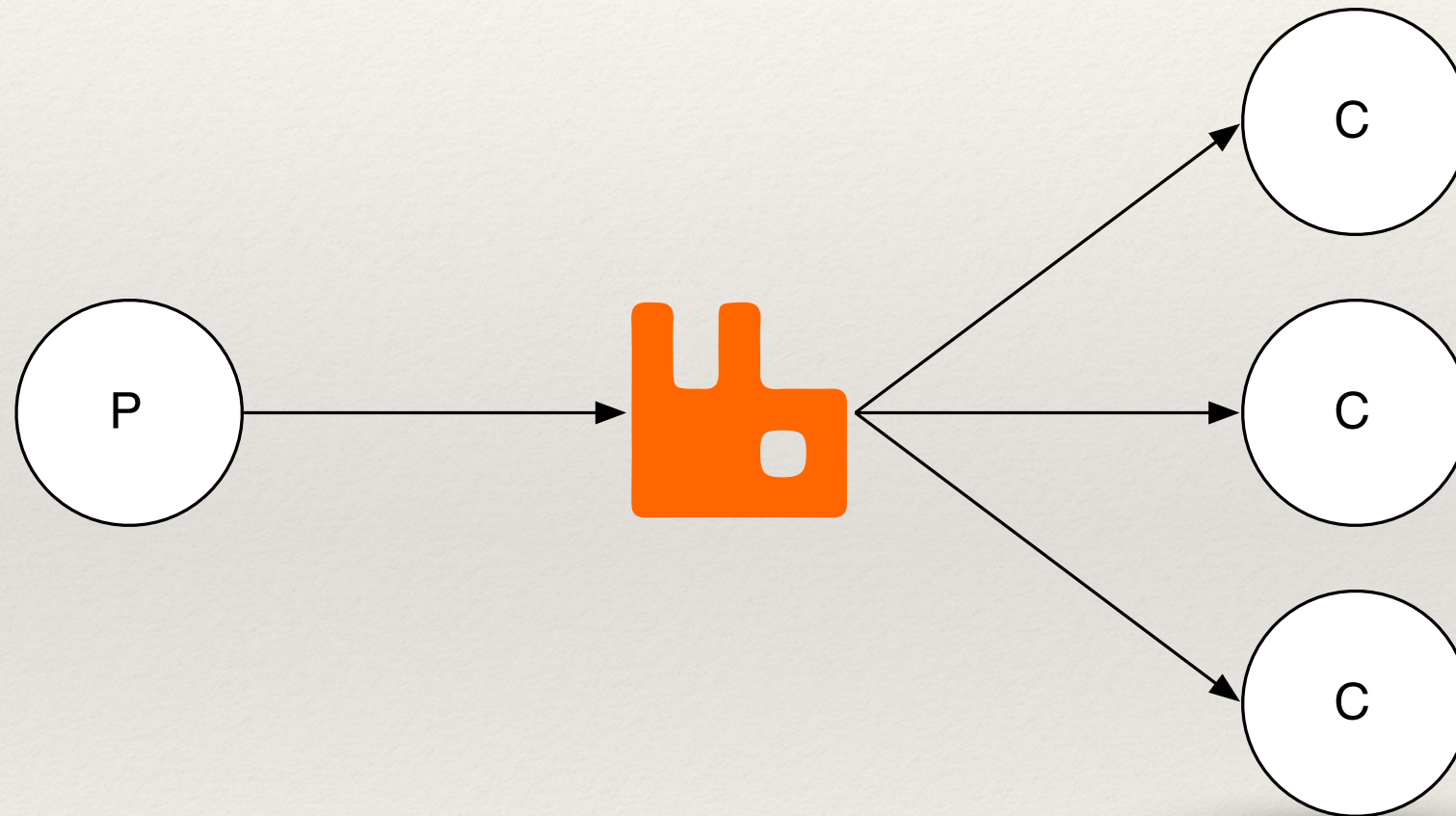# Publishers and Consumers

Multiple Publishers

Multiple Consumers

AMQP Messages

- Comprised of 3 or more frames:

  - **Method Frame**
    Basic.Publish, Basic.Deliver, etc

  - **Content Header** with body size & message properties
    timestamp, message-id, app-id, etc

  - *n* **Body** Frames with the opaque message payload

AMQP 0-9-1 Issues & Gotchas

- Ambiguous

- Authentication

- Asynchronous

- Connection Negotiation

- Exceptions

RabbitMQ

- ❖ Open Source (MPL)

- ❖ Written in Erlang/OTP

- ❖ Developed/Maintained by Pivotal

- ❖ Multi-Protocol
  AMQP 0-9-1 & 1.0, MQTT, STOMP, XMPP, HTTP, Web-STOMP & More

- ❖ Roots in AMQP 0-8/0-9-1

Who Uses It?

Agora Games

Chef

Google AdMob

Instagram

MeetMe

Mercado Libre

Mozilla

NASA

New York Times

National Science Foundation

Openstack

Rapportive

Reddit

Soundcloud

(and many more)

Why Use RabbitMQ?

❖ Create loosely-coupled applications

❖ Communicate across applications or platforms

❖ Tap into pre-existing message flows for new purposes

❖ Scale-out clustering for growth, throughput, and HA

❖ Federation for WAN latencies and network partitions

❖ Extensible plugin-in architecture

RabbitMQ Extensions to AMQP

Authentication Failure

Exchange to Exchange Bindings

Delivery Confirmations

Basic.Nack

Consumer cancellations

Consumer priorities

Dead Letter Exchanges

Alternate Exchanges

Connection blocking

Message CC & BCC Routing

Queue Length Limits

Per Queue Message TTL

Per Message TTL

Queue TTL

Message User ID Validation

Auto-delete exchanges

RabbitMQ Clustering

❖ LAN Only

❖ Adds highly-available queues

❖ Is cohesive, publish and consume from any node

❖ Leverages native Erlang clustering and communication

❖ Has multiple strategies for dealing with network partitions

❖ Manually configured via configuration or command line*
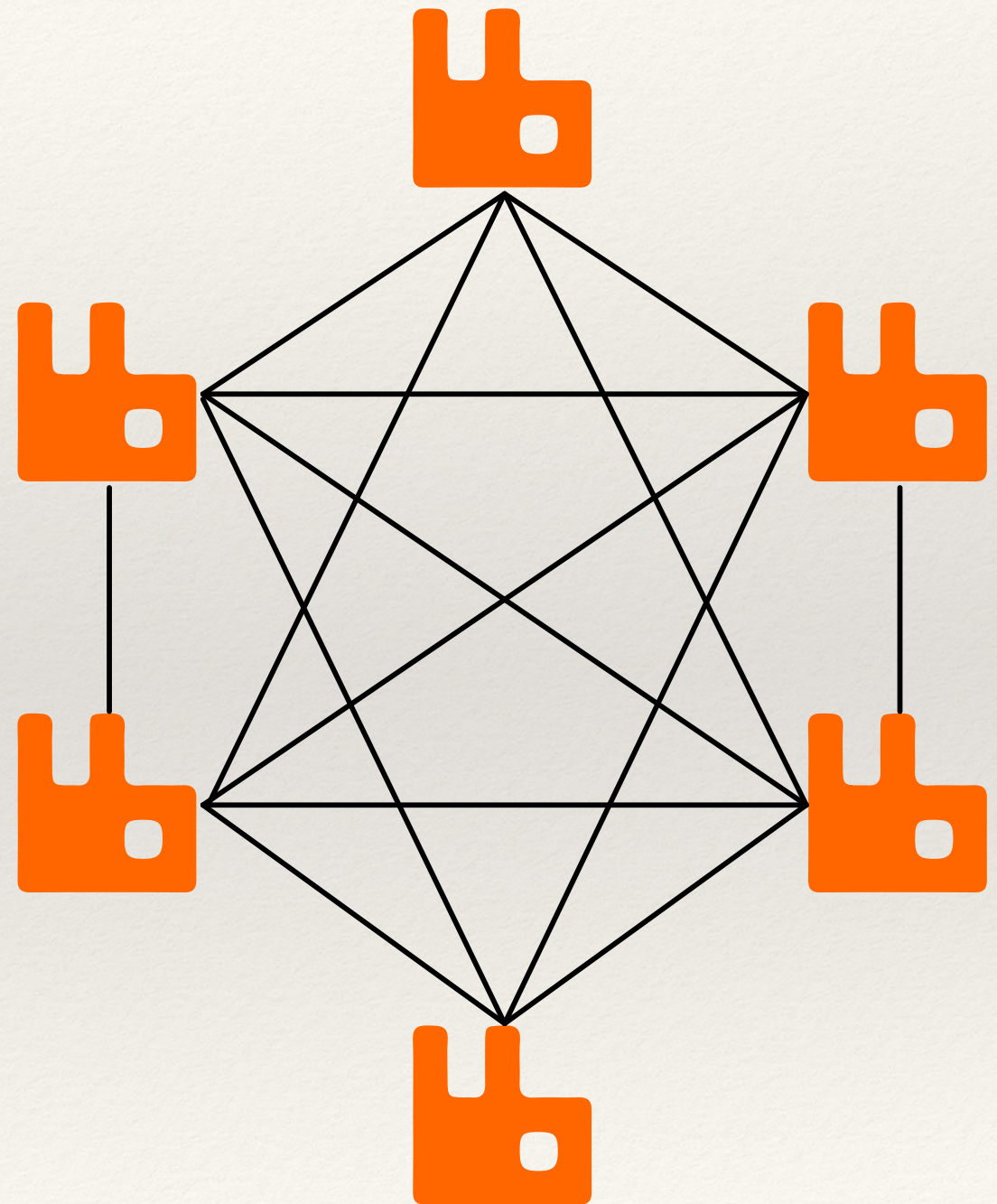
RabbitMQ Plugins

auth-backend-amqp
Use AMQP to respond to RabbitMQ auth requests

auth-backend-http
Use a custom web service to provide authentication

autocluster-consul
Automatically create clusters using Consul

influxdb-storage-exchange
Store JSON messages as events in InfluxDB

presence-exchange
Publishes messages upon binding changes

rabbitmq-toke
Tokyo Cabinet backing store for queue messages

rabbitmq-top
Top like view of RabbitMQ processes in the management UI

sharding
Scale out RabbitMQ with automatic queue sharding

(and many more)

Message Routing

Built-In Exchange Types

❖ **Direct**

String matching on the routing key

❖ **Fanout**

No routing key, messages delivered to all bound queues

❖ **Topic**

Pattern matching in the routing key

❖ **Headers**

No routing key, value matching in the headers property

Topic Exchange Binding Keys

Routing Key: namespace.delimited.keys

#
Receive all messages

namespace.#
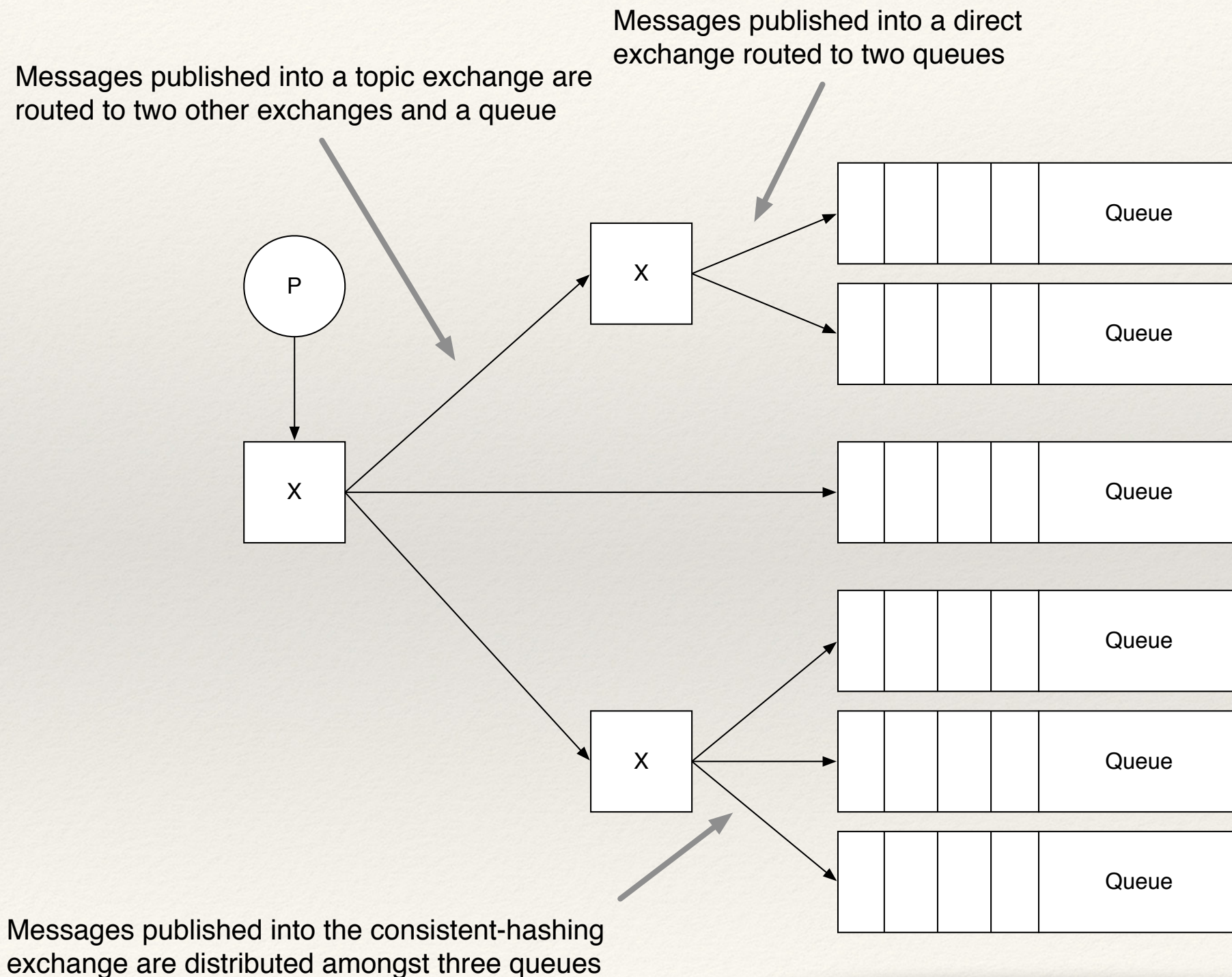*Receive all messages in namespace*

namespace.delimited.*
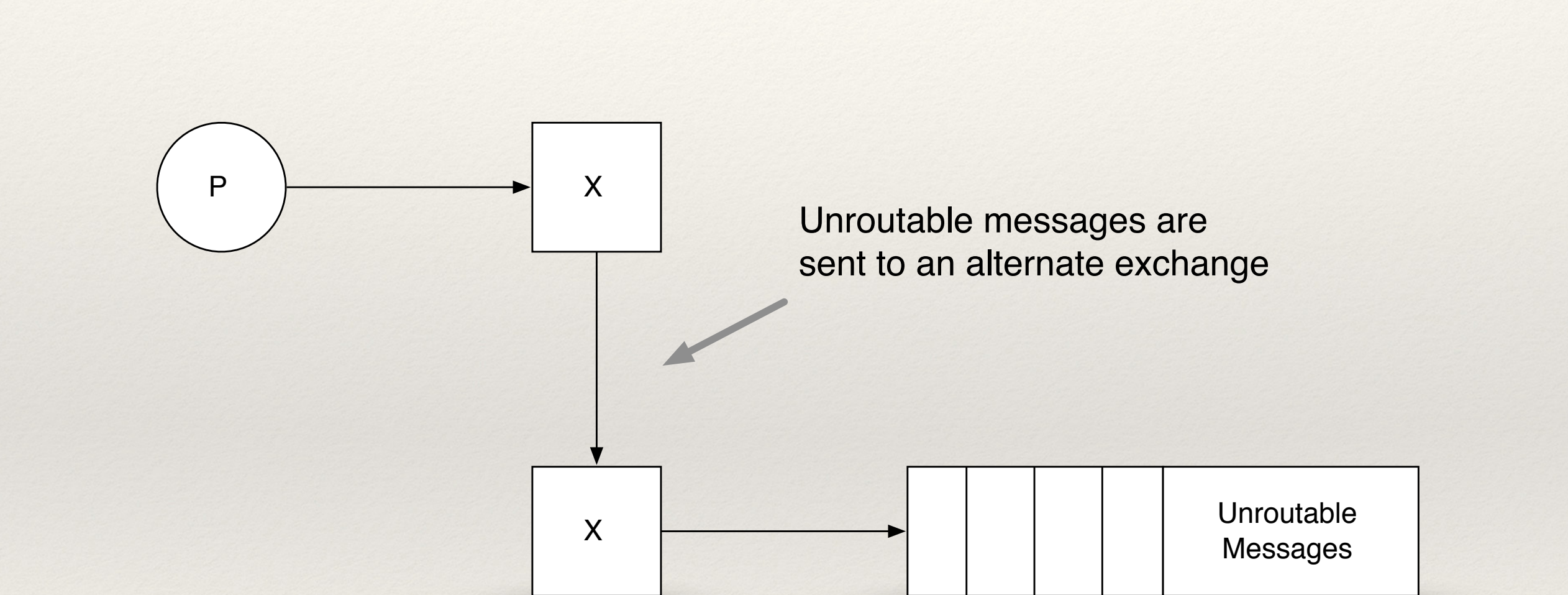*Receive all namespace.delimited messages*

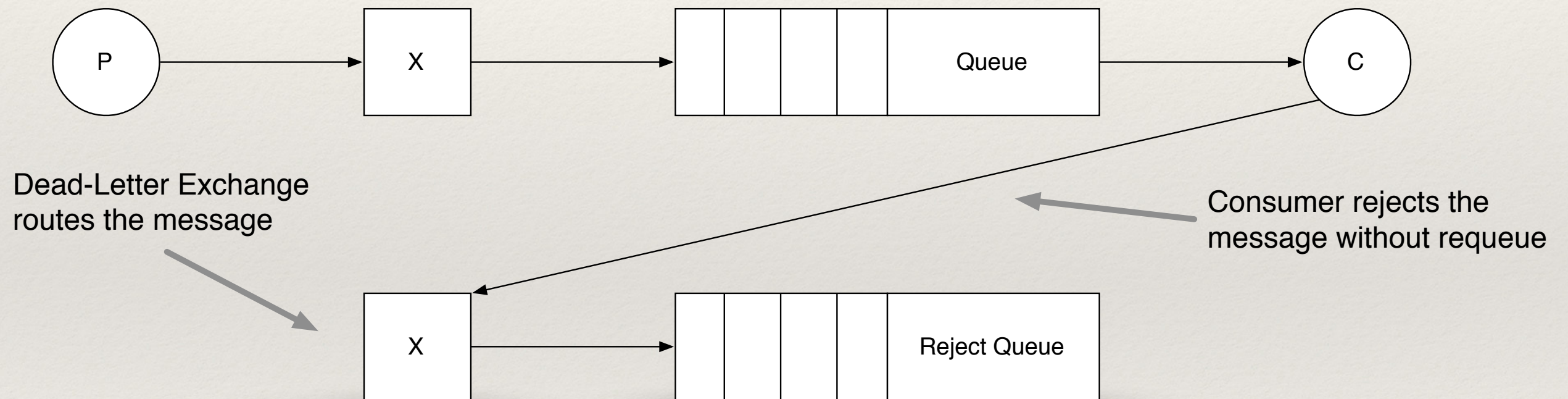**namespace.*.keys**
Receive all namespace messages ending with keys

Exchange to Exchange Binding

Messages published into a topic exchange are routed to two other exchanges and a queue

Messages published into a direct exchange routed to two queues

P

X

X

X

Queue

Queue

Queue

Queue

Queue

Queue

Messages published into the consistent-hashing exchange are distributed amongst three queues

Alternate Exchanges

P

X

Unroutable messages are
sent to an alternate exchange

X

Unroutable
Messages

Dead Letter Exchanges

P → X → Queue → C

Dead-Letter Exchange routes the message

Consumer rejects the message without requeue

X → Reject Queue

Exchange Plugins

Consistent Hashing
Distribute messages via hashed value of routing key

Event
Publishes messages on AMQP events such as queue creation

Random
Distribute messages across all bound queues randomly

PostgreSQL LISTEN
Subscribes to and publishes PostgreSQL notifications

Recent History
Sends the last n messages to any newly bound queue

Reverse Topic
Allows for routing patterns at publish time and not via binding
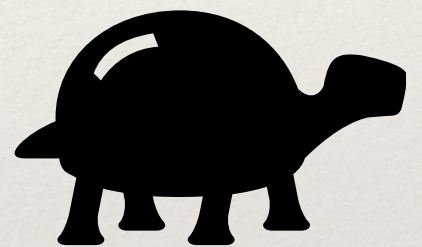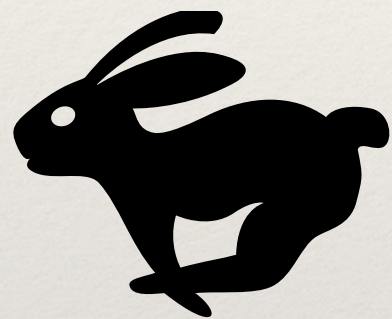
Riak Storage
Stores messages published through the exchange into Riak

Script Exchange
Calls out to external scripts for message routing

Performance Considerations

Publishing Performance Scale

No guarantees

Notification on failure

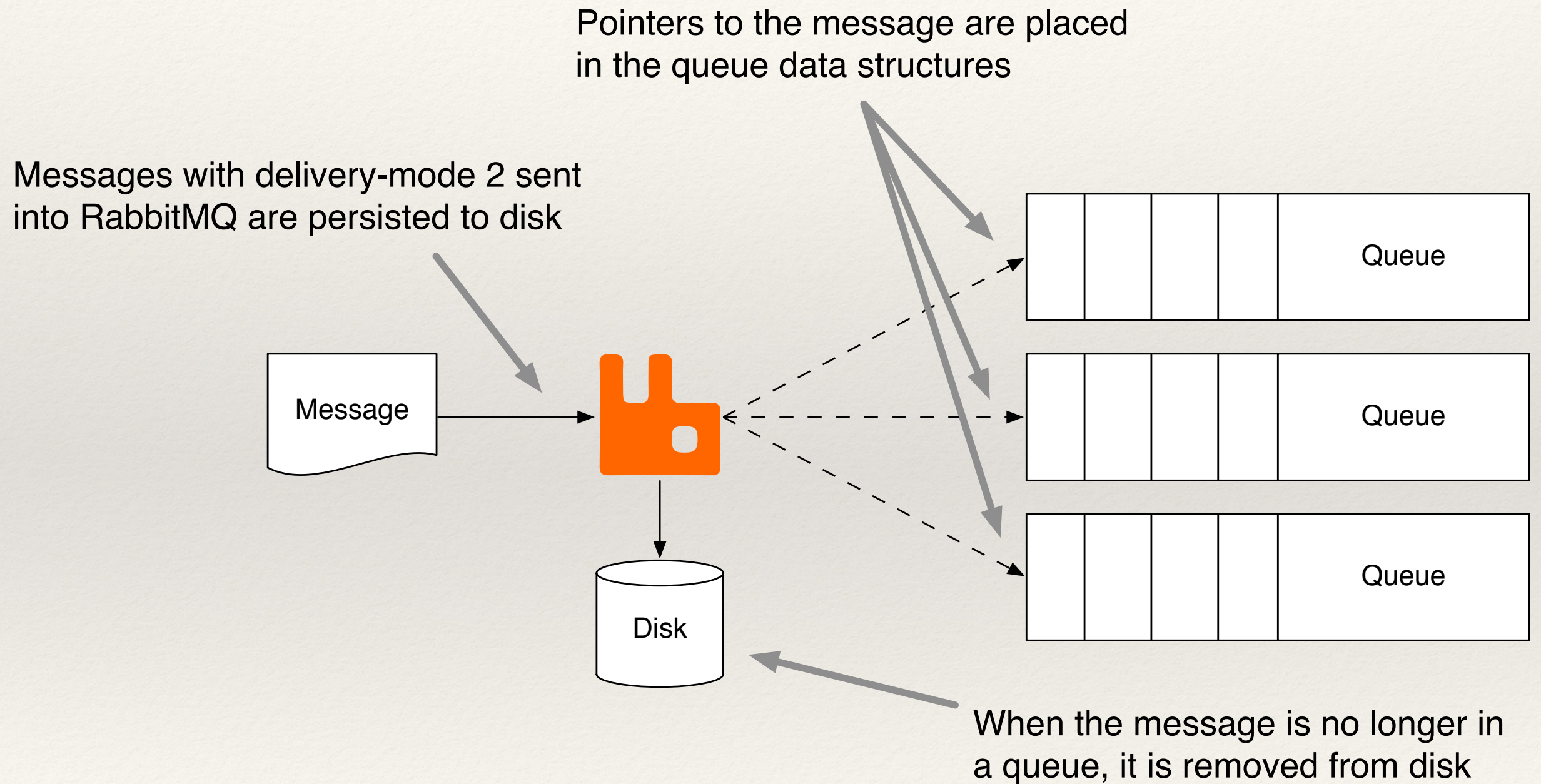Publisher confirms
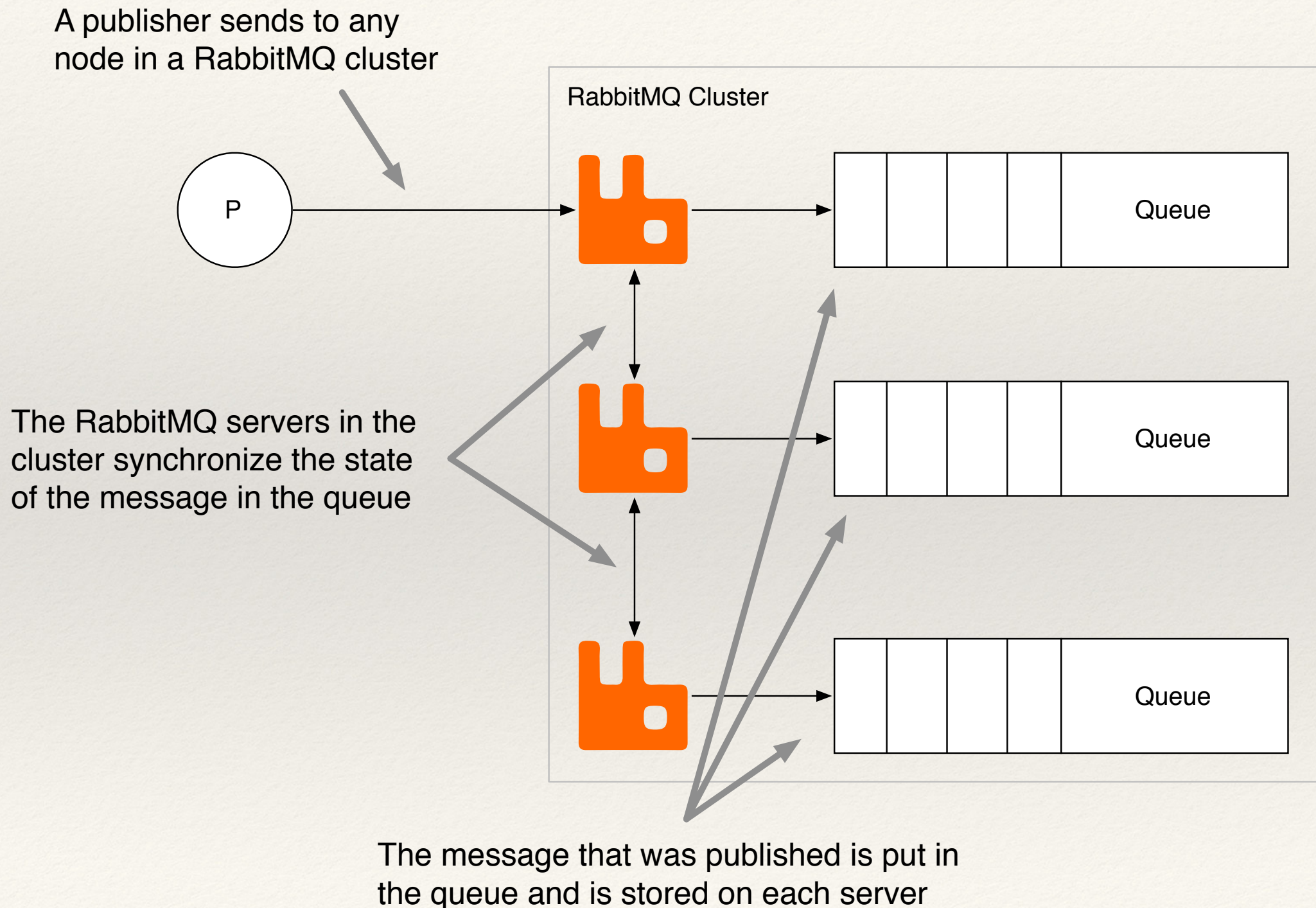
Alternate exchanges

HA Queues

Transactions

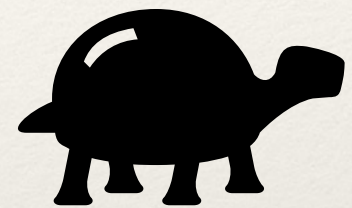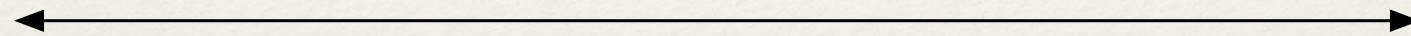HA Queues w/ Transactions

Persisted Messages

Persisted Messages IO Model

Pointers to the message are placed
in the queue data structures

Messages with delivery-mode 2 sent
into RabbitMQ are persisted to disk

Message

Queue

Queue

Queue

Disk

When the message is no longer in
a queue, it is removed from disk

HA Queues & Performance

A publisher sends to any
node in a RabbitMQ cluster

RabbitMQ Cluster

P

Queue

The RabbitMQ servers in the
cluster synchronize the state
of the message in the queue

Queue

Queue

The message that was published is put in
the queue and is stored on each server

Consumer Performance Scale

Consuming with Acknowledgements and QoS > 1

Consuming with "No Ack Mode" enabled

Consume with Acknowledgements

Consuming and using Transactions

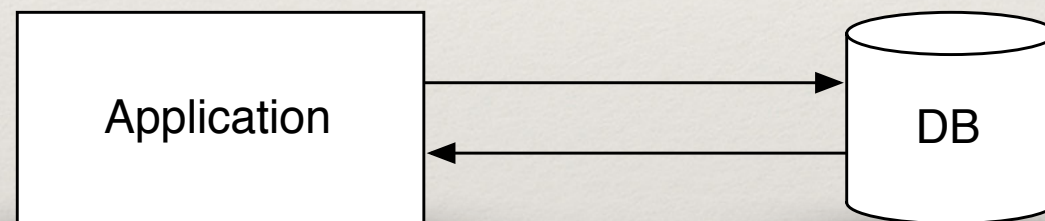Getting Messages

Operational Concerns

Trending & Monitoring

❖ RabbitMQ Management Plugin provides internal stats

  ❖ *Queue depths, connection counts, throughput, memory usage, etc*

❖ Monitor with common tools such as Nagios or Sensu to services such as Boundary and NewRelic

❖ Stream based monitoring with Riemann for anomaly detection

Thoughts on Configuration

- Use configuration management!

  - Even for exchanges, queues, and bindings if possible

  - Helpful for disaster recovery

- Use RabbitMQ's Policies when possible

  - Exchanges and queues are immutable

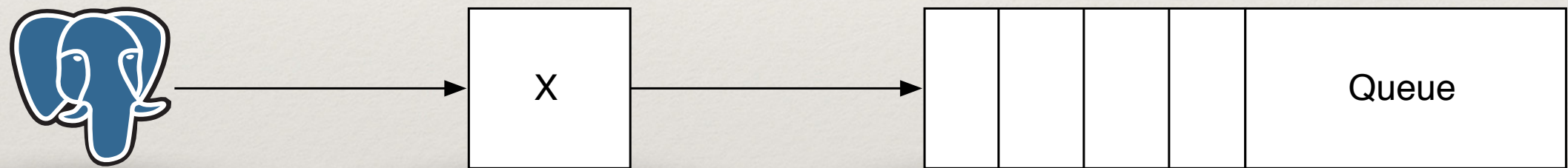 - Deleting and redeclaring for changes can be disruptive

Sample Usage Patterns

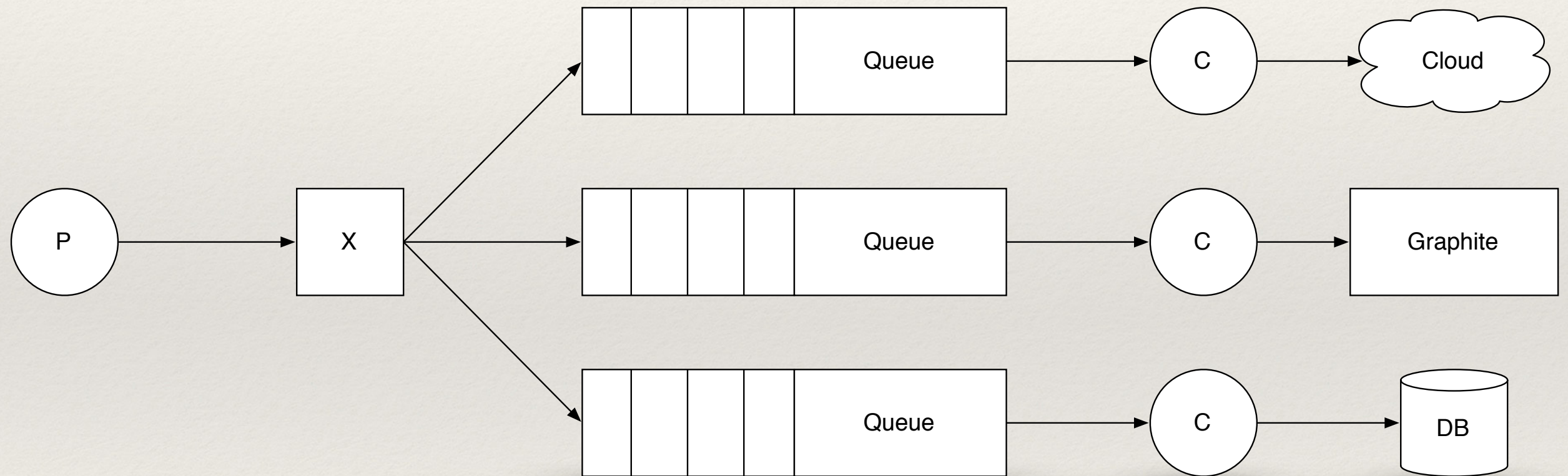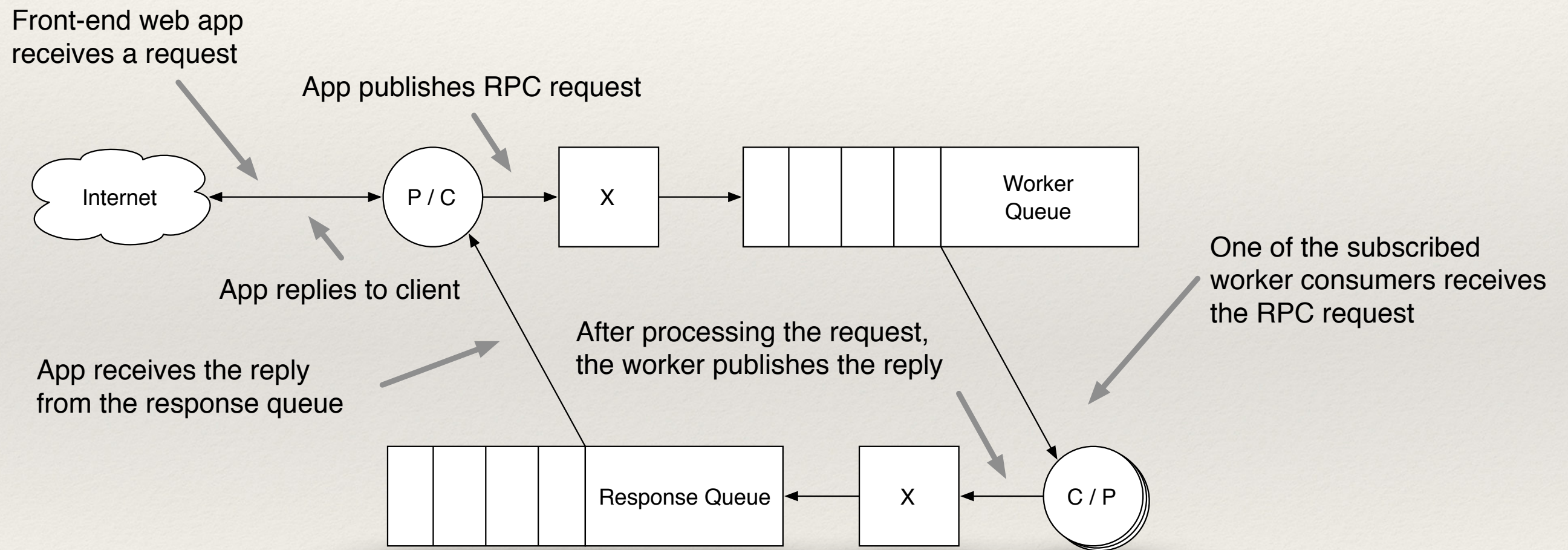# Evolve Tightly-Coupled Applications

Decoupling Database Writes

Listen for Database Notifications

Multi-Purposed Messages

RPC

Front-end web app receives a request

App publishes RPC request

Internet

P / C

X

Worker Queue

App replies to client

One of the subscribed worker consumers receives the RPC request

App receives the reply from the response queue

After processing the request, the worker publishes the reply

Response Queue

X

C / P

Delayed Messages

Messages that will be delivered later are routed to a wait queue

The wait queue is declared with a message TTL and a dead-letter exchange

P → X → Wait Queue

When a message times out due to the TTL, it is published to the DLX

X → Ready Queue → C

The internally re-published message is routed to the queue with active consumers
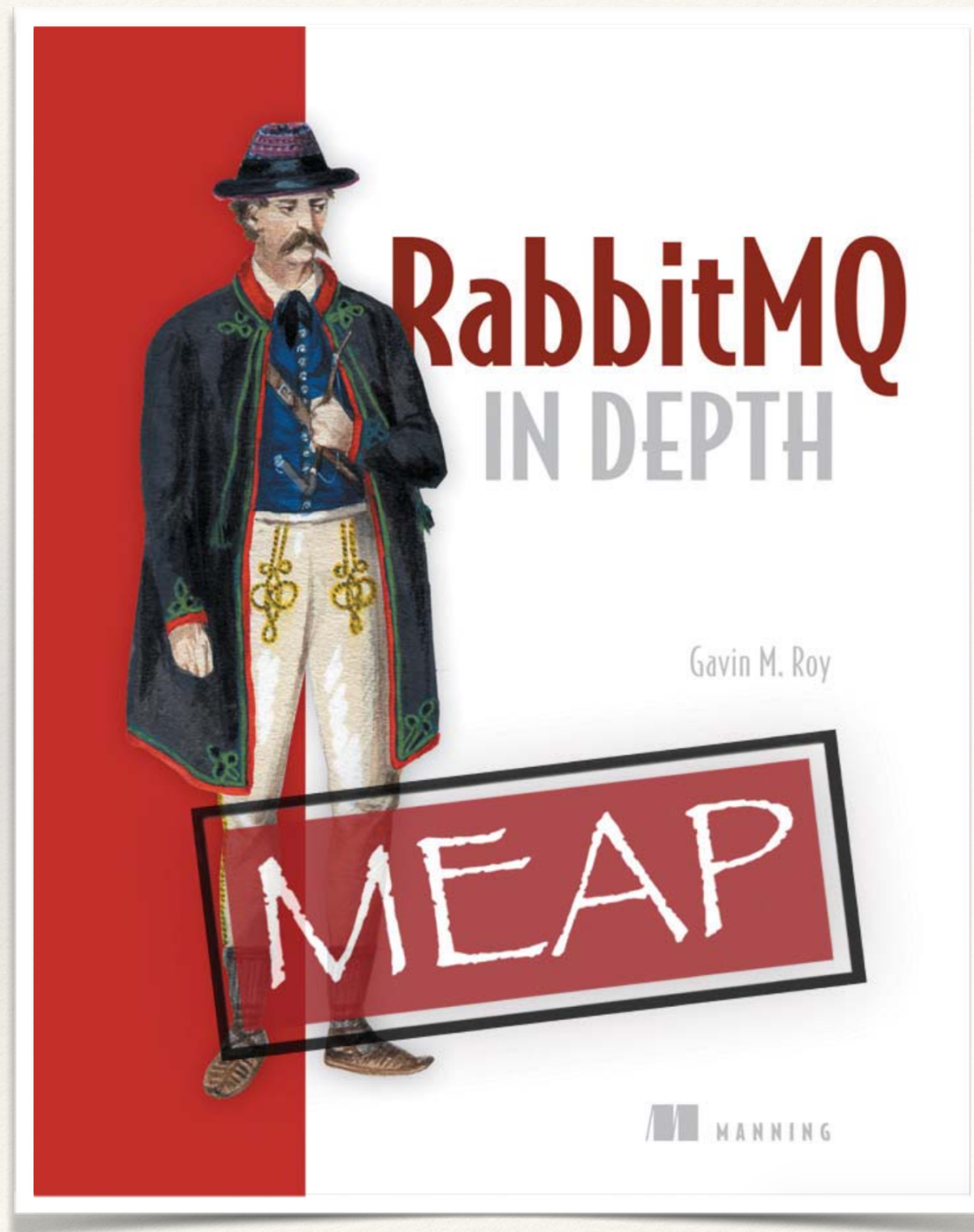
WARNING: This pattern is a hack and can cause undesirable situations should the wait queue overwhelm the system!

Questions?

manning.com

Discount code

ato2014cftw