April , 2021

RabbitMQ

BuildingBlocks and messaging patterns

RabbitMQ

The Building Blocks

Routing
Fanout Exchange
Direct Exchange
Topic Exchange
Header Exchange
Consistent Hashing Exchange
Random Exchange
Sharding

Queues and Consumers
Queues
Competing Consumers
Non-Competing Consumers
Ephemeral Queues
Lazy Queues
Priority Queues

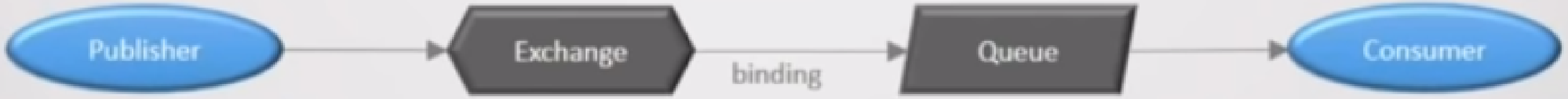
More
Deadletter Exchange
Alternate Exchange
Virtual Hosts

RabbitMQ Building Blocks

Traditional queues on steroids

Exchanges, Queues and Bindings

- Publishers send messages to Exchanges
- Exchanges route messages to Queues and even other Exchanges.
- Consumers read from Queues
- Bindings link Exchanges to Queues and even Exchanges to Exchanges

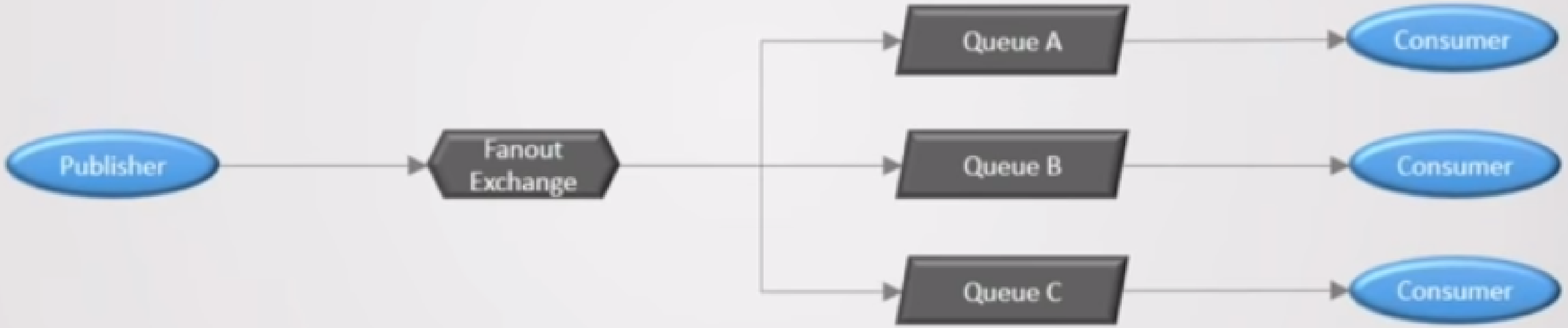


Publish - Subscribe

Fanout

Simplest Publish-Subscribe pattern.

Each consumer receives every message

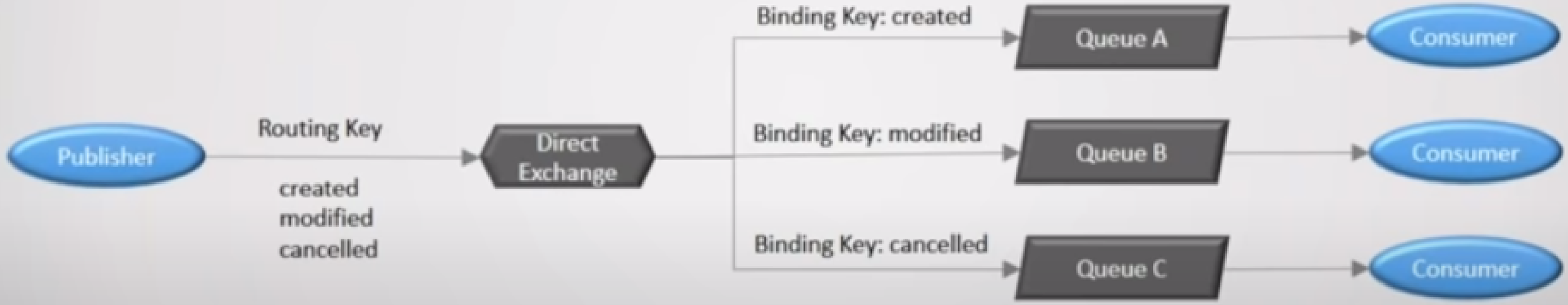


Publish - Subscribe with Routing

Direct Exchange

Routing by **Routing Key**

Exact match routing key = binding key

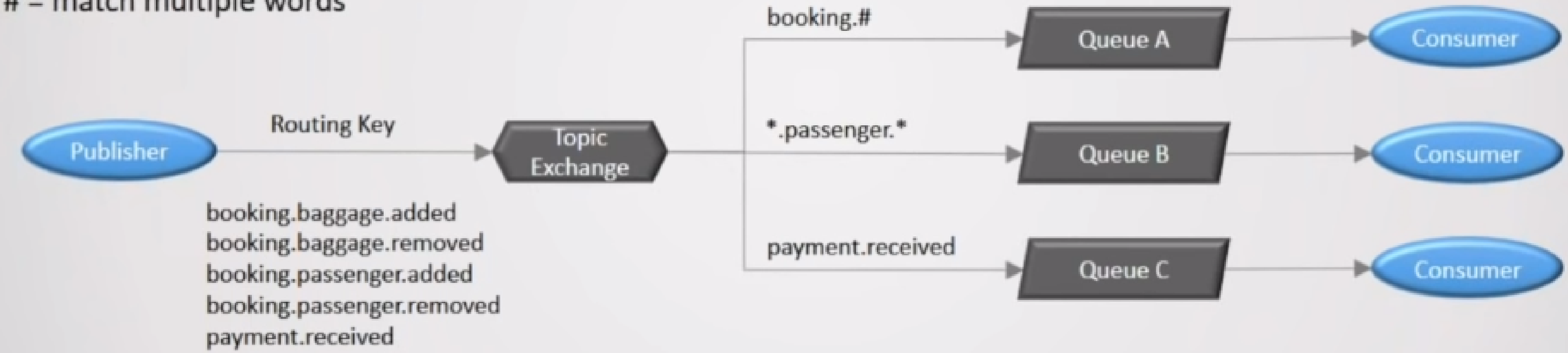


Publish - Subscribe with Routing

Topic Exchange

Wildcard routing by **Routing Key**

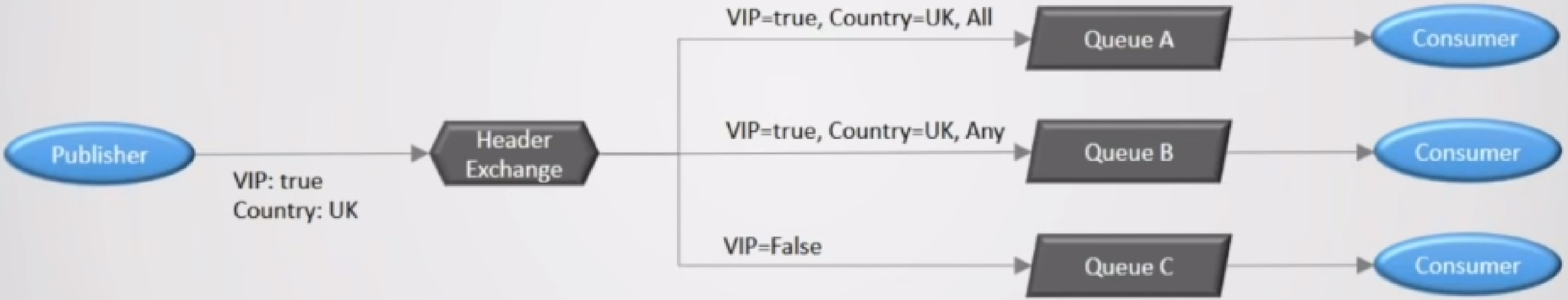
* = match 1 word
# = match multiple words



Publish - Subscribe with Routing

Header Exchange

Routing by message headers



Point-to-Point Messaging

Default Exchange

Direct exchange type

All queues have implicit binding to the default exchange.

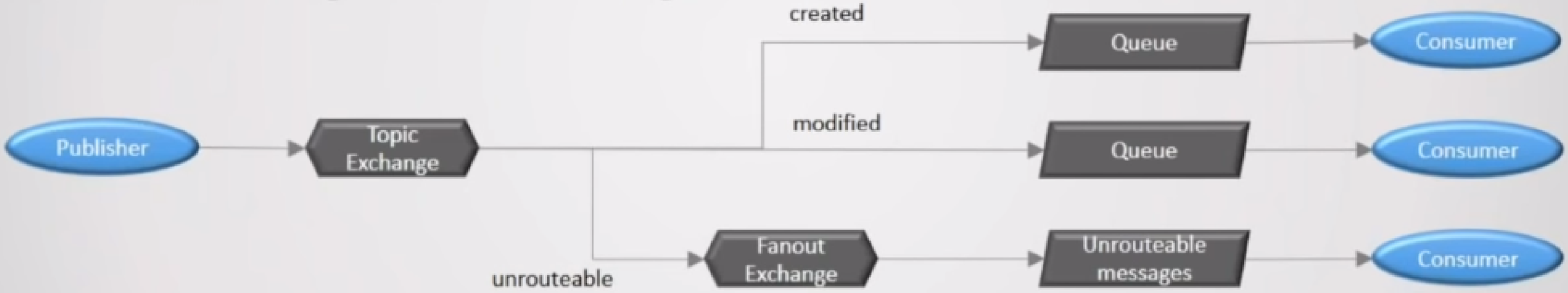
Routes messages by Routing Key to Queue Name

Protecting Against Routing Failures

Alternate Exchange

Not an exchange type but a configured relationship between two exchanges.

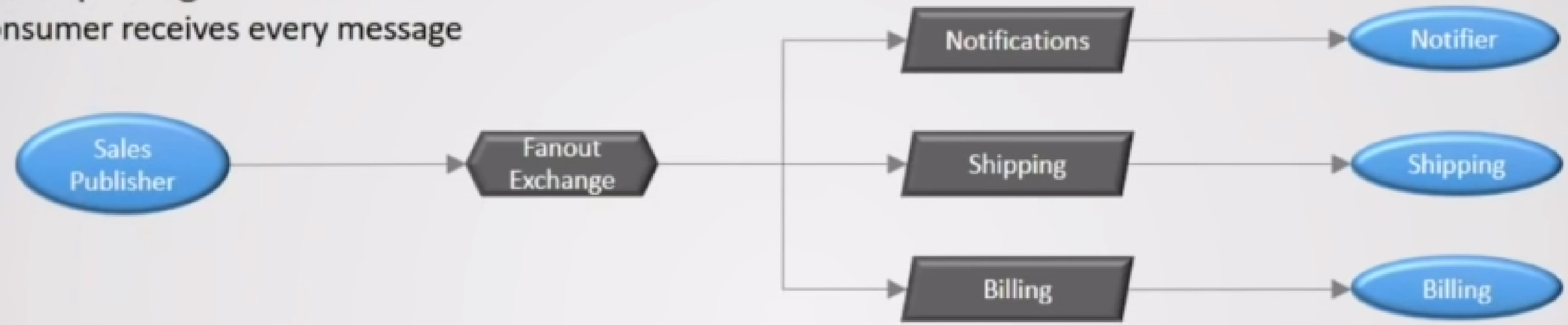
Route unrouteable messages to an alternate exchange



Scaling Out Consumers

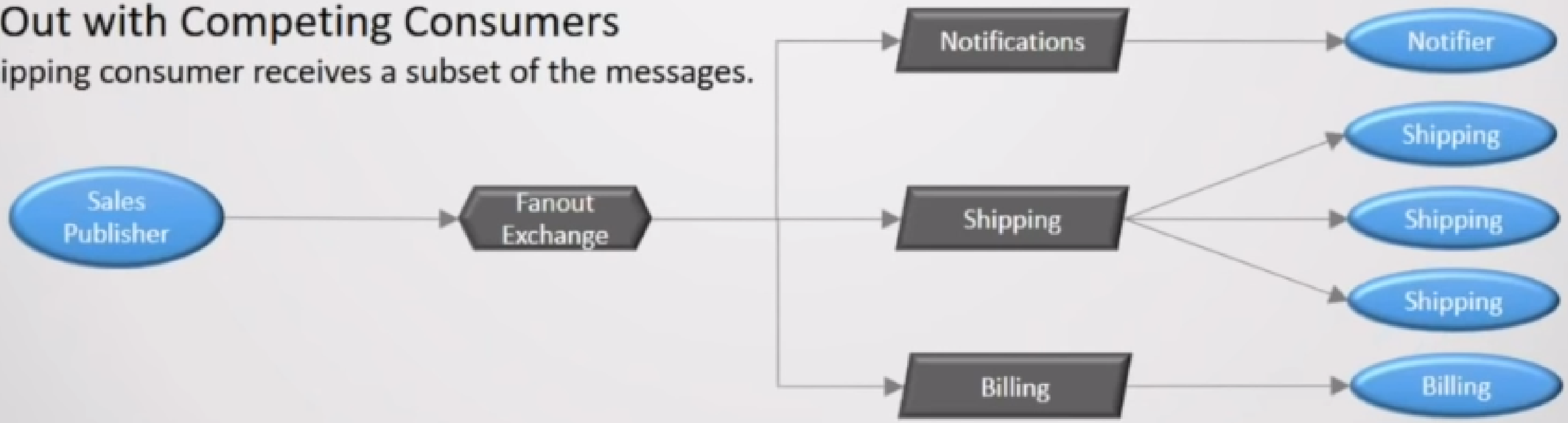
Non-Competing Consumers

Each consumer receives every message



Scale Out with Competing Consumers

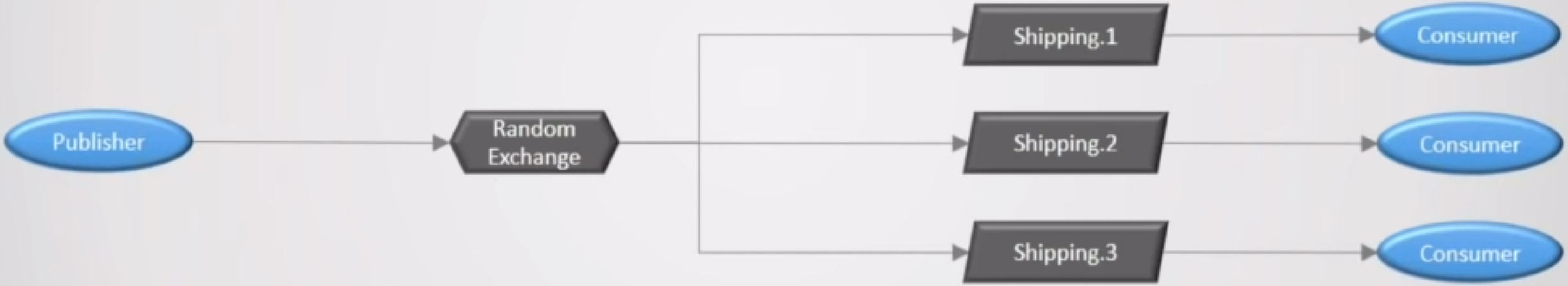
Each Shipping consumer receives a subset of the messages.



Scaling Out Queues

Random Exchange

Load balance messages across queues randomly



Scaling Out Queues With Data Locality

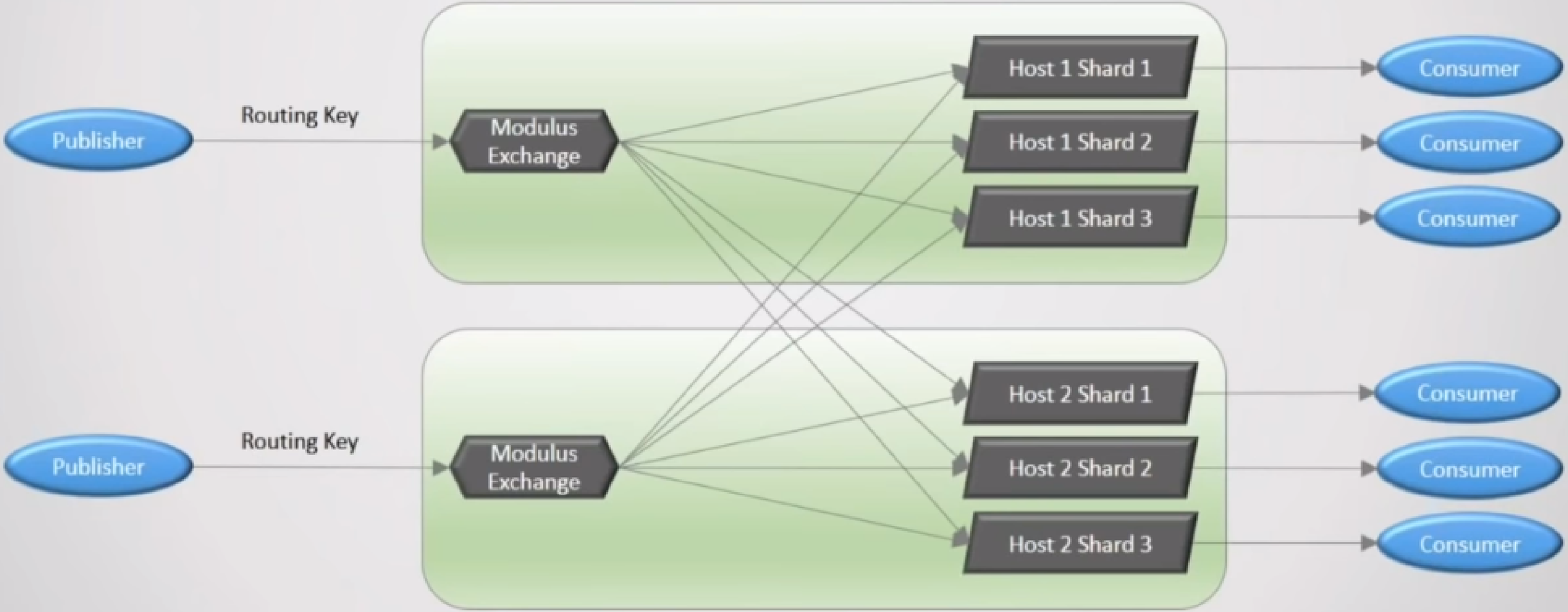
Consistent Hashing Exchange

Partition messages across queues via hashing function over routing key, message header or message property

Scaling Out Queues With Data Locality

Sharding Plugin and Modulus Exchange

Partition messages across queues over multiple hosts via hashing function on the routing key.



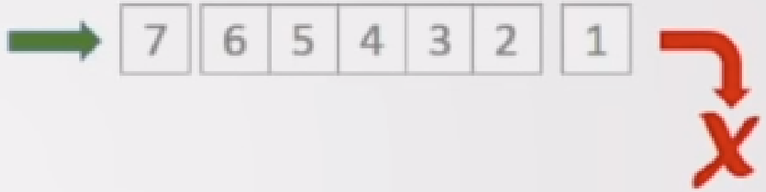
Queue Limits and Deadletter Exchanges

Apply limits to queues

- Length
- Size
- Time limits (TTL)

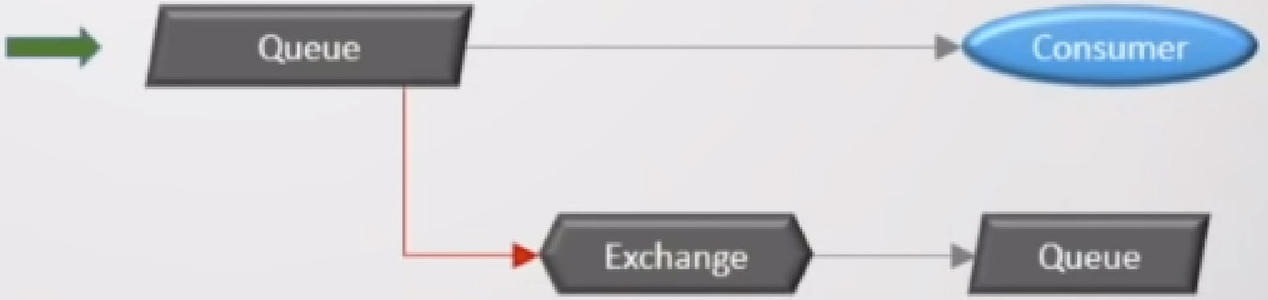
Eject messages from a queue when:

- A queue size/length limit reached.



- A message has spent longer than the TTL time limit in the queue

Route to a deadletter exchange to avoid message loss.



Ephemeral Exchanges and Queues

Lazy Queues

Priority Queues

Ephemeral

- Auto-Delete Queue
- Queue Expiration (TTL)
- Exclusive Queues
- Auto-Delete Exchanges

Lazy Queues

- Memory optimized queues.

Priority Queues

- No longer FIFO.
- Publisher sets priority on messages
- Priority Queue moves higher priority messages ahead of lower
- Drawbacks – blocked low priority messages, low priority can eject high priority when queue is full

Virtual Hosts

- Allows Multi-Tenant architecture
- Isolation of groups of exchanges, queues and users
- No routing between two virtual hosts

Decoupling of publishers from subscribers

Endpoint Decoupling

RabbitMQ: The endpoint we publish to is decoupled from the endpoint we consume from.

Kafka: The endpoint we publish to is the same as we subscribe from.

Temporal Decoupling

RabbitMQ: Consumers read a message once, most likely close to the time of publishing.

Kafka: Consumers are decoupled temporally from the publisher. Consumers can read messages multiple times, and go back in history to read old messages.

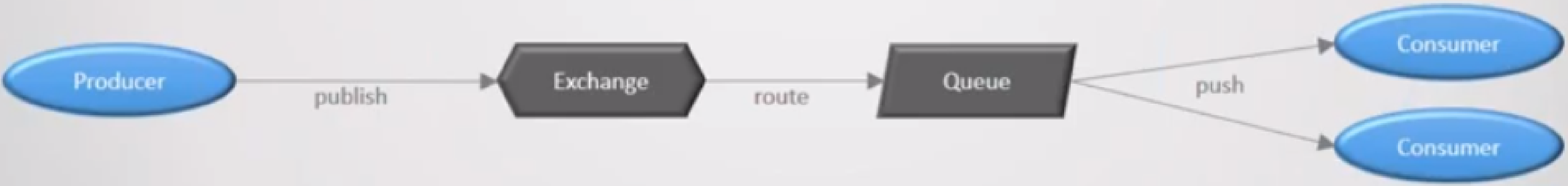
RabbitMQ - Push Model

Producer Publish

- Send messages one at a time

Consumer Push

- Long-lived TCP connection
- Consumer registers interest in queues
- Broker pushes messages down connection in real-time



RabbitMQ – Why Push?

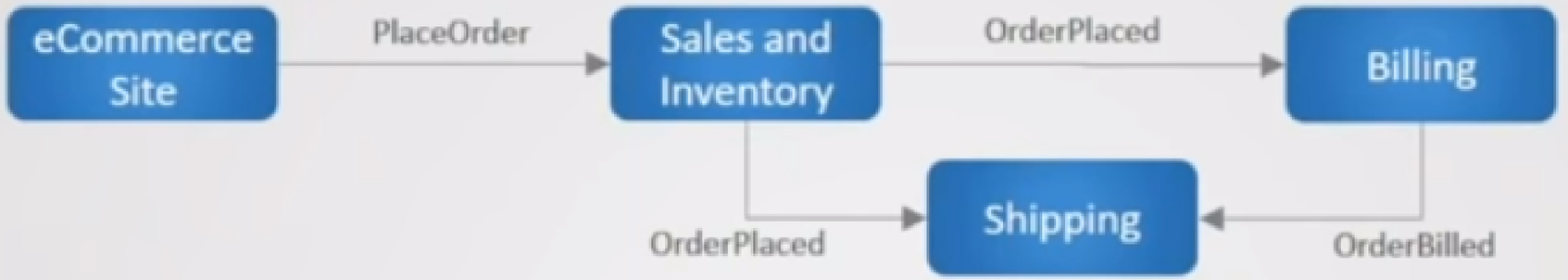
The push model allows RabbitMQ to:

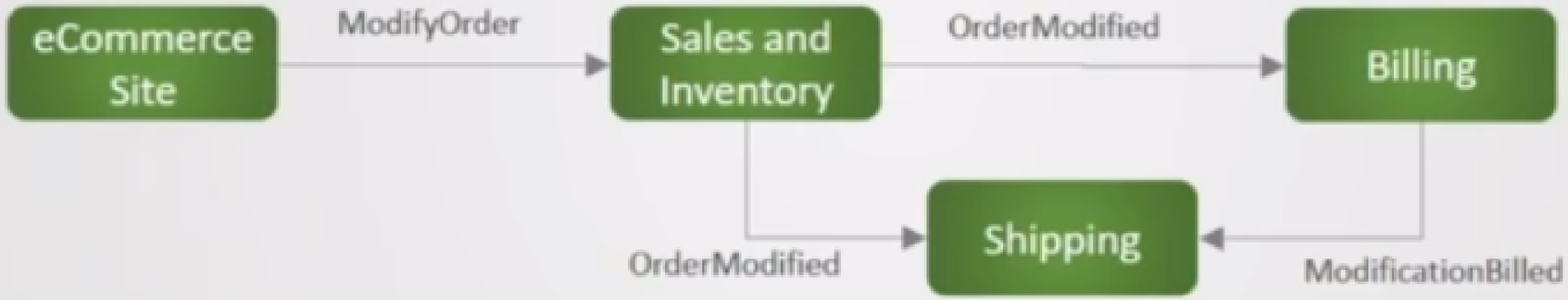
- Offer low latency messaging.
- Evenly distribute messages across competing consumers.
- Keep processing order closer to delivery order in the face of competing consumers.

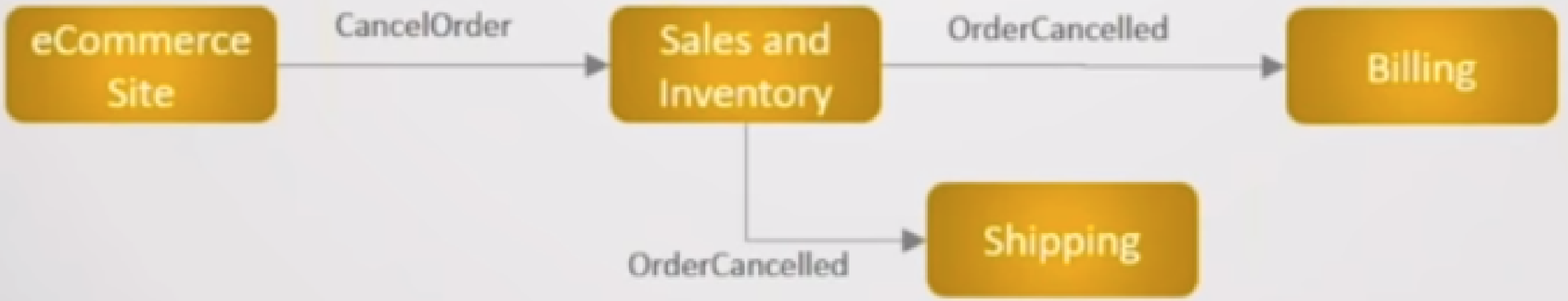
A push model requires Back-Pressure: Consumer Prefetch.

CASE STUDY

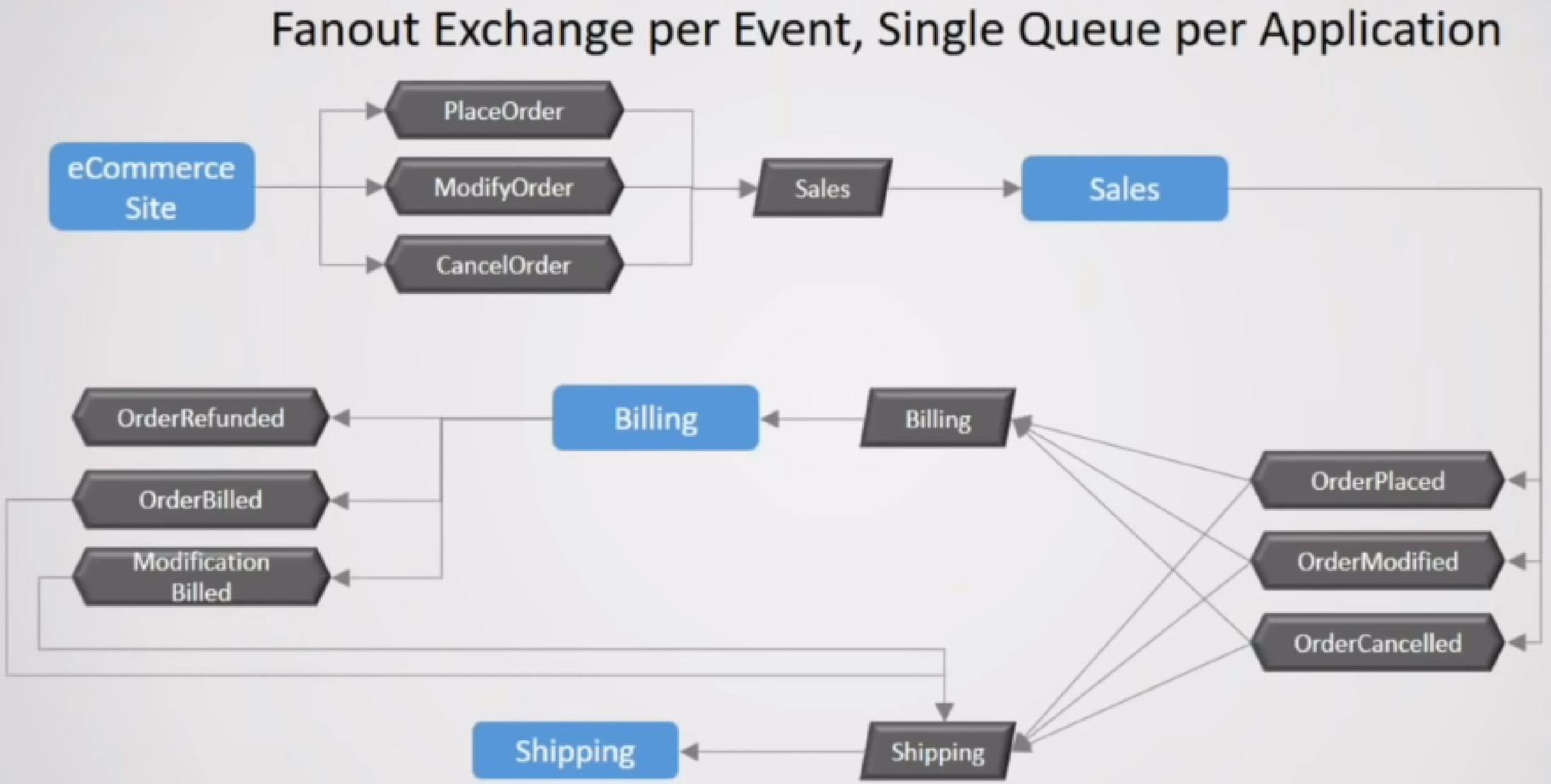
Example Scenario







Example Scenario - RabbitMQ Fanout Exchanges

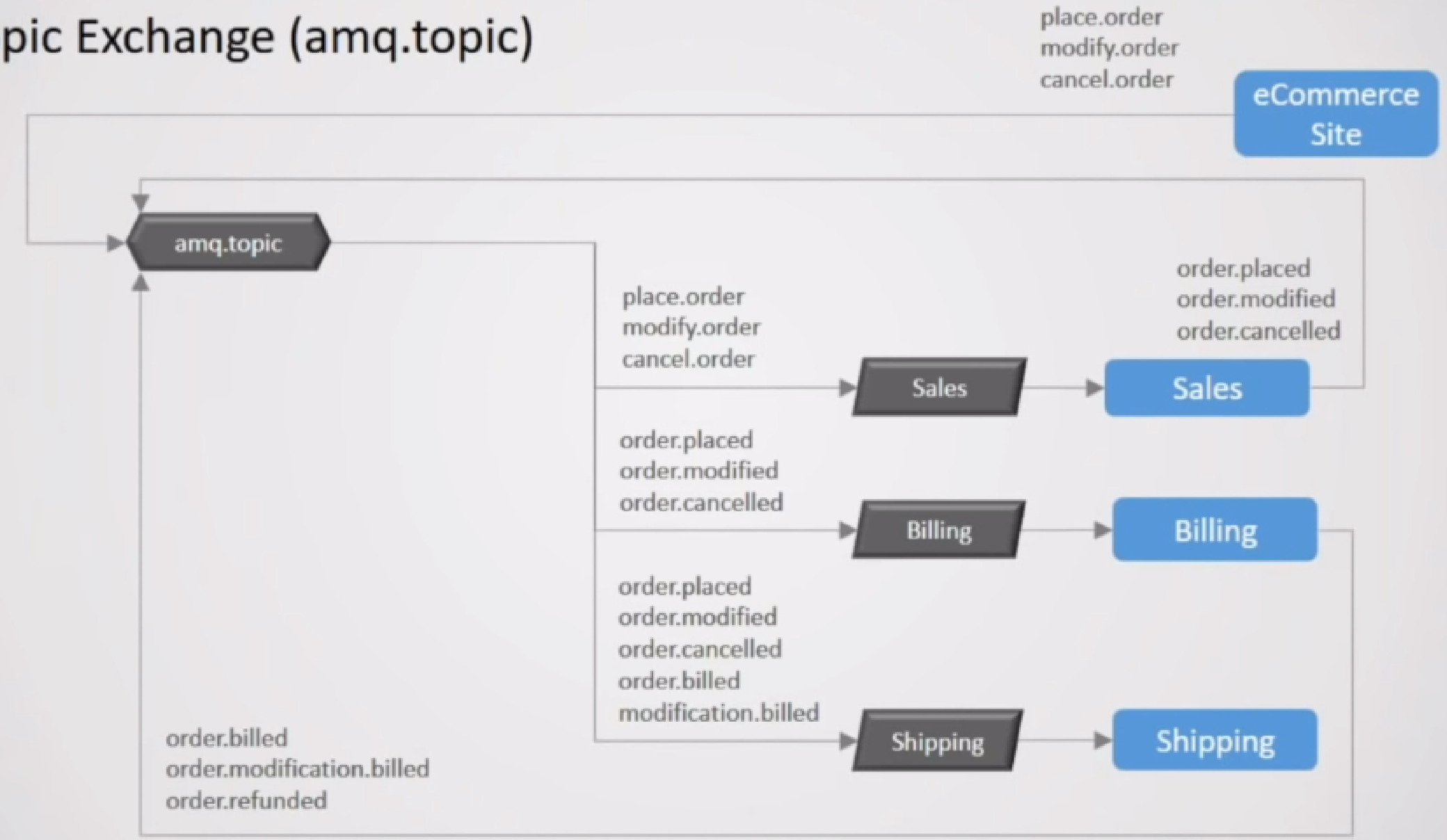


Example Scenario - RabbitMQ Fanout Exchanges

Scaling our Single Queue, Maintaining Relative Ordering

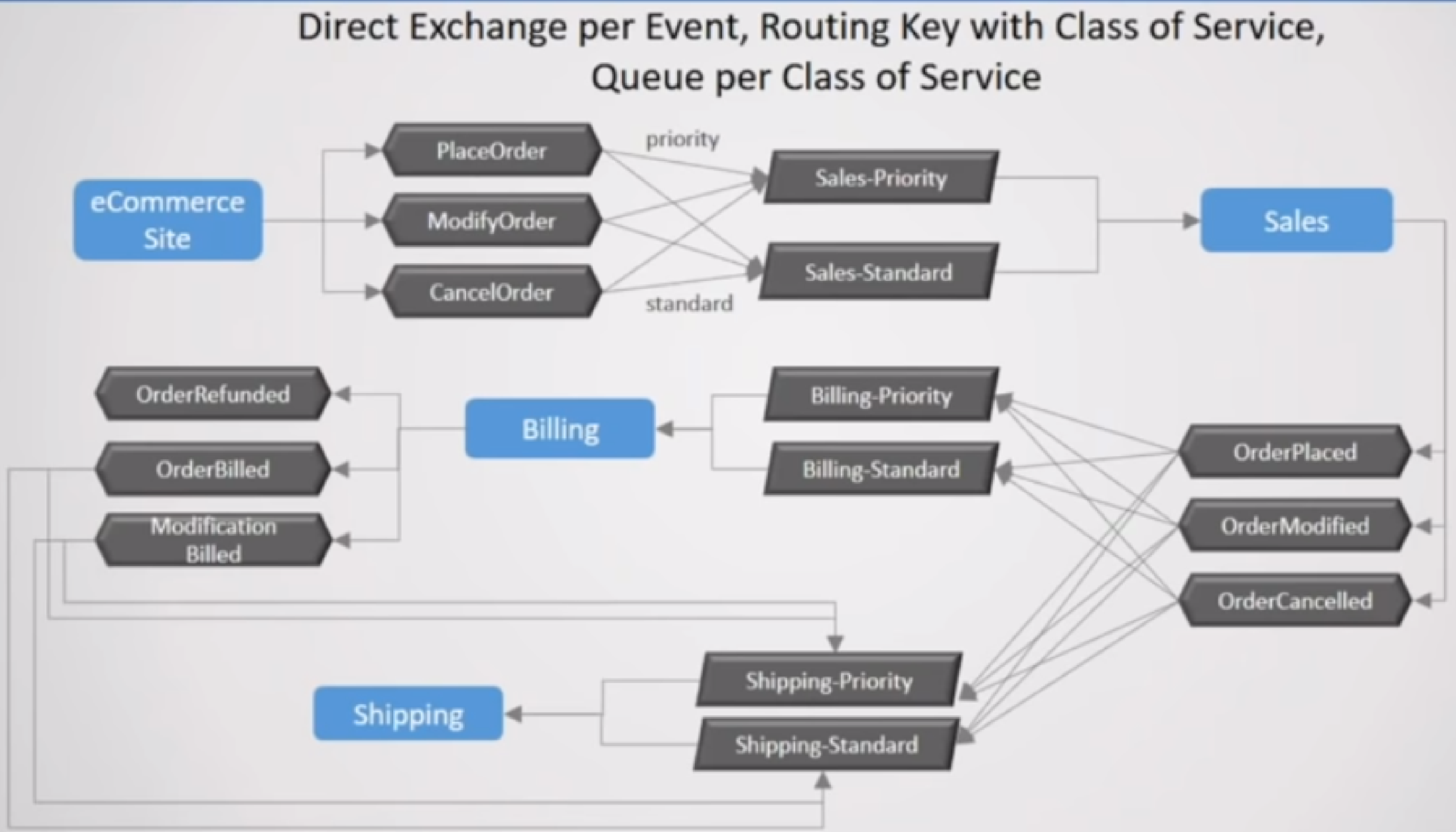
Example Scenario - RabbitMQ Topic Exchanges

Single Topic Exchange (amq.topic)



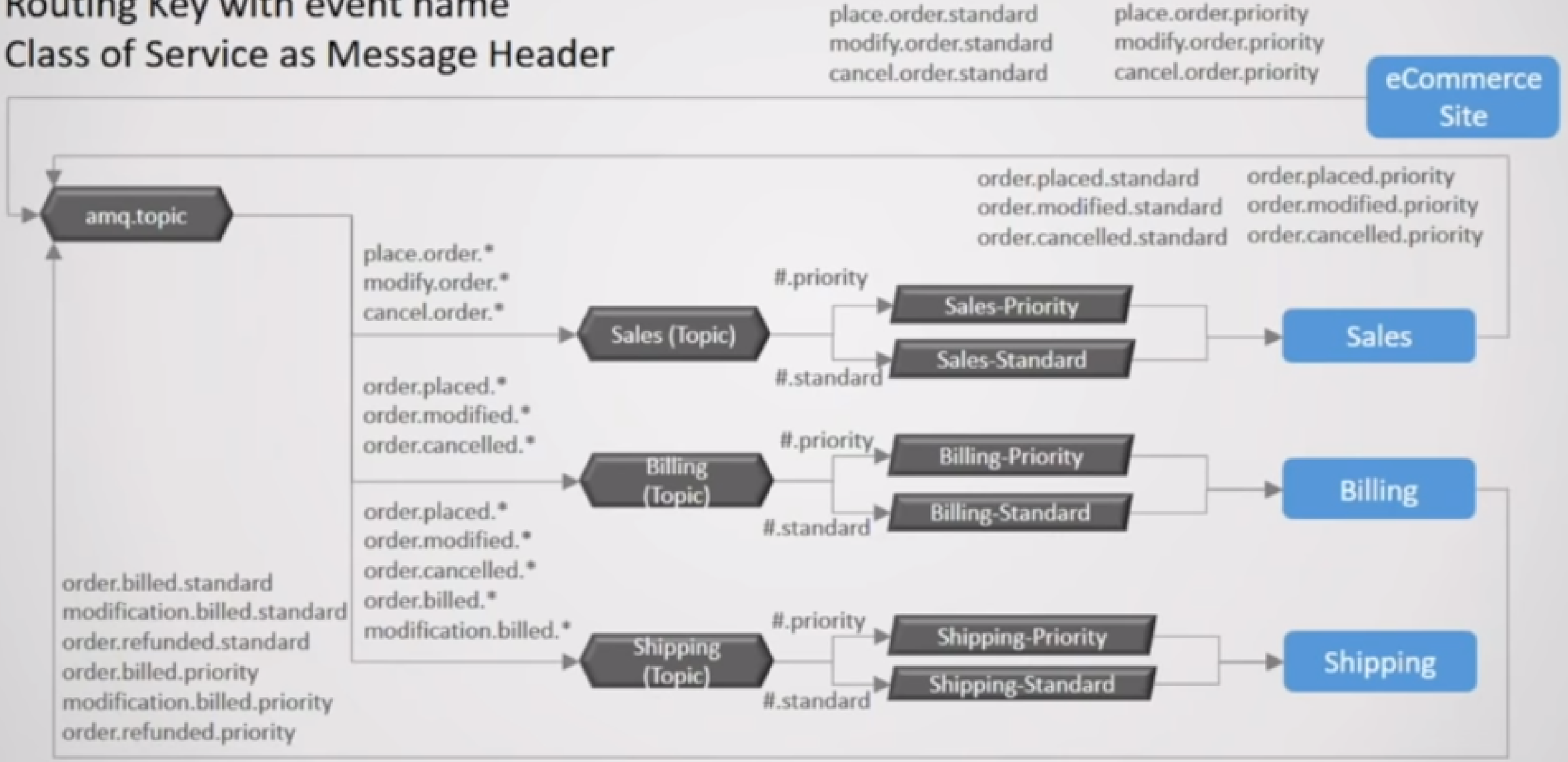
Modifying the Scenario - Class of Service

Scenario #2 - RabbitMQ Topic Exchanges



Scenario #2 - RabbitMQ Two Layered Topic Exchanges

Single Topic Exchange (amq.topic)
Routing Key with event name
Class of Service as Message Header



Scenario #2 – RabbitMQ Topic Exchanges

Replicate preferred routing topology without Class of Service,
in two separate virtual hosts

Priority VHost

Standard VHost