Chicago, October 19 - 22, 2010

Grails + messaging with AMQP/ RabbitMQ

Peter Ledbrook - SpringSource

A history of messaging

A history of messaging

A history of messaging

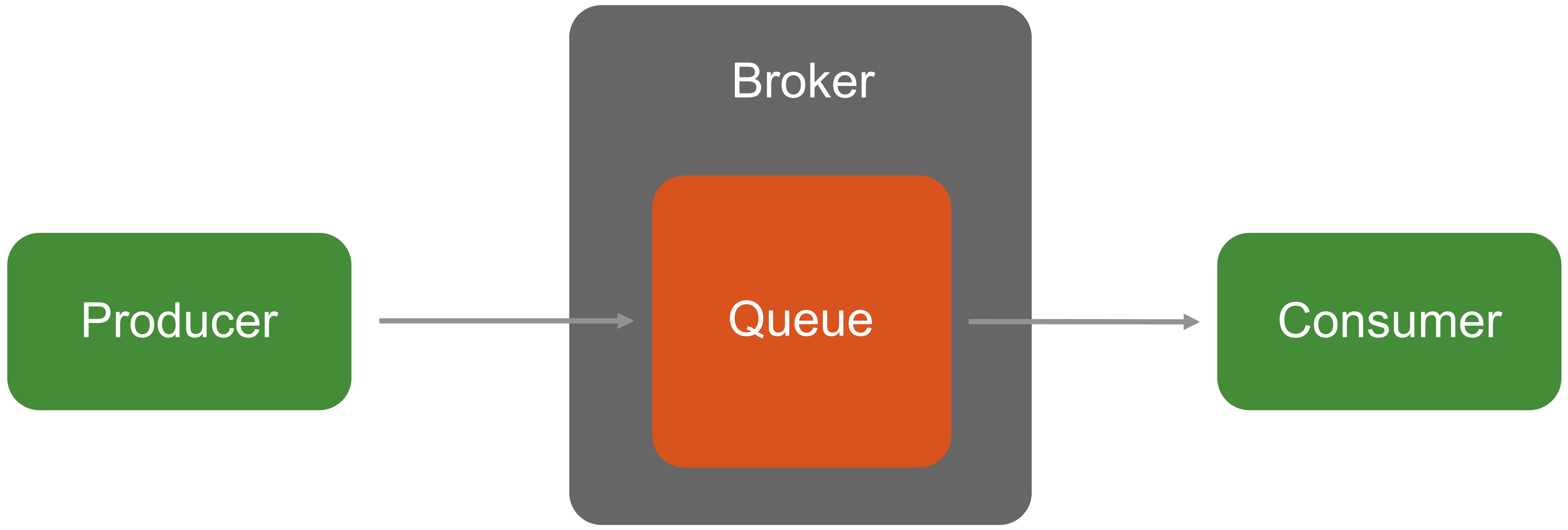
Royal Mail

Email

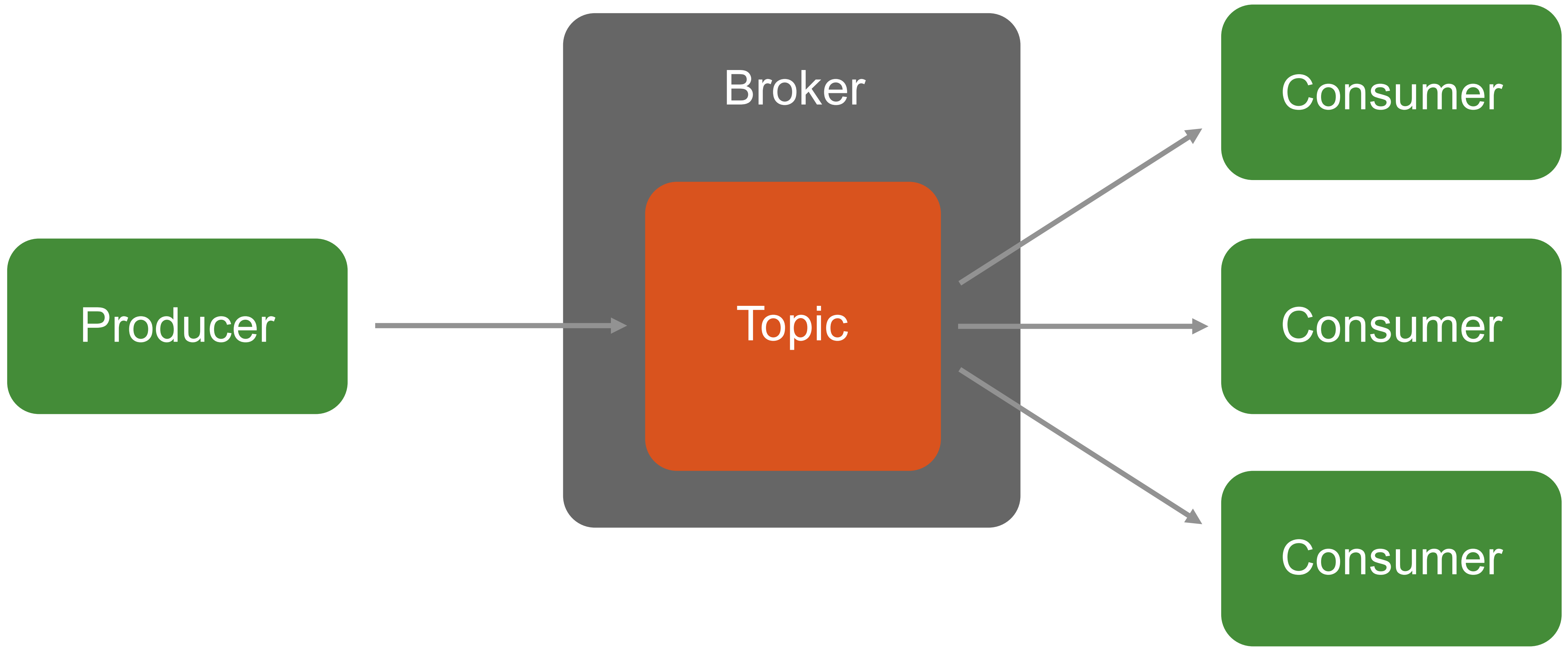
Java messaging - JMS

- Java API
- Synchronous and asynchronous messaging
- Point-to-point and broadcast
  - P2P via Queues
  - Broadcast via Topics
- No standard for communication between client and broker

JMS Queues



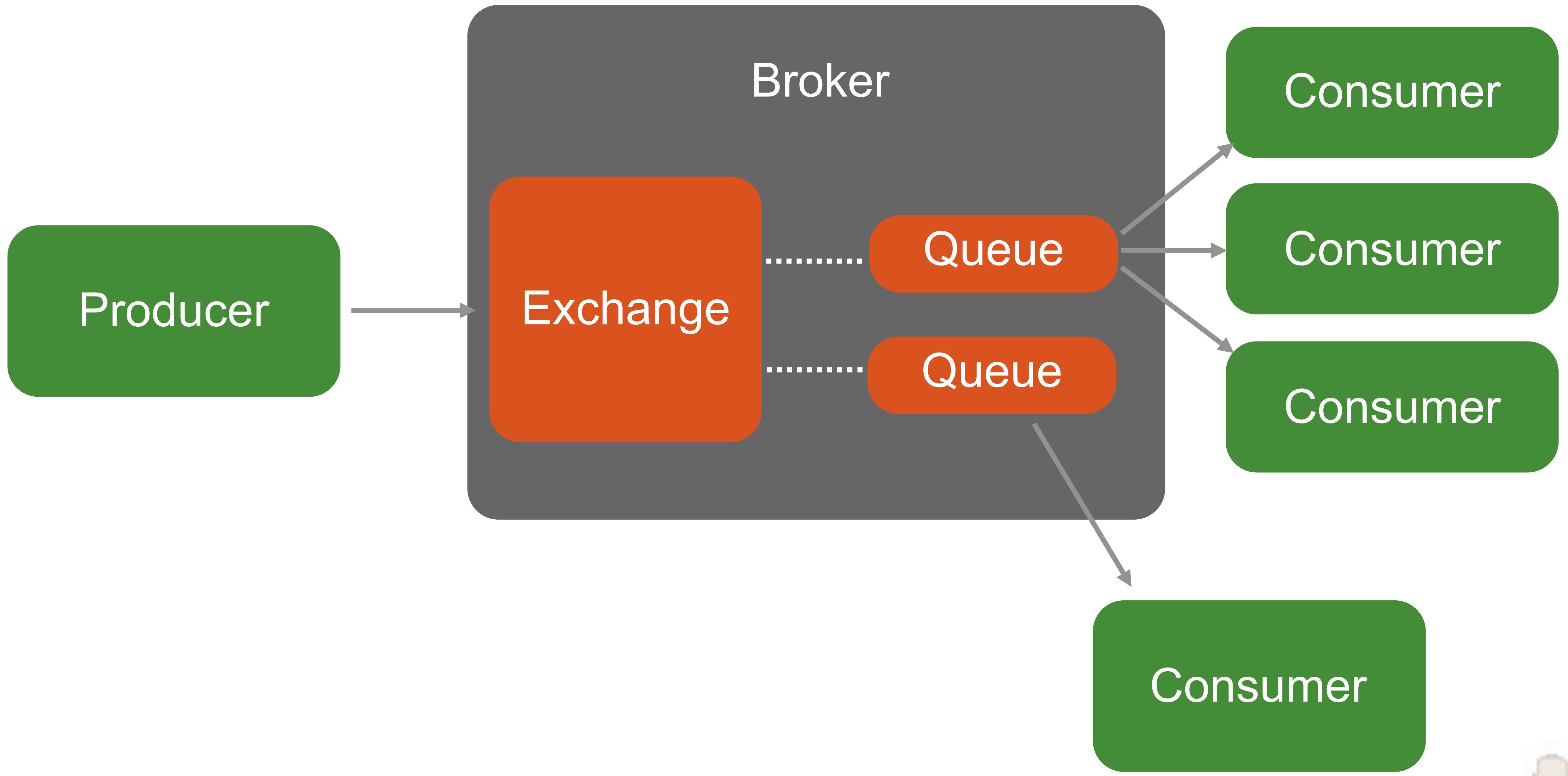
JMS Topics



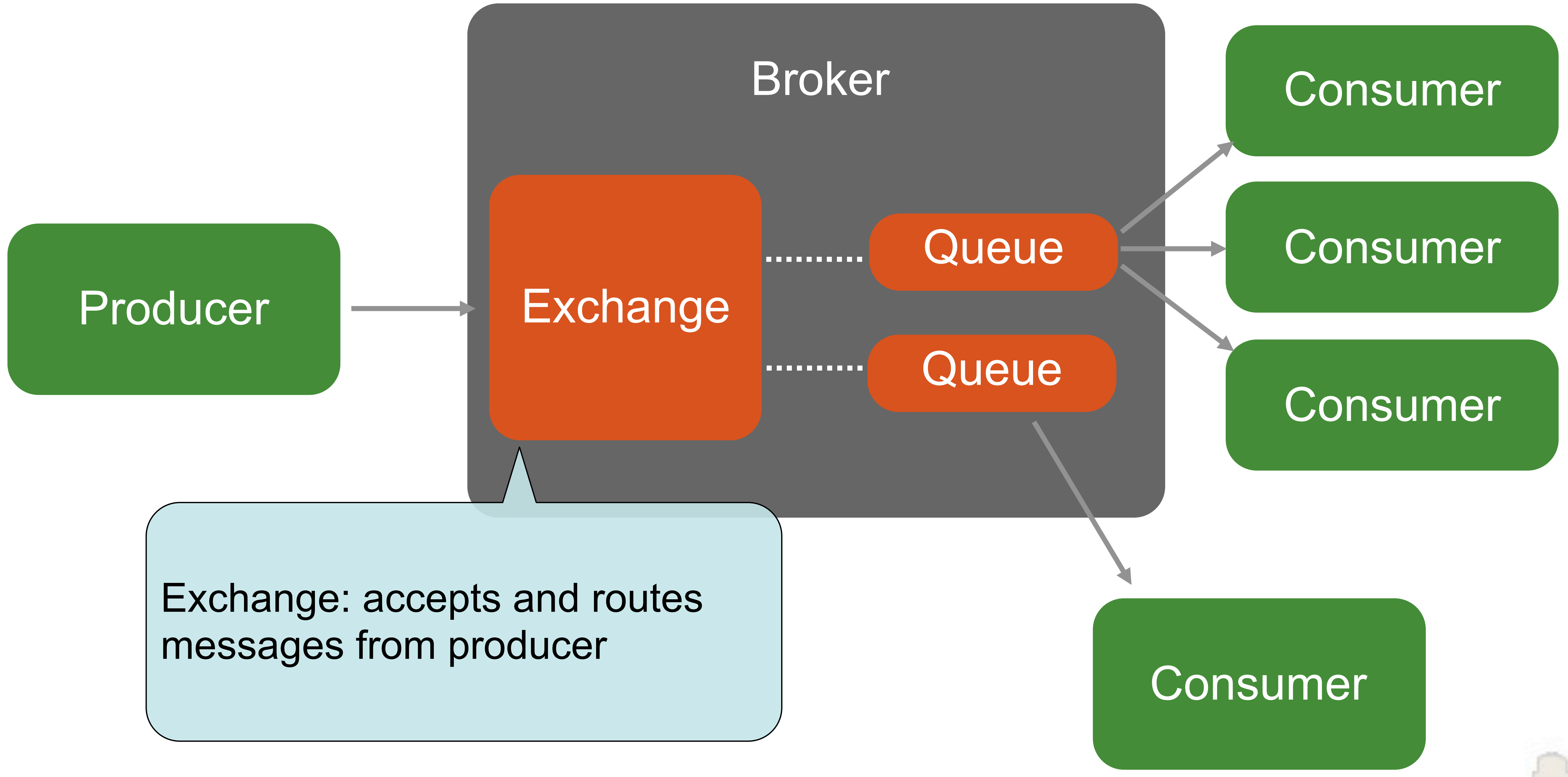
The challenger - AMQP

- Advanced Message Queuing Protocol
- Wire-level protocol
  - Any type of client
  - Client-broker communication standardised
- Synchronous and asynchronous messaging
- Point-to-point, broadcast, and more
  - Single, flexible model
- Simple management part of the protocol
  - Create exchanges and queues

Basic structure

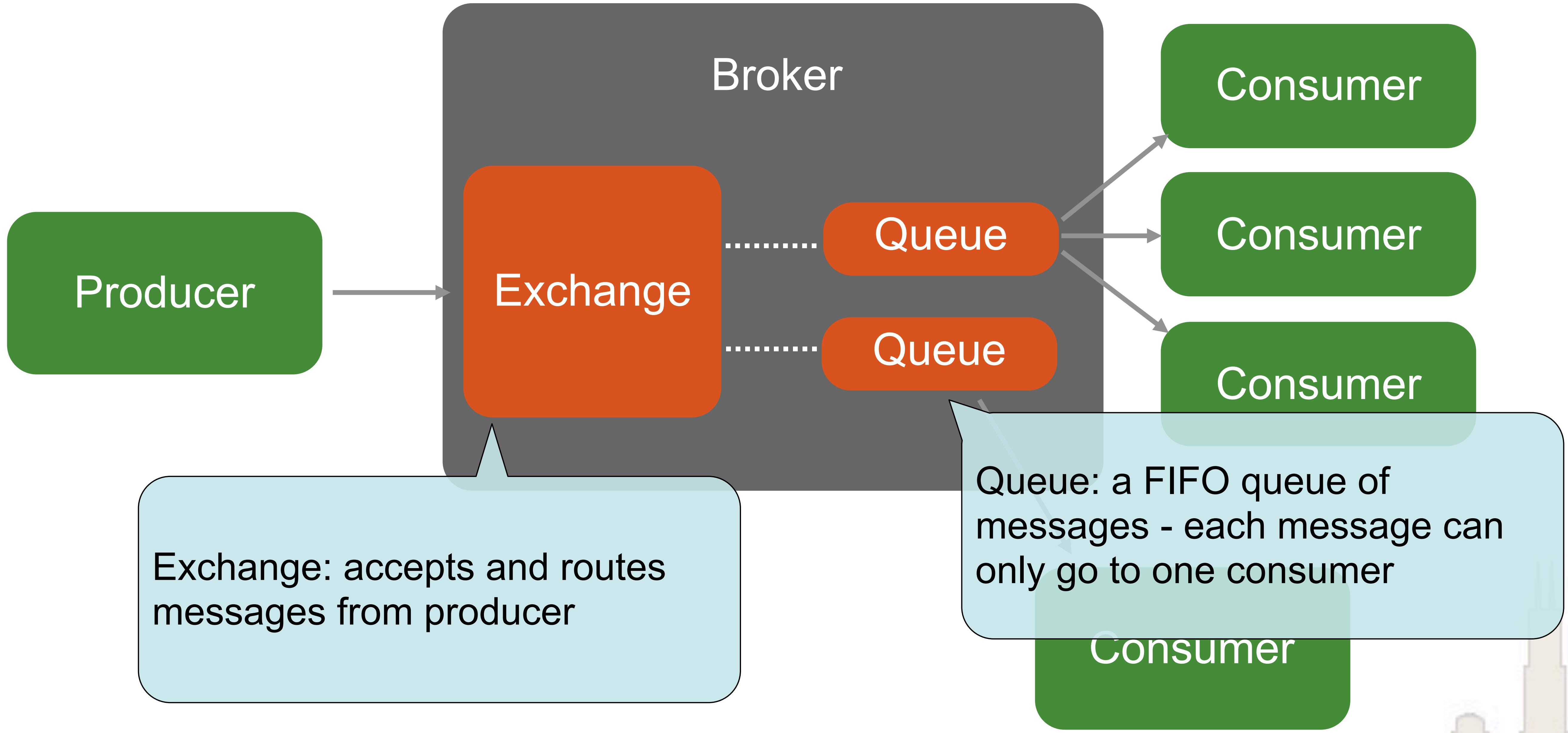


Basic structure



Exchange: accepts and routes messages from producer

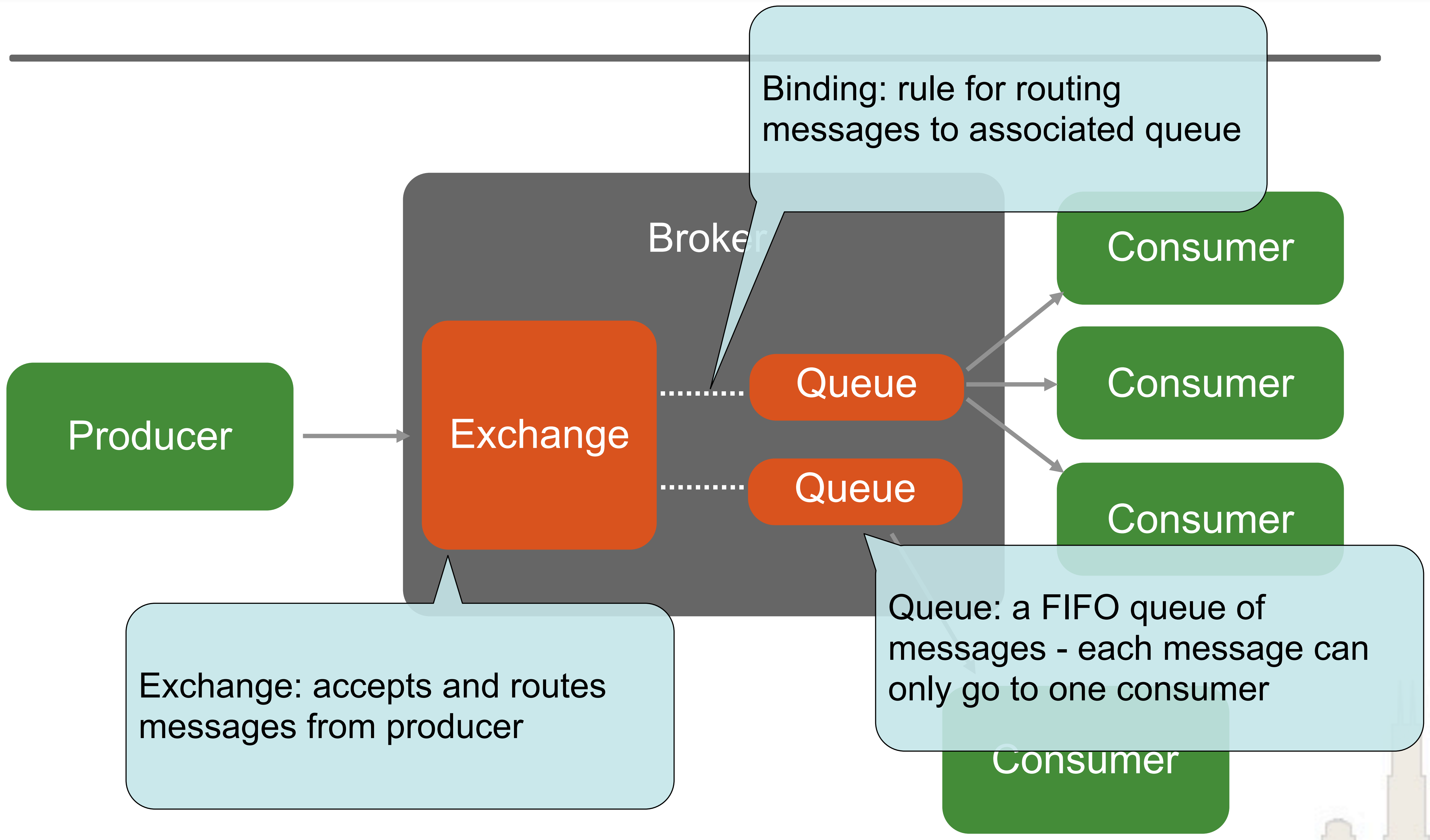
Basic structure



Exchange: accepts and routes messages from producer

Queue: a FIFO queue of messages - each message can only go to one consumer

Basic structure


Binding: rule for routing messages to associated queue
Broker
Consumer
Consumer
Consumer
Consumer
Producer
Exchange
Queue
Queue
Queue: a FIFO queue of messages - each message can only go to one consumer
Exchange: accepts and routes messages from producer

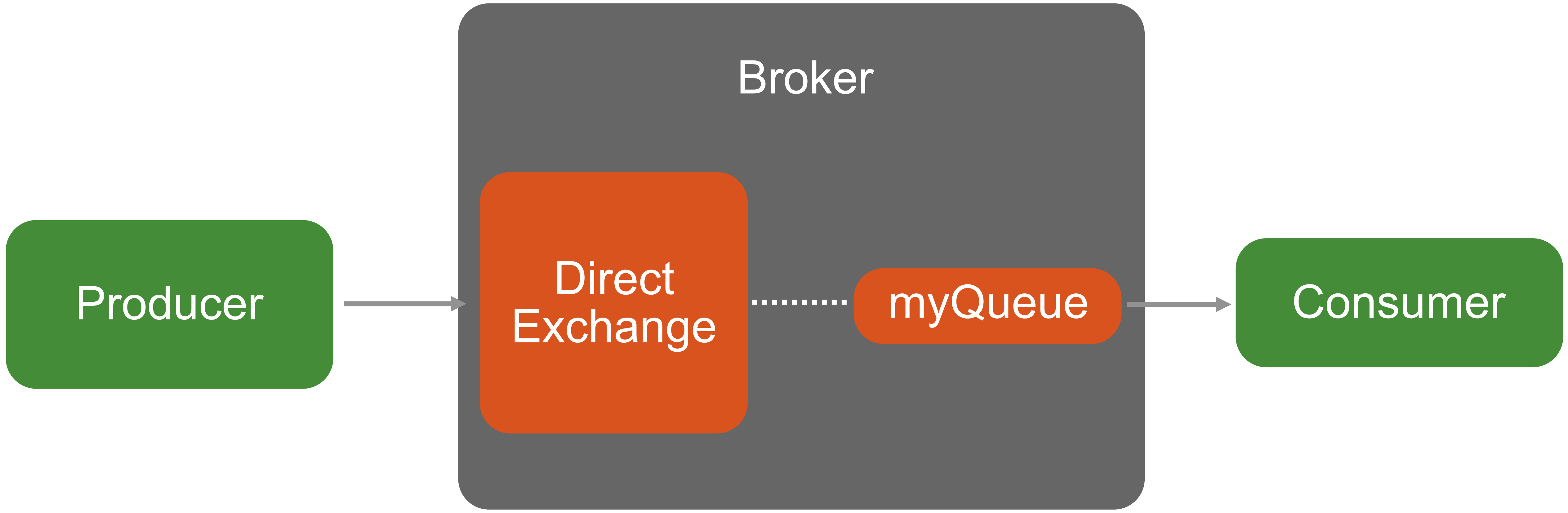
Exhanges

- Only producers talk to the exchange directly
- Message routing depends on
  - Exchange type
  - Message’s ‘routing key’, e.g. “stocks.nasdaq.vmw”
  - Binding between exchange and queue
- Routing and binding keys are typically strings
  - Allow for filtering - similar to JMS selectors

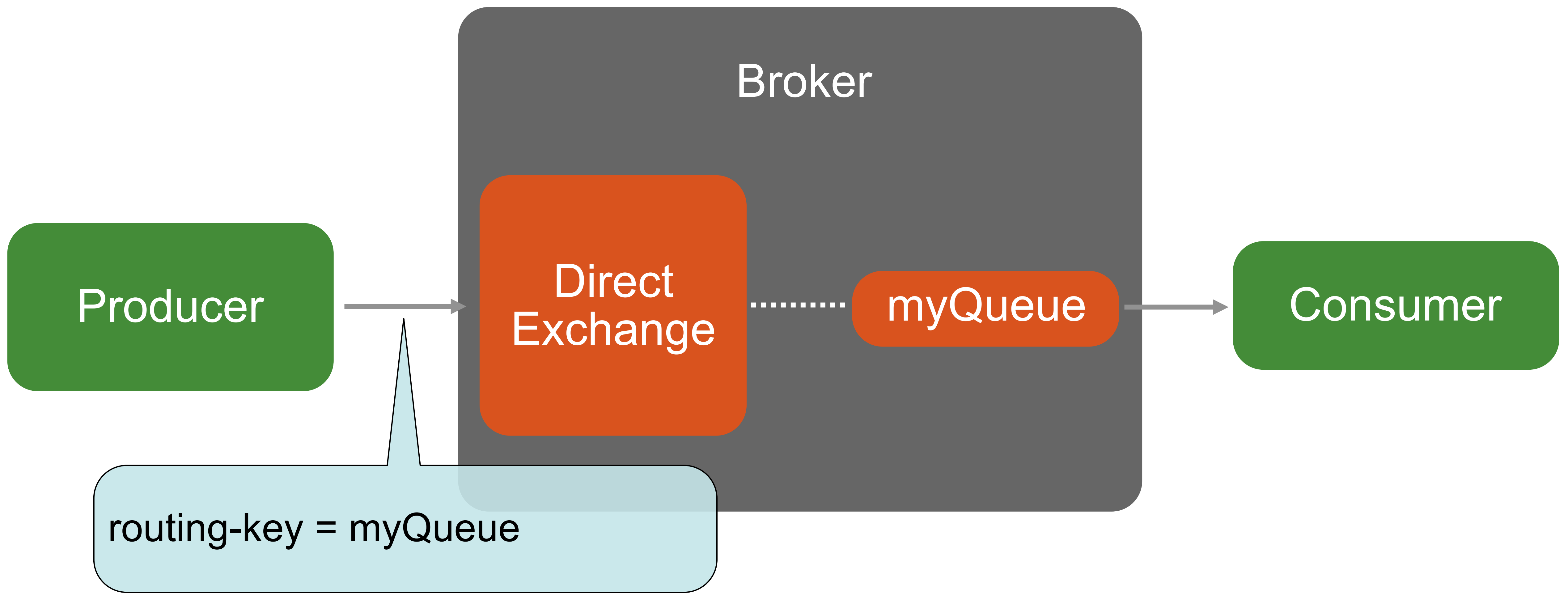
Exhange types

- Fanout
  - Messages go to all bound queues
  - Routing and binding keys are ignored
- Direct
  - Messages only go to queues with a binding key that exactly matches the routing key
  - Typically routing key is the queue name
- Topic
  - Like Direct exchange but binding key can have wildcards
  - ‘#’ like regex ‘*’, ‘*’ like regex ‘?’
- Headers
  - Routing based on message headers

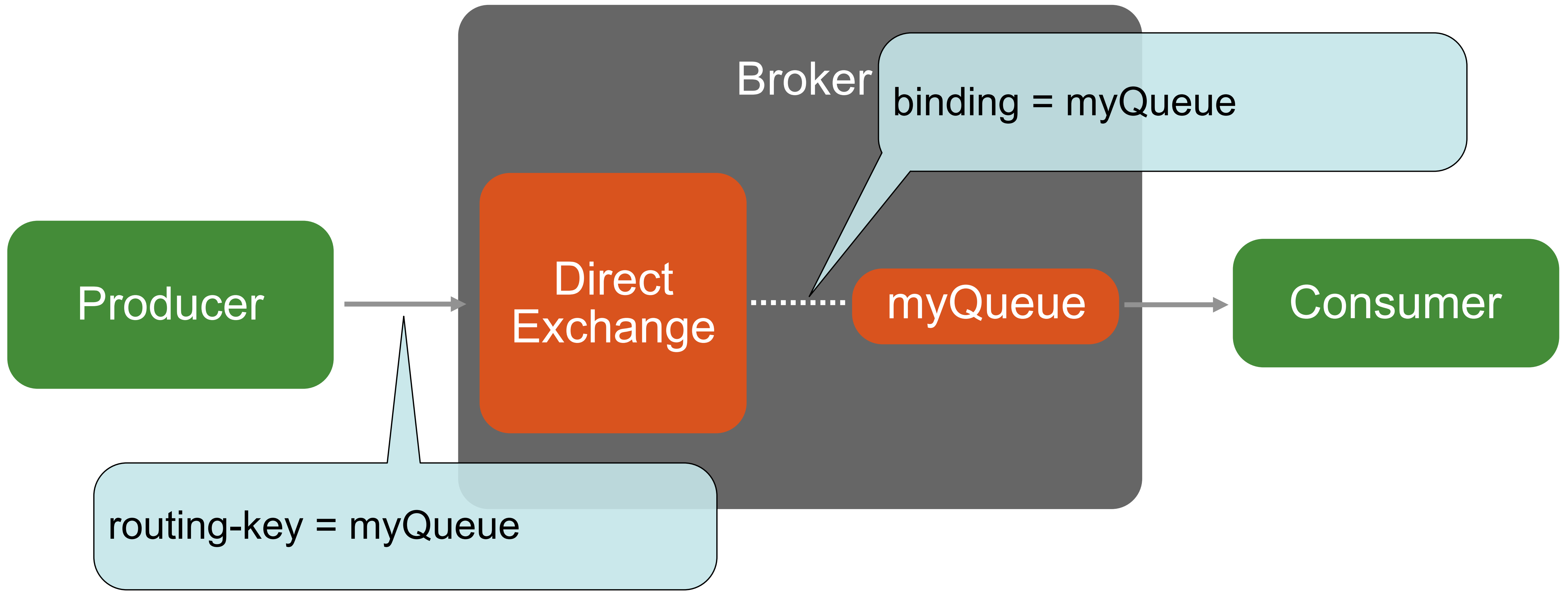
Example: JMS-like Queue



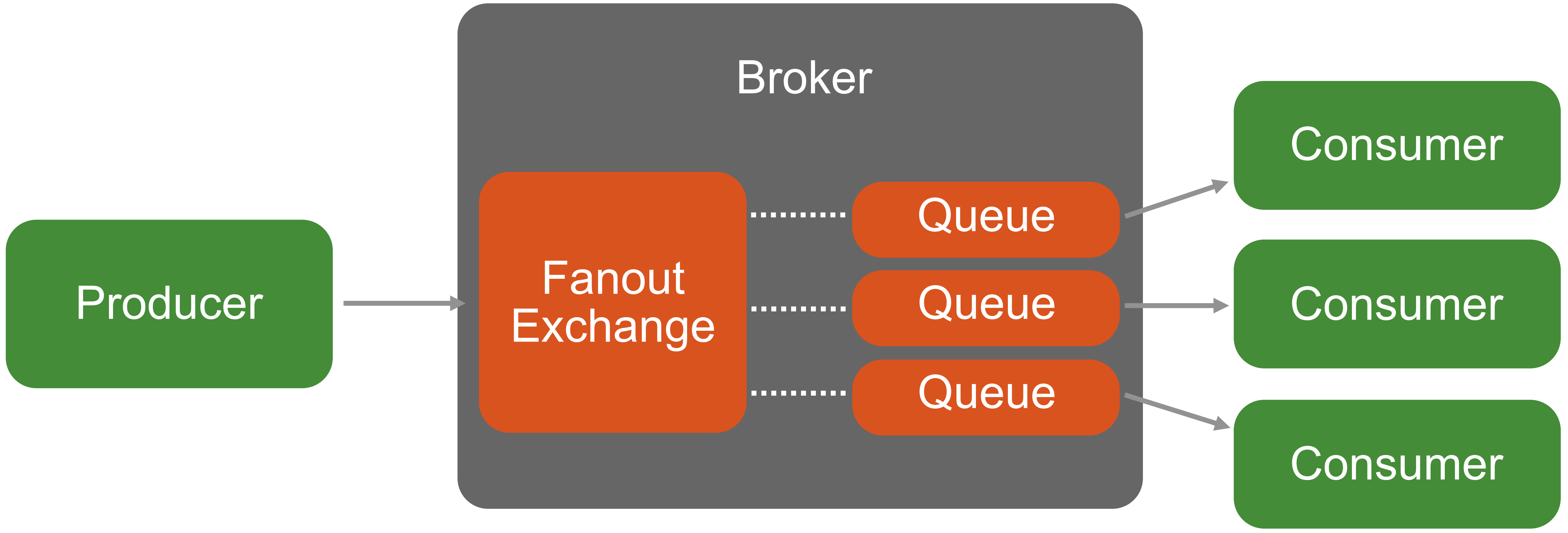
Example: JMS-like Queue



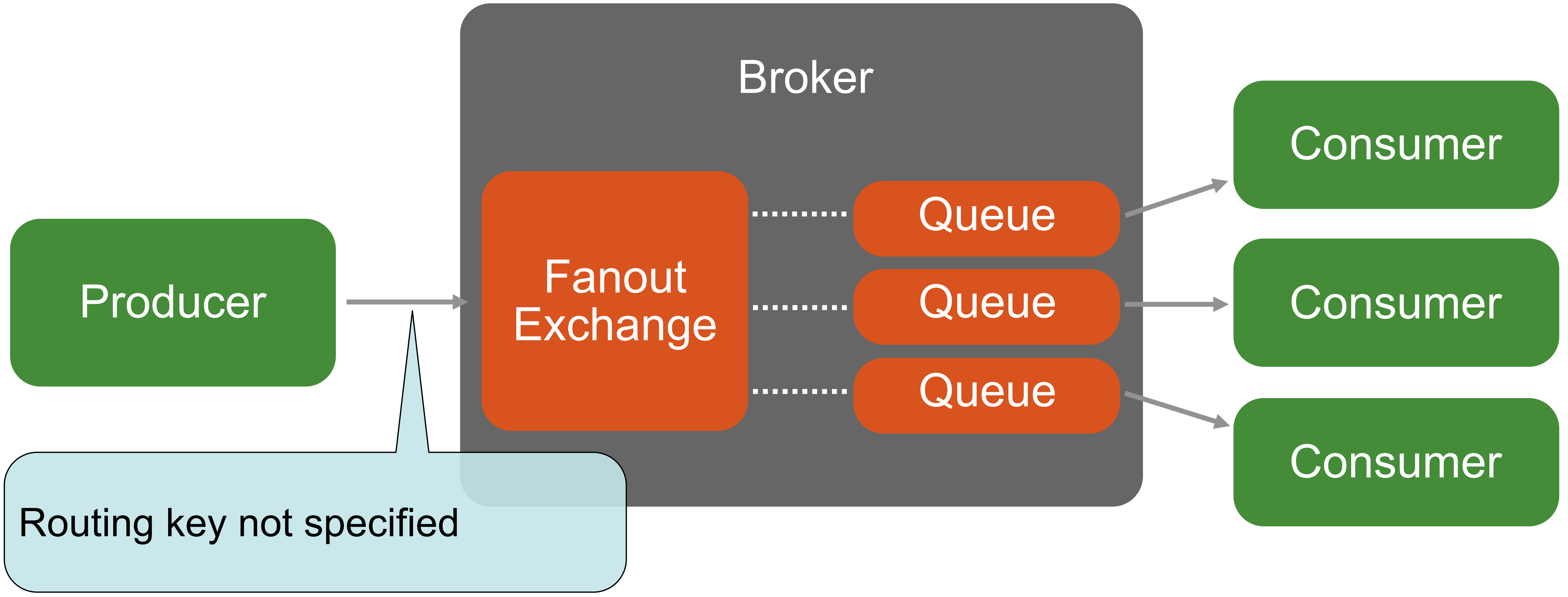
Example: JMS-like Queue



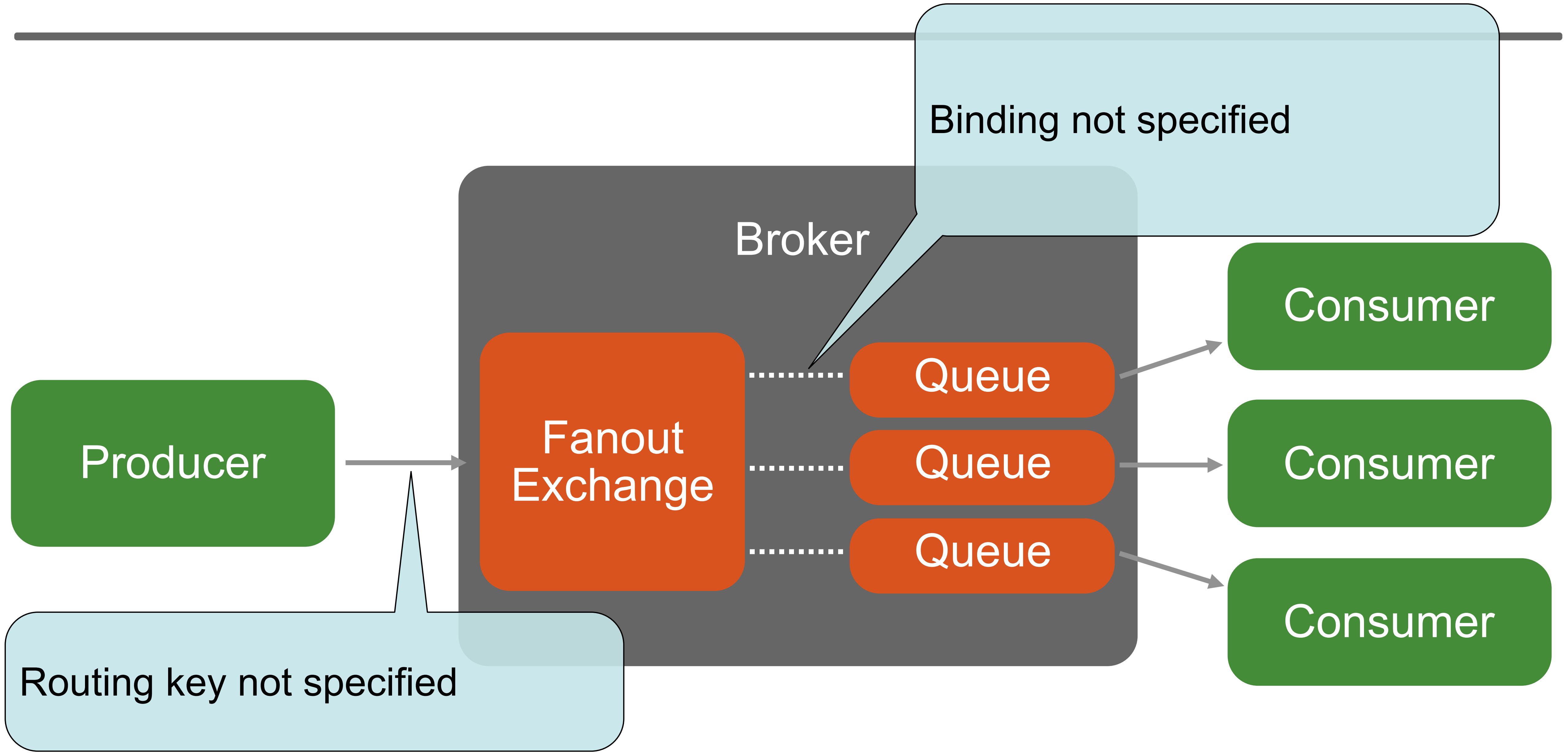
Example: JMS-like Topic



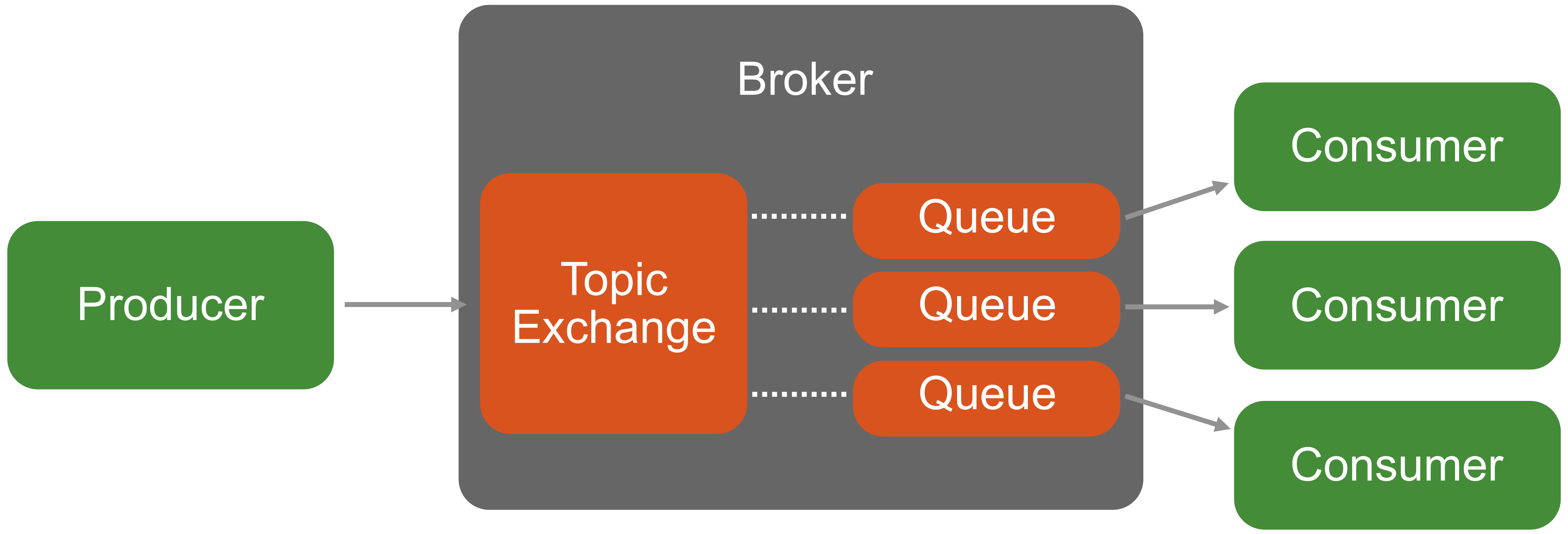
Example: JMS-like Topic



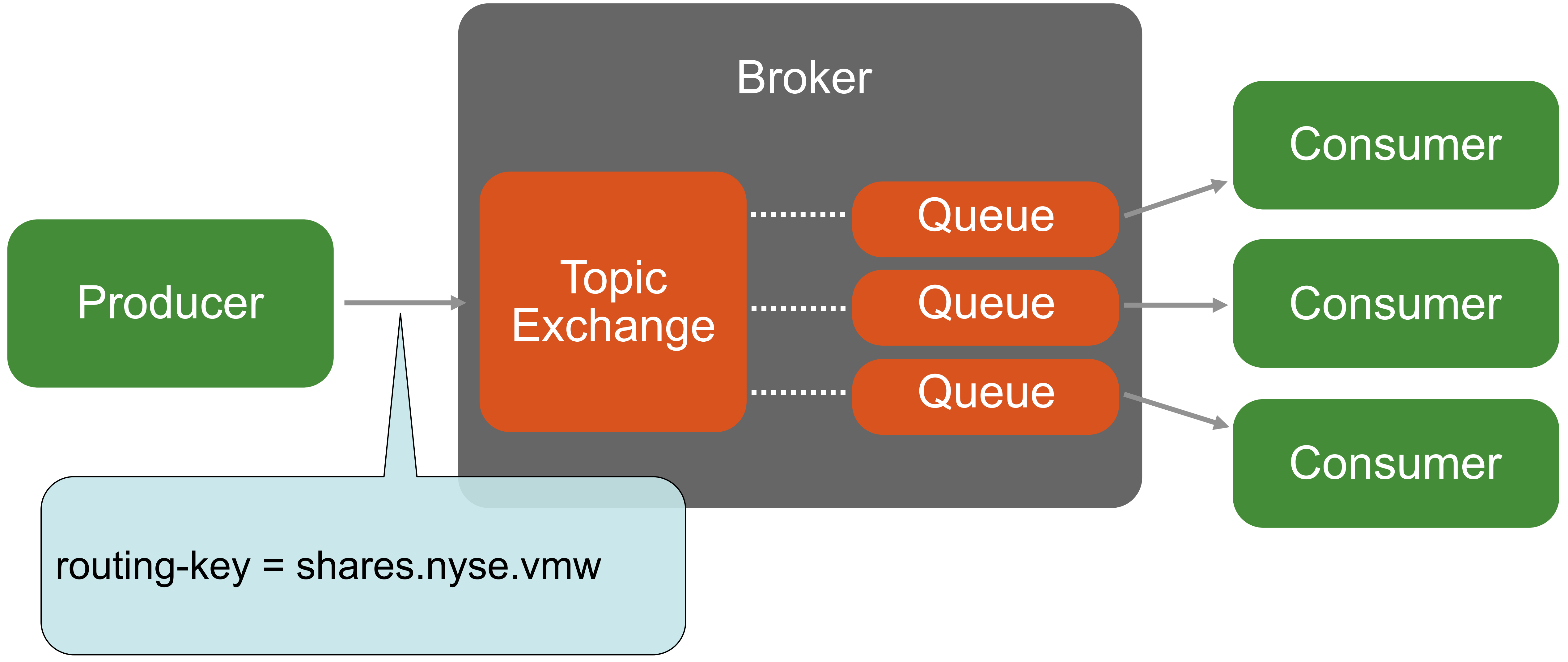
Example: JMS-like Topic



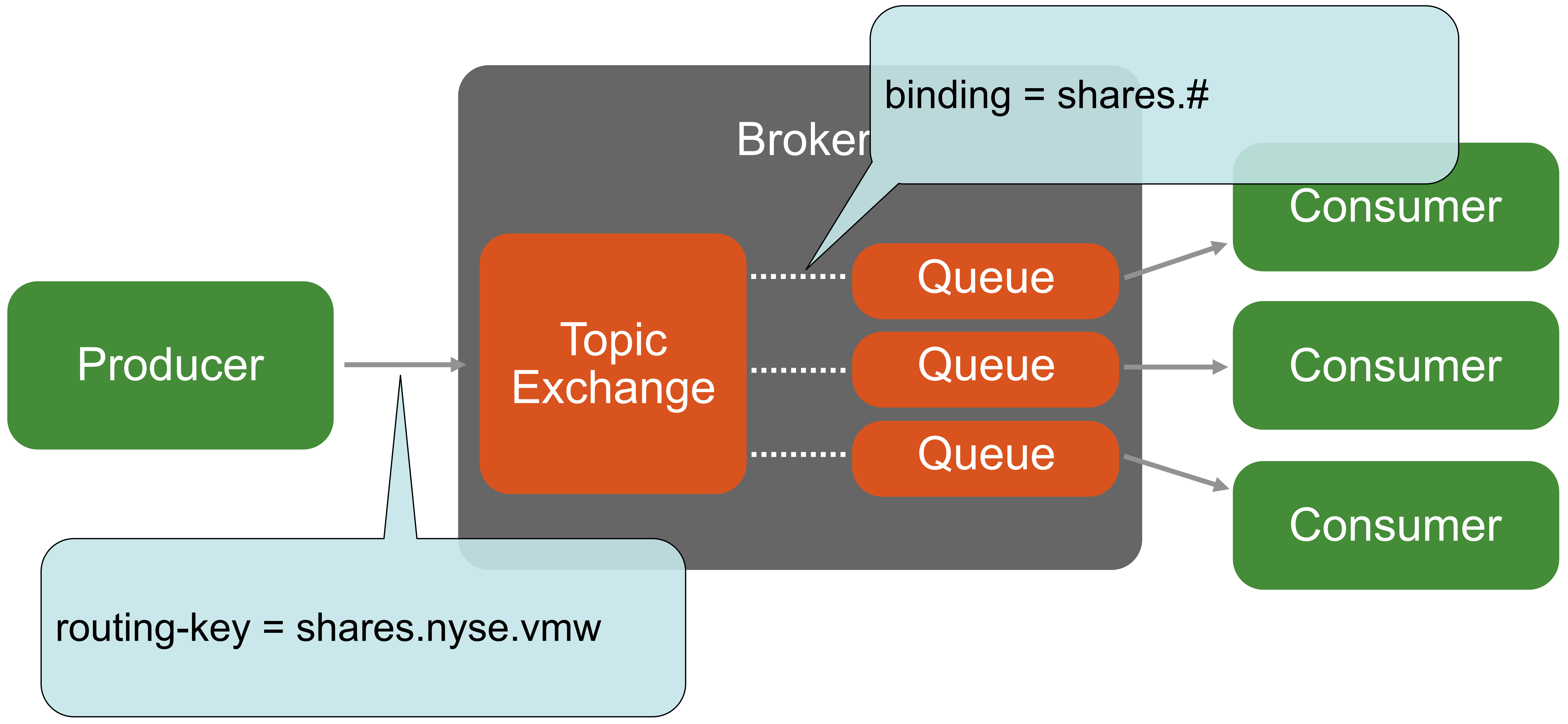
Example: broadcast with filtering



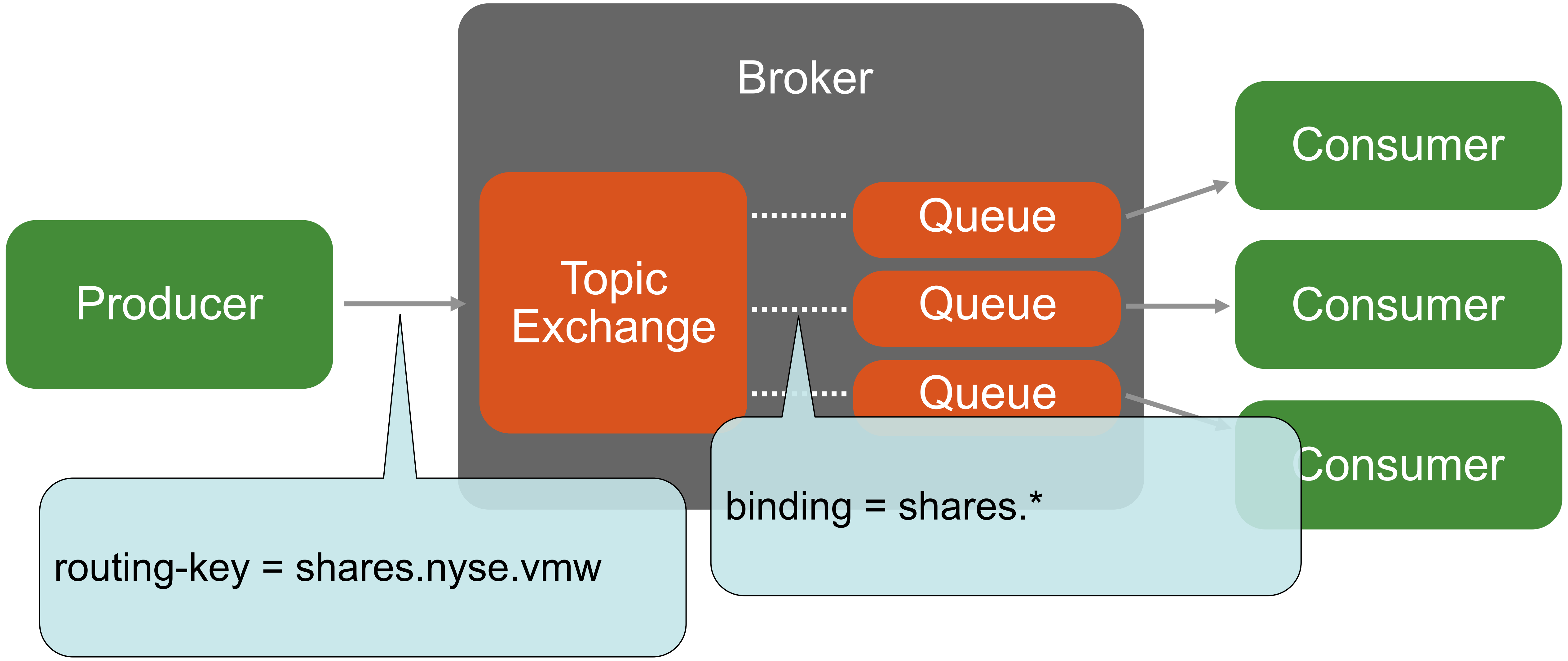
Example: broadcast with filtering



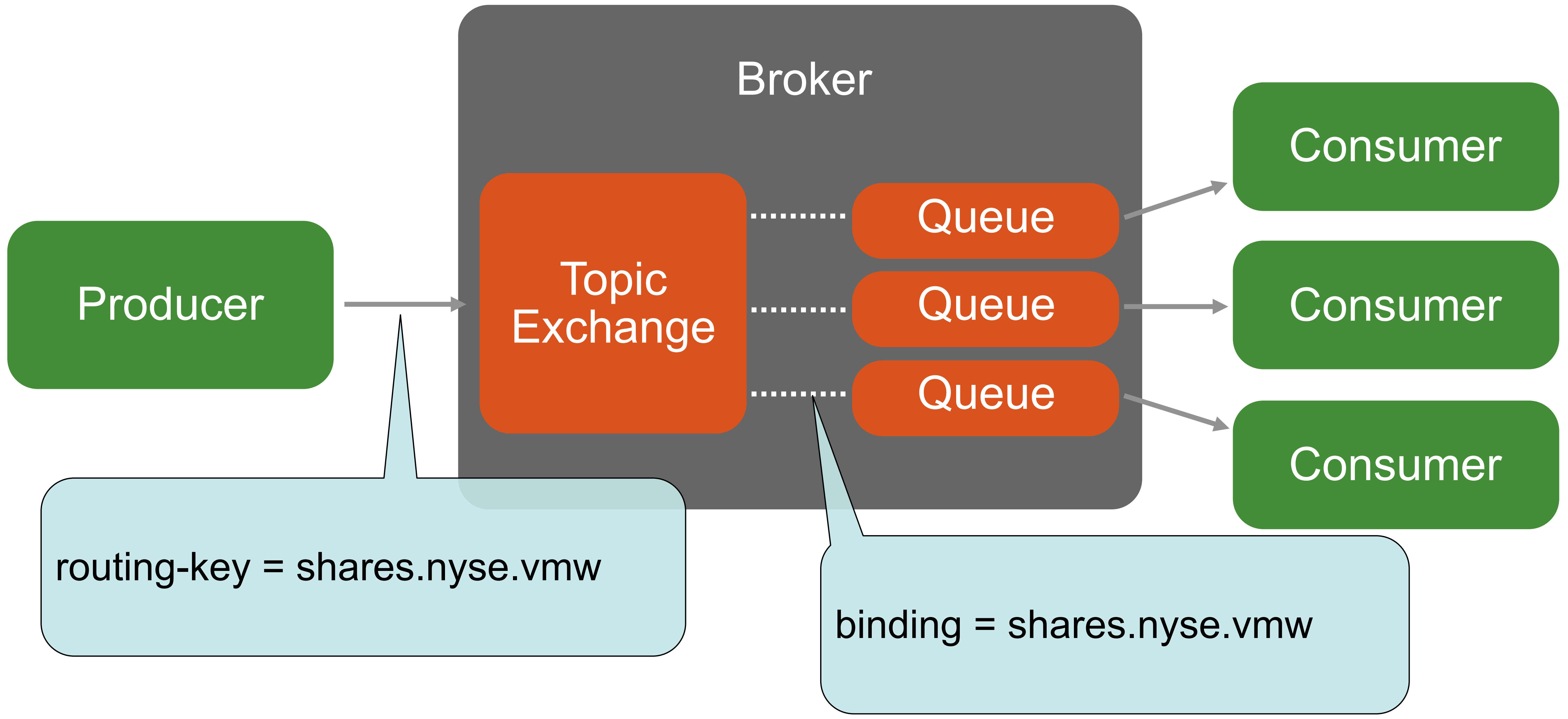
Example: broadcast with filtering



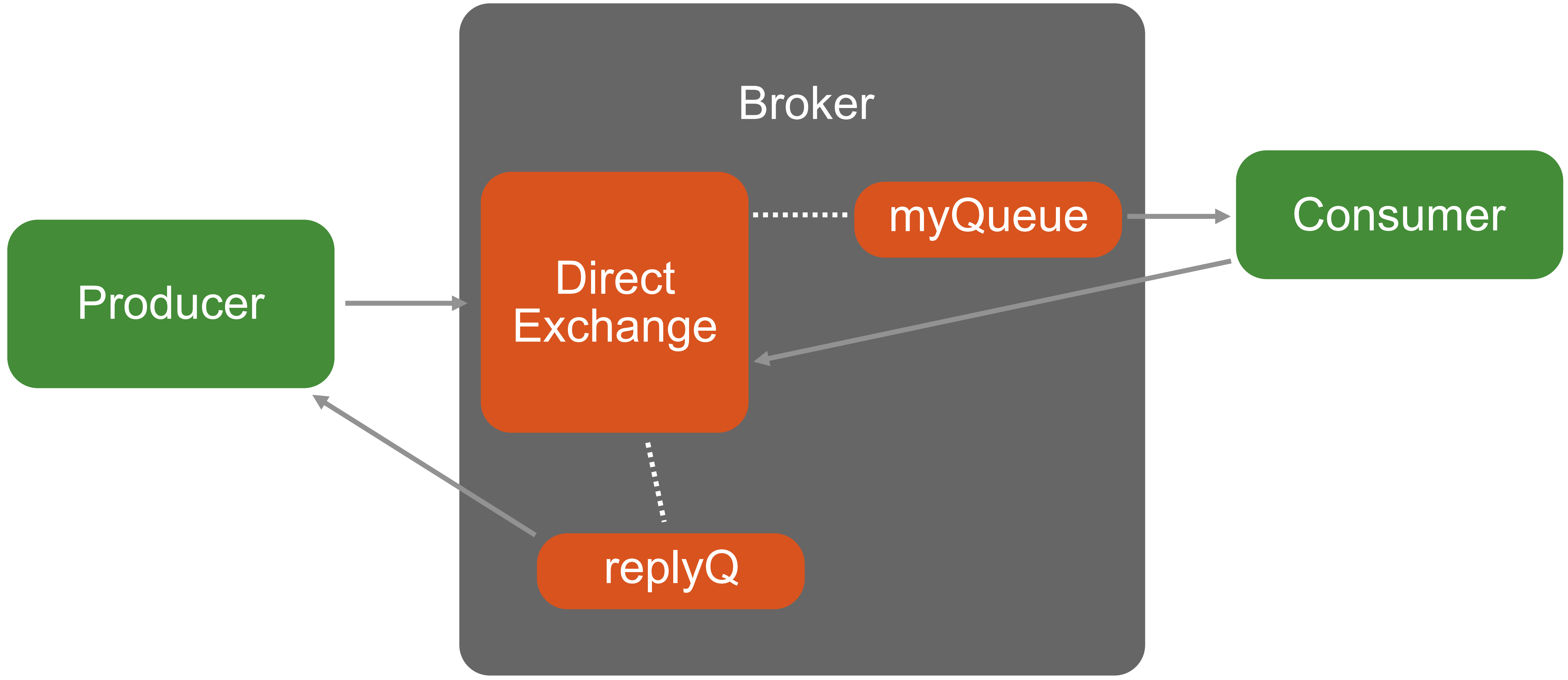
Example: broadcast with filtering



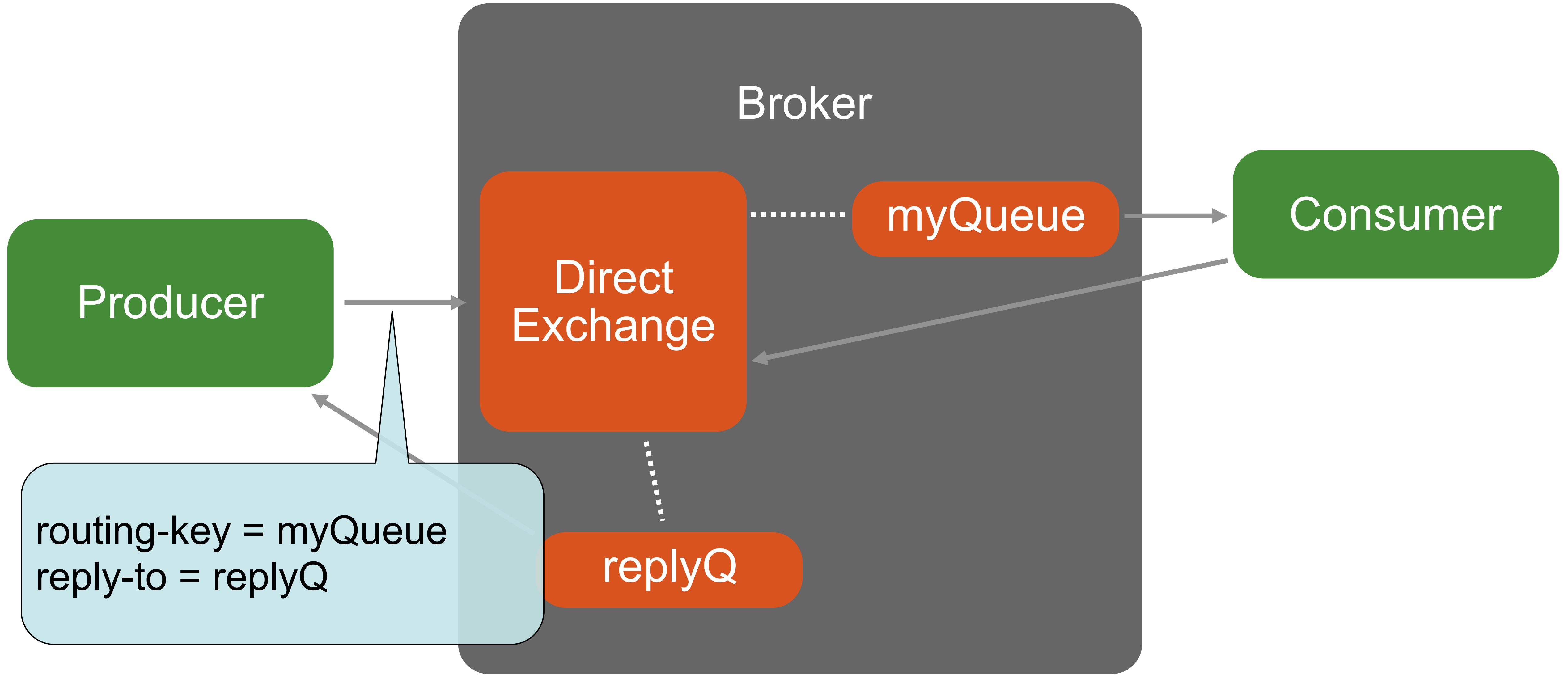
Example: broadcast with filtering



Example: RPC

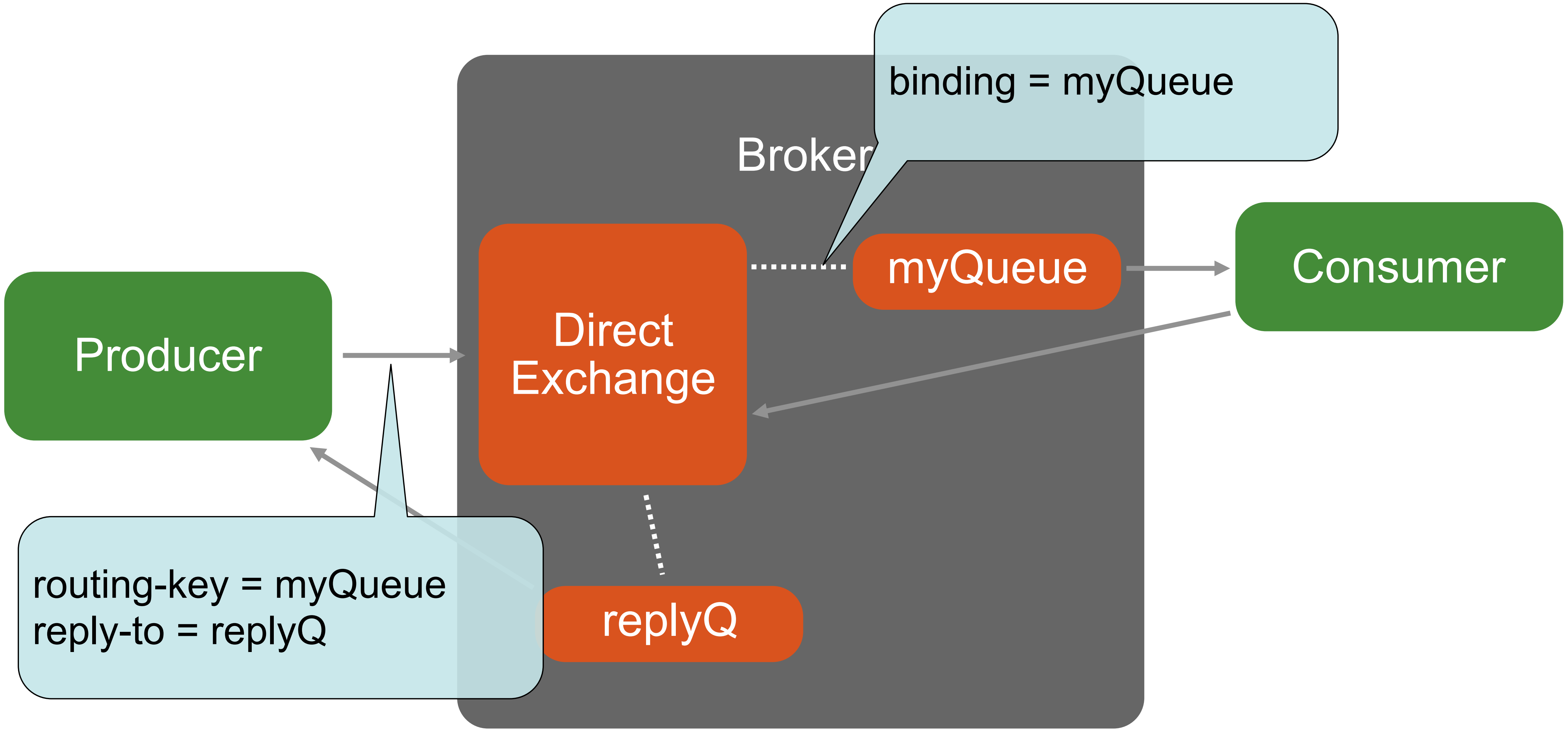


Example: RPC

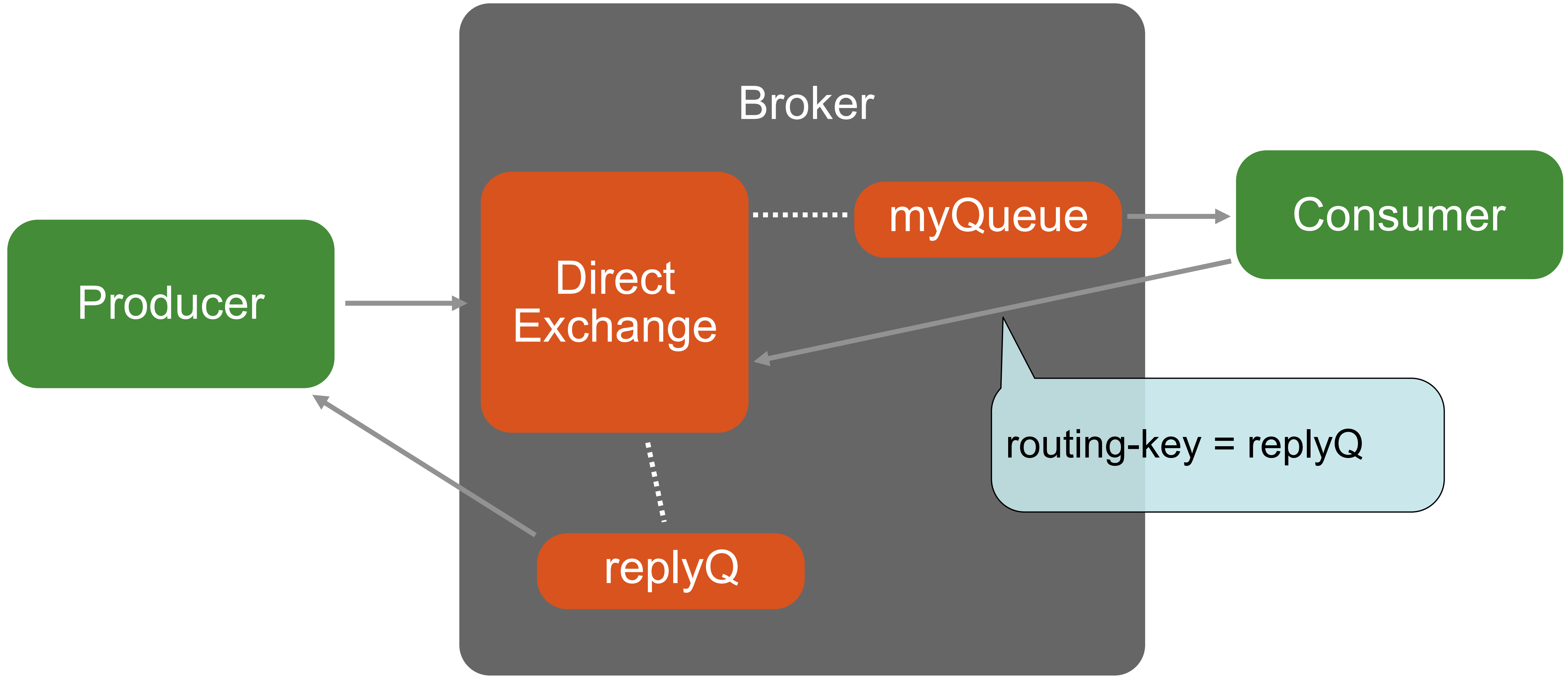


routing-key = myQueue
reply-to = replyQ

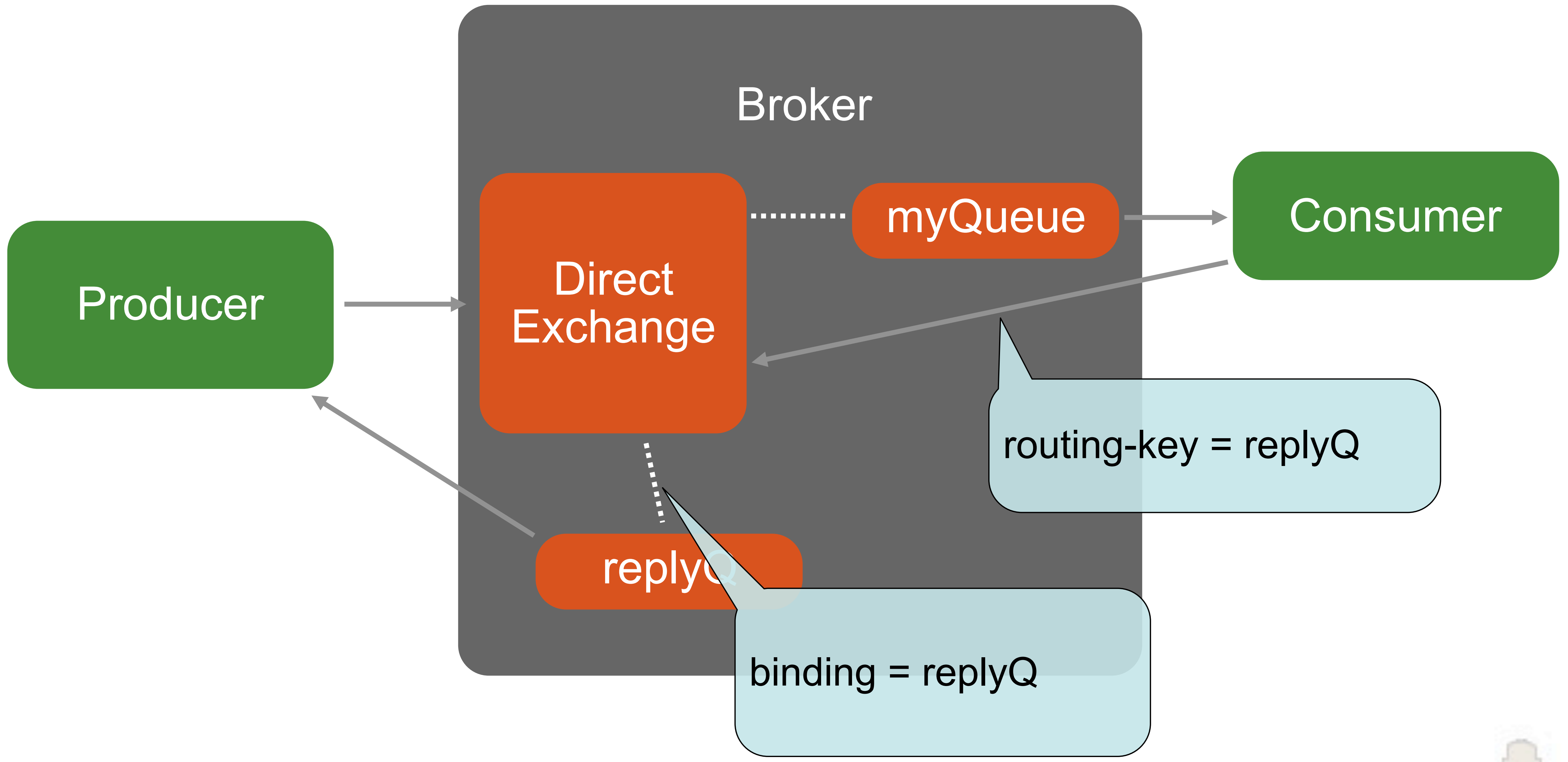
Example: RPC


binding = myQueue
Broker
myQueue
Consumer
Producer
Direct Exchange
routing-key = myQueue
reply-to = replyQ
replyQ

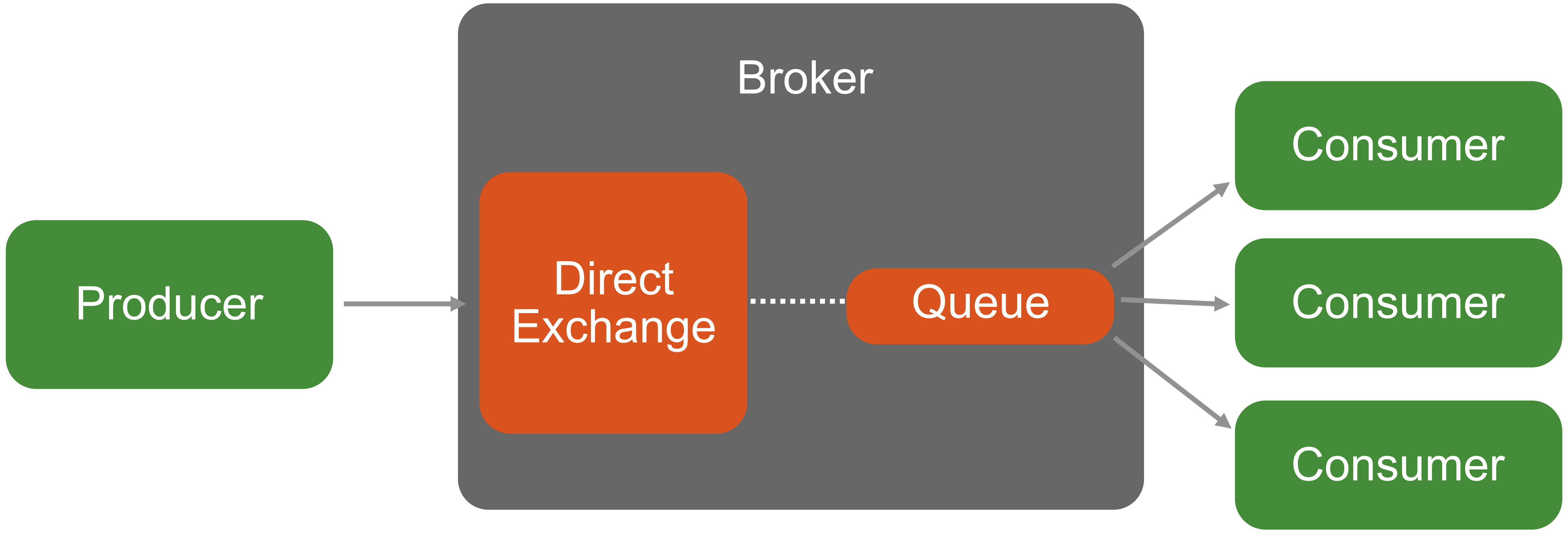
Example: RPC


Broker
Producer
Direct Exchange
myQueue
Consumer
routing-key = replyQ
replyQ

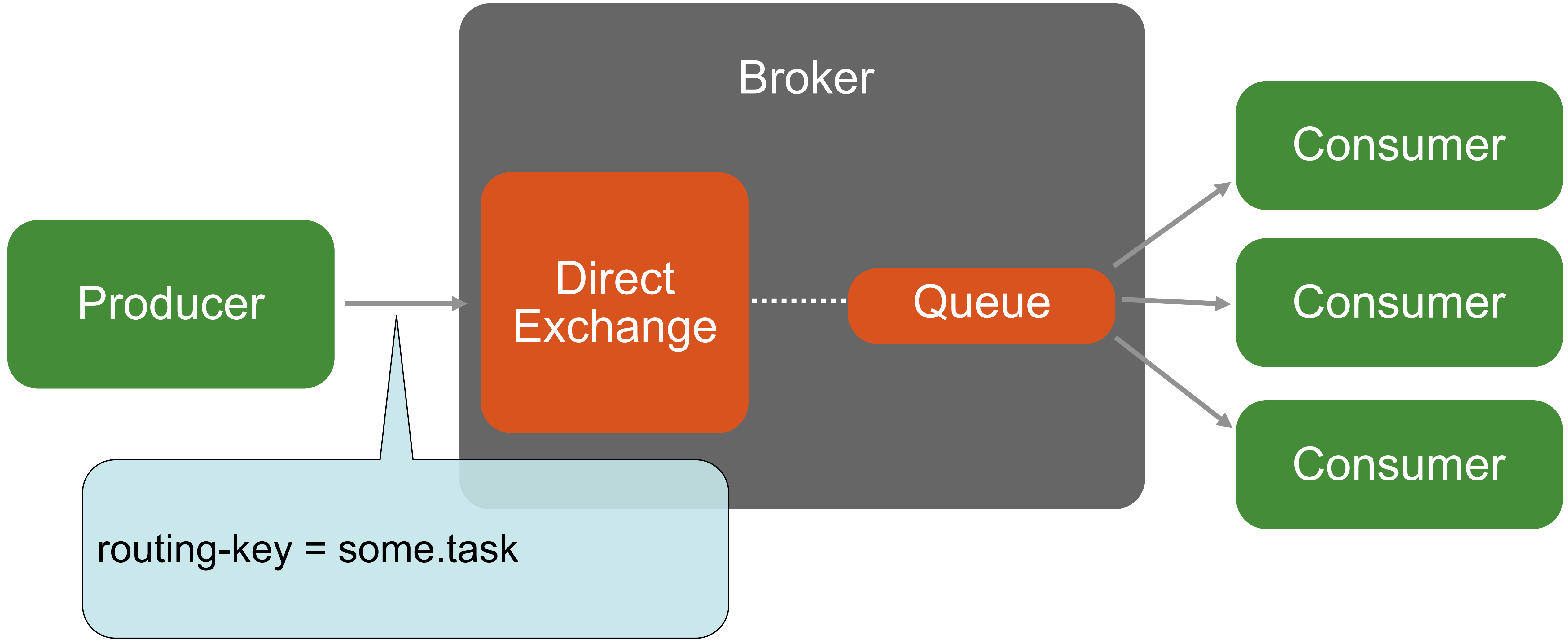
Example: RPC


Broker
Producer
Direct Exchange
myQueue
Consumer
routing-key = replyQ
replyQ
binding = replyQ

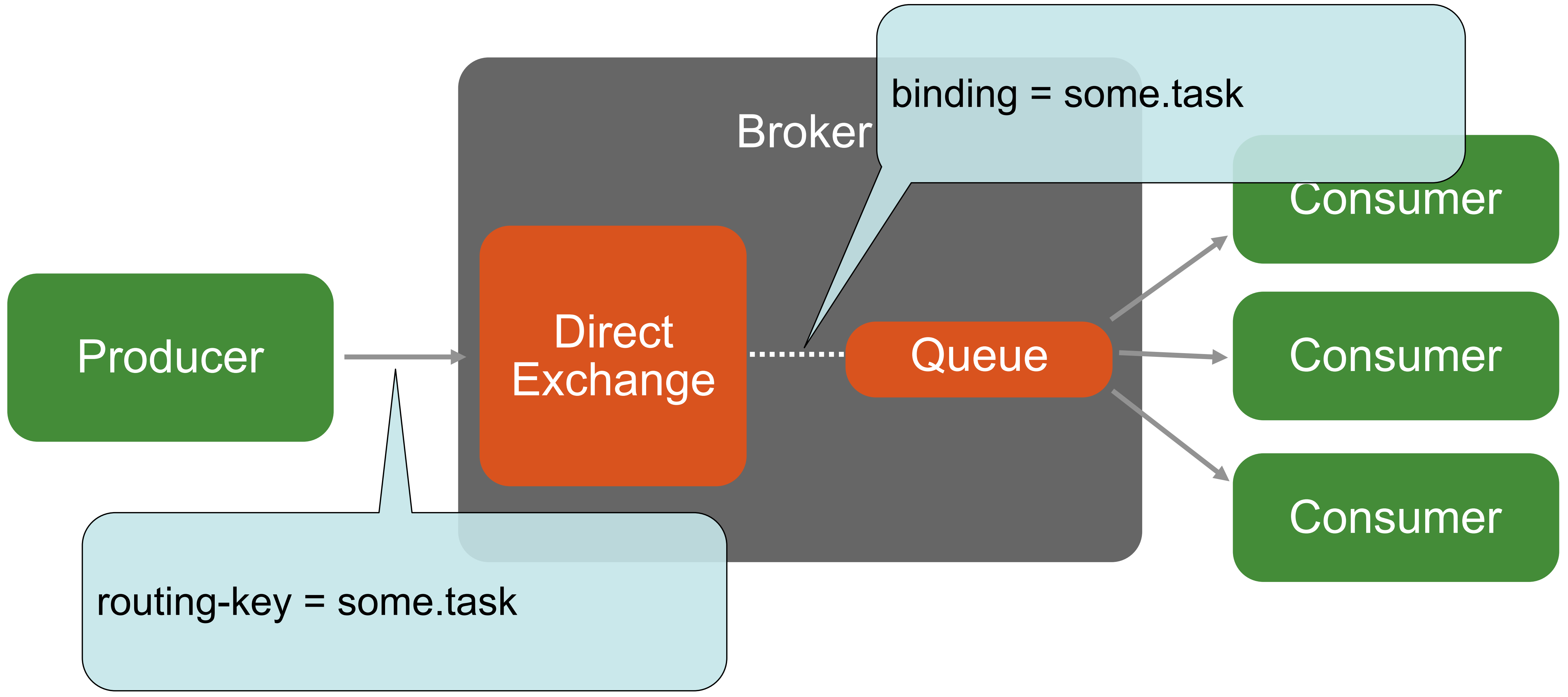
Example: work distribution



Example: work distribution



Example: work distribution



Messages

- Headers
  - routing-key
  - reply-to
  - content-type, etc.
- Custom properties
- Body
  - Byte data
  - Producer and consumer must agree on the format of the content
  - ... or use content-type header
  - AMQP does not define a meaning for content-type!

Queue and exchange properties

- Durable
  - Survives a broker restart
  - Applies to exchanges and queues
- Auto delete
  - Exchange will be deleted when all its bindings are gone
  - Queue will be deleted when all consumers are gone
- Exclusive
  - Only the owner can read messages from the queue
  - Doesn’t apply to exchanges

The Grails integration

- RabbitMQ plugin
- Declare exchanges and queues
- Configure services as queue consumers
  - Simple static properties
- Dynamic method for sending AMQP messages

Consuming messages

```
class ListenerService {
    // Declare name of queue to listen to
    static rabbitQueue = "msgs"

    void handleMessage(msg) {
        // Do something with the message
    }
}
```

```
class AnotherListenerService {
    // Subscribe to a topic exchange
    static rabbitSubscribe = "sharesExchange"

    void handleMessage(msg) {
        // Do something with the message
    }
}
```

Consuming messages

```
class ListenerService {
    // Declare name of queue to listen to
    static rabbitQueue = "msgs"

    void handleMessage(msg) {
        // Do something with the message
    }
}
```

```
class AnotherListenerService {
    // Subscribe to a topic exchange
    static rabbitSubscribe = [ name: "myEx", routingKey: "shares.#" ]

    void handleMessage(msg) {
        // Do something with the message
    }
}
```

Sending messages

```
class PublisherService {

    def notify() {
        rabbitSend "msgs", "app.event", "The event details"
    }
}
```

Sending messages

```
class PublisherService {

    def notify() {
        rabbitSend "msgs", "app.event", "The event details"
    }
}
```

Routing key

Exchange name (optional)

Message body

Sending messages

```
class PublisherService {

    def notify(String itemName) {
        rabbitSend "msgs", "app.event", [event: "publish", item: itemName ]
    }
}
```

Declaring exchanges and queues

```
// Config.groovy
rabbitmq {
    connectionfactory {
        ...
    }

    queues = {
        msgs durable: false, autoDelete: true

        exchange name: "shares", type: topic, durable: true, {
            allShares durable: true, autoDelete: false, binding: 'shares.#'
        }
    }
}
```

Declaring exchanges and queues

```
// Config.groovy
rabbitmq {
    conne
        ...
    }

    queue  = {
        msgs durable: false, autoDelete: true

        exchange name: "shares", type: topic, durable: true, {
            allShares durable: true, autoDelete: false, binding: 'shares.#'
        }
    }
}
```

Standalone queue (msgs) - bound to default direct exchange

Topic exchange (shares)

Queue (allShares) bound to exchange (shares) with routing key ('shares.#')

A word about message content

- In the broker, it's just byte data
- Plugin interprets data based on content-type header
  - Spring AMQP SimpleMessageConverter
  - String → text/plain; charset=utf-8
  - Serializable → application/x-java-serialized-object
  - Otherwise, just byte[]
- Producers & consumers typically agree on format
- Not all clients set the content-type!
- You still have to agree on format even if you use JSON or XML message content

Demo

Chicago, October 19 - 22, 2010

Q&A