Decouple all the things

Asynchronous messaging keeps it simple

Kerstin Puschke
@titanoboa42

Codemotion Berlin 2016

XING

Machine to Machine Communication

418 I’m a teapot

Improving maintainablity and stability with

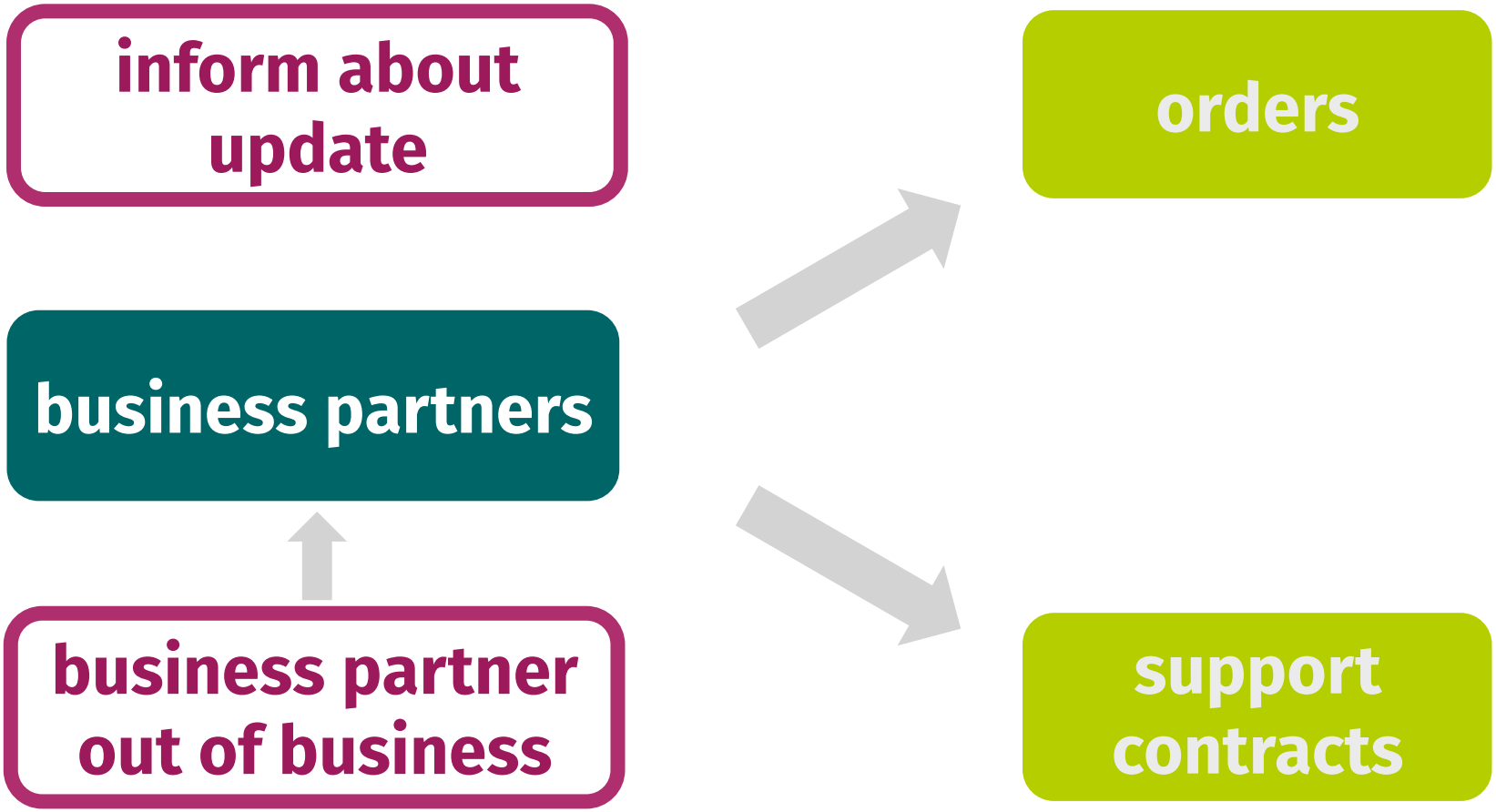
asynchronous communication

Index

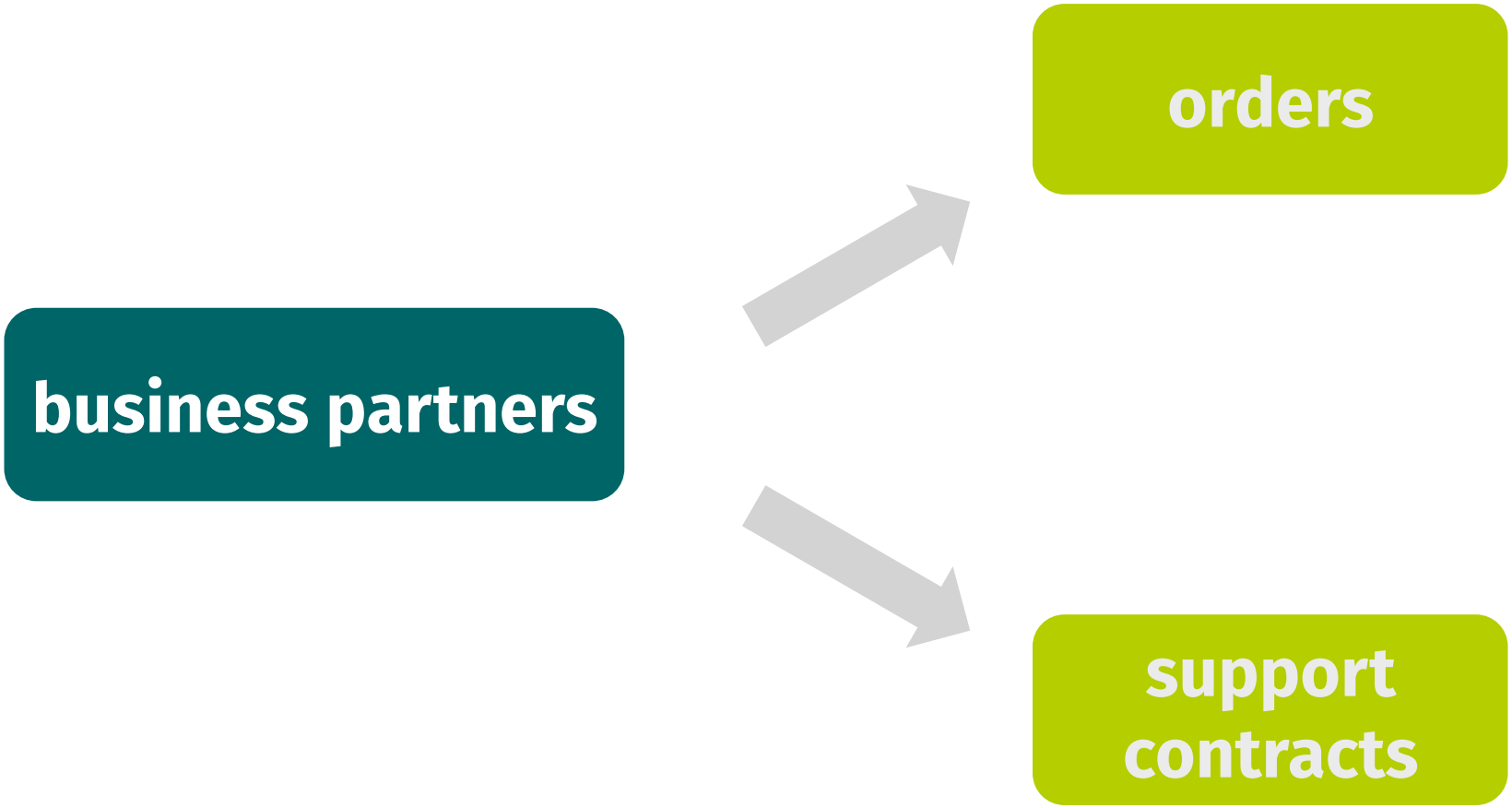
Synchronous Communication

and its disadvantages

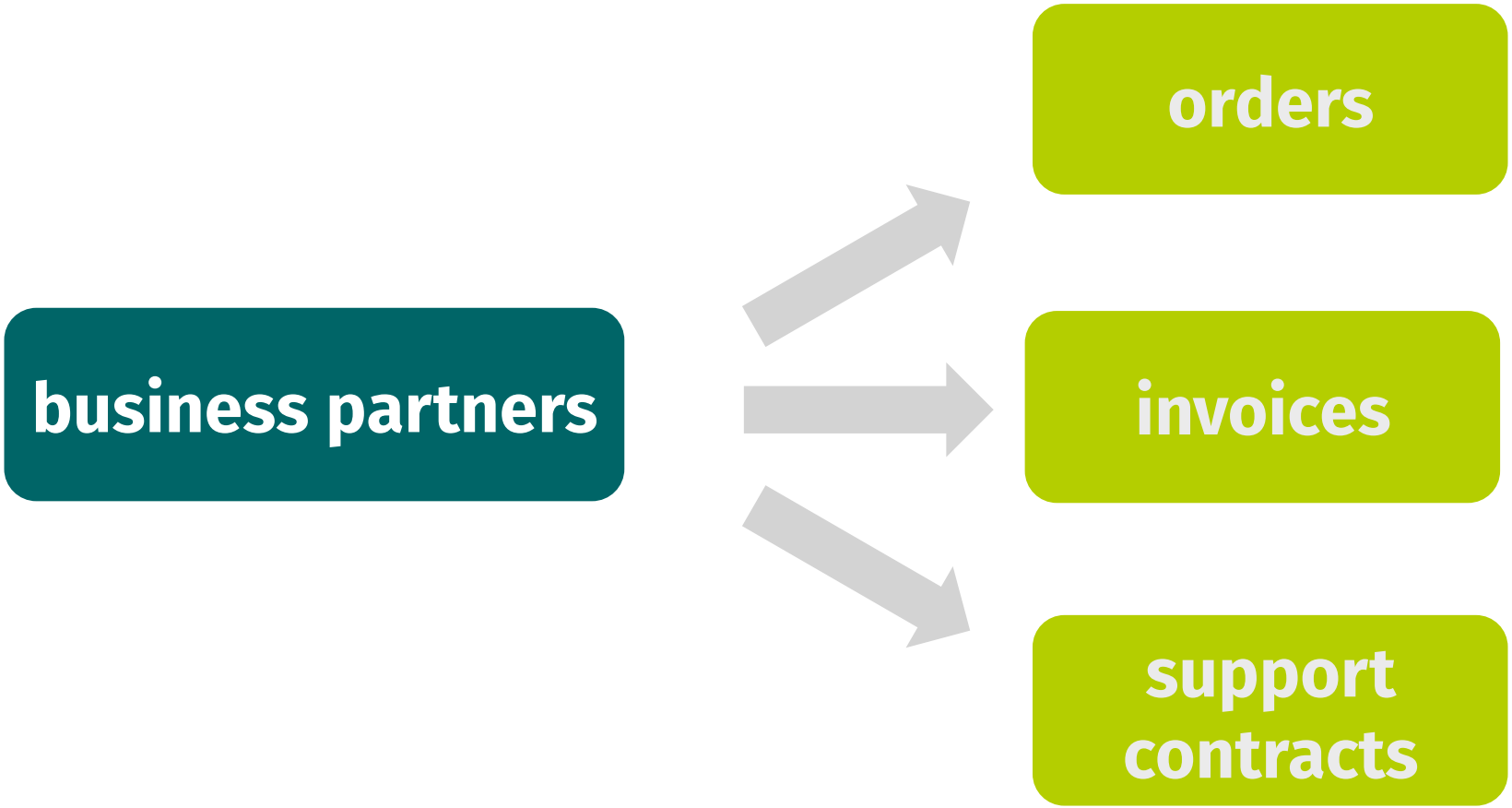
Updates need to propagate from the owner of the data to its clients



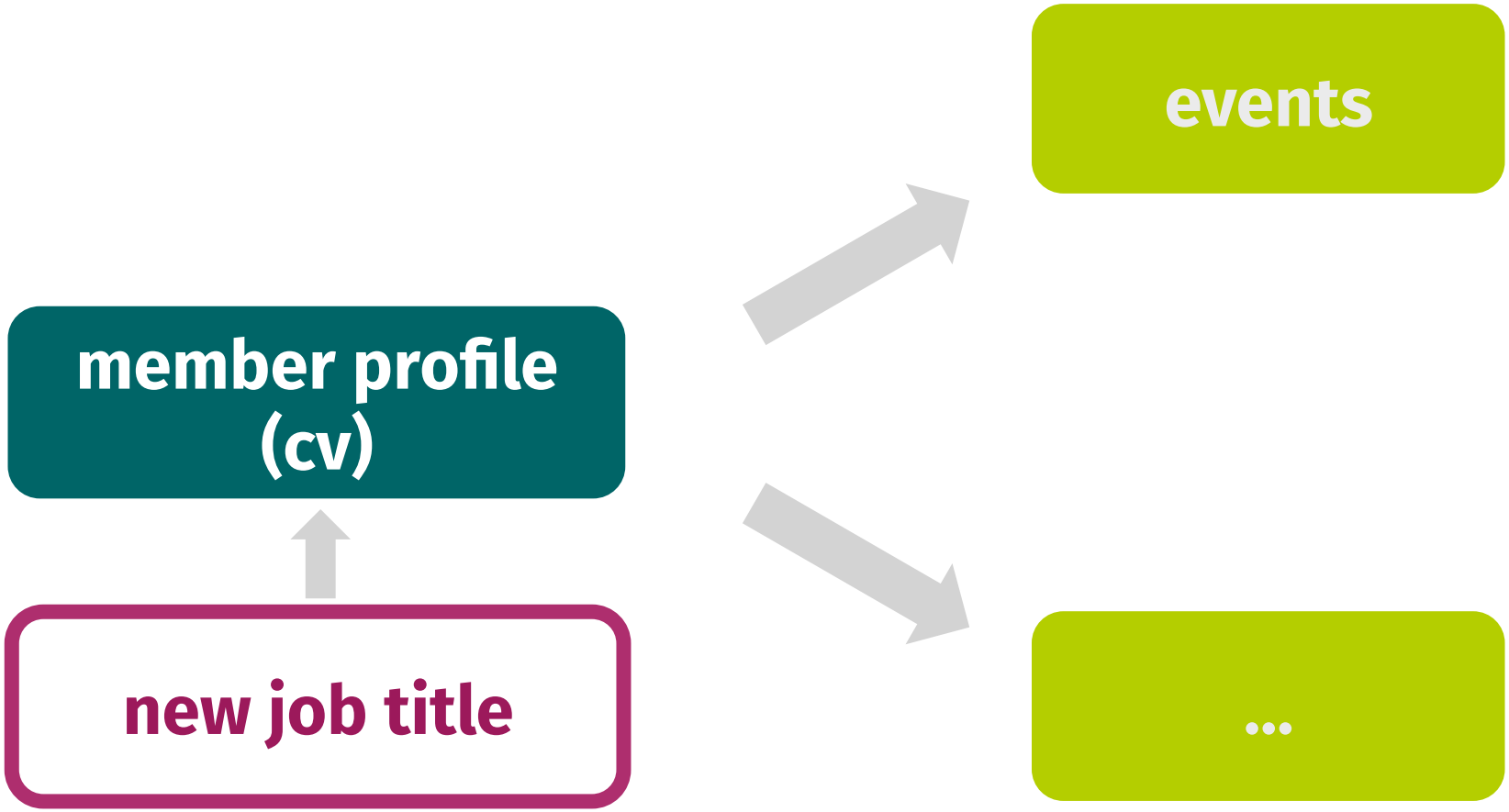
downside of REST:
client misses update if temporarily down



downside of REST:
app needs to know its clients



Updates need to propagate
Example from xing.com



Asynchronous Messaging

using the Advanced Message Queuing Protocol

Advanced Message Queuing Protocol

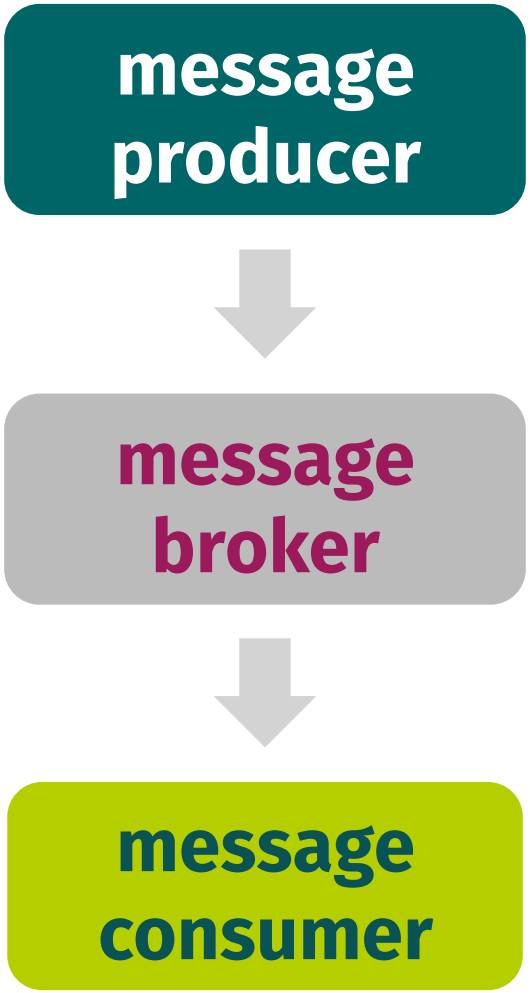
“The capable, commoditized, multi-vendor communications ecosystem which AMQP enables creates opportunities for commerce and innovation which can transform the way business is done on the Internet, and in the cloud.”

amqp.org/about/what

“open standard application layer protocol for message-oriented middleware”

https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol

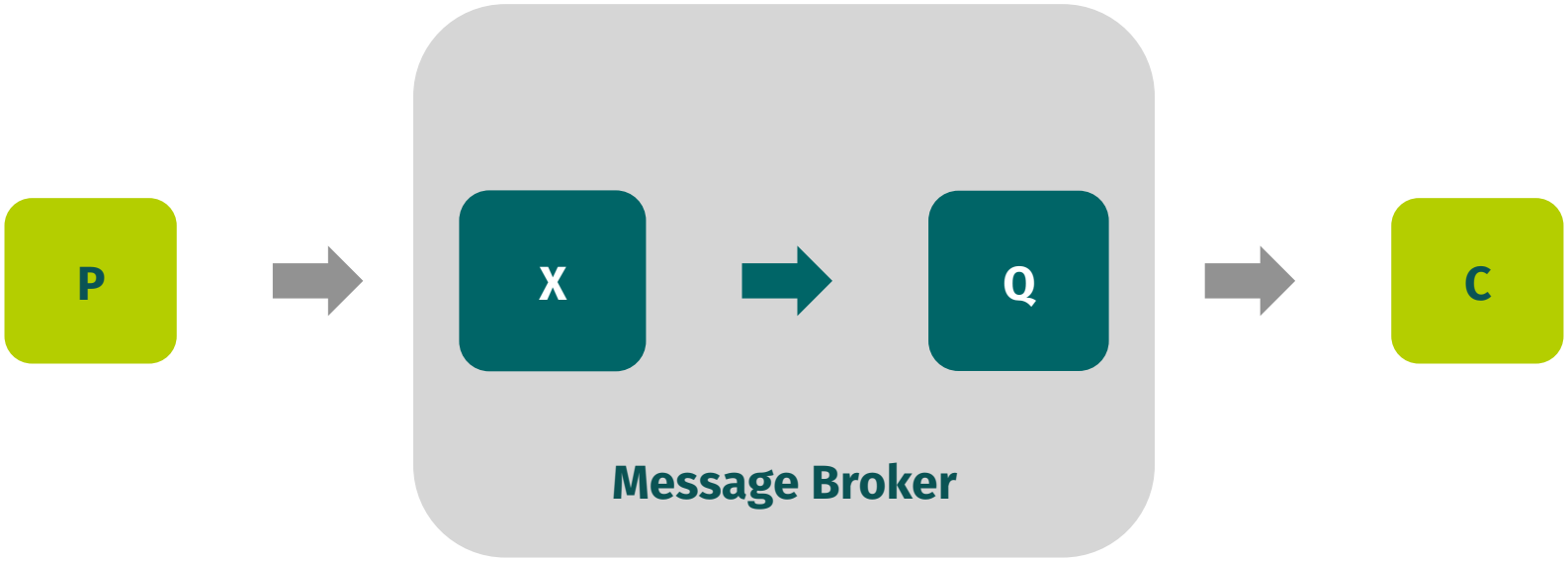
Interoperability



Messages contain payload and routing key

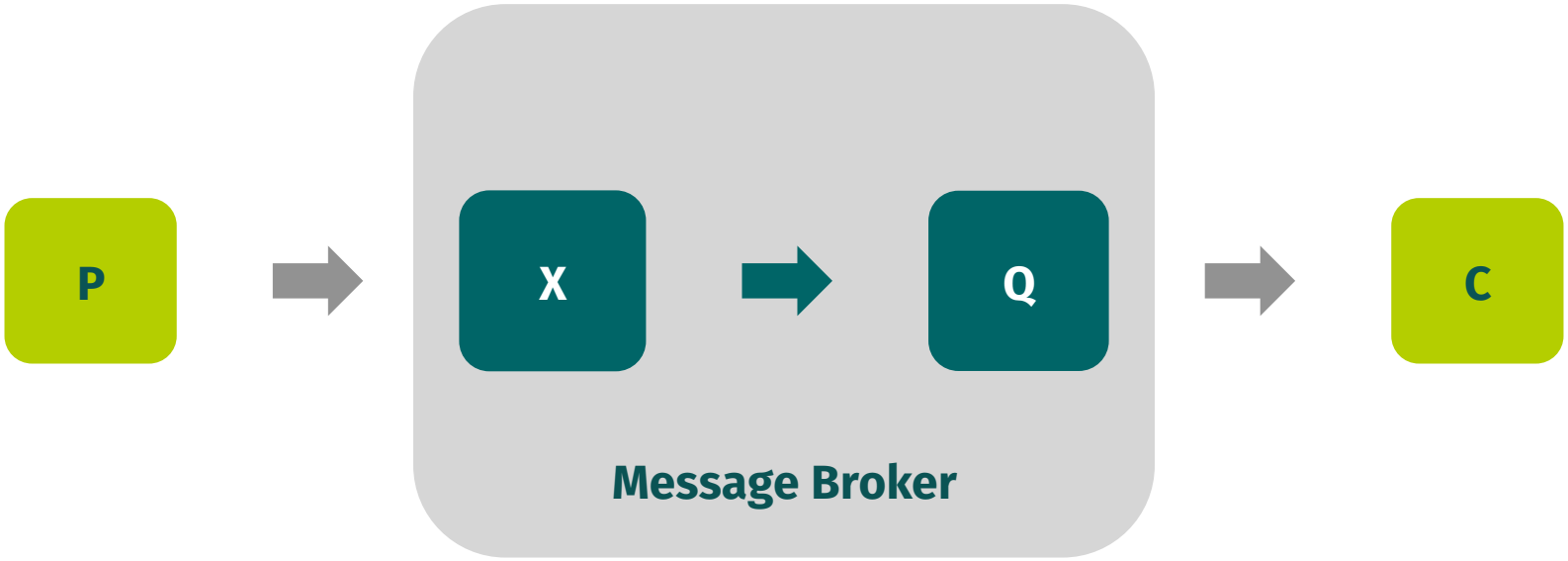
- **Routing Key**
- **Payload**
  - application data
  - any form, any encoding...
- **...and more**
  - Structured app-specific data (properties / attributes)
  - Headers, Annotations
  - ...

Message Flow



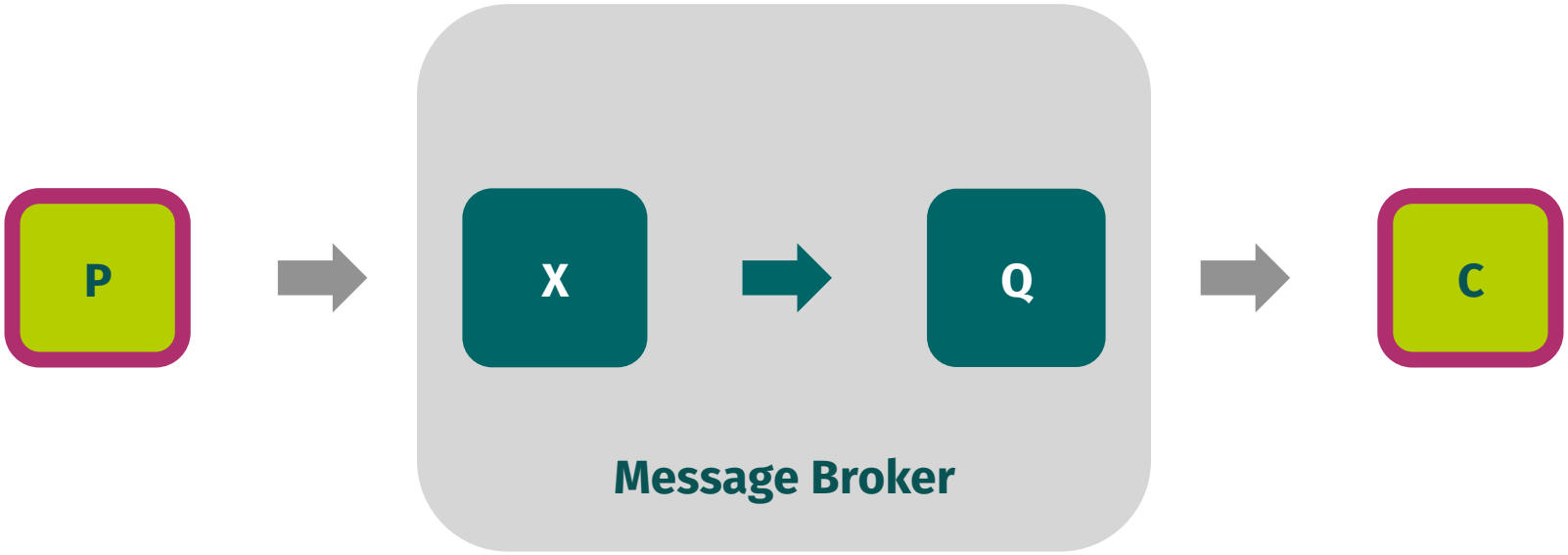
Producer **Exchange** **Queue** **Consumer**

No information lost if consumer down



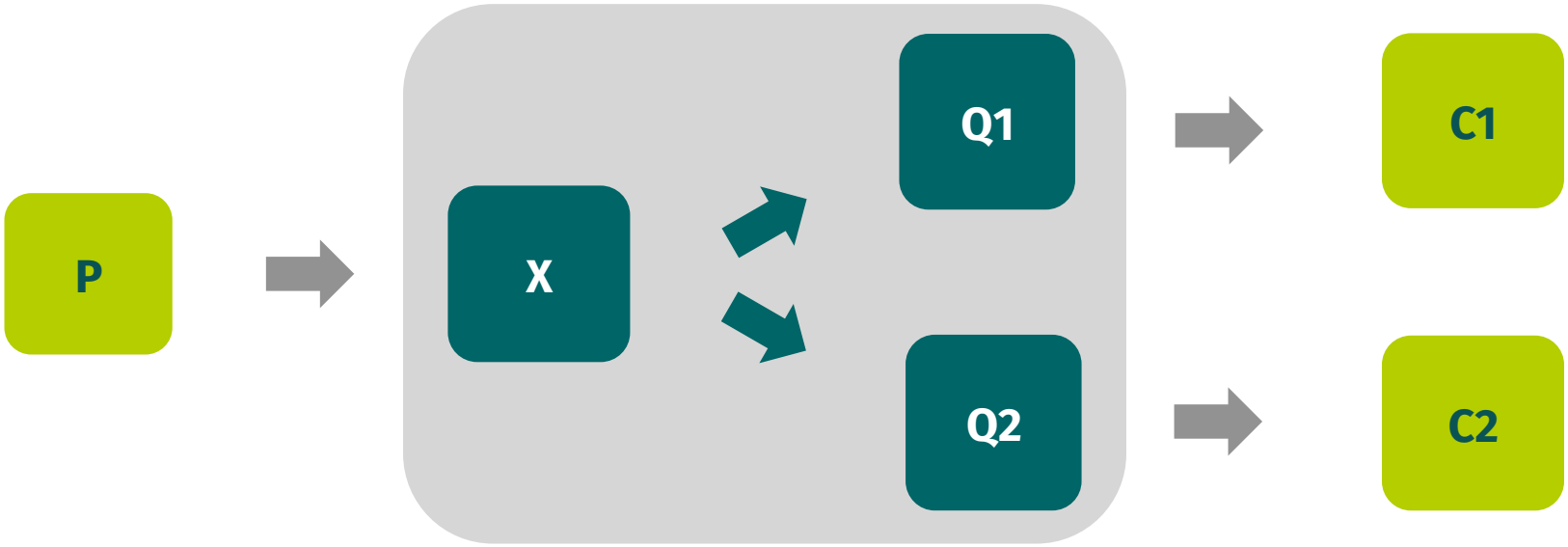
Producer **Exchange** **Queue** **Consumer**

Applications are in power



Producer **Exchange** **Queue** **Consumer**

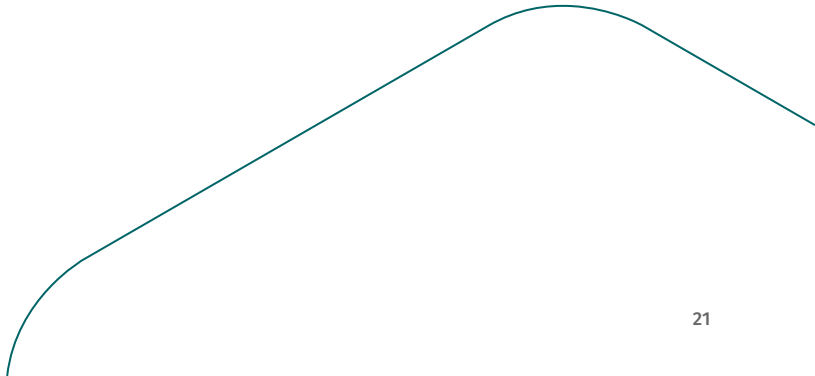
Producer does not have to know consumers



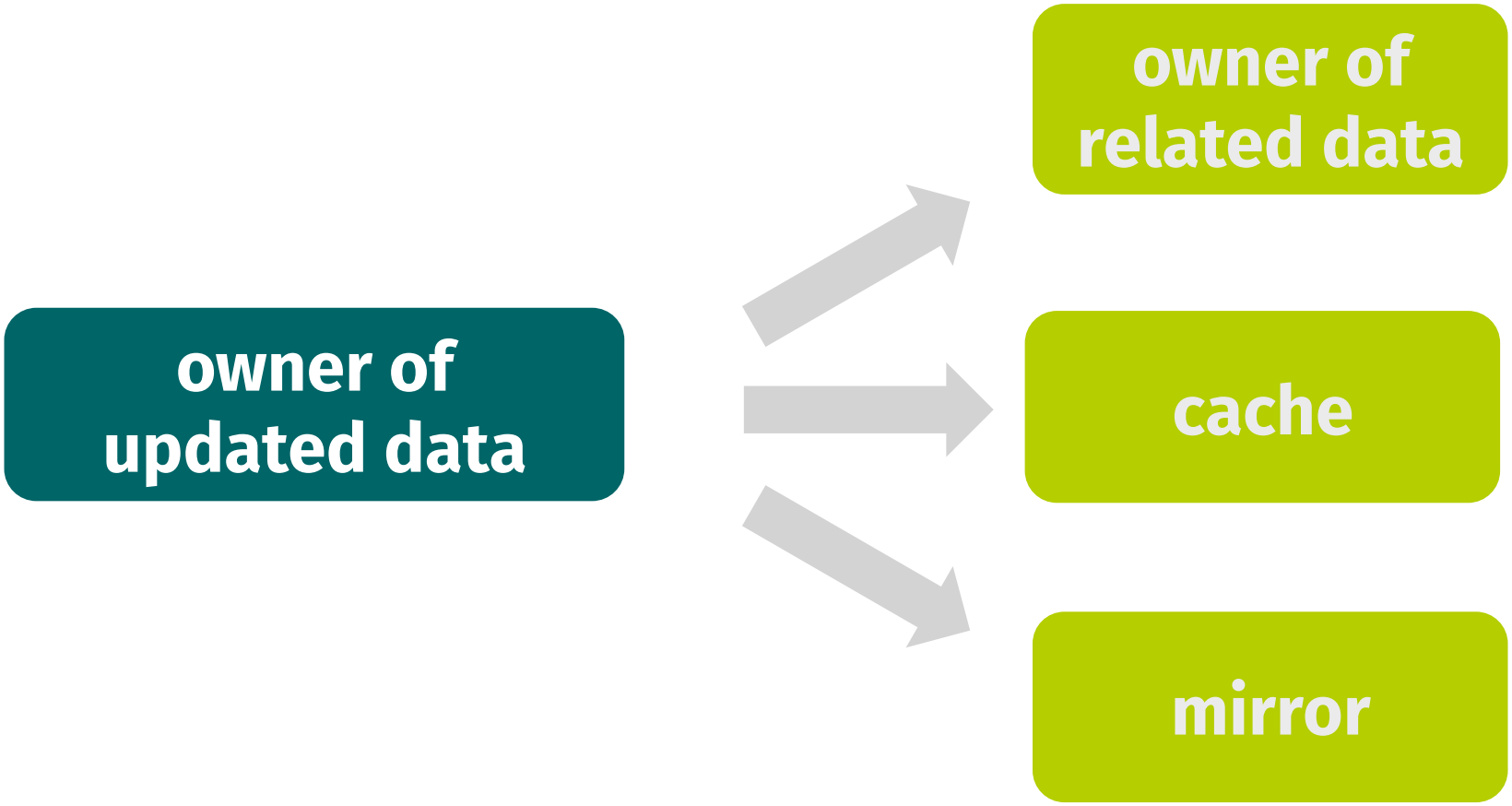
Producer **Exchange** **Queue** **Consumer**

Publish Subscribe

using the Advanced Message Queuing Protocol



Propagating updates
Clients have different needs – use different messages



Example message

Routing key

<...>.user.deleted

Payload

```
{
  “user_id” : 42
}
```

Example message

Routing key

<...>.profile.updated

Payload

```
{
  “user_id” : 42,
  “fields” : [ “country”, “city” ]
}
```

Flexible routing via different exchange types

- **Direct Exchange**
  - routes to all queues whose binding_key equals routing_key
- **Topic Exchange**
  - routing on multiple attributes
  - dot-separated words as routing keys, plus wildcards
- **Header Exchange**
  - routing on multiple attributes based on headers

Flexible routing via different exchange types

- **Fanout Exchange**
  - broadcasts to every queue it has a binding for
  - ignores routing key and queue name

Don’t include updated data

Routing key

<...>.profile.updated

Payload

```
{
  “user_id” : 42,
  “fields” : [ “country”, “city” ]
}
```

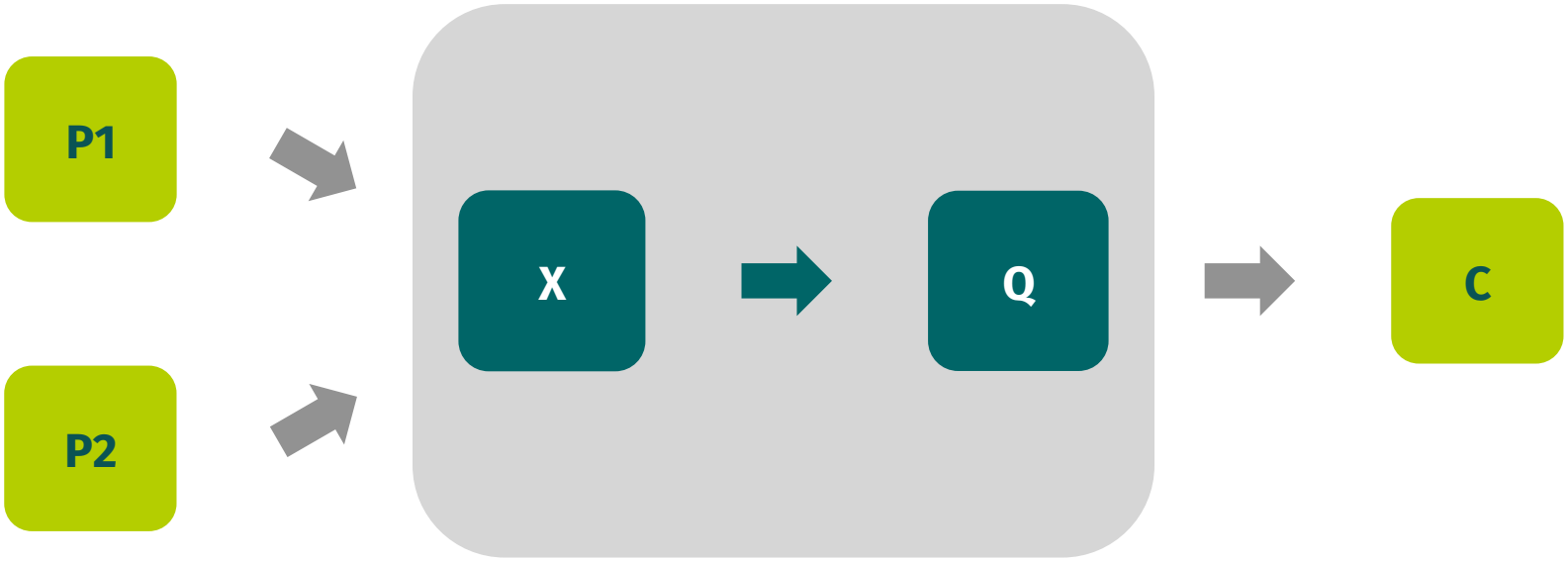
No guaranteed order – aim for commutativity

- Order of messages not guarantueed
- Commutativity
- Complement with synchronous API (e.g. REST)
  - most recent data
  - leading system
  - message consumers follow up with REST if necessary

No exactly-once delivery
aim for idem-potent messages

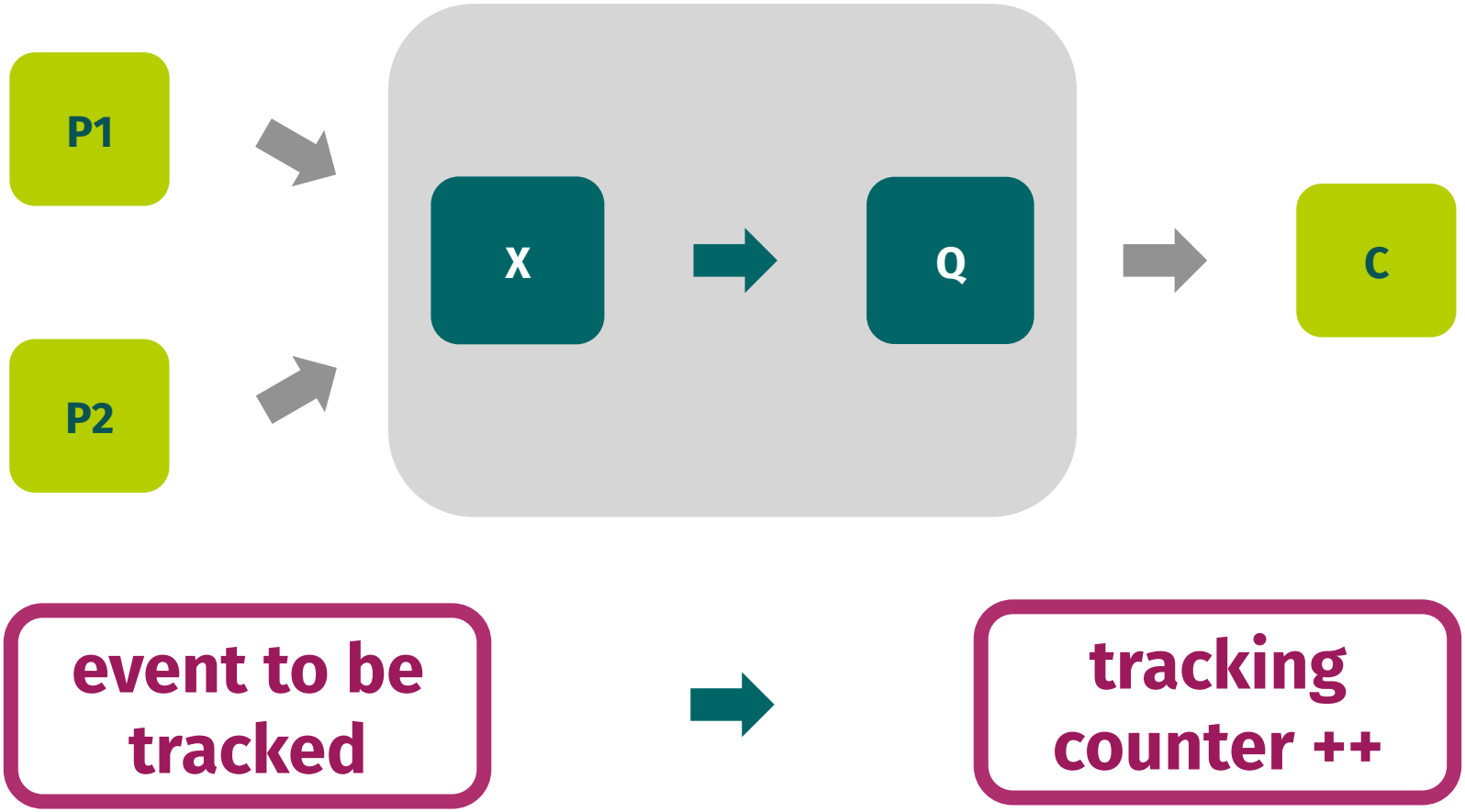
- **Duplicated messages**
  - possible e.g. if producer re-sends message after connection failure
  - use idem-potent message handling

Consumer does not have to know producers

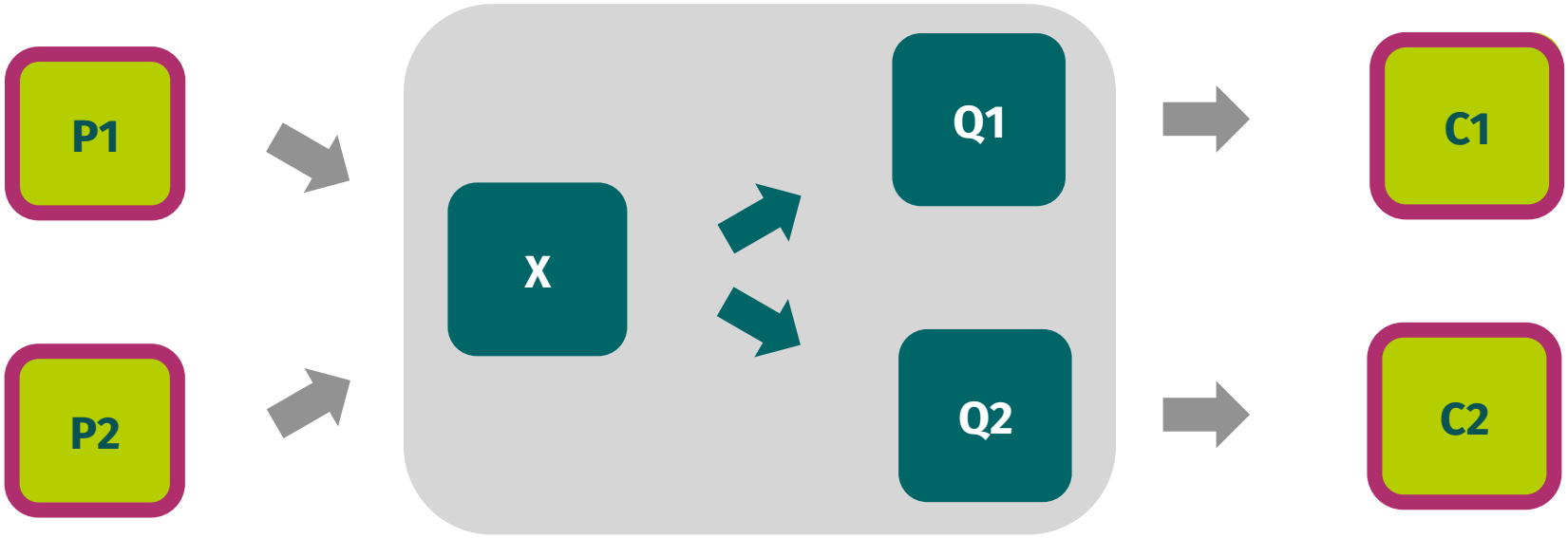


Producer **Exchange** **Queue** **Consumer**

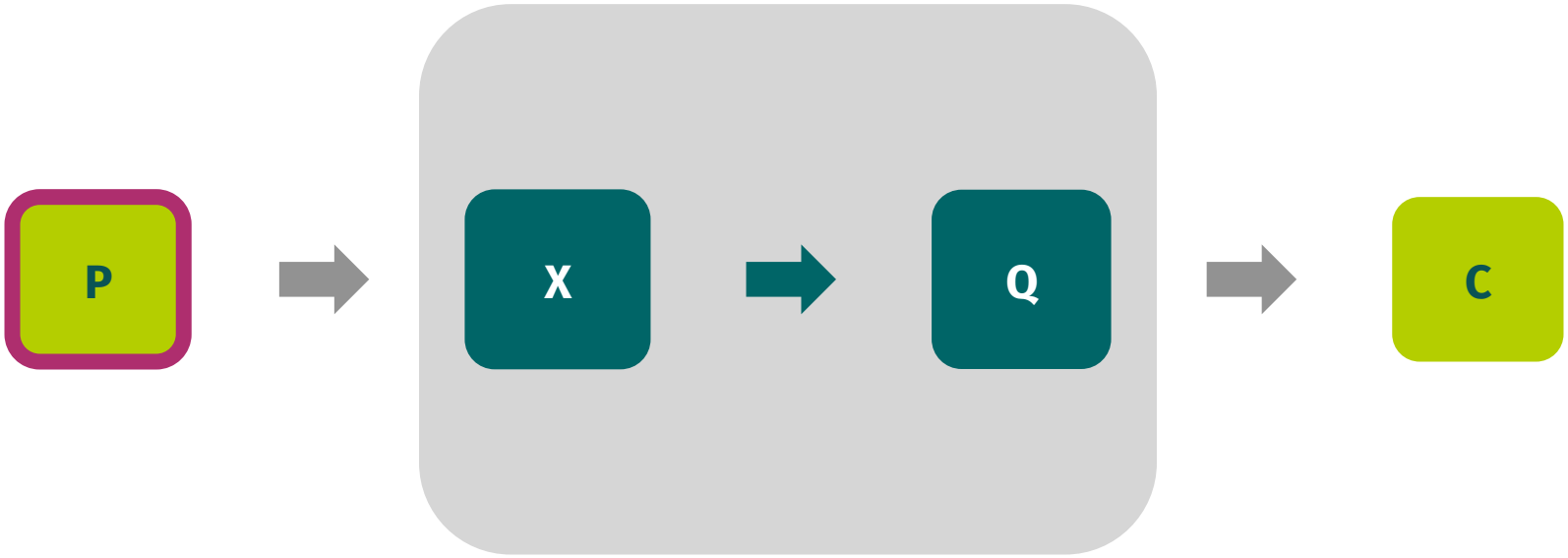
Centralized tracking of decentralized events



Avoid n-m routing
Keep breaking changes manageable

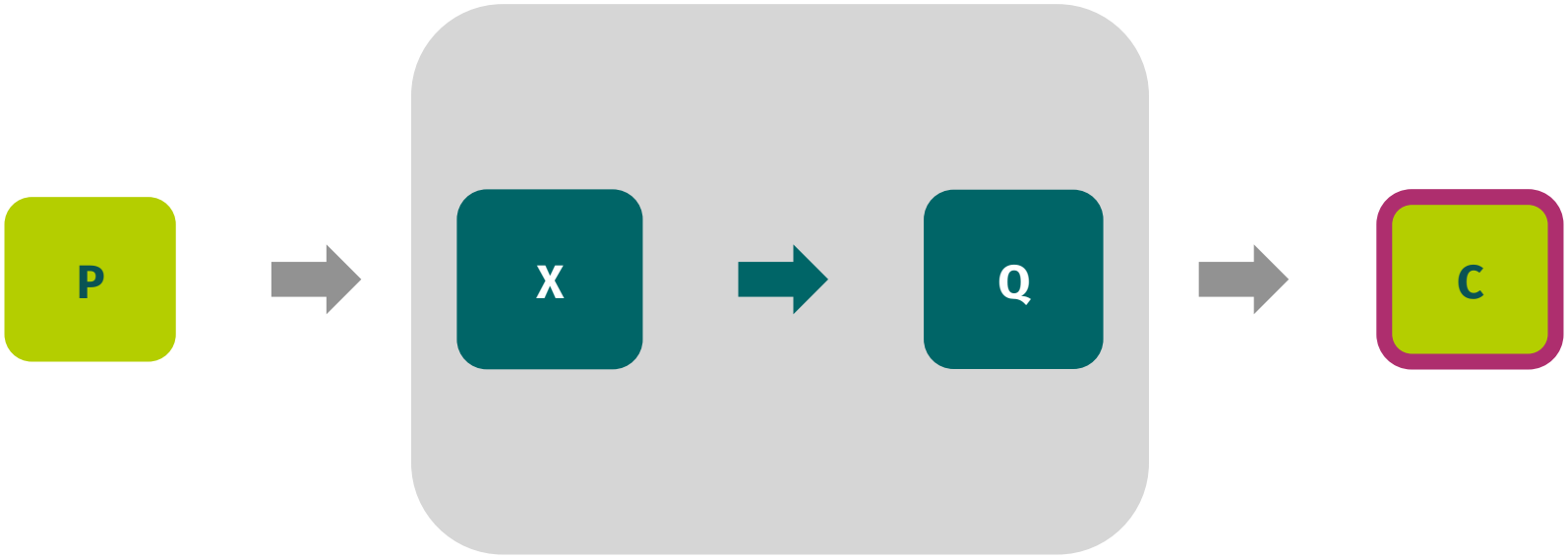


Notifications / Events
producer state change



event
state changed

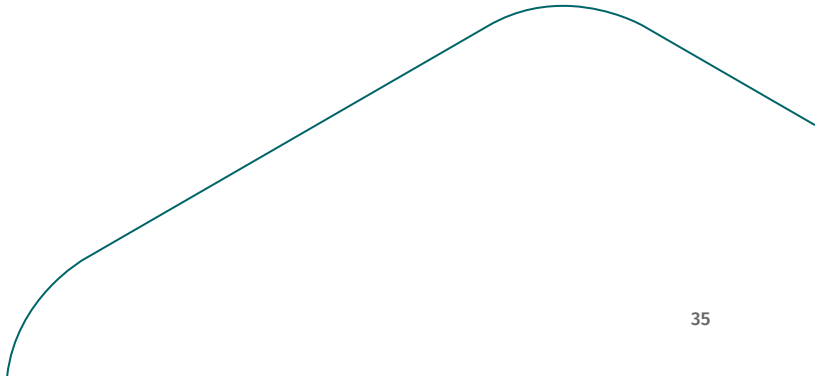
Task / Command
consumer state to change



event
state change

Task Queue

using the Advanced Message Queuing Protocol



Run tasks asynchronously

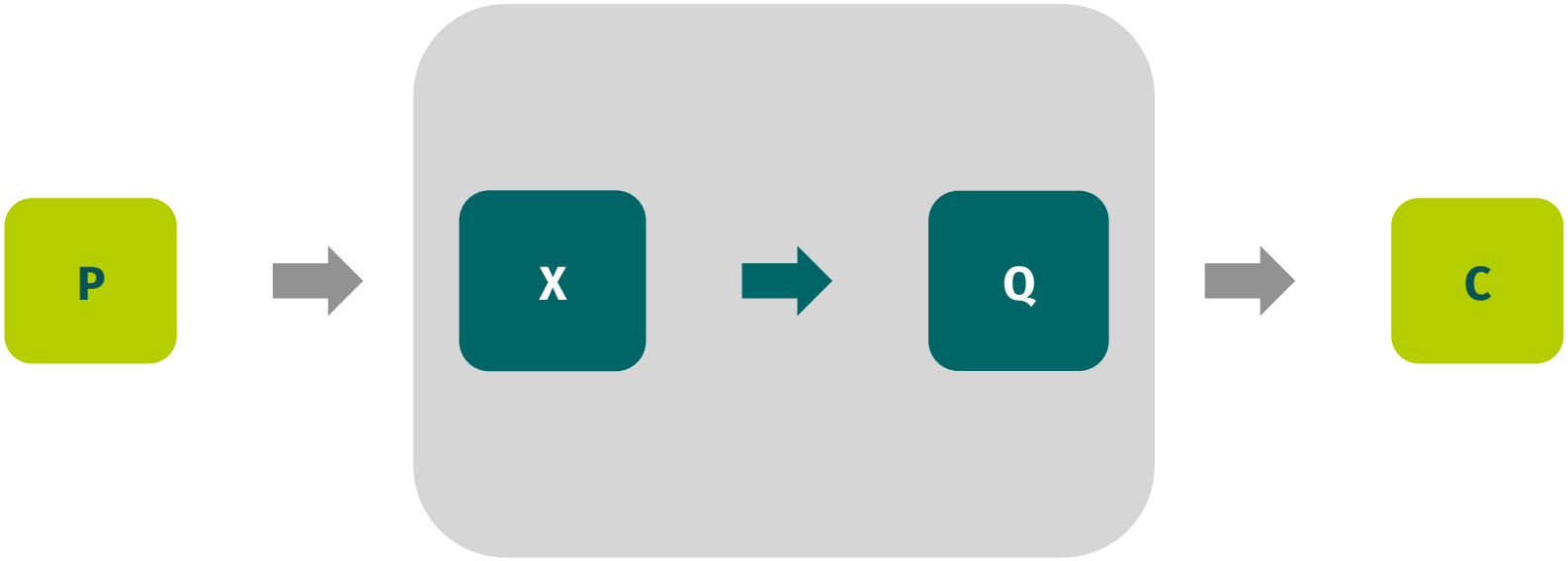


image upload user facing

image processing

Improve response time
Avoid downtimes

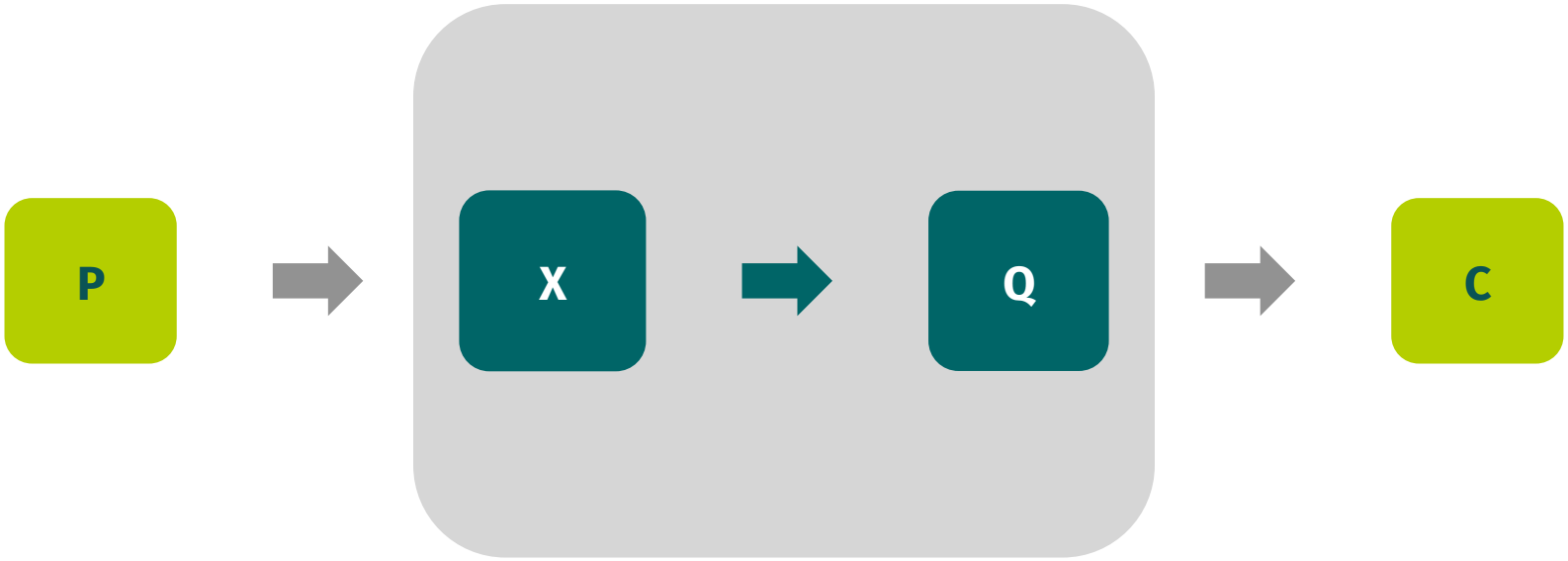
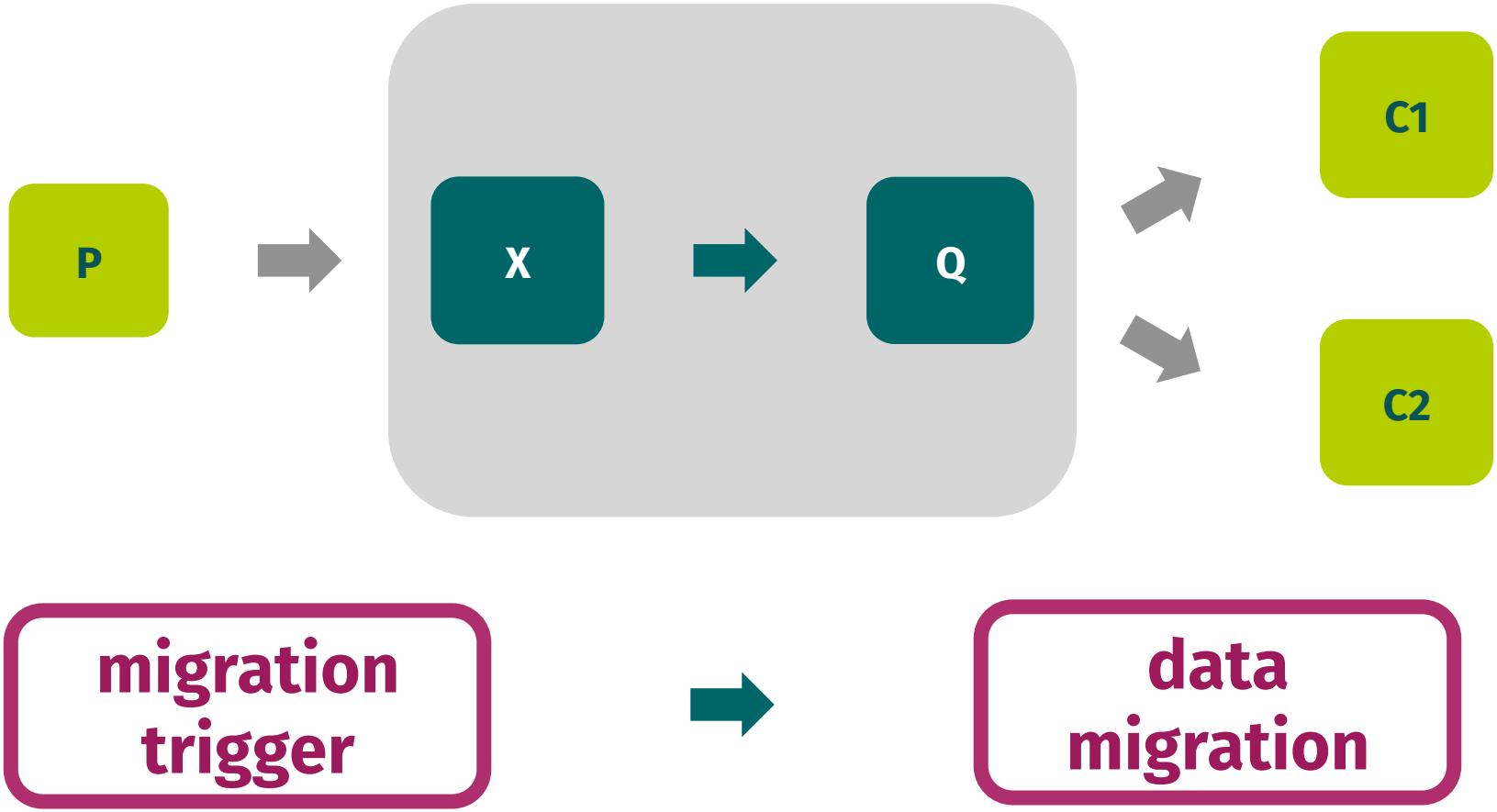
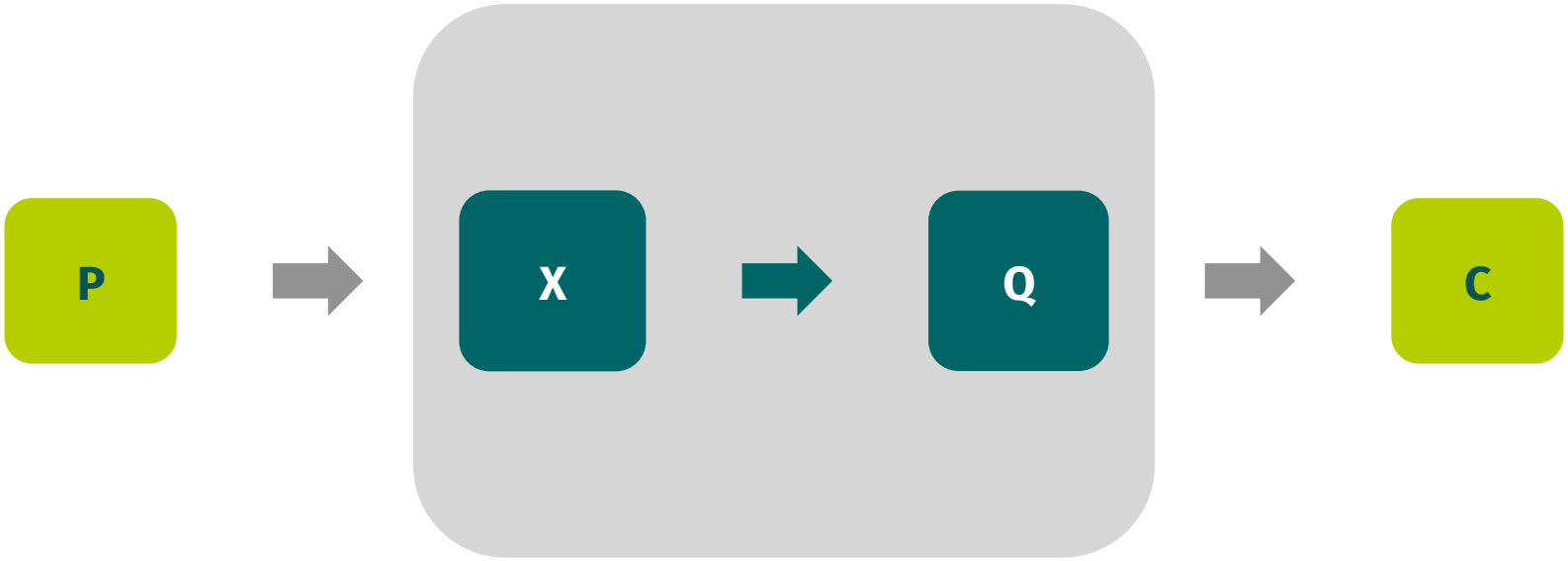


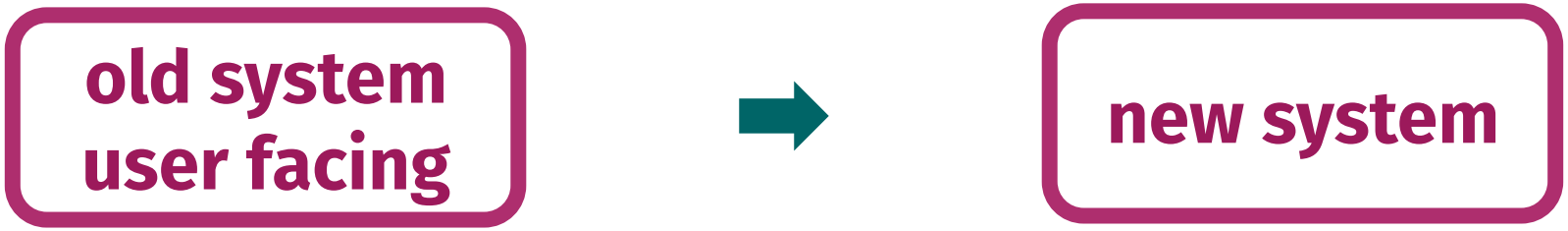
image upload user facing → **image processing**

Scaling on the fly

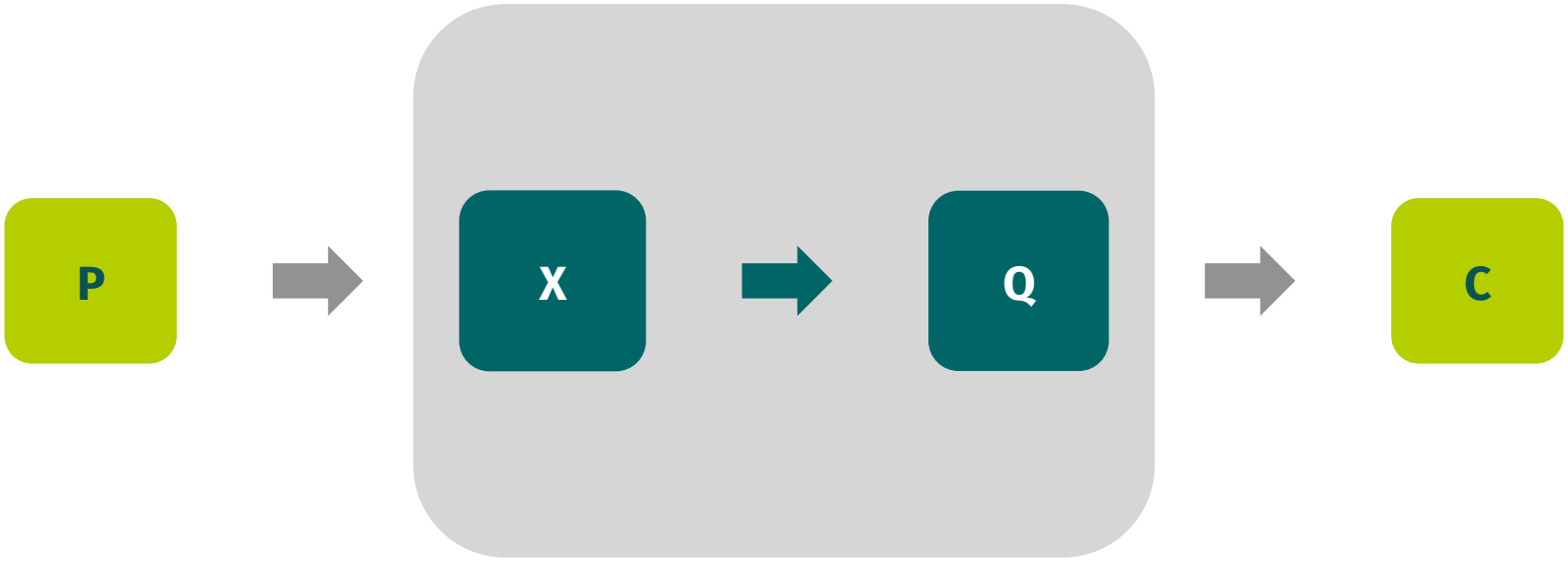


Migration without downtime





Test without impacting user



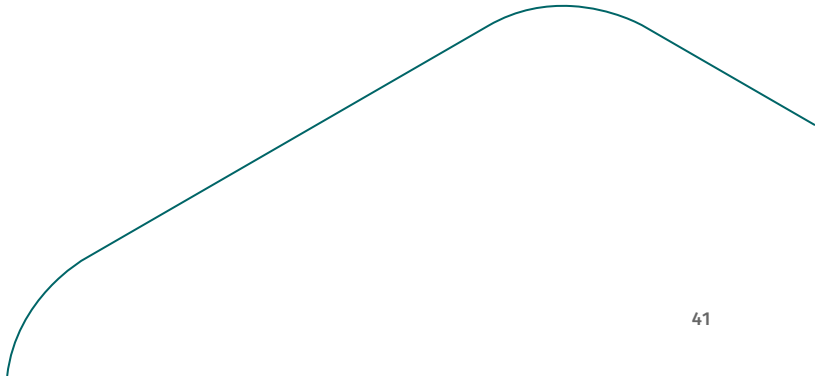
old system user facing

shadow calls

Example

using the Advanced Message Queuing Protocol

in Ruby



Ruby example
Connect to the broker

```
require "bunny"

connection = Bunny.new
connection.start

channel = connection.create_channel
```

Ruby example
Create queue

```
task_queue = channel.queue(
     "awesome_task_queue",
     durable: true
     )
```

Ruby example
Publish message

```
msg = "Hello “+Time.now.strftime('%T')

channel.default_exchange.publish(
      msg,
      routing_key: task_queue.name,
      persistent: true
)
```

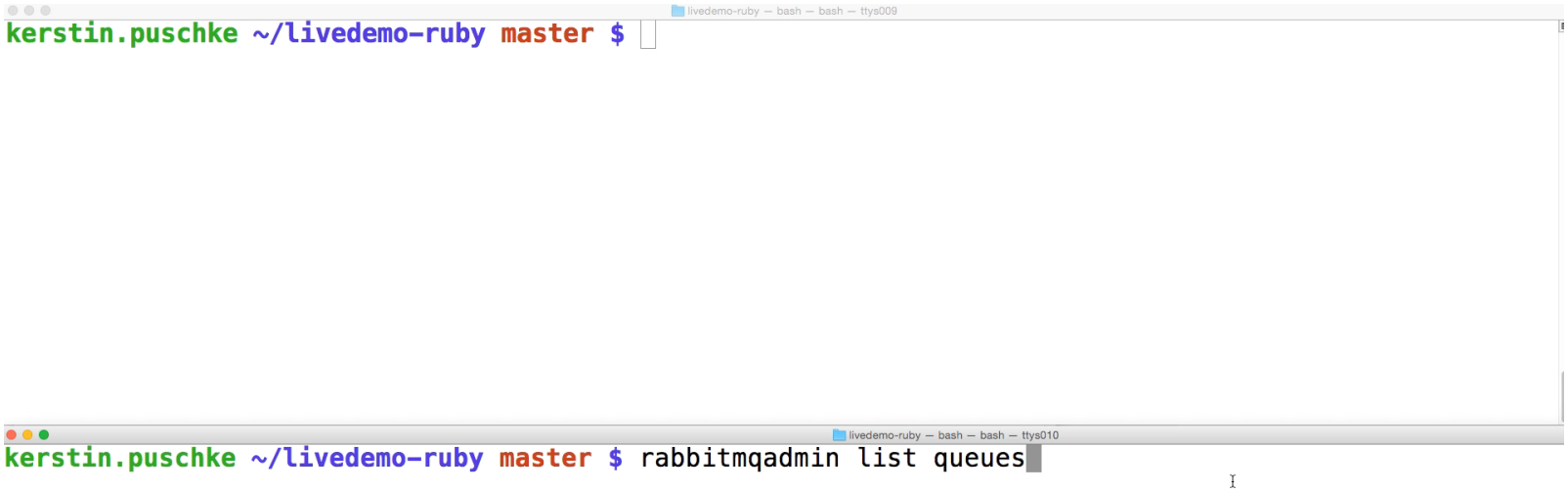
Ruby example
Consume message

```
task_queue.subscribe(block: true) do
|delivery_info, properties, payload|

  puts "Received '#{payload}'"
  sleep 2 # hard work

end
```

Demo


livedemo-ruby — bash — bash — ttys009
kerstin.puschke ~/livedemo-ruby master $
livedemo-ruby — bash — bash — ttys010
kerstin.puschke ~/livedemo-ruby master $ rabbitmqadmin list queues

In a nutshell

- Interoperability
- Improve response times
- Avoid downtimes
- Scale on the fly
- Simplify code
- Easy to get started

Decouple all the things

Asynchronous messaging keeps it simple

Kerstin Puschke
@titanoboa42

Codemotion Berlin 2016