Practical Messaging

A 101 guide to messaging

Ian Cooper

Twitter: ICooper

Who are you?

- Software Developer for more than 25 years
  - Stuff I care about: Messaging, EDA, Microservices, TDD, XP, OO, RDD & DDD, Code that Fits in My Head, C#
  - Places I have worked: DTI, Reuters, Sungard, Beazley, Huddle, Just Eat Takeaway
- No smart folks
  - Just the folks in this room

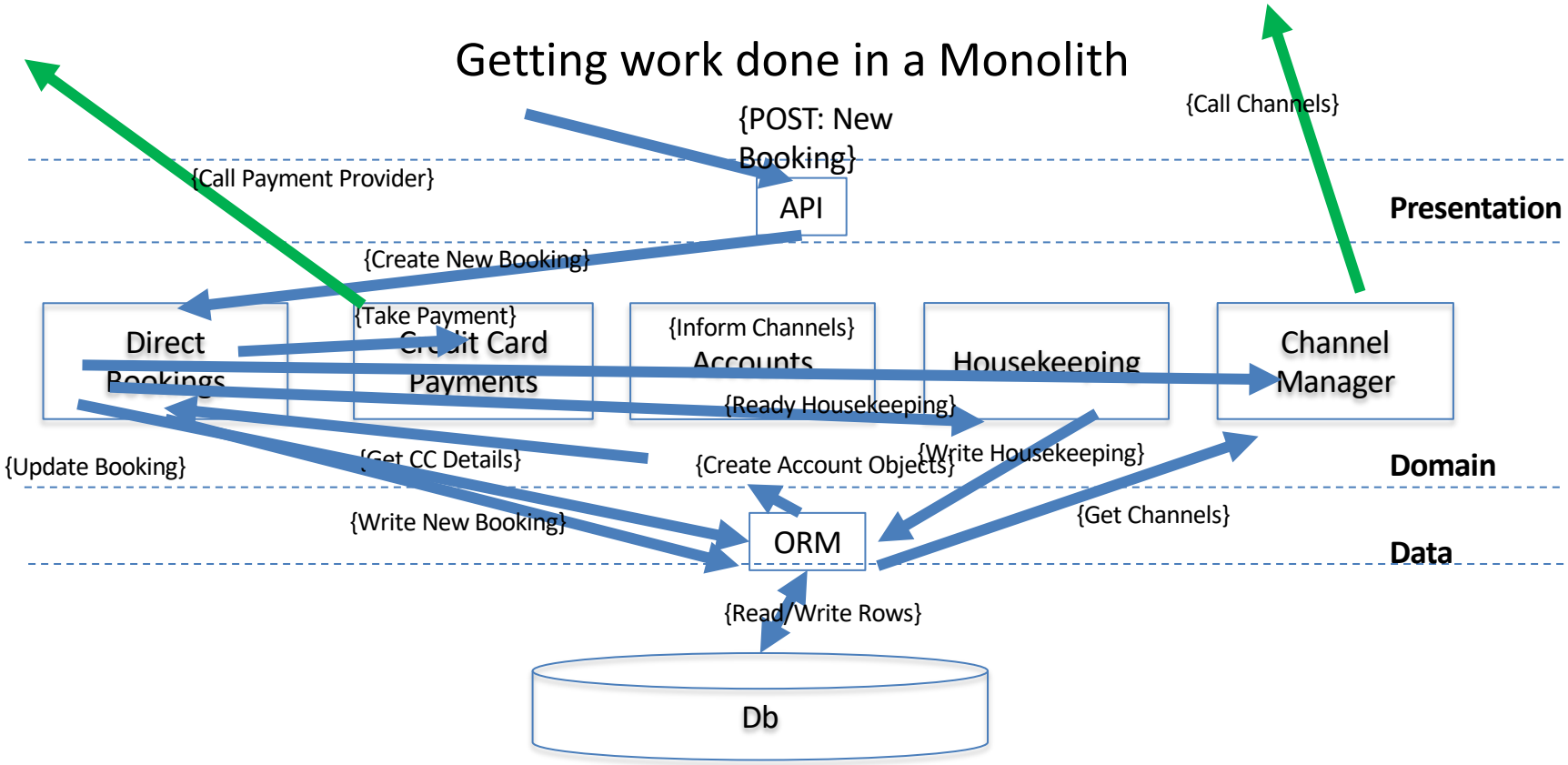
Day Two Agenda

- Event Driven Collaboration
- Reliability
- Order
- Consumers
- Documentation
- Middleware & Frameworks Discussion (Separate Decks)

Day Two

3.1 EVENT DRIVEN COLLABORATION

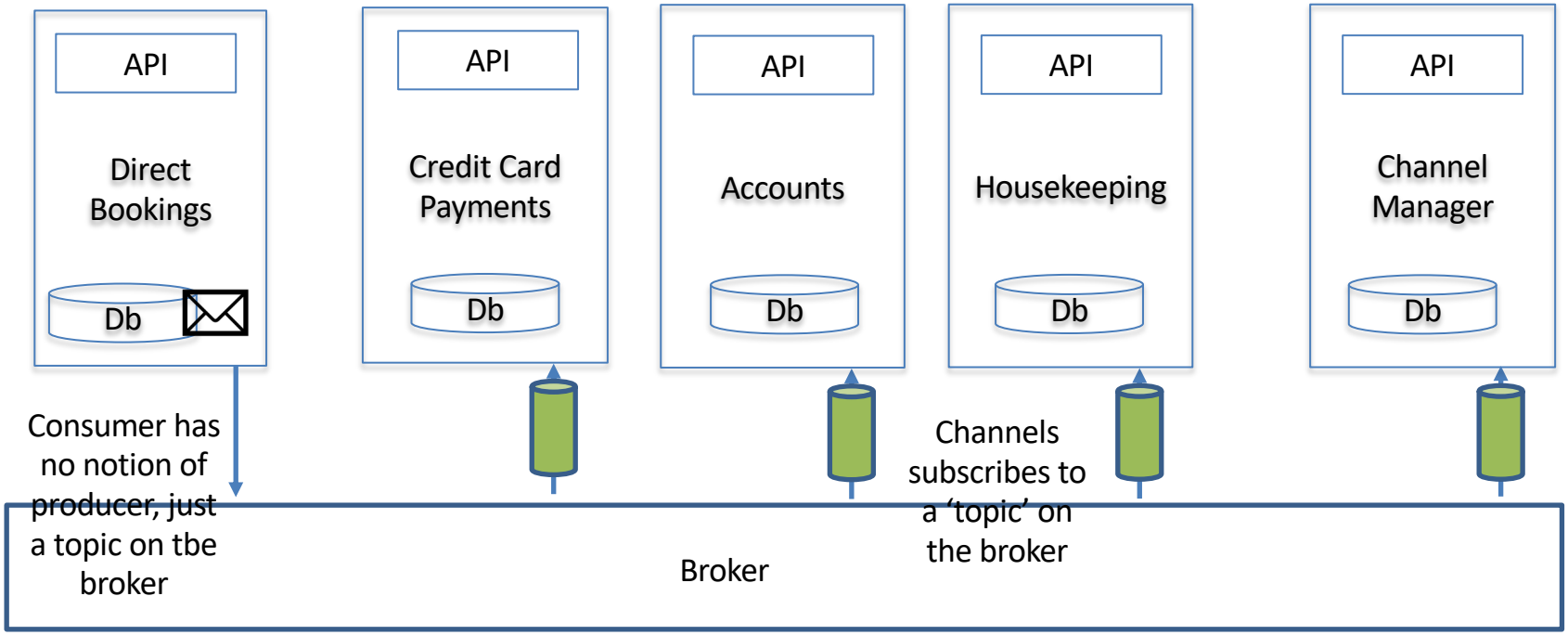
Getting work done in a Monolith



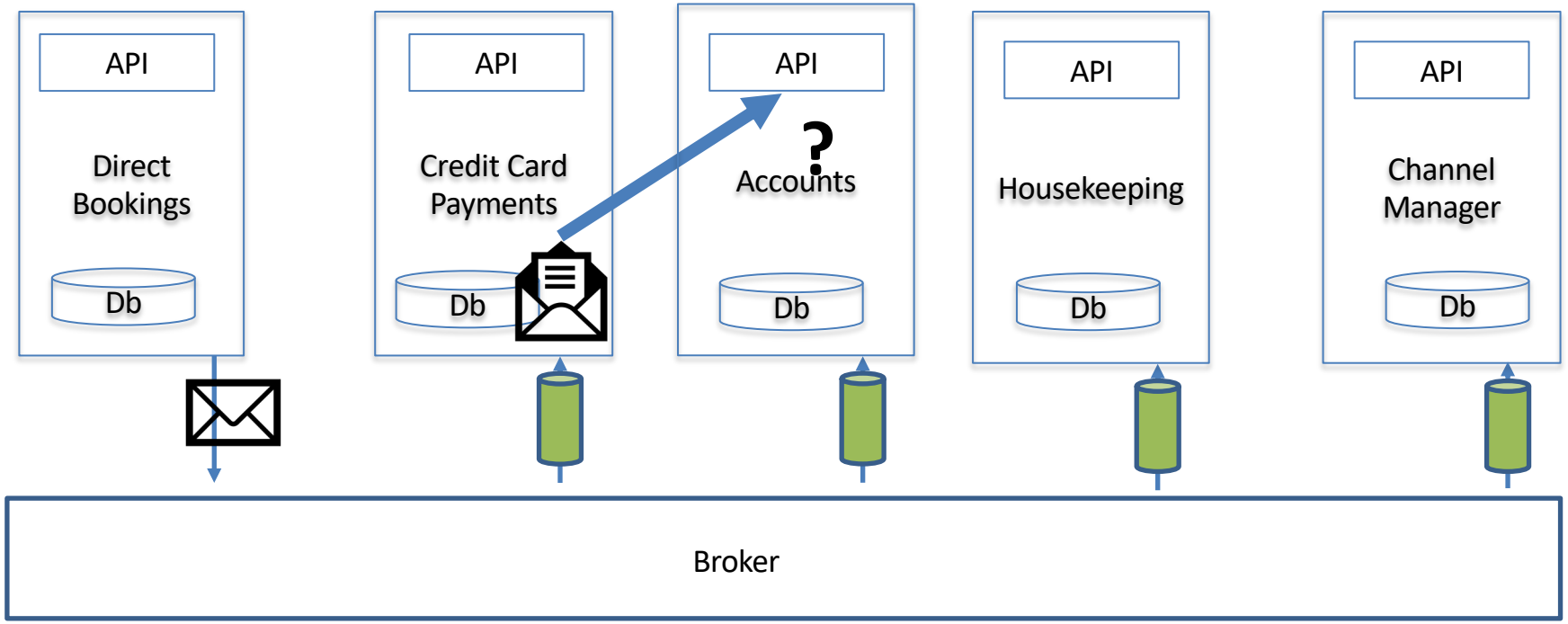
Event Driven Architecture

"Messaging over a lightweight message bus such as RabbitMQ"

Fowler and Lewis



Event Driven Architecture

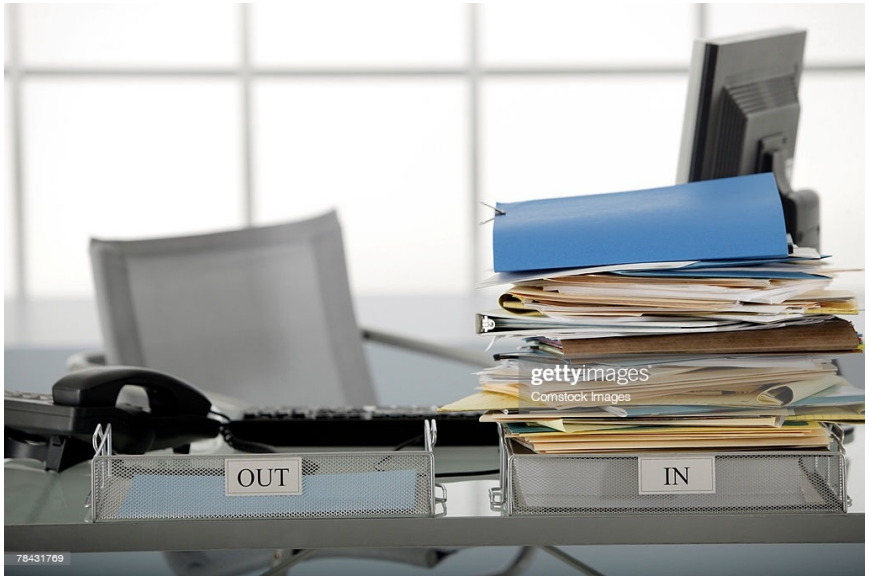


Paper Workflows

"My life looked good on paper - where, in fact, almost all of it was being lived." - Martin Amis

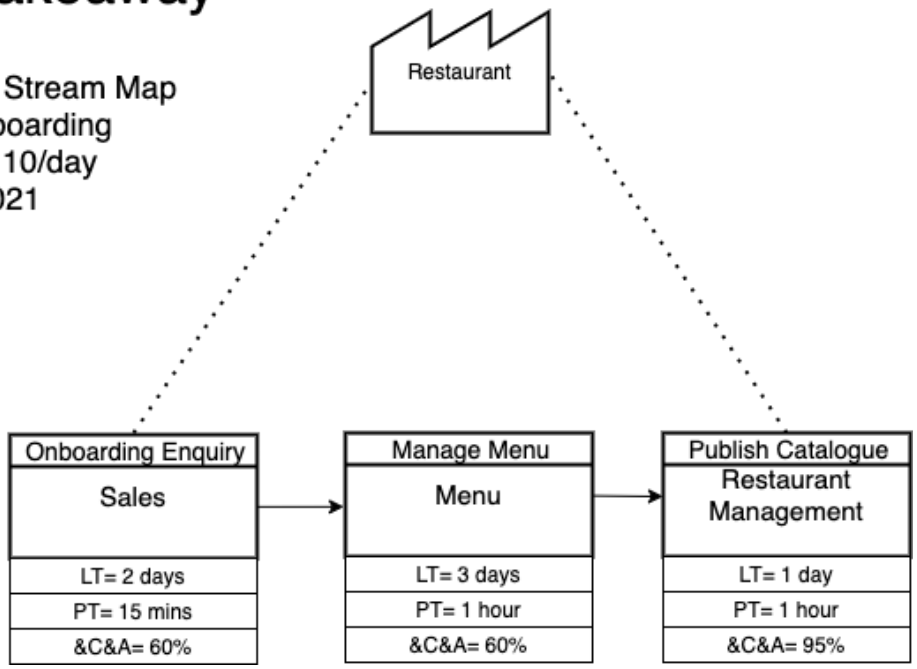
(Message)Event – Skinny, Notification

(Message)Document – Fat

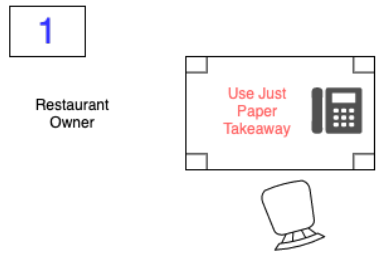

OUT
IN
gettyimages
Comstock Images
78431769

Just Paper Takeaway

Current State Value Stream Map
Restaurant Onboarding
Demand Rate 10/day
29 SEP 2021



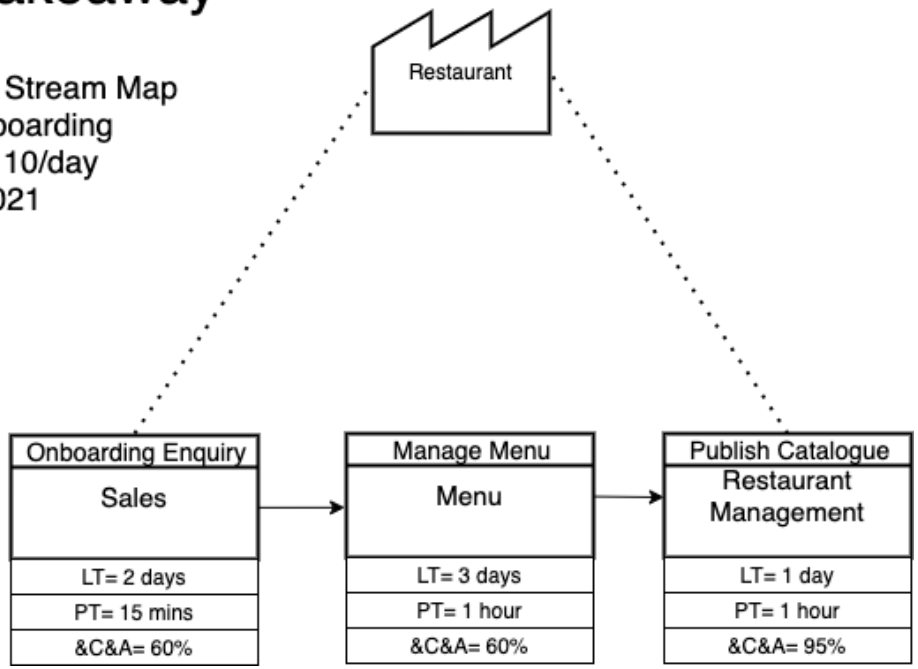
JUST EAT Takeaway.com

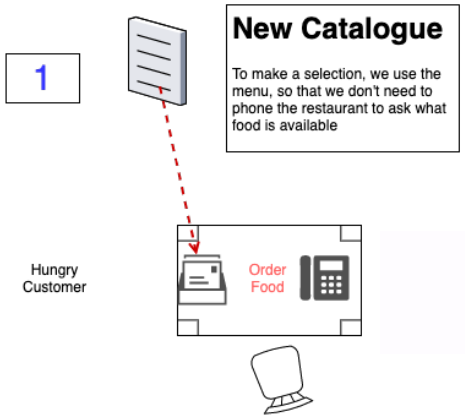


Restaurant Onboarding

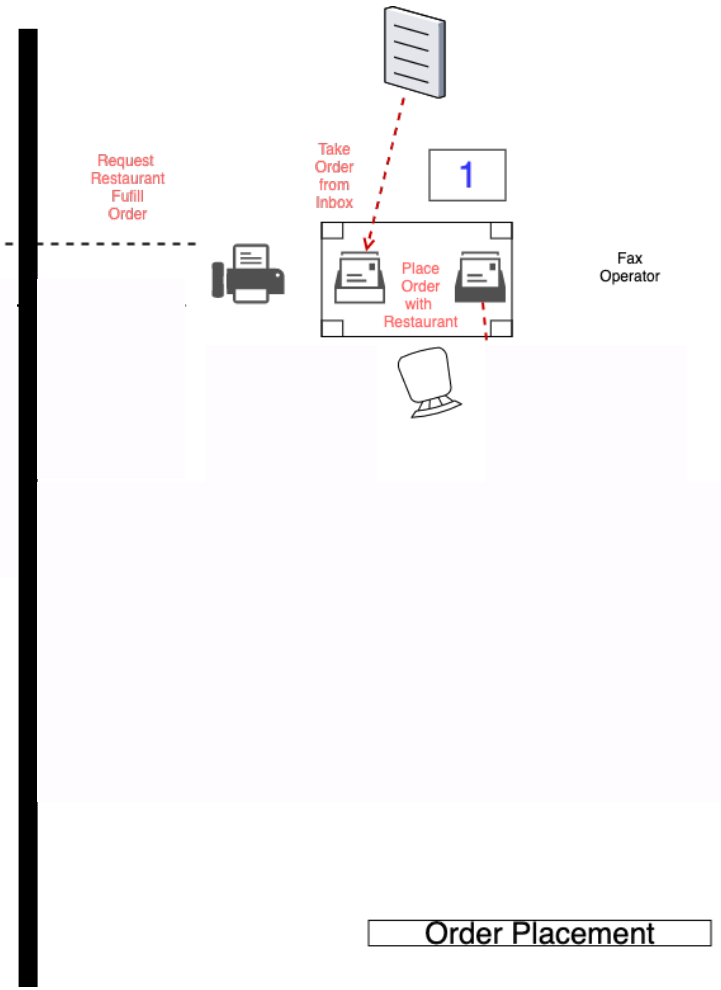
Just Paper Takeaway

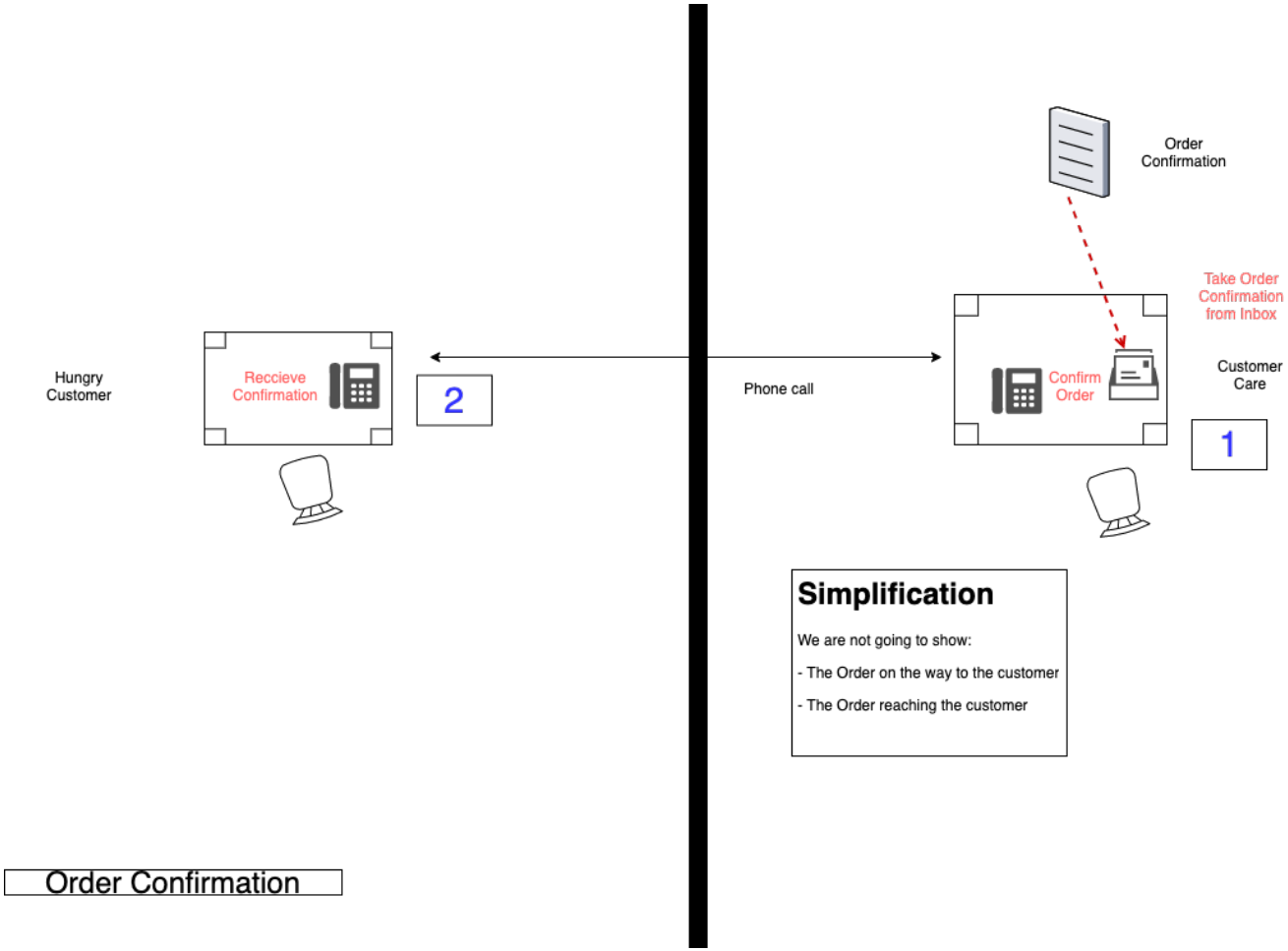
Current State Value Stream Map
Restaurant Onboarding
Demand Rate 10/day
29 SEP 2021



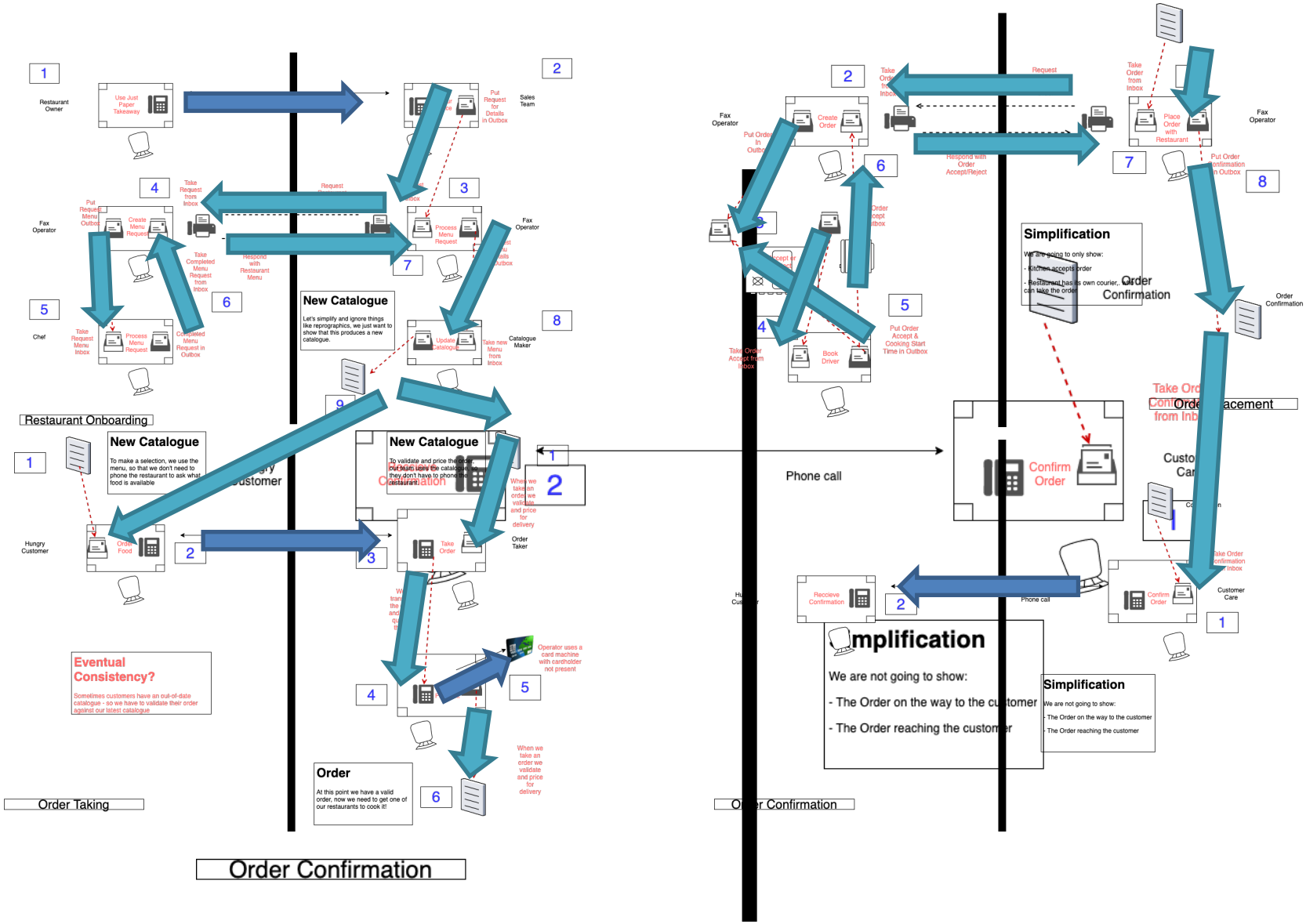


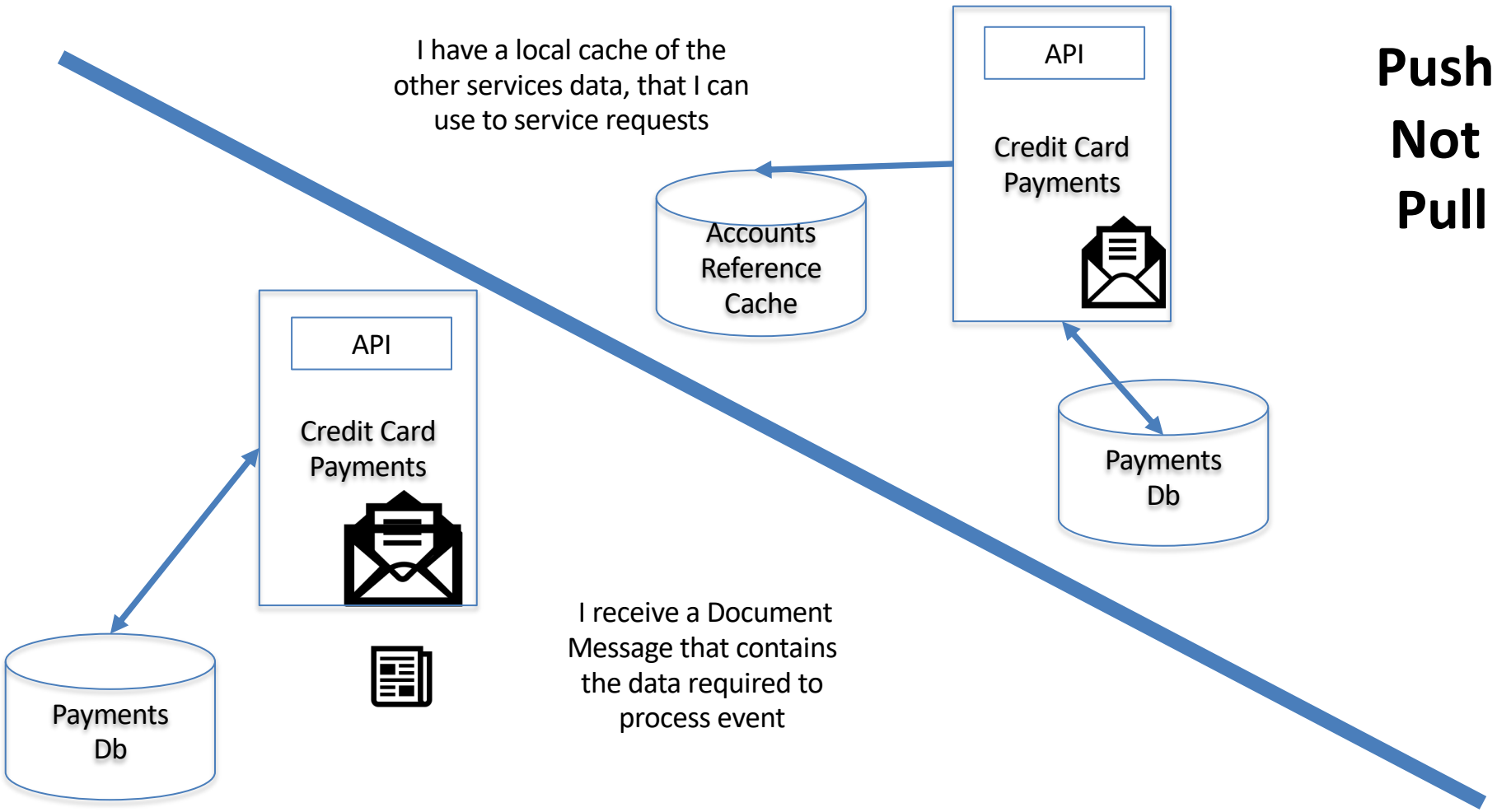
Order Taking


Request
Restaurant
Fufill
Order
Take
Order
from
Inbox
1
Place
Order
with
Restaurant
Fax
Operator
Order Placement



JUST EAT Takeaway.com


Restaurant Onboarding
New Catalogue
Let's simplify and ignore things like reprographics, we just want to show that this produces a new catalogue.
New Catalogue
To make a selection, we use the menu, so that we don't need to phone the restaurant to ask what food is available
New Catalogue
Eventual Consistency?
Sometimes customers have an out-of-date catalogue - so we have to validate their order against our latest catalogue
Order
At this point we have a valid order, now we need to get one of our restaurants to cook it!
Order Taking
Order Confirmation
Phone call
Simplification
We are going to only show:
- Kitchen accepts order
- Restaurant has its own courier, who can take the order
Order Confirmation
Simplification
We are not going to show:
- The Order on the way to the customer
- The Order reaching the customer
Simplification
We are not going to show:
- The Order on the way to the customer
- The Order reaching the customer



Push Not Pull

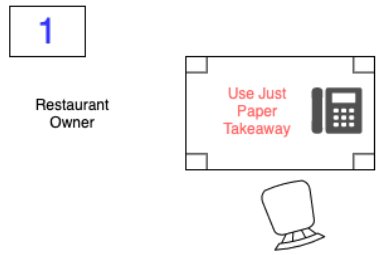
Orchestration and Choreography

3.1.1 PIPES AND FILTERS

What is a microservice?

SOA is focused on business *processes*. These *processes* are performed in different steps (also called *activities* or *tasks*) on different systems. The primary goal of a **service** is to represent a "natural" step of business functionality. That is, according to the domain for which it's provided, *a service should represent a self-contained functionality that corresponds to a real-world business activity*.

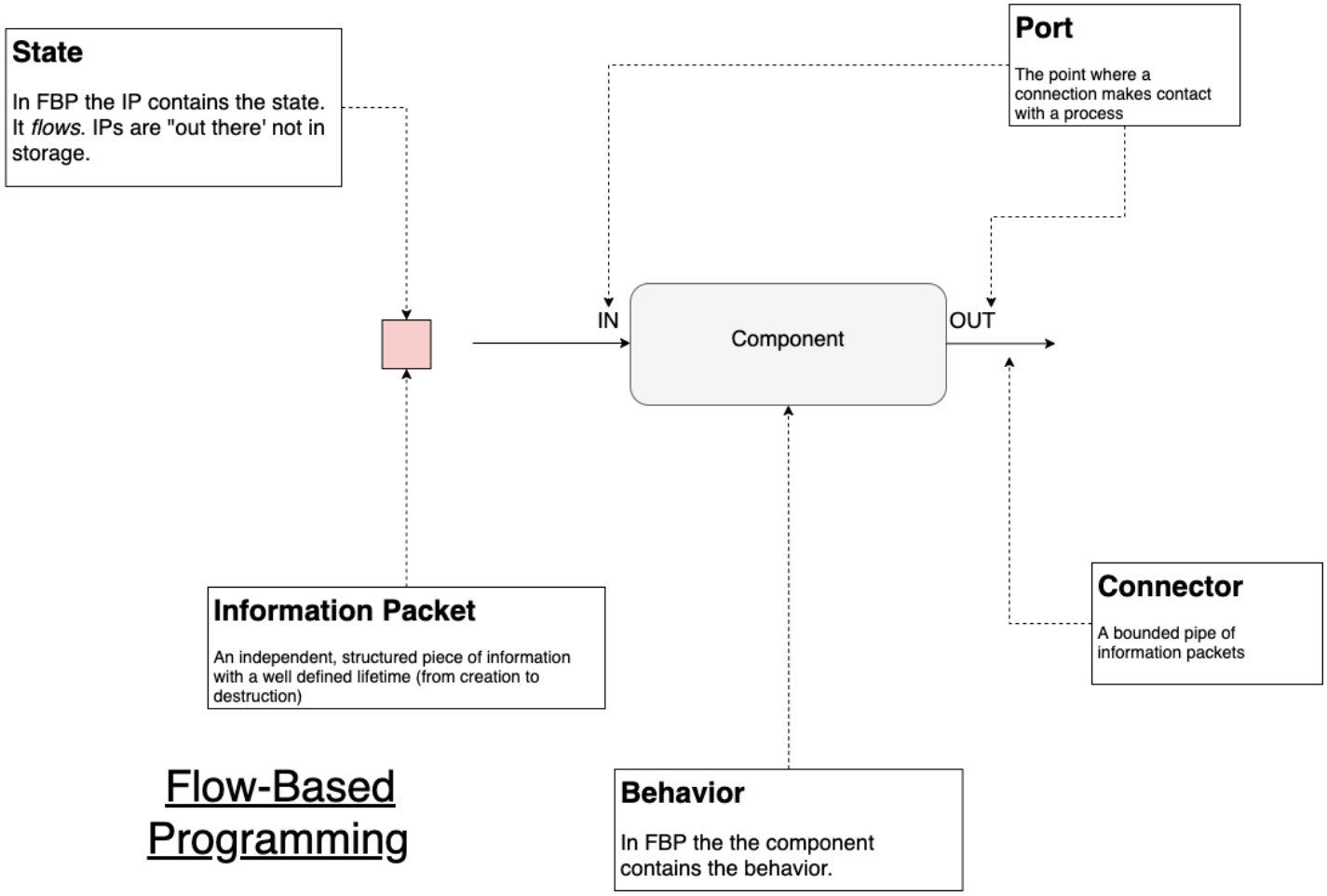
Josuttis, Nicolai M.. SOA in Practice: The Art of Distributed System Design . O'Reilly Media. Kindle Edition.

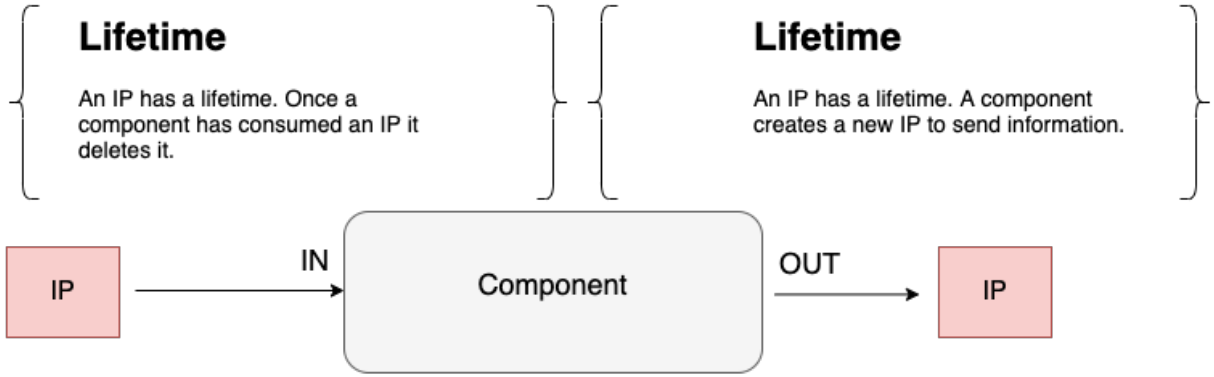


Restaurant Onboarding

Flow Based Programming

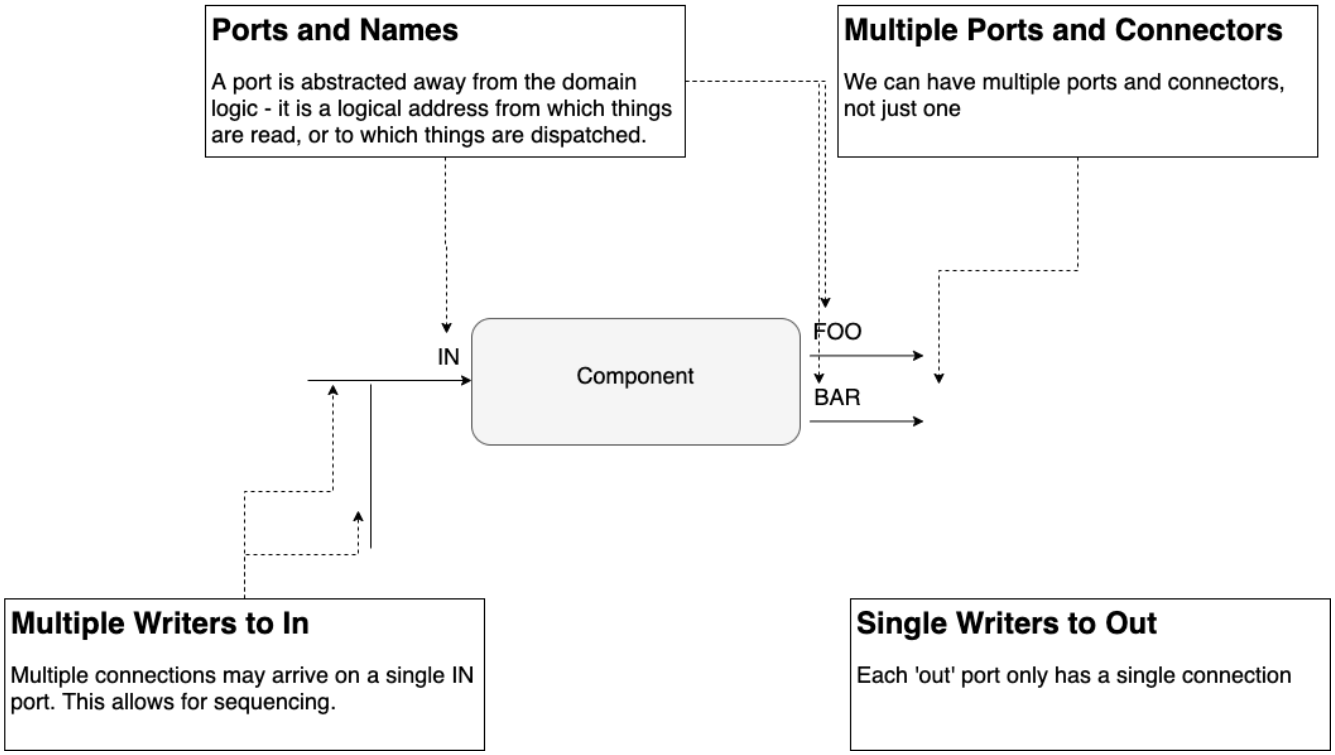
"Πάντα ῥεῖ – Everything flows" (Heraclitus of Ephesus, ca. 500 BCE)

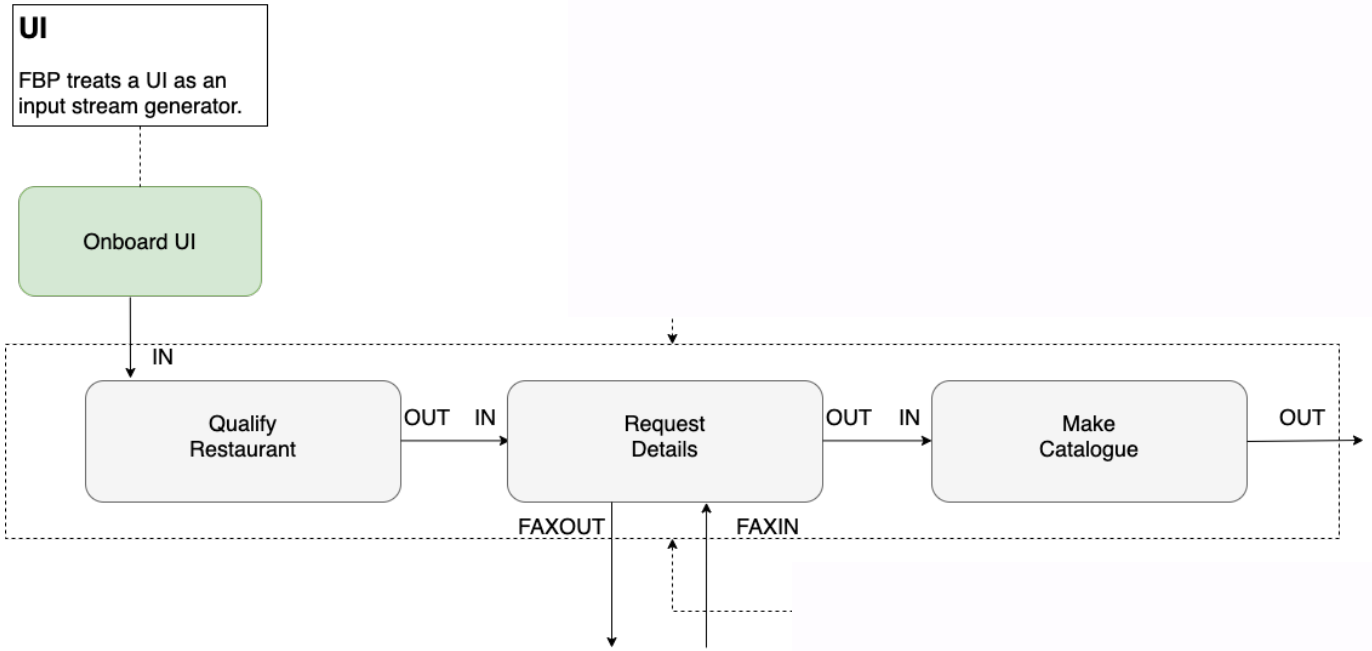

State
In FBP the IP contains the state. It *flows*. IPs are "out there' not in storage.
Port
The point where a connection makes contact with a process
IN
Component
OUT
Information Packet
An independent, structured piece of information with a well defined lifetime (from creation to destruction)
Connector
A bounded pipe of information packets
Flow-Based Programming
Behavior
In FBP the the component contains the behavior.



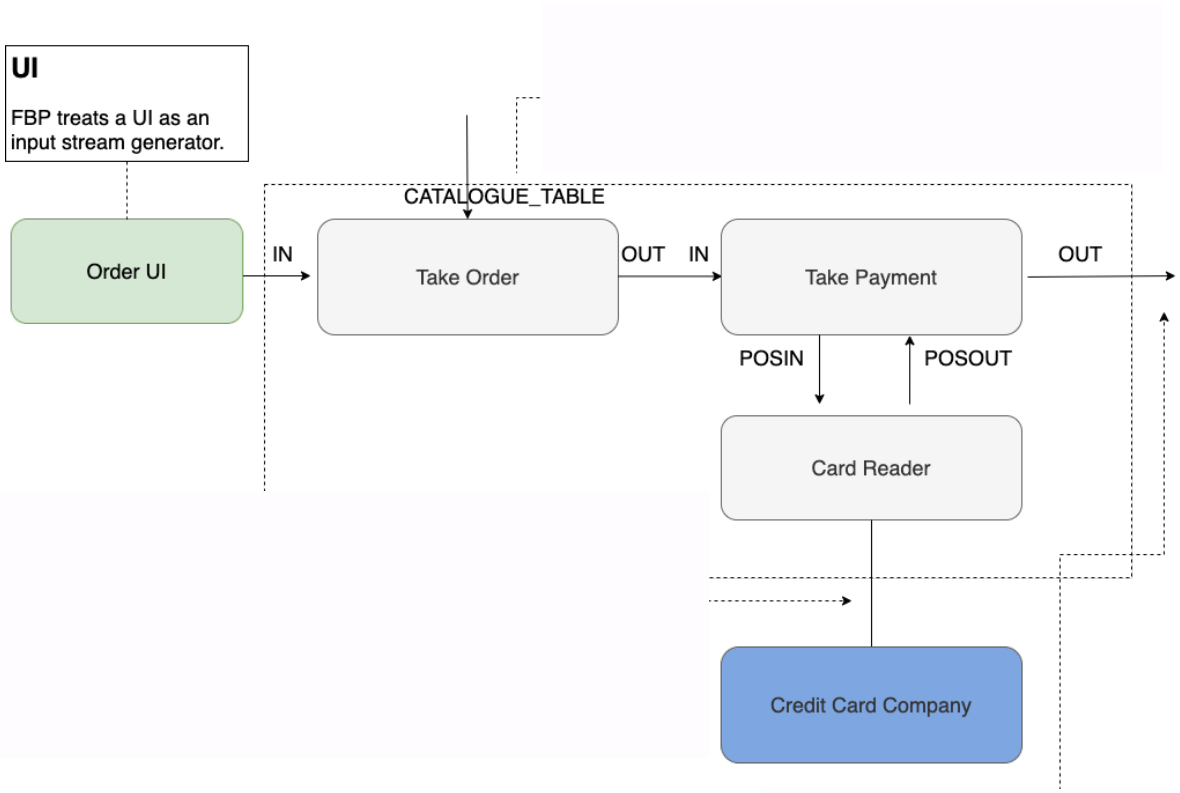
Flow-Based Programming

Ports and Connectors

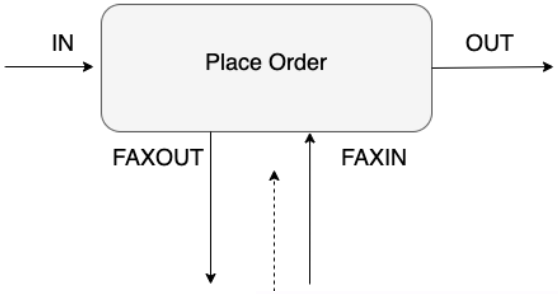




Onboard Restaurant


UI
FBP treats a UI as an input stream generator.
CATALOGUE_TABLE
Order UI
IN
Take Order
OUT
IN
Take Payment
OUT
POSIN
POSOUT
Card Reader
Credit Card Company

Order Food

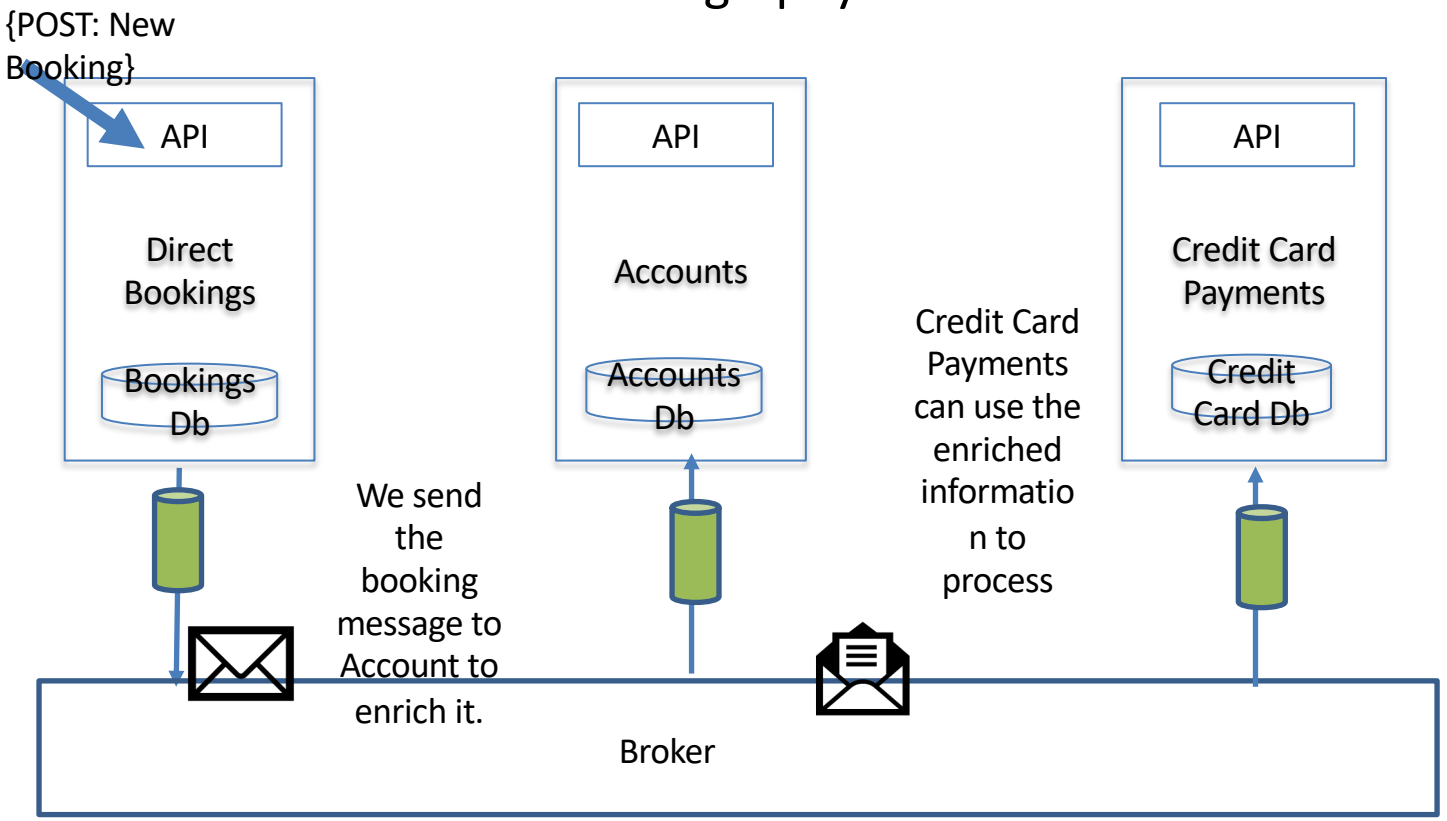


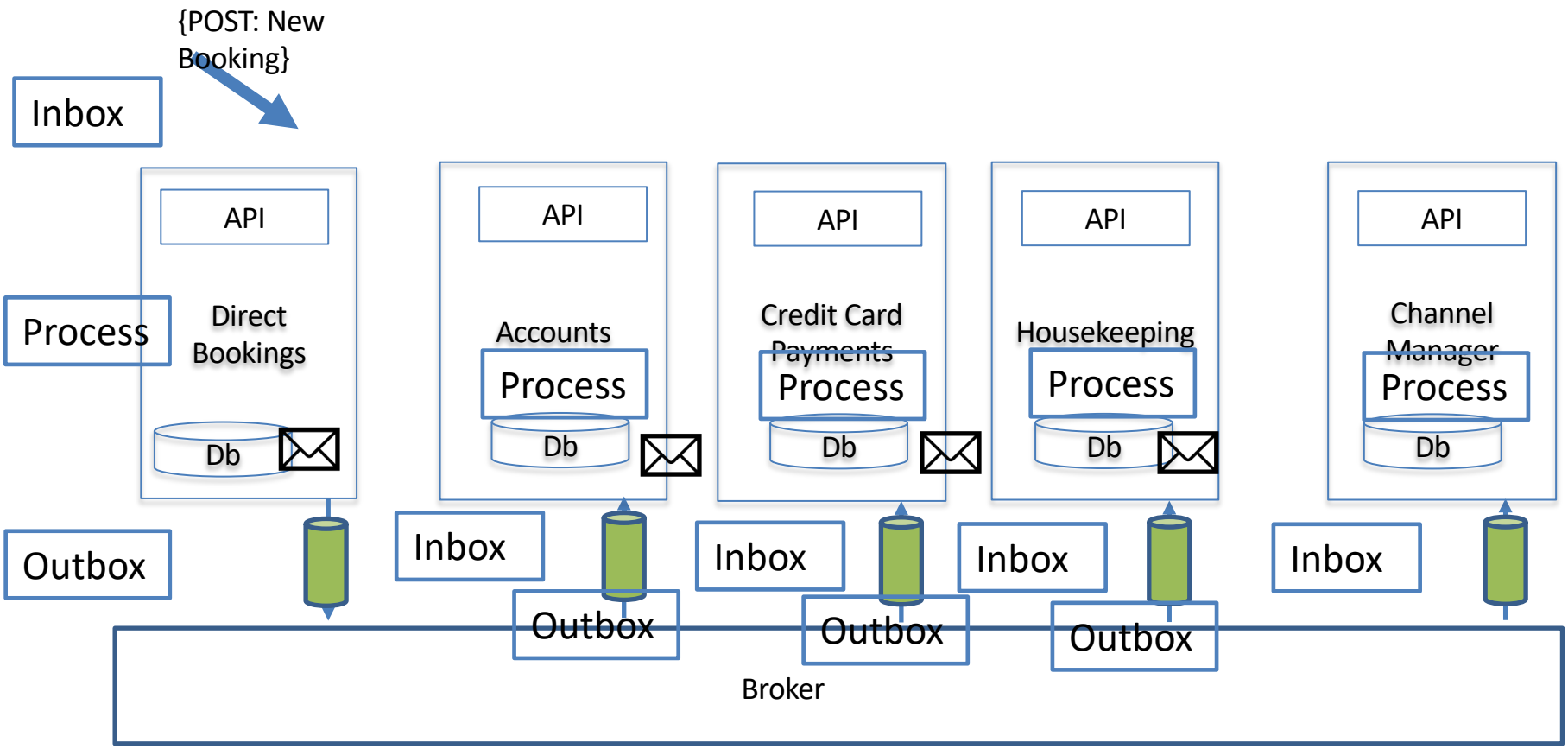
Order Placement

Choreography

Event-Oriented

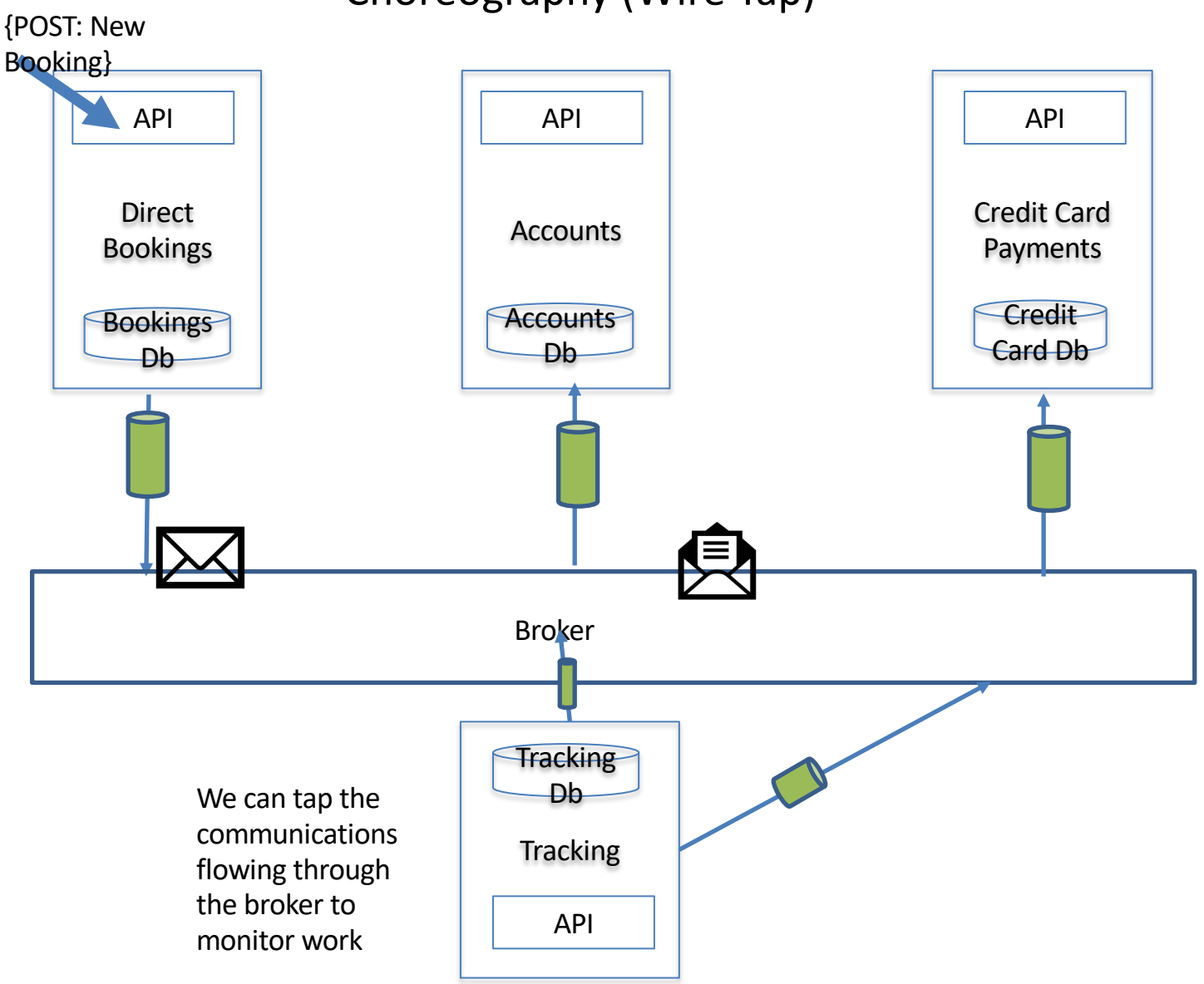
Choreography



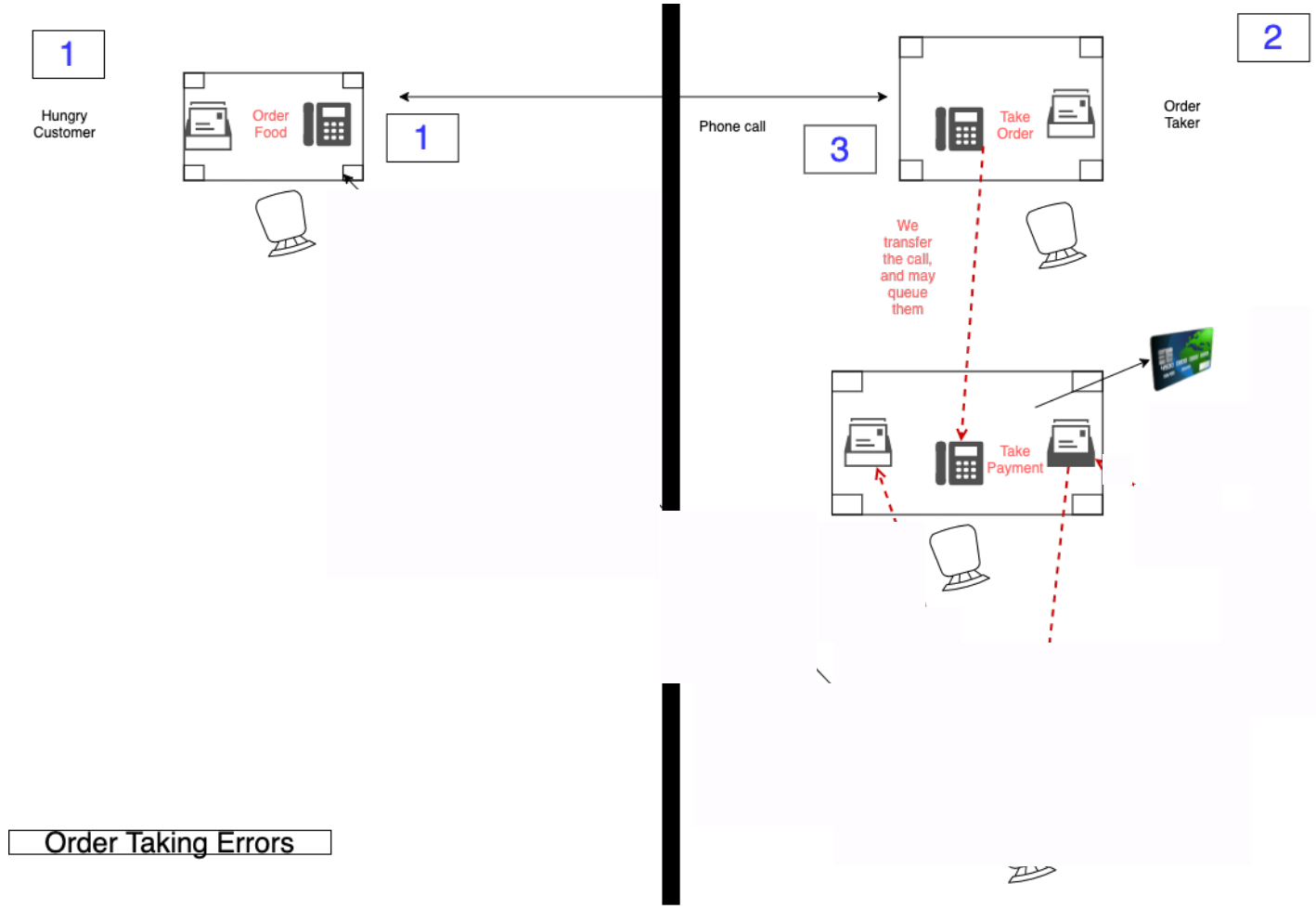

{POST: New Booking}
Inbox
API
Process
Direct Bookings
Db
Outbox
API
Accounts
Process
Db
Inbox
Outbox
API
Credit Card Payments
Process
Db
Inbox
Outbox
API
Housekeeping
Process
Db
Inbox
Outbox
API
Channel Manager
Process
Db
Inbox
Broker

Event-Oriented

Choreography (Wire Tap)

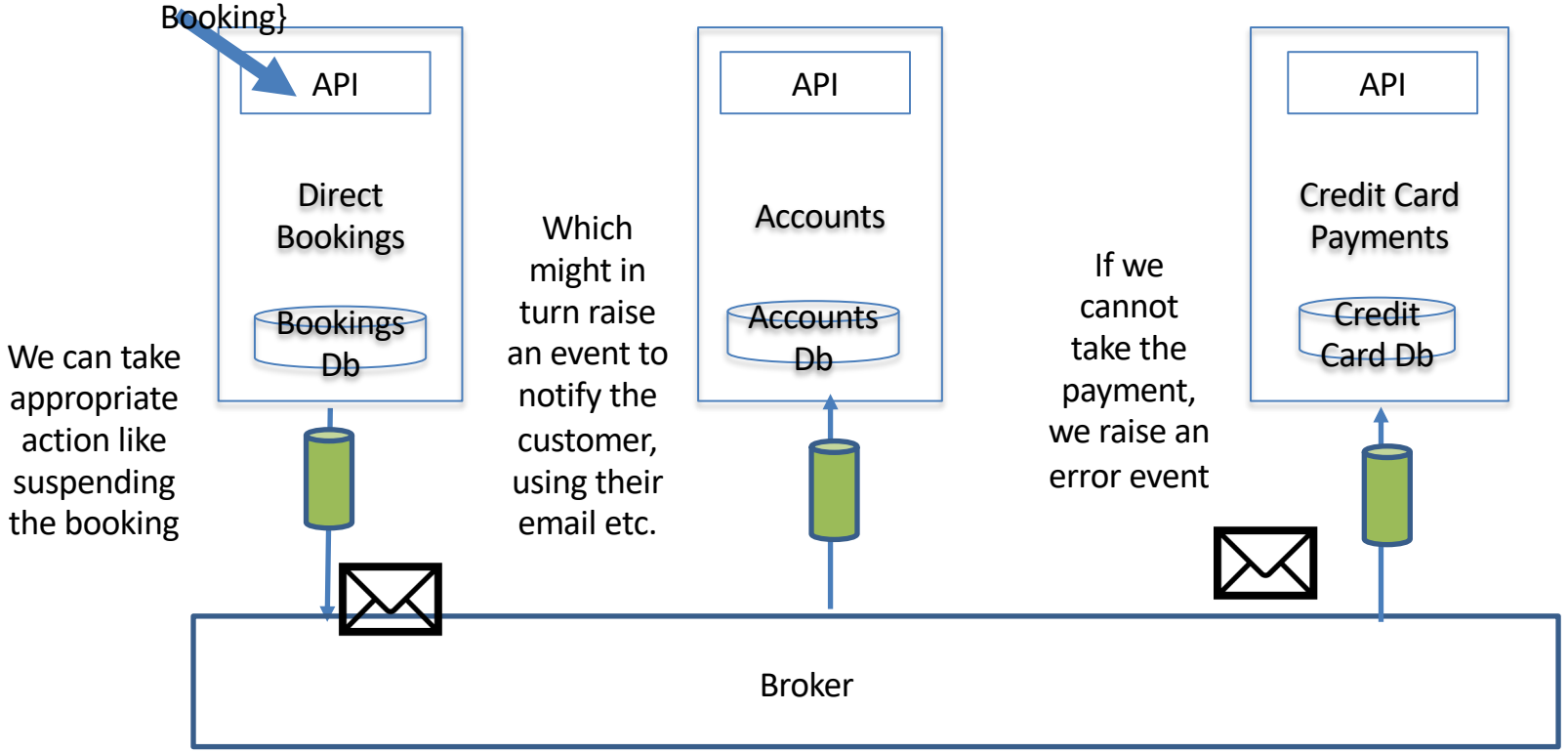


We can tap the communications flowing through the broker to monitor work

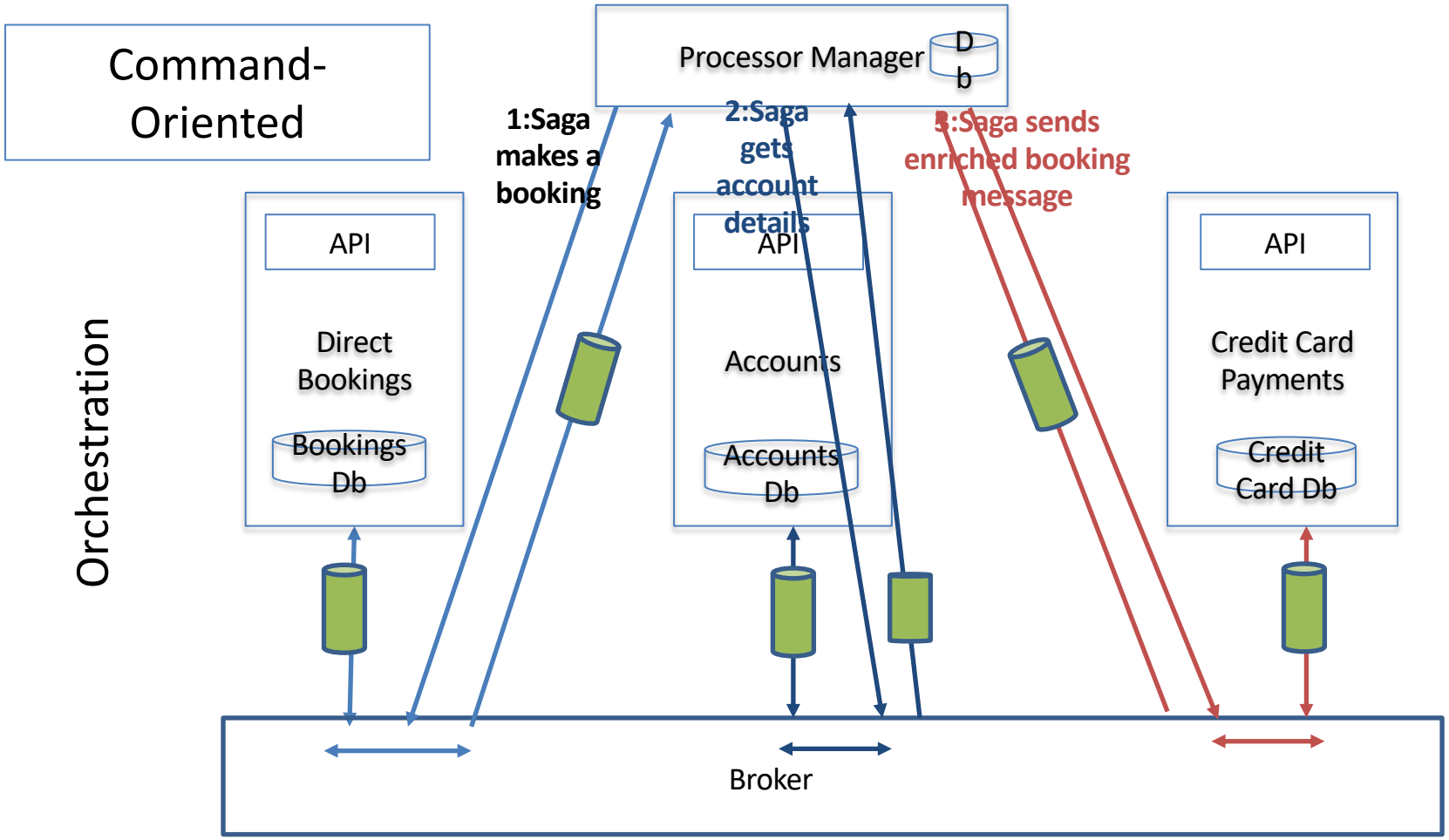

1
Hungry Customer
Order Food
1
Phone call
3
2
Take Order
Order Taker
We transfer the call, and may queue them
Take Payment
Order Taking Errors

Event-Oriented

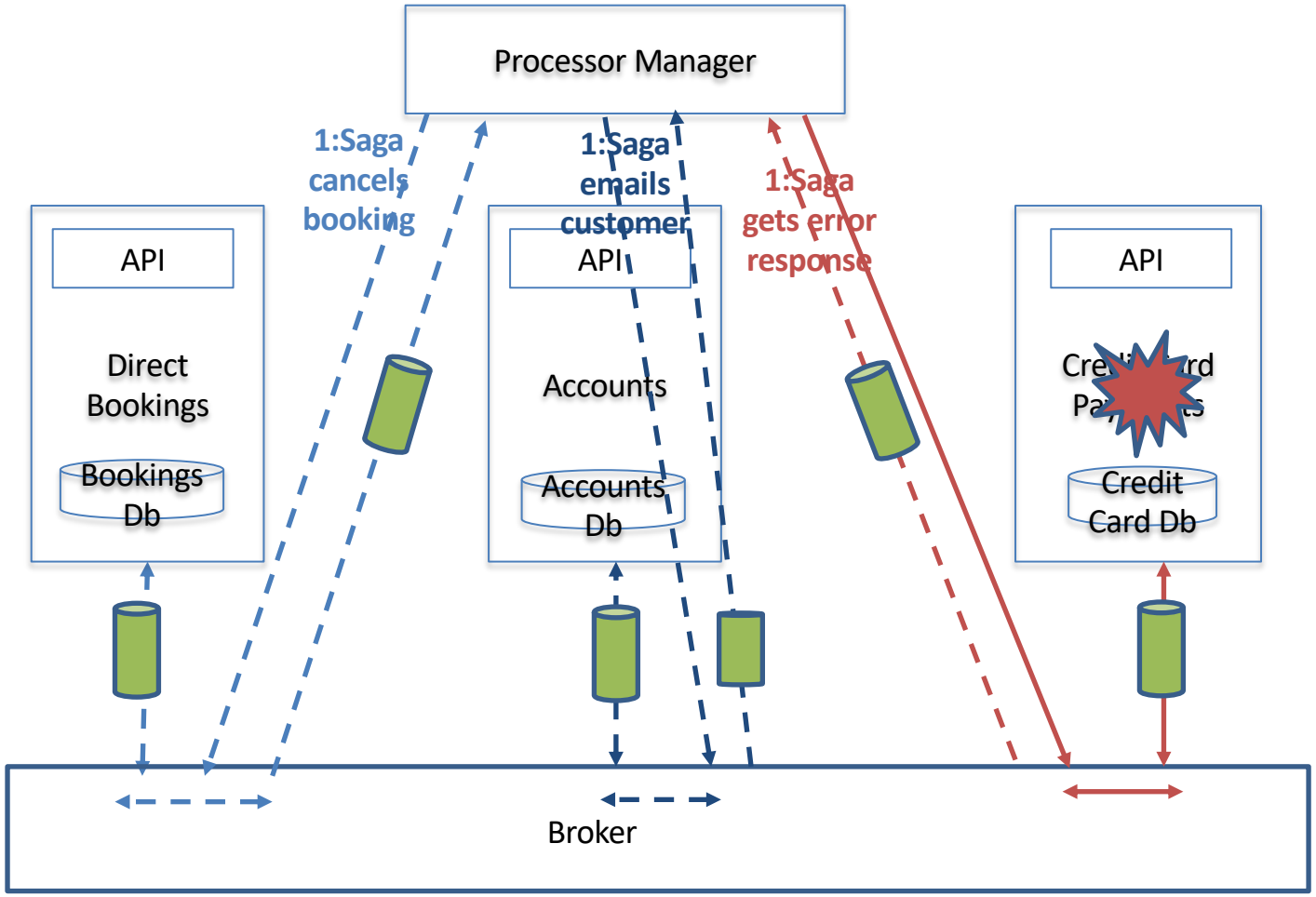
Choreography (Errors)



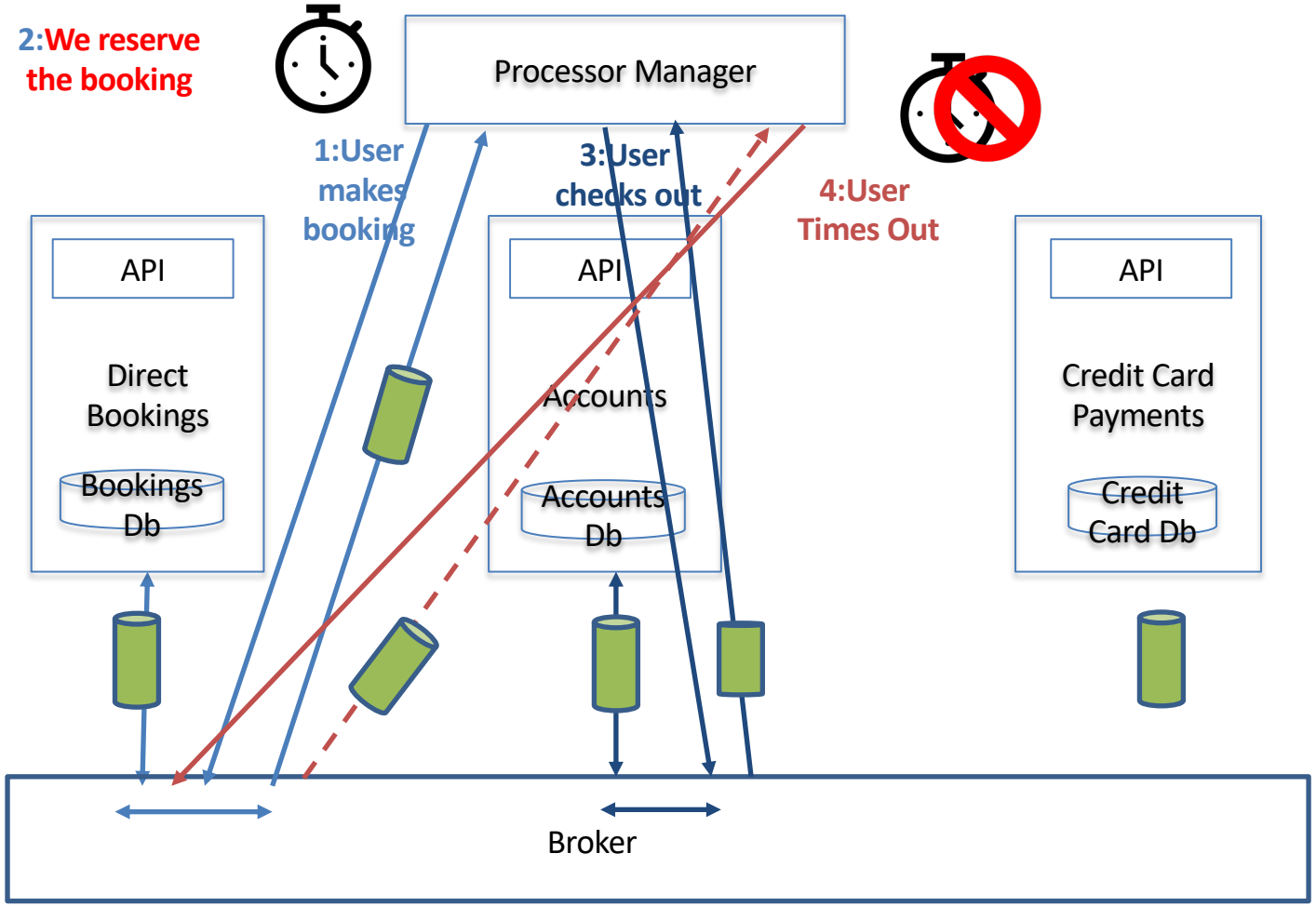
Orchestration


Command-Oriented
Orchestration
Processor Manager
Db
1:Saga makes a booking
2:Saga gets account details
3:Saga sends enriched booking message
API
Direct Bookings
Bookings Db
API
Accounts
Accounts Db
API
Credit Card Payments
Credit Card Db
Broker

Compensation (Error)

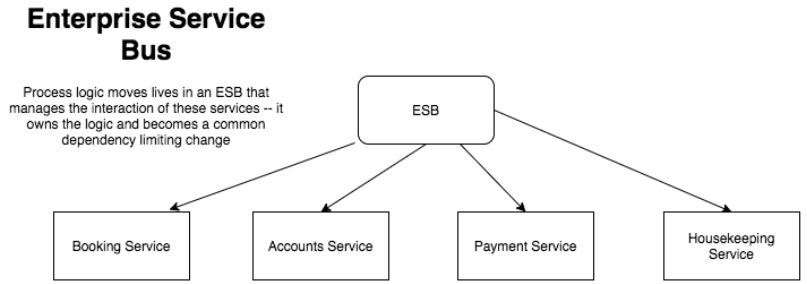


Reservations



Choreography ? Orchestration

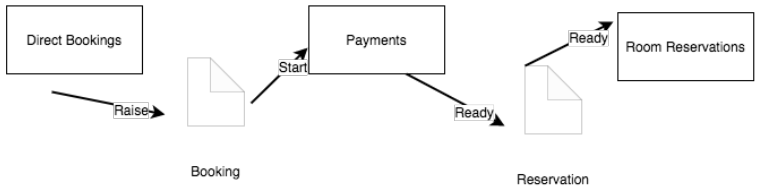
BAD



ENTITY SERVICE

CRUD based entity services just manage the lifecycle of entities we understand, but don't know how to handle a business process

GOOD



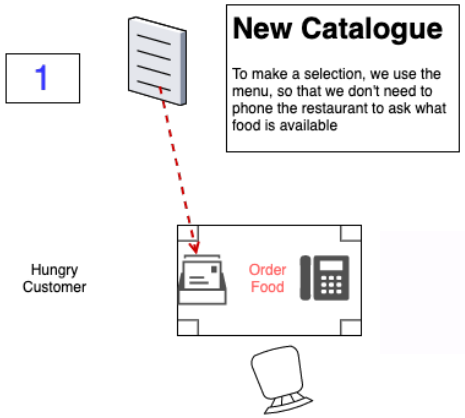
PROCESS SERVICE

A service that owns a business process does not rely on an external orchestrator and can simply pass messages to other services

Smart Endpoints and Dumb Pipes

=Reference Data

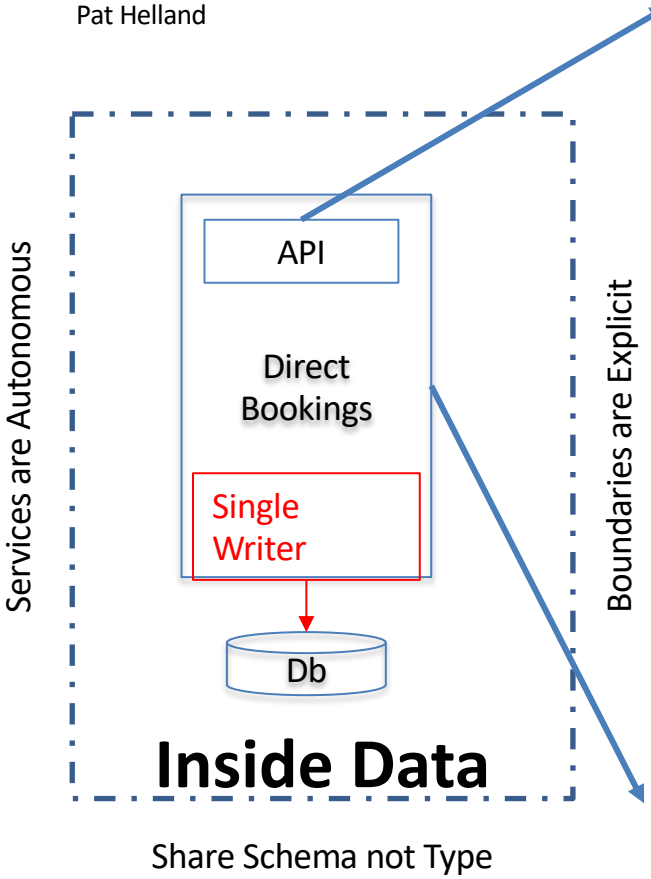
3.1.2 EVENT CARRIED STATE TRANSFER



Order Taking

Reference Data

Pat Helland



Share Schema not Type

GET \booking\12345 HTTP 1.1

Cache-Control: max-age=30
Last-Modified: Fri 20 April 2019 09:00 GMT
ETag: “33a64df551425fcc55e4d42a14879”

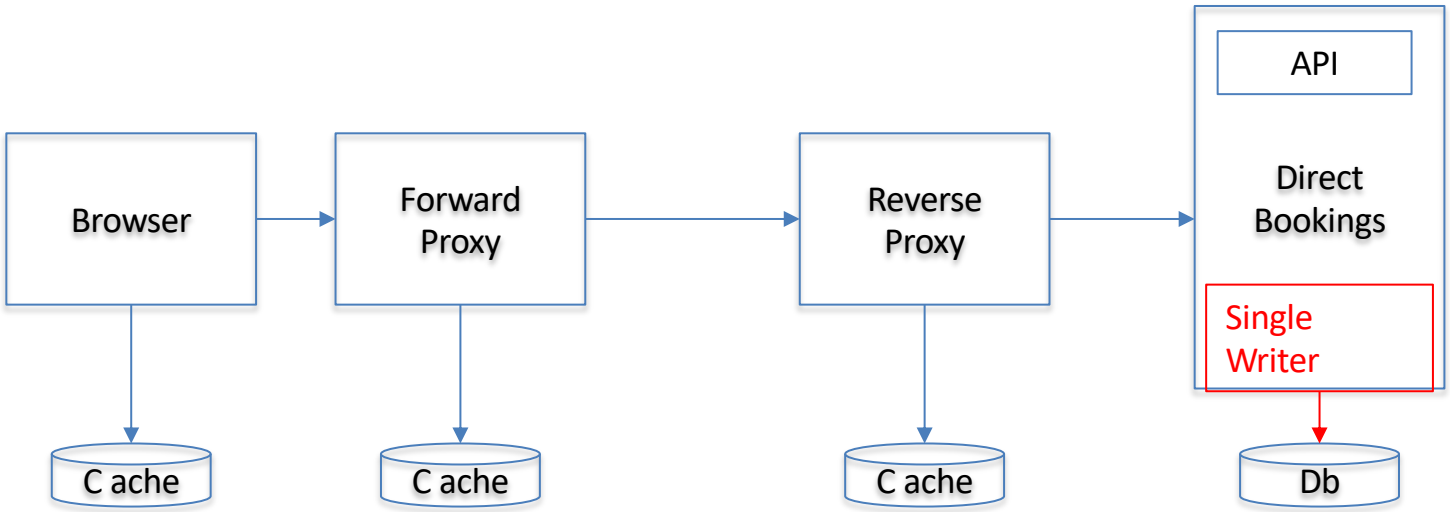
Outside Data

Immutable, stale, versioned

Booking.Created.Event
Message-Id: aa95b387-a17f-4907-a1d8-e597c322bfc6
Correlation-Id: d060cb3f-ec71-4257-b5c8-819d1a9ca6cb
Format: application/json
Schema: 1.1
{ items : [Booking {… “Version” : “1”}]}

Reference Data is Highly Cacheable

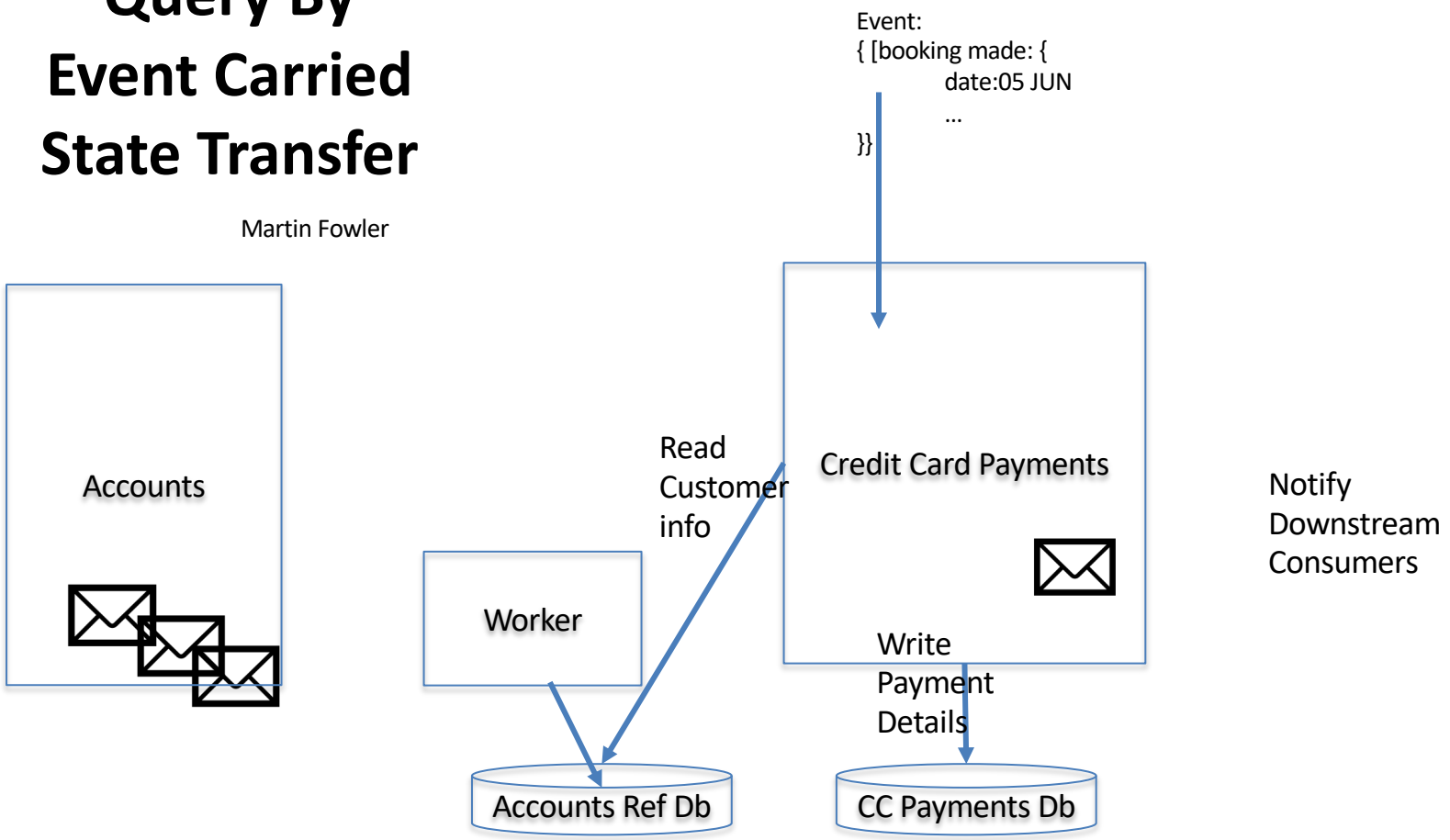
GET /booking/12345 HTTP 1.1
If-Modified-Since: Fri 19 April 2019 09:00 GMT



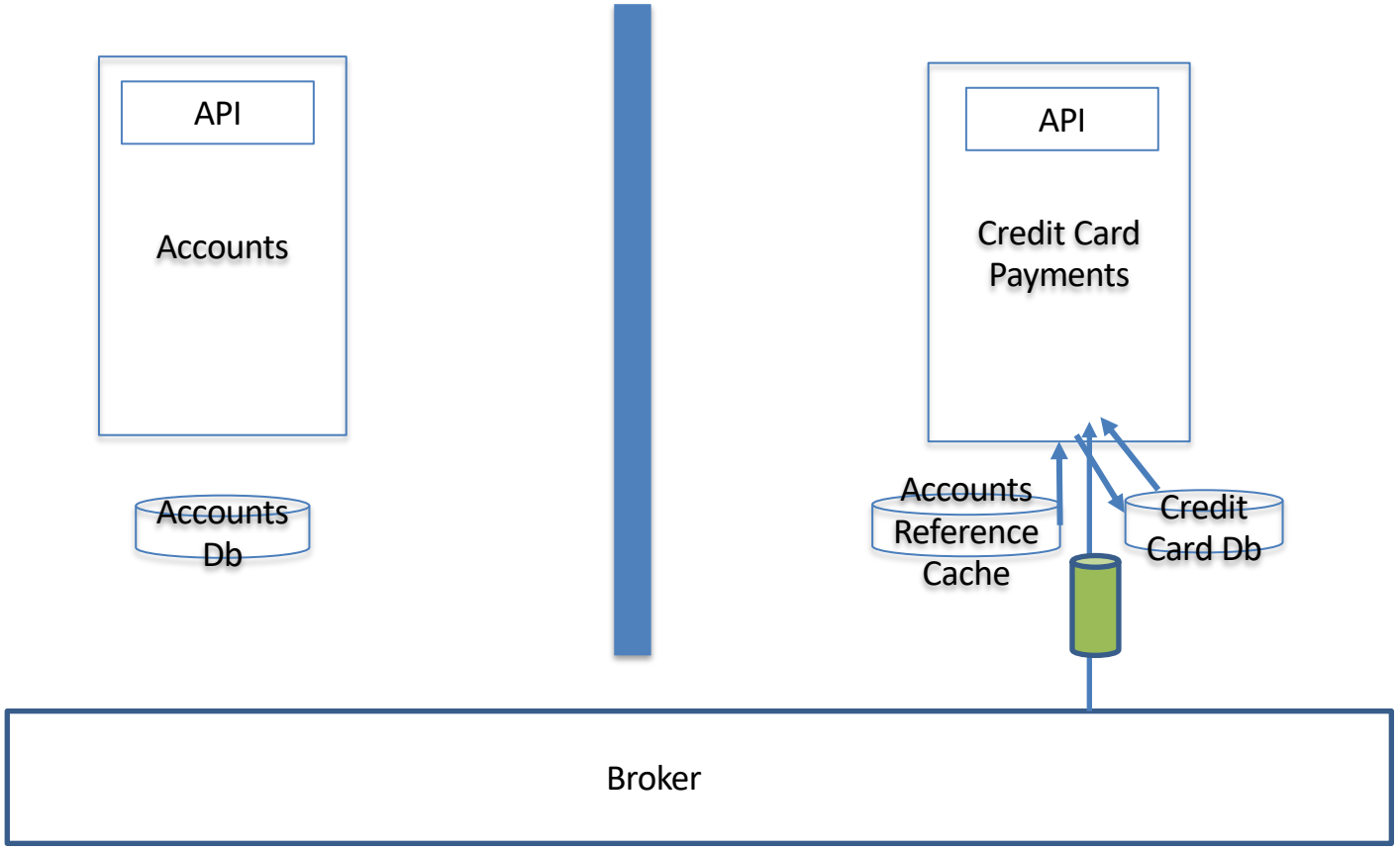
The Web scales by caching

Query By Event Carried State Transfer

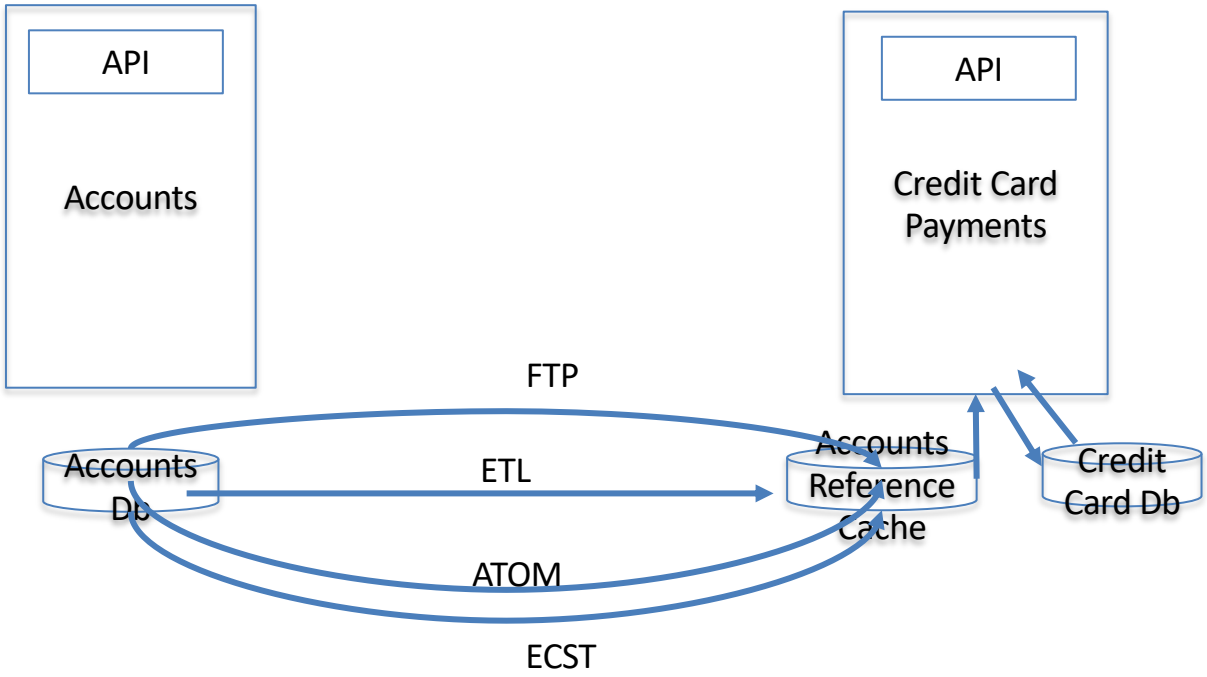
Martin Fowler



Messaging has Bulkheads



Reference Data is Protocol Agnostic



Types of Reference Data

- Operand Data (e.g. Product Catalog)
- Shared Collections (e.g. Customer, User)
- Historic Artefacts (e.g. Order History)

Would I cache you as Request Data?

We cache Outside Data not Inside Data!

Reference Data does not know the rules!

Query by Event Carried State Transfer

The Good

Decoupled

Autonomous

Low Latency/Pre-Calculated

Fault Tolerant

Query by Event Carried State Transfer

The Bad

Eventual Consistency

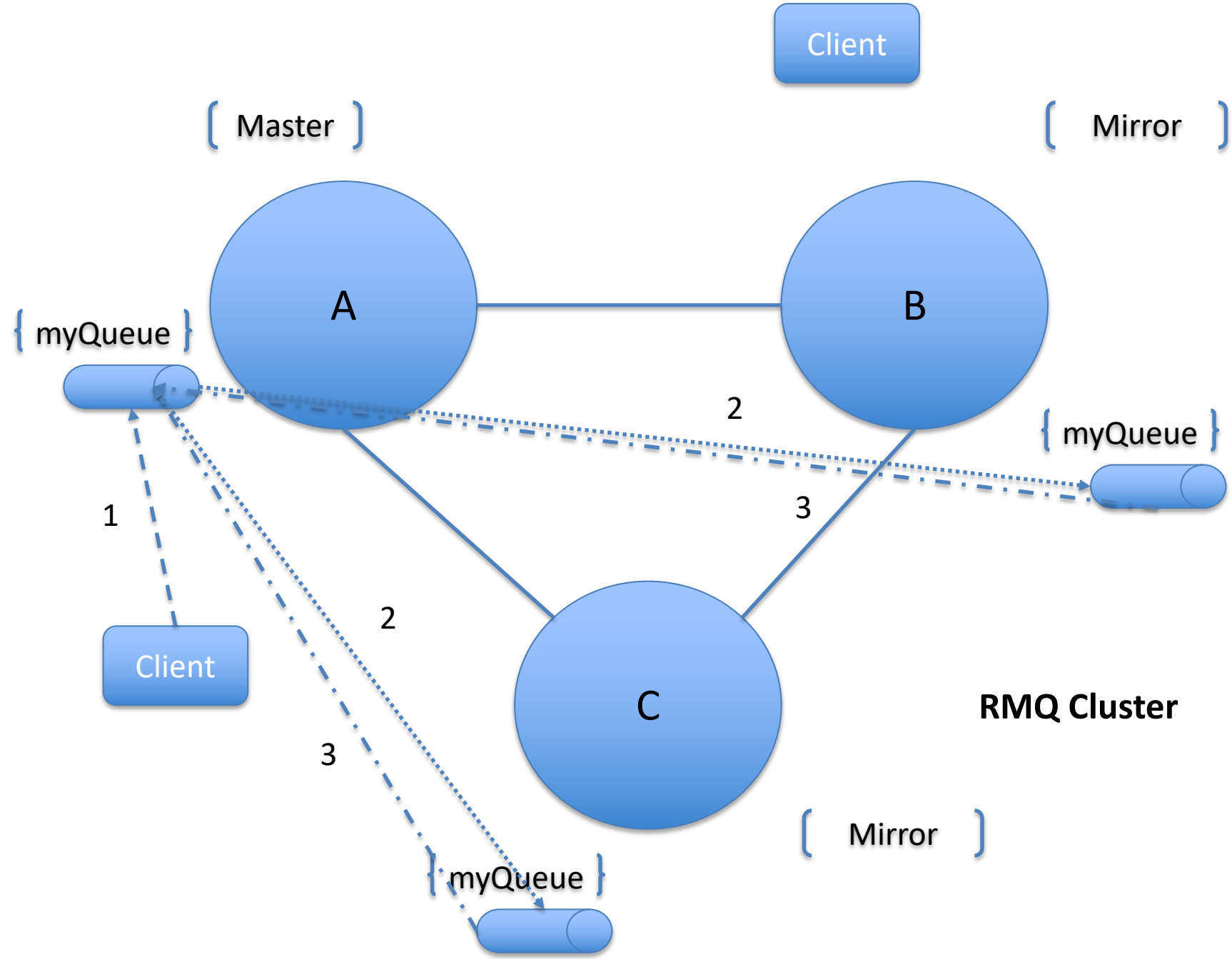
Replication of Data

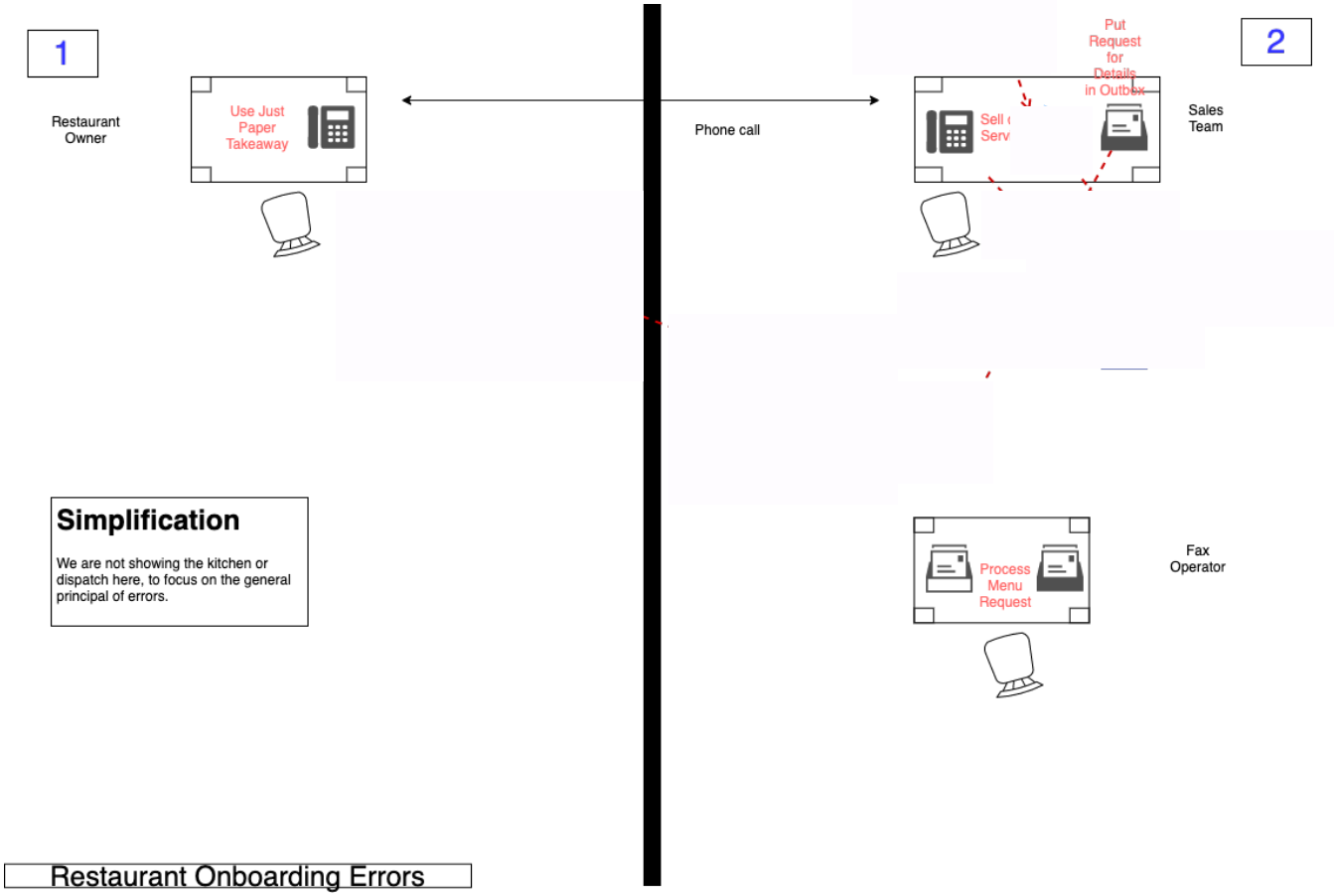
Synchronization

4. RELIABILITY

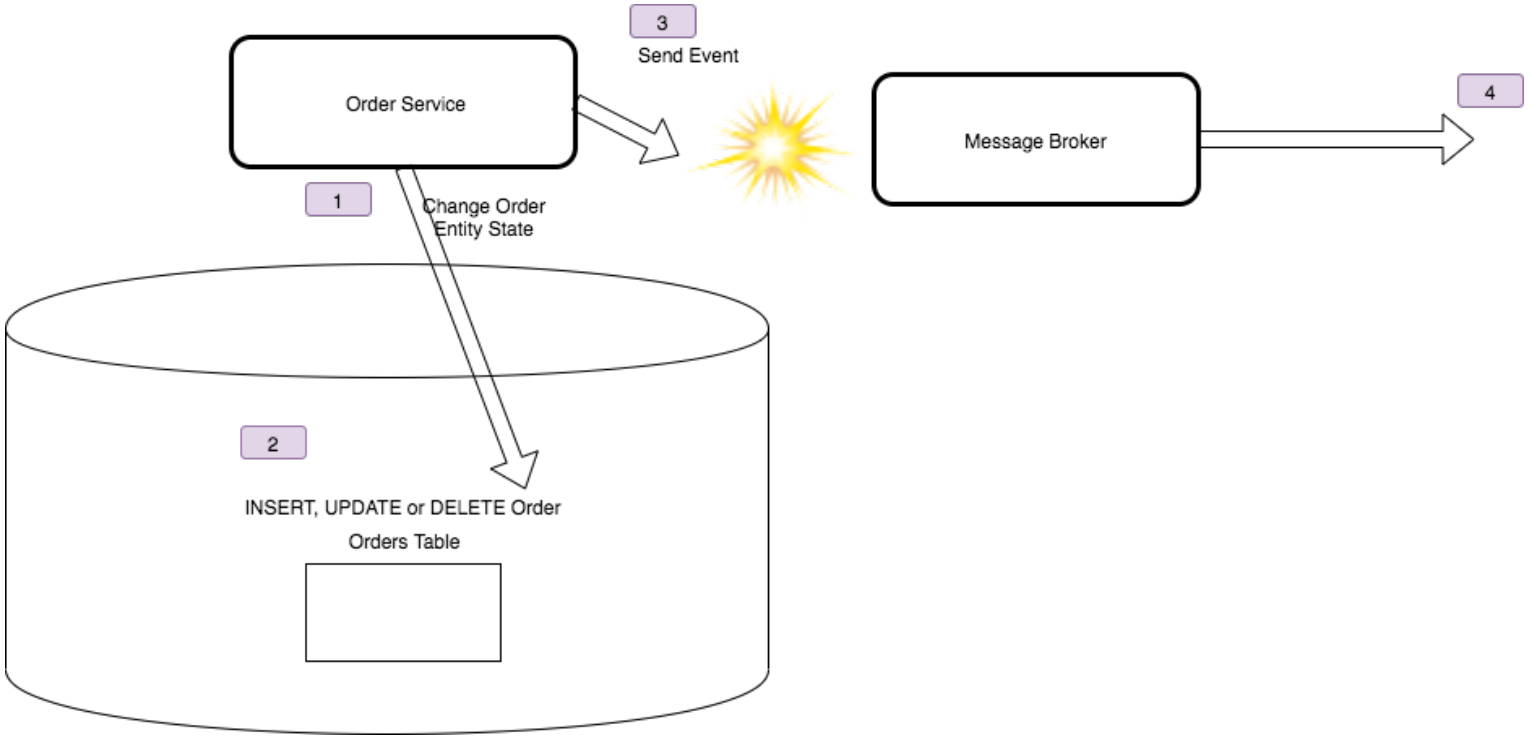
At Least Once

4.1 GUARANTEED DELIVERY

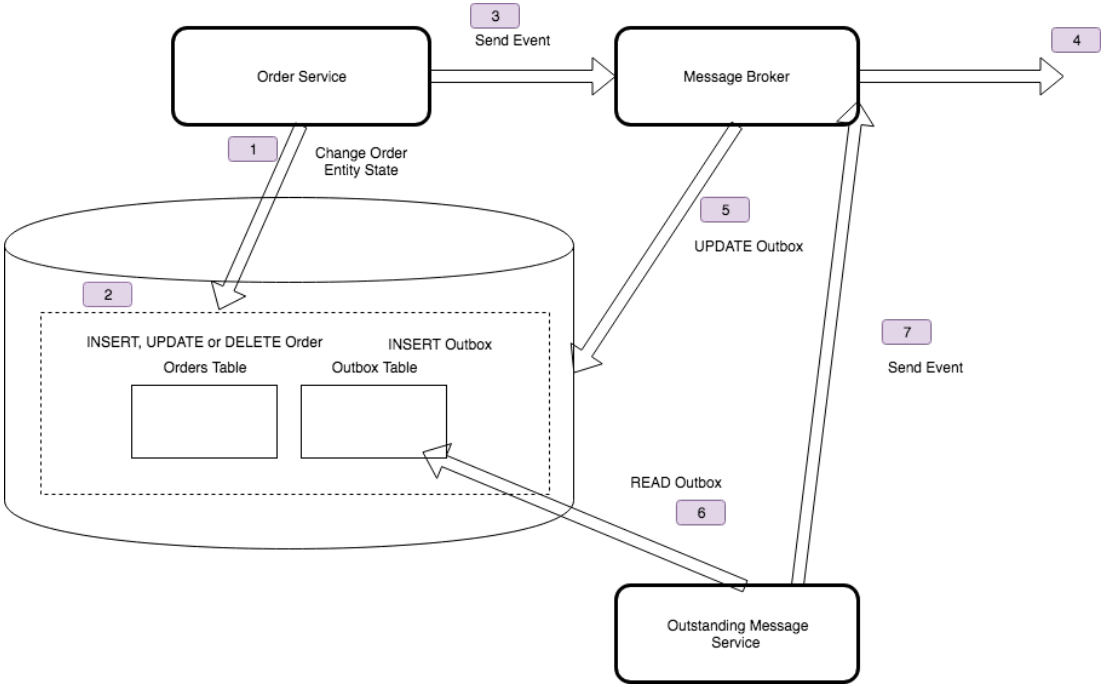

Client
Master
Mirror
A
B
myQueue
2
myQueue
1
3
2
Client
C
RMQ Cluster
3
Mirror
myQueue


1
Restaurant Owner
Use Just Paper Takeaway
Phone call
Put Request for Details in Outbox
2
Sales Team
Simplification
We are not showing the kitchen or dispatch here, to focus on the general principal of errors.
Process Menu Request
Fax Operator
Restaurant Onboarding Errors

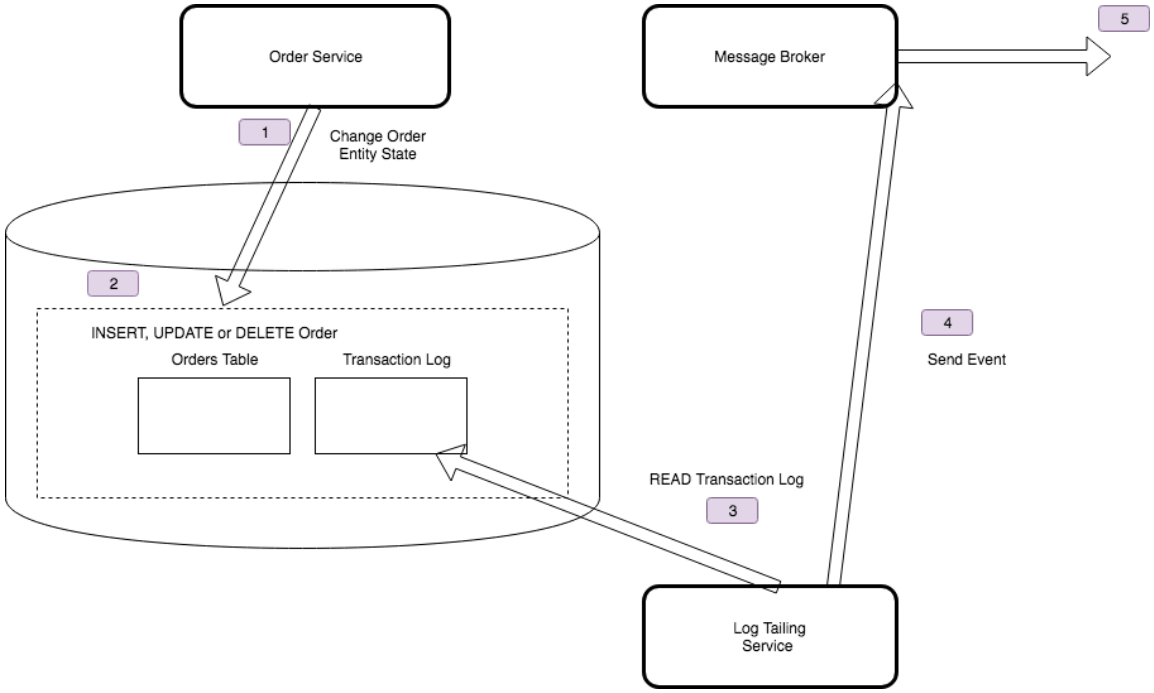
Correctness Problem



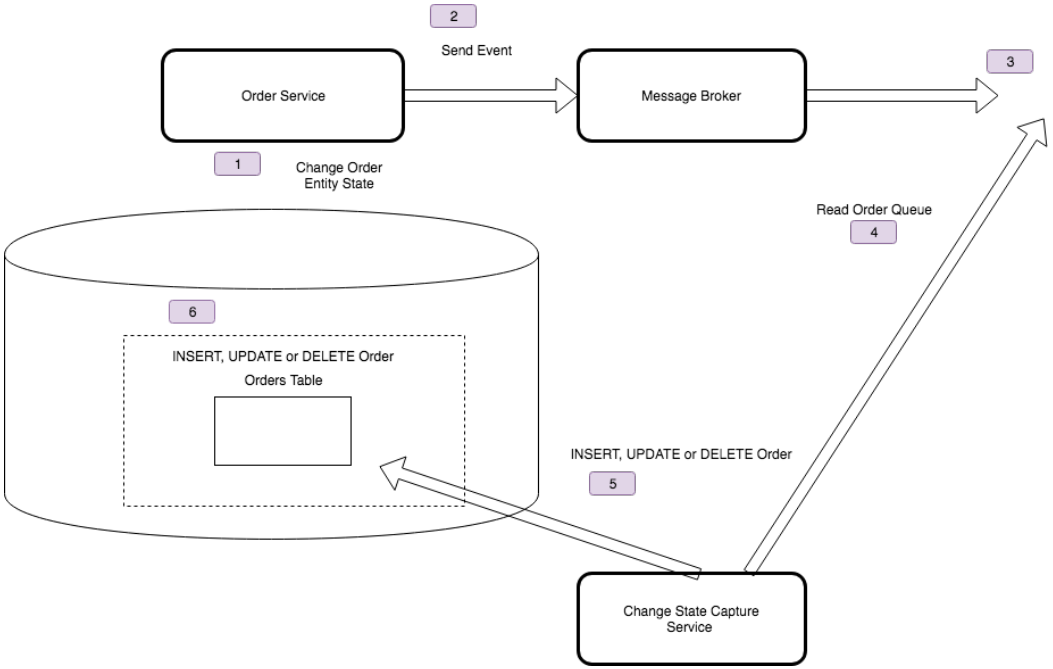
Outbox Pattern



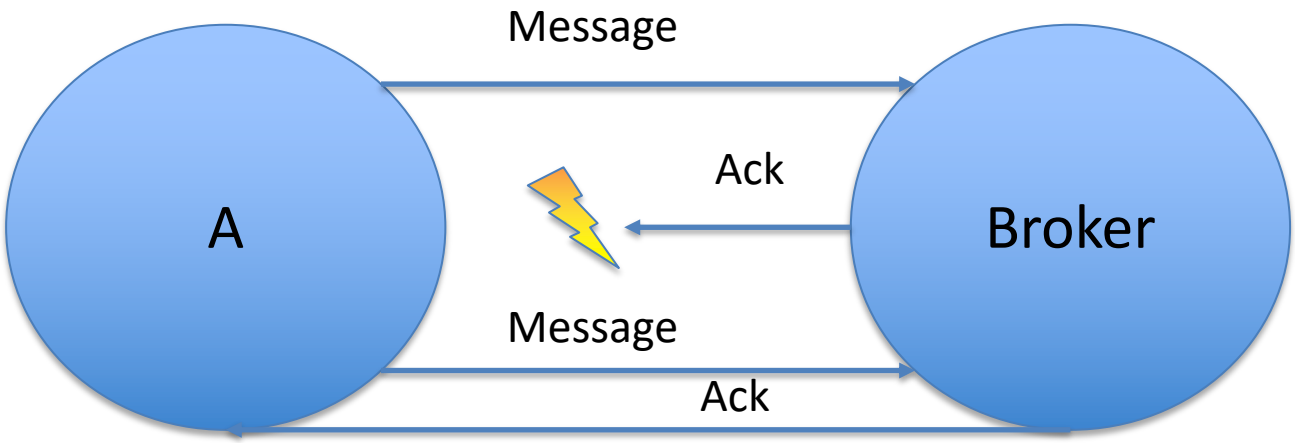
Log Tailing

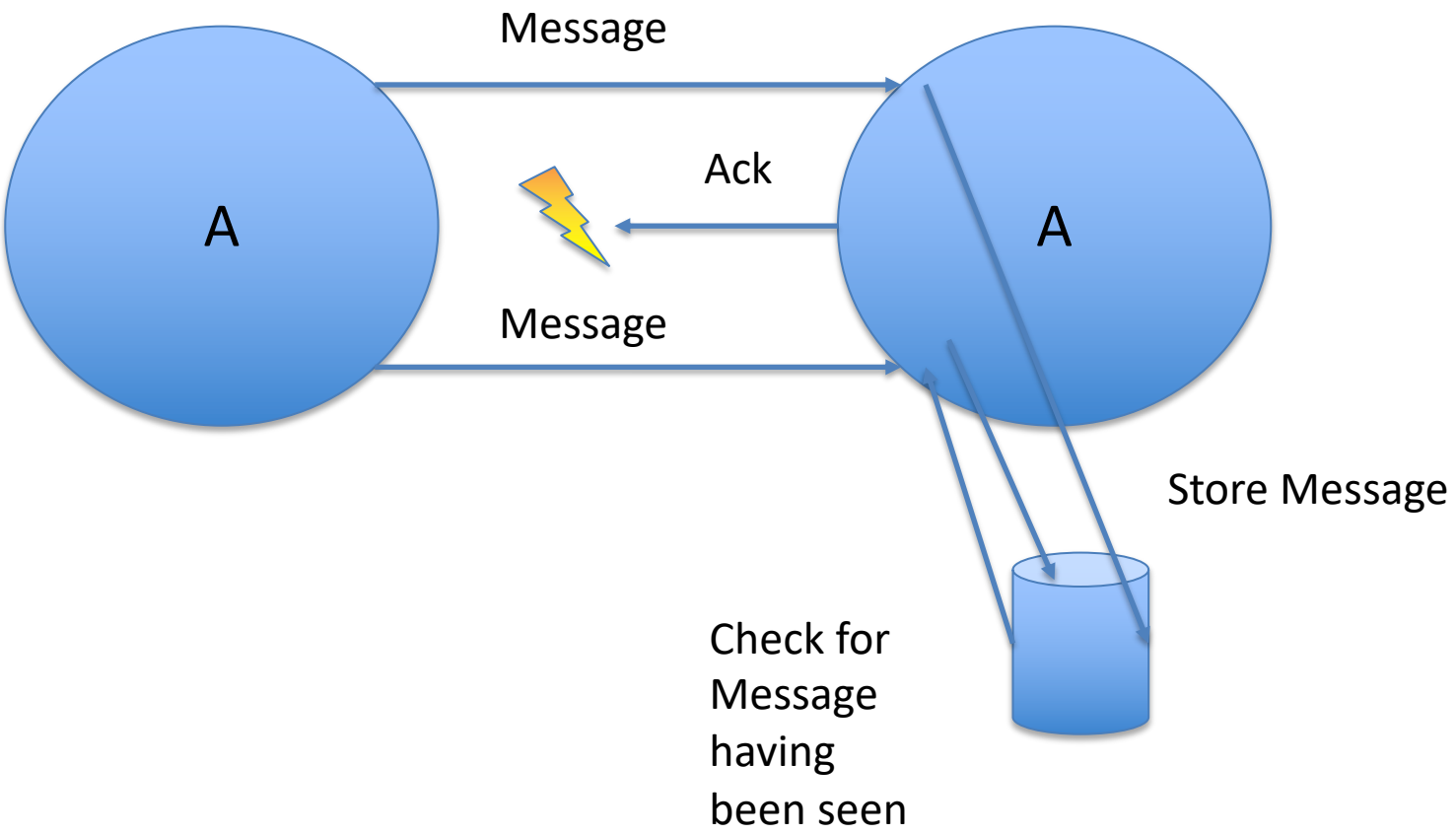

Order Service
Message Broker
5
1
Change Order
Entity State
2
INSERT, UPDATE or DELETE Order
Orders Table
Transaction Log
4
Send Event
READ Transaction Log
3
Log Tailing
Service

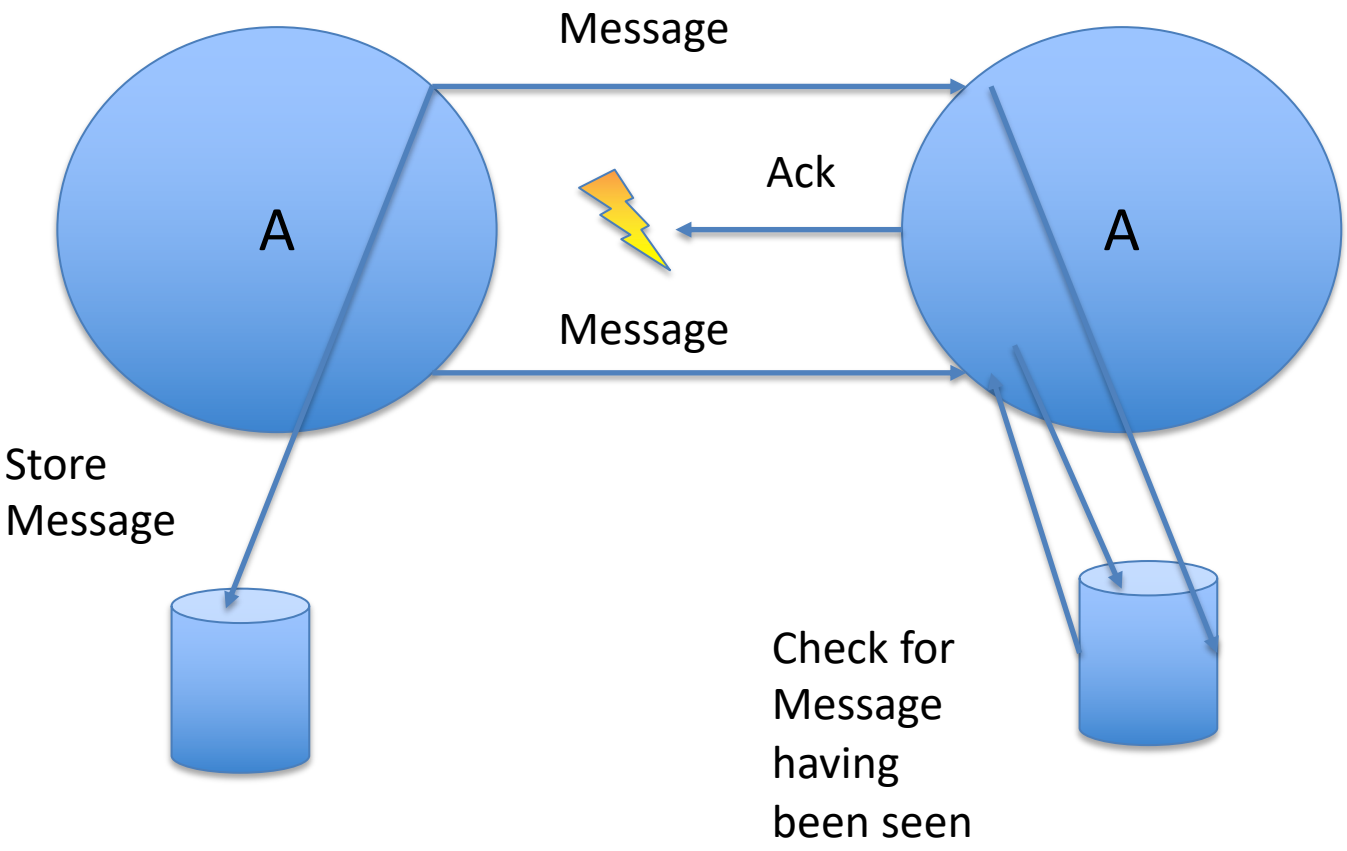
State Change Capture



4.2 AT LEAST ONCE, EXACTLY ONCE

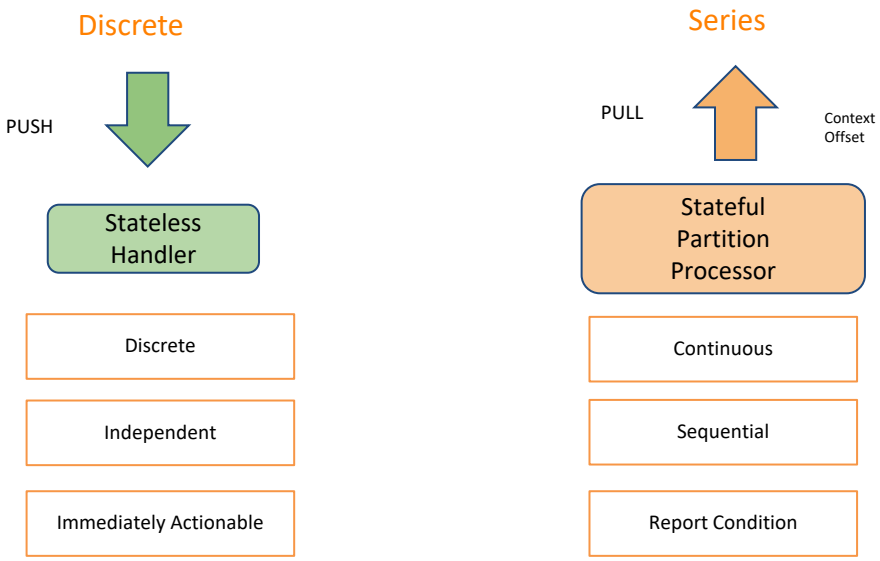

Message
A
Ack
Broker
Message
Ack


Message
A
Ack
A
Message
Store Message
Check for
Message
having
been seen


Message
Ack
Message
A
A
Store
Message
Check for
Message
having
been seen

4.3 SERIES AND DISCRETE

Event Types



After Clemens Vasters: https://skillsmatter.com/skillscasts/10191-keynote-events-data-points-jobs-and-commands-the-rise-of-messaging

See also: https://en.wikipedia.org/wiki/Discrete_time_and_continuous_time

(Message)Event – Skinny, Notification

Discrete, Immediately Actioned

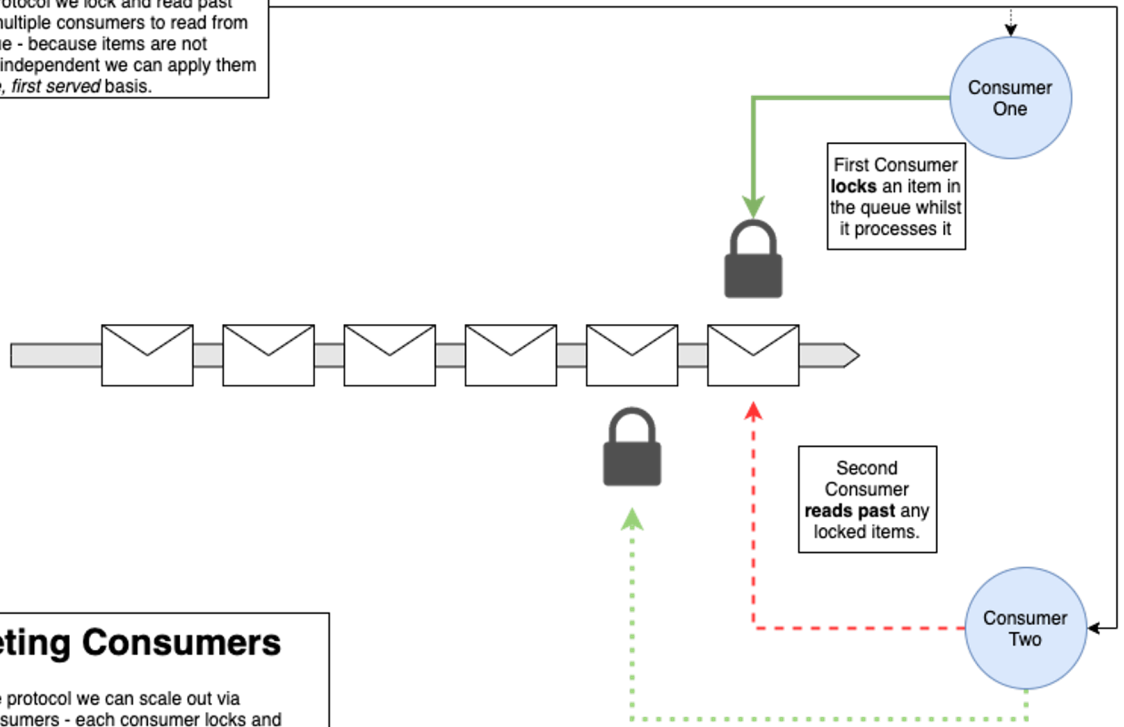
(Message)Document – Fat

Series, Supports Action

Discrete Events

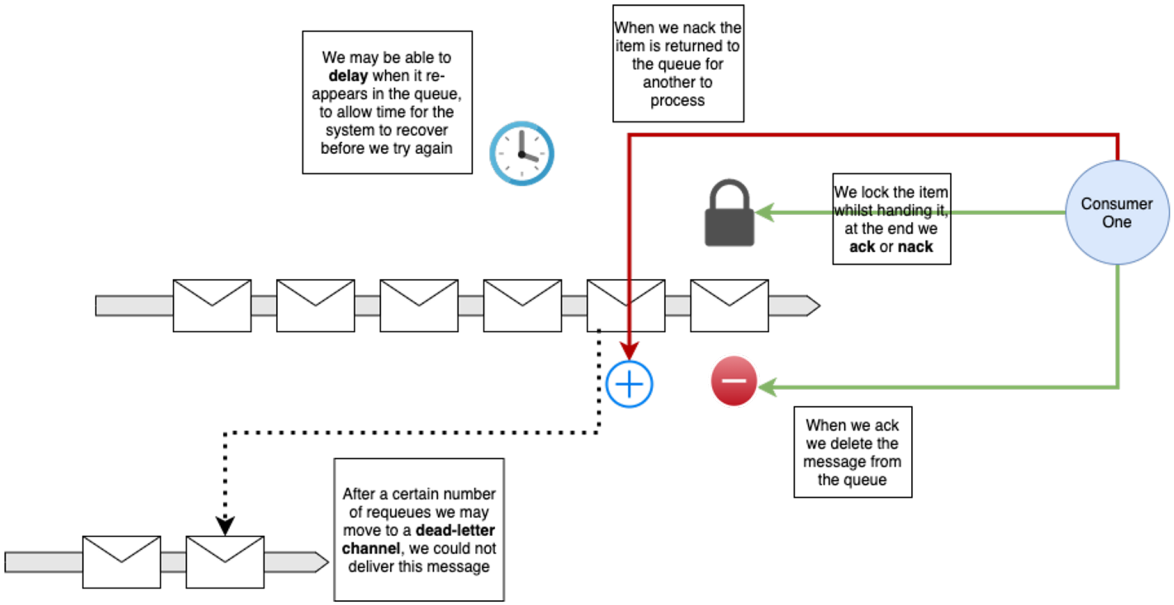
Discrete Protocols

In a discrete protocol we lock and read past which allows multiple consumers to read from the same queue - because items are not sequential but independent we can apply them on a *first come, first served* basis.



Competing Consumers

With a discrete protocol we can scale out via competing consumers - each consumer locks and processes the first available message



No Archive and Replay

With discrete messages we delete a message once it is consumed - it is a task that we want to do once - but that means we have no built in archive and replay. Replay depends on the producer, and in a pub-sub model would require broadcast to all subscribers who need to discard duplicates unless we can send directly to one consumer.

Discrete Messaging as Tasks

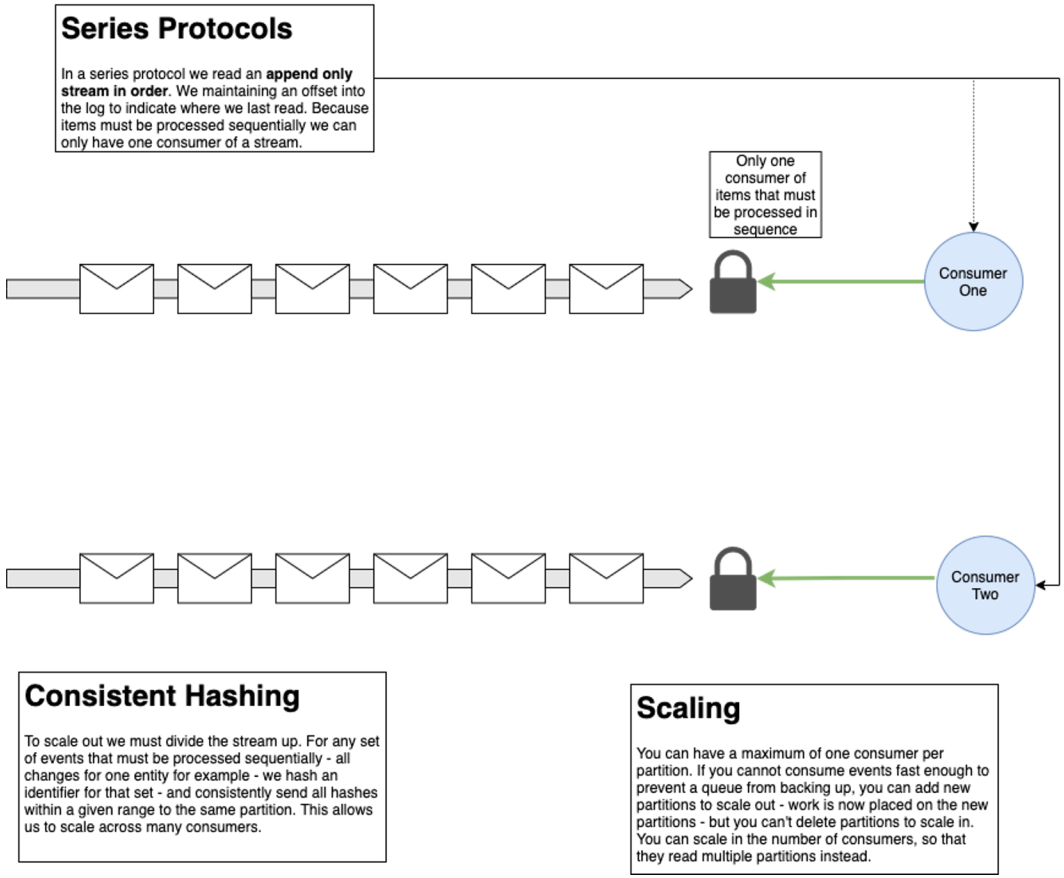
With discrete messages we can think of information packets that are tasks - we do something in response to receiving them. Once the task is done we delete the task,

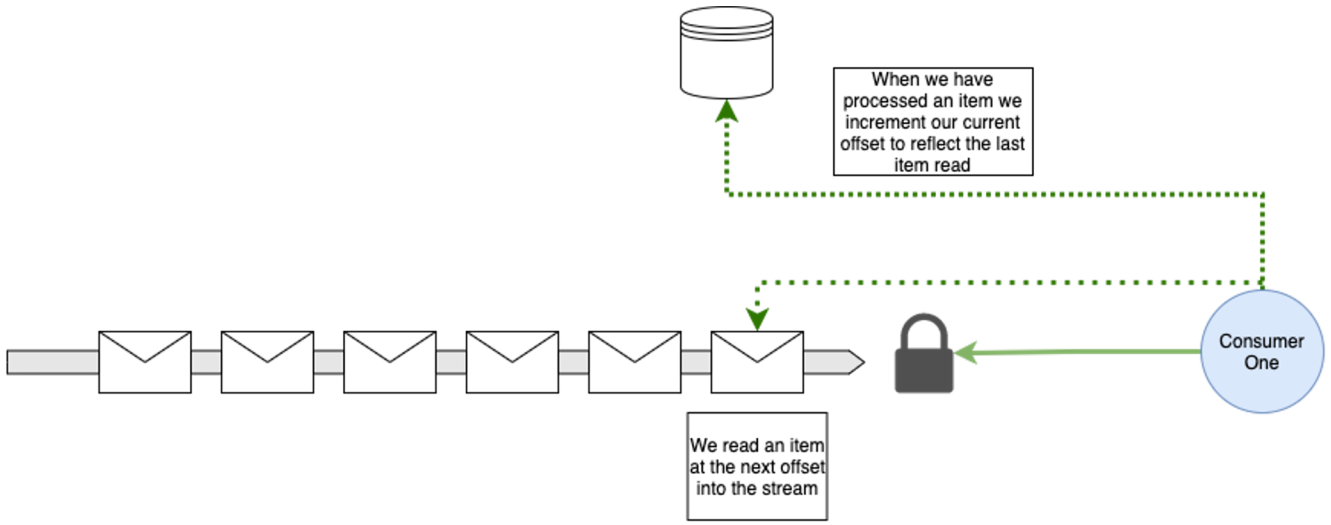
+ We don't want anyone else to attempt it, it's already done.

+ An originator should re-raise the task if it should be repeated.

+ If we can't process it but it is processable, we can nack, so that someone else can attempt it

Series Events





No Replay or DLQ

Out-of-the-box series brokers do not support requeue or dead letter queues (DLQs). This is because they are not processing tasks, that should be done, or failure reported, but consuming a stream to report a condition on. As such there is no need to requeue a task or send it to a DLQ.

It is possible to copy an event to another topic, to fake a requeue or DLQ and try to force a series broker to behave like a discrete broker but this tends to be bespoke code..

Series Messaging as Streams

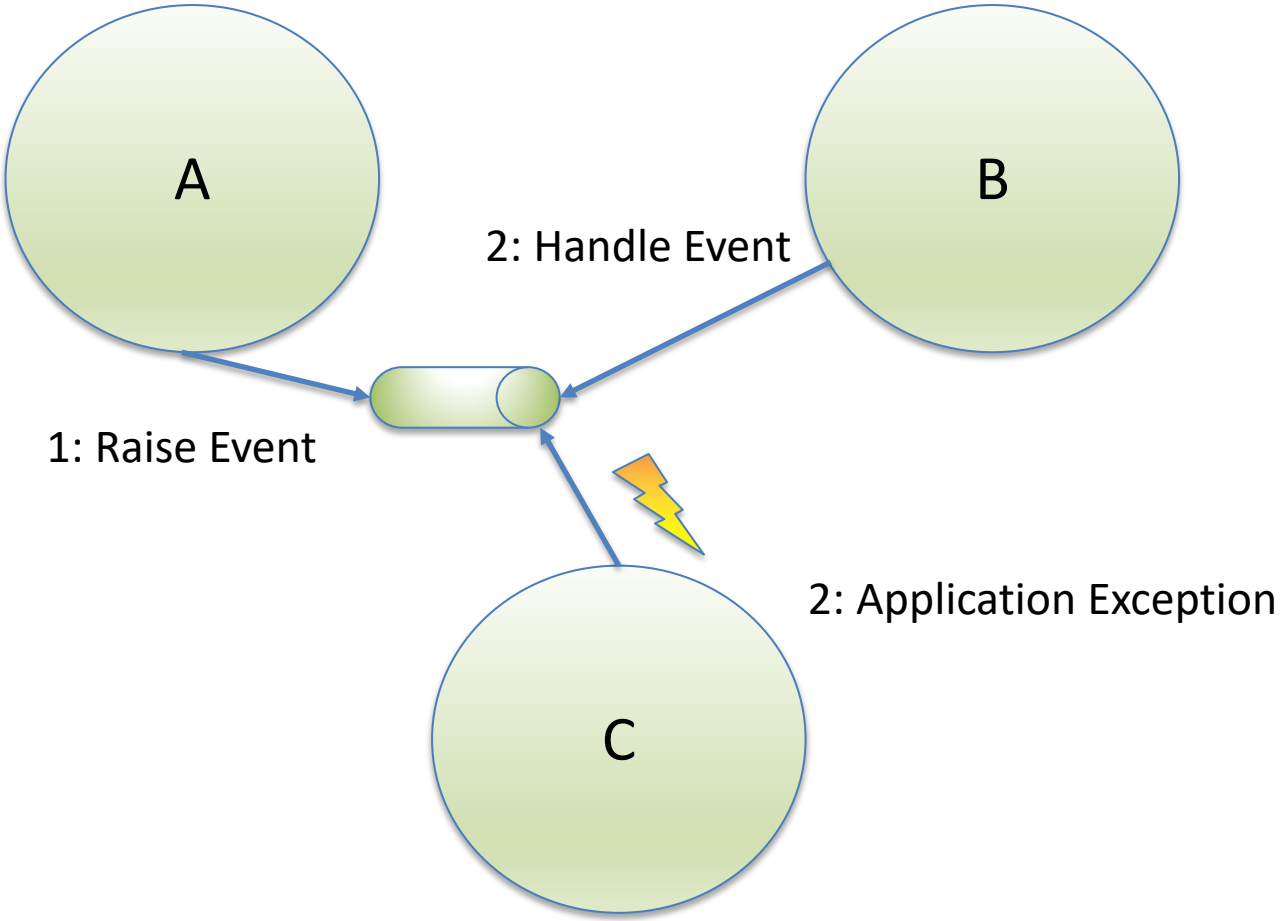
With series messages, the stream is append-only. We do not ack or nack, instead we increment our offset into the stream.

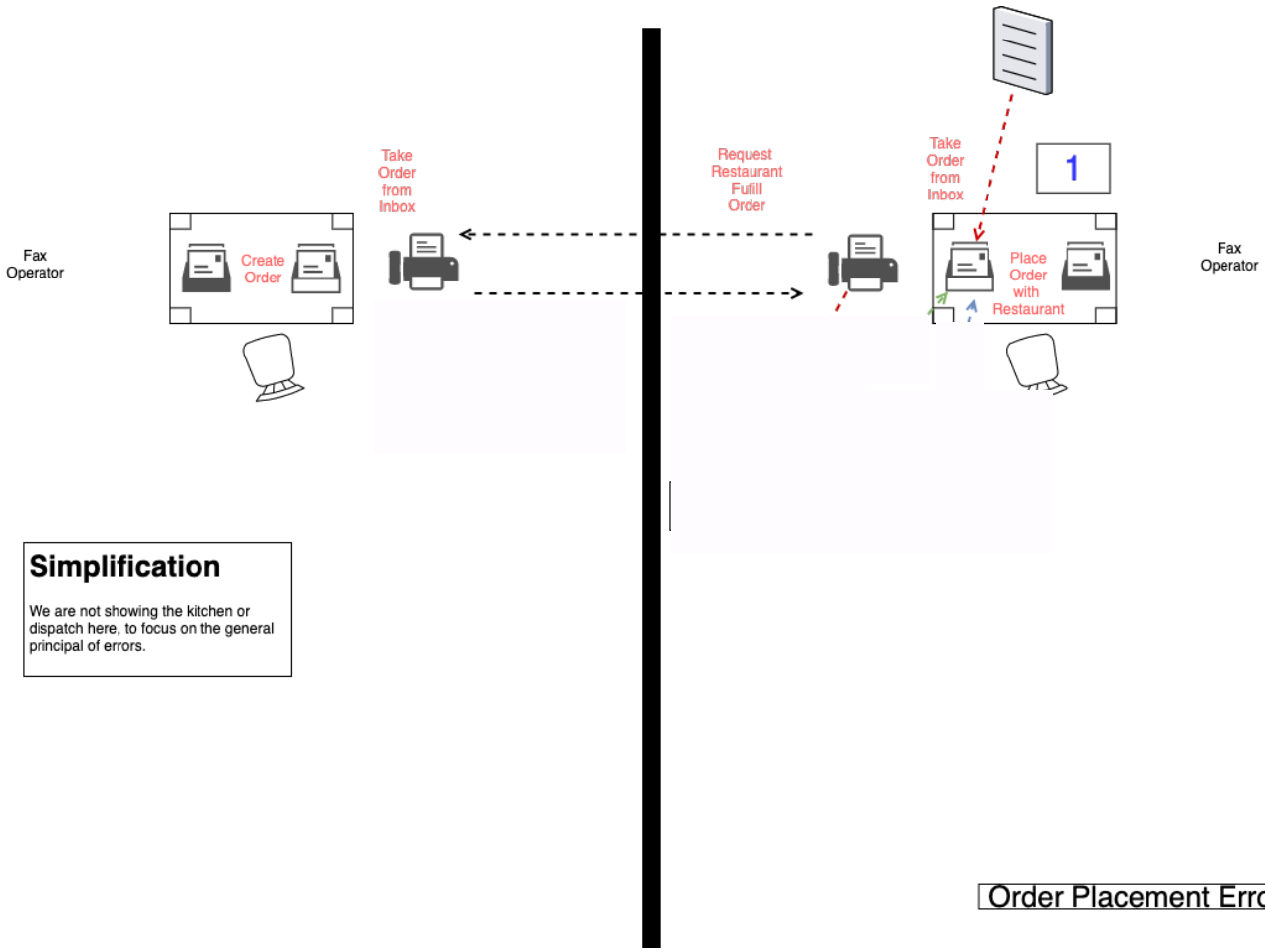
+ We can replay the stream if we want to.

+ The stream can be used for complex event processing i.e. create a table over the stream for a given window, often called inverting the database.

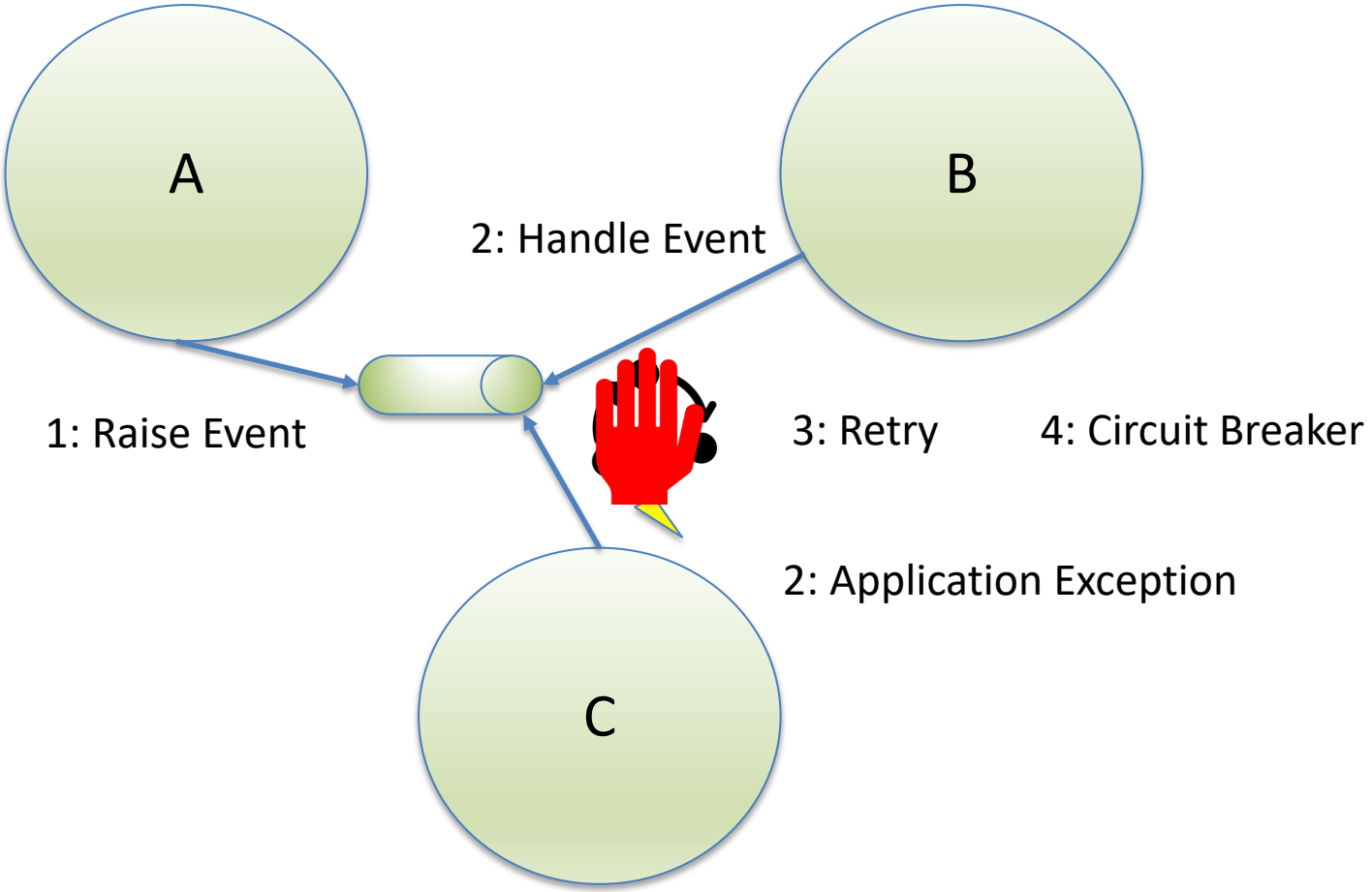
4.4 COMPENSATION

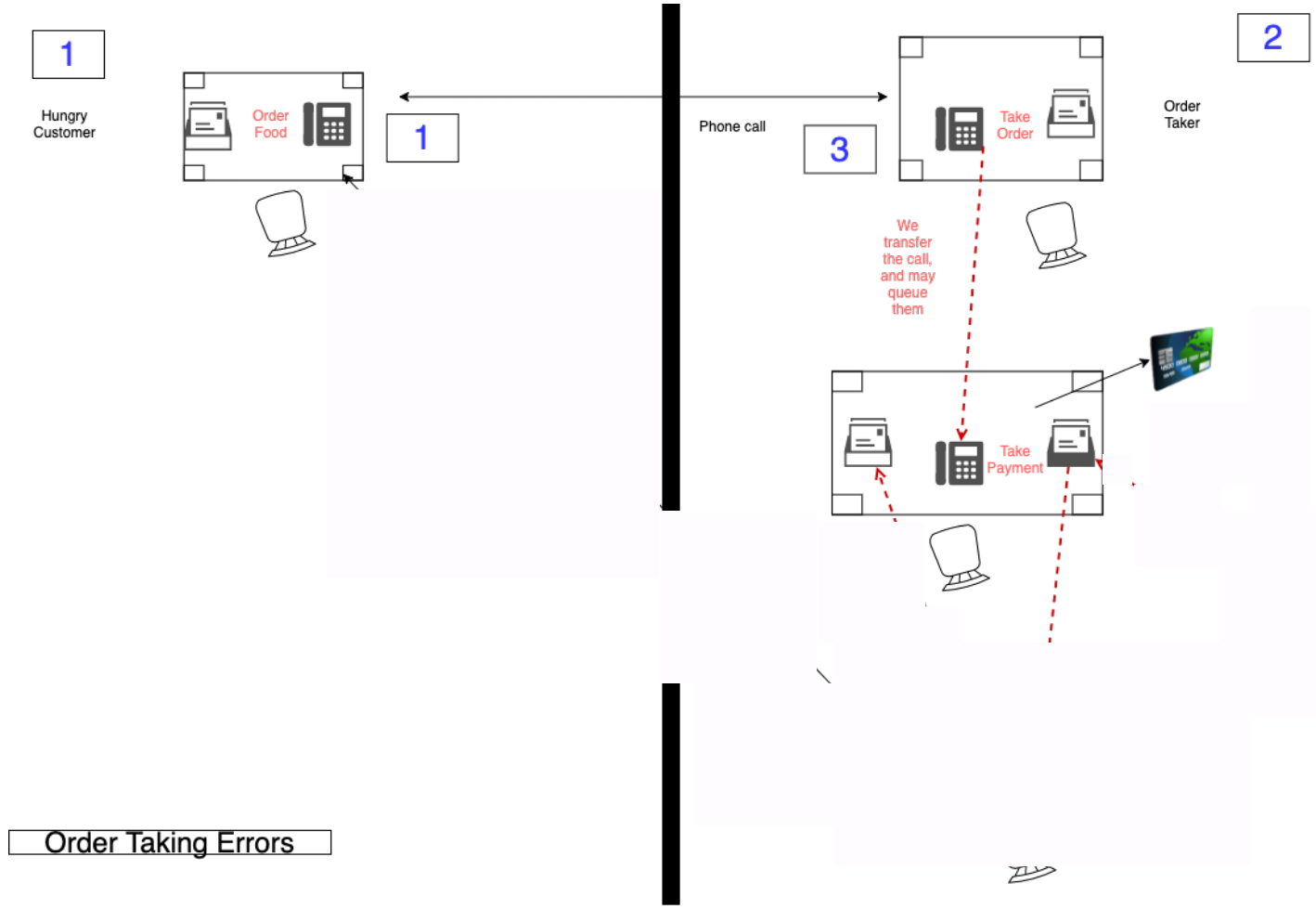
Compensation: Write Off



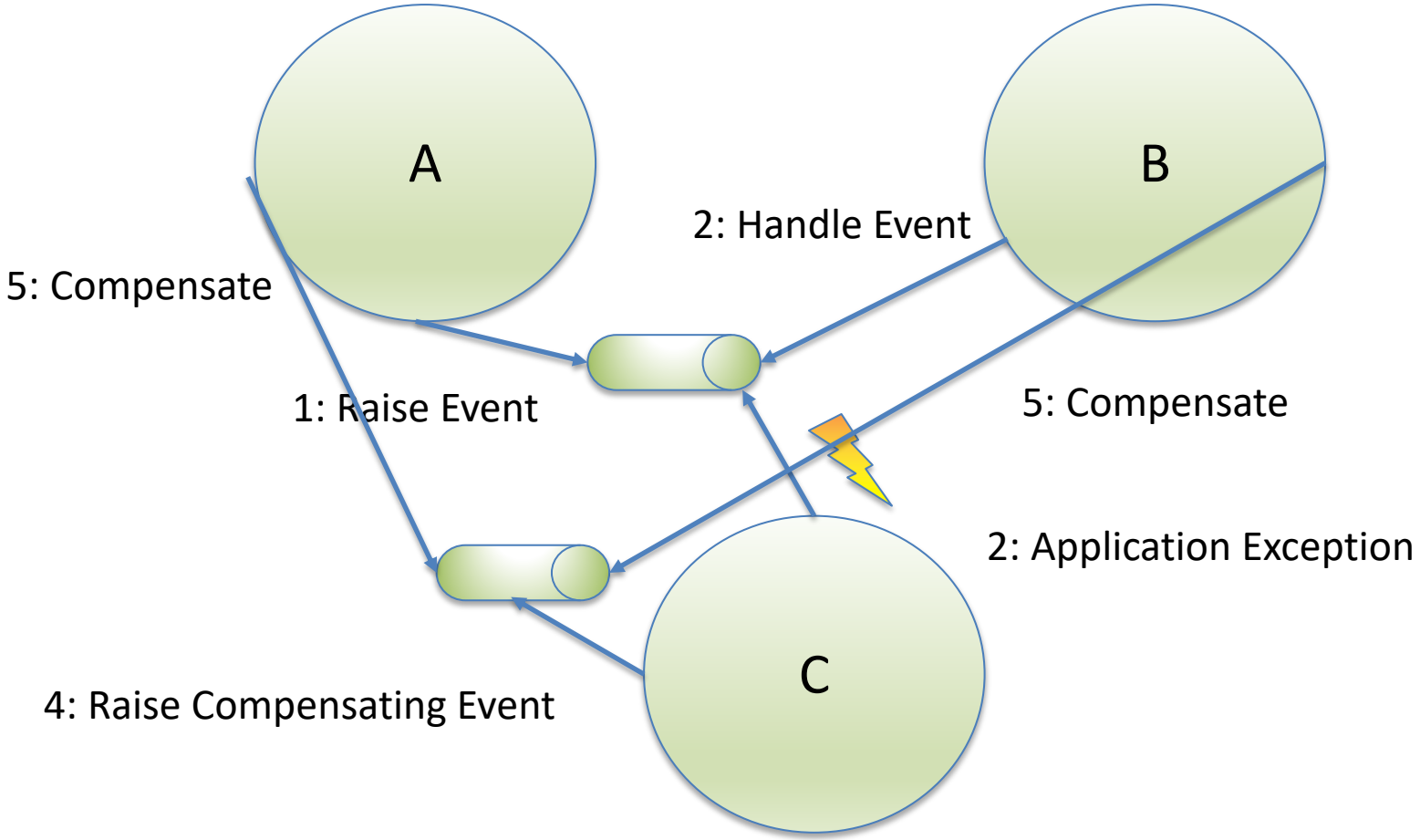

Fax
Operator
Create
Order
Take
Order
from
Inbox
Request
Restaurant
Fufill
Order
Take
Order
from
Inbox
1
Place
Order
with
Restaurant
Fax
Operator
Simplification
We are not showing the kitchen or dispatch here, to focus on the general principal of errors.
Order Placement Errors

Compensation: Retry

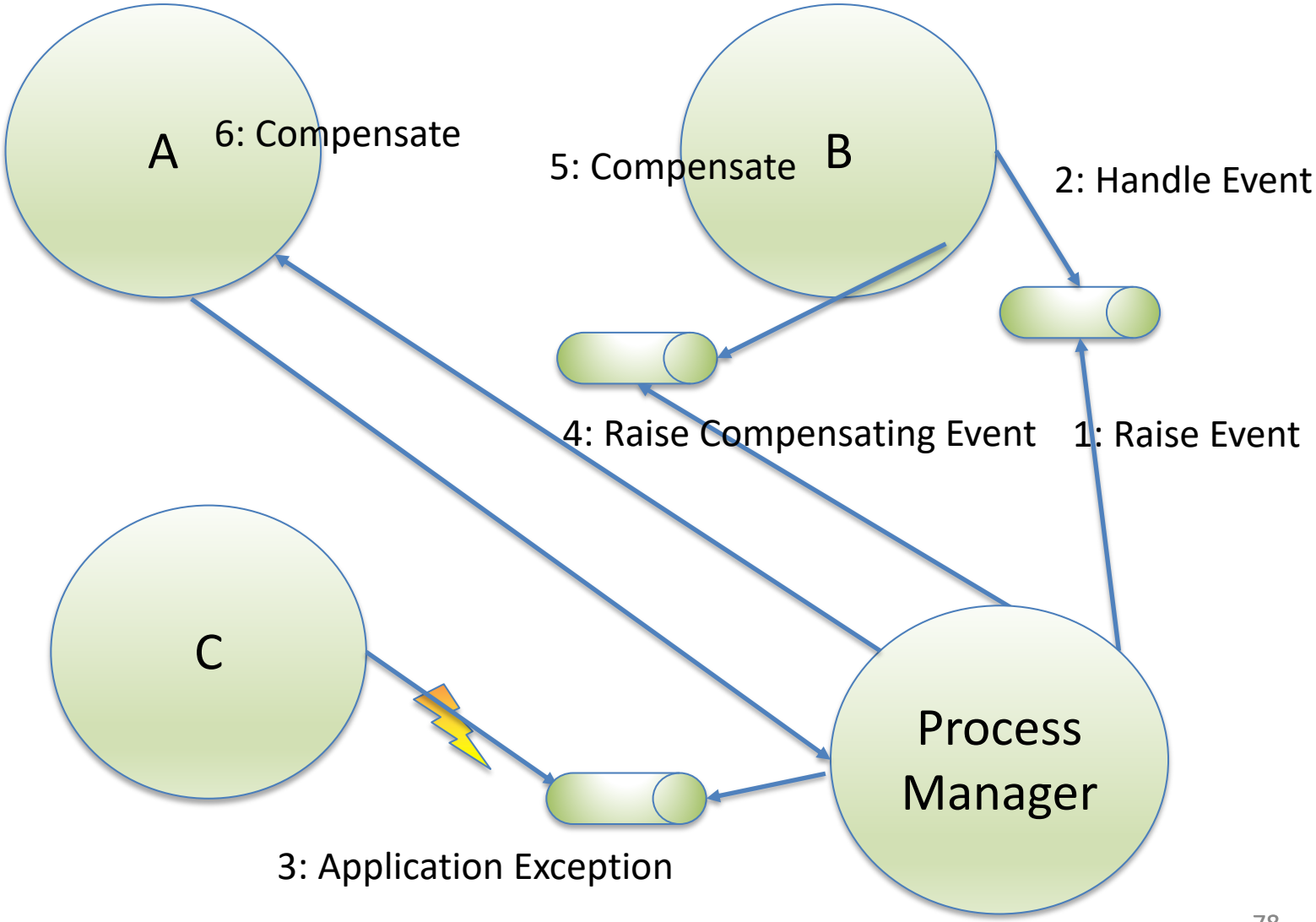



1
Hungry Customer
Order Food
1
Phone call
3
2
Take Order
Order Taker
We transfer the call, and may queue them
Take Payment
Order Taking Errors

Compensation: Compensating Event



Compensation: Process Manager

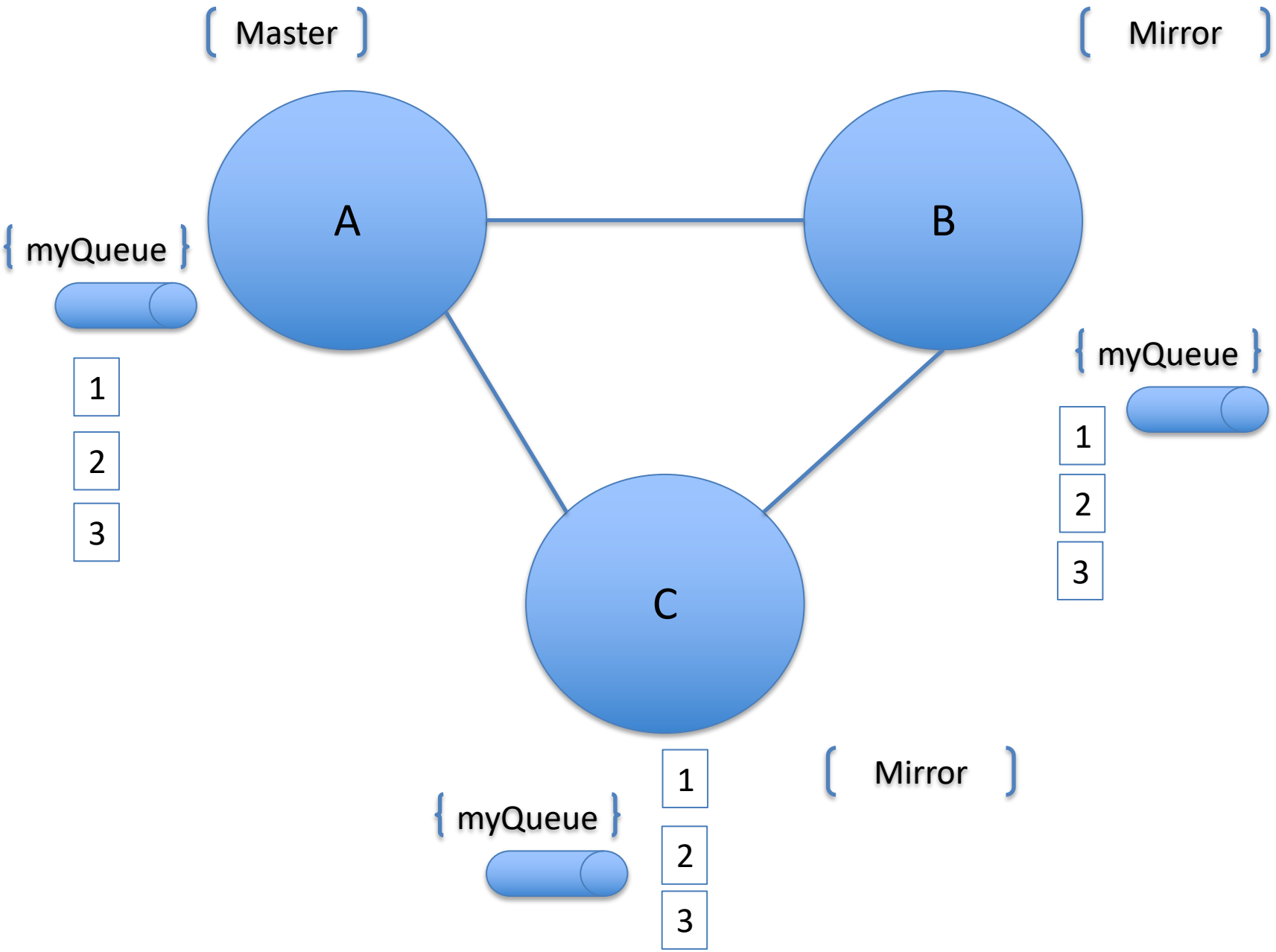


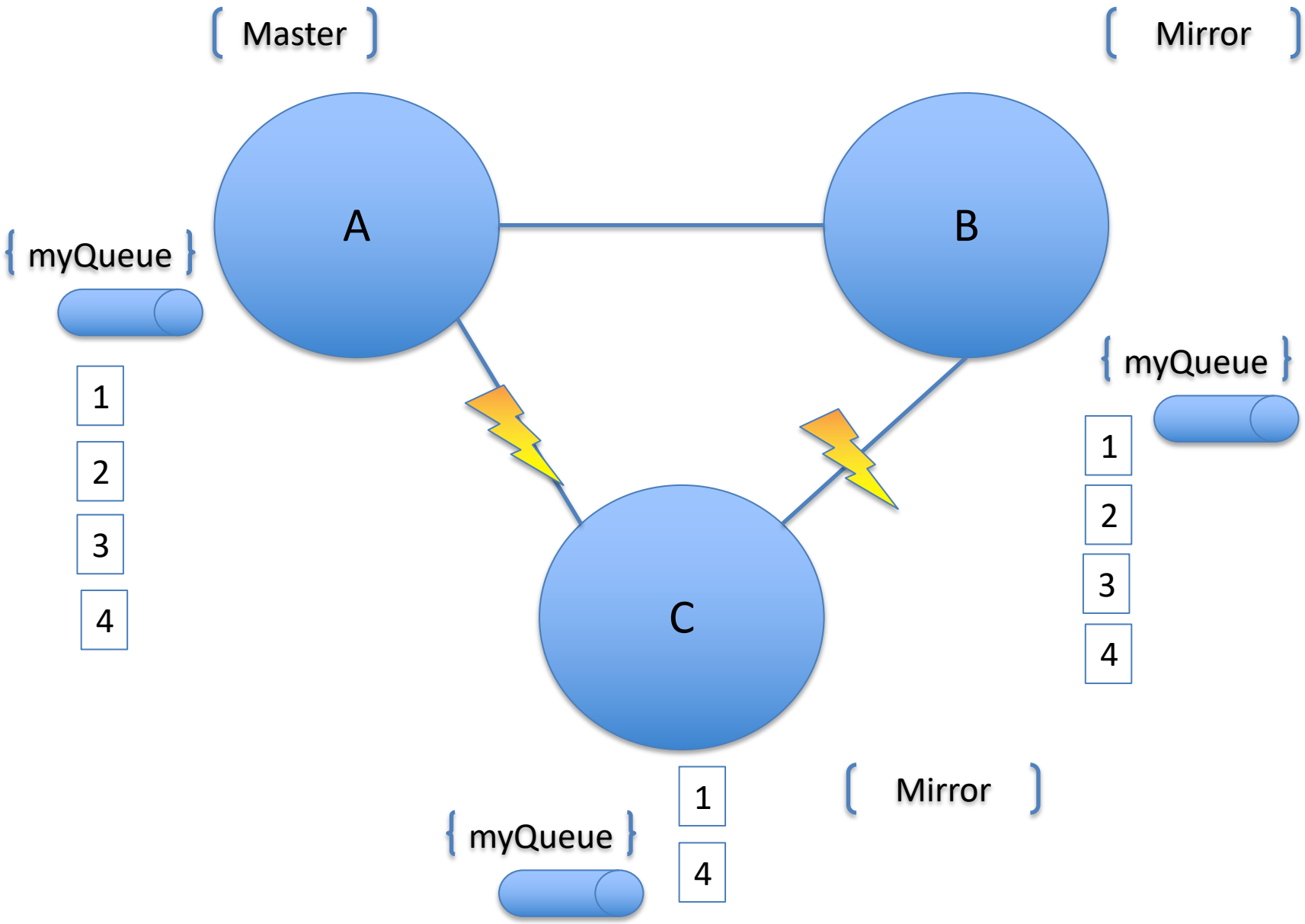

„Starbucks does not use
Two-Phase Commit“
http://www.enterpriseintegrationpatterns.com/ramblings/18_starbucks.html

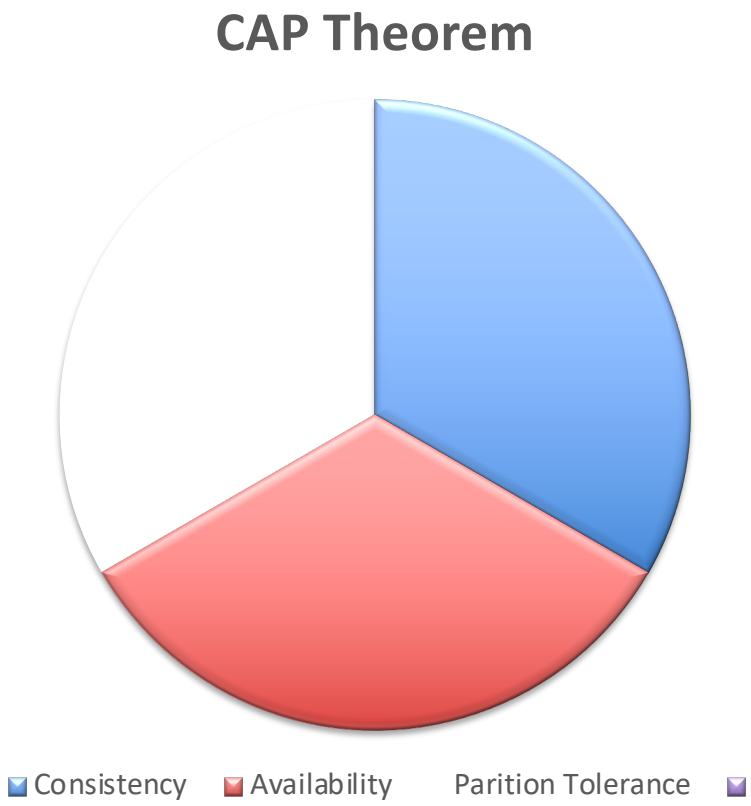
4.5 CAP THEOREM

CAP Theorem

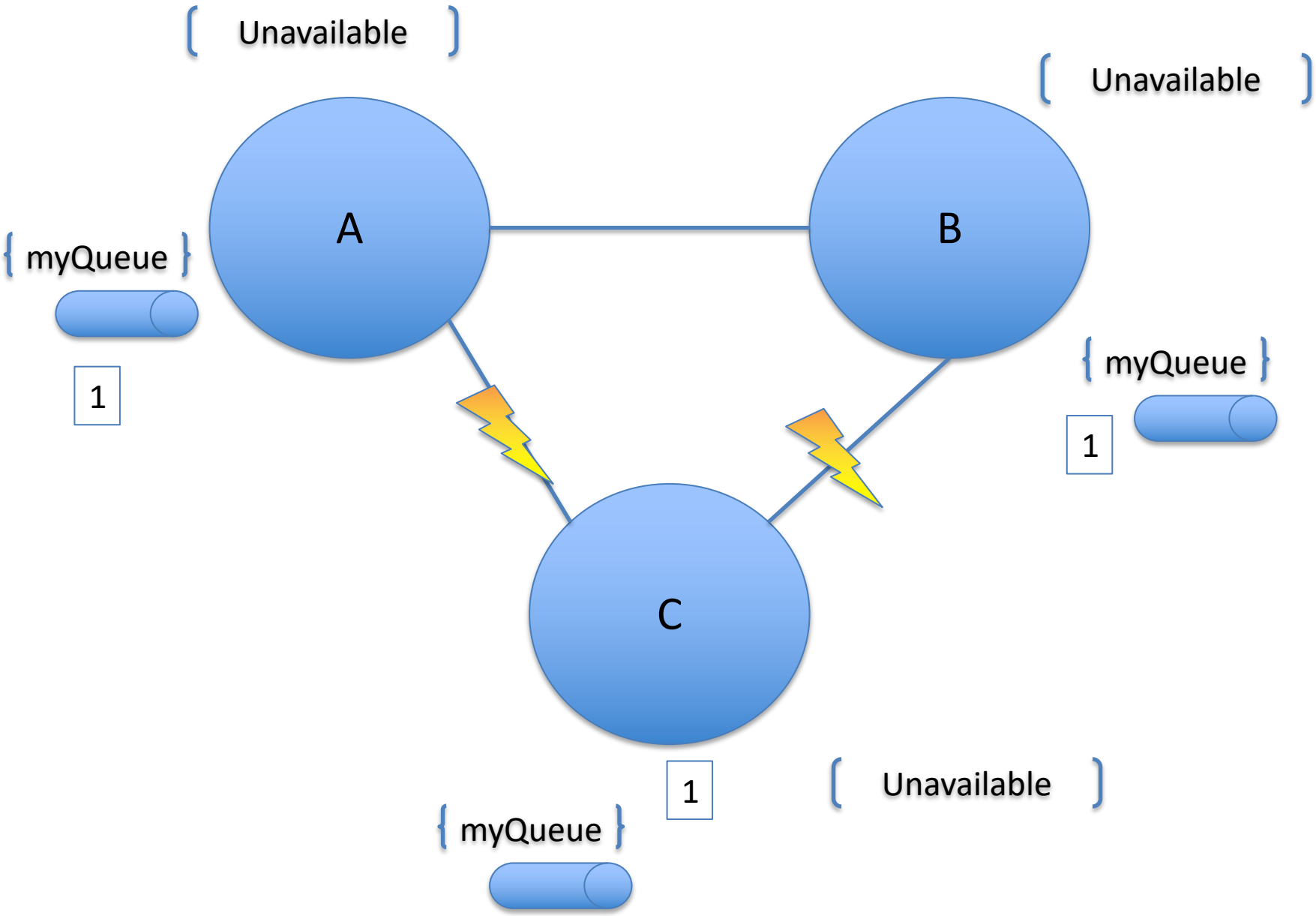
You can have at most two of these properties for any shared data system


Master
Mirror
A
B
myQueue
1
2
3
myQueue
1
2
3
C
Mirror
myQueue
1
2
3


Master
Mirror
A
B
myQueue
1
2
3
4
myQueue
1
2
3
4
C
1
4
myQueue
Mirror

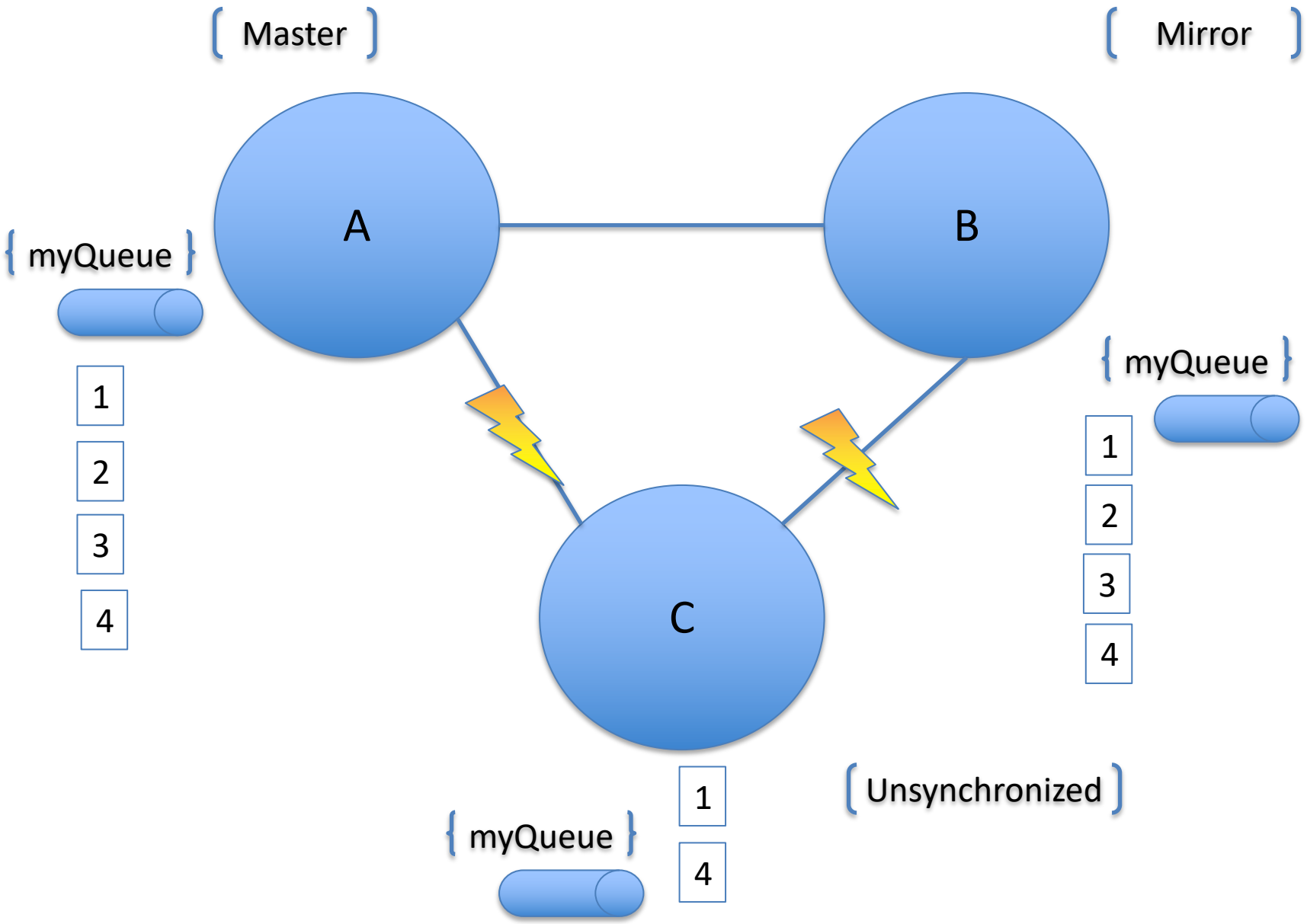


Forfeit Partitions

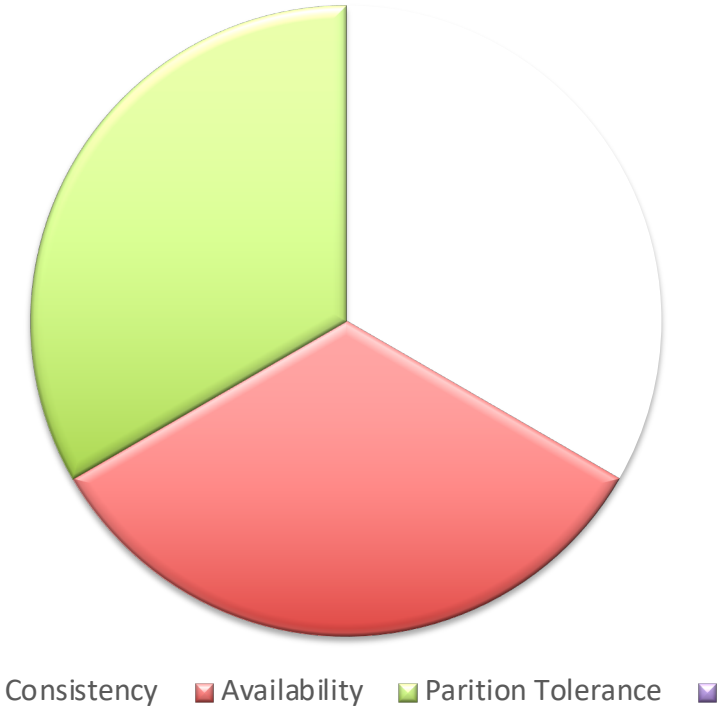

Unavailable
Unavailable
A
B
myQueue
1
myQueue
1
C
1
Unavailable
myQueue

CAP Theorem

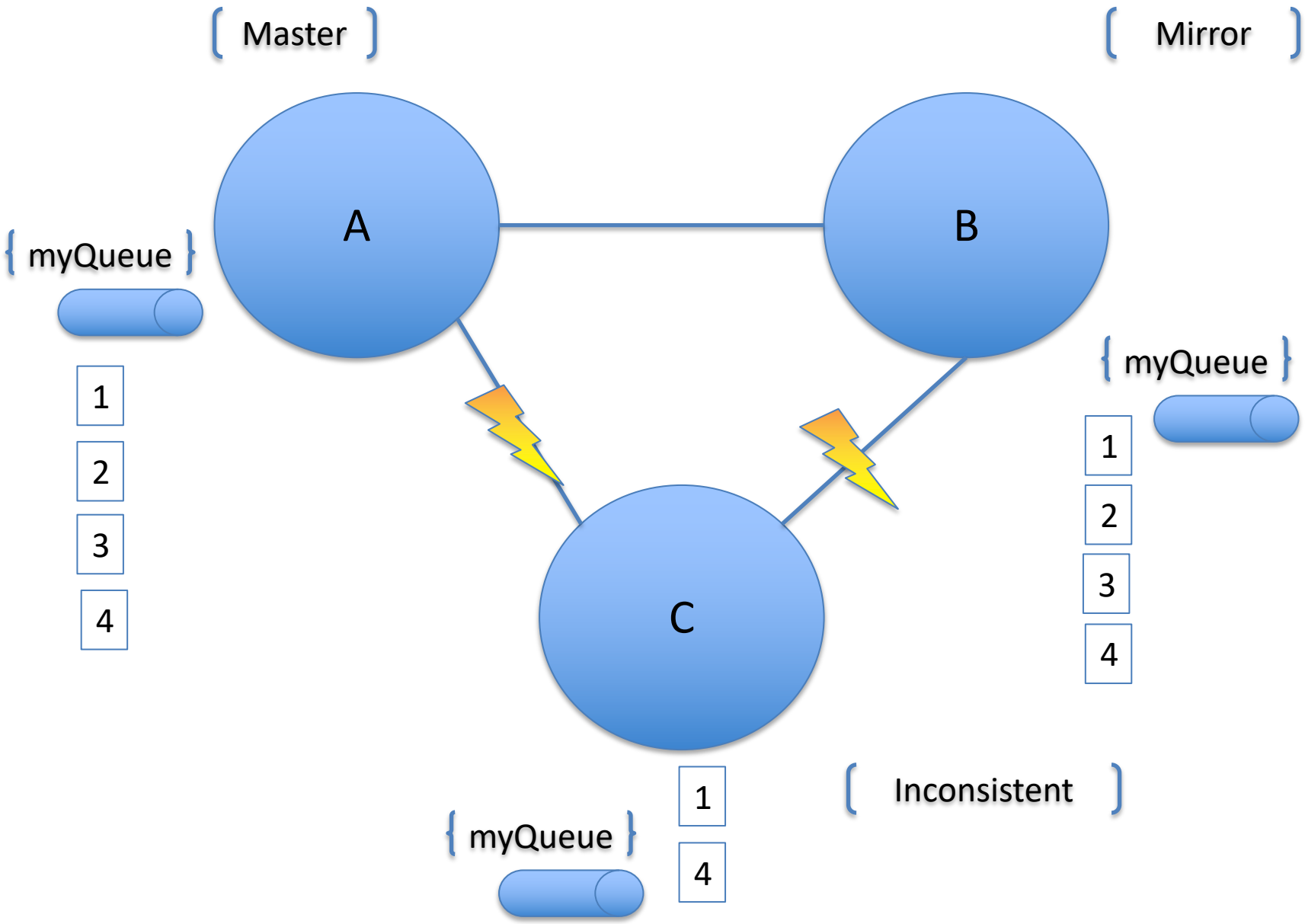
Forfeit Availability

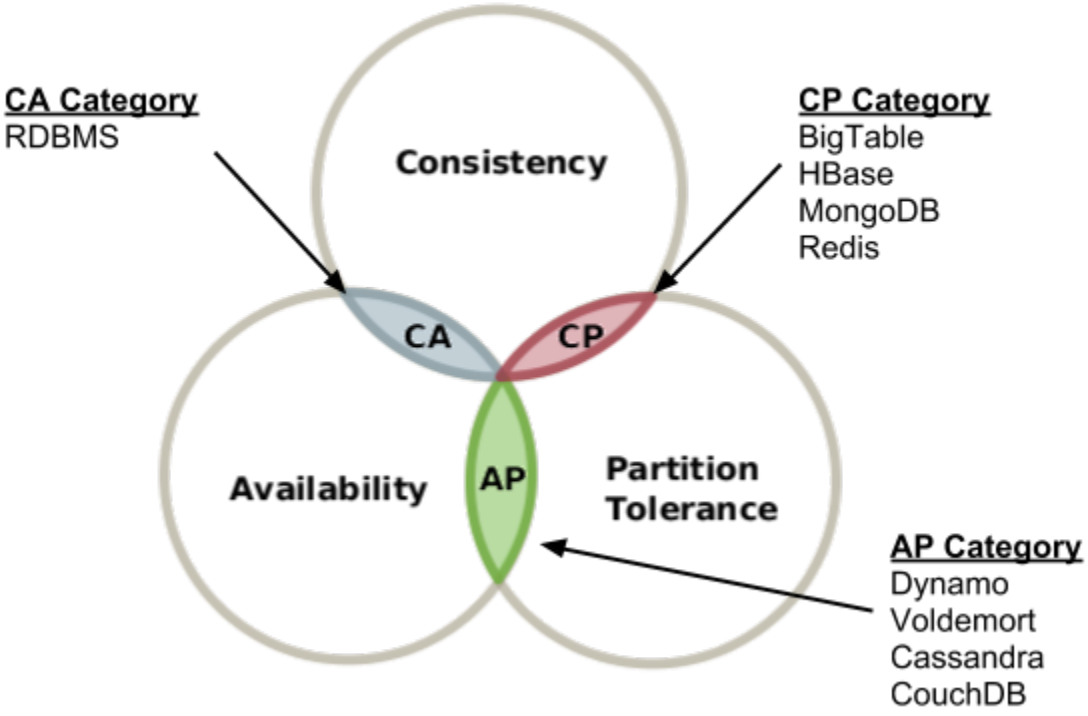

Master
Mirror
A
B
C
myQueue
1
2
3
4
myQueue
1
2
3
4
myQueue
1
4
Unsynchronized

CAP Theorem

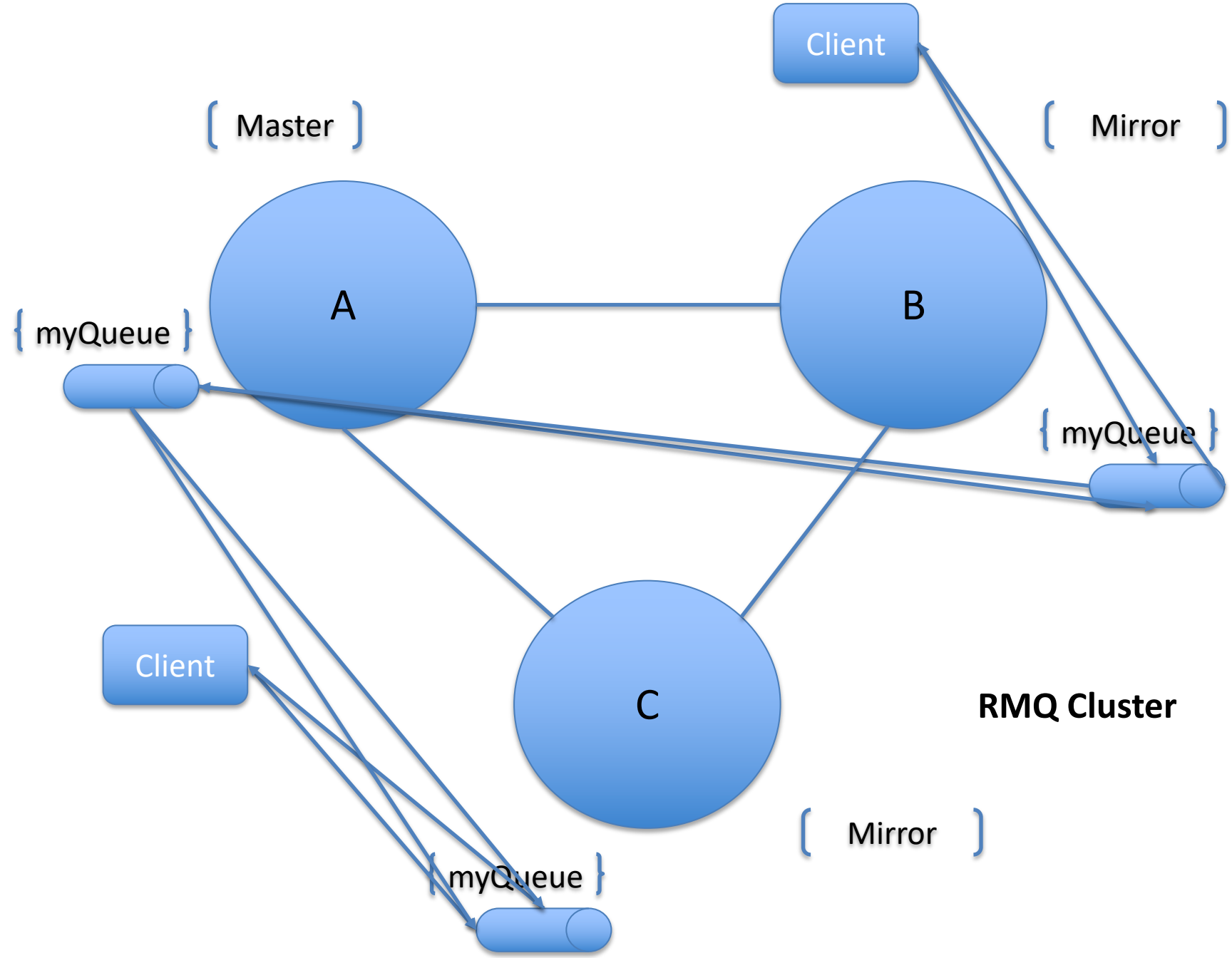


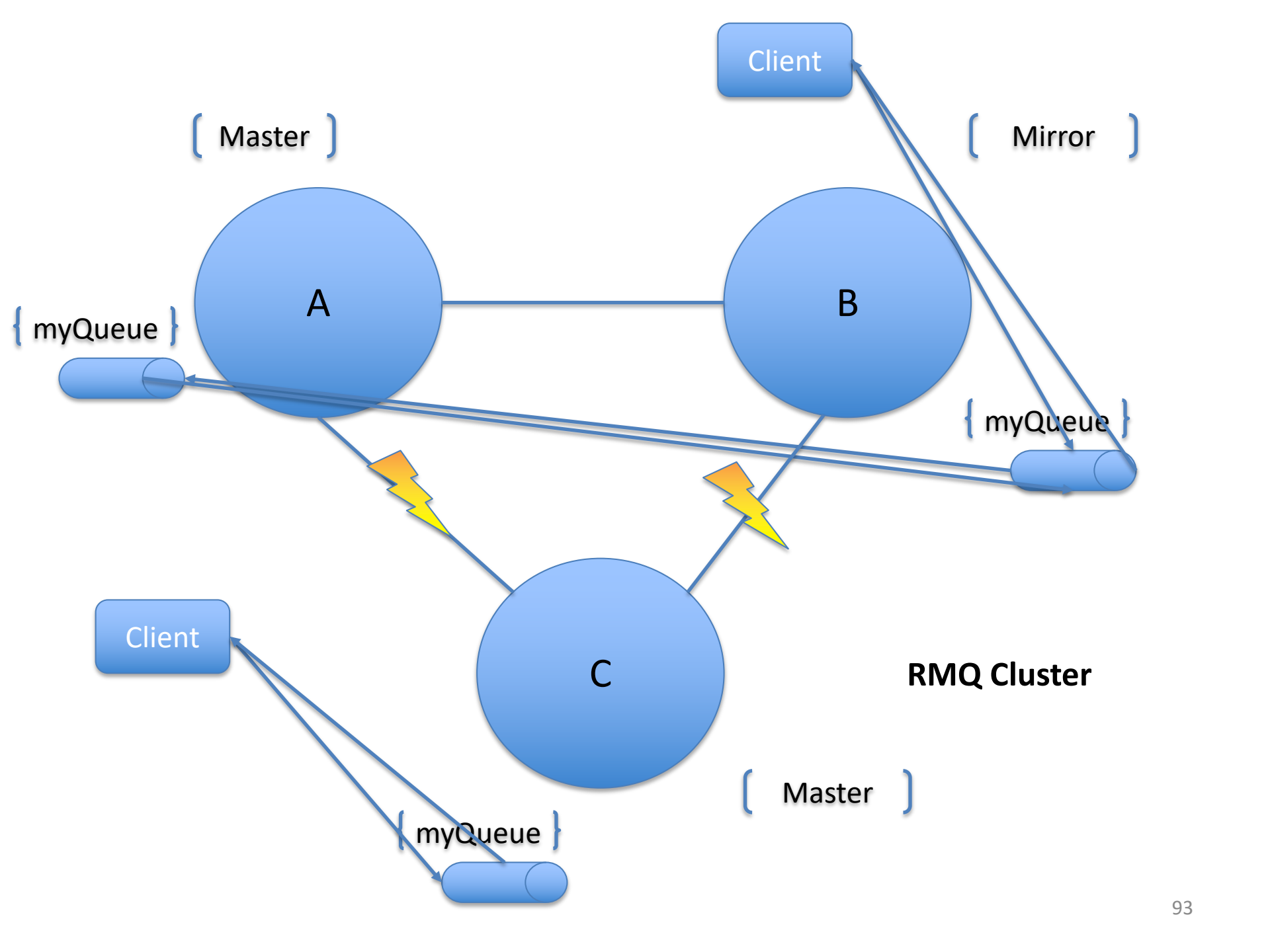
Forfeit Consistency

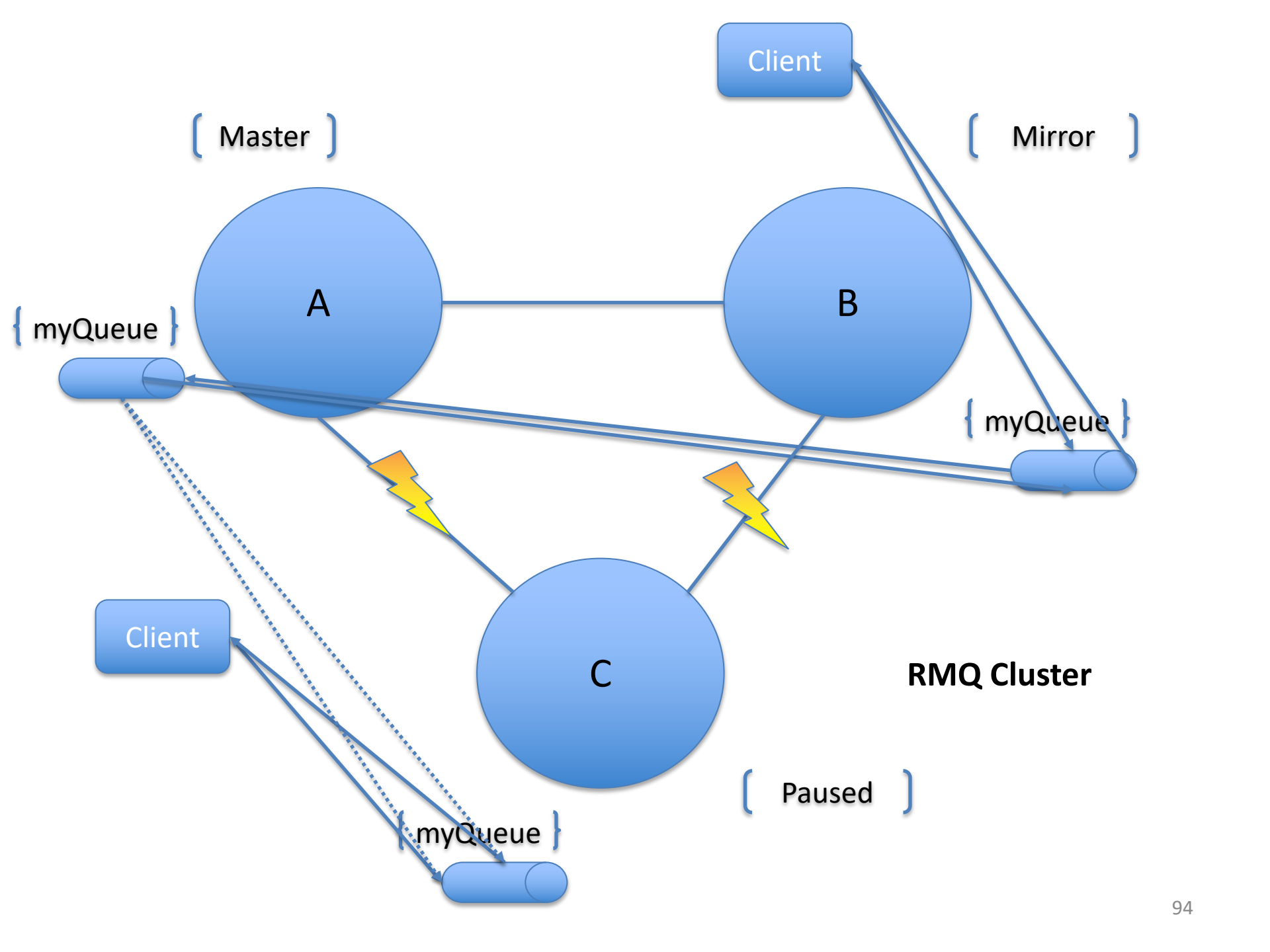

Master
Mirror
A
B
C
myQueue
1
2
3
4
myQueue
1
2
3
4
myQueue
1
4
Inconsistent

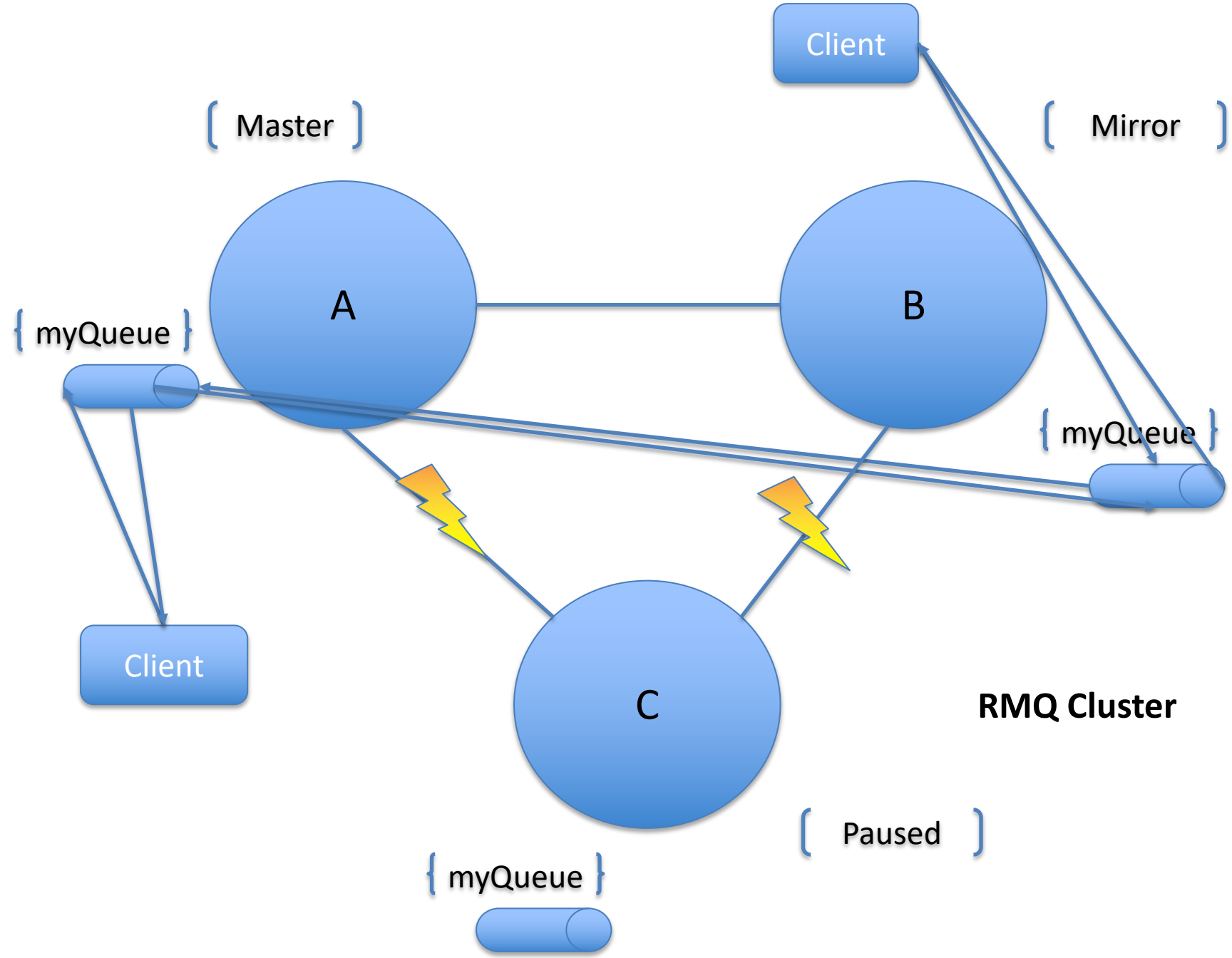

CA Category
RDBMS
Consistency
CP Category
BigTable
HBase
MongoDB
Redis
CA
CP
Availability
AP
Partition
Tolerance
AP Category
Dynamo
Voldemort
Cassandra
CouchDB

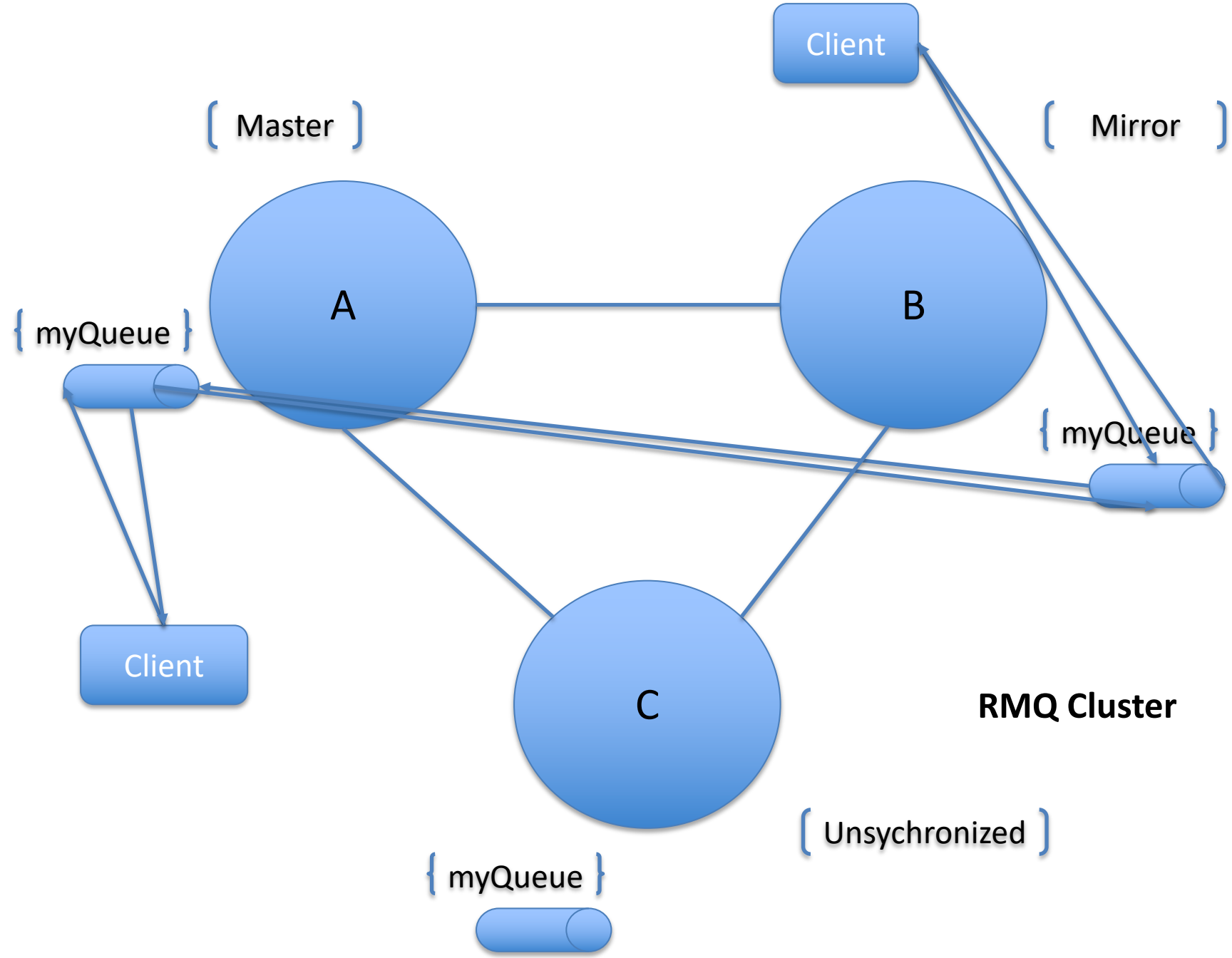
4.6 RABBITMQ AND CAP THEOREM

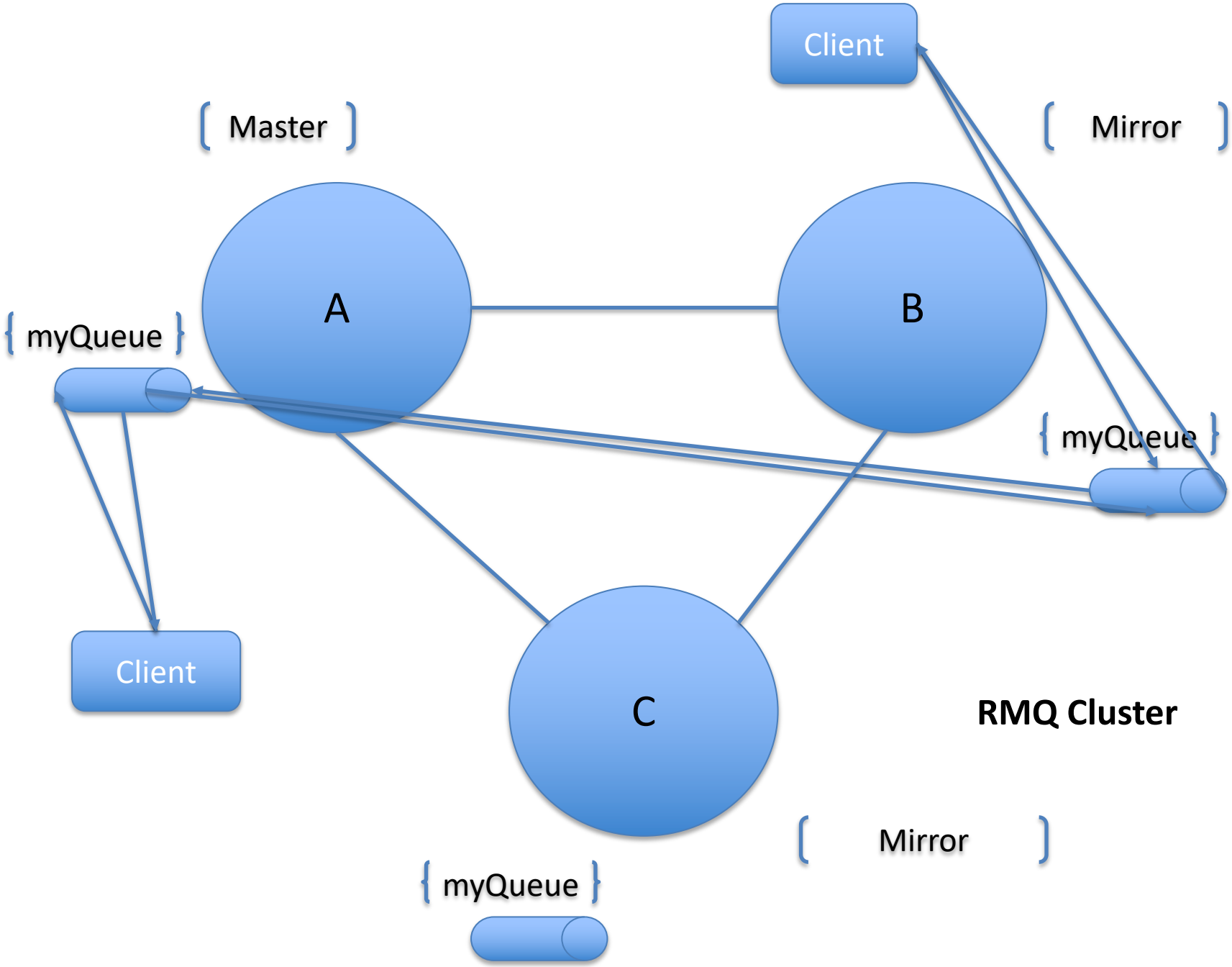

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

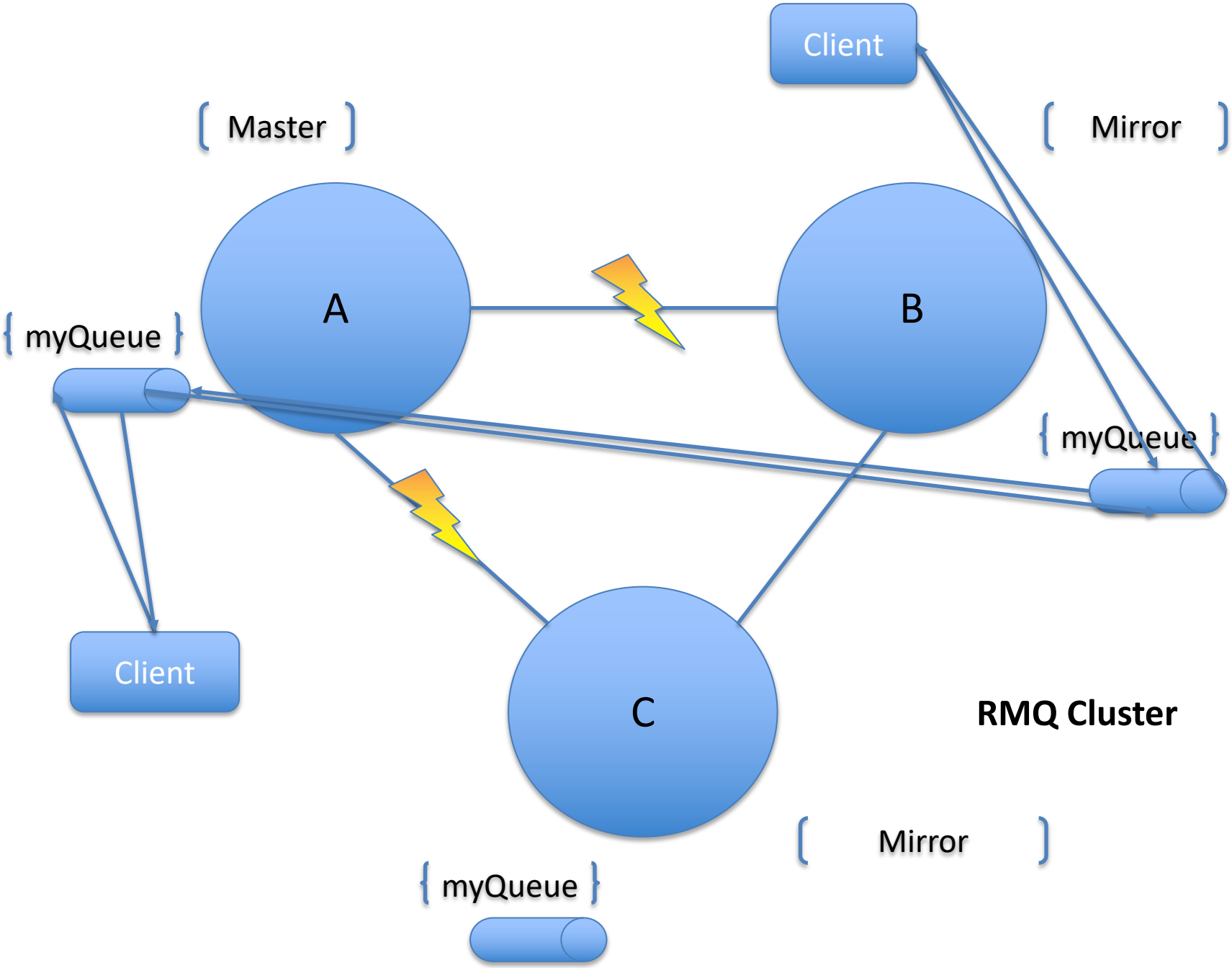

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Master
myQueue

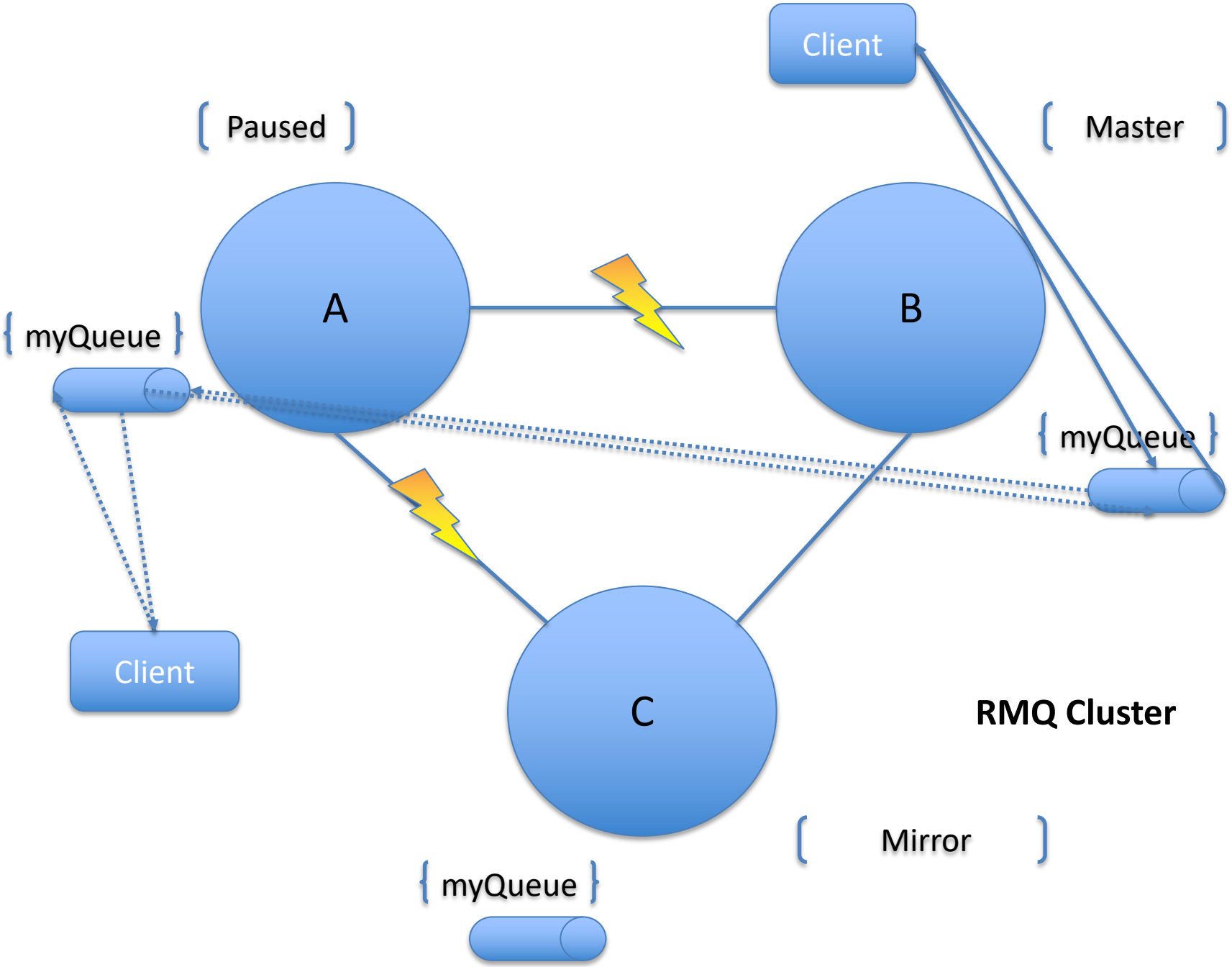

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Paused
myQueue

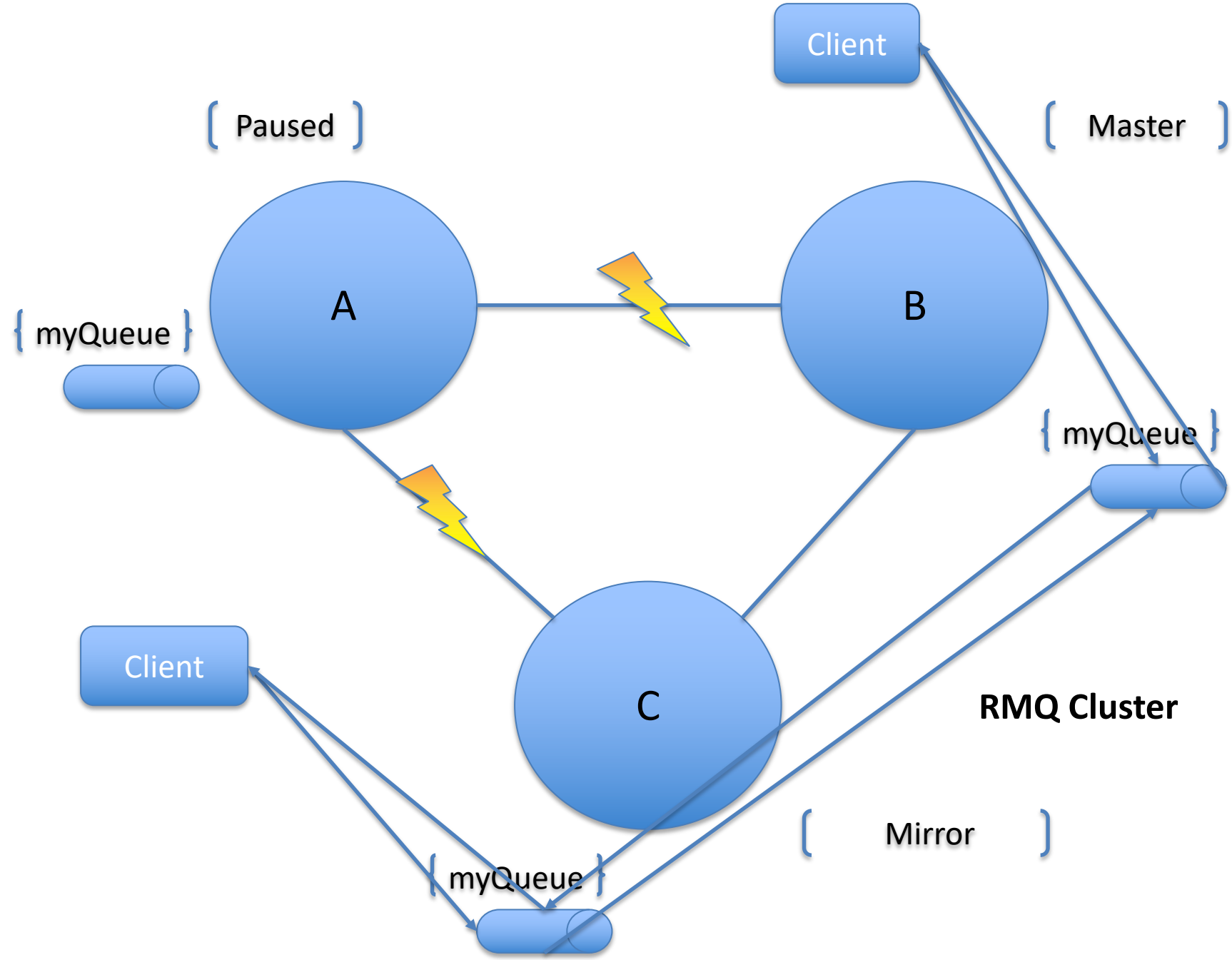

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Paused
myQueue

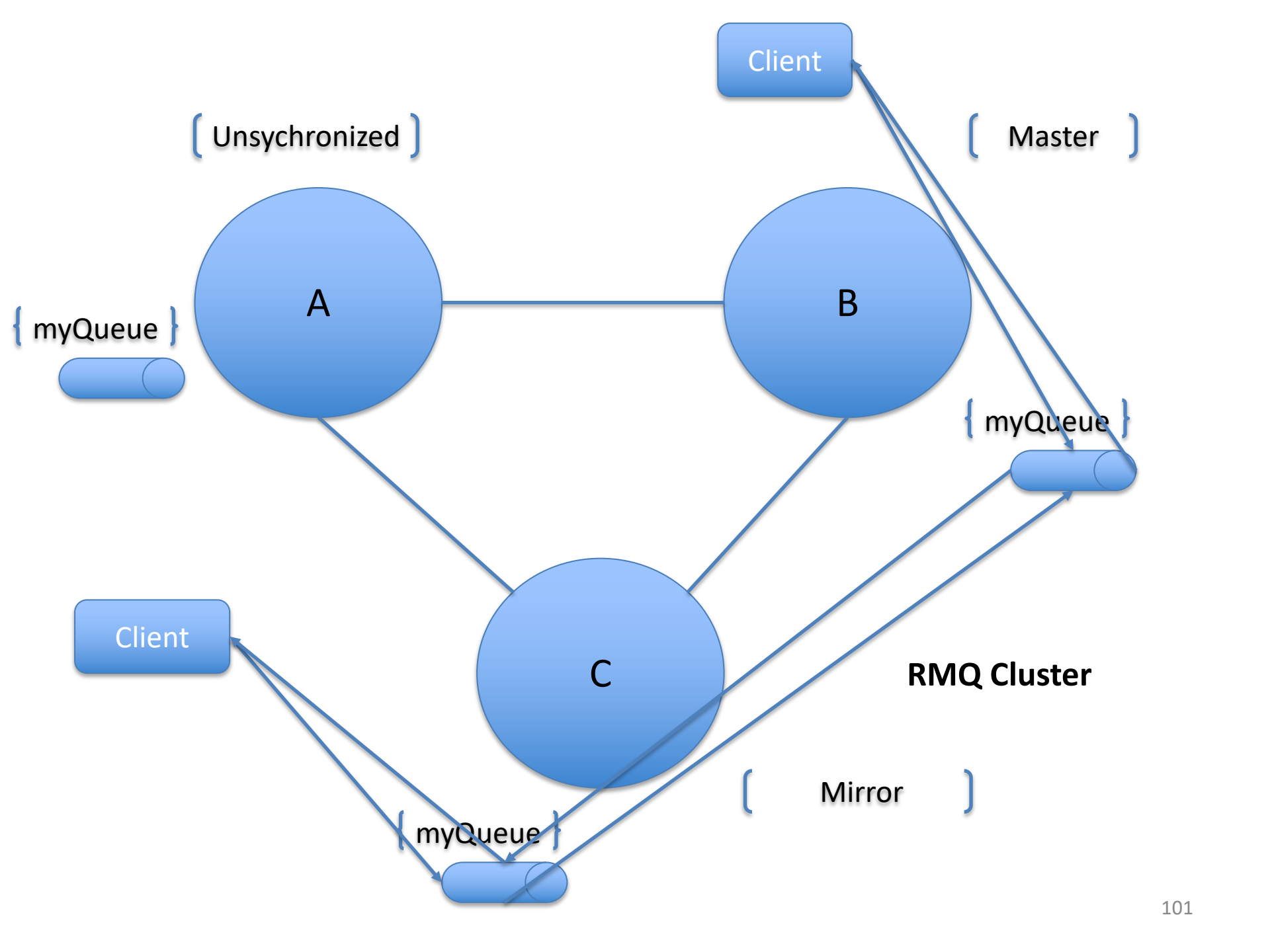

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Unsychronized
myQueue

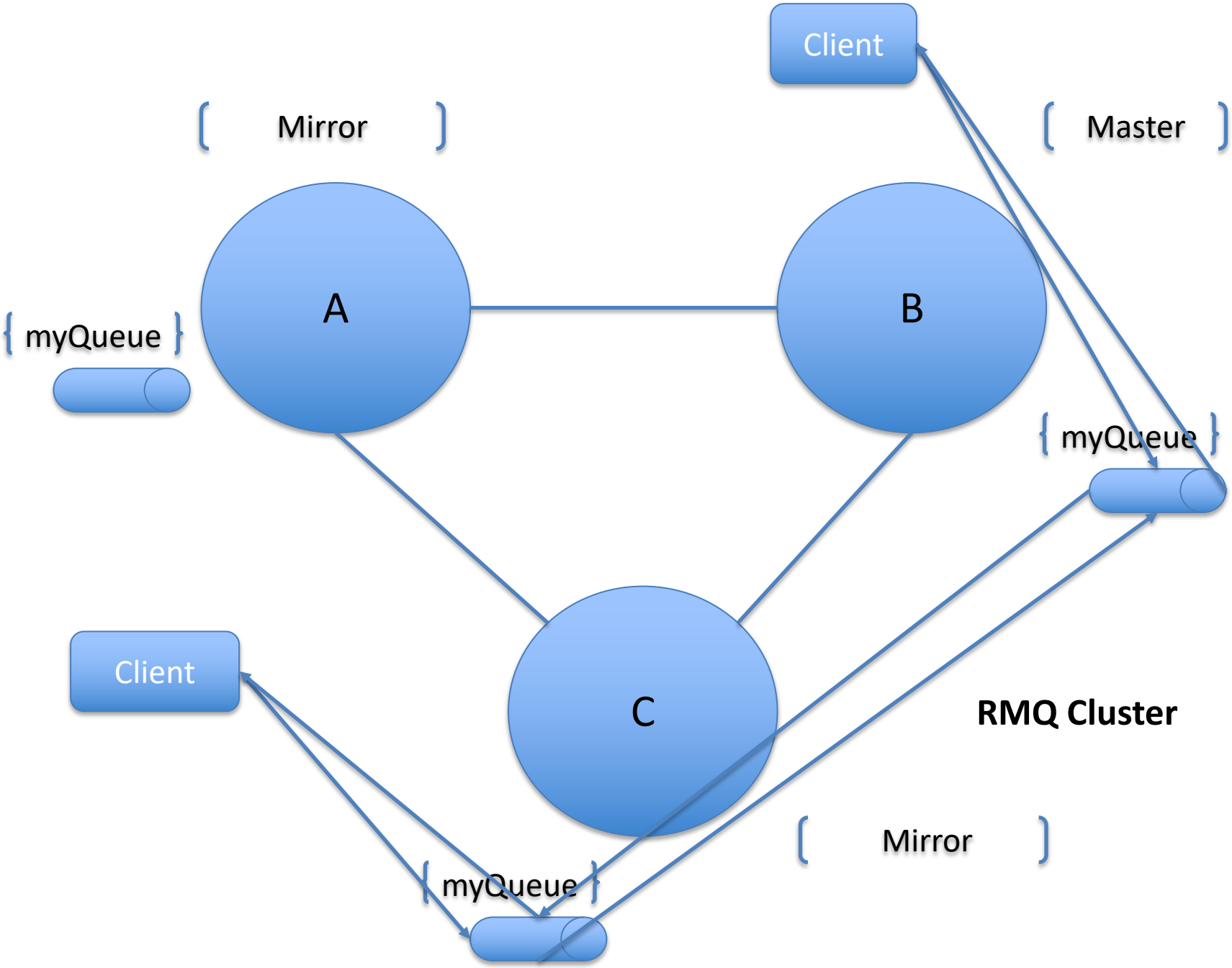

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

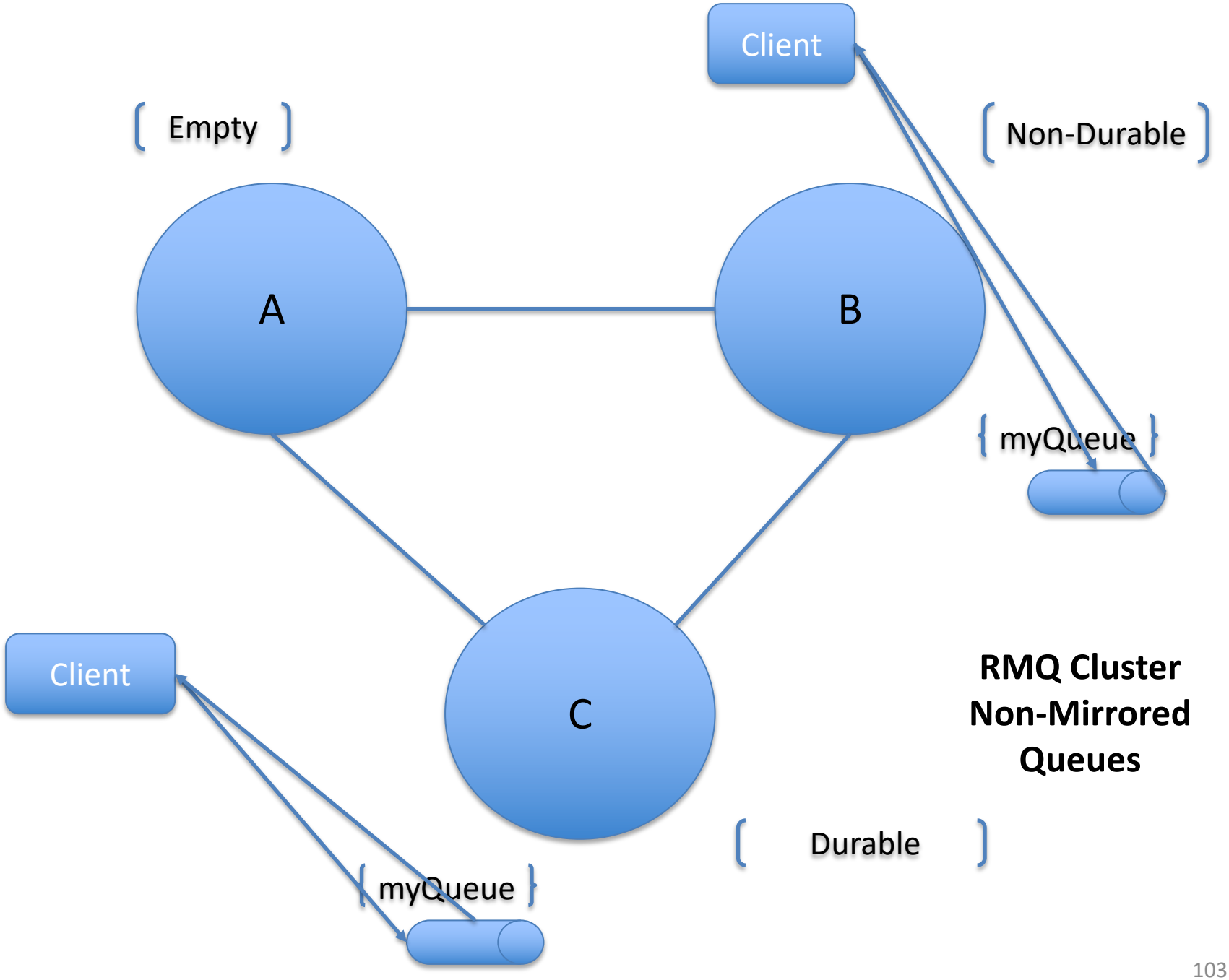

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

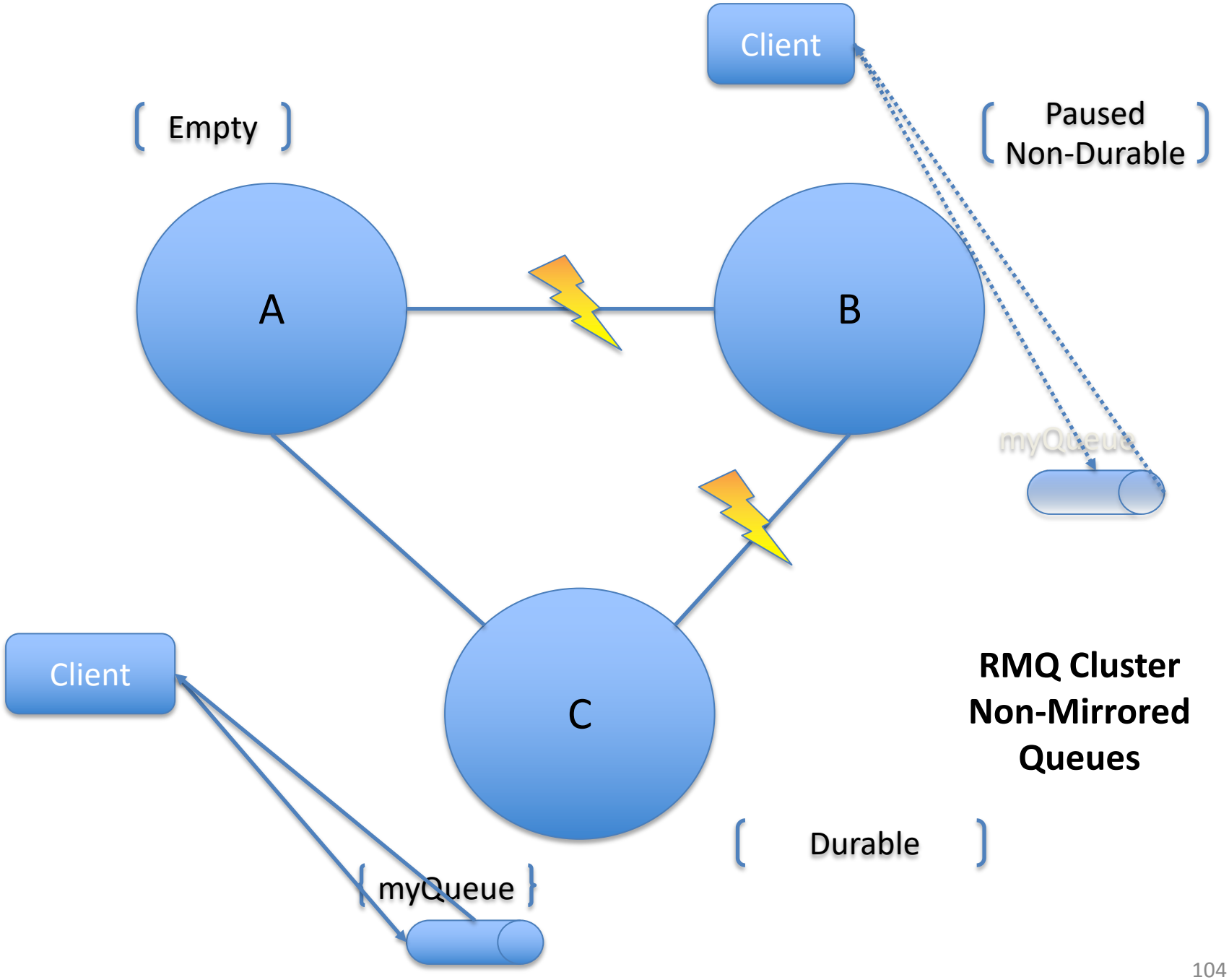

Client
Paused
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

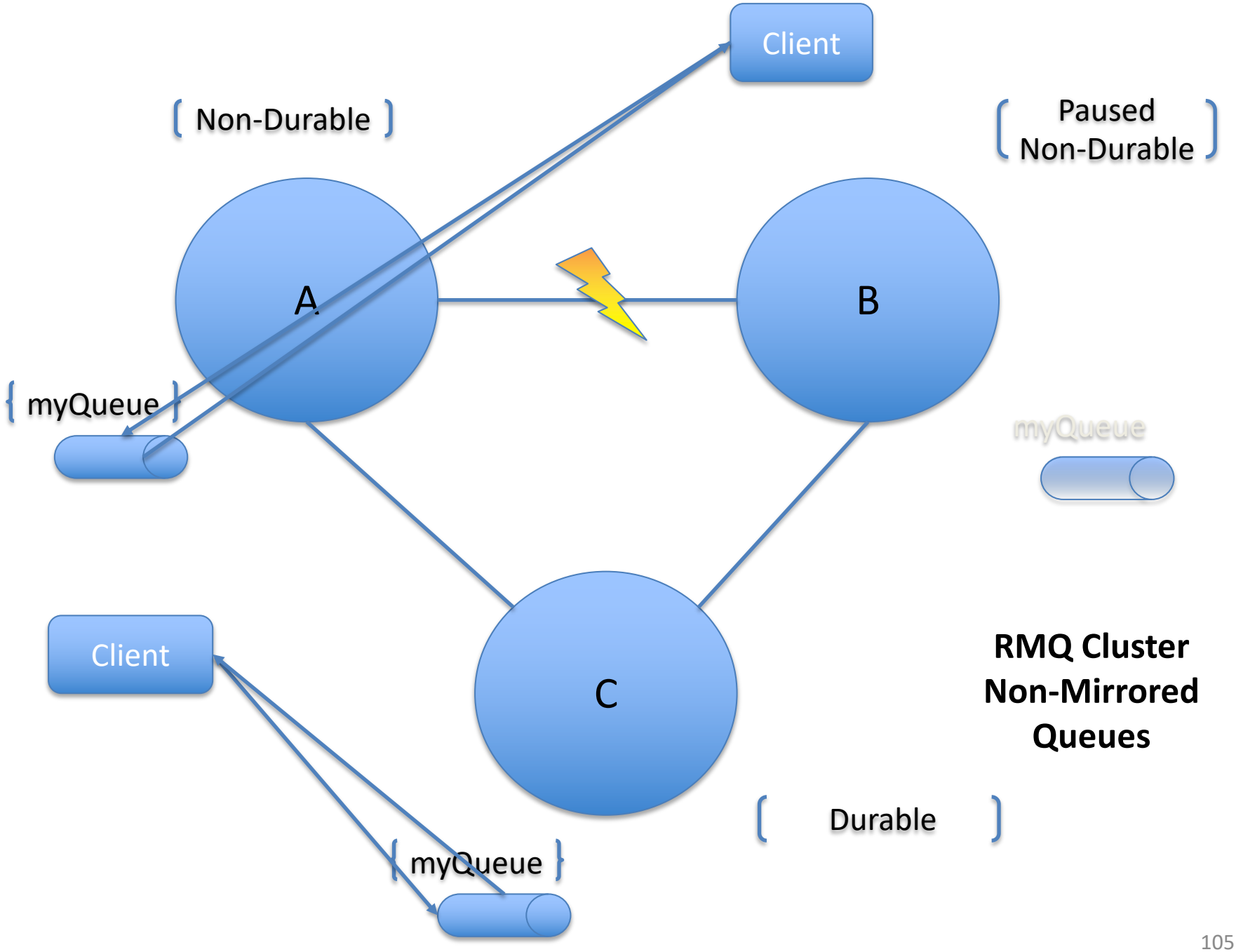

Client
Paused
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

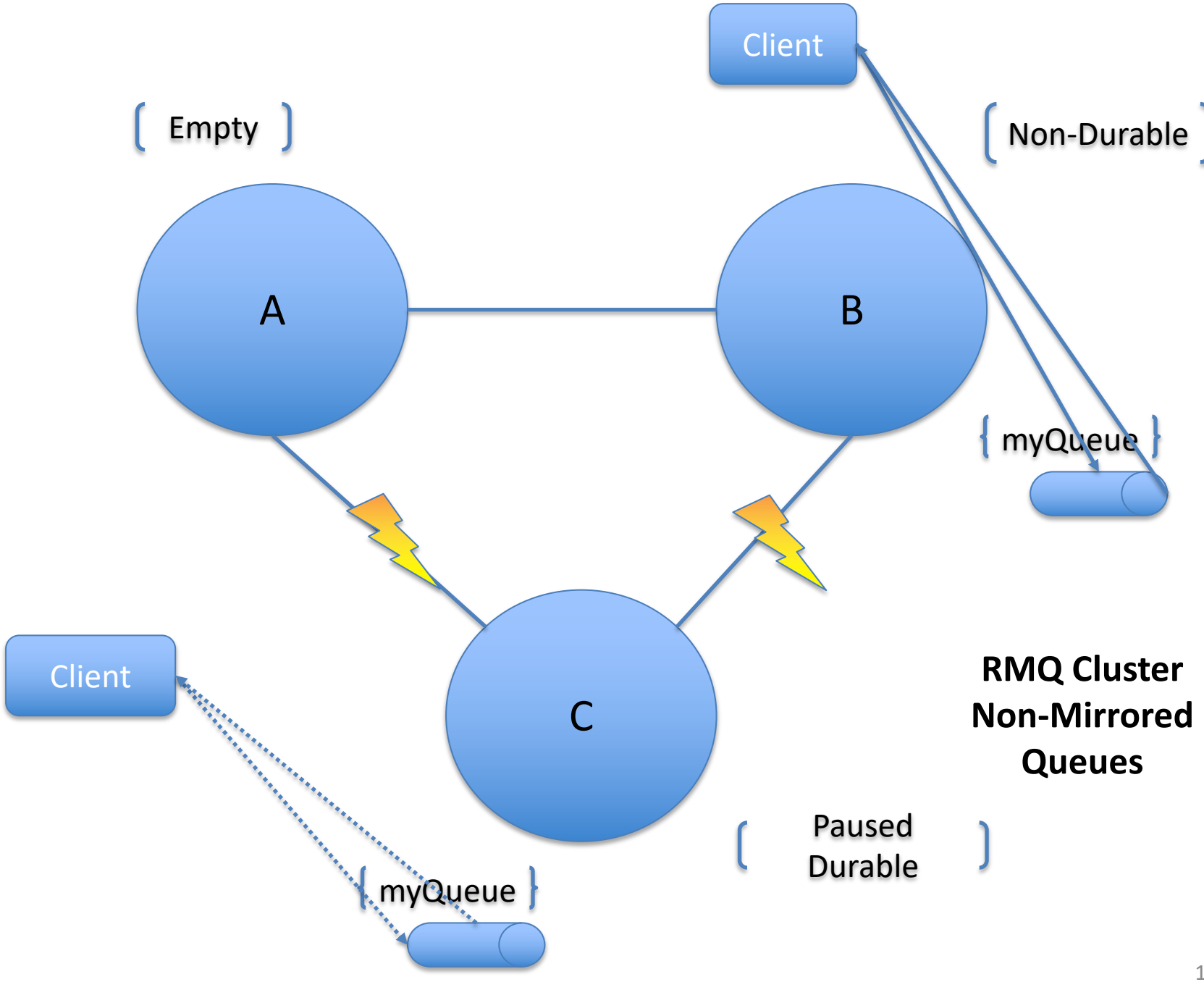

Client
Unsychronized
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

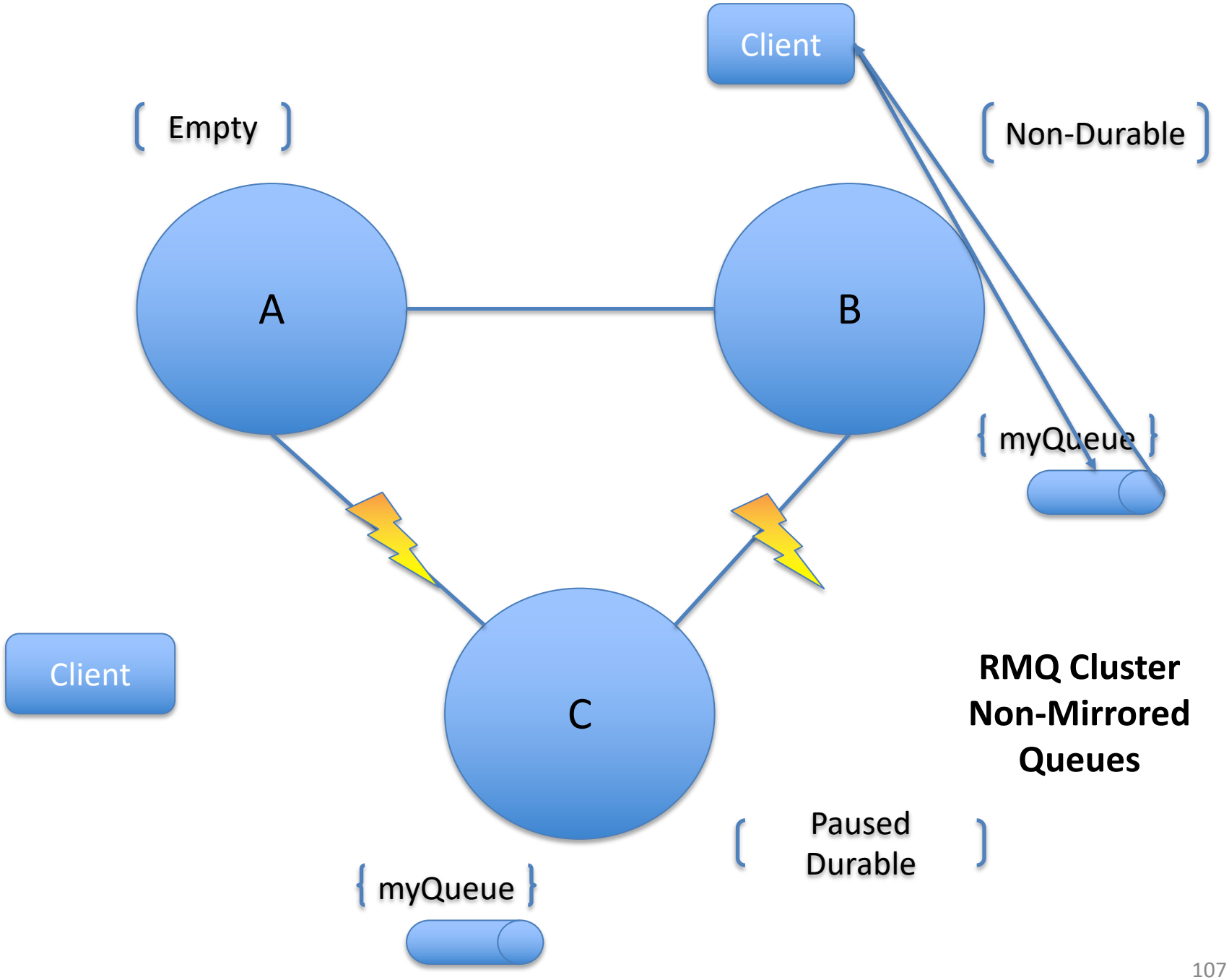

Client
Mirror
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

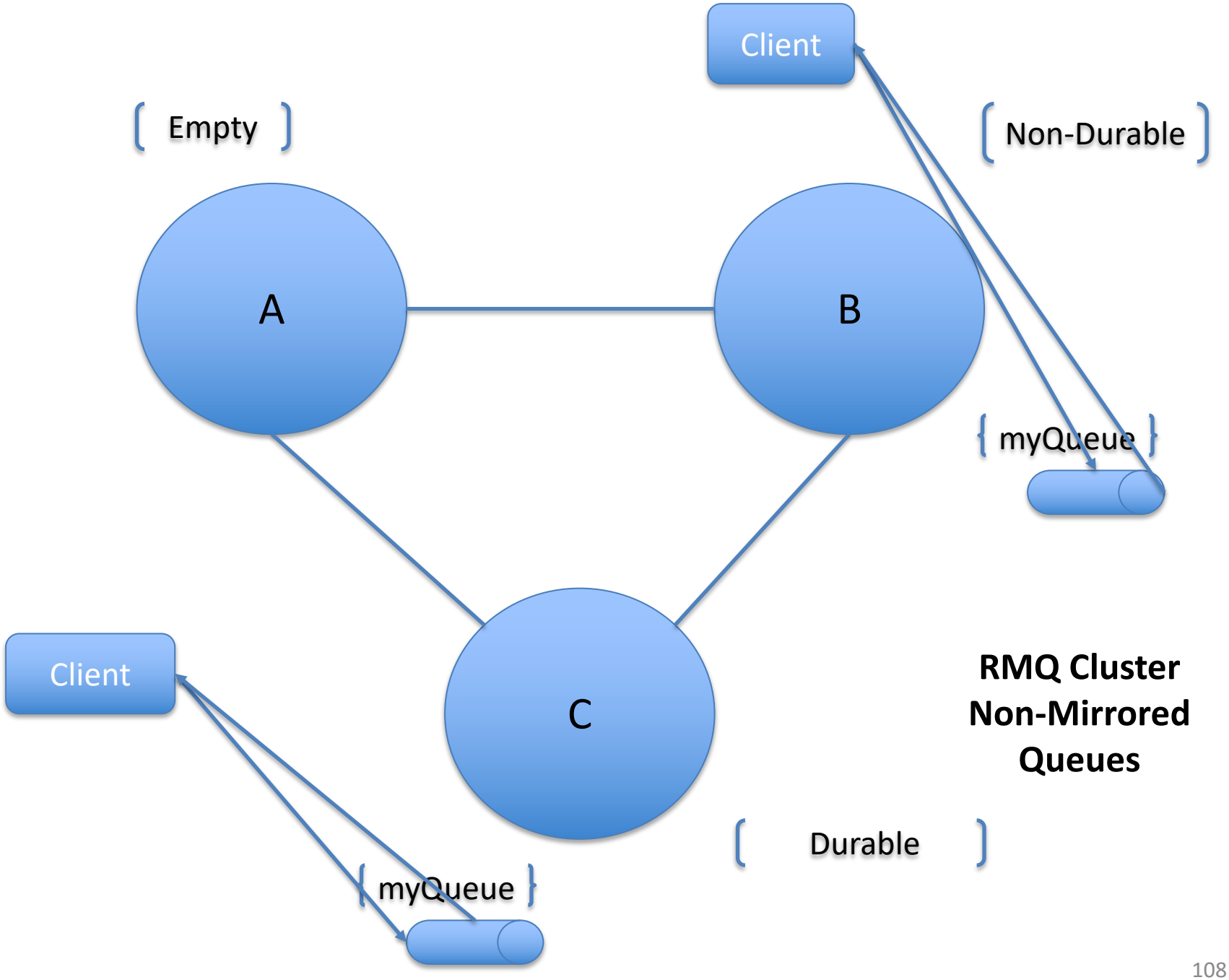

Client
Empty
Non-Durable
A
B
myQueue
Client
C
RMQ Cluster
Non-Mirrored
Queues
Durable
myQueue


Client
Empty
Paused
Non-Durable
A
B
myQueue
Client
C
RMQ Cluster
Non-Mirrored
Queues
Durable
myQueue


Client
Non-Durable
Paused
Non-Durable
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Non-Mirrored
Queues
Durable
myQueue


Client
Empty
Non-Durable
A
B
myQueue
Client
C
RMQ Cluster
Non-Mirrored
Queues
Paused
Durable
myQueue


Client
Empty
Non-Durable
A
B
myQueue
Client
C
RMQ Cluster
Non-Mirrored
Queues
Paused
Durable
myQueue


Client
Empty
Non-Durable
A
B
myQueue
Client
C
RMQ Cluster
Non-Mirrored
Queues
Durable
myQueue

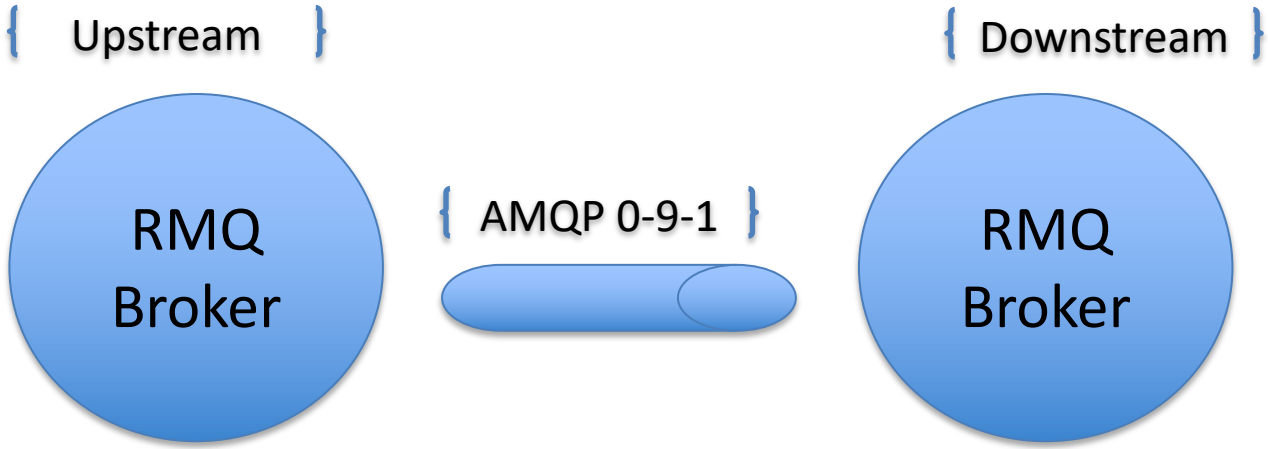
So which CAP options does an RMQ cluster support?

Pause Minority is Consistency (C) and Partition Tolerance (P)

In normal operation we sacrifice latency—time taken for a message to propagate to all nodes for consistency—all nodes will have a copy of the message in case of failure.

Non-durable non-mirrored queues are Availability (A) and Partition Tolerance (P)

In normal operation we do sacrifice consistency—there are no copies—for improved latency as we do not have to copy the data to the same number of nodes.
But a message queue tends to always sacrifice L for A



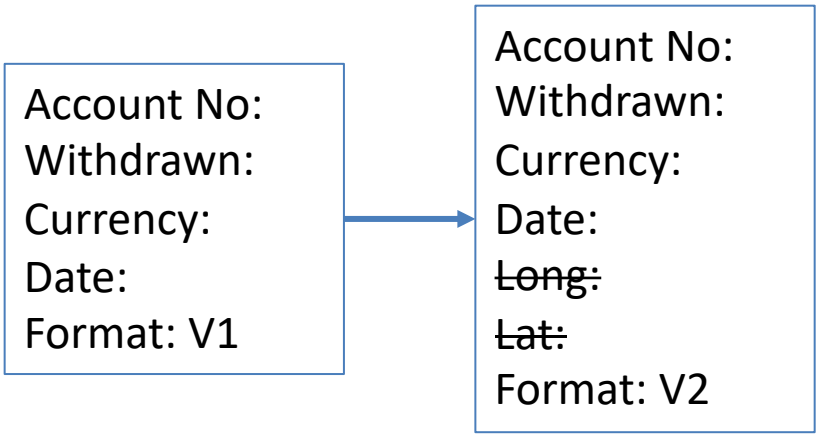
RMQ Federation/Shovel

4.7 VERSIONING

Be strict when sending and tolerant when receiving.
Implementations must follow specifications precisely when sending to the network, and tolerate faulty input from the network.

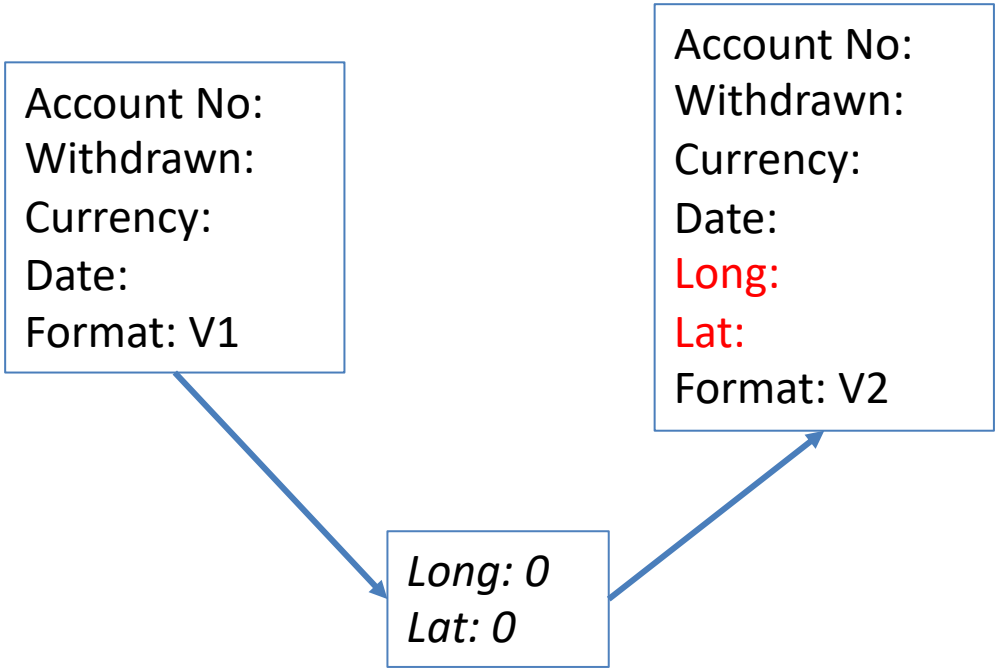
Robustness Principal or Postel's Law – Jon Postel RFC 1958

Tolerant Reader



Ignore New Fields

Tolerant Reader



Default Missing Fields

Breaking Change

Account No:
Withdrawn:
Currency:
Date:
Format: V1

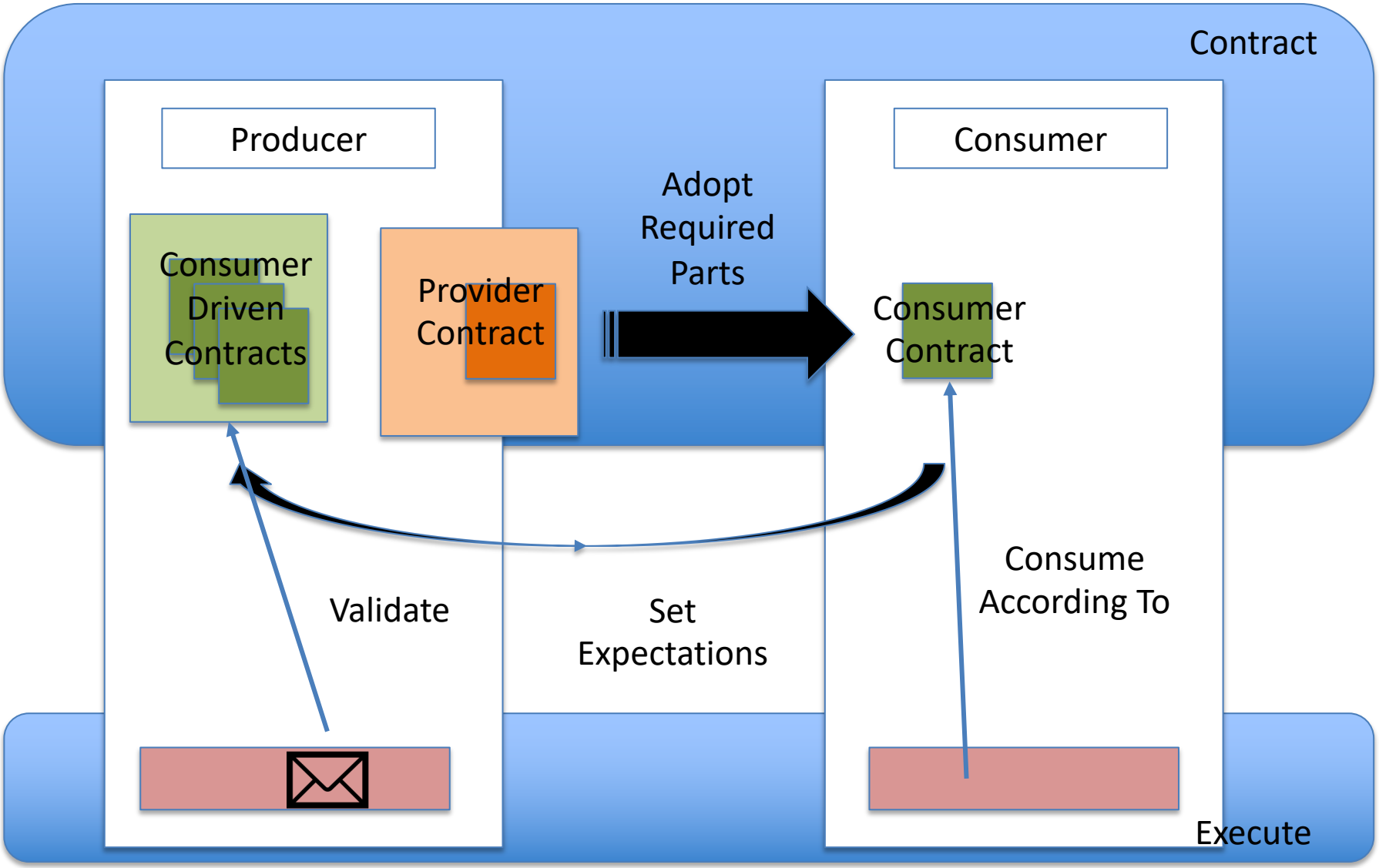
Account.Withdrawal.Event

Account No:
New Balance:
Date:
Format: V2

Account.NewBalance.Event

New Message

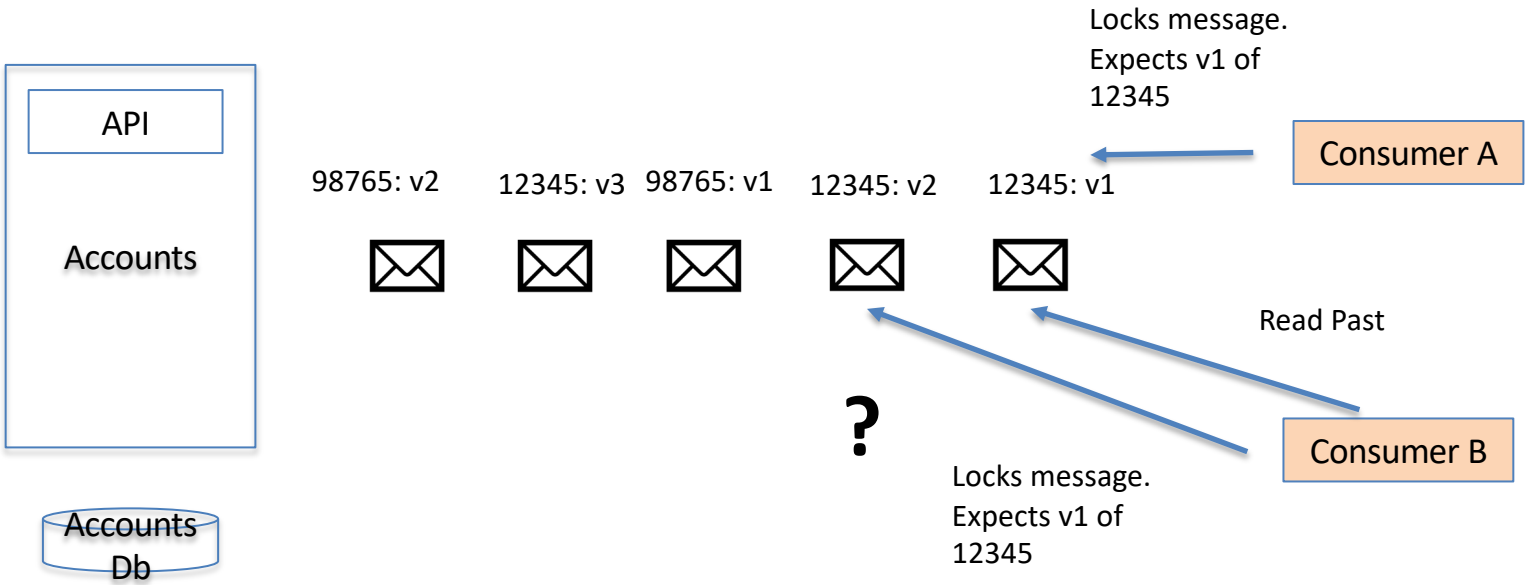
Consumer Driven Contracts



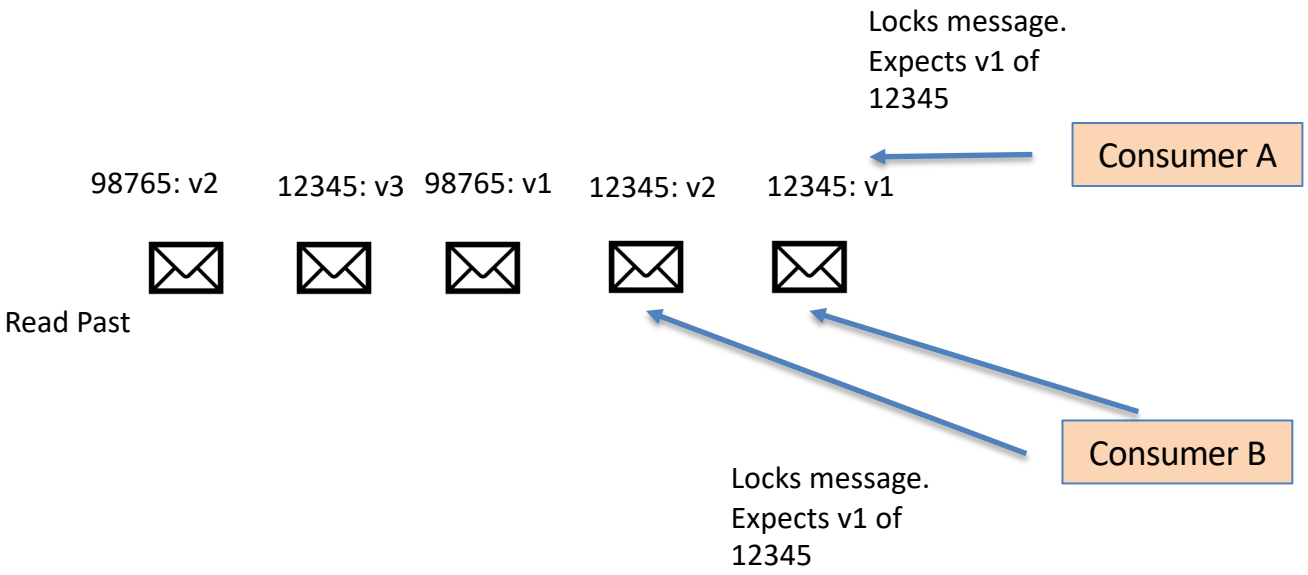
In Order Delivery

5. ORDER

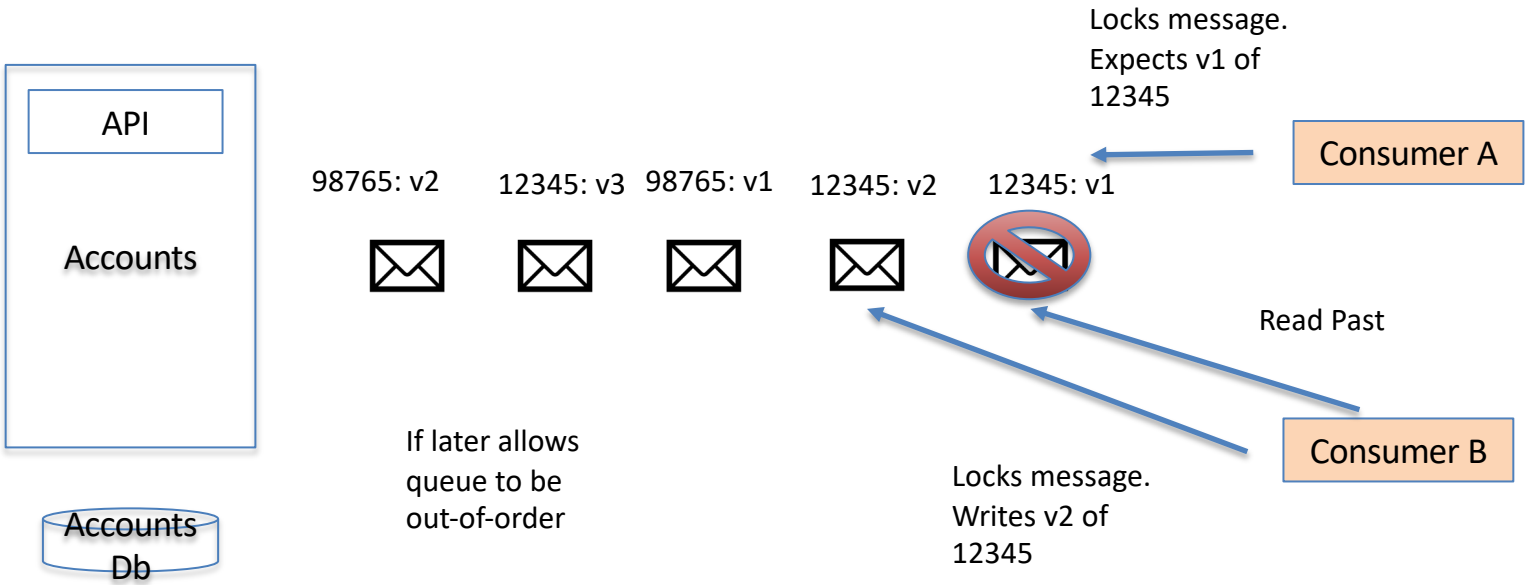
Messaging Order and Competing Consumers



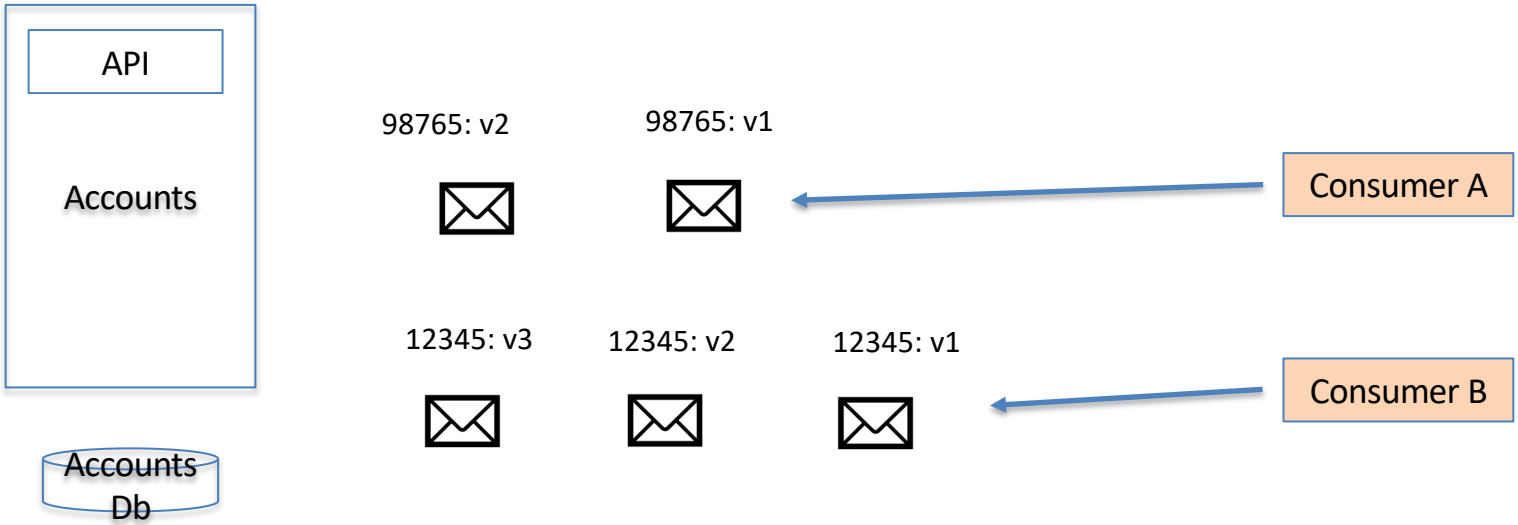
Requeue?


Locks message.
Expects v1 of
12345
Consumer A
98765: v2
12345: v3
98765: v1
12345: v2
12345: v1
Read Past
Consumer B
Locks message.
Expects v1 of
12345

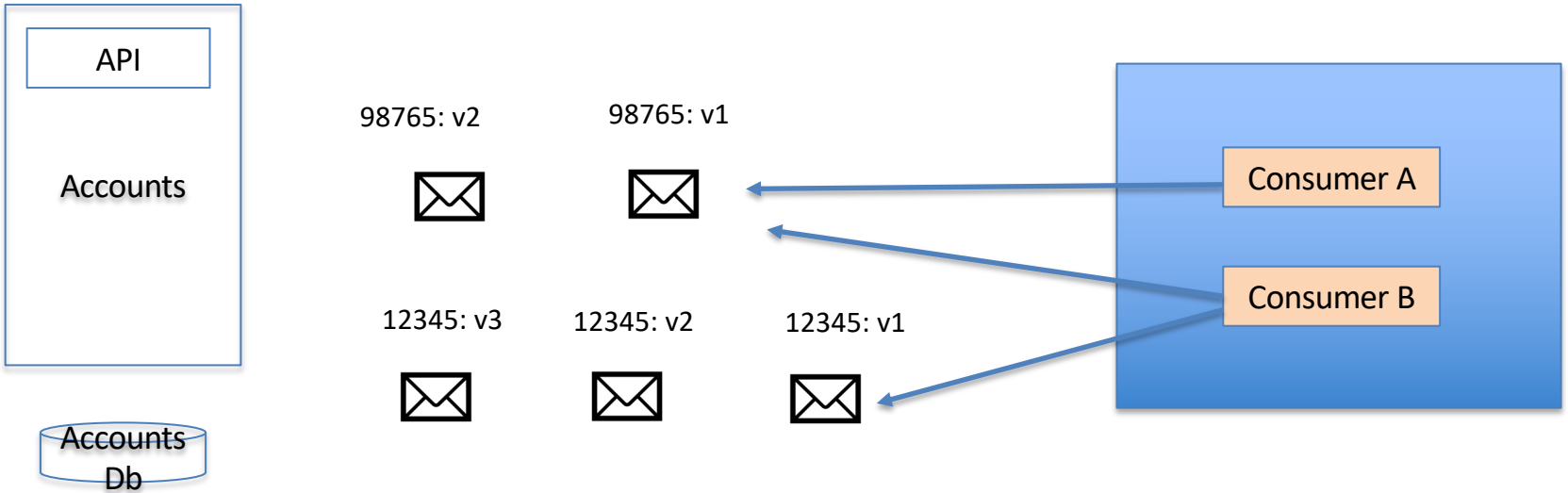
If Later


API
Accounts
Accounts Db
98765: v2
12345: v3
98765: v1
12345: v2
12345: v1
Locks message. Expects v1 of 12345
Consumer A
Read Past
Consumer B
If later allows queue to be out-of-order
Locks message. Writes v2 of 12345

Consistent Hashing



Consistent Hashing w. Consumer Groups



Standards & How to Describe Messaging

6. DOCUMENTATION

Why AsyncAPI?

Improving the current state of Event-Driven Architectures (EDA)

Specification

Allows you to define the interfaces of asynchronous APIs and is protocol agnostic.

Documentation

Document APIs

Use our tools to generate documentation at the build level, on a server, and on a client.

HTML Template React Component

Code Generation

Generate documentation, Code (TypeScript, Java, C#, etc), and more out of your AsyncAPI files.

Generator Modelina

Community

We're a community of great people who are passionate about AsyncAPI and event-driven architectures.

Join our Slack

Open Governance

Our Open-Source project is part of Linux Foundation and works under an Open Governance model.

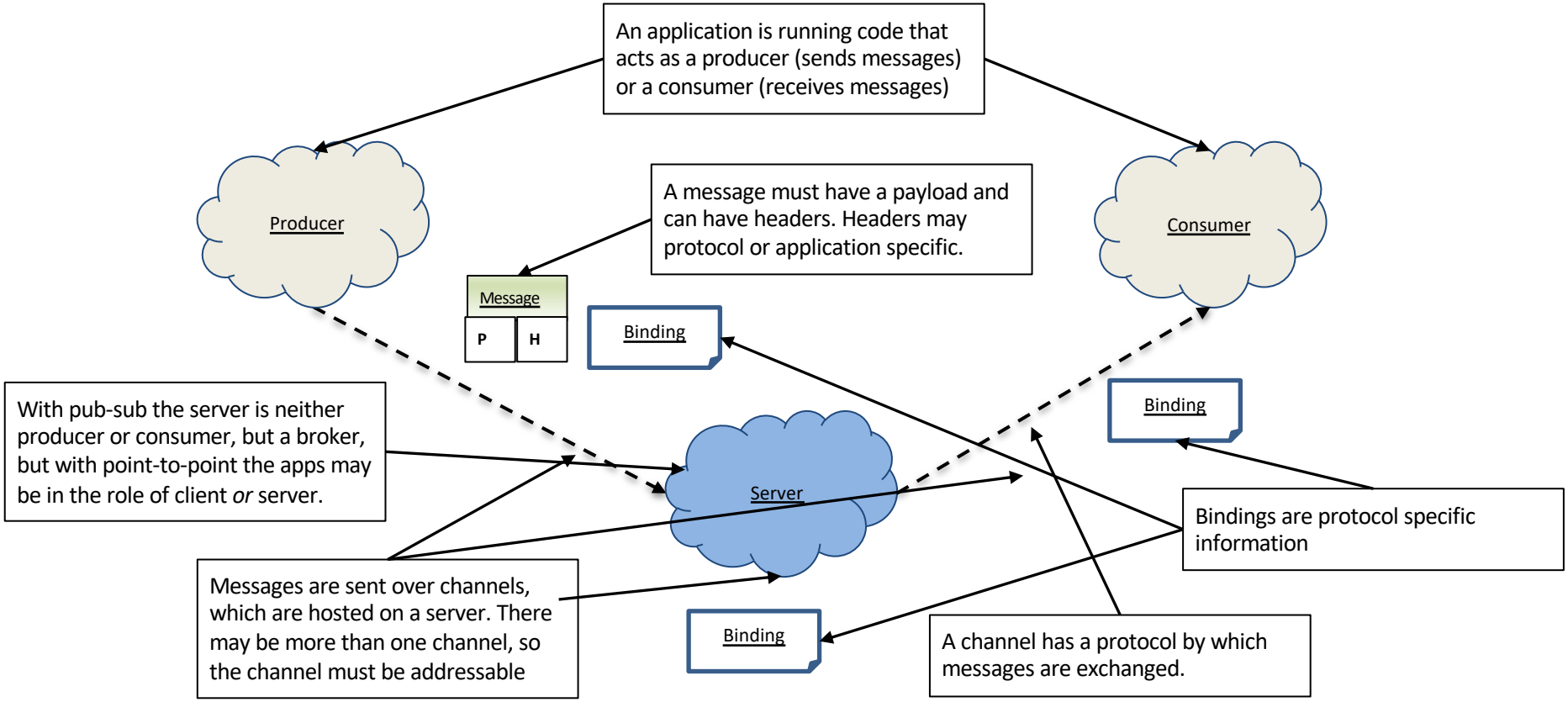
Read more about Open Governance TSC Members

And much more...

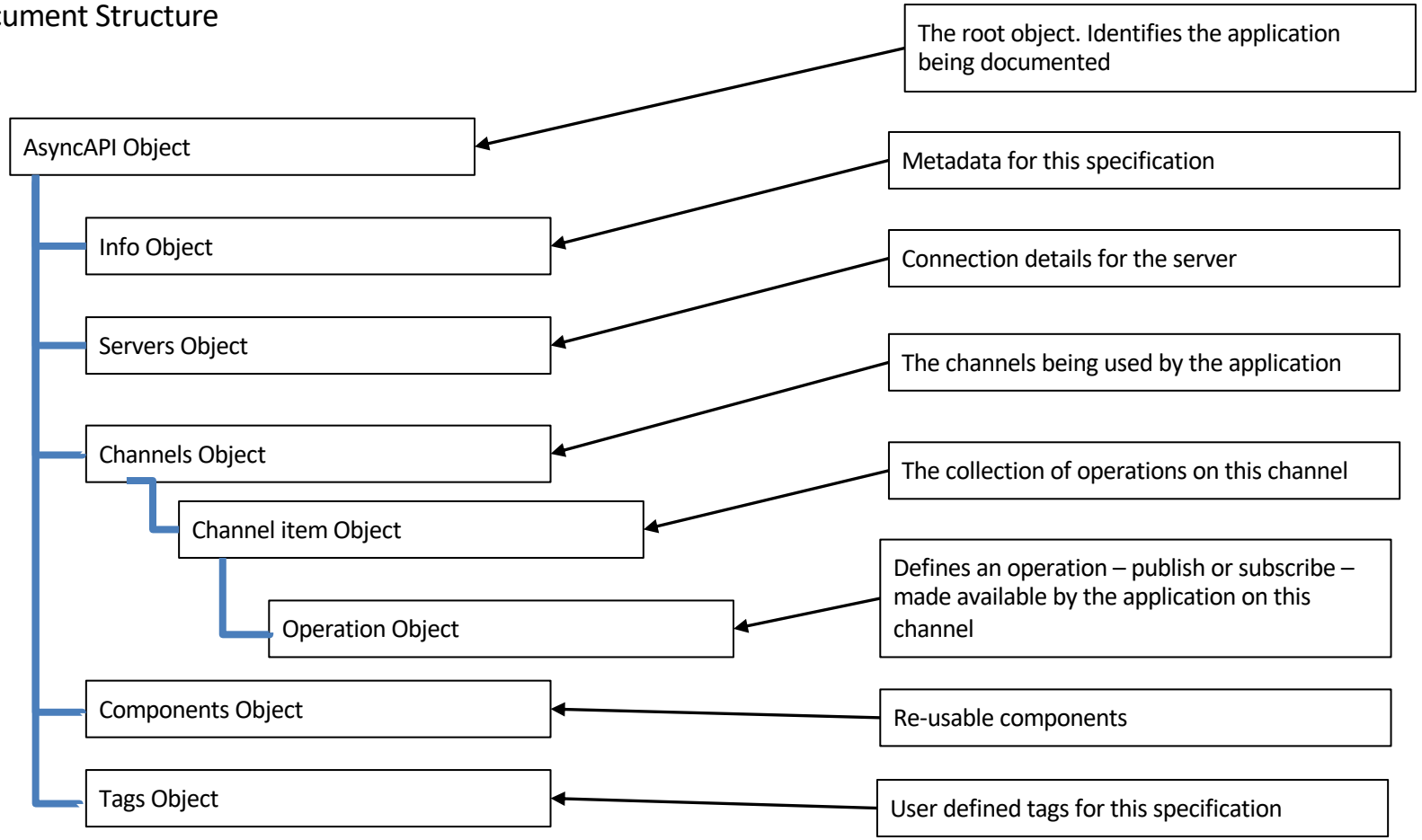
We have many different tools and welcome you to explore our ideas and propose new ideas to AsyncAPI.

View GitHub Discussions

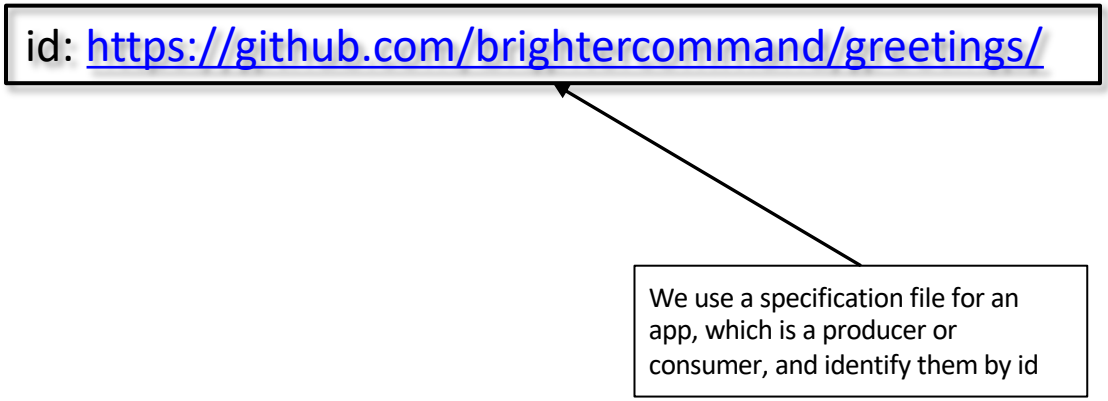
AsyncAPI Elements



Document Structure



AsyncAPI Object



Info Object

```
info:
  contact:
    name: Paramore Brighter
    url: https://goparamore.io/support
    email: support@goparamore.io
  license:
    name: Apache 2.0
    url: https://www.apache.org/licenses/LICENSE-2.0.html
  description: Demonstrates sending a greeting over a messaging
transport.
  title: Brighter Sample App
  version: 1.0.0
tags:
  - name: brighter examples
```

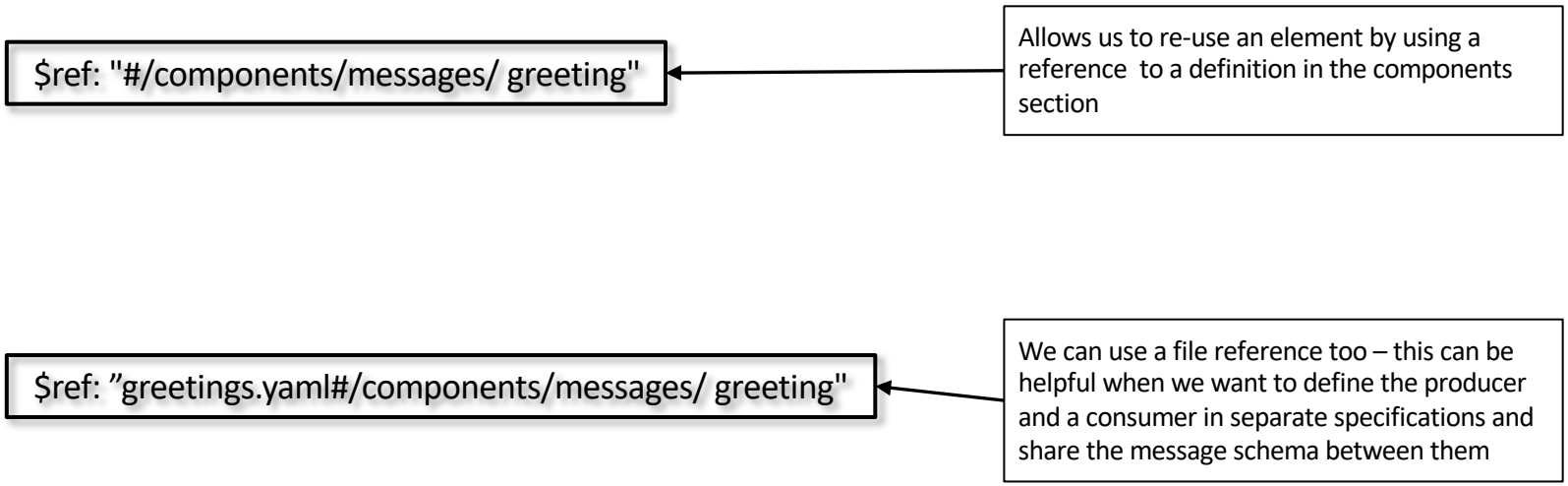
Servers Object

```
local:
  url: localhost:9092
  protocol: kafka
```

Channels

```
greeting:
  subscribe:
    operationid: sendGreeting
    summary: sends a greeting
    description: This service lets you send the ‘Hello World’ greeting to another service.
    message:
      $ref: "#/components/messages/greeting”
```

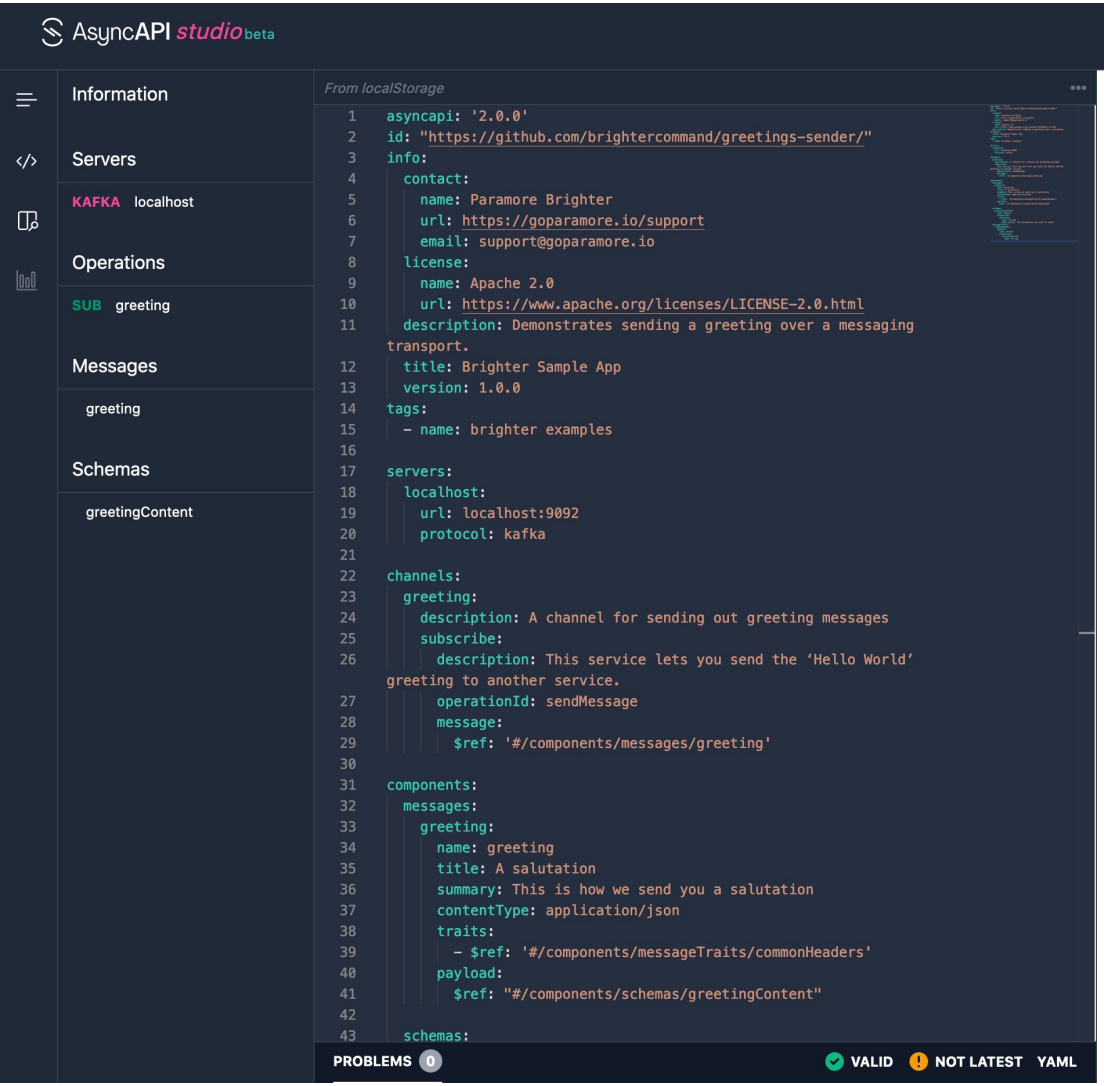
Ref


$ref: "#/components/messages/ greeting"
Allows us to re-use an element by using a reference to a definition in the components section
$ref: ”greetings.yaml#/components/messages/ greeting"
We can use a file reference too – this can be helpful when we want to define the producer and a consumer in separate specifications and share the message schema between them

Components

```
components:
  messages:
    greeting:
      name: greeting
      title: A salutation
      summary: This is how we send you a salutation
      contentType: application/json
      traits:
        - $ref: '#/components/messageTraits/commonHeaders'
      payload:
        $ref: "#/components/schemas/greetingContent"

  schemas:
    greetingContent:
      type: object
      properties:
        greeting:
          type: string
          description: The salutation you want to send
...
```



Brighter Sample App 1.0.0

APACHE 2.0 | PARAMORE BRIGHTER | SUPPORT@GOPARAMORE.IO

ID: HTTPS://GITHUB.COM/BRIGHTERCOMMAND/GREETINGS-SENDER/

Demonstrates sending a greeting over a messaging transport.

#brighter examples

Servers

localhost:9092 KAFKA LOCALHOST

Security:

SECURITY.PROTOCOL: PLAINTEXT

Operations

SUB greeting

A channel for sending out greeting messages

This service lets you send the 'Hello World' greeting to another service.

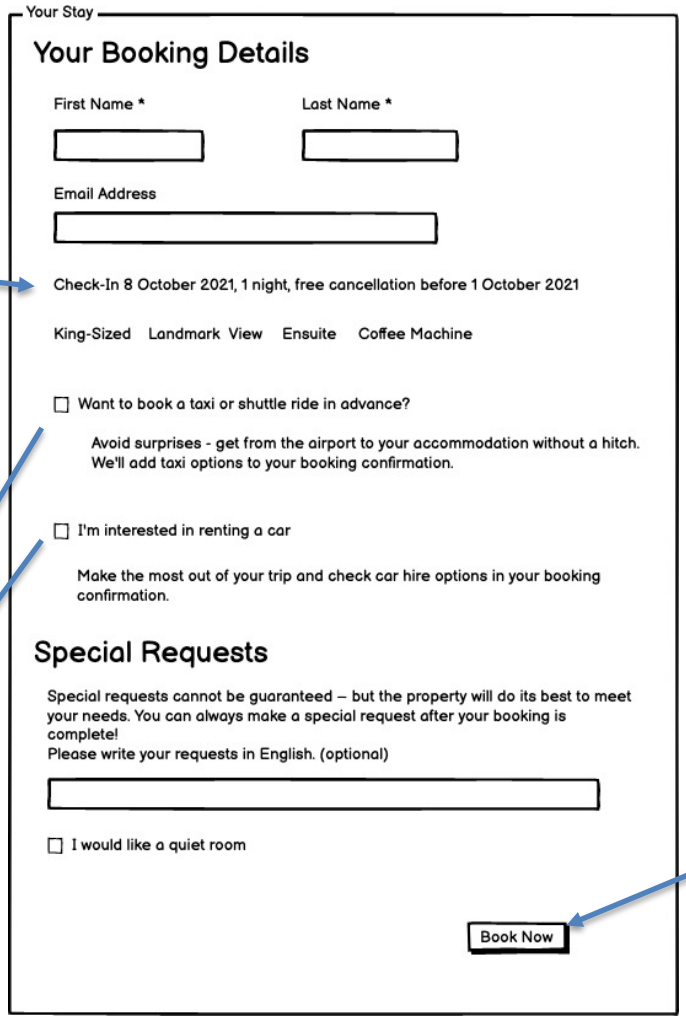
Operation ID sendMessage

Accepts the following message:

A salutation greeting

7. CONSUMERS

Task Based UI

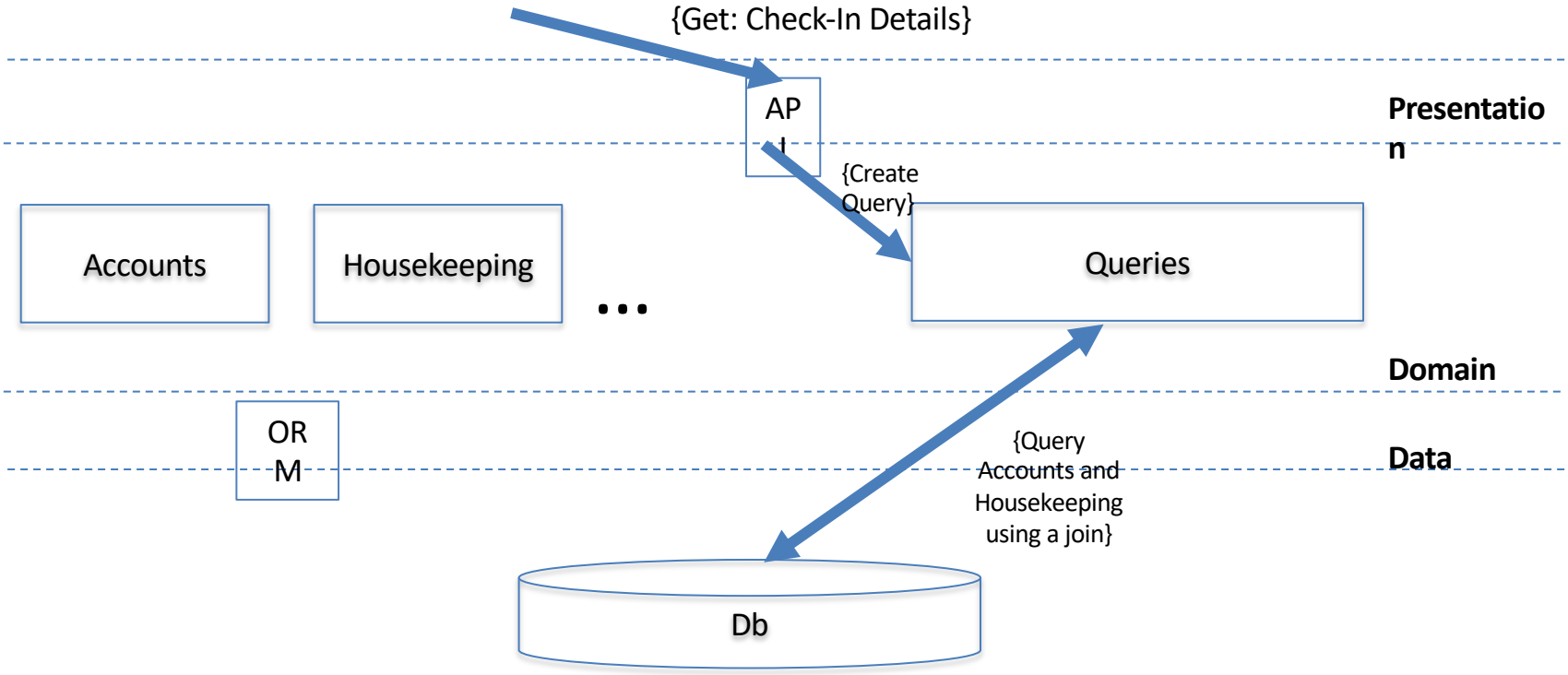


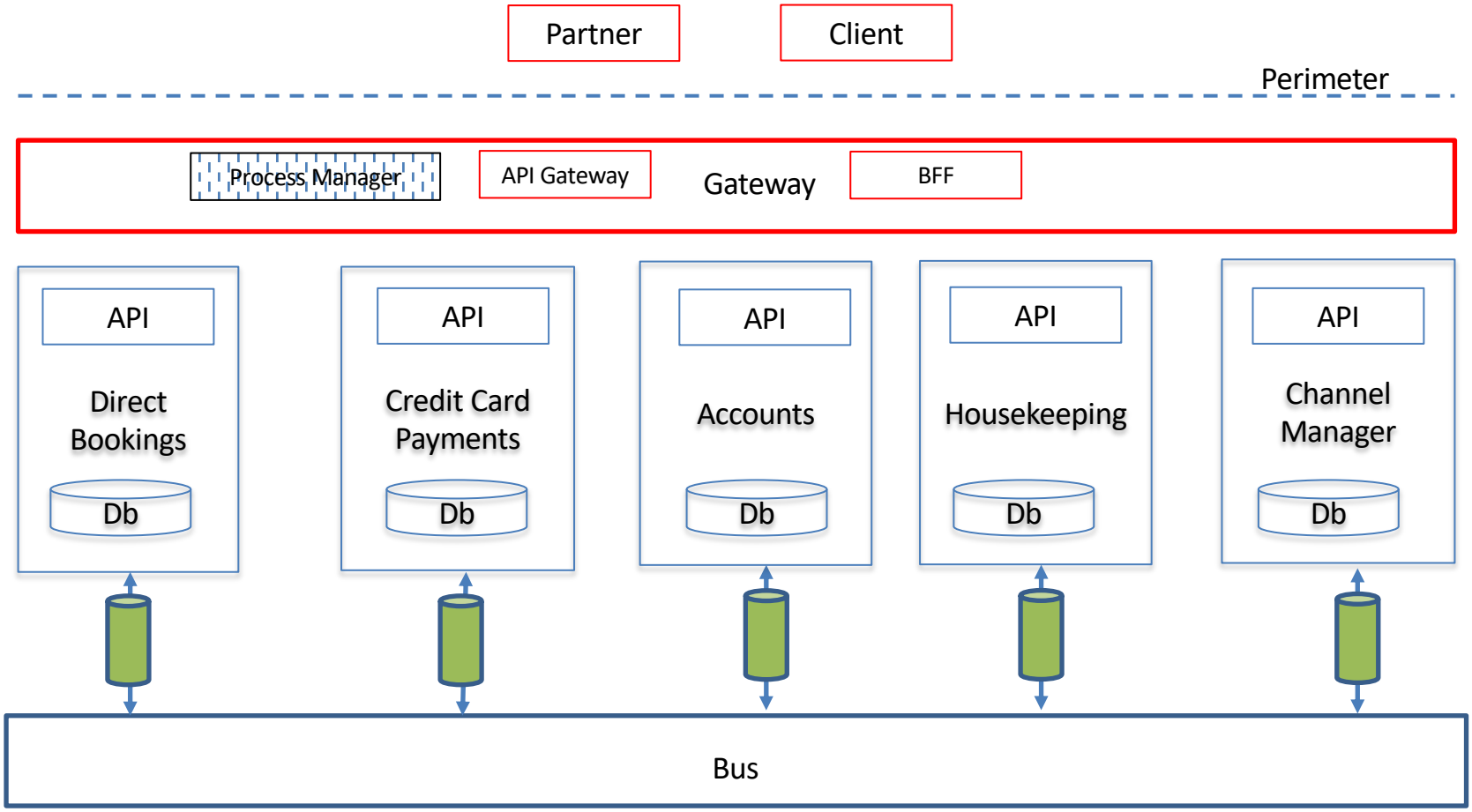
A sequence of screens can build up a request

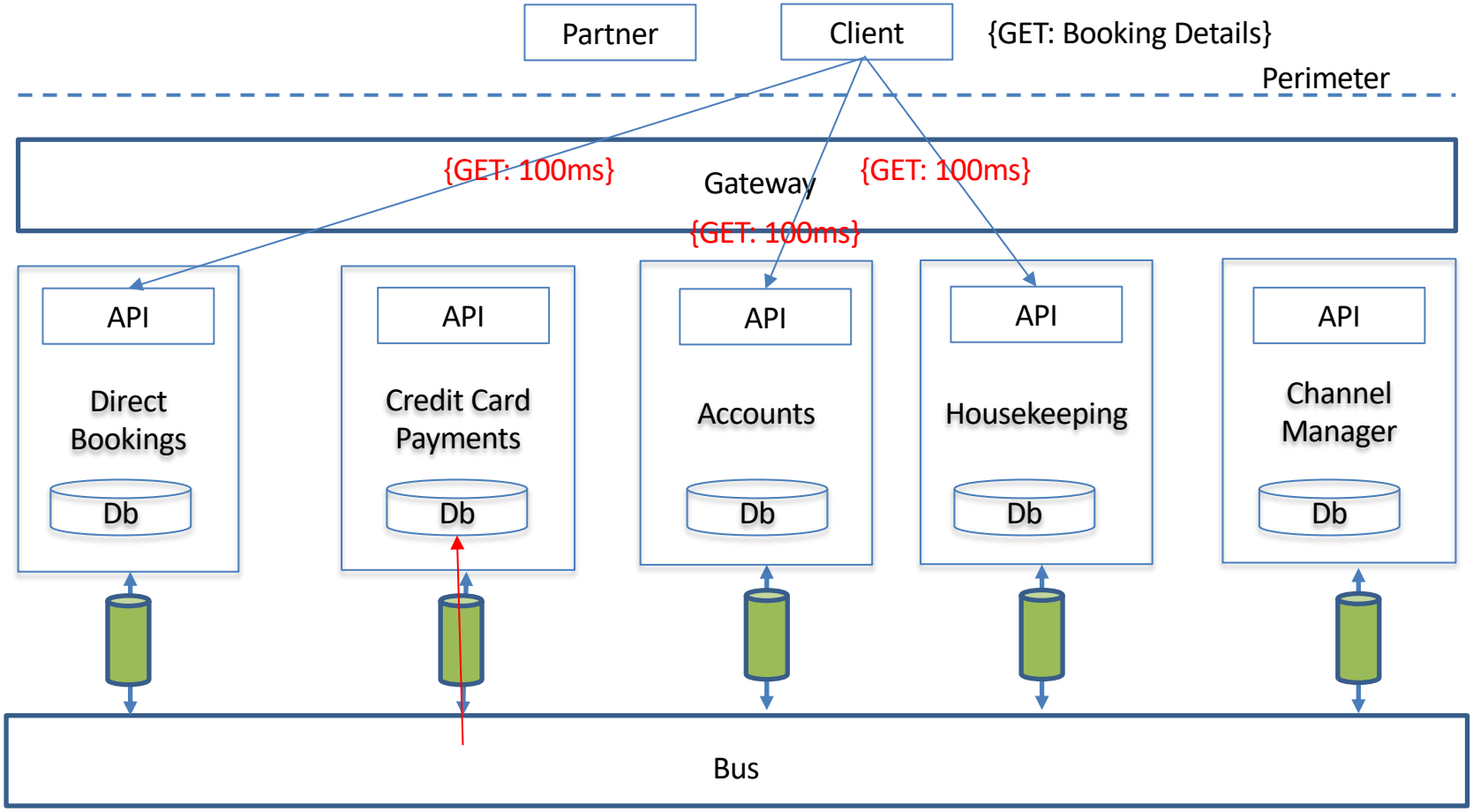
A sequence of screens can build up a request

At this point, the user expects async

Getting work done in a Monolith

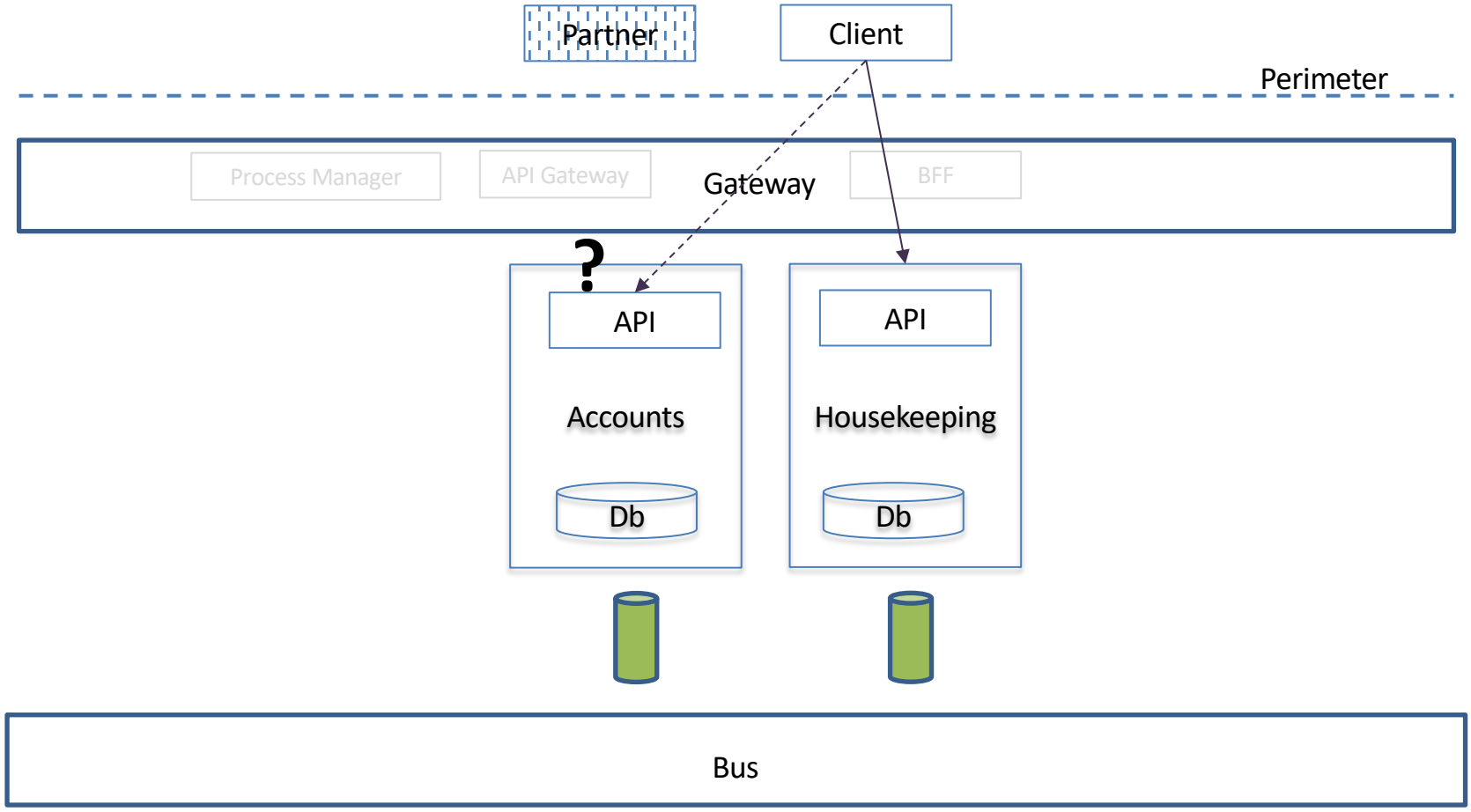



Partner
Client
Perimeter
Process Manager
API Gateway
Gateway
BFF
API
Direct Bookings
Db
API
Credit Card Payments
Db
API
Accounts
Db
API
Housekeeping
Db
API
Channel Manager
Db
Bus


Partner
Client
{GET: Booking Details}
Perimeter
{GET: 100ms}
Gateway
{GET: 100ms}
{GET: 100ms}
API
Direct Bookings
Db
API
Credit Card Payments
Db
API
Accounts
Db
API
Housekeeping
Db
API
Channel Manager
Db
Bus

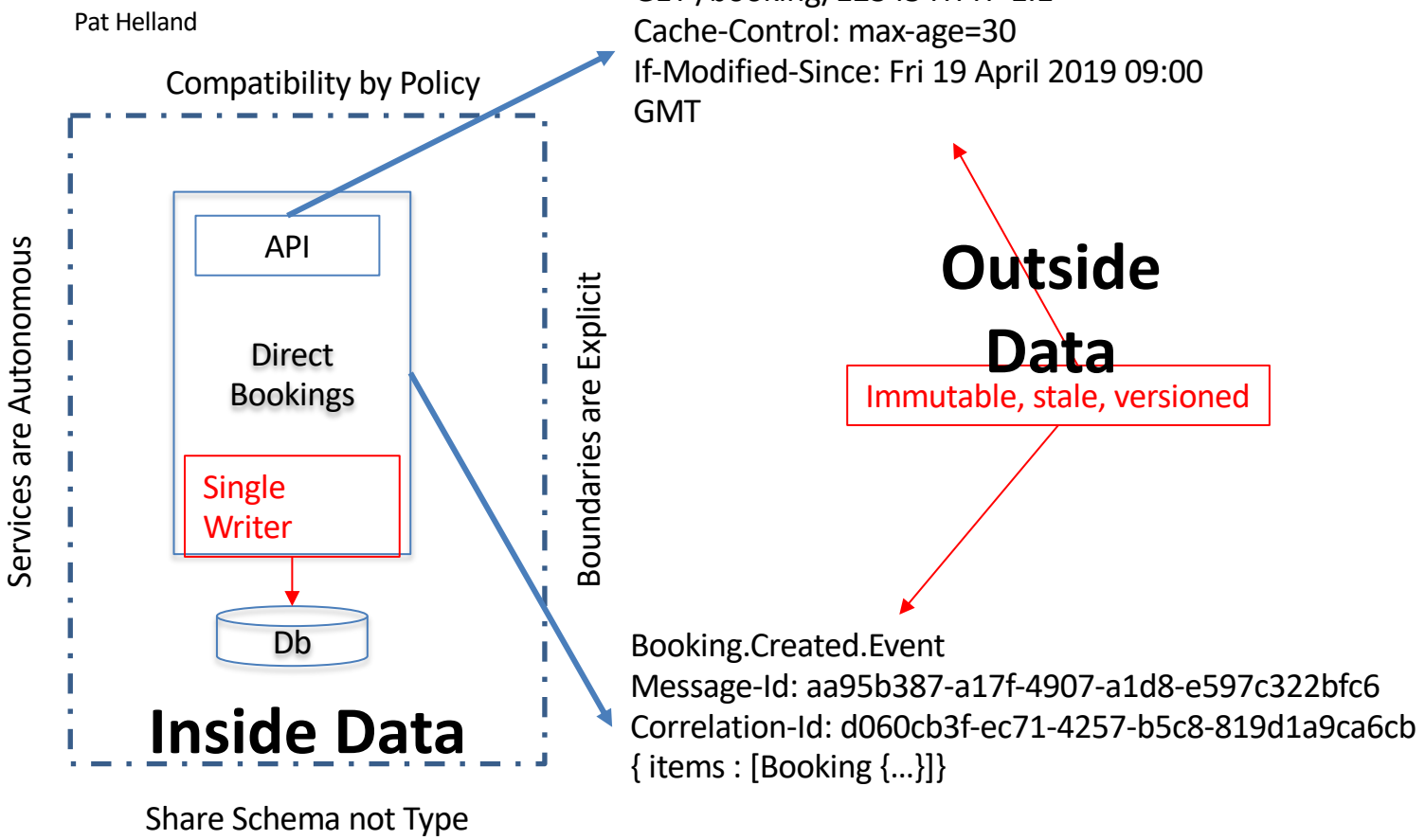
Check-In

Check-In


Partner
Client
Perimeter
Process Manager
API Gateway
Gateway
BFF
?
API
Accounts
Db
API
Housekeeping
Db
Bus

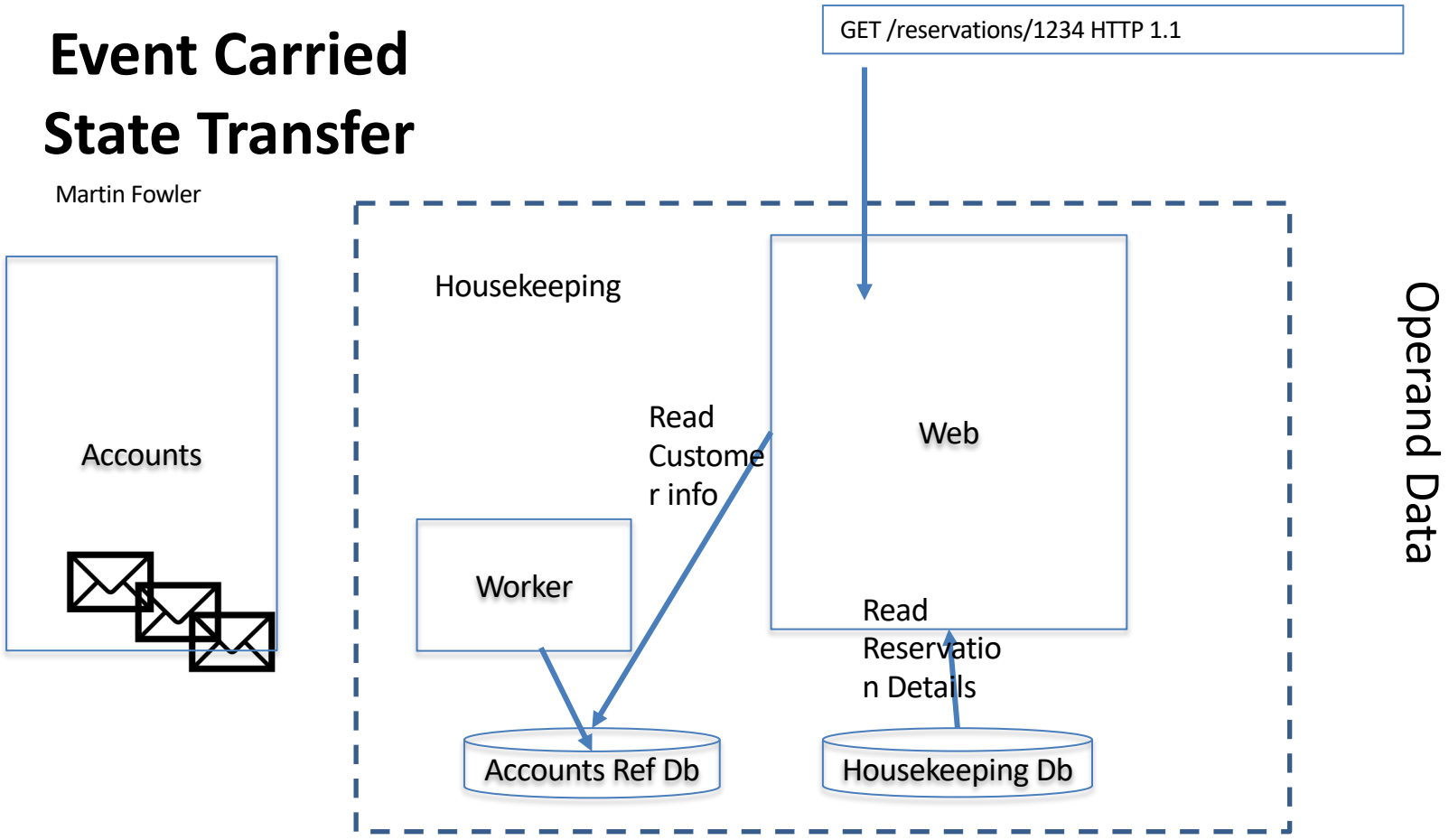
Reference Data

Pat Helland



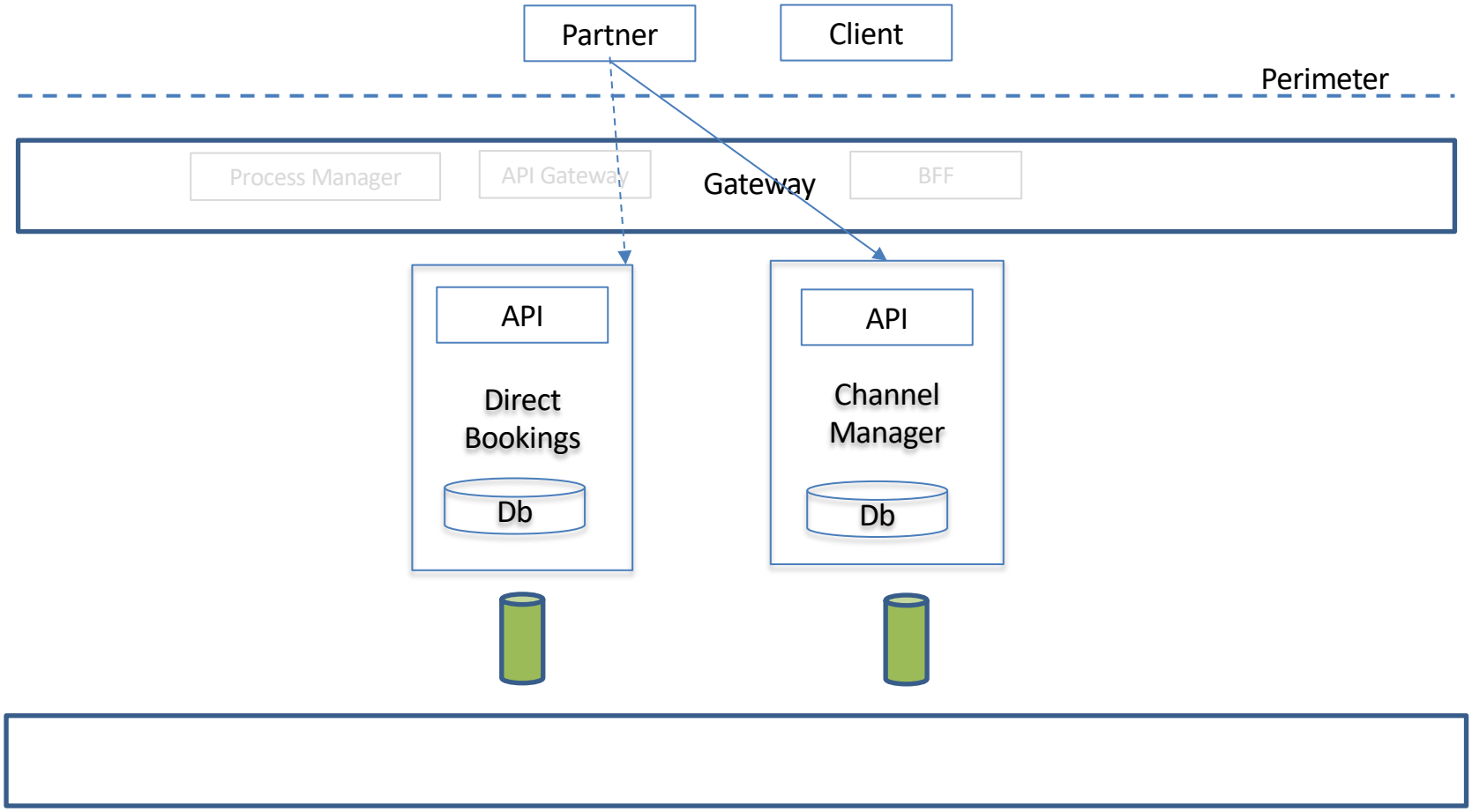
Event Carried State Transfer

Martin Fowler



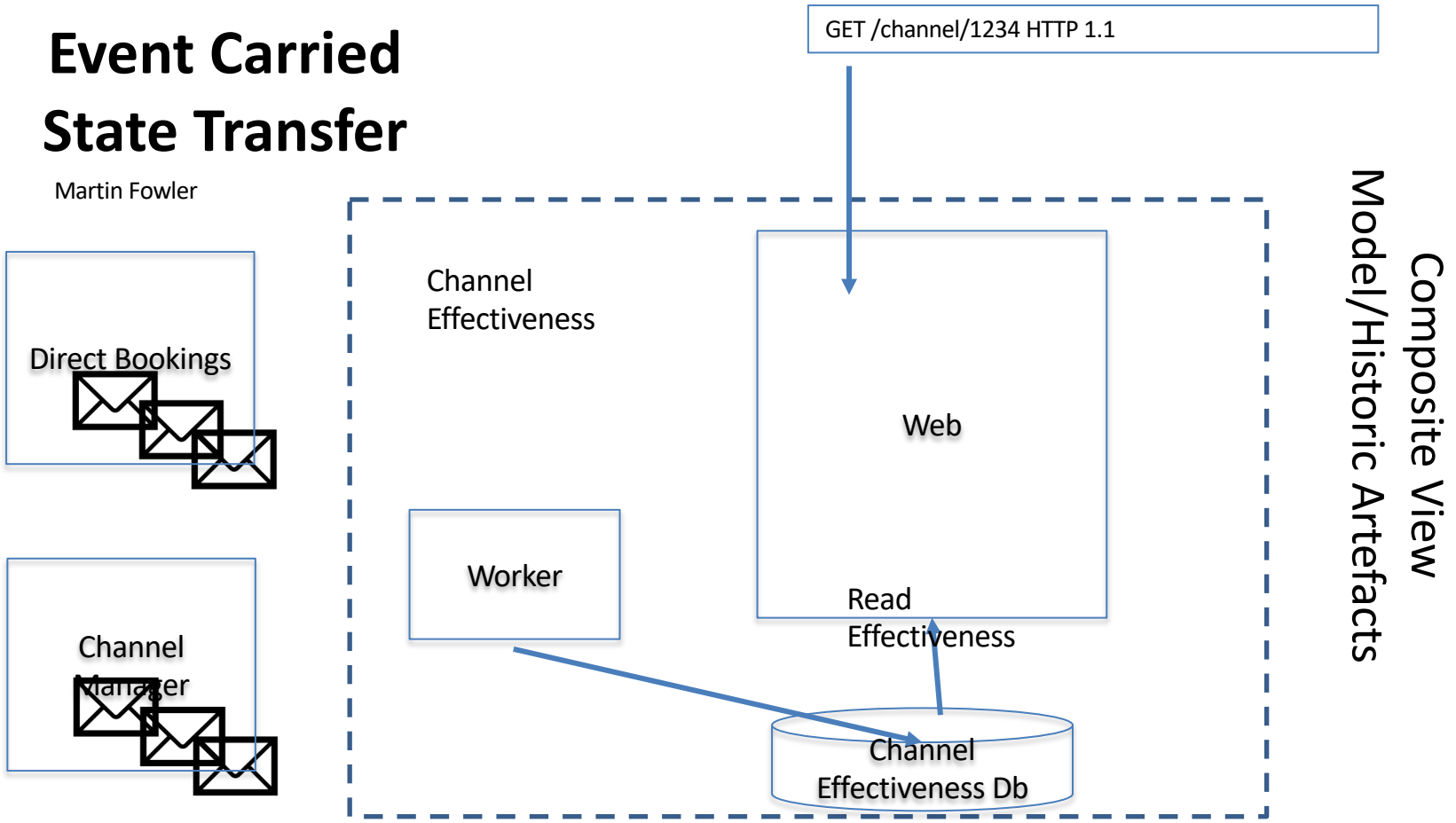
Channel Effectiveness

Channel Effectiveness



Event Carried State Transfer

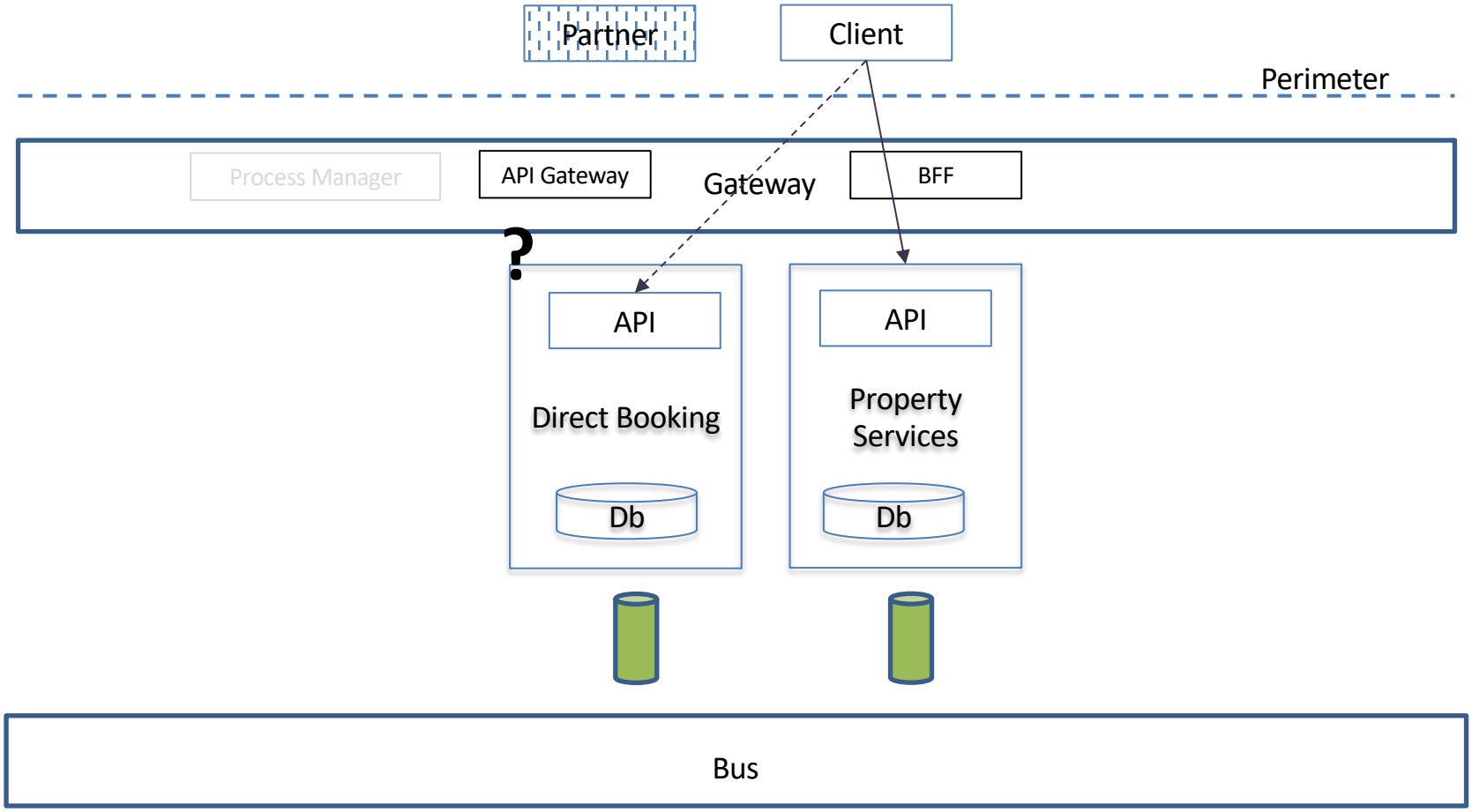
Martin Fowler



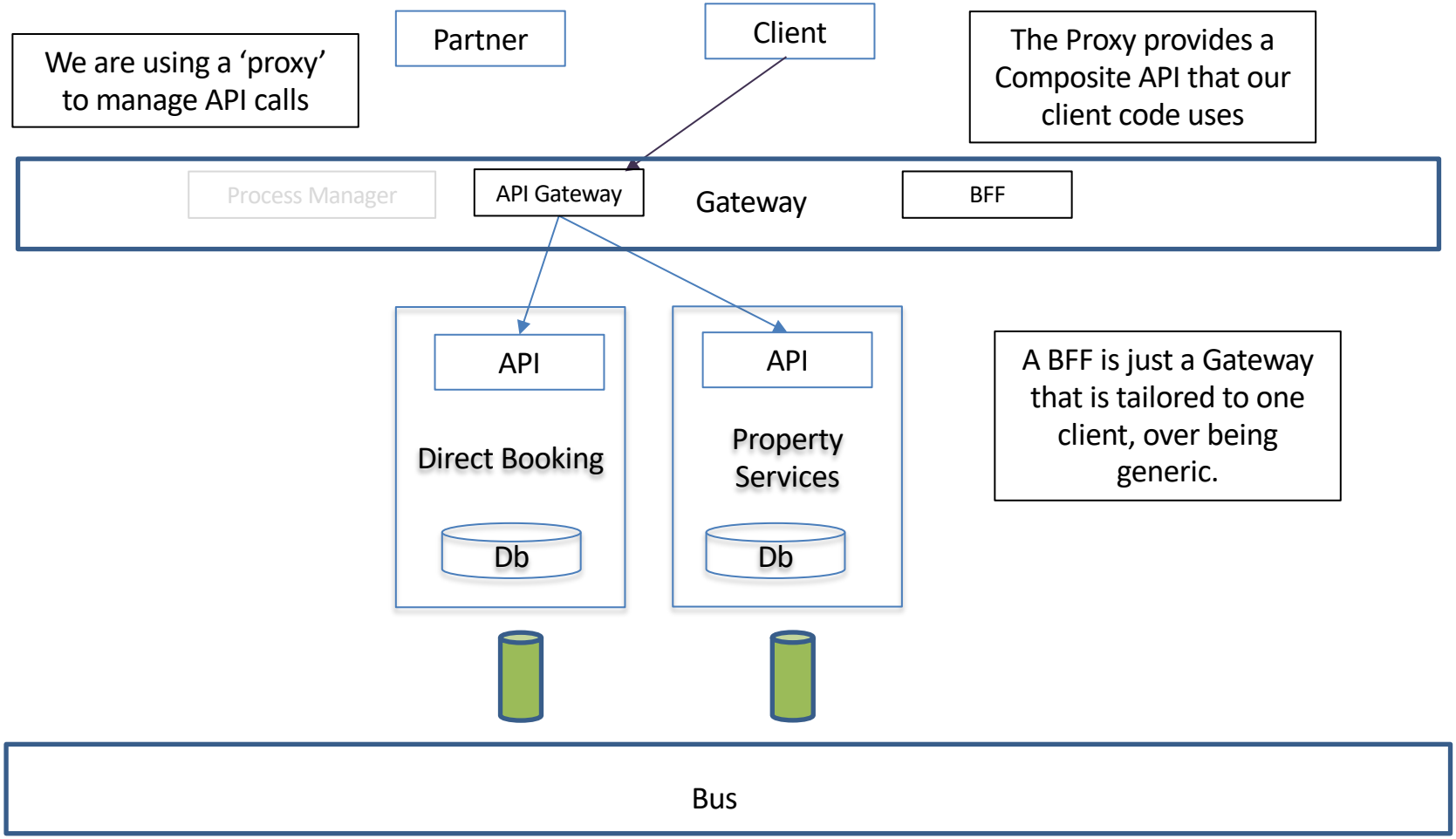
Composite View
Model/Historic Artefacts

Hotel Group Search

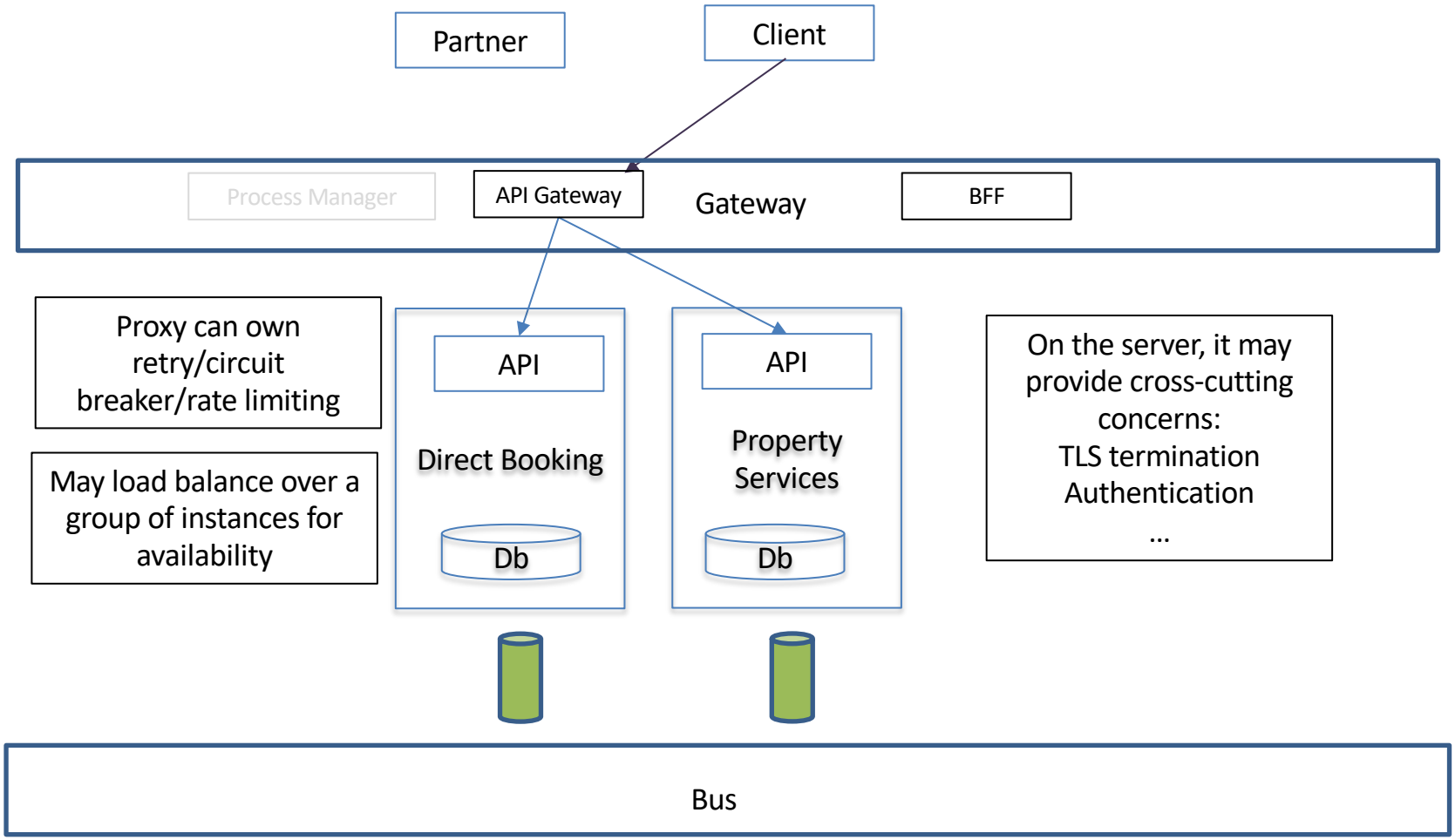
Group Room Search



Composition

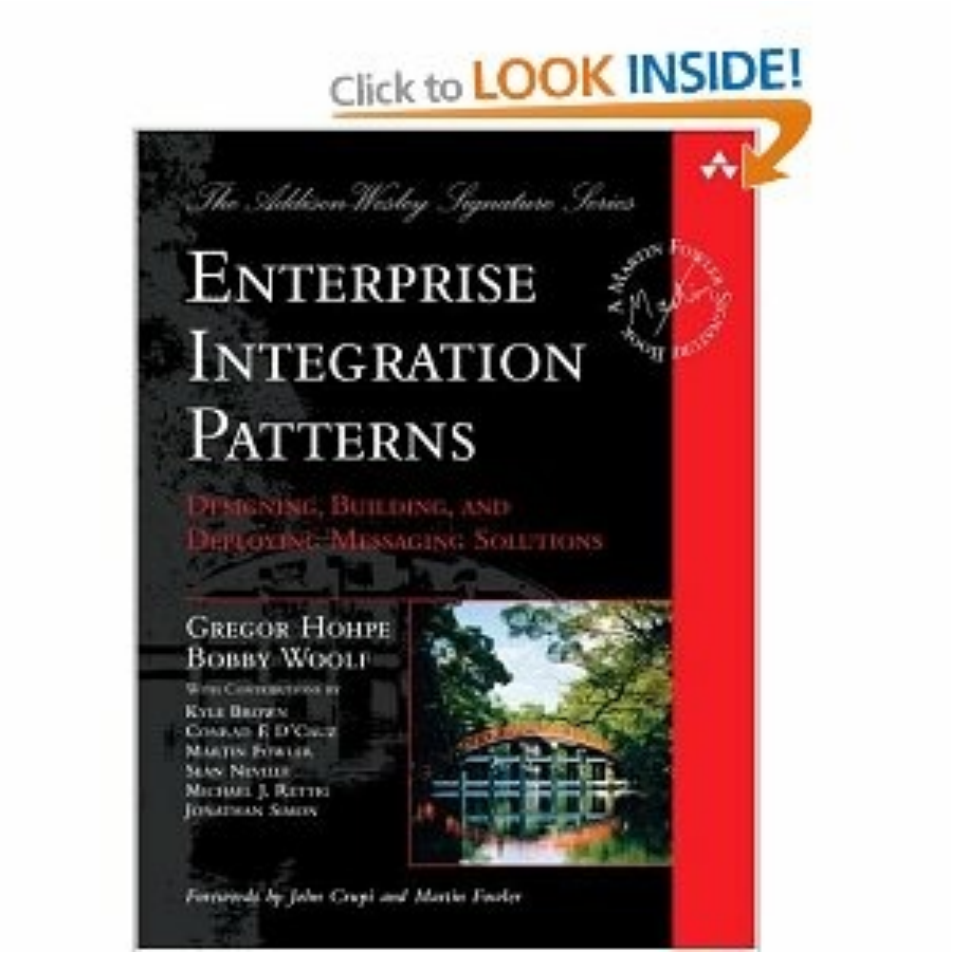


Composition



8. NEXT STEPS

Further Reading



Q&A