Practical Messaging

A 101 guide to messaging

Ian Cooper

Twitter: ICooper

Who are you?

- Software Developer for over 20 years
  - Worked mainly for ISVs
    - Reuters, SunGard, Misys, Huddle
  - Worked for a couple of MIS departments
    - DTI, Beazley
- Microsoft MVP for C#
  - Interested in OO design
  - Interested in Agile methodologies and practices
- No smart guys
  - Just the guys in this room

Day Two Agenda

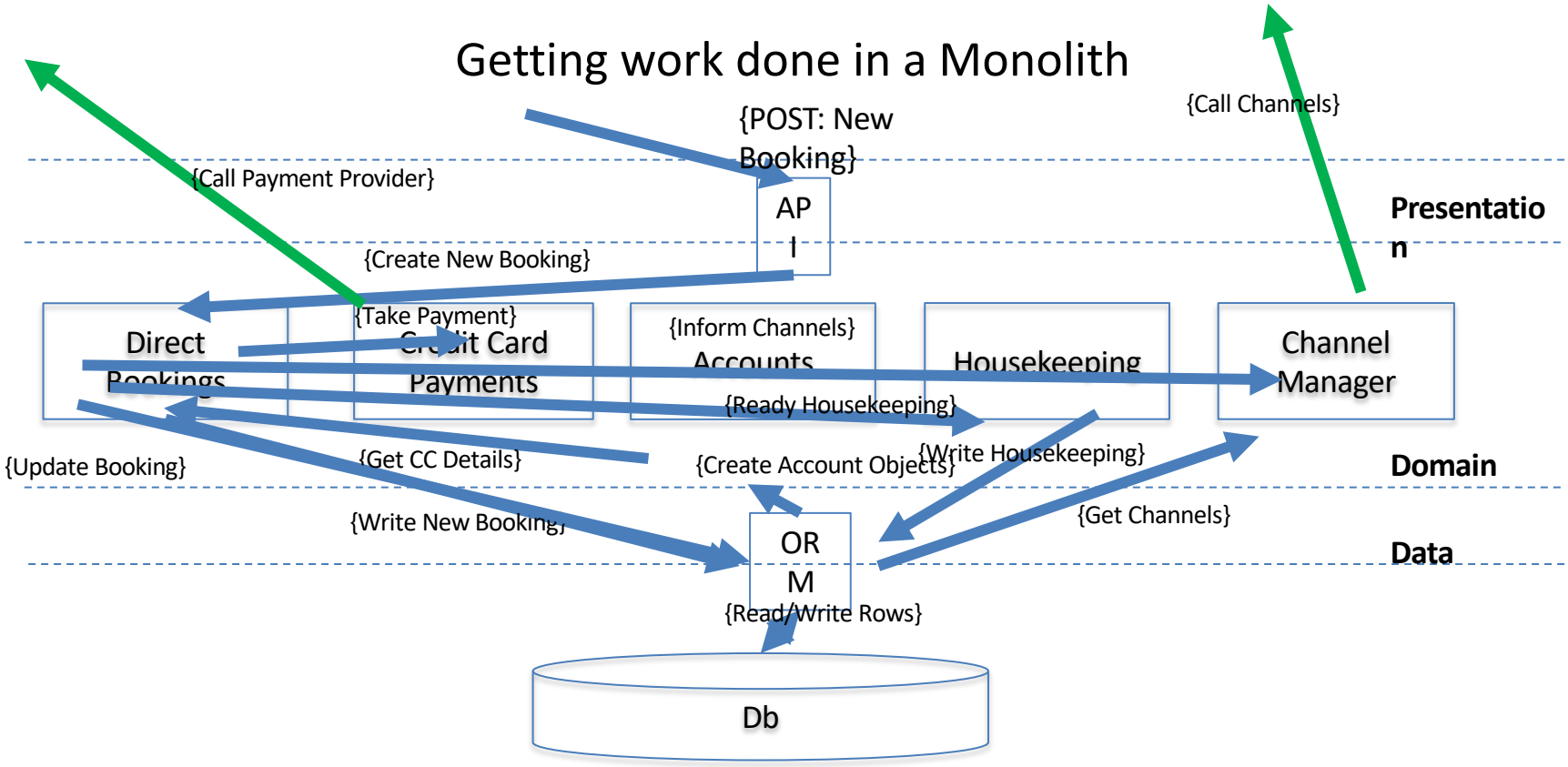
- Event Driven Collaboration
- Reliability
- Order
- Consumers
- Documentation
- Middleware & Frameworks Discussion (Separate Decks)

Day Two

Event Carried State Transfer

3.1 EVENT DRIVEN COLLABORATION

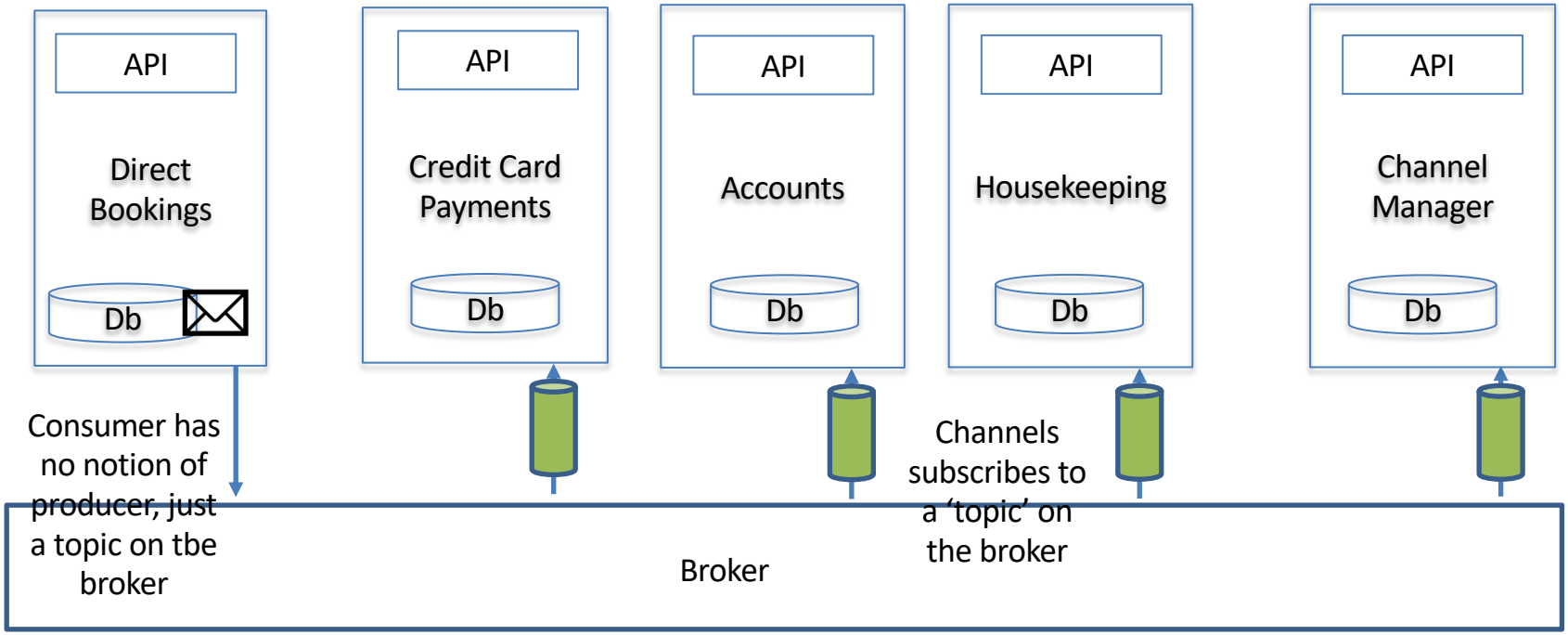
Getting work done in a Monolith



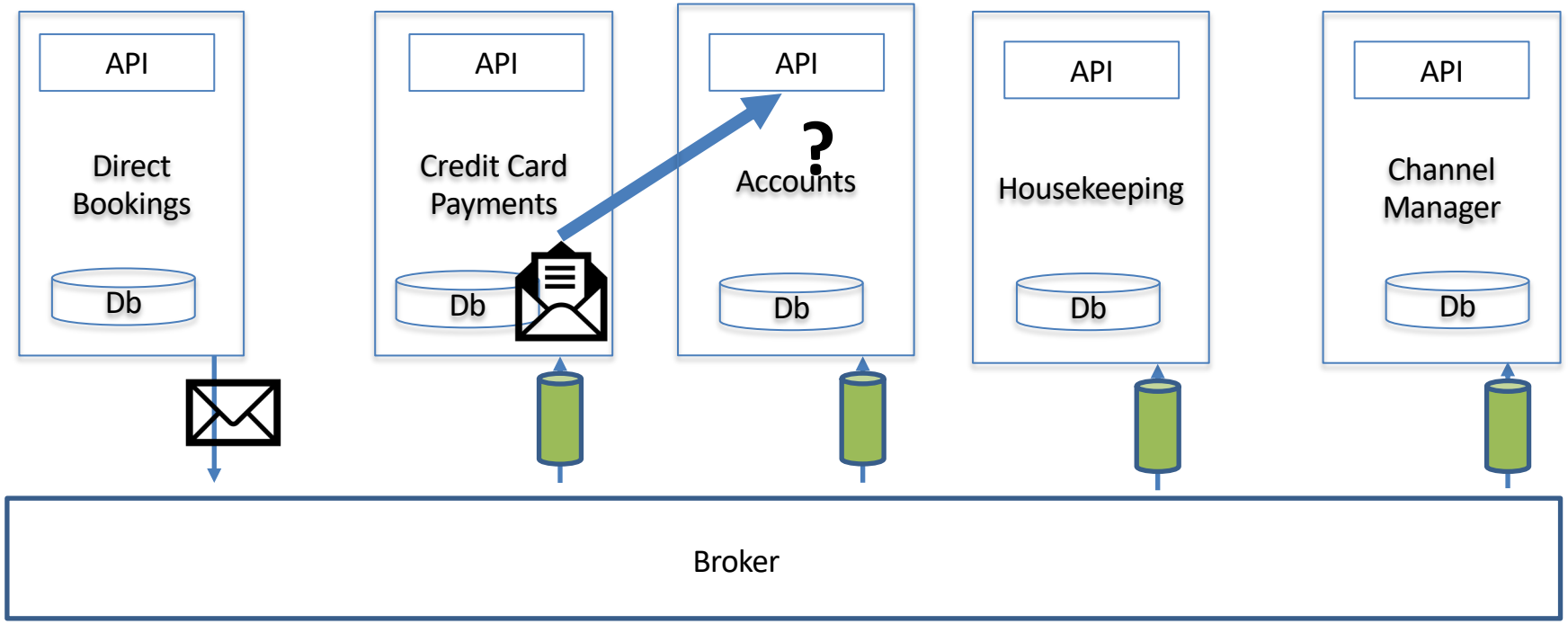
Event Driven Architecture

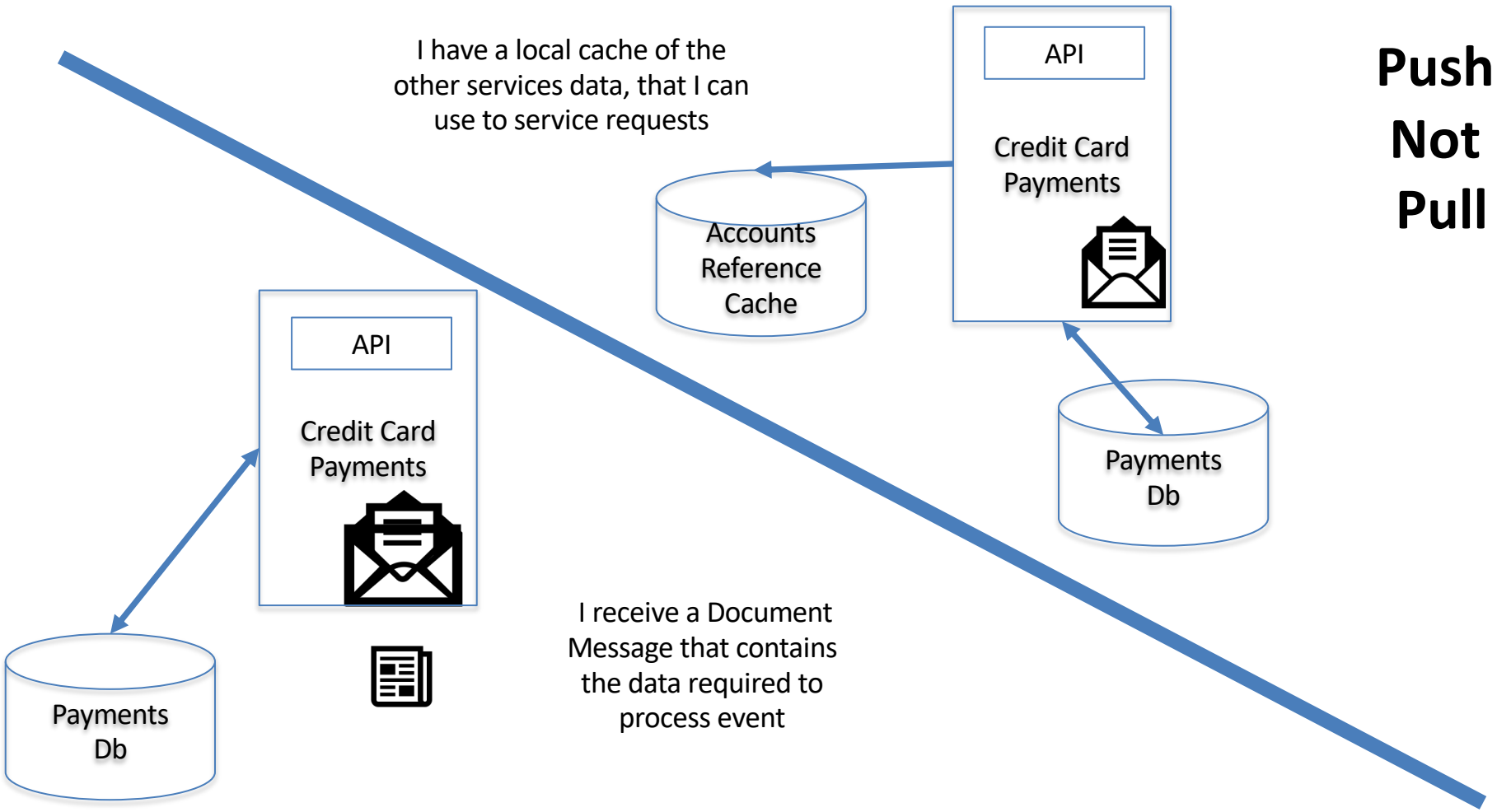
"Messaging over a lightweight message bus such as RabbitMQ"

Fowler and Lewis



Event Driven Architecture





Push Not Pull

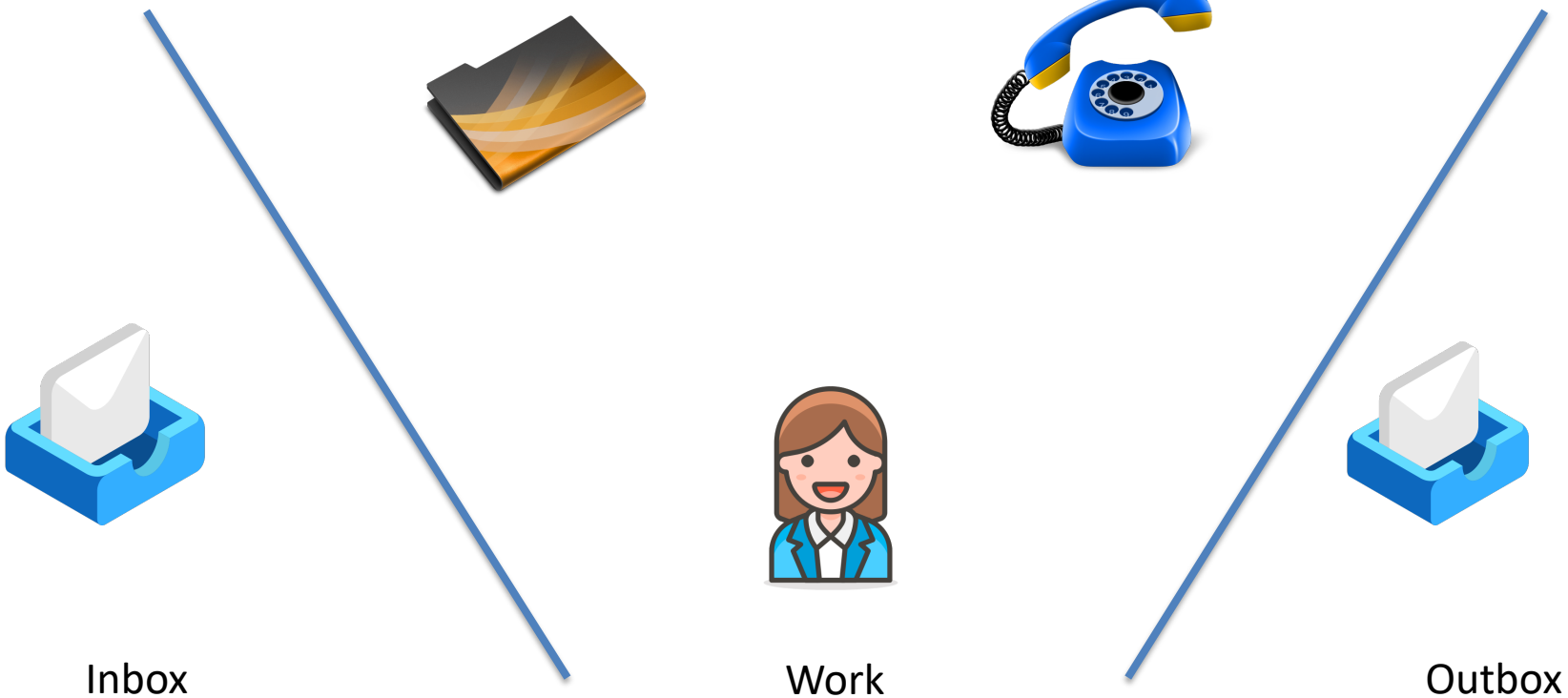
Orchestration and Choreography

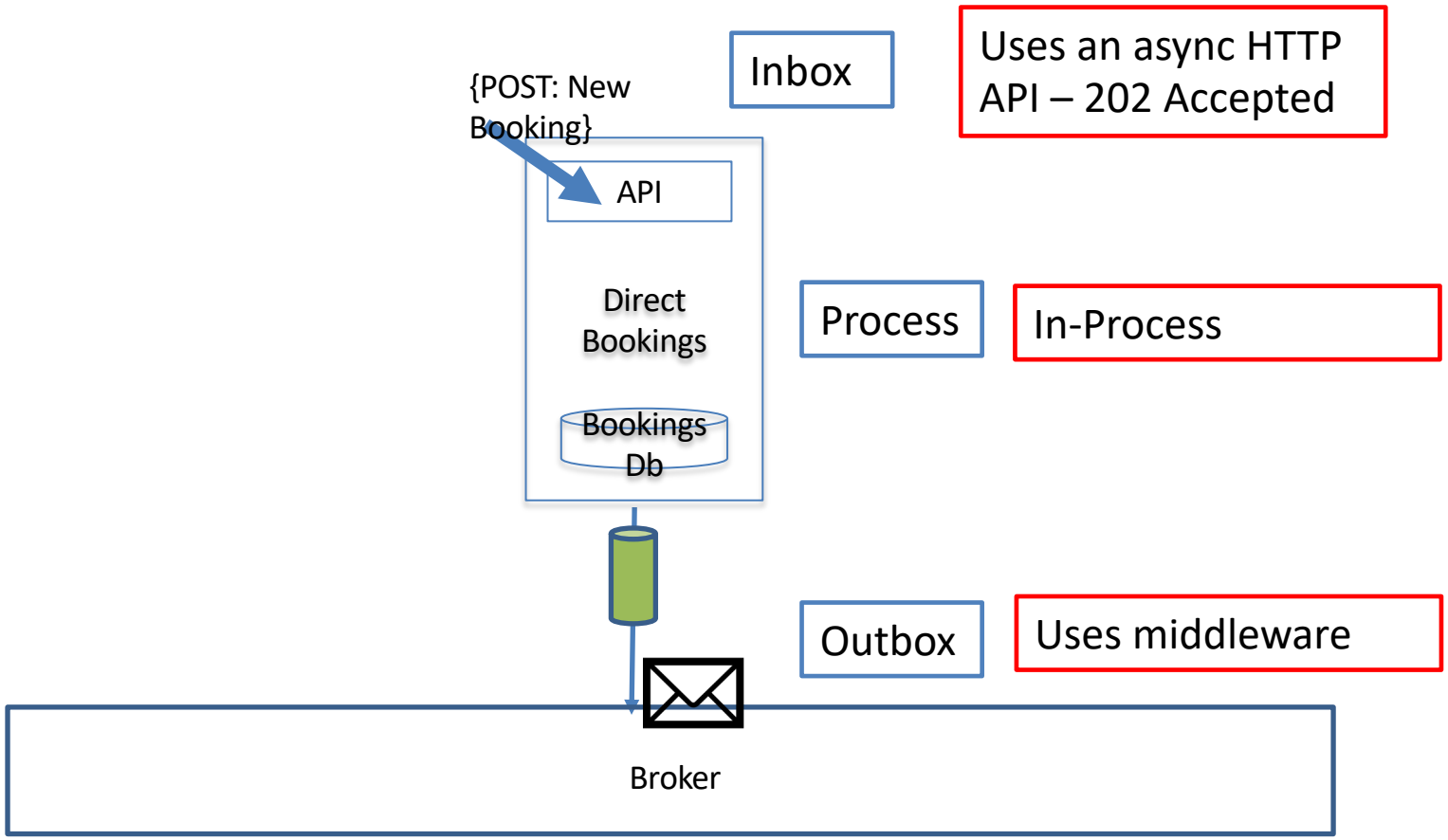
3.1.1 PIPES AND FILTERS

What is a microservice?

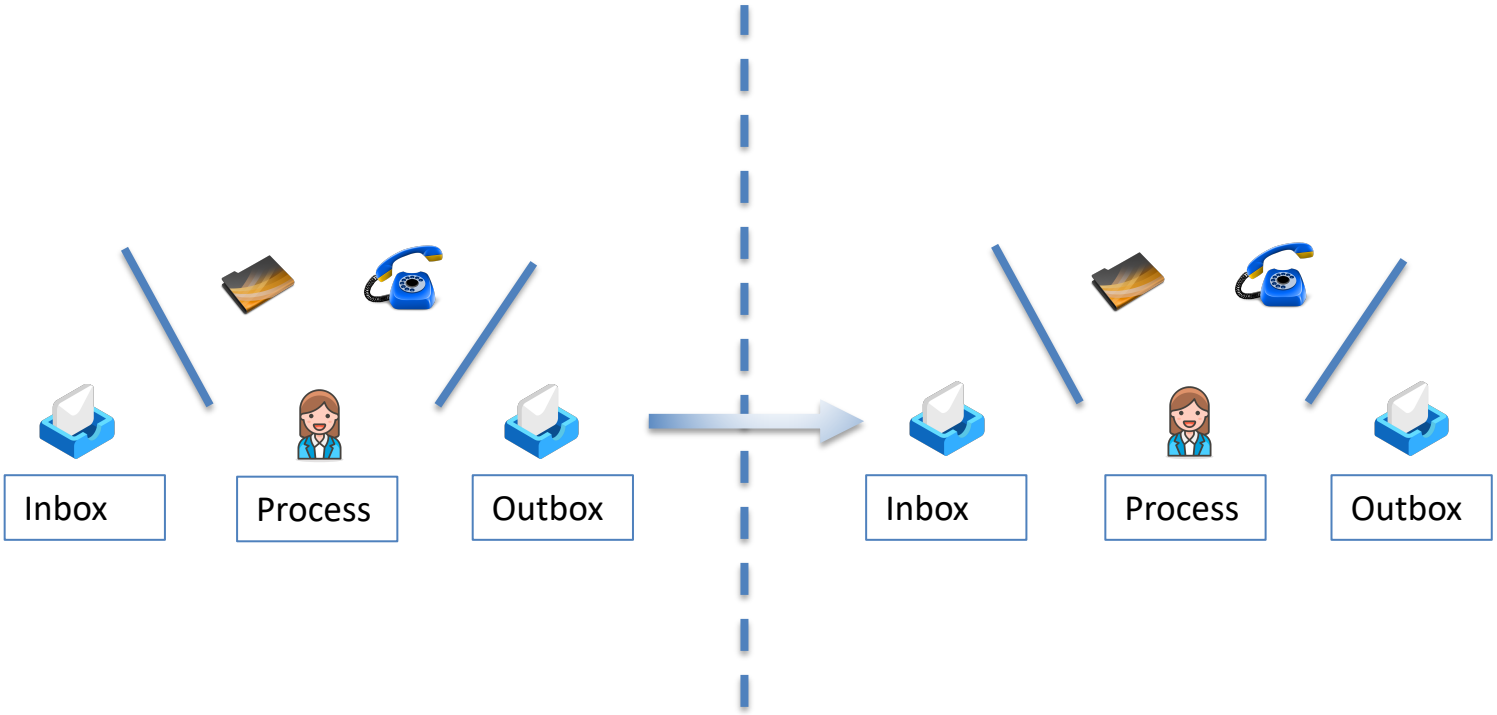
SOA is focused on business *processes*. These *processes* are performed in different steps (also called *activities* or *tasks*) on different systems. The primary goal of a **service** is to represent a "natural" step of business functionality. That is, according to the domain for which it's provided,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


Inbox
Work
Outbox
Async
Sync
Async


{POST: New Booking}
Inbox
Uses an async HTTP API – 202 Accepted
API
Direct Bookings
Bookings Db
Process
In-Process
Outbox
Uses middleware
Broker

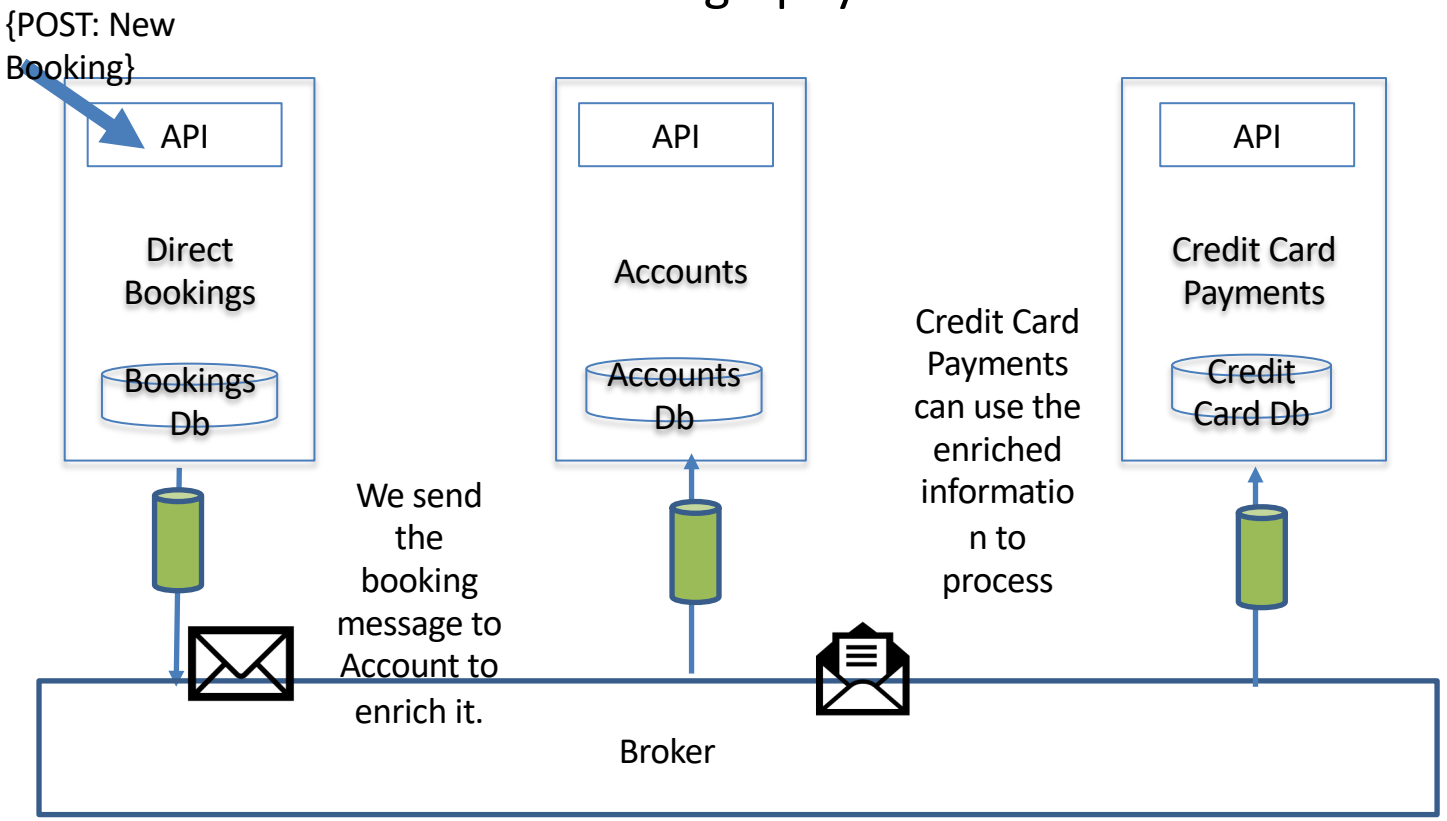
Pipeline

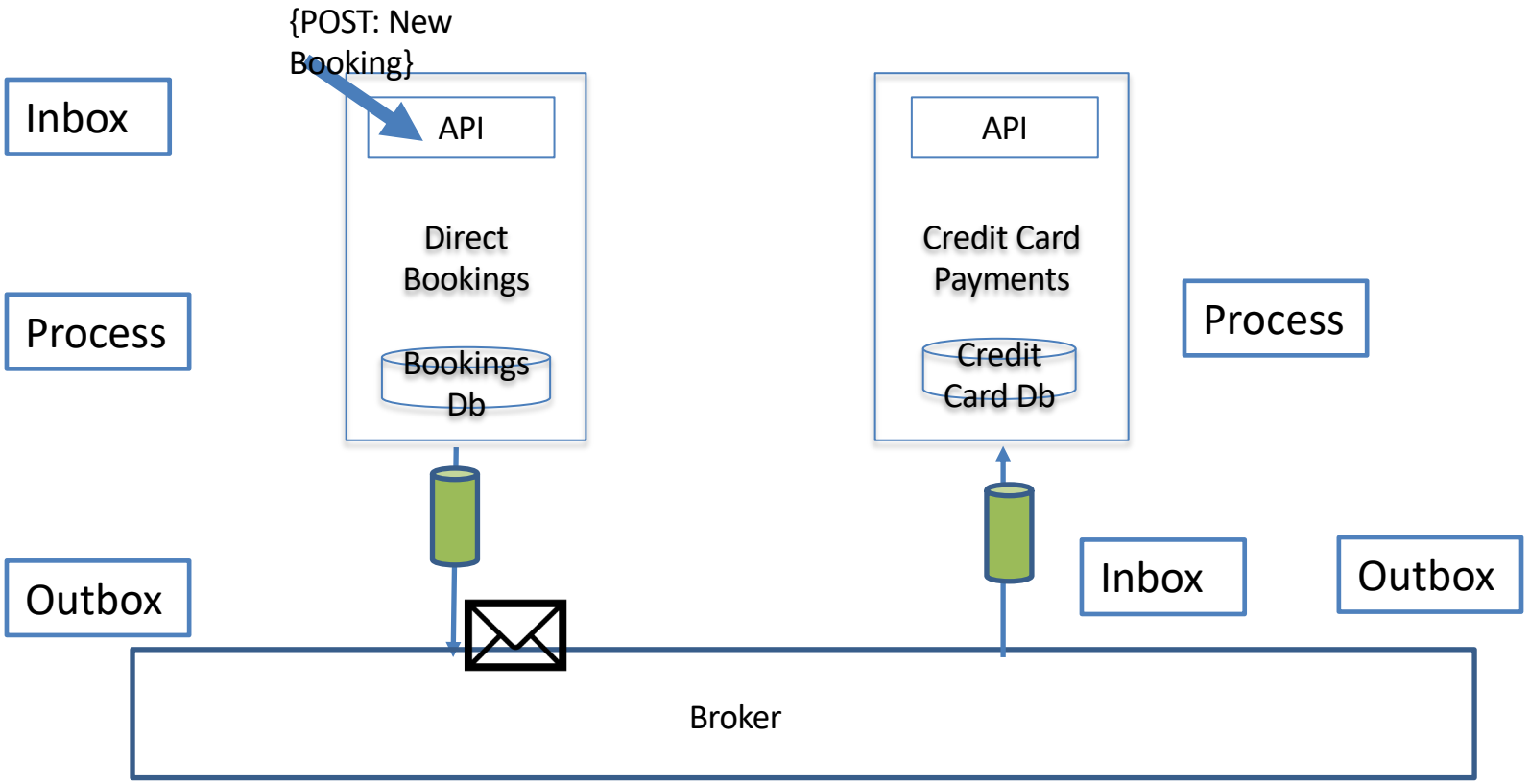


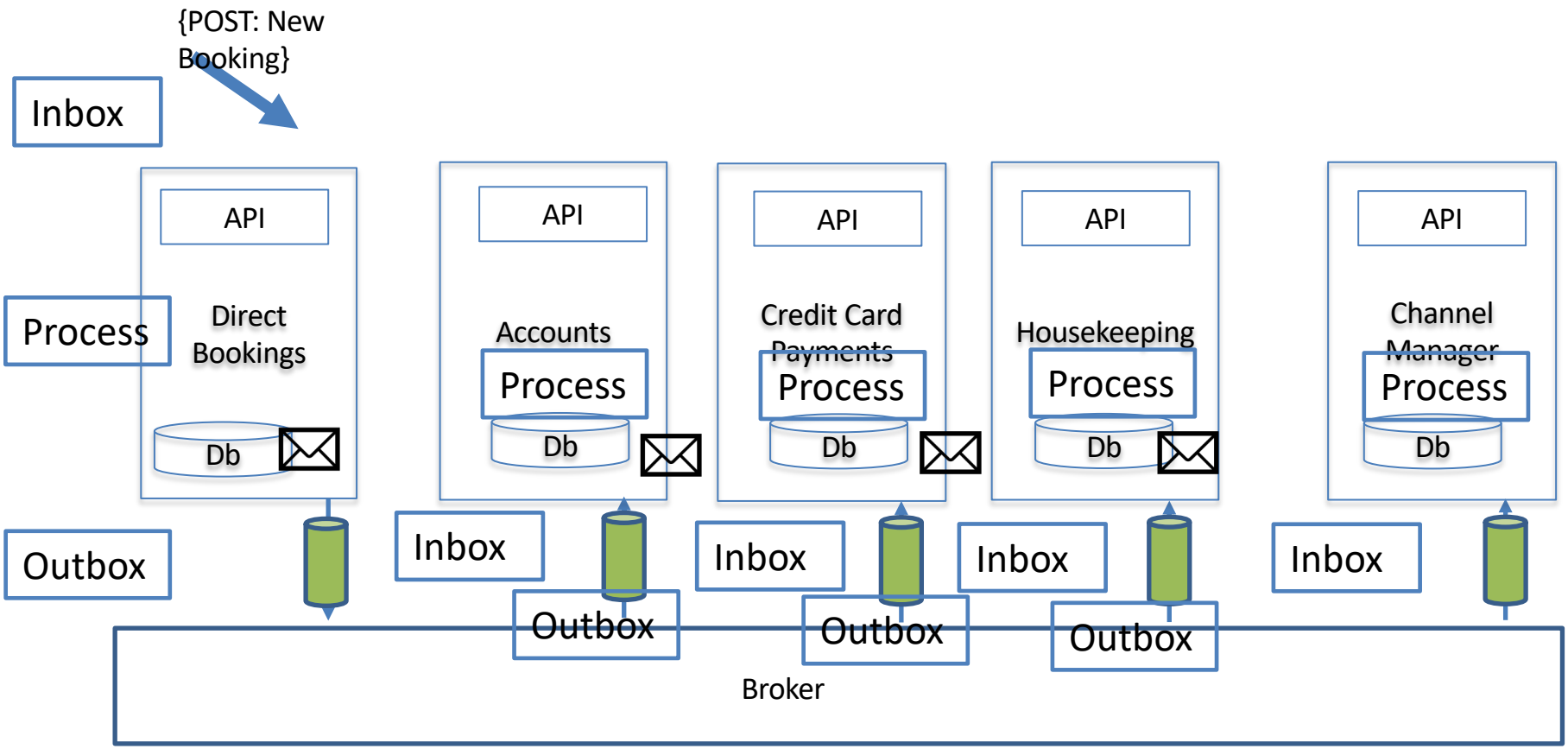
Choreography

Event-Oriented

Choreography

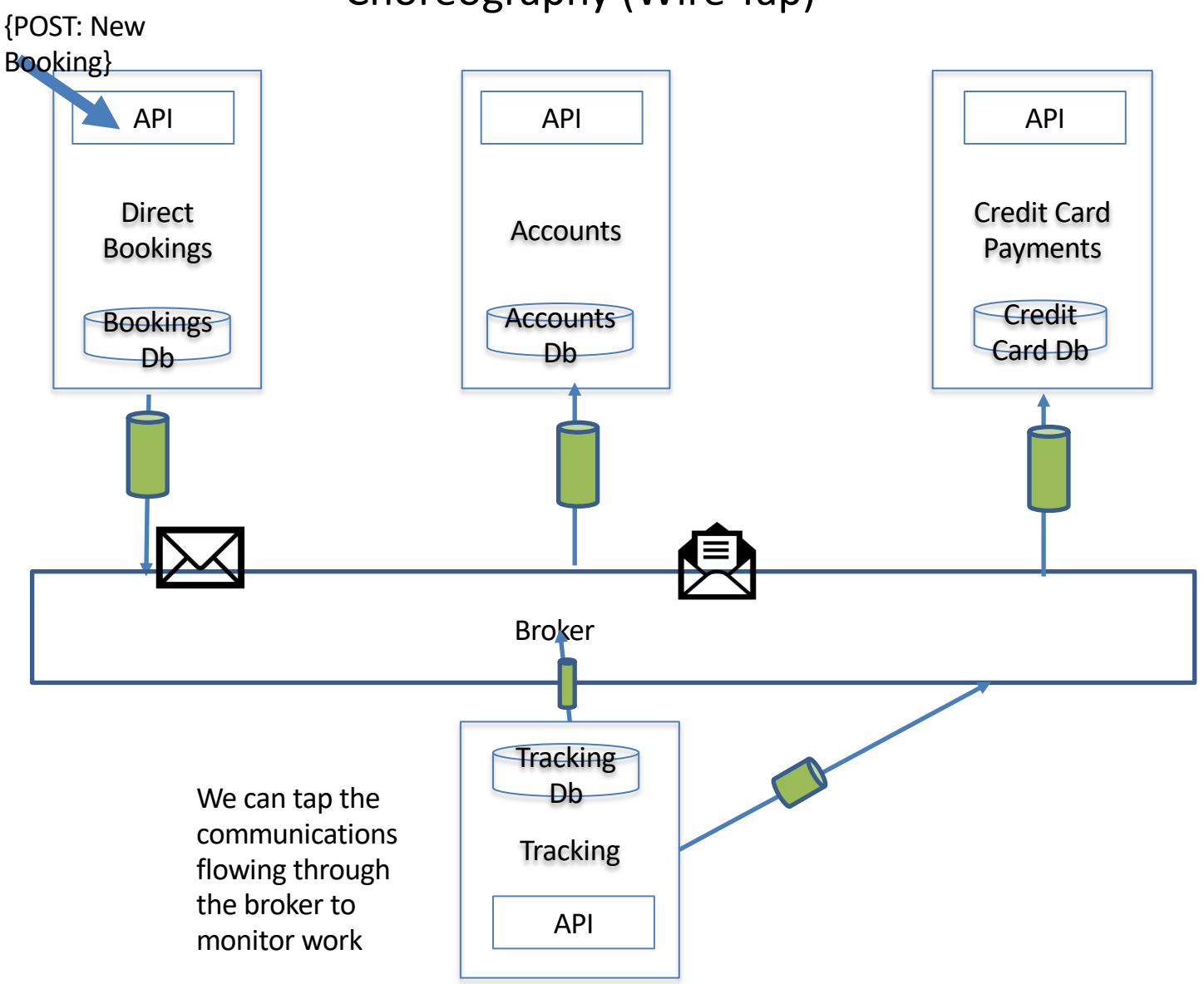



{POST: New Booking}
Inbox
Process
Outbox
API
Direct Bookings
Bookings Db
API
Credit Card Payments
Credit Card Db
Process
Inbox
Outbox
Broker


{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

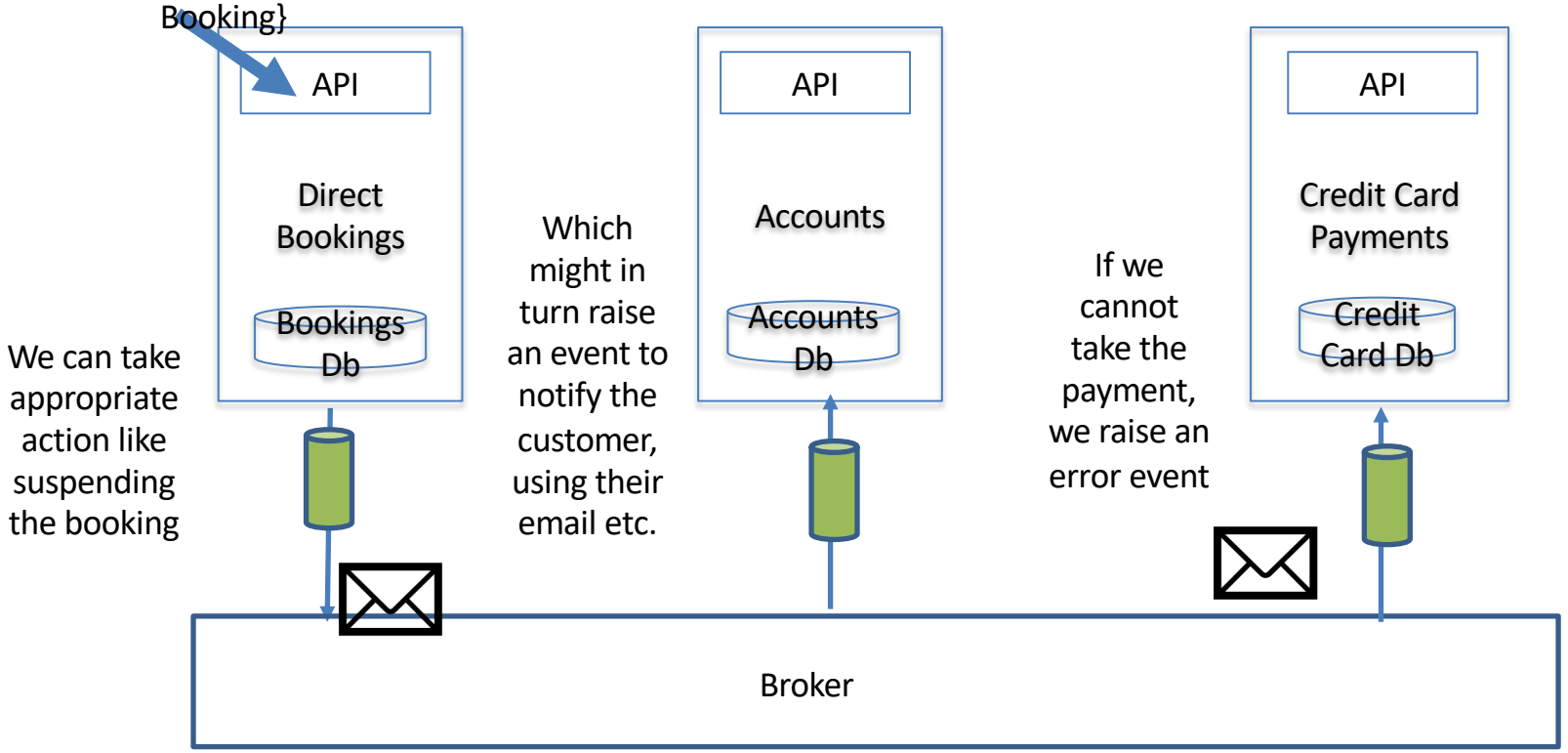
Choreography (Wire Tap)



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

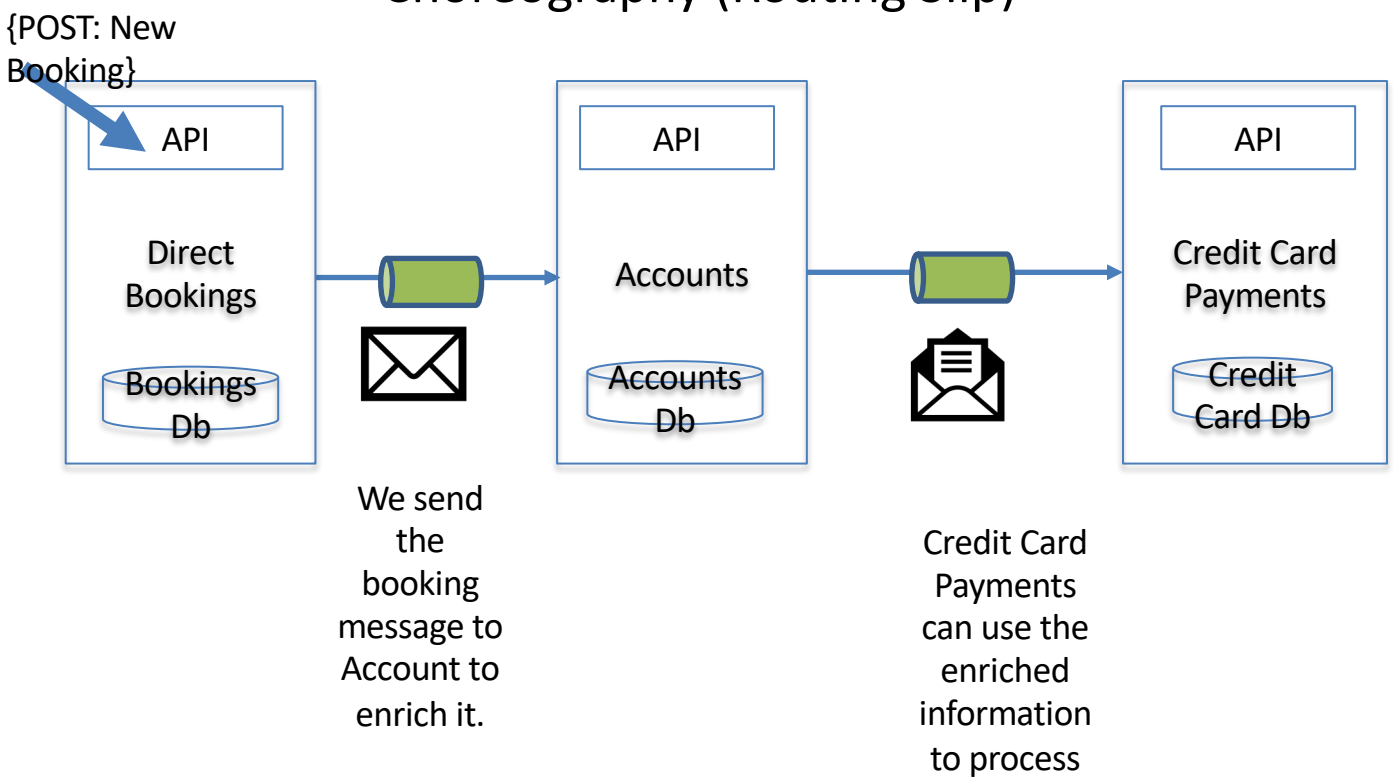
Event-Oriented

Choreography (Errors)

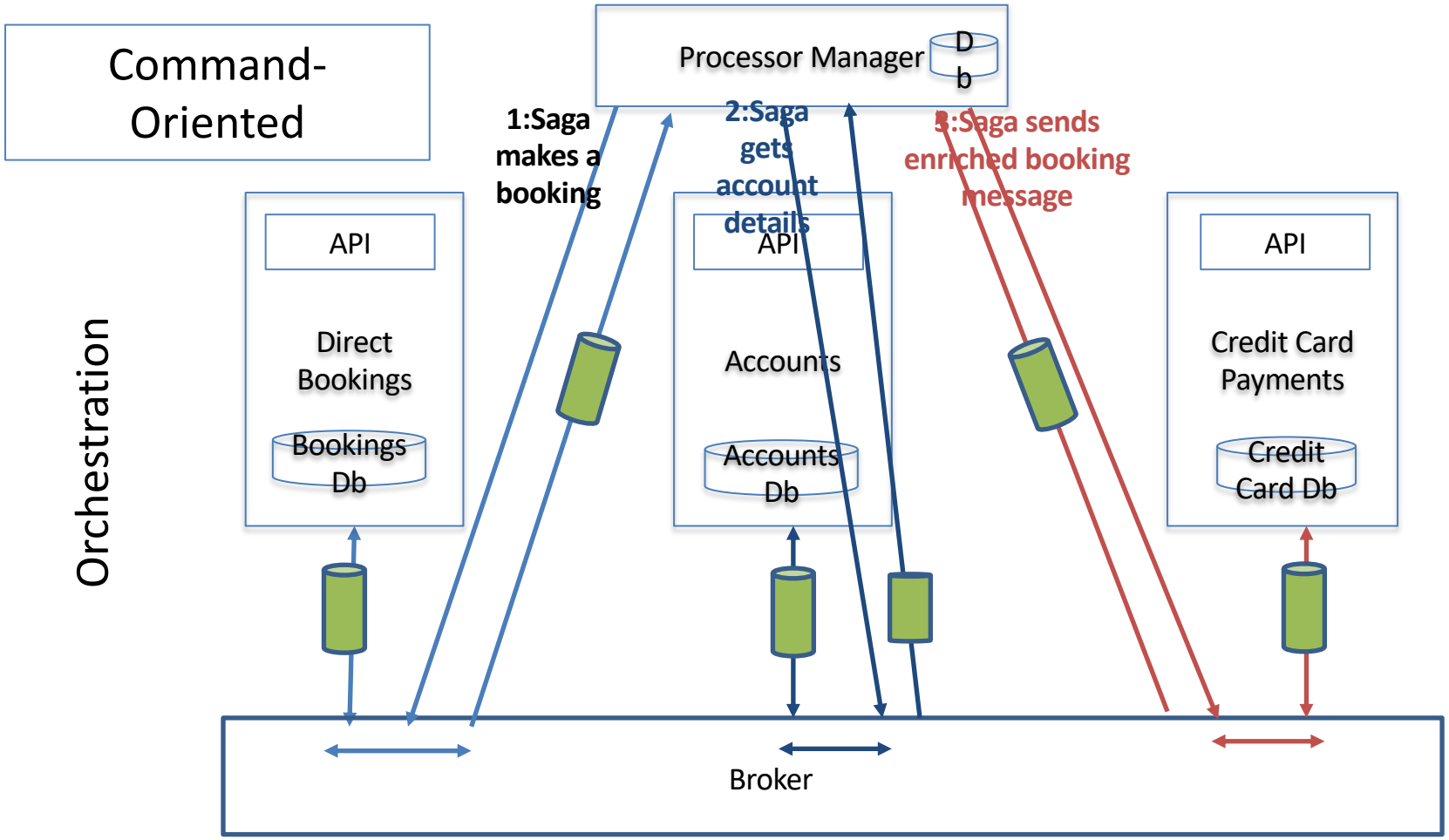


Event-Oriented

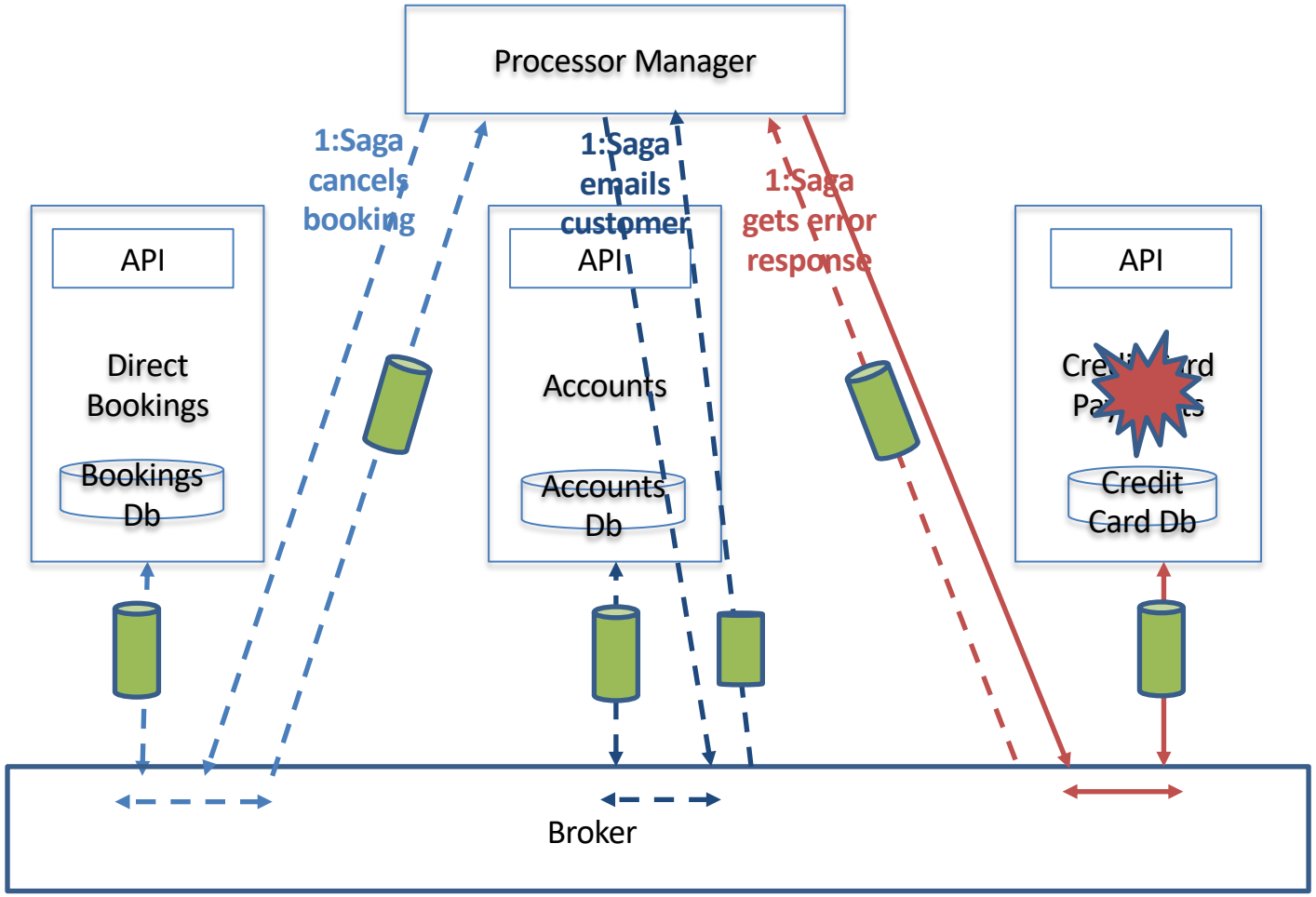
Choreography (Routing Slip)



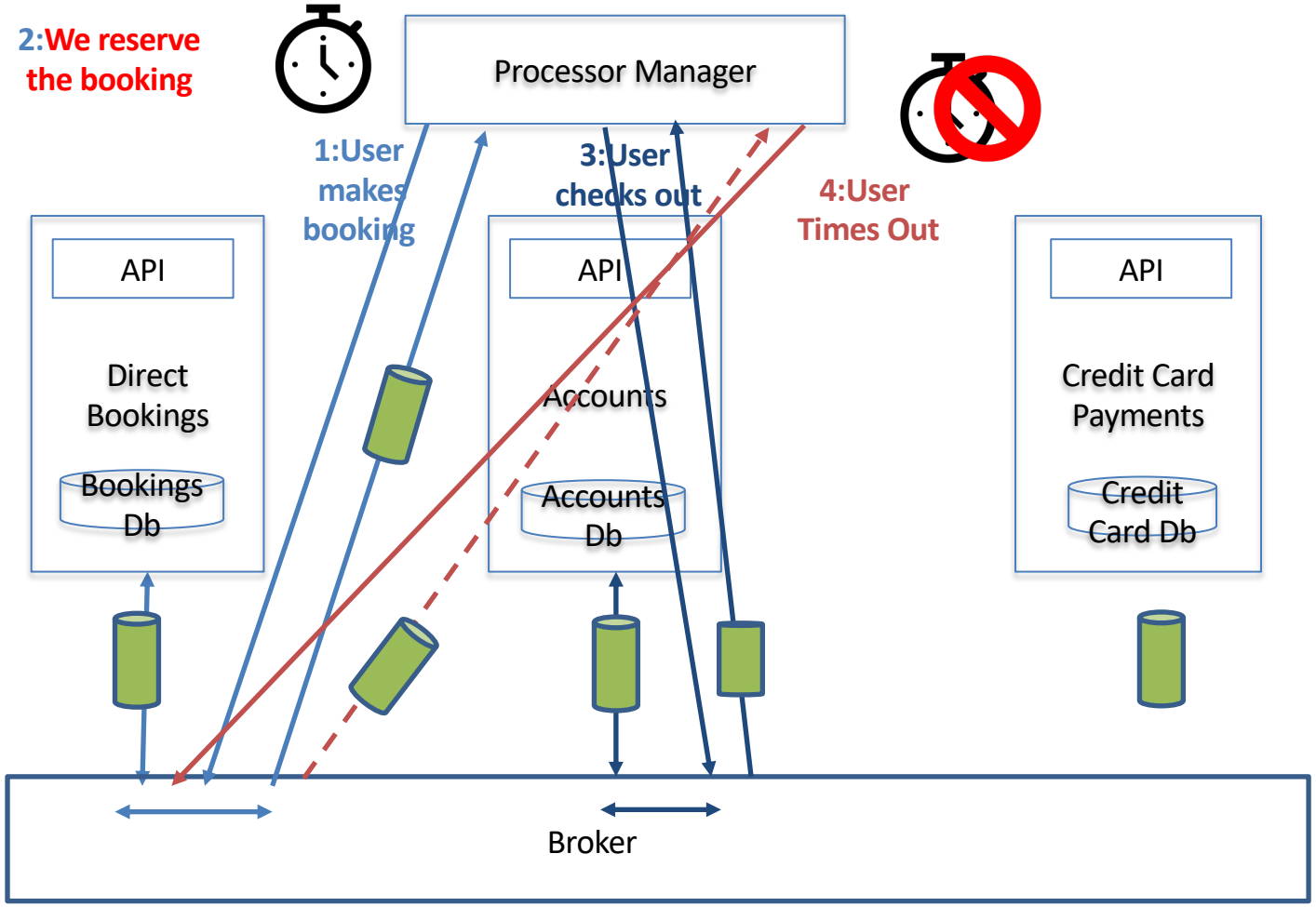
Orchestration


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)

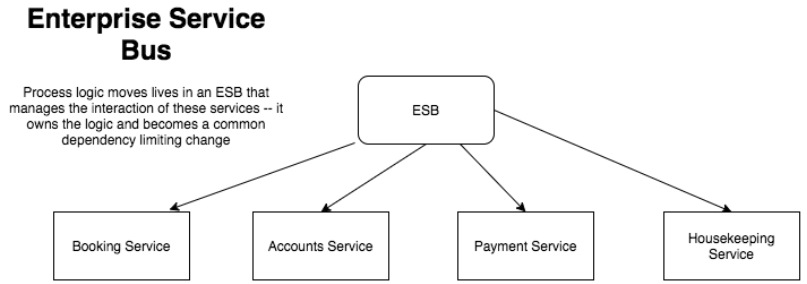


Reservations



Choreography ? Orchestration

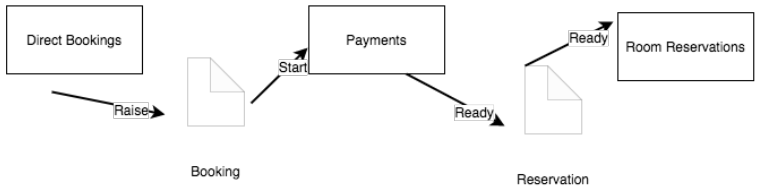
BAD



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

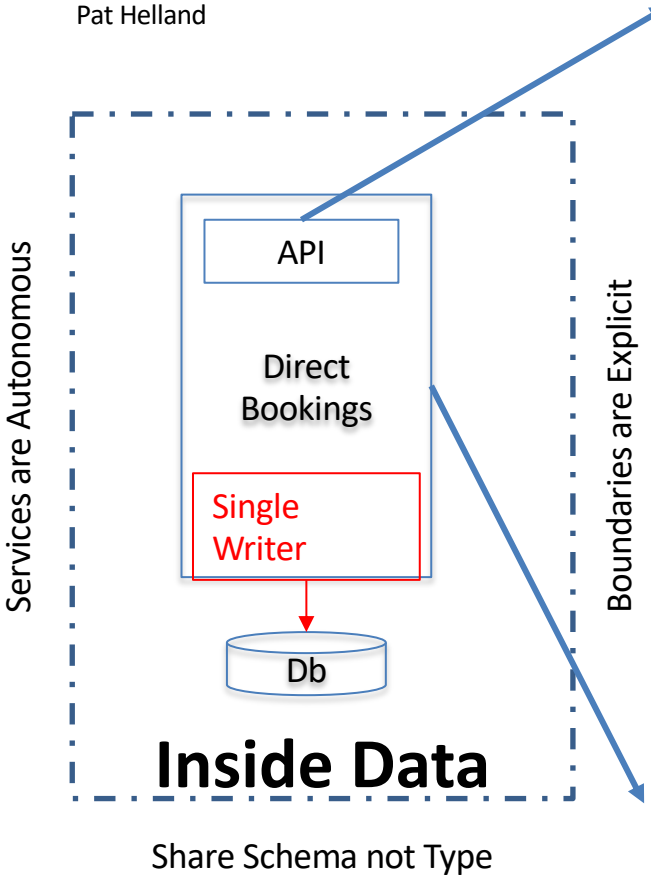
Smart Endpoints and Dumb Pipes

=Reference Data

3.1.2 EVENT CARRIED STATE TRANSFER

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”

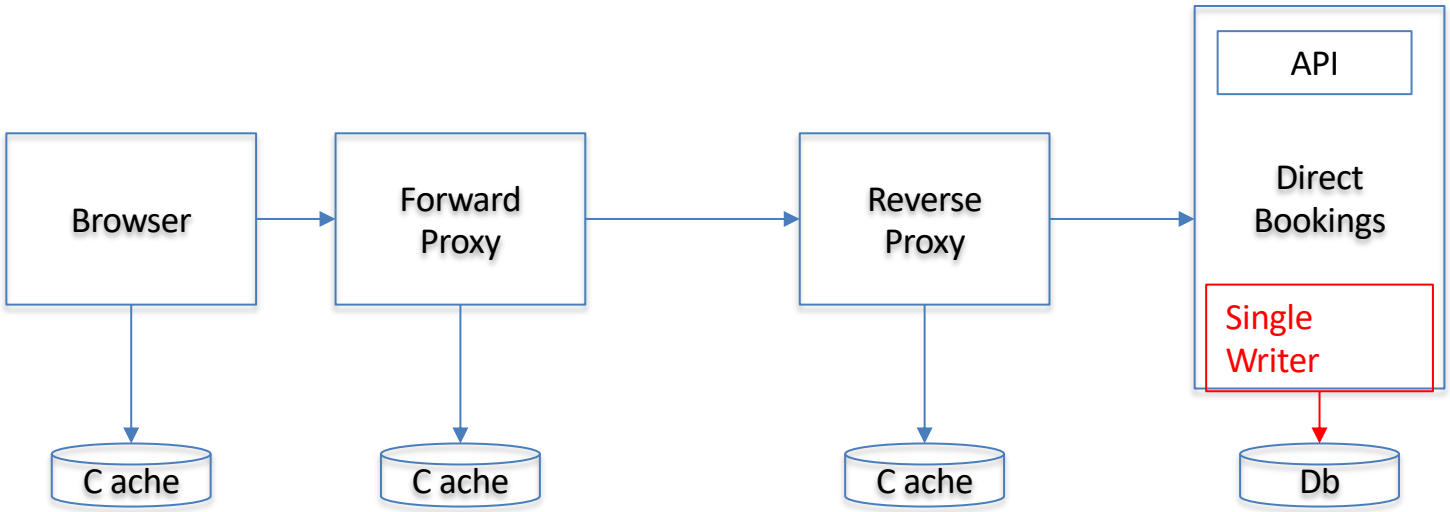
Outside Data

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

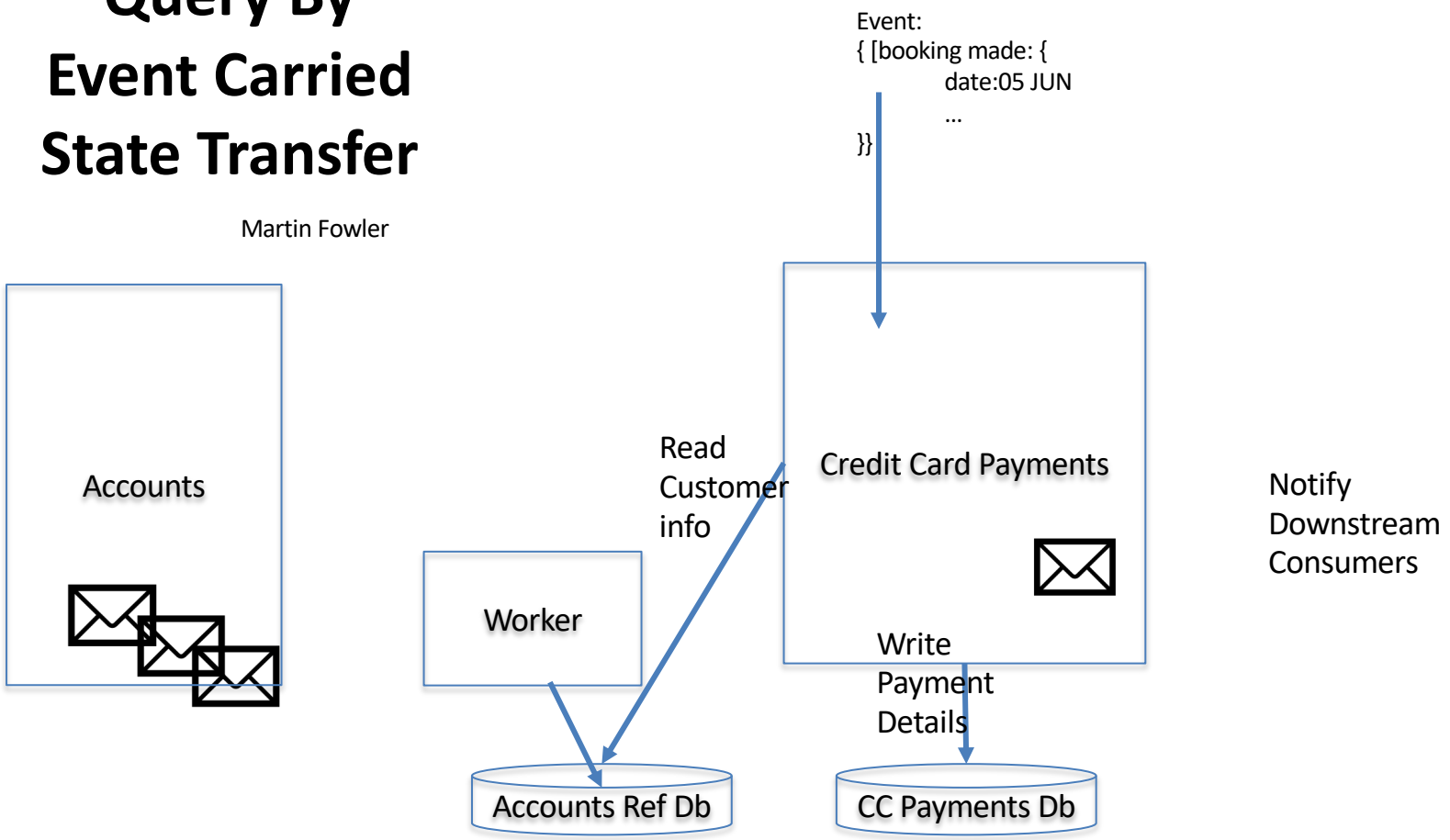
GET /booking/12345 HTTP 1.1
If-Modified-Since: Fri 19 April 2019 09:00 GMT



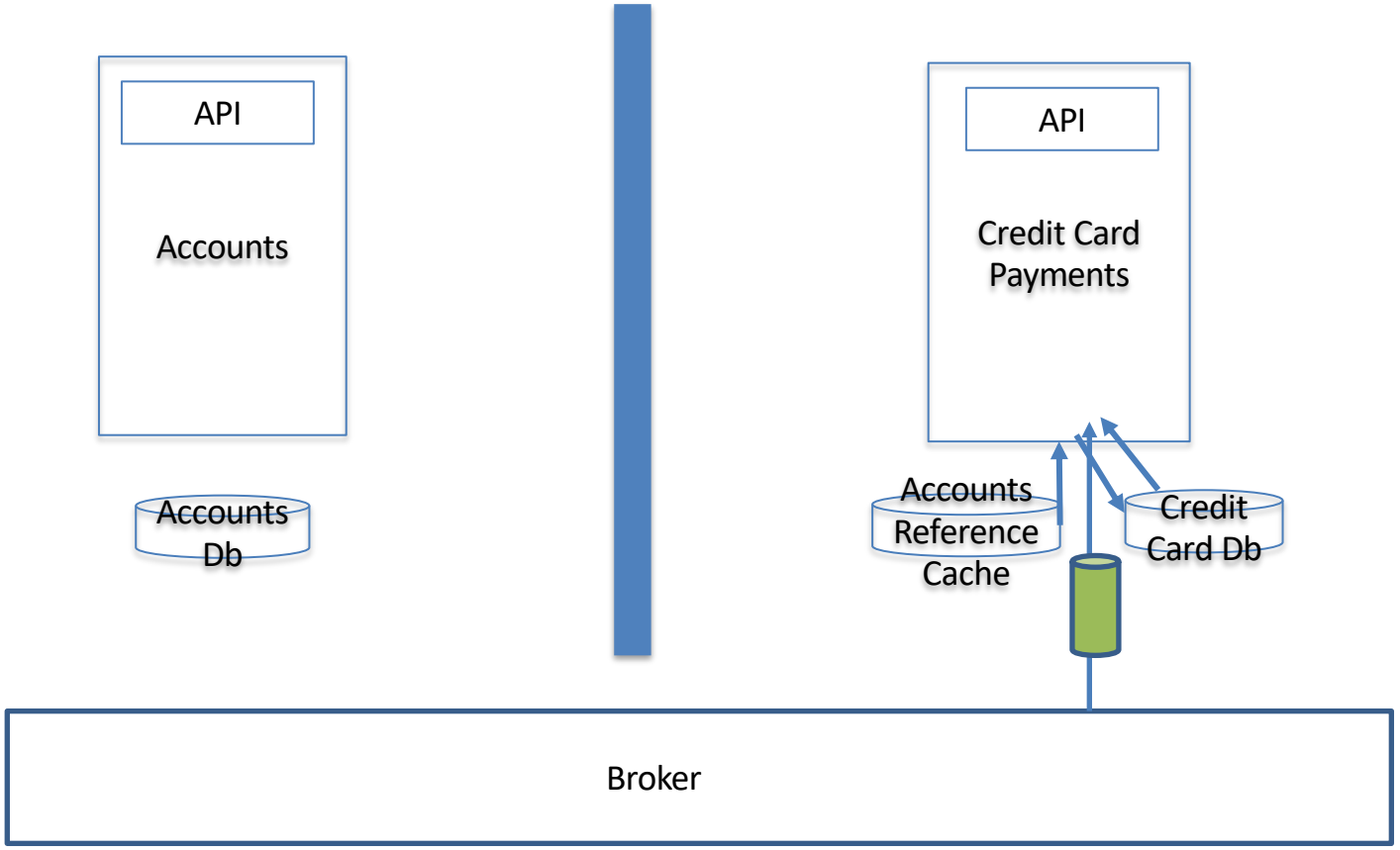
The Web scales by caching

Query By Event Carried State Transfer

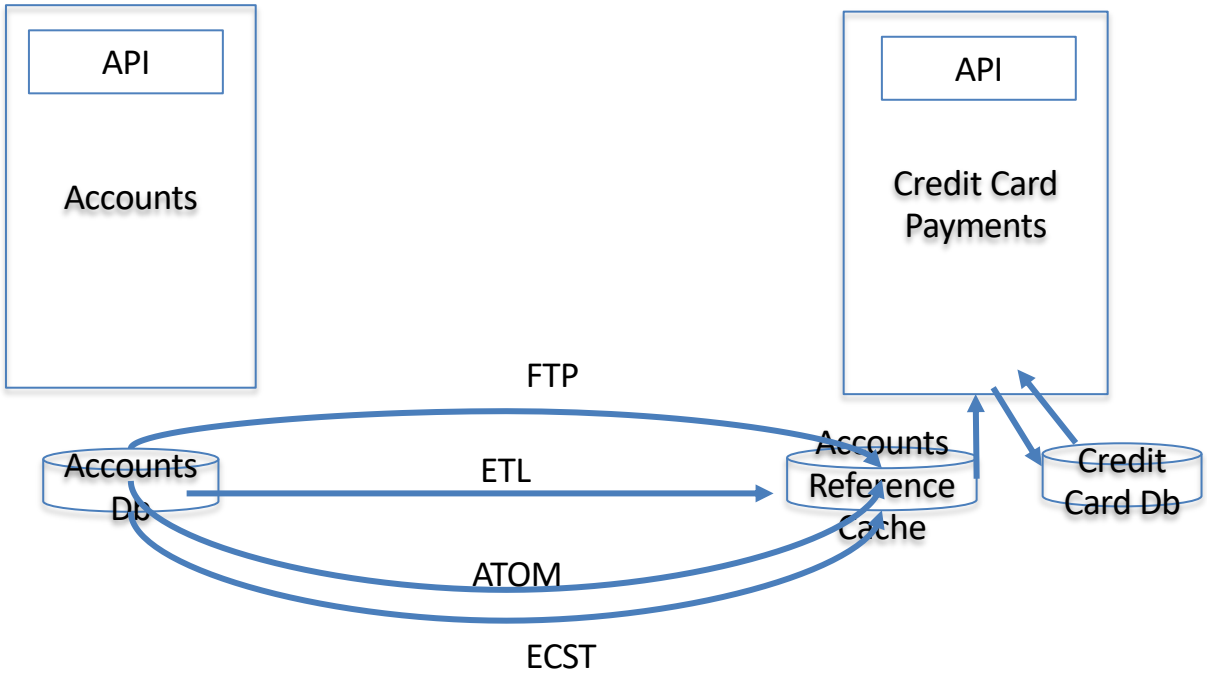
Martin Fowler



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

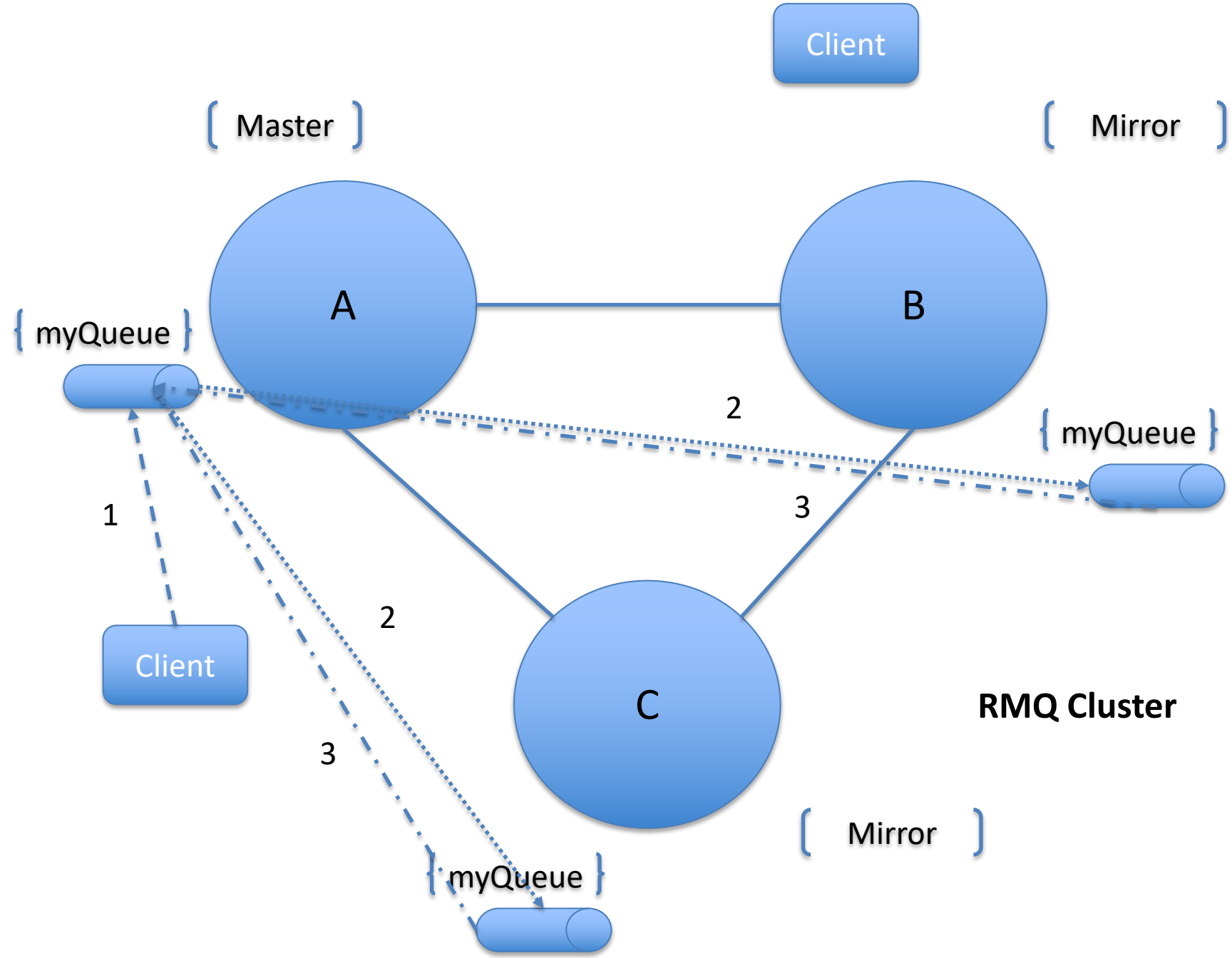
Replication of Data

Synchronization

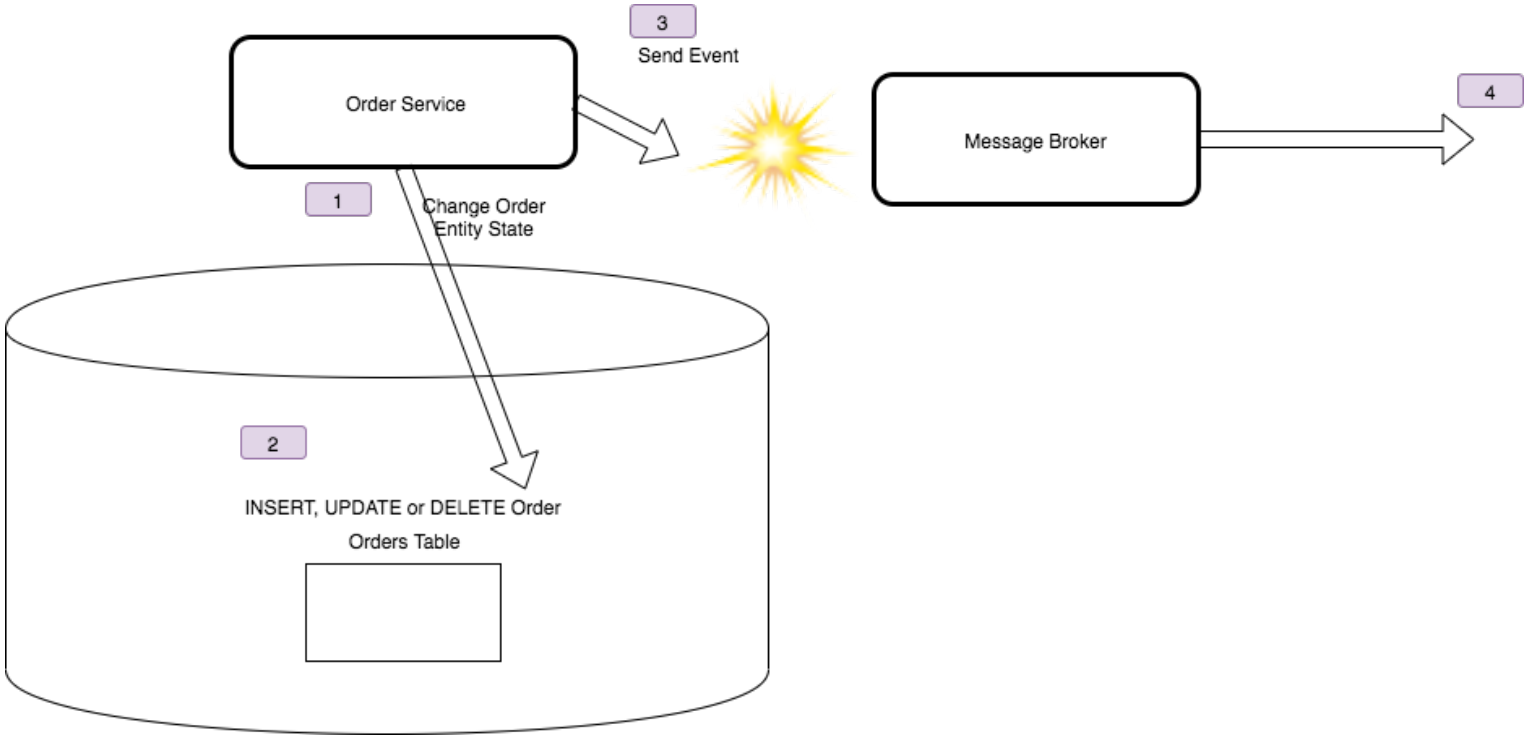
4. RELIABILITY

At Least Once

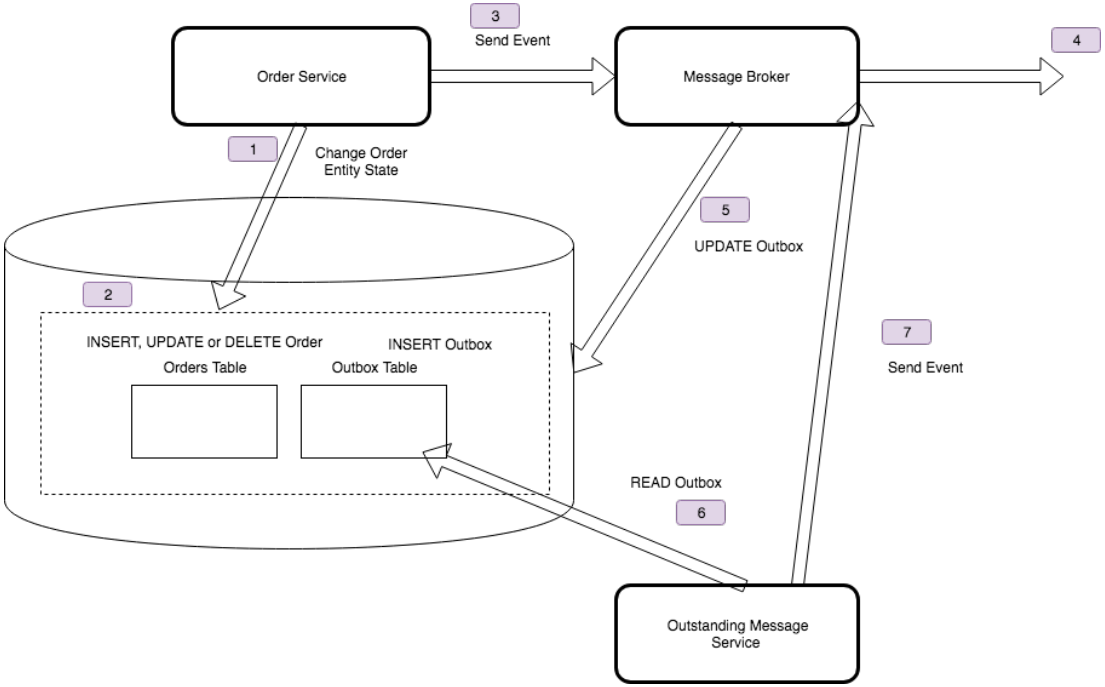
4.1 GUARANTEED DELIVERY


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

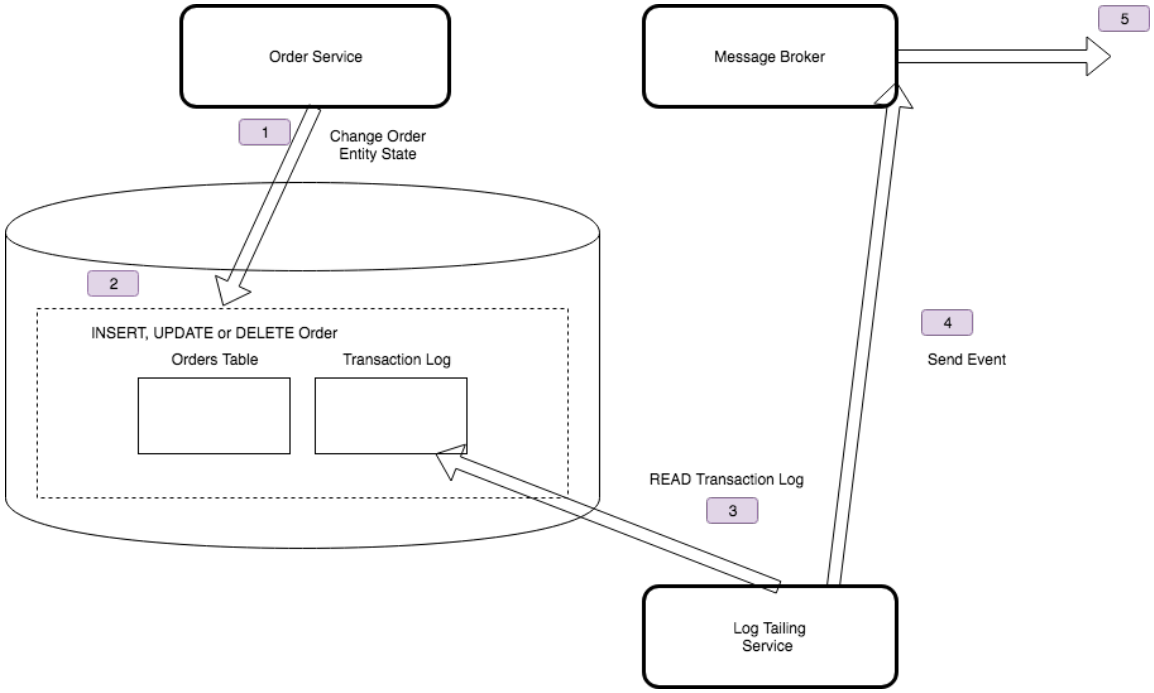
Correctness Problem



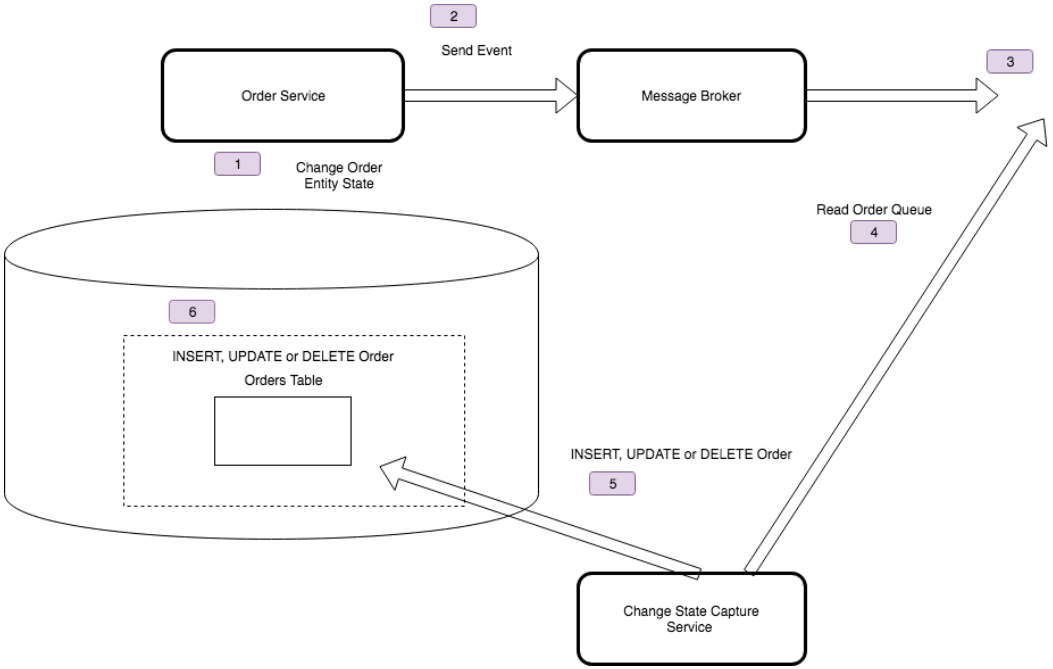
Outbox Pattern



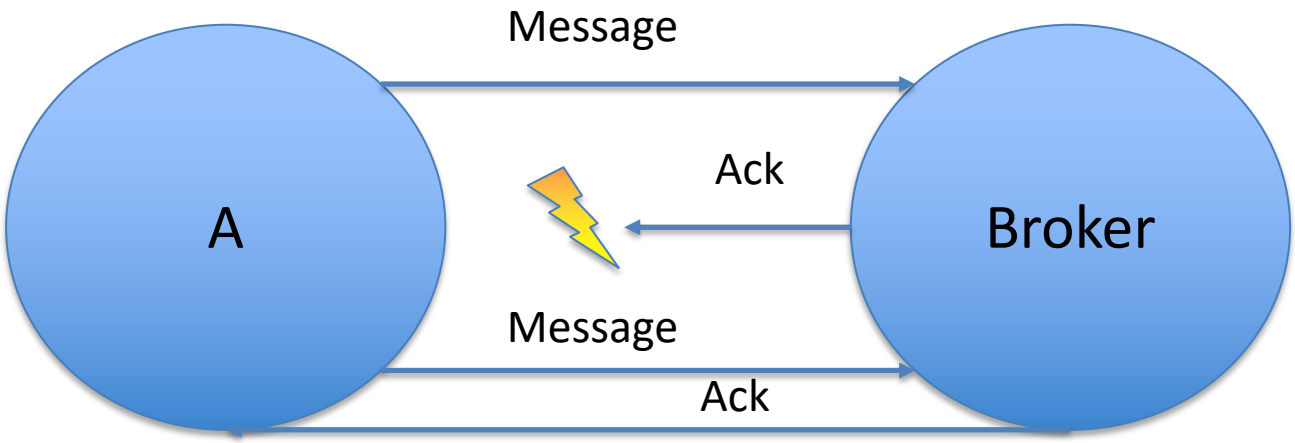
Log Tailing

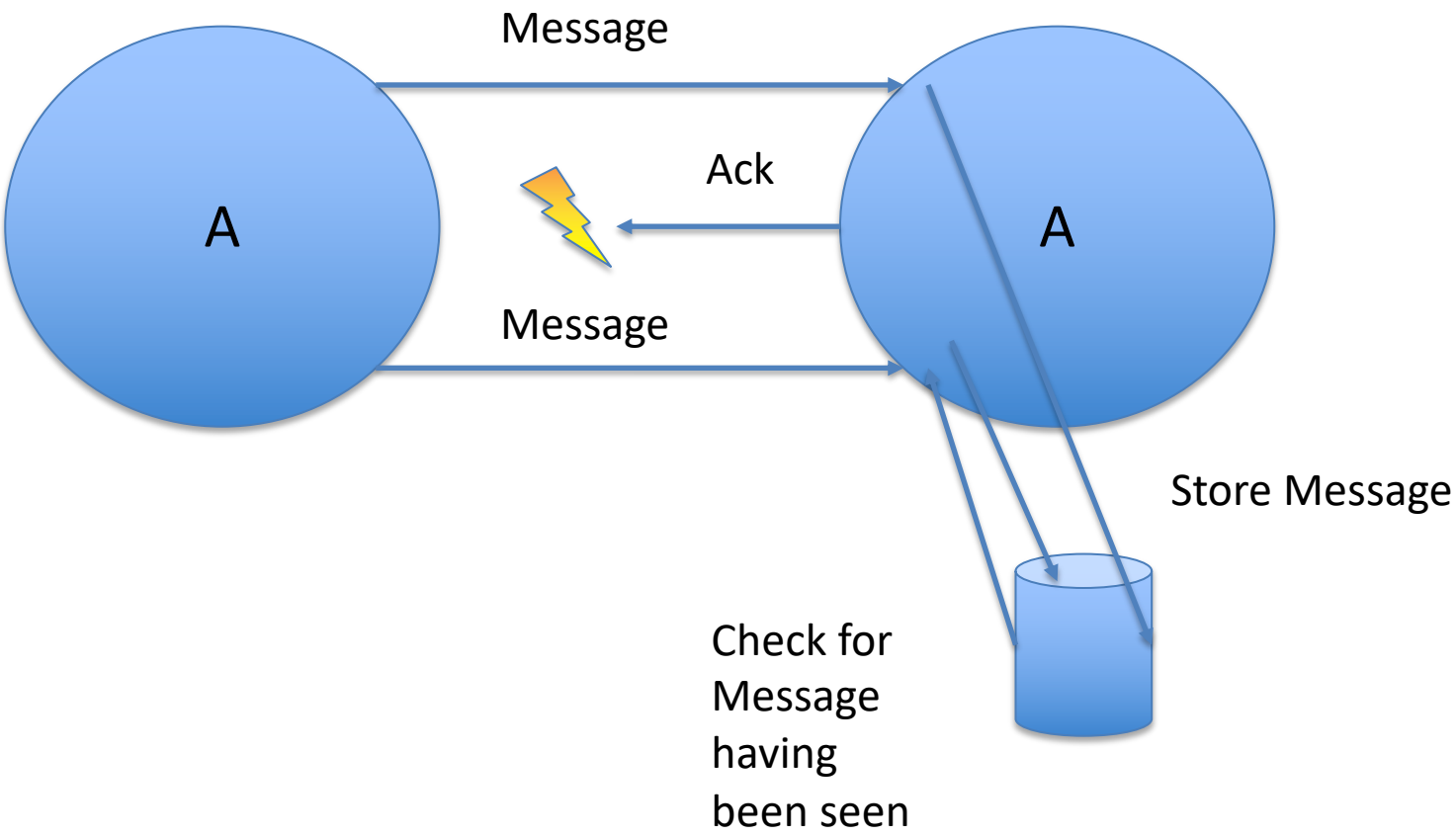

Order Service
Message Broker
5
1
Change Order
Entity State
2
INSERT, UPDATE or DELETE Order
Orders Table
Transaction Log
4
Send Event
READ Transaction Log
3
Log Tailing
Service

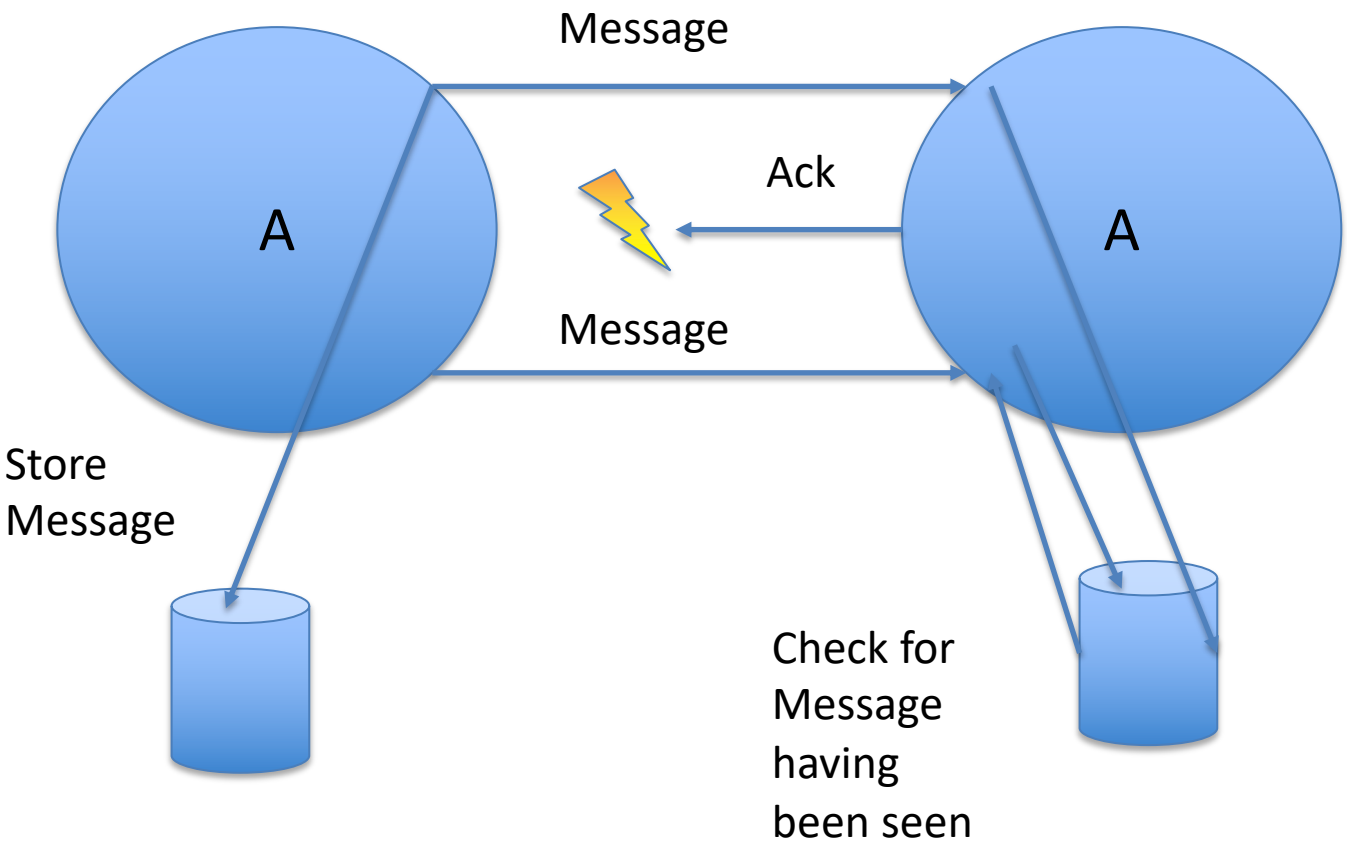
State Change Capture



4.2 AT LEAST ONCE, EXACTLY ONCE

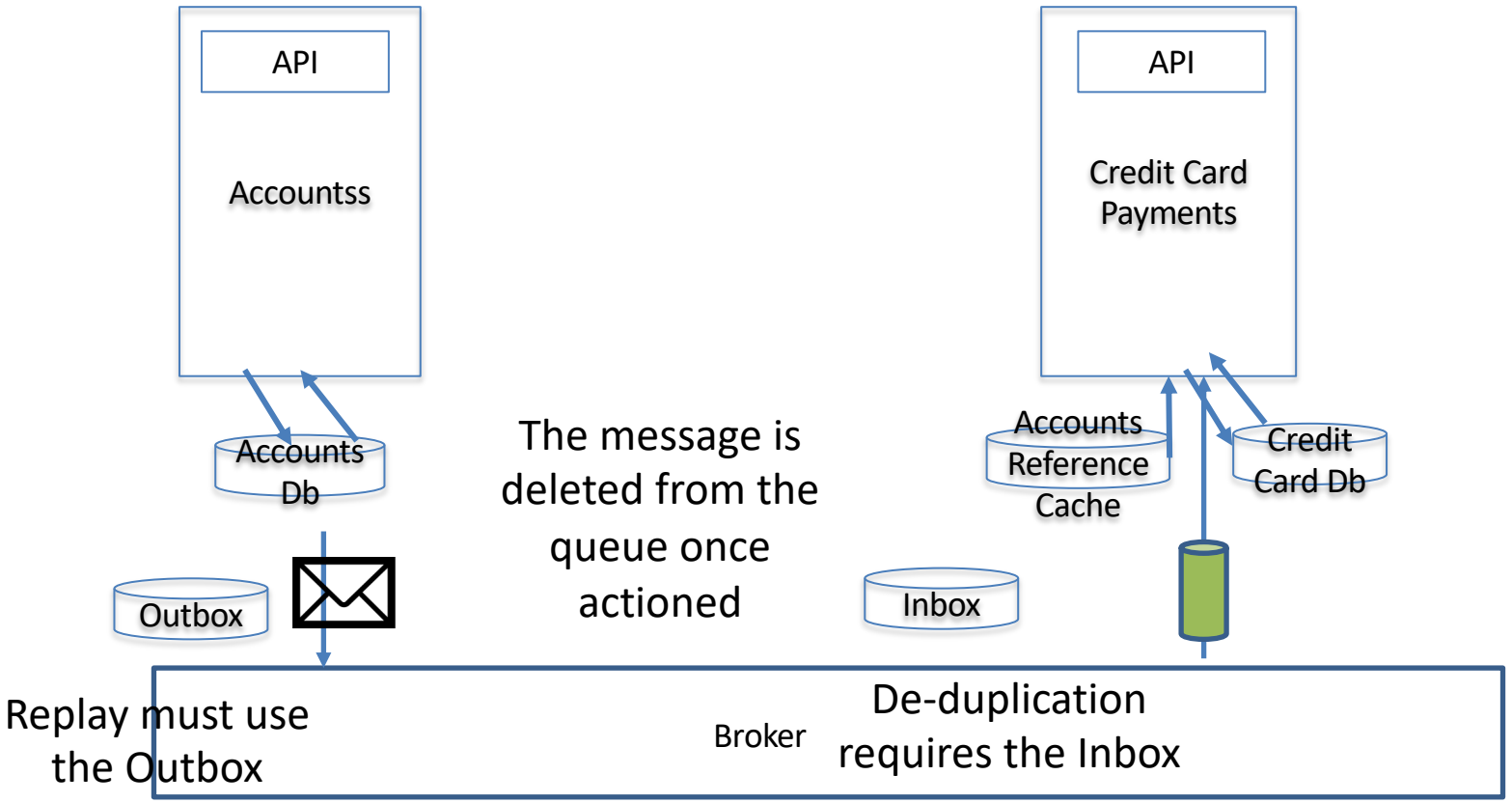

Message
A
Ack
Broker
Message
Ack


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

Shared Queue



Event Types

Discrete

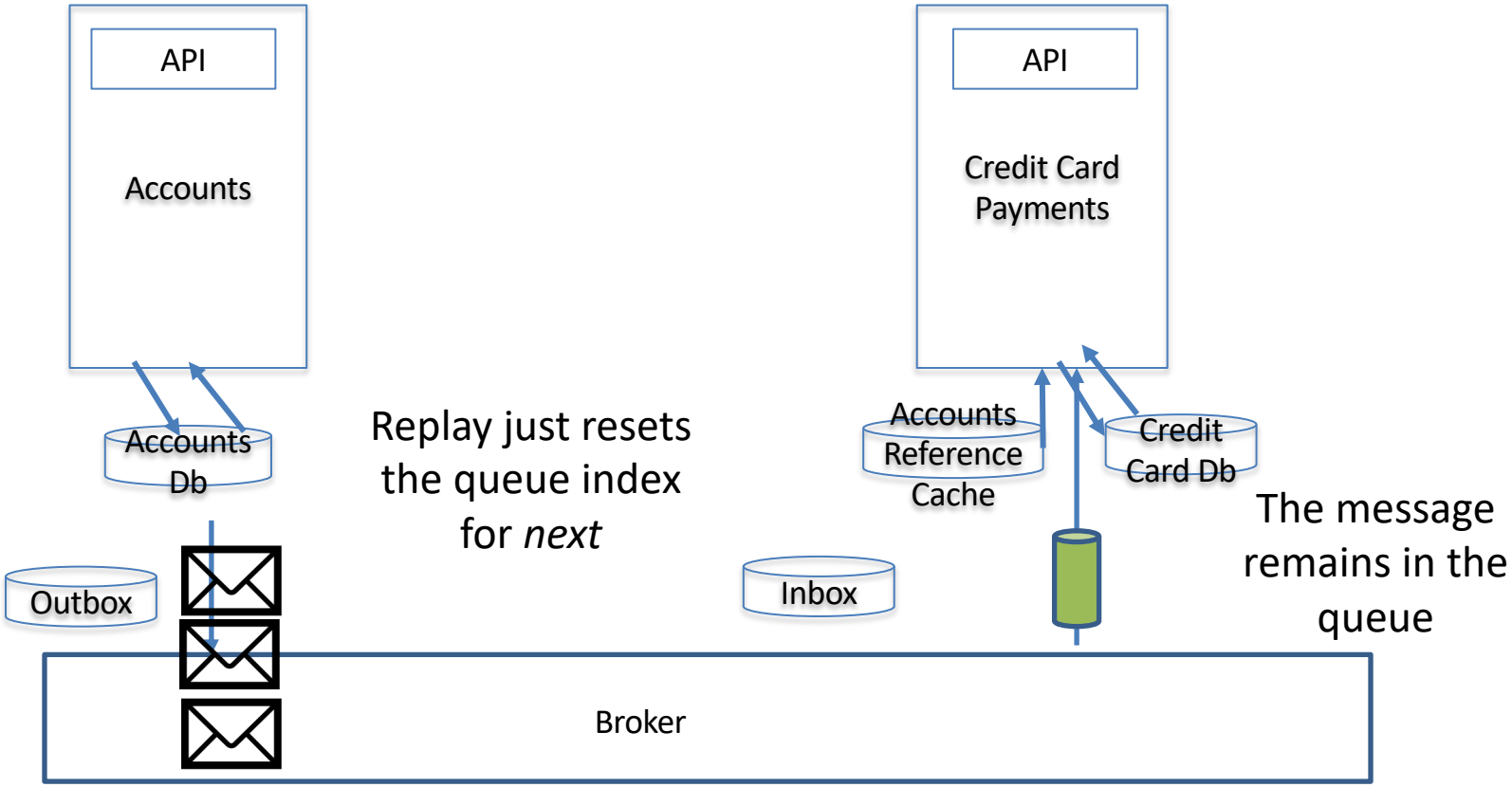
- Independent
- Report State Change
- Actionable

Series

- Time Ordered
- Context Partitioned
- Report Condition
- Analyzable

PUSH requires an open socket, so not used with Cloud Infrastructure brokers, who use PULL in this context but still have a stateless handler. Adds latency to notification depending on polling interval

Message Log



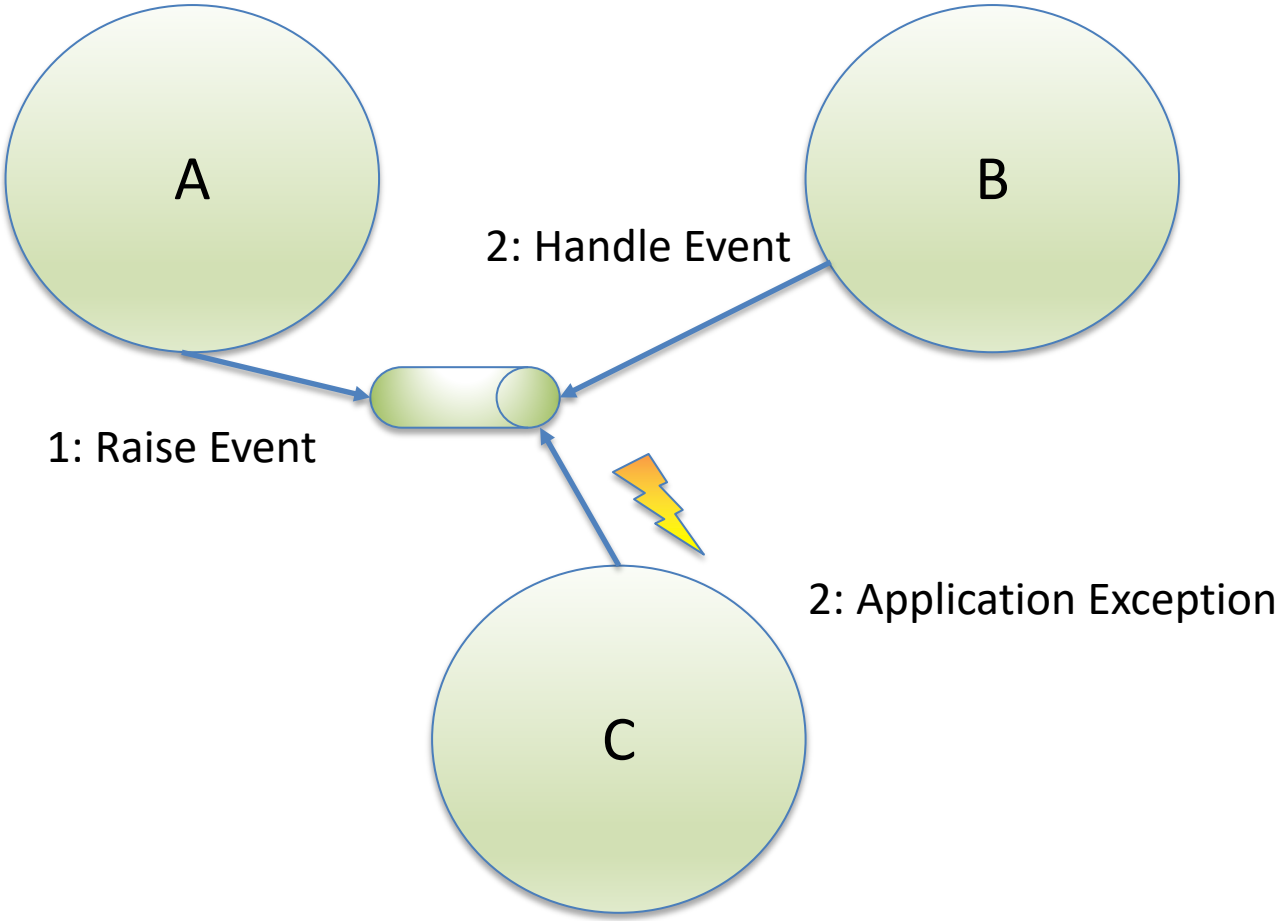
Outbox only needed if required for correctness

Inbox only needed if log does not support once-only

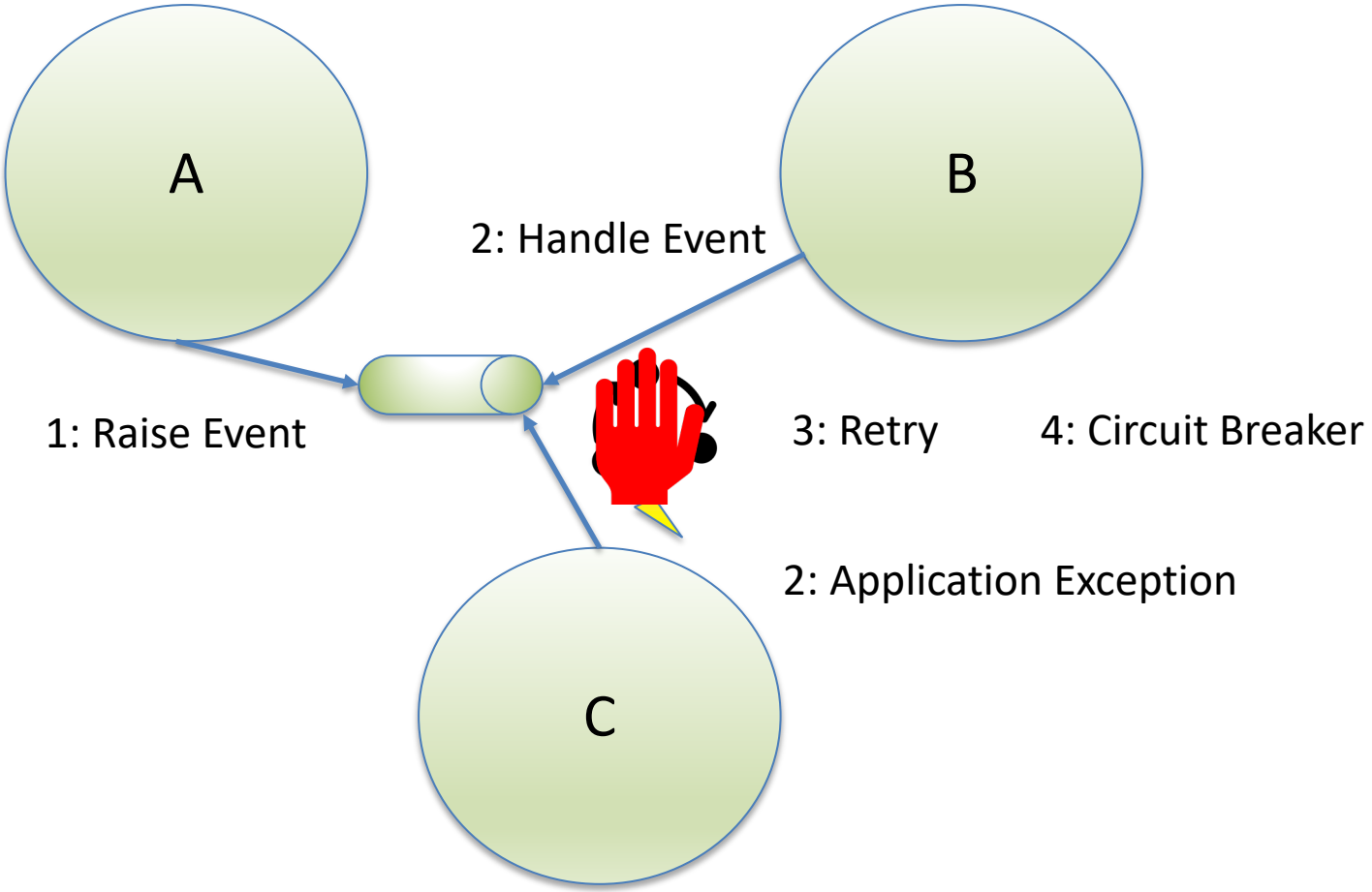
4.4 COMPENSATION


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

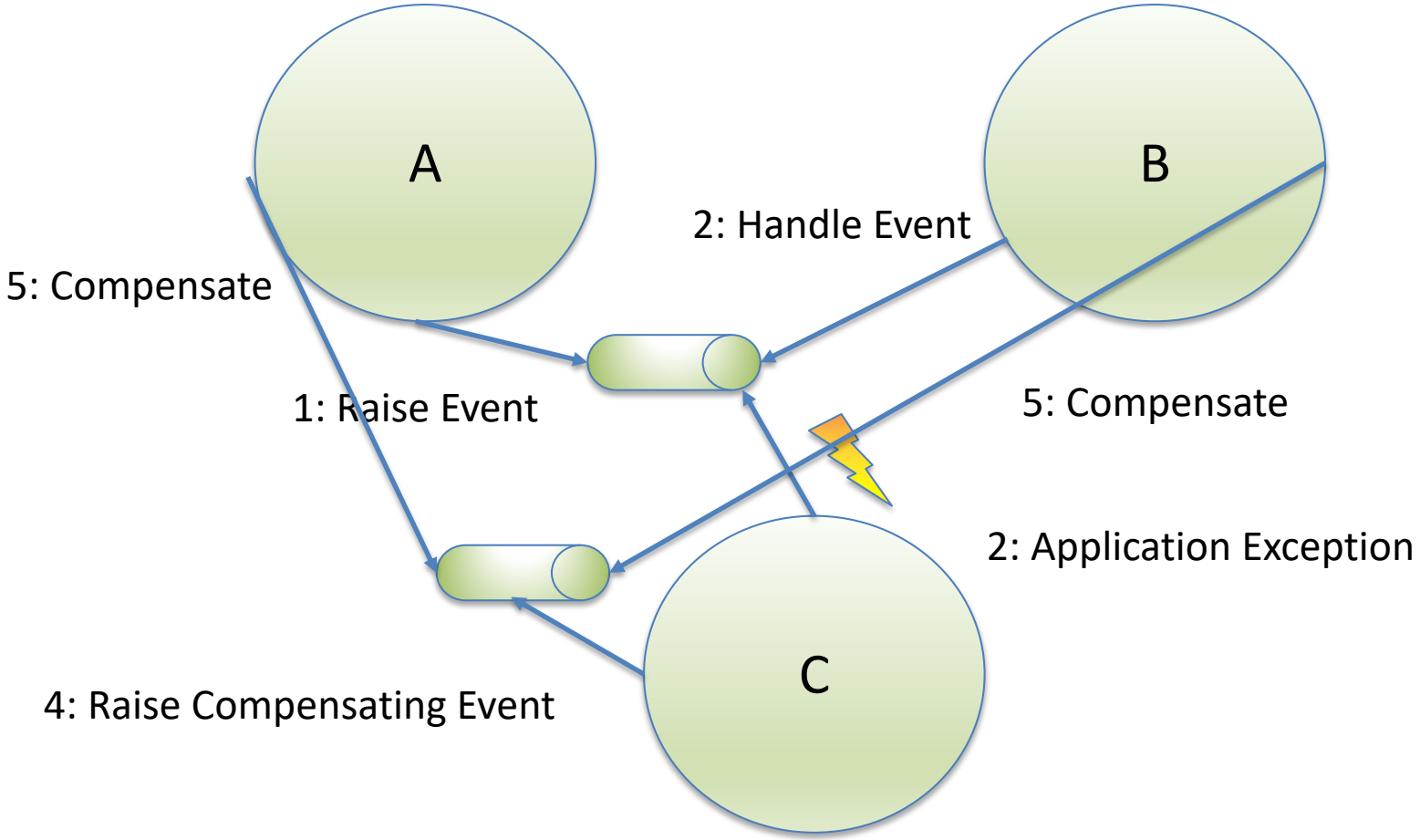
Compensation: Write Off



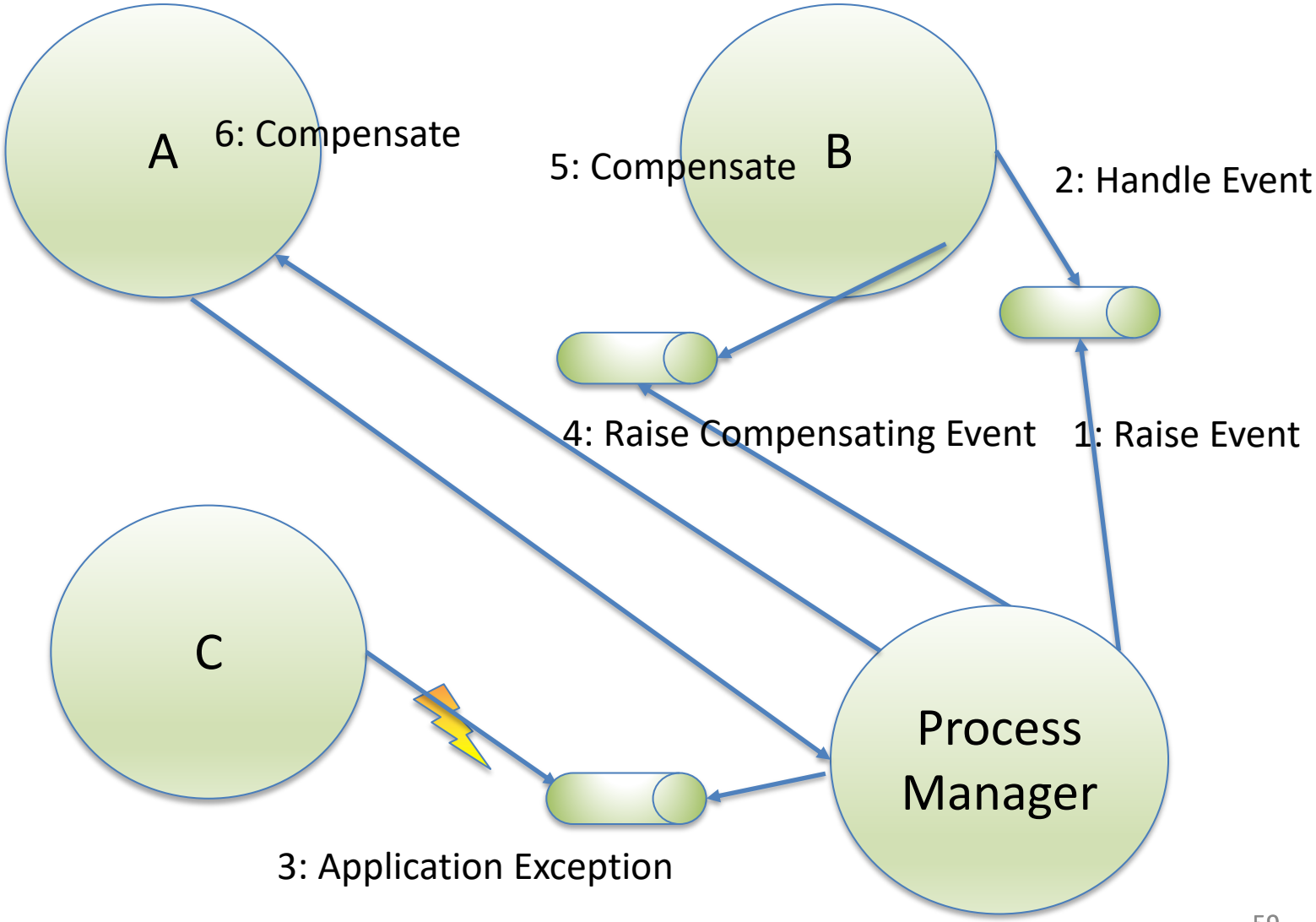
Compensation: Retry



Compensation: Compensating Event



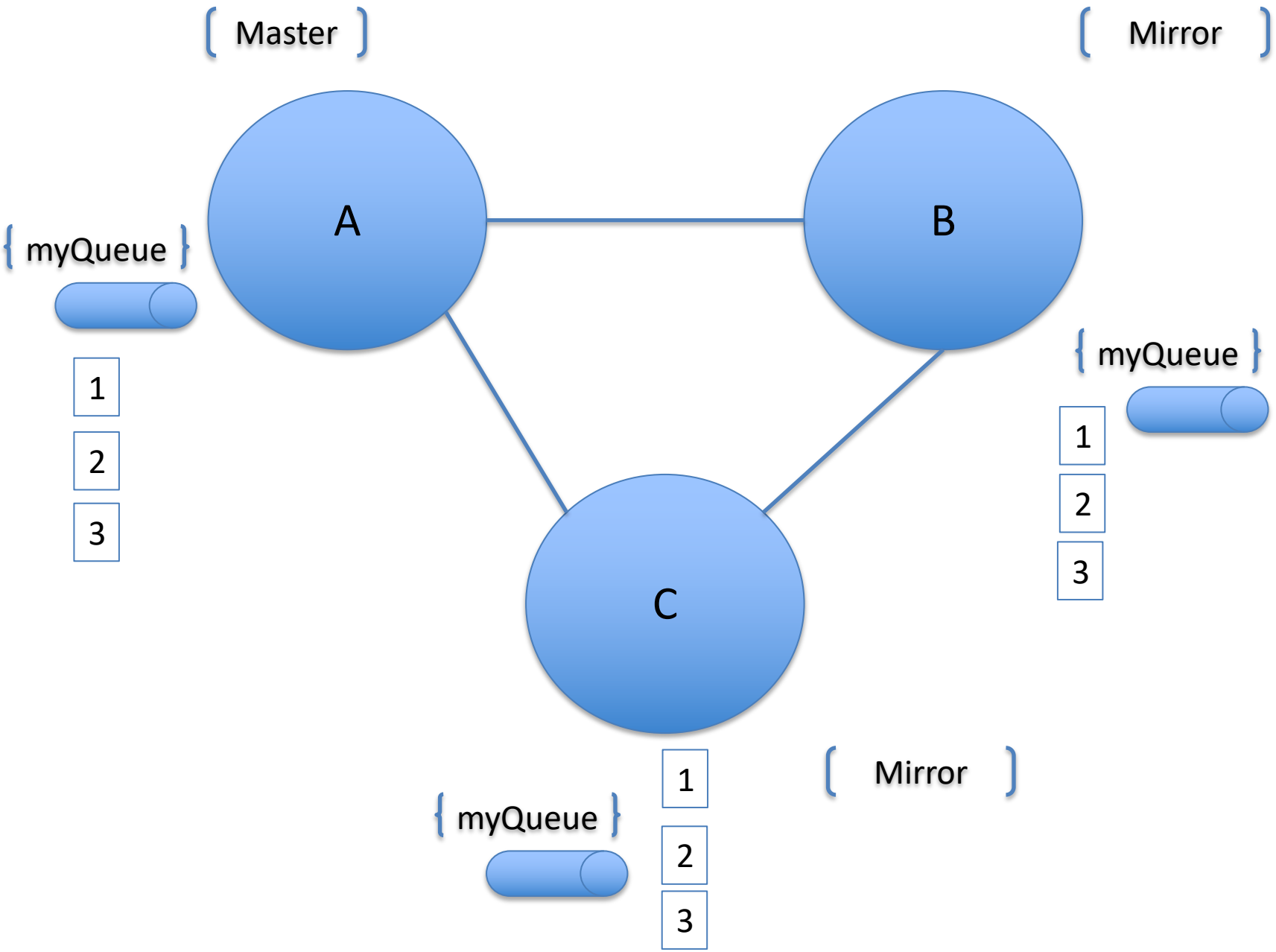
Compensation: Process Manager

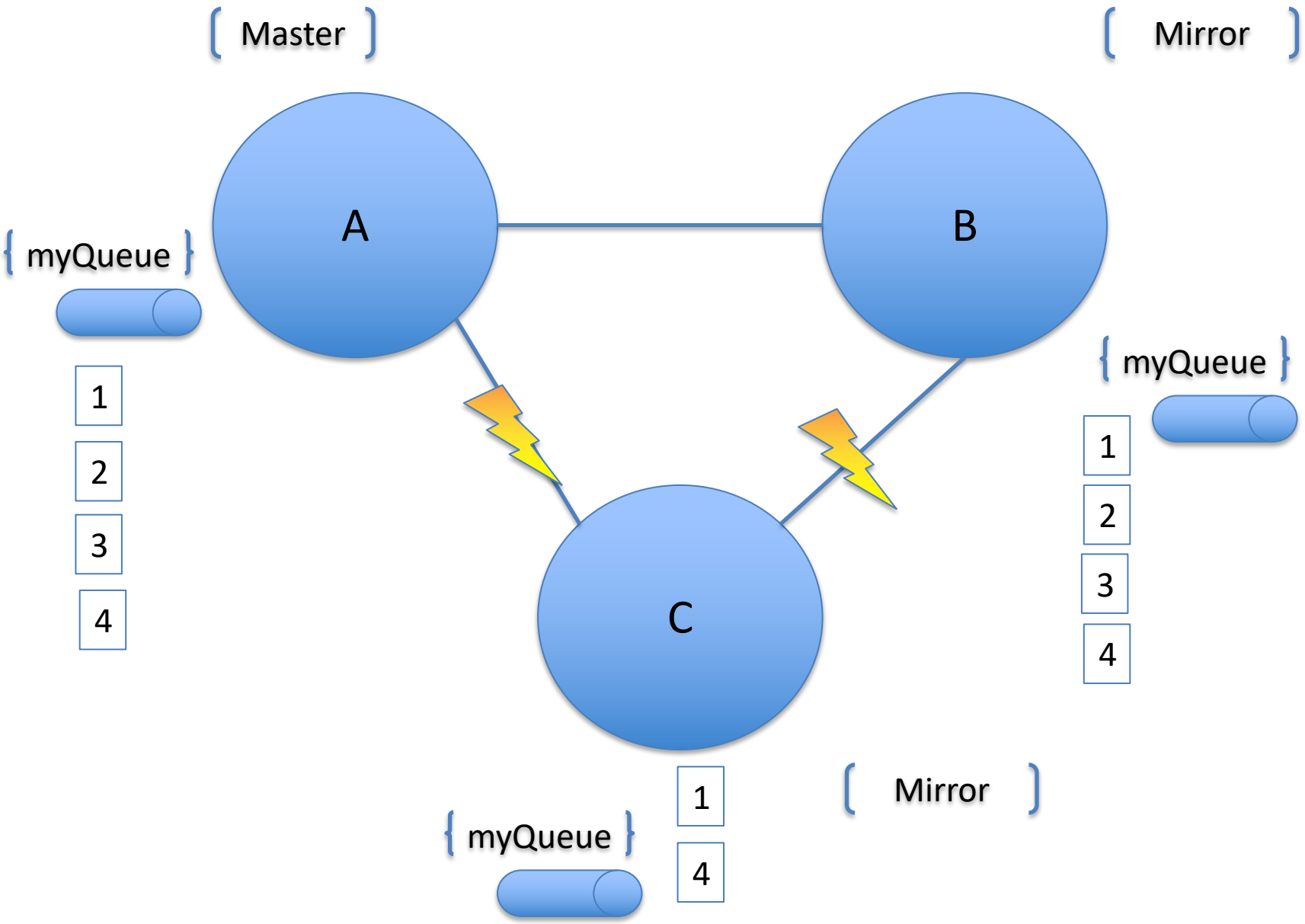


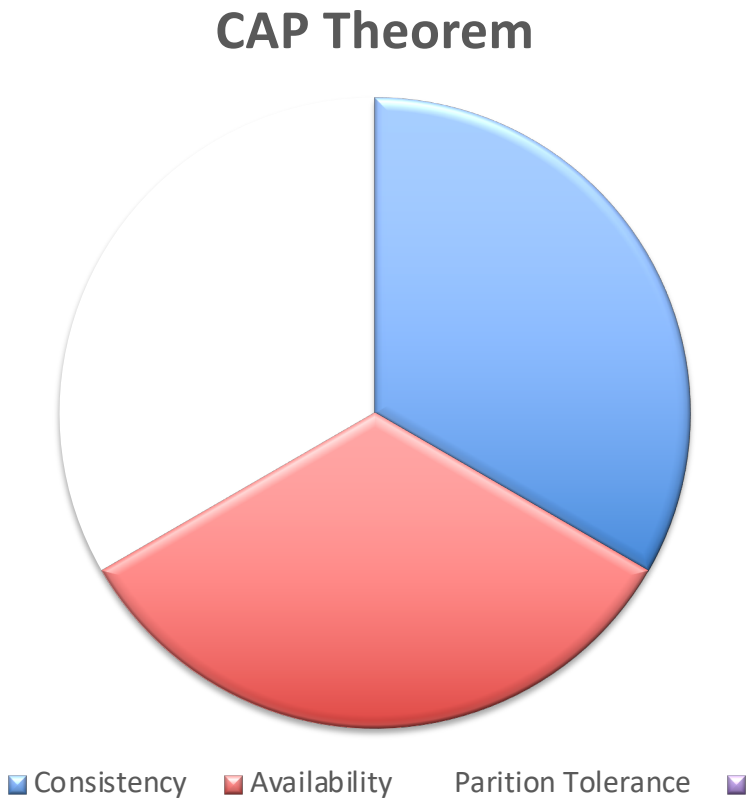
4.5 CAP THEOREM

CAP Theorem

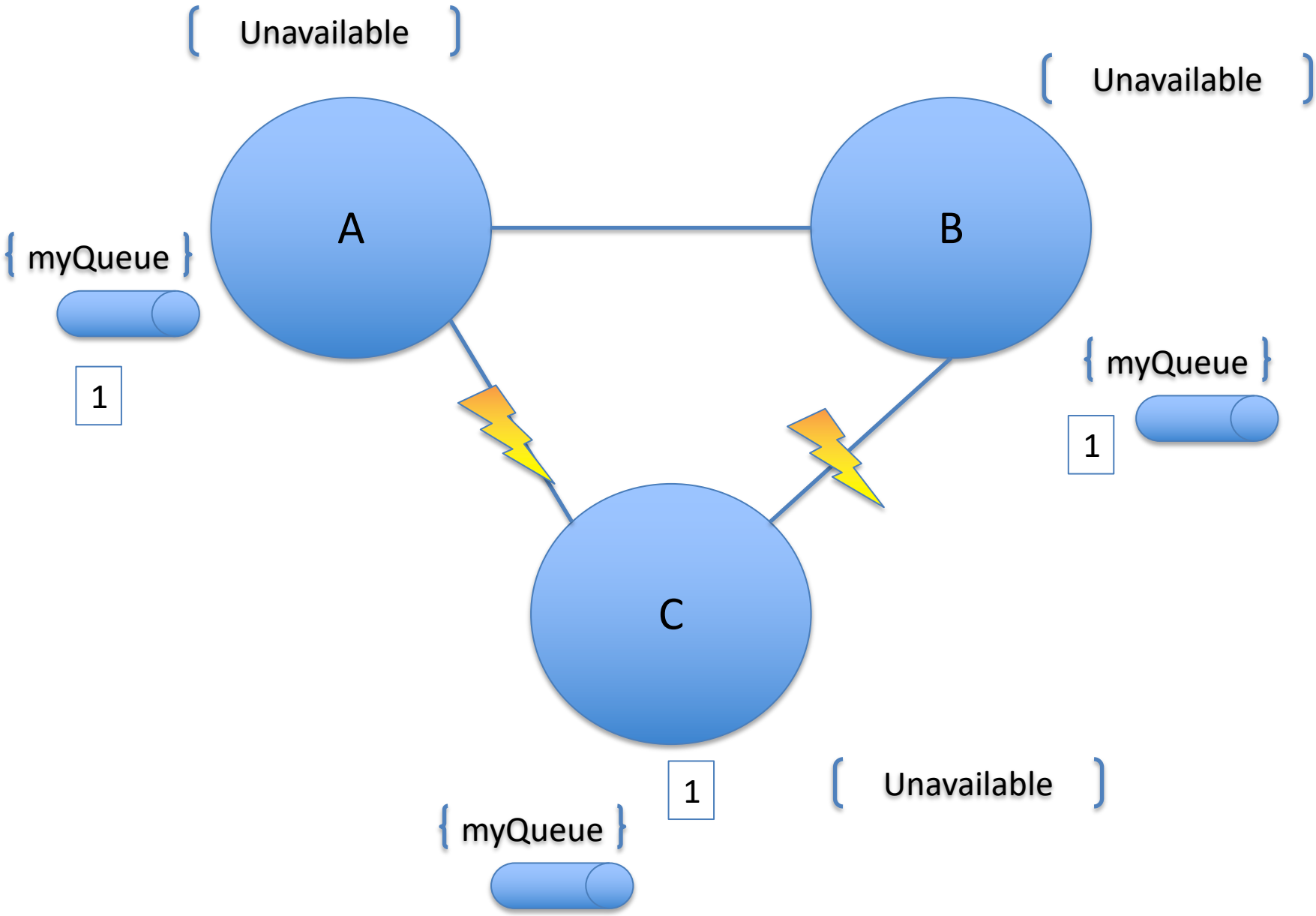
You can have at most two of these properties for any shared data system

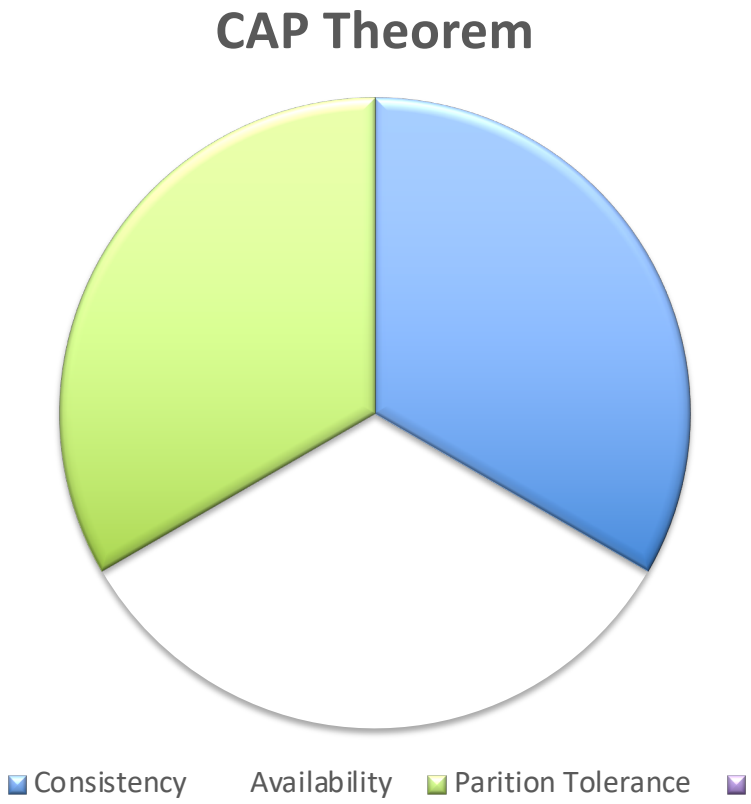

Master
Mirror
A
B
myQueue
1
2
3
myQueue
1
2
3
C
Mirror
myQueue
1
2
3


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

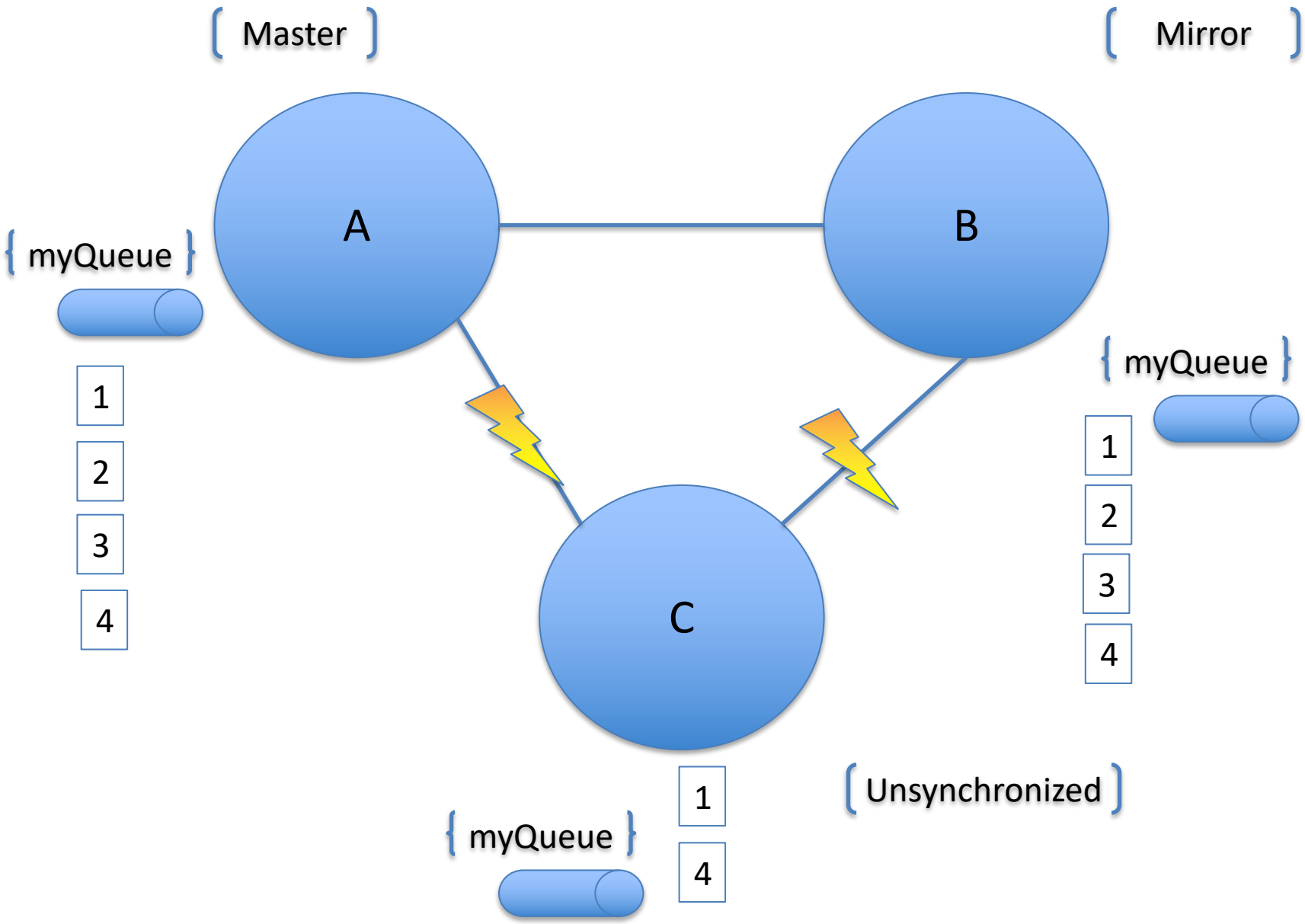


Forfeit Partitions

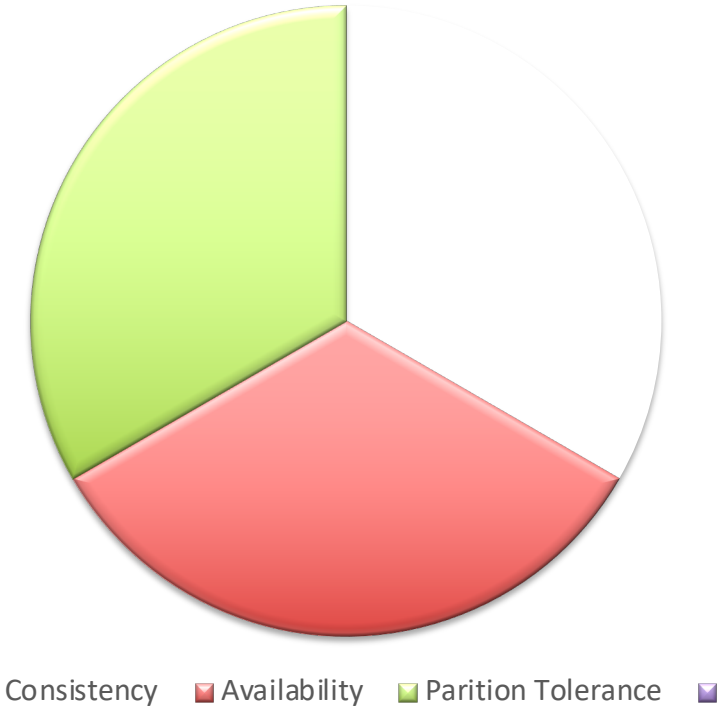

Unavailable
Unavailable
A
B
myQueue
1
myQueue
1
C
1
Unavailable
myQueue



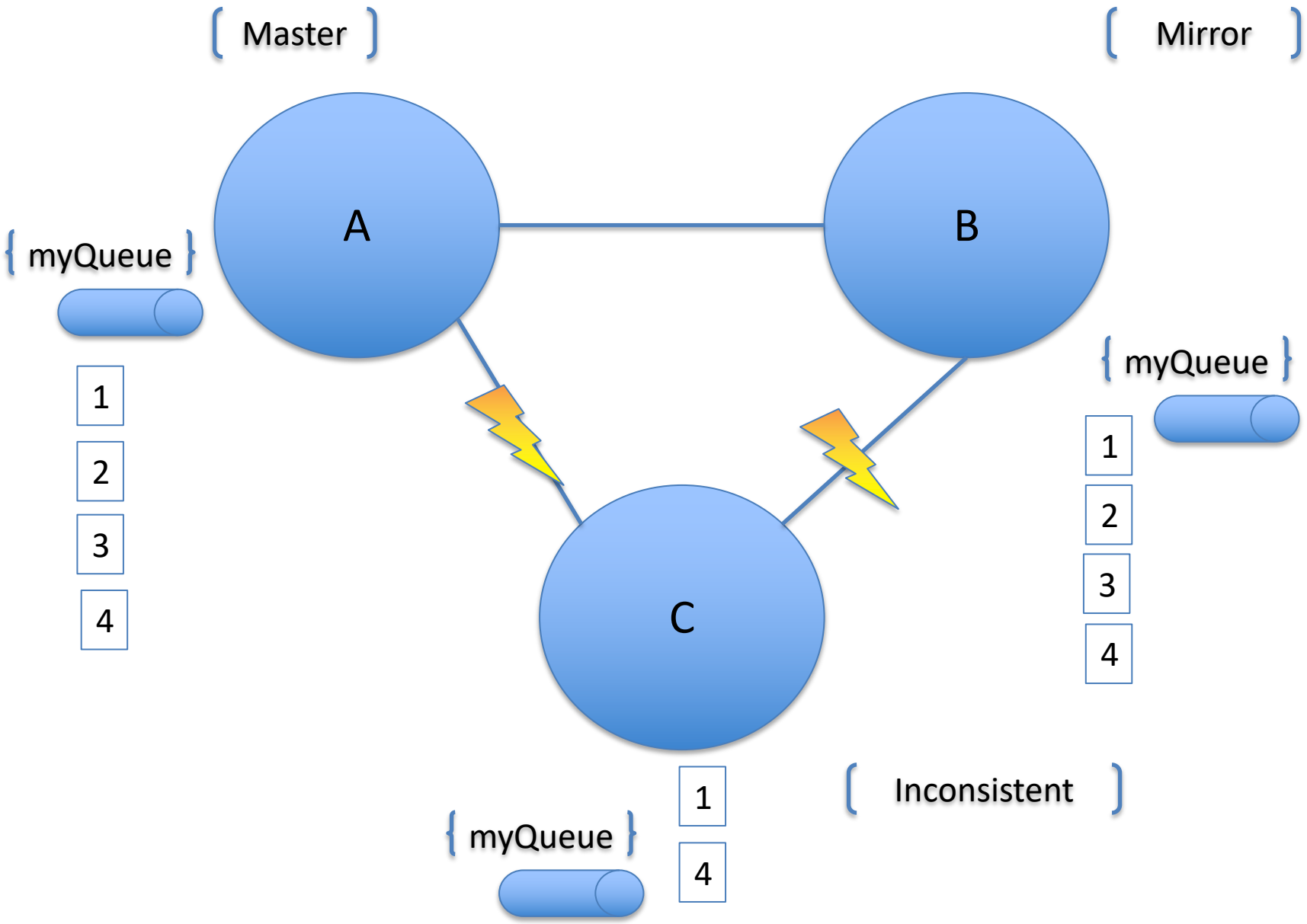
Forfeit Availability

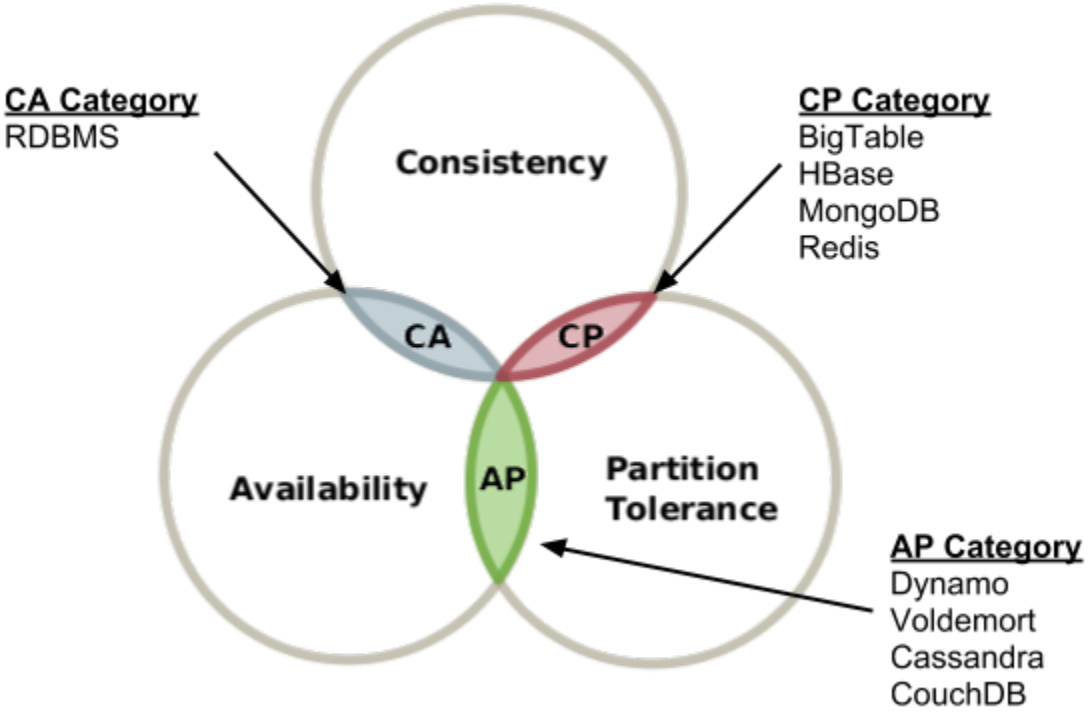

Master
Mirror
A
B
C
myQueue
1
2
3
4
myQueue
1
2
3
4
myQueue
1
4
Unsynchronized

CAP Theorem

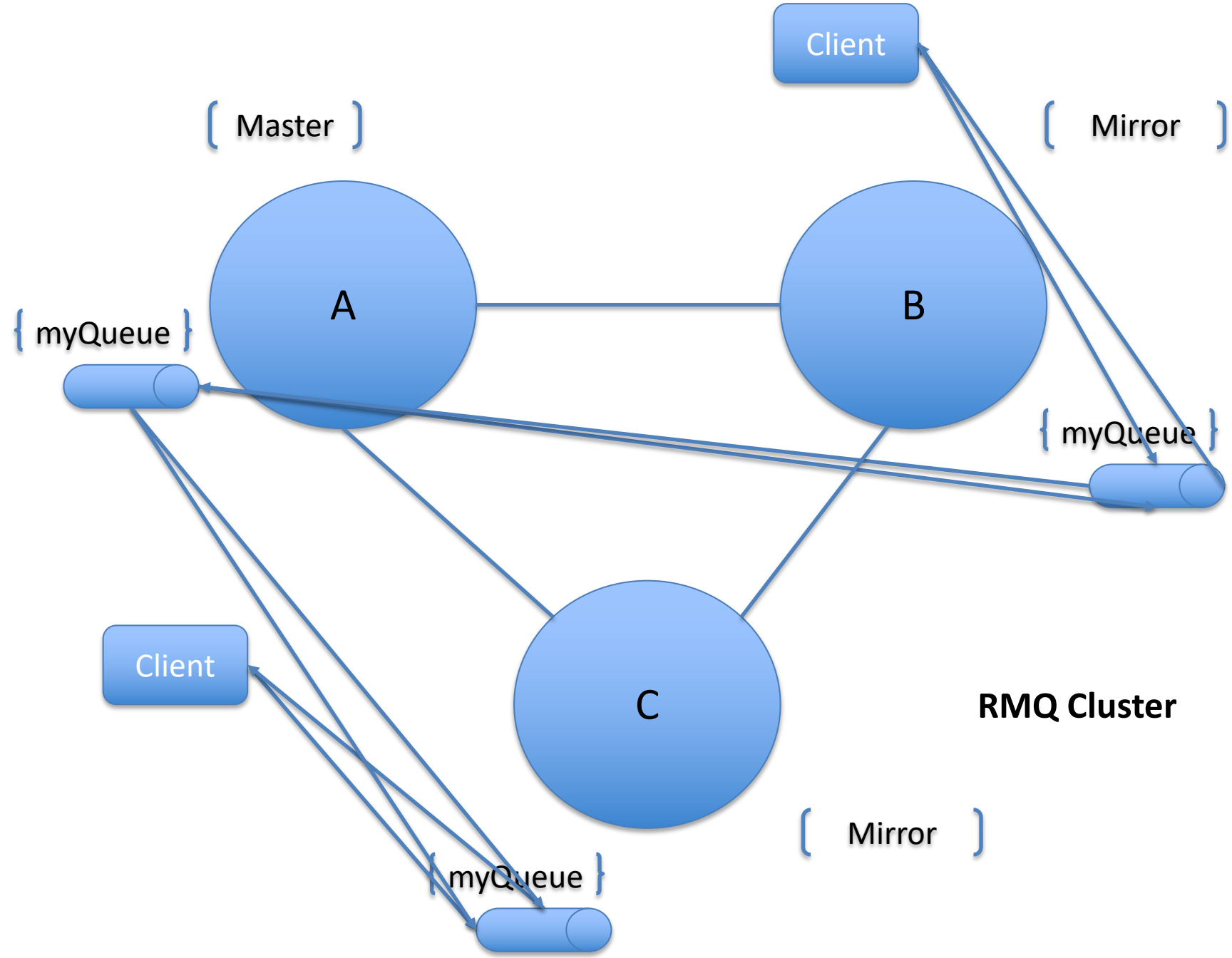


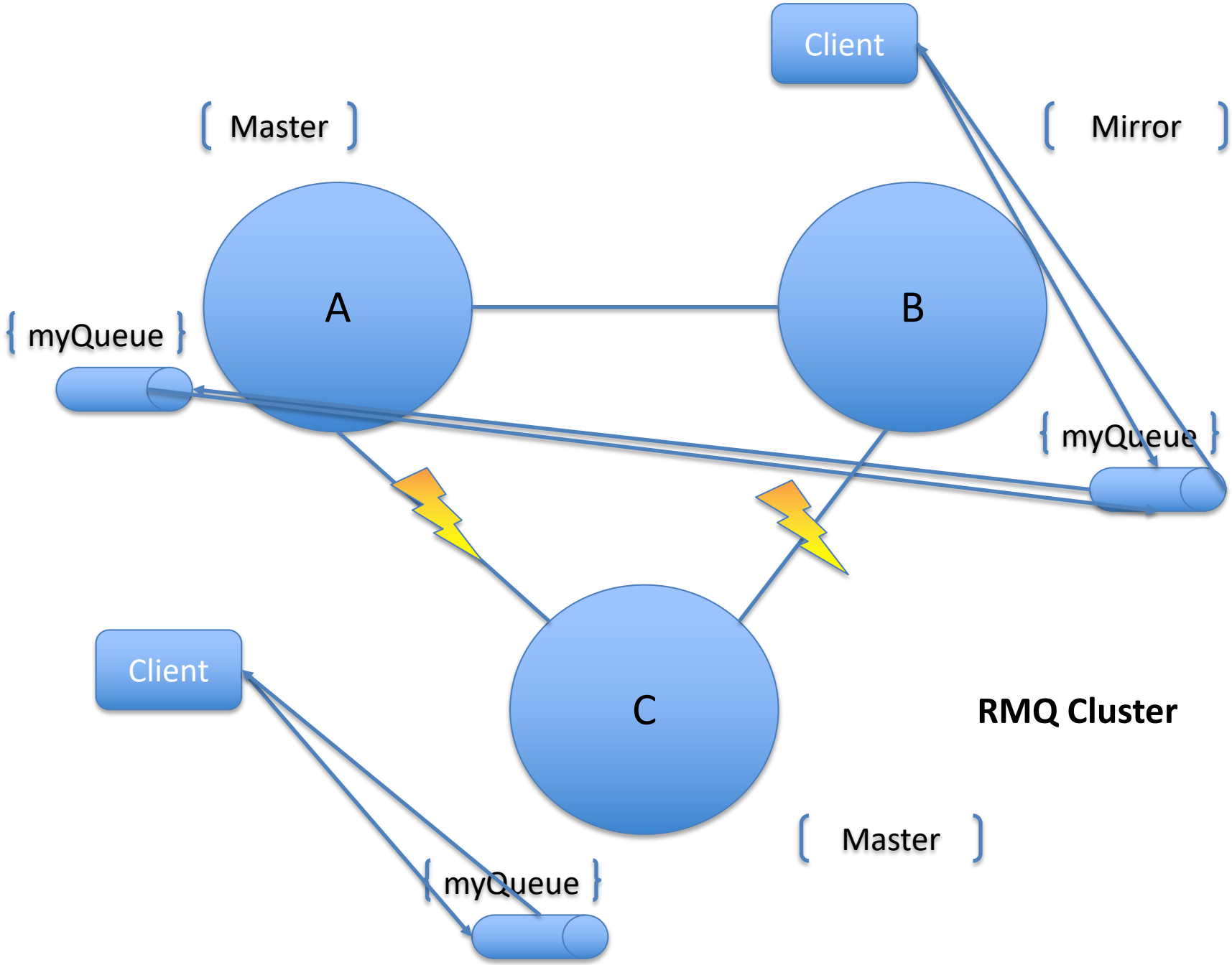
Forfeit Consistency

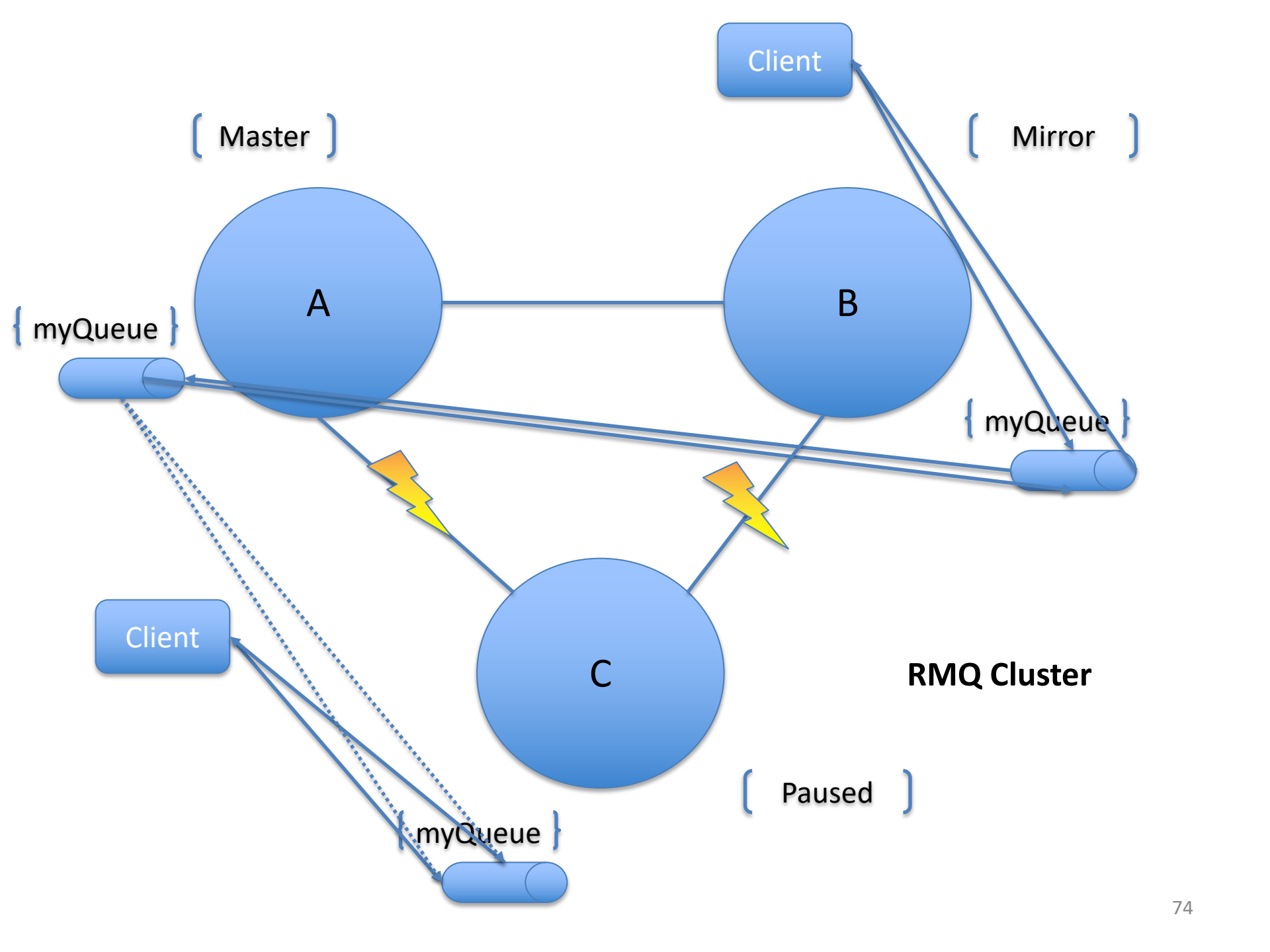

Master
Mirror
A
B
C
myQueue
1
2
3
4
myQueue
1
2
3
4
myQueue
1
4
Inconsistent

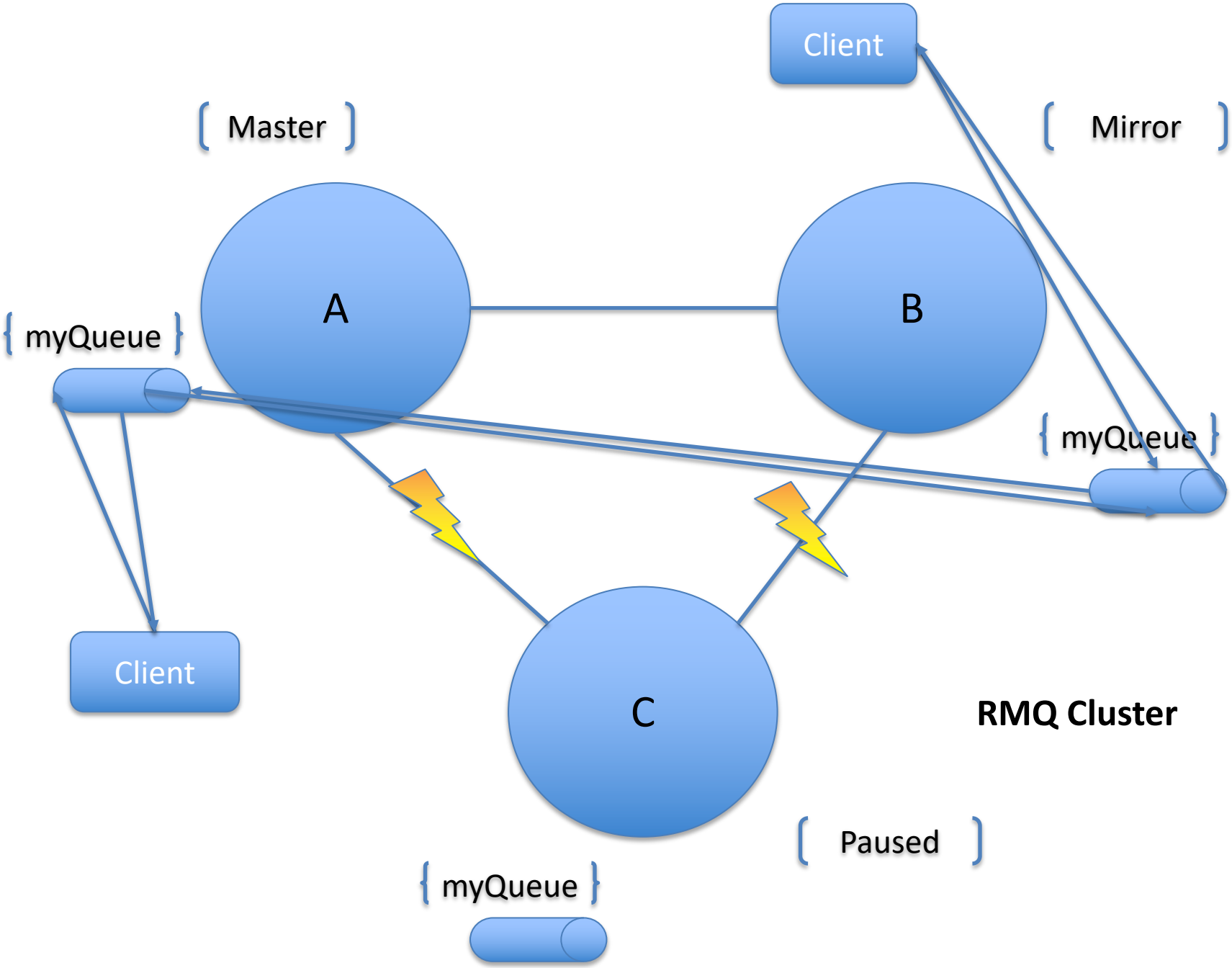

CA Category
RDBMS
Consistency
CP Category
BigTable
HBase
MongoDB
Redis
CA
CP
Availability
AP
Partition
Tolerance
AP Category
Dynamo
Voldemort
Cassandra
CouchDB

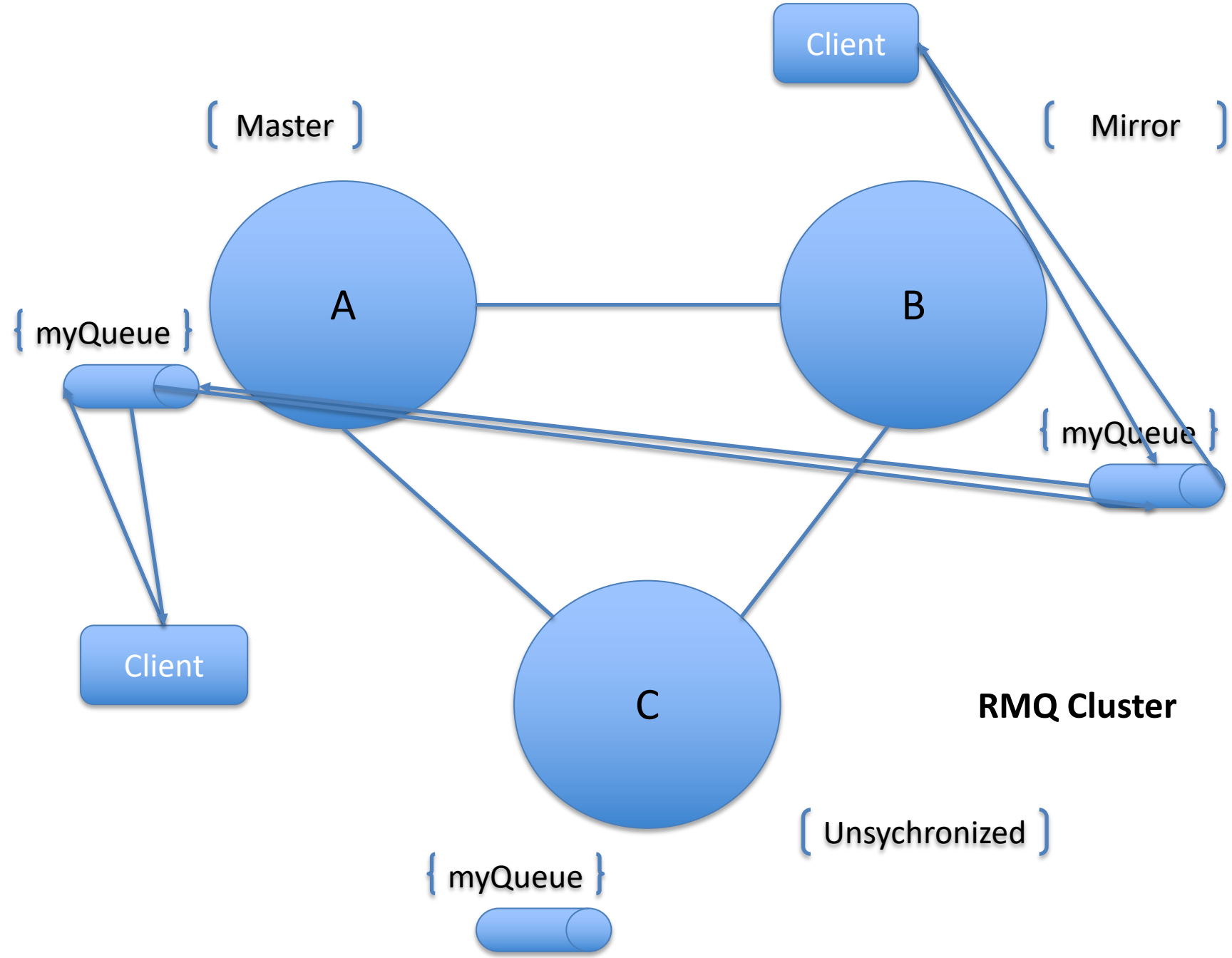
4.6 RABBITMQ AND CAP THEOREM

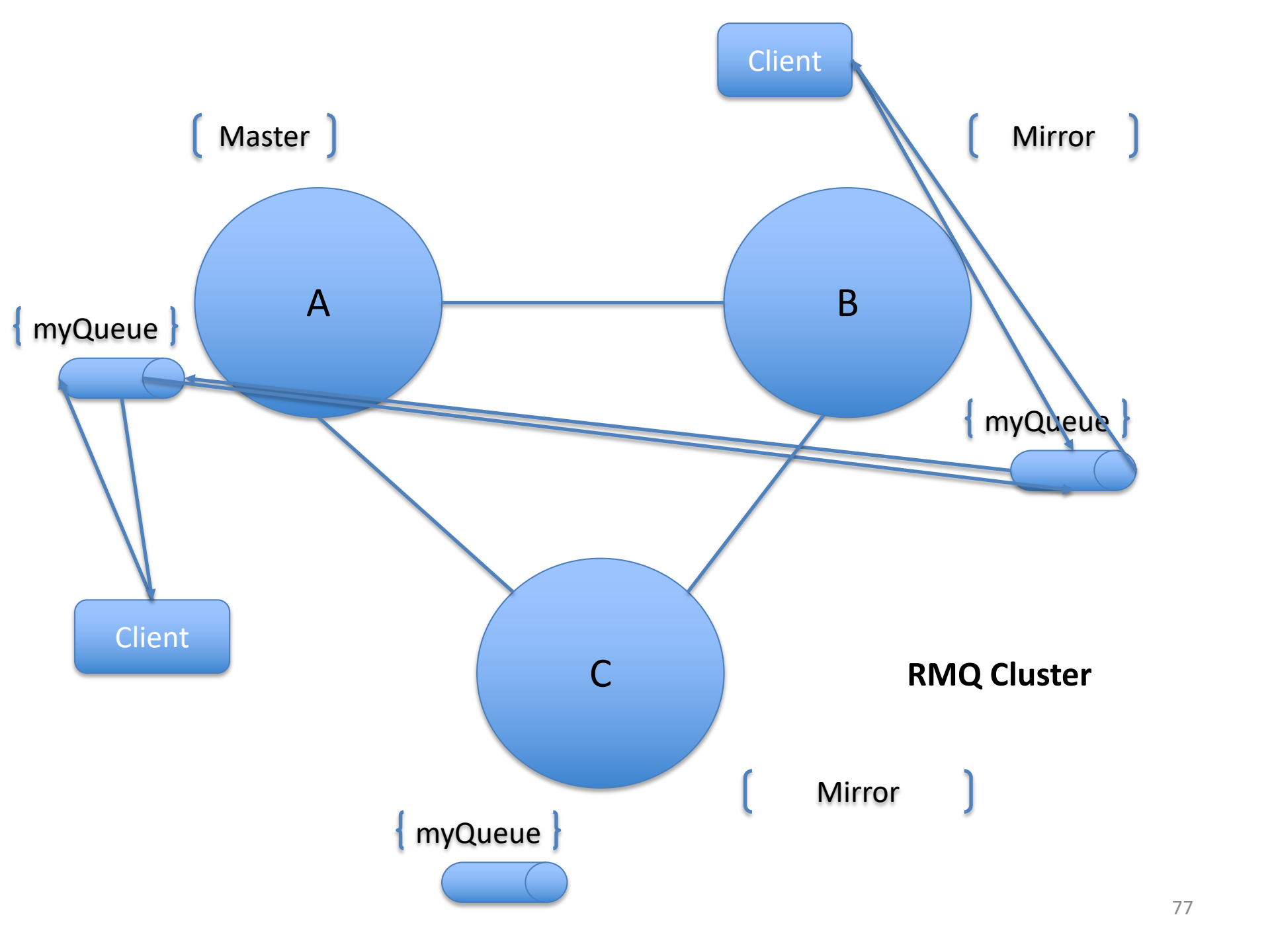

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

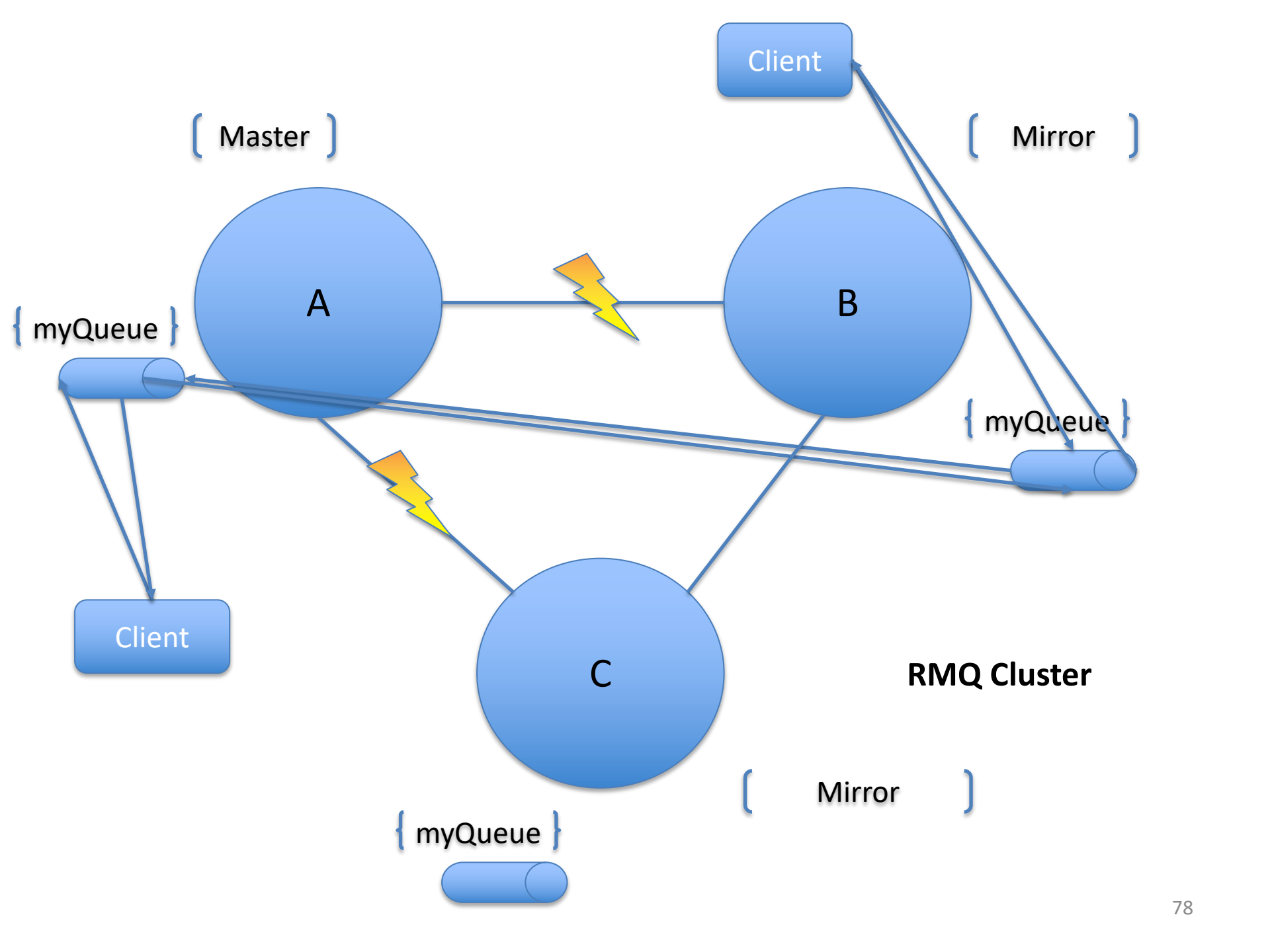

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Master
myQueue

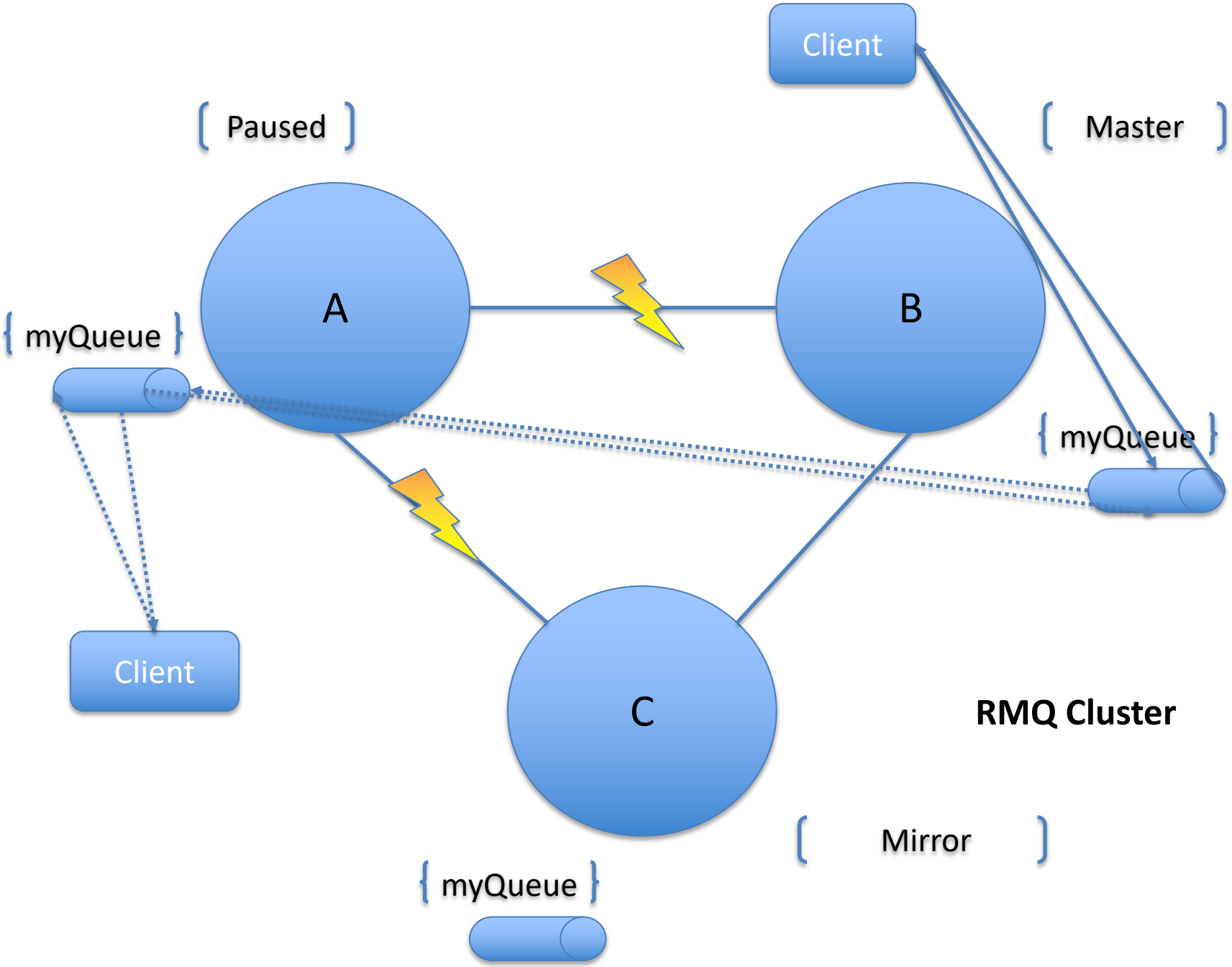

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Paused
myQueue

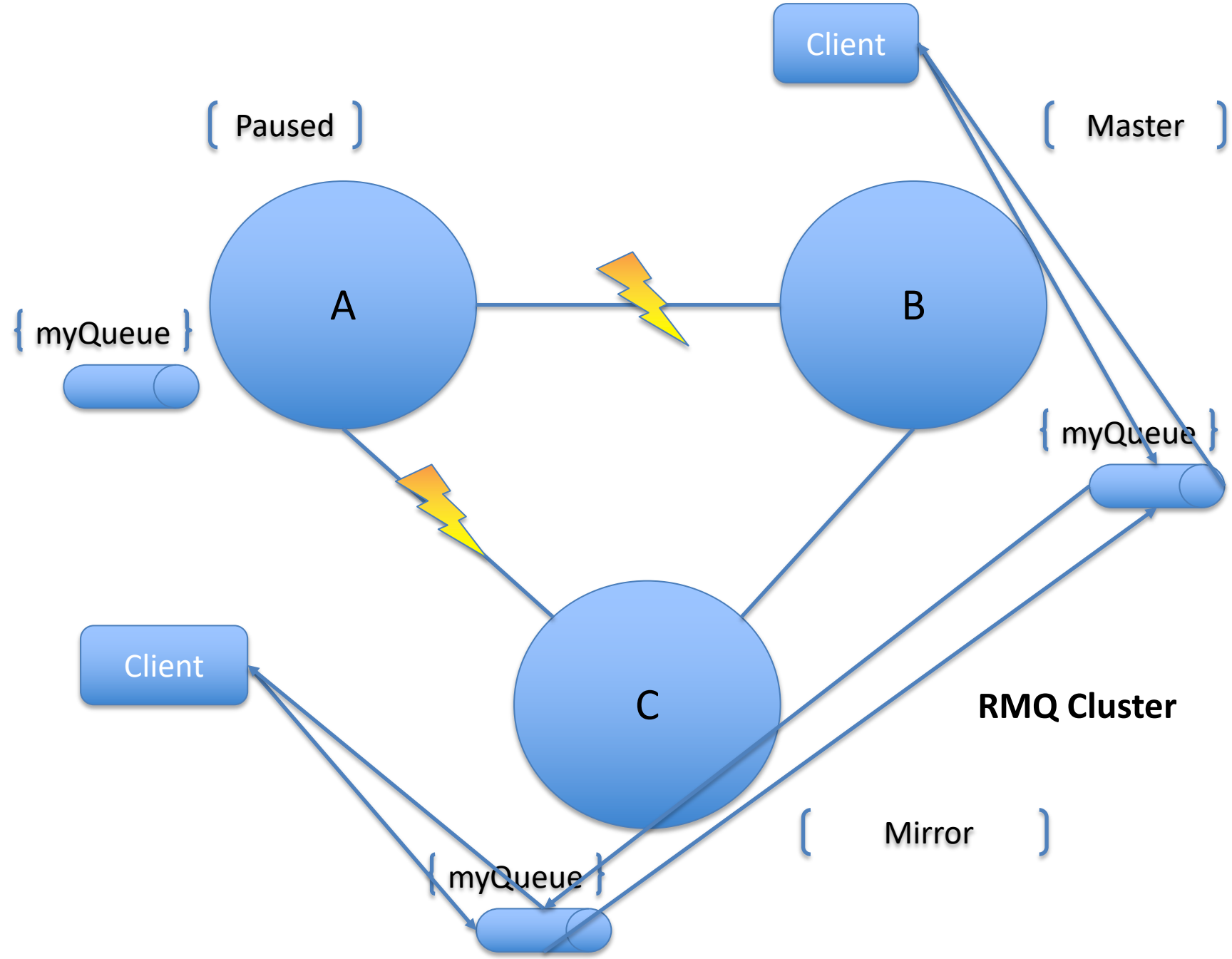

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Paused
myQueue

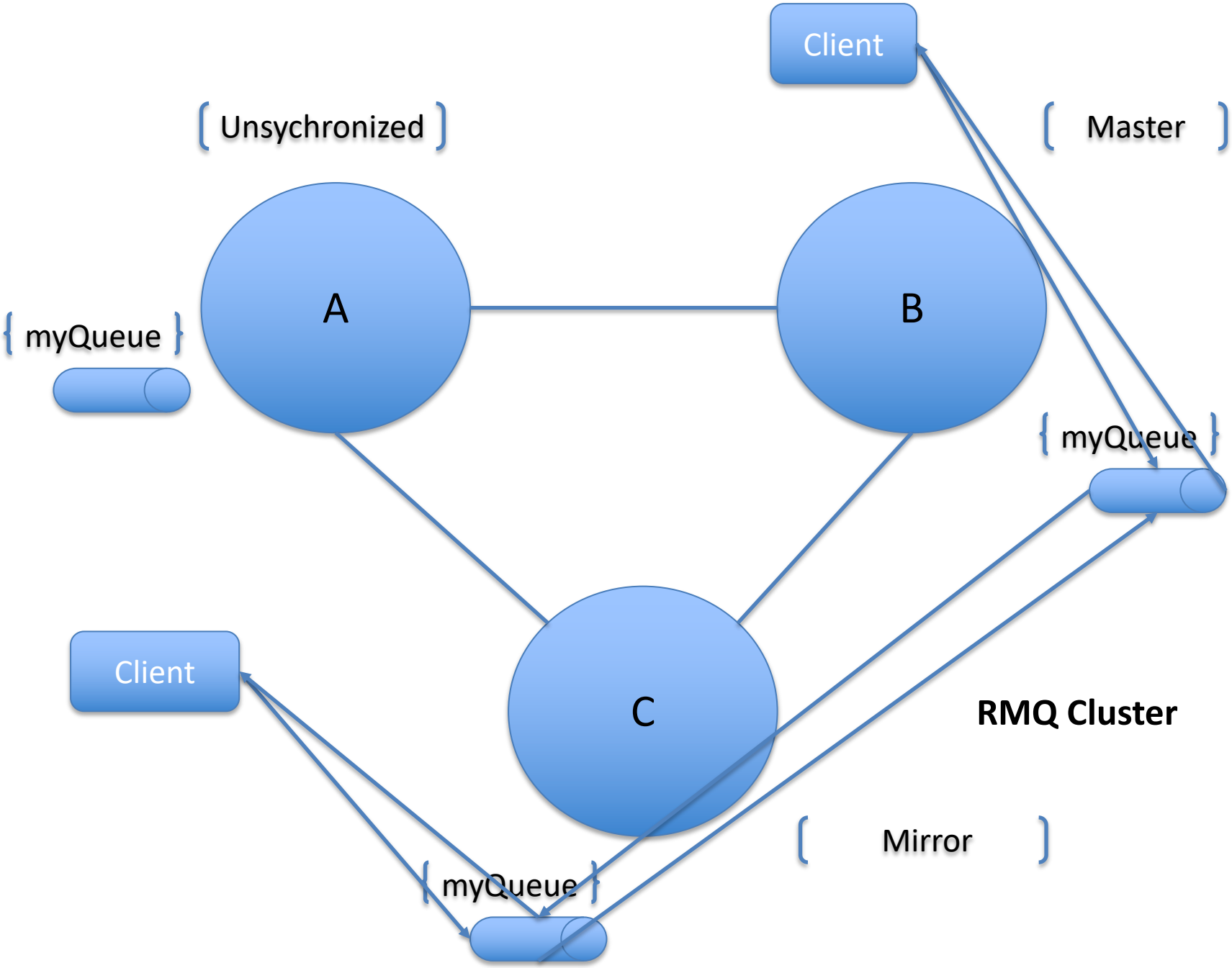

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Unsychronized
myQueue

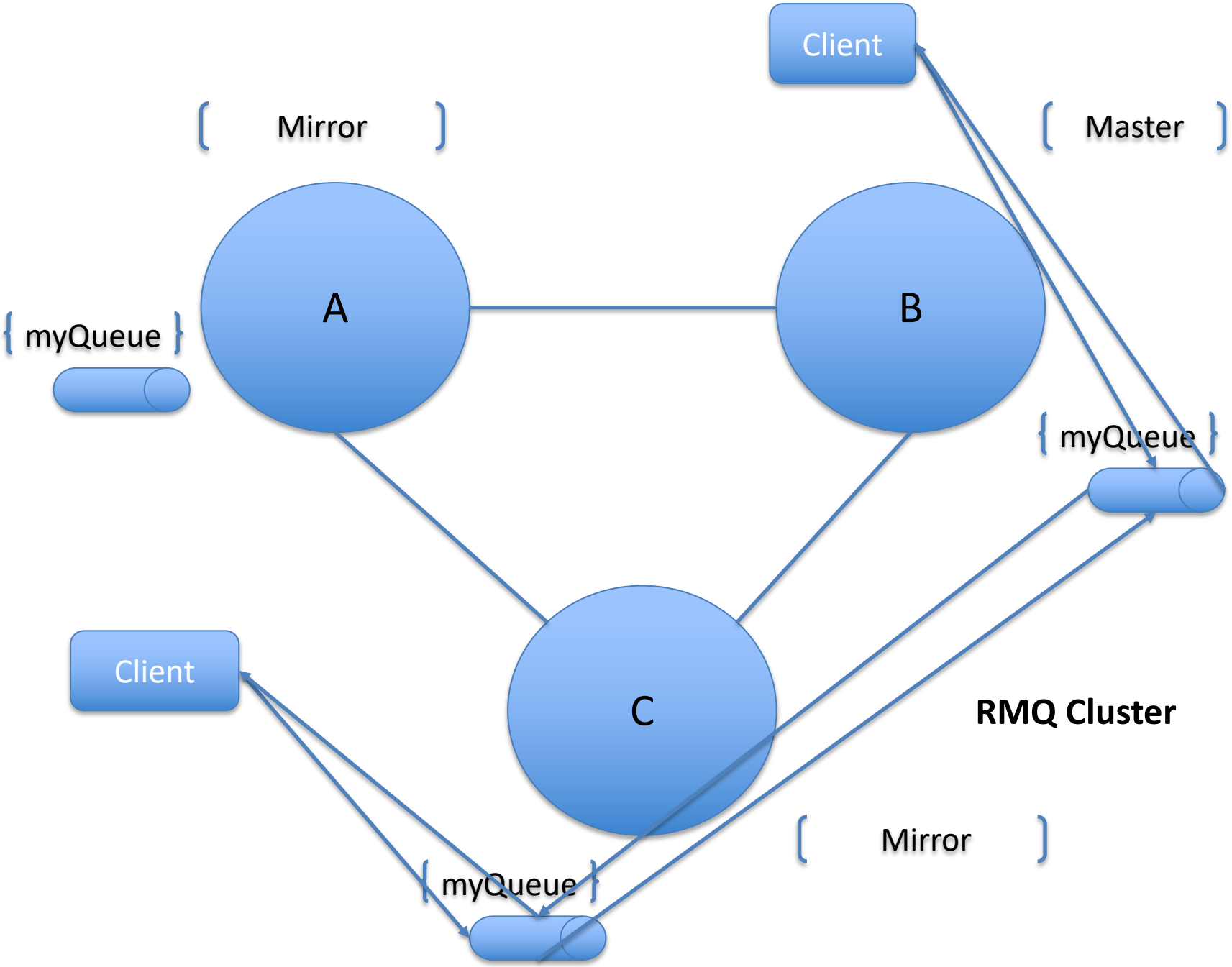

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

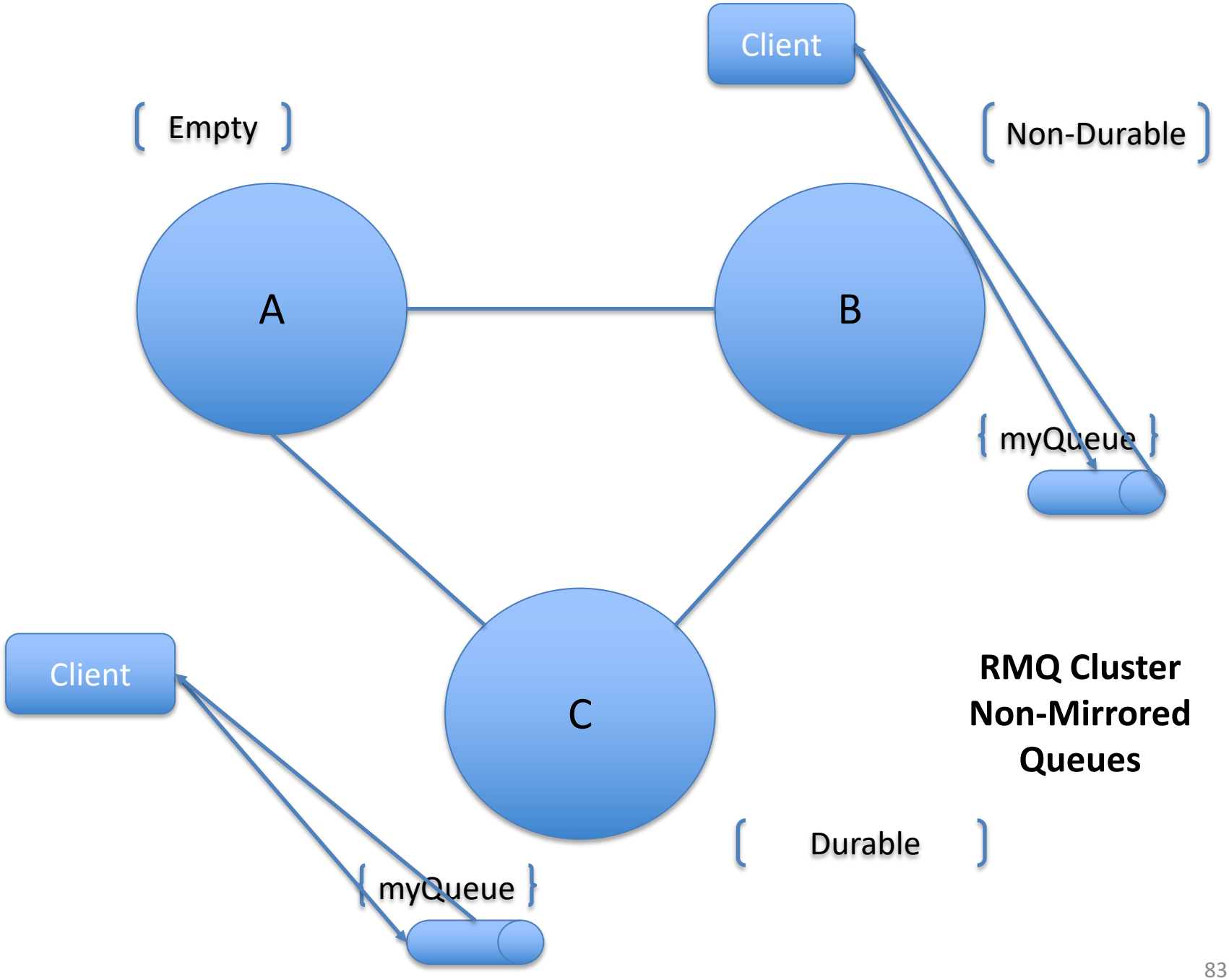

Client
Master
Mirror
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

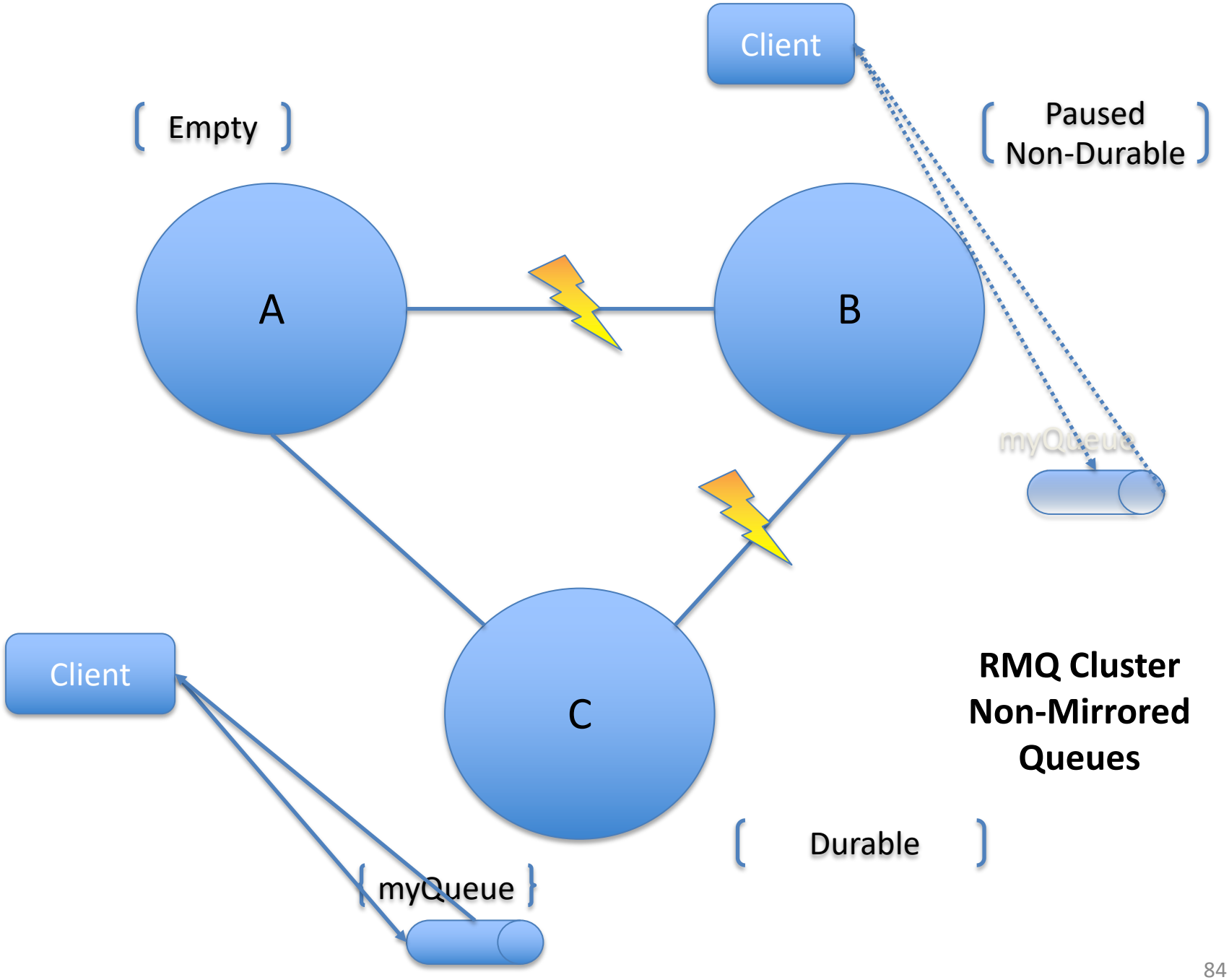

Client
Paused
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

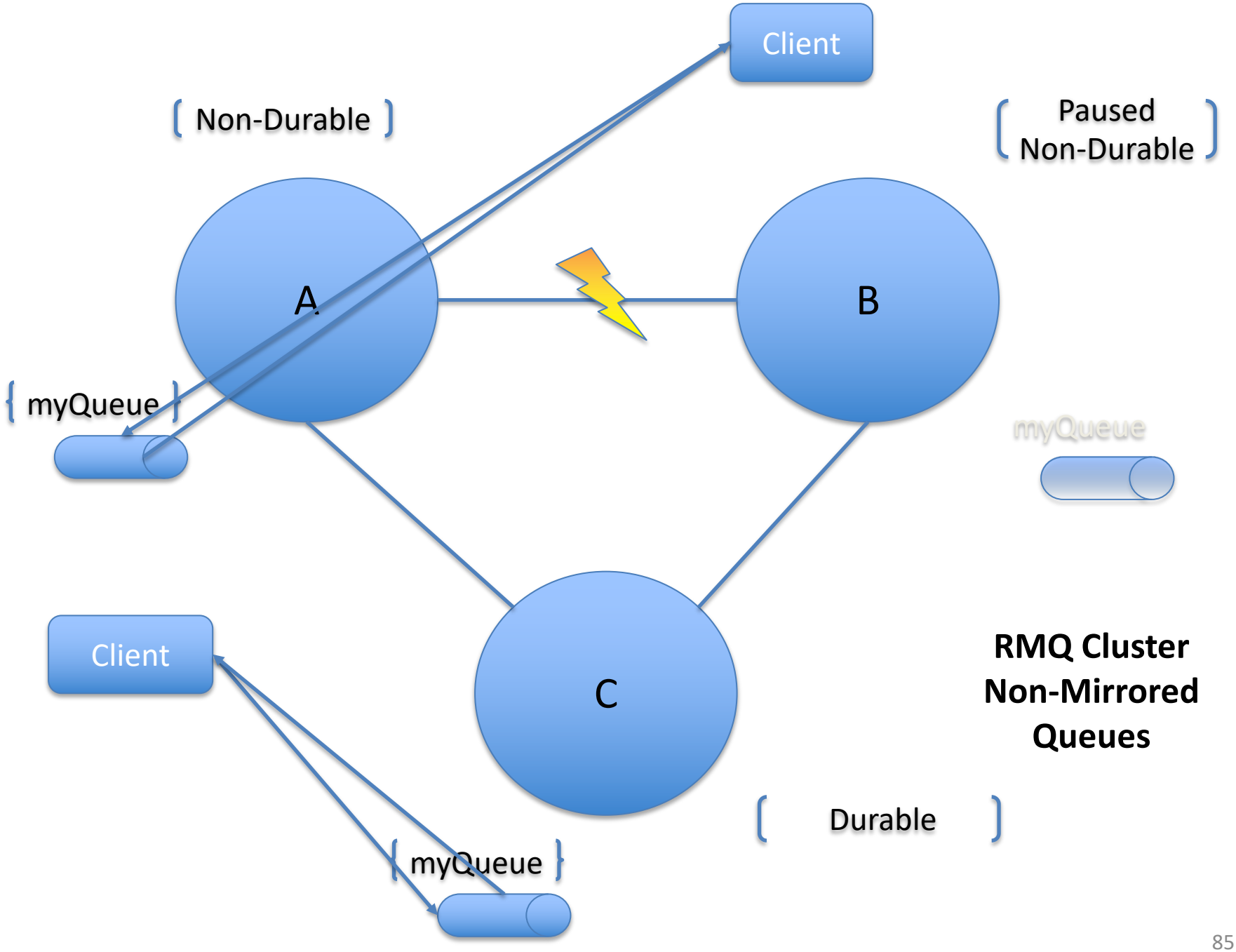

Client
Paused
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

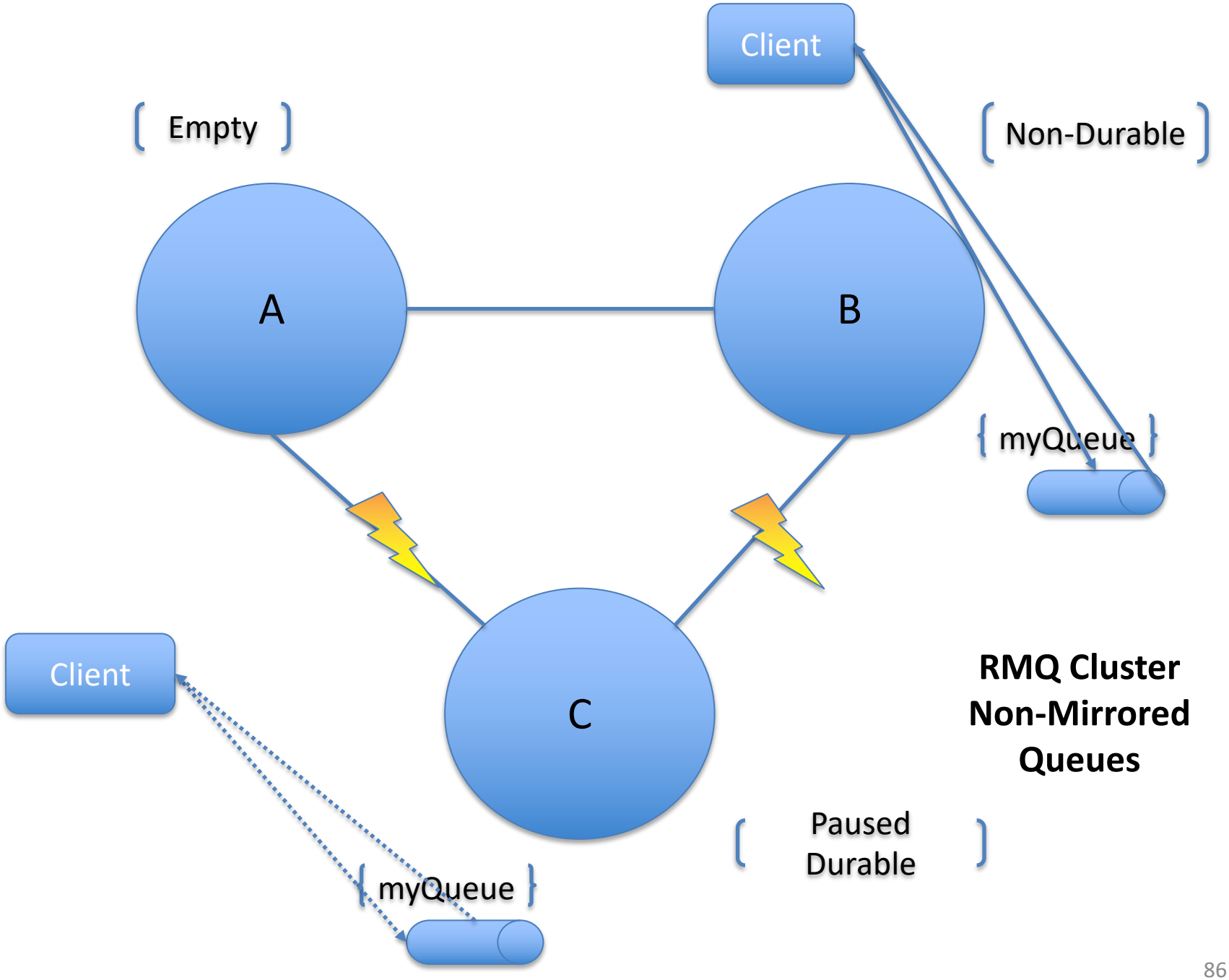

Client
Unsychronized
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

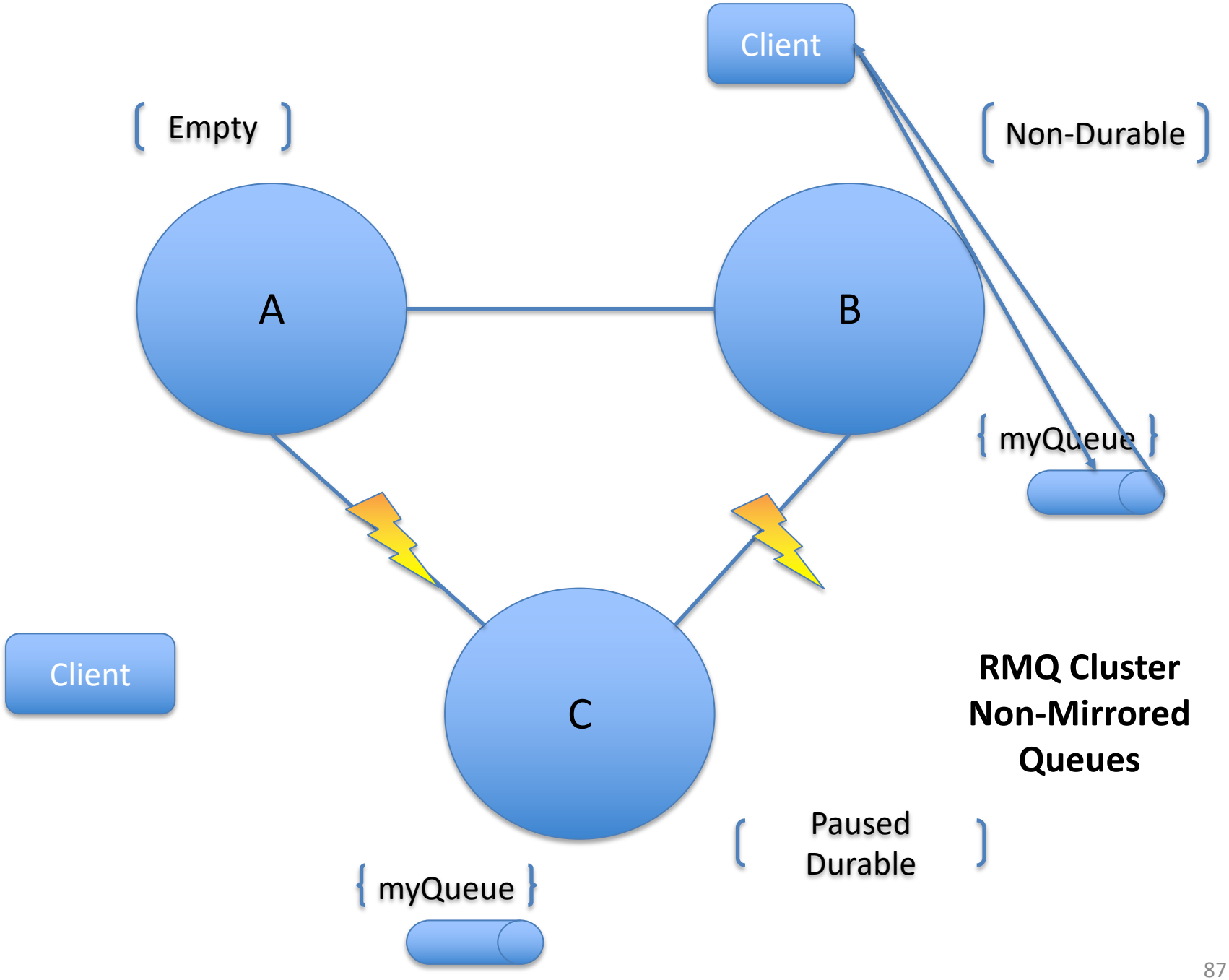

Client
Mirror
Master
A
B
myQueue
myQueue
Client
C
RMQ Cluster
Mirror
myQueue

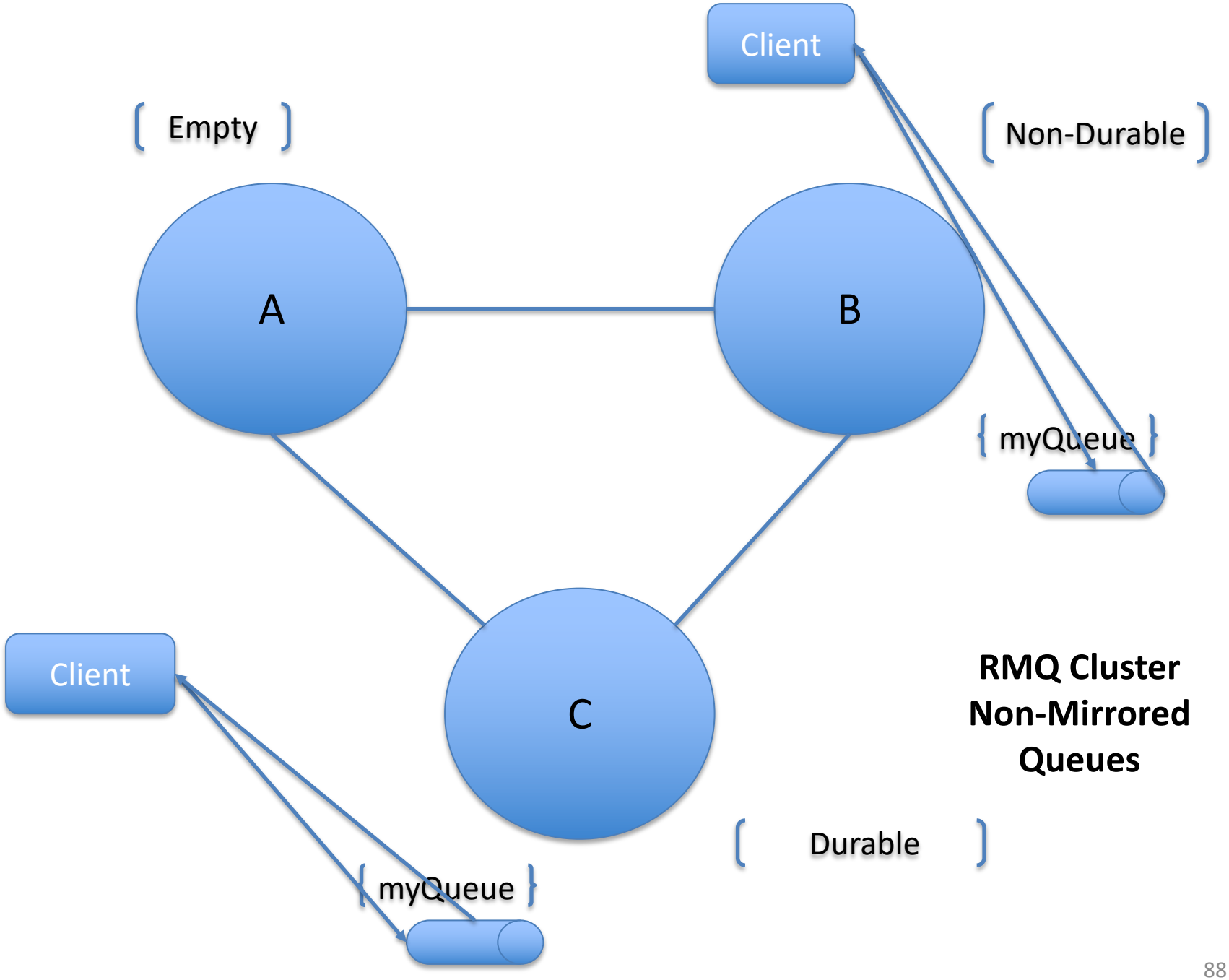

Client
Empty
Non-Durable
A
B
myQueue
Client
C
RMQ Cluster
Non-Mirrored
Queues
Durable
myQueue


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

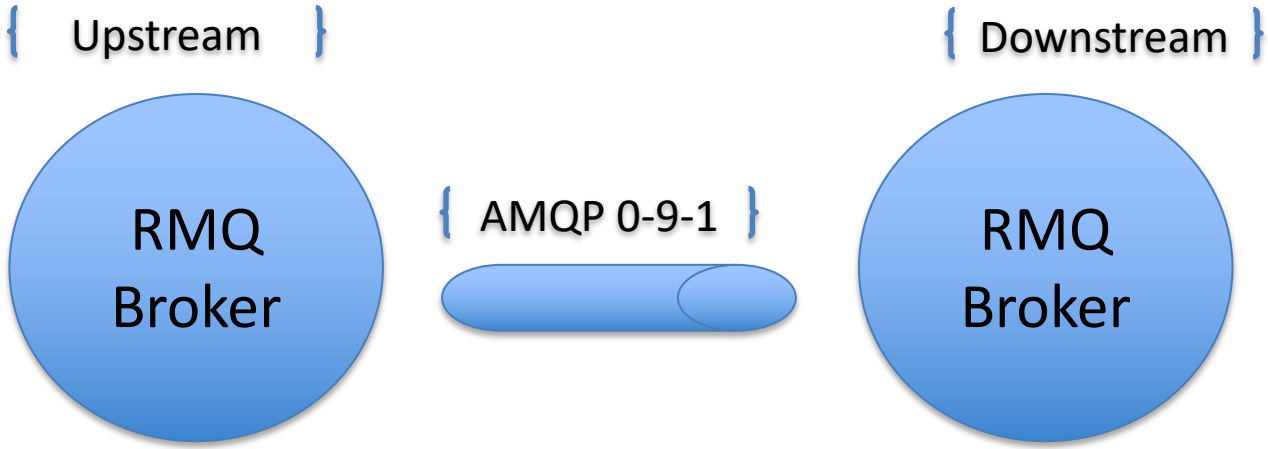
So which CAP options does an RMQ cluster support?

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



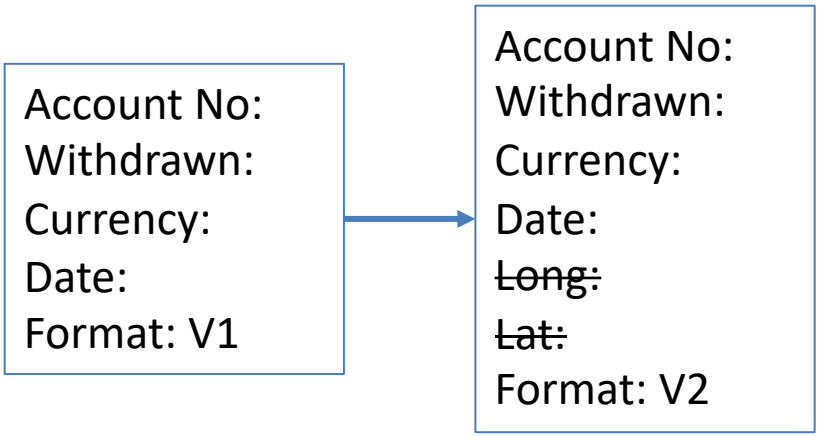
RMQ Federation/Shovel

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.

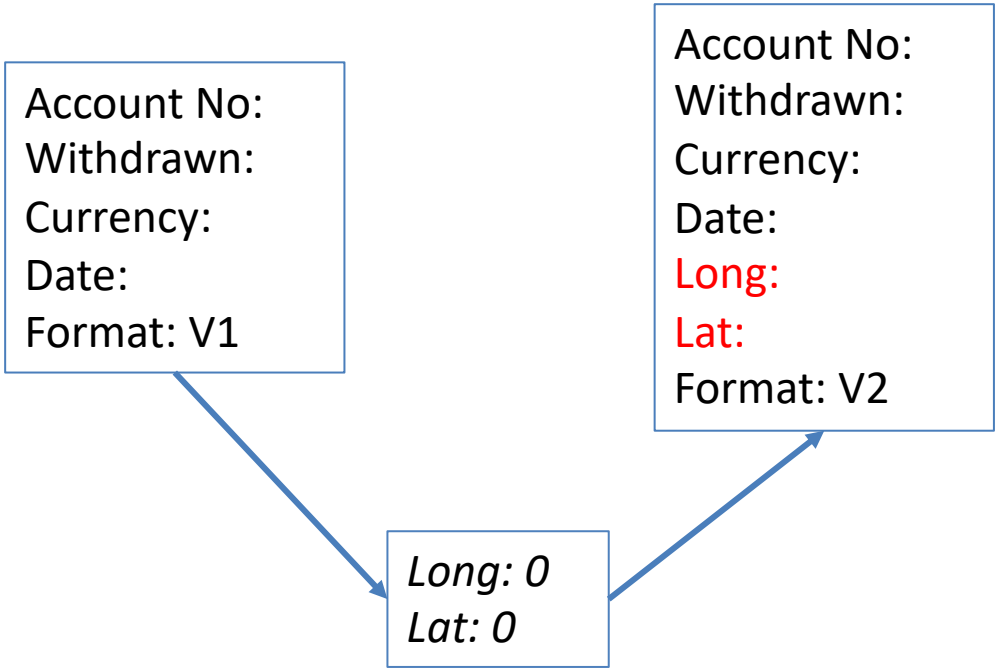
Robustness Principal or Postel's Law – Jon Postel RFC 1958

Tolerant Reader



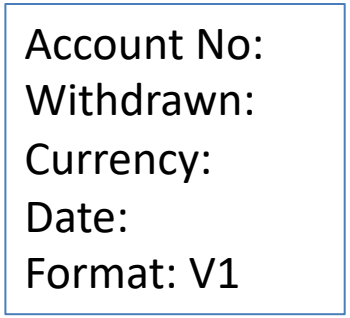
Ignore New Fields

Tolerant Reader

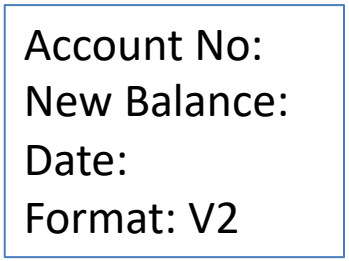


Default Missing Fields

Breaking Change



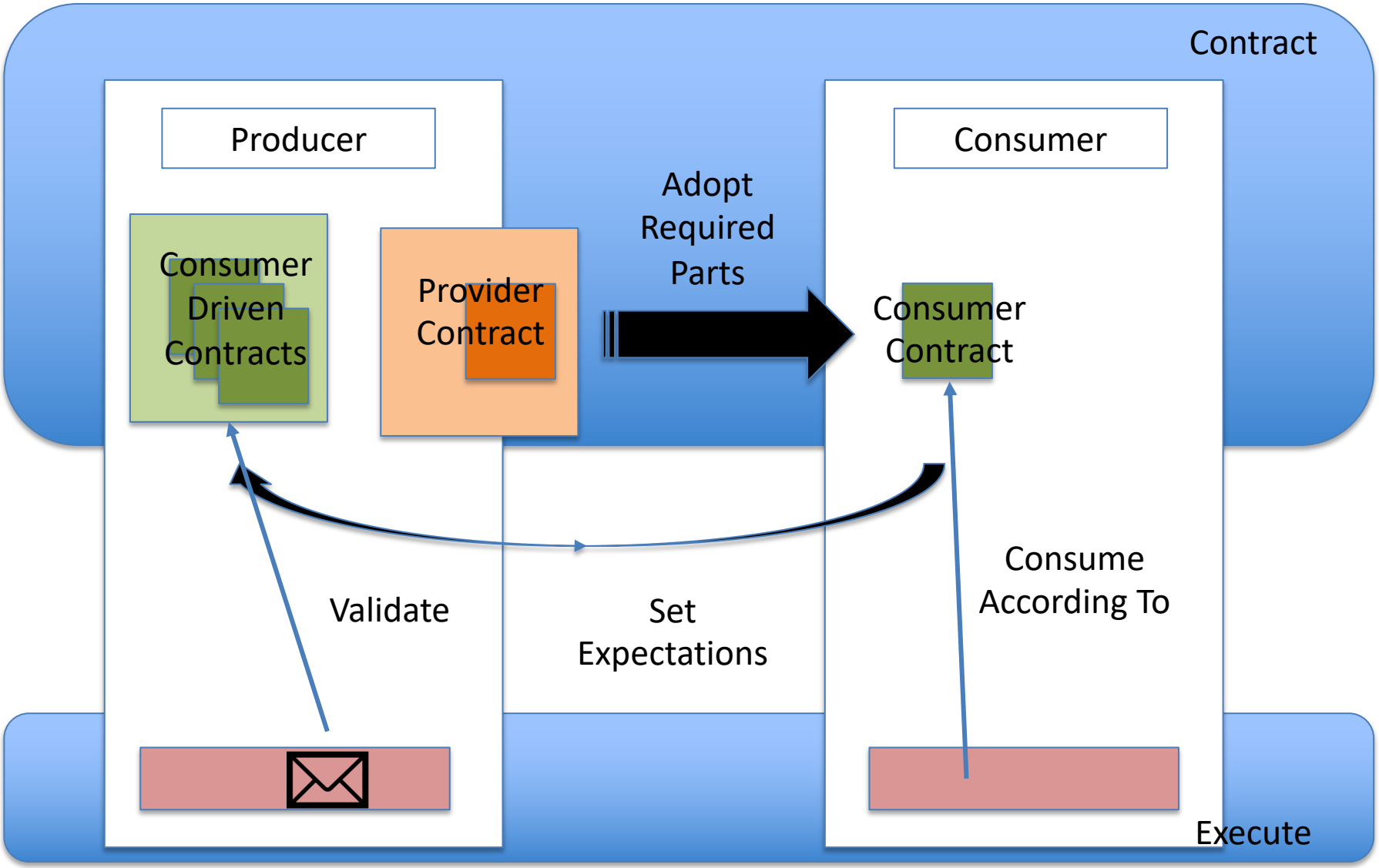
Account.Withdrawal.Event



Account.NewBalance.Event

New Message

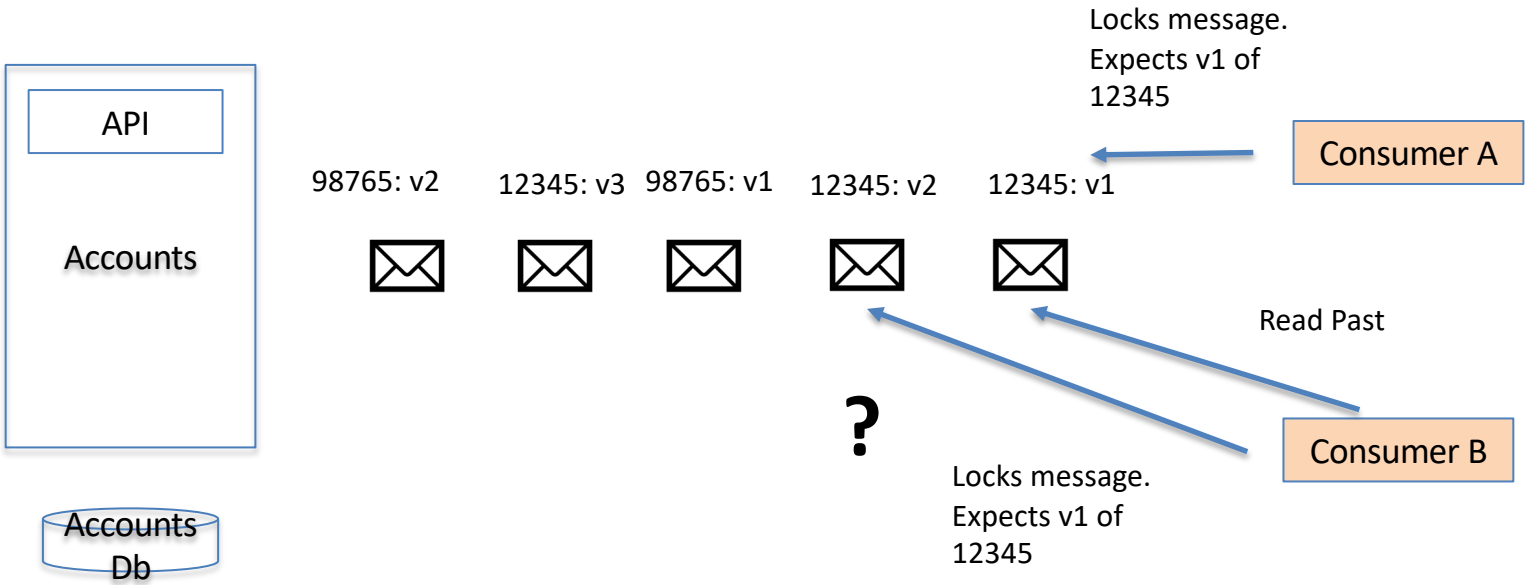
Consumer Driven Contracts



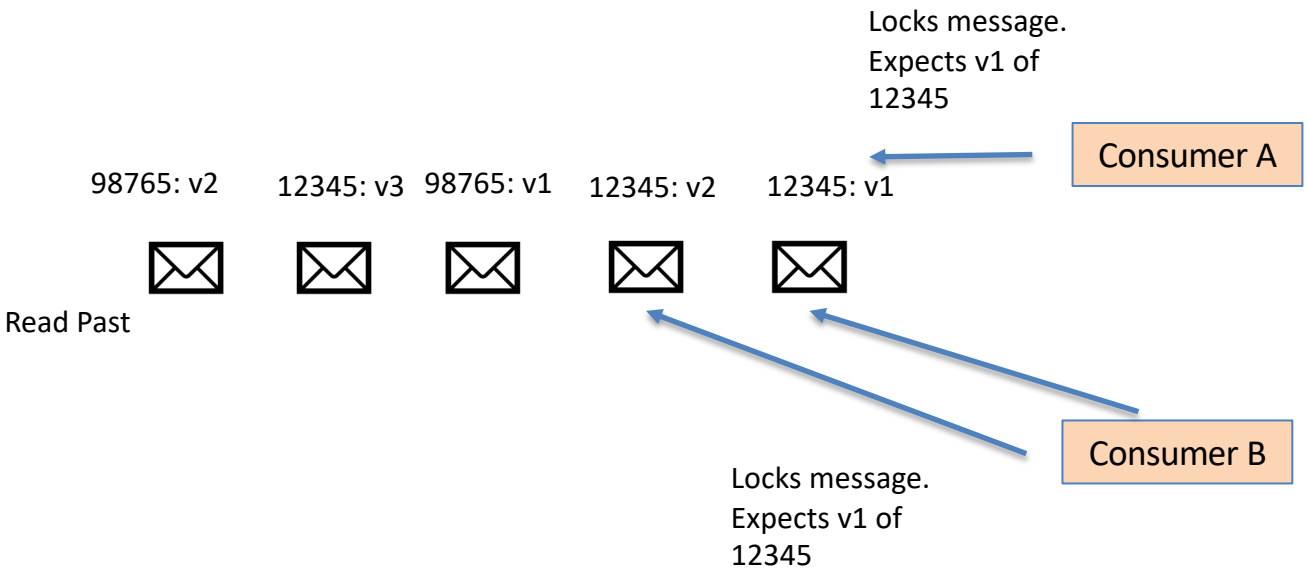
In Order Delivery

5. ORDER

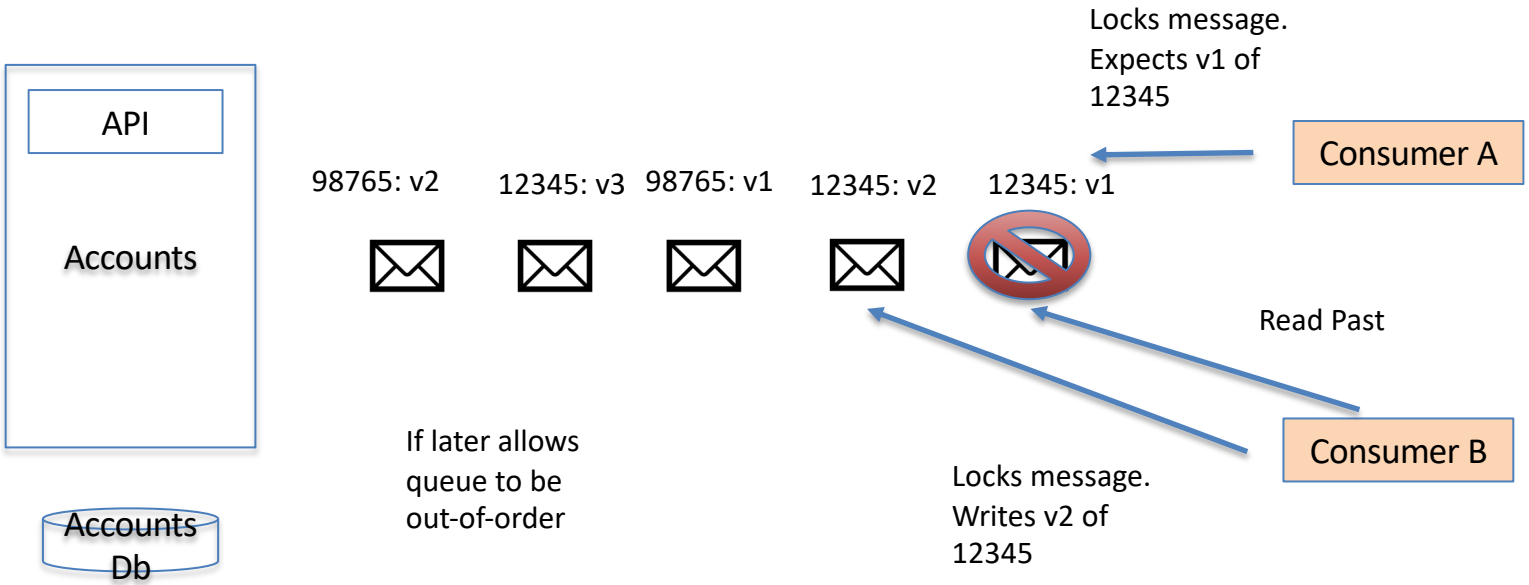
Messaging Order and Competing Consumers



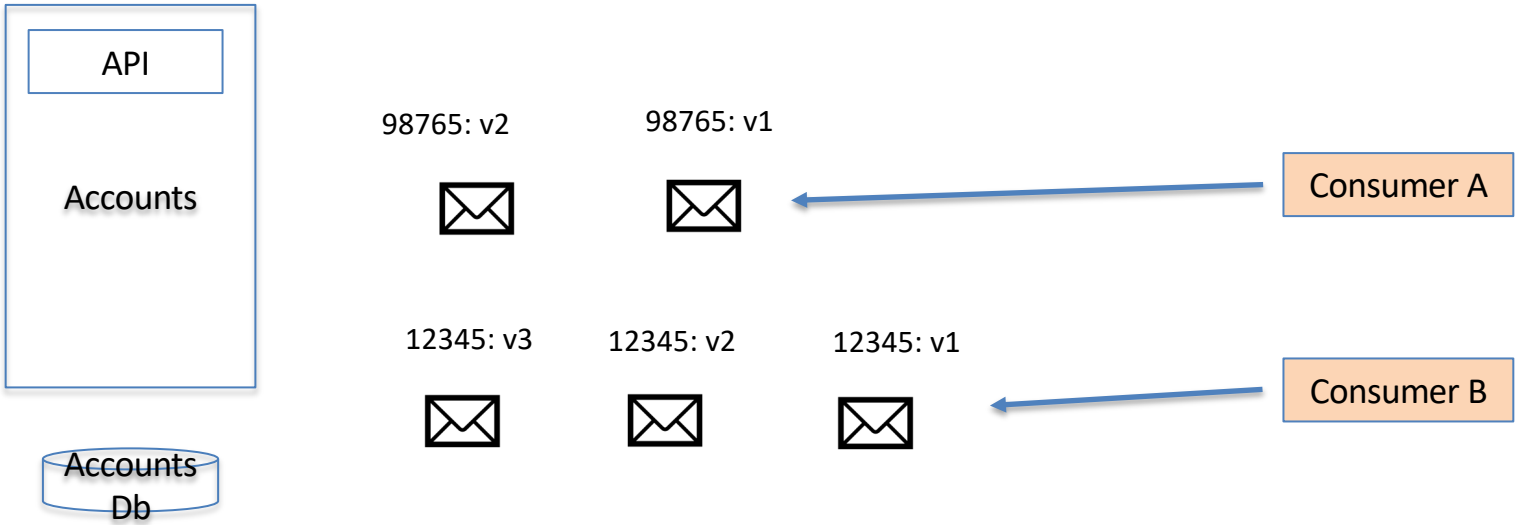
Requeue?


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

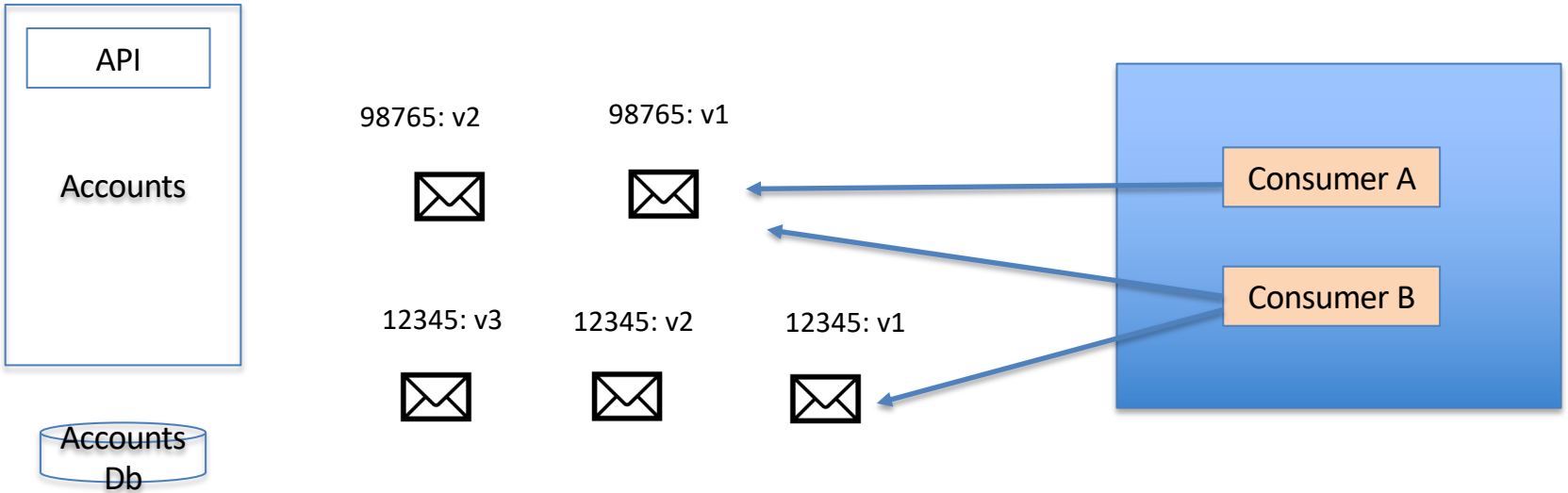
If Later


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

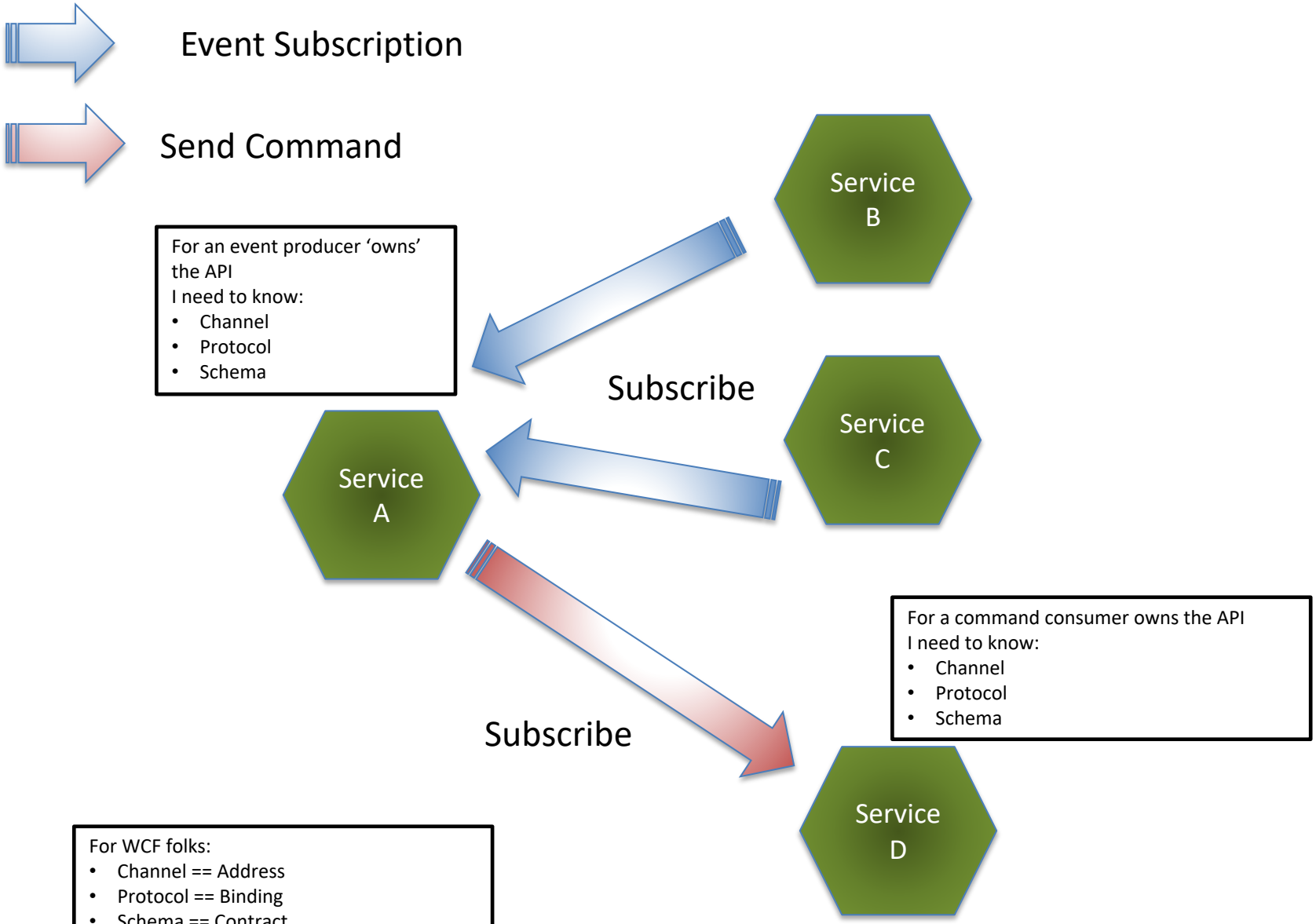


Consistent Hashing w. Consumer Groups

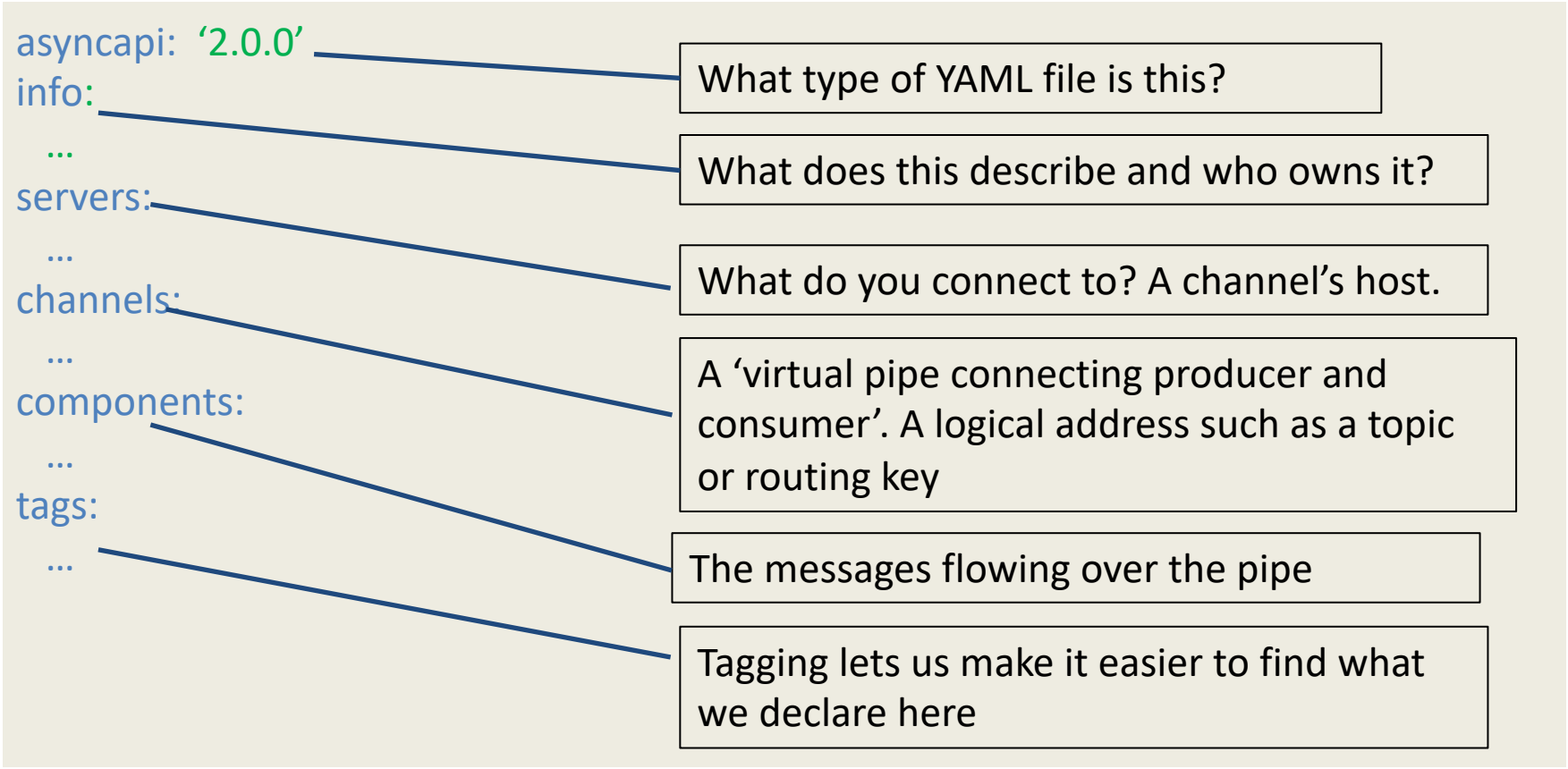


Standards & How to Describe Messaging

6. DOCUMENTATION

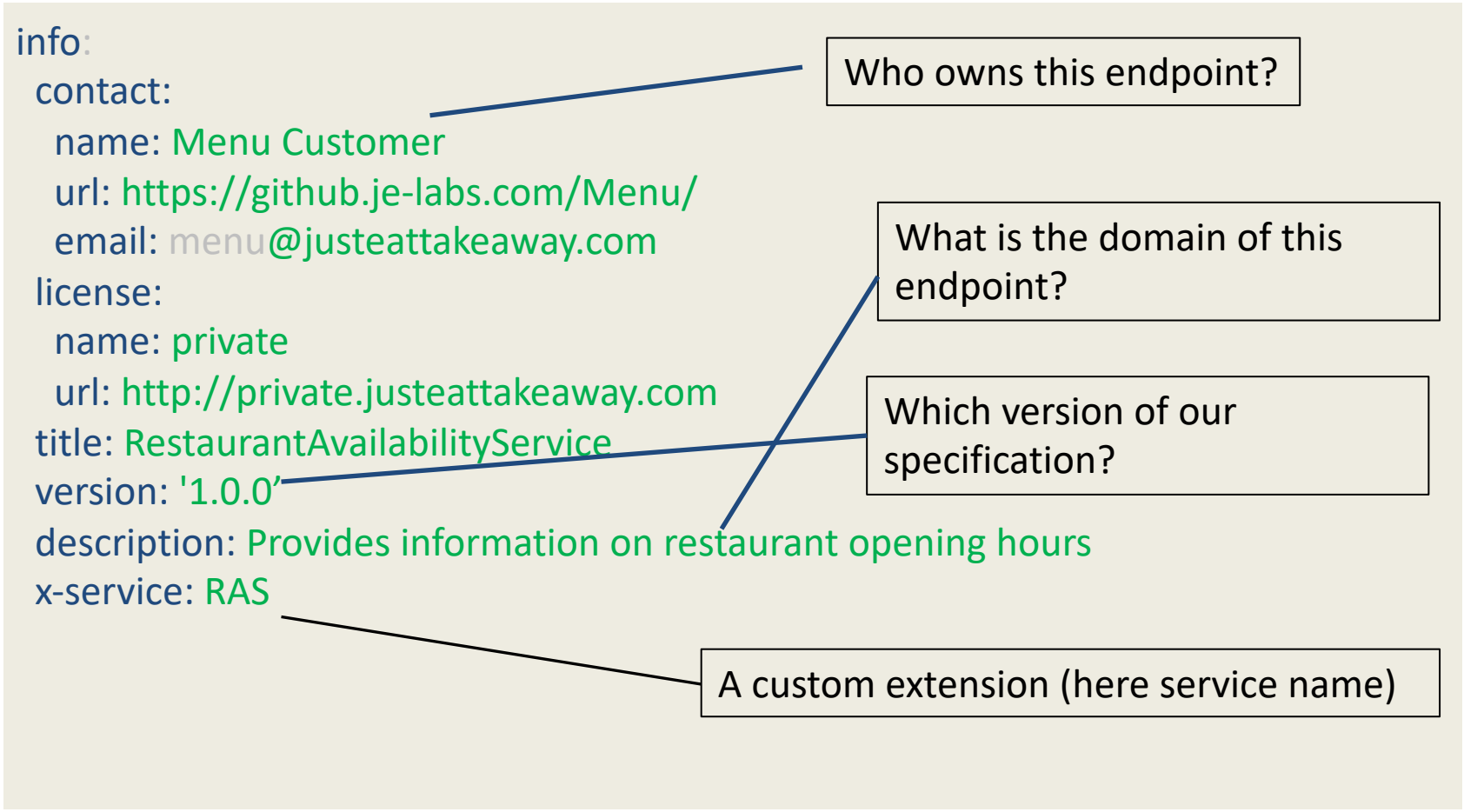

Event Subscription
Send Command
Service B
For an event producer 'owns' the API
I need to know:
• Channel
• Protocol
• Schema
Subscribe
Service C
Service A
For a command consumer owns the API
I need to know:
• Channel
• Protocol
• Schema
Subscribe
Service D
For WCF folks:
• Channel == Address
• Protocol == Binding
• Schema == Contract

AsyncAPI Document Structure



Info Object

```
info:
  contact:
    name: Menu Customer
    url: https://github.je-labs.com/Menu/
    email: menu@justeattakeaway.com
  license:
    name: private
    url: http://private.justeattakeaway.com
  title: RestaurantAvailabilityService
  version: '1.0.0'
  description: Provides information on restaurant opening hours
  x-service: RAS
```



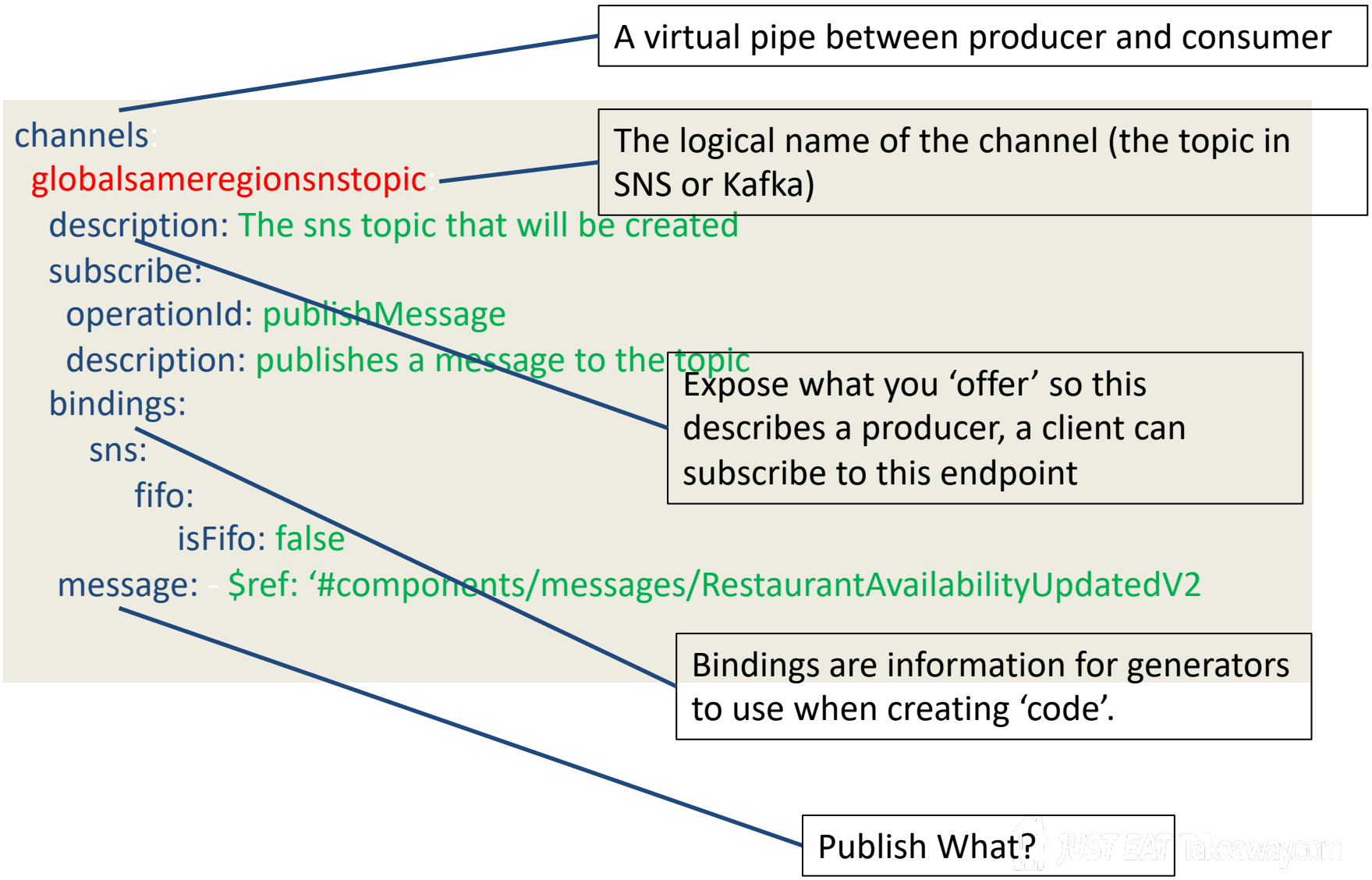
Server Object

```
servers:
  production-sns:
    url: https://sns.{region}.amazonaws.com
    protocol: http
    description: Simple Notification Service
    variables:
      region:
       description: The Region. For example, us-east-2 for US East (Ohio)
       default: eu-west-1
      enum:
        - eu-west-1
        - us-east-1
        - us-west-1
        - us-west-2
```

Who hosts the messaging middleware? Usually – the broker.

The protocol we talk to the server

Channels - Producer



Channels - Consumer

A virtual pipe between producer and consumer

```
channels:
  globalsameregionsnstopic
    description: The sns topic that will be created in same region as the queue
    publish:
      operationId: receiveMessage
      description: receive messages from the topic
      bindings:
        sqs:
          name: queueName
          visibilityTimeout: 10
          redrivePolicy:
            maxReceiveCount: 30
            dlq:
              delaySeconds: 0
    message: $ref: '#components/messages/RestaurantAvailabilityUpdatedV2
```

The logical name of the channel

Expose what you 'offer' so this describes a consumer, a producer can publish to this endpoint

If I do code generation, what do I call this?

How do we generate the code required to consume? In this case how do we define a queue to listen with?

Receive What?

Components

```
components:
  messages:
    RestaurantActualHoursUpdatedV2:
      name: RestaurantActualHoursUpdated
      title: Restaurant Actual Hours Updated
      summary: Changes to the delivery and collection times of the restaurant
      contentType: application/json
      payload: $ref: ”#/components/schemas/RestaurantActualHoursUpdatedV2"
      traits: $ref: ”#/components/messageTraits/JustSayingHeaders"
```

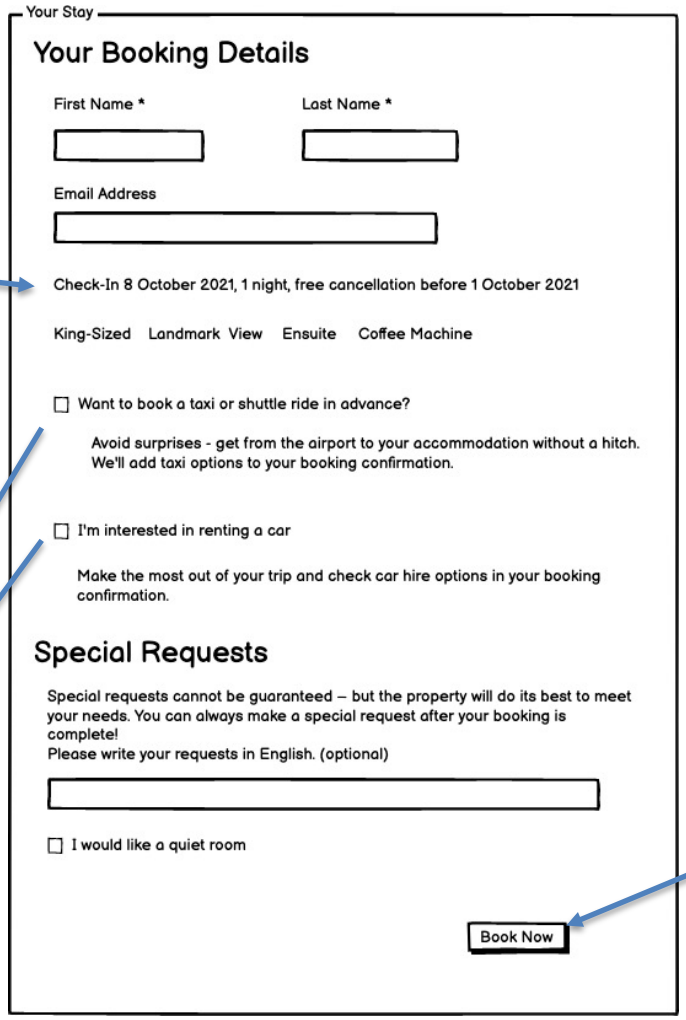
Components includes schemas that are shared in the file

What is our payload schema?

We can reference schemas within schemas here, breaking down into smaller ‘atomic’ units then composing up.

7. CONSUMERS

Task Based UI

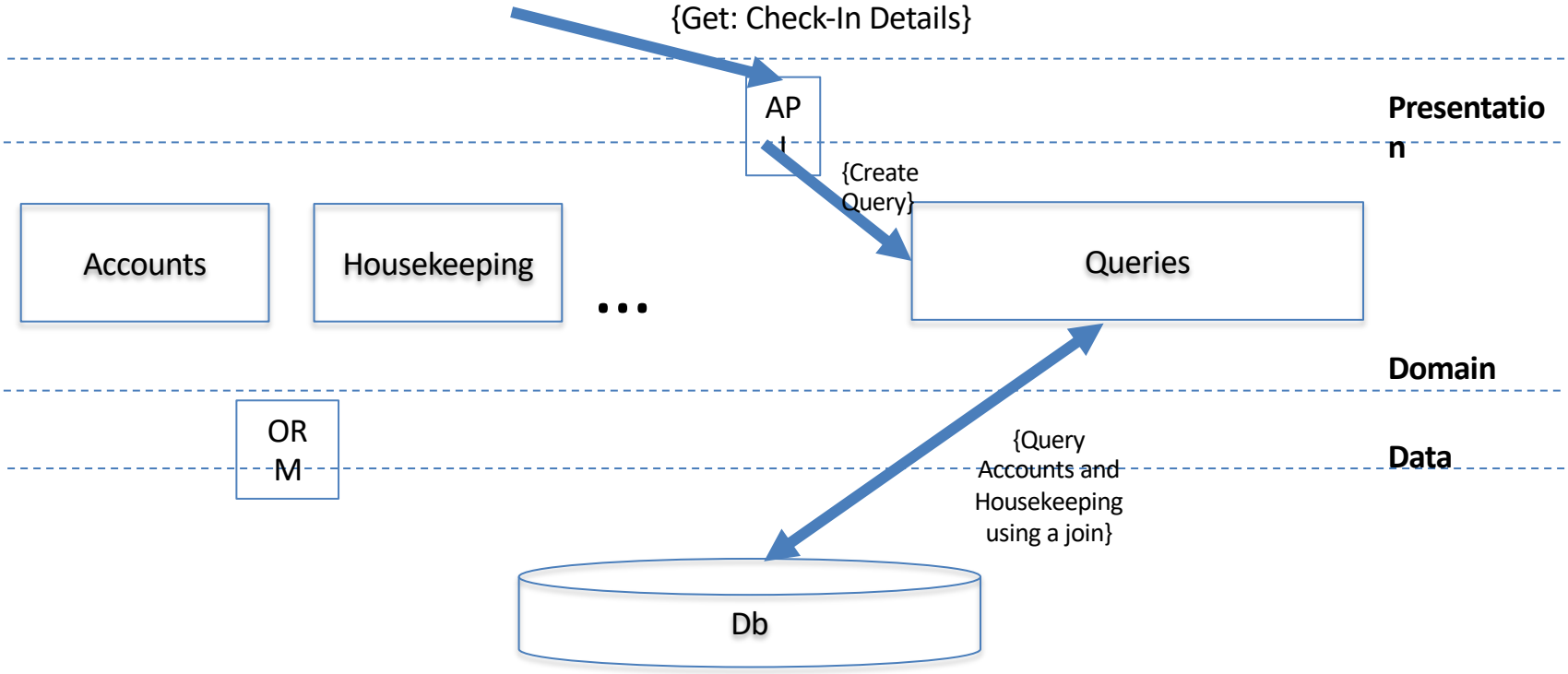


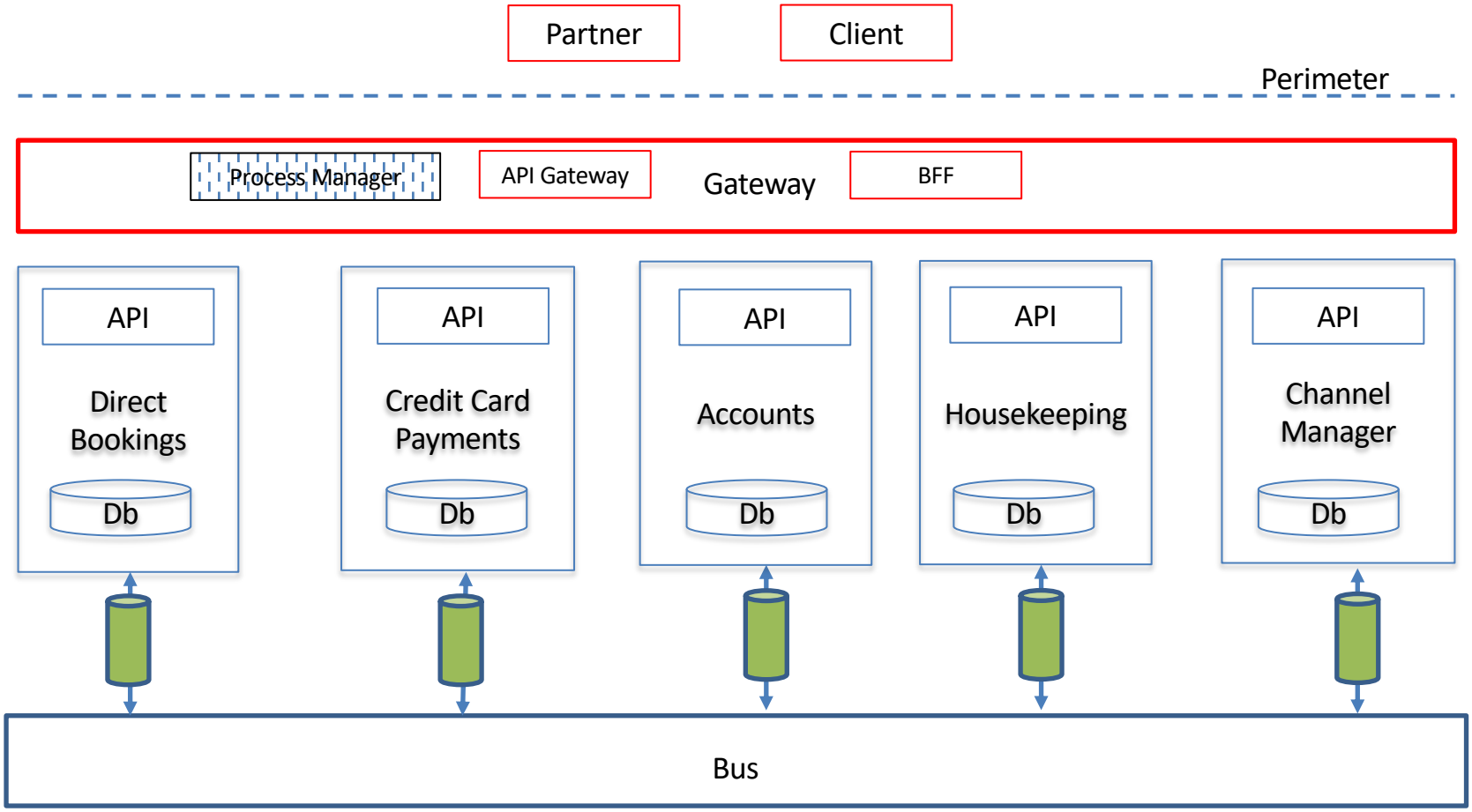
A sequence of screens can build up a request

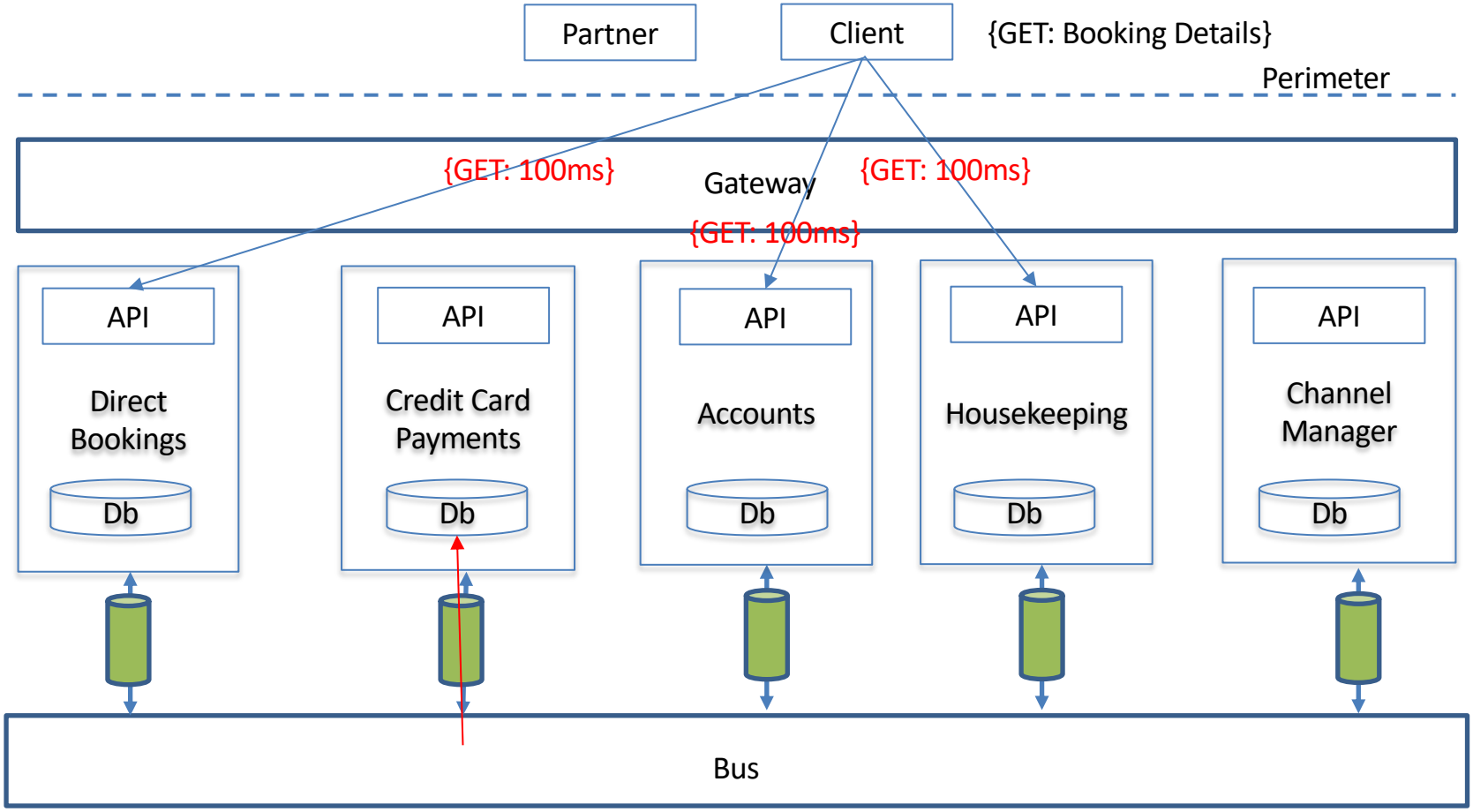
A sequence of screens can build up a request

At this point, the user expects async

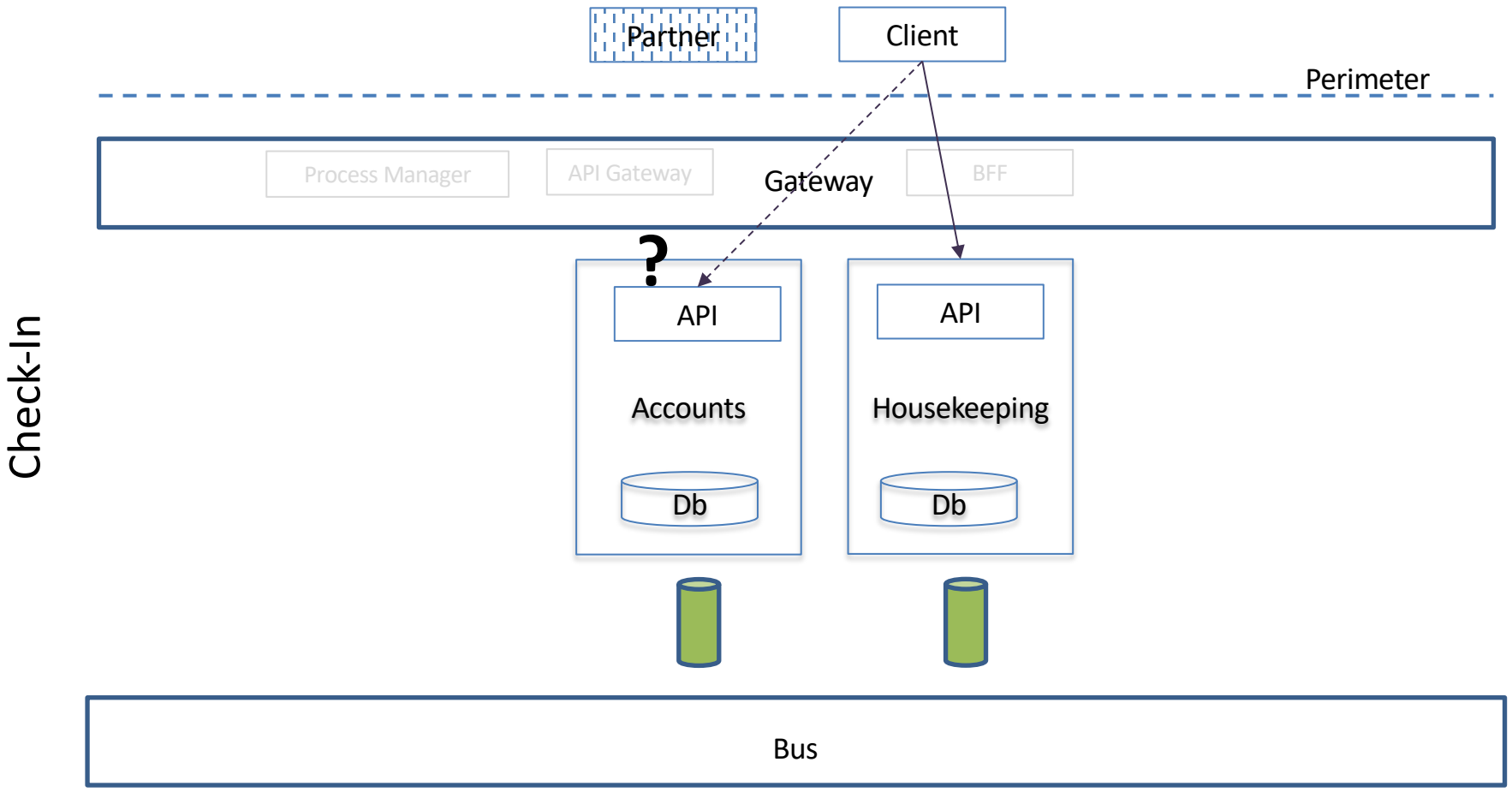
Getting work done in a Monolith




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

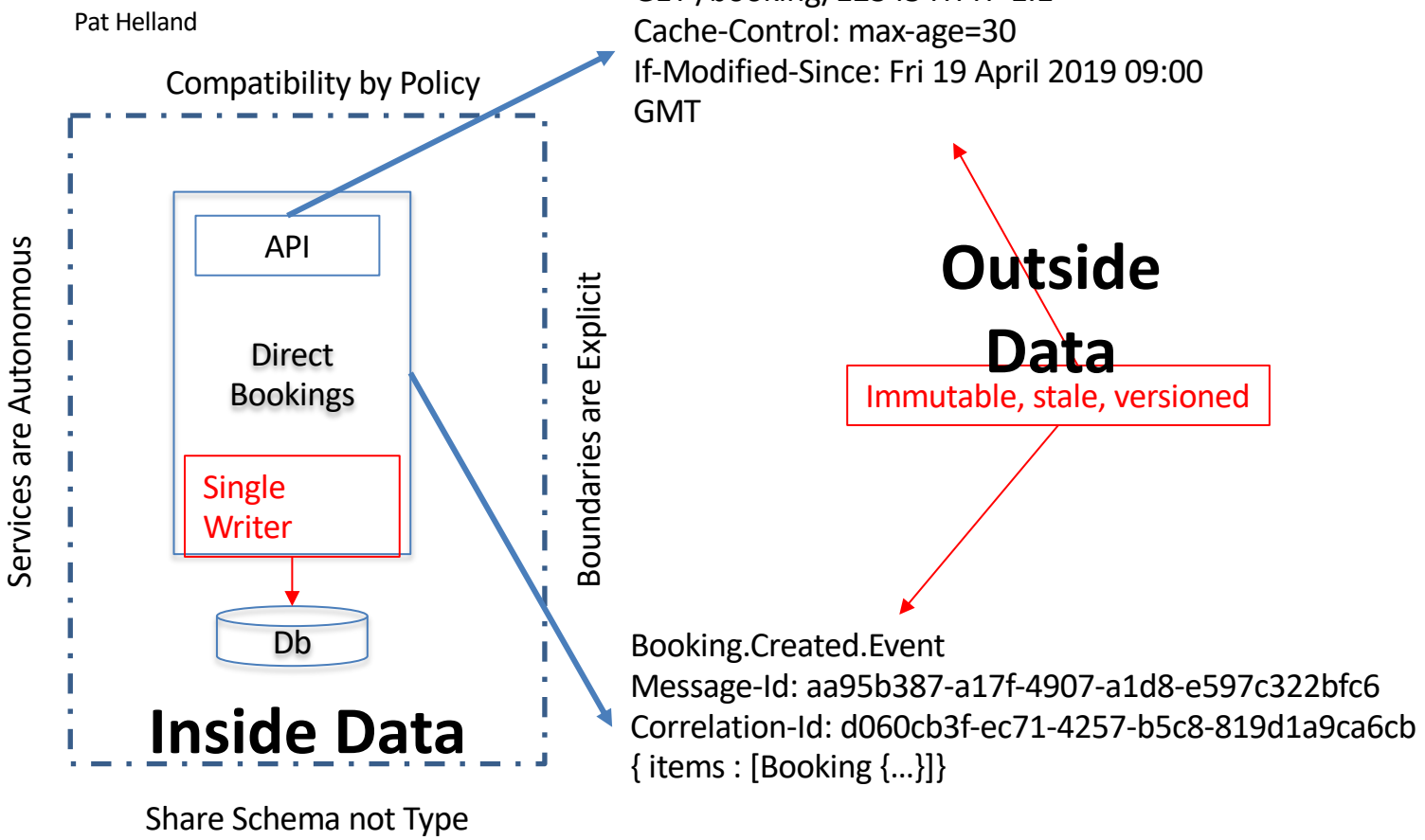

Partner
Client
{GET: Booking Details}
Perimeter
{GET: 100ms}
Gateway
{GET: 100ms}
{GET: 100ms}
API
Direct Bookings
Db
API
Credit Card Payments
Db
API
Accounts
Db
API
Housekeeping
Db
API
Channel Manager
Db
Bus

Check-In


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

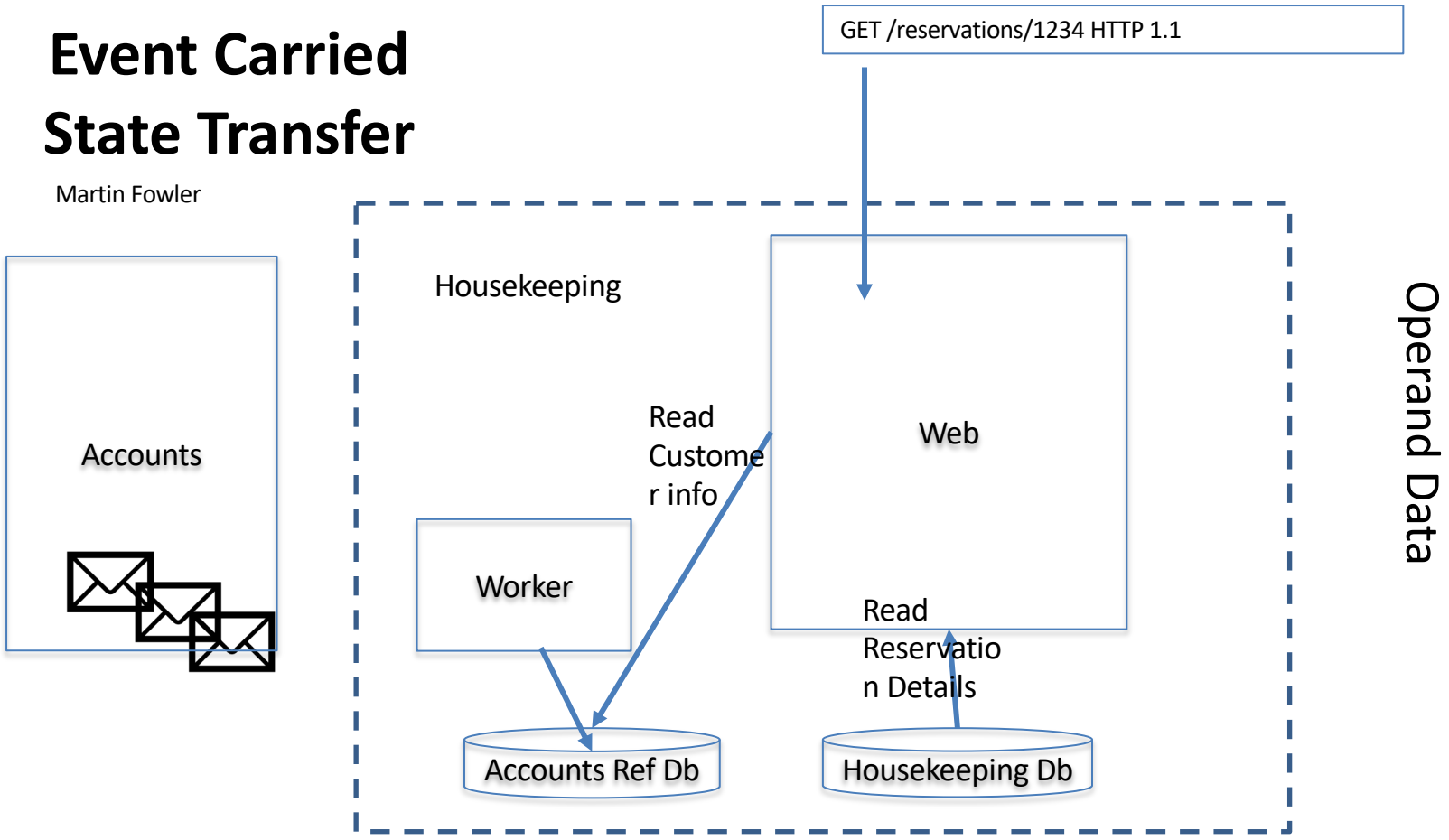
Reference Data

Pat Helland



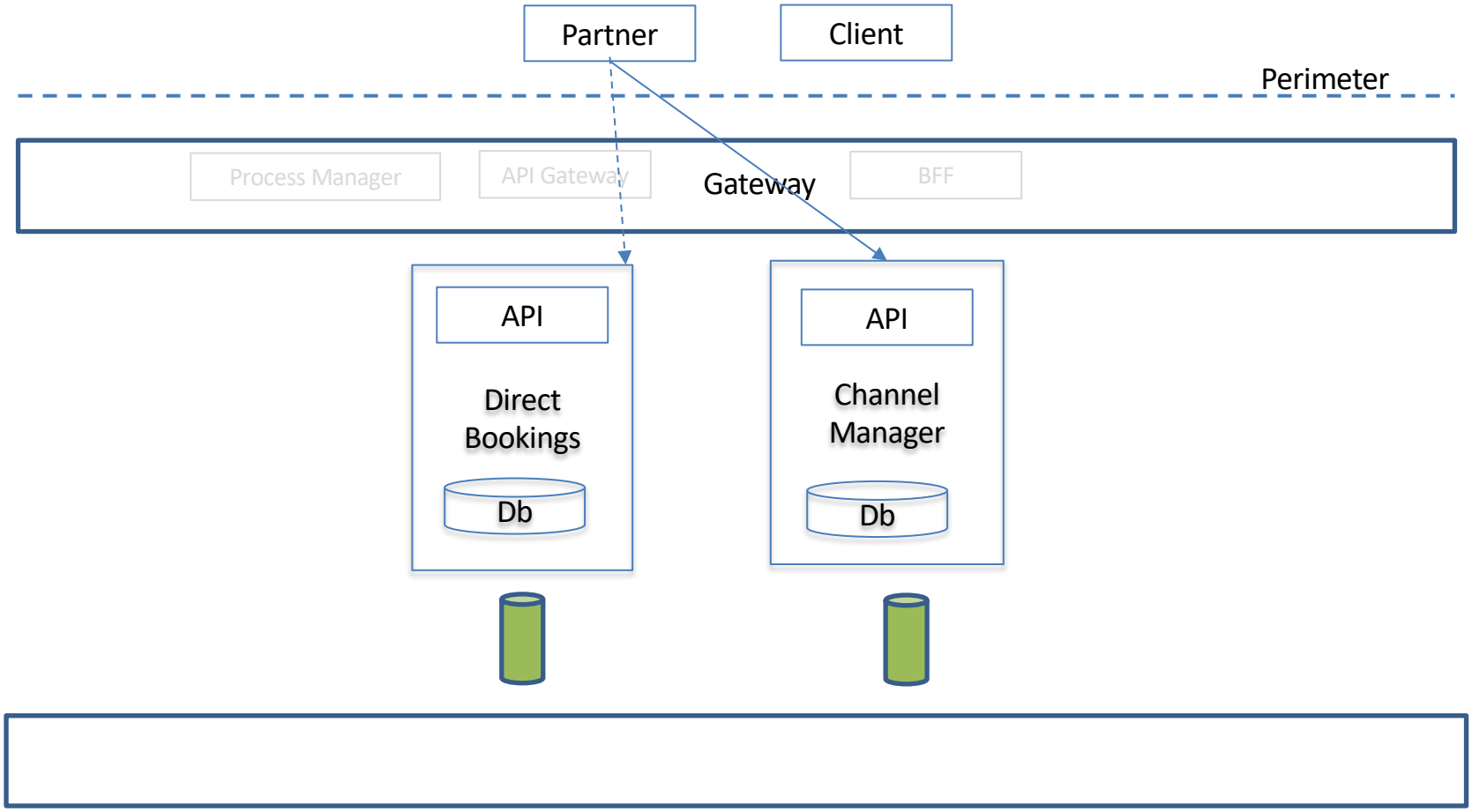
Event Carried State Transfer

Martin Fowler



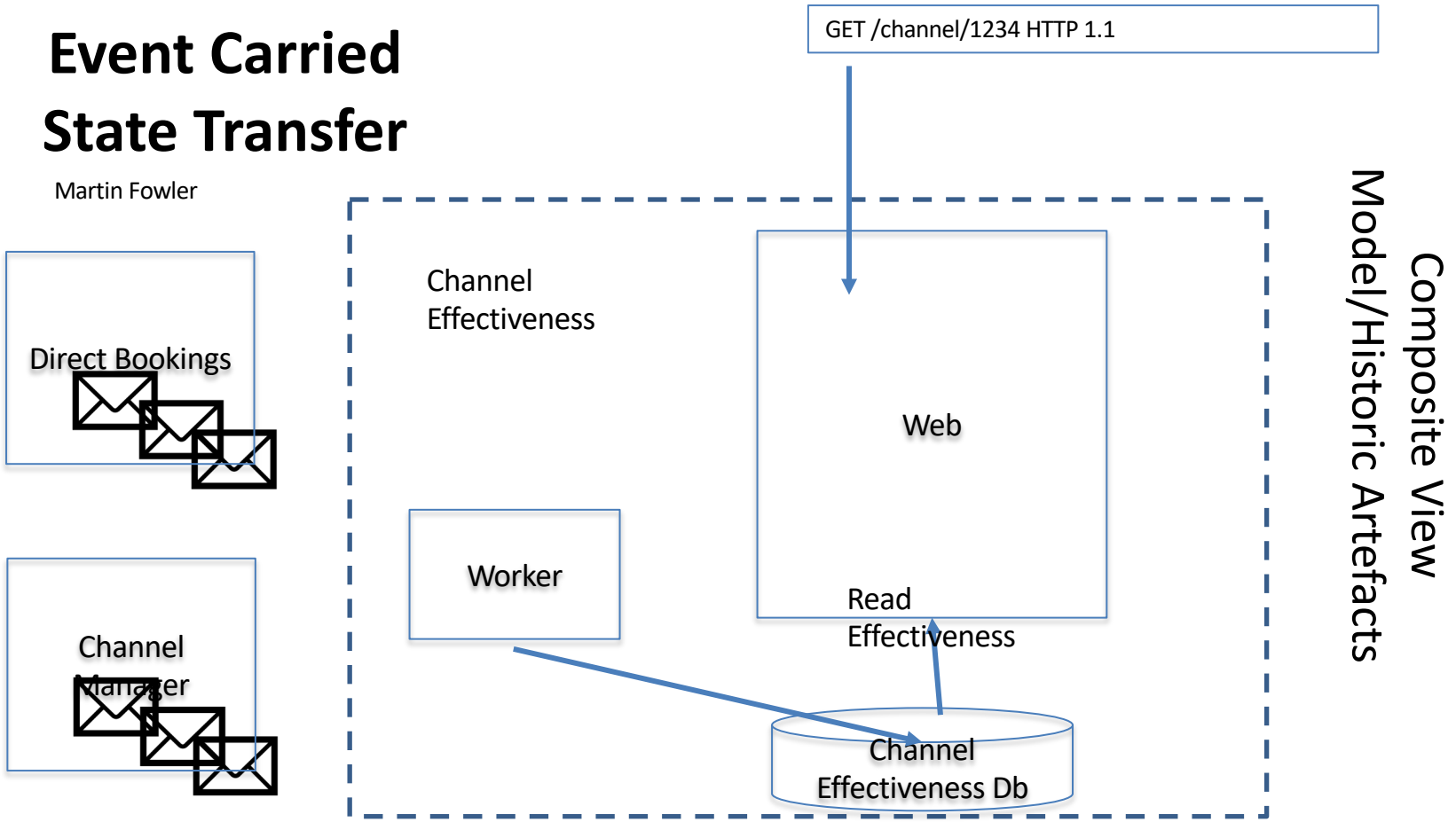
Channel Effectiveness

Channel Effectiveness



Event Carried State Transfer

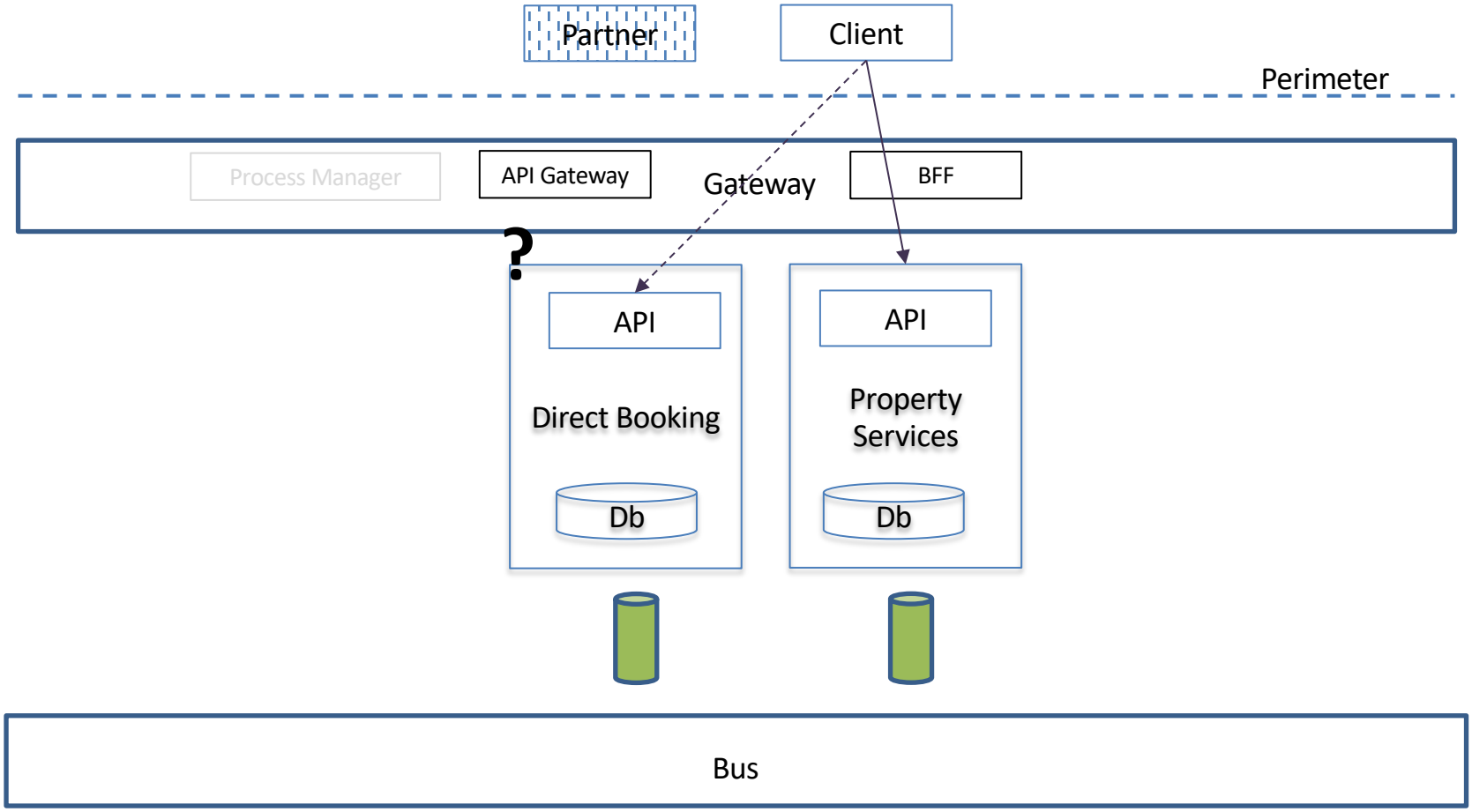
Martin Fowler



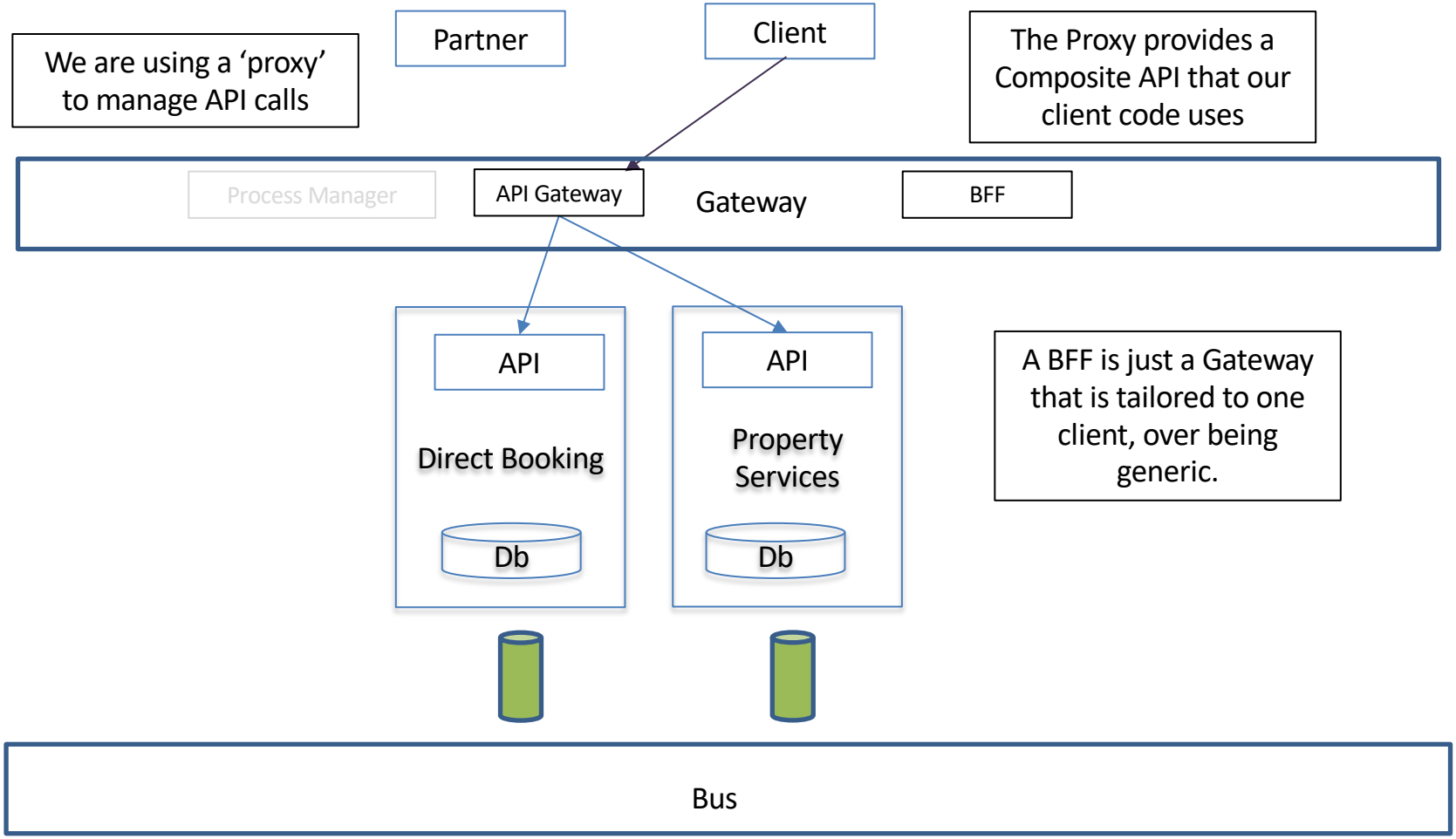
Composite View
Model/Historic Artefacts

Hotel Group Search

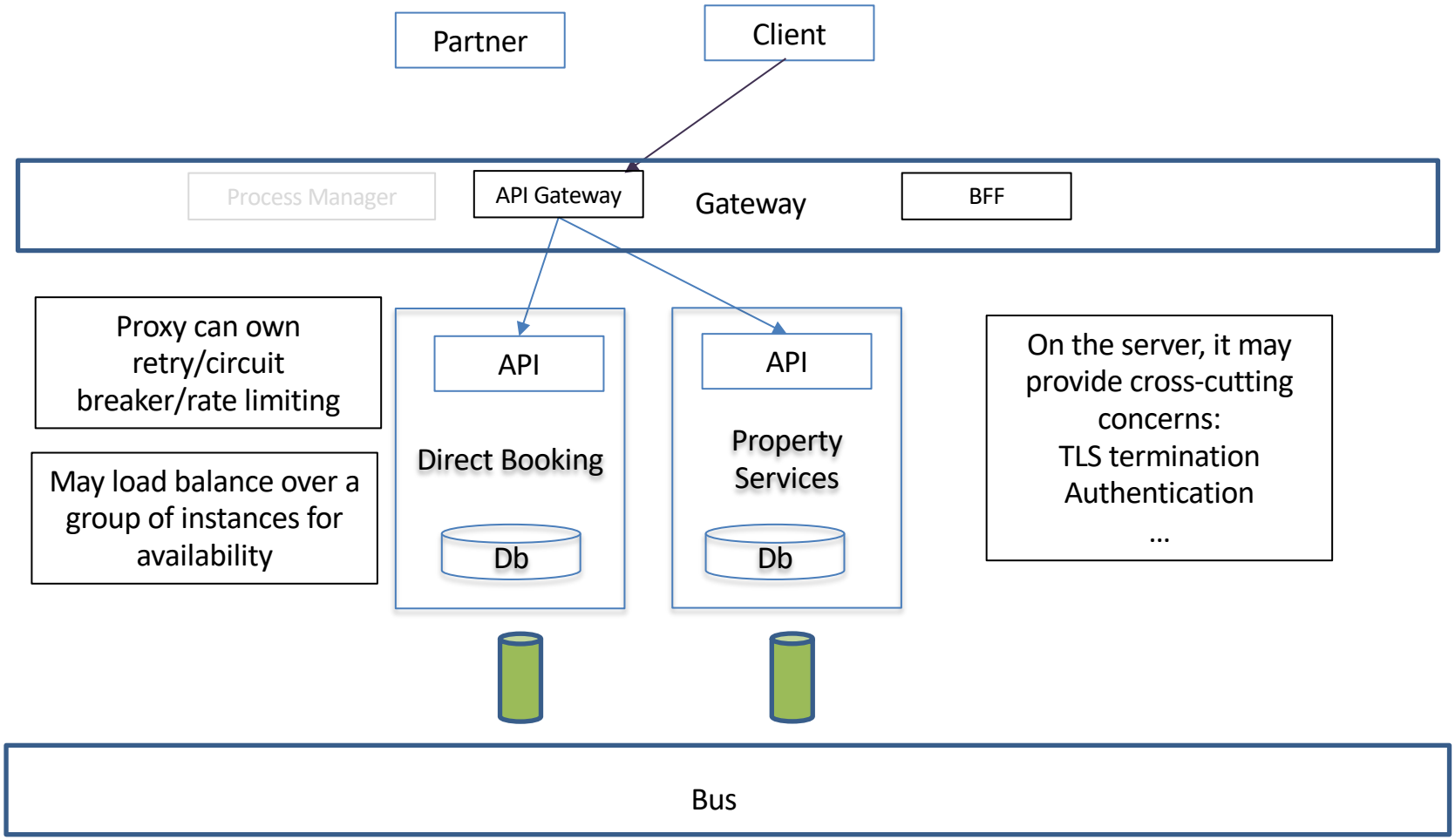
Group Room Search



Composition

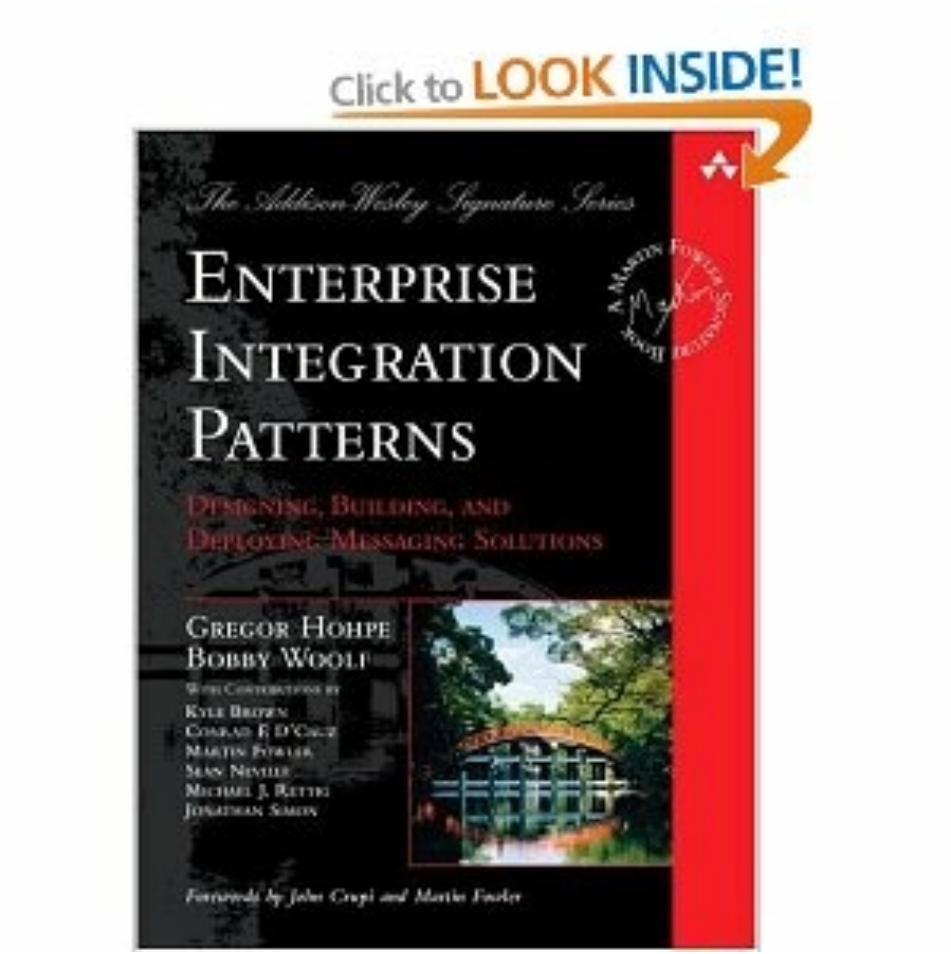


Composition



8. NEXT STEPS

Further Reading



Q&A