

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

- Messaging and Events
- Order
- Guaranteed Delivery
- Event Driven Collaboration
- Versioning
- Documentation
- Observability
- CAP Theorem
- Consumers
- Next Steps

Housekeeping

- Some of these slides are dense
 - Part of the goal is to give you a takeaway that you can refer to when you need the detail after this course
 - But it can make some hard to follow
 - Worse in person than remote
- Download the PDF or Slides
 - You can refer to that on your laptop for the detail if it helps you
 - <https://github.com/iancooper/Presentations>
 - A PDF/Slides for each day

Day Two

MESSAGING AND EVENTS

Message Types

Messaging

Has Intent

Request An Answer
(Query)
Transfer of Control
(Command)
Transfer of Value

Part of a Workflow
Part of a Conversation

Concerned with
the Future

Eventing

Provides Facts

Things you Report On

No Expectations

History

Context

Concerned
with the Past

After Clemens Vasters <https://youtu.be/lTrlLErsqzY>

Message Types

Messaging

Command

Request-Reply

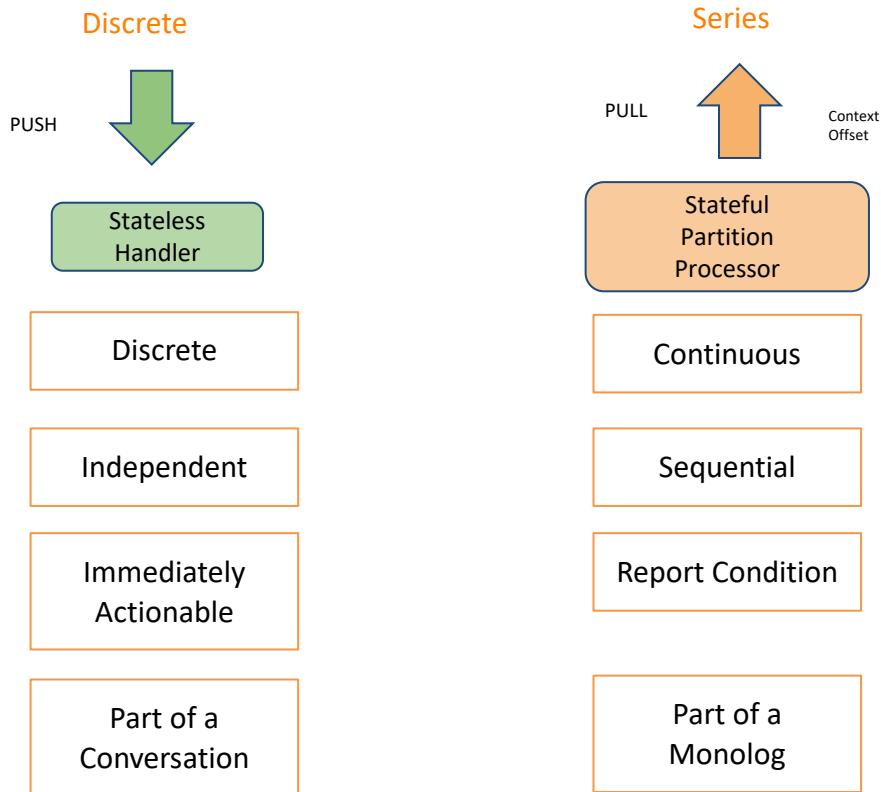
Eventing

Event (Notification)

Document

See Gregor Hohpe: <https://www.enterpriseintegrationpatterns.com/patterns/messaging/Message.html>

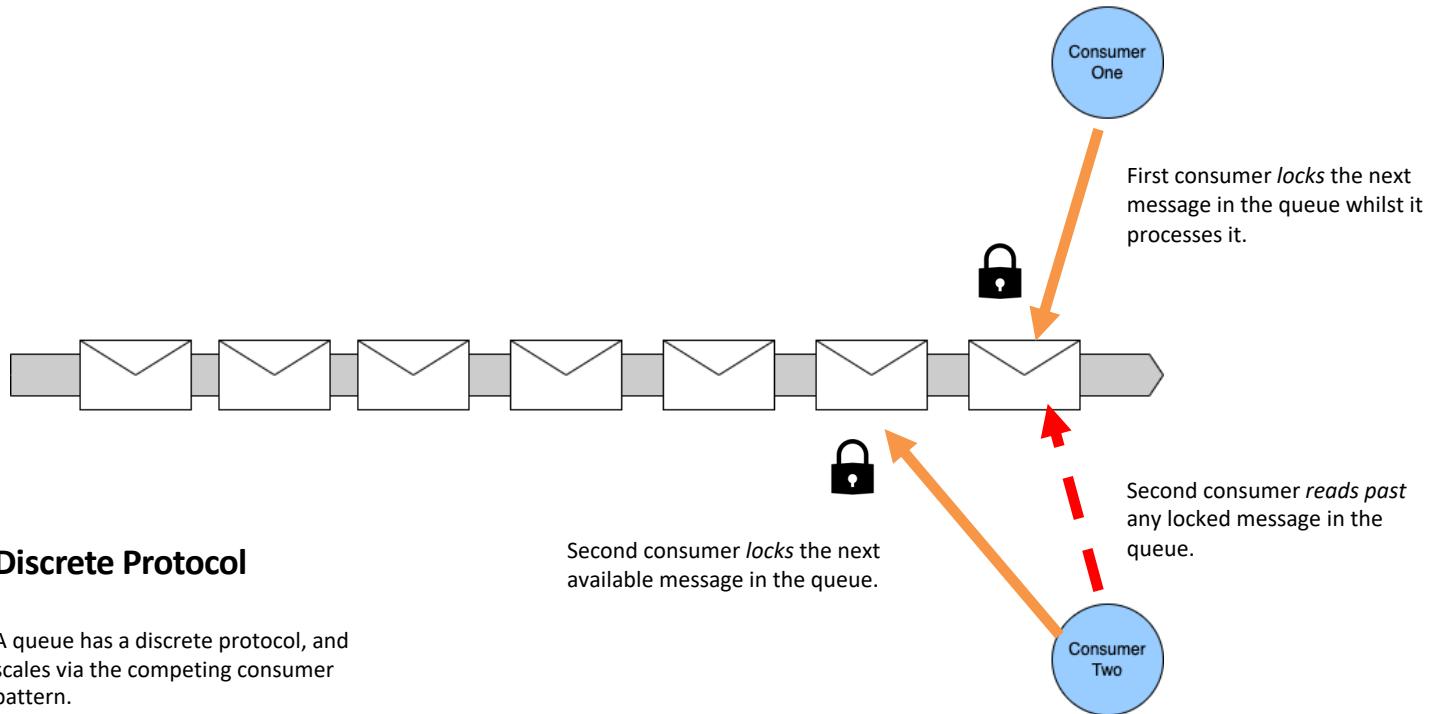
Event Types



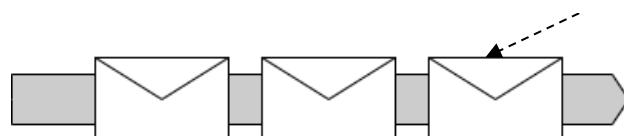
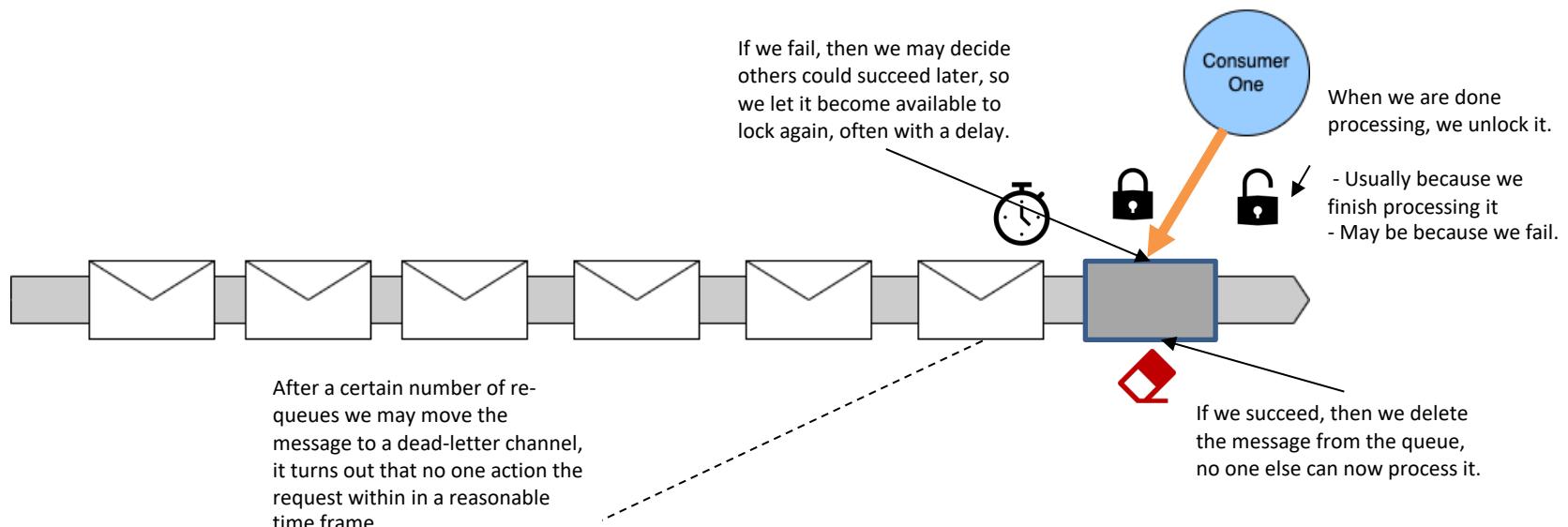
After Clemens Vasters: <https://skillsmatter.com/skillscasts/10191-keynote-events-data-points-jobs-and-commands-the-rise-of-messaging>

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

Queues



Queues

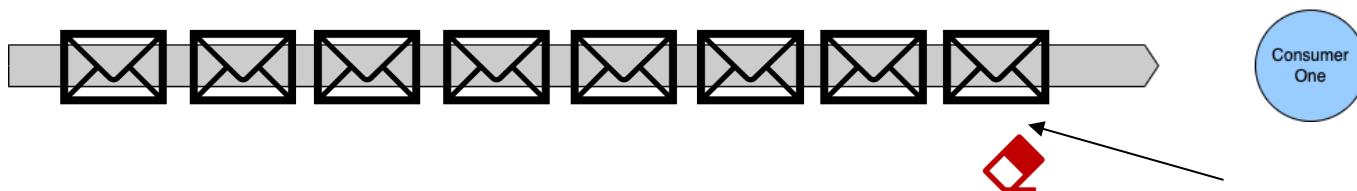


Messaging as Tasks

With messaging we can think of our messages as tasks - they are a request for us to carry out an action. Once the action is done, we can delete the task.

- We don't anyone else to action it, it's already been done.
- Someone receiving a done task will have to discard it.
- If we can't action it, someone else will need to action it.
- A reply is often a discrete event.

Queues



No Archive and Replay

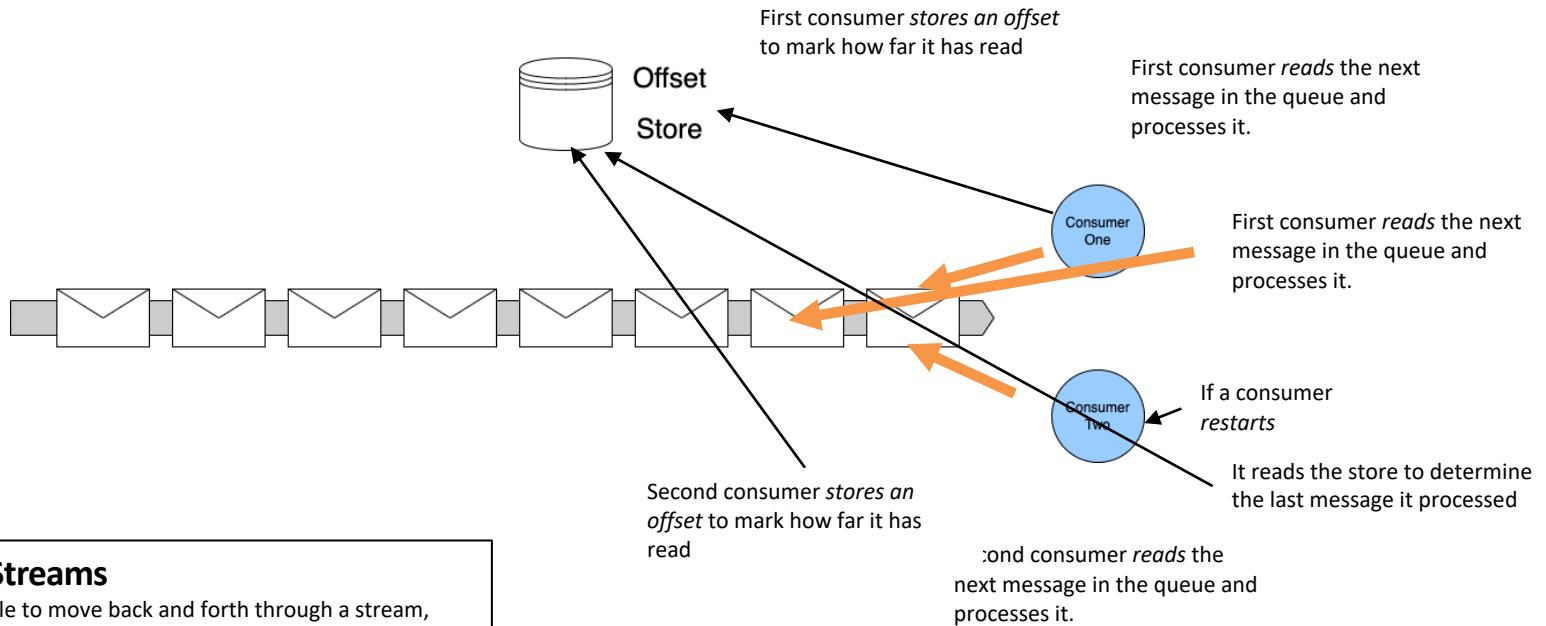
With queues we delete a message once we have completed the associated action. That means we have no way to replay the request for work. Our only option is to ask the sender to resend their request.

This works for tasks or replies.

This is not how series events work.

If we succeed, then we delete the message from the queue, no one else can now process it.

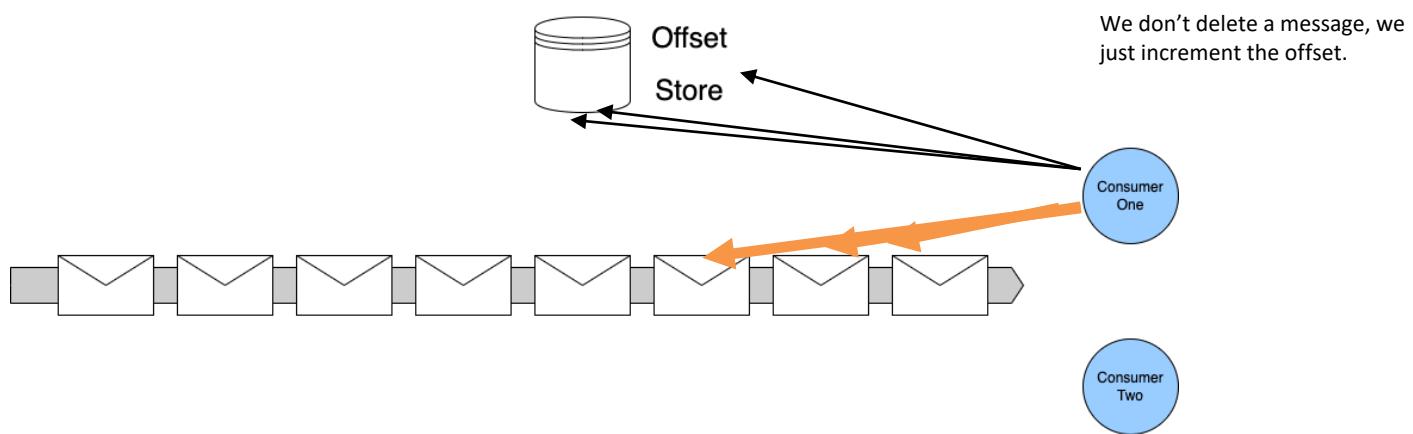
Streams



Series Events as Streams

Because we want to be able to move back and forth through a stream, series messaging is best supported by a stream.

Streams



Archive and Replay

Archive and Replay is straightforward as nothing is deleted. We simply reset the consumer's offset to re-read the stream

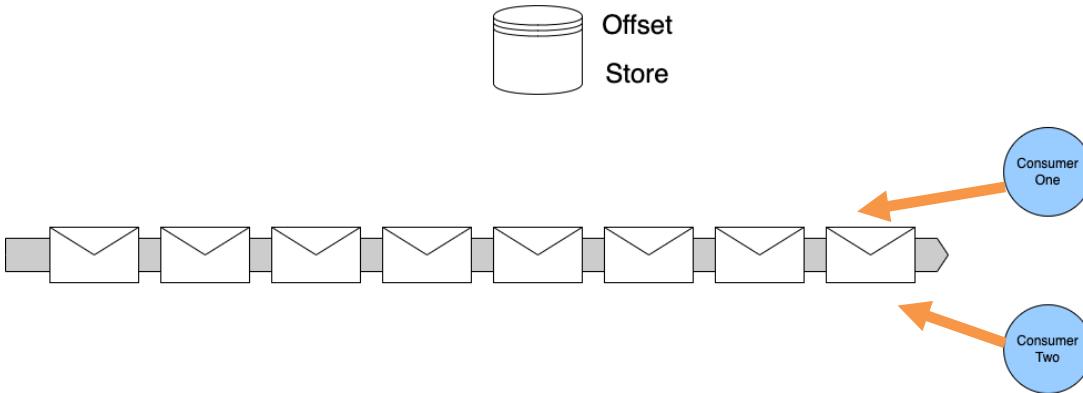
No Requeue or DLQ

Because we do not lock items, we do not requeue items, including requeue with delay. Your strategy is:

- Ignore and Continue
- Retry
- Copy to another stream (a delay or DLQ stream)

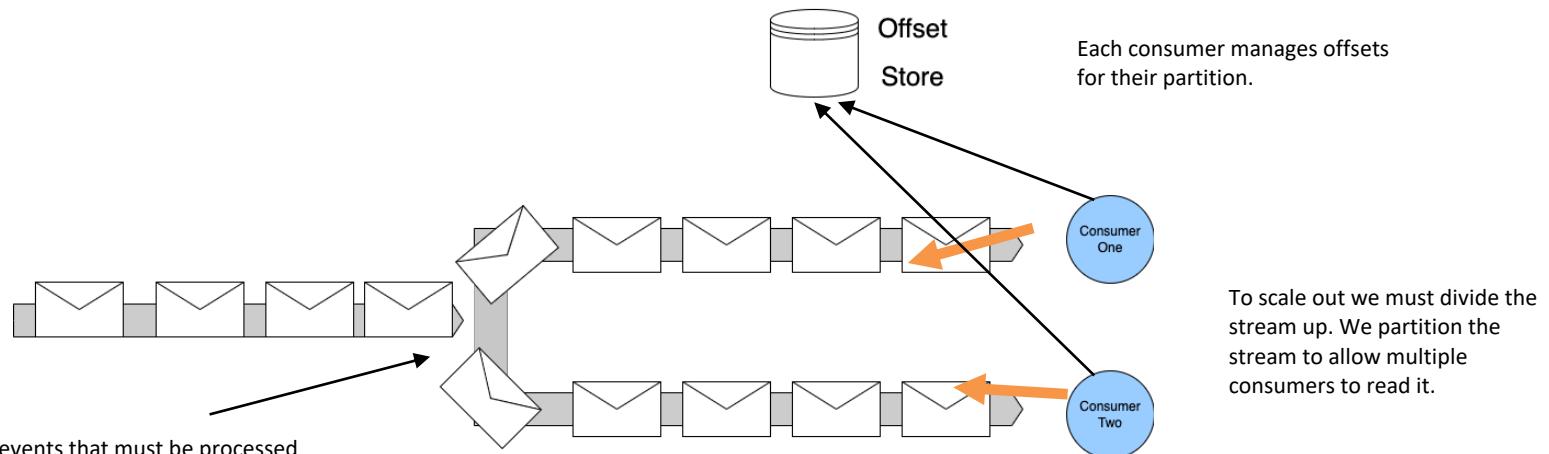
Look for possibility that you have task or reply and want a queue

Streams

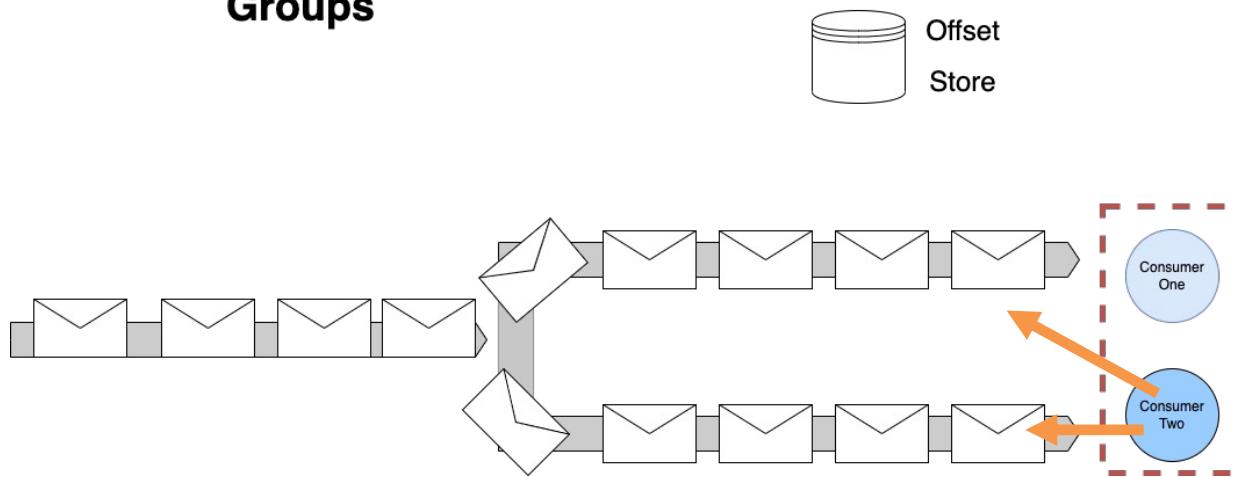


If each consumer *reads* the next message in the queue how do we *scale out*, when the rate of arrival on the queue exceeds our ability to process them?

Partitions



Consumer Groups

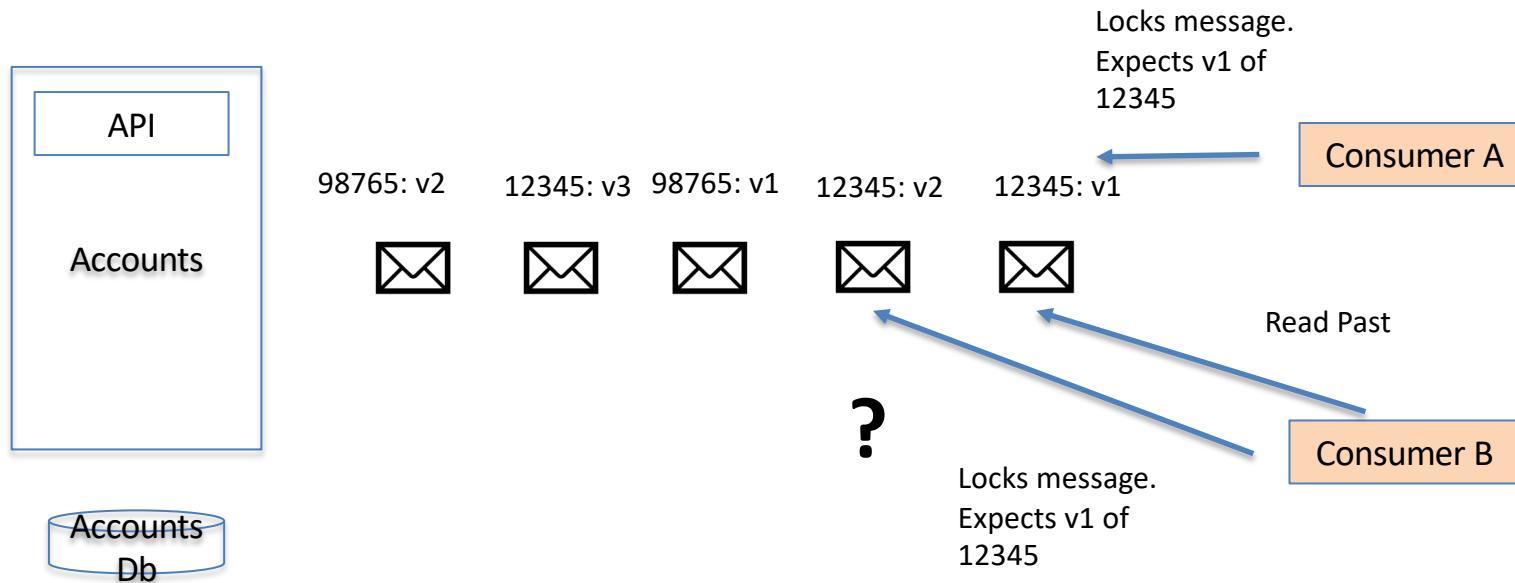


To provide availability – only one consumer in a group can read from a partition at a time – but a consumer in a group may read from more than one of the partitions owned by that group

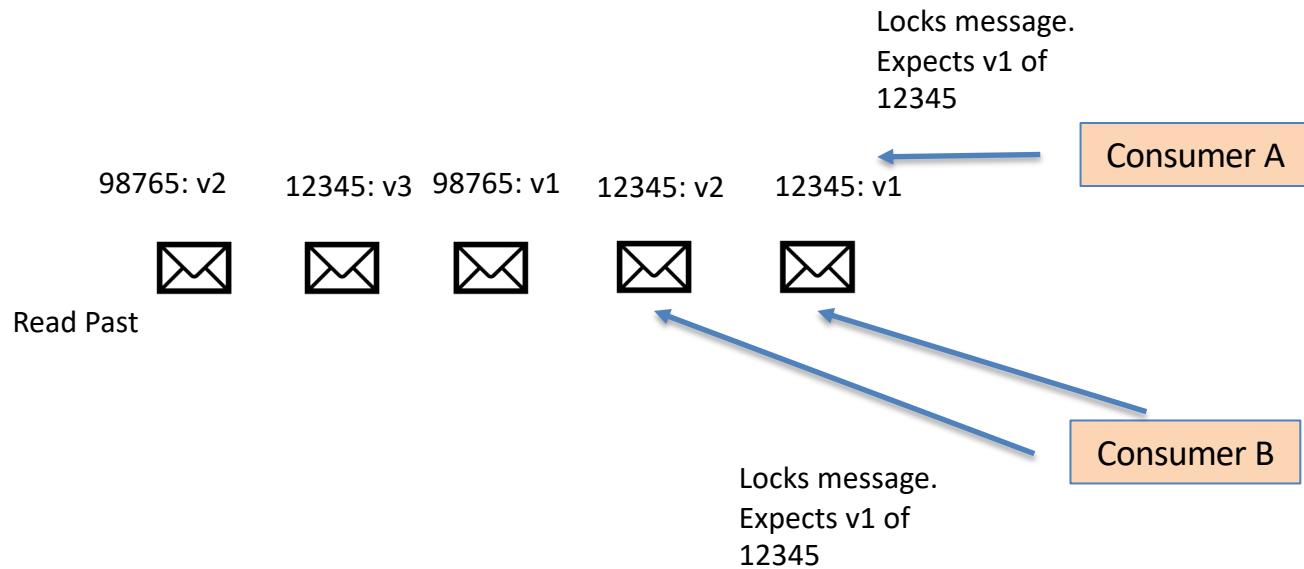
In Order Delivery

2 ORDER

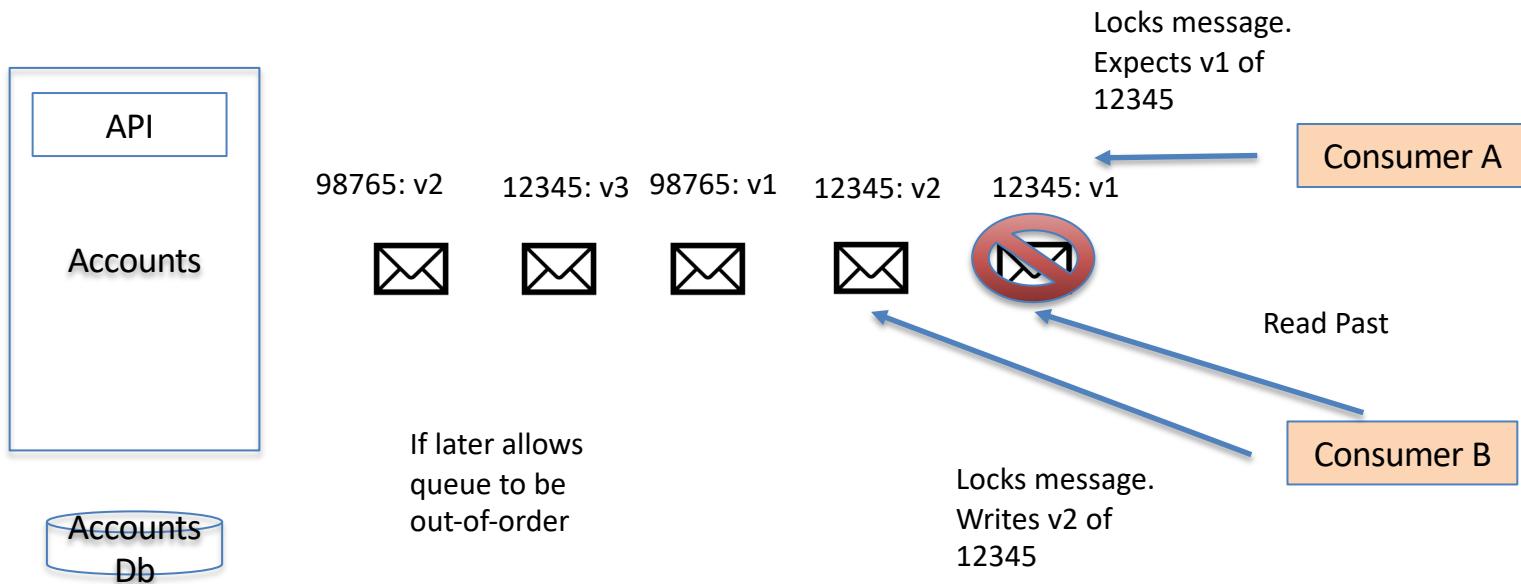
Order and Competing Consumers



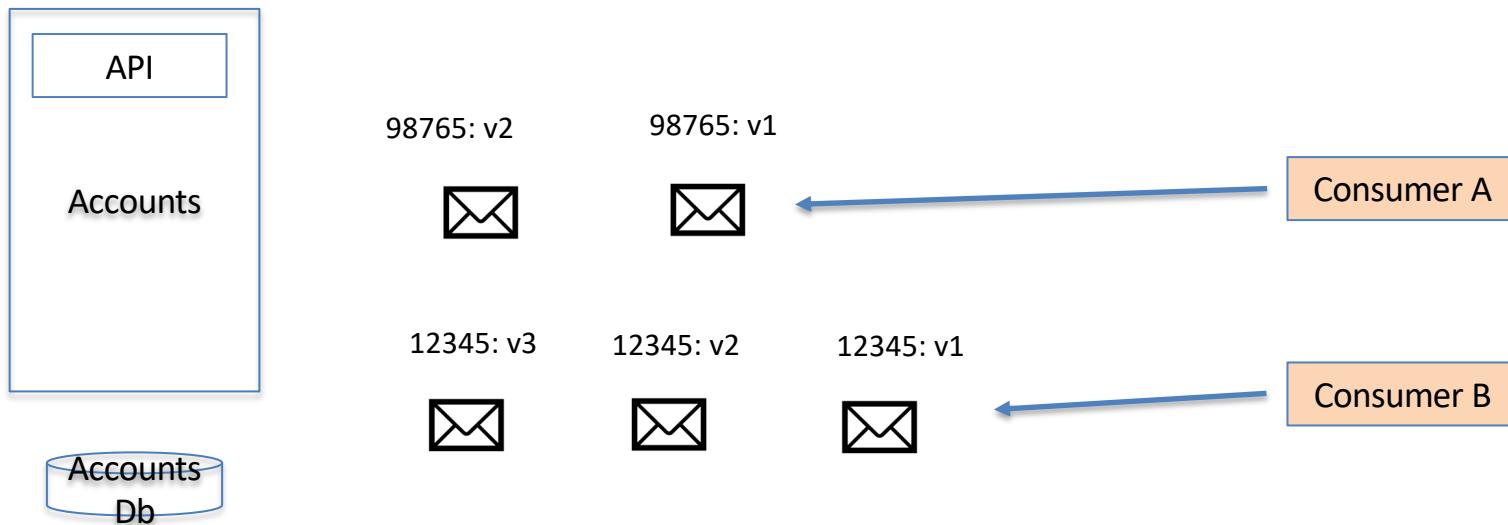
Requeue?



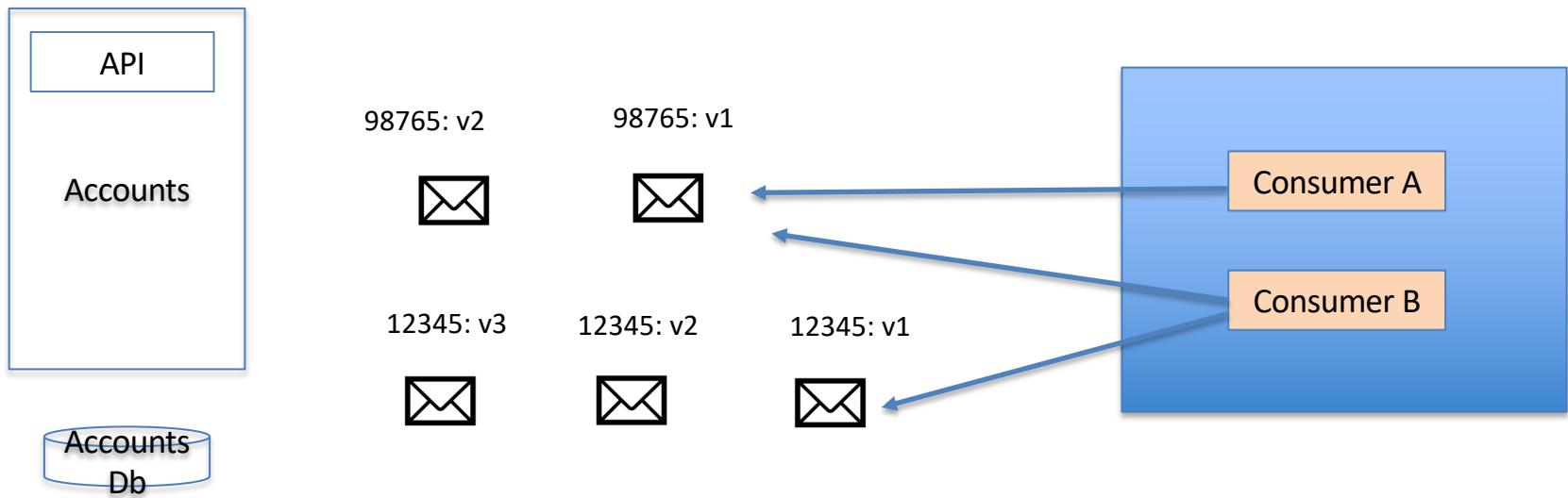
If Later



Consistent Hashing

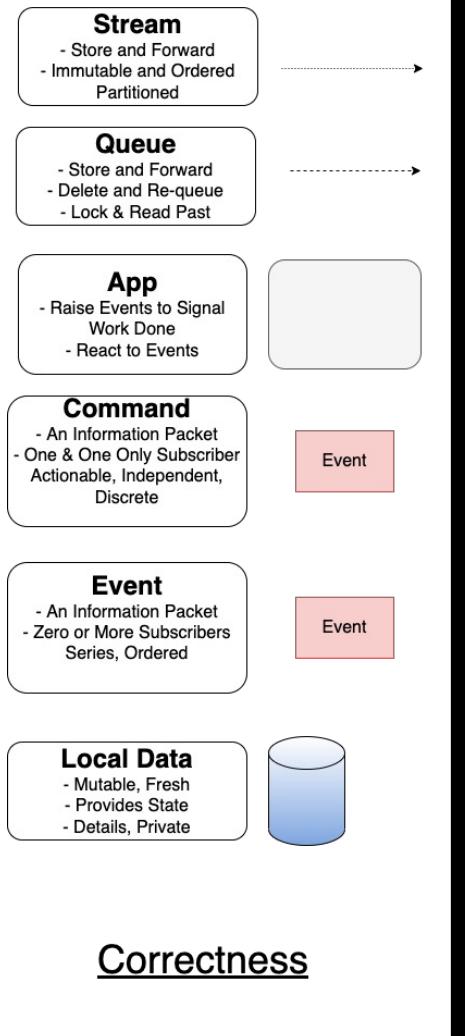


Consistent Hashing w. Consumer Groups

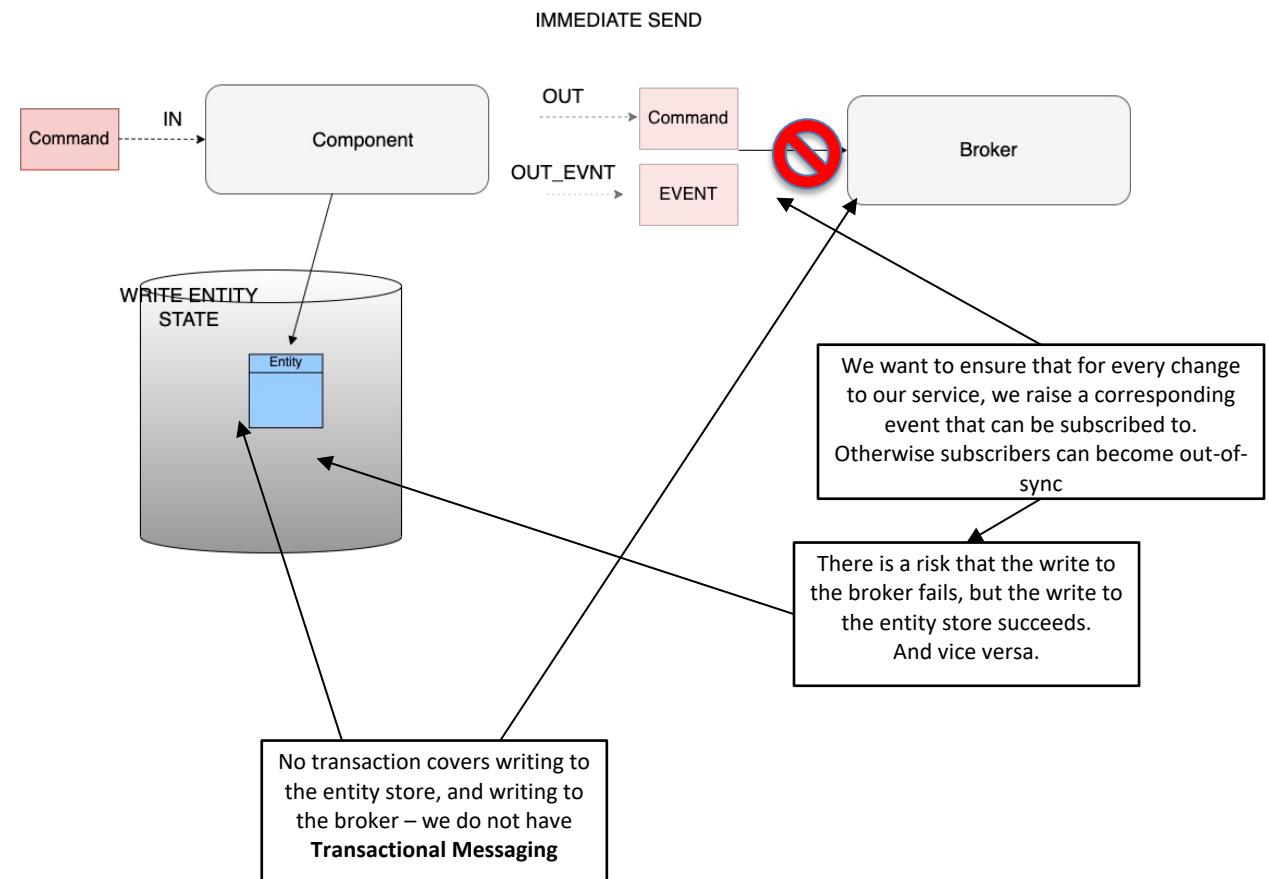


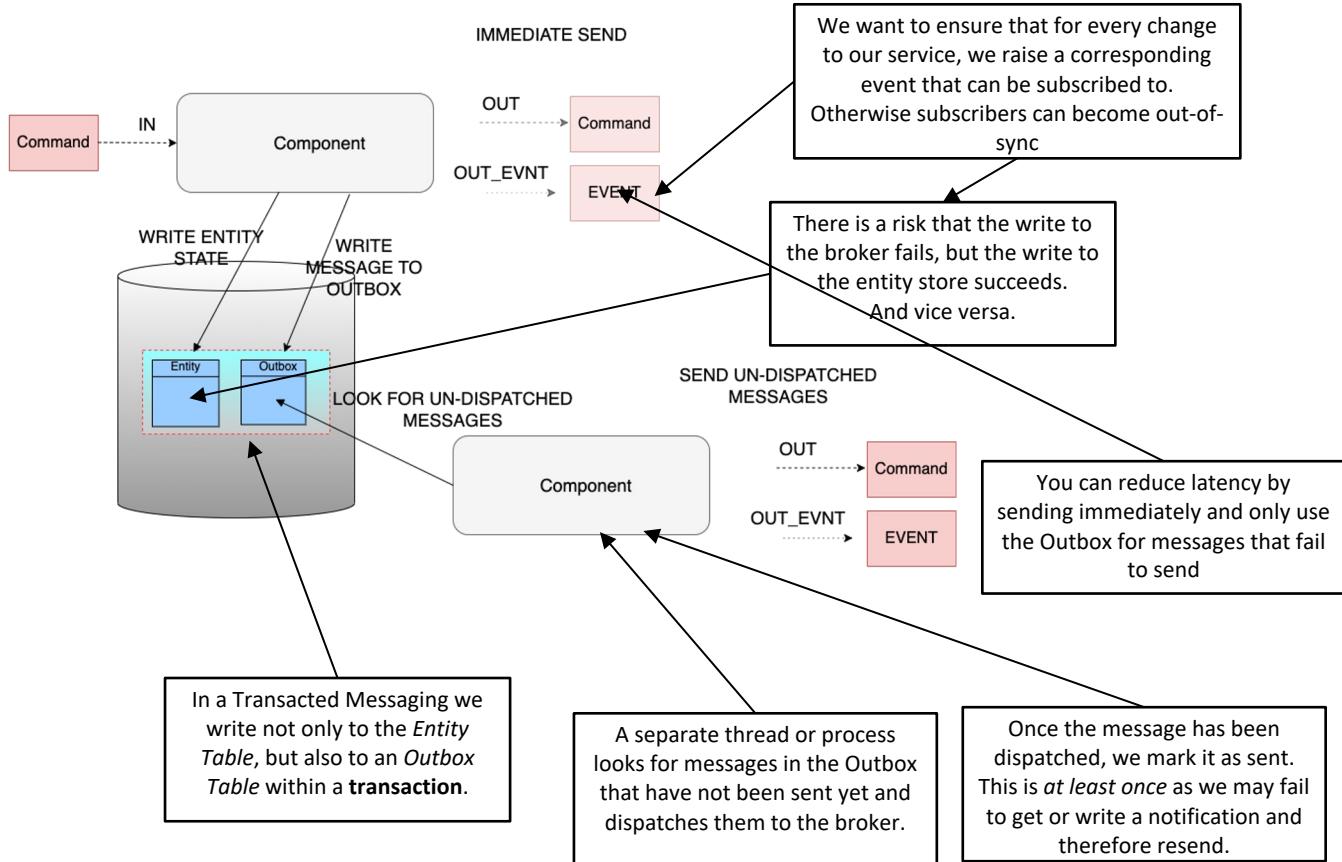
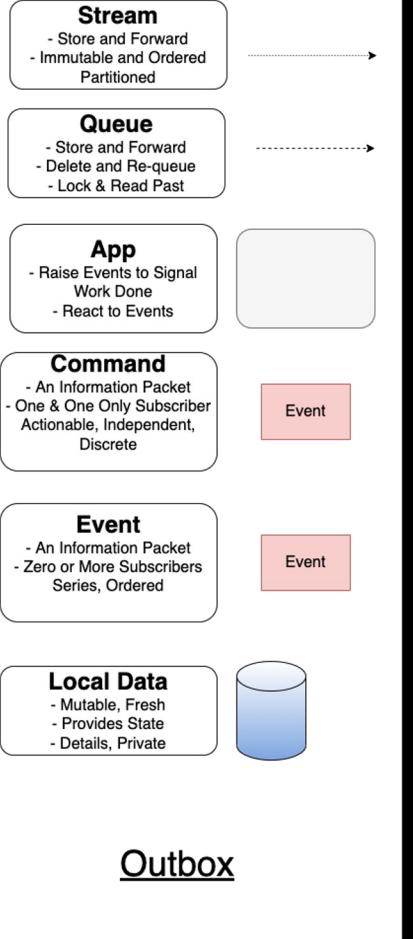
Transactional Messaging

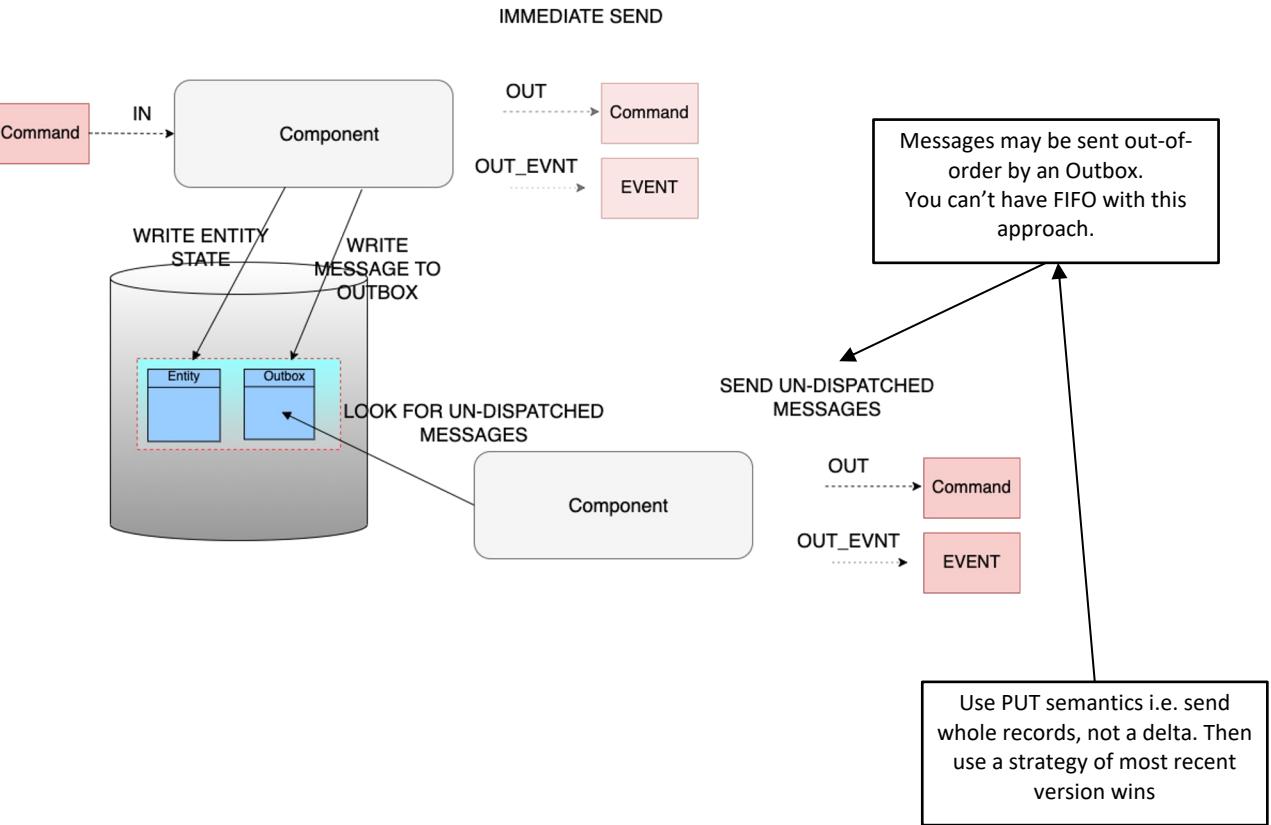
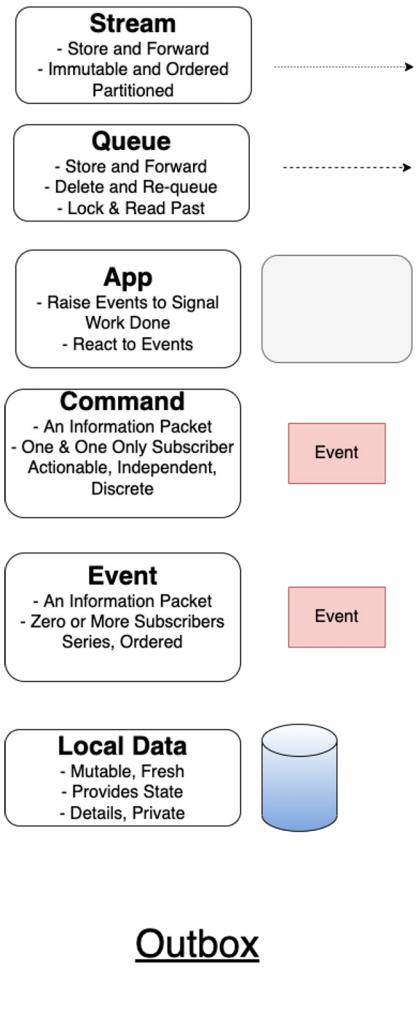
3 GUARANTEED DELIVERY

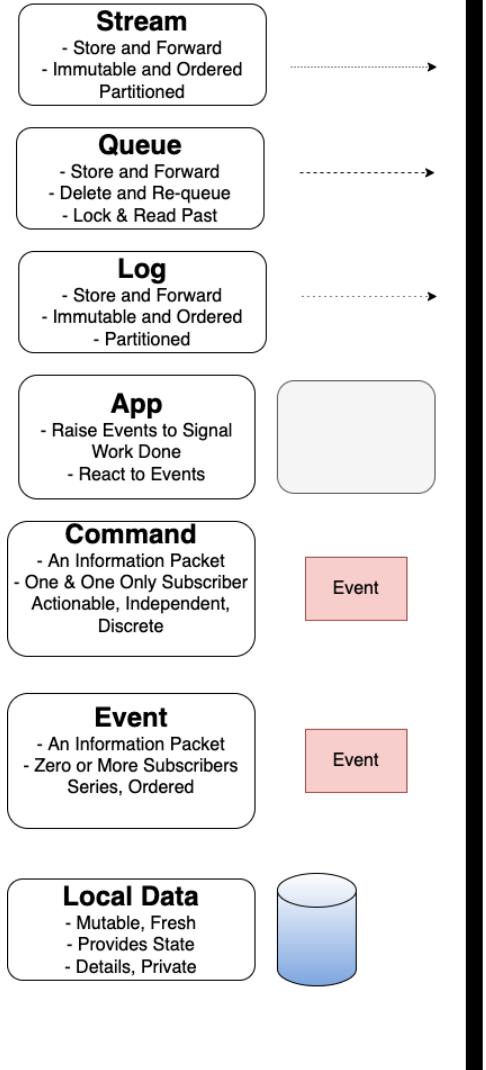


Correctness

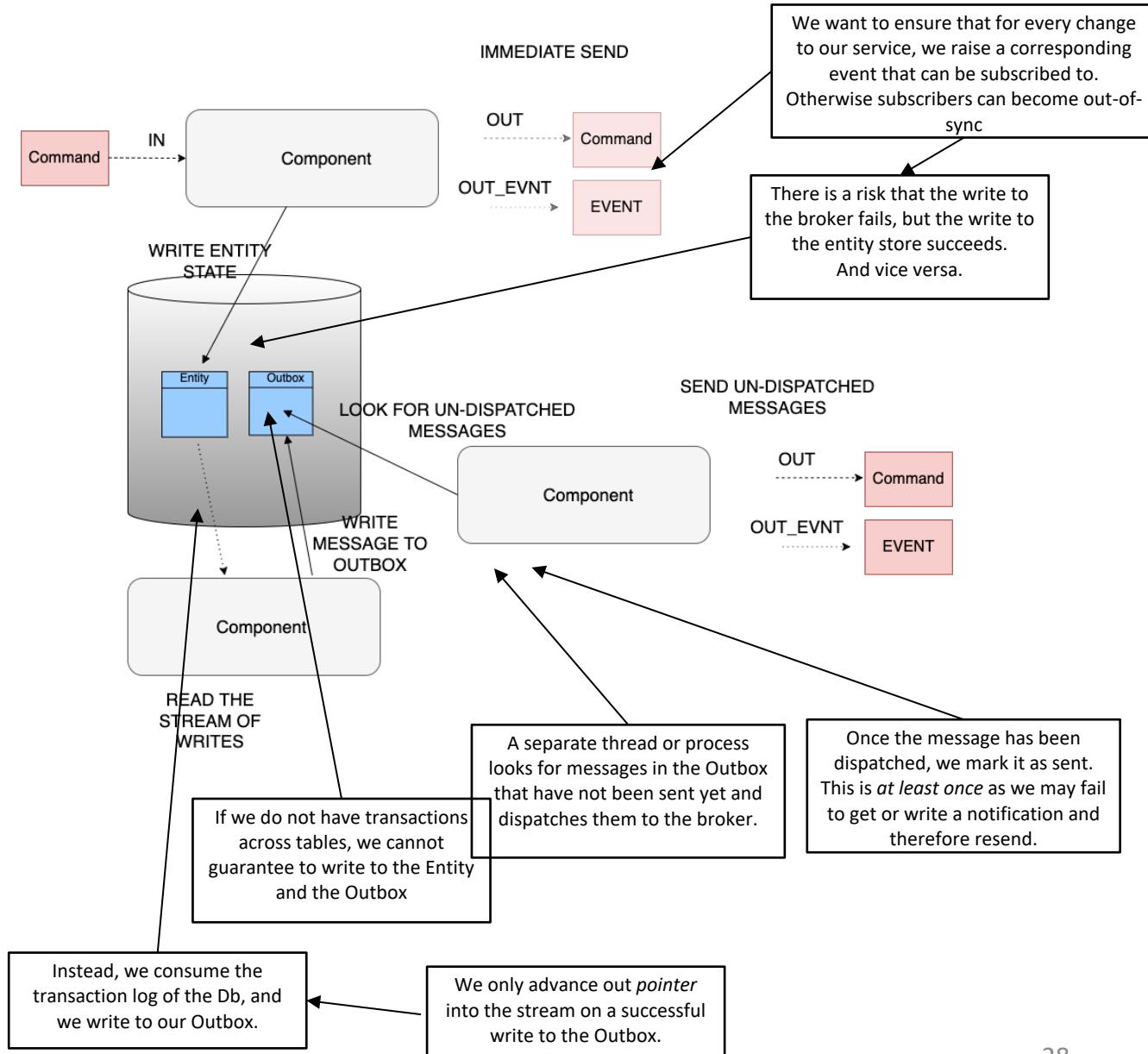


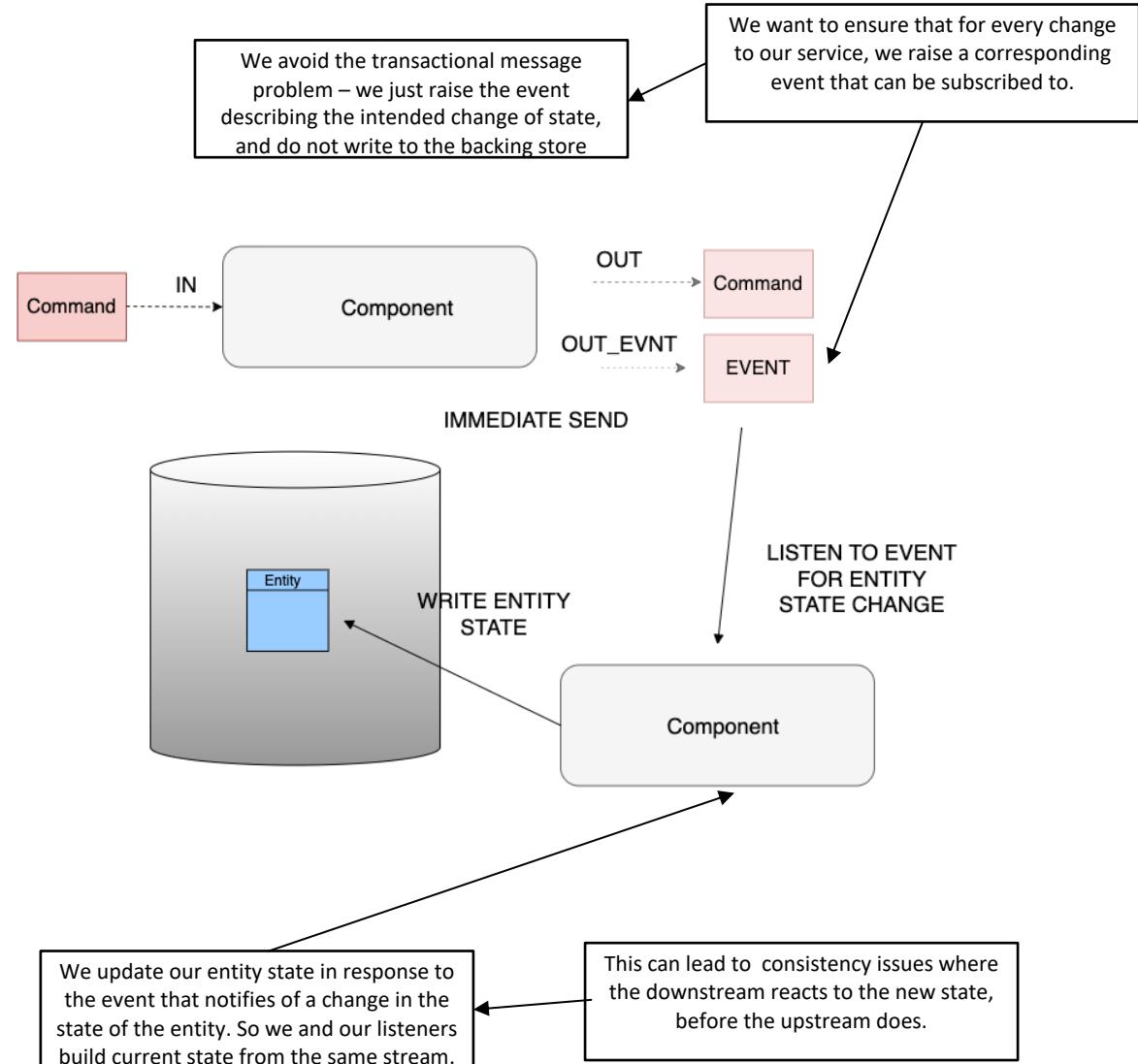




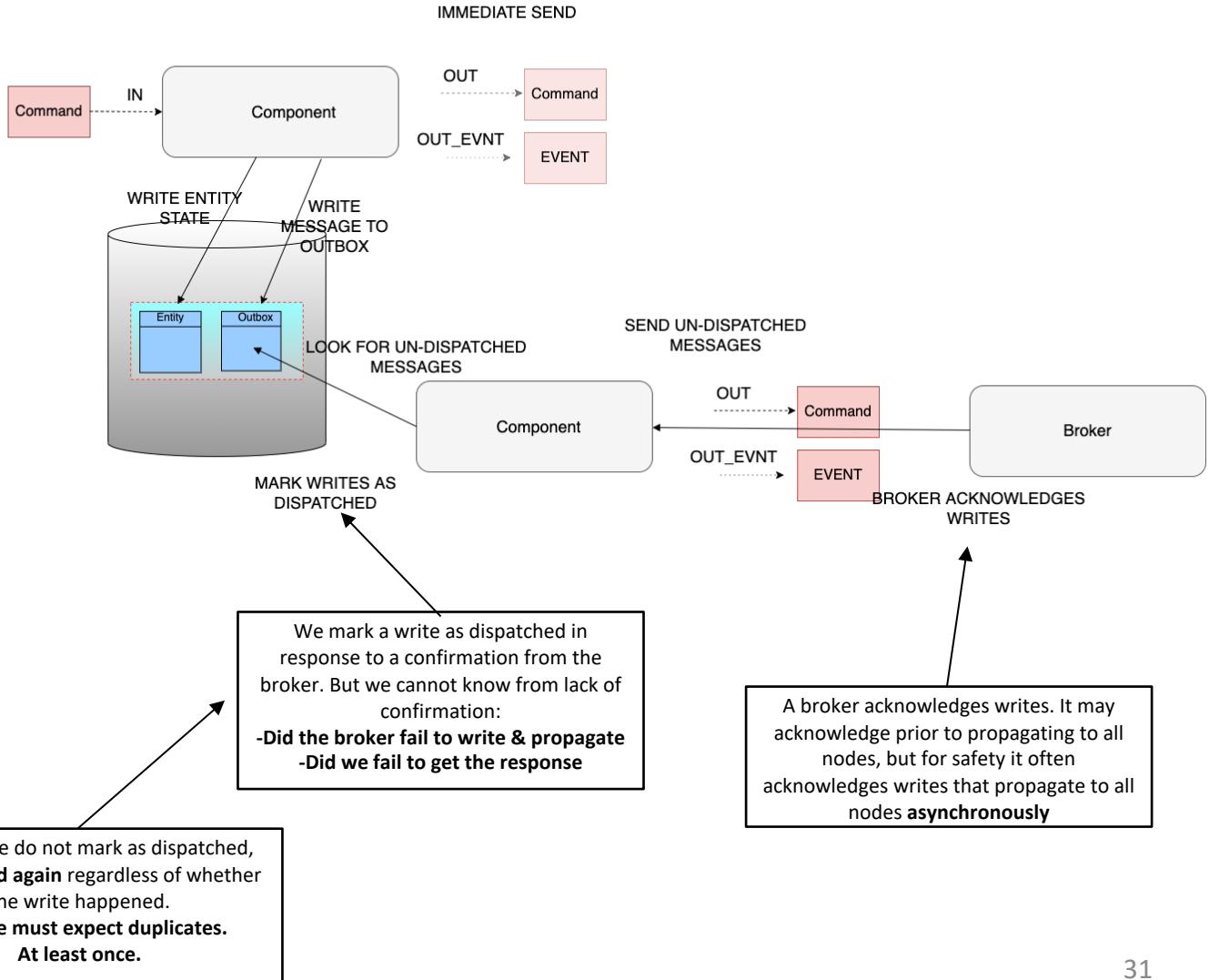
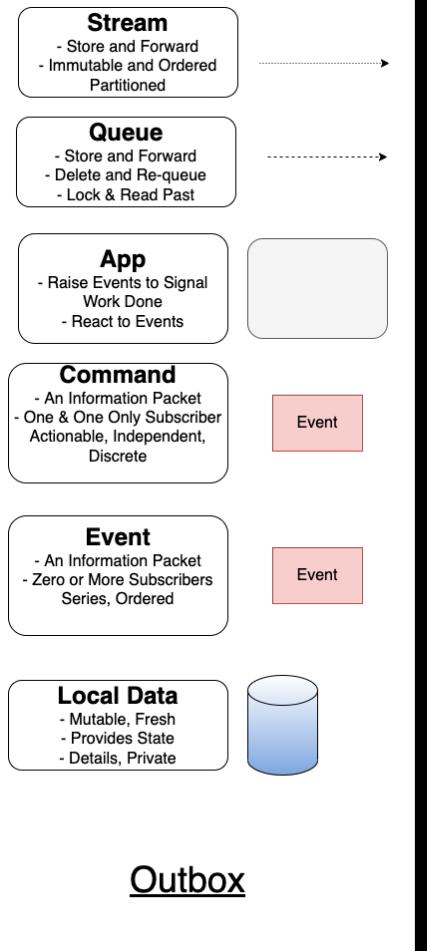


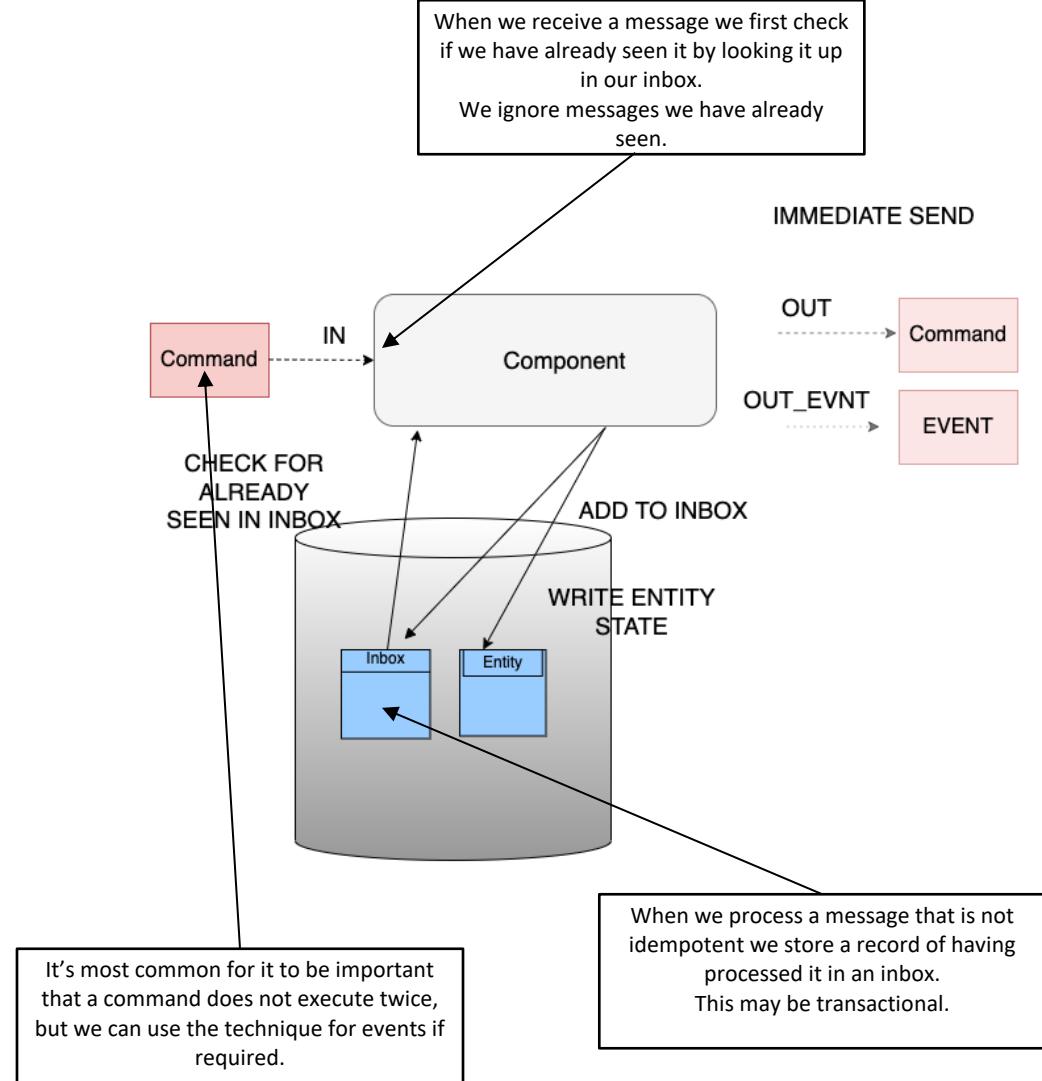
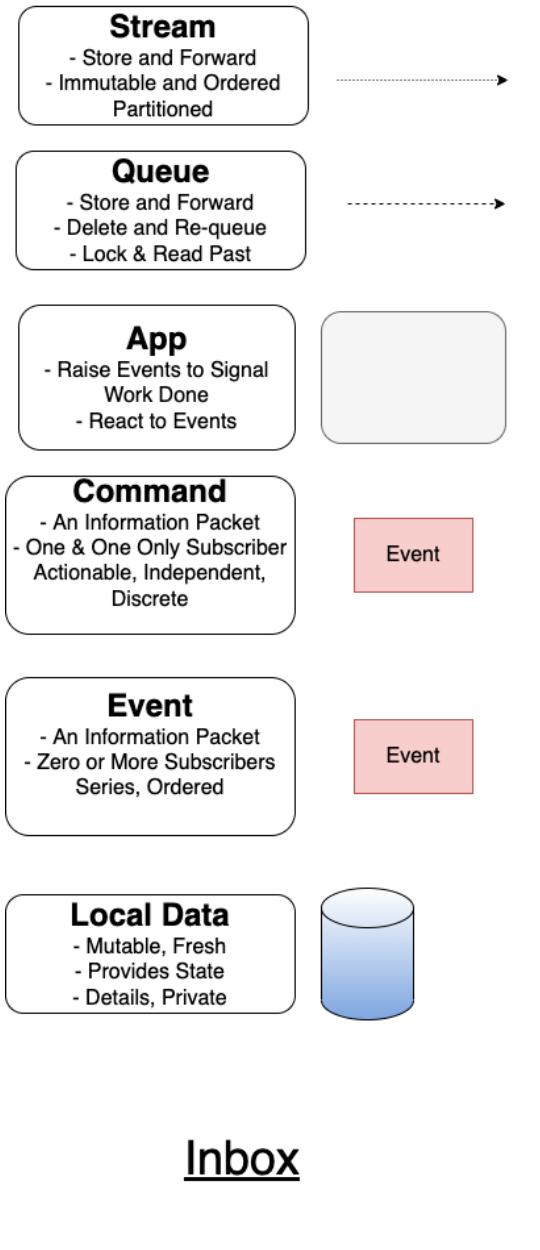
Log Tailing

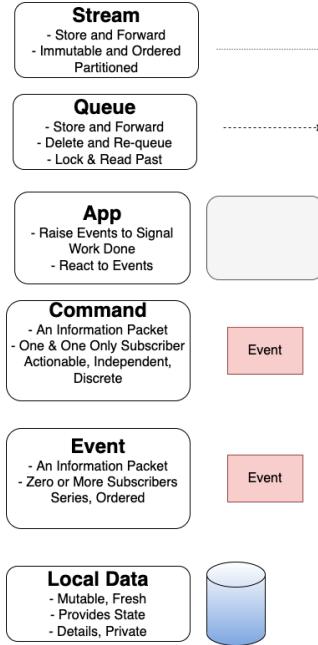




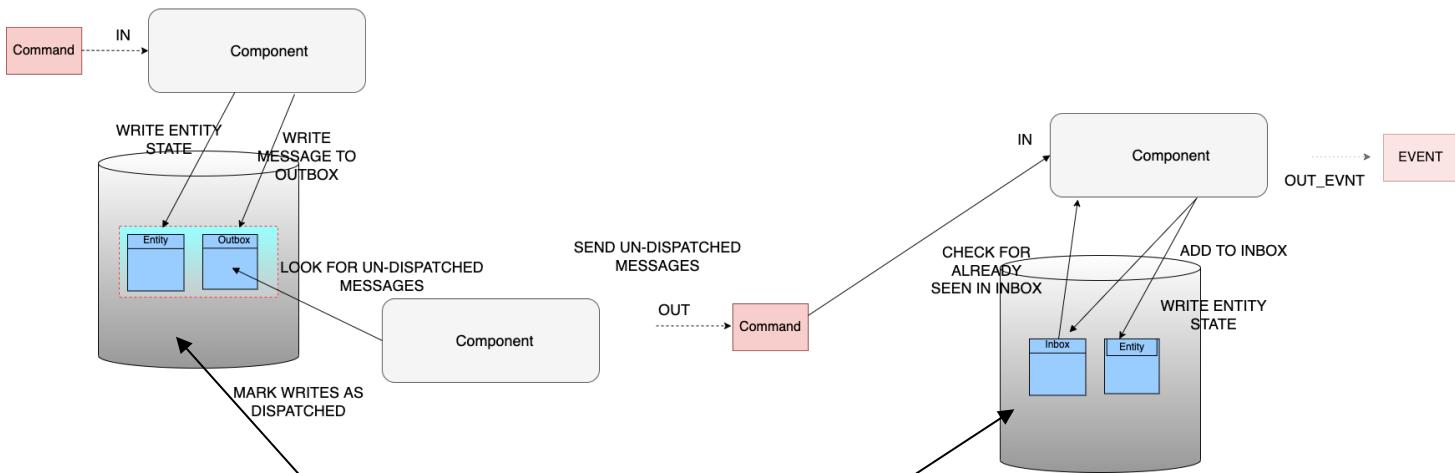
At Least Once







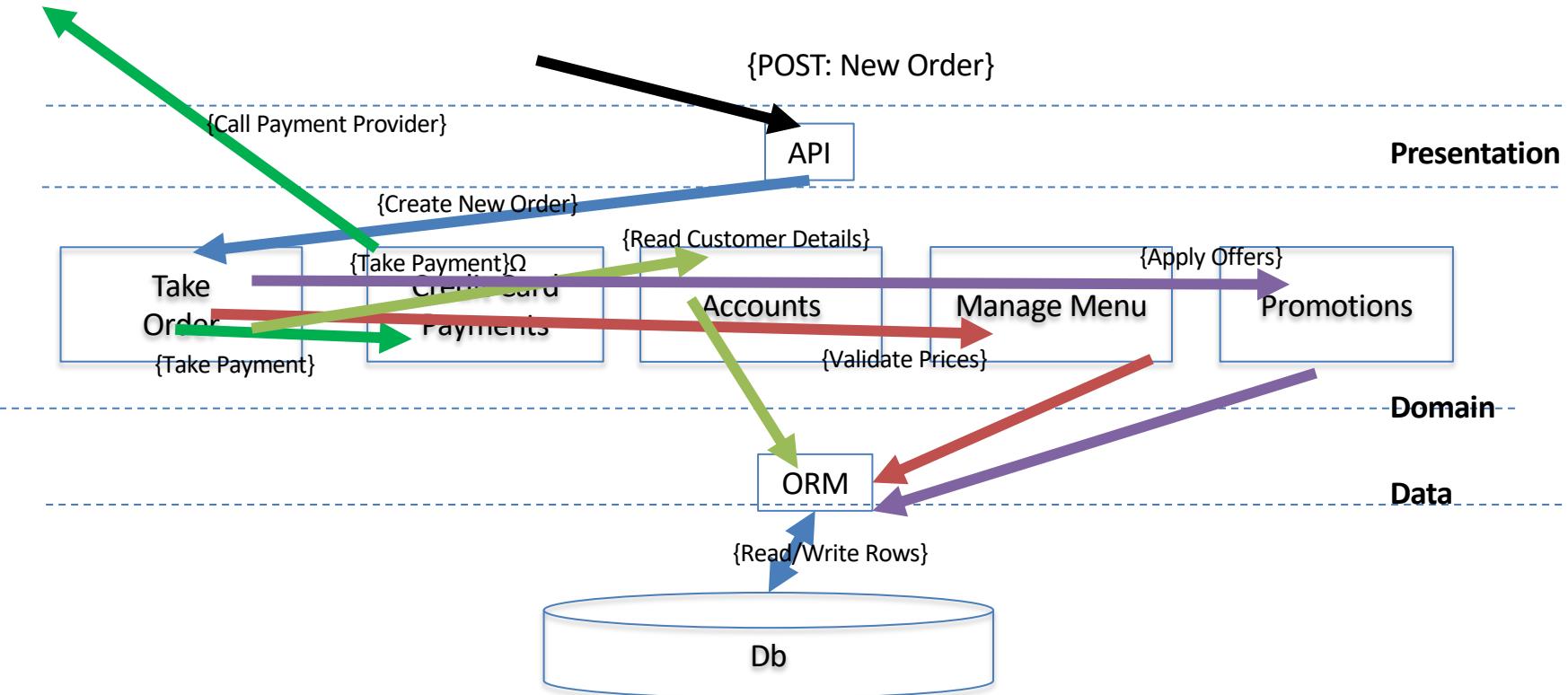
Outbox and Inbox



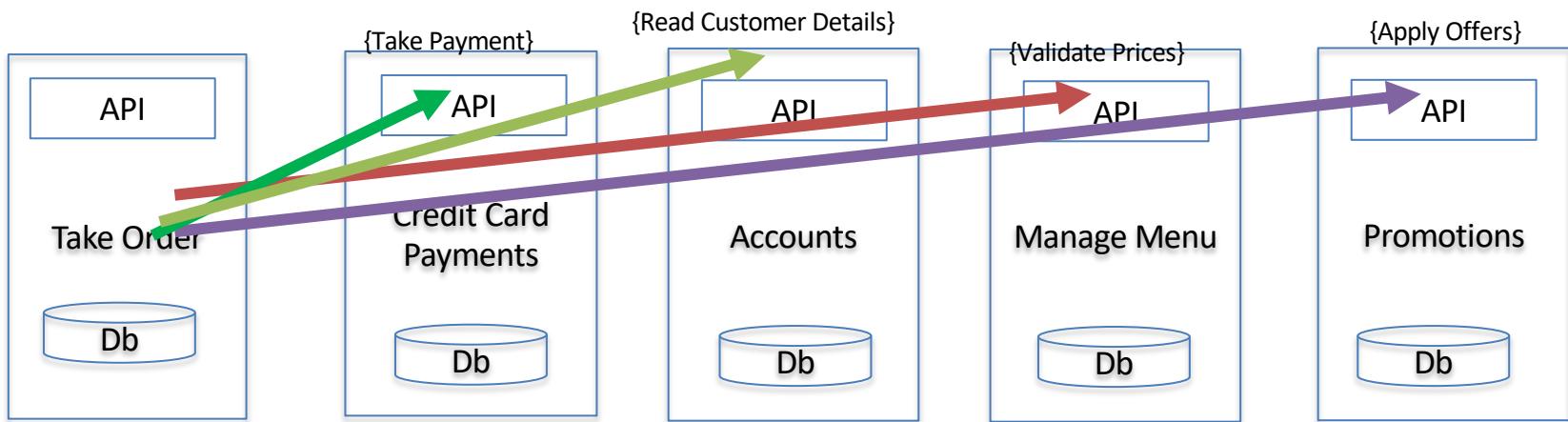
The Outbox and Inbox work together to create **guaranteed once-only delivery**.

4 EVENT DRIVEN COLLABORATION

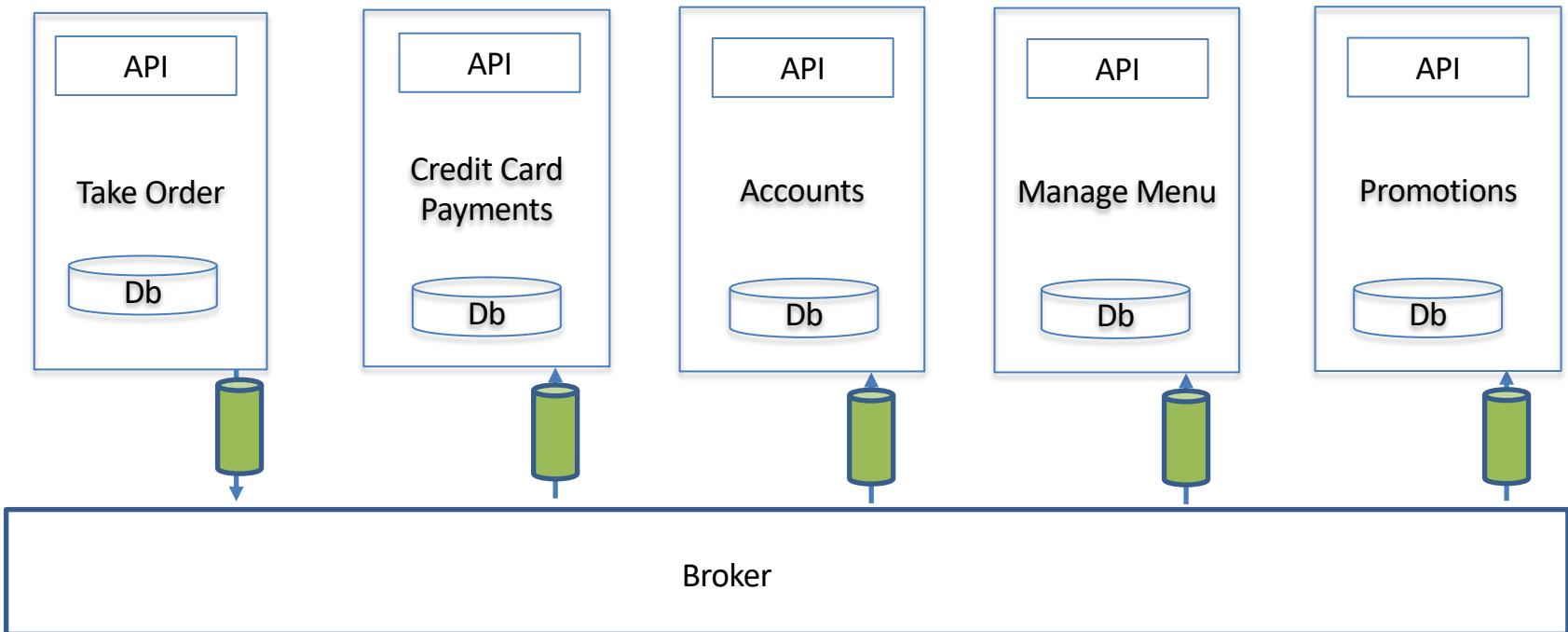
Getting work done in a Monolith



Microservices



Event Driven Architecture



Paper Workflows

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



Messaging/Discrete Event

Discrete, Immediately Actioned

Skinny



Series Event (Document)

Series, Supports Action

Fat





O E D I DIVISION
O E D I ROUTING SHEET

TITLE: EID-E15 (23 July 84)

REQD legend: I = Information
A = Action
R = Reference

Upon Completion, Return to **ETI Barmag**

Destroy
F10

DPS/OE Form No. XXXXXXXX



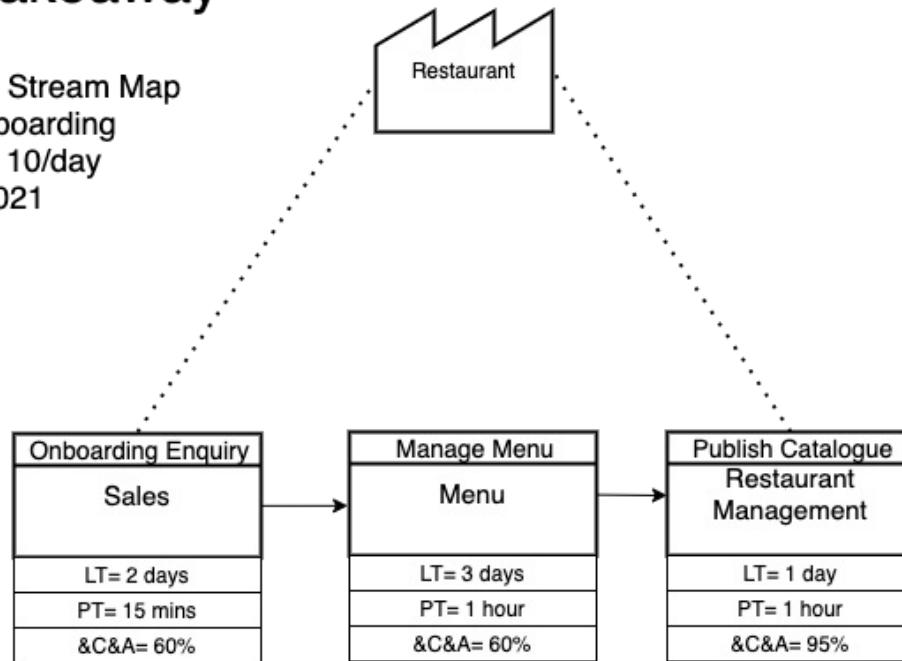
What is a microservice?

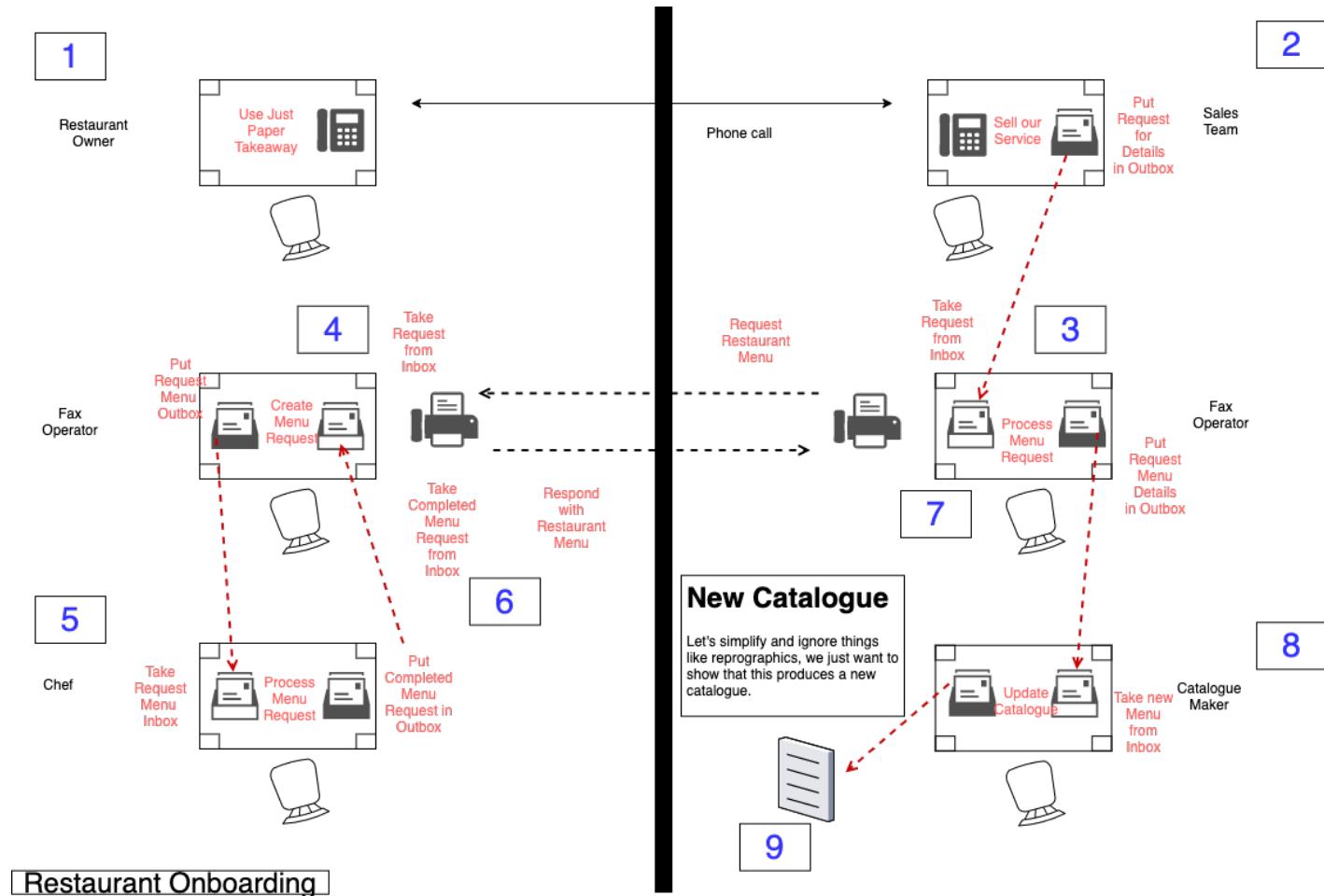
SOA is focused on business *processes*. These *processes* are performed in different steps (also called *activities* or *tasks*) on different systems. The primary goal of a **service** is to represent a “natural” step of business functionality. That is, according to the domain for which it’s provided,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

Just Paper Takeaway

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





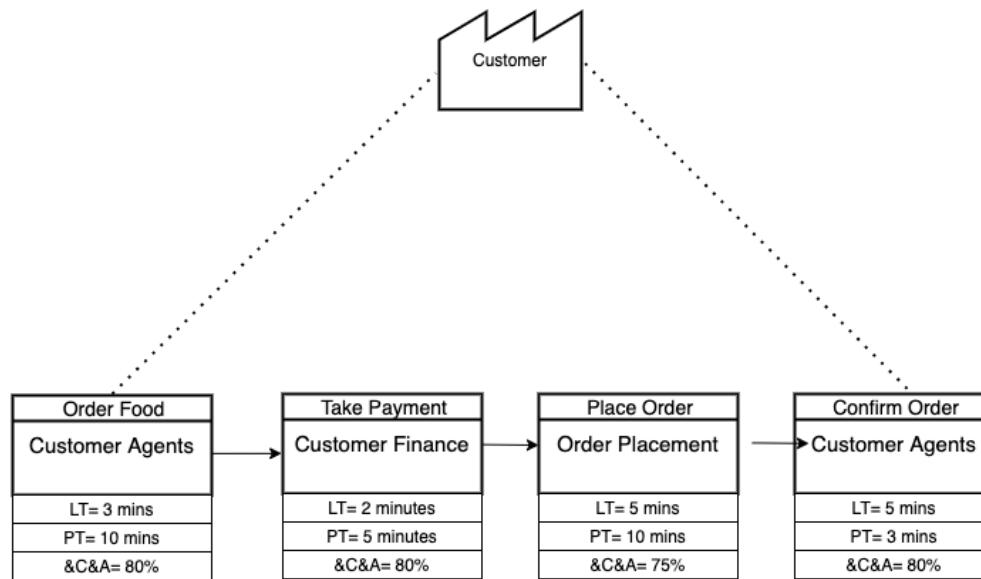
Just Paper Takeaway

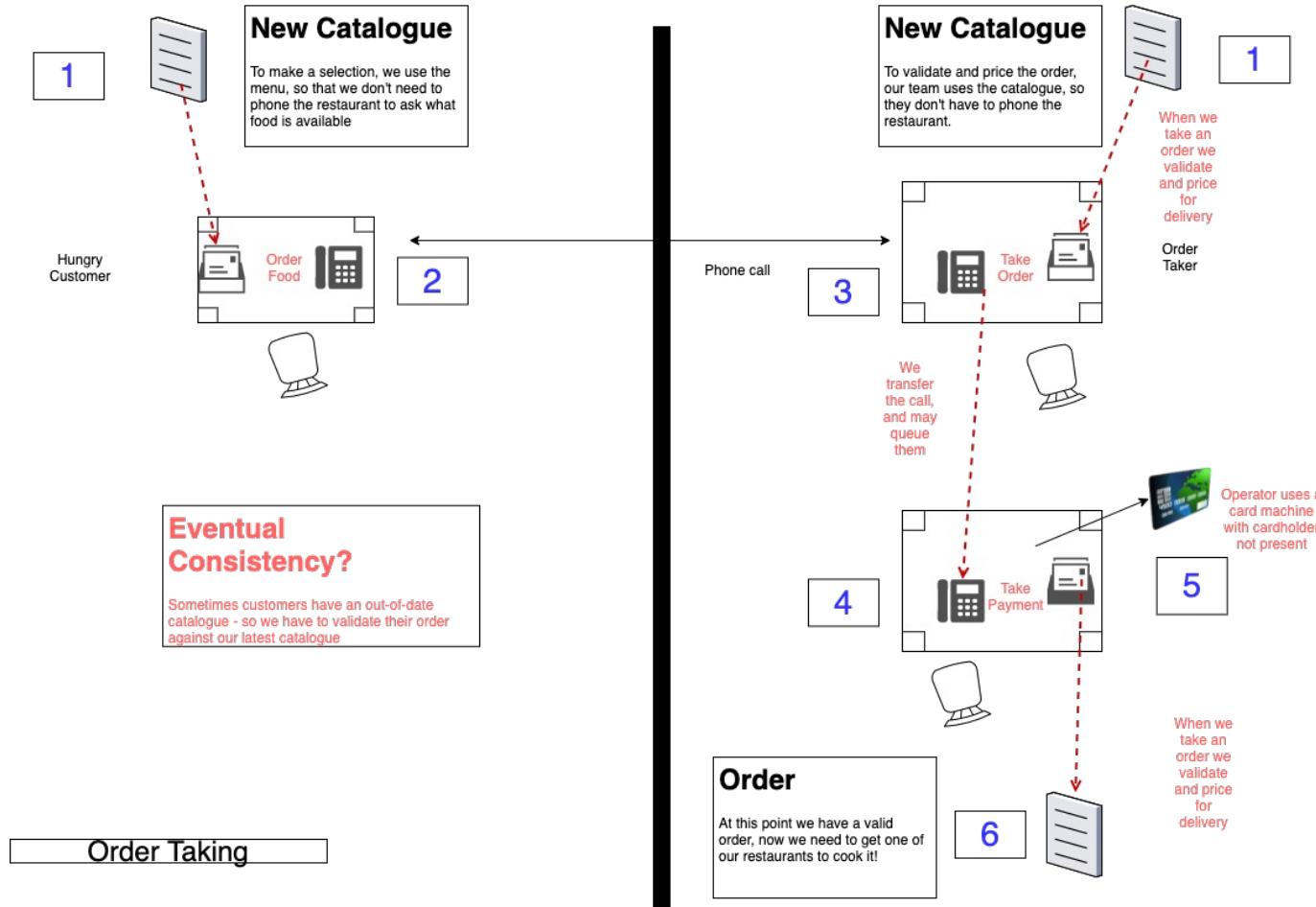
Current State Value Stream Map

Order Flow

Demand Rate 10/day

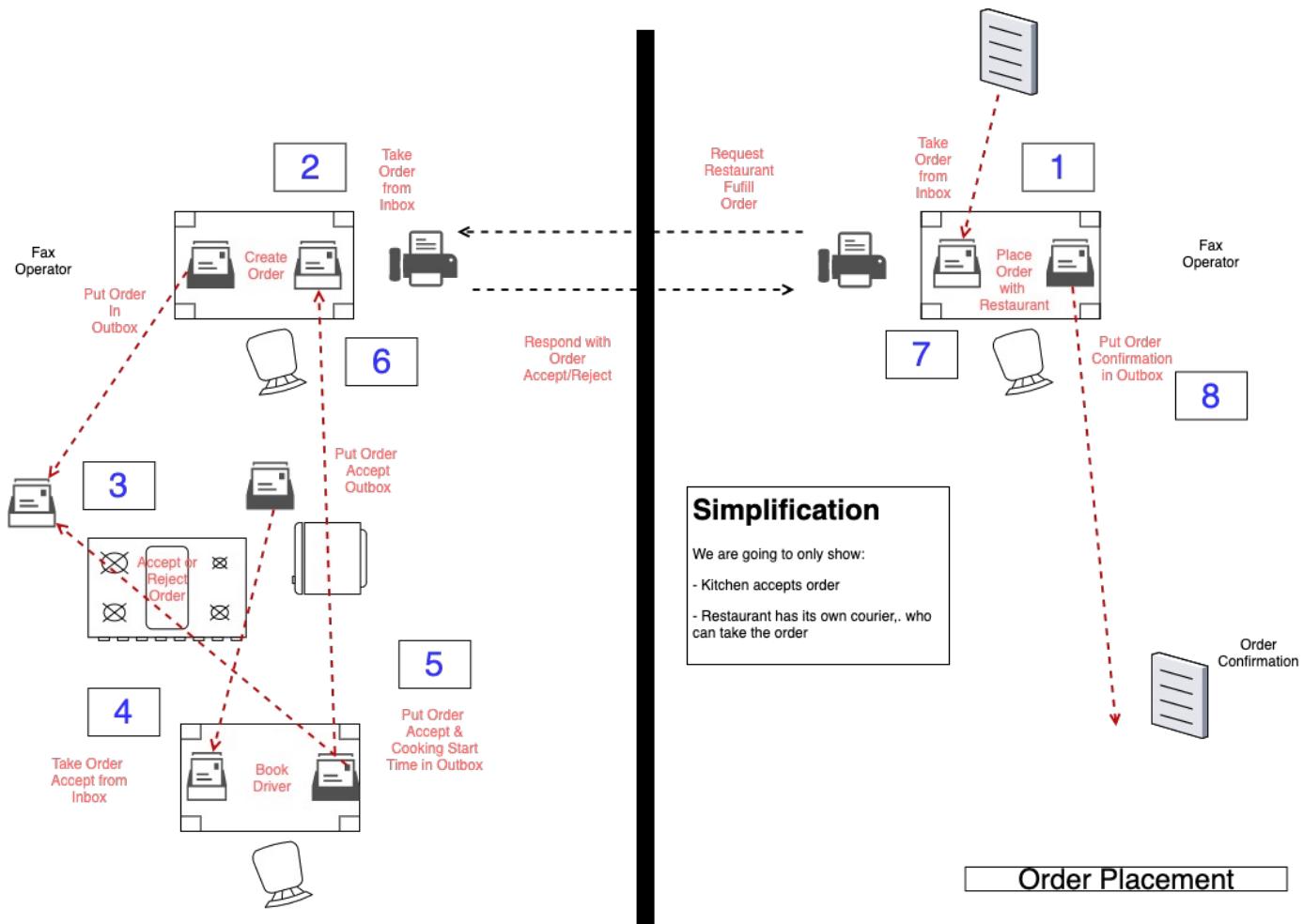
29 SEP 2021

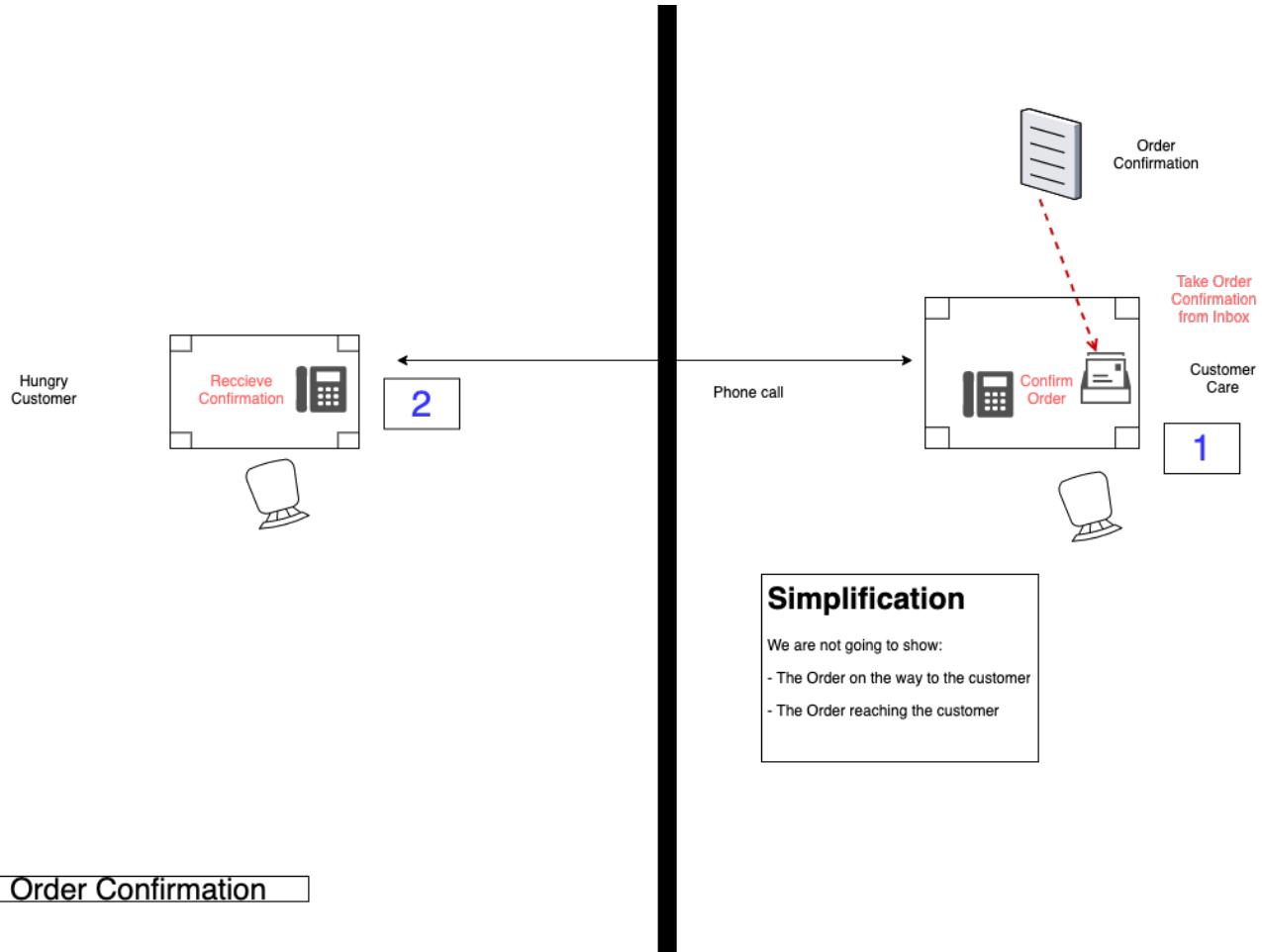


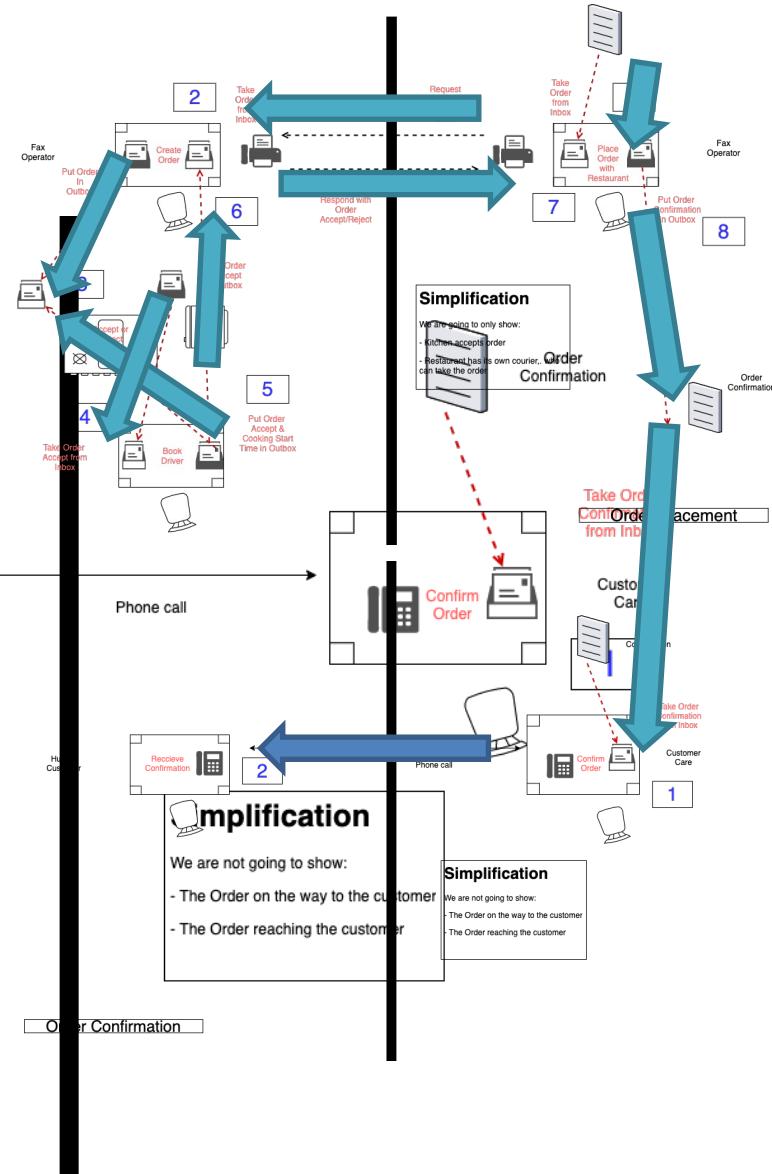
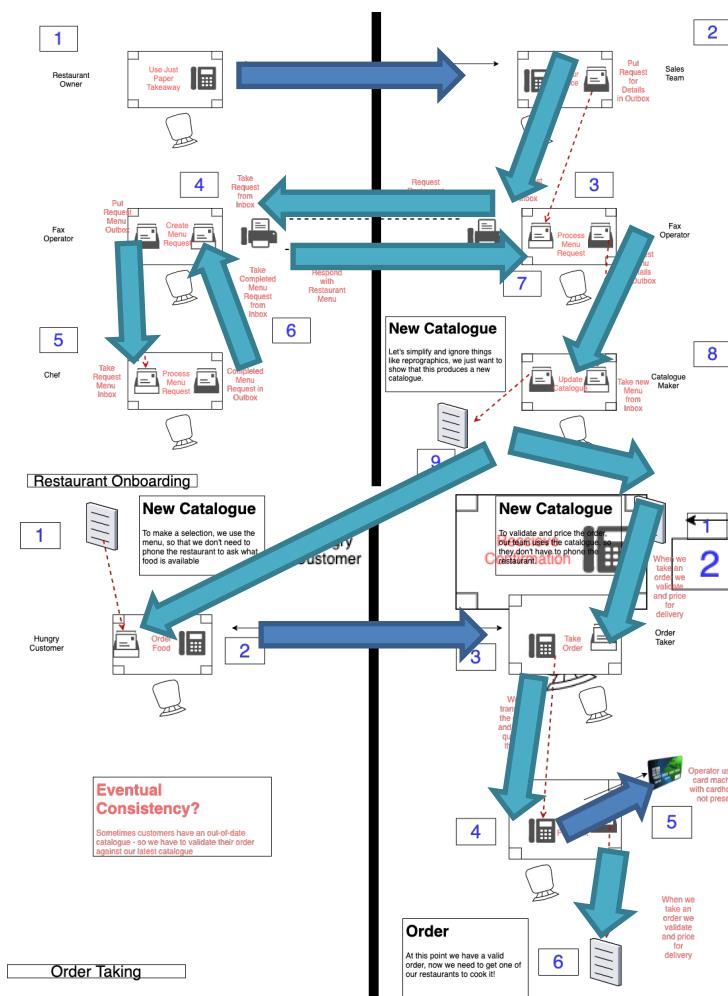


Order Wheel

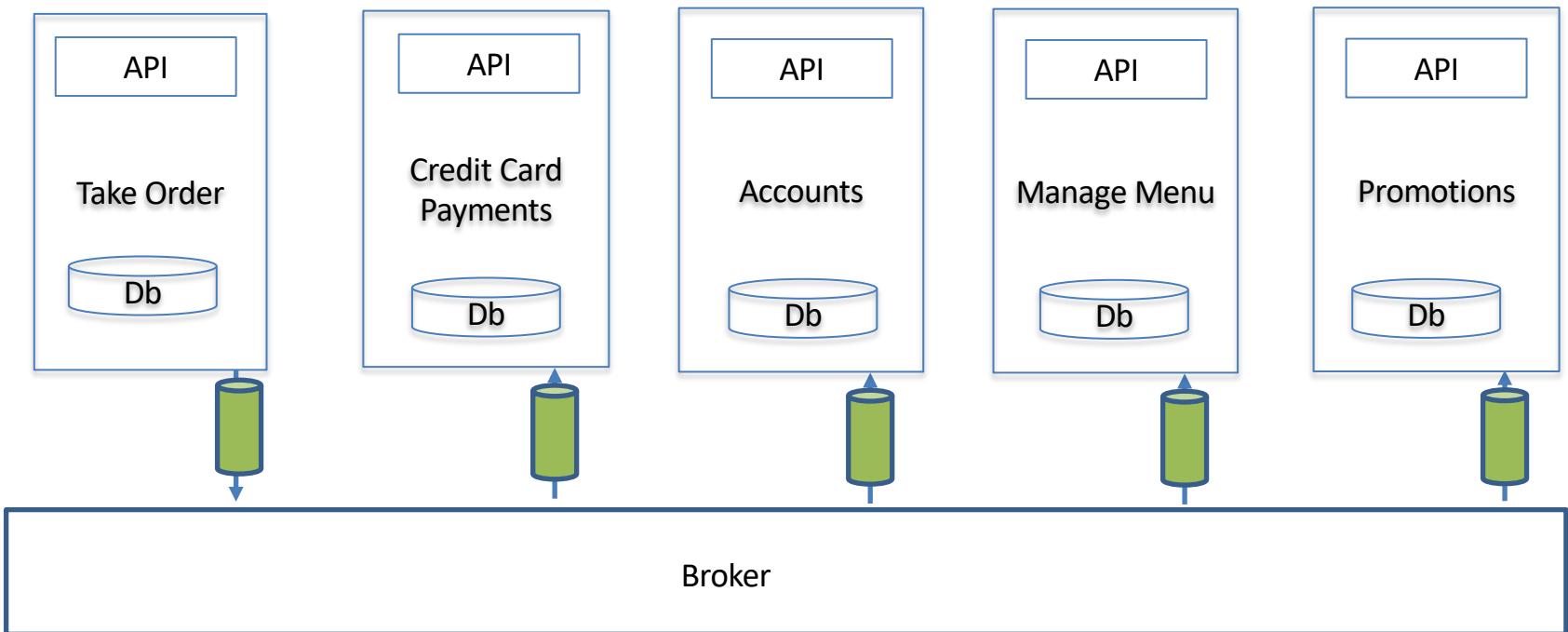




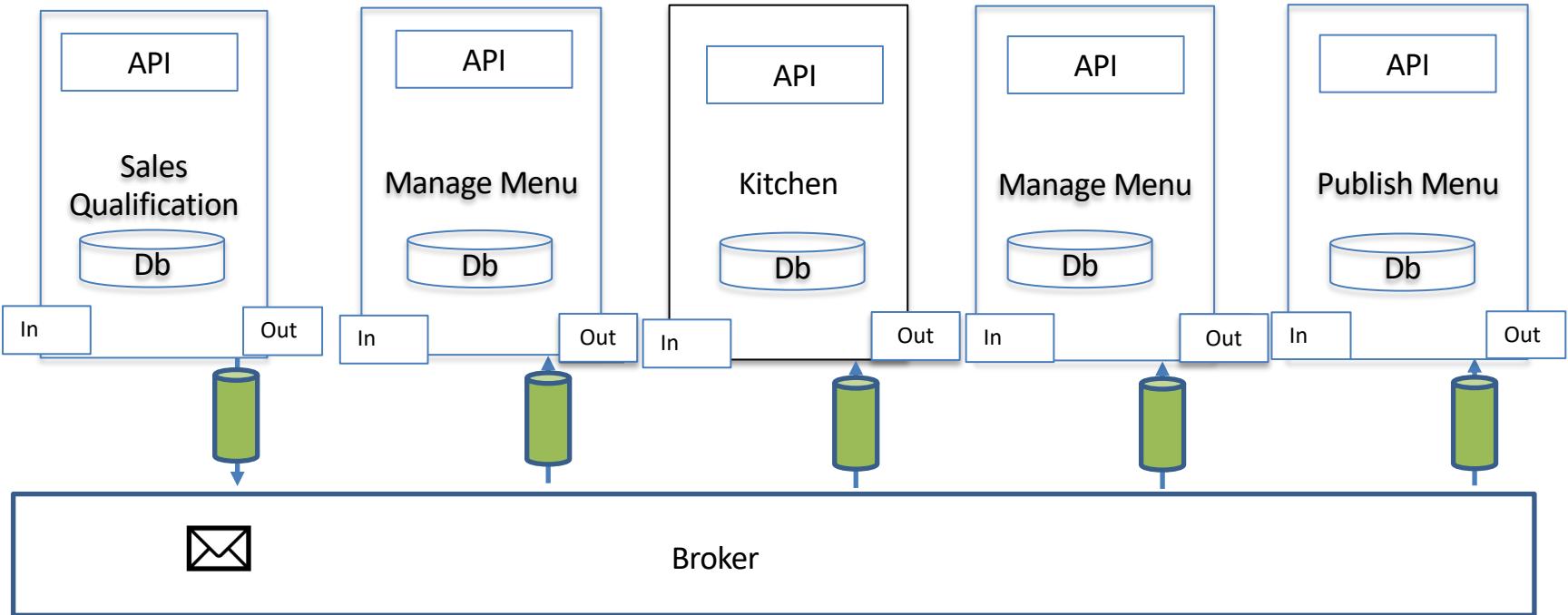




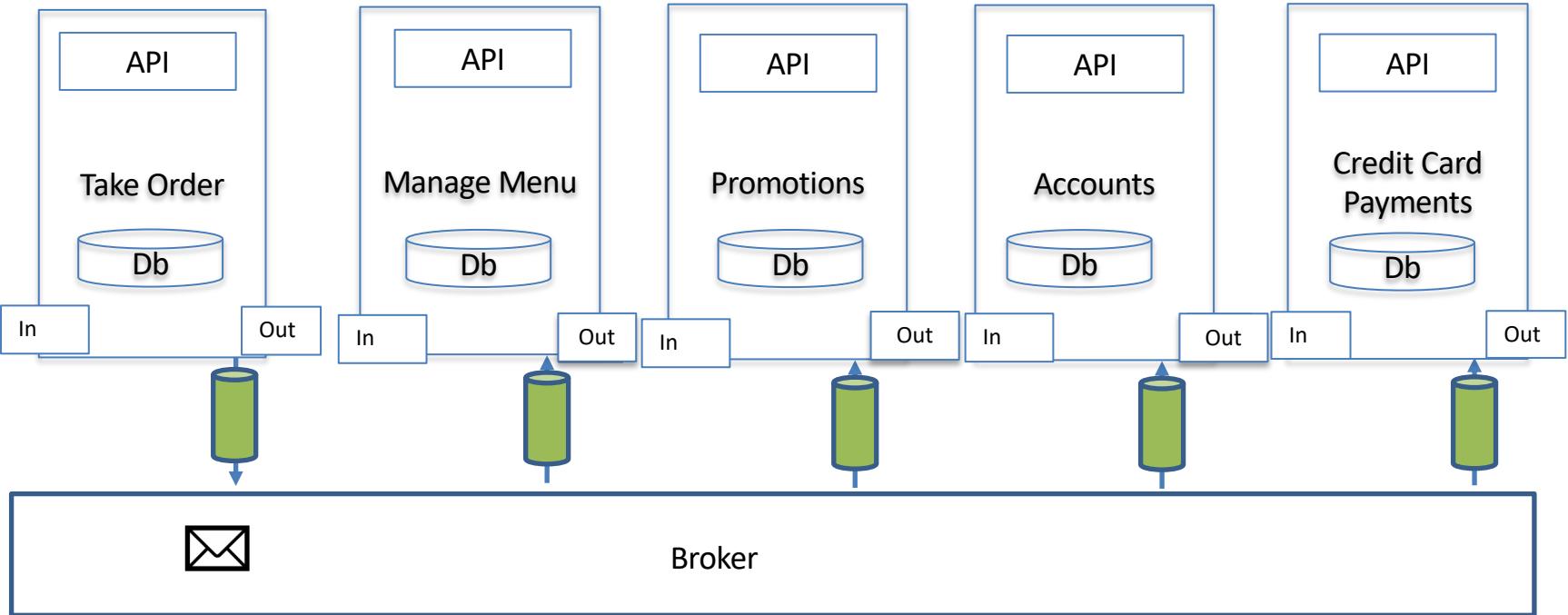
Event Driven Architecture



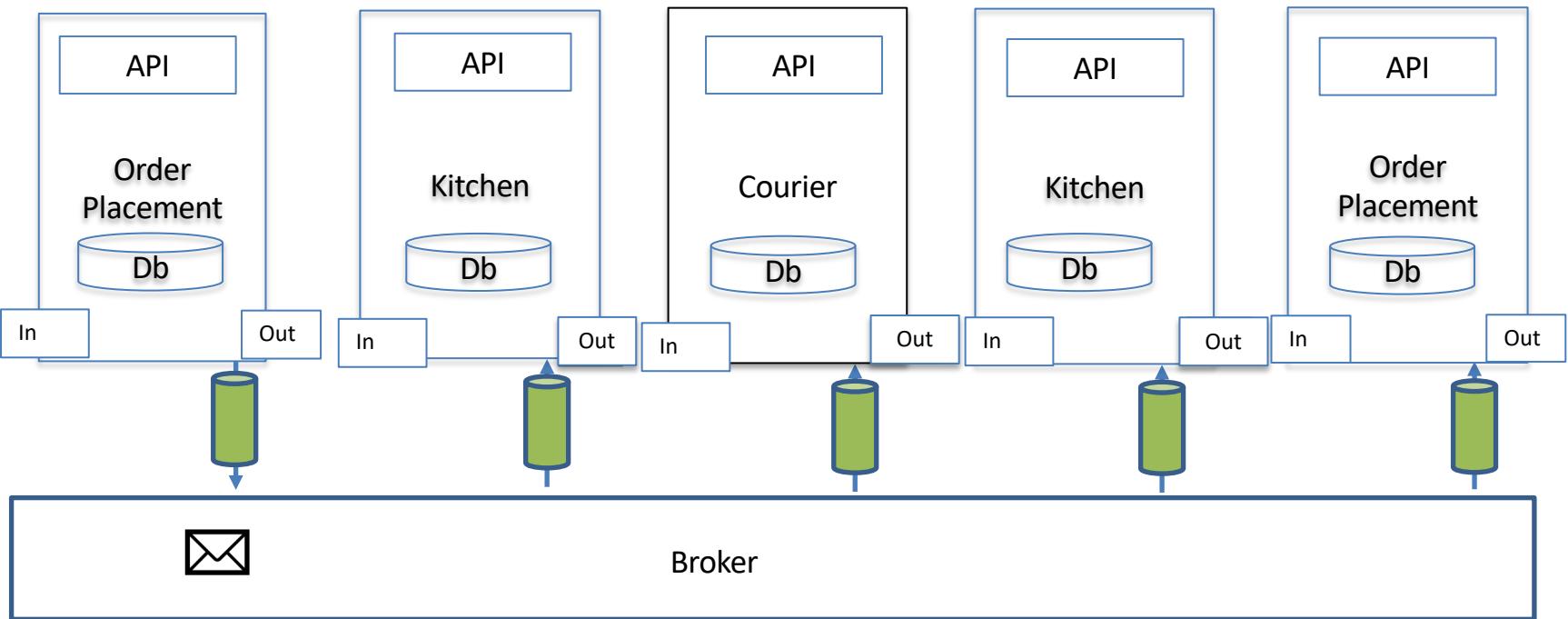
Onboard Restaurant



Take Order



Order Placement



Compensation

DO NOT USE!

SECTION A: TO BE COMPLETED BY PERSON WHO IDENTIFIED THE PROBLEM

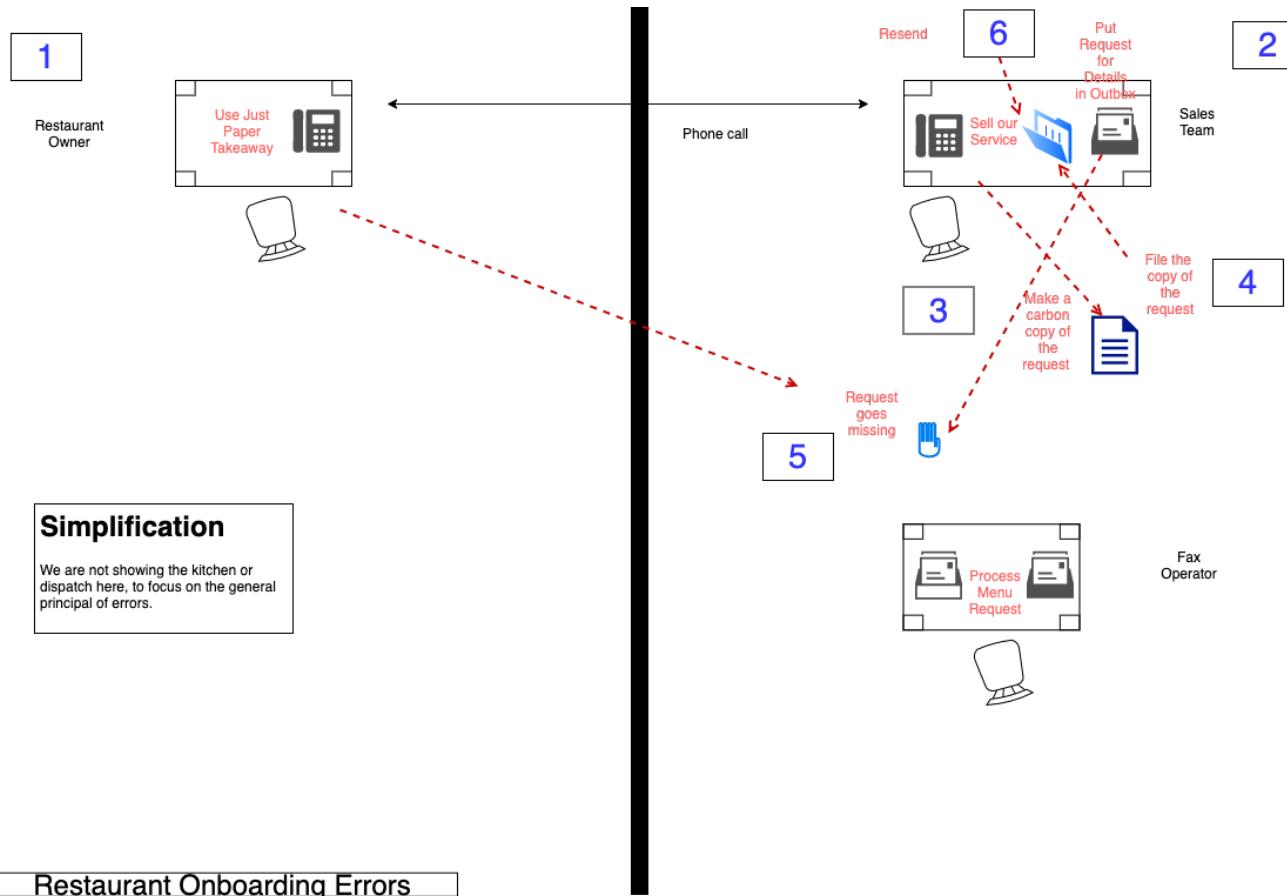
Print _____ Rev. B
Date _____
Description of Problem _____
Qty Rejected _____
Disposition _____

SECTION B: TO BE COMPLETED BY QUALITY (Even if corrected, no basis with respect to quality)

Sort _____ Date _____ Reward _____ Order Qty _____
Stop _____ Rework _____ Other _____
Disposition _____
Signature _____
Quality Signature _____
SECTION C: TO BE COMPLETED BY PERSON COMPLETING THE RECORDED ACTIONS

Quantity Inspected _____ Sort No. _____ Date _____ Reward _____ Order Qty _____
Quantity Rejected _____ Reward _____ Order Qty _____
Parts Re-inspected: Yes No Date _____
Disposition _____
SECTION D: TO BE COMPLETED BY QUALITY

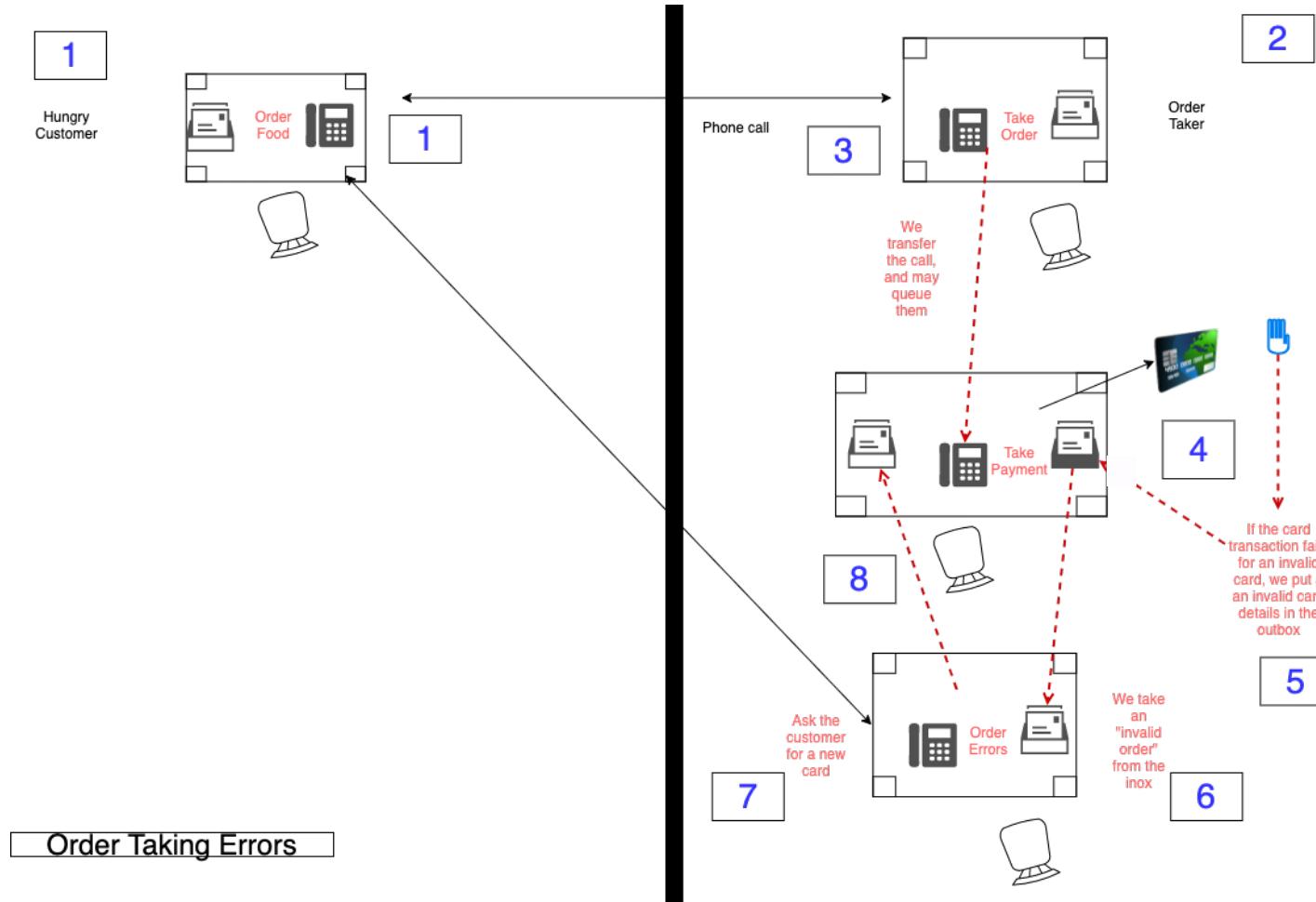
Sort Cost _____ Date _____ Reward Cost _____
Scrap Cost: _____

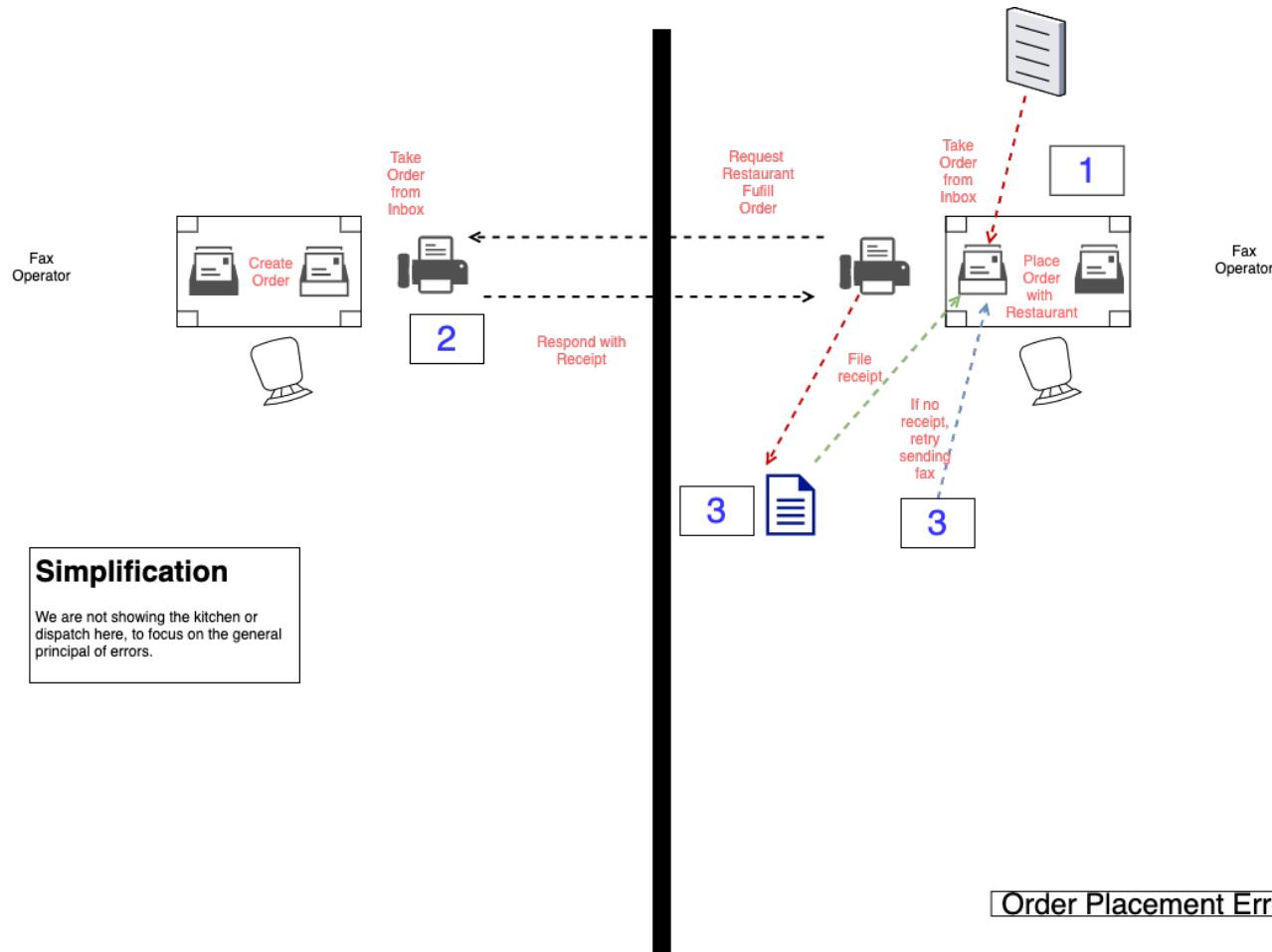


Fax Call Log

Monday, 2010-11-08 11:19

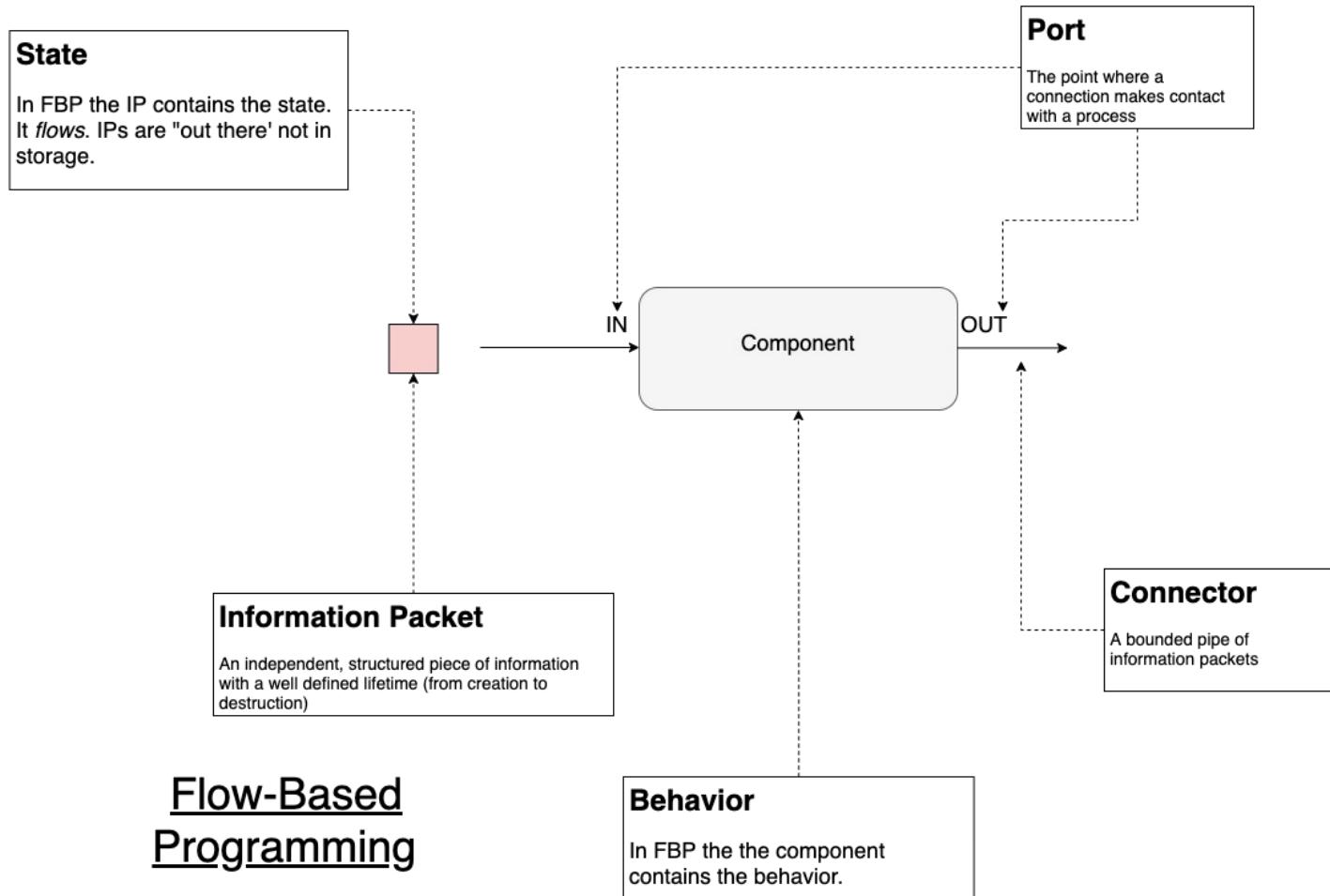
Date	Time	Type	Job #	Length	Speed	Station Name/Number	Pages	Status
2010-06-18	08:31	SCAN	92	0:25	28800	[REDACTED]	0	E-705 V.34 1M31
2010-03-22	11:39	RECV	59	0:20	26400	[REDACTED]	1	OK -- V.34 BM31
2010-03-22	11:45	RECV	60	0:20	26400	[REDACTED]	1	OK -- V.34 BM31
2010-03-22	12:29	RECV	61	0:21	26400	[REDACTED]	1	OK -- V.34 BM31
2010-09-14	14:46	SCAN	129	1:40	9600	[REDACTED]	2	E-606 V.29 AR30



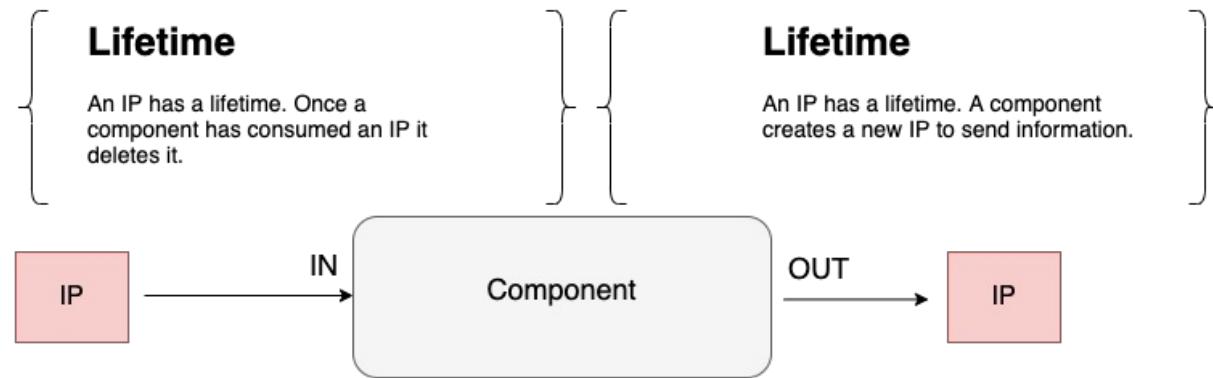


Flow Based Programming

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

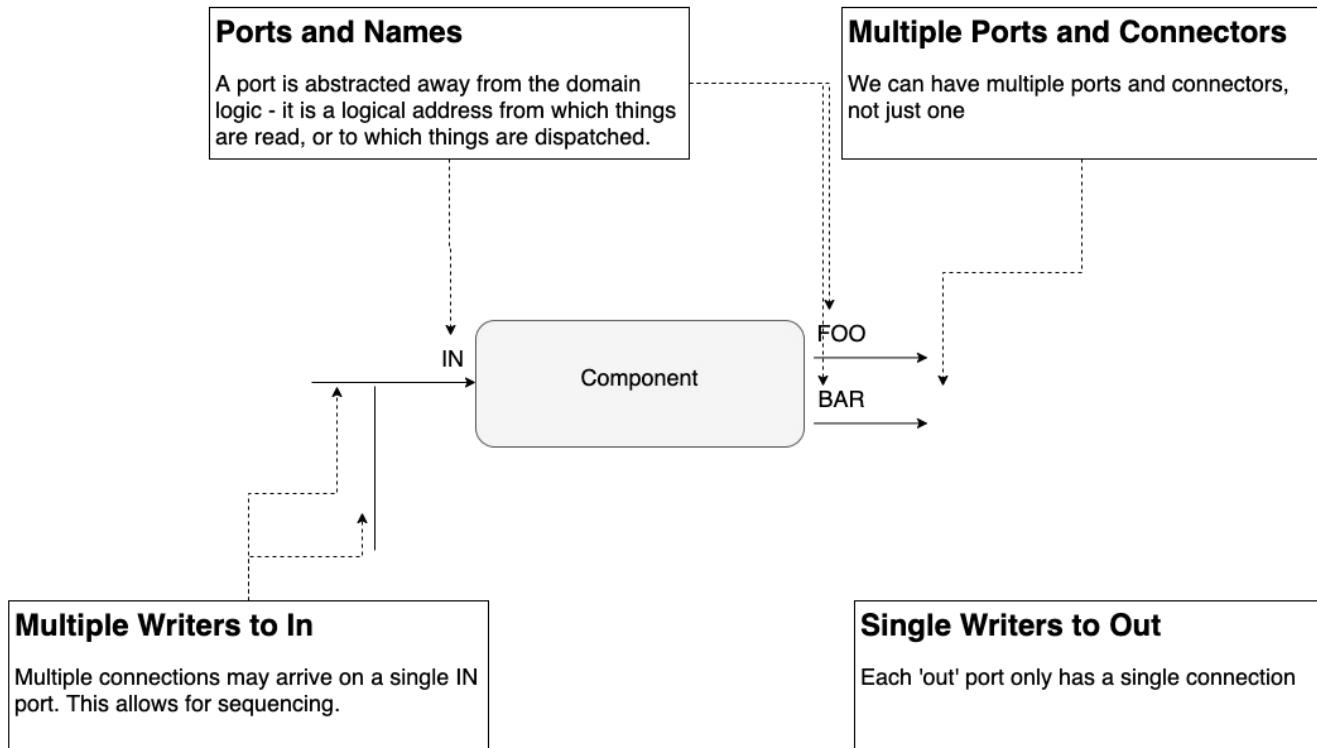


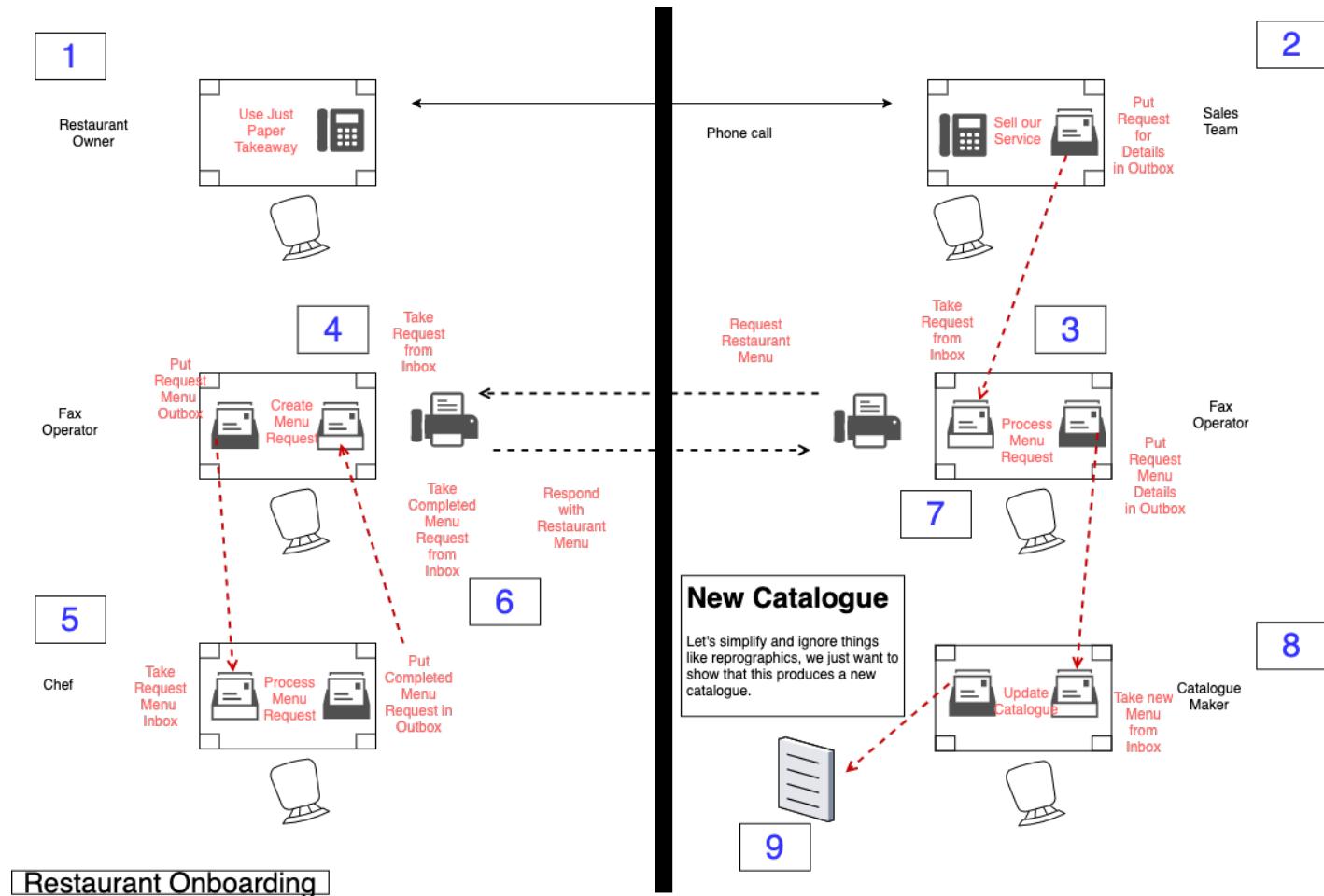
Flow-Based Programming

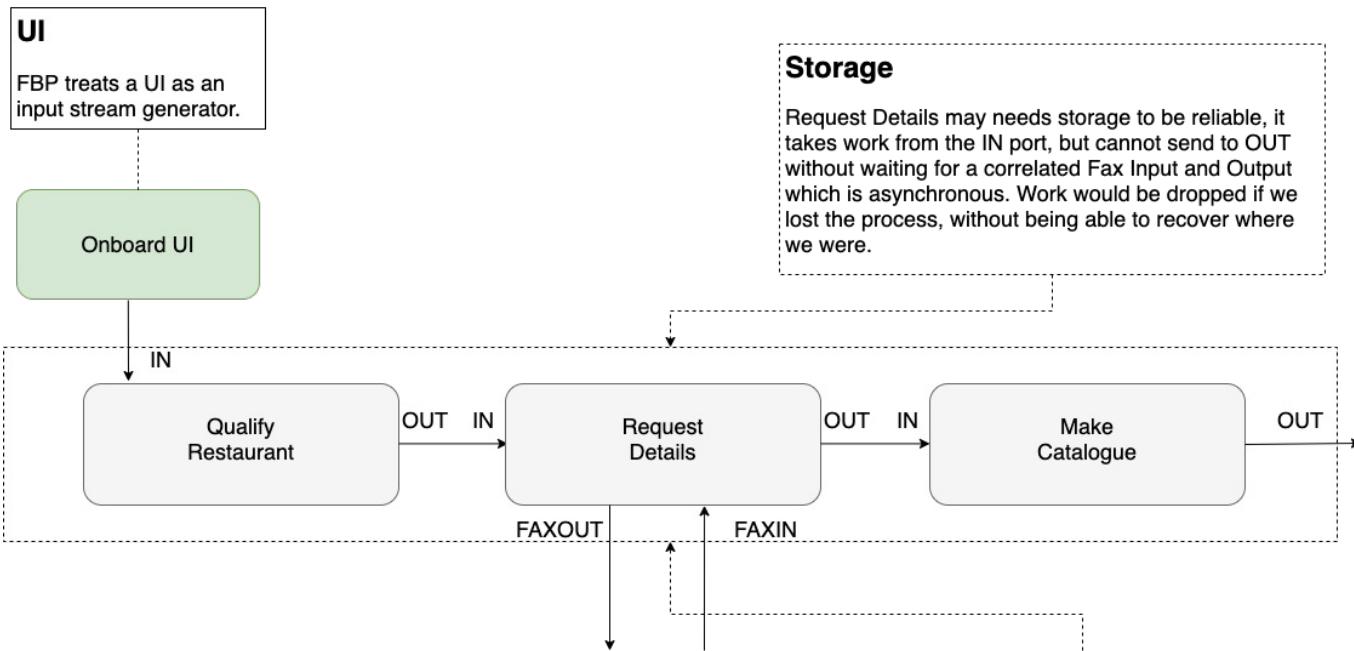


Flow-Based Programming

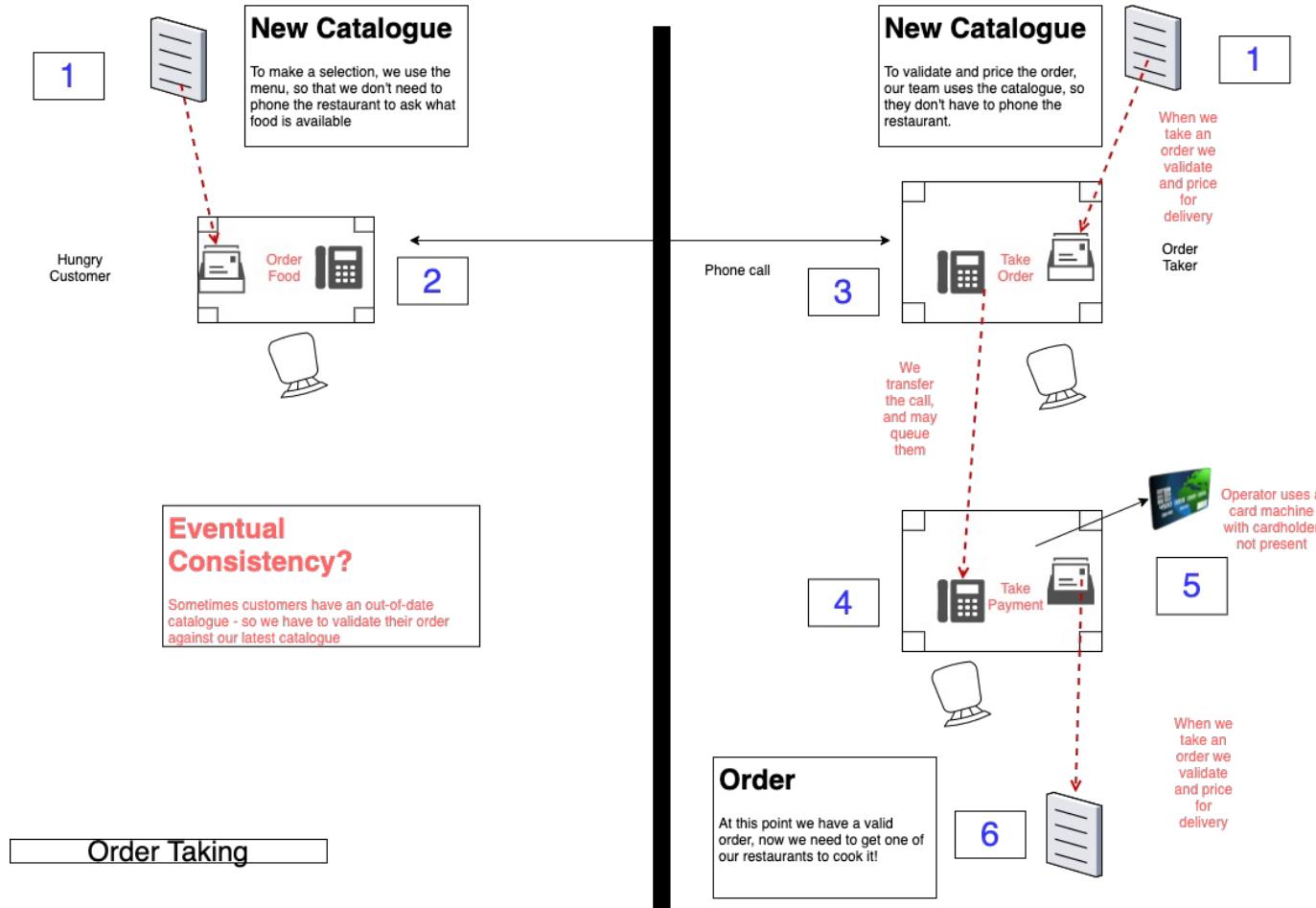
Ports and Connectors

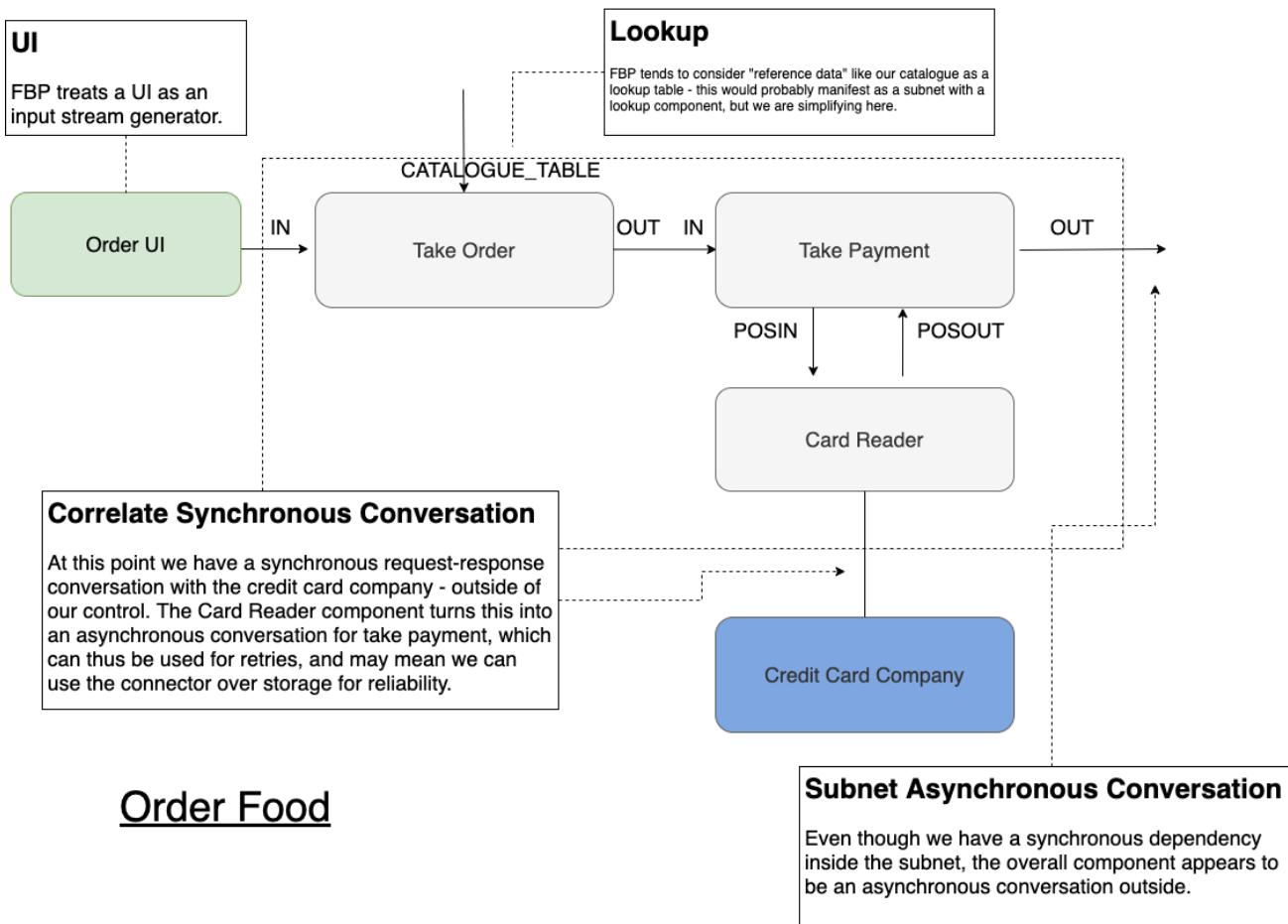


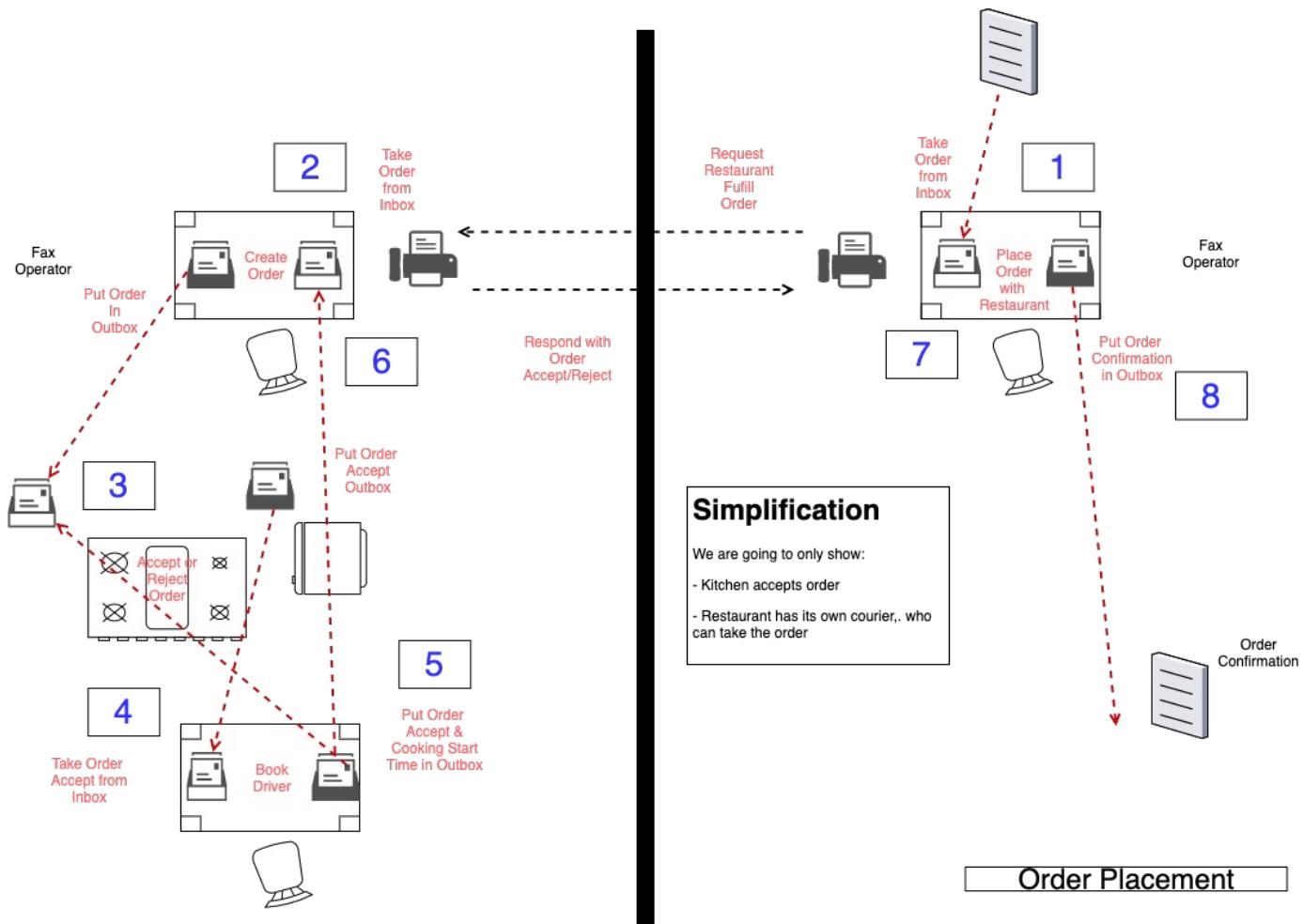




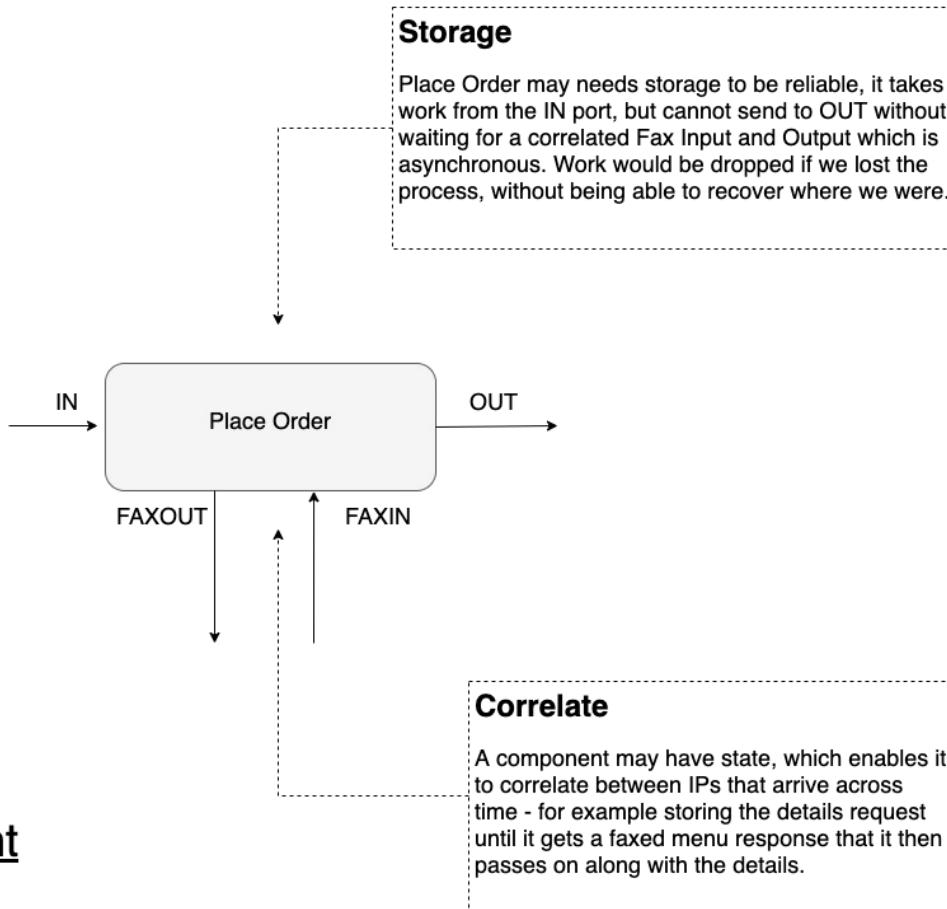
Onboard
Restaurant

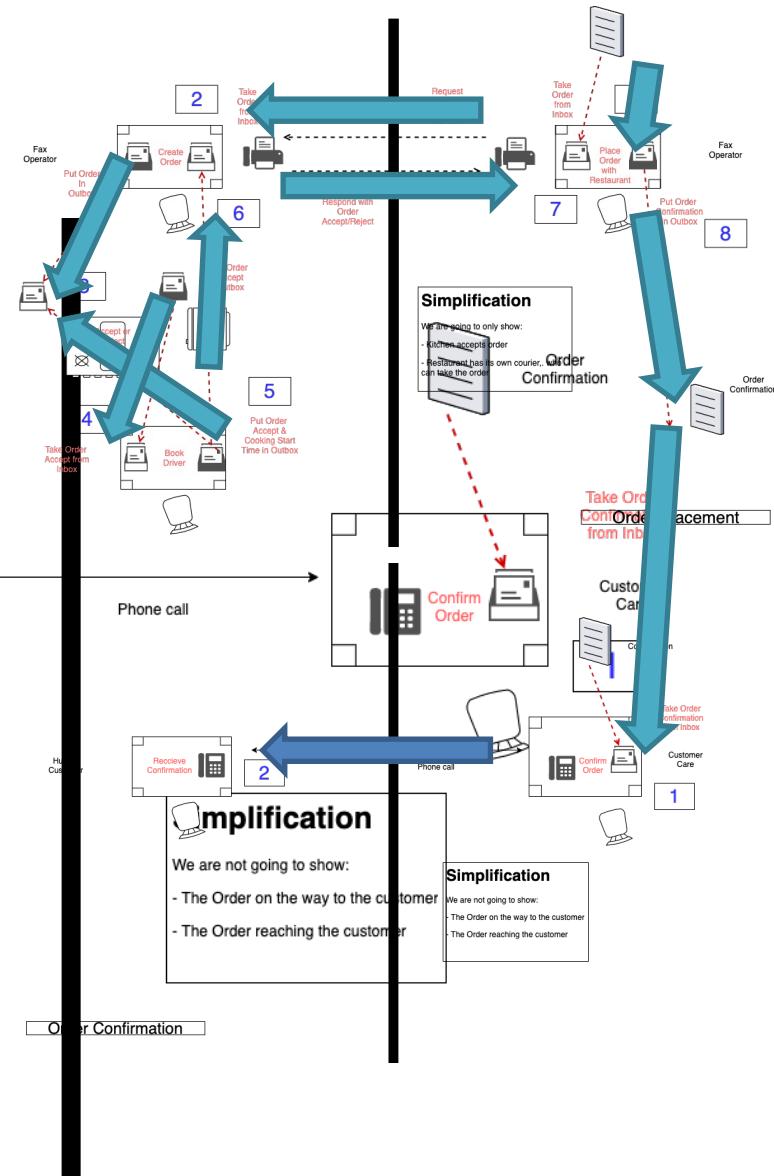
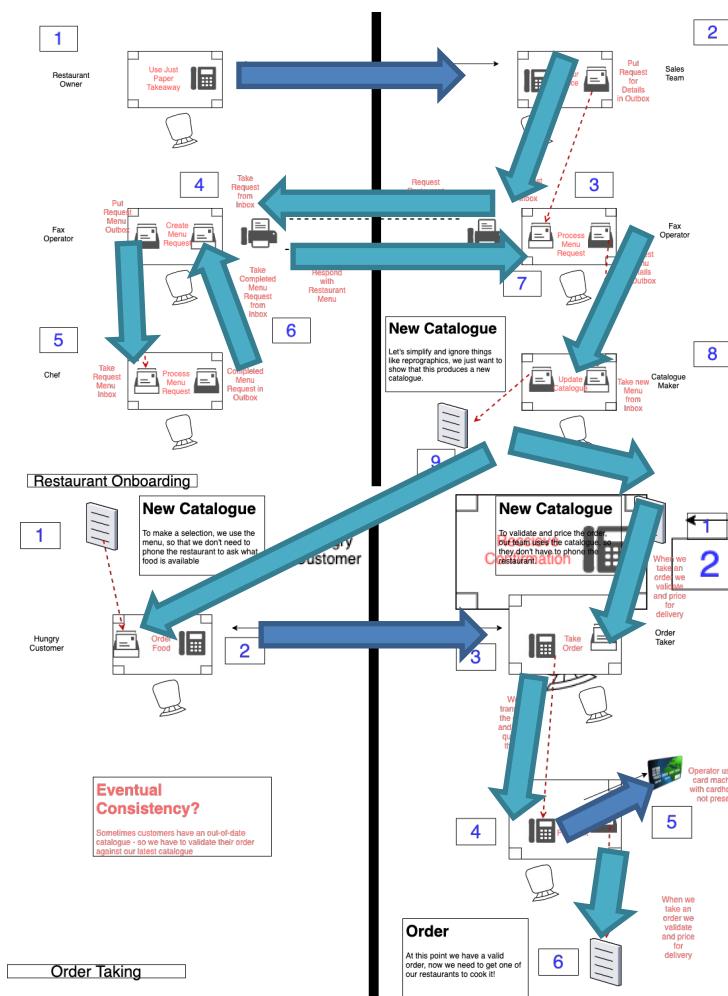


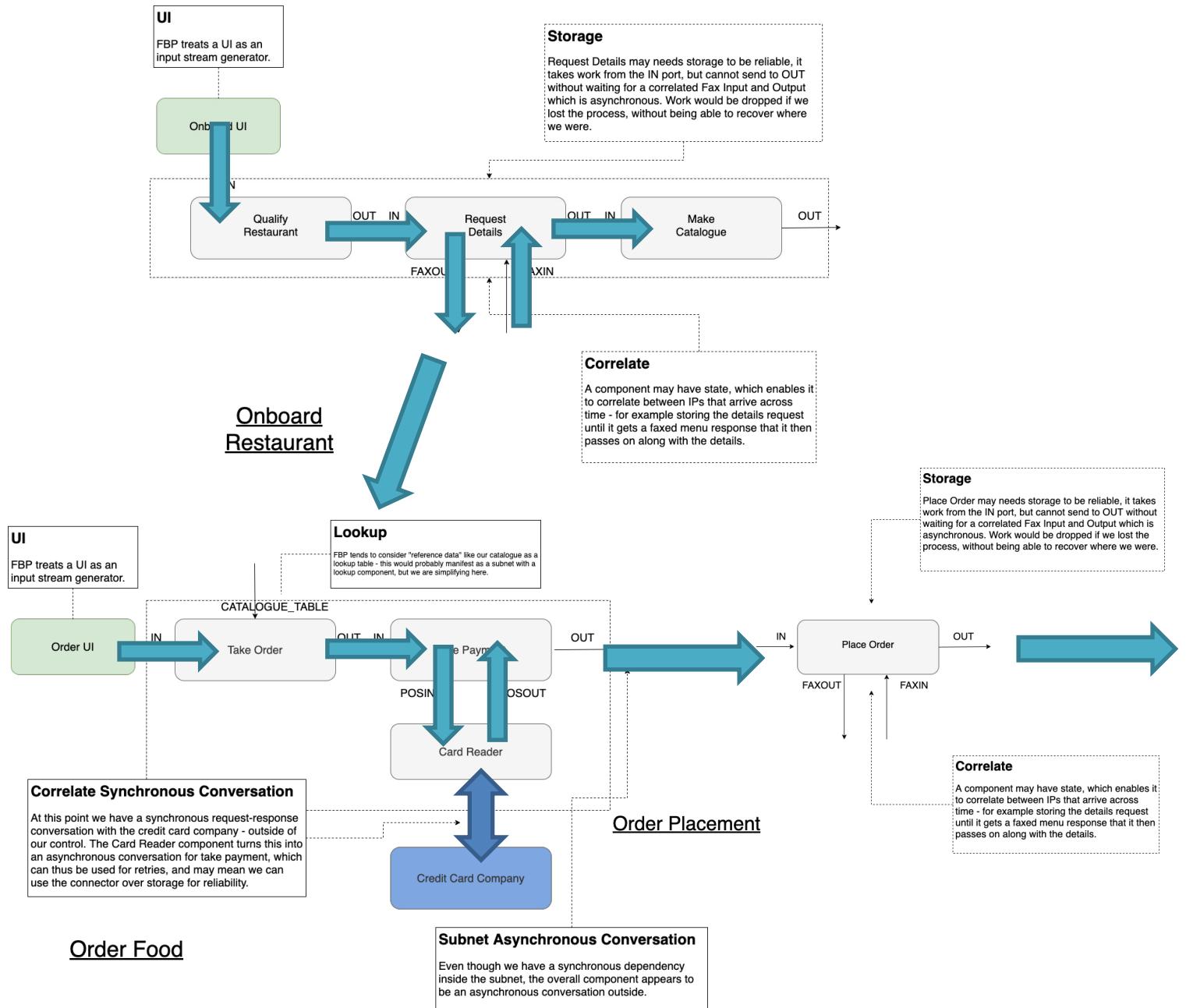




Order Placement









Choreography



Orchestration

Queue

- Store and Forward
- Delete and Re-queue
- Lock & Read Past

App

- Process and Acknowledge Work
- Raise New Work

Command

- An Information Packet
- One & One Only Subscriber
- Actionable, Independent, Discrete

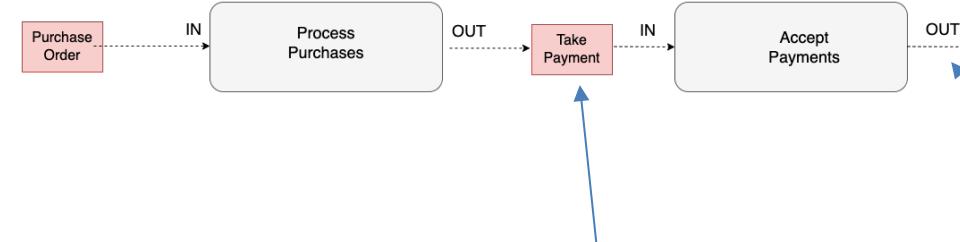
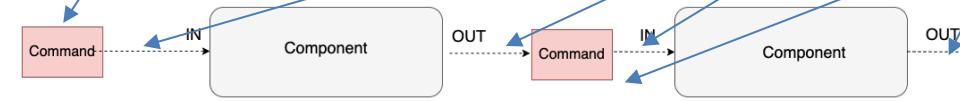
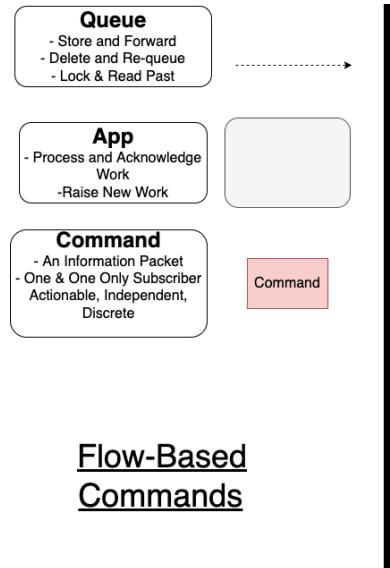
Flow-Based Commands

A Command is an Imperative. An instruction to another party to do something. It is usually issued as part of a workflow.

A useful question is: is there another service I expect to handle this request? If so you have a Command.

A Command is Discrete. We use a Queue for a Command

A Command may be *fire-and-forget* or *request-reply*



I expect that Accept Payments will take the customer's payment for this order.
This is a Command.

I expect that Accept Payments will reply to my request.
What does this?

Stream

- Store and Forward
- Immutable and Ordered
- Partitioned

Queue

- Store and Forward
- Delete and Re-queue
- Lock & Read Past

App

- Process and Acknowledge Work
- Raise New Work

Command

- An Information Packet
- One & One Only Subscriber
- Actionable, Independent, Discrete

Event

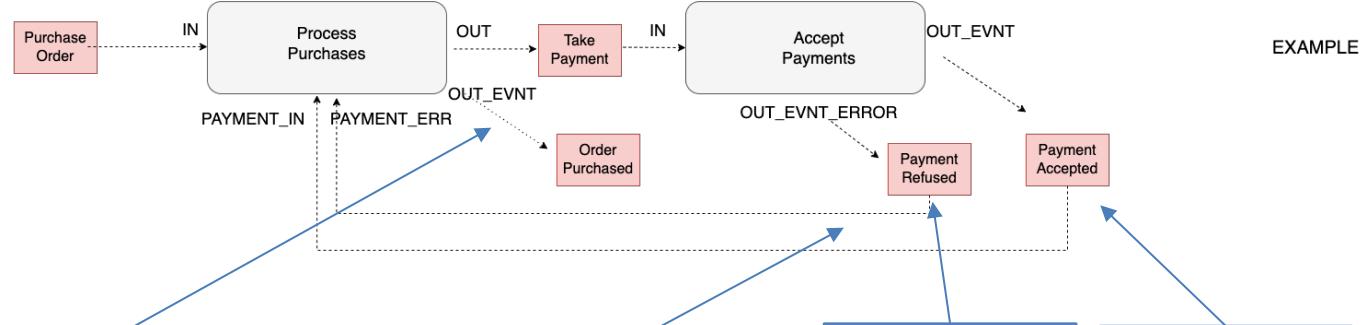
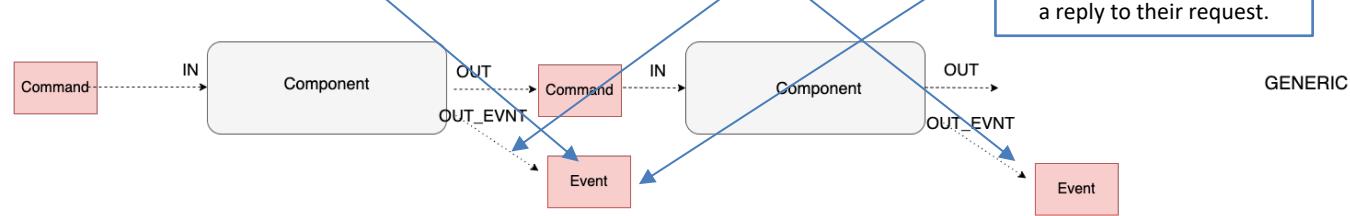
- An Information Packet
- One & One Only Subscriber
- Actionable, Independent, Discrete

Flow-Based Events

An Event is a Fact that tells us something happened. It is often issued because we have serviced a Command.

An Event may be Series or Discrete. We can use a Stream or Queue for an Event

An Event may be the Reply in a Request-Reply. By using a correlation id, the sender can asynchronously receive a reply to their request.



A Status Update is often Series. If it has zero or more subscribers we may prefer series – over a Stream

A Reply could be Series or Discrete. If it is a reply to just one sender we may prefer Discrete – over a Queue

An error is just an event as a **reply** that indicates failure.

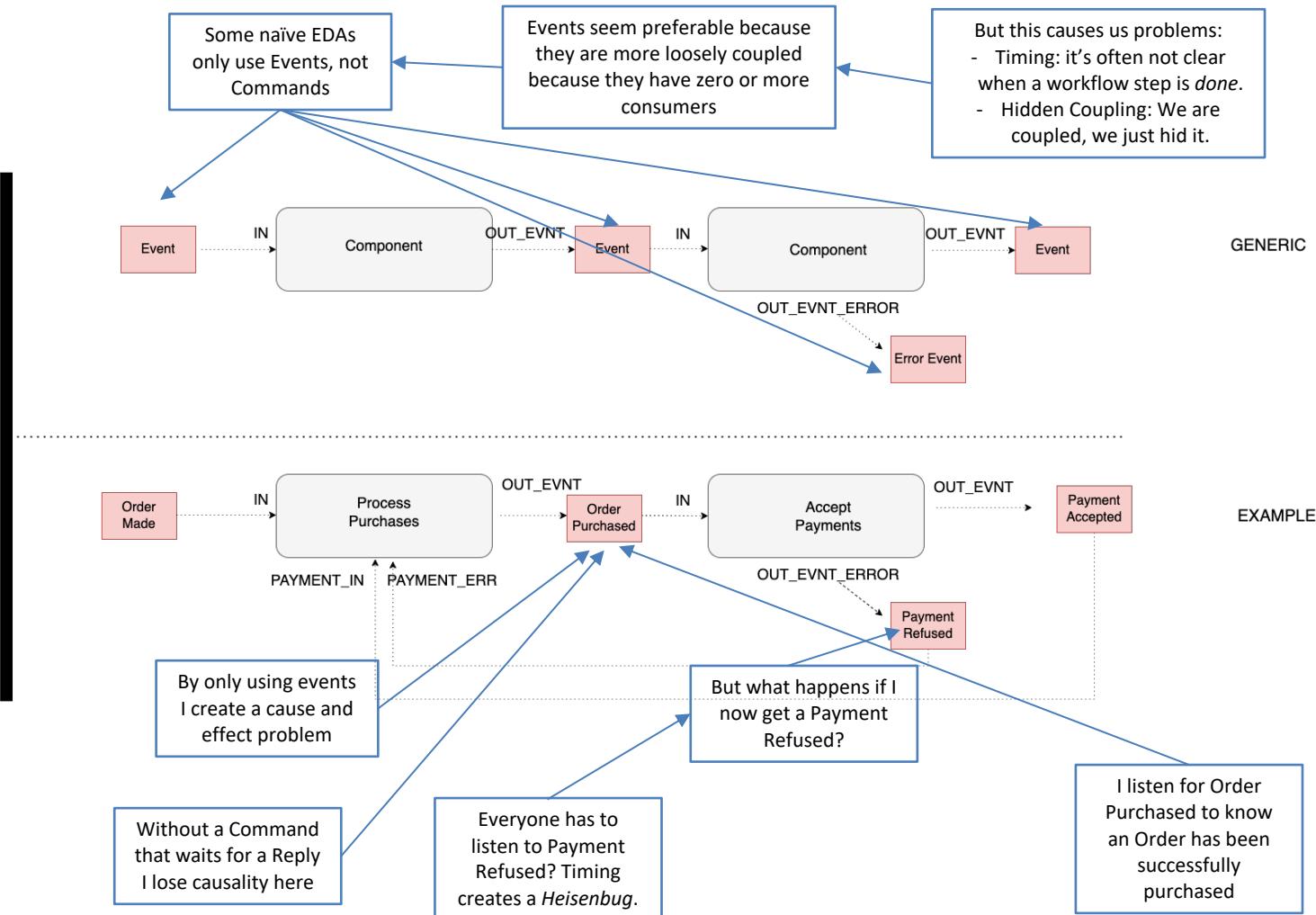
Accept Payments is a branch in the flow, so it does not raise a command, just an event as a **reply**.

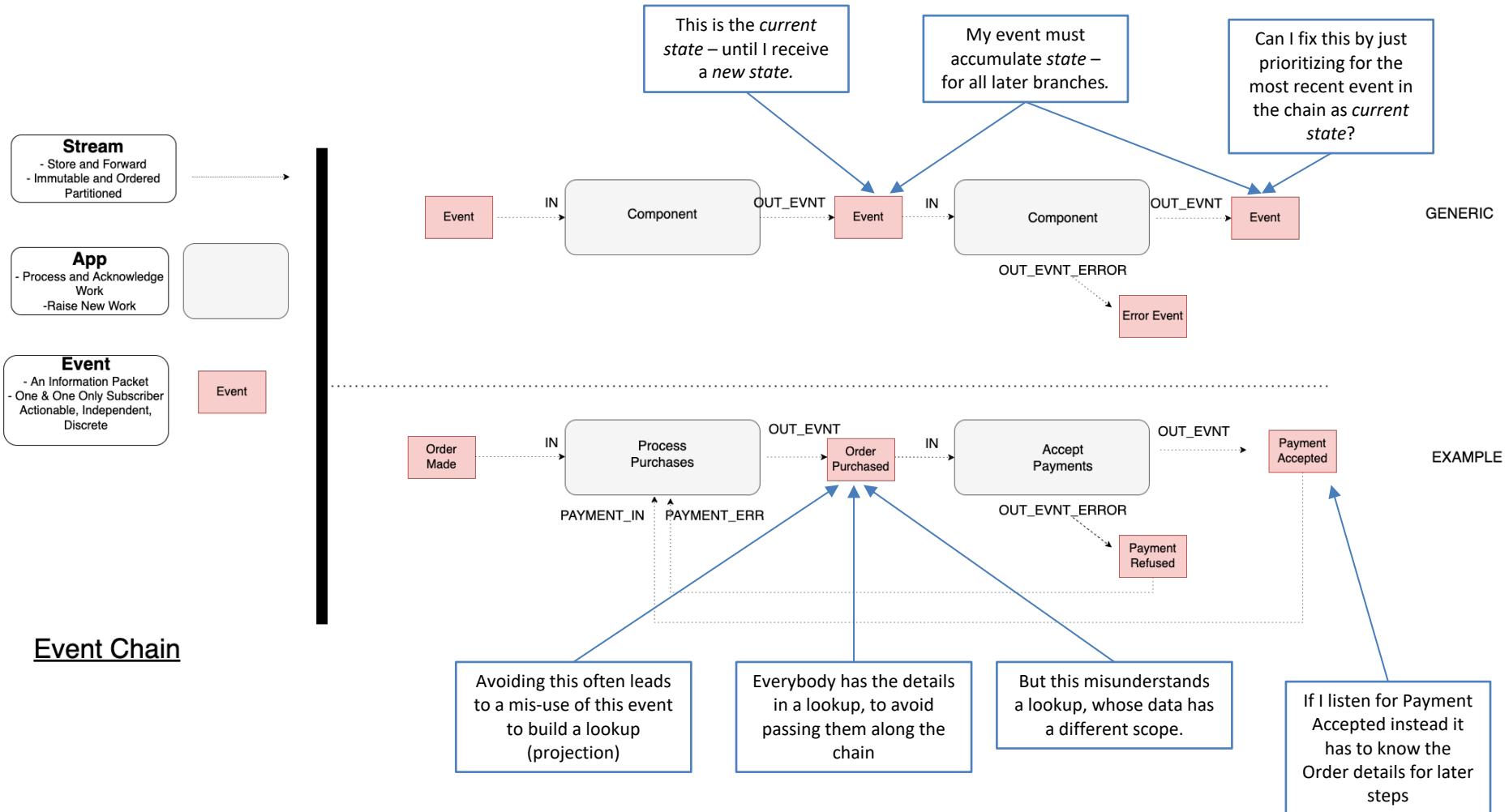
Event Chain

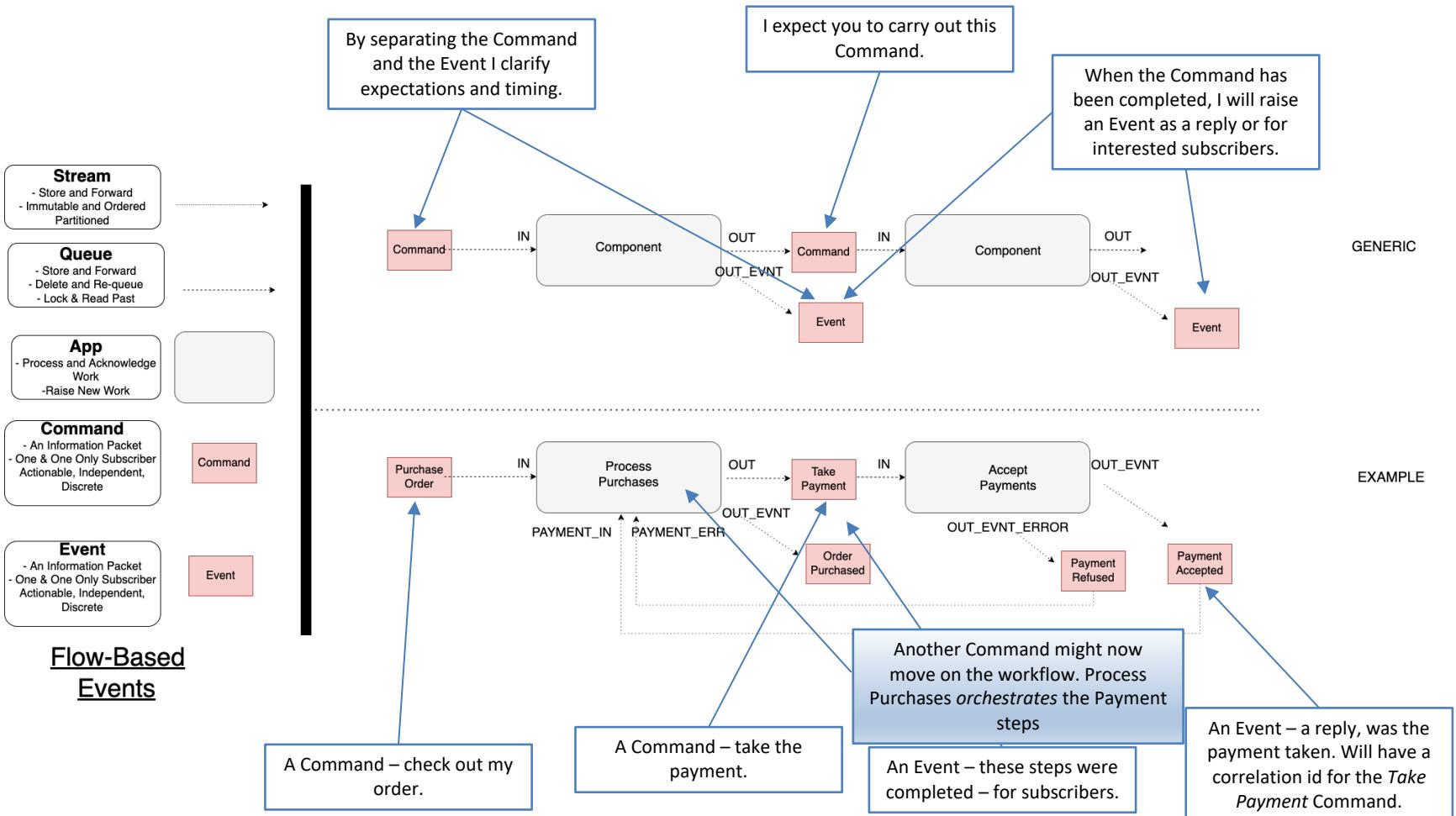
Stream
 - Store and Forward
 - Immutable and Ordered
 - Partitioned

App
 - Process and Acknowledge Work
 - Raise New Work

Event
 - An Information Packet
 - One & One Only Subscriber
 - Actionable, Independent, Discrete

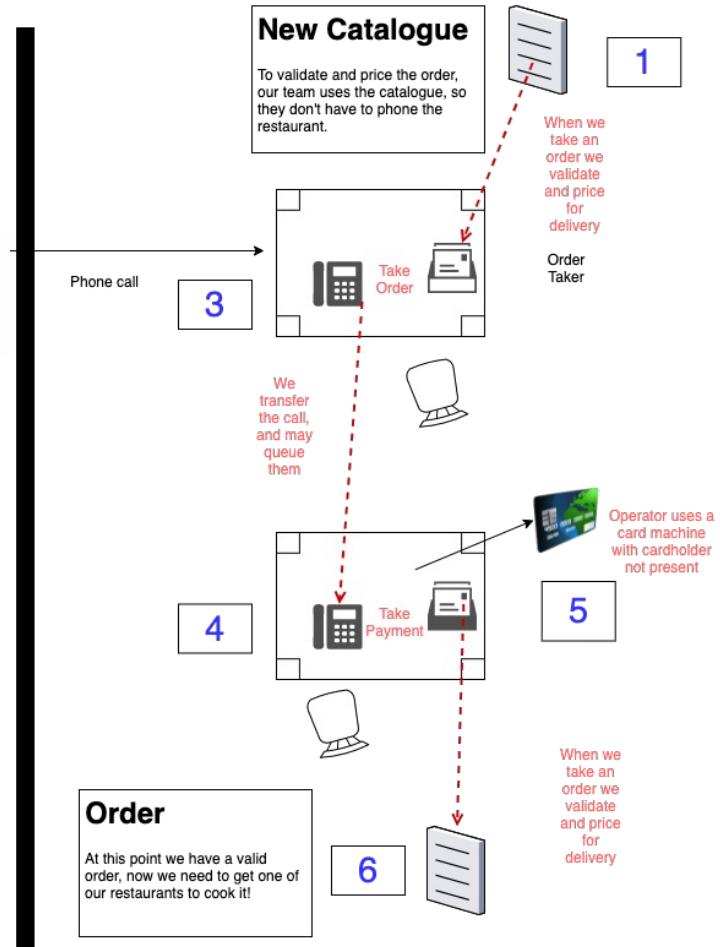
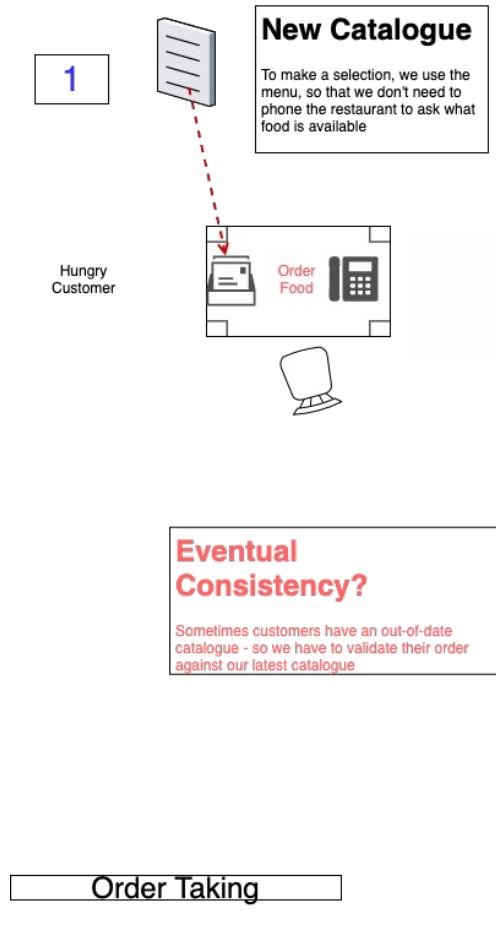


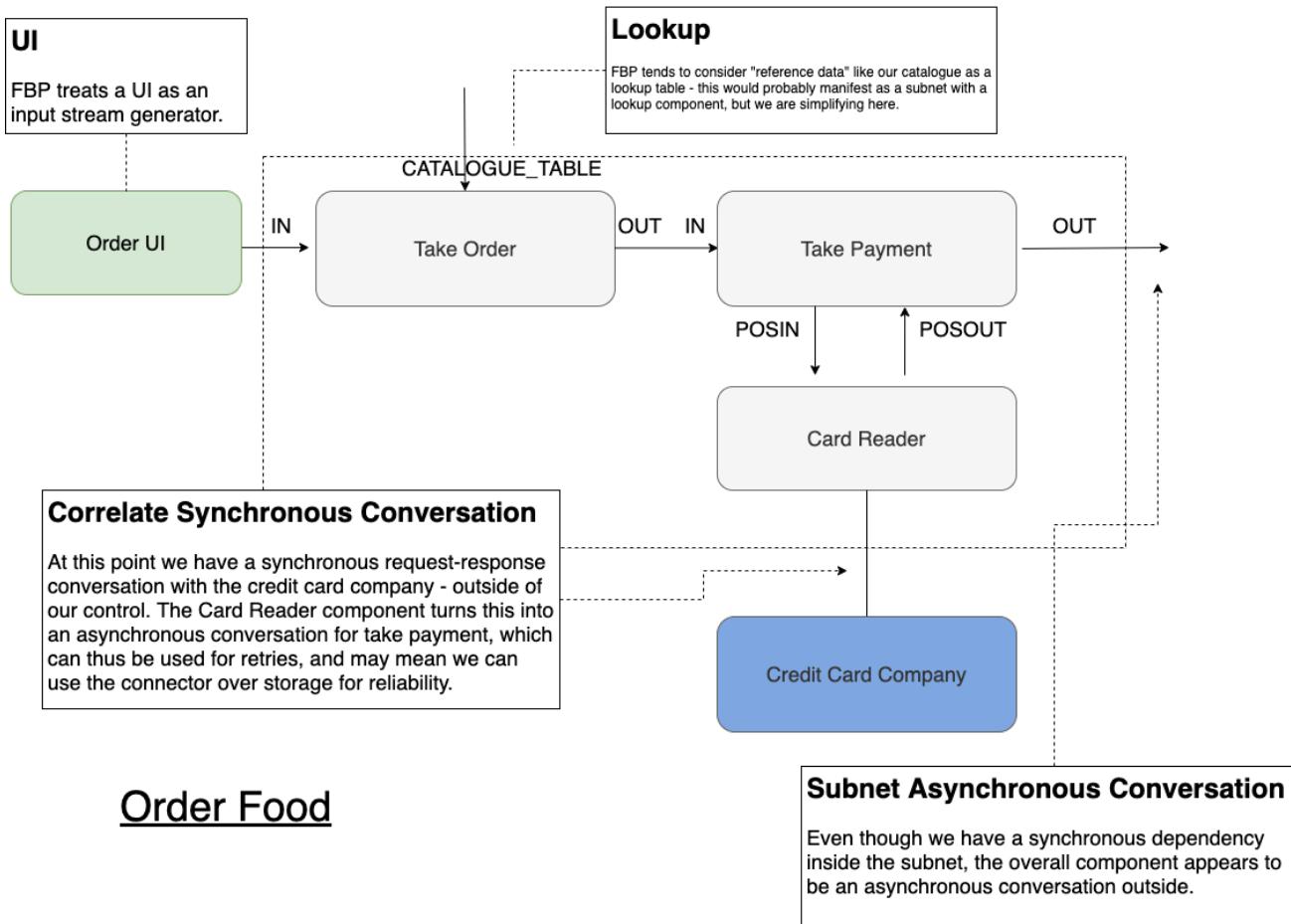


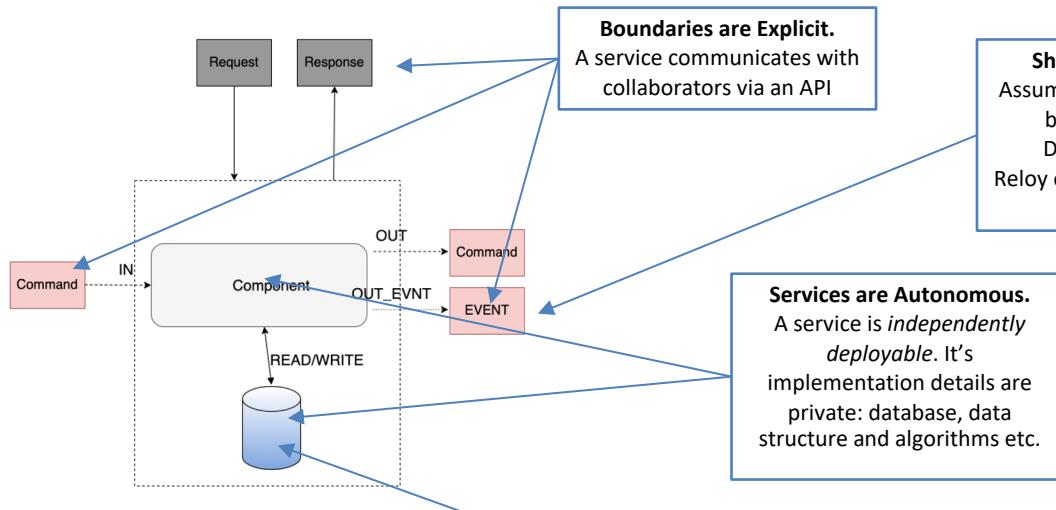
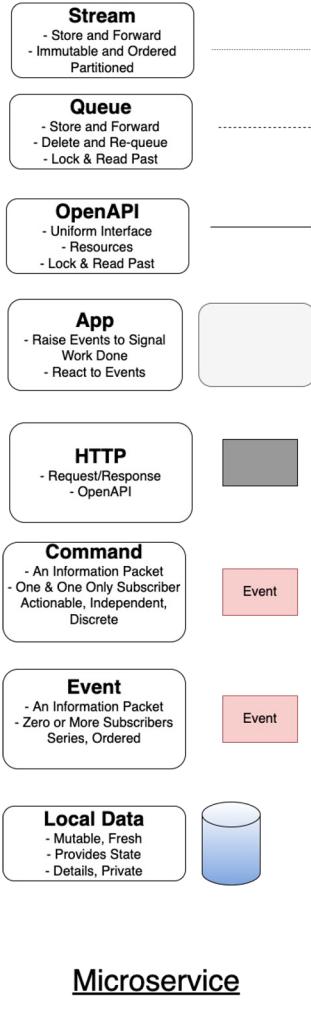


Event Carried State Transfer

Reference Data/Forward Cache/Data Pump/View/Projection





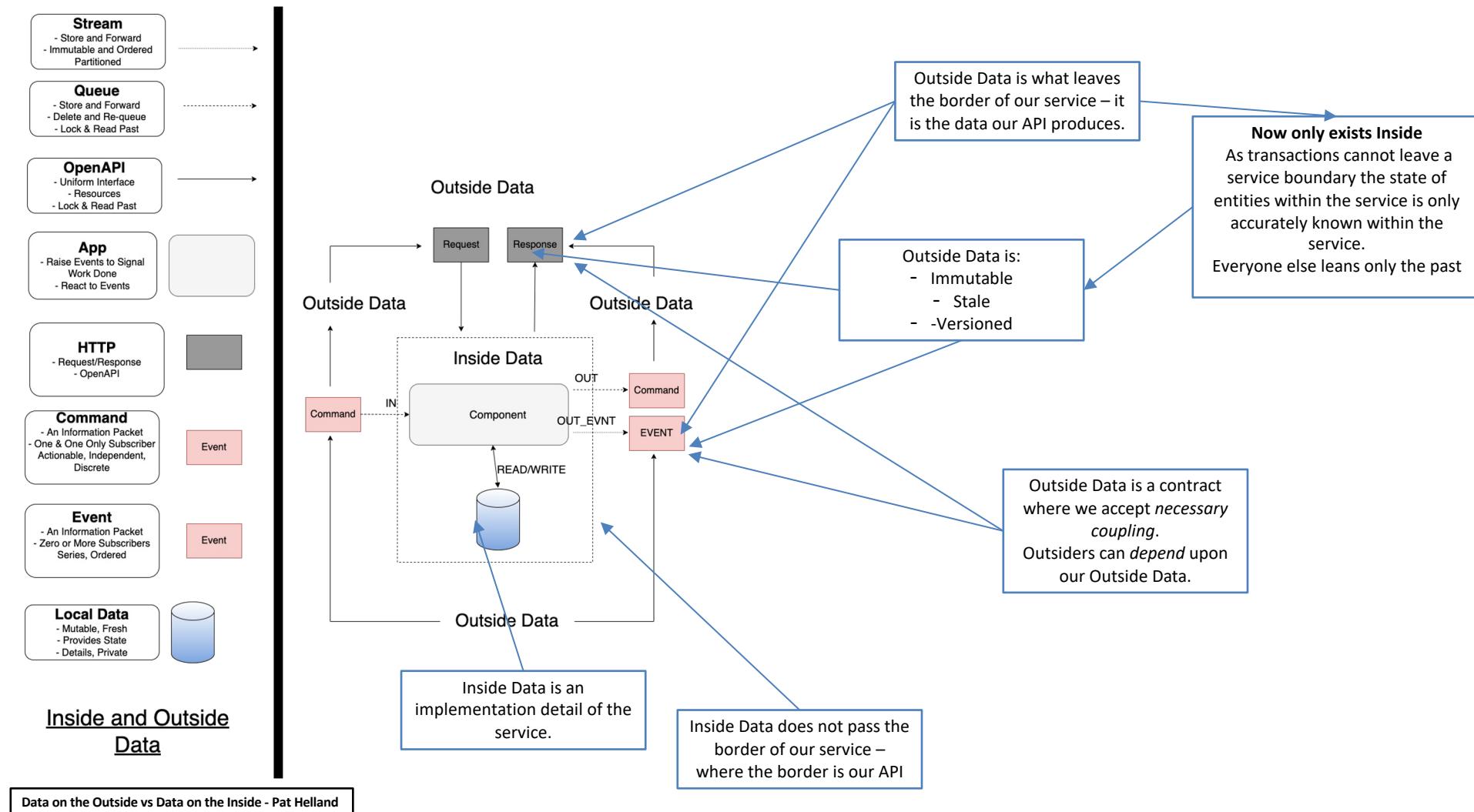


Boundaries are Explicit.
A service communicates with collaborators via an API

Share Schema Not Type
Assume that collaborators may be on different stacks
Describe your schema
Relay on a stacks tooling to gen. code from schema

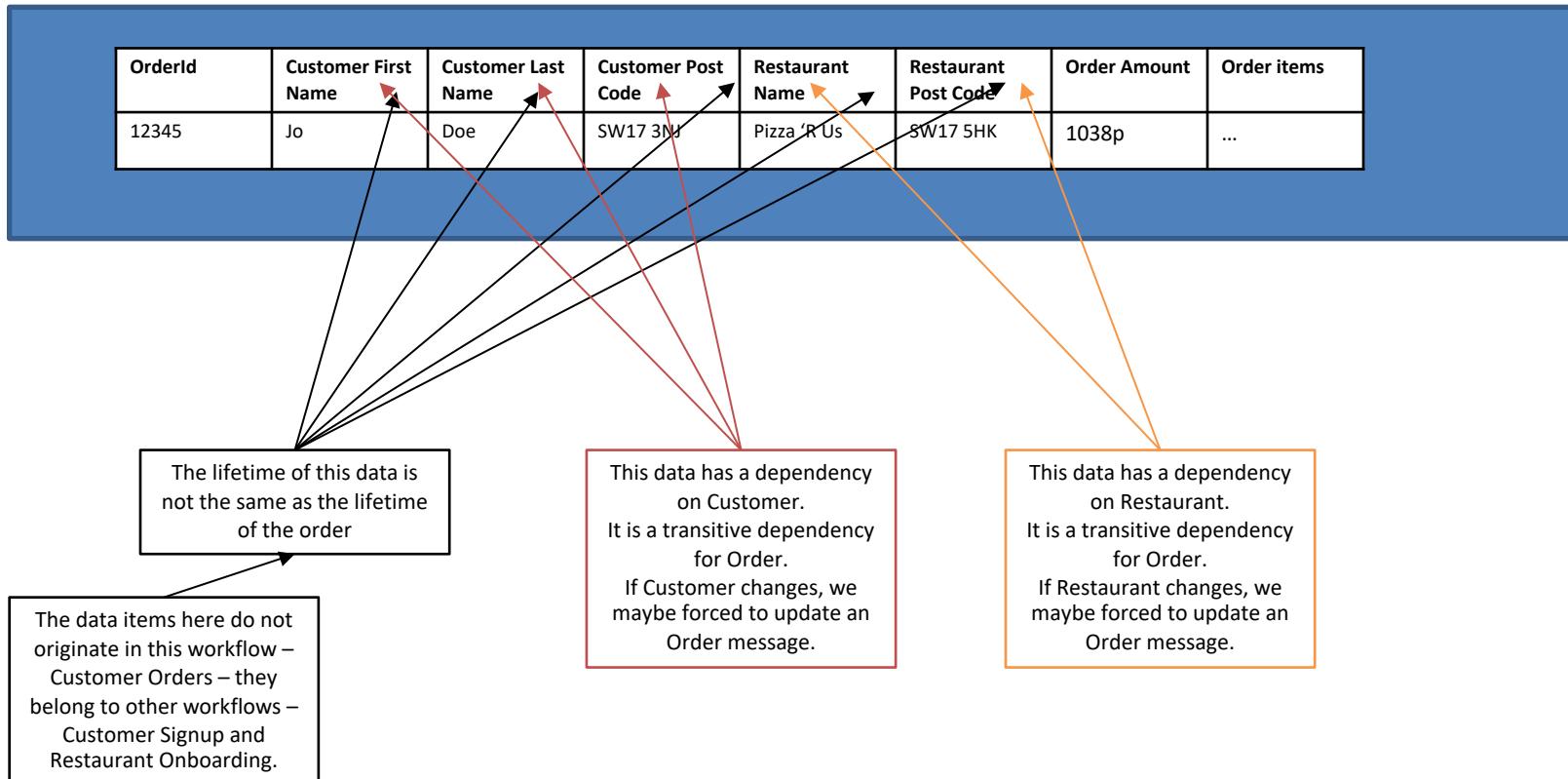
Services are Autonomous.
A service is *independently deployable*. Its implementation details are private: database, data structure and algorithms etc.

No Distributed Transactions.
A service does not participate in a distributed transaction.
It will not hold a lock for another service.



Transitive Dependencies

Purchase Order Message



Normalized

Purchase Order Message

OrderId	Order Amount	Order items	CustomerId	RestaurantId
12345	1038p	...		

We normalize the message contents.
We have an Id for data from other workflows/services

What we need to do to use the Purchase Order Message is to “join” it with the Customer and Restaurant data.

Just like a database we can sometimes inline – denormalize – for performance, but this should be an exception, not the rule

Customer Upserted Message

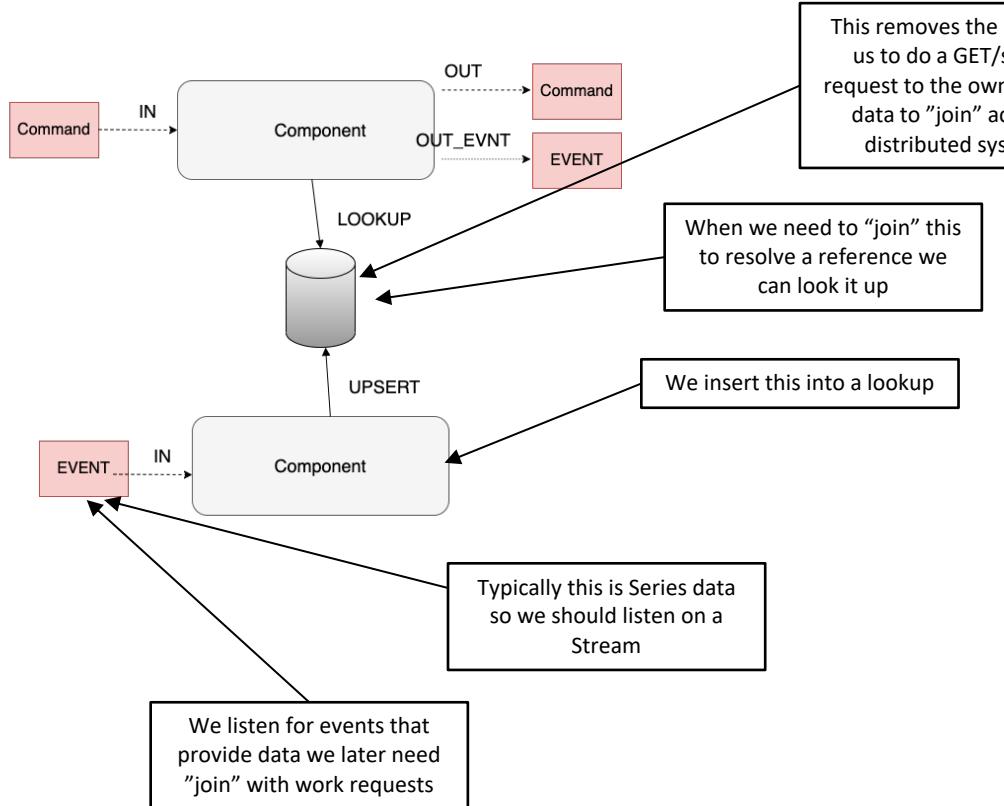
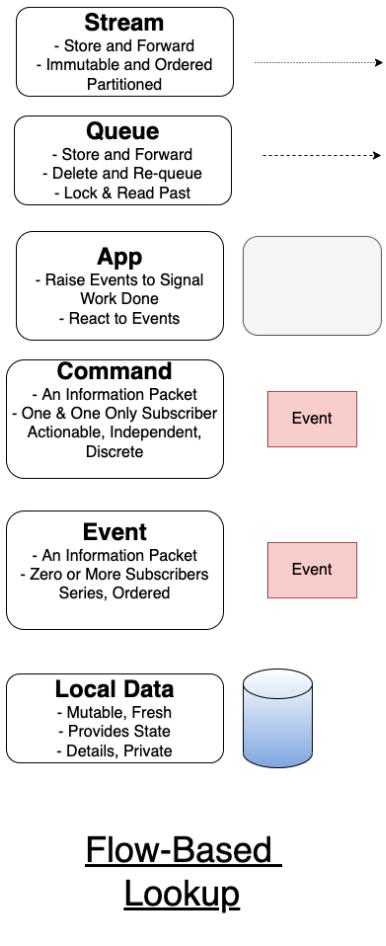
Customer First Name	Customer Last Name	Customer Post Code
Jo	Doe	SW17 3NJ

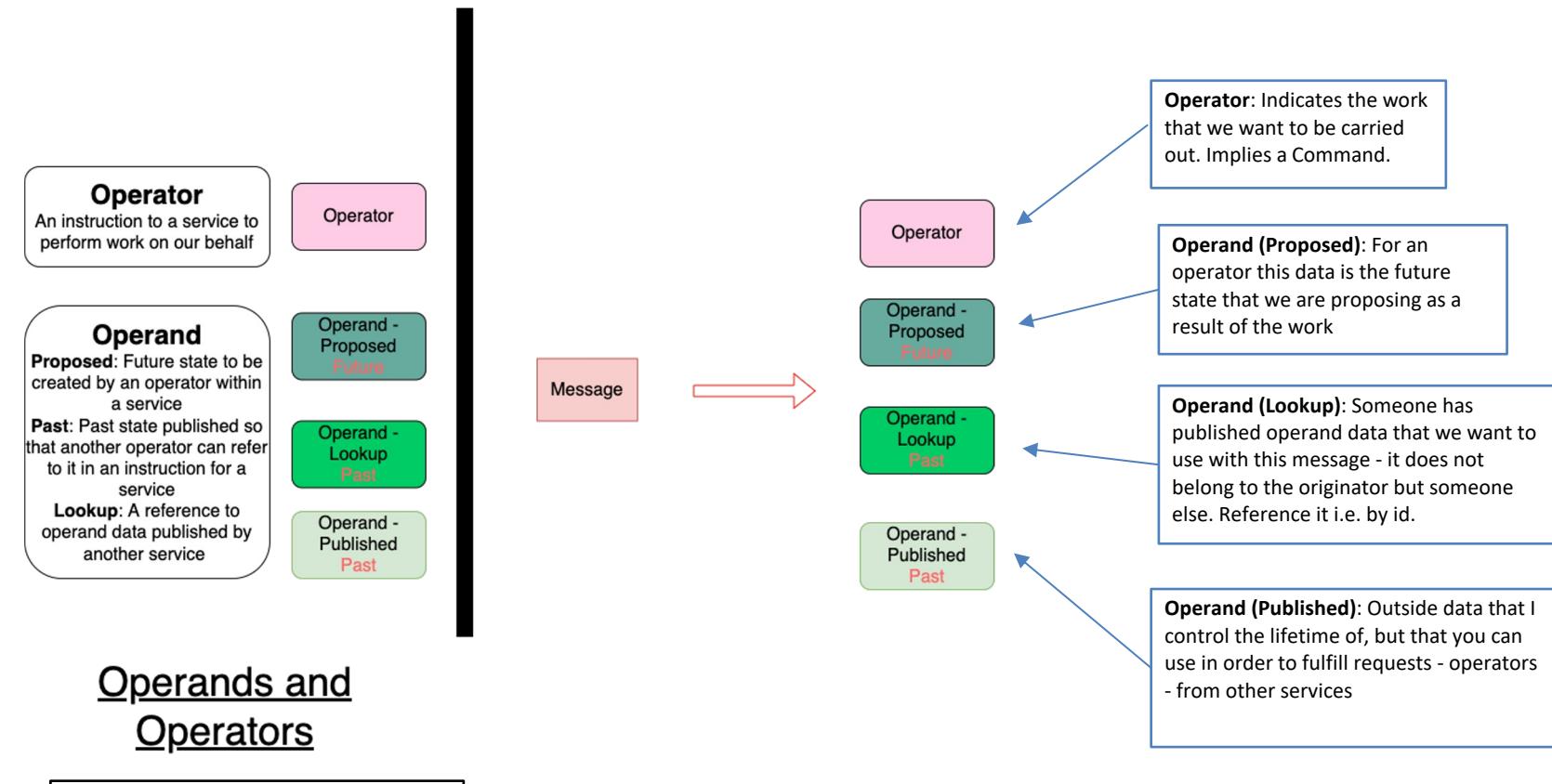
Restaurant Onboarded Message

Restaurant Name	Restaurant Post Code
Pizza 'R Us	SW17 5HK

The Customer workflow gives us Customer messages

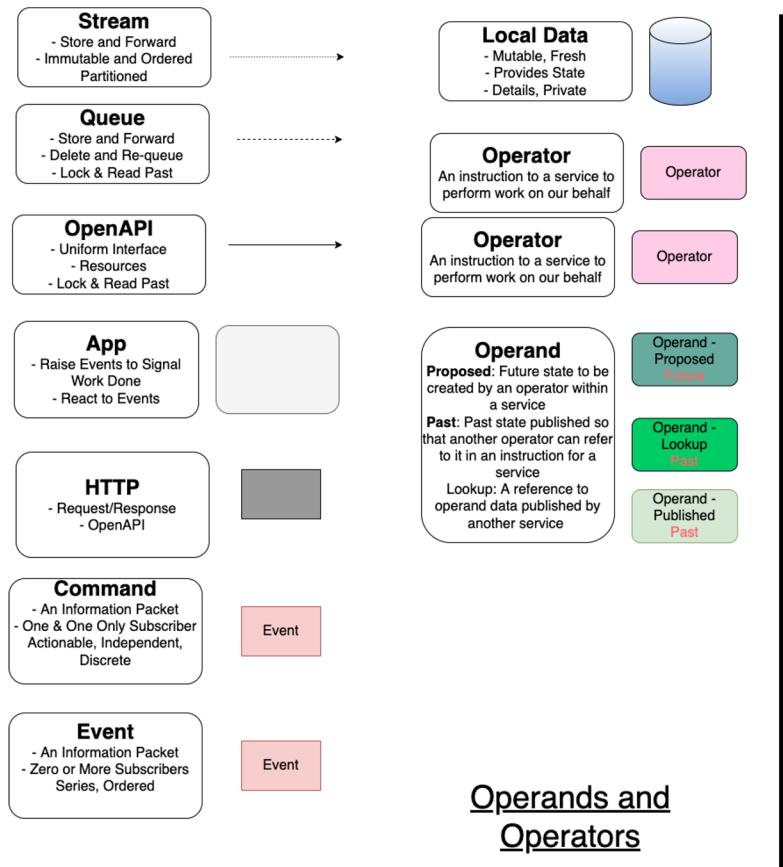
This Restaurant workflow gives us Restaurant messages



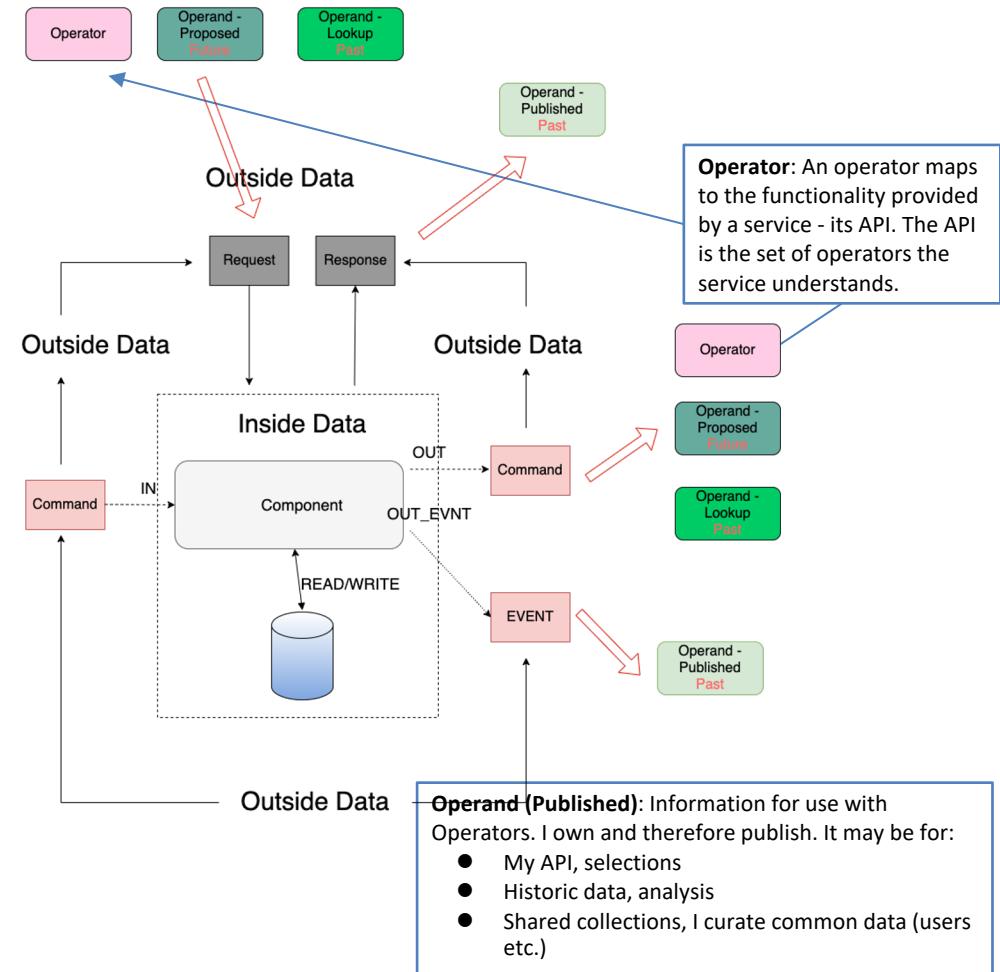


Operands and Operators

Data on the Outside vs Data on the Inside - Pat Helland



Data on the Outside vs Data on the Inside - Pat Helland



Stream
 - Store and Forward
 - Immutable and Ordered
 Partitioned



Queue
 - Store and Forward
 - Delete and Re-queue
 - Lock & Read Past



App
 - Raise Events to Signal Work Done
 - React to Events



Command
 - An Information Packet
 - One & One Only Subscriber
 Actionable, Independent, Discrete



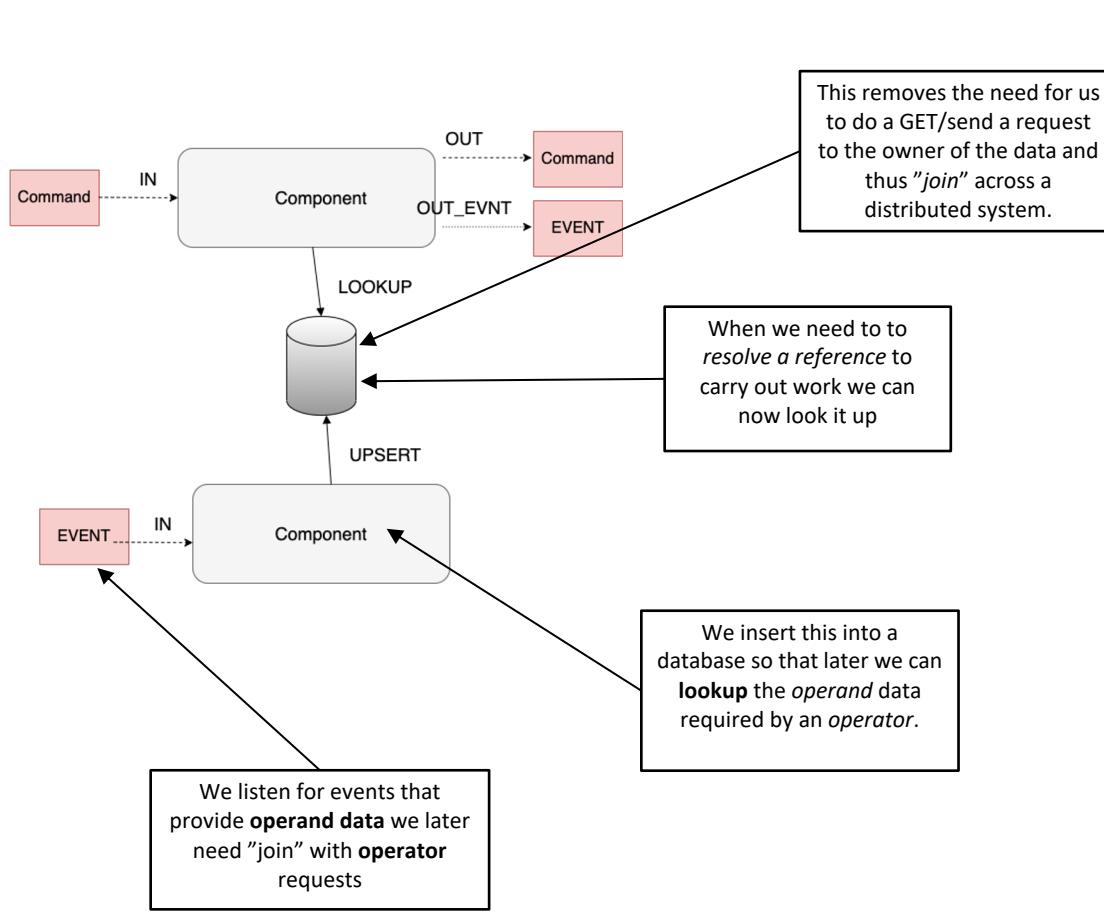
Event
 - An Information Packet
 - Zero or More Subscribers
 Series, Ordered

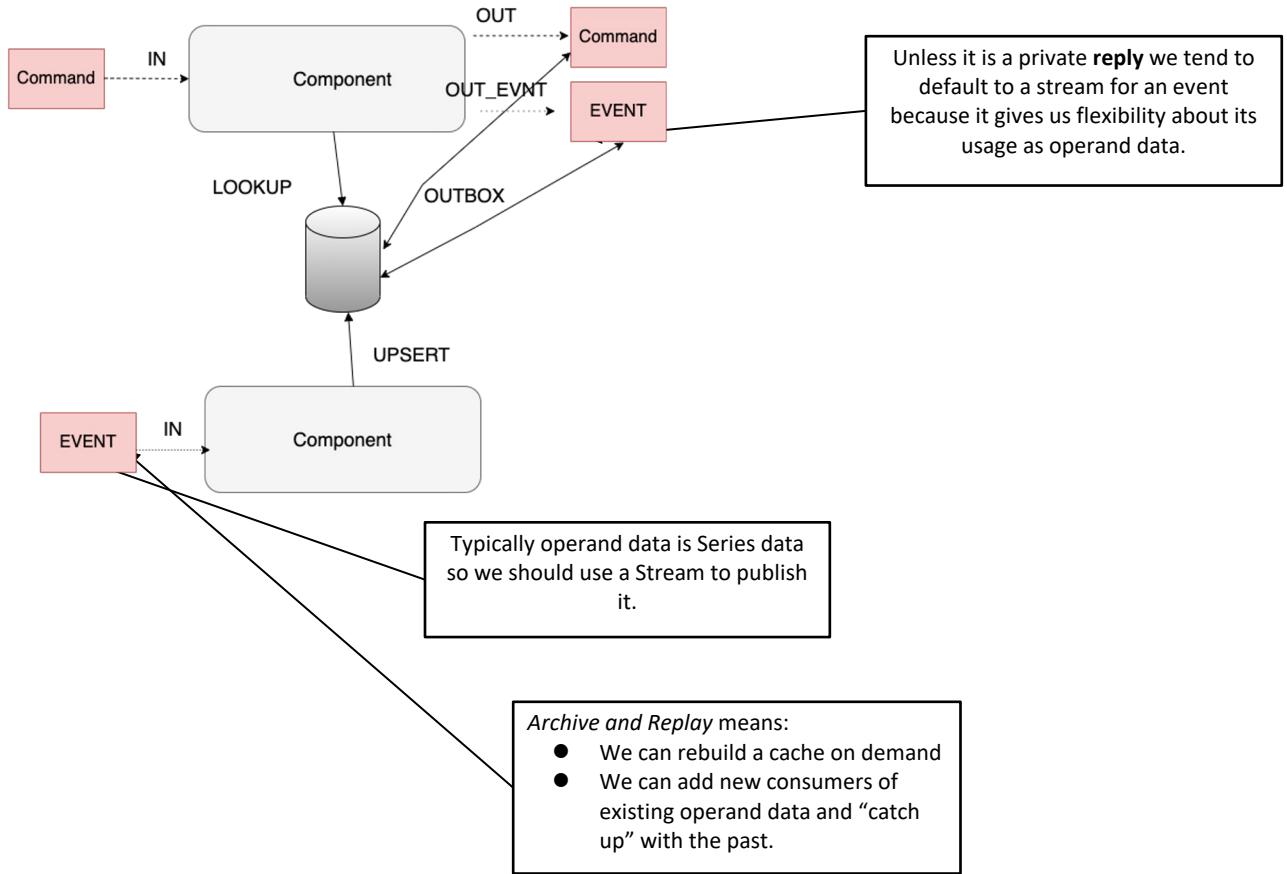
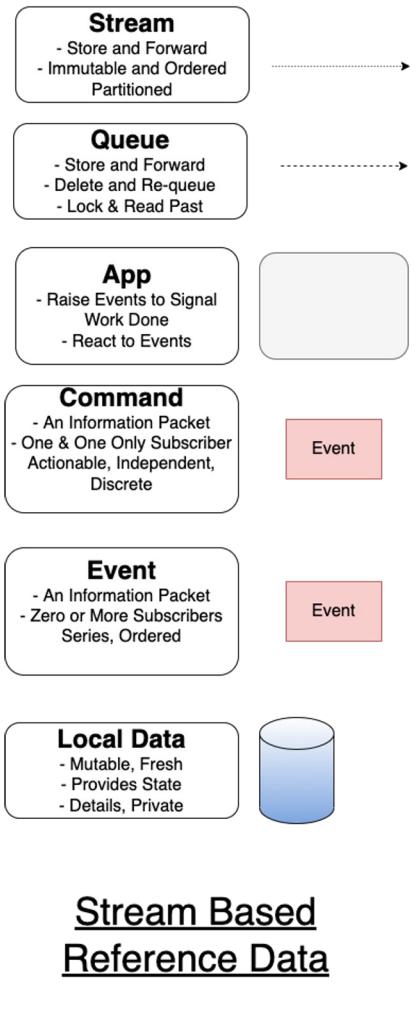


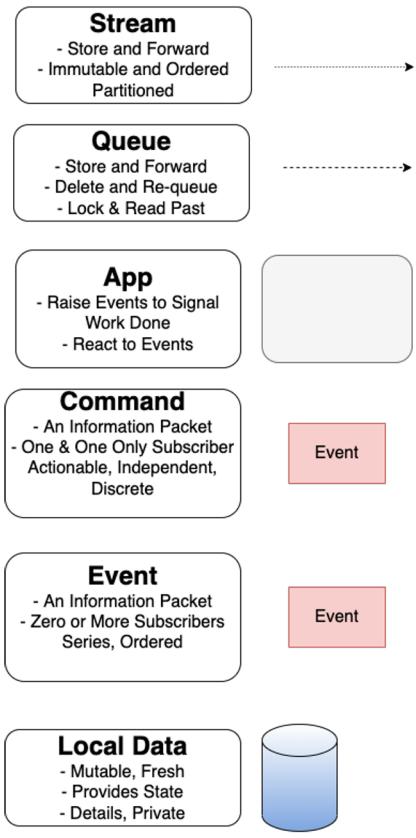
Local Data
 - Mutable, Fresh
 - Provides State
 - Details, Private



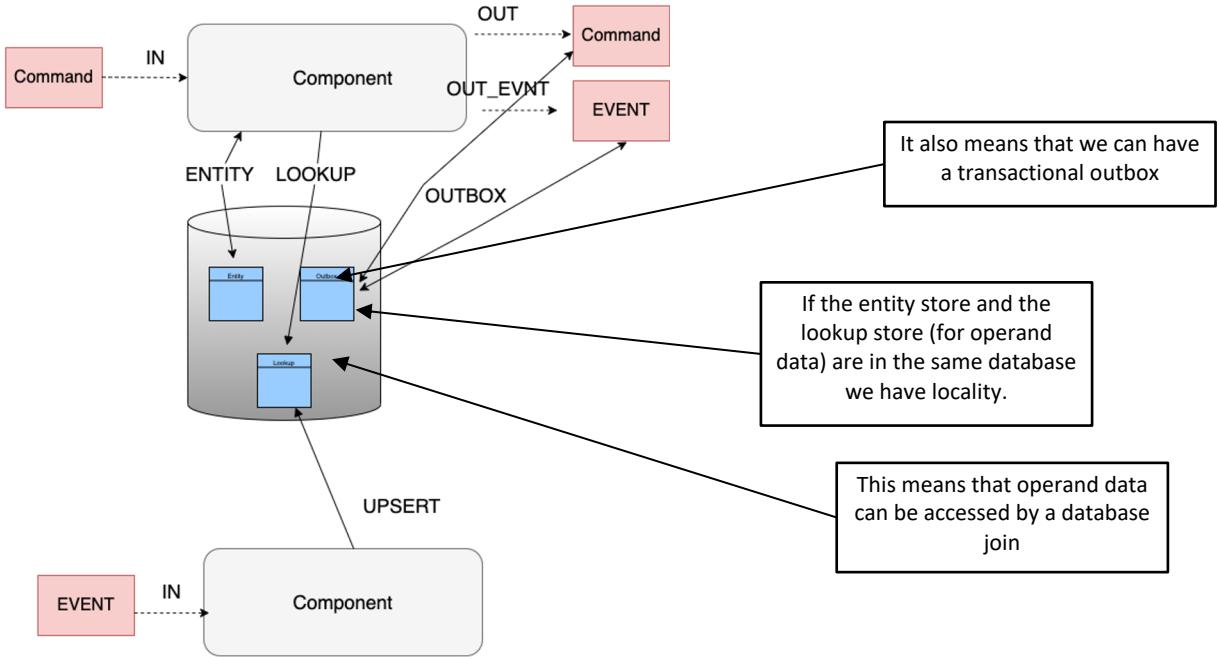
Flow-Based Lookup

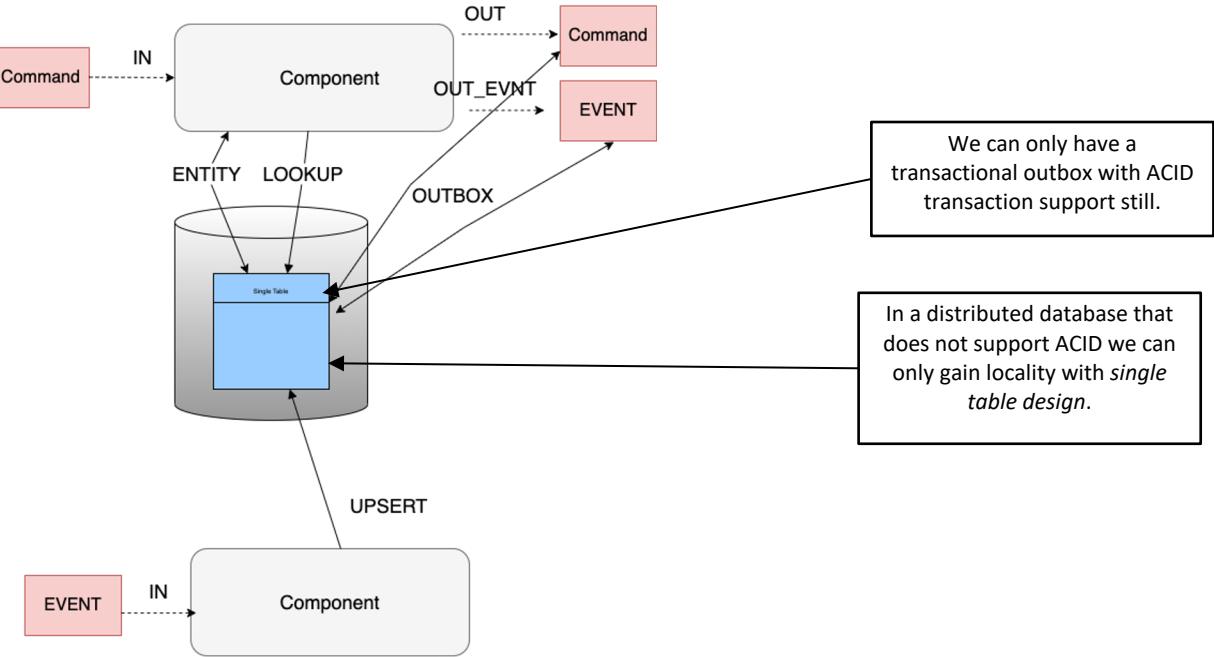
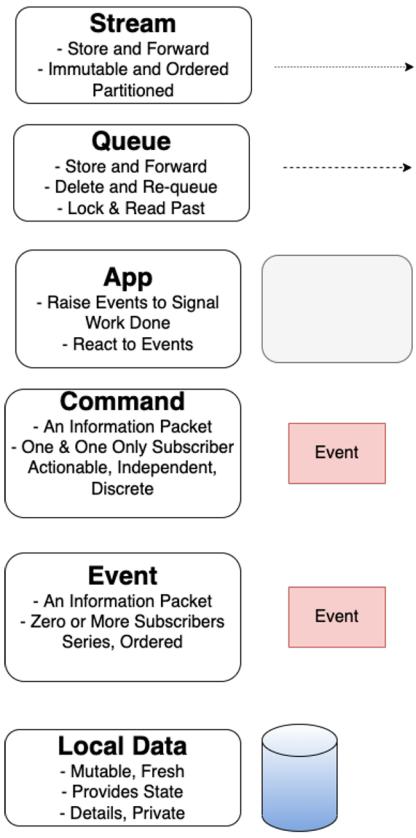






Locality





Locality No ACID

Stream

- Store and Forward
- Immutable and Ordered
- Partitioned



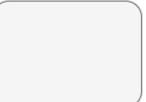
Queue

- Store and Forward
- Delete and Re-queue
- Lock & Read Past



App

- Raise Events to Signal Work Done
- React to Events



Command

- An Information Packet
- One & One Only Subscriber
- Actionable, Independent, Discrete



Event

- An Information Packet
- Zero or More Subscribers
- Series, Ordered

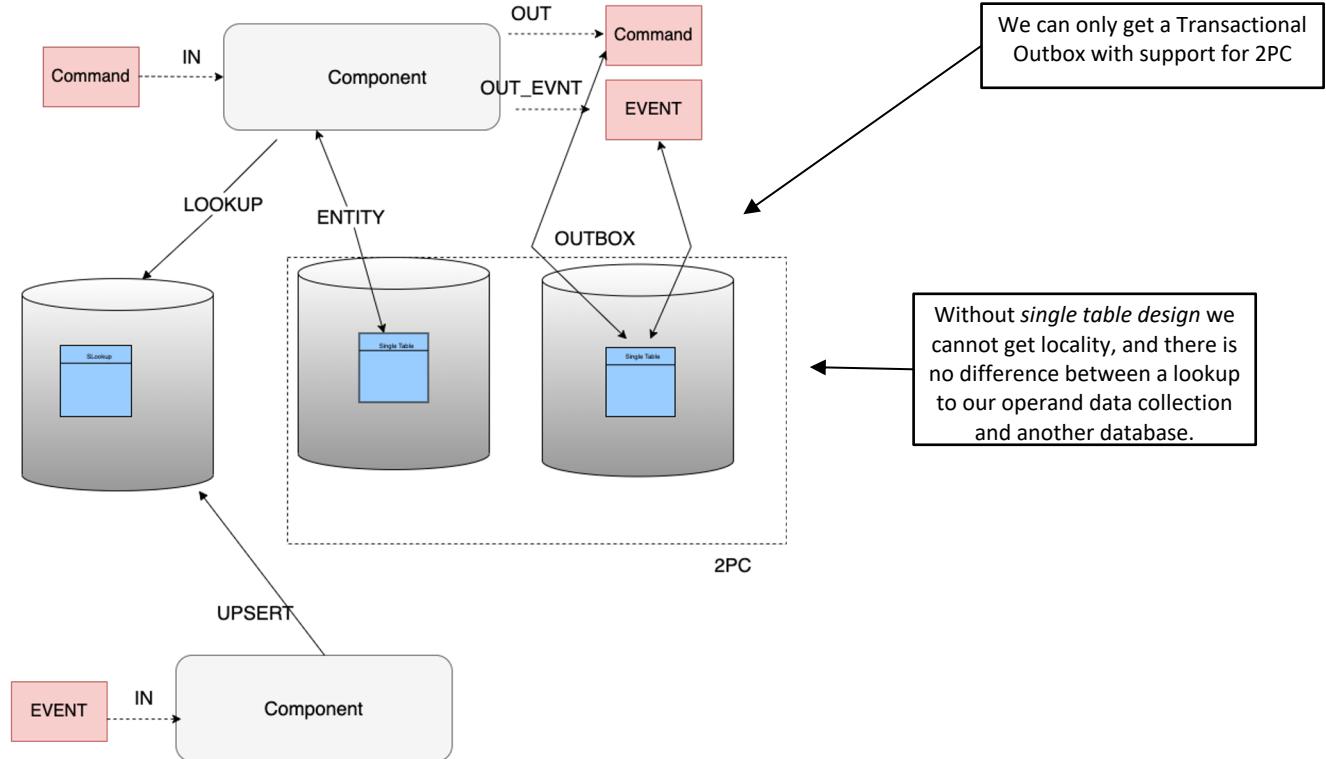


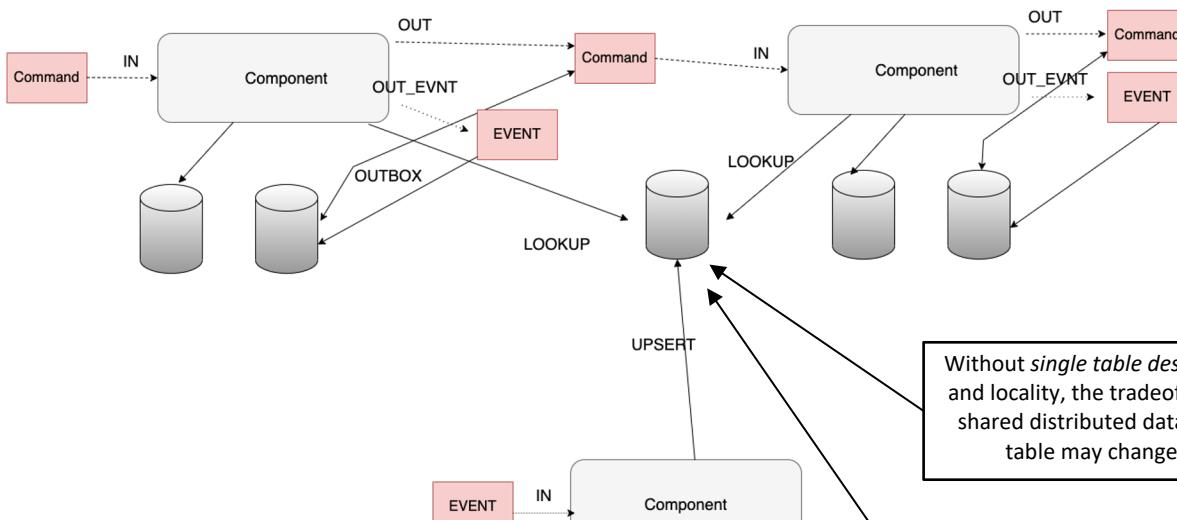
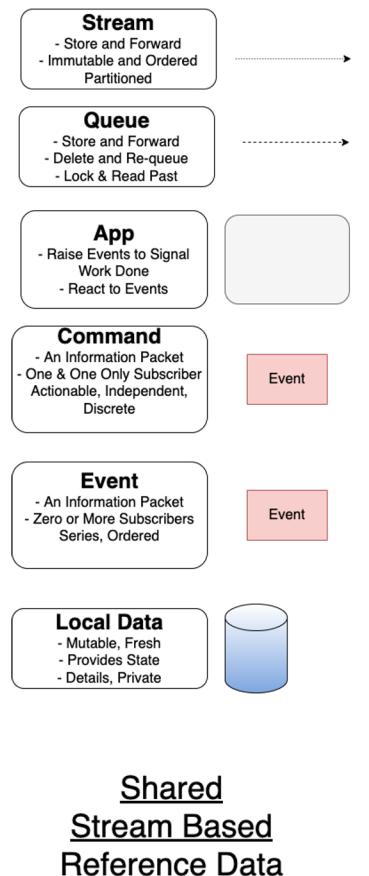
Local Data

- Mutable, Fresh
- Provides State
- Details, Private



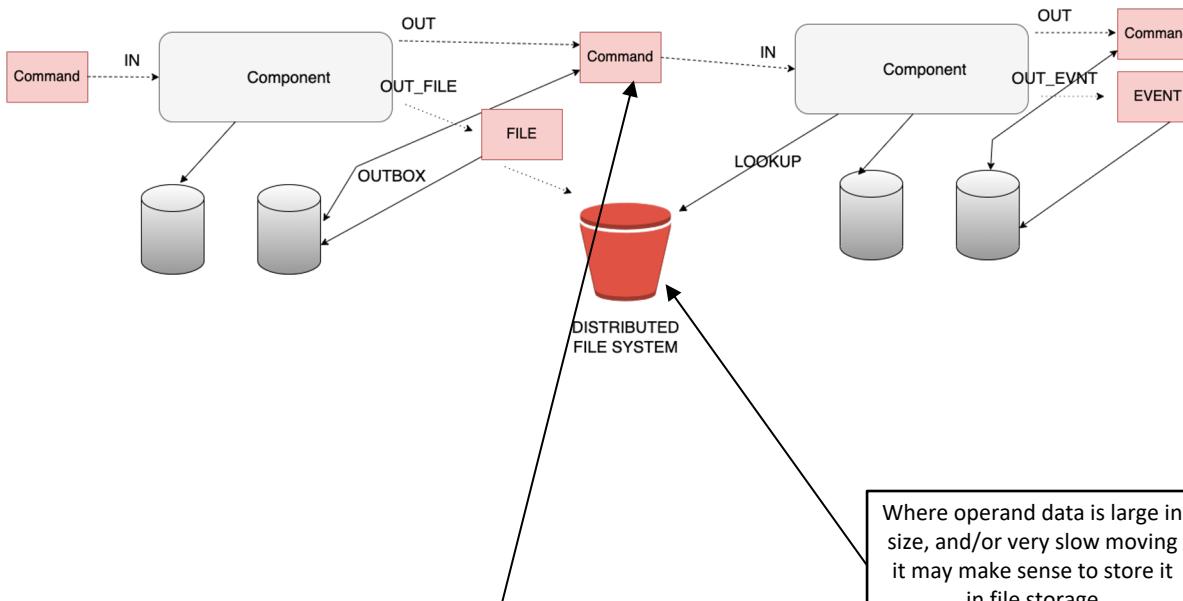
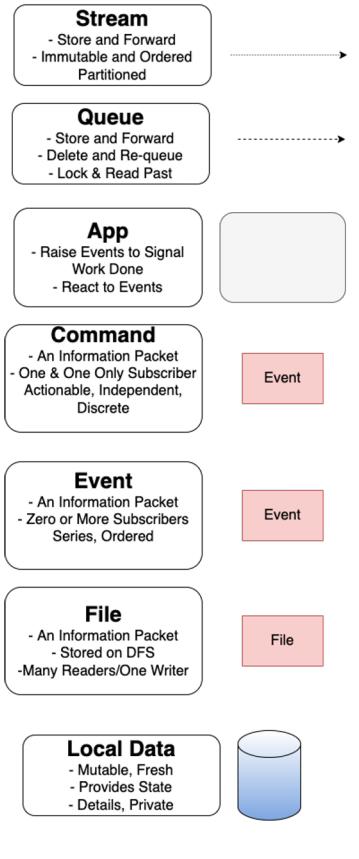
No Locality, No ACID





Without *single table design* we and locality, the tradeoffs to a shared distributed database table may change.

If we agree to fork for a different schema, we may be able to share with others to reduce costs of replication.

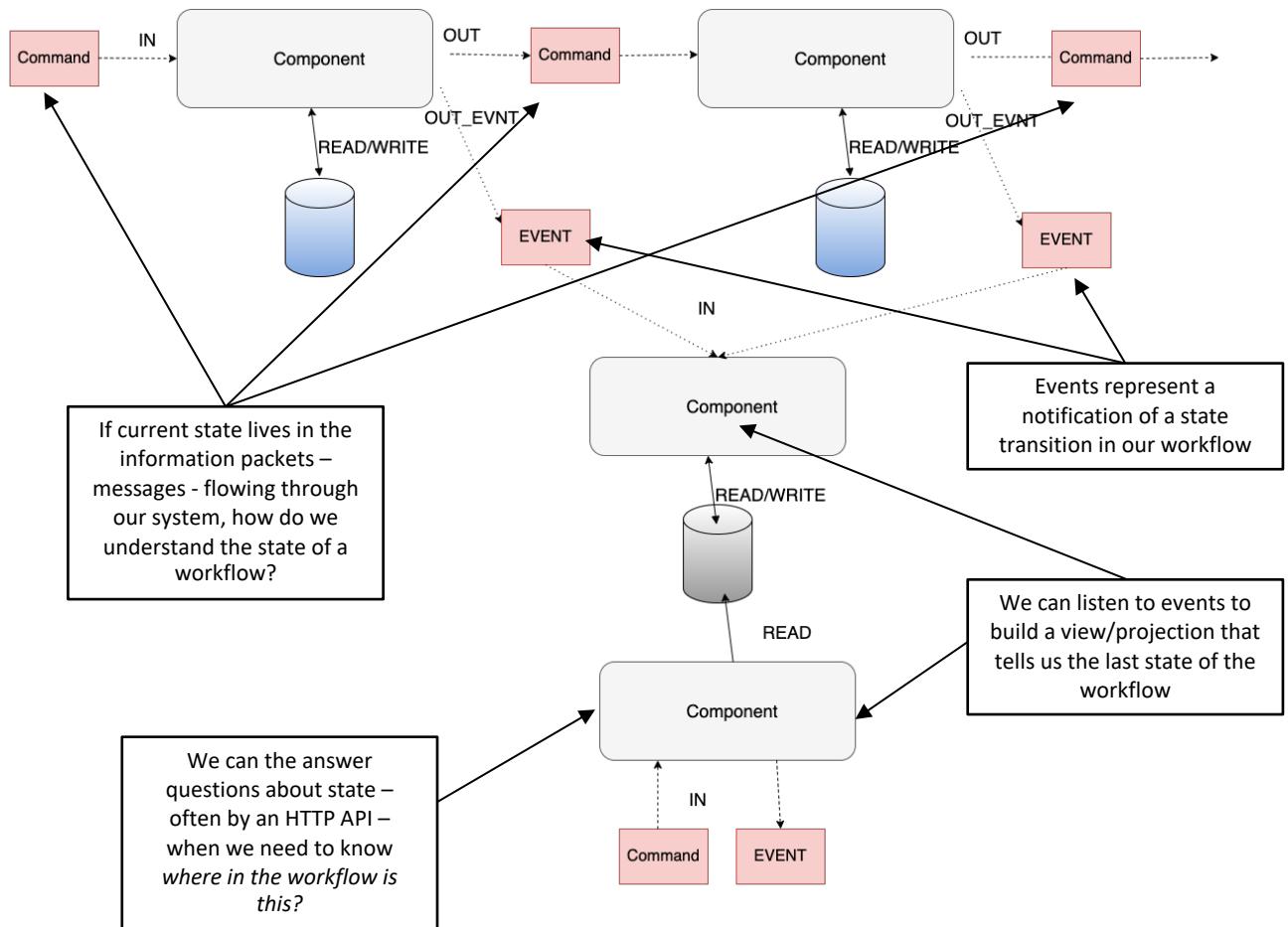
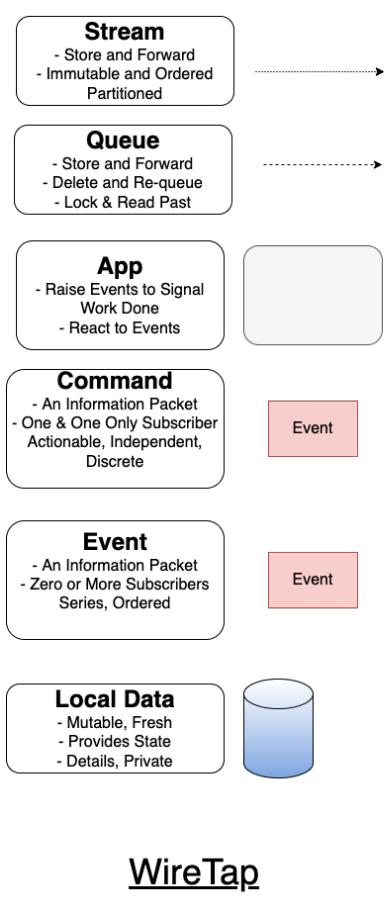


Either the command needs to provide a URI to the reference data it uses
Or an event needs to be provided to allow consumers to build a lookup table of URIs

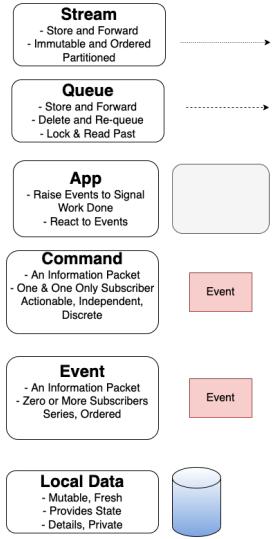
Where operand data is large in size, and/or very slow moving it may make sense to store it in file storage

Shared File Based Reference Data

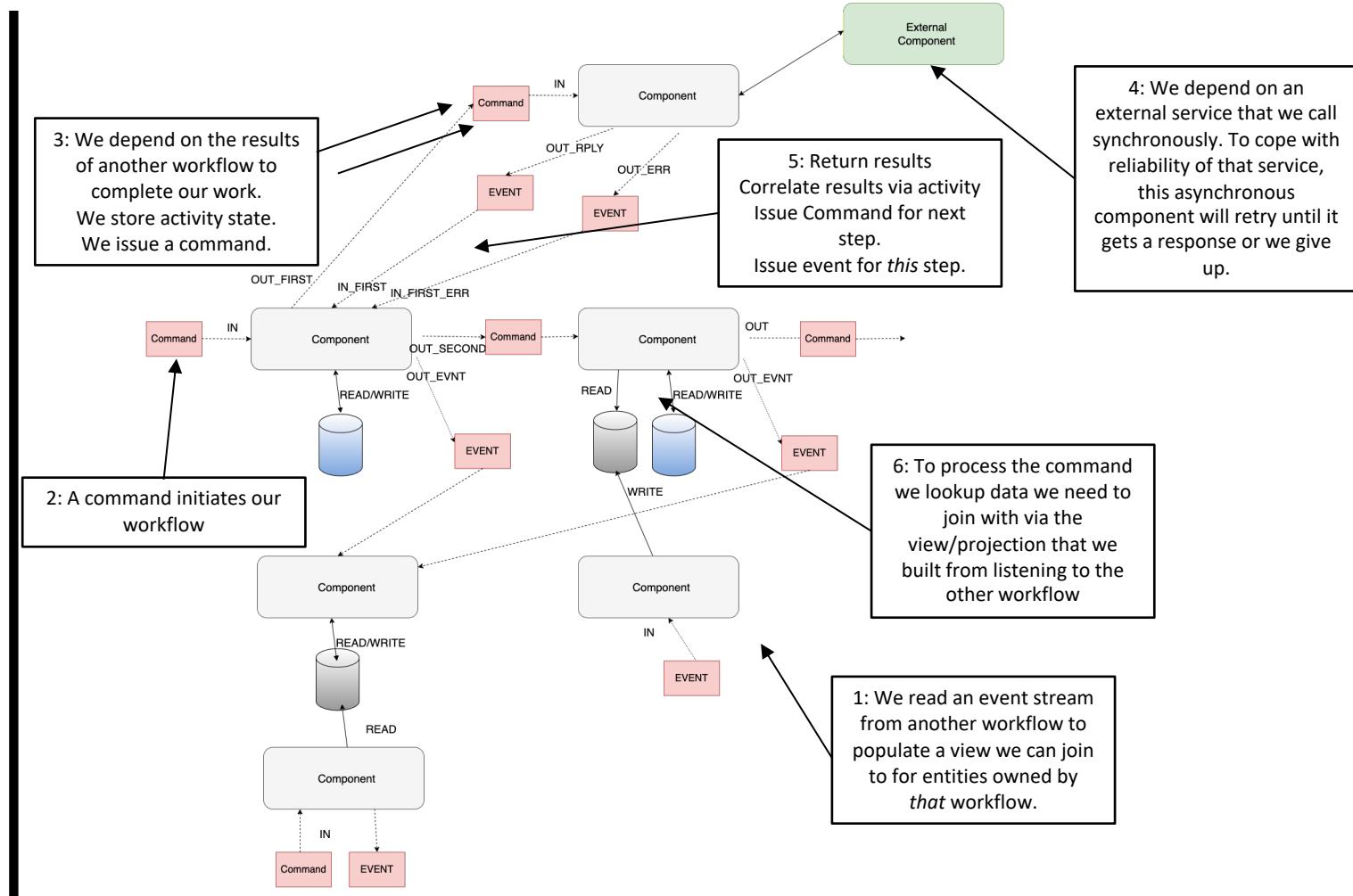
Pipeline Observability



Putting It Together



All Together

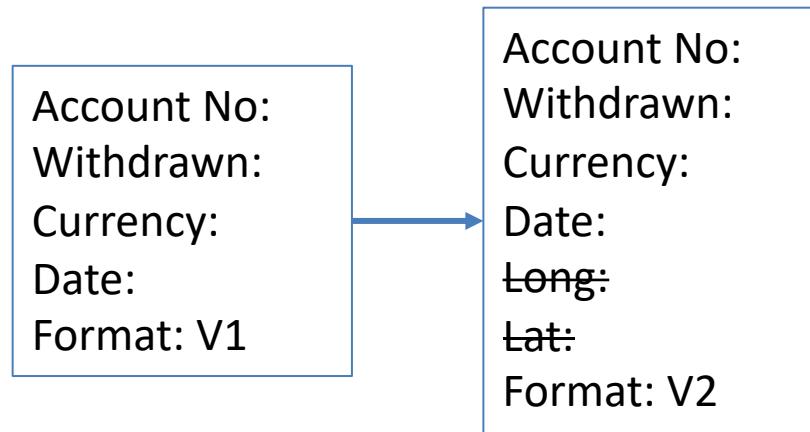


5 VERSIONING

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.

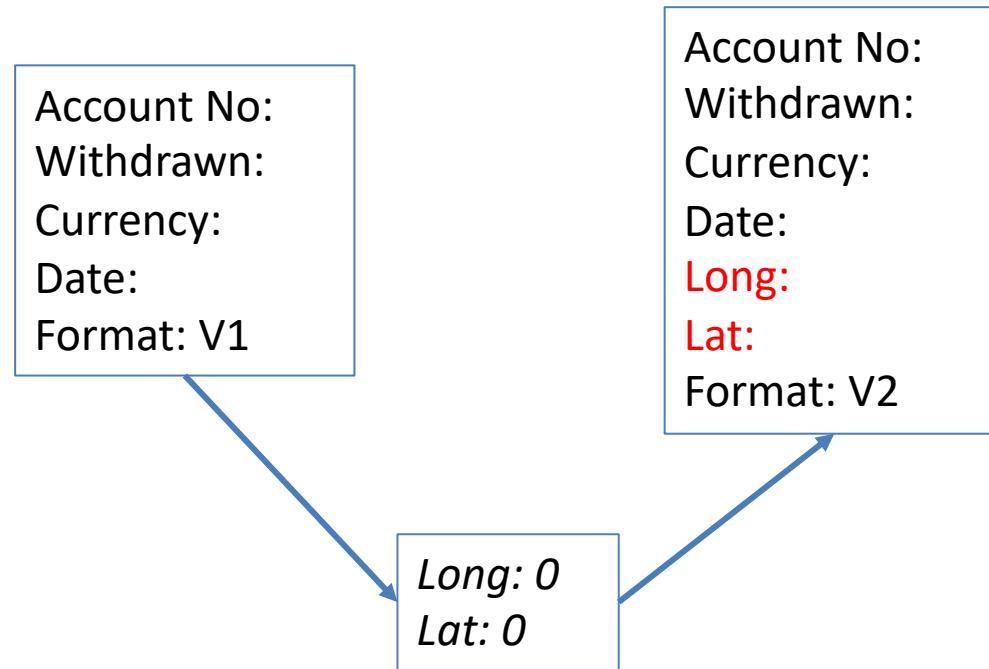
Robustness Principal or Postel's Law – Jon Postel RFC 1958

Tolerant Reader



Ignore New Fields

Tolerant Reader



Default Missing Fields

Breaking Change

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

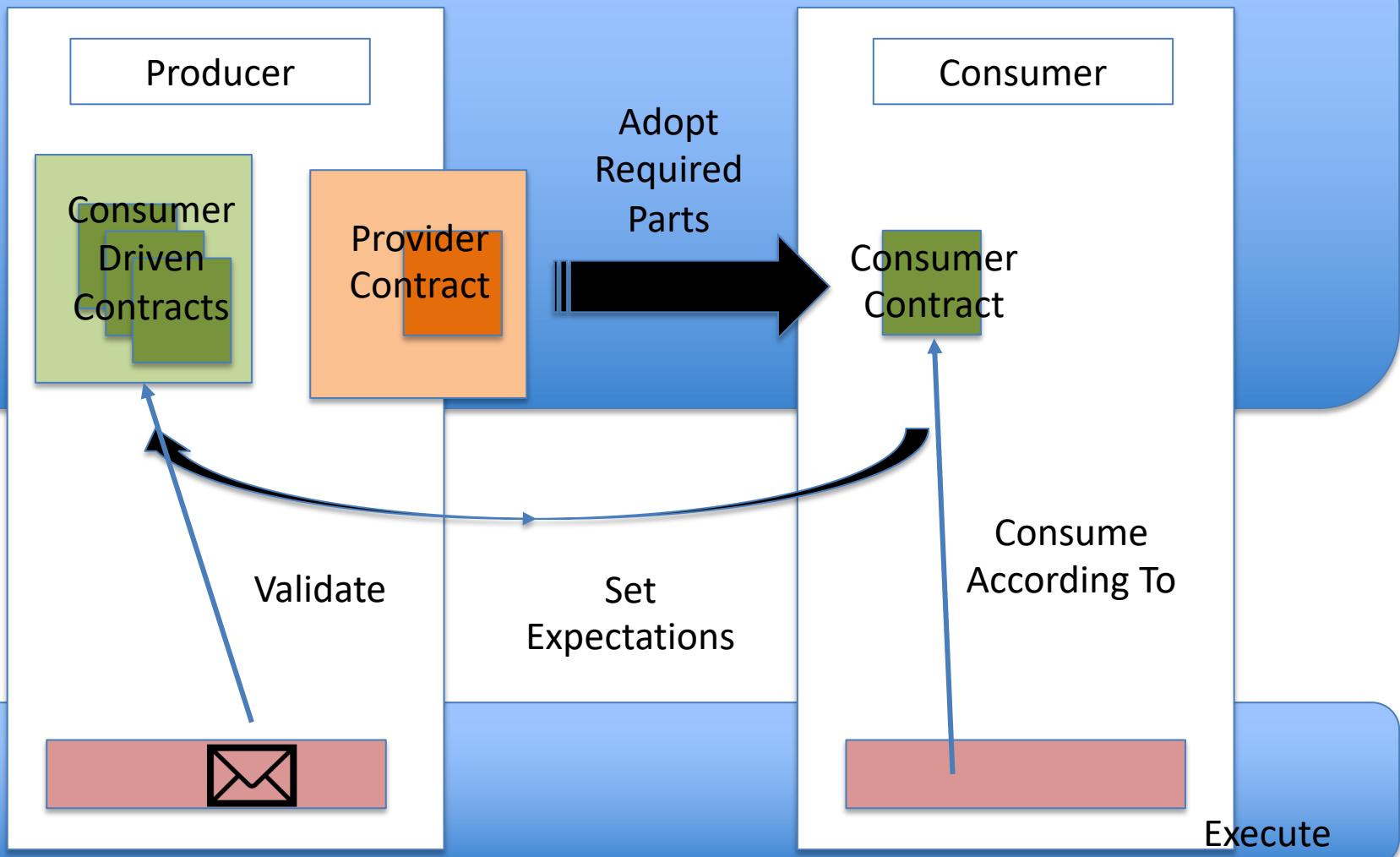
Account.Withdrawal.Event

Account No:
New Balance:
Date:
Format: V2

Account.NewBalance.Event

New Message

Consumer Driven Contracts



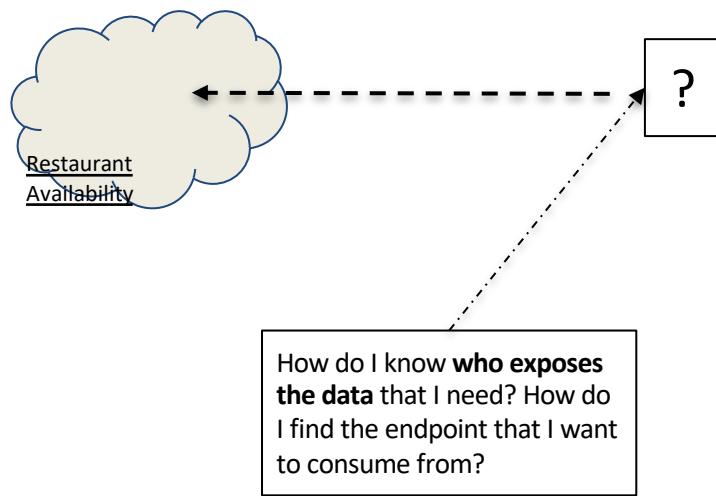
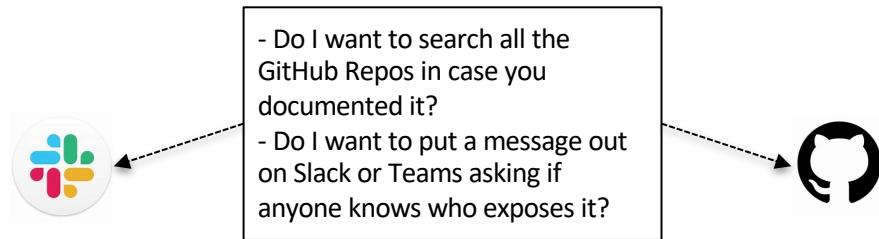
Standards & How to Describe Messaging

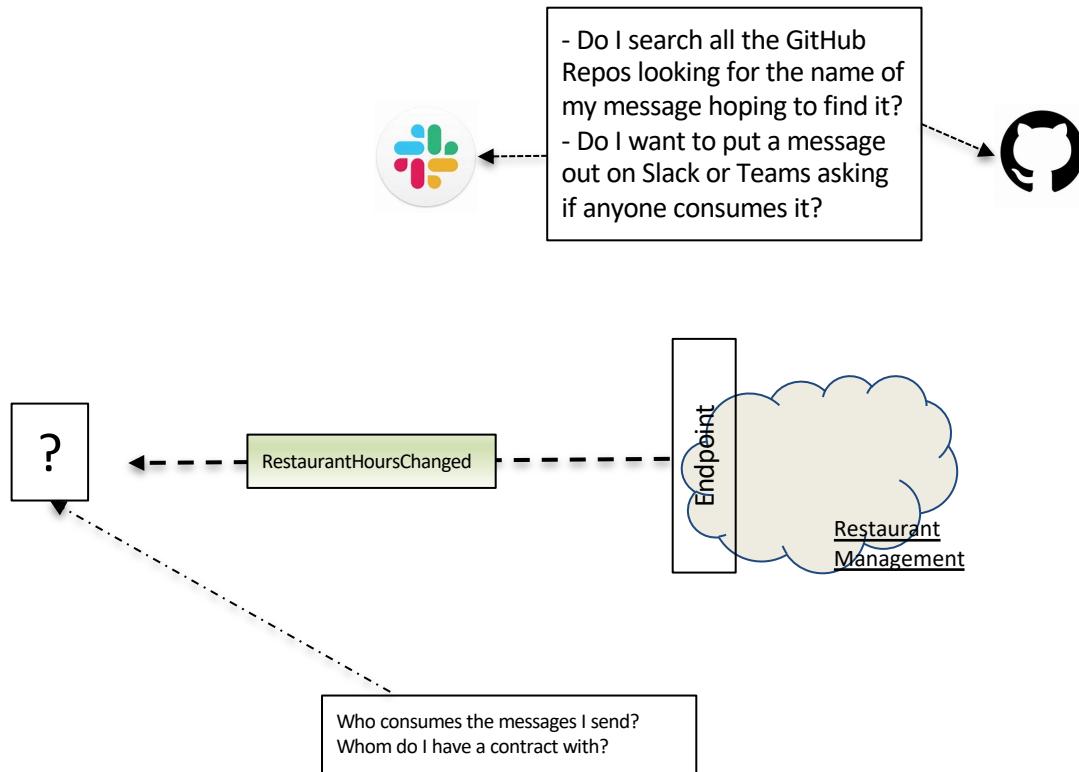
6 DOCUMENTATION

Endpoints are **places where messages are sent or received** (or both), and they define all the information required for the message exchange.

An *endpoint* describes in a standard-based way **where messages should be sent, how they should be sent, and what the messages should look like**.

<https://docs.microsoft.com/en-us/dotnet/framework/wcf/fundamental-concepts>

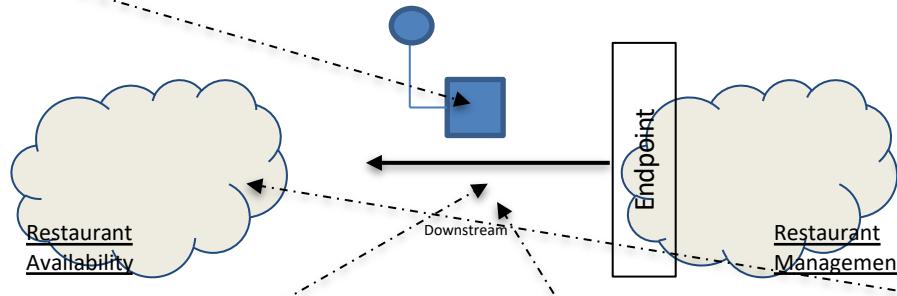




Asynchronous Endpoints

Our Asynchronous APIs need documenting just like any other API (HTTP etc).

We need to document the message, because it is the contract

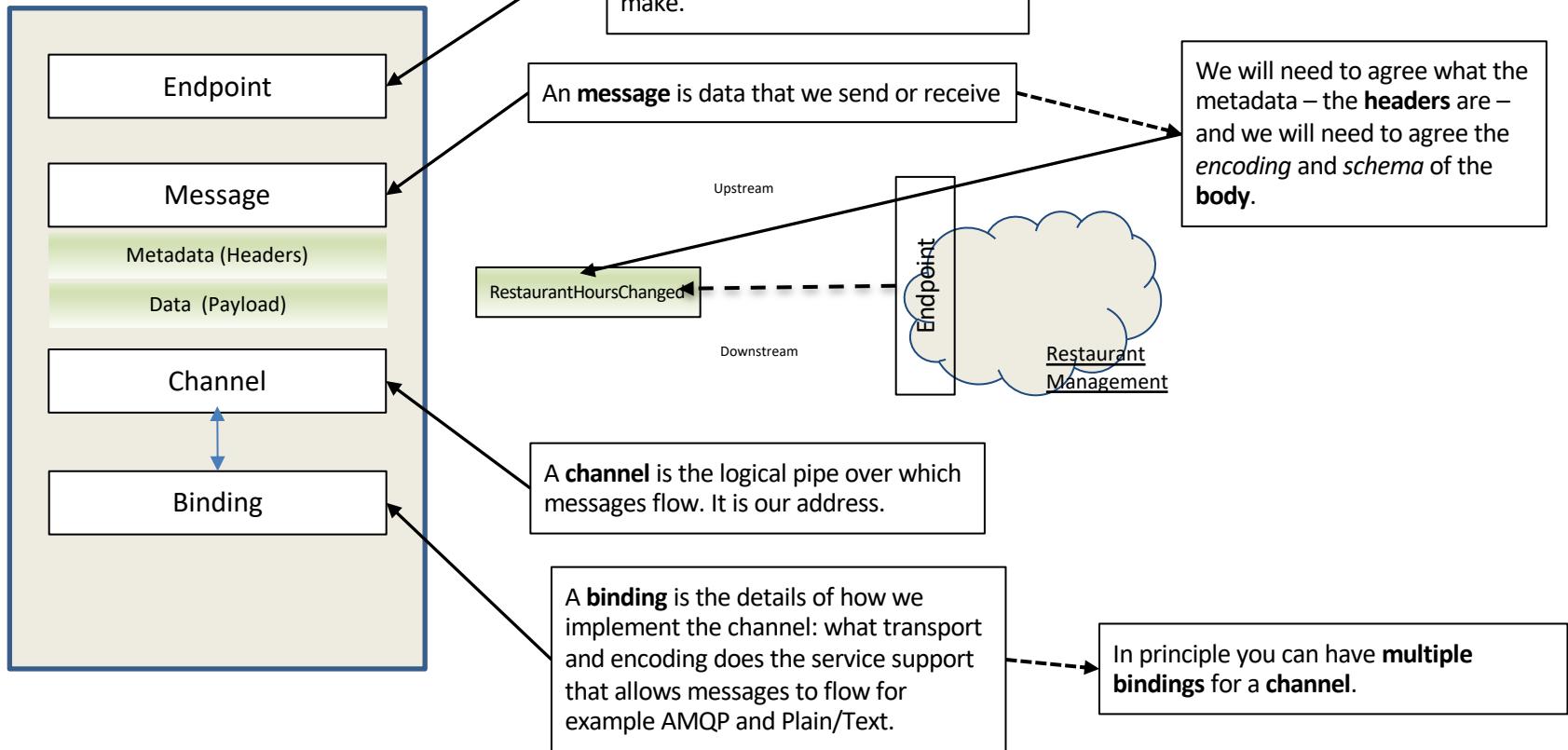


We need to document the channel so we know where the message is flowing

We need to document the protocol so we know how to send-receive

We need to document who sends and receives to understand flow

Endpoint – Documenting the Contract



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](#)

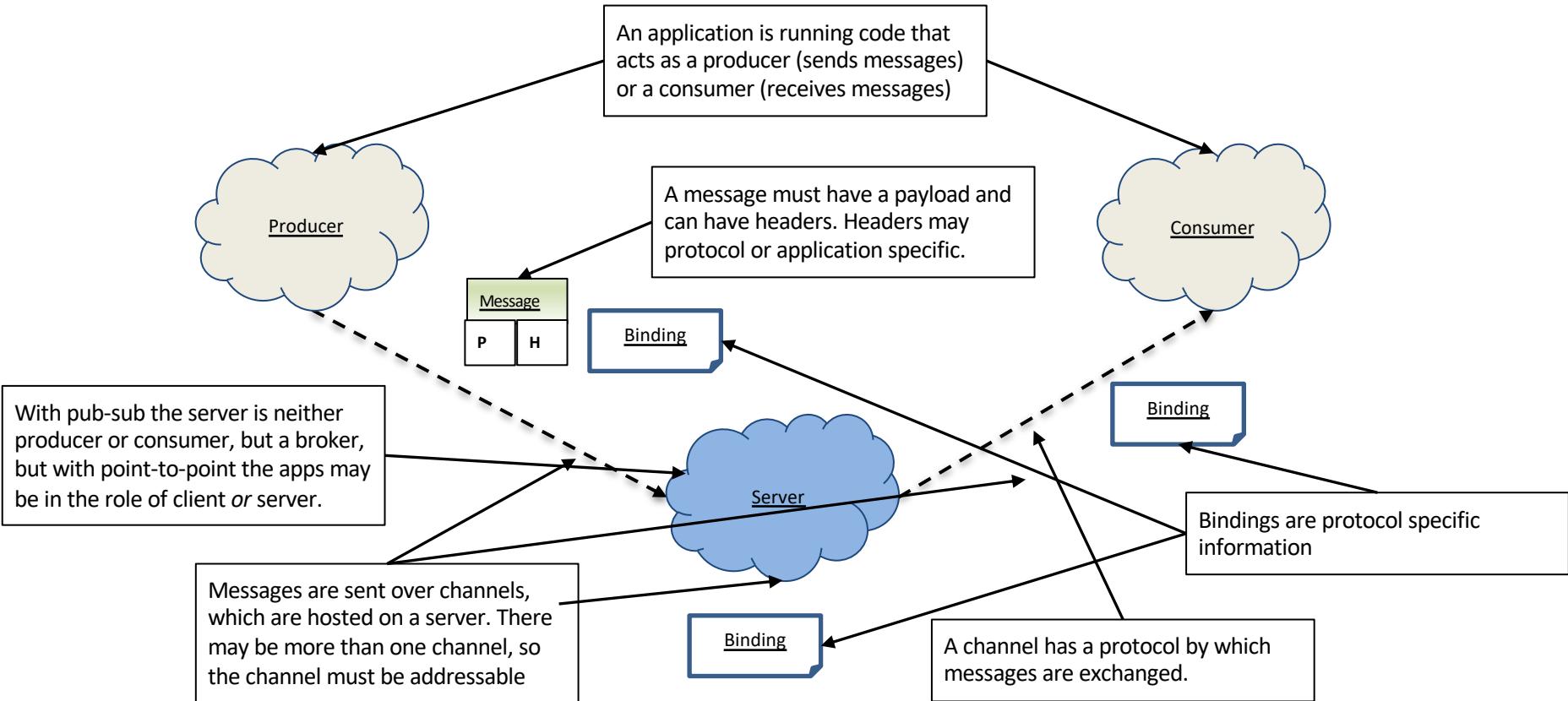
TSC
Members

And much more...

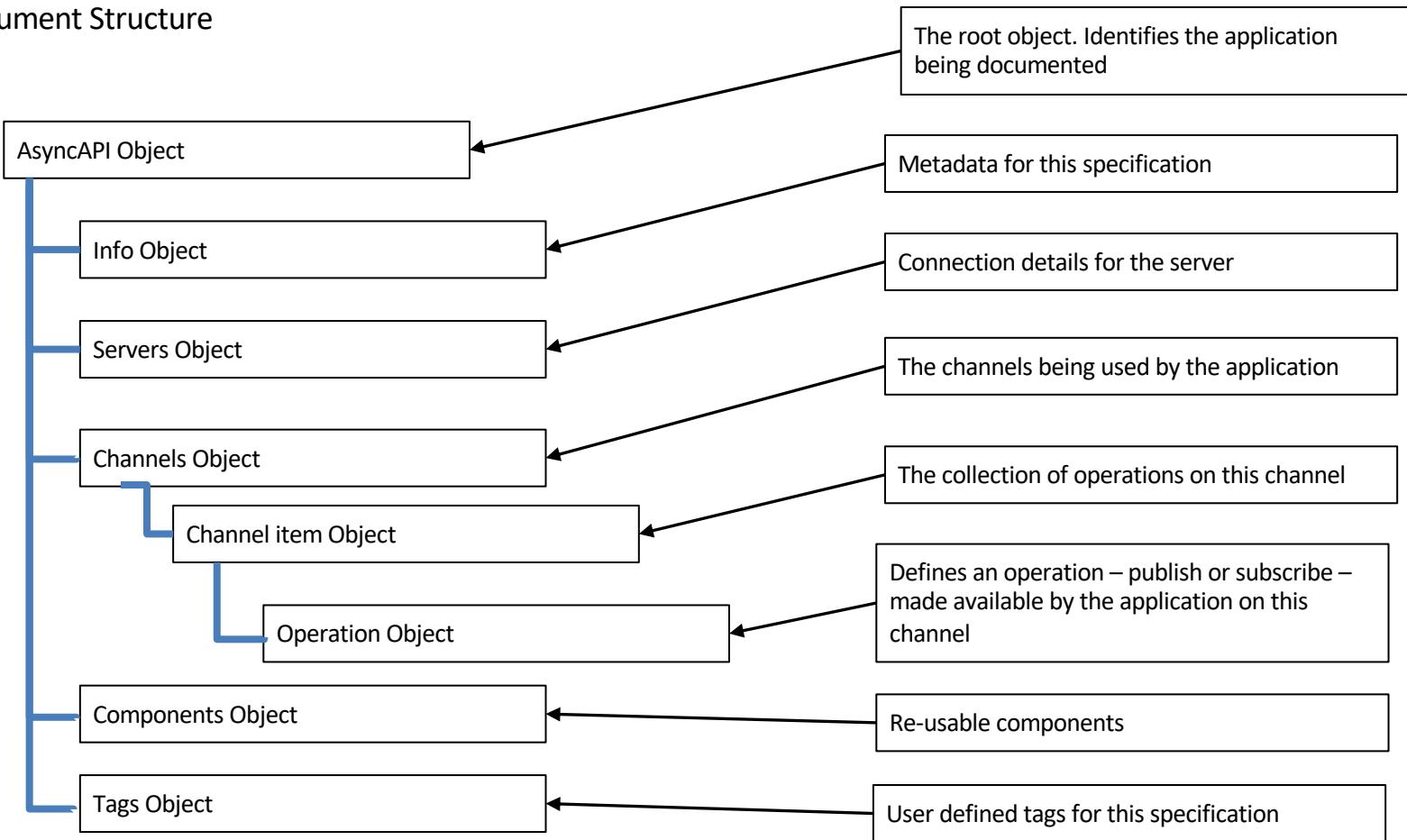
We have many different tools and welcome you to explore our ideas and propose new ideas to AsyncAPI.

[View GitHub Discussions](#)

AsyncAPI Elements



Document Structure



AsyncAPI Object

id: <https://github.com/brightercommand/greetings/>

We use a specification file for an app, which is a producer or consumer, and identify them by id

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:

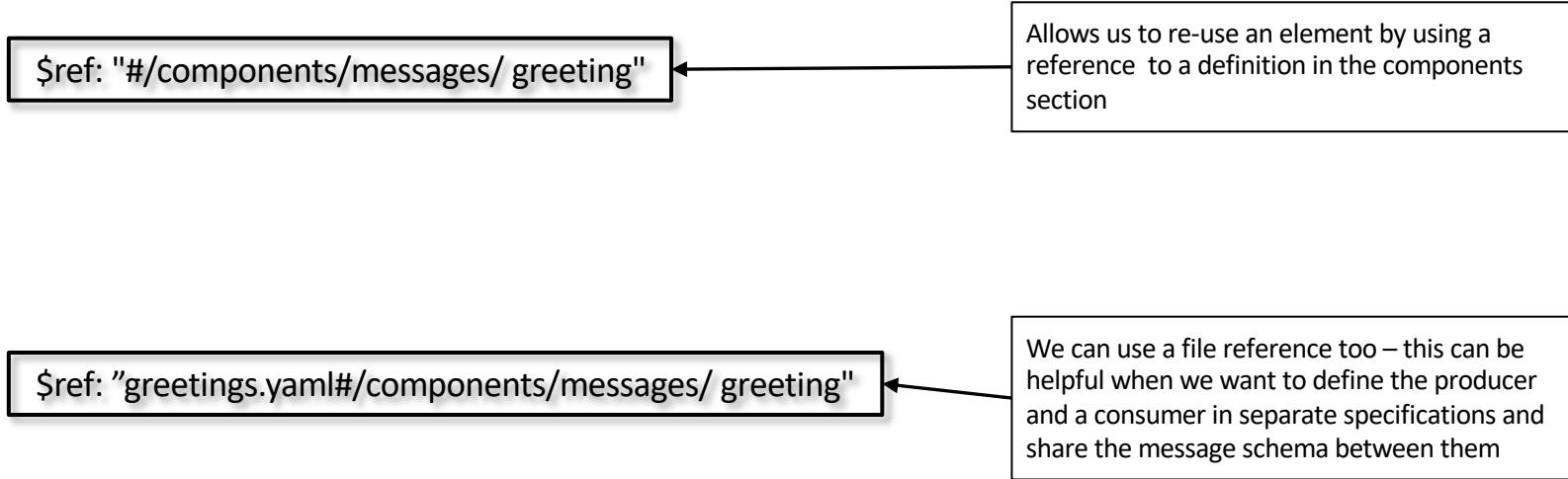
url: localhost:9092

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



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  
...
```

Information

```

From localStorage
1  asyncapi: '2.0.0'
2  id: "https://github.com/brightercommand/greetings-sender/"
3  info:
4    contact:
5      name: Paramore Brighter
6      url: https://goparamore.io/support
7      email: support@goparamore.io
8    license:
9      name: Apache 2.0
10     url: https://www.apache.org/licenses/LICENSE-2.0.html
11     description: Demonstrates sending a greeting over a messaging
12       transport.
13     title: Brighter Sample App
14     version: 1.0.0
15     tags:
16       - name: brighter examples
17   servers:
18     localhost:
19       url: localhost:9092
20       protocol: kafka
21
22   channels:
23     greeting:
24       description: A channel for sending out greeting messages
25       subscribe:
26         description: This service lets you send the 'Hello World'
27         greeting to another service.
28         operationId: sendMessage
29         message:
30           $ref: '#/components/messages/greeting'
31
32   components:
33     messages:
34       greeting:
35         name: greeting
36         title: A salutation
37         summary: This is how we send you a salutation
38         contentType: application/json
39         traits:
40           - $ref: '#/components/messageTraits/commonHeaders'
41         payload:
42           $ref: "#/components/schemas/greetingContent"
43
44   schemas:

```

PROBLEMS 0

VALID NOT LATEST YAML

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

! brighter-greetings-sender.yaml × Extension: AsyncAPI Preview ! brighter-greeting-rec ...

```

! brighter-greetings-sender.yaml > YAML > {} info > {} contact > url
  1  asyncapi: '2.0.0'
  2  id: "https://github.com/brightercommand/greetings-sender/"
  3  info:
  4    contact:
  5      name: Paramore Brighter
  6      url: https://goparamore.io/support
  7      email: support@goparamore.io
  8    license:
  9      name: Apache 2.0
 10     url: https://www.apache.org/licenses/LICENSE-2.0.html
 11   description: Demonstrates sending a greeting over a messaging transpo
 12   title: Brighter Sample App
 13   version: 1.0.0
 14   tags:
 15     - name: brighter examples
 16
 17   servers:
 18     localhost:
 19       url: localhost:9092
 20       protocol: kafka
 21
 22   channels:
 23     greeting:
 24       description: A channel for sending out greeting messages
 25       subscribe:
 26         description: This service lets you send the 'Hello World' greeting
 27         operationId: sendMessage
 28         message:
 29           $ref: '#/components/messages/greeting'
 30
 31   components:
 32     messages:
 33       greeting:
 34         name: greeting
 35         title: A salutation
 36         summary: This is how we send you a salutation
 37         contentType: application/json
 38         traits:
 39           - $ref: '#/components/messageTraits/commonHeaders'
 40         payload:
 41           $ref: "#/components/schemas/greetingContent"
 42
 43     schemas:
 44       greetingContent:

```

! AsyncAPI - brighter-greetings-sender.yaml × ...

Brighter Sample App 1.0.0

APACHE 2.0

Demonstrates sending a greeting over a messaging transport.

Contact link: [PARAMORE BRIGHTER](#) Contact email: SUPPORT@GOPARAMORE.IO

Servers

localhost:9092 **KAFKA**

Operations

SUB **greeting**

A channel for sending out greeting messages

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

Accepts the following message:

A salutation **greeting**

This is how we send you a salutation

main 0 1↑ CodeTogether 0 △ 0 ⏪ 2 ⏴ Live Share docker-desktop Cloud Code default minikube -- NORMAL -- AsyncAPI Preview Spell 🔍 ↻

Overview	>
Getting Started	>
Local Development	>
Core Features	▼
Software Catalog	
Overview	
The Life of an Entity	
Catalog Configuration	
System Model	
YAML File Format	
Entity References	
Well-known Annotations	
Well-known Relations	

spec.type [required]

The type of the API definition as a string, e.g. `openapi`. This field is required.

The software catalog accepts any type value, but an organization should take great care to establish a proper taxonomy for these. Tools including Backstage itself may read this field and behave differently depending on its value. For example, an OpenAPI type API may be displayed using an OpenAPI viewer tooling in the Backstage interface.

The current set of well-known and common values for this field is:

- `openapi` - An API definition in YAML or JSON format based on the [OpenAPI](#) version 2 or version 3 spec.
- `svncapi` - An API definition based on the [AsyncAPI](#) spec
- `graphql` - An API definition based on [GraphQL schemas](#) for consuming [GraphQL](#) based APIs.
- `grpc` - An API definition based on [Protocol Buffers](#) to use with [gRPC](#).

Contents

[Overall Shape Of An Entity](#)

[Substitutions In The Descriptor Format](#)

[Common to All Kinds: The Envelope](#)

`apiVersion` and `kind` [required]

`metadata` [required]

`spec` [varies]

[Common to All Kinds: The Metadata](#)

`name` [required]

`namespace` [optional]

`title` [optional]

`description` [optional]

`labels` [optional]

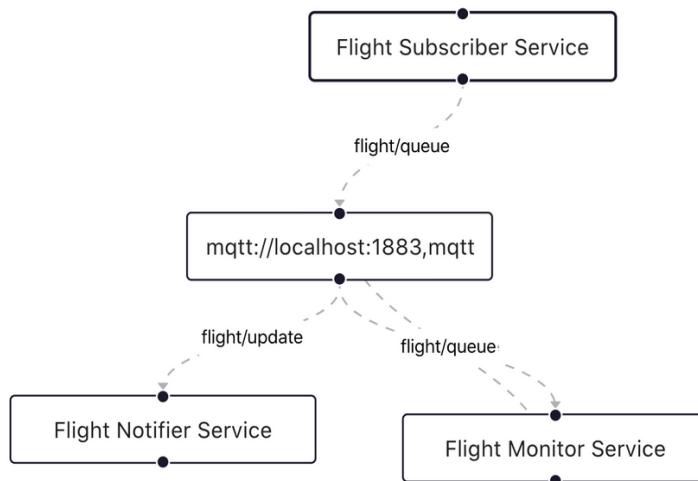
`annotations` [optional]

`tags` [optional]

`links` [optional]

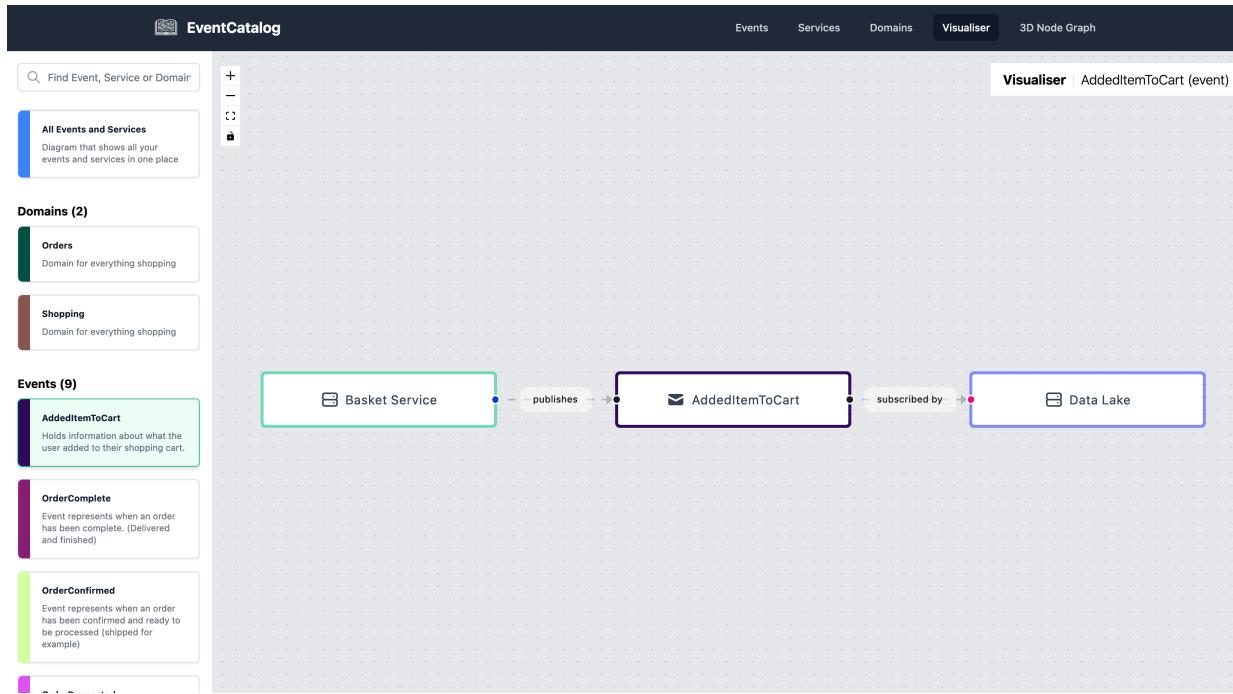
Cupid

React Flow Nodes



<https://github.com/asyncapi/cupid>

Event Catalog



<https://github.com/boynley123/eventcatalog>



cloudevents

A specification for describing event data in a common way

Why CloudEvents?

Events are everywhere, yet event publishers tend to describe events differently.

Consistency

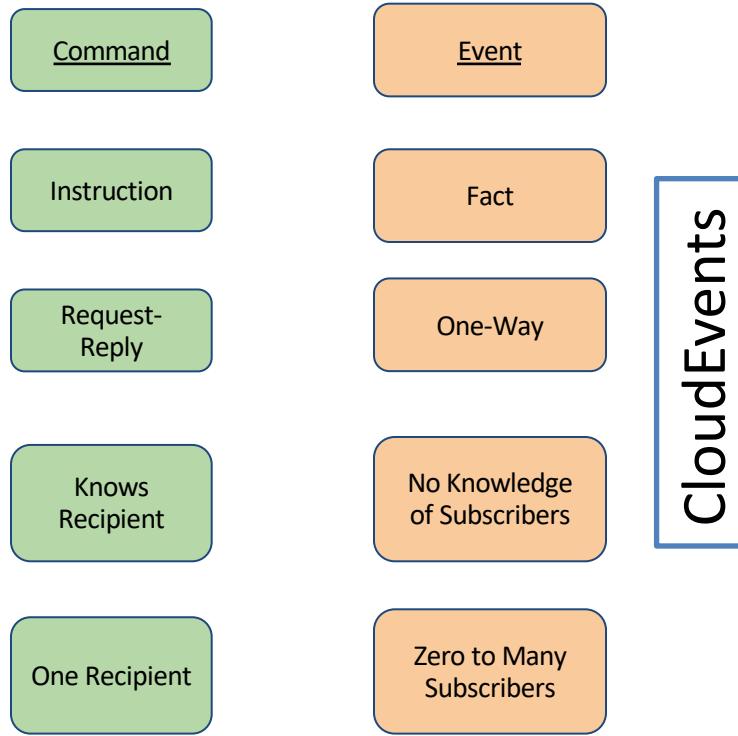
The lack of a common way of describing events means developers have to write new event handling logic for each event source.

Accessibility

No common event format means no common libraries, tooling, and infrastructure for delivering event data across environments. CloudEvents provides SDKs for [Go](#), [JavaScript](#), [Java](#),

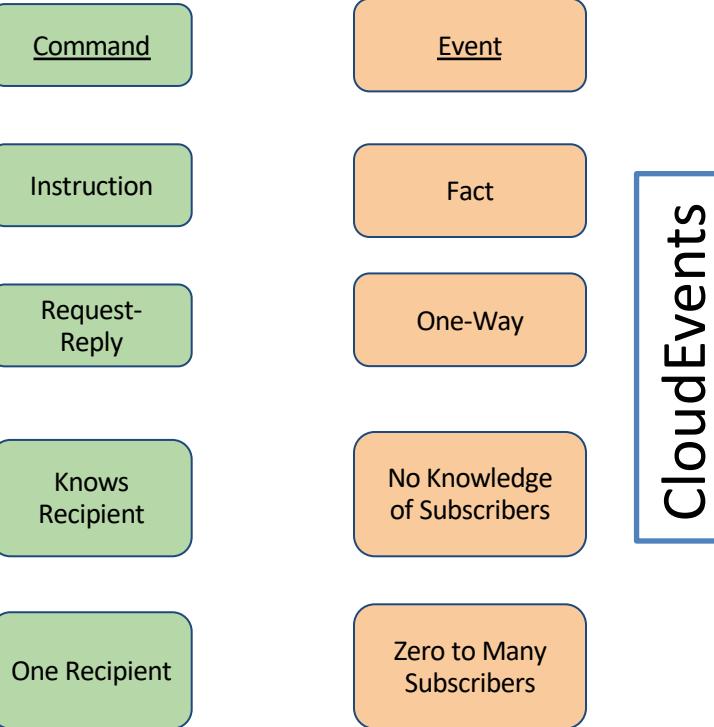
Portability

The portability and productivity we can achieve from event data is hindered overall.



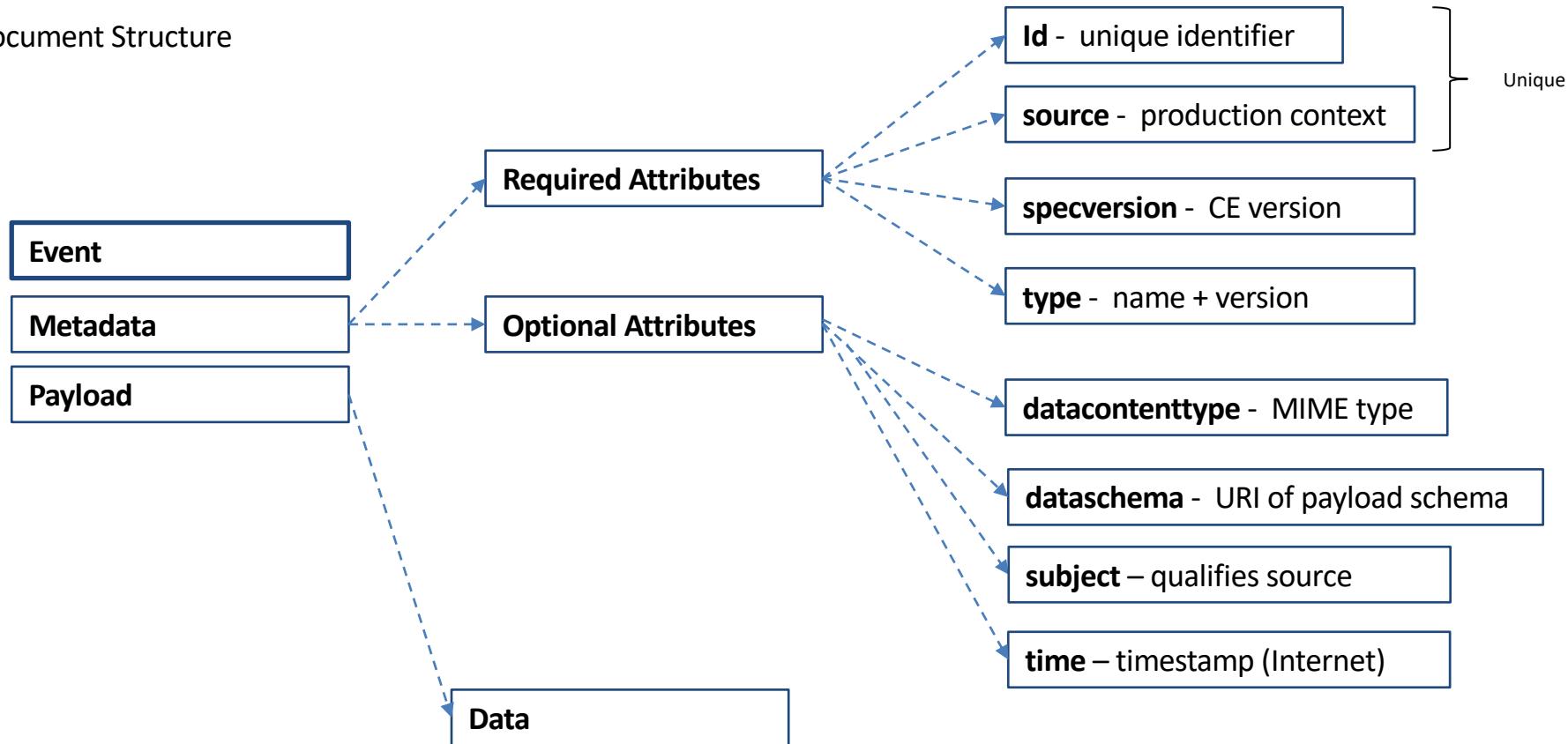
No Routing Information

- Protocol Specific Concern
- Event May Flow

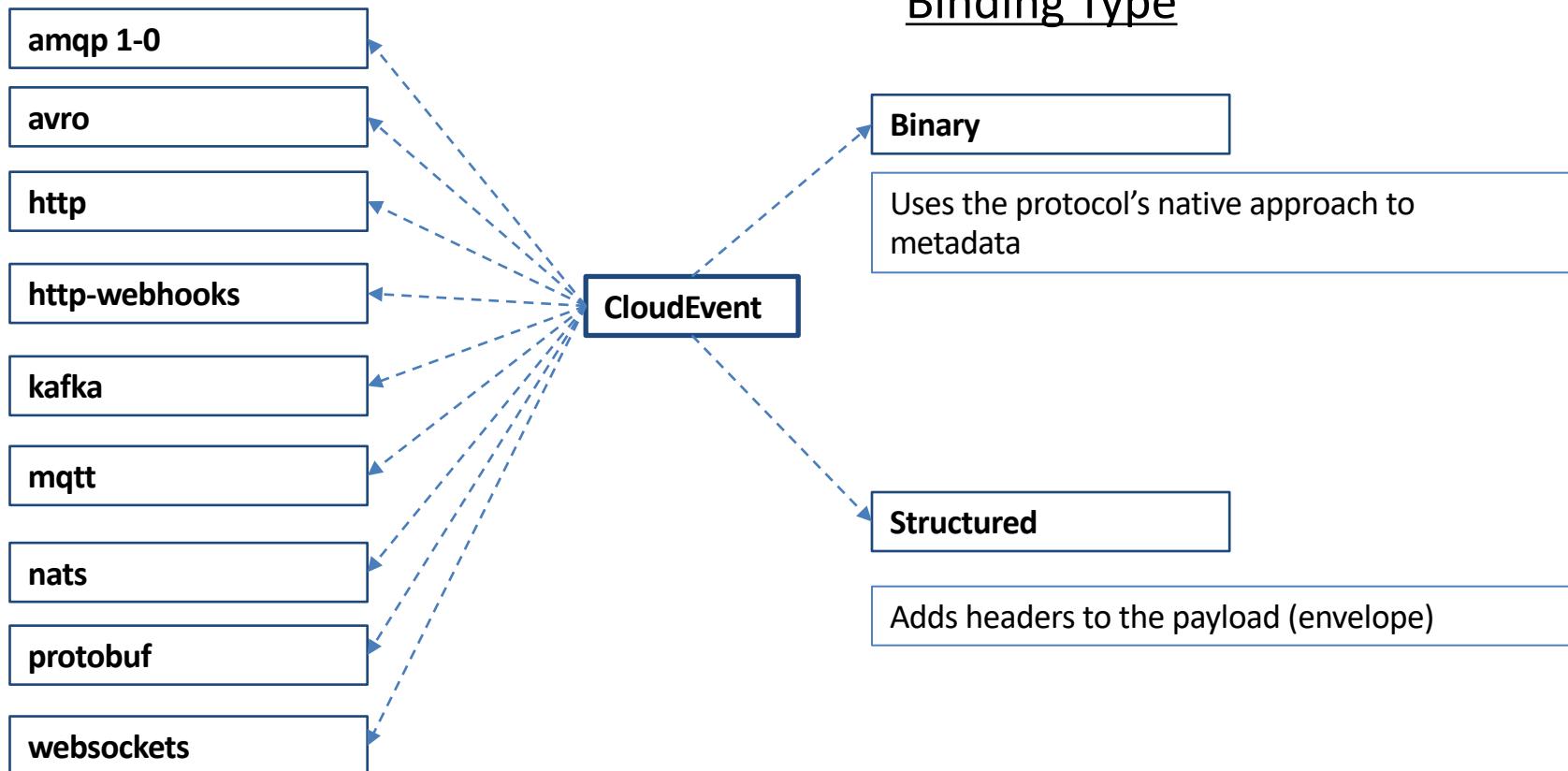


CloudEvents

Document Structure



Protocol Binding



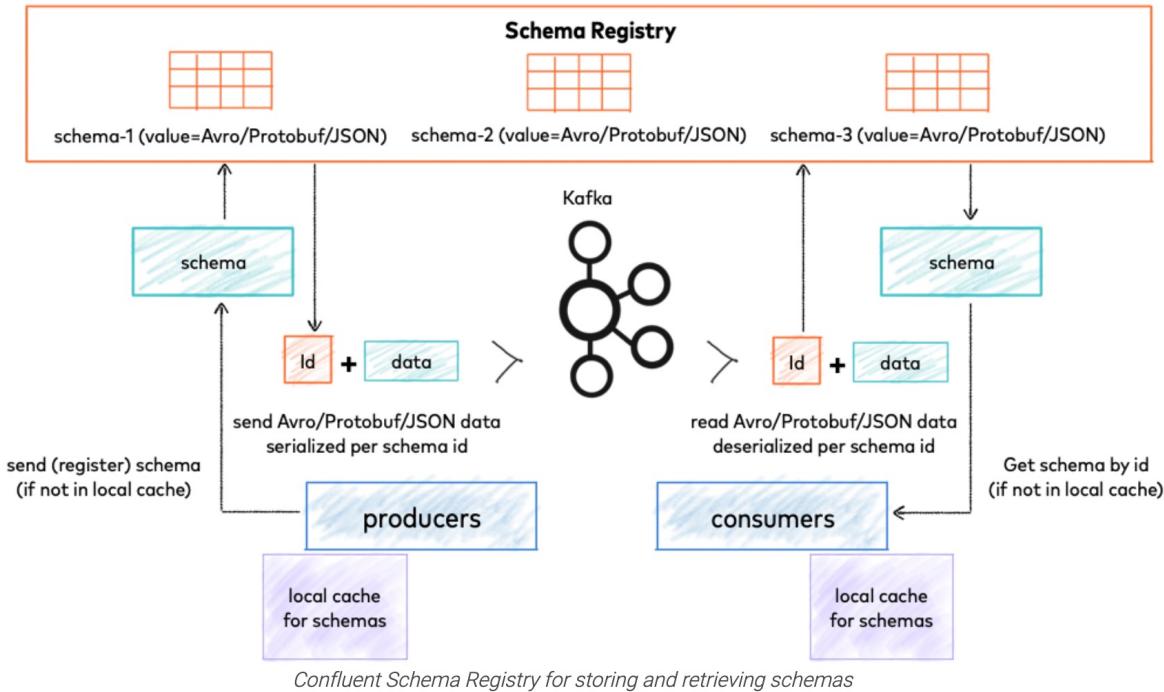
Binary

----- Message -----
Topic Name: mytopic
----- key -----
Key: mykey
----- headers -----
ce_specversion: "1.0"
ce_type: "com.example.someevent"
ce_source: "/mycontext/subcontext"
ce_id: "1234-1234-1234"
ce_time: "2018-04-05T03:56:24Z"
content-type: application/avro
----- value -----
... application data encoded in Avro ...

Structured

----- Message -----
Topic Name: mytopic
----- key -----
Key: mykey
----- headers -----
content-type: application/cloudevents+json; charset=UTF-8
----- value -----
{
 "specversion" : "1.0",
 "type" : "com.example.someevent",
 "source" : "/mycontext/subcontext",
 "id" : "1234-1234-1234",
 "time" : "2018-04-05T03:56:24Z", "datacontenttype"
 : "application/json",
 "data" : {
 ... application data encoded in JSON ...
 }
}

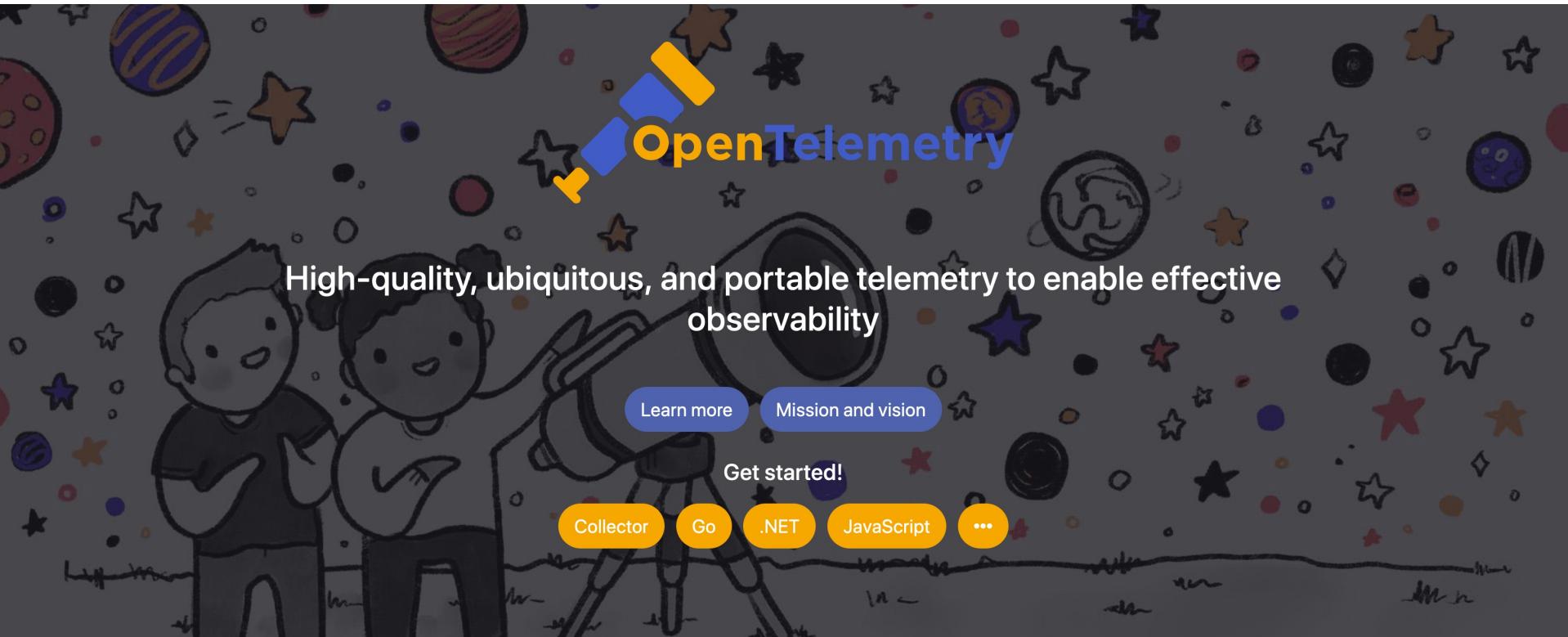
Maintaining Event Schemas



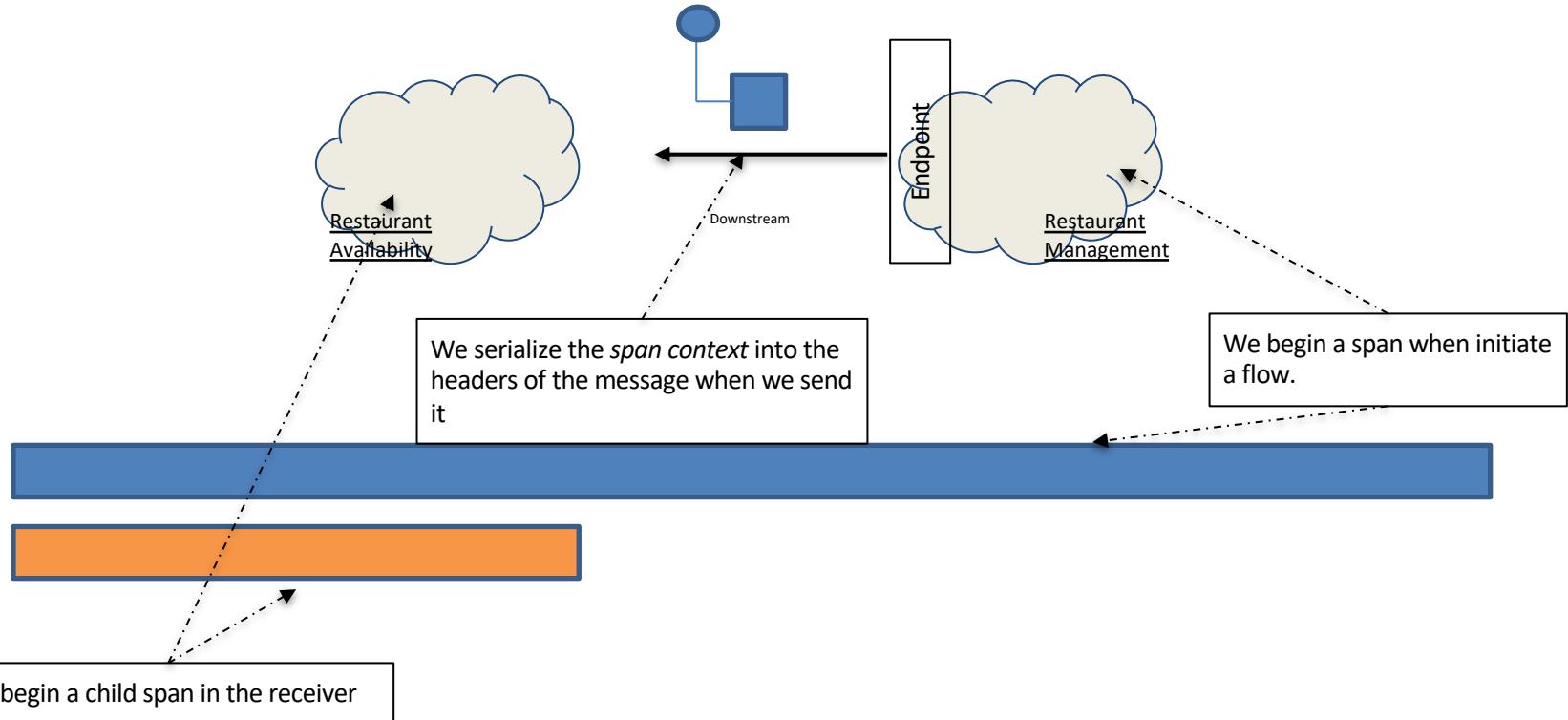
<https://docs.confluent.io/platform/current/schema-registry/index.html>

OpenTelemetry etc.

7 OBSERVABILITY



OpenTelemetry Tracing

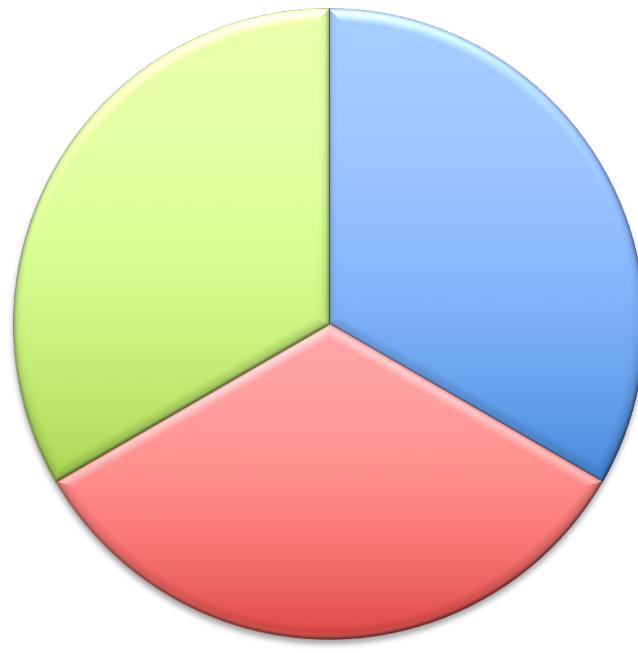


Spans

- Operation Name
- Start and Finish Timestamps
- Span Context (span_id, trace_id)
- Attributes (standard and defined)
- Events

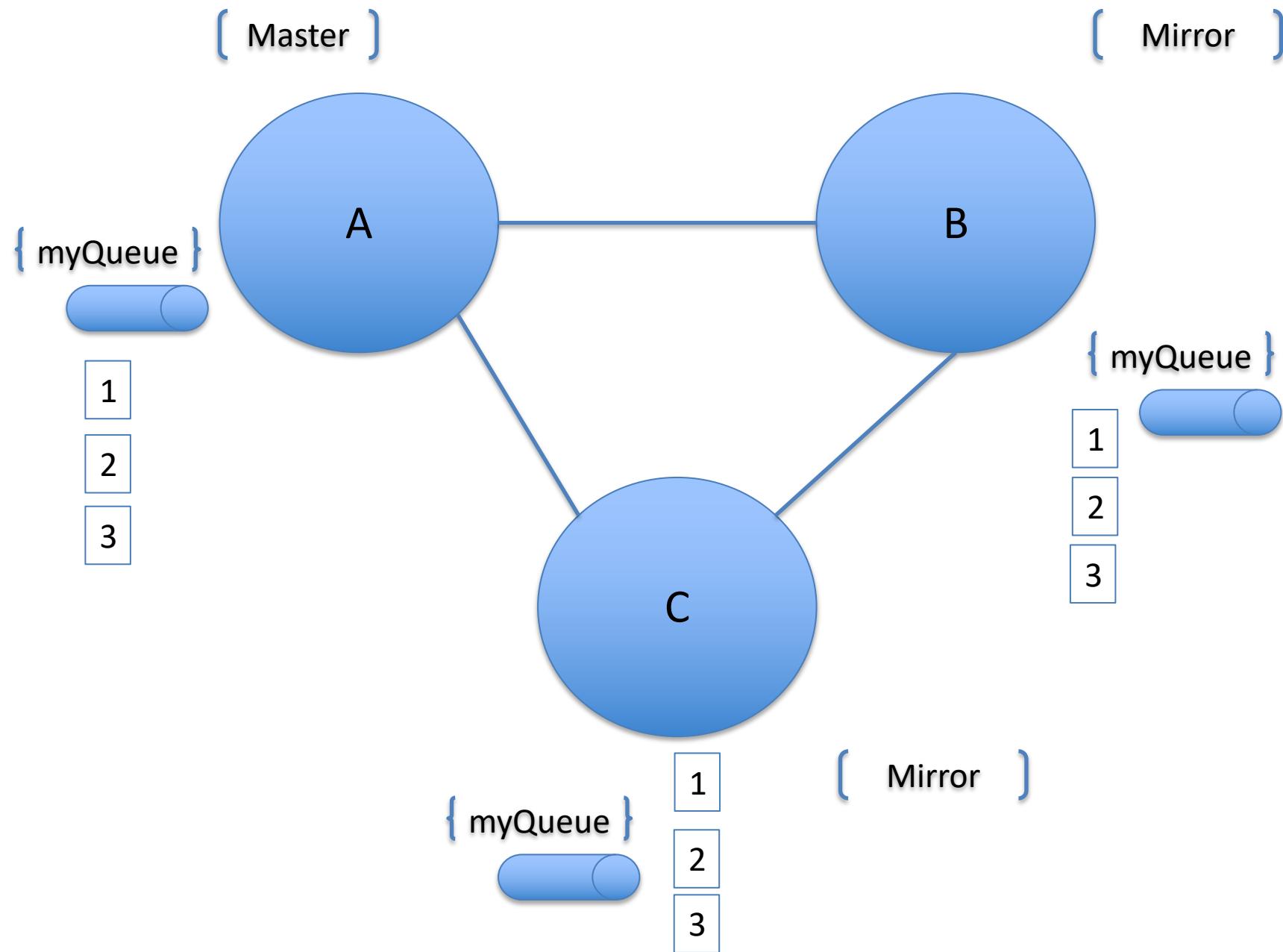
8 CAP THEOREM

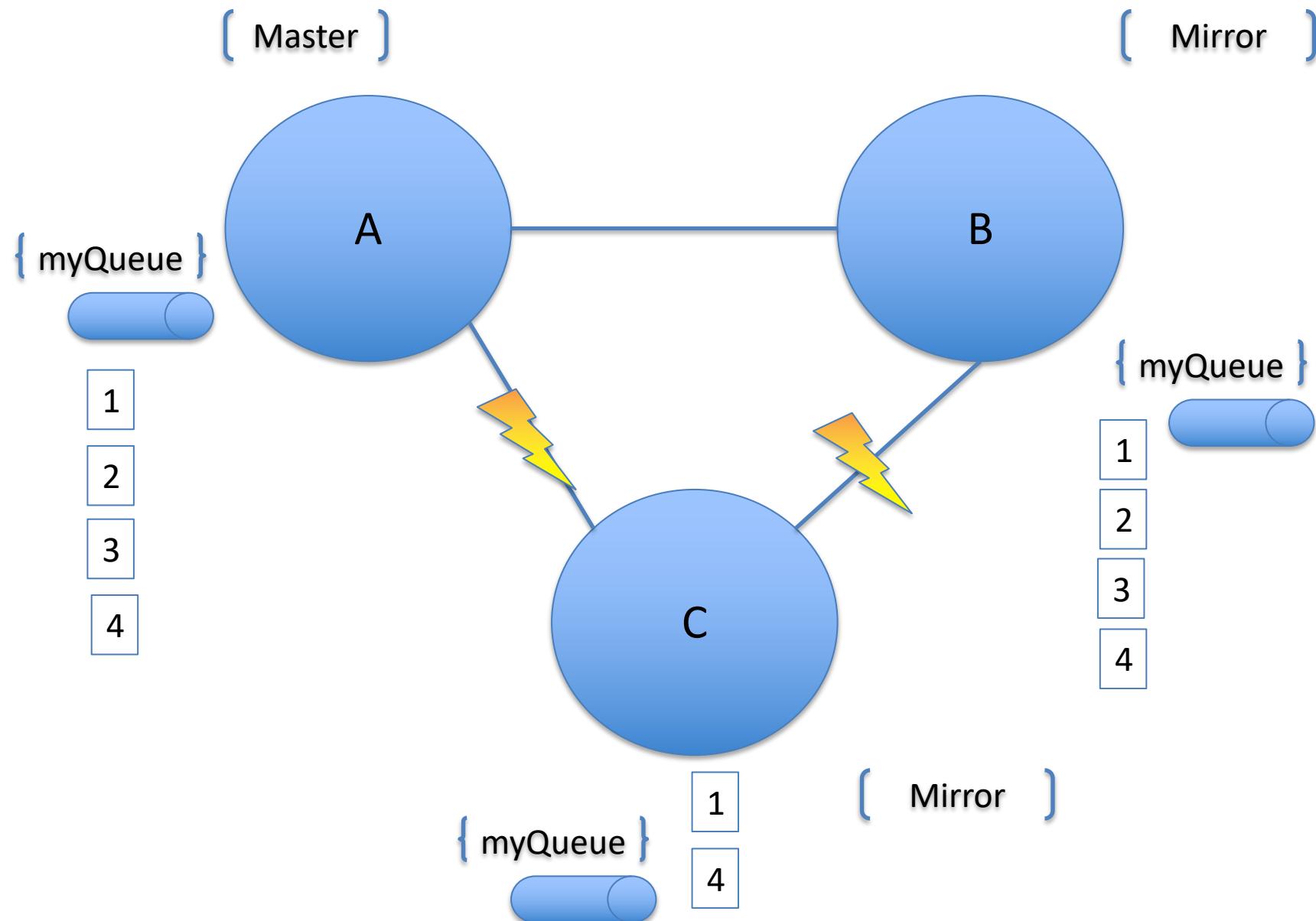
CAP Theorem



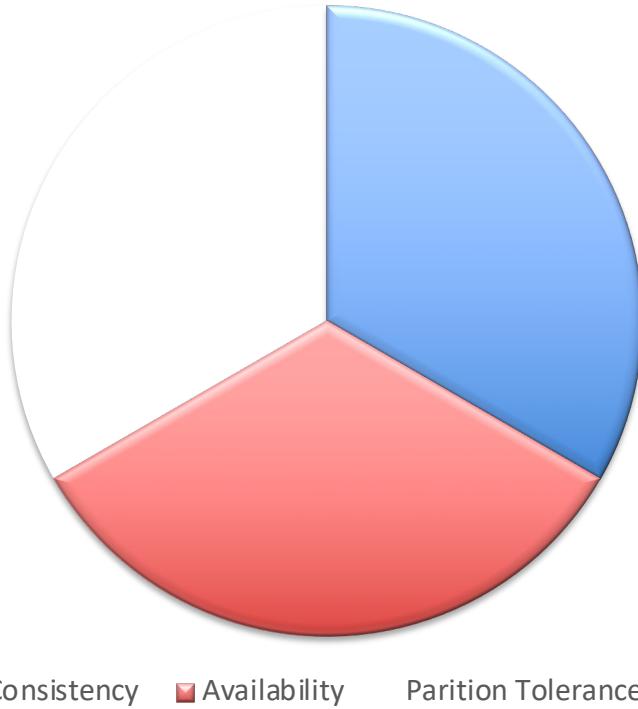
■ Consistency ■ Availability ■ Partition Tolerance ■

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

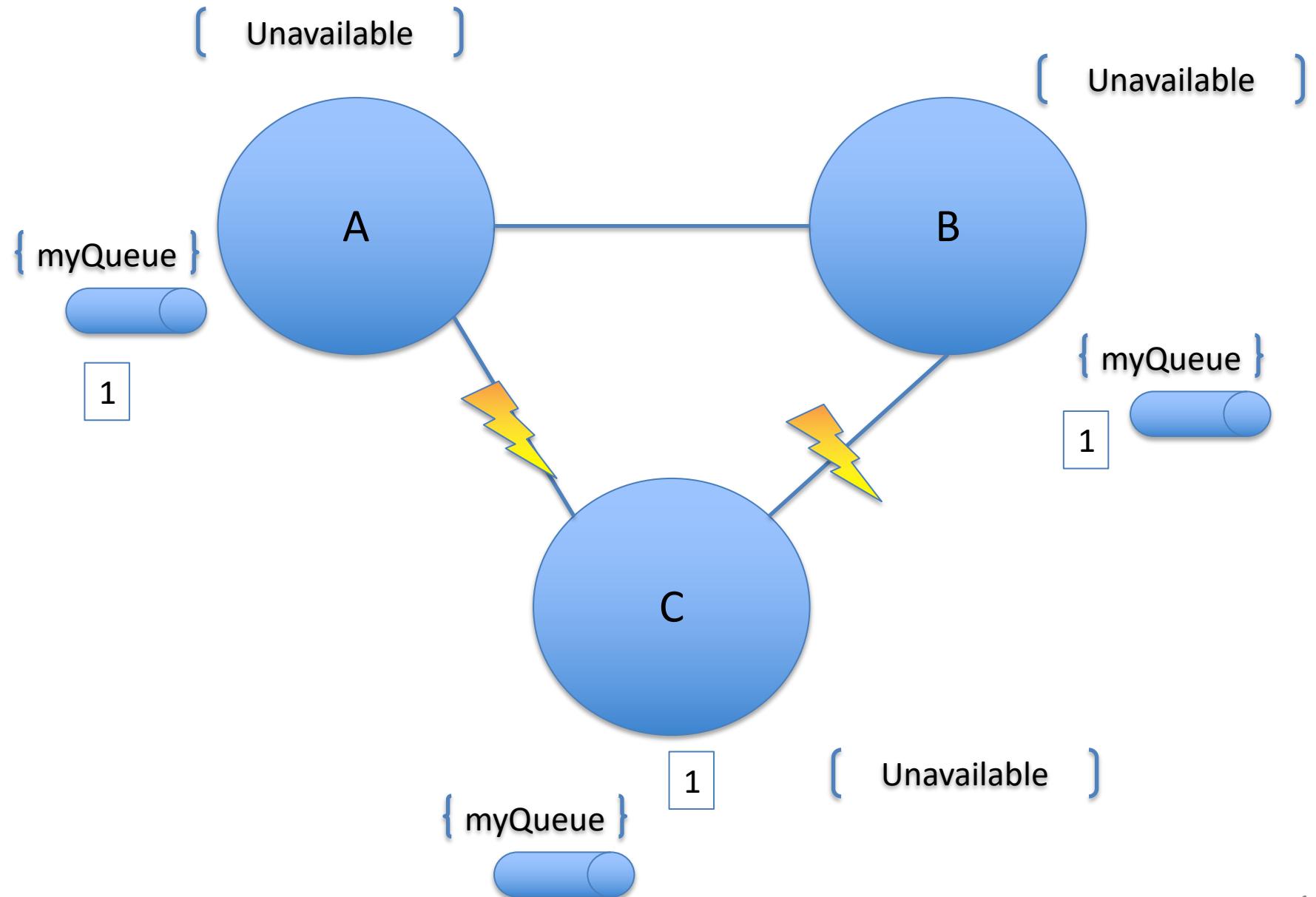




CAP Theorem



Forfeit Partitions

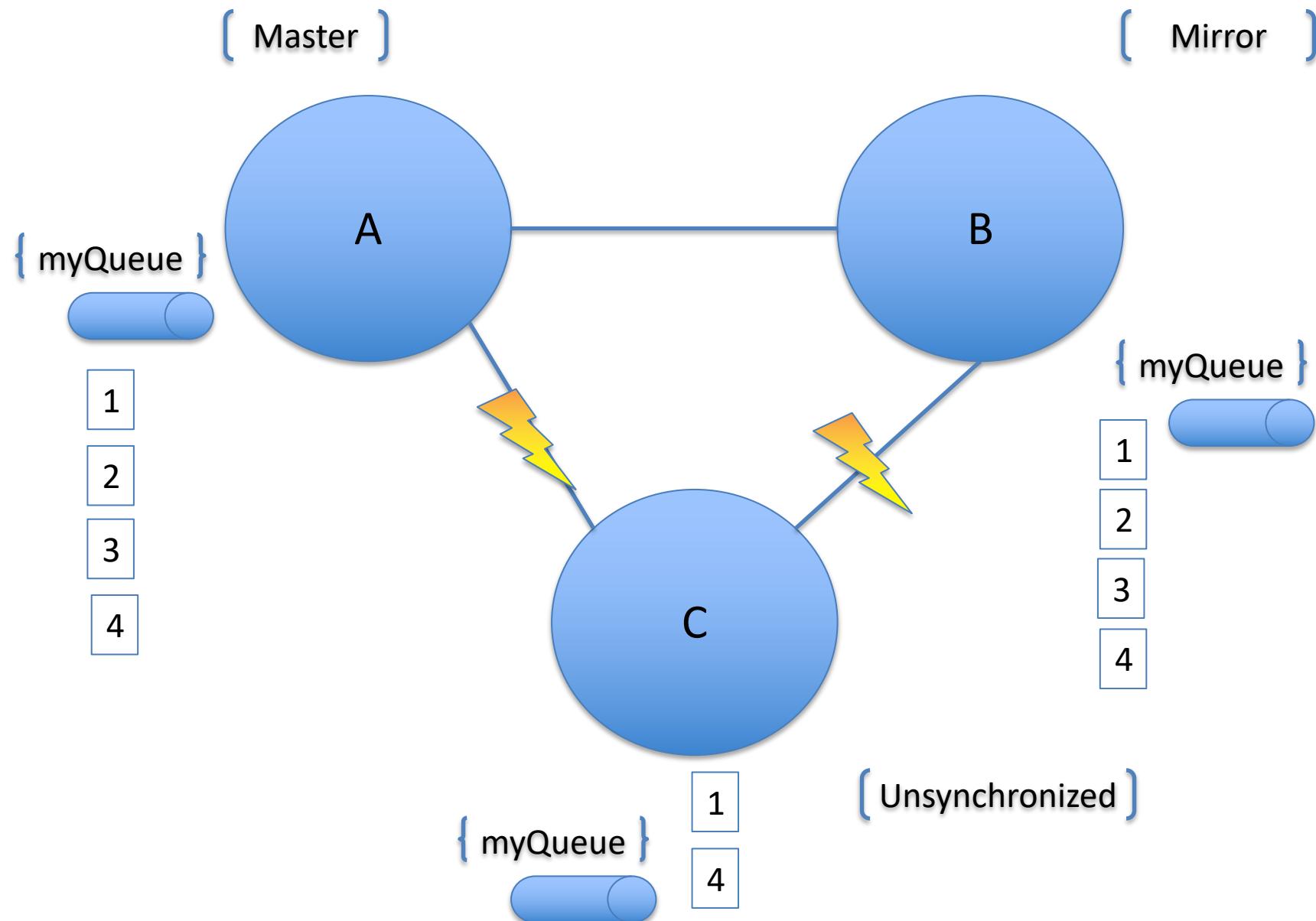


CAP Theorem

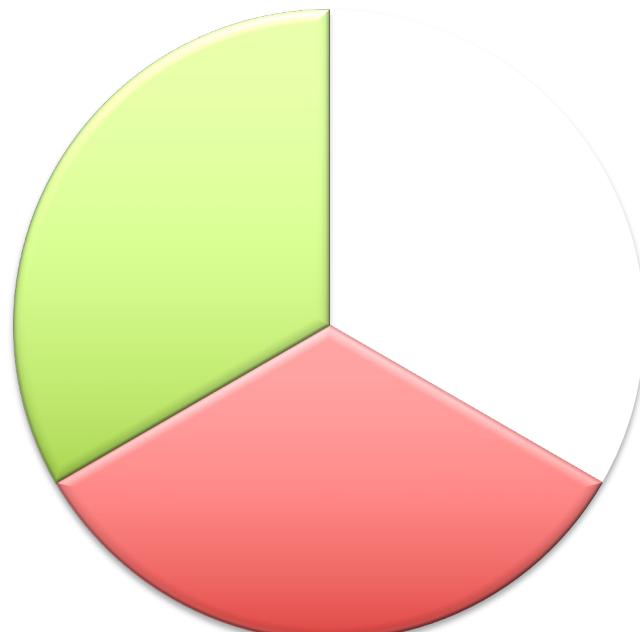


■ Consistency Availability ■ Partition Tolerance ■

Forfeit Availability

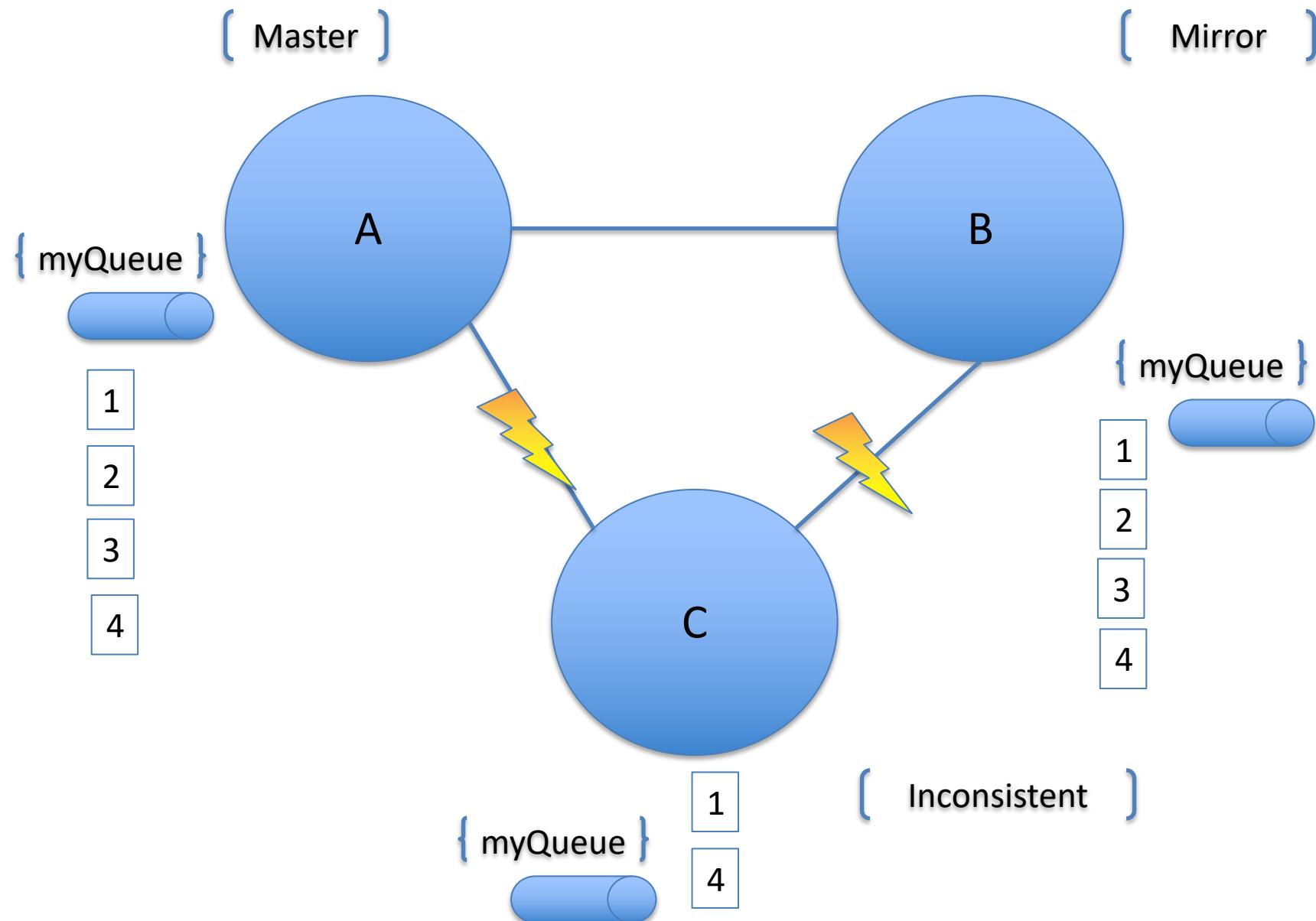


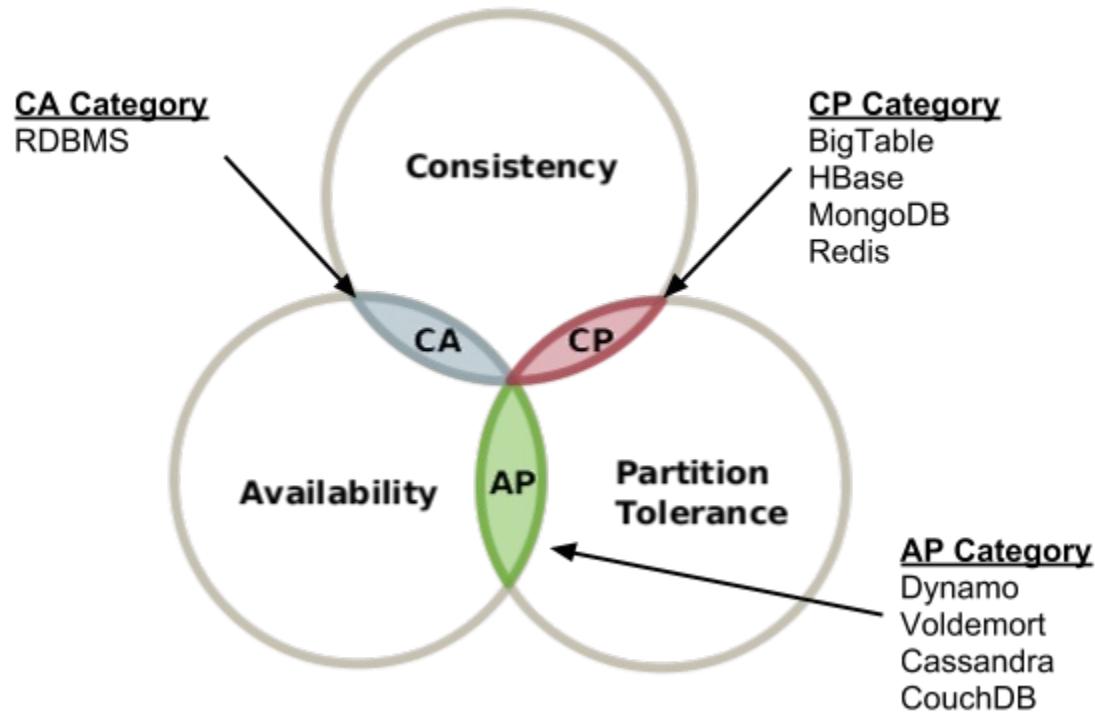
CAP Theorem



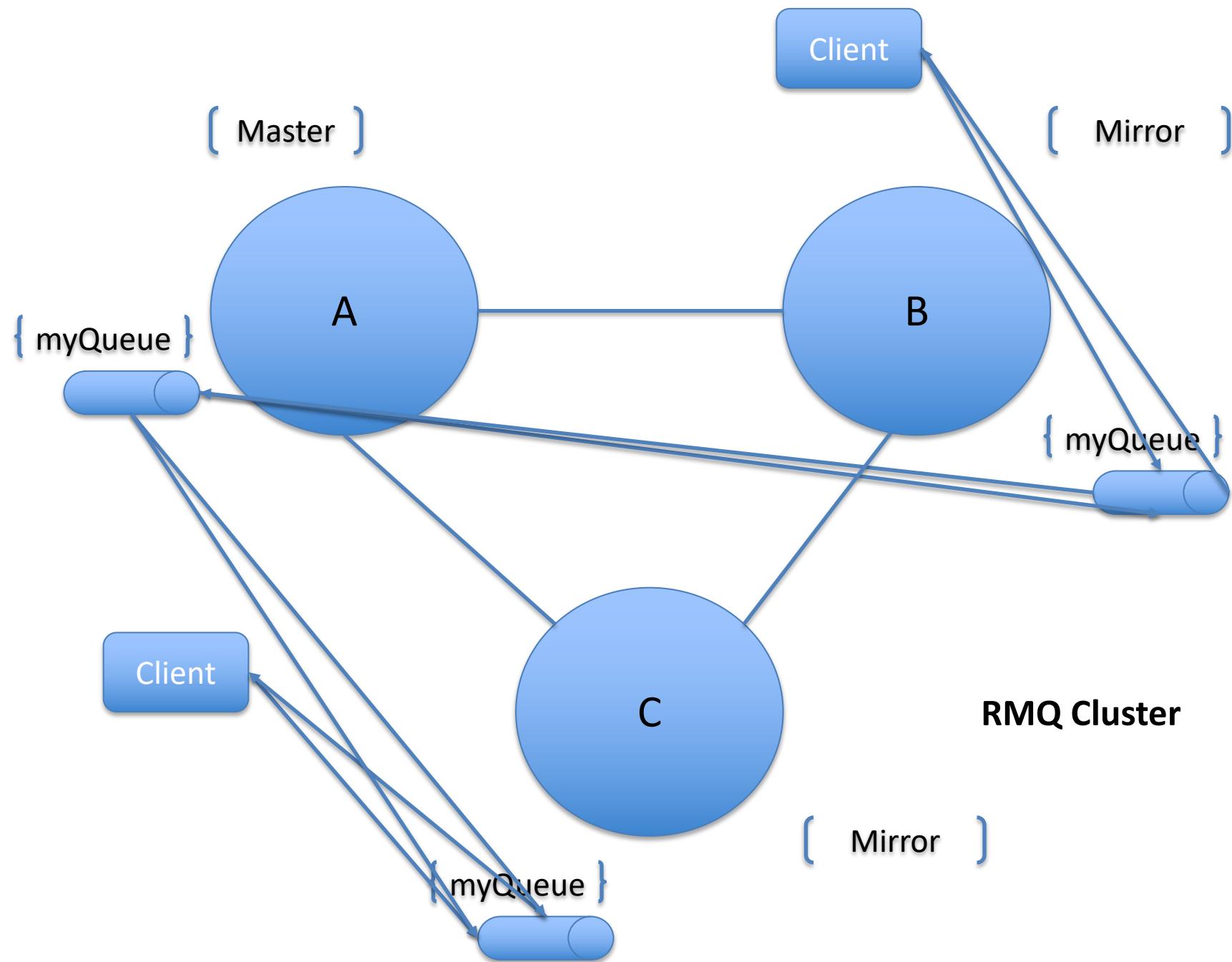
Consistency Availability Partition Tolerance

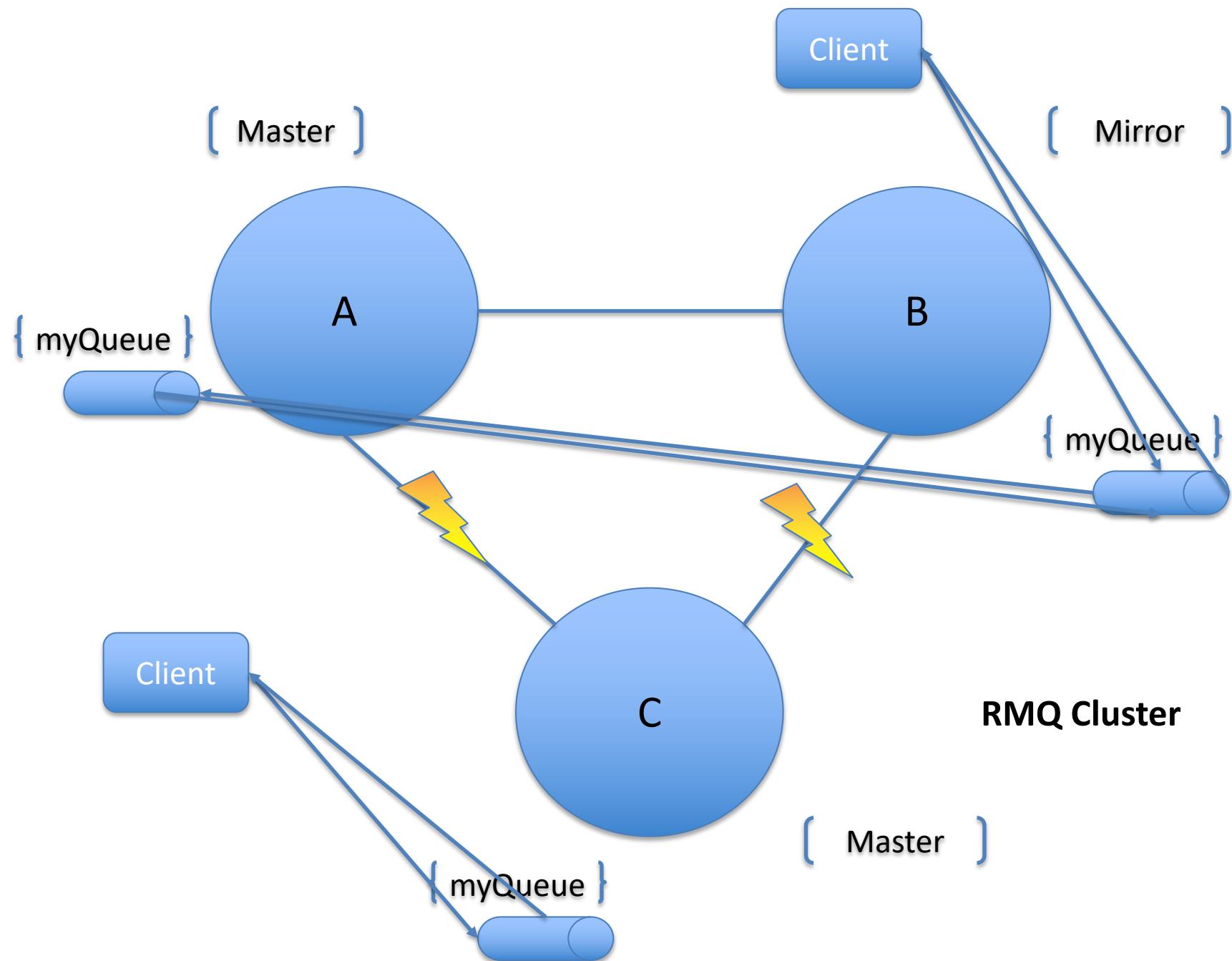
Forfeit Consistency

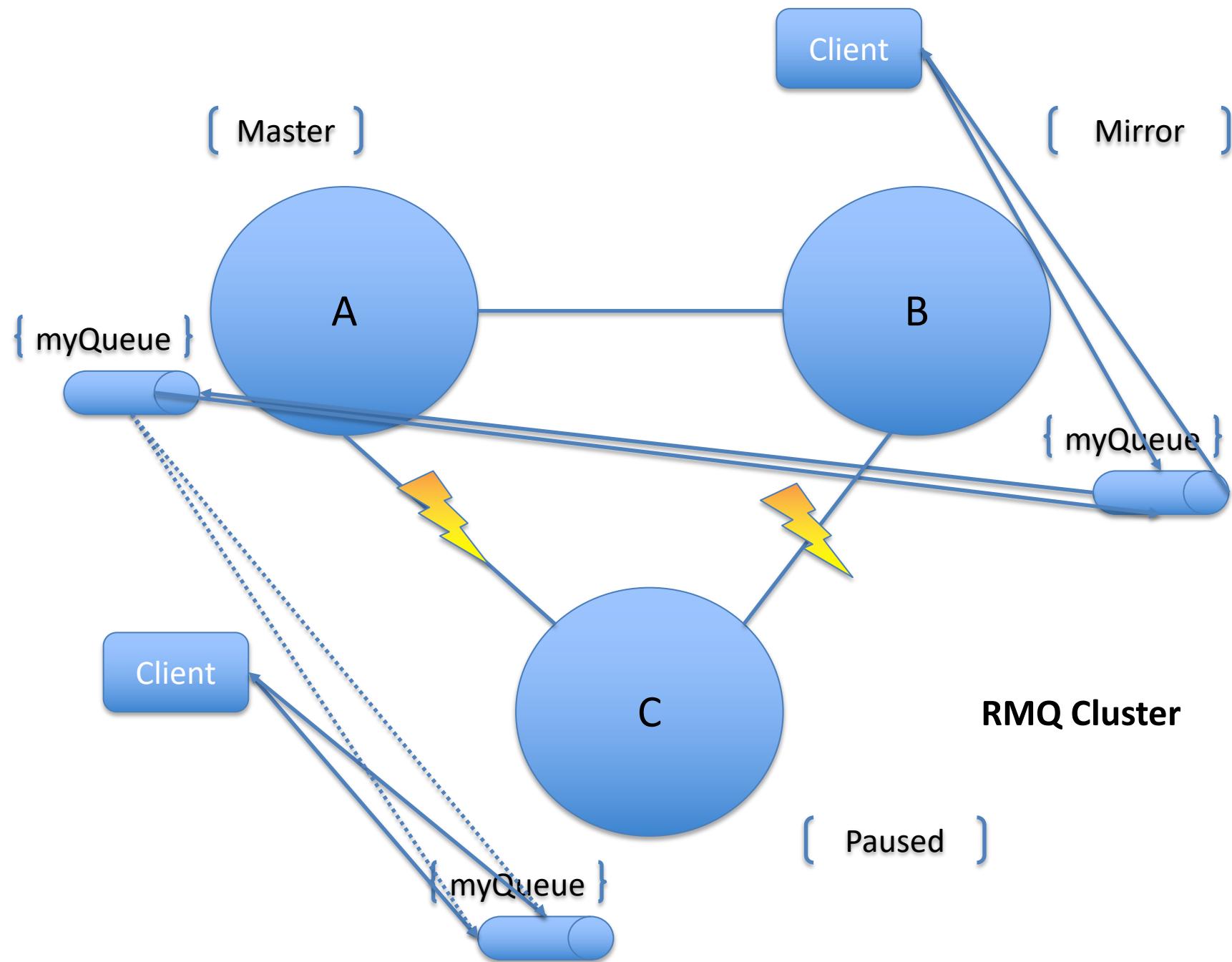


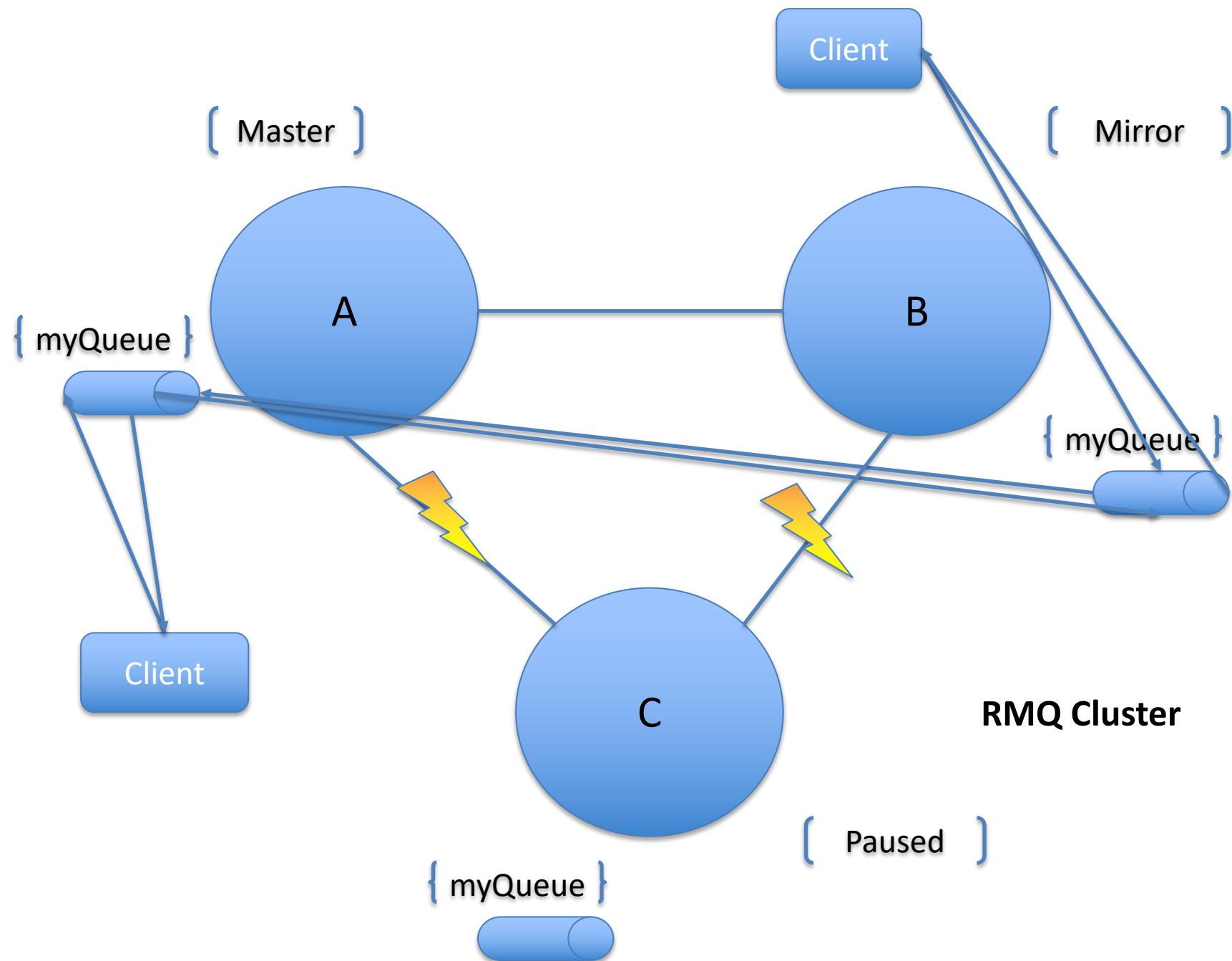


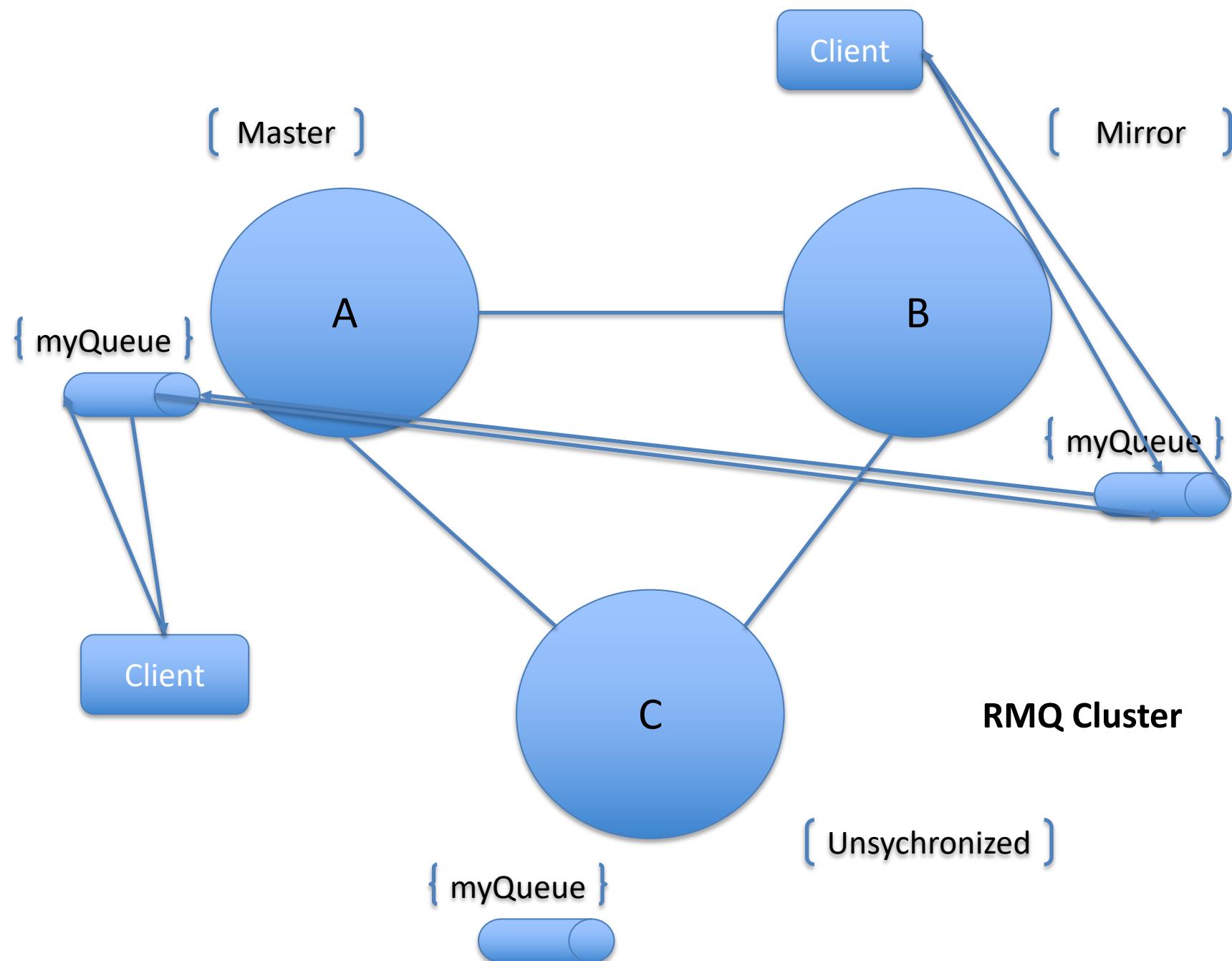
RabbitMQ and CAP Theorem

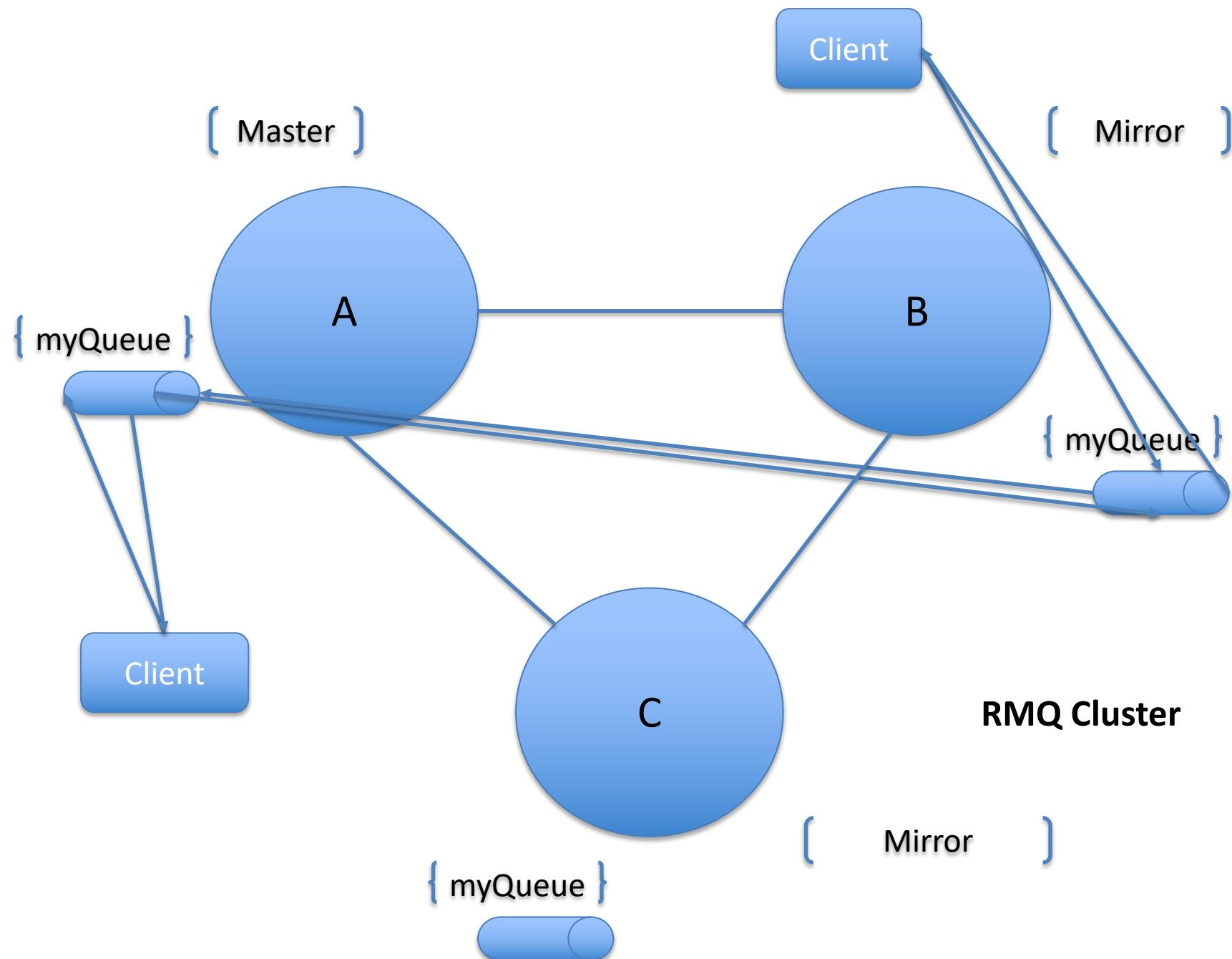


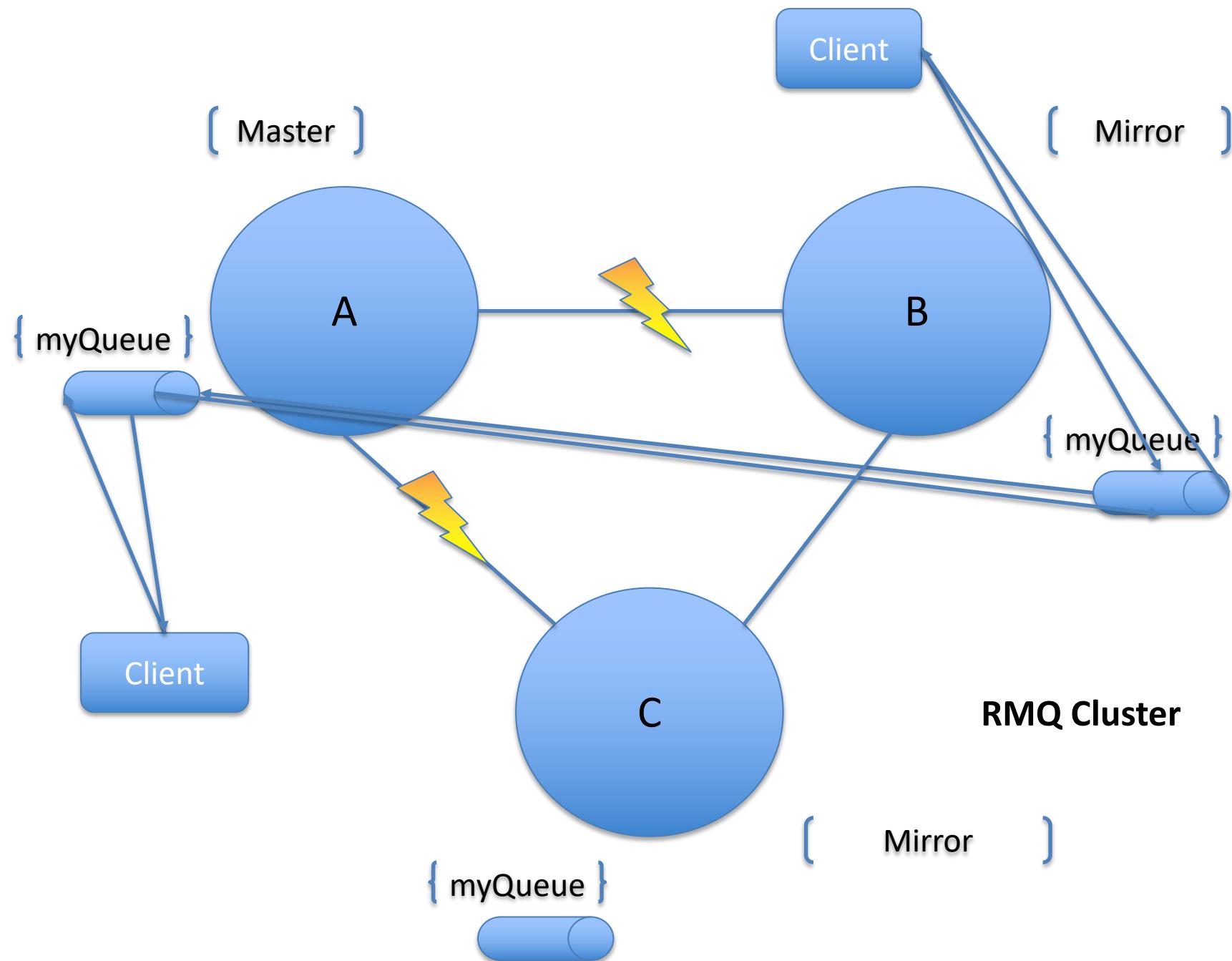


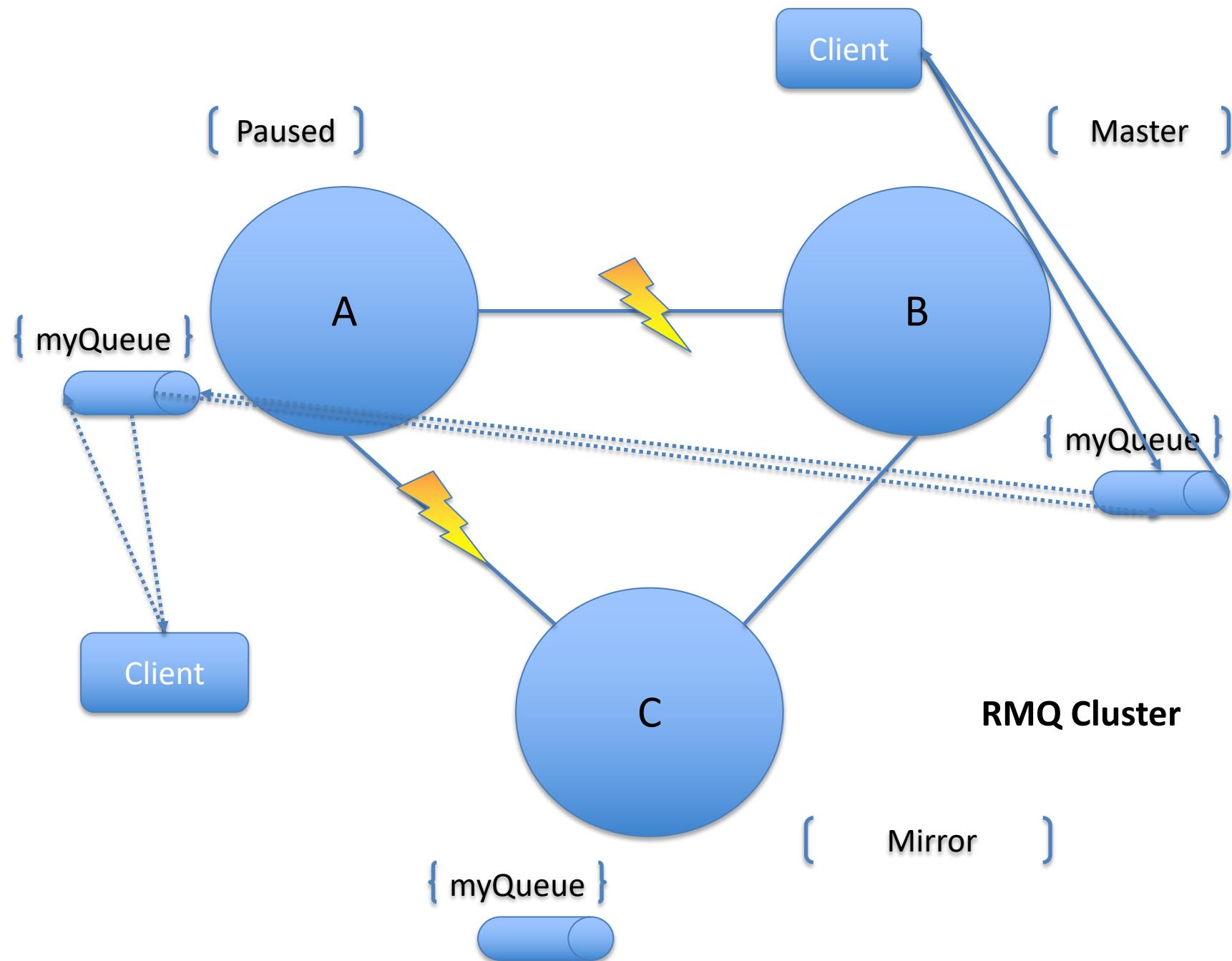


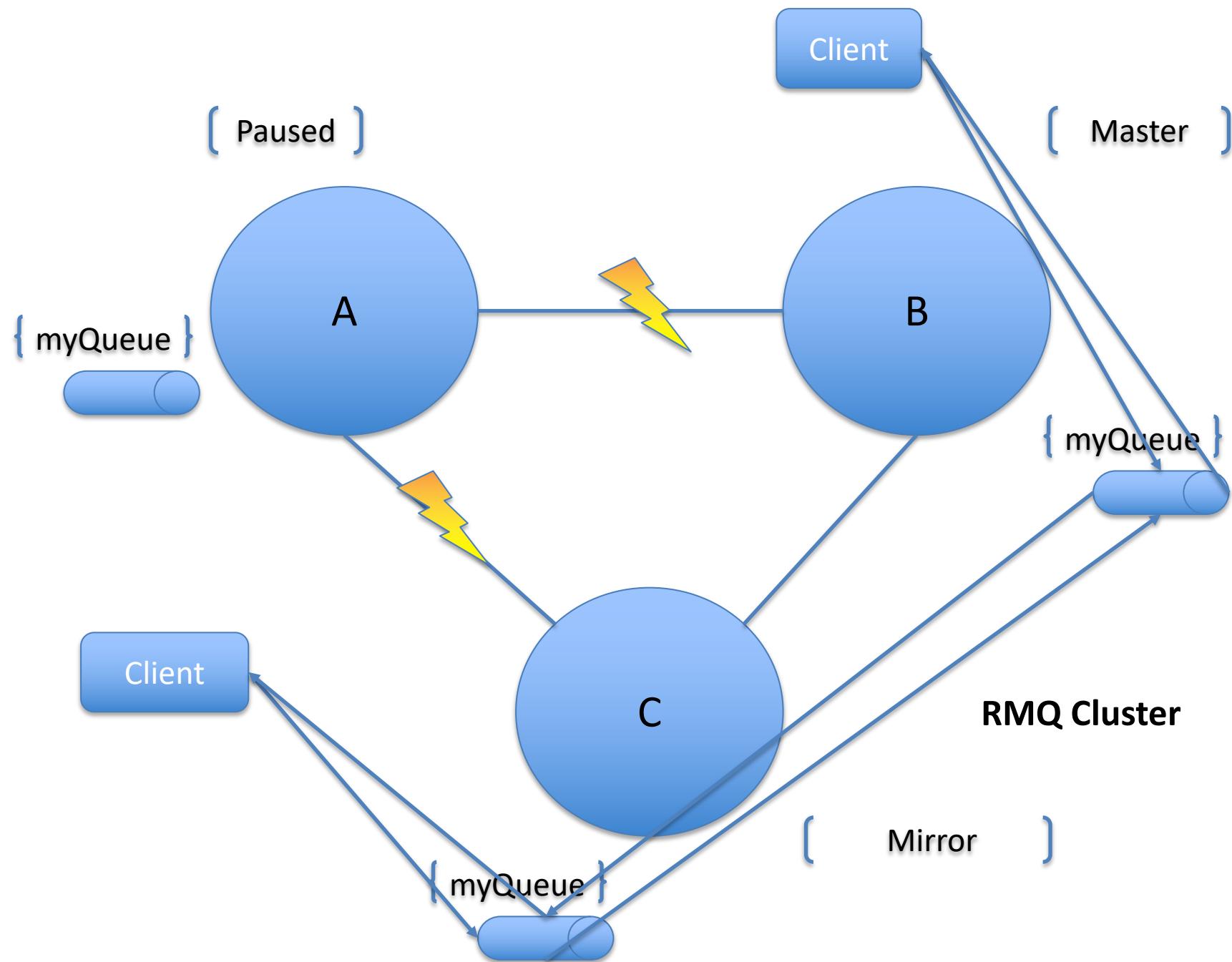


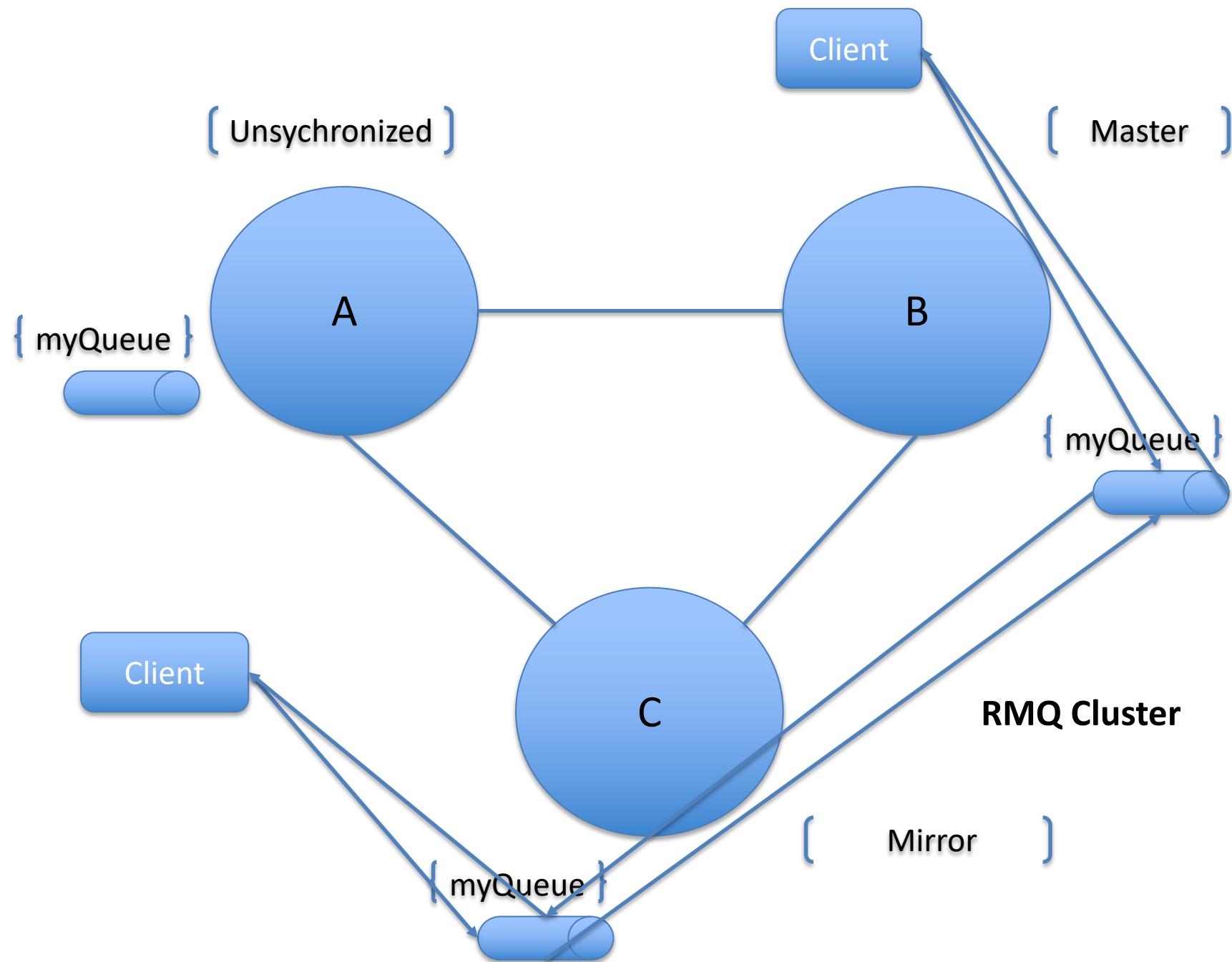


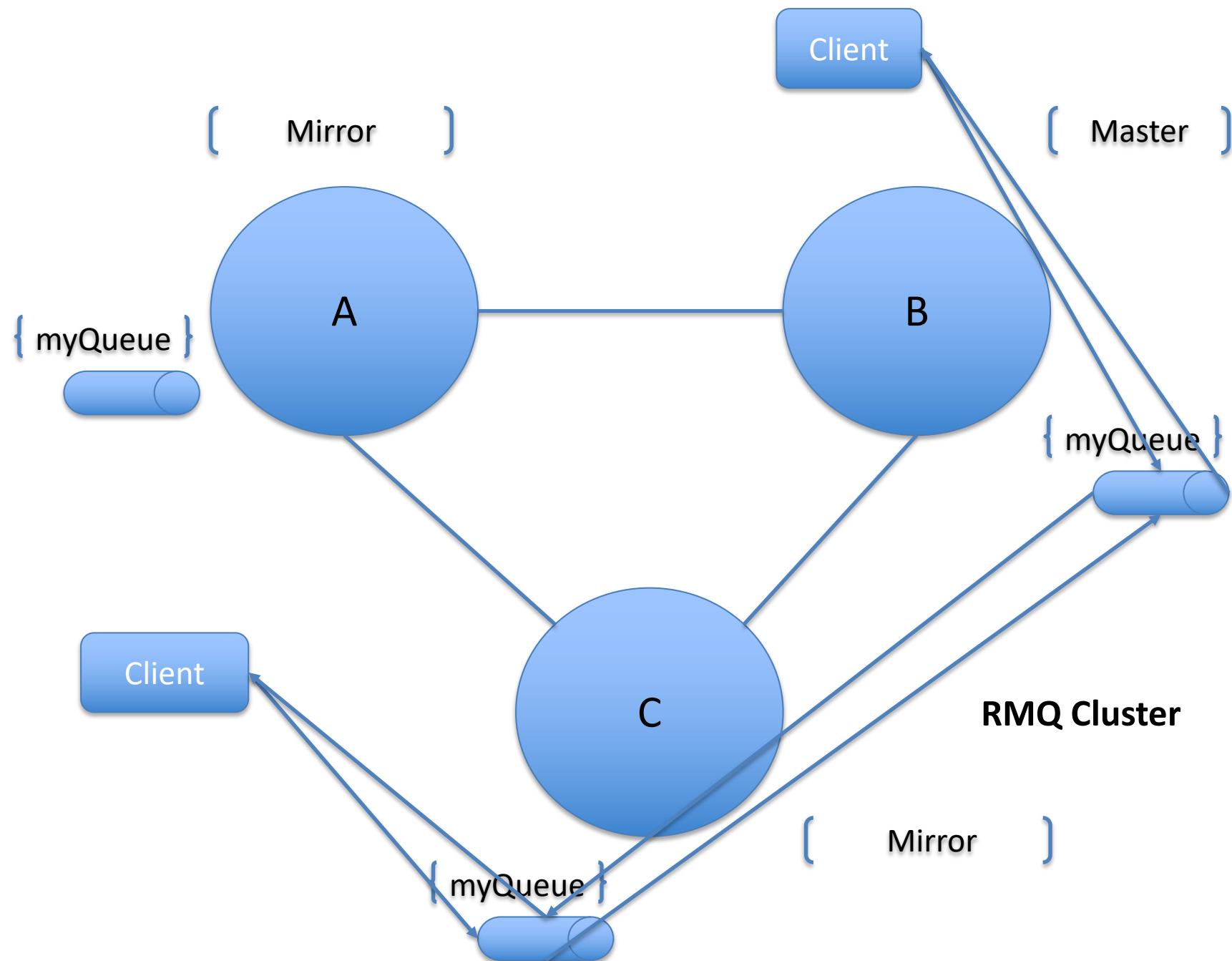


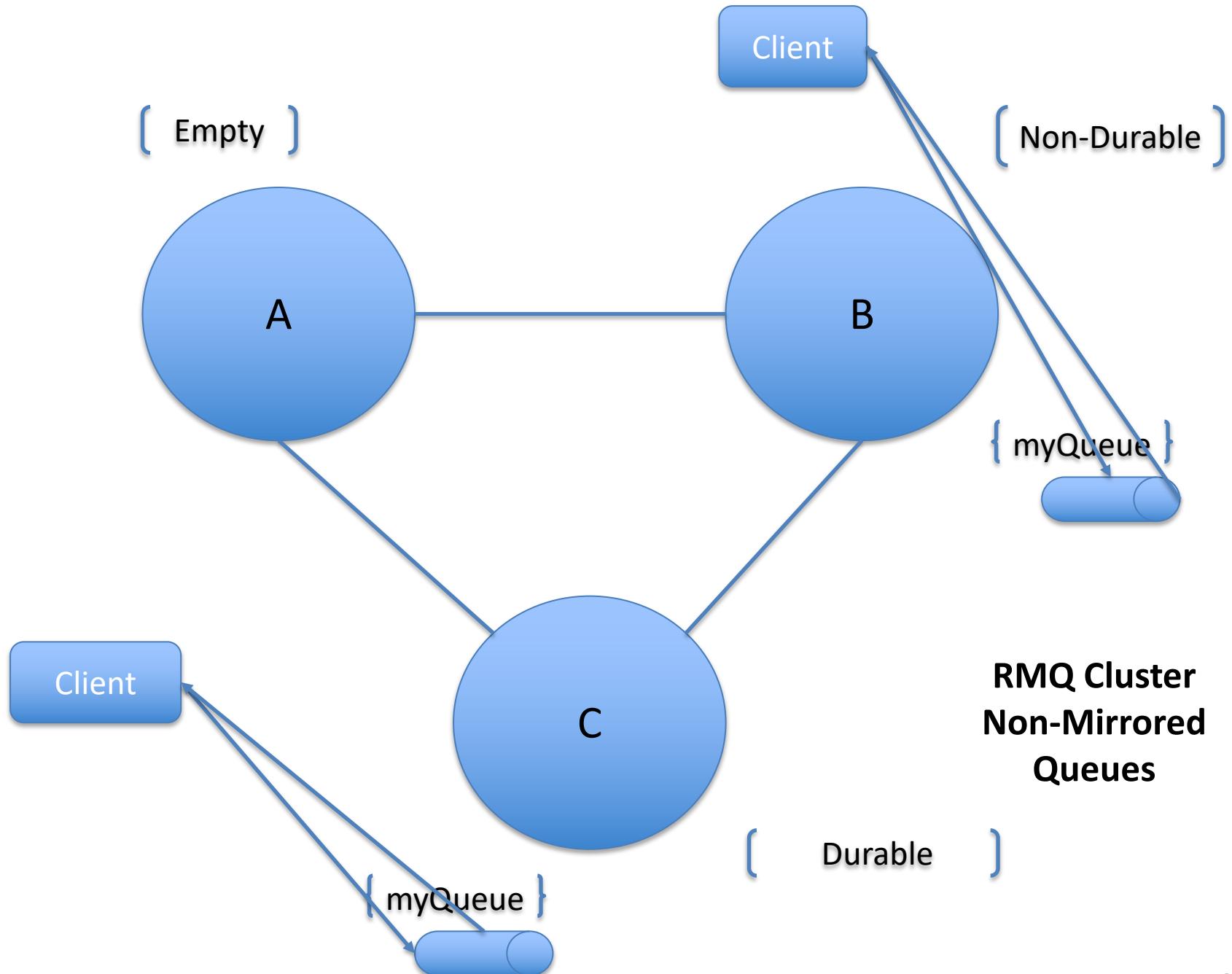


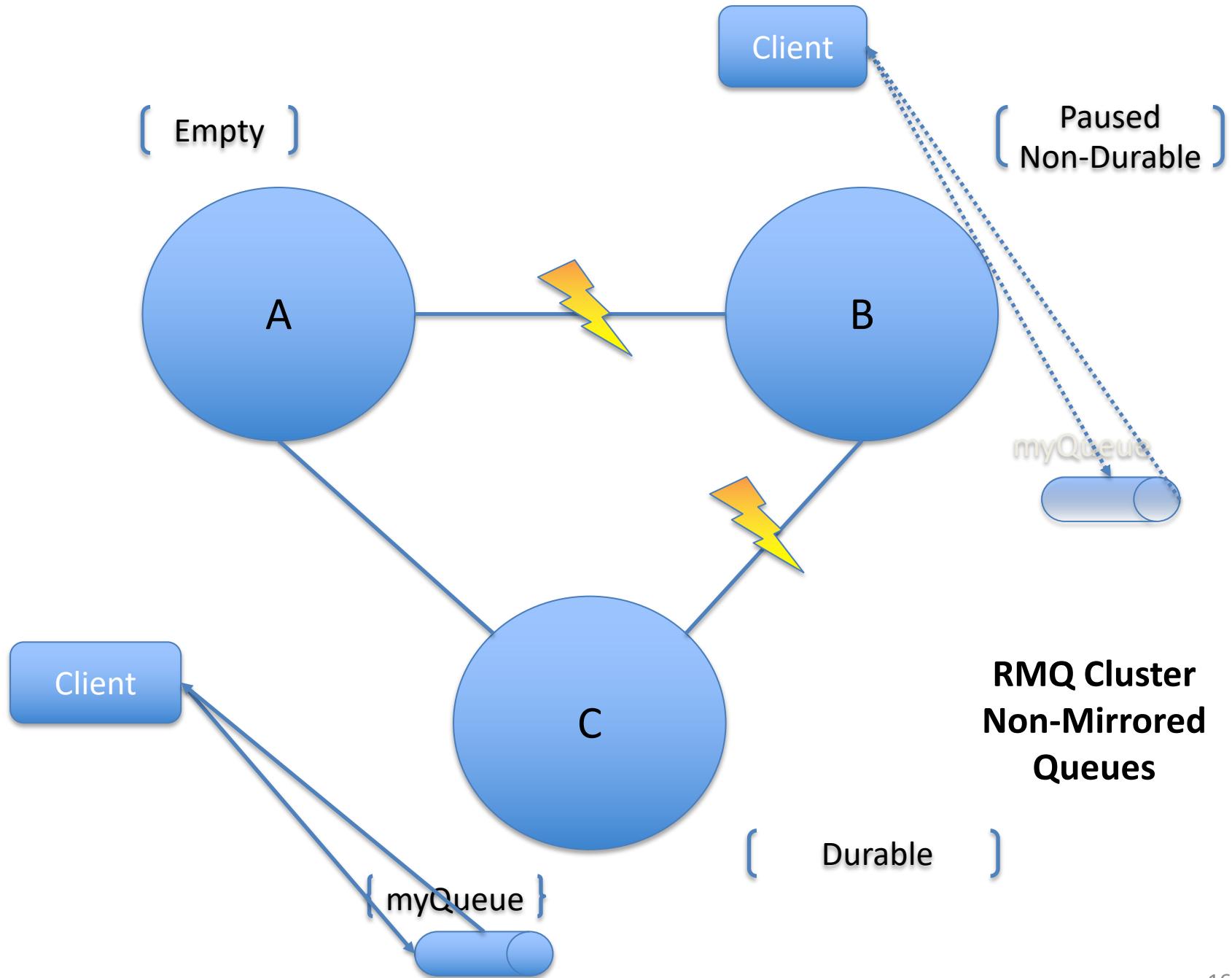


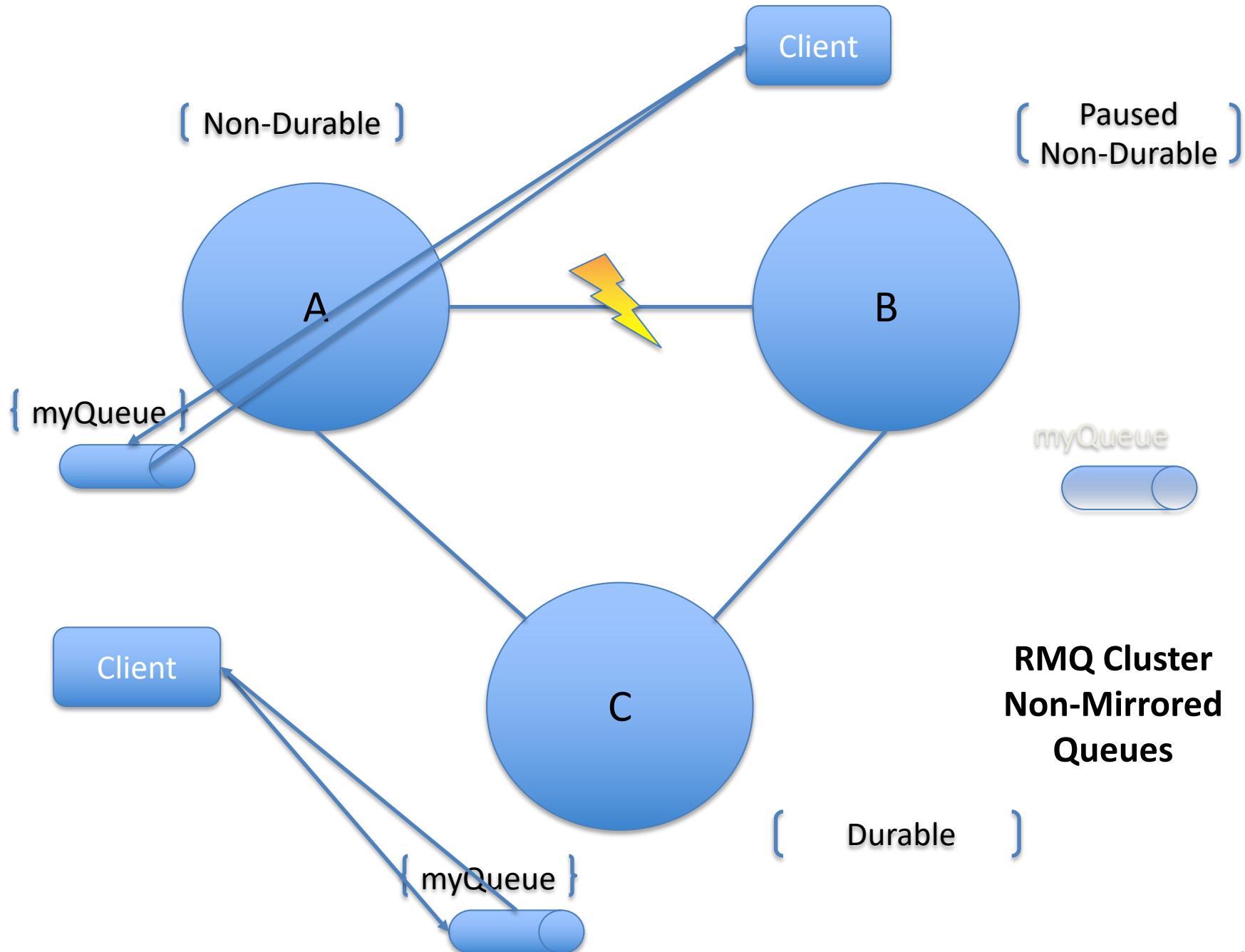


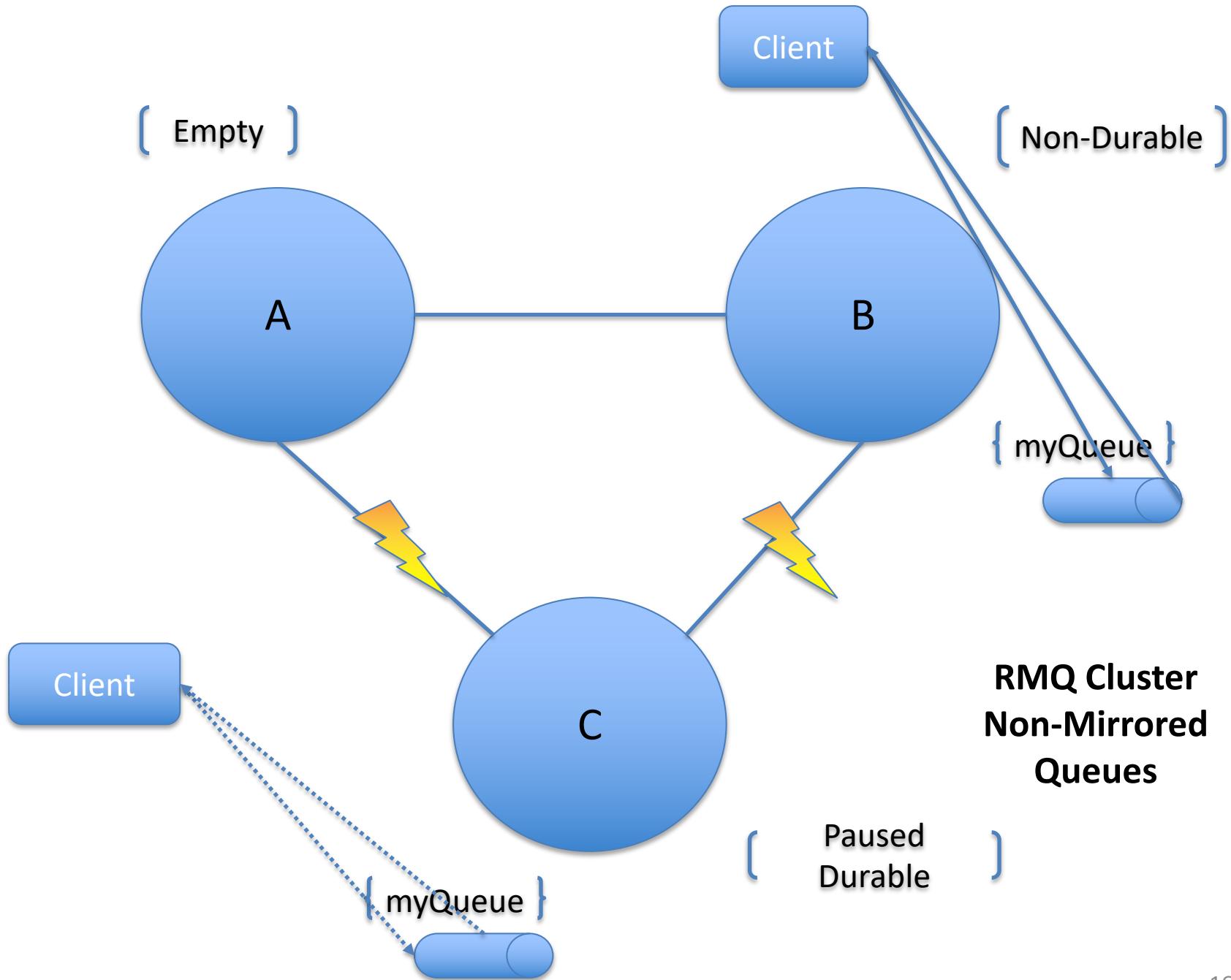


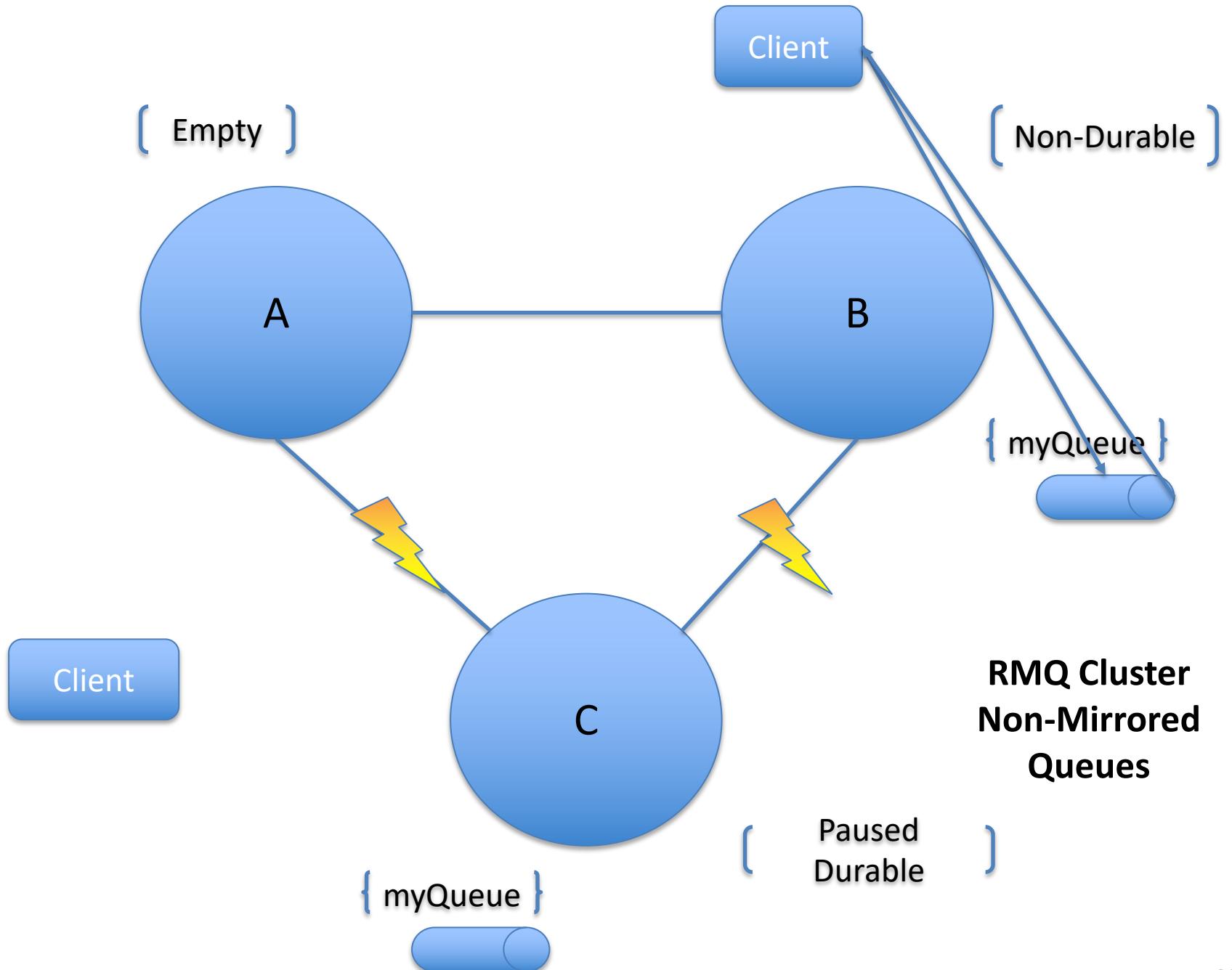


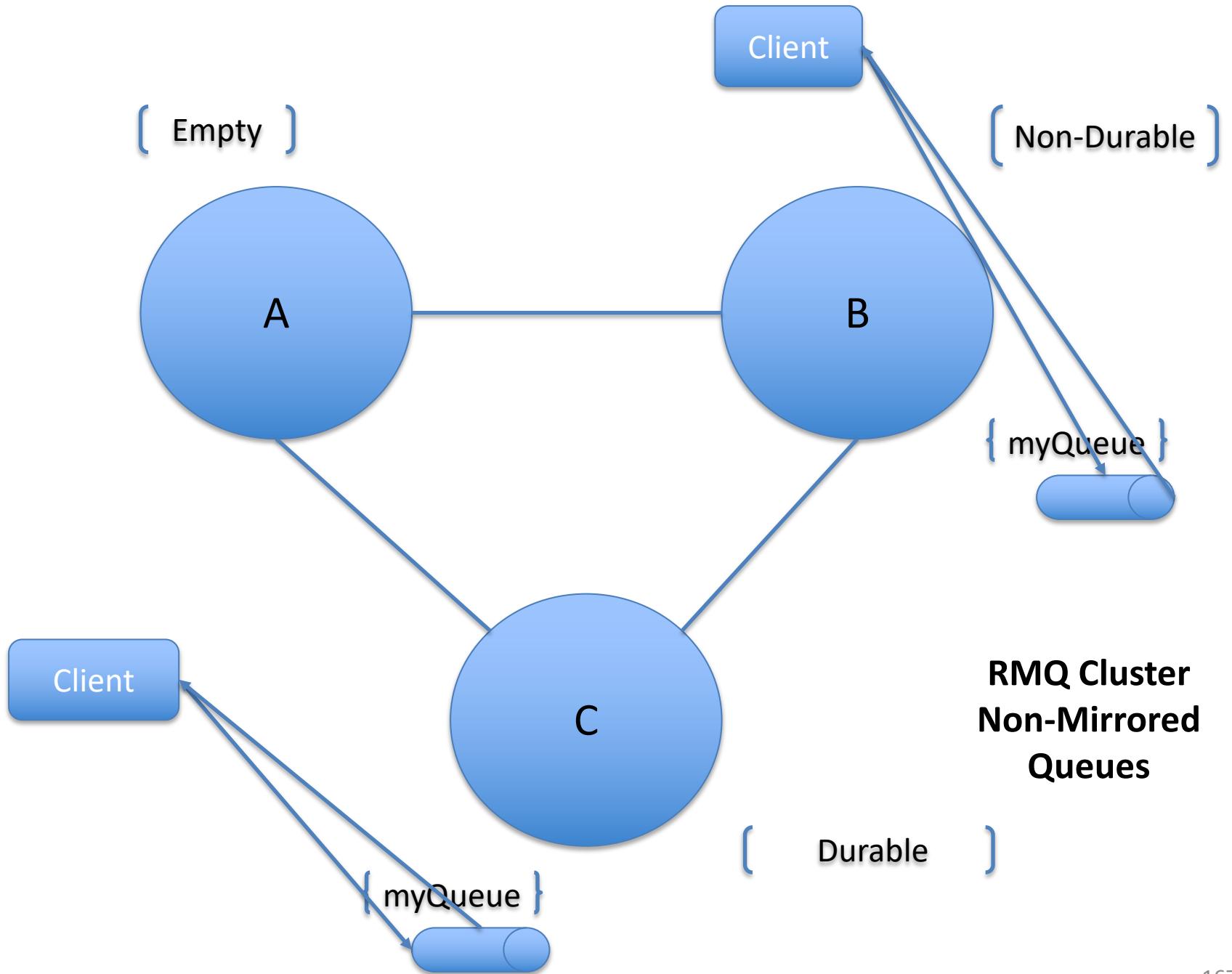












So which CAP options does an RMQ cluster support?

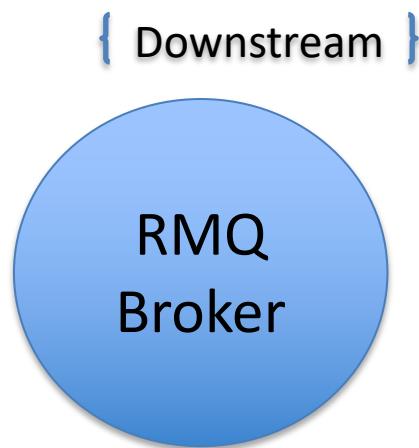
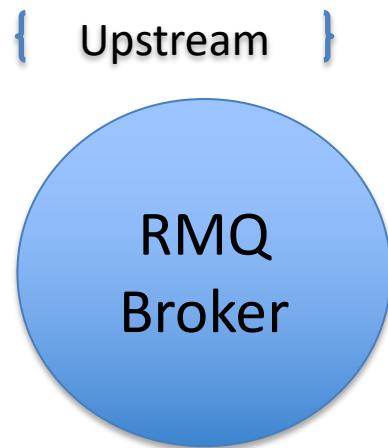
Pause Minority is Consistency (C) and Partition Tolerance (P)

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**

9. CONSUMERS

Task Based UI

Your Stay

Your Booking Details

First Name * Last Name *

Email Address

Check-In 8 October 2021, 1 night, free cancellation before 1 October 2021

King-Sized Landmark View Ensuite Coffee Machine

Want to book a taxi or shuttle ride in advance?
Avoid surprises - get from the airport to your accommodation without a hitch.
We'll add taxi options to your booking confirmation.

I'm interested in renting a car
Make the most out of your trip and check car hire options in your booking confirmation.

Special Requests

Special requests cannot be guaranteed – but the property will do its best to meet your needs. You can always make a special request after your booking is complete!
Please write your requests in English. (optional)

I would like a quiet room

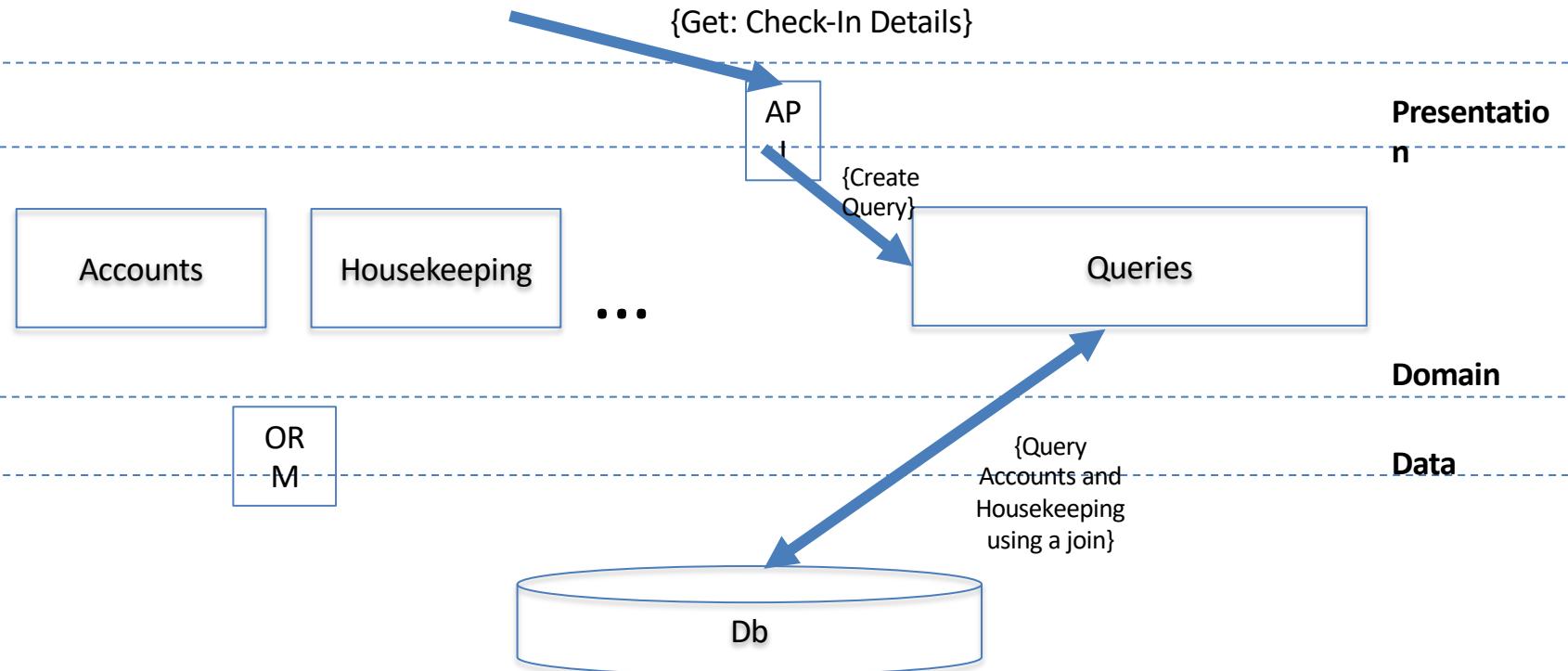
Book Now

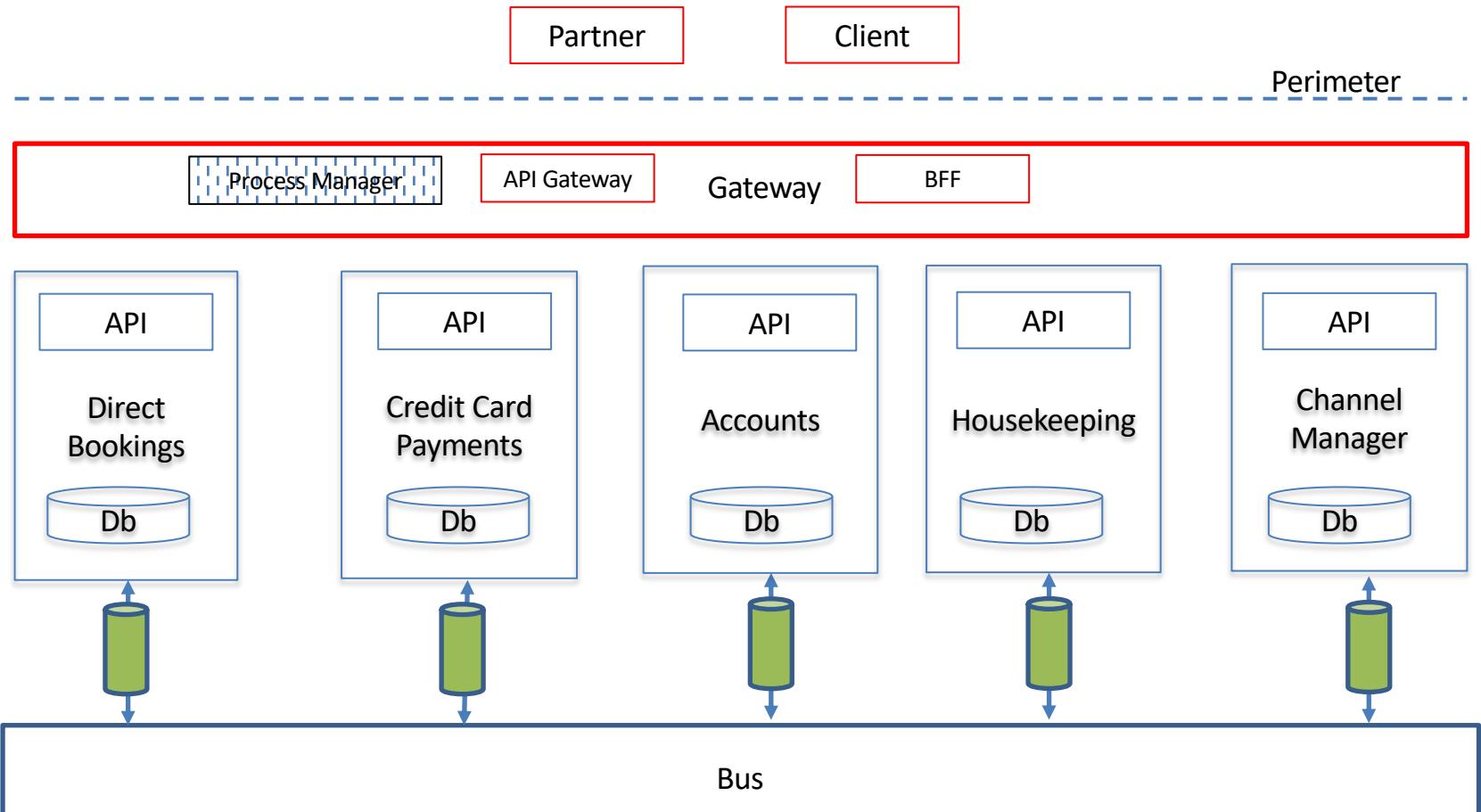
A sequence of screens can build up a request

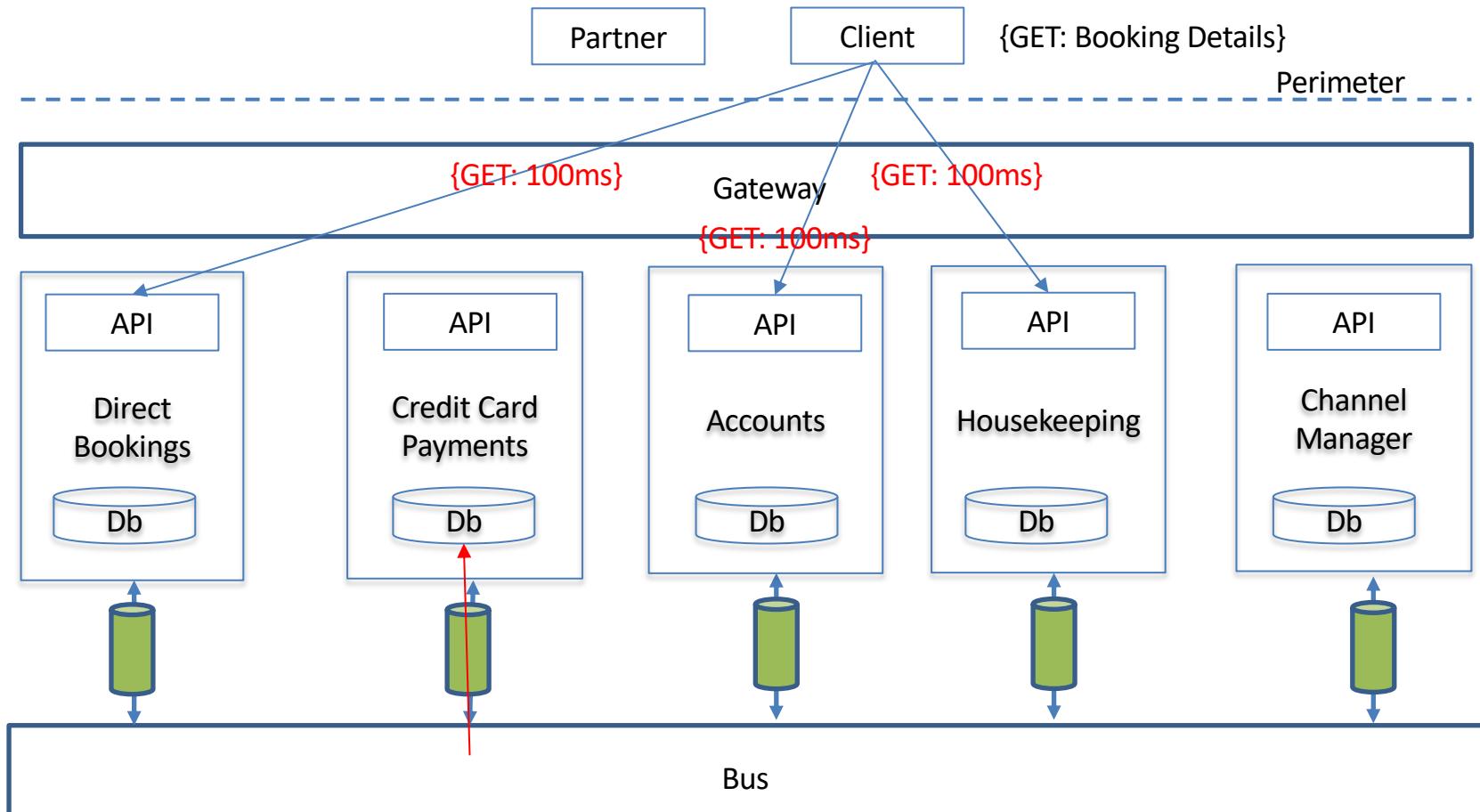
A sequence of screens can build up a request

At this point, the user expects async

Getting work done in a Monolith



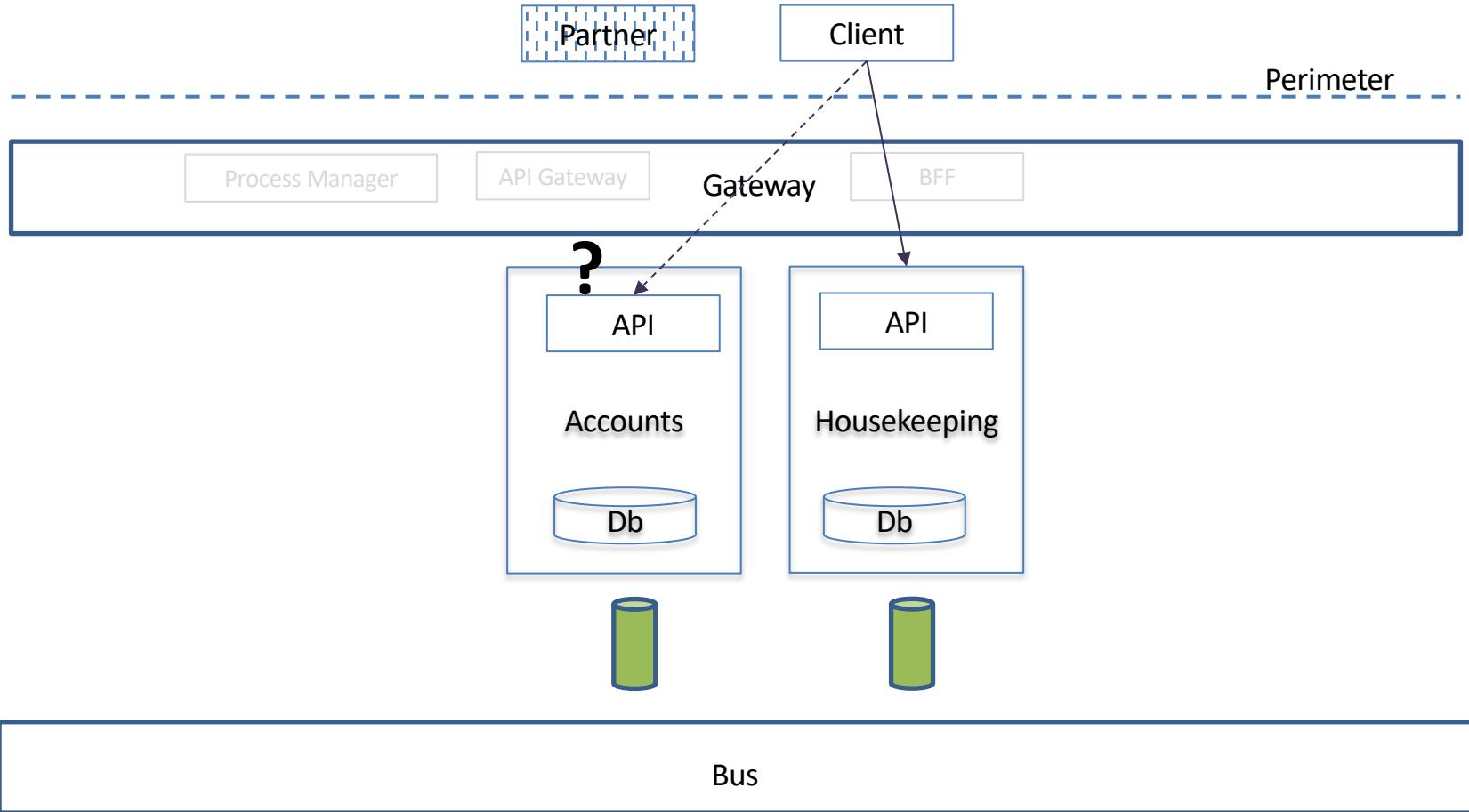


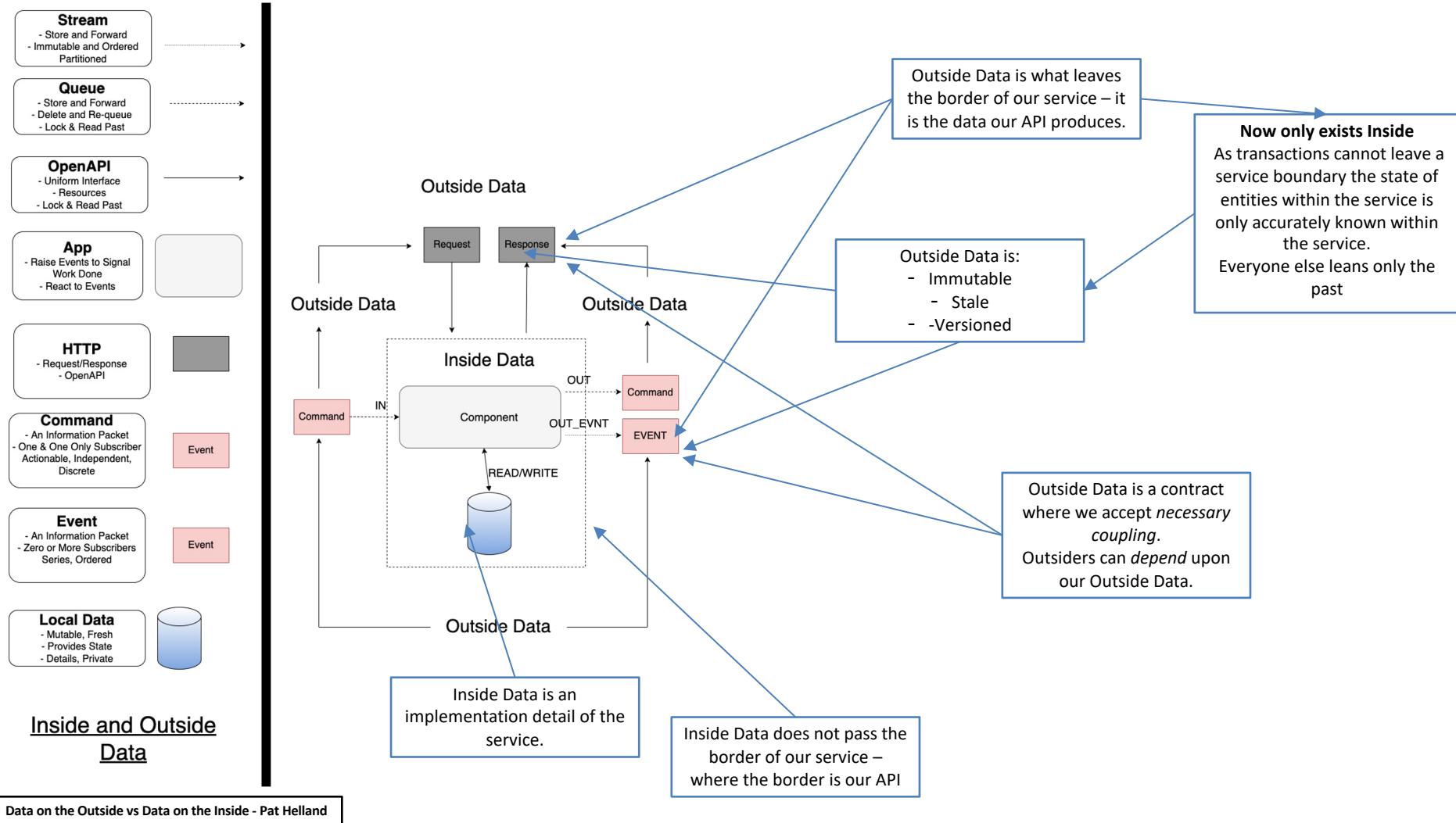




Check-In

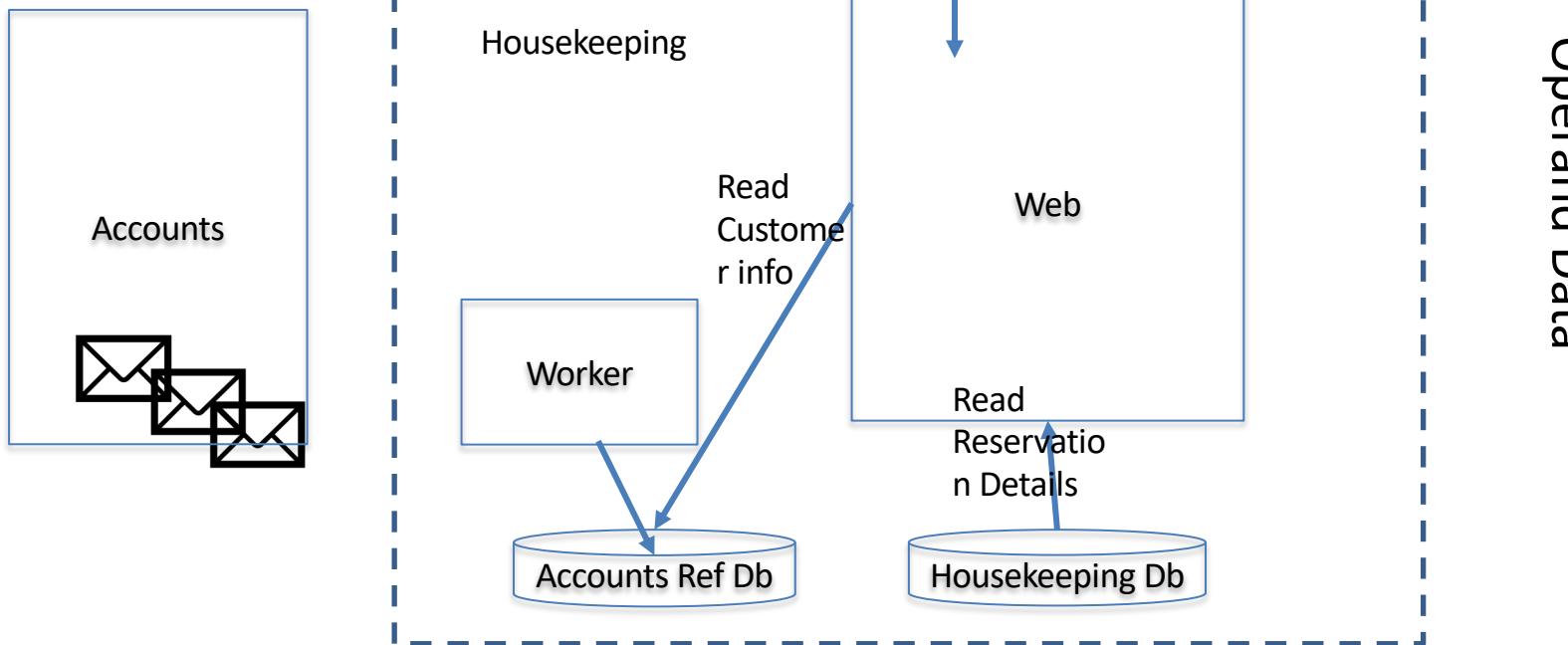
Check-In





Event Carried State Transfer

Martin Fowler



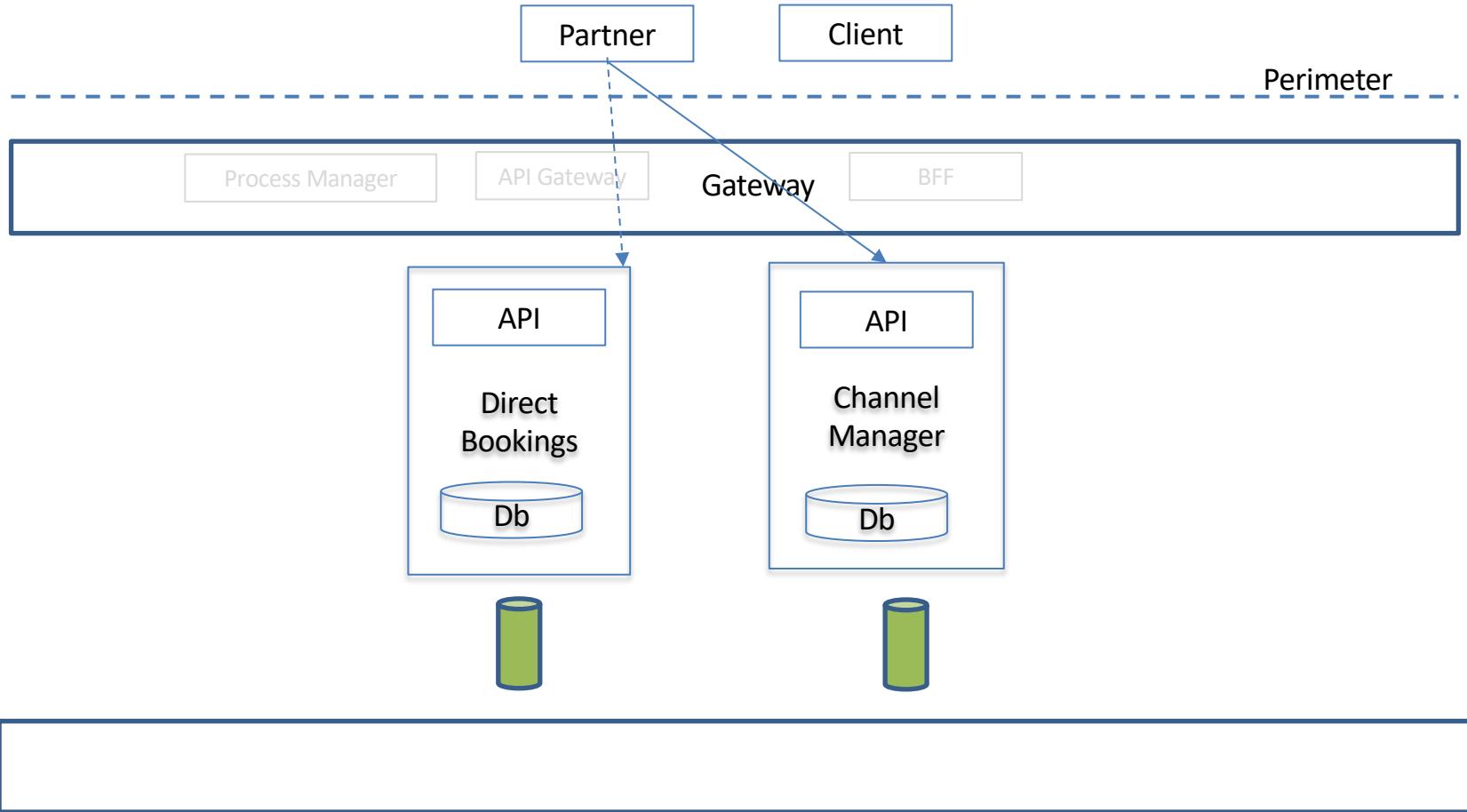
Operand Data



Illustration by Chris Gash

Channel Effectiveness

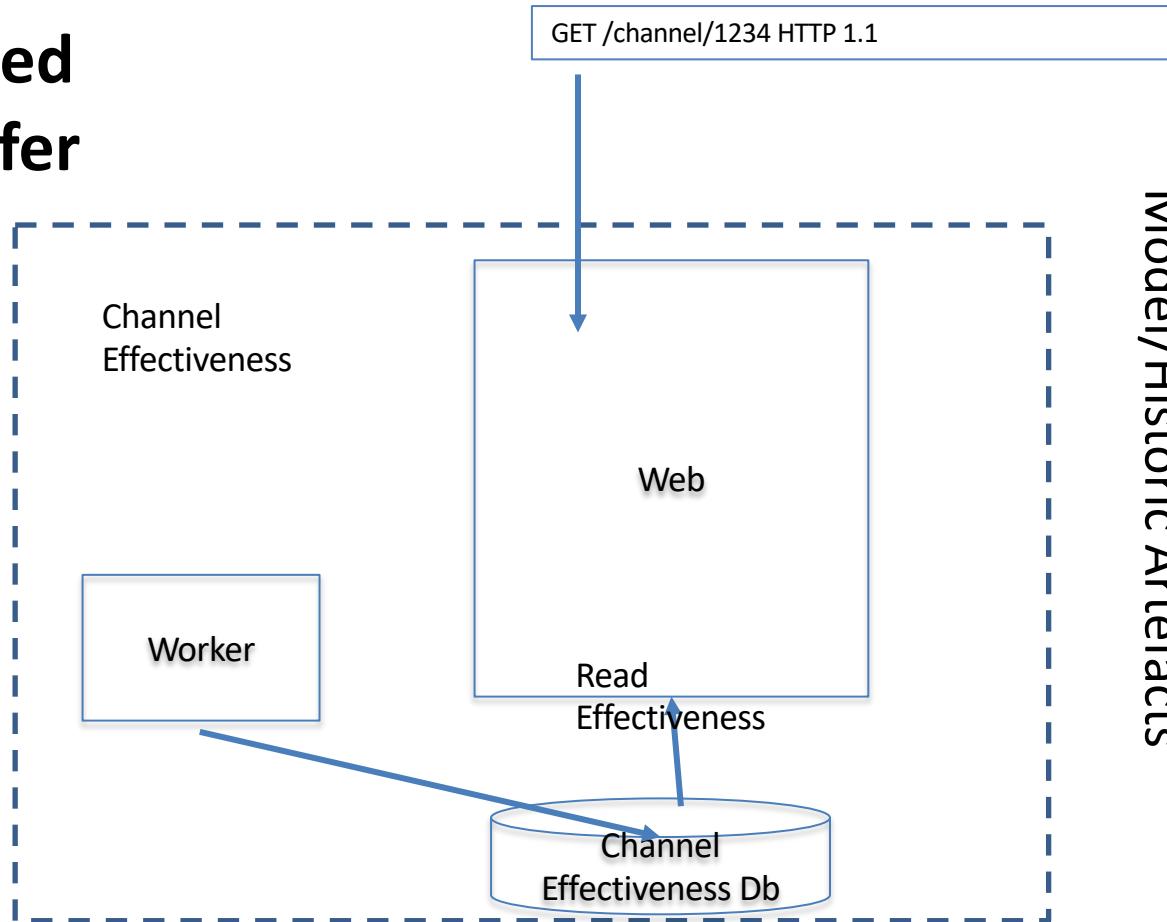
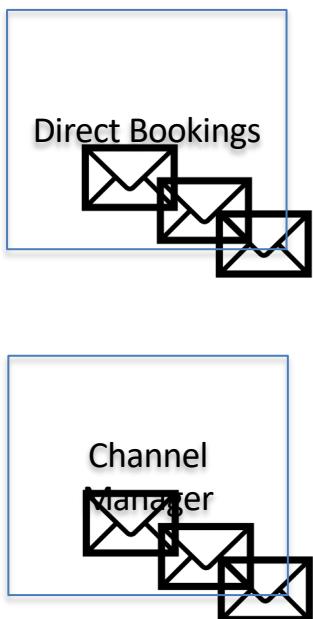
Channel Effectiveness



Composite View Model/Historic Artefacts

Event Carried State Transfer

Martin Fowler

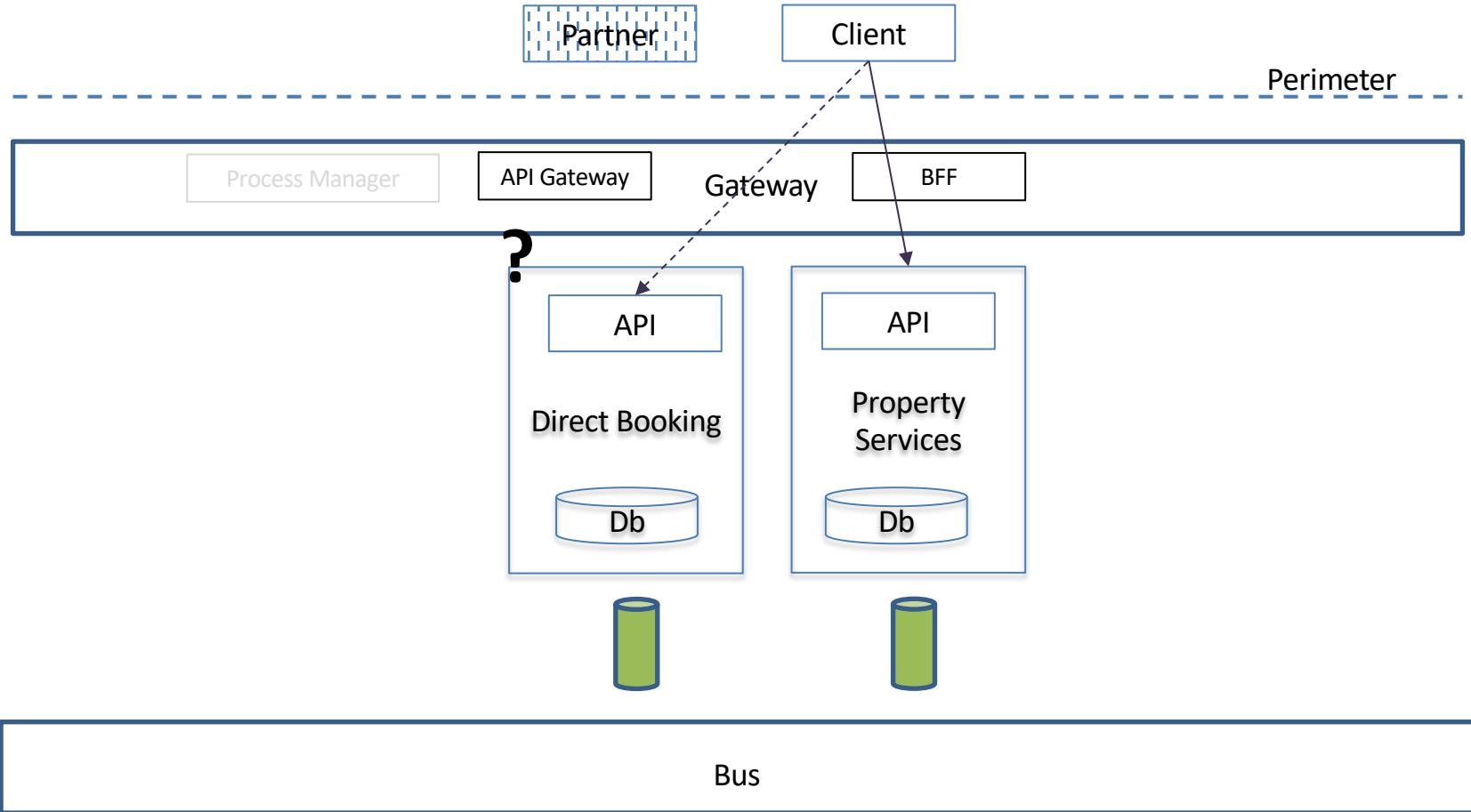




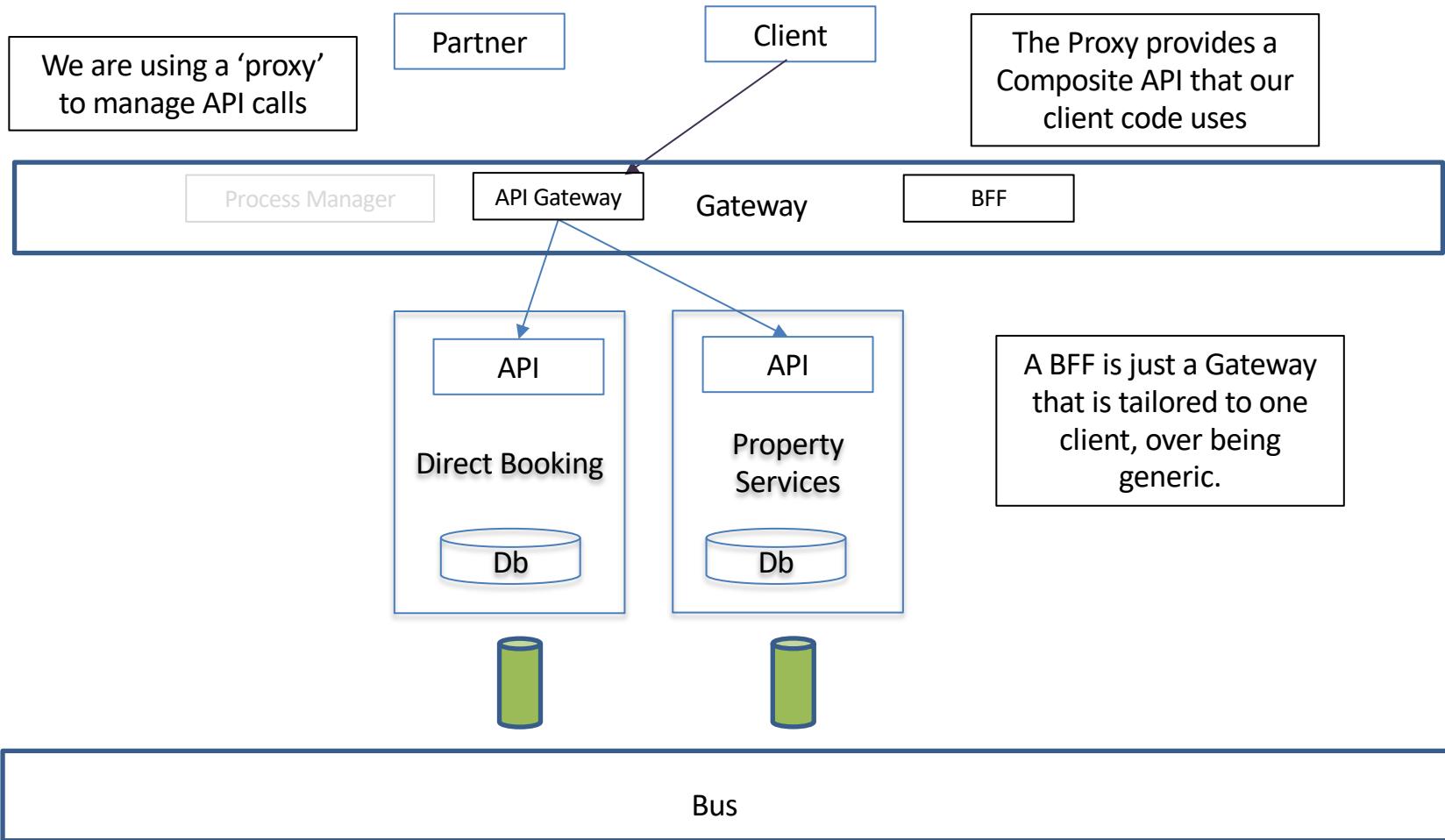
© Can Stock Photo - csp12541701

Hotel Group Search

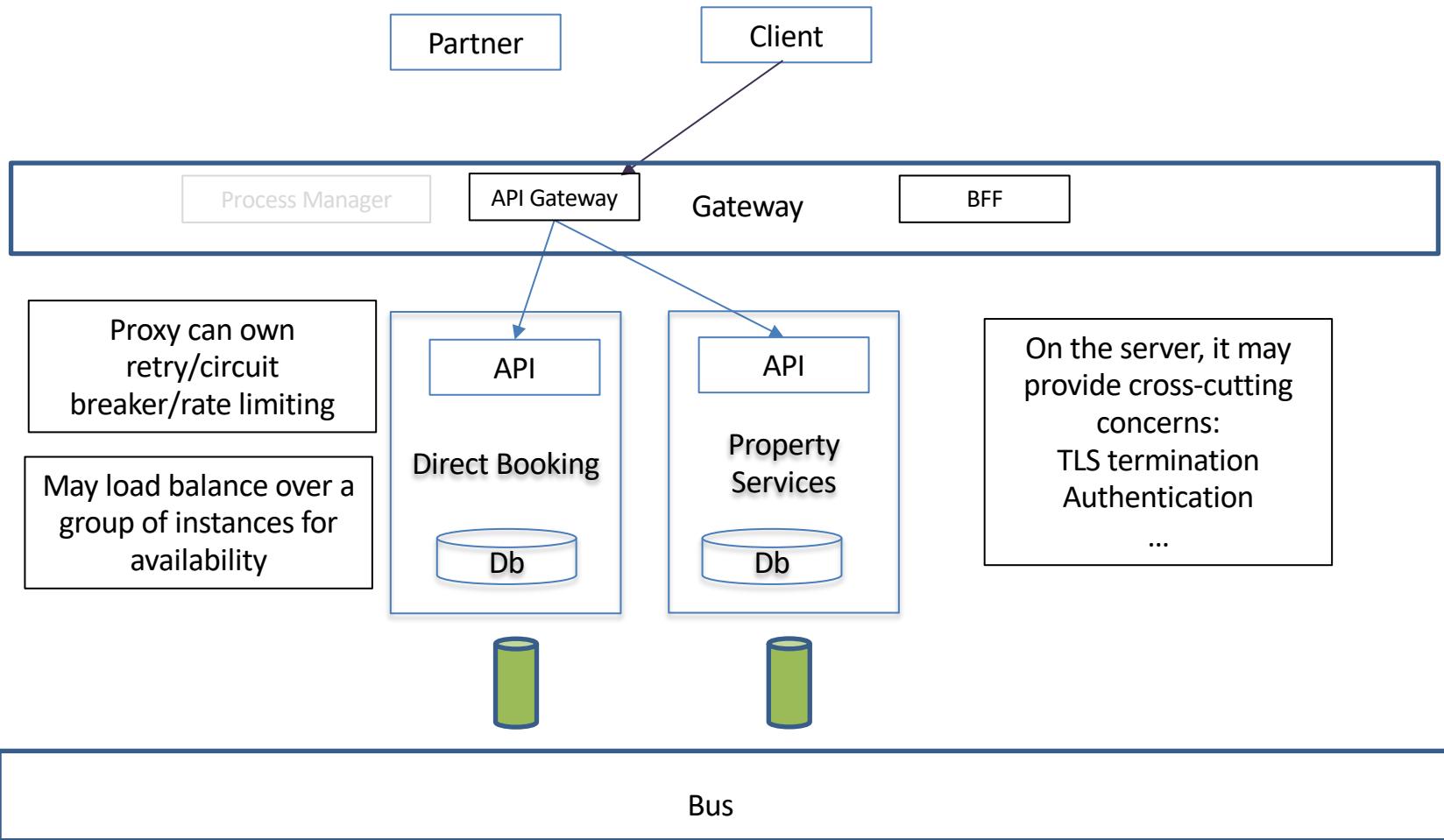
Group Room Search



Composition



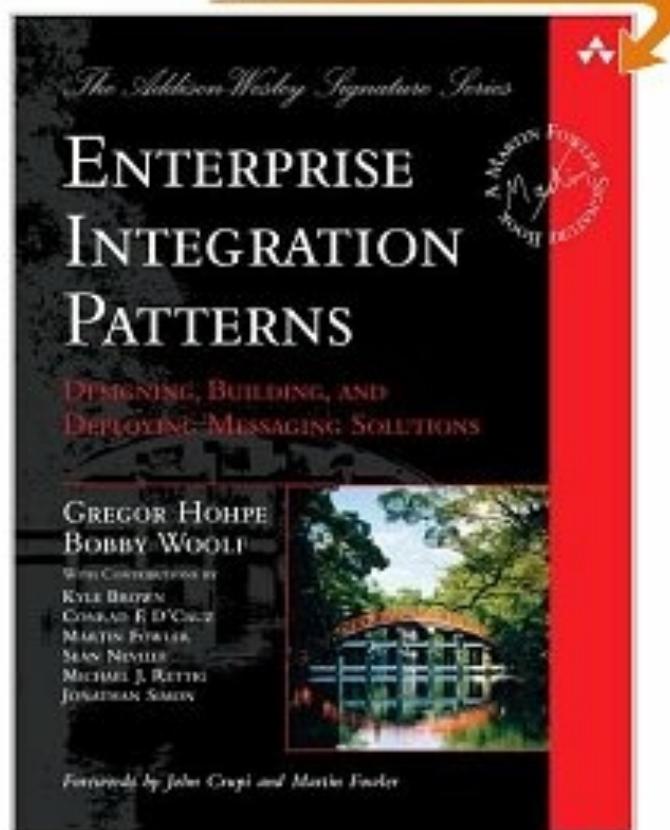
Composition



10. NEXT STEPS

Further Reading

[Click to LOOK INSIDE!](#)



Q&A