It’s like 10,000 streams when what you need is a queue



chrissimon.au
chrissimon-au
@chrissimon.au

It's like 10,000 streams when what you need is a queue






Alanis Morissette - Ironic (1996)
♪ It's a black fly
in your Chardonnay ♪
chrissimon.au


Alanis Morissette - Ironic (1996)
♪ A traffic jam when
you're already late ♪
chrissimon.au


Alanis Morissette - Ironic (1996)
♪ It's like 10,000 spoons
when all ya need is a knife ♪
chrissimon.au


♪ It's like 10,000 STREAMS
when all ya need is a QUEUE
chrissimon.au



Event-Driven Architecture

Events



Events



Events



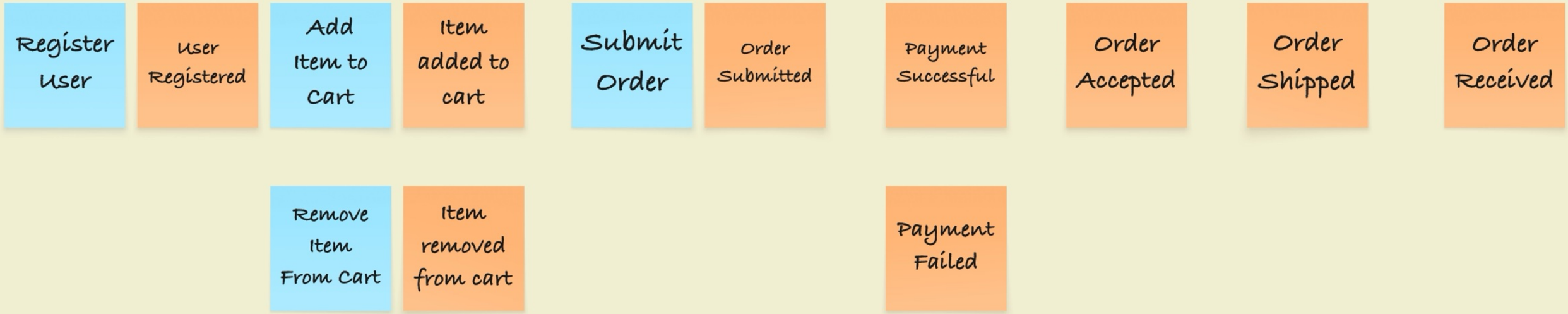
Events



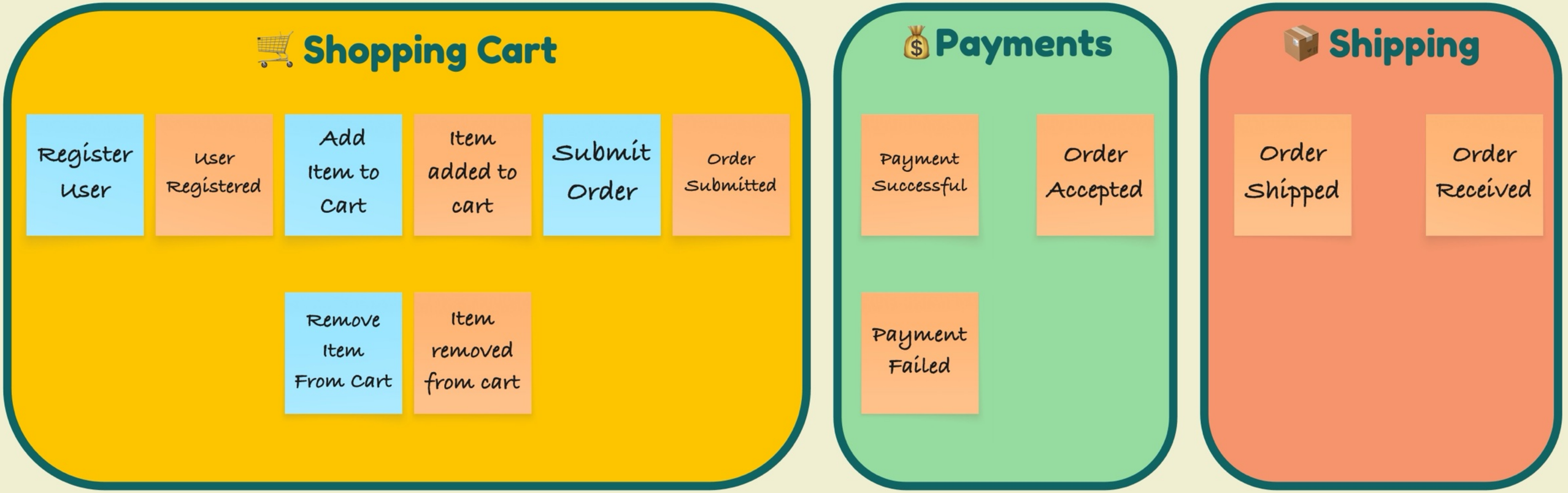
Commands

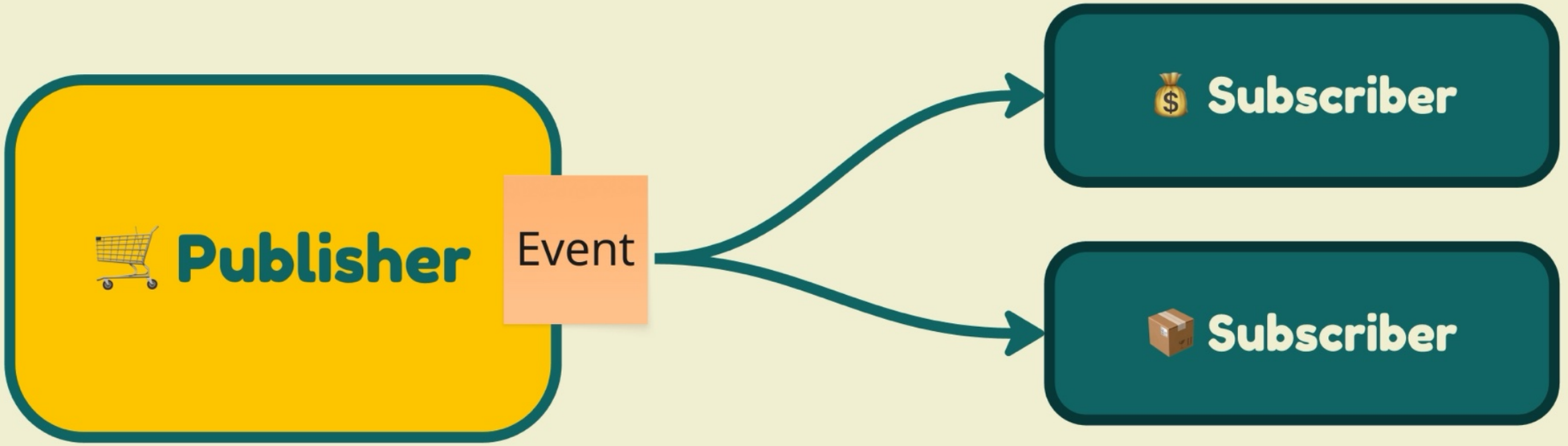


Events

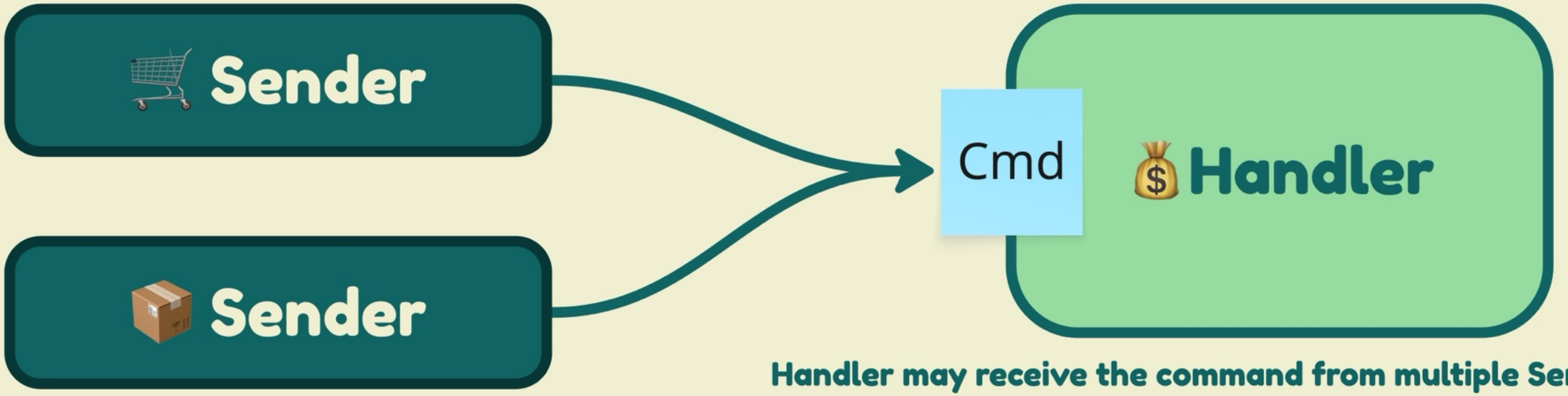

Register User
User Registered
Add Item to Cart
Item added to cart
Submit Order
Order Submitted
Payment Successful
Order Accepted
Order Shipped
Order Received
Remove Item From Cart
Item removed from cart
Payment Failed

Events


🛒 Shopping Cart
Register User
User Registered
Add Item to Cart
Item added to cart
Submit Order
Order Submitted
Remove Item From Cart
Item removed from cart
💰 Payments
Payment Successful
Order Accepted
Payment Failed
📦 Shipping
Order Shipped
Order Received

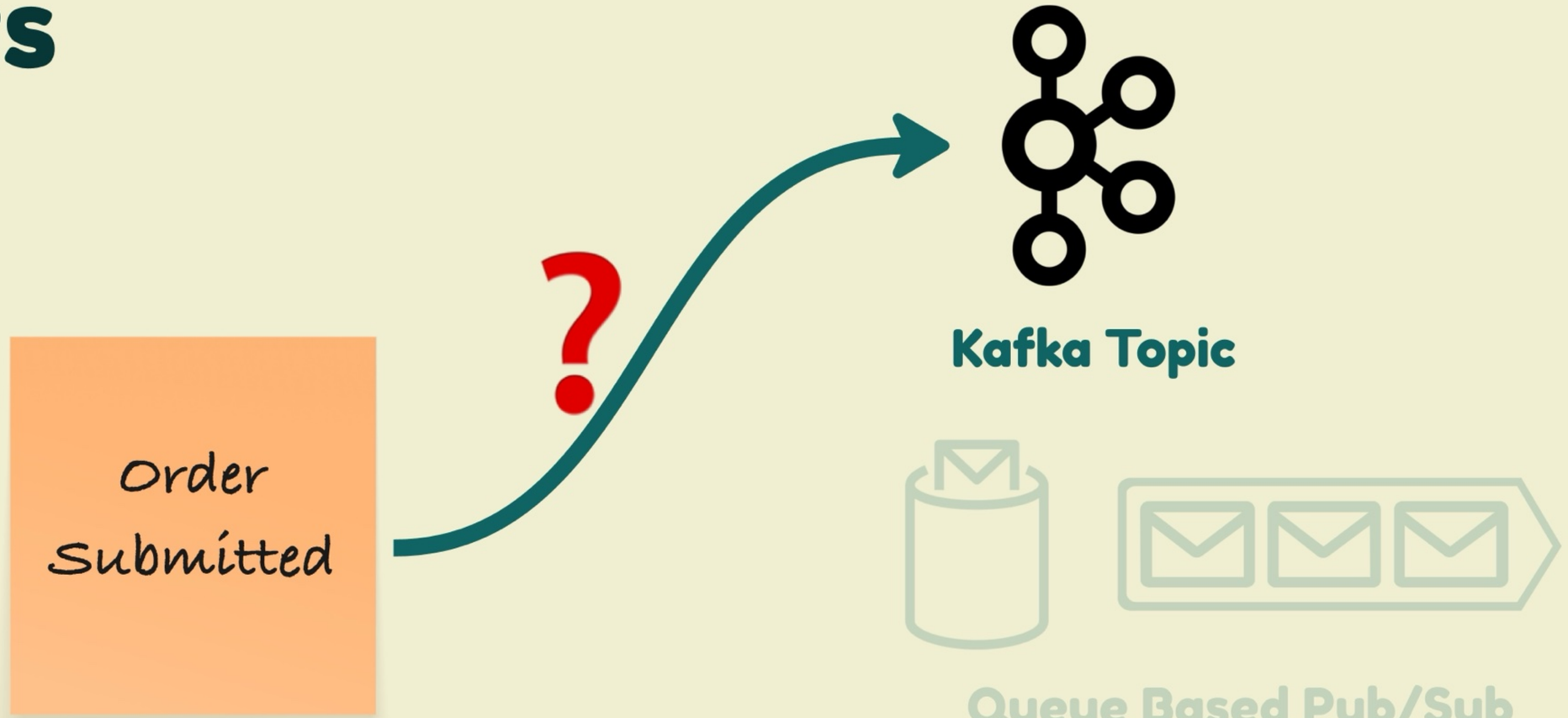


Publisher may have multiple Subscribers - it's not coupled to any of them, but they are coupled to it.

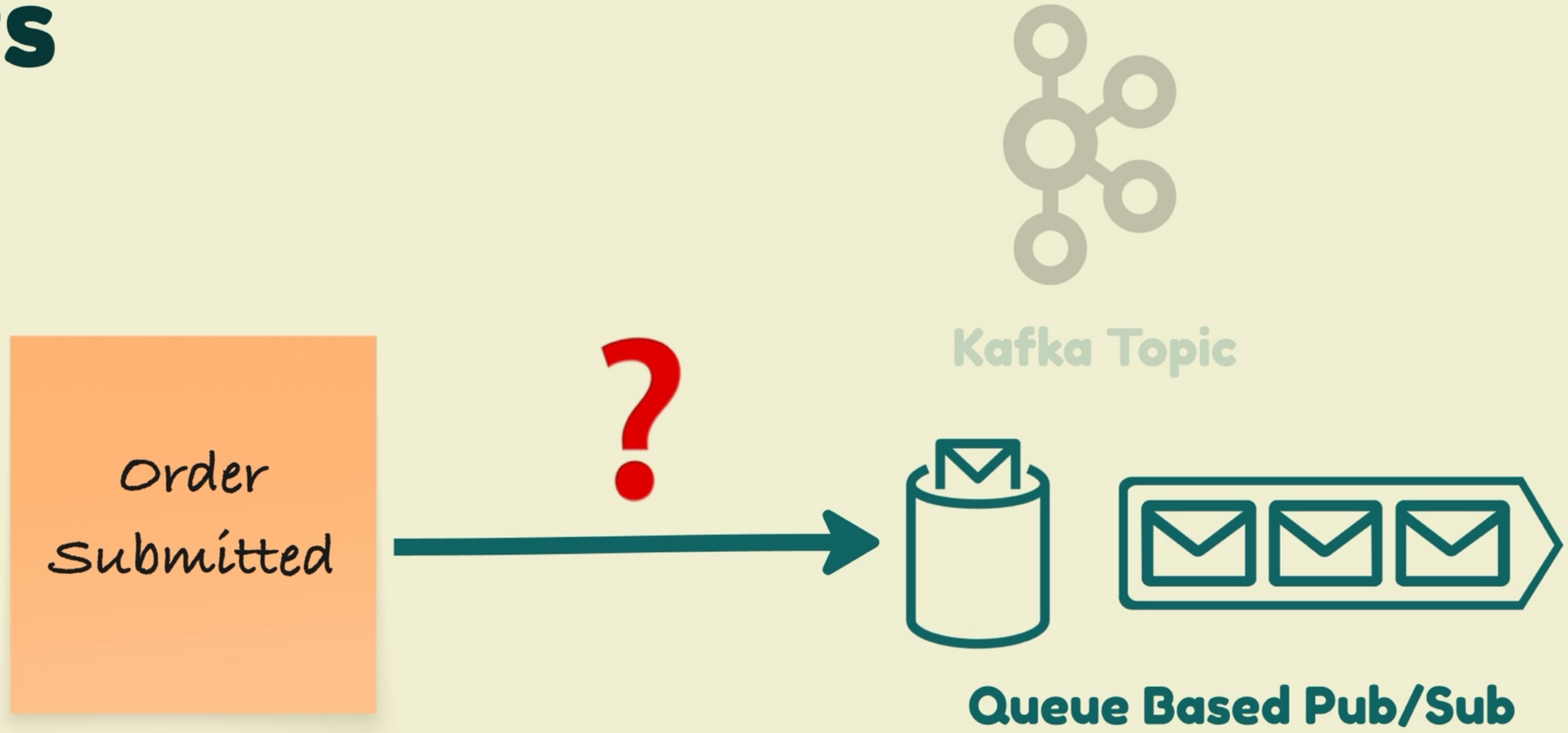


Handler may receive the command from multiple Senders - it's not coupled to any of them, but they are coupled to it.

Events

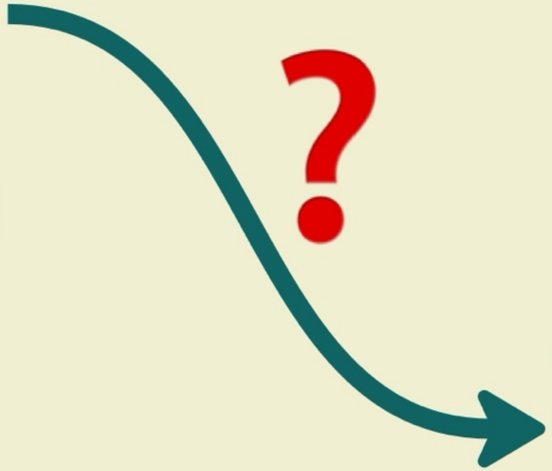

Order Submitted
?
Kafka Topic
Queue Based Pub/Sub
EntityStream
1 2023-11-01T01:01:10 OrderShipped <guid> { "field": "value" }
2 2023-11-01T01:02:10 PaymentSuccessful <guid> { "field": "value" }
Event Store

Events


Kafka Topic
Order Submitted
?
Queue Based Pub/Sub
EntityStream
1
2023-11-01T01:01:10
OrderShipped
<guid>
{ "field": "value" }
2
2023-11-01T01:02:10
PaymentSuccessful
<guid>
{ "field": "value" }
Event Store

Events







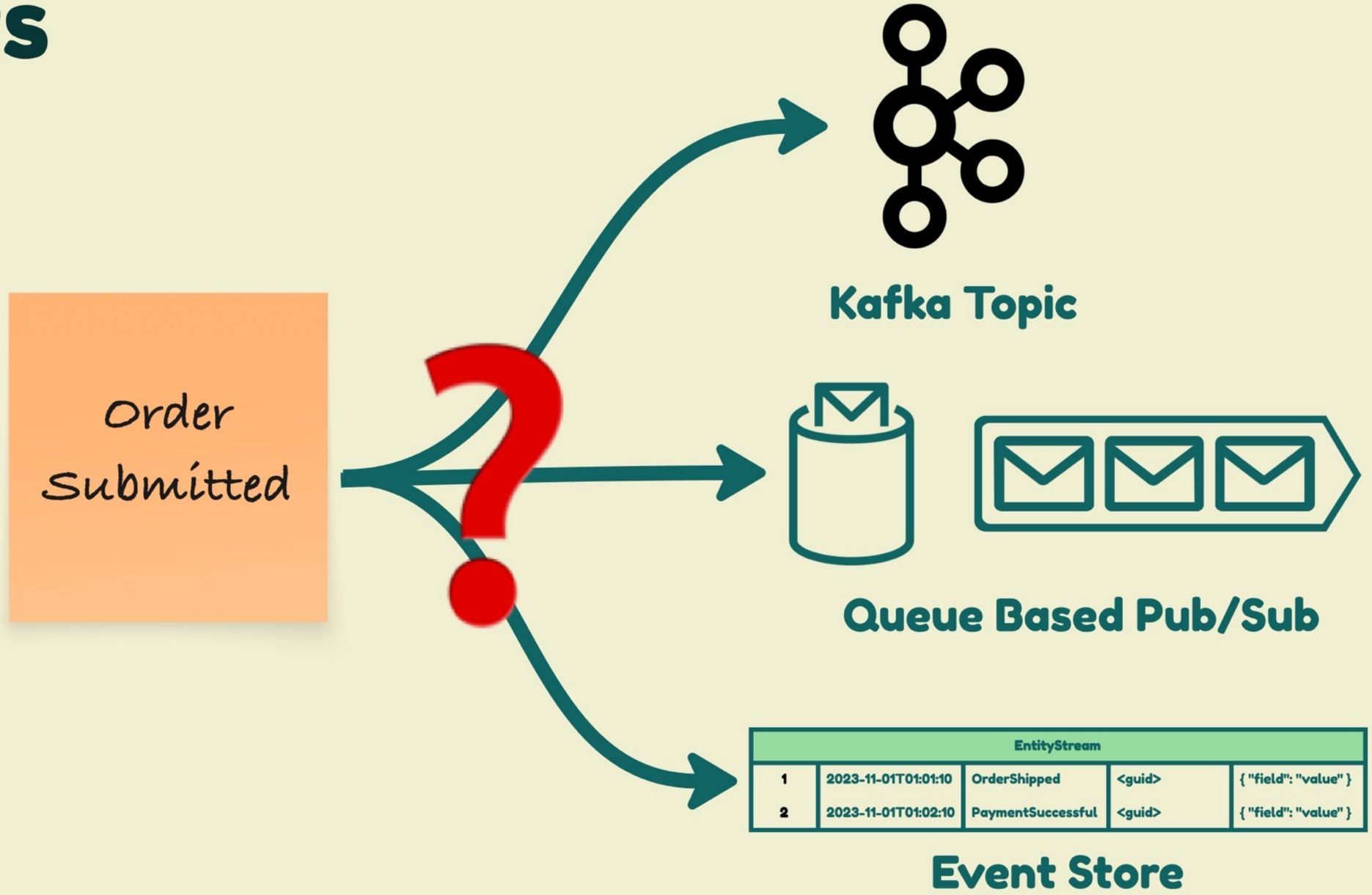
Kafka Topic

Queue Based Pub/Sub

EntityStream				
1	2023-11-01T01:01:10	OrderShipped	<guid>	{ "field": "value" }
2	2023-11-01T01:02:10	PaymentSuccessful	<guid>	{ "field": "value" }

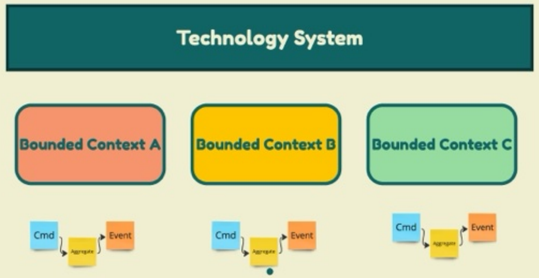
Event Store

Events

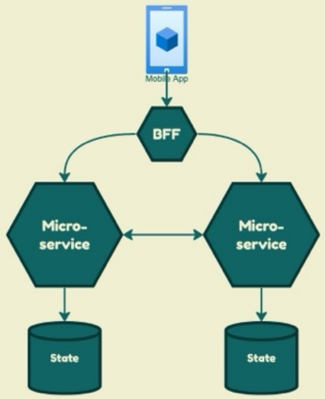

Order Submitted
Kafka Topic
Queue Based Pub/Sub
EntityStream
1 2023-11-01T01:01:10 OrderShipped <guid> { "field": "value" }
2 2023-11-01T01:02:10 PaymentSuccessful <guid> { "field": "value" }
Event Store

Kruchten's 4+1 Views

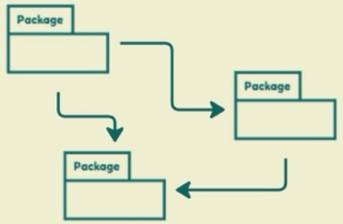
Logical View



Process View

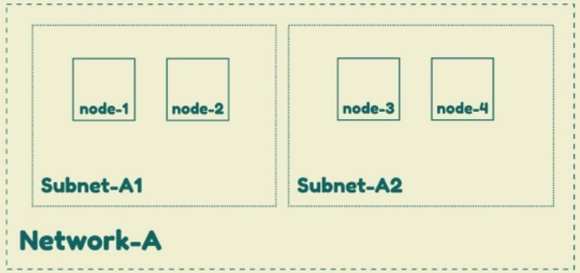


Development View



Scenario View

Physical View

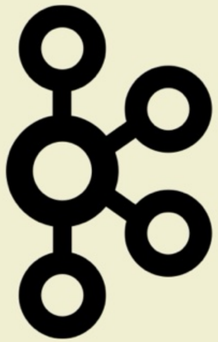


Different Views

Logical View

Order Submitted

Process View



Kafka Topic



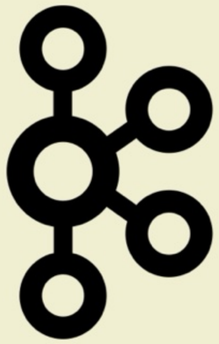
Queue Based Pub/Sub

EntityStream				
1	2023-11-01T01:01:10	OrderShipped	<guid>	{ "field": "value" }
2	2023-11-01T01:02:10	PaymentSuccessful	<guid>	{ "field": "value" }

Event Store

To decide, we need to understand

Process View



Kafka Topic

vs.



Queue Based Pub/Sub

later

EntityStream				
1	2023-11-01T01:01:10	OrderShipped	<guid>	{ "field": "value" }
2	2023-11-01T01:02:10	PaymentSuccessful	<guid>	{ "field": "value" }

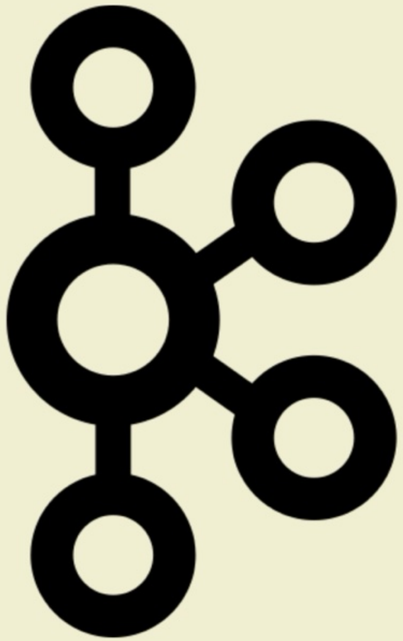
Event Store

Queues vs Streams



Queue Based Pub/Sub

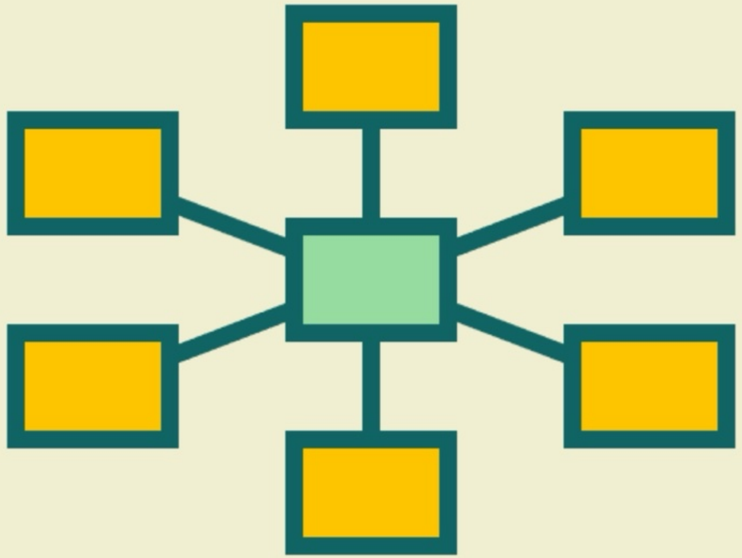
vs



RabbitMq
AWS SNS/SQS
Azure Service Bus
Google Pub/Sub (ish)

Kafka
AWS Kinesis
Azure Event Hubs
Google Pub/Sub (ish)

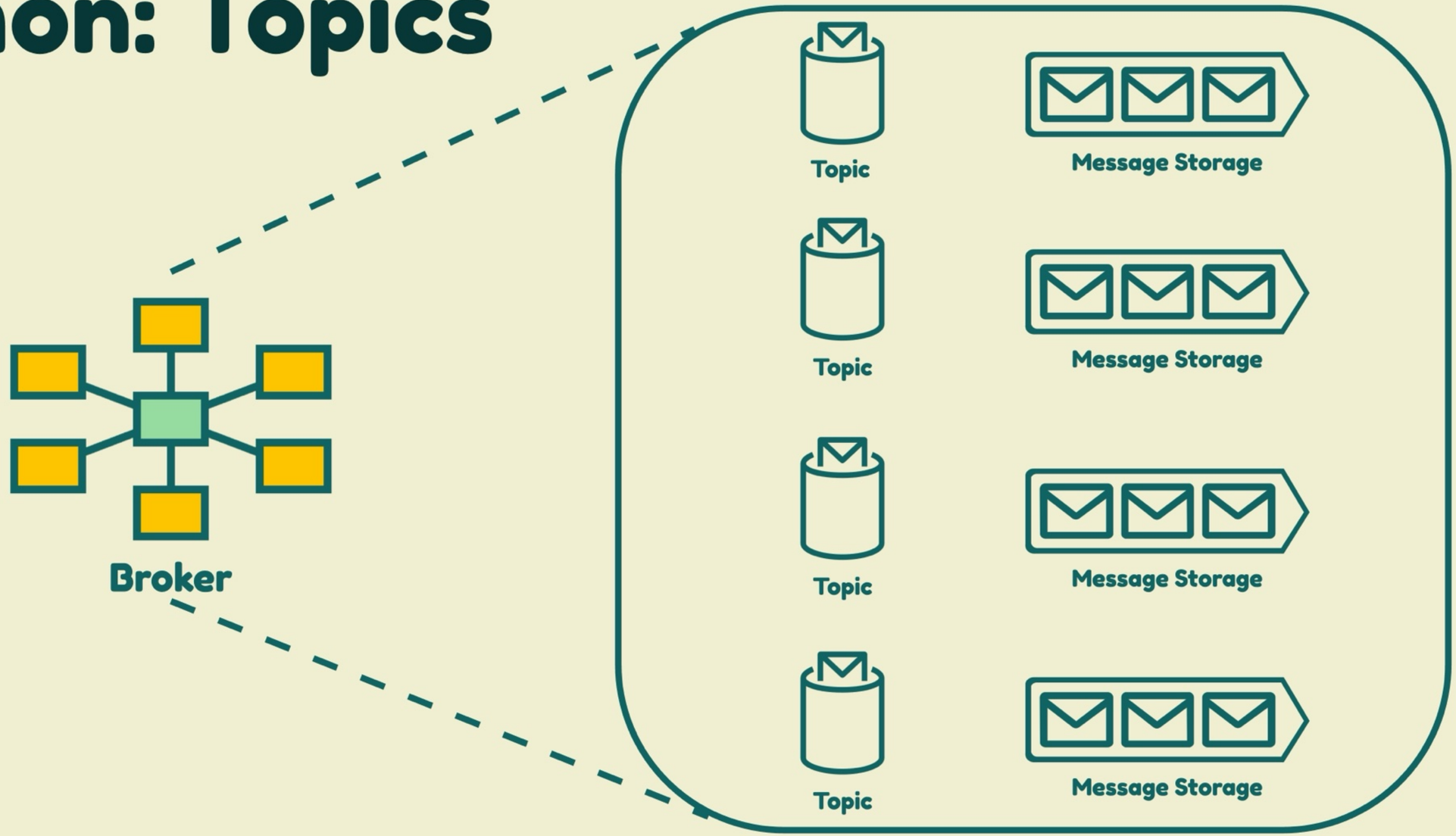
What do they have in common?



Broker

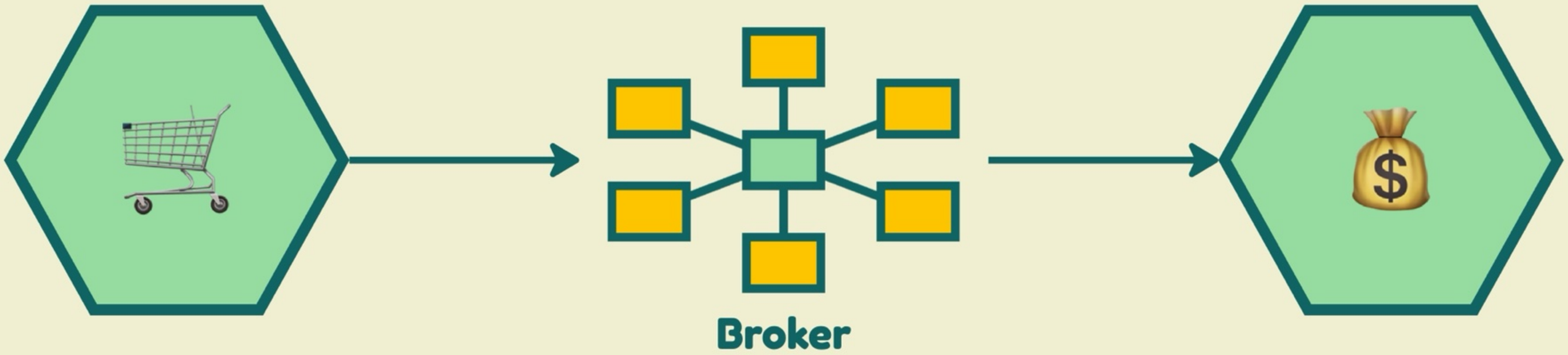
They are all considered "message brokers" - a generic term for anything that stores and facilitates message distribution

Common: Topics



Brokers organise messages according to 'topics' (aka 'channels' / 'exchanges') and are able to store messages for a period of time.

Common: Producers & Consumers



"Producers" (🛒) push messages into the broker.

"Consumers" (💰) consume messages from the broker.

Common: Logical Message Intent

Logical View

Sender
Sends Command

Command

Handler
Handles Command

Publisher
Publishes Event

Event

Subscriber
Handles Event

Process View

Producer
Produces Message

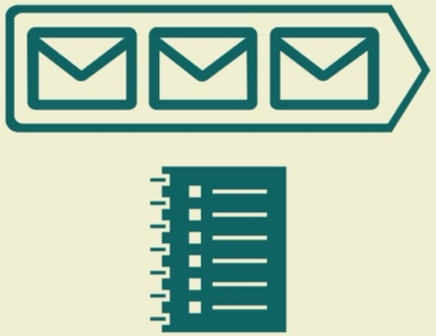


Consumer
Consumes Message

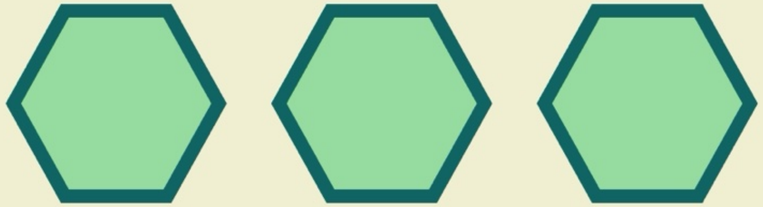
Common: Delivery Guarantees

	Producer Produces	Consumer Consumes
At Most Once		Msg 2 is dropped 
Exactly Once		
At Least Once		 Msg 1 is Consumed Twice

What is different?



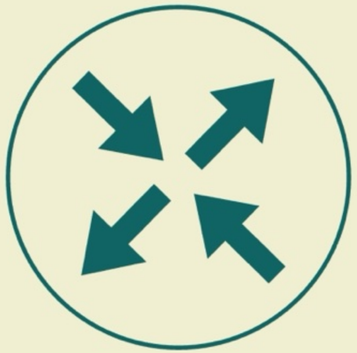
Message Distribution



Consumer Scale-out

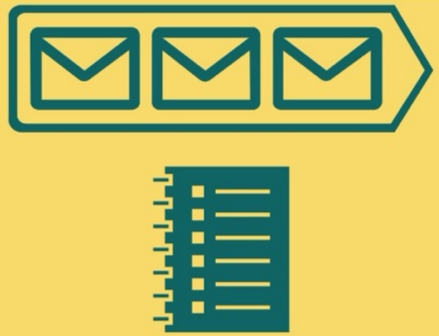


Message Processing Failure

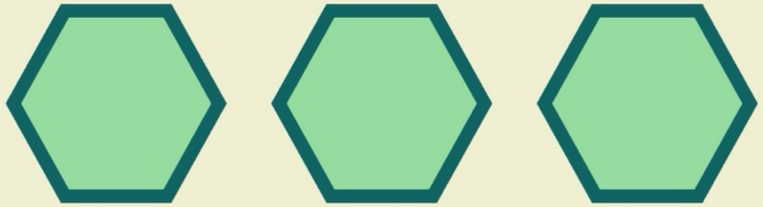


Message Routing

What is different?



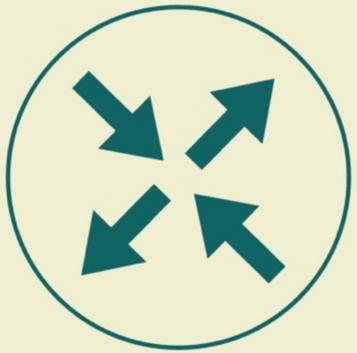
Message Distribution



Consumer Scale-out



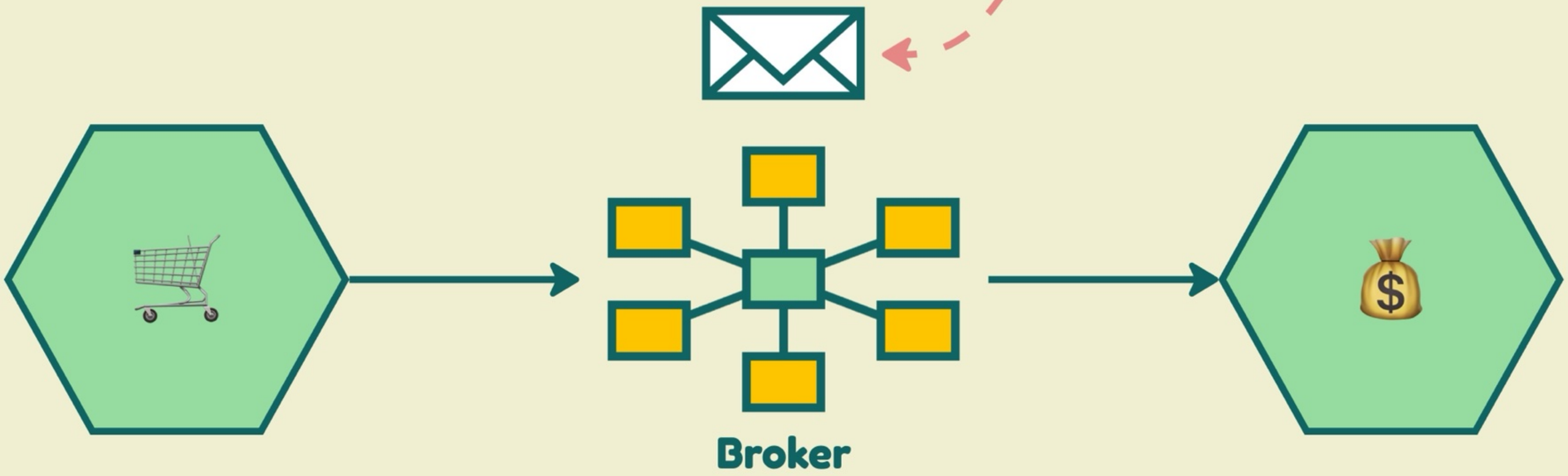
Message Processing Failure



Message Routing

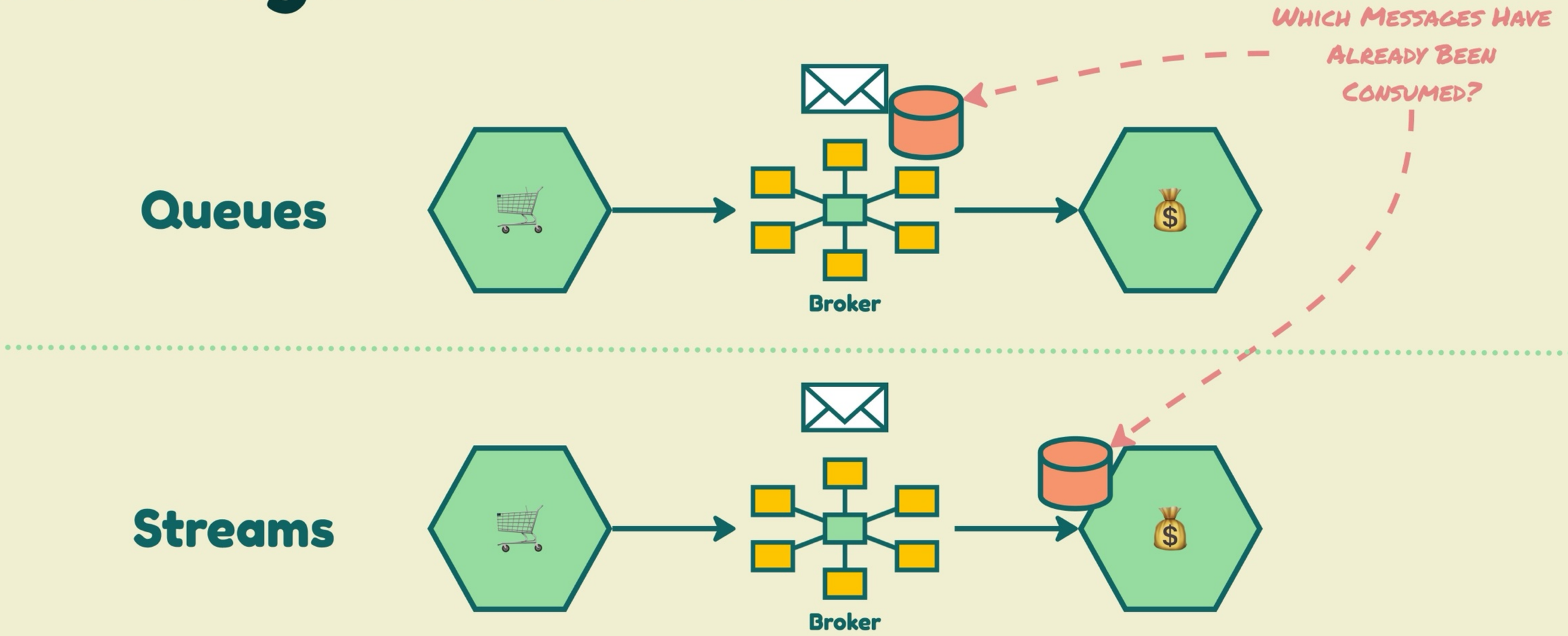
Message Distribution

Who decides if Payments 💰 should receive this message?



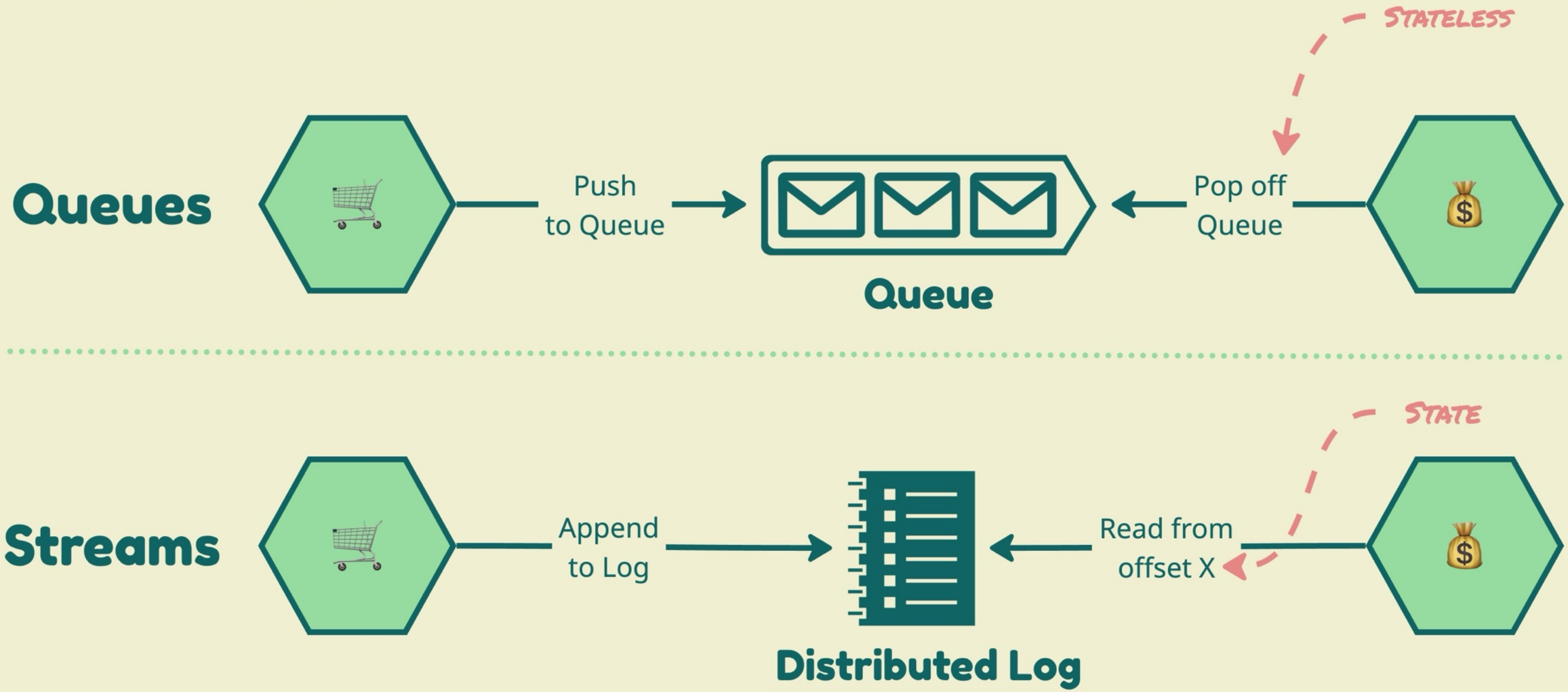
Queues & Streams Manage Message State Differently

Message Distribution



Message State Management

Message Distribution



Produce & Consume Interactions

Message Distribution

Queues



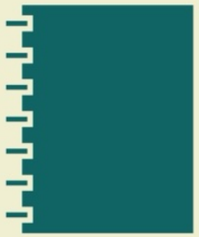


Queue

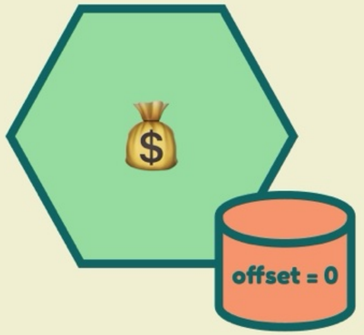


Streams



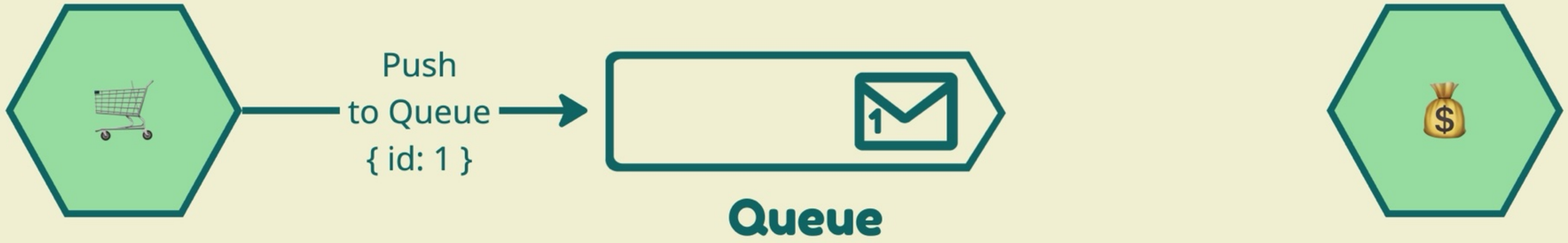


Distributed Log

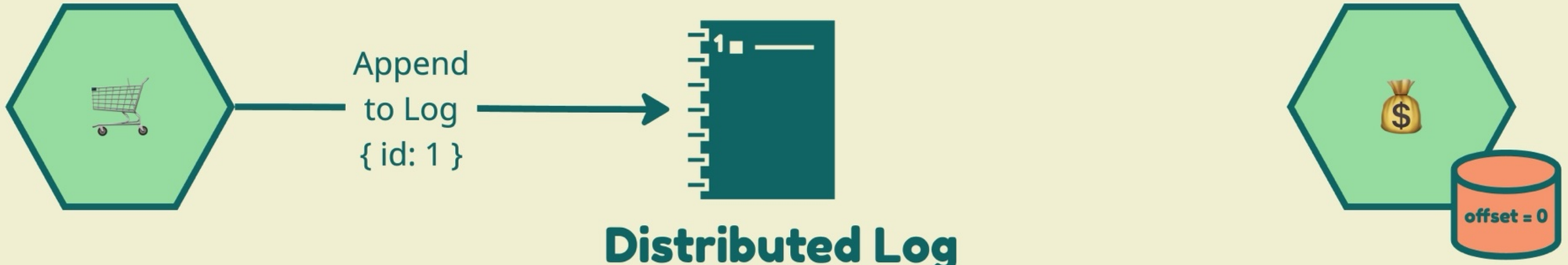


Message Distribution

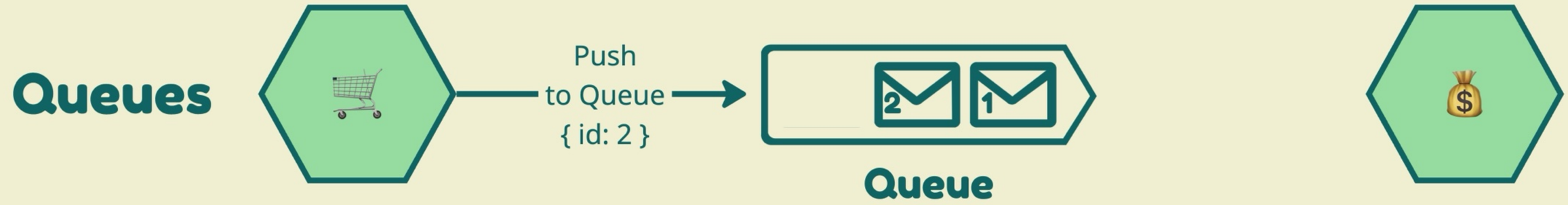
Queues

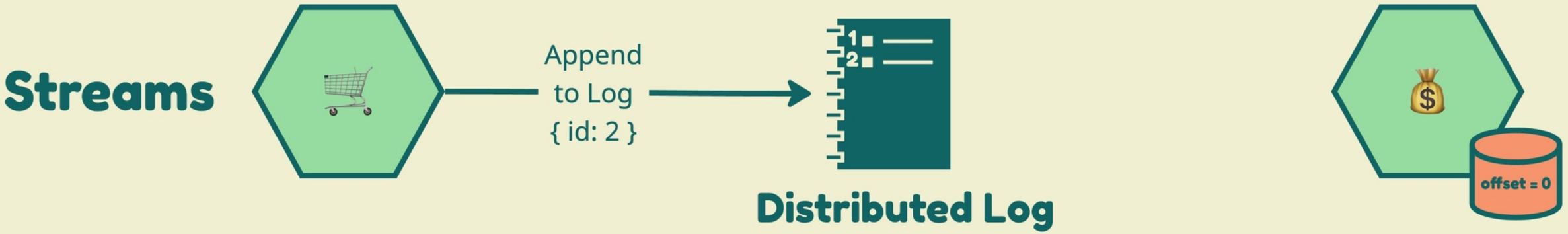


Streams



Message Distribution

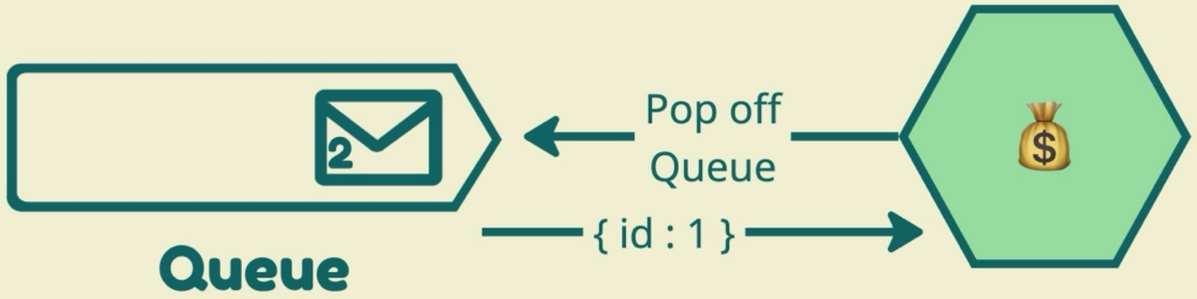




Message Distribution

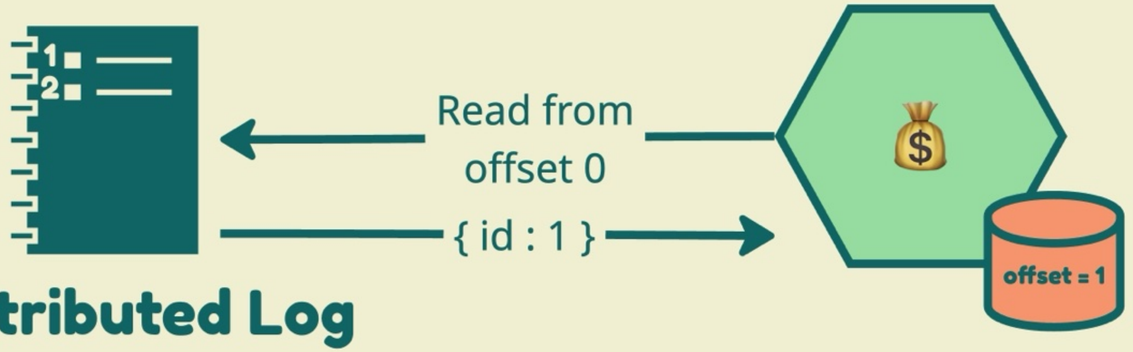
Queues





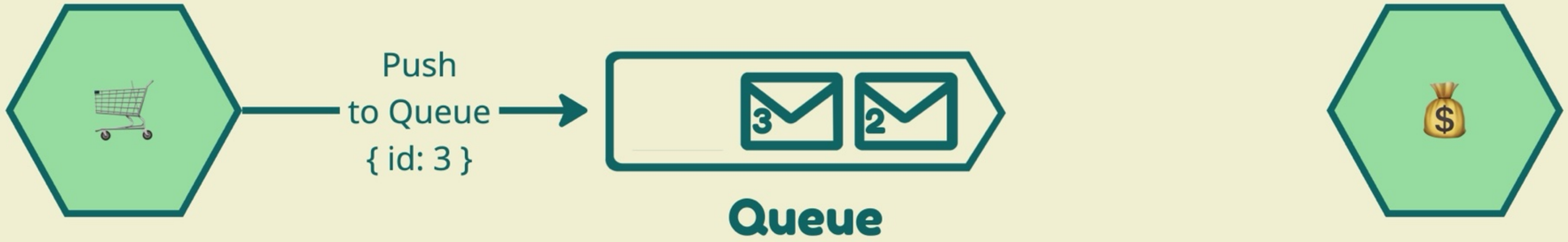
Streams



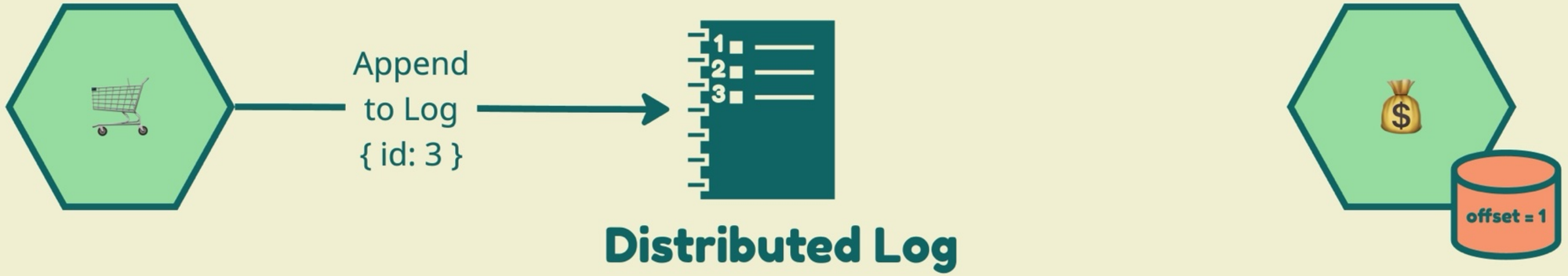


Message Distribution

Queues



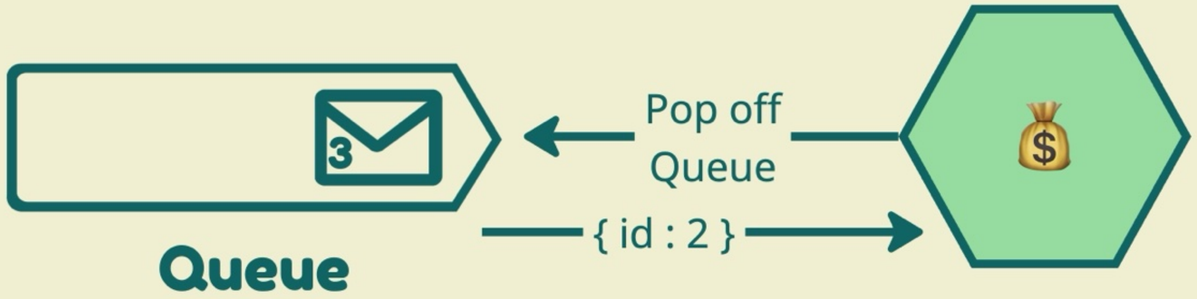
Streams



Message Distribution

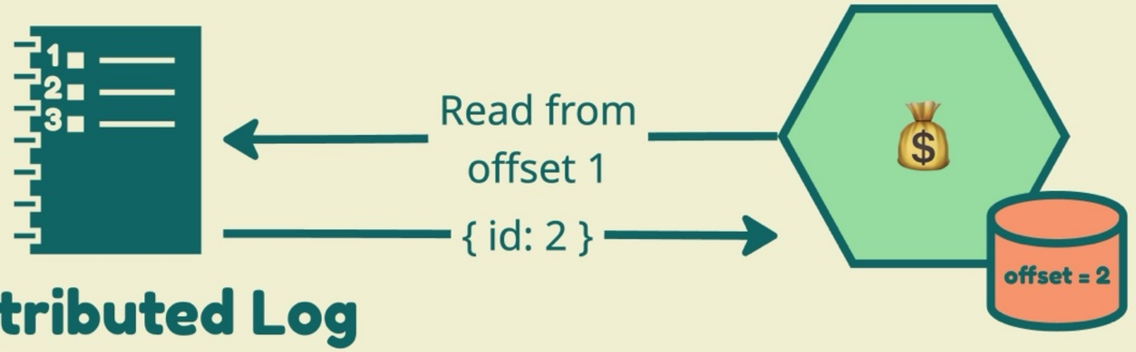
Queues





Streams

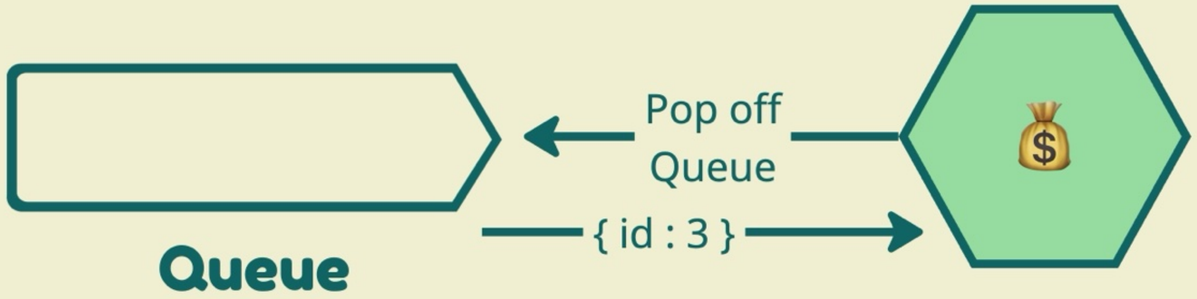




Message Distribution

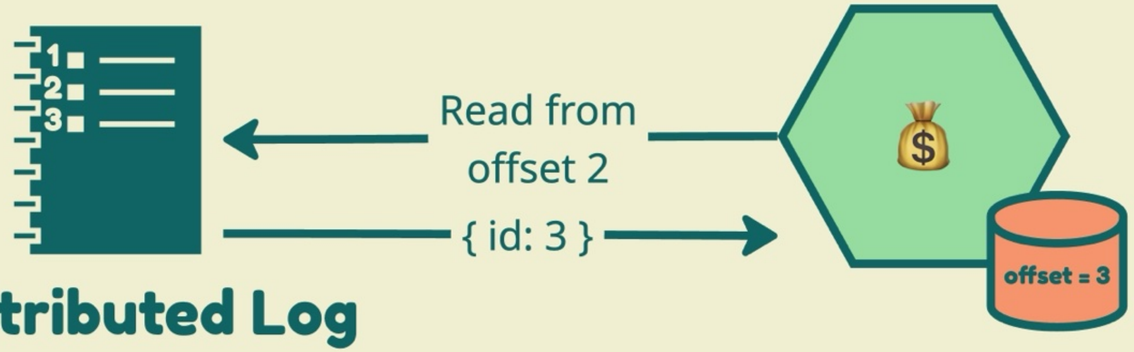
Queues





Streams





Message Distribution

Queue is empty (That's the "State")

Queues



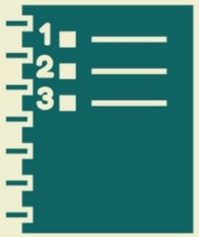
Queue



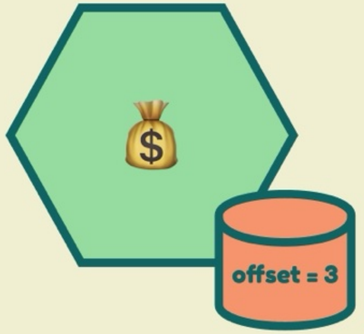
Streams



Distributed Log still has all entries (it's not tracking "read state")



Distributed Log



Message Distribution

Queues





Queue

Queue is empty (That's the "State")

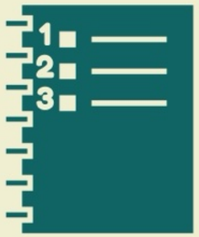
Stateless Consumer



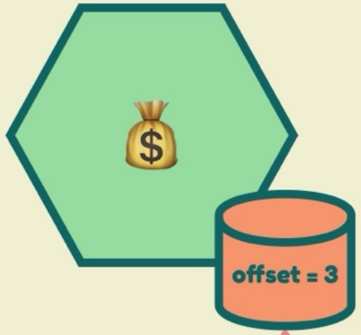
Streams



Distributed Log still has all entries (it's not tracking "read state")



Distributed Log



Stateful Consumer

Message Distribution

Queues



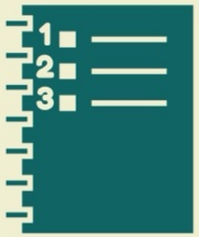


Queue

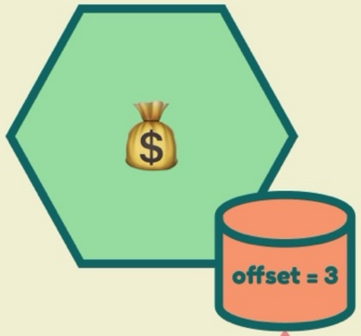


Streams





Distributed Log



Yes, Kafka client libraries store this state back into the __consumer_offsets topic in Kafka itself - but it's managed by the client library, not by the server

CONFLUENT
an IBM Company

WHY CONFLUENT PRODUCTS PRICING SOLUTIONS RESOURCES Try Confluent

Data in Motion – Definition, Use Cases & Challenges

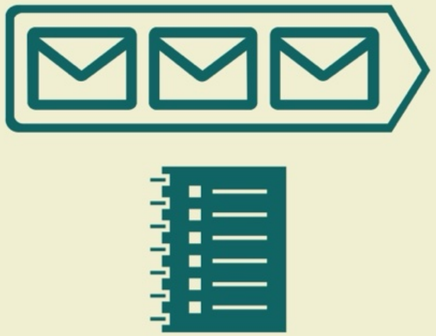
Why are leading enterprises embracing data in motion? With data in motion, businesses gain a powerful competitive edge—immediate insights, operational efficiency, and the ability to scale confidently. The Confluent data streaming platform powered by Apache Kafka® and Apache Flink® drives critical outcomes, such as real-time inventory visibility, fraud prevention, AI-powered search, faster forecasting, and seamless personalization. By replacing static data pipelines and legacy batch systems, Confluent unifies fragmented systems, improves agility, and accelerates innovation.


Alanis Morissette - Ironic (1996)
♪ And isn't it ironic... ♪
chrissimon.au



chrissimon.au

What is different?



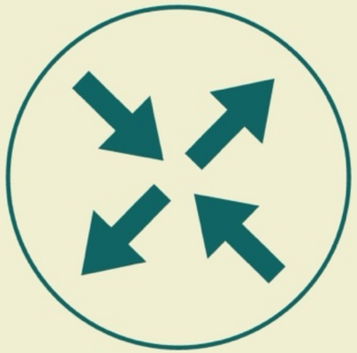
Message Distribution



Consumer Scale-out

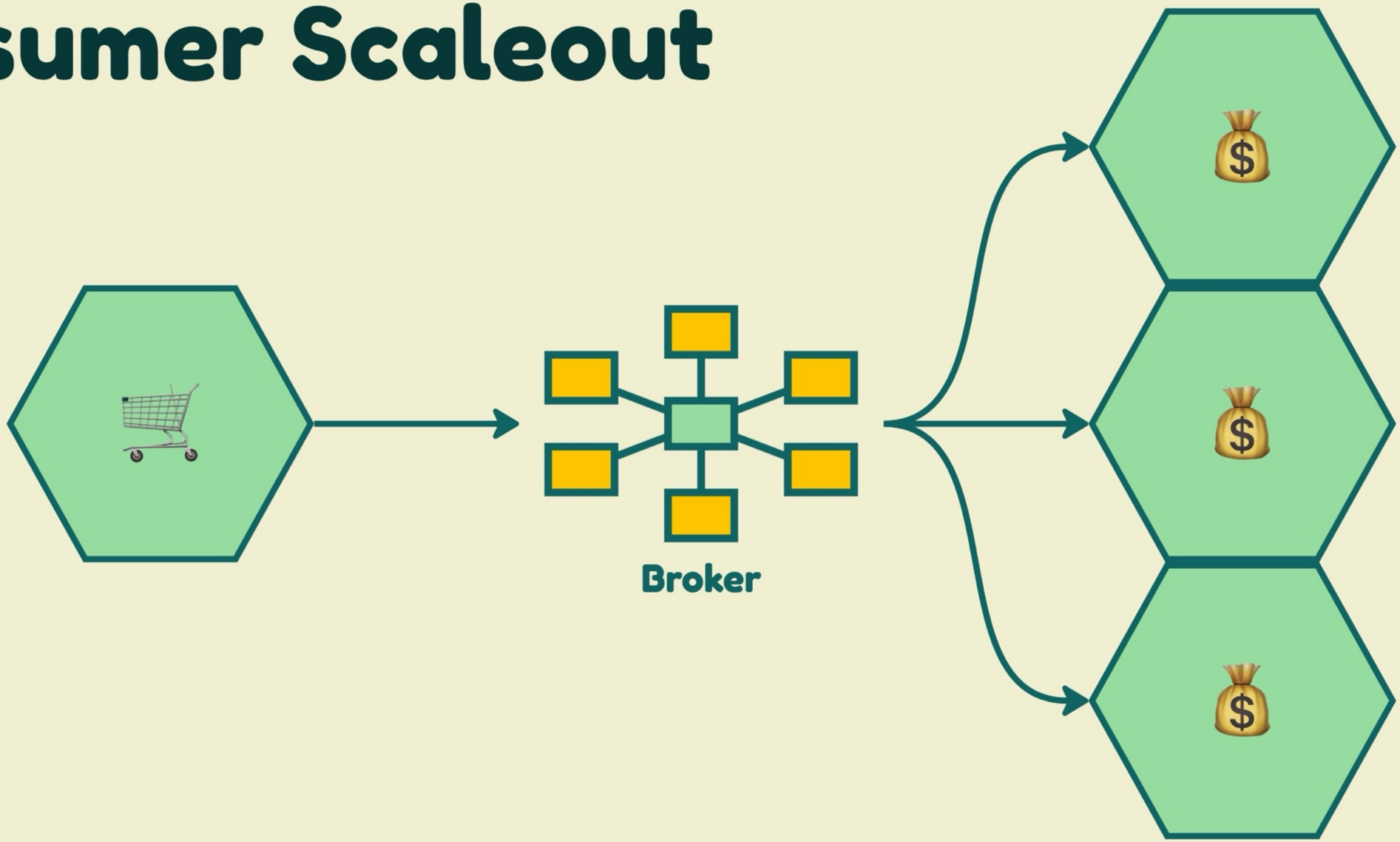


Message Processing Failure



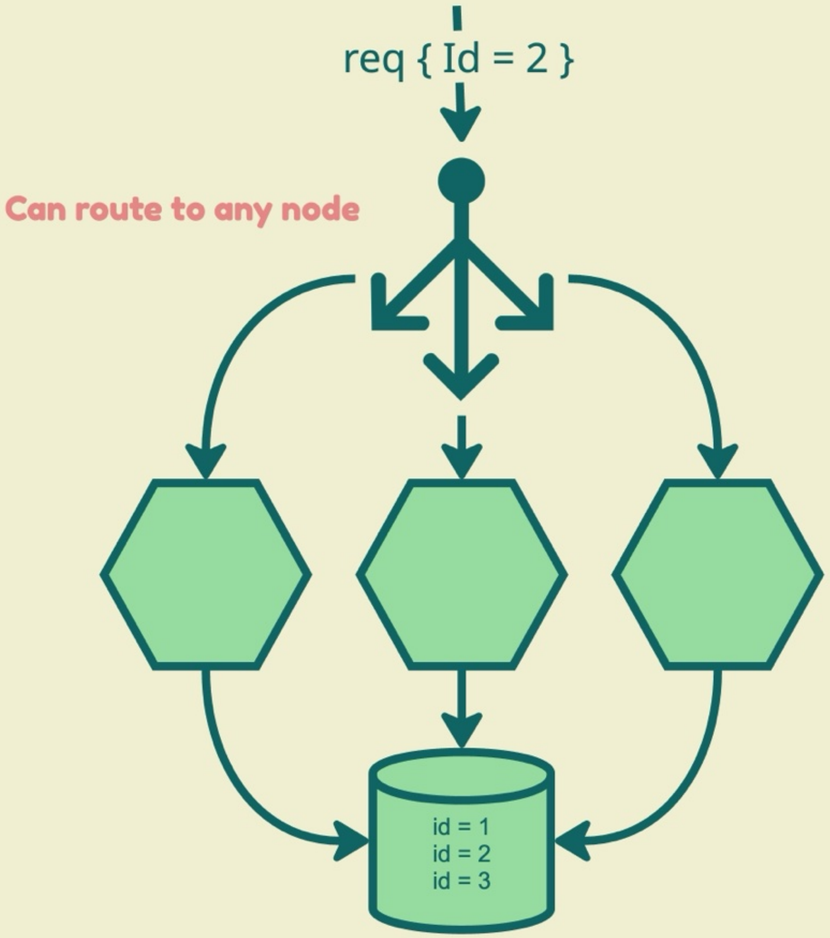
Message Routing

Consumer Scaleout



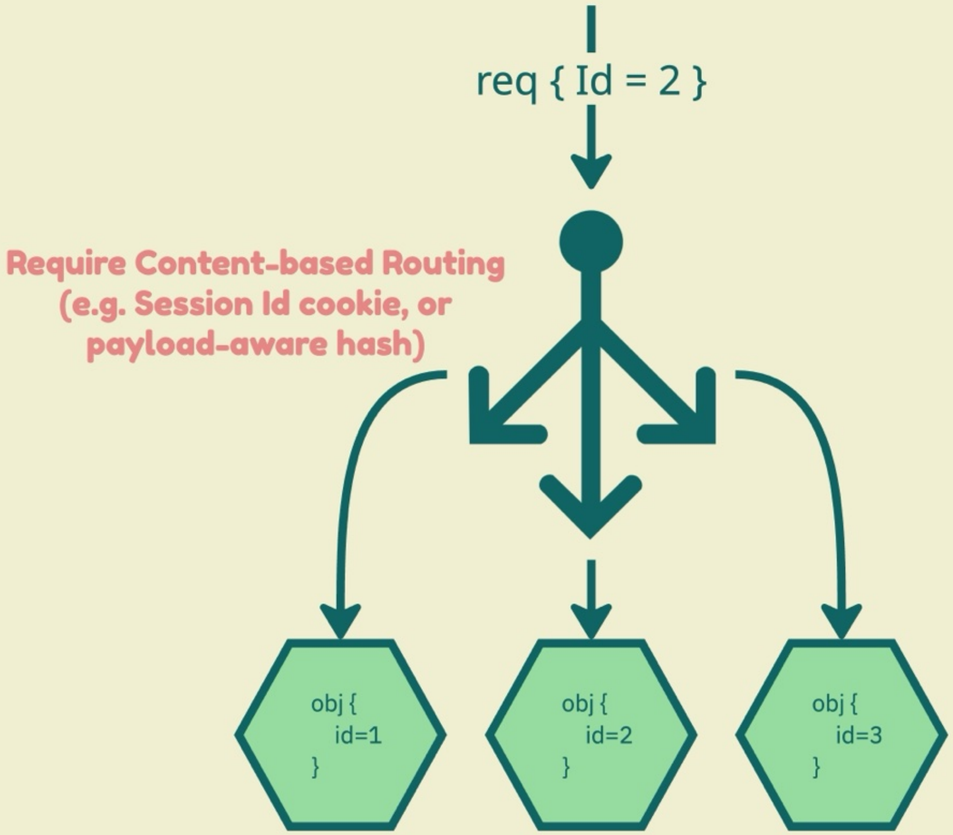
Because Queue consumer are stateless, and Stream consumers are stateful, scaling out consumers must work differently.

Clusters can be Stateless or Stateful



Stateless

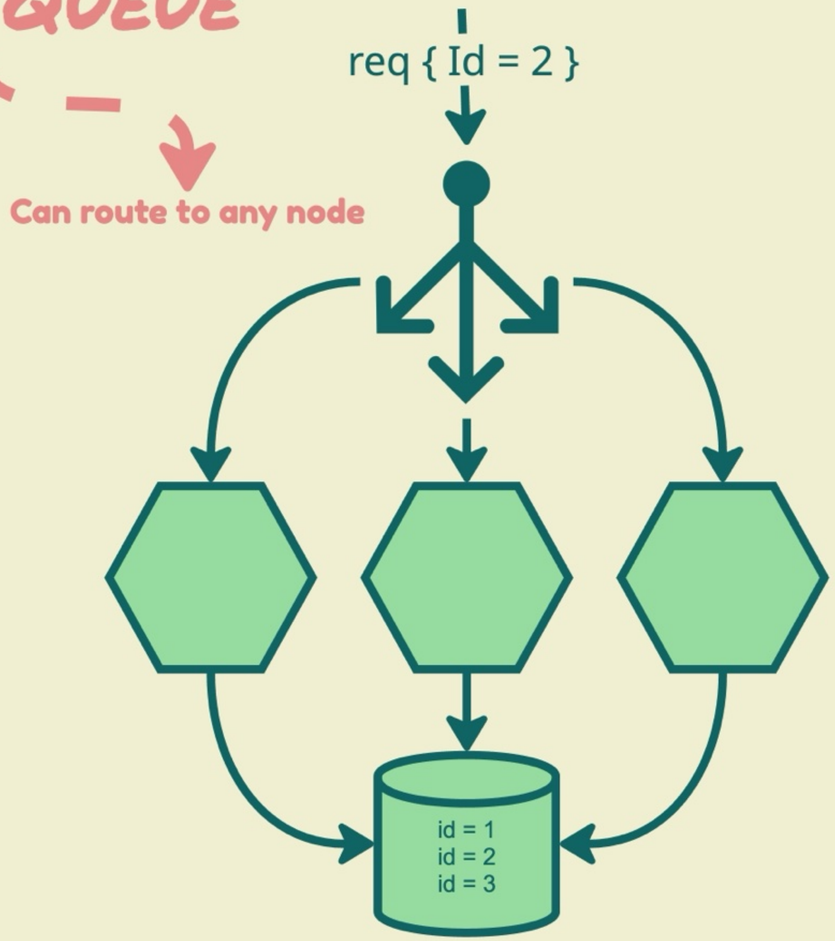
(The state still needs to go somewhere...)



Stateful

Clusters can be Stateless or Stateful

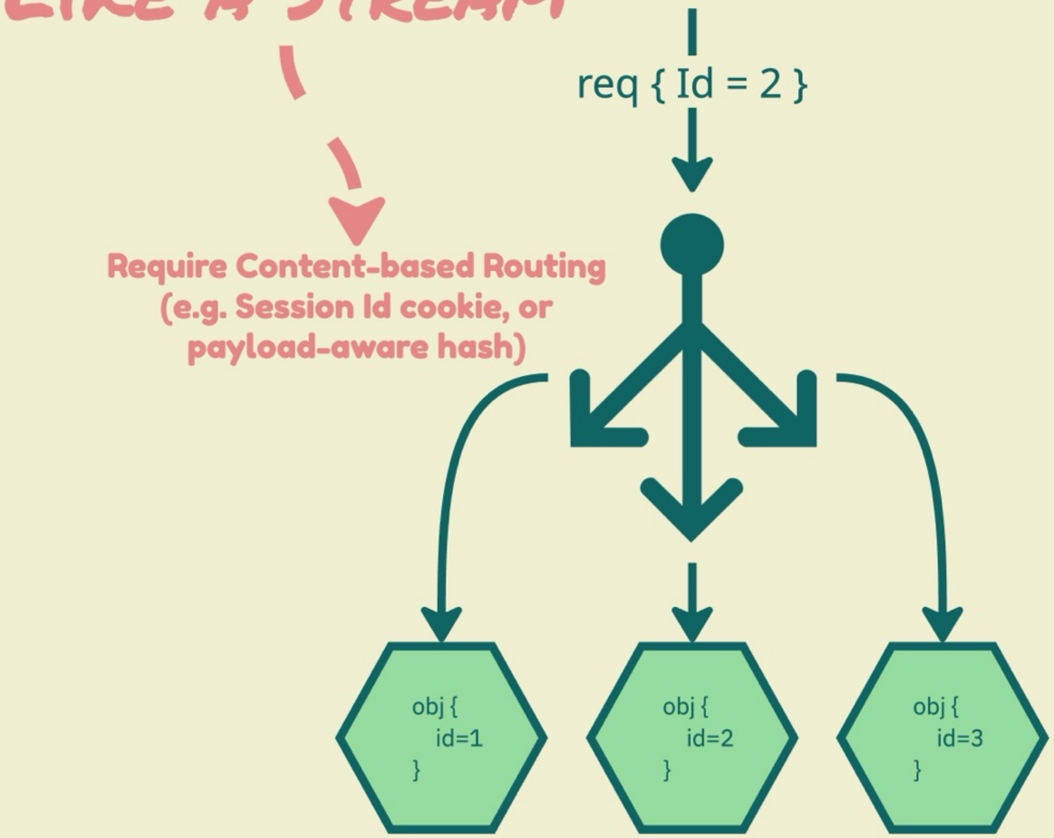
Like a Queue



Stateless

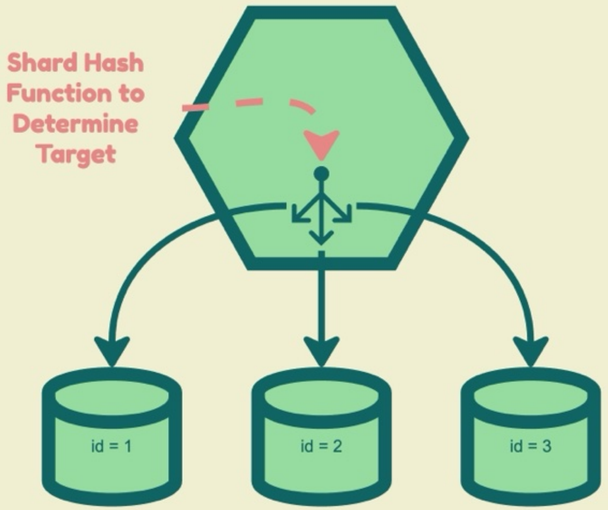
(The state still needs to go somewhere...)

Like a Stream

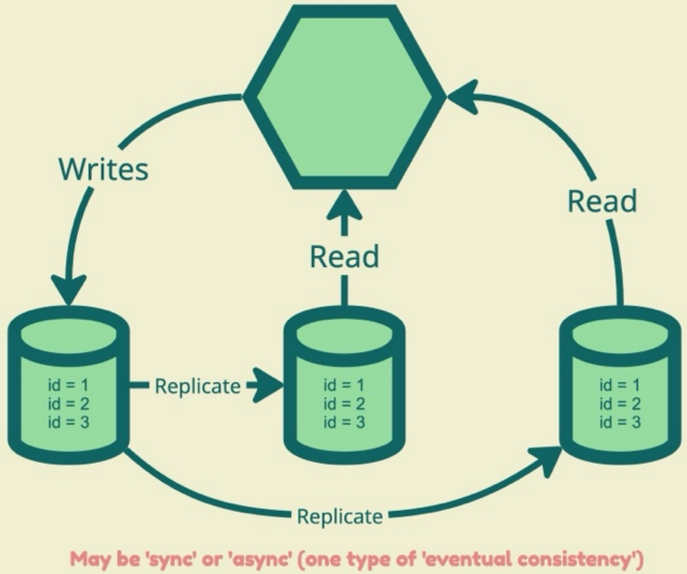


Stateful

Scaling out Stateful Clusters



Sharding

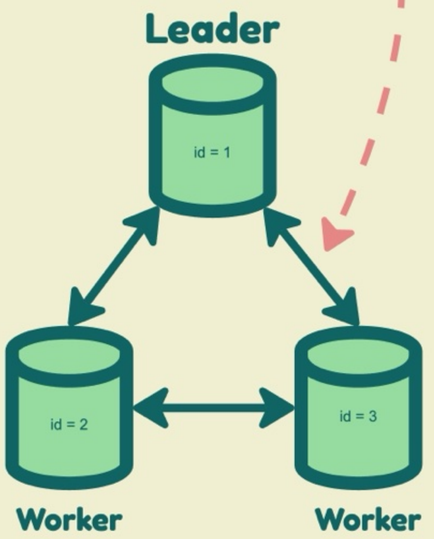


May be 'sync' or 'async' (one type of 'eventual consistency')

Traffic-type Splitting

Inter-process comms for:

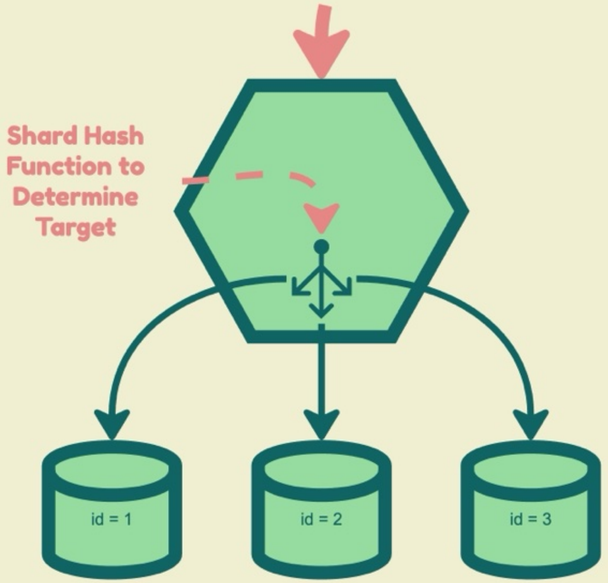
- quorum detection
- leader election
- task allocation (e.g. kafka consumer group partition allocation)



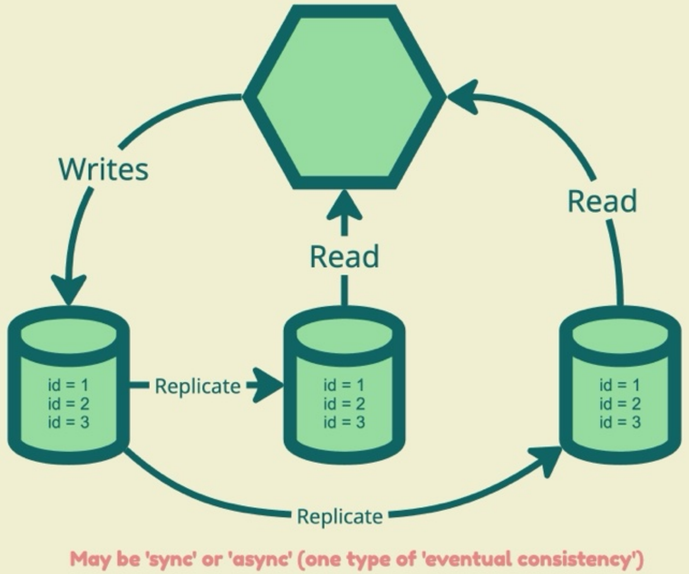
Workload Allocation

Scaling out Stateful Clusters

Streams use These Patterns



Sharding

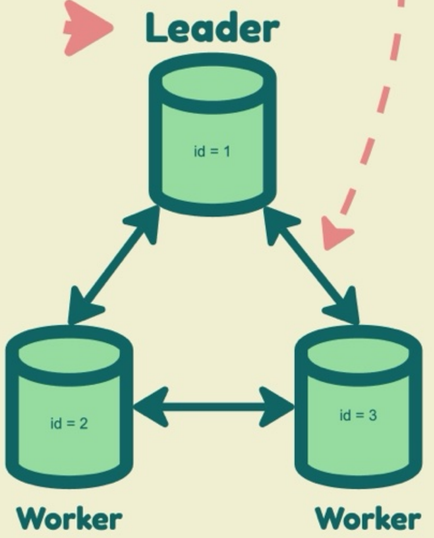


May be 'sync' or 'async' (one type of 'eventual consistency')

Traffic-type Splitting

Inter-process comms for:

- quorum detection
- leader election
- task allocation (e.g. kafka consumer group partition allocation)



Workload Allocation

Consumer Scaleout



Queues



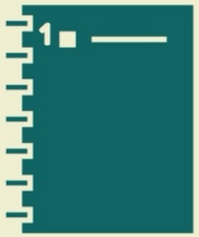


Queue



Streams





Distributed Log



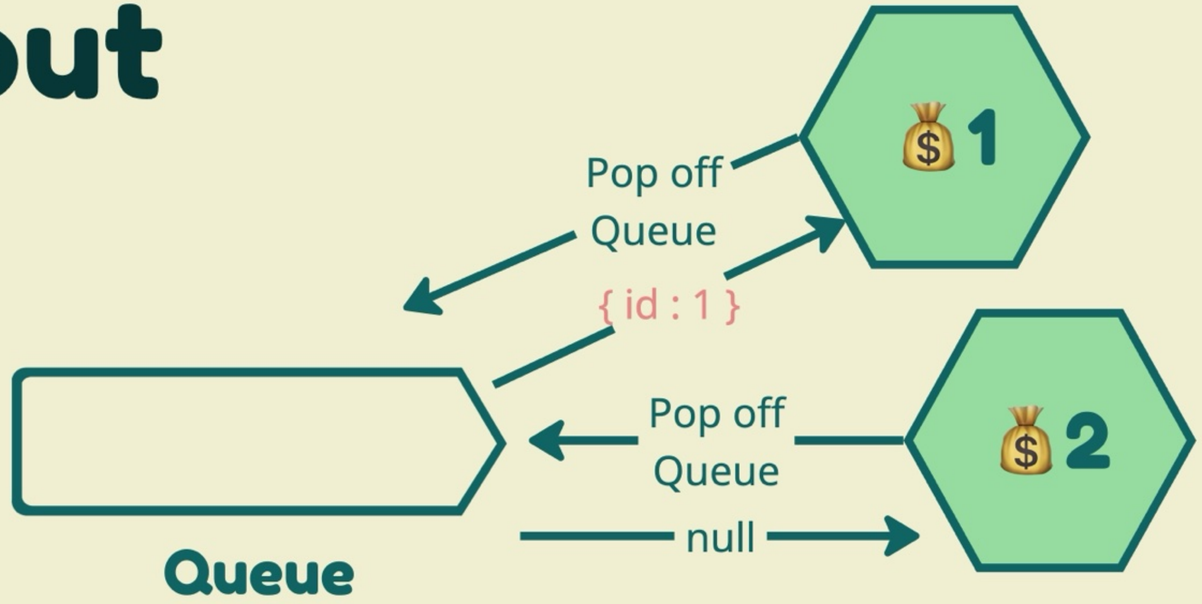




Consumer Scaleout

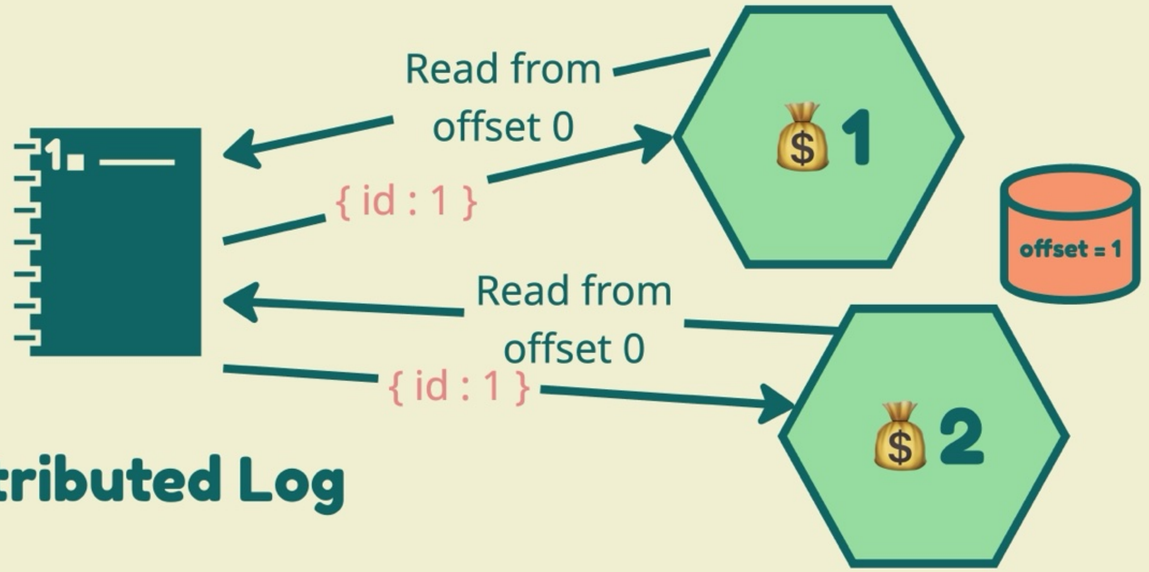
Queues





Streams



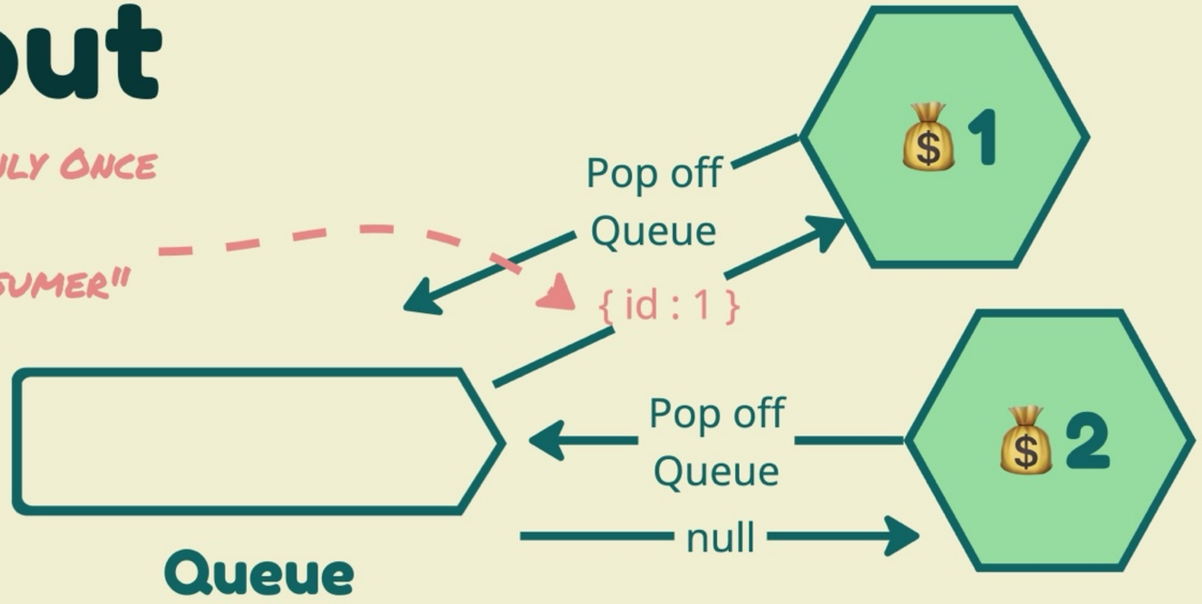


Consumer Scaleout

Message Processed Only Once
by Default:
The "Competing Consumer"
Pattern

Queues

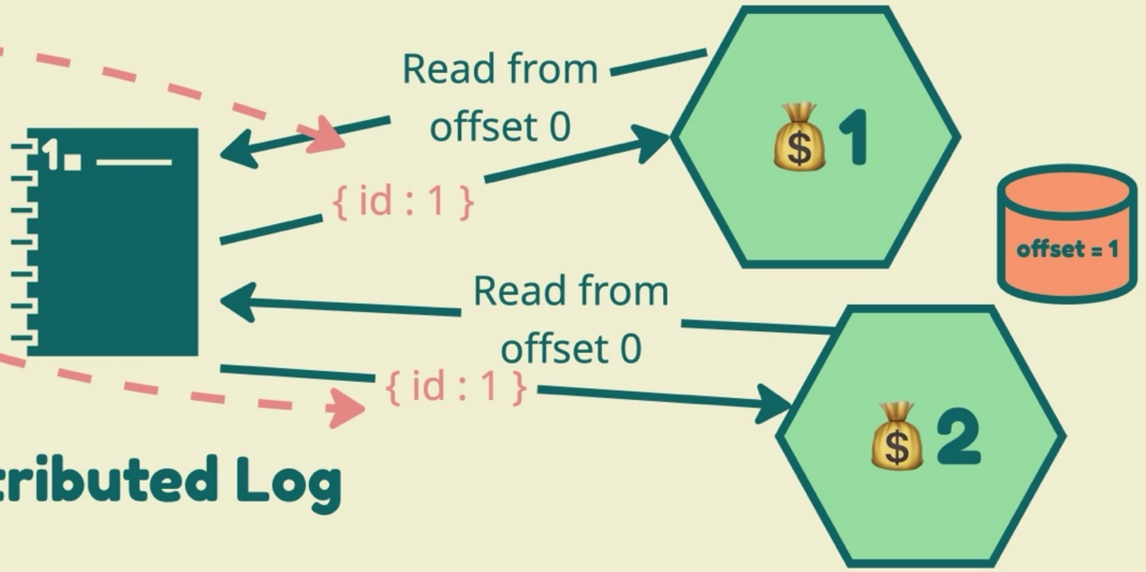




Streams



Message
Processed Twice!

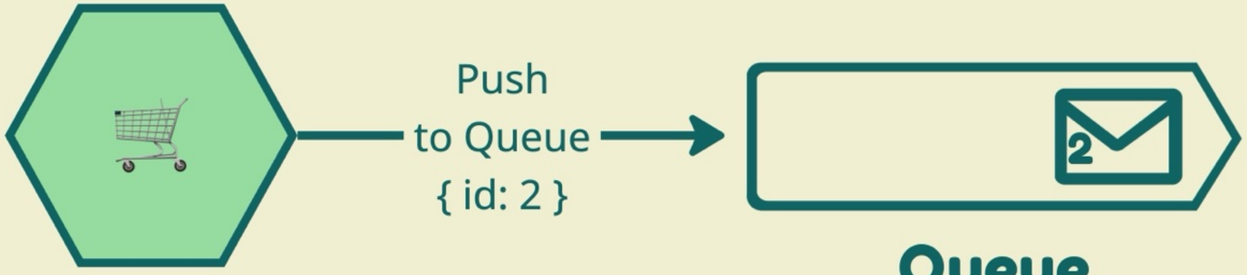


Kafka Low Level Consumer

Consumer Scaleout

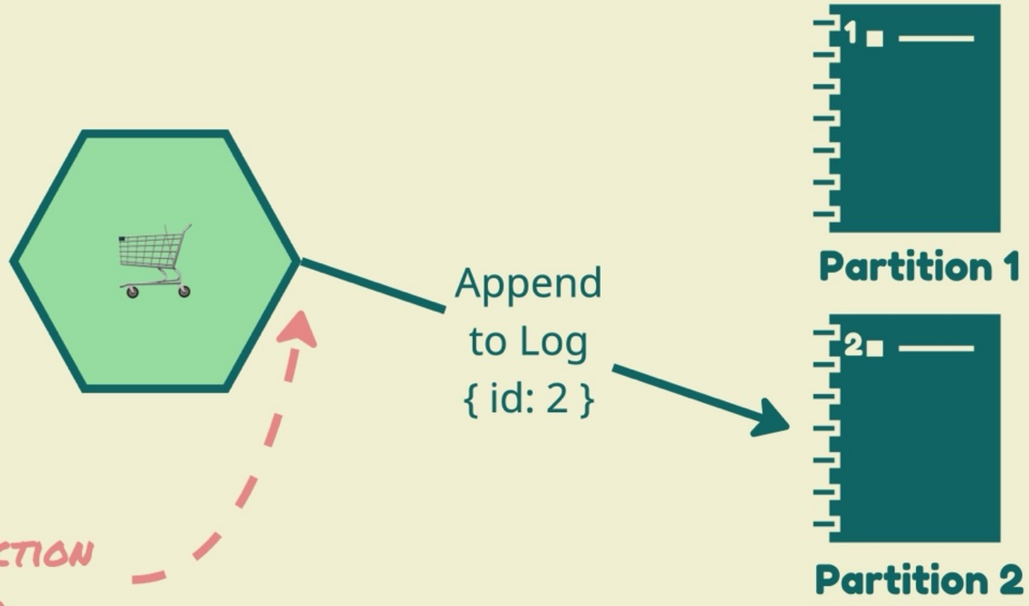


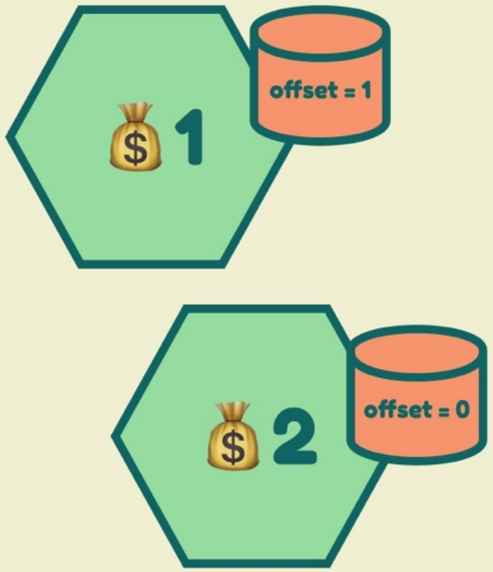
Queues





Streams

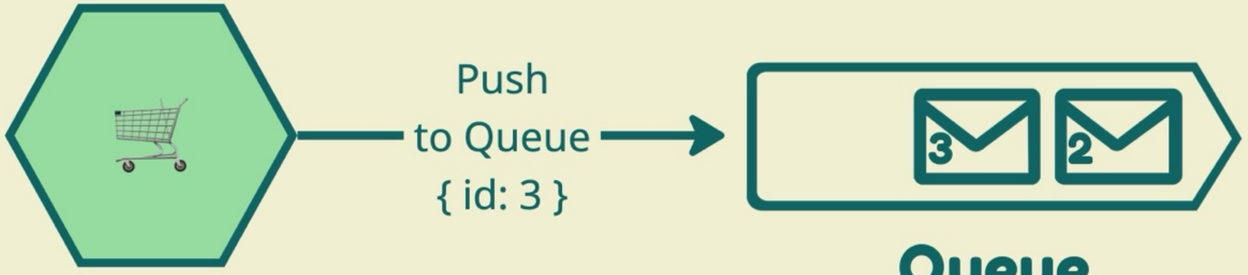




Consumer Scaleout

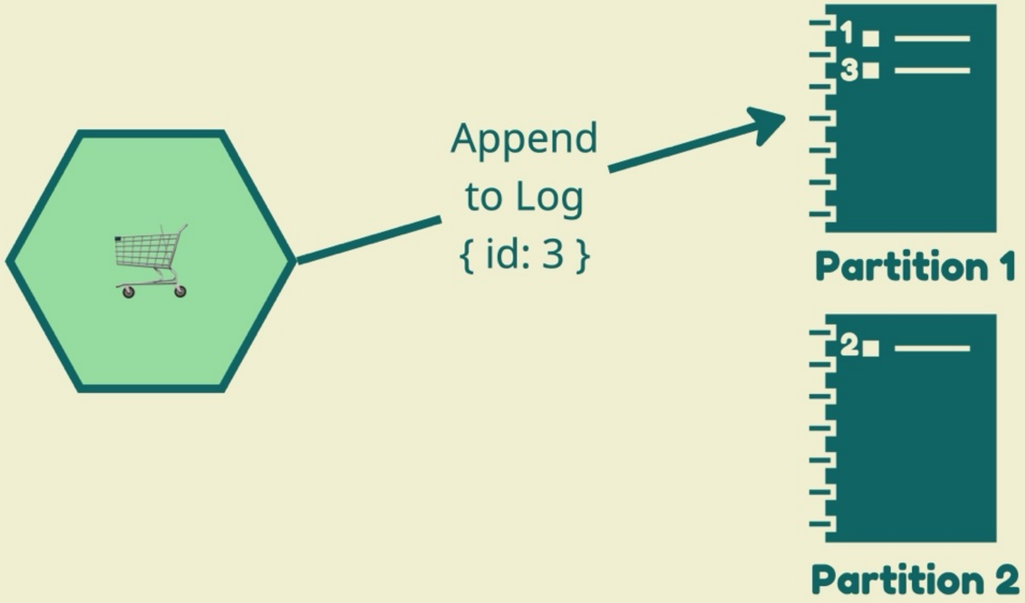


Queues





Streams





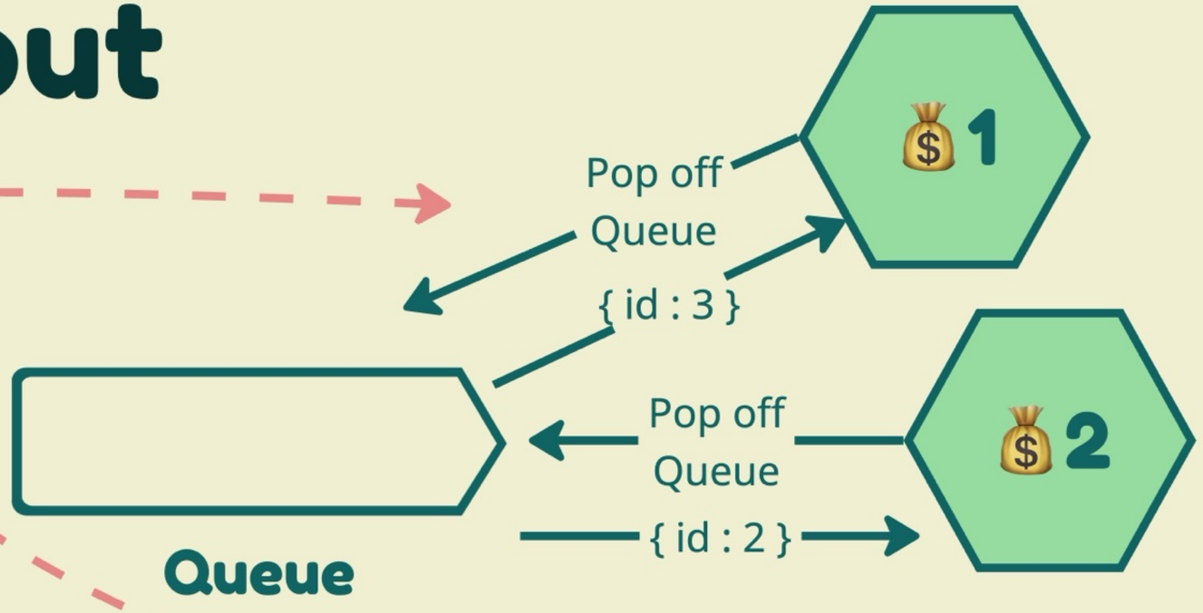


Consumer Scaleout

Each Message
Processed Once

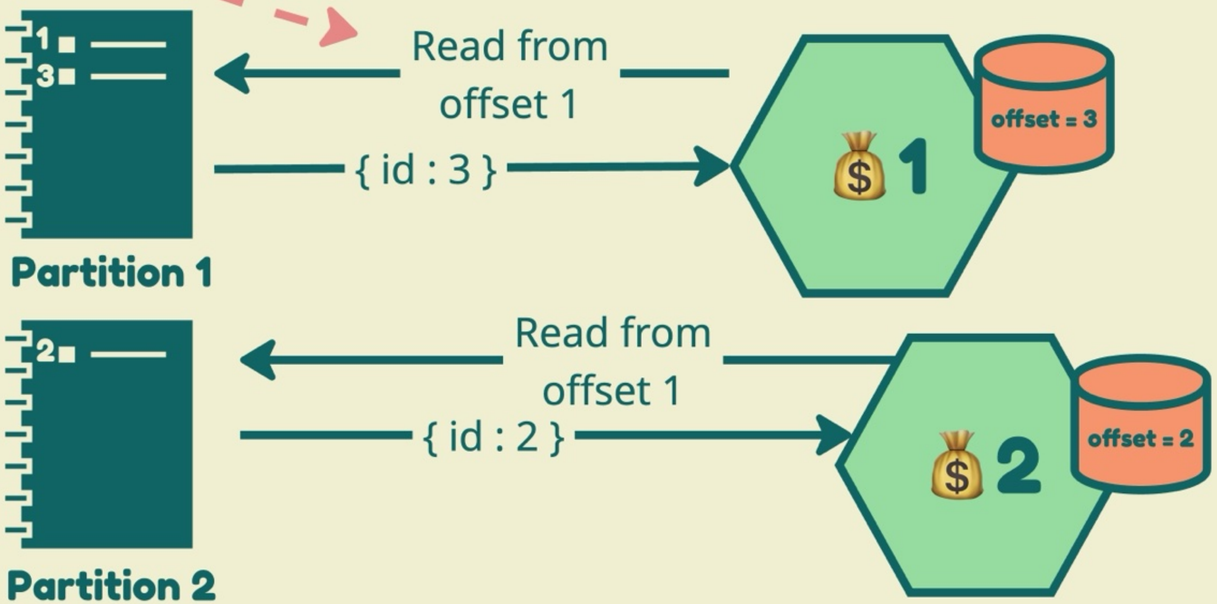
Queues





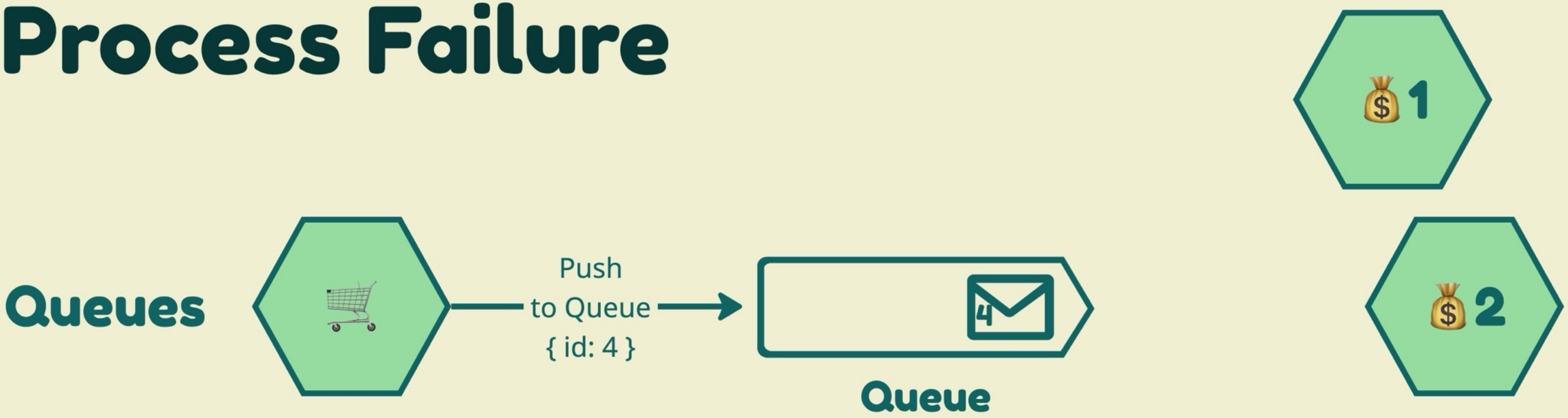
Streams



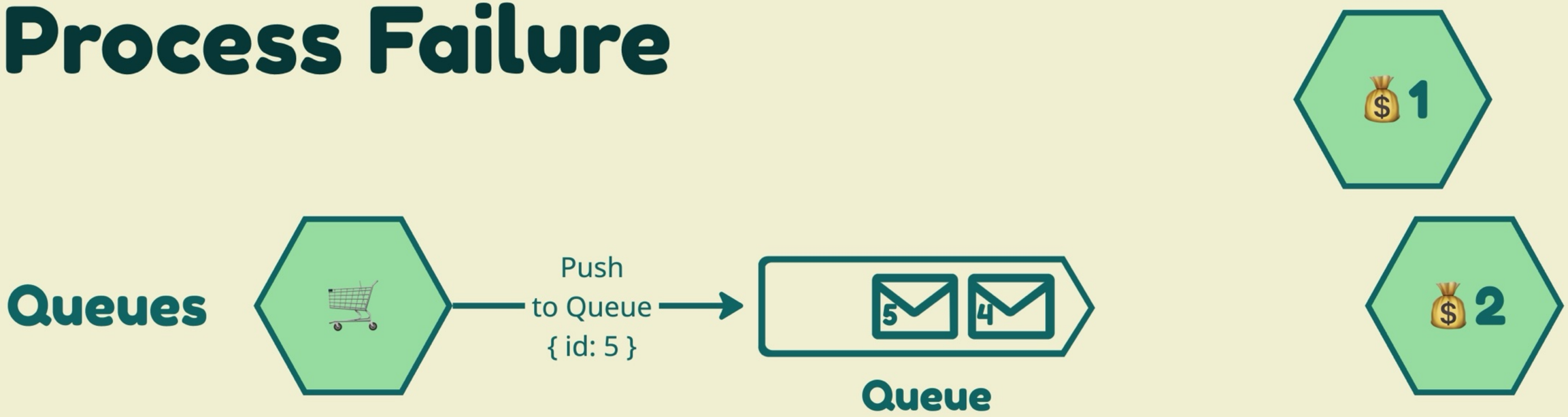


Distributed Log

Process Failure



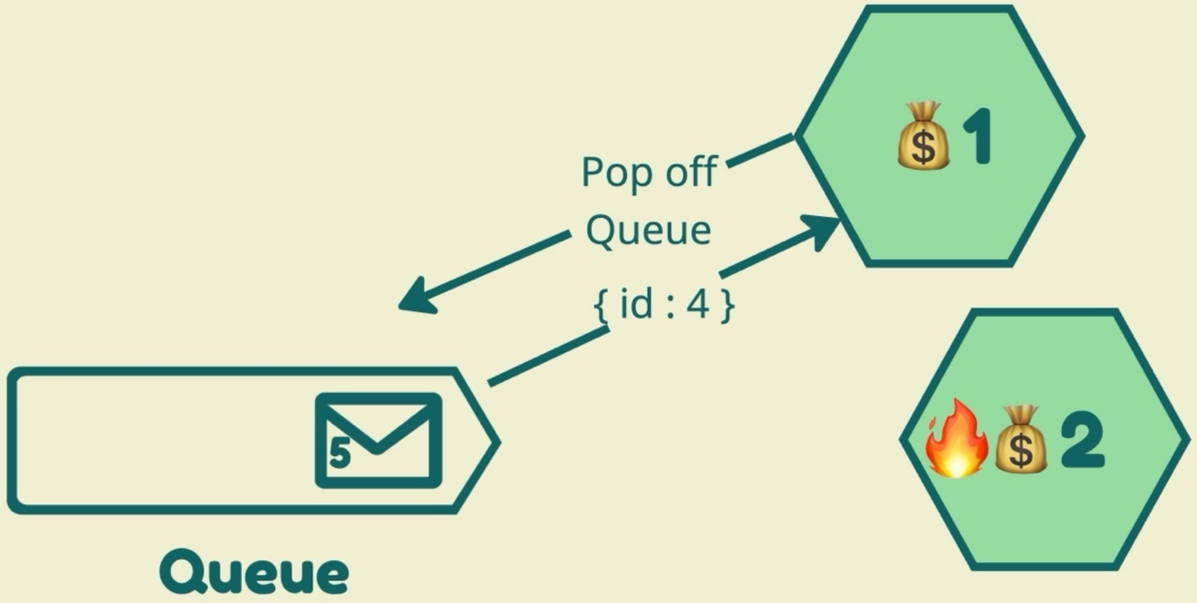
Process Failure



Process Failure

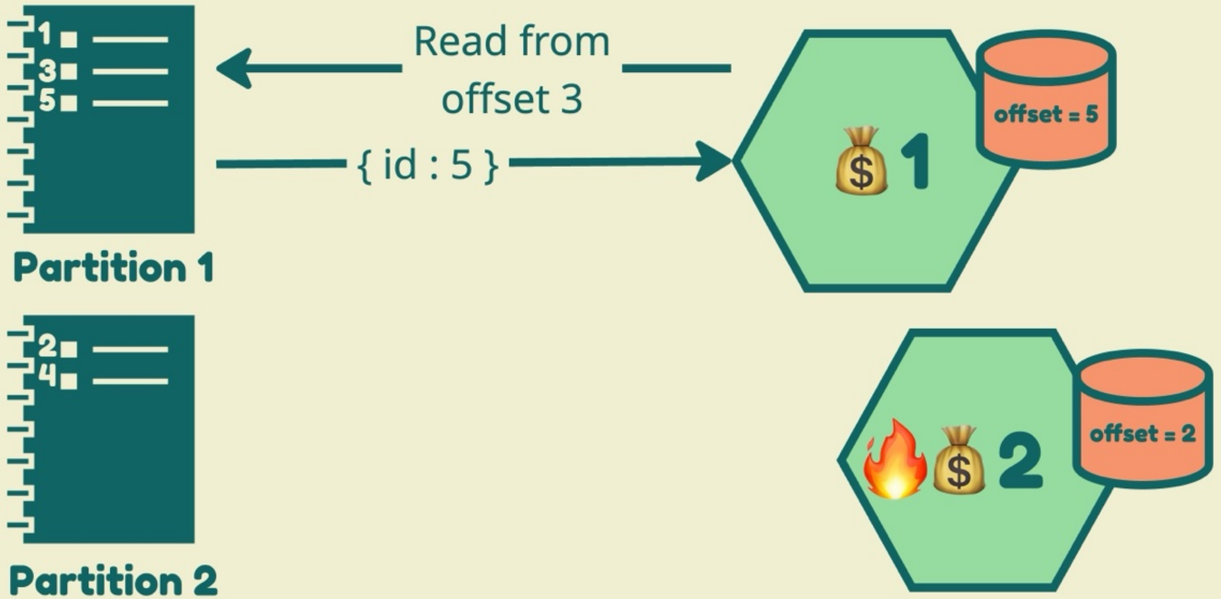
Queues





Streams

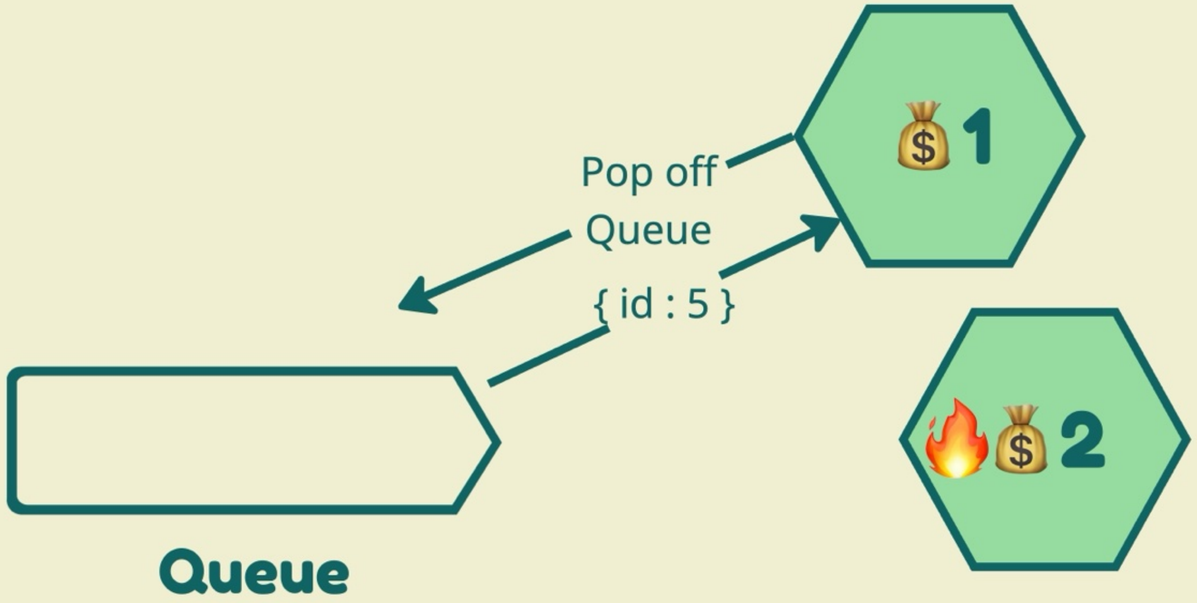




Process Failure

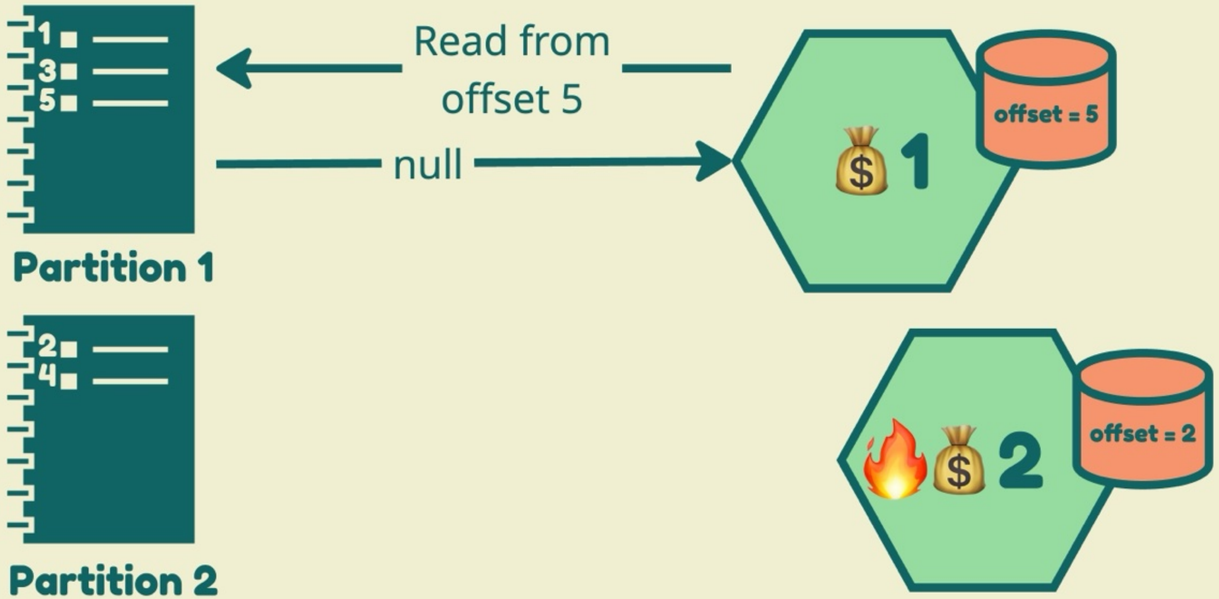
Queues





Streams

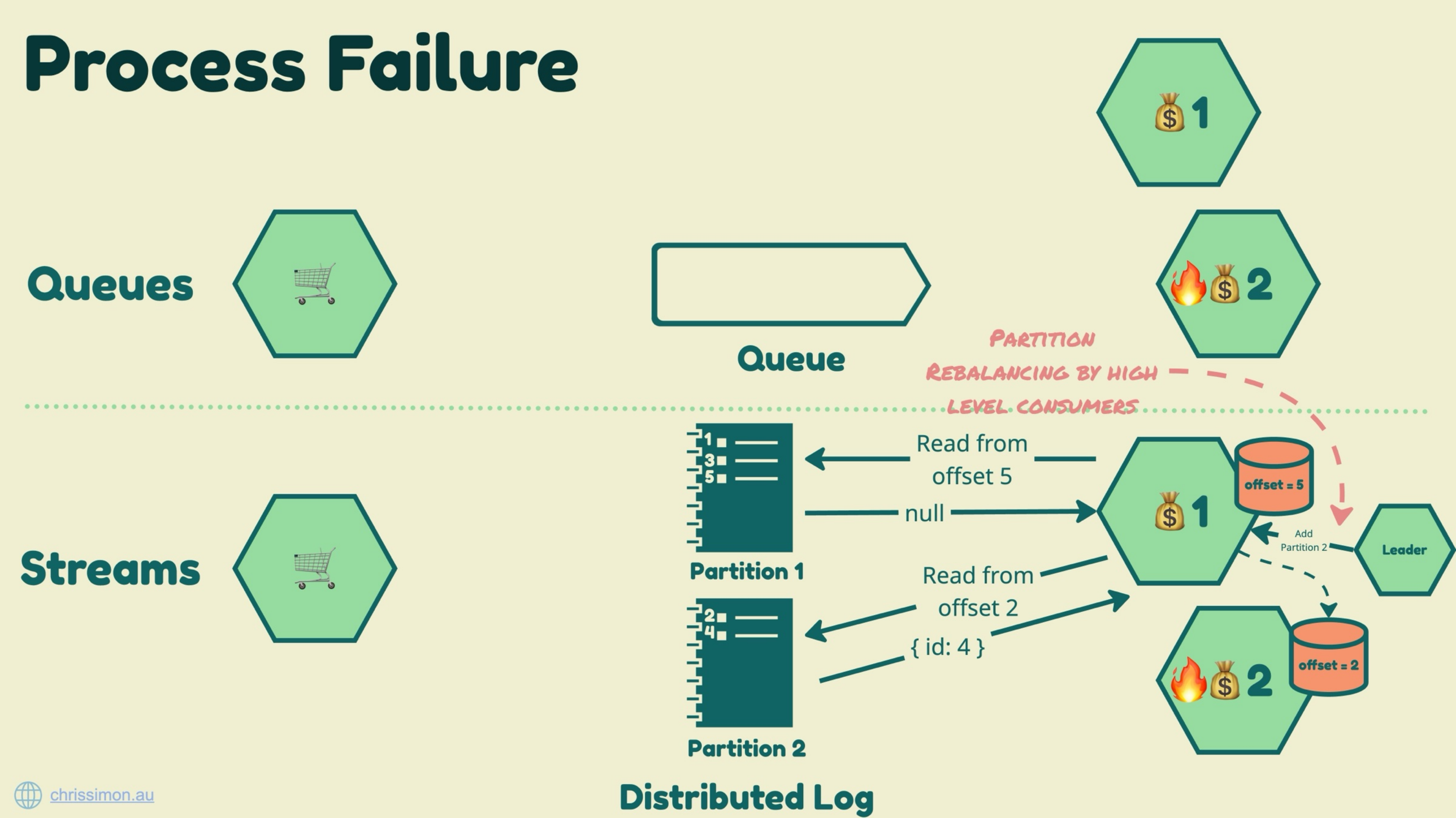




Message 4 is Never Processed
(With a low level consumer)!

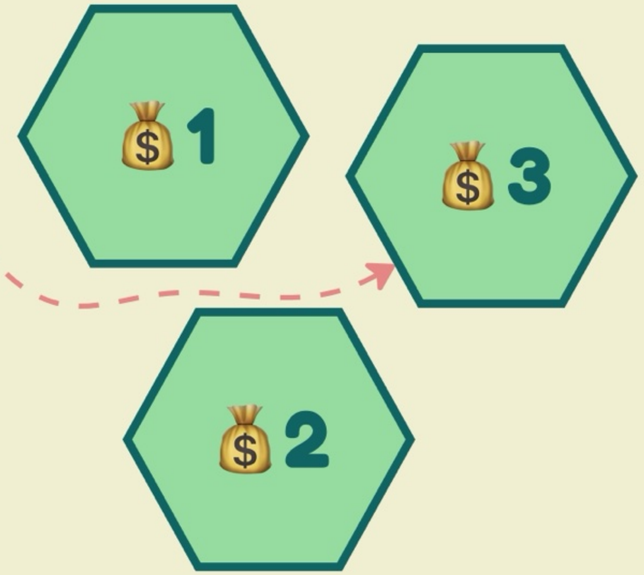
Distributed Log

Process Failure



Elastic Scale

Immediately adds to the processing capacity of the cluster



Queues

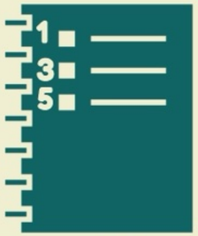




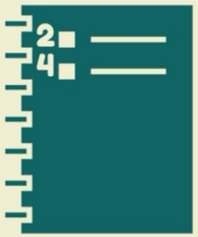
Queue

Streams

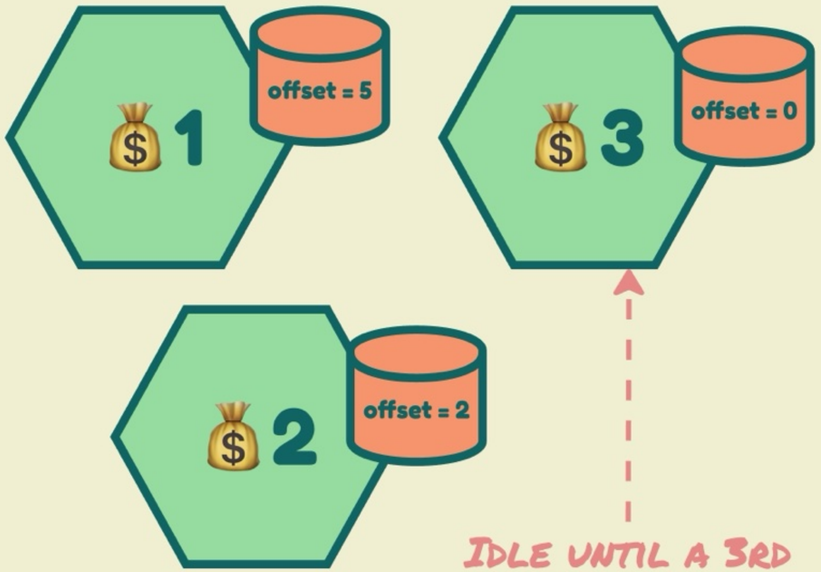




Partition 1



Partition 2



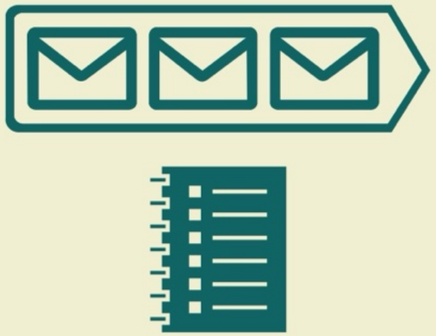
Idle until a 3rd Partition is Added

Distributed Log

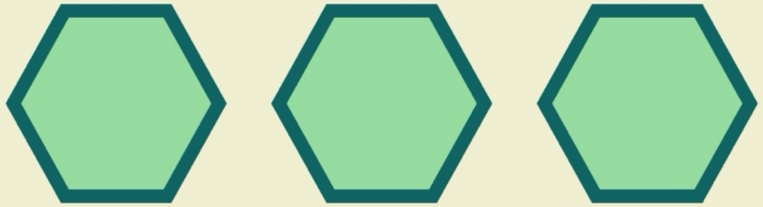

Alanis Morissette - Ironic (1996)
♪ And isn't it ironic... ♪


It's when you need to scale out,
But your partition count's too low...
chrissimon.au

What is different?



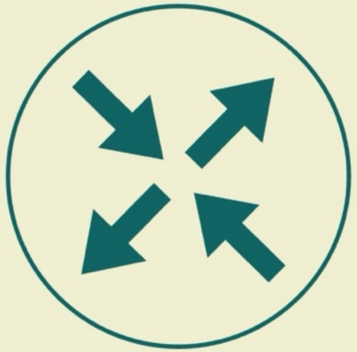
Message Distribution



Consumer Scale-out



Message Processing Failure



Message Routing

Message Processing Failure

Queues



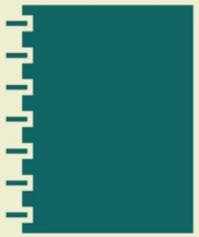


Queue

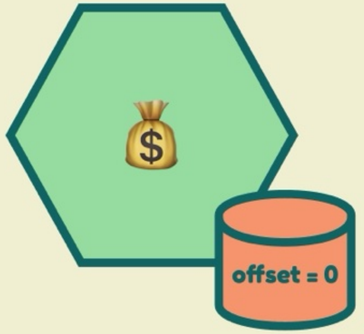


Streams



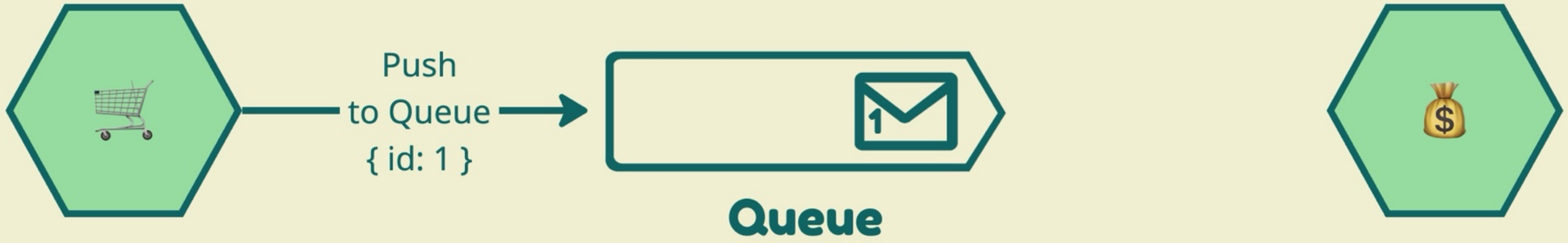


Distributed Log

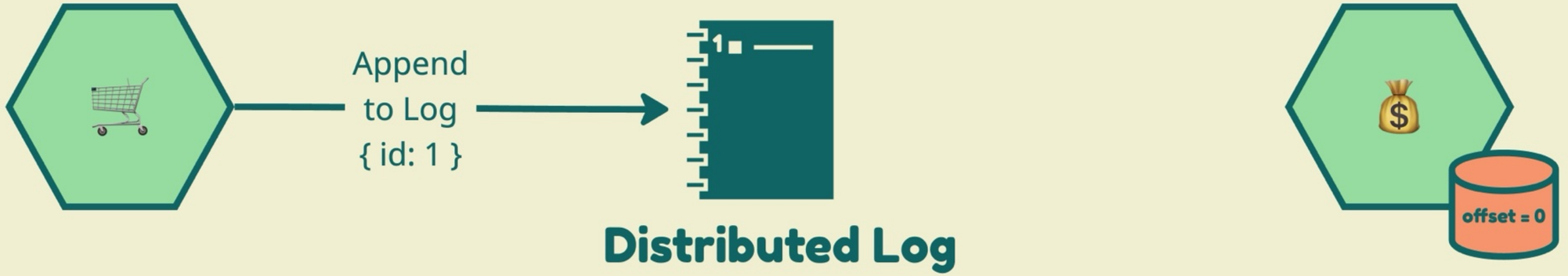


Message Processing Failure

Queues

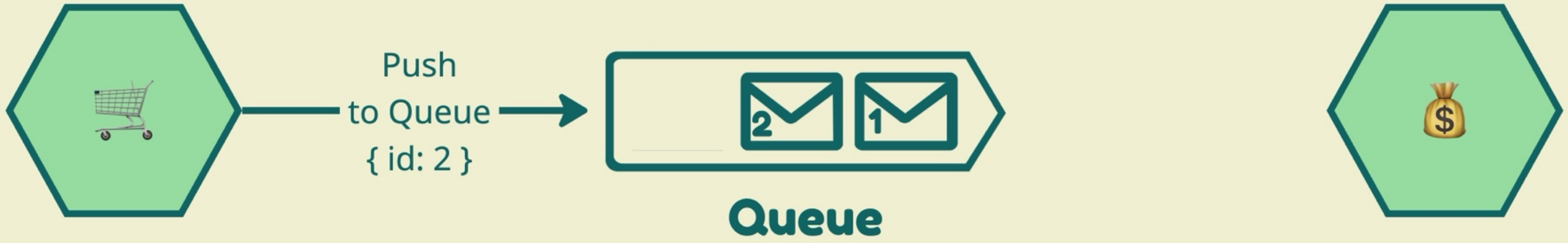


Streams

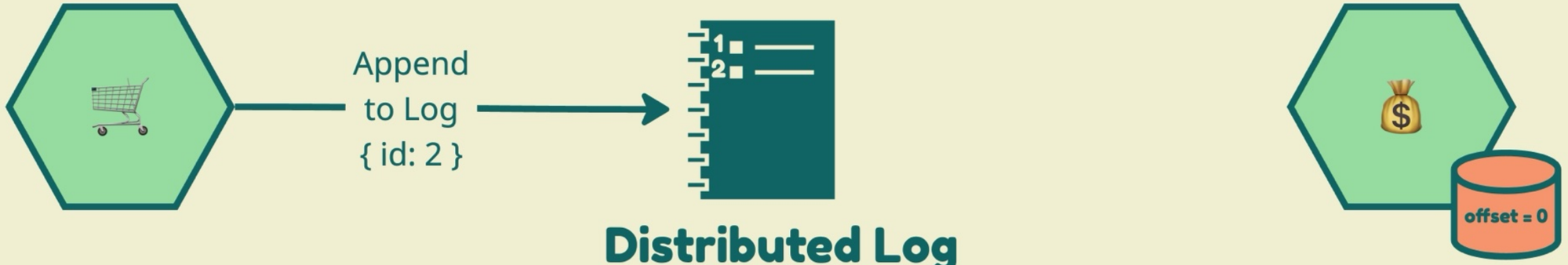


Message Processing Failure

Queues



Streams

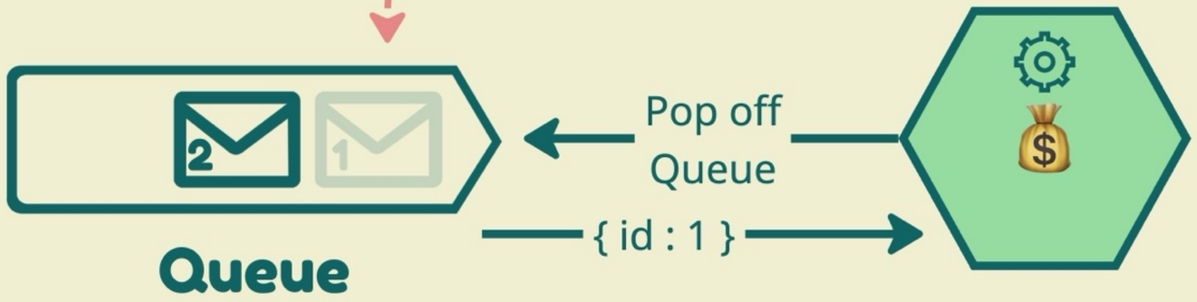


Message Processing Failure

While processing, the message is invisible to other consumer processes

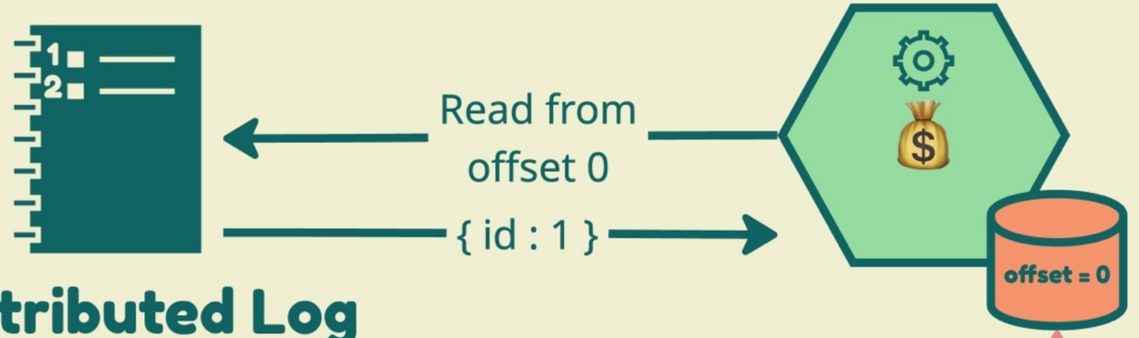
Queues





Streams





While processing, the offset is not yet committed

Message Processing Failure

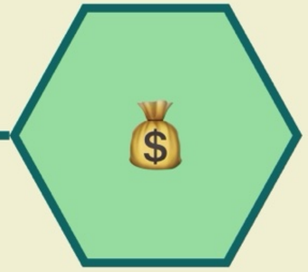
Explicitly confirm msg successfully processed

Queues





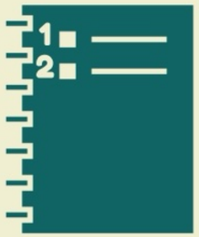
Delete Msg 1



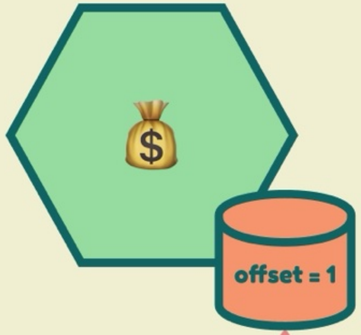
Queue

Streams





Distributed Log

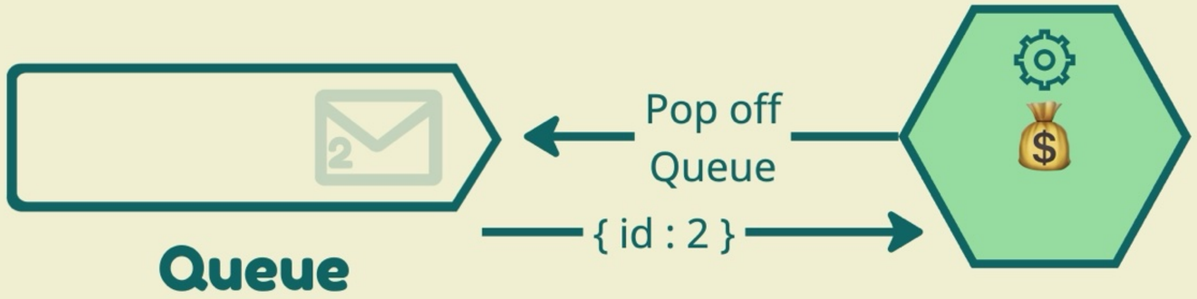


Once processed, the updated offset can be committed (usually async with processing)

Message Processing Failure

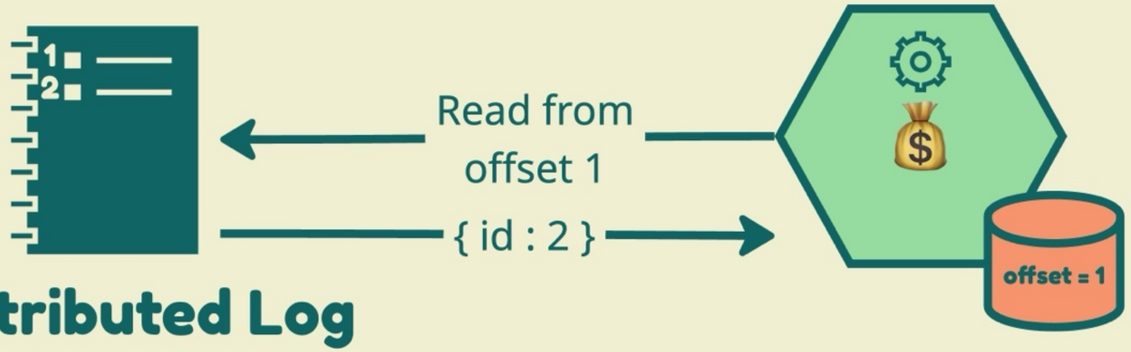
Queues





Streams





Message Processing Failure

Failure before message deleted

Queues



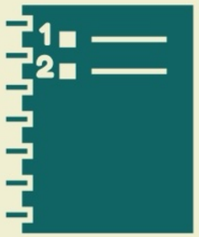


Queue

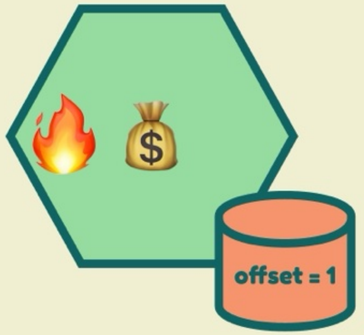


Streams





Distributed Log



Failure before Offset Committed

Message Processing Failure



After some time, the msg becomes visible again (Specifics vary per system)

Queues



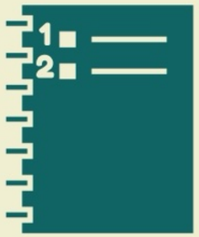


Queue

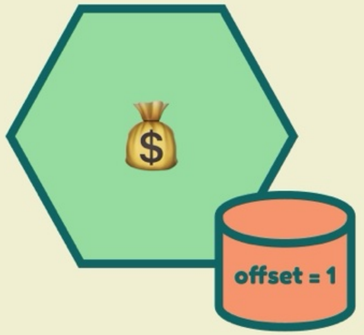


Streams





Distributed Log

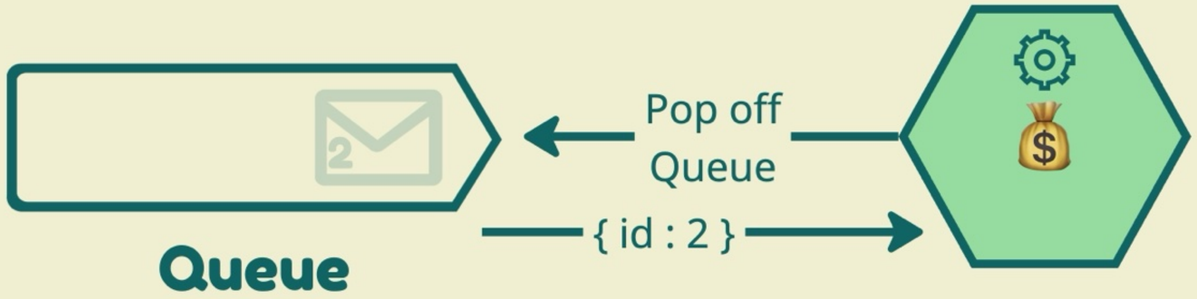


Message Processing Failure

We get a second chance to process Message 2

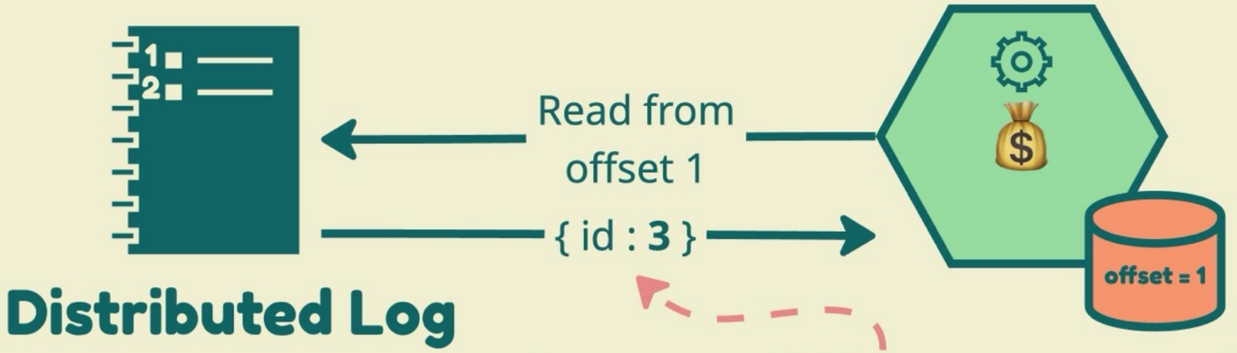
Queues





Streams





Default is to SKIP the faulty message!

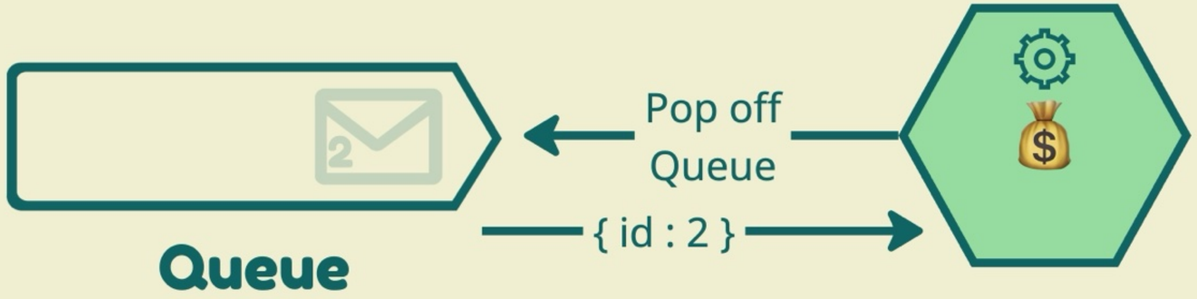
(The offset of the failed message is still kept in memory, even with manual commits)

Message Processing Failure

We get a second chance to process Message 2

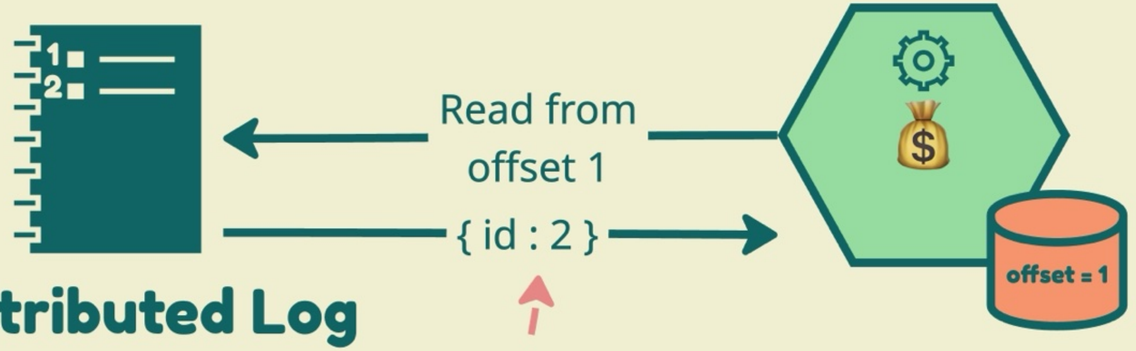
Queues





Streams



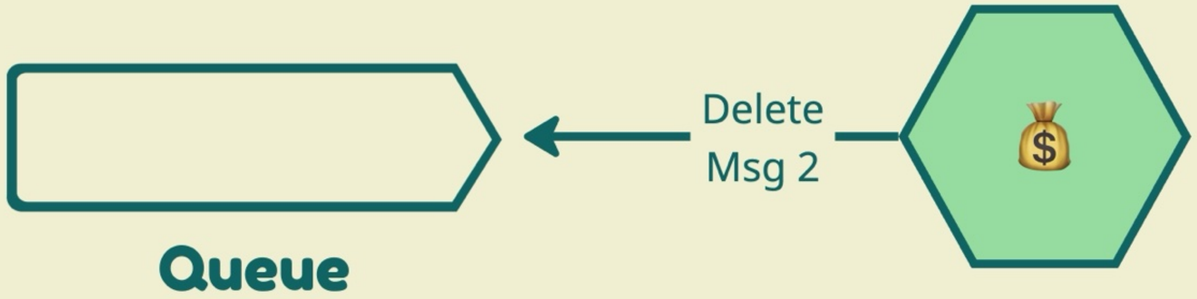


To retry, your application code must explicitly SEEK back to the failed message

Message Processing Failure

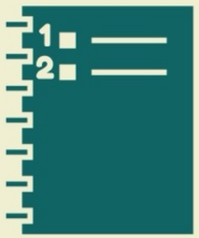
Queues

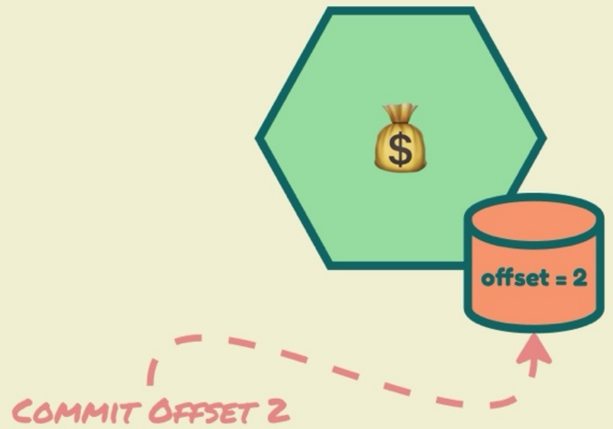




Streams







Repeated Failure

After repeated failures, the Broker treats the msg as 'poison' and moves it to the DLQ



Dead Letter Queue

Queues



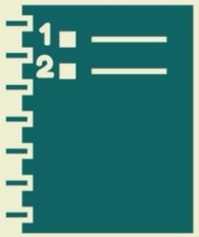


Queue

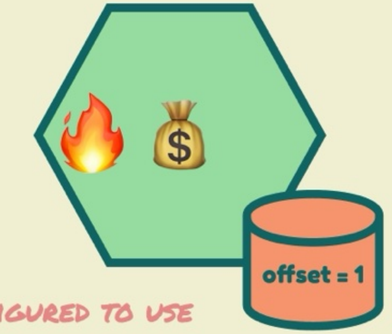


Streams





Distributed Log



If consumer is configured to use Seek and retry, then you must also manually track the retry count (somewhere) and decide what to do if too many retries

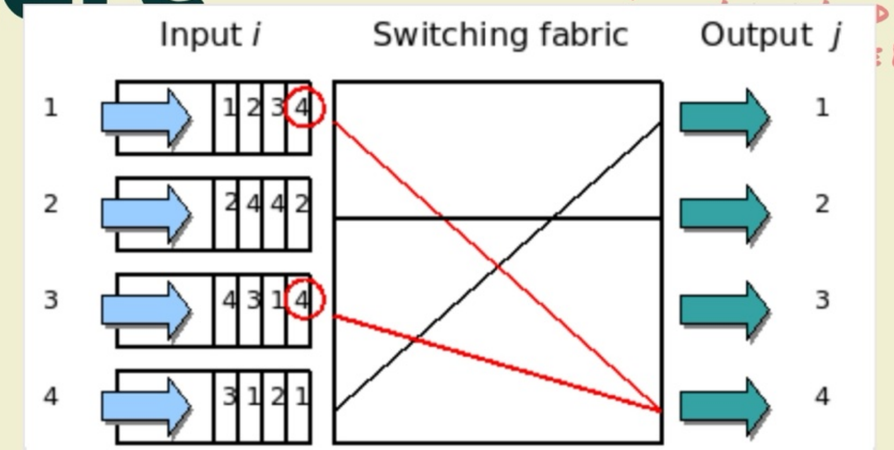
Repeated Failure

After repeated failures, the Broker treats the msg as [illegible] moves it [illegible] DLQ

Queues







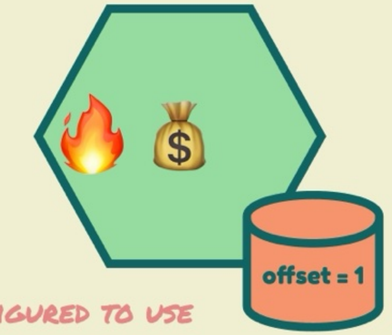
W en.wikipedia.org

Head-of-line blocking - Wikipedia

Head-of-line blocking (HOL blocking) in computer networking is a performance-limiting phenomenon that occurs when a queue of packets is held up by the first packet in the queue. This occurs, for example, in input-buffered network switches, out-of-order...

Streams





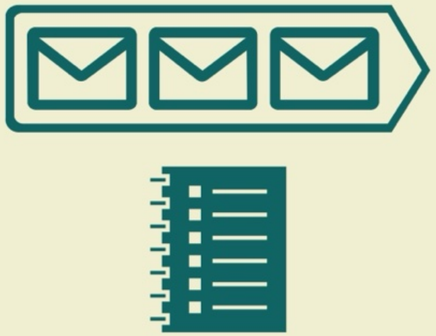
[illegible] sumer is configured to use [illegible] nd retry, then you must also [illegible] ally track the retry count (somewhere) and decide what to do if too many retries


Alanis Morissette - Ironic (1996)
♪ And isn't it ironic... ♪
chrissimon.au

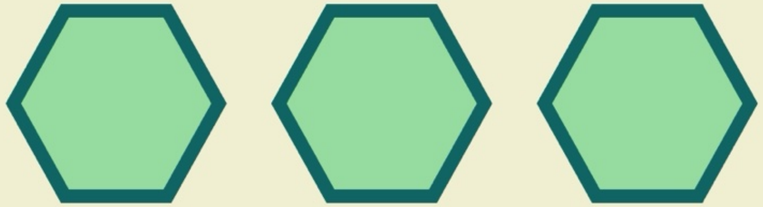


chrissimon.au

What is different?



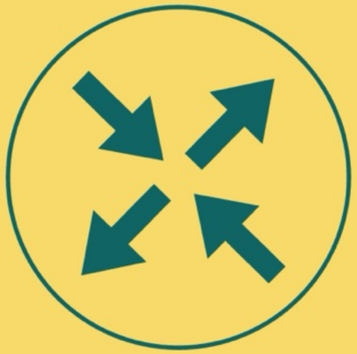
Message Distribution



Consumer Scale-out



Message Processing Failure



Message Routing

Message Routing

Queues





Topic

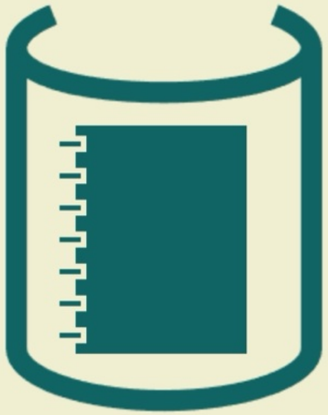


Queue

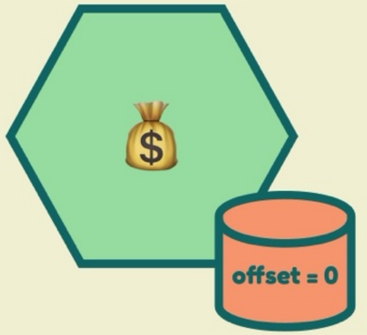


Streams





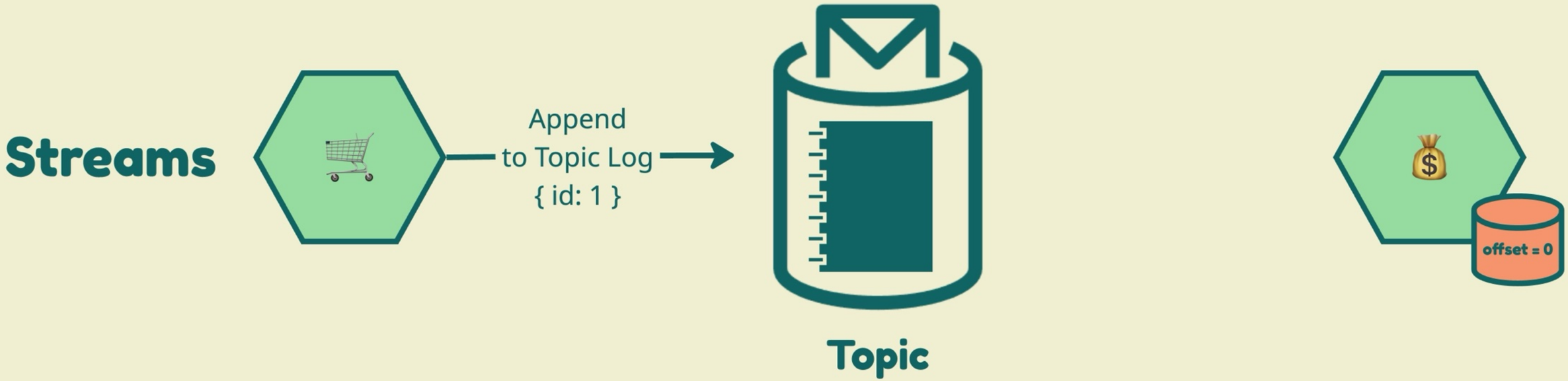
Topic



Topologies are different

Message Routing





Message Routing

Queues





Topic

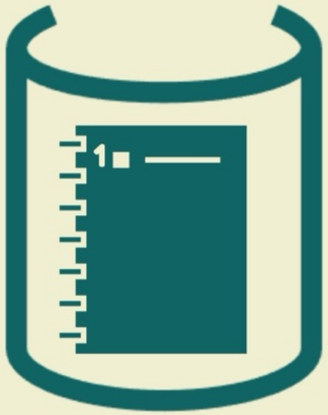


Queue

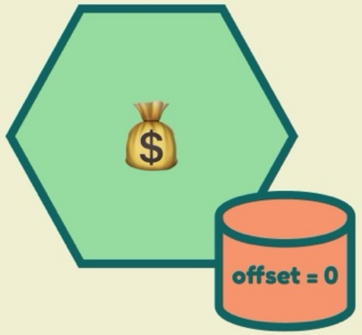


Streams





Topic



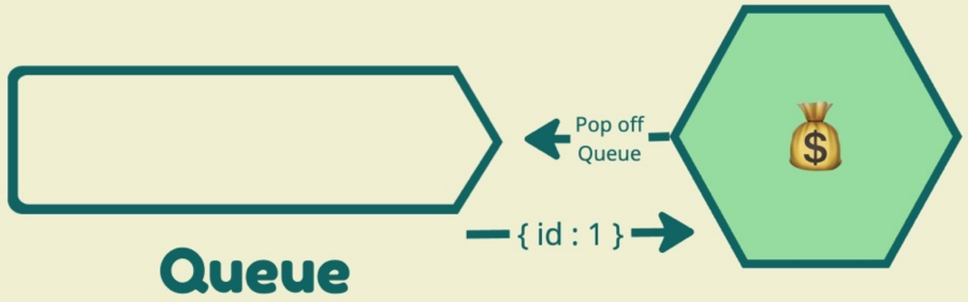
Message Routing

Queues





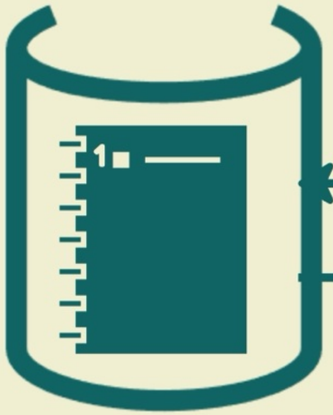
Topic



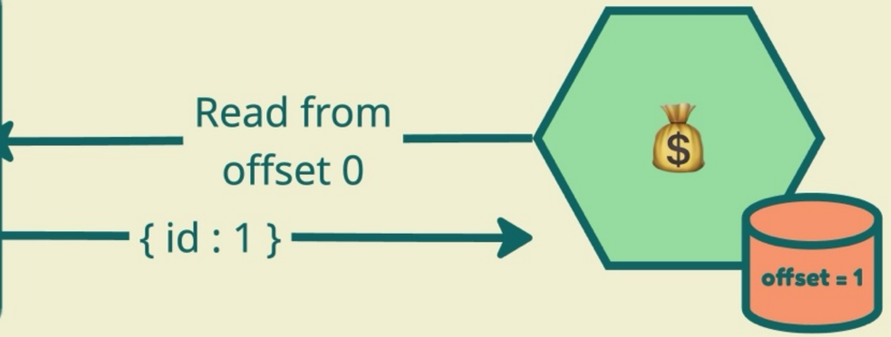
Queue

Streams





Topic



Message Routing

"Fan Out" Pub/Sub

Queues





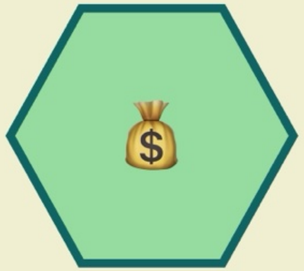
Topic



💰 -Queue



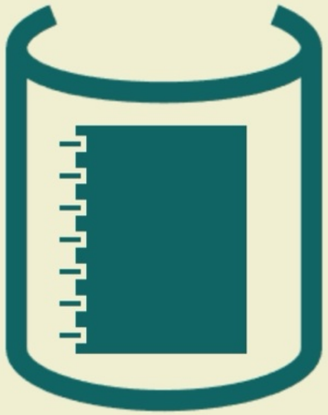
📧 -Queue



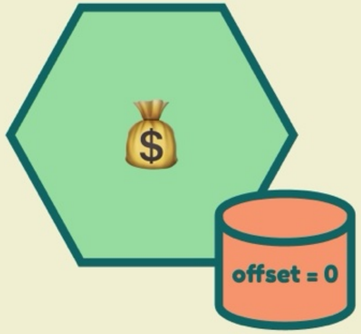


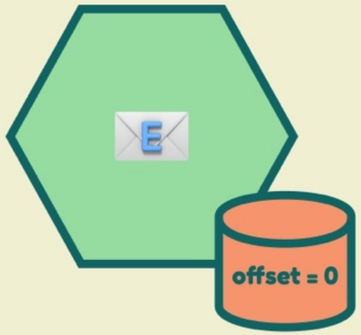
Streams





Topic





Message Routing

Often called "Logical Subscribers"

Queues





Topic

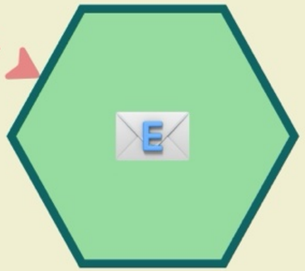


💰 -Queue



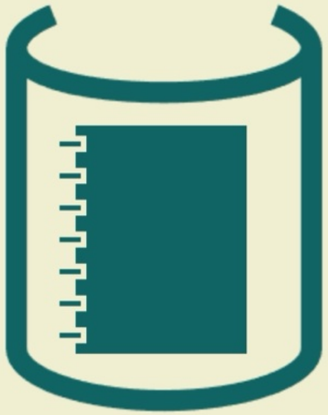
📧 -Queue





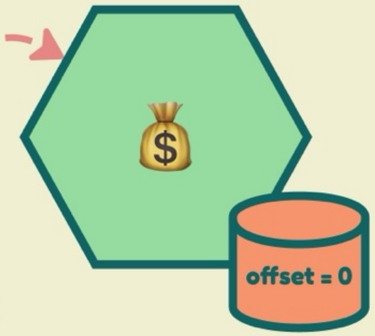
Streams

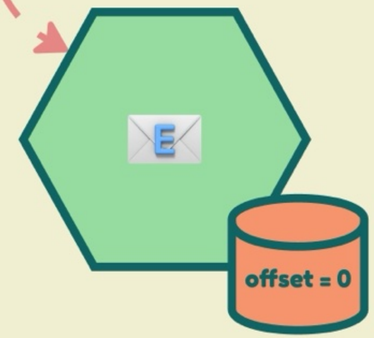




Topic

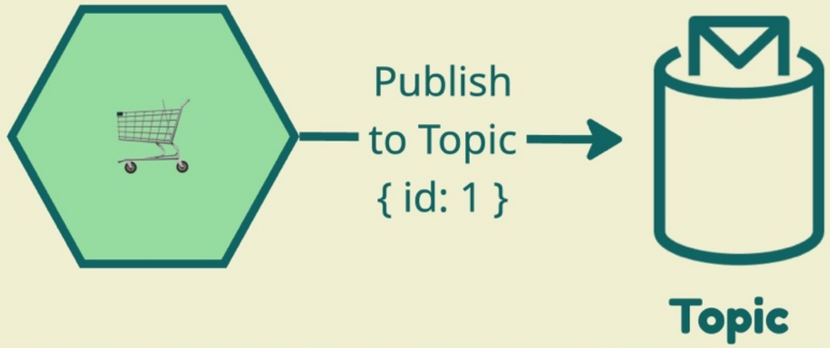
Kafka calls these "Consumer Groups"



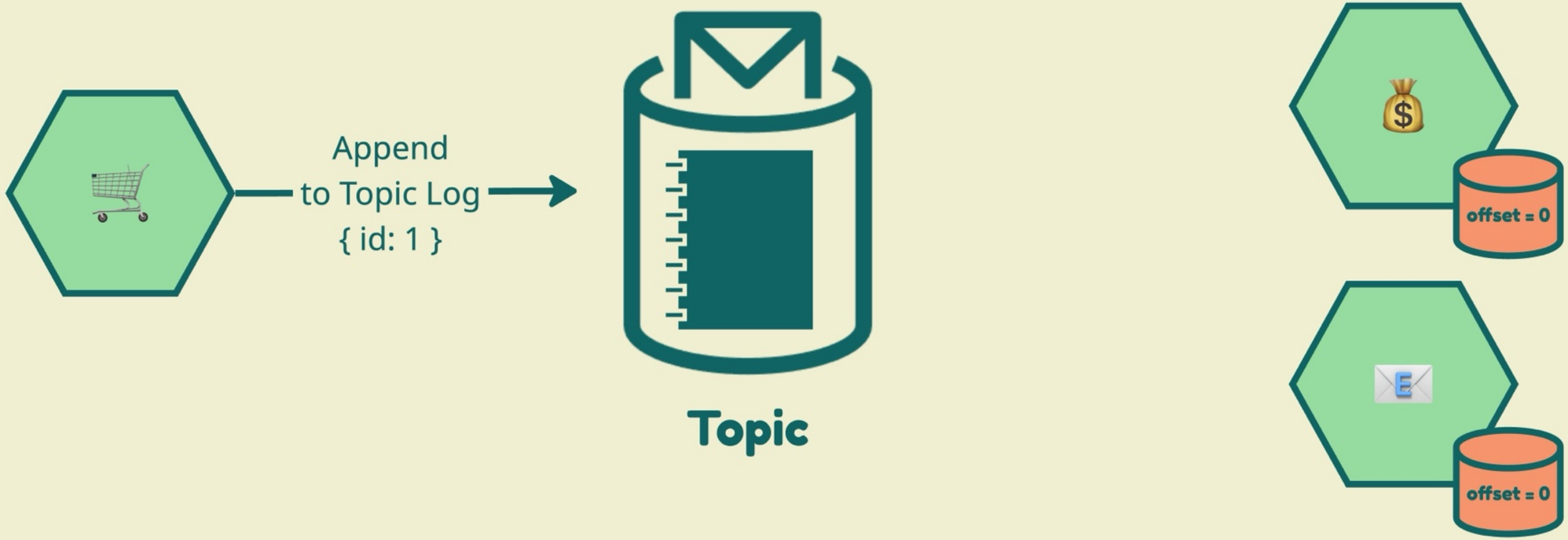


Message Routing

Queues



Streams



Message Routing

Queues





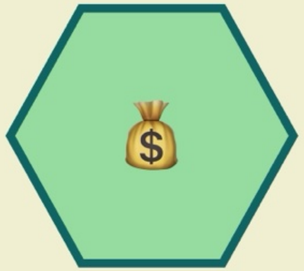
Topic



💰 -Queue



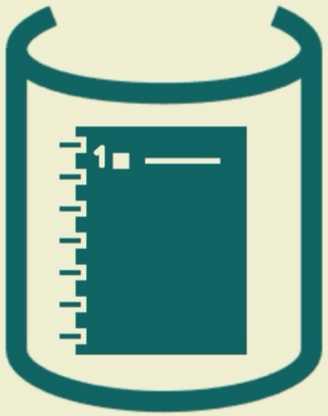
📧 -Queue



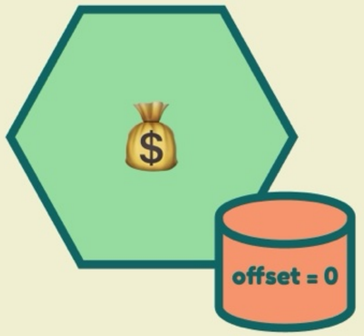


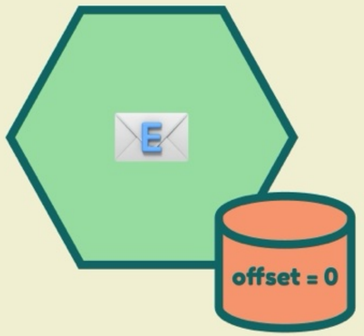
Streams



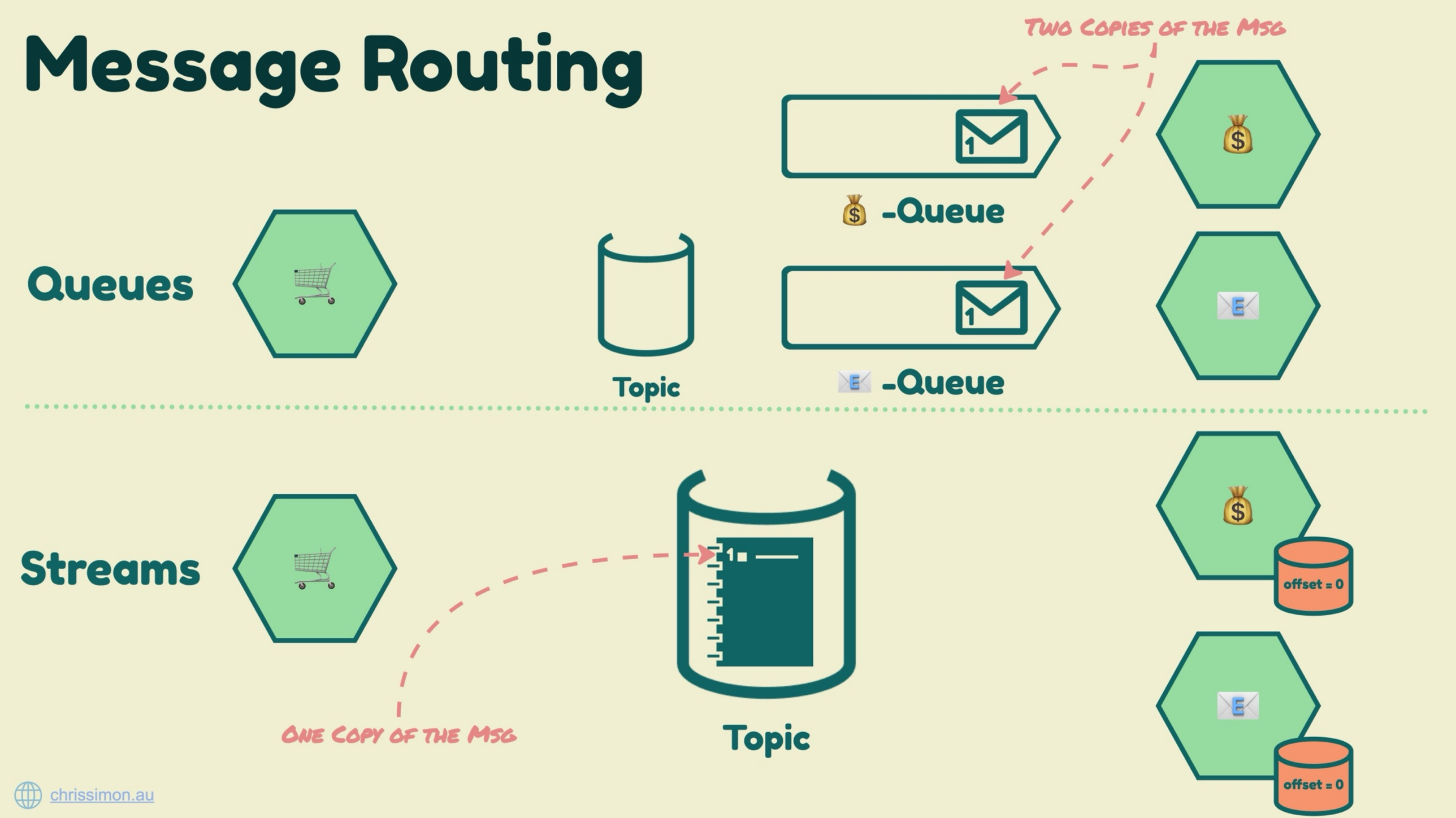


Topic





Message Routing



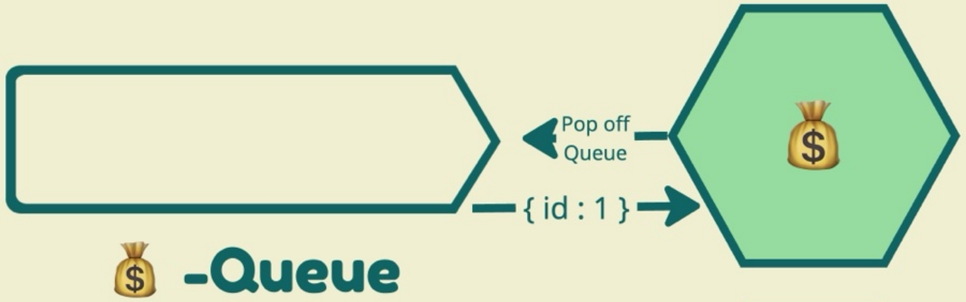
Message Routing

Queues





Topic



💰 -Queue

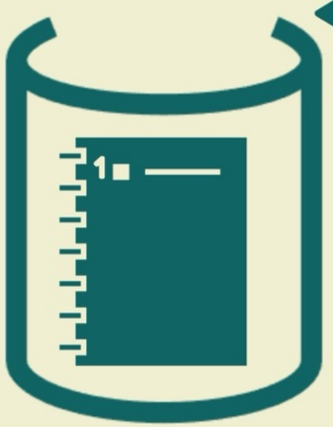


📧 -Queue

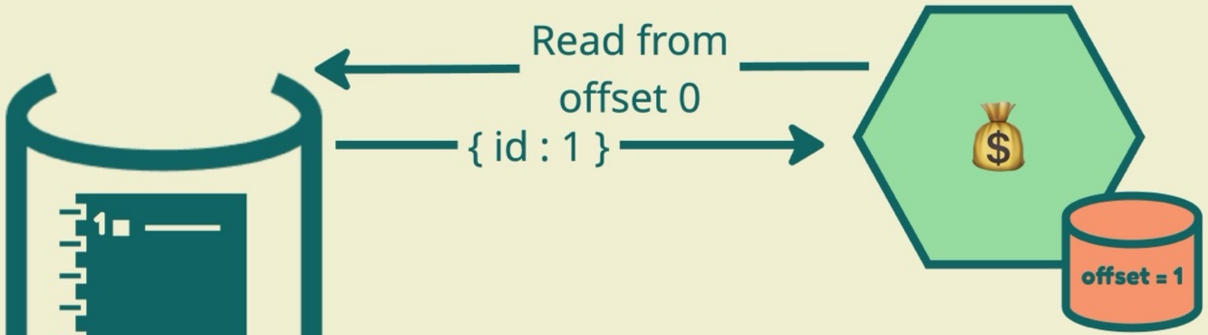


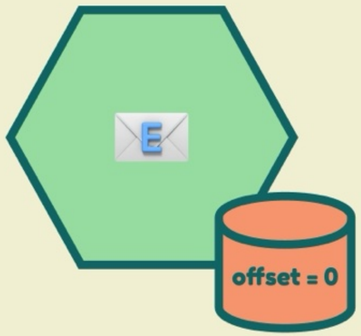
Streams





Topic





Message Routing

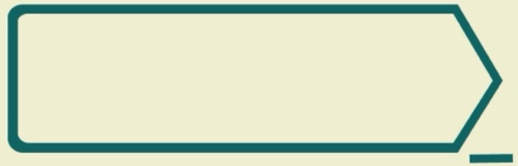
Queues



Topic



💰 -Queue

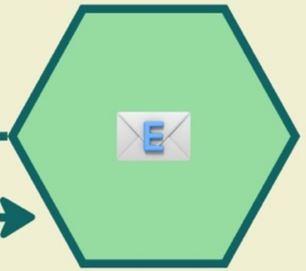


-Queue

Pop off Queue

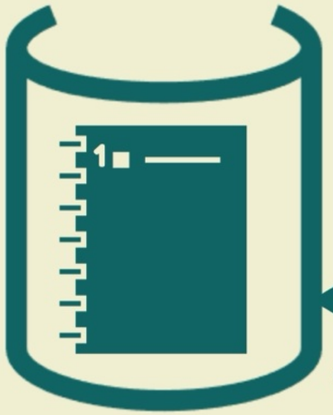
{ id : 1 }





Streams

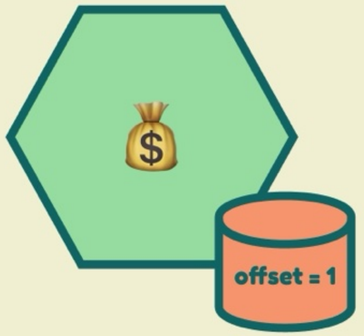


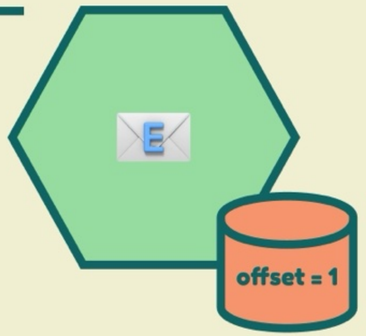


Topic

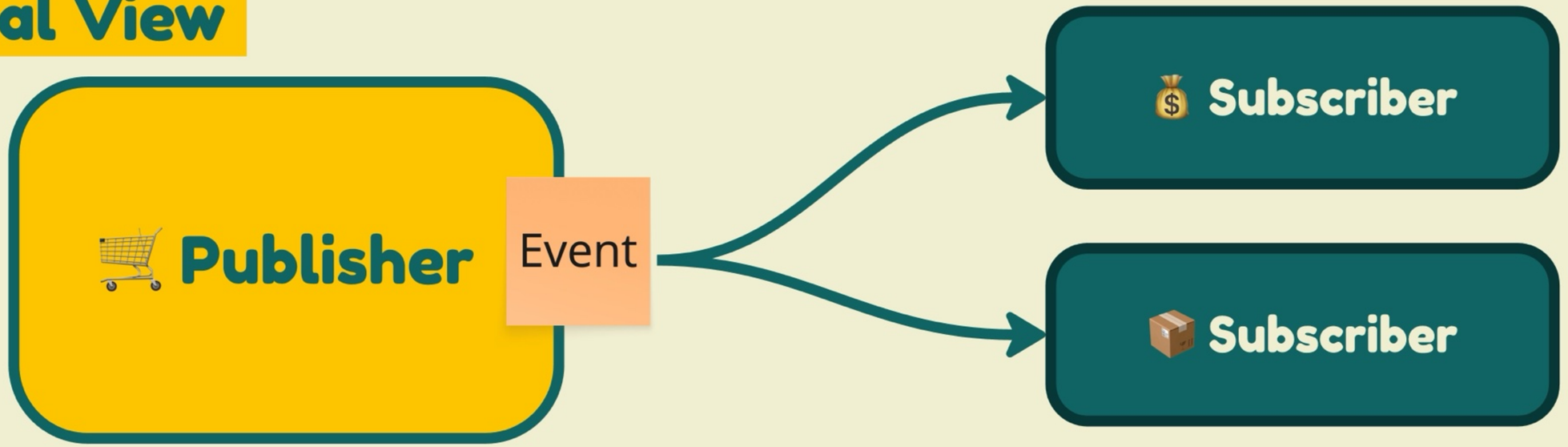
Read from offset 0

{ id : 1 }

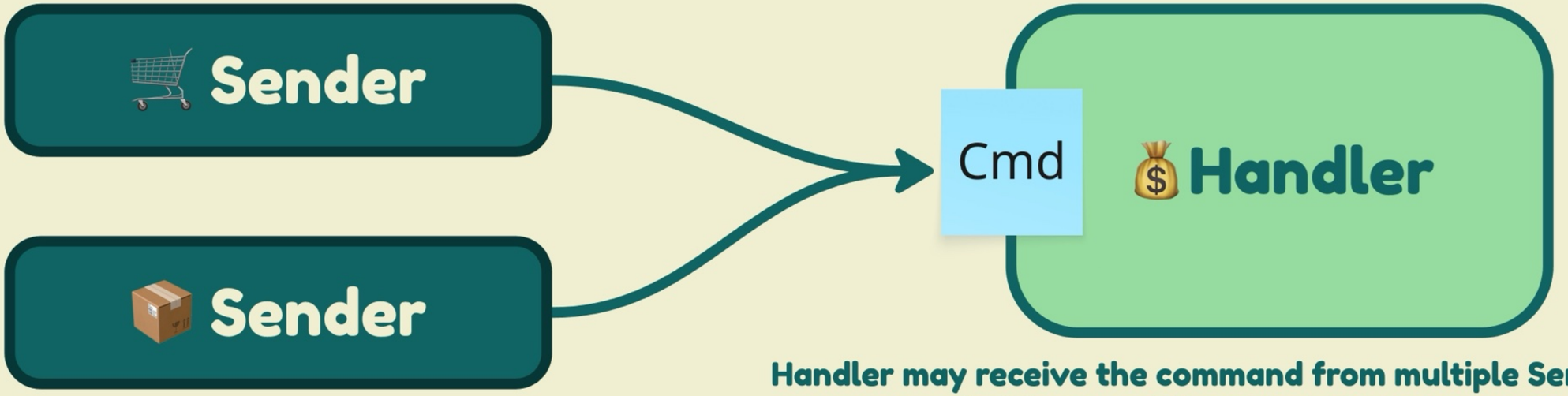




Logical View



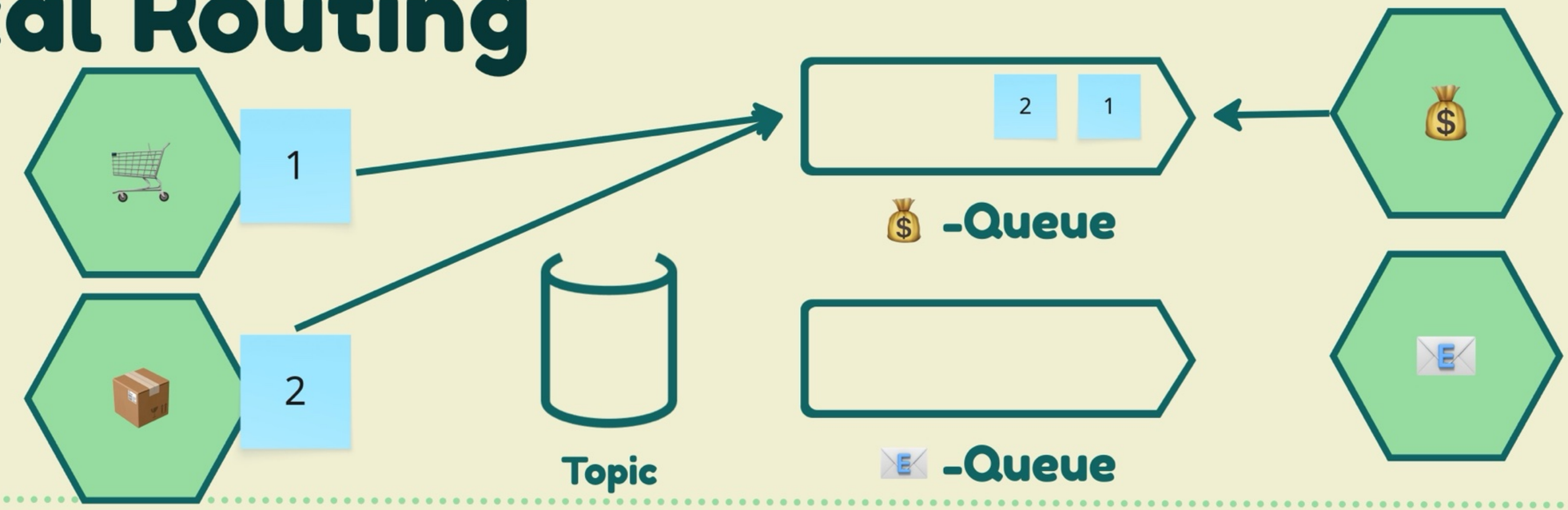
Publisher may have multiple Subscribers - it's not coupled to any of them, but they are coupled to it.



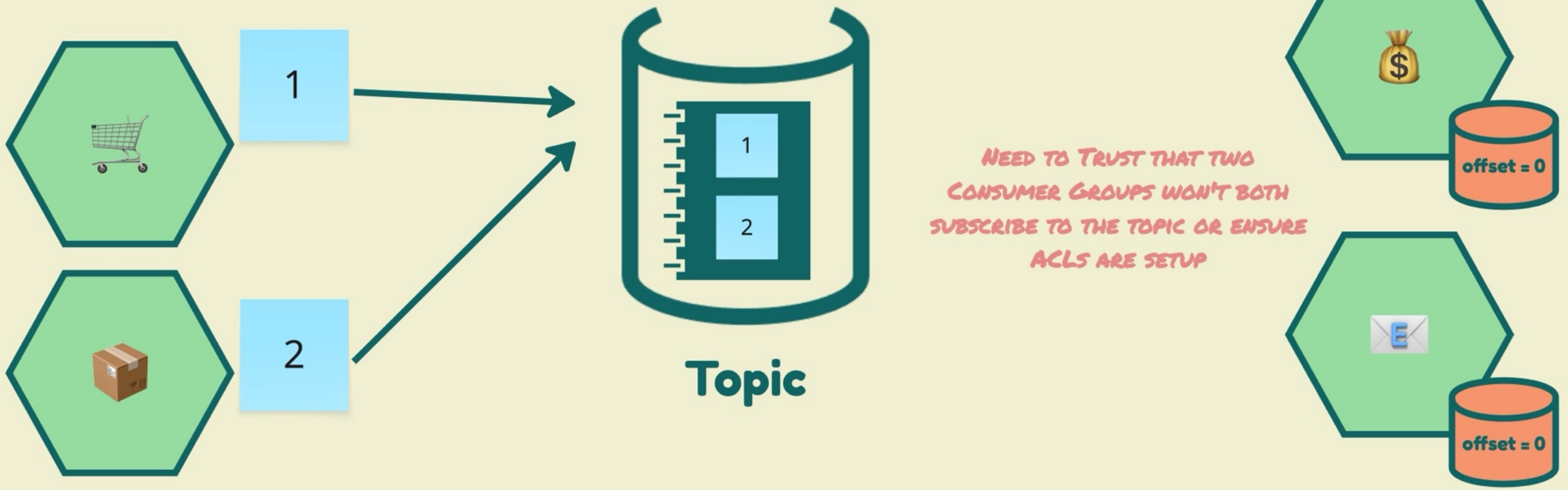
Handler may receive the command from multiple Senders - it's not coupled to any of them, but they are coupled to it.

Logical Routing

Queues



Streams



Need to Trust that two Consumer Groups won't both subscribe to the topic or ensure ACLs are setup

Outcome: Faster adaptation to changing conditions. Compounding performance improvement over time without manual model maintenance.

5. Adaptive Workflow Orchestration

Workflows are dynamically assembled at runtime based on current entity state, active policies, and contextual signals — not executed from predefined static DAGs. Each workflow step is initiated by a command event and confirmed by a completion event before the next step begins.

Outcome: Workflows adapt to context without separate process definitions for each case. Reduced exception handling overhead and improved end-to-end completion rates.

6. Self-Healing Infrastructure

Infrastructure telemetry streams feed continuous health signals into the streaming backbone. Stream processing detects degradation before failure thresholds are reached. Decisioning agents select remediation strategies — restart, failover, circuit break — and execution agents verify recovery within the same control loop.

Outcome: Higher system availability. Significant reduction in on-call burden for routine infrastructure failures.

chrissimon.au


Alanis Morissette - Ironic (1996)
♪ And isn't it ironic... ♪
chrissimon.au


Send a command-event and...
I can't even finish this rhyme, it just makes no sense
chrissimon.au


WTF??
Send a command-event and...
I can't even finish this rhyme, it just makes no sense

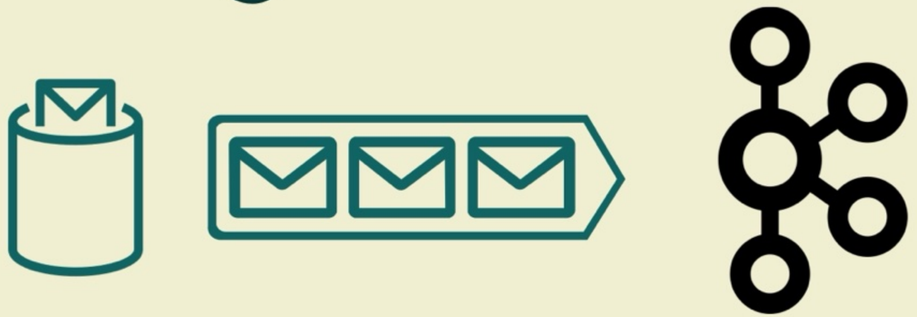
Broker Differences - Summary

Broker Type:	Queue	Stream / Distributed Log
Message Read Status Tracking	Broker	Consumer A
Poison Message Handling	Broker	Consumer A
Can auto-scale Consumer Clusters	Yes	No
Can auto-recover from consumer process failure	Yes	Yes (ish)
Support for Semantic Commands vs Events	Yes	No
Storage of Historical Messages	Consumer A	Broker
New consumers can access historical messages	No	Yes

BEYOND PARTITION LIMIT

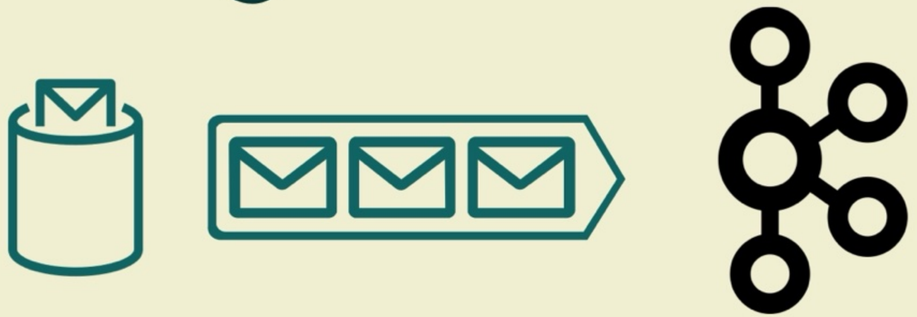
CLIENT LIBRARIES DO THE HEAVY LIFTING

Event Sourcing



Queues and Streams are about inter-process communication

Event Sourcing



Queues and Streams are about inter-process communication

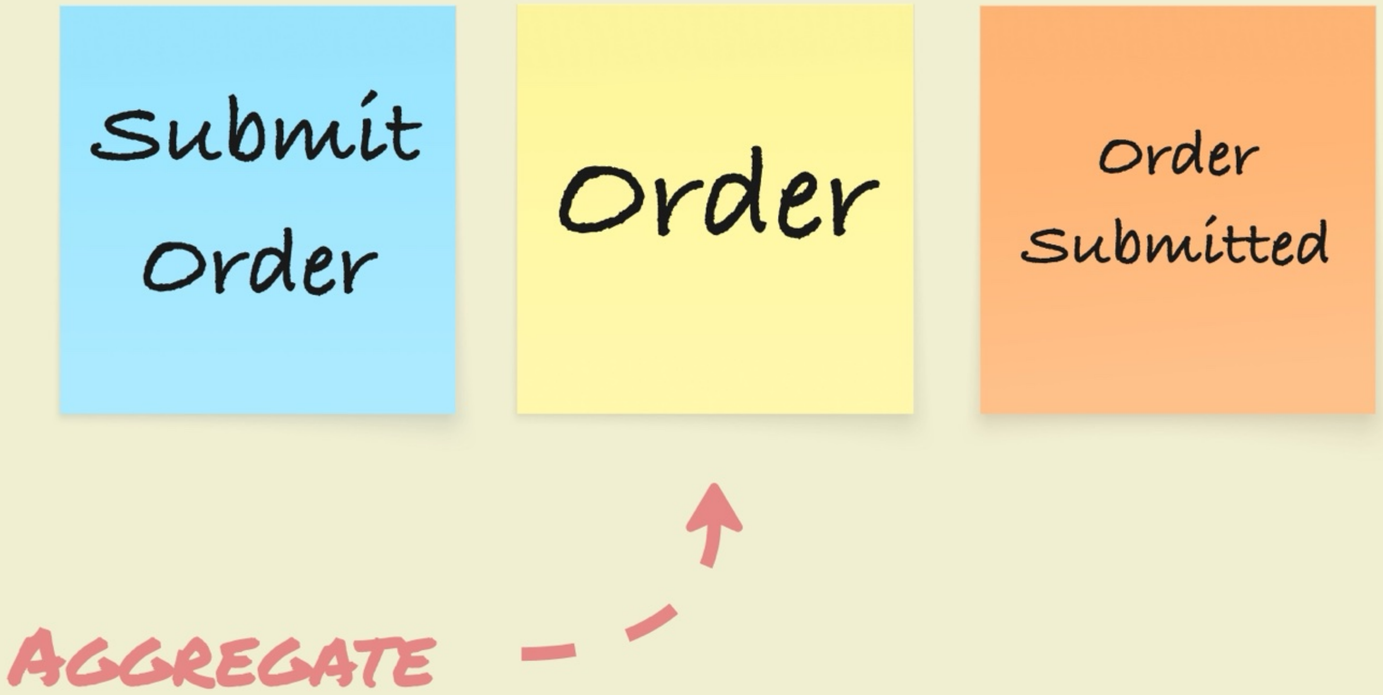
EntityStream				
1	2023-11-01T01:01:10	OrderShipped	<guid>	{ "field": "value" }
2	2023-11-01T01:02:10	PaymentSuccessful	<guid>	{ "field": "value" }

Event Stores are about storing State as events

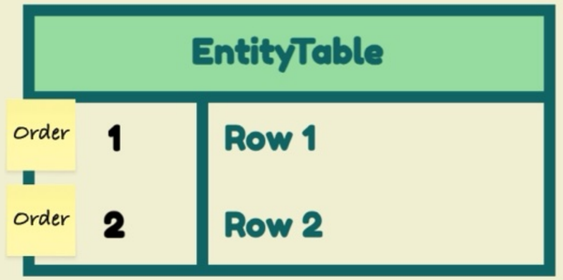
Event Sourcing



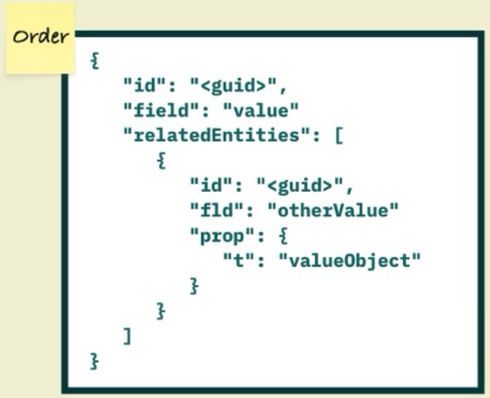
Event Sourcing



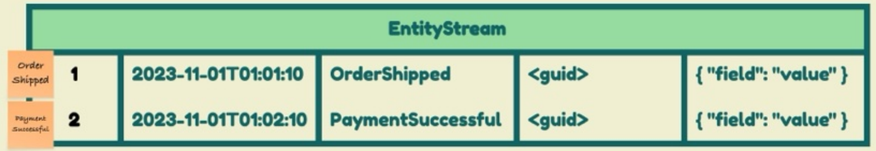
Storing State



Map to Relational Table



Serialize to NoSQL



Log of Events in an Event Store

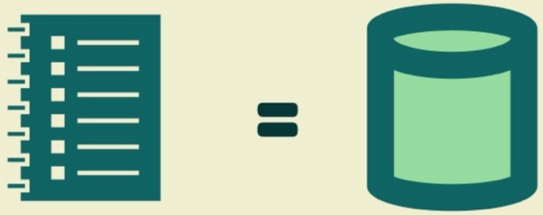
State Persistence

May publish events as a side-effect of persisting state.

Event Sourcing

State is computed 'on the fly' as a side-effect of persisting the event

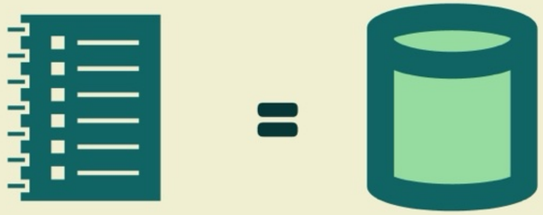
Streams vs Event Sourcing



Distributed Logs are optimised for:

```
SELECT * FROM Events WHERE Topic = %Topic% AND Offset > %Last_Offset%
```

Streams vs Event Sourcing



Distributed Logs are optimised for:

```
SELECT * FROM Events WHERE Topic = %Topic% AND Offset > %Last_Offset%
```

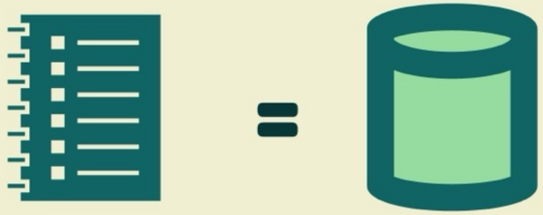
EntityStream				
1	2023-11-01T01:01:10	OrderShipped	<guid>	{ "field": "value" }
2	2023-11-01T01:02:10	PaymentSuccessful	<guid>	{ "field": "value" }



Event Stores are optimised for:

```
SELECT * FROM Events WHERE StreamId = %EntityId%
```

Streams vs Event Sourcing



A Stream Per Topic

Distributed Logs are optimised for:

```
SELECT * FROM Events WHERE Topic = %Topic% AND Offset > %Last_Offset%
```

EntityStream				
1	2023-11-01T01:01:10	OrderShipped	<guid>	{ "field": "value" }
2	2023-11-01T01:02:10	PaymentSuccessful	<guid>	{ "field": "value" }

=



A Stream Per Entity Instance

Event Stores are optimised for:

```
SELECT * FROM Events WHERE StreamId = %EntityId%
```

Messaging as the Single Source of Truth

Technology > Big Ideas

AUG 2, 2017 READ TIME: 14 MIN

This post discusses Event Sourcing in the context of Apache Kafka®, examining the need for a single source of truth that spans entire service estates.

Events are Truth

One of the trickiest parts of building microservices is managing data. The reason is simple enough. In traditional systems there is a system of record, typically a database; the trusted resource for facts. In service-oriented systems there is no single source of truth, there are many. Different datasets are owned by a variety of different services. This is by design. Services will have a range of different roles: individual pieces in a far larger puzzle.

Get started free w
Confluent

Get started →

Watch demo: Kafk
streaming in 10 m


Alanis Morissette - Ironic (1996)
♪ And isn't it ironic... ♪
chrissimon.au


It's an event streaming log
When your state's out of date

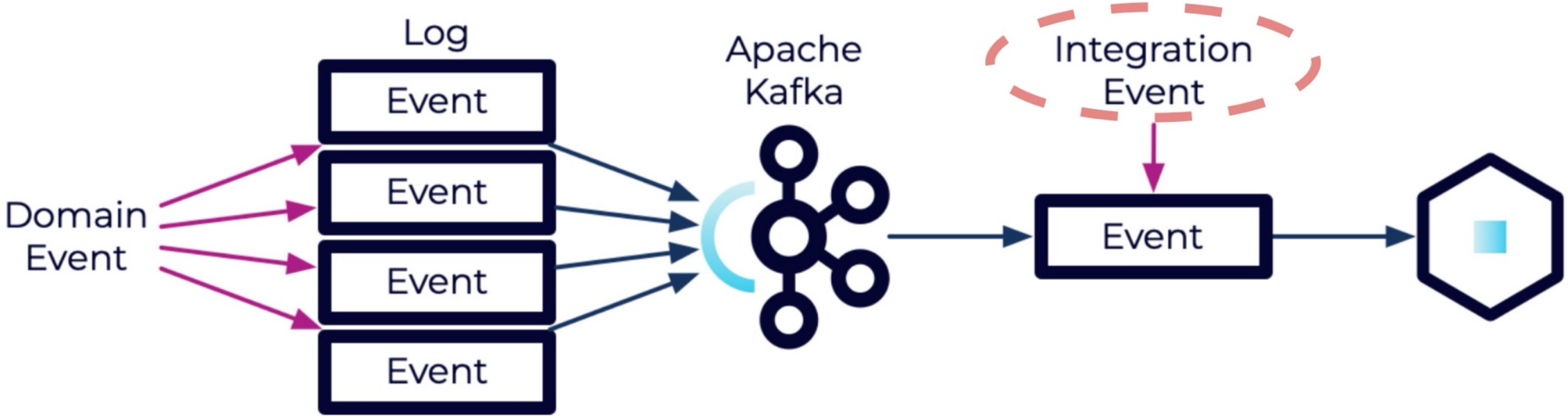
What is Event Sourcing?

Event sourcing is an approach for tracking changes to application state, stored as a sequence of events. Each change is associated with an event, which contains the timestamp and nature of the specific change. Instead of working with a sequence of static snapshots of our data, we use event sourcing to capture and store these events.

Using event sourcing when designing software puts a greater emphasis on how data evolves over time. It captures the journey, rather than just the destination. This allows systems based on event sourcing to have access to richer data that can be used for analytical purposes, but also to help develop new features. Event-sourced systems—and the developers who build them—are more capable of evolving rapidly to meet the changing needs of the business.

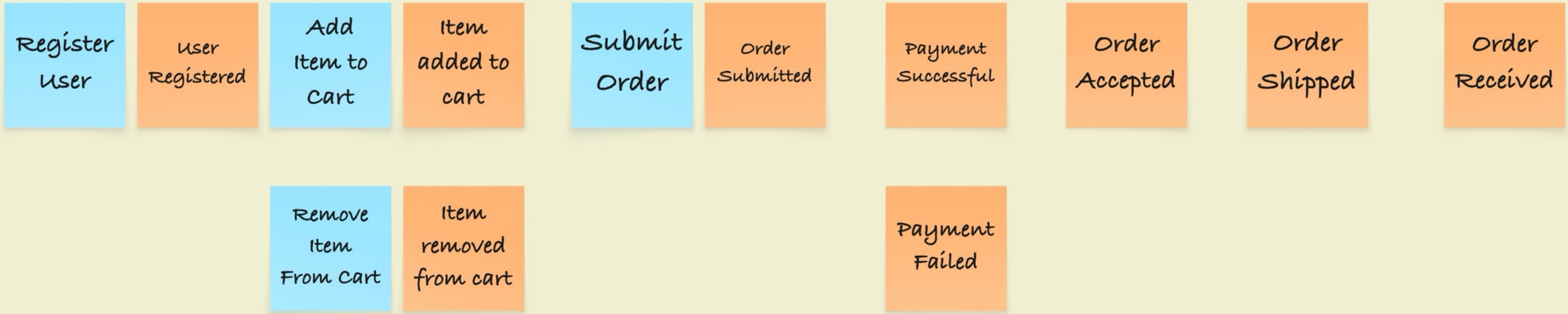
Furthermore, it is a mistake to assume that every event stored in an event-sourced system is also emitted to Kafka. In reality, the events that are emitted to Kafka can be quite different from what has been stored. The stored events are often called domain events and represent an internal implementation detail. Meanwhile the emitted events are often called integration events and represent a public-facing API.

It is quite common to have a translation step that converts the domain events into their corresponding integration events. This allows the application to evolve internally without impacting the public API.

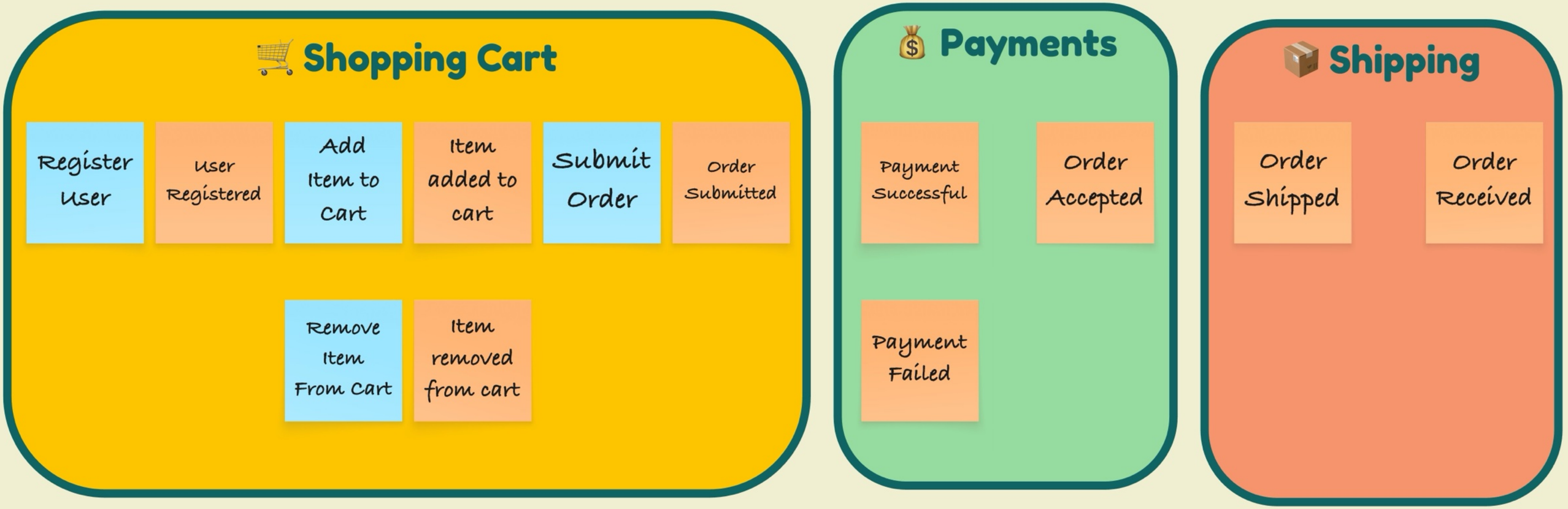


Event Storm - Logical View

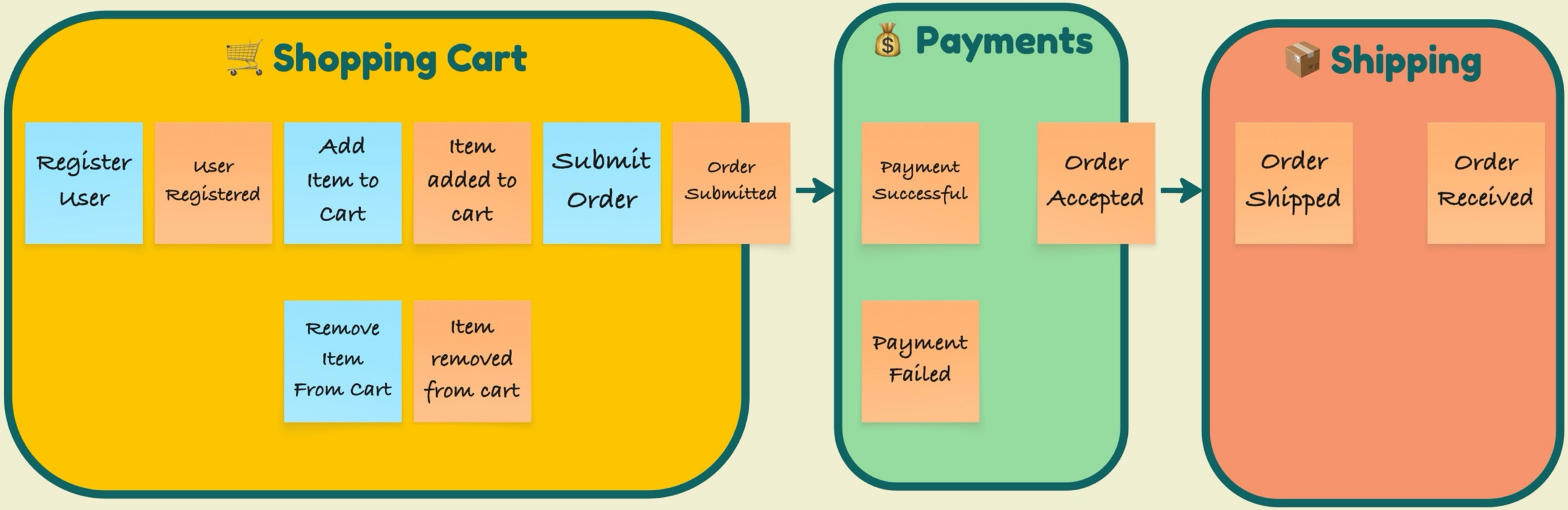
Logical View



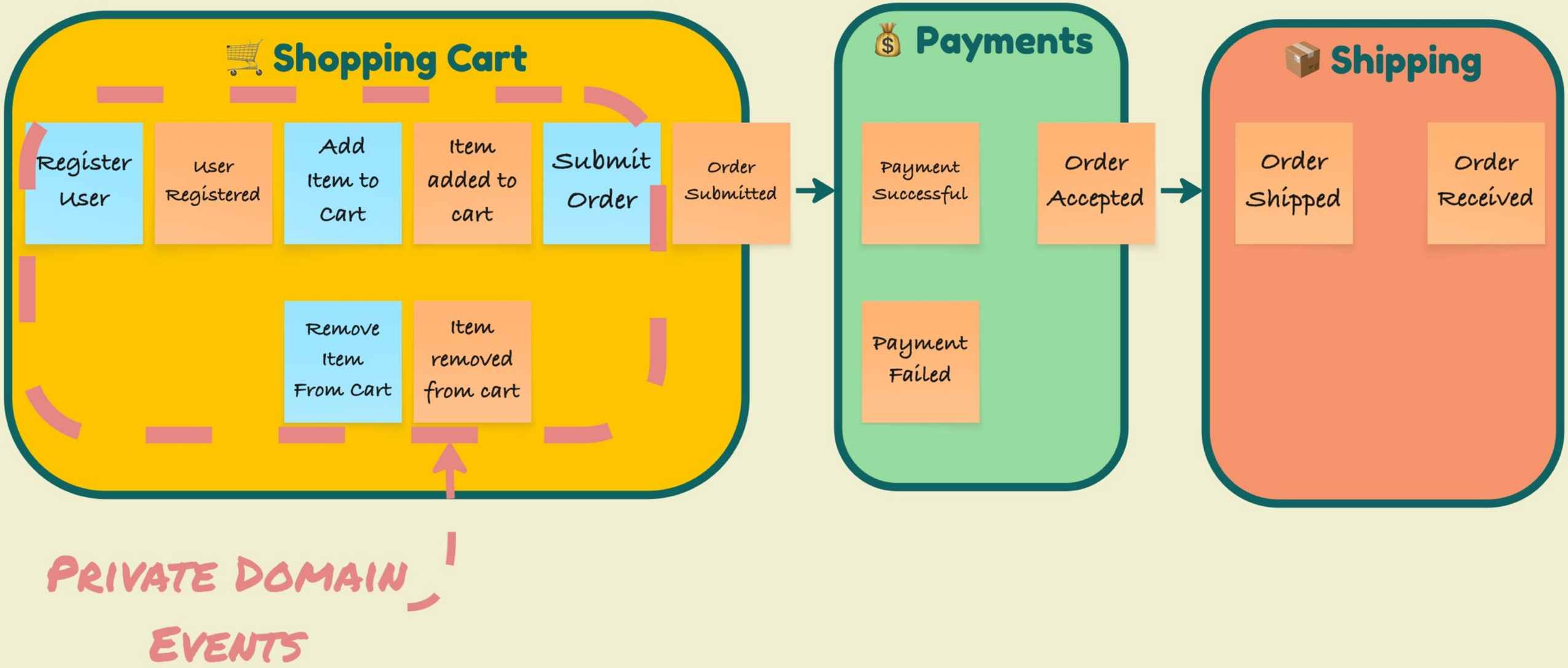
Event Storm - Bounded Contexts



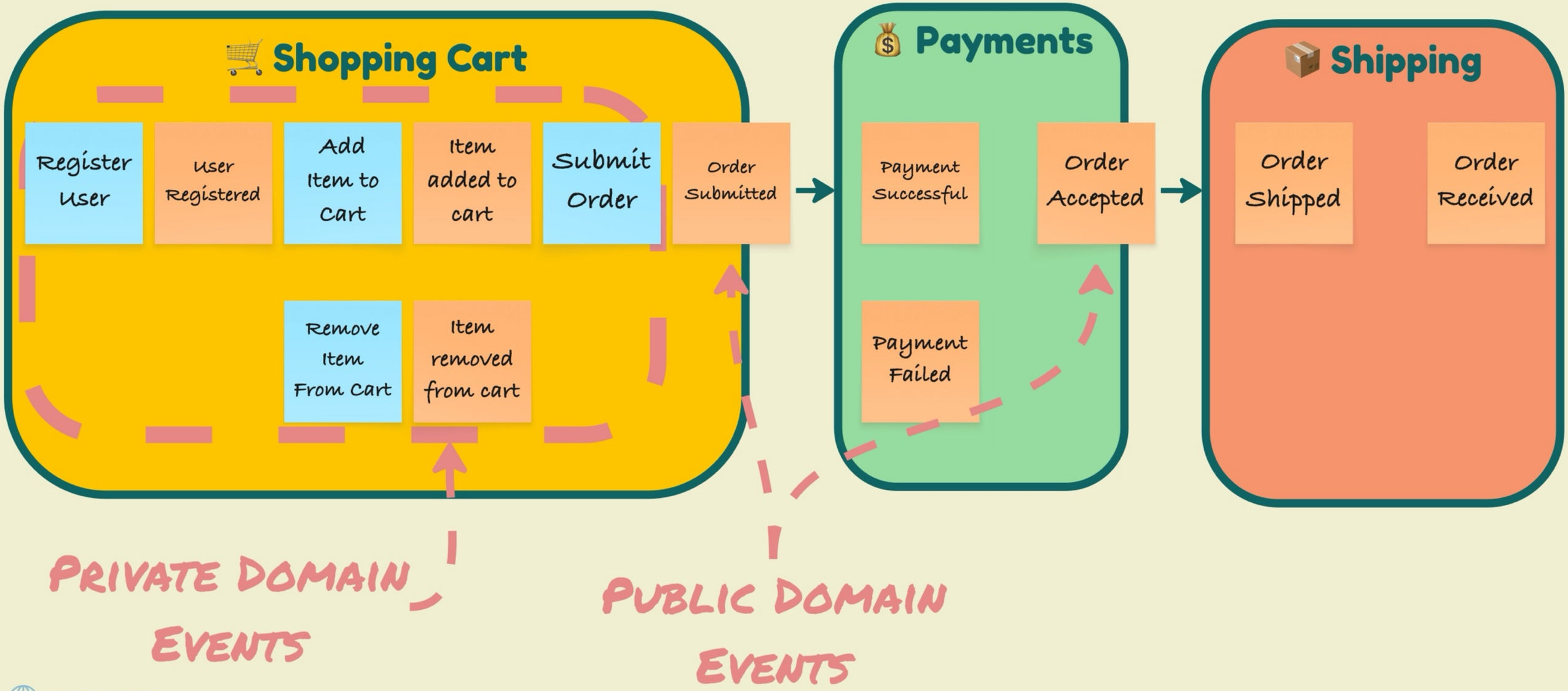
Event Storm - Bounded Contexts



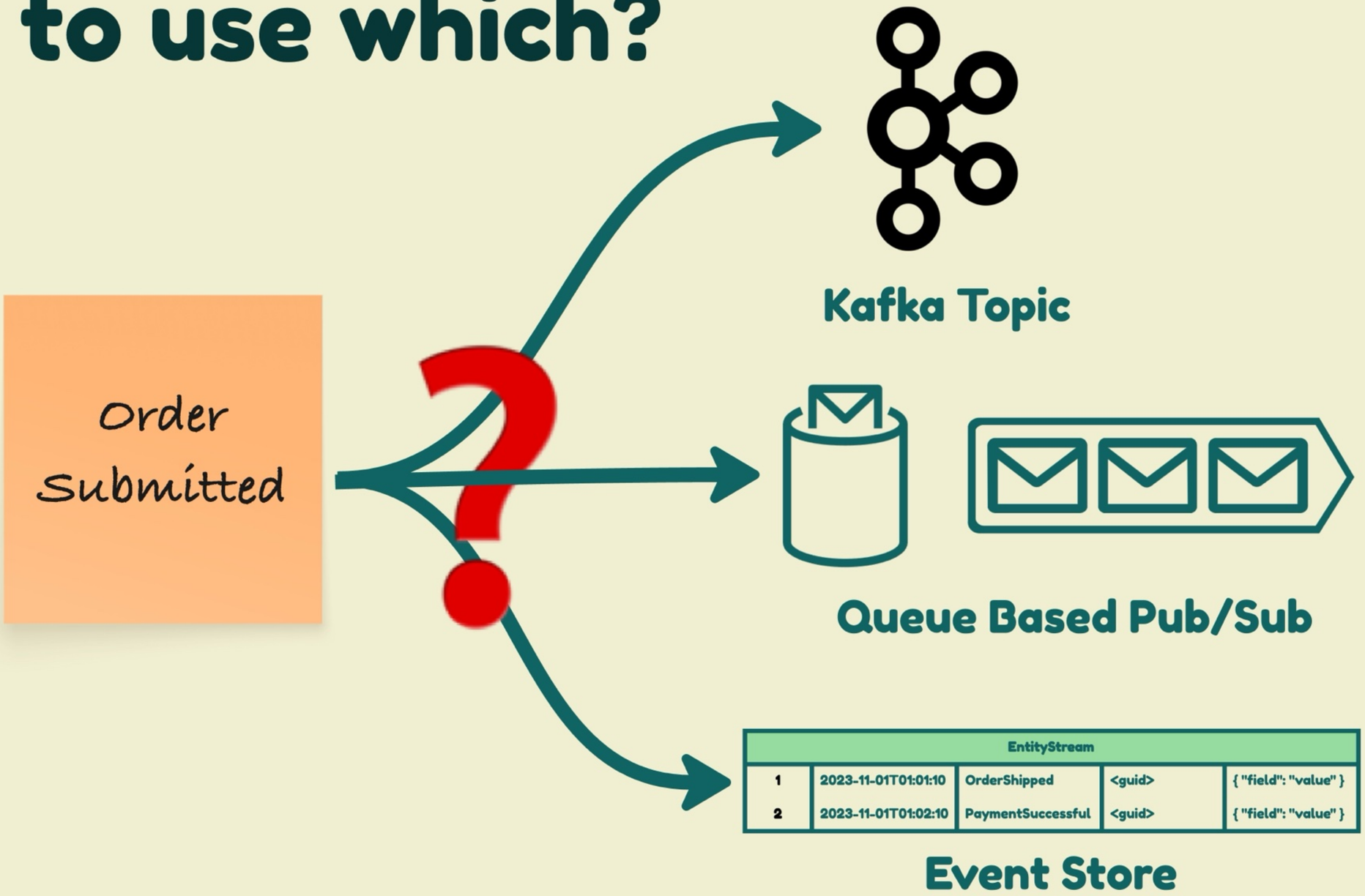
Private Events



Public Events



When to use which?



Private Domain Events




```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Private Domain Events




```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Purpose:

To use events as the 'source of truth' / 'historical record' of the bounded context

Private Domain Events



Purpose:

To use events as the 'source of truth' / 'historical record' of the bounded context

```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

✅ **Event Sourcing with an Event Store**

EntityStream				
1	2023-11-01T01:01:10	OrderShipped	<guid>	{ "field": "value" }
2	2023-11-01T01:02:10	PaymentSuccessful	<guid>	{ "field": "value" }

Private Domain Events




```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Purpose:

To use events to trigger the next step in an internal workflow (Business Process Choreography)

Private Domain Events



Each event is critical

Purpose:

To use events to trigger the next step in an internal workflow (Business Process Choreography)

```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Private Domain Events

Each event is critical



Purpose:

To use events to trigger the next step in an internal workflow (Business Process Choreography)

```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

✅ **Messaging with Queue Based PubSub**



Private Domain Events



Purpose:

To use events to trigger the next step in an internal workflow (Business Process Choreography)

```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

✅ **Messaging with Queue Based PubSub**



Private Domain Events



Purpose:

To use events to trigger the next step in an internal workflow (Business Process Choreography)

```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Messaging with Queue Based PubSub



Producer and Consumer belong to the same Bounded Context

Private Domain Events




```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Purpose:

IOT, clickstreams, anomaly detection, moving average threshold crossing, statistical aggregation over time

Private Domain Events

INDIVIDUAL EVENTS ARE LESS IMPORTANT THAN THE STREAM



Purpose:

IOT, clickstreams, anomaly detection, moving average threshold crossing, statistical aggregation over time

```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Private Domain Events

INDIVIDUAL EVENTS ARE LESS IMPORTANT THAN THE STREAM

🛒 **Shopping Cart**

Item added to cart

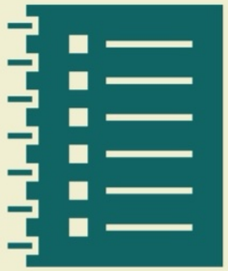
```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Purpose:

IOT, clickstreams, anomaly detection, moving average threshold crossing, statistical aggregation over time

✅

Streaming with a Distributed Log



Private Domain Events




```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Purpose:

IOT, clickstreams, anomaly detection, moving average threshold crossing, statistical aggregation over time

Streaming with a Distributed Log



Producer and Consumer belong to the same Bounded Context

Private Domain Events

Shopping Cart

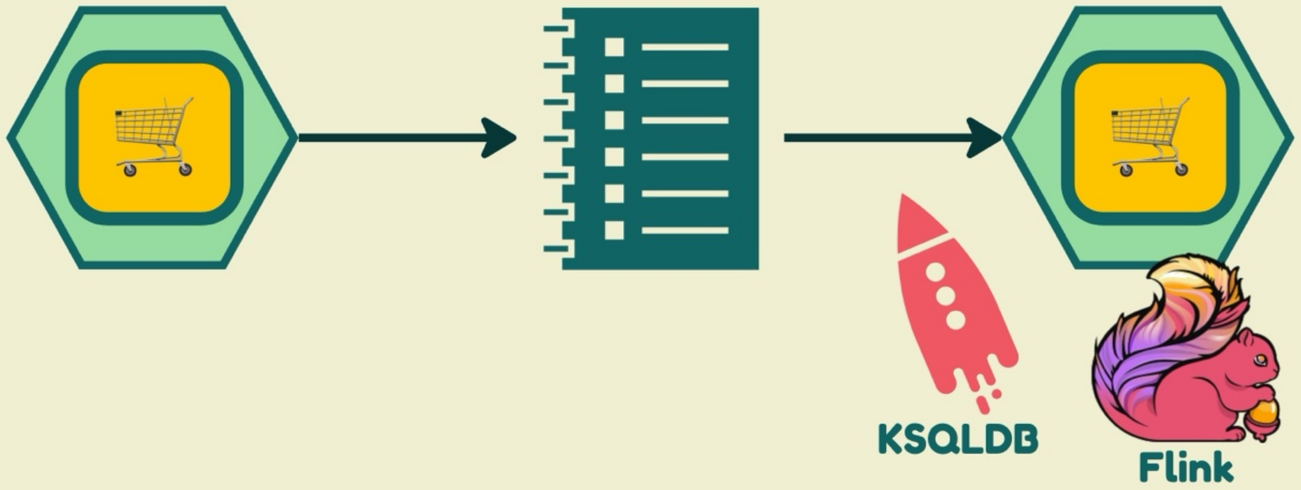
Item added to cart

```
{
    id: "<guid>",
    lots: 5,
    of: "",
    data: ""
}
```

Purpose:

IOT, clickstreams, anomaly detection, moving average threshold crossing, statistical aggregation over time

Streaming with a Distributed Log



Public Domain Events




```
{
    id: "<guid>"
}
```

Public Domain Events




```
{
    id: "<guid>"
}
```

Preferrably lightweight, small contract, less risk of change propagation

Public Domain Events

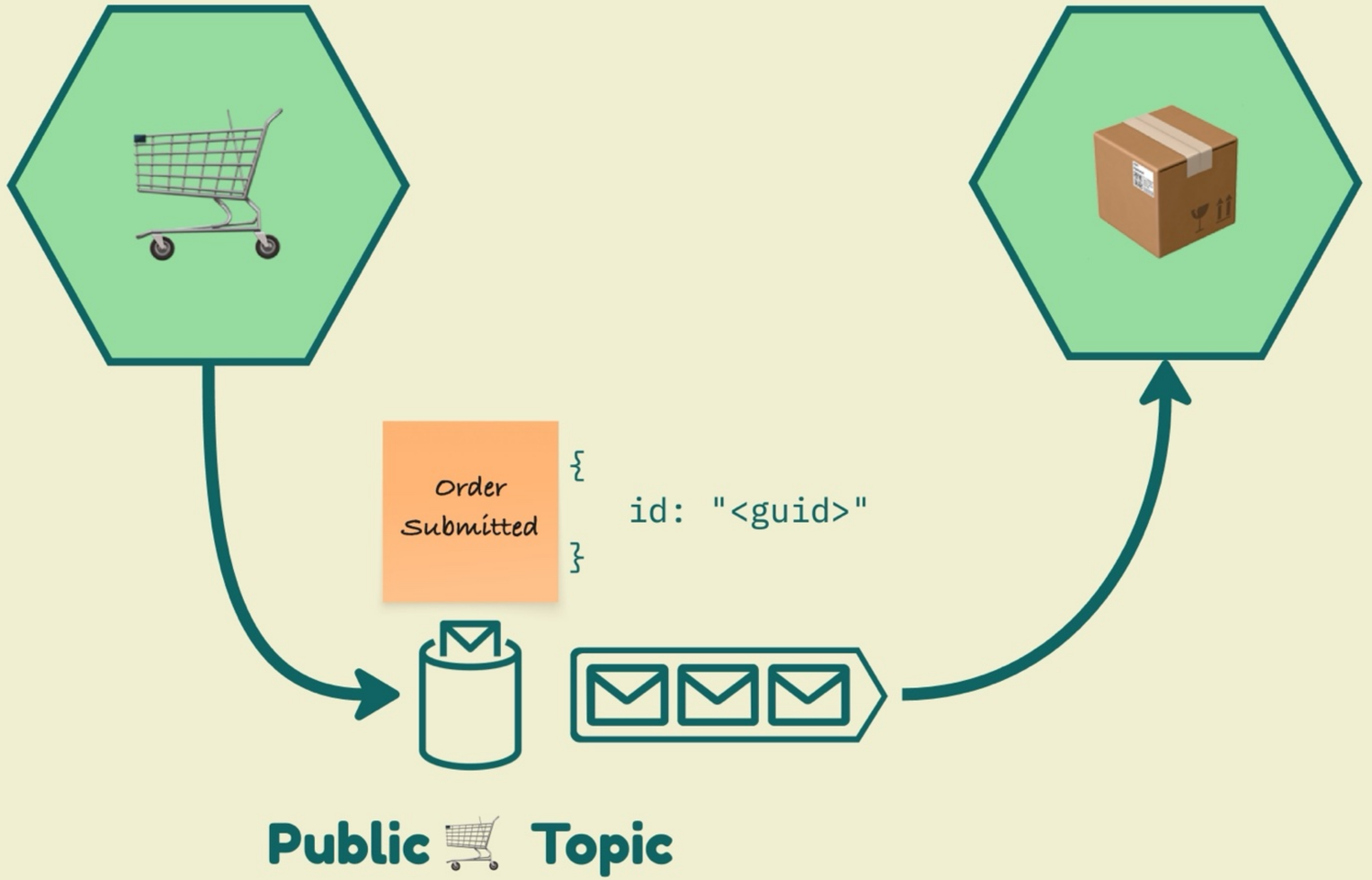



```
{
    id: "<guid>"
}
```

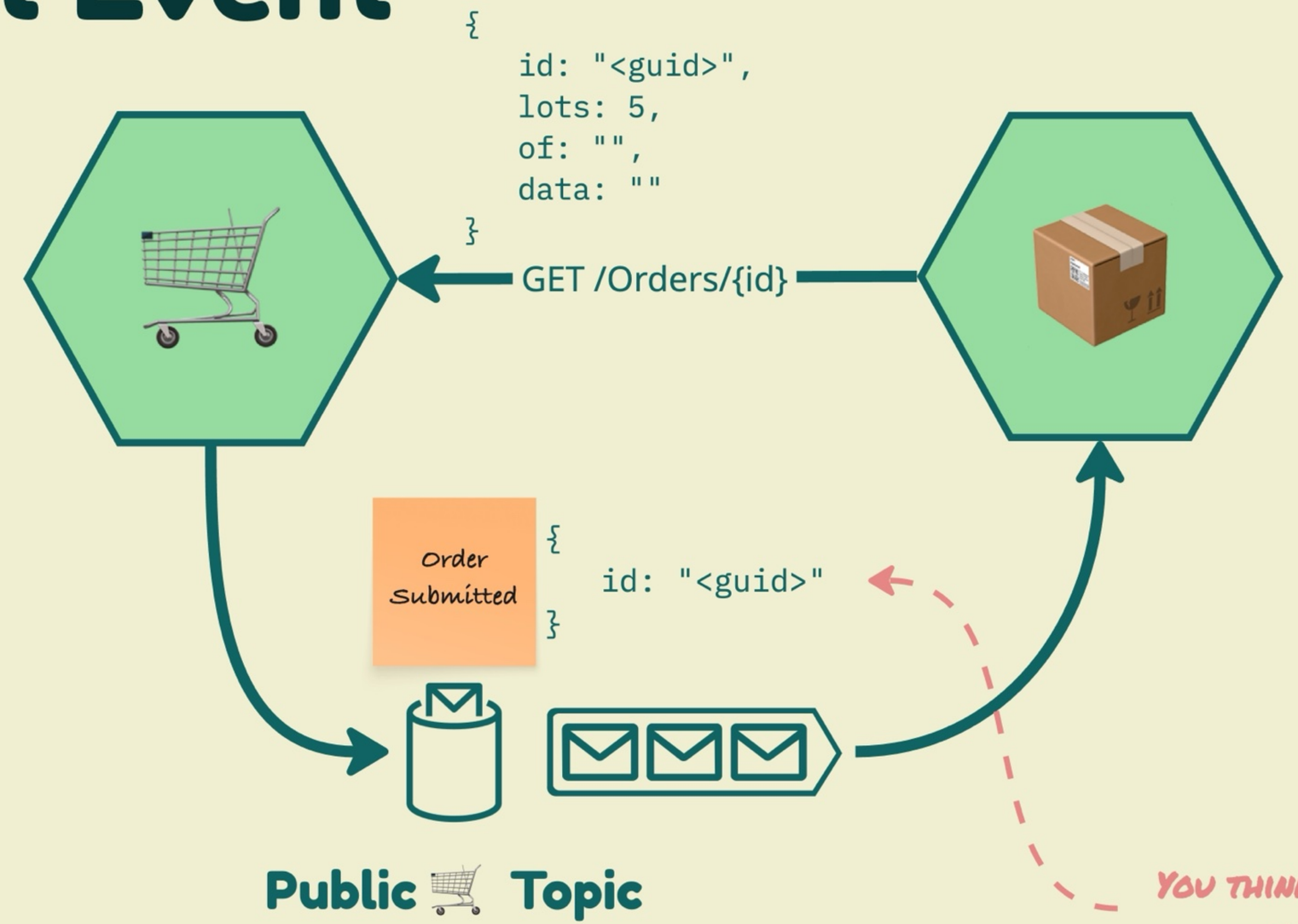
Preferrably lightweight, small contract, less risk of change propagation

But isn't this a 'clickbait' event?

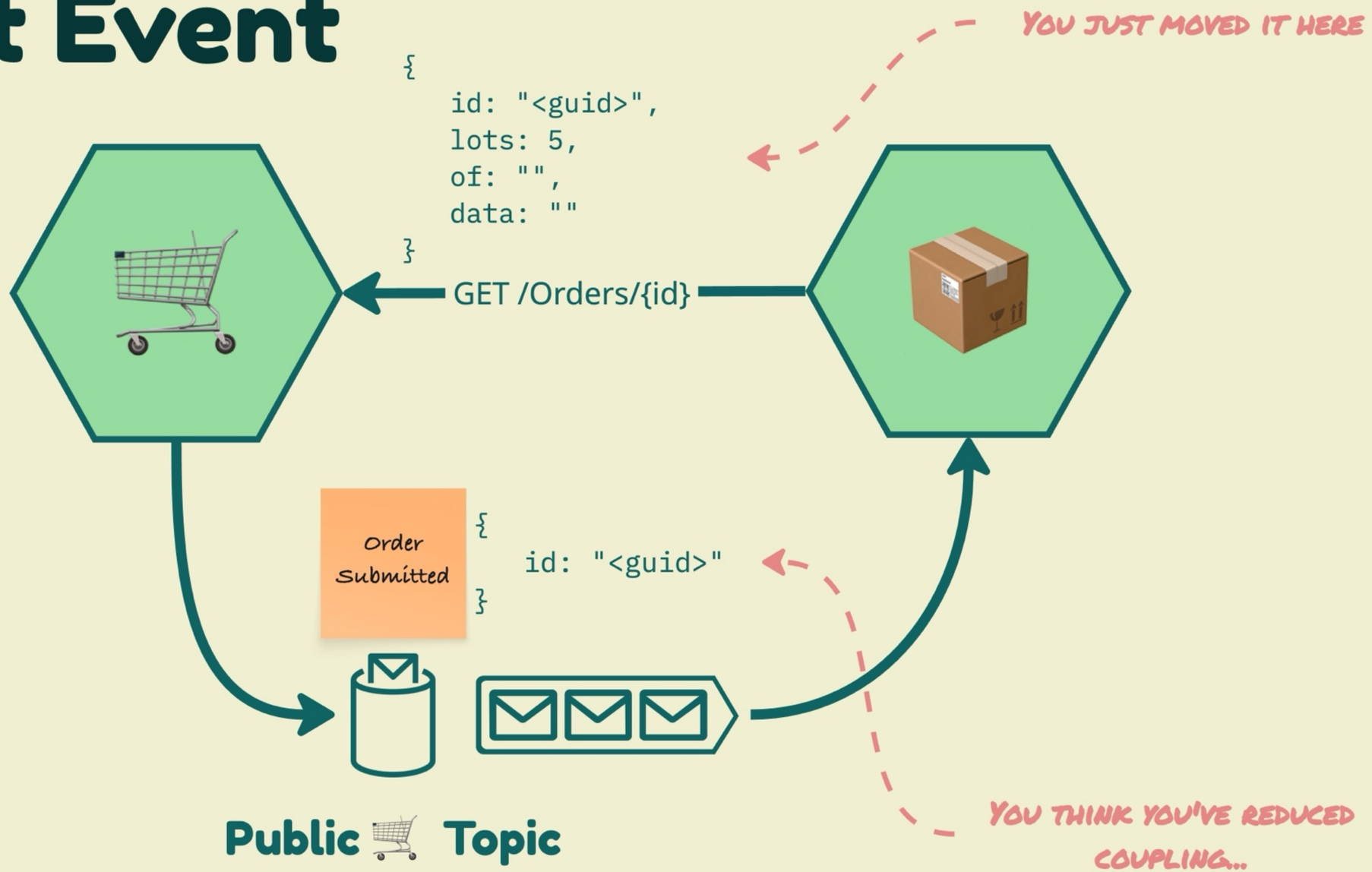
Clickbait Event



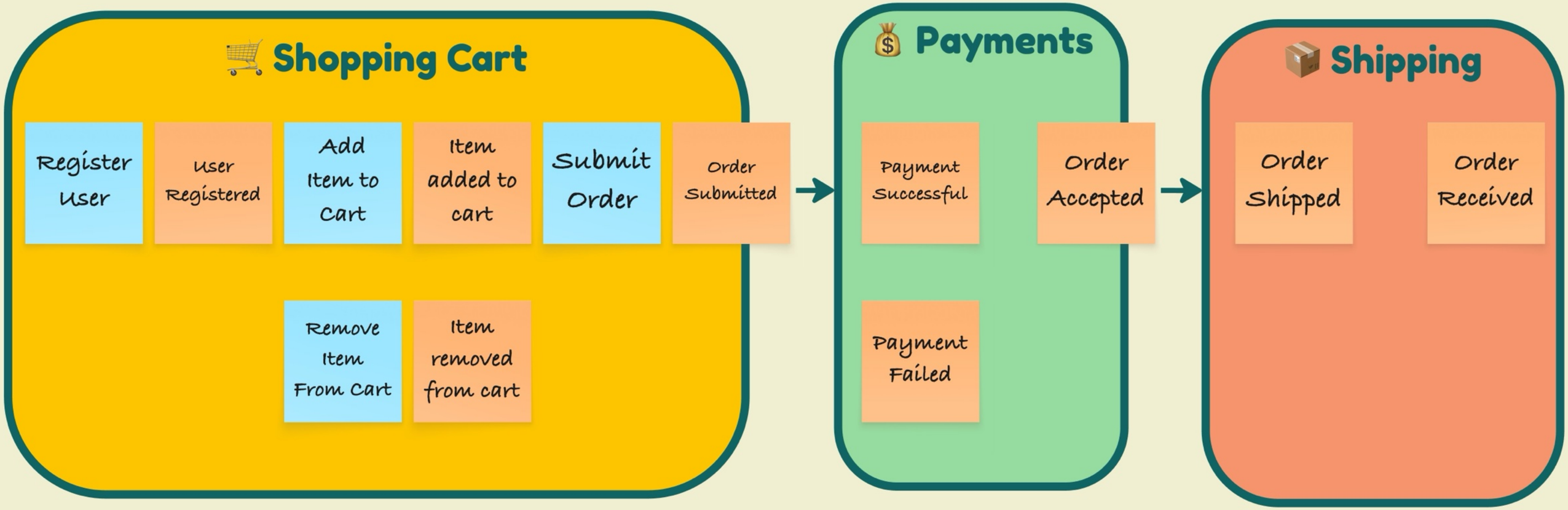
Clickbait Event



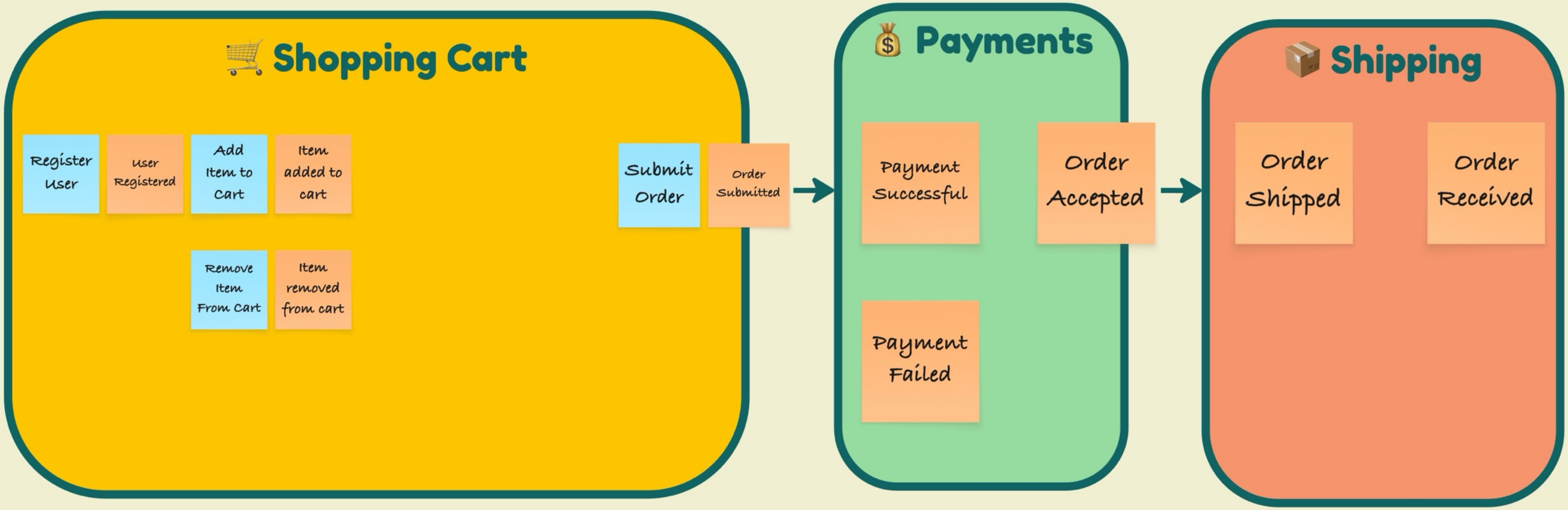
Clickbait Event



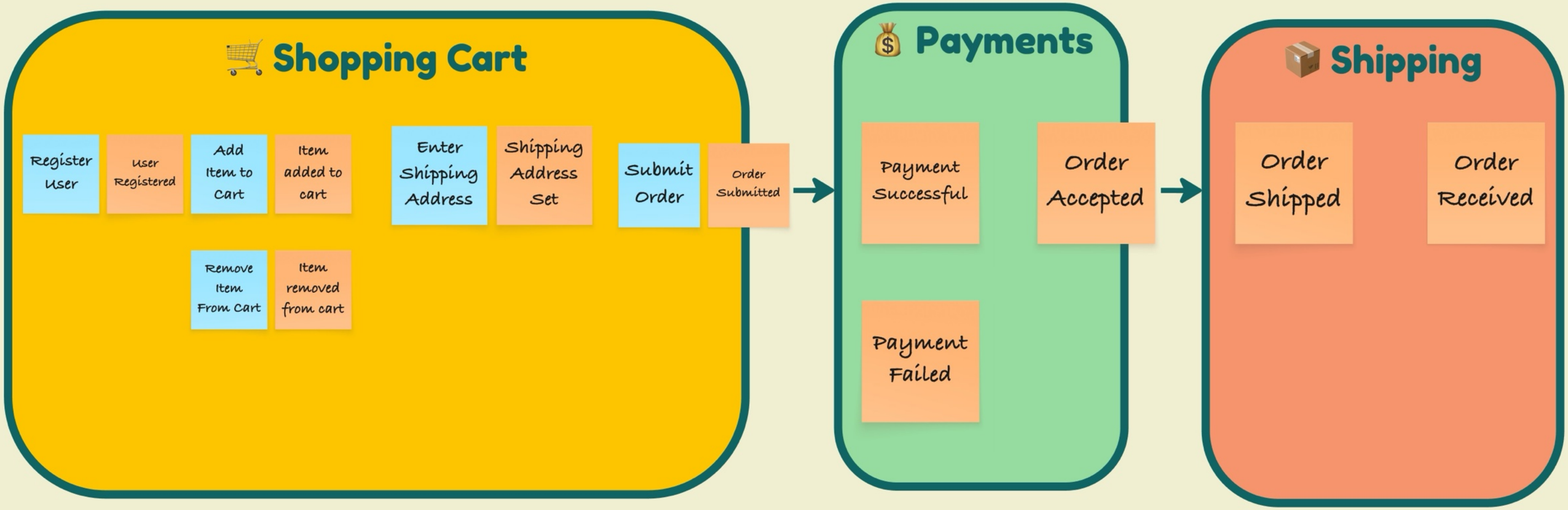
Event Storm - Bounded Contexts



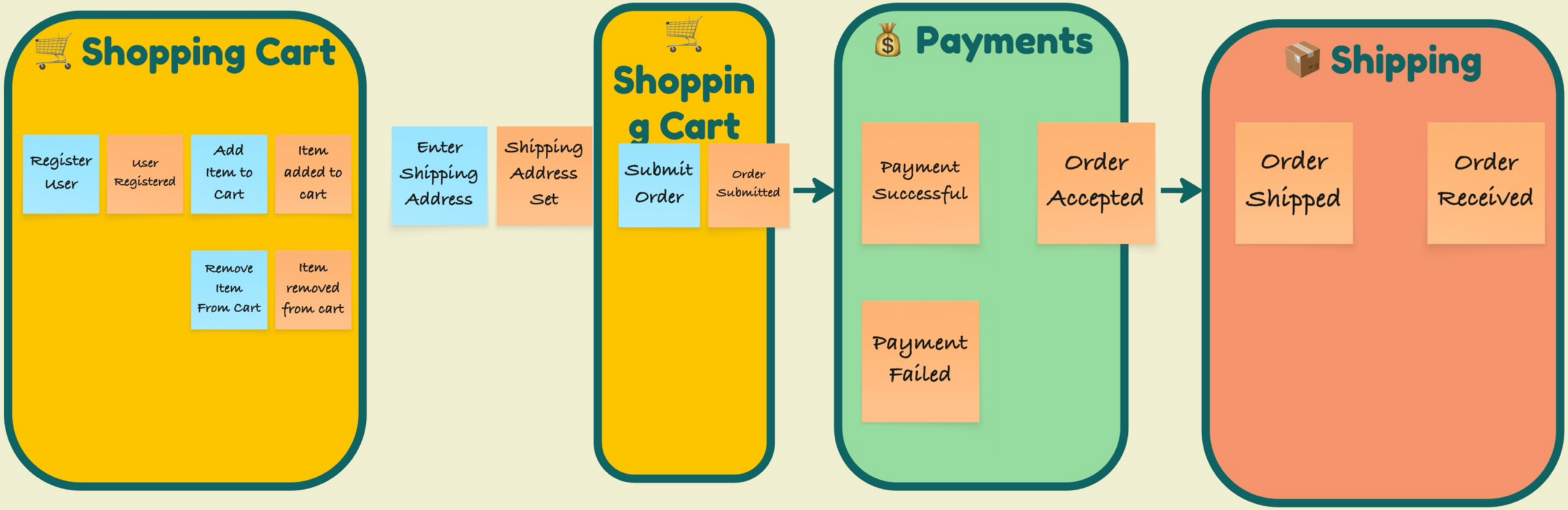
Event Storm - Bounded Contexts



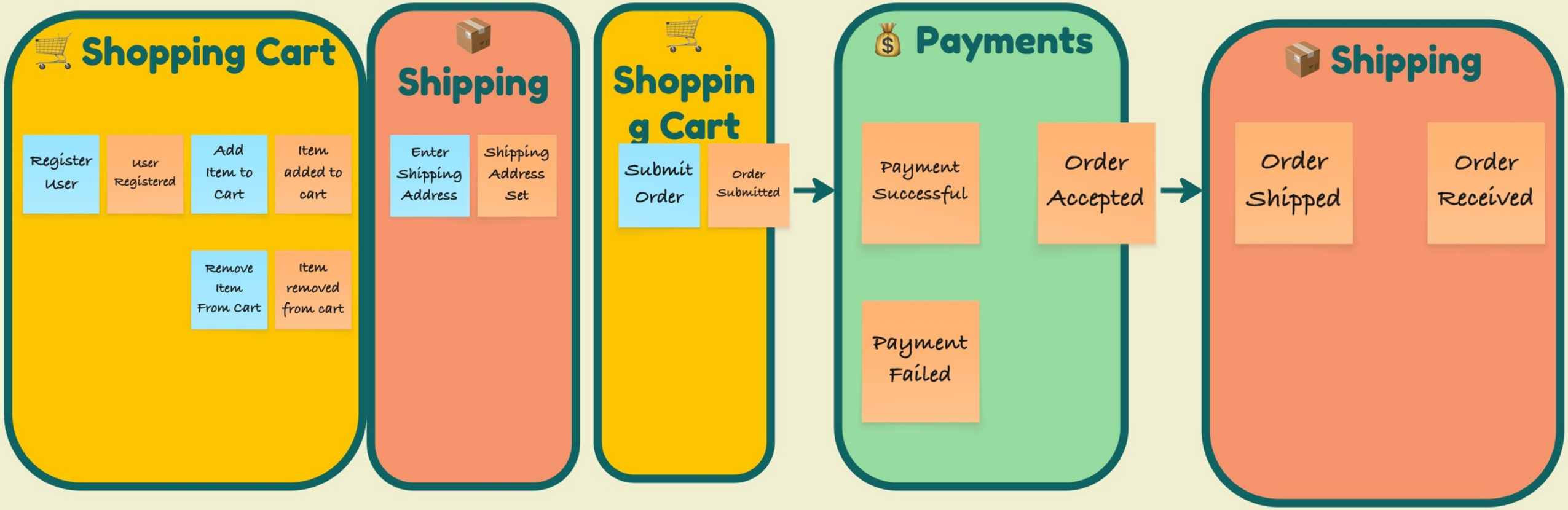
Event Storm - Bounded Contexts



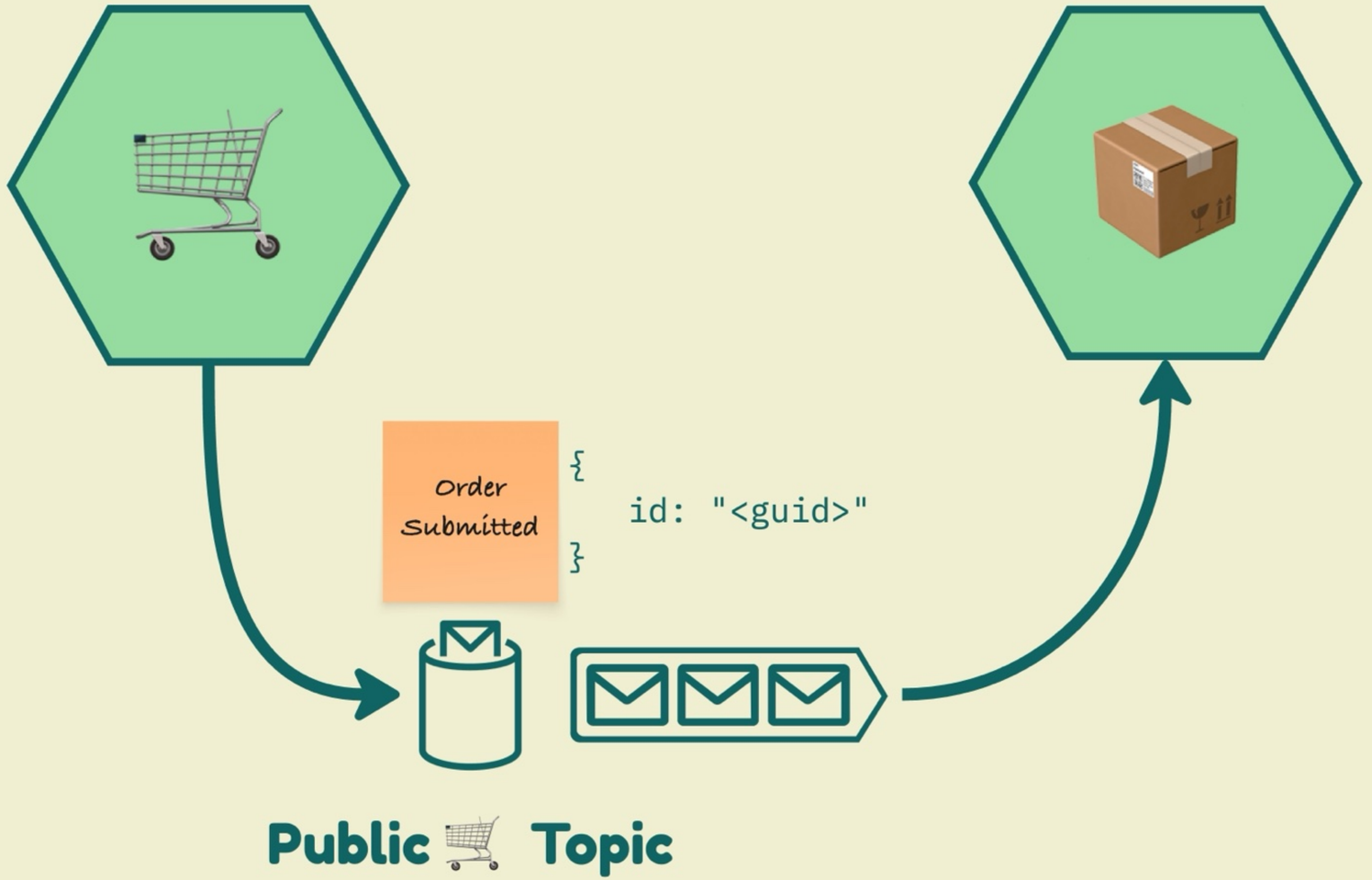
Event Storm - Bounded Contexts



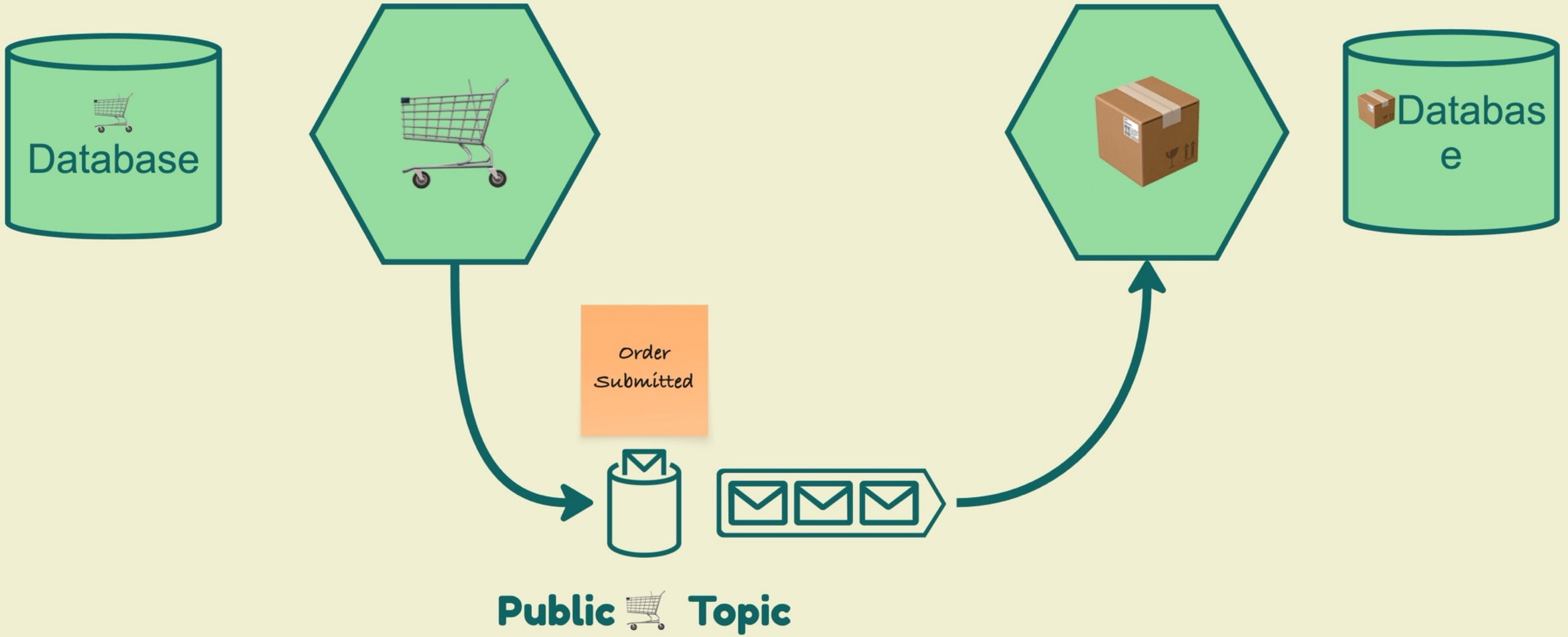
Event Storm - Bounded Contexts



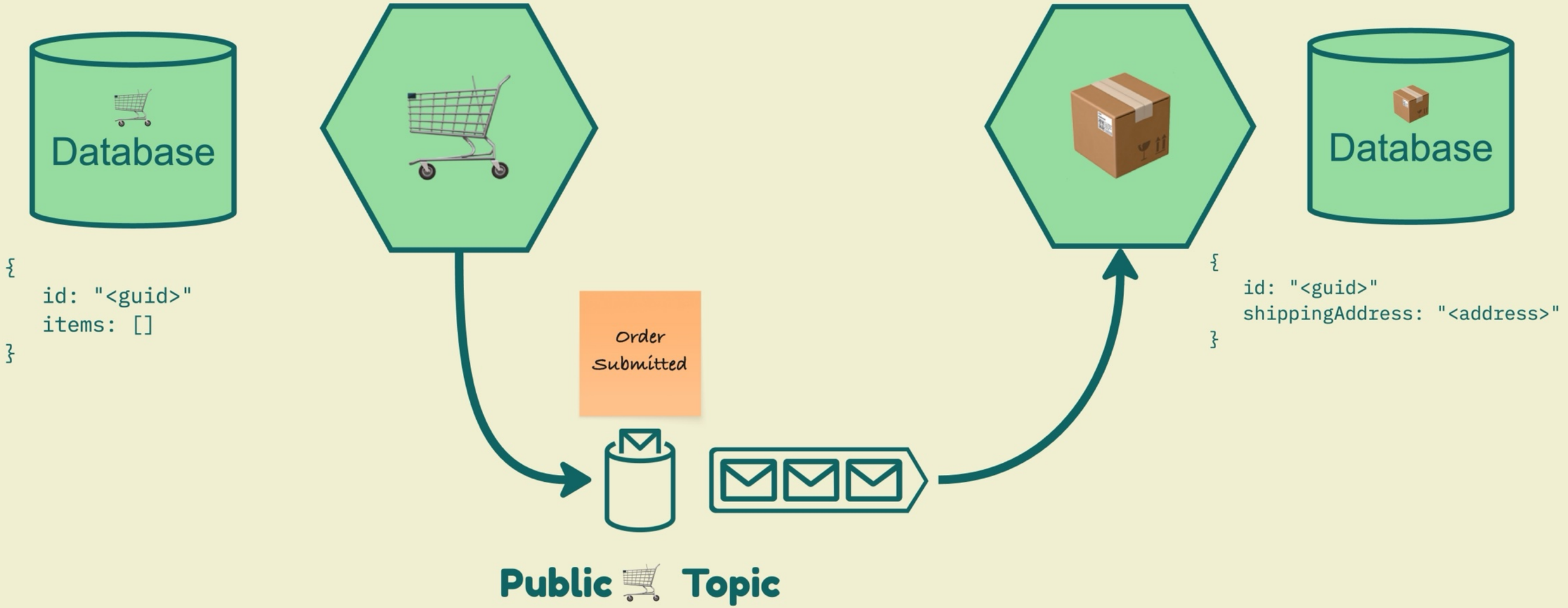
Clickbait Event



Clickbait Event



Clickbait Event



Public Domain Events



Purpose:

To use events to allow subscribing bounded contexts to react and execute next steps in larger scale Business Process Workflow(s)

```
{
    id: "<guid>"
}
```

Public Domain Events




```
{
    id: "<guid>"
}
```

EACH EVENT IS CRITICAL

Purpose:

To use events to allow subscribing bounded contexts to react and execute next steps in larger scale Business Process Workflow(s)

Public Domain Events

Each event is critical




```
{
    id: "<guid>"
}
```

Purpose:

To use events to allow subscribing bounded contexts to react and execute next steps in larger scale Business Process Workflow(s)

✅ **Messaging with Queue Based PubSub**



Public Domain Events



Purpose:

To use events to allow subscribing bounded contexts to react and execute next steps in larger scale Business Process Workflow(s)

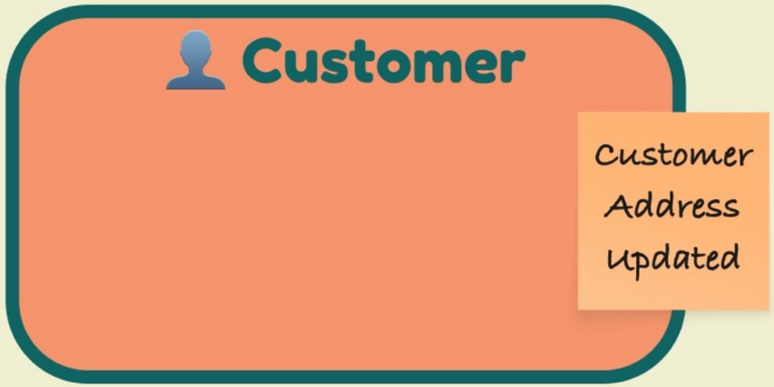
```
{
    id: "<guid>"
}
```

Messaging with Queue Based PubSub



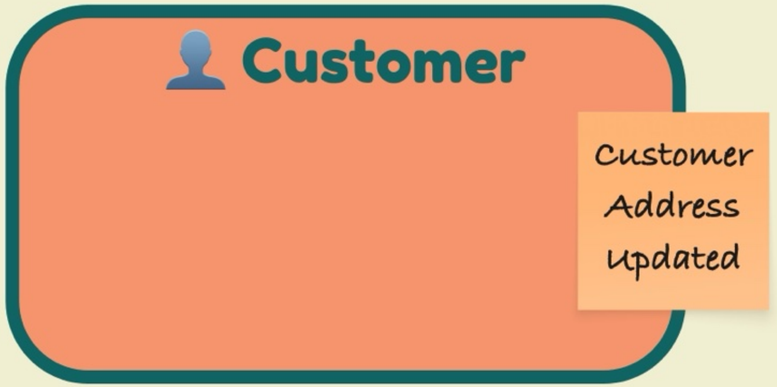
Producer and Consumer ARE different Bounded Contexts

Public Domain(?) Events




```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

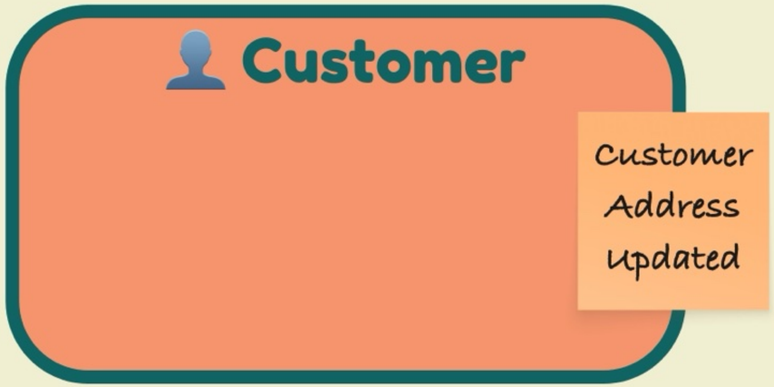
Public Domain(?) Events




```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

Large complex data structure - risk of change affecting subscribers

Public Domain(?) Events

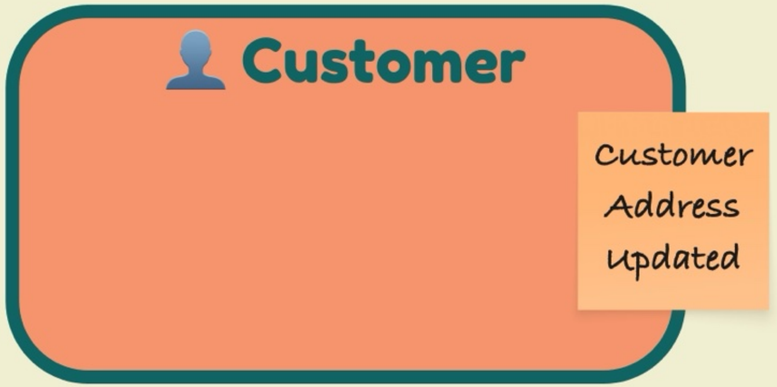


Purpose:

Event Carried State Transfer (ECST) data sharing

```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

Public Domain(?) Events



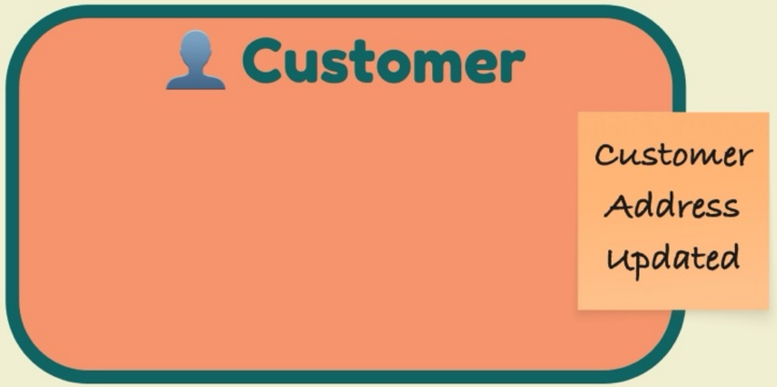
NOT A DOMAIN EVENT

Purpose:

Event Carried State Transfer (ECST) data sharing

```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

Public Integration Events



Purpose:

Event Carried State Transfer (ECST) data sharing

```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

Public Integration Events

Individual events don't matter – later events are more up to date

👤 **Customer**

Customer Address Updated

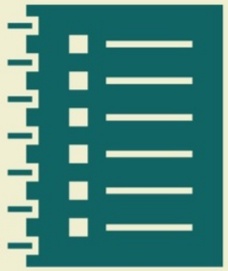
Purpose:

Event Carried State Transfer (ECST) data sharing

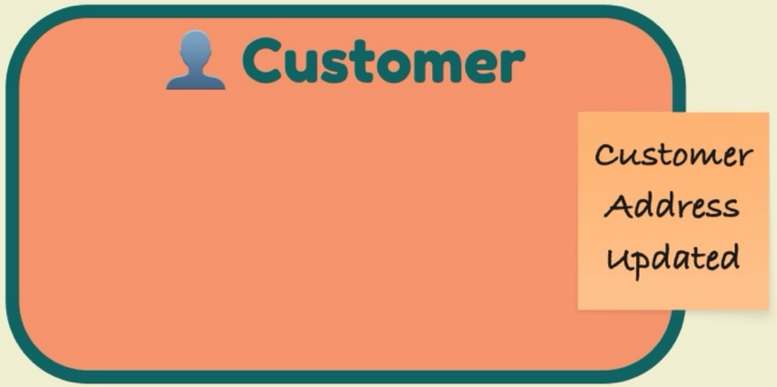
```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

✅

Streaming with a Distributed Log



Public Integration Events

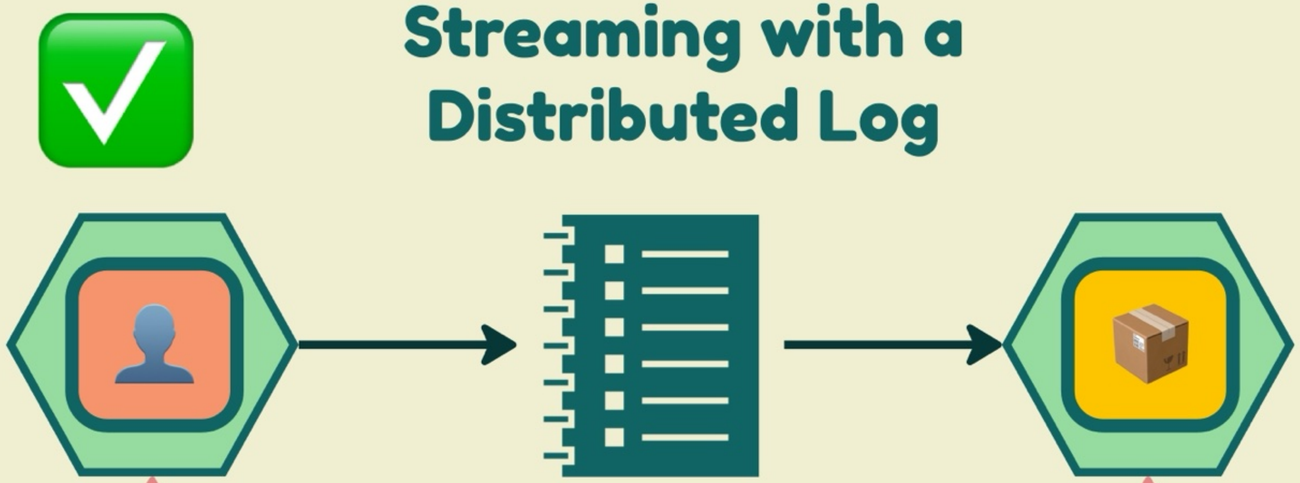


Purpose:

Event Carried State Transfer (ECST) data sharing

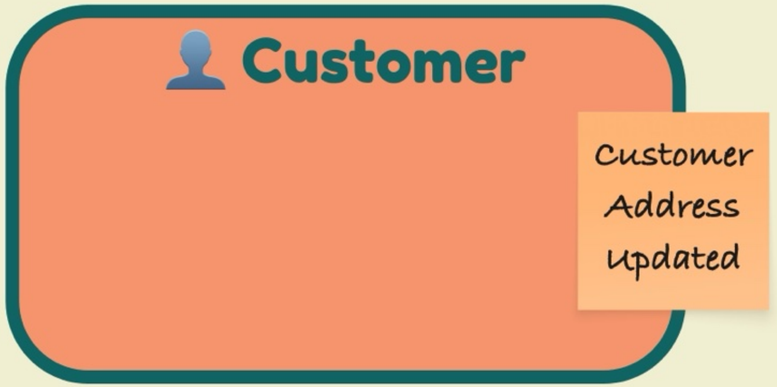
```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

Streaming with a Distributed Log



Producer and Consumer belong to the same Bounded Context

Public Integration Events

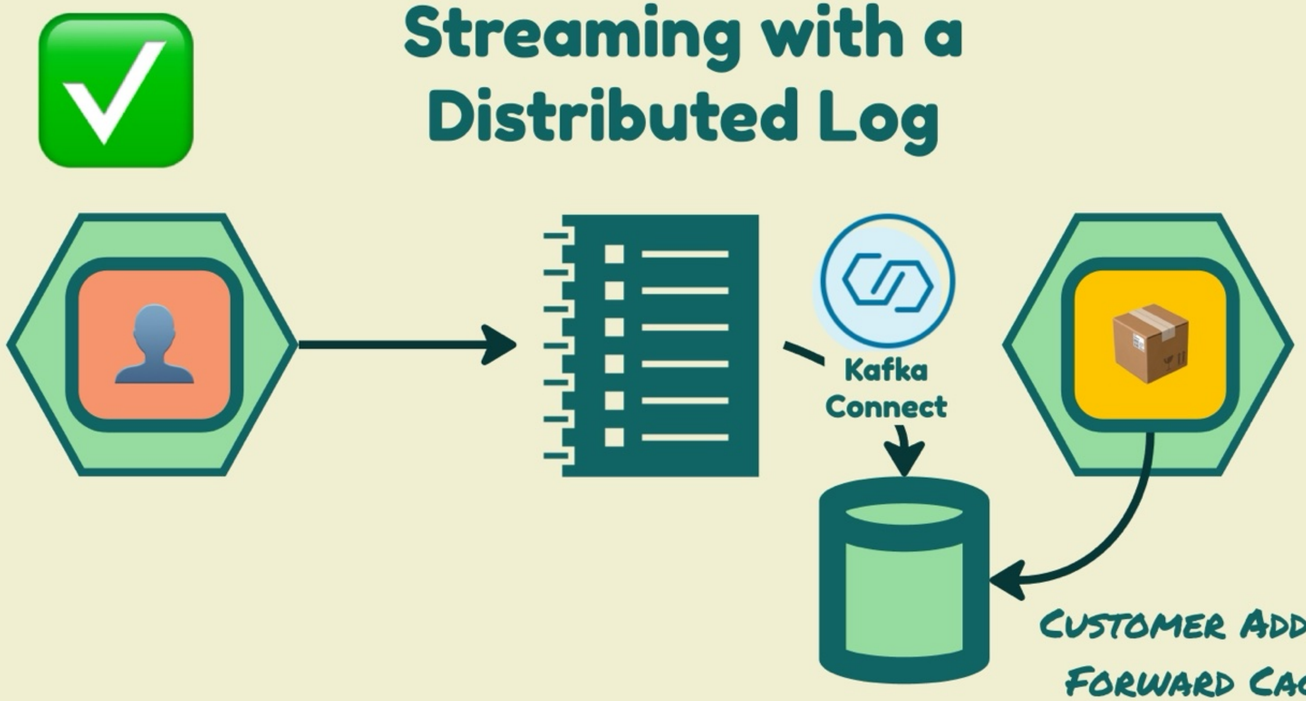


Purpose:

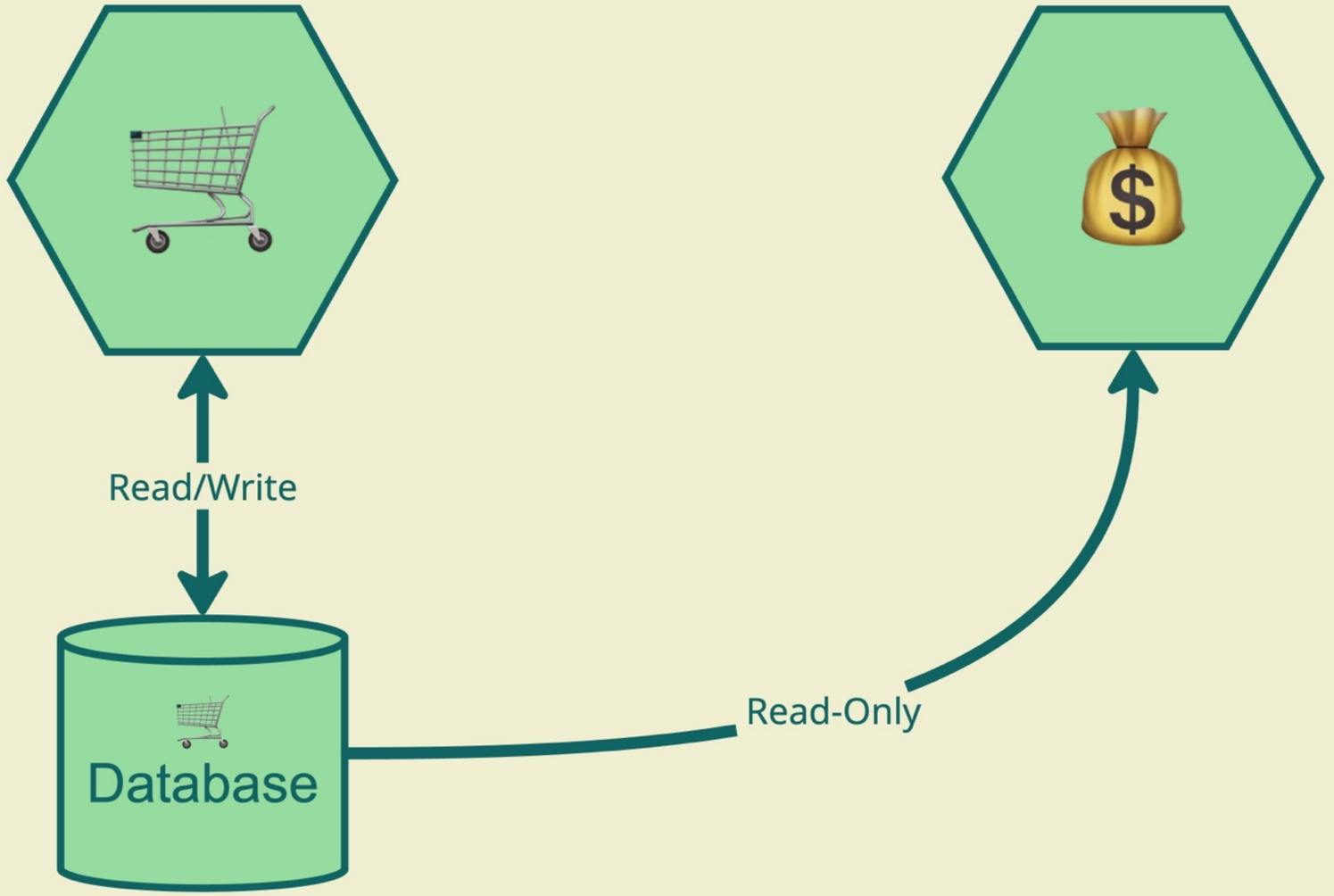
Event Carried State Transfer (ECST) data sharing

```
{
   id: "<guid>",
   address: {
     number: "123",
     street: "Some",
     type: "Road",
     ...
}
```

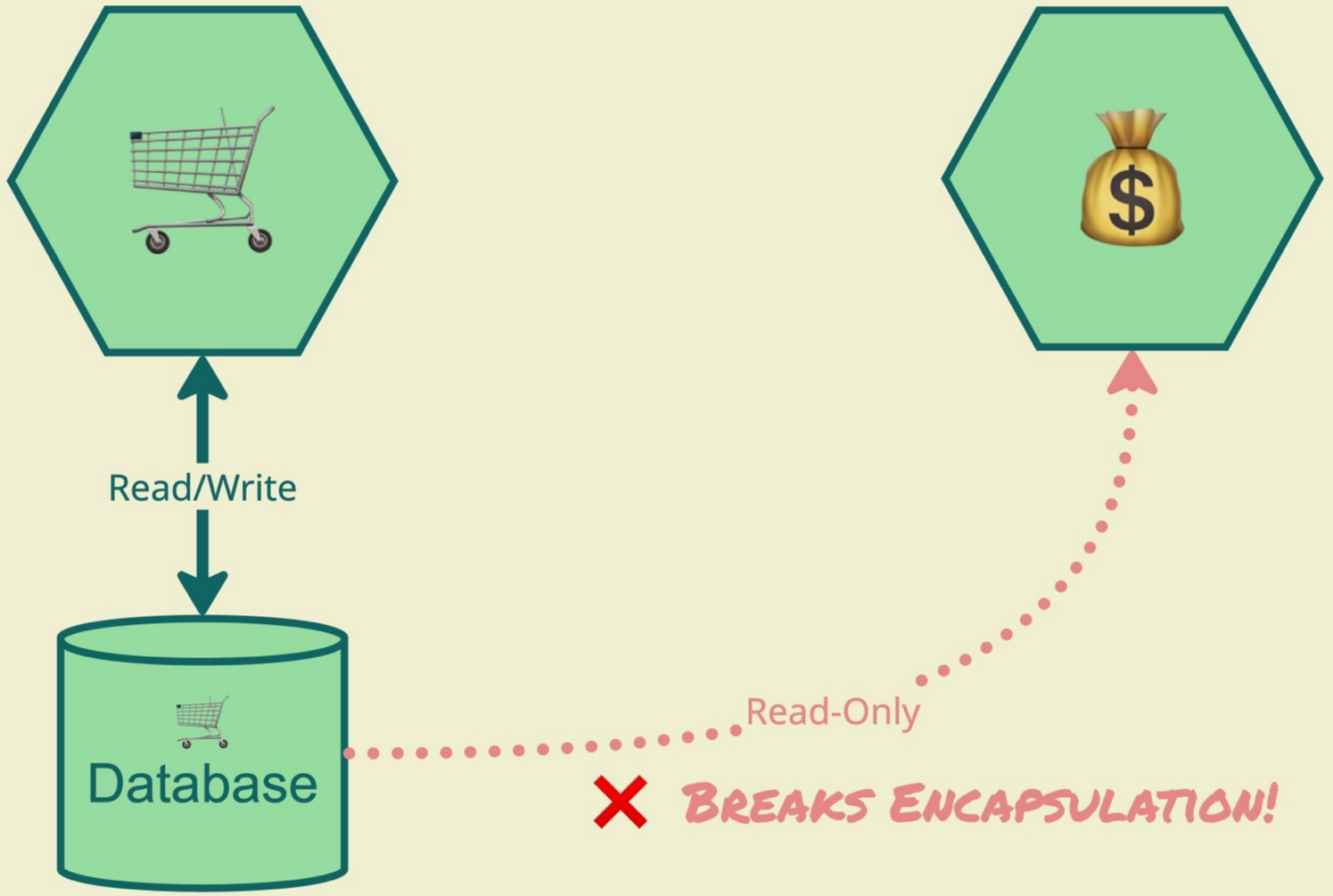
Streaming with a Distributed Log



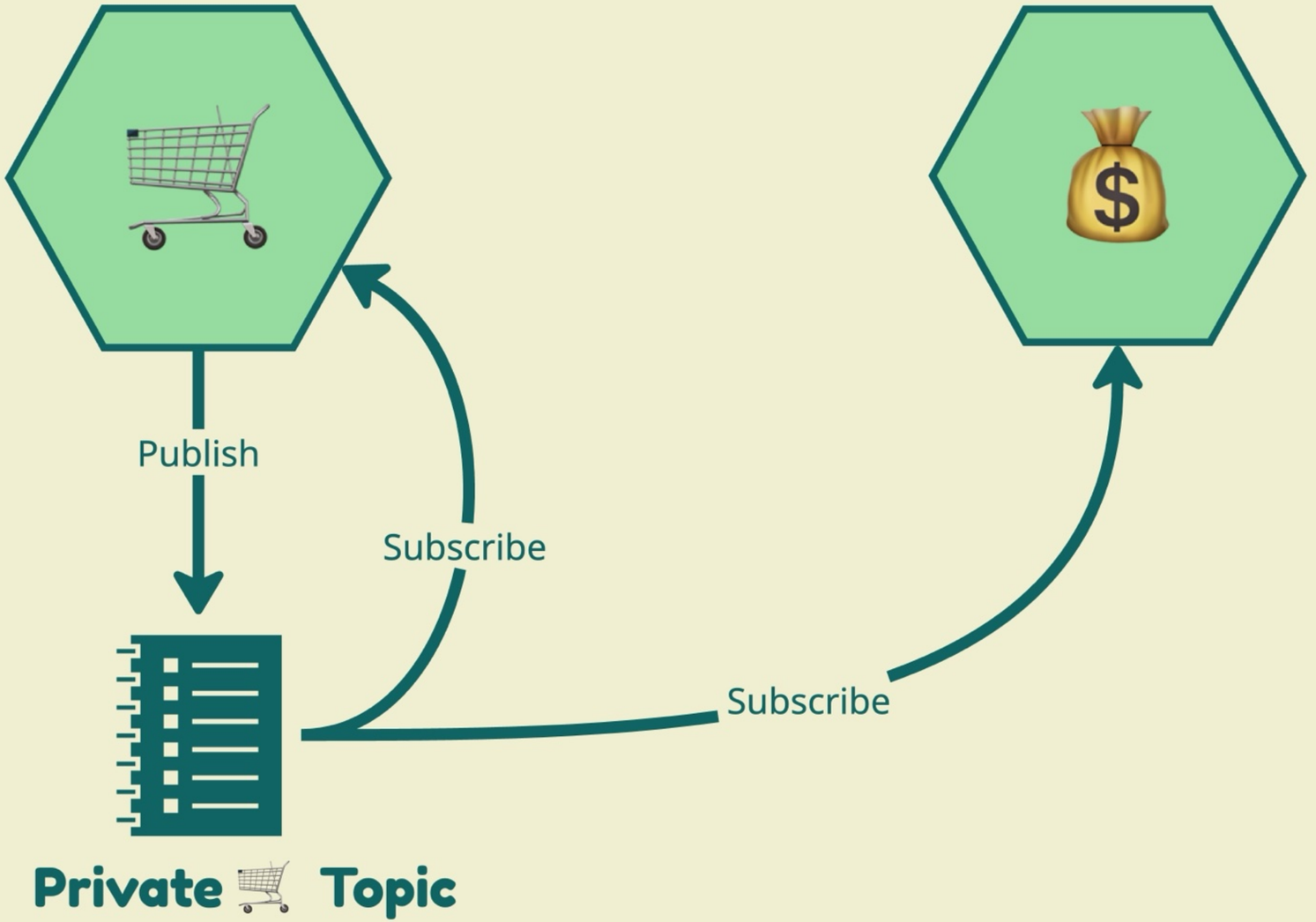
Encapsulation



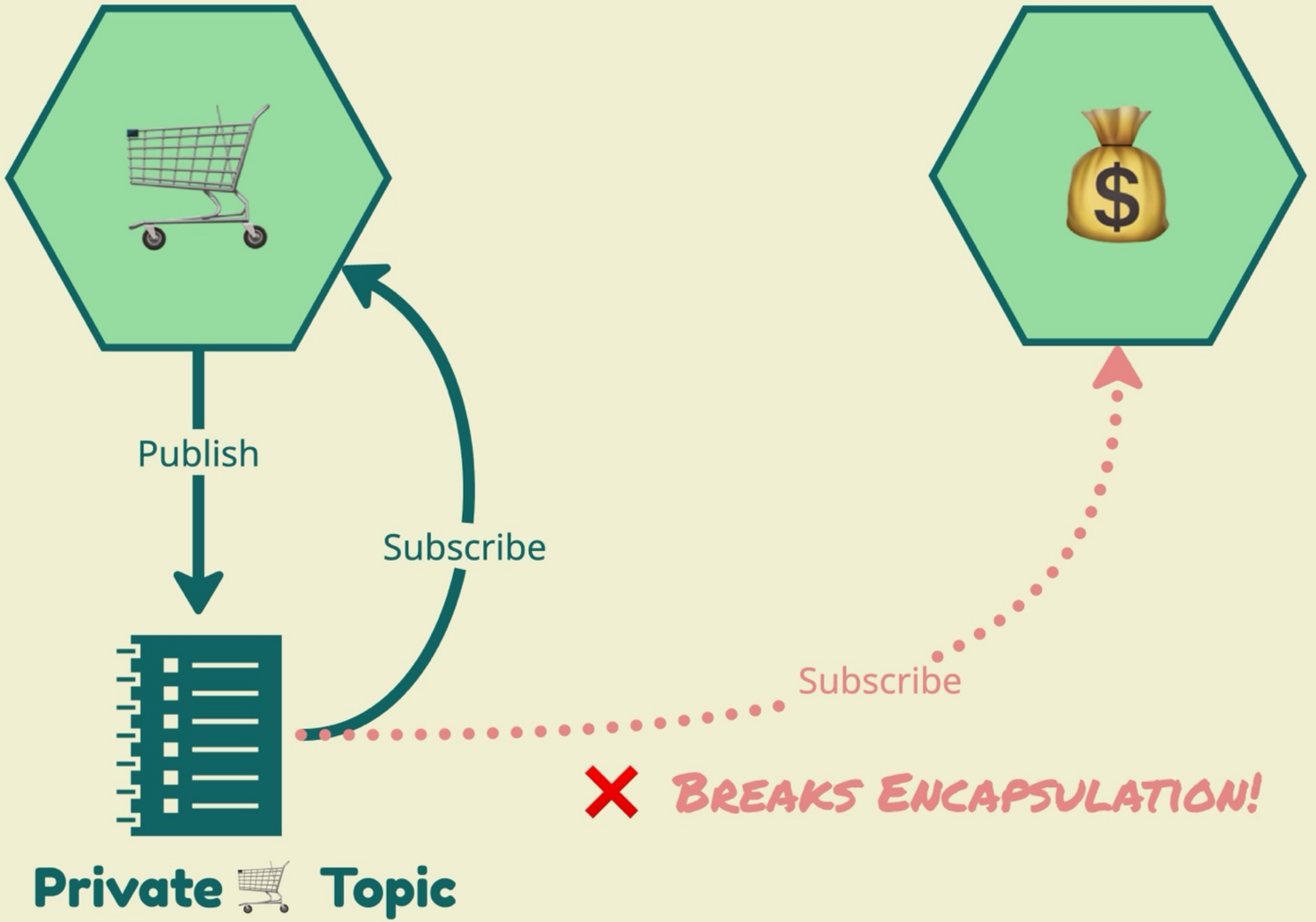
Encapsulation



Encapsulation

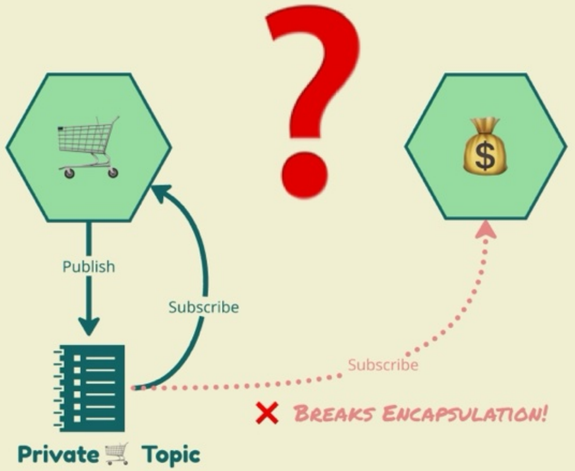


Encapsulation

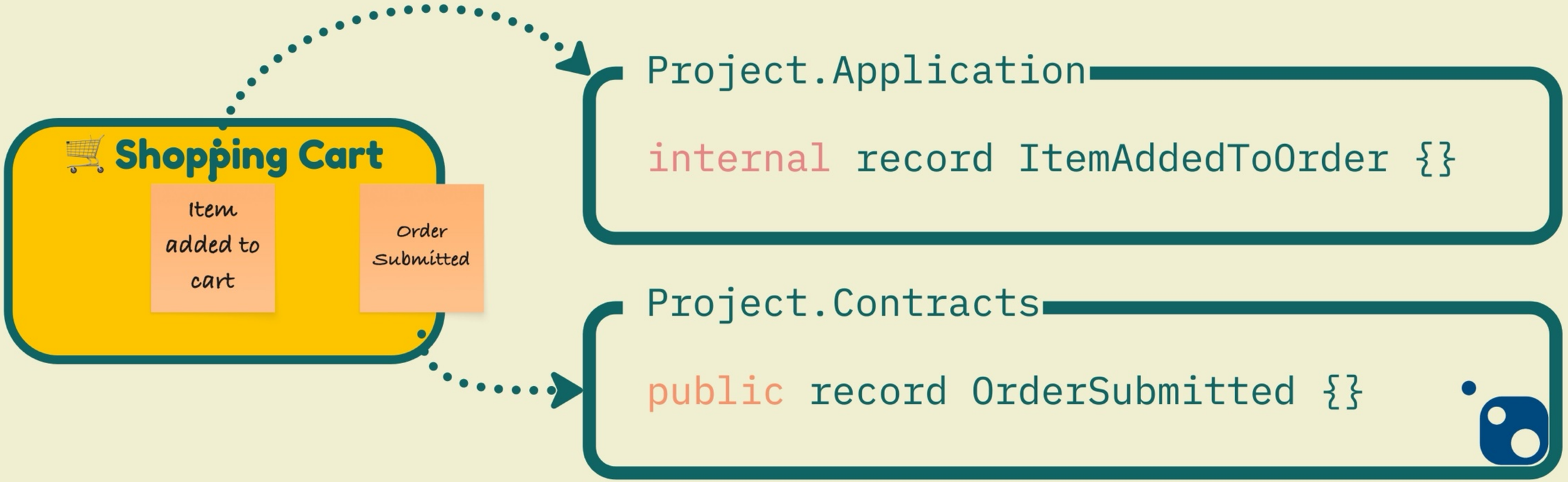


Keeping track of public vs private

How do you ensure private events don't 'leak'/break encapsulation when the topics co-exist with public events in the same broker?

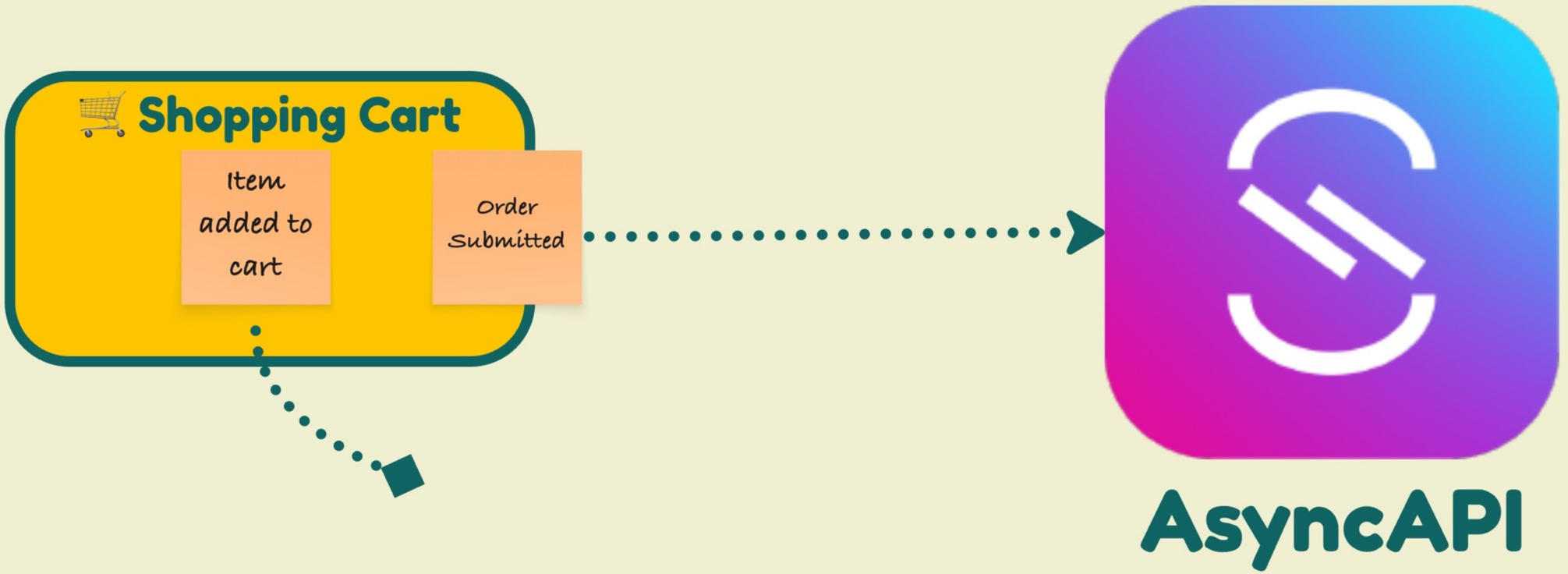


Keeping track of public vs private



Use your coding language features

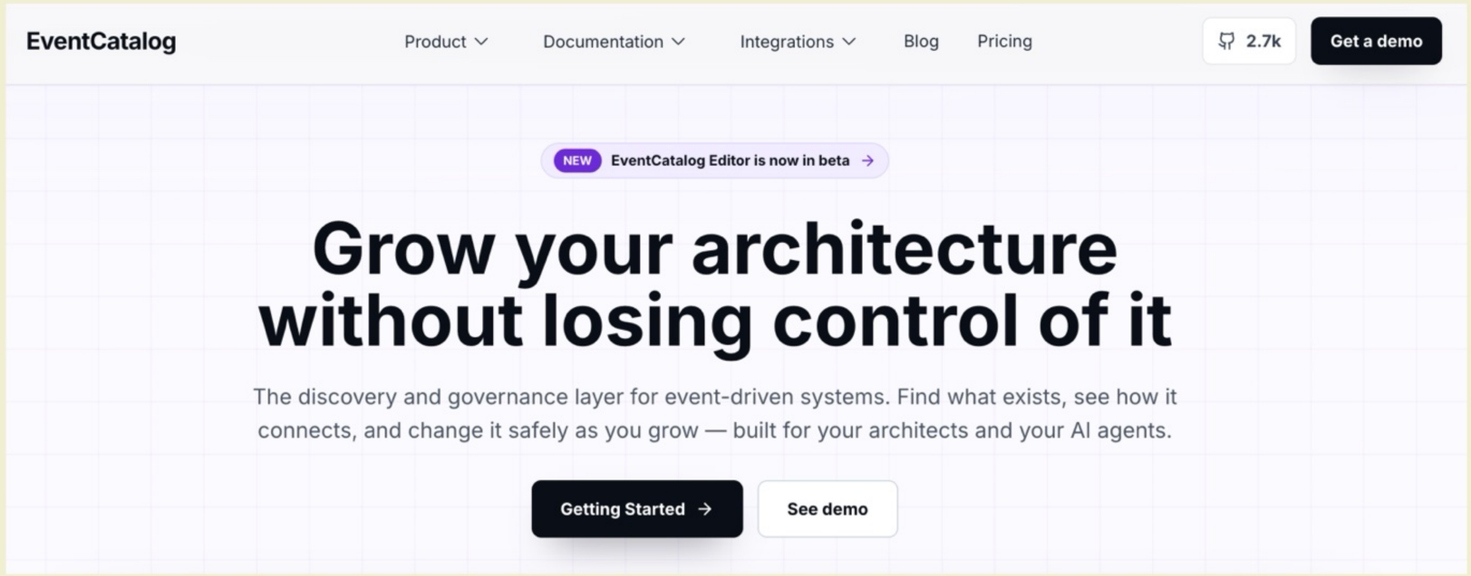
Keeping track of public vs private



Only include public events in your public schema documentation

Keeping track of public vs private





Public + Private Indicators Coming Soon!

Heuristic Summary

	State	Business Process		
Private				
Public				

Heuristic Summary

	State	Business Process		
Private	 Event Store			
Public	⛔			

Heuristic Summary

	State	Business Process		
Private	 Event Store	 Queue-based PubSub		
Public		 Queue-based PubSub		

Heuristic Summary

	State	Business Process	Streaming Analytics	
Private	 Event Store	 Queue-based PubSub	 Distributed Log	
Public		 Queue-based PubSub	 Distributed Log	

Heuristic Summary

	State	Business Process	Streaming Analytics	Forward Caching
Private	 Event Store	 Queue-based PubSub	 Distributed Log	
Public		 Queue-based PubSub	 Distributed Log	 Distributed Log


Alanis Morissette - Ironic (1996)
♪ It's the good advice
that you just didn't take ♪
chrissimon.au

Code Examples



Next Session: Oct 6 – 9, 2026

Imagine the Future and Build for the Now

Advanced | Remote - Online | English

Book your seat



YOUR INSTRUCTORS



Chris Simon

DDDACADEMY Upcoming Catalog Blog DDD Europe Conferences About us FAQ Contact

Next Session: Oct 6 – 9, 2026

Imagine th Future and for the No

Advanced Remote - Online

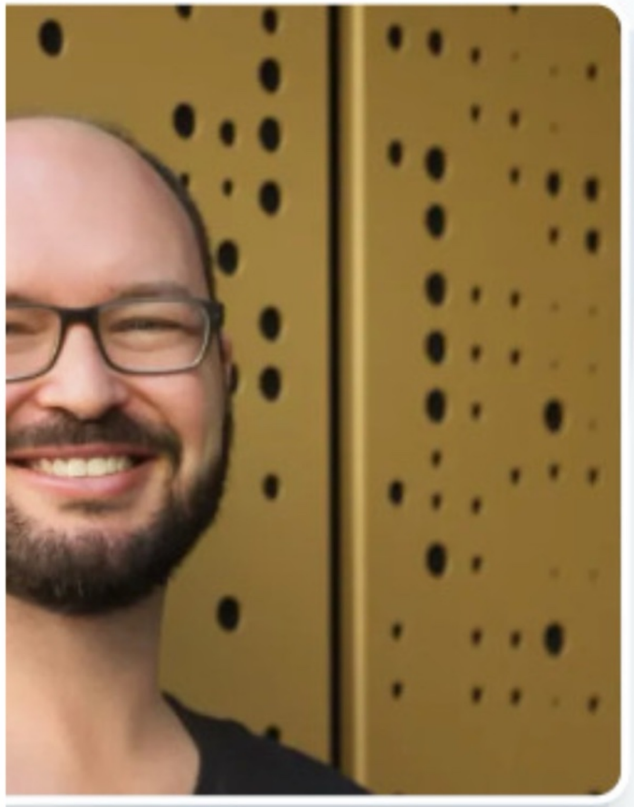
Book your seat

YOUR INSTRUCTORS

Chris Simon

Curriculum

Day 1	Day 2
Start - 09:00	
Getting started, Intro to Systems, Logical View	Inter-process Communications
Morning Tea - 10:30-10:45	
Logical View (cont)	Inter-process Communications, Physical View
Lunch - 12:00-13:00	
Logical View (cont.	Developer View (pt1), Scenario View (pt1)
Afternoon Tea - 14:30-14:45	
Process View	Developer View (pt2), Scenario View (pt2)
Finish - 5pm	



DDDACADEMY Upcoming Catalog Blog DDD Europe Conferences About us FAQ Contact

Next Session: Oct 6 – 9, 2026

Imagine th
Future and
for the No

Advanced Remote - Online

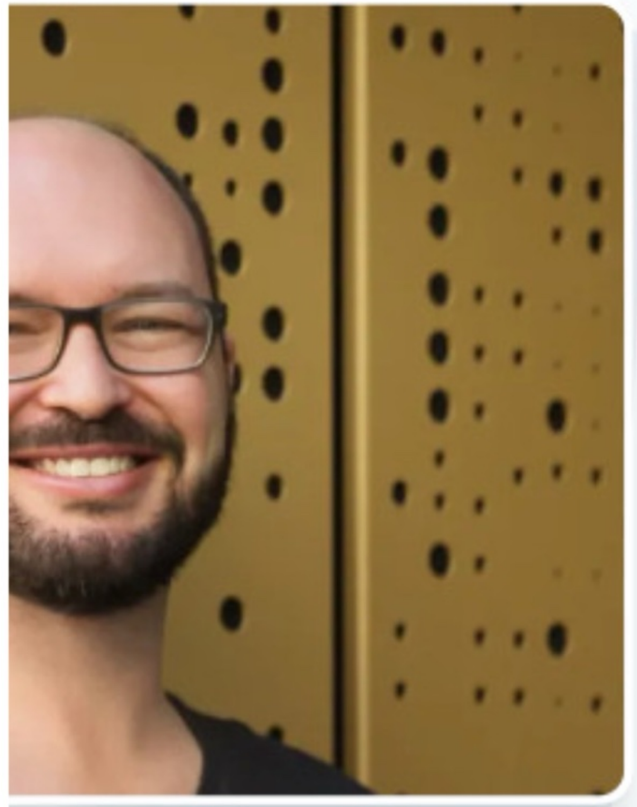
Book your seat

YOUR INSTRUCTORS

Chris Simon

Curriculum

Day 1	Day 2
Start - 09:00	
Getting started, Intro to Systems, Logical View	Inter-process Communications
Morning Tea - 10:30-10:45	
Logical View (cont)	Inter-process Communications, Physical View
Lunch - 12:00-13:00	
Logical View (cont.	Developer View (pt1), Scenario View (pt1)
Afternoon Tea - 14:30-14:45	
Process View	Developer View (pt2), Scenario View (pt2)
Finish - 5pm	



Let's Connect

chrissimon.au

chrissimon-au

@chrissimon.au

@ChrisSimonAu@hachyderm.io

