Event driven architectures

Joran Bergfeld
Solution Architect

Fancy a ?

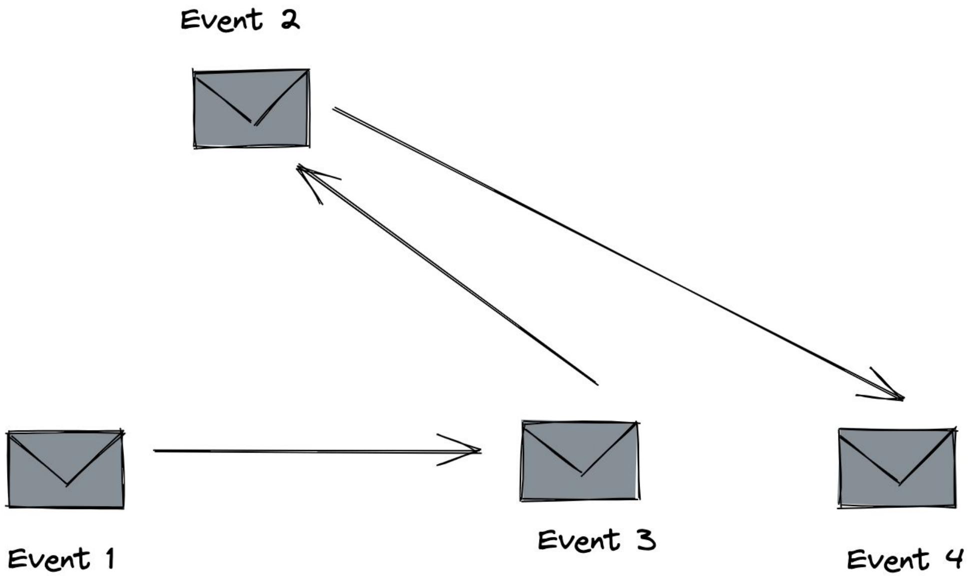
Connect with me on

Worked with computers since teenage years
From malware, to network, to software
A few months at Red Hat
Cooking, cocktails, general nerd and explorer
Just came back from Bali

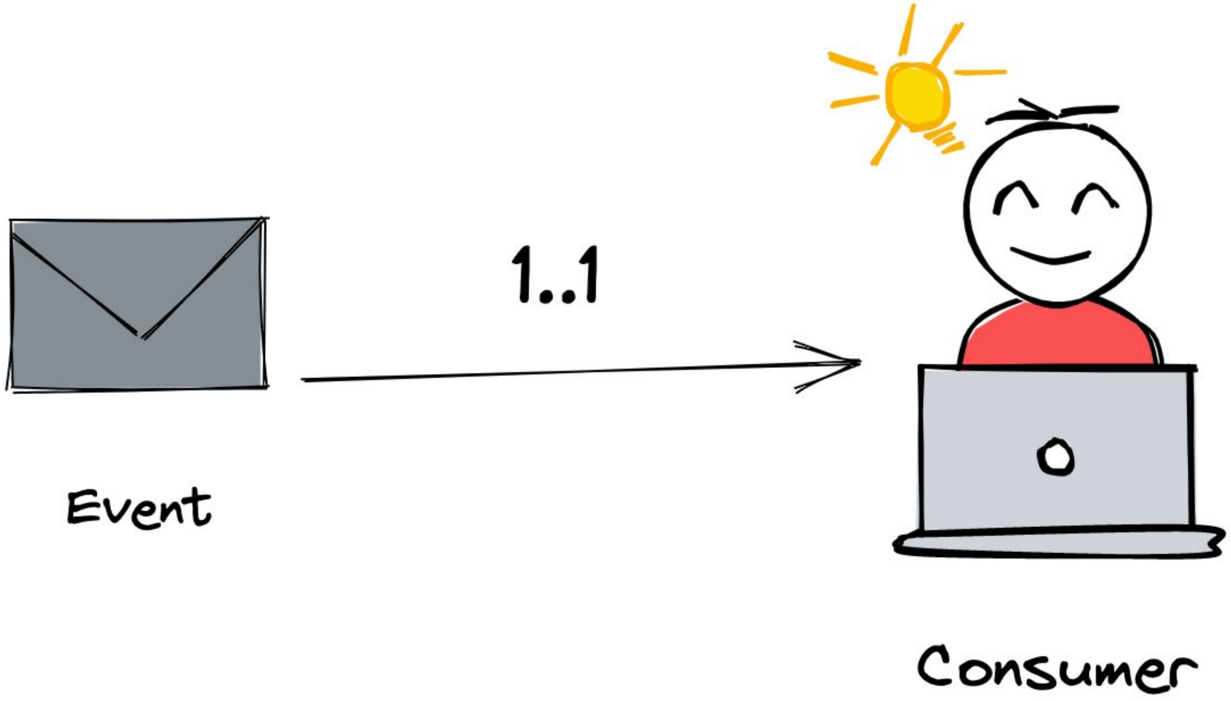
LET’S TALK COOKING

LET'S TALK EVENT SYSTEMS

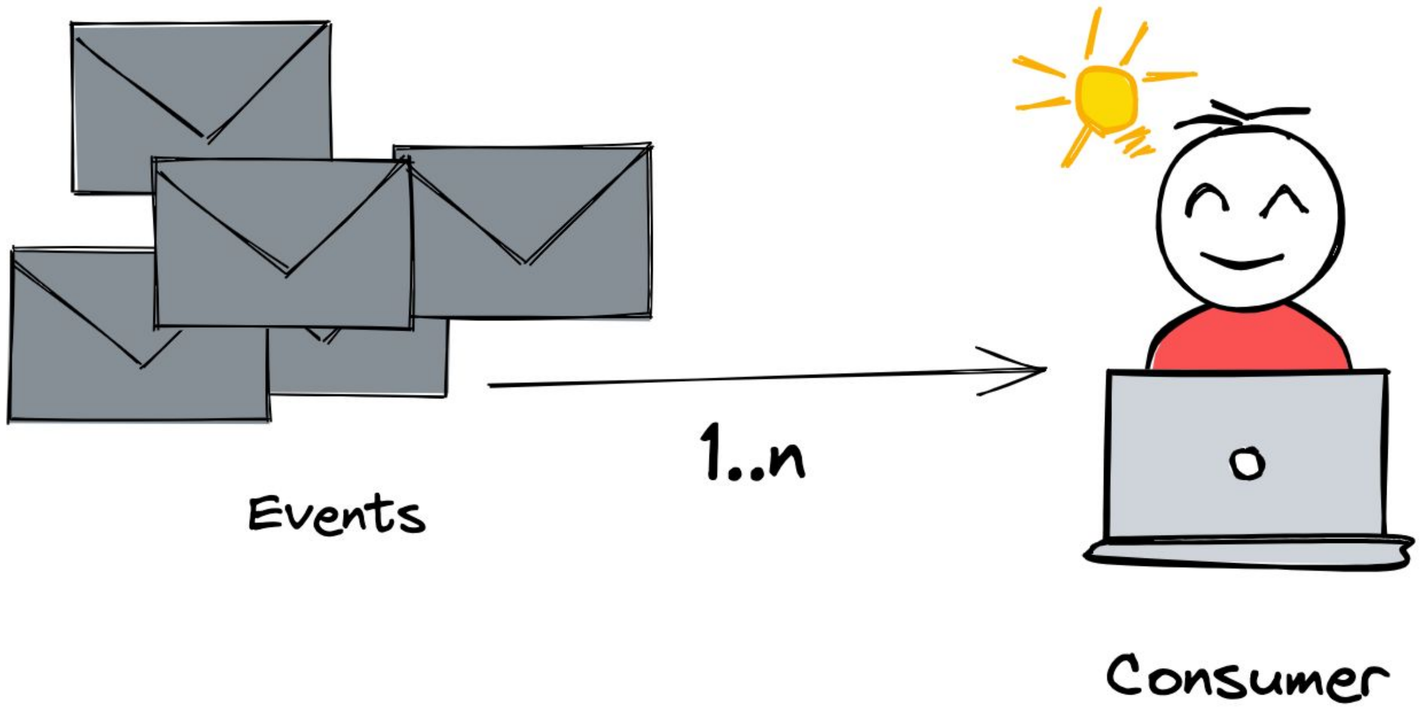
Message Ordering


Event 2
Event 1
Event 3
Event 4

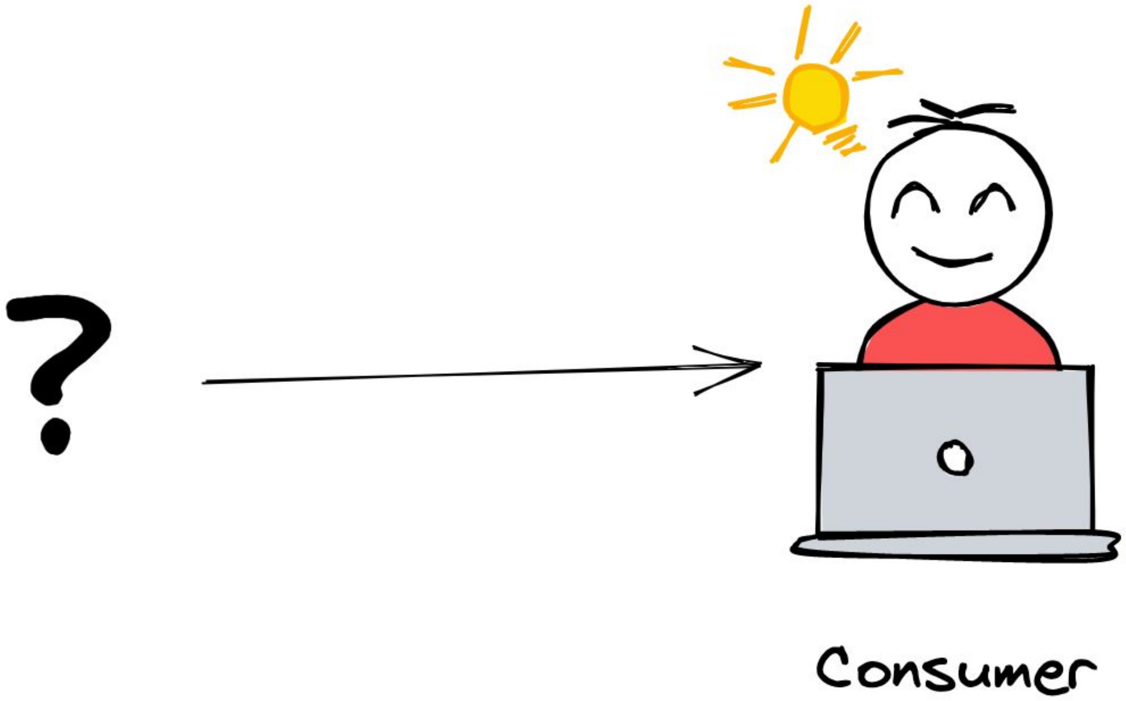
Exactly once delivery



Atleast once delivery



At most once delivery

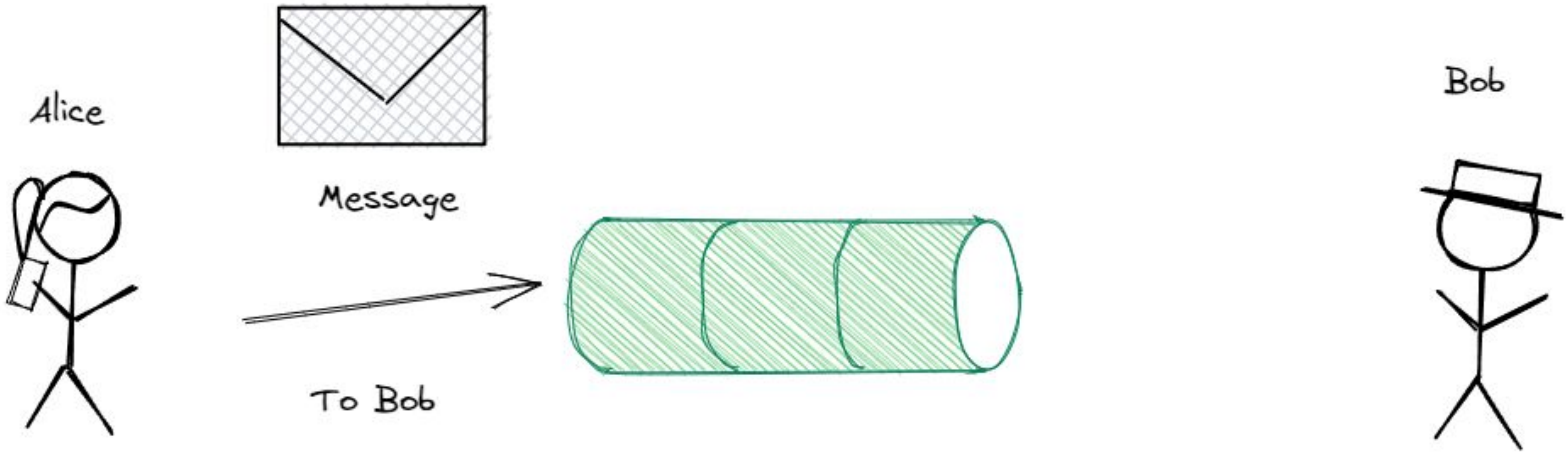


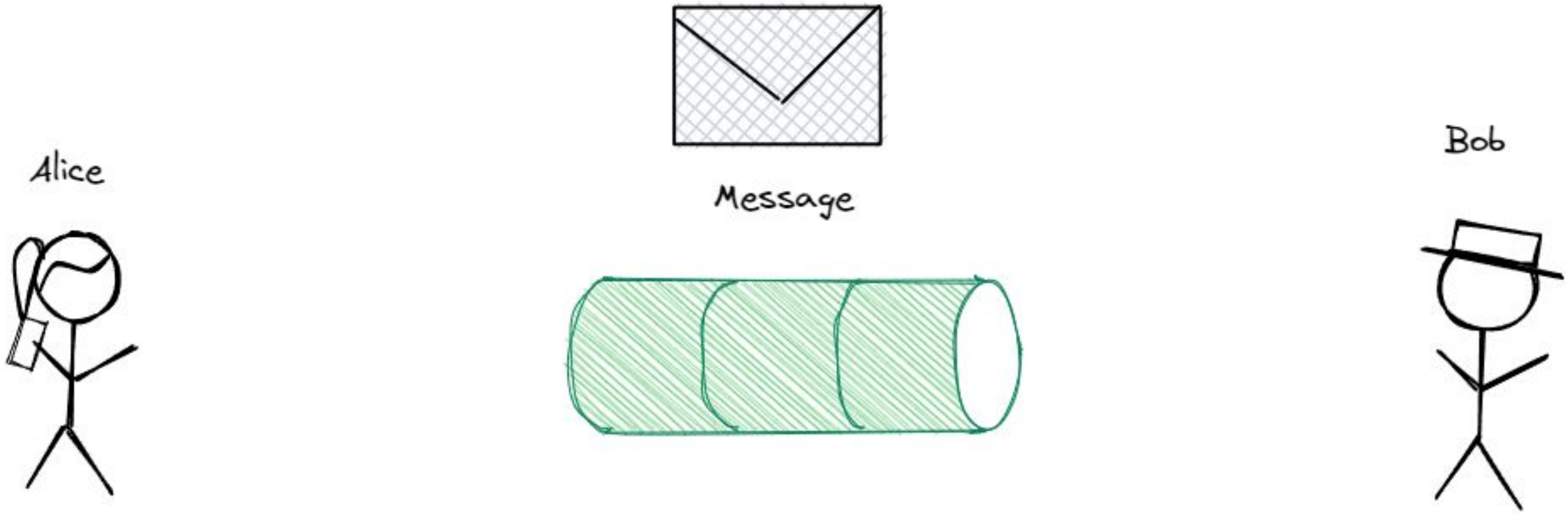
TODAY IS NOT ABOUT YOU
IT'S ABOUT ALL OF YOU

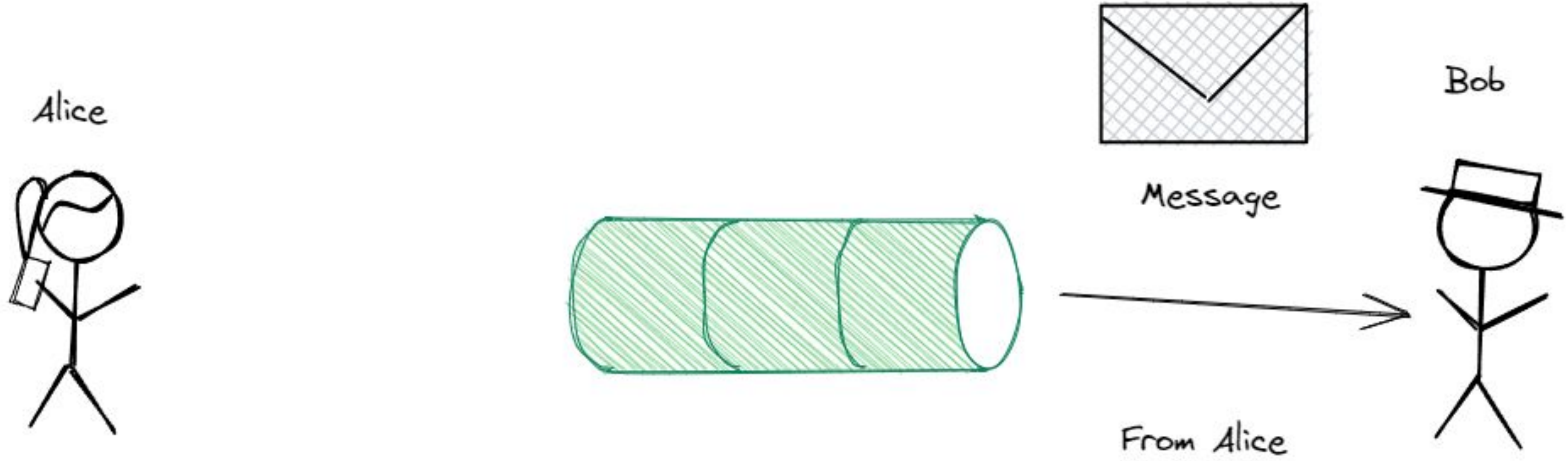
MESSAGES AND EVENTS

MESSAGE

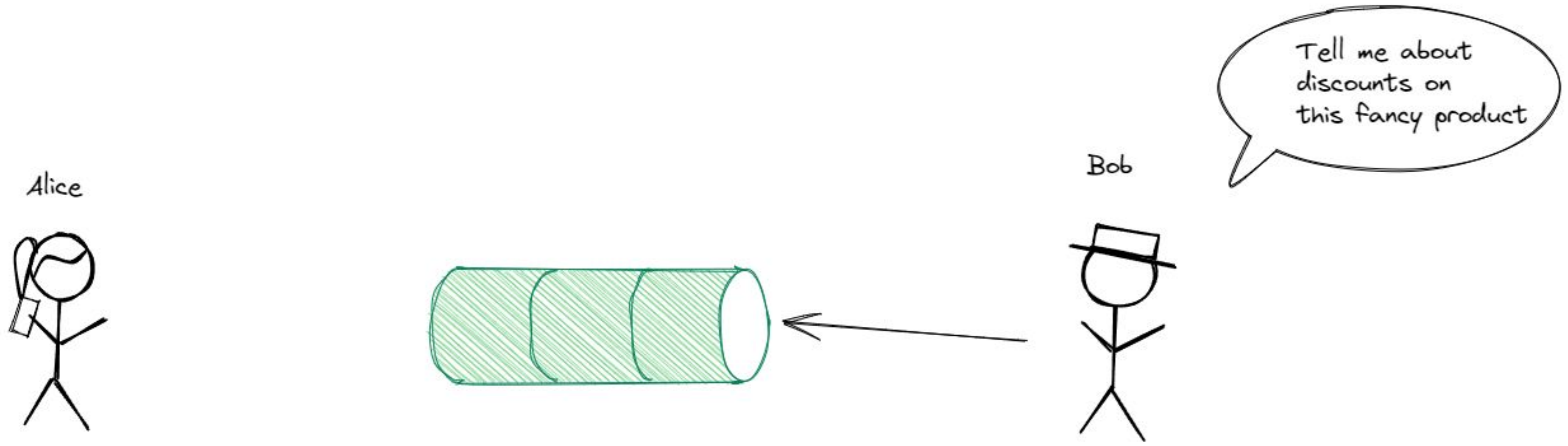
Alice
Bob
Message

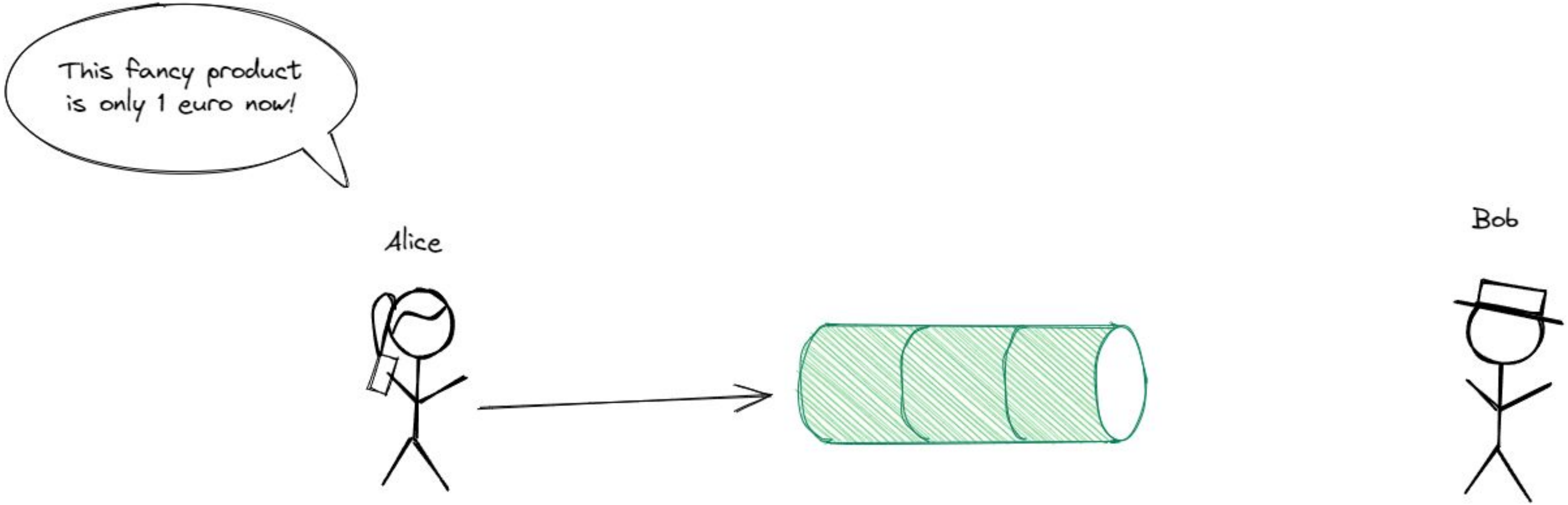

Alice
Message
To Bob
Bob

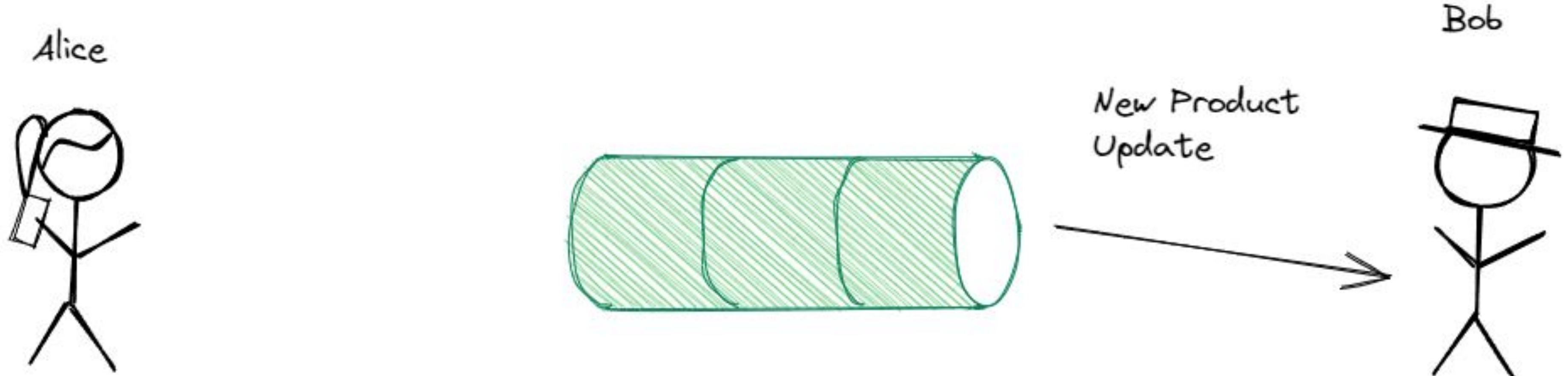

Alice
Message
Bob


Alice
Message
Bob
From Alice

EVENT


Alice
Bob
Tell me about discounts on this fancy product


This fancy product is only 1 euro now!
Alice
Bob


Alice
Bob
New Product
Update

RECAP

Message

- You receive a message (command)
- As producer, you care about the consumer
- Exactly one consumer
- Business flows are usually more tightly coupled

- Result of the above: Centralized technology

Event

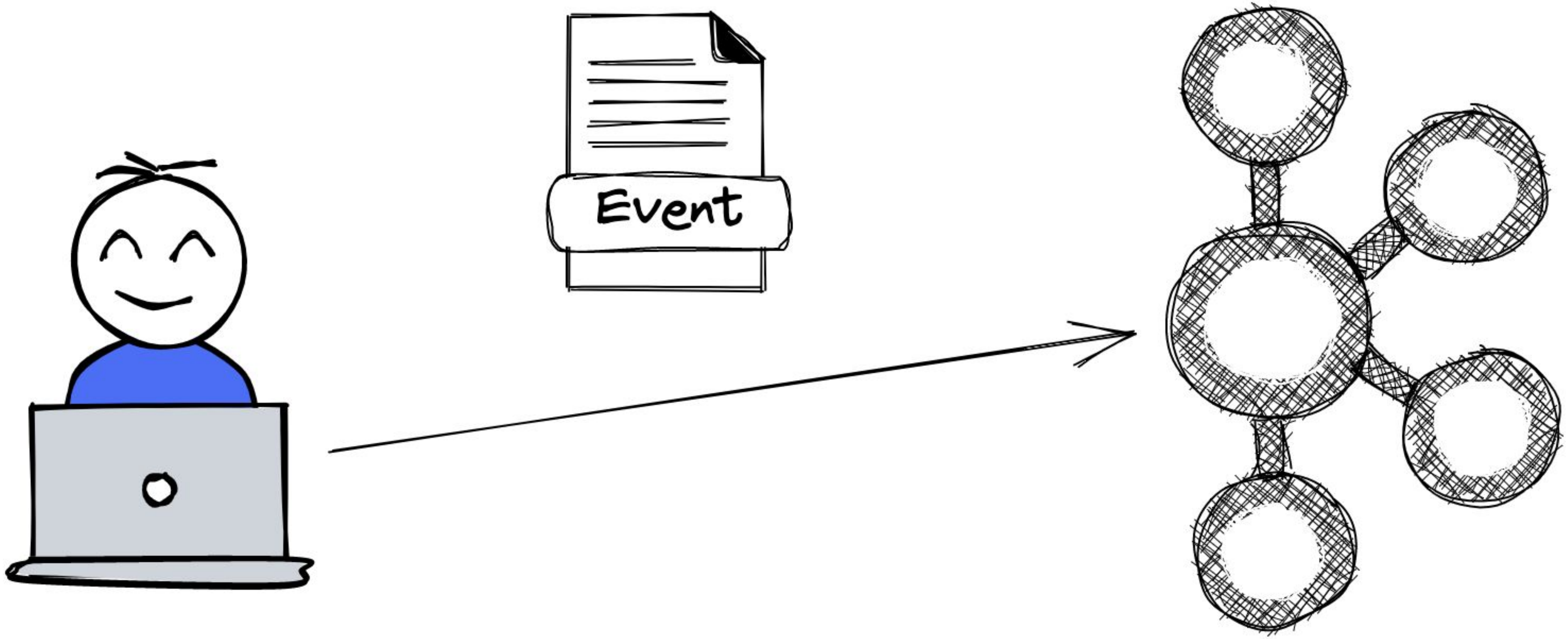
- You react to an event
- As producer, you don’t really care about the consumer
- Zero to many consumers
- Business flows are usually less tightly coupled

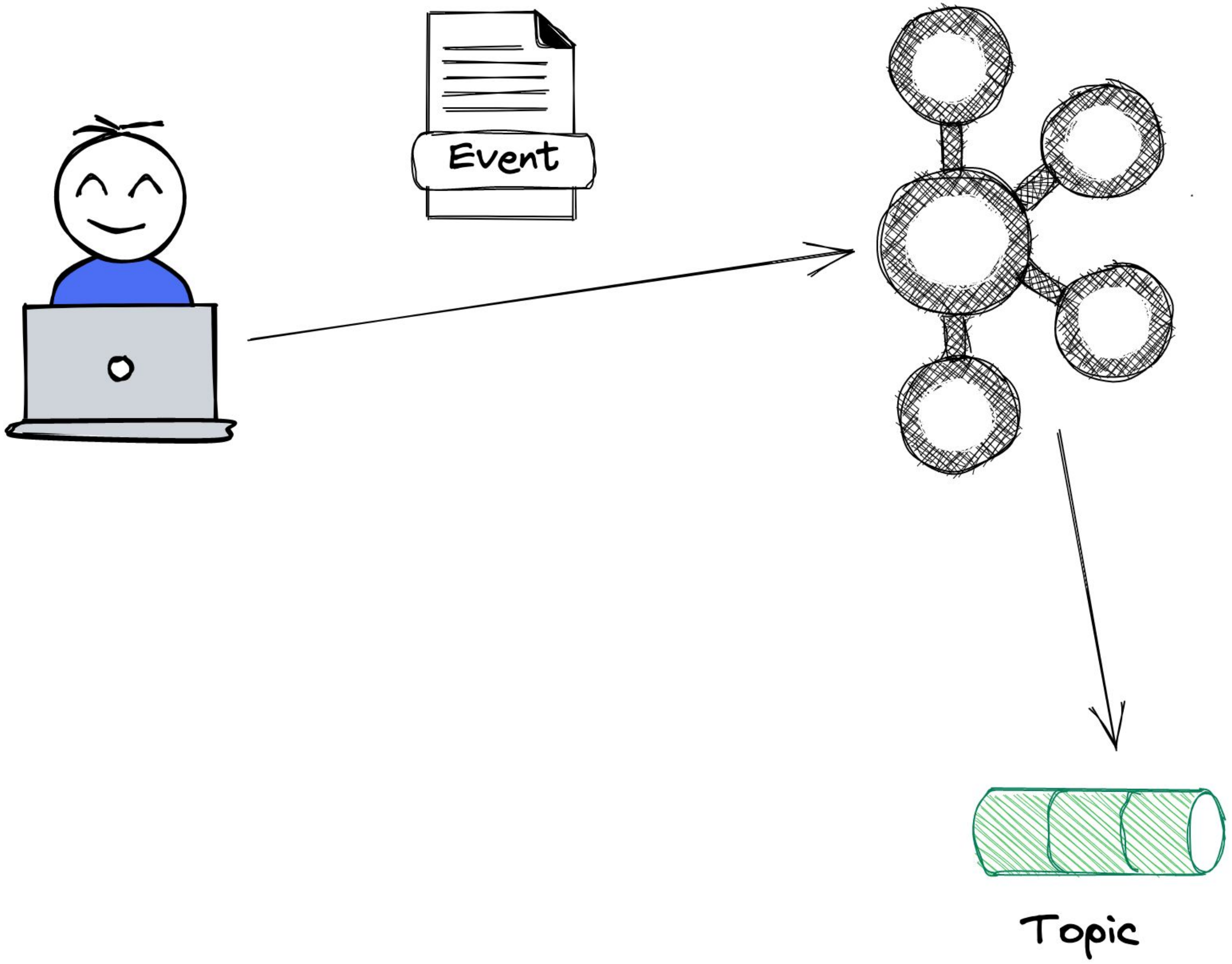
- Result of the above: Decentralized technology

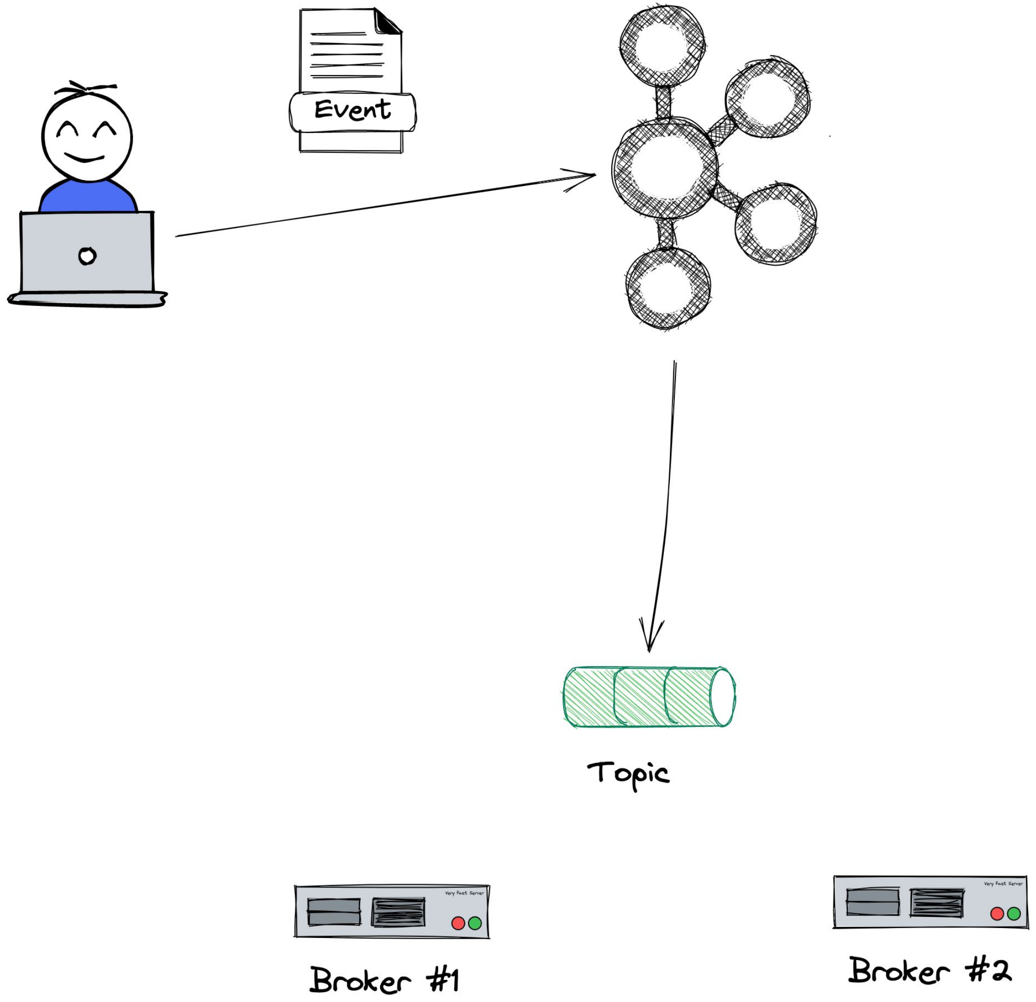
WHAT IS A KAFKA

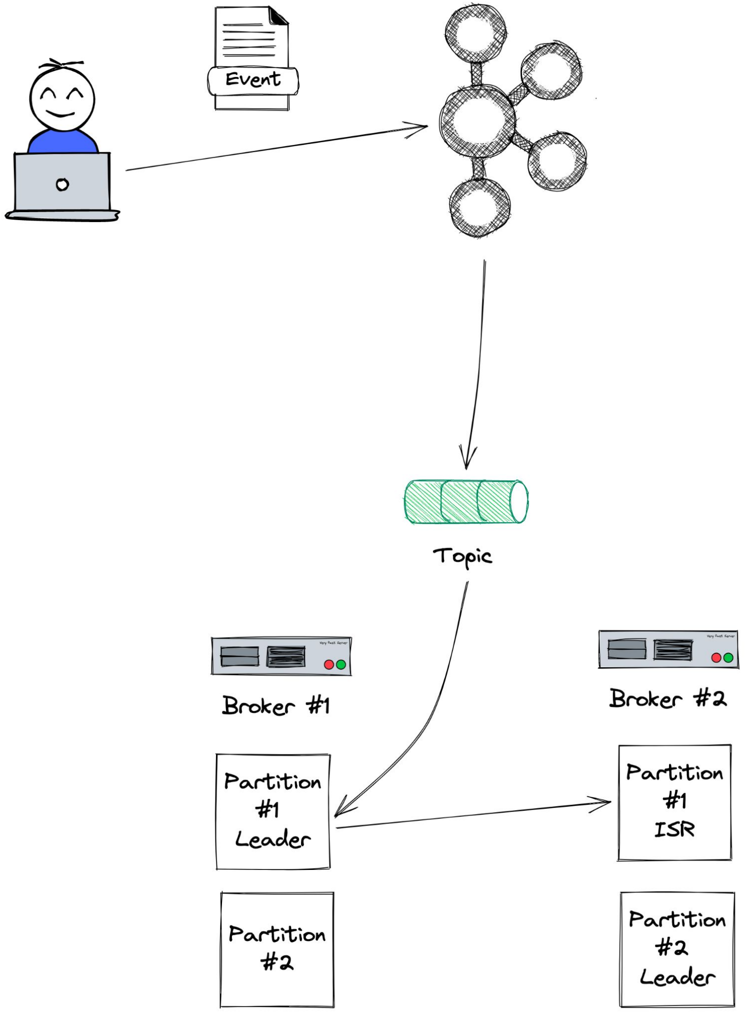
Kafka is the answer to everything
CHANGE MY MIND
imgflip.com

PRODUCER PERSPECTIVE


Event

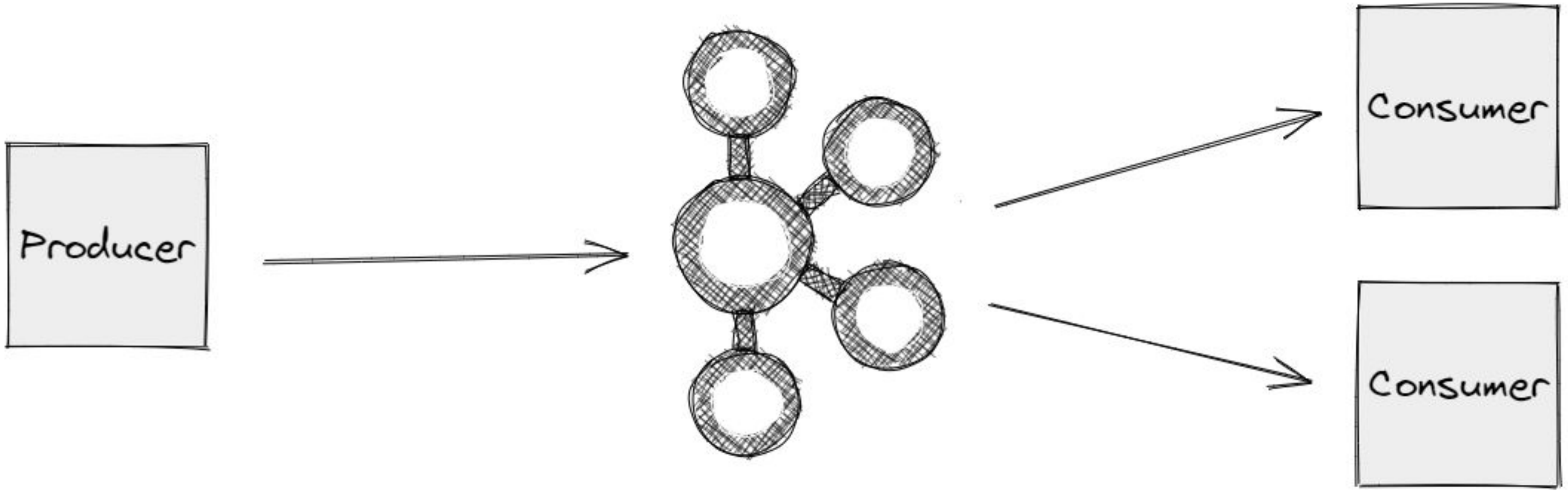

Event
Topic

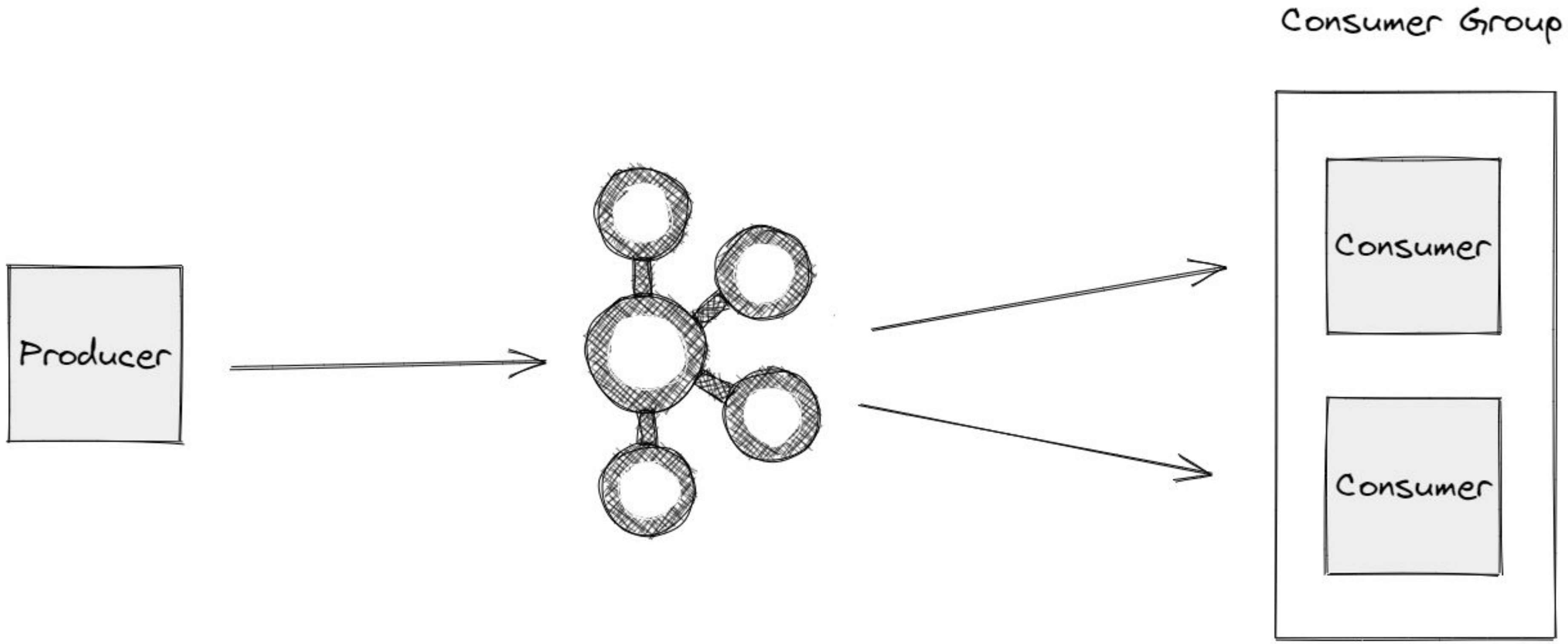

Event
Topic
Very Fast Server
Broker #1
Very Fast Server
Broker #2

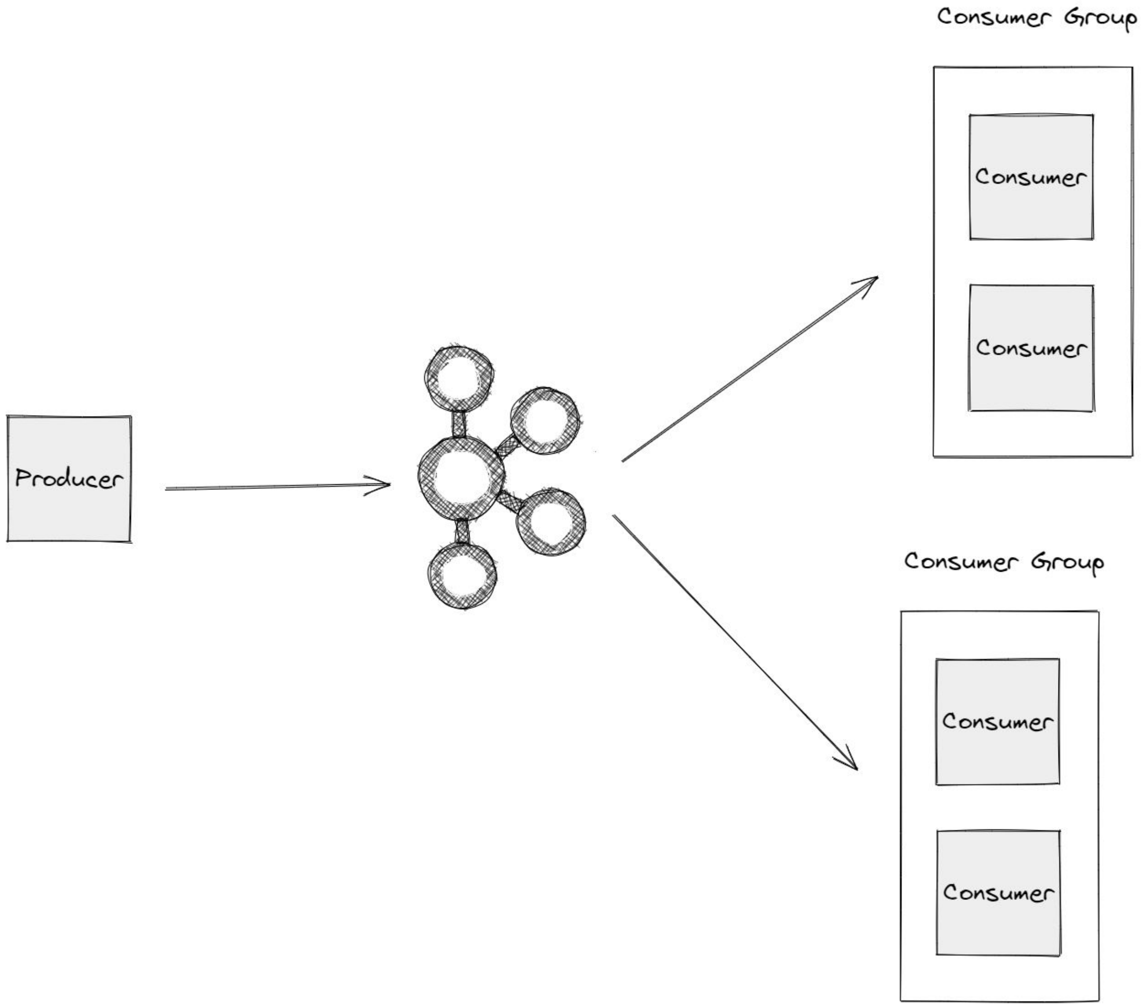

Event
Topic
Broker #1
Broker #2
Partition #1 Leader
Partition #1 ISR
Partition #2
Partition #2 Leader

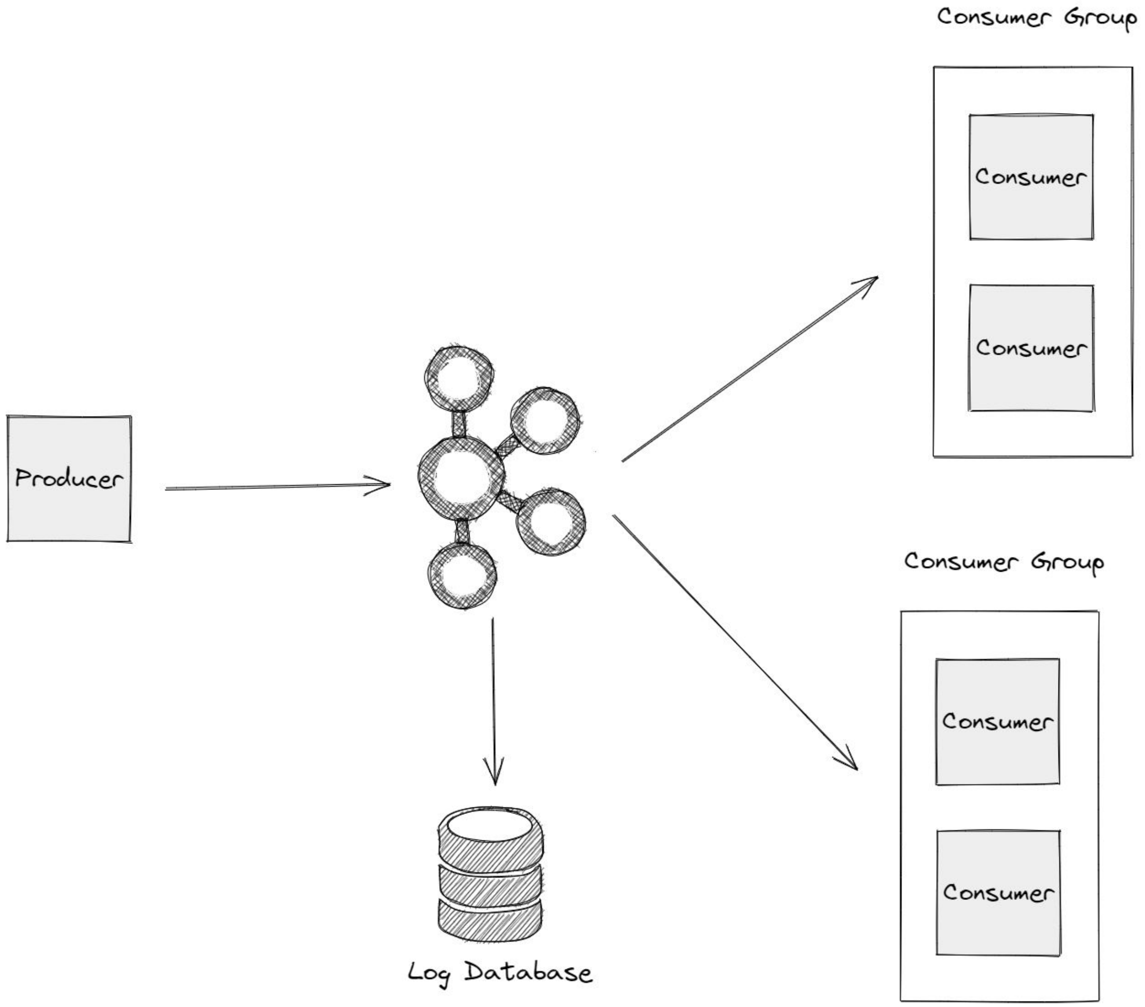
IN SYNC REPLICA?

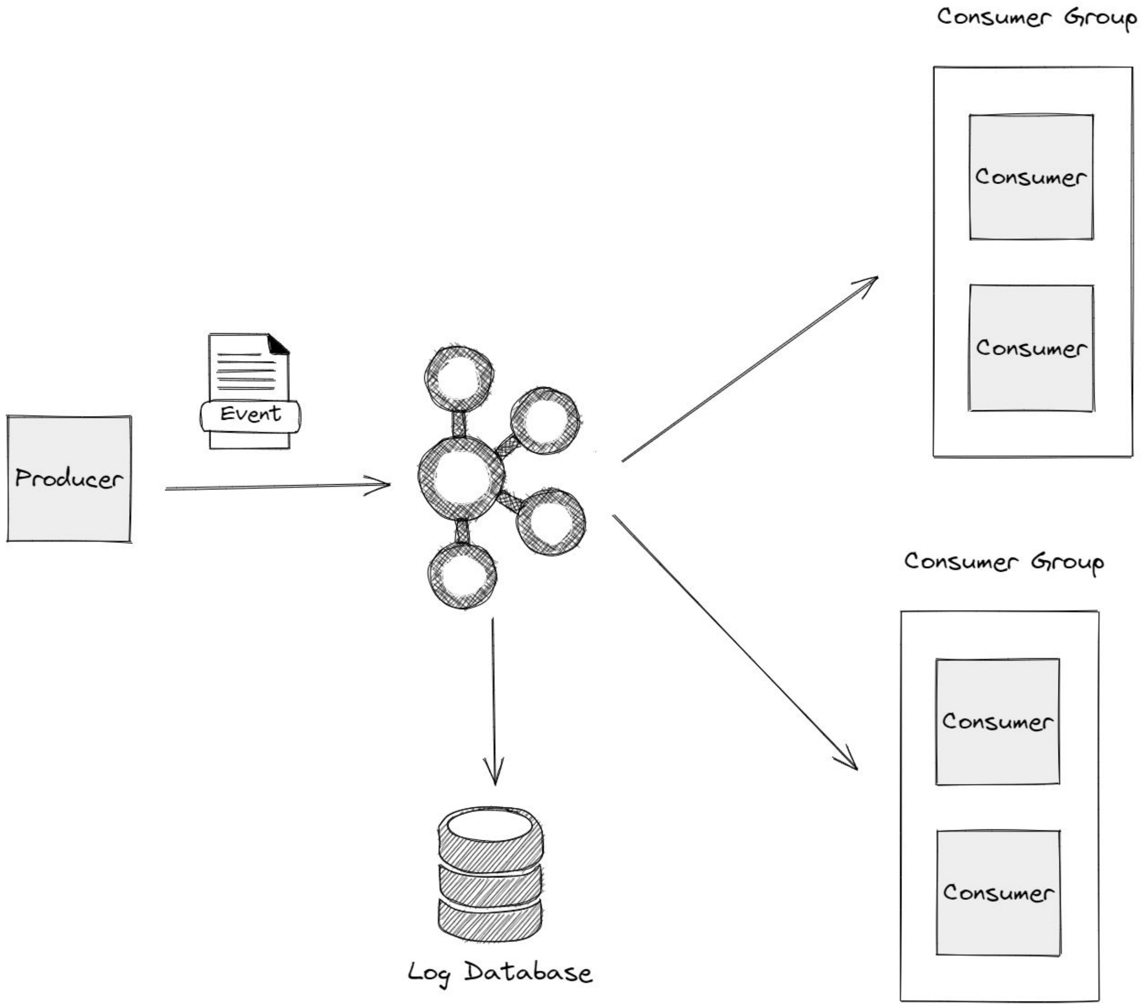
CONSUMER PERSPECTIVE

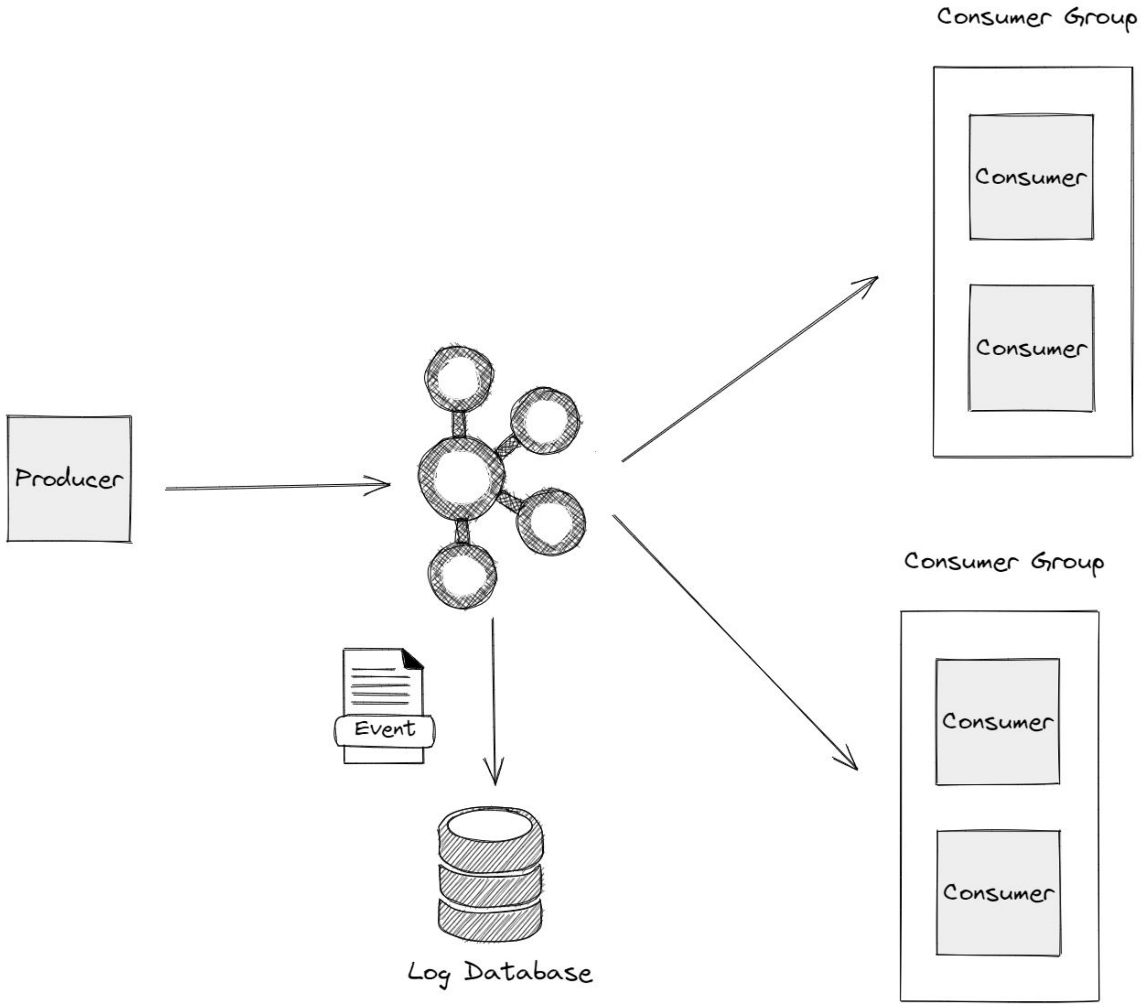

Producer
Consumer
Consumer

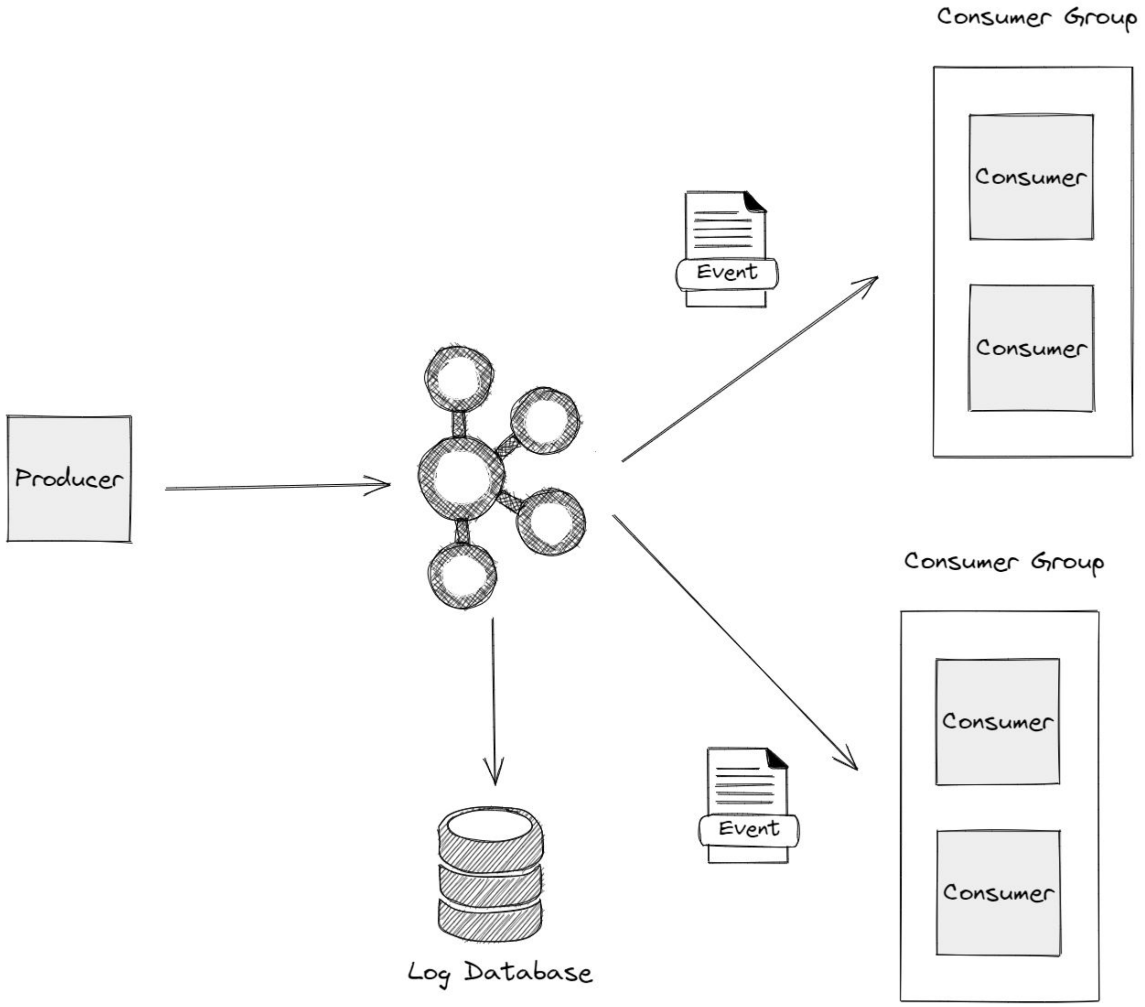

Consumer Group
Producer
Consumer
Consumer

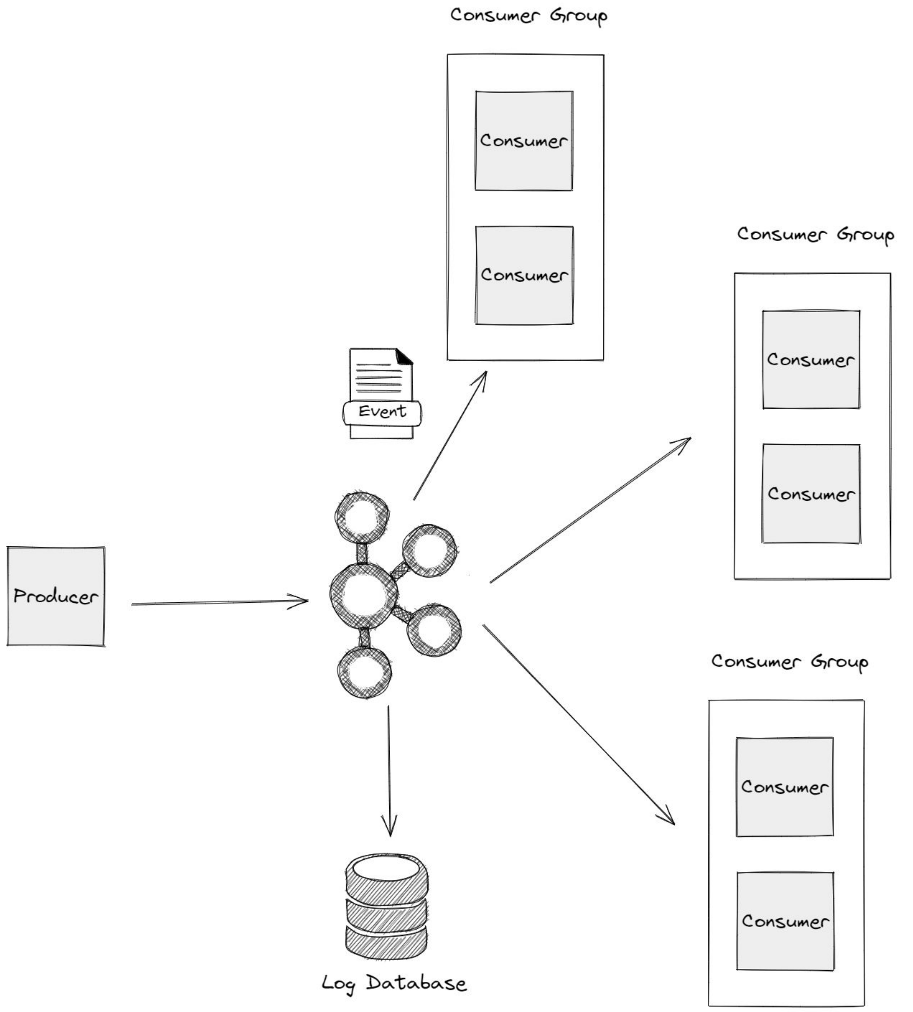

Producer
Consumer Group
Consumer
Consumer
Consumer Group
Consumer
Consumer


Consumer Group
Consumer
Consumer
Producer
Consumer Group
Consumer
Consumer
Log Database


Consumer Group
Consumer
Consumer
Event
Producer
Consumer Group
Consumer
Consumer
Log Database


Consumer Group
Consumer
Consumer
Producer
Consumer Group
Consumer
Consumer
Event
Log Database


Consumer Group
Consumer
Consumer
Event
Producer
Consumer Group
Consumer
Consumer
Event
Log Database


Consumer Group
Consumer
Consumer
Consumer Group
Consumer
Consumer
Event
Producer
Consumer Group
Consumer
Consumer
Log Database

REPLAYING, FAILURES AND YOU: IDEMPOTENCY

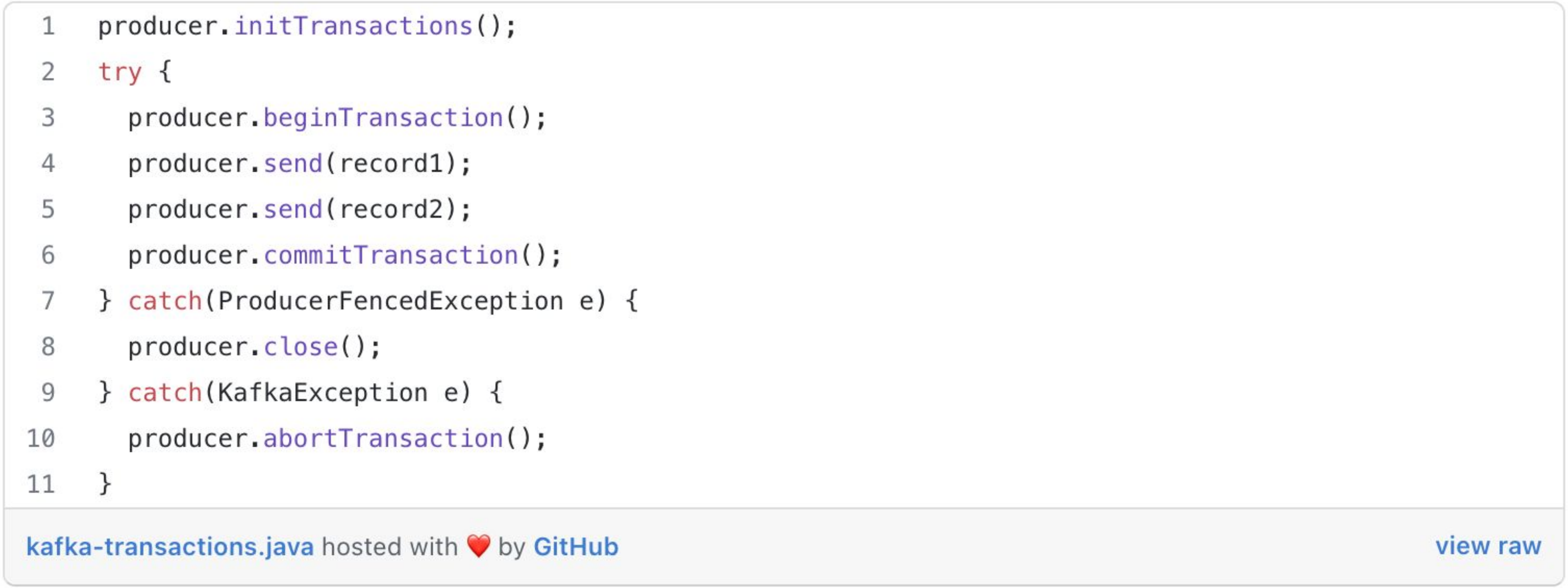
Idempotency & Kafka: Failure

- Imagine a world where the broker fails after writing the event to the log, but before sending an ack to the producer
- The producer only can assume it has not been successful
- The producer send it again
- You’ll have two times the event on the log

Idempotency & Kafka: Leverage Kafka Transaction API

- Topics with a transactional producer can be leveraged.
  - Producer
    - enable.idempotency=true
    - transaction.id=some-id
  - Consumer
    - isolation.level=read_committed/read_uncomitted

Idempotency & Kafka: Leverage Kafka Transaction API



```
producer.initTransactions();
try {
  producer.beginTransaction();
  producer.send(record1);
  producer.send(record2);
  producer.commitTransaction();
} catch(ProducerFencedException e) {
  producer.close();
} catch(KafkaException e) {
  producer.abortTransaction();
}
```

kafka-transactions.java hosted with ❤ by GitHub   view raw

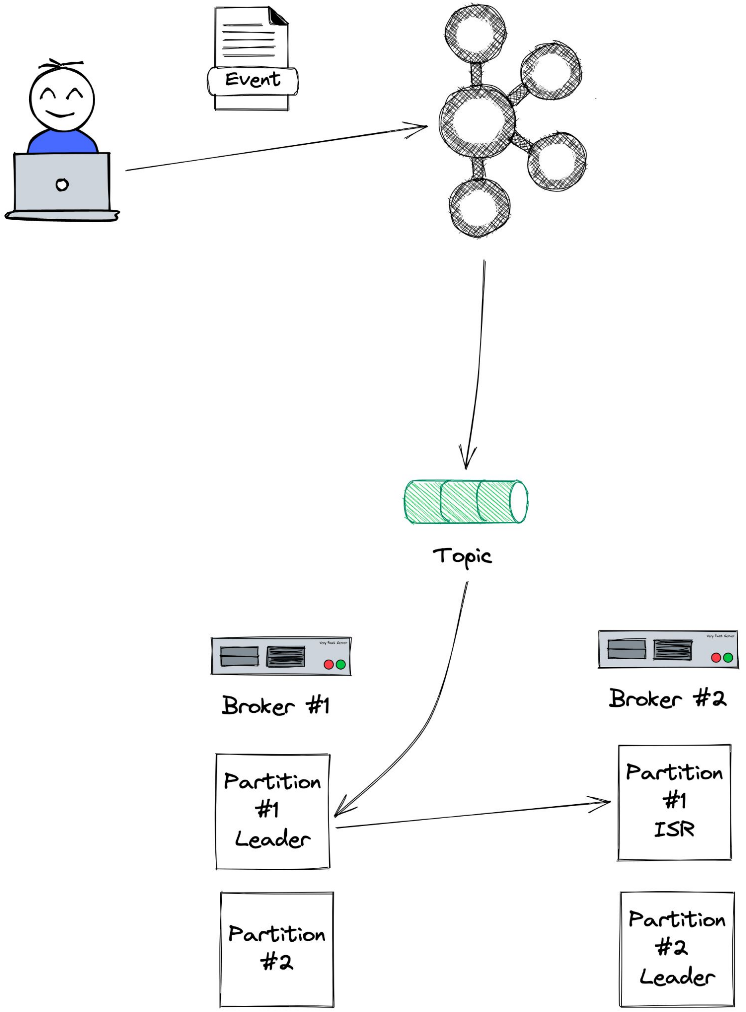
Idempotency & Kafka: Leverage Kafka Transaction API

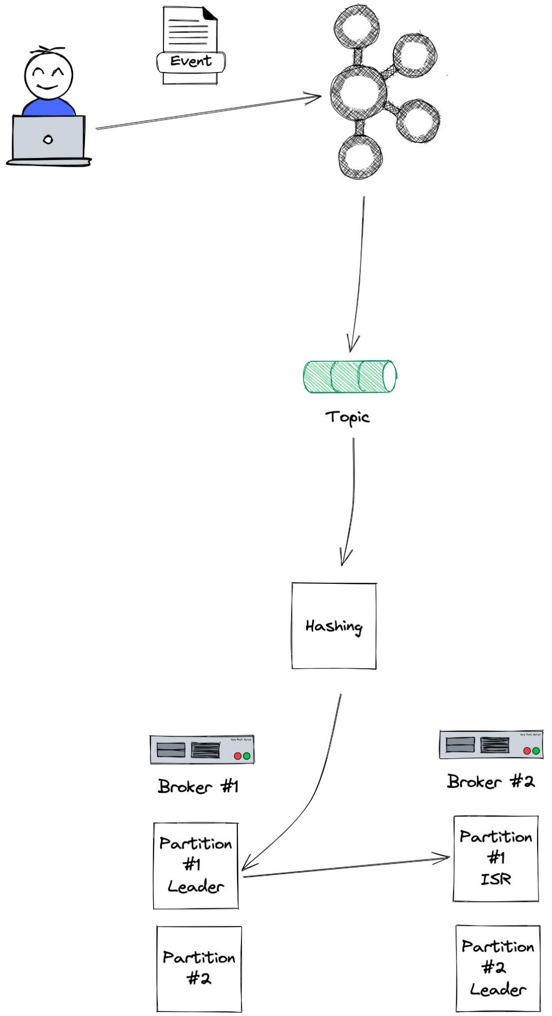
- So why is this tricky?
    - You’re forcing consumer and producer to communicate about enabling idempotency
    - You’re forcing consumer to think more about complexity
    - Exactly once processing works for this case, but you have to look bigger picture. If this is triggered by a user pressing a button? What if they press it twice?
    - Bottom line: Less flexibility, more reliability

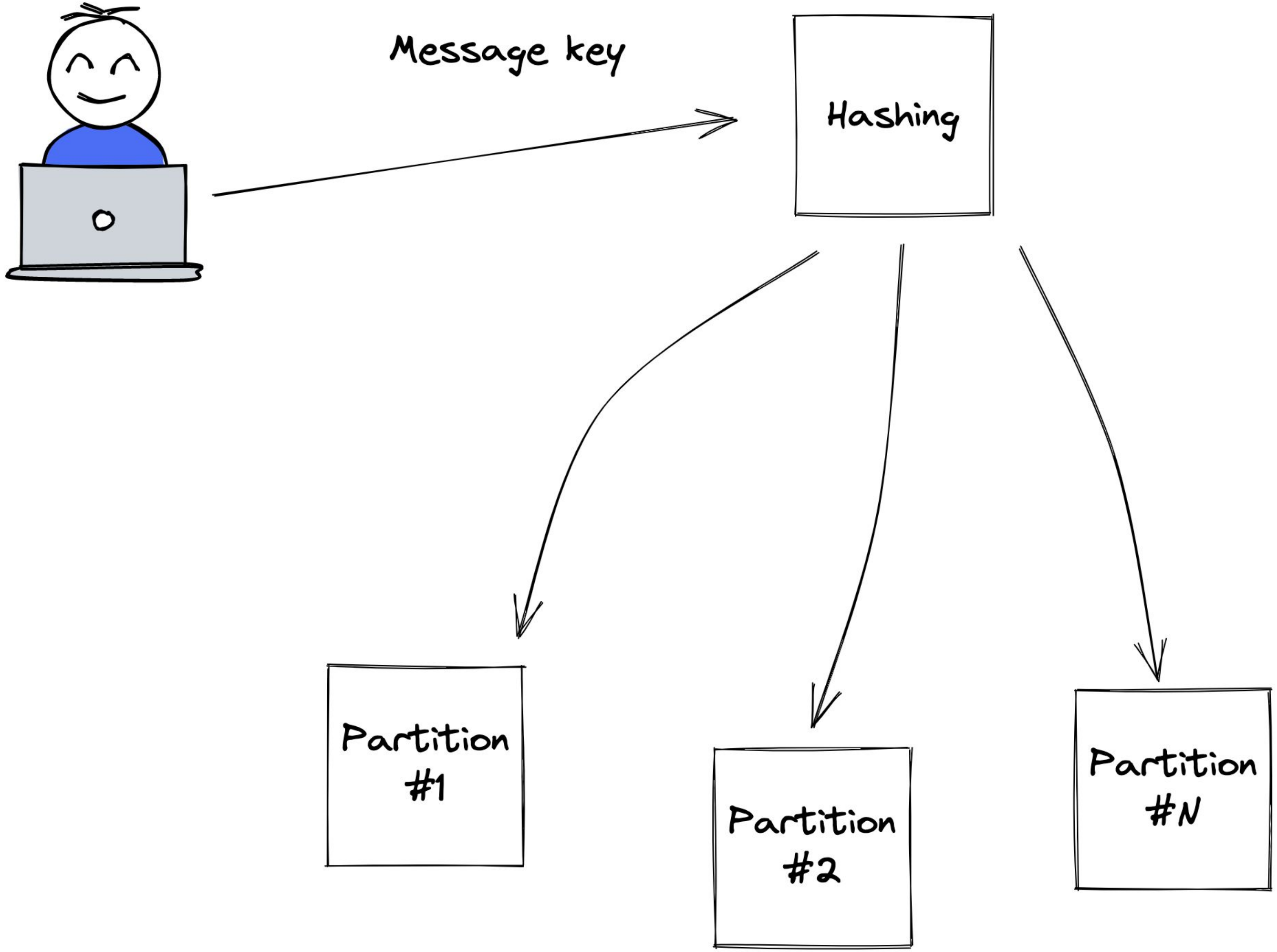
Idempotency & Kafka: Leverage Kafka Transaction API

Idempotency & Kafka: DIY

- Considering Idempotency depends on the producer being transactional “enabled”, and your consumer and producer are loosely coupled, you cannot know for certain
- Use a DIY way of achieving idempotency
- Supply a message key when sending your event to Kafka. Kafka will guarantee the ordering of the same key.
    - Key choice is vital.


Event
Topic
Broker #1
Broker #2
Partition #1 Leader
Partition #1 ISR
Partition #2
Partition #2 Leader


Event
Topic
Hashing
Broker #1
Broker #2
Partition #1 Leader
Partition #1 ISR
Partition #2
Partition #2 Leader



Red Hat

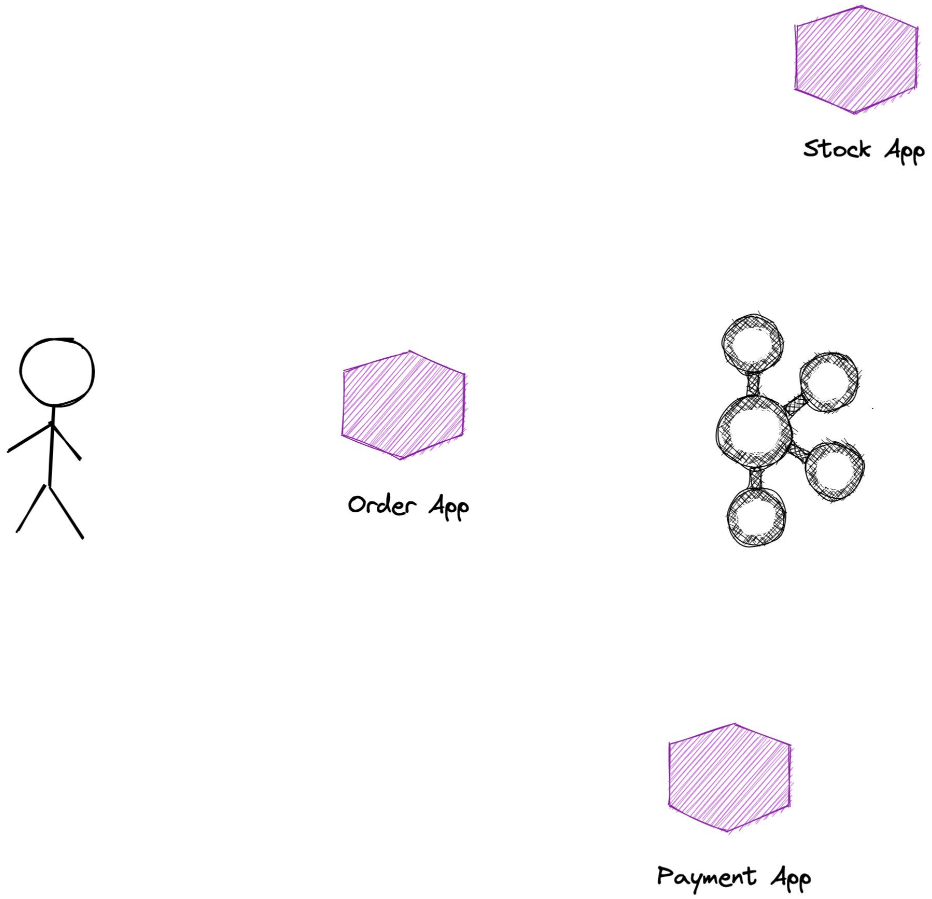
DEMO

Please don’t take this as reference implementation, I am bad at coding

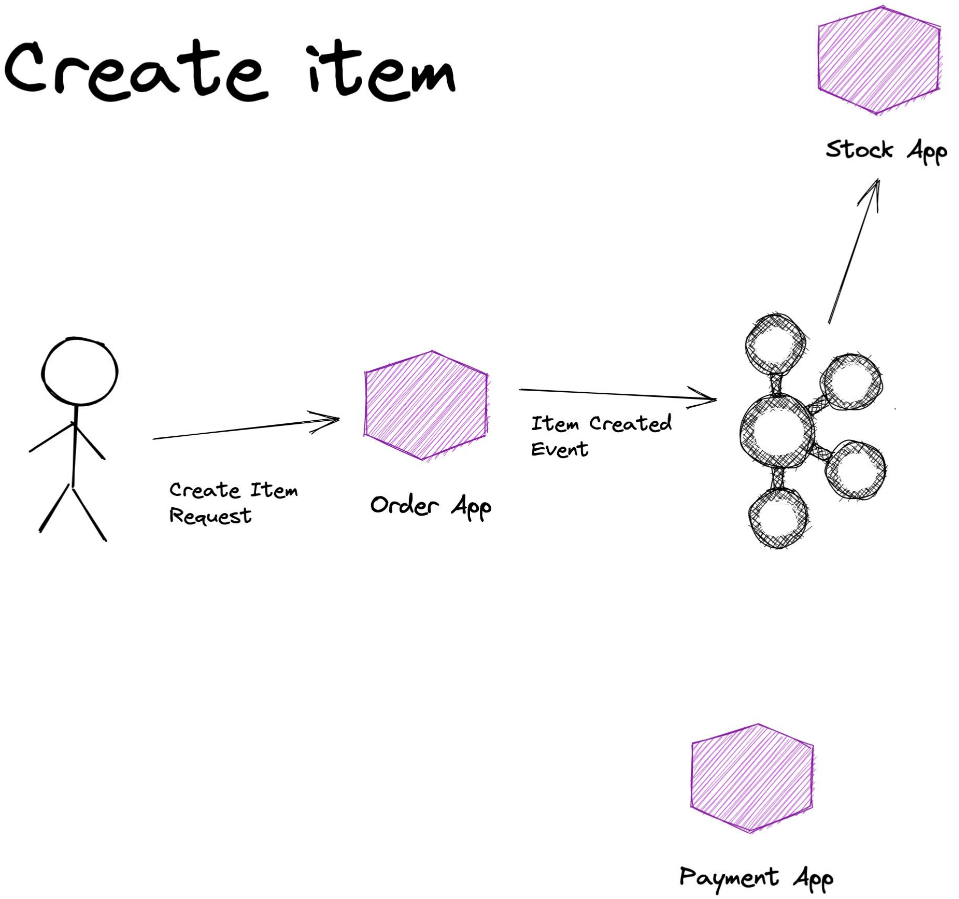
GitHub URL for project

Example deployment

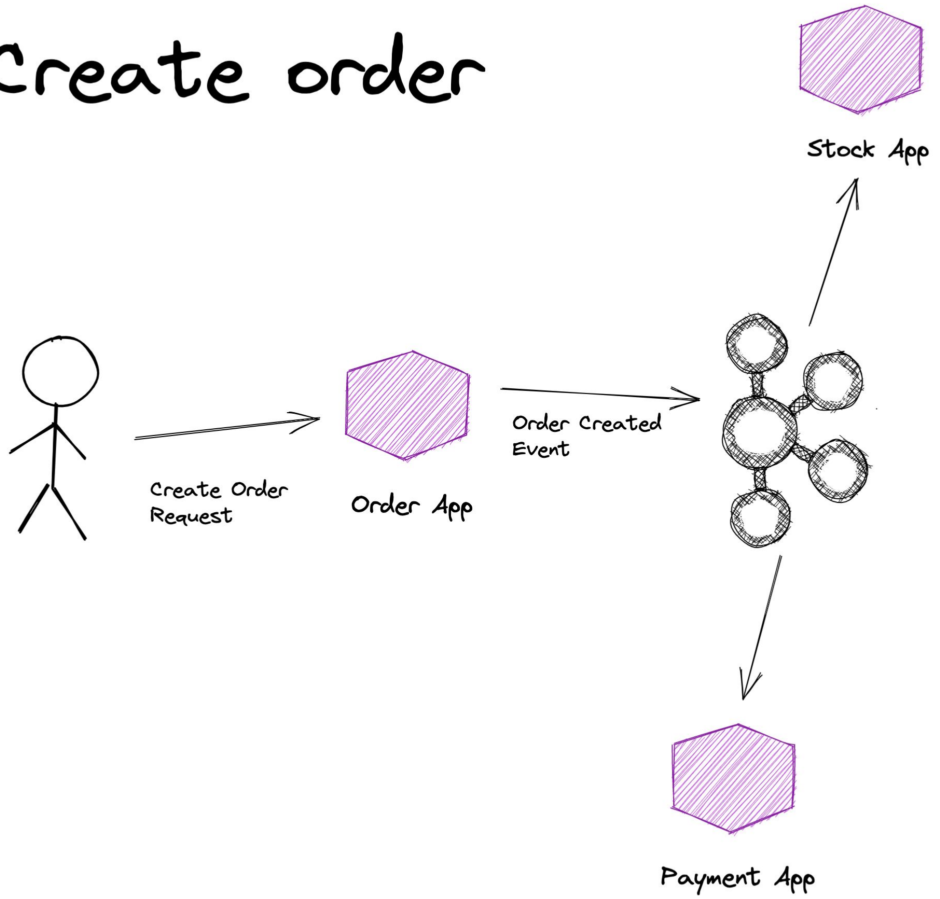
Red Hat


Stock App
Order App
Payment App

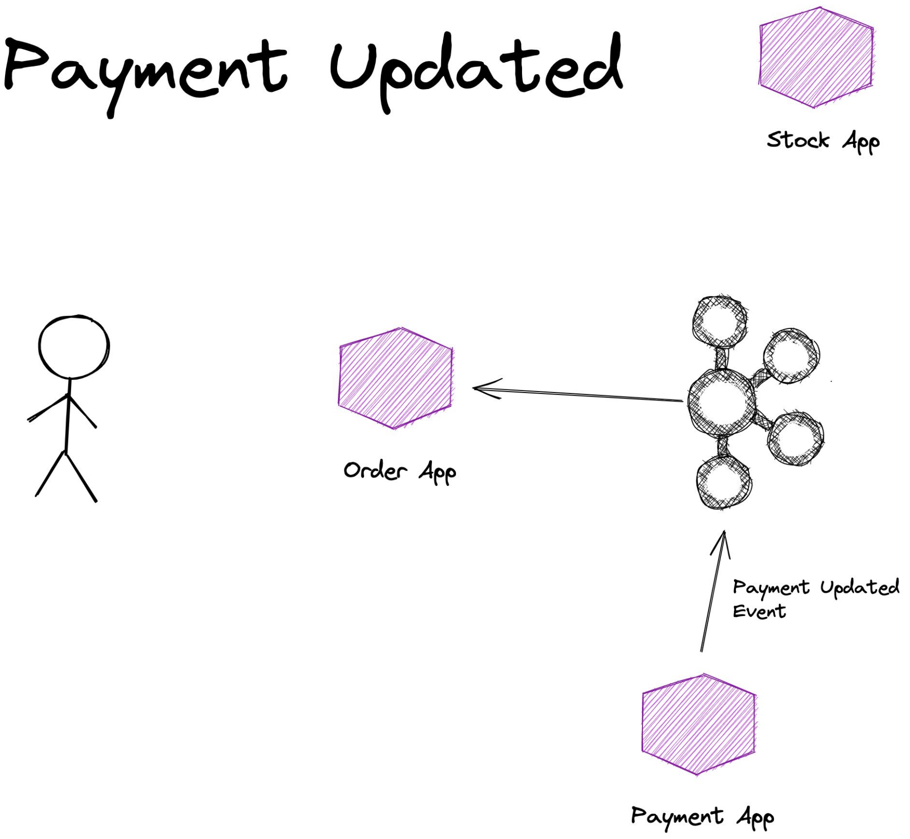
Create item


Stock App
Create Item Request
Order App
Item Created Event
Payment App

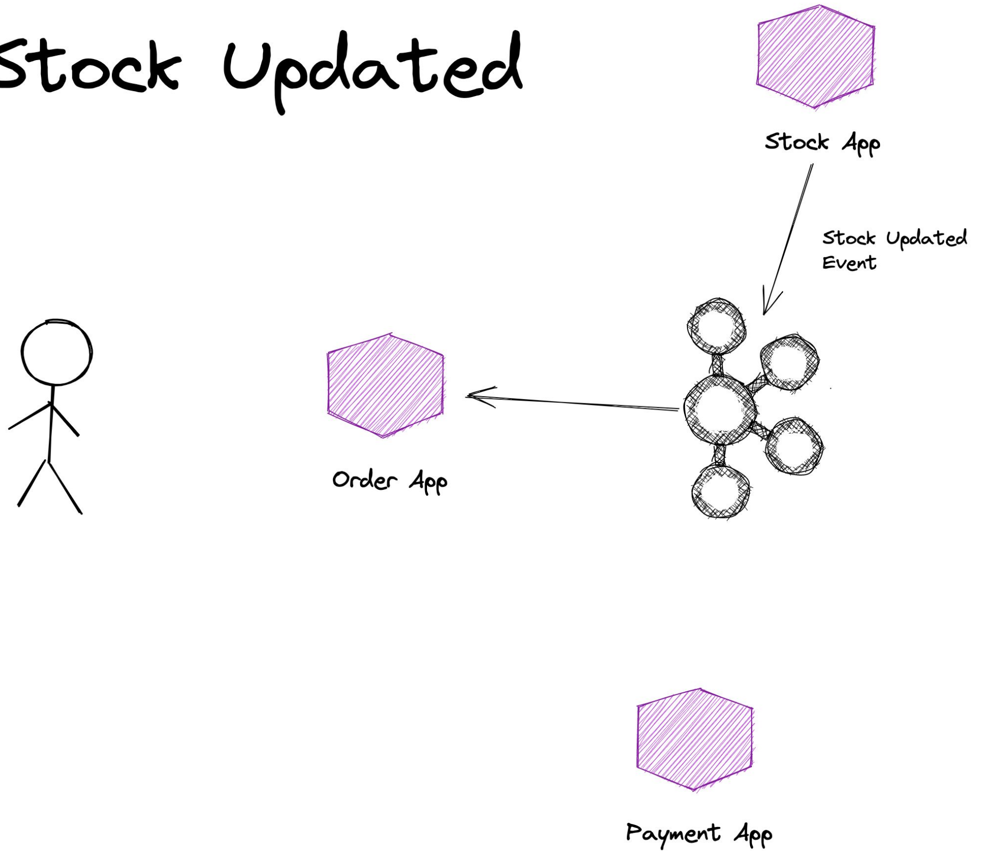
Create order



Payment Updated



Stock Updated



Demo: Issues

- Processing of PaymentUpdatedEvent is prone to duplication and inconsistency
- Rollback of order “transaction” is not implemented
- Not in scope: Message ordering. Not really relevant here, but state engine could encounter issues in some cases.

CONCLUSION

Conclusion

You can do a lot with Kafka, but know the **tradeoffs**.
Kafka helps when you're looking for **decentralized** control.
Don't **mindlessly adopt** Kafka without considering it's implications for both **consumer and producer**
Be sure to match an event driven **architecture** with your **business case**

USING KAFKA FOR EVERY USE CASE
ME
WHY IS THIS SO COMPLEX
imgflip.com

Thank you

linkedin.com/company/Red-Hat

facebook.com/RedHatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat