instaclustr

Introducing Apache Kafka®

Paul Brebner
Technical Evangelist

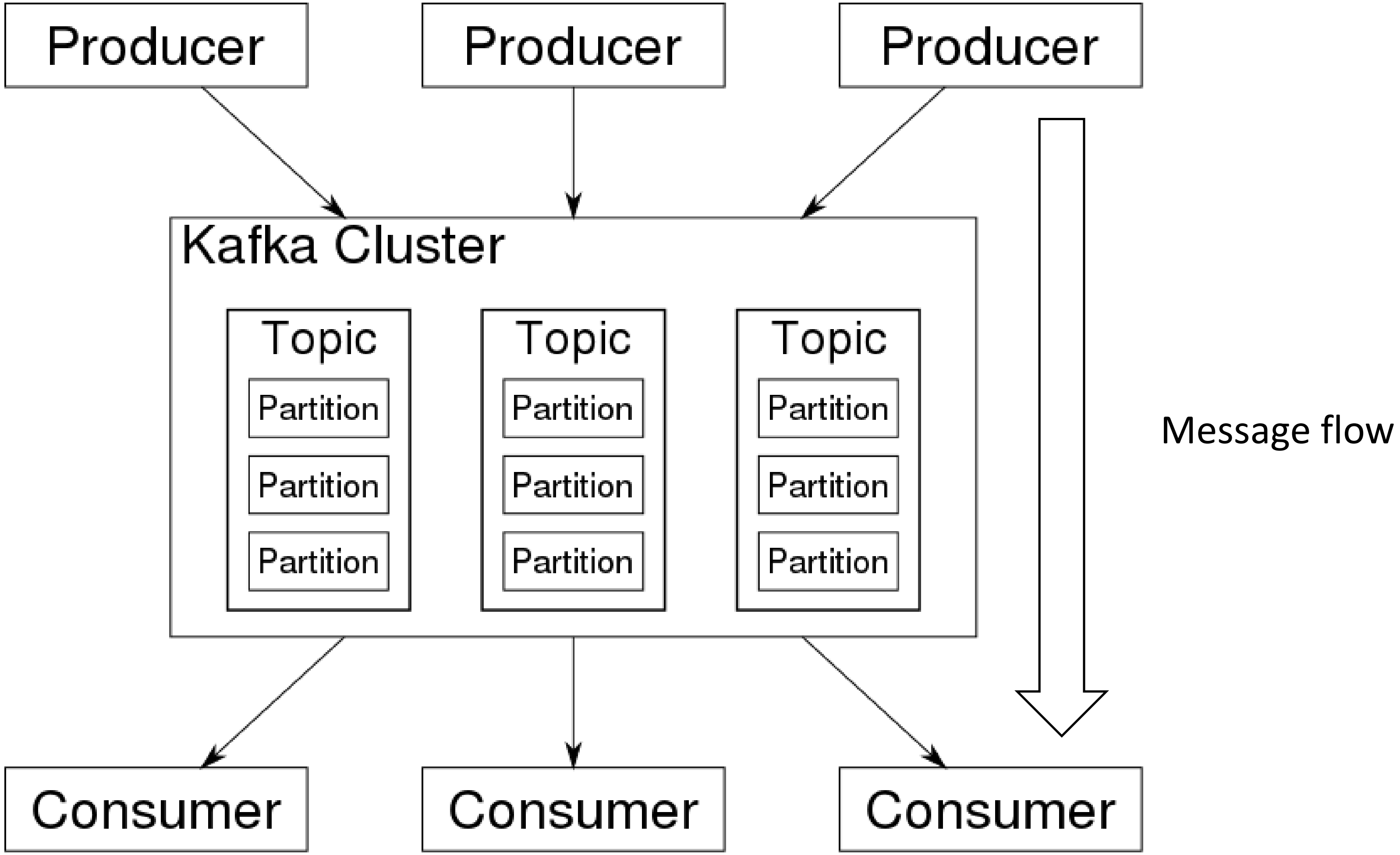
February 2019

What is Kafka?

instaclustr

Distributed streams Processing System

Messages sent by
Distributed Producers
to
Distributed Consumers
Consumers
via
Distributed Kafka Cluster
Cluster

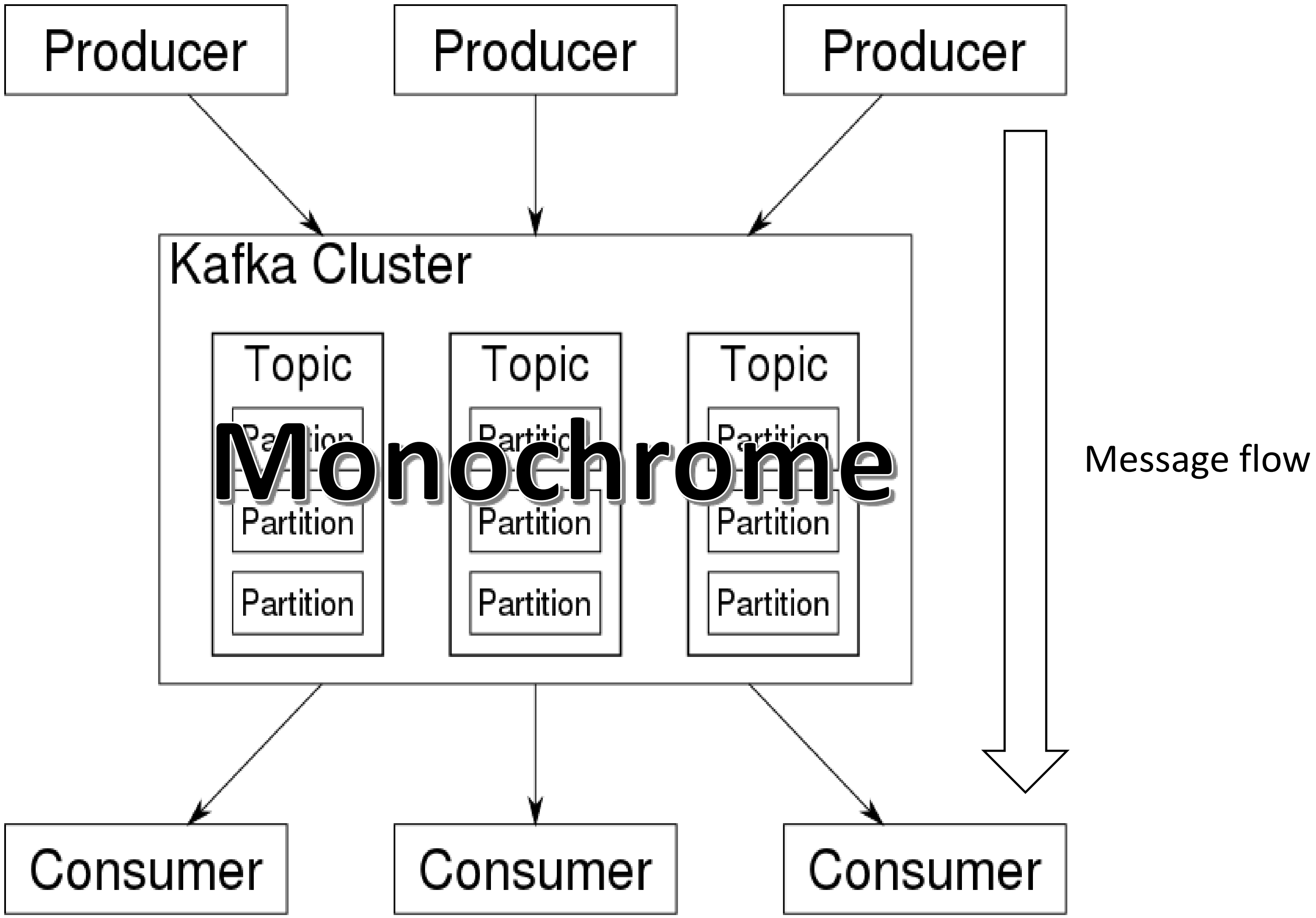


Kafka benefits

- Fast
- Scalable
- Reliable
- Durable
- Open Source
- Managed Service

Kafka benefits

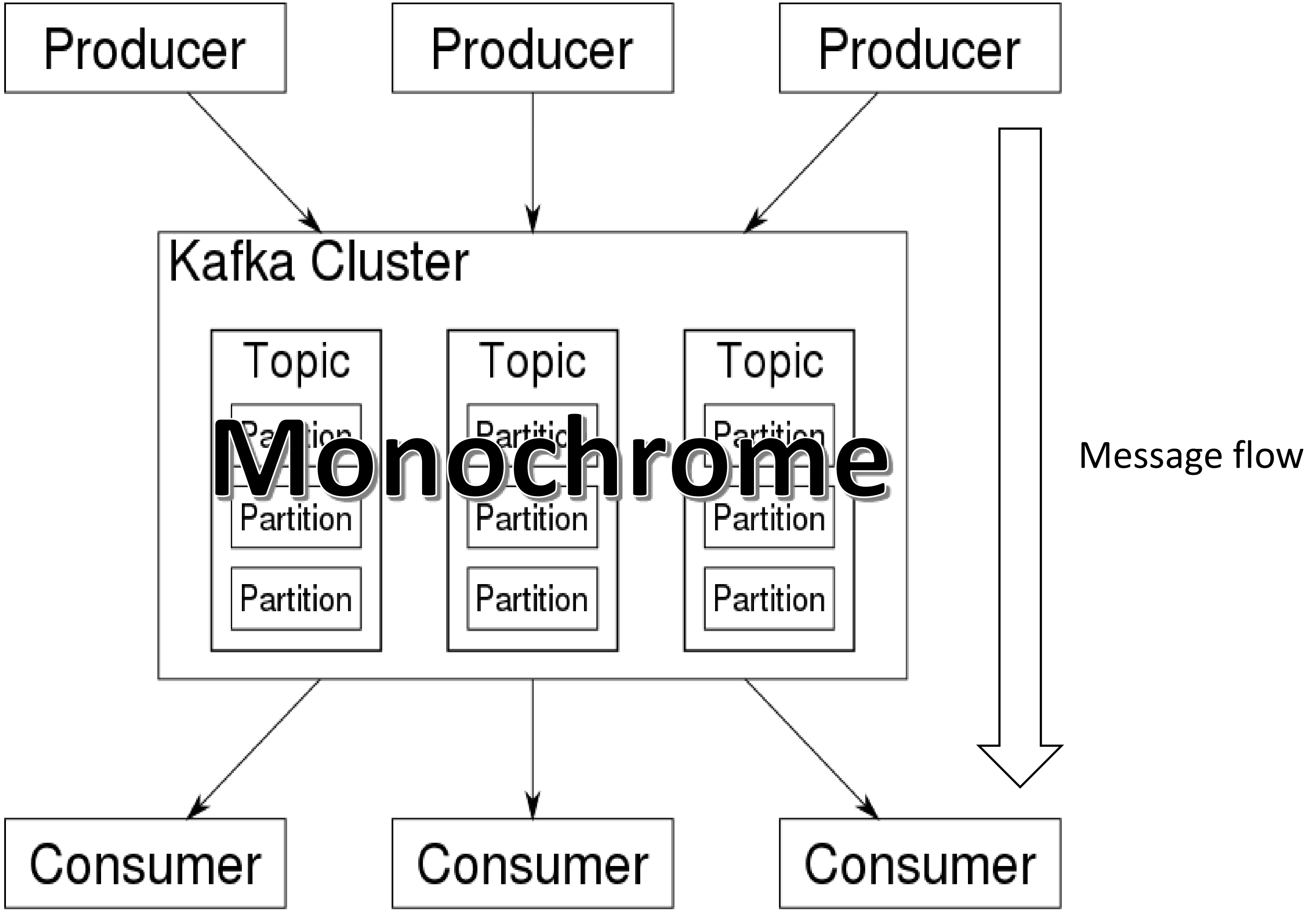
- Fast – high throughput and low latency
- Scalable – horizontally scalable with nodes and partitions
- Reliable – distributed and fault tolerant
- Durable - zero data loss, messages persisted to disk with immutable log
- Open Source – An Apache project
- Available as a Managed Service - on multiple cloud platforms


Producer
Producer
Producer
Kafka Cluster
Topic
Topic
Topic
Partition
Partition
Partition
Partition
Partition
Partition
Partition
Partition
Partition
Message flow
Consumer
Consumer
Consumer

Monochrome

This talk Colourful Extended Story

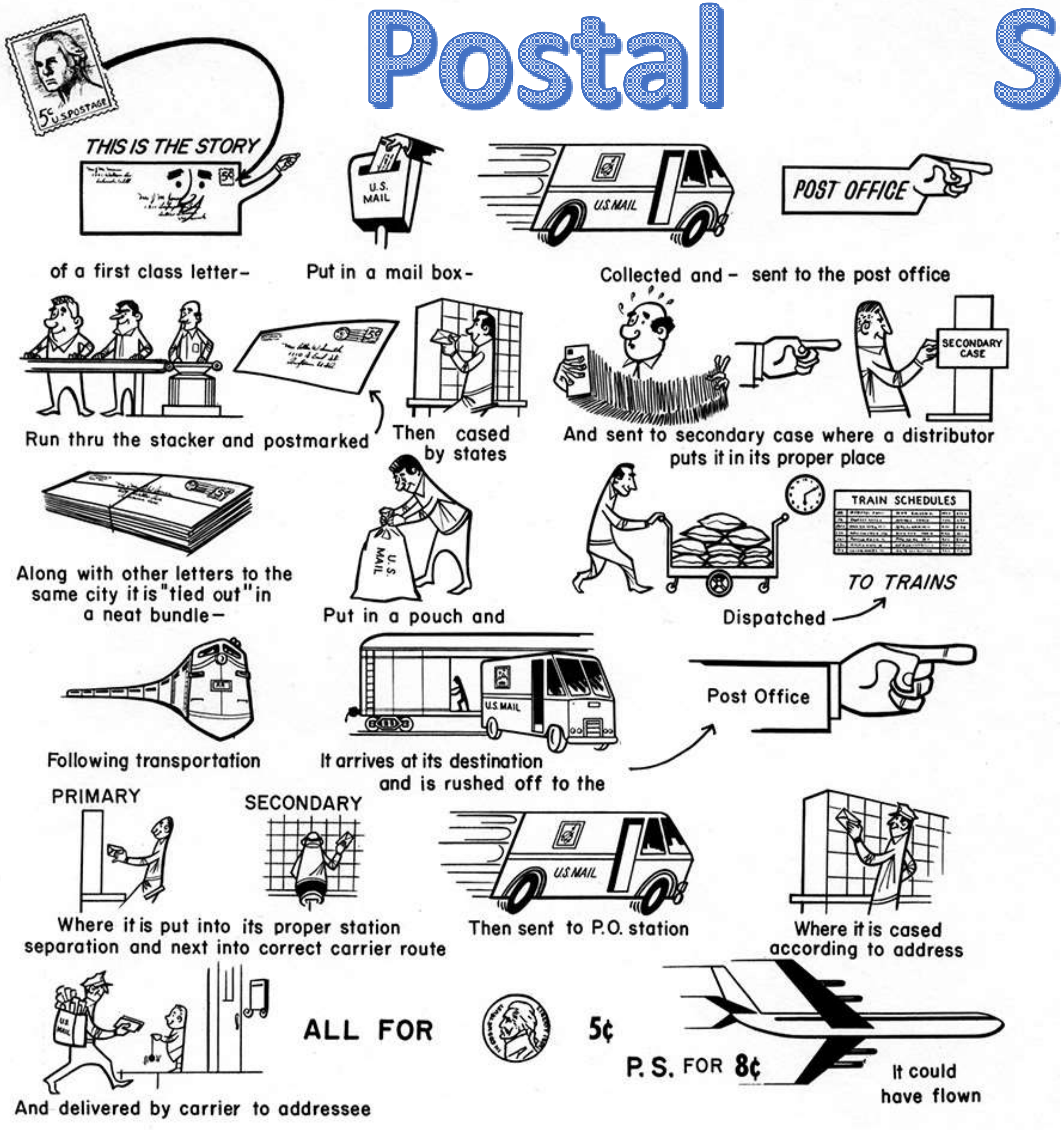
instaclustr



Monochrome

Let's build the

Kafka Postal Service


5¢ U.S. POSTAGE
THIS IS THE STORY
of a first class letter-
U.S. MAIL
Put in a mail box-
U.S. MAIL
Collected and -
POST OFFICE
sent to the post office
Run thru the stacker and postmarked
Then cased by states
SECONDARY CASE
And sent to secondary case where a distributor puts it in its proper place
Along with other letters to the same city it is "tied out" in a neat bundle-
U.S. MAIL
Put in a pouch and
TRAIN SCHEDULES
TO TRAINS
Dispatched
Following transportation
U.S. MAIL
It arrives at its destination and is rushed off to the
Post Office
PRIMARY
SECONDARY
Where it is put into its proper station separation and next into correct carrier route
U.S. MAIL
Then sent to P.O. station
Where it is cased according to address
And delivered by carrier to addressee
ALL FOR
5¢
P. S. FOR 8¢
It could have flown

To send messages from A to B

A

B

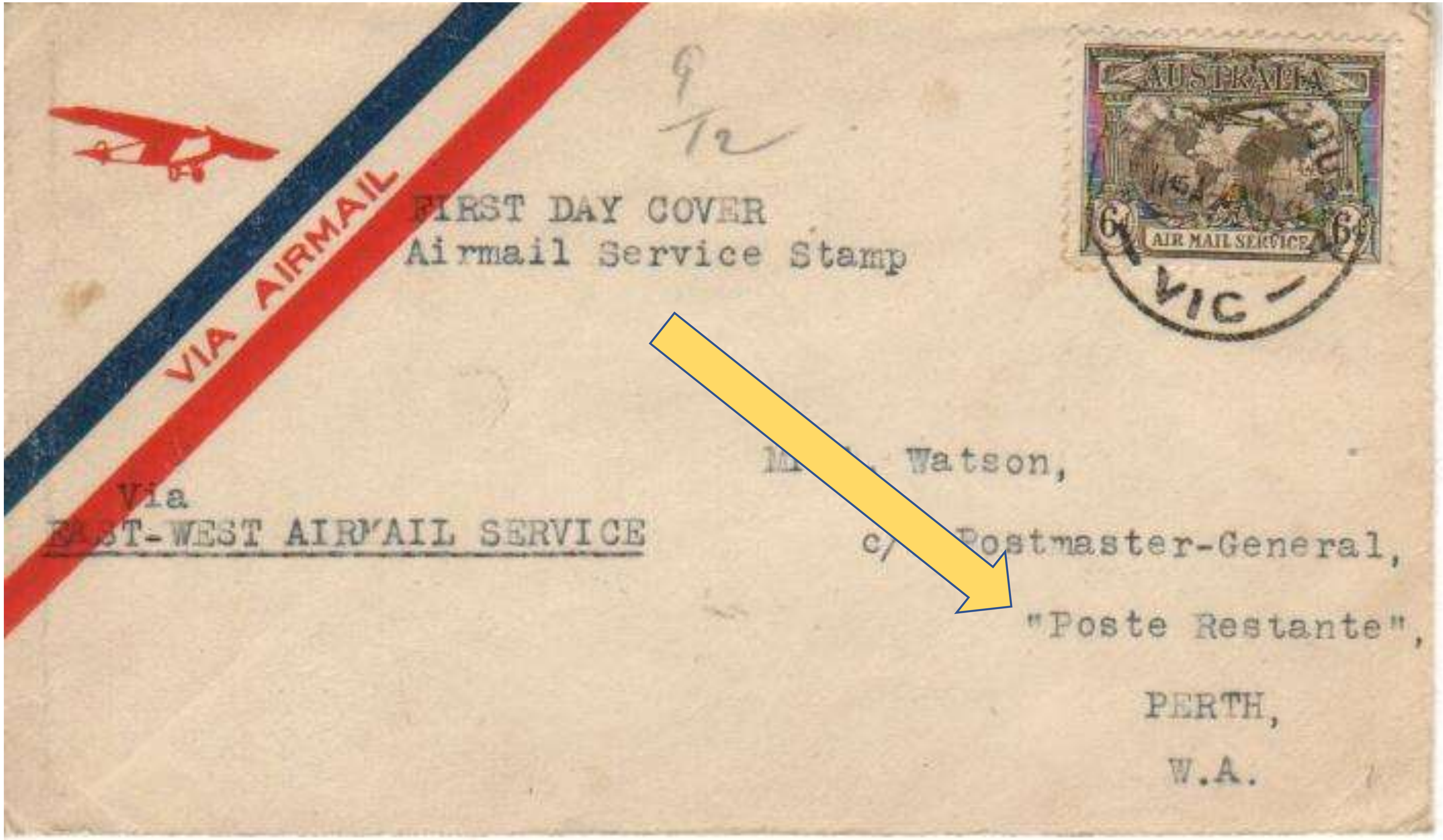
“A” is the Producer – sends a message, to

“B” the Consumer (recipient) of the message

Due to decline in “snail mail” direct deliveries

Due to decline in “snail mail” direct deliveries

Instead … "Poste Restante"



9/12

FIRST DAY COVER
Airmail Service Stamp

Via
EAST-WEST AIRMAIL SERVICE

M[illegible] Watson,
c/[illegible] Postmaster-General,
"Poste Restante",
PERTH,
W.A.

“Poste Restante”?

- Not a post office in a restaurant
- General delivery (in the US)
- The mail is delivered to a post office, they hold it for you until you call

Consumers “poll” for messages by visiting the Poste Restante counter at the post office

POSTE
RESTANTE

Kafka topics act like a Post Office

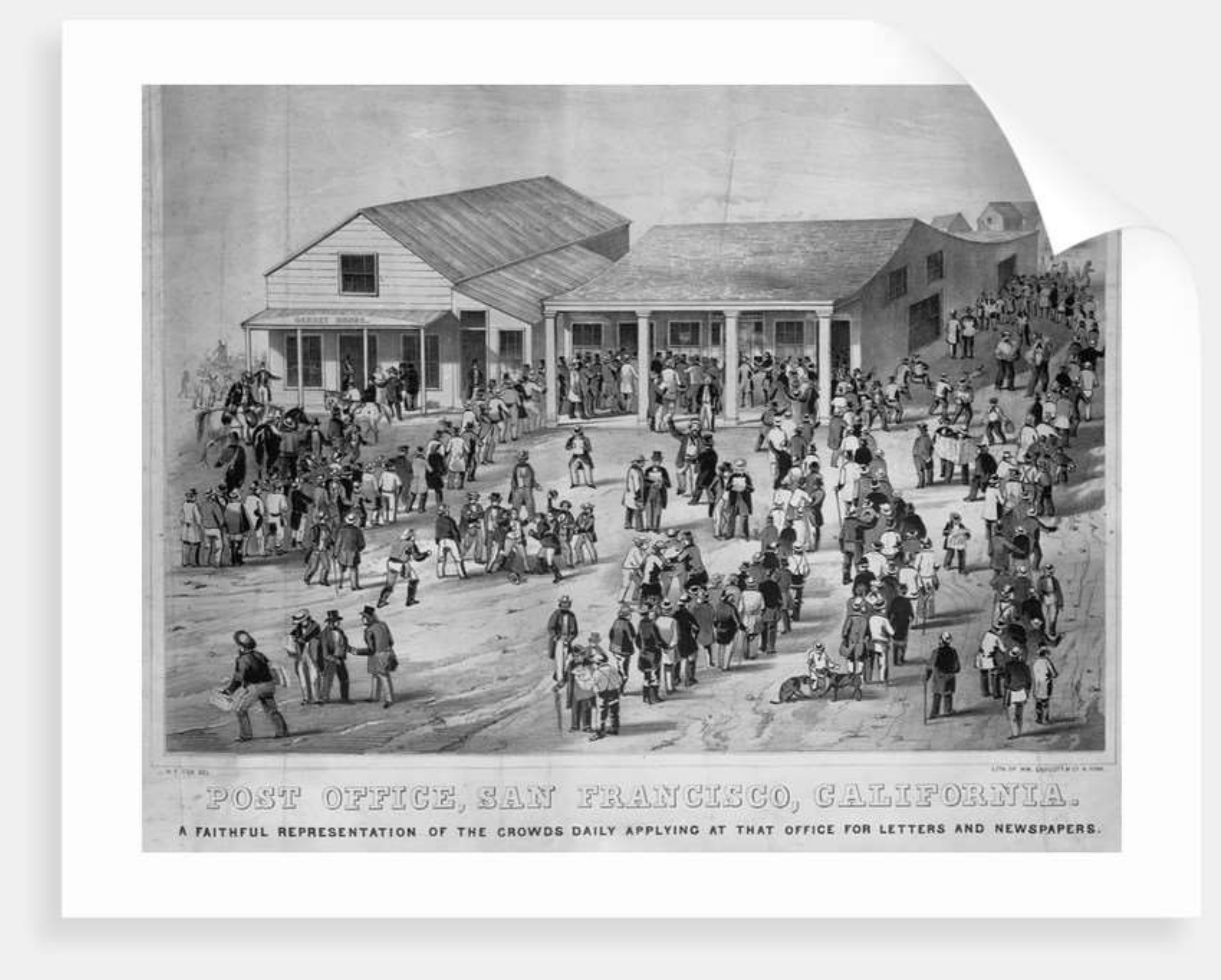
Benefits include

- Disconnected delivery – consumer doesn’t need to be available to receive messages

- Less effort for the messaging service – only has to deliver to a few locations not every consumer

- Can scale better and handle more complex delivery semantics!

Scalability? Many consumers for a topic?



POST OFFICE, SAN FRANCISCO, CALIFORNIA.

A FAITHFUL REPRESENTATION OF THE CROWDS DAILY APPLYING AT THAT OFFICE FOR LETTERS AND NEWSPAPERS.

A single counter introduces delays

AUSTRALIAN WAR MEMORIAL P00784.042

More counters increases concurrency

Kafka topics have >= 1 Partitions (“counters”)

- Partitions increase consumer concurrency
- Increase throughput
- Reduce latency

What’s a Kafka message?

A Record – like a letter

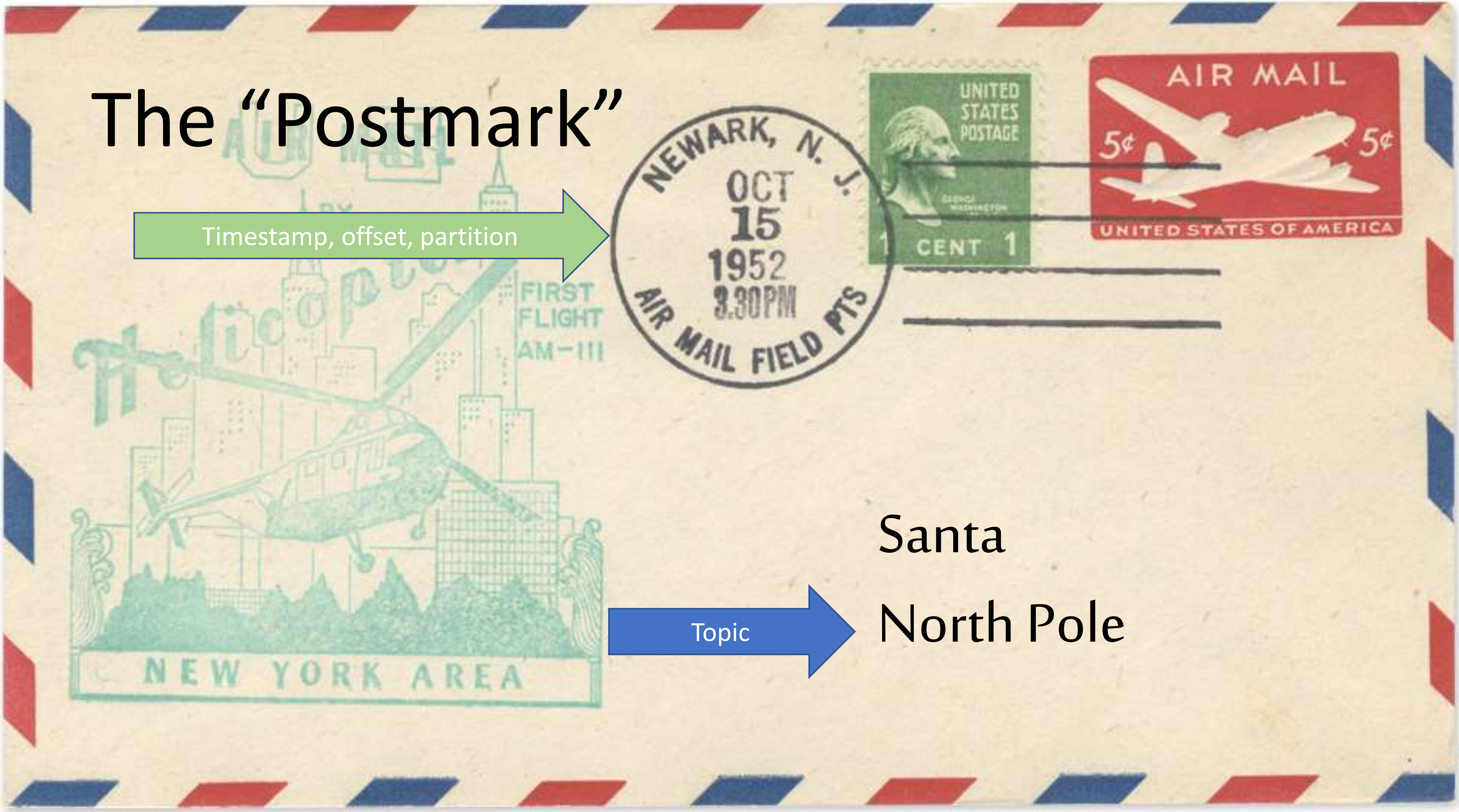
Santa
North Pole

Topic is the destination

Santa

Topic → North Pole

The “Postmark”

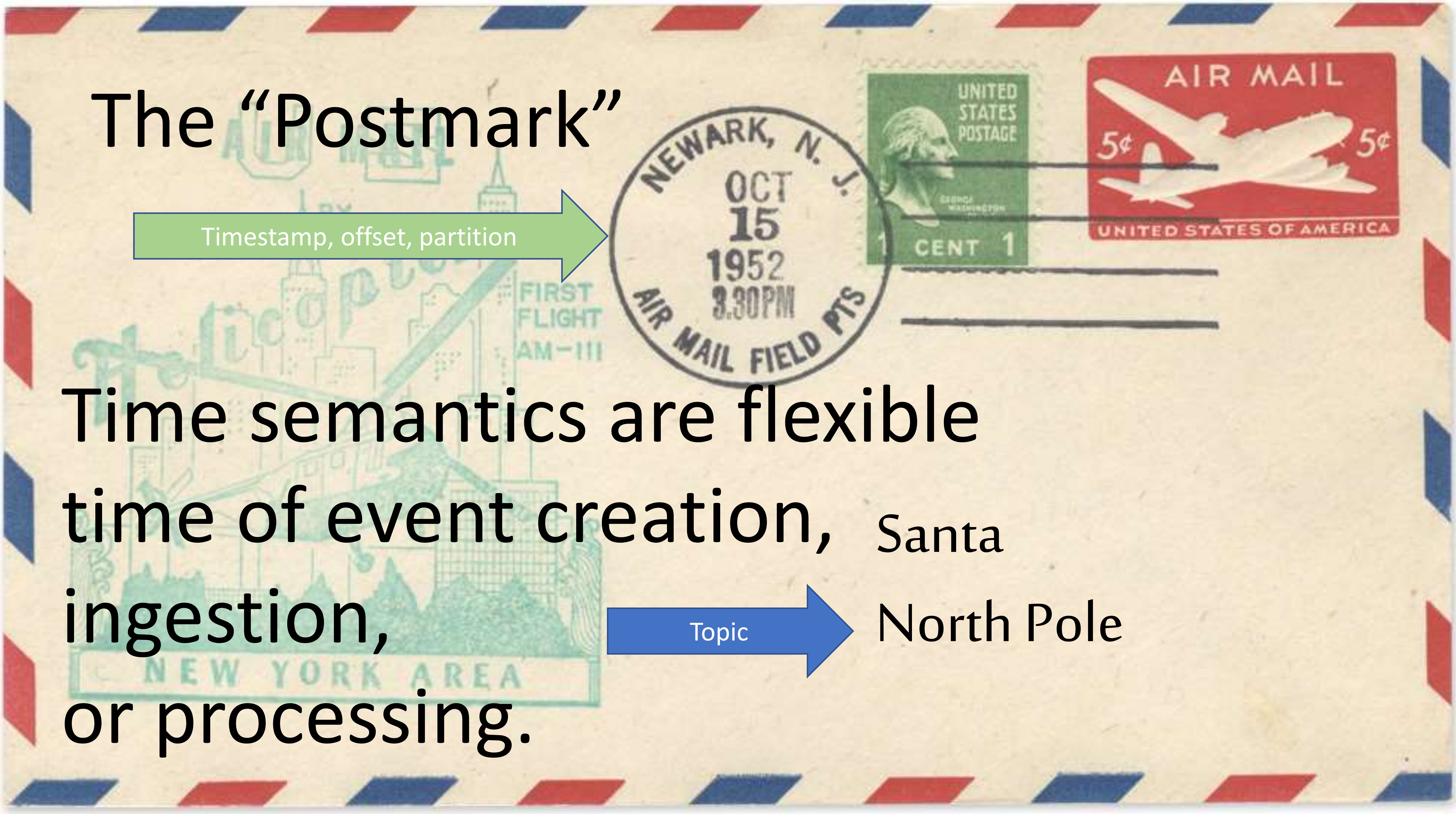


Timestamp, offset, partition

Topic

Santa
North Pole

The “Postmark”



Timestamp, offset, partition

Time semantics are flexible

time of event creation,

ingestion,

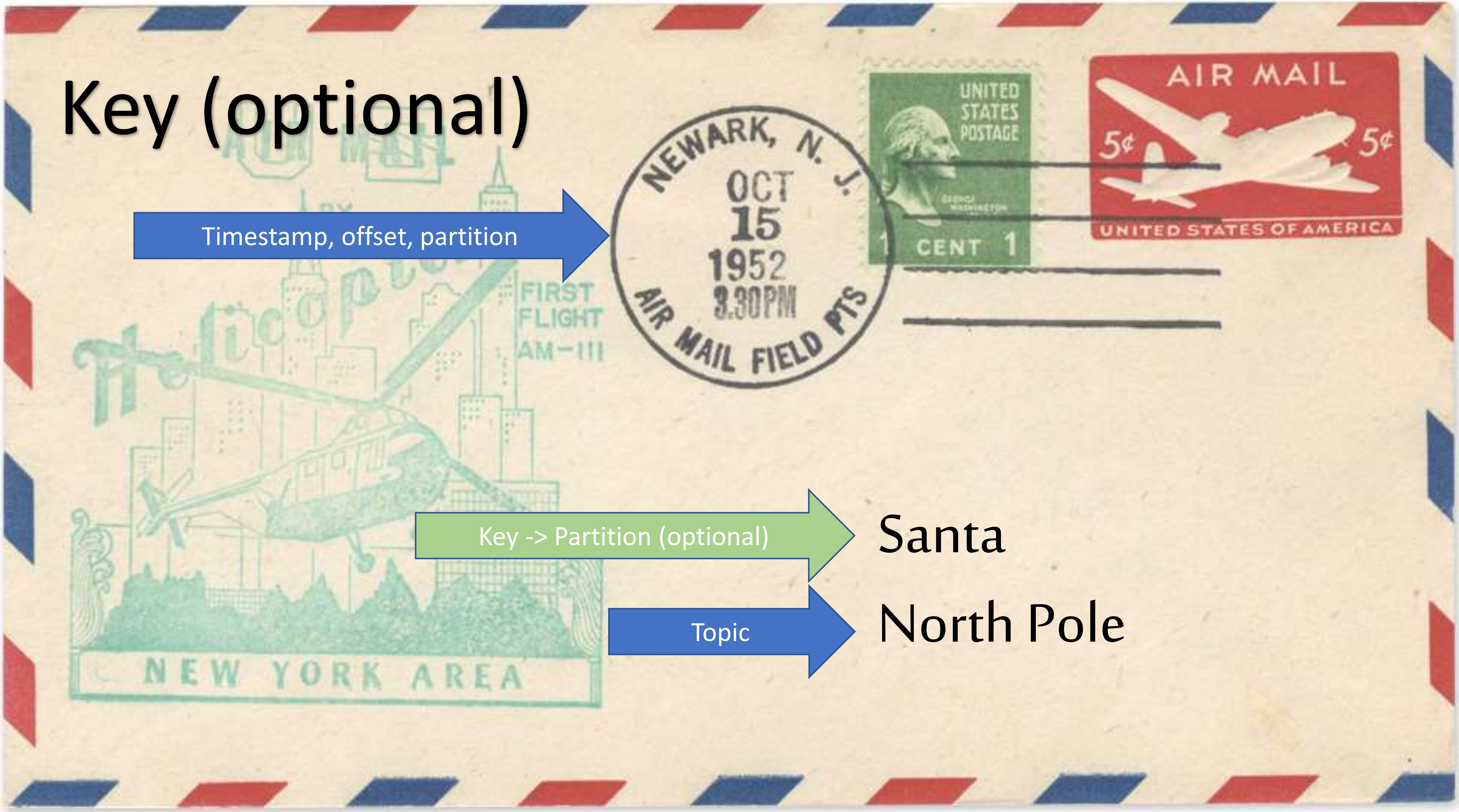
or processing.

Topic

Santa

North Pole

Key (optional)



Timestamp, offset, partition

Key -> Partition (optional)

Santa

Topic

North Pole

Key (optional)

Timestamp, offset, partition

Refines the destination

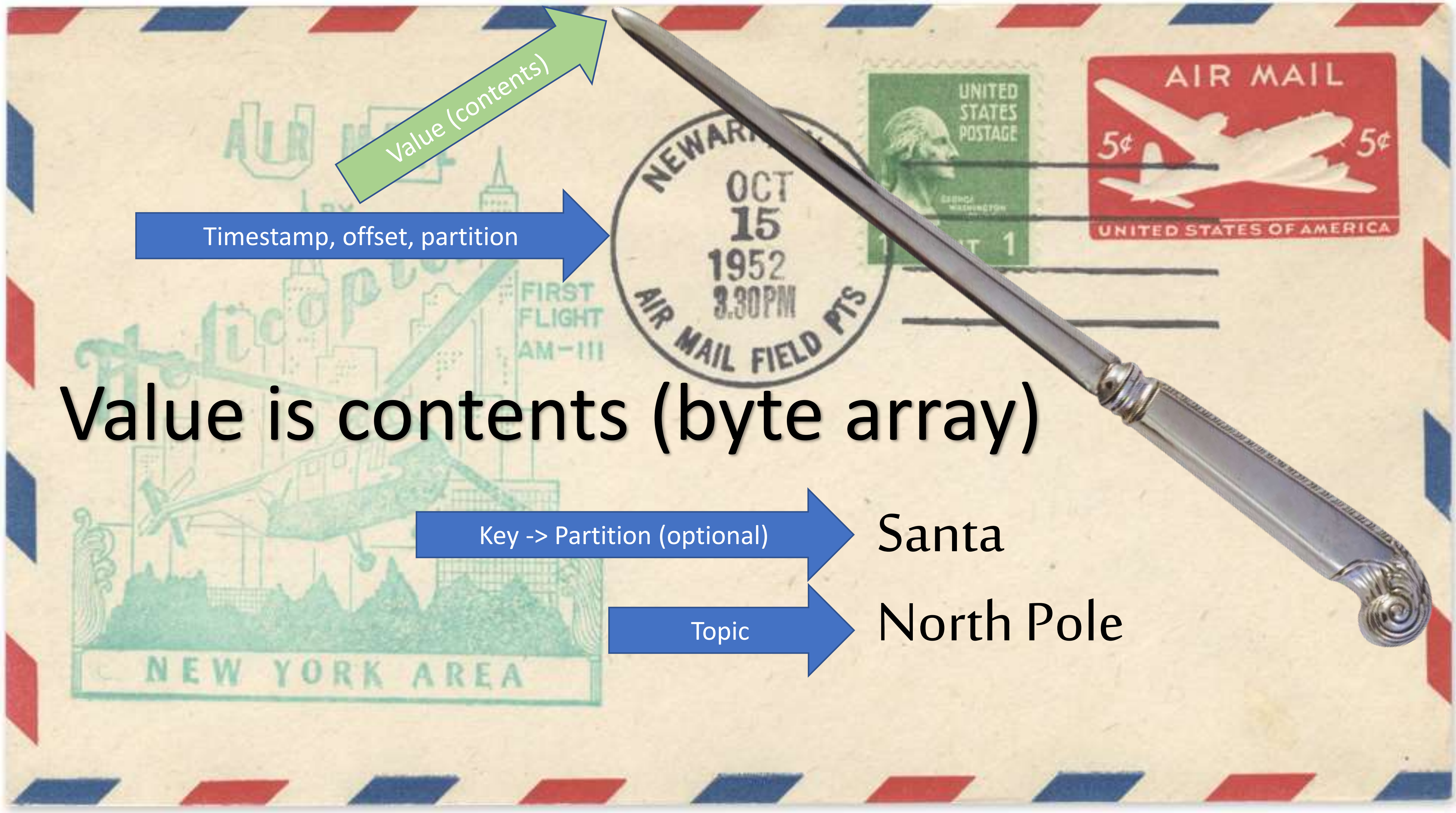
Send to Santa not just any Elf

Key -> Partition (optional)

Santa

Topic

North Pole

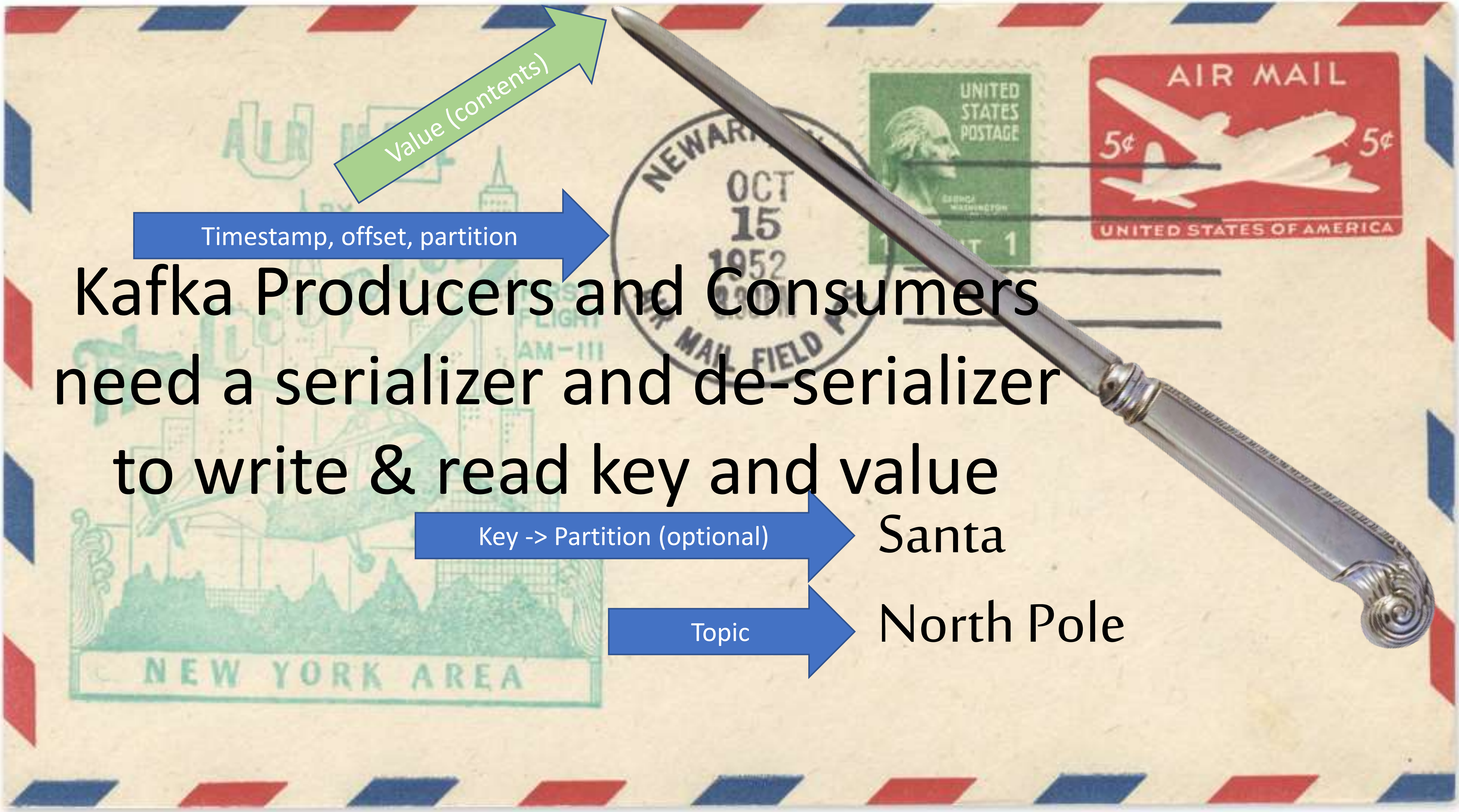


Value is contents (byte array)

Santa

North Pole

Kafka Producers and Consumers need a serializer and de-serializer to write & read key and value



Value (contents)

Timestamp, offset, partition

Key -> Partition (optional)

Topic

Santa

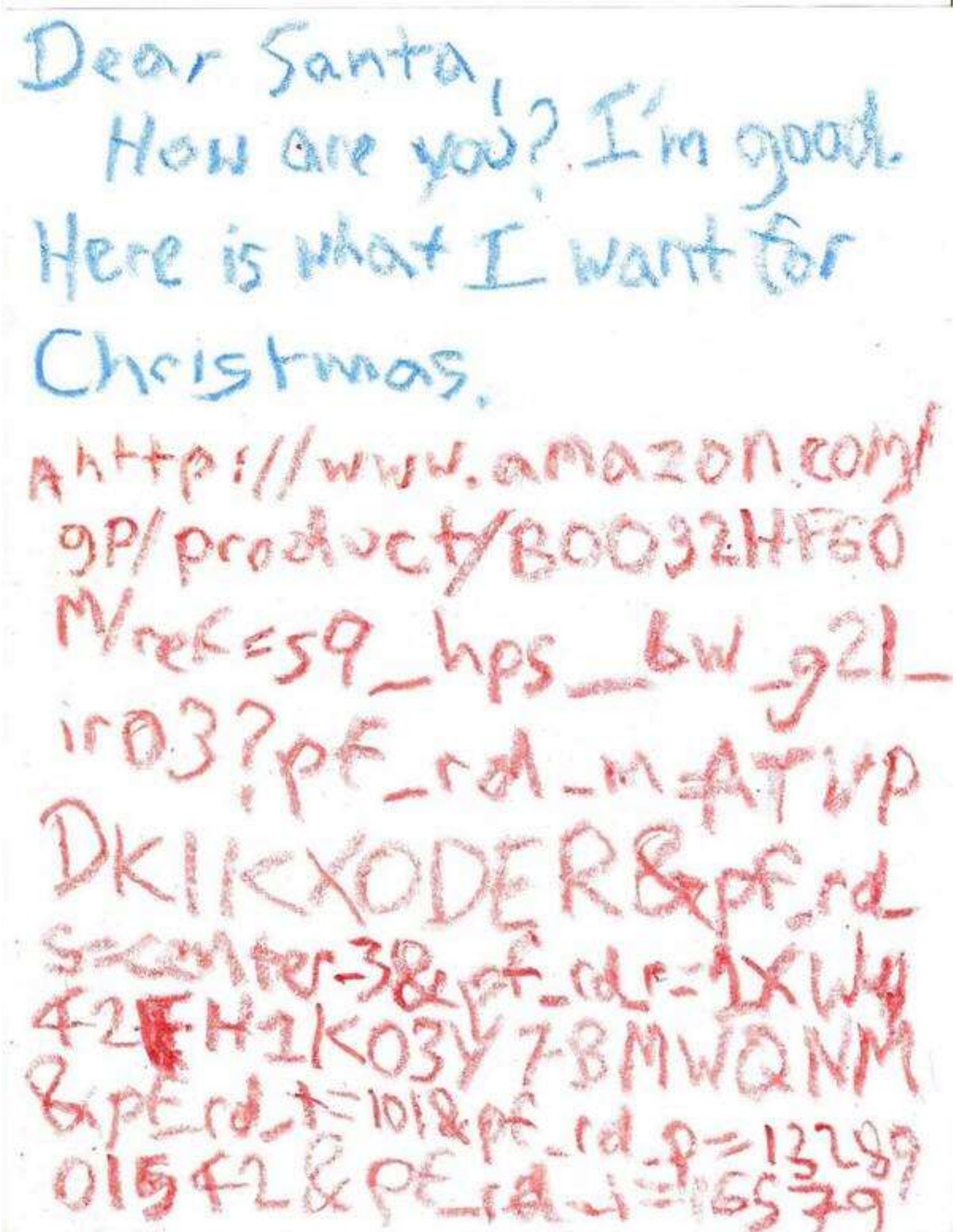
North Pole

- Kafka doesn’t look at the value

- Consumer can read value

- And try to make sense of the message

- What will Santa be delivering?!

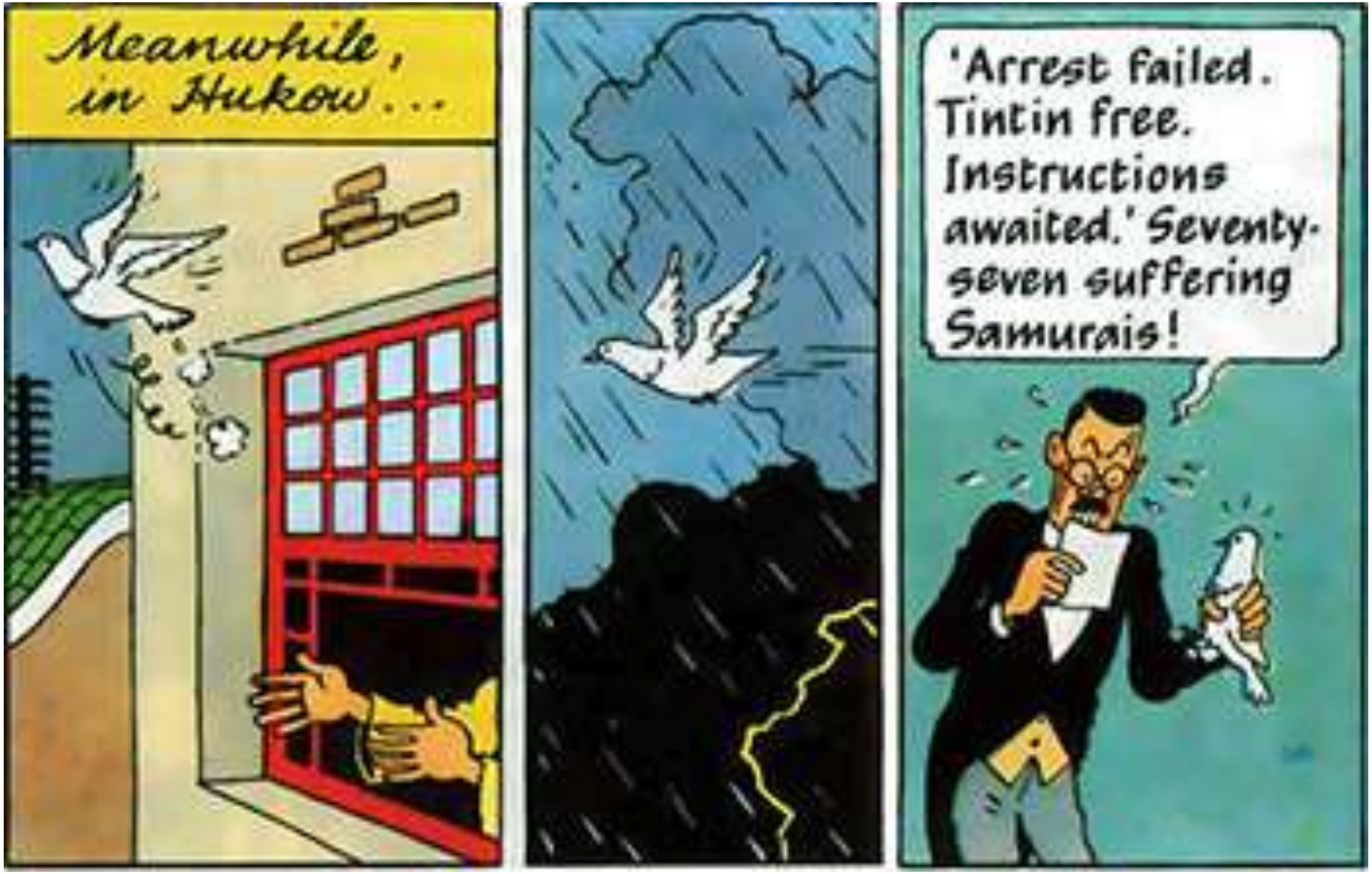


Dear Santa,
How are you? I'm good.
Here is what I want for
Christmas.
Ahttp://www.amazon.com/
gp/product/B0032HF60
M/ref=s9_hps_bw_g21_
ir03?pf_rd_m=ATVP
DKIKX0DER&pf_rd_
s=center-3&pf_rd_r=1XWy
42FH1K03Y7BMWQNM
&pf_rd_t=101&pf_rd_p=13289
015f2&pf_rd_i=16579

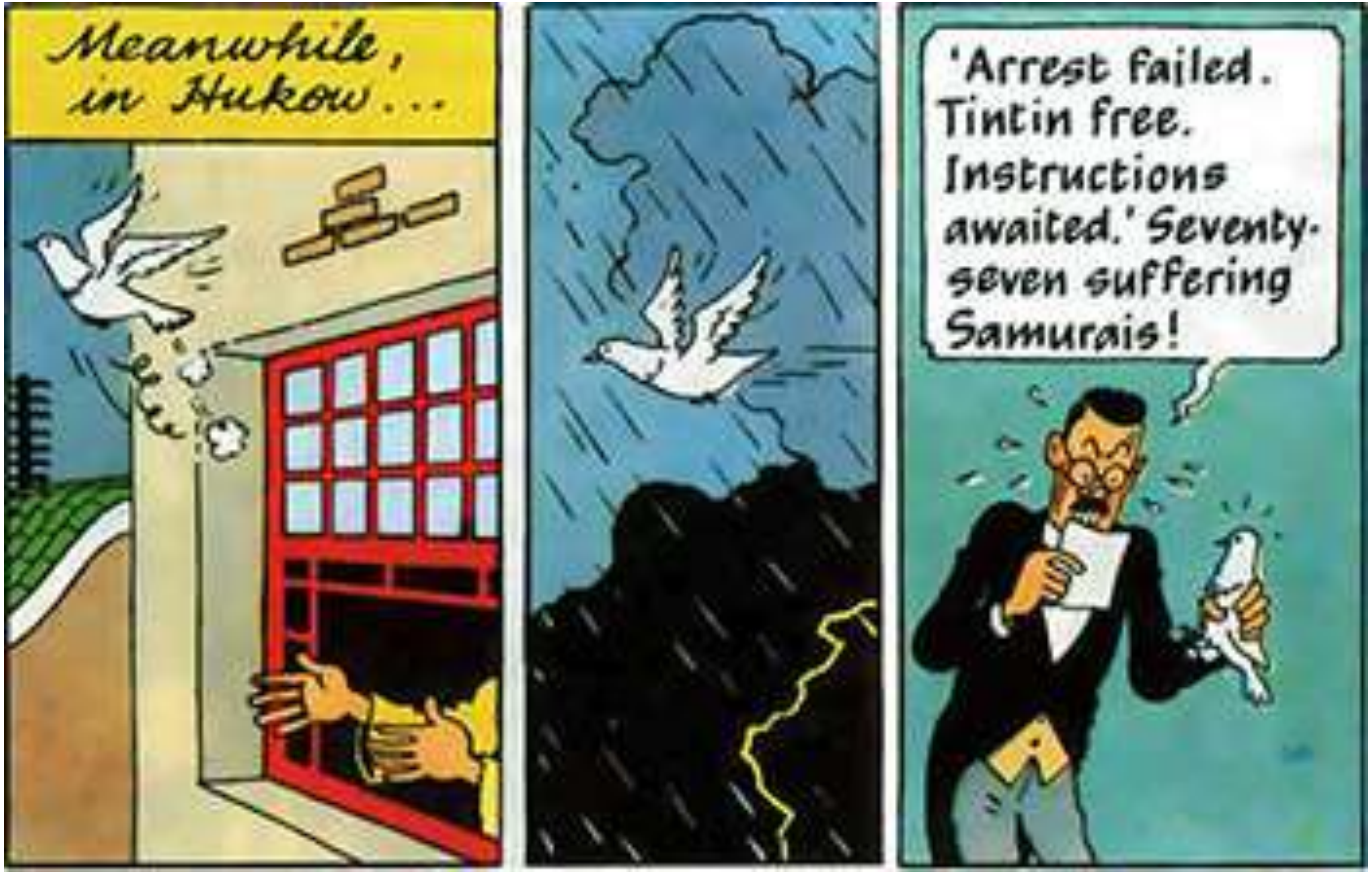
Next…
Delivery Semantics

Do we care
if the message arrives?

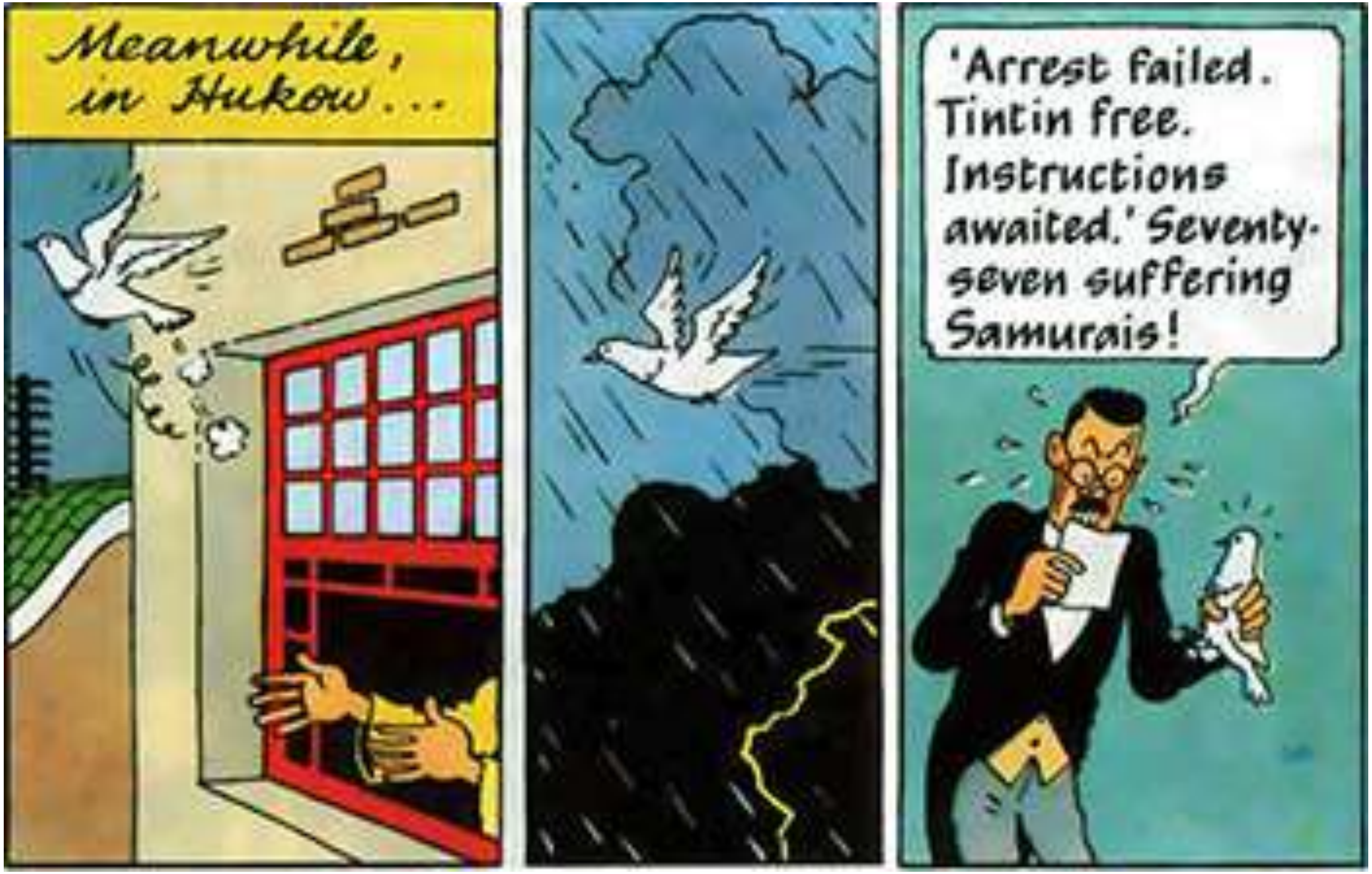
Yes! Guaranteed delivery is desirable


Meanwhile, in Hukow...
'Arrest failed. Tintin free. Instructions awaited.' Seventy-seven suffering Samurais!

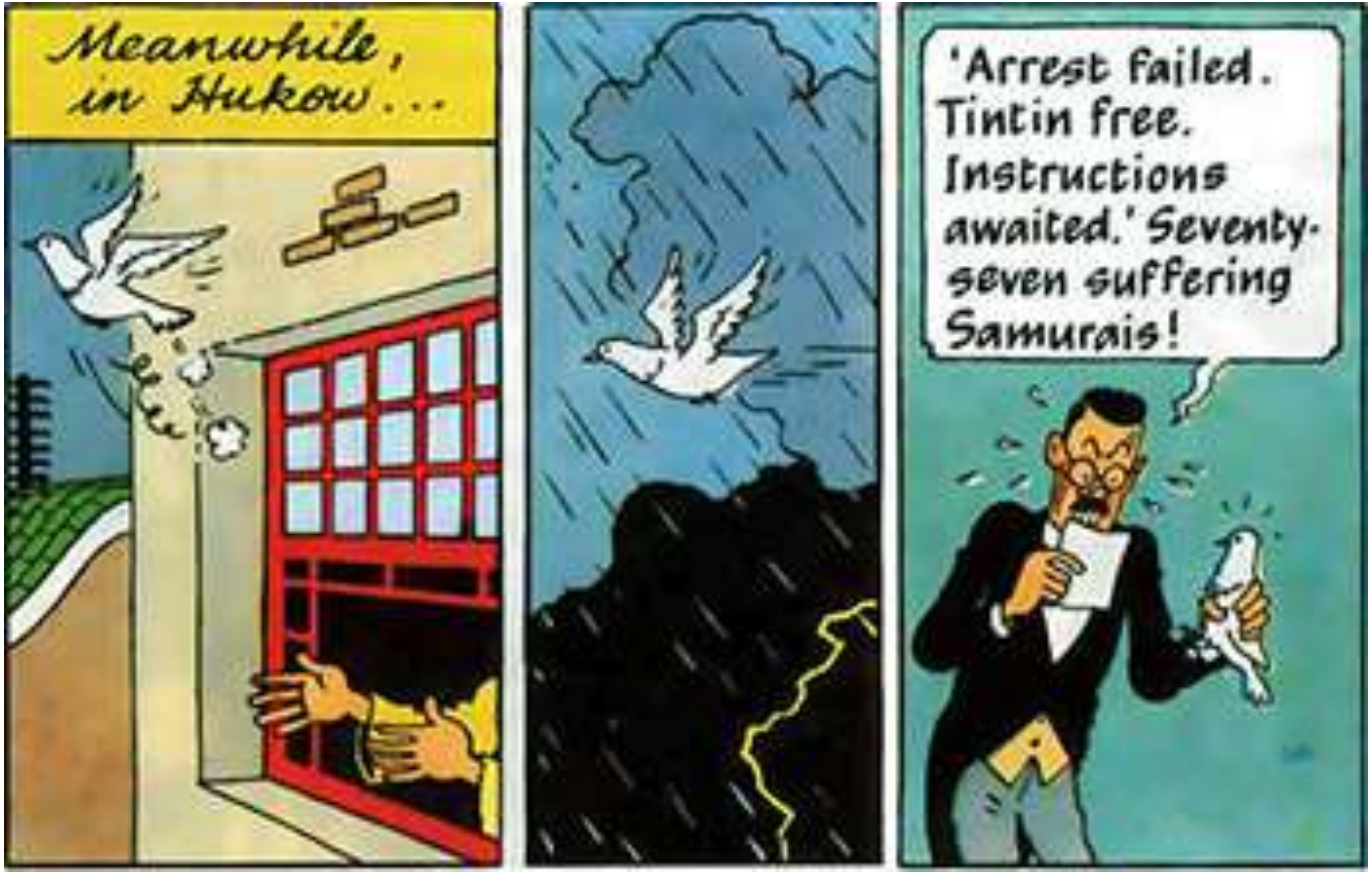
But homing pigeons got lost or eaten


Meanwhile, in Hukow...
'Arrest failed. Tintin free. Instructions awaited.' Seventy-seven suffering Samurais!

Send multiple pigeons


Meanwhile, in Hukow...
'Arrest failed. Tintin free. Instructions awaited.' Seventy-seven suffering Samurais!

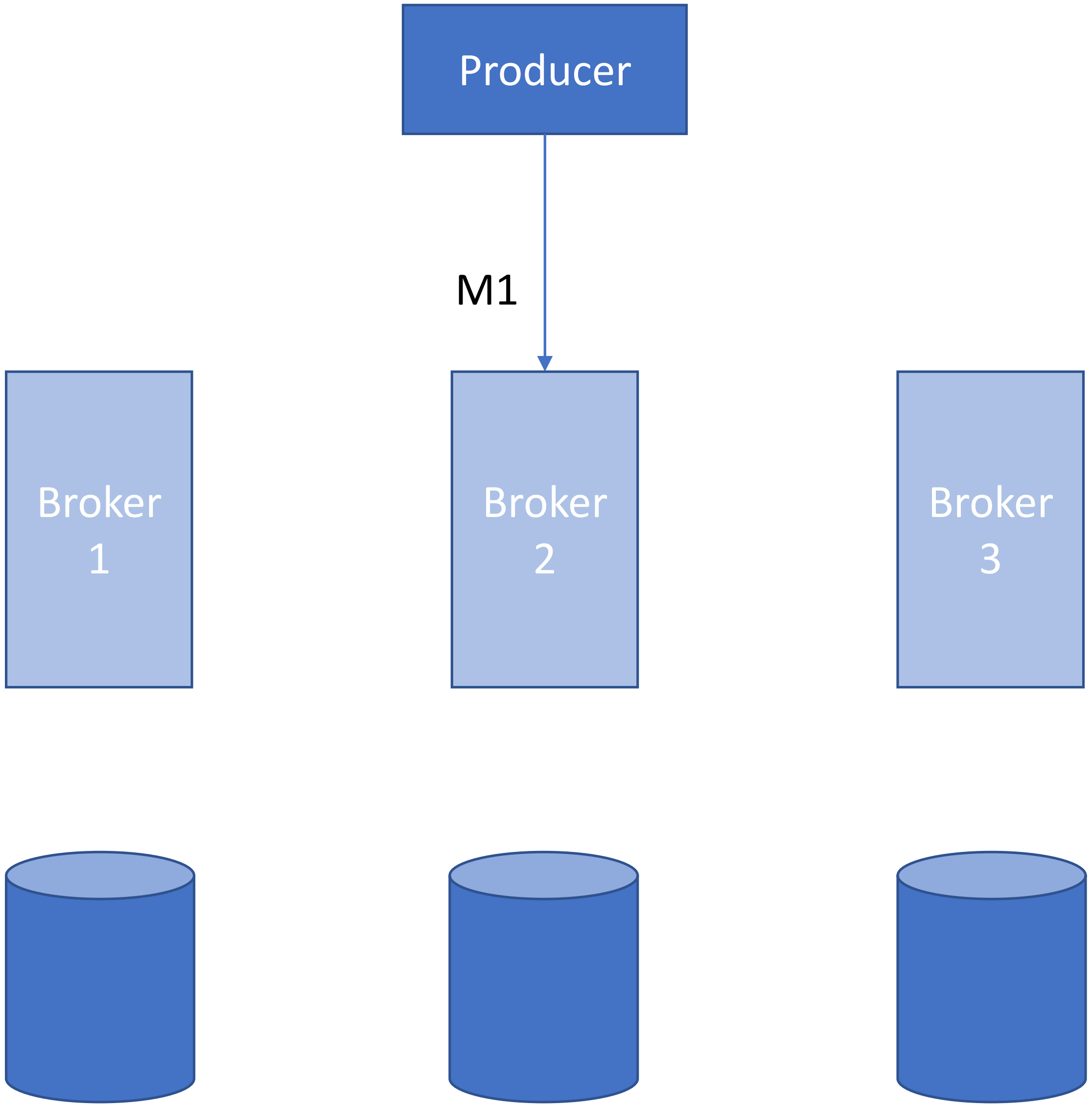
One pigeon may make it


Meanwhile, in Hukow...
'Arrest failed. Tintin free. Instructions awaited.' Seventy-seven suffering Samurais!

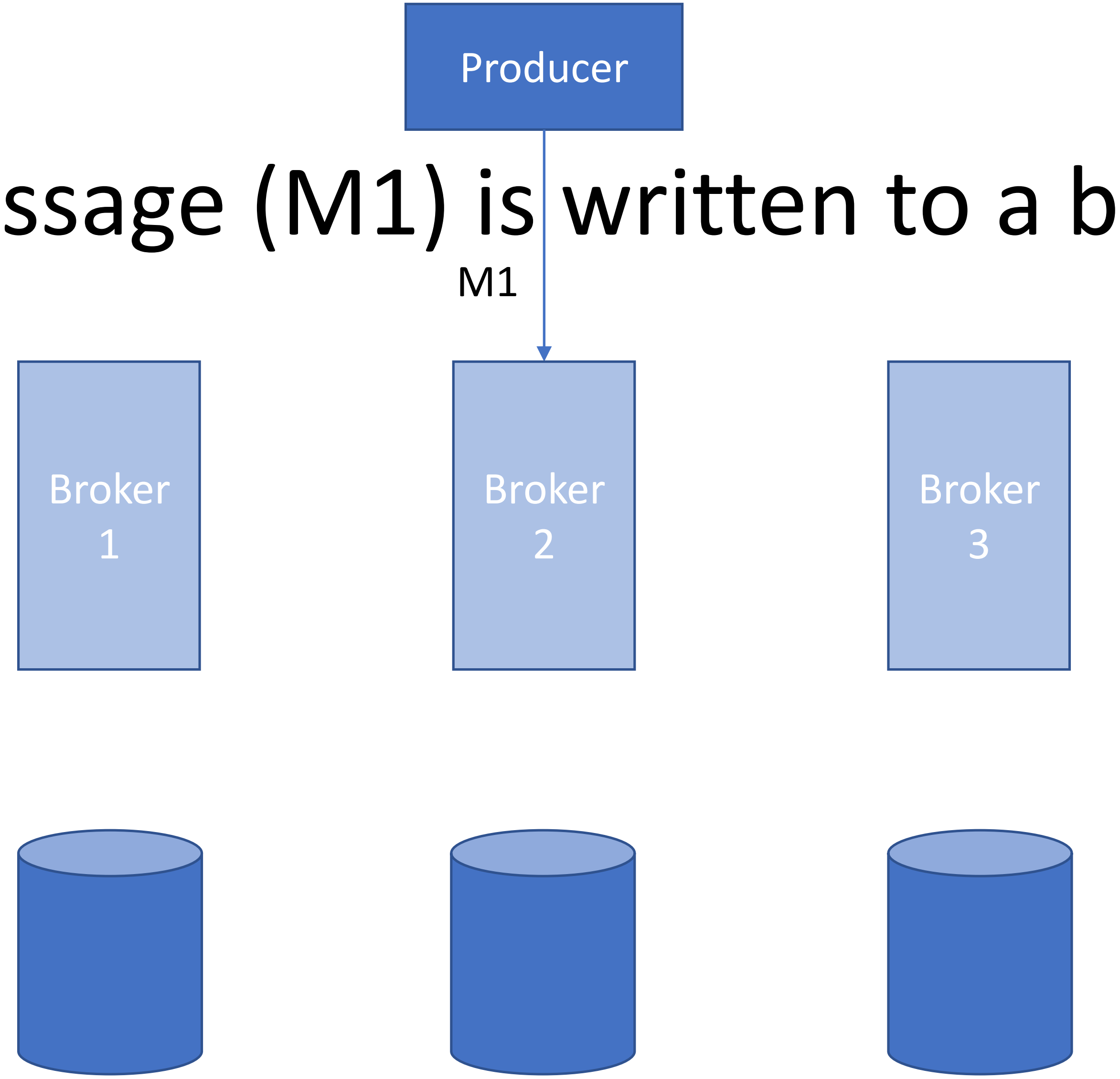
Lost

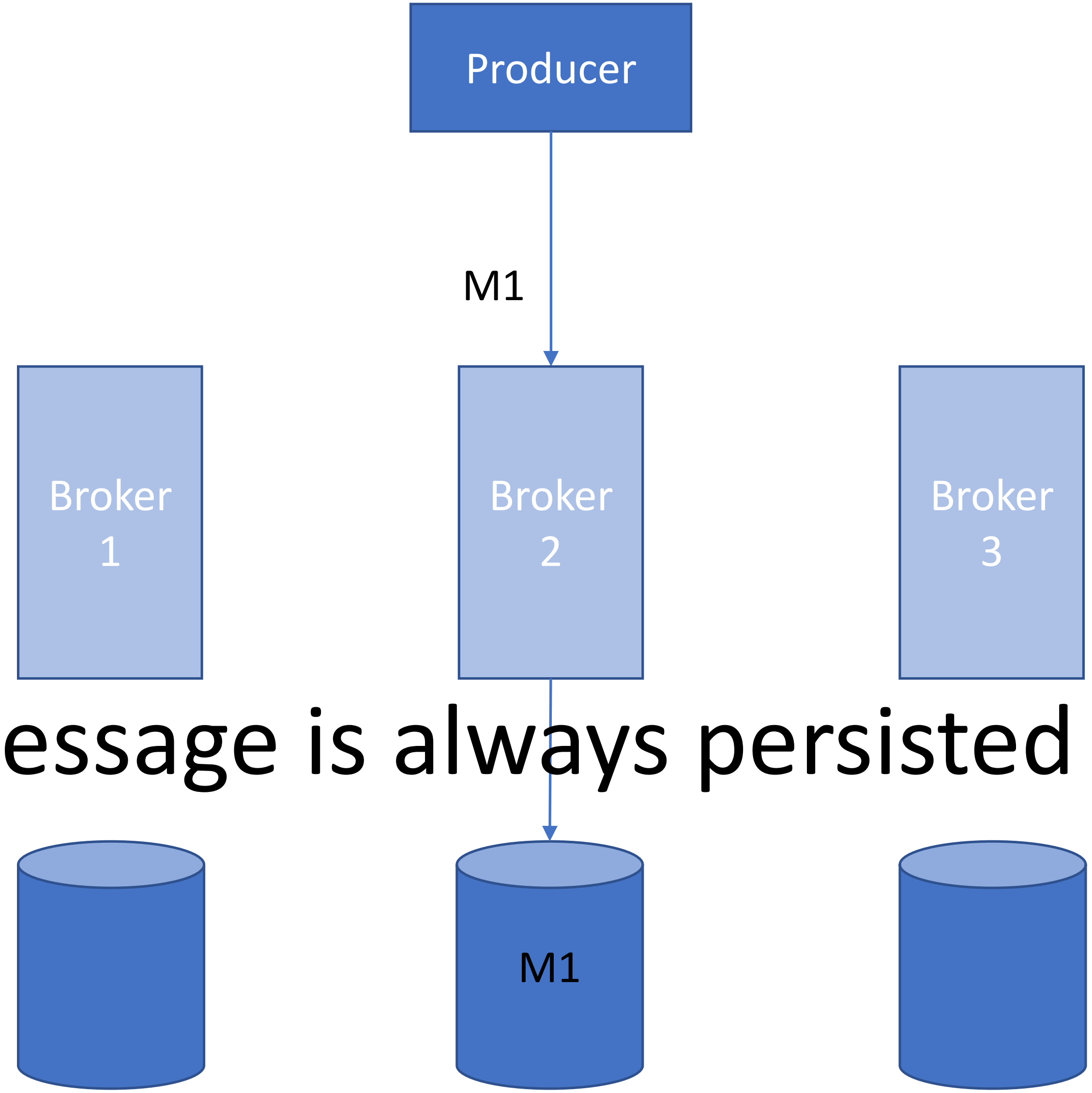
Eaten

How does Kafka guarantee delivery?



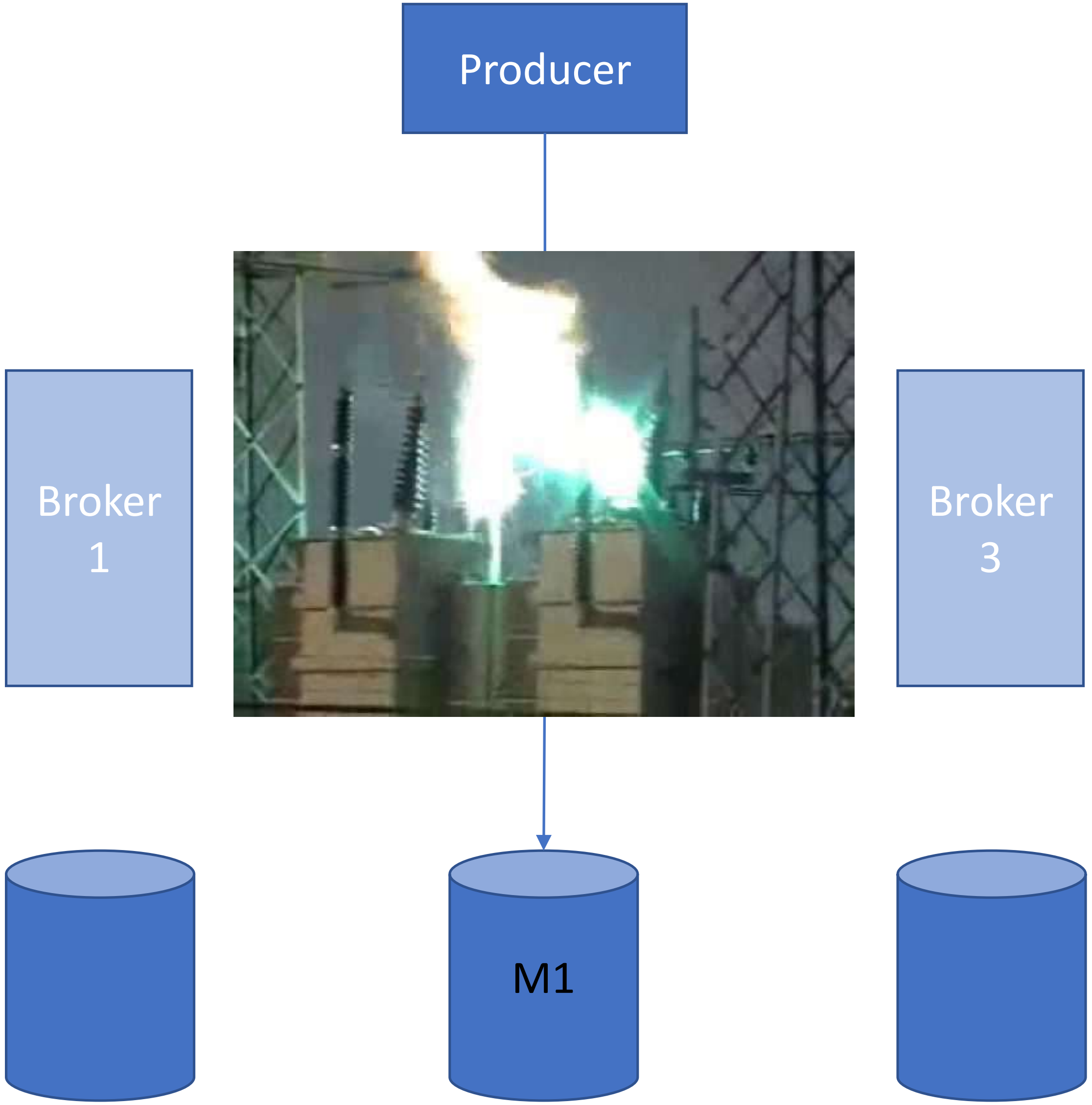
A Message (M1) is written to a broker (2)



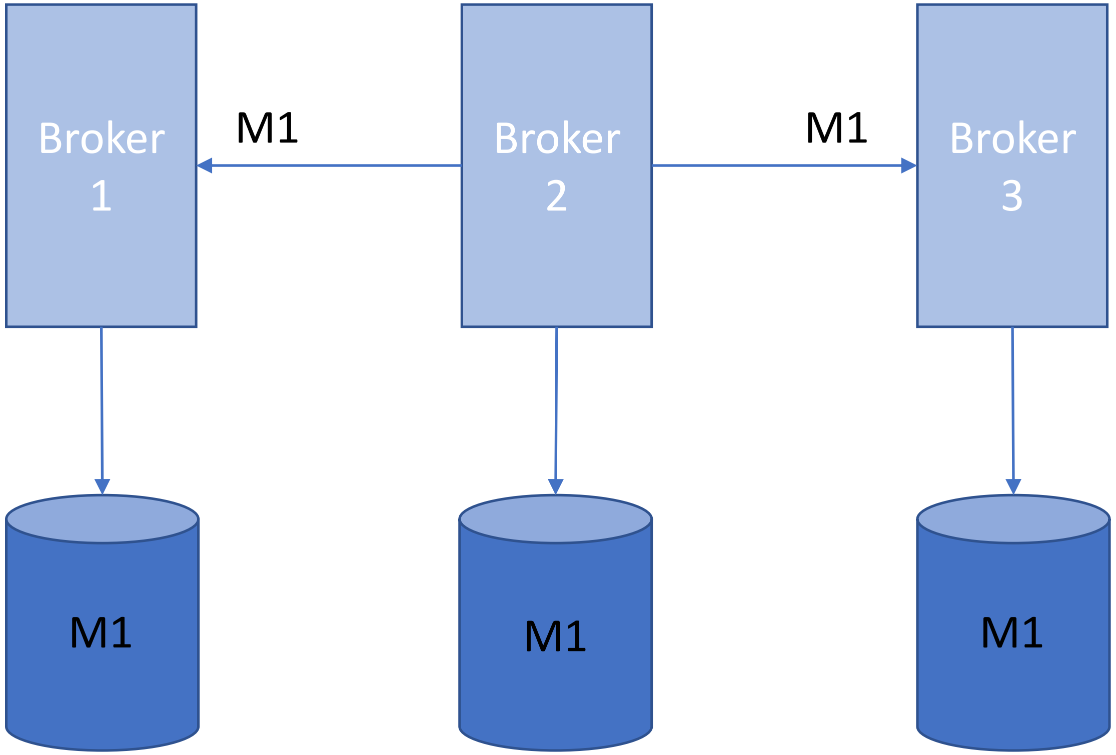


The message is always persisted to disk.

This makes it resilient to loss of power.

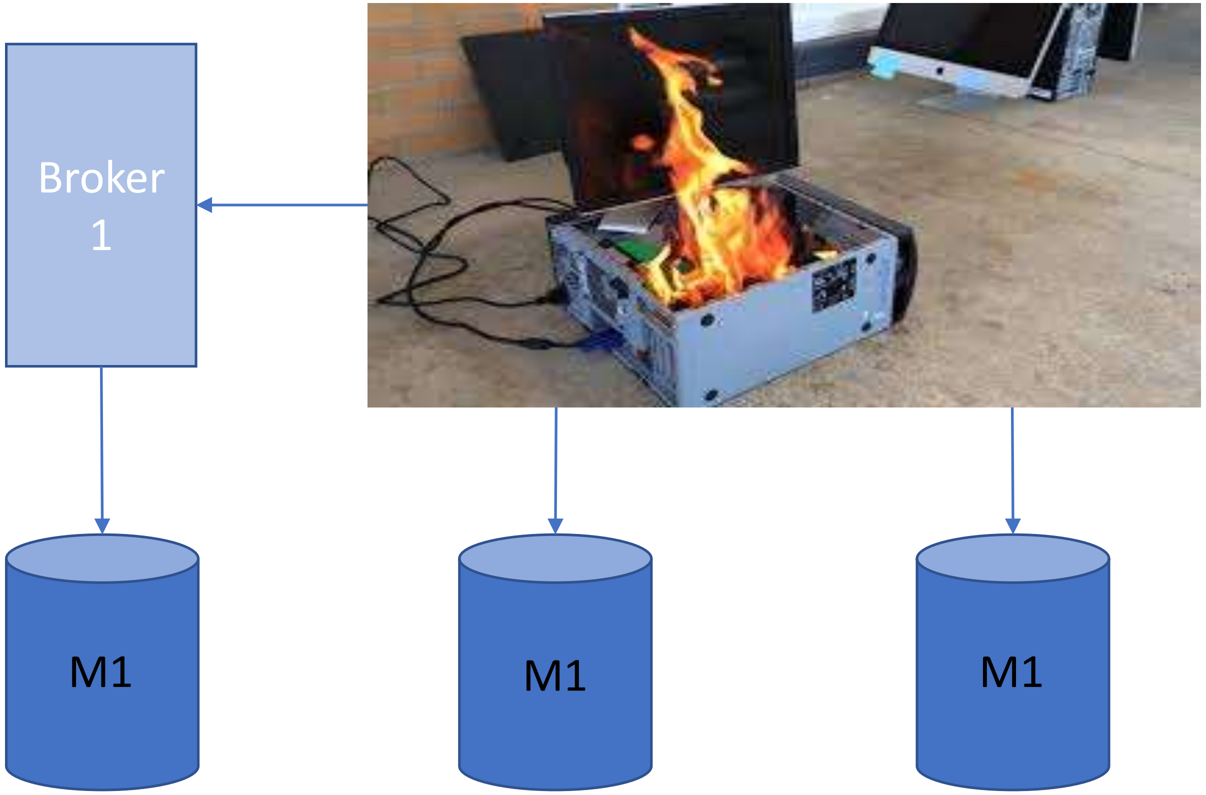


The message is also replicated on multiple “brokers”

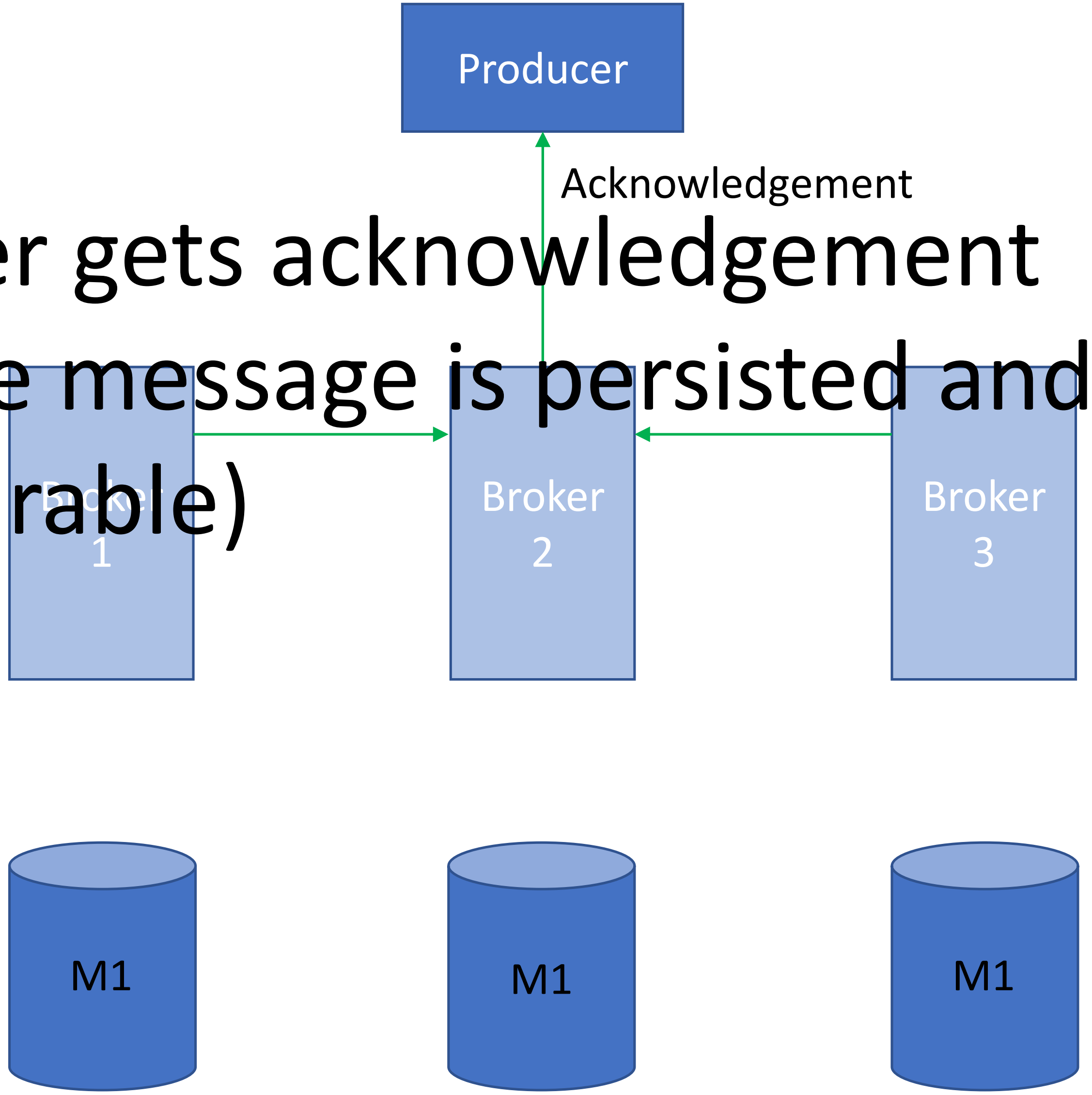


Producer

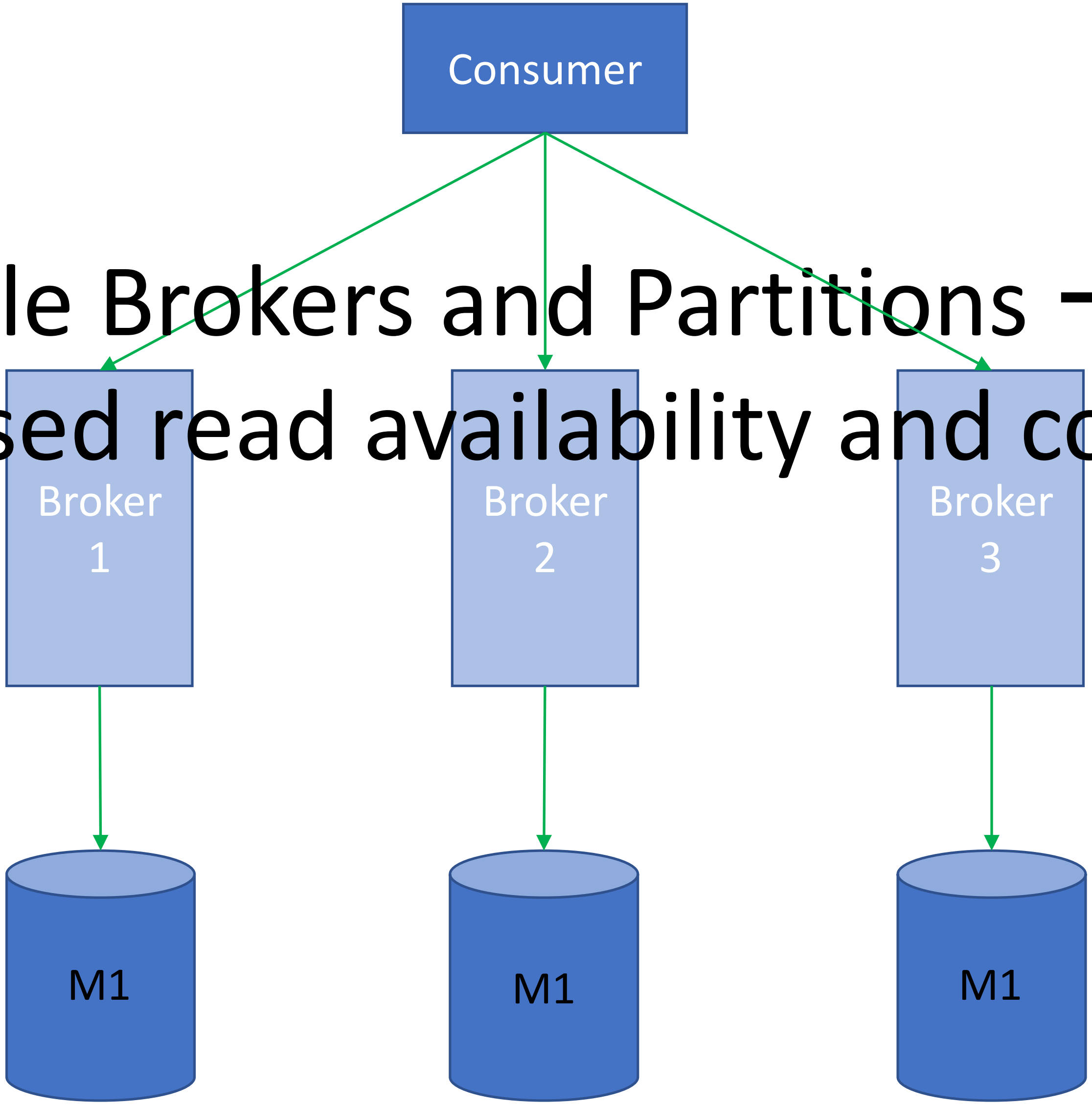
Which makes it resilient to loss of most servers



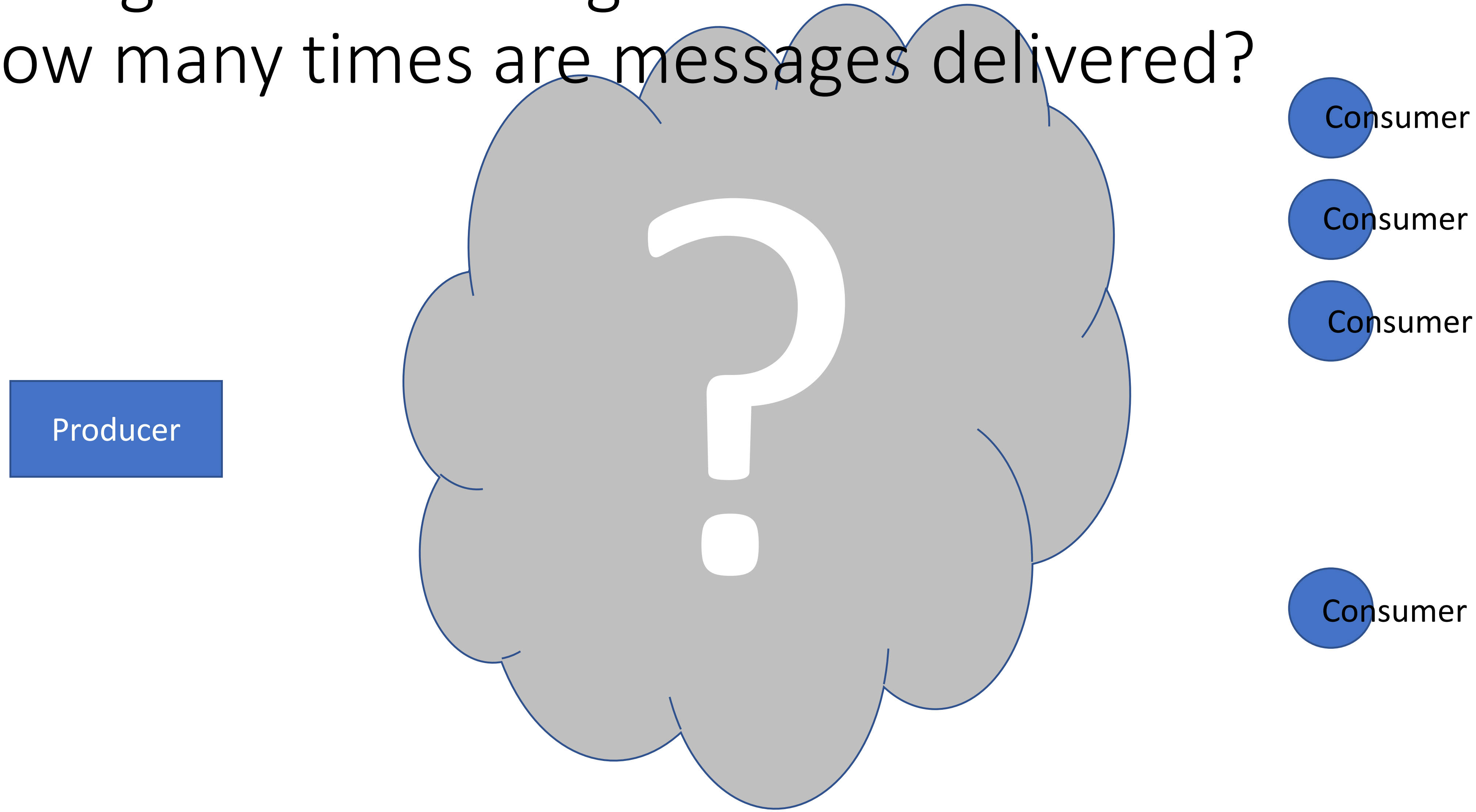
Producer gets acknowledgement once the message is persisted and replicated (configurable)



Multiple Brokers and Partitions → increased read availability and concurrency

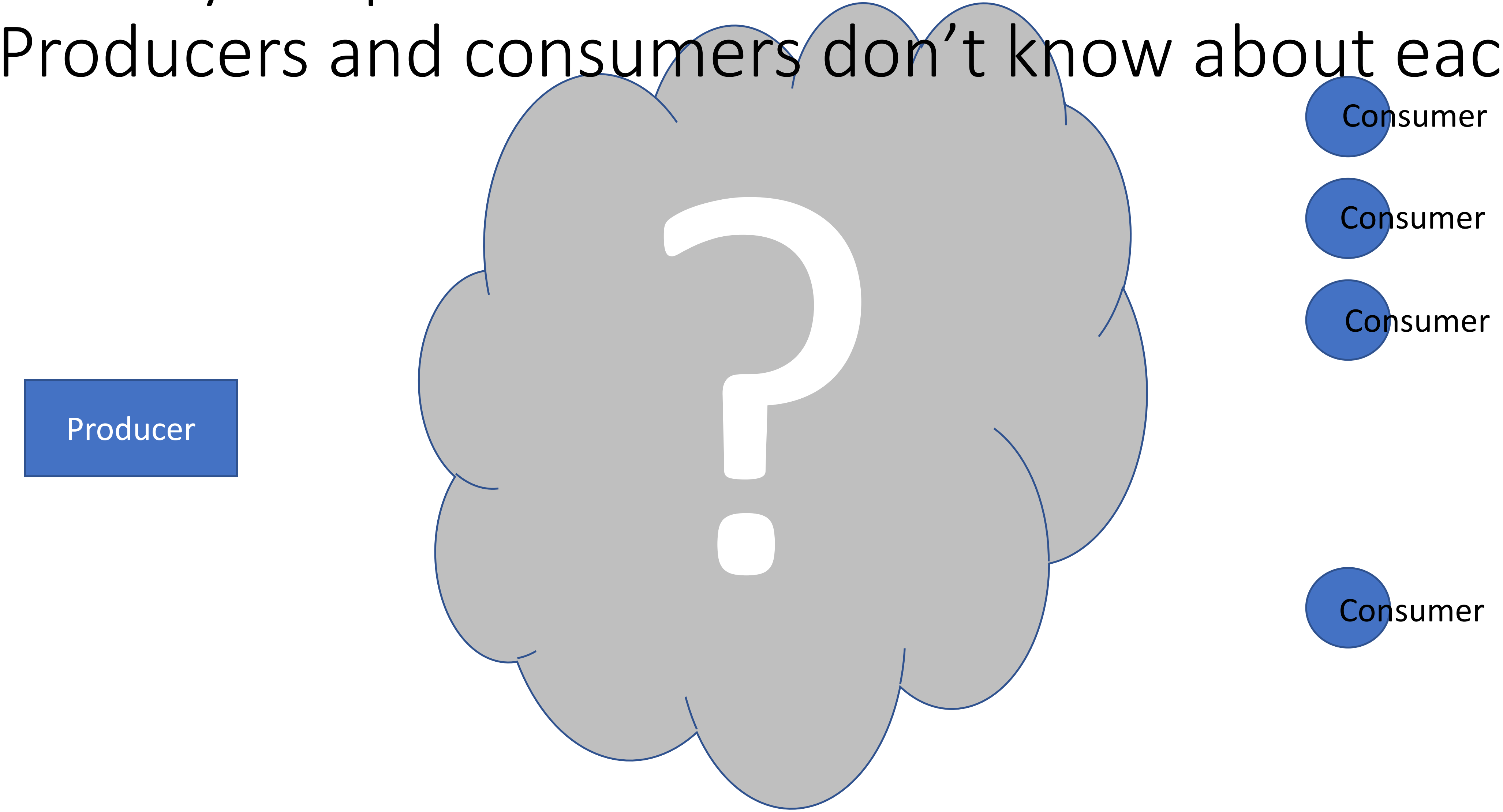


The 2nd aspect of delivery semantics:
Who gets the messages?
How many times are messages delivered?

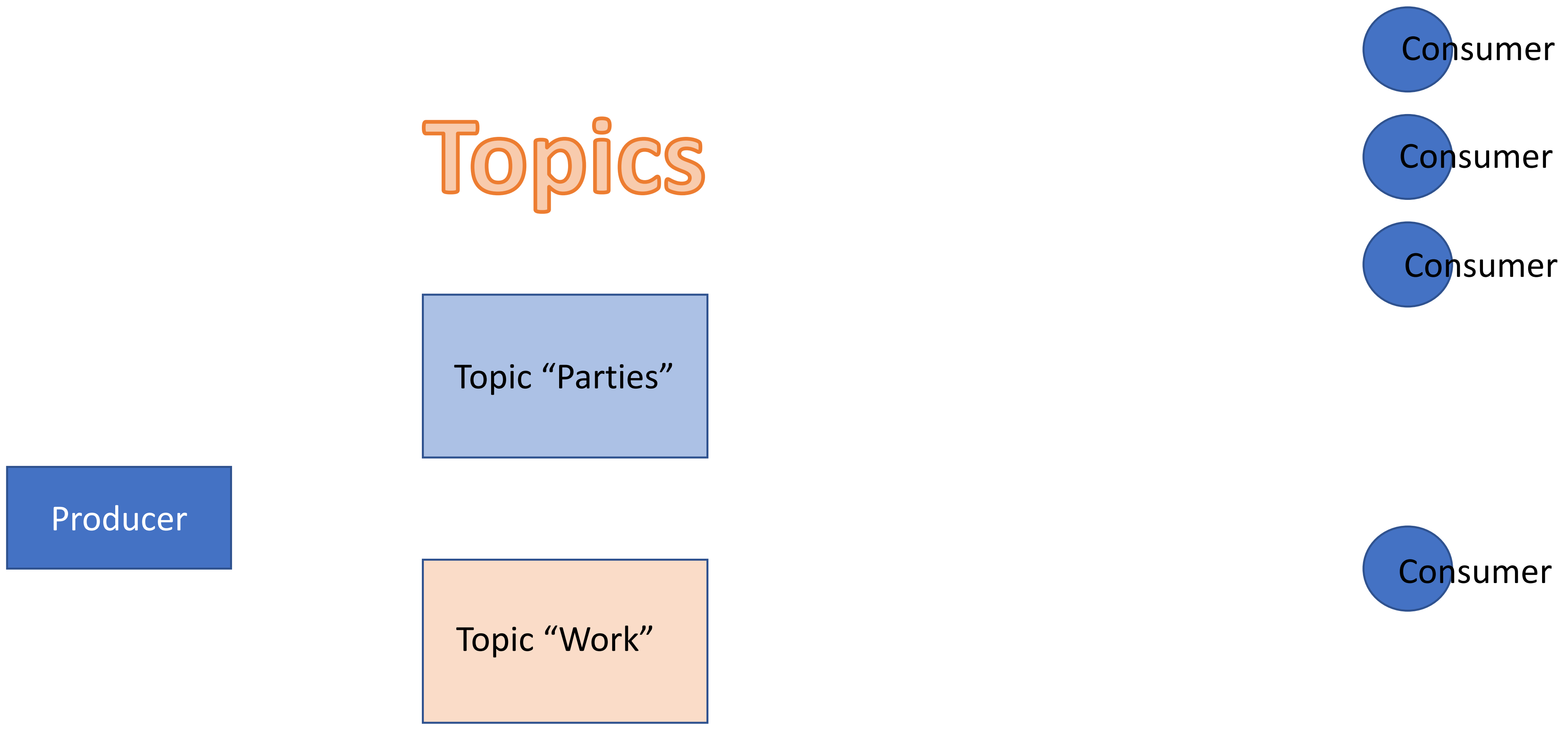


Delivery Semantics - Kafka is “pub-sub”

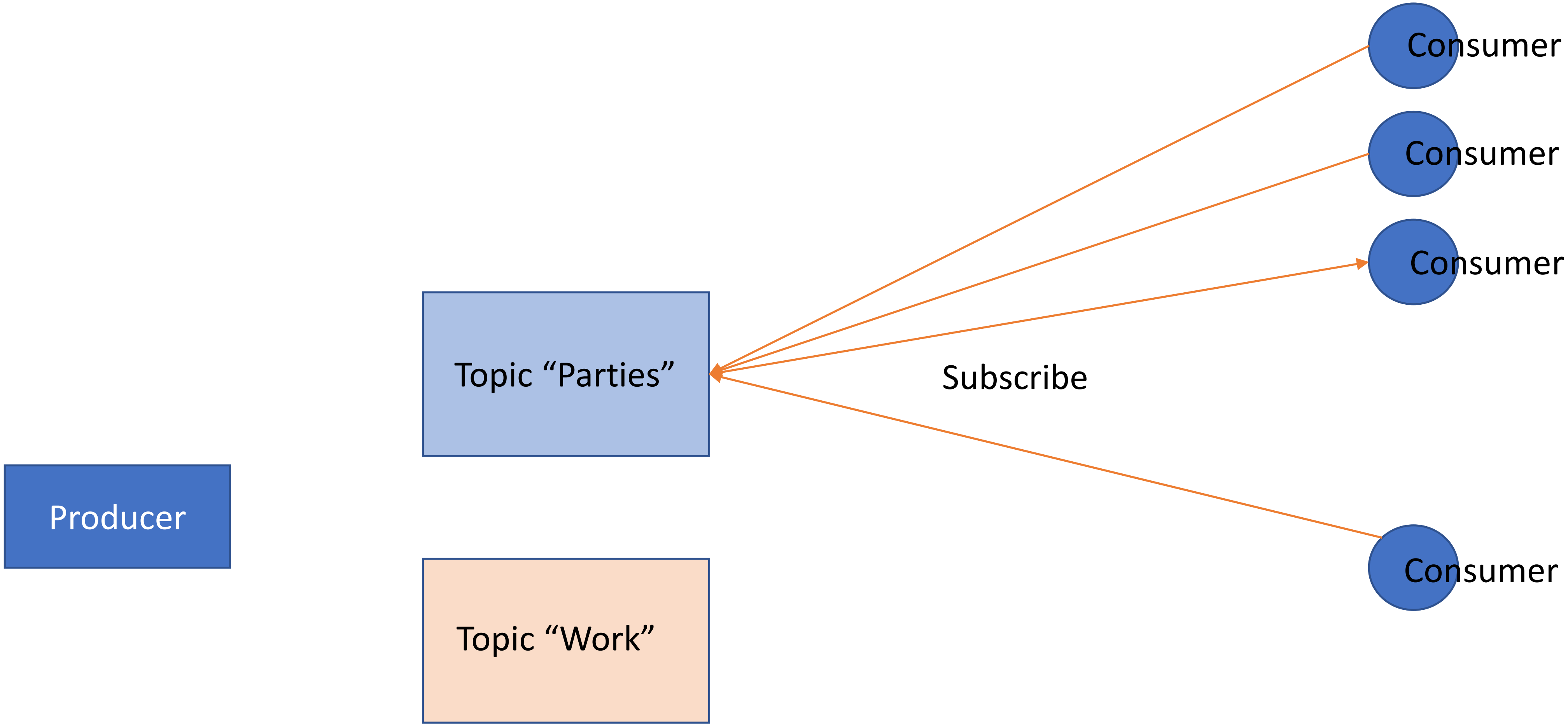
- Loosely coupled
- Producers and consumers don’t know about each other



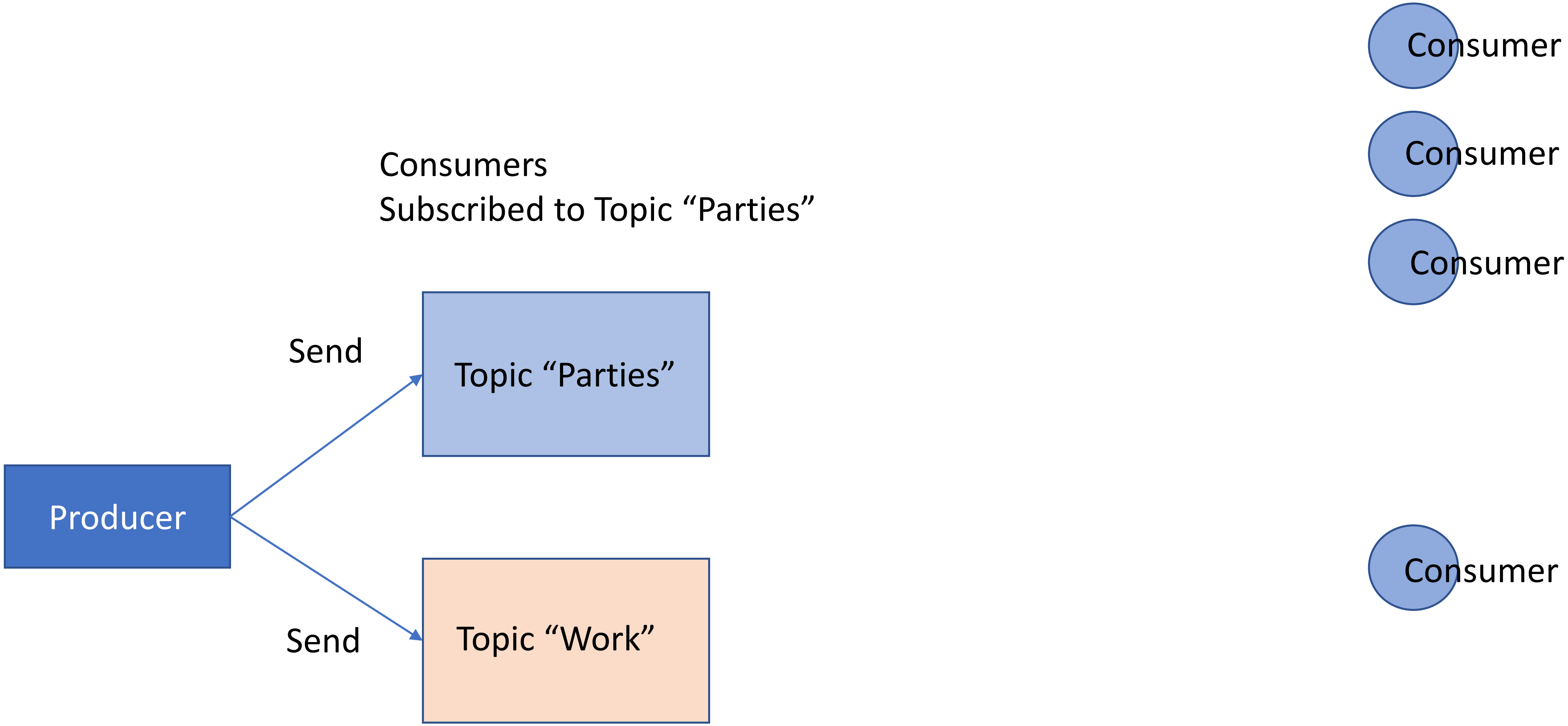
Which consumers get which messages (filtering), is topic based



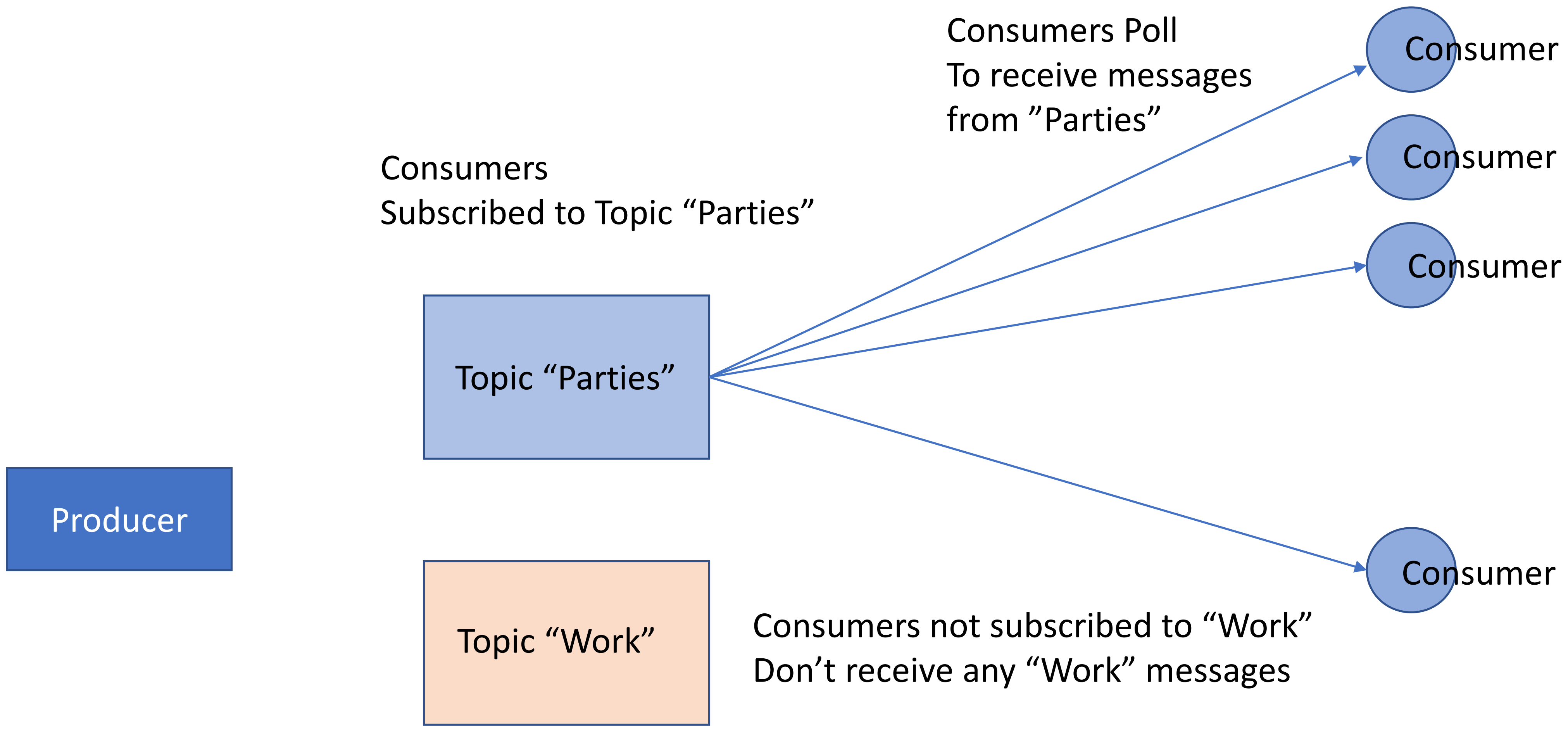
Consumers Subscribe to topic “Parties”



Publishers send messages to topics


Consumers
Subscribed to Topic “Parties”

Consumers only receive messages from subscribed topics



Consumers Poll
To receive messages
from ”Parties”

Consumer

Consumer

Consumer

Consumers
Subscribed to Topic “Parties”

Topic “Parties”

Producer

Consumer

Topic “Work”

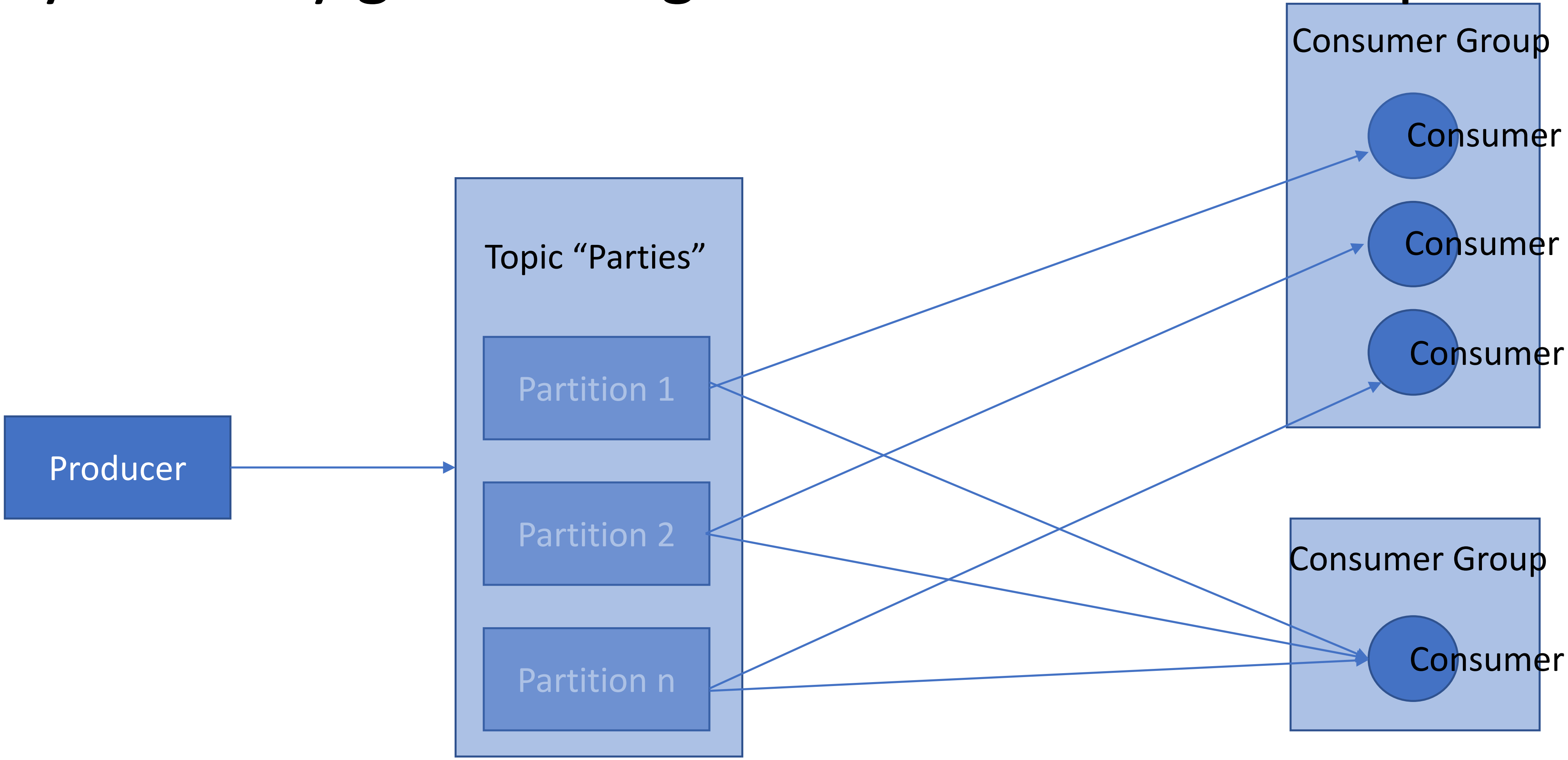
Consumers not subscribed to “Work”
Don’t receive any “Work” messages

Partitions and Consumer Groups Enable sharing of work across consumers

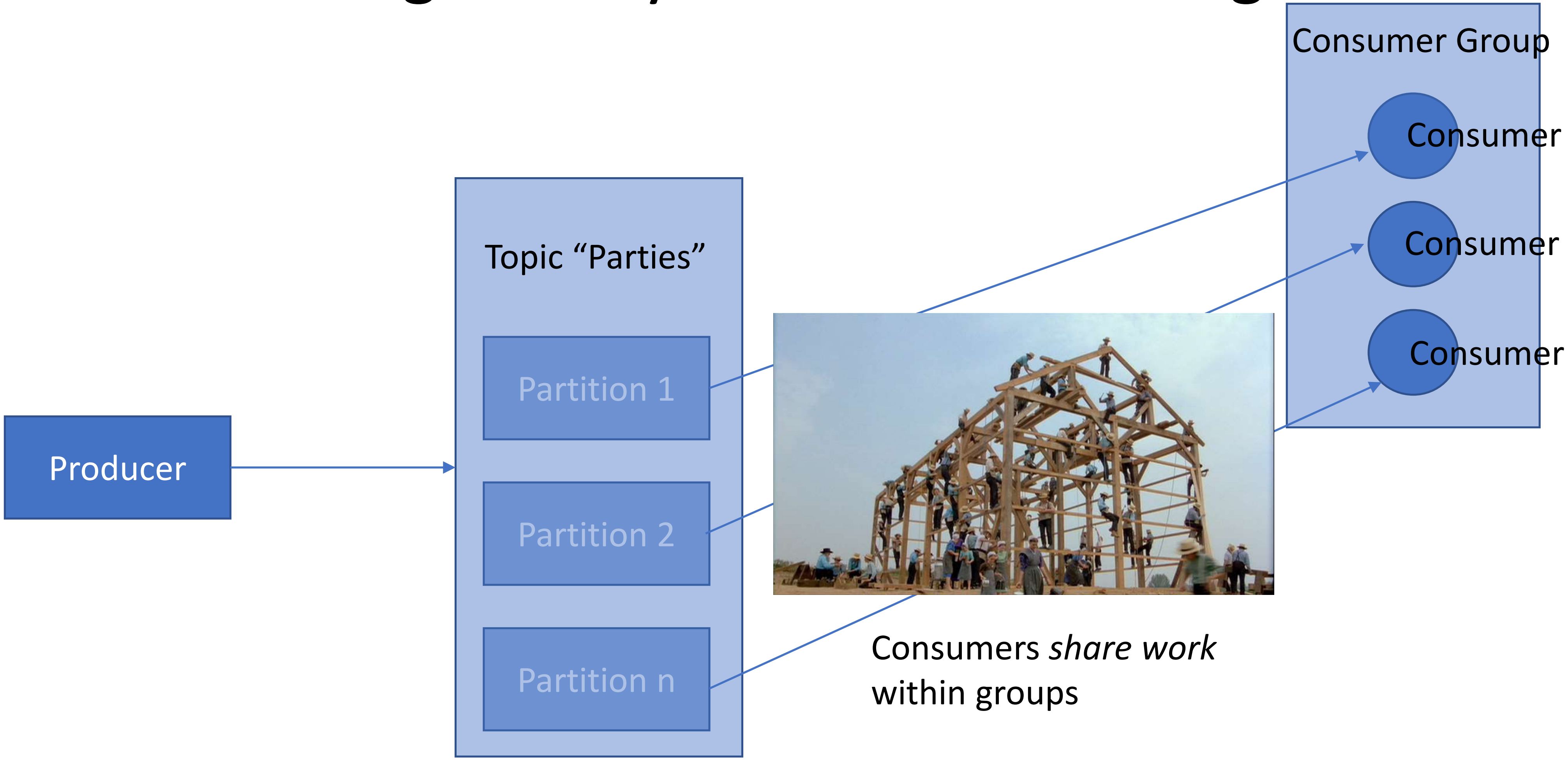
Duplicate message delivery

Each message is delivered to each subscribed consumer group

Consumers subscribed to topic are allocated partitions They will only get messages from their allocated partitions.



Consumers in the same group share the work around
Each consumer gets only a subset of messages



Consumer Group

Consumer

Consumer

Consumer

Topic “Parties”

Partition 1

Partition 2

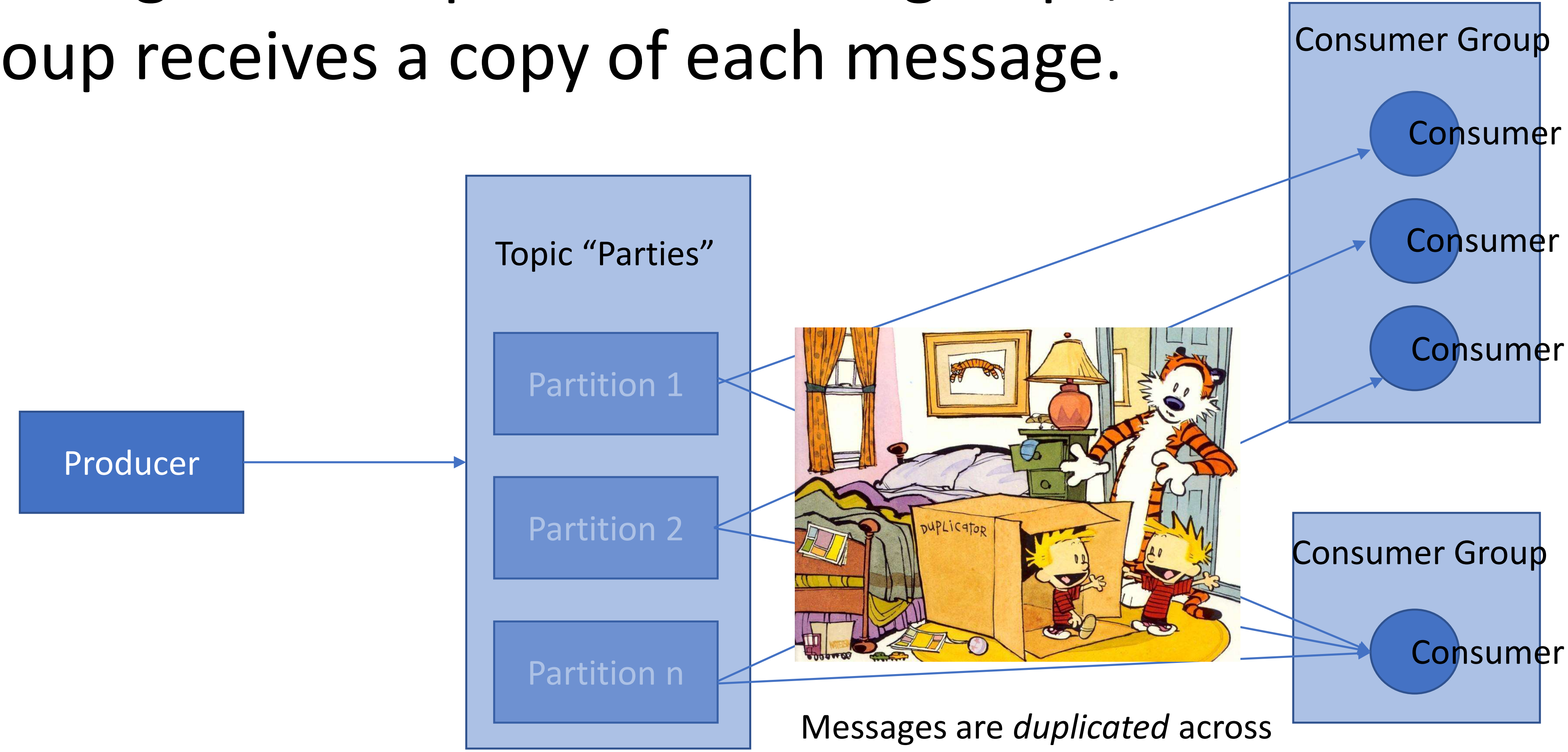
Partition n

Producer

Consumers *share work* within groups

Multiple groups enable message broadcasting

Messages are duplicated across groups, each consumer group receives a copy of each message.



Messages are *duplicated* across Consumer groups

Key

Partition based delivery

Which messages are delivered to which consumers?

If a message has a key, then Kafka uses Partition based delivery.

Messages with the same key are always sent to the same partition and therefore the same consumer.

And the order is guaranteed.

If the key is null, then Kafka uses round robin delivery

Each message is delivered to the next partition

No Key

Round robin delivery

Time for an Example, with 2 consumer groups.

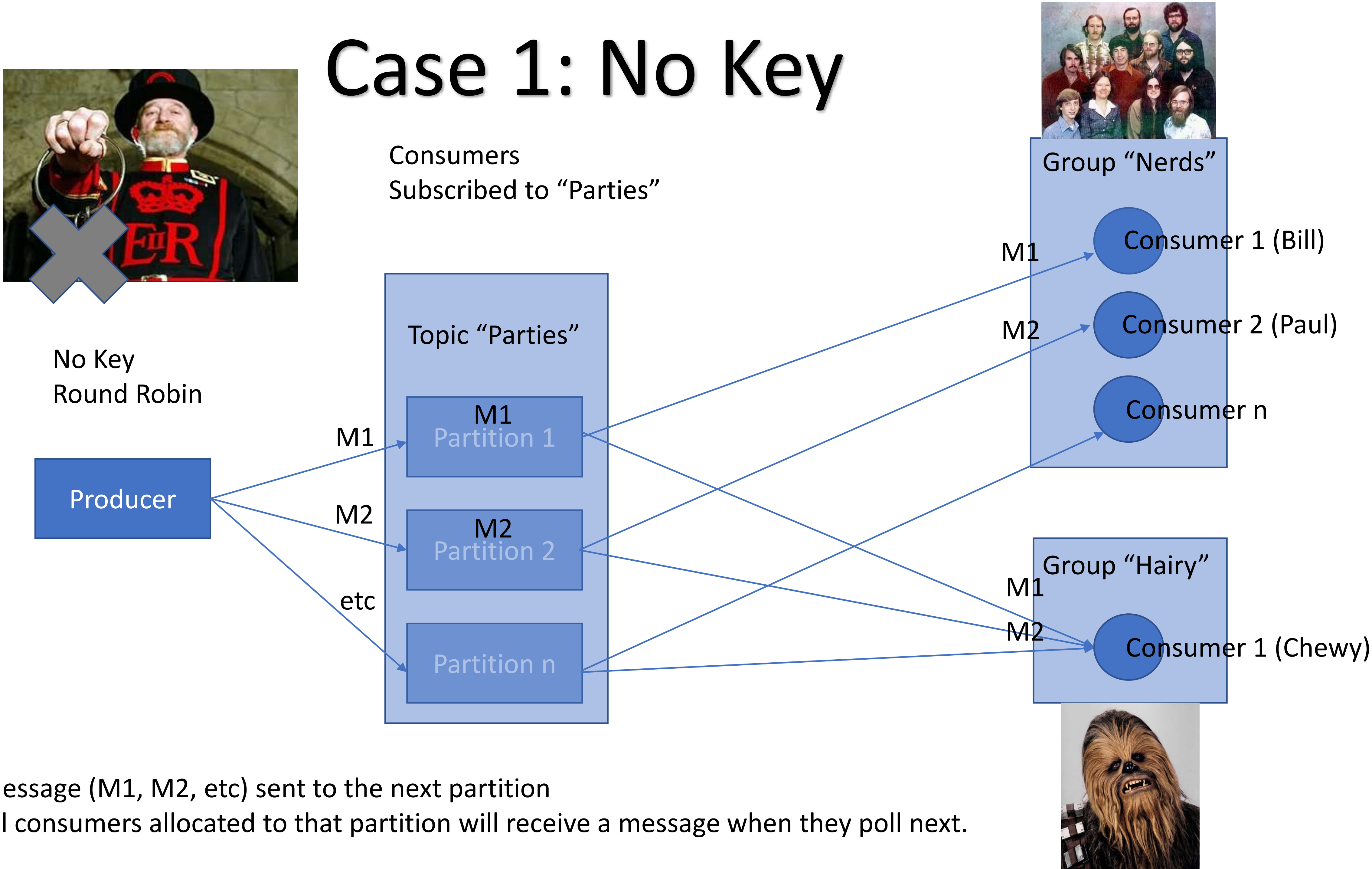
Consumer Group = Nerds

Multiple consumers

Consumer Group = Nerds
Multiple consumers

Consumer Group = Hairy
Single consumer

Case 1: No Key



Consumers
Subscribed to “Parties”

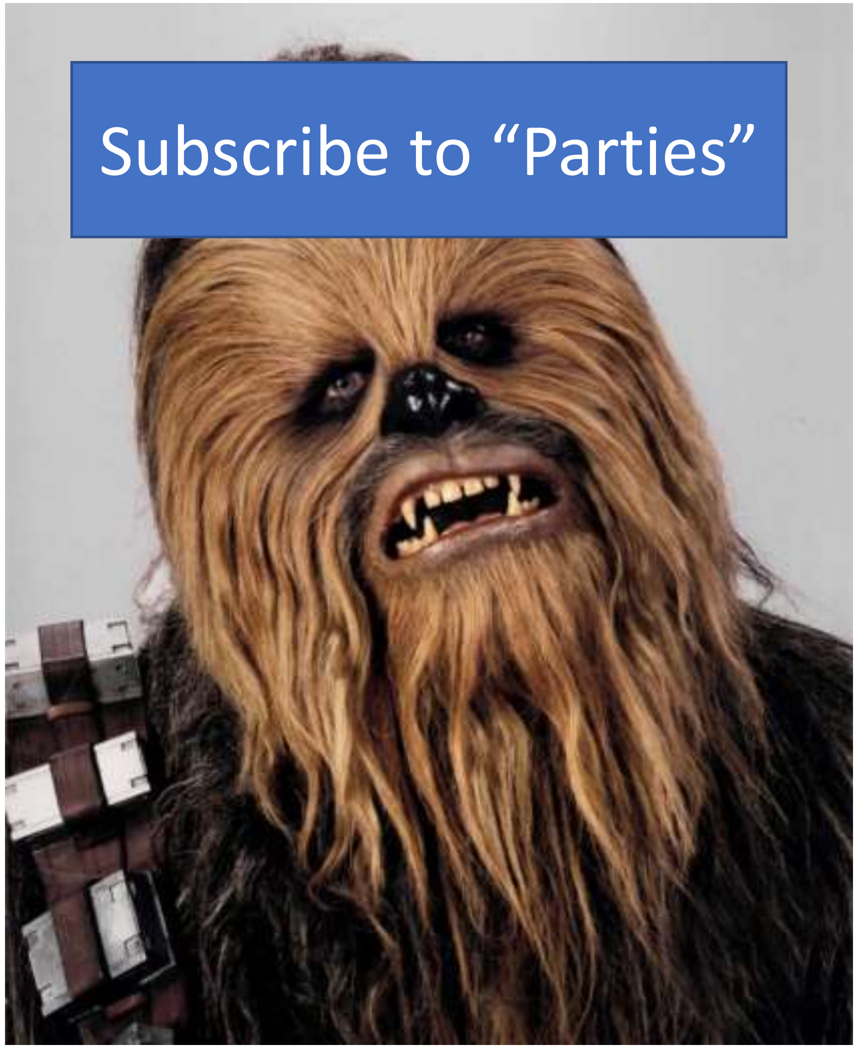
No Key
Round Robin

Message (M1, M2, etc) sent to the next partition
All consumers allocated to that partition will receive a message when they poll next.

Here’s what happens (not showing producer or topics, have to imagine them)

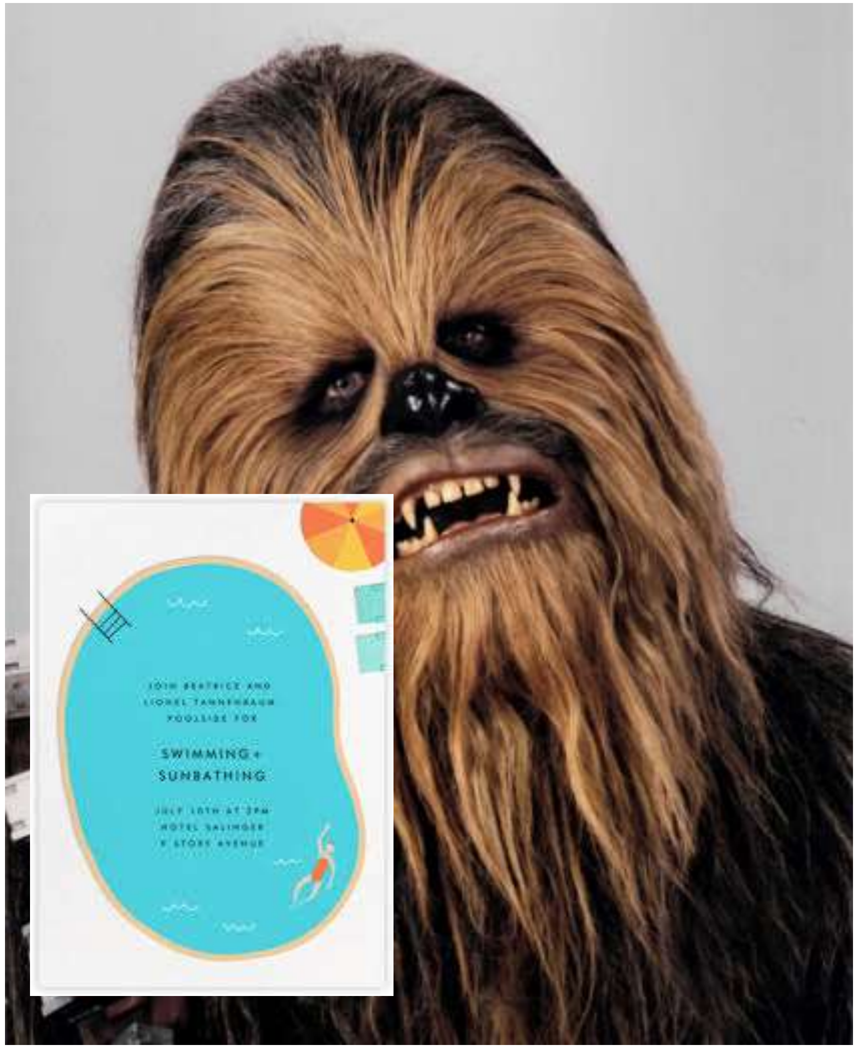
1. Both Groups subscribe to Topic “parties” (11 partitions, so 1 consumer per partition).

Subscribe to “Parties”


Subscribe to “Parties”

1. Both Groups subscribe to Topic “parties” (11 partitions, so 1 consumer per partition).
2. Producer sends record “Cool pool party – Invitation”
   <key=null, value=“Cool pool party - Invitation”> to “parties” topic (no key)

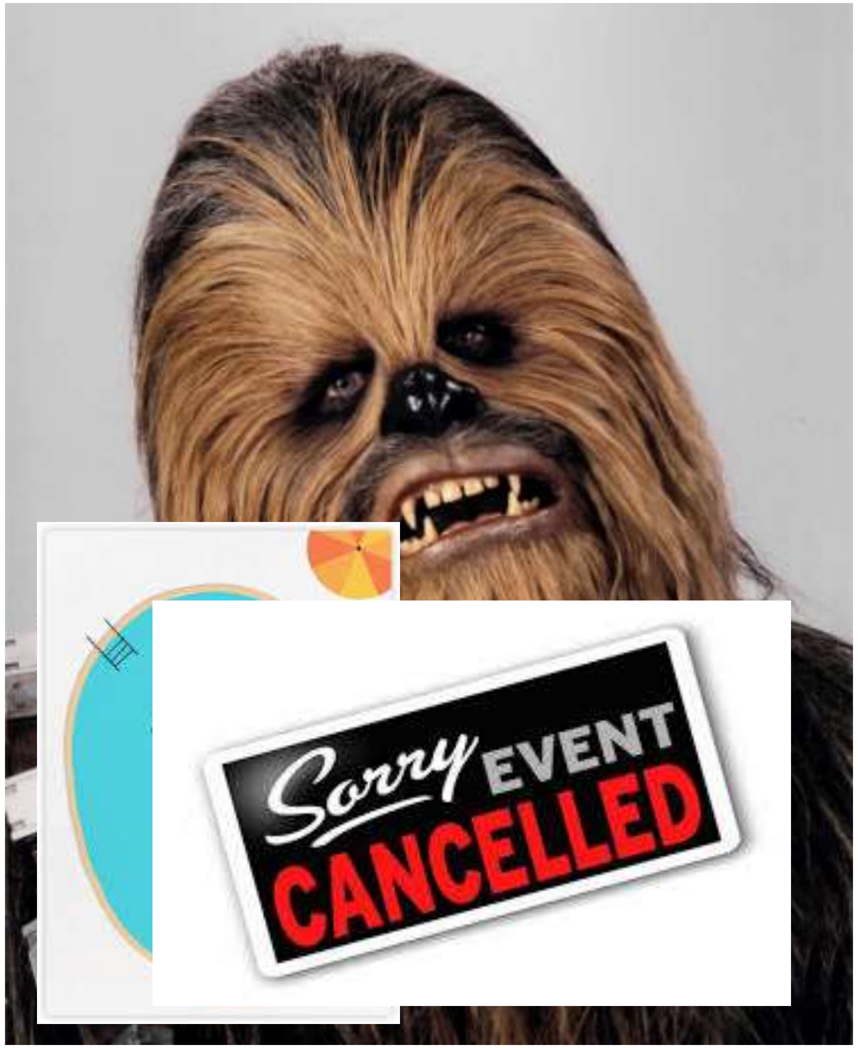
1. Both Groups subscribe to Topic “parties” (11 partitions, so 1 consumer per partition).
2. Producer sends record “Cool pool party - Invitation”> to “parties” topic
3. Bill and Chewbacca receive a copy of the invitation and plan to attend



4. Producer sends another record “Cool pool party – Cancelled” <key=null, value=“Cool pool party - Cancelled”> to “parties” topic

4. Producer sends another record <key=null, value=“Cool pool party - Cancelled”> to “parties” topic
5. Paul and Chewbacca receive the cancellation.

Paul gets the message this time as it’s round robin, ignores it as he didn’t get the invitation. Bill wastes his time trying to go to cancelled party. The rest of the gang aren’t surprised at not receiving any party invites and stay at home to do some hacking. Chewy is only consumer in his group so gets all messages, plans something fun instead…

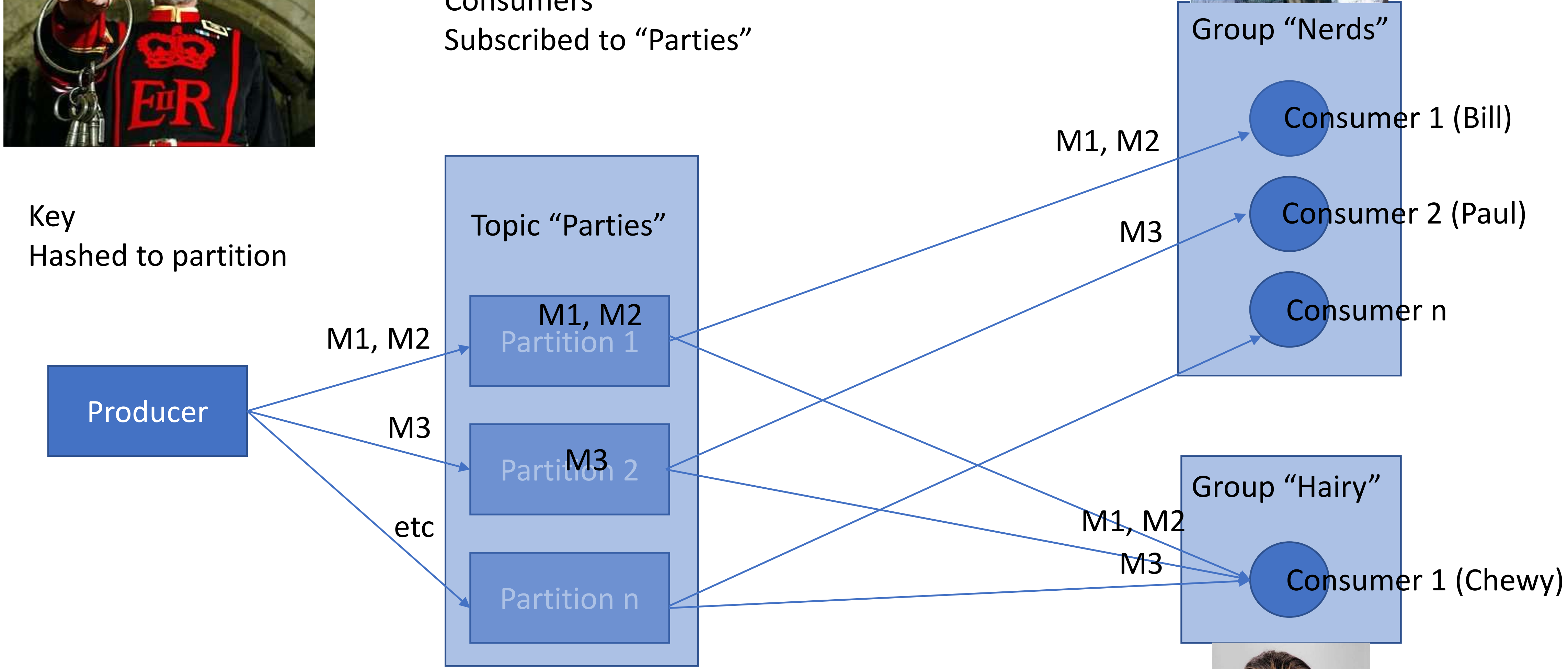


A visit to the hairdressers!

Case 2: If there is a Key

Consumers
Subscribed to “Parties”

Key
Hashed to partition



Producer

M1, M2

M3

etc

Topic “Parties”

M1, M2
Partition 1

M3
Partition 2

Partition n

M1, M2

M3

Group “Nerds”

Consumer 1 (Bill)

Consumer 2 (Paul)

Consumer n

M1, M2

M3

Group “Hairy”

Consumer 1 (Chewy)

A key is hashed to a partition, and a Message with that key is always sent to that partition.
Assume there are 3 messages, and messages 1 and 2 are hashed to same partition.

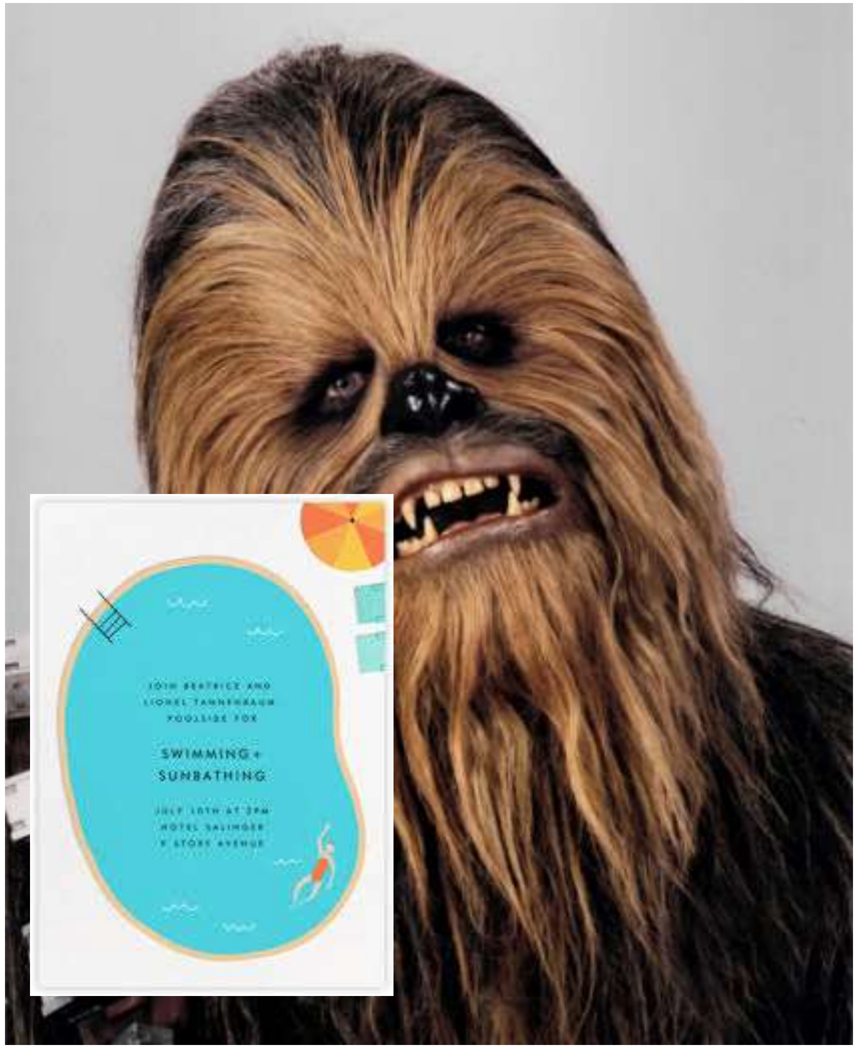
Here’s what happens with a key: key is “title” of the message (e.g. “Cool pool party”)

Same set up as before:

1. Both Groups subscribe to Topic “parties” (11 partitions).

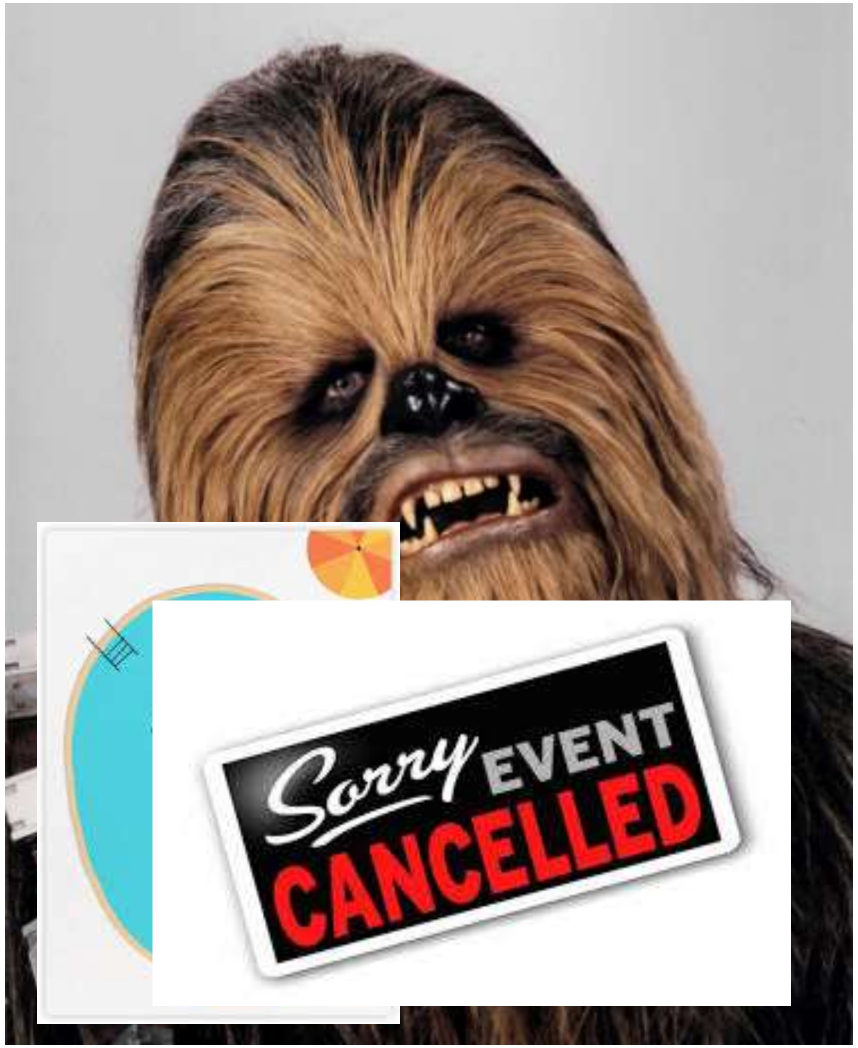
1. Both Groups subscribe to Topic “parties” (11 partitions).
2. Producer sends record <key=“Cool pool party”, value=“Invitation”> to “parties” topic

1. Both Groups subscribe to Topic “parties” (11 partitions).
2. Producer sends record <key=“Cool pool party”, value=“Invitation”> to “parties” topic
3. As before Bill and Chewbacca receive a copy of the invitation and plan to attend



4. Producer sends another record <key="Cool pool party", value="Cancelled"> to "parties" topic

4. Producer sends another record <key="Cool pool party", value="Cancelled"> to "parties" topic
5. Bill and Chewbacca receive the cancellation (same consumers this time, as identical key)



6. Producer sends another record <key=“Horrible Halloween party”, value=“Invitation”> to ”parties” topic

TRICK OR TREAT

HALLOWEEN

Party

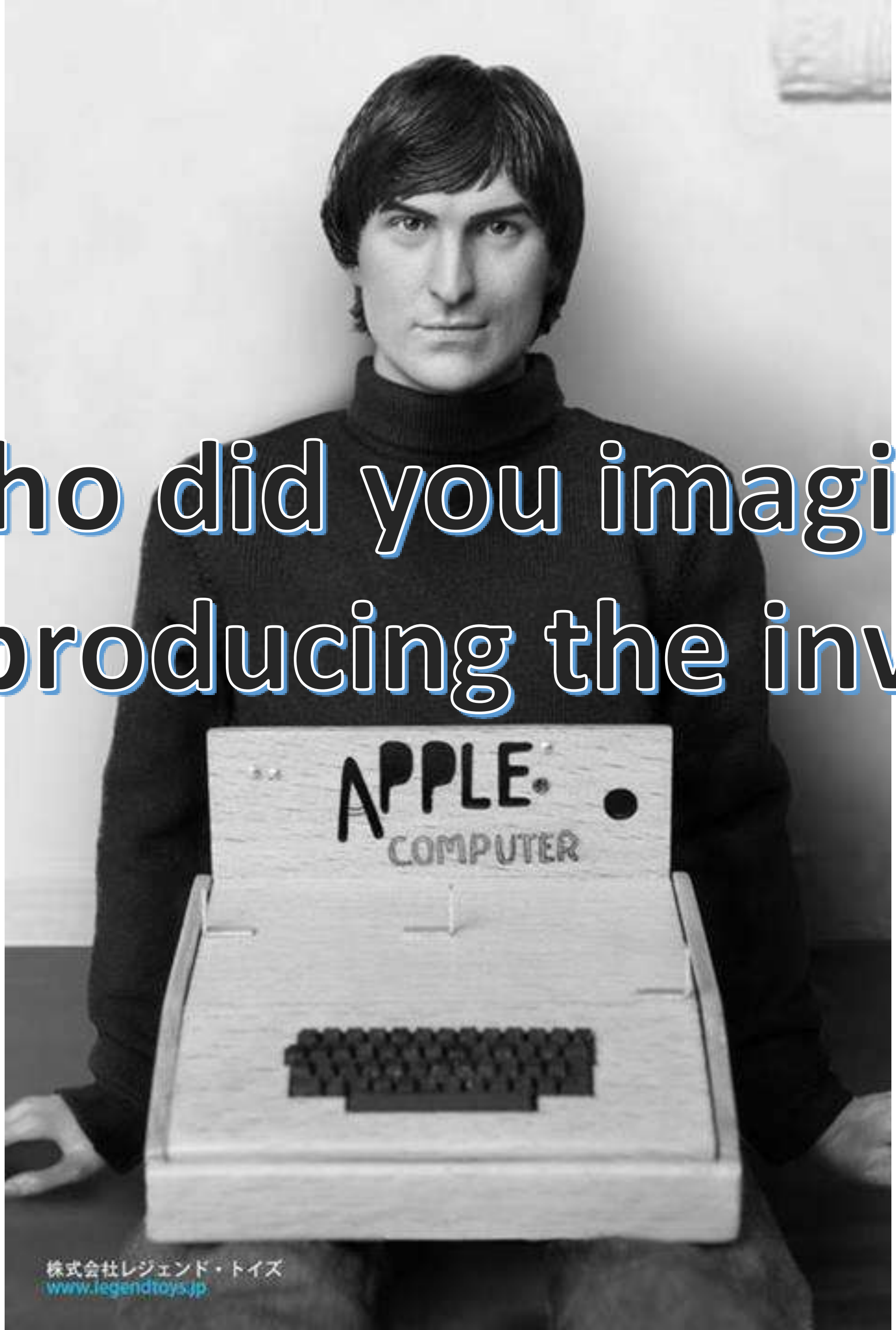
Friday, October 31st, 9 pm
Al and Courtneys house
5712 McCloud Street
R.S.V.P. to Al by the 24th 203.456.7890

6. Producer sends another record <key=“Horrible Halloween party”, value=“Invitation”> to ”parties” topic
7. Paul and Chewy receive the invitation

Paul receives the Halloween invitation as the key is different and the record is sent to the partition that Paul is allocated to

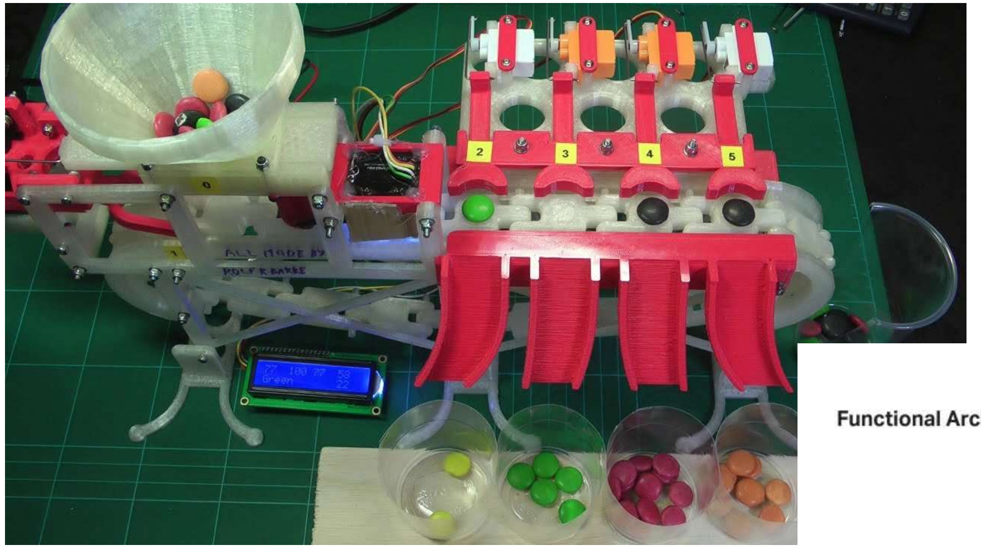
Chewy is the only consumer in his group so he gets every record no matter what partition it’s sent to

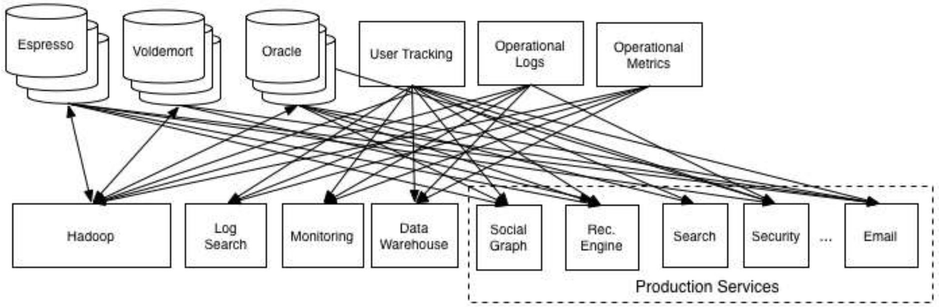
Chewy gets dressed up
And goes to the party



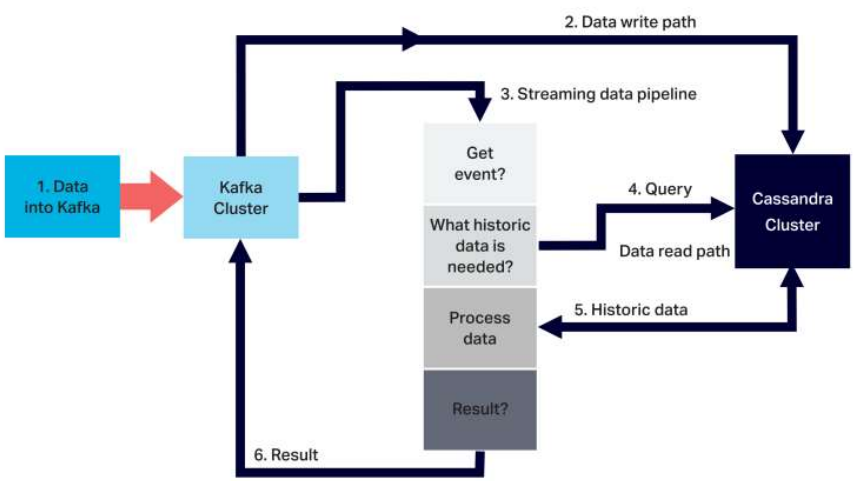
Who did you imagine was producing the invites?

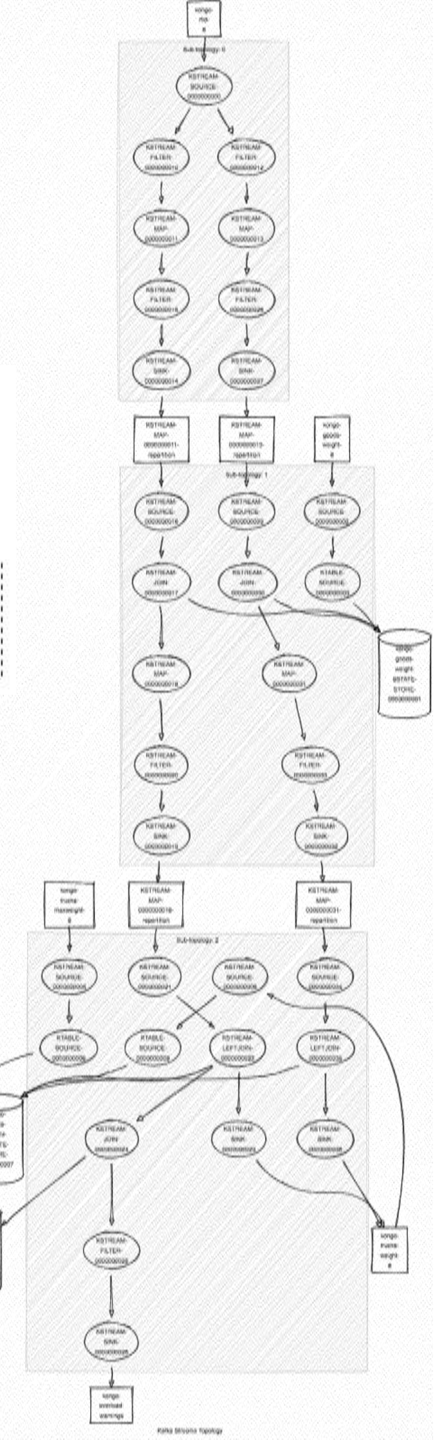
Example Kafka Use Cases



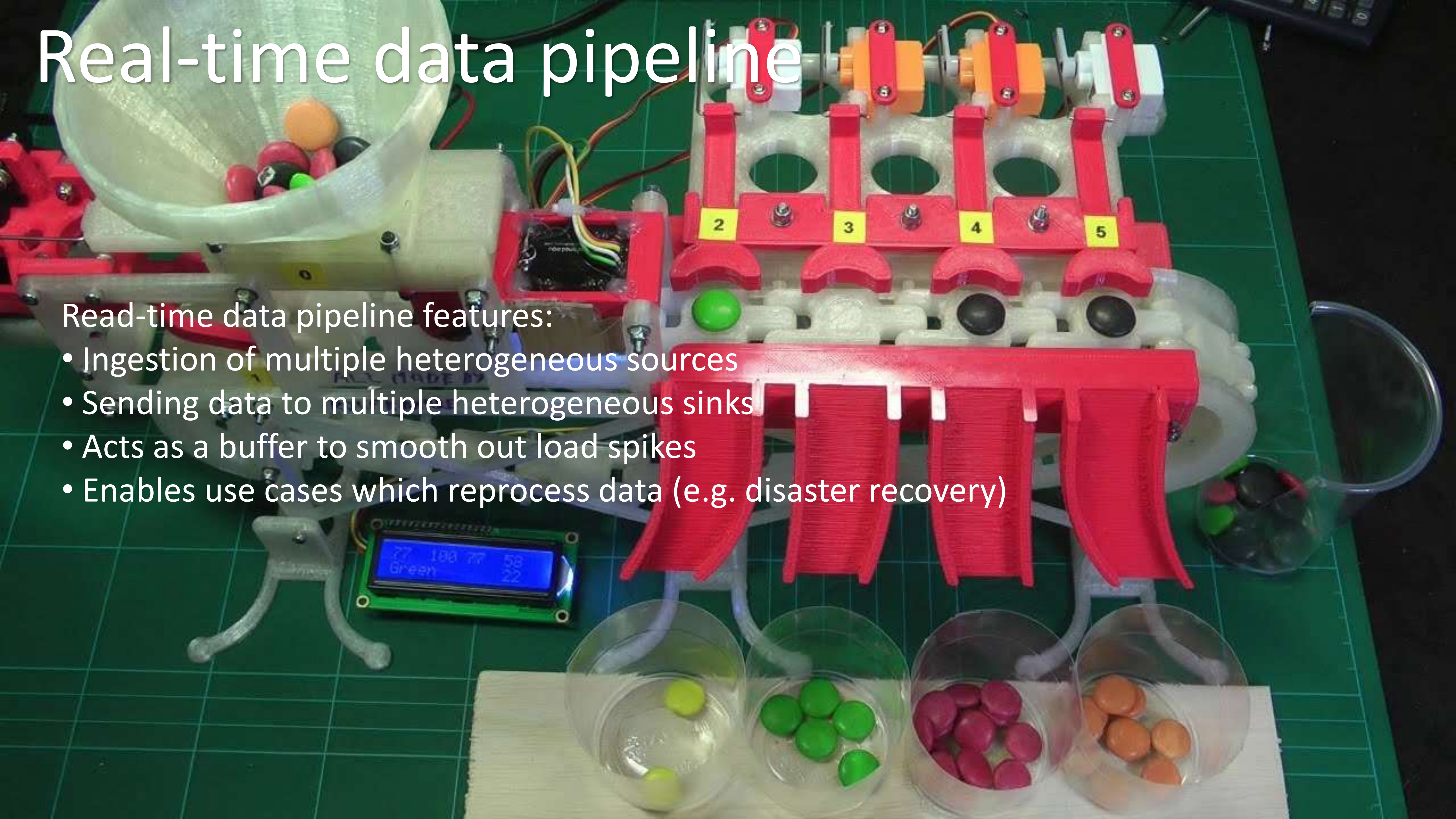


Functional Architecture Overview: High Level





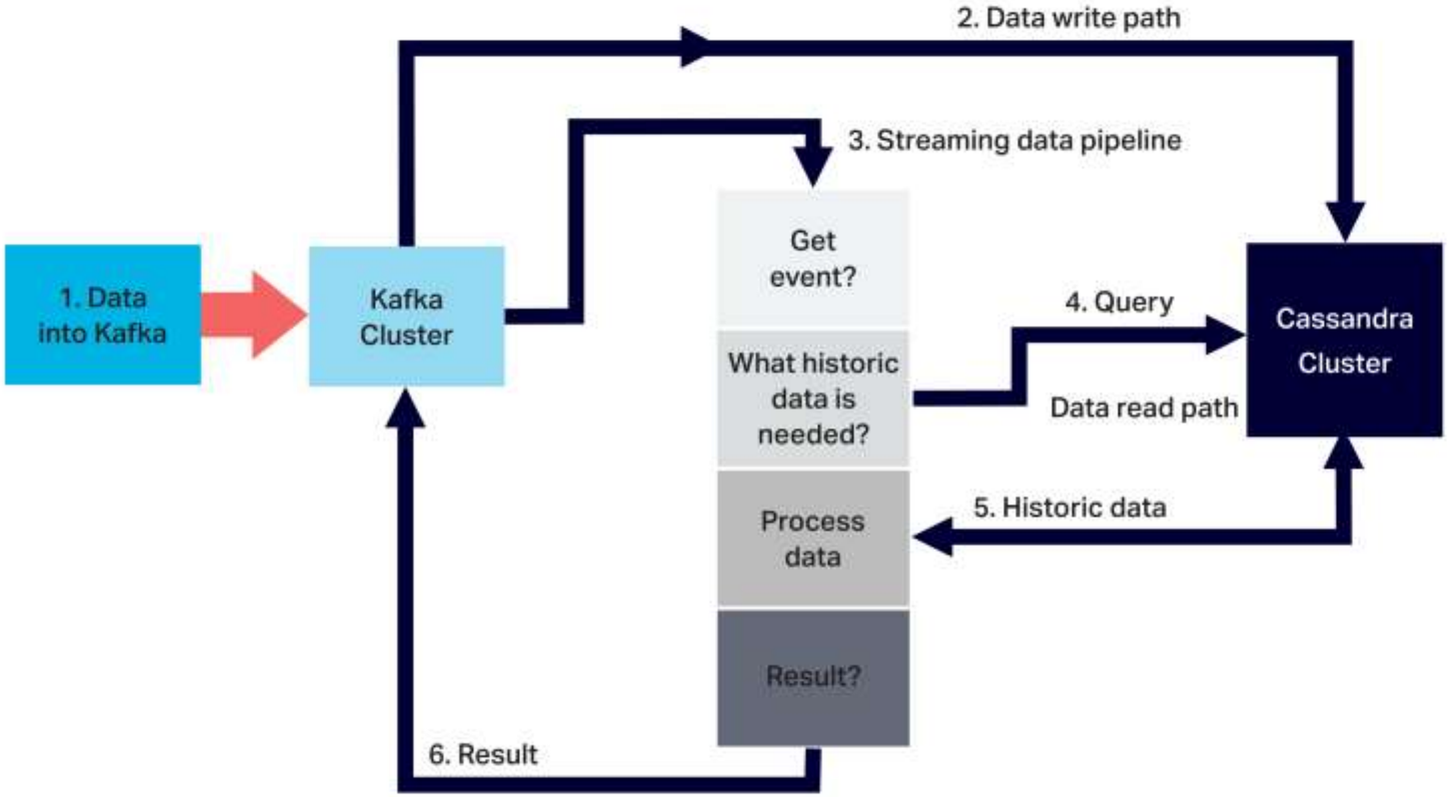
Real-time data pipeline



Read-time data pipeline features:

- Ingestion of multiple heterogeneous sources
- Sending data to multiple heterogeneous sinks
- Acts as a buffer to smooth out load spikes
- Enables use cases which reprocess data (e.g. disaster recovery)

Anomaly Detection Pipeline



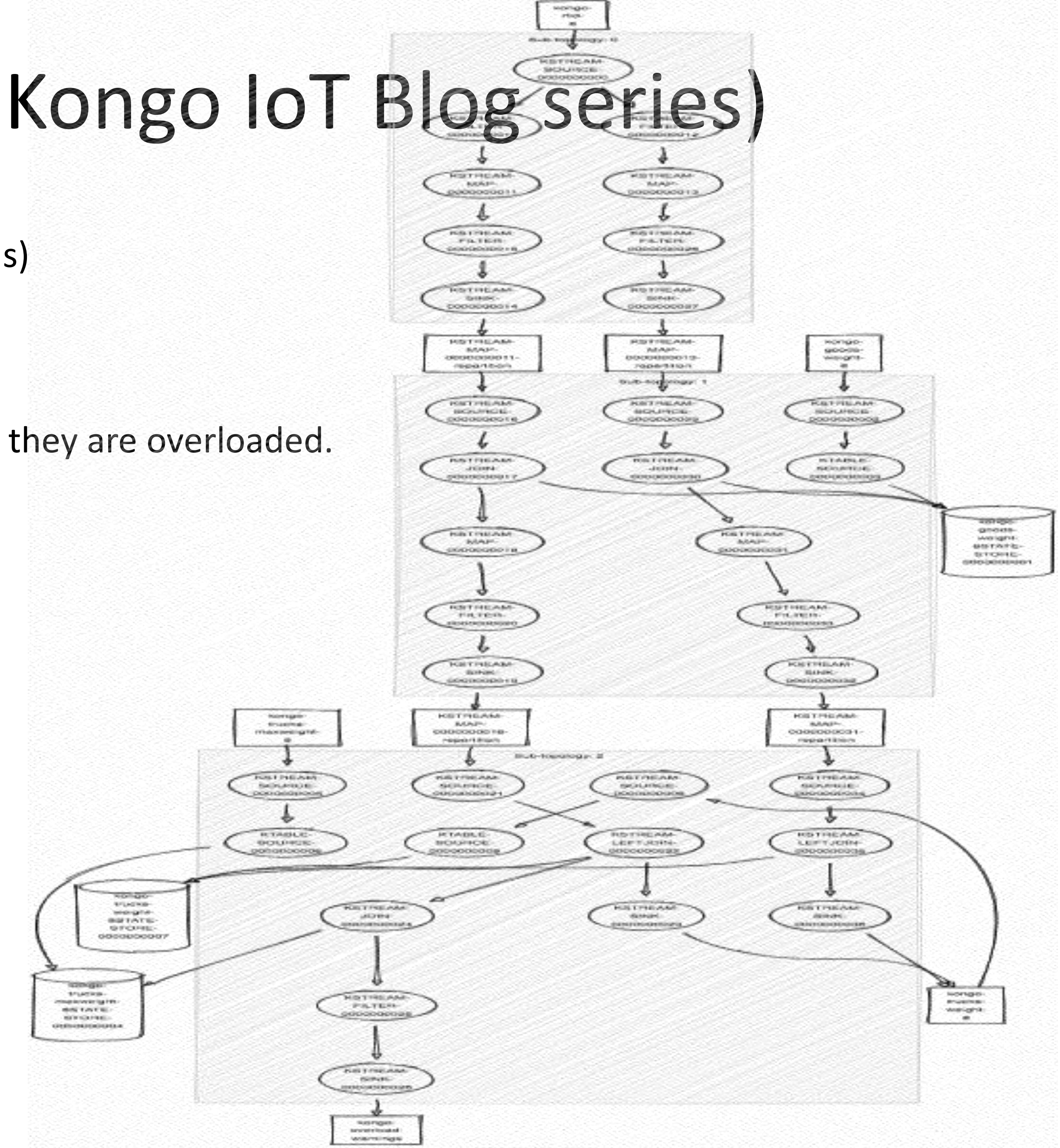
Real-time Event processing pipeline:

- Simple event driven applications (If X then Y…)
- May write and read from other data sources (e.g. Cassandra)
- New Events sent back to Kafka or to other systems
- E.g. Anomaly Detection, check out my current blog series if you are interested in this example.

Kafka Streams Processing (Kongo IoT Blog series)

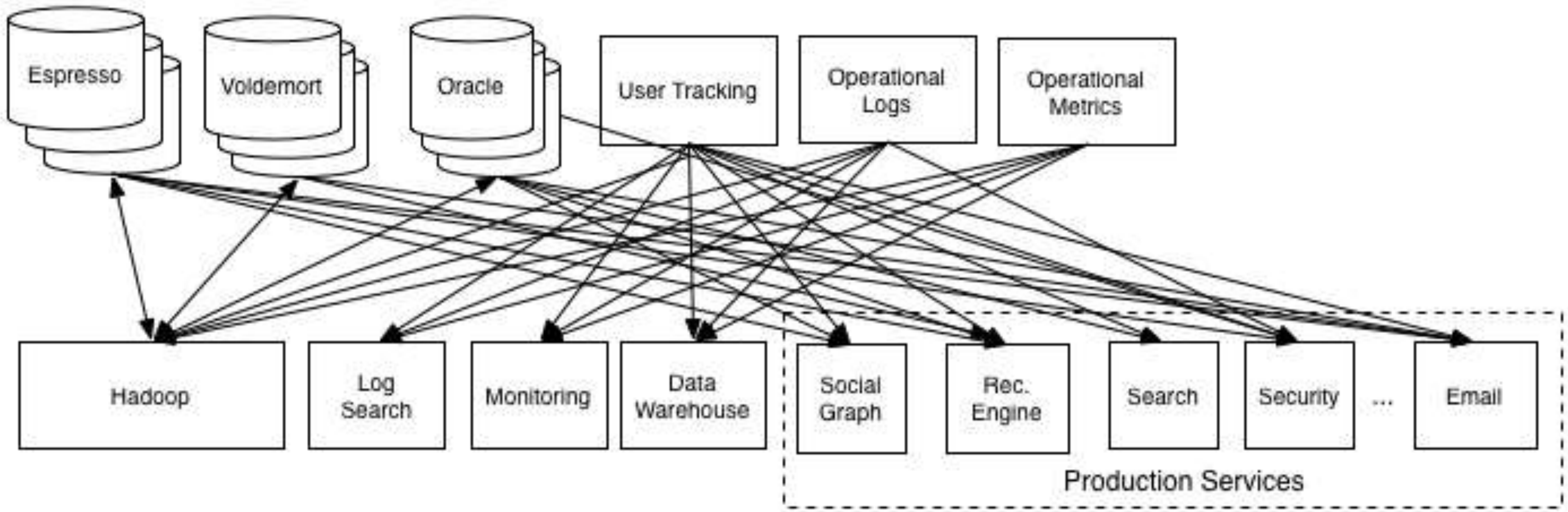
Streams processing features:

- Complex streams processing (multiple events and streams)
- Time, windows, and transformations
- Uses Kafka Streams API, includes state store
- Visualization of the streams topology
- Continuously computes the loads for trucks and checks if they are overloaded.


kongo-rfid-8
Sub-topology: 0
KSTREAM-SOURCE-0000000000
KSTREAM-FILTER-0000000010
KSTREAM-FILTER-0000000012
KSTREAM-MAP-0000000011
KSTREAM-MAP-0000000013
KSTREAM-FILTER-0000000015
KSTREAM-FILTER-0000000026
KSTREAM-SINK-0000000014
KSTREAM-SINK-0000000027
KSTREAM-MAP-0000000011-repartition
KSTREAM-MAP-0000000013-repartition
kongo-goods-weight-8
Sub-topology: 1
KSTREAM-SOURCE-0000000016
KSTREAM-SOURCE-0000000029
KSTREAM-SOURCE-0000000002
KSTREAM-JOIN-0000000017
KSTREAM-JOIN-0000000030
KTABLE-SOURCE-0000000003
kongo-goods-weight-8STATE-STORE-0000000001
KSTREAM-MAP-0000000018
KSTREAM-MAP-0000000031
KSTREAM-FILTER-0000000020
KSTREAM-FILTER-0000000033
KSTREAM-SINK-0000000019
KSTREAM-SINK-0000000032
kongo-trucks-maxweight-8
KSTREAM-MAP-0000000018-repartition
KSTREAM-MAP-0000000031-repartition
Sub-topology: 2
KSTREAM-SOURCE-0000000005
KSTREAM-SOURCE-0000000021
KSTREAM-SOURCE-0000000008
KSTREAM-SOURCE-0000000034
KTABLE-SOURCE-0000000006
KTABLE-SOURCE-0000000009
KSTREAM-LEFTJOIN-0000000023
KSTREAM-LEFTJOIN-0000000035
kongo-trucks-weight-8STATE-STORE-0000000007
KSTREAM-JOIN-0000000024
KSTREAM-SINK-0000000029
KSTREAM-SINK-0000000036
kongo-trucks-maxweight-8STATE-STORE-0000000004
kongo-trucks-weight-8
KSTREAM-FILTER-0000000025
KSTREAM-SINK-0000000026
kongo-overload-warnings

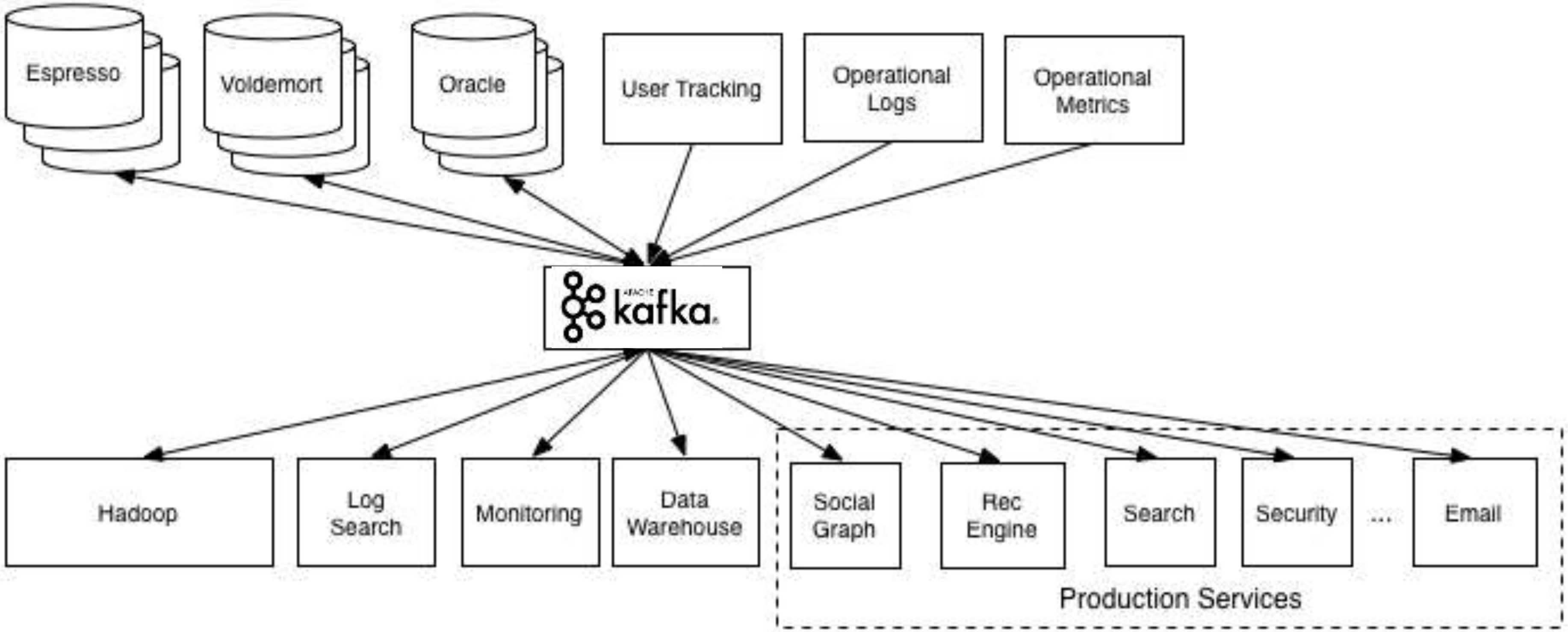
Kafka Streams Topology

Linkedin - Before Kafka (BK)



A real example from Linkedin, who developed Kafka.
Before Kafka they had spaghetti integration of monolithic applications.
To accommodate growing membership and increasing site complexity, they migrated from a monolithic application infrastructure to one based on microservices, *which made the integration even more complex!*

After Kafka (AK)



Rather than maintaining and scaling each pipeline individually, they invested in the development of a single, distributed pub-sub platform - Kafka was born.
The main benefit was *better Service decoupling and independent scaling*.

The End (of the introduction) - *Find out more*

instaclustr

Apache Kafka: https://kafka.apache.org/

Instaclustr blogs

- Mix of Cassandra, Spark, Zeppelin and Kafka

https://www.instaclustr.com/paul-brebner/

- Kafka introduction

https://insidebigdata.com/2018/04/12/developing-deeper-understanding-apache-kafka-architecture/
https://insidebigdata.com/2018/04/19/developing-deeper-understanding-apache-kafka-architecture-part-2-

- Kongo – Kafka IoT logistics application blog series

https://www.instaclustr.com/instaclustr-kongo-iot-logistics-streaming-demo-application/

- Anomaly detection with Kafka and Cassandra (and Kubernetes), current blog series

https://www.instaclustr.com/anomalia-machina-1-massively-scalable-anomaly-detection-with-apache-kafka-

Instaclustr's Managed Kafka (Free trial)

https://www.instaclustr.com/solutions/managed-apache-kafka/