

Apache Kafka

A Visual Introduction

NOTE:

The round-robin behaviour with no key is correct for Kafka version < 2.4, or for Kafka >= 2.4 if explicitly using the round robin partitioner; from 2.4 the default non-key partitioner is the new "sticky partitioner"

Paul Brebner
Technology Evangelist



www.instacluster.com



sales@instacluster.com

© Instacluster Pty Limited, 2019, 2022 [<https://www.instacluster.com/company/policies/terms-conditions/>]. Except as permitted by the copyright law applicable to you, you may not reproduce, distribute, publish, display, communicate or transmit any of the content of this document, in any form, but any means, without the prior written permission of Instacluster Pty Limited.

In this *Visual Introduction to Kafka*,
we're going to build a



kafka®

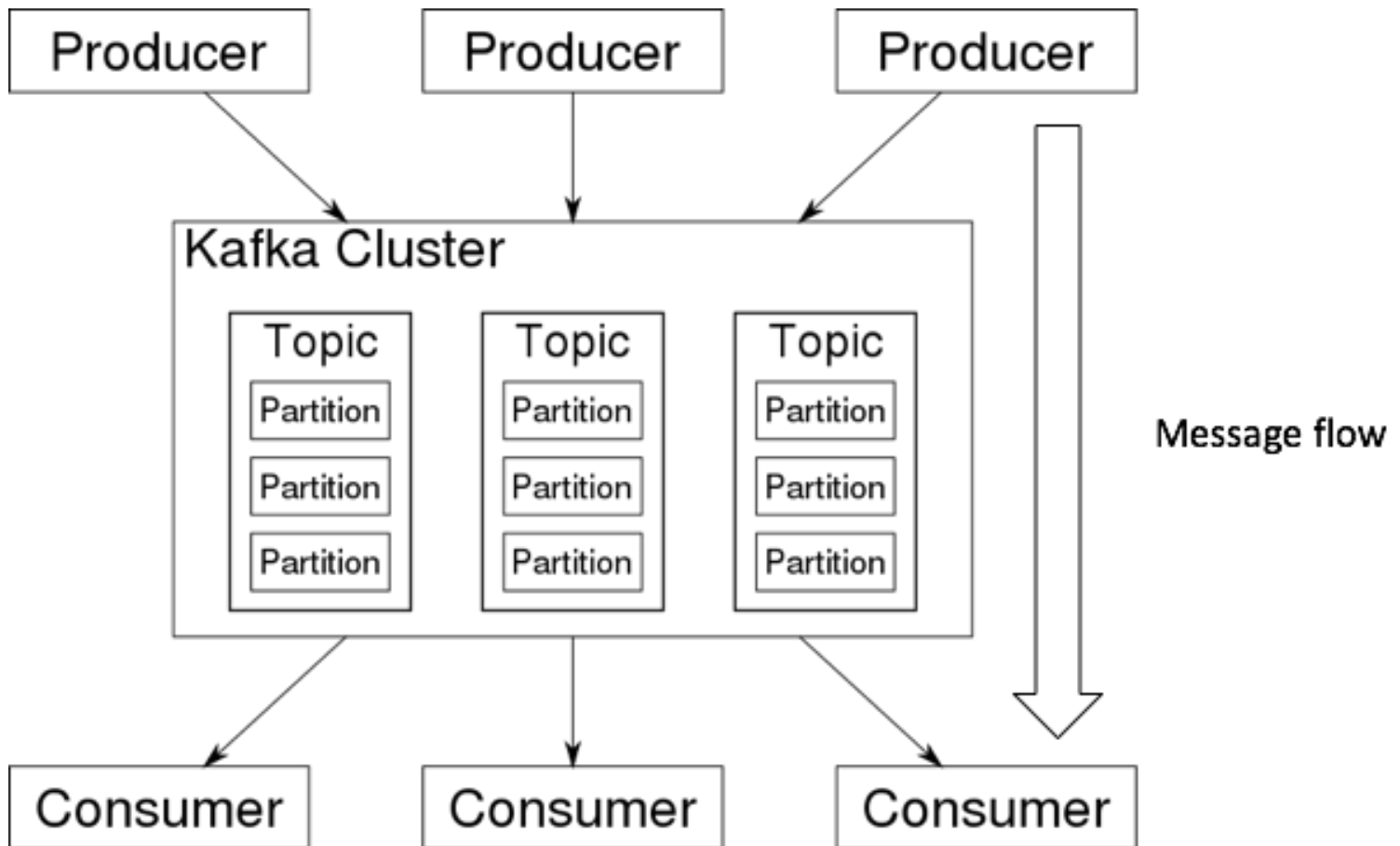
Postal

Service

We'll learn about Kafka Producers,
Consumers, Topics, Partitions,
Keys, Records, Delivery Semantics
(Guaranteed delivery, and who gets
what messages), Consumer Groups
and time Travel (event reprocessing)!

What is Kafka?

Kafka is a **distributed streams processing system**, it allows distributed producers to send messages to distributed consumers via a Kafka cluster.



Fast

Reliable

Open Source

Scalable

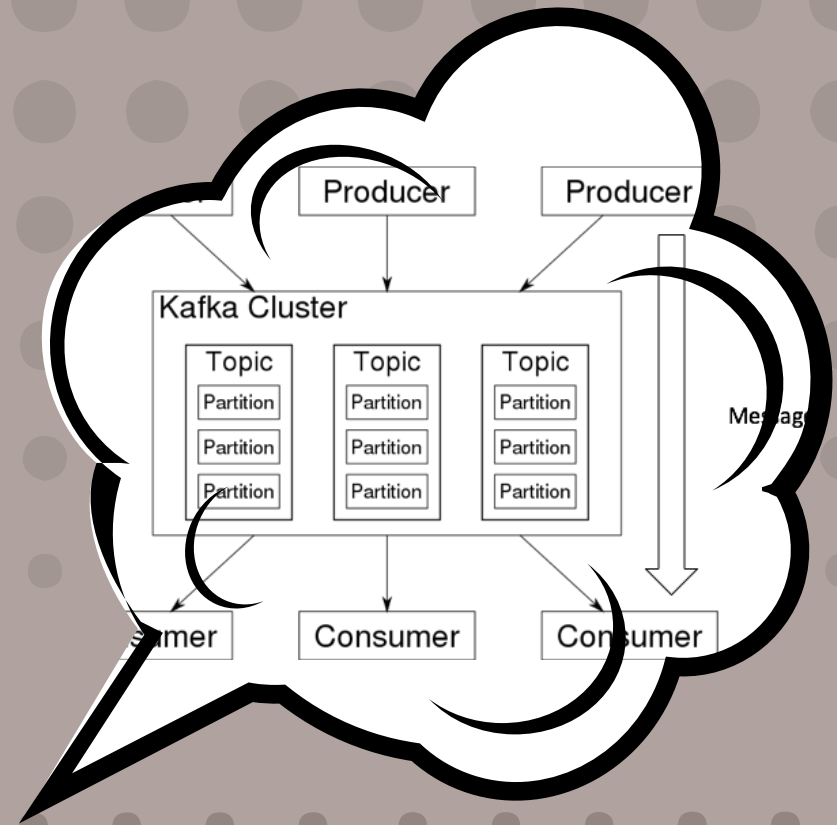
Durable

Managed Service

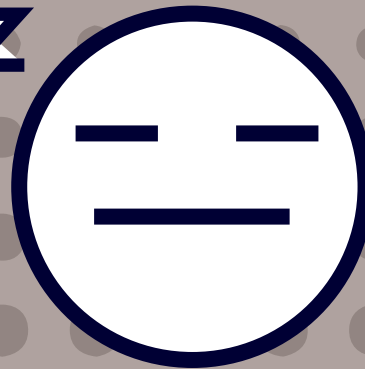
Kafka has lots of benefits:

- **It's Fast:** It has high throughput and low latency
- **It's Scalable:** It's horizontally scalable, to scale just add nodes and partitions
- **It's Reliable:** It's distributed and fault tolerant
- **It has Zero Data Loss:** Messages are persisted to disk with an immutable log
- **It's Open Source:** An Apache project
- **And it's available as an Instaclustr Managed Service:** On multiple cloud platforms

But the usual
Kafka diagram
(right) is a bit
monochrome
and boring.



Zz



This *visual introduction*
will be more
colourful
and it's going to be
an extended story...

Let's build a modern day fully electronic postal service



To send messages from A to B



First, we need an "A".



A is a producer,
it sends a message...

To B, the consumer,
the recipient of the message.



**Actually,
not.**

Due to the decline in
“snail mail” volumes,
direct deliveries have
been canceled.



*Instead we have
"Poste Restante"*

Poste Restante is not a post office in a restaurant, it's called **general delivery** (in the US). The mail is delivered to a post office, and they hold it for you until you call for it.

Consumers poll for messages by **visiting the counter** at the post office.

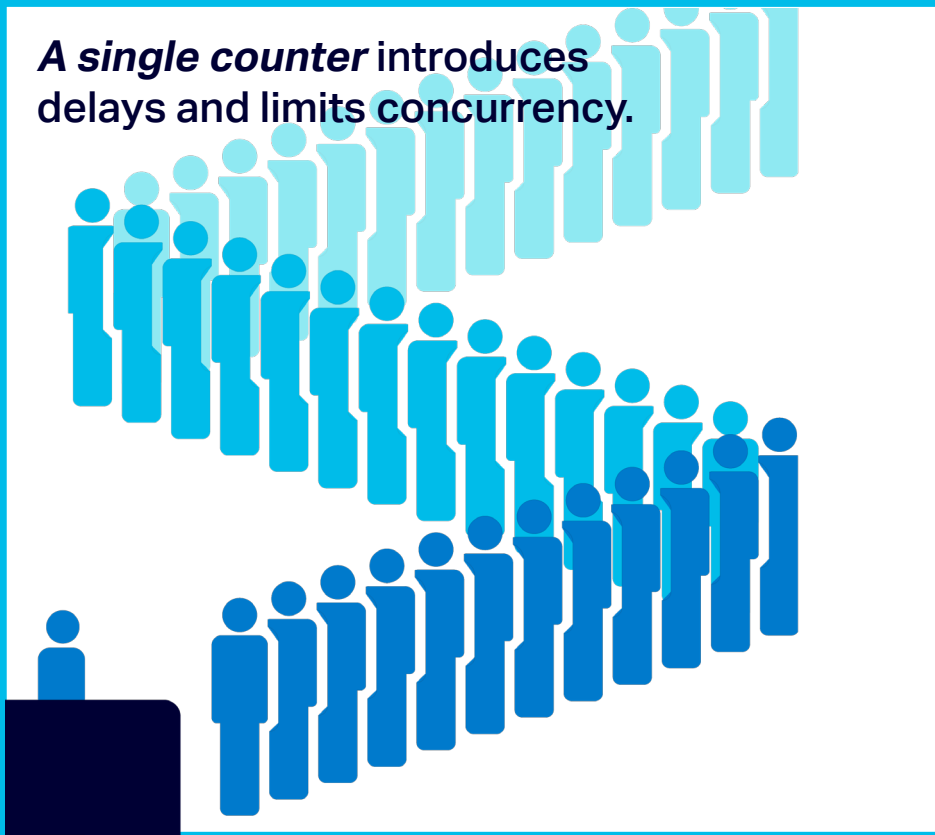
*Kafka topics act like a Post Office. What are the **benefits**?*

- ★ **Disconnected delivery**—consumer doesn't need to be available to receive messages
- ★ **There's less effort for the messaging service**—only has to delivery to a few locations not larger number of consumer addresses
- ★ **And it can scale better and handle more complex delivery semantics!**

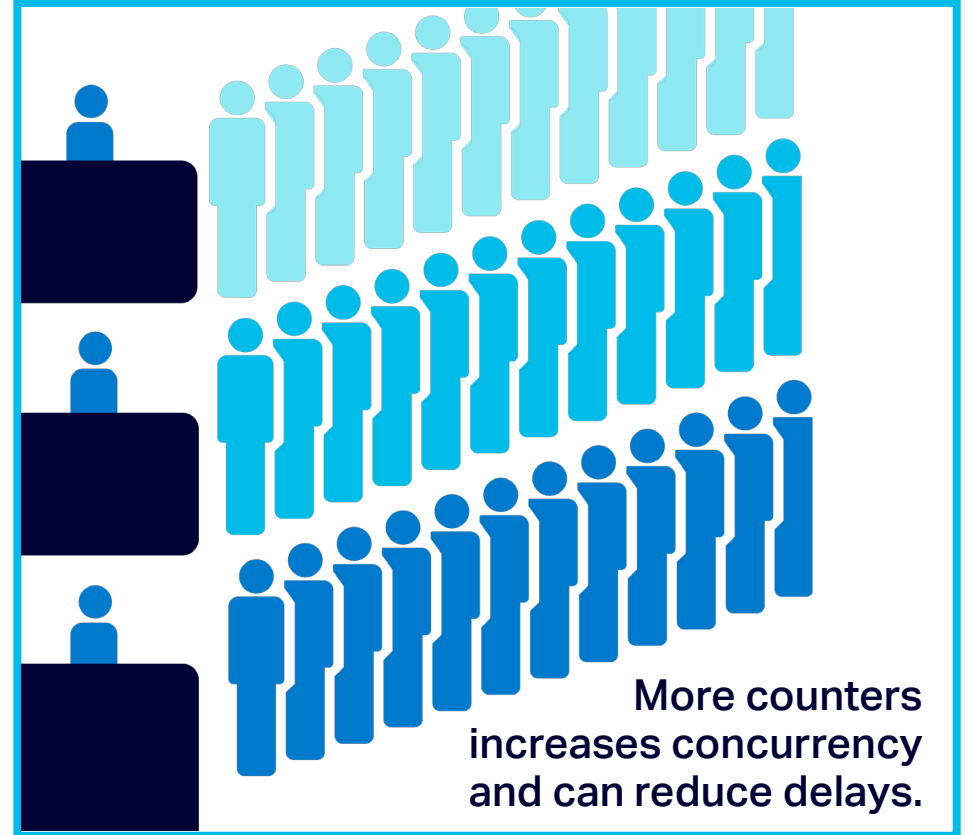


First lets see how it scales.
What if there are many consumers for a topic?

A single counter introduces delays and limits concurrency.



More counters increases concurrency and can reduce delays.



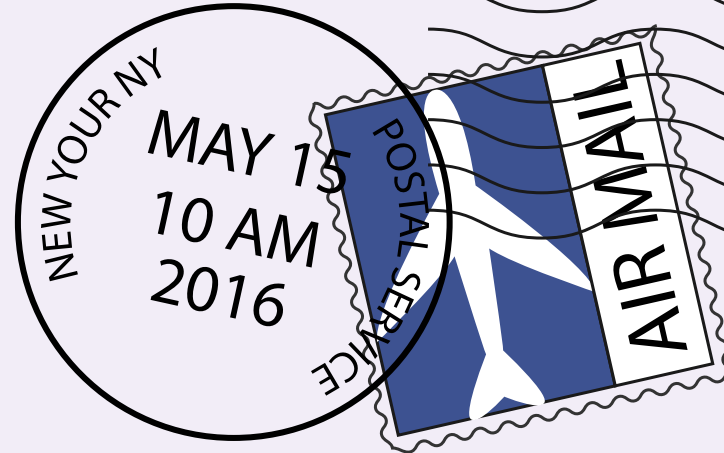
Kafka Topics have 1 or more Partitions. Partitions function like multiple counters and enable high concurrency.

*Before looking at delivery semantics,
let's see what a **message** looks like.*

In Kafka a message is called a Record and is a bit like a letter.

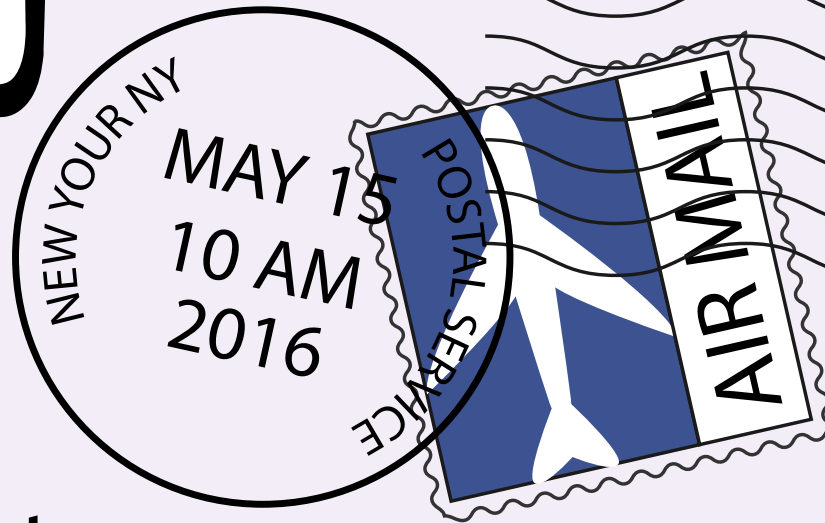
The topic is the *destination*,
The North Pole.

Santa
North Pole



The "Postmark" includes a *timestamp*, *offset* in the topic, and the *partition* it was sent to.

timestamp,
offset,
partition



Topic

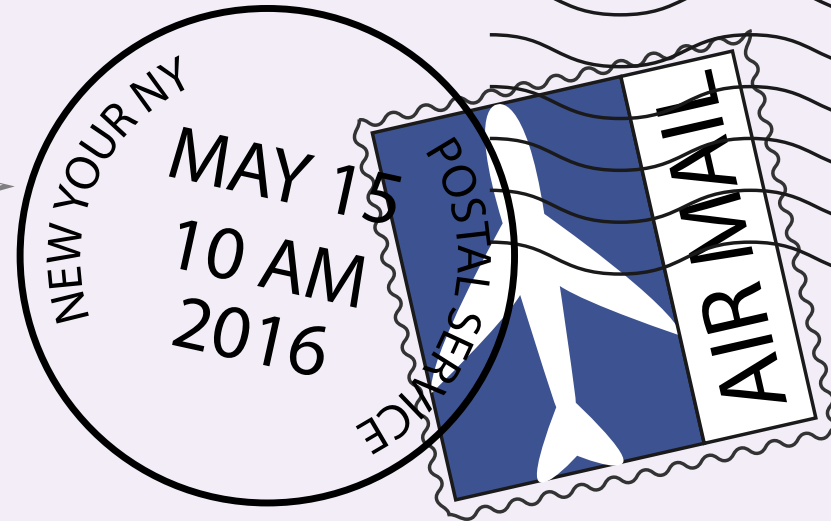
Santa
North Pole

Time semantics are flexible, either the time of event creation, ingestion, or processing.

There's also a thing called a **Key**, which is **optional**. It refines the destination so it's a bit like the rest of the address.

We want this letter sent to Santa not just a random Elf.

*timestamp,
offset,
partition*



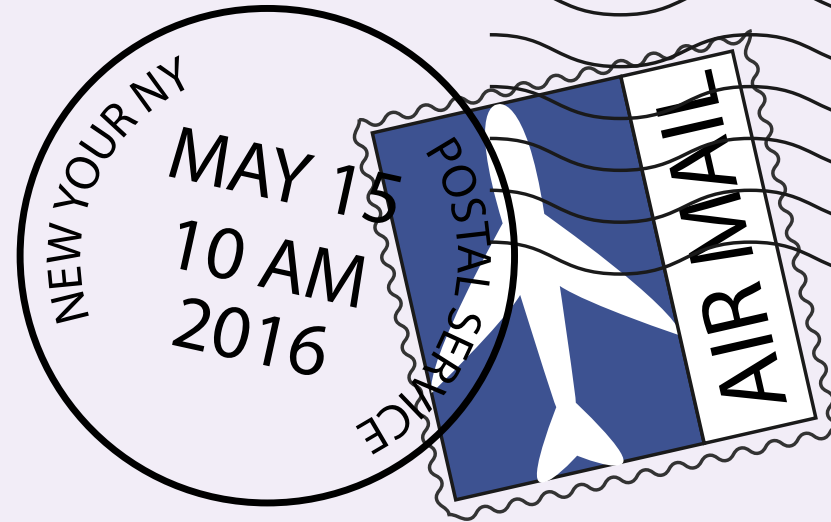
**Key Partition
(optional)**

Santa
North Pole

Topic

Value (Content)

*timestamp,
offset,
partition*



*Key Partition
(optional)*

**Santa
North Pole**

Topic

And the **value** is the contents (just a byte array).
Kafka Producers and consumers need to have a shared
serializer and de-serializer for both the key and value.

Dear Santa,
How are you? I'm good.
Here is what I want for
Christmas.

A http://www.amazon.com/gp/product/B0032HF60M/ref=59_hps_bw_g21_in03?pf_rd_m=ATVPDKIKXODER&pf_rd_s=center-3&pf_rd_t=1&pf_rd_p=1328901542&pf_rd_i=16579

Kafka doesn't look inside the value, but the Producer and Consumer do, and the Consumer can try and make sense of the message

(Can you?!)



Next

let's look at
delivery semantics

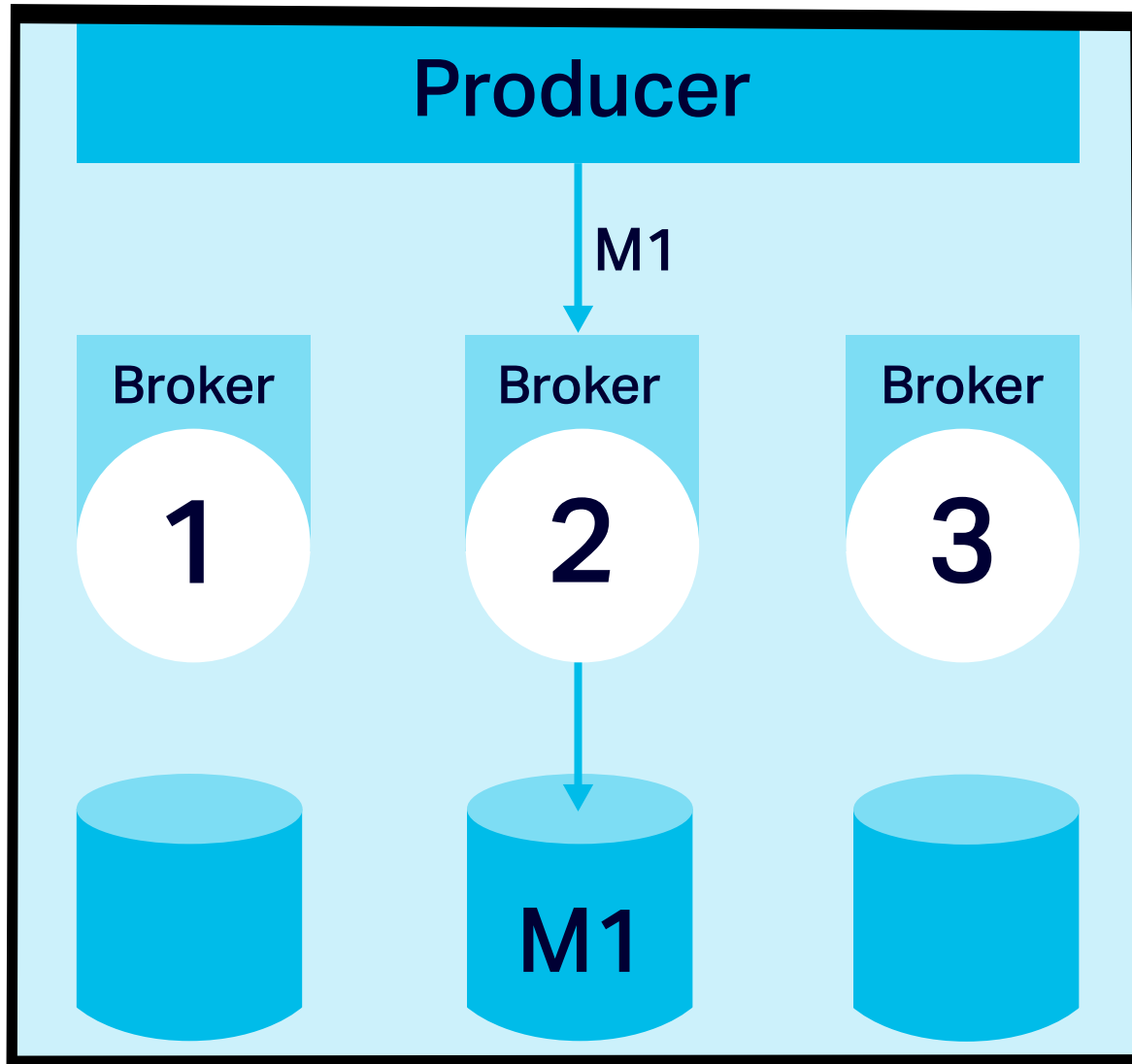
For example, do we care if the
message actually arrives or not?



**Yes we do!
Guaranteed
message
delivery is
desirable.**

Last century, homing pigeons were prone to getting lost or eaten by predators, which necessitated sending the same message with several pigeons.

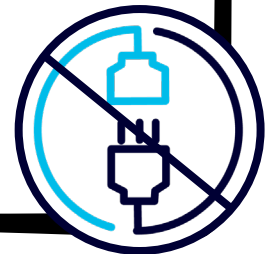
How does Kafka guarantee delivery?



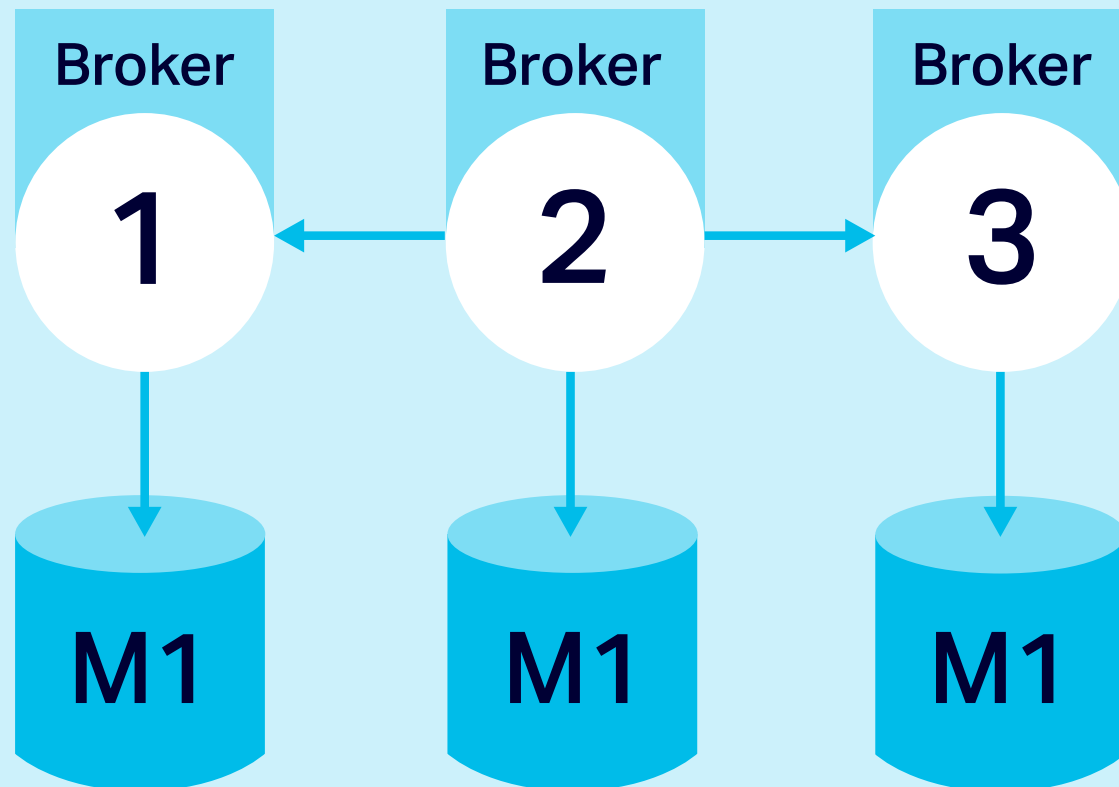
A Message (M1) is written to a broker (2).

The message is always persisted to disk.

This makes it resilient to power failure



Producer



The message is also replicated on multiple "brokers", 3 is typical.

And makes it resilient to loss of some servers (all but one).

Producer

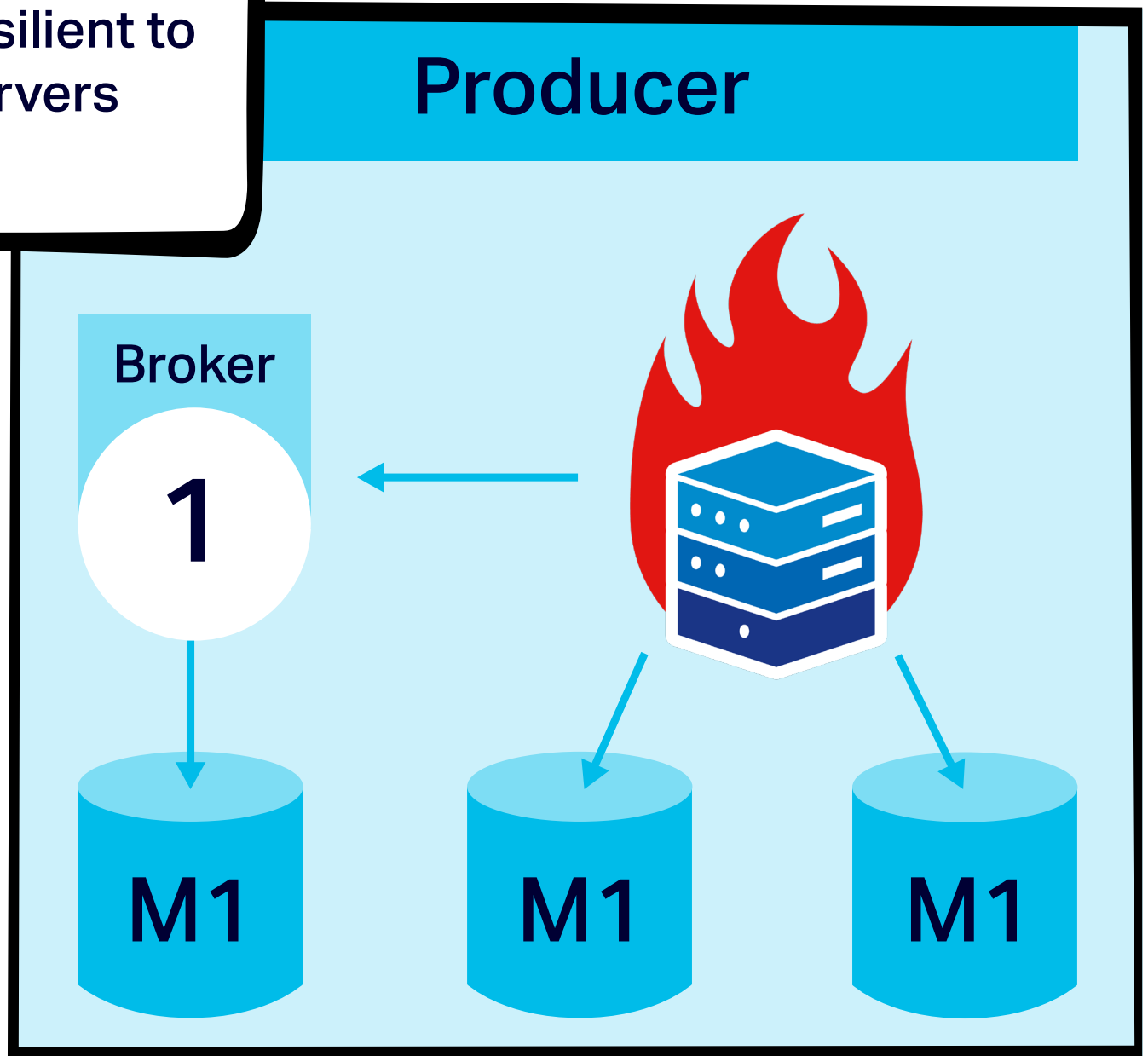
Broker

1

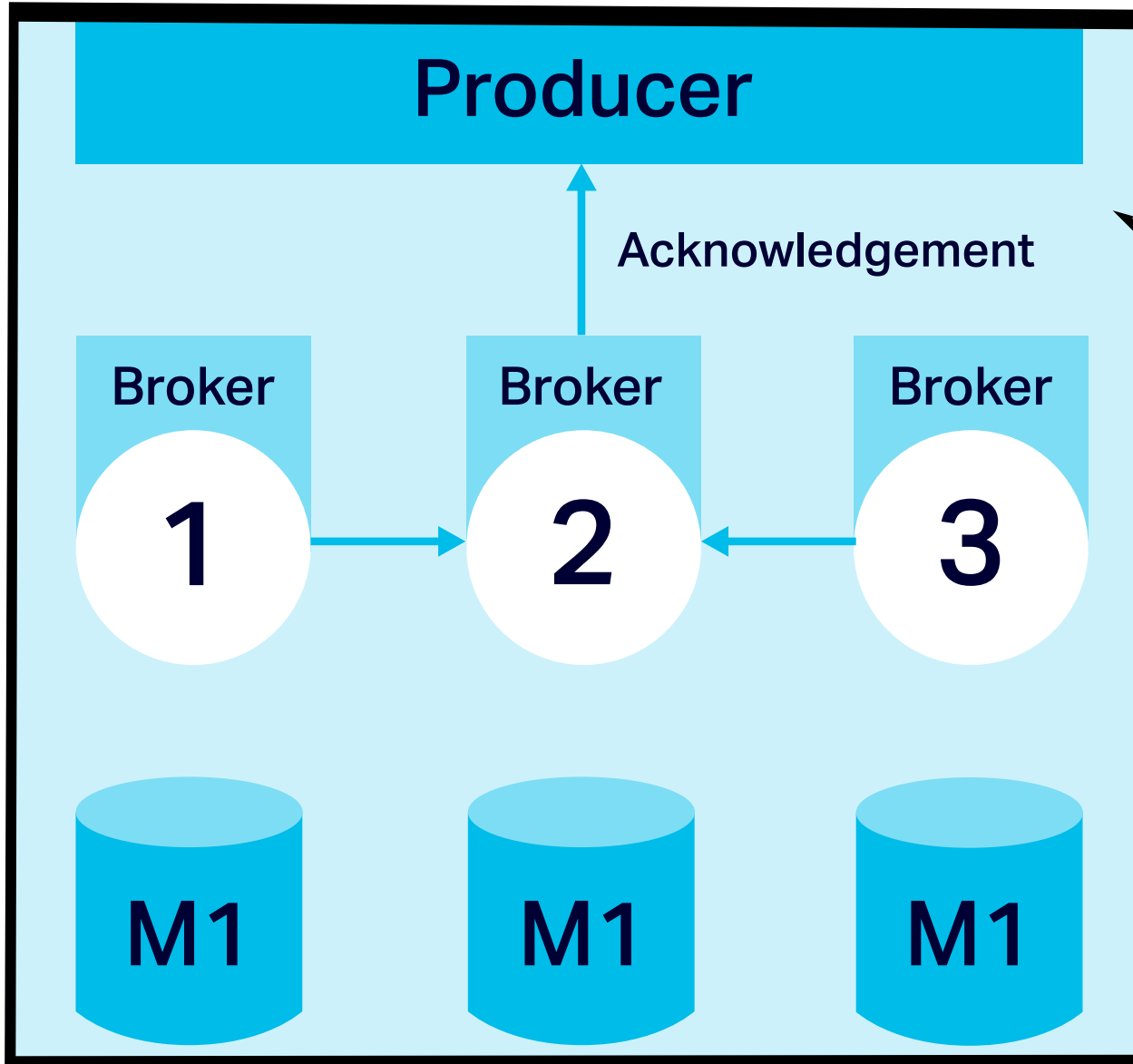
M1

M1

M1



Finally the producer gets *acknowledgement* once the message is persisted and replicated (configurable for number, and sync or async).



The message is now available from more than one broker in case some fail.

This also increases the read concurrency as partitions are spread over multiple brokers.

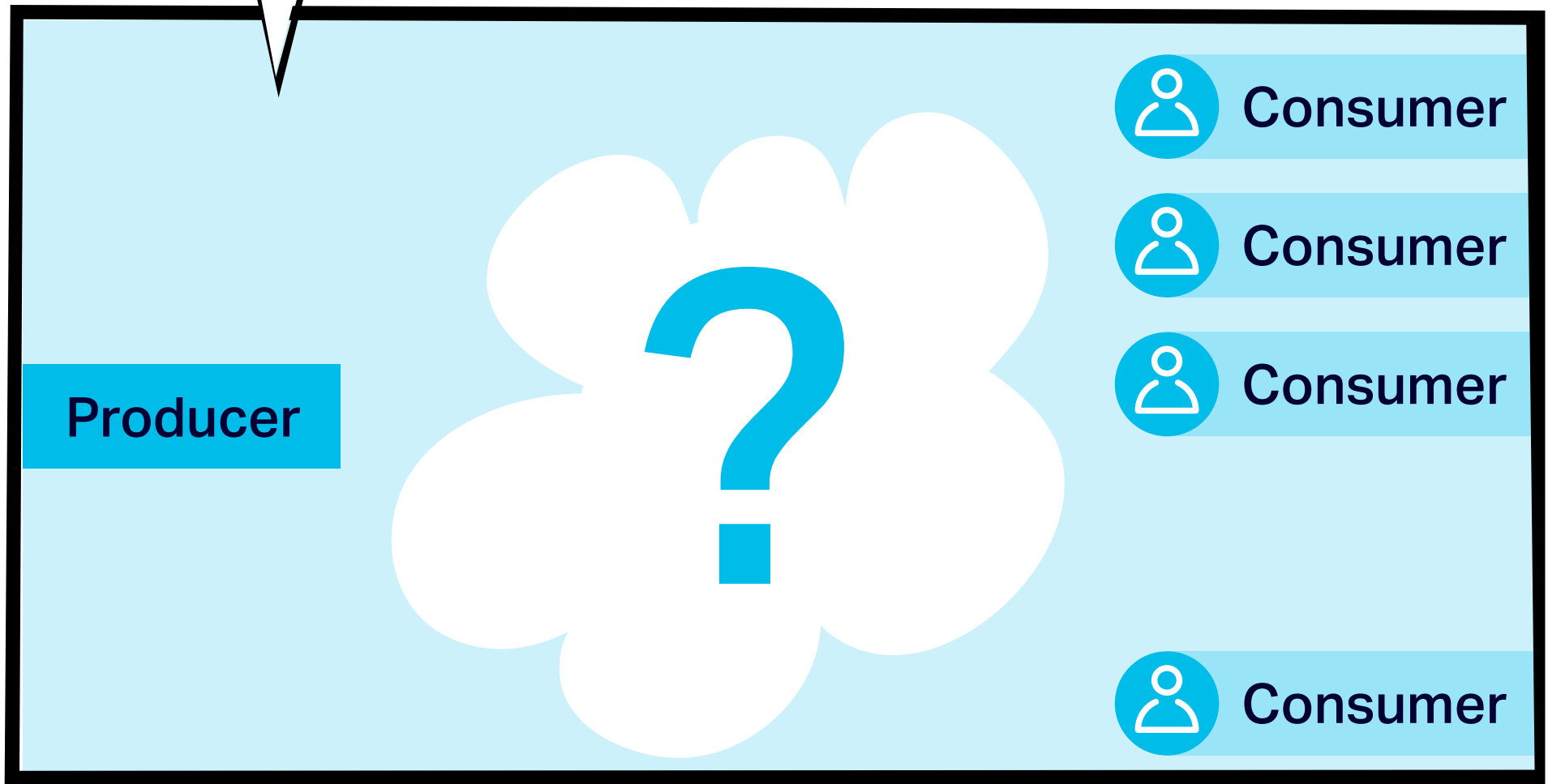


NOW

let's look at another aspect
of ***delivery semantics***

Who gets the messages and how
many times are messages delivered?

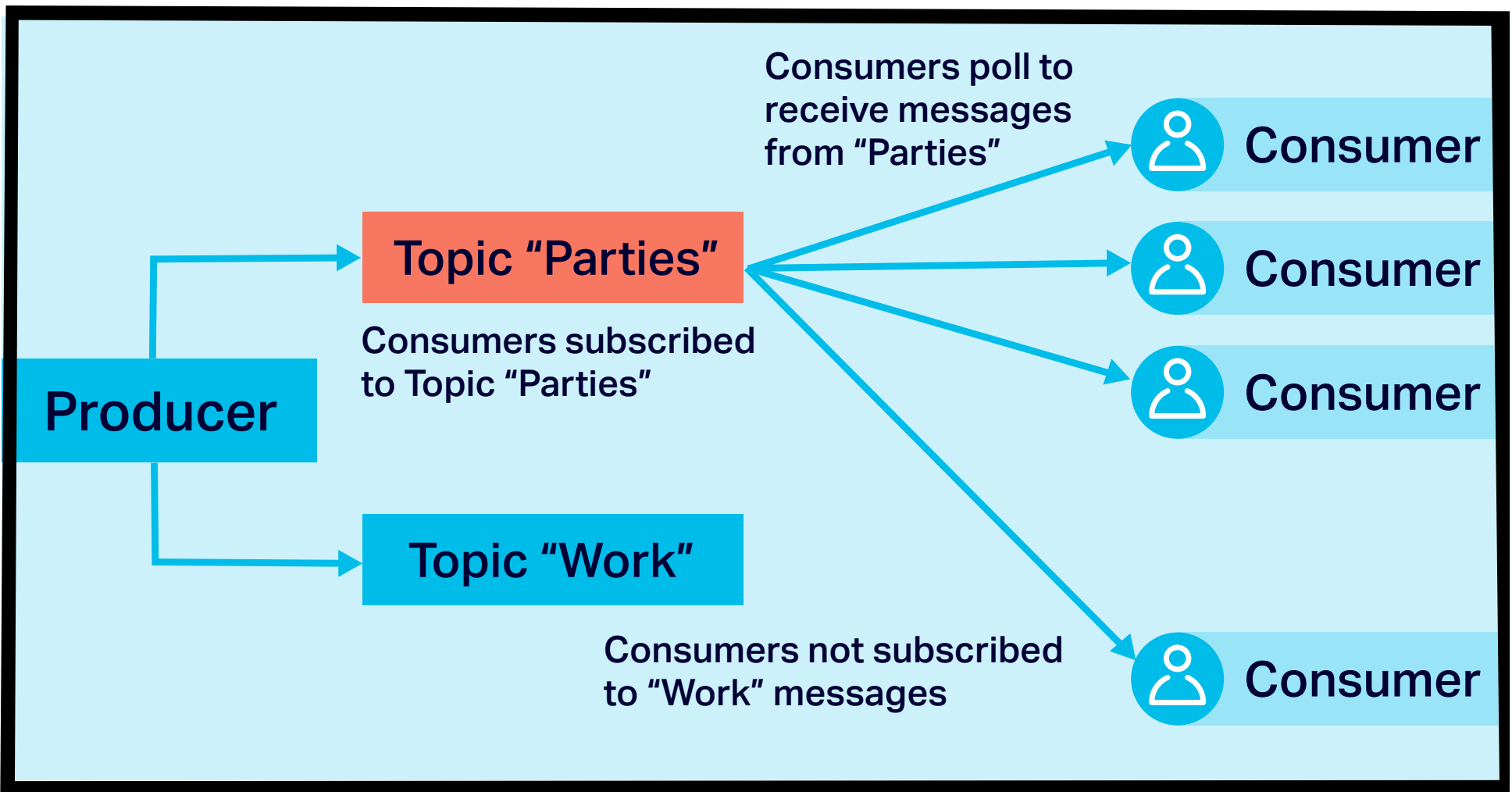
Kafka is "pub-sub". It's loosely coupled, producers and consumers don't know about each other.



Filtering, or which consumers get which messages, is **topic based**.

- Producers send messages to topics.
- Consumers subscribe to topics of interest, e.g. parties.
- When they poll they only receive messages sent to those topics.

None of these consumers will receive messages sent to the "Work" topic.



A few more details and we can see how this works.
Kafka works like an Amish Barn raising.

Partitions and a consumer group share work across multiple consumers, the more partitions a topic has the more consumers it supports.

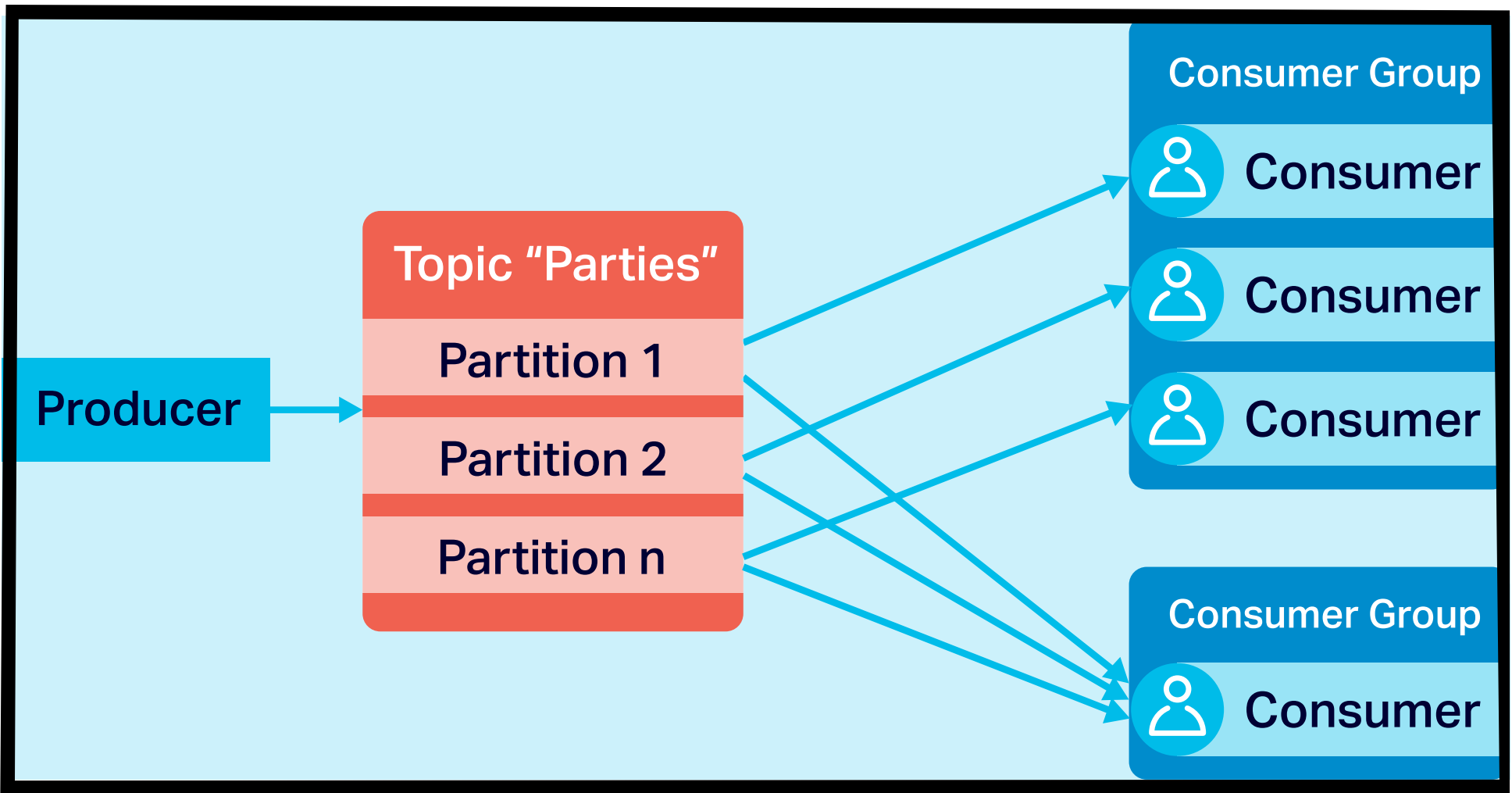


Kafka also works like Clones.

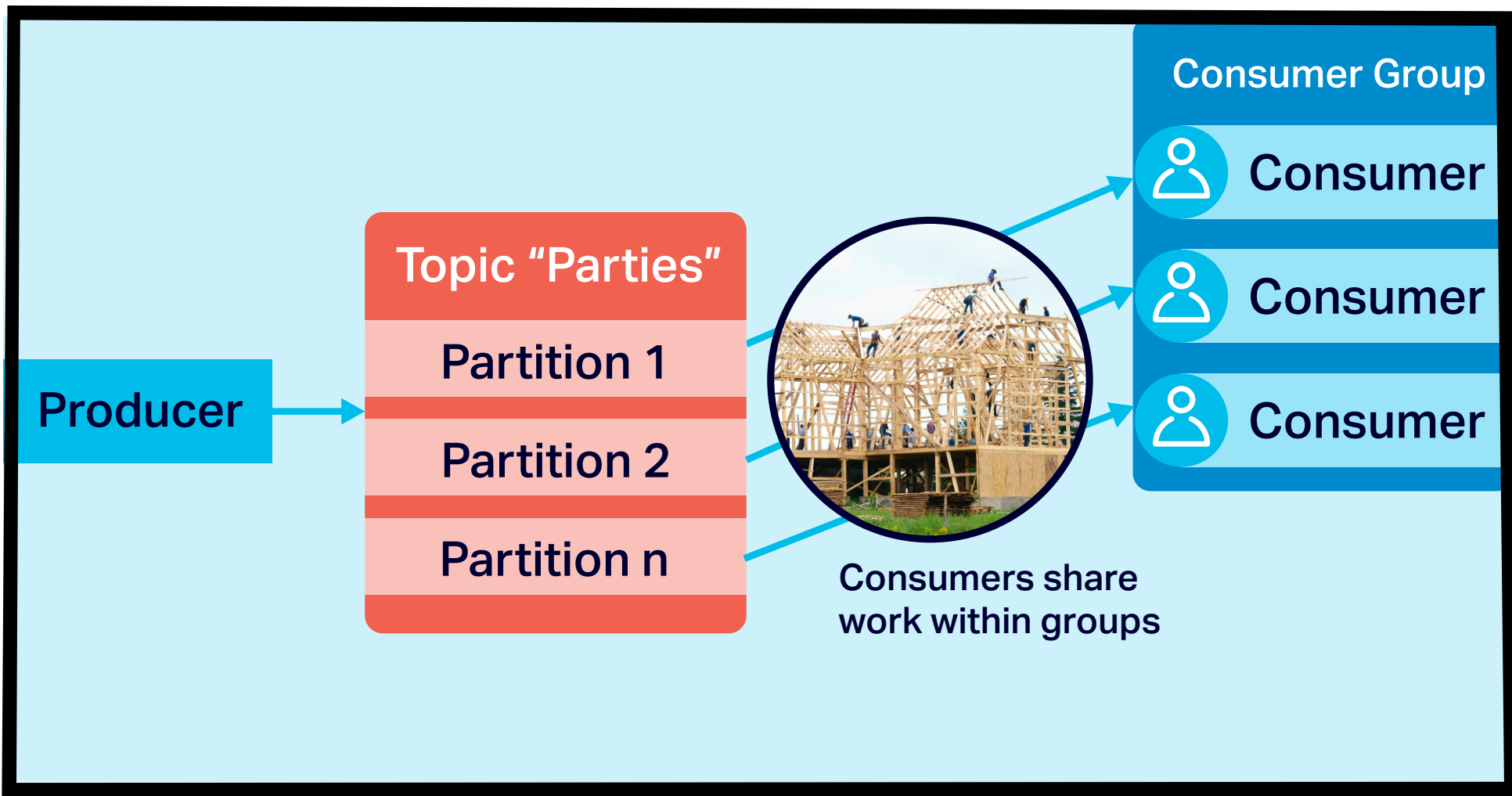
It supports delivery of the same message to multiple consumers with consumer groups.

Kafka doesn't throw messages away immediately they are delivered, so the same message can be delivered to multiple consumer groups.

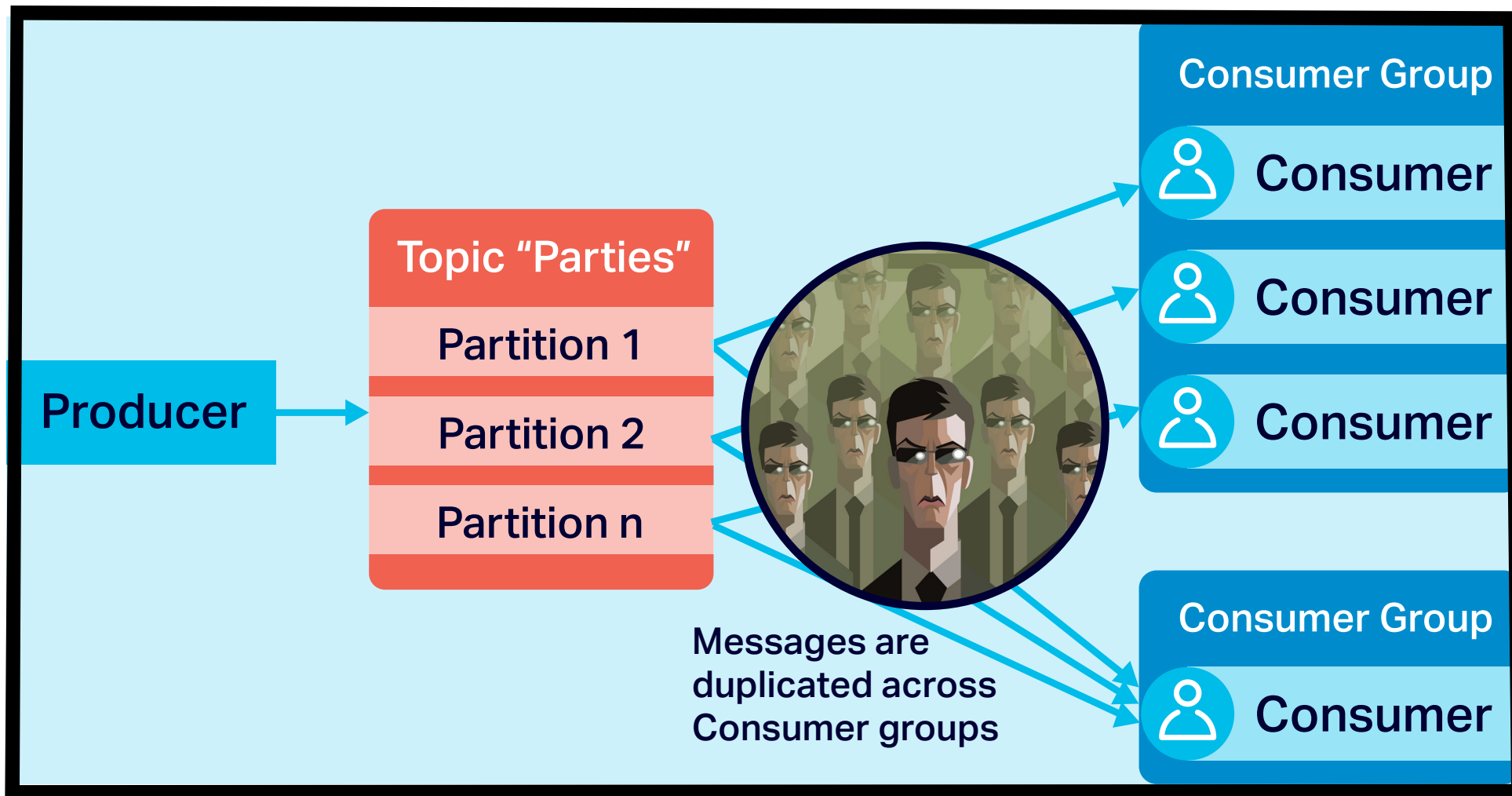
Consumers subscribed to "parties" topic are allocated partitions. When they poll they will **only** get messages from their allocated partitions.



This enables consumers in the same group to share the work around. Each consumer gets **only** a subset of the available messages.



Multiple groups enable message broadcasting. Messages are duplicated across groups, as each consumer group receives a copy of each message.



Which messages are delivered to which consumers?

The final aspect of delivery semantics is to do with *message keys*.

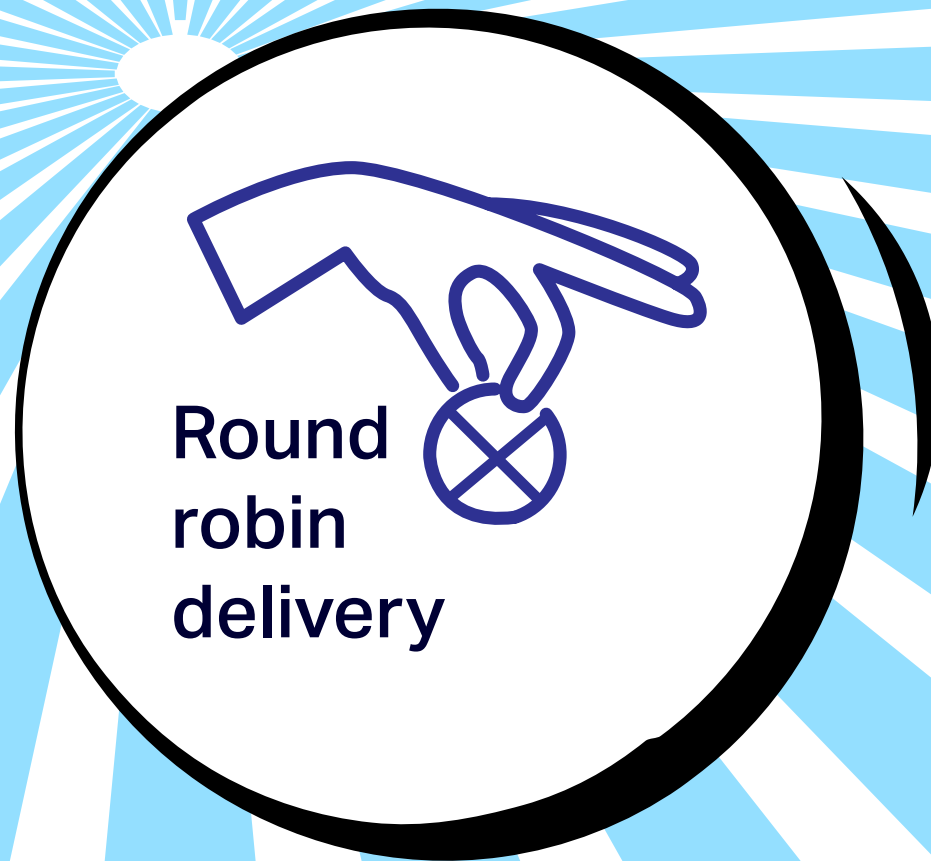
If a message has a key, then Kafka uses *Partition based* delivery.

Messages with the same key are always sent to the same partition and therefore the same consumer. And the order (within partitions) is guaranteed.



Key

But if the key is null, then Kafka uses
round robin delivery.
Each message is delivered to the next partition.



Let's look at a concrete example with two *consumer groups*:



Image: Nenad Aksic / Shutterstock.com

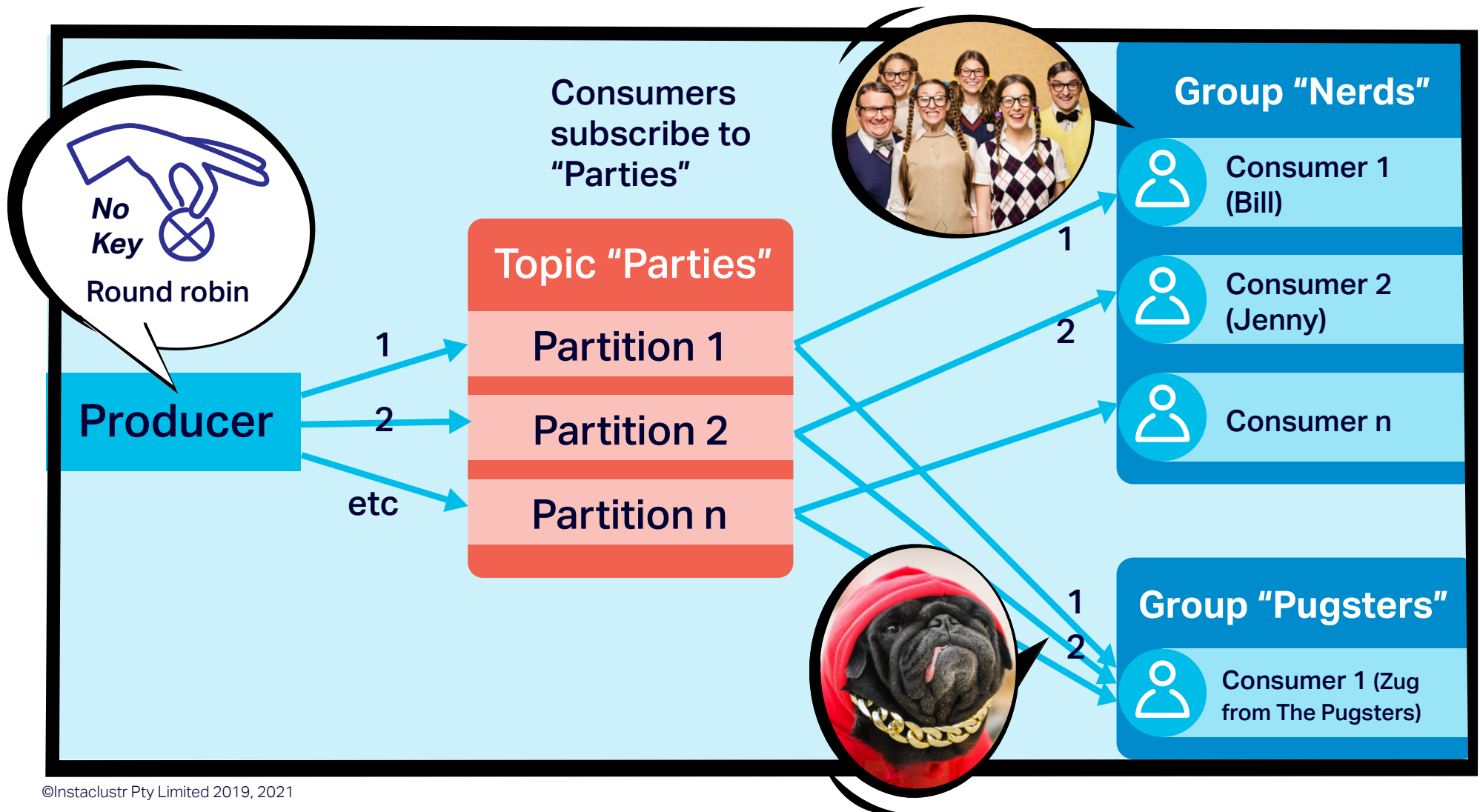
Group 1: Nerds
which has multiple consumers

Group 2: The Pugsters
which has a single consumer, Zug



Looking at the case
where there's
No Key first

Each message (1, 2, etc.) is sent to the
next partition, and all consumers allocated
to that partition will receive the message
when they poll next.



Here's what actually happens.

We're **not** showing the producer, topics, or partitions for simplicity. You'll have to imagine them.



1

Both Groups subscribe to **Topic** "parties"
(assuming 6 partitions, each consumer in
the Nerds groups gets 1 partition each)

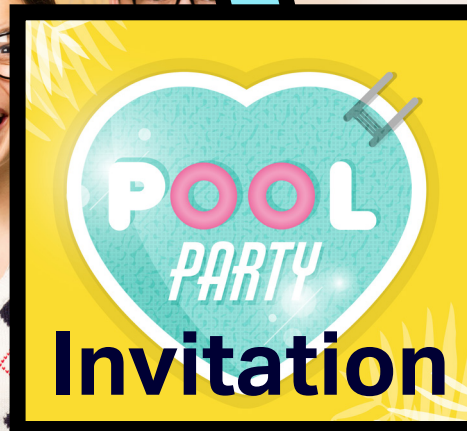


Subscribe to
"Parties"



2

Producer sends record with the **value** "Pool party—Invitation" to "**parties**" topic (there's no key)



3

Bill and Zug receive a copy of the invitation and plan to attend



4

The Producer sends another record with the **value** "Pool party—Canceled"



5

In the Nerds group, Jenny gets the message this time as it's round robin, and Zug gets it as he's the only consumer in his group:

- ▶ Jenny ignores it as she didn't get the original invite
- ▶ Bill wastes his time trying to go (as he doesn't know it's canceled)
- ▶ The rest of the gang aren't surprised at not receiving any invites and stay home to do some hacking

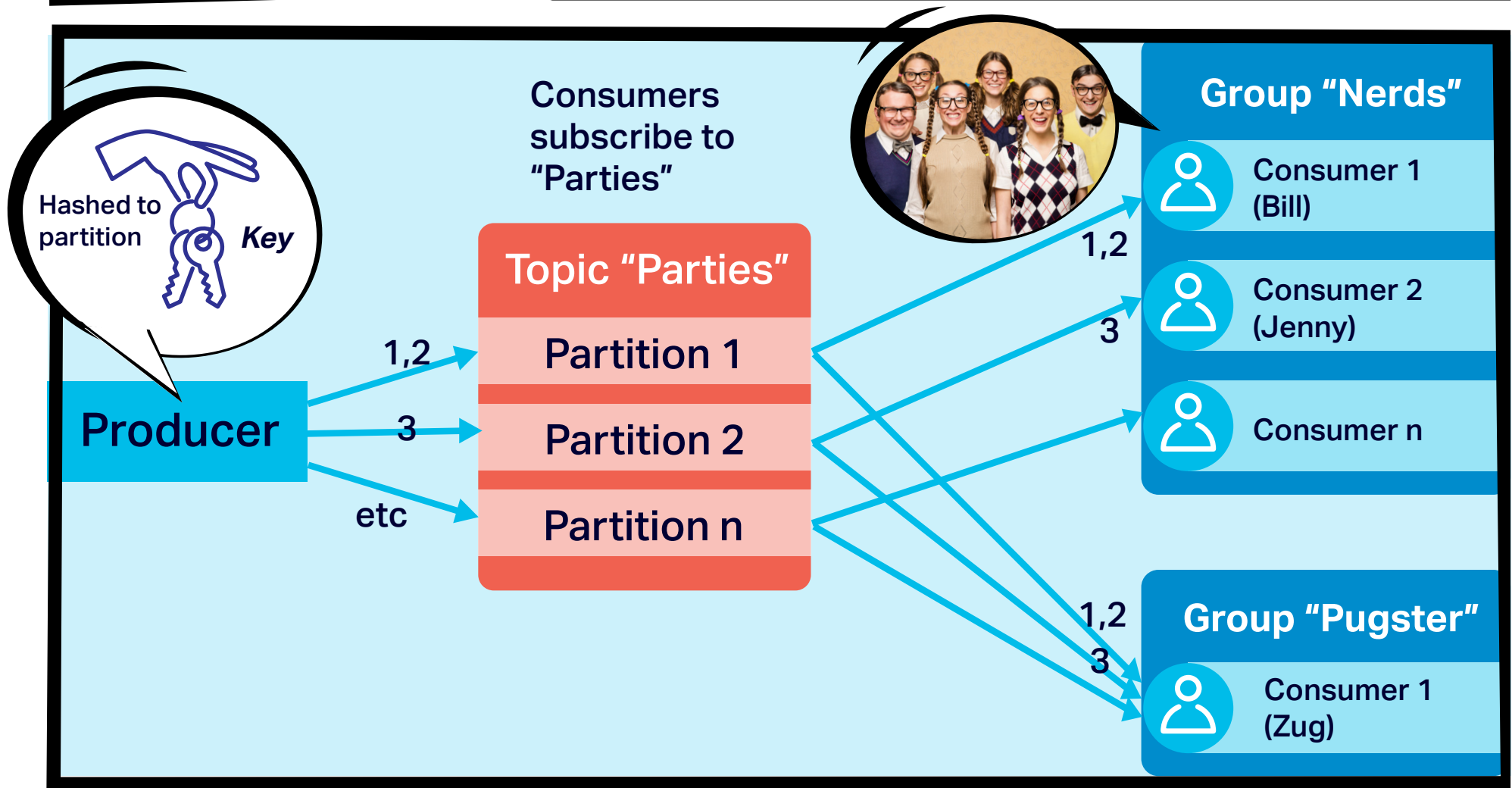


Zug plans
something else
fun instead... A
jam session with
his band



How does it work if *there is a Key?*

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



Here's what happens with a key, assuming that the key is the **"title" of the message** ("Pool Party"), and the value is *invitation or canceled*

1

As before **Both** Groups subscribe to Topic "parties"

2

The Producer sends a record with the **key** equal to "Pool Party" and the **value** equal to "Invitation" to "parties" topic



Key

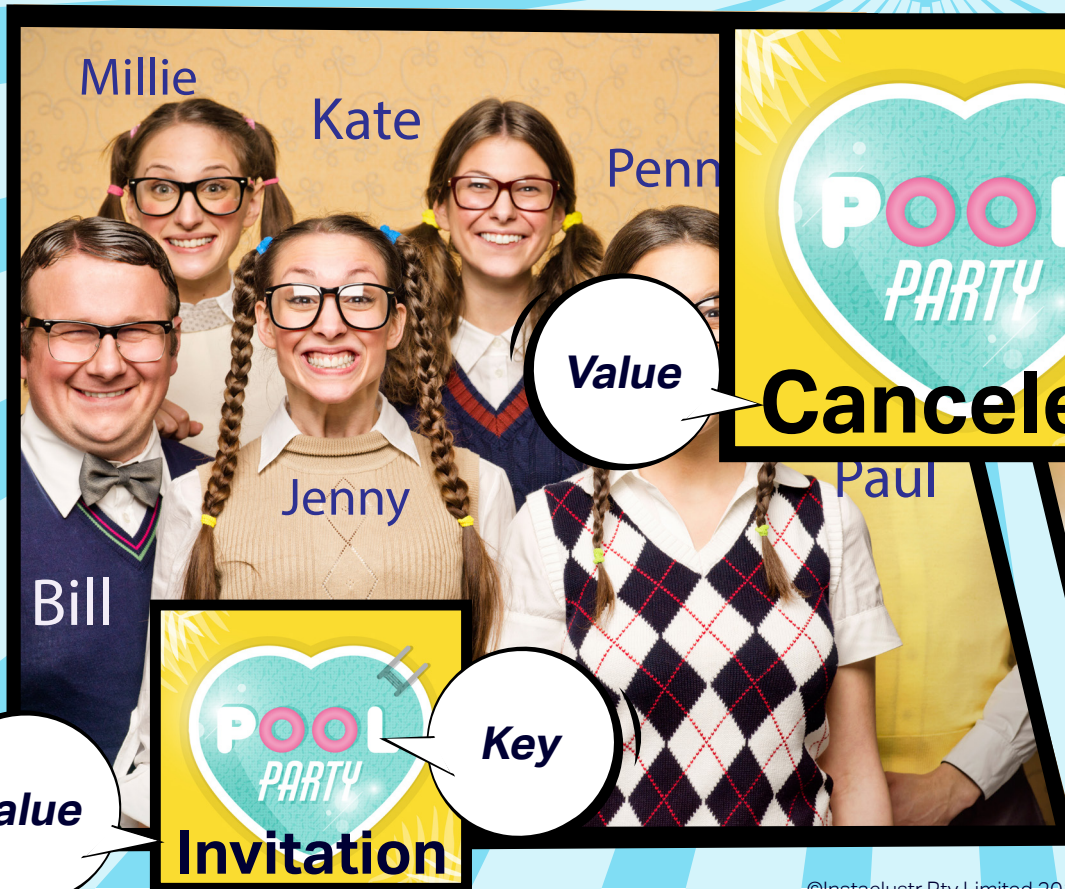
3

As before, Bill and Zug receive a copy of the invitation and plan to attend



4

The Producer sends another record with the **same key** but with the value "cancelation" to "parties" topic



5

This time, Bill and Zug receive the cancellation (the same consumers as the key is identical)



Key

Value



Canceled

Invitation



Key

Value



Canceled

Invitation

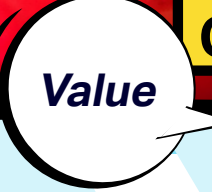
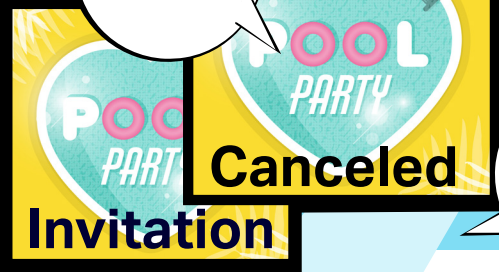
6


The Producer sends out another invitation to a Halloween party. The key is different this time.



7

Jenny receives the Halloween invitation as the key is different and the record is sent to Jenny's partition. Zug is the only consumer in his group so he gets every record no matter what partition it's sent to.





**This time Zug
gets dressed up
and has fun at
the party.**

But wait! There's more—
event reprocessing
(time travel)!

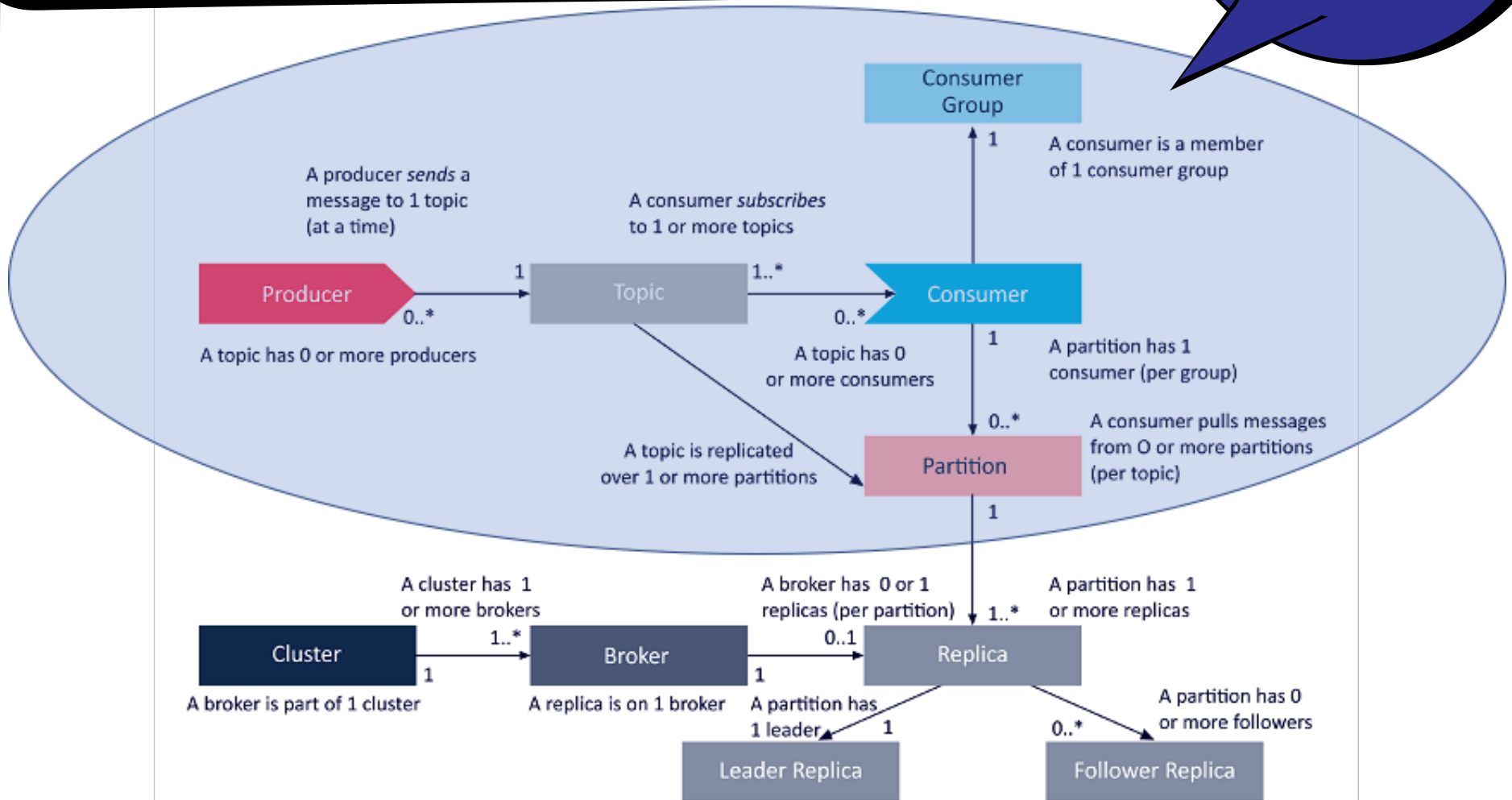


Kafka stores message streams on disk,
so Consumers can go back and request the
same messages they've already received, earlier
messages, or ignore some messages etc.

Image: Shutterstock.com

In this Visual Introduction we used Producers, Topics, Partitions, Consumers, and Consumer Groups. There's still a lot more of Kafka to explore, including how Kafka provides replication, and the Connect and Streaming APIs.

Here's a UML diagram with the main Kafka components



***That's it for this
short visual
introduction to
Apache Kafka.***

For more information
please have a look at the
Apache Kafka docs, the
Instaclustr Blogs, and check
out our free Kafka trial.

There's also another
"Visual" introduction
to Kafka that I found, a
video "*Understanding
Kafka with Lego*"!

Apache Kafka:

<https://kafka.apache.org/>

***Another visual introduction
(Kafka with Lego!) video:***

<https://youtu.be/Q5wOegcVa8E>

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-write-read-scalability/>

Kongo—Kafka IoT Logistics Application Blog Series

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

Anomaly Detection With Kafka and Cassandra, New Blog Series

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

Instaclustr's Managed Kafka (Free Trial)

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