databases, data engineering & big data

Kafka & event driven architectures

ESME SUDRIA

Luc Marchand - luc.marchand.pro@proton.me
Maxence Talon - maxencetallon@gmail.com

Global Syllabus

01 Introduction and main concepts

02 SQL, set up env and practical work

03 NoSQL world

04 Introduction to Big Data & Data Engineering

05 **Kafka & event driven architectures**

06 Spark & Delta

07 Warehouse, DBT & BI

08 IA - MLOps & RAG

Course syllabus

01 What is and why consider event world ?

02 Key concepts

03 Kafka

04 Design patterns

05 Kafka ecosystem

Before starting the course

Go to https://github.com/Esme-Sudria-Database/lab-kafka

- Clone the repository (git clone <https-github-url>
- Run make start to launch the stack (and download all the needed images)
- Listen to this interesting course during downloading ;-)

01

What is and why consider event world ?

So... what is an event ?

- An event is **anything** that takes place in the world.

- An event is **immutable**, ie it can't be deleted.
  - If you want to cancel an event, you have to generate another event
  - It is so a **trustable** information.

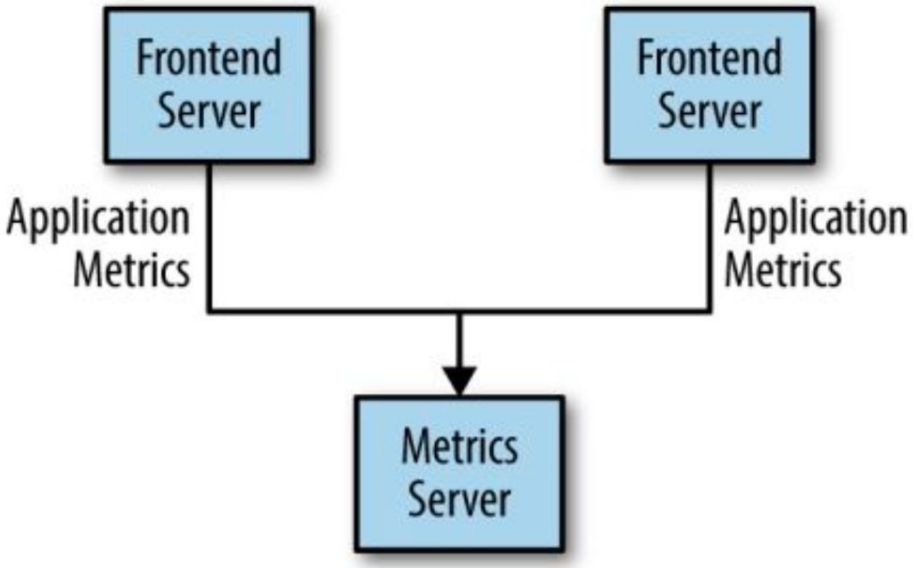
- It is (usually) composed of a *header* and a *body*.
  - It is often written as json

Why may you want to use an architecture based on events ?

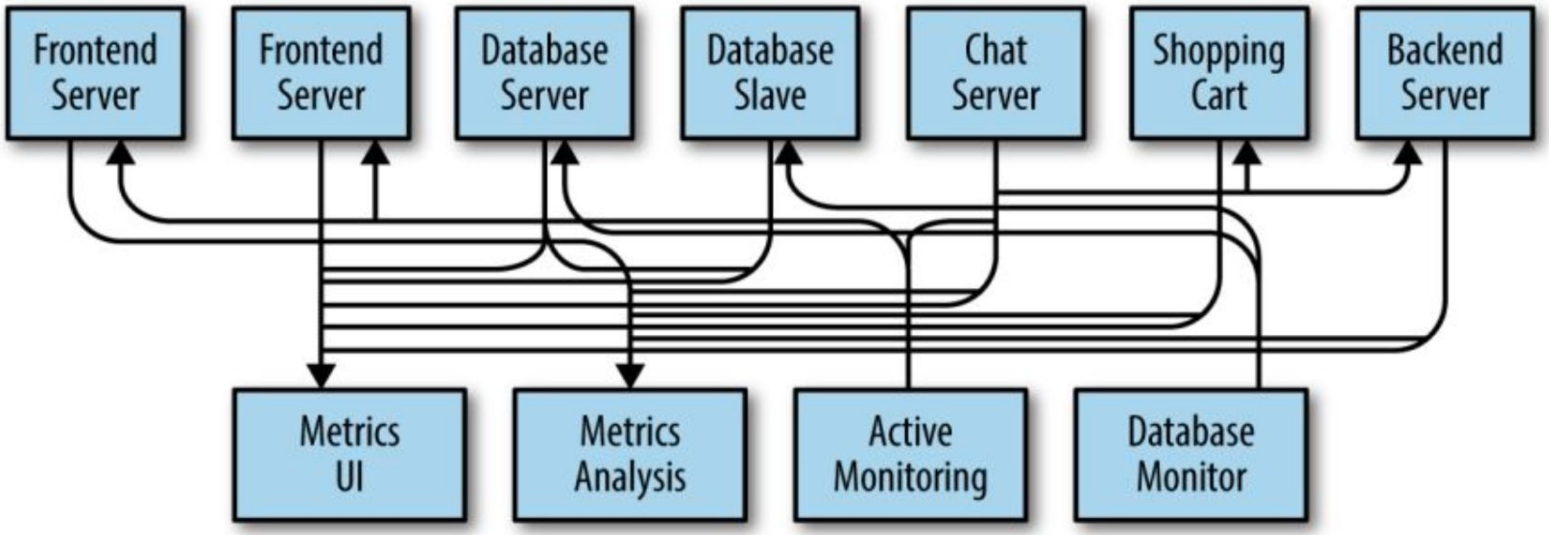
- You may want to:
  - aggregate different kinds of data sources
  - decouple elements of your information system
  - react almost in real-time based on events from your system
  - migrate data from A to B easily
  - show a live dashboard
  - ...

A situation as an example

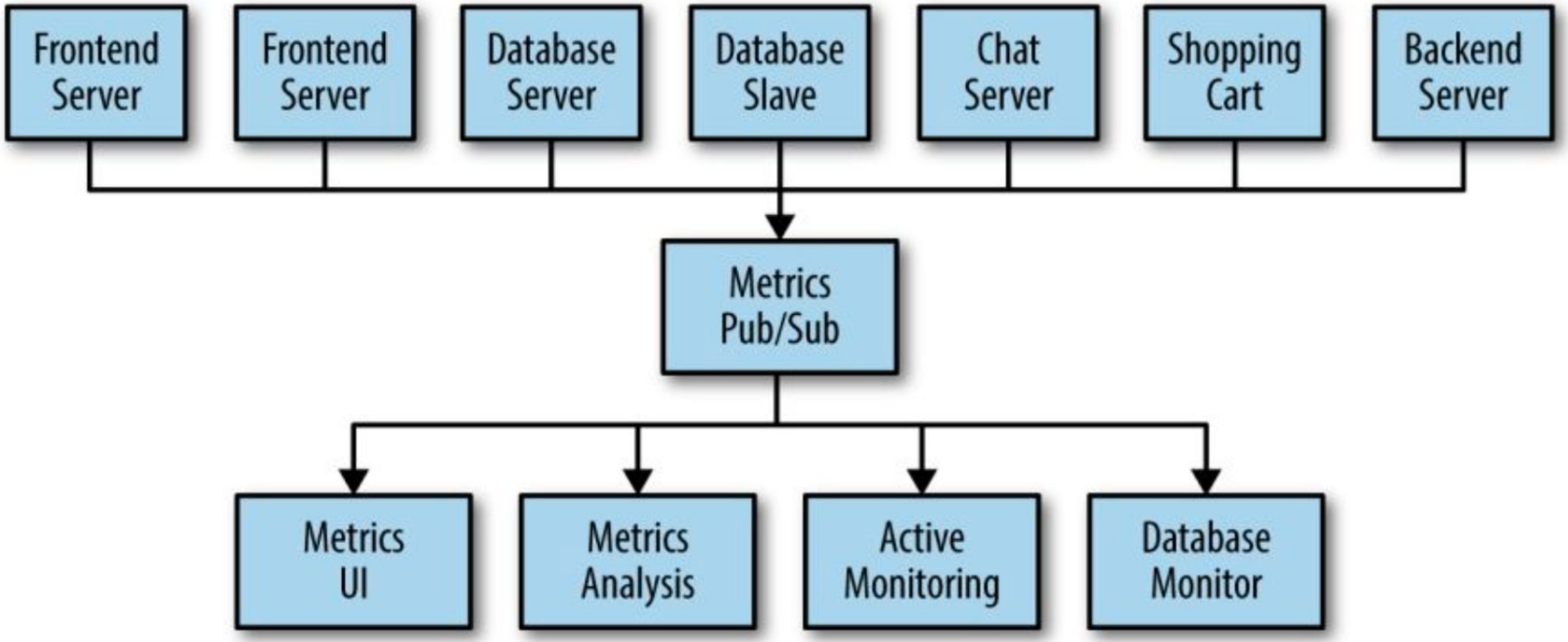
Un projet qui débute



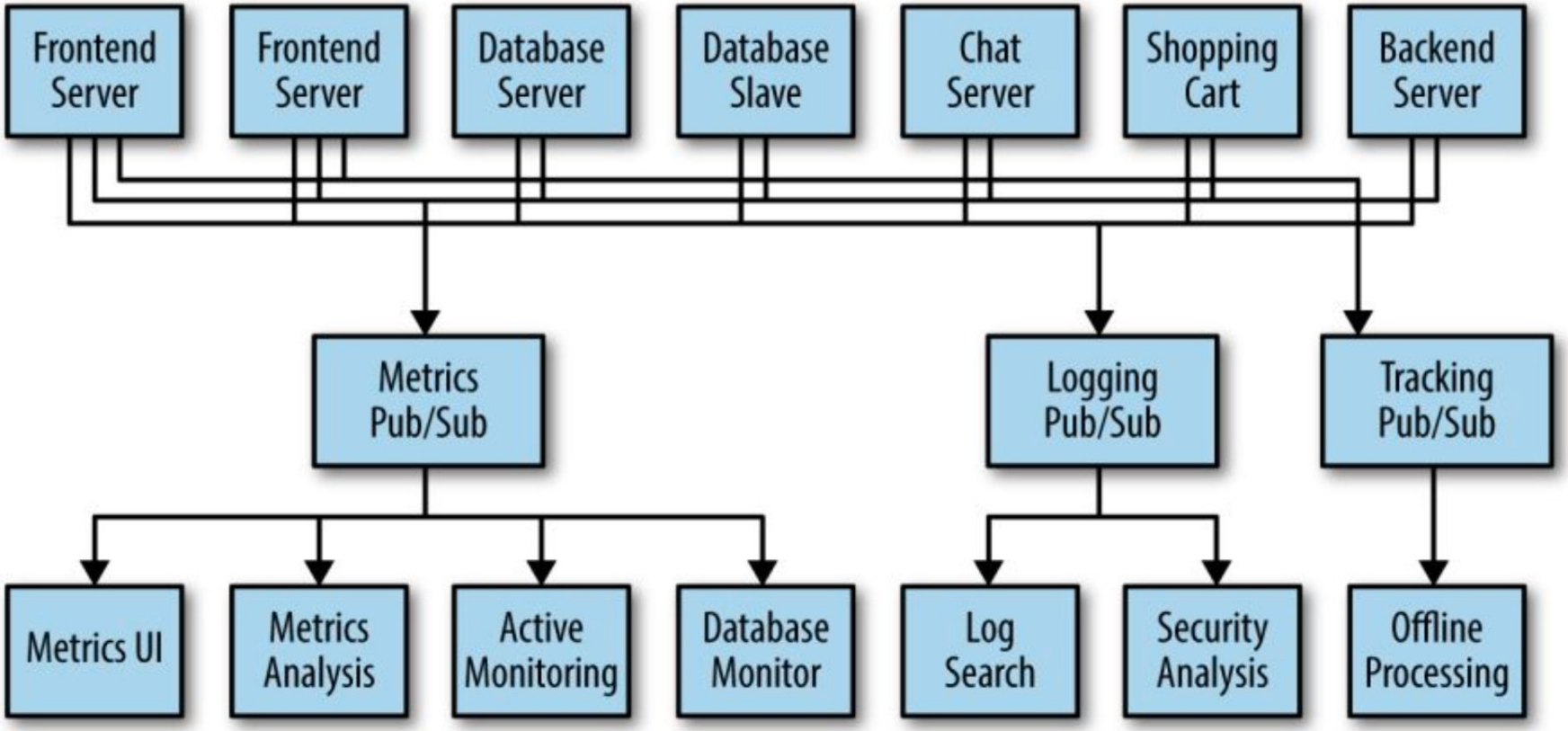
Un Capharnaüm naissant



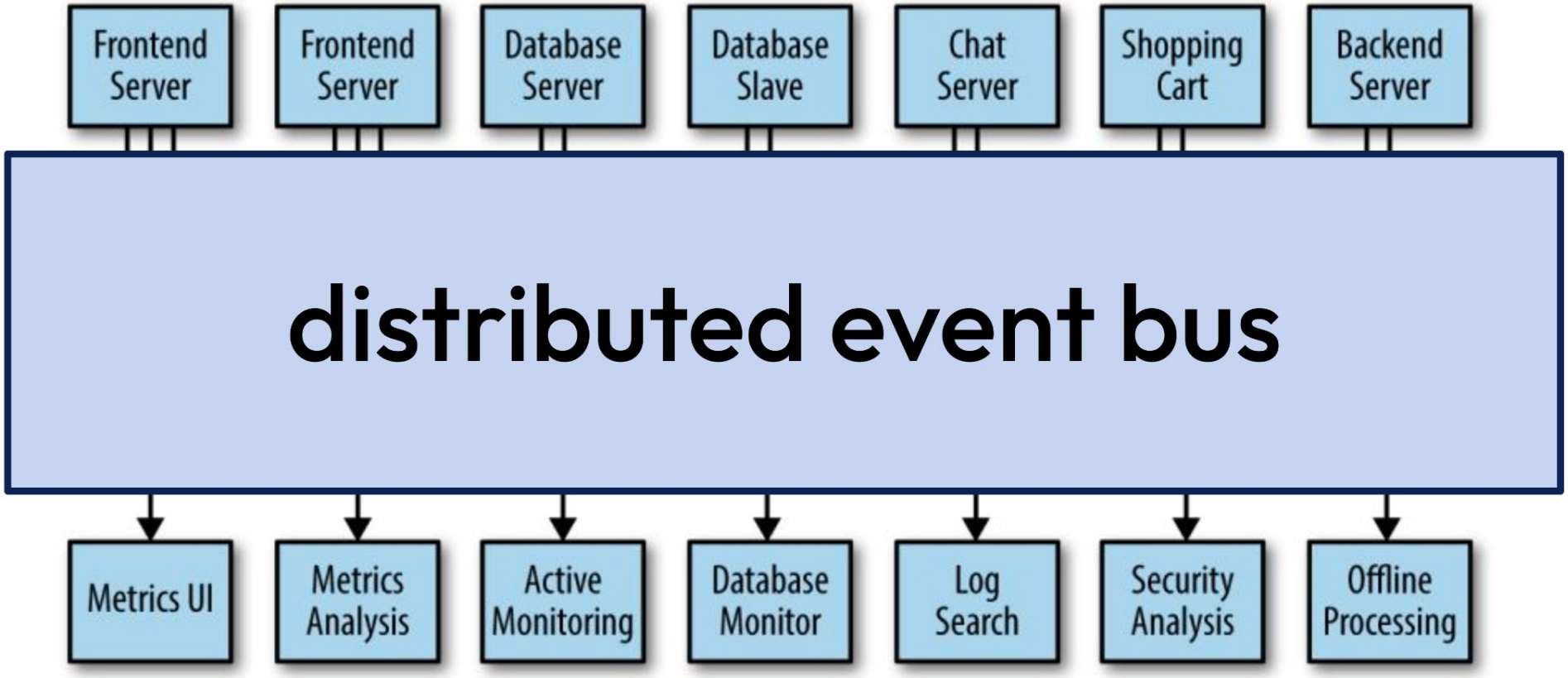
Une boîte au milieu pour nettoyer tout ça



Un bordel naissant de nouveau...



A distributed event bus to run them all



02

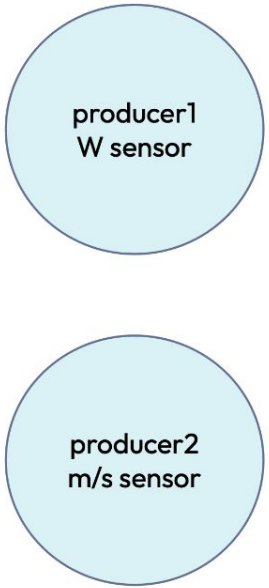
Key concepts

What are the key concepts of a distributed event bus?

- Publish / subscribe pattern
- Queuing vs streaming
- High availability & resilience

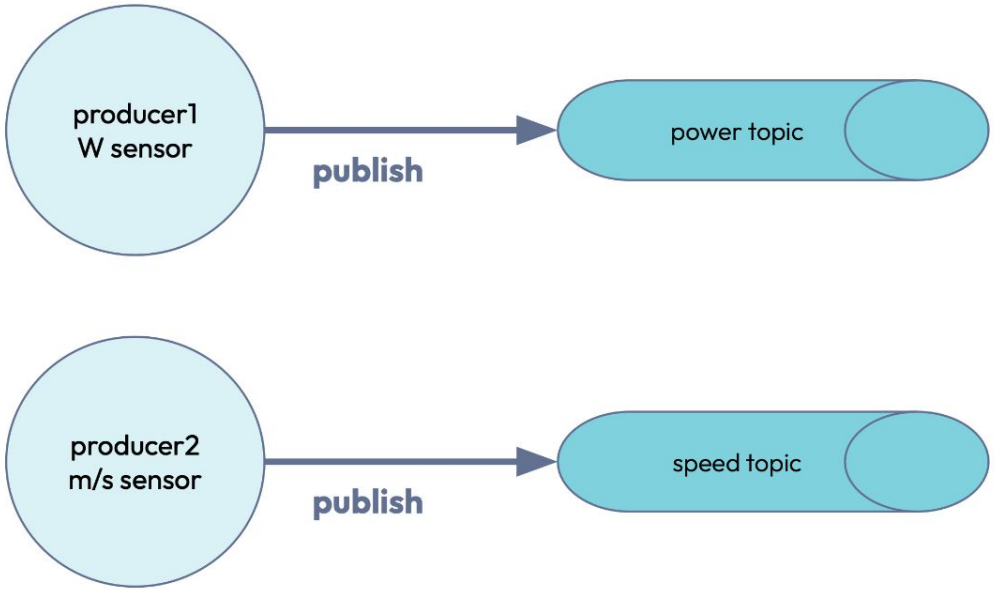
Publish / Subscribe pattern

Let's start with **producers**...



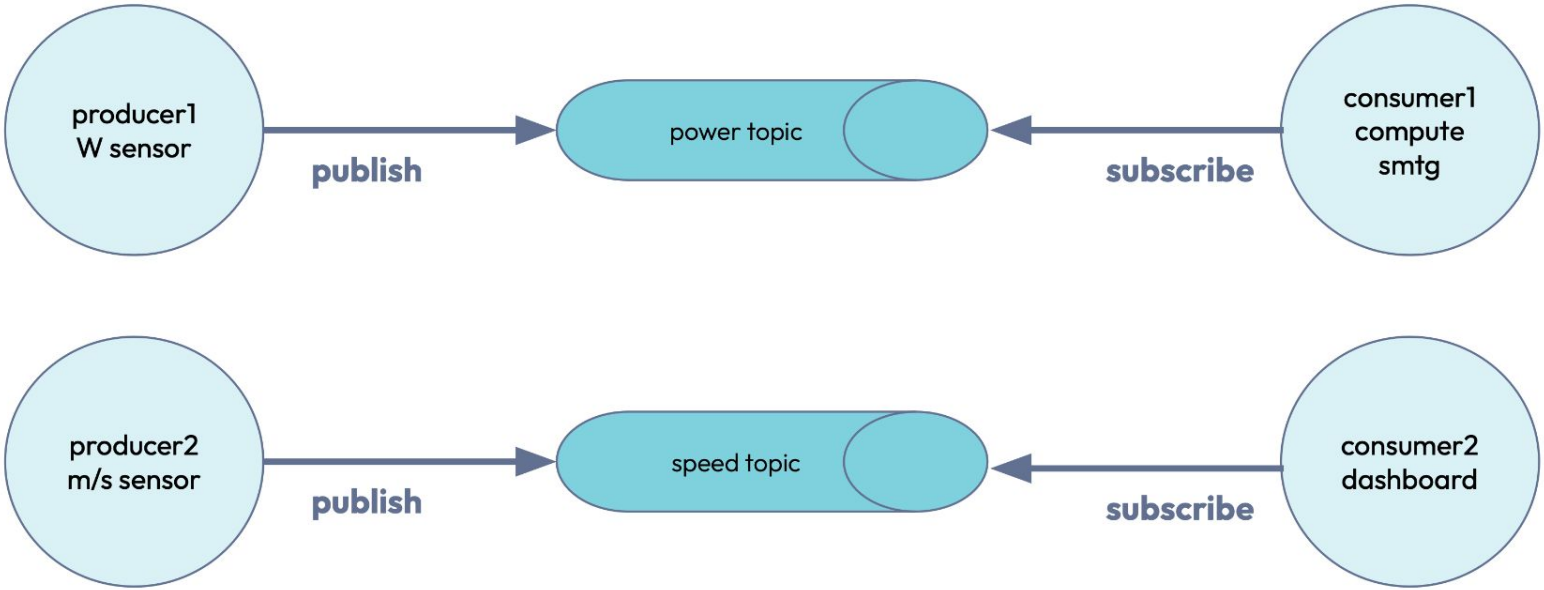
Publish / Subscribe pattern

... which publish into **topics**...



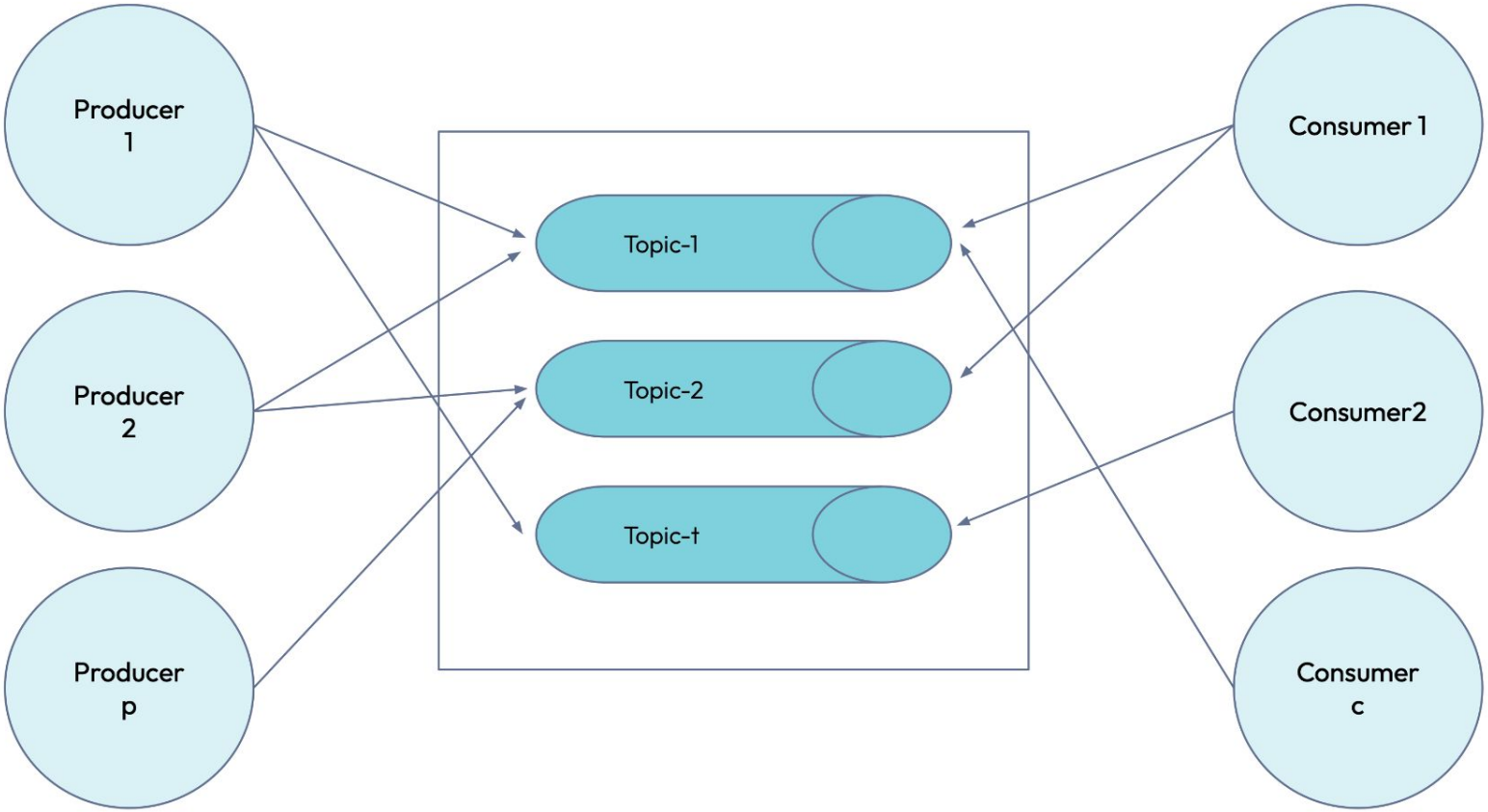
Publish / Subscribe pattern

... and **consumers** subscribe (or read) from topics



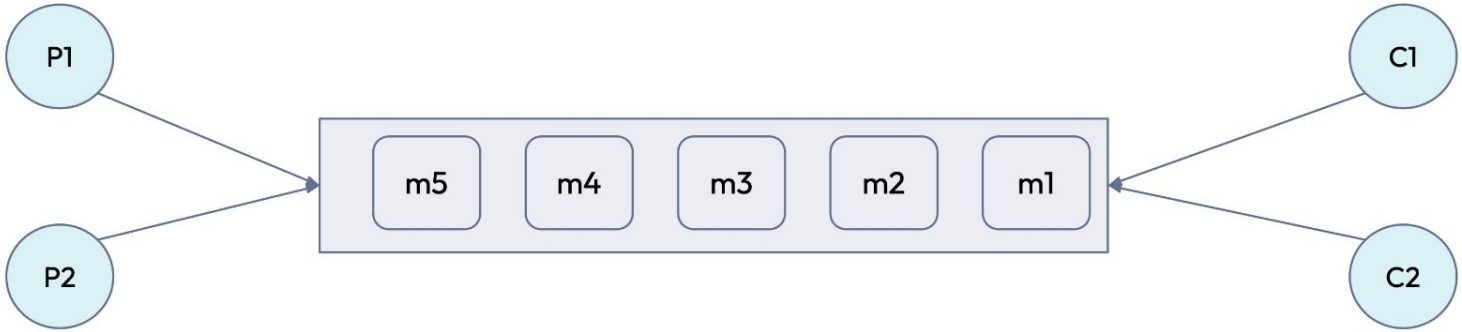
Publish / Subscribe pattern

A more generic view



Queuing vs Streaming

Queuing



- As soon as a consumer read a message, this one is deleted from the queue
  - Multiple consumers can't read the same message from the queue
- This pattern is useful to decouple components and can handle high throughput

Queuing

Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes

Queuing

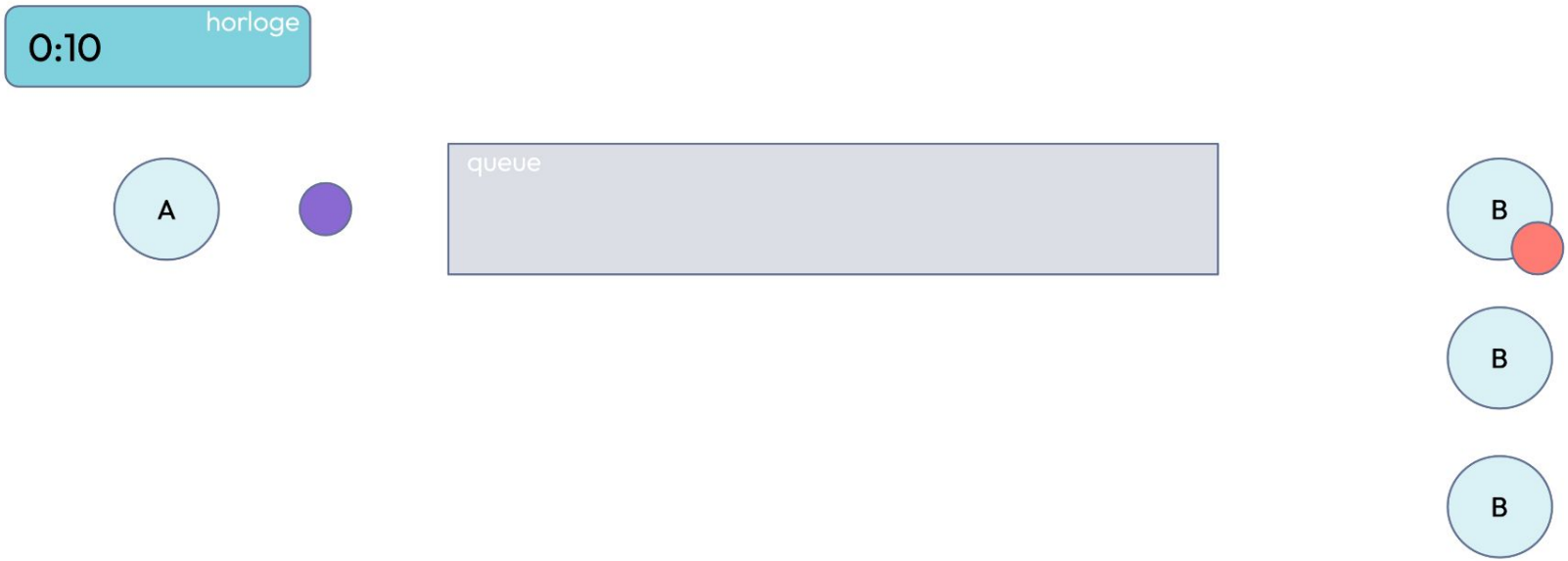
Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes

Queuing

Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes



Queuing

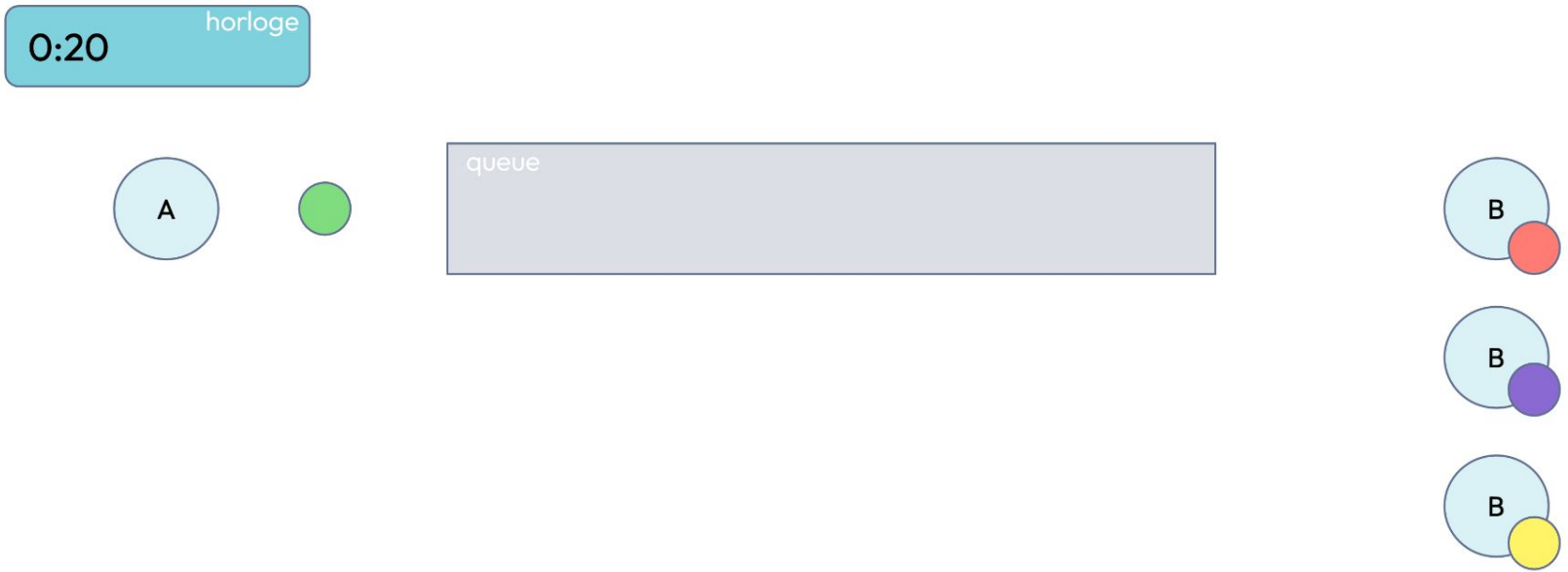
Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes

Queuing

Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes



Queuing

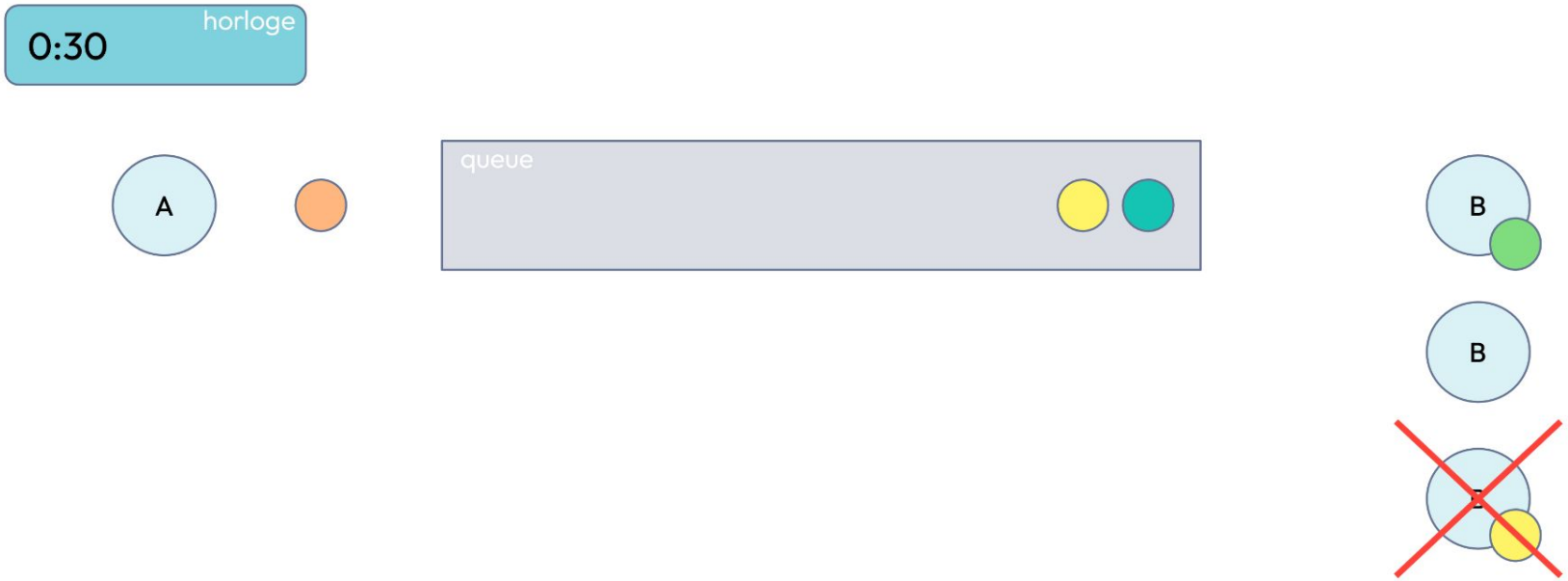
Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes

Queuing

Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes



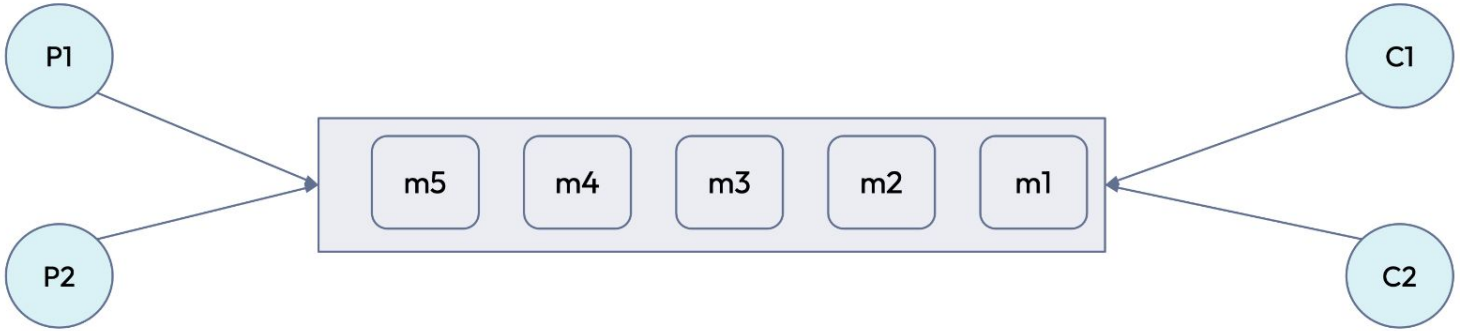
Queuing

Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- B vérifie les demandes de vérification toutes les 5 secondes

Queuing vs Streaming

Streaming

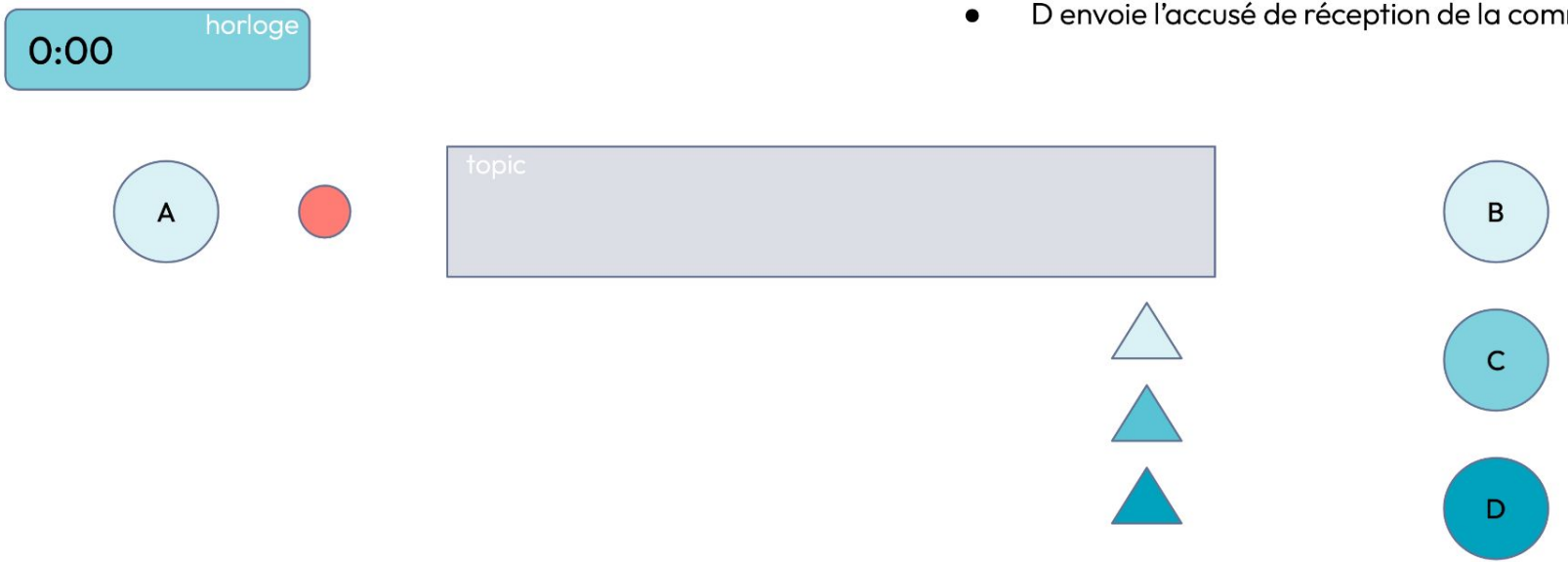


- In a streaming architecture, both consumers C1 and C2 will consume all messages in the queue
  - Each consumer implements a different application
- This pattern is useful when using the same data for different purposes and / or allowing replays

Streaming

Example

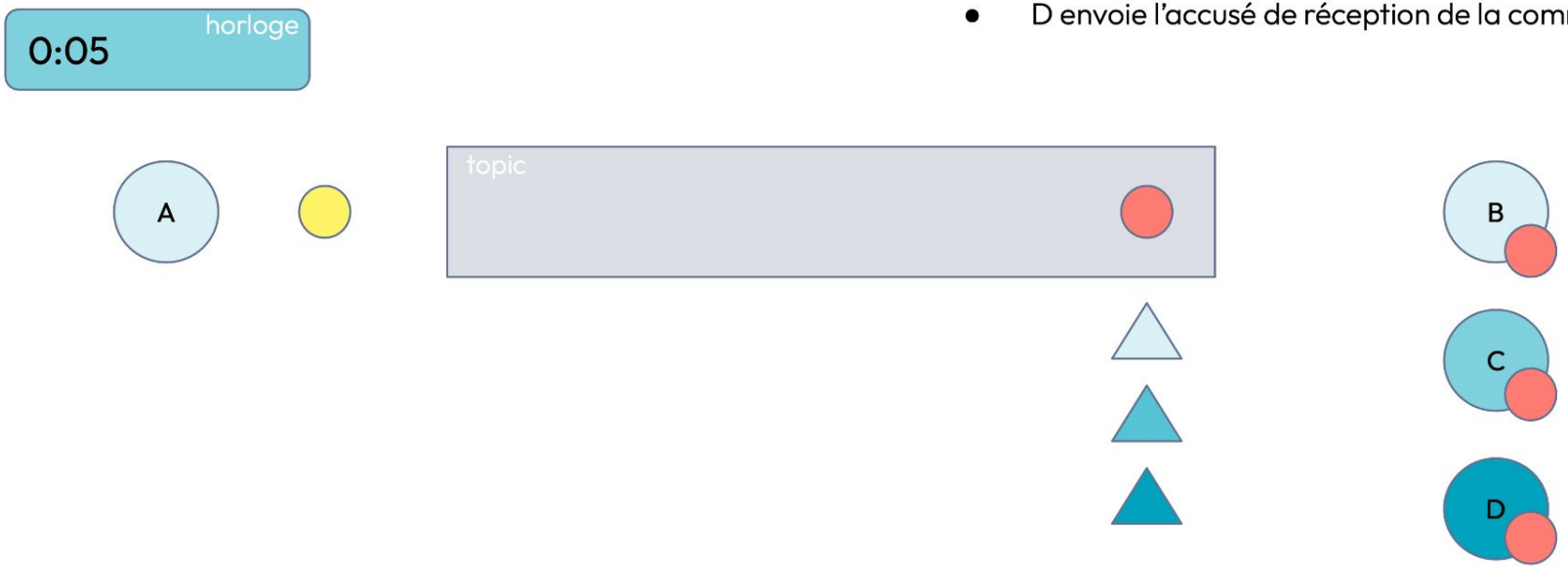
- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s



Streaming

Example

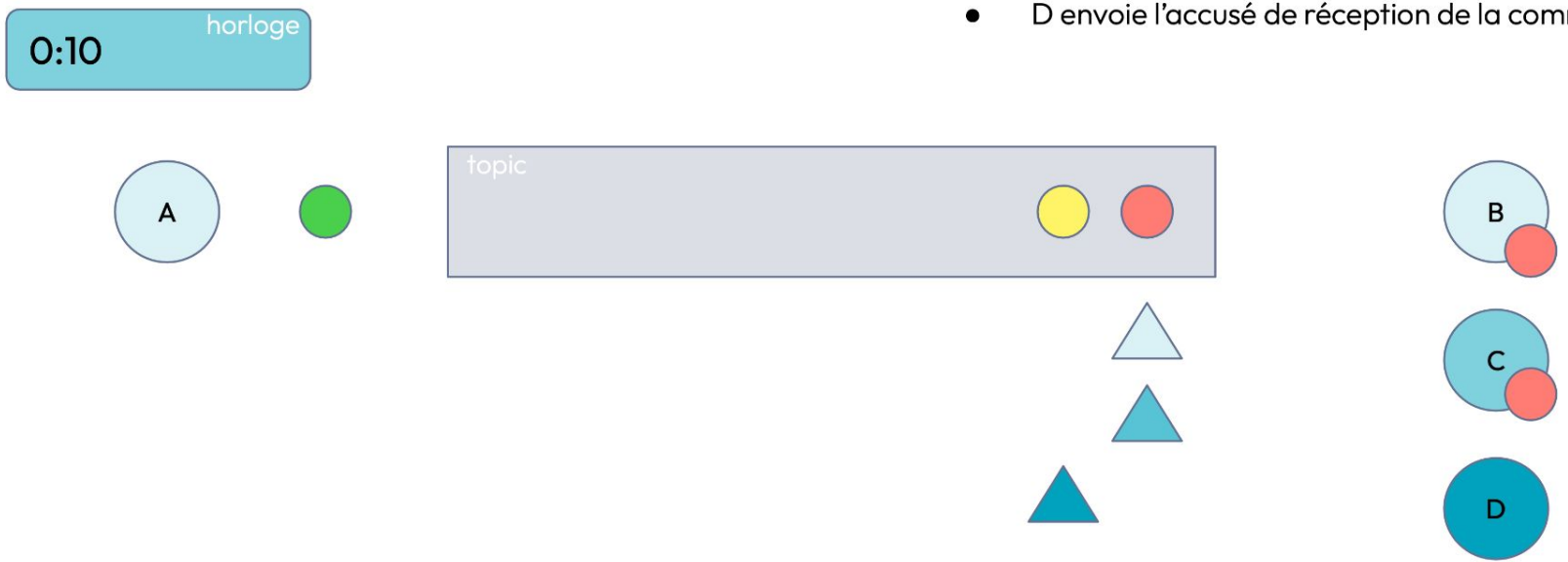
- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s



Streaming

Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s



Streaming

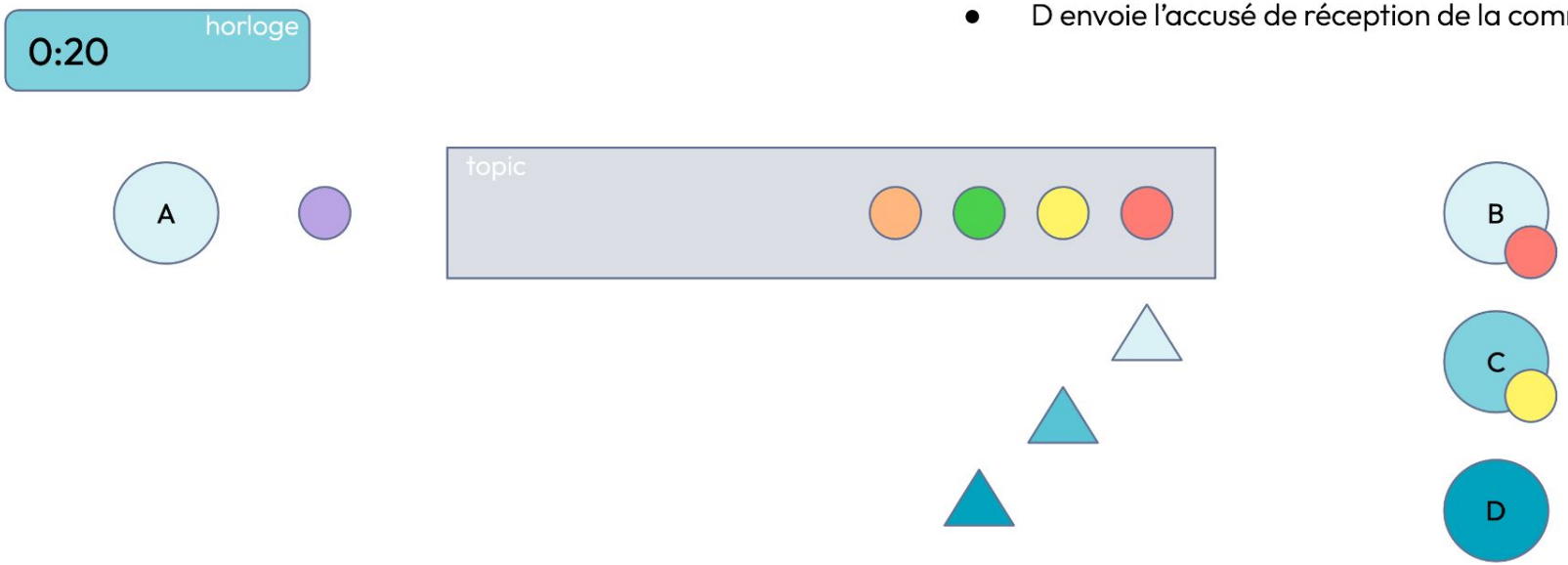
Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s

Streaming

Example

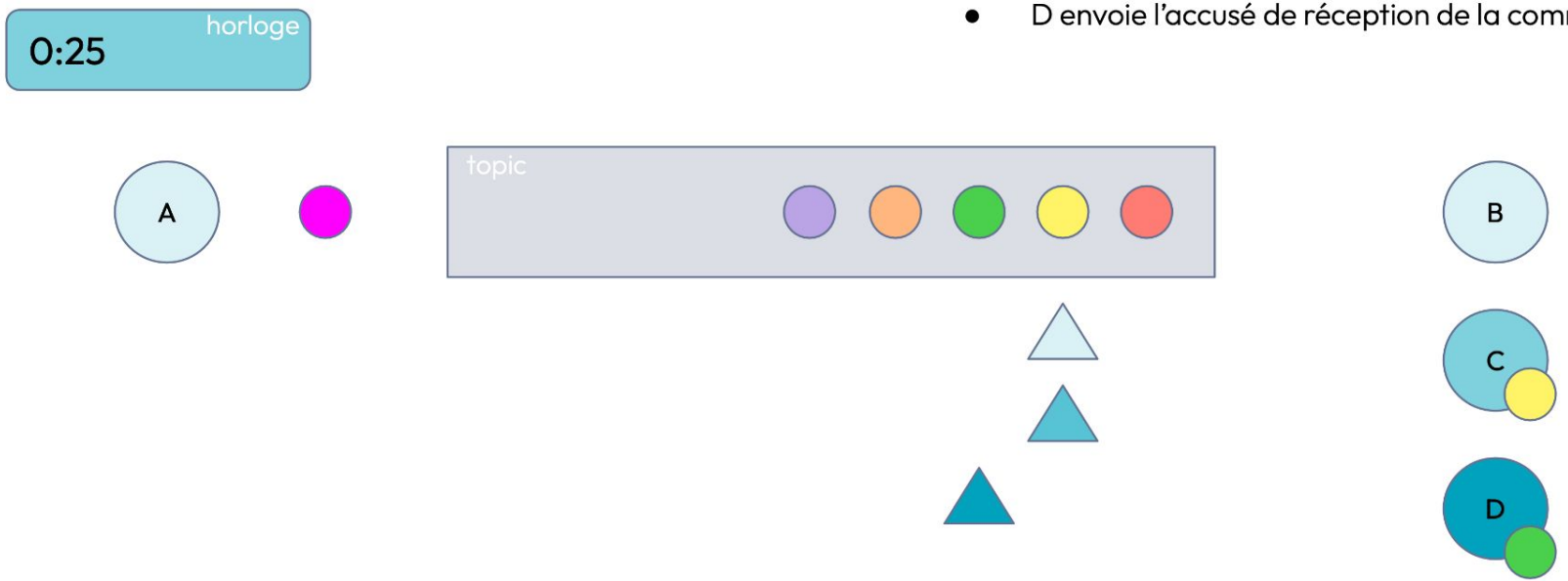
- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s



Streaming

Example

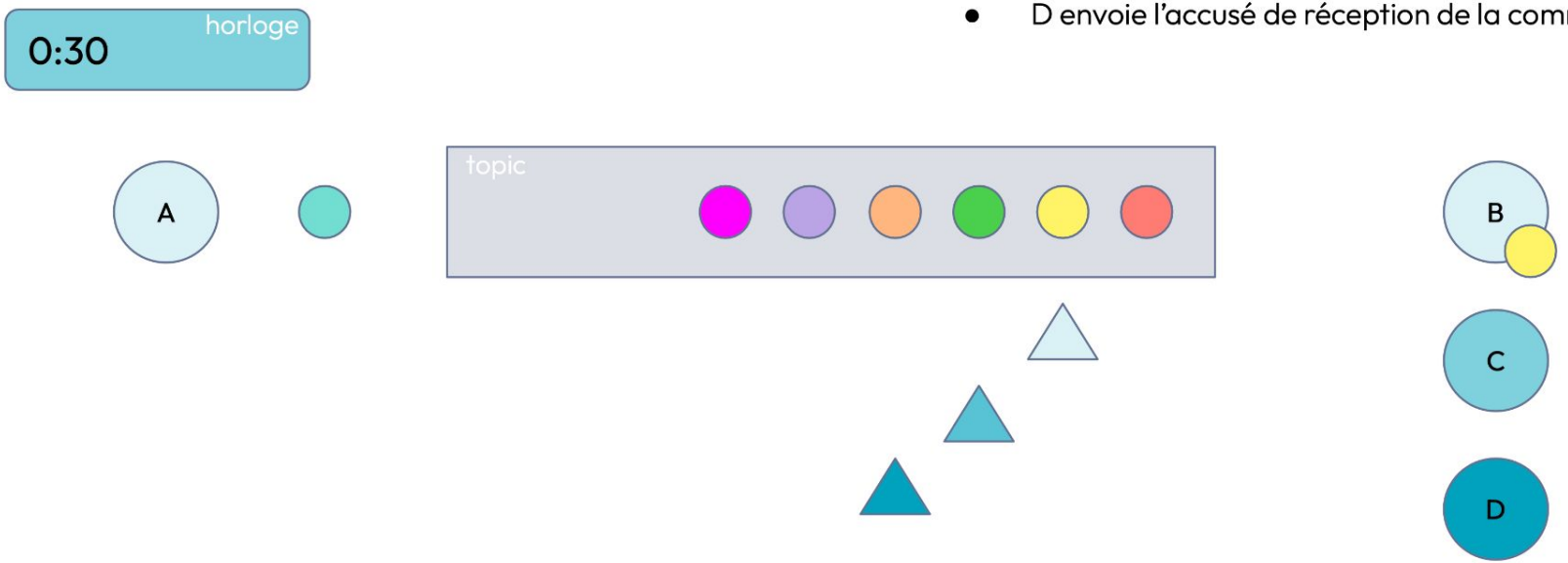
- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s



Streaming

Example

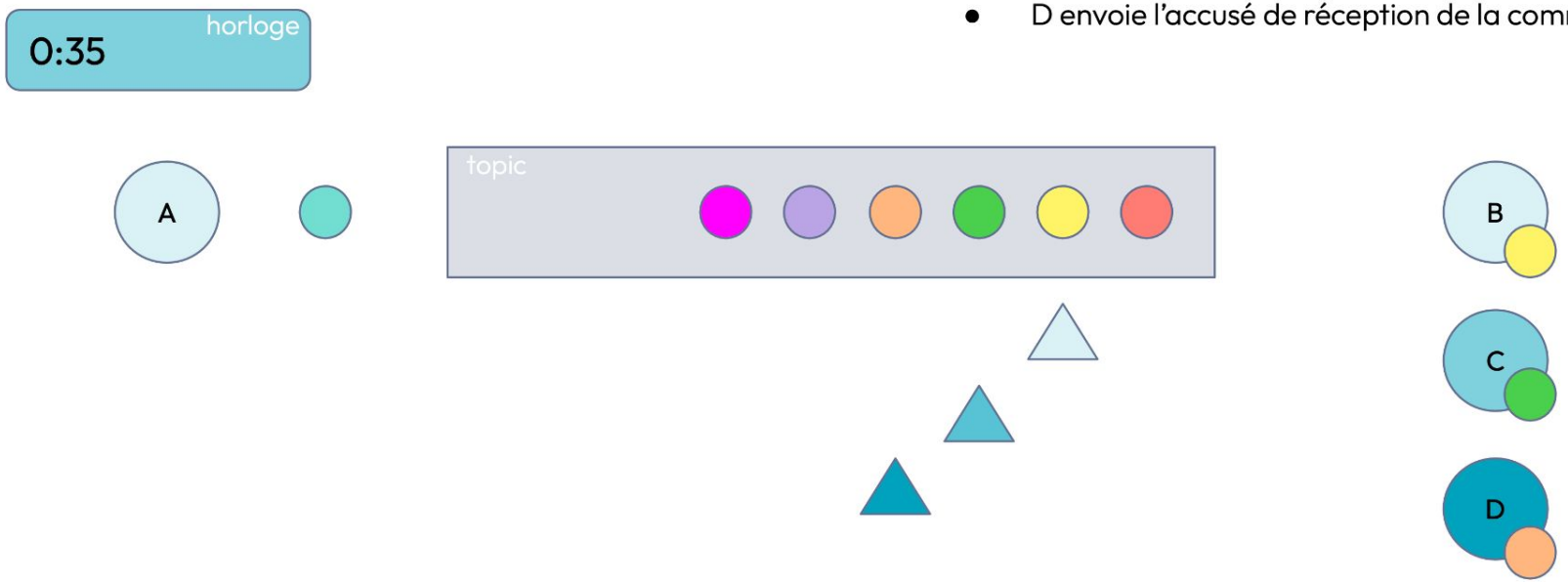
- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s



Streaming

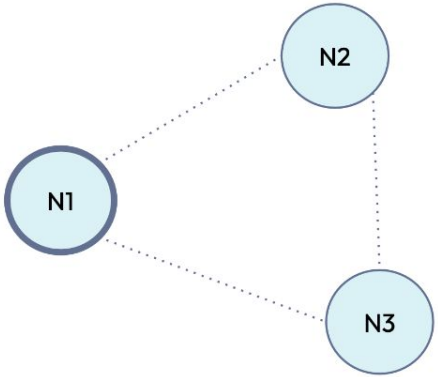
Example

- Un client commande un produit sur A toutes les 5s
- B vérifie en 20s la solvabilité du client pour la commande
- C vérifie la disponibilité du stock en 10s
- D envoie l'accusé de réception de la commande en 5s



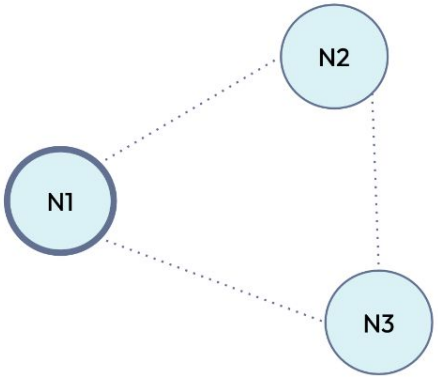
High-availability & resilience

Who says *distributed* implicitly says *HA & resilience*



High-availability & resilience

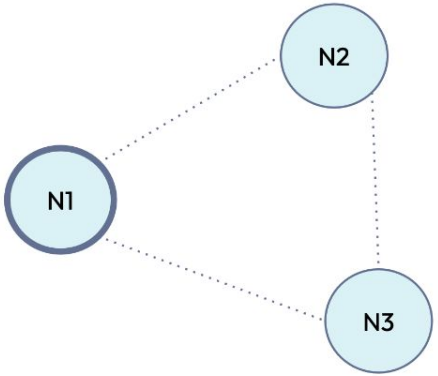
Who says *distributed* implicitly says *HA & resilience*



- What happens to my data if :
  - A main / replica node crash or does not respond
  - there is a network failure
  - a copy to replica fails
  - ...

High-availability & resilience

Who says *distributed* implicitly says *HA & resilience*



- What happens to my data if :
  - A main / replica node crash or does not respond
  - there is a network failure
  - a copy to replica fails
  - ...

→

Consensus Algorithm
(quide / animation)

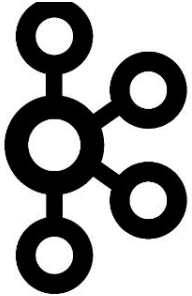
Resiliency

Replication strategy

03

Kafka

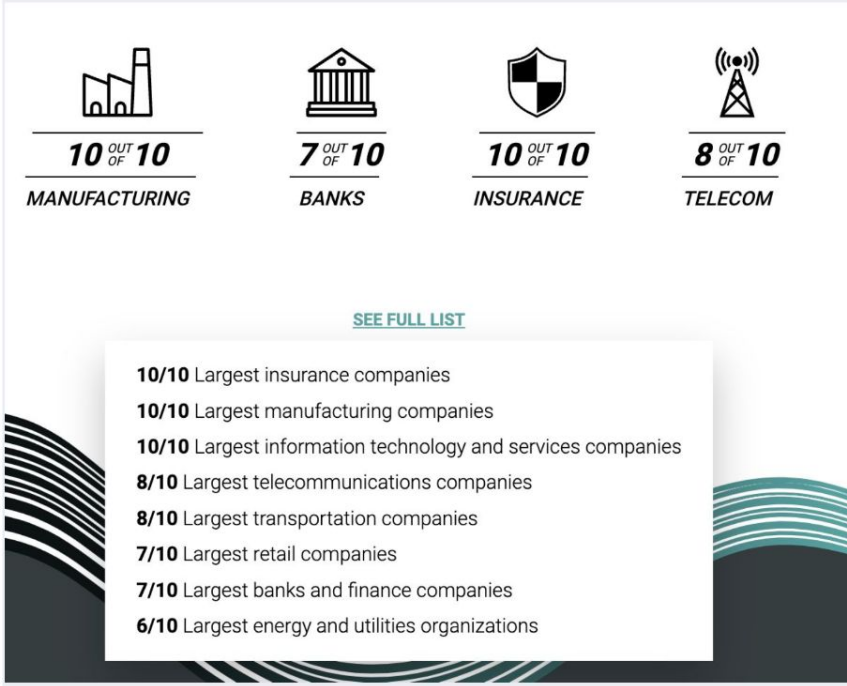
What is Kafka ?



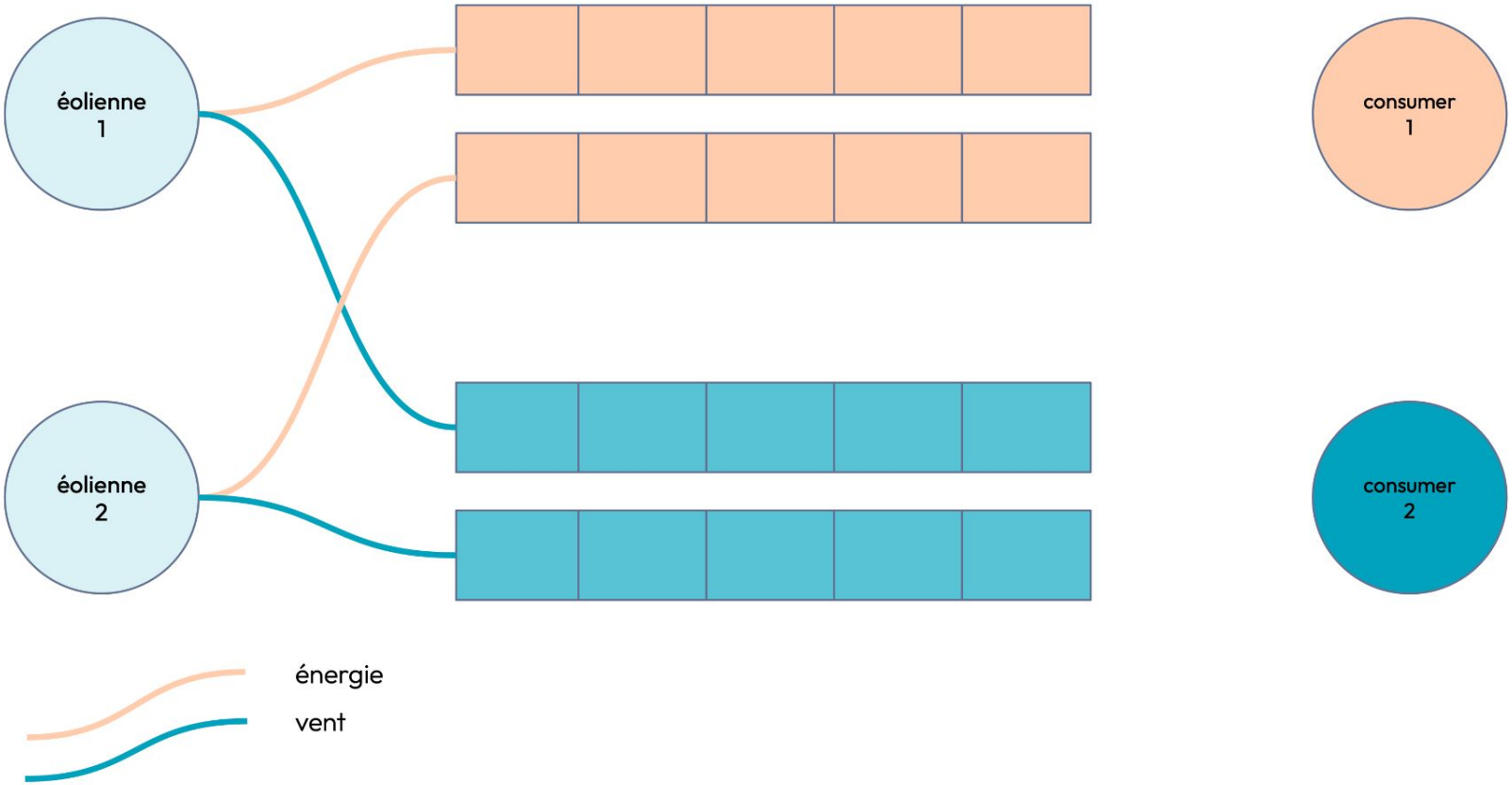
- message distributor system
- open-source stream-processing software
- provide a unified, high-throughput, low-latency platform for handling real-time data feeds
- originally developed at LinkedIn, started in 2008
- open-sourced in 2011 at Apache Foundation

Kafka reach an industrial maturity

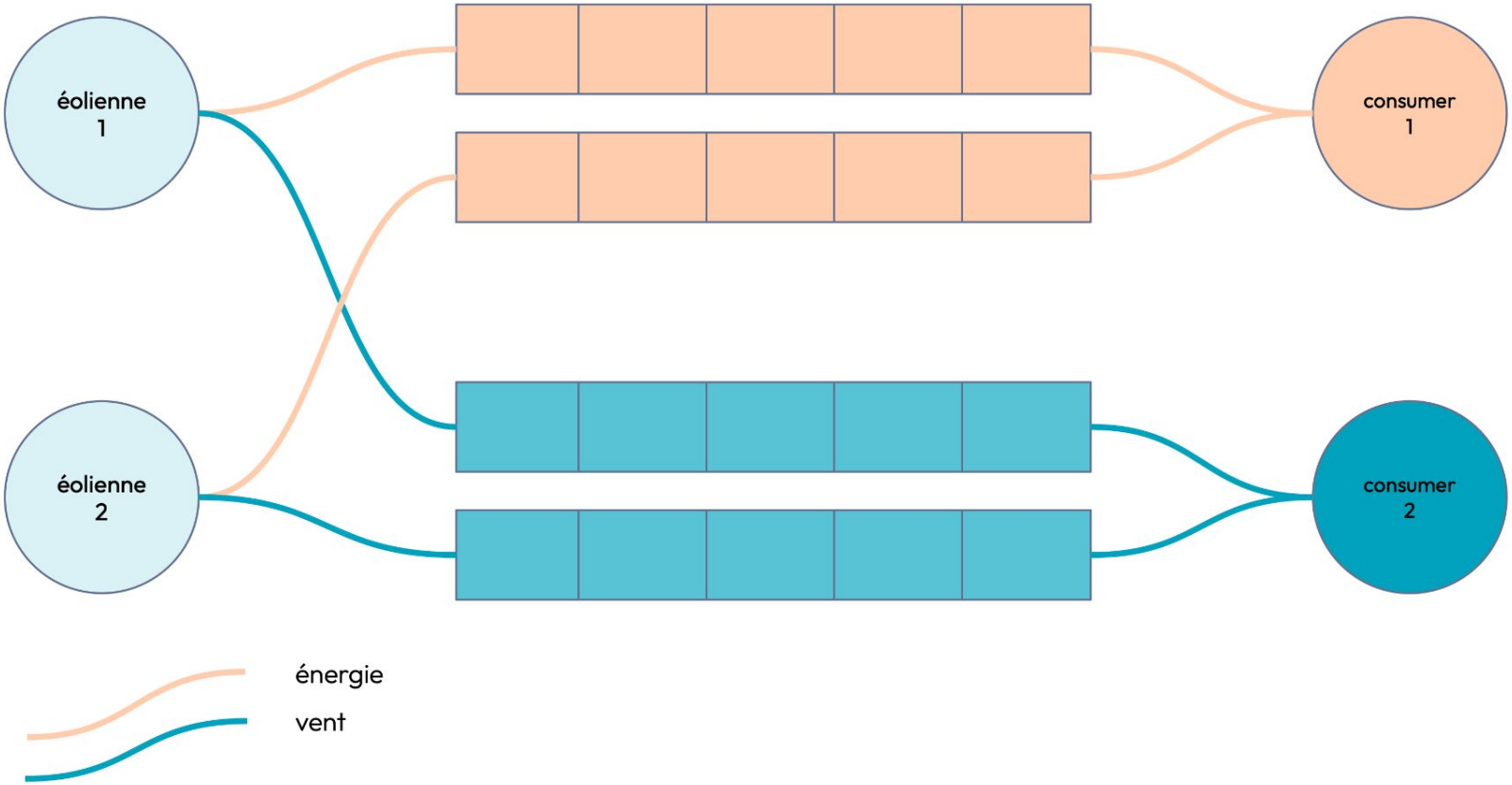
- Backed by Confluent
- Used in big companies (Netflix, Uber, Spotify, Goldman Sachs, ...) →
- Inspiration for cloud providers
- Now on the cloud with Confluent Cloud



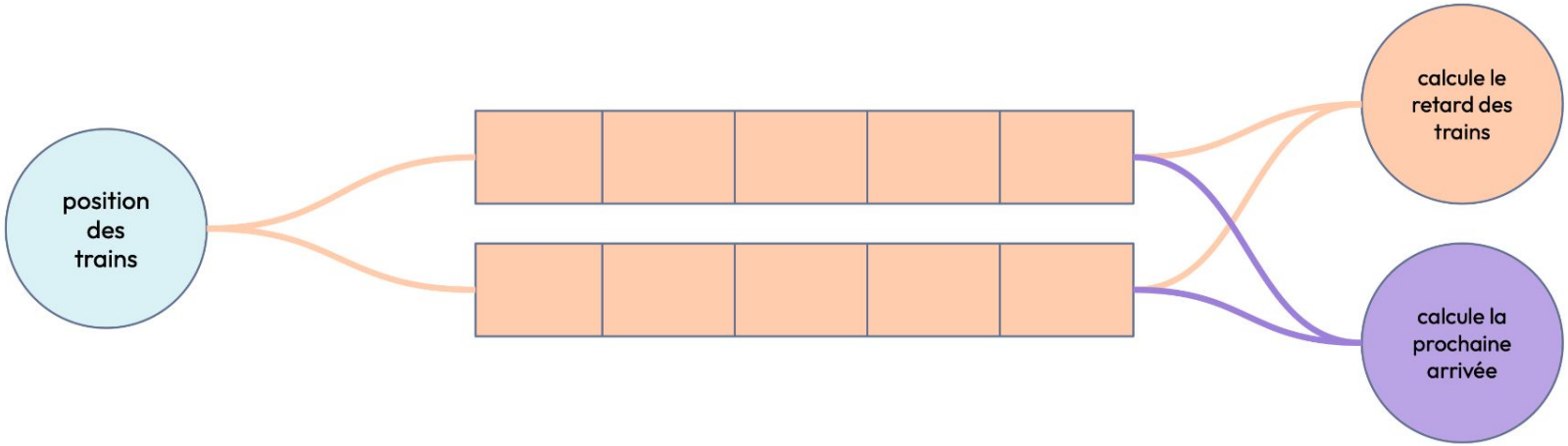
Un cluster kafka concentre les flux de plusieurs applications



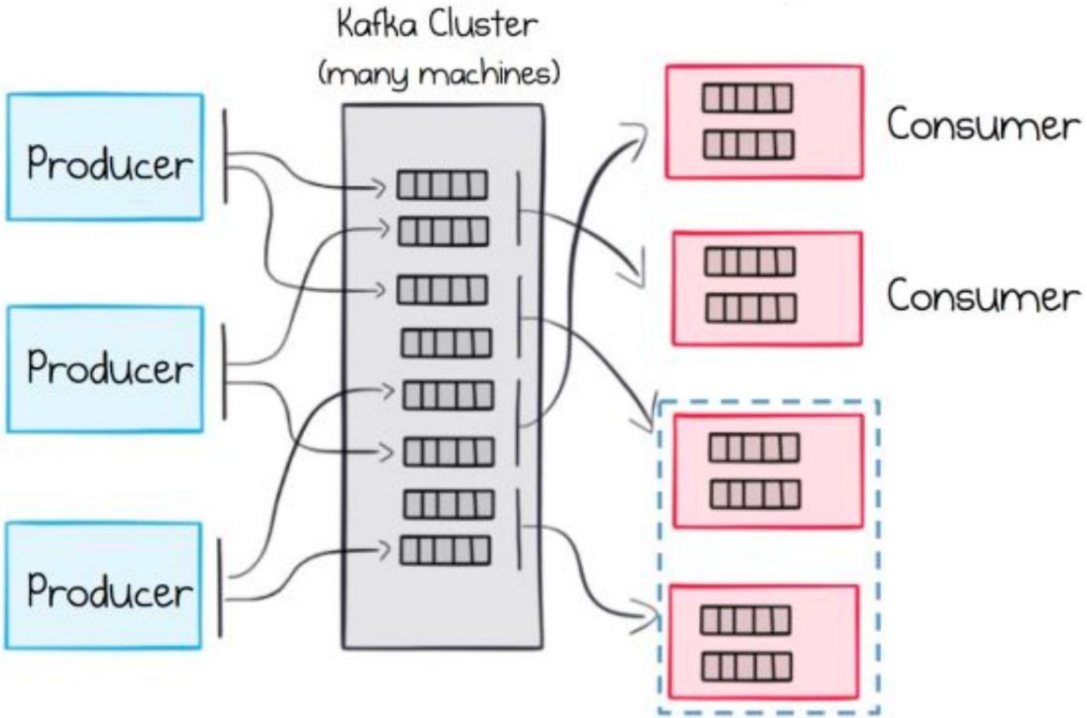
Un consumer traite un flux en continue



Plusieurs consumers peuvent consommer le même flux en parallèle

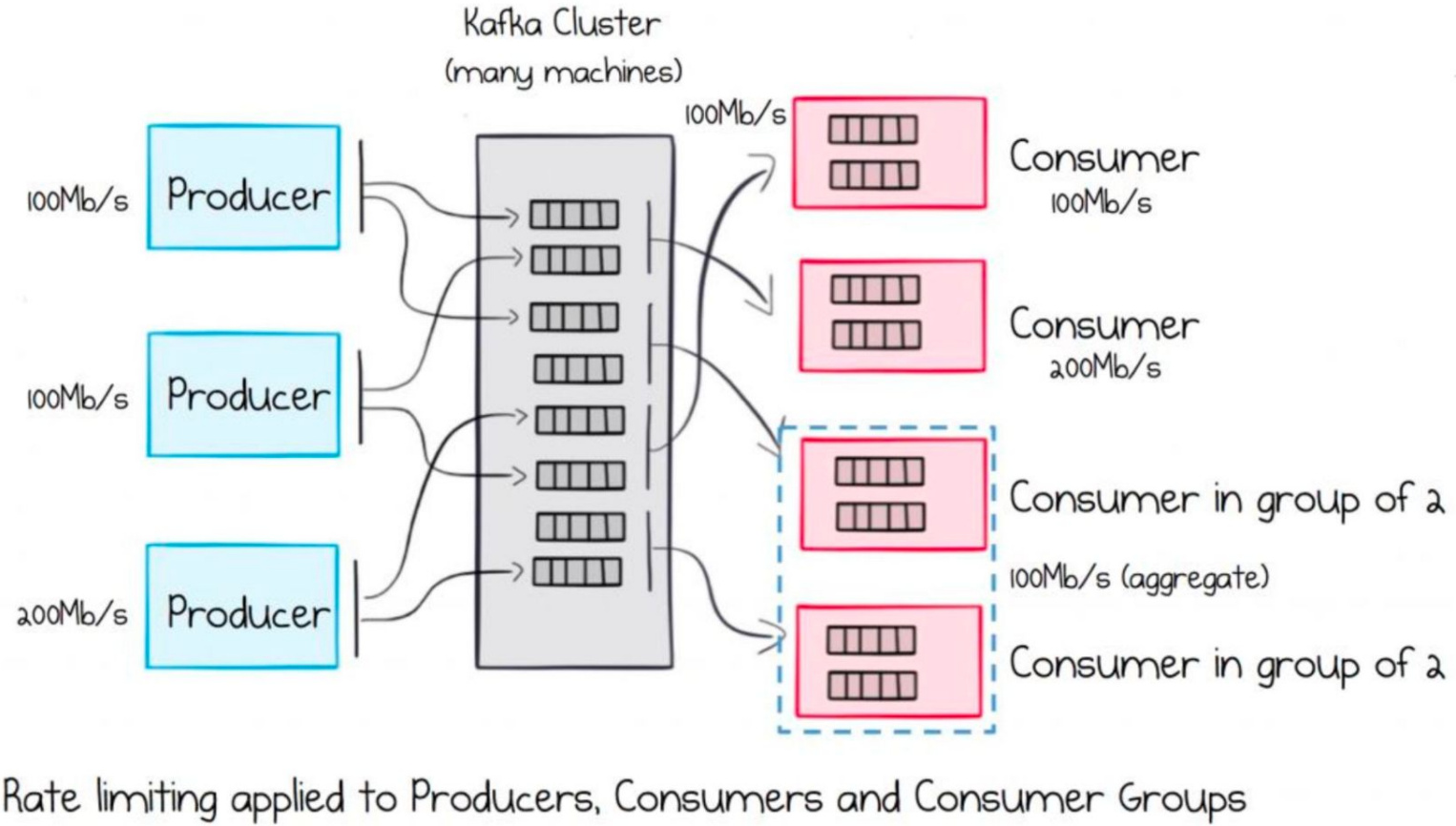


Kafka to scale out



Producers spread messages over many partitions, on many machines, where each partition is a little queue. Load balanced consumers (denoted a Consumer Group) share the partitions between them.

What about throughput ?



Rate limiting applied to Producers, Consumers and Consumer Groups

DOJO

Before starting this hands-on

Go to https://github.com/Esme-Sudria-Database/lab-kafka

- Clone the repository (git clone <https-github-url>
- Run make start to launch the stack (and download all the needed images)
- Listen to this interesting course during downloading ;-)

What we will do

1. Create a topic to handle all our messages

2. Open one terminal to simulate a producer

3. Open another terminal to simulate a consumer

1. Create a topic

The usual command is this one :

*kafka-topics *

*--bootstrap-server broker:9092 *

*--topic <TOPIC_NAME> *

*--create *

*--partitions 3 *

--replication-factor 1

Open a terminal and do *make create-topic*, which we will do the job.

2. Be a producer

The usual command is this one :

*kafka-console-producer *

*--bootstrap-server broker:9092 *

--topic <TOPIC_NAME>

Open a new terminal and do *make produce*. Write some text inside the terminal then go to next slide.

3. Be a consumer

The usual command is this one :

kafka-console-consumer \

--bootstrap-server localhost:9092 \

--topic <TOPIC_NAME>

Or with to get all messages from beginning

kafka-console-consumer \

--bootstrap-server localhost:9092 \

--topic <TOPIC_NAME>

--from-beginning

Open two new terminals and do *make consume-from-latest* and *make consume-from-beginning*. To see the differences

More about...

Instead of using pre-configured make target, you can use ***make broker*** to connect to broker and copy / paste commands to have the same behavior.

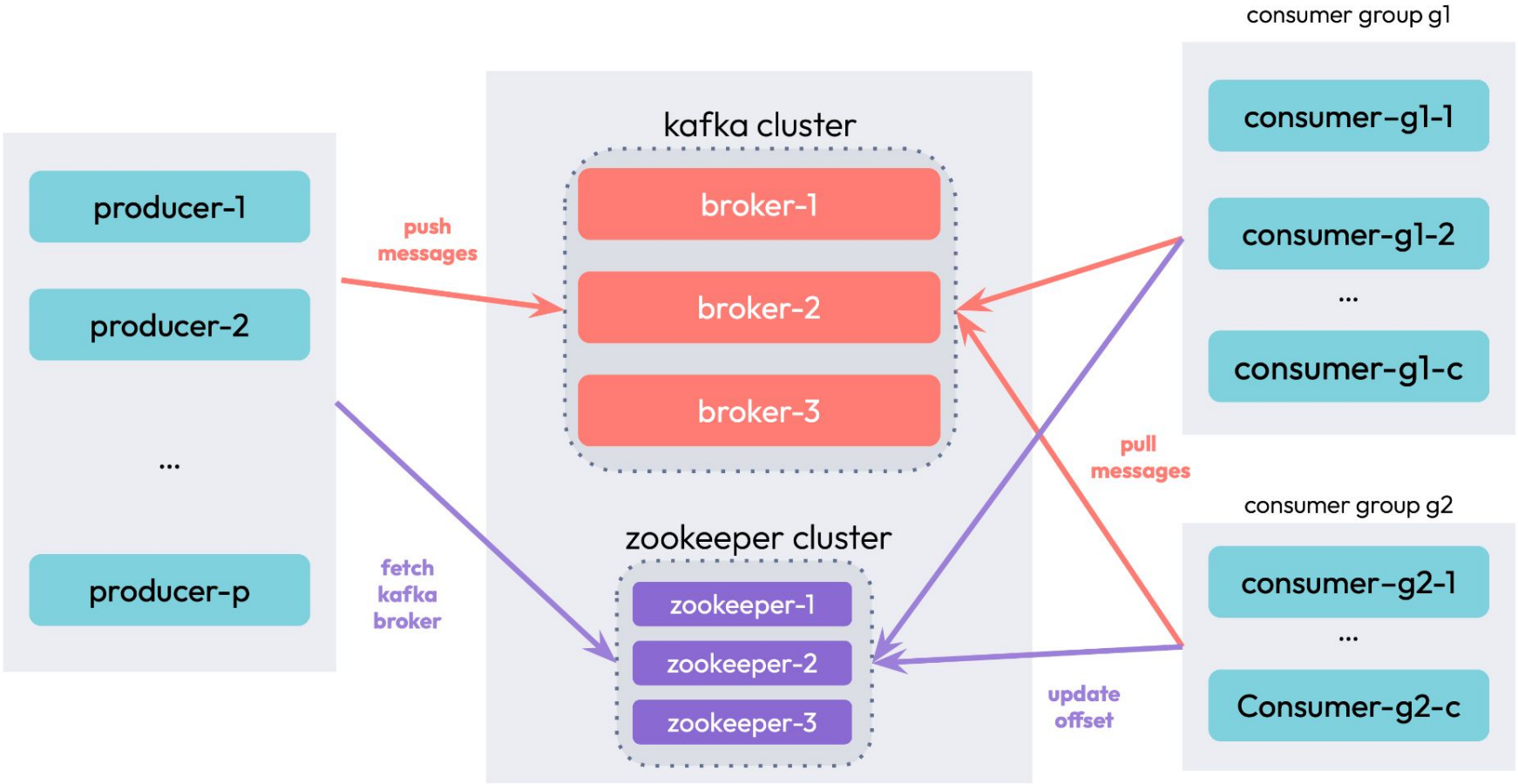
Some words about…

Topic retention

- By default, an event is available 7 days on the topic.
  - This value can be changed.
  - Design the right machine because more days implies more data storage
  - Specially with Confluent Cloud #FinOps

- It allows to have different speed processing (1x/day or 1x/5days per example)

Kafka architecture



04

Design patterns

Change data capture (CDC)

Émettre seulement les modifications apportées aux données

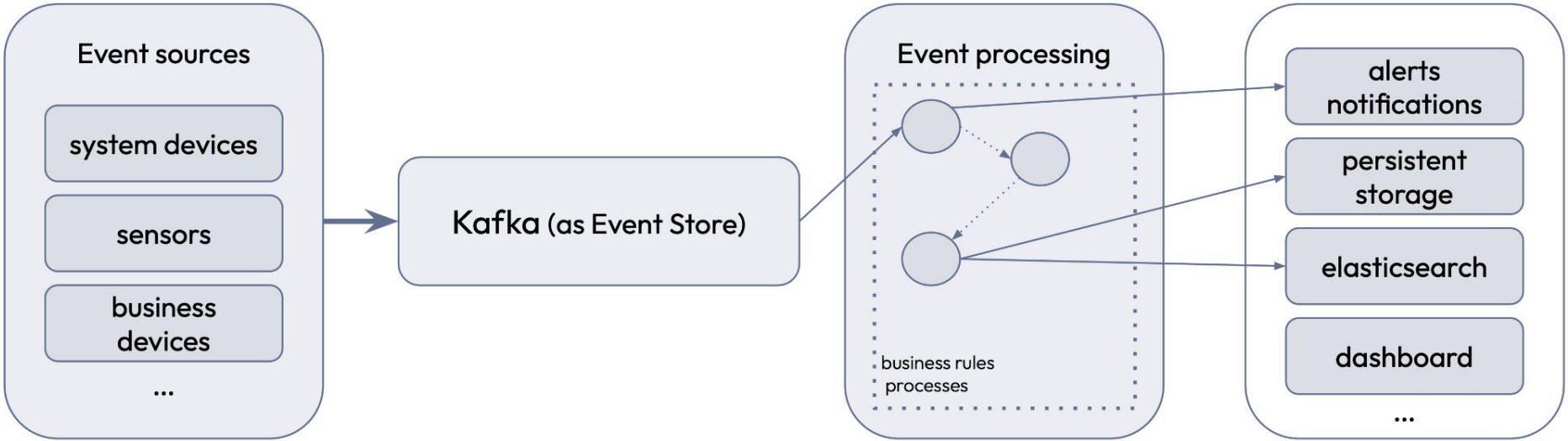
- You want to take actions based on changes into your database

CDC is an approach to data integration that is based on the identification, capture and delivery of the changes made to enterprise data sources.

It allows you to stream all the new changes from your data source (like a postgresql database) and operates calculus, aggregations, ... on those data.

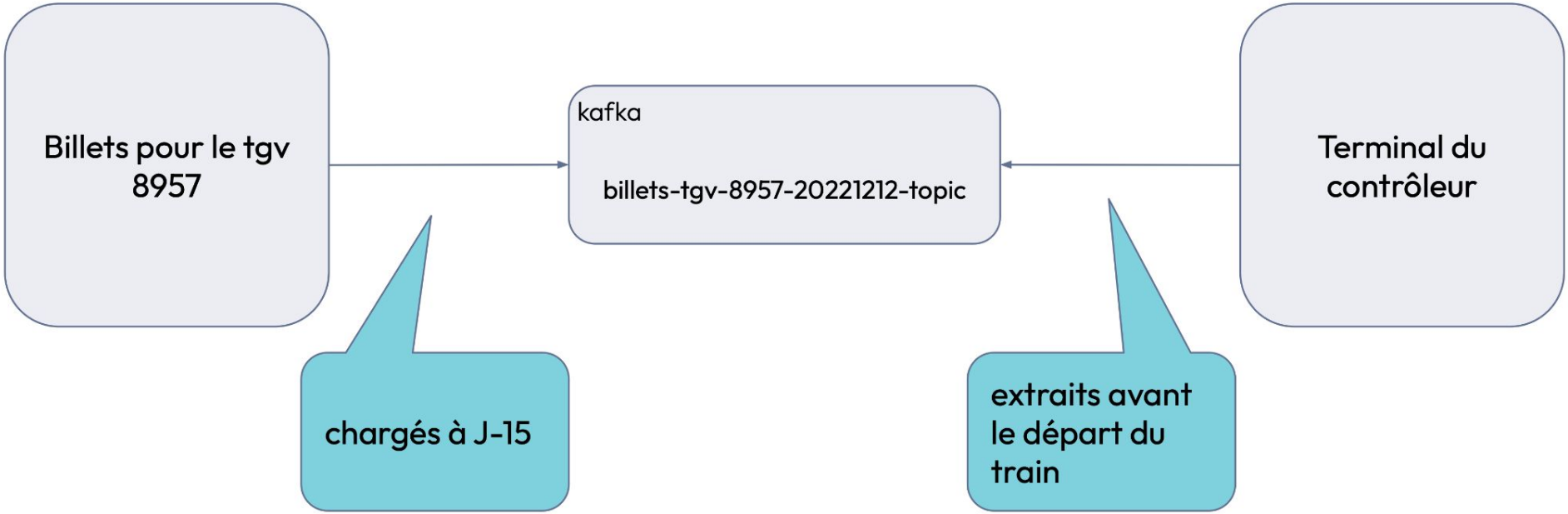
Complex Event Processing (CEP)

- You want to identify meaningful events (opportunities / threats / ...) in real-time situations and respond to them as quickly as possible.

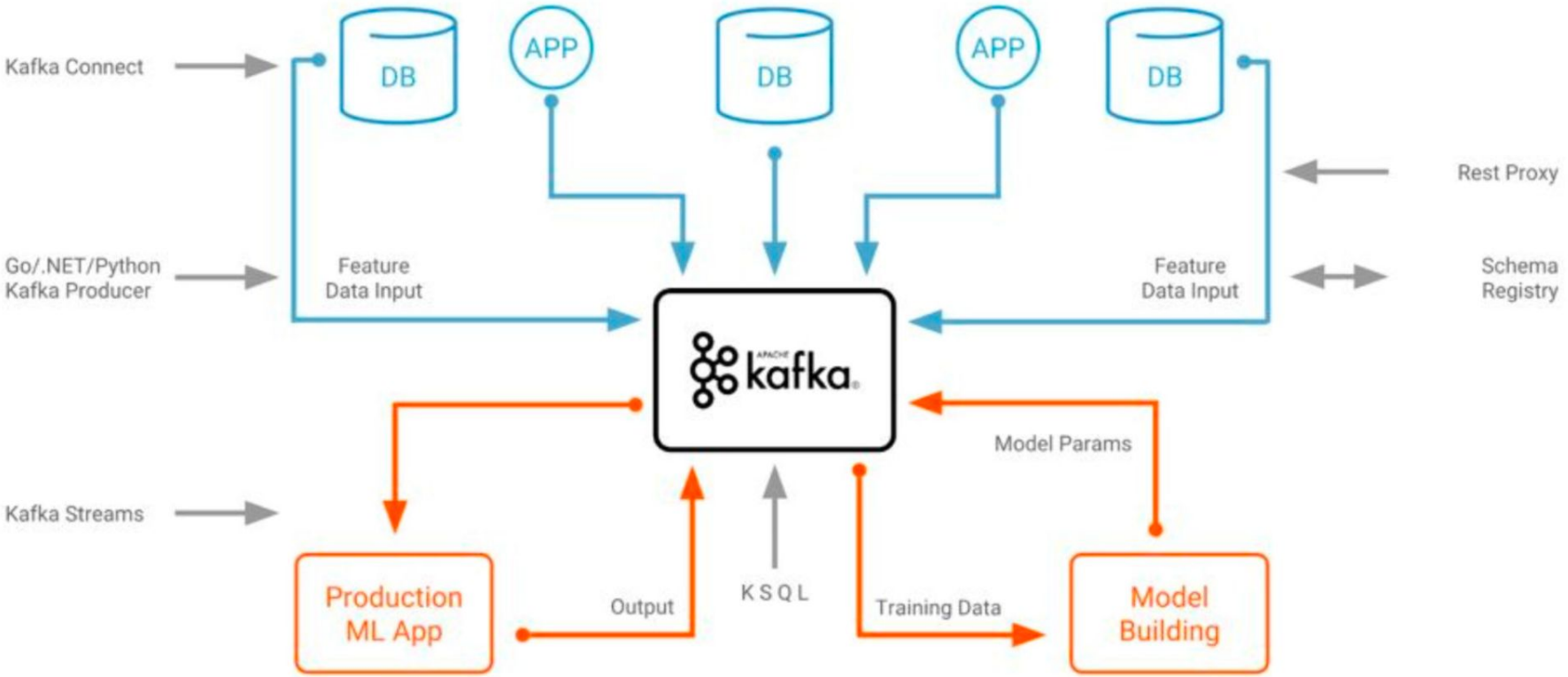


Event Store

- Stockage **temporaire** des billets du TGV n°8957 à destination de Nancy le 12 décembre 2022



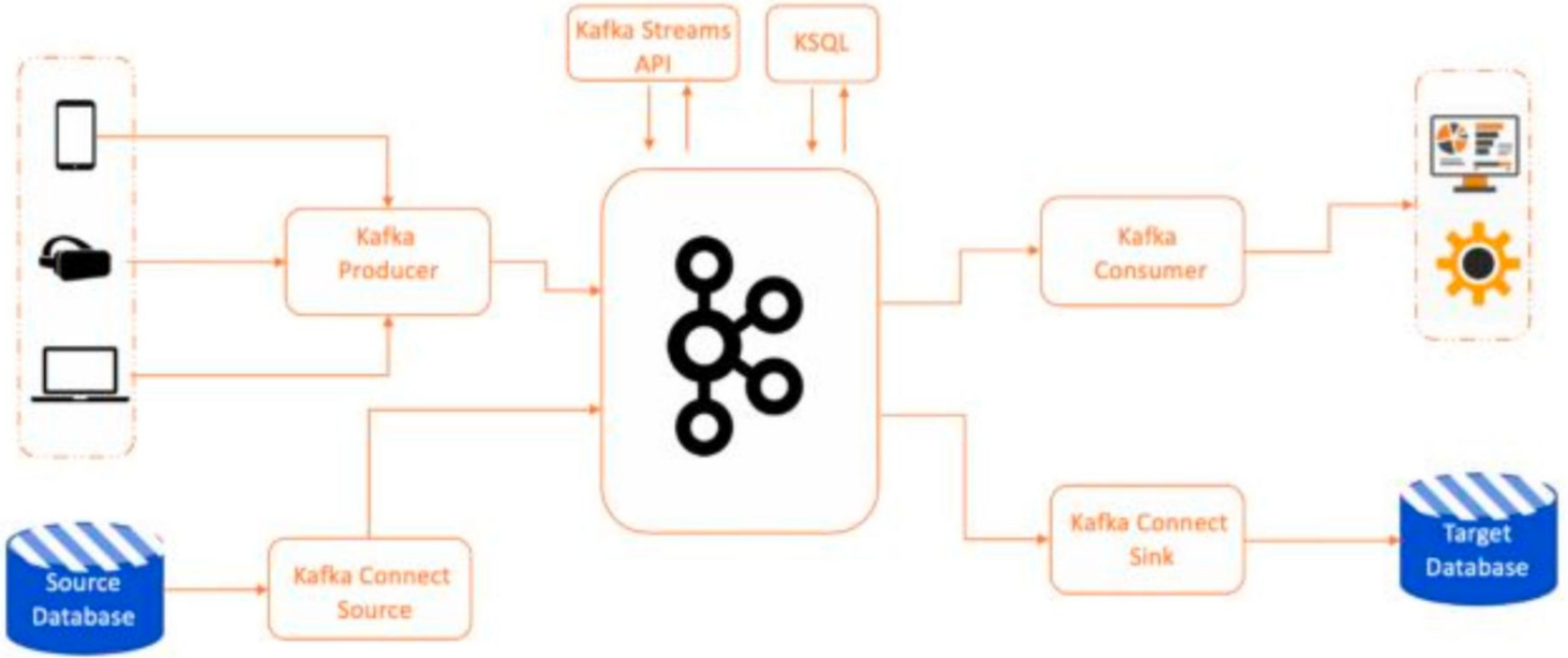
Online Prediction



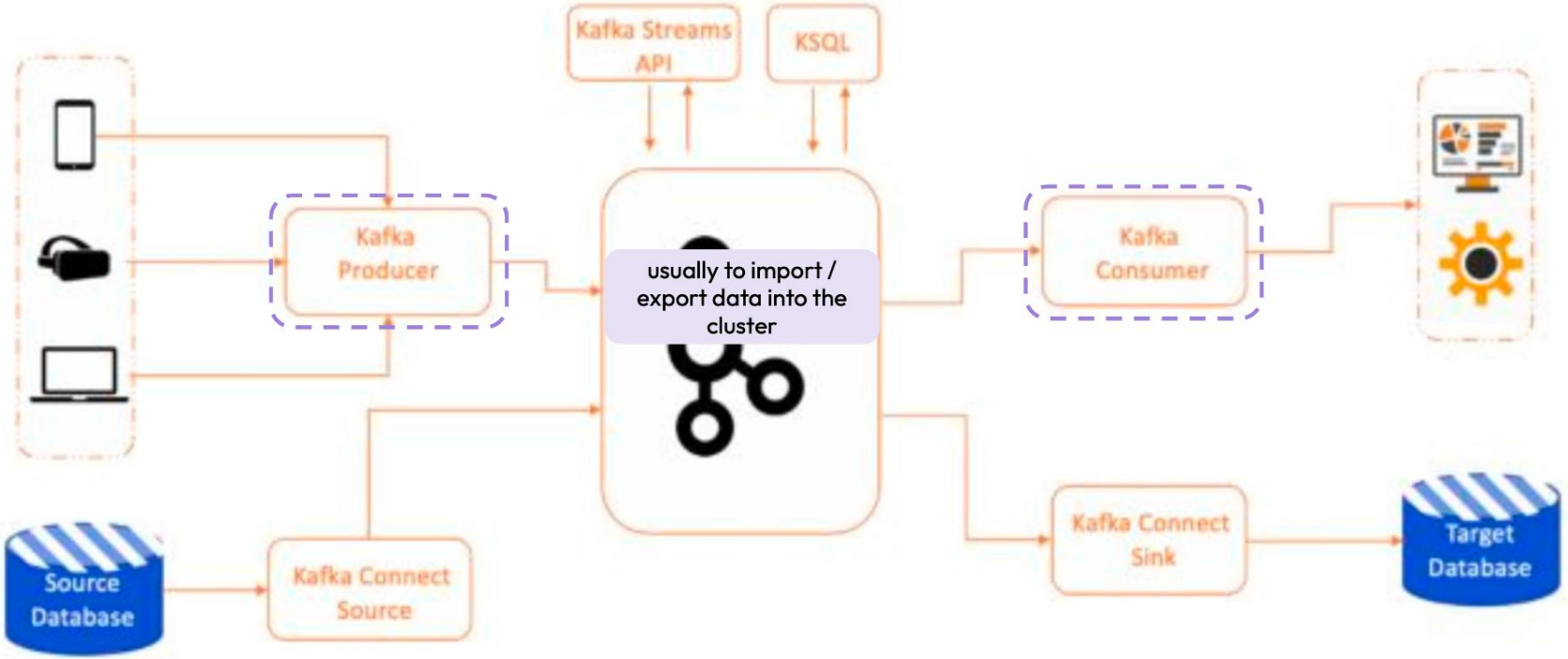
05

Kafka Ecosystem

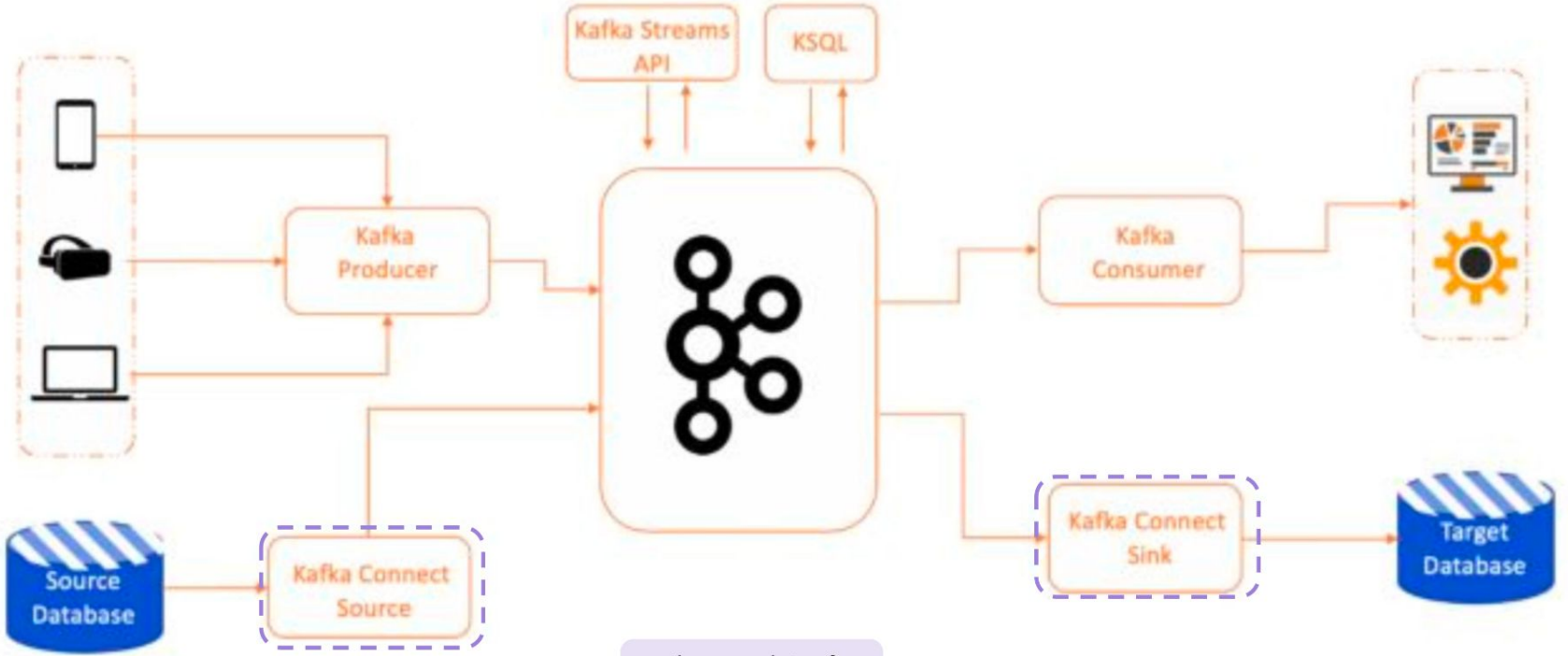
Kafka ecosystem



Kafka ecosystem



Kafka ecosystem



there are lots of connectors available

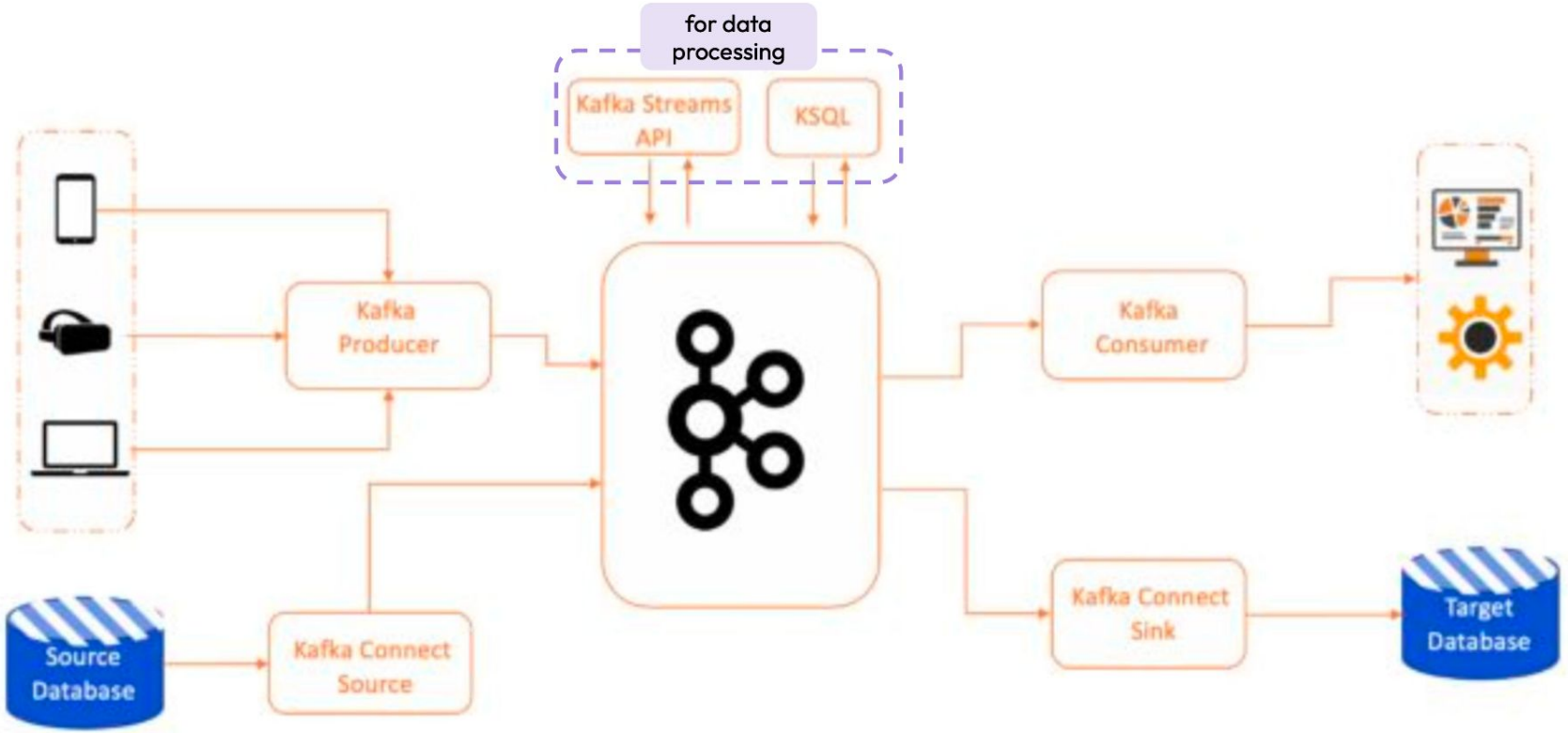
Kafka Connect

Interact easily with your existing system and the outside world

- Kafka Connect is a tool for scalability and reliably streaming data between Kafka and other systems.
  - It makes it simple to quickly define connectors that move large collections of data into and out of Kafka
  - It can ingest entire databases or collect metrics from application servers into Kafka topics, making the data available for stream processing with low latency

- There are connectors available for major data storage tool (like MySQL, ElasticSearch, aws / gcp, ...)
  - Theses connectors are maintained in part by the community
  - The other part is provided by Confluent, the major Kafka contributor

Kafka ecosystem



Kafka Stream / ksqlDB

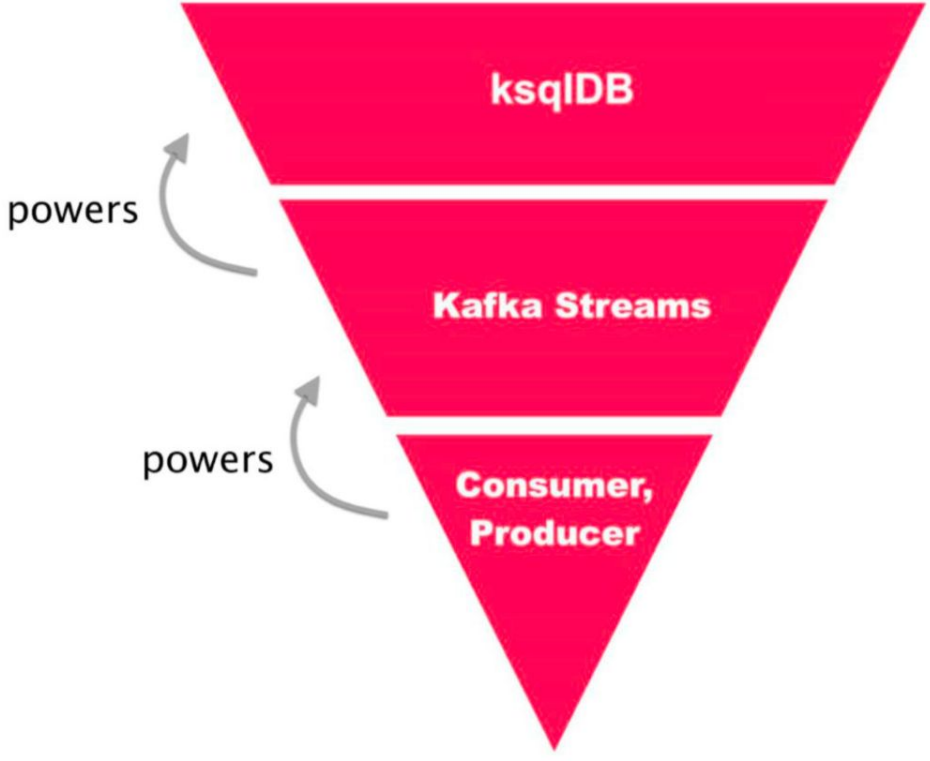
How to read, process and write your data

- There are two ways to perform transformations / computations on messages inside a kafka cluster:
  - ksqlDB, a SQL-like query language to easily operate on streams
  - Kafka Stream, a code library to write specific programs; usually when external systems are needed

- These tools allow you to
  - consume message from topics
  - perform any operations on data when a message is available on topics
    - joins and aggregations are possible too
    - windowing can be needed for joins
  - produce output back into kafka topic

Kafka Stream / ksqlDB

A meaningful picture



ease of use

```
CREATE STREAM, CREATE TABLE,
SELECT, JOIN, GROUP BY, SUM, …
```

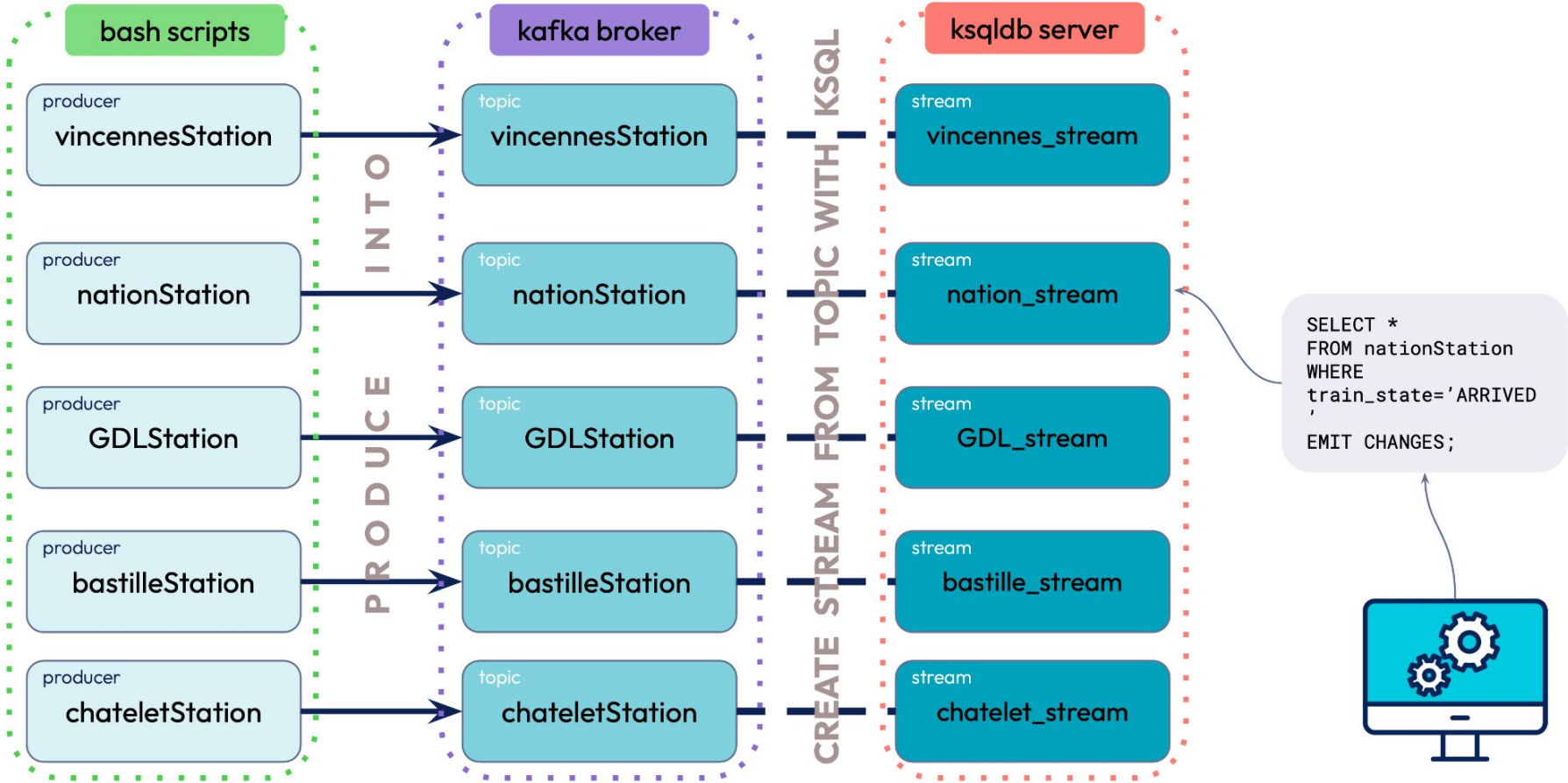
```
KStream, KTable,
filter(), map(), flatMap(),
join(), aggregate(), …
```

```
subscribe(), poll(), send(),
flush(), beginTransaction(), …
```

flexibility

DOJO

Focus on the lab architecture



Environment

Once you have launch the stack (to know when it's fully loaded, wait until the logs speed is slow), you can see what's happening on the environment.

Two interfaces :

- Kafka UI : localhost:8080
- Console to query kafka with ksqldb : localhost:7681
    - Once in this interface, write ***ksql http://ksqldb-server:8088*** *on the console and you'll be ready to write some requests*

THE END

See you next week ;-)