Data Engineering

LTAT.02.007

Ass Prof. Riccardo Tommasini

Assistants: Fabiano Spiga, Mohamed Ragab, Hassan Eldeeb

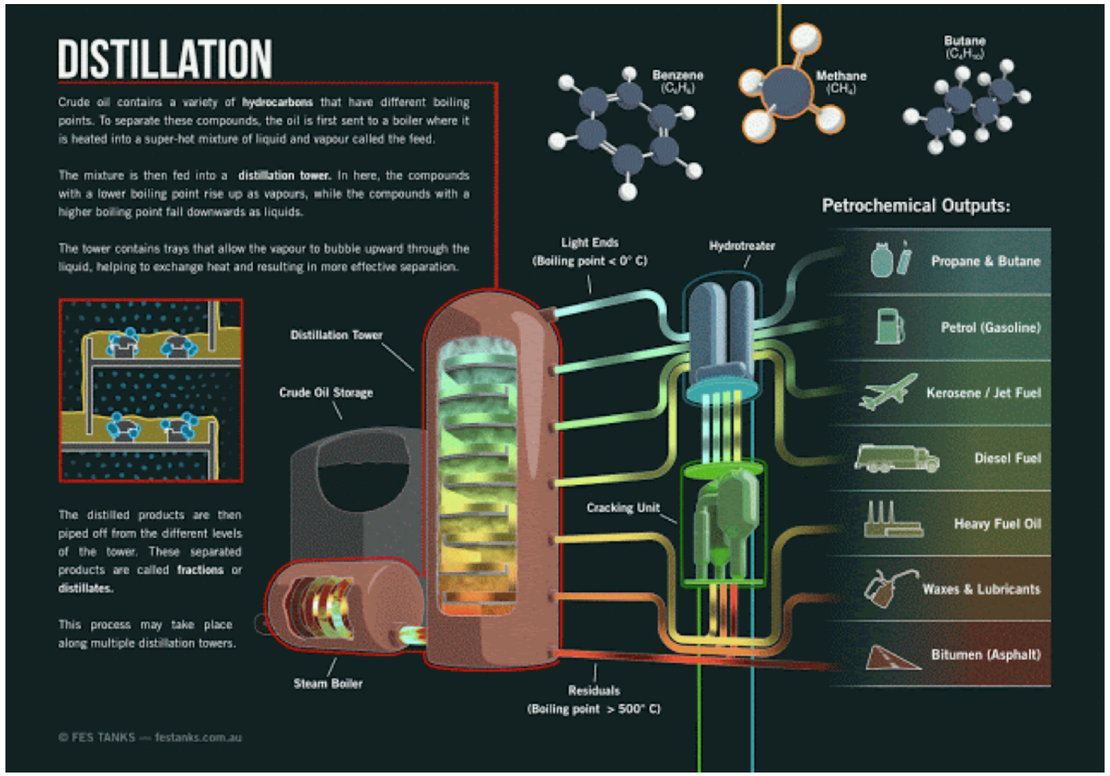
https://courses.cs.ut.ee/2020/dataeng

Forum

Moodle

Data Science vs Data Engineering Pipelines

Data Science is...



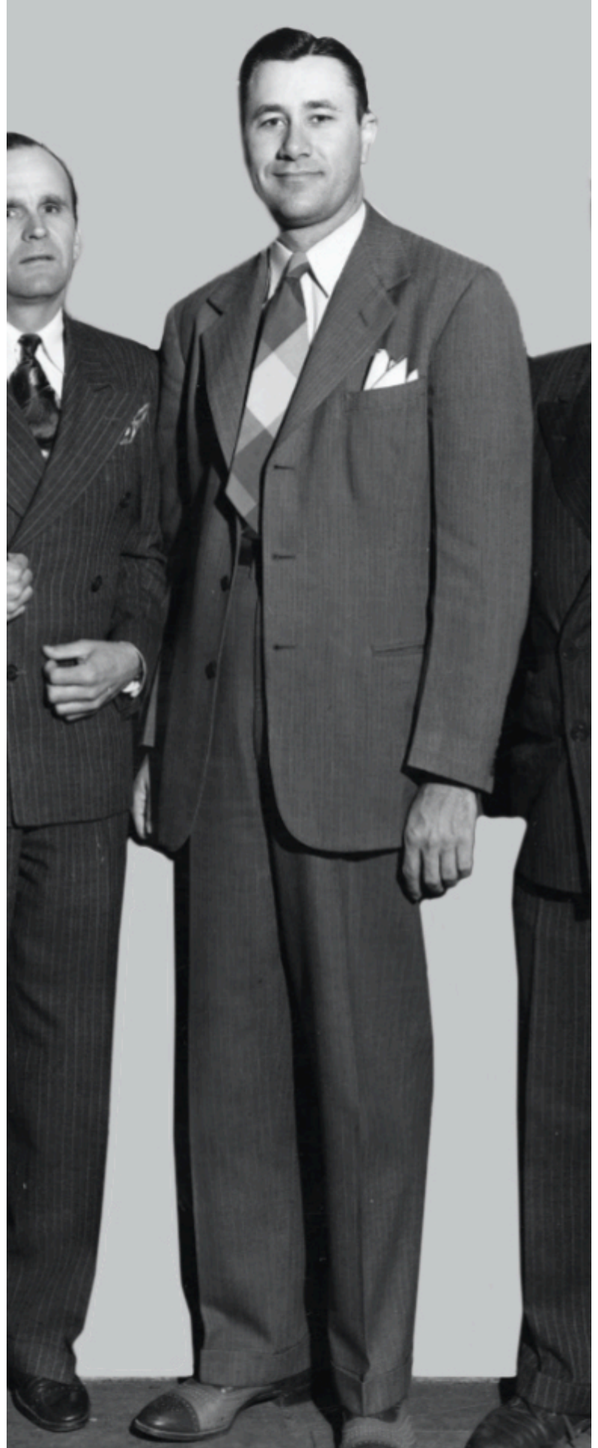
...refining crude oil

Data Engineering is...

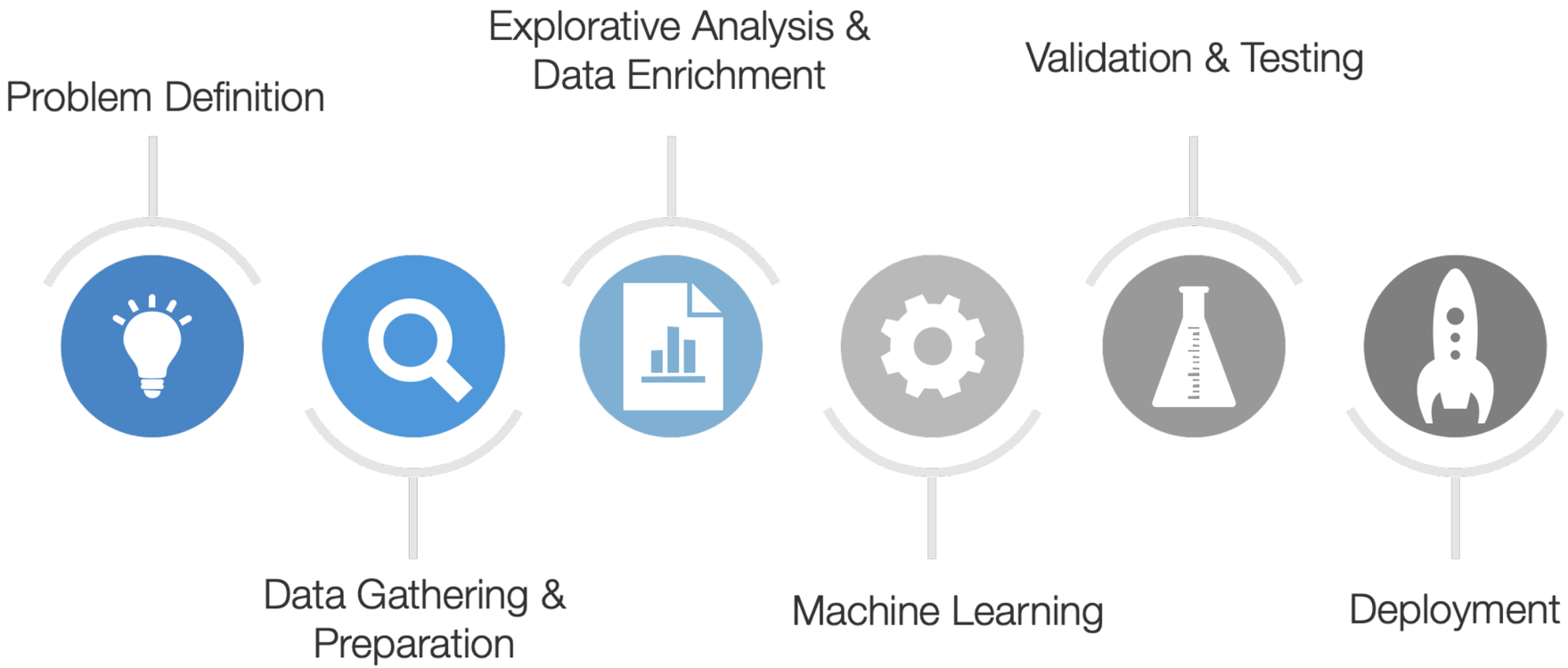
...build the refinery.

Quote

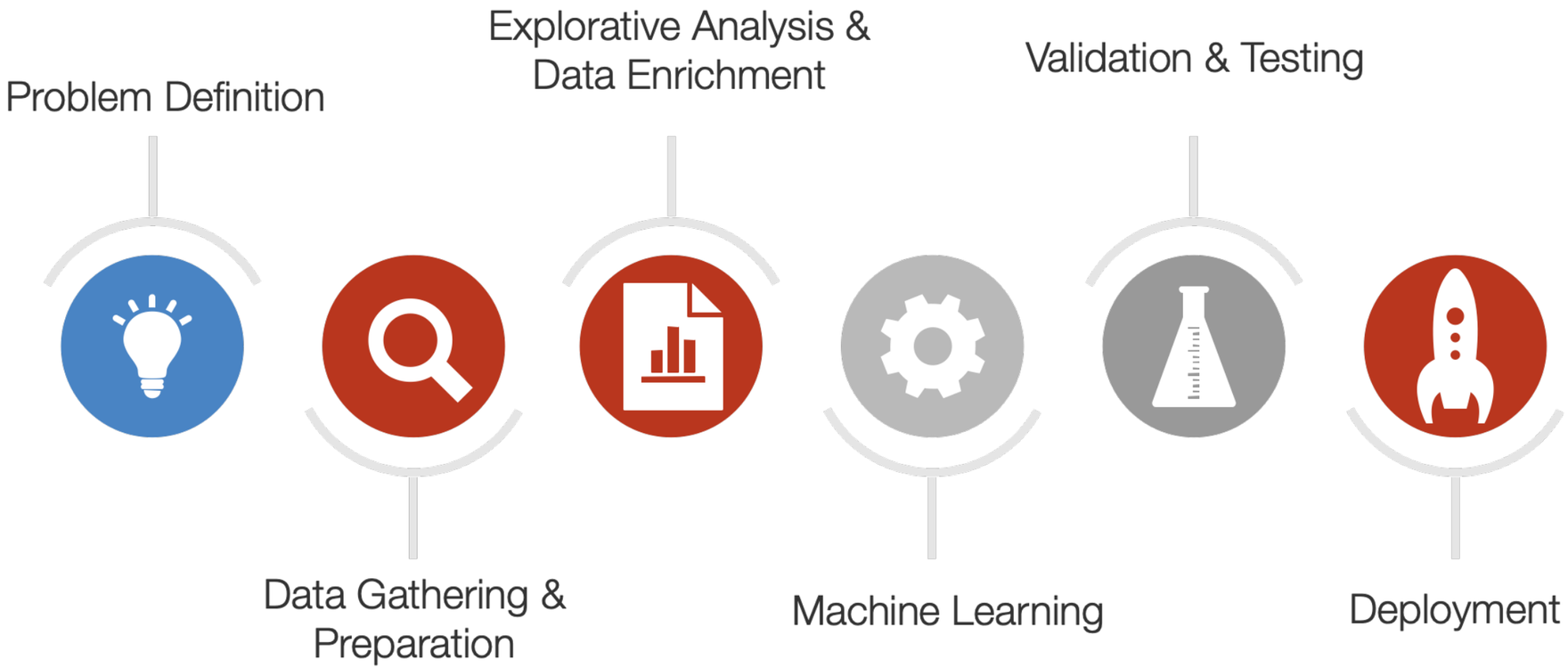
“A scientist can discover a new star,
but he cannot make one.
He would have to ask an engineer
to do it for him.”
– Gordon Lindsay Glegg



Data Analysis Pipeline



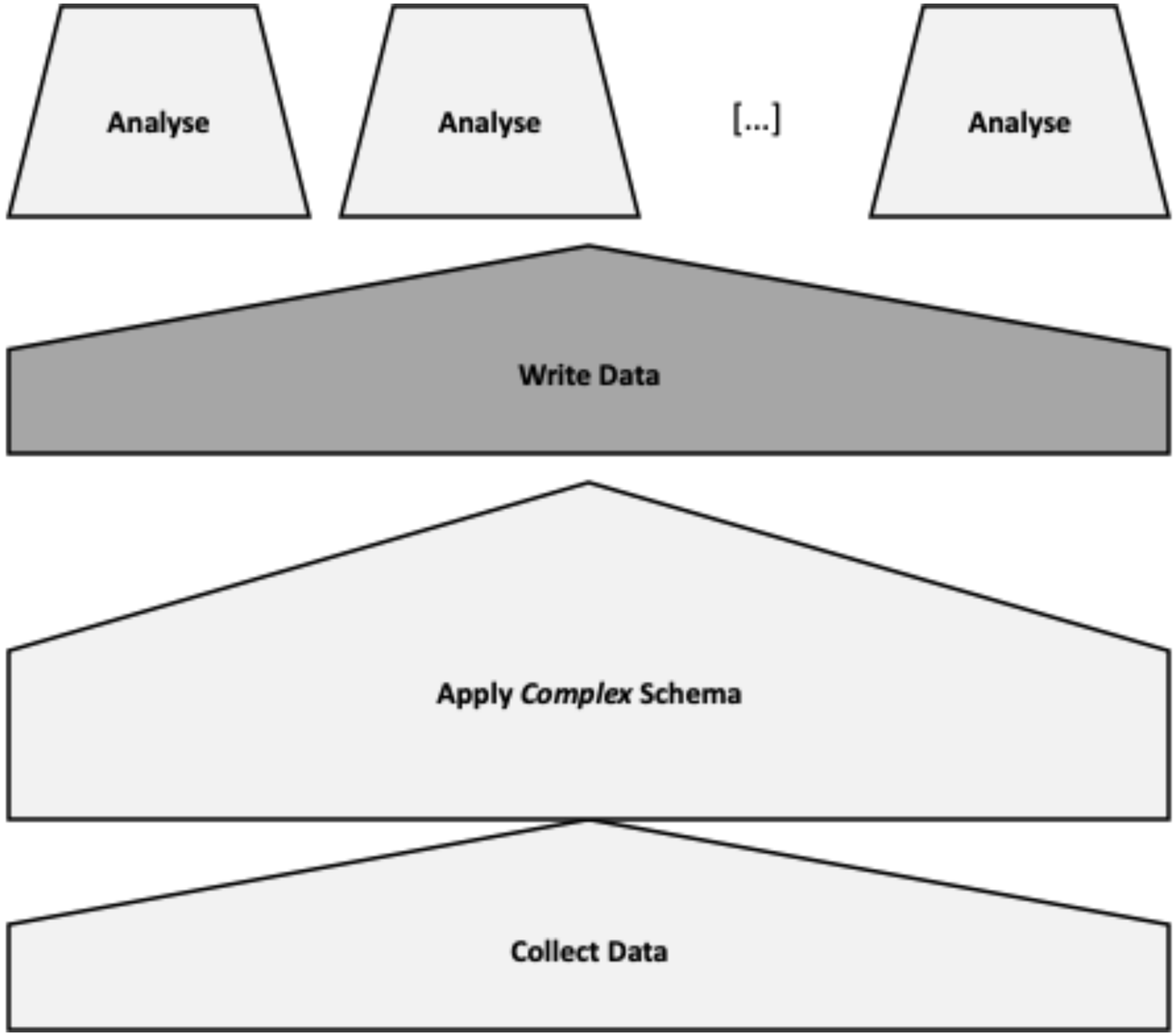
Data Engineering Parts



Recall

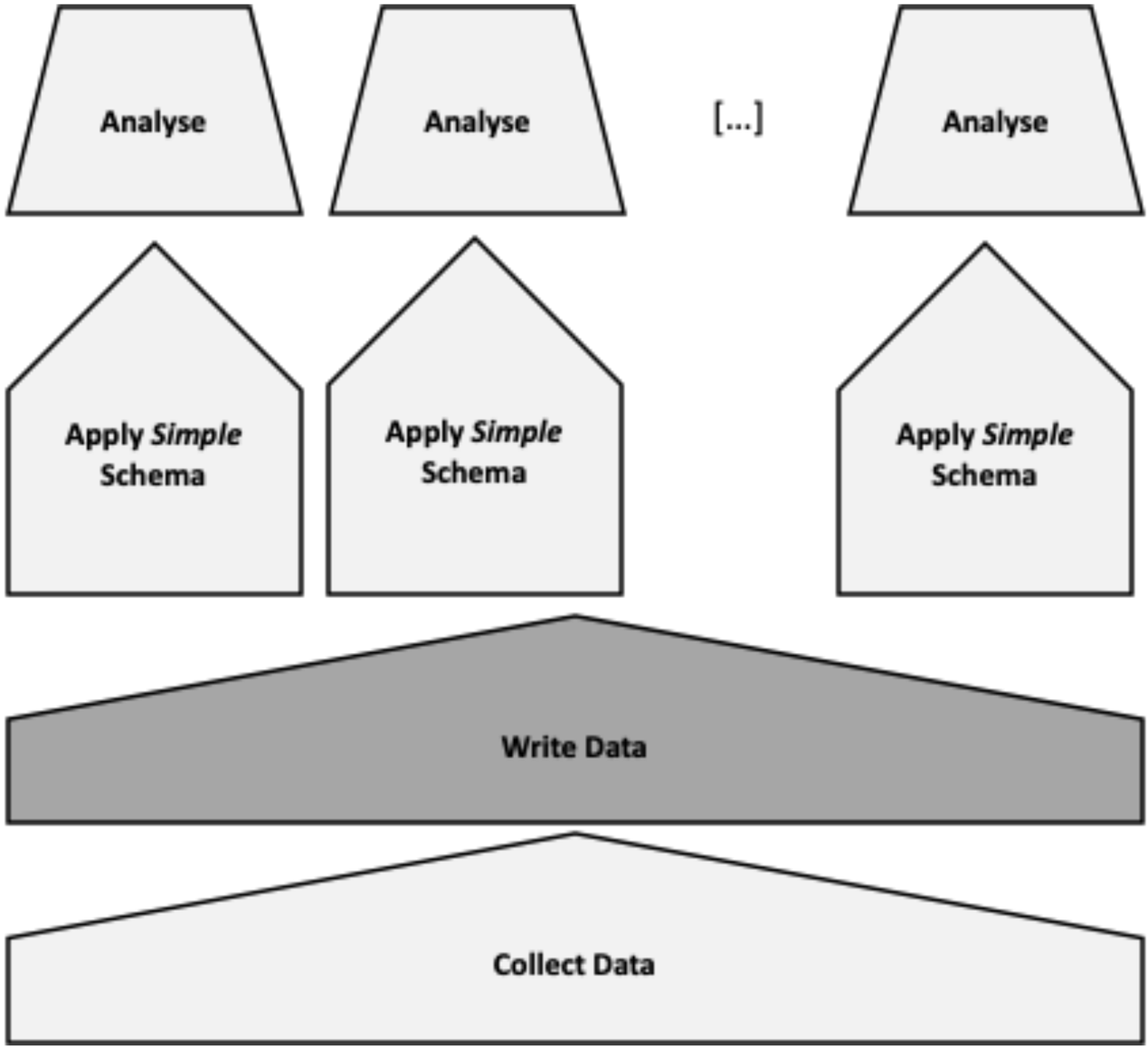
Traditional Data Modelling Workflow

- Known as Schema on Write
- Focus on the modelling a schema that can accommodate all needs
- Bad impact on those analysis that were not envisioned

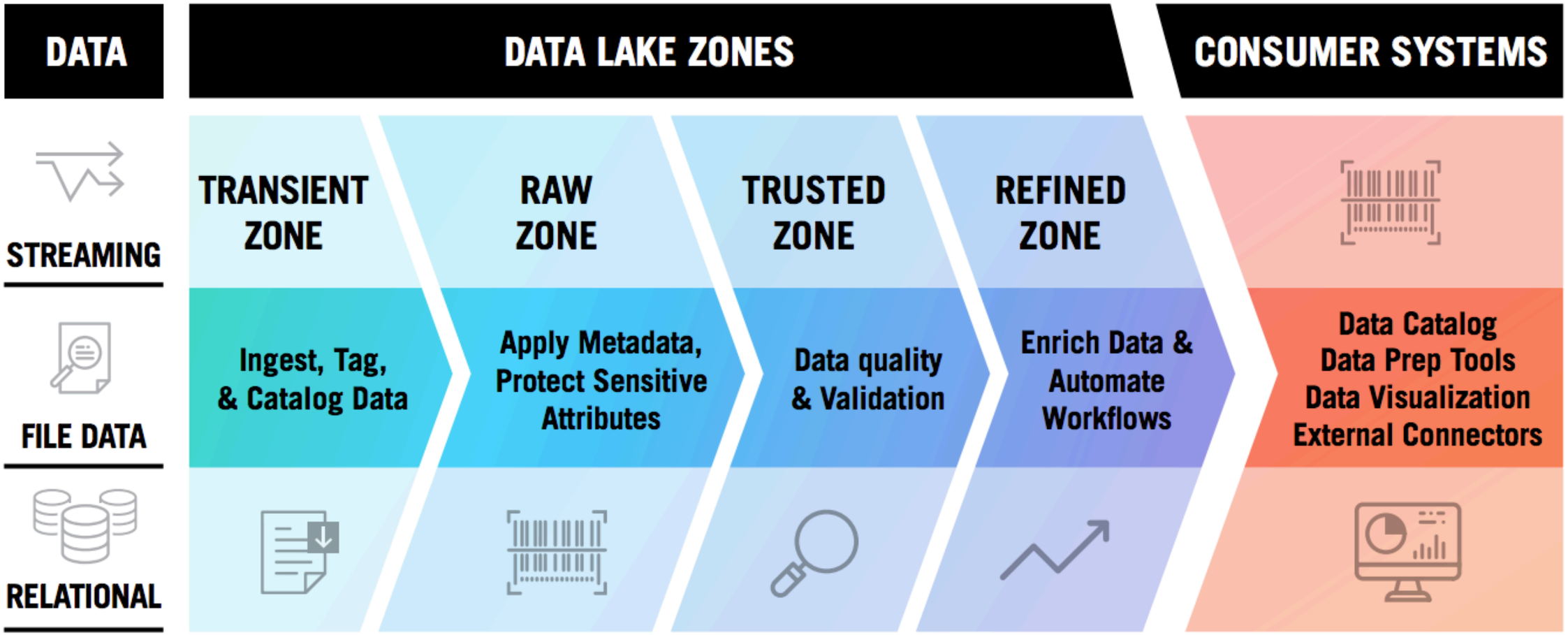


Schema on Read

- Load data first, ask question later
- All data are kept, the minimal schema need for an analysis is applied when needed
- New analyses can be introduced in any point in time

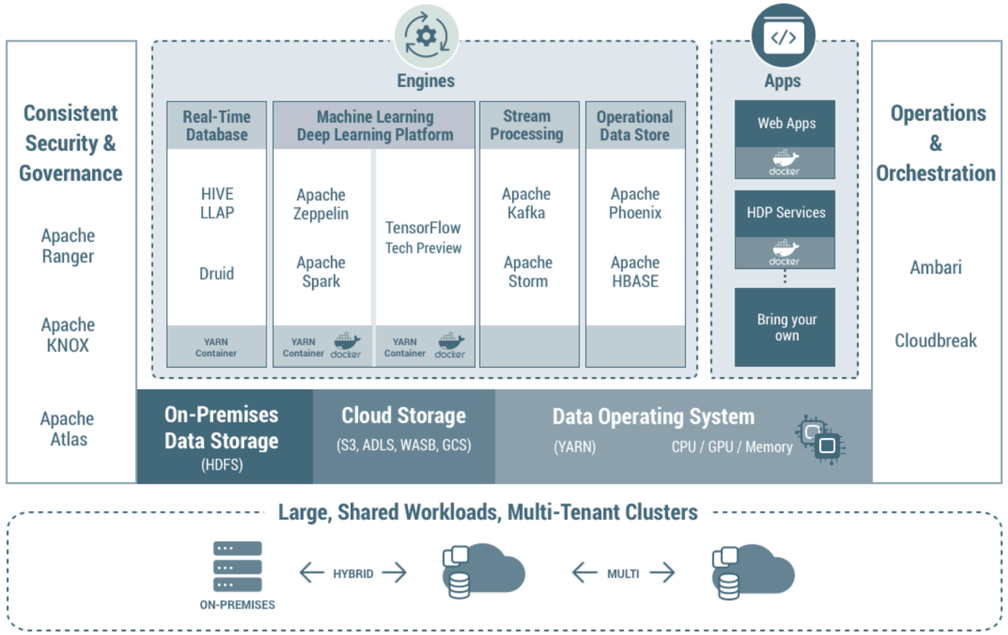


Data Lakes (Conceptual View)



Data Lakes (Physical View)

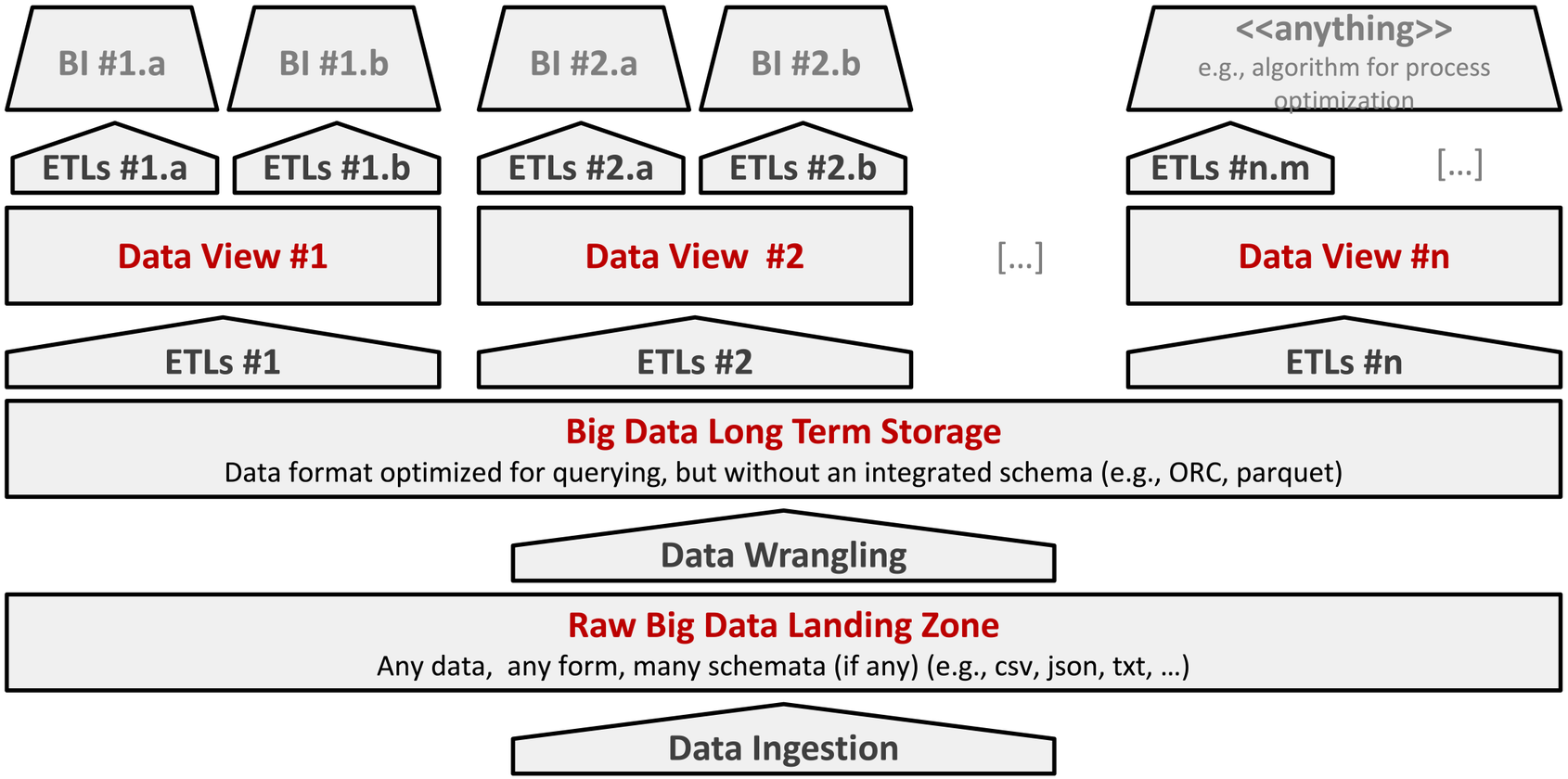
Physical Architecture of a Big Data Platform



8

So, what’s a logical architecture for a data engineering pipeline?

Logical architecture of a data engineering pipeline



Data Acquisition

- It is the process of collecting raw data from various silo databases or files and integrating it into a data lake on the data processing platform, e.g., Hadoop data lake.
- It involves loading data from a variety of sources
- It can involve altering and modification of individual files to fit into a format that optimizes the storage
- For instance, in Big Data small files are concatenated to form files of 100s of MBs and large files are broken down in files of 100s of MB

Data Acquisition[120]

[120] Synonsymns: **Data Ingestion]] [[synonimns/Data Collection]]llection**

Let's fix the naming (arbitrary)

- We call **Data Collection** the process of finding and accessing new data sources
- We call **Data Ingestion** the process of filling our Data Lake with new data

Data Collection Examples

- Accessing Databases ✅
- Crowling the Web
- Calling APIs
- Consuming WebSockets

Data Ingestion Examples

- Maintaining a Distributed File Systems
- Using a Distributed Message Queue
- Using a Publishing Subscribe System

Two Schools

- Batch
- Pull
- Query-Based

- Streaming
- Push
- Dataflow based

REST API | REST API and Beyond

The standard way to interact with Web data sources in a programmatic manner.

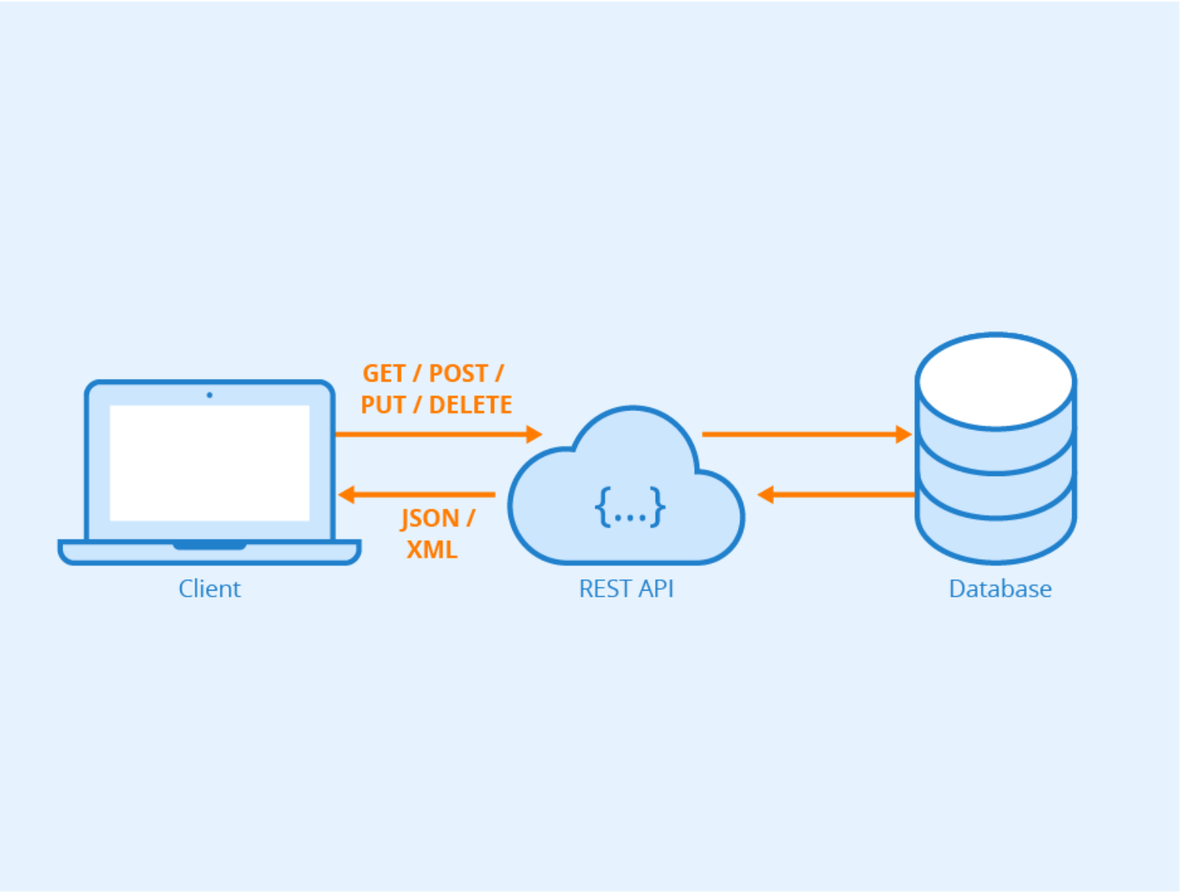
HDFS | Distributed File Systems

A distributed file system stores files across a large collection of machines while giving a single-file-system view to clients.

Apache Kafka | Distributed Message Queues

A distribured message quque stores file in a log an allos sequential reads.

REST API



World Wide Web

International ecosystem of applications and services that allows us to search, aggregate, combine, transform, replicate, cache, and archive the information that underpins society.

The Web is the result of millions of simple, small-scale interactions between agents and resources that use the founding technologies of HTTP and URI. [121]

The Web is a set of widely commoditised servers, proxies, caches, and content delivery networks [an engineers]

---

[121] Architecture of the World Wide Web, Volume One

Resources

Resources are the fundamental building blocks

Anything we can expose, i.e., documents, images, videos, audio, devices, people, things…

We can represent them by abstracting the useful information and identifying using a Uniform Resource Identifier (URI)

URIs

URL format (RFC 2396):
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

E.G.:
git@github.com:nodejs/node.git
mongodb://root:pass@localhost:27017/TestDB?options?replicaSet=test
http://example.com

HTTP

- GET
  - Uniform Interface
  - read-only operation
  - idempotent
- POST
  - like a resource upload
  - idempotent
- DELETE
  - remove resources
  - idempotent
- HEAD
  - HEAD is like GET except it returns only a response code
- PUT
  the only non-idempotent and unsafe operation
  is allowed to modify the service in a unique way
- OPTIONS is used to request information about the communication options of the resource

Representation

Access to a resource is mediated by a representation

This separation is convenient to promote loose-coupling between server (producers) and client (consumers)

Multiple Views and Content Negotiation are the basis for interoperability

Web API

An API that uses the HTTP protocol. Non standard/unpredictable URL format.

E.g., `POST http://example.com/getUserData.do?user=1234`

Representational state transfer (REST)

- A set of constraints that inform an architecture
  - Resource Identification
  - Uniform Interface
  - Self-Describing Messages
  - Stateless Interactions
- Claims: scalability, mashup-ability, usability, accessibility

RESTful API

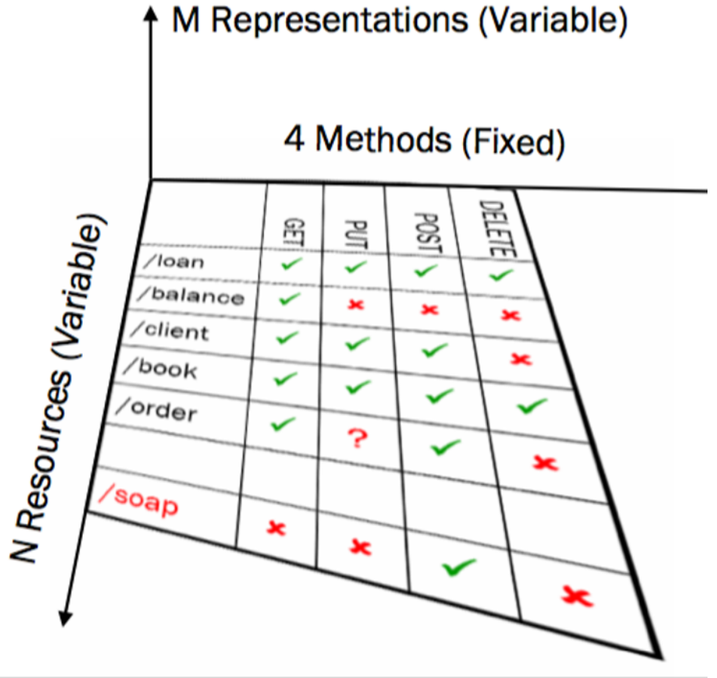
Resource based WebAPI. Standard/predictable URL format.

E.g., `GET http://example.com/user/1234`

VERB	**Collection**	**Item**
POST	Create a new item.	Not used
GET	Get list of elements.	Get the selected item.
PUT	Not used	Update the selected item.
DELETE	Not used	Delete the selected item.

Design Space

The RESTful API uses the available HTTP verbs to perform CRUD operations based on the “context”:
Collection: A set of items (e.g.: /users)
Item: A specific item in a collection (e.g.: /users/{id})



Real-Time Web API

Websocket

The Problem with HTTP

- The communication is only in one direction at any point. The request comes from the client; the API processes it, responds and that's the end of the communication.
- The client continuously polls the API to get the most recent data. Not only it wastes bandwidth, but it also adds latency to the process

We want to let the server push some data to the client over http (or https) without the client explicitly requesting it.

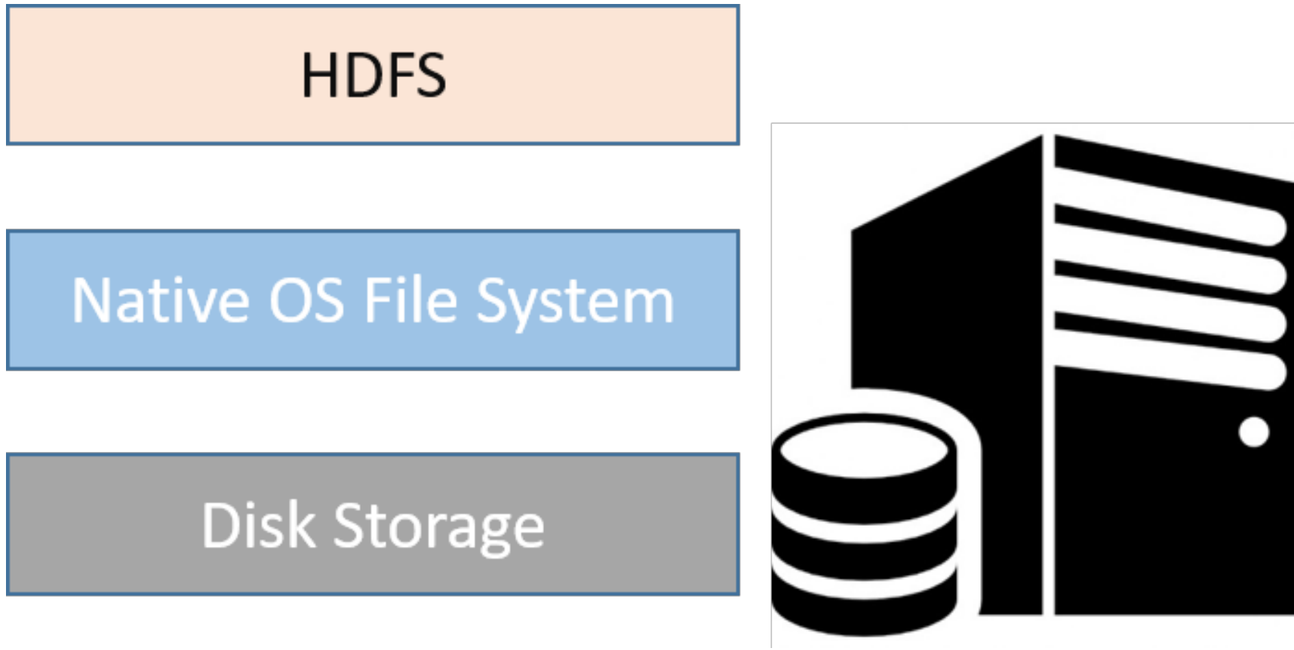
Options

- HTTP long poll: keep the connection between the client and API open for a long time, allowing the client to get the update in realtime.
- Need of a standardized process to enable full duplex data transfer over the http protocol
  - WebSockets is a protocol, allowing full duplex data transfer.
  - Server-Sent Events

HTTP allows you to upgrade the requested connection to something else. There are only a handful of upgrades allowed in HTTP 1.1, which are h2c, HTTPS/1.3, IRC/6.9, RTA/x11, websocket. Upgrades allow the request to come as a normal http one but then upgraded to something else in those list of protocols.

Hadoop Distributed File System (HDFS)[12]

- Abstracts physical location (Which node in the cluster) from the application
- Partition at ingestion time
- Replicate for high-availability and fault tolerance



[12] Inspired by Google File System

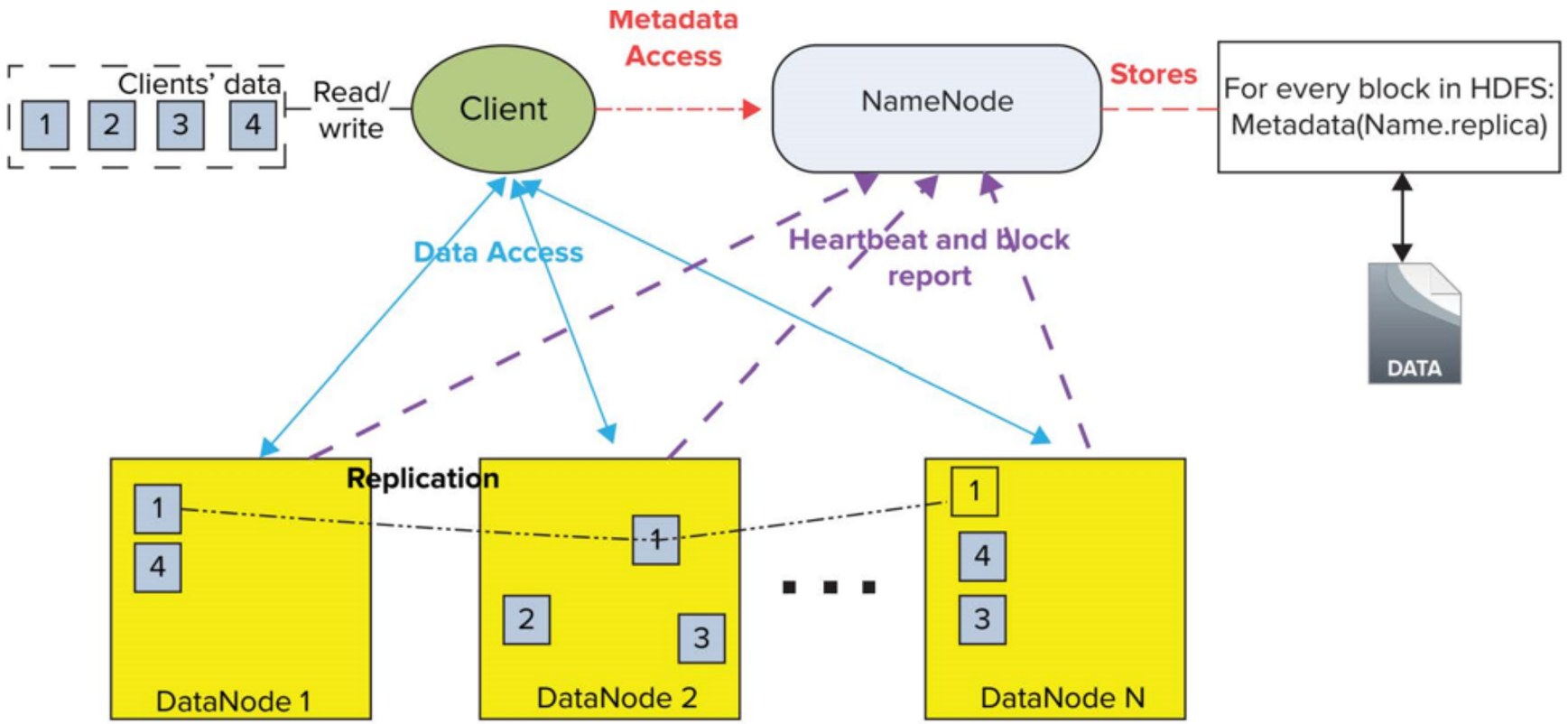
Design Objectives

- Partition and distribute a single file across different machines
- Favor larger partition sizes
- Data replication
- Local processing (as much as possible)

Optimizations

- Reading sequentially versus (random access and writing)
- No updates on files
- No local caching

HDFS Architecture[13]



[13] Figure 2-1 in book Professional Hadoop Solutions

HDFS Files

- A single large file is partitioned into several blocks
  - Size of either 64 MB or 128MB
  - Compare that to block sizes on ordinary file systems
  - This is why sequential access is much better as the disk will make less numbers of seeks

Data Node

- It stores the received blocks in a local file system;
- It forwards that portion of data to the next DataNode in the list.
- The operation is repeated by the next receiving DataNode until the last node in the replica set receives data.

Name Node

- A single node that keeps the metadata of HDFS
  - Keeps the metedata in memory for fast access
  - Periodically flushes to the disk (FsImage file) for durability
  - Name node maintains a daemon process to handle the requests and to receive heartbeats from other data nodes

Writing to HDSF (NameNode)

- When a client is writing data to an HDFS file, this data is first written to a local file.
- When the local file accumulates a full block of data, the client consults the NameNode to get a list of DataNodes that are assigned to host replicas of that block.
- The client then writes the data block from its local storage to the first DataNode in 4K portions.

Writing a File to HDSF (DataNode).

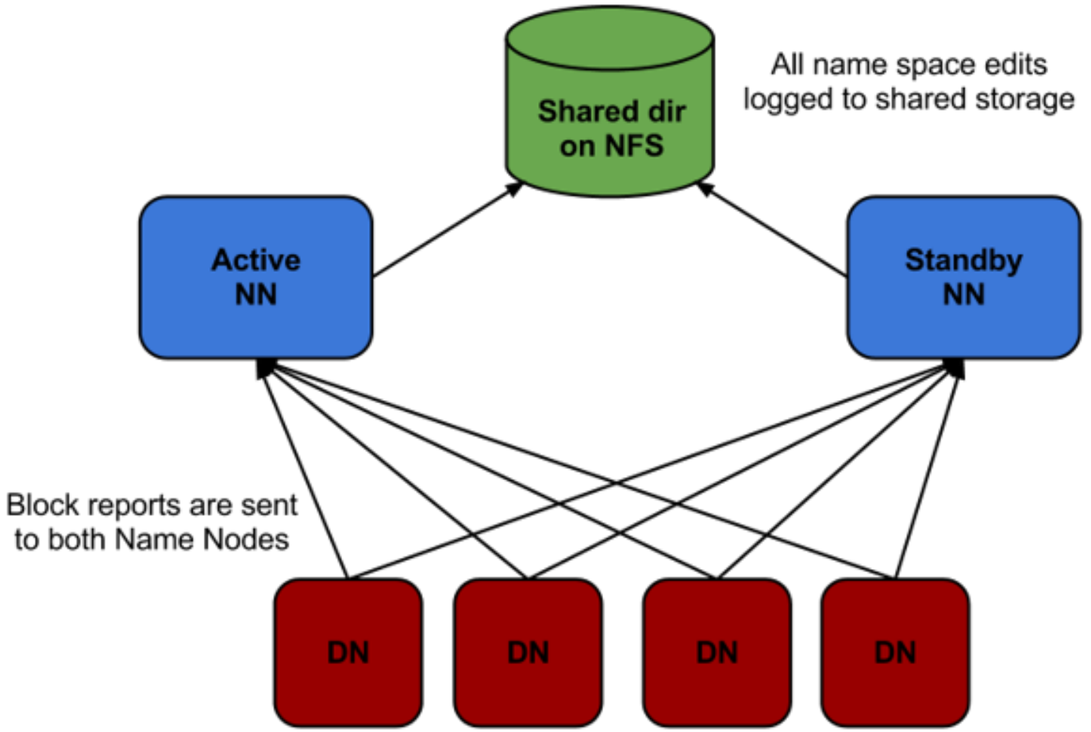
- This DataNode stores data locally without sending it any further
- If one of the DataNodes fails while the block is being written, it is removed from the pipeline and NameNode re-replicates
- When a file is closed, the remaining data in the temporary local file is pipelined to the DataNodes
- If the NameNode dies before the file is closed, the file is lost.

Replica Placement

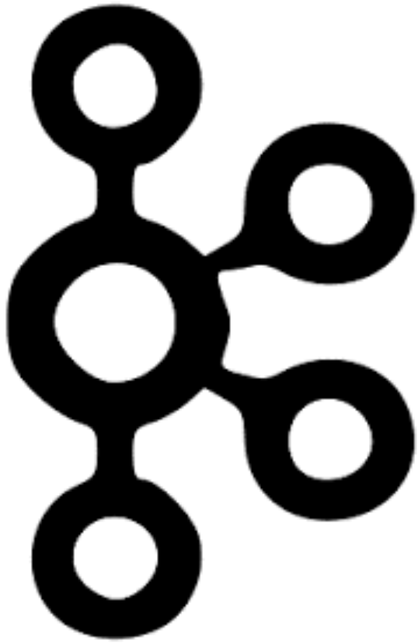
- Replica placement is crucial for reliability of HDFS
  - Should not place the replicas on the same rack
- All decisions about placement of partitions/replicas are made by the NameNode
- NameNode tracks the availability of Data Nodes by means of Heartbeats

HDFS High-availability

- Each NameNode is backedup with a slave other NameNode that keeps a copy of the catalog
- The slave node provides a failover replacement of the primary NameNode
- Both nodes must have access to a shared storage area
- Data nodes have to send heartbeats and block reports to both the master and slave NameNodes.



Apache Kafka[1]



An overview

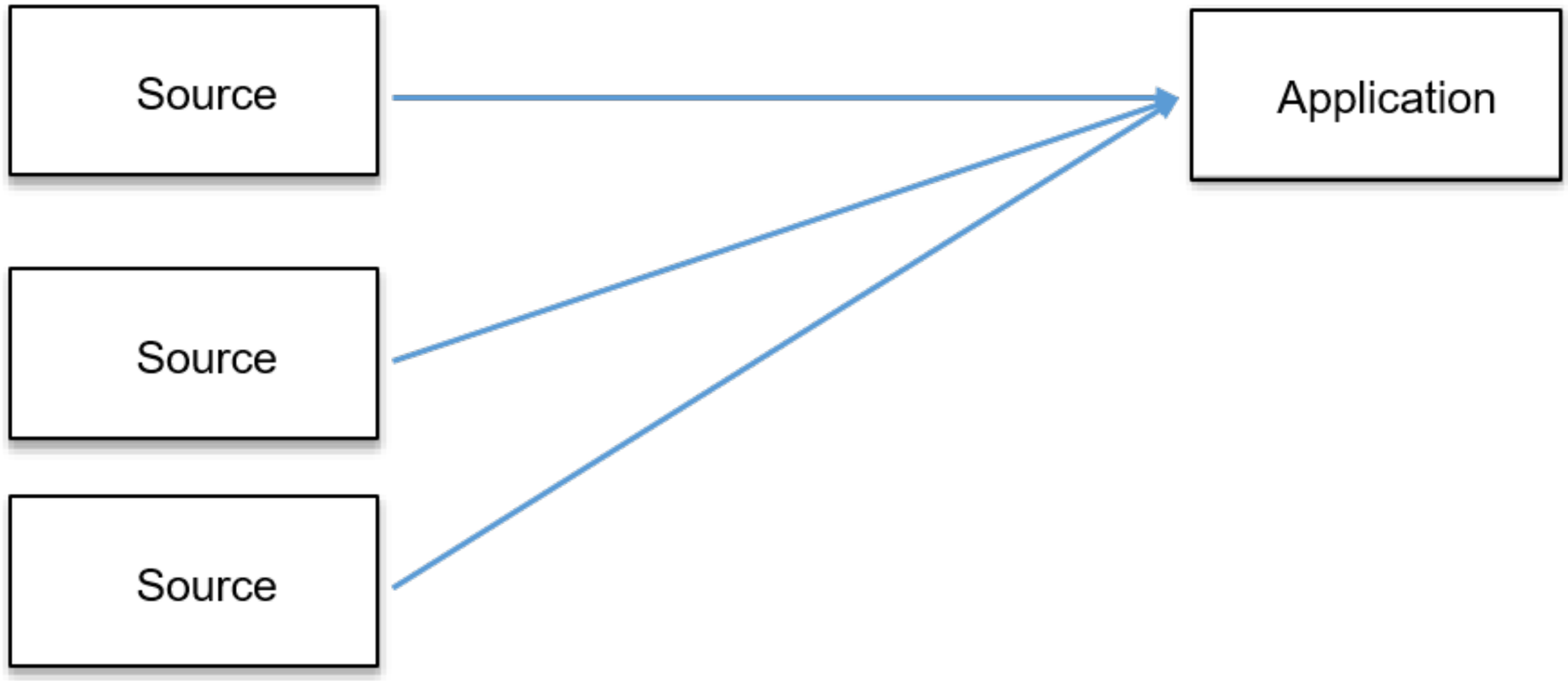
[1] https://kafka.apache.org/

Motivation

Data pipelines start with a small number of systems to integrates. A single ETL (extract, transform, load) process move data from the source to the interested applications.

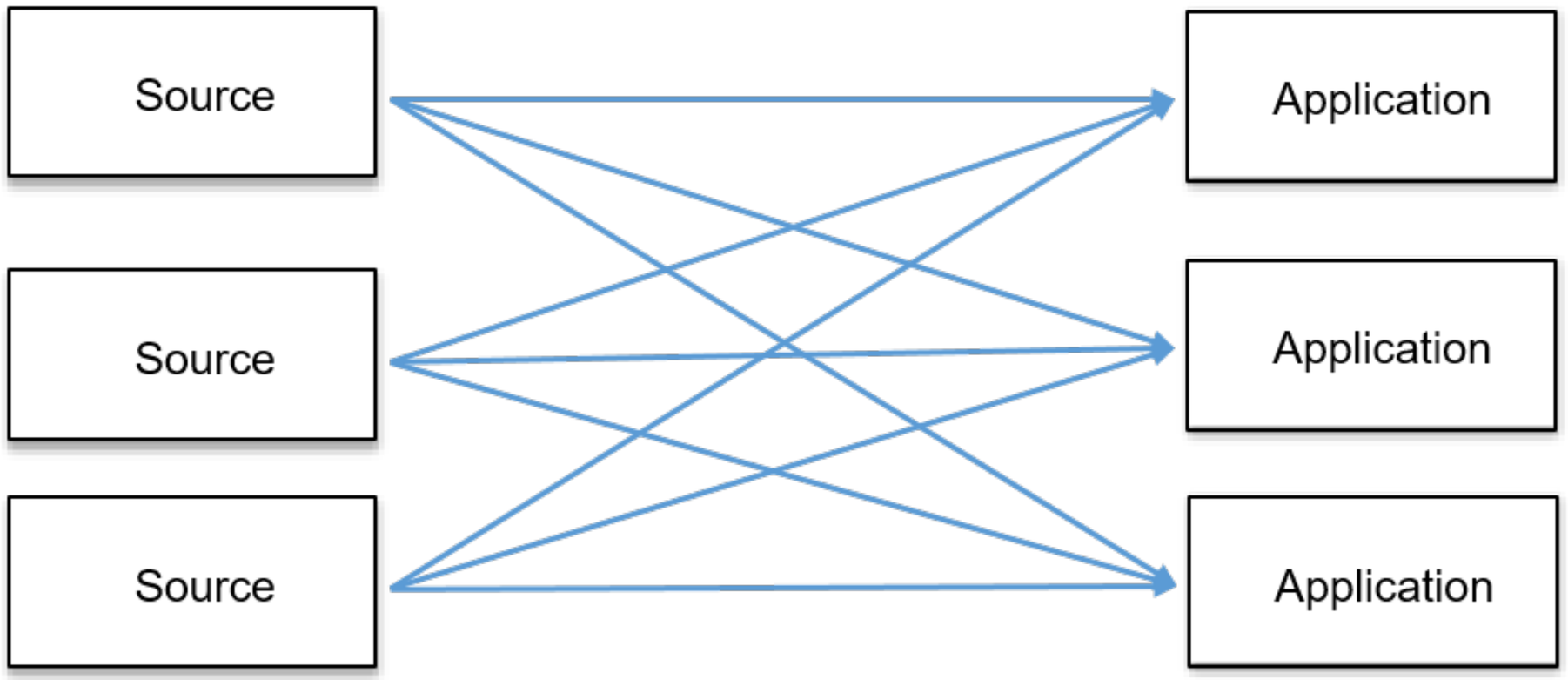
Motivation

But data pipeline grow over time. Adding new system causes the need of new ETL process. The code-base grows together with data formats and services.



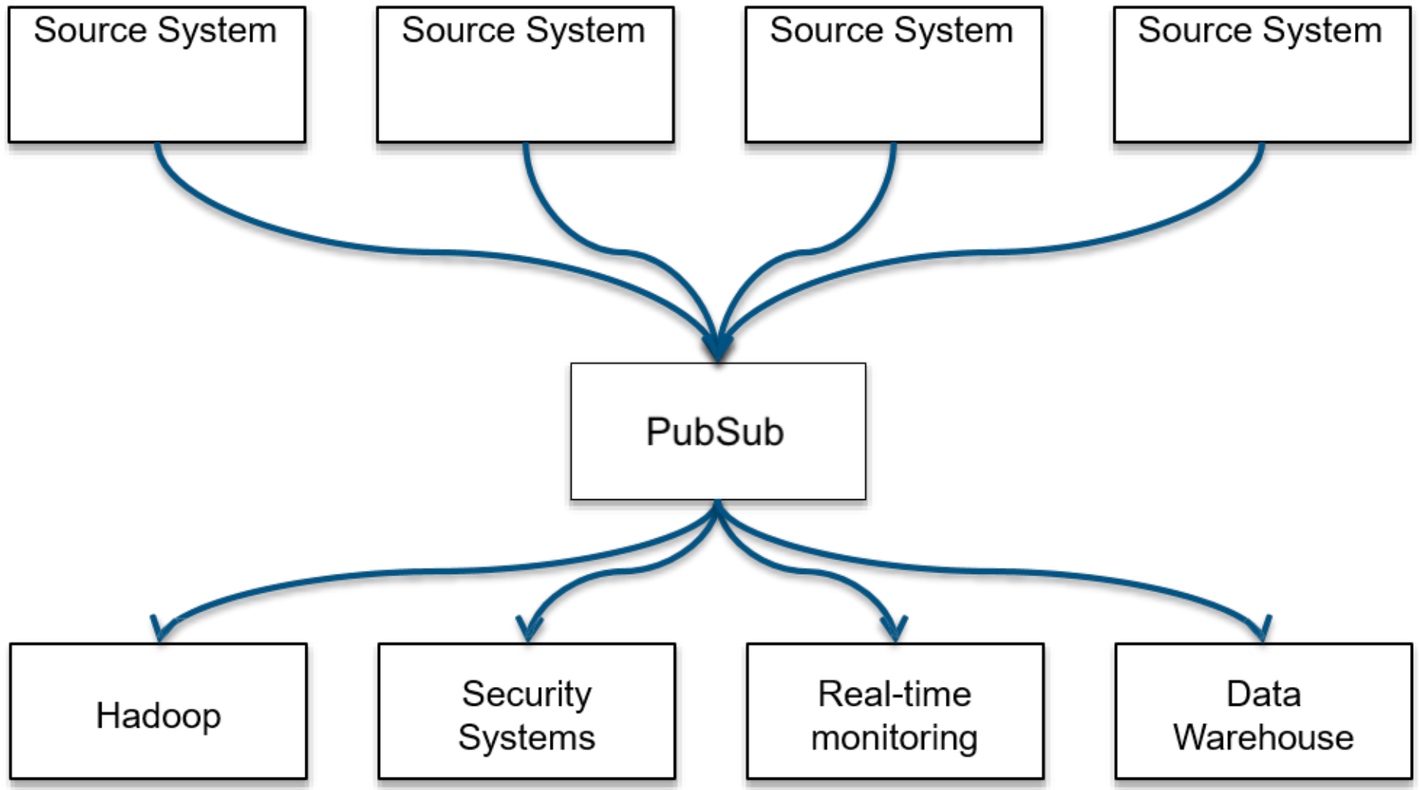
Motivation

Things end up messy when sources and sinks are coupled!



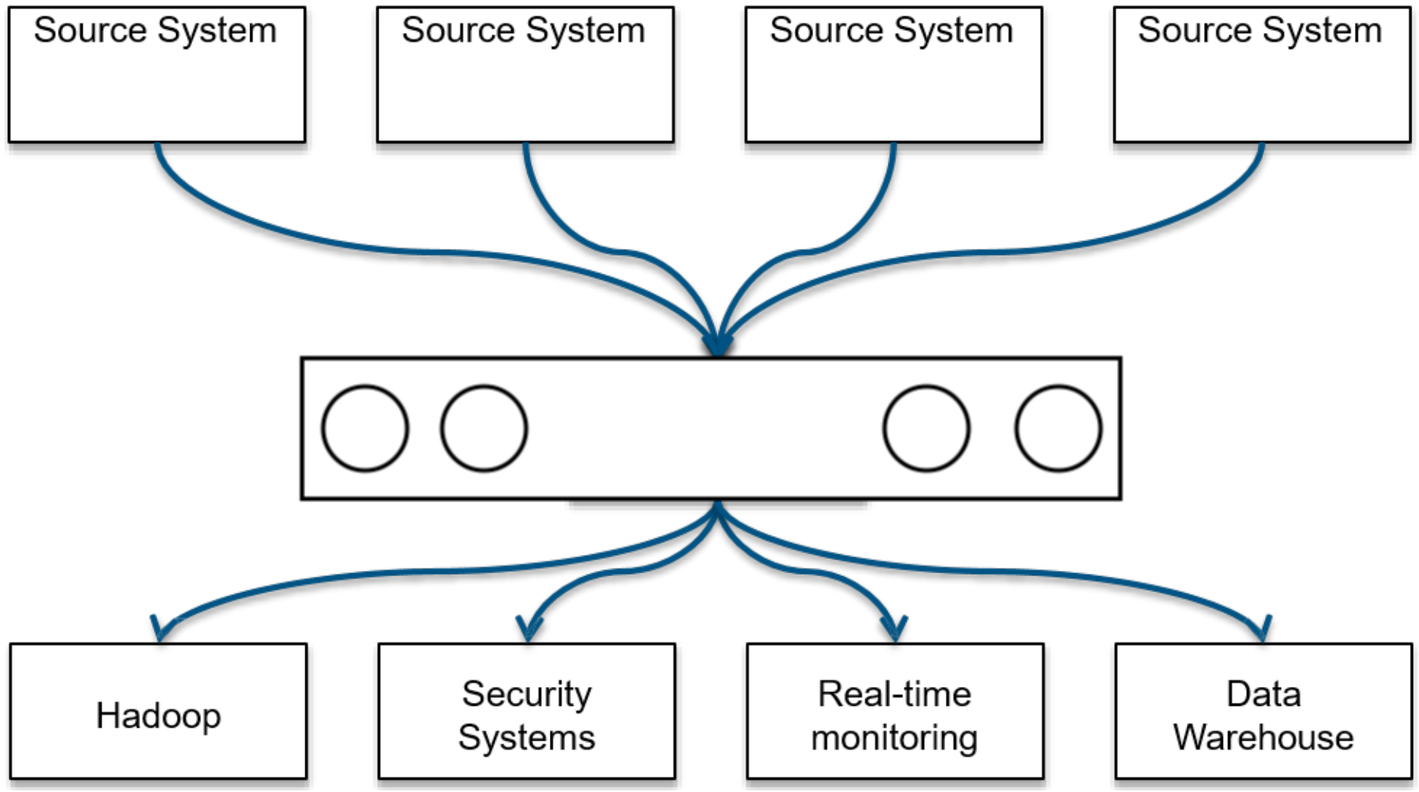
An alternative: Publish/Subscribe

PubSubs decouple data sources and their consumers making communication asynchronous and processing scalable.



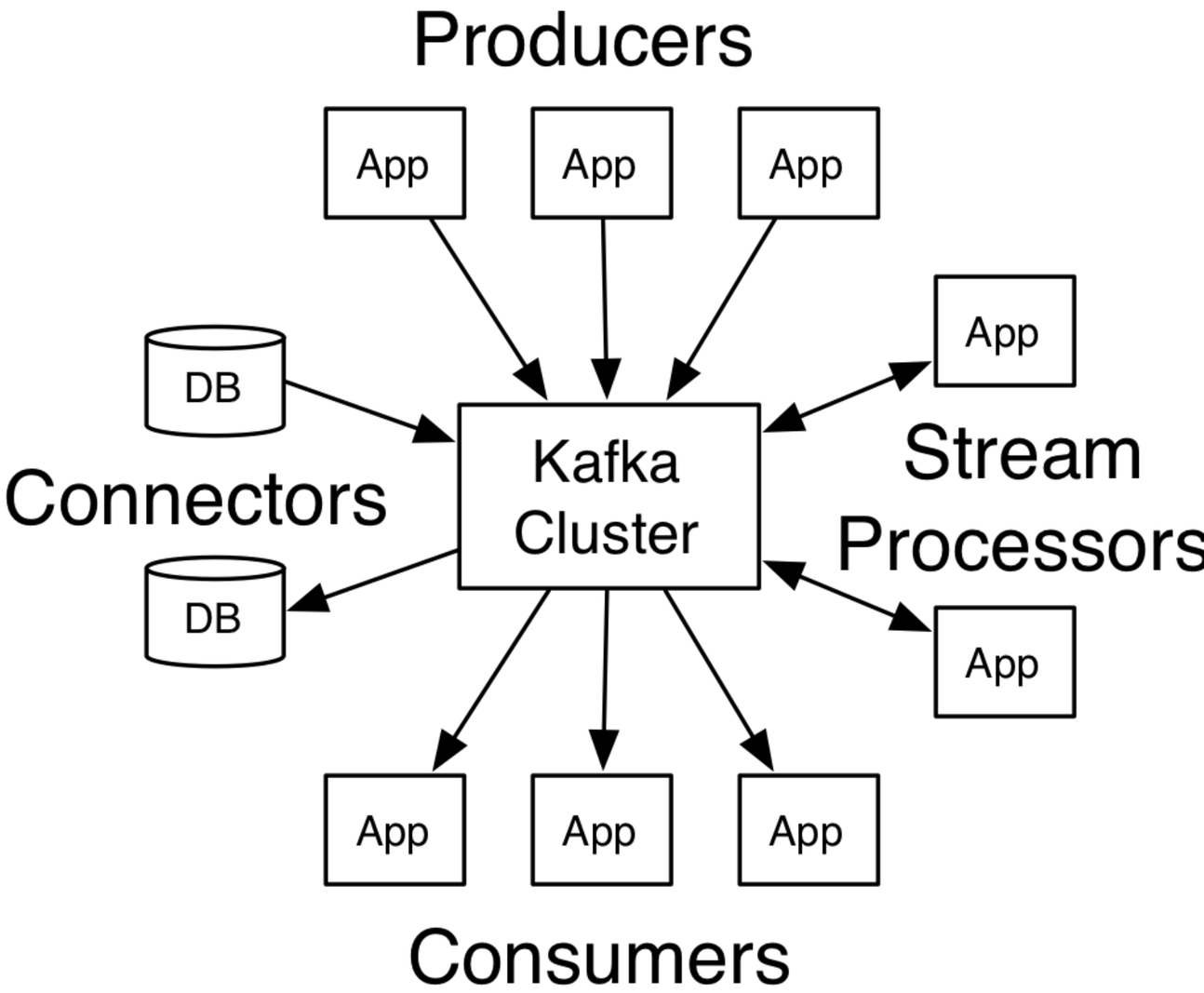
An alternative: Publish/Subscribe

PubSubs organize messages logically so that it is easier for the interested consumers to access.



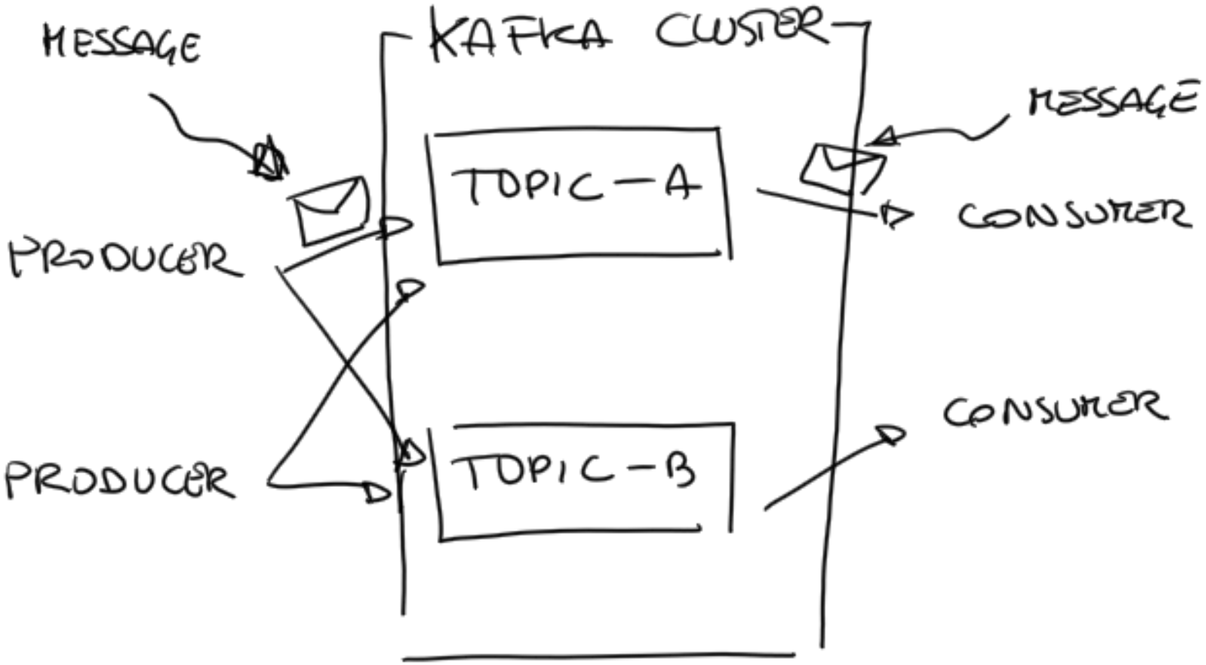
Apache Kafka

Apache Kafka is an horizontally scalable, fault-tolerant, publish-subscribe system. It can process over 1 trillion messages without neglecting durability, i.e., it persists data on disk.

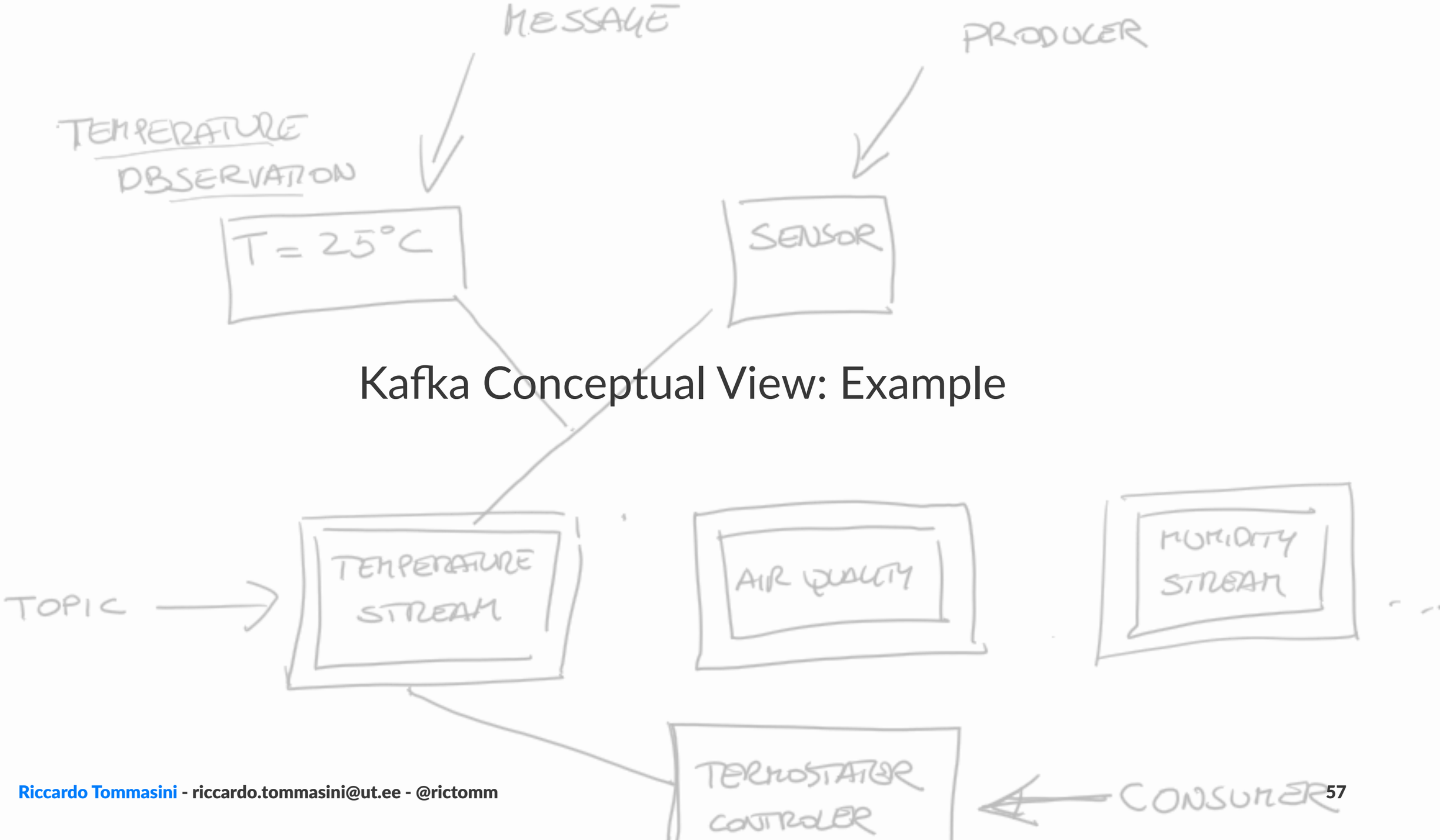


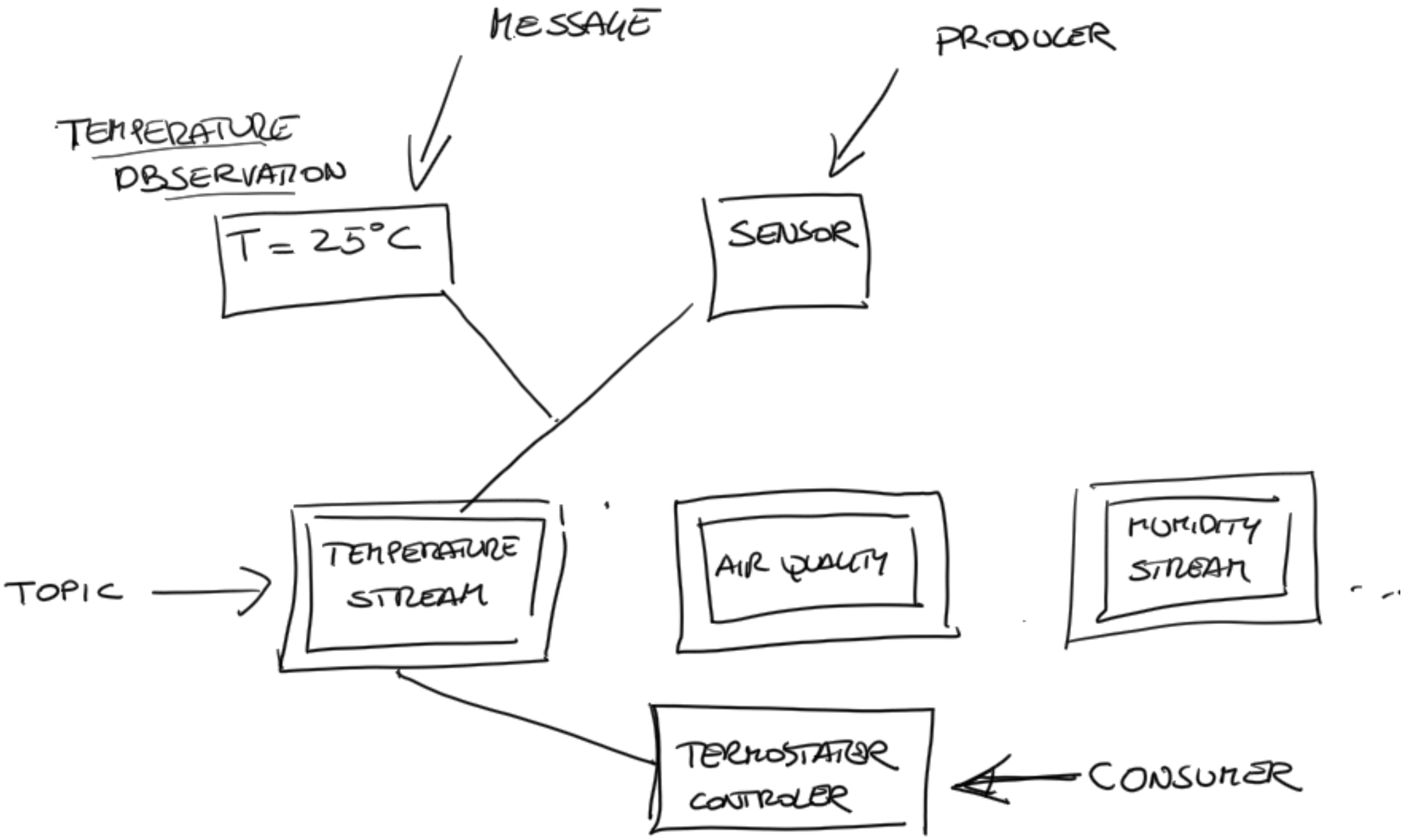
Kafka Conceptual View

- **Messages**, the basic unit in Kafka, are organized in **Topics**
- **Producers** write messages topics
- **Consumers** read messages by from topics

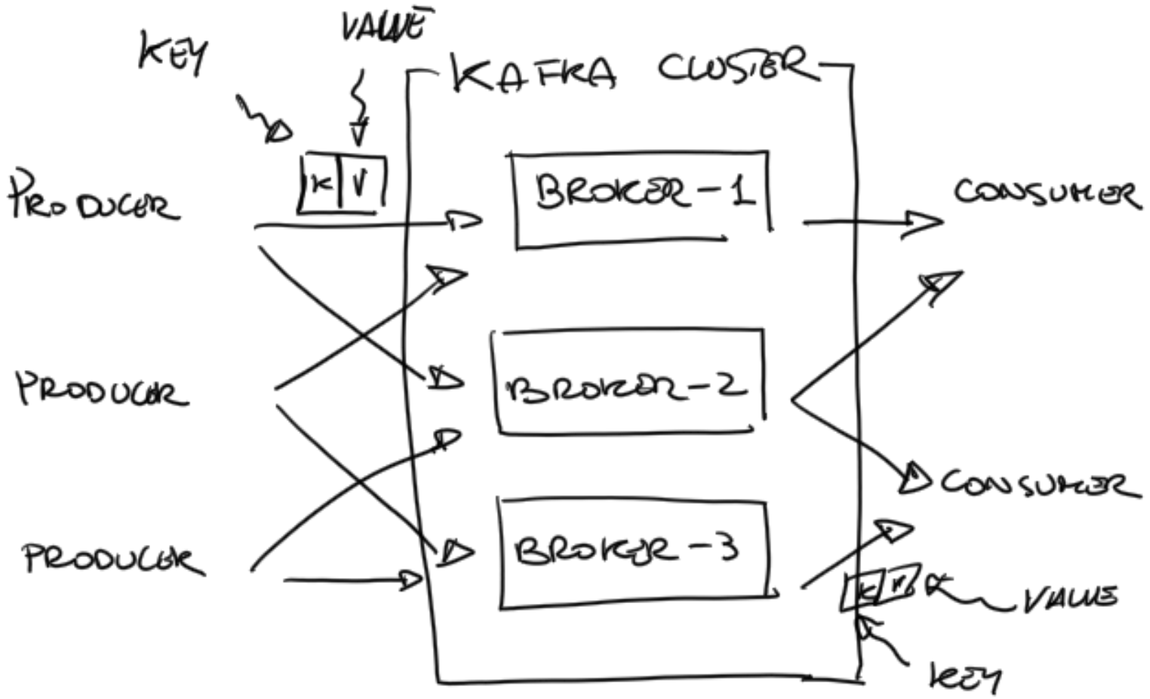


Kafka Conceptual View: Example




MESSAGE
PRODUCER
TEMPERATURE OBSERVATION
T = 25°C
SENSOR
TOPIC
TEMPERATURE STREAM
AIR QUALITY
HUMIDITY STREAM
THERMOSTATIC CONTROLLER
CONSUMER

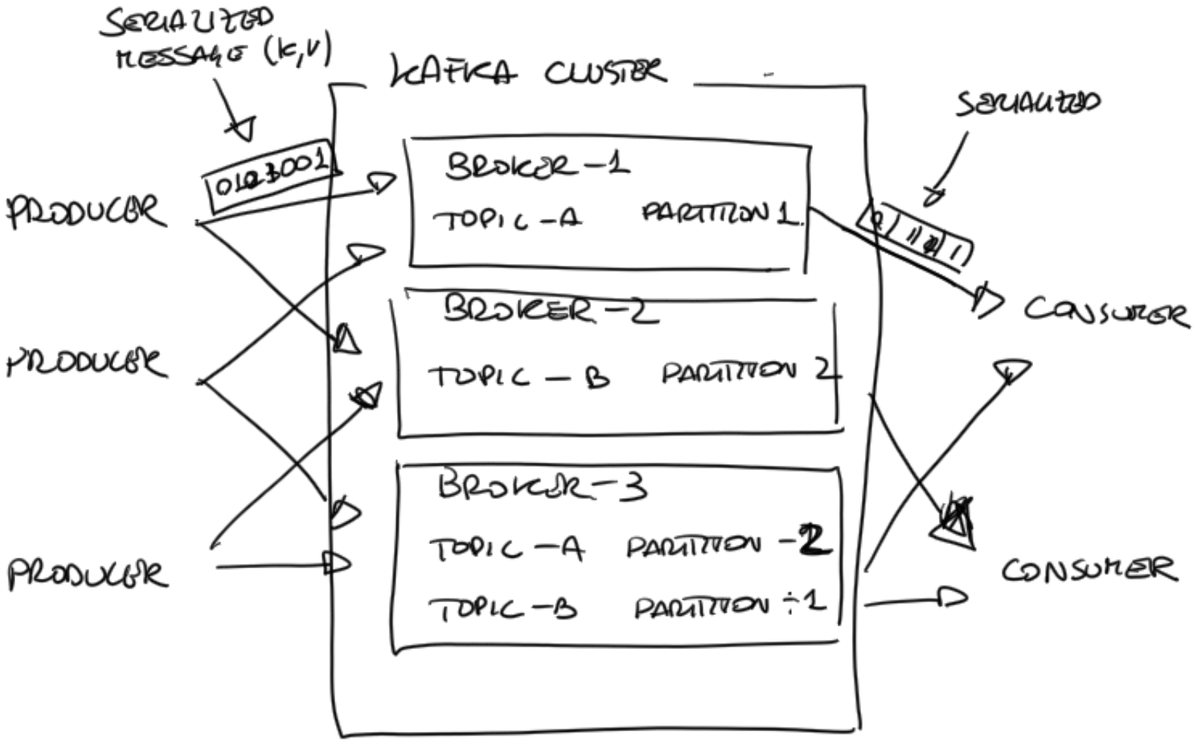
Kafka Logical View



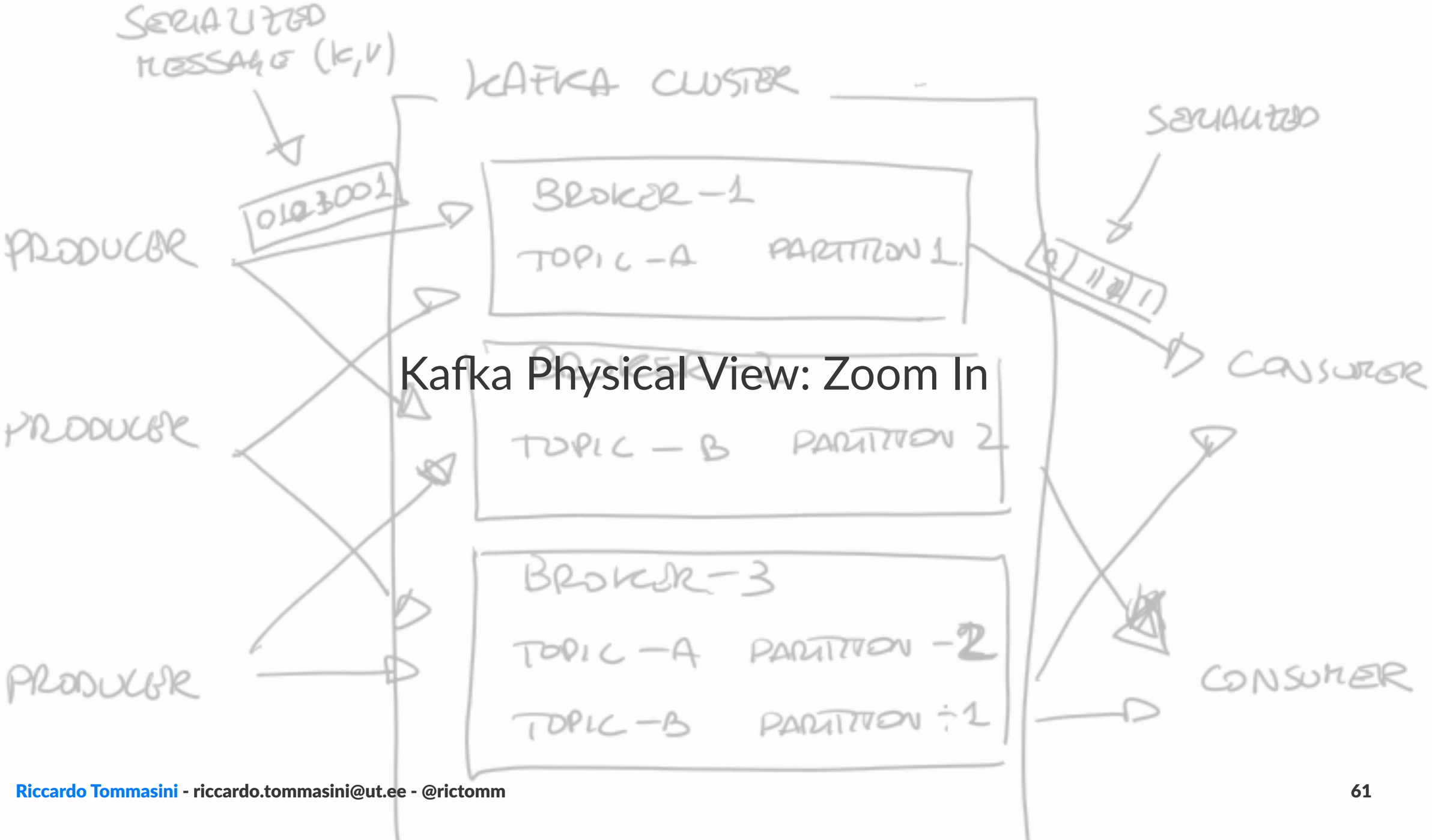
- **Messages** are key-value pairs
- **Brokers** are the main component inside the Kafka Cluster.
- **Producers** write messages to a certain broker
- **Consumers** read messages by from a certain broker

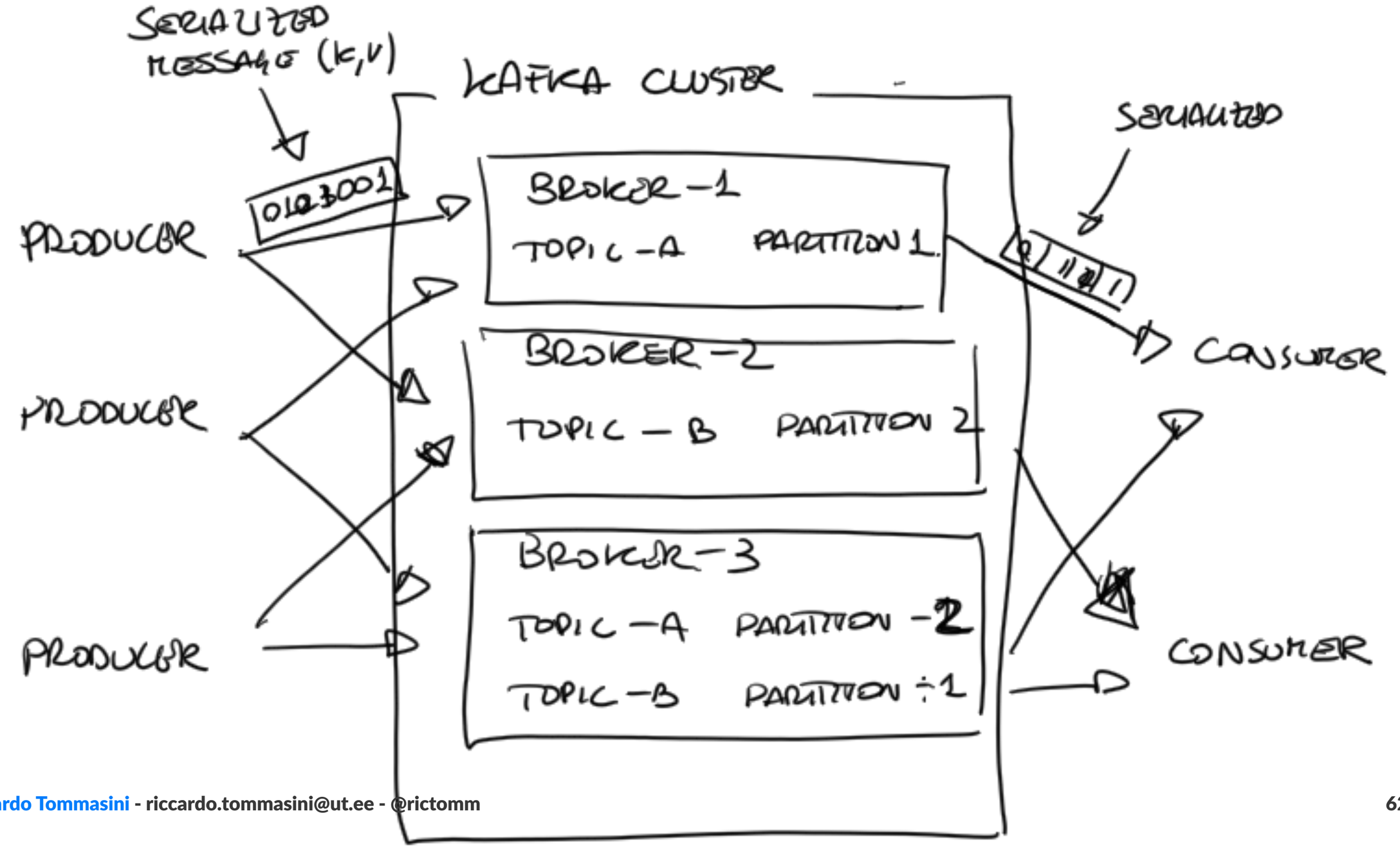
Kafka Physical View

- **Topics** are partitioned across brokers using the message **Key**.
- Typically, **Producers** has the message key to determine the partition. Also they serialize the message
- **Consumers** read messages by from brokers and de-serialize them



Kafka Physical View: Zoom In




SERIALIZED MESSAGE (K,V)
KAFKA CLUSTER
SERIALIZED
PRODUCER
PRODUCER
PRODUCER
BROKER-1
TOPIC-A
PARTITION 1
BROKER-2
TOPIC-B
PARTITION 2
BROKER-3
TOPIC-A
PARTITION -2
TOPIC-B
PARTITION ÷1
CONSUMER
CONSUMER

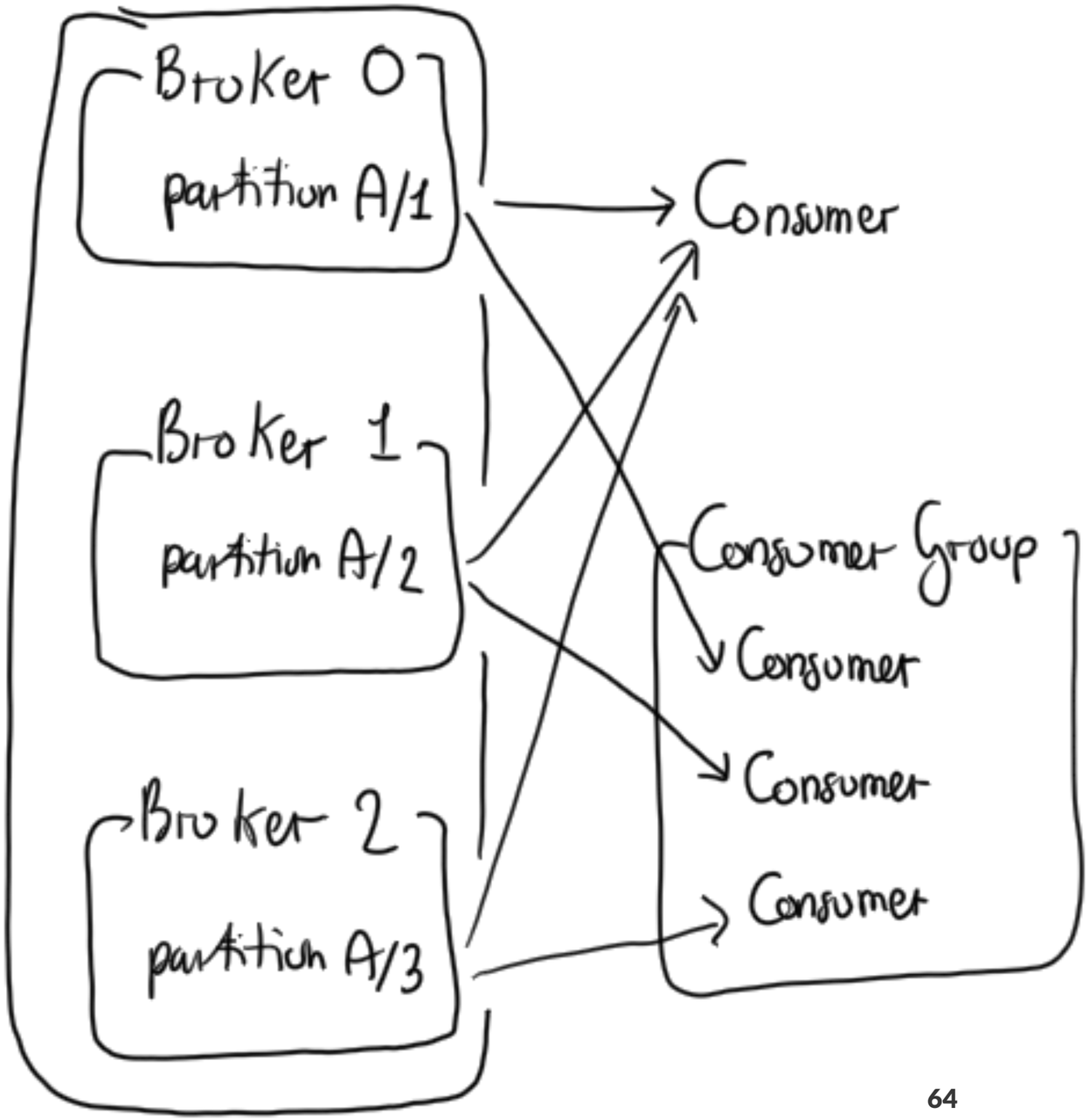
Topics Partitions

Producers shard data over a set of Partitions

- Each Partition contains a subset of the Topic's messages
- Typically, the message key is used to determine which Partition a message is assigned to
- Each Partition is an ordered, immutable log of messages

Topics Partitions and Distributed Consumption

- Different Consumers can read data from the same Topic
  - By default, each Consumer will receive all the messages in the Topic
- Multiple Consumers can be combined into a Consumer Group
  - Consumer Groups provide scaling capabilities
  - Each Consumer is assigned a subset of Partitions for consumption



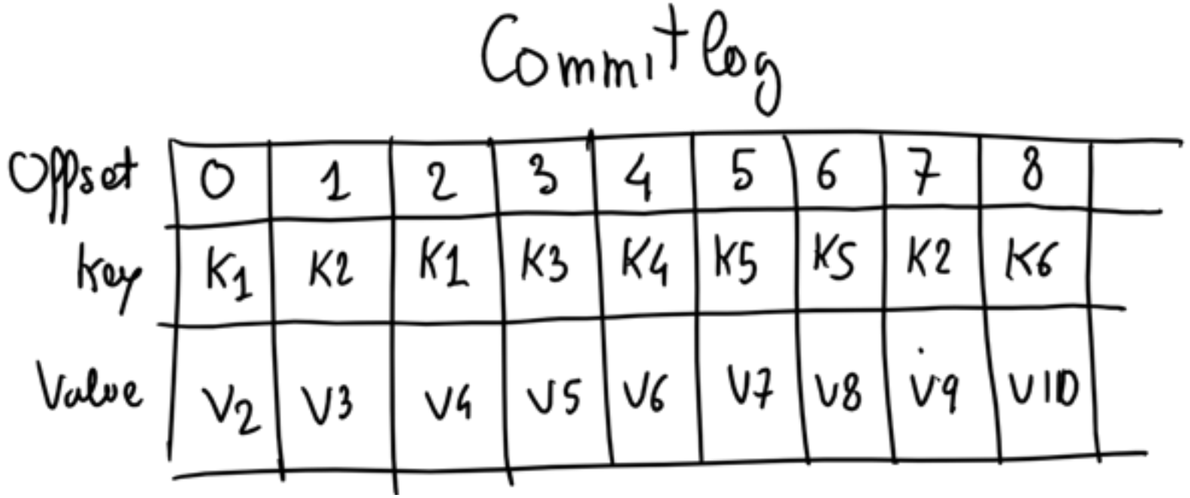
Apache Kafka Internals

Messages and Metadata

Messages are Key-Value pairs and there is not restriction on what each of them can be.

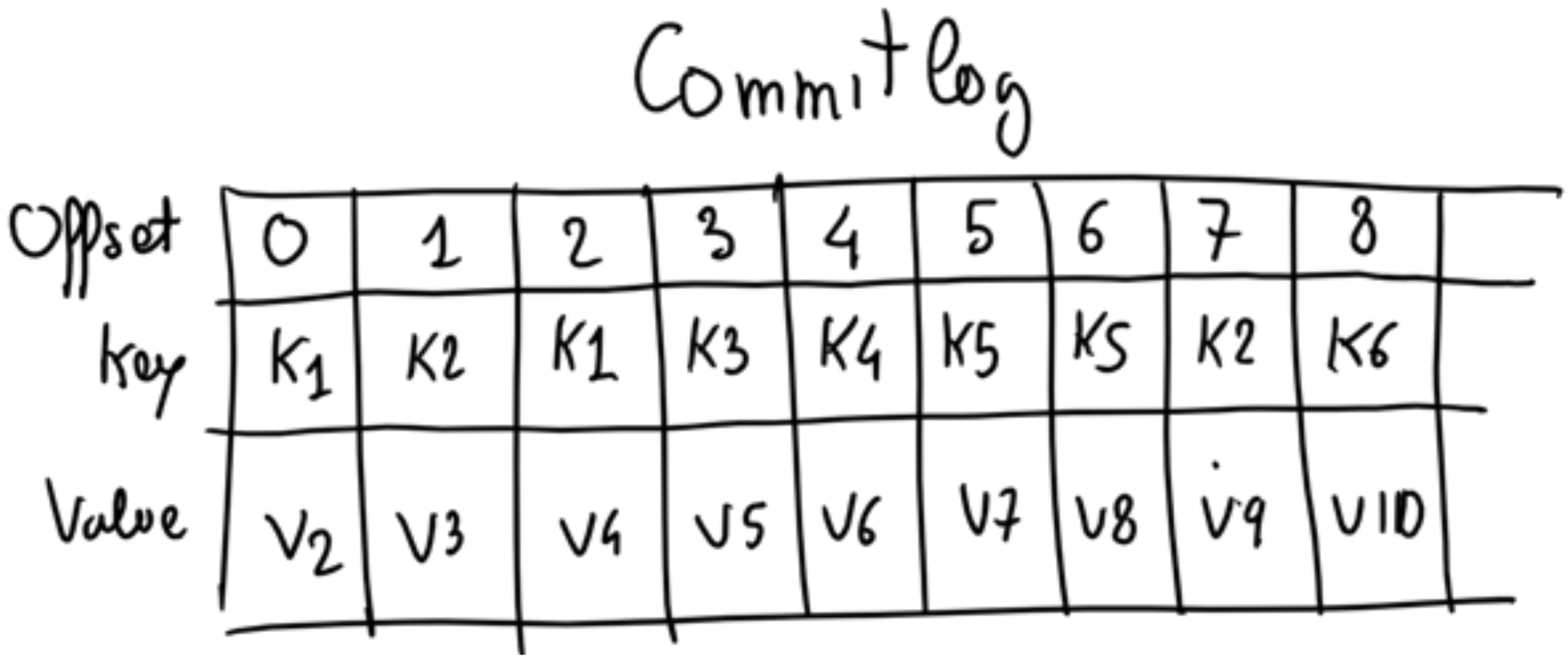
Additionally, messages are enriched with metadata:

- Offset
- Timestamp
- Compression type
- Magic byte
- Optional message headers API
- Application teams can add custom key-value paired metadata to messages
- Additional fields to support batching, exactly once semantics, replication protocol



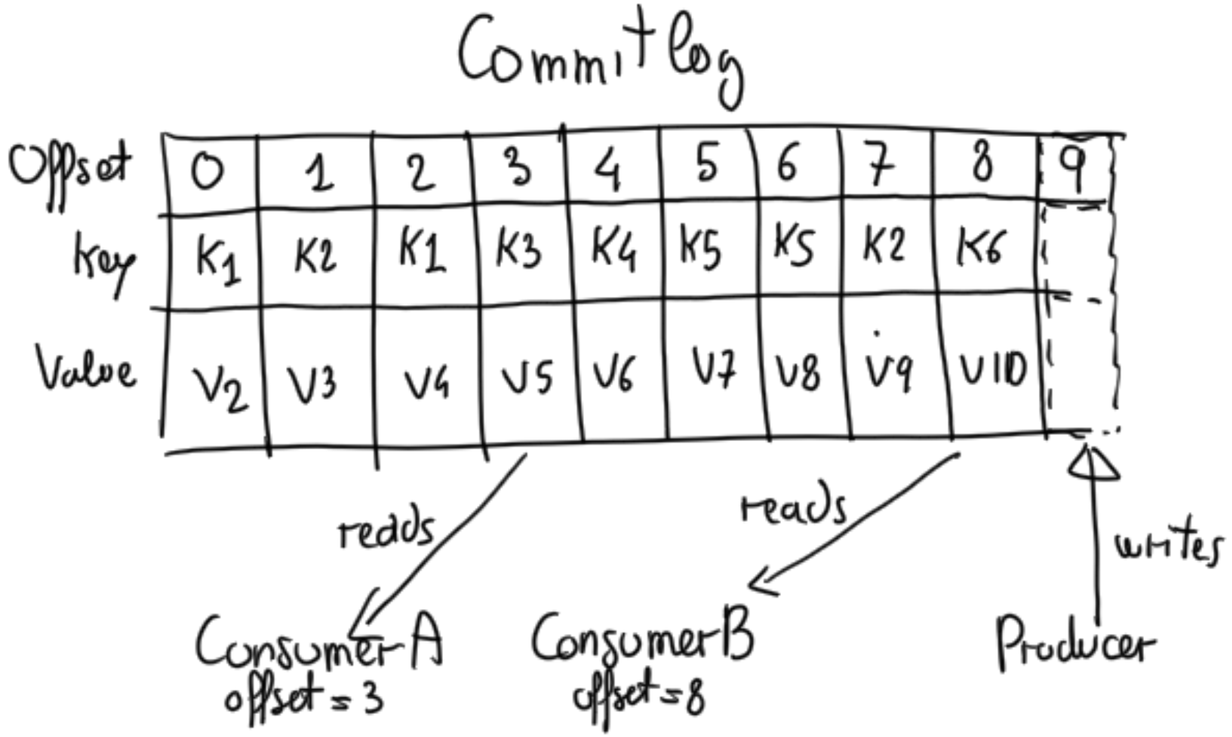
Topics Partitions: Physical View

Each Partition is stored on the Broker's disk as one or more log files Each message in the log is identified by its offset number



Topics Partitions: Physical View

Messages are always appended. Consumers can consume from different offset. Brokers are single thread to guarantee consistency



Topics Partitions: Load Balancing

Producers use a partition strategy to assign each message a partition

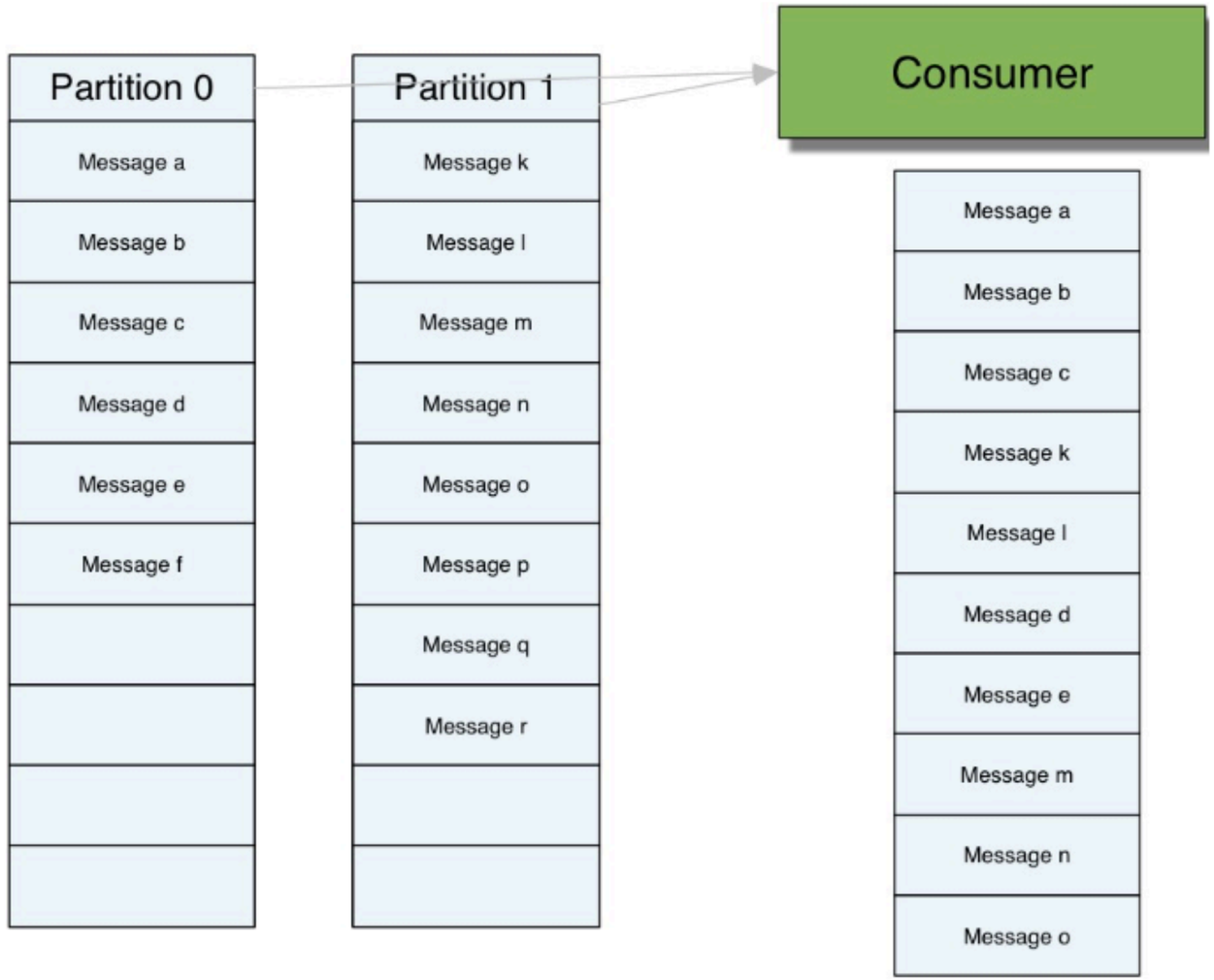
- To ensure load balancing across the Brokers
- To allow user-specified key

You can customize the partition strategy, but!

- it must ensure load balancing across the Brokers too, i.e., hash(key) % number*of*partitions
- if key is not specified, messages are sent to Partitions on a round-robin basis

Important: About Ordering

If there are multiple Partitions, you will not get total ordering across all messages when reading data

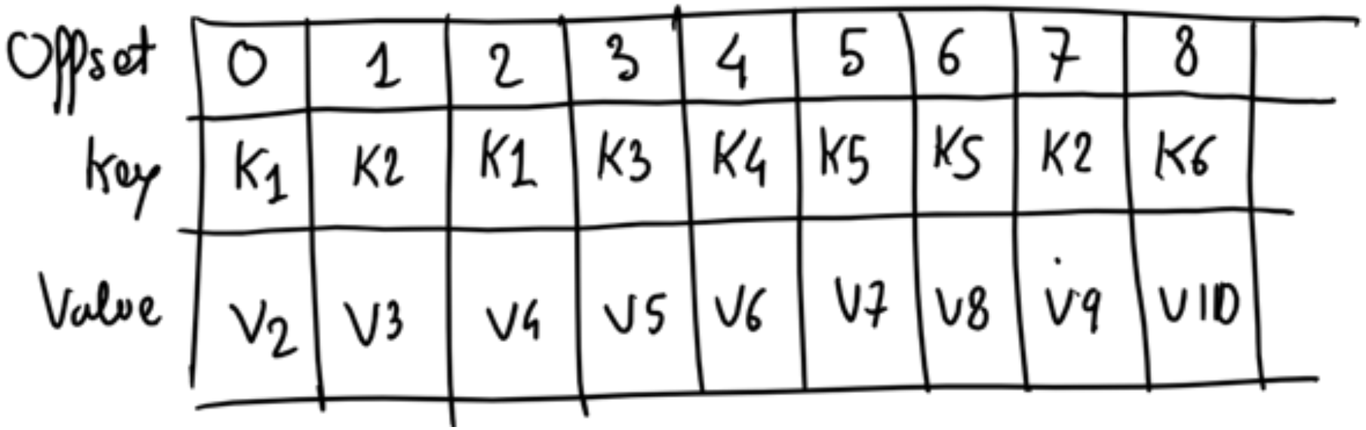


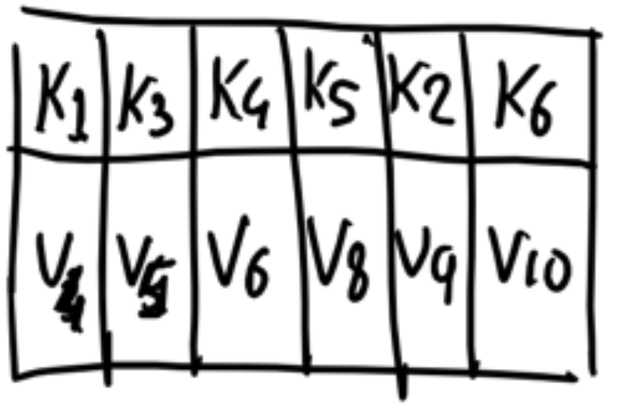
Log Retention

- Duration default: messages will be retained for seven days
- Duration is configurable per Broker by setting
  - a time period
  - a size limit
- Topic can override a Broker's retention policy
- When cleaning up a log
  - the default policy is delete
  - An alternate policy is compact

Log Compaction

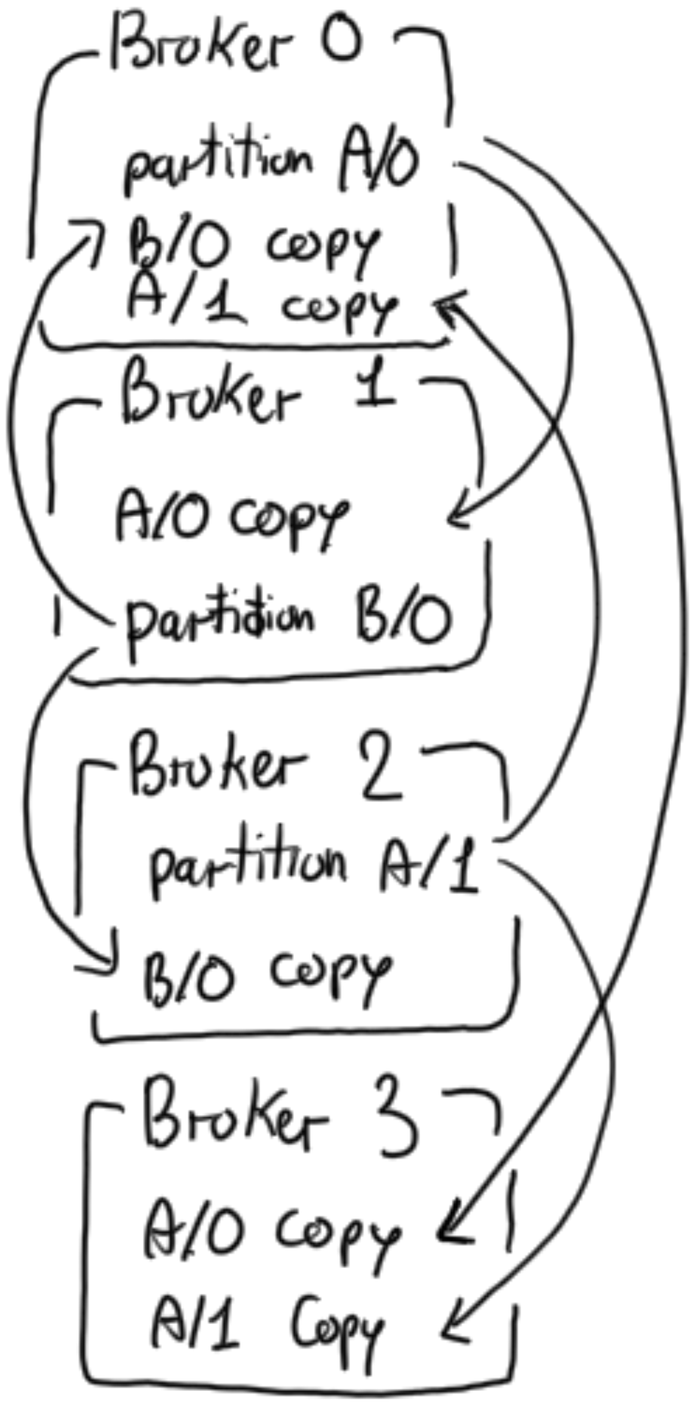
A compacted log retains at least the last known message value for each key within the Partition
Before After





Fault Tolerance via a Replicated Log

- Kafka maintains replicas of each partition on other Brokers in the cluster
  - Number of replicas is configurable
- One Broker is the leader for that Partition
  - All writes and reads go to and from the leader
  - Other Brokers are followers
- Replication provides fault tolerance in case a Broker goes down

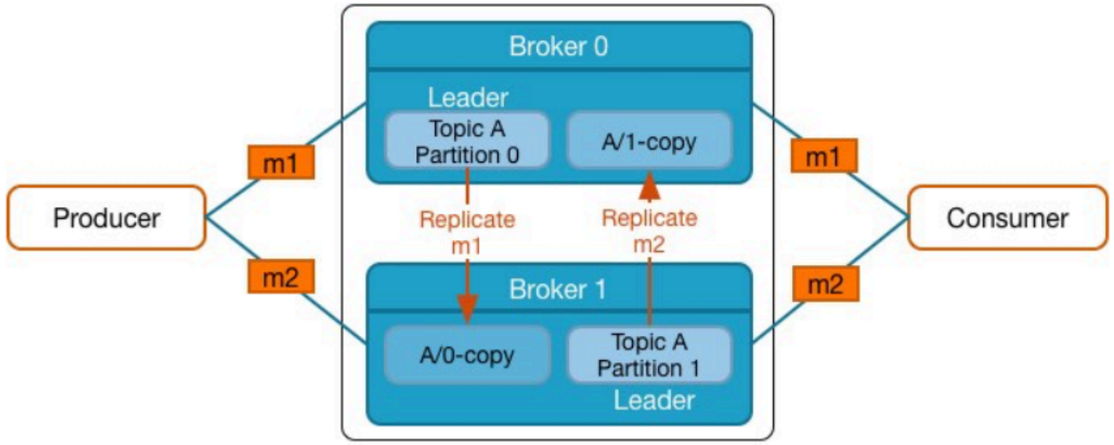


Important: Clients do not Access Followers

It is important to understand that Producers and Consumers only write/read to/from the leader

- Replicas only exist to provide reliability in case of
  Broker failure
- If a leader fails,
  the Kafka cluster will
  elect a new leader
  from among the followers

In the diagram, m1 hashes to Partition 0 and m2 hashes to Partition 1



Delivery Semantics

- At least once
  - Messages are never lost but may be redelivered
- At most once
  - Messages are lost but never redelivered
- Exactly once
  - Messages are delivered once and only once

Zookeeper

- ZooKeeper is a centralized service that stores configurations for distributed applications
- Kafka Brokers use ZooKeeper for a number of important internal features
  - Cluster management
  - Failure detection and recovery
  - Access Control List (ACL) storage

Quiz

Provide the correct relationship - 1:1, 1:N, N:1, or N:N -

- Broker to Partition - ?
- Key to Partition - ?
- Producer to Topic - ?
- Consumer Group to Topic - ?
- Consumer (in a Consumer Group) to Partition - ?

Quiz

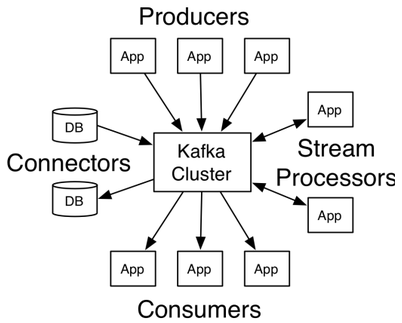
Provide the correct relationship - 1:1, 1:N, N:1, or N:N -

- Broker to Partition - N:N
- Key to Partition - N:1
- Producer to Topic - N:N
- Consumer Group to Topic - N:N
- Consumer (in a Consumer Group) to Partition - 1:N

Getting Exactly Once Semantics

- Must consider two components
  - Durability guarantees when publishing a message
  - Durability guarantees when consuming a message
- Producer
  - What happens when a produce request was sent but a network error returned before an ack?
  - Use a single writer per partition and check the latest committed value after network errors
- Consumer
  - Include a unique ID (e.g. UUID) and de-duplicate.
  - Consider storing offsets with data

Systems Overview: **Apache Kafka**



- Apache Kafka is a scalable replicated commit log that enables stream processing at scale.
- It can handle huge numbers of concurrent reads and writes
- It comes with connector to a number of Big Data Framework, e.g., `Storm`, `Samza`, `Flink`, `Spark`.
- It persists messages on disk and replicated within the cluster.

References:

Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." Proceedings of the NetDB. Vol. 11. 2011.
Wang, Guozhang, et al. "Building a replicated logging system with Apache Kafka." Proceedings of the VLDB Endowment 8.12 (2015): 1654-1655.