Distributed Transport and Streaming

SDU

Agenda

→ Flume

→ Scoop

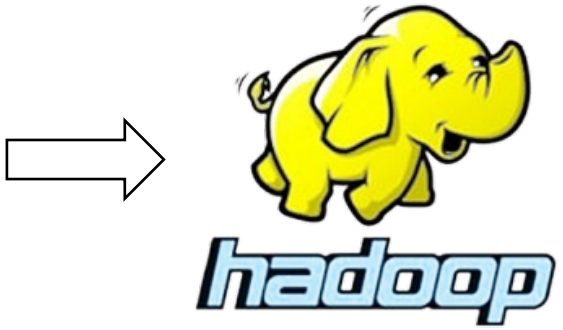
→ Kafka

→ ksqlDB

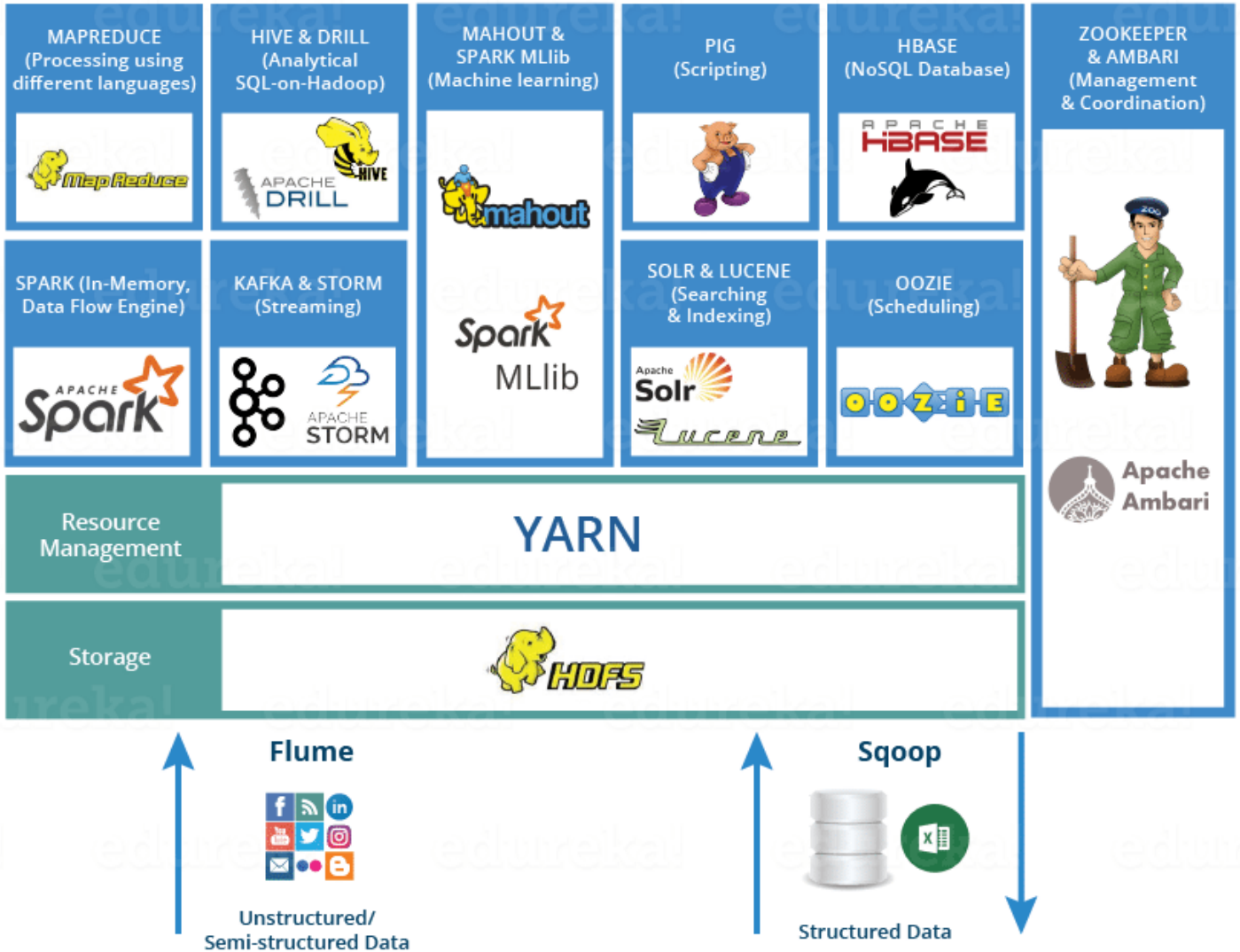
→ Kafka Connect

Flume and Scoop

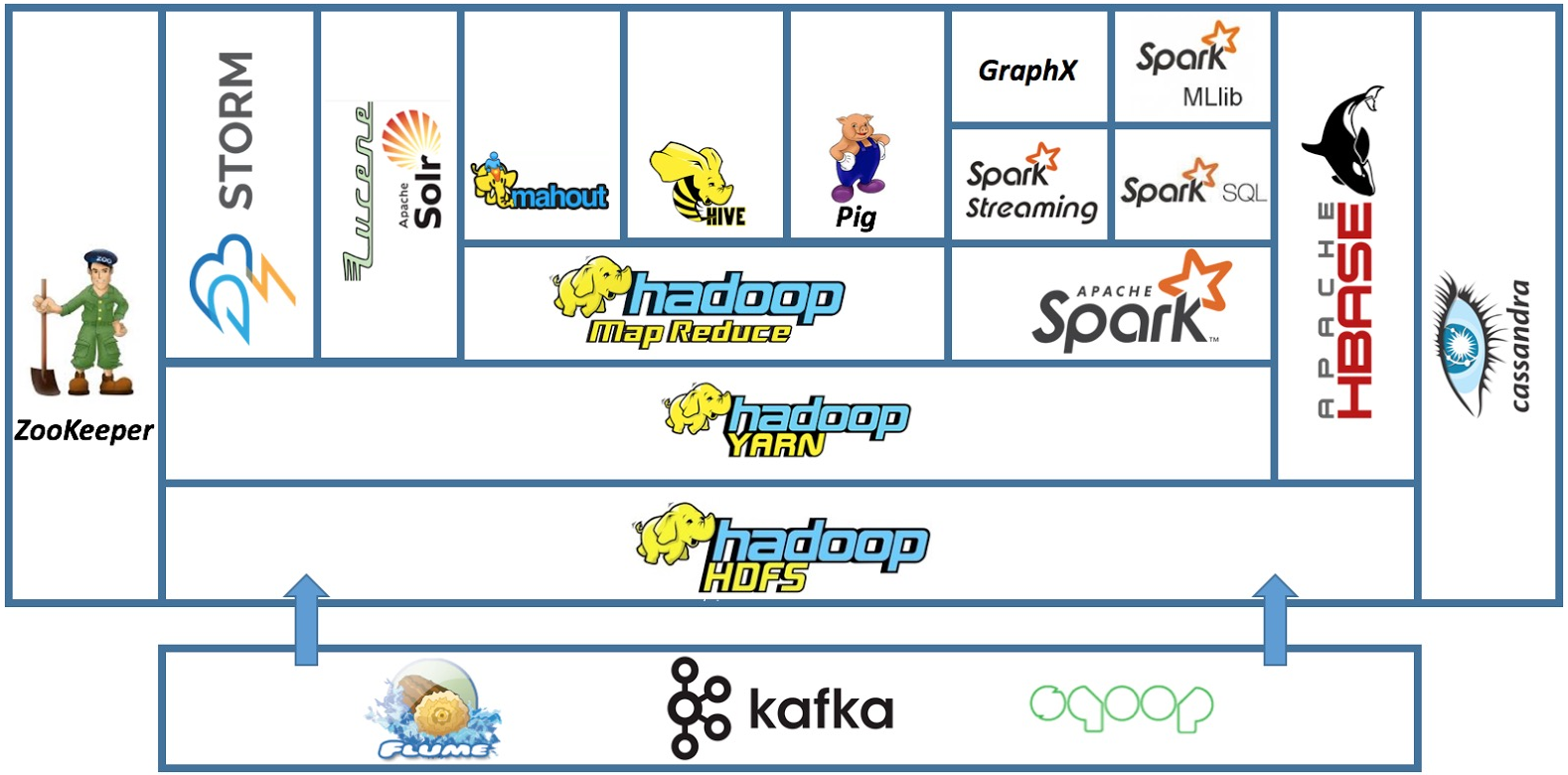
What is Sqoop and Flume



Sqoop and Flume in context of Hadoop



And it has many variations…



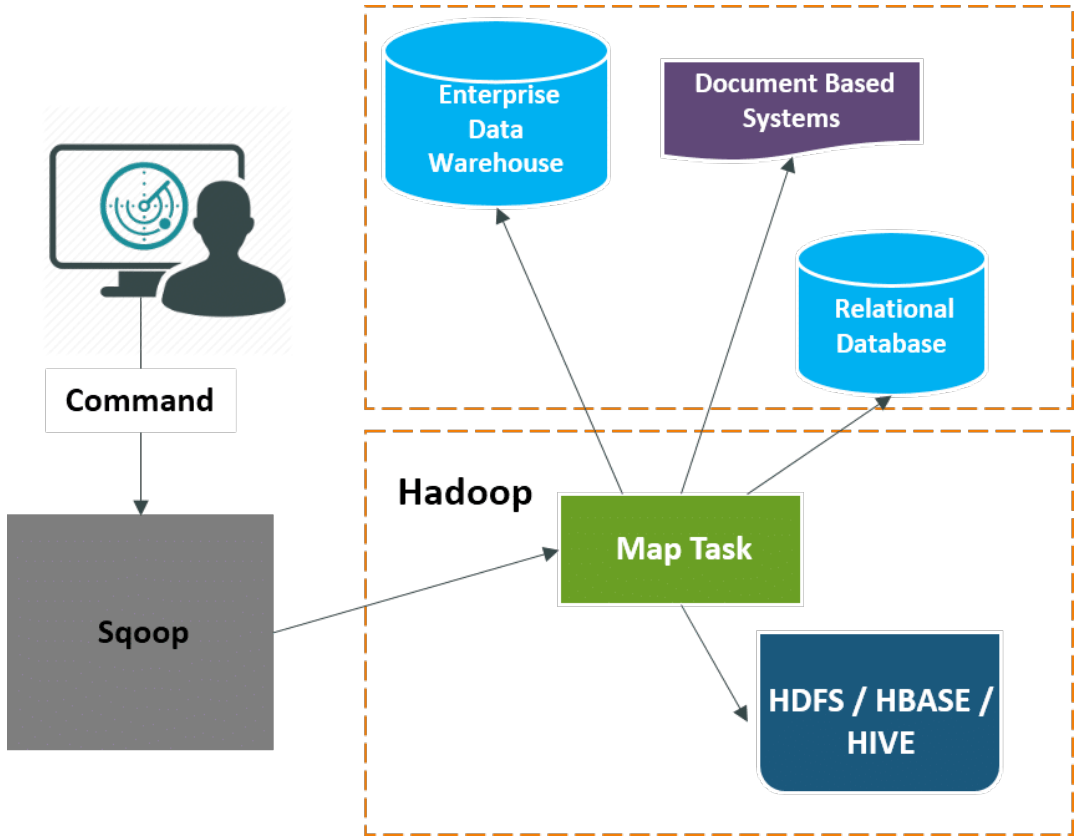
vs.

Sqoop vs. Flume

- → Sqoop
  - → Importing structured data (Oracle, MySQL, Postgres etc.)
  - → Can import directly to HBase and Hive too.
  - → Not Event Driven
- → Flume
  - → Designed for importing logs into Hadoop
  - → Importing unstructured data (twitter, logs, etc.)
  - → Works well with streaming data sources that is generated continuously
  - → Fault tolerant and linearly scalable streaming architecture
  - → Event Driven
- → Commonalities
  - → Both can import data into HDFS

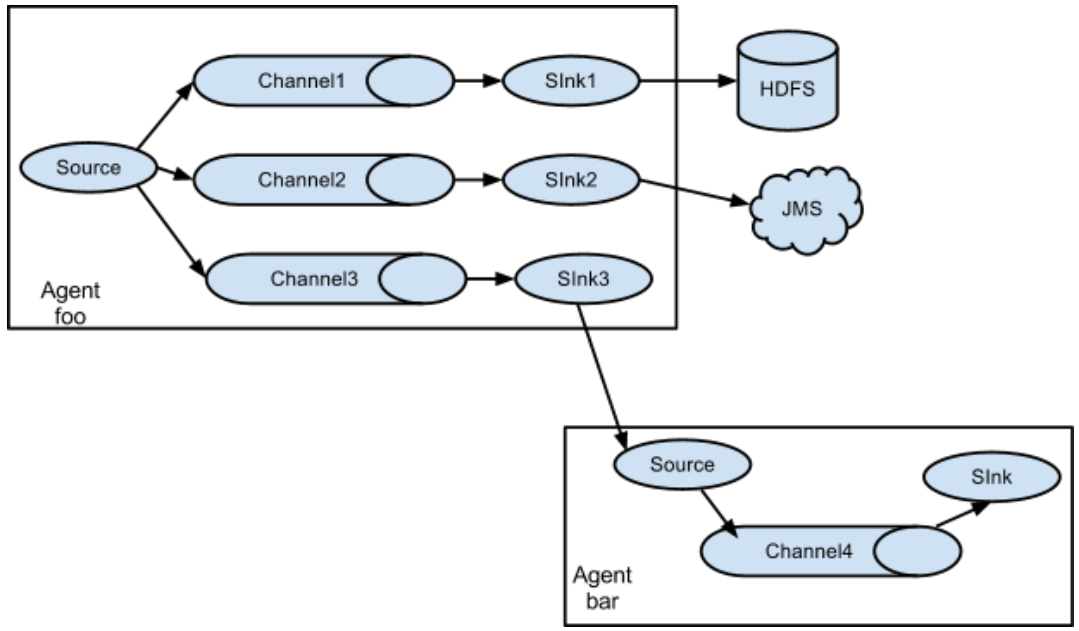
SDU

Sqoop



→ Used for:
- → Import into HDFS/HBase/Hive from databases and more.
- → Export from HDFS/Hbase/Hive to databases and more.
- → Commands are initiated from CLI or REST interfaces
- → Map Tasks define how data is mapped into the new format.

Flume



- → Application for:
  - → Ingesting large amounts of streaming data
  - → Buffer it
  - → Pass it on to processing, storage or applications like Kafka
- → Flume is for high performance ingestion of data
- → Scales Linearly
- → Agents consist of:
  - → Sources
    - → Passively waiting
    - → Actively Polling
  - → Channels
  - → Sinks

Flume Components

- Flume Agent:
  - Is an independent Java virtual machine daemon process which receives the data (events) from clients and transports to the subsequent destination (sink or agent).
- Source:
  - Is the component of Flume agent which receives data from the data generators say, twitter, Facebook, weblogs from different sites and transfers this data to one or more channels in the form of Flume event.
  - The external source sends data to Flume in a format that is recognized by the target Flume source. Example, an Avro Flume source can be used to receive Avro data from Avro clients or other Flume agents in the flow that send data from an Avro sink, or the Thrift Flume source will receive data from a Thrift sink, or a Flume Thrift RPC client or Thrift Clients are written in any language generated from the Flume thrift protocol.
- Channel:
  - Once, the Flume source receives an Event, it stores this data into one or more channel and buffers them till they are consumed by sinks. It acts as a bridge between the source and sinks. These channels are implemented to handle any number of sources and sinks.
- Sink:
  - It stores the data into the centralized stores like HDFS, Kafka and HBase.

(Slide text from: https://acadgild.com/blog/streaming-twitter-data-using-flume)

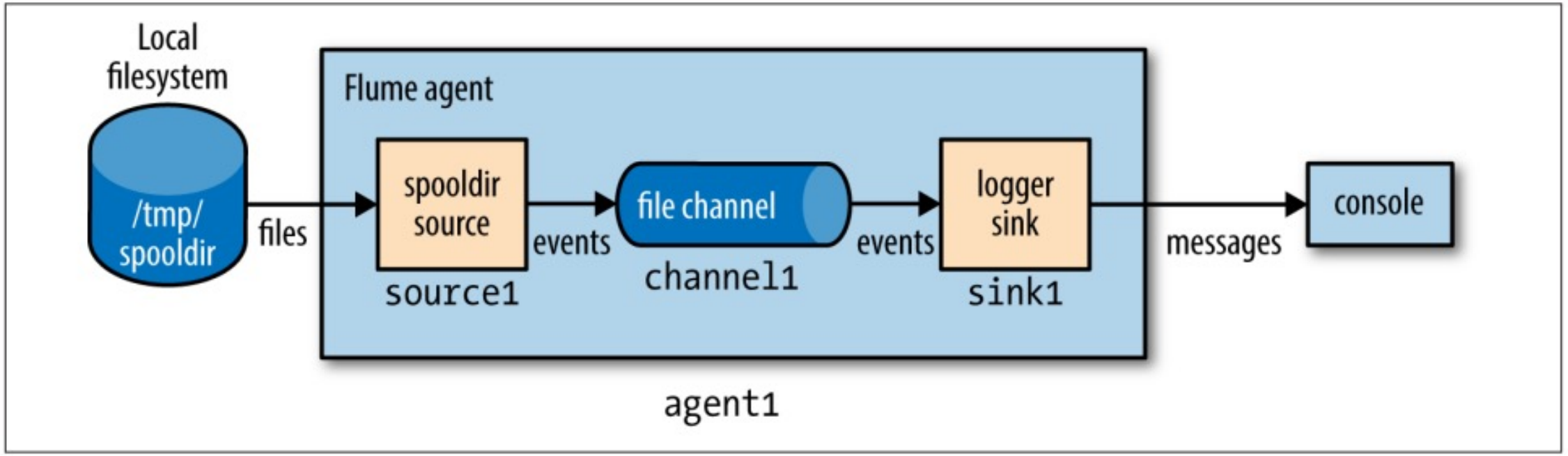


Figure 14-1. Flume agent with a spooling directory source and a logger sink connected by a file channel

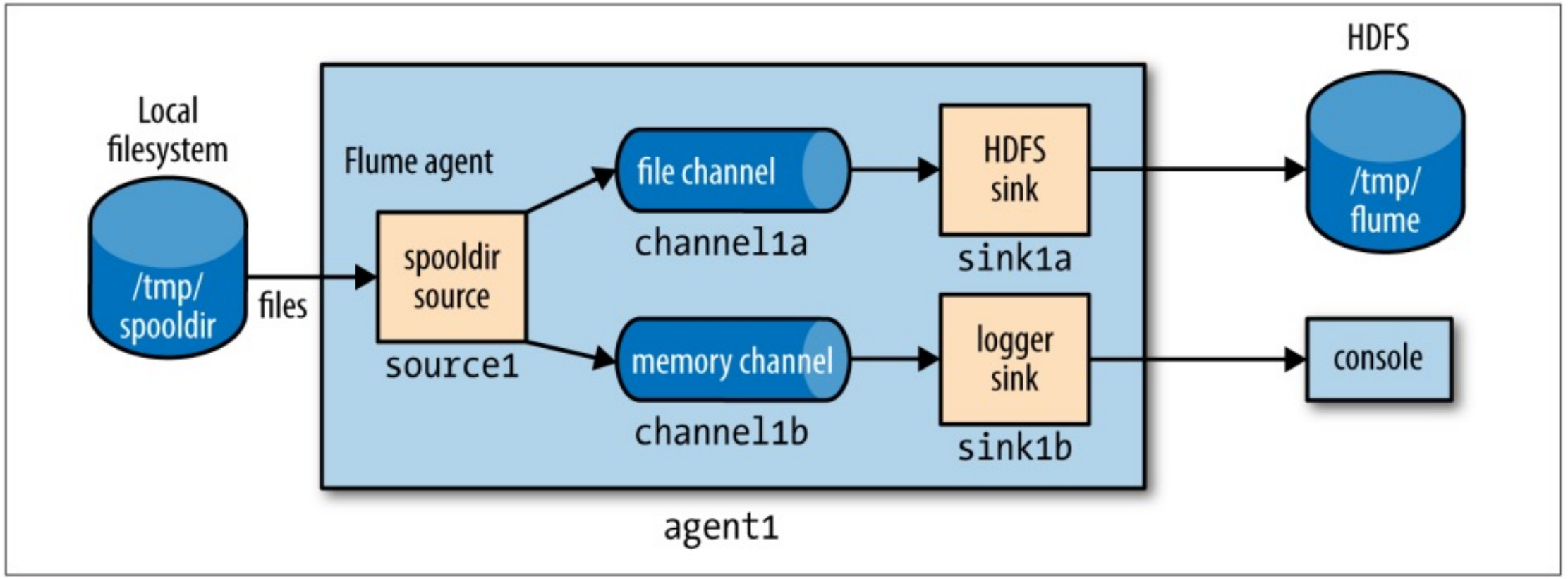


Figure 14-2. Flume agent with a spooling directory source and fanning out to an HDFS sink and a logger sink

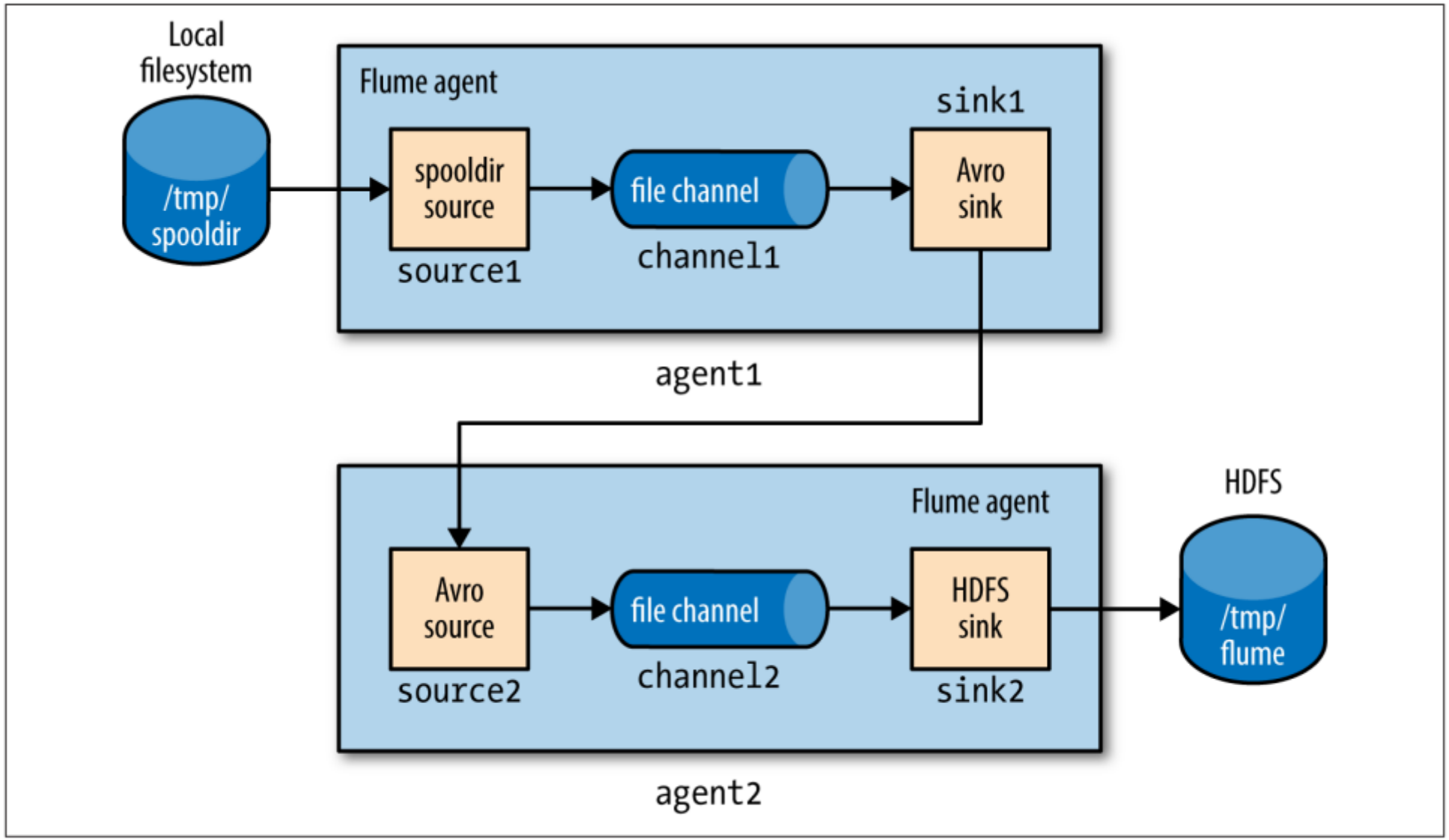


Figure 14-4. Two Flume agents connected by an Avro sink-source pair

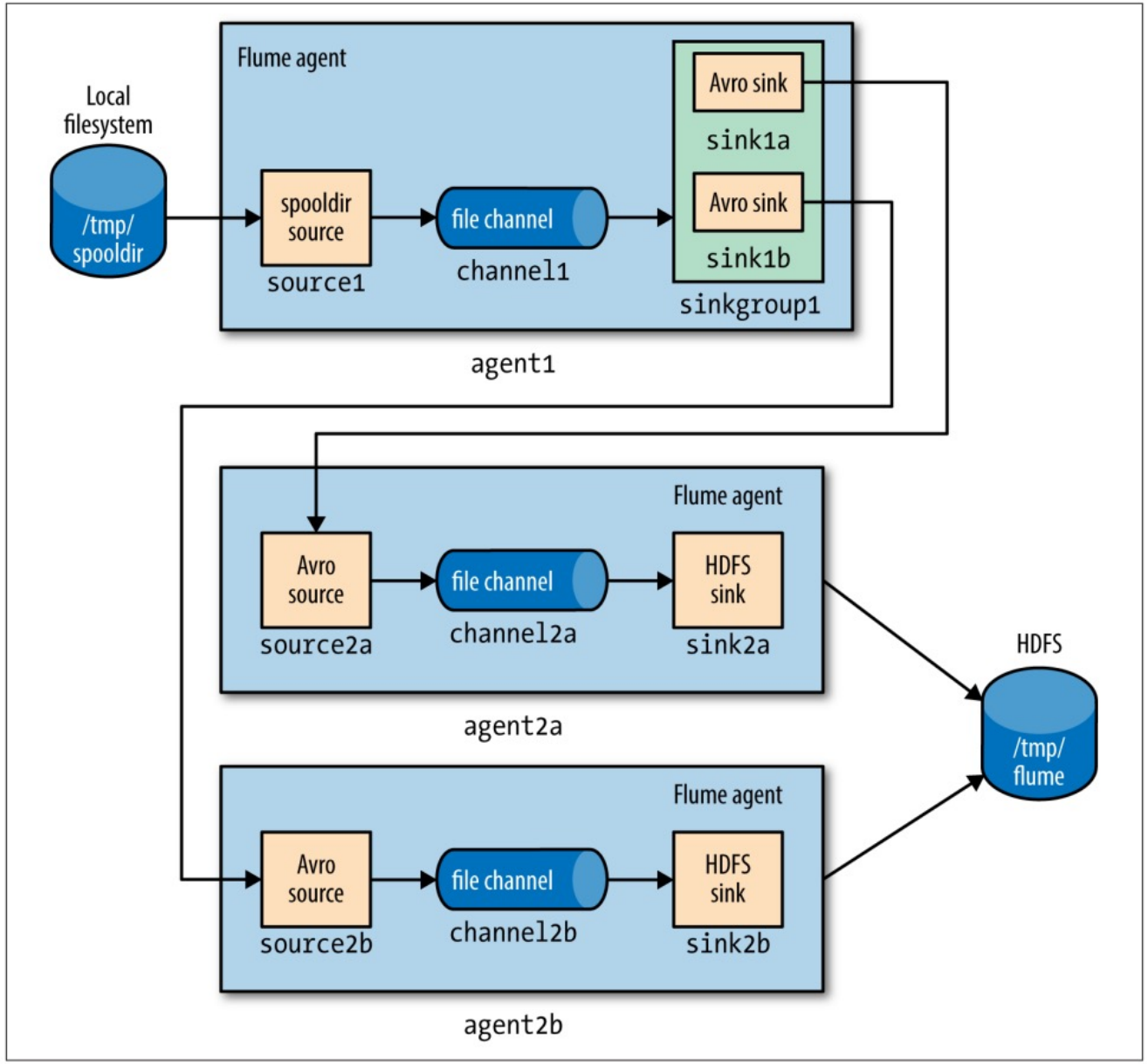
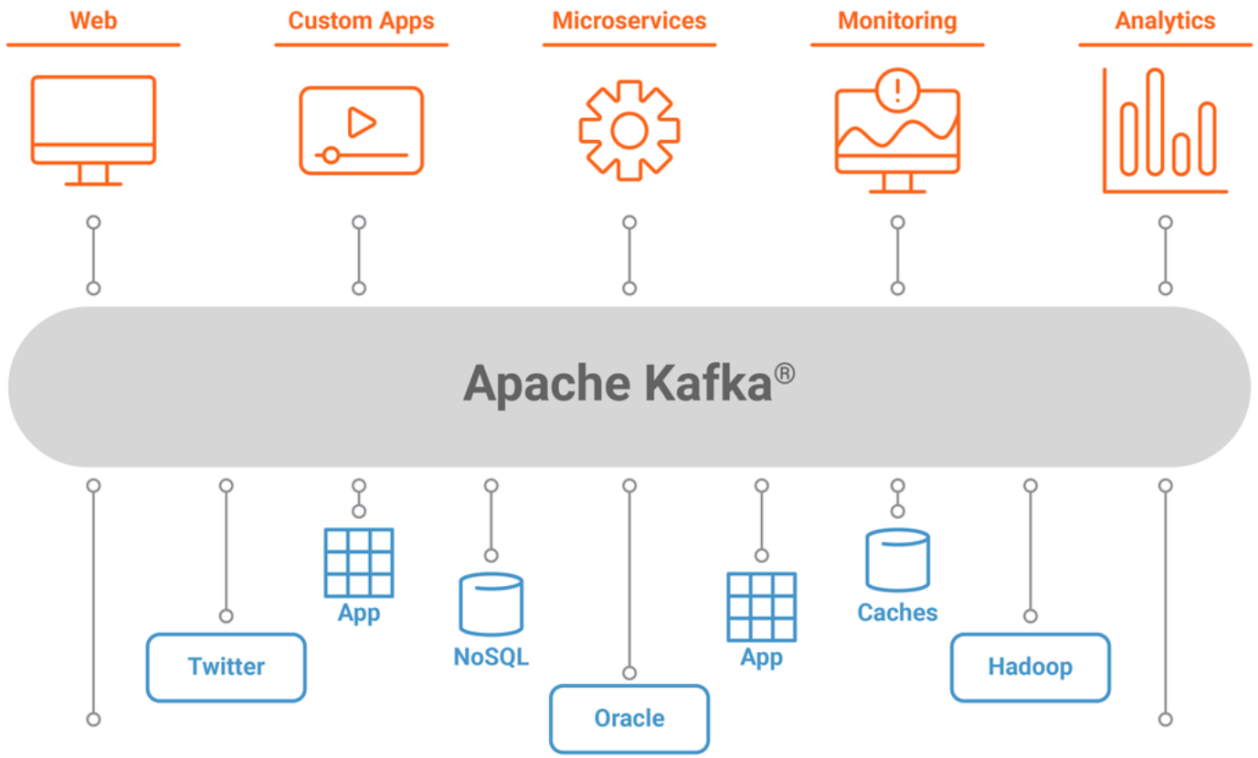


Figure 14-6. Load balancing between two agents

Kafka

What is Kafka



→ A server that allows for:

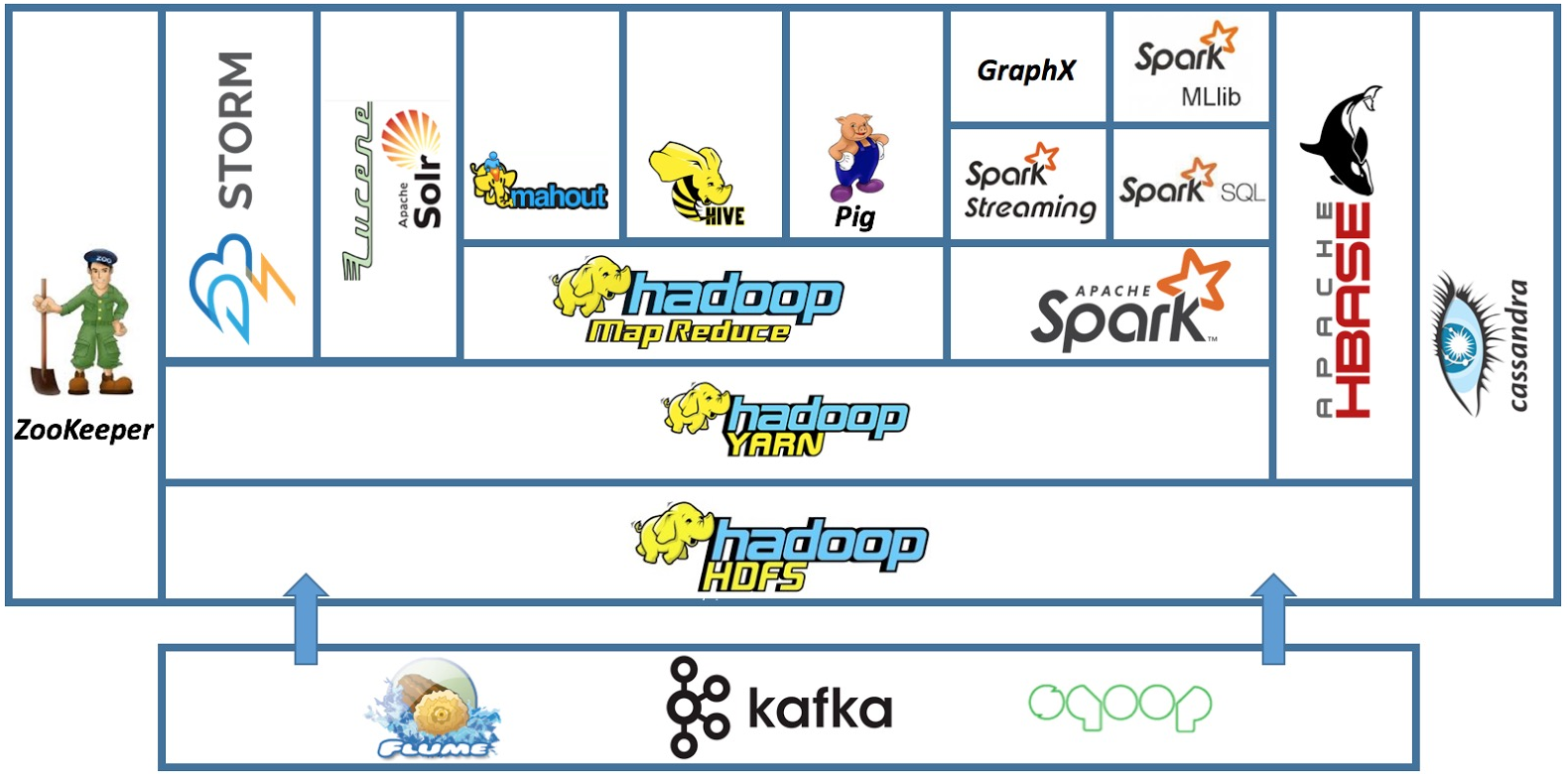
- → Publish and subscribe to streams of records, like a message queue or enterprise messaging system.
- → Store streams of records in a fault-tolerant durable way. (Temporarily, usually 2 weeks)
- → Process streams of records as they occur.

SDU

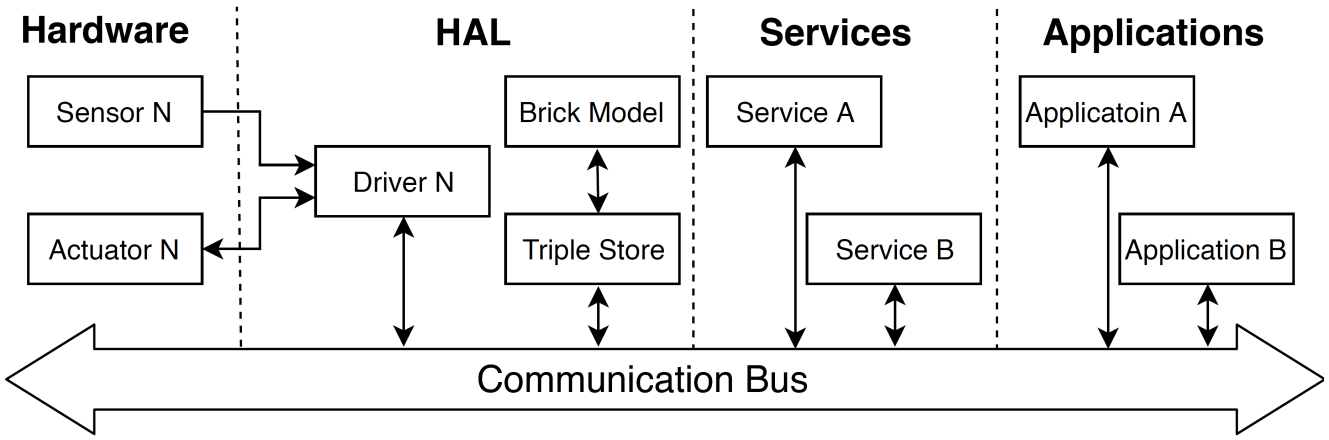
Wide language support

- → Language agnostic (TCP interface)
- → Libraries already written in:
  - → Go
  - → .Net (C#)
  - → Java
  - → Python
  - → C / C++
  - → NodeJS
  - → And many more …

Kafka in the Hadoop context

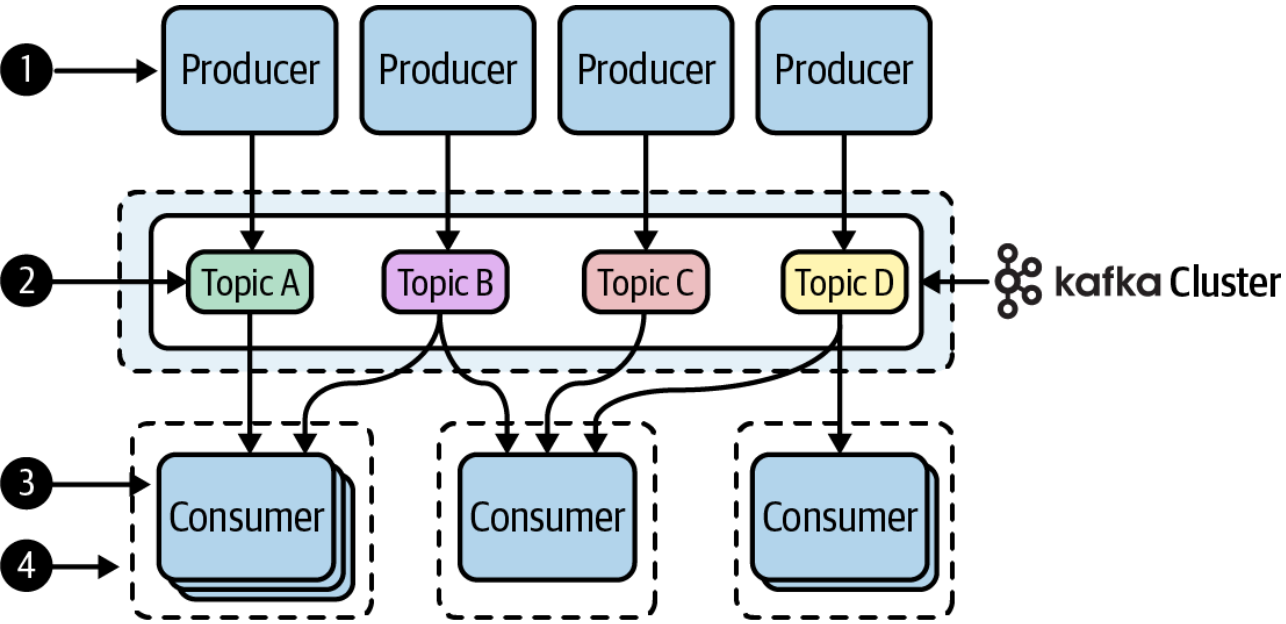


The Publish / Subscribe Architecture



- → Building Operating System Example.
  - → OU44 has over 40.000 datapoints.
  - → Can scale to campus or city
- → Publishes without knowing the receiver.
- → Subscribes waiting for data.

Kafka overview



→ 1. Producer, 2. Topic/data stream, 3. Consumer, 4. Consumer Group

→ Publishes without knowing the receiver.

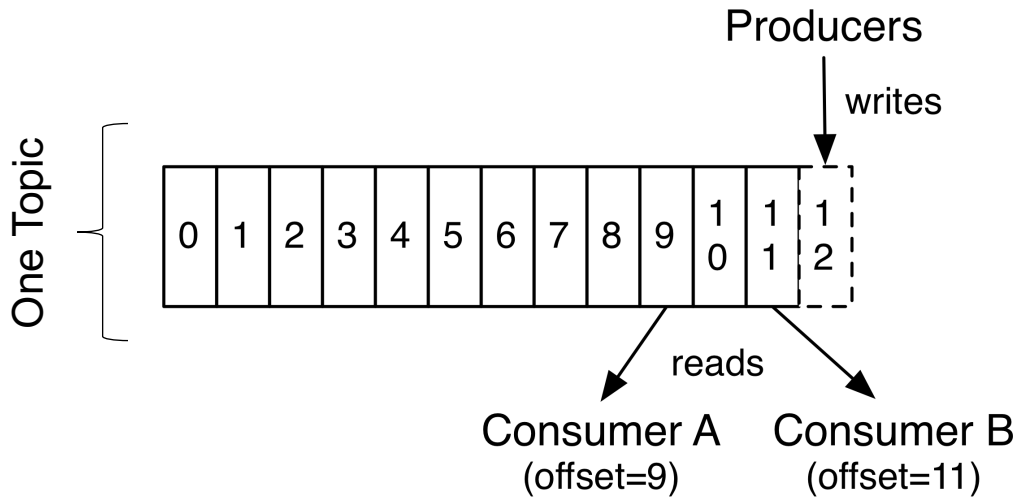
→ Subscribes waiting for data.

→ Decouples dependencies on specific applications.

→ Still requires standard formats for communication

→ Allows for 1-*, 1-1, and *-* relationships

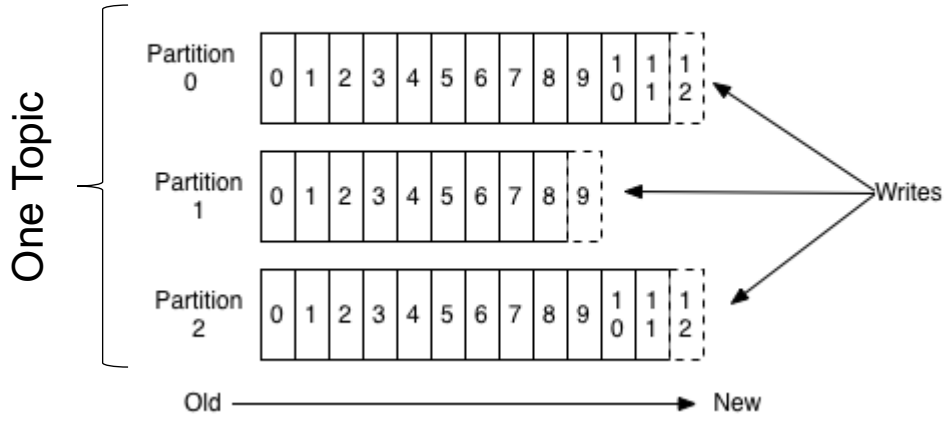
Producers and Consumers



- → Producers
  - → Write to a topic, **not** a partition.
  - → Partitions CAN be specified, but only for special usecases.
  - → Producers can create topics and define the replication factor and partitions
- → Consumers
  - → Reads from the beginning or a specific **offset**.
  - → Offset is saved in the Kafka broker.
  - → Belongs to a consumer group

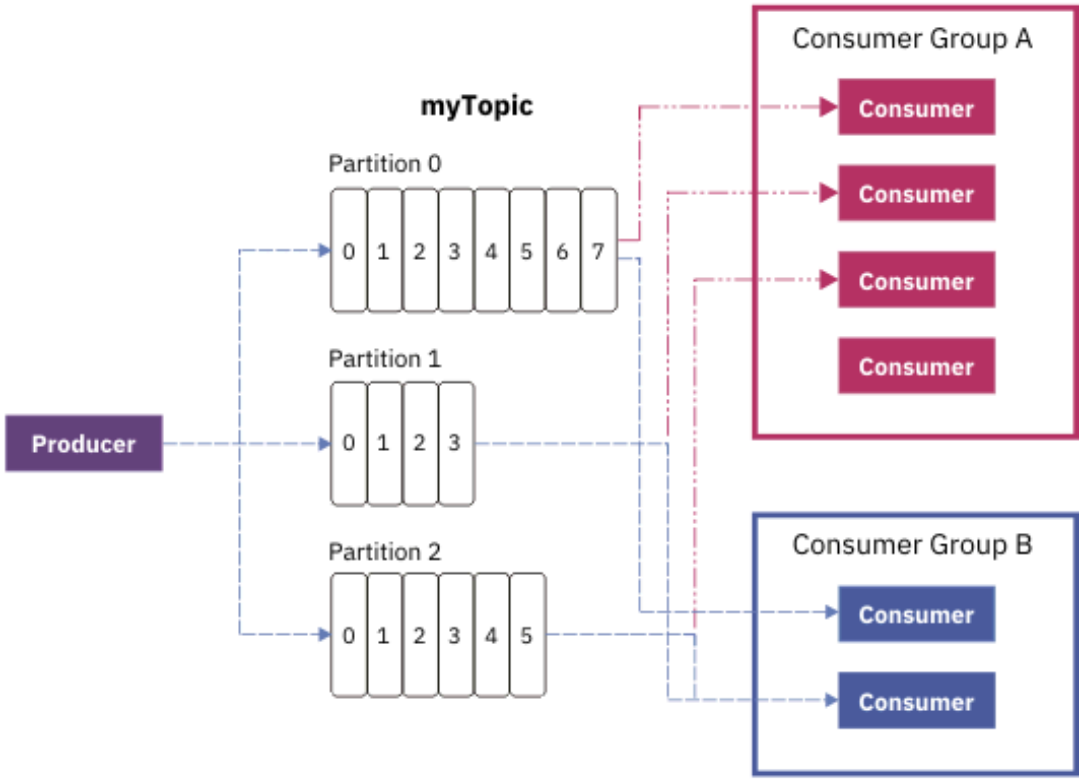
SDU

Topics and Partitions

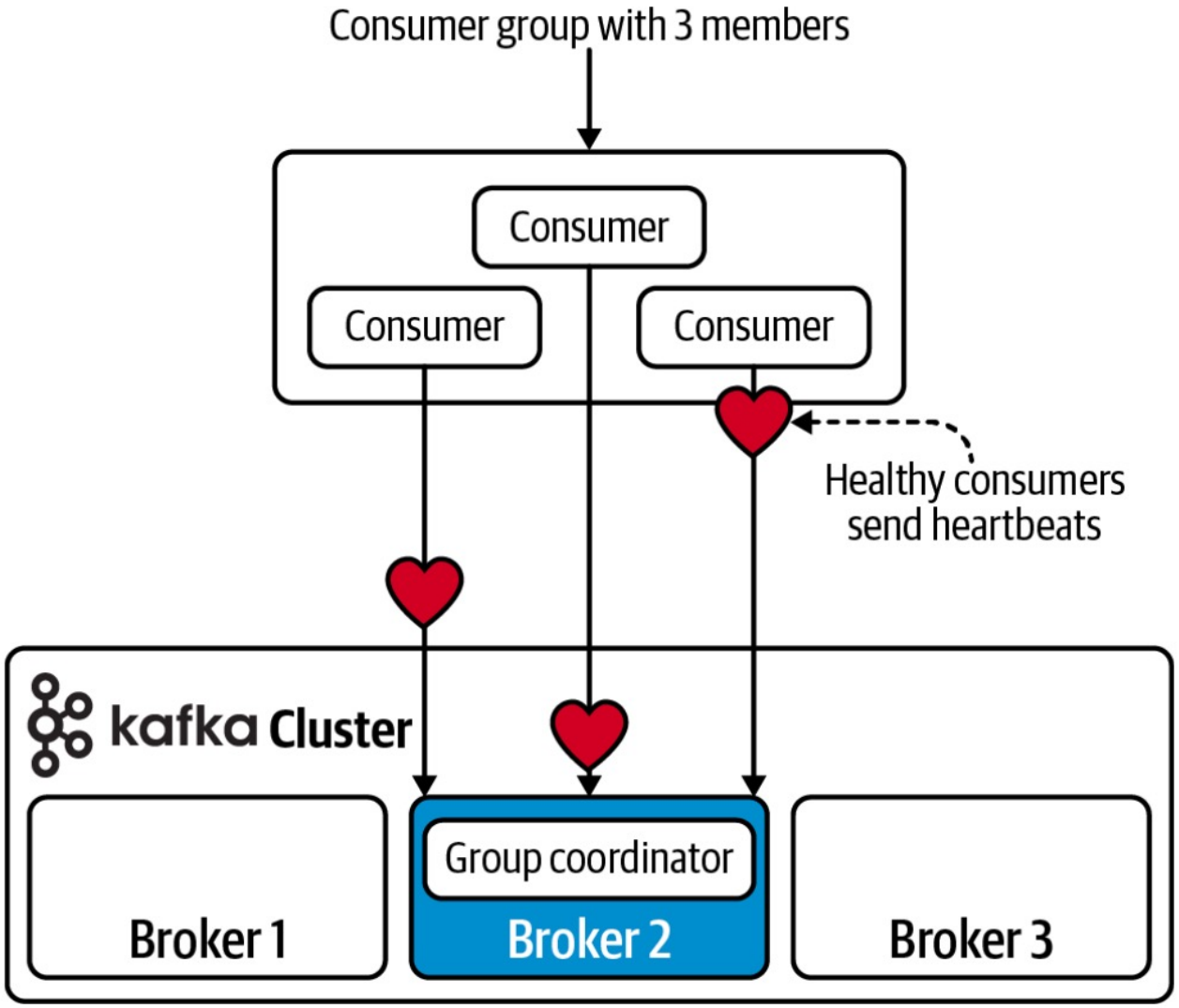


- → Topics: a particular stream of data
  - → Identified by a name
  - → Number of topics are not limited
  - → Has a partition size
  - → Has a replication factor (explained later)
- → Topics are split in partitions
  - → Each partition is ordered
  - → Each message within a partition gets an incremental id called an **offset**.

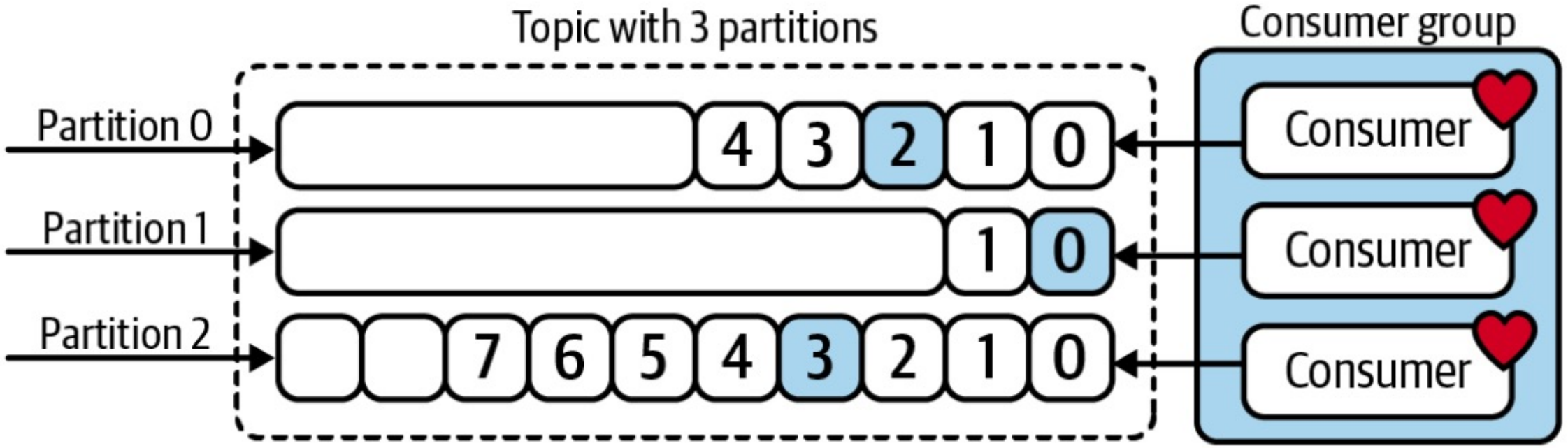
Consumer Groups



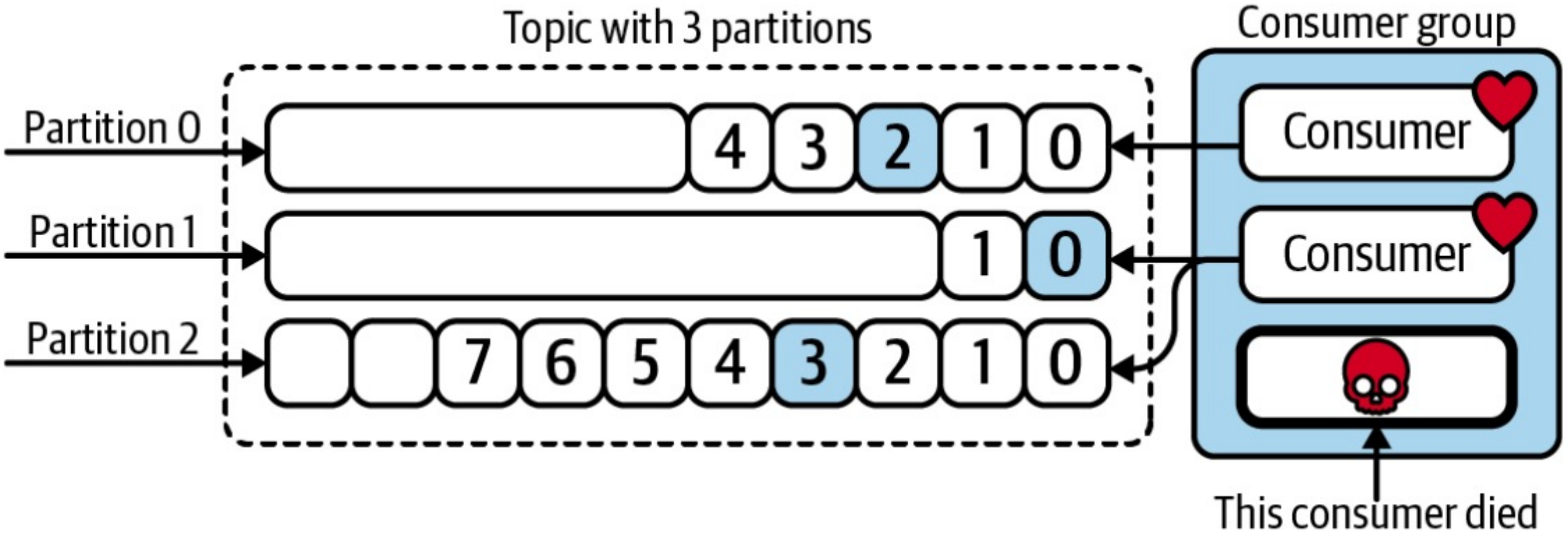
Consumer Groups

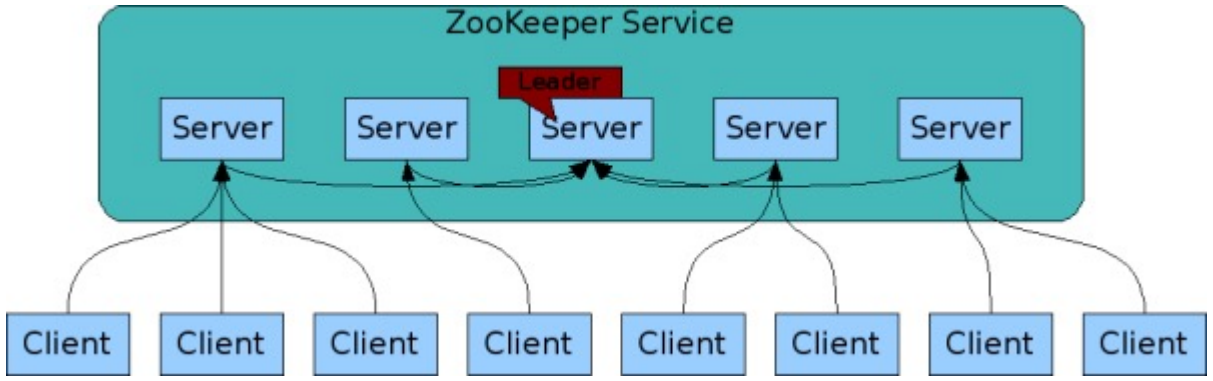


Consumer Groups

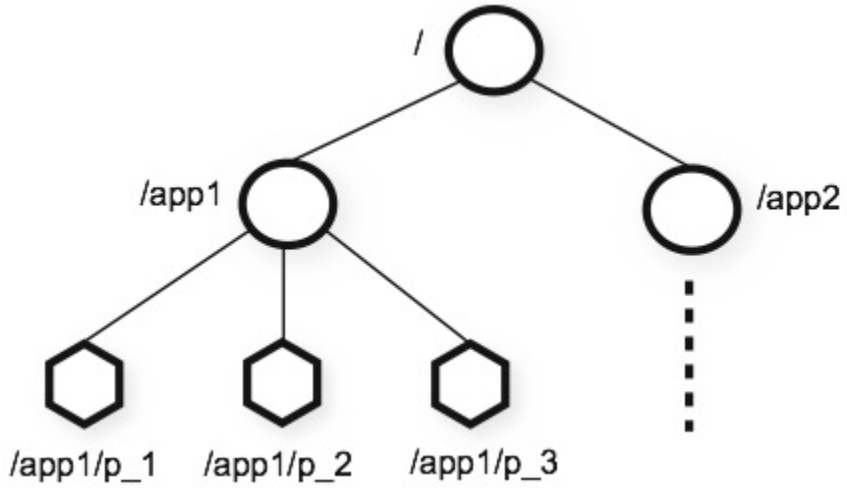


Consumer Groups



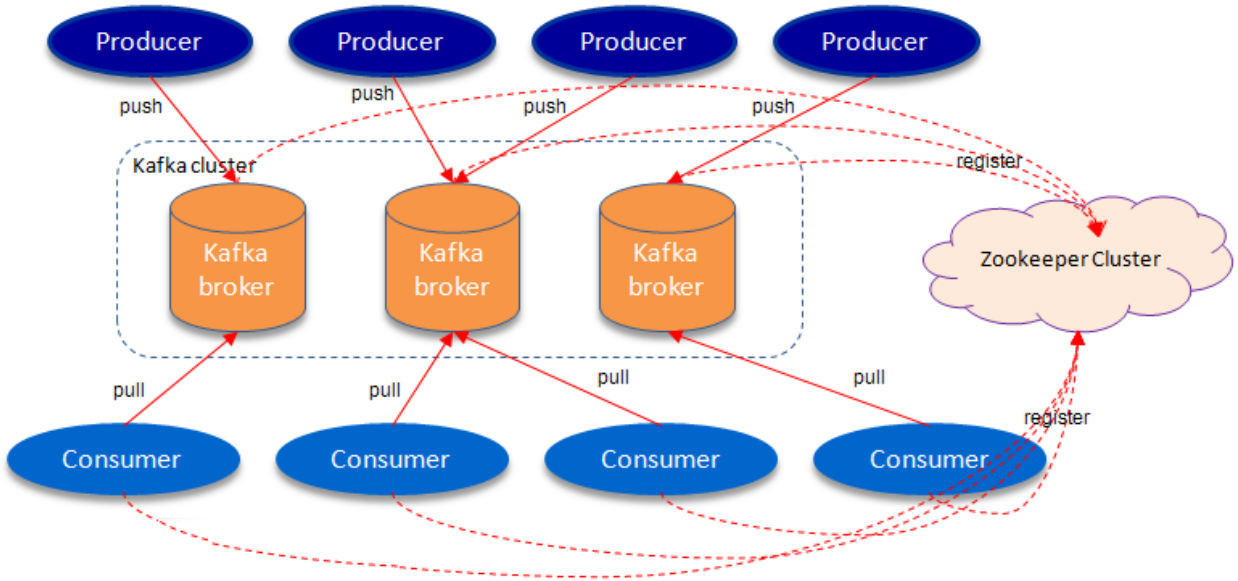


High Availability for Kafka with ZooKeeper



- → ZooKeeper: A Distributed Coordination Service for Distributed Applications
- → Zookeeper is used by:
  - → Apache Hadoop
  - → Apache Accumulo
  - → Apache HBase
  - → Apache Hive
  - → Apache Kafka
    - → (KIP-500 removes Zookeeper from 3.0, and beta available from 2.8)
  - → Apache Solr
  - → And more…

High Availability and Replication - Kafka



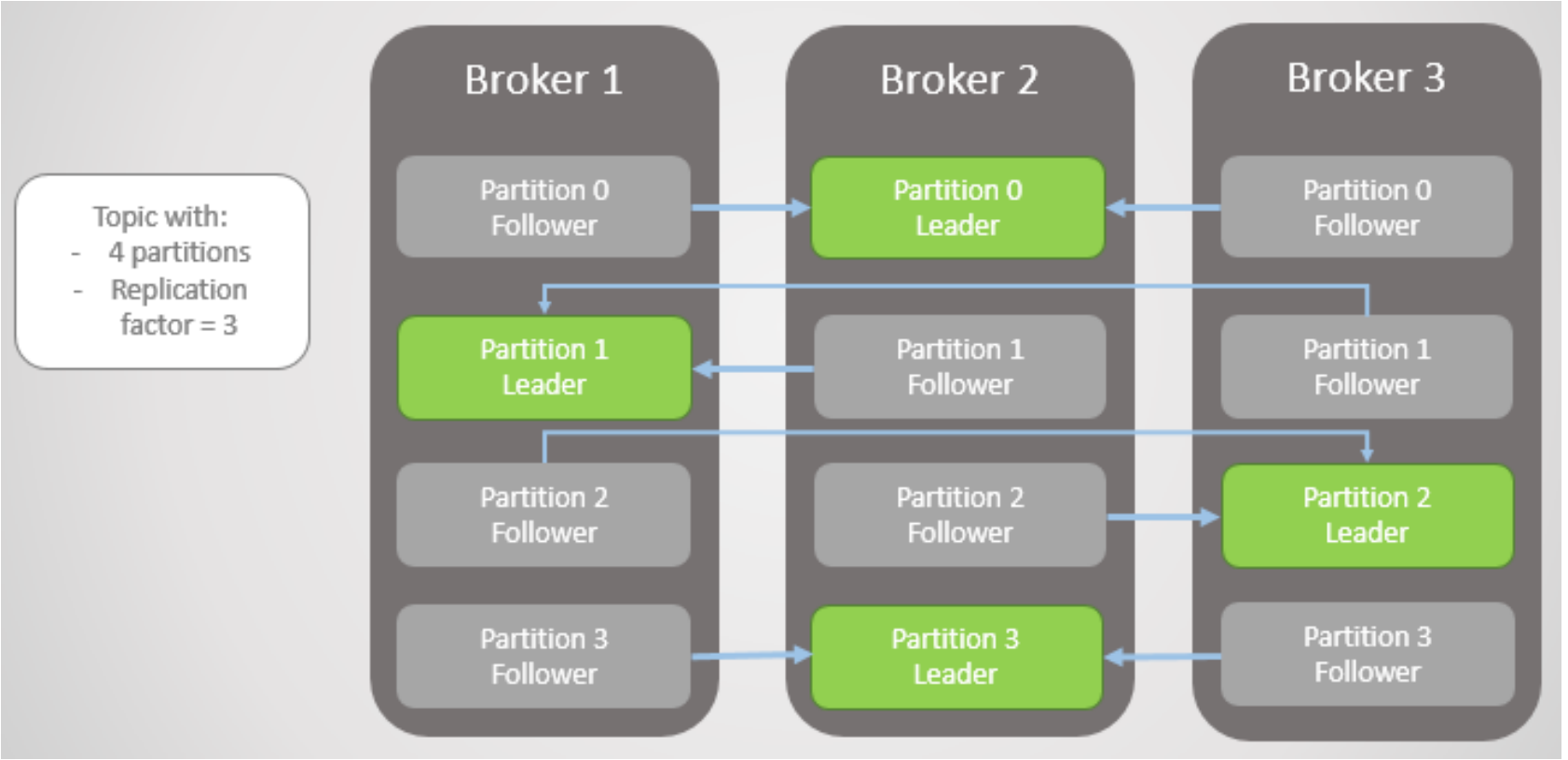
→ A cluster consists of several Kafka instances (Brokers).

→ Can distribute reads and writes to the message bus.

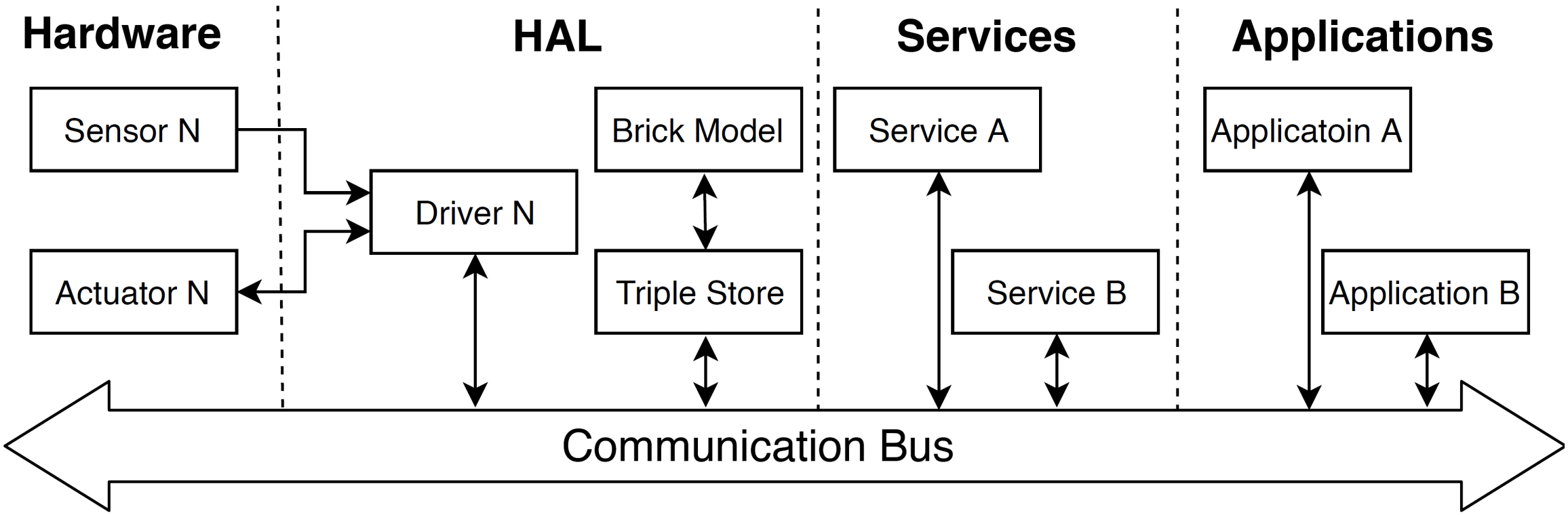
→ All orchestrated by ZooKeeper.

SDU

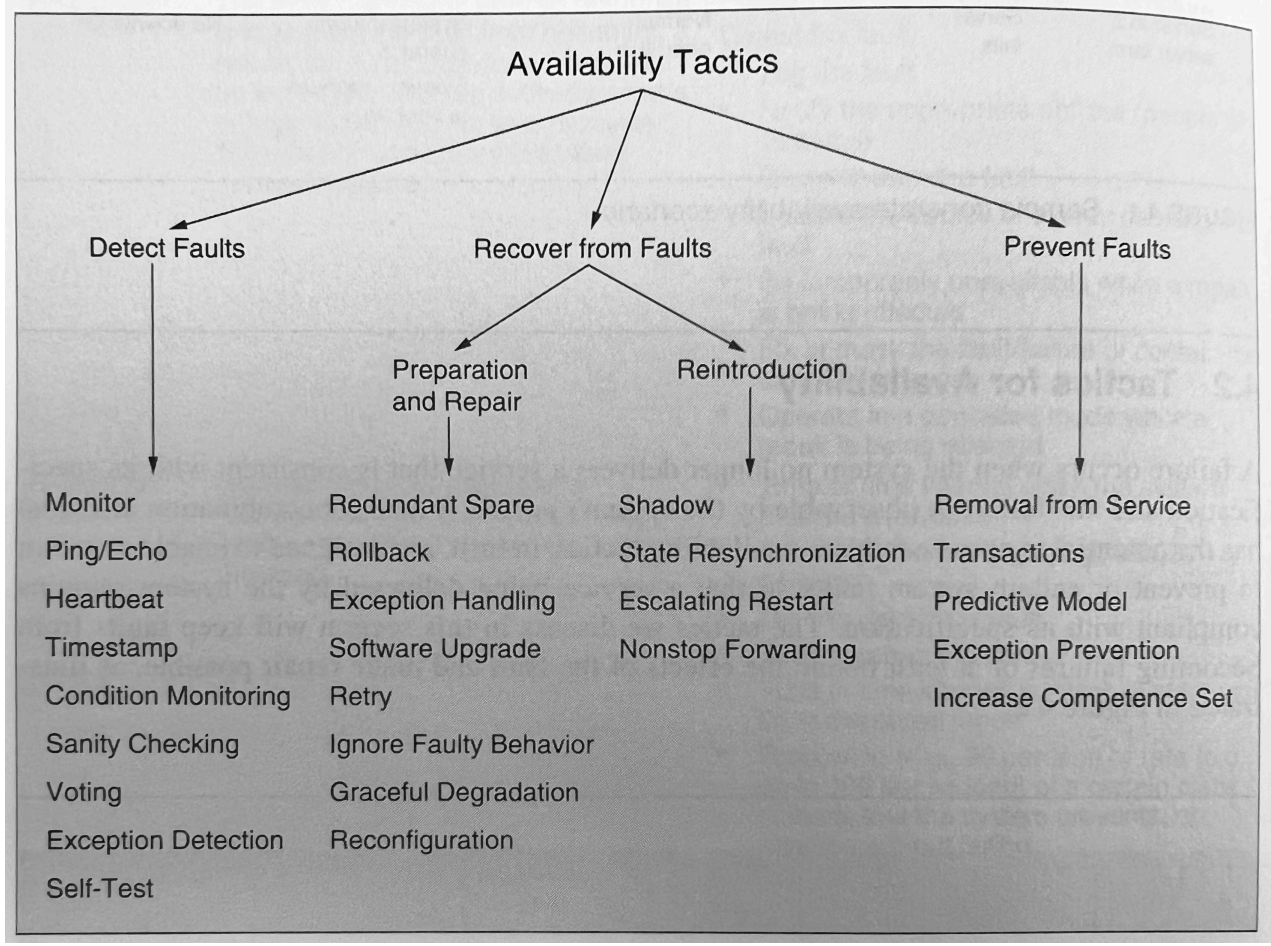
Load Distribution and Replication Factor



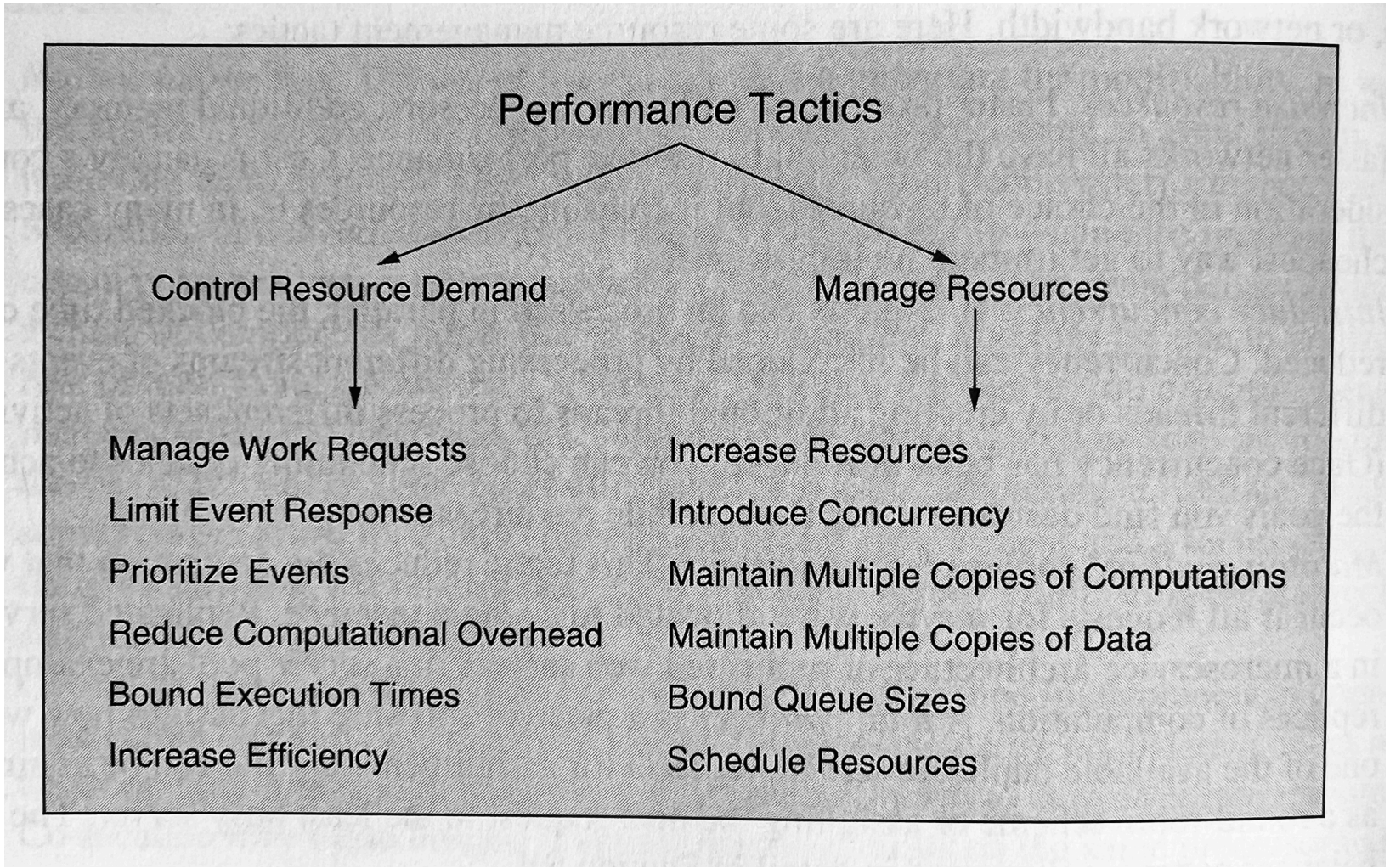
Another example



Availability Tactics



Performance



ksqlDB

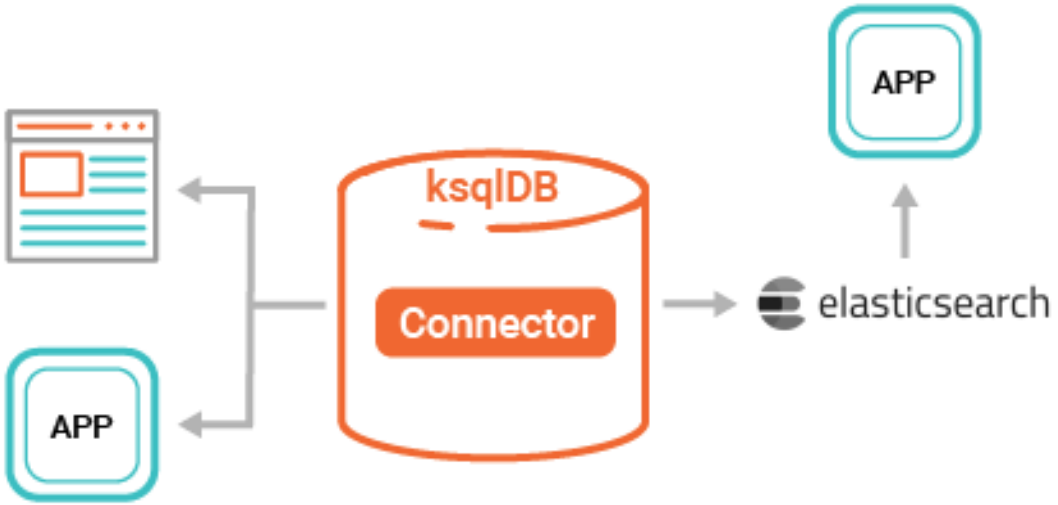
```
CREATE TABLE rated_movies AS
    SELECT title,
           SUM(rating)/COUNT(rating) AS avg_rating
      FROM ratings
      INNER JOIN movies
          ON ratings.movie_id=movies.movie_id
      GROUP BY title EMIT CHANGES;
```

Specialized database optimized for stream processing

```
CREATE STREAM widgets_red AS
SELECT * FROM widgets
WHERE colour='RED';
```

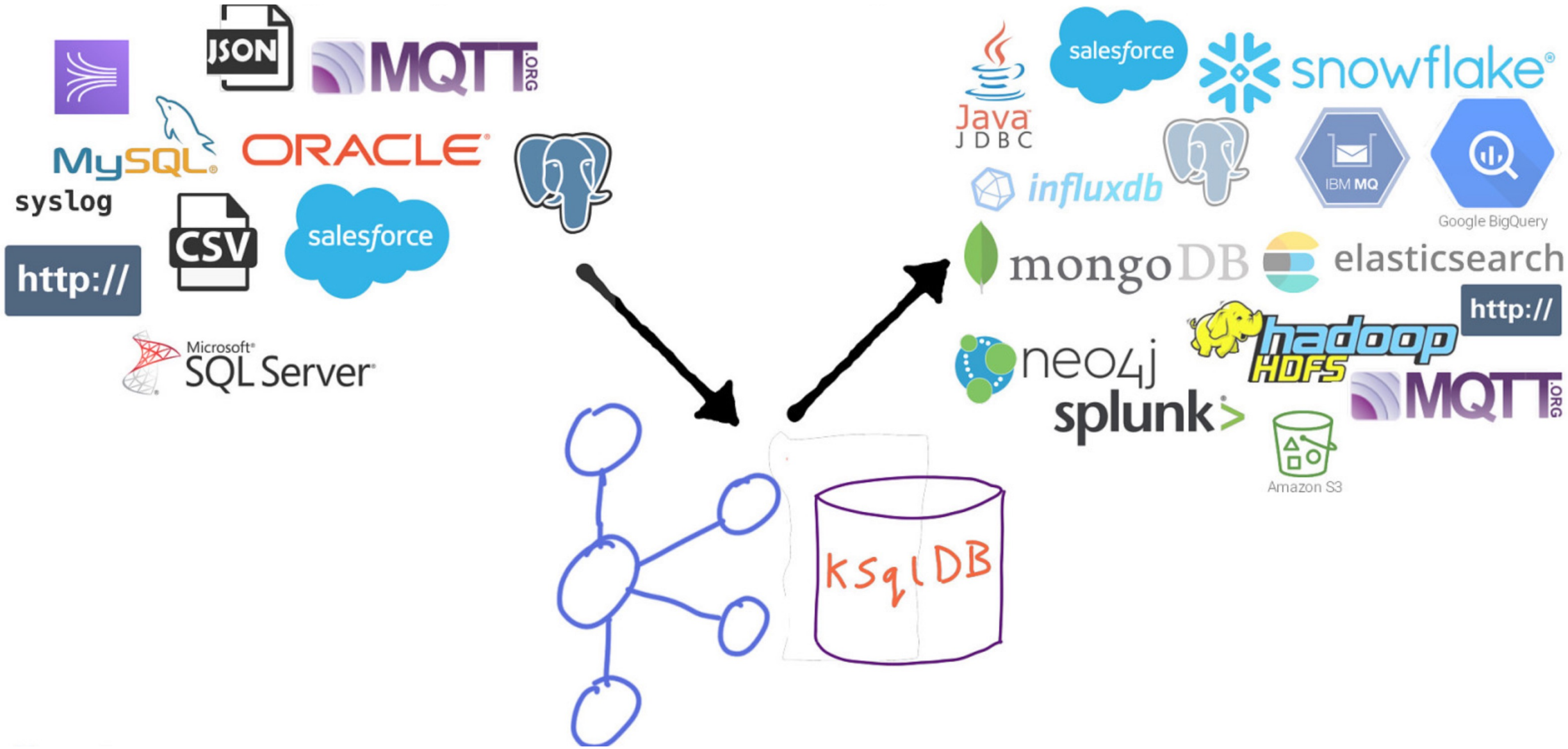
```
CREATE SINK CONNECTOR dw WITH (
'connector.class' = 'S3Connector',
'topics' = 'widgets'
…);
```

- → Can run its own fault tolerant cluster
- → Exposes streams through a SQL like language called kSQL
- → Continuous processing of event streams to expose table like data via REST

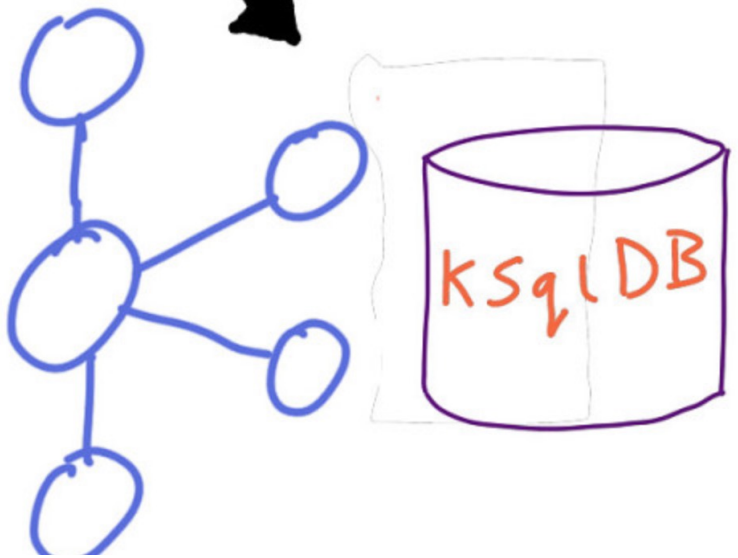



```
CREATE SINK CONNECTOR elasticConnector WITH (
  'connector.class' = '...ElasticsearchSinkConnector',
  'topics' = 'VEHICLES',
  'connection.url' = 'http://localhost:9200',
  'type.name' = 'kafka-connect',
  ...
);
```

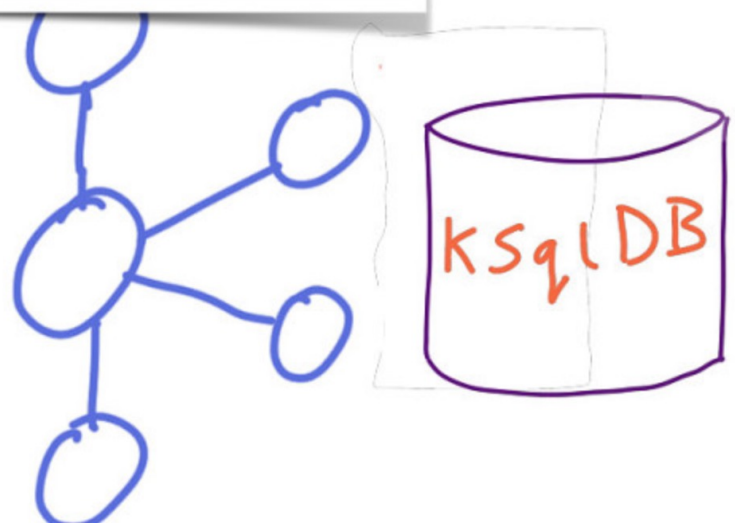

DB
DB
APP
APP
ksqlDB
Stream Processing
Connectors
Materialized Views
PULL
PUSH
APP


JSON
MQTT.ORG
MySQL
ORACLE
syslog
CSV
salesforce
http://
Microsoft SQL Server
Java JDBC
salesforce
snowflake
influxdb
IBM MQ
Google BigQuery
mongoDB
elasticsearch
http://
hadoop HDFS
neo4j
splunk>
MQTT.ORG
Amazon S3
ksqlDB


```
CREATE SOURCE CONNECTOR
SOURCE_MYSQL_01 WITH (
 'connector.class' =
'i.d.c.mysql.MySqlConnector',
 'database.hostname' = 'mysql',
 'table.whitelist' = 'demo.customers');
```



```
CREATE SINK CONNECTOR
SINK_ELASTIC_01 WITH (
 'connector.class' =
'…ElasticsearchSinkConnector',
 'connection.url' =
'http://elasticsearch:9200',
 'topics' = 'orders');
```



Kafka topic (k/v bytes)

```
{ "event_ts": "2020-02-17T15:22:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

```
{ "event_ts": "2020-02-17T17:23:00Z",
  "person"  : "robin",
  "location": "London"
}
```

```
{ "event_ts": "2020-02-17T22:23:00Z",
  "person"  : "robin",
  "location": "Wakefield"
}
```

```
{ "event_ts": "2020-02-18T09:00:00Z",
  "person"  : "robin",
  "location": "Leeds"
}
```

ksqlDB Stream

```
+---------------------+------+---------+
|EVENT_TS             |PERSON|LOCATION |
+---------------------+------+---------+
|2020-02-17 15:22:00 |robin  |Leeds    |
|2020-02-17 17:23:00 |robin  |London   |
|2020-02-17 22:23:00 |robin  |Wakefield|
|2020-02-18 09:00:00 |robin  |Leeds    |
```

Stream: Topic + Schema

ksqlDB Table

```
+------+---------+
|PERSON |LOCATION |
+------+---------+
|robin  |Leeds    |
```

Table: **state** for given **key**

Topic + Schema

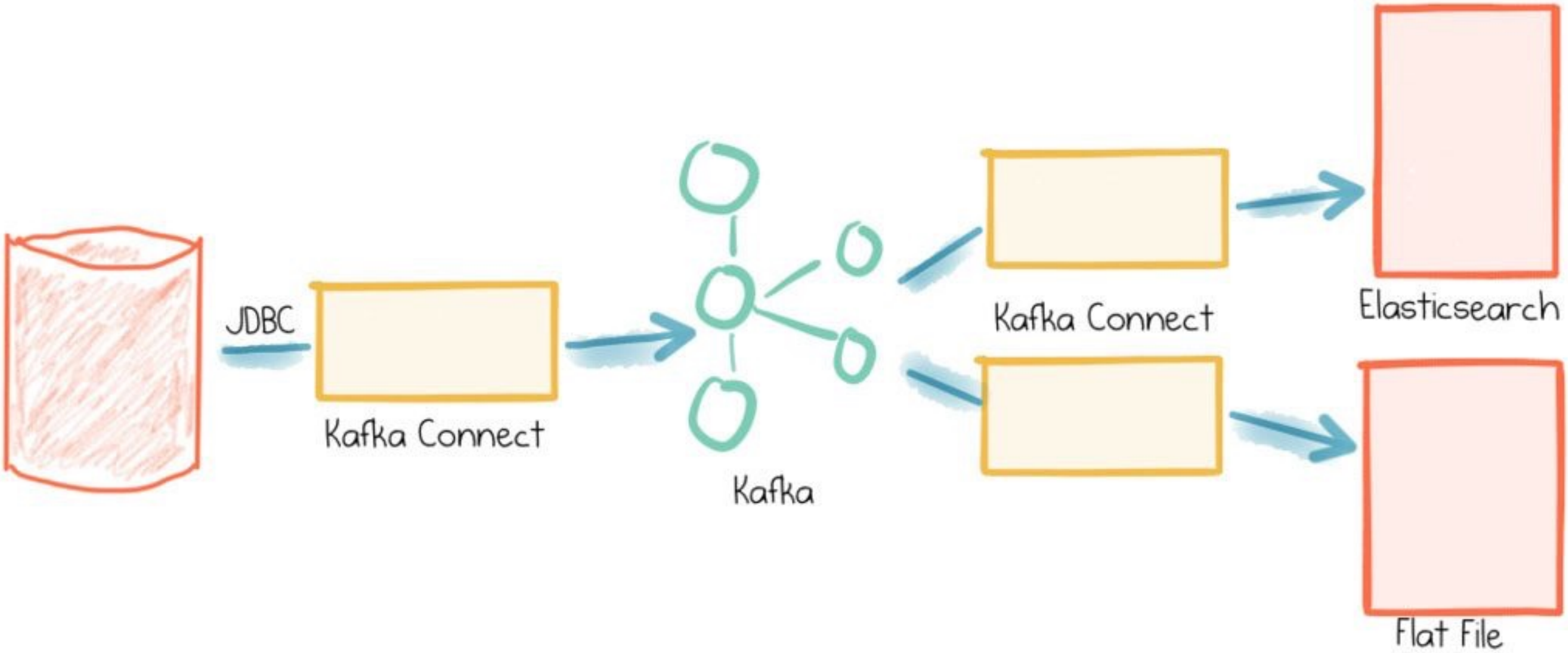
SDU

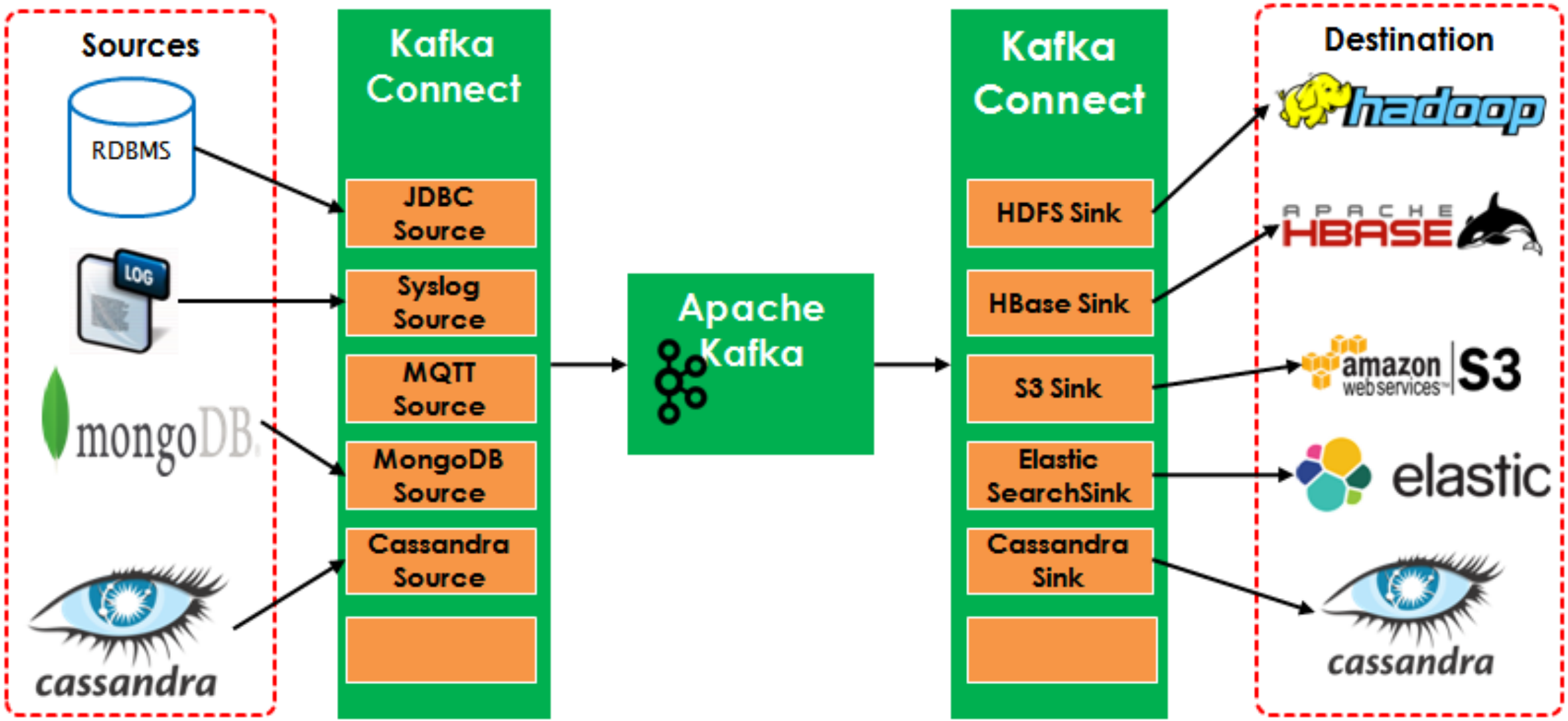
Kafka Connect

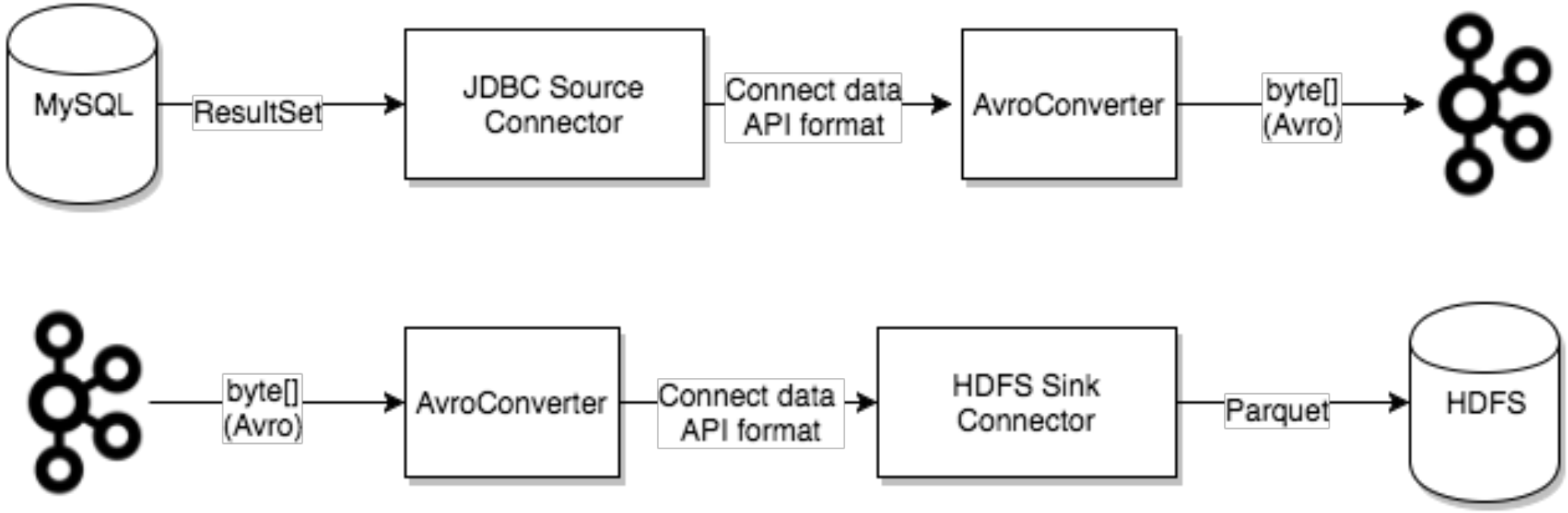
What is Kafka connect?

- → Consists of
  - → **Connectors** – the high level abstraction that coordinates data streaming by managing tasks
  - → **Tasks** – the implementation of how data is copied to or from kafka
  - → **Workers** – the running process that executes connectors and tasks
  - → **Converters** – the code used to translate data between connect and the system sending or receiving data
  - → **Transforms** – simple logic to alter each message produced by, or sent by a connector
  - → **Dead Letter Queue** – how Connect handles Connector Errors
- → Java based, requires manual work if outside of confluent platform

SDU


JDBC
Kafka Connect
Kafka
Kafka Connect
Elasticsearch
Flat File


Sources
RDBMS
LOG
mongoDB.
cassandra
Kafka Connect
JDBC Source
Syslog Source
MQTT Source
MongoDB Source
Cassandra Source
Apache Kafka
Kafka Connect
HDFS Sink
HBase Sink
S3 Sink
Elastic SearchSink
Cassandra Sink
Destination
hadoop
APACHE HBASE
amazon web services S3
elastic
cassandra


MySQL
ResultSet
JDBC Source Connector
Connect data API format
AvroConverter
byte[] (Avro)
byte[] (Avro)
AvroConverter
Connect data API format
HDFS Sink Connector
Parquet
HDFS

Exercises!