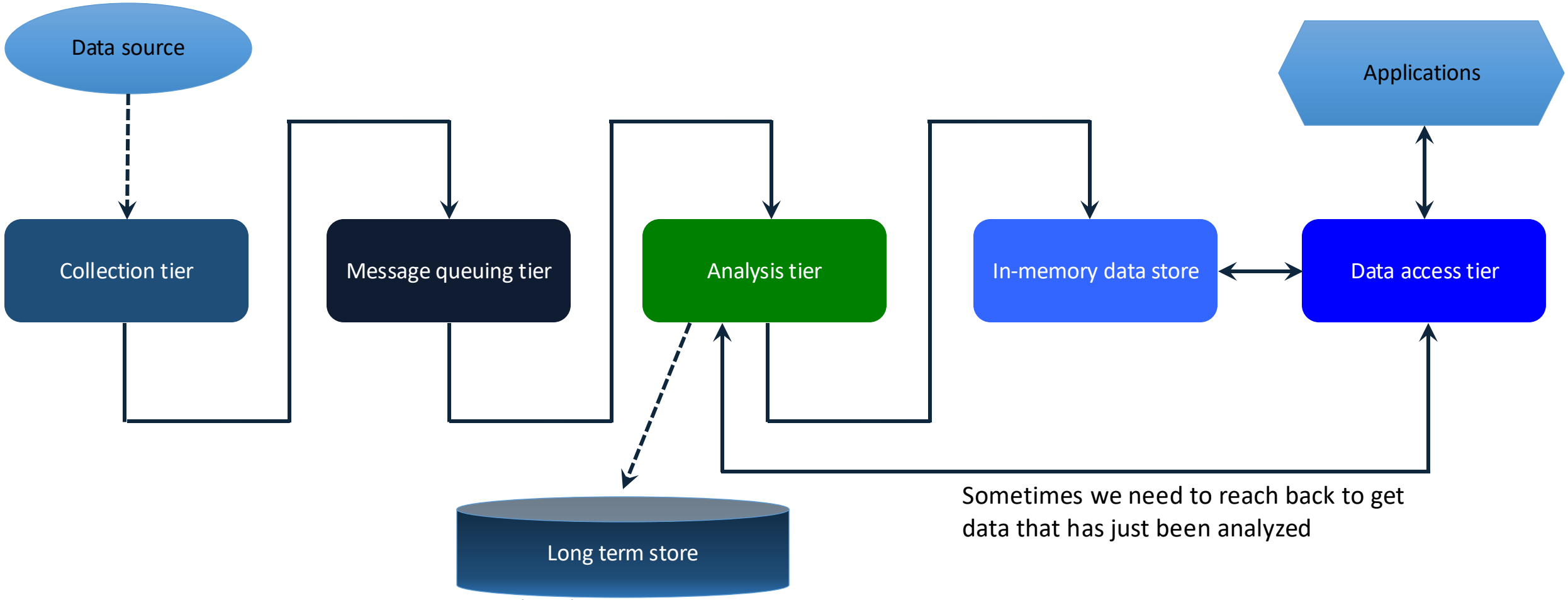
Data Streaming Tools

Thoai Nam

High Performance Computing Lab (HPC Lab)
Faculty of Computer Science and Engineering
HCMC University of Technology

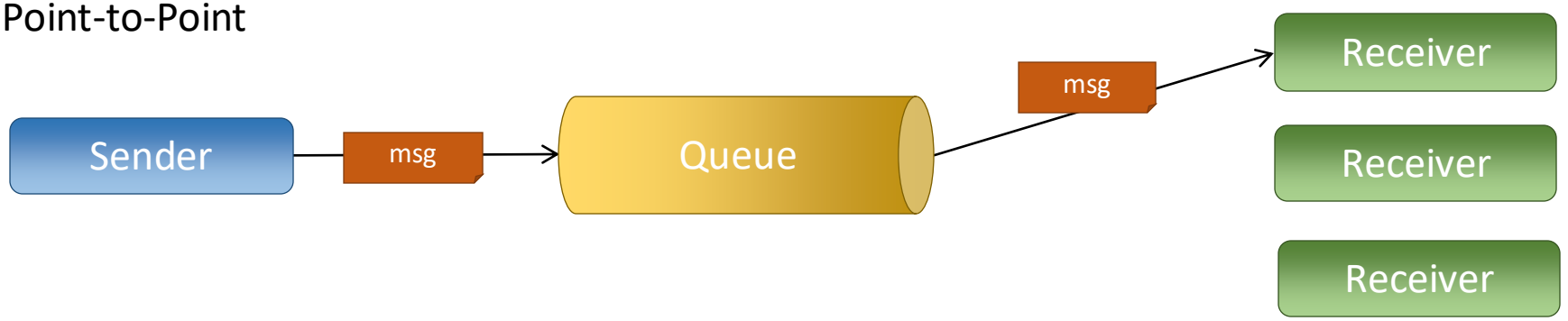
Data streaming architecture



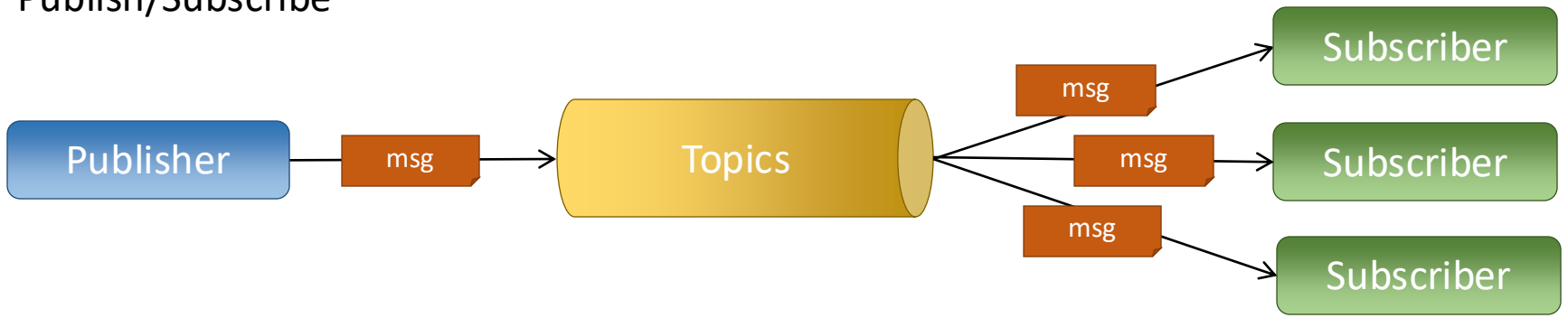
Kafka

Message models

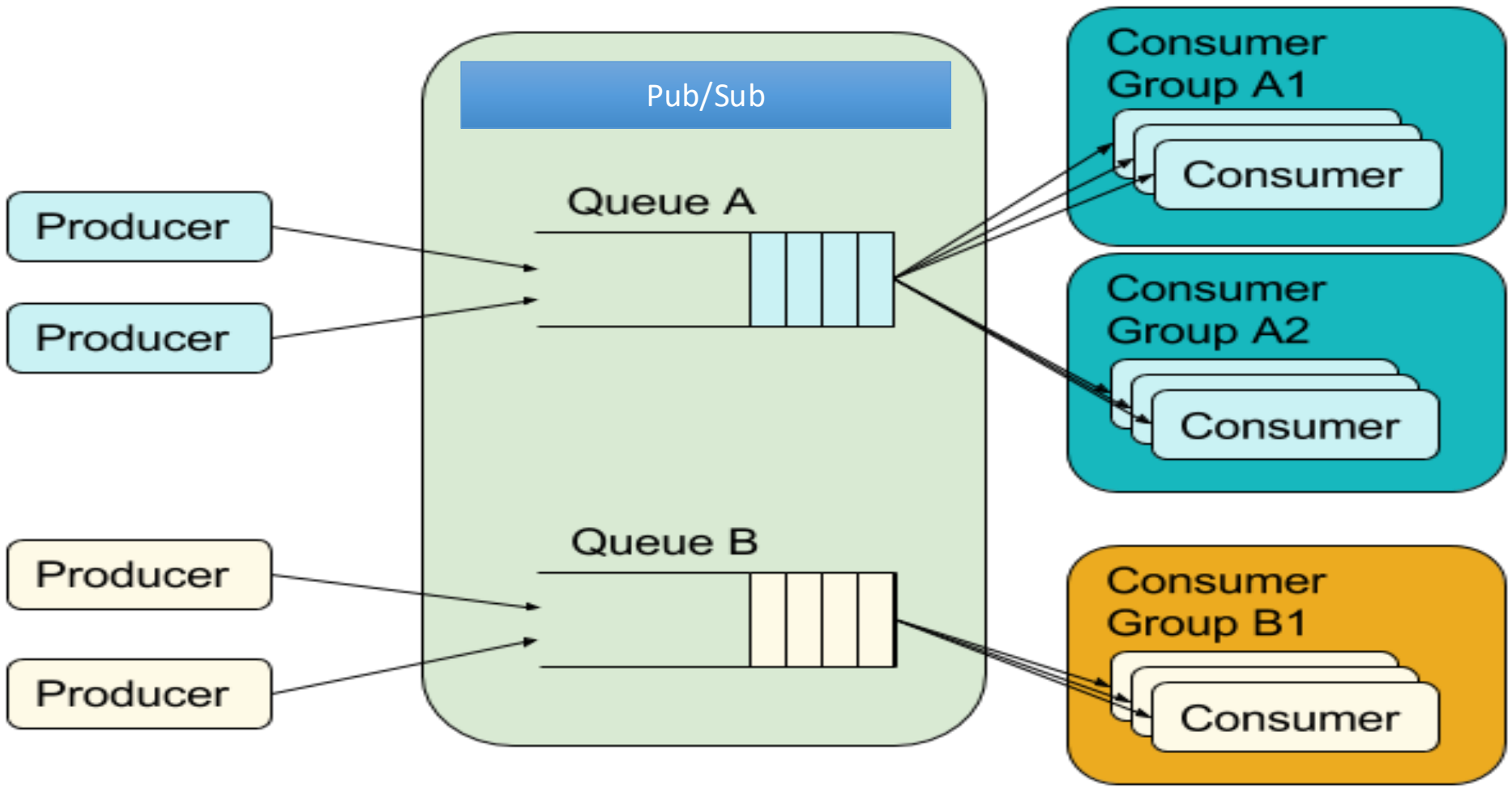
Point-to-Point



Publish/Subscribe



Publish/Subscribe model



Kafka

- Kafka is a “publish-subscribe messaging rethought as a distributed commit log”
- Fast
- Scalable
- Durable
- Distributed

Apache Kafka

Topic

Offset

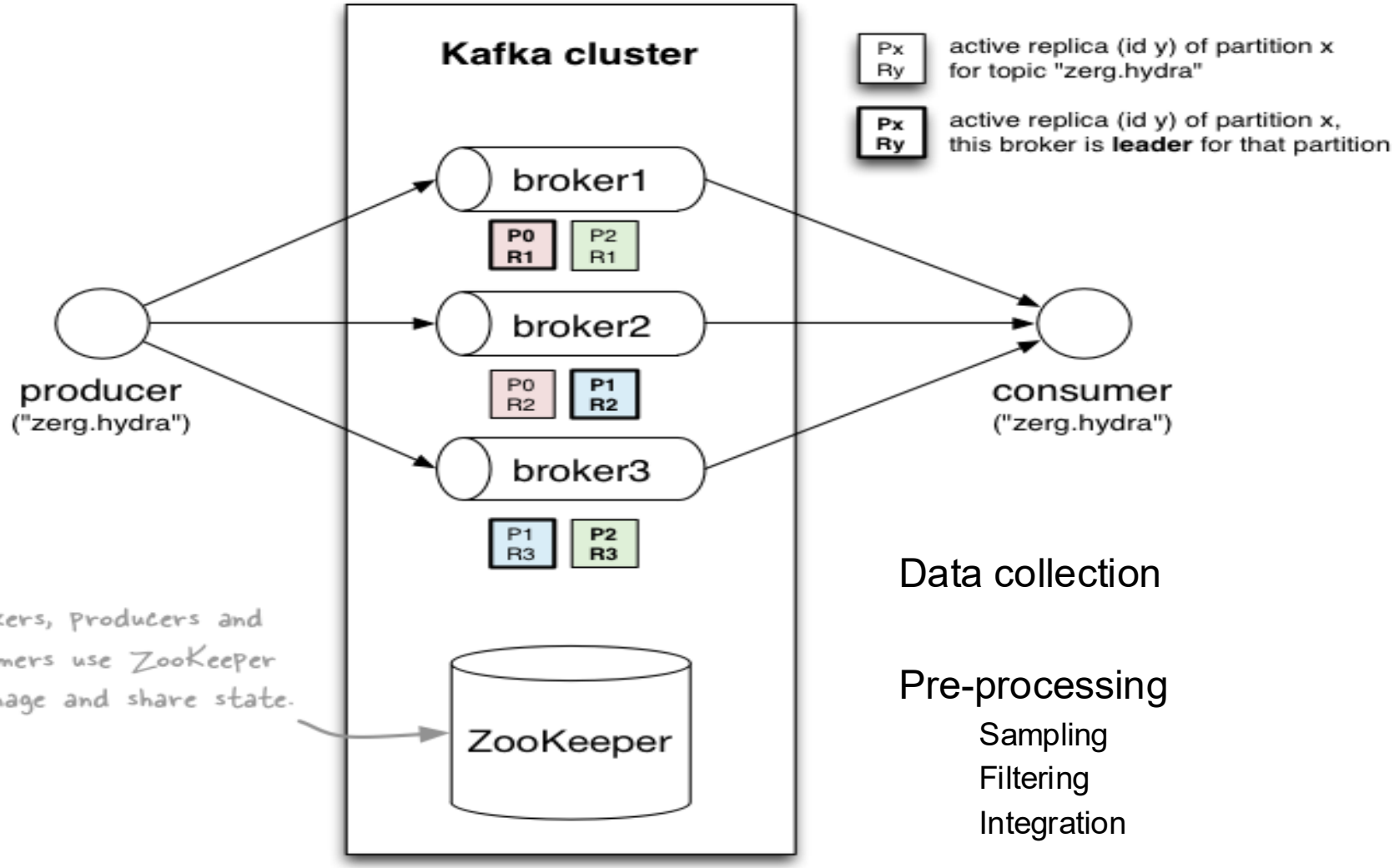
Broker

Partition

Replica

Kafka cluster

ZooKeeper



Data collection

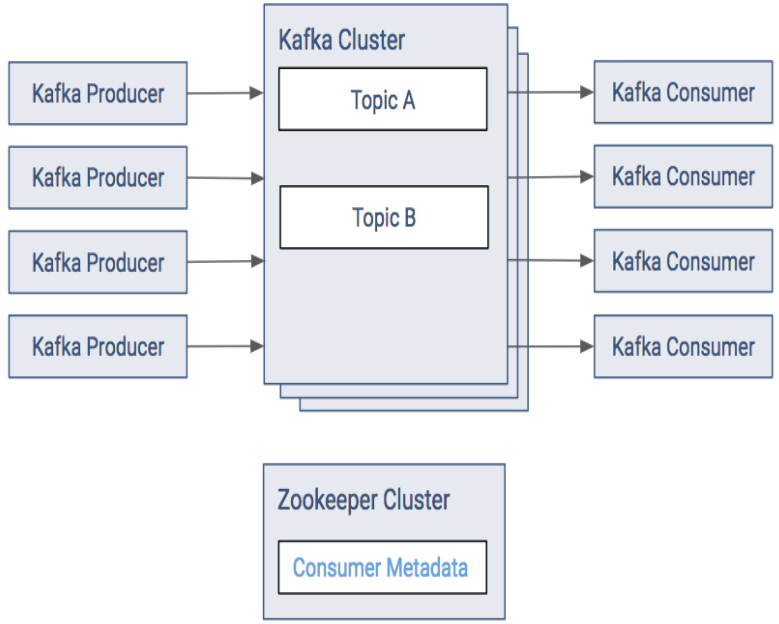
Pre-processing

- Sampling
- Filtering
- Integration

ZooKeeper

Zookeeper is a centralized service to maintain naming and configuration data, provide flexible and robust synchronization within distributed systems

- Zookeeper keeps track of status of the Kafka brokers, topics, partitions,... and notify all changes to Kafka
- **Kafka cannot run without Zookeeper!**



Semantics

There are 3 delivery semantics to commit the new offset:

Exactly one:

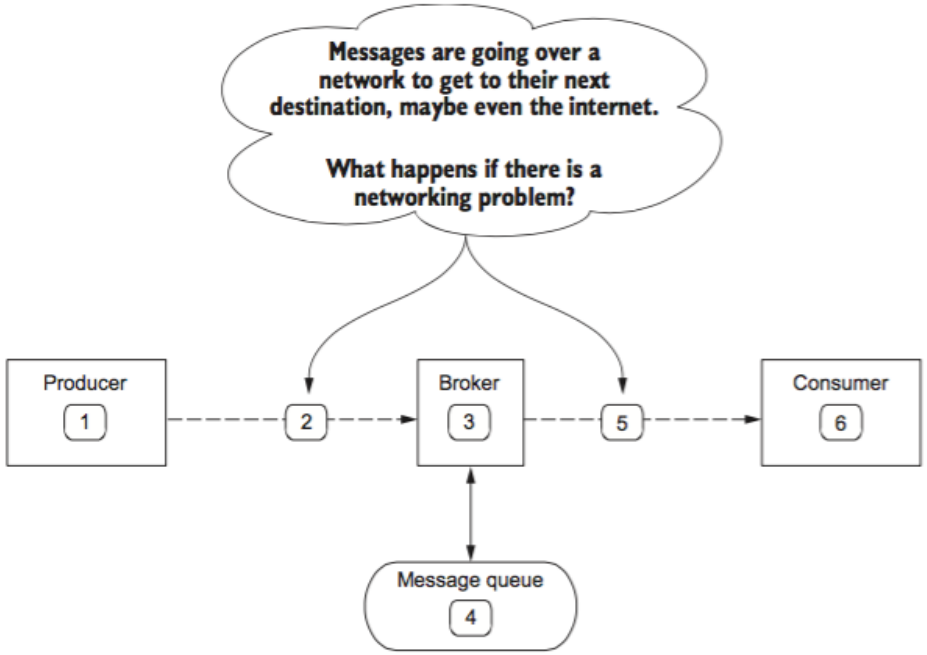
- Guarantees that all messages
  will always be delivered exactly once
- Only possible through the Streams API
  in Apache Kafka

At most once:

- Offsets are updated as soon as the message is retrieved
- Will lose current message if there are any exceptions while processing

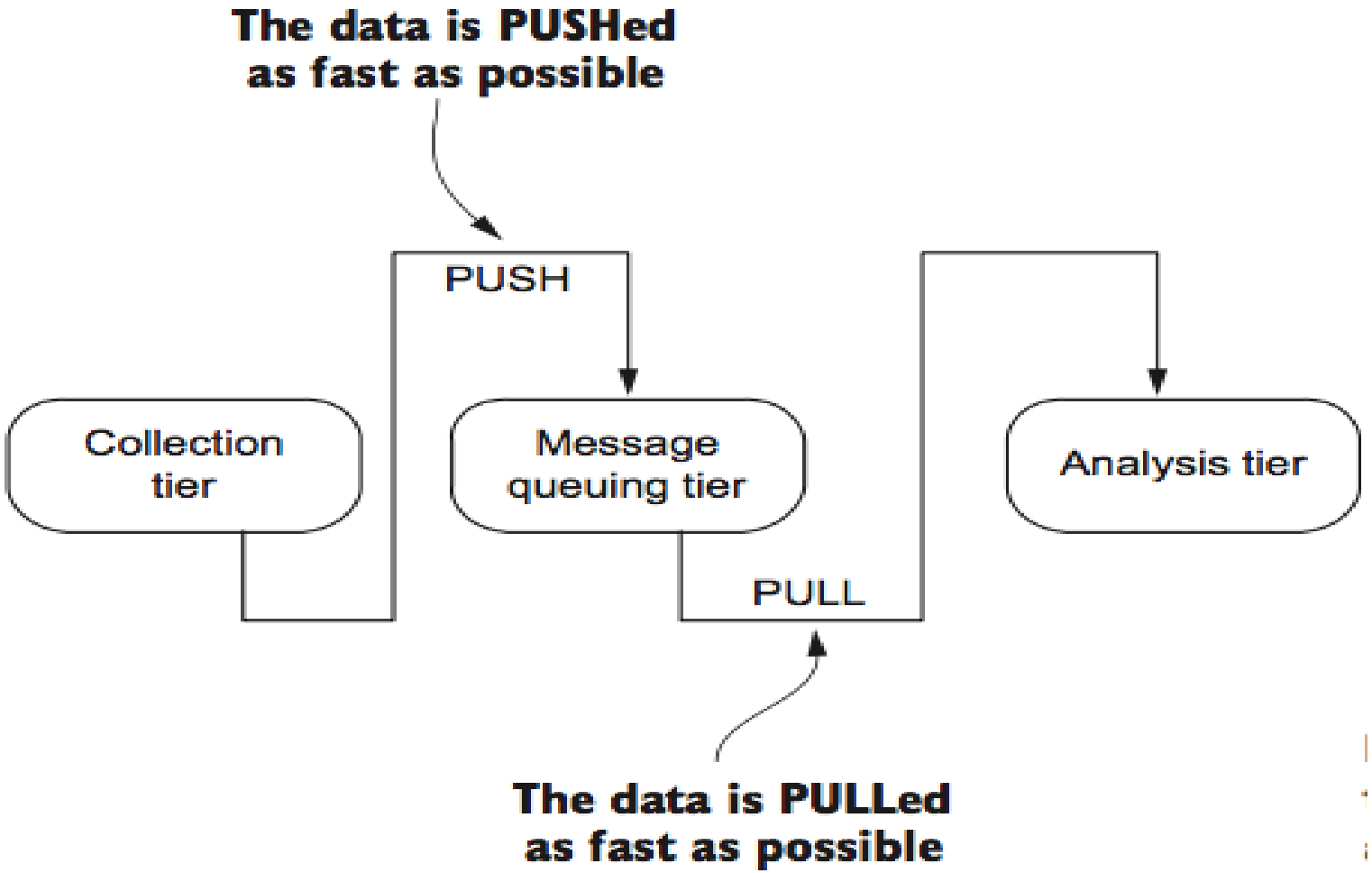
At least once (*preferred*):

- Offsets are updated after the message is processed successfully
- If there are any errors, the message will be read again until operation successfully -> can cause infinite loop
- A message can be processed twice or more -> Cause multiple effects
  (ex: duplicated notifications, emails)

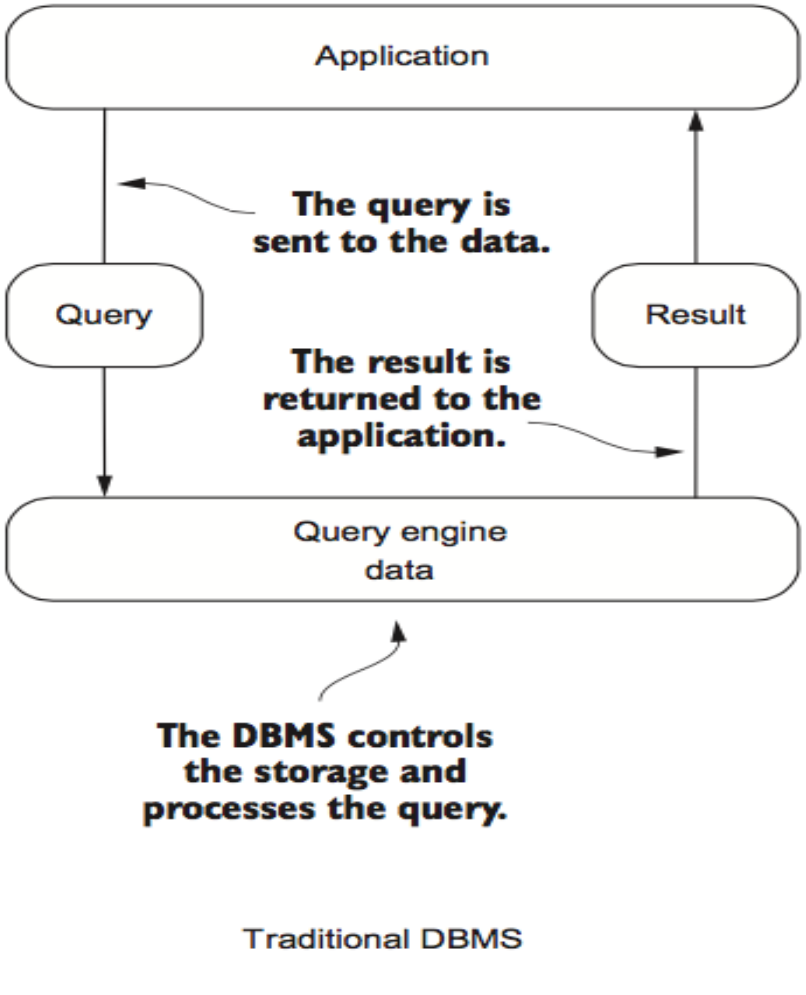


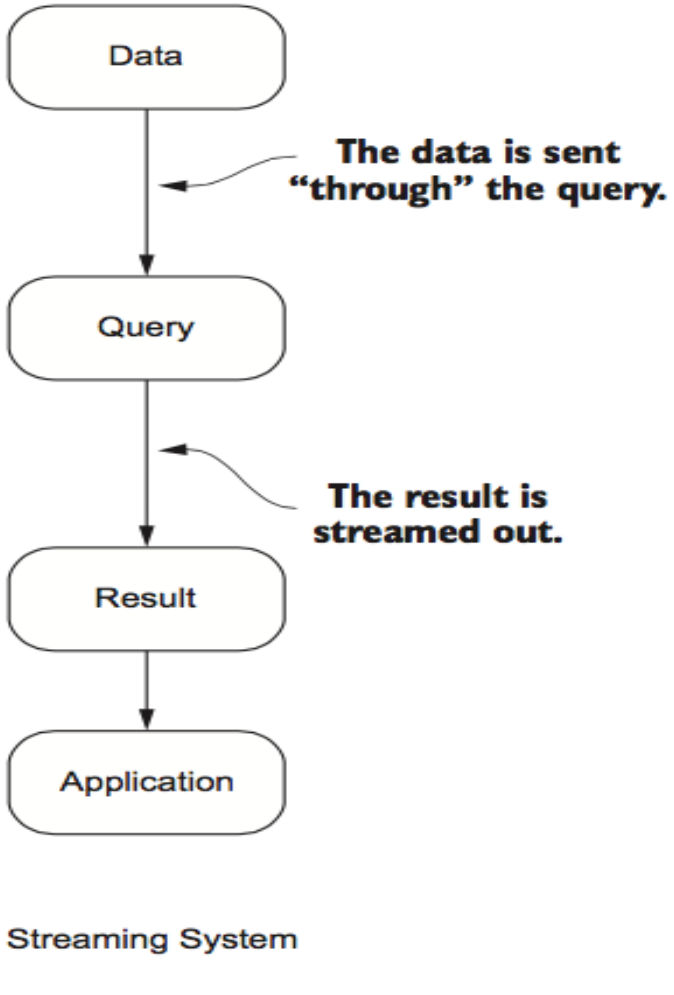
Analyzing streaming data

PUSH/PULL


The data is PUSHed
as fast as possible
PUSH
Collection
tier
Message
queuing tier
Analysis tier
PULL
The data is PULLed
as fast as possible

Non-streaming & streaming system





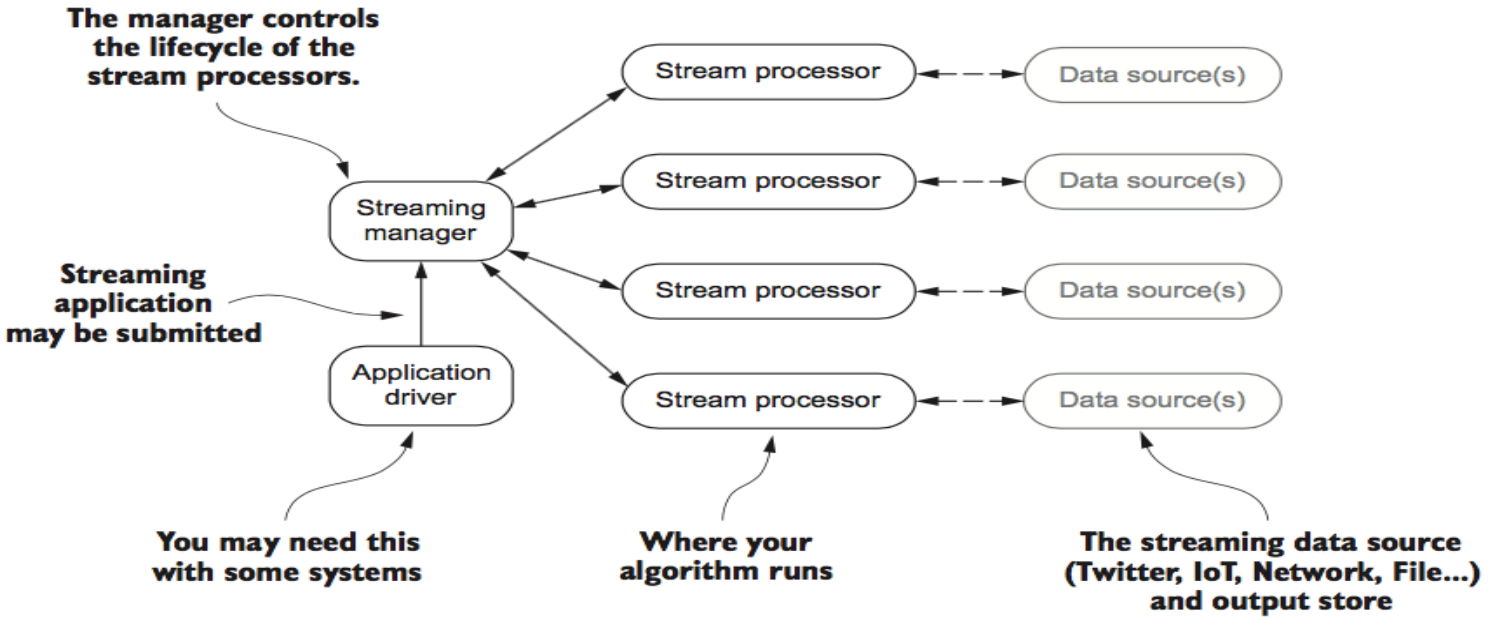
- A traditional DBMS (RDBMS, Hadoop, HBase, Cassandra, and so on):
  - In those non-streaming systems the data is at rest, and we query it for answers
- A streaming system:
  - *In-flight data:* The data is moved through the query
  - *Continuous query model*: the query is constantly being evaluated as new data arrive

Comparison of traditional DBMS to streaming system

	DBMS	Streaming system
Query model	Queries are based on a one-time model and a consistent state of the data. In a one-time model, the user executes a query and gets an answer, and the query is forgotten. This is a pull model.	The query is continuously executed based on the data that is flowing into the system. A user registers a query once, and the results are regularly pushed to the client.
Changing data	During down time, the data cannot change.	Many stream applications continue to generate data while the streaming analysis tier is down, possibly requiring a catch-up following a crash.
Query state	If the system crashes while a query is being executed, it is forgotten. It is the responsibility of the application (or user) to re-issue the query when the system comes back up.	Registered continuous queries may or may not need to continue where they left off. Many times it is as if they never stopped in the first place.

Distributed stream-processing architecture (1)

- Tools: (Apache) Spark Streaming, Storm, Flink, and Samza
  - A component that your streaming application is submitted to; this is similar to how Hadoop Map Reduce works. Your application is sent to a node in the cluster that executes your application
  - Separate nodes in the cluster execute your streaming algorithms
  - Data sources are the input to the streaming algorithms

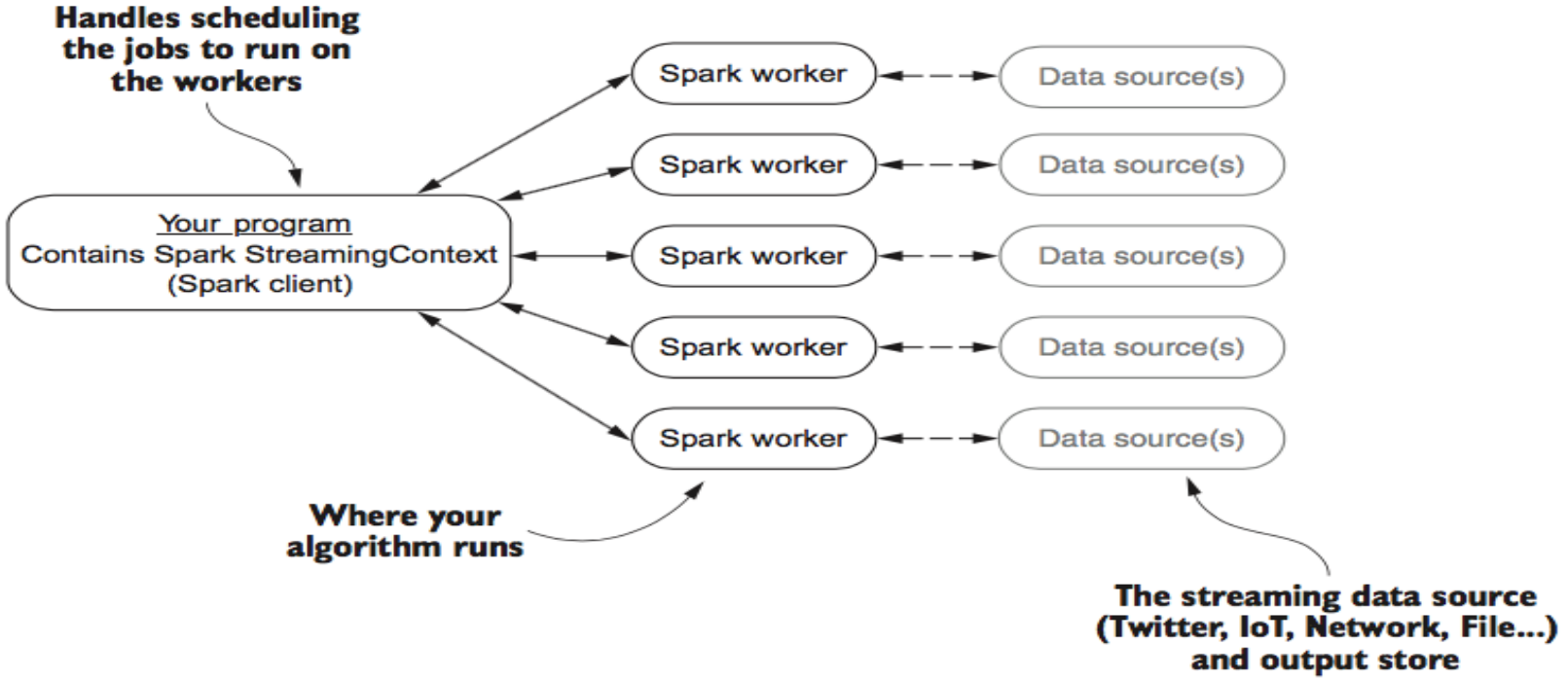


Distributed stream-processing architecture (2)

- *Application driver*: with some streaming systems, this will be the client code that defines your streaming programming and communicates with the streaming manager
- *Streaming manager*: the streaming manager has the general responsibility of getting your streaming job to the stream processor(s); in some cases it will control or request the resources required by the stream processors
- *Stream processor*: the place where your job runs; although this may take many shapes based on the streaming platform in use, the job remains the same: to execute the job that was submitted
- *Data source(s)*: This represents the input and potentially the output data from your streaming job. With some platforms your job may be able to ingest data from multiple sources in a single job, whereas others may only allow ingestion from a single source.

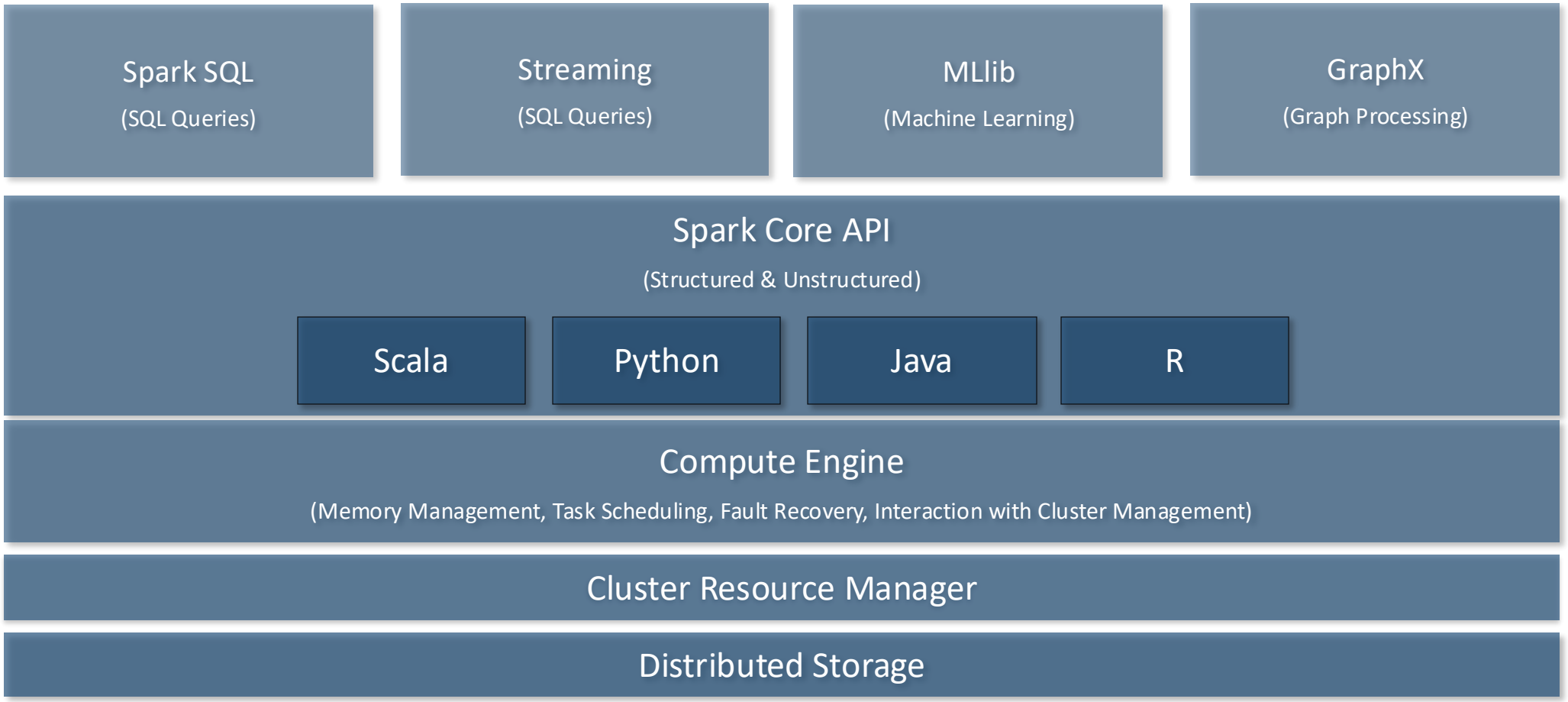
Apache Spark

- The de facto platform for general-purpose distributed computation
- Programming languages: Java, Scala, Python, and R
- Modules:
  - Spark Streaming
  - MLlib (machine learning)
  - SparkR (integration with R)
  - GraphX (for graph processing)



- Spark StreamingContext is the driver
- A *job* in Spark Streaming is the logic of your program that's bundled and passed to the Spark workers
- Spark workers: which run on any number of computers (from one to thousands) and are where your job (your streaming algorithm) is executed. They receive data from an external data source and communicate with the Spark StreamingContext that's running as part of the driver.

Apache Spark

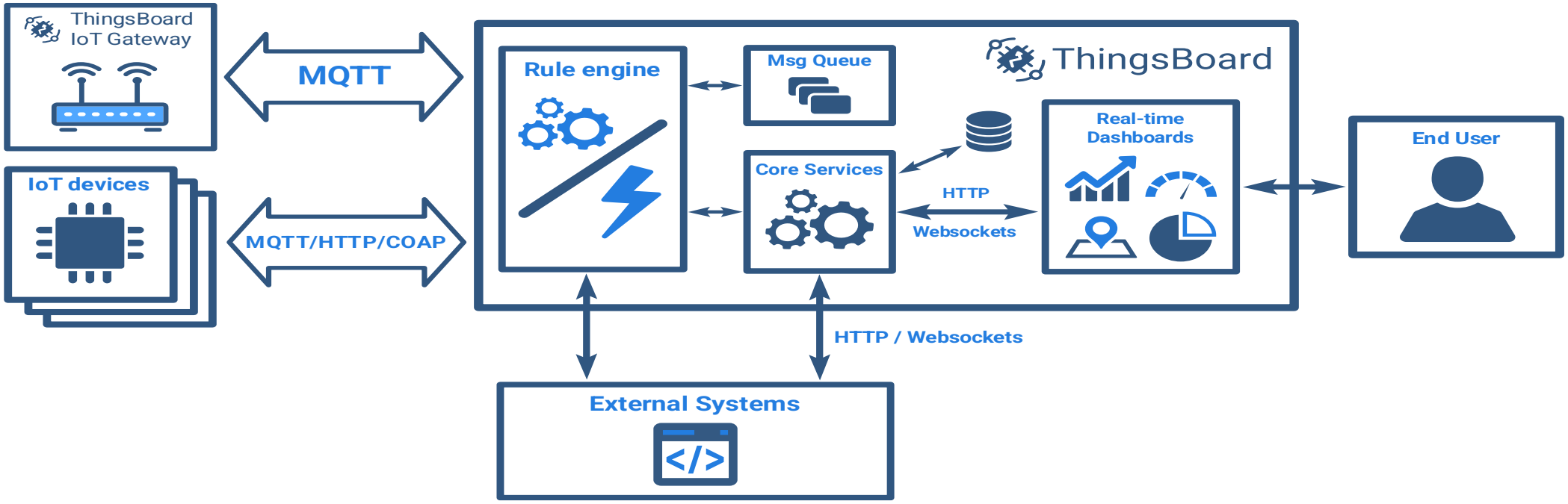


Data analytics

Message delivery semantics

- *At-most-once* - *a* message may get lost, but it will never be processed a second time => **`Simple`**
- *At-least-once* - *a* message will never be lost, but it may be processed more than once
  - If every time the streaming job receives the same message, it produces the same result => the duplicate-messages situation
- *Exactly-once* - *a* message is never lost and will be processed only once
  - Detect and ignore duplicates

ThingsBoard



- Collection data node
- Support many protocols
- Rule engine
  - Pre-processing: sampling, filtering, integration
- Message queue + etc.

=> A small scale solution (do not need Kafka + Spark)