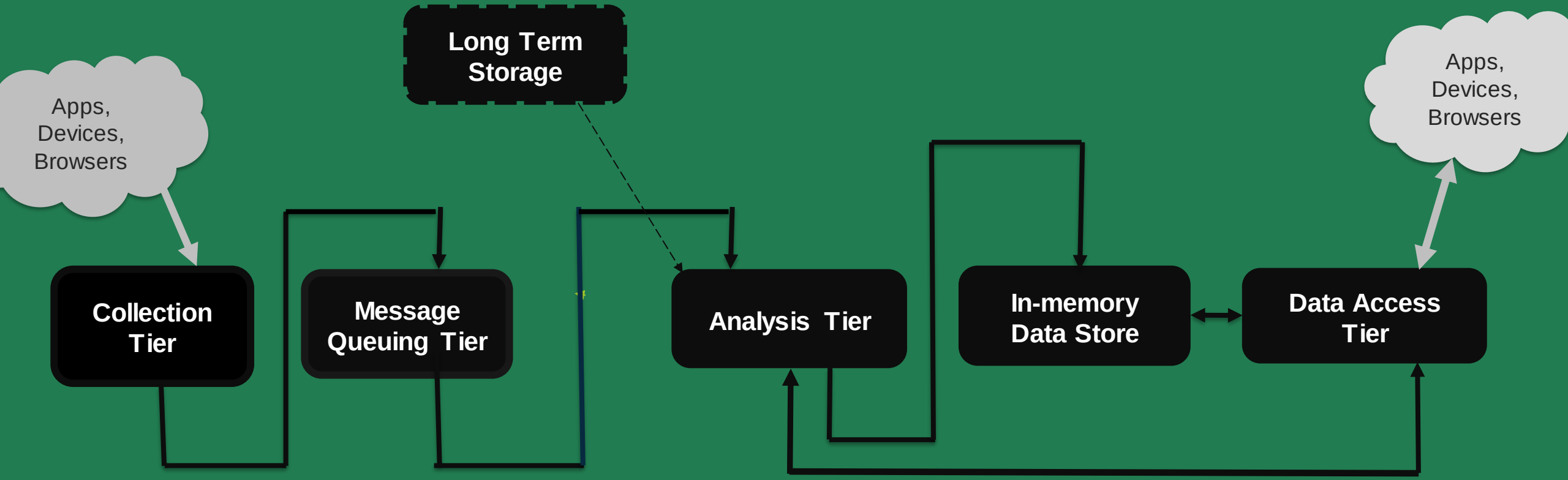
Building a Streaming (E-health) Data Pipeline: When and How?

By:Zohreh Jafari
Head of Data Team at Avihang



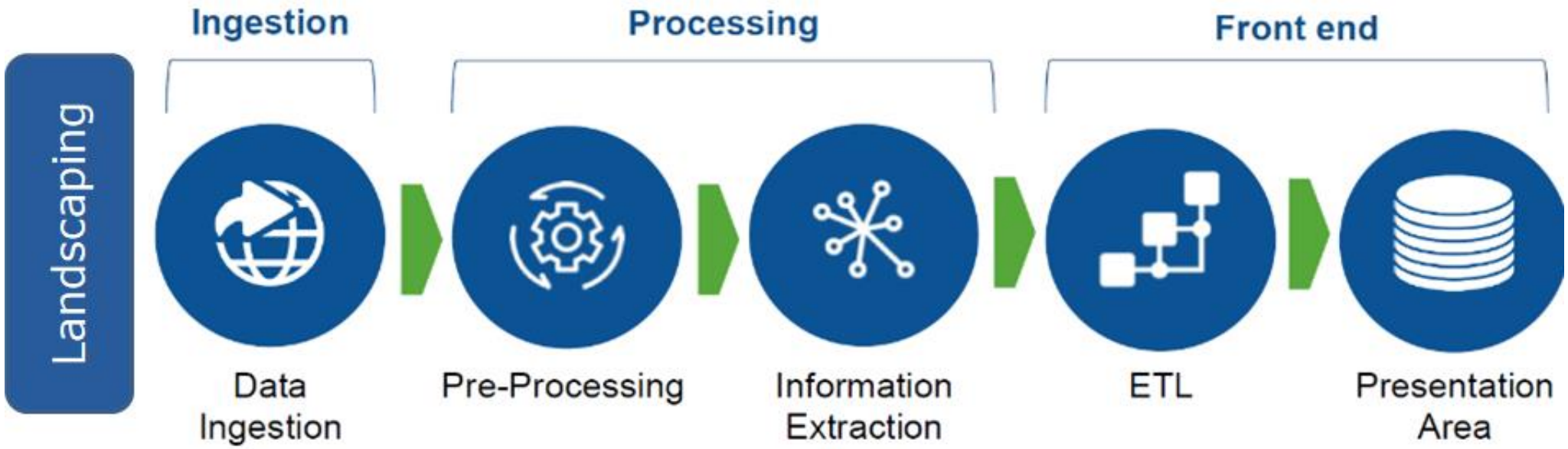
What Is A Data Pipeline?

A **data pipeline** is a series of data processing steps.

Each step delivers an output that is the input to the next step.

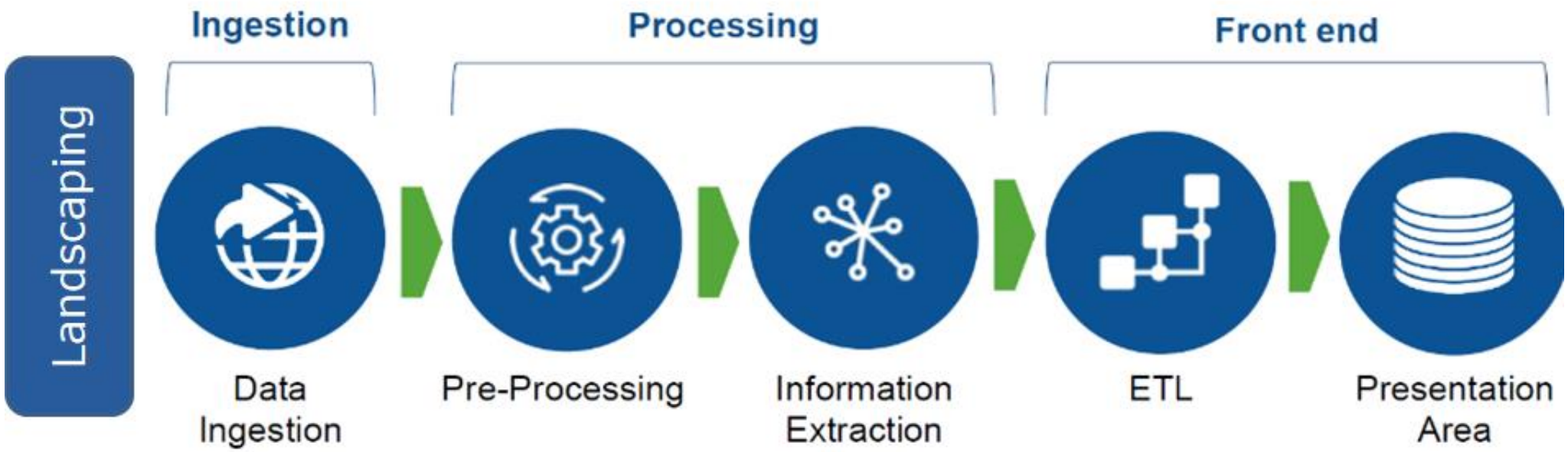
This continues until the pipeline is complete.

In some cases, independent steps may be run in parallel.



What Is A Data Pipeline? HideSlide

A **data pipeline** is a series of data processing steps. If the data is not currently loaded into the data platform, then it is ingested at the beginning of the pipeline. Then there are a series of steps in which each step delivers an output that is the input to the next step. This continues until the pipeline is complete. In some cases, independent steps may be run in parallel.



PyData

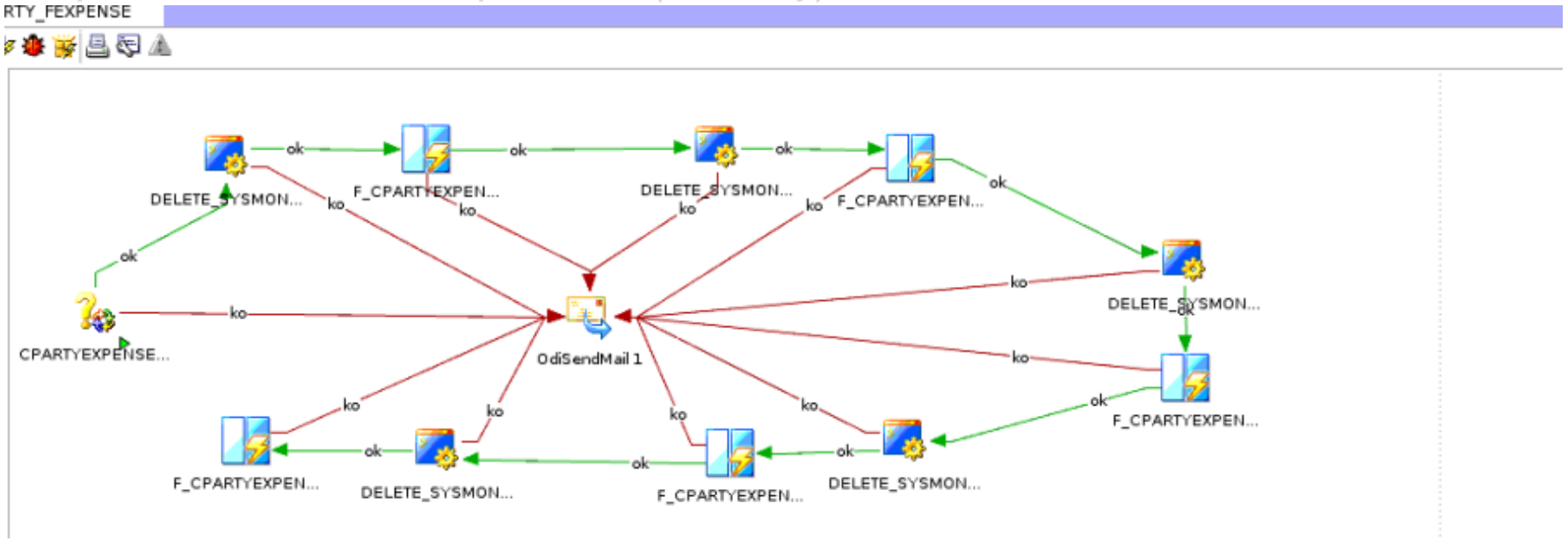
It's Really a Pipeline

Data pipelines consist of three key elements: a source, a processing step or steps, and a destination.

In some data pipelines, the destination may be called a sink.

Data pipelines enable the flow of **data from an application to a data warehouse**, from a **data lake to an analytics database**, or **into a payment processing system**, for example.

- Data Sum-Up for Doctors at the end of the day (every 10 minutes)
- Data for searching products&services and suggesting them to medicines (daily)
- Data for finance performance of each province (monthly)



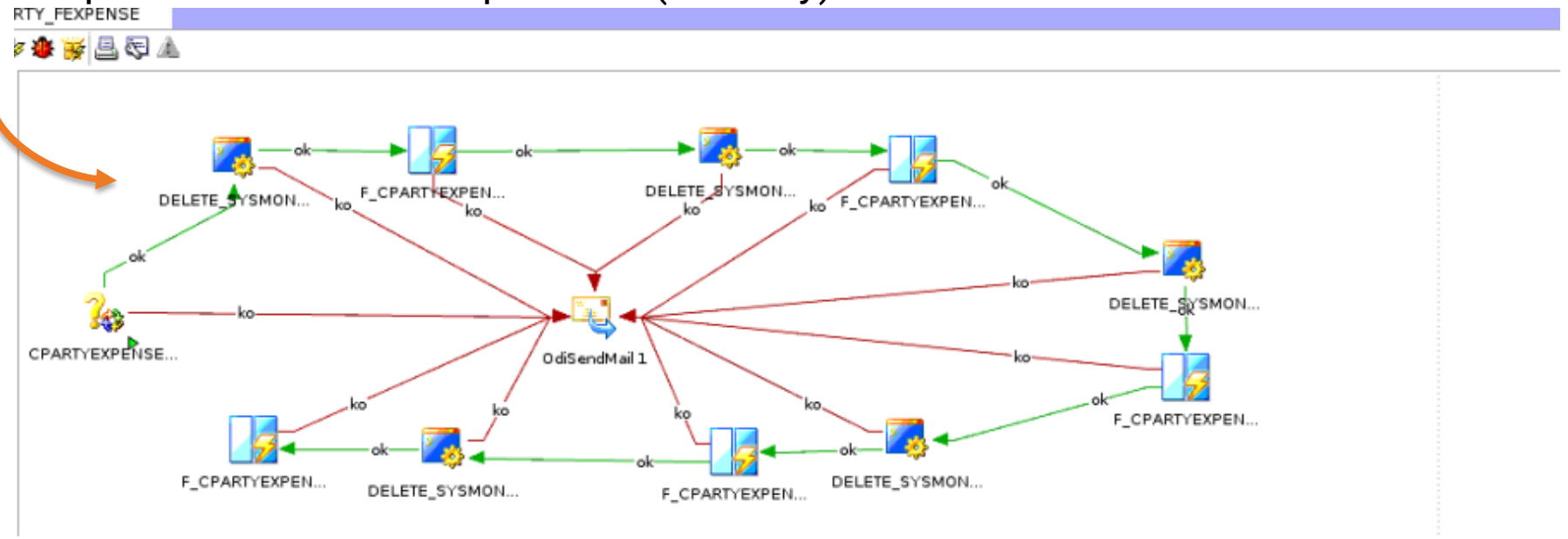
It's Really a Pipeline

Data pipelines consist of three key elements: a source, a processing step or steps, and a destination.

In some data pipelines, the destination may be called a sink.

Data pipelines enable the flow of **data from an application to a data warehouse**, from a **data lake to an analytics database**, or **into a payment processing system**, for example.

- Data Sum-Up for Doctors at the end of the day (every 10 minutes)
- Data for searching products&services and suggesting them to medicines (daily)
- Data for finance performance of each province (monthly)

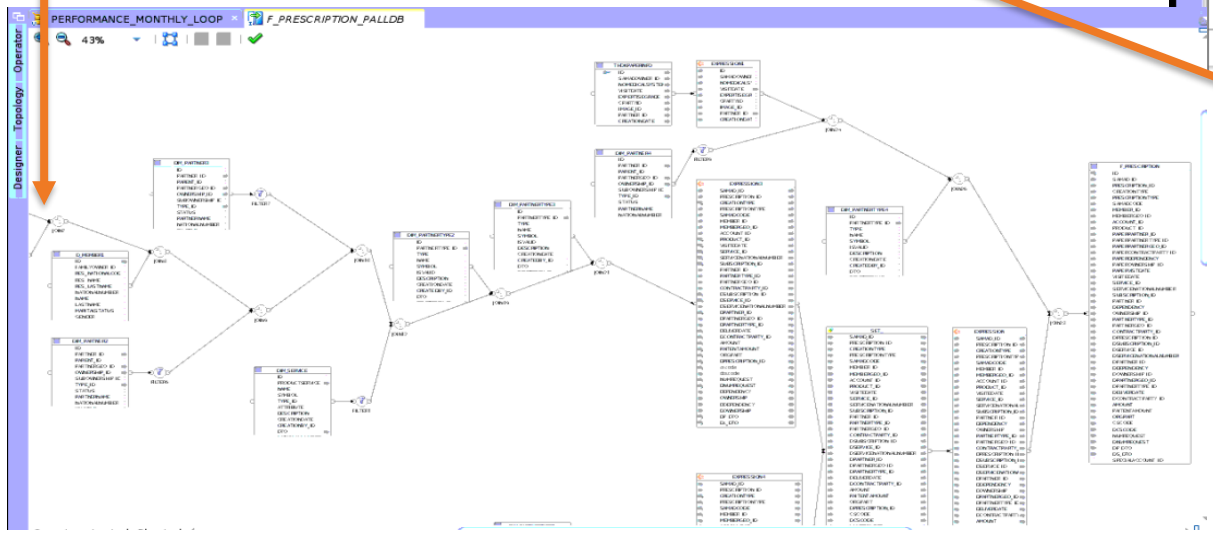


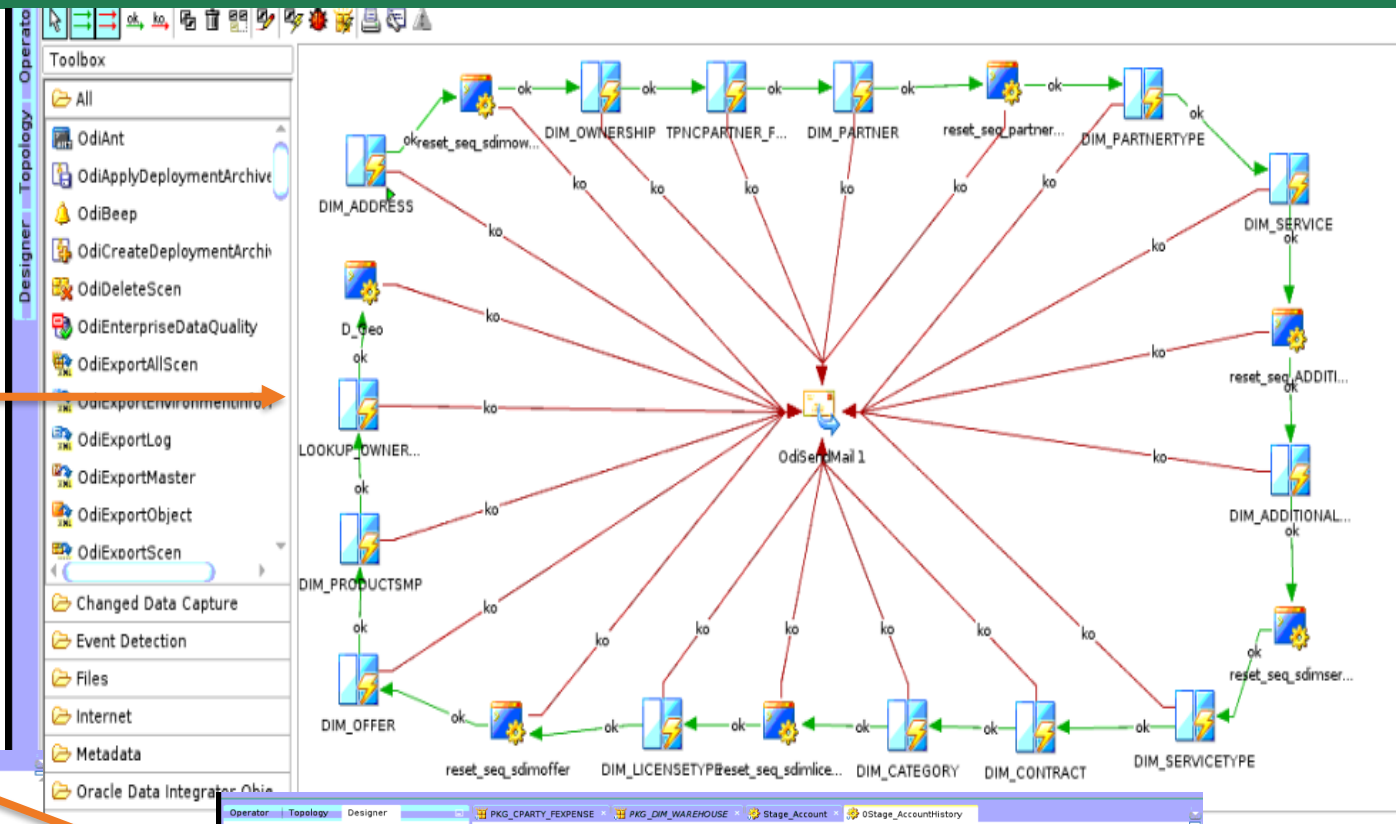
Graphic View- Oracle

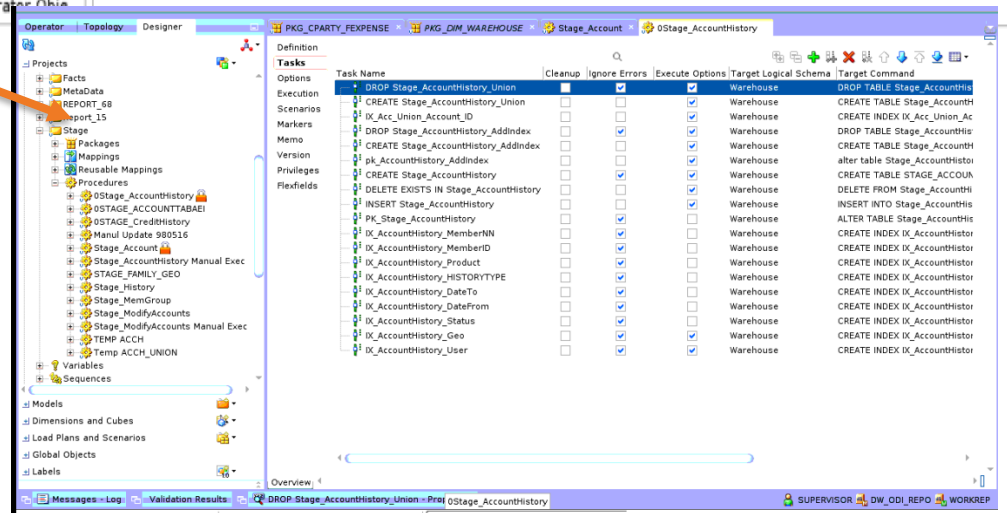
Mapping

Package: sequence of steps

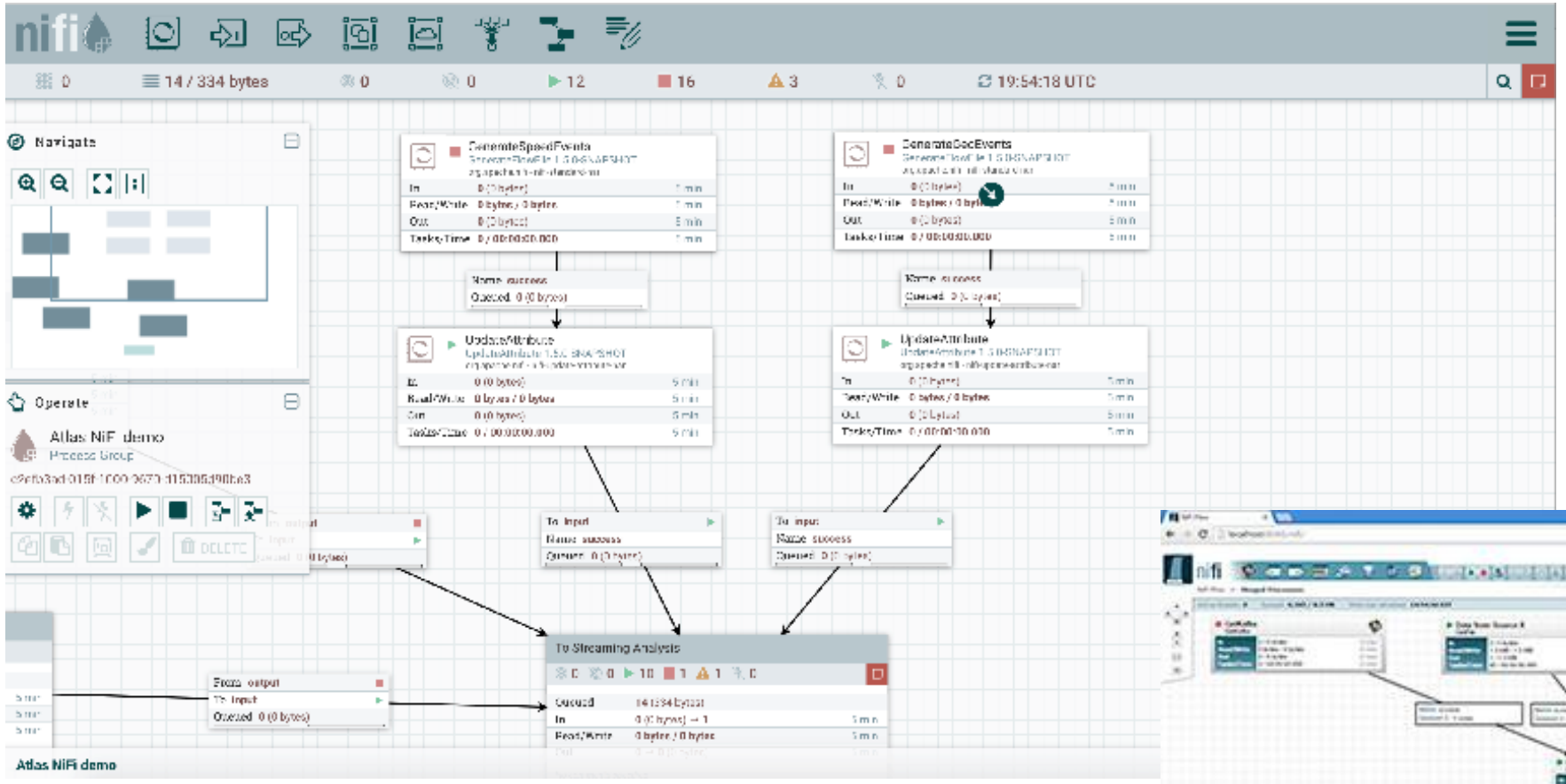
Schedule



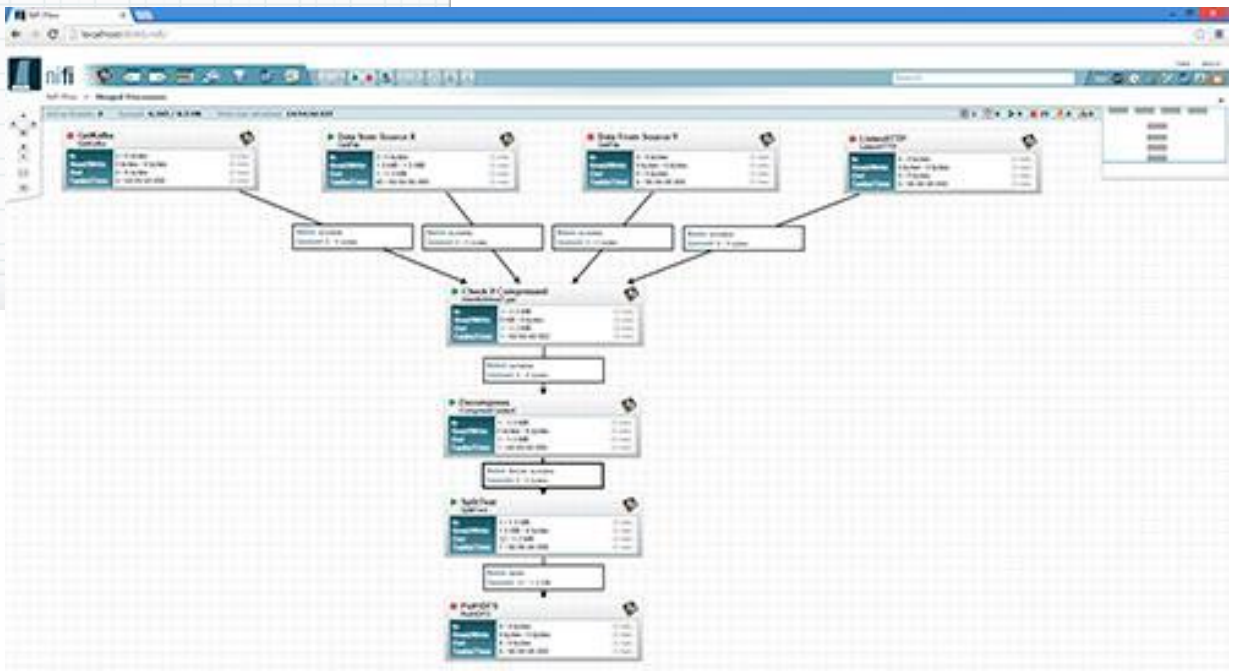




Graphic View-NIFI



- **Interaction with data sources**
- **Data conversion**
- **Information processing**
- **Delegation of functionalities**
- **Parallel**



Apache NiFi a **data transformation and routing stream execution engine.**
It loads data from one source, passes it through a process flow for processing, and dumps it into another source.

Hide Slide About NIFI

Processors

Each of the *boxes* that are part of the flow, the processors, provide a specific functionality to the flow. NiFi comes pre-loaded with a long list of operations on the data, in the form of processors:
Interaction with data sources. Uploading data from and to a wide variety of formats and technologies: HDFS, ElasticSearch, FTP, SQL databases, MongoDB, files... among many others.
Data conversion. Information structure changes, from and to JSON, XML, Avro, CSV, etc.
Information processing. Data joining and division operations, as well as their validation and transformation.
Delegation of functionalities. Processors that can transfer data to other processing systems, to perform tasks on them, that are already implemented in other systems: For example, the bi-directional communication with a third party, through Kafka queues, or the transparent execution of Apache Flume processes.

UI: Web base

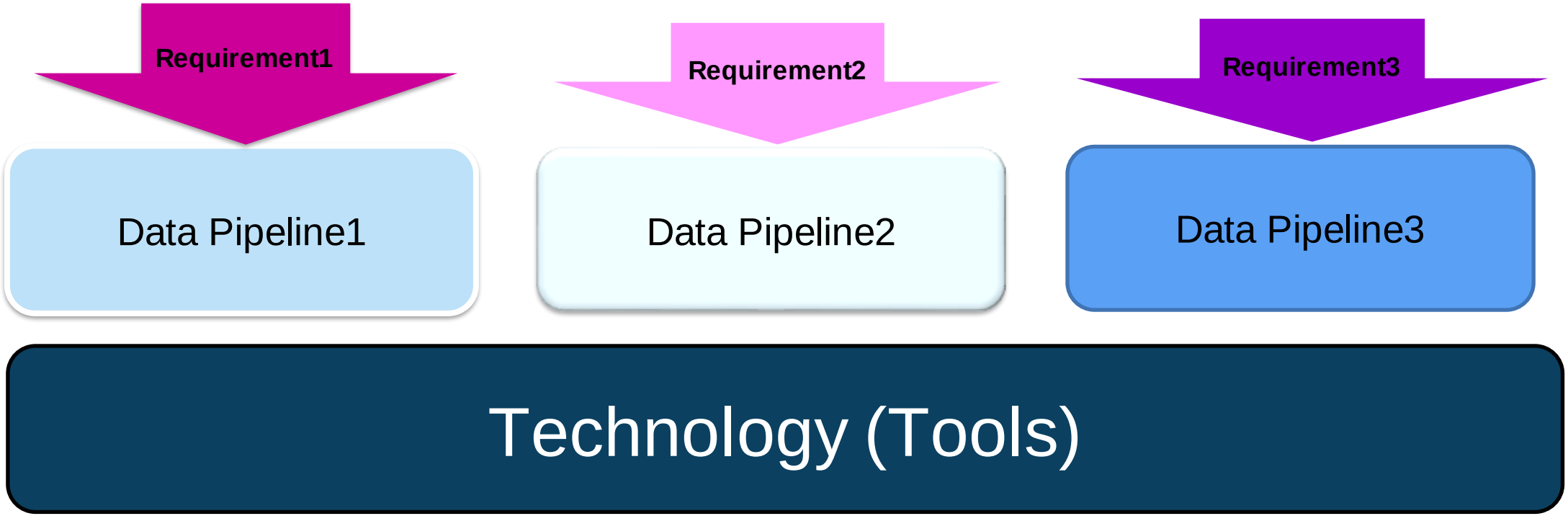
Cluster execution
To ensure optimum performance, the process defined in this tree can be executed in parallel: Several machines on which NiFi is running collaborate to perform the operations defined in one of them.

One step beyond ETL
After this first contact with Apache NiFi, we can realize that because of the simplicity of its use (completely integrated in a web application), and the power of its capabilities (with complex processor flows, provided or created to measure) is a very important tool for the transfer of data between systems.

It's Really a Pipeline

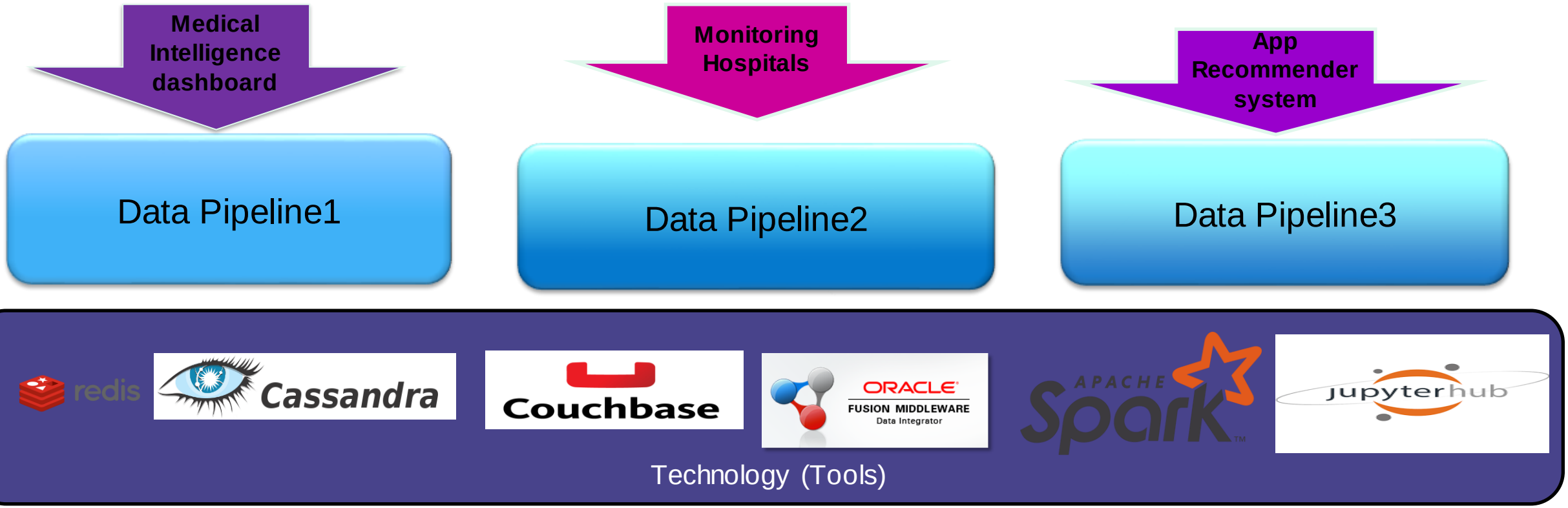
Data pipelines also may have the same source and sink, such that the pipeline is purely about modifying the data set. Any time data is processed between point A and point B (or points B, C, and D), there is a data pipeline between those points.

Common steps: data transformation, augmentation, enrichment, filtering, grouping, aggregating, and the running of algorithms against that data.



From Business to Stack

Business requirements imposes pipeline and the pipelines are built on a data stack



PyData

A Big Data Pipeline

Dramatically change in volume, variety, and velocity of data ➡ Big Data

Big data pipelines are data pipelines built to accommodate one or more of the three traits of big data.

The velocity of big data: makes it appealing to build streaming data pipelines for big data. Then data can be captured and processed in real time so some action can then occur.

The volume of big data: requires that data pipelines must be scalable, as the volume can be variable over time. In practice, there are likely to be many big data events that occur simultaneously or very close together, so the big data pipeline must be able to scale to process significant volumes of data concurrently.

The variety of big data: requires that big data pipelines be able to recognize and process data in many different formats—structured, unstructured, and semi-structured.

PyData

A Big Data Pipeline

Dramatically change in volume, variety, and velocity of data ➡ Big Data

Big data pipelines are data pipelines built to accommodate one or more of the three traits of big data.

The velocity of big data: makes it appealing to build streaming data pipelines for big data. Then data can be captured and processed in real time so some action can then occur.

The volume of big data: requires that data pipelines must be scalable, as the volume can be variable over time. In practice, there are likely to be many big data events that occur simultaneously or very close together, so the big data pipeline must be able to scale to process significant volumes of data concurrently.

The variety of big data: requires that big data pipelines be able to recognize and process data in many different formats—structured, unstructured, and semi-structured.

A Big Data Pipeline

Dramatically change in volume, variety, and velocity of data ➡ Big Data

Big data pipelines are data pipelines built to accommodate one or more of the three traits of big data.

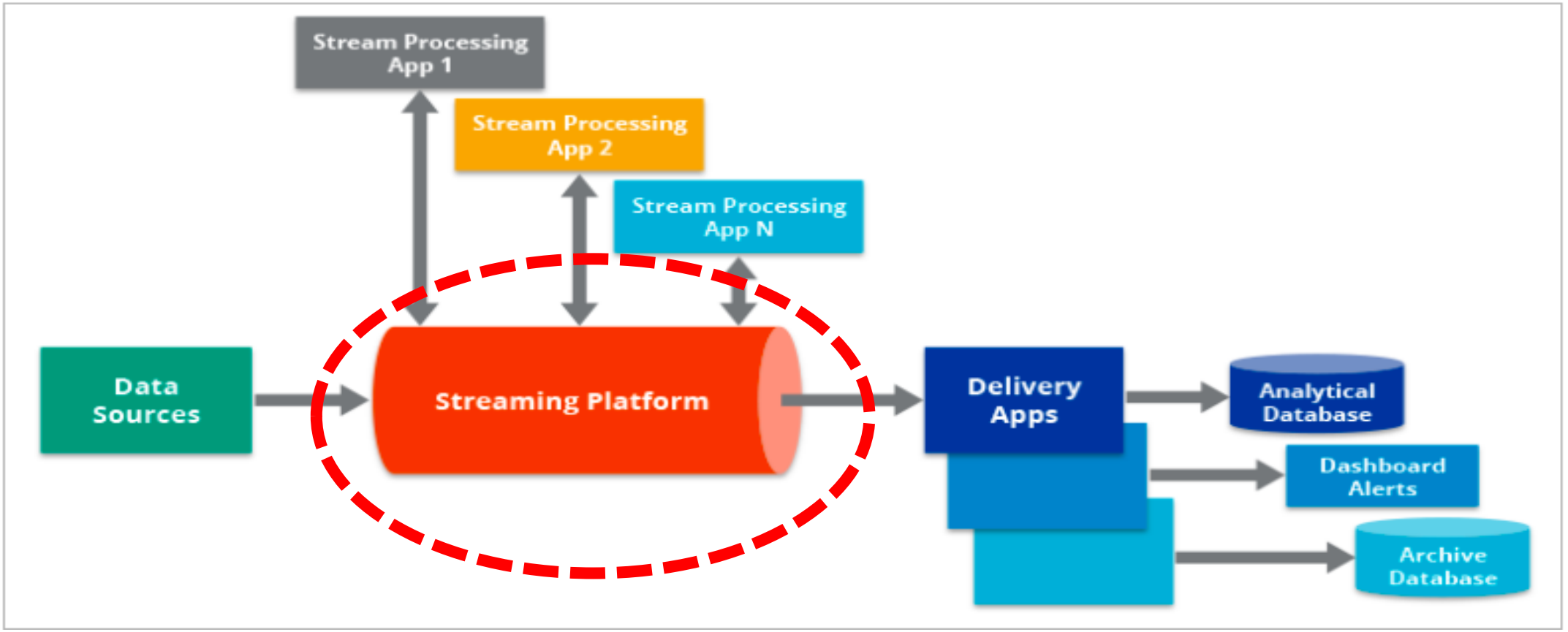
The velocity of big data: makes it appealing to build streaming data pipelines for big data. Then data can be captured and processed in real time so some action can then occur.

The volume of big data: requires that data pipelines must be scalable, as the volume can be variable over time. In practice, there are likely to be many big data events that occur simultaneously or very close together, so the big data pipeline must be able to scale to process significant volumes of data concurrently.

The variety of big data: requires that big data pipelines be able to recognize and process data in many different formats—structured, unstructured, and semi-structured.

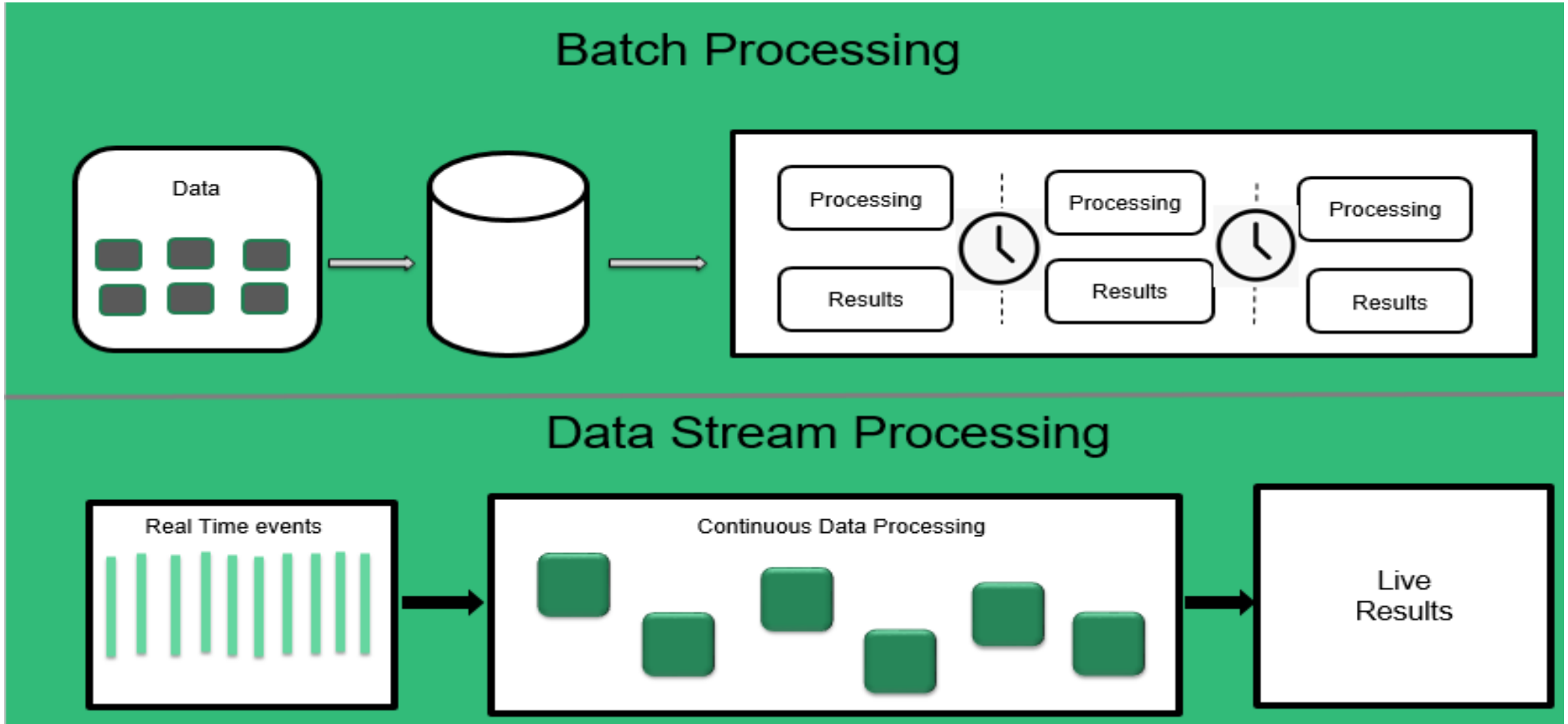
Streaming Data Pipeline

A streaming data pipeline **flows data continuously from source to destination as it is created**, making it useful along the way. Streaming data pipelines are used to populate data lakes or data warehouses, or to publish to a messaging system or data stream.



Streaming Data Pipeline

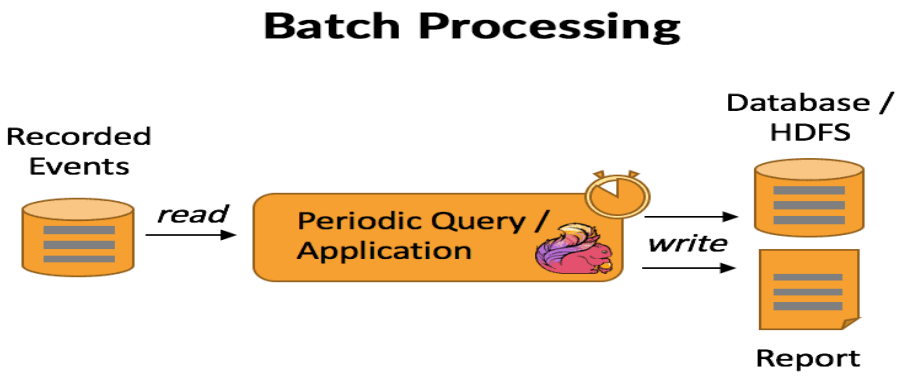
Data as an ocean, then batch data can be referred to as a bucket of water, multiple buckets of different sizes, whereas stream data can be considered to be a water pipe that is continuously pumping water from the ocean.

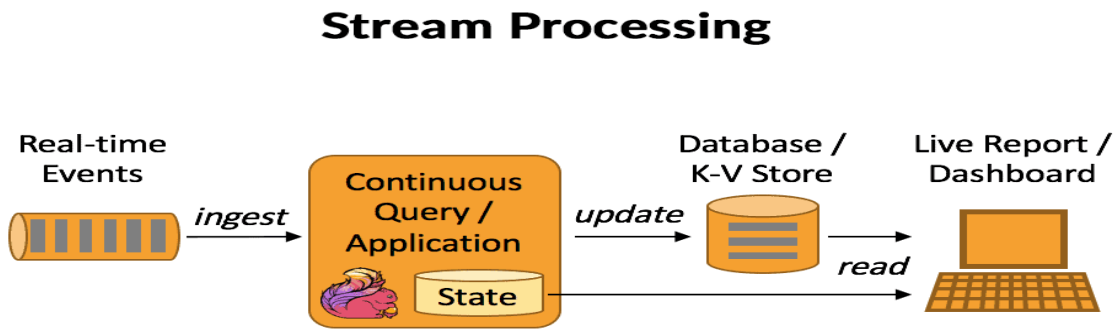


Way to Streaming

Batch Processing vs. Stream Processing

- **The nightly job that, during "quiet" time, would process the previous day's transactions;**
- **The monthly report that provided summary statistics for dashboards, etc.**
- Batch processing was straightforward, scalable and predictable, and enterprises tolerated the latency inherent in the model
- While batch sizes were shrinking, data volumes grew, along with a demand for fault tolerance. Distributed storage architectures blossomed with Hadoop, Cassandra, S3 and many other technologies.
- **Spark** presented a consistent API whether running on a cluster or as a standalone application.
- Now developers could write distributed applications and test them at small scale – even on their own laptop – before rolling them out to a cluster of hundreds or thousands of nodes.
- Many data engineers looking for higher-level tools to build data *pipelines* – the plumbing that moves events from where they are generated to where they can be analyzed.

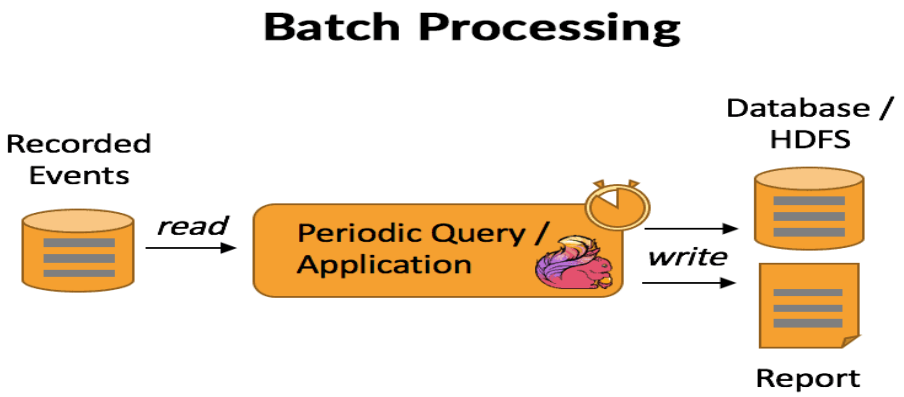


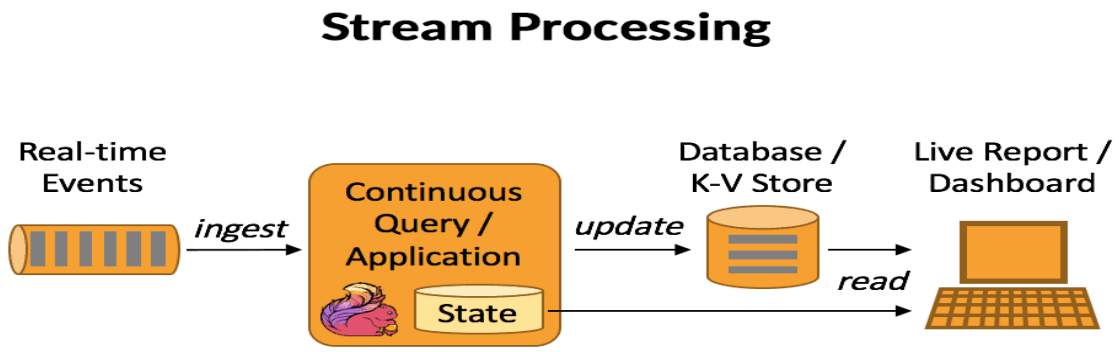


Way to Streaming

Batch Processing vs. Stream Processing

- The nightly job that, during "quiet" time, would process the previous day's transactions;
- The monthly report that provided summary statistics for dashboards, etc.
- **Batch processing was straightforward, scalable and predictable, and enterprises tolerated the latency inherent in the model**
- While batch sizes were shrinking, data volumes grew, along with a demand for fault tolerance. Distributed storage architectures blossomed with Hadoop, Cassandra, S3 and many other technologies.
- **Spark** presented a consistent API whether running on a cluster or as a standalone application.
- Now developers could write distributed applications and test them at small scale – even on their own laptop – before rolling them out to a cluster of hundreds or thousands of nodes.
- Many data engineers looking for higher-level tools to build data *pipelines* – the plumbing that moves events from where they are generated to where they can be analyzed.



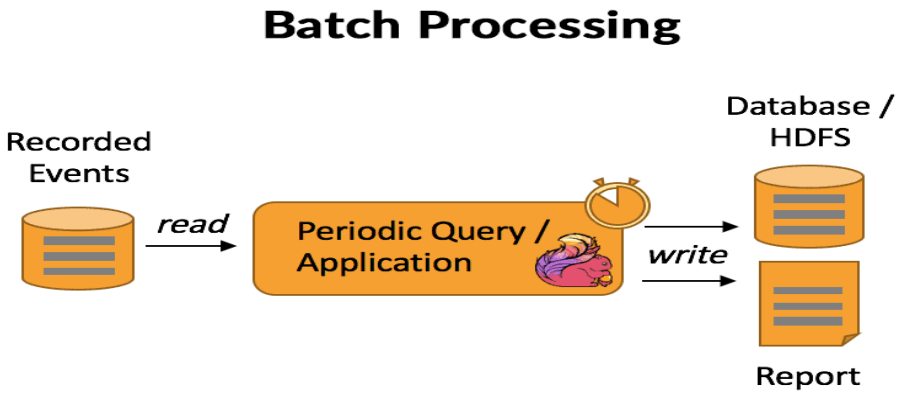


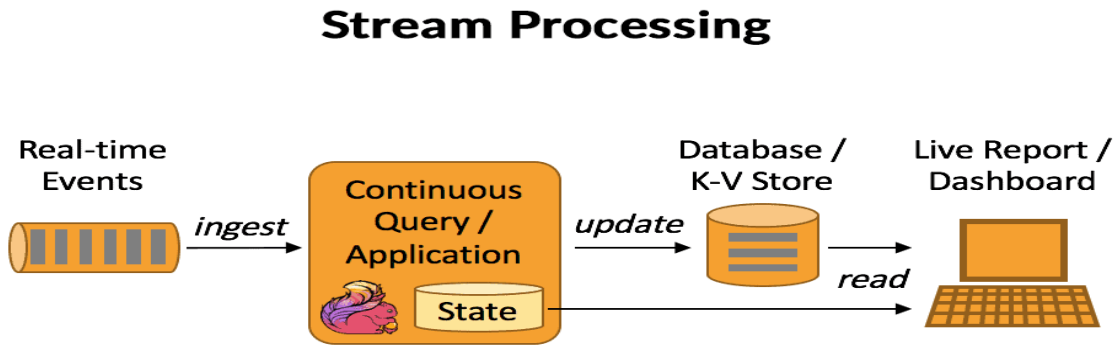
Way to Streaming

Batch Processing vs. Stream Processing

- The nightly job that, during “quiet” time, would process the previous day’s transactions;
- The monthly report that provided summary statistics for dashboards, etc.
- Batch processing was straightforward, scalable and predictable, and enterprises tolerated the latency inherent in the n
- Whil uted storage arch
- Spa
- Now aptop – befo
- Man vents from where they are generated to where they can be analyzed.

ETL drawbacks
global schema / for very large amount of data
error - prone
operational cost of ETL is high, slow and resource intensive

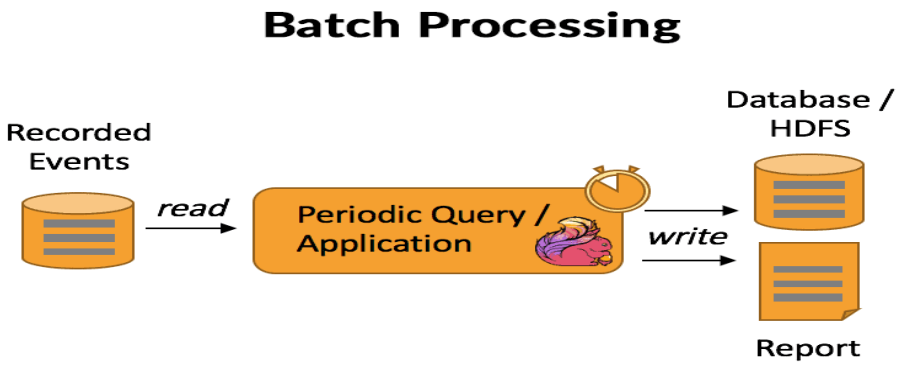


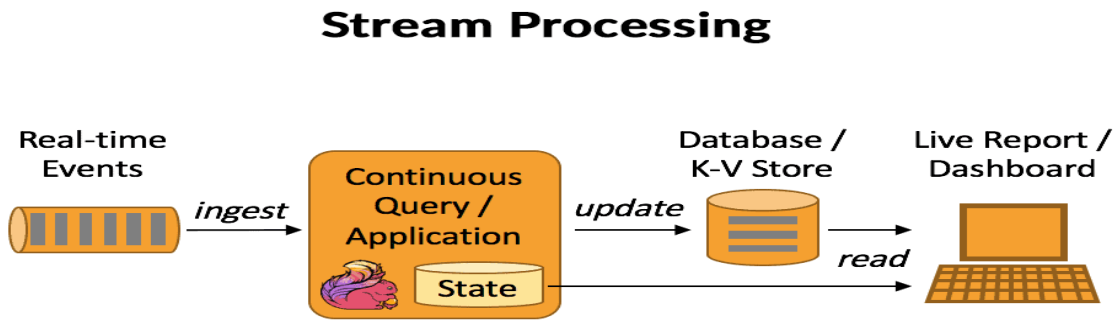


Way to Streaming

Batch Processing vs. Stream Processing

- The nightly job that, during "quiet" time, would process the previous day's transactions;
- The monthly report that provided summary statistics for dashboards, etc.
- Batch processing was straightforward, scalable and predictable, and enterprises tolerated the latency inherent in the model
- **While sizes were shrinking, data volumes grew, along with a demand for fault tolerance. Distributed storage architectures blossomed with Hadoop, Cassandra, S3 and many other technologies**.
- **Spark** presented a consistent API whether running on a cluster or as a standalone application.
- Now developers could write distributed applications and test them at small scale – even on their own laptop – before rolling them out to a cluster of hundreds or thousands of nodes.
- Many data engineers looking for higher-level tools to build data *pipelines* – the plumbing that moves events from where they are generated to where they can be analyzed.

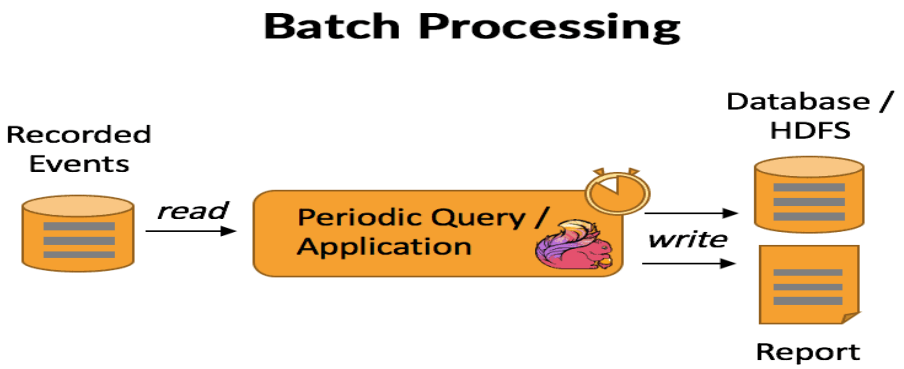


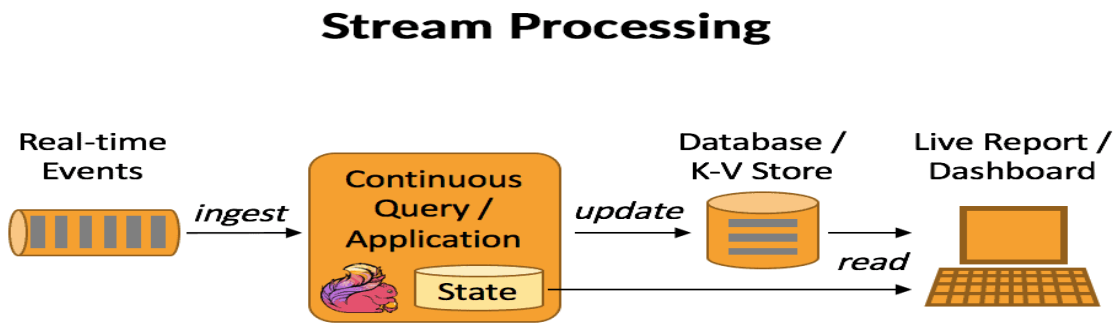


Way to Streaming

Batch Processing vs. Stream Processing

- The nightly job that, during "quiet" time, would process the previous day's transactions;
- The monthly report that provided summary statistics for dashboards, etc.
- Batch processing was straightforward, scalable and predictable, and enterprises tolerated the latency inherent in the model
- While batch sizes were shrinking, data volumes grew, along with a demand for fault tolerance. Distributed storage architectures blossomed with Hadoop, Cassandra, S3 and many other technologies.
- **Spark presented a consistent API whether running on a cluster or as a standalone application**.
- Now developers could write distributed applications and test them at small scale – even on their own laptop – before rolling them out to a cluster of hundreds or thousands of nodes.
- Many data engineers looking for higher-level tools to build data *pipelines* – the plumbing that moves events from where they are generated to where they can be analyzed.

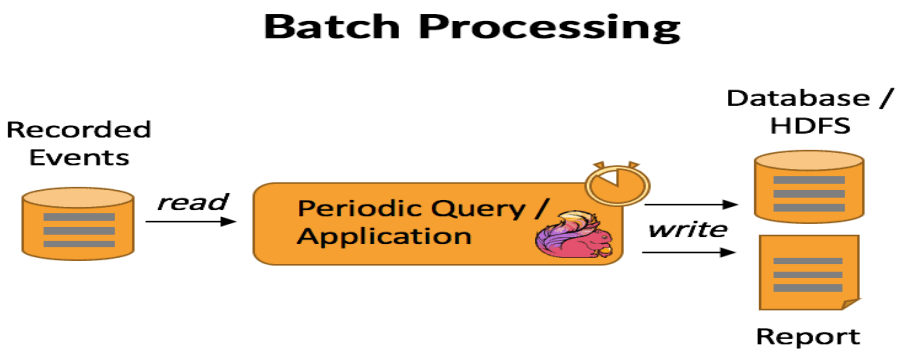


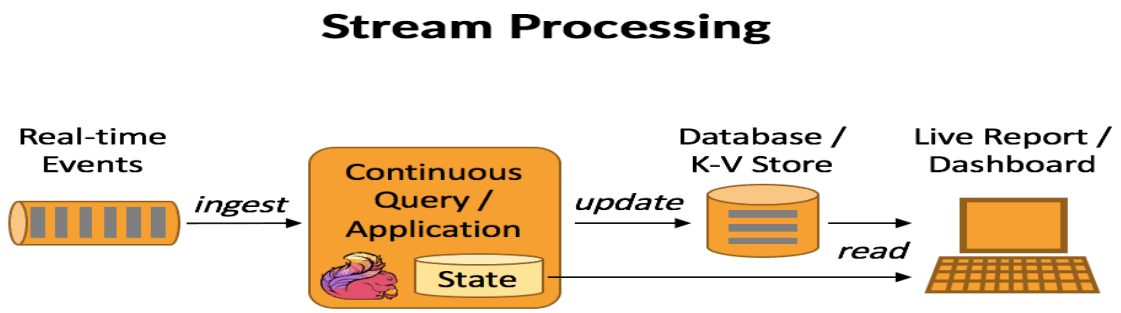


Way to Streaming

Batch Processing vs. Stream Processing

- The nightly job that, during "quiet" time, would process the previous day's transactions;
- The monthly report that provided summary statistics for dashboards, etc.
- Batch processing was straightforward, scalable and predictable, and enterprises tolerated the latency inherent in the model
- While batch sizes were shrinking, data volumes grew, along with a demand for fault tolerance. Distributed storage architectures blossomed with Hadoop, Cassandra, S3 and many other technologies.
- **Spark** presented a consistent API whether running on a cluster or as a standalone application.
- **Now developers could write distributed applications and test them at small scale – even on their own laptop – before rolling them out to a cluster of hundreds or thousands of nodes.**
- Many data engineers looking for higher-level tools to build data *pipelines* – the plumbing that moves events from where they are generated to where they can be analyzed.

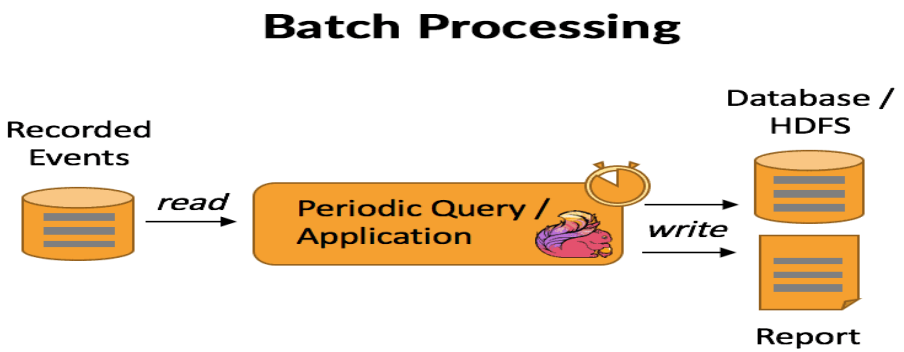


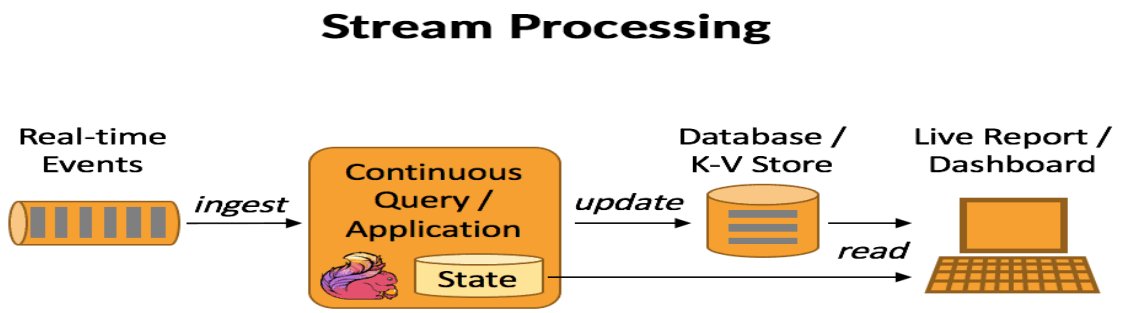


Way to Streaming

Batch Processing vs. Stream Processing

- The nightly job that, during "quiet" time, would process the previous day's transactions;
- The monthly report that provided summary statistics for dashboards, etc.
- Batch processing was straightforward, scalable and predictable, and enterprises tolerated the latency inherent in the model
- While batch sizes were shrinking, data volumes grew, along with a demand for fault tolerance. Distributed storage architectures blossomed with Hadoop, Cassandra, S3 and many other technologies.
- Spark presented a consistent API whether running on a cluster or as a standalone application.
- Now developers could write distributed applications and test them at small scale – even on their own laptop – before rolling them out to a cluster of hundreds or thousands of nodes.
- **For higher-level tools to build data *pipelines* – the plumbing that moves events from where they are generated to where they can be analyzed.**





When ?

Businesses crave ever more timely data, and switching to streaming is a good way to achieve lower latency.

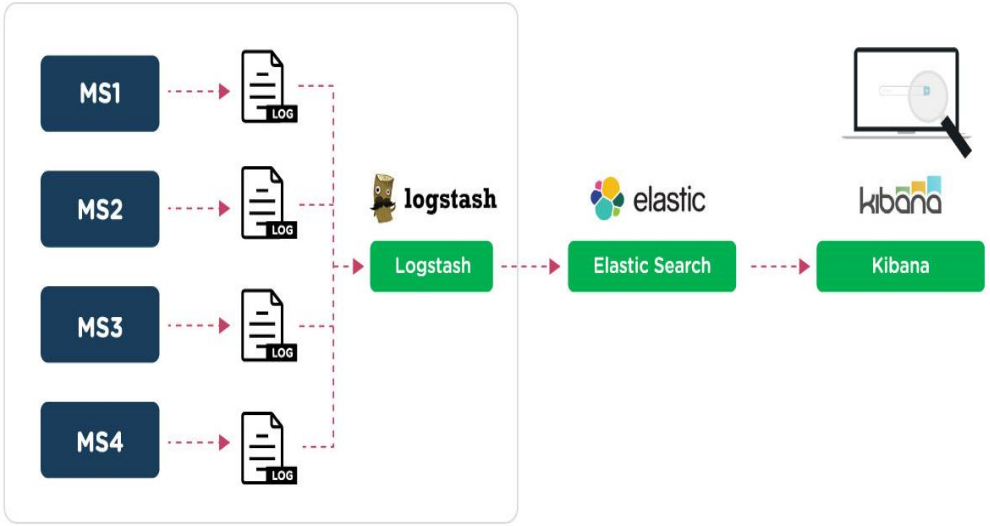
JFK John F Kennedy Intl Airport (JFK/KJFK) - FlightAware

PyData

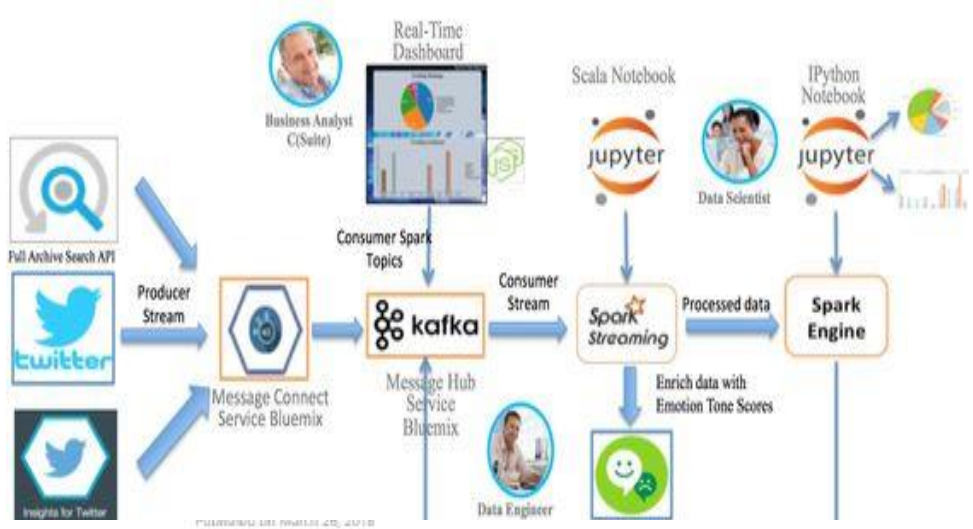
When ?

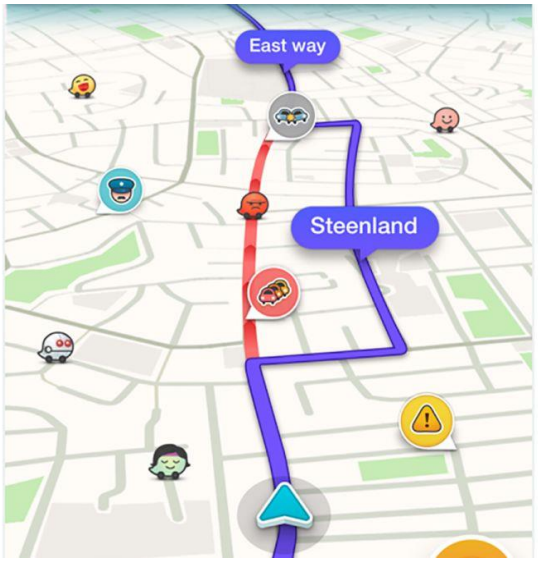
The massive, unbounded data sets that are increasingly common in modern business are more easily tamed using a system designed for such never-ending volumes of data.

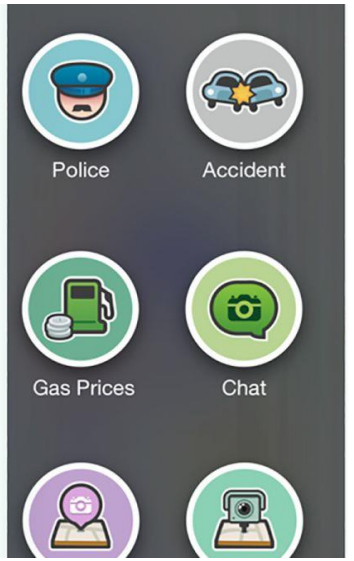
Single-server databases are replaced by a myriad of distributed data platform that operate at company-wide scale;



When ?







Processing data as they arrive spreads workloads out more evenly over time, yielding more consistent and predictable consumption of resources.

Interesting Comparison

- Think of batch processing as producing a movie. The production has a beginning, middle, and an end. When the work is complete, there is a whole, finished product that will not change in the future.

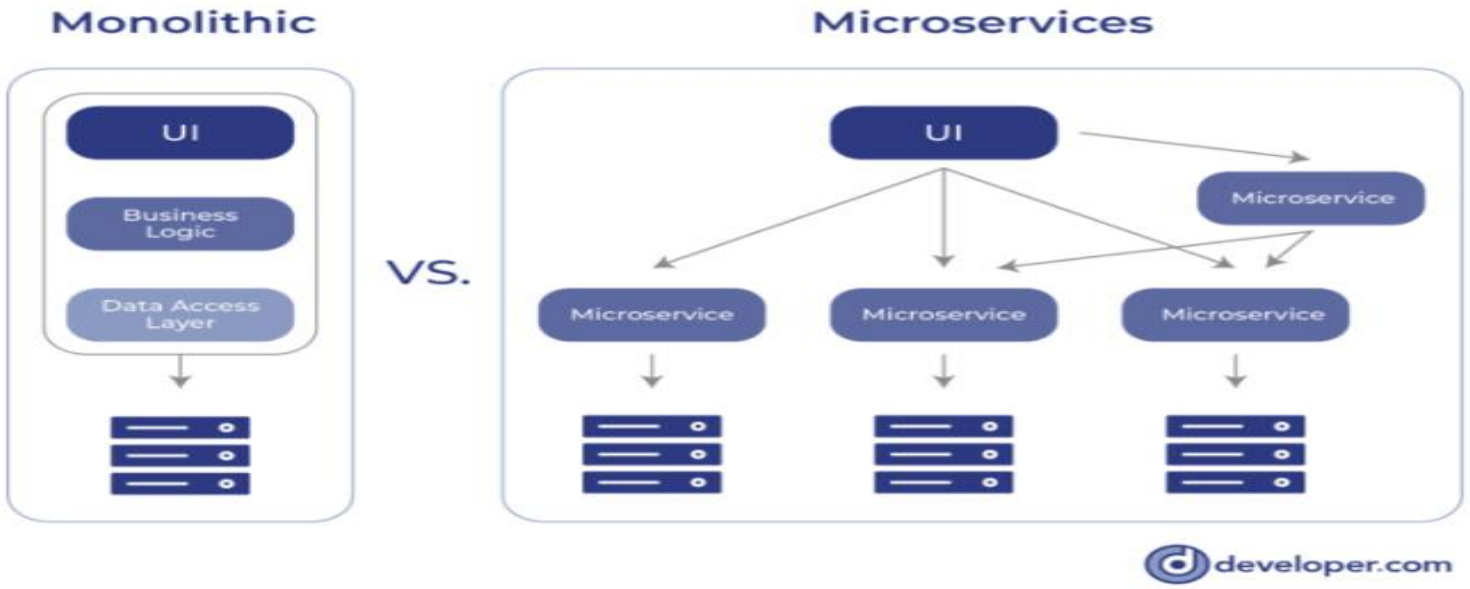
- Stream processing is more like an episodic show. All of the production tasks still happen, but on a rolling time frame with endless permutations.

VS

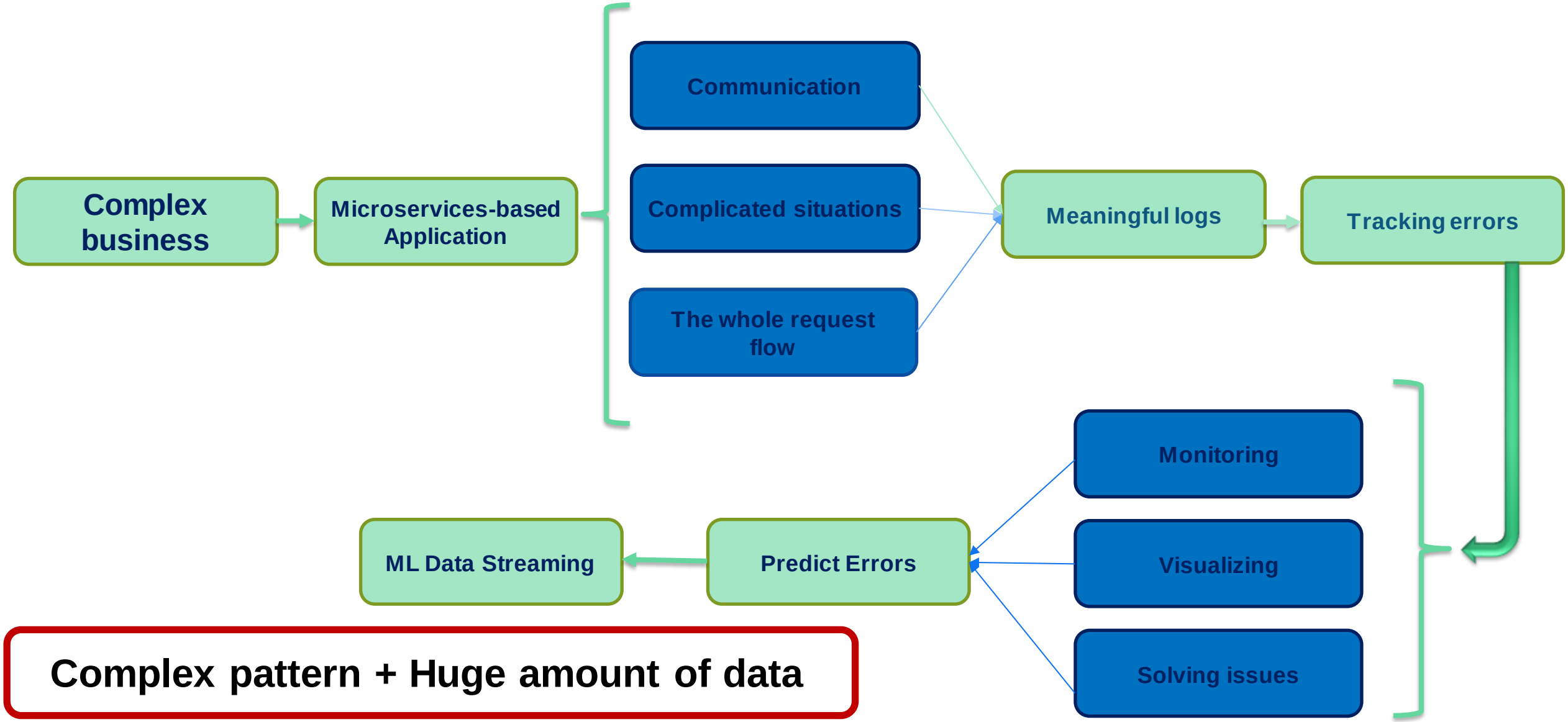
Our Case: Logs

A microservices-based application might have several services running and communicating amongst themselves. Things get complicated when one or more services fail, and you need to know which one failed and why.
It's also essential that you're able to comprehend the whole request flow — which services were invoked, how many times, and in what order.

To answer these questions, you should have meaningful logs available and a unique Id to correlate requests.
You should have a proper way to track errors that might span several microservices.
Moreover, the complexities of monitoring and logging grow exponentially when the business logic of your applications spans across several microservices.
Logging in a monolith is as easy as publishing data to a single log file and retrieving it later.



Our Case: Logs



Complex pattern + Huge amount of data

PyData

Fraud Detection/Prevention

Database

Cassandra
Oracle
Redis

WH

ETL

Data Analysis

Output

Dashboard
Table

Real Data

Tool

Oracle Data Integrator

Language

ORACLE P/L SQL

AI

KMeans Clustering

Statistics

Standardization
Cosine Similarity

Tool

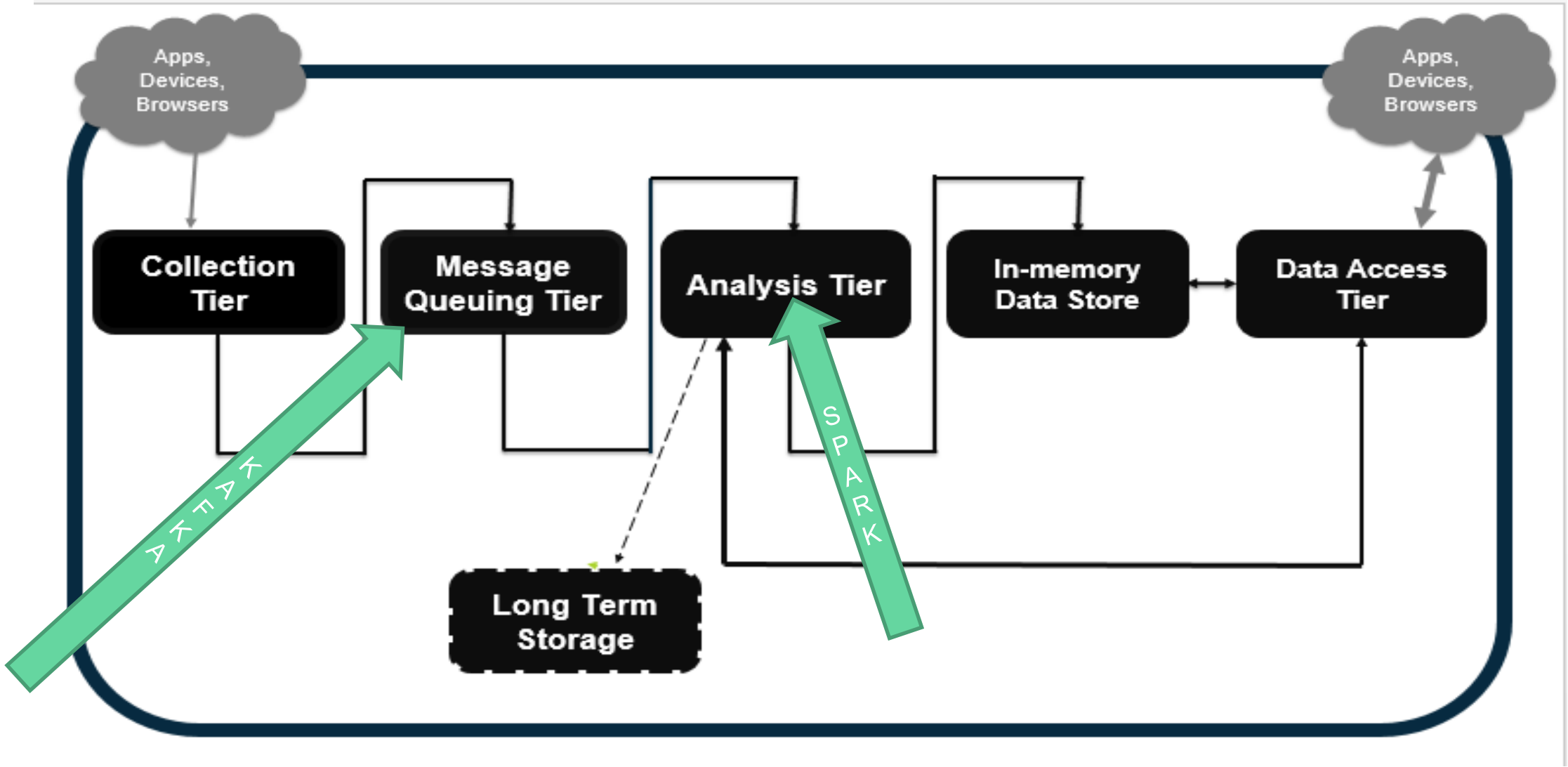
PySpark

Language

Python

How?

The streaming data architecture blueprint



Example - Online Insurance Guidelines

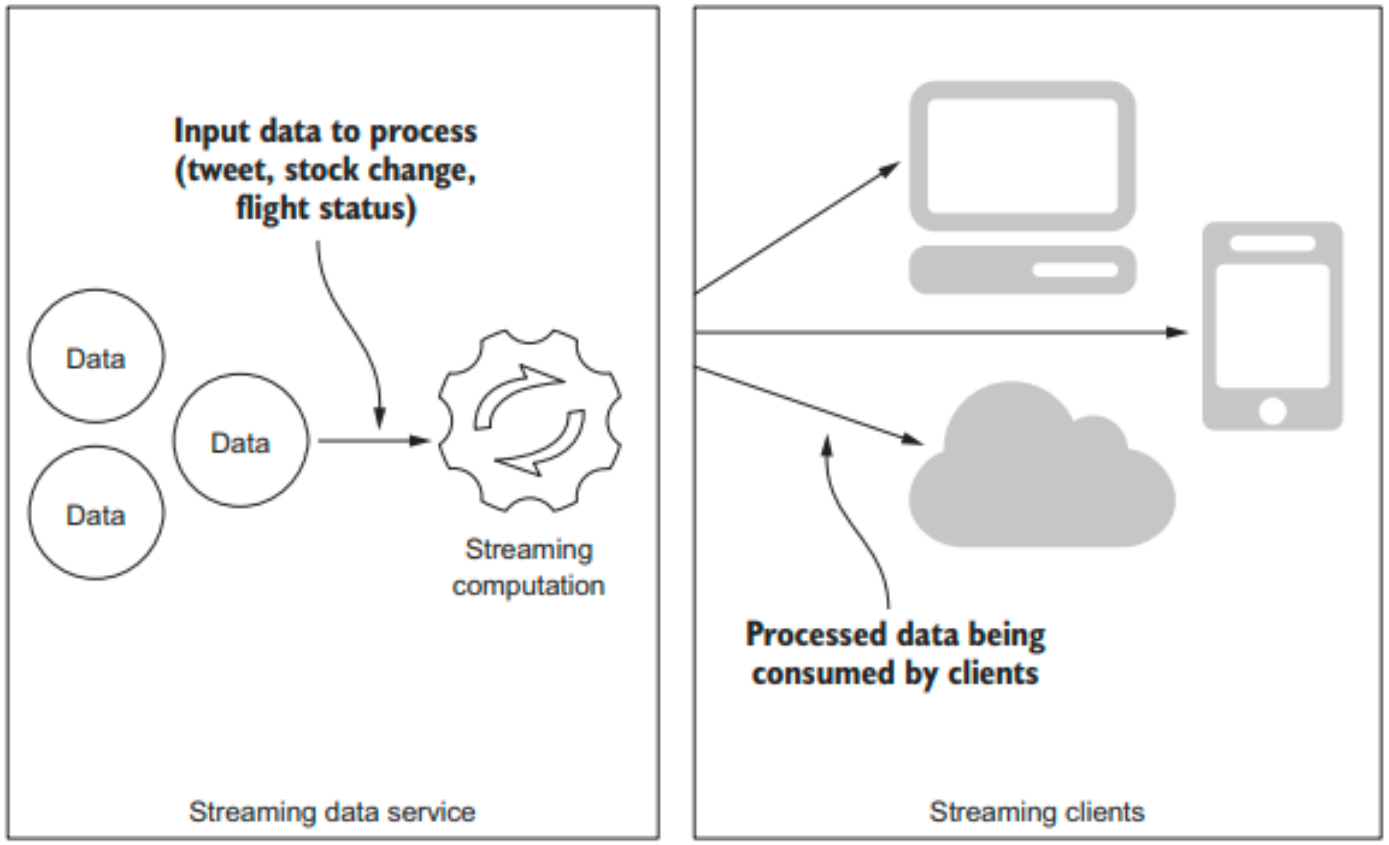
- ✓ Collection tier—When a doctor prescribes a treatment or some medication, it is collected by the ERX services. (or a medication is delivered)

- ✓ Message queuing tier — All the services runs data centers in locations across the country, and conceivably the collection of a prescription doesn't happen in the same location as the analysis of it.

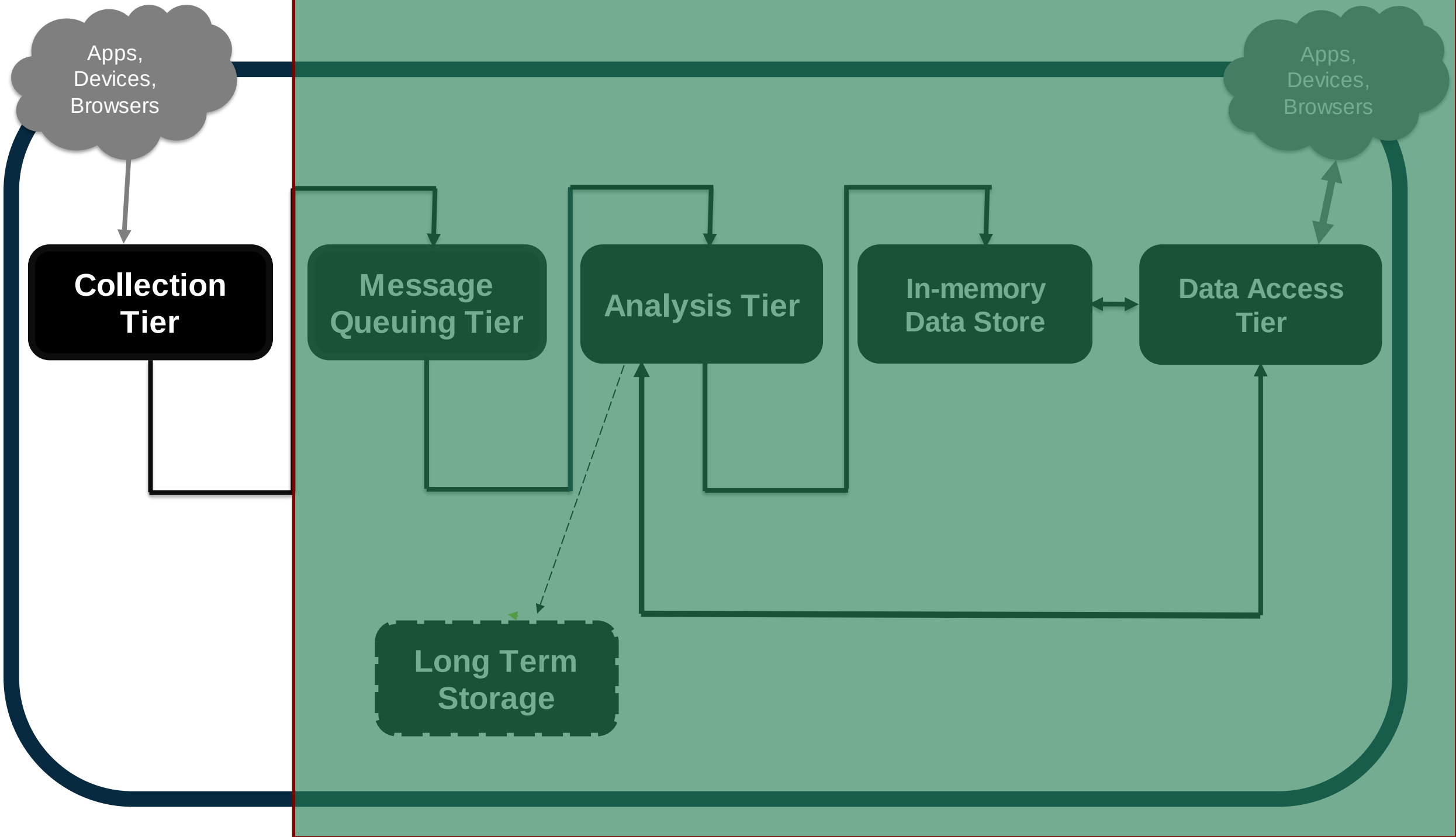
- ✓ Analysis tier— The minimum of example is calculating the price.

- ✓ Long-term storage tier—Even though we're not going to discuss this optional tier in depth here, you can probably guess that summary of all actions are gathered in tables.
- ✓ In-memory data store tier—The prescriptions that are mere seconds old are most likely held in an in-memory data store for, say, recommendation.
- ✓ Data access—All system clients need to be connected to ERX to access the service, see the history and detail of their performance.

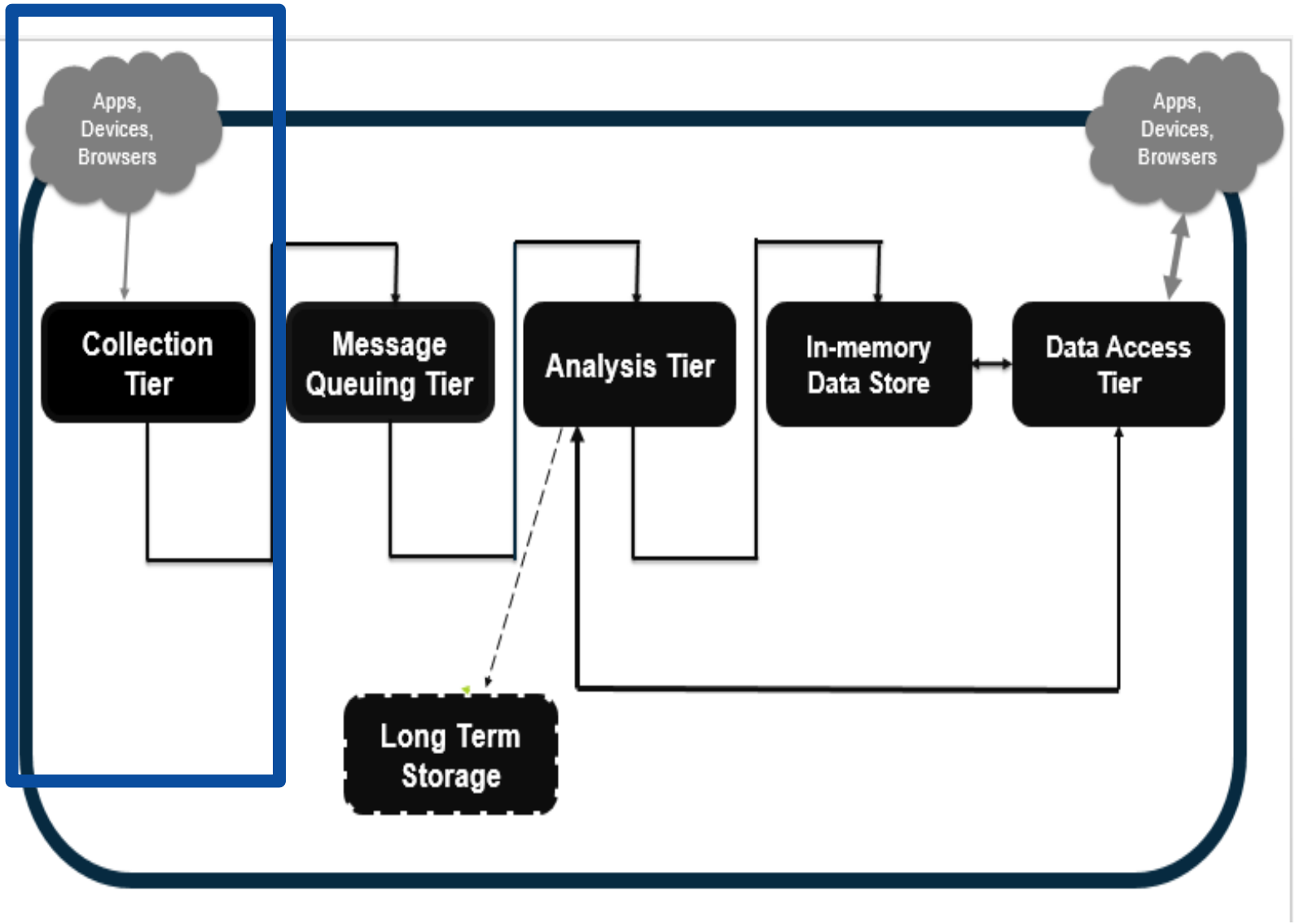
A precise Definition ?



The main Goal:
designing systems that deliver the information a client requests at the moment it is needed


Apps,
Devices,
Browsers
Apps,
Devices,
Browsers
Collection Tier
Message Queuing Tier
Analysis Tier
In-memory Data Store
Data Access Tier
Long Term Storage

Collection Tier



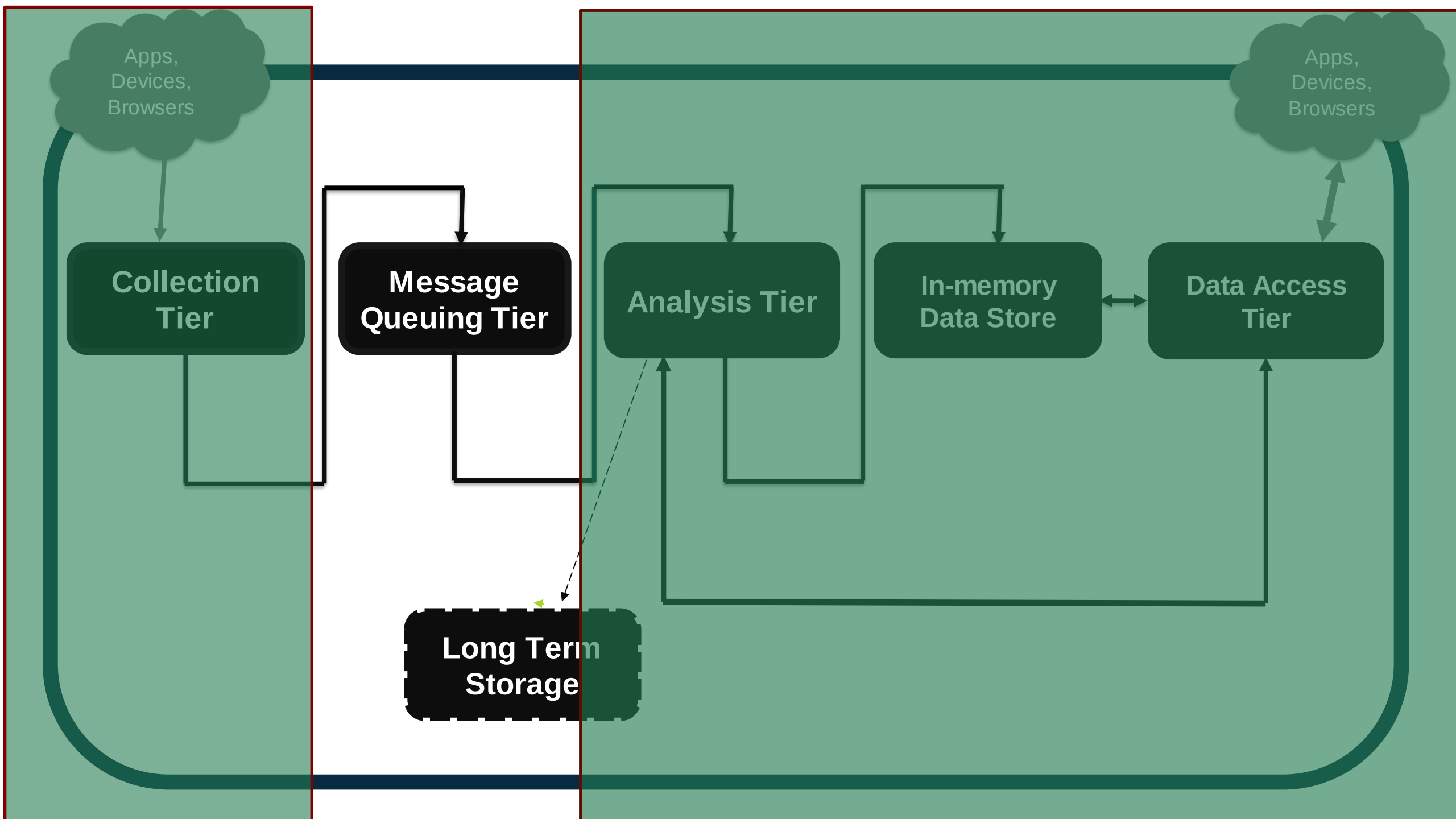
Fault tolerance

scale

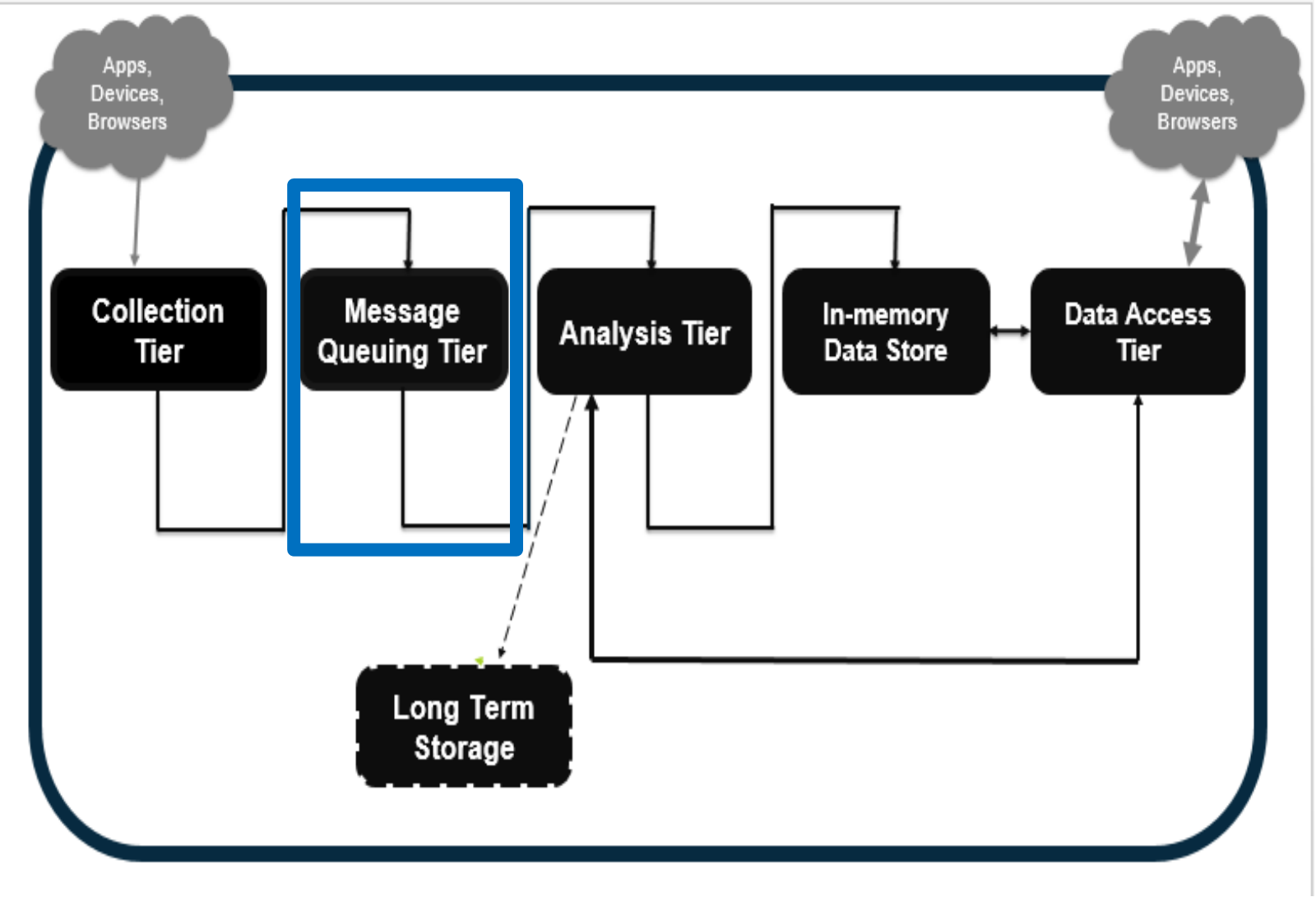
logging

- **Request/response pattern**
- **Publish/subscribe pattern**
- **One-way pattern**
- **Request/acknowledge pattern**
- **Stream pattern**

https://stream.meetup.com/2/rsvps


Apps,
Devices,
Browsers
Apps,
Devices,
Browsers
Collection Tier
Message Queuing Tier
Analysis Tier
In-memory Data Store
Data Access Tier
Long Term Storage

Messaging Tier



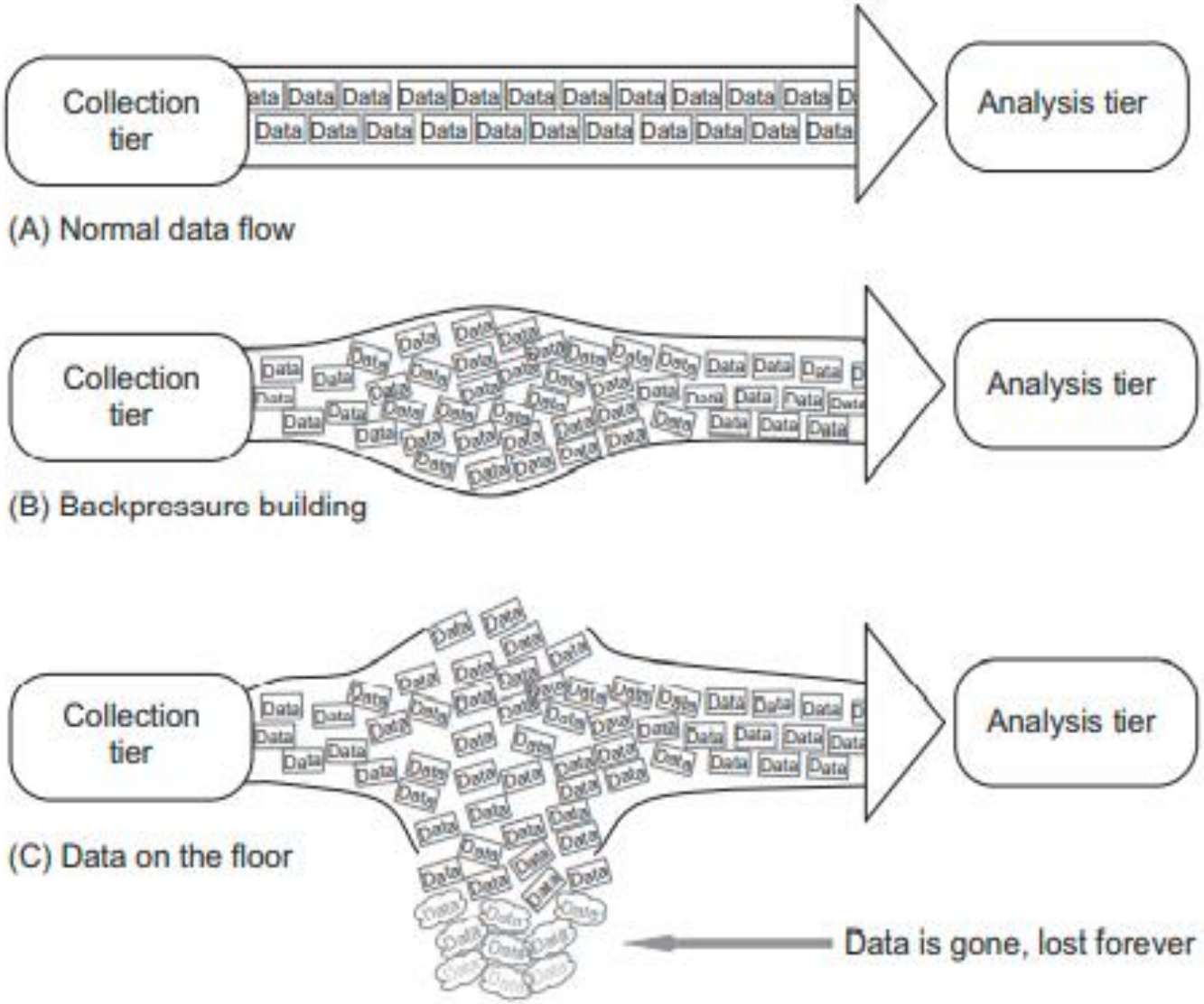
Message Queuing : Line of messages in order to communicate between two systems

Apache Kafka is a distributed stream processing software developed by LinkedIn and written in Scala and Java.

Why We need Message queuing tier?

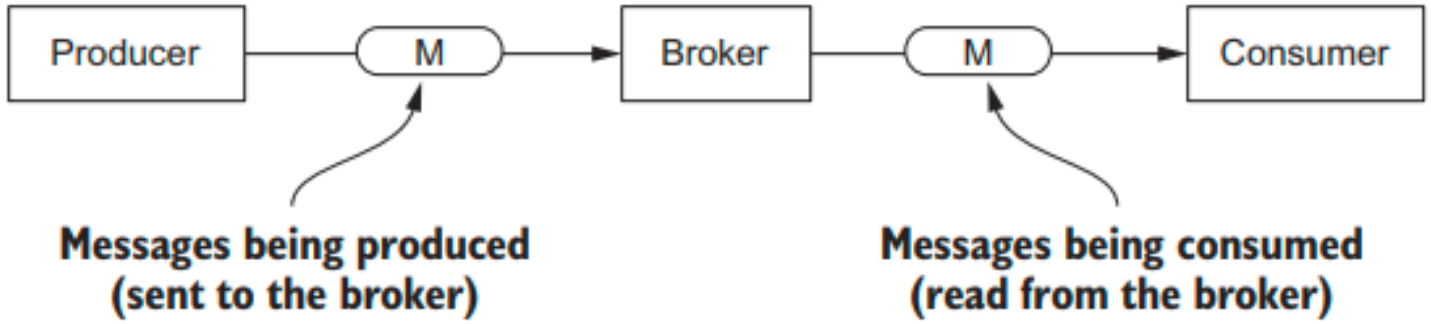
- ✓ you may be able to bring up an entire streaming system on a single machine and have each layer directly call the next, but your streaming system will span across many machines.
- ✓ When designing a software system, one desirable quality to strive for is the decoupling of the various components. With a streaming system we want the same: to decouple the components in each tier—but more importantly, decoupling the data pipeline from each other.
- ✓ At the heart of a streaming system, like any distributed system, is the communication between the numerous machines that compose the system.
- ✓ By adopting this model, our collection tier will be decoupled from our analytics tier.
- ✓ This decoupling allows our tiers to work at a higher level of abstraction, by passing messages and not having explicit calls to the next layer.
- ✓ These are two good properties to have in any system, let alone a distributed streaming one.

Why We need Message queuing tier?



Core Concept of Message Queuing

broker may manage multiple queues.

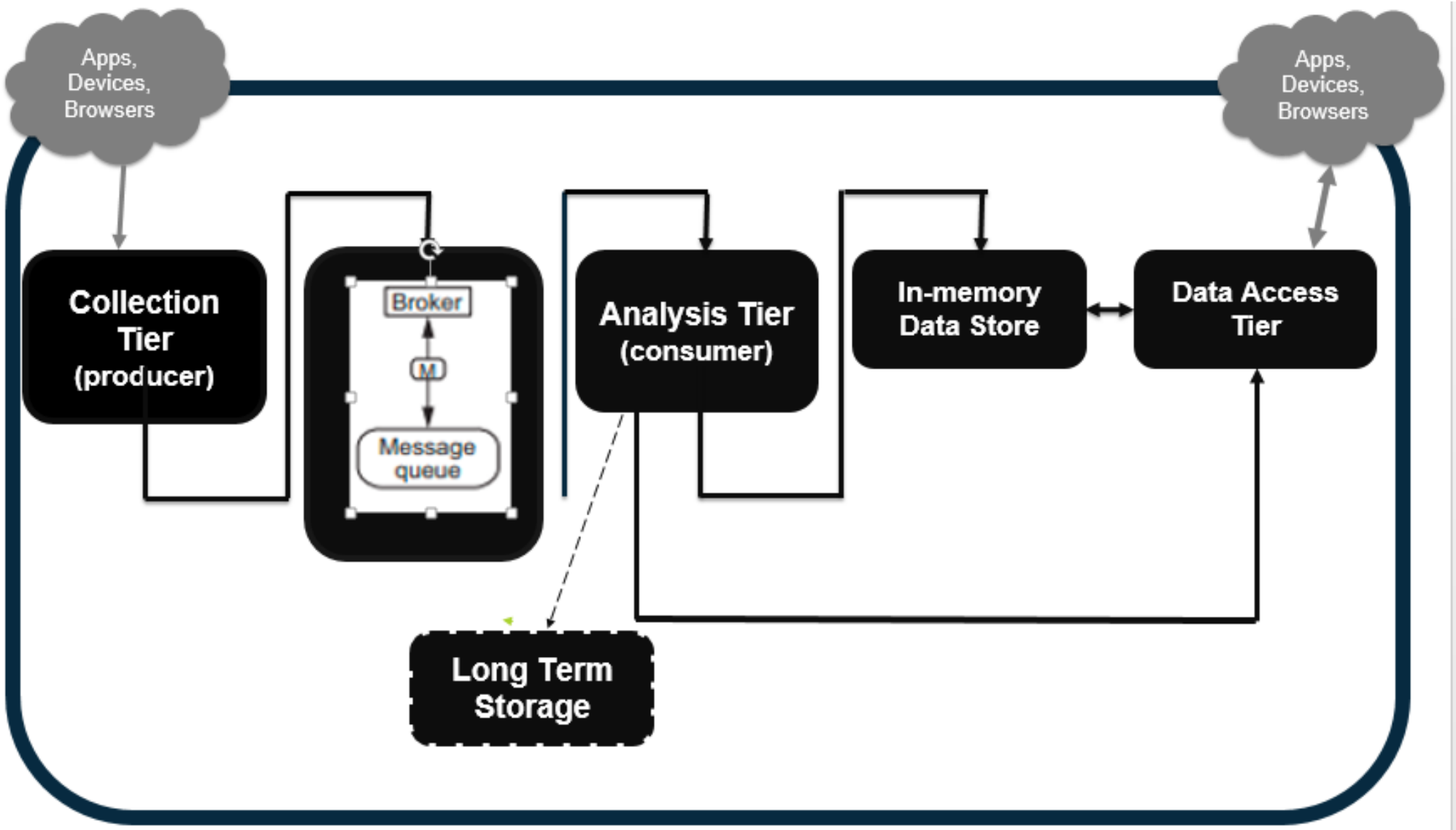


The producer sends a message to a broker
The broker puts the message into a queue
The consumer reads the message from the broker

Decoupling producers and Consumers

- Producers and consumers are decoupled
- Slow consumers do not affect producers
- Add consumers without affecting Producers
- Failure of Consumer does not affect system

Message Queuing in the Architecture



Type of Message Delivery

At most once—A message may get lost, but it will never be reread by a consumer.

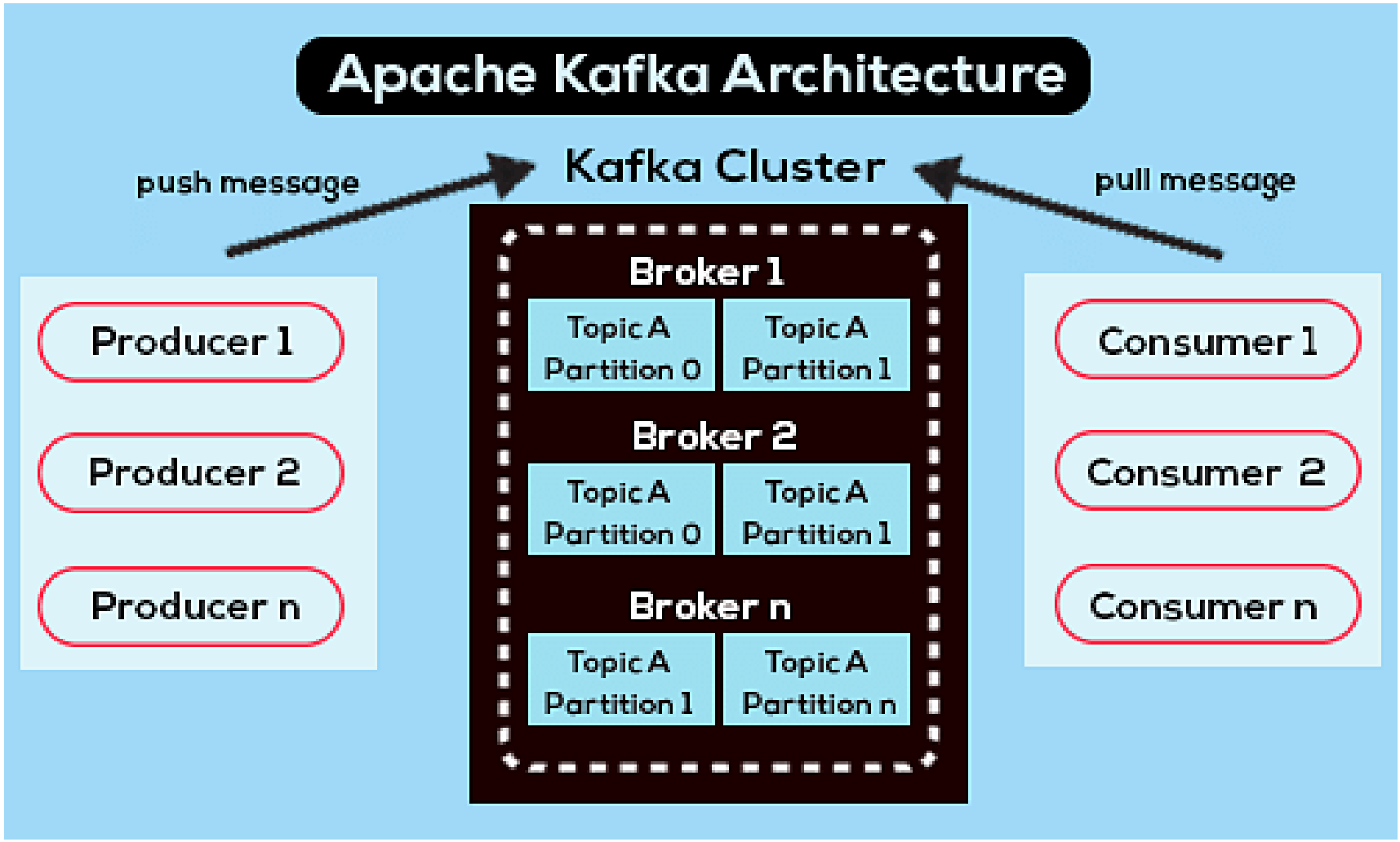
At least once—A message will never be lost, but it may be reread by a consumer.

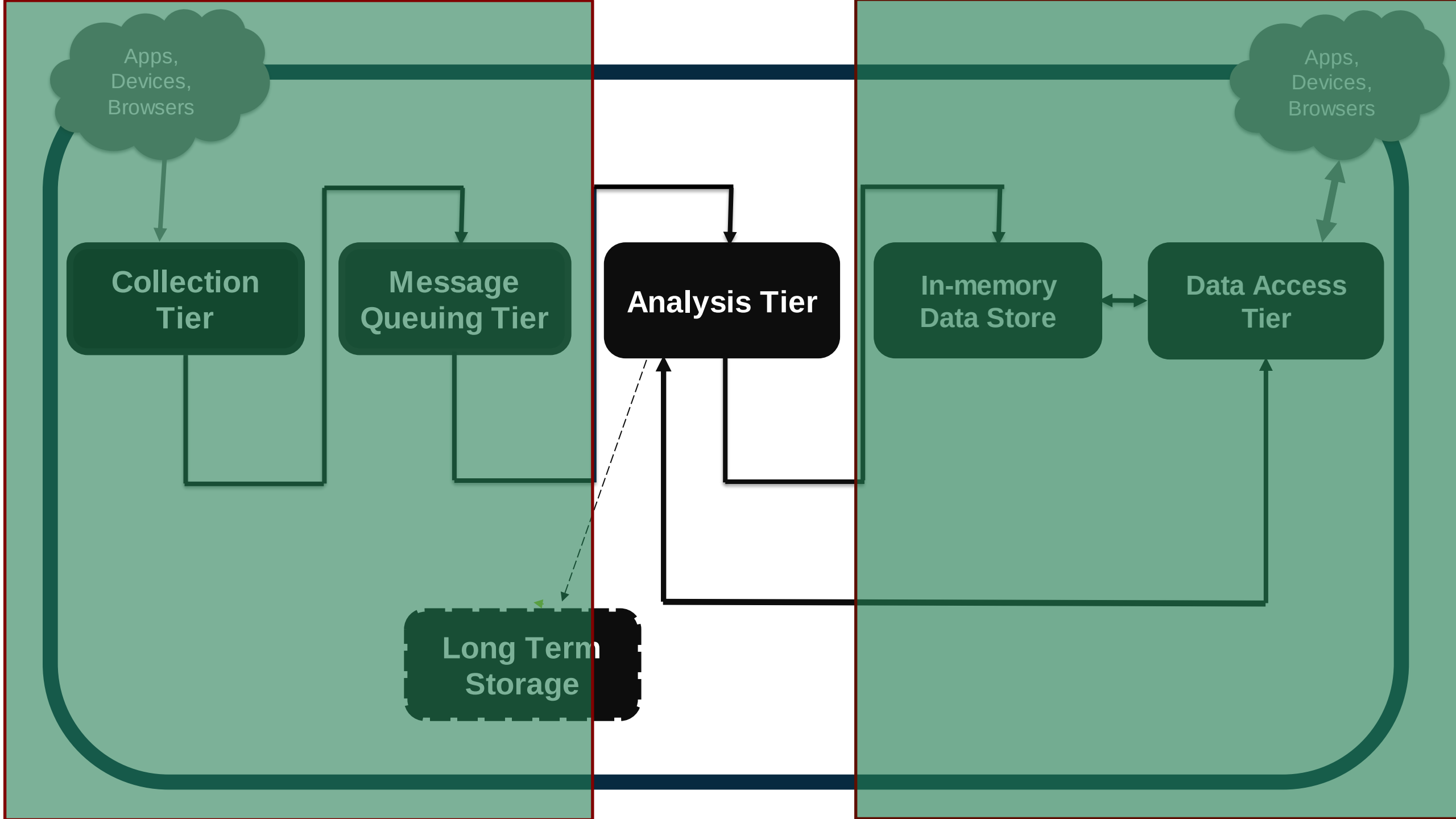
Exactly-once—A message is never lost and is read by a consumer once and only once

What is the business?

What would be the impact to fraud detection business if the communication between collection tier and analysis tier interrupted for period of time?	Prevention: Detection:
How many days of worth of data can the system tolerate losing?	Prevention: Detection:
Do the system need historical data?	Prevention: Detection:
What type of message delivery semantics does the system need?	Prevention: Detection:

KAFKA


Apache Kafka Architecture
push message
Kafka Cluster
pull message
Producer 1
Producer 2
Producer n
Broker 1
Topic A Partition 0
Topic A Partition 1
Broker 2
Topic A Partition 0
Topic A Partition 1
Broker n
Topic A Partition 1
Topic A Partition n
Consumer 1
Consumer 2
Consumer n


Apps,
Devices,
Browsers
Collection Tier
Message Queuing Tier
Analysis Tier
In-memory Data Store
Data Access Tier
Apps,
Devices,
Browsers
Long Term Storage

PyData

Analysis Tier

1 Gather the data from your site

2 Load the data into the DBMS

3 Execute a query to determine if the link is broken or the article is trending

4 Take action

5 Rinse and repeat every x minutes or, more likely, hours

---

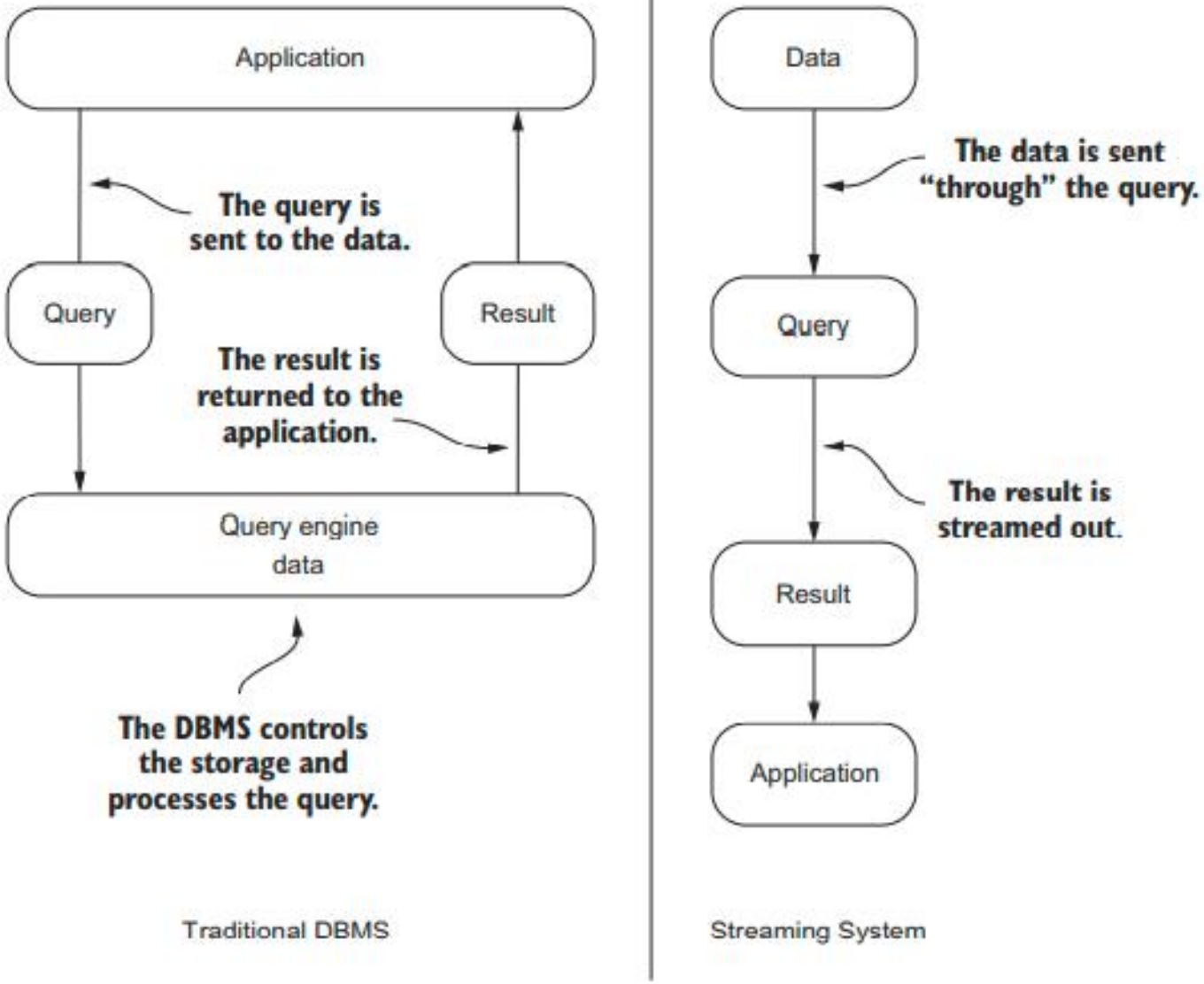
1 Collect the stream of data

2 Start a query that determines if the link is broken or the article is trending

3 Take action

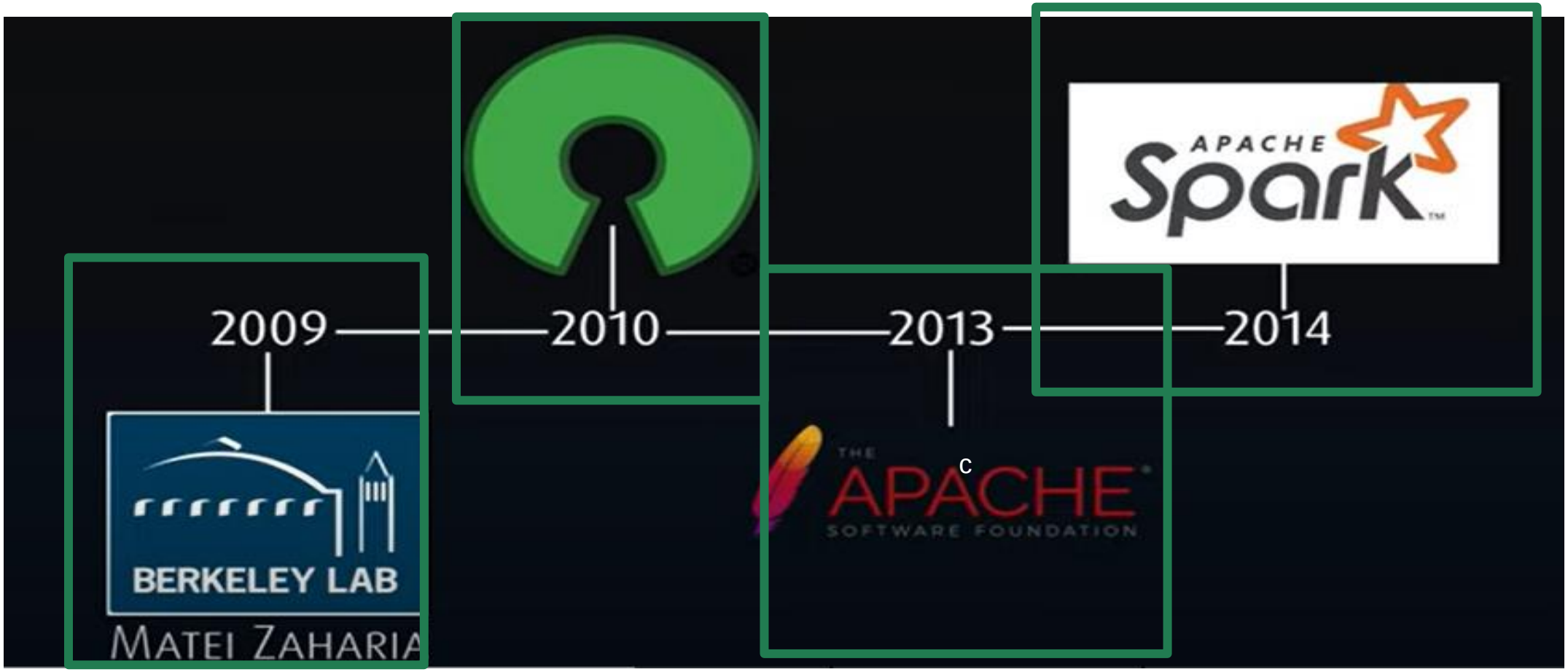
PyData

Analysis Tier



PyData

Spark History

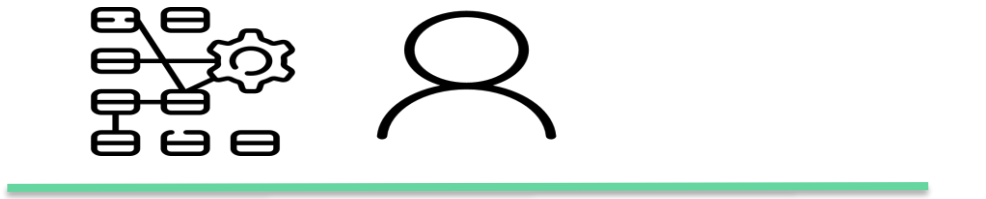


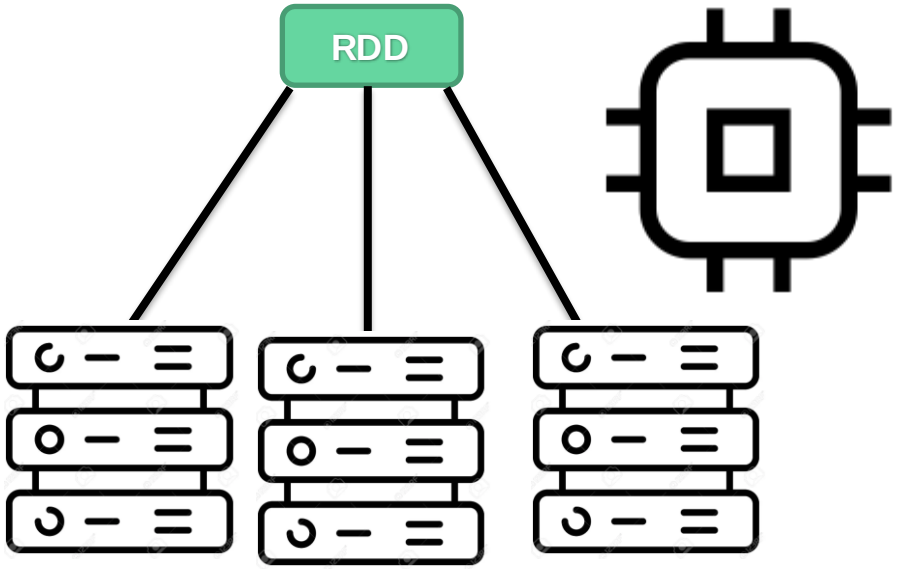
2009 Developed in Berkley Lab
2010 became Open Source
2013 was given to Apache Project
2014 became top level project

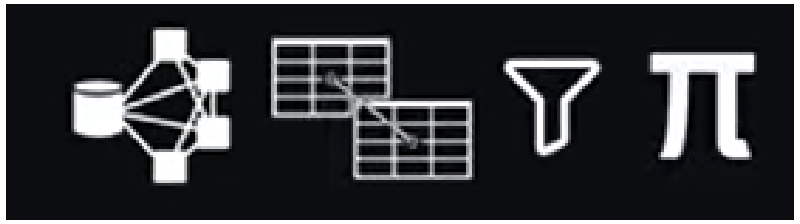
PyData

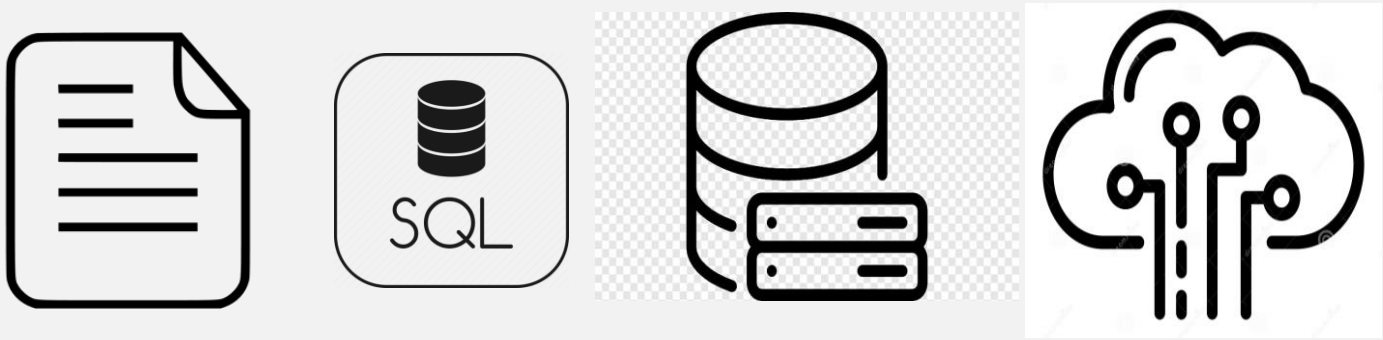
Spark RDD

Hides complexity from user



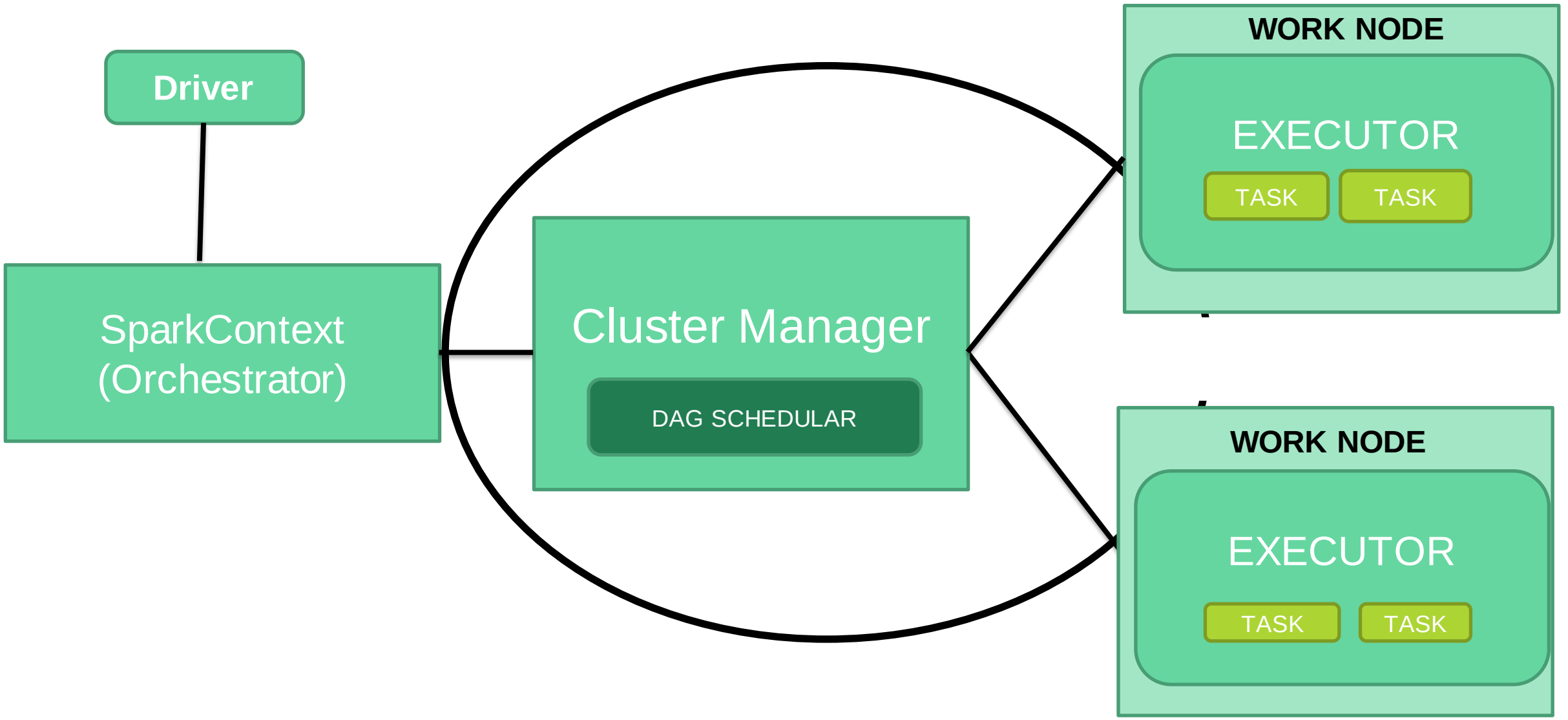






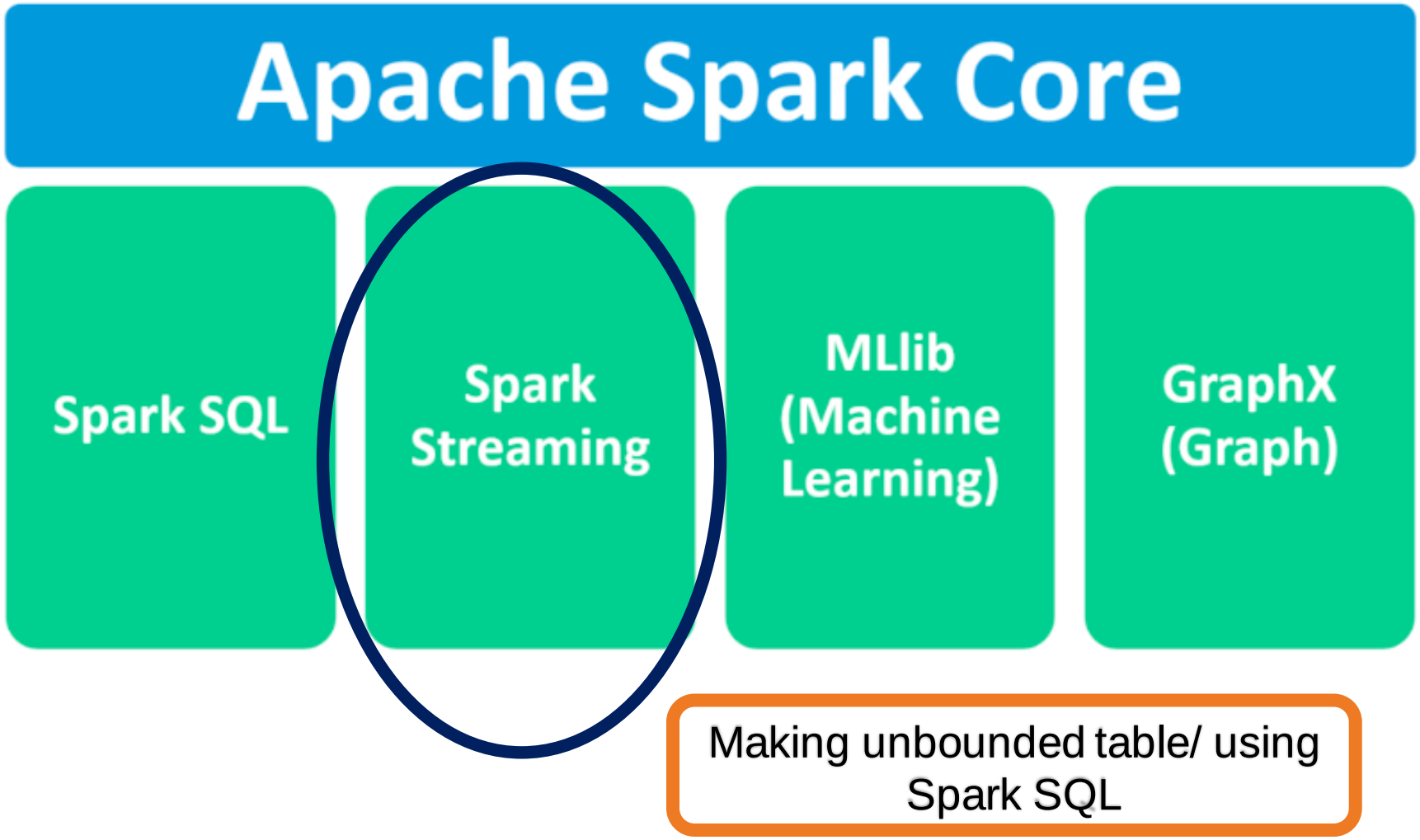
PyData

Spark Core



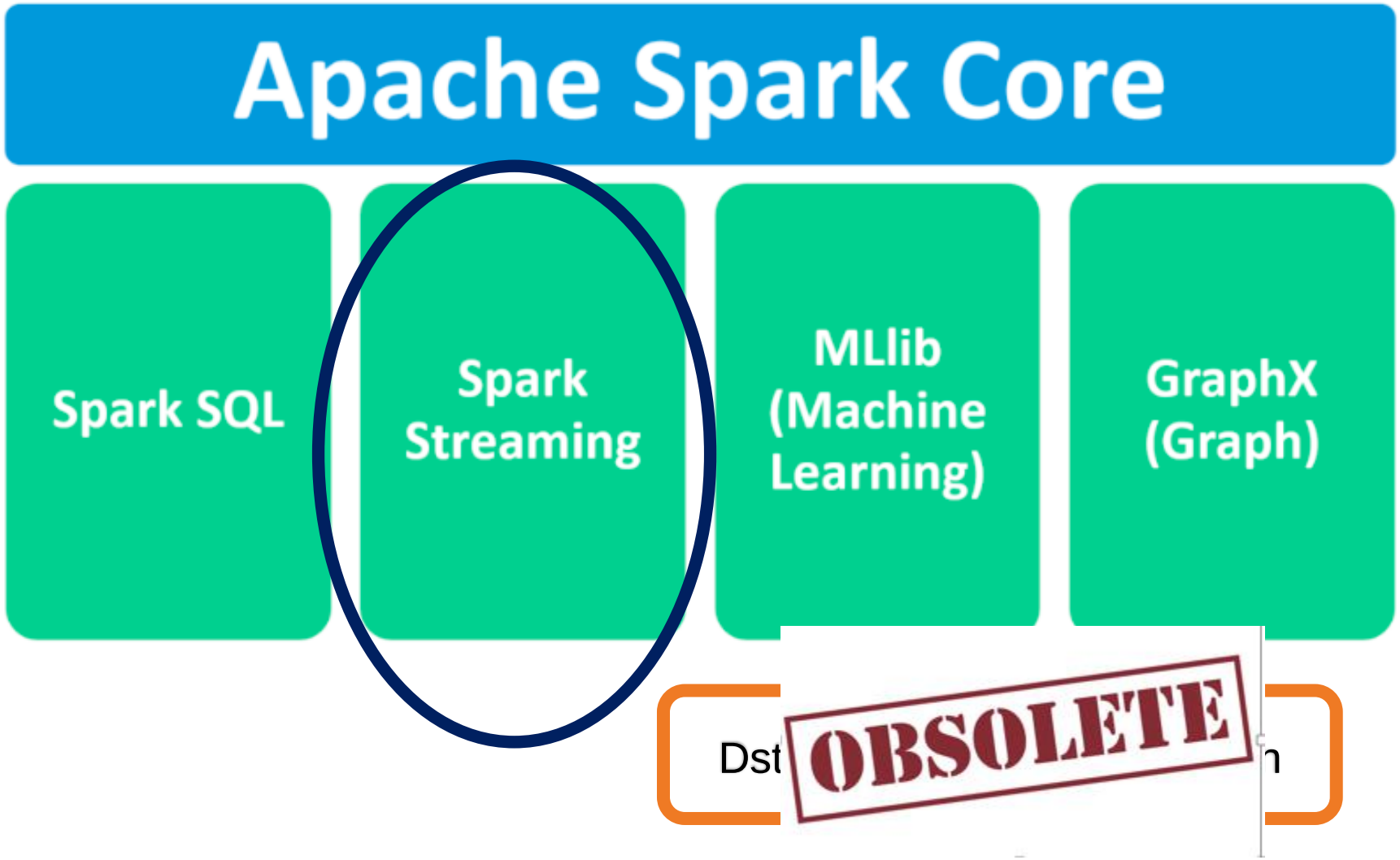
PyData

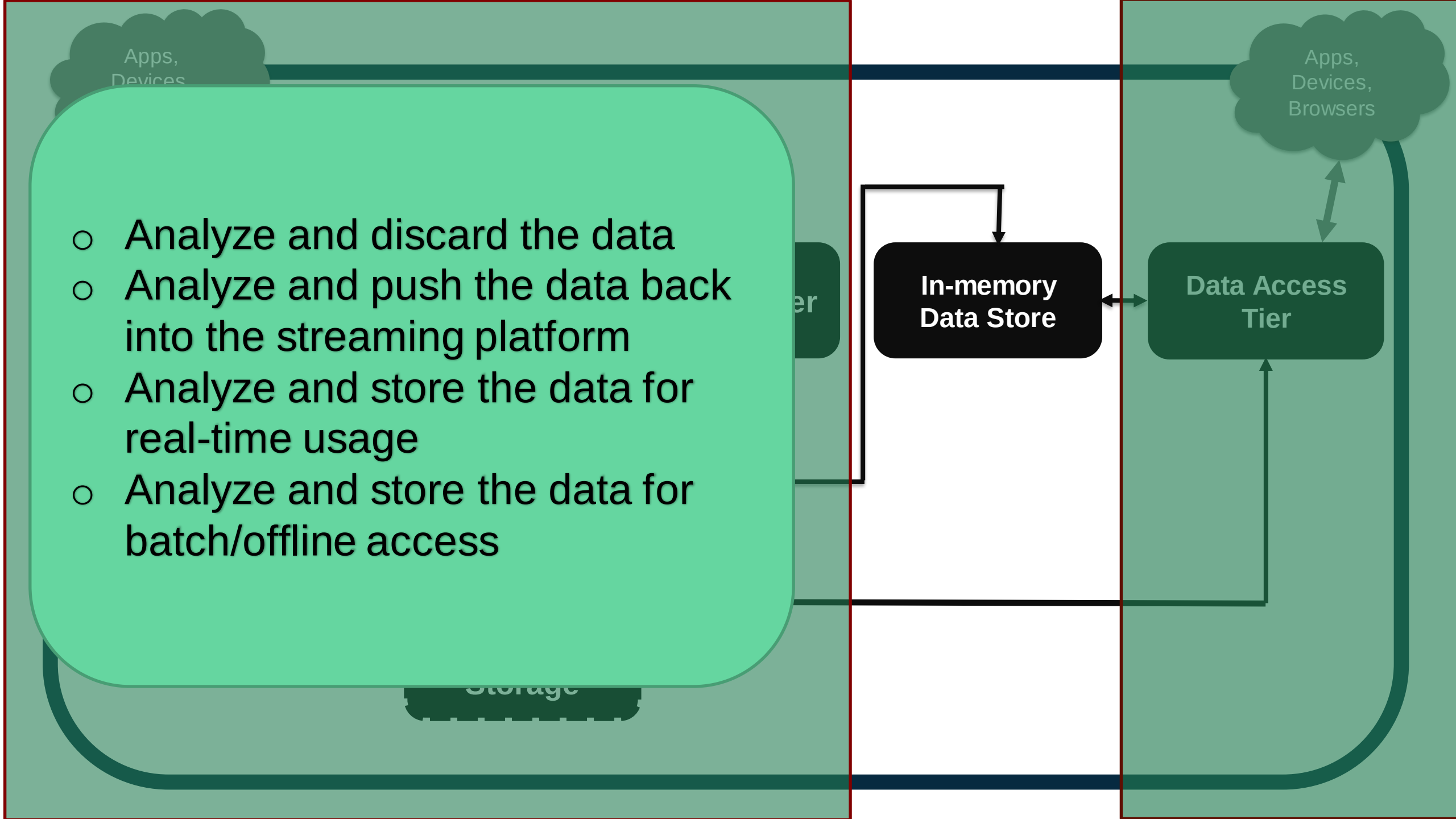
Spark Components



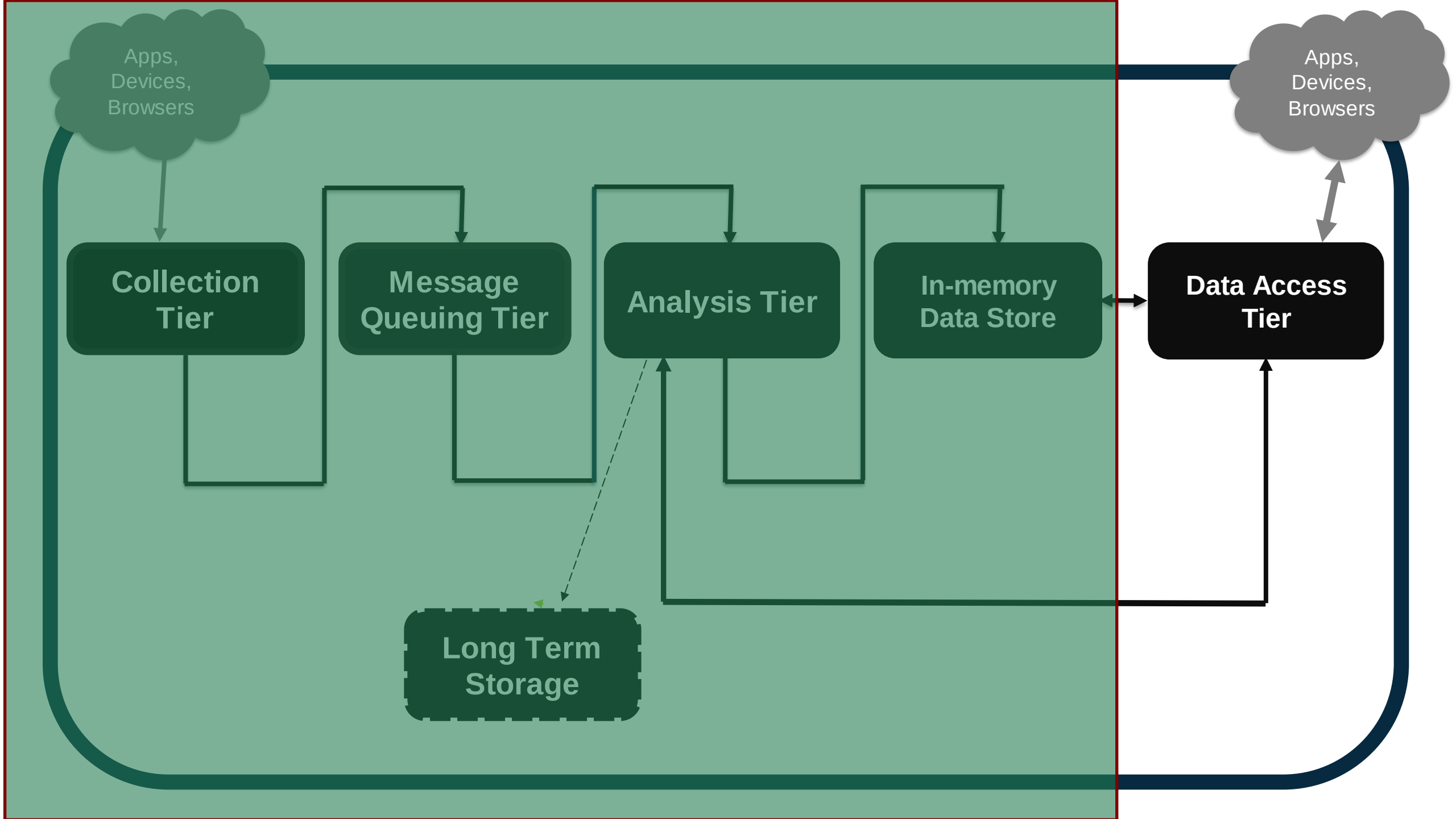
PyData

Spark Components





- Analyze and discard the data
- Analyze and push the data back into the streaming platform
- Analyze and store the data for real-time usage
- Analyze and store the data for batch/offline access


Apps,
Devices,
Browsers
Collection Tier
Message Queuing Tier
Analysis Tier
In-memory Data Store
Data Access Tier
Long Term Storage
Apps,
Devices,
Browsers