Intro to Data Stream Distributed Processing Platforms

By: Shahab Safaee
Big Data Researcher
Email: safaee.shx@gmail.com
shahabsafaee
@shahabsafaee
@shahabsafaee.ir

Agenda

- Data Stream Concepts
  - What is Data Stream?
  - Data Stream Applications
  - Data Stream Challenges
  - Data Stream Processing Model & Architecture
  - DSMS vs DBMS
- Data Stream Distributed Processing Platforms
  - Big Data Technology Stack
  - Type of Processing Systems
  - Storm
  - Spark
  - Kafka
- Running Example With Spark

What is Streaming

- Streaming is unstructured data that is generated continuously by thousands of data sources.
- This Streaming data includes a wide variety of data such as
  - log files generated by customers using your mobile or web applications, in-game player activity, information from social networks, Financial trading and telemetry from connected devices or instrumentation in data centers.

Data Streams - Terms

- A data stream is a (potentially unbounded) sequence of tuples
- Each tuple consist of a set of attributes, similar to a row in database table
- Transactional data streams: log interactions between entities
  - Credit card: purchases by consumers from merchants
  - Telecommunications: phone calls by callers to dialed parties
  - Web: accesses by clients of resources at servers
- Measurement data streams: monitor evolution of entity states
  - Sensor networks: physical phenomena, road traffic
  - IP network: traffic at router interfaces
  - Earth climate: temperature, moisture at weather stations

Data Models

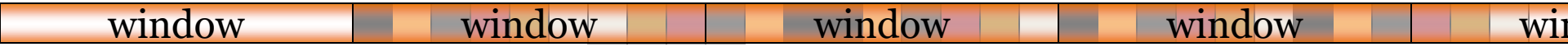
- Real-time data stream: sequence of items that arrive in some order and may only be seen once.
- Stream items: like relational tuples
  - Relation-based: e.g., STREAM, TelegraphCQ and Borealis
  - Object-based: e.g., COUGAR, Tribecca
- Window models
  - Direction of movements of the endpoints: fixed window, sliding window, landmark window
  - Time-based vs. Tuple-based
  - Update interval: eager (for each new arriving), lazy (batch processing), non-overlapping tumbling windows.

More on Windows

- Mechanism for extracting a finite relation from an infinite stream

Sliding:

Jumping:



Overlapping

Timestamps

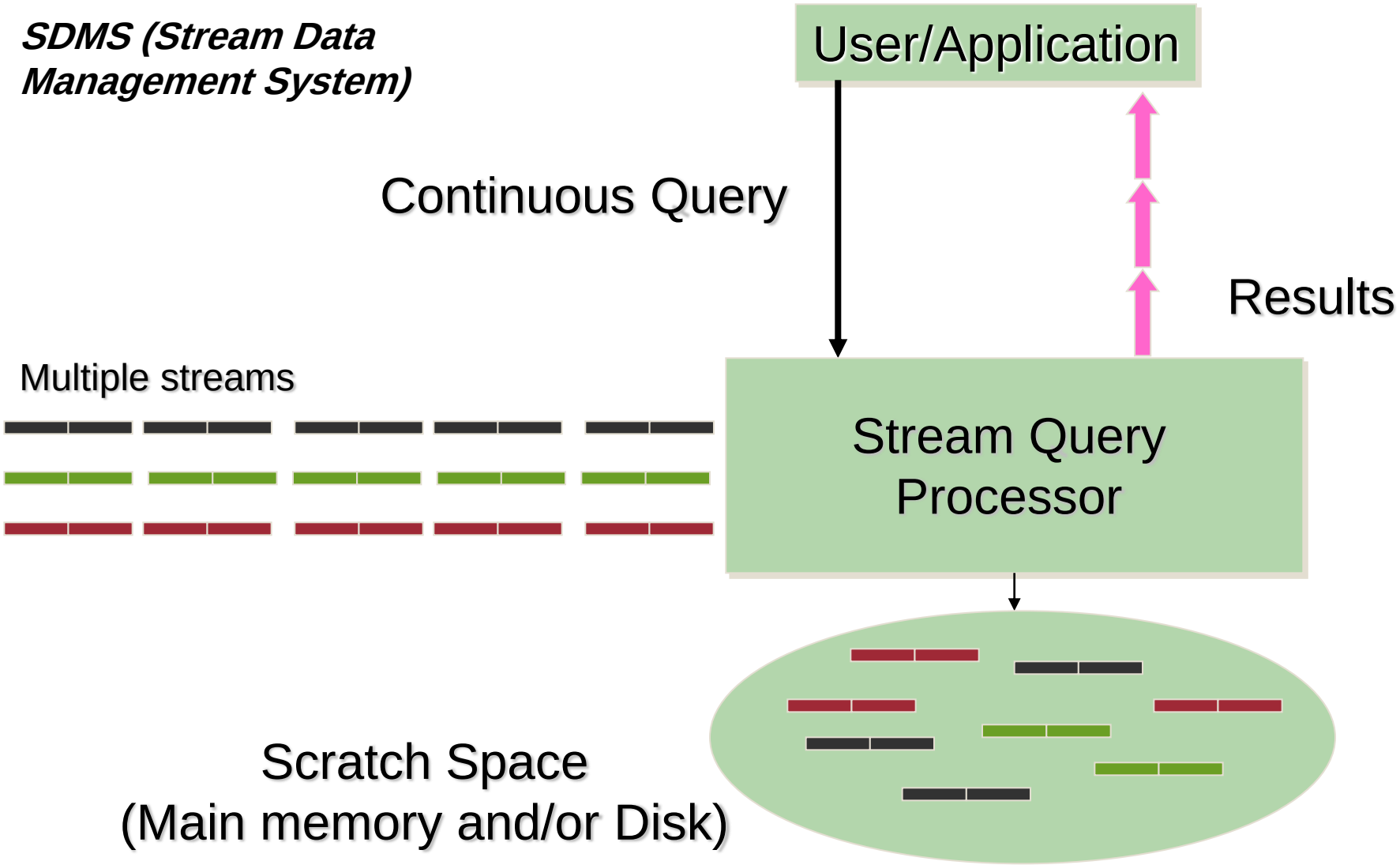
- Used for tuple ordering and by the DSMS for defining window sizes (time-based)
- Useful for the user to know when the tuple originated
- Explicit:
  - set by the source of data
- Implicit:
  - set by DSMS, when it has arrived
- Ordering is an issue
- Distributed systems:
  - no exact notion of time

Characteristics of Data Streams

- Data Streams Model:
  - Data enters at a high speed rate
  - The system cannot store the entire stream, but only a small fraction
  - How do you make critical calculations about the stream using a limited amount of memory?
- Characteristics
  - Huge volumes of continuous data, possibly infinite
  - Fast changing and requires fast, real-time response
  - Random access is expensive—single scan algorithms(can only have one look)

Architecture: Stream Query Processing

SDMS (Stream Data Management System)



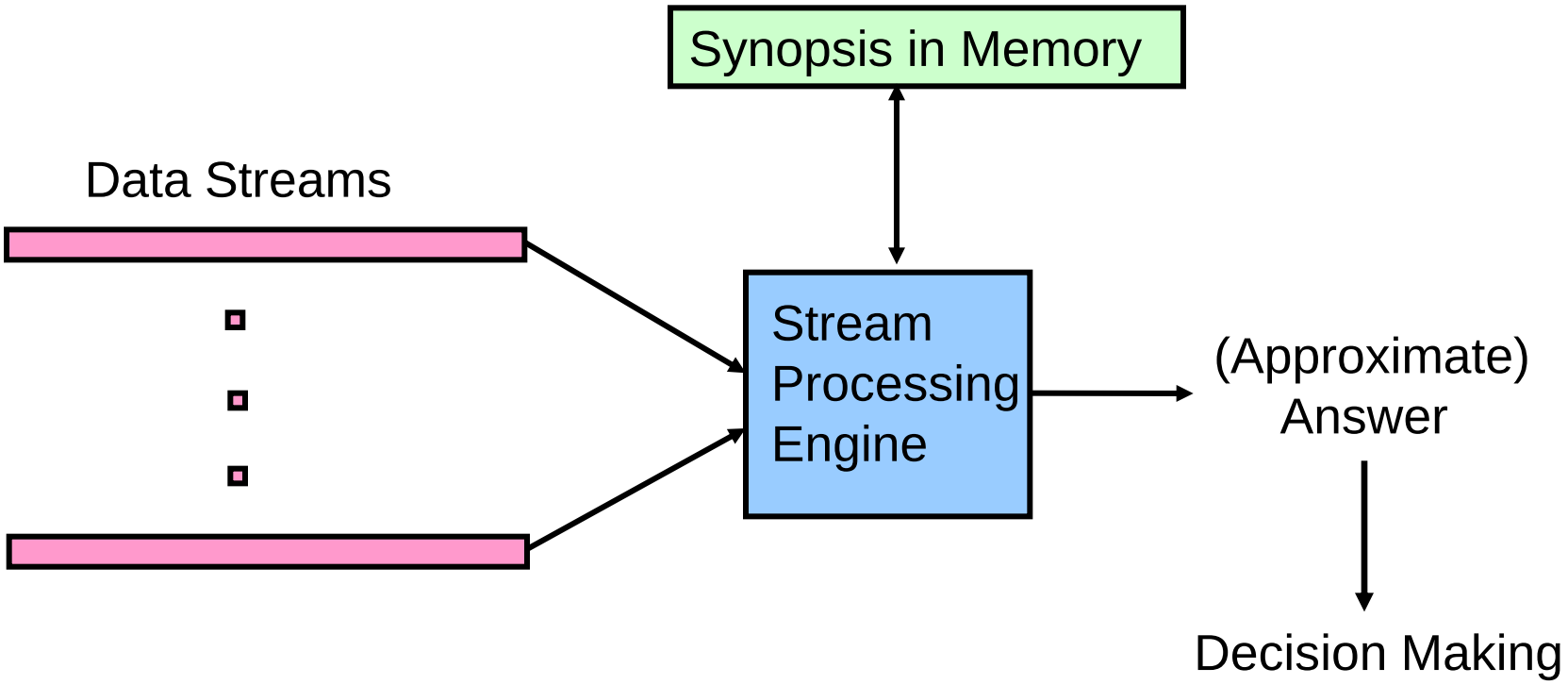
Stream Data Applications

- Telecommunication calling records
- Business: credit card transaction flows
- Network monitoring and traffic engineering
- Financial market: stock exchange
- Engineering & industrial processes: power supply & manufacturing
- Sensor, monitoring & surveillance: video streams, RFIDs
- Web logs and Web page click streams
- Massive data sets (even saved but random access is too expensive)

Problems/Challenges

- Zillions of data
  - Continuous/Unbounded
  - Examples arrive faster than they can be mined
  - Application may require fast, real-time response
- Time/Space constrained
  - Not enough memory
  - Can't afford storing/revisiting the data
    - Single pass computation
  - External memory algorithms for handling data sets larger than main memory cannot be used.
    - Do not support continuous queries
    - Too slow real-time response

Computation Model



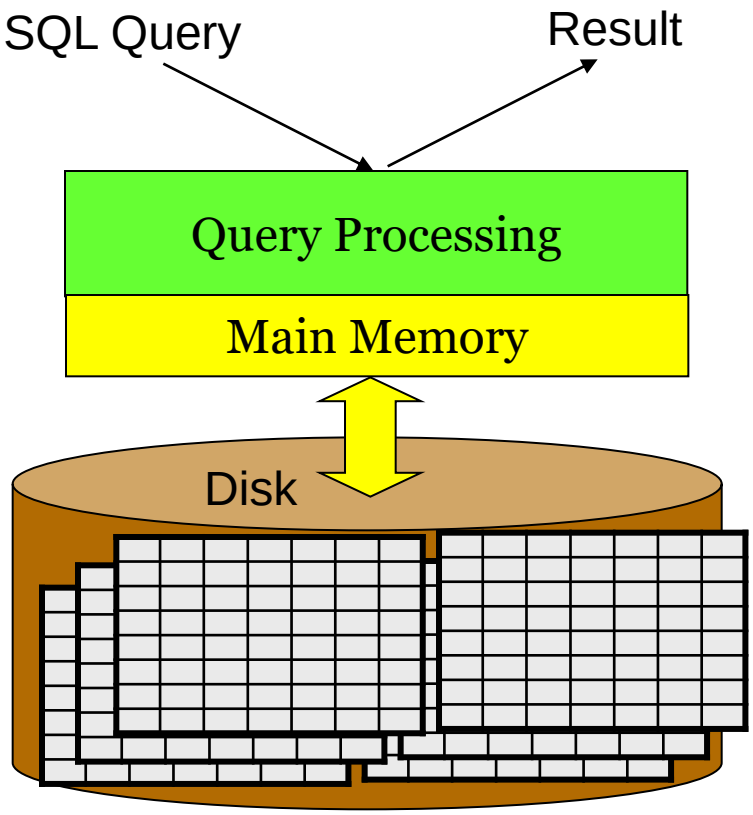
Model Components

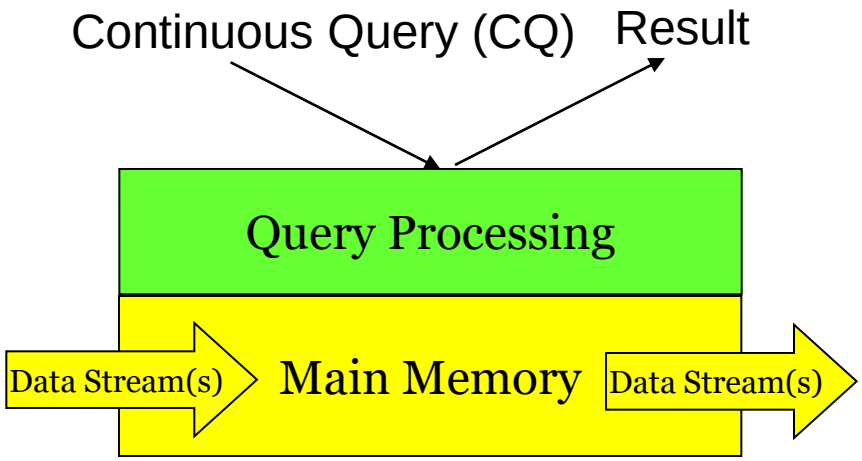
- Synopsis
  - Summary of the data
    - Samples, …
- Processing Engine
  - Implementation/Management System
    - STREAM (Stanford): general-purpose
    - Aurora (Brown/MIT): sensor monitoring, dataflow
    - Telegraph (Berkeley): adaptive engine for sensors
- Decision Making

DSMS vs DBMS (1)

- Traditional DBMS:
  - data stored in finite, persistent data sets
- Data Streams:
  - distributed, continuous, unbounded, rapid, time varying, noisy, . .
- Data-Stream Management: variety of modern applications
  - Network monitoring and traffic engineering
  - Sensor networks
  - Telecom call-detail records
  - Network security
  - Financial applications
  - Manufacturing processes
  - Web logs and clickstreams
  - Other massive data sets…

DSMS vs DBMS (2)





DSMS vs DBMS (3)

Feature	DBMS	DSMS
Model	Persistent Data	Transient Data
Table	Set or Bag of Tuples	Infinite Sequence of Tuples
Updates	All	Append Only
Queries	Transient	Persistent
Query Answers	Exact	Often Approximate
Query Evaluation	Multi-Pass	One Pass
Operators	Blocking and Non-Blocking	Non-Blocking
Query Plan	Fixed	Adaptive
Data Processing	Synchronous	Asynchronous
Concurrency	High	Low

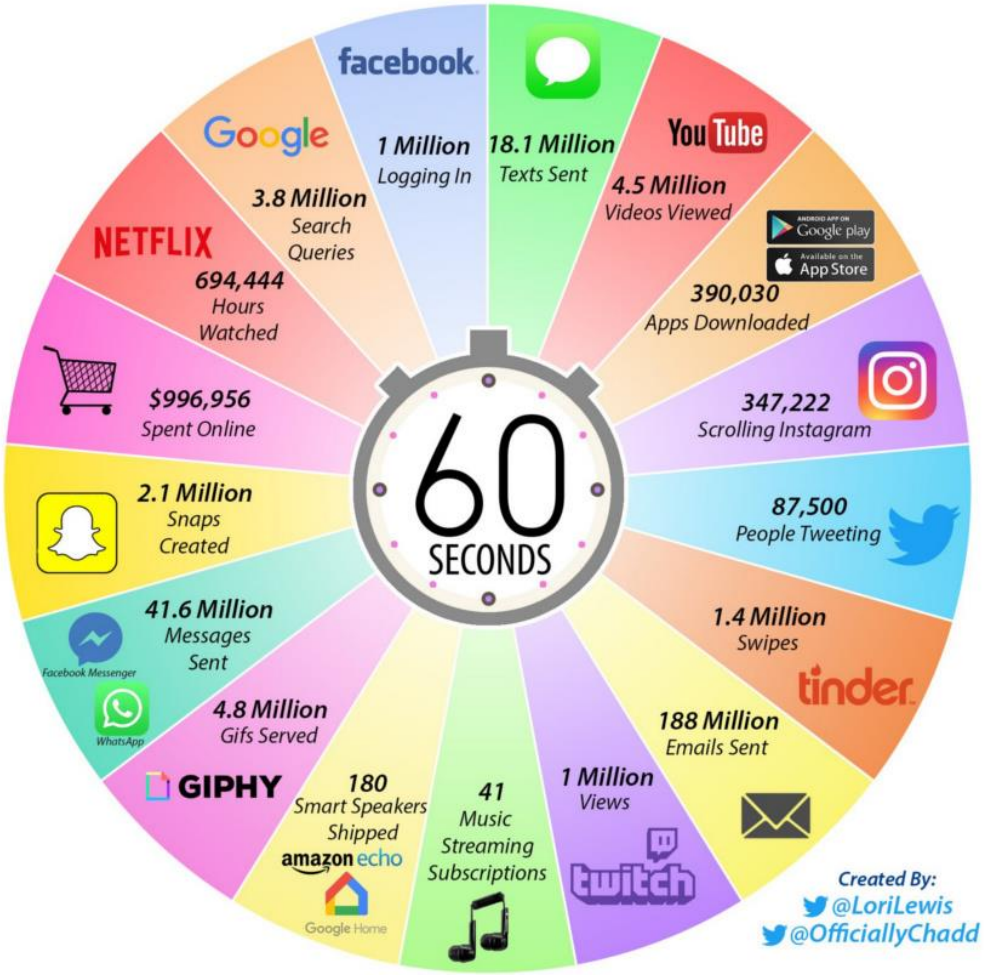
Queries

- DBMS:
  - one-time (transient) queries
- DSMS:
  - continuous (persistent) queries
- Unbounded memory requirements
- Blocking operators: window techniques
- Queries referencing past data
- DBMS:
  - (mostly) exact query answer
- DSMS:
  - (mostly) approximate query answer

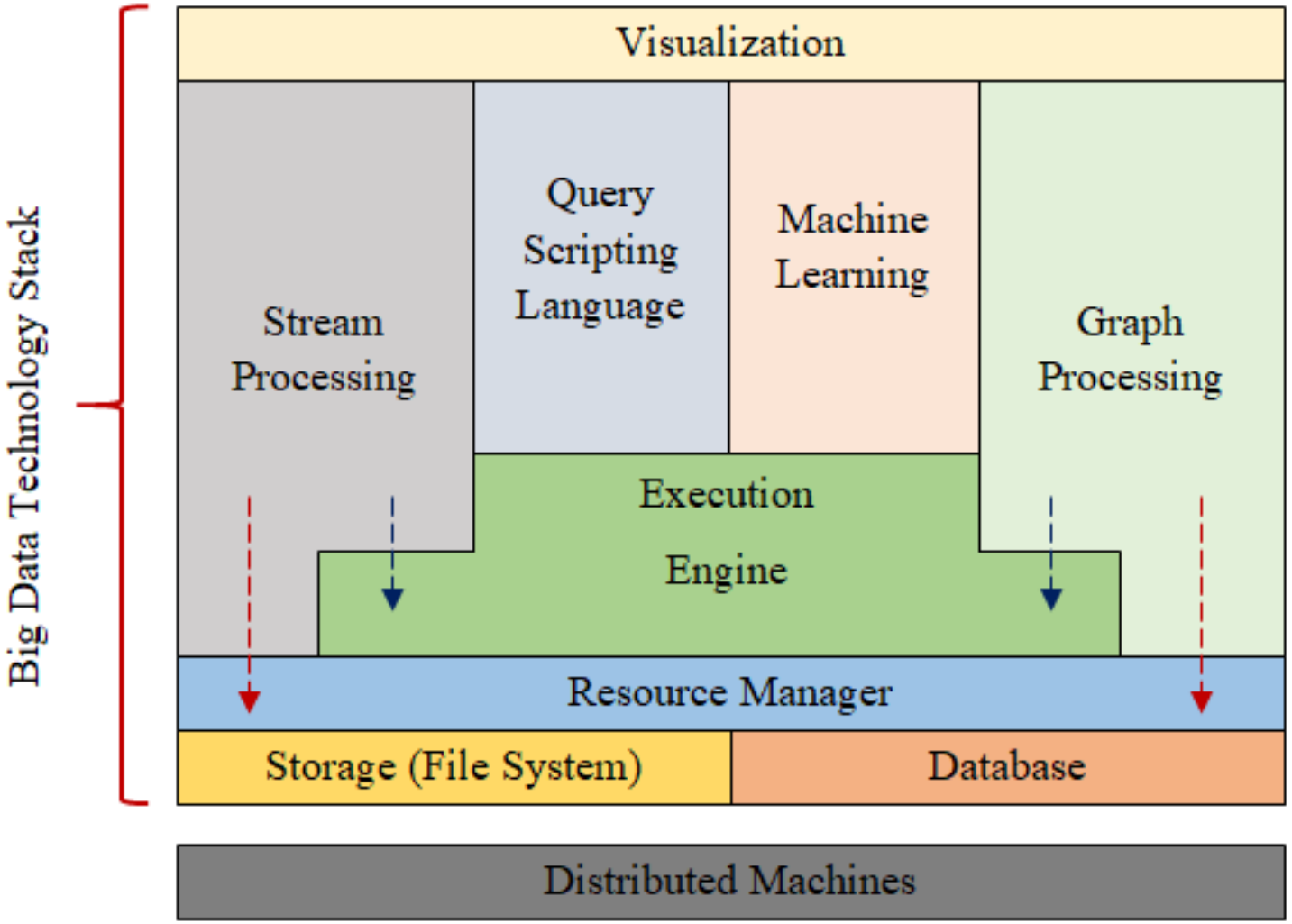
Query Languages

- Stream query language issues (windows)
- SQL-like proposals suitably extended for a stream environment:
- Query operators (selection/projection, join, aggregation)
- Examples:
  - GSQL (Gigascope)
  - CQL (STREAM)
  - EPL (ESPER)

2019 This Is What Happens In An Internet Minute



Big Data Technology Stack



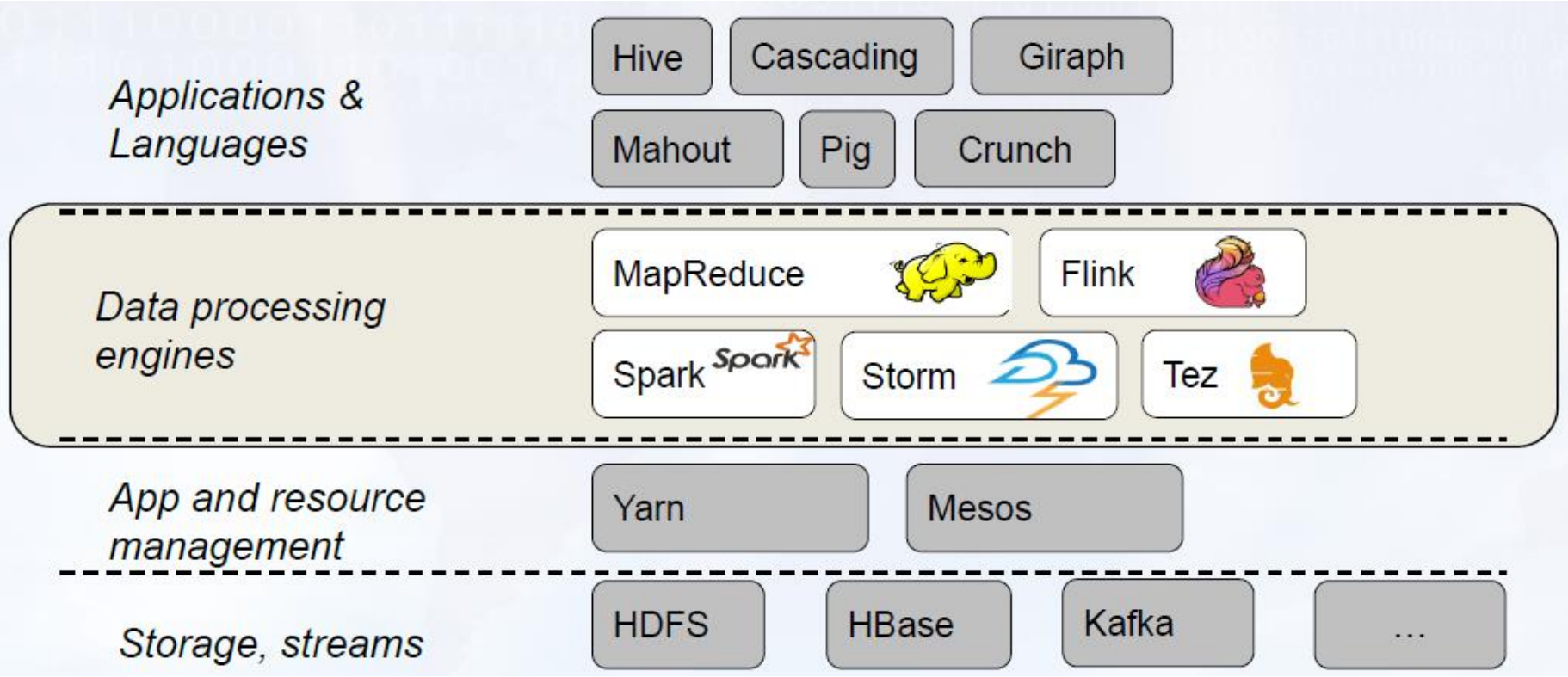
Big Data Execution Engine

- Features
  - Scalable
  - Fault Tolerant
  - Parallelism
- Infrastructure Processing
  - Commodity Clustered Machines
- Programming Models
  - A programming model is the fundamental style and interfaces for developers to write computing programs and applications.
- Instance of Execution Engines:
  - MapReduce, Spark, Stratosphere, Dryad, Hyracks, ...

Main Characteristics of Distributed Execution Engines

- The design purpose of Distributed Execution Engines is to consider the fundamental characteristics which are as follows:
  - Task Scheduling
  - Data Distribution
  - Load Balancing
  - Transparent Fault Tolerance
  - Control Flow
  - Tracking Dependencies

Big Data Processing Platforms



Type of Processing Systems (1)

- Batch Processing Systems
  - Batch processing involves operating over a large, static dataset and returning the result at a later time when the computation is complete.
- The datasets in batch processing are typically...
  - Bounded: batch datasets represent a finite collection of data
  - Persistent: data is almost always backed by some type of permanent storage
  - Large: batch operations are often the only option for processing extremely large sets of data
  - Batch processing is well-suited for calculations where access to a complete set of records is required.

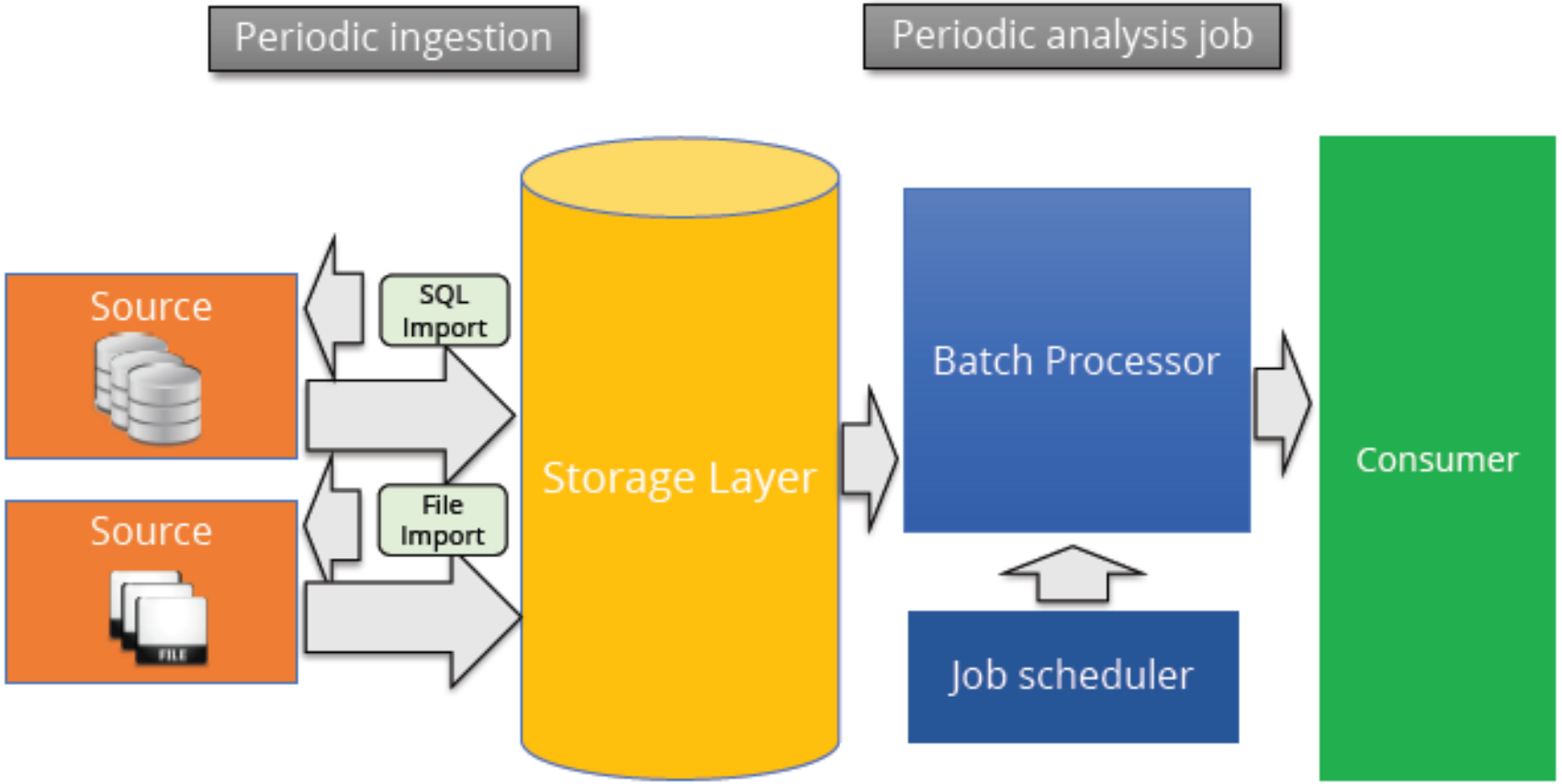
Type of Processing Systems (2)

- Stream Processing Systems
  - Stream processing systems compute over data as it enters the system.
  - Instead of defining operations to apply to an entire dataset, stream processors define operations that will be applied to each individual data item as it passes through the system.
- The datasets in stream processing are considered "unbounded". This has a few important implications:
  - The total dataset is only defined as the amount of data that has entered the system so far.
  - The working dataset is perhaps more relevant, and is limited to a single item at a time.
  - Processing is event-based and does not "end" until explicitly stopped. Results are immediately available and will be continually updated as new data arrives.

Processing Types of Big Data Processing Frameworks

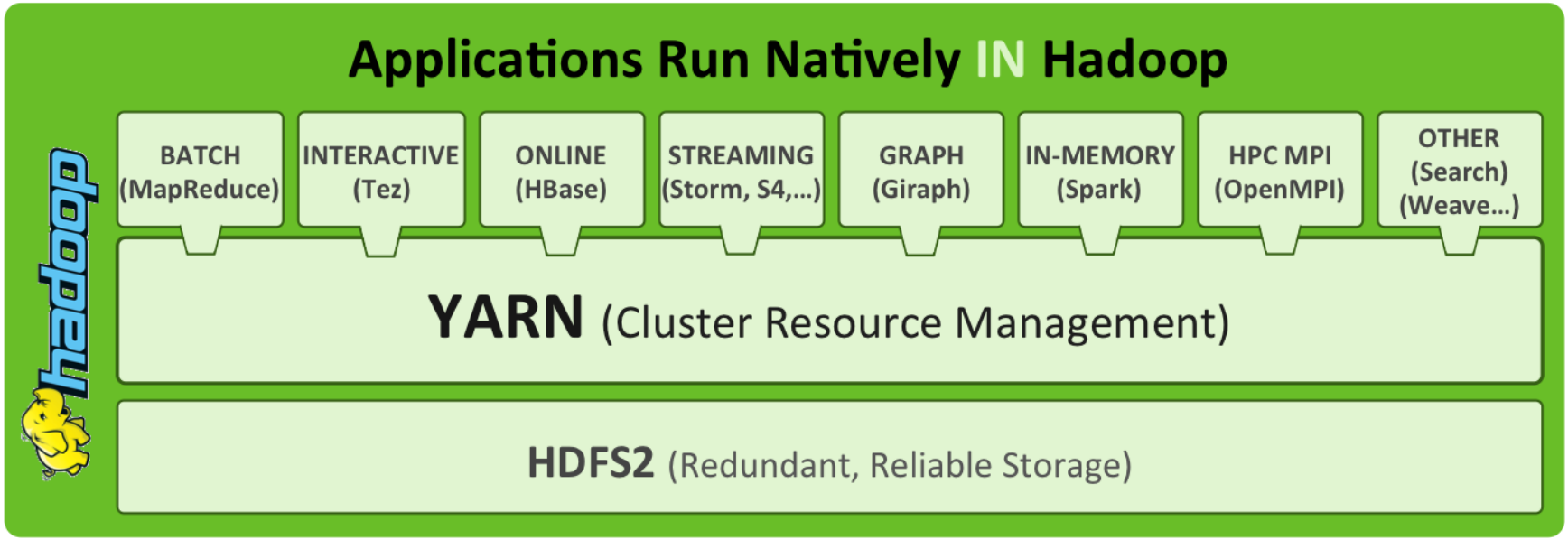
- Batch-only frameworks:
  - Apache Hadoop
- Stream-only frameworks:
  - Apache Storm
  - Apache Samza
- Hybrid frameworks:
  - Apache Spark
  - Apache Flink

Batch-only frameworks

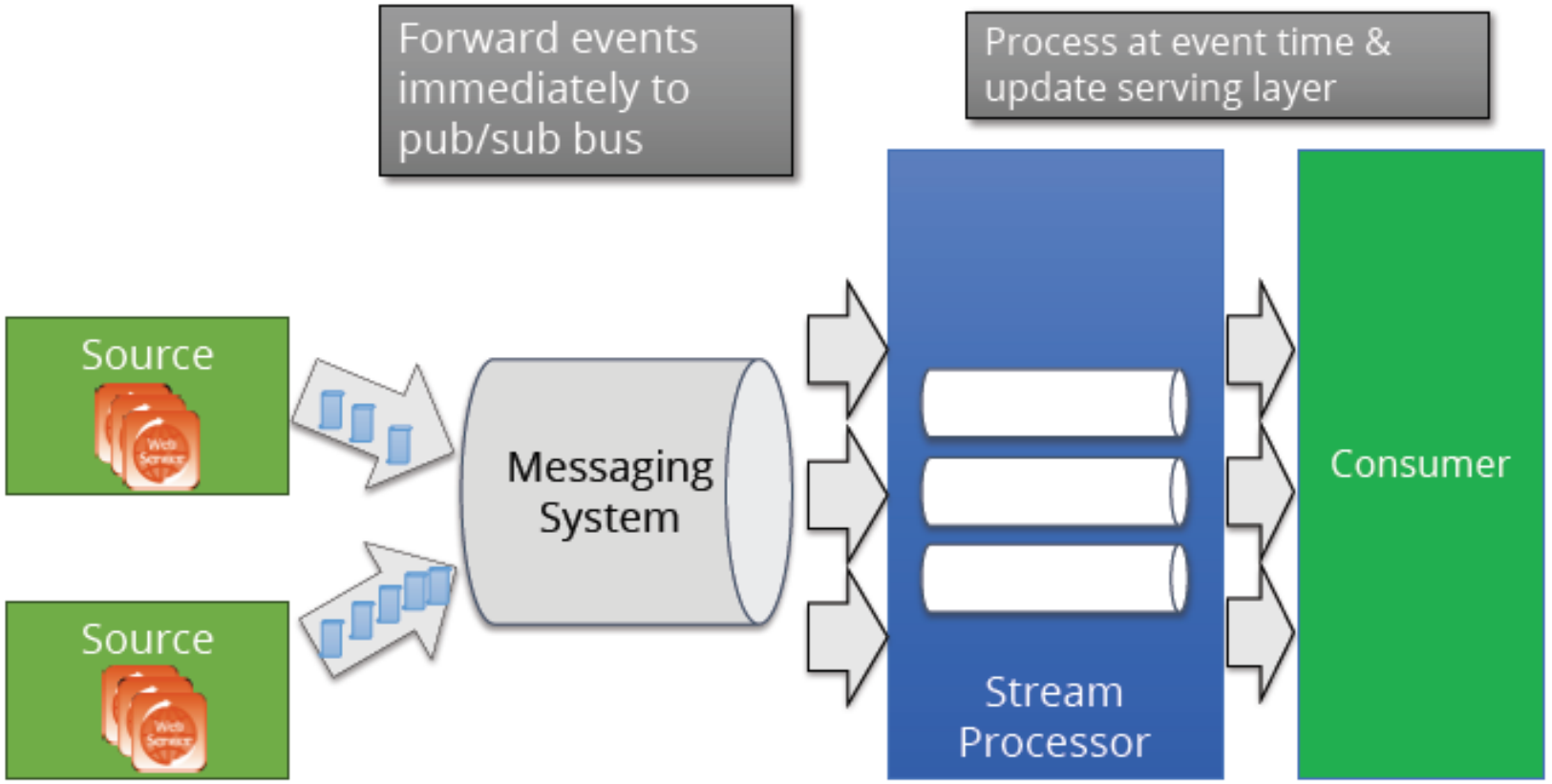


Apache Hadoop

- Apache Hadoop is a processing framework that exclusively provides batch processing.



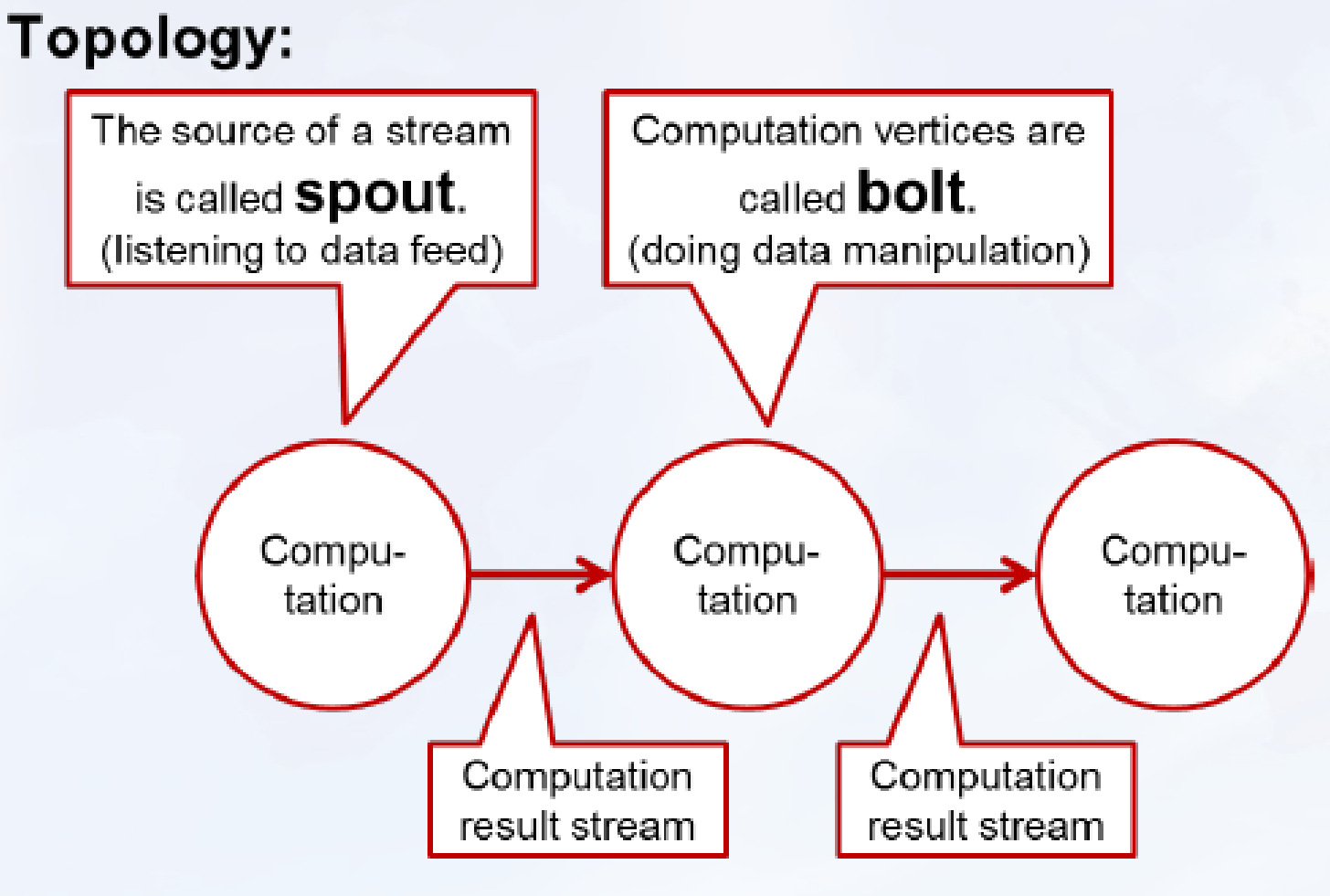
Stream-only frameworks/Kappa Architecture



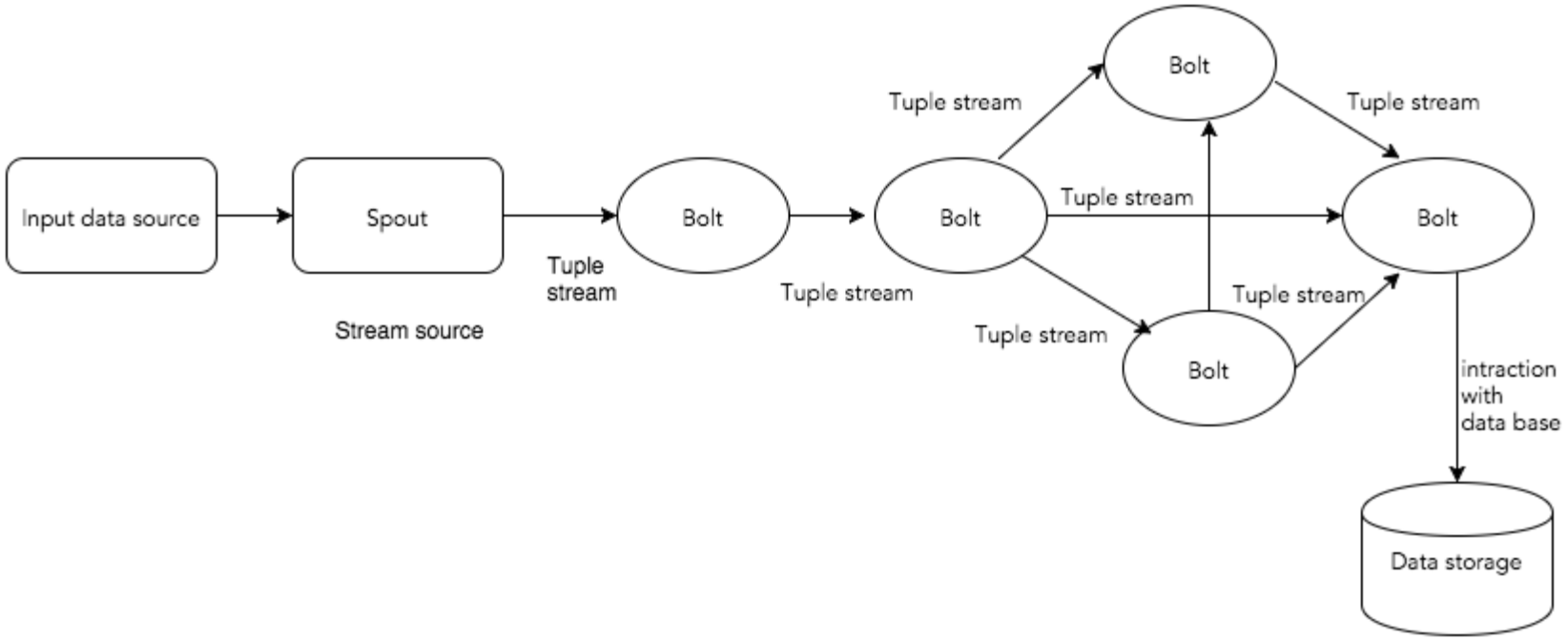
Apache Storm (1)

- Scalable Stream Processing Platform by Twitter
- Tuple wise computation
- Programs are represented in a Topology graph
  - vertices are computations / data transformations
  - edges represent data streams between the computation nodes
  - streams consist of an unbounded sequence of data-items/tuples
- Low-level stream processing engine

Apache Storm (2)



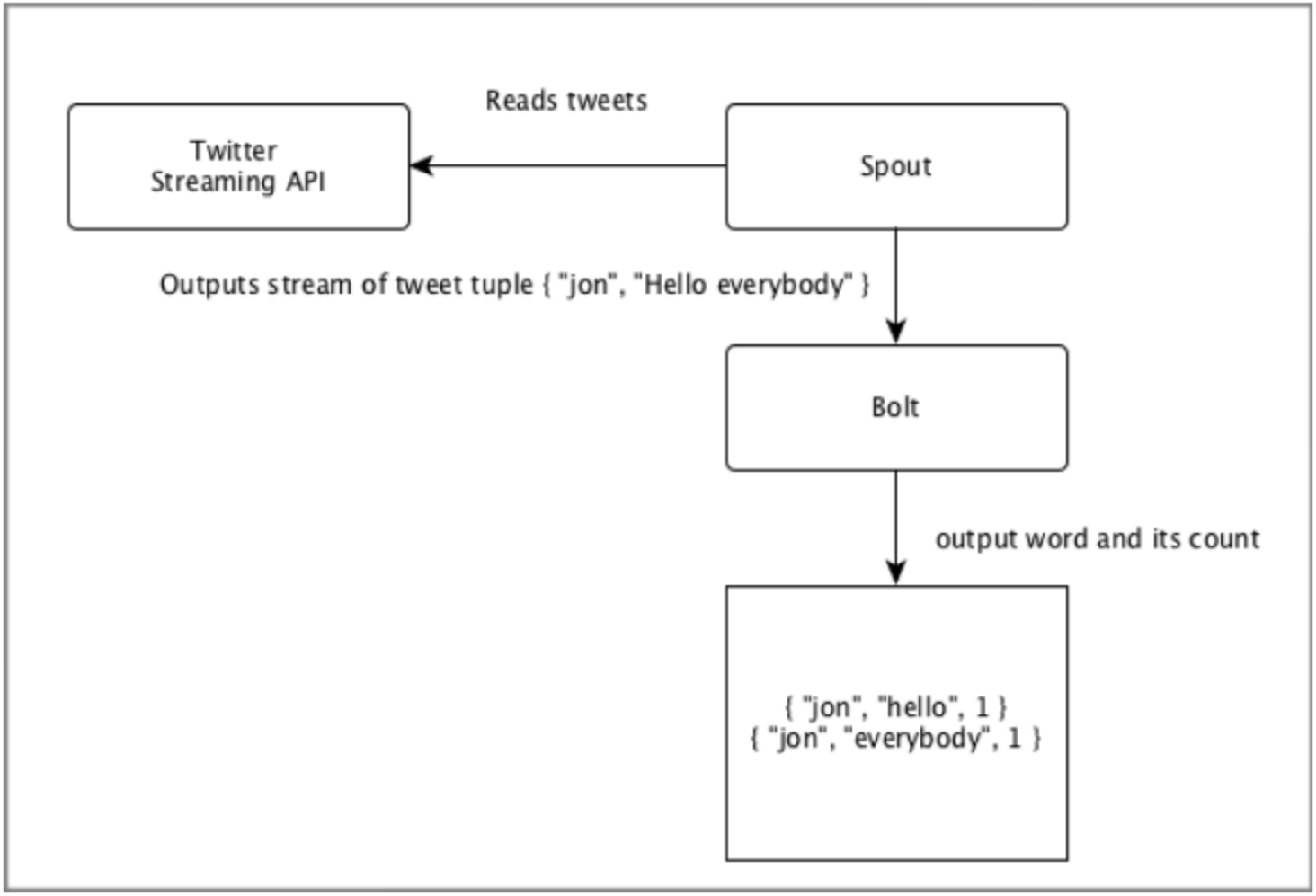
Apache Storm (3)



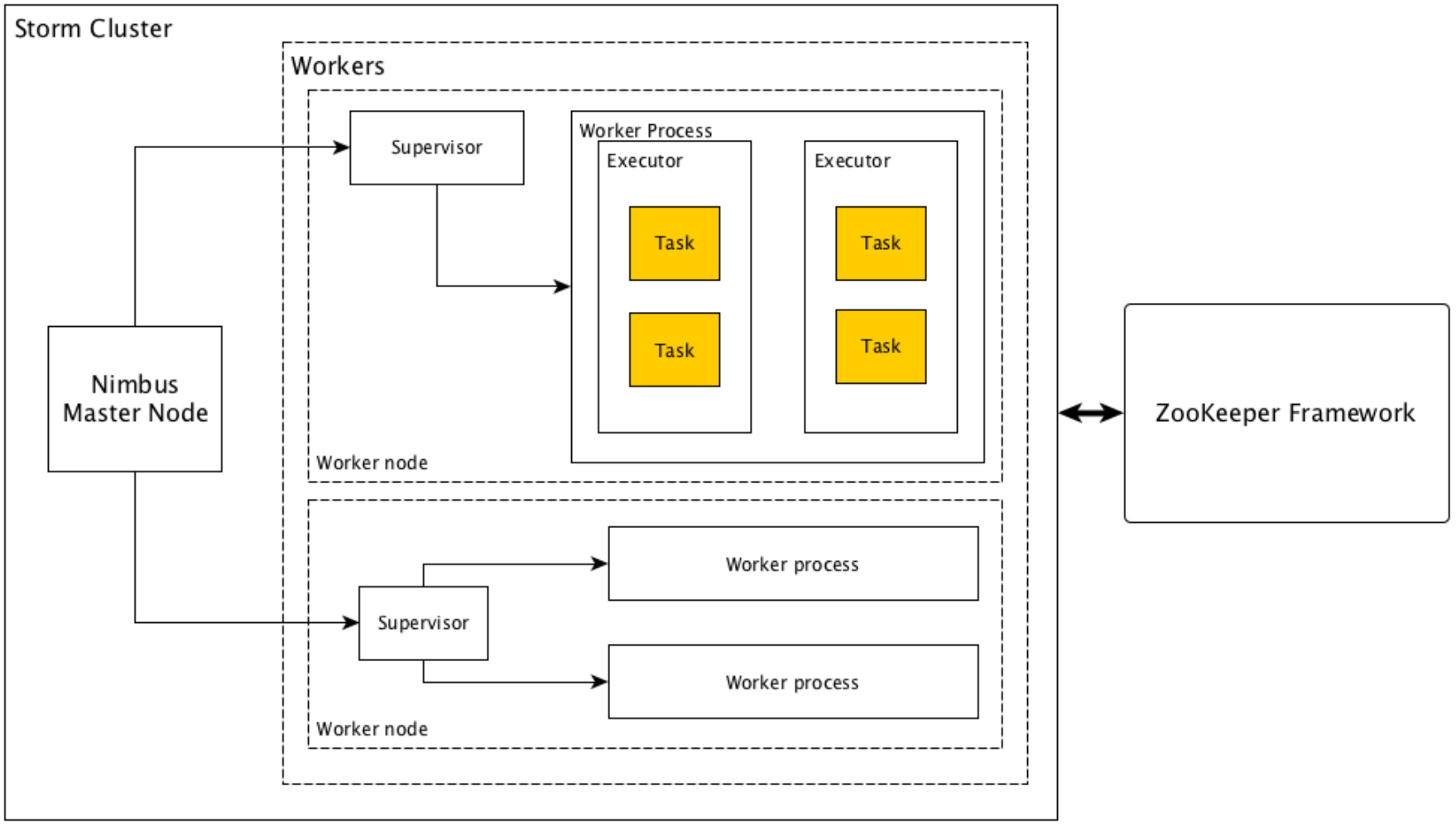
Apache Storm (4)

Components	Description
Tuple	Tuple is the main data structure in Storm. It is a list of ordered elements. By default, a Tuple supports all data types. Generally, it is modelled as a set of comma separated values and passed to a Storm cluster.
Stream	Stream is an unordered sequence of tuples.
Spouts	Source of stream. Generally, Storm accepts input data from raw data sources like Twitter Streaming API, Apache Kafka queue, Kestrel queue, etc. Otherwise you can write spouts to read data from datasources. "ISpout" is the core interface for implementing spouts. Some of the specific interfaces are IRichSpout, BaseRichSpout, KafkaSpout, etc.
Bolts	Bolts are logical processing units. Spouts pass data to bolts and bolts process and produce a new output stream. Bolts can perform the operations of filtering, aggregation, joining, interacting with data sources and databases. Bolt receives data and emits to one or more bolts. "IBolt" is the core interface for implementing bolts. Some of the common interfaces are IRichBolt, IBasicBolt, etc.

Apache Storm (5)



Storm Cluster

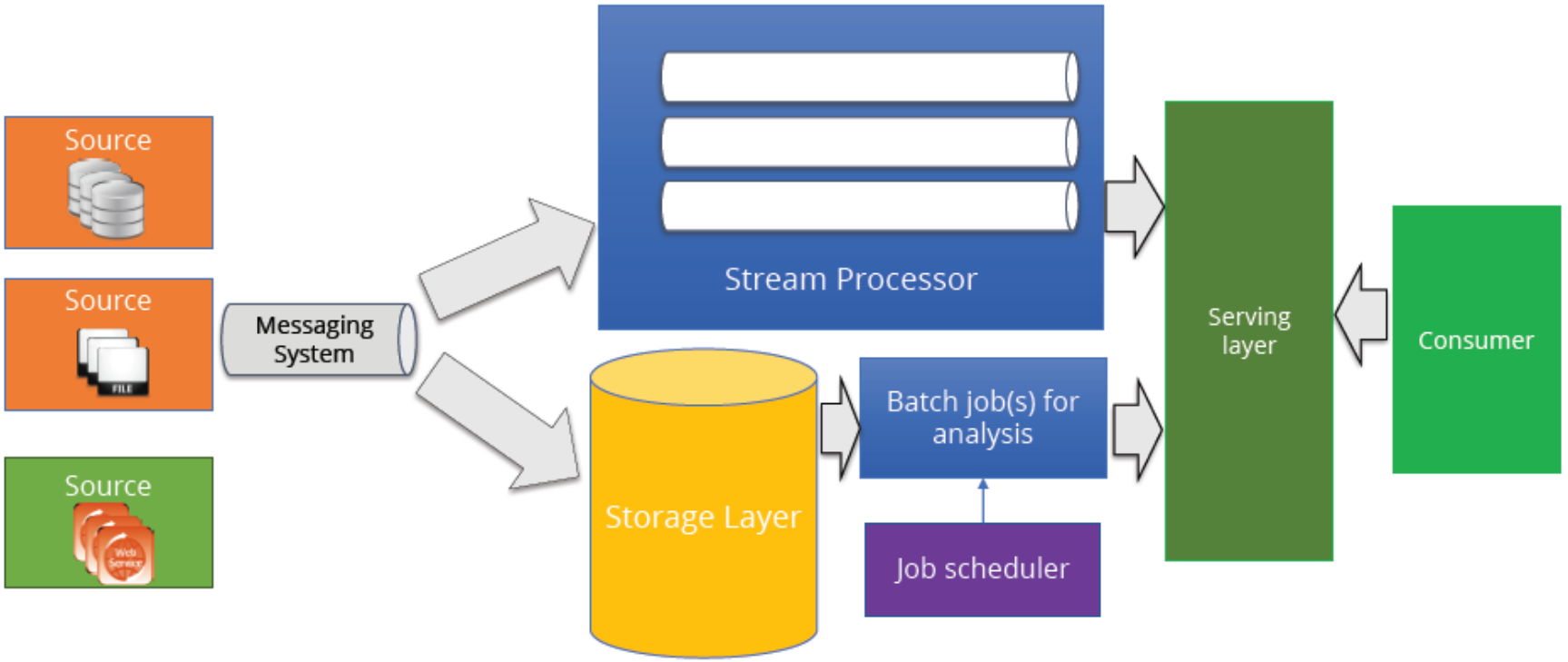


Apache Samza

- Apache Samza is a stream processing framework that is tightly tied to the Apache Kafka messaging system.
- While Kafka can be used by many stream processing systems, Samza is designed specifically to take advantage of Kafka's unique architecture and guarantees.
- It uses Kafka to provide fault tolerance, buffering, and state storage.
- Apache Samza is a good choice for streaming workloads where Hadoop and Kafka are either already available or sensible to implement.
- Samza itself is a good fit for organizations with multiple teams using (but not necessarily tightly coordinating around) data streams at various stages of processing.
- Samza greatly simplifies many parts of stream processing and offers low latency performance.

Hybrid Processing Systems: Batch and Stream Processors/Lambda processing

- Some processing frameworks can handle both batch and stream workloads.
  - These frameworks simplify diverse processing requirements by allowing the same or related components and APIs to be used for both types of data.

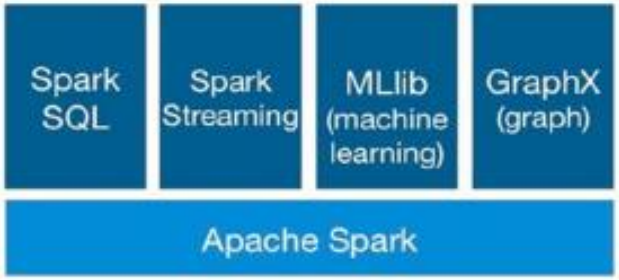


Apache Spark (1)

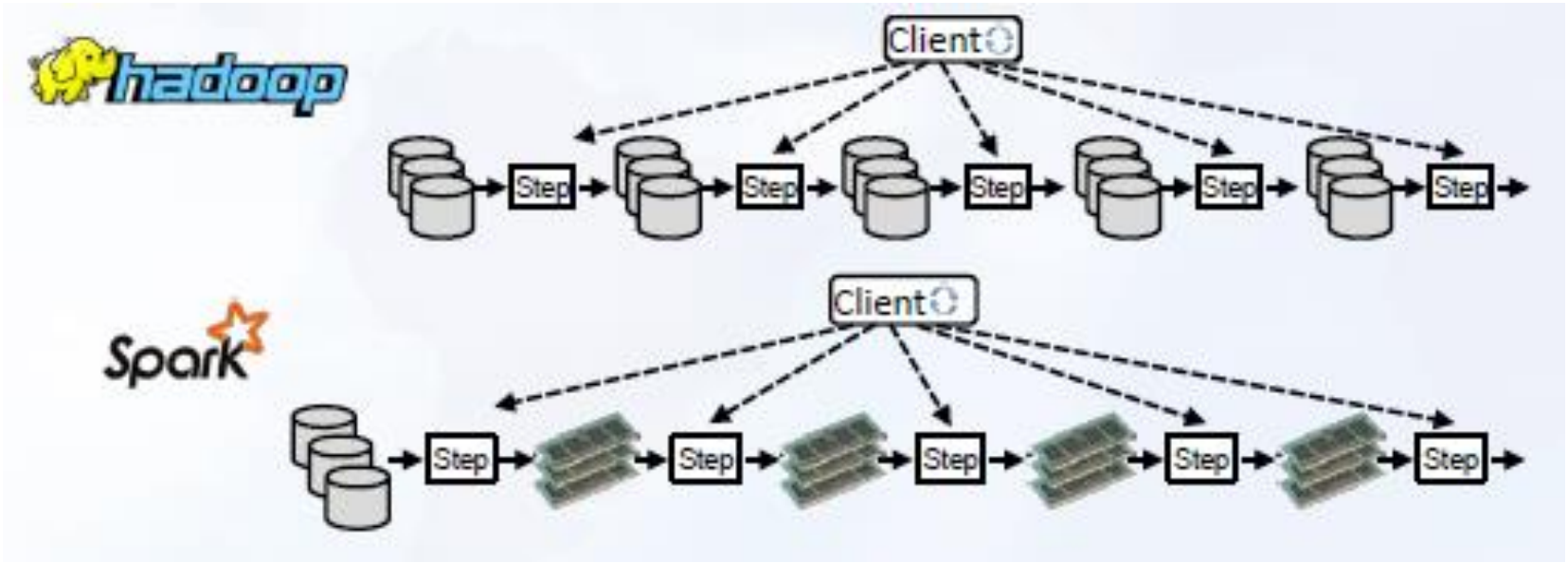
- Initially started at UC Berkeley in 2009
- Apache Spark is a next generation batch processing framework with stream processing capabilities.
- Fast and general purpose cluster computing system
- 10x (on disk) - 100x (In-Memory) faster
- Most popular for running Iterative Machine Learning Algorithms.
- Provides high level APIs in
  - Java
  - Scala
  - Python
- Spark Streaming is a good stream processing solution for workloads that value throughput over latency.
- Integration with Hadoop and its eco-system and can read existing data.

Apache Spark (2)

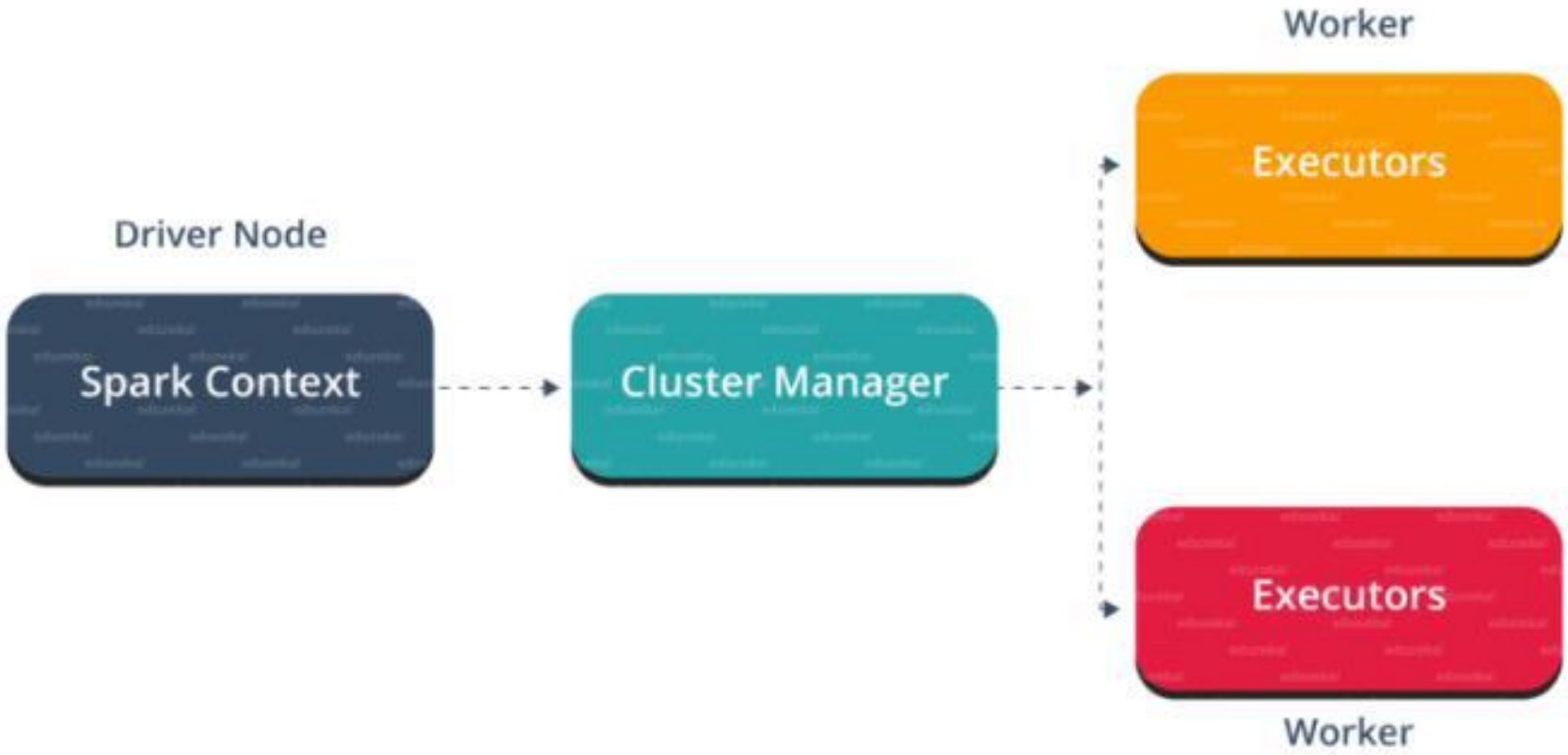
- Distributed data analytics engine, generalizing Map/Reduce
- Core engine, with streaming, SQL, machine learning, and graph processing modules



Spark vs Hadoop



Spark Architecture



Spark Ecosystem

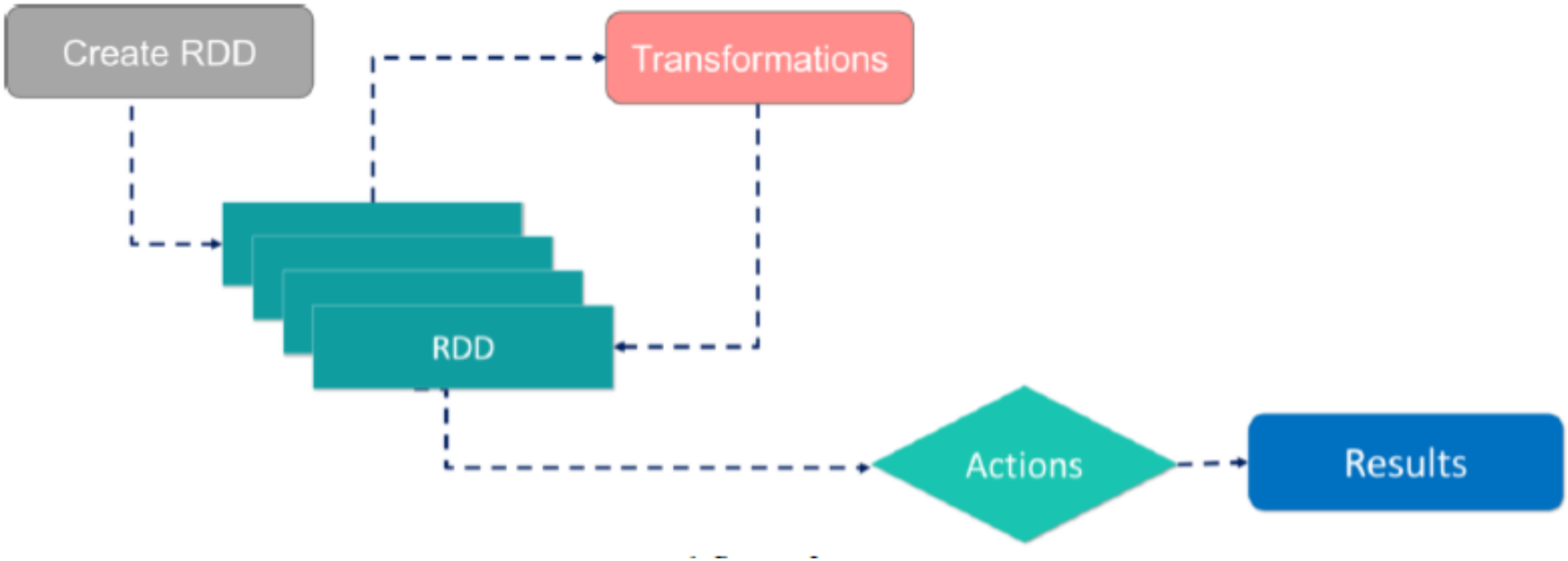
Resilient Distributed Dataset (RDD)

- RDDs represent data or transformations on data
- RDDs Stands for:
  - Resilient: Fault tolerant and is capable of rebuilding data on failure
  - Distributed: Distributed data among the multiple nodes in a cluster
  - Dataset: Collection of partitioned data with values
- With RDDs, you can perform two types of operations:
  - Transformations: They are the operations that are applied to create a new RDD.
  - Actions: They are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver.

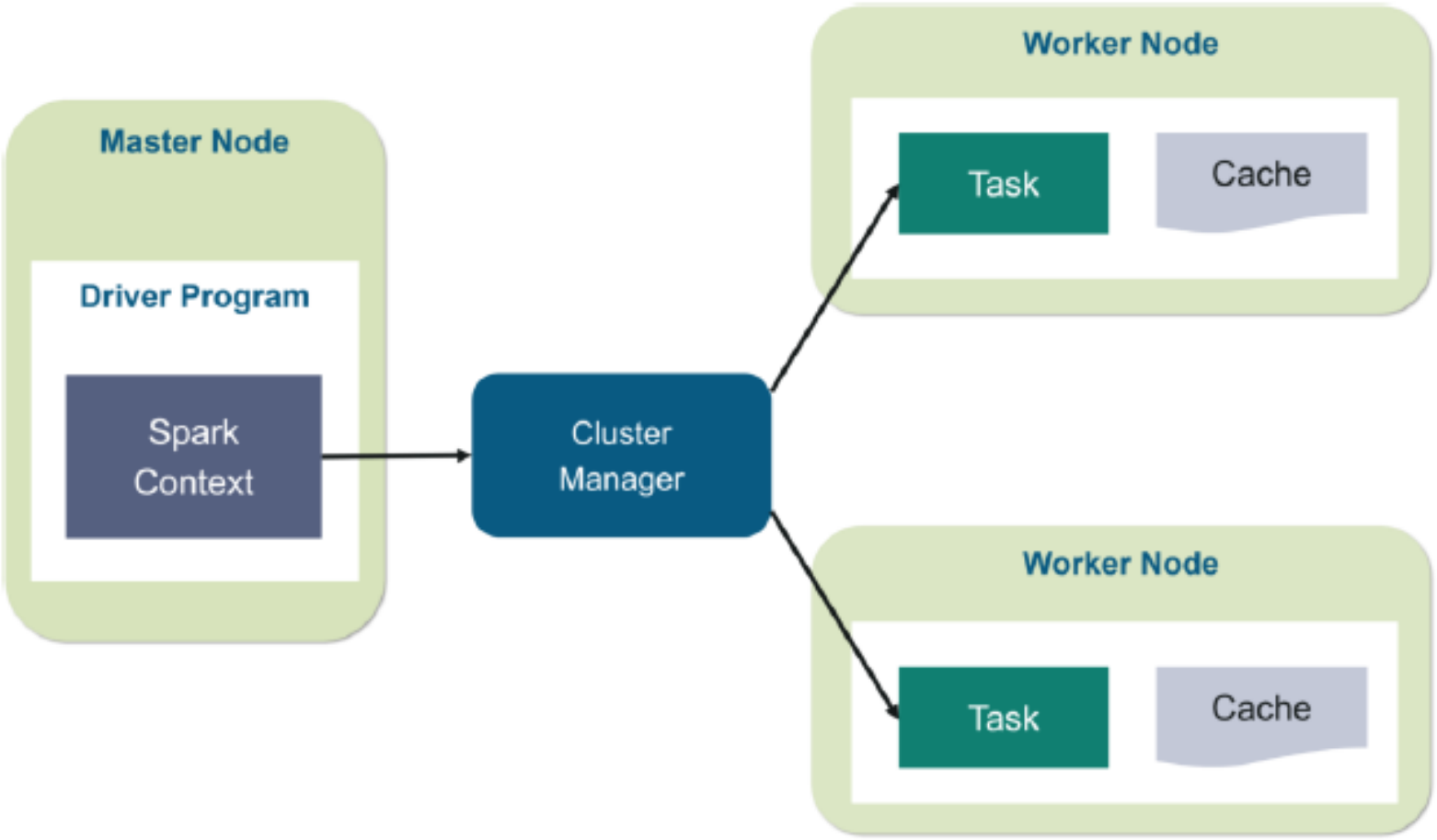
DataFrame & Dataset

- DataFrame:
  - Unlike an RDD, data organized into named columns, e.g. a table in a relational database.
  - Imposes a structure onto a distributed collection of data, allowing higher-level abstraction
- Dataset:
  - Extension of DataFrame API which provides type-safe, object-oriented programming interface (compile-time error detection)
- Both built on Spark SQL engine & use Catalyst to generate optimized logical and physical query plan; both can be converted to an RDD

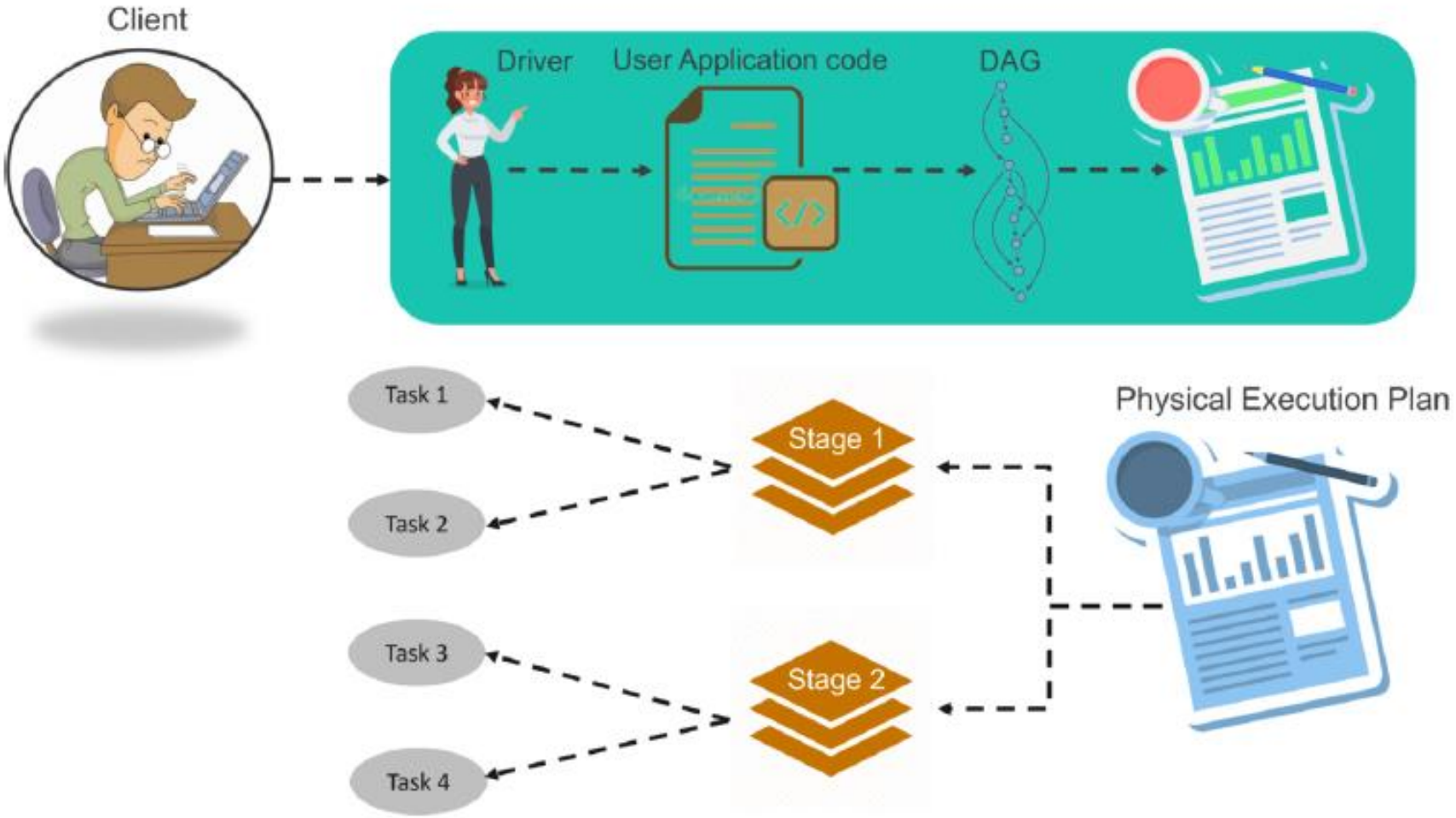
Workflow of RDD



Spark Cluster Architecture

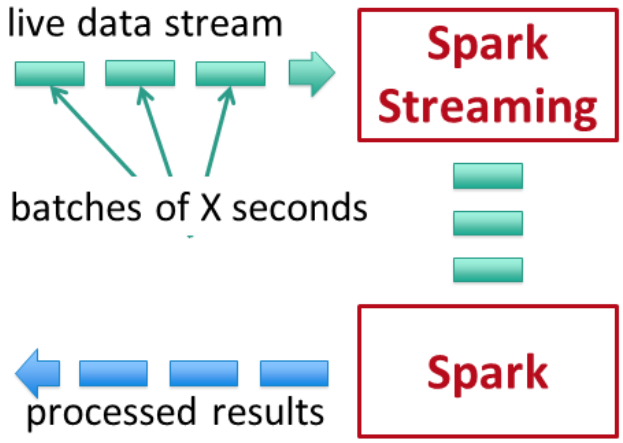


Spark Architecture Infographic

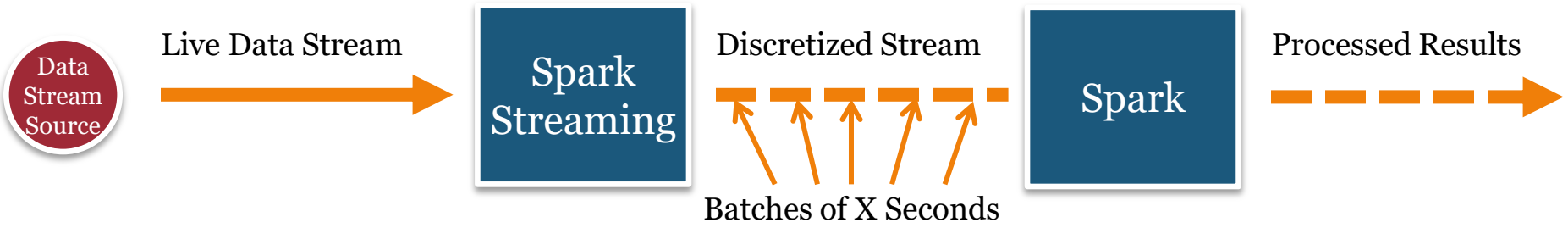


Spark Streaming (1)

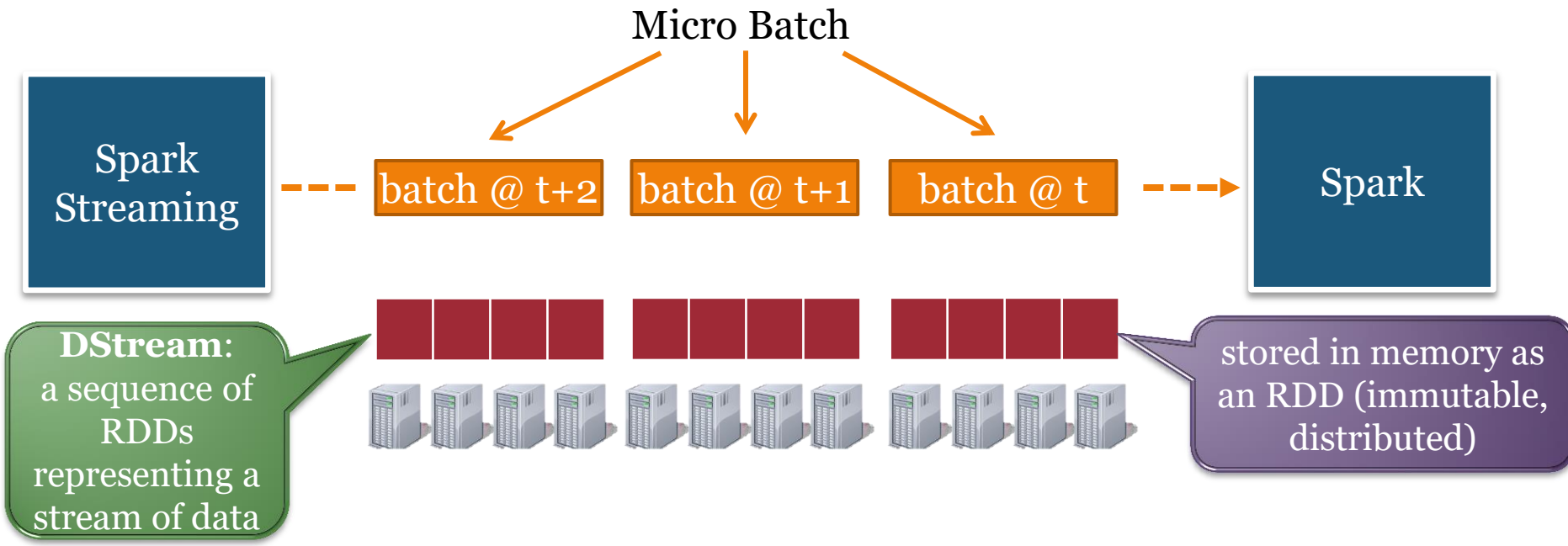
- Discretized Stream Processing
- Run a streaming computation as a series of very small, deterministic batch jobs
  - Chop up the live stream into batches of X seconds
  - Spark treats each batch of data as RDDs and processes them using RDD operations
  - Finally, the processed results of the RDD operations are returned in batches



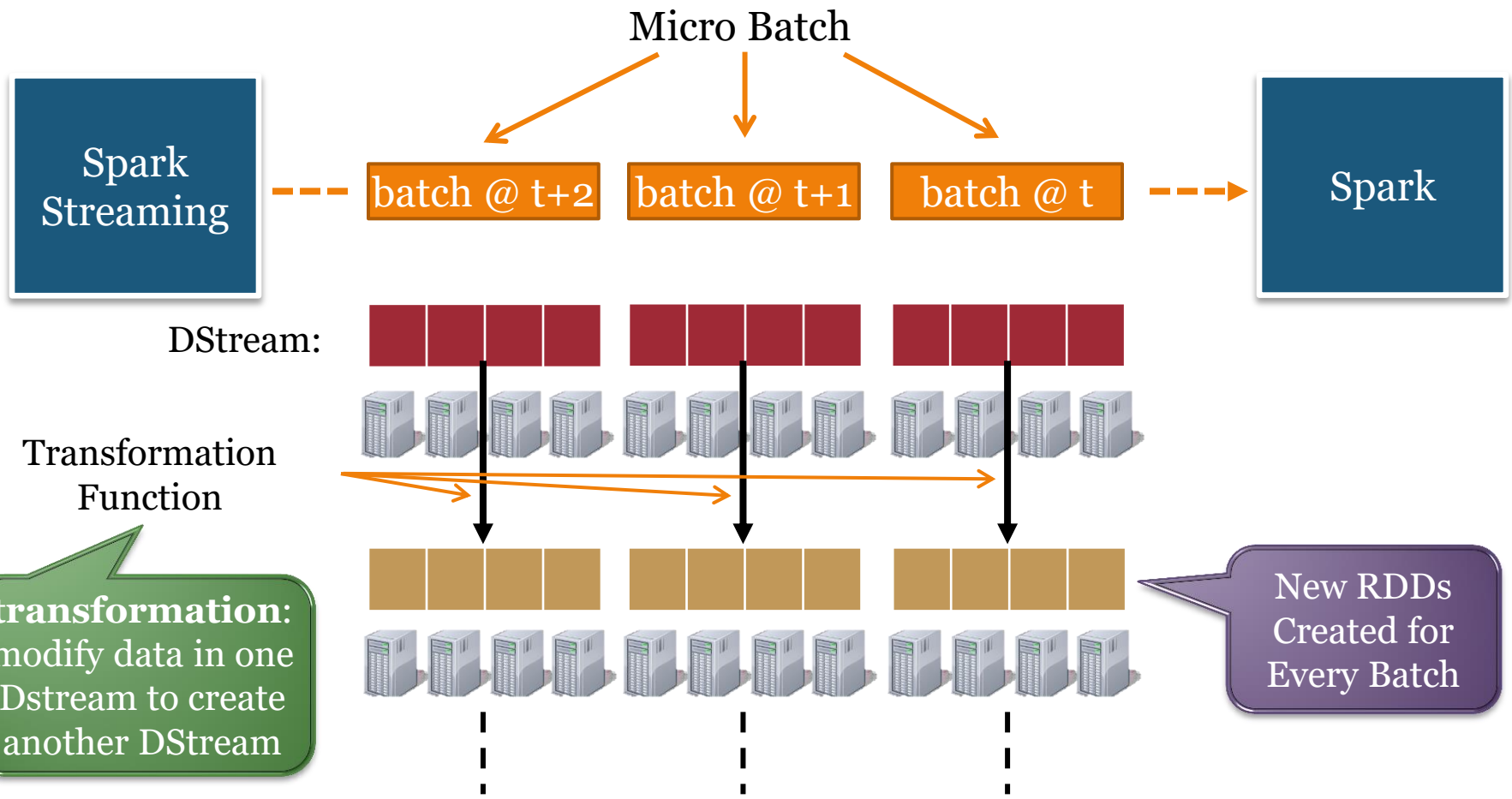
Spark Streaming (2)



Spark Streaming (3)



Spark Streaming (4)

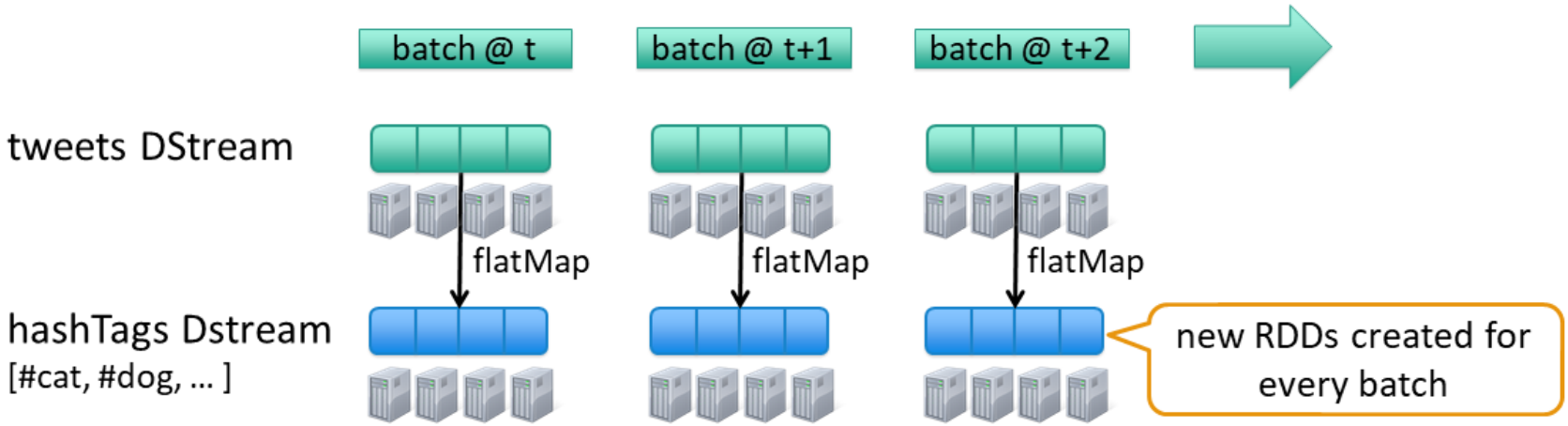


Example 1– Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
```

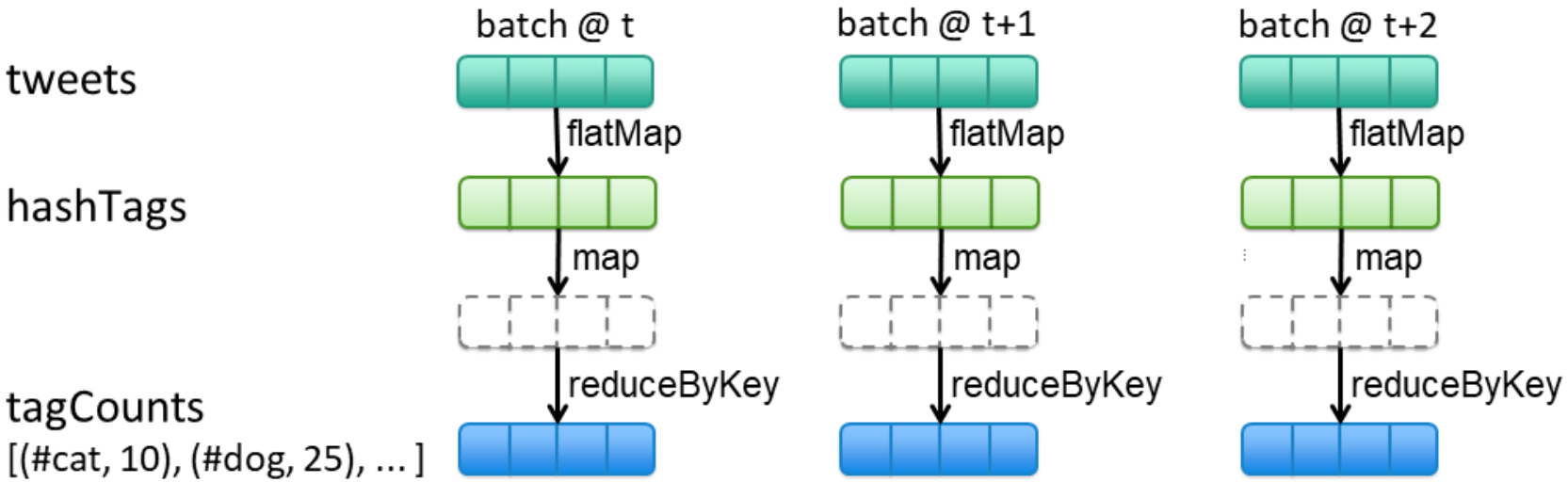
new DStream

transformation: modify data in one Dstream to create another DStream



Example 2 – Count the hashtags

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.countByValue()
```

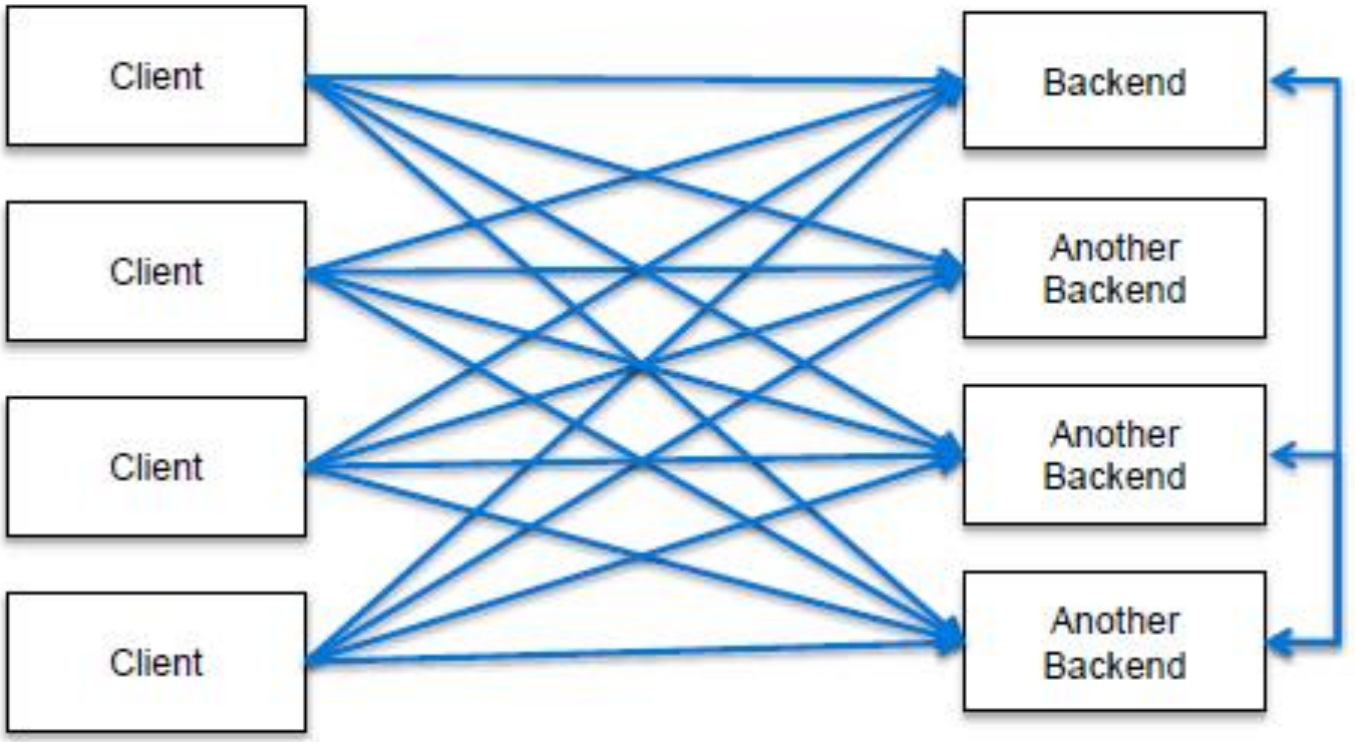


Big Data Processing Frameworks Comparison

- There are plenty of options for processing within a big data system.
  - For batch-only workloads that are not time-sensitive, Hadoop is a good choice that is likely less expensive to implement than some other solutions.
  - For stream-only workloads, Storm has wide language support and can deliver very low latency processing, but can deliver duplicates and cannot guarantee ordering in its default configuration.
  - For mixed workloads, Spark provides high speed batch processing and micro-batch processing for streaming. It has wide support, integrated libraries and tooling, and flexible integrations.
  - The best fit for your situation will depend heavily upon the state of the data to process

Kafka

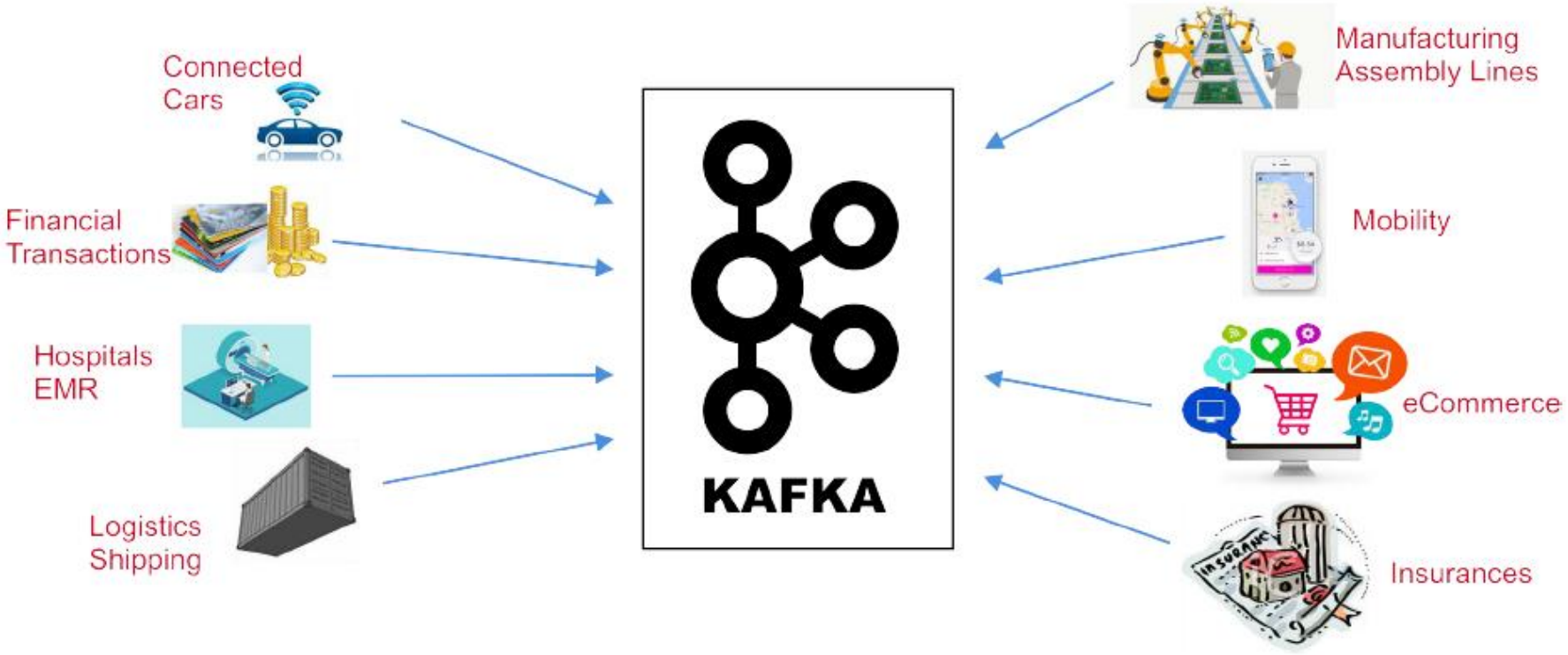
- As distributed systems and services increasingly become part of a modern architecture, this makes for a fragile system



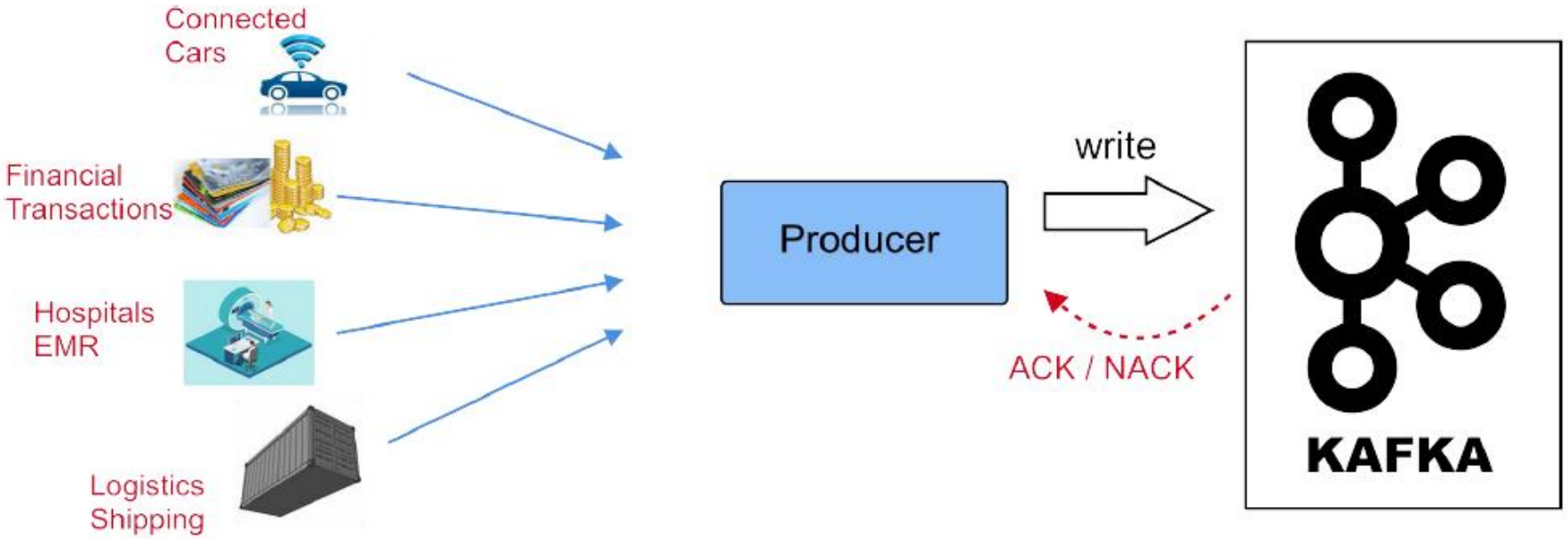
Why is Kafka needed?

- Real time streaming data processed for real time analytics
  - Service calls, track every call, IOT sensors
- Apache Kafka is a fast, scalable, durable, and fault tolerant publish-subscribe messaging system
- Kafka is
  - higher throughput, reliability and replication
- Kafka can works in combination with
  - Spark Streaming, Storm, HBase and Spark for real-time analysis and processing of streaming data

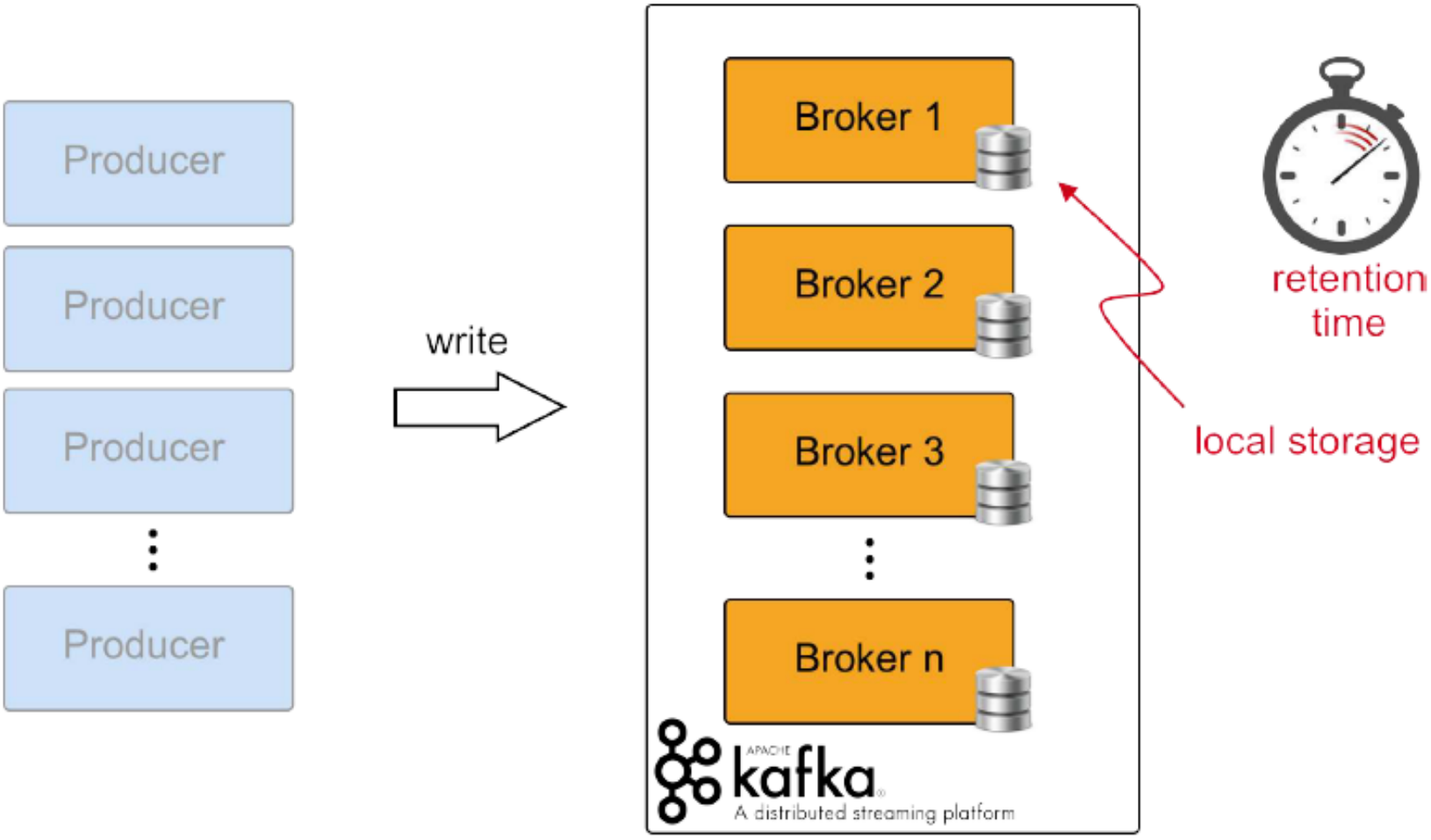
The World Produces Data



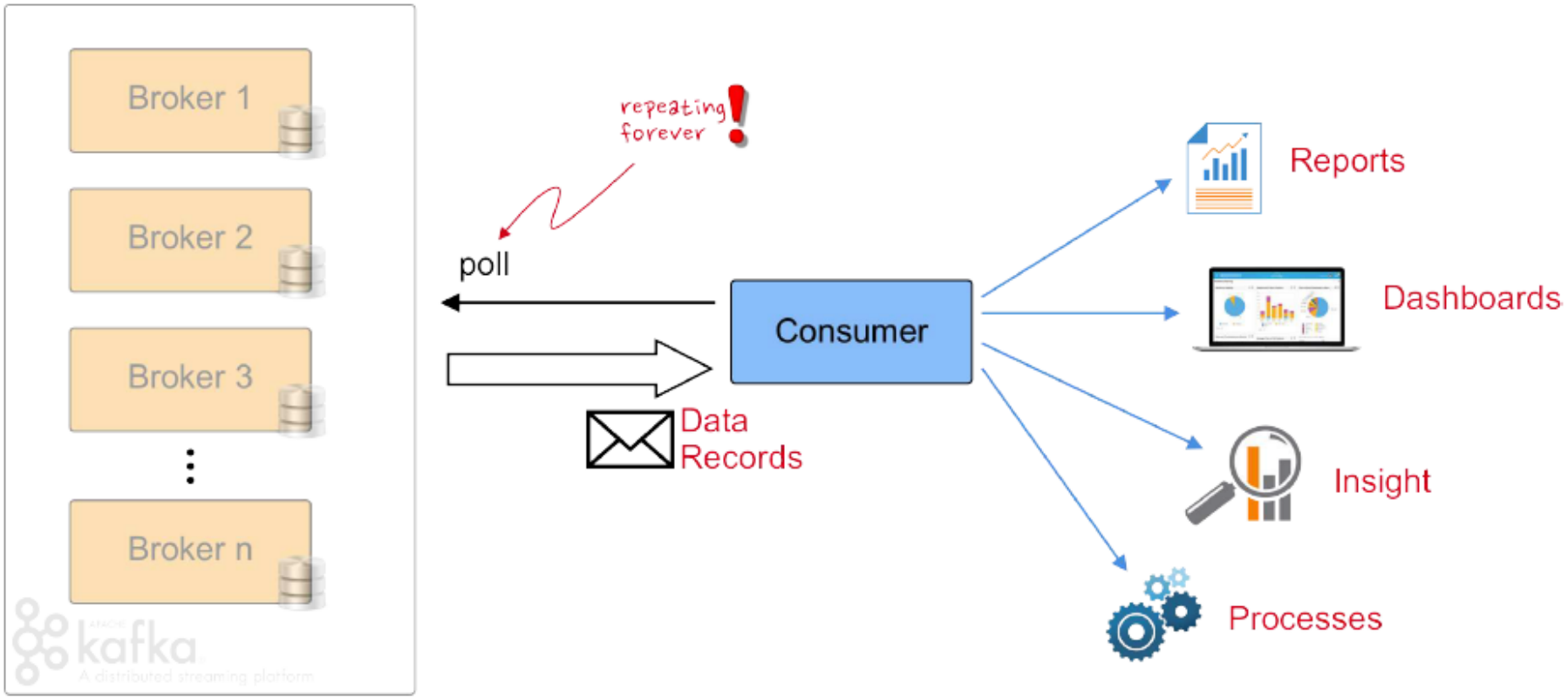
Producers


Connected
Cars
Financial
Transactions
Hospitals
EMR
Logistics
Shipping
Producer
write
ACK / NACK
KAFKA

Kafka Brokers



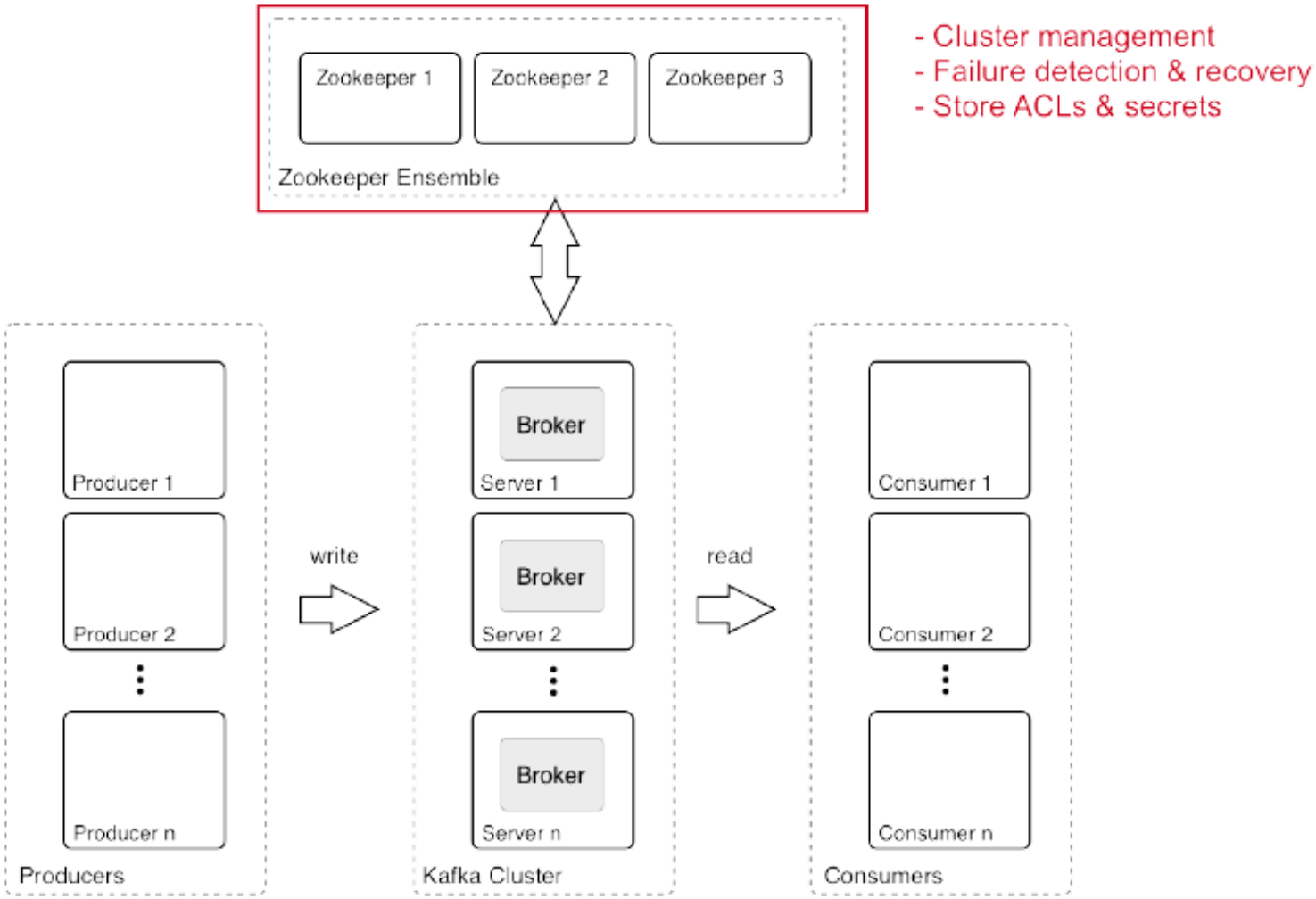
Kafka Brokers



Key terminology

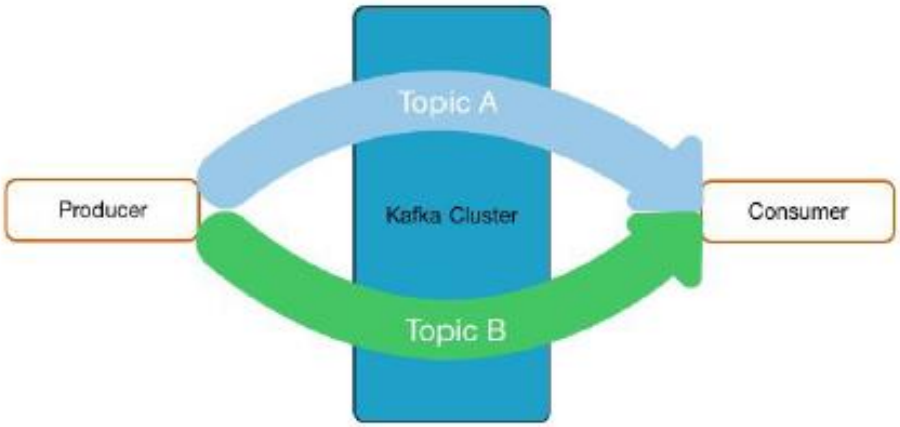
- Kafka maintains feeds of messages in categories called ***topics***.
- Processes that publish messages to a Kafka topic are called ***producers***
- Processes that subscribe to topics and process the feed of published messages are called ***consumers***.
- Kafka is run as a cluster comprised of one or more servers each of which is called a ***broker***.
- Communication between all components is done via a high performance simple binary API over TCP protocol

Kafka Architecture



Topics

- Topics:
  - Streams of “related” Messages in Kafka
  - Is a Logical Representation
  - Categorizes Messages into Groups
- Developers define Topics
- Producer - Topic: N to N Relation
- Unlimited Number of Topics




Thank
You
Merci
Gracias
TapadhLeat
Köszönöm
Murakoze
aDank
BarakAllahuFiik
Grazzi
Grazie
Nouari
Shukriya
Buznyg
WaadMahadsantahay
Takk
Enkosi
ThintKo
TapadhLeibh
FaafetaiLava
Ngiyabonga
M-Sapo
Bedankt
Zikomo
Mèsi
Chokrane
Kiitos
AsanteSana
Blagodaram
Waita
Rahmat
Dhanyavaad
Xièxie
Dziakuju
Mauruuru
GoRaibhMaithAgat
Trugarez
KyayTzuTinPaTe
Mochchakkeram
Matondo
Aabhar
Welalin
TangioTumas
Dhanyavad
TeşekkürEderim
Sobodi
AabhariAahe
Barkal
KurreSumanga
GratiasAgimus
Mercé
TerimaKasih
Mamnun
Tanan
Sukriya
Yekeniele
Dhanyabaad
TesekkurEdirem
Taiku
KeYaLeboha
Vinaka
Tanemirt
Tenki
Chnorakaloutioun
TänanVäga
Ačiū
KopKhunKrap
Barka
Dankon
Dankie
Tau
Nanni
aNiKié
KhobChaiDeu
Motashakkeram
Dakujem
Toda
Spacibo
Danke
Gratzias
Trugéré
Akpé
Tak
Sagolun
Nizzik ajr
ANiKié
Misaotra
Mersi
Aguyjé
Diolch
Dhanyavadagalu
Grandmercé
Dhanyavadalu
Sulpáy
EskerrikAsko
Dankewol
Rahmet
GratiasAgo
Tanmirt
Hvala
Arigatô
Akiba
Gràcies
Blagodaria
Nandri
Gracie
MaloTeOfa
Gràçie
Spas
KopKhunKha
Faleminderit
Merkzi
Akun
Wado
Asante
NajisTuke
Mahalo
Graciñas
KamSahHamnida
CámOn
Efharisto
DankJuWel
Marahaba
Obrigado
ÇoxSagOlun
Diky
Meharbani
Multumesc
Salamat
Saha
Nirringrazzjak
Bayarlalaa
TakkFyri
Miigwetch