Intro to Big Data Processing Frameworks

By: Shahab Safaee
Computer Software Engineering PhD
Email: safaee.shx@gmail.com


cibtrc.ir


t.me/cibtrc


www.instagram.com/cibtrc/

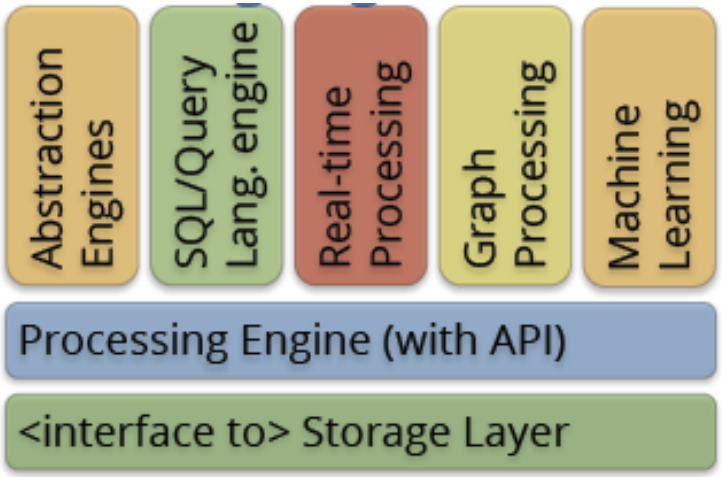
Agenda

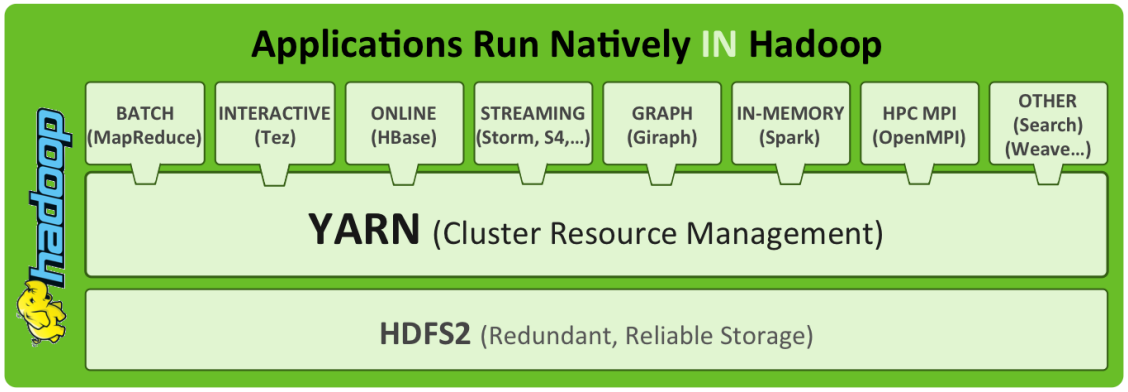
- Introduction
- Big Data Processing Architecture
- Type of Processing Systems
- Processing Types of Big Data Processing Frameworks
- Big Data Execution Engine
- Taxonomy of Programming Models
- Hadoop Execution Engine

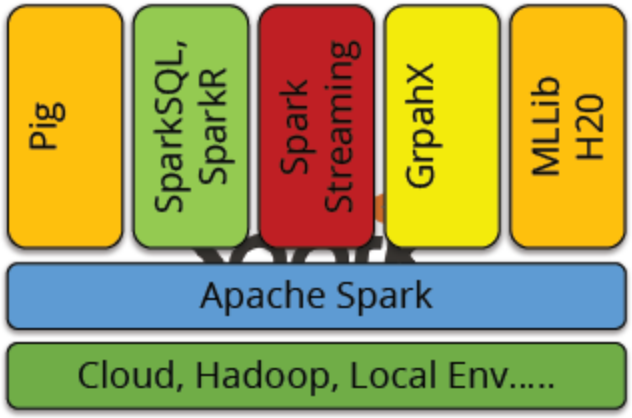
Introduction

- Big Data Processing Terminology
  - Big Data Processing Framework
    - Actual component responsible for operating on data
  - Big Data Execution Engine
    - Provide the computational model that performs several independent sequential computations in parallel which comprises of subcomponents of a larger computation.
  - Big Data Programming Models
    - Programming models normally the core feature of big data frameworks as they implicitly affects the execution model of big data processing engines and also drives the way for users to express and construct the big data applications and programs.

Big Data Processing Architecture







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.

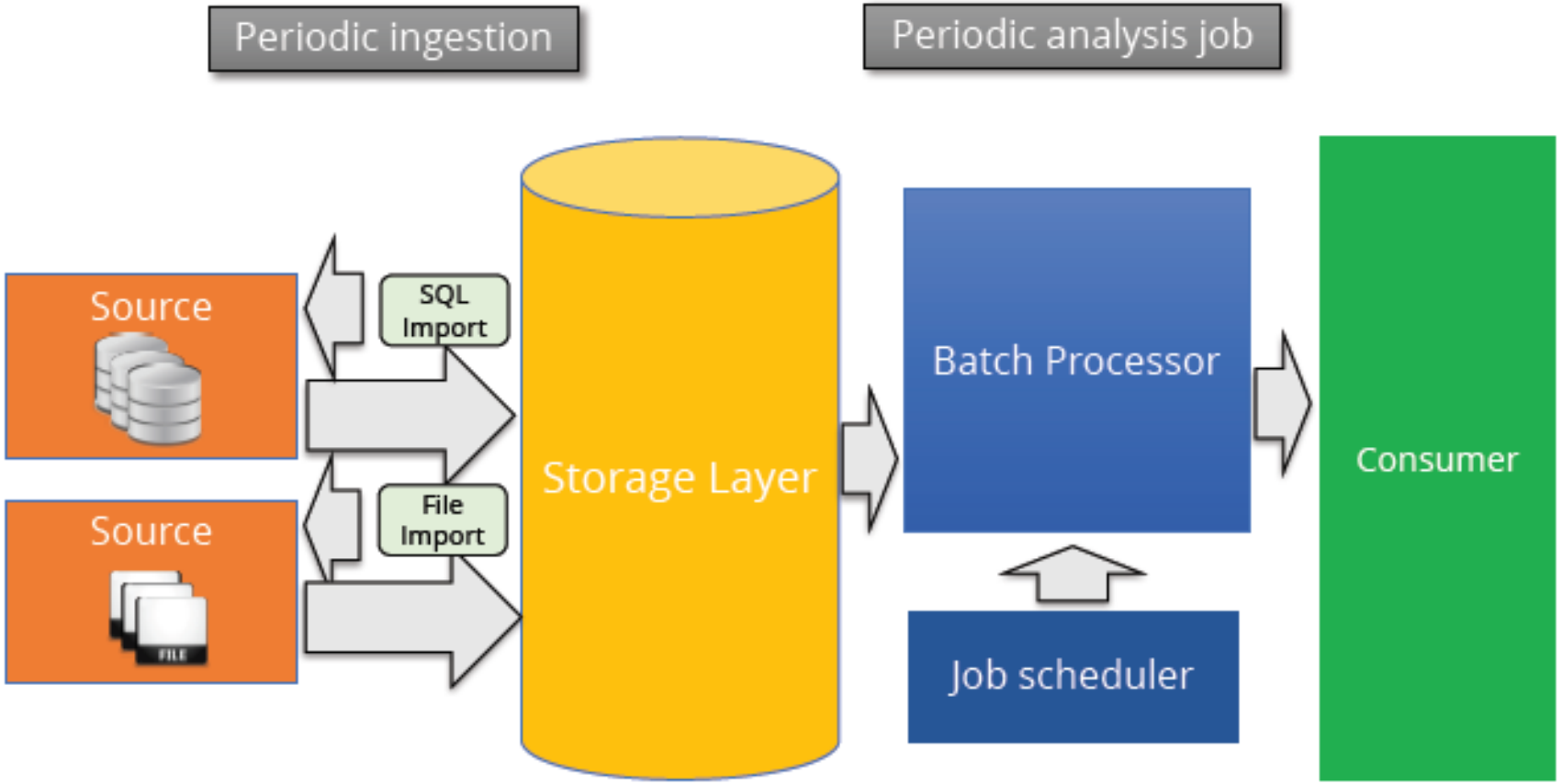
Type of Processing Systems (2)

- 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



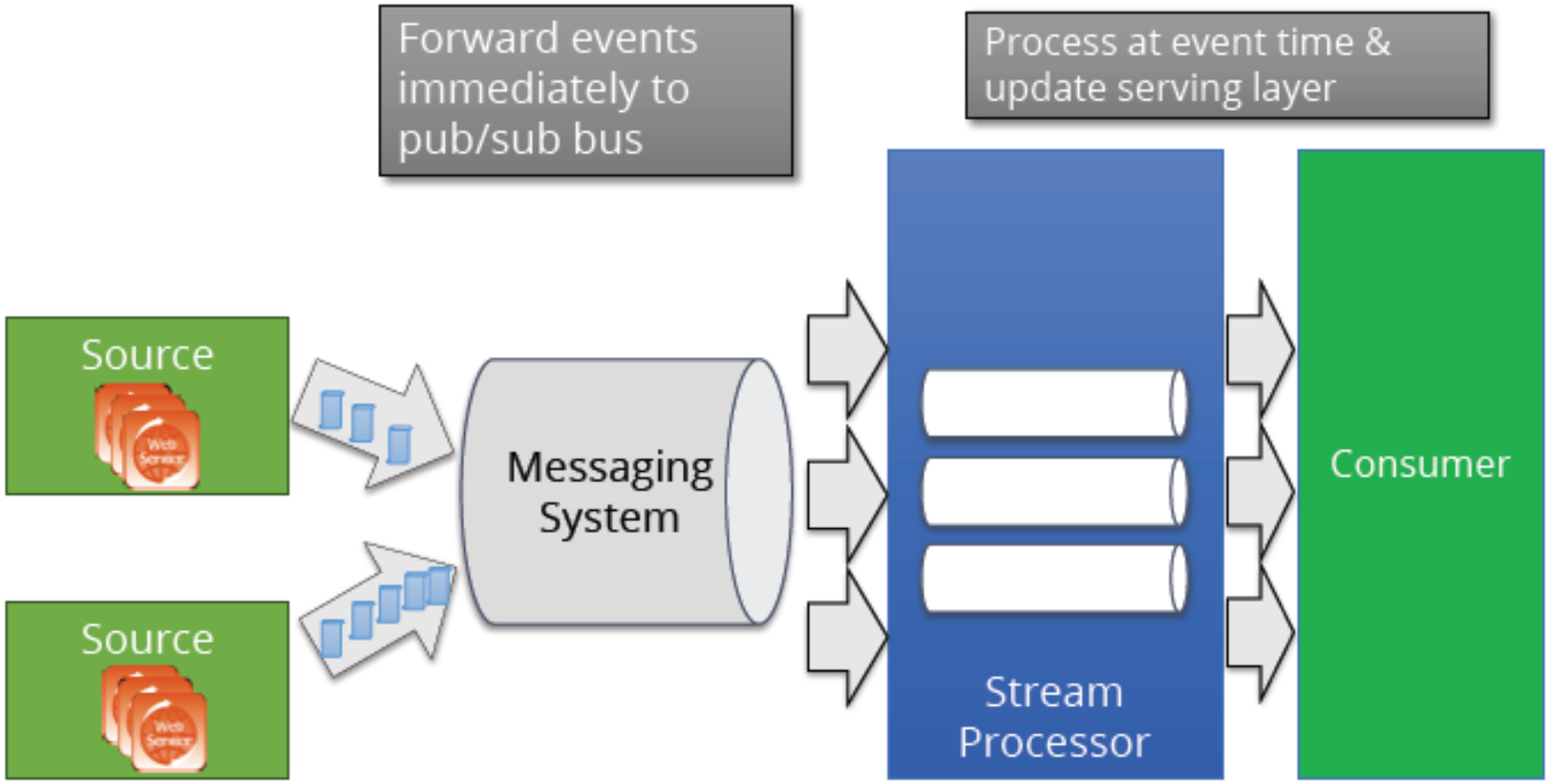
Apache Hadoop (1)

- Apache Hadoop is a processing framework that exclusively provides batch processing.
- Hadoop was the first big data framework to gain significant traction in the open-source community.
- Modern versions of Hadoop are composed of several components or layers, that work together to process batch data:
  - HDFS:
    - HDFS is the distributed file system layer that coordinates storage and replication across the cluster nodes.
    - HDFS ensures that data remains available in spite of inevitable host failures
  - YARN:
    - YARN, which stands for Yet Another Resource Negotiator, is the cluster coordinating component of the Hadoop stack. It is responsible for coordinating and managing the underlying resources and scheduling jobs to be run.
  - MapReduce:
    - MapReduce is Hadoop's native batch processing engine.

Apache Hadoop (2)

- Apache Hadoop and its MapReduce processing engine offer a well-tested batch processing model that is best suited for handling very large data sets where time is not a significant factor.
- The low cost of components necessary for a well-functioning Hadoop cluster makes this processing inexpensive and effective for many use cases.
- Compatibility and integration with other frameworks and engines mean that Hadoop can often serve as the foundation for multiple processing workloads using diverse technology.

Stream-only frameworks/Kappa Architecture



Apache Storm

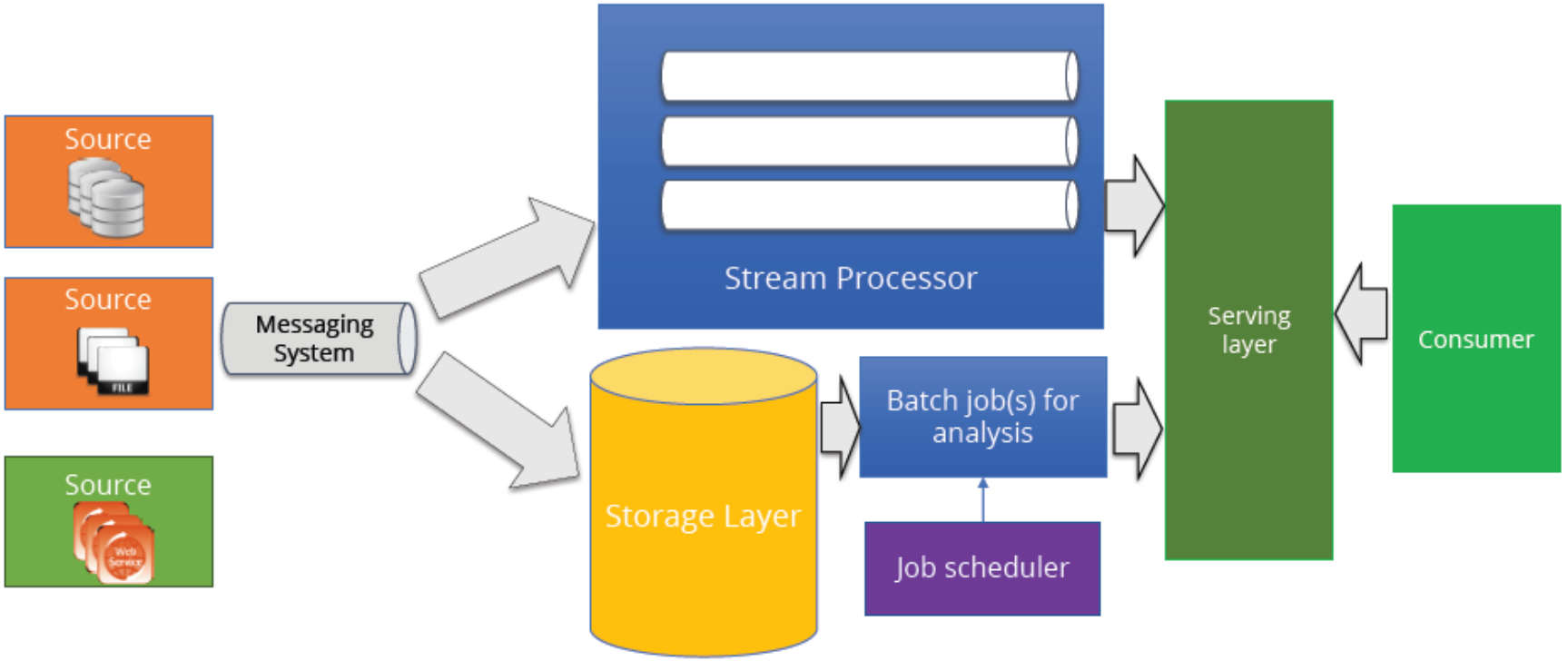
- Apache Storm is a stream processing framework that focuses on extremely low latency and is perhaps the best option for workloads that require near real-time processing.
- It can handle very large quantities of data with and deliver results with less latency than other solutions.
- For pure stream processing workloads with very strict latency requirements, Storm is probably the best mature option.
- It can guarantee message processing and can be used with a large number of programming languages.
- Because Storm does not do batch processing, you will have to use additional software if you require those capabilities.
- If you have a strong need for exactly-once processing guarantees, Trident can provide that.

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

- Apache Spark is a next generation batch processing framework with stream processing capabilities.
- Built using many of the same principles of Hadoop's MapReduce engine, Spark focuses primarily on speeding up batch processing workloads by offering full in-memory computation and processing optimization.
- Spark is a great option for those with diverse processing workloads. Spark batch processing offers incredible speed advantages, trading off high memory usage.
- Spark Streaming is a good stream processing solution for workloads that value throughput over latency.

Apache Flink (1)

- Apache Flink is a stream processing framework that can also handle batch tasks. It considers batches to simply be data streams with finite boundaries, and thus treats batch processing as a subset of stream processing. This stream-first approach to all processing has a number of interesting side effects.
- This stream-first approach has been called the Kappa architecture, in contrast to the more widely known Lambda architecture (where batching is used as the primary processing method with streams used to supplement and provide early but unrefined results).
- Kappa architecture, where streams are used for everything, simplifies the model and has only recently become possible as stream processing engines have grown more sophisticated.

Apache Flink (2)

- Flink offers both low latency stream processing with support for traditional batch tasks.
- Flink is probably best suited for organizations that have heavy stream processing requirements and some batch-oriented tasks.
- Its compatibility with native Storm and Hadoop programs, and its ability to run on a YARN-managed cluster can make it easy to evaluate.
- Its rapid development makes it worth keeping an eye on.

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. Flink provides true stream processing with batch processing support. It is heavily optimized, can run tasks written for other platforms, and provides low latency processing, but is still in the early days of adoption.
  - The best fit for your situation will depend heavily upon the state of the data to process

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, ...

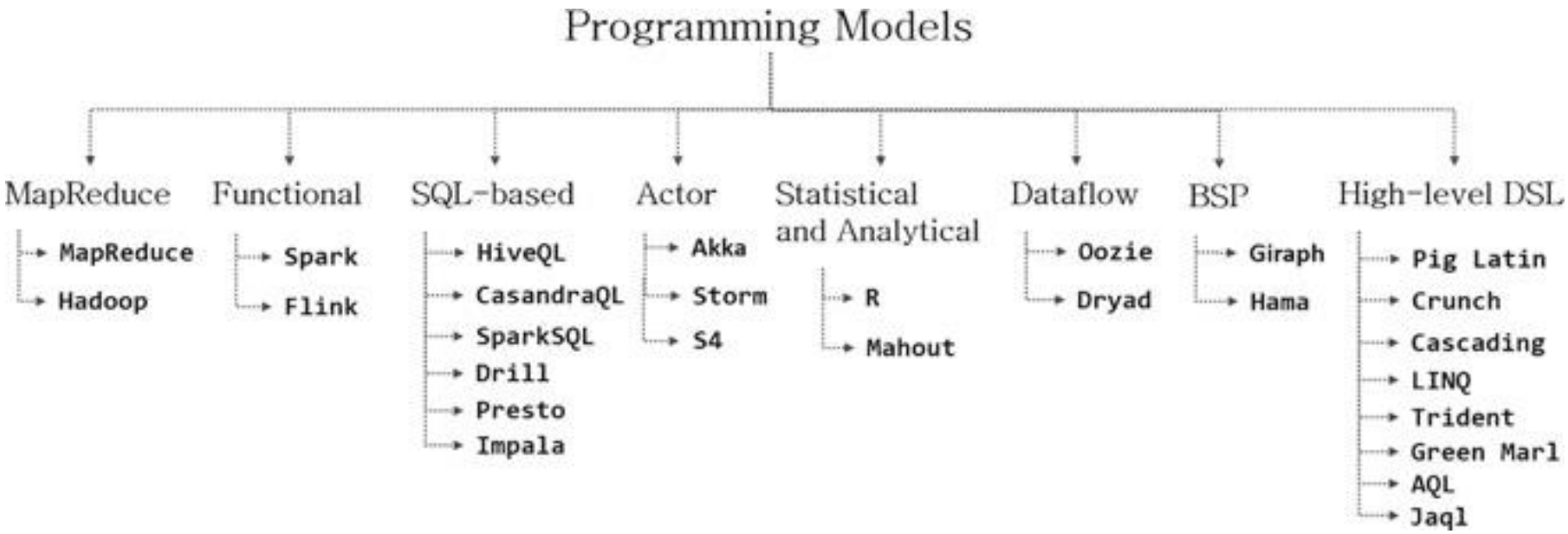
Distributed Data-Parallel Patterns and Execution Engines

- The are many distributed data-parallel patterns
- The main advantages of using these patterns:
  - Support the distribution of data and parallel processing of data with distributed data on multiple nodes/cores.
  - Provide a higher-level programming model in order to facilitate the user program parallelization.
  - Follow a principle of "moving computations to data" that reduces the data movements overheads.
  - Have a good scalability and efficiency in performance when executing on distributed resources.
  - Support run time features such as load balancing, fault-tolerance, etc.
  - Simplify the difficulties for parallel programming as compared to the traditional programming interfaces MPI and openMP.

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

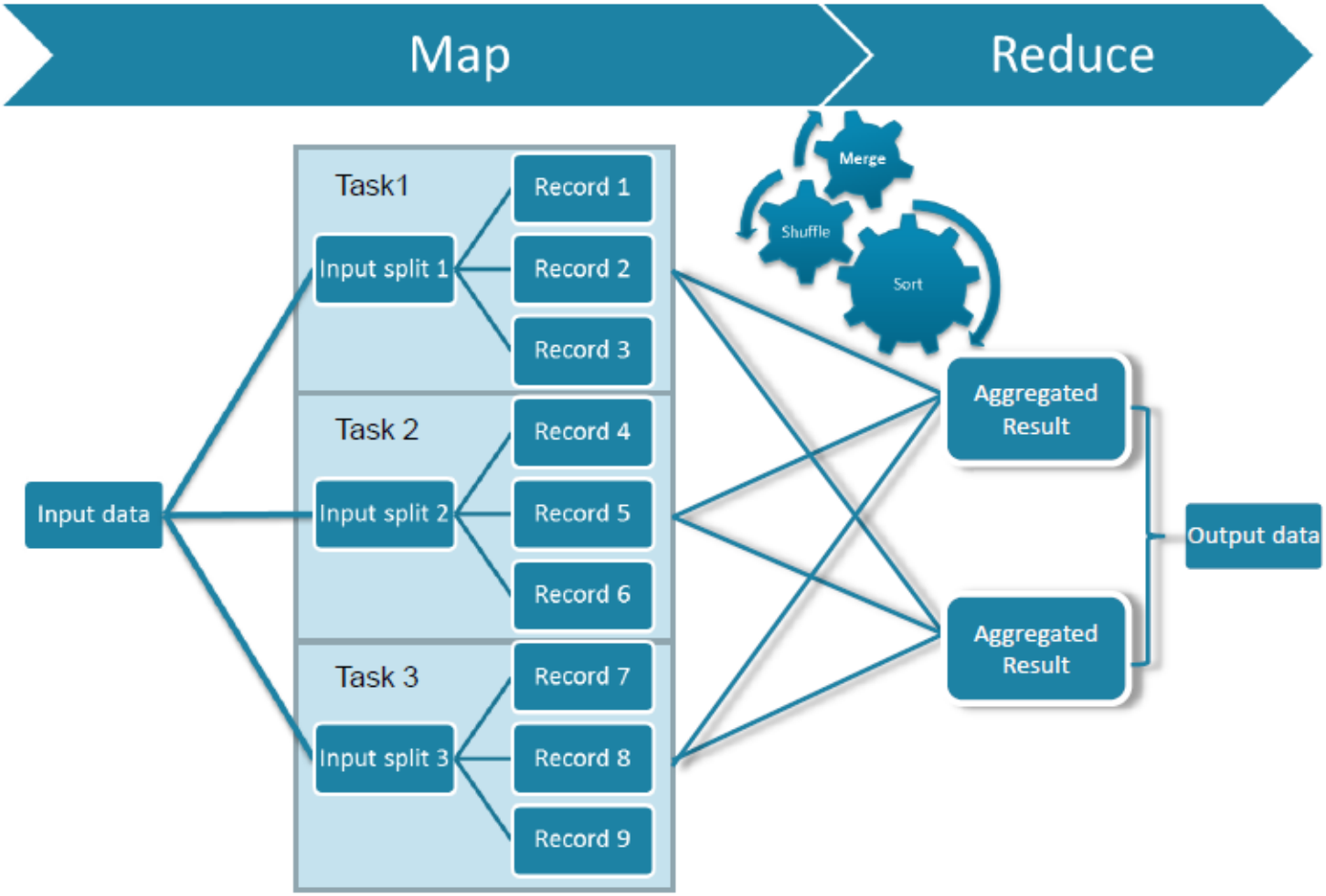
Taxonomy of Programming Models



MapReduce

- MapReduce the current framework/paradigm for writing data-centric parallel applications in both industry and academia.
- MapReduce is inspired by the commonly used functions - Map and Reduce in combination with the divide-and conquer parallel paradigm.
- For a single MapReduce job, users implement two basic procedure objects Mapper and Reducer for different processing stages as shown in Figure.

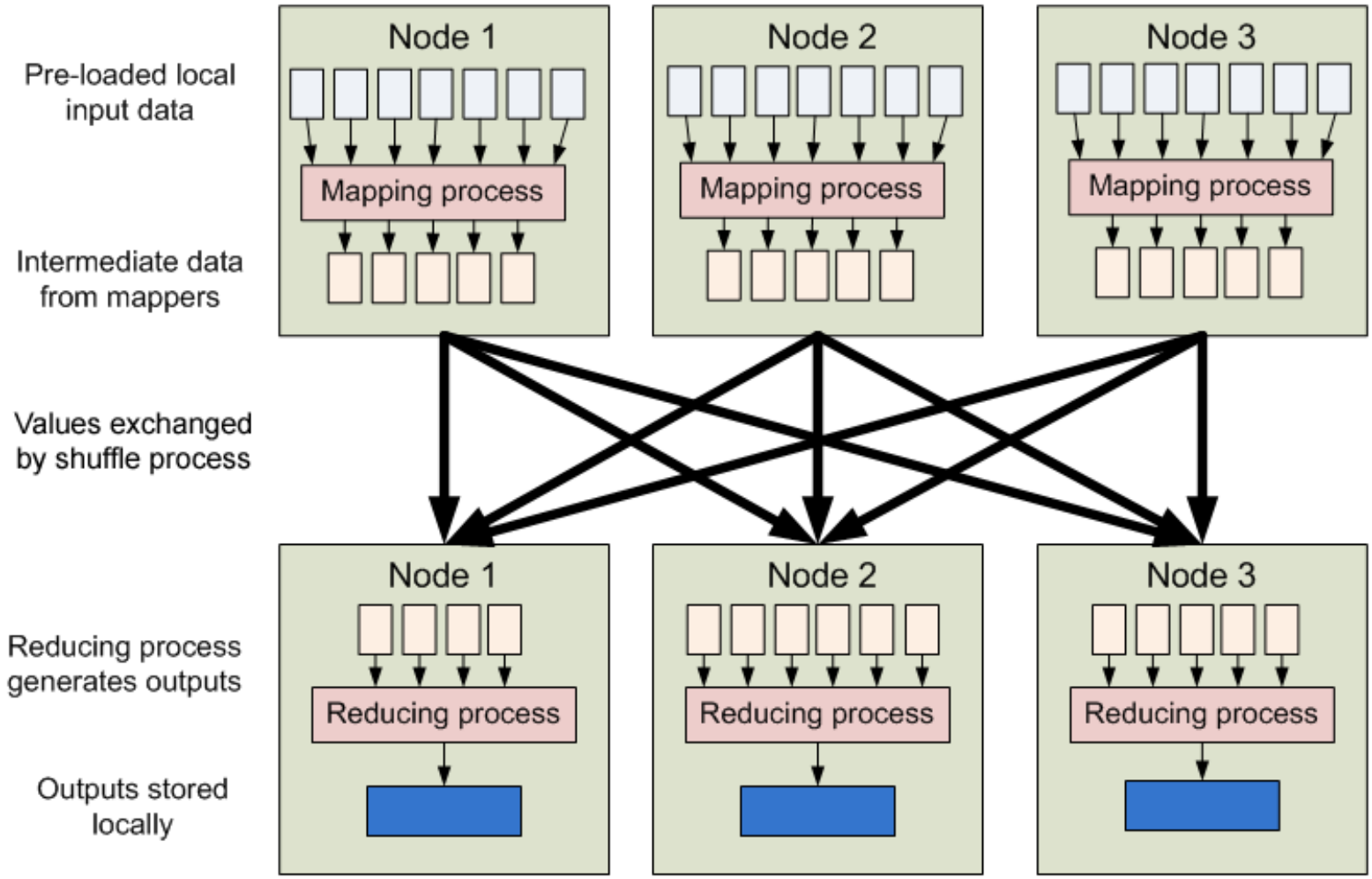
MapReduce Paradigm (1)



MapReduce Paradigm (2)

- Then the MapReduce program is automatically interpreted by the execution engine and executed in parallel in a distributed environments.
- MapReduce is considered as a simple yet powerful enough programming model to support a variety of the data-intensive programs.

MapReduce Dataflow



MapReduce Features (1)

- *Map and Reduce functions*
  - A MapReduce program contains a Map function doing the parallel transformation and a Reduce function doing the parallel aggregation and summary of the job.
  - Between Map and Reduce an implied Shuffle step is responsible for grouping and sorting the Mapped results and then feeding it into the Reduce step.

MapReduce Features (2)

- *Simple paradigm*
  - In MapReduce programming, users only need to write the logic of Mapper and Reducer while the logic of shuffling, partitioning and sorting is automatically done by the execution engine.
  - Complex applications and algorithms can be implemented by connecting a sequence of MapReduce jobs.
  - Due to this simple programming paradigm, it is much more convenient to write data-driven parallel applications, because users only need to consider the logic of processing data in each Mapper and Reducer without worrying about how to parallelize and coordinate the jobs.

MapReduce Features (3)

- *Key-Value based*
  - In MapReduce, both input and output data are considered as Key-Value pairs with different types.
  - This design is because of the requirements of parallelization and scalability.
  - Key-value pairs can be easily partitioned and distributed to be processed on distributed clusters.
- *Parallelable and Scalable*
  - Both Map and Reduce functions are designed to facilitate parallelization, so MapReduce applications are generally linearly-scalable to thousands of nodes.

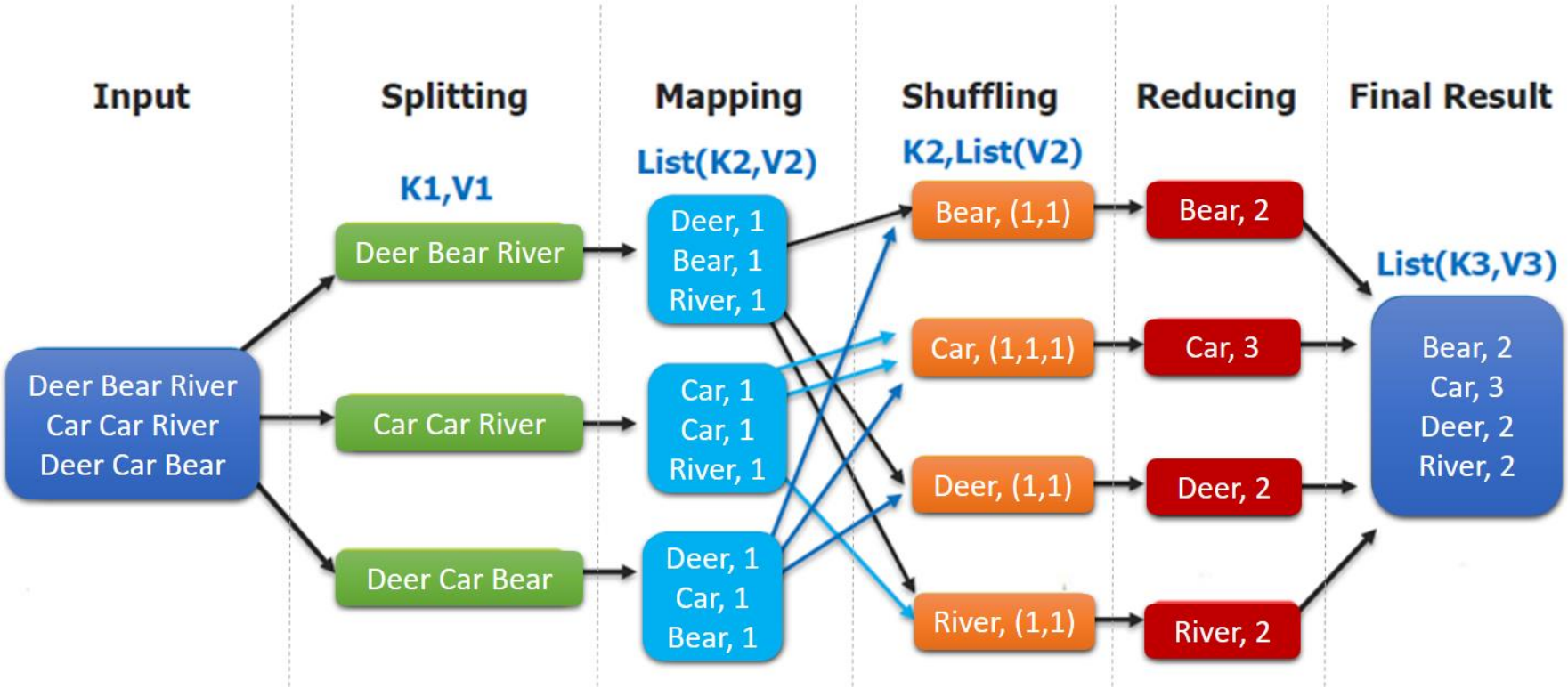
Hadoop Execution Engine (1)

- Hadoop MapReduce
  - Hadoop is the open-source implementation of Google's MapReduce paradigm.
  - The native programming primitives in Hadoop are Mapper and Reducer interfaces which can be implemented by programmers with their actual logic of processing map and reduce stage transformation and processing.
  - To support more complicated applications, users may need to chain a sequence of MapReduce jobs each of which is responsible for a processing module with well defined functionality.

Hadoop Execution Engine (2)

- Hadoop MapReduce
  - Hadoop is mainly implemented in Java, therefore, the map and reduce functions are wrapped as two interfaces called Mapper and Reducer.
  - The Mapper contains the logic of processing each key-value pair from the input.
  - The Reducer contains the logic for processing a set of values for each key.
  - Programmers build their MapReduce application by implementing those two interfaces and chaining them as an execution pipeline.

MapReduce Word Count Process



Word Count example in Hadoop

```
public void map(Object key,
                Text value, Context context) {
  String text = value.toString();
  StringTokenizer itr = new StringTokenizer(text);
  while(itr.hasMoreTokens()) {
    word.set(itr.nextToken());
    context.write(word, one);
  }
}

public void reduce(Text key,
       Iterable<IntWritable> values, Context context) {
  int sum = 0;
  for (IntWritable val : values) {
    sum += val.get();
  }
  result.set(sum);
  context.write(key, result);
}
```


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