TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

René Jäkel
Center for Information Services and High Performance Computing (ZIH)

Current trends in big data analysis: second generation data processing

Science at Extreme Scales: Where Big Data Meets Large-Scale Computing Tutorials

Sept. 17 2018, IPAM

Course overview

Part 1 – Challenges

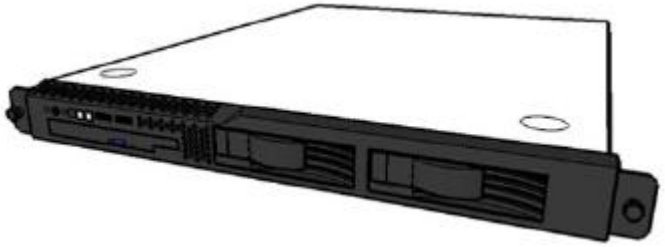
- Fundamentals and challenges in Big Data ((big) data analytics)
- Big Data methods useful for managing/transforming data and provide fast access to analytics functionality
- Complex analytics data-driven workflows vs. static parallel applications

Part 2 – Second generation (big) data processing

- Extending the Hadoop ecosystem
- Possible future directions

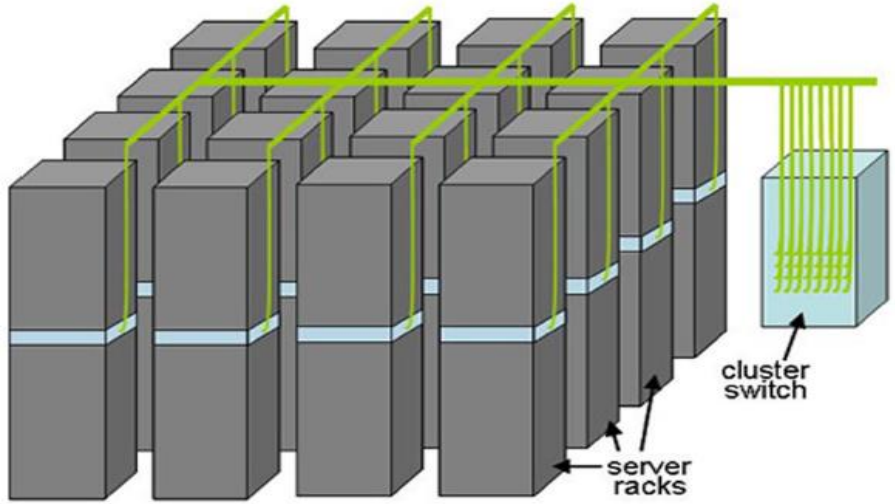
Part 3 – Introduction to Apache Spark and Hands-On

Recap: Infrastructure for Big Data



Commodity server
- CPUs
- DRAM
- Disks

Rack
- 40-80 Server
- Ethernet Switch



Cluster

Example: Google DataCenter

Source: http://www.google.com/about/datacenters/gallery/#/all/10

Shift in commodity trends

200x

- **Hardware**
  - Disk space **cheap** (primary storage solutions)
  - Network was costly
  - RAM was very **expensive**
  - Single core machines were dominant
- **Software**
  - Object orientation and optimization for single core
  - SQL as primary analysis language
  - some specific frameworks (e.g. Mathlab)
    OpenSource not widely seen as valid option

Now (late 201x)

- **Hardware**
  - RAM/Flash got **cheaper** and **faster** – tend to become primary storage, disk as fall-back
  - Network is faster / Virtualization
  - **Multi core** machines are dominating, different architectures
- **Software**
  - More functional programming and **frameworks**
  - **Multicore**-programming and distribution
  - NoSQL alternatives

History & evolution of big data

200x: First generation of data processing frameworks (Big Data)

Purpose/Background

- ONLY few companies had real big data need
- Batch processing was still dominant way to distribute workloads (reporting++)
- Primarily **volume** was biggest concern (not 5 V's)
- Mostly used for **search/basic behavior analysis** (logging data)

Hadoop-implementation offered simple programming approach

- Batch orientated
- Underlying HDFS for data distribution

History & evolution of big data

201x: Second generation of data processing frameworks (Big Data)

Purpose/Background

- Most companies have need to use big data technologies
- **Velocity** now dominating, also “Value”
- Diverse use cases
  - **Real time processing** of data
  - **Learning approaches, iterations, interactions**

Hadoop-approach not flexible enough any more

- Need for **in-memory processing** (handle iterations efficiently)
- Very **flexible hardware** options (Shared-Nothing vs. HPC)

How it got started

The **Google File System** (Sanjay Ghemawat, H. Gobioff, S. Leung. SOSP 2003)

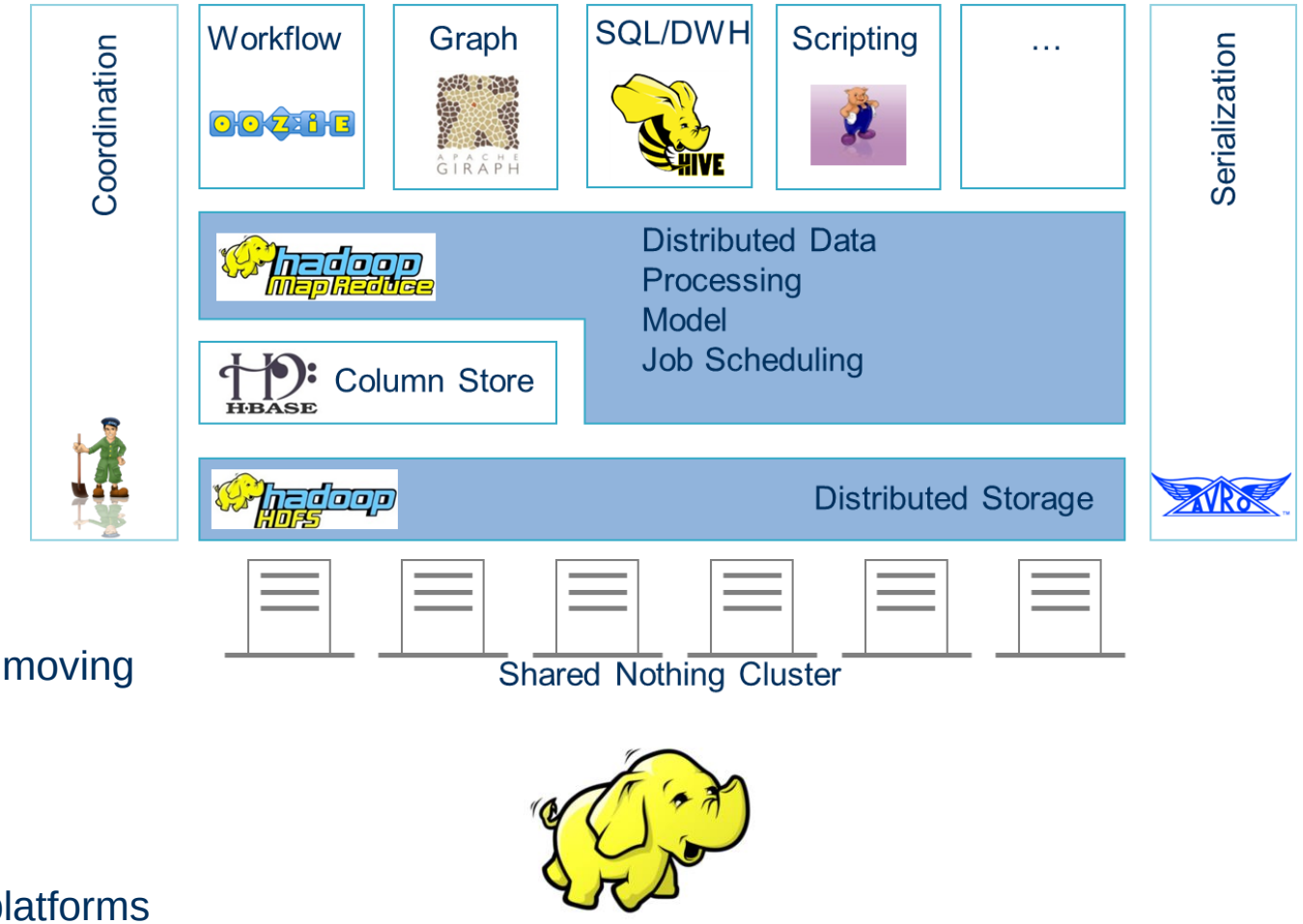
MapReduce: Simplified Data Processing on Large Clusters (Jeff Dean, S. Ghemawat. OSDI 2004)

Hadoop is an open-source MR-implementation provides complete software framework

- supports data-intensive distributed applications and clones the Google‘s MapReduce
- designed to process very large amounts of unstructured and complex data
- designed to run on a large number of machines

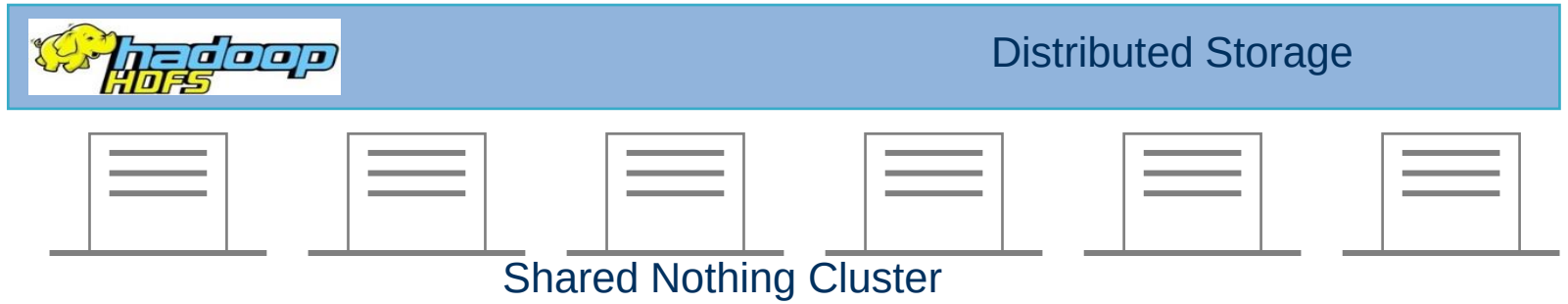
How it got started - Assumptions and goals

- Hardware failure
  - Hundreds to thousands of commodity servers per cluster
  - Failure is the norm rather than the exception
- Moving computation to data
  - Computation „near“ the data it operates on
  - Knowledge about data location is needed
- Large data sets
  - Tens of millions of files each with gigabytes to terabytes in size
- Distributed data access
  - Applications need fast access to the data, avoid moving data
  - Batch processing rather than interactive use
- Portability
  - Run on heterogeneous hardware and software platforms



HDFS

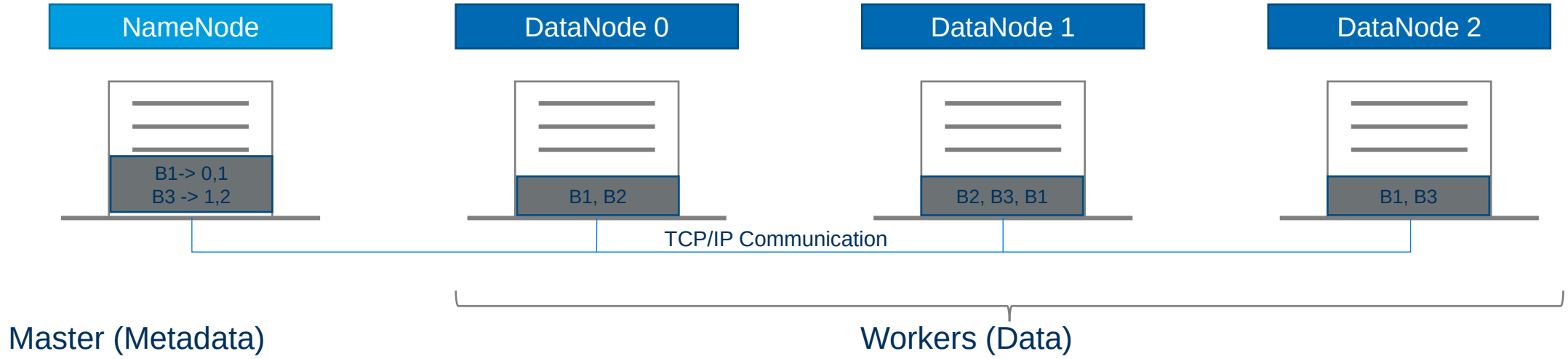
- File system component of Hadoop
- Open source under Apache License, version 2.0
- Initially created at Yahoo!
- Written entirely in Java
- Typically runs on GNU/Linux operating systems



HDFS - Blocks

HDFS files are broken into block-sized chunks

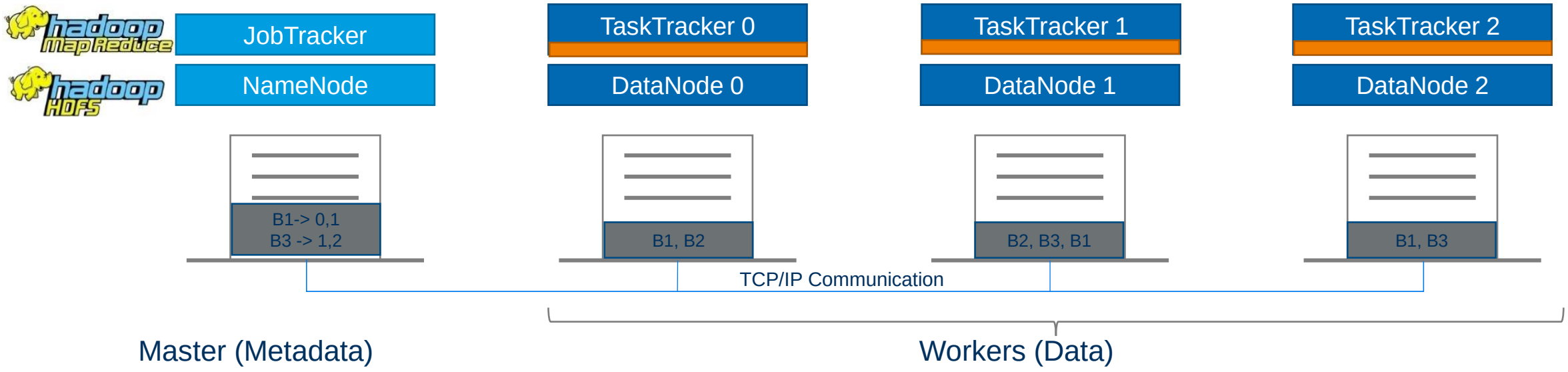
- Blocks are basic unit for replication (no RAID)
- Block is input for a processing task
- MyBigFile → B1, B2, B3



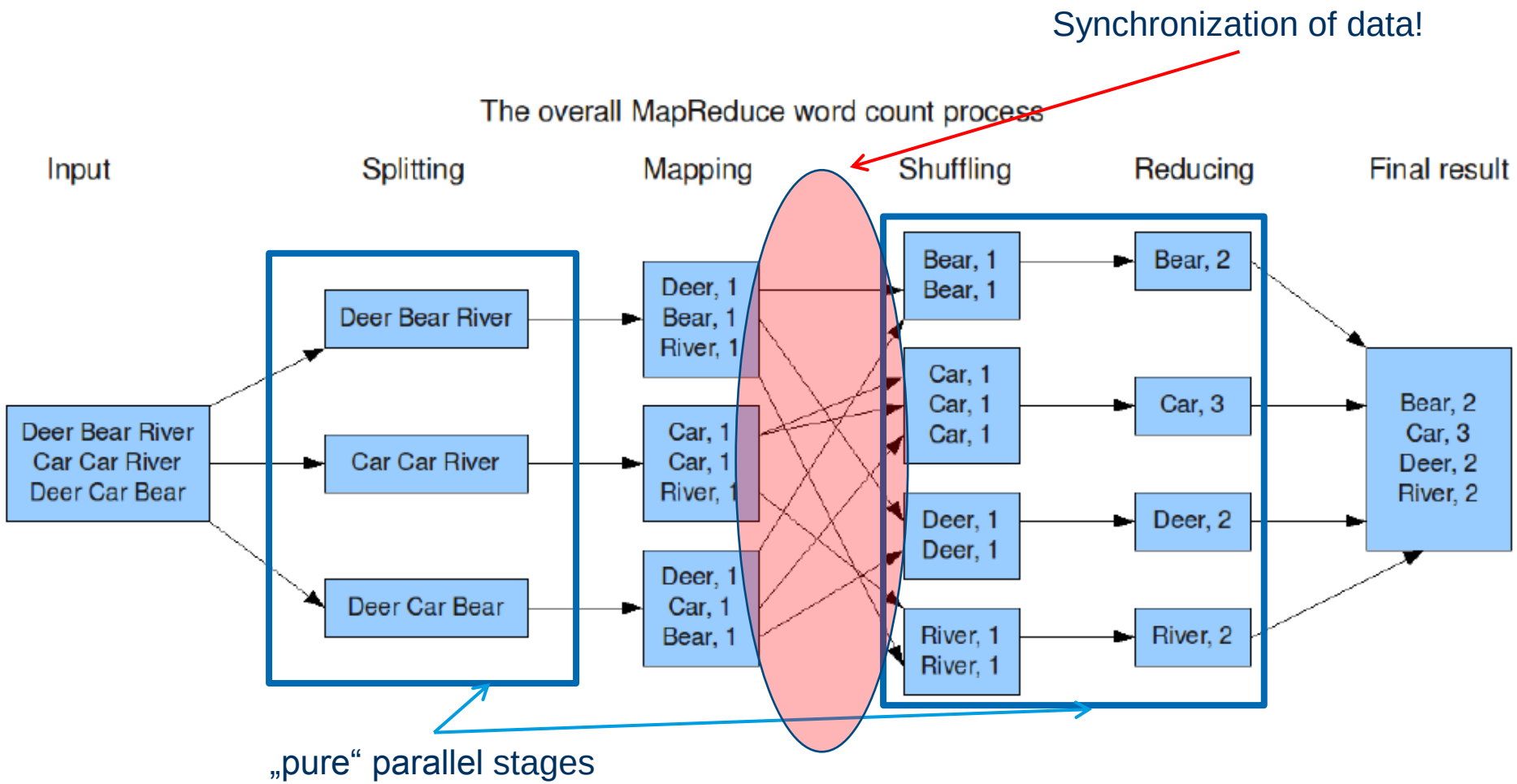
Map/Reduce

Map/Reduce is process layer on top of distributed storage

- Spread task on machines to be processed in parallel on data
- Map: Data is put to a number of machines, Output is partitioned (grouped) by a key
- Reduce: for each key-group, data is aggregated (reduced)



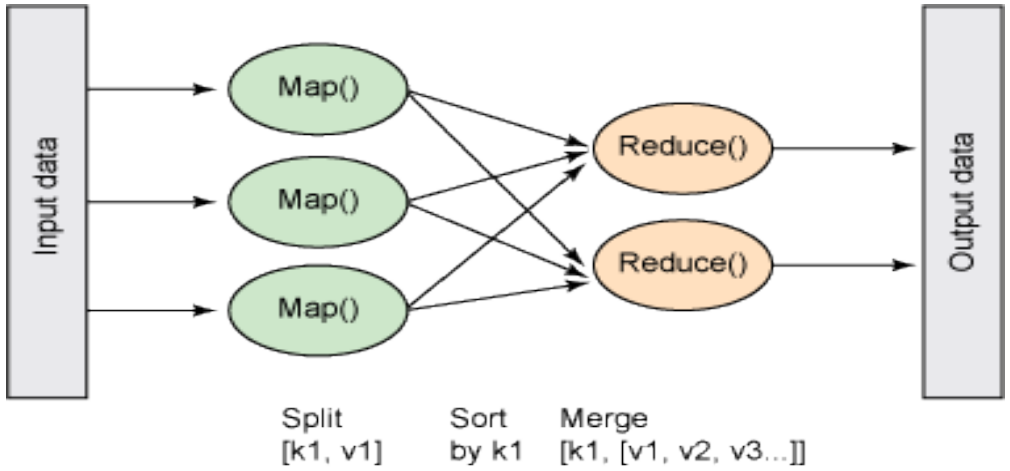
MapReduce example

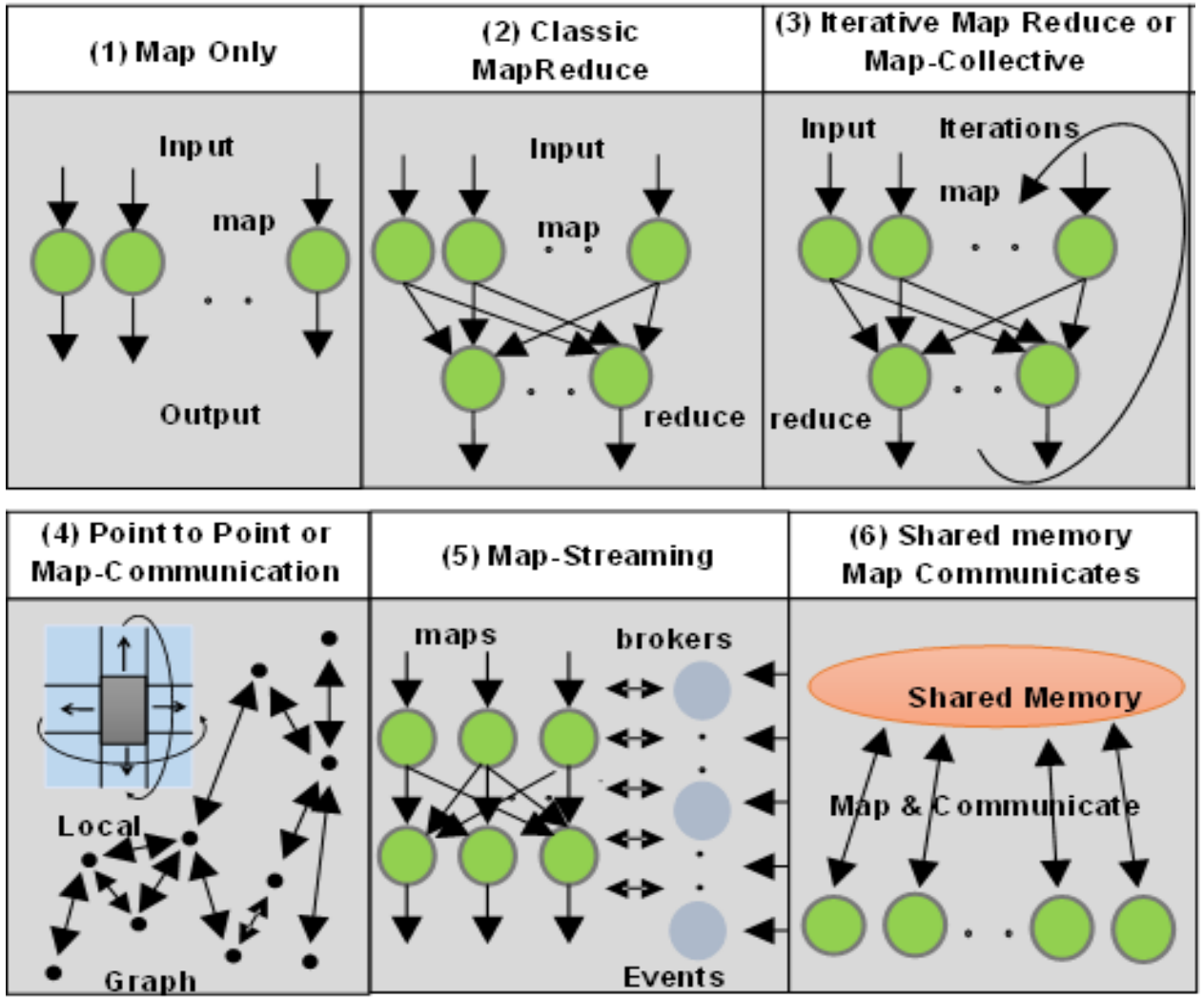


MapReduce – Limitations!

Widely adopted processing model, but:

- Non-Interactive
- Simple programming model but complex to transform tasks in map reduce model
- No iterations





[Geoffrey Fox: Lecture: Building a Library at the Nexus of High Performance Computing and Big Data, Indiana University]

MapReduce criticism (from the DBMS perspective - 2008)

Criticism from DB research perspective:

- programming paradigm for large-scale data intensive applications – step backwards
- Missing core DBMS features
- Incompatible with many tools in DBMS-Worlds
- Missing features:
  - Indexing – MapReduces simple has no index, only “brute force”
  - Updates -- to change the data in the data base
  - Transactions -- parallel update &recovery from failures during update
  - Integrity constraints and references
  - Views -- schema change without rewriting the application program

https://homes.cs.washington.edu/~billhowe/mapreduce_a_major_step_backwards.html

Apache HIVE – the data warehouse for MapReduce

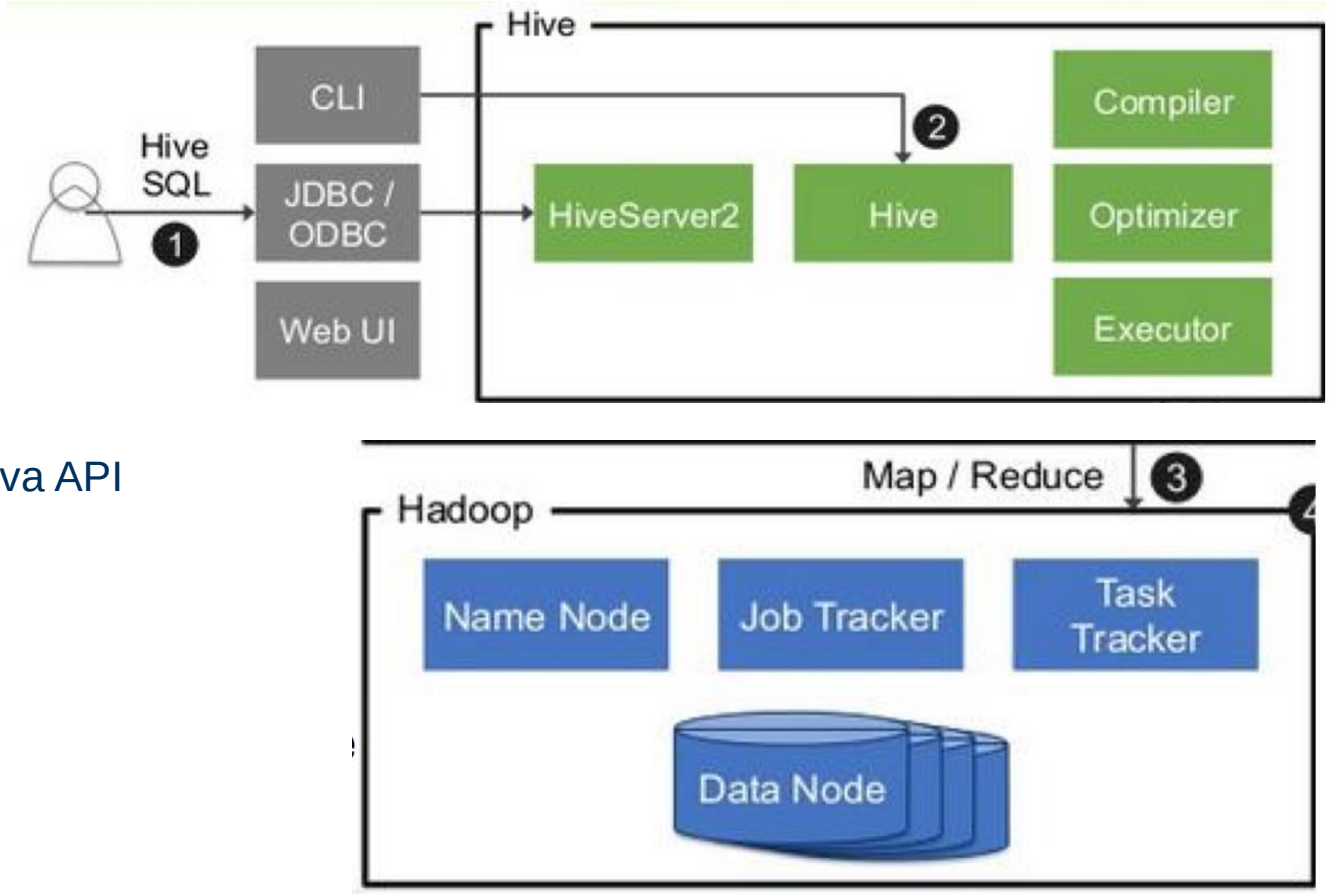
Strongly influenced by data warehouse concepts

Hive = MapReduce + SQL

- SQL simple to use
- Keep MapReduce scalability and fault tolerance mechanism

HiveQL = SQL-like query language

- No need to implement queries in the low-level Java API
- Extensible with MapReduce-Scripts



1. User issues SQL query
2. Hive parses and plans query
3. Query converted to MapReduce
4. MapReduce tasks run by Hadoop

Hive query usage

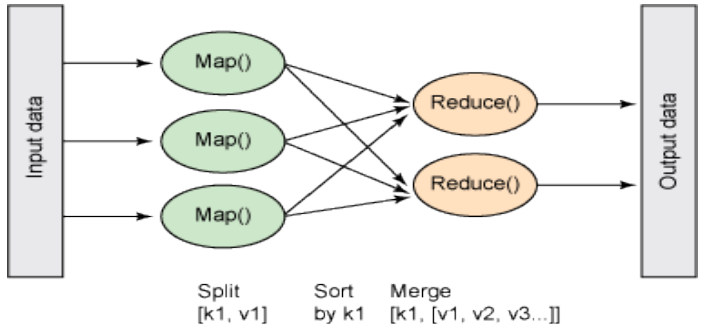
Hive is still widely used as extension in the Hadoop ecosystem (first released 2010)

- Initially developed by Facebook
- Still used by other companies (e.g. Netflix and FINRA (Financial Industry Regulatory Authority))
- Amazon developed on version of Hive for its Amazon Web Services to be included into Amazon Elastic MapReduce

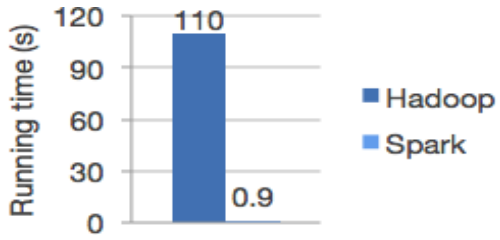
```
DROP TABLE IF EXISTS docs;
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'input_file' OVERWRITE
INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
 (SELECT explode(split(line, '\s')) AS
  word FROM docs) temp
GROUP BY word
ORDER BY word;
```

Data-driven analytics – types of analytics scenarios

Batch-processing



In-memory computing



Stream-processing

Batch oriented:

— Ask a specific question to bunch/collection of data

In-Memory processing:

— Load all data into memory and perform multiple operations on it (avoid staging data (out) if possible)

Stream Processing:

— Analyze continuously changing data, e.g. over time or in its specific content or interpretation, where historical background is not important

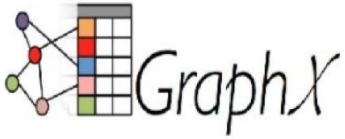
Big Data 2.0 processing platforms

General Purpose

Not a complete list!

Big SQL

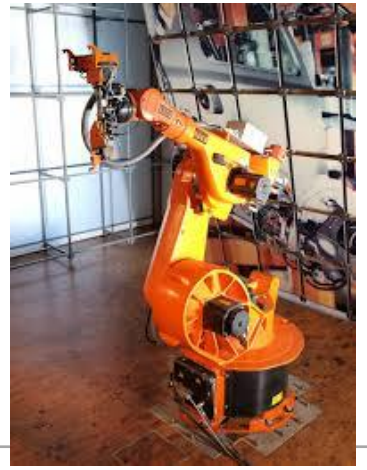
Big Graph



Big Stream

Big data streaming

Availability of sensors & cyber physical Systems



Big data streaming – use case traffic monitoring

Wide range of applications:

— Toll collection, Speed control, traffic-jam detection, route planning

Data collection:

— Toll stations (fixed), smart phones, logistics tracking (GPS)

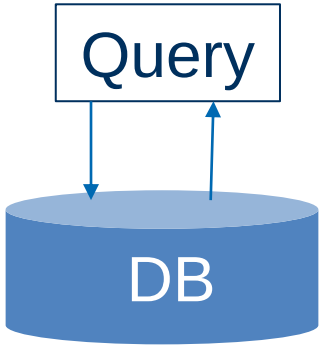
Stream Processing Model

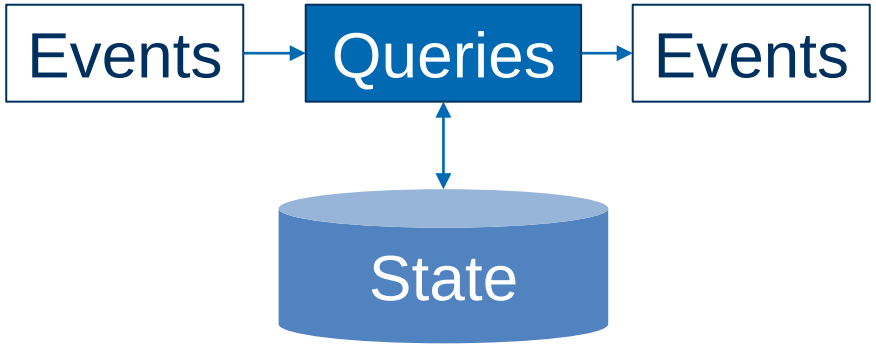
Batch Processing

- Data is „fixed“
- Workload of queries run into the system

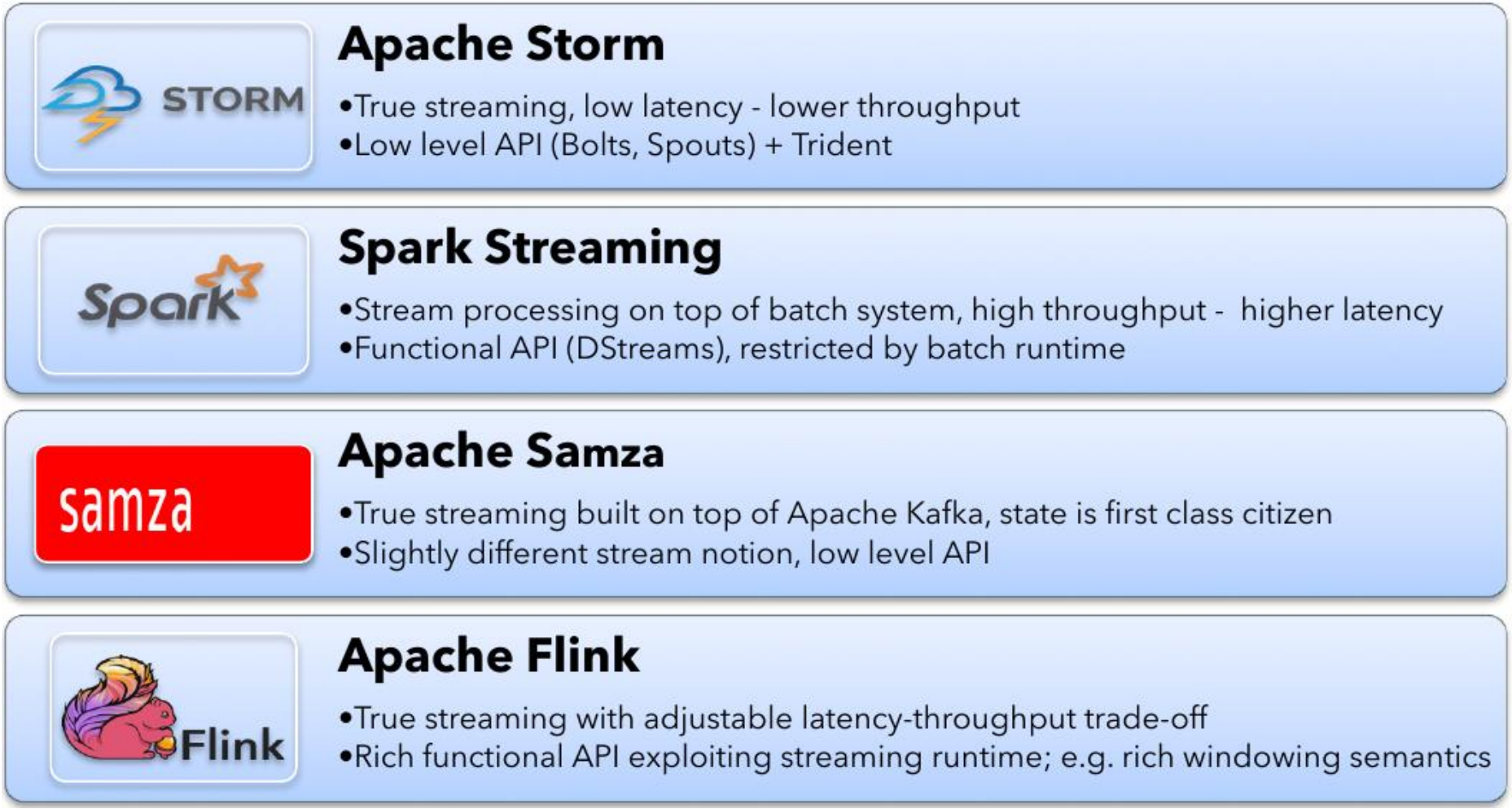
Stream Processing

- Continuous (standing) query
- Data is streaming through the system
- Unbounded stream of data
- Limited resources for state





Stream Processing Alternatives



Younger Generation “Hadoop“

APACHE FLINK

APACHE SPARK

More Operators

Streaming Support

In-Memory, for interactive applications

Built in machine learning integration

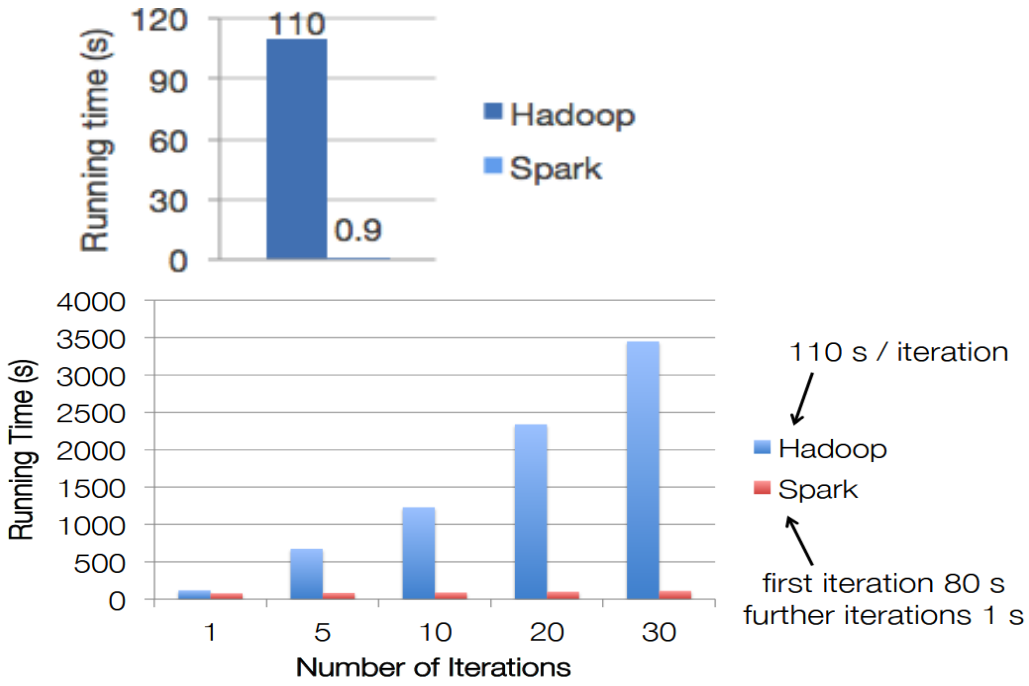
...

Apache Spark

http://spark-project.org

- Apache Spark offers Directed Acyclic Graph (DAG) execution engine that supports **cyclic data flow** and **in-memory computing**.
- Offers over 80 high-level operators to build **parallel applications**
- Interactive use from the Scala, Java, Python and R shells
- Combine libraries seamlessly in the same application
- Completely written in Scala

Logistic regression in Hadoop and Spark



Apache Spark

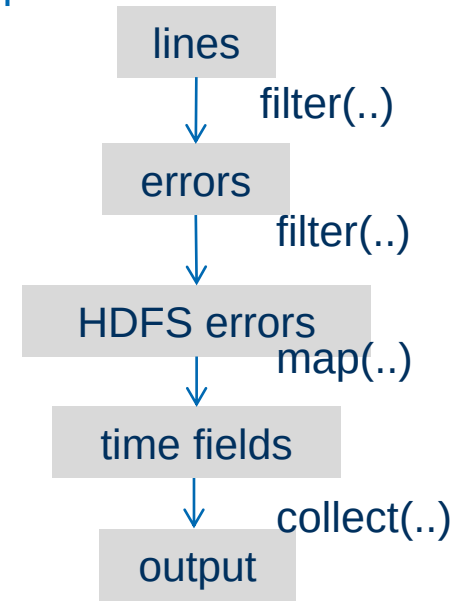
http://spark-project.org

RDD

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.filter(_.contains("HDFS")).map(_.split('\t')(3)).collect()
```

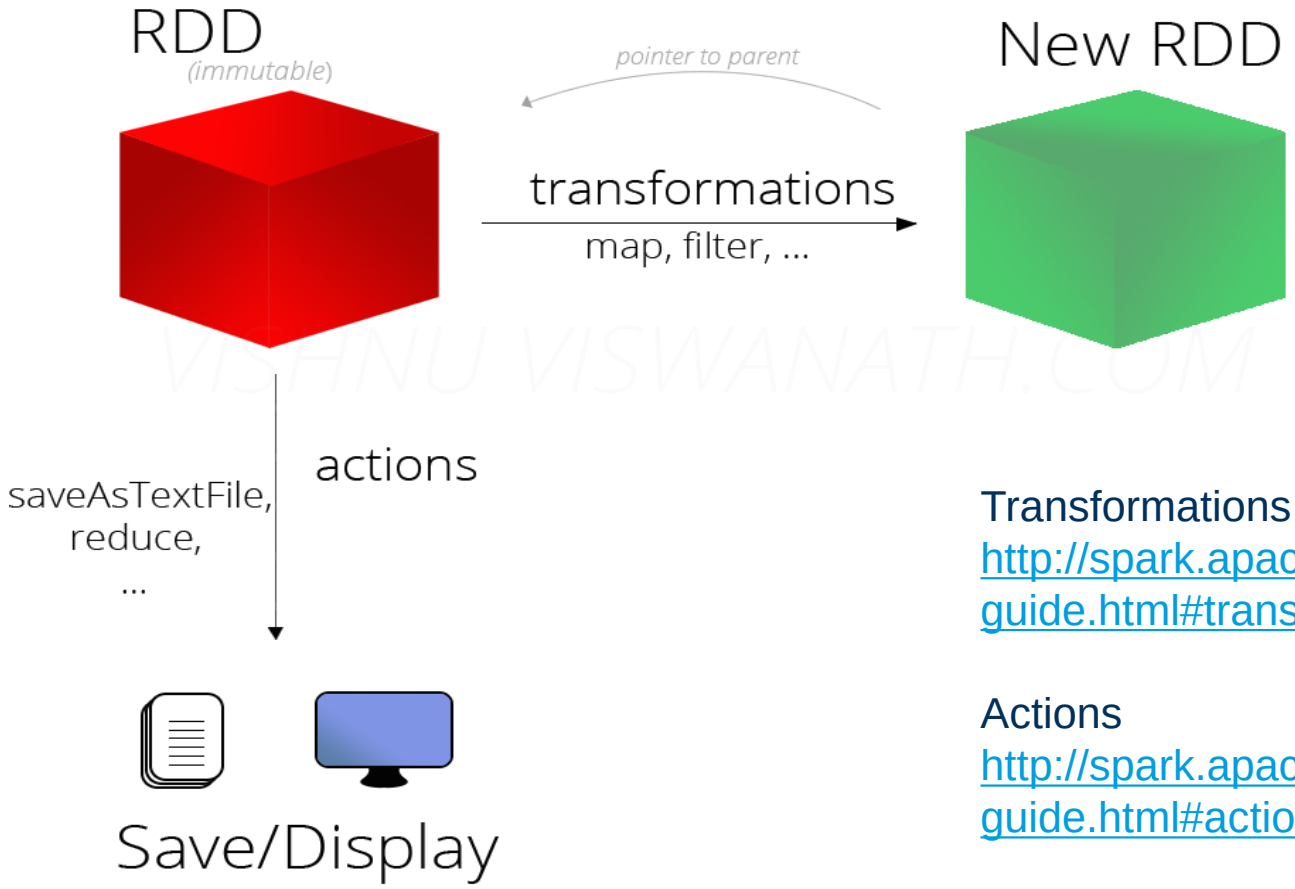
Transformation

Action



- Apache Spark offers Directed Acyclic Graph (DAG) execution engine that supports cyclic data flow and in-memory computing
- RDD: Resilient Distributed Dataset
  Sparks core concept

RDD – Resilient Distributed Dataset



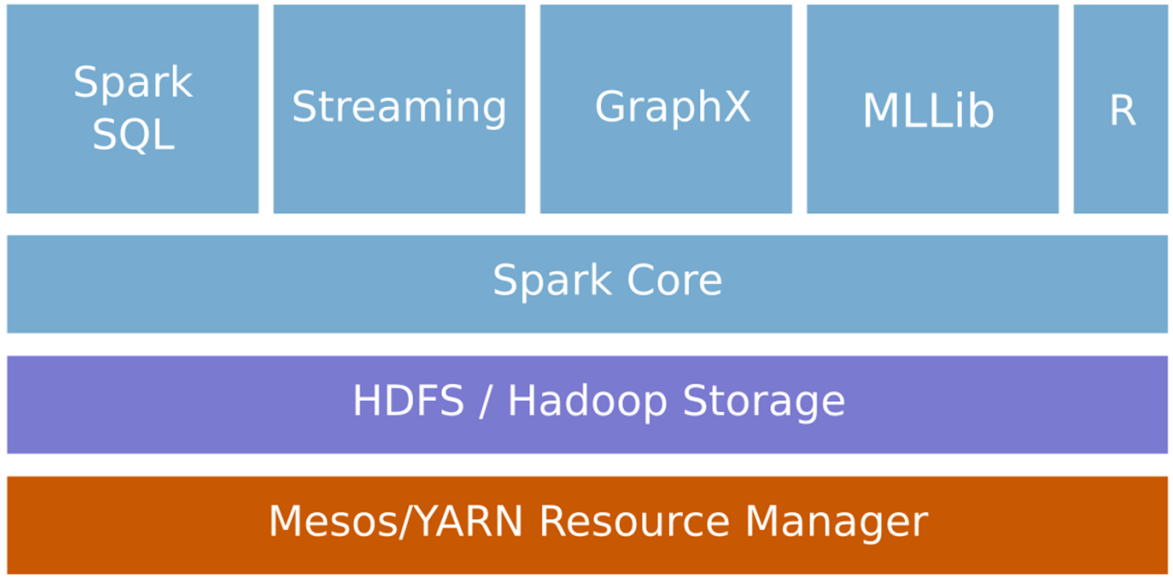
Transformations:
http://spark.apache.org/docs/latest/rdd-programming-guide.html#transformations

Actions
http://spark.apache.org/docs/latest/rdd-programming-guide.html#actions

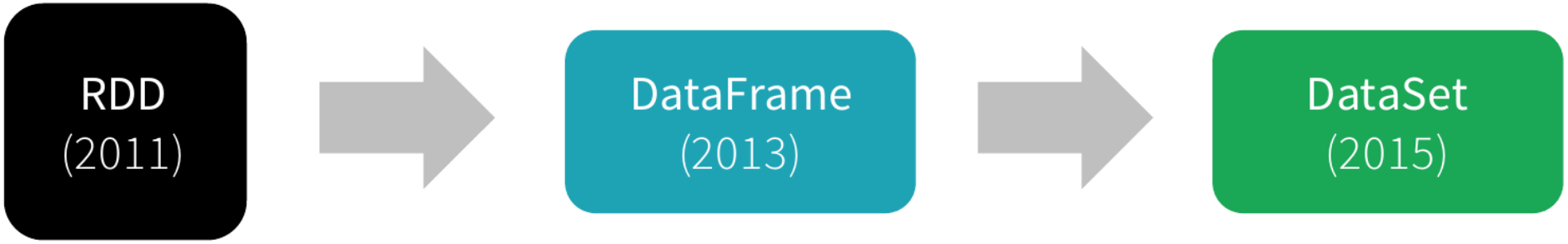
Apache Spark

http://spark-project.org

- Offers stack of libraries for **analytics** including SQL and DataFrames, MLlib for **machine learning**, **GraphX**, and Spark **Streaming**
- Data access can be realized via HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source
- Run Spark standalone or in cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos



Data structure evolution



- Distribute collection of JVM objects
- Functional Operators (map, filter, etc.)

- Distribute collection of Row objects
- Expression-based operations and UDFs
- Logical plans and optimizer
- Fast/efficient internal representations

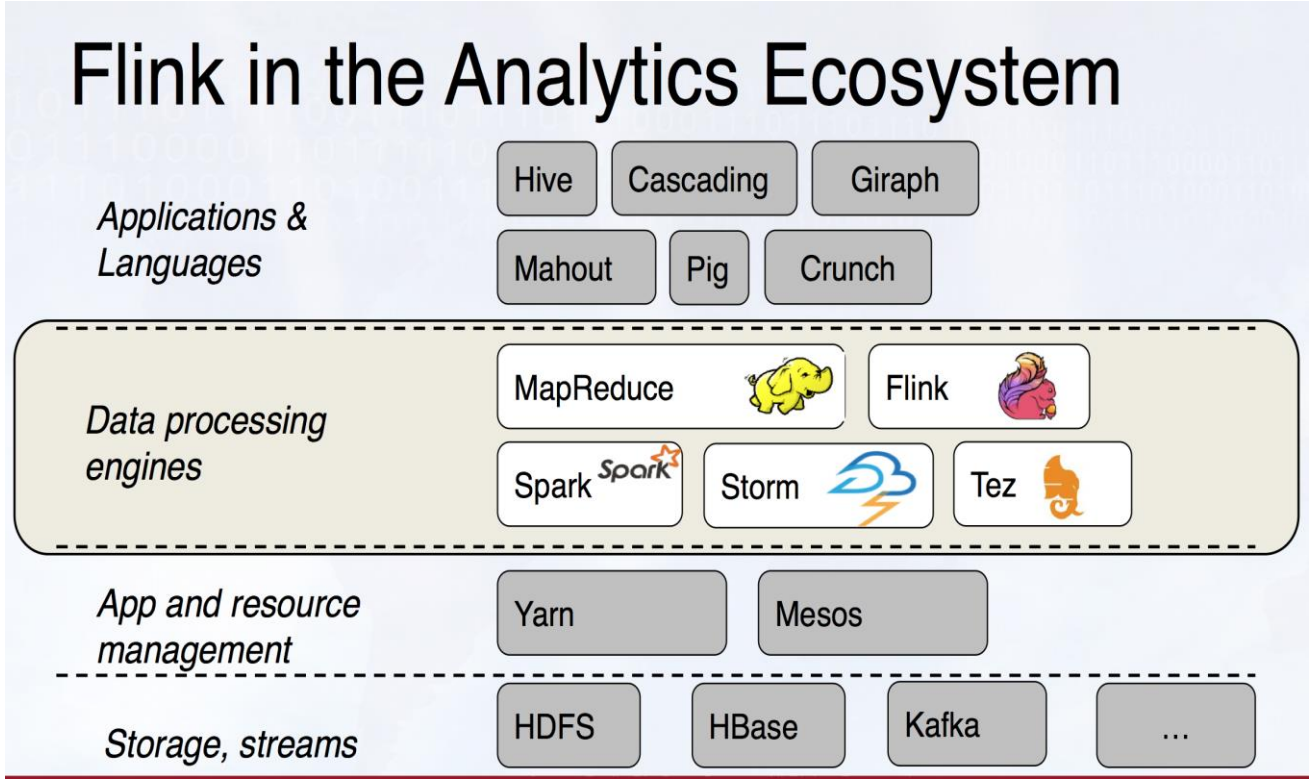
- Internally rows, externally JVM objects
- "Best of both worlds" **type safe + fast**

Jump Start into Apache® Spark™ and Databricks; Denny Lee @Databricks

Apache Flink

Large-scale data processing engine – similar to Spark

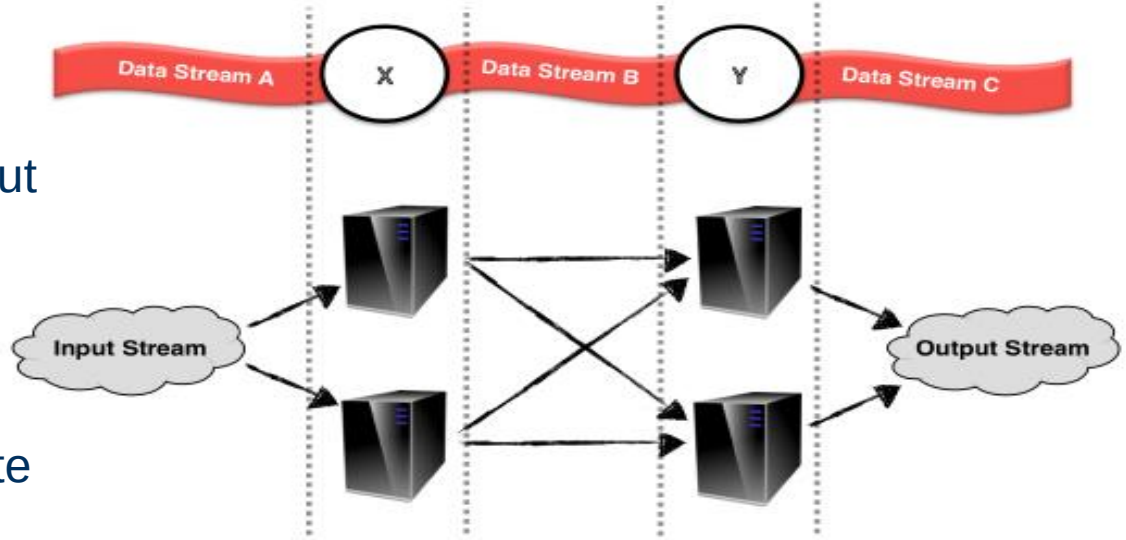
- Derived from early developments in Stratosphere project
- Many similarities to Apache Spark
- Support for iterations (machine learning), graph and batch processing
- Good support for streaming
- Streaming of results between operators

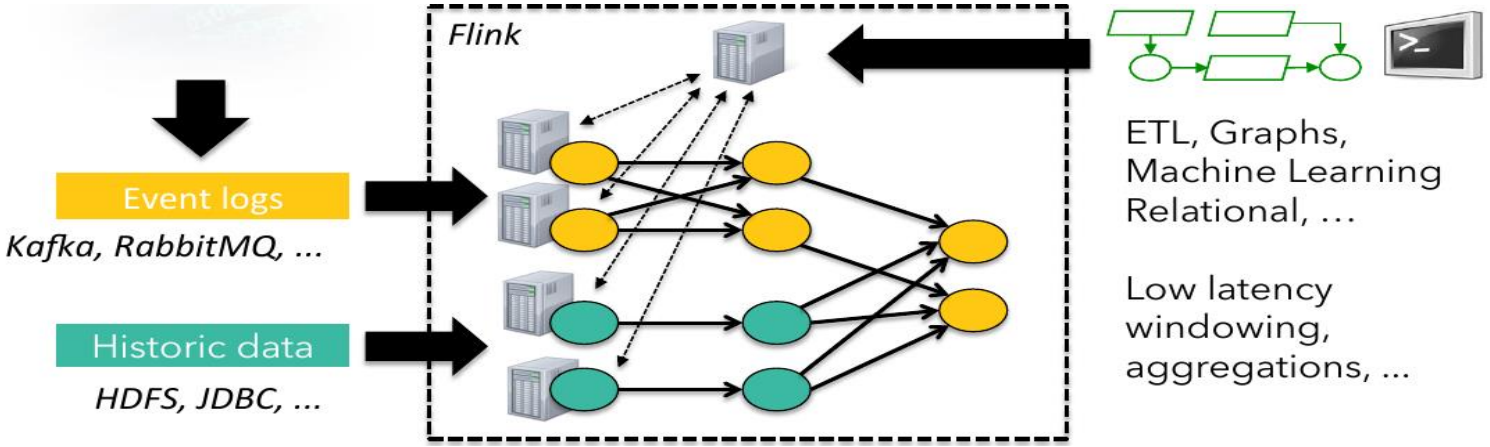


source: Dr. Tilmann Rabl

Stream Processing: Apache Flink

- True streaming with adjustable latency and throughput
- Rich functional API exploiting streaming runtime
- Flexible windowing semantics
- Exactly-once processing guarantees with (small) state
- Data stream outputs

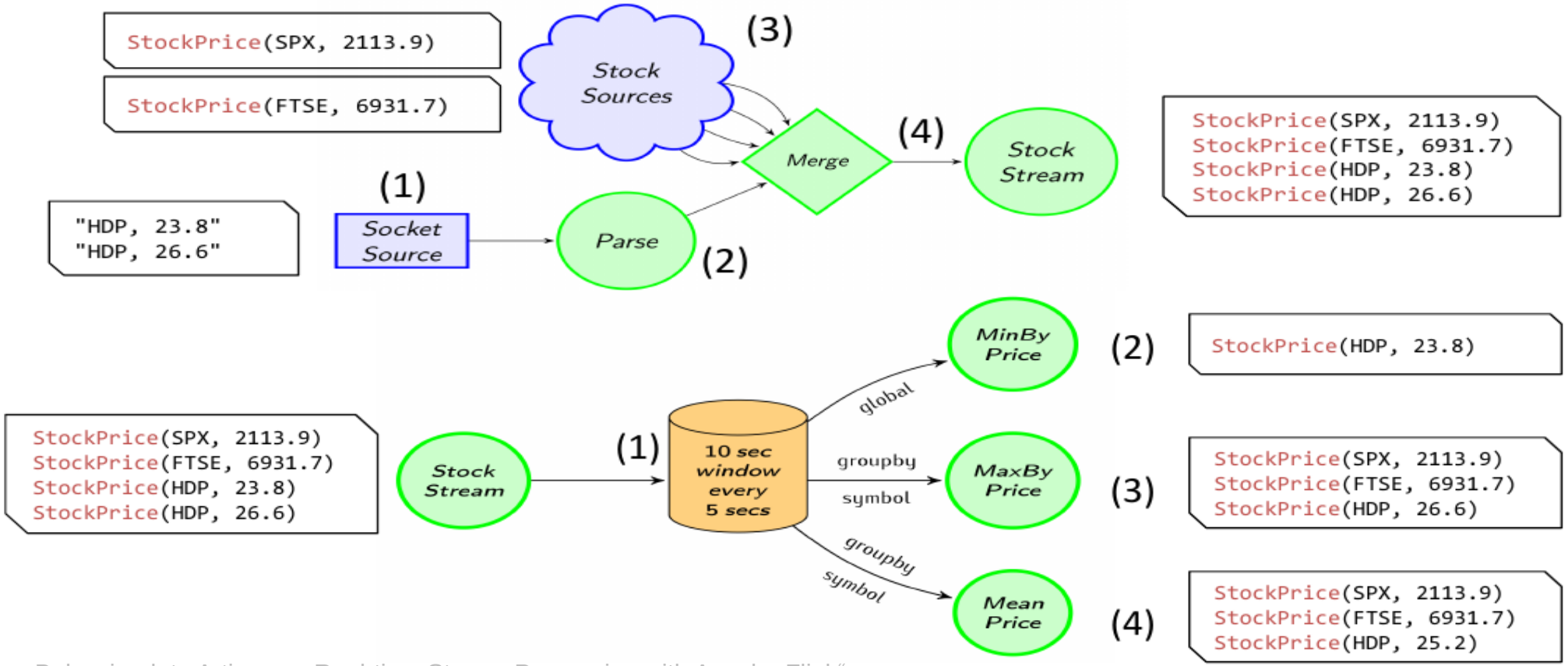




Marton Balassi – data Artisans: «Real-time Stream Processing with Apache Flink“

Stream Processing: Apache Flink - Examples

— Scala examples: Reading from data sources



Marton Balassi – data Artisans: «Real-time Stream Processing with Apache Flink“

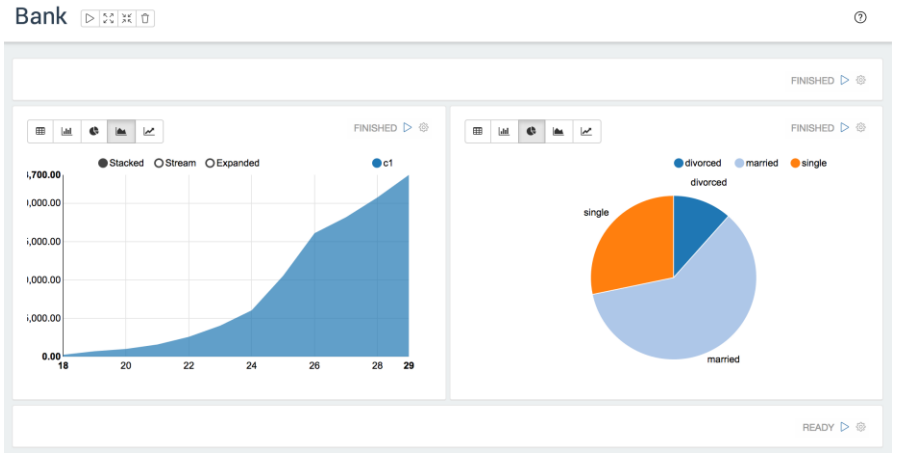
Road ahead

Road Ahead – Technology Proliferation!

Road ahead: Simplification / Interaction

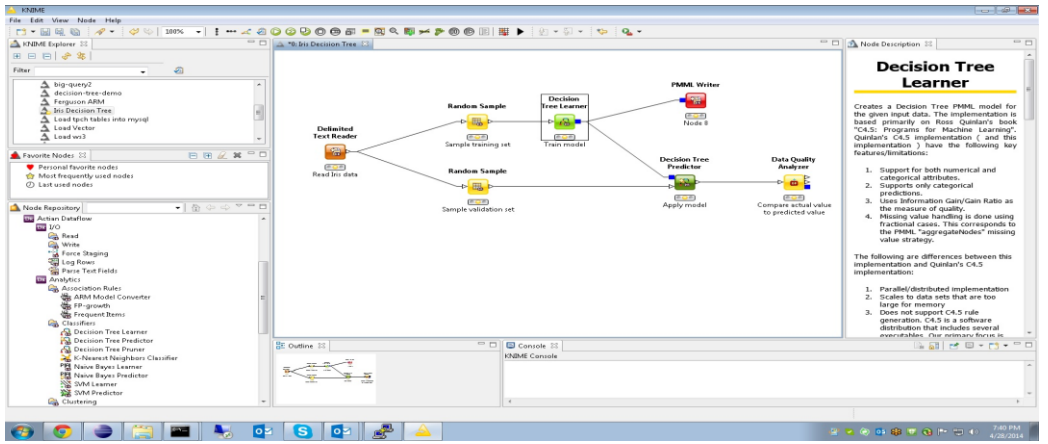
Apache Zeppelin

- Data visualization
- Spark integration
- Collaboration



KNIME

- Visual programming of Big Data Workflows
- Comes with add-on to connect to Hadoop cluster
- Open for other cluster integration options (e.g. UNICORE)



Road AHEAD: Simplification(2)

Docker for Cluster/Task Management

Docker is an open-source project started 2013, combining existing technologies for a **lightweight virtualization** with **modularization** and **portability** in mind.

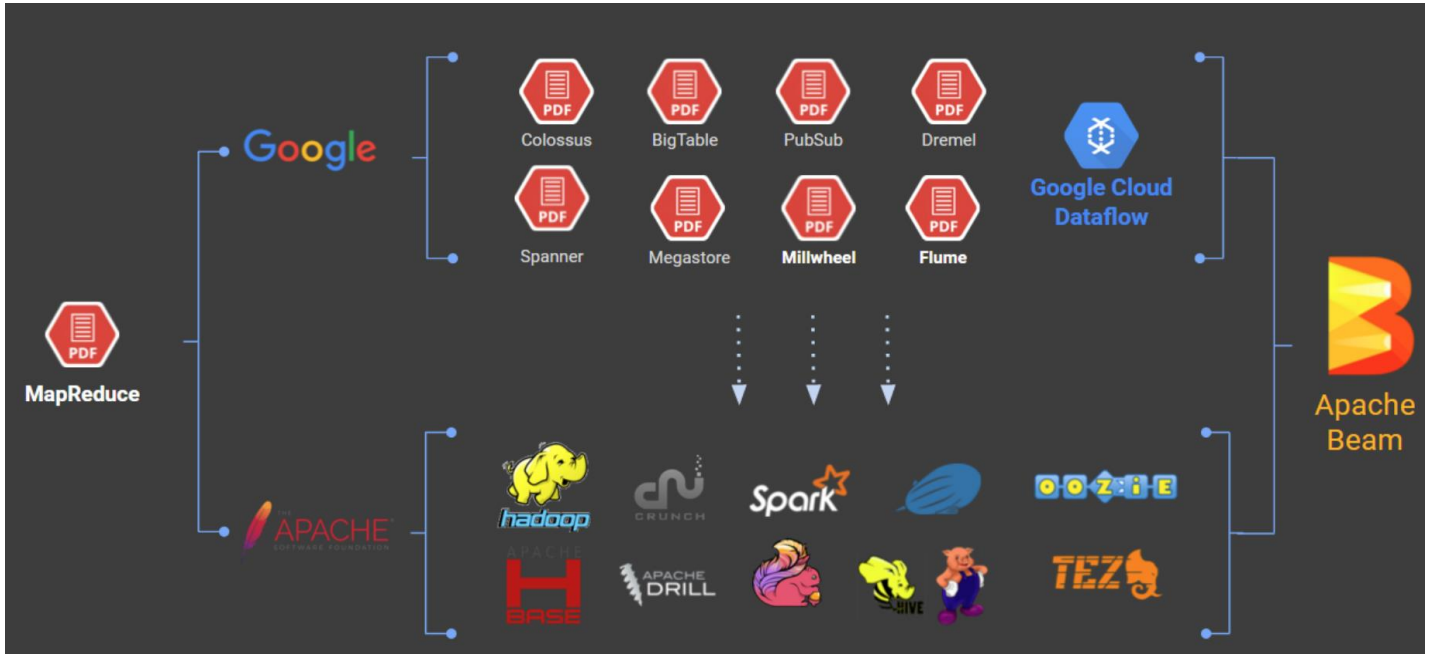
“If it can run on the host, it can run in the container”
(Source: Docker Introduction Slides November 2013)

Road AHEAD: Unification and specialization

API for Java, Python and Go

Current Apache Beam Pipeline Runners:

- Apache Apex
- Apache Flink
- Apache Gearpump (incubating)
- Apache Samza
- Apache Spark
- Google Cloud Dataflow



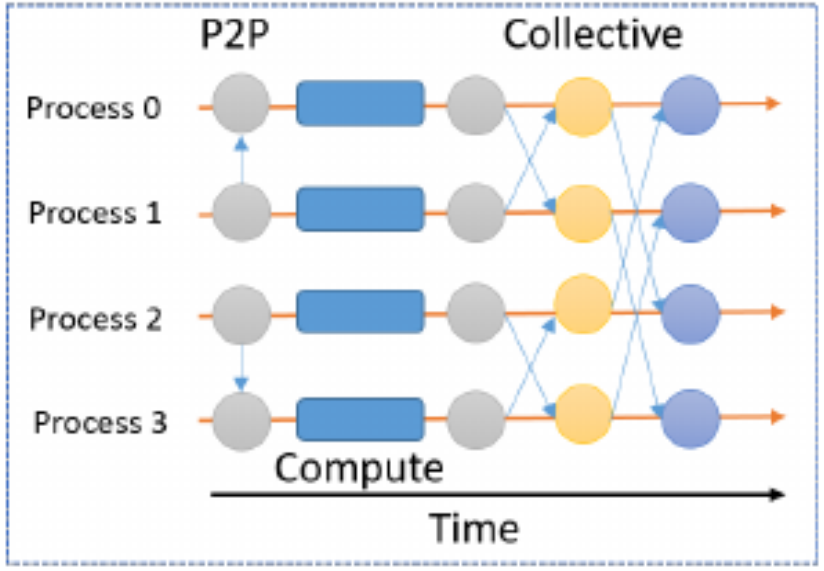
Specialization:

- particularly useful for “Embarrassingly Parallel2 data processing tasks
- use Beam for Extract, Transform, and Load (ETL) tasks and pure data integration.

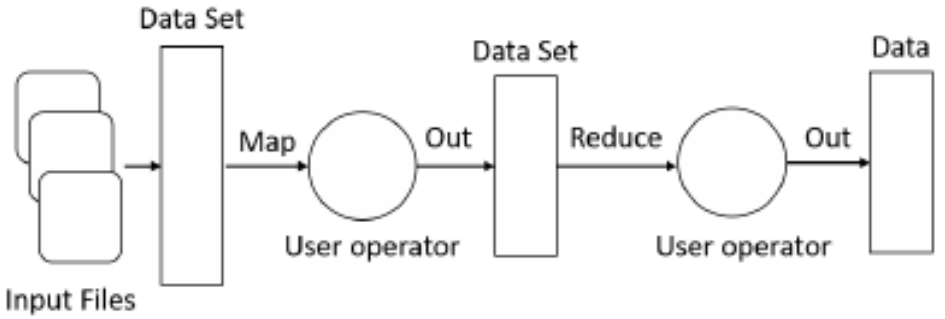
Performance Characterization of different workloads

Differences in data flow and execution between big data and MPI-based workloads

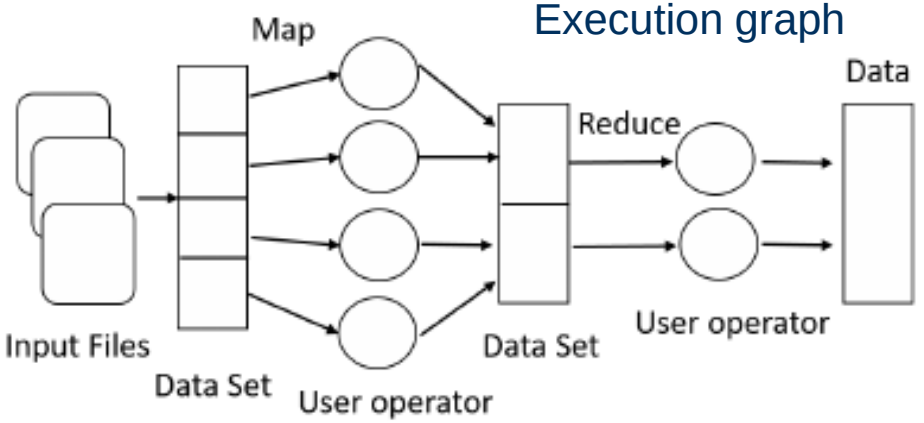
MPI execution flow



Logical graph of data flow



Execution graph



Supun Kamburugamuve, Pulasthi Wickramasinghe, Saliya Ekanayake, and Geoffrey C Fox, Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink , The International Journal of High Performance Computing Applications, Vol 32, Issue 1, pp. 61 - 73

Performance Characterization of different workloads

Comparison of ML algorithms in Apache Spark, Apache Flink and MPI :

- Multi-dimensional scaling (MDS) algorithms are e.g. used for projecting high dimensional data into a lower dimension; Computational expensive ~O(N^2)
- Often used to visualize high dimensional data by projecting into 3D or 2D models

Implementations:

- **Java OpenMPI**: involves intense gather/reduce of large matrices; uses shared memory to reduce cost of collective operations; utilizes modern multi-core machines.

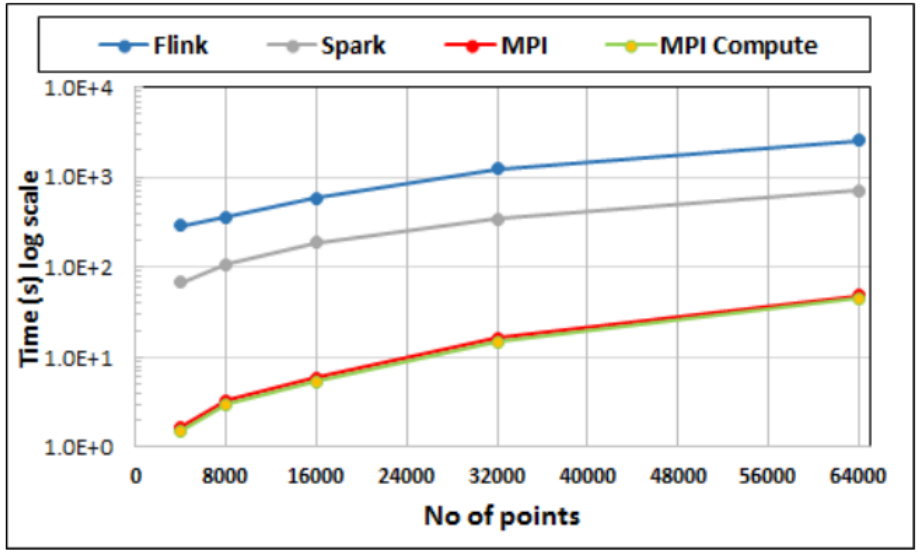


Fig. 8. MDS execution time on 16 nodes with 20 processes in each node with varying number of points

Supun Kamburugamuve, Pulasthi Wickramasinghe, Saliya Ekanayake, and Geoffrey C Fox, Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink , The International Journal of High Performance Computing Applications, Vol 32, Issue 1, pp. 61 - 73

Performance Characterization of different workloads

Comparison of ML algorithms in Apache Spark, Apache Flink and MPI :

- Multi-dimensional scaling (MDS) algorithms are e.g. used for projecting high dimensional data into a lower dimension; Computational expensive ~O(N^2)
- Often used to visualize high dimensional data by projecting into 3D or 2D models

Implementations:

- **Spark**: programming model does not allow tasks to communicate with one another: all-to-all collectives are not supported

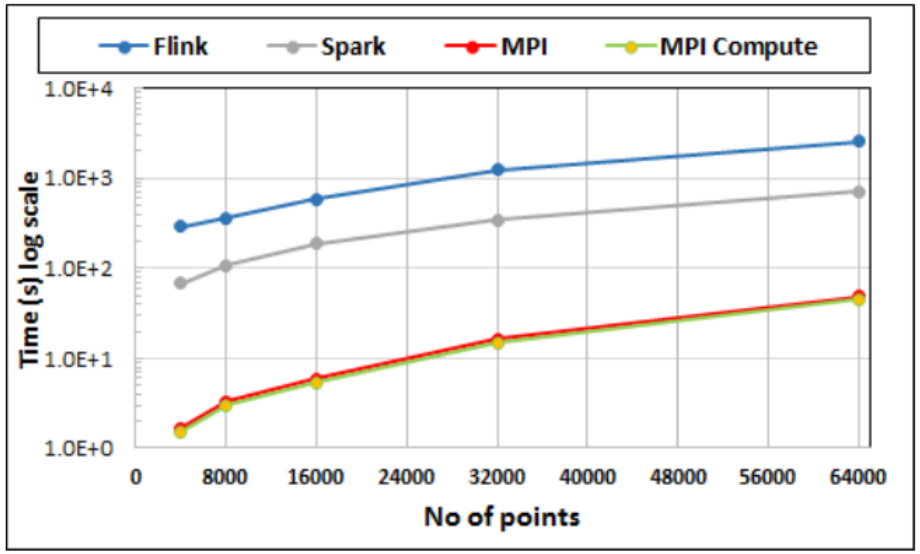


Fig. 8. MDS execution time on 16 nodes with 20 processes in each node with varying number of points

Supun Kamburugamuve, Pulasthi Wickramasinghe, Saliya Ekanayake, and Geoffrey C Fox, Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink , The International Journal of High Performance Computing Applications, Vol 32, Issue 1, pp. 61 - 73

Performance Characterization of different workloads

Comparison of ML algorithms in Apache Spark, Apache Flink and MPI :

- Multi-dimensional scaling (MDS) algorithms are e.g. used for projecting high dimensional data into a lower dimension; Computational expensive ~O(N^2)
- Often used to visualize high dimensional data by projecting into 3D or 2D models

Implementations:

- **Flink**: doesn't support nested iterations, only inner loop is implemented as a data-flow; uses separate jobs for outer loops

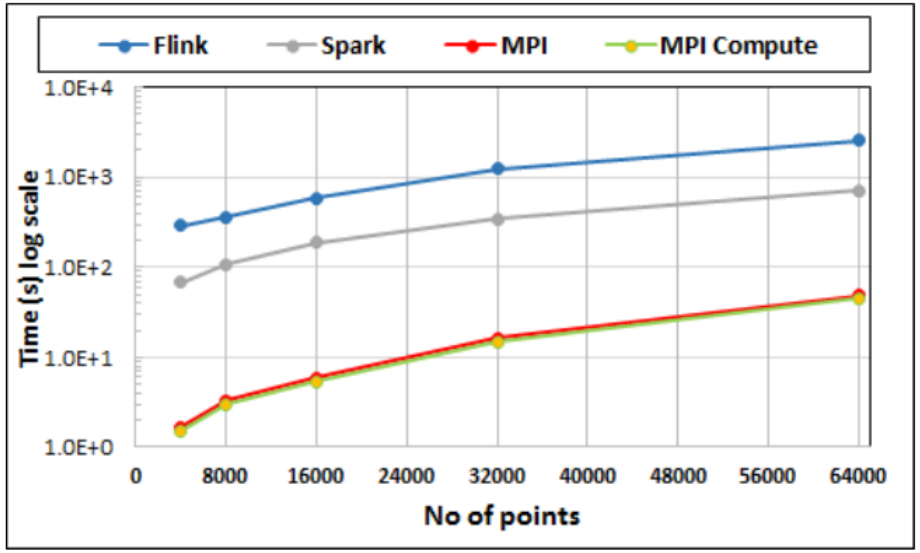


Fig. 8. MDS execution time on 16 nodes with 20 processes in each node with varying number of points

Supun Kamburugamuve, Pulasthi Wickramasinghe, Saliya Ekanayake, and Geoffrey C Fox, Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink , The International Journal of High Performance Computing Applications, Vol 32, Issue 1, pp. 61 - 73

Performance Characterization

Comparison of K-Means implementation in Apache Spark, Apache Flink and MPI

- Flink/Spark: show better performance, especially with lower number of centers
- Runtime depends on number of points and centroids
- Depending on analysis settings, behavior for general purpos frameworks changes stronger

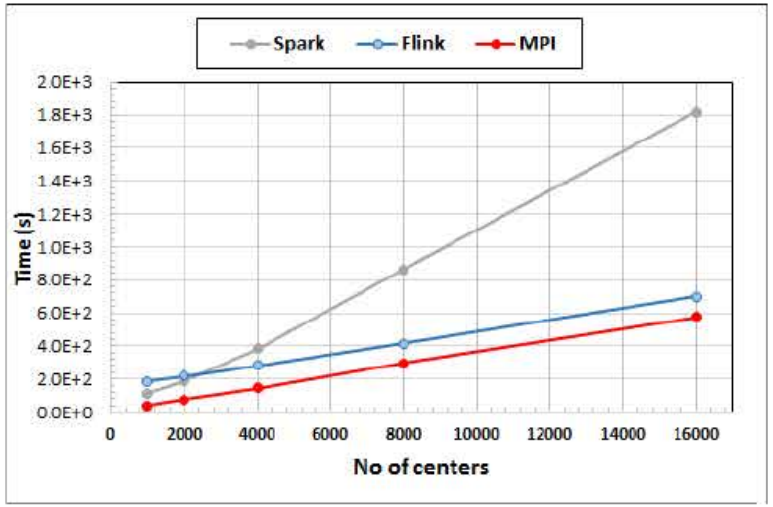


Fig. 10. K-Means execution time on 16 nodes with 20 parallel tasks in each node with 10 million points and varying number of centroids. Each point has 100 attributes.

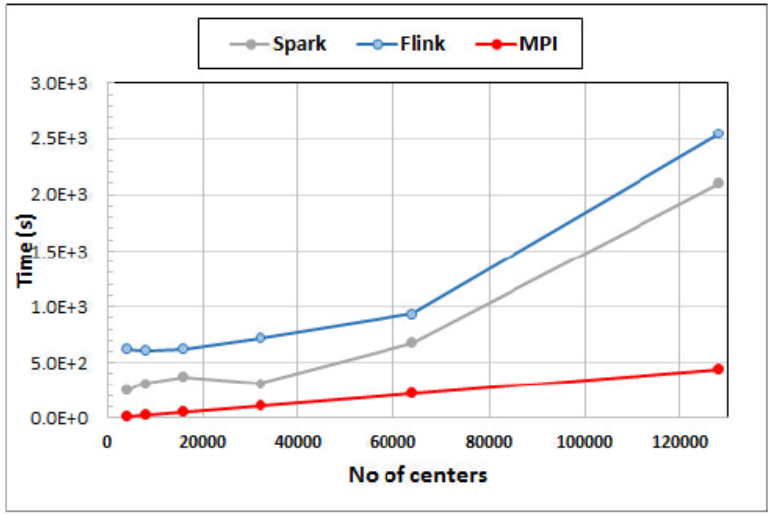


Fig. 12. K-Means execution time on 8 nodes with 20 processes in each node with 1 million points and varying number of centroids. Each point has 2 attributes.

Supun Kamburugamuve, Pulasthi Wickramasinghe, Saliya Ekanayake, and Geoffrey C Fox, Anatomy of r Spark, and Flink , The International Journal of High Performance Computing Applications, Vol 32, Issue 1

Performance Characterization

Example: comparison of Terasort implementation in Apache Spark, Apache Flink and MPI

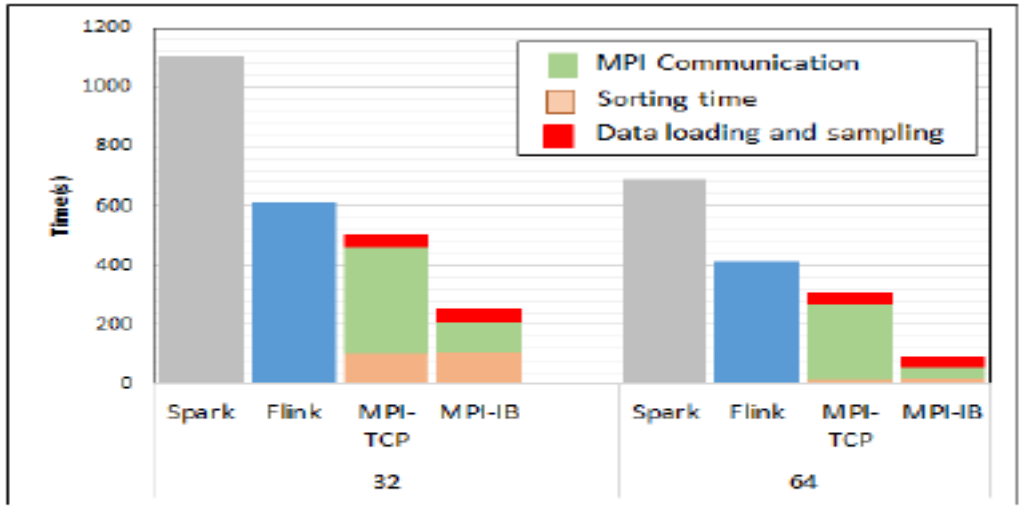


Fig. 14. Terasort execution time in 64 and 32 nodes. Only MPI shows the sorting time and communication time as other two frameworks doesn't provide a viable method to accurately measure them. Sorting time includes data save time. MPI-IB - MPI with Infiniband

Supun Kamburugamuve, Pulasthi Wickramasinghe, Saliya Ekanayake, and Geoffrey C Fox, Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink , The International Journal of High Performance Computing Applications, Vol 32, Issue 1, pp. 61 - 73

Summary

Data processing 2.0 with newer family of (big) data analysis engines

- Extending limitations coming from traditional Hadoop framework
  - Streamlined on requirements coming from analytics side and different view on data
  - Still batch-processing widely use
  - Applications often just need temporary status of data – stream processing
  - Use of available hardware options for in-memory processing

Future directions:

- Interoperability and specialization of functionalities
- Fusion of concepts; BD as data preparation steps for higher-level analytics concepts (e.g. ML/DL)
- Constant improvements: Frameworks will adopt to other hardware architectures away from pure Shared-Nothing clusters
- Performance considerations have become and issue

Thanks!

Acknowledgements:
Eric Peukert, Sunna Torge, Jan Frenzel, Tilmann Rabl