Spark I

Big Data Management

Knowledge objectives

1. Name the main Spark contributions and characteristics
2. Compare MapReduce and Spark
3. Define a dataframe
4. Distinguish dataframe from relation and matrix
5. Distinguish Spark and Pandas dataframe
6. Enumerate some abstraction on top of Spark

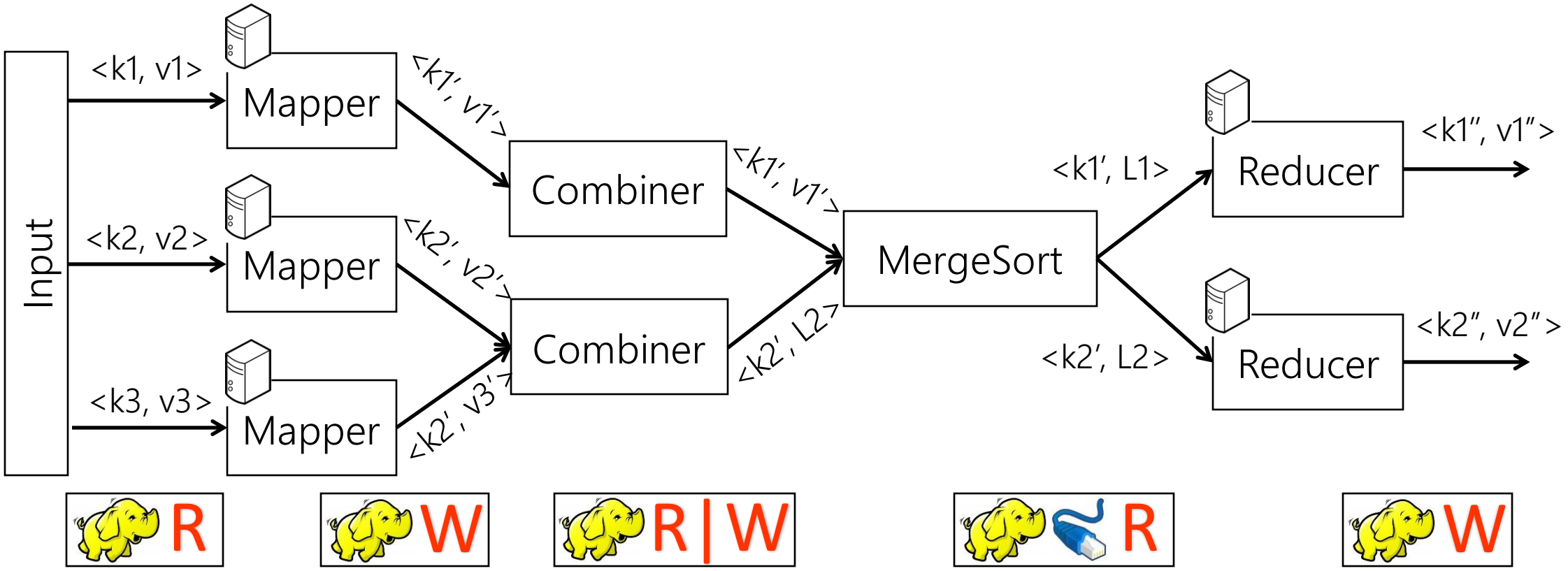
Application Objectives

- Provide the Spark pseudo-code for a simple problem using dataframes

Background

MapReduce limitations

MapReduce intra-job coordination

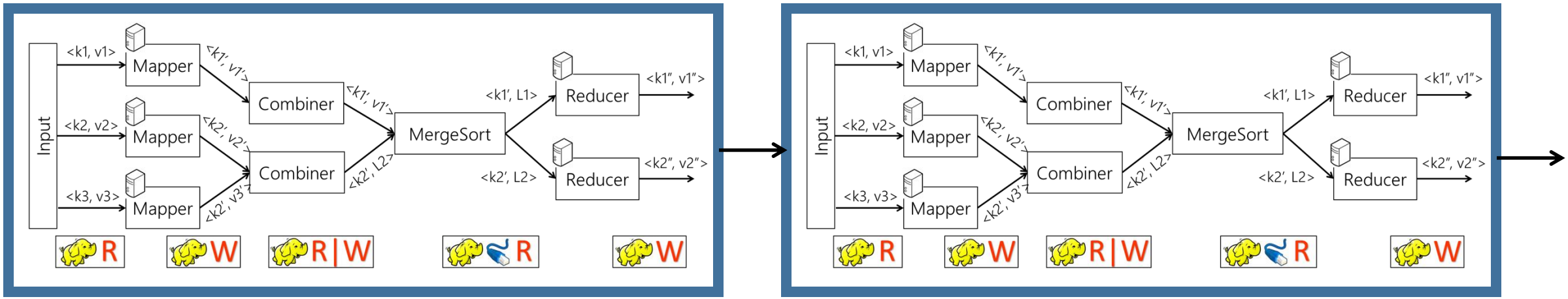


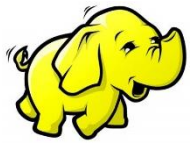
MapReduce inter-job coordination

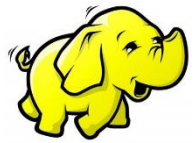
Count

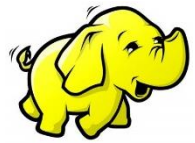
Rank

...



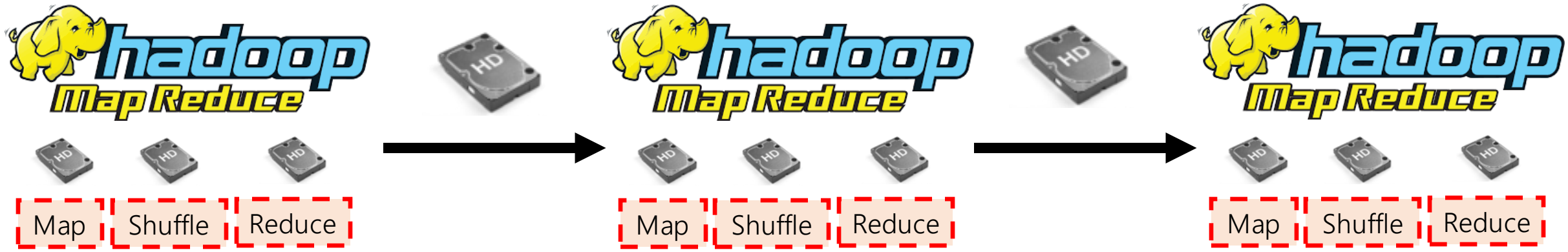




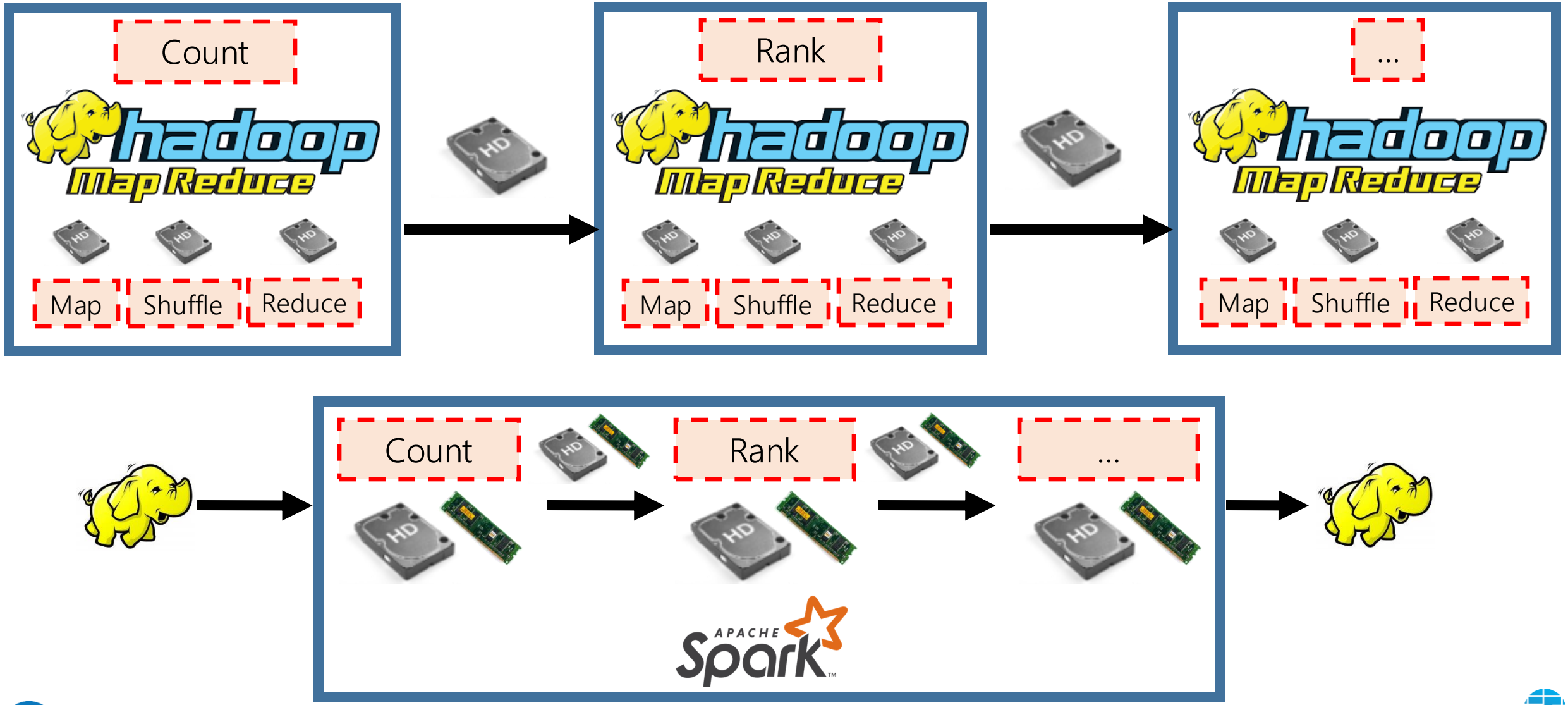


MapReduce limitations

- Coordination between phases using DFS
  - Map, Shuffle, Reduce
- Coordination between jobs using DFS
  - Count, rank, aggregate, …



Main memory coordination in Spark



Dataframes

Problems of relational tables in data exploration

- Schema needs to be defined before examining the data
- Not well-structured data is difficult to query
- Generating queries requires familiarity with the schema
- Complex declarative queries are hard to debug
  - SQL was not conceived for REPL (Read, Evaluate, Print Loop)

Characteristics of dataframes

- First introduced to S in 1990
- Symmetrical treatment of rows and columns
  - Both can be referenced explicitly
    - By position (data is ordered row- and column-wise)
    - By name
- Data has to adhere to a schema
  - Defined at runtime
    - Useful for data cleaning
- A variety of operations
  - Relational-like (e.g., filter, join)
  - Spreadsheet-like (e.g., pivot)
  - Linear algebra (e.g., multiply)
- Incrementally composable query syntax
- Native embedding in an imperative language

Relation, Dataframe and Matrix

Relation

R

T_1	…	T_n
A_1	…	A_n

Original dataframe

	T_1	T_2	…	T_n
	$1/A_1$	$2/A_2$	…	n/A_n
$1/r_1$	x	“x”	…	T
$2/r_2$	y	“y”	…	F
…	…	…		…
m/r_m	z	None	…	T

Spark dataframe

T_1	T_2	…	T_n
$1/A_1$	$2/A_2$	…	n/A_n
x	“x”	…	T
y	“y”	…	F
…	…		…
z	None	…	T

Matrix

Numeric

	1	2	…	n
1	a_{11}	a_{12}	…	a_{1n}
2	a_{21}	a_{22}	…	a_{2n}
…	…	…		…
m	a_{m1}	a_{m2}	…	a_{mn}

Spark Dataframe definition

"A Dataset is a strongly typed collection of domain-specific objects that can be transformed in parallel using functional or relational operations."

"A Dataframe is **an immutable collection of data** organized into named columns, potentially distributed in the nodes of a cluster. It is implemented as an indexed Dataset of Rows."

- Resembles a Relational table
- Row class does not fix a schema at compile time, but at execution time
  - Uses `StructType`
  - Allows to infer schemas from the file (e.g., CSV or JSON)
- Can be **partitioned and distributed**
  - Implemented on top of Resilient Distributed Datasets (RDD)

https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/Dataset.html
https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/Row.html
https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/types/StructType.html

Type safety

DataFrame: No type safety

```
// Sample DataFrame
val df = Seq(
  ("Alice", 30),
  ("Bob", 25)
).toDF("name", "age")

// Compile-time: OK (even if we make a mistake)
df.select("naem").show()  // ← Typo: "naem" instead of "name"
```

Dataset: Type-safe (not supported in Python)

```
case class Person(name: String, age: Int)
val ds = df.as[Person]


// Compile-time: This fails if we make a mistake
ds.map(_.naem).show()  // ← Typo: "naem" instead of "name”
```

Dataframe vs Matrix/Array/Tensor

Dataframe	Matrix
Heterogeneously typed	Homogeneously typed
Both numeric and non-numeric types	Only numeric types
Explicit column names (also row in Pandas)	No names at all
Supports relational algebra	Does not support relational algebra

D. Petersohn, et al. *Towards Scalable Dataframe Systems. Proc. VLDB Endow. 13(11), 2020*

Dataframe vs Relation/Table

Pandas Dataframe	Spark Dataframe	Relation
Ordered	Unordered	
Named rows	Unnamed rows	
Lazily-induced schema		Rigid schema
Column-row symmetry	Columns and rows are different	
Supports linear algebra	Does not support linear algebra	

D. Petersohn, et al. *Towards Scalable Dataframe Systems. Proc. VLDB Endow. 13(11), 2020*

Dataframe implementations

Pandas	Spark
Eager evaluation of transformations	Lazy evaluation of transformations
Resides in memory	Requires a SparkSession
Not scalable (multithread operators exist, but manual split is required)	Transparently scalable in the Cloud
Transposable	Non-transposable (problems with too many rows)

D. Petersohn, et al. *Towards Scalable Dataframe Systems. Proc. VLDB Endow. 13(11), 2020*

Spark dataframe operations

a) Input/Output

b) Transformations

- Lower abstraction: O-O interface (similar to RDDs)
  - Functions over columns
- Higher abstraction: SQL

do not execute immediately, they build the computation plan – DAG (lazy evaluation)

c) Actions

d) Schema management

compute and return the final result (eager execution)

Input/Output

- Matrix
- Pandas dataframe
- CSV
- JSON
- RDBMS
- HDFS file formats:
  - ORC
  - Parquet
  - Delta

Transformations available

- `select` `df.select("name", "age")`
- `filter/where` `df.filter(df.gender == "F")`
- `sample` `df.sample(0.5)`
- `distinct/dropDuplicates` `df.select("gender").distinct()`
- `sort` `df.sort(col("age").desc)`
- `replace` `df.na.replace({"gender": {"F": "Female"}})`
- `groupBy+agg` `df.groupBy("gender").agg(avg("age").as("avg_age"))`
- `union/unionAll/unionByName` `df.union(anotherDF)`
- `substract` `df.subtract(another_df)`
- `join` `df1.join(df2, df1.id == df2.id, "inner")`
- …

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/dataframe.html

Functions over columns

- Normal (*lit*, *isNull*, …)
- Math (*sqrt*, *sin*, *ceil*, *log*, …)
- Daytime (*current_date*, *dayofweek*, …)
- Collection (*array_sort*, *forall*, *zip_with*, …)
- Aggregate (*avg*, *count*, *first*, *corr*, *max*, *min*, …)
- Sort (*asc*, *desc*, *asc_nulls_first*, …)
- String (*length*, *lower*, *trim*, …)

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html

Actions available

- `count`
- `first`
- `collect` ☠
- `take/head/tail`
- `show`
- `write`
- `toPandas` ☠
- …

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/dataframe.html

Schema operations

- `summary/describe`
- `printSchema`
- `columns`

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/dataframe.html

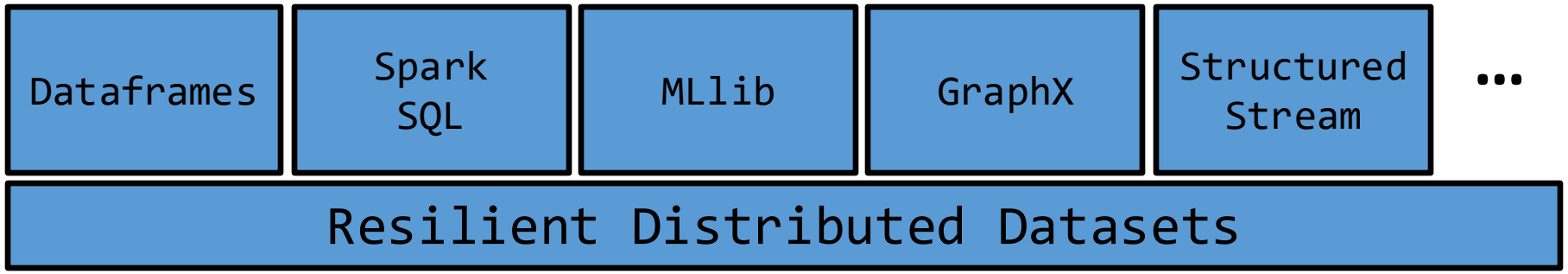
Optimizations

- Lazy evaluation
  - `cache/persist` – *`cache`* stores data in memory by default, *`persist`* offers flexibility to store data in memory or disk or combination of both
    - `unpersist` – free up memory/disk resources
  - `checkpoint` - used to truncate lineage for fault tolerance by saving RDDs or DataFrames to a reliable storage location (e.g., HDFS). This prevents excessive lineage growth during iterative operations and thus loss in case of failures
- Parallelism
  - `repartition` - increases or decreases partitions (shuffling involved)
  - `coalesce` - reduces partitions without shuffling (efficient for downsizing)

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/dataframe.html

Abstractions

Spark Abstractions



Spark SQL

- Besides the O-O interface of dataframes, there is a declarative one
- It can be used independently of the kind of source
  - Not only Relational tables
- It is translated into functional programming by an optimizer
  - Based on
    a) Rules
       - Predicate push down
       - Column pruning
    b) Cost model
  - Extensible

https://spark.apache.org/sql

Spark SQL interface

- There is a catalog with all tables available

  `SparkSession.catalog`

- Dataframes are registered as views in the catalog

  `DataFrame.createOrReplaceTempView(<tablename>)`

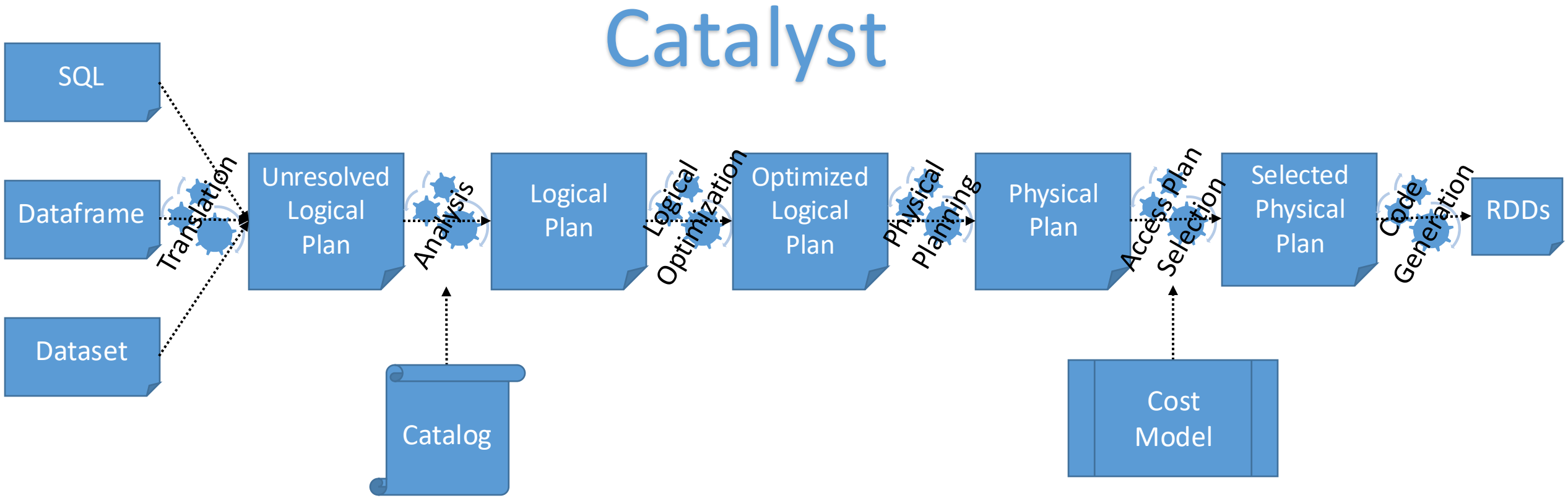
- Queries:

  `SparkSession.sql(<query>)`

  - Input is simply a string
  - Output is a dataframe

https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/catalog.html

Shared Optimization and Execution



Closing

Summary

- Overcoming MapReduce limitations
- Dataframes
  - Comparison
    - Differences with Relations
    - Differences with Matrixes
    - Differences in Pandas and Spark
  - Operations
    - Transformations
    - Actions
  - Optimizations
    - Lazy evaluation
    - Parallelism
- Abstraction

References

- H. Karau et al. *Learning Spark*. O'Really, 2015
- D. Petersohn, W. W. Ma, D. Jung Lin Lee, S. Macke, D. Xin, X. Mo, J. Gonzalez, J. M. Hellerstein, A. D. Joseph, A. G. Parameswaran. *Towards Scalable Dataframe Systems. Proc. VLDB Endow. 13(11), 2020*
- A. Hogan. *Procesado de Datos Masivos* (Universidad de Chile). http://aidanhogan.com/teaching/cc5212-1-2020