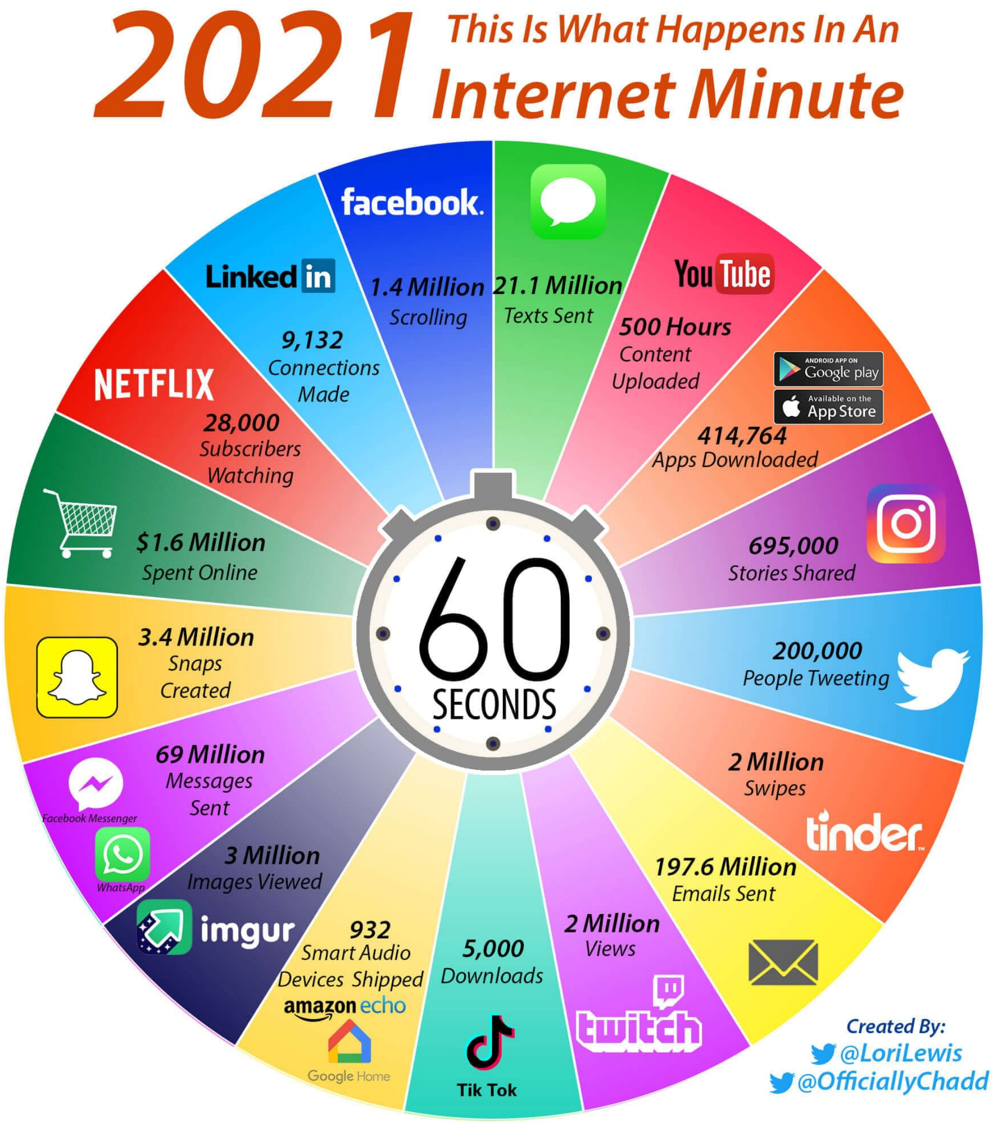
Distributed platforms for Big Data

Prof. Carlo Ferrari

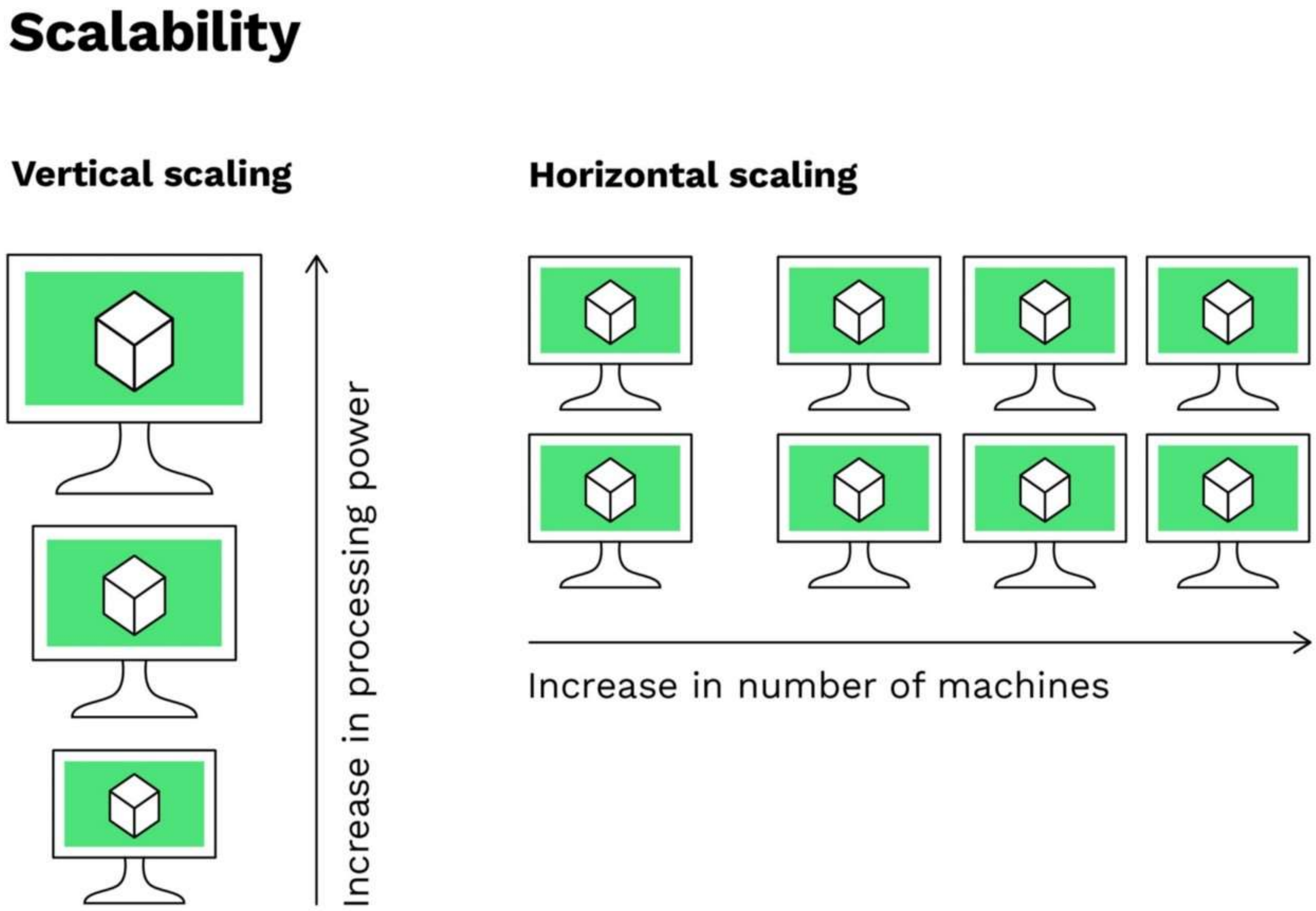
Michele Stecca, Ph.D.

a.y. 2023-2024

Data, data, data



Vertical vs. Horizontal scalability

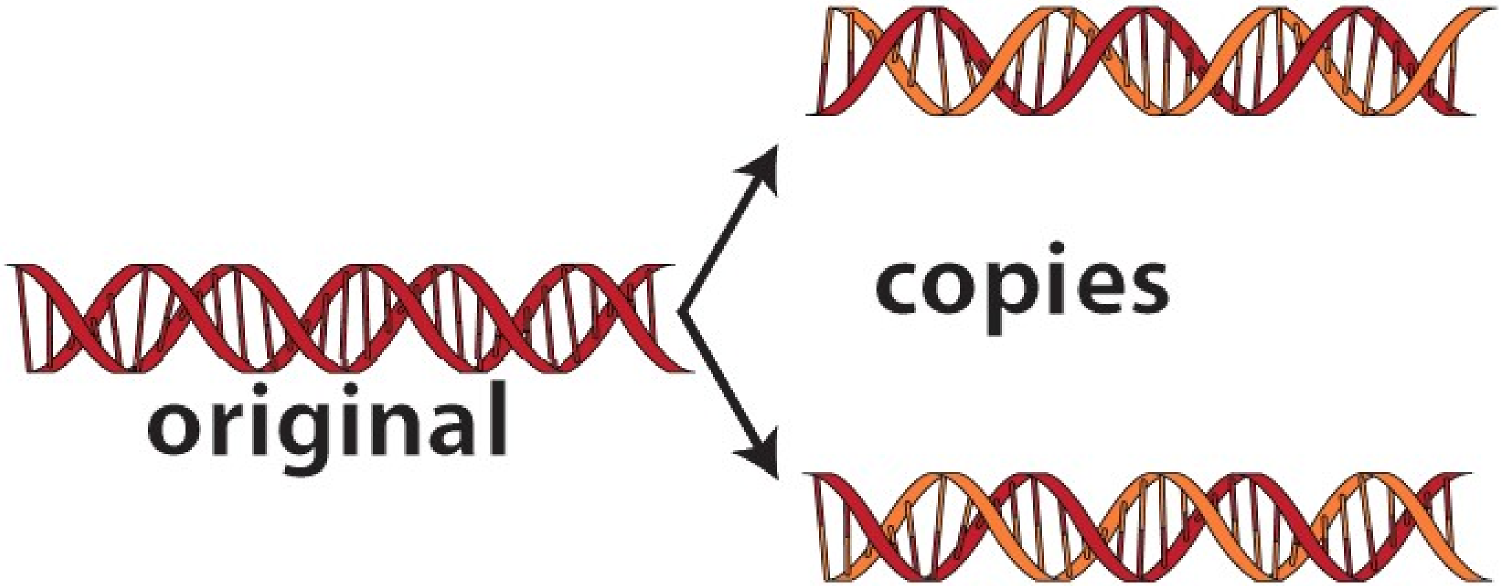


Design principles

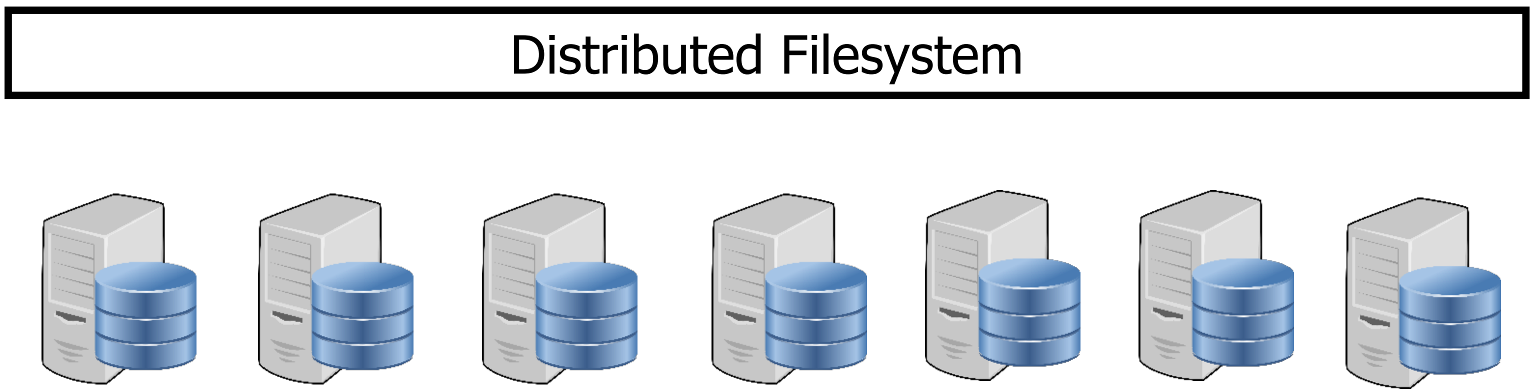
(Data) Partitioning

Design principles

(Data) Replication

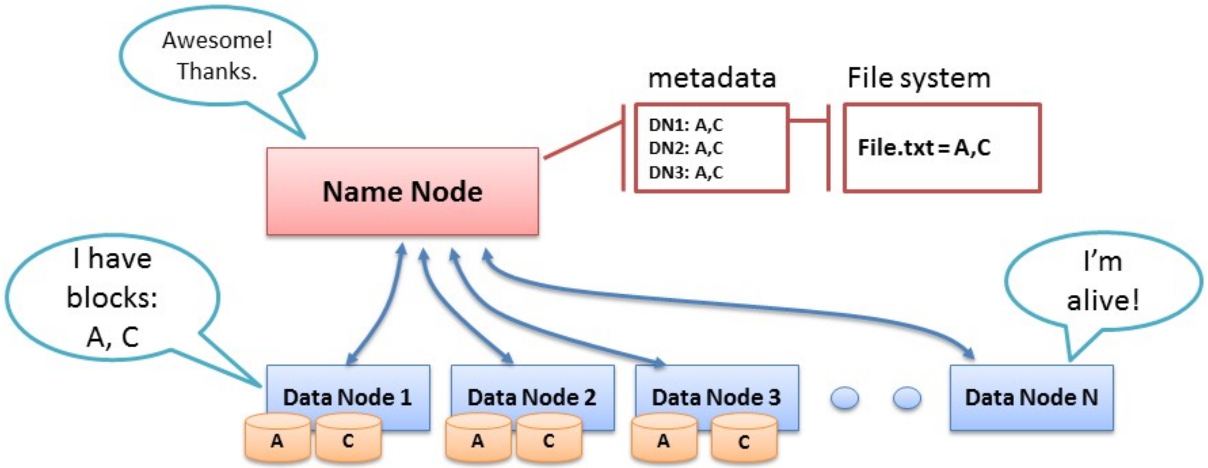


An architecture for Big Data



HDFS

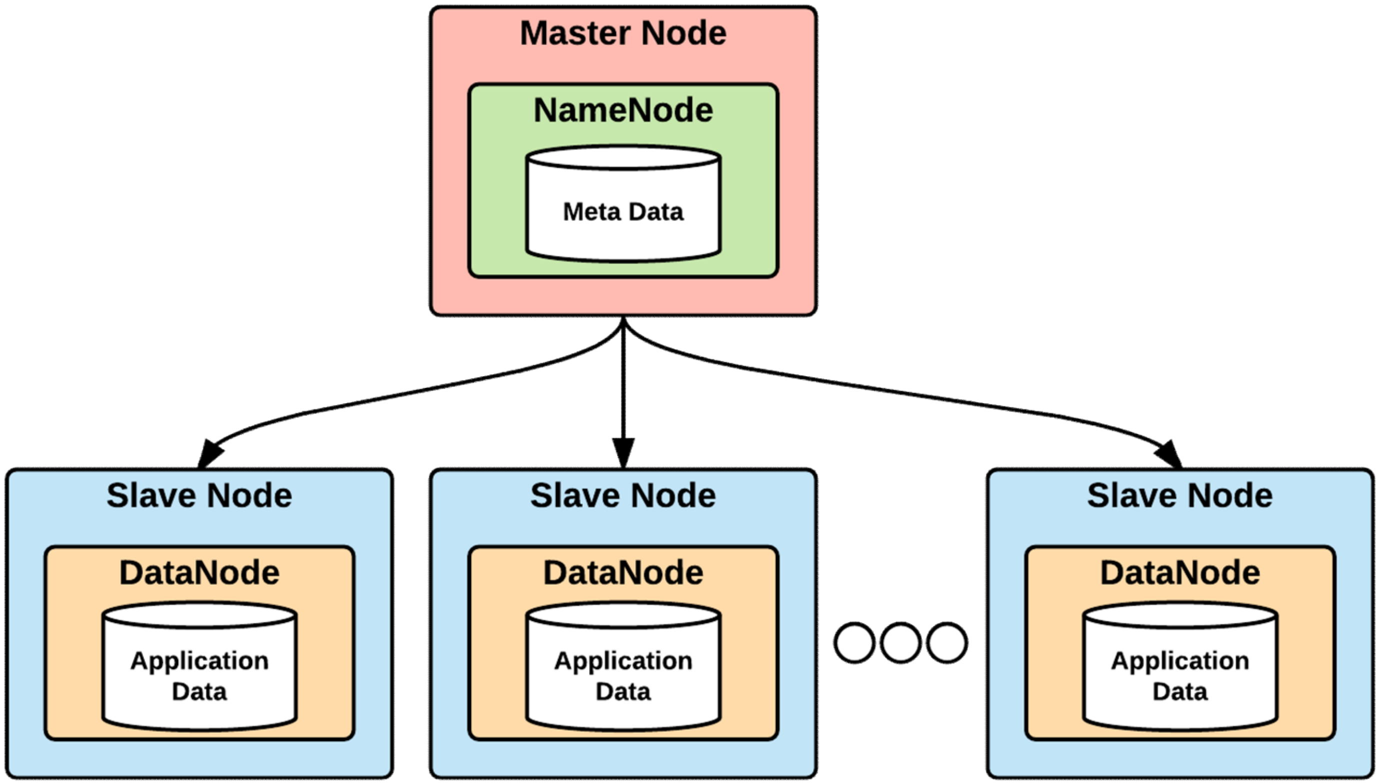
HaDoop FileSystem Architecture



- Files are splitted into >64MB blocks
- Throughput optimized
- Write once – read many
- Failure resistant: manages failures using block replication

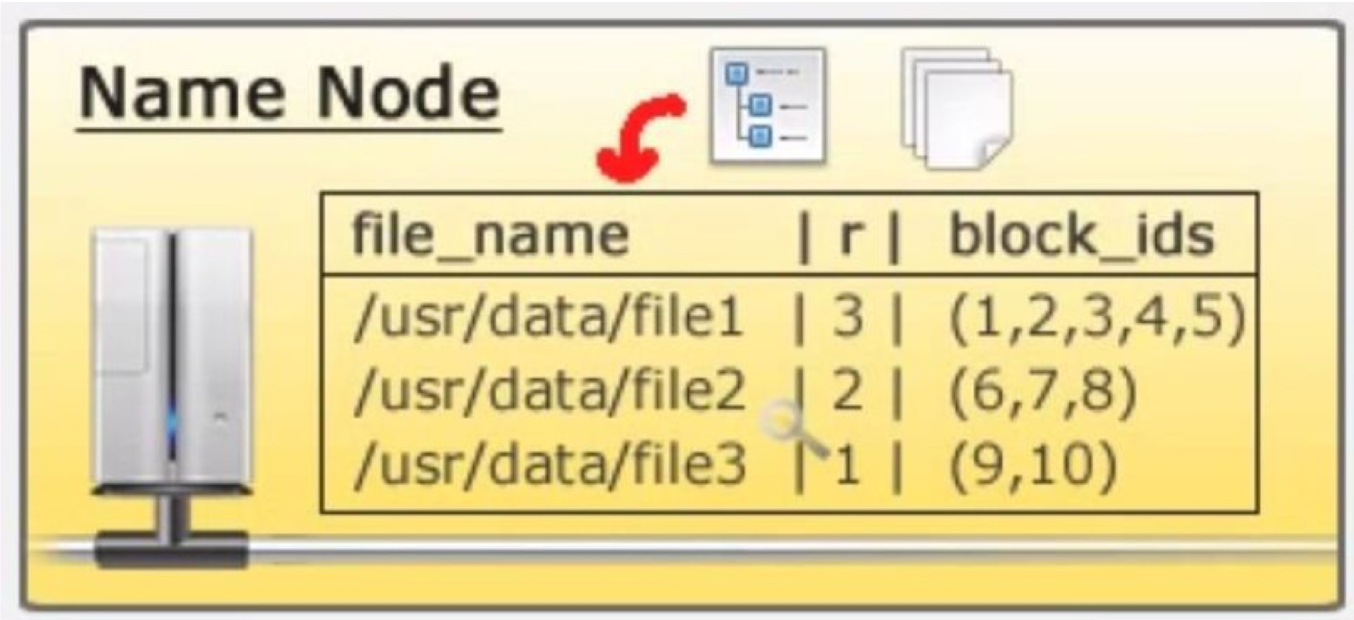
HDFS

Master & Slave
architecture



HDFS

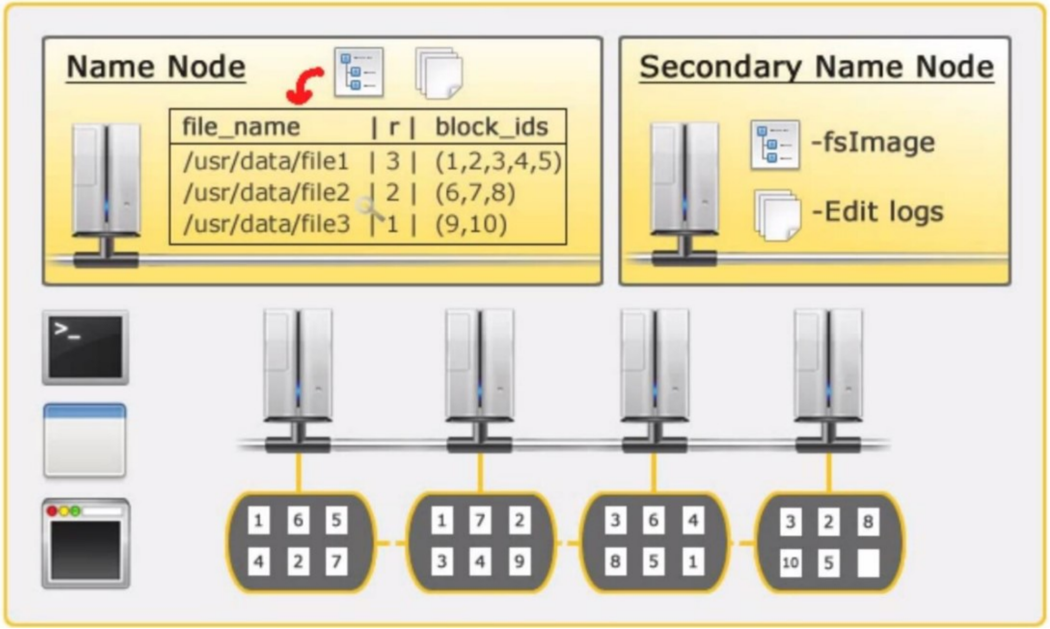
HaDoop FileSystem Architecture: Name Node



- Cluster's SPOF!
- Manages the file system
- Lists the files and blocks in which they are divided
- Manages strategies replication and block allocation
- Checks nodes's reliability

HDFS

HaDoop FileSystem Architecture: DataNode



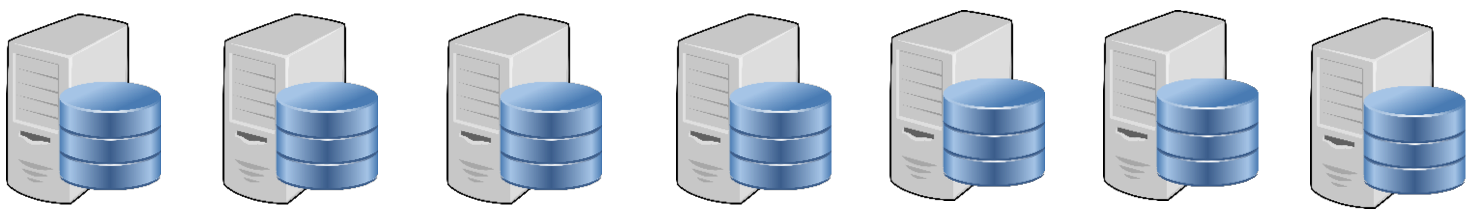
- *Manages the storage and the client's requests*
- *Sends Heartbeat to NameNod*

An architecture for Big Data

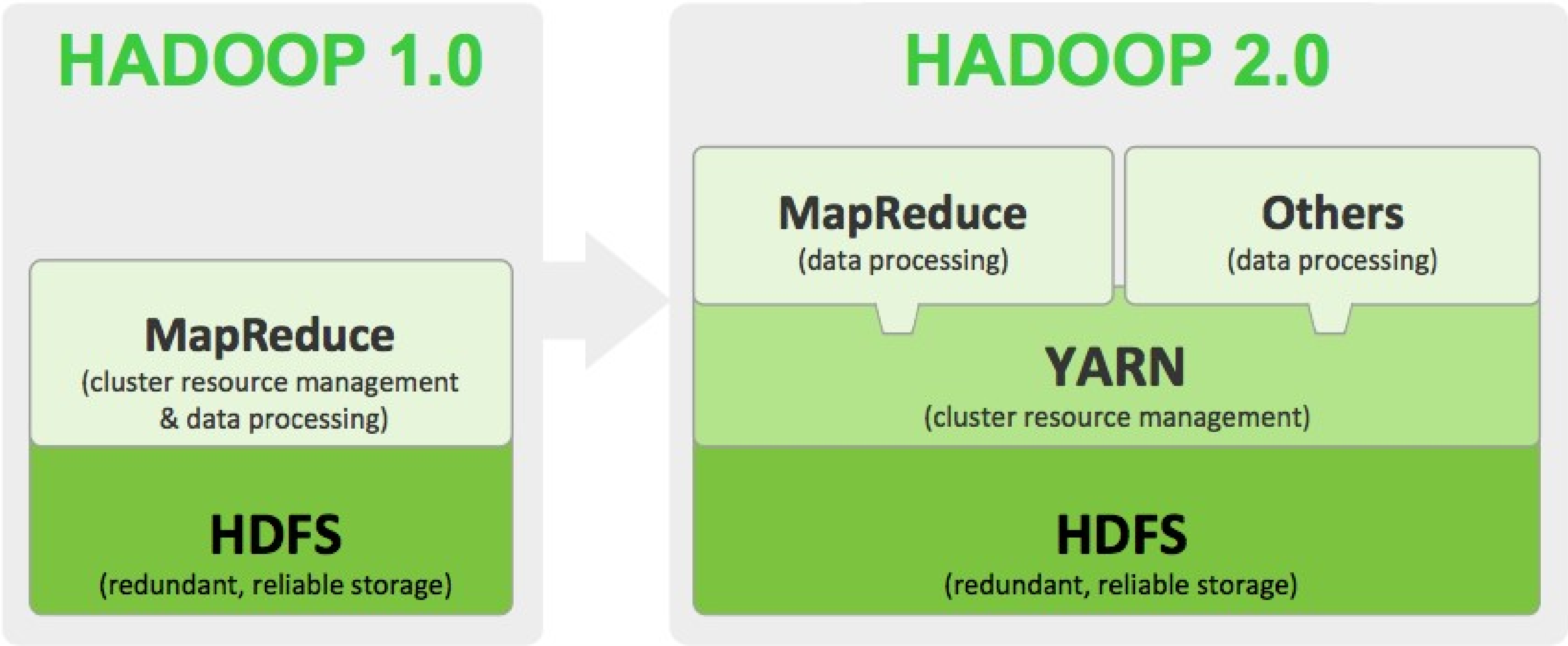
Move computation close to data!!!

Distributed Computational Model
+ Execution Engine

Distributed Filesystem (HDFS)

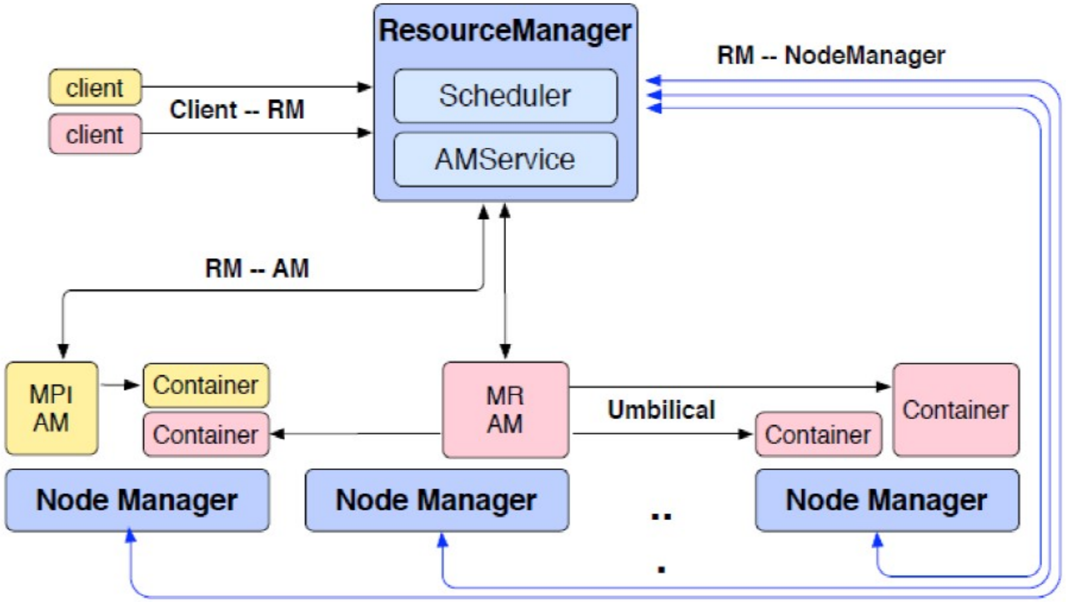


Hadoop versions



Cluster management: Yarn

Yet Another Resource Negotiator architecture:



Resource Manager:

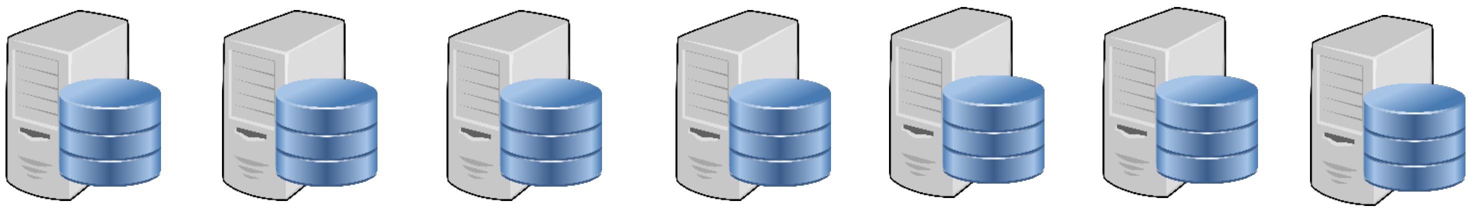
- One per cluster – global view
- No static resource partitioning
- Handle Job request
- Find a container to Application Manager

An architecture for Big Data

Distributed Computational Model
+ Execution Engine: Map Reduce

Resource Manager (YARN)

Distributed Filesystem (HDFS)



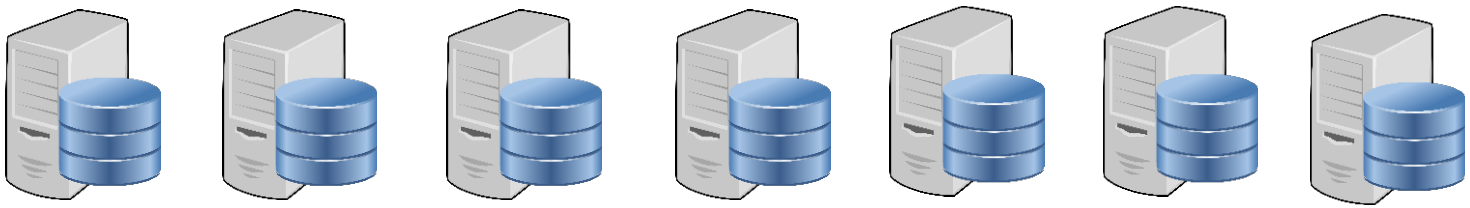
An architecture for Big Data

Applications («Pure» MR Apps, SQL, Machine Learning, Graphs, Streaming, etc.)

Distributed Computational Model
+ Execution Engine (Map Reduce)

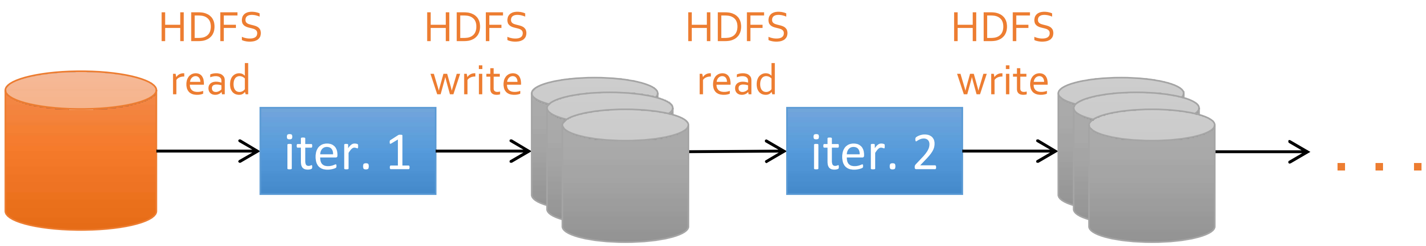
Resource Manager (YARN)

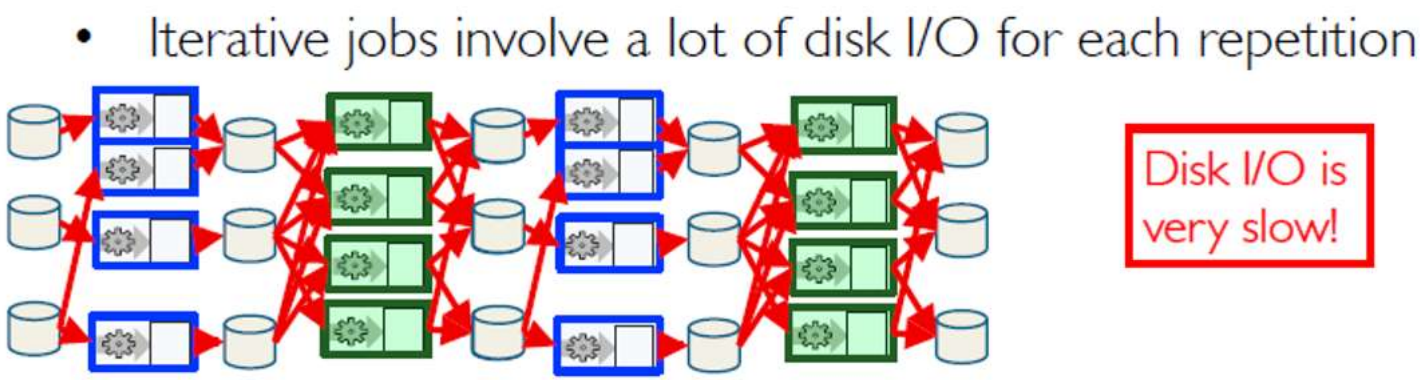
Distributed Filesystem (HDFS)



Map Reduce limitations

- Not so flexible from a programmer point of view
- Not so efficient
- …





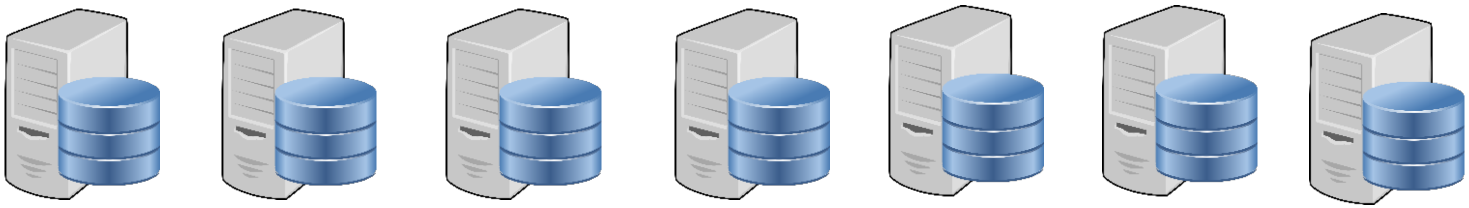
From Hadoop/MapReduce to Spark

Applications («Pure» MR Apps, SQL, Machine Learning, Graphs, Streaming, etc.)

Distributed Computational Model
+ Execution Engine: Map Reduce

Resource Manager (YARN)

Distributed Filesystem (HDFS)



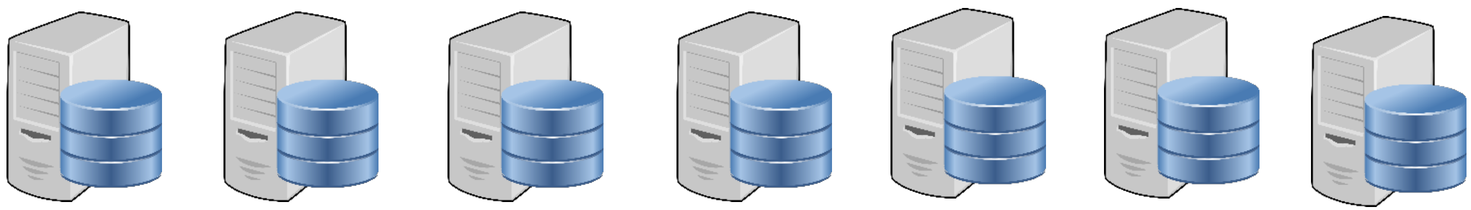
From Hadoop/MapReduce to Spark

Applications («Pure» Spark Apps, SQL, Machine Learning, Graphs, Streaming, etc.)

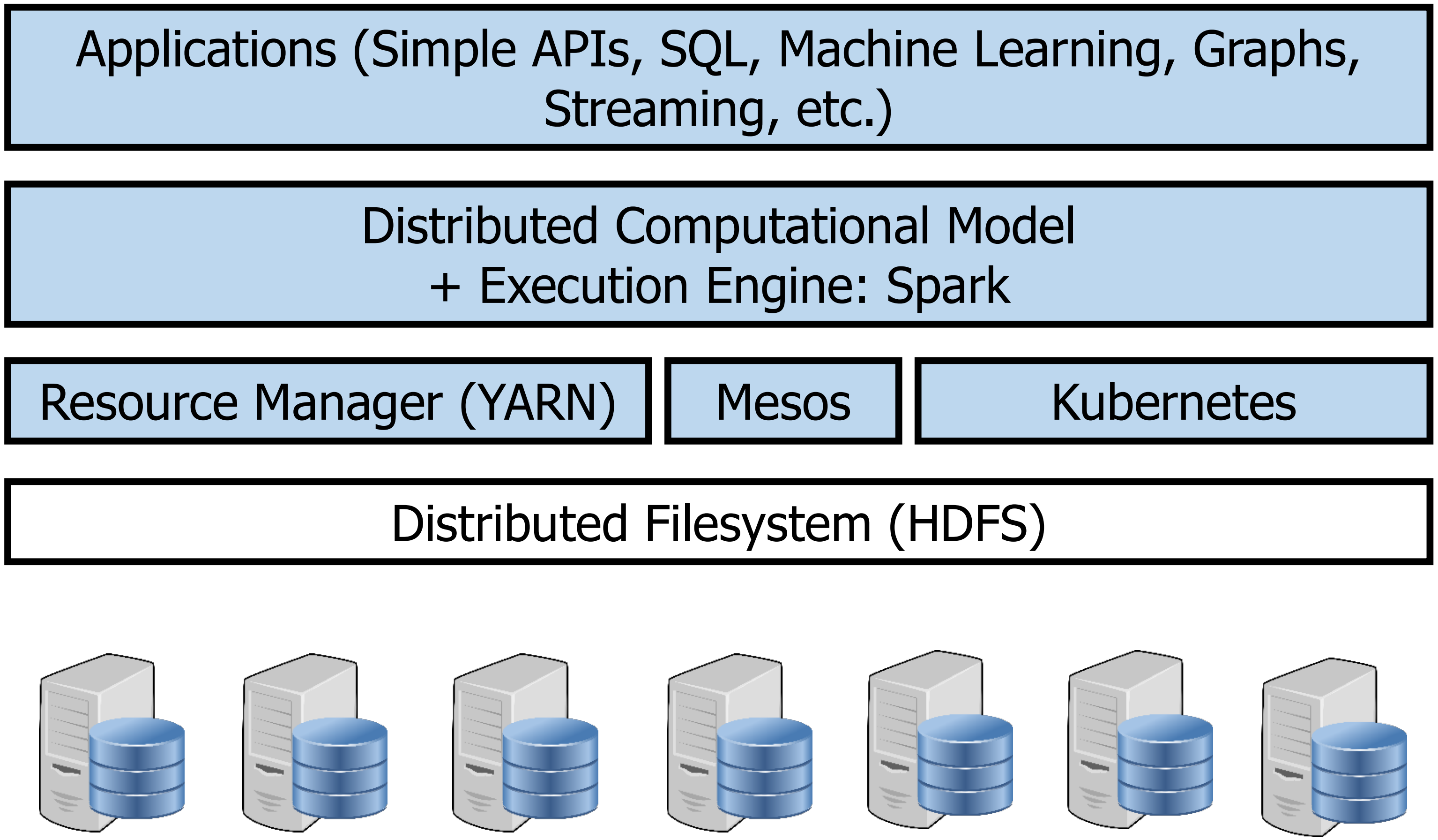
Distributed Computational Model
+ Execution Engine: Spark

Resource Manager (YARN)

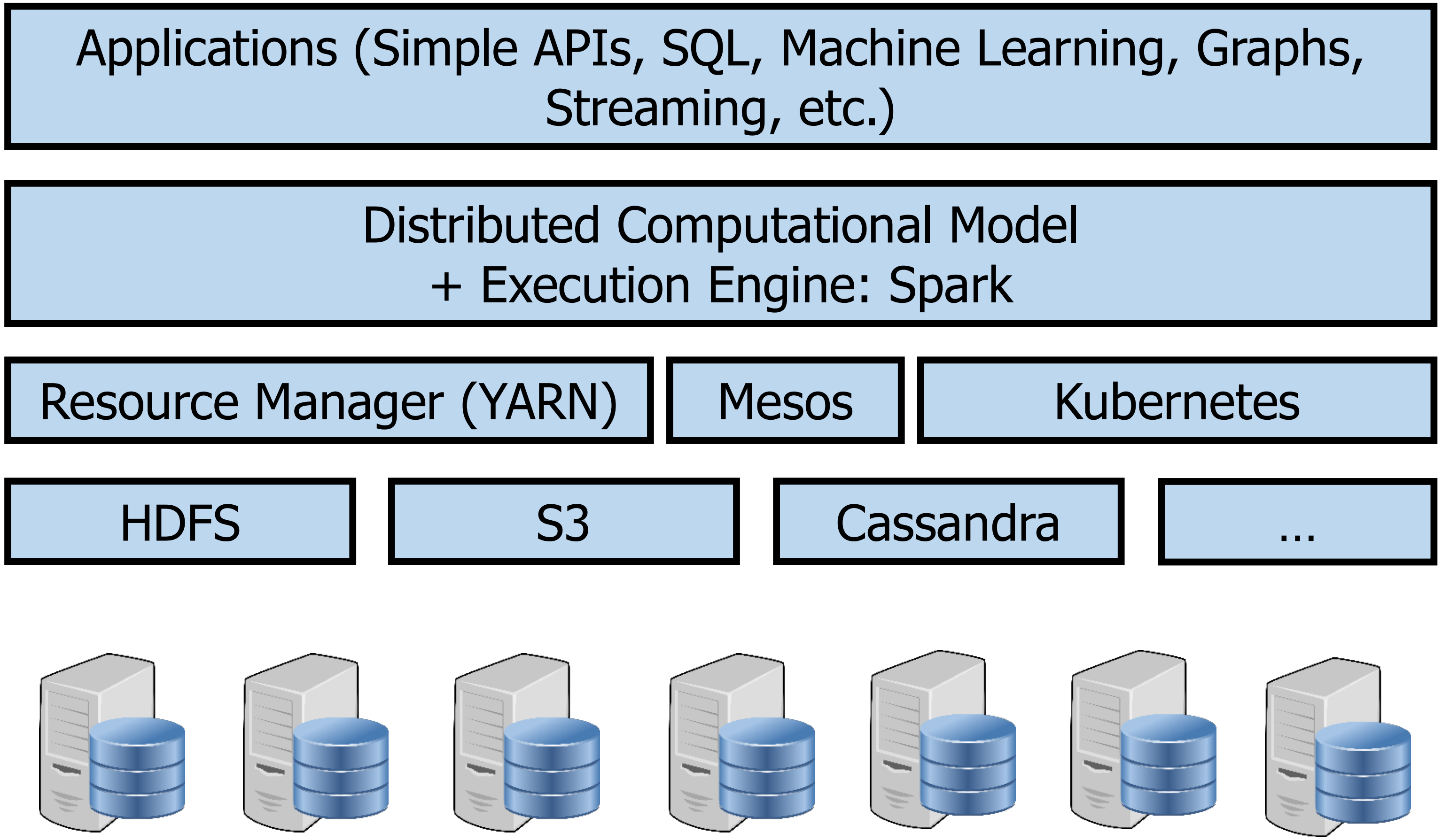
Distributed Filesystem (HDFS)



From Hadoop/MapReduce to Spark



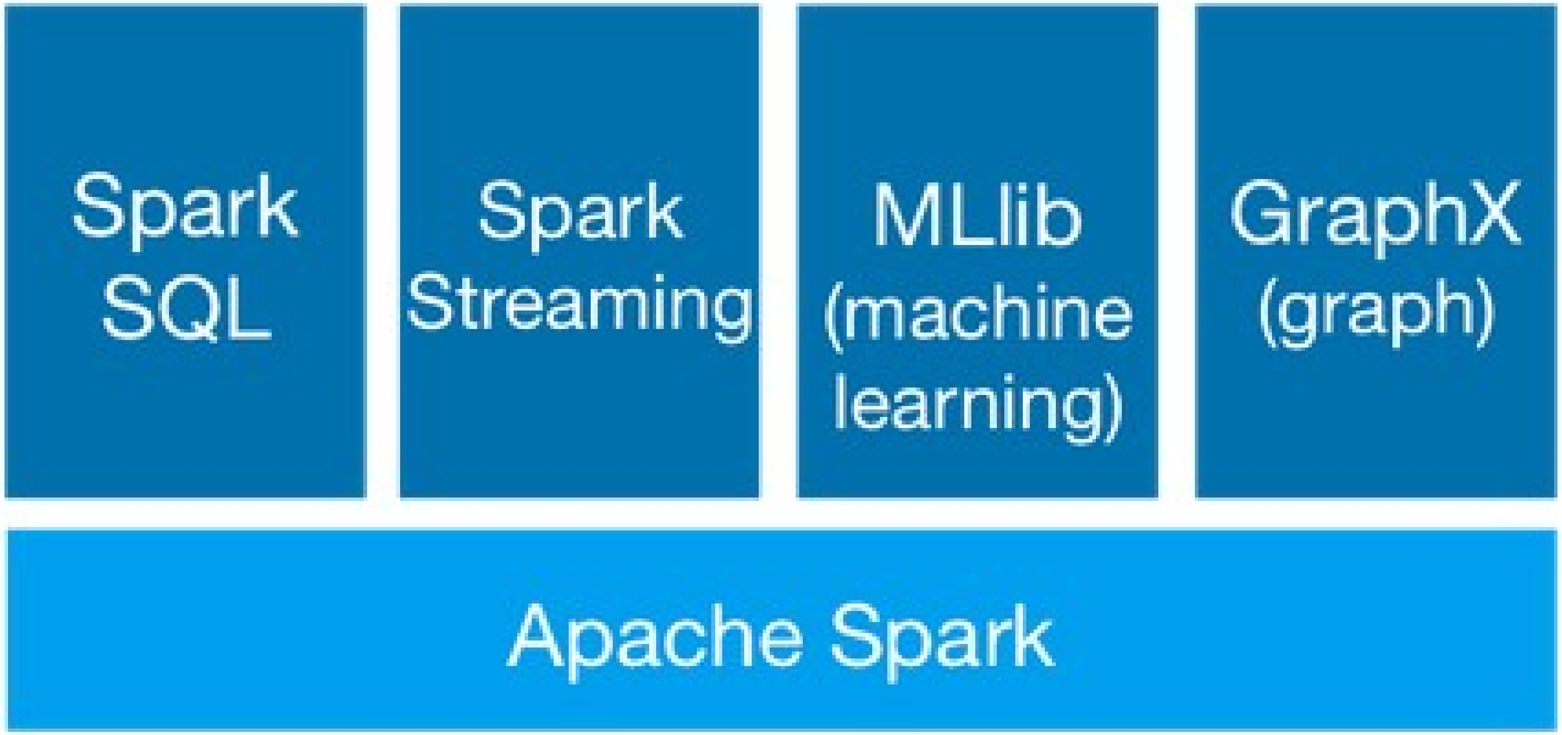
From Hadoop/MapReduce to Spark



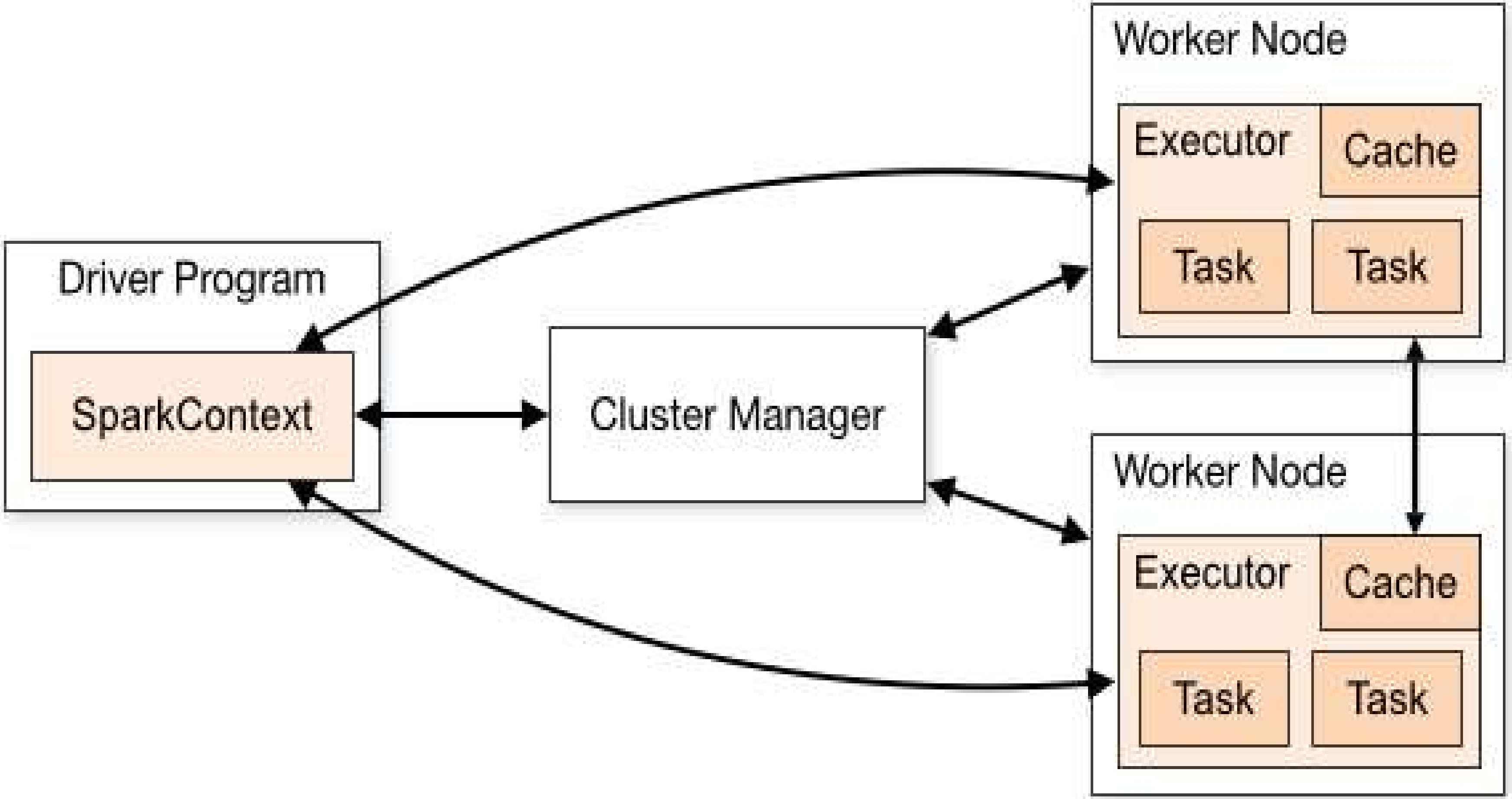
MapReduce vs. Spark

- Spark is in-memory
- Less expensive shuffles
- There are many more primitives
- It supports Java, Scala, Python & R
- Interactive shells are available
- Generalized patterns
  - unified engine for many use cases
- Lazy evaluation of the lineage graph
  - reduces wait states, better pipelining
- Lower overhead for starting jobs

Spark ecosystem



Spark (Deployment)



RDDs

- **R**esilient **D**istributed **D**atasets (RDDs) are the primary abstraction in Spark – a **fault-tolerant** collection of elements that can be operated on in **parallel**
- 2 types of operations on RDDs:
  - *transformations* and *actions*
    - transformations are lazy (not computed immediately)
- however, an RDD can be *persisted* into storage in memory or disk

Some Primitives

transformation	*description*
`map(func)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(func)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`flatMap(func)`	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)
`sample(withReplacement, fraction, seed)`	sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*
`union(otherDataset)`	return a new dataset that contains the union of the elements in the source dataset and the argument
`distinct([numTasks]))`	return a new dataset that contains the distinct elements of the source dataset

Some Primitives

transformation	*description*
groupByKey([*numTasks*])	when called on a dataset of (K, V) pairs, returns a dataset of (K, Seq[V]) pairs
reduceByKey(*func*, [*numTasks*])	when called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function
sortByKey([*ascending*], [*numTasks*])	when called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument
join(*otherDataset*, [*numTasks*])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
cogroup(*otherDataset*, [*numTasks*])	when called on datasets of type (K, V) and (K, W), returns a dataset of (K, Seq[V], Seq[W]) tuples – also called groupWith
cartesian(*otherDataset*)	when called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements)

Some Primitives

action	*description*
`reduce(`*`func`*`)`	aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one), and should also be commutative and associative so that it can be computed correctly in parallel
`collect()`	return all the elements of the dataset as an array at the driver program – usually useful after a filter or other operation that returns a sufficiently small subset of the data
`count()`	return the number of elements in the dataset
`first()`	return the first element of the dataset – similar to *take(1)*
`take(`*`n`*`)`	return an array with the first *n* elements of the dataset – currently not executed in parallel, instead the driver program computes all the elements
`takeSample(`*`withReplacement, fraction, seed`*`)`	return an array with a random sample of *num* elements of the dataset, with or without replacement, using the given random number generator seed

Some Primitives

action	*description*
`saveAsTextFile(`*`path`*`)`	write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call `toString` on each element to convert it to a line of text in the file
`saveAsSequenceFile(`*`path`*`)`	write the elements of the dataset as a Hadoop `SequenceFile` in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. Only available on RDDs of key-value pairs that either implement Hadoop's `Writable` interface or are implicitly convertible to `Writable` (Spark includes conversions for basic types like `Int`, `Double`, `String`, etc).
`countByKey()`	only available on RDDs of type `(K, V)`. Returns a `Map` of `(K, Int)` pairs with the count of each key
`foreach(`*`func`*`)`	run a function *func* on each element of the dataset – usually done for side effects such as updating an accumulator variable or interacting with external storage systems

Some Primitives

- Spark can *persist* (or cache) a dataset in memory across operations
- Each node stores in memory any slices of it that it computes and reuses them in other actions on that dataset – often making future actions more than 10x faster
- The cache is *fault-tolerant*: if any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it

Spark SQL

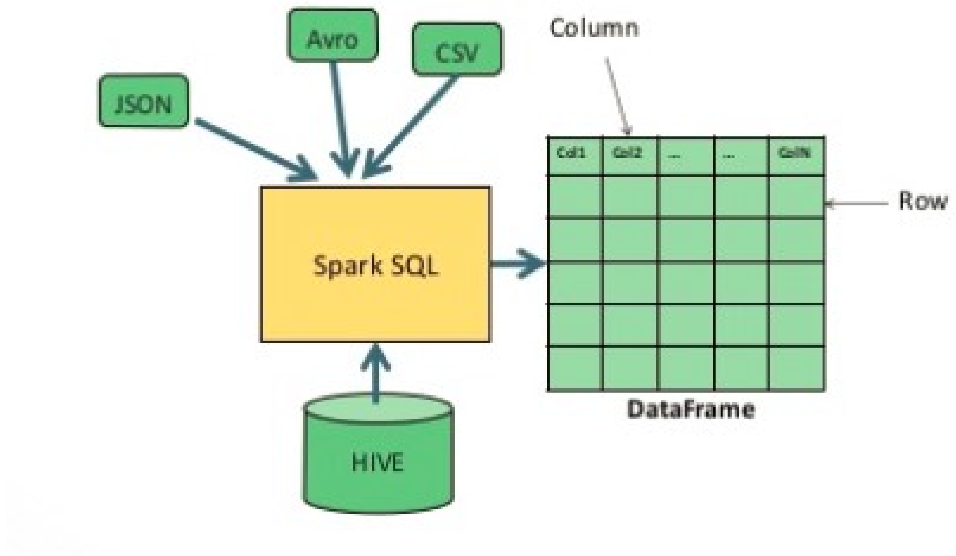
- Spark SQL is a Spark module for structured data processing
- Uses more information about the structure of both the data and the computation being performed
- Spark SQL uses this extra information to perform extra optimizations
- Integrated with Hive metastore

Spark SQL: the Dataframe abstraction

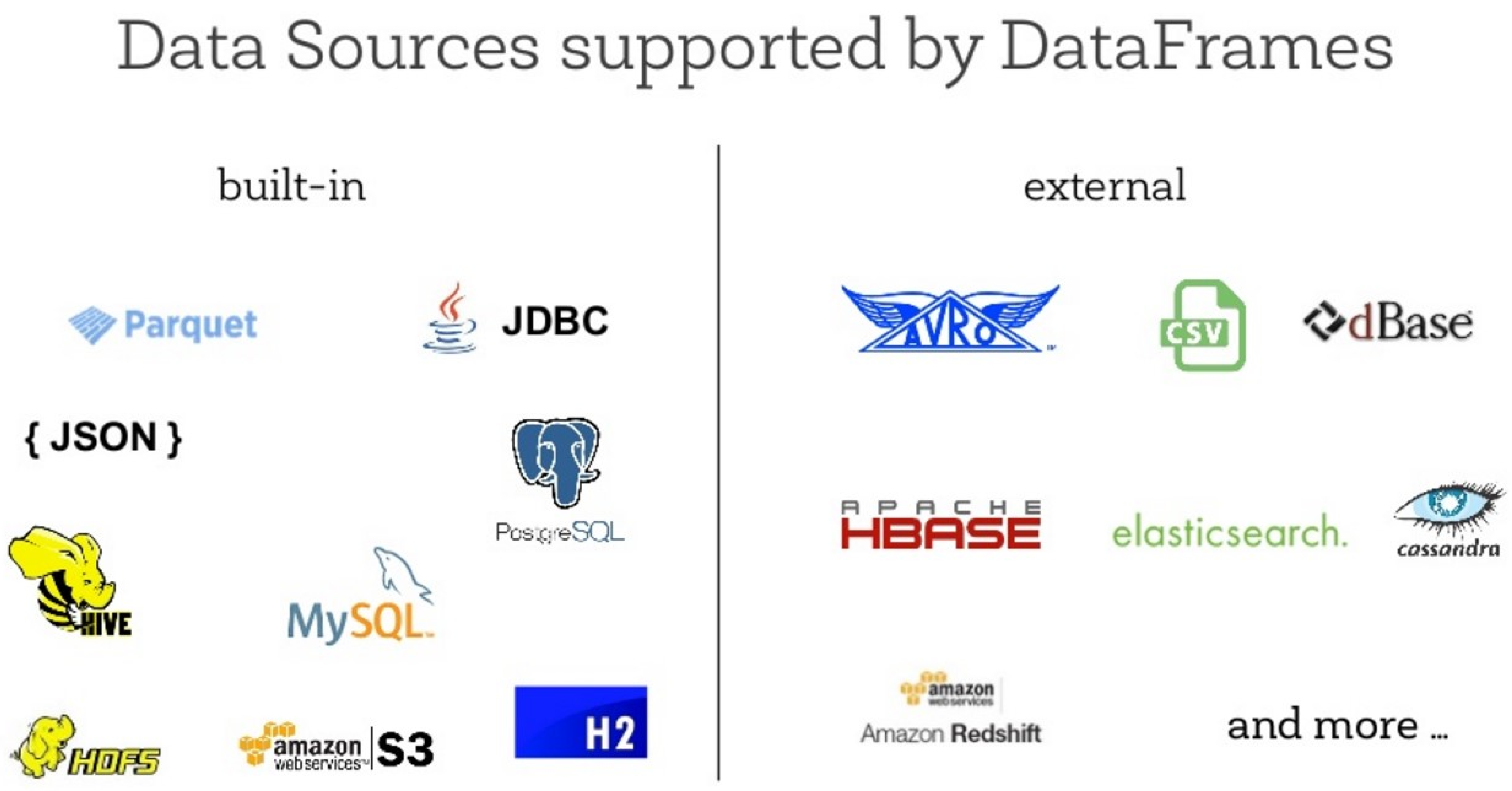
- The **DataFrame API** provides a **higher-level abstraction**, allowing you to use a query language to manipulate data. In fact, you can use **SQL**, as well.
- This code does essentially the same thing the previous RDD code does. Look how much easier it is to read.
- You have probably met DataFrames already in Python or R

Spark SQL: the Dataframe abstraction

- It provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine
- **It is conceptually equivalent to a table in a relational database or a data frame** in R/Python, but with richer optimizations under the hood
- DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs

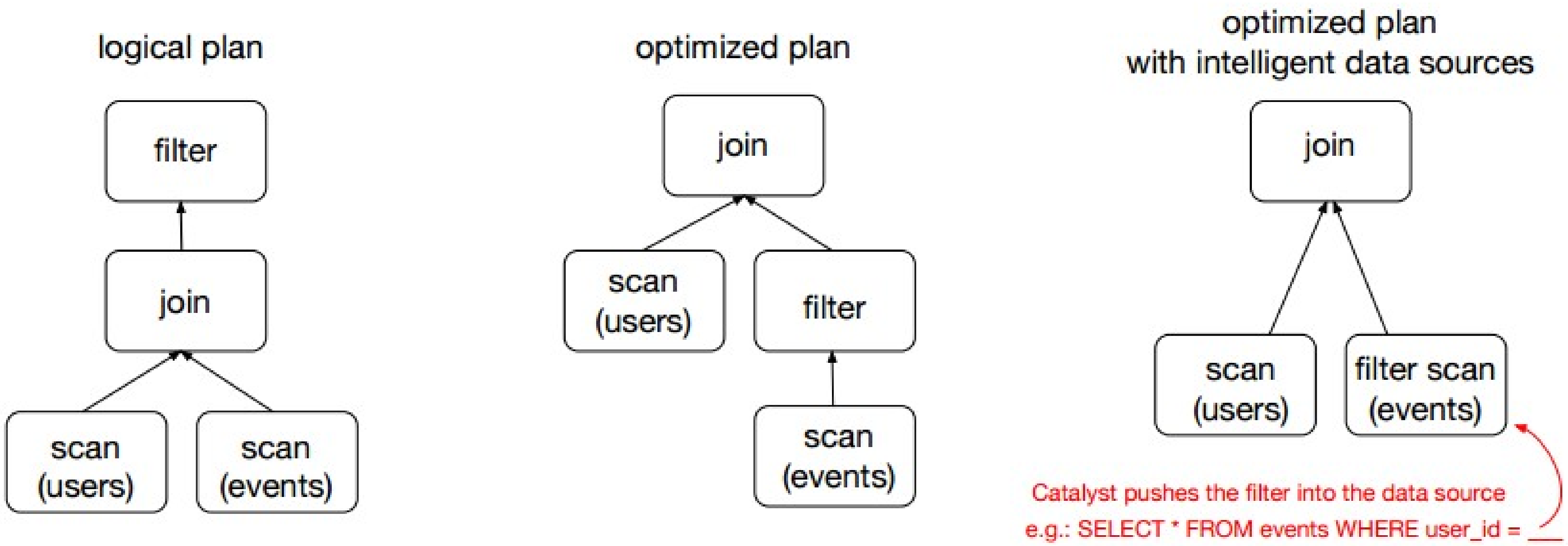


Spark SQL: the Dataframe abstraction



Example Optimization

```
users.join(events, users("id") === events("uid"))
     .filter(events("date") > "2015-01-01")
```



Spark SQL: JOIN Operations

(Distributed) JOIN Types

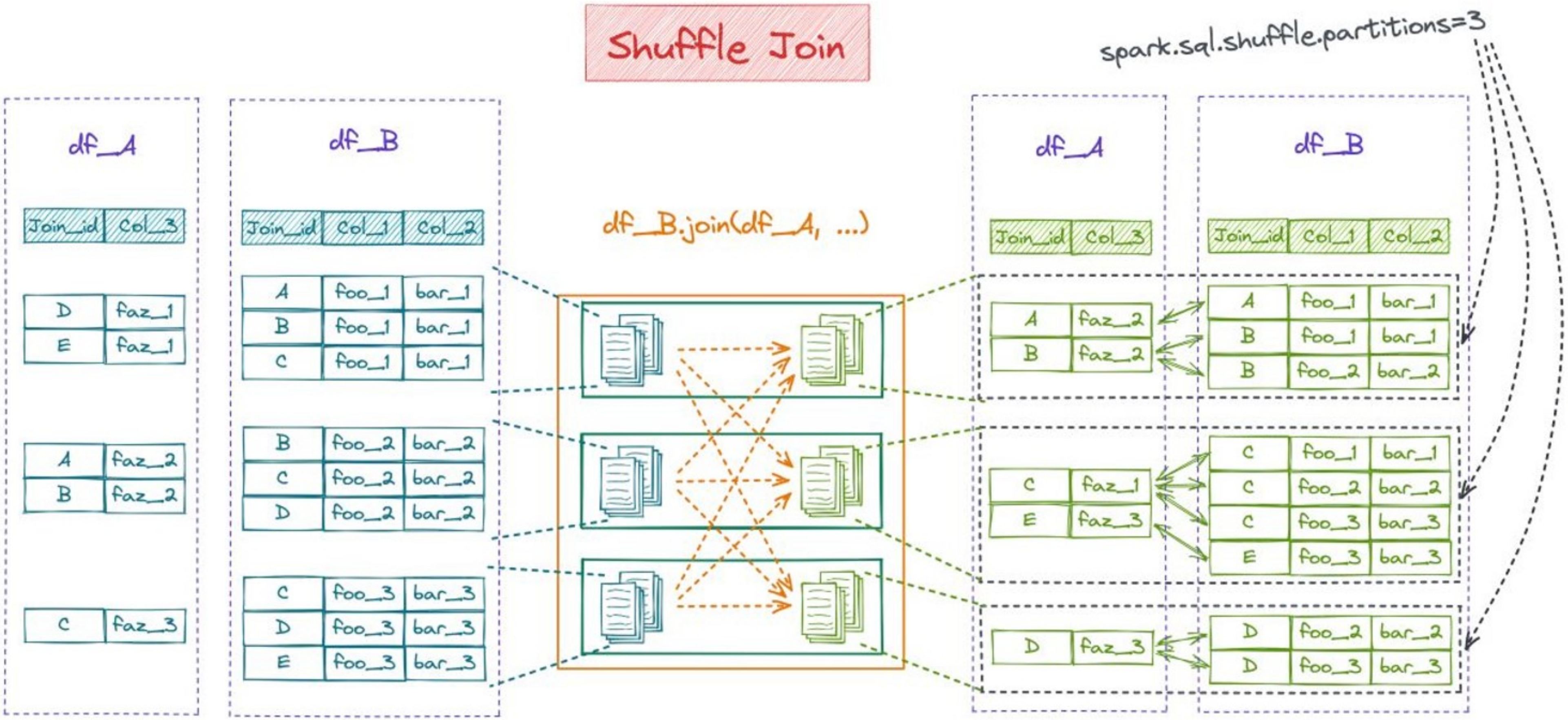
- Shuffle JOIN
- Broadcast JOIN
- Merge Sort JOIN
- Skew JOIN

JOIN Optimization parameters

spark.sql.adaptive.enabled - if this option is set to True Spark will make use of the runtime statistics to choose the most efficient query execution plan, one of the optimizations is automated conversion of shuffle join to a broadcast join.

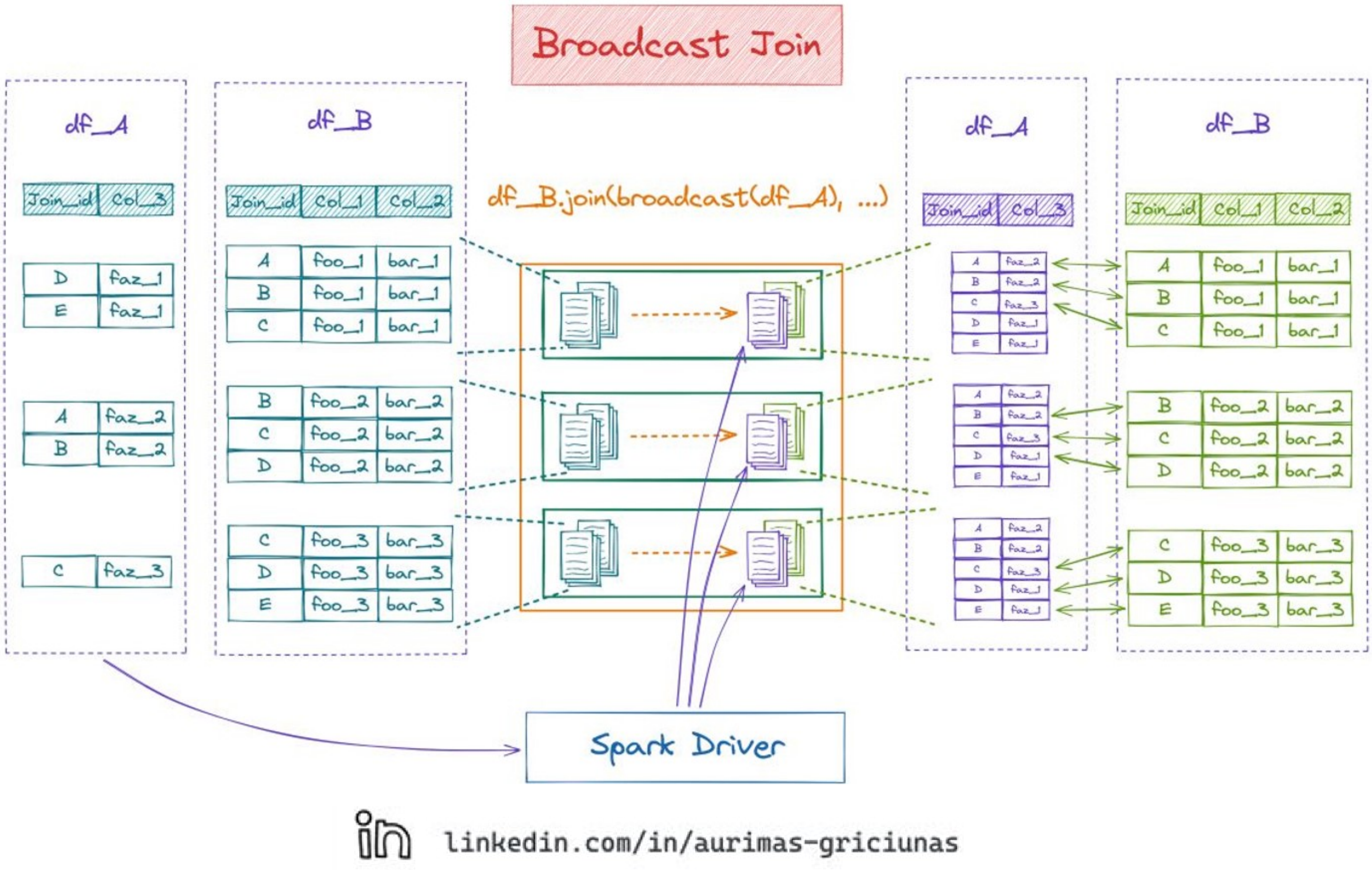
spark.sql.autoBroadcastJoinThreshold - denotes the maximum size of a dataset that would be automatically broadcasted.

Spark SQL: JOIN Operations



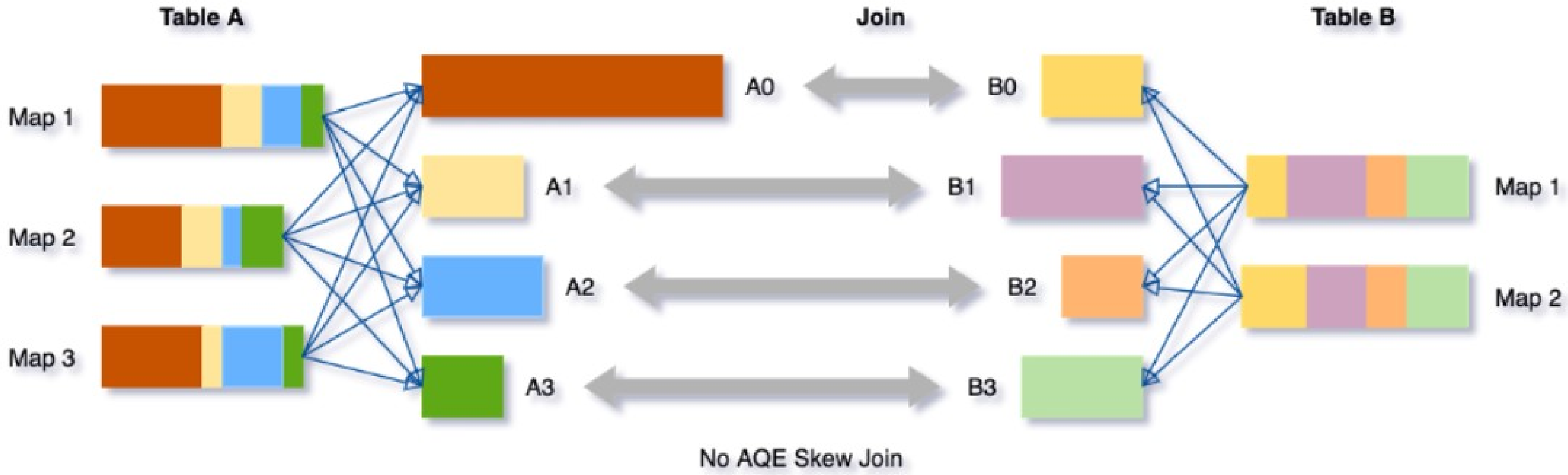
linkedin.com/in/aurimas-griciunas

Spark SQL: JOIN Operations



Spark SQL: JOIN Operations (Data Skew case)

- Data skew is a condition in which a table's data is unevenly distributed among partitions in the cluster.
- Data skew can severely downgrade performance of queries, especially those with joins.
- Joins between big tables require shuffling data and the skew can lead to an extreme imbalance of work in the cluster.

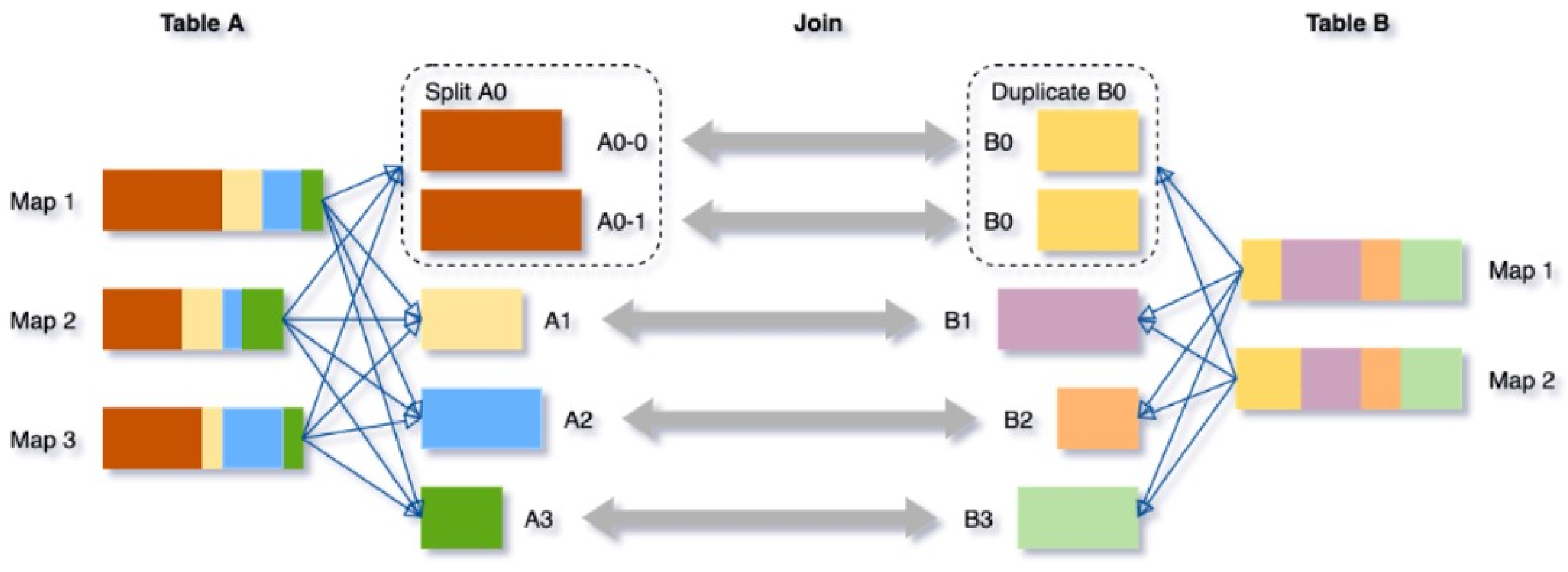


Spark SQL: JOIN Operations (Data Skew case)

JOIN Optimization parameters

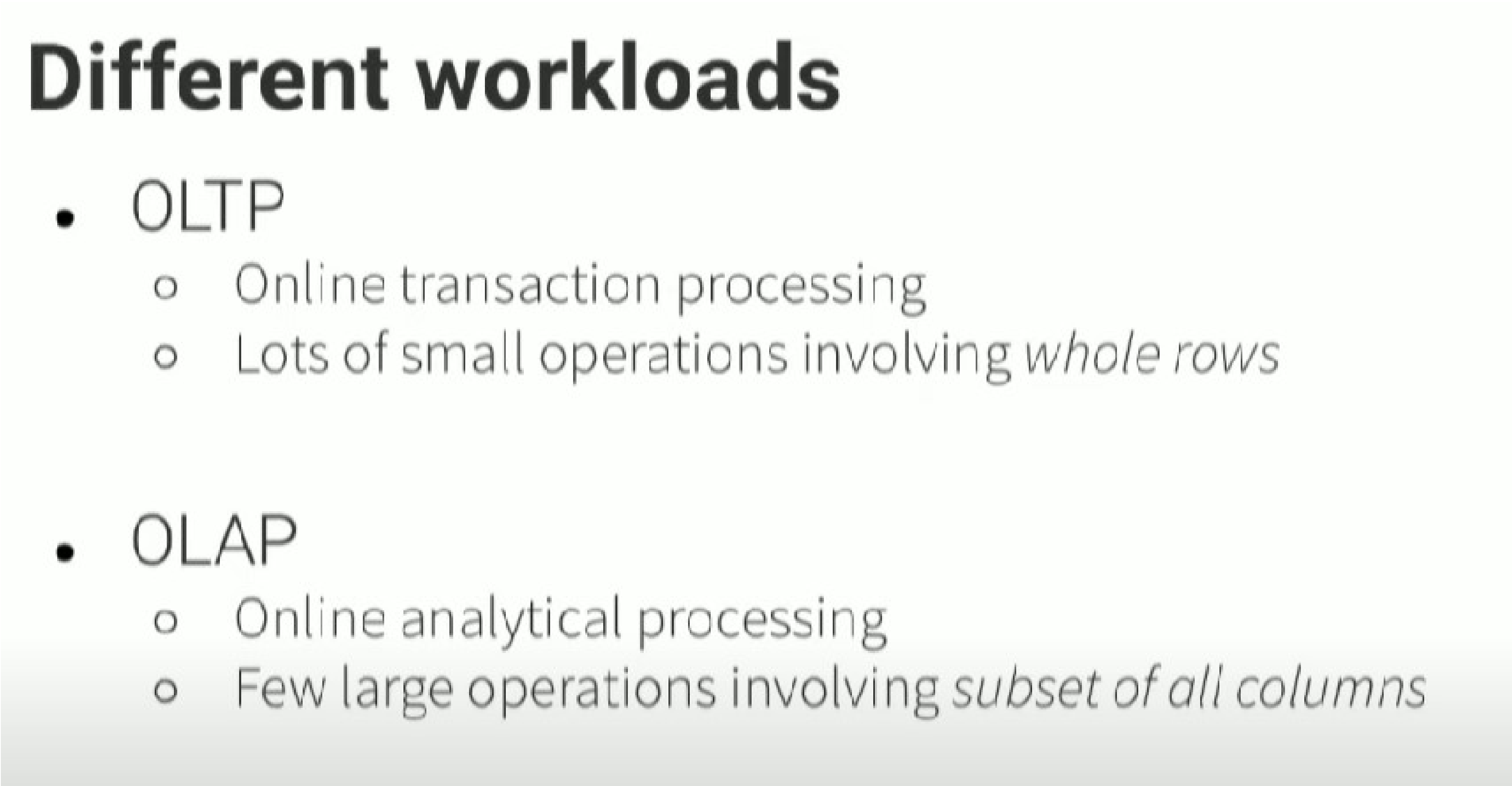
spark.sql.adaptive.optimizeSkewsInRebalancePartitions.enabled
- When true and spark.sql.adaptive.enabled is true, Spark will optimize the skewed shuffle partitions in RebalancePartitions and split them to smaller ones

AQE mechanisms transparently discover and optimize implementation.

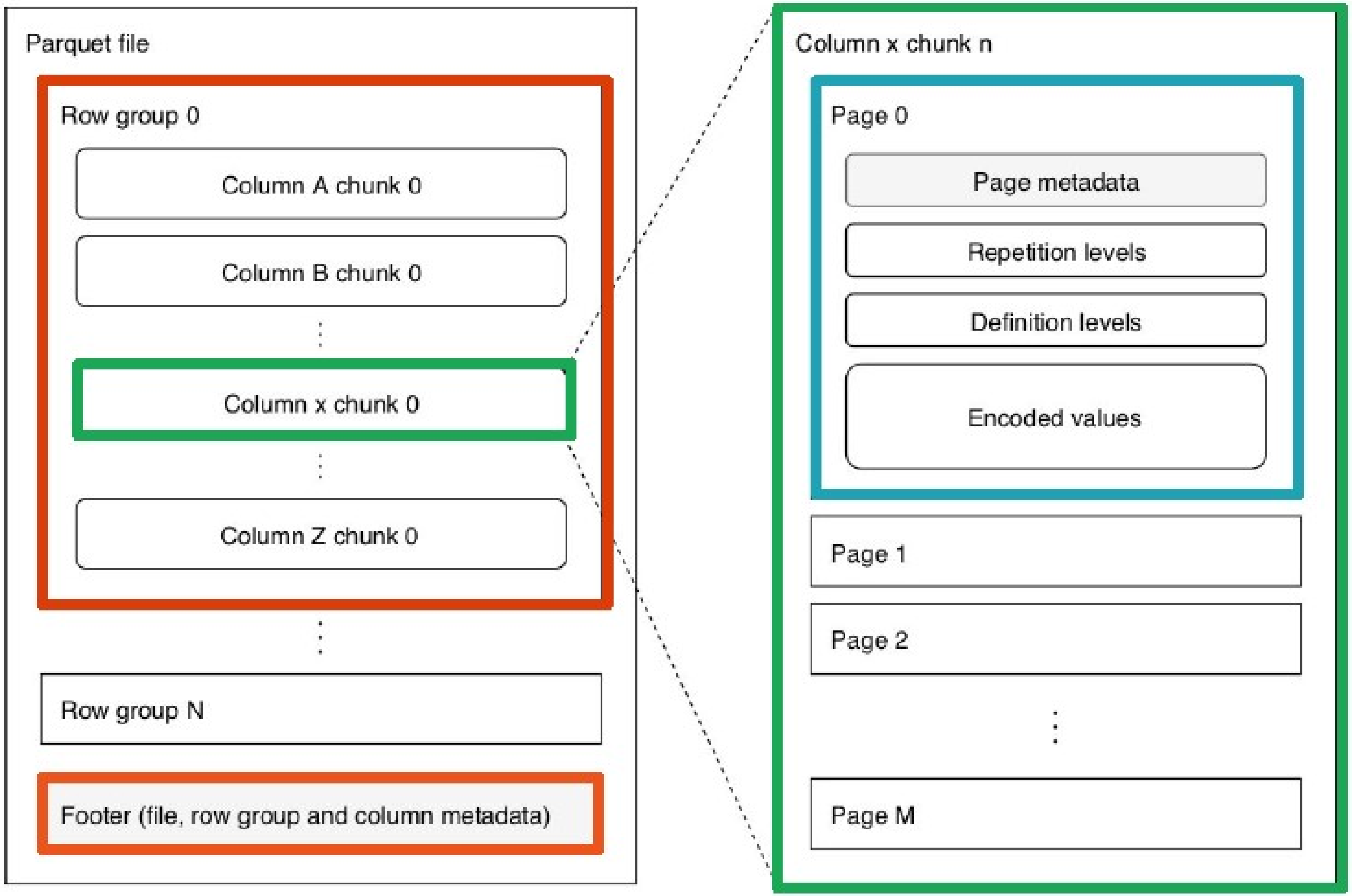


With AQE Skew Join

About data formats



The Parquet (hybrid) file format



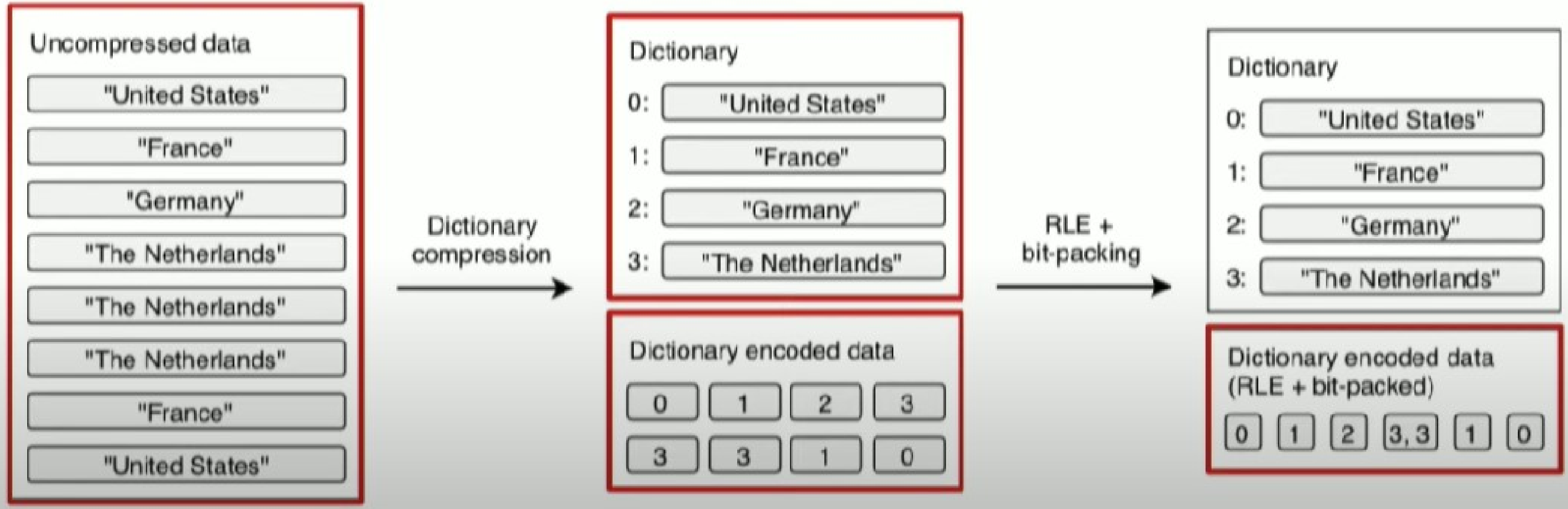
Optimization: predicate pushdown

```
SELECT * FROM table WHERE x > 5

Row-group 0: x: [min: 0, max: 9]
Row-group 1: x: [min: 3, max: 7]
Row-group 2: x: [min: 1, max: 4]
...
```

Parquet: encoding schemes

- `RLE_DICTIONARY`



References

https://spark.apache.org/

Performance Tuning: https://spark.apache.org/docs/latest/sql-performance-tuning.html

https://www.agilelab.it/blog/spark-3-0-first-hands-on-approach-with-adaptive-query-execution-part-3