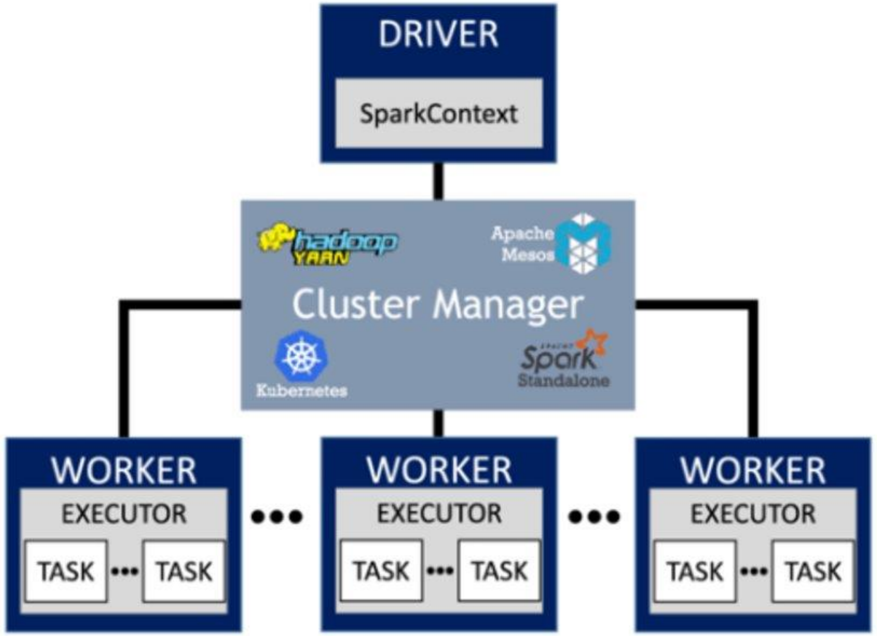
Systems, paradigms and algorithms for Big Data
TD 2

Reminder : tasks, stages and lazy operations



- Narrow transformations: parallel tasks
  - map, mapValues, filter...
- Shuffle operations: move data across workers
  - reduceByKey, join...
- Actions: evaluate
  - count, take, collect
- Stage: sequence of tasks between shuffles

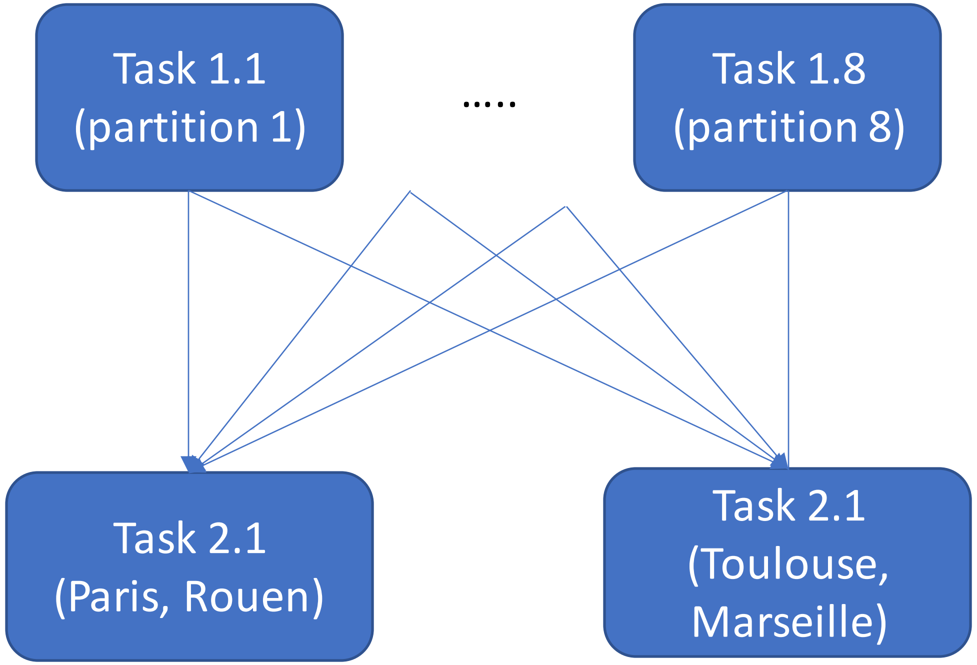
Reminder : Shuffle

Spark.read(…).csv(/path/to/csv).keyBy('City').mapValues(x ➔ x['consumption']).reduceByKey(x,y ➔ x+y)

Client-id	City	Yearly Energy Consumption (kWh)
Alice	Paris	380
Bob	Toulouse	200
..	…	…
Jean	Paris	390
Isabelle	Rouen	350

Partition 1

Partition 8



No shuffle if your dataset is already partitionned by the key you want to aggregate-on !

DataFrame vs RDD

Feature	DataFrame	RDD
Data Format	Structured, organized in columns	Anything, but schema to be passed to spark.
Compile-time safety	No	Yes
API	High Level, spark take care of optimizations for you	Low Level
Memory Management	Off-heap (because spark knows the schema it is working with)	Heap (implies serialization, garbage collection)

DataFrame Concepts

- **DataFrame**
  - contains rows and columns
  - immutable
  - dag of operations
- **Column**
  - there is more to it than a mere ‘column’
  - can be an expression of other columns (e.g. ‘a+b’)
  - can be an aggregation of a column : ‘avg(rating)’ (n rows ➔ 1 row)
  - can be an explosion of a column (1row ➔ k rows)
  - ***pyspark.sql.functions***
- **GroupedData**
  - similar to pandas
  - intermediary object when doing groupBy
  - one needs to call aggregation function on it to get back to a dataframe

RDD vs DataFrame - Actions

RDD
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":"Sam", "movie":"Blade Runner", "rating":3.5}

→ **count** → **3**

df.count()

RDD
{"user":"John"}
{"user":"Louise"}
{"user":"Sam"}

→ **take(2)** →

list
{"user":"John"}
{"user":"Louise"}

df.take(2)

df.show(2)

RDD
{"user":"John"}
….
{"user":"Sam"}

→ **collect()** →

list
{"user":"John"}
….
{"user":"Sam"}

df.collect()

df.toPandas()

RDD vs Dataframe – Narrow transformations

RDD
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":"Sam", "movie":"Blade Runner", "rating":3.5}

→ **map(lambda: x x['movie'])** →

df.select('movie')
Df.withColumn(...)

RDD
"Blade Runner"
"Dirty Dancing"
"Blade Runner"

RDD of (K, V) pairs
("John", "Blade Runner")
("Louise", "Dirty dancing")
("Sam“, "Blade Runner")

→ **mapValues(lambda x: len(x[1]))** →

RDD of (K, V) pairs
("John", 12)
("Louise", 13)
("Sam", 12)

RDD
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":"Sam", "movie":"Blade Runner", "rating":3.5}

→ **filter(lambda x: x['rating']>4.0)** →

df.filter('rating>3.5')

RDD
{"user":"John", "movie":"Bl…
{"user":"Louise", "movie":"Dir…

RDD vs DataFrame – Aggregations

RDD
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":"Sam", "movie":"Blade Runner", "rating":3.5}

keyBy(lambda x: x['user'])

RDD of (K, V) pairs
("John", …)
("Louise", …)
("Sam", …)

RDD
5.0
5.0
3.5

reduce(lambda x,y: x+y)

13.5

import pyspark.sql.functions as F
df.select(F.sum('rating'))

RDD
{"user":"John", "movie":"Blade Runner", "rating":5.0}
{"user":"Louise", "movie":"Dirty dancing", "rating":5.0}
{"user":"Sam", "movie":"Blade Runner", "rating":3.5}

reduceByKey(lambda x,y : x+y)

RDD of (K, V) pairs
("Blade Runner", 8.5)
("Dirty dancing", 5.0)

import pyspark.sql.functions as F
df.groupBy('user').agg (F.sum('rating'))
df.groupBy('user').agg ({'rating':'sum'})

RDD vs DataFrame - Explosion

RDD
{"movie":"Blade Runner", "genres":"cyberpunk|scifi|action"}
{"movie":"Dirty dancing", "genres":"music|danse|romance"}

flatmap(lambda x: x['genres'].split(';'))

RDD
"cyberpunk"
"scifi"
"action"
"music"
"danse"
"romance"

```
@udf("array<string>")
def get_genres(genres: str):
   return genres.split('|’)

import pyspark.sql.functions as F
df.select(F.explode(get_genres(‘genres’)))
```

Scraping around...

Other operations

- Join
- Sort
- Windowing (when you need context on previous/following records to process a record, e.g. compute moving average, get the rank of a record...)

SQL syntax

```
ratings_df.createOrReplaceTempView("Ratings")
df = sql('''select Ratings.id_movie, SUM(Ratings.rating) as s
        from Ratings
        where Ratings.user_id=2
        group by Ratings.id_movie''')
```

Appendix - Explaining Explain

Keyword	Meaning
FileScan	Data read
InMemoryRelation InMemoryTableScan	When caching has been done
Exchange	Shuffle
HashAggregate SortAggregate	When aggregating!
BatchEvalPython	User defined function
Project	Defining new column
AdaptativeSparkPlan	Spark may want to change the physical plan at runtime based on statistics collected.

Appendix - Explaining Explain - Partitionning

nb_records_by_key : dict<string, int> : amount of records in dataset, for each distinct key
n : total amount of records

Exchange RangePartitioning(k)

- Divides dataset in k partitions
- Each partition contains all records with same key
- Each partition roughly contains n/k records
- *nb_records_by_key* estimated with Reservoir Sampling algorithm

Exchange RoundRobinPartitioning

- When called by repartition
- First record goes to first partition
- Second goes to second partition, etc...
- Modulo amount of partitions

Appendix - Parquet

Columnar Storage

- One doesn't need to read all lines completely if only one column needed.

Metadata

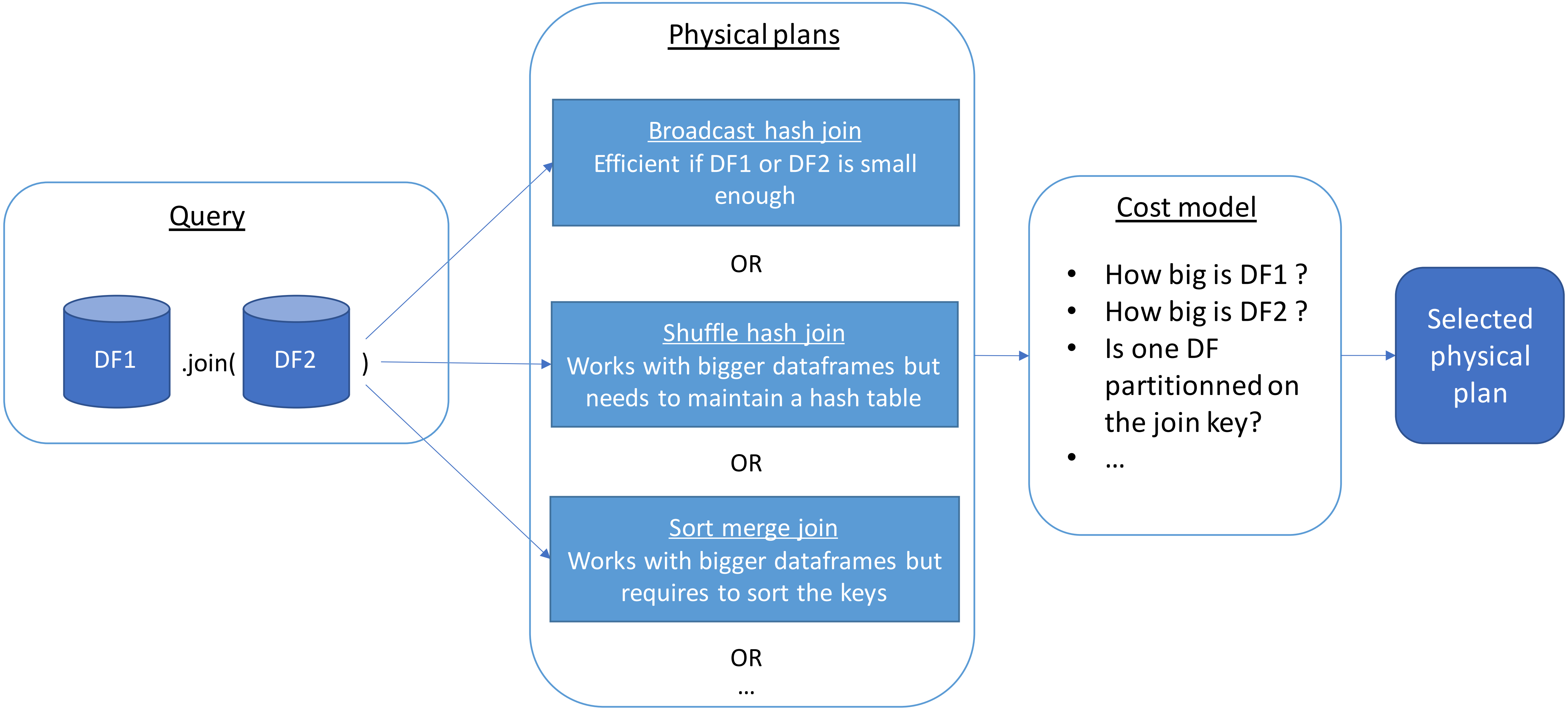
- Associated to each column chunk
- Min/Max values stored in metadata or even distinct values
- One doesn't need necessarily needs to read a chunk when filtering on a given column

Sorting

- Dataset sorted on **ONE** column
- Filtering on this column particularly efficient

Appendix - Catalyst

A smart engine to optimize your operations



Appendix - Nice reads

- Nice read about partitionning, shuffles, execution plans, lazyness : https://ggbaker.ca/data-science/content/spark-calc.html
- RDD vs Dataframe : https://data-flair.training/blogs/apache-spark-rdd-vs-dataframe-vs-dataset
- Execution plans : https://medium.com/datalex/sparks-logical-and-physical-plans-when-why-how-and-beyond-8cd1947b605a