Spark

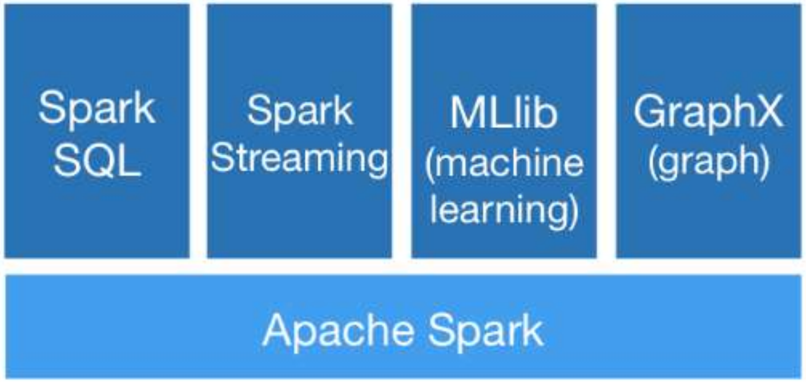
Jakub Yaghob

Spark

- Apache Spark
  - Unified analytics engine for large-scale data processing
  - Initial release: 2014
  - Speed
    - Much faster then Hadoop
  - Easy of use
    - Write applications in Java, Python, R, and Scala
    - Interactive shell
  - Generality
    - Combine streaming, ML, SQL, and analytics
  - Runs everywhere
    - Spark runs on Hadoop, Mesos, Kubernetes, standalone, or in a cloud

Spark

- Contains
  - High-level API in Java, Python, R, and Scala
  - Optimized engine for general execution graph
    - DAG
    - MapReduce with only 2 levels
  - High-level tools
    - SparkSQL
    - MLlib
    - Spark Streaming
    - GraphX

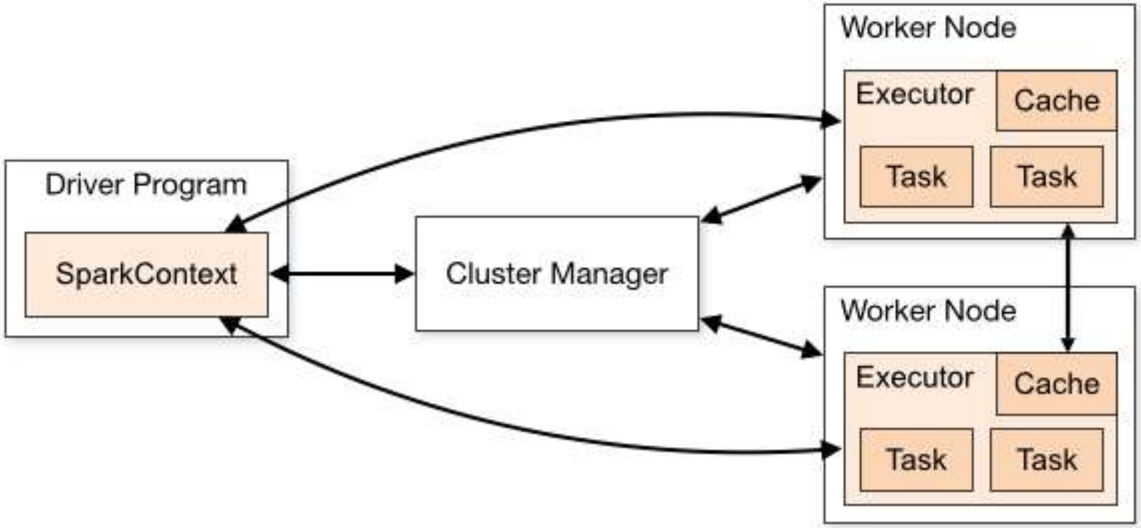


Spark application

- Components
  - Application
    - Independent set of processes on a cluster (+isolation, -no shared data among applications without writing to an external storage)
    - Coordinated by `SparkContex` object in your main program (driver program)
    - Driver program must listen for incoming connections from its executors – must be network addressable from executors
    - Driver schedules tasks – it should be on the same local network
    - Driver program has a web UI – tasks, executors, storage
  - Cluster manager
    - Allocate resources across applications
  - Executor
    - On nodes of the cluster
    - Executes computations and stores data
    - Stays up for the duration of the application
    - Runs task in multiple threads

Spark application structure

- App structure
  - Create `SparkContext` object
  - Connect to the cluster manager(s)
  - Acquire executors from cluster manager(s)
  - Send your application code to the executors
    - Pass Java/Python code/files to the `SparkContext`
  - `SparkContext` sends tasks to the executors to run



Cluster managers

- Cluster managers
  - Spark is agnostic to the underlying cluster manager
  - Supported cluster managers
    - Standalone
      - Simple CM included with Spark, easy to setup a cluster
    - Apache Mesos
      - General CM able to run Hadoop MapReduce and service applications
    - Hadoop YARN
      - Resource manager in Hadoop 2
    - Kubernetes
      - Automated deployment, scaling, and management of containerized applications

Data holders

- Possible data holders
  - RDD
    - Resilient distributed dataset
    - Immutable collection of elements partitioned across the nodes that can be operated in parallel
    - Possibly in-memory
    - Automatically recover from node failures
    - Core API, from initial release, all languages
  - Dataset
    - Distributed collection of data
    - Since v1.6, available only for Java and Scala
    - Strong typing and lambdas from RDD
    - Using Spark SQL optimized execution engine
  - DataFrame
    - Like Dataset, organized into named columns
    - Since v1.3, all languages
    - Like a table in a relational DB

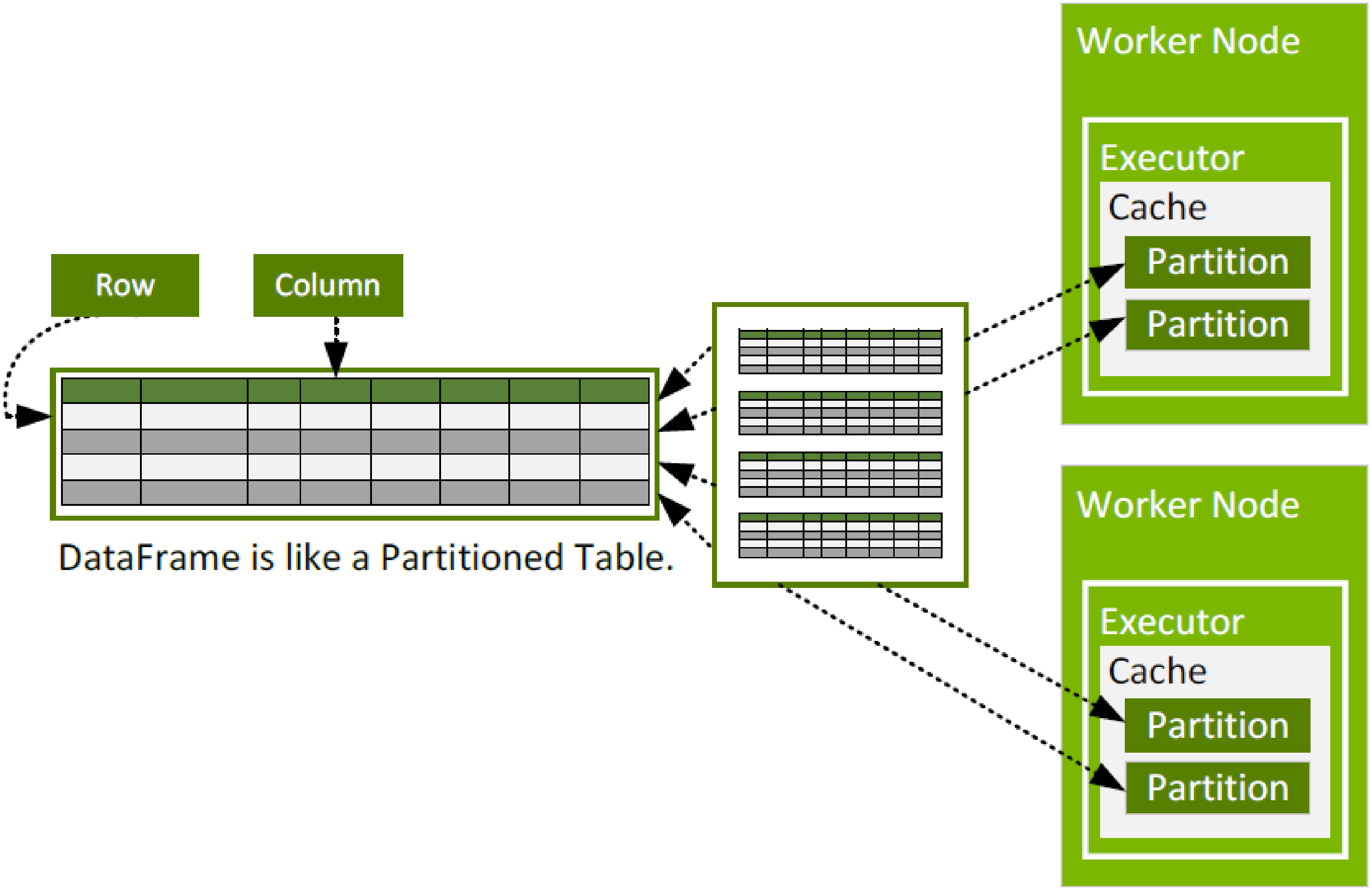
DataFrame

- Construction
  - Well known data file formats
    - CSV, JSON, …
  - Other sources
    - External DB, existing RDD, tables from Hadoop, …
- Transformations
  - Create a new DataFrame from the existing one
  - Lazily evaluated, triggered by an action
- Actions
  - Returns a result to the driver or writes to disk

DataFrame transformations and actions

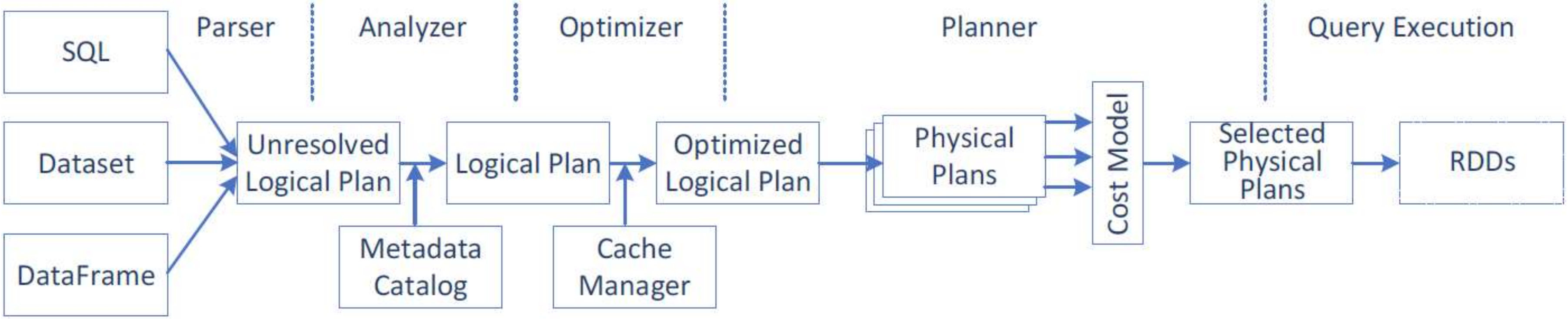
- Transformations (examples)
  - select – select a set of columns
  - join – join with another DataFrame
  - groupBy – groups using specified columns
  - filter – filter rows using a condition (bool or string)
- Actions (examples)
  - show(n) – display the first n rows
  - count – number of rows in DataFrame
  - collect – return data back to the driver
- Example
  - `df.groupBy("hour").count().show(4)`
  - `df.filter(df.age > 3).collect()`

DataFrame partitioning



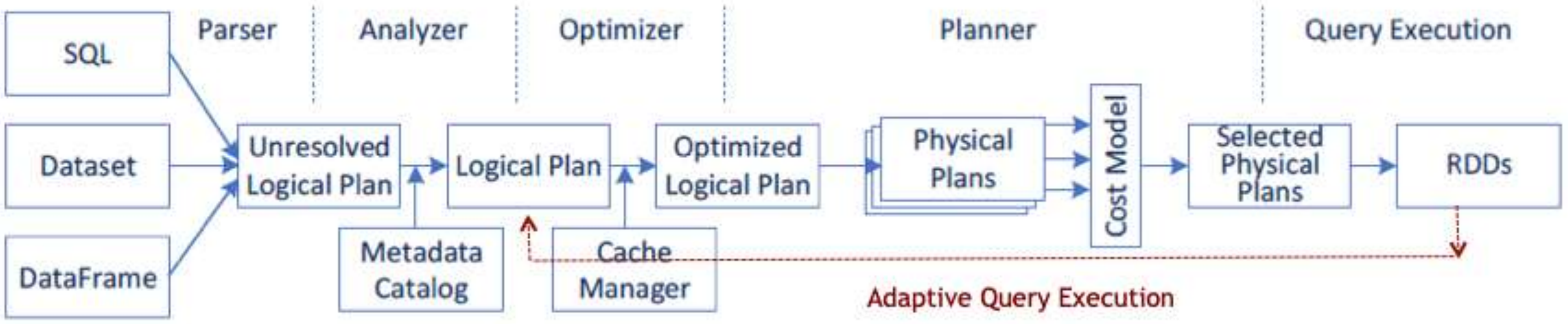
Spark query execution

- Query execution steps
  - Create a logical plan
  - Transform the logical plan to a physical plan
  - Generate code
  - Execute the tasks on a cluster



Advanced Spark 3.x features

- Adaptive Query Execution (AQE)
  - Runtime statistics retrieved from completed stages of the query plan are used to re-optimize the execution plan
  - Dynamically coalescing shuffle partitions
    - Combine adjacent small partitions into bigger partitions, reducing the number of tasks
  - Dynamically switching join strategies
    - Optimize join strategy at runtime based on the join relation size
  - Dynamically optimizing skew joins
    - Detect data skew and split skew partitions into smaller sub-partitions



Advanced Spark 3.x features

- Dynamic partition pruning
  - Data warehouse queries
    - One or more fact tables referencing any number of dimensional tables
  - Pruning at runtime by reusing the dimension table broadcast results in hash joins
- Accelerator-aware scheduling
  - GPU/CUDA with RAPIDS

Parallel aspects

- `SparkContext` type
  - parallelize(col, slices)
    - Distribute a local collection to form an RDD
  - accumulator(ival)
    - Creates an Accumulator with initial value
  - broadcast(ival)
    - Broadcast read-only Broadcast variable to the cluster
- `RDD` type
  - aggregate(zeroval, seqop, combop)
    - Aggregate elements of each partition and then the results for all partitions
  - barrier
    - All tasks launched together
  - cache
    - Partitions cached in memory
  - reduce(op)

Shared variables

- Accumulator
  - Worker tasks can call `add(v)` operation
  - Only driver can read accumulator by calling `value()` function
  - No other operations are defined
- Broadcast
  - Cached read-only variable
  - Tasks can read it by calling `value()` function

Launching application

- Interactive shell
  - Only Scala and Python
    - `YOUR_SPARK_HOME/bin/pyspark`
- Standalone applications
  - Unified launcher
    - `YOUR_SPARK_HOME/bin/spark-submit`
    - Important parameters
      - `--master` – URL of the master node for the cluster
      - `--class` – entry point for the application
  - Master URL
    - `local` - locally with one worker thread (no para)
    - `local[K]` – locally with K worker threads
    - `local[*]` – locally with max number of worker threads (=cores)
    - `spark://HOST:PORT` – standalone Spark cluster master
    - `CM://HOST:PORT` – connect to cluster manager [mesos,yarn,k8s]

Launching application in SLURM environment

- Use prepared environment
  - Environment home
    - `/home/_teaching/para/04-spark`
  - Spark cluster startup script
    - `spark-slurm.sh`
    - Requires
      - Spark Charliecloud image directory – `spark`
      - Network interface with IP networking – `eno1` for w[201-208]
      - R/W directory – your home or project dir, mounted as `/mnt/1`
      - Application – path from the container (/mnt/1/…)
  - Launch the script using `sbatch` command