Part 8 : RHADOOP

R:

R is an open-source programming language that is extensively used for **statistical and graphical analysis.**

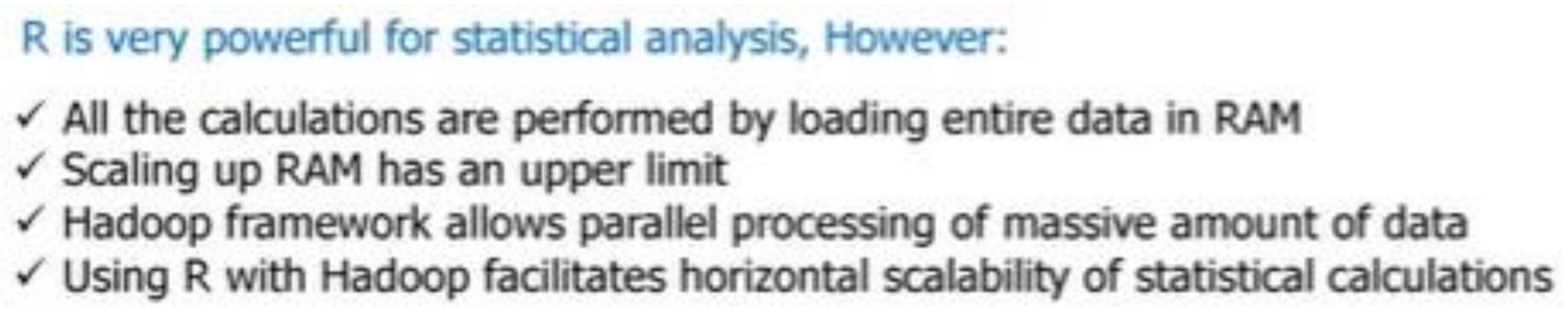
One major quality of **R's** is that it produces well-designed **quality plots** with greater ease, including mathematical **symbols and formulae where needed**.

Some reasons for which R is considered the best fit for data analytics :

- A robust collection of **packages**
- Powerful **data visualization** techniques
- Commendable **Statistical and graphical** programming features
- Object-oriented programming language
- It has a wide smart **collection of operators** for calculations of arrays,

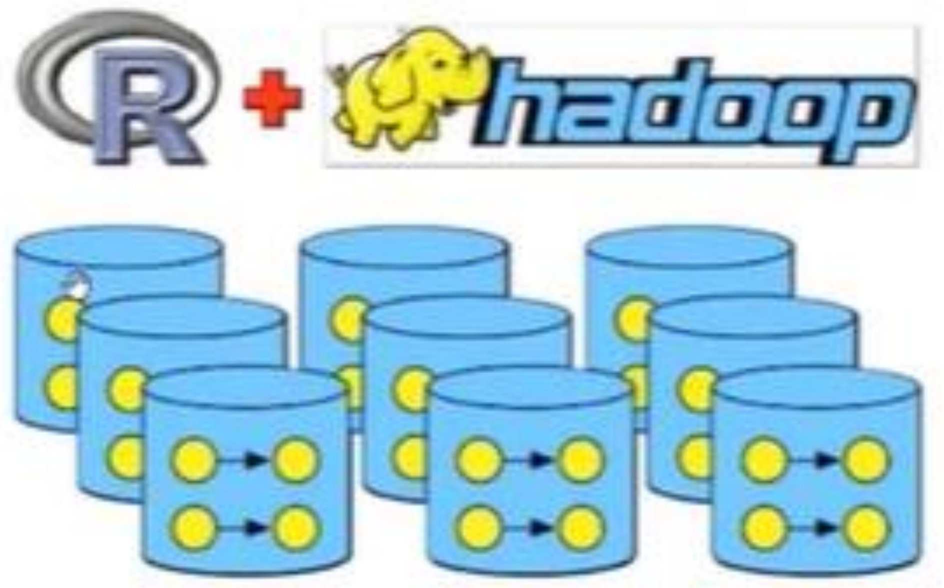
  particular matrices, etc
- Graphical representation capability on display or on hard copy.

R:



R is very powerful for statistical analysis, However:

- ✓ All the calculations are performed by loading entire data in RAM
- ✓ Scaling up RAM has an upper limit
- ✓ Hadoop framework allows parallel processing of massive amount of data
- ✓ Using R with Hadoop facilitates horizontal scalability of statistical calculations

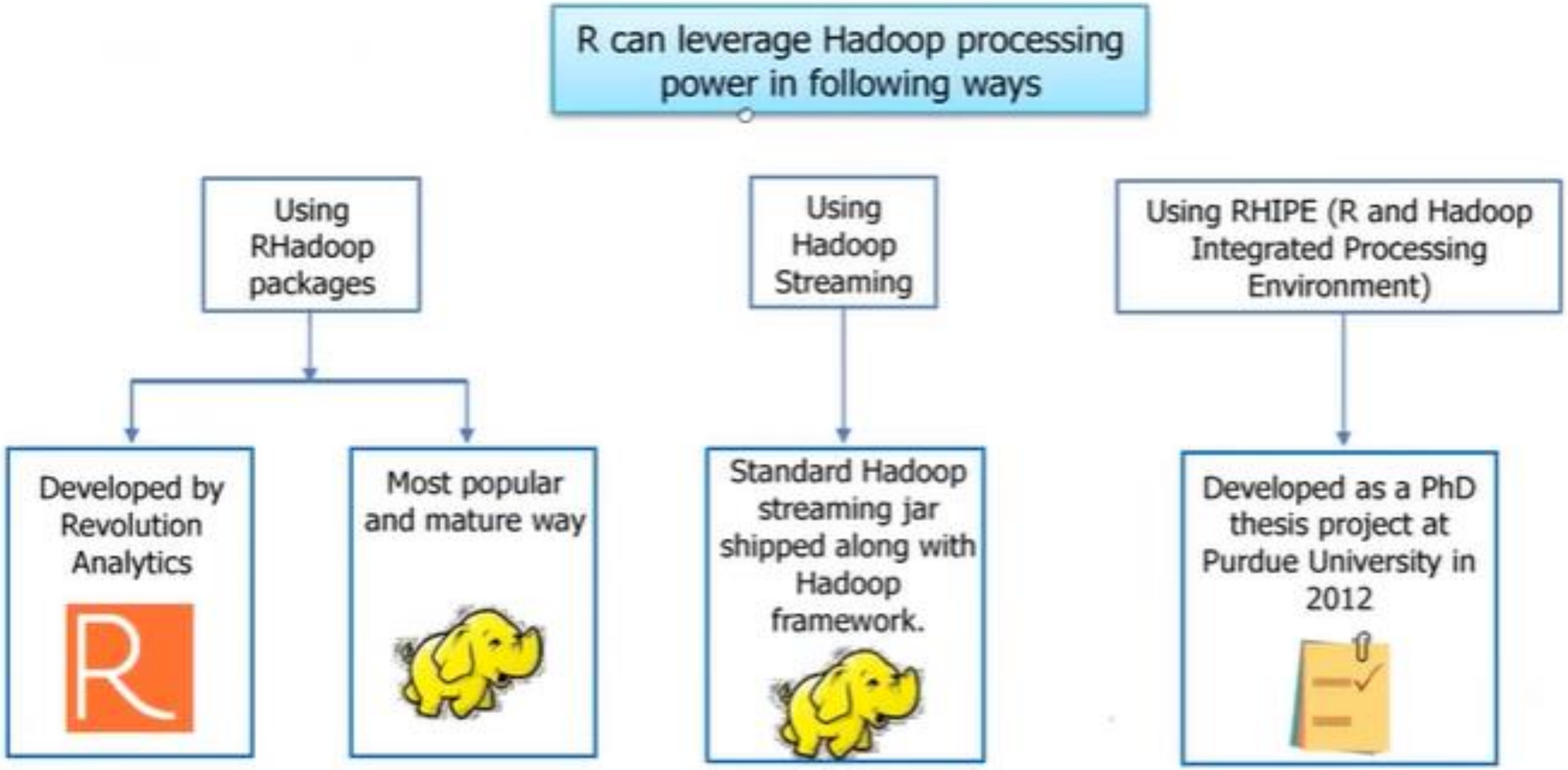


- Rhadoop: Collection of 3 packages for performing large data operations with an R environment.

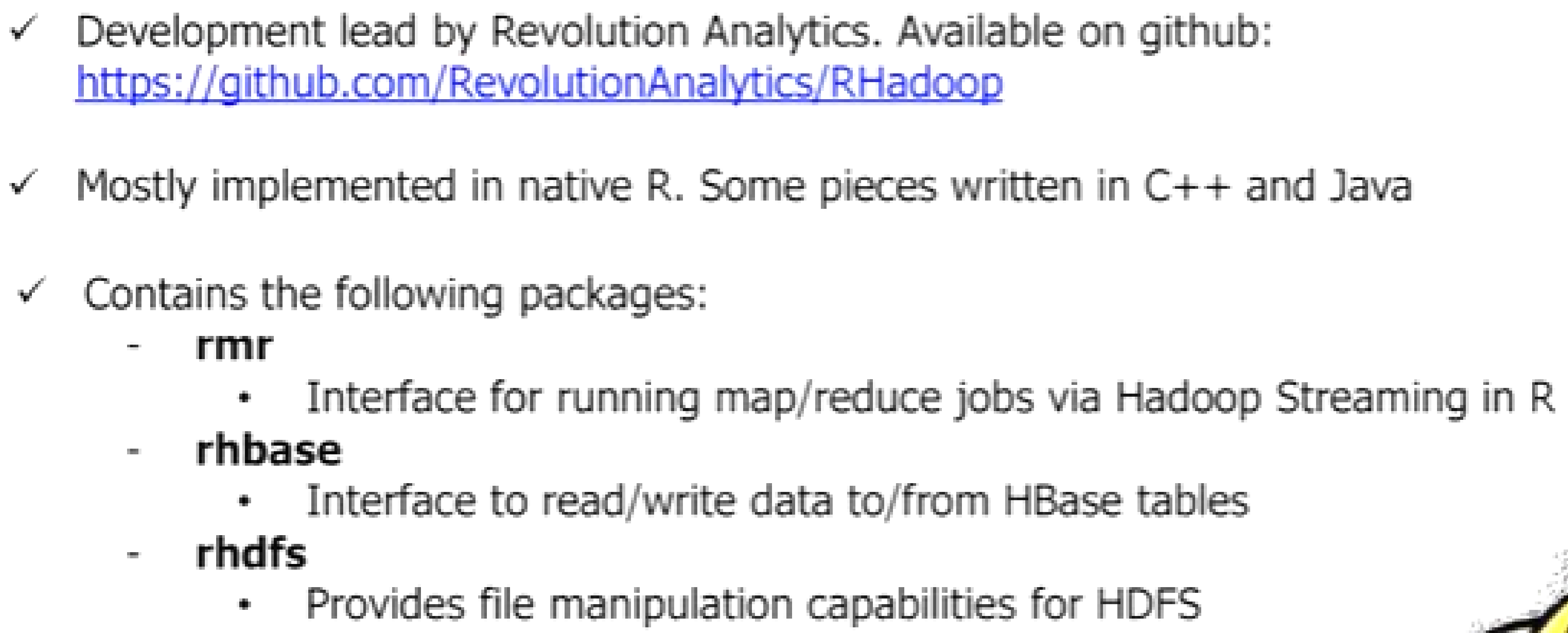
Hadoop & R

- Hadoop for R
  - Datasets that do not fit in memory but can be naturally partitioned
  - Harness well known infrastructure for embarrassingly parallel tasks
- R for Hadoop
  - Very large number of implementations of different functions and libraries (machine learning, text processing etc)
  - Native C implementation – very fast number crunching, matrix processing
  - All in-memory calculations

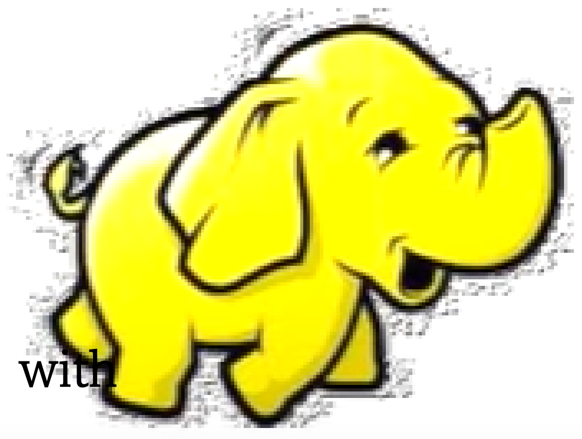
Integrate R and Hadoop: Different Ways


R can leverage Hadoop processing power in following ways
Using RHadoop packages
Using Hadoop Streaming
Using RHIPE (R and Hadoop Integrated Processing Environment)
Developed by Revolution Analytics
Most popular and mature way
Standard Hadoop streaming jar shipped along with Hadoop framework.
Developed as a PhD thesis project at Purdue University in 2012

Exploring RHadoop

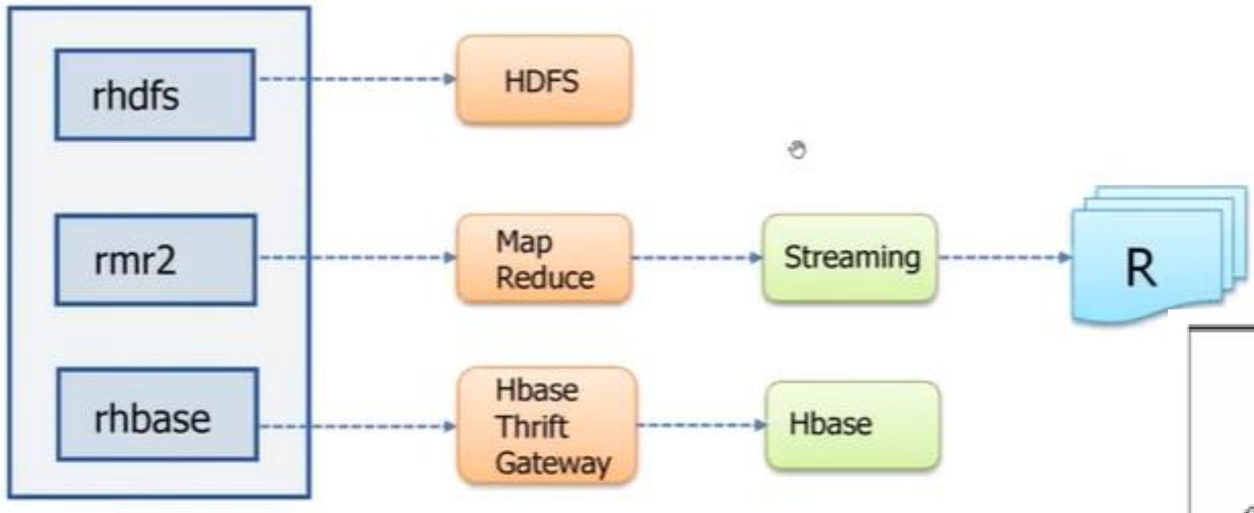


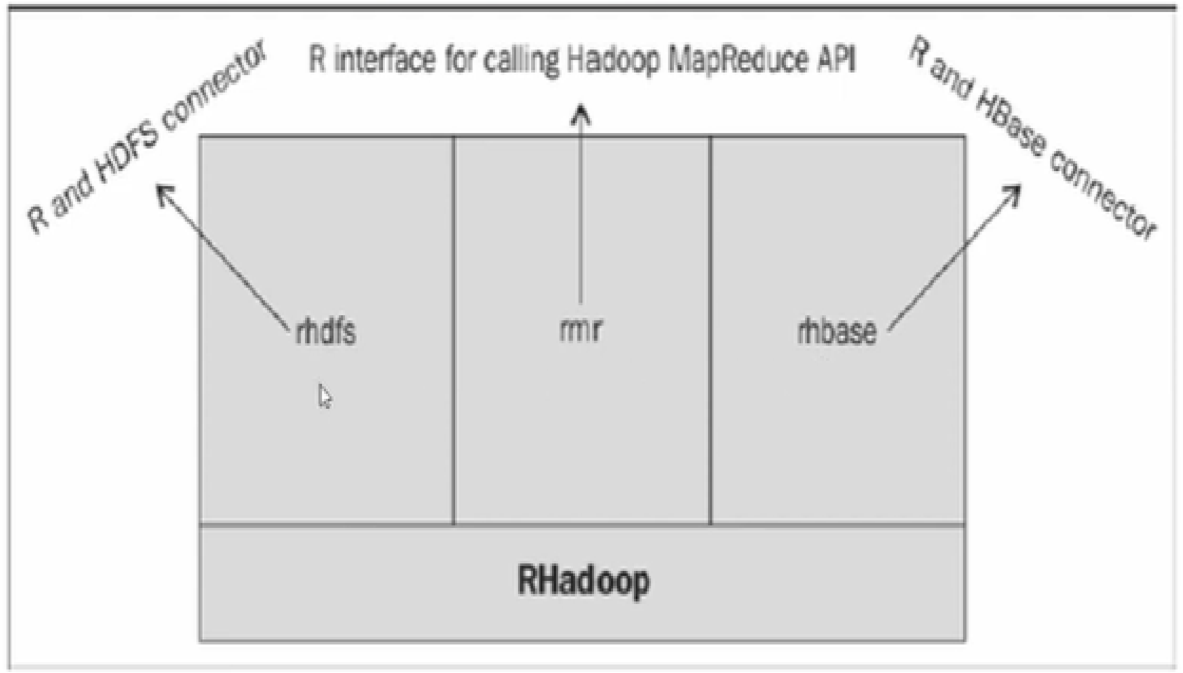
- ✓ Development lead by Revolution Analytics. Available on github: https://github.com/RevolutionAnalytics/RHadoop
- ✓ Mostly implemented in native R. Some pieces written in C++ and Java
- ✓ Contains the following packages:
  - **rmr**
    - Interface for running map/reduce jobs via Hadoop Streaming in R
  - **rhbase**
    - Interface to read/write data to/from HBase tables
  - **rhdfs**
    - Provides file manipulation capabilities for HDFS



- rmr: Helps to execute the Mapping & Reducing codes in R
- rhdfs: Provides file management capabilities by integrating with HDFS.
- rhbase: provides R database management capability with integration with Hbase.

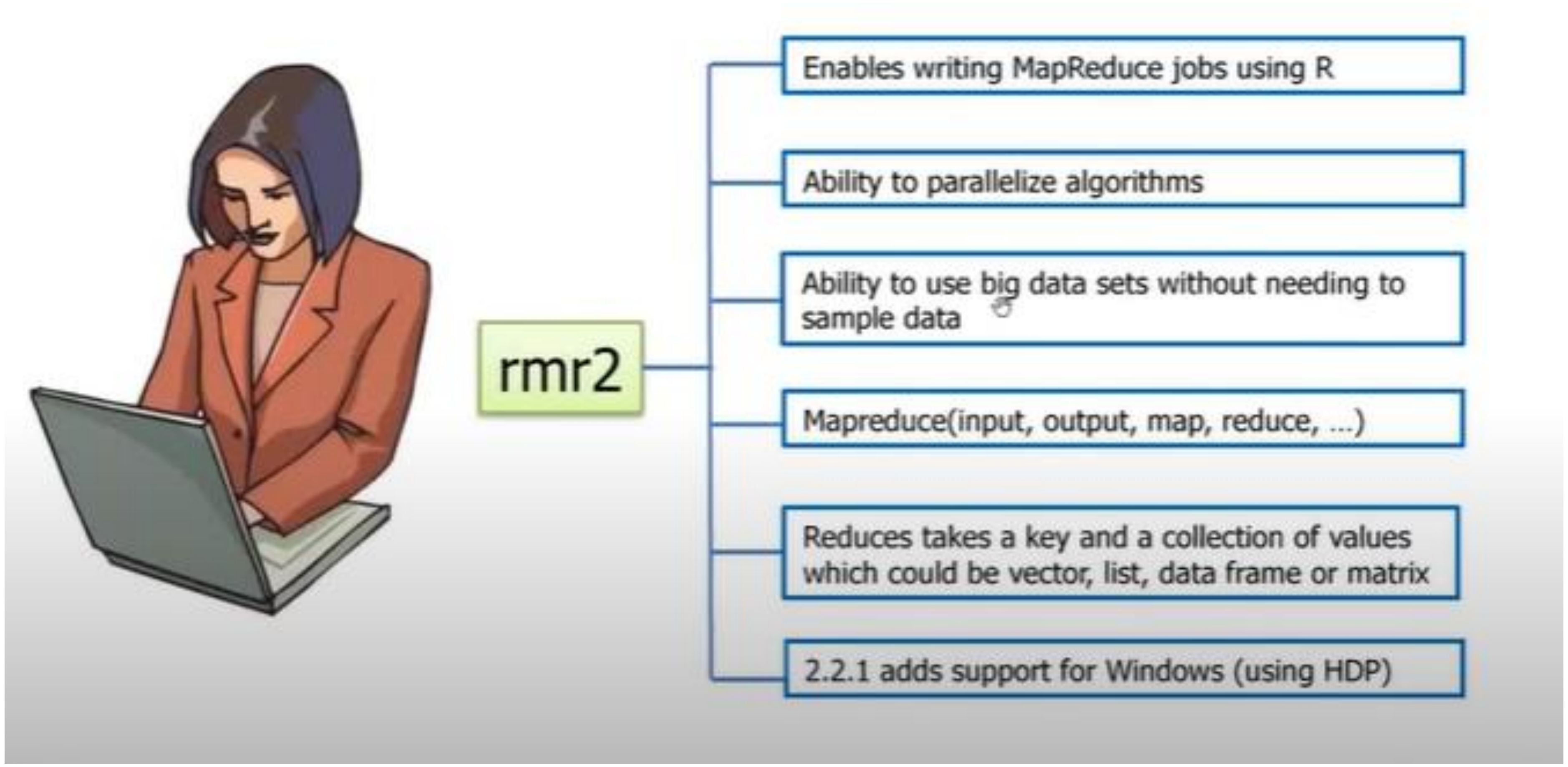
RHadoop Architecture





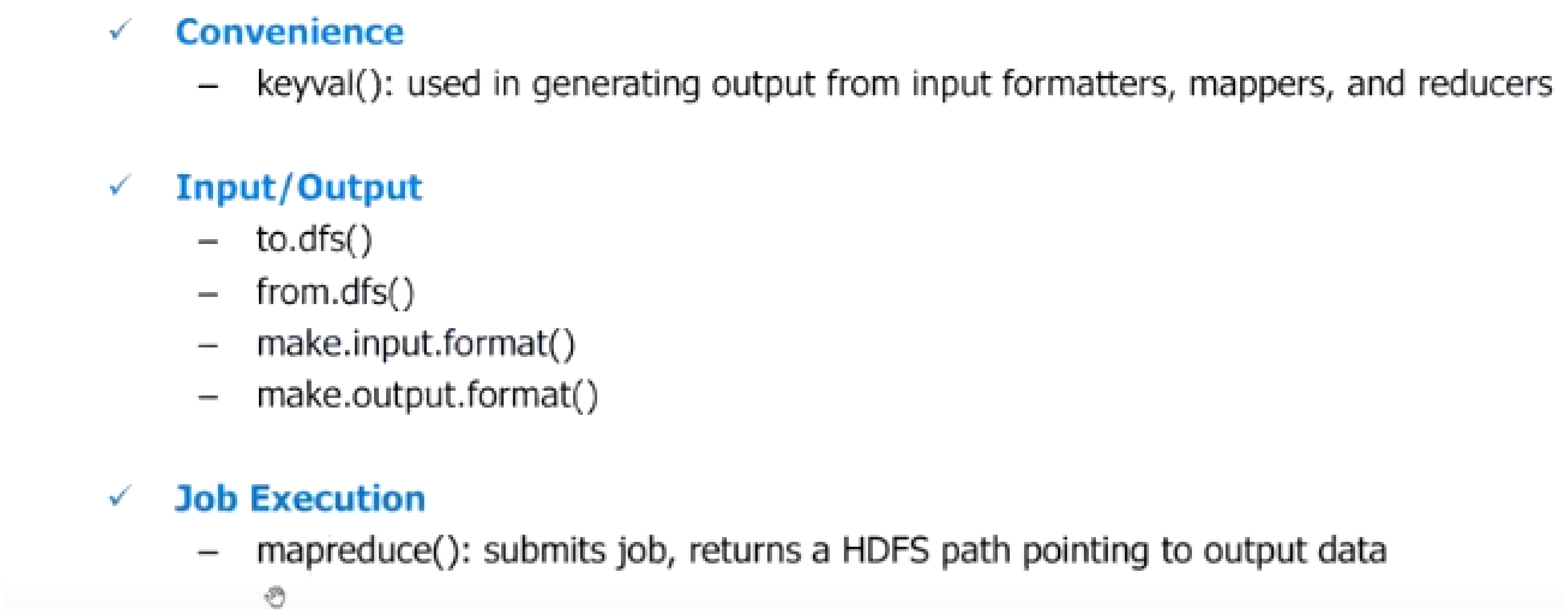
RHadoop Ecosystem

rmr2



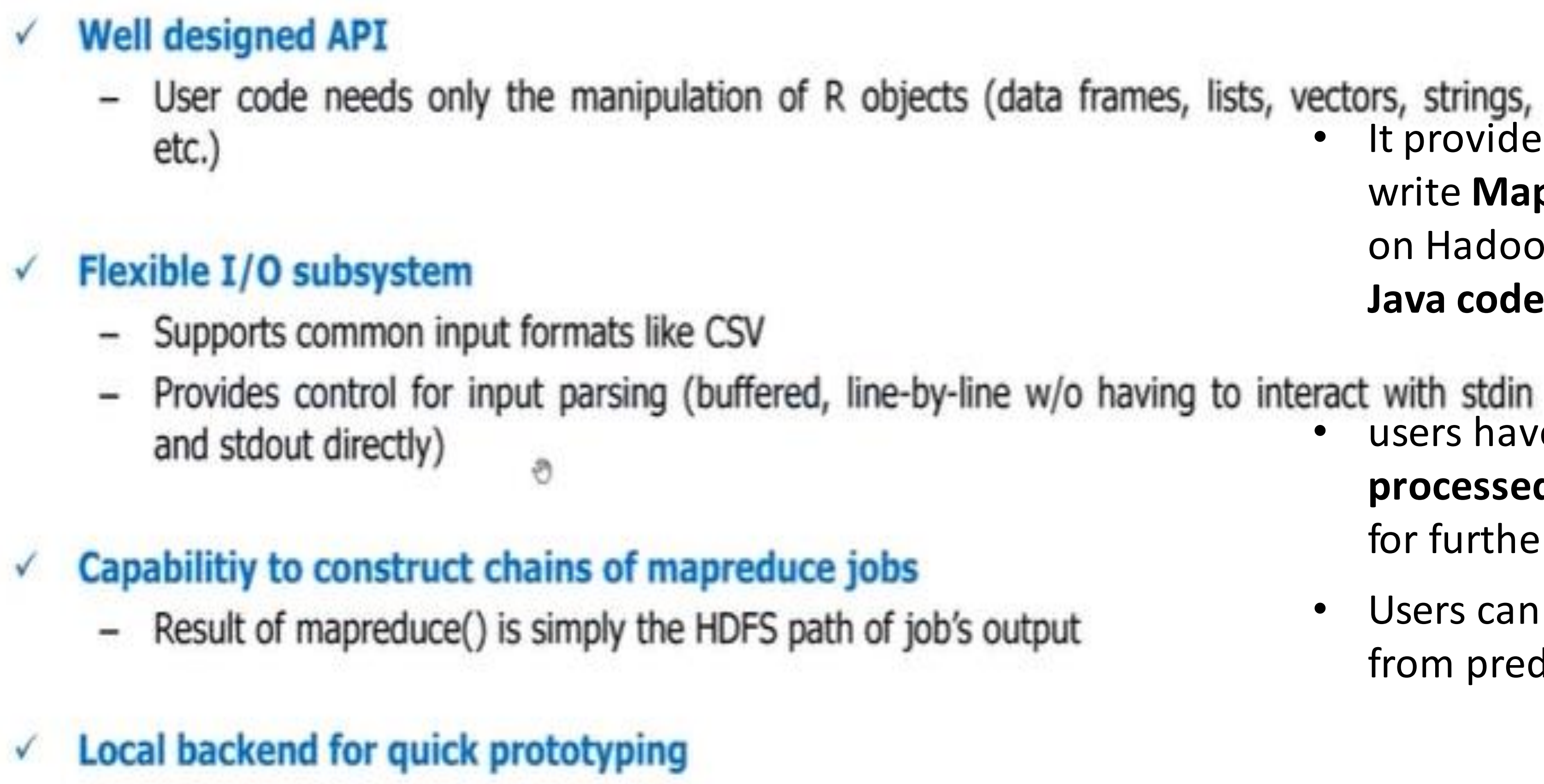
- Enables writing MapReduce jobs using R
- Ability to parallelize algorithms
- Ability to use big data sets without needing to sample data
- Mapreduce(input, output, map, reduce, ...)
- Reduces takes a key and a collection of values which could be vector, list, data frame or matrix
- 2.2.1 adds support for Windows (using HDP)

Functionalities of rmr:



- ✓ **Convenience**
  - keyval(): used in generating output from input formatters, mappers, and reducers
- ✓ **Input/Output**
  - to.dfs()
  - from.dfs()
  - make.input.format()
  - make.output.format()
- ✓ **Job Execution**
  - mapreduce(): submits job, returns a HDFS path pointing to output data

rmr Advantages:



✓ **Well designed API**

- User code needs only the manipulation of R objects (data frames, lists, vectors, strings, etc.)

✓ **Flexible I/O subsystem**

- Supports common input formats like CSV
- Provides control for input parsing (buffered, line-by-line w/o having to interact with stdin and stdout directly)

✓ **Capabilitiy to construct chains of mapreduce jobs**

- Result of mapreduce() is simply the HDFS path of job's output

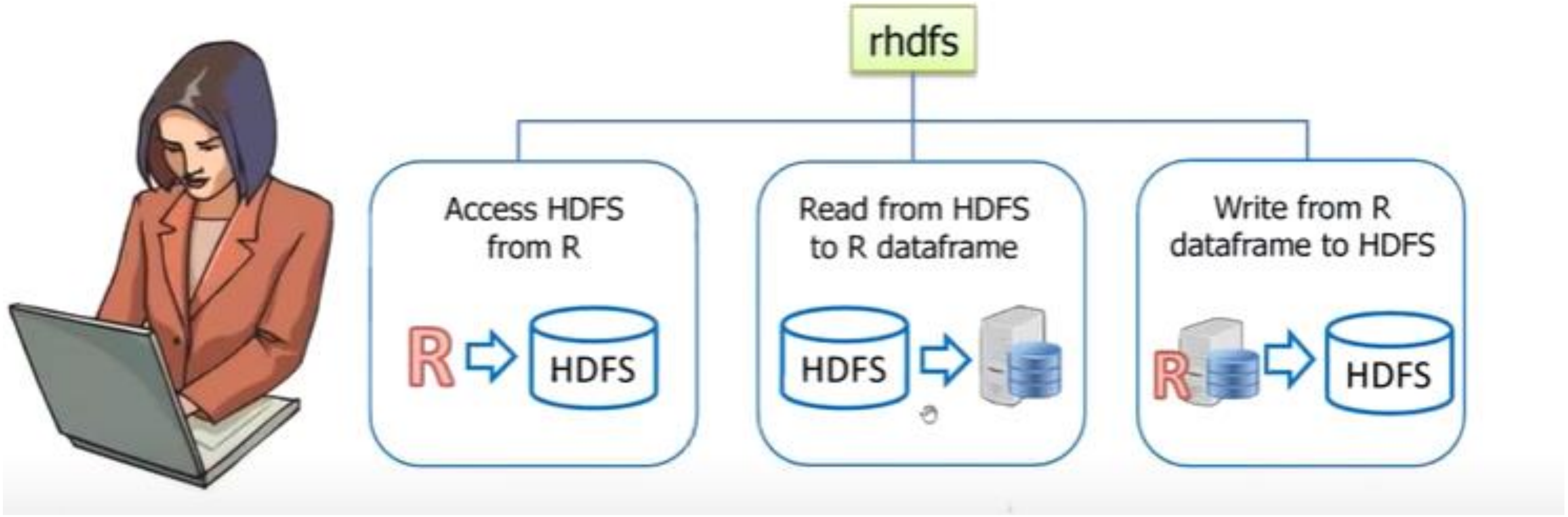
✓ **Local backend for quick prototyping**

- It provides a **convenient interface for R users** to write **MapReduce** programs and execute them on Hadoop clusters **without needing to write Java code directly.**
- users have **full control over how input data is processed and transformed into key-value pairs** for further processing.
- Users can define **custom output formats** or select from predefined formats like text or sequence file.

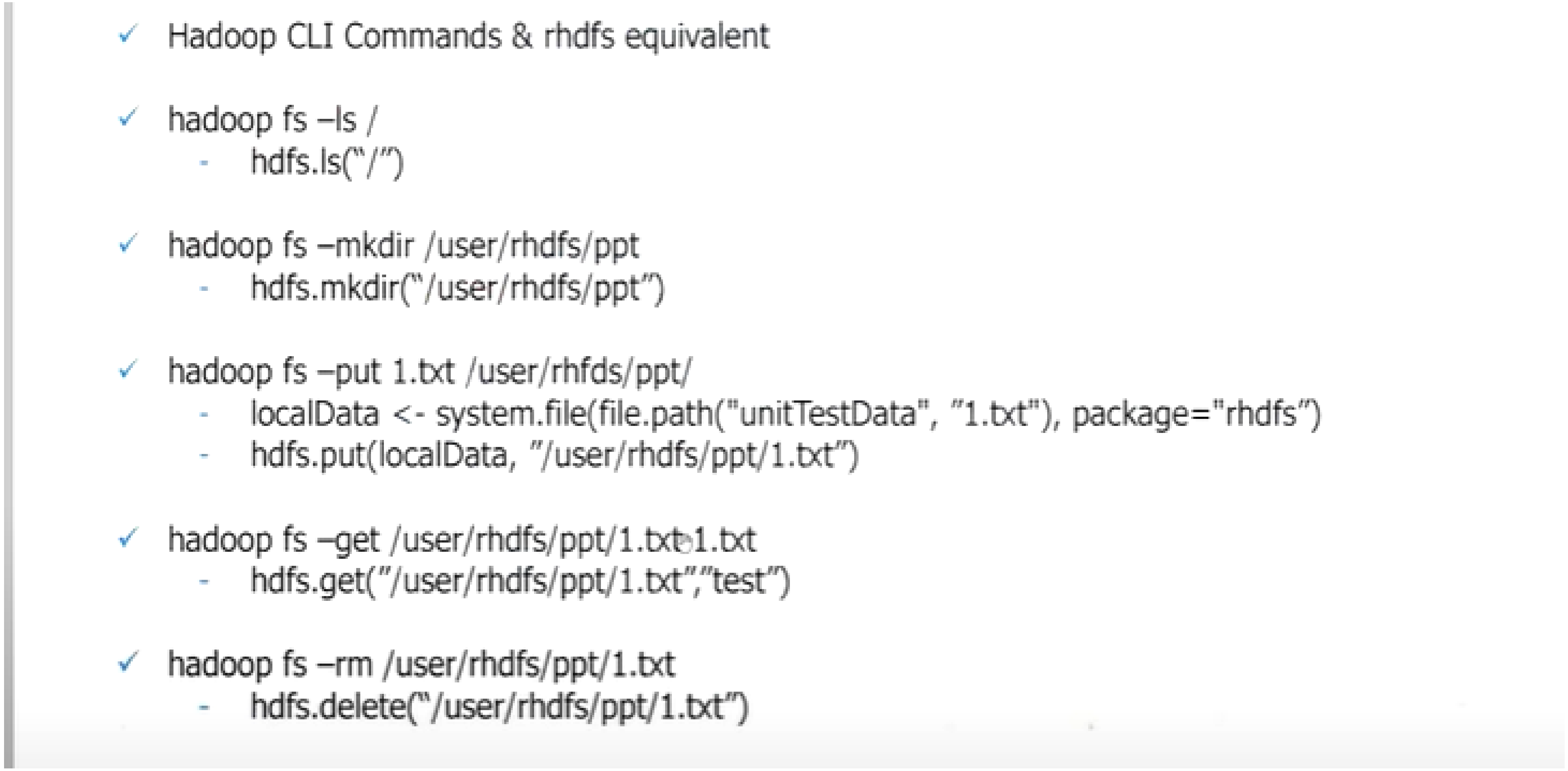
- The "local backend" refers to **running MapReduce jobs** on a **single machine**, utilizing the **local file system** instead of Hadoop's distributed file system (HDFS) and the **local computing resources** instead of a Hadoop cluster. This setup is primarily used for **development, testing, and prototyping purposes.**

- MapReduce job will be executed **locally** within the R environment on the same machine where R is runnin

rhdfs

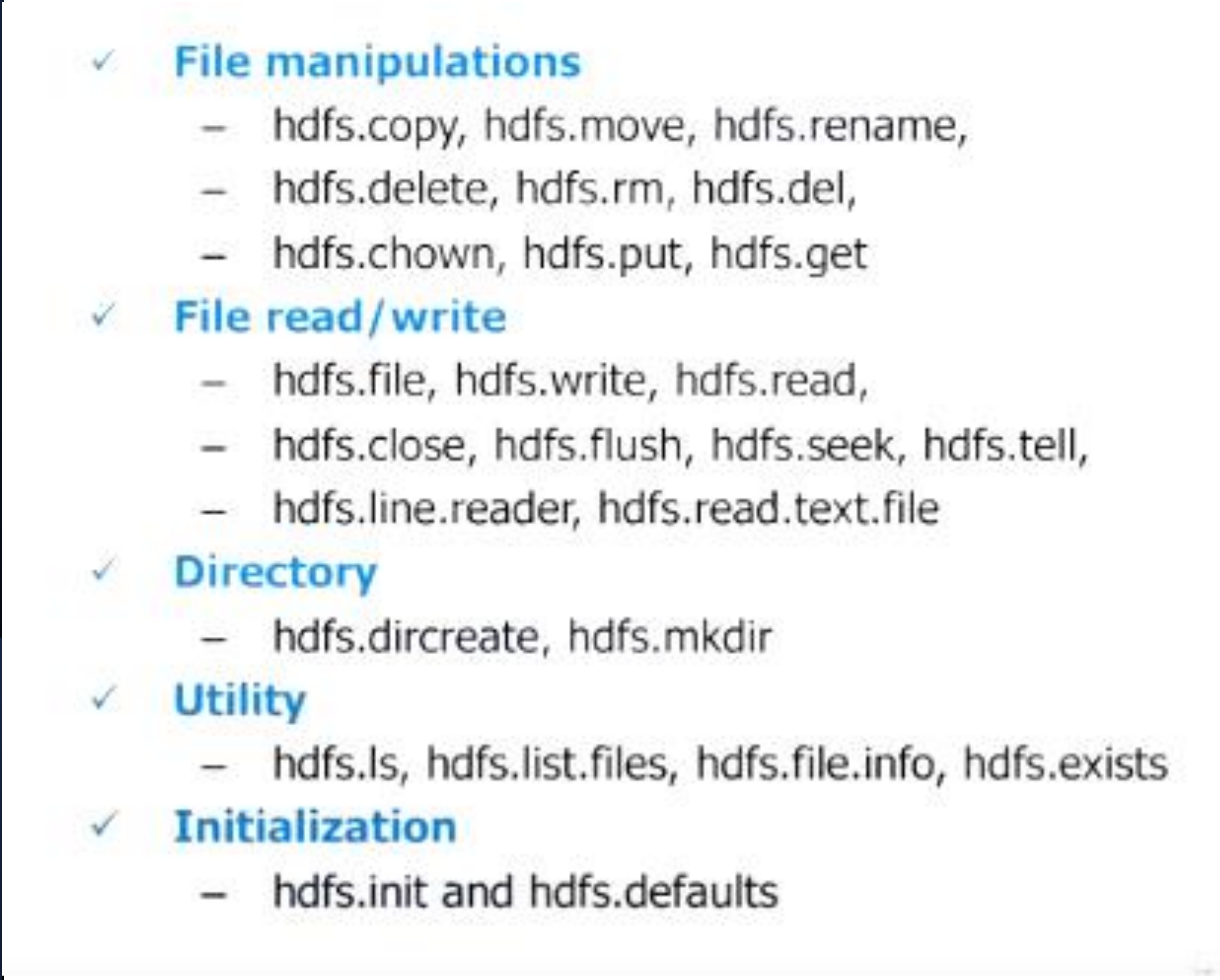

rhdfs
Access HDFS from R
R
HDFS
Read from HDFS to R dataframe
HDFS
Write from R dataframe to HDFS
R
HDFS

rhdfs



- ✓ Hadoop CLI Commands & rhdfs equivalent
- ✓ hadoop fs –ls /
  - hdfs.ls(“/”)
- ✓ hadoop fs –mkdir /user/rhdfs/ppt
  - hdfs.mkdir(“/user/rhdfs/ppt”)
- ✓ hadoop fs –put 1.txt /user/rhfds/ppt/
  - localData <- system.file(file.path("unitTestData", ”1.txt"), package="rhdfs”)
  - hdfs.put(localData, ”/user/rhdfs/ppt/1.txt”)
- ✓ hadoop fs –get /user/rhdfs/ppt/1.txt 1.txt
  - hdfs.get(”/user/rhdfs/ppt/1.txt”,”test”)
- ✓ hadoop fs –rm /user/rhdfs/ppt/1.txt
  - hdfs.delete(“/user/rhdfs/ppt/1.txt”)

Functionalities in HDFS



- ✓ **File manipulations**
  - hdfs.copy, hdfs.move, hdfs.rename,
  - hdfs.delete, hdfs.rm, hdfs.del,
  - hdfs.chown, hdfs.put, hdfs.get
- ✓ **File read/write**
  - hdfs.file, hdfs.write, hdfs.read,
  - hdfs.close, hdfs.flush, hdfs.seek, hdfs.tell,
  - hdfs.line.reader, hdfs.read.text.file
- ✓ **Directory**
  - hdfs.dircreate, hdfs.mkdir
- ✓ **Utility**
  - hdfs.ls, hdfs.list.files, hdfs.file.info, hdfs.exists
- ✓ **Initialization**
  - hdfs.init and hdfs.defaults

`hdfs.copy`, `hdfs.move`, `hdfs.rename`, `hdfs.delete`, `hdfs.rm`, `hdfs.del`, `hdfs.chown`, `hdfs.put`, `hdfs.get`

- File Read/Write

`hdfs.file`, `hdfs.write`, `hdfs.close`, `hdfs.flush`, `hdfs.read`, `hdfs.seek`, `hdfs.tell`, `hdfs.line.reader`, `hdfs.read.text.file`

- Directory

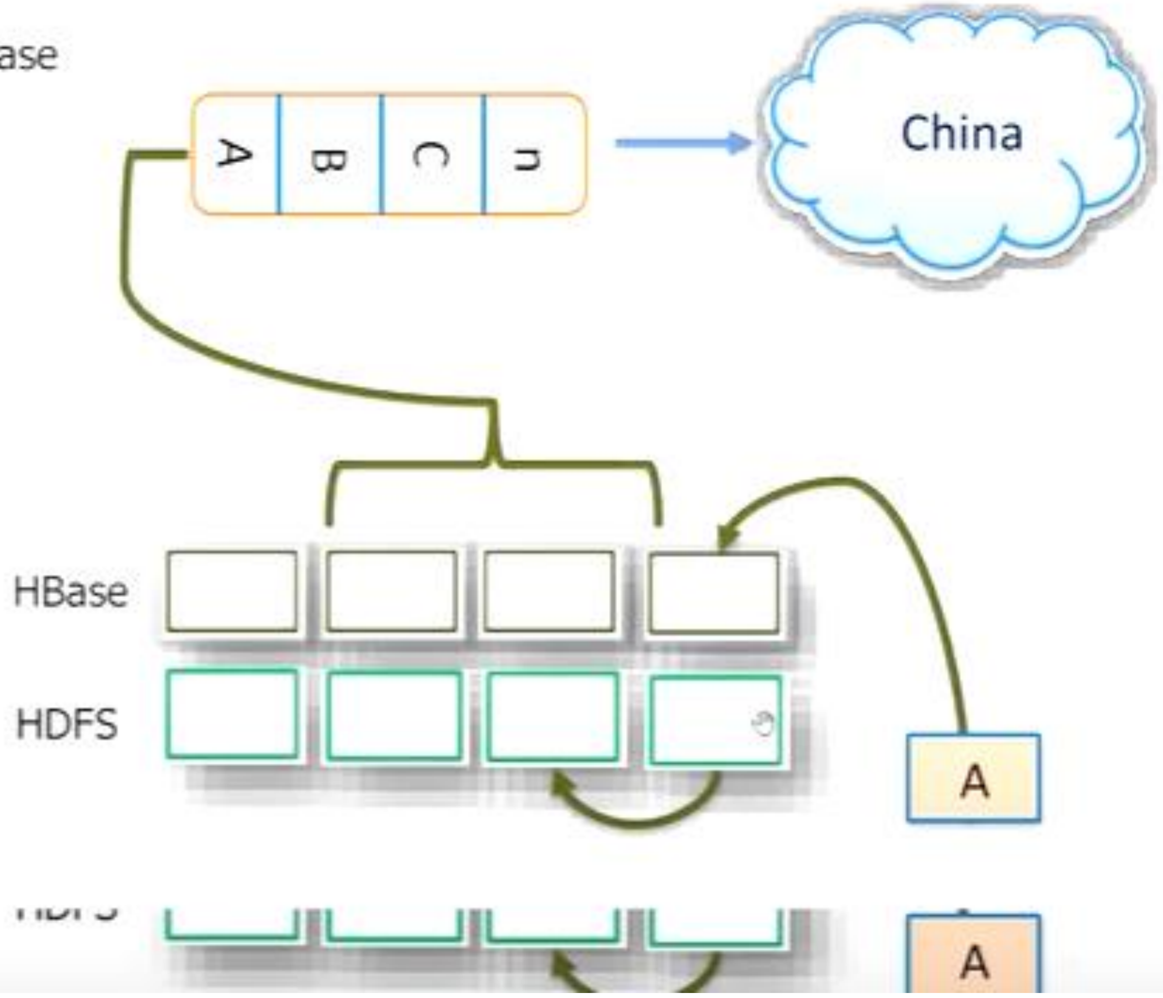
`hdfs.dircreate`, `hdfs.mkdir`

- Utility

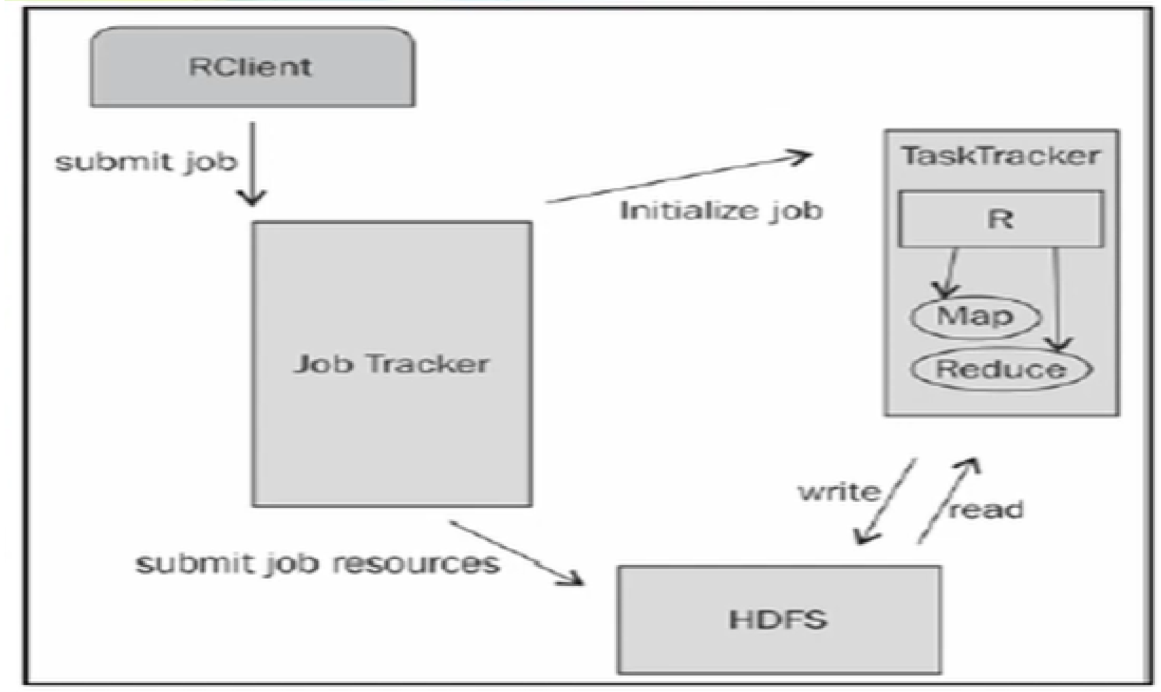
`hdfs.ls`, `hdfs.list.files`, `hdfs.file.info`, `hdfs.exists`

rhbase

- ✓ Access and change data within HBase
- ✓ Uses Thrift API
- ✓ Command Examples
  - hb.new.table
  - hb.insert
  - hb.scan.ex
  - hb.scan



RHIPE:



Components of RHIPE

- RHIPE: Stands for R and Hadoop Integrated Programming Environment
- RHIPE involves working with R and Hadoop Integrated Programming Environment

- R client : submits Query (Job)to job tracker.
- Job Tracker : Main coordinator – Responsible to allocating job/assign task/job to task tracker.
- Task Tracker : May be present in same location or different nodes.
- Task Tracker : Perform Map Reduce - Responsible for "Processing of Data".
- Parallel to distributed computing.
- HDFS: During Processing : It can read and write data from HDFS

RHIPE Components:

RClient:

RClient is an R application that calls the JobTracker to execute the job with an indication of several MapReduce job resources such as Mapper, Reducer, input format, output format, input file, output file, and other several parameters that can handle the MapReduce jobs with RClient.

- **JobTracker:**

A JobTracker is the master node of the Hadoop Map Reduce operations for initializing and monitoring the MapReduce jobs over the Hadoop cluster..

RHIPE Components:

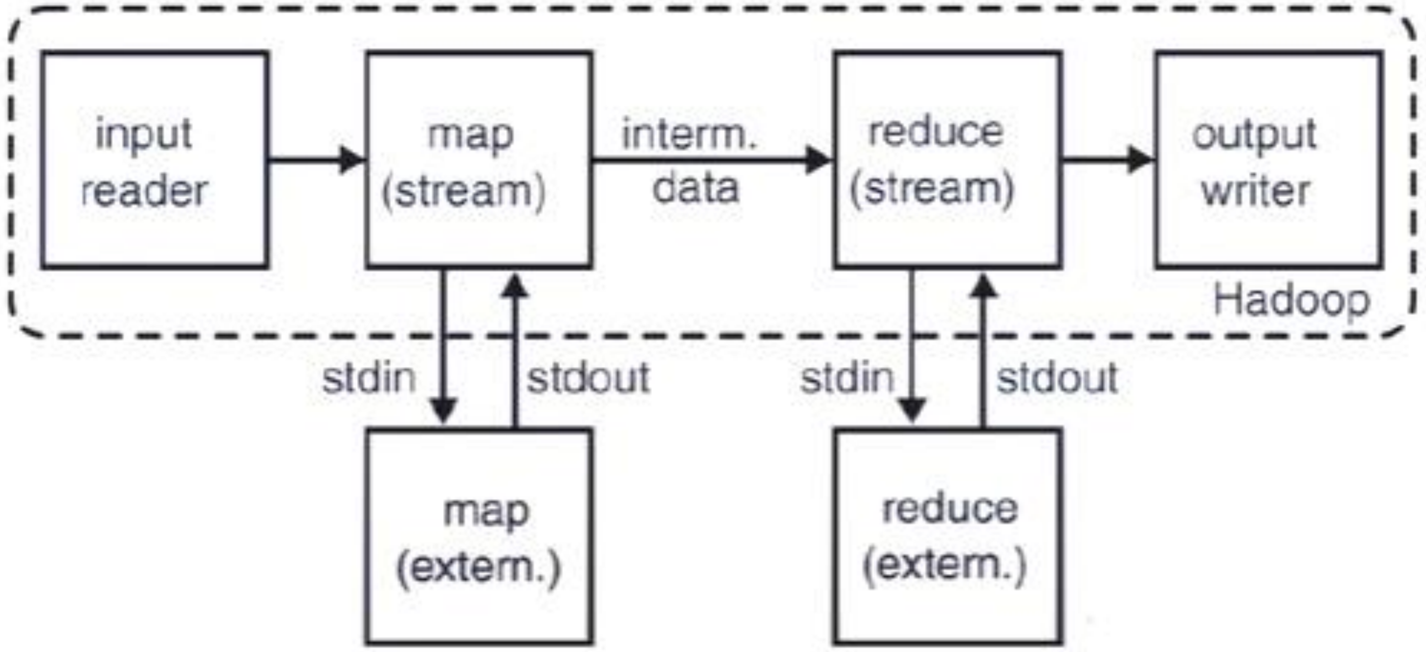
Task Tracker:
Task Tracker is a slave node in the Hadoop cluster. It executes the Map Reduce jobs as per the orders given by Job Tracker, retrieve the input data chunks, and run R-specific Mapper and Reducer over it. Finally, the output will be written on the HDFS directory.

- **HDFS**:

HDFS is a file system distributed over Hadoop clusters with several data nodes. It provides data services for various data operations.

Hadoop Streaming:

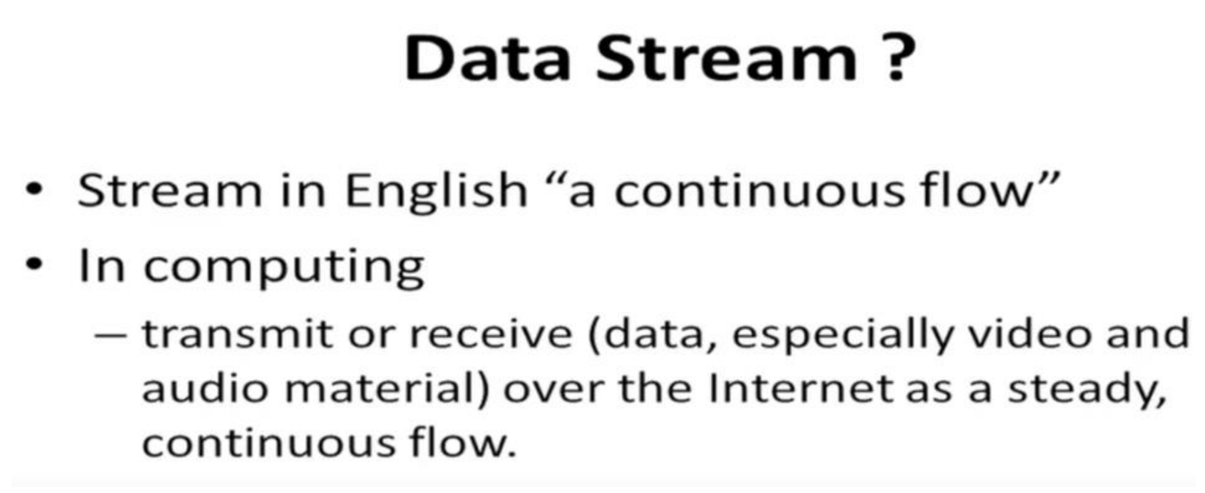
- ✓ A utility that allows one to run map/reduce jobs on the cluster with any executable; e.g. shell scripts, Perl, Python, etc
- ✓ Executable reads input from STDIN and writes output to STDOUT
- ✓ Maps input line data from STDIN to (key, value) pairs in STDOUT, in our case



ORCH:

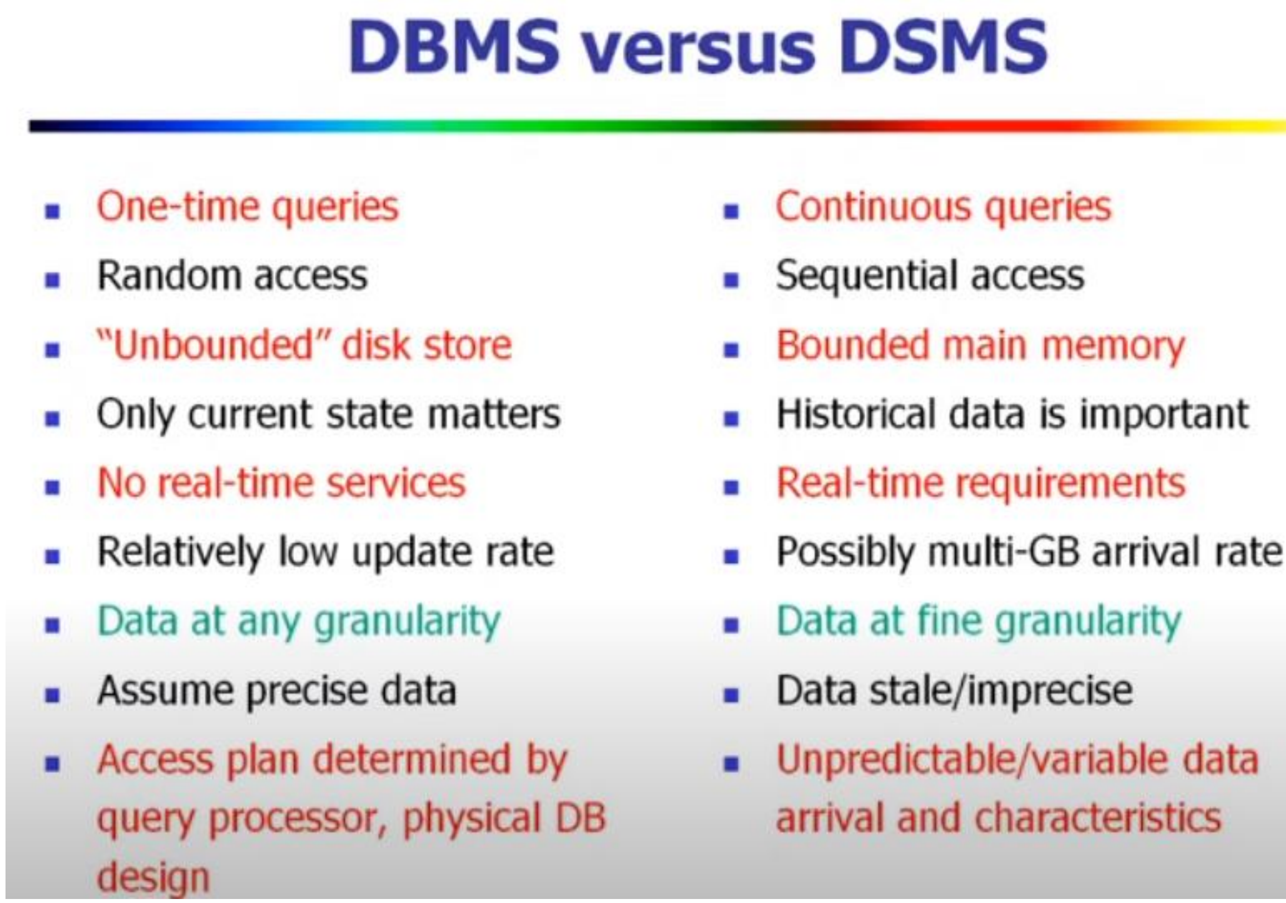
- ORCH is known as Oracle R Connector.
- It helps in accessing the Hadoop cluster with the help of R.
- It allows us to manipulate the data residing in the Hadoop Distributed File System.

MINING DATA STREAMS



Data Stream ?

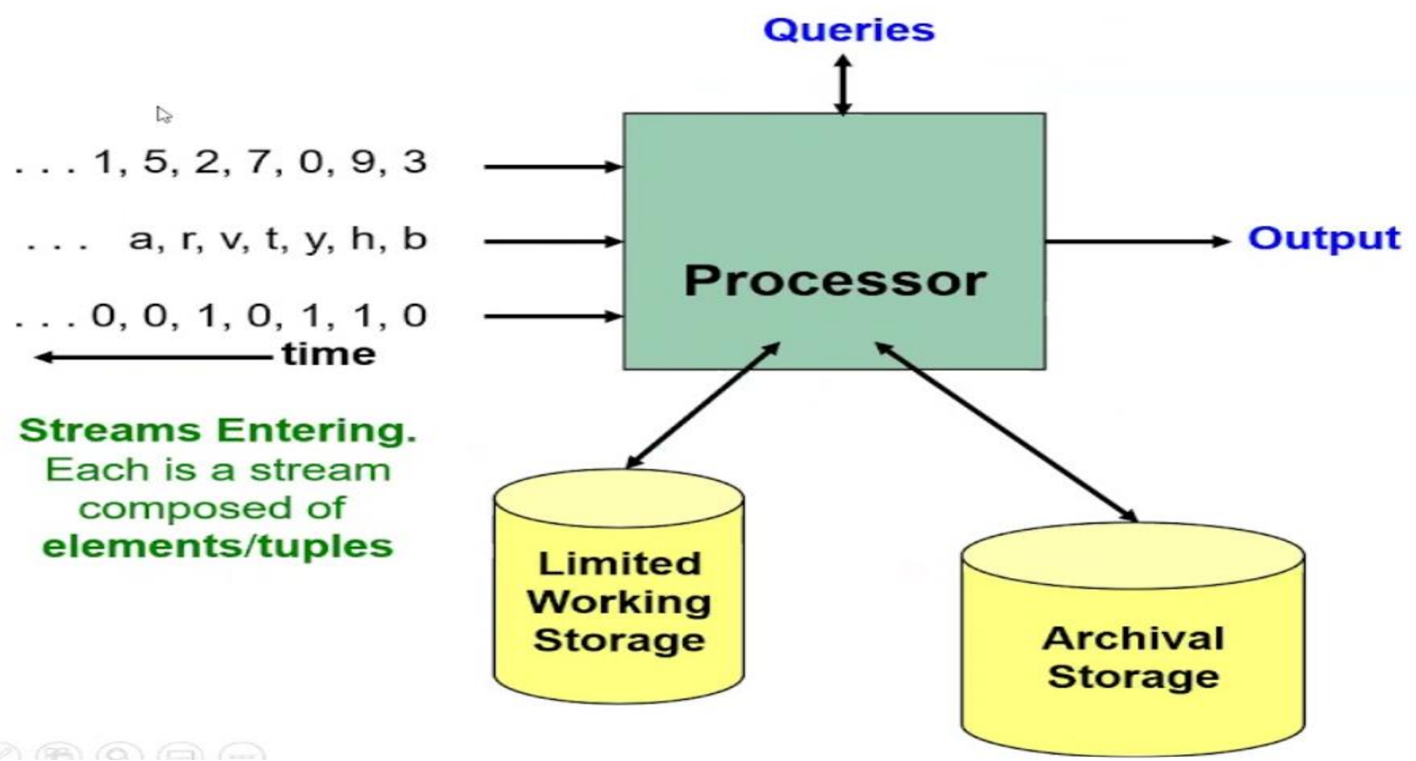
- Stream in English “a continuous flow”
- In computing
  - transmit or receive (data, especially video and audio material) over the Internet as a steady, continuous flow.



DBMS versus DSMS

One-time queries	Continuous queries
Random access	Sequential access
"Unbounded" disk store	Bounded main memory
Only current state matters	Historical data is important
No real-time services	Real-time requirements
Relatively low update rate	Possibly multi-GB arrival rate
Data at any granularity	Data at fine granularity
Assume precise data	Data stale/imprecise
Access plan determined by query processor, physical DB design	Unpredictable/variable data arrival and characteristics

General Stream Processing Model



Challenges in Working with Streaming Data

- A storage layer
- A processing layer

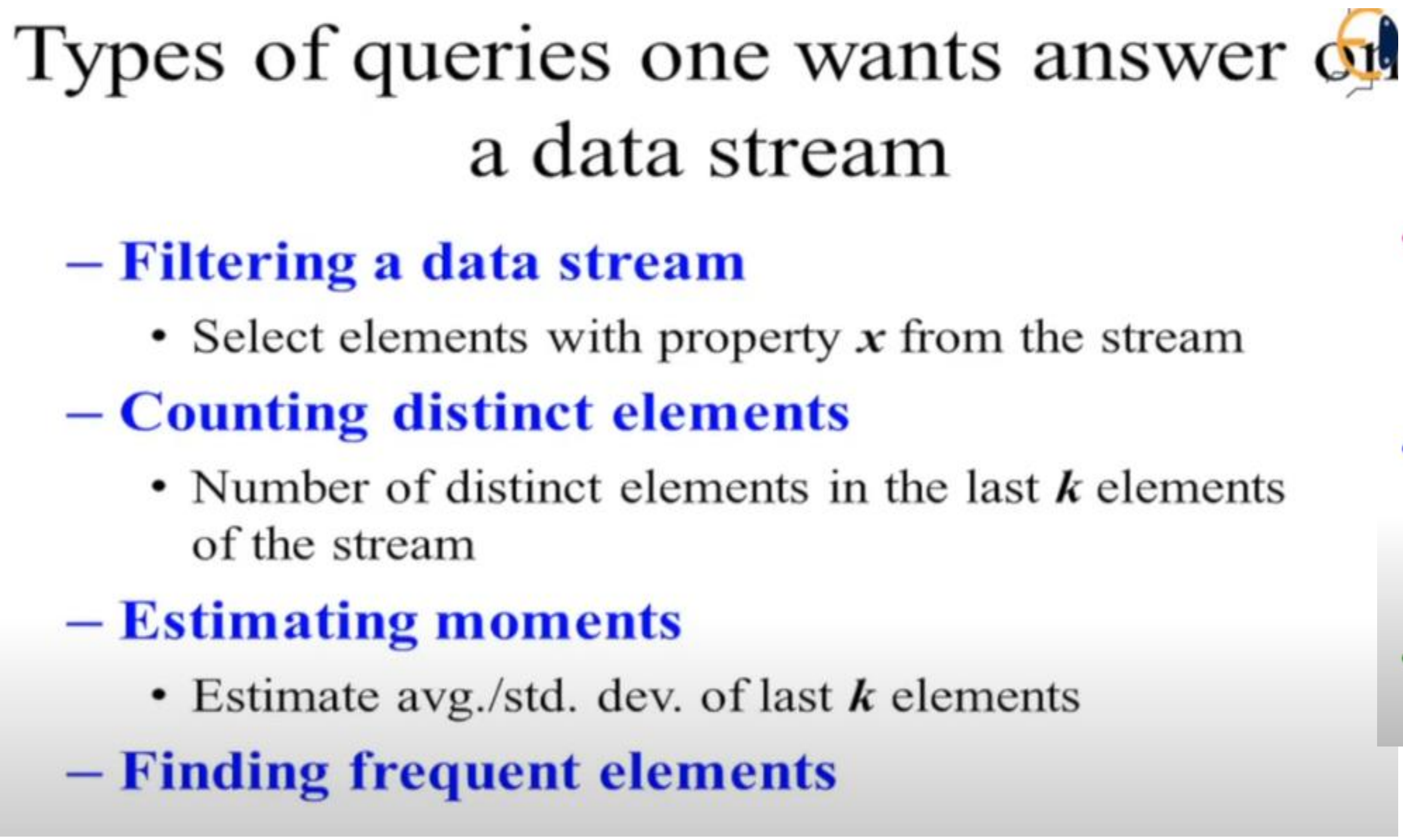
Needs plan for scalability, data durability, and fault tolerance in both the storage and processing layers.

Stream Processing

Parameter	Stream processing
Data scope	Queries or processing over data within a rolling time window, or on just the most recent data record.
Data size	Individual records or micro batches consisting of a few records.
Performance	Requires latency in the order of seconds or milliseconds.
Analyses	Simple response functions, aggregates, and rolling metrics.

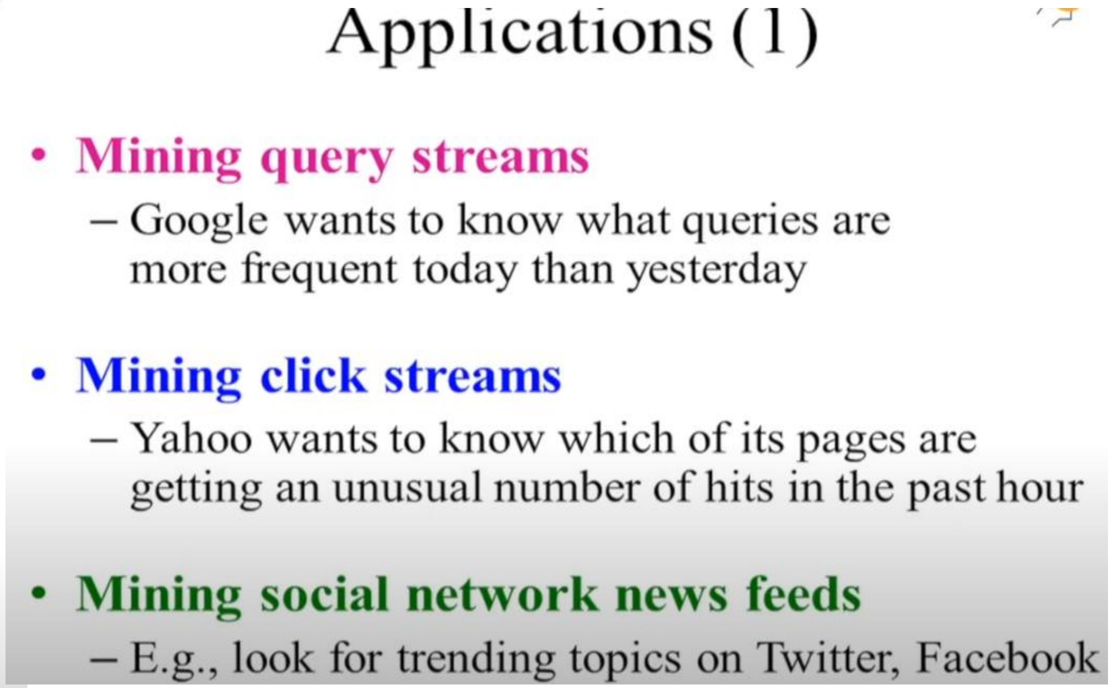
Streaming Data Examples

- Sensors in transportation vehicles, industrial equipment, and farm machinery.
- A financial institution
- A real-estate website
- A solar power company
- A media publisher
- An online gaming company



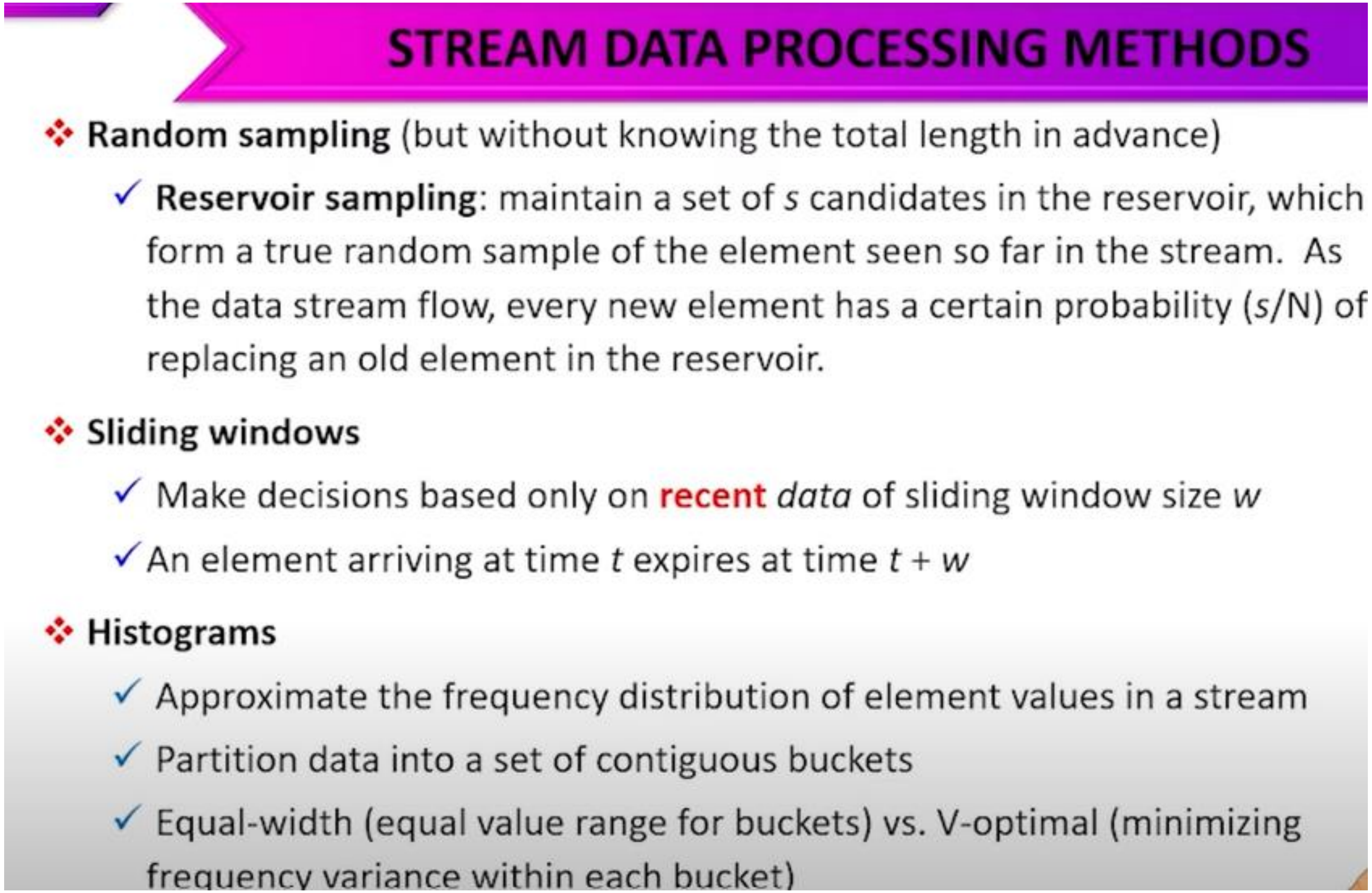
Types of queries one wants answer o a data stream

- **Filtering a data stream**
  - Select elements with property ***x*** from the stream
- **Counting distinct elements**
  - Number of distinct elements in the last ***k*** elements of the stream
- **Estimating moments**
  - Estimate avg./std. dev. of last ***k*** elements
- **Finding frequent elements**



Applications (1)

- **Mining query streams**
  - Google wants to know what queries are more frequent today than yesterday
- **Mining click streams**
  - Yahoo wants to know which of its pages are getting an unusual number of hits in the past hour
- **Mining social network news feeds**
  - E.g., look for trending topics on Twitter, Facebook

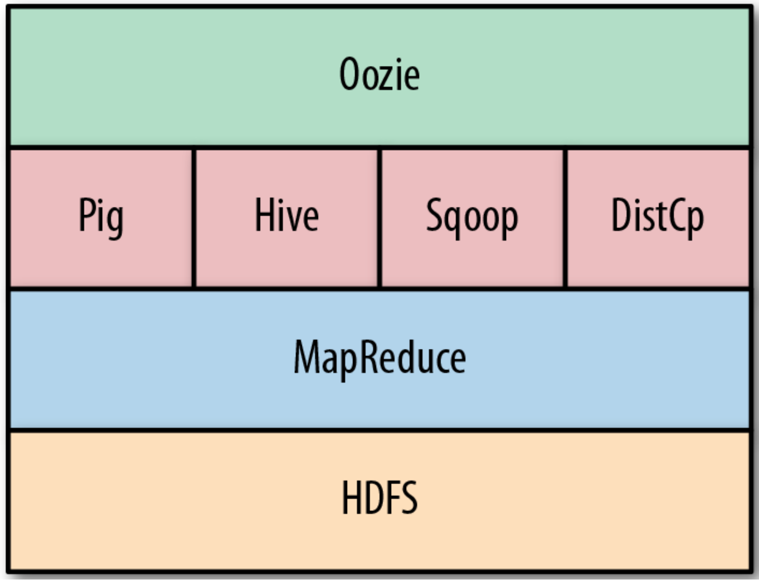


STREAM DATA PROCESSING METHODS

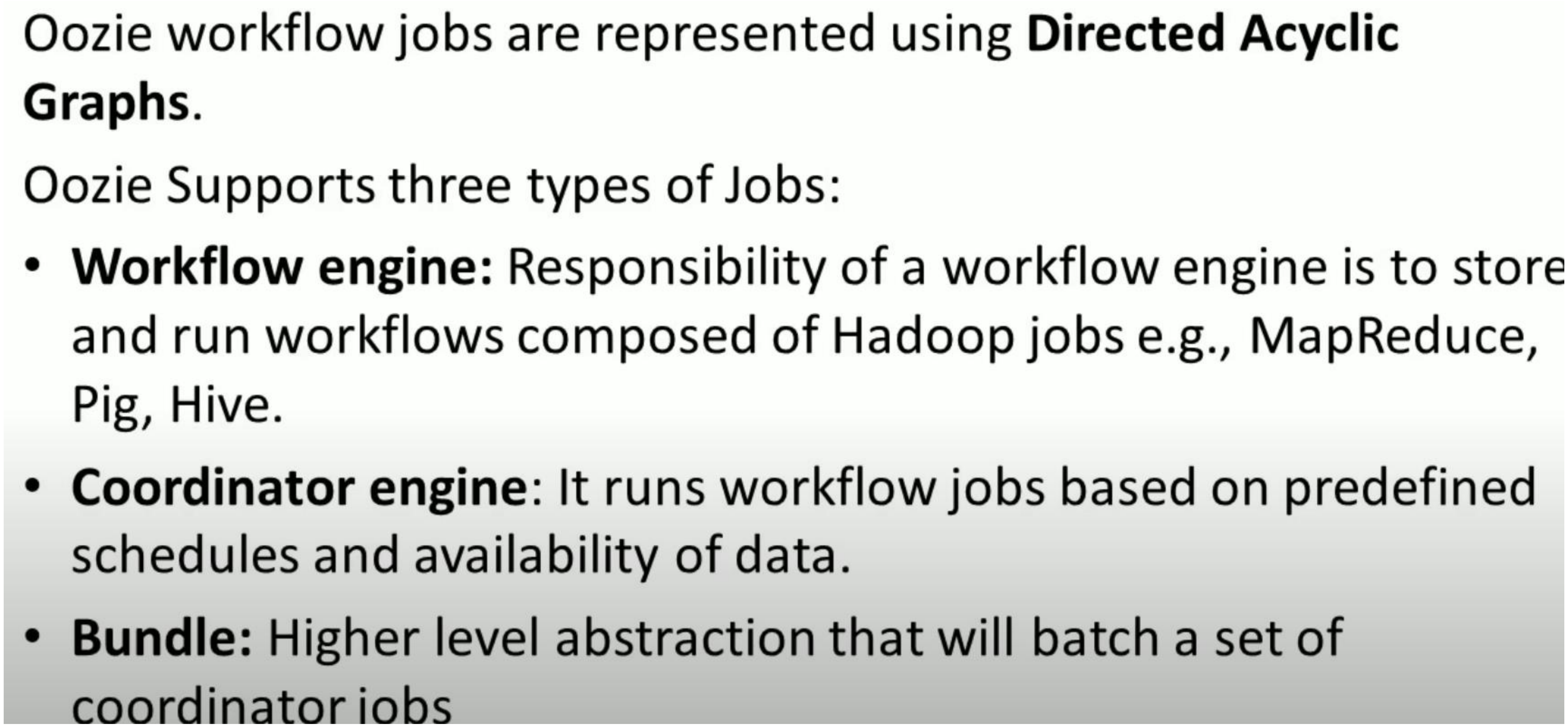
- **Random sampling** (but without knowing the total length in advance)
  - ✓ **Reservoir sampling**: maintain a set of *s* candidates in the reservoir, which form a true random sample of the element seen so far in the stream. As the data stream flow, every new element has a certain probability (*s*/N) of replacing an old element in the reservoir.
- **Sliding windows**
  - ✓ Make decisions based only on **recent** *data* of sliding window size *w*
  - ✓ An element arriving at time *t* expires at time *t* + *w*
- **Histograms**
  - ✓ Approximate the frequency distribution of element values in a stream
  - ✓ Partition data into a set of contiguous buckets
  - ✓ Equal-width (equal value range for buckets) vs. V-optimal (minimizing frequency variance within each bucket)

Apache OOZIE

- Apache Oozie is a workflow director designed to run and mana
  multiple related Apache Jobs.
- For instance, complete data input and analysis may require several related Hadoop jobs to be run as a workflow in which t output of one job may input to the successive job.
- Oozie is designed to construct and manage these workflows.
- Ozzie is not a substitute for YARN scheduler.
- YARN manages the resources of individual jobs and Oozie provides a mechanism to connect and control Hadoop jobs on



Apache OOZIE

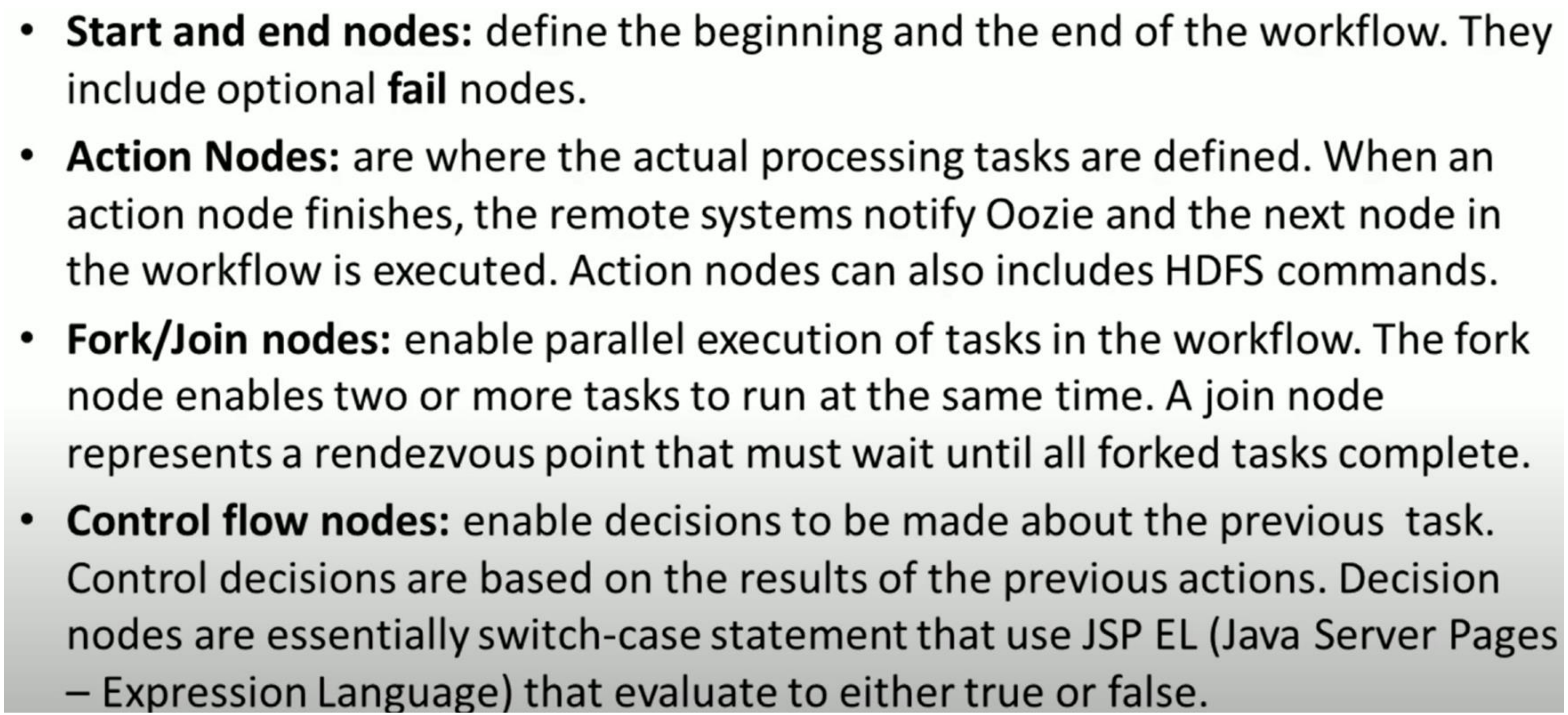


Oozie workflow jobs are represented using **Directed Acyclic Graphs**.

Oozie Supports three types of Jobs:

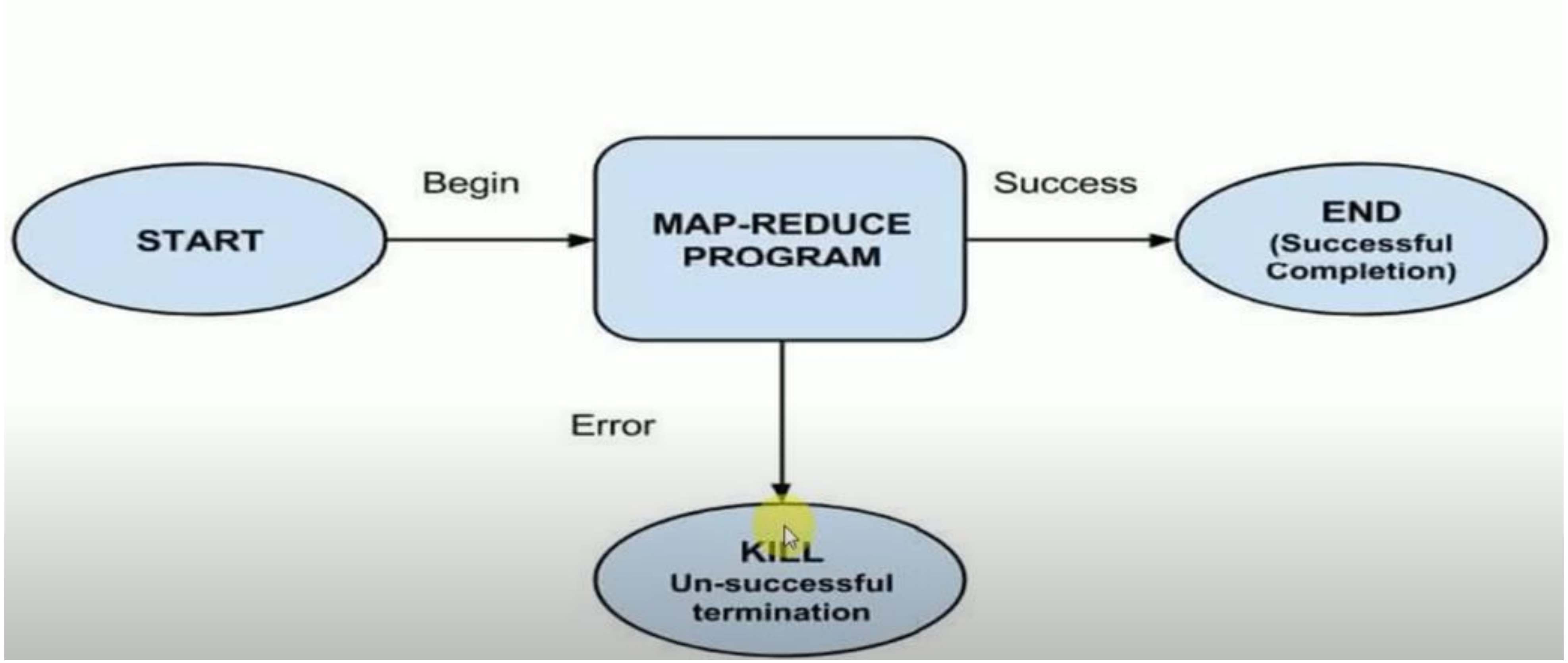
- **Workflow engine:** Responsibility of a workflow engine is to store and run workflows composed of Hadoop jobs e.g., MapReduce, Pig, Hive.
- **Coordinator engine**: It runs workflow jobs based on predefined schedules and availability of data.
- **Bundle:** Higher level abstraction that will batch a set of coordinator jobs

Apache OOZIE: Types of Nodes

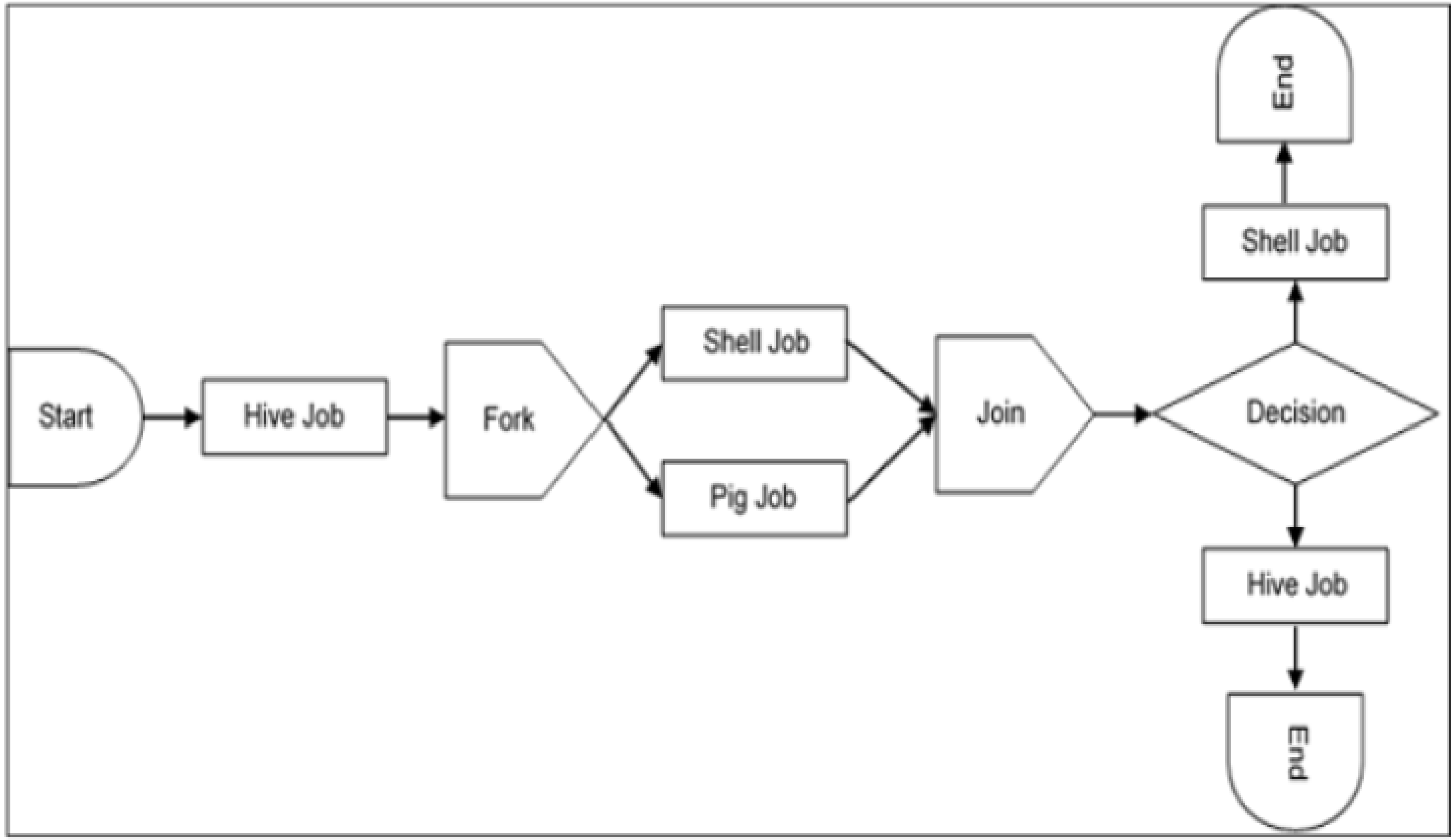


- **Start and end nodes:** define the beginning and the end of the workflow. They include optional **fail** nodes.
- **Action Nodes:** are where the actual processing tasks are defined. When an action node finishes, the remote systems notify Oozie and the next node in the workflow is executed. Action nodes can also includes HDFS commands.
- **Fork/Join nodes:** enable parallel execution of tasks in the workflow. The fork node enables two or more tasks to run at the same time. A join node represents a rendezvous point that must wait until all forked tasks complete.
- **Control flow nodes:** enable decisions to be made about the previous task. Control decisions are based on the results of the previous actions. Decision nodes are essentially switch-case statement that use JSP EL (Java Server Pages – Expression Language) that evaluate to either true or false.

Apache OOZIE: workflow



Apache OOZIE: Complex workflow



THANK YOU.

* ALL THE BEST *