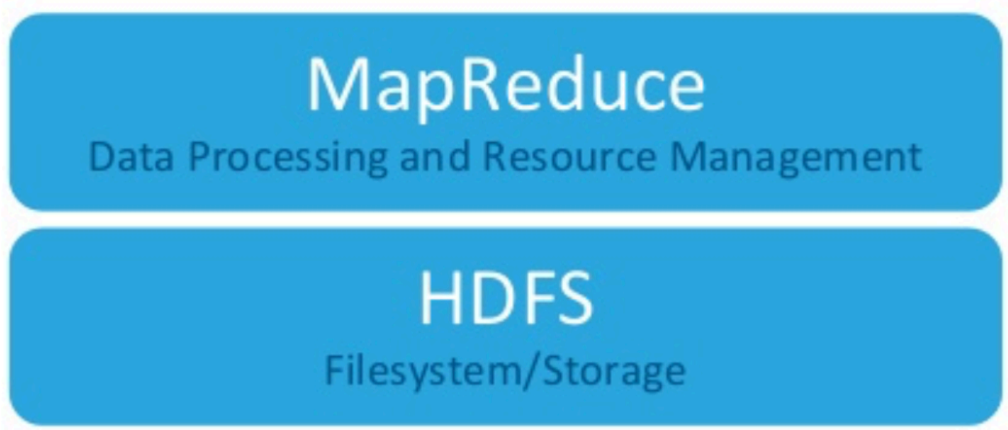
Apache Spark

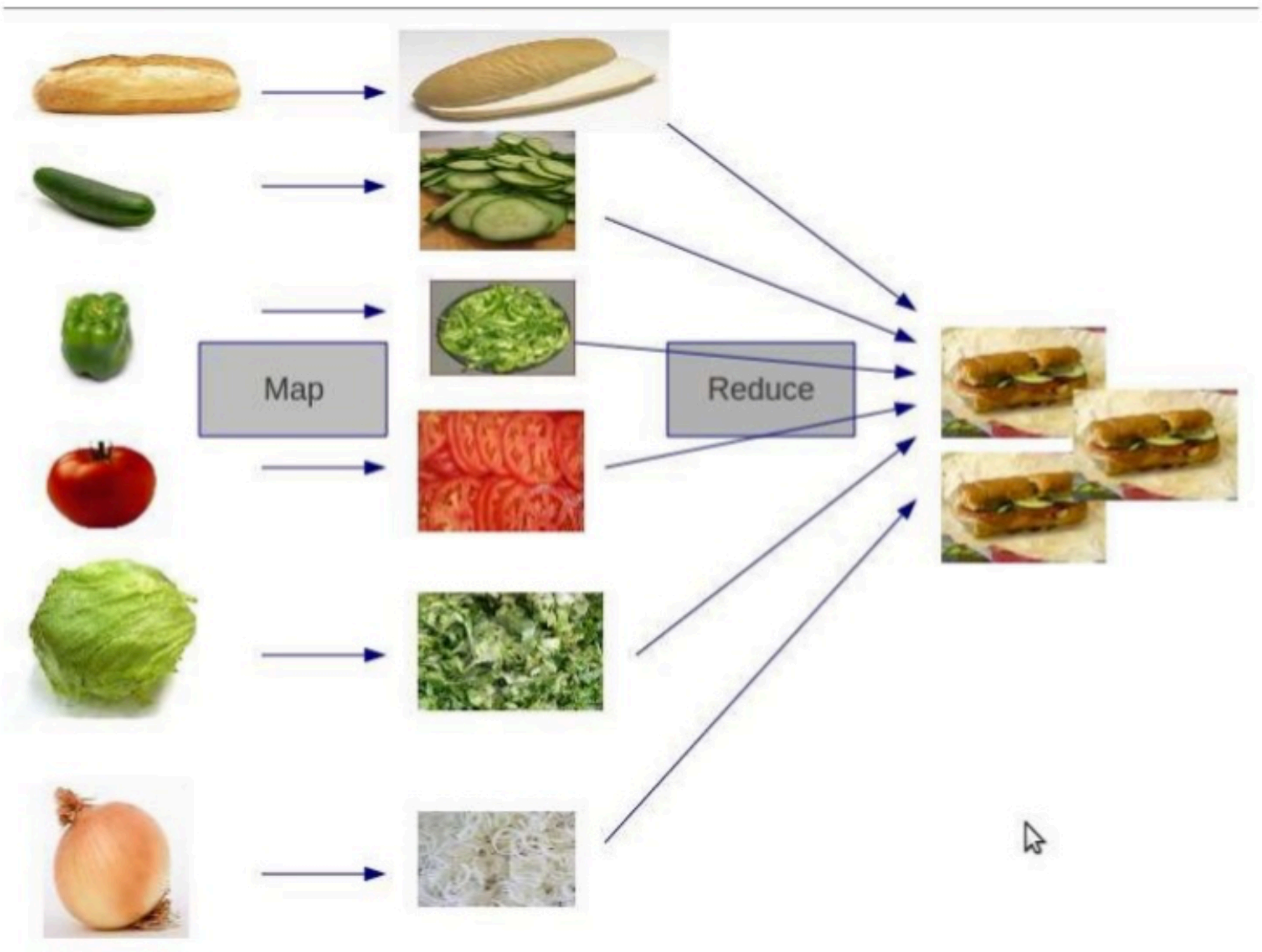
Prajakta Kalmegh
PhD Student, Duke University

MapReduce

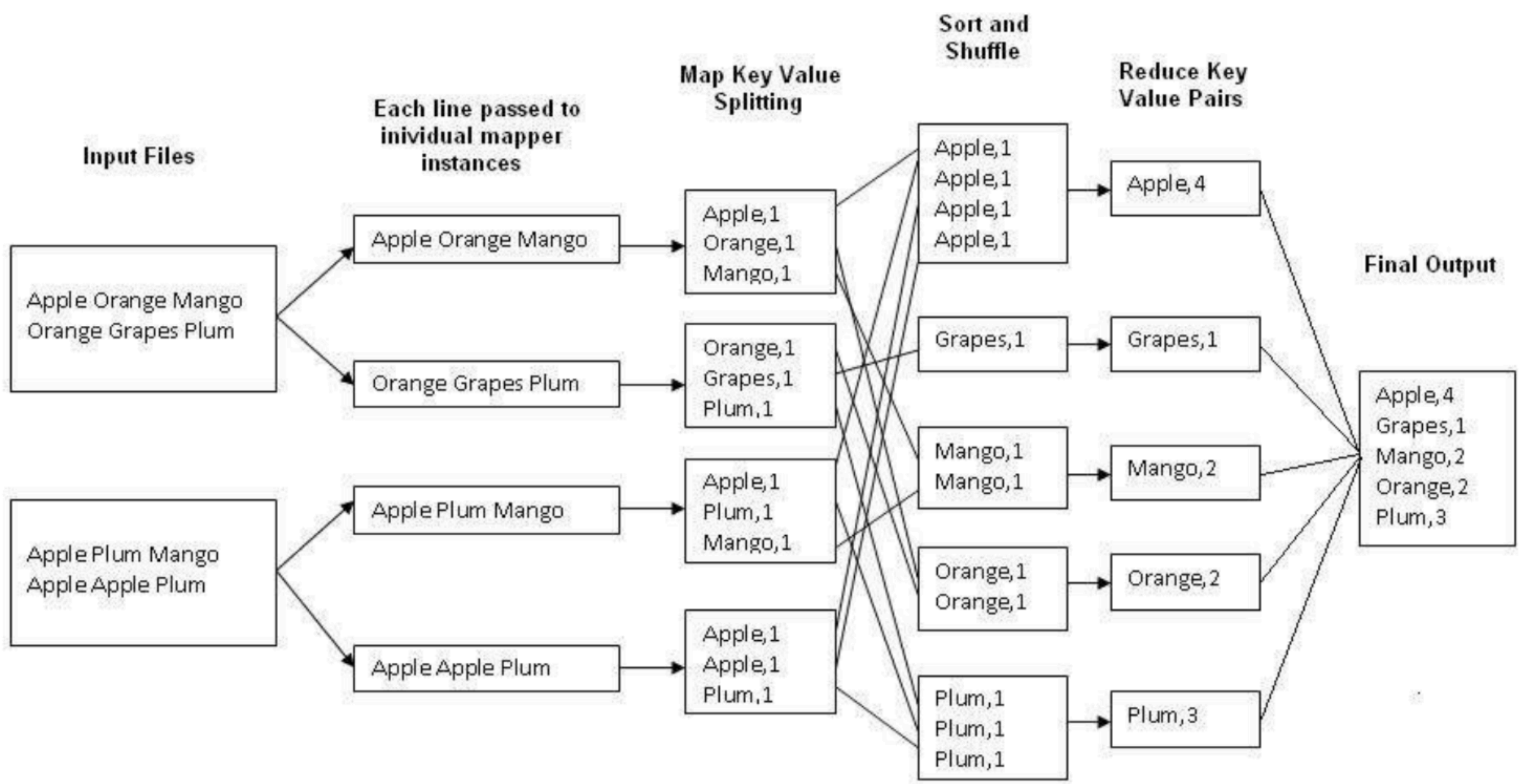
Hadoop’s Original Architecture



cloudera

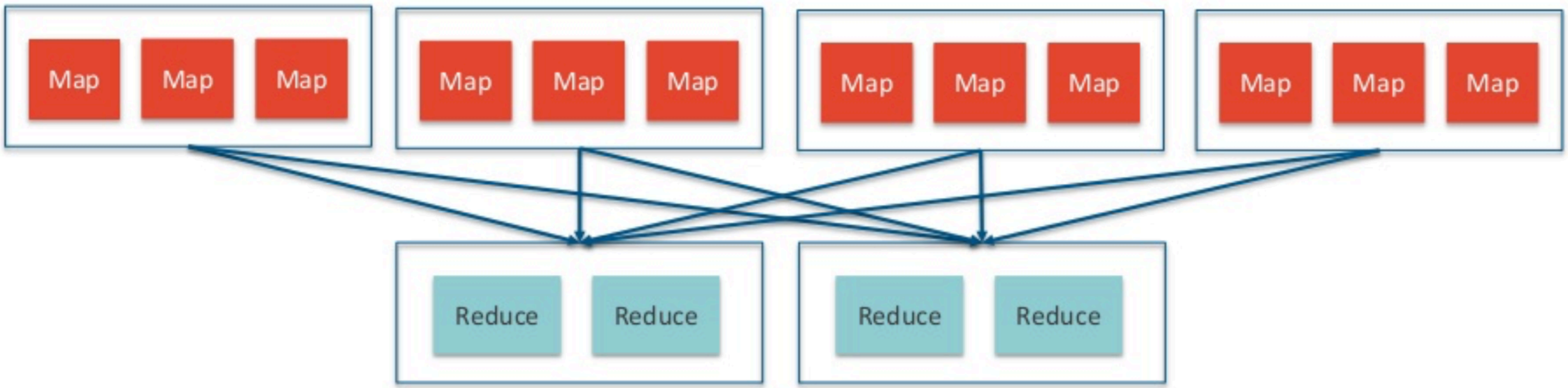

Map
Reduce

Word Count



Input Files
Each line passed to inividual mapper instances
Map Key Value Splitting
Sort and Shuffle
Reduce Key Value Pairs
Final Output
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple,1
Orange,1
Mango,1
Orange,1
Grapes,1
Plum,1
Apple,1
Plum,1
Mango,1
Apple,1
Apple,1
Plum,1
Apple,1
Apple,1
Apple,1
Apple,1
Grapes,1
Mango,1
Mango,1
Orange,1
Orange,1
Plum,1
Plum,1
Plum,1
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3

The MapReduce Breakthrough



Key advances in MapReduce:

- Data locality: Automatic split computation and appropriate launch of mappers
- Fault-tolerance: Write-out of intermediate results and restartable mappers provides ability to run on commodity hardware
- Linear scalability: Combination of locality + programming model forces developers to write generally scalable solutions

cloudera

Compute Engine

Why a New ~~Programming Model~~?

MapReduce greatly simplified big data analysis

But as soon as it got popular, users wanted more:

» More complex, multi-stage **iterative** applications (graph algorithms, machine learning)

» More **interactive** ad-hoc queries

» More **real-time** online processing

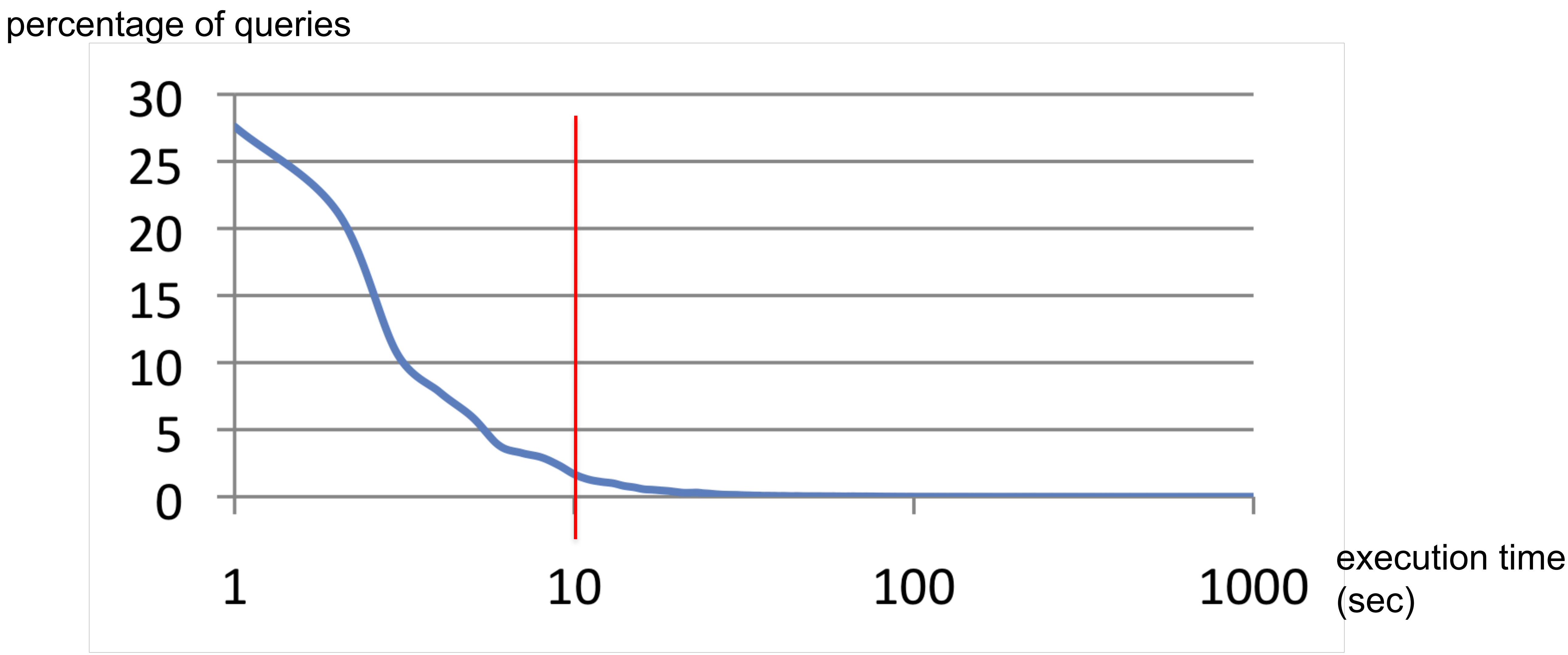
All three of these apps require **fast data sharing** across parallel jobs

NOTE: What were the workarounds in MR world?
Ysmart [1], Stubby[2], PTF[3], Haloop [4], Twister [5]

Borrowed slide

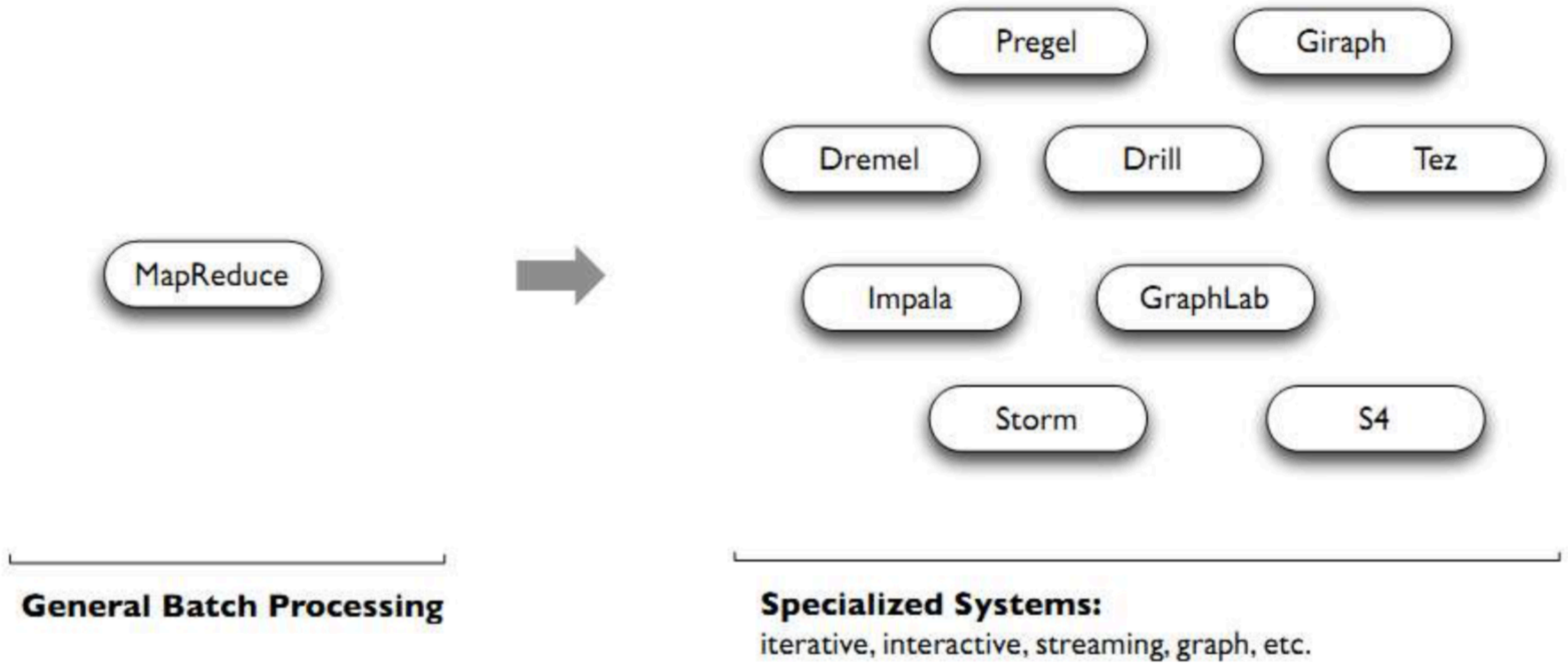
Interactive speed

Monthly query workload
of one 3000-node Dremel instance



Most queries complete under 10 sec

Therefore, people built specialized systems as workarounds…



Originally developed by UC Berkeley starting in 2009 Moved to an Apache project in 2013

Apache Spark: A Better MapReduce

Distributed in-memory large scale data processing engine!

Easy, Expressive API

- Rich API (Java, Scala, and Python)
- Interactive shell
- 2-5x less code needed than MR

Unlike the various specialized systems, Spark's goal was to generalize MapReduce to support new apps within same engine

*and powerful optimizations

- General execution graphs
- In-memory storage
- Order-of-magnitude improvement over MR

cloudera

What is Spark Used For?

- Stream processing
  - log files
  - sensor data
  - financial transactions
- Machine learning
  - store data in memory and rapidly run repeated queries
- Interactive analytics
  - business analysts and data scientists increasingly want to explore their data by asking a question, viewing the result, and then either altering the initial question slightly or drilling deeper into results. This interactive query process requires systems such as Spark that are able to respond and adapt quickly
- Data integration
  - Extract, transform, and load (ETL) processes

Checkpoint!

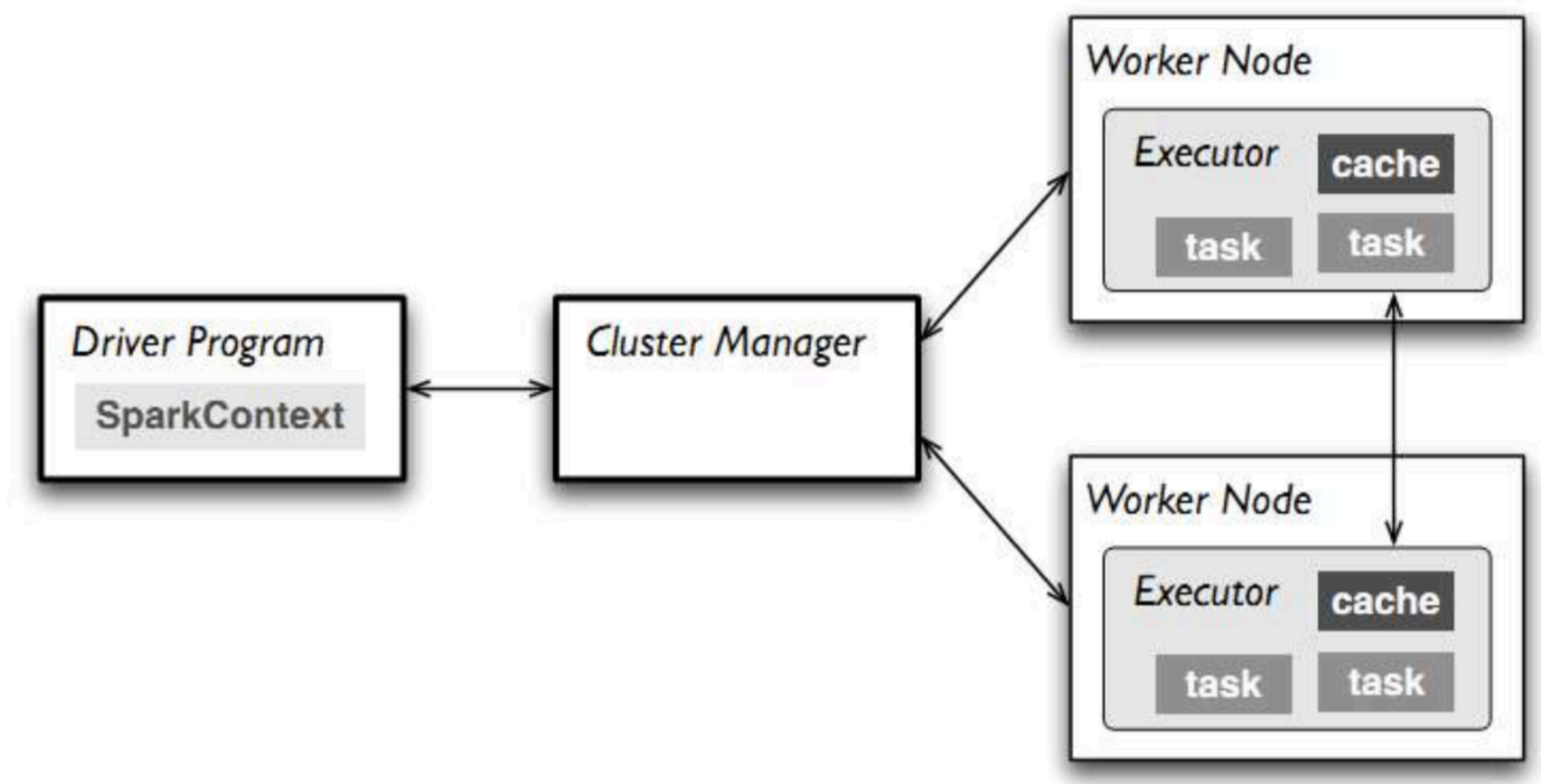
- Introduction to Spark

Spark

- Runs
  - Locally
  - distributed a cross a cluster
    - Requires a cluster manager
      - Yarn
      - Spark Stand alone
      - Mesos
- spark
  - Interactive shell
    - Data exploration
    - Ad-hoc analysis
  - Submit an application
- is offten used alongside Hadoop’s data storage module, HDFS
- can also integrate equally well with other popular data storage subsystems such as HBase, Cassandra,…

Spark execution model

- Application
- Driver
- Executer
- Job
- Stage



Spark execution model

- At runtime, a Spark application maps to a single ***driver*** process and a set of ***executor*** processes distributed across the hosts in a cluster
- The driver process manages the job flow and schedules tasks and is available the entire time the application is running.
  - Typically, this driver process is the same as the client process used to initiate the job
  - In interactive mode, the shell itself is the driver process
- The executors are responsible for executing work, in the form of *tasks*, as well as for storing any data that you cache.
- Invoking an action inside a Spark application triggers the launch of a ***job*** to fulfill it
- Spark examines the dataset on which that action depends and formulates an execution plan.
- The execution plan assembles the dataset transformations into stages. A ***stage*** is a collection of tasks that run the same code, each on a different subset of the data.

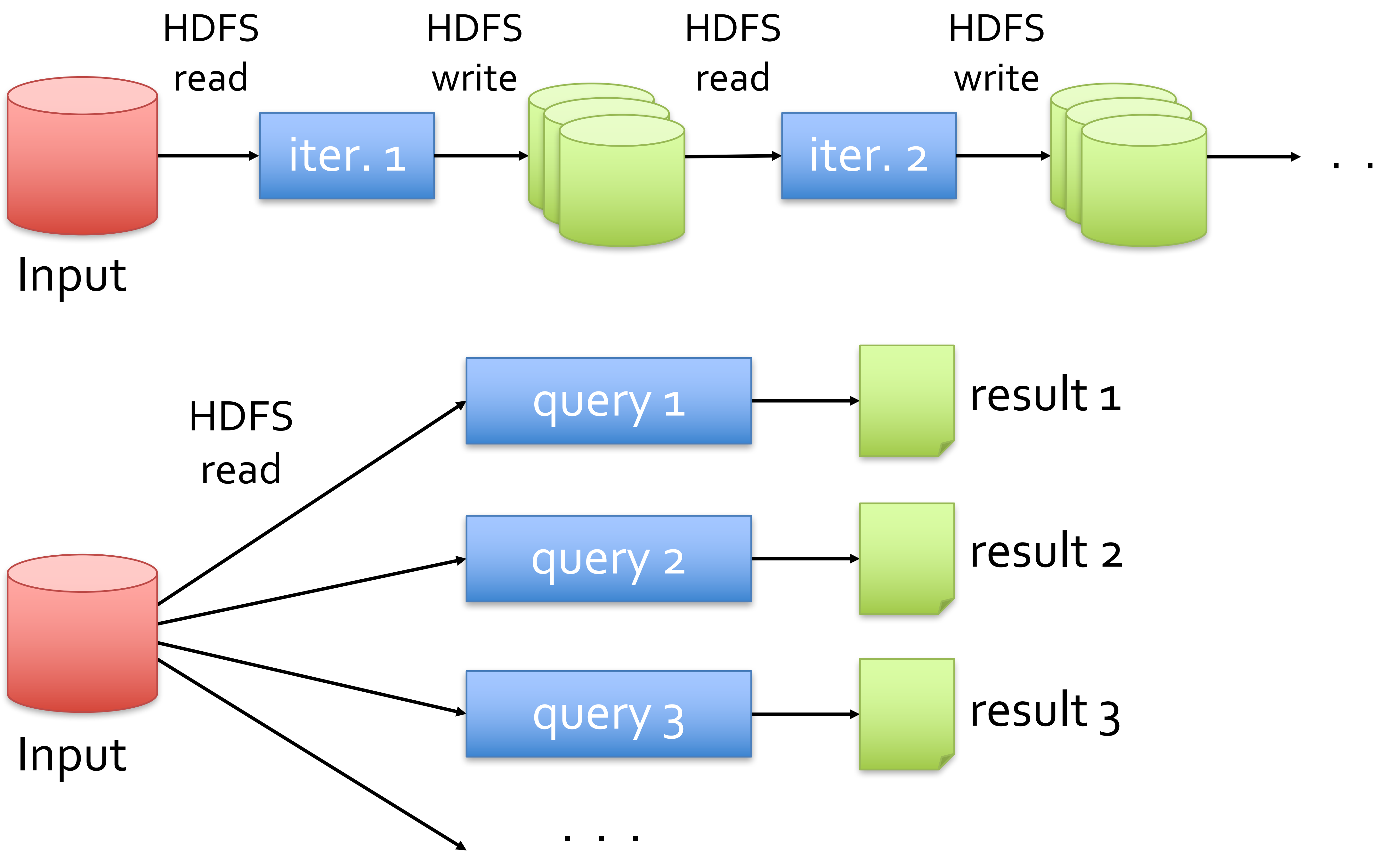
Spark APIs

- APIs for
    - Java
    - Python
    - Scala
    - R
- Spark itself is written in Scala
- Percent of Spark programmers who use each language , In 2016
    - 88% Scala, 44% Java, 22% Python
- I think if it were done today, we would see the rank as Scala, Python, and Java

Checkpoint!

- Introduction to Spark
- **Spark Execution Model**

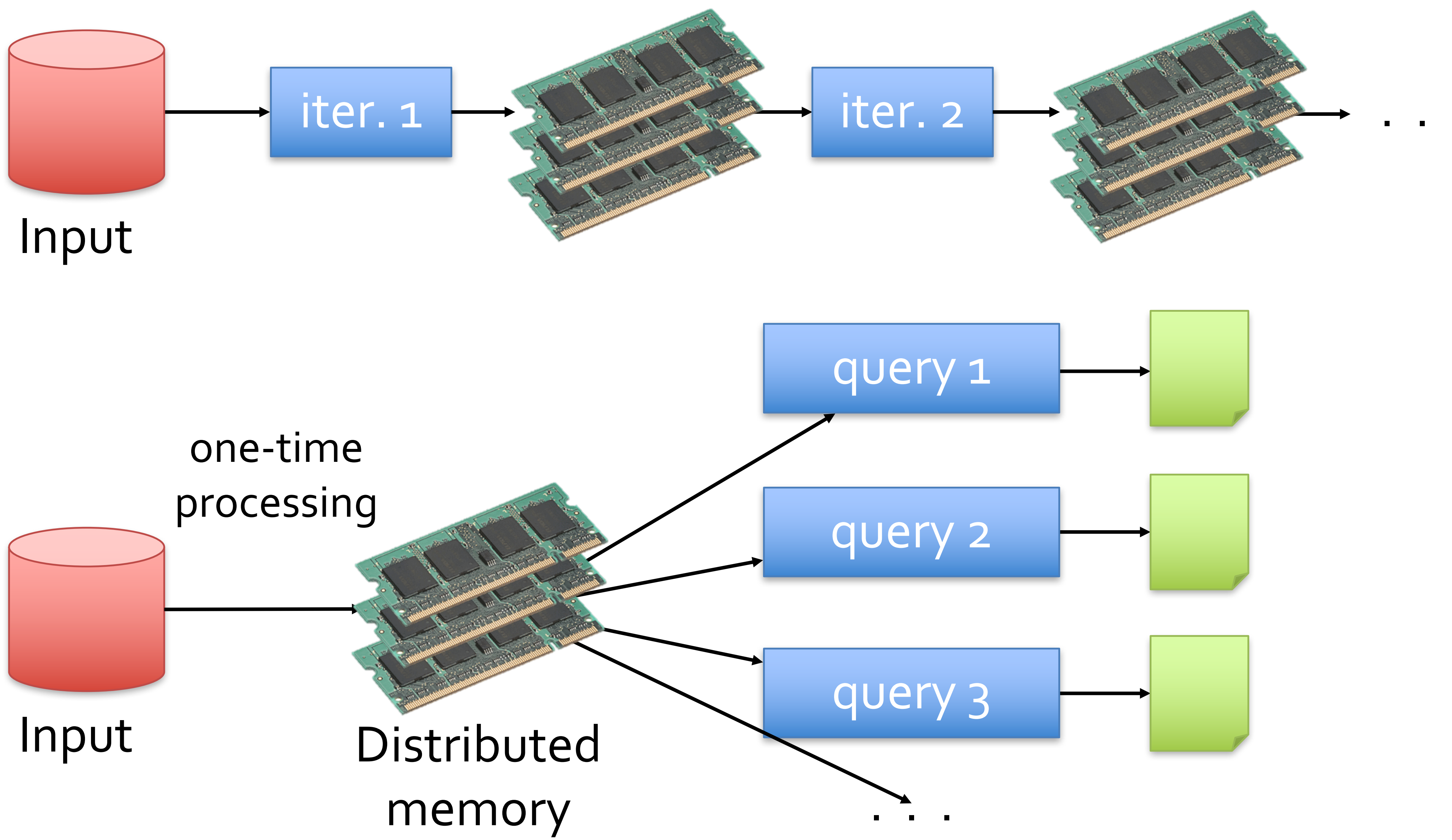
Recall: Data Sharing in MapReduce



Slow due to replication, serialization, and disk IO

Borrowed slide

Data Sharing in Spark



10-100× faster than network and disk

Borrowed slide

Basic Programming Model

- Spark's basic data model is called a Resilient Distributed Dataset (RDD)
- It is designed to support in-memory data storage, distributed across a cluster
  - fault-tolerant
    - tracking the lineage of transformations applied to data
  - Efficient
    - parallelization of processing across multiple nodes in the cluster
    - minimization of data replication between those nodes.

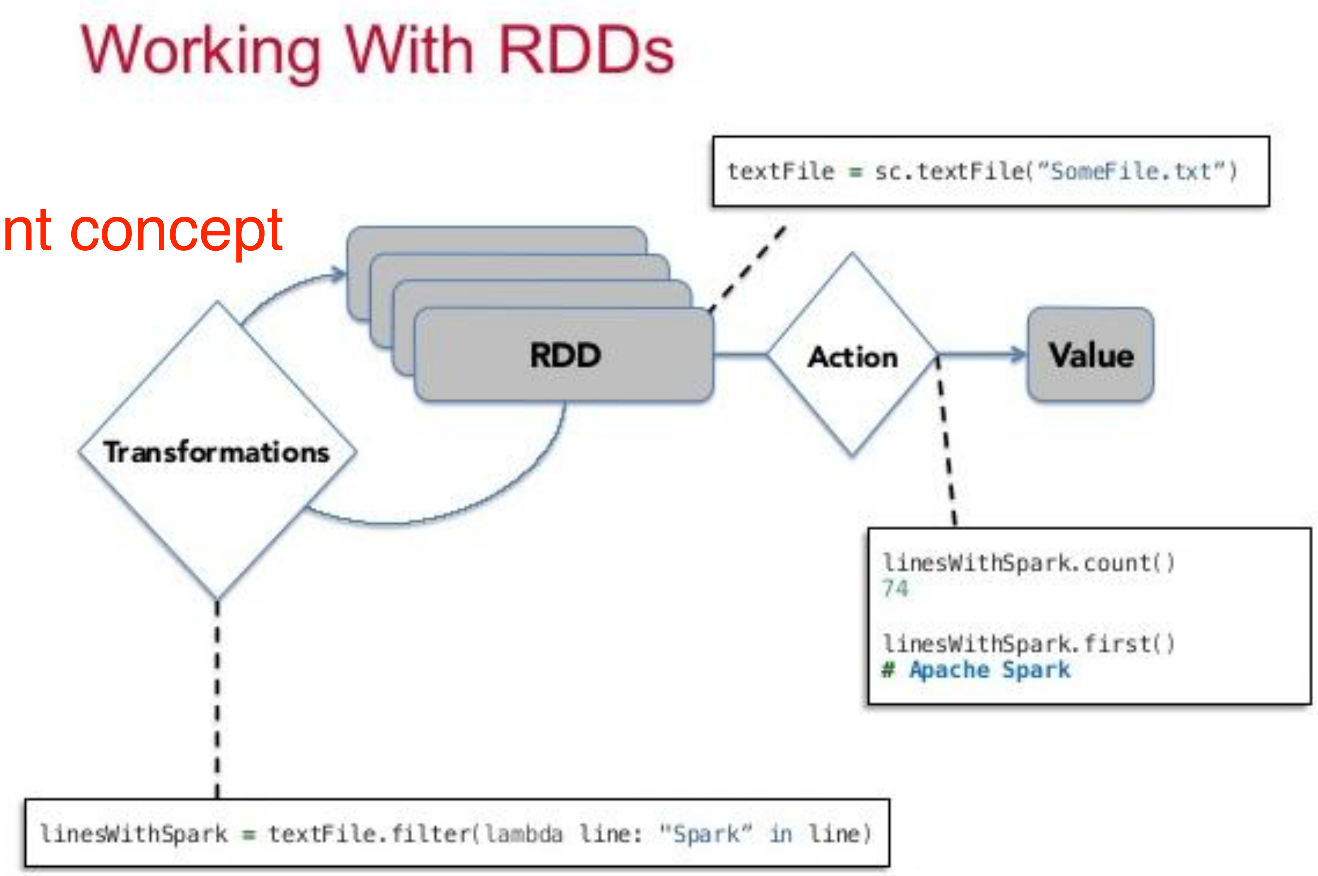
RDDs

- Two basic types of operations on RDDs
  - Transformations
    - Transform an RDD into another RDD, such as mapping, filtering, and more
  - Actions:
    - Process an RDD into a result , such as count, collect, save , …
- The original RDD remains unchanged throughout
- The chain of transformations from RDD1 to RDDn are logged
  - and can be repeated in the event of data loss or the failure of a cluster node

Transformations

- Transformations are lazily processed, only upon an action
- Transformations create a new RDD from an existing one
- Transformations might trigger an RDD repartitioning, called a shuffle
- Intermediate results can be manually cached in memory/on disk
- Spill to disk can be handled automatically

Note: Lazy Evaluation: A very important concept



*http://www.tothenew.com/blog/spark-1o3-spark-internals/

Representing RDDs

- Five components :

Operation	Meaning	
partitions()	Return a list of Partition objects	
preferredLocations(*p*)	List nodes where partition *p* can be accessed faster due to data locality	
dependencies()	Return a list of dependencies	
iterator(*p*, *parentIters*)	Compute the elements of partition *p* given iterators for its parent partitions	Computation function
partitioner()	Return metadata specifying whether the RDD is hash/range partitioned	helps in partitions based optimization

Table 3: Interface used to represent RDDs in Spark.

Representing RDDs (Dependencies)

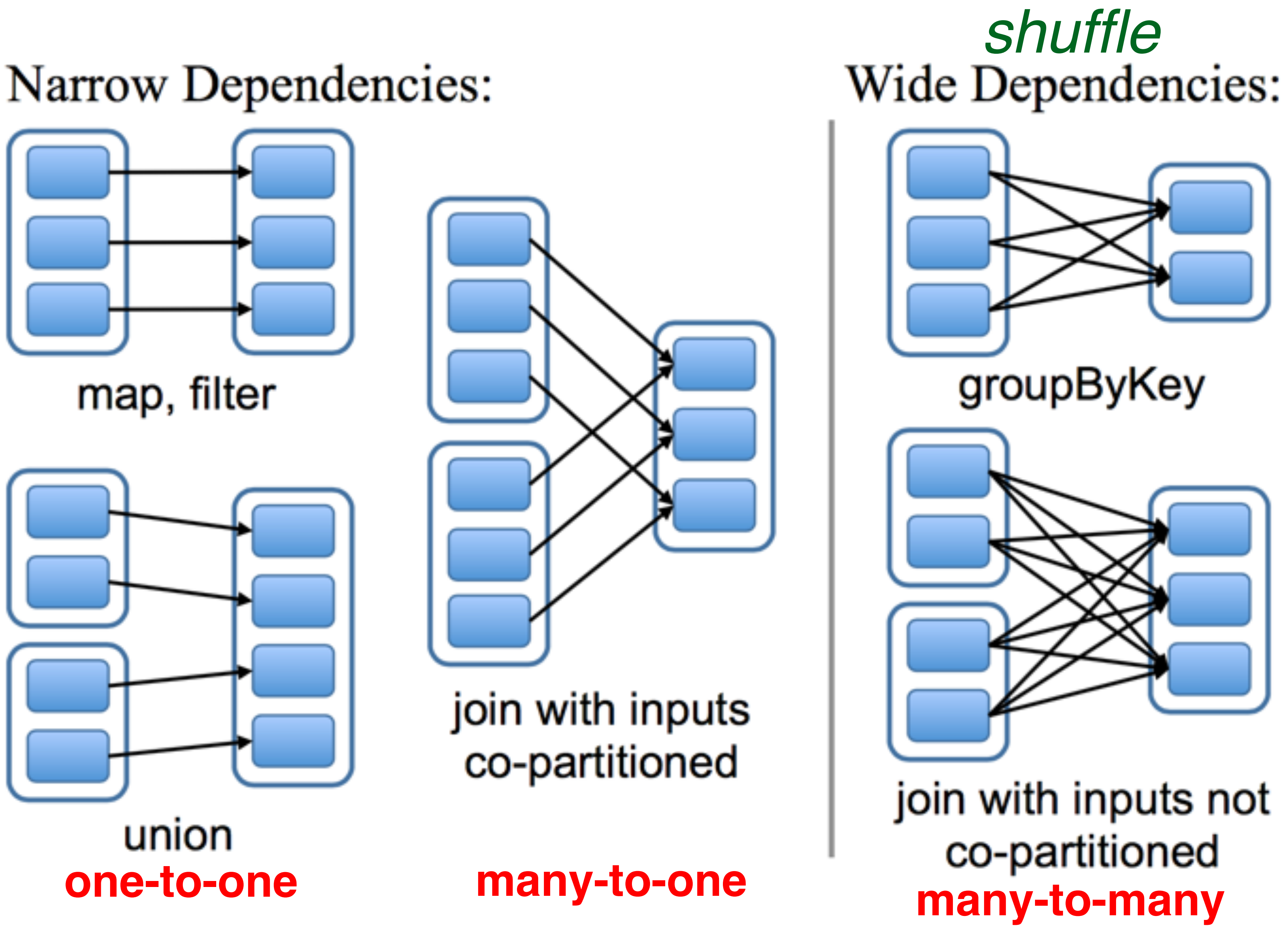


Figure 4: Examples of narrow and wide dependencies. Each box is an RDD, with partitions shown as shaded rectangles.

Representing RDDs (An example)

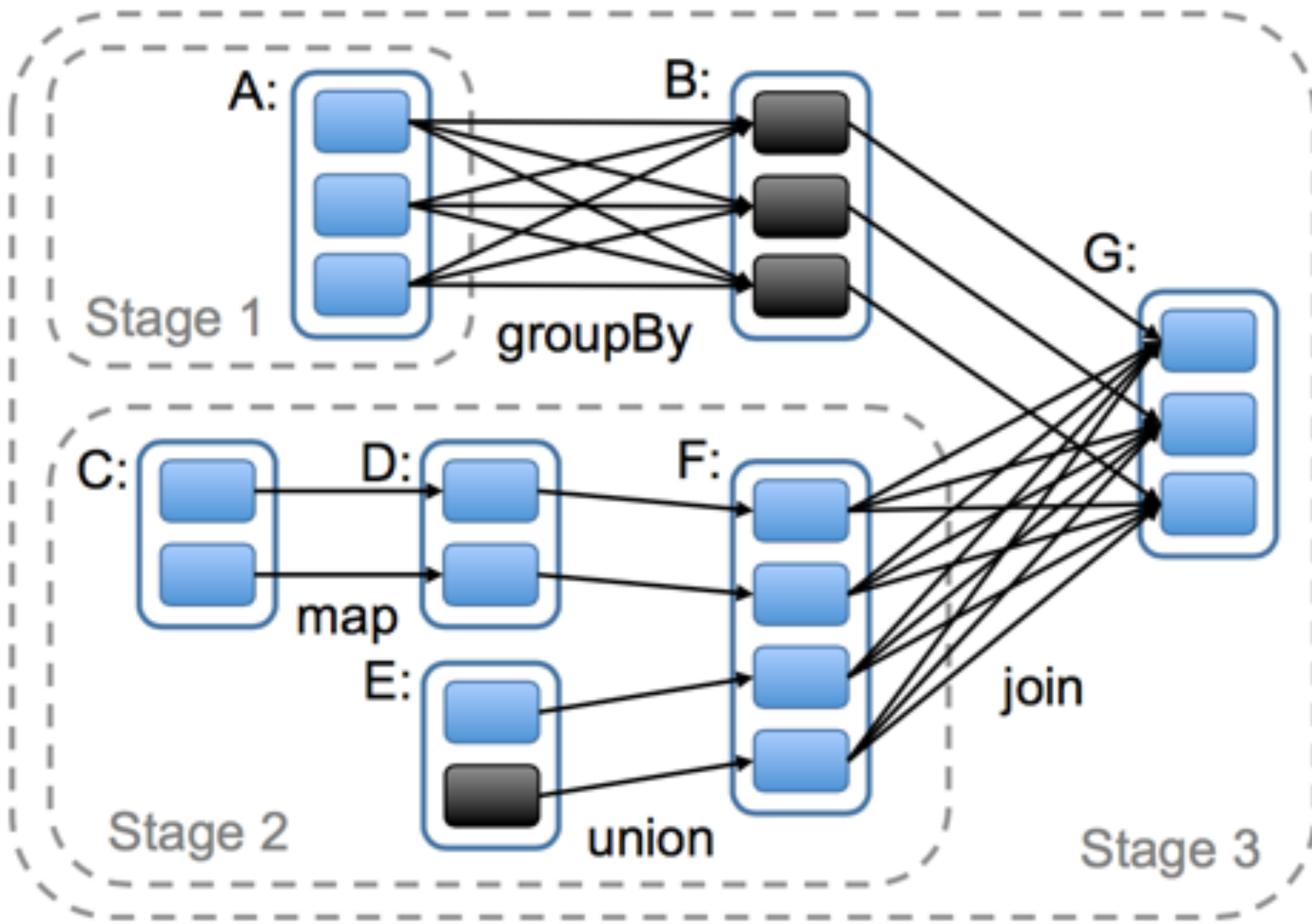


Figure 5: Example of how Spark computes job stages. Boxes with solid outlines are RDDs. Partitions are shaded rectangles, in black if they are already in memory. To run an action on RDD G, we build build stages at wide dependencies and pipeline narrow transformations inside each stage. In this case, stage 1's output RDD is already in RAM, so we run stage 2 and then 3.

Creating RDDs

#Turn a Scala collection into an RDD

- sc.Parallelize(List(1, 2, 3, 4))

#Load Text File from FS or HDFS

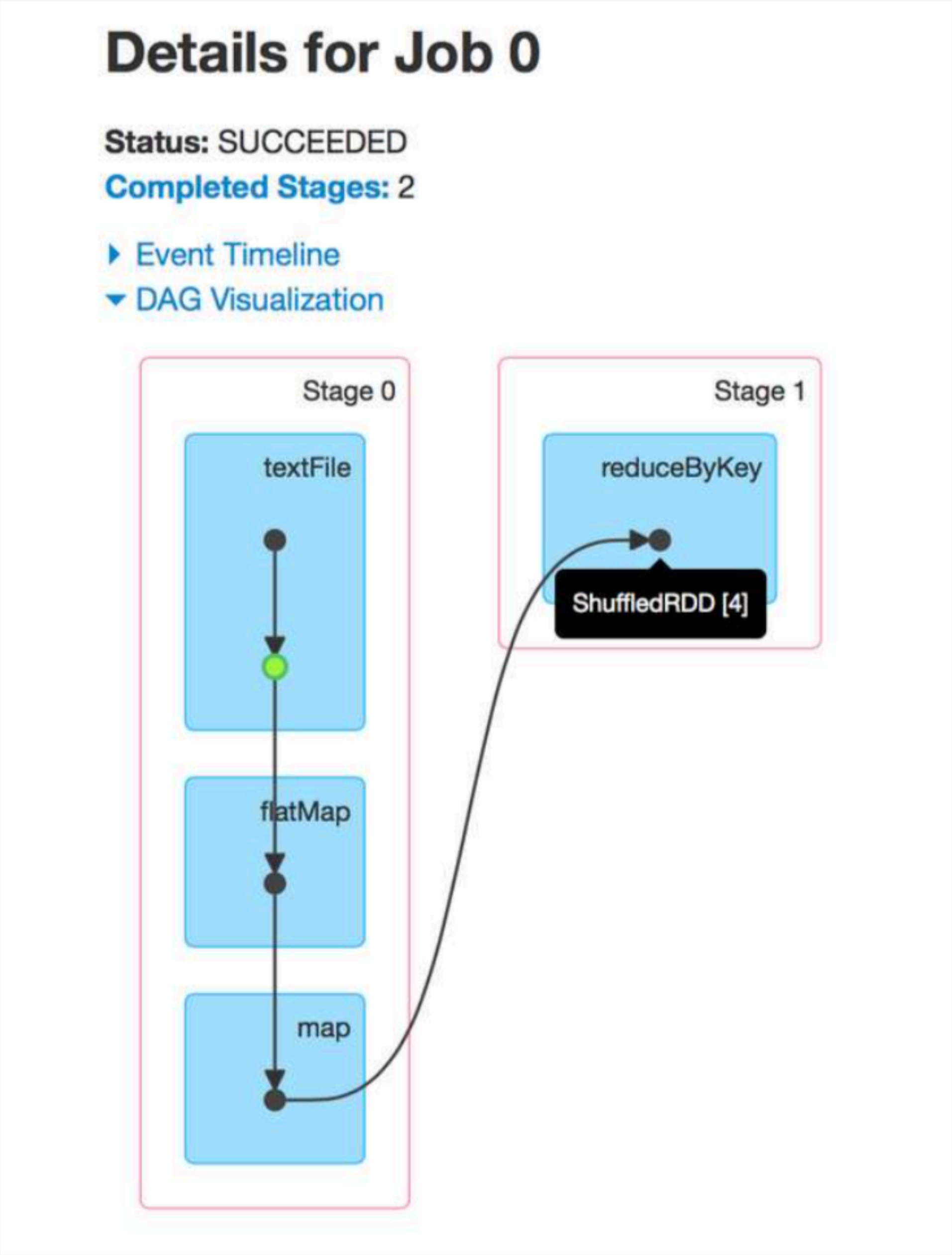
- sc.textFile(“file.txt”)
- sc.textFile(“directory/*.txt”)
- sc.textFile(“hdfs/namenode:9000/path/file.txt”)

#Use Existing Hadoop InputFormat

- Sc.hadoopFile(keyClass, valueClass , inputFmt , conf)

DAG

- Stands for Directed Acyclic Graph
- For every spark job a DAG of tasks is created

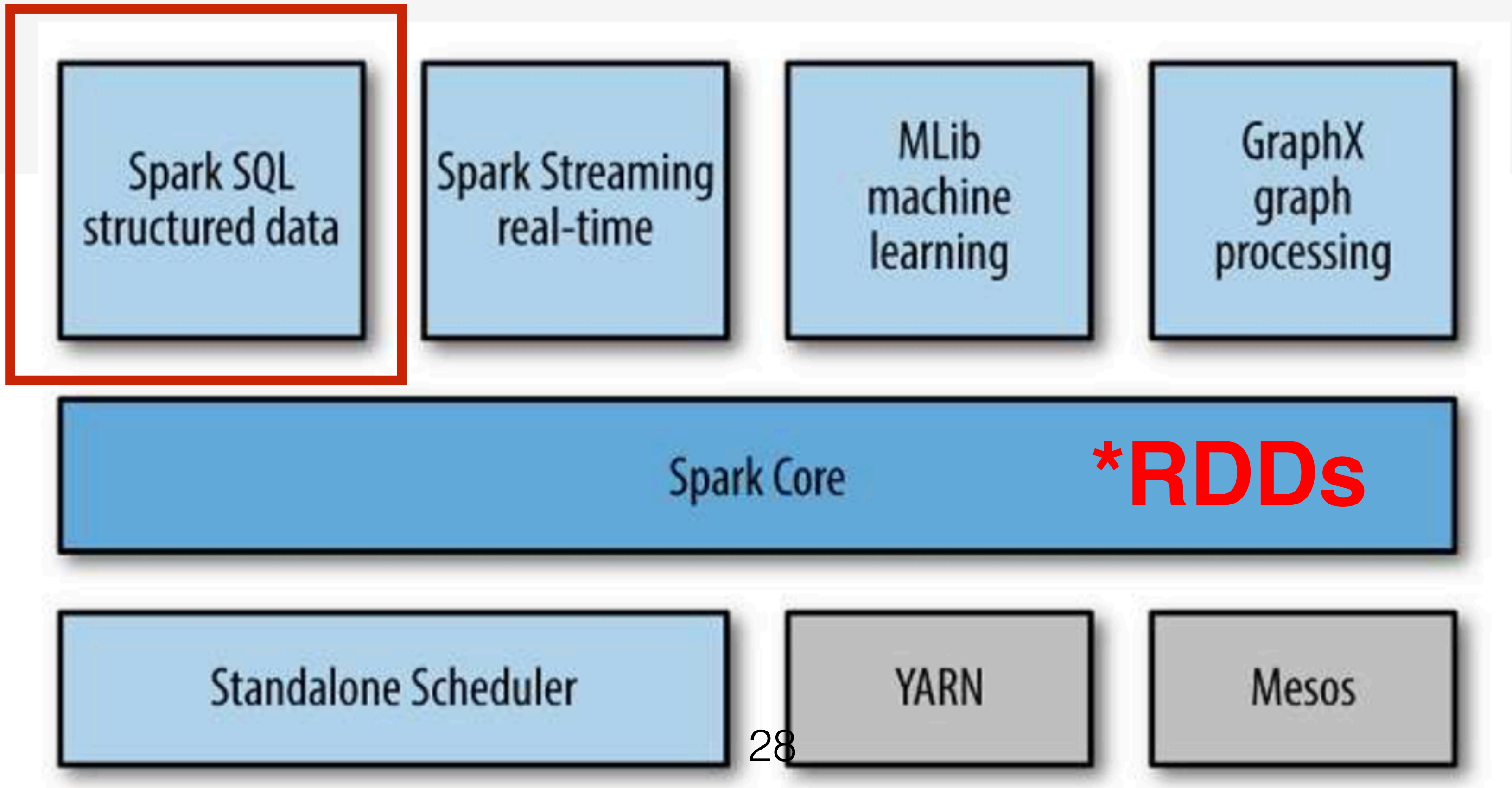


Checkpoint!

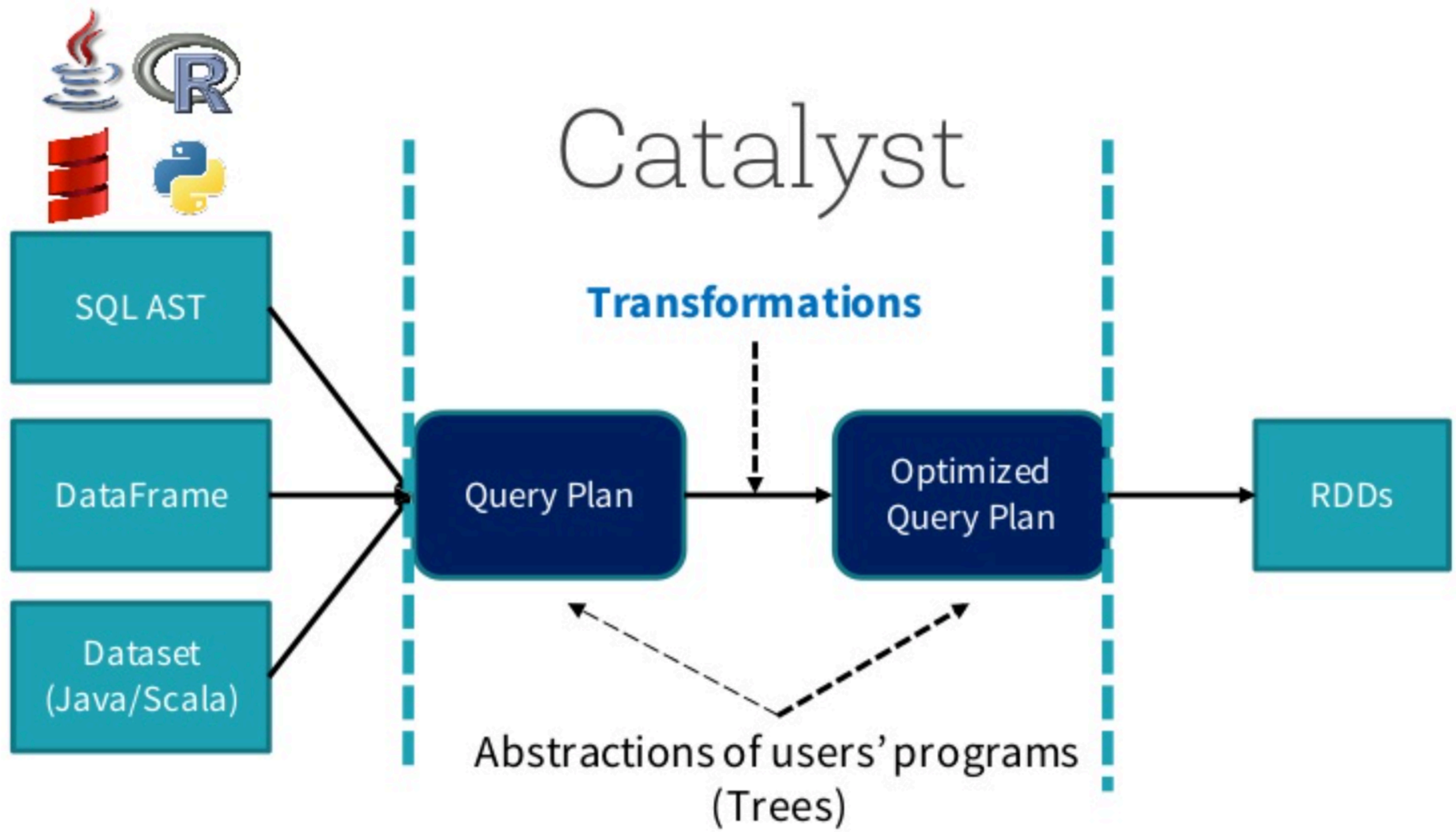
- Introduction to Spark
- Spark Execution Model
- **RDD Definition, Model, Representation, Advantages**

Spark Libraries

- Spark SQL
  - For working withstructured data. Allows you to seamlessly mix SQL queries with Spark programs
- Spark Streaming
  - Allows you to build scalable fault-tolerant streaming applications
- MLlib
  - Implements common machine learning algorithms
- GraphX
  - For graphs and graph-parallel computation



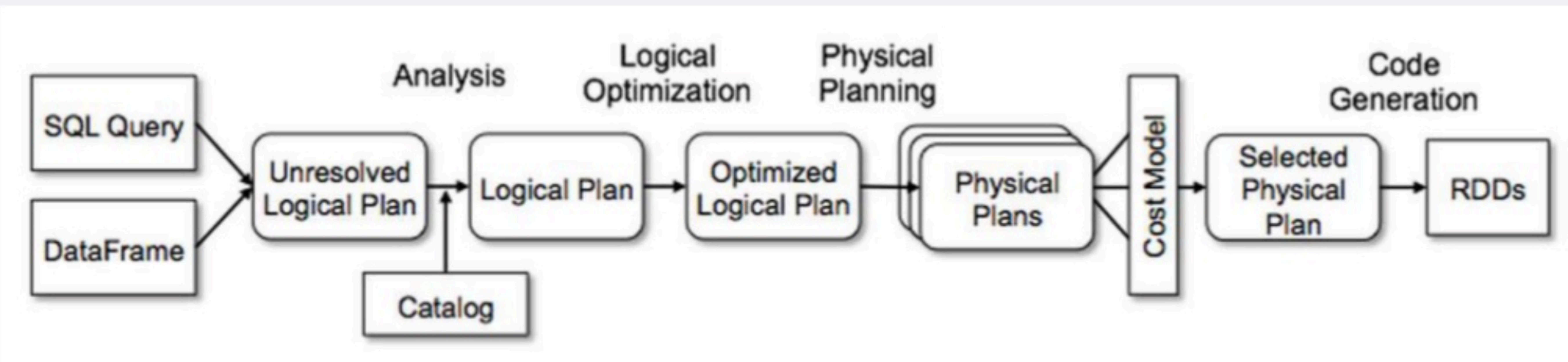
How Catalyst Works: An Overview



databricks 17

Catalyst Optimizer

- Applied to Spark SQL and DataFrame API
- Extensible Optimizer
- Automatically finds the most efficient plan to execute data operations in the users operation

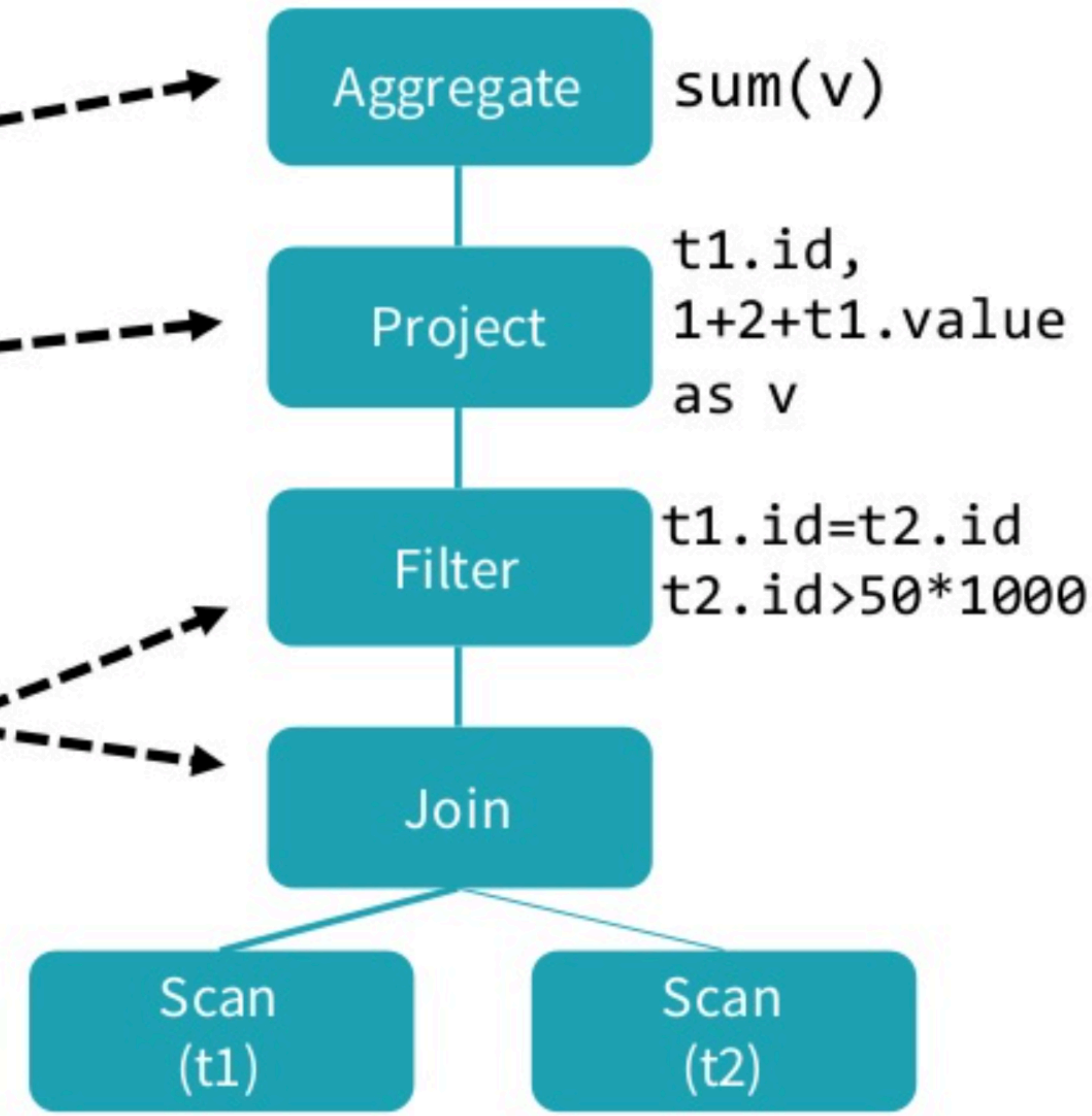


Databricks, Catalyst Optimizer Workflow
https://databricks.com/blog/2015/04/13/deep-dive-into-spark-sqls-catalyst-optimizer.html

Trees: Abstractions of Users' Programs

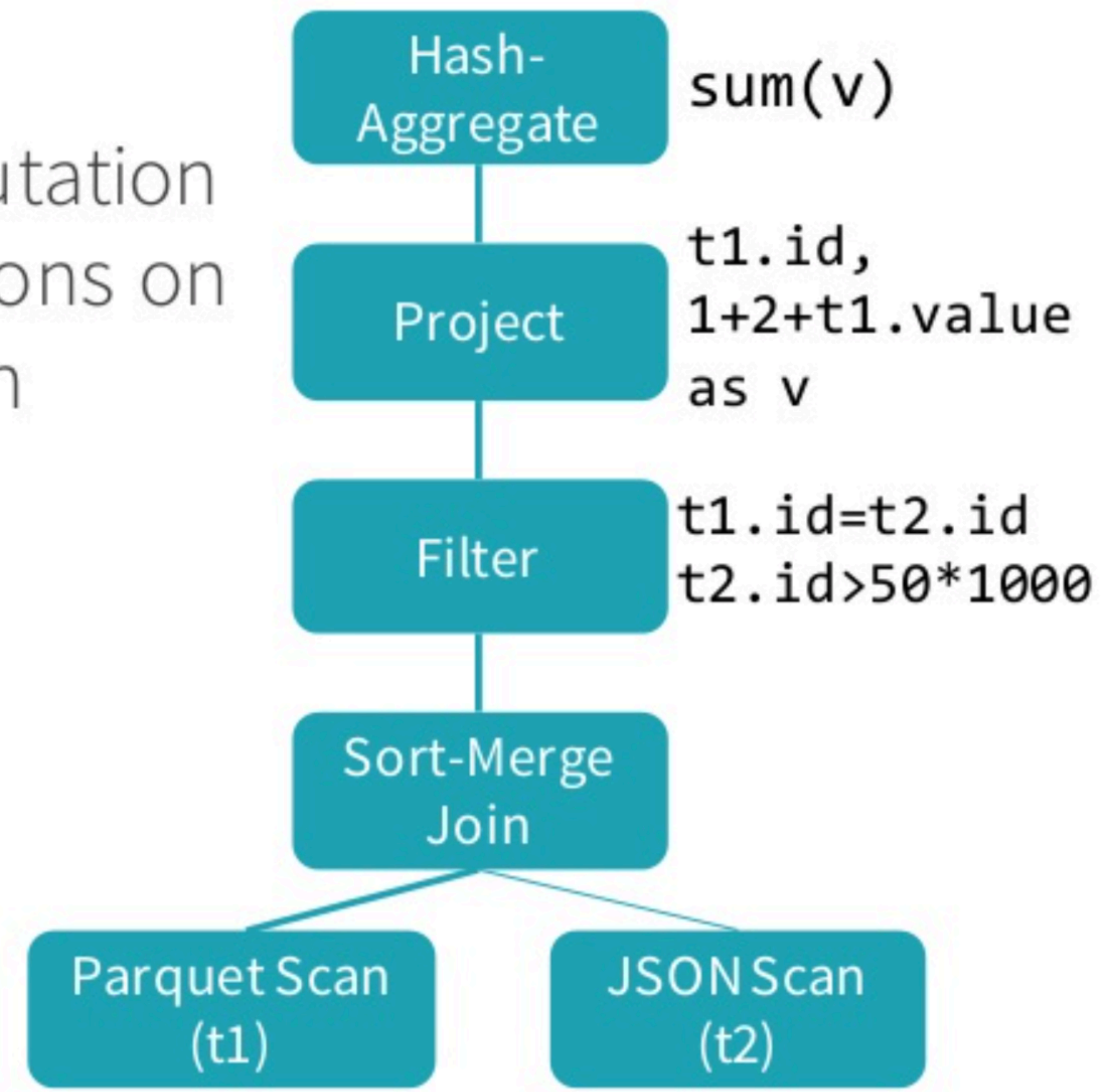
Query Plan

```
SELECT sum(v)
FROM (
  SELECT
    t1.id,
    1 + 2 + t1.value AS v
  FROM t1 JOIN t2
  WHERE
    t1.id = t2.id AND
    t2.id > 50 * 1000) tmp
```



Physical Plan

- A Physical Plan describes computation on datasets with specific definitions on how to conduct the computation



Checkpoint!

- Introduction to Spark
- Spark Execution Model
- RDD Definition, Model, Representation
- **Spark Architecture and Introduction to SparkSQL Data Processing Engine**

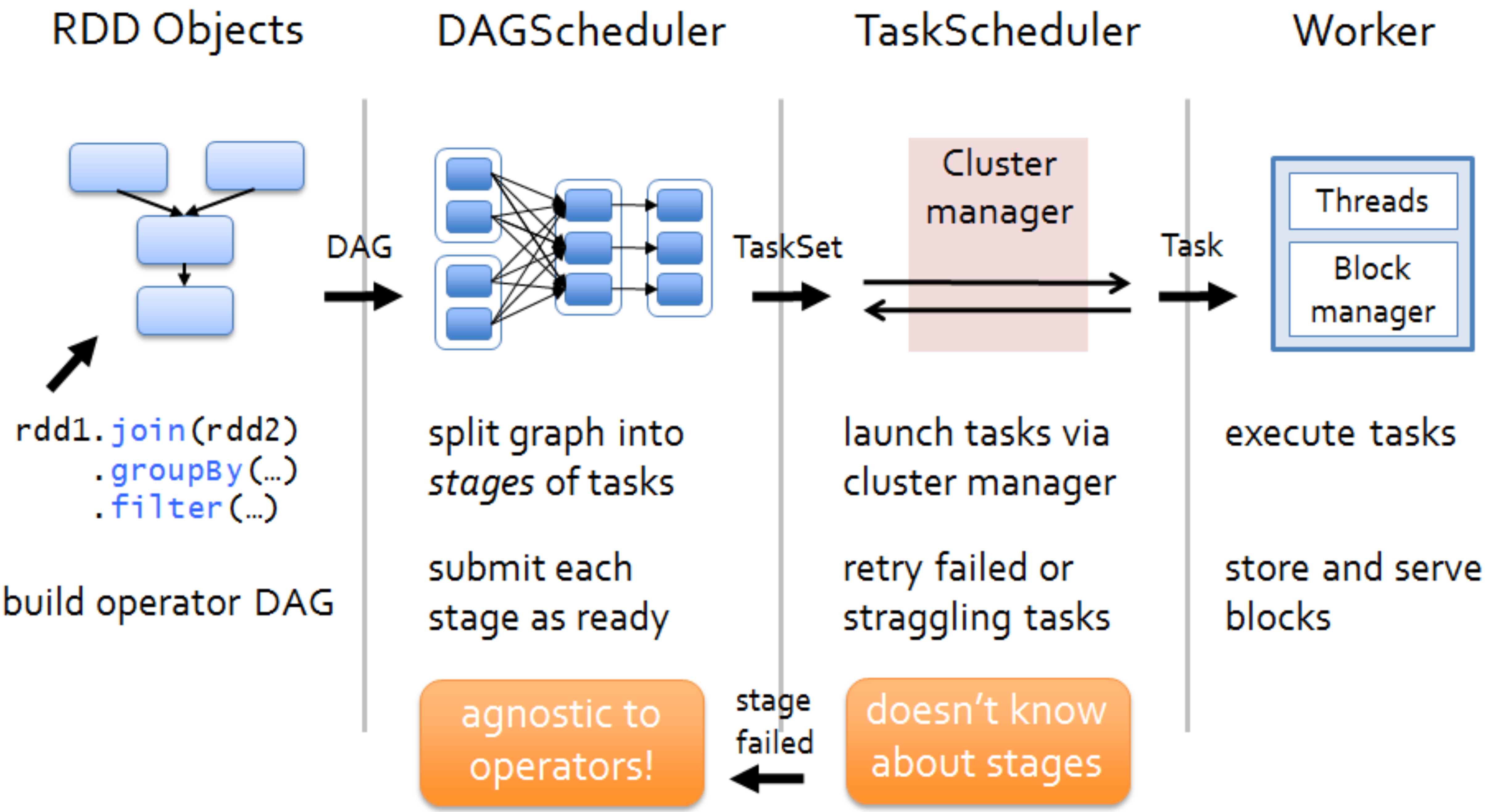
More on RDDs

- **Lineage**: RDD has enough information about how it was derived from other datasets

- **Immutable**: RDD is a read-only, partitioned collection of records
  - Checkpointing of RDDs with long lineage chains can be done in the background.
  - Mitigating stragglers: We can use backup tasks to recompute transformations on RDDs

- **Persistence level**: Users can choose a *re-use* storage strategy (caching in memory, storing the RDD only on disk or replicating it across machines; also chose a persistence priority for data spills)

DAG of RDDs

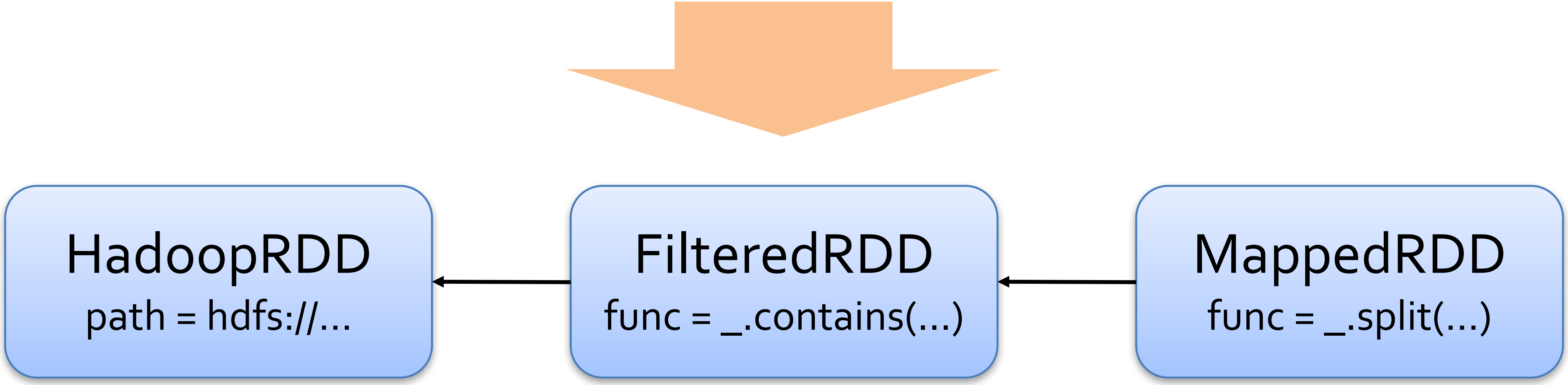


*https://trongkhoanguyenblog.wordpress.com/2014/11/27/understand-rdd-operations-transformations-and-actions/

Fault Tolerance

RDDs track the series of transformations used to build them (their *lineage*) to recompute lost data

E.g:
```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```



Borrowed slide

Advantages of the RDD model

Aspect	RDDs	Distr. Shared Mem.
Reads	Coarse- or fine-grained	Fine-grained
Writes	Coarse-grained	Fine-grained
Consistency	Trivial (immutable)	Up to app / runtime
Fault recovery	Fine-grained and low-overhead using lineage	Requires checkpoints and program rollback
Straggler mitigation	Possible using backup tasks	Difficult
Work placement	Automatic based on data locality	Up to app (runtimes aim for transparency)
Behavior if not enough RAM	Similar to existing data flow systems	Poor performance (swapping?)

Table 1: Comparison of RDDs with distributed shared memory.

Other Engine Features: Implementation

- Not covered in details
- Some **Summary**:
  - Spark local vs Spark Standalone vs Spark cluster (Resource sharing handled by Yarn/Mesos)
  - *Job Scheduling:* DAGScheduler vs TaskScheduler (Fair vs FIFO at task granularity)
  - *Memory Management: de*serialized in-memory(fastest) VS serialized in-memory VS on-disk persistent
  - *Support for Checkpointing:* Tradeoff between using lineage for recomputing partitions VS checkpointing partitions on stable storage

Checkpoint!

- Introduction to Spark
- Spark Execution Model
- RDD Definition, Model, Representation
- Spark Architecture and Introduction to SparkSQL Data Processing Engine
- **Final words on RDD Features and Advantages**

Slide Credits

1. https://www.slideshare.net/ZahraEskandari1/apache-spark-fundamentals-95407778

2. https://www.slideshare.net/indicthreads/scrap-your-mapreduce-apache-spark

3. https://www.slideshare.net/databricks/sparksql-a-compiler-from-queries-to-rdds