Spark

Fast, Interactive, Language-Integrated Cluster Computing

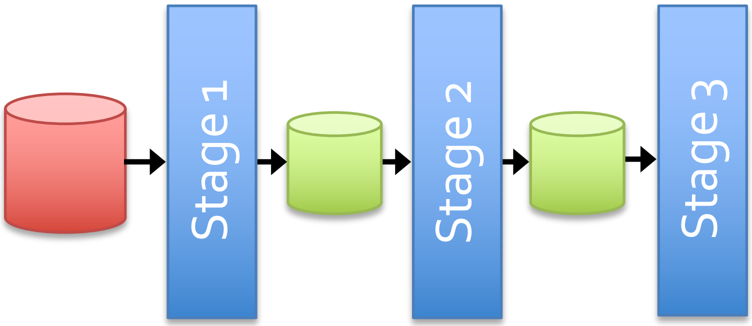
Muhammad Bilal

*Adapted from ampLabs introductory Spark slides

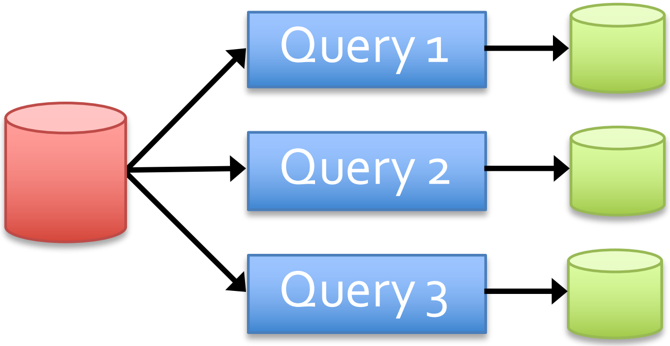
Why go Beyond MapReduce?

Complex jobs and interactive queries both need one thing that MapReduce lacks:

Efficient primitives for **data sharing**



Iterative algorithm



Interactive data mining

Why go Beyond MapReduce?

Complex jobs and interactive queries both need one thing that MapReduce lacks:

Efficient primitives for **data sharing**

Iterative algorithm

Interactive data mining

In MapReduce, the only way to share data across jobs is stable storage (e.g. HDFS) -> **slow!**

Spark - Features

Extends MapReduce model to better support two common classes of analytics apps:

» **Iterative** algorithms (machine learning, graphs)

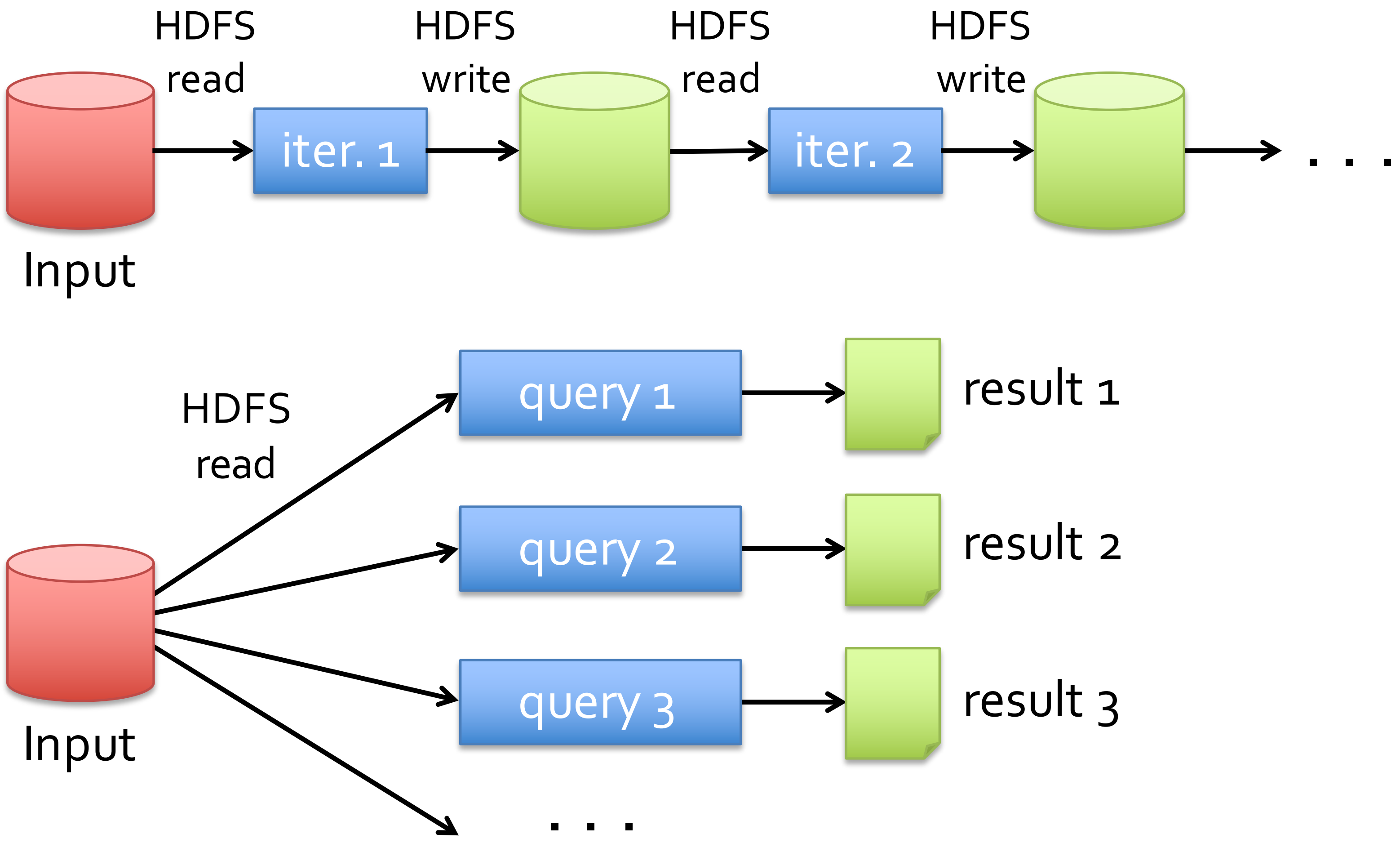
» **Interactive** data mining

Enhance programmability:

» Integrate into Scala programming language

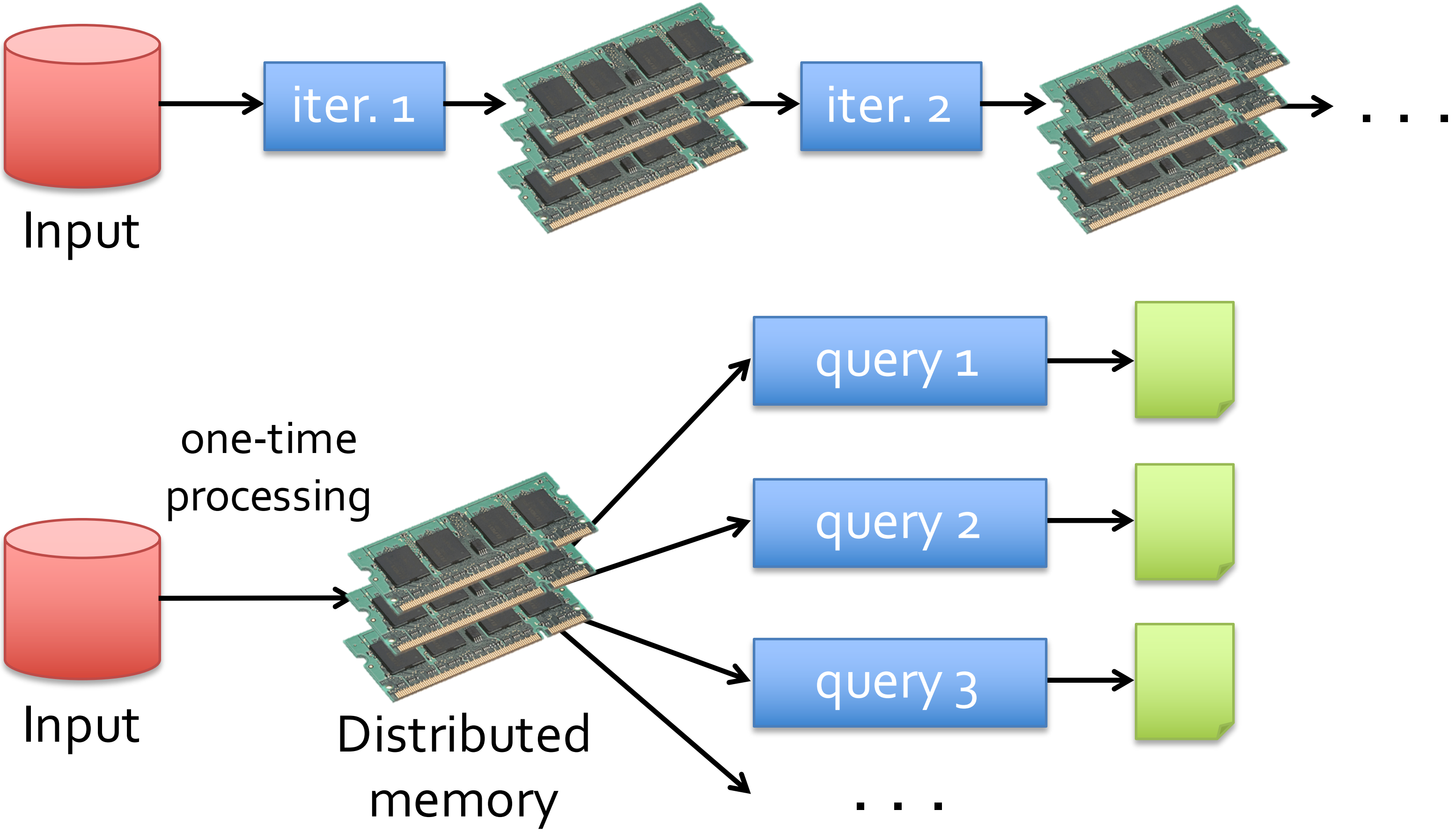
» Allow interactive use from Scala interpreter

Examples



I/O and serialization can take **90%** of the time

Goal: In-Memory Data Sharing



10-100× faster than network and disk

Solution: Resilient Distributed Datasets (RDDs)

Distributed collections of objects that can be stored in memory for fast reuse

Automatically recover lost data on failure

Support a wide range of applications

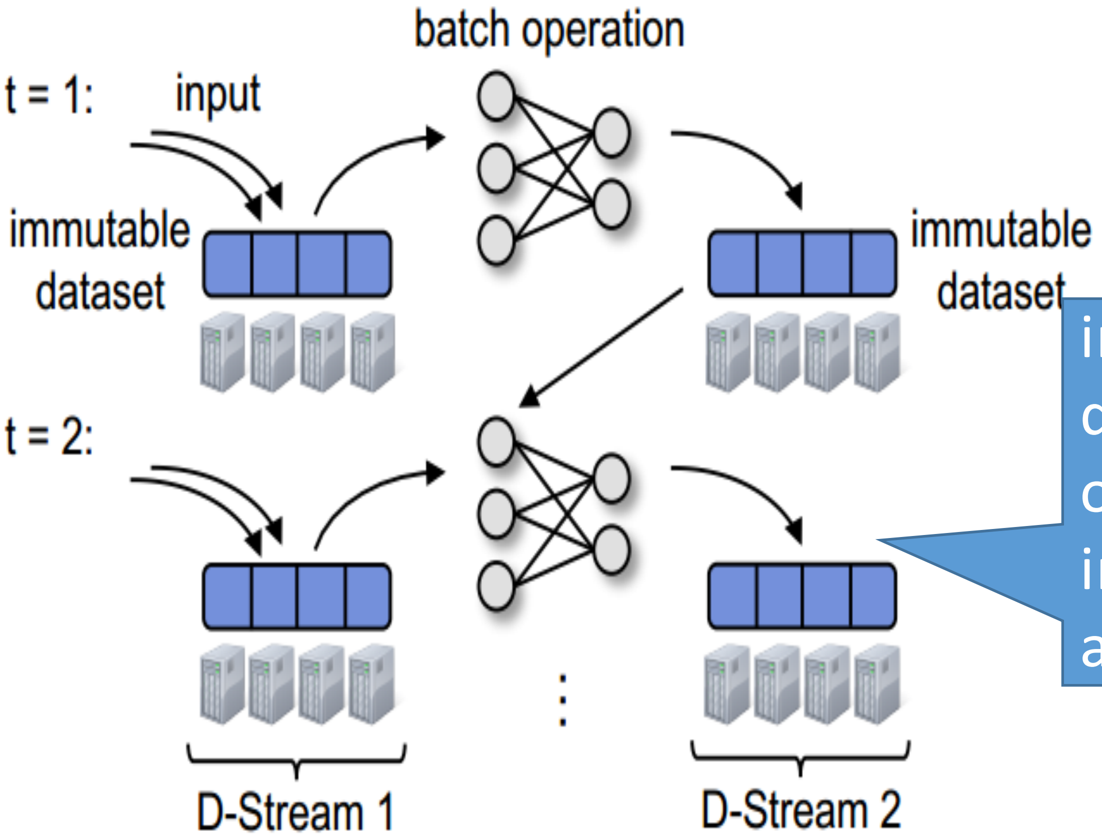
RDDs maintain **lineage** information that can be used to reconstruct lost partitions

RDD Operations

Transformations (define a new RDD)	map filter sample groupByKey reduceByKey sortByKey flatMap union join cogroup cross mapValues
Actions (return a result to driver program)	collect reduce count save lookupKey take

Discretized Streams

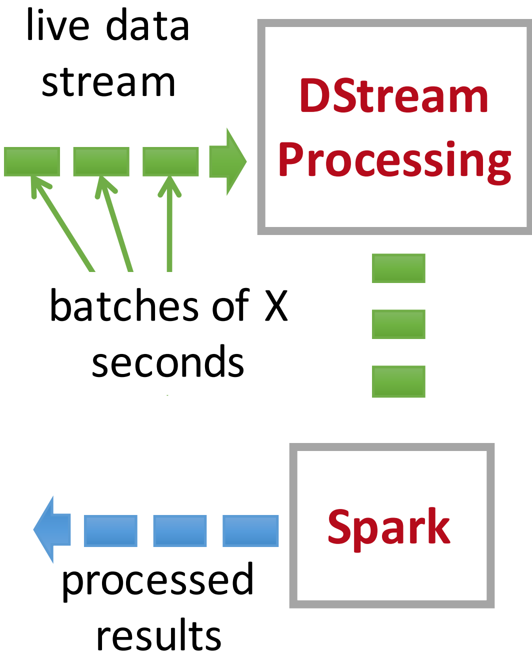
- A streaming computation as a series of very small, deterministic batch jobs



immutable dataset (output or state); stored in memory as Spark RDD

Discretized Streams

- **Faults/Stragglers recovery without replication**
  - State between batches kept in memory
  - Deterministic operations→ fault-tolerant
- **Second-scale performance**
  - Try to make batch size as small as possible
  - Smaller batch size → lower end-to-end latency
- **A rich set of operations**
  - Combined with historical datasets
  - Interactive queries



DStream Operations

- Dstream Object: A stream of RDDs
- Transformations
  - Map, filter, groupBy, reduceBy, sort, join…
  - Windowing
  - Incremental aggregation
- Output operator
  - Save RDDs to outside systems(screen, external storage…)

Windowing

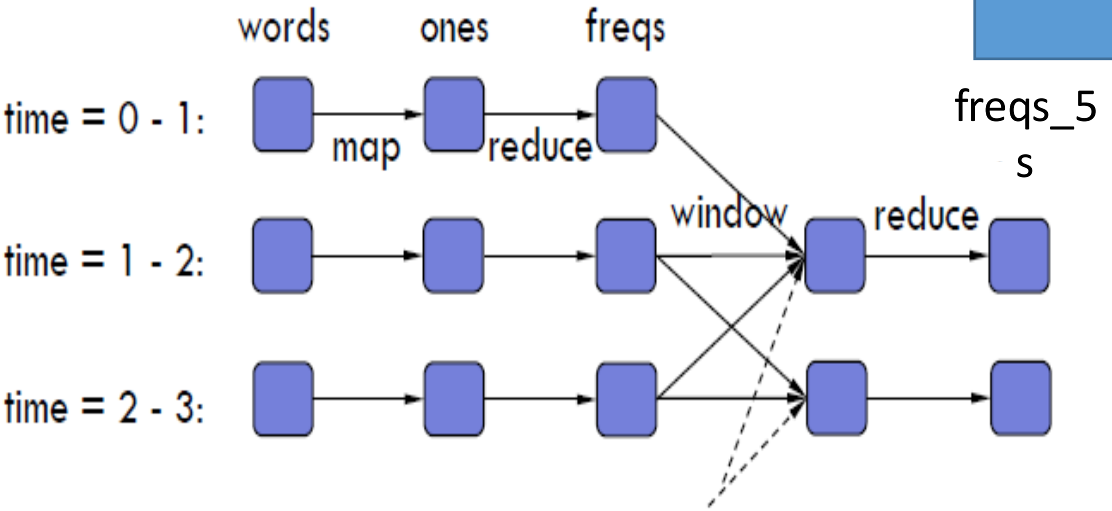
- Count frequency of words received in last 5 seconds

```
words = createNetworkStream("http://...”)
ones = words.map(w => (w, 1))
freqs_5s = ones.reduceByKeyAndWindow(_ + _, Seconds(5),
Seconds(1))
```

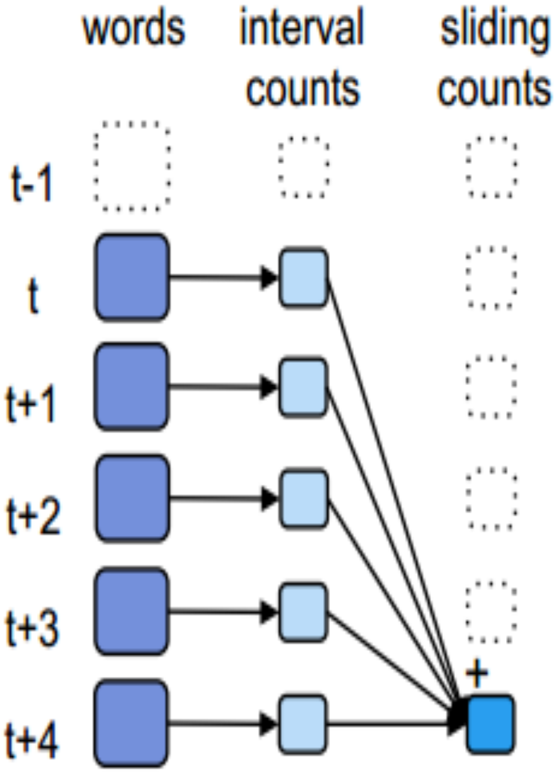
DStream

Transformation

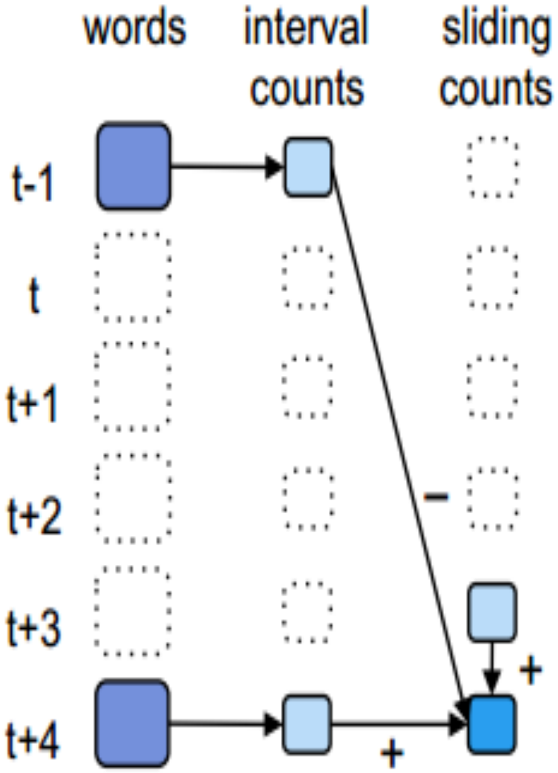
Sliding Window Ops



Incremental aggregation



Aggregation Function
freqs = ones.reduceByKeyAndWindow
(_ + _, Seconds(5), Seconds(1))



Invertible aggregation Function
freqs =
ones.reduceByKeyAndWindow
(_ + _, _ - _, Seconds(5), Seconds(1))

Tutorial Exercises

- Refer to the Lab 6 handout
  - Machine learning and Streaming exercises
- You can reference Spark’s programming guides for extra help:
  - https://spark.apache.org/docs/1.4.1/programming-guide.html