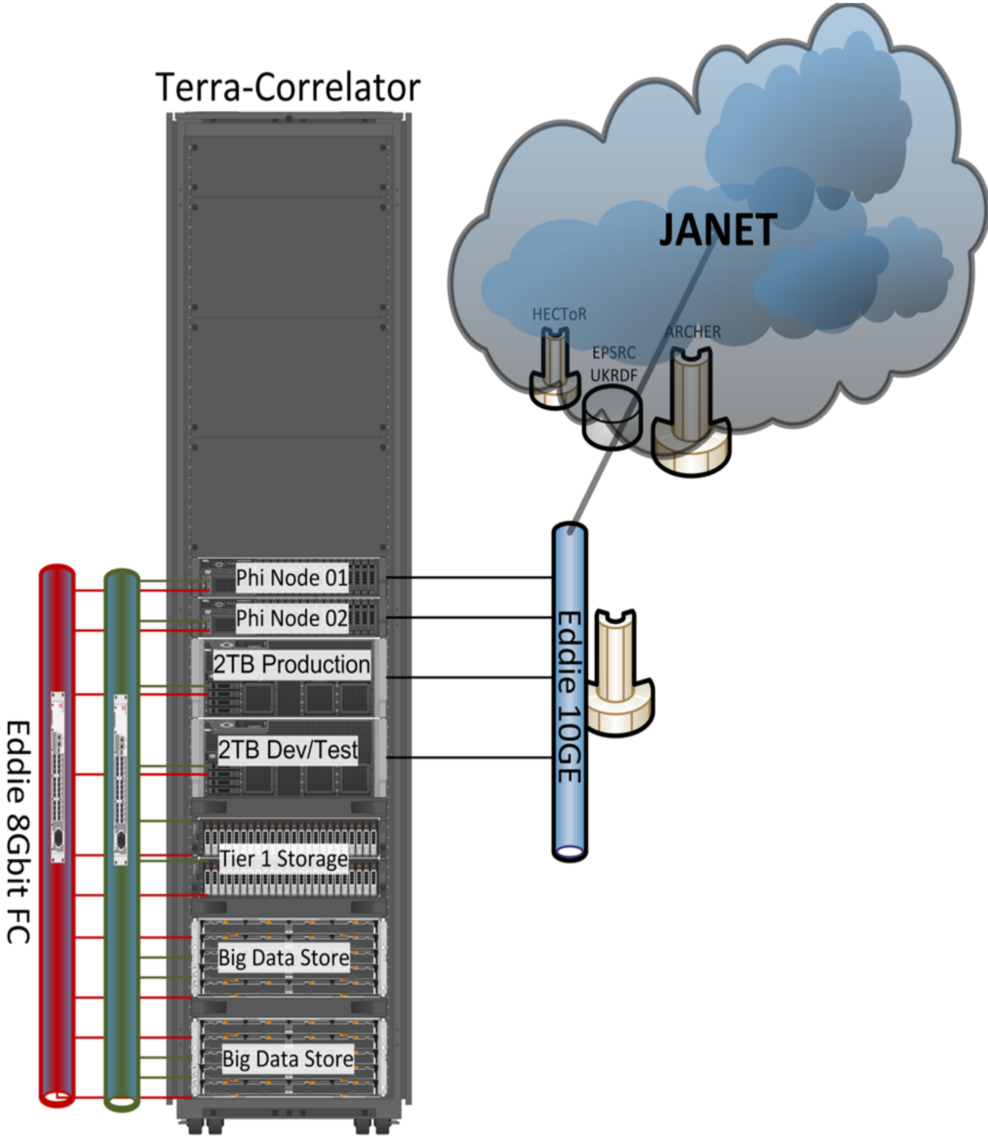
Batch vs Streaming Processing

Rosa Filgueira

Terracorrelator Machine (TC)

- New HPC facility
- Designed for real-time cross-correlational analyses.
- 2 nodes each with 2TB shared memory and 32 cores.
  - cross-correlation
  - post-processing
- 2 Intel Xeon Phi nodes
  - pre-processing

Also used by:
X-ray tomography,
and climate science



Batch Processing

- Batch data processing is an efficient way of processing high volumes of data which is collected over a period of time.

- Data is collected, entered, processed and then the batch results are produced

Streaming processing

- Streaming processing involves a continual input, process and output of data.
- Data must be processed in a small time period (or near real time).
- Processing messages one at time

What we want

- Continuous processing:
  - Source ?
    - Is the data going to be stored in the Terracorrelator ?
    - If so, where ?
      - » Shared memory (/dev/shm/)?
      - » File System
    - Or, Is the data going to be stored in somewhere else ?
      - » Another machine
      - » Cluster ?
- Compatible with Shared Memory
- Avoid writing intermediate files

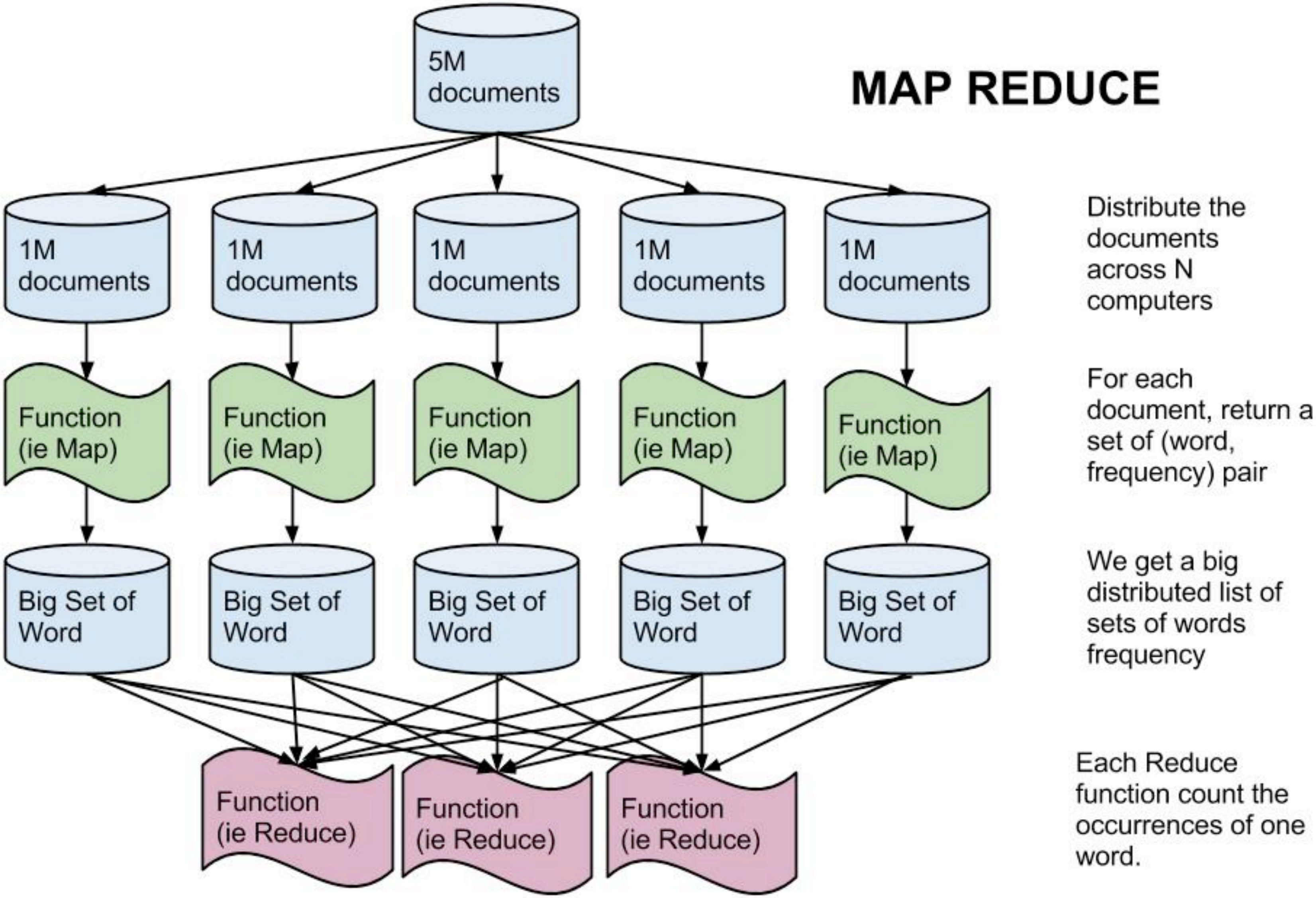
Batch Technology

- Distributed memory:
  - MPI
  - Based on Hadoop:
    - Spark
    - MapReduce
  - dispel4py
- <u>*Shared memory:*</u>
  - One one machine:
    - Mutiple CPUs/GPUS share memory
  - On multiple machines:
    - Shared memory via network
  - OpenMP
  - Multiprocessing Python Libraries
  - pthreads - POSIX

Hadoop

- Hadoop clusters consist of
  - up to thousands of commodity computers
  - a distributed file system called HDFS
- Distributed Programming Model
  - MapReduce:
    - Run code where data resides
    - Based on Maps & Reduce functions
    - Reads and writes from File System
  - Spark
    - Faster than MapReduce
    - DAG graphs Directed acyclic graph
    - Handles most its operations in memory:
    - Spark Straeming → data stream processing in near real time by using micro-batches
    - Mlib -→ Library for machine learing
    - Relative immature

Map Reduce Example



MapReduce - Problems

- State between steps goes to distributed file systems
- Slow due to replication and disk storage
- It is needed HDFS file system

Spark

- Resilient Distributed Dataset (RDD)
  - Fault tolerant collection of elements distributed across many serves on which we can perform parallel operations
  - Persistence:
    - HDFS
    - Memory
- Operations:
  - Transformations : Operations on RDDs that return new RDDs
  - Actions: Operations on RDD witch return a final value or write the data to an external storage system

```
val sc = new SparkContext()
val lines = sc.textFile("log.txt")   // RDD[String]

// Transform using standard collection operations
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split('\t')(2))
```

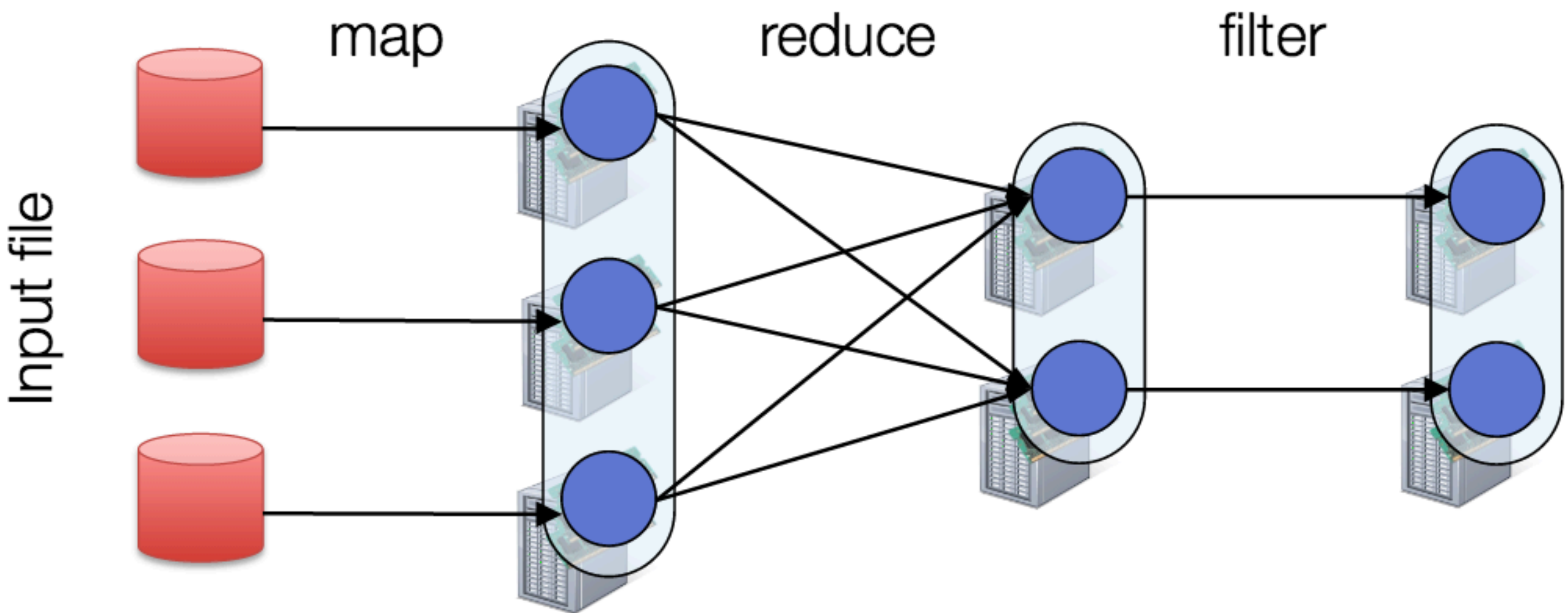
lazily evaluated

```
messages.saveAsTextFile("errors.txt")
```

kicks off a computation

Spark

```
file.map(lambda rec: (rec.type, 1))
    .reduceByKey(lambda x, y: x + y)
    .filter(lambda (type, count): count > 10)
```



Spark -RDD

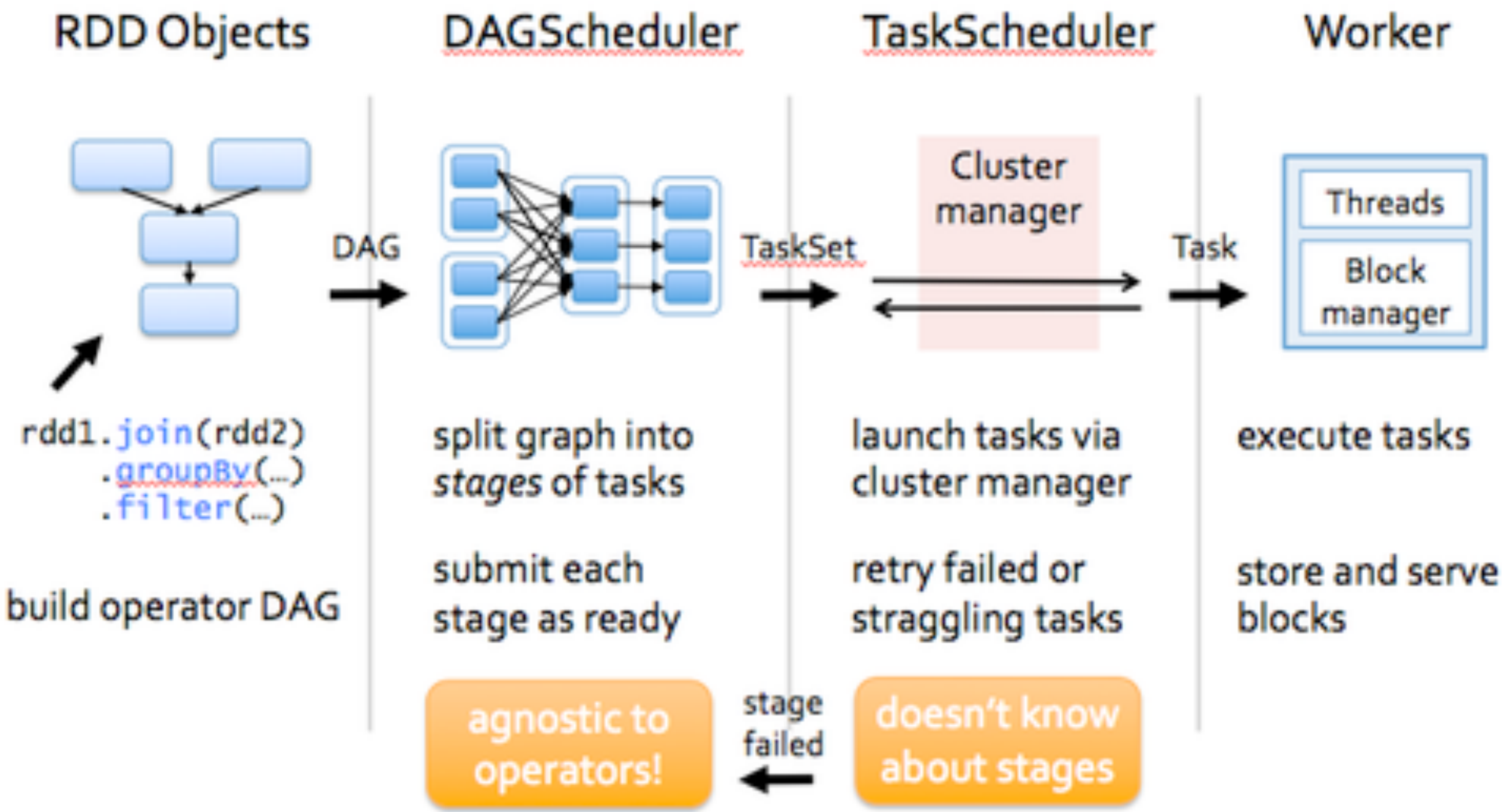
RDD

CAKESOLUTIONS
ENTERPRISE SOFTWARE SERVICES

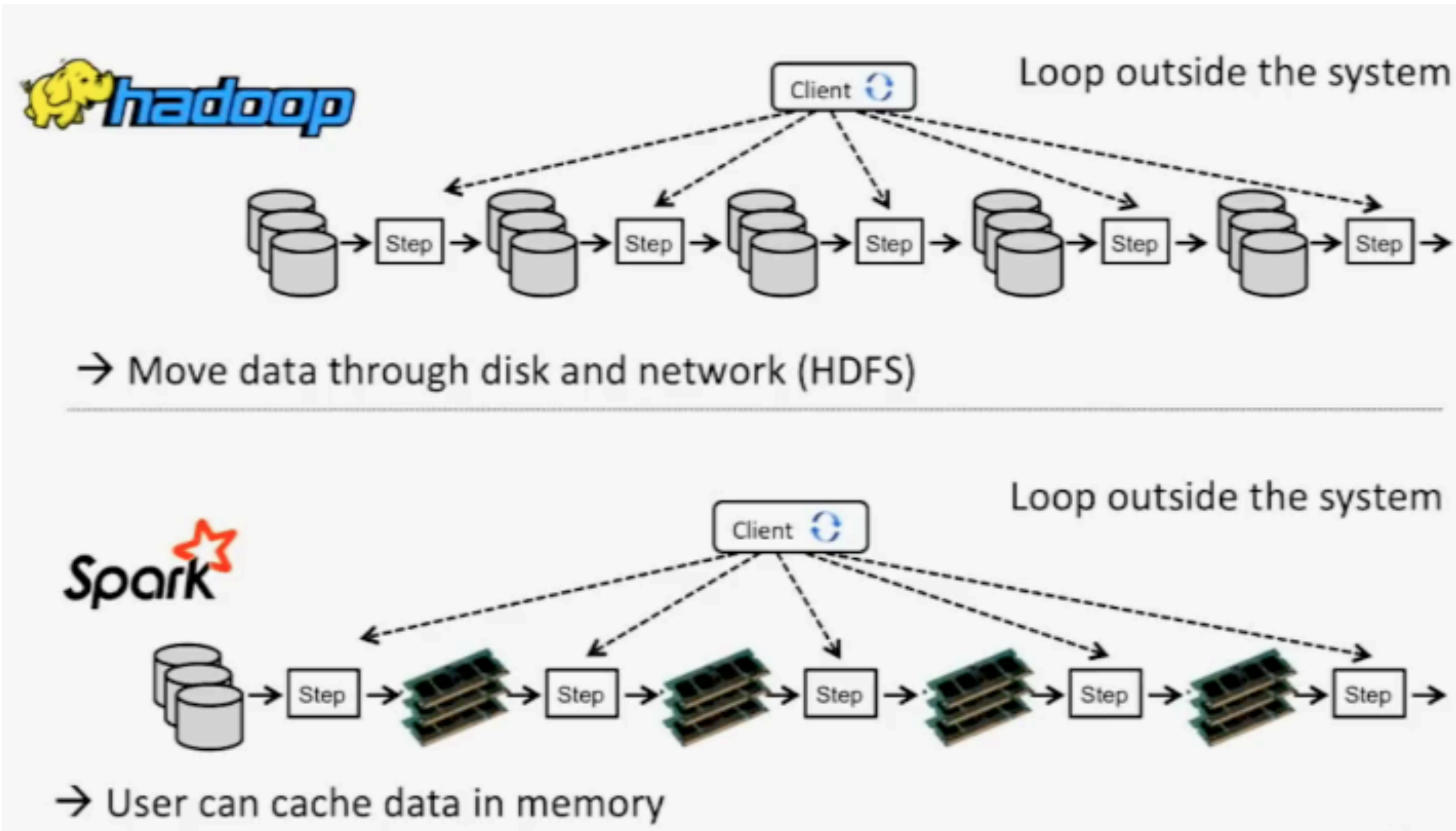
- transformations
  - lazy, form the DAG
  - map, filter, flatMap, mapPartitions, mapPartitionsWithIndex, sample, union, intersection, distinct, groupByKey, reduceByKey, sortByKey, join, cogroup, repatition, cartesian, glom, ...
- actions
  - execute DAG
  - retrieve result
  - reduce, collect, count, first, take, foreach, saveAs…, min, max, ...

- different categories of transformations with different complexity, performance and sematics
- e.g. mapping, filtering, grouping, set operations, sorting, reducing, partitioning
- full list https://spark.apache.org/docs/1.3.0/api/scala/index.html#org.apache.spark.rdd.RDD

Spark Scheduler



MapReduce Vs Spark

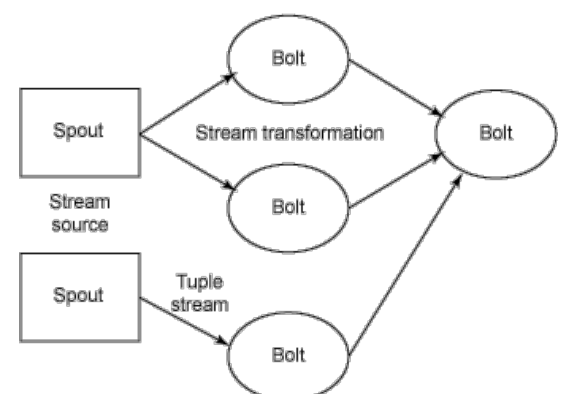


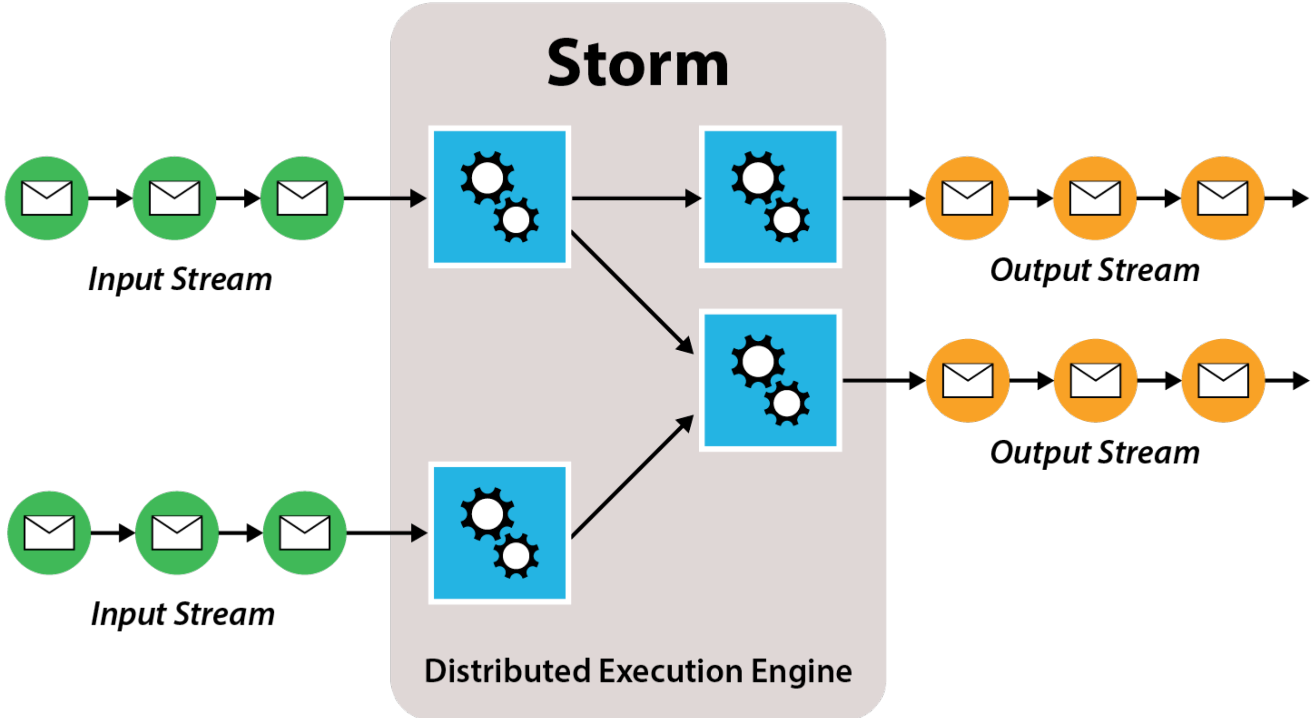
Streaming Technology

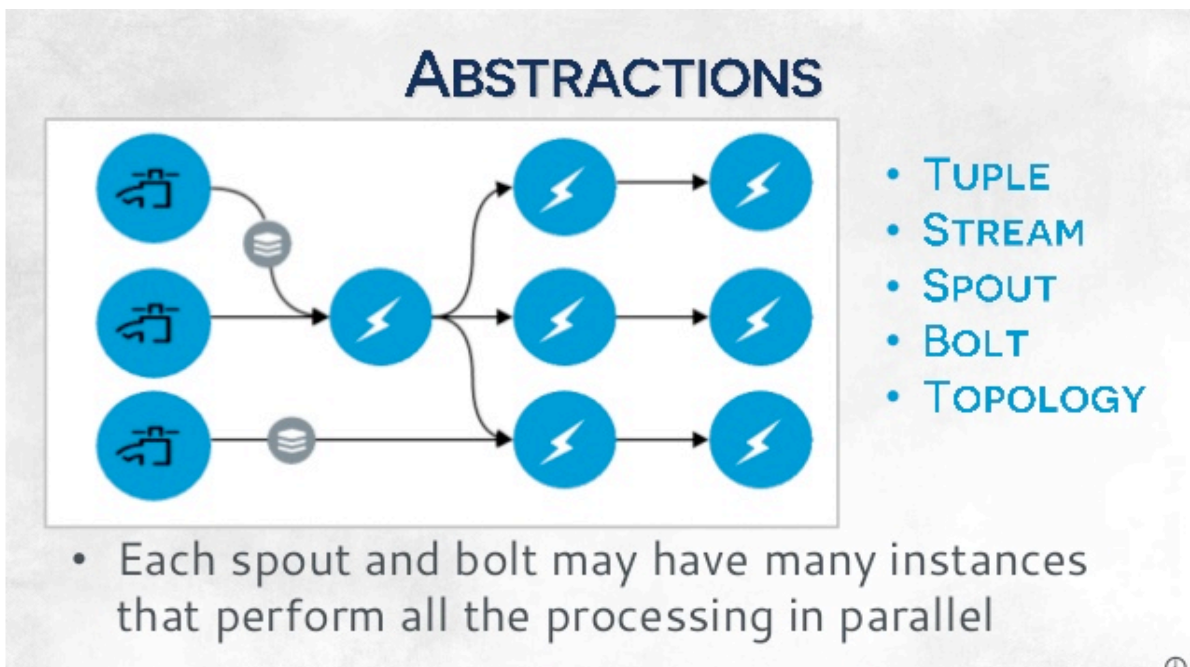
- Distributed Memory:
  - Storm
  - Spark Streaming
  - dispel4py
- <u>*Shared Memory:*</u>
  - dispel4py
  - Storm ??

Storm

- Storm supports **true streaming** processing model (via Core storm layer) in strictest sense, with an **additional micro-batching model**
- Main concepts:
  - Stream: sequence of tuples
    - Storm provides primitives to transform a stream into a new one.
  - Spout: Source of stream
  - Bolt: Consumes any number of input streams, does some processing, and possible emits new stream.
  - Topology: The abstraction of the program to submit to a Storm cluster for execution.
  - Edges: Links between the nodes.
  - ***Communication between nodes:***
    - ***Same node: Inter-thread messaging library***
    - Different nodes: ZeroMQ or Netty (network)




Storm
Input Stream
Input Stream
Output Stream
Output Stream
Distributed Execution Engine


ABSTRACTIONS
• TUPLE
• STREAM
• SPOUT
• BOLT
• TOPOLOGY
• Each spout and bolt may have many instances that perform all the processing in parallel

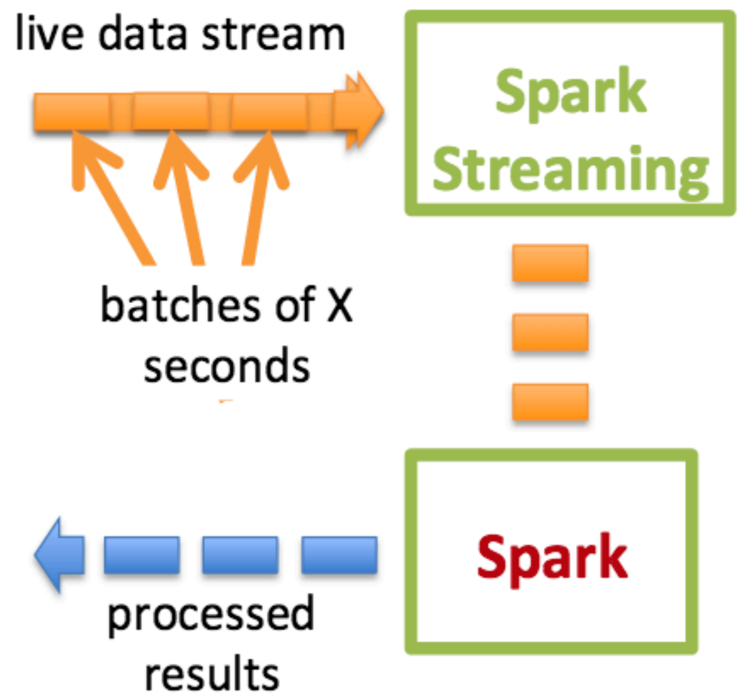
Storm -- Problems

- It is necessary to have a Storm cluster

Spark Streaming

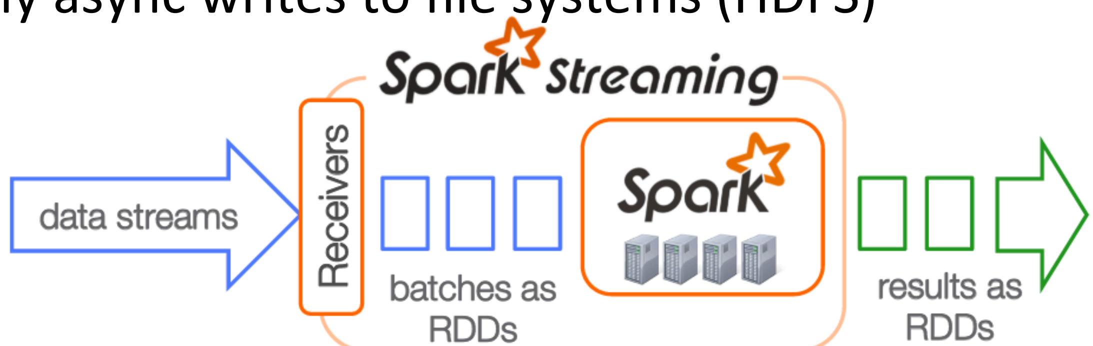
Run a streaming computation as a **series of very small, deterministic batch jobs**

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches



Spark Streaming

- Spark Streaming is a wrapper over Spark (Batch processing) framework
- It internally converts the incoming stream into small micro-batches, which are then handed over to Spark framework for processing
- DStream:Abstraction in Spark Streaming, is a continuous sequence of RDDs (of the same type) representing a continuous stream of data
- Two operations:
  - Stream transformations:
    - Convert a DStream to another
  - Output operators
    - Write data to external systems.
- Periodically async writes to file systems (HDFS)



Spark Streaming

- Data receiving:
  - Receive data and store data in Spark
- Data processing
  - Transfer the data stored in Spark into the Dstream
  - Then you can apply the two operations on the Dstream
- Two types of parallelism:
  - Receiving the stream
  - Processing the stream

- http://stanford.edu/~rezab/dao/slides/lec12.pdf

Spark Streaming – Problems

- Problems:
  - Run Spark on the same nodes as HDFS
  - While Spark can perform a lot of its computation in memory, it still uses local disks to store data that doesn't fit in RAM, as well as to preserve intermediate output between stages
  - Distributed enviroments

- http://es.slideshare.net/DavorinVukelic/realtime-streaming-with-apache-spark-streaming-and-apache-storm

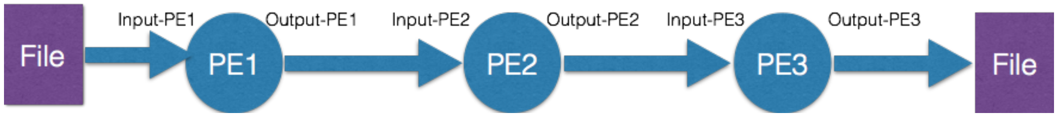
dispel4py

- **Stream-based**
  - Tasks are connected by streams and not by intermediate files
  - Multiple streams in & out
  - Optimization based on avoiding IO
- **Maps workflows dynamically** onto multiple enactment systems:
  - Automatic parallelization
  - Without cost to users
- **Python** language for describing tasks and connections

dispel4py – Processing element

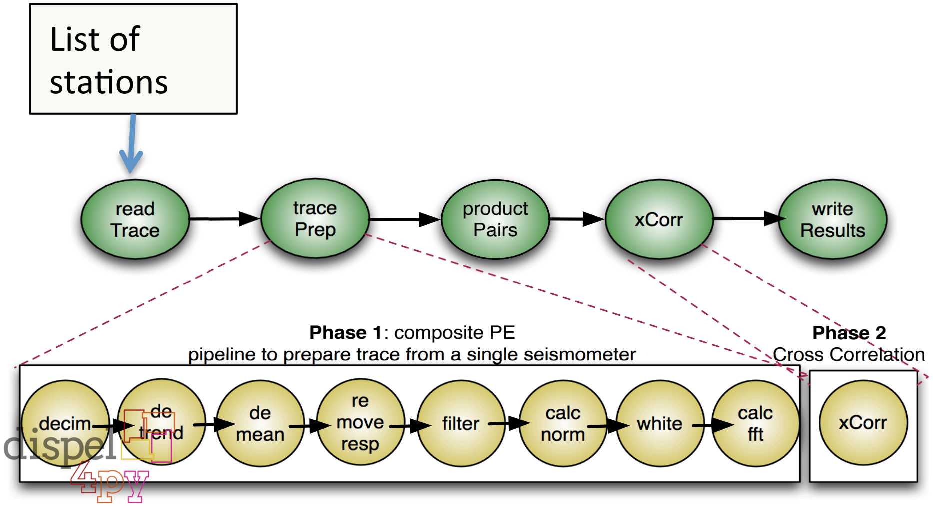
- **PEs represent the basic** computational**:**
  Algorithm, service, data transformation
- Shared**: Storing them into the registry**
- **PEs ~ "Lego bricks" of tasks.**
  Users can assemble **them into workflow as they** wish**.**

- **Consume/Produce any number**
  and types of input/output streams



Seismic ambient noise cross-correlation – dispel4py workflow + TC + multi mapping

- Phase 1- Preprocess: Time series data (traces) from seismic stations are preprocessed in parallel
- Phase 2: Cross-Correlation: Pairs all of the stations and calculates the cross-correlation for each pair (complexity O(n2)).



- Select vertical component (channel BHZ)
- Decimation to 4 Hz sampling rate
- Remove mean & trend
- Remove station response
- Bandpass filter for 0.01 Hz - 1.00 Hz
- Moving average re-normalization
- Spectral whitening

dispel4py mappings

- **Sequential**
  - Sequential mapping for local testing
  - Ideal for local resources: Laptops and Desktops
- **Multiprocessing**
  - Python's multiprocessing library
  - Ideal for shared memory resources
- **MPI**
  - Distributed Memory, message-passing parallel programming model
  - Ideal for HPC clusters
- **STORM**
  - Distributed Real-Time computation System
  - Fault-tolerant and scalable

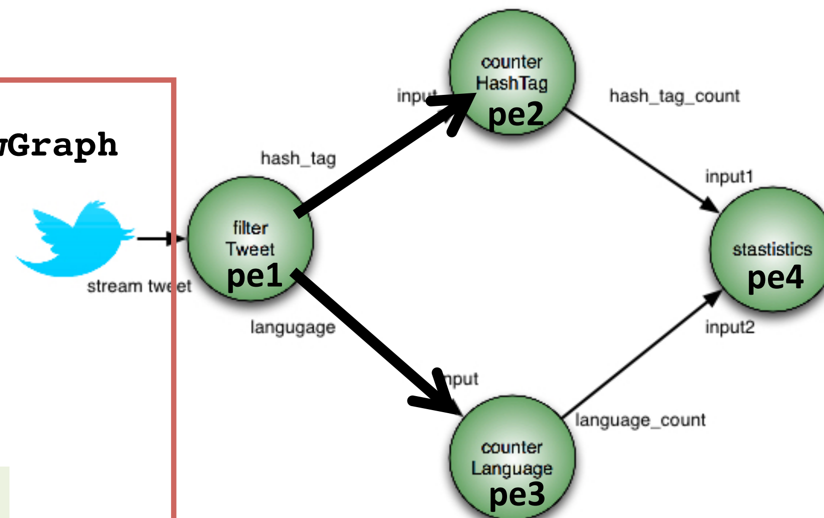
dispelp4y basic concepts– Example of a dispel4py workflow

```
from dispel4py.workflow_graph import WorkflowGraph


pe1 = filterTweet ()
pe2 = counterHashTag ()
pe3 = counterLanguage ()
pe4 = statistics ()

graph = WorkflowGraph ( )

graph.connect(pe1,'hash_tag',pe2,'input')
graph.connect(pe1,'language',pe3,'input')
graph.connect(pe2,'hash_tag_count',pe4,'input1')
graph.connect(pe3,'language_count',pe4,'input2')
```

PEs objects

Graph

Connections



Users only have to implement:

- PEs
- Connections

dispel4py --- Installations and Links

- This is all you need:

pip install dispel4py

- Web site http://dispel4py.org/
- GitHub: https://github.com/dispel4py/dispel4py
- Documentation: http://dispel4py.org/documentation/
- I have plenty material that I could share ☺

dispel4py – problems

- The workflow finishes after processing an input stream data
- Cronjobs are needed for continuing data processing.

Real time and Batch processing

- The decision to select the best data processing depends on:
  - the types and sources of data
  - processing time needed to get the job done
  - hardware

Conclusions

- For the Terracorrelator Machine:
  - Continuous processing technology
  - Compatible with Shared Memory
  - Avoid writing intermediate files
    - *dispel4py*
    - Could be Storm, but I am not 100 % convinced.