

BIG DATA CONCEPTS

Serkan ÖZAL

**Middle East Technical University
Ankara/TURKEY**

October 2013

Contents

- Big Data and Scalability
- NoSQL
 - Column Stores
 - Key-Value Stores
 - Document Stores
 - Graph Database Systems
- Batch Data Processing
 - MapReduce
 - Hadoop
 - Running Analytical Queries over Offline Big Data
 - Hive
 - Pig
- RealTime Data Processing
 - Storm

Big Data

- Collect.
- Store.
- Organize.
- Analyze.
- Share.

Data growth outruns the ability to manage it so we need **scalable** solutions.

Scalability - Vertical

- Increasing server capacity.
- Adding more CPU, RAM.
- Managing is hard.
- Possible down times

Scalability - Horizontal

- Adding servers to existing system with little effort, aka Elastically scalable.
 - Bugs, hardware errors, things fail all the time.
 - It should become cheaper. Cost efficiency.
- Shared nothing.
- Use of commodity/cheap hardware.
- Heterogeneous systems.
- Controlled Concurrency (avoid locks).
- Service Oriented Architecture. Local states.
 - Decentralized to reduce bottlenecks.
 - Avoid Single point of failures.
- Asynchrony.
- Symmetry, you don't have to know what is happening. All nodes should be symmetric.

NoSQL

- Stands for **Not Only SQL**
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins
- Offers BASE properties instead of ACID (Atomicity, Consistency, Isolation, Durability) properties
 - BAsically available: Nodes in the a distributed environment can go down, but the whole system shouldn't be affected.
 - Soft State (scalable): The state of the system and data changes over time.
 - Eventual Consistency: Given enough time, data will be consistent across the distributed system.

What is Wrong With RDBMS?

- One size fits all? Not really.
- Rigid schema design.
- Harder to scale.
- Replication.
- Joins across multiple nodes? Hard.
- How does RDMS handle data growth? Hard.

Advantages & Disadvantages

- **Advantages**

- Flexible schema
- Quicker/cheaper to set up
- Massive scalability
- Eventual consistency → higher performance & availability

- **Disadvantages**

- No declarative query language → more programming
- Eventual consistency → fewer guarantees

NoSQL Systems

- **Several incarnations**
 - Column Stores
 - Key-Value Stores
 - Document Stores
 - Graph Databases

Column Stores

- Each storage block contains data from only one column
- More efficient than row (or document) store if:
 - Multiple row/record/documents are inserted at the same time so updates of column blocks can be aggregated
 - Retrievals access only some of the columns in a row/record/document
- Example Systems
 - HBase

Key-Value Stores

- Extremely simple interface
 - Data model: (key, value) pairs
 - Operations:
 - Insert(key,value), Fetch(key), Update(key), Delete(key)
- Implementation: efficiency, scalability, fault-tolerance
 - Records distributed to nodes based on key
 - Replication
 - Single-record transactions, “eventual consistency”
- Example systems
 - Google BigTable, Amazon Dynamo, Cassandra, Voldemort, ...

Document Stores

- Like Key-Value Stores except value is document
 - Data model: (key, document) pairs
 - Document: JSON, XML, other semistructured formats
 - Basic operations:
 - Insert(key,document), Fetch(key),Update(key), Delete(key)
 - Also Fetch based on document contents
- Example systems
 - CouchDB, MongoDB, SimpleDB, ...

Graph Database Systems

- A graph is a collection nodes (things) and edges (relationships) that connect pairs of nodes.
- Attach properties (key-value pairs) on nodes and relationships
- Relationships connect two nodes and both nodes and relationships can hold an arbitrary amount of key-value pairs.
- A graph database can be thought of as a key-value store, with full support for relationships.
- Example systems
 - Neo4j, FlockDB, Pregel, ...

Batch Data Processing

- Batch data processing is an efficient way of processing high volumes of data is where a group of transactions is collected over a period of time.
- Data is collected, entered, processed and then the batch results are produced
- Data is processed by worker nodes and managed by master node.
- Analytical queries can be executed as distributed on offline big data. User queries are converted to Map-Reduce jobs on cluster to process big data.
 - Hive: Supports SQL like query language
 - Pig: More likely a scripting language

Map-Reduce

- MapReduce is a programming model and software framework first developed by Google (Google's MapReduce paper was submitted in 2004)
- A method for distributing a task across multiple nodes
- Each node processes data assigned to that node
- Consists of two developer-created phases
 - Map
 - Reduce
- Nice features of MapReduce systems include:
 - reliably processing a job even when machines die
 - parallellization on thousands of machines

Map

- **"Map" step:** The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

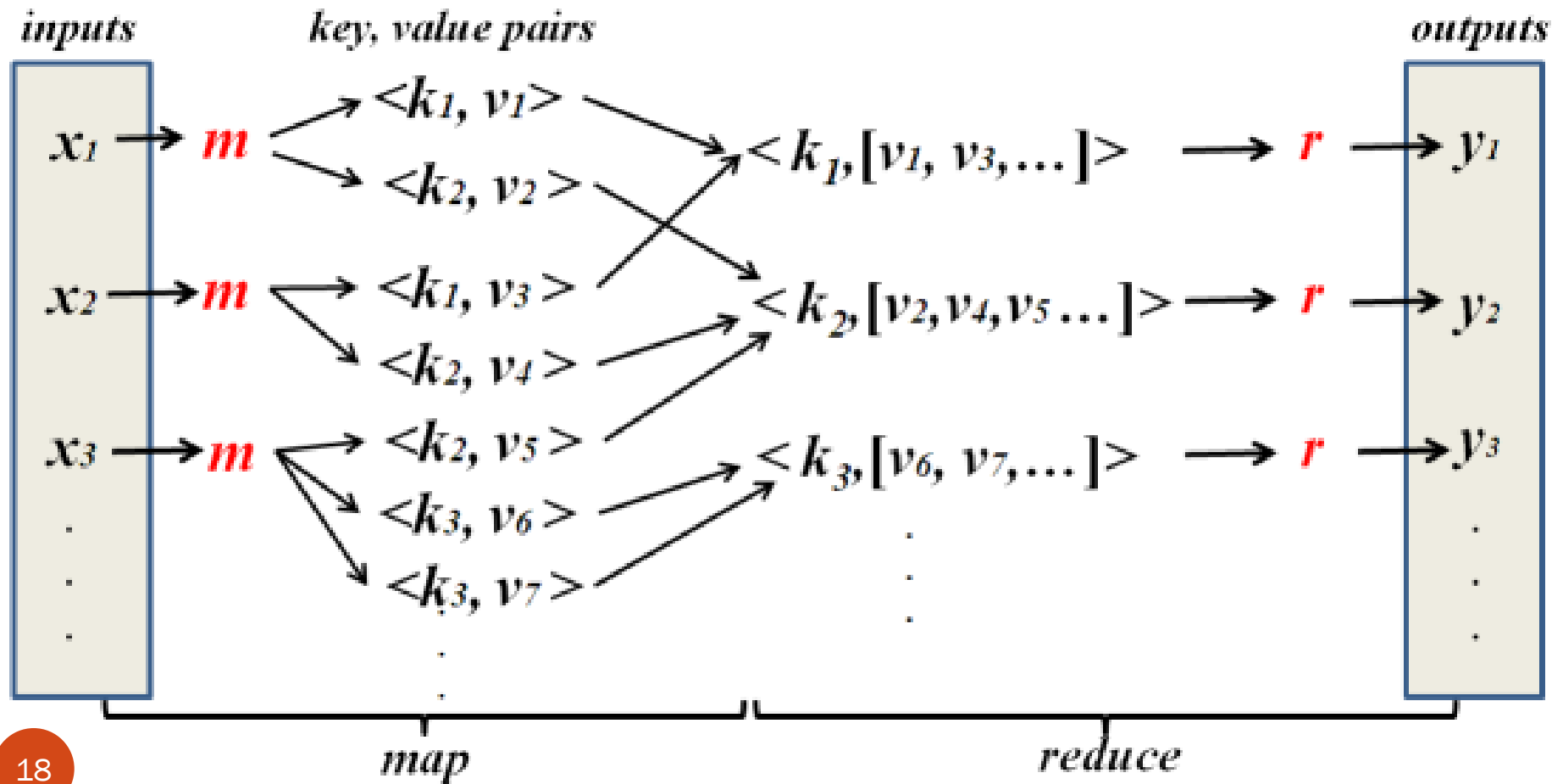
```
function map(String name, String document):  
    // name: document name  
    // document: document contents  
    for each word w in document:  
        emit (w, 1) // emit(key, value)
```


Reduce

- **"Reduce" step:** After map step, the master node collects the answers of all the sub-problems and classifies all values with their keys. Each unique key and its values are sent to a distinct reducer. This means that each reducer processes a distinct key and its values independently from each other.

```
function reduce(String word, Iterator partialCounts):  
  // word: a word  
  // partialCounts: a list of aggregated partial counts  
  sum = 0  
  for each pc in partialCounts:  
    sum += ParseInt(pc)  
  emit (word, sum)
```

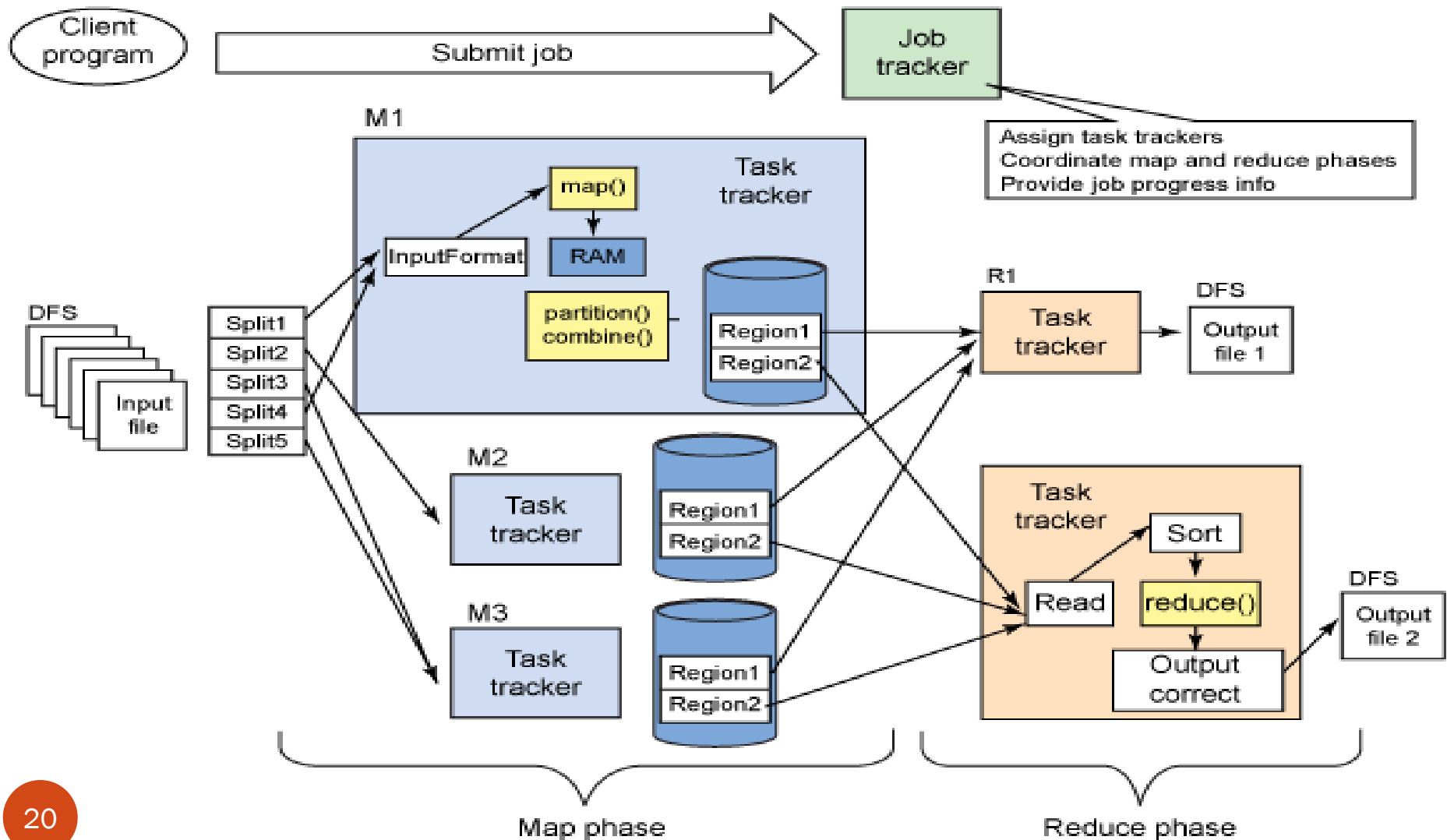
Map-Reduce Overview



Hadoop

- Hadoop is a scalable fault-tolerant distributed system for data storage and processing.
- Core Hadoop has two main components:
 - (1) **HDFS**: Hadoop Distributed File System is a reliable, self-healing, redundant, high-bandwidth, distributed file system optimized for large files.
 - (2) **MapReduce**: Fault-tolerant distributed processing
- Operates on unsuctured and structured data
- Open source under the friendly Apache License

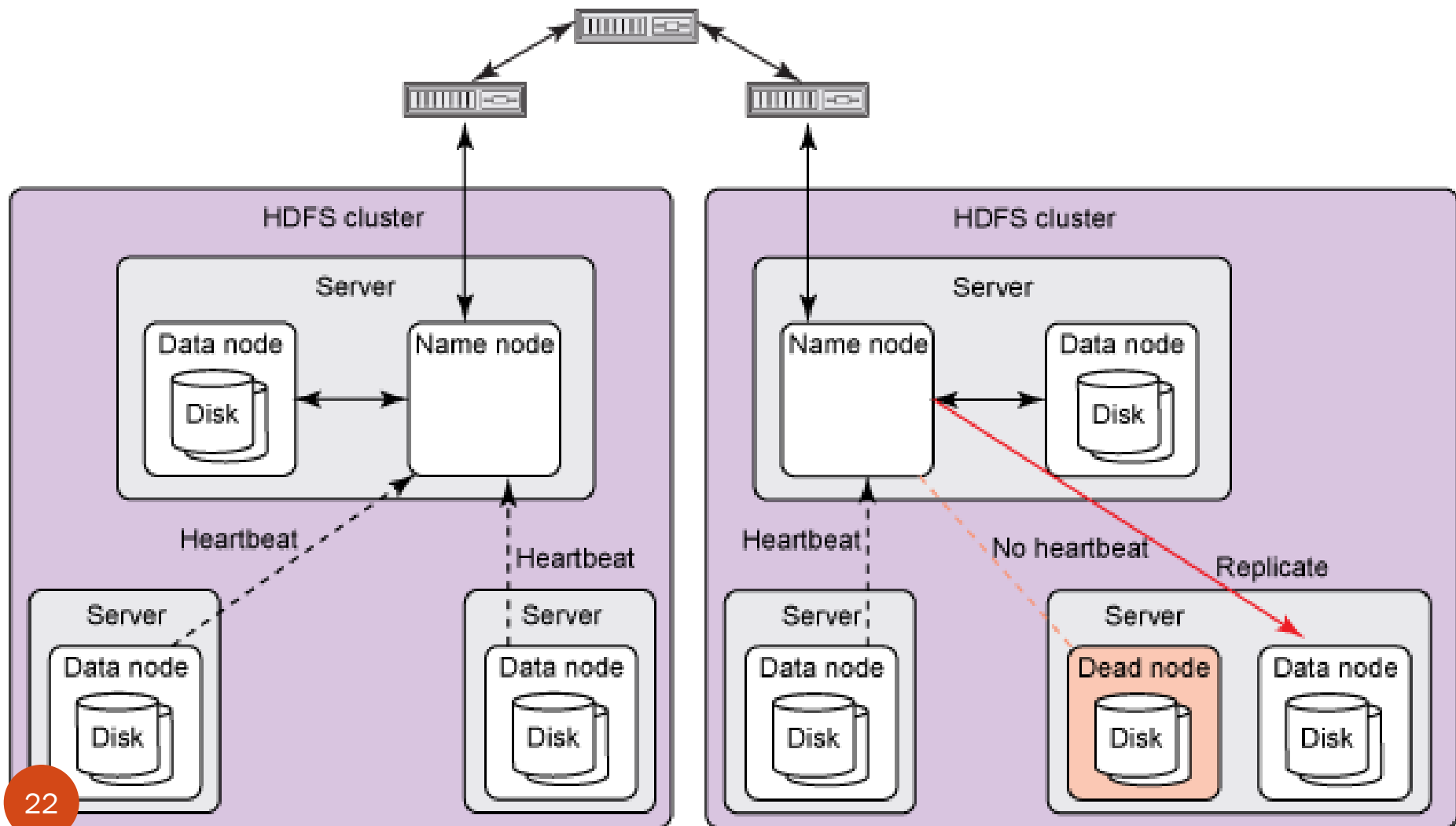
Hadoop Architecture



HDFS

- Inspired by Google File System
- Scalable, distributed, portable file system written in Java for Hadoop framework
 - Primary distributed storage used by Hadoop applications
- HDFS can be part of a Hadoop cluster or can be a stand-alone general purpose distributed file system
- An HDFS cluster primarily consists of
 - NameNode that manages file system metadata
 - DataNode that stores actual data
- Stores very large files in blocks across machines in a large cluster
 - Reliability and fault tolerance ensured by replicating data across multiple hosts

HDFS Architecture



Running Analytical Queries over Offline Big Data

- Hadoop is great for large-data processing!
 - But writing Java programs for everything is verbose and slow
 - Not everyone wants to (or can) write Java code
- Solution: develop higher-level data processing languages
 - Hive: HQL is like SQL
 - Pig: Pig Latin is a bit like Perl

Hive

A system for querying and managing structured data built on top of Hadoop

- Uses Map-Reduce for execution
- HDFS for storage – but any system that implements Hadoop FS API


Key Building Principles:

- Structured data with rich data types (structs, lists and maps)
- Directly query data from different formats (text/binary) and file formats (Flat/Sequence)
- SQL as a familiar programming tool and for standard analytics
- Allow embedded scripts for extensibility and for non standard applications
- Rich MetaData to allow data discovery and for optimization

Hive - Example


▶ **SELECT** foo **FROM** sample **WHERE** ds='2012-02-24';

Create HDFS dir for the output



▶ **INSERT OVERWRITE DIRECTORY** '/tmp/hdfs_out' **SELECT** * **FROM** sample **WHERE** ds='2012-02-24';

Create local dir for the output




▶ **INSERT OVERWRITE LOCAL DIRECTORY** '/tmp/hive-sample-out' **SELECT** * **FROM** sample;


▶ **SELECT** MAX(foo) **FROM** sample;

▶ **SELECT** ds, COUNT(*), SUM(foo) **FROM** sample **GROUP BY** ds;

Hive allows the From clause to come first !!!



Store the results into a table



▶ **FROM** sample s **INSERT OVERWRITE TABLE** bar **SELECT** s.bar, count(*) **WHERE** s.foo > 0 **GROUP BY** s.bar;

Pig

- A platform for analyzing large data sets that consists of a high-level language for **expressing** data analysis programs.
- Compiles down to MapReduce jobs
- Developed by Yahoo!
- Open-source language
- Common design patterns as key words (joins, distinct, counts)
- Data flow analysis
 - A script can map to multiple map-reduce jobs
- Can be interactive mode
 - Issue commands and get results

Pig - Example

Read file from HDFS

The input format (text, tab delimited)

Define run-time schema

```
raw = LOAD 'excite.log' USING PigStorage('\t') AS (user, id, time, query);
clean1 = FILTER raw BY id > 20 AND id < 100;
clean2 = FOREACH clean1 GENERATE
    user, time,
    org.apache.pig.tutorial.sanitize(qu
user_groups = GROUP clean2 BY (user, query);
user_query_counts = FOREACH user_groups
    GENERATE group, COUNT(clean2), MIN(clean2.time);
STORE user_query_counts INTO 'uq_counts.csv' USING PigStorage(',');
```

Filter the rows on predicates

For each row, do some transformation

Grouping of records

Compute aggregation for each group

Store the output in a file

Text, Comma delimited

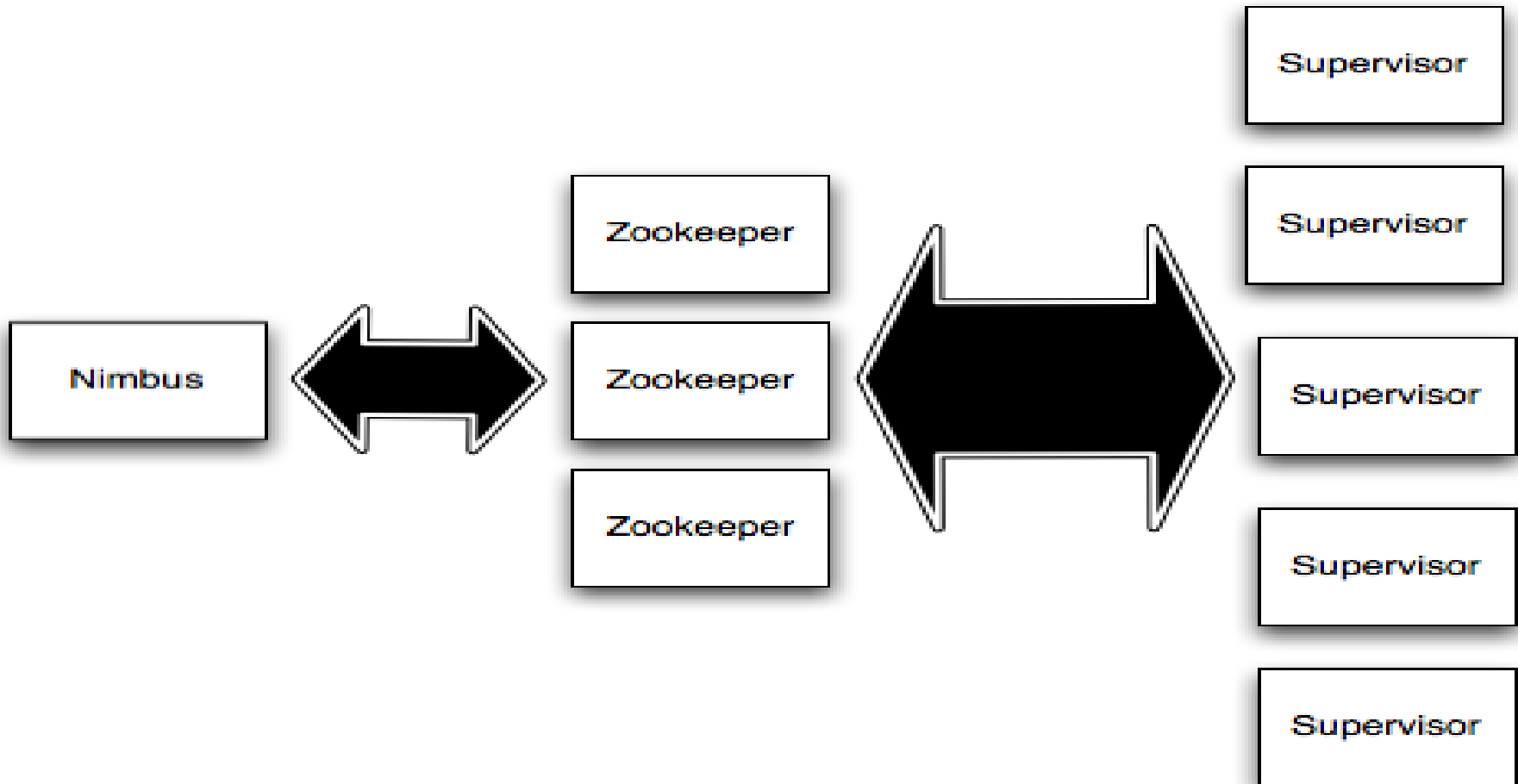
RealTime Data Processing

- In contrast, real time data processing involves a continual input, process and output of data.
- Size of data is not certain at the beginning of the processing.
- Data must be processed in a small time period (or near real time).
- Storm is a most popular open source implementation of realtime data processing.
- Most importantly used by Twitter.

Storm

- Is a highly distributed realtime computation system.
- Provides general primitives to do realtime computation.
- Can be used with any programming language.
- It's scalable and fault-tolerant.

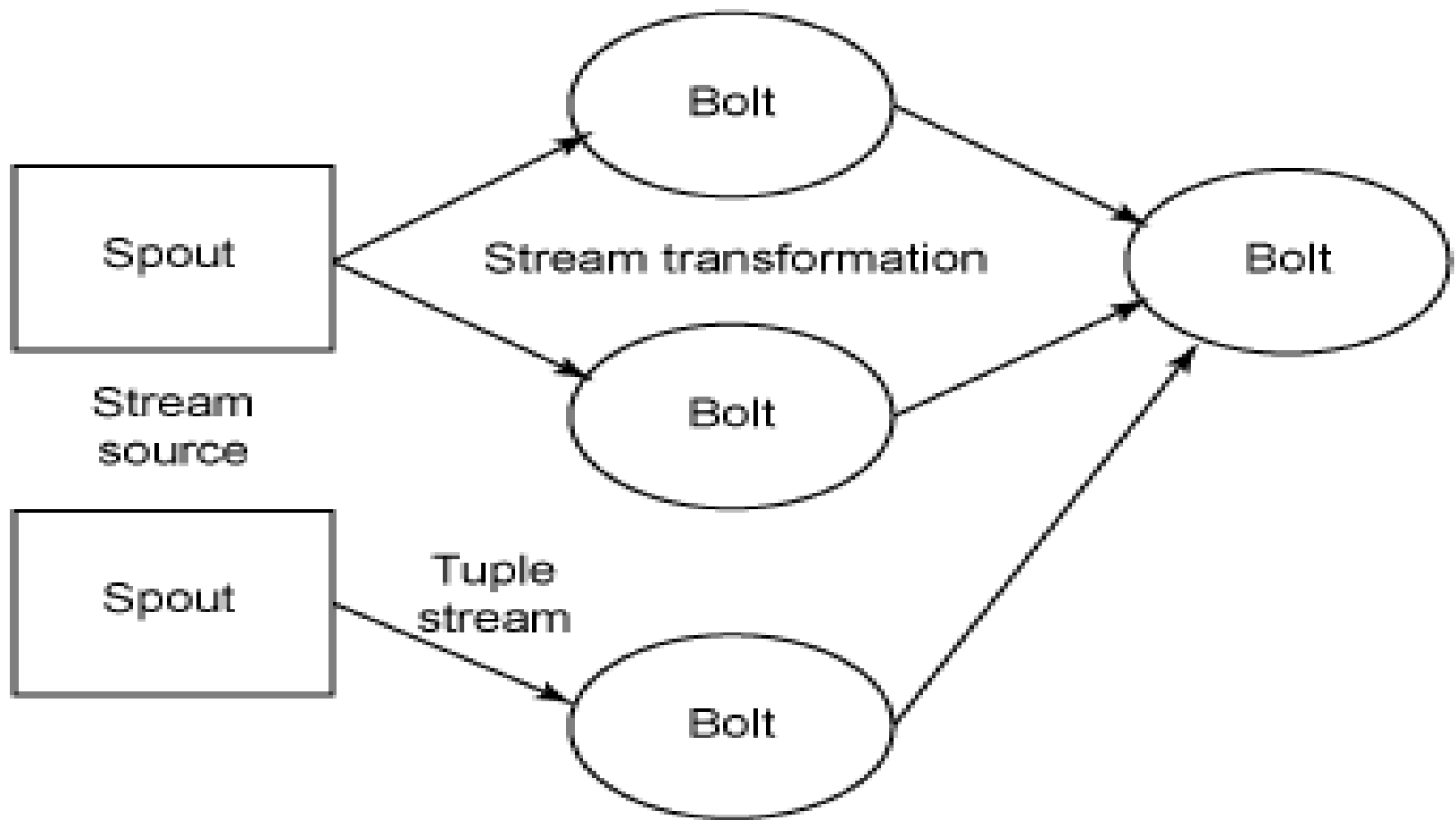
Architecture – General Overview



Architecture - Components

- Nimbus
 - Master node (similar to Hadoop JobTracker)
 - Responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures.
- ZooKeeper
 - Used for cluster coordination between Nimbus and Supervisors.
 - Nimbus and Supervisors are fail-fast and stateless; all state is kept in Zookeeper or on local disk.
- Supervisor
 - Run worker processes.
 - The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it.

Main Concepts – General Overview



Main Concepts - Components

- Streams
 - Unbounded sequence of tuples/datas (input or output)
- Spouts
 - Source of streams
 - Such as reading data from Twitter streaming API
- Bolts
 - Processes input streams, does some processing and possibly emits new streams
- Topologies
 - Network of spouts and bolts

Thanks for Listening

