Hadoop Subprojects

Pig , Hive , HBase , Zookeeper

Hadoop Related Subprojects

- Pig
  - High-level language for data analysis
- Hive
  - SQL-like Query language and Metastore
- HBase
  - Table storage for semi-structured data
- Storm
  - Streaming data realtime computation

Pig

Original Slides by

Matei Zaharia

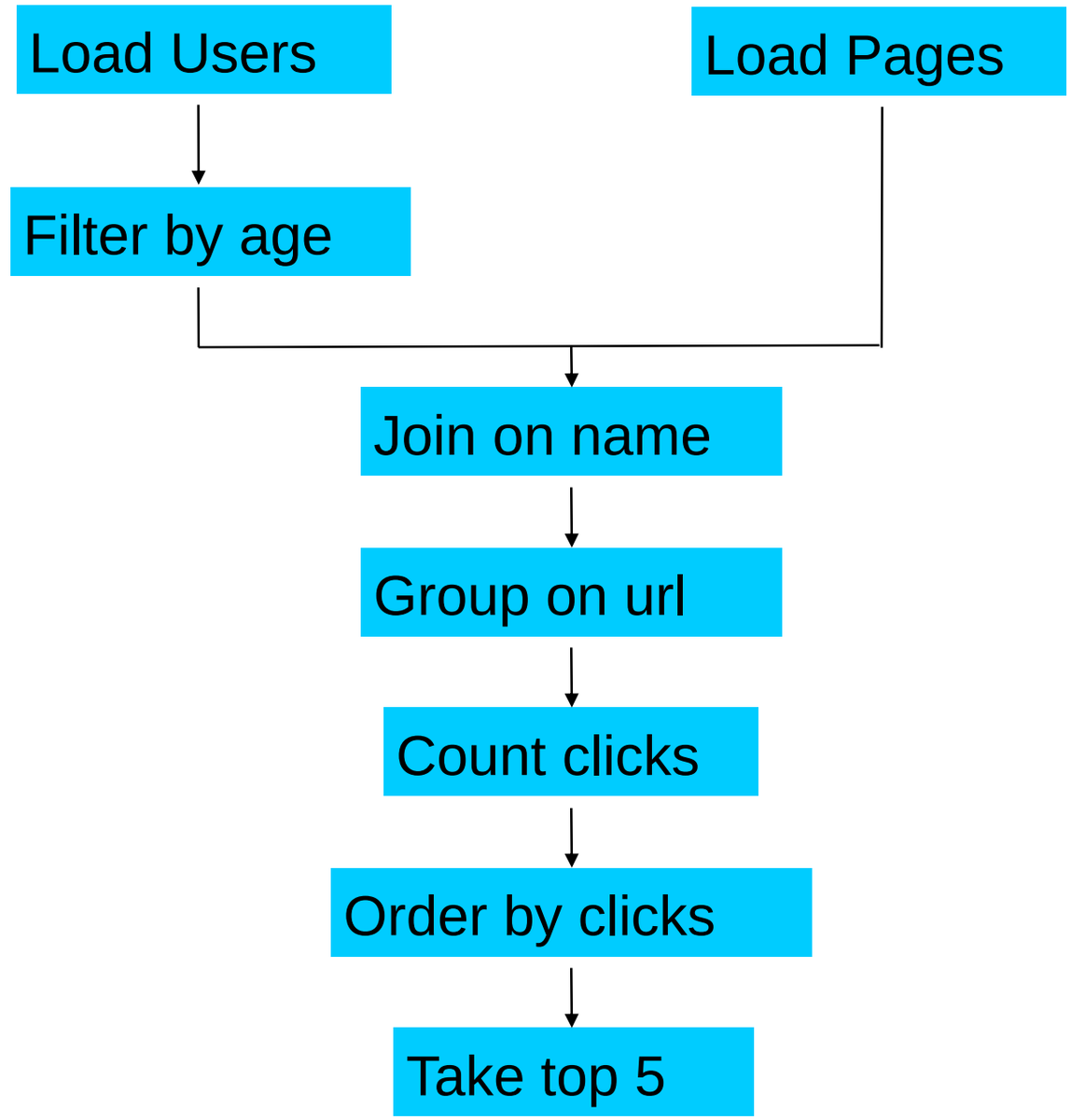
UC Berkeley RAD Lab

Pig

- Started at Yahoo! Research
- Now runs about 30% of Yahoo!'s jobs
- Features
  - Expresses sequences of MapReduce jobs
  - Data model: nested "bags" of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc.)
  - Easy to plug in Java functions

An Example Problem

- Suppose you have user data in a file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25



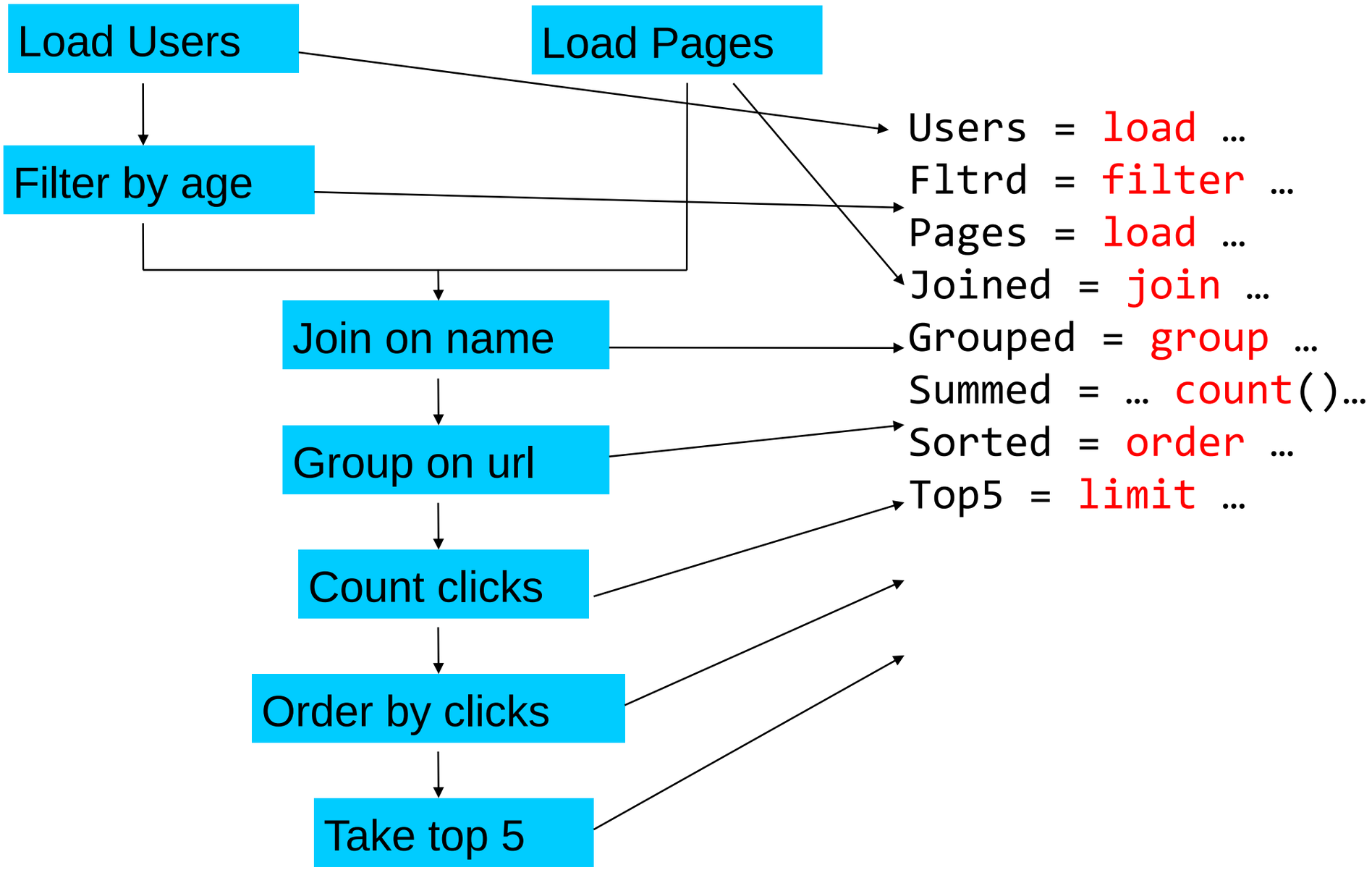
In MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
                LongWritable key,
                Iterator<Text> iter,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

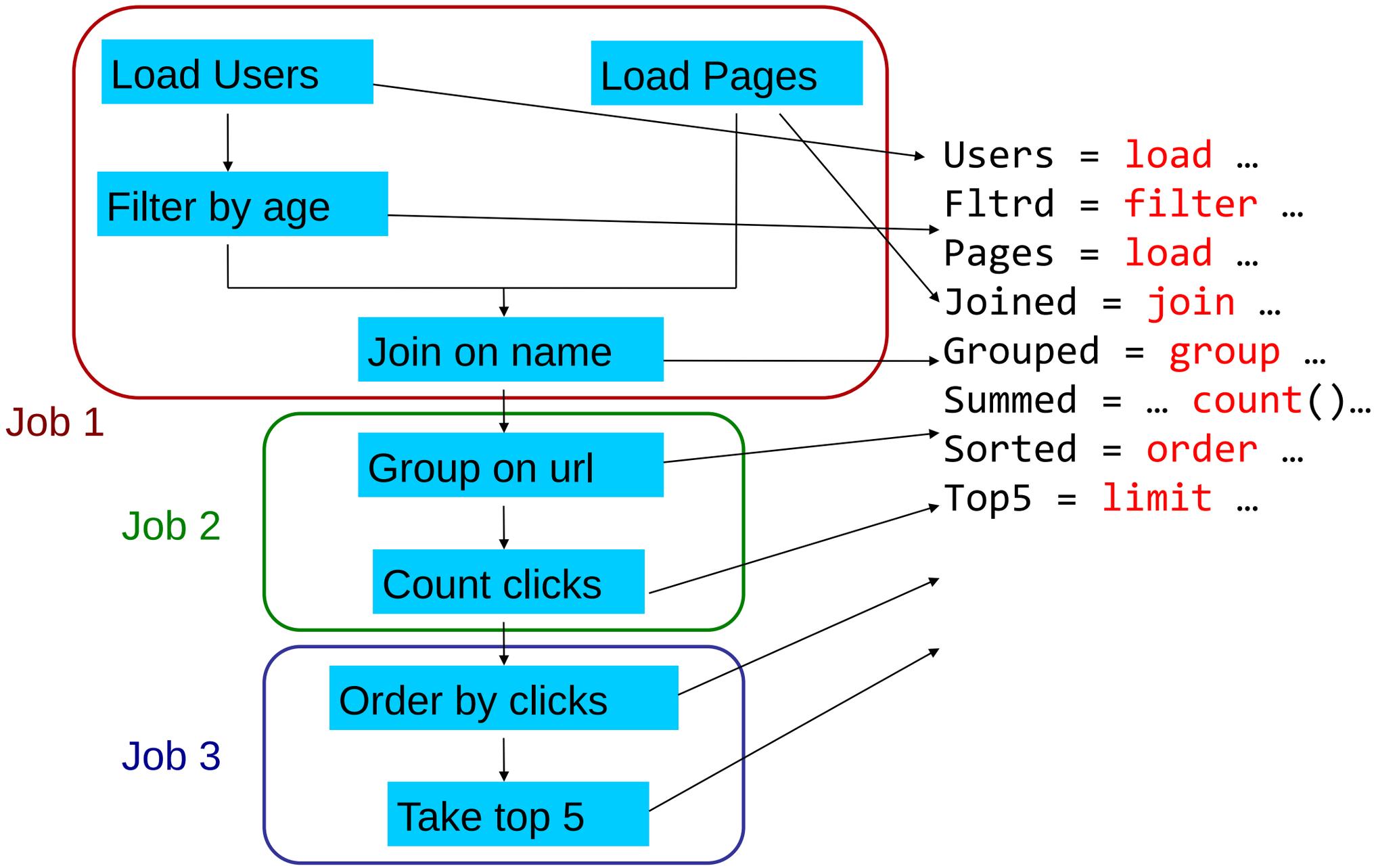
In Pig Latin

```
Users = load ‘users’ as (name, age);
Filtered = filter Users by age >= 18 and age <=
   25;
Pages = load ‘pages’ as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
                    count(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;
store Top5 into ‘top5sites’;
```

Ease of Translation



Ease of Translation



Hive

Original Slides by

Matei Zaharia

UC Berkeley RAD Lab

Hive

- Developed at Facebook
- Used for majority of Facebook jobs
- “Relational database” built on Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HiveQL)
  - Can call Hadoop Streaming scripts from HiveQL
  - Supports table partitioning, clustering, complex data types, some optimizations

Creating a Hive Table

```
CREATE TABLE page_views(viewTime INT, userid BIGINT,
                        page_url STRING, referrer_url STRING,
                        ip STRING COMMENT 'User IP address')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

- Partitioning breaks table into separate files for each (dt, country) pair

  Ex: /hive/page_view/dt=2008-06-08,country=USA

  /hive/page_view/dt=2008-06-08,country=CA

A Simple Query

- Find all page views coming from xyz.com on March 31st:

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31'
AND page_views.referrer_url like '%xyz.com';
```

- Hive only reads partition `2008-03-01,*` instead of scanning entire table

Aggregation and Joins

- Count users who visited each page by gender:

```
SELECT pv.page_url, u.gender, COUNT(DISTINCT u.id)
FROM page_views pv JOIN user u ON (pv.userid = u.id)
GROUP BY pv.page_url, u.gender
WHERE pv.date = '2008-03-03';
```

- Sample output:

page_url	gender	count(userid)
home.php	MALE	12,141,412
home.php	FEMALE	15,431,579
photo.php	MALE	23,941,451
photo.php	FEMALE	21,231,314

HBase

Original Slides by

Tom White

Lexeme Ltd.

HBase - What?

- Modeled on Google's Bigtable
- Row/column store
- Billions of rows/millions on columns
- Column-oriented - nulls are free
- Untyped - stores byte[]

HBase - Data Model

Row	Timestamp	Column family: animal:		Column family repairs:
		animal:type	animal:size	repairs:cost
enclosure1	t2	zebra		1000 EUR
	t1	lion	big	
enclosure2	…	…	…	…

HBase - Data Storage

Column family animal:

(enclosure1, t2, animal:type)	zebra
(enclosure1, t1, animal:size)	big
(enclosure1, t1, animal:type)	lion

Column family repairs:

(enclosure1, t1, repairs:cost)	1000 EUR

HBase - Code

```
HTable table = …
Text row = new Text(“enclosure1”);
Text col1 = new Text(“animal:type”);
Text col2 = new Text(“animal:size”);
BatchUpdate update = new BatchUpdate(row);
update.put(col1, “lion”.getBytes(“UTF-8”));
update.put(col2, “big”.getBytes(“UTF-8));
table.commit(update);

update = new BatchUpdate(row);
update.put(col1, “zebra”.getBytes(“UTF-8”));
table.commit(update);
```

HBase - Querying

- Retrieve a cell

  Cell = table.getRow(“enclosure1”).getColumn(“animal:type”).getValue();

- Retrieve a row

  RowResult = table.getRow(“enclosure1”);

- Scan through a range of rows

  Scanner s = table.getScanner(new String[] { “animal:type” });

Storm

Original Slides by

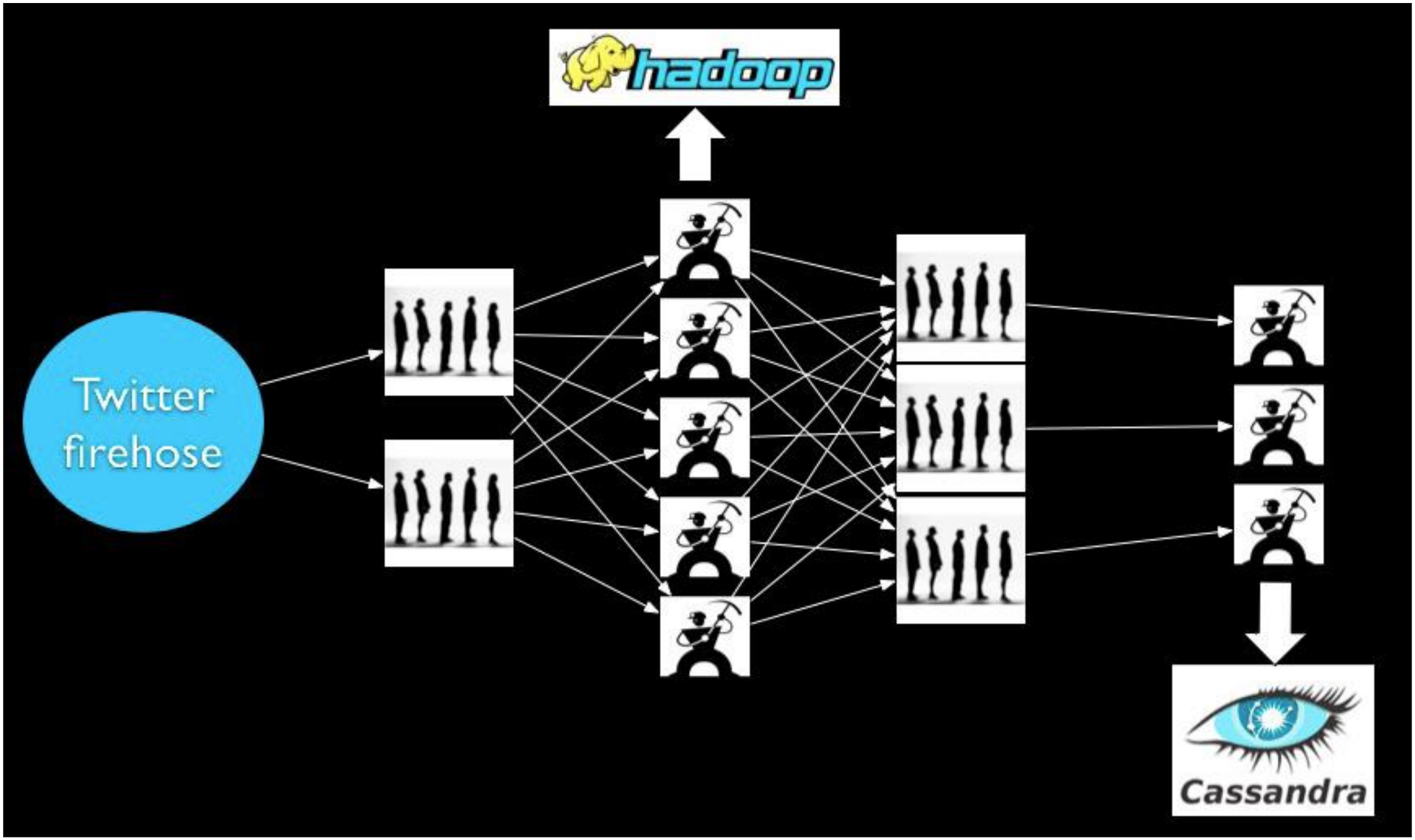
Nathan Marz

Twitter

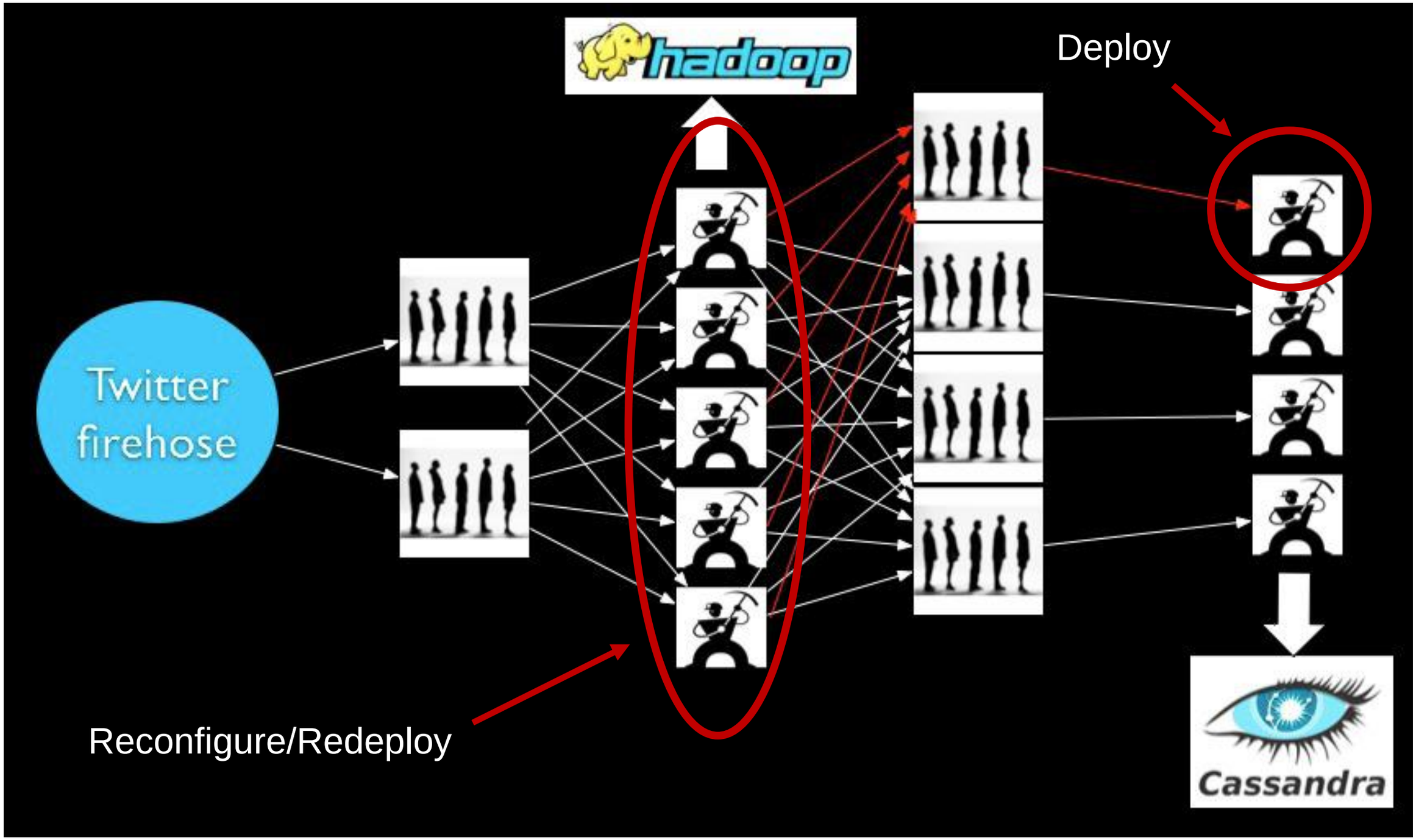
Storm

- Developed by BackType which was acquired by Twitter
- Lots of tools for data (i.e. batch) processing
  - Hadoop, Pig, HBase, Hive, …
- None of them are realtime systems which is becoming a real requirement for businesses
- Storm provides realtime computation
  - Scalable
  - Guarantees no data loss
  - Extremely robust and fault-tolerant
  - Programming language agnostic

Before Storm



Before Storm – Adding a worker



Problems

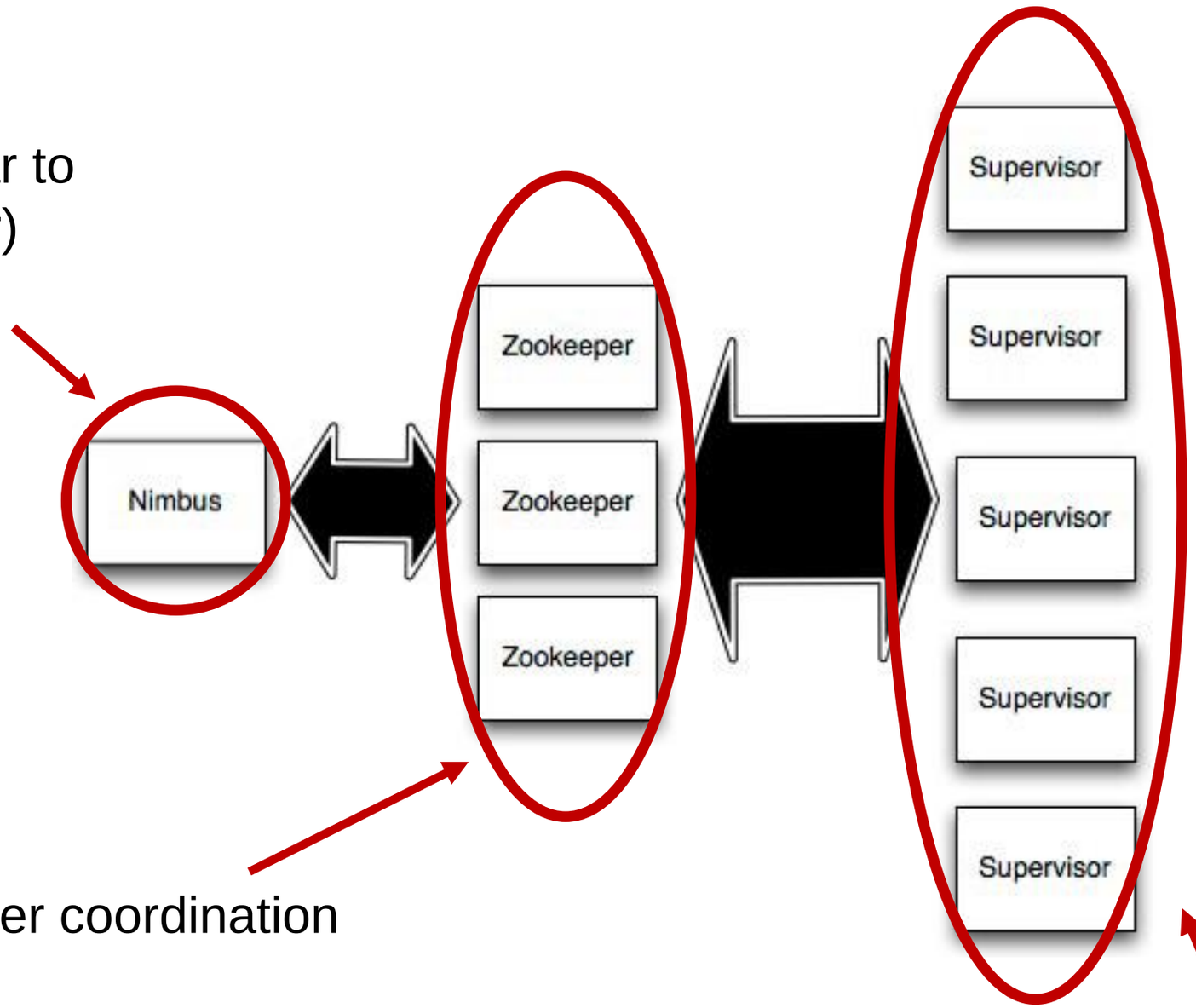
- Scaling is painful
- Poor fault-tolerance
- Coding is tedious

What we want

- Guaranteed data processing
- Horizontal scalability
- Fault-tolerance
- No intermediate message brokers!
- Higher level abstraction than message passing
- “Just works” !!

Storm Cluster

Master node (similar to Hadoop JobTracker)



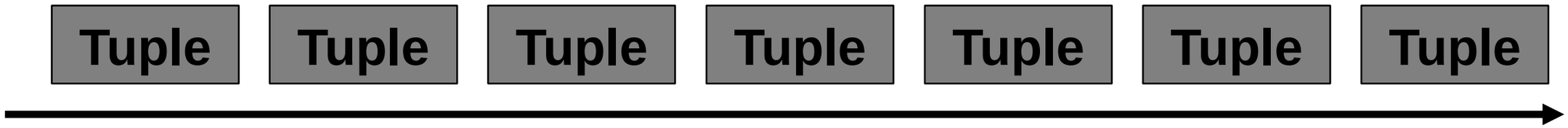
Used for cluster coordination

Run worker processes

Concepts

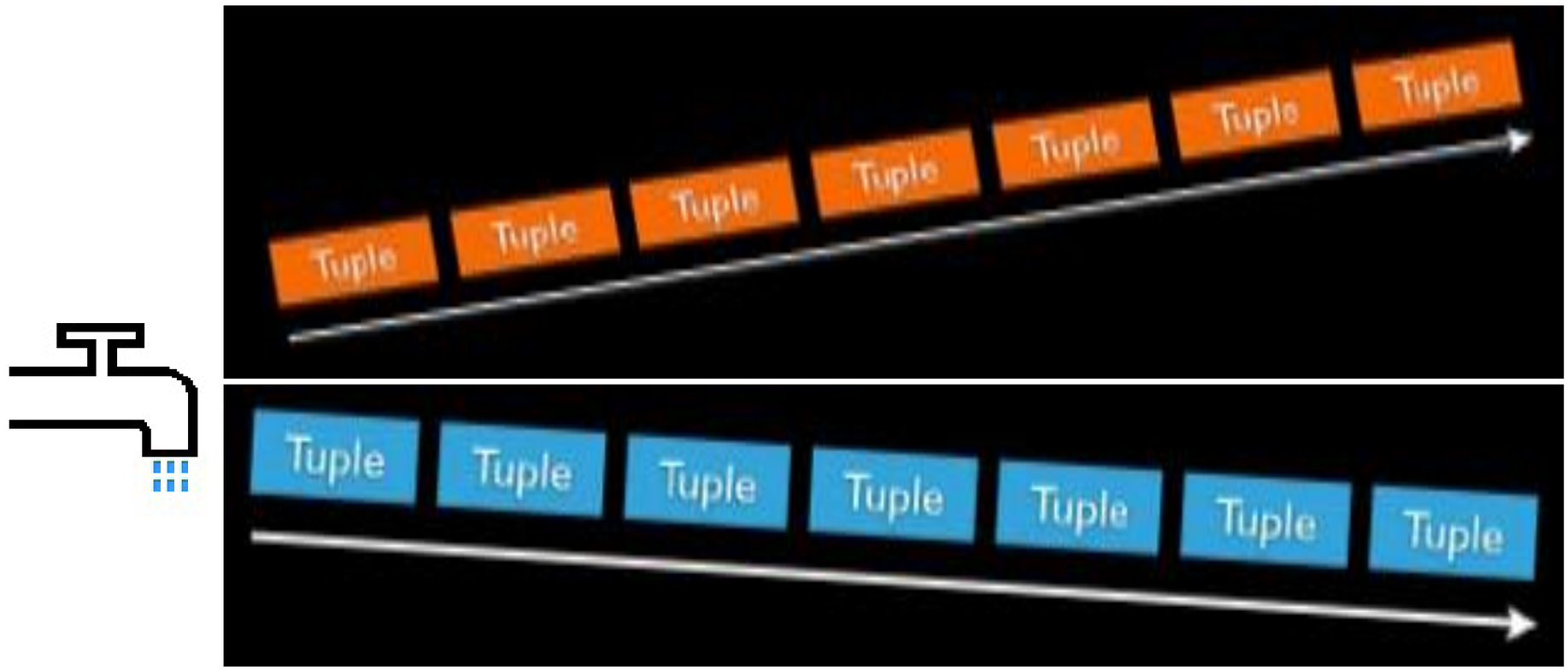
- Streams
- Spouts
- Bolts
- Topologies

Streams



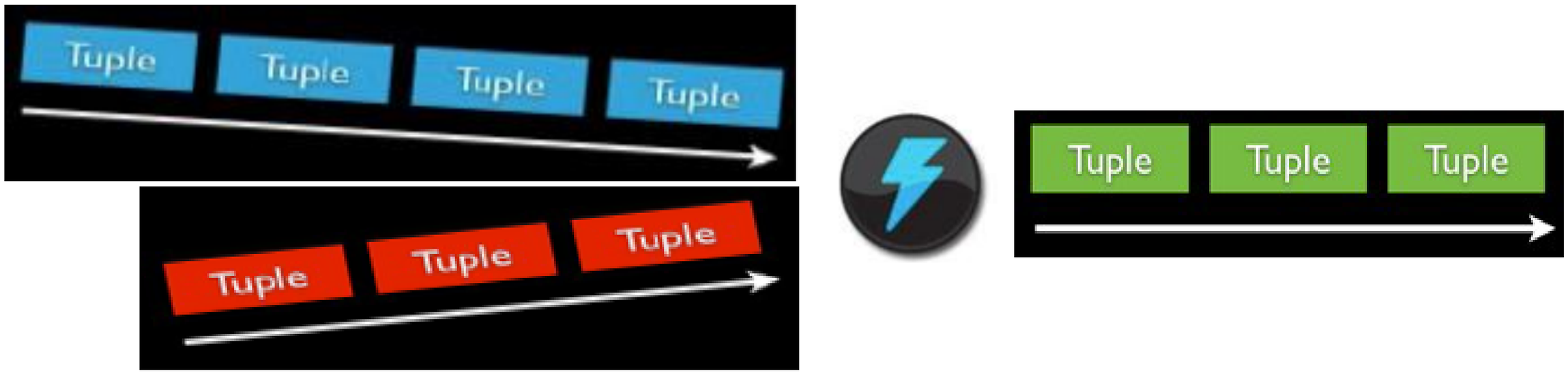
Unbounded sequence of tuples

Spouts



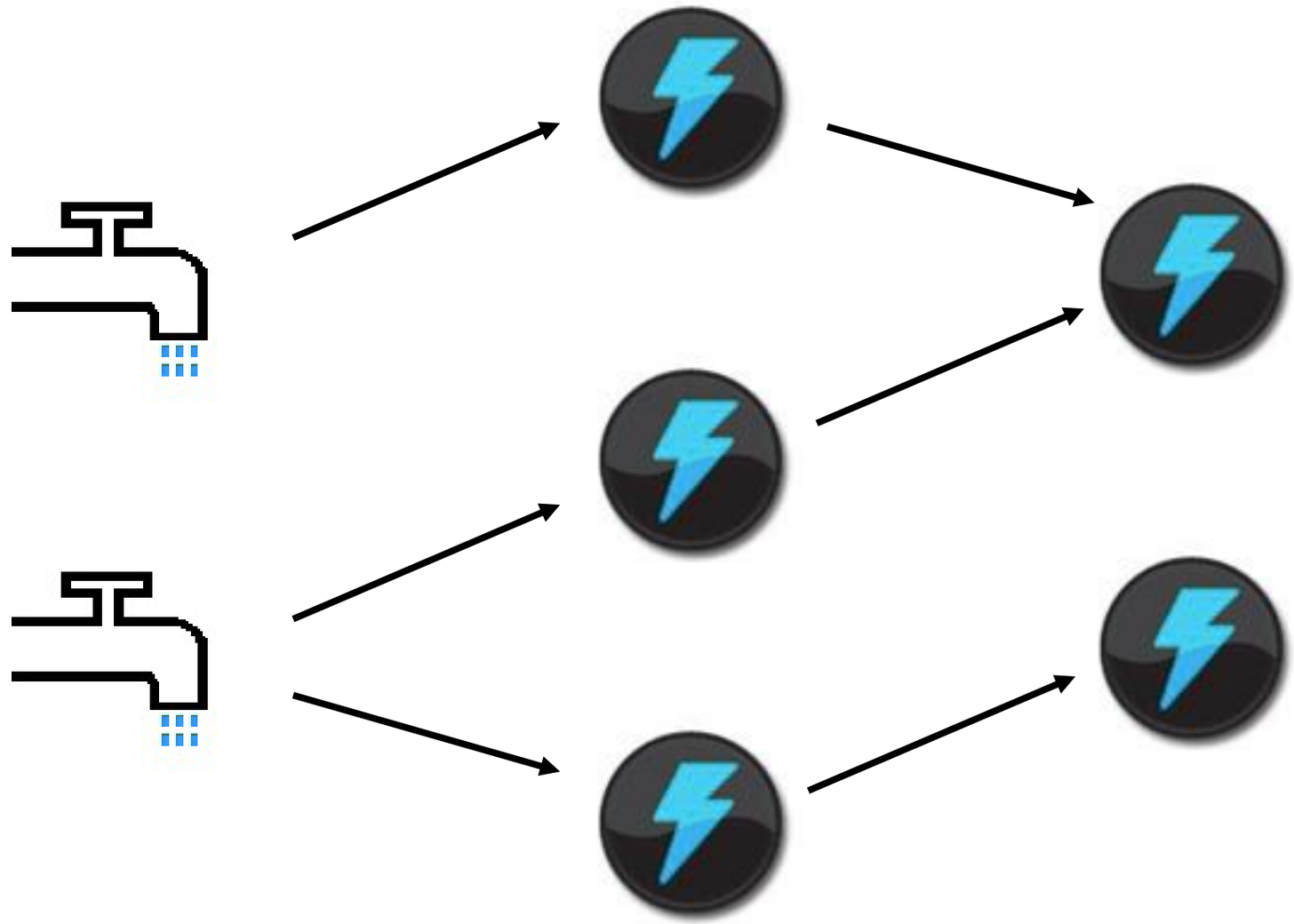
Source of streams

Bolts



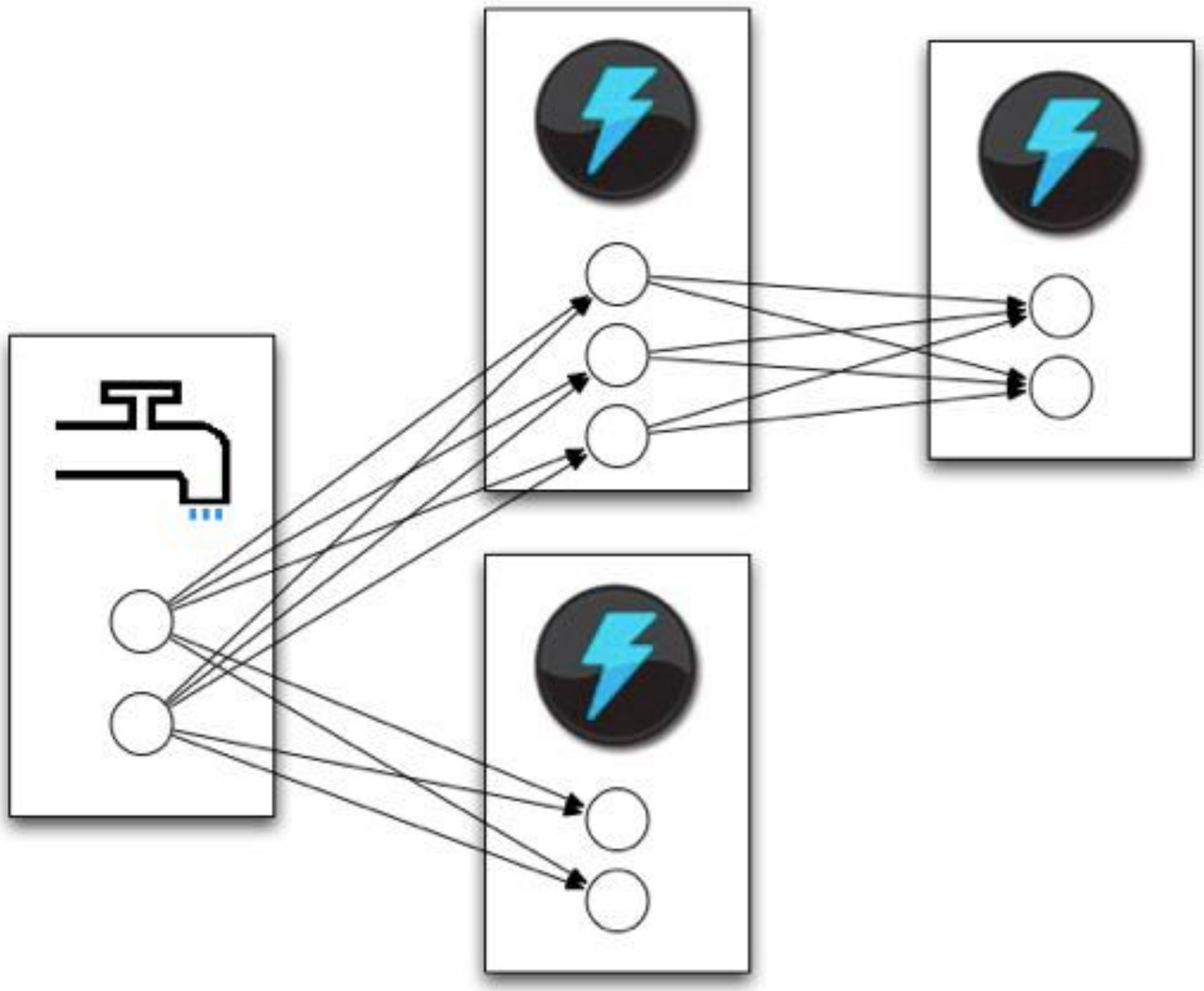
Processes input streams and produces new streams:
Can implement functions such as filters, aggregation, join, etc

Topology



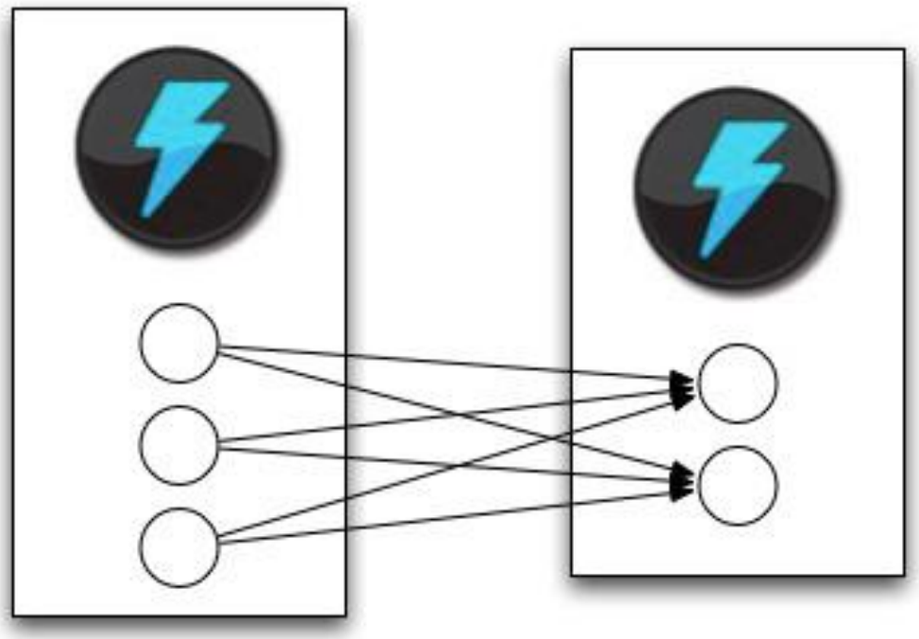
Network of spouts and bolts

Topology



Spouts and bolts execute as
many tasks across the cluster

Stream Grouping

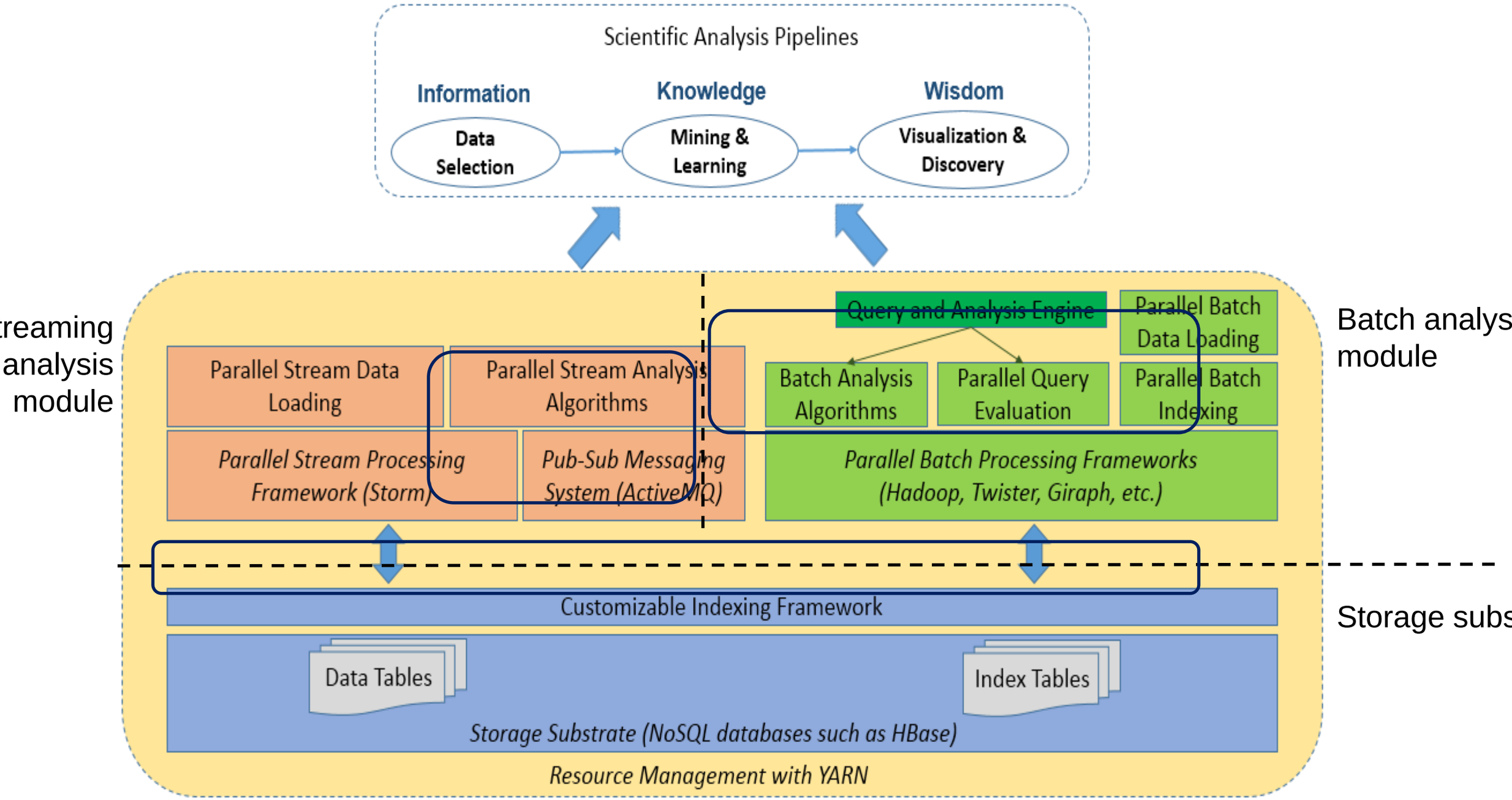


When a tuple is emitted which task does it go to?

Stream Grouping

- **Shuffle grouping**: pick a random task
- **Fields grouping**: consistent hashing on a subset of tuple fields
- **All grouping**: send to all tasks
- **Global grouping**: pick task with lowest id

Batch and Streaming Analysis for Social Media Data



Streaming Analysis

- Non-trivial parallel stream processing algorithm with novel global synchronization and cluster-delta data transfer to achieve scalability

- Clustering of social media streams: real-time processing of 10% Twitter (“Gardenhose”)
- Recent progress in learning **data representations** and **similarity metrics**
- High-dimensional vectors: textual and network information
- Expensive similarity computation: 43.4 hours to cluster 1 hour’s data with sequential algorithm
  - Online K-Means with sliding time window and outlier detection
  - Group tweets as **protomemes**: hashtags, mentions, URLs, and phrases

Xiaoming Gao, Emilio Ferrara, Judy Qiu. Parallel Clustering of High-Dimensional Social Media Data Streams. To appear at 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2015).

Social media data – an example data record


```
{
    "text":"RT @sengineland: My Single Best... ",
    "created_at":"Fri Apr 15 23:37:26 +0000 2011",
    "retweet_count":0,
    "id_str":"5903764764925952l",
          "entities":{
        "user_mentions":[{
                "screen_name":"sengineland",
                "id_str":"1059801",
                "name":"Search Engine Land"
            }],
        "hashtags":[],
        "urls":[{
                "url":"http:\/\/selnd.com\/e2QPS1",
                "expanded_url":null
            }]},
    "user":{
        "created_at":"Sat Jan 22 18:39:46 +0000 2011",
        "friends_count":63,
        "id_str":"241622902",
        ...},
    "retweeted_status":{
        "text":"My Single Best... ",
        "created_at":"Fri Apr 15 21:40:10 +0000 2011",
        "id_str":"59008136320786432",
        ...},
    ...
}
```

Sequential clustering algorithm

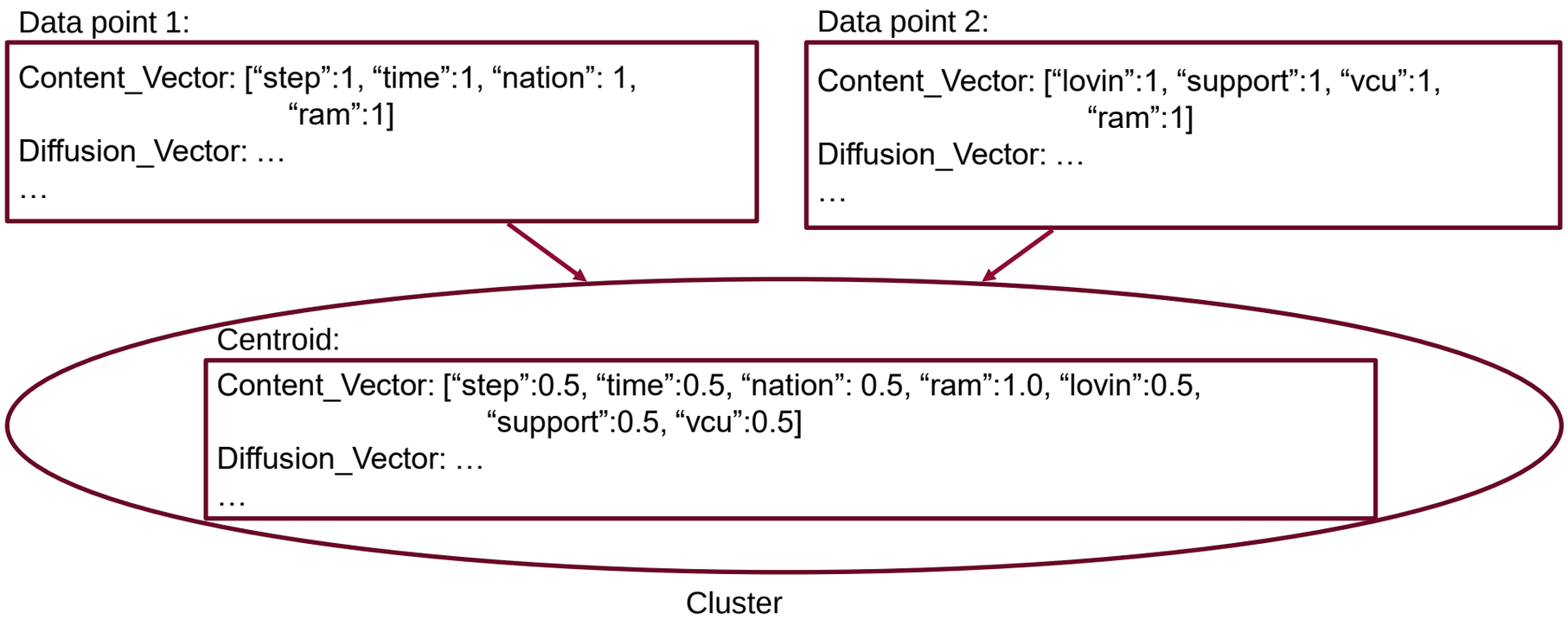
- Final step statistics for a sequential run over 6 minutes data:

Time Step Length (s)	Total Length of Centroids' Content Vector	Similarity Compute time (s)	Centroids Update Time (s)
10	47749	33.305	0.068
20	76146	78.778	0.113
30	128521	209.013	0.213

120 clusters, time window length: 6 steps, outlier: 2 standard deviation

Parallelization with Storm - challenges

- DAG organization of parallel workers: hard to synchronize cluster information
- Sparsity of high-dimensional vectors make any synchronization expensive



- Cluster-delta synchronization strategy reduces message traffic and synchronization overhead

Solution – enhanced Apache Storm topology