POLITECNICO
MILANO 1863

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

# HADOOP Subprojects

Marco Brambilla

marco.brambilla@polimi.it

@marcobrambi

# Hadoop Related Subprojects

Pig

- High-level language for data analysis

HBase

- Table storage for semi-structured data

Hive

- SQL-like Query language and Metastore

# HBase

# HBase - What?

Modeled on Google's Bigtable
Key-valued 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” } );
```

# Pig

# Pig

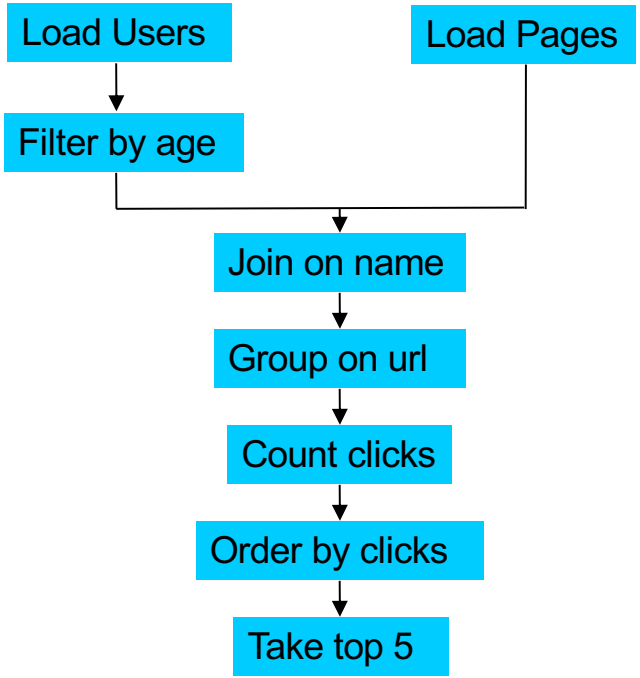
Started at Yahoo! Research

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



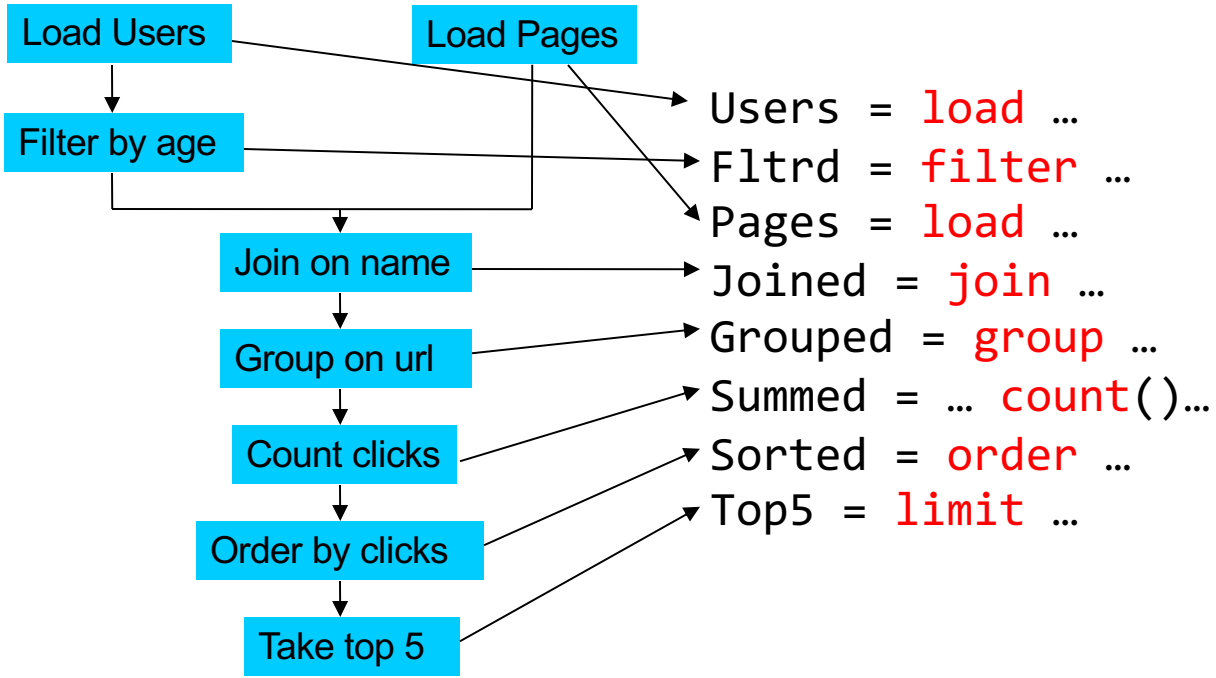
# In MapReduce

```
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();
    }
}
```

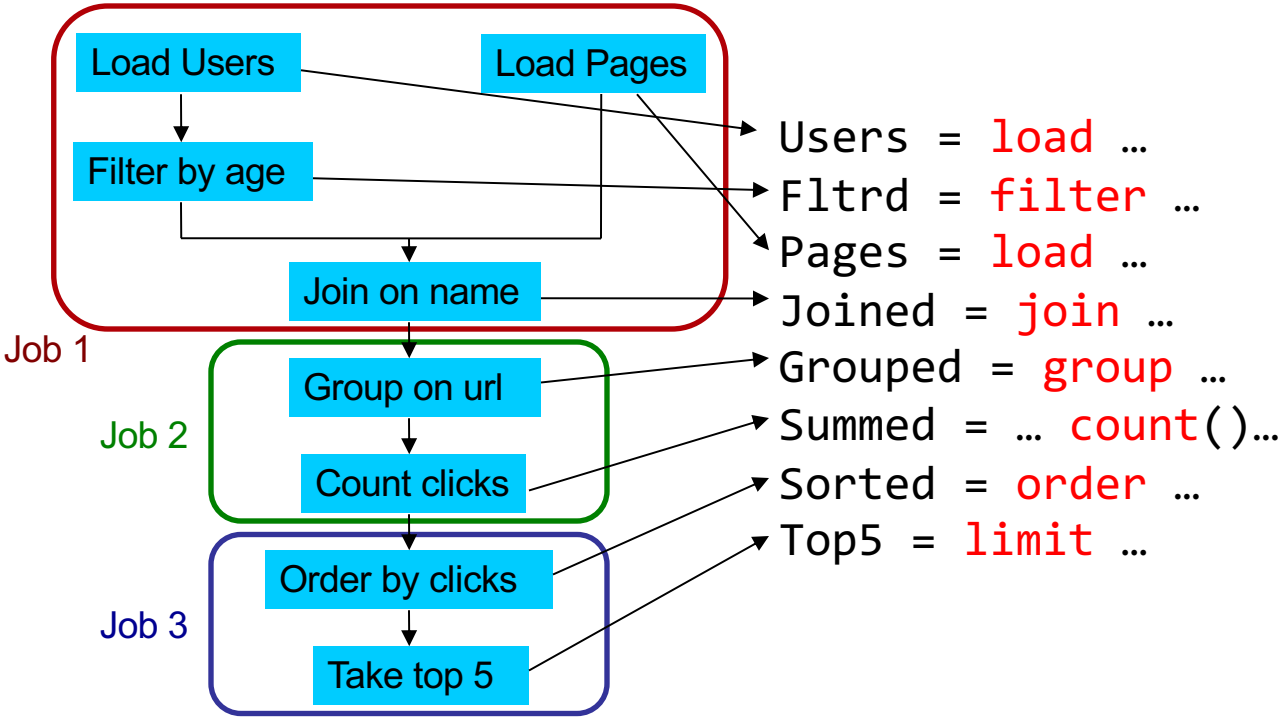
# In Pig

```
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

# Hive

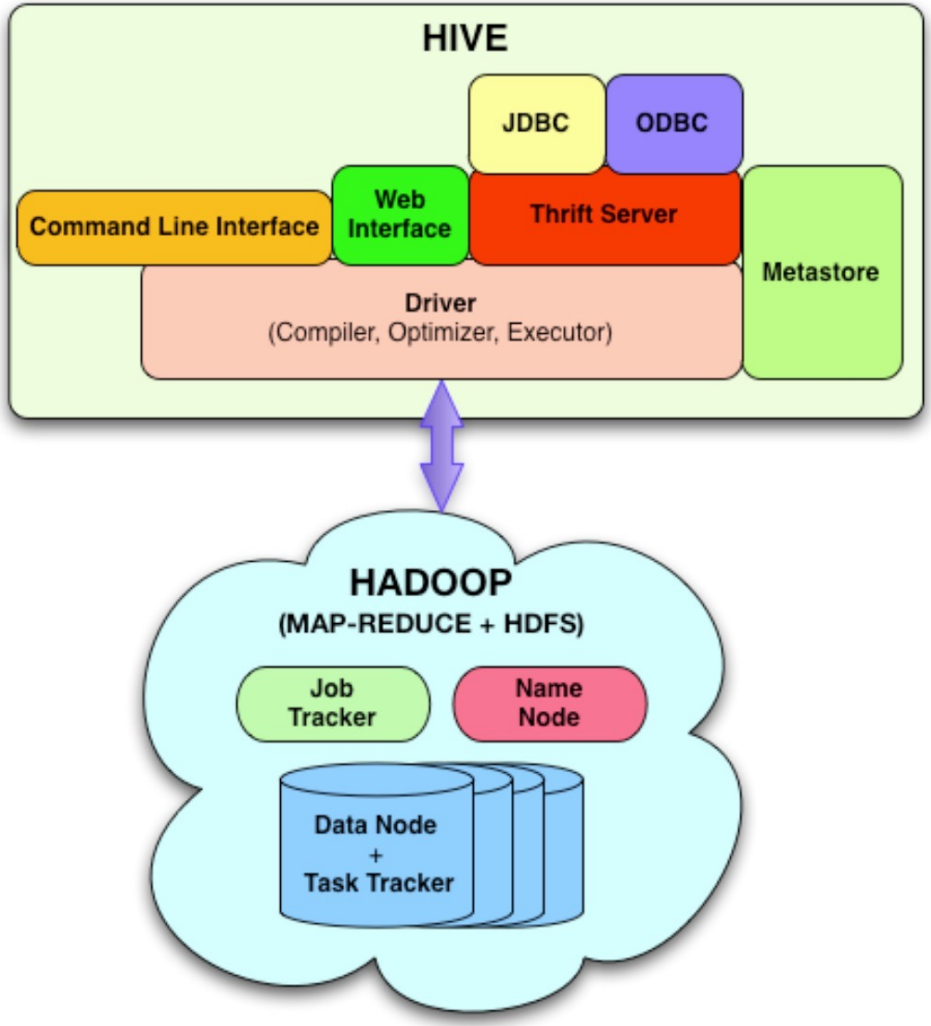
Developed at Facebook

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

# Architecture



# Data Model

## Tables

– Typed columns (int, float, string, date, boolean)
– Also, array/map/struct for JSON-like data

## Partitions

– e.g., to range-partition tables by date

## Buckets

– Hash partitions within ranges (useful for sampling, join optimization)

# Storage

Warehouse directory in HDFS

– e.g., /user/hive/warehouse

- Table row data stored in subdirectories of warehouse
- Partitions form subdirectories of table directories

- Actual data stored in flat files

– Control char-delimited text, or SequenceFiles

– With custom SerDe, can use arbitrary format

# 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 |

# Hive CLI

List tables:
– hive> show tables;

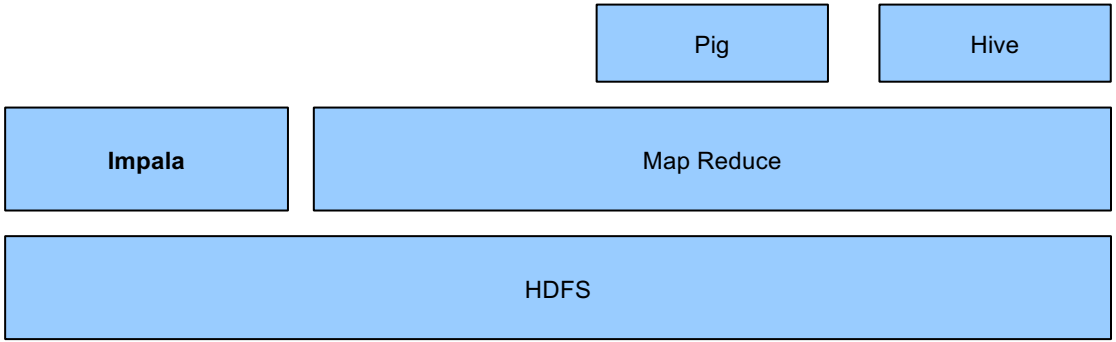
Describe a table:
– hive> describe <tablename>;

More information:
– hive> describe extended <tablename>;

# Impala

# What is impala

- Massive parallel processing (MPP) database engine, developed by Cloudera.
- Integrated into Hadoop stack on the same level as MapReduce, and not above it (as Hive and Pig)



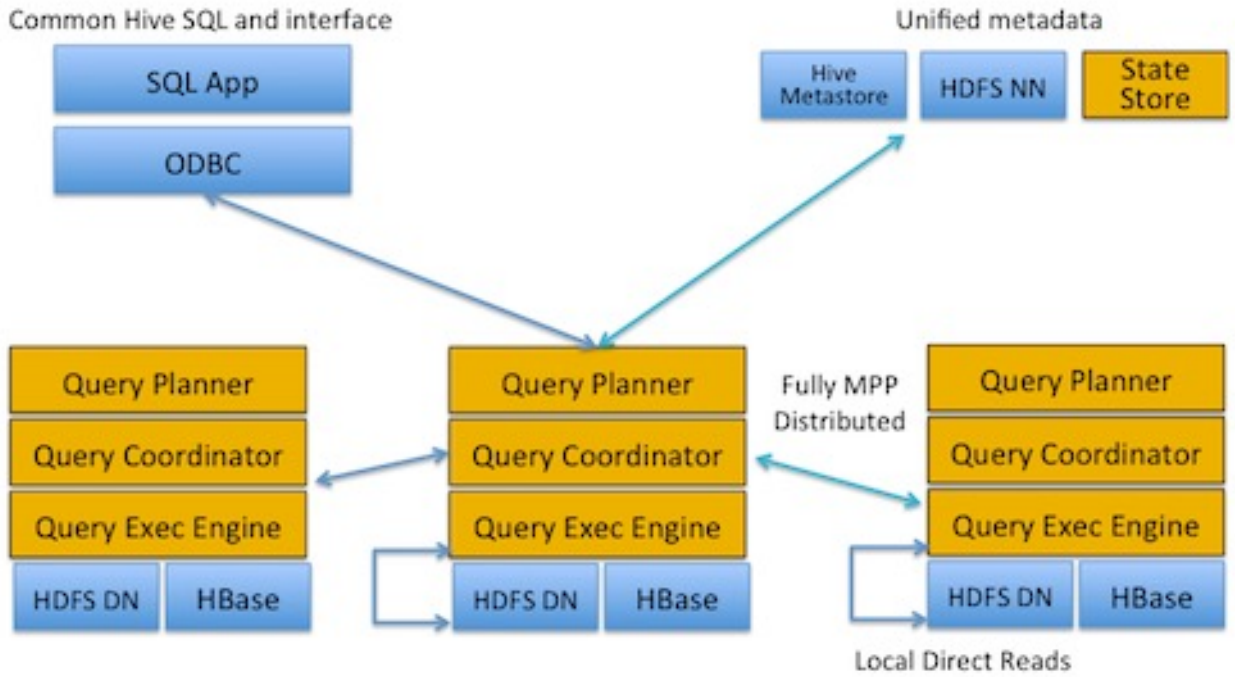
# Why impala

- Data has a gravity
- Today a lot of data live in HDFS
- It is not practical to move big data
- It is practical to bring engine to the data
- In the same time – MapReduce is not must
- Impala process data in Hadoop cluster **without** using MapReduce

# MapReduce bypass

- Several other modern Database engines also realized the opportunity to bypass MapReduce but work right with HDFS.
- They takes various approaches.


# Impala architecture



# Impala – Hive

Impala supports multiple formats. It is kind of schema on read.

- Impala shares metastore with Hive, which enables very simple adoption
- Internally Impala have well defined way to add new formats

# Impala – unique things

- Impala “format adapters”, called scanners have **predicate pushdown capability**.
- Open source MPP engine
- Runs hundreds of CPU cores in one query **efficiently** without expensive license.
- Hive give us the same but not efficiently.

# Impala – Hive

- Hive is doing things Impala can not do yet, like joins between several big tables.
- Hive has convenient java UDF, while impala has not
- Impala does not have inter-query fault tolerance.
- In the same time – MapReduce is not good framework for the database engine
- Always faster then Hive at least 10 times
- Impala in the cloud is not elastic

# Impala – Data Formats

- There are scanners for the following types:
- RCFile
- Parquet (native dremel format)
- CSV
- AVRO
- Sequence File

# Storm

Marco Brambilla.

# Storm

Developed by BackType which was acquired by Twitter

Lots of tools for data (i.e. batch) processing

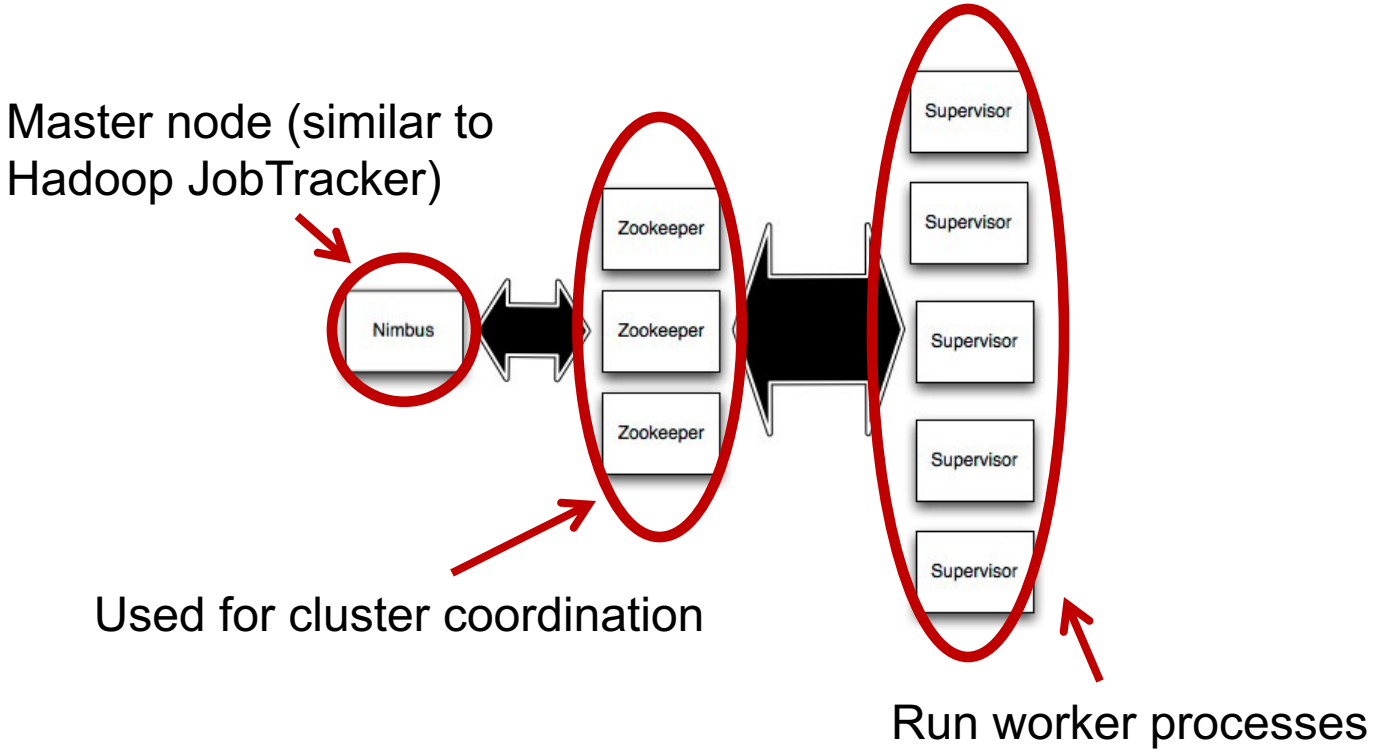
- Hadoop, Pig, HBase, Hive, ...

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

# Storm Cluster



# Concepts

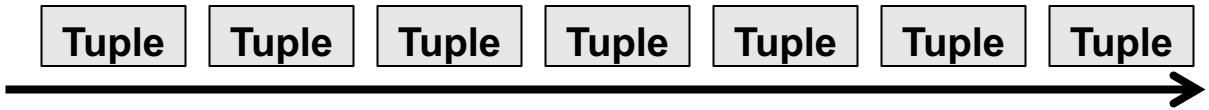
Streams

Spouts

Bolts

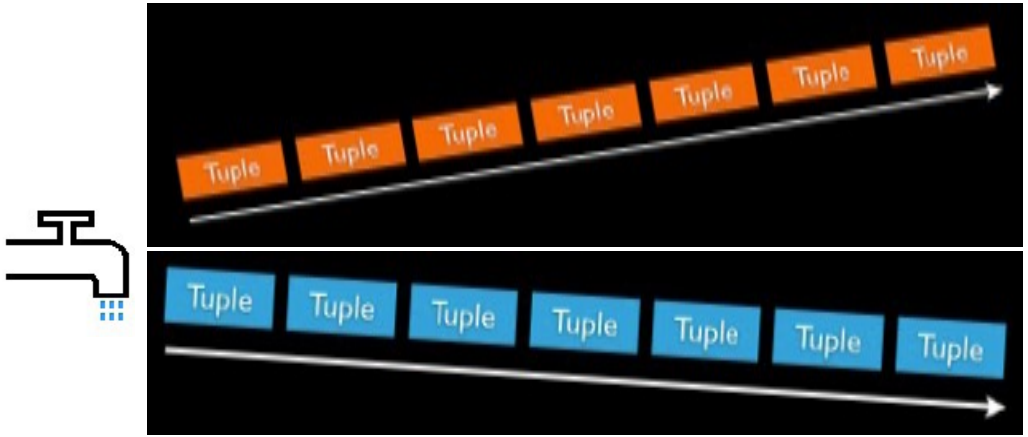
Topologies

# Streams



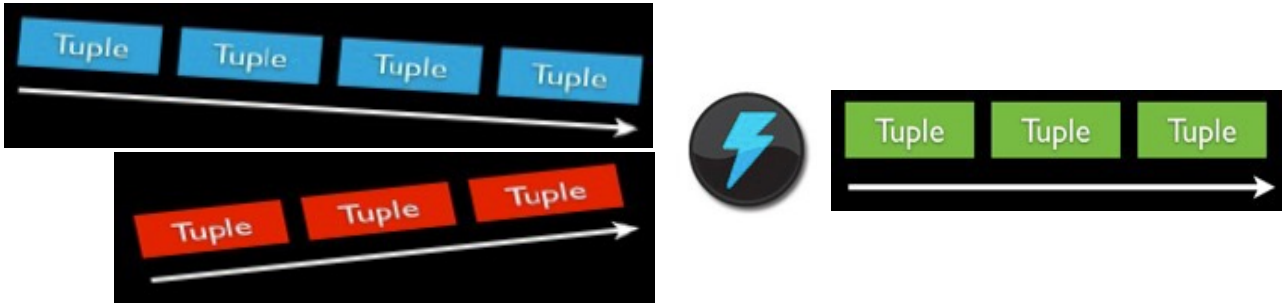

Unbounded sequence of tuples

# Spouts



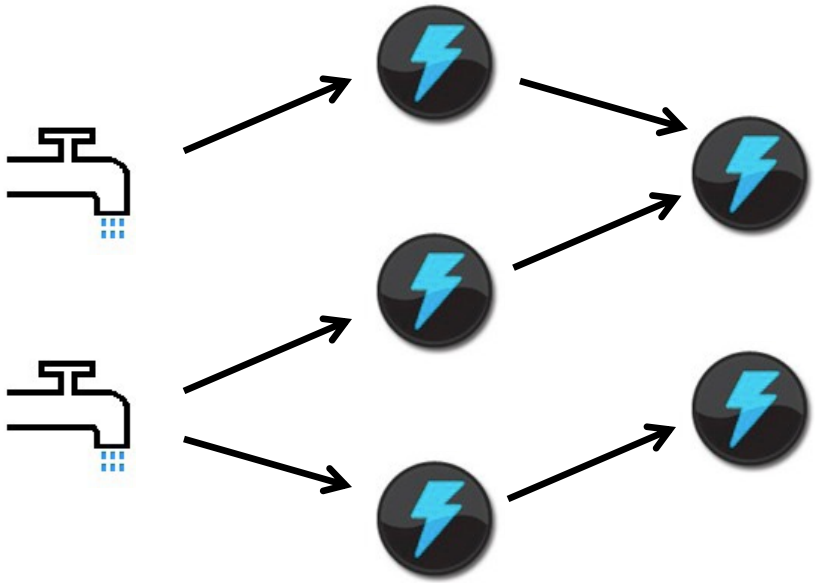

Source of streams

# Bolts



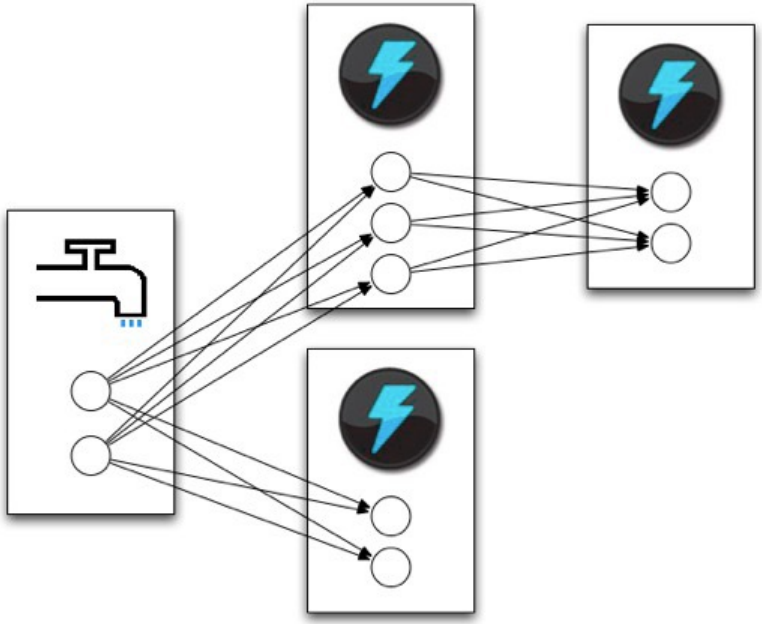

Processes input streams and produces new streams:
Can implement functions such as filters, aggregation, join, etc

# Topology



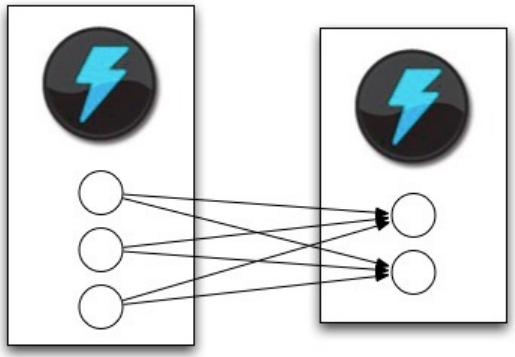
Network of spouts and bolts

# 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

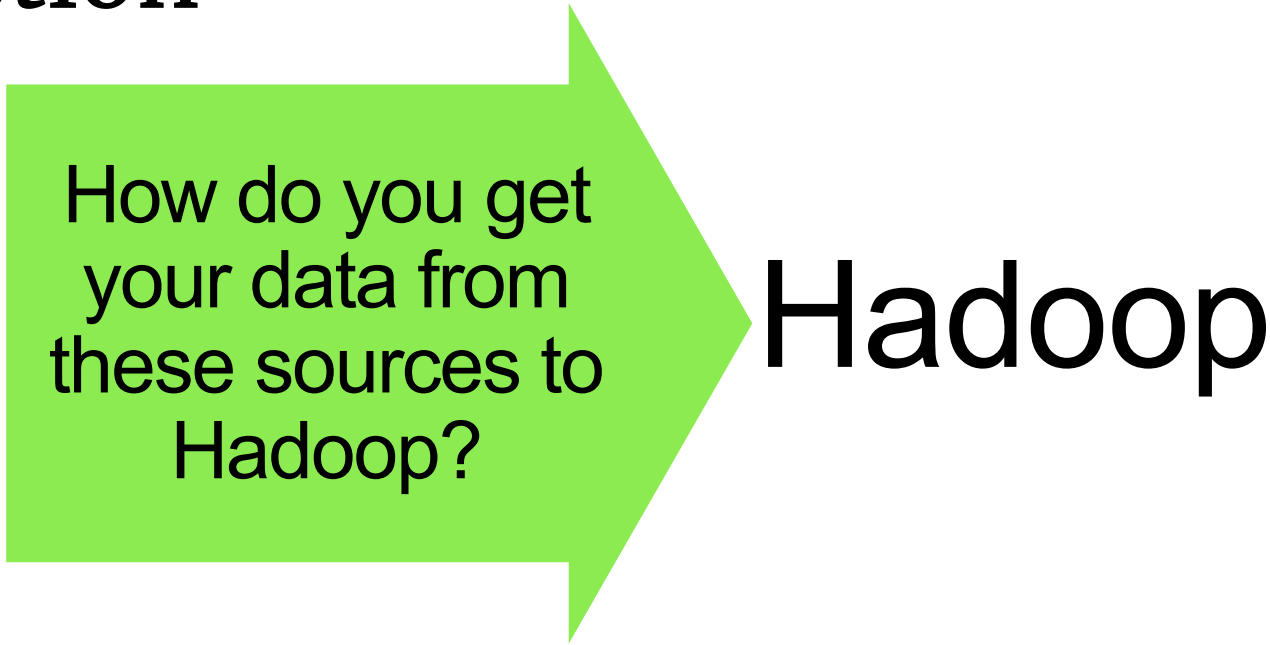
- **Shuffle grouping**: pick a random task
- **Fields grouping**: consistent hashing on a subset of tuple fields
- **All grouping**: send to all tasks
- **Global grouping**: pick task with lowest id

# Data Ingestion

# Data Ingestion

RDBMS

Applications



Application

RDBMS


Hadoop

Normally, Hadoop ecosystem technologies expose Java APIs:

use these APIs to write to HDFS, HBase etc.

# Issues

- Multiple events
- Streaming
- Diverse sources
- Buffering
- …

Flume and Sqoop can help!

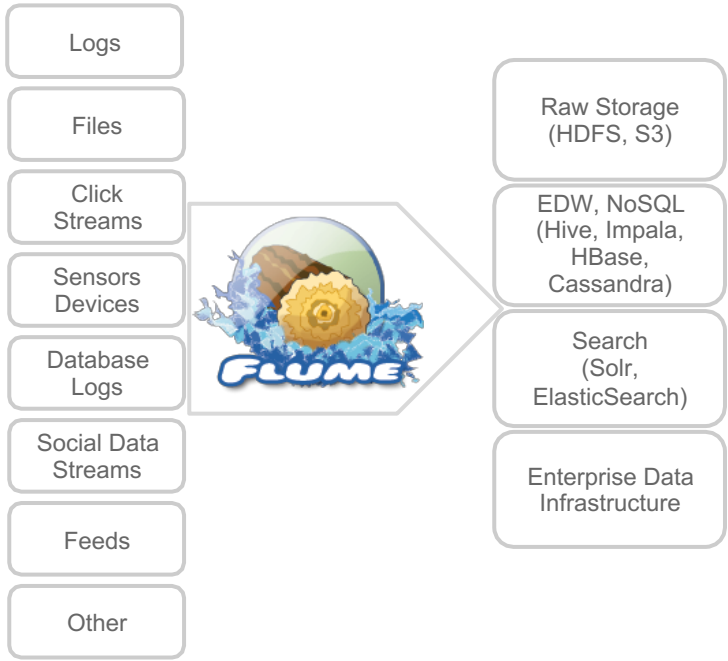
# 1. Apache Flume

# Apache Flume

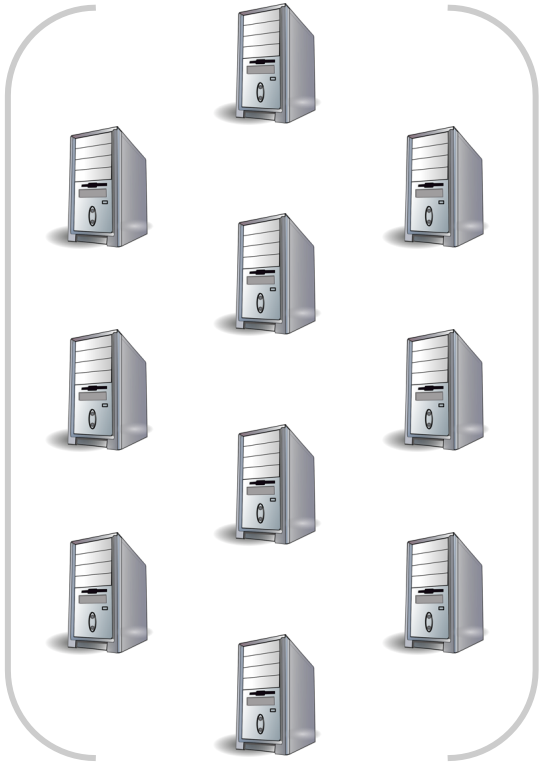
**Apache Flume** is a **continuous data ingestion system** that is...

- *open-source,*
- *reliable,*
- *scalable,*
- *manageable,*
- *customizable,*

...and designed for ***Big Data* ecosystem**.



# Multiple Sources

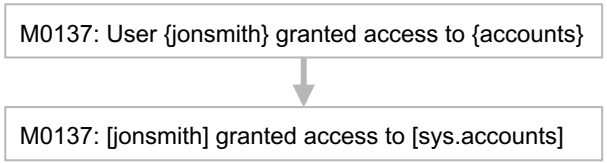


- **Many** physical sources that produce data
- Number of physical sources **changes constantly**
- Sources may exist in **different governance zones**, data centers, continents...

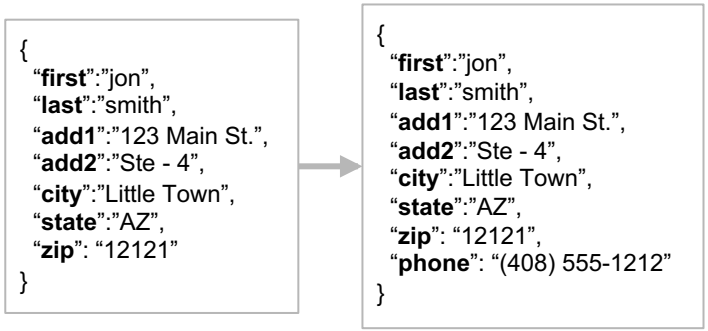
# Ever-changing Data

- One of your data centers upgrade to IPv6

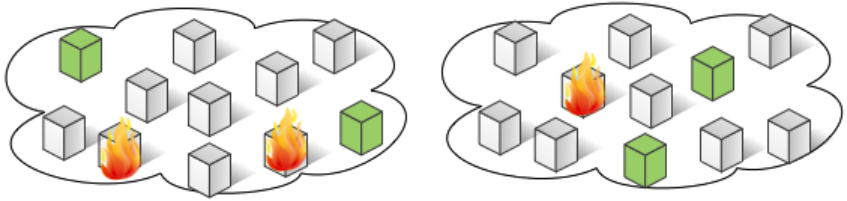

- Application developer changes logs (again)




- JSON data may contain more attributes than expected

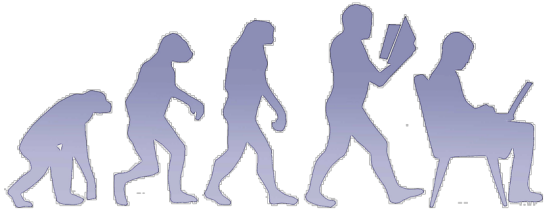


# Data Ingestion Perspective



Massive collection of ever changing physical sources...

Never ending data production...



Continuously evolving data structures and semantics...

# Flume History

- Originally designed to be a log aggregation system by Cloudera Engineers
- Evolved to handle any type of streaming event data
- Low-cost of installation, operation and maintenance
- Highly customizable and extendable

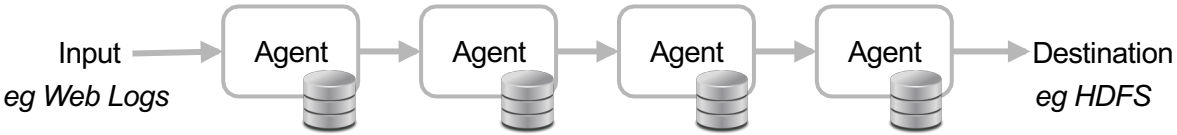
http://flume.apache.org/

# Flume Agent

Simplest unit in Flume

Can connect  any number of sources to any number of data stores

# The Big Picture




- Distributed Pipeline Architecture
- Optimized for commonly used data sources and destinations
- Built in support for contextual routing
- Fully customizable and extendable

# Flume Events

A Flume Event is the base unit of communication between components


Source pushes events

Sink polls for events

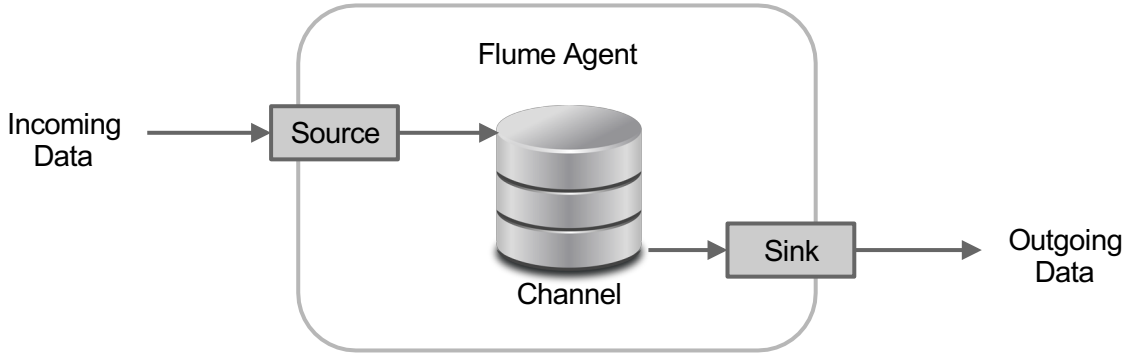
# Events

Header: metadata

- Can be used for routing content

Body: actual content

# A Flume Agent

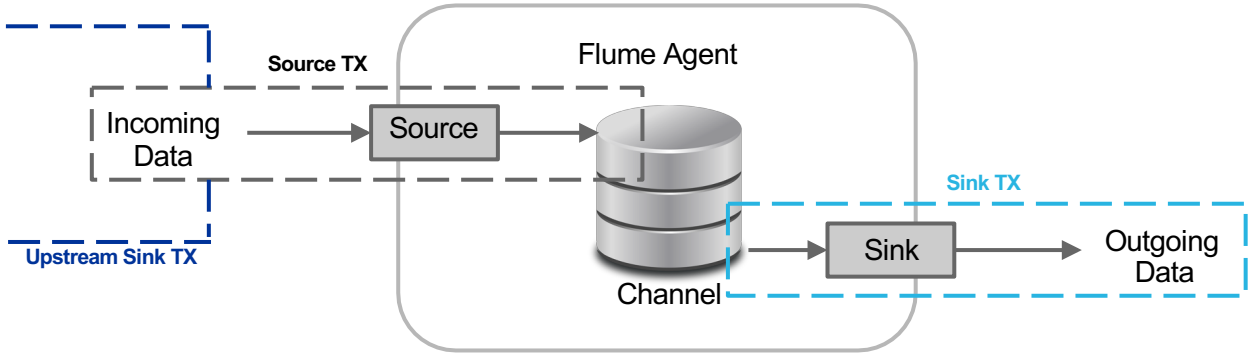



**Source**
- Accepts incoming Data
- Scales as required
- Writes data to Channel

**Channel**
- Stores data in the order received

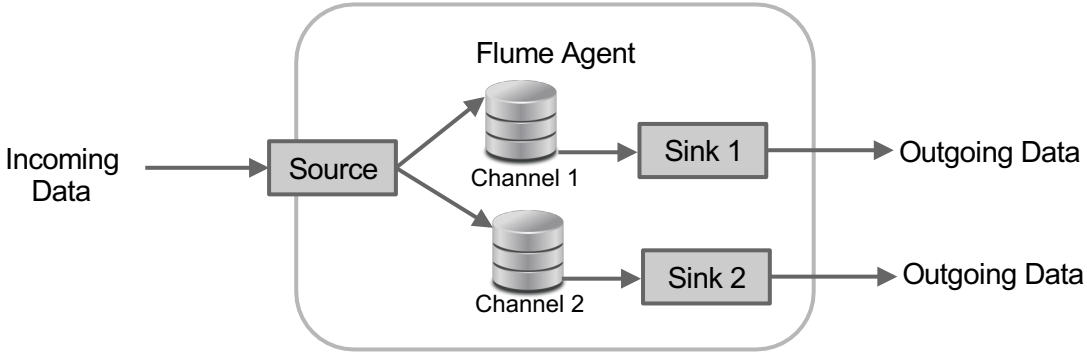
**Sink**
- Removes data from Channel
- Sends data to downstream Agent or Destination

# Transactional Data Exchange



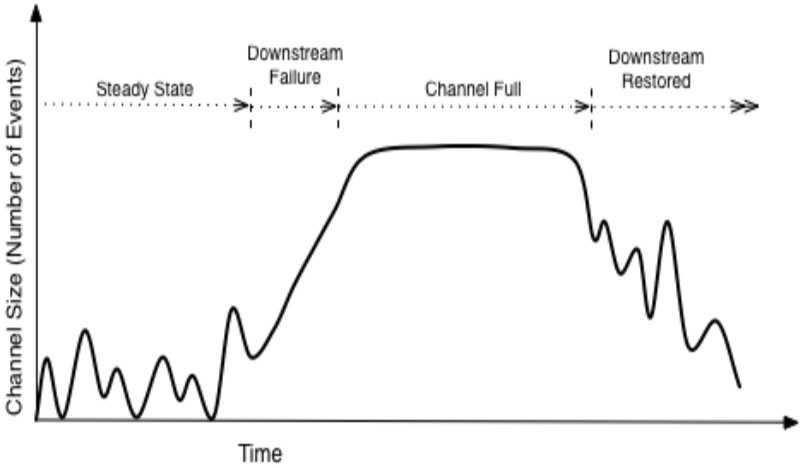

- Source uses transactions to write to the channel
- Sink uses transactions to remove data from the channel
- Sink transaction commits only after successful transfer of data
- This ensures no data loss in Flume pipeline

# Routing and Replicating




- Source can **replicate** or **multiplex** data across many channels
- Metadata headers can be used to do **contextual selection** of channels
- Channels can be drained by different sinks to different destinations or pipelines

# Why Channels?

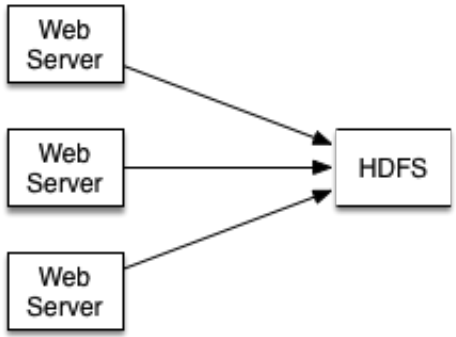



- Buffers data and insulates downstream from load spikes
- Provides persistent store for data in case the process restarts
- Provides flow ordering* and transactional guarantees

# Log Aggregation Case

# No Flume

- You would like to move your web-server logs to HDFS
- Let's assume there are only 3 web servers at the time of launch
- Ad-hoc solution will likely suffice!



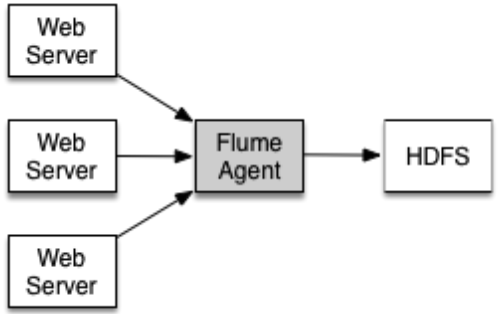

**Challenges**

- How do you manage your output paths on HDFS?
- How do you maintain your client code in face of changing environment as well as requirements?

# 1 Flume Agent

**Advantages**

- Insulation from HDFS downtime
- Quick offload of logs from Web Server machines
- Better Network utilization



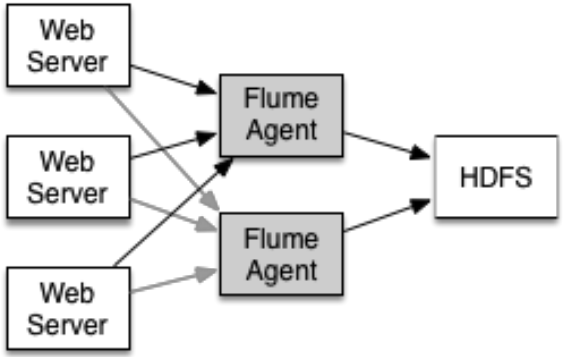

**Challenges**

- What if the Flume node goes down?
- Can one Flume node accommodate all load from Web Servers?

# 2 Flume Agents

**Advantages**

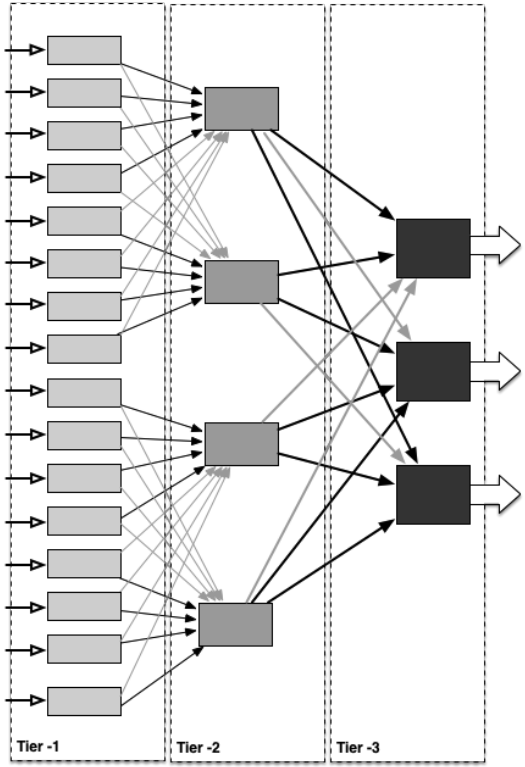
- Redundancy and Availability
- Better handling of downstream failures
- Automatic load balancing and failover




**Challenges**

- What happens when new Web Servers are added?
- Can two Flume Agents keep up with all the load from more Web Servers?

# Multi-Step Flow



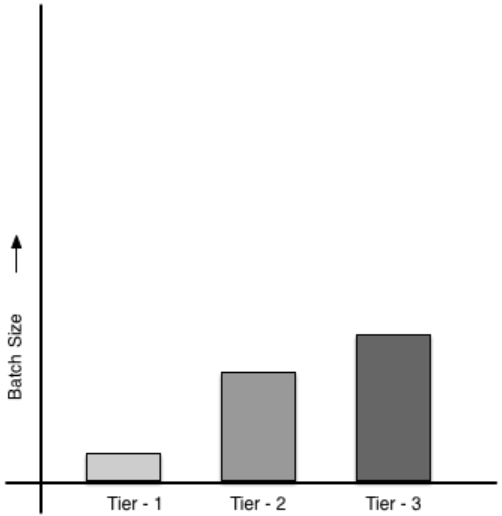

**A Converging Flow**

- Traffic is aggregated by Tier-2 and Tier-3 before being put into destination system
- Closer a tier is to the destination, larger the batch size it delivers downstream
- Optimized handling of destination systems

# Planning and Sizing

Marco Brambilla.

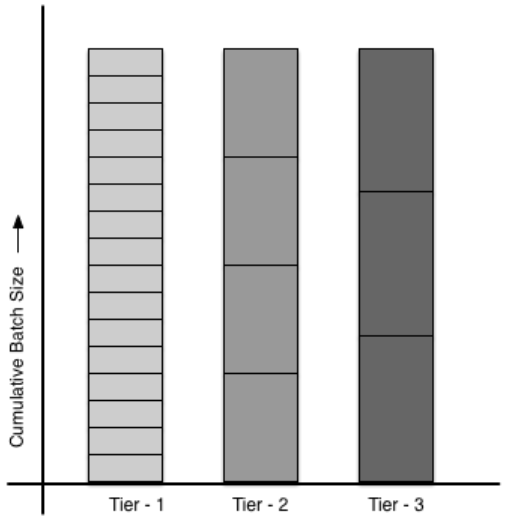
# Data Volume Distribution - Agent




**Batch Size Variation per Agent**

- Event volume is least in the outermost tier
- Event volume increases as the flow converges
- Event volume is highest in the innermost tier

# Data Volume Distribution - Tier




**Batch Size Variation per Tier**

- In steady state, all tiers carry same event volume
- Transient variations in flow are absorbed and ironed out by channels
- Load spikes are handled smoothly without overwhelming the infrastructure

# Planning and Sizing

**What we know:**

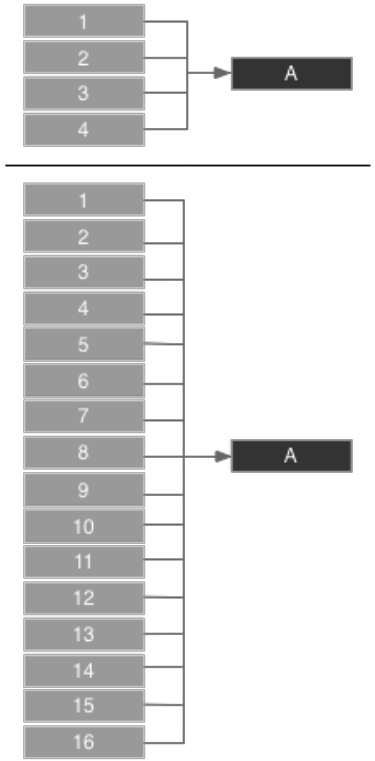
- Number of Web Servers
- Log volume per Web Server per unit time
- Destination System and layout (Routing Requirements)
- Worst case downtime for destination system

**What we will calculate:**

- Number of tiers
- Exit Batch Sizes
- Channel capacity

## Rule of Thumb

One Aggregating Agent (A) can be used with anywhere from 4 to 16 client Agents




## Considerations

- Must handle projected ingest volume
- Resulting number of tiers should provide for routing, load-balancing and failover requirements

## Test

Load test to ensure that steady state and peak load are addressed with adequate failover capacity

# Exit Size

**Rule of Thumb**

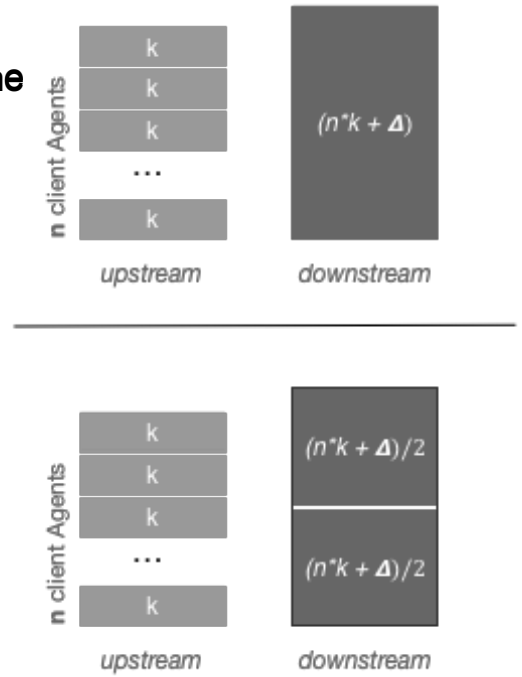
Exit batch size is same as total exit data volume divided by number of Agents in a tier

**Considerations**

- Having some extra room is good
- Keep contextual routing in mind
- Consider duplication impact when batch sizes are large

**Test**

Load test fail-over scenario to ensure near steady-state drain



# Channel Capacity

**Rule of Thumb**

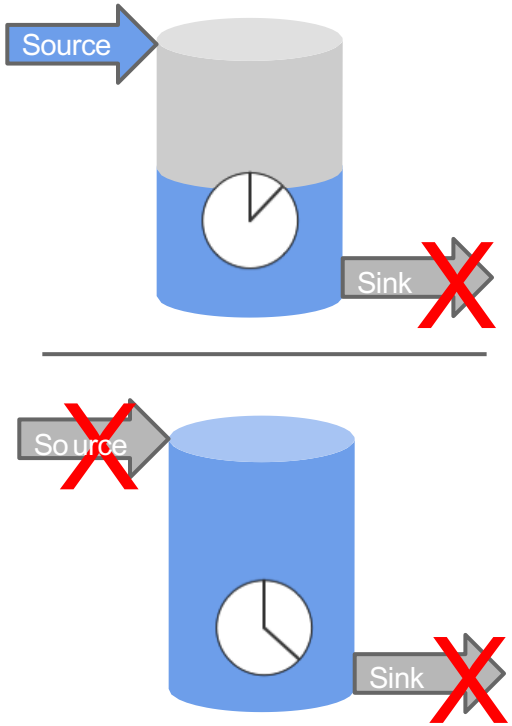
Equal to worst case data ingest rate sustained over the worst case downstream outage interval

**Considerations**

- Multiple disks will yield better performance
- Channel size impacts the back-pressure buildup in the pipeline

**Test**

You may need more disk space than the physical footprint of the data size



# Summary

### Number of Tiers

Calculated with upstream to downstream Agent ratio ranging from **4:1** to **16:1**. Factor in routing, failover, load-balancing requirements...

### Exit Batch Size

Calculated for steady state data volume exiting the tier, divided by number of Agents in that tier. Factor in contextual routing and duplication due to transient failure impact...

### Channel Capacity

Calculated as worst case ingest rate sustained over the worst case downstream downtime. Factor in number of disks used etc...

# Implementation

# Source-Channel-Sink

Source pushes data to channel

Channel stores it until Sink reads and writes it in output store (polling)

Source – Channel: many-to-many

Channel– Sink: one-to-many

(sink reads only from one channel)

# Types of Sources

Spooling Directory

- Text Files, Immutable, Unique name

SysLog

Exec

Custom

# Types of Channels

Memory

- In-memory queue
- Fast
- Not persistent
- Limited capacity

Disk

# Types of Sinks

HDFS

HBase

Console Log

Local

Directory

# Example: Spool to log collector

1. Configure agent(s) in property file

2. Run Flume agent from console

```
$ flume-ng agent
--conf-file spool-to-logger.properties
--name agent1
```

# Properties File: 1-1-1

## spool-to-logger.properties

```
agent1.sources =source1
agent1.sinks = sink1
agent1.channels = channel1


agent1.sources.source1.channels = channel1
agent1.sinks.sink1.channel = channel1

agent1.sources.source1.type = spooldir
agent1.sources.source1.spoolDir  = /Users/udi/tmp/spooldir

agent1.sinks.sink1.type = logger
agent1.channels.channel1.type = file
```

# HDFS

Sink = `org.apache.flume.sink.hdfs.HDFSEventSink`

```
agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /tmp/flume
agent1.sinks.sink1.hdfs.filePrefix = events
agent1.sinks.sink1.hdfs.fileSuffix = .log
agent1.sinks.sink1.hdfs.inUsePrefix = _
agent1.sinks.sink1.hdfs.fileType = DataStream
```

# Transfer

By default, files are rolled over every 30 seconds.

Configurable:

- rollCount
- rollSize

# Output file types

SequenceFile = binary format

DataStream = uncompressed files (text)

CompressedStream = compressed files

# HTTP to HDFS

```
httpagent.sources = http-source
httpagent.sinks = hdfs-sink
httpagent.channels = ch

httpagent.sources.http-source.channels = ch
httpagent.sinks.hdfs-sink.channel = ch

# Define / Configure Source
################################
httpagent.sources.http-source.type = org.apache.flume.source.http.HTTPSource
httpagent.sources.http-source.channels = ch
httpagent.sources.http-source.bind = localhost
httpagent.sources.http-source.port = 8989


# HDFS File Sink
################################
httpagent.sinks.hdfs-sink.type = hdfs
httpagent.sinks.hdfs-sink.hdfs.path = /tmp/flume/hdfs
httpagent.sinks.hdfs-sink.hdfs.filePrefix = events
httpagent.sinks.hdfs-sink.hdfs.fileSuffix = .log
httpagent.sinks.hdfs-sink.hdfs.inUsePrefix = _
httpagent.sinks.hdfs-sink.hdfs.fileType = DataStream

# Channels
################################
httpagent.channels.ch.type = memory
httpagent.channels.ch.capacity = 1000
```

# Send and get HTTP Posts

## Fixed format: header + body

```
$ curl -X POST \
-H 'Content-Type: application/json; charset=UTF-8' \
-d ‘[ {"headers" : {"eventheader1" : "event1", \
”eventheader2" : "event2" } , \
”body" : "This is the
body1 }]' \
http://localhost:8989
```

# Bucketing

```
httpagent.sinks.hdfs-sink.hdfs.path = /tmp/flume/http
```

Dynamic assignment of files to folders

```
httpagent.sinks.hdfs-sink.hdfs.path = /tmp/flume/http/%{topic}
```

Files assigned to folder based on the “Topic” event header value

```
{
   "headers" : {"topic" : "topic1"} ,
   ”body" : "This is the body1"
}
```

# Time-base bucketing

```
agent1.sinks.sink1.hdfs.useLocalTimeStamp = true
```

```
agent1.sinks.sink1.hdfs.path = /tmp/flume/twitterinterceptor/%Y/%m/%d/%H
```

# Spool to HBASE

```
agent1.sinks.hbaseSink.type = org.apache.flume.sink.hbase.AsyncHBaseSink
agent1.sinks.hbaseSink.table = test_table
agent1.sinks.hbaseSink.columnFamily = test
agent1.sinks.hbaseSink.serializer=org.apache.flume.sink.hbase.SimpleAsyncHbaseEventSerializer
agent1.sinks.hbaseSink.serializer.payloadColumn = pCol
```

Stores in specific Table and Column

They must exist already in HBASE

# Many Channels and Sinks

```
httpagent.sources = http-source
httpagent.sinks = hdfs-sink log-sink
httpagent.channels = ch1 ch2

httpagent.sources.http-source.channels = ch1 ch2
httpagent.sinks.hdfs-sink.channel = ch1
httpagent.sinks.log-sink.channel = ch2
# Define / Configure Source
################################
httpagent.sources.http-source.type = org.apache.flume.source.http.HTTPSource
httpagent.sources.http-source.bind = localhost
httpagent.sources.http-source.port = 8989
# HDFS File Sink
################################
httpagent.sinks.hdfs-sink.type = hdfs
httpagent.sinks.hdfs-sink.hdfs.path = /tmp/flume/http
httpagent.sinks.hdfs-sink.hdfs.filePrefix = events
httpagent.sinks.hdfs-sink.hdfs.fileSuffix = .log
httpagent.sinks.hdfs-sink.hdfs.inUsePrefix = _
httpagent.sinks.hdfs-sink.hdfs.fileType = DataStream
# Logger sink
################################
httpagent.sinks.log-sink.type = logger
# Channels
################################
httpagent.channels.ch1.type = memory
httpagent.channels.ch1.capacity = 1000

httpagent.channels.ch2.type = file
```

# Many channels and sinks

## Replicating

- Default
- Every event sent everywhere

## Multiplexing

- routes events to channels based on their headers

# Multiplexing Selector

```
# Selector
#####################################
httpagent.sources.http-source.selector.type = multiplexing
httpagent.sources.http-source.selector.header = show
httpagent.sources.http-source.selector.mapping.1 = ch1
httpagent.sources.http-source.selector.mapping.2 = ch2
```

Based on header:

- If show = 1, route to ch1
- If show = 2, route ch2

# Twitter Source

```
agent1.sources.source1.type = org.apache.flume.source.twitter.TwitterSource
agent1.sources.source1.consumerKey = **
agent1.sources.source1.consumerSecret = **
agent1.sources.source1.accessToken = **
agent1.sources.source1.accessTokenSecret = **
agent1.sources.source1.keywords = @realDonaldTrump, @HillaryClinton
```

# Interceptors

Interceptors can process events based on their contents  or headers

Simple data processing

Overhead for Flume

To be used with caution!

# RegEx Filter Interceptor

```
agent1.sources.source1.interceptors.regexInterceptor.type = regex_filter
agent1.sources.source1.interceptors.regexInterceptor.regex = .*election.*
agent1.sources.source1.interceptors.regexInterceptor.excludeEvents = false
```

# Flume Summary

- Flume is suitable for large volume data collection, especially when data is being produced in multiple locations
- Once planned and sized appropriately, Flume will practically run itself without any operational intervention
- Flume provides weak ordering guarantee, i.e., in the absence of failures the data will arrive in the order it was received in the Flume pipeline
- Transactional exchange ensures that Flume never loses any data in transit between Agents. Sinks use transactions to ensure data is not lost at point of ingest or terminal destinations.
- Flume has rich out-of-the box features such as contextual routing, and support for popular data sources and destination systems

# 2. Apache Sqoop

# Sqoop

- Importing from relational DB sources
- PULL-based (no streams)
- Bulk imports
- With one command line action, import data from RDBMS to HDFS/Hive
- Sqoop comes with connectors for many popular RDBMS: MySQL, Oracle, SQL Server, PostgreSQL, etc

# Use Cases

Occasional copy for running map-reduce tasks

Keep RDBMS for transactional needs and periodically dump data on HDFS

# Command Line Instructions

```
sqoop import \
--connect jdbc:mysql://localhost:3306/nseProd \
--username=qt \
--password=password \
--table=tradingDays \
--target-dir /mysql/nseProd \
--m 1
```

# Command Line Instructions

Sqoop uses the primary key to decide how many mappers to use, and for splitting the rows among mappers

`--m 1` fixes 1 mapper

mandatory if the table you are importing doesn't have a primary key, or if you are importing a query

# Split-by

explicitly specify the column to use for splitting  rows between mappers

# Query based importer

```
--query 'select year, month, day from
tradingDays where year=2016 and $CONDITIONS'
```

- Instead of full table
- $CONDITIONS mandatory

# Import in Hive

```
sqoop import \
--connect jdbc:mysql://localhost:3306/nseProd \
--username=qt \
--password=password \
--table=tradingDays \
--hive-import \
--hive-table=tradingDays \
--target-dir /mysql/table/tradingDays2 \
--m 1
```

Sqoop will create an external table and point the table to the directory in HDFS

# Incremental Import: Jobs

Periodically archiving on incremental content on HDFS/Hive

Import only the new data

Save an import configuration as a **JOB** and call it later on repetitively

# Job Definition

```
sqoop job \
--create myjob \
--import \
--connect jdbc:mysql://localhost:3306/nseProd \
--username=qt \
--password=password \
--table=tradingDays \
--target-dir /mysql/nseProd \
--m 1
```

# Incremental Job Definition

```
sqoop job \
--create myjob \
--import \
--connect jdbc:mysql://localhost:3306/nseProd \
--username=qt \
--password=password \
--table=tradingDays \
--target-dir /mysql/nseProd \
--m 1
--incremental lastmodified
–check-column ts
```

# Behaviour

Only rows added or modified will be imported

Based on the value of --check-column

- Ideally a timestamp of last update of the row
- LastValue saves the max value of the column at every import
- Further import will get data > LastValue

# Job Execution

```
sqoop job --exec myjob
```

# Acknowledgements

Matei Zaharia, UC Berkeley RAD Lab

Tom White, Lexeme Ltd.

Nathan Marz, Twitter