APACHE STORM

A scalable distributed & fault tolerant real time computation system

(Free & Open Source)

Shyam Rajendran

17-Feb-15

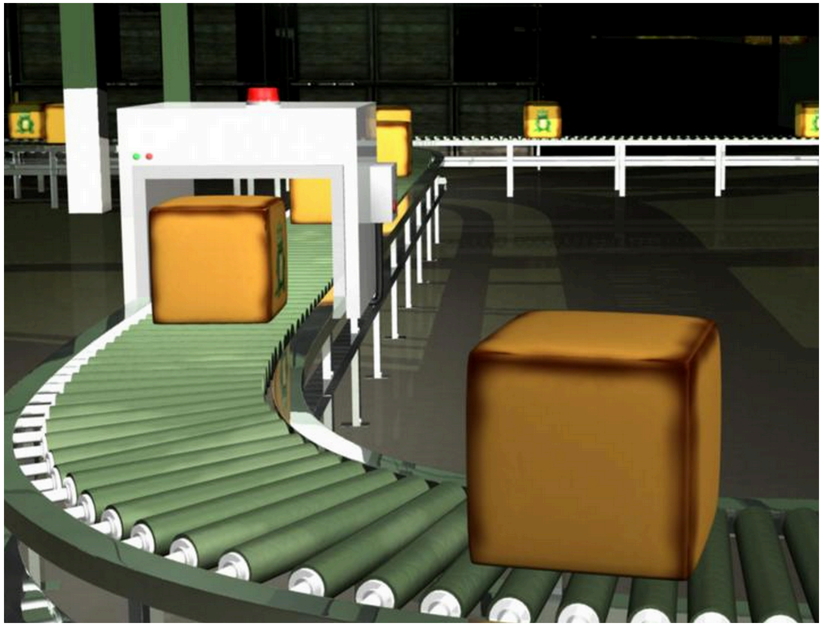
Agenda

- History & the whys
- Concept & Architecture
- Features
- ***Demo!***

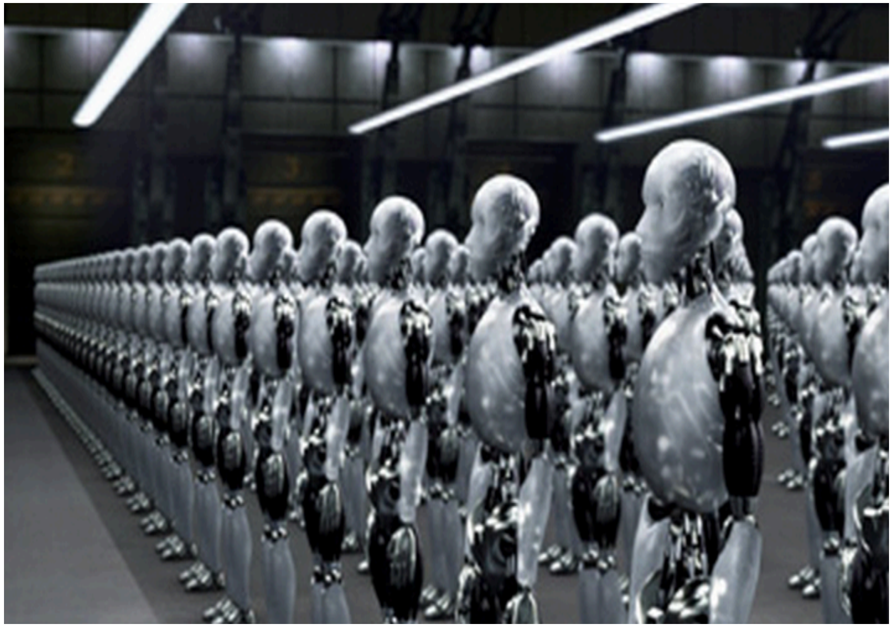
History

- ***Before the Storm***

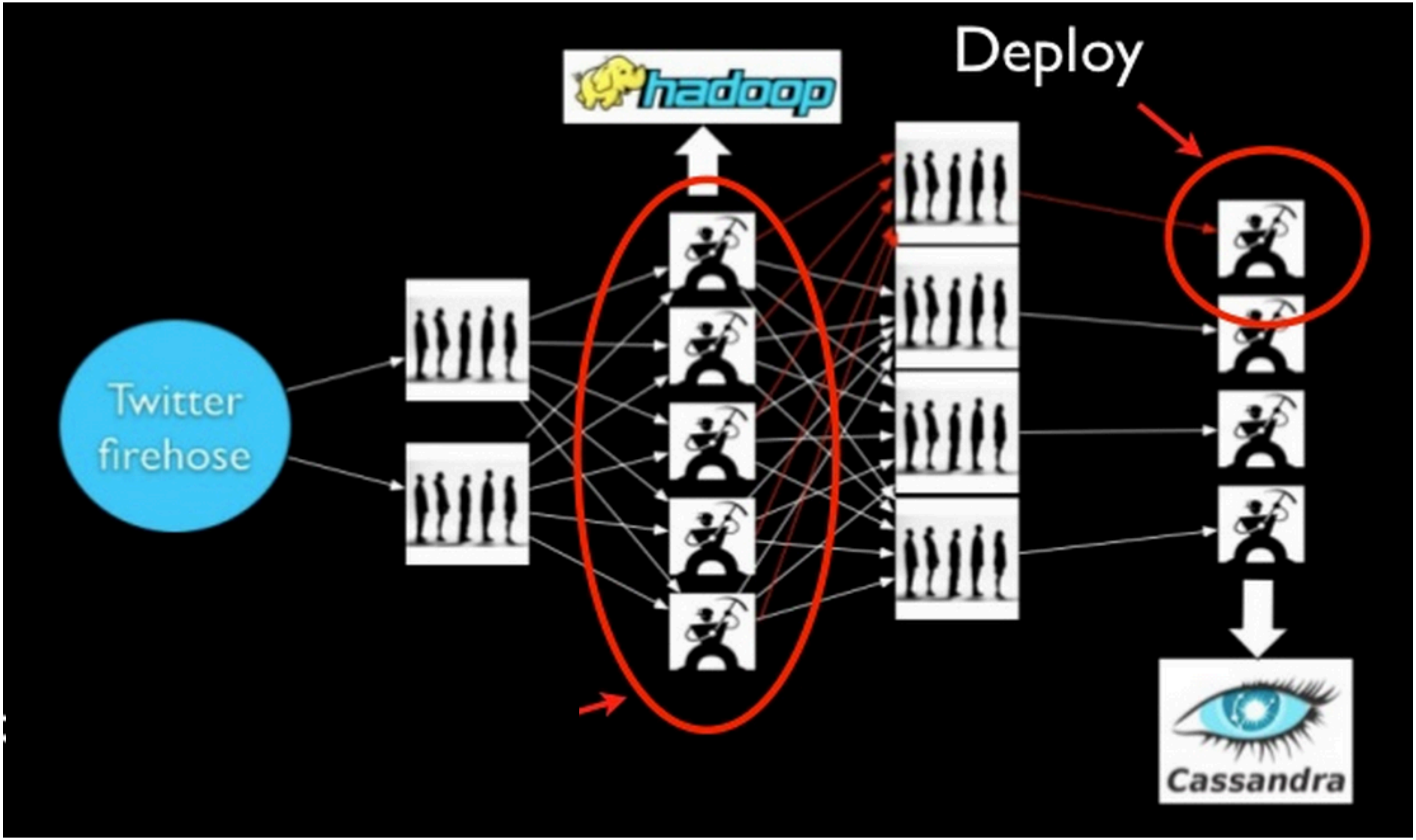
Queues



Workers



Analyzing Real Time Data (old)



http://www.slideshare.net/nathanmarz/storm-distributed-and-faulttolerant-realtime-computation

History

- **Problems?**
  - **Cumbersome** to build applications (manual + tedious + serialize/deserialize message)
  - **Brittle** (No fault tolerance)
  - **Pain to scale** - same application logic spread across many workers, deployed separately

- **Hadoop ?**
  - For parallel batch processing : No Hacks for realtime
  - Map/Reduce is built to leverage data localization on HDFS to distribute computational jobs.
  - Works on big data.

Why not as one self-contained application?

http://nathanmarz.com.

Enter the Storm!

- BackType (Acquired by Twitter)

Nathan Marz* + Clojure

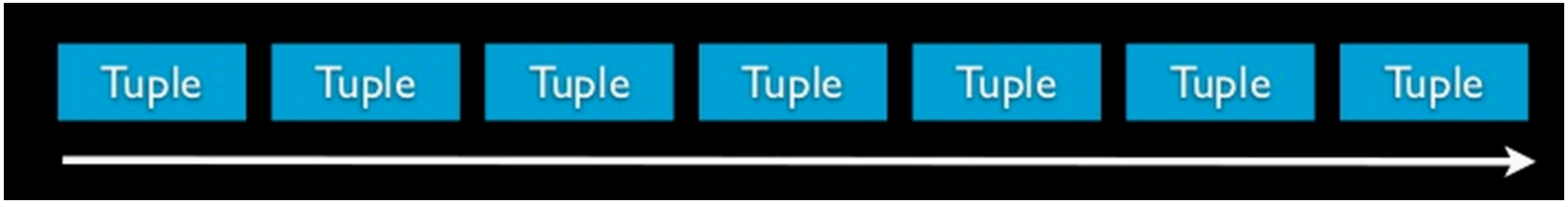
- **Storm !**
  - Stream process data in realtime with no latency!
  - Generates big data!

Features

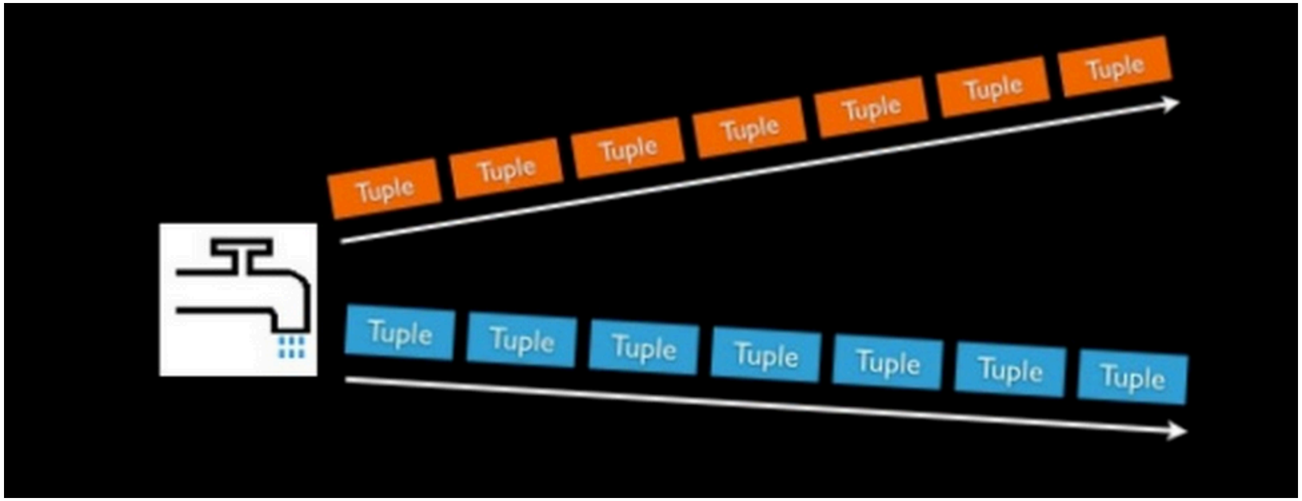
- **Simple programming model**
  - Topology - Spouts – Bolts
- **Programming language agnostic**
  - (*Clojure, Java, Ruby, Python default*
- **Fault-tolerant**
- **Horizontally scalable**
  - Ex: 1,000,000 messages per second on a 10 node cluster
- **Guaranteed message processing**
- **Fast : Uses zeromq message queue**
- **Local Mode : Easy unit testing**

Concepts – Steam and Spouts

- **Stream**
  - Unbounded sequence of tuples (storm data model)
  - <key, value(s)> pair ex. <“UIUC”, 5>

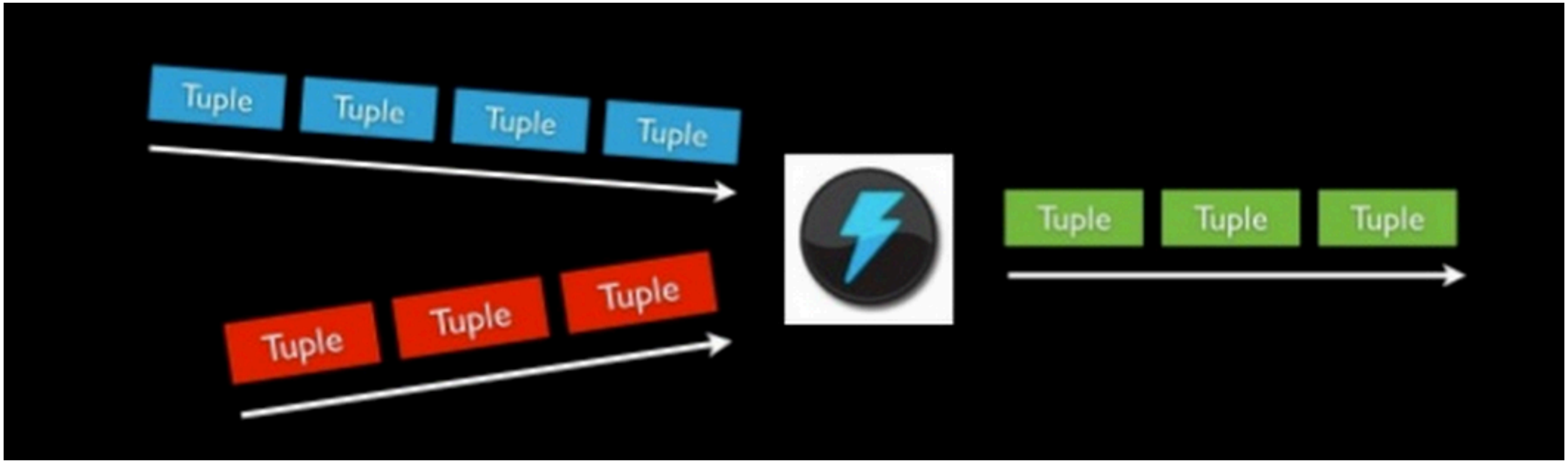


- **Spouts**
  - Source of streams : Twitterhose API
  - Stream of tweets or some crawler



Concept - Bolts

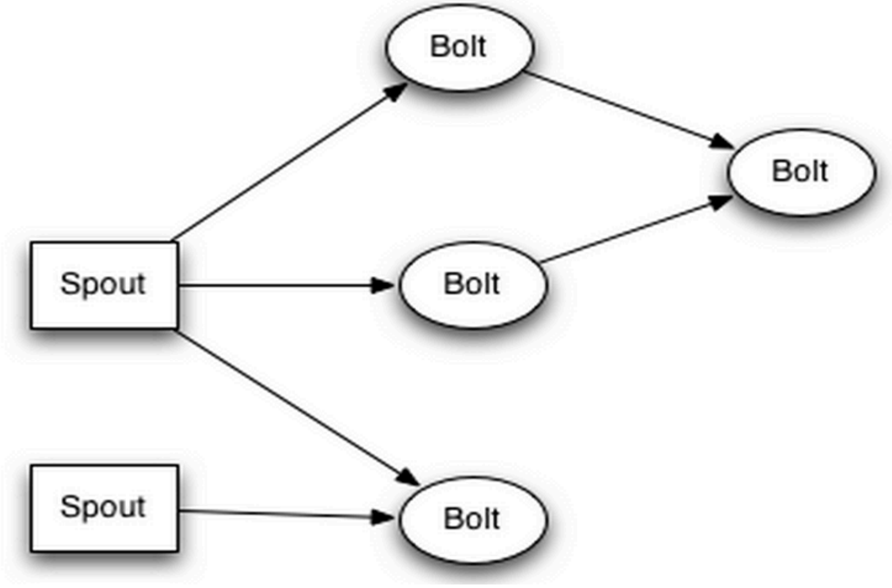
- **Bolts**
  - Process (one or more) input stream and produce new streams



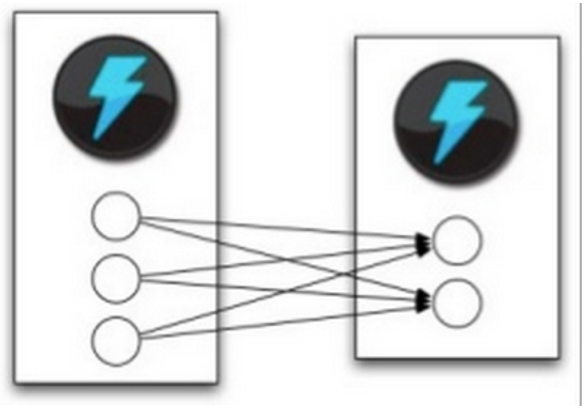
- **Functions**
  - Filter, Join, Apply/Transform etc
  - Parallelize to make it fast! – multiple processes constitute a bolt

Concepts – Topology & Grouping

- **Topology**
  - Graph of computation – can have cycles
  - Network of Spouts and Bolts
  - Spouts and bolts execute as many tasks across the cluster

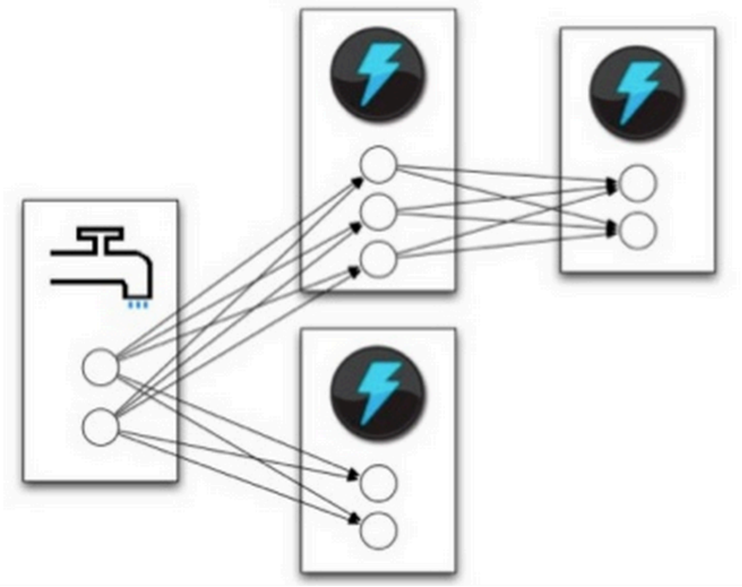


- **Grouping**
  - How to send tuples between the components / tasks?



Concepts – Grouping

- **Shuffle Grouping**
  - Distribute streams "*randomly*" to bolt's tasks

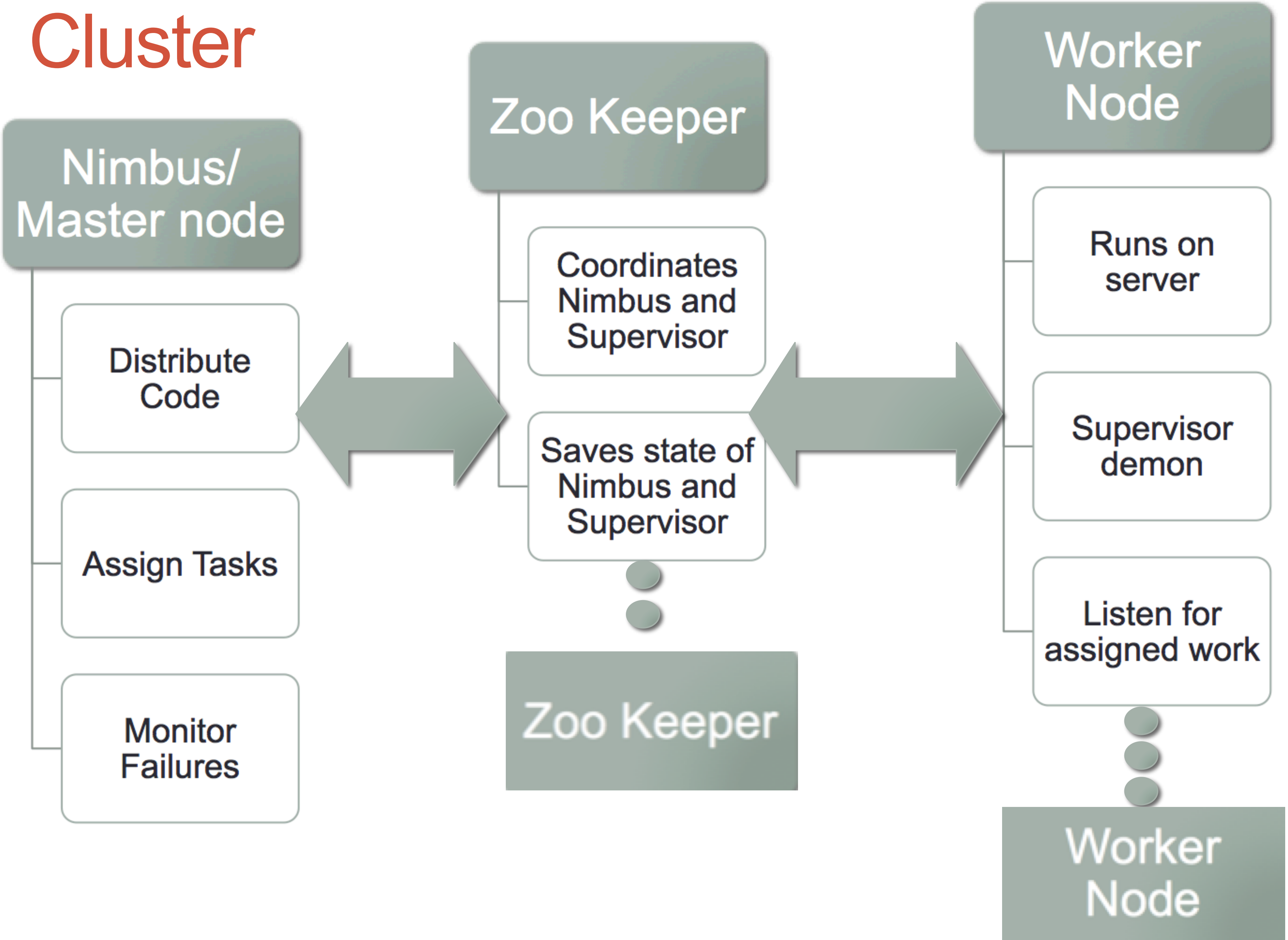


- **Fields Grouping**
  - Group a stream by a subset of its fields
- **All Grouping**
  - All tasks of bolt receive all input tuples
  - Useful for joins
- **Global Grouping**
  - Pick task with lowers id

Zookeeper

- Open source server for highly reliable distributed coordination.
- As a replicated synchronization service with eventual consistency.
- **Features**
  - Robust
    - Persistent data replicated across multiple nodes
  - Master node for writes
  - Concurrent reads
  - Comprises a tree of **znodes**, - entities roughly representing file system nodes.
  - Use only for saving small configuration data.

Cluster



Nimbus/ Master node

Distribute Code

Assign Tasks

Monitor Failures

Zoo Keeper

Coordinates Nimbus and Supervisor

Saves state of Nimbus and Supervisor

Zoo Keeper

Worker Node

Runs on server

Supervisor demon

Listen for assigned work

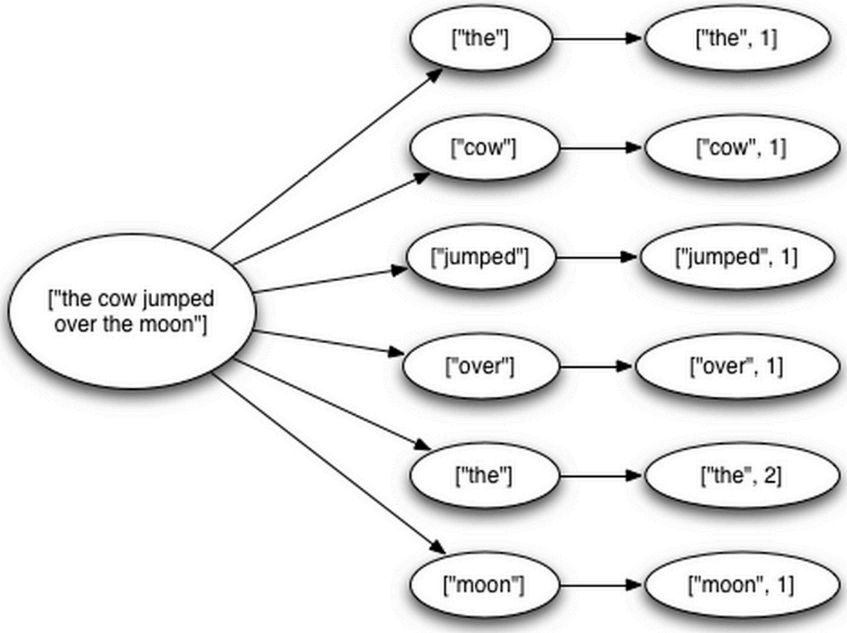
Worker Node

Features

- Simple programming model
  - Topology - Spouts – Bolts
- Programming language agnostic
  - (*Clojure, Java, Ruby, Python default*)
- **Guaranteed message processing**
- Fault-tolerant
- Horizontally scalable
  - Ex: 1,000,000 messages per second on a 10 node cluster
- Fast : Uses zeromq message queue
- Local Mode : Easy unit testing

Guranteed Message Processing

- **When is a message “Fully Proceed” ?**



"fully processed" when the tuple tree has been exhausted and every message in the tree has been processed

A tuple is considered failed when its tree of messages fails to be fully processed within a specified timeout.

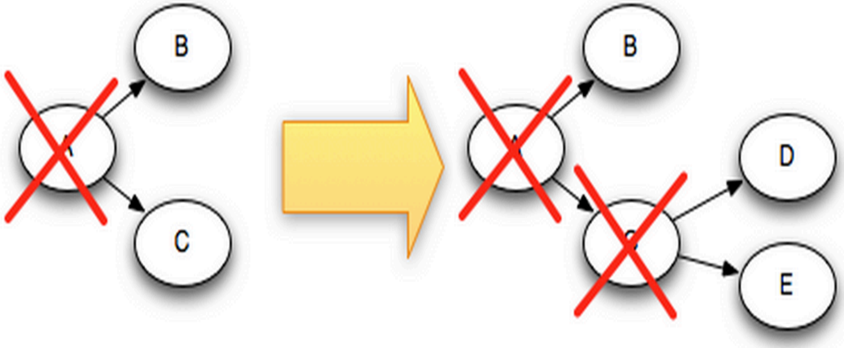
- **Storms’s reliability API ?**
  - Tell storm whenever you create a new link in the tree of tuples
  - Tell storm when you have finished processing individual tuple

Fault Tolerance APIS

- **Emit**(tuple, output)
  - Emits an output tuple, perhaps anchored on an input tuple (first argument)
- **Ack**(tuple)
  - Acknowledge that you (bolt) finished processing a tuple
- **Fail**(tuple)
  - Immediately fail the spout tuple at the root of tuple topology if there is an exception from the database, etc.
- Must remember to ack/fail each tuple
  - Each tuple consumes memory. Failure to do so results in memory leaks.

Fault-tolerant

- **Anchoring**
  - Specify link in the tuple tree. (anchor an output to one or more input tuples.)
  - At the time of emitting new tuple
  - Replay one or more tuples.



How?

- Every individual tuple must be acked.
- If not task will run out of memory!
- Filter Bolts ack at the end of execution
- Join/Aggregation bolts use multi ack .

"acker" tasks

- Track DAG of tuples for every spout
- Every tuple (spout/bolt) given a random 64 bit id
- Every tuple knows the ids of all spout tuples for which it exits.

What's the catch?

Failure Handling

- A tuple isn't acked because the task died:
  - Spout tuple ids at the root of the trees for the failed tuple will time out and be replayed.
- **Acker task dies:**
  - All the spout tuples the acker was tracking will time out and be replayed.
- **Spout task dies:**
  - The source that the spout talks to is responsible for replaying the messages.
    - For example, queues like Kestrel and RabbitMQ will place all pending messages back on the queue when a client disconnects.

Storm Genius

- **Major breakthrough** : Tracking algorithm
- Storm uses mod hashing to map a spout tuple id to an acker task.
- **Acker task:**
  - Stores a map from a spout tuple id to a pair of values.
    - Task id that created the spout tuple
    - Second value is 64bit number : Ack Val
      - XOR all tuple ids that have been created/acked in the tree.
  - Tuple tree completed when Ack Val = 0

- **Configuring Reliability**
  - Config.TOPOLOGY_ACKERS to 0.
  - you can emit them as unanchored tuples

Exactly Once Semantics ?

- **Trident**
  - High level abstraction for realtime computing on top of storm
  - Stateful stream processing with low latency distributed quering
  - Provides ***exactly-once*** semantics (avoid over counting)

How can we do ?

Store the transaction id with the count
in the database as an atomic value

https://storm.apache.org/documentation/Trident-state

Exactly Once Mechanism

Lets take a scenario

- Count aggregation of your stream
- Store running count in database. Increment count after processing tuple.
- **Failure!**

Design

- Tuples are processed as small batches.
- Each batch of tuples is given a unique id called the "transaction id" (txid).
- If the batch is replayed, it is given the exact same txid.
- State updates are ordered among batches.

Exactly Once Mechanism (contd.)

Design

- Processing txid = 3
- Database state

```
man      => [count=3, txid=1]
dog      => [count=4, txid=3]
apple    => [count=10, txid=2]
```

["man"]
["man"]
["dog"]

- If they're the same : SKIP
  (Strong Ordering)
- If they're different,
  you increment the count.

```
man      => [count=5, txid=3]
dog      => [count=4, txid=3]
apple    => [count=10, txid=2]
```

https://storm.apache.org/documentation/Trident-state

Improvements and Future Work

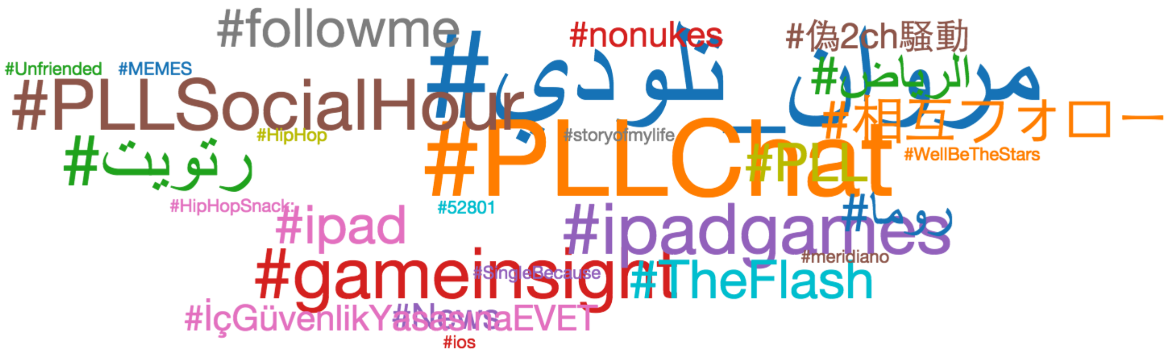
- **Lax security policies**
- **Performance and scalability improvements**
  - Presently with just 20 nodes SLAs that require processing more than a million records per second is achieved.
- **High Availability (HA) Nimbus**
  - Though presently not a single point of failure, it does affect degrade functionality.
- **Enhanced tooling and language support**

python
node js
cascading
Java
Scala
Groovy
R
PERL

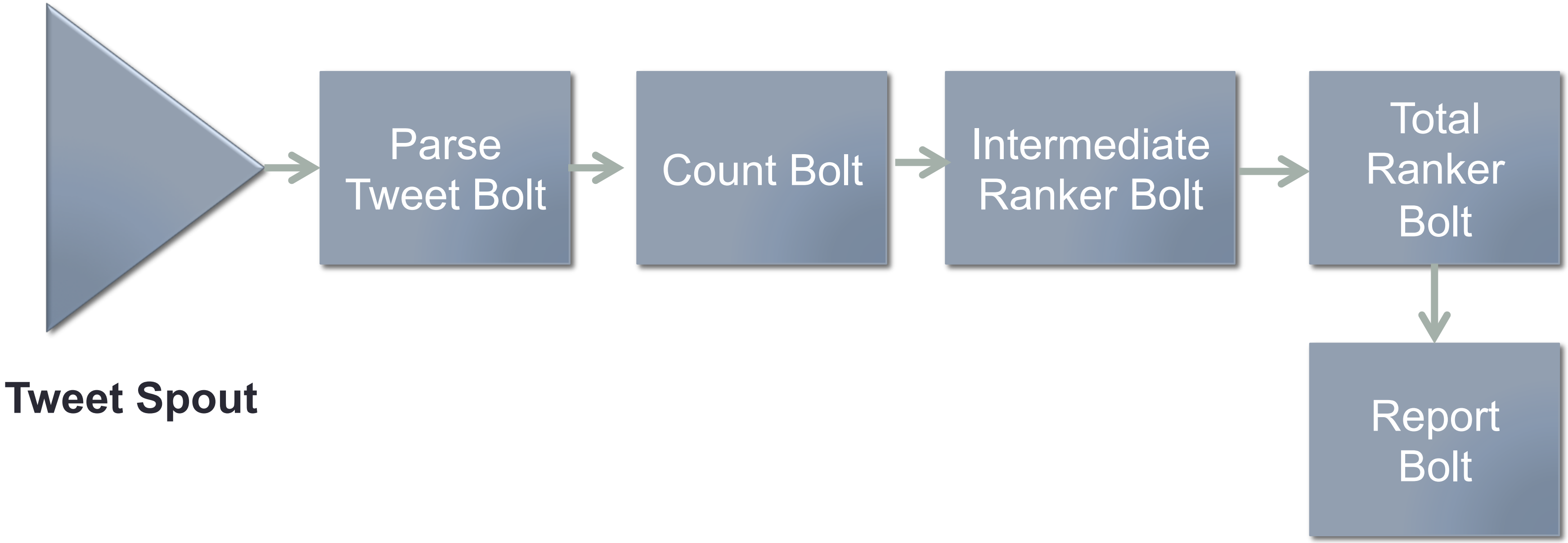
DEMO

Twitter Real-Time Analytics using Apache Storm

A demo for CS-525 Advanced Distributed Systems



Topology



Tweet Spout

Parse Tweet Bolt

Count Bolt

Intermediate Ranker Bolt

Total Ranker Bolt

Report Bolt

Downloads

- Download the binaries, Install and Configure - **ZooKeeper**.
- - Download the code, build, install - **zeromq** and **jzmq**.
- - Download the binaries, Install and Configure – **Storm**.

References

- https://storm.apache.org/
- http://www.slideshare.net/nathanmarz/storm-distributed-and-faulttolerant-realtime-computation
- http://hortonworks.com/blog/the-future-of-apache-storm/
- http://zeromq.org/intro:read-the-manual
- http://www.thecloudavenue.com/2013/11/InstallingAndConfiguringStormOnUbuntu.html
- https://storm.apache.org/documentation/Setting-up-a-Storm-cluster.html