

IERG4330/ ESTR4316/ IERG5730

Spring 2022

# Stream Processing and Apache Storm

Prof. Wing C. Lau

Department of Information Engineering

wclau@ie.cuhk.edu.hk

# Acknowledgements

- The slides used in this chapter are adapted from the following sources:
  - Nathan Marz, “Storm – Distributed and Fault-tolerant real-time computation,” 2011, <http://cloud.berkeley.edu/data/storm-berkeley.pdf>
  - Krishna Gade of Twitter, “storm - Stream Processing @twitter,” June 2013.
  - Michael G. Noll of Verisign, “Apache Storm 0.9 basic training,” July, 2014, <http://www.slideshare.net/miguno/apache-storm-09-basic-training-verisign>
  - Guido Schmutz of Trivadis, “Apache Storm vs. Spark Streaming – Two Stream Processing Platforms compared,” DBTA Workshop on Stream Processing, Berne, Dec 2014.
  - Bobby Evans of Yahoo!, “From Gust to Tempest: Scaling Storm,” talk at Hadoop Summit 2015.
  - Sean T. Allen, Matthew Jankowski, Peter Pathirana , Storm Applied, Published by Manning, 2015.
  - Rahul Jain, “Real time Analytics with Apache Kafka and Spark,” Big Data Hyderabad Meetup, Oct 2014
- All copyrights belong to the original authors of the materials.

# Example Use Case: Data Driven Personalization

<http://visualize.yahoo.com/core>

40,508,219 Homepage Views Today on **YAHOO!**  
POWERED BY C.O.R.E.

Gender



Age

18-24

25-34

**35-44**

45-54

55+

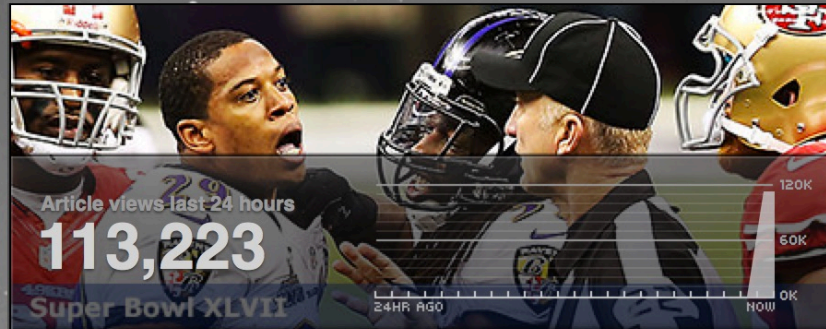
Infographics



Interest



Cities



## Player gets away with shoving referee

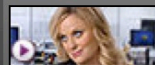
An eruption by Baltimore's Cary Williams should have led to his ejection from the game.

[Photo evidence >](#)

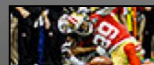
- [Super Bowl live chat](#)
- [Best of the action](#)
- [Complete coverage](#)



Demographic Data



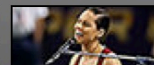
Poehler lines you didn't hear



Controversial player struggles



A Super Bowl first for 49ers



Keys's anthem sets record

MORE LIKE THIS

These are the most viewed stories by

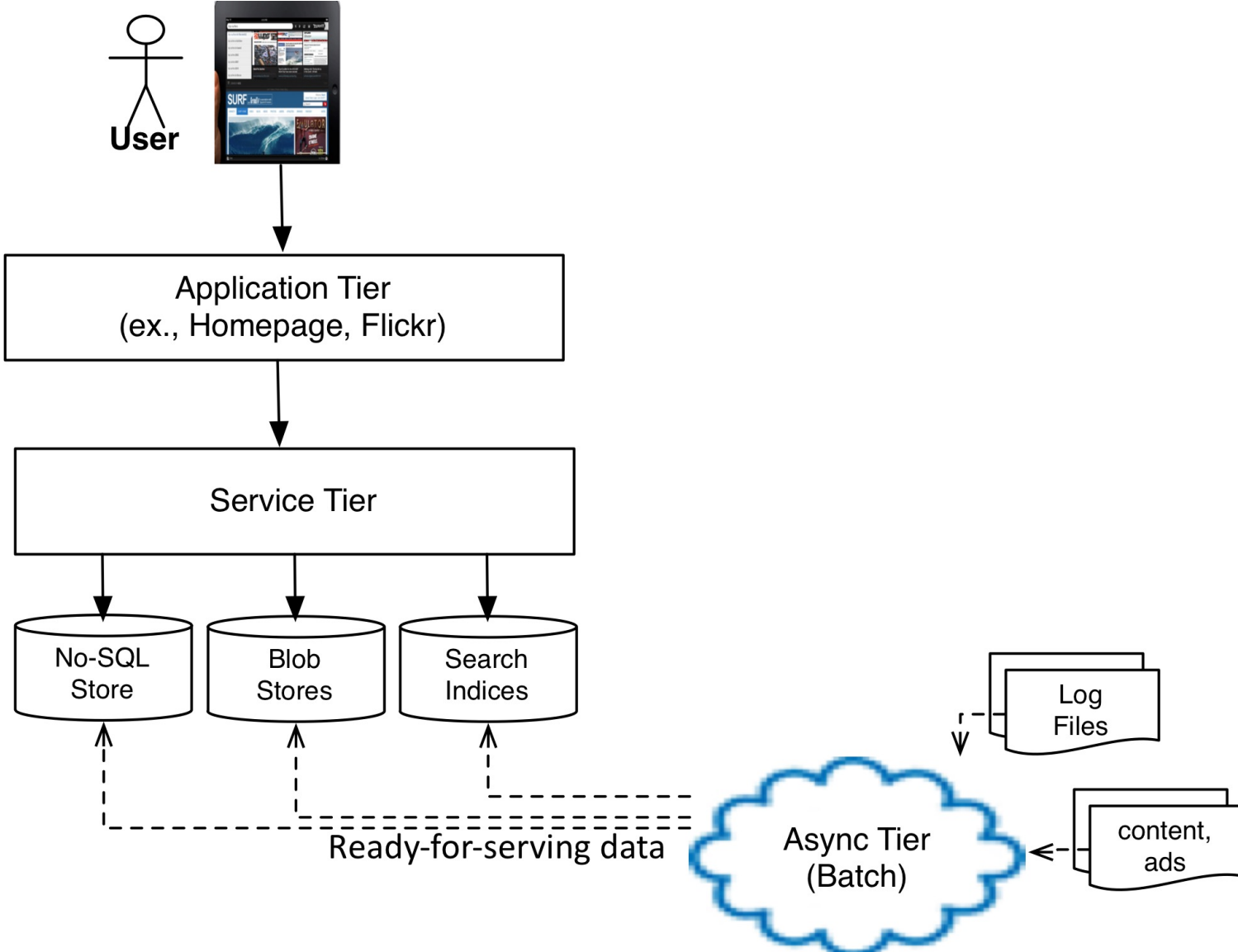
Men aged 35-44 interested in Sports

Explore the most viewed stories today

24 HOURS AGO

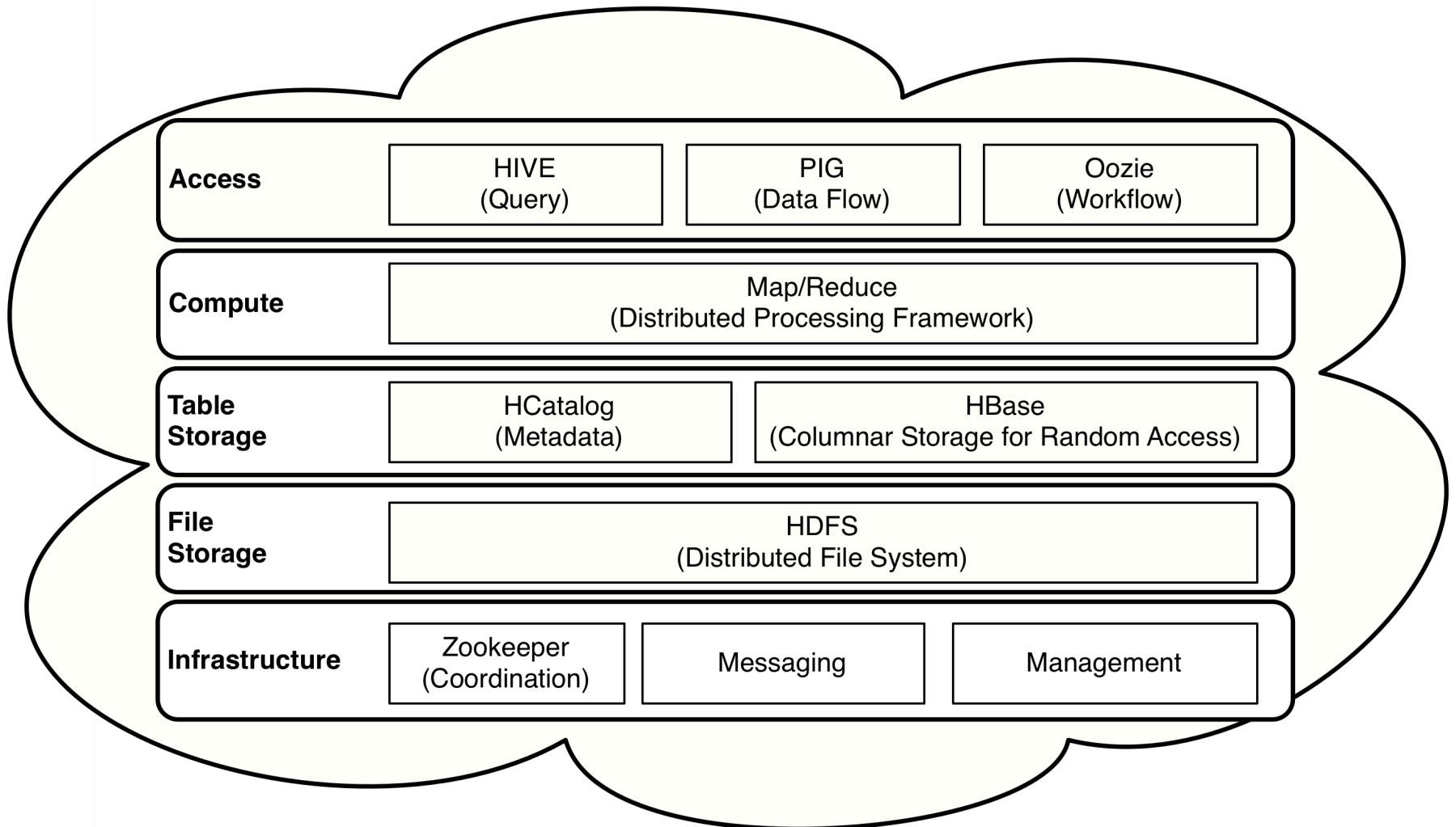
Explore

# System Architecture for Data Driven Personalization based on Offline, Asynchronous (e.g. Daily/Weekly) Log Processing/ Data Analytics





## @ Async Tier (before 2010)



# Pros and Cons of Async Processing via Hadoop

## Strength

- Batch processing
  - simple programming model
- Massively scalable
  - 1000' s node cluster w/ commodity hardware
- High throughput
  - Move computation to the data nodes
- Highly available
  - Built-in failover

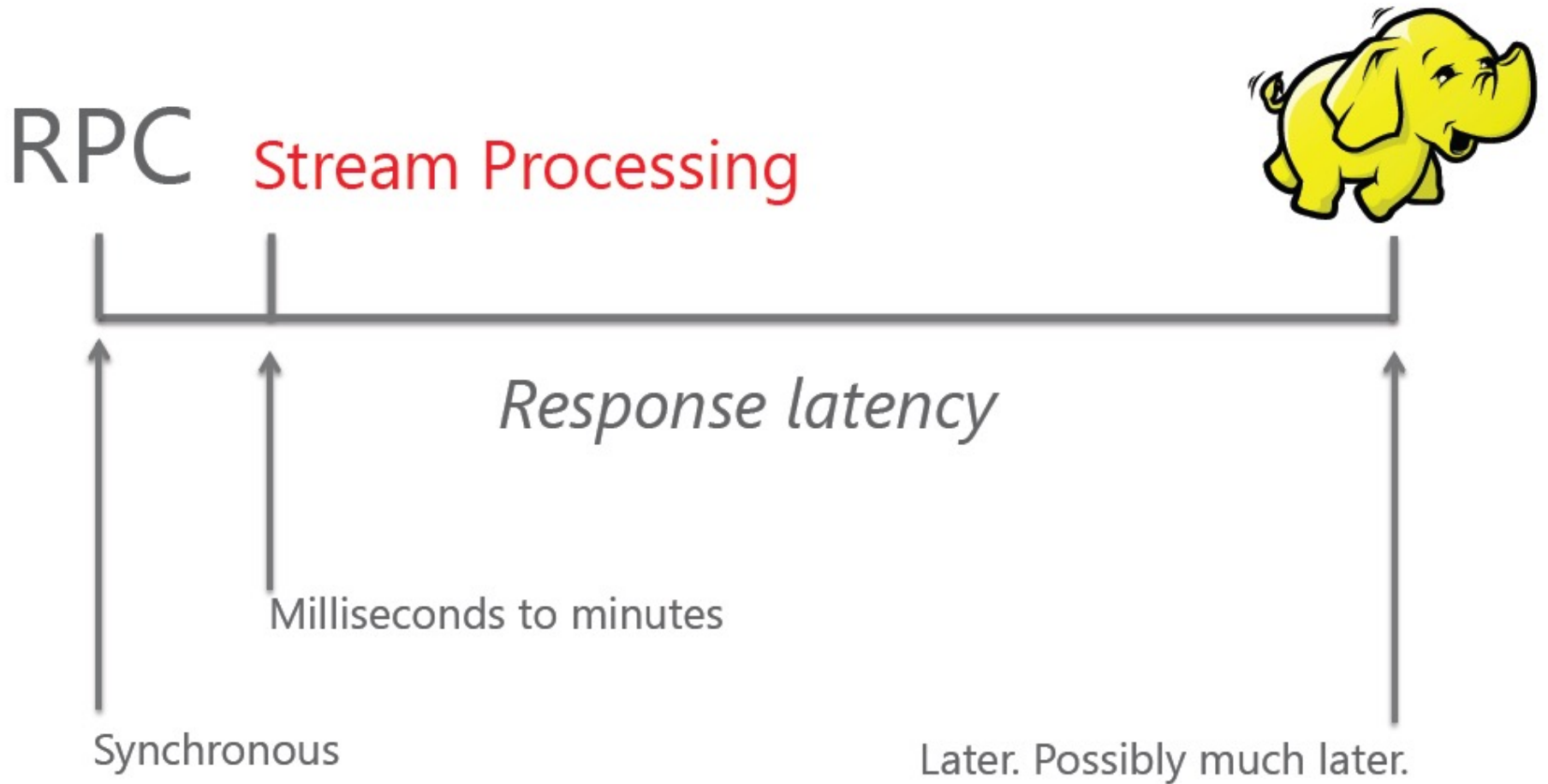
## Weakness

- High Latency
  - Minutes or even hours
  - Poor support for Interactive Analysis
  - Inability to Rapidly Respond to Special/ Unexpected Events

If a company can react to data more quickly, it can make more



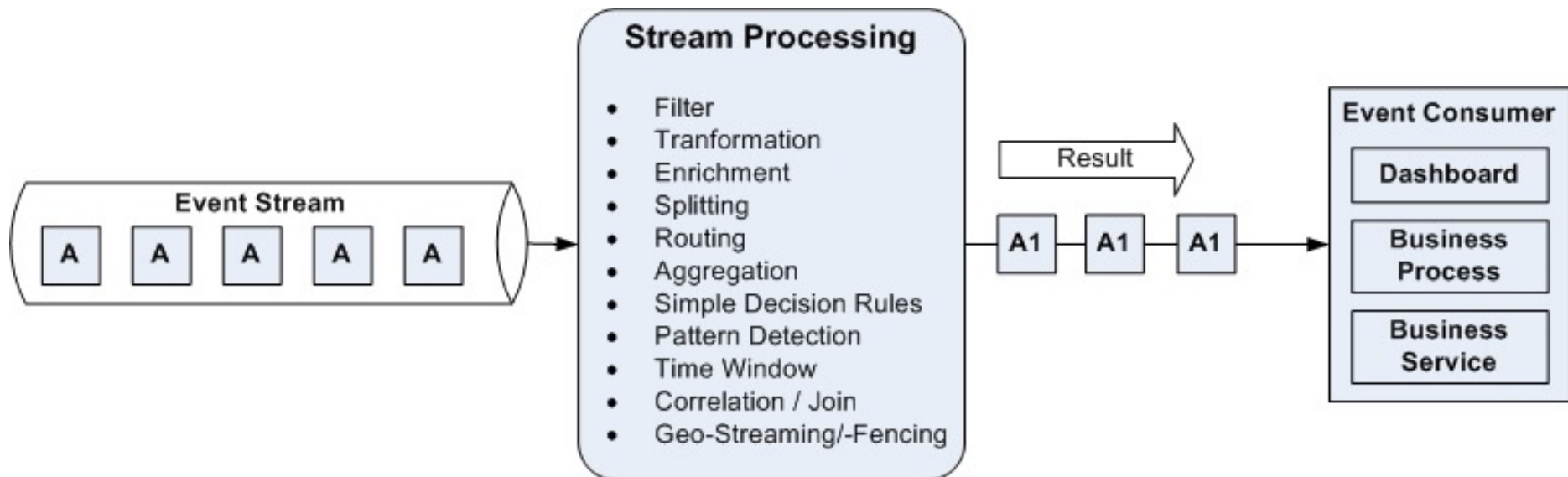
# Why Stream Processing ?



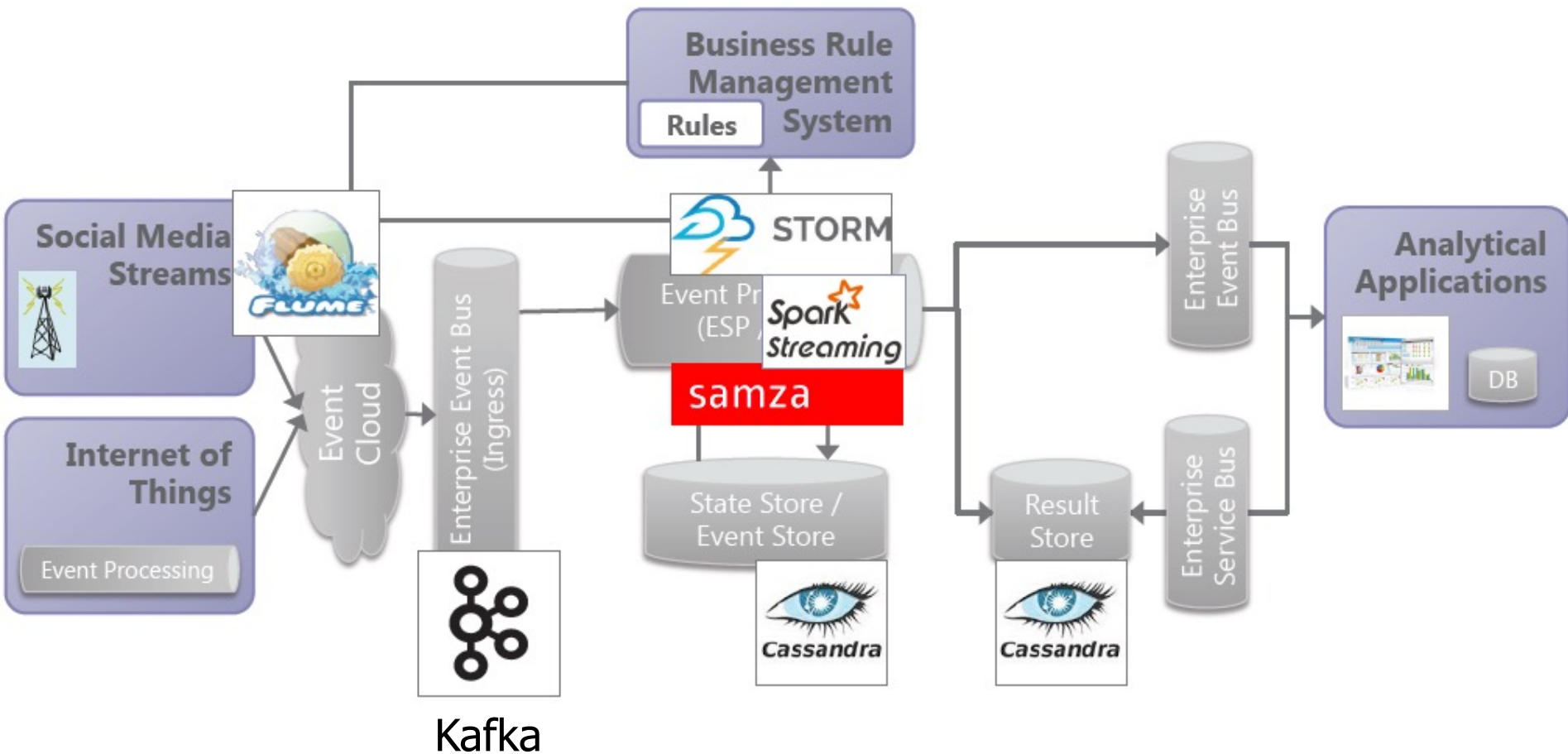


# What is Stream Processing ?

- Infrastructure for **continuous (non-stopped, never-ending)** data processing
- Computational model can be as general as MapReduce but with the ability to produce results under low-latency constraint
- Input Data collected continuously is naturally processed continuously
- Also known as Event Processing or Complex Event Processing (CEP)



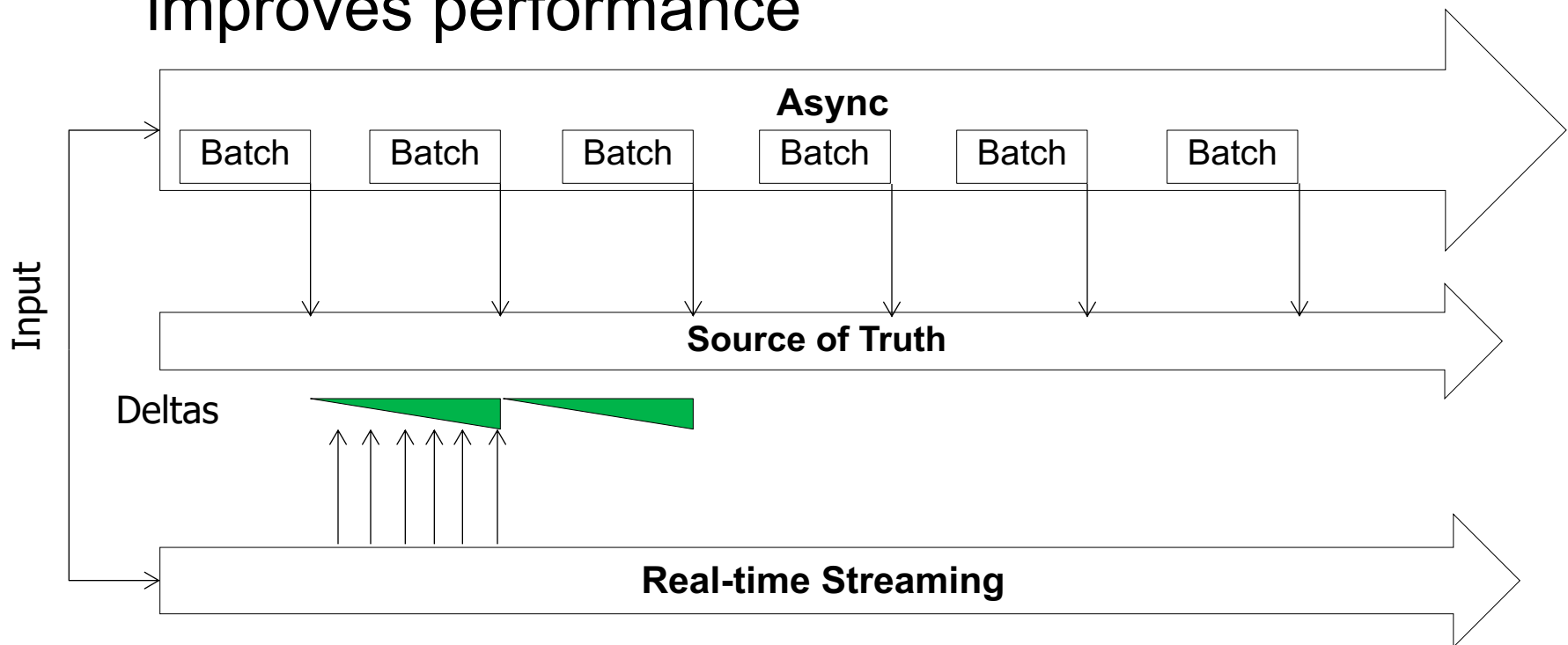
# Architectural Pattern #1: A Standalone Event Stream Processing System:



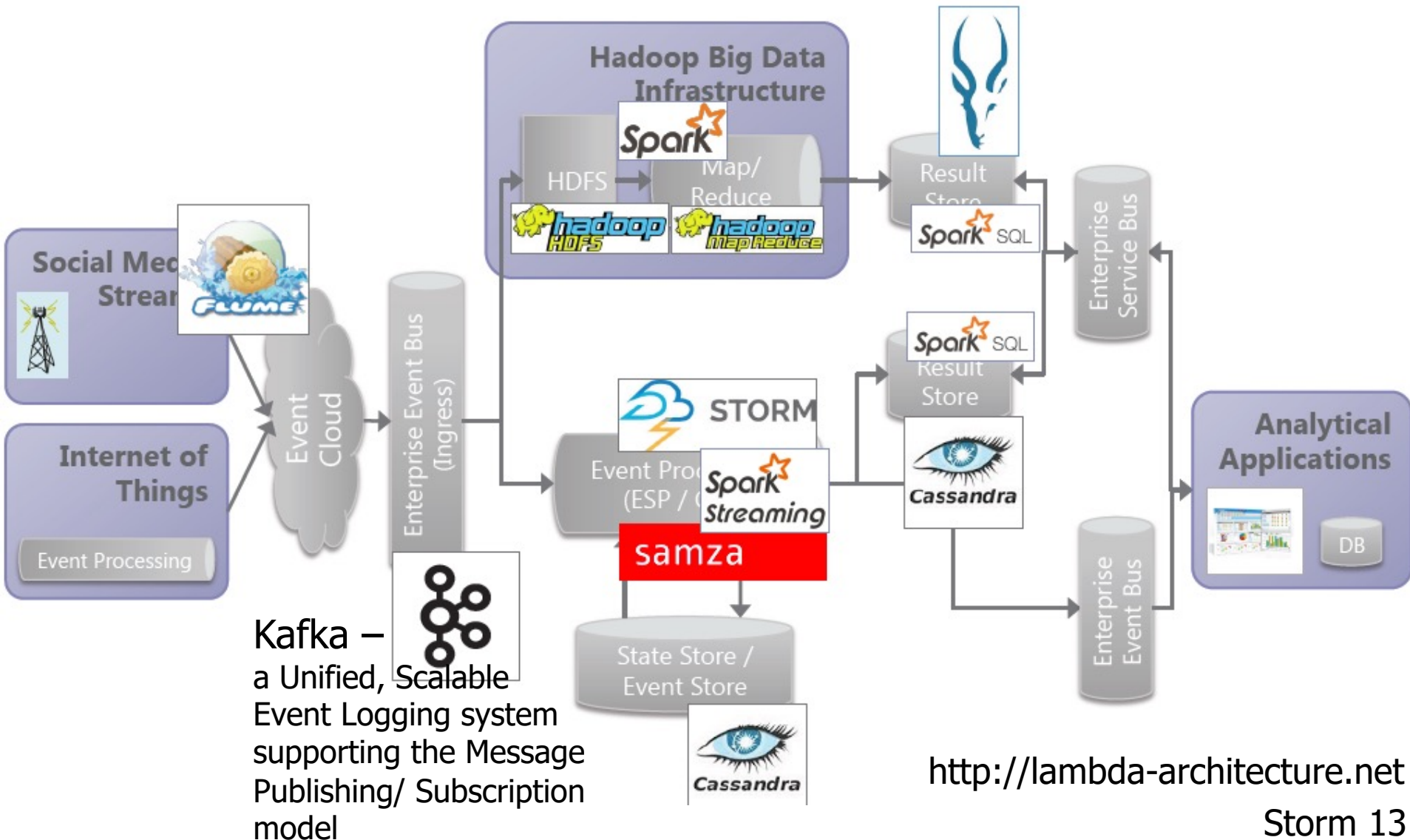
Architectural Patterns  
to support  
BOTH Real-Time and Batched  
Big Data Processing

# The Two-Pronged Approach

- <http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html>
- The interesting take-away: **Fast Real-Time path** with **Batch Backup** reduces complexity and improves performance



# Architectural Pattern #2: Event Stream Processing as part of the Lambda Architecture (proposed by Nathan Marz)

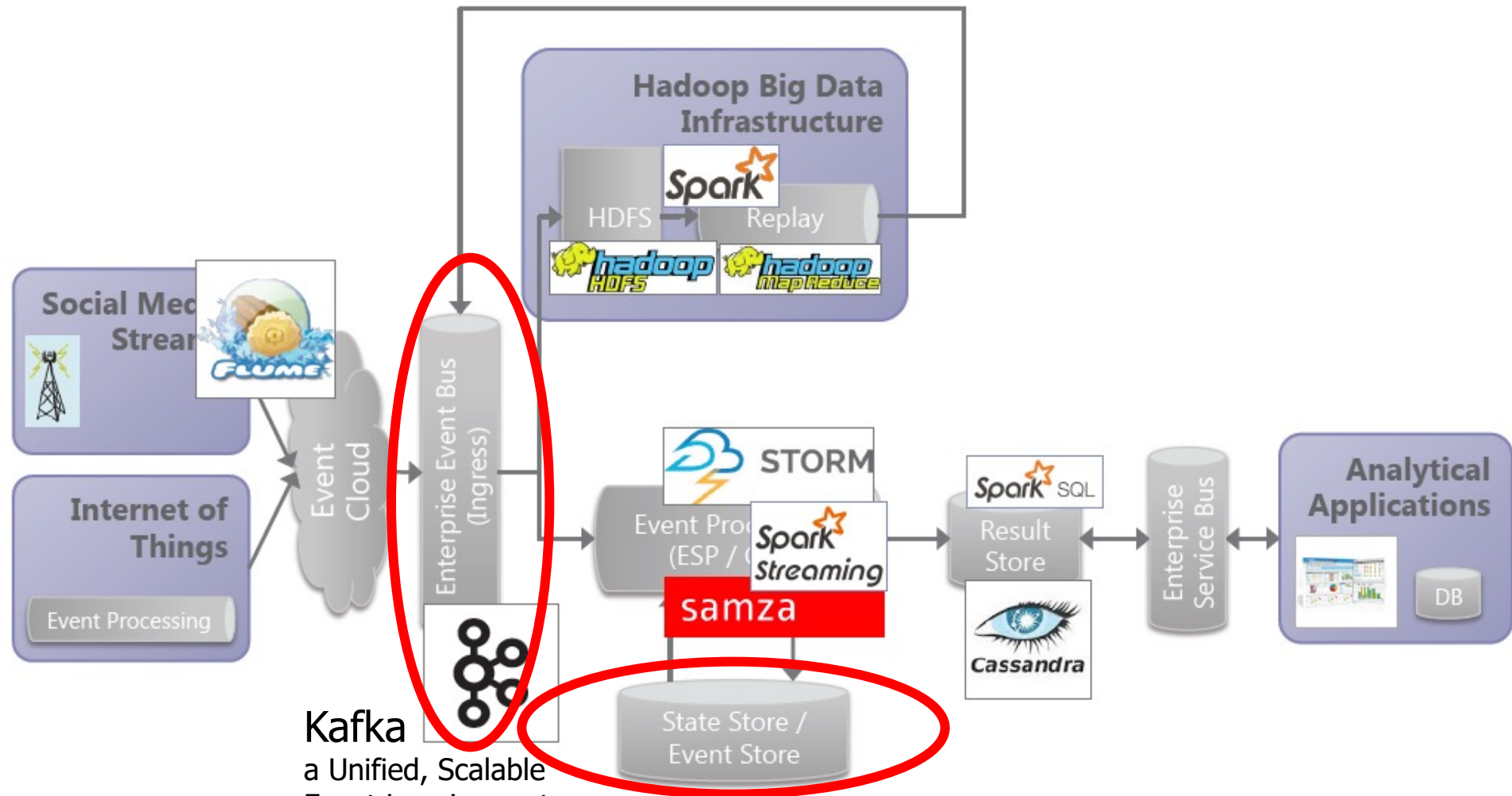


# Summing Bird

- <https://github.com/twitter/summingbird>

Write the same code (script) and then compile to be run on Storm as well as one Hadoop.

# Architectural Pattern #3: Event Stream Processing as part of the Kappa Architecture (from LinkedIn)



**Kafka**  
a Unified, Scalable  
Event Logging system  
supporting the message  
Publishing/ Subscription model

<http://milinda.pathirage.org/kappa-architecture.com/>

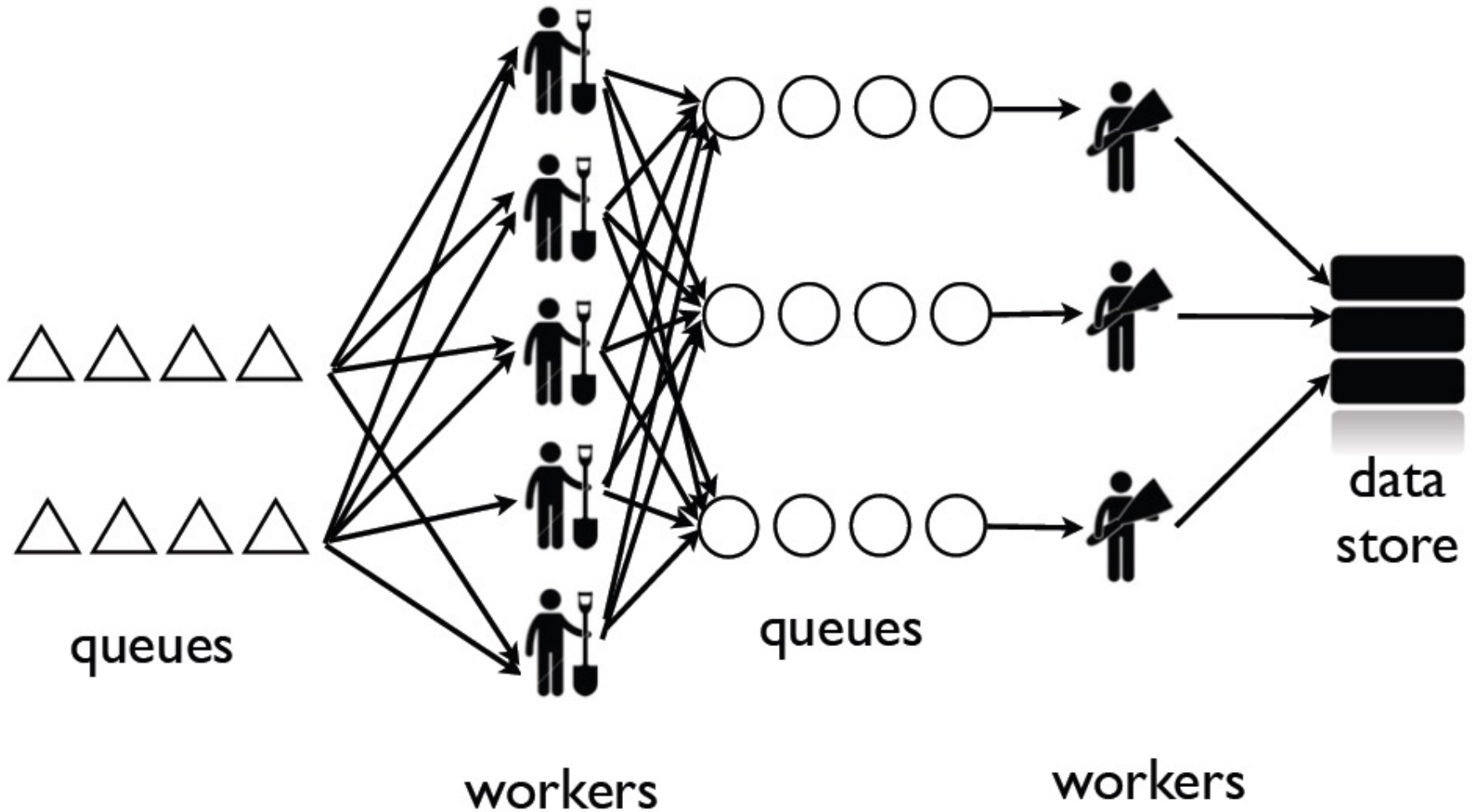
# Stream Processing with Apache Storm



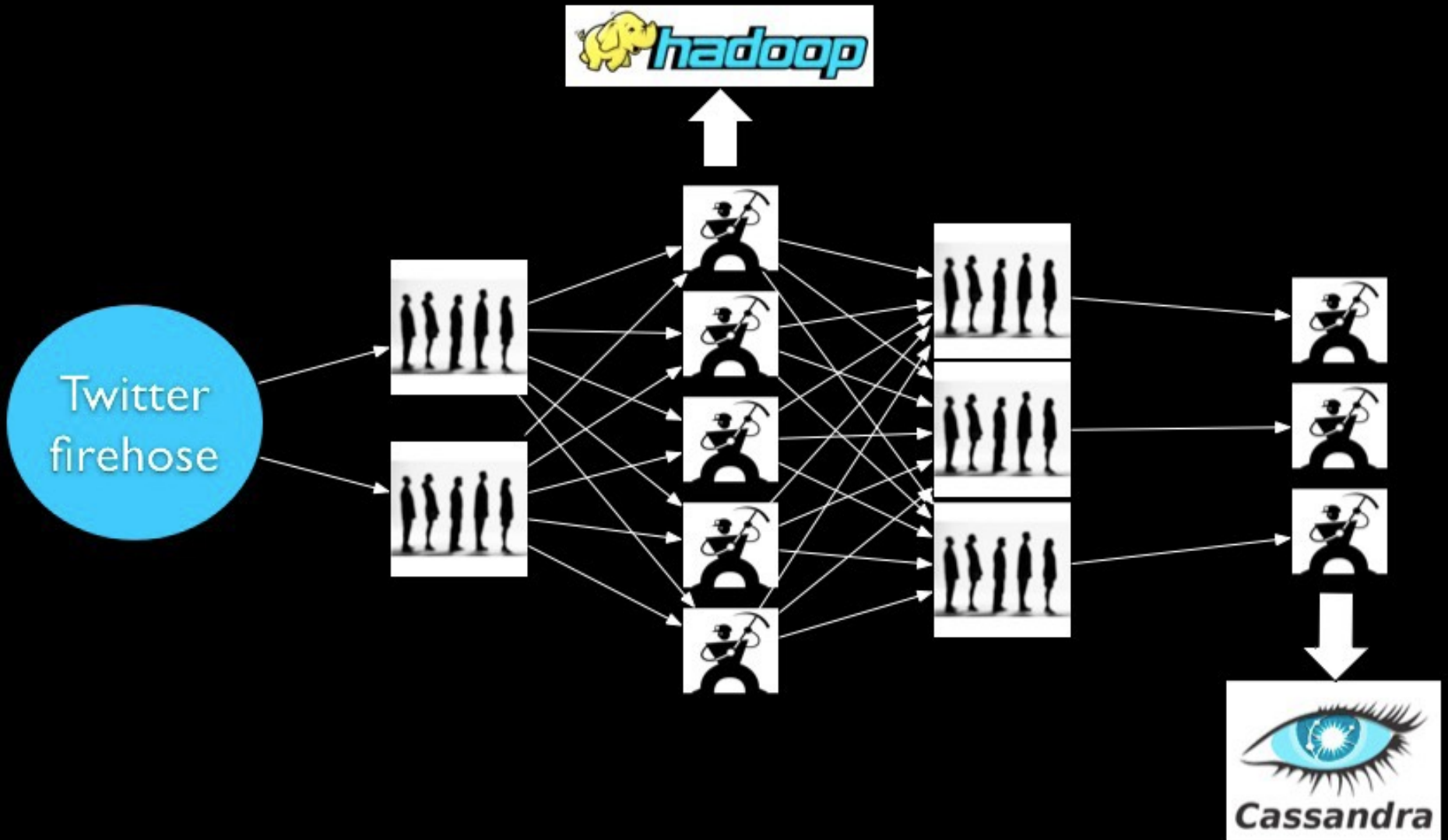
# Agenda

- Motivation for Stream Processing
- Apache Storm
  - Stream-based Programming Models and Examples
  - Different Flavors of Processing Guarantees
  - Additional Computational Models with Storm
    - Distributed Remote Procedure Call (DRPC)
    - Transactional Processing with Trident (over Storm)
  - Storm System Architecture
  - ~~■ Operational Guidelines for Storm~~
  - ~~■ Adoption Statistics and Real-time Use Cases~~
  - Future Extensions (Researchie)

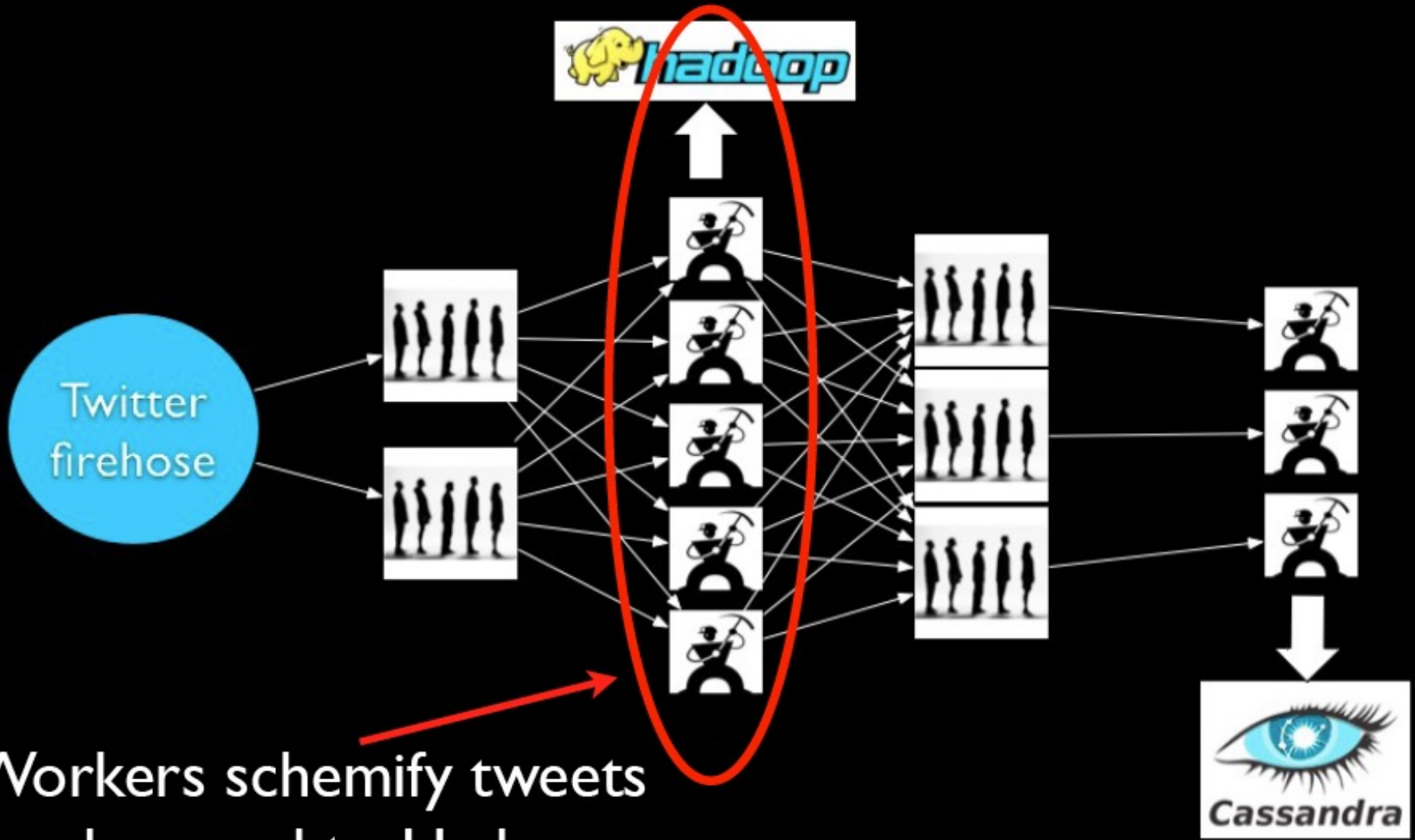
# Traditional Workflow under the Queues-Workers Model for Event Processing



# A Simplified Example

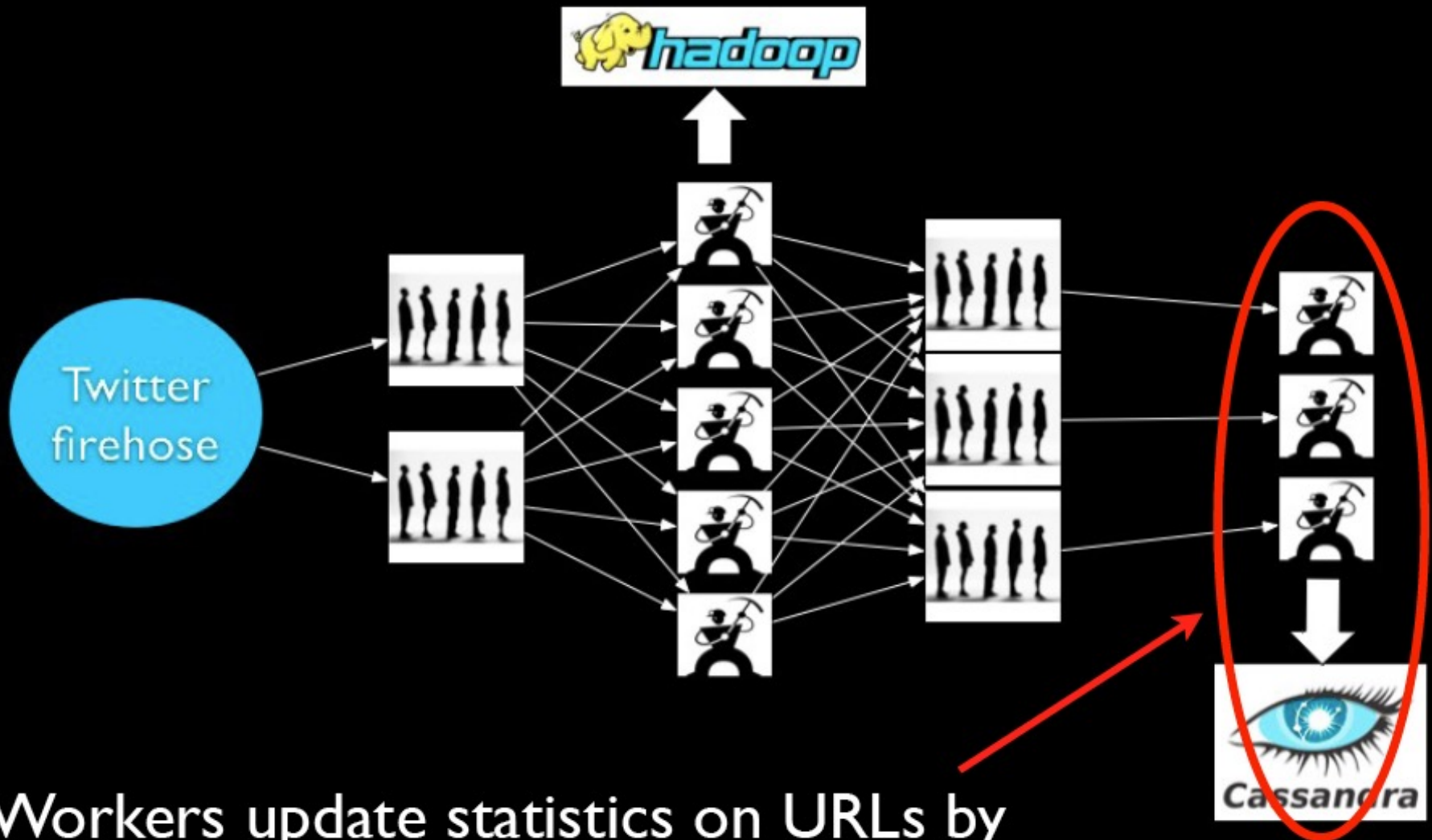


# A Simplified Example



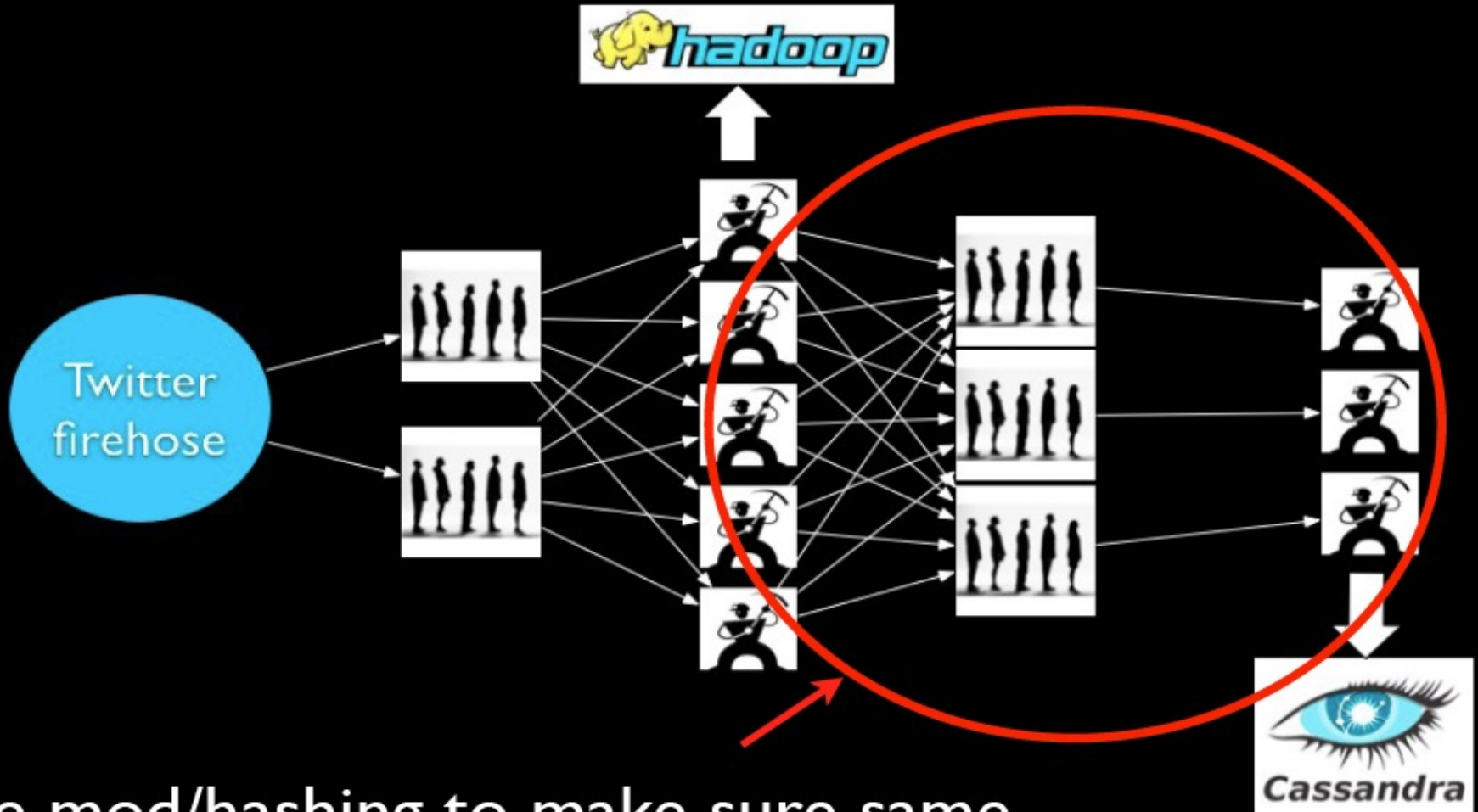
Workers schemify tweets  
and append to Hadoop

# A Simplified Example



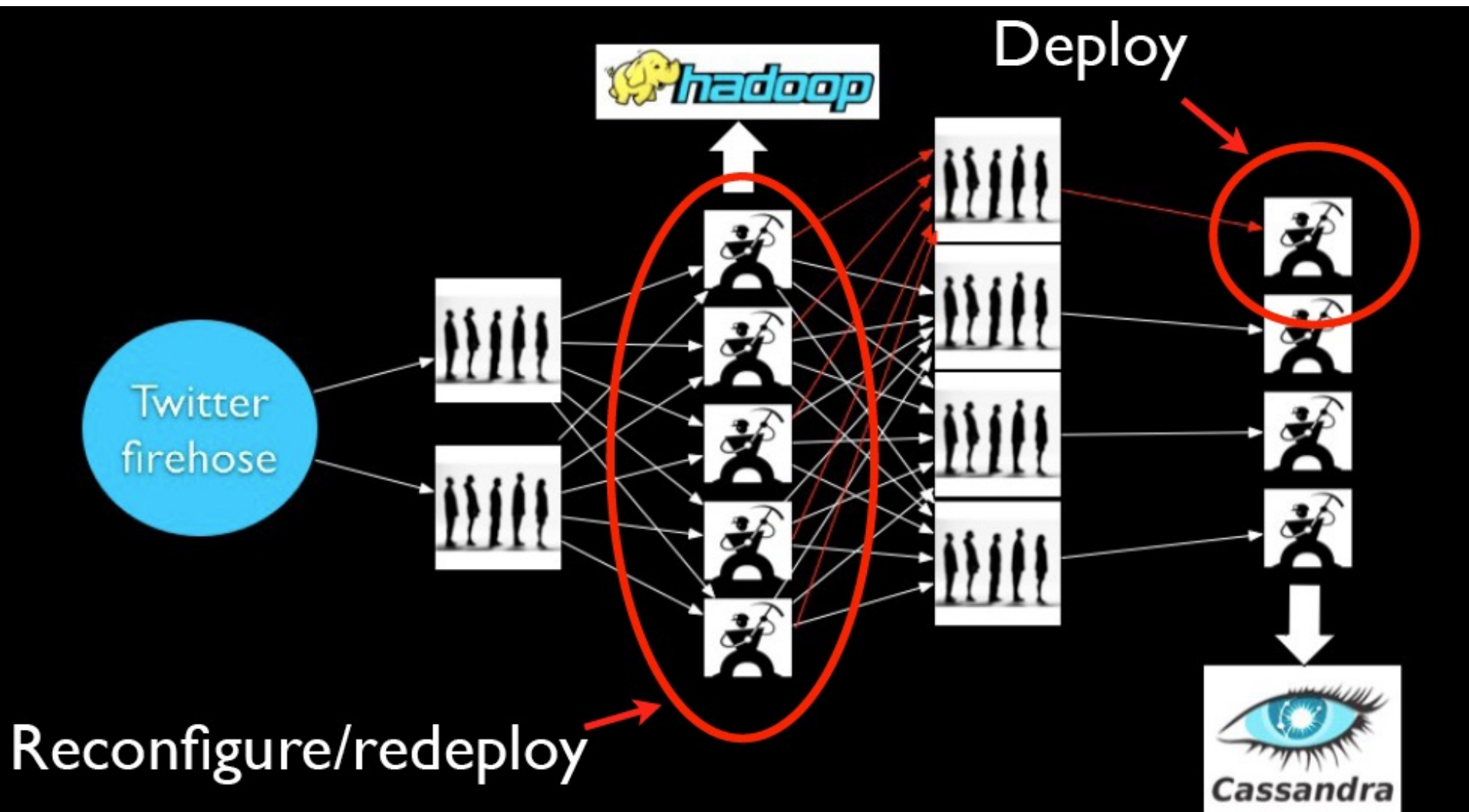
Workers update statistics on URLs by incrementing counters in Cassandra

# A Simplified Example



Use mod/ hashing to make sure same URL always goes to same worker

# The procedure to scale-up the system



# Problems of Traditional Workflow model

- Scaling is Painful as it involves Queue-partitioning and deployment of additional Workers (processes/nodes)
- Operational overhead due to Worker failures and Queue-Backups
- Coding is Tedious
- No guarantees on whether incoming Data is being processed



# A Solution: Apache Storm



- Focus on the support of Real-Time Streaming jobs
- To simplify dealing with queues (for tasks) and many workers (for load balancing/parallelization)
- Higher level abstraction than message passing
  - No intermediate message brokers!
- Support Guaranteed data processing (at least once) ; default: at most once
- Horizontal scalability ;
- Fault-tolerance
- Complementary to Hadoop:
  - The “Hadoop” of real time streaming jobs
  - The “**Summingbird**” system by Twitter can actually compile a single programming script into a Storm and Hadoop version separately
- Built by Nathan Marz et al at Backtype, acquired and hardened by Twitter in 2011 ; Open-sourced under Apache license since 2013
  - Written in Clojure (a dialect of LISP): **a Functional Programming Language** which generates bytecodes for JVM ;
  - Let users (programmers) program in Java and Clojure
- At Twitter, Storm had been decommissioned as of summer 2015. Storm has been replaced by Heron
  - Heron uses **the SAME programming abstraction** and is **100% API-compatible with Storm**
  - **Under Apache incubation (as of Feb 2019)** <https://github.com/apache/incubator-heron>

# Key Concepts in Storm

## Stream

Tuple

Tuple

Tuple

Tuple

Tuple

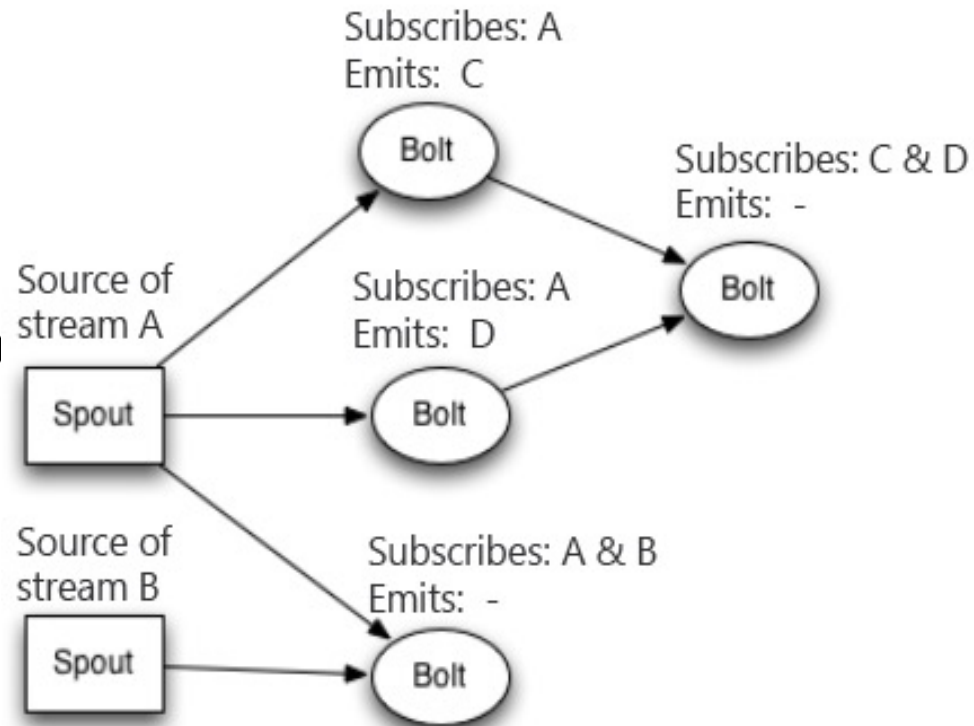
Tuple

Tuple

- An Unbounded sequence of Tuples
- Core Abstraction in Storm
- Defined with a Schema that names the fields in the Tuple
- Value must be serializable
- Every Stream has an ID

## Topologies

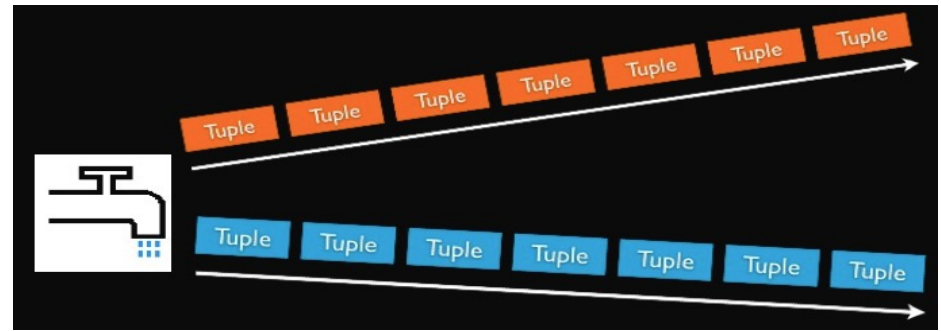
- A Directed Acyclic Graph (DAG) where each node is either a Data source (Spout) or a Processing node (Bolt)
- An Edge indicates which Bolt subscribes to which Stream



# Key Concepts in Storm (cont'd)

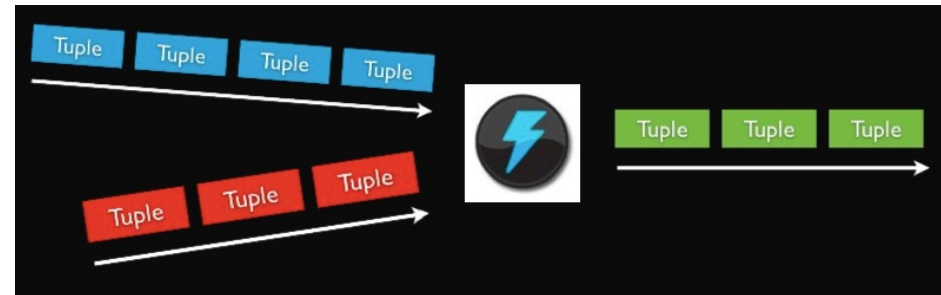
## Spout

- Source of data stream (tuples), e.g.
  - Read from the Twitter streaming API (tuples = tweets)
  - Read from a http server log (tuples = http requests)
  - Read from a Kafka queue (tuples = events)



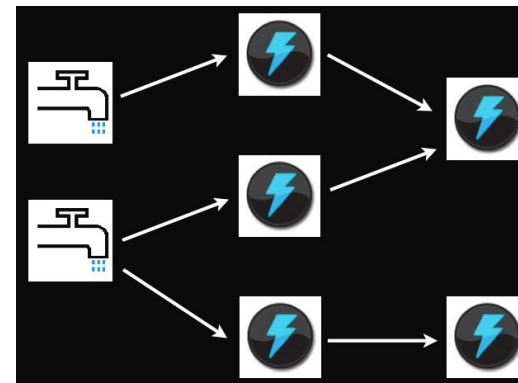
## Bolt

- Processes 1+ input stream(s) and produces 1+ new stream(s)
  - e.g. Calculate, Functions, Filters, Aggregation, Joins, talk to database



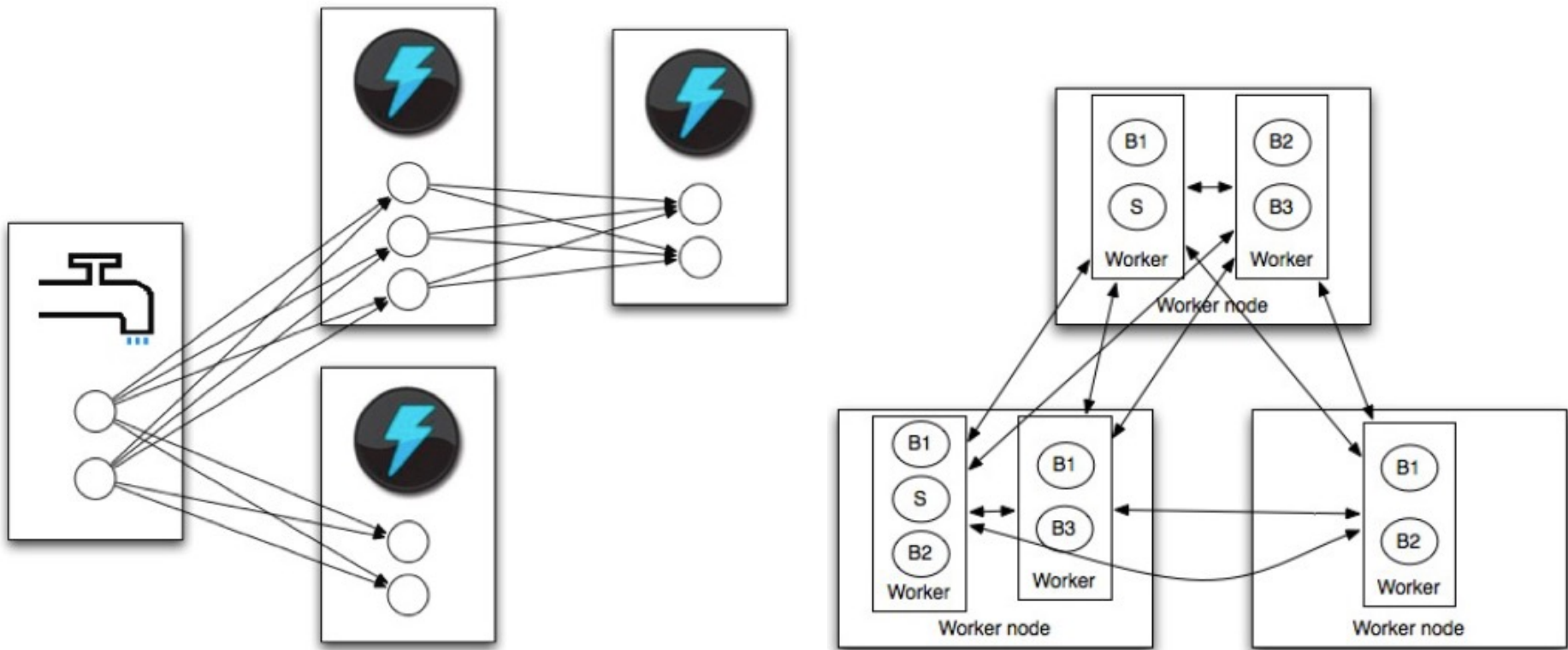
## A sample Storm Topology

- Compiled to be executed on many machines similar to a MapReduce job in Hadoop



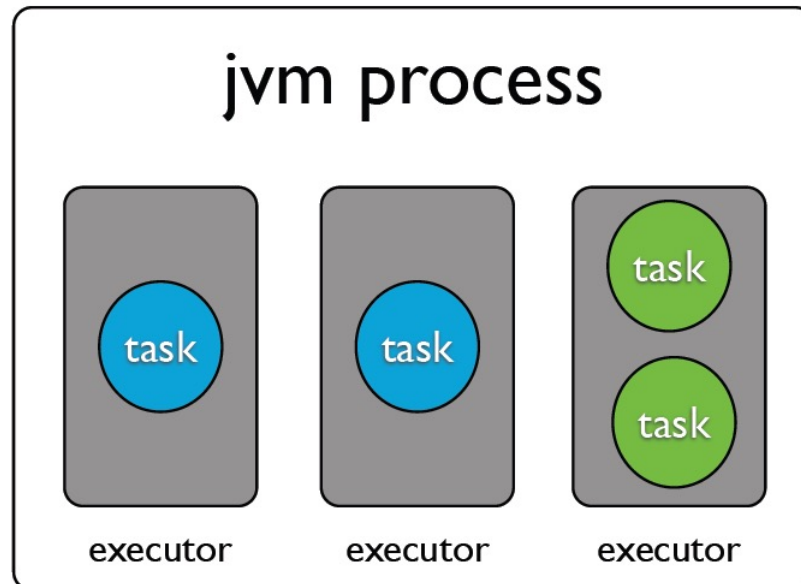
# Storm Tasks

- Each Spout or Bolt is executed as one or more Tasks (instances) across the cluster



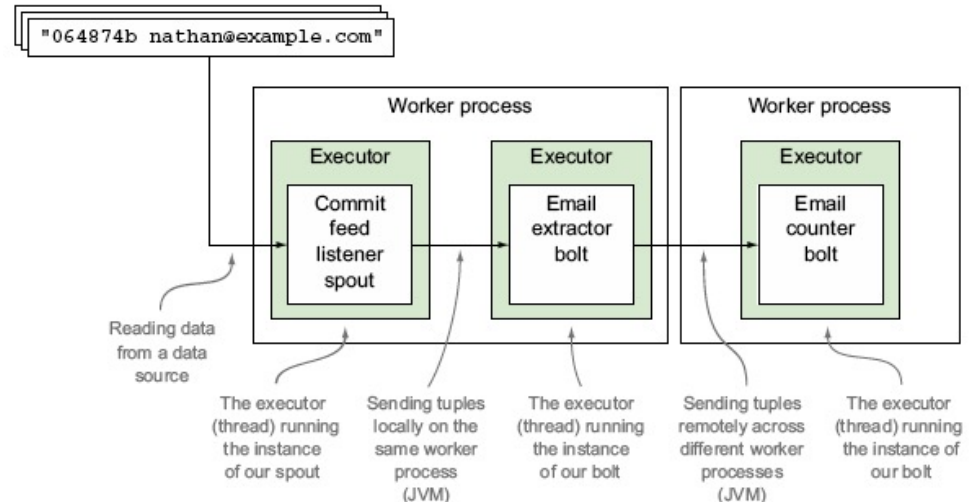
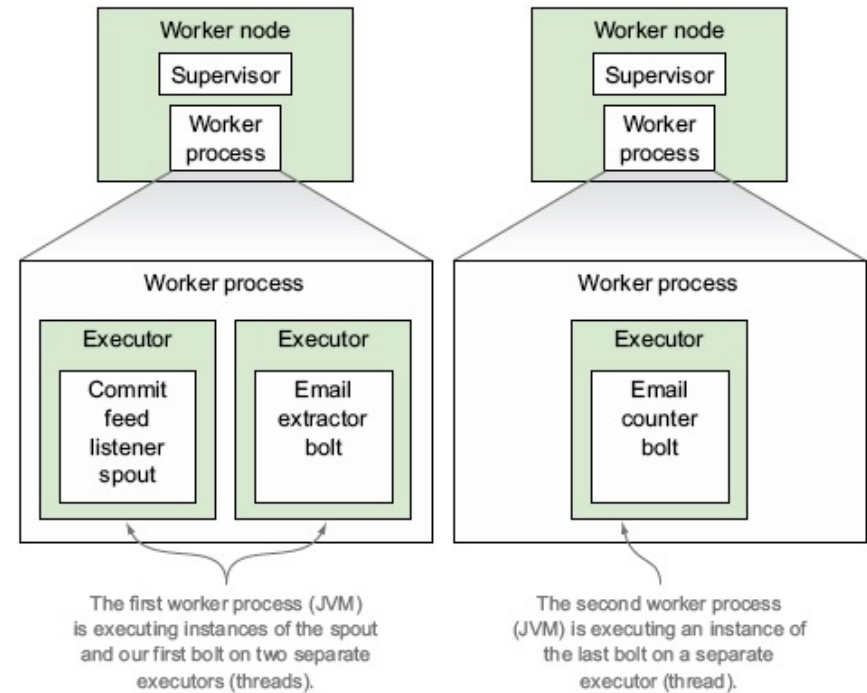
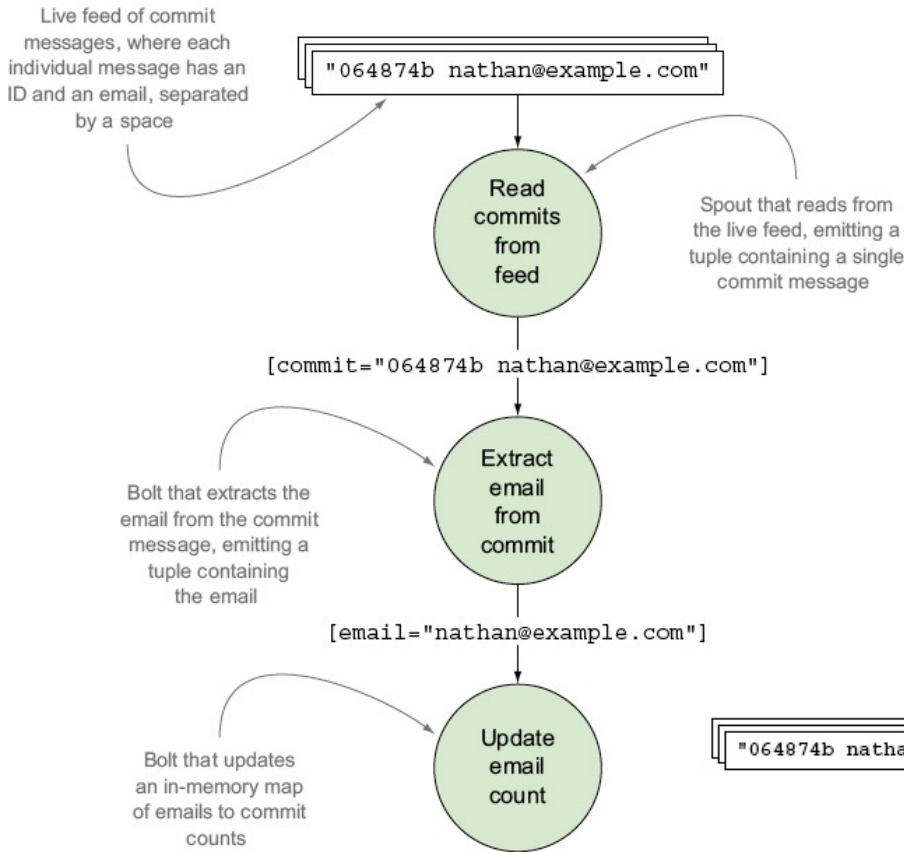
# Key Concepts in Storm (cont'd)

- Worker (JVM) Process
  - Executes subset of a Topology ;
  - May run 1 or more threads (Executors) for one or more components
  - One Thread per Executor
- Task
  - The actual data processing instance executing by the thread
  - It is possible for multiple tasks to share one thread (Why ? to facilitate dynamic scaling)

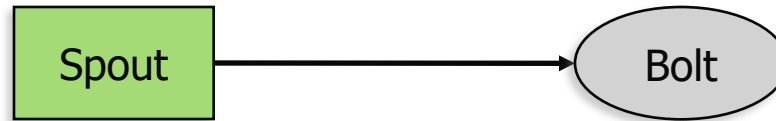


# An Example on deploying a Topology across a Cluster

Our topology is executing across two worker nodes (physical or virtual machines), where each worker node is running a single worker process (JVM).



# A trivial “Hello, Storm” topology



“emit random  
number < 100”



“multiply  
by 2”

**(74)**

**(148)**

# Code

## Spout

```
1 ▾ public void nextTuple() {  
2     final Random rand = new Random(); // normally this should be an instance field  
3     int nextRandomNumber = rand.nextInt(100);  
4     collector.emit(new Values(nextRandomNumber)); // auto-boxing  
5 }
```

## Bolt

```
1  Override  
2 ▾ public void prepare(Map conf, TopologyContext context, OutputCollector collector) {  
3     this.collector = collector;  
4 }  
5  
6 @Override  
7 ▾ public void execute(Tuple tuple) {  
8     Integer inputNumber = tuple.getInteger(0);  
9     collector.emit(tuple, new Values(inputNumber * 2)); // auto-boxing  
10    collector.ack(tuple);  
11 }  
12  
13 @Override  
14 ▾ public void declareOutputFields(OutputFieldsDeclarer declarer) {  
15    declarer.declare(new Fields("doubled-number"));  
16 }
```



# Code

## Topology config – for running *on your local laptop*

```
1 Config conf = new Config();
2 conf.setNumWorkers(1);
3
4 topologyBuilder.setSpout("my-spout", new MySpout(), 2);
5
6 topologyBuilder.setBolt("my-bolt", new MyBolt(), 2)
7     .shuffleGrouping("my-spout");
8
9 StormSubmitter.submitTopology(
10     "mytopology",
11     conf,
12     topologyBuilder.createTopology()
13     );
```

# Code

## Topology config – for running *on a production Storm cluster*

```
1 Config conf = new Config();
2 conf.setNumWorkers(200);
3
4 topologyBuilder.setSpout("my-spout", new MySpout(), 100);
5
6 topologyBuilder.setBolt("my-bolt", new MyBolt(), 200)
7     .shuffleGrouping("my-spout");
8
9 StormSubmitter.submitTopology(
10     "mytopology",
11     conf,
12     topologyBuilder.createTopology()
13 );
```

# Creating a spout

- Very often, it suffices to use an existing spout (Kafka spout, Redis spout, etc).
- But you usually needs to implement your own **bolts** to realize your specific computation.

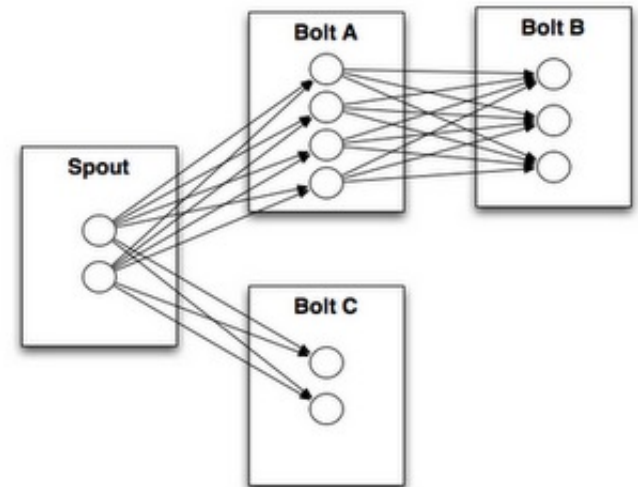
# Creating a Bolt

- Storm is polyglot – but we focus on Java.
- Two main options for JVM users:
  - Implement the [IRichBolt](#) or [IBasicBolt](#) interfaces
  - Extend the [BaseRichBolt](#) or [BaseBasicBolt](#) abstract classes
- `BaseBasicBolt`
  - Auto-acks the incoming tuple at the end of its `execute()` method.
  - With the right type of Spout (reliable one), “at-least-once” processing guarantee for each tuple is already supported automatically (and implicitly).
  - These bolts are typically simple functions or filters.
- `BaseRichBolt`
  - Allow one to specify complex tuple-anchoring/ack mechanism explicitly.
  - Need to use this type of bolt if one wants “at-most-once”, i.e. no guarantee in tuple-processing ;
  - You must – and are able to – manually `ack()` an incoming tuple.
  - Can be used to delay acking a tuple, e.g. for algorithms that need to work across multiple incoming tuples.

# Creating a topology

- When creating a topology you're essentially defining the DAG – that is, which spouts and bolts to use, and how they interconnect.
  - `TopologyBuilder#setSpout()` and `TopologyBuilder#setBolt()`
  - Groupings between spouts and bolts, e.g. `shuffleGrouping()`

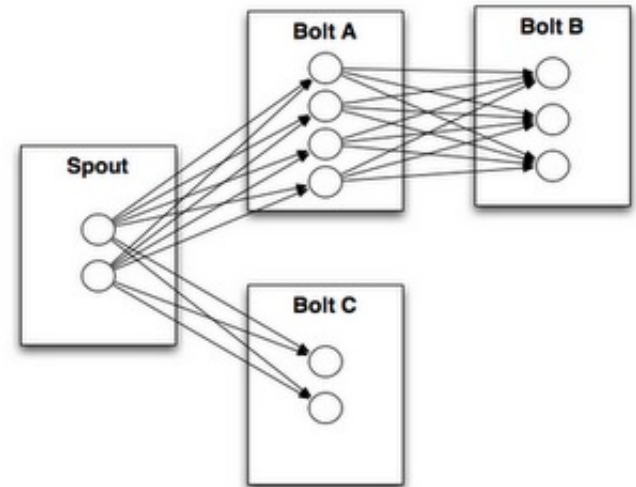
```
1 Config conf = new Config();
2 conf.setNumWorkers(200);
3
4 TopologyBuilder builder = new TopologyBuilder();
5 builder.setSpout("my-spout", new MySpout(), 100);
6 builder.setBolt("my-bolt", new MyBolt(), 200)
7     .shuffleGrouping("my-spout");
8 StormTopology topology = builder.createTopology();
```



# Creating a topology

- You must specify the initial *parallelism* of the topology.
  - Crucial for Performance & Scaling but no rule of thumb.
  - You must understand concepts such as workers/executors/tasks.
- Only some aspects of parallelism can be changed later, i.e. at run-time.
  - You can change the #executors (threads).
  - You cannot change #tasks, which remains static during the topology's lifetime.

```
1 Config conf = new Config();
2 conf.setNumWorkers(200);
3
4 TopologyBuilder builder = new TopologyBuilder();
5 builder.setSpout("my-spout", new MySpout(), 100);
6 builder.setBolt("my-bolt", new MyBolt(), 200)
7     .shuffleGrouping("my-spout");
8 StormTopology topology = builder.createTopology();
```



# Creating/ Submitting a topology

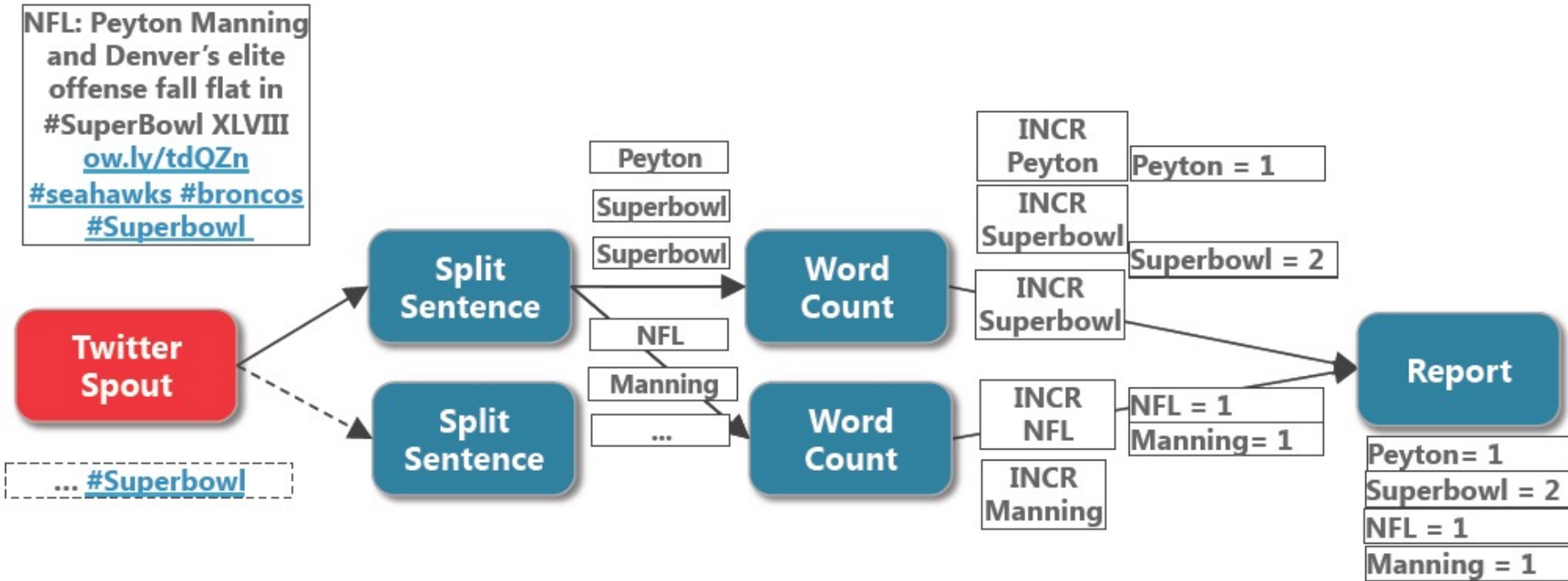
- You submit a topology either to a “local” cluster or to a real cluster.
  - LocalCluster#submitTopology
  - StormSubmitter#submitTopology() and #submitTopologyWithProgressBar()
  - In your code you may want to use both approaches, e.g. to facilitate local testing.
- Notes
  - A StormTopology is a static, serializable Thrift data structure. It contains instructions that tell Storm how to deploy and run the topology in a cluster.
  - The StormTopology object will be *serialized*, including *all* the components in the topology's DAG.
  - Only when the topology is *deployed* (and serialized in the process) and *initialized* (i.e. prepare() and other life cycle methods are called on components such as bolts) does it perform any actual message processing.

# Alright, my topology runs – now what?

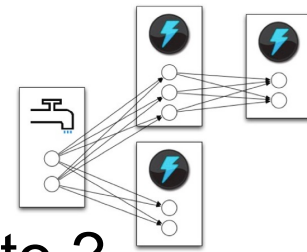
- The topology will run forever or until you kill it.
- Check the status of your topology
  - Storm UI (default: 8080/tcp)
  - Storm CLI, e.g. `storm [list | kill | rebalance | deactivate | ...]`
  - Storm REST API
- FYI:
  - Storm will guarantee that no data (tuple) is lost, even if machines go down and messages are dropped (as long as you don't disable this feature).
  - But if you store states using your own variables in the bolts/ spouts, the state information would be lost when the bolts/spouts die/ crashes
    - One (new) way to overcome this is to use “Stateful Bolts with Automatic Checkpointing” by extending from the `BaseStatefulBolt` class available from the recently released Storm Ver1.0.
  - Storm will automatically restart failed tasks, and even re-assign tasks to different machines if e.g. a machine dies.



# Another Example: Word Counting with Storm



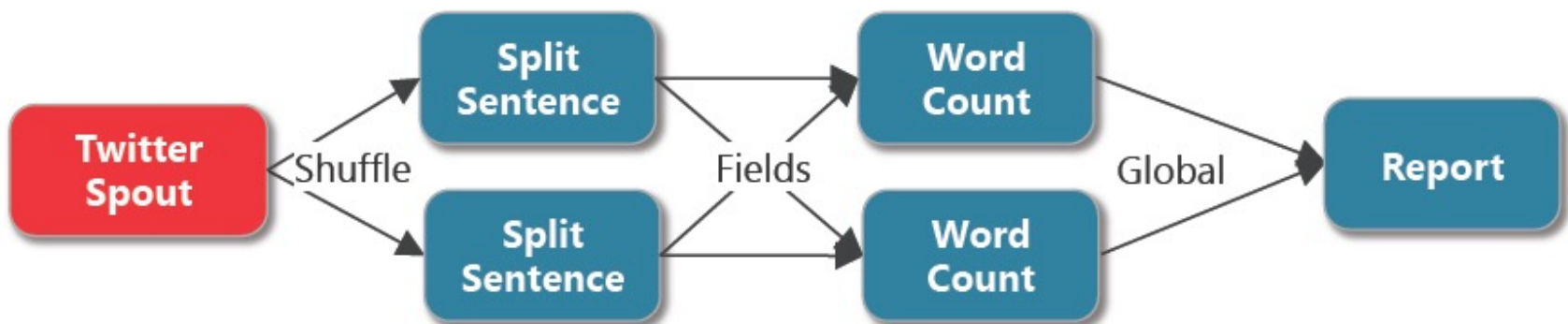
# How/ Where to route a tuple?



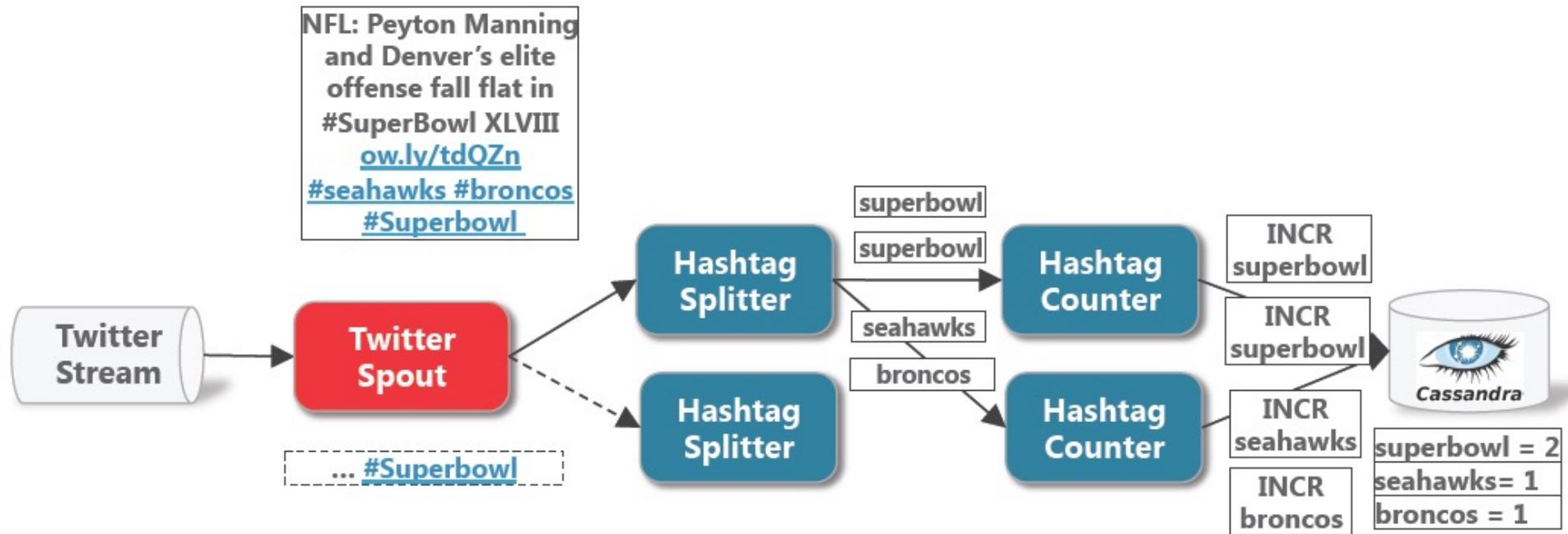
When a tuple is emitted, which task should it be routed to ?  
Ans: It is User-Programmable

<b>Shuffle grouping</b>	is random grouping
<b>Fields grouping</b>	is grouped by value, such that equal value results in equal task
<b>All grouping</b>	replicates to all tasks
<b>Global grouping</b>	makes all tuples go to one task
<b>None grouping</b>	makes bolt run in the same thread as bolt/spout it subscribes to
<b>Direct grouping</b>	producer (task that emits) controls which consumer will receive
<b>Local or Shuffle grouping</b>	similar to the shuffle grouping but will shuffle tuples among bolt tasks running in the same worker process, if any. Falls back to shuffle grouping behavior.

- e.g. For the previous Superbowl Tweet Analysis Example:



# Using a NoSQL database for storing the results (Keeping state with the counter-type columns)



Here, Cassandra serves as a persistent datastore so that the accumulated counter statistics can survive the crashing of some of the topology's components e.g. one or more of the Hashtag Counter bolt(s)

# WordCount in Storm (part of the code)

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("spout", new KestrelSpout("kestrel.twitter.com",
    22133, "sentence_queue", new StringScheme()),5);
builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 12)
    .fieldGrouping("split", new Fields("word"));

//=====
public static class SplitSentence extends ShellBolt implements IRichBolt {
    //Code to split a sentence
}

public static class WordCount implements IBasicBolt {
    //Code to count words, have to override the execute method
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        //...
    }
}
//=====
StormSubmitter.submitTopology("word-count", builder.createTopology());
```

**Parallelism Degree**  
=(Number of threads for a Spout or Bolt) ;  
Default = 1 task per thread ;  
Can be overridden by setTaskNum( )

# Actual Code to Create the Topology of the Example

```
public class WordCountTopology {

    private static final String SENTENCE_SPOUT_ID = "sentence-spout";
    private static final String SPLIT_BOLT_ID = "split-bolt";
    private static final String COUNT_BOLT_ID = "count-bolt";
    private static final String REPORT_BOLT_ID = "report-bolt";
    private static final String TOPOLOGY_NAME = "word-count-topology";

    public static void main(String[] args) throws Exception {
        SentenceSpout spout = new SentenceSpout();
        SplitsentenceBolt splitBolt = new SplitsentenceBolt();
        WordCountBolt countBolt = new WordCountBolt();
        ReportBolt reportBolt = new ReportBolt();

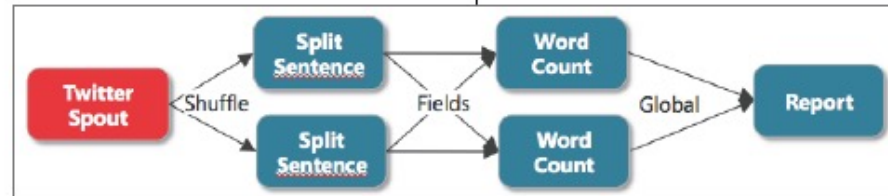
        TopologyBuilder builder = new TopologyBuilder();

        builder.setSpout(SENTENCE_SPOUT_ID, spout, 2);
        builder.setBolt(SPLIT_BOLT_ID, splitBolt, 2).setNumTasks(4).shuffleGrouping(SENTENCE_SPOUT_ID);
        builder.setBolt(COUNT_BOLT_ID, countBolt, 4).fieldsGrouping(SPLIT_BOLT_ID, new Fields("word"));
        builder.setBolt(REPORT_BOLT_ID, reportBolt).globalGrouping(COUNT_BOLT_ID);

        Config config = new Config();
        config.setNumWorkers(2);

        if (args.length == 0) {
            LocalCluster cluster = new LocalCluster();

            cluster.submitTopology(TOPOLOGY_NAME, config, builder.createTopology());
            waitForSeconds(10);
            cluster.killTopology(TOPOLOGY_NAME);
            cluster.shutdown();
        } else {
            StormSubmitter.submitTopology(args[0], config, builder.createTopology());
        }
    }
}
```

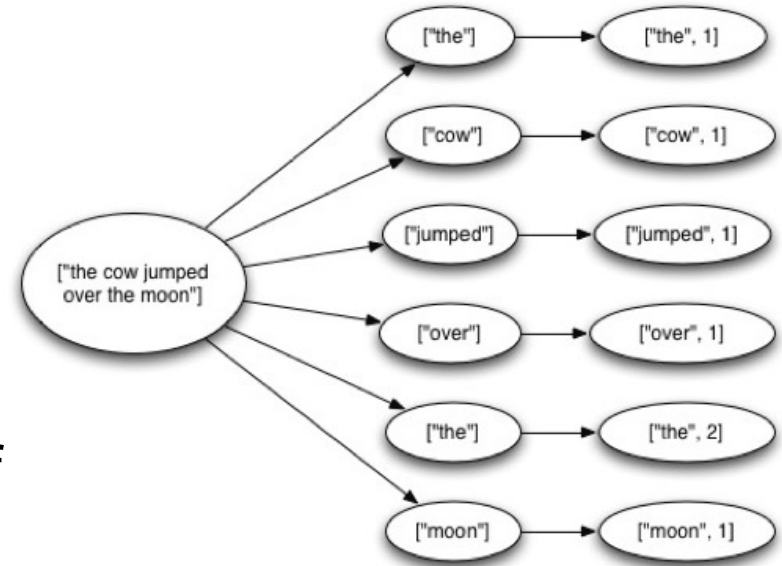


# Storm supports 3 different flavors of Message/Tuple Processing Guarantee

1. No Guarantee at all (like S4 of Yahoo)
2. At Least Once -- i.e. it is possible for some tuple(s) to be repeatedly processed by the topology more than once
3. Exactly Once (like Transaction)
  - but this feature has been deprecated
  - Now, one should use Trident (is built on the top of Storm) to support Transaction-oriented processing

# At Least Once

- Tuple Tree
- A spout tuple is not fully processed until all tuples in the tree have been completed
- If the tuple tree is not completed within a specified timeout, the spout tuple is replayed
- Uses acker tasks to keep track of tuple progress
- Reliability API for the user:



```
public void execute(Tuple tuple) {  
    String sentence = tuple.getString(0);  
    for(String word: sentence.split(" ")) {  
        _collector.emit(tuple, new Values(word));  
    }  
    _collector.ack(tuple);  
}
```

“Anchoring”  
creates a new  
edge in the tuple  
tree

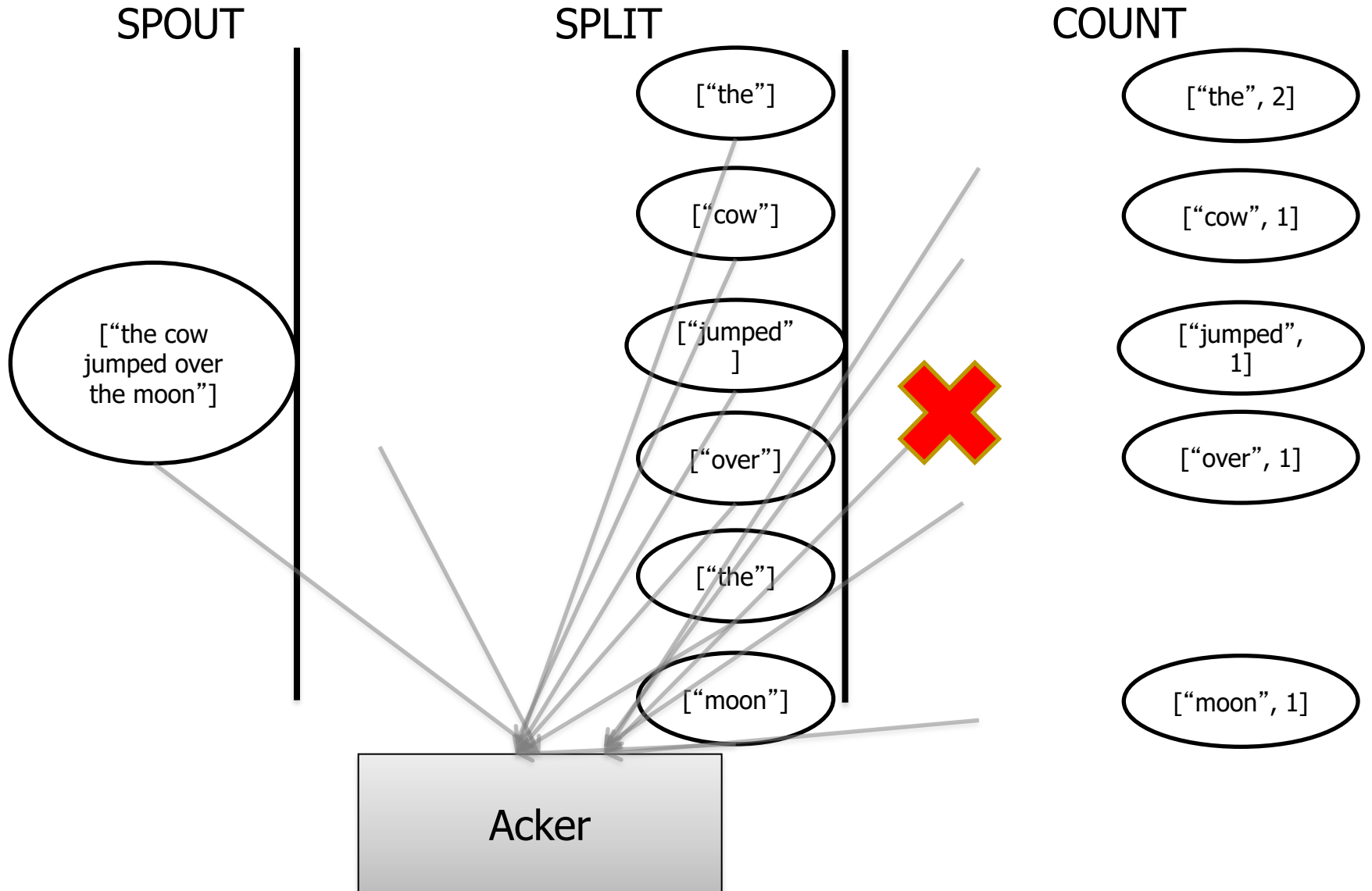
Marks a single  
node in the tree as  
complete

# At Least Once

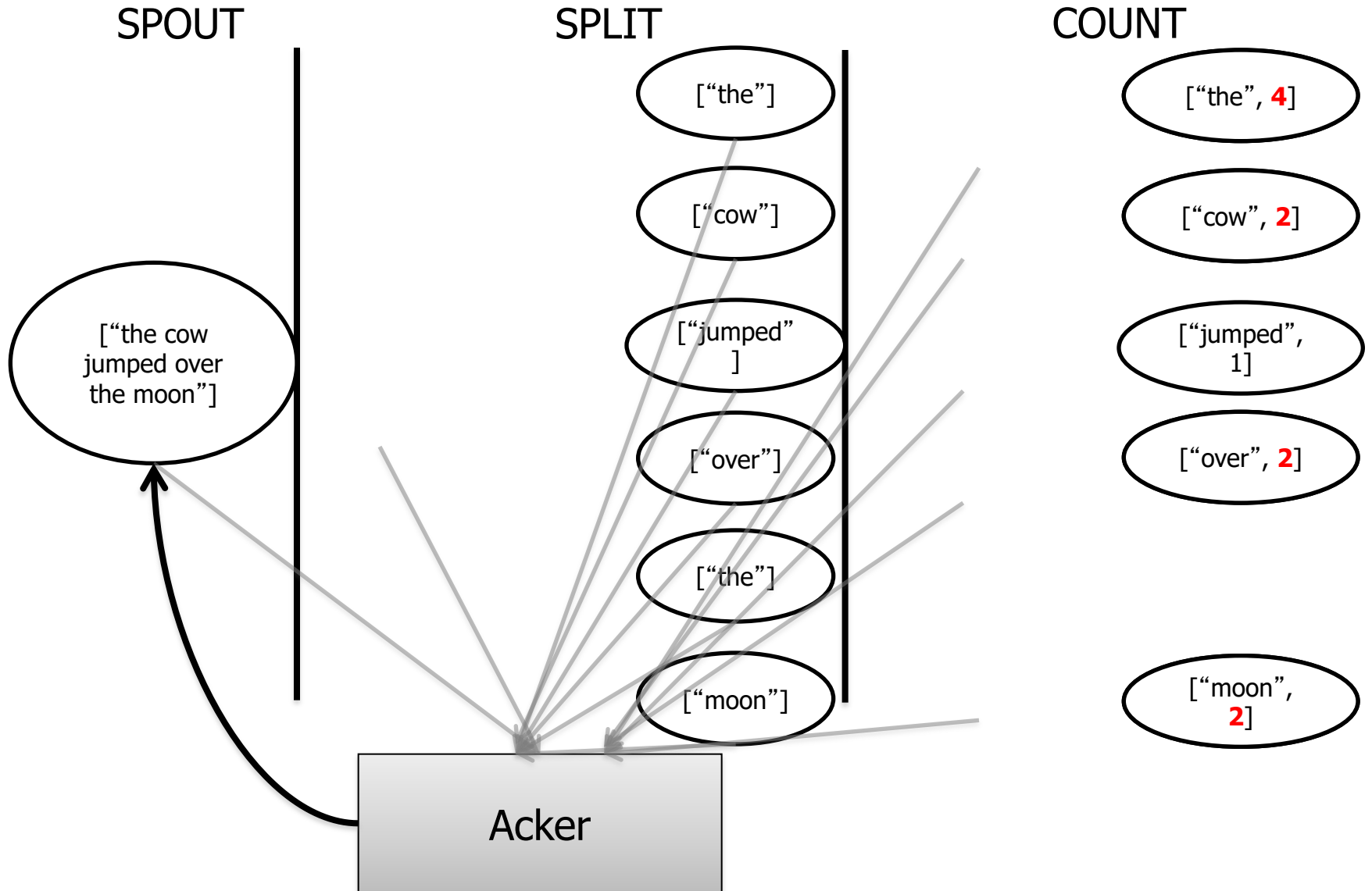
- What happens if there is a failure?
  - You can double process events.
  - This is not so critical if you have something like Hadoop to back you up and correct the issue later.
  - Or if you are looking at statistical trends and replay does not happen that often.
- This requires you to have a spout that supports replay. Not all messaging infrastructure does.



# At Least Once



# At Least Once



# Exactly Once - Transactional Topologies (Deprecated in Storm ; Use Trident instead)

- Transactional Topologies provide a strong ordering of processing.
- A small batch of tuples are processed at a time.
- Each batch completely succeeds or completely fails.
- Each batch is “committed” in order.
- Partial processing is pipelined
- Requires the spout to be able to replay a batch.

# Additional Computation models with Storm

## ■ Storm DRPC

- Parallelize the computation of really intense functions on the fly.
- Input is a stream of function arguments, and output is a stream of the results for each of those function calls.

## ■ Storm Trident

- High-level abstraction on top of Storm, which intermixes high throughput and stateful stream processing with low latency distributed querying.
  - Joins, aggregations, grouping, functions, filters.
  - Adds primitives for doing stateful, incremental processing on top of any database or persistence store.
- Has consistent, exactly-once semantics.
- Processes a stream as small batches of messages
  - (cf. Spark Streaming)

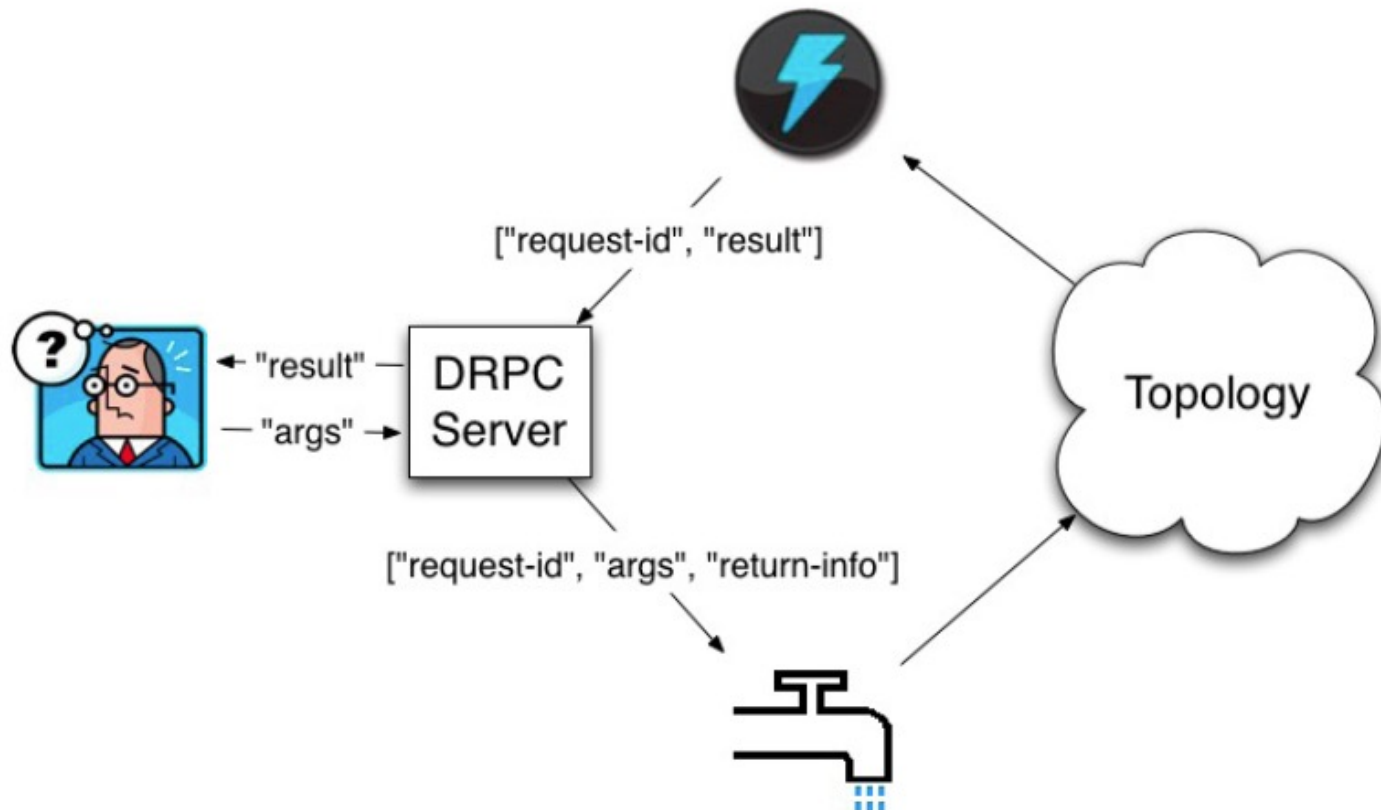
# Distributed Remote Procedure Call (DRPC)

# DRPC

- Distributed Remote Procedure Call
- Turn an RPC call into a tuple sent from a spout
- Take a result from that and send it back to the user.

# DRPC

- Distributed Remote Procedure Call
- Turn an RPC call into a tuple sent from a spout
- Take a result from that and send it back to the user.



Trident



# But What About State

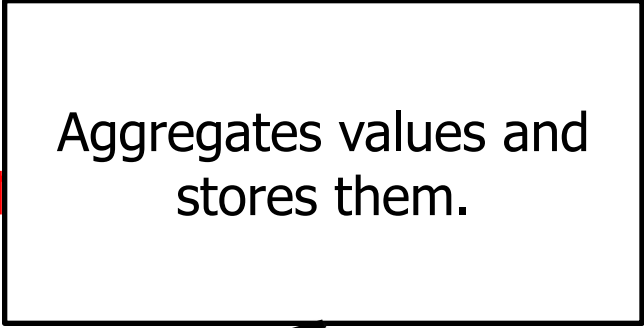
- For most cases, state storage in Storm is left up to you (the programmer).
- If your Bolt goes down after accumulating 3 weeks of aggregated data that you have not stored any where -- too bad.

# Enter Trident

- Trident is a high-level abstraction for doing Real-Time computing on the top of Storm
  - Similar to high-level batch processing tools like Pig or Cascading
- Provides Exactly-Once semantics like transactional topologies.
- In Trident, state is a first class citizen, but the exact implementation of state is up to you.
  - There are many pre-built connectors to various NoSQL stores like HBase
- Provides a high level API (similar to cascading for Hadoop)

# Trident Example

```
TridentTopology topology =  
    new TridentTopology();  
TridentState wordCounts =  
    topology.newStream("spo  
        .each(new Fields("sender  
new Fields("word"))  
        .groupBy(new Fields("word"))  
        .persistentAggregate(new  
MemoryMapState.Factory(), new Count(), new  
Fields("count"))  
        .parallelismHint(6);
```



Aggregates values and stores them.

# Trident Example

```
public class Split extends BaseFunction {  
    public void execute(TridentTuple tuple,  
TridentCollector collector) {  
        String sentence = tuple.getString(0);  
        for(String word: sentence.split(" ")) {  
            collector.emit(new Values(word));  
        }  
    }  
}
```

No Acking Required

# Storm System Architecture

- Nimbus

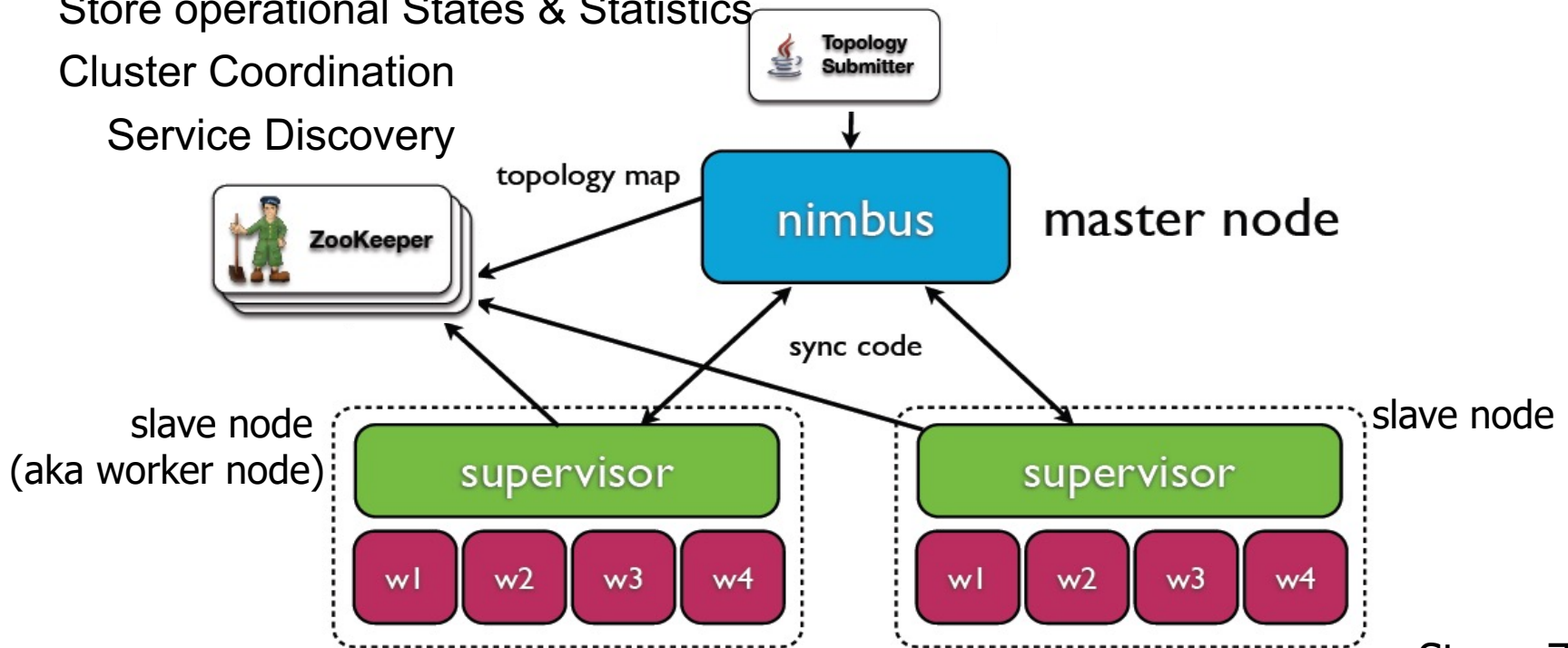
- Master node (like the Job Tracker in Hadoop ver1.0)
- Manage Topologies ; Distribute Code ; Assign Tasks ; Monitor Failure

- Supervisor

- Runs on Slave nodes (aka Worker nodes) ; listen to assignment and then launch & manage Worker (JVM) processes
- Coordinate with Zookeeper for Fault-Tolerance/ Synchronization, etc

- Zookeeper

- Store operational States & Statistics
- Cluster Coordination
- Service Discovery



## A Summary on What Storm does

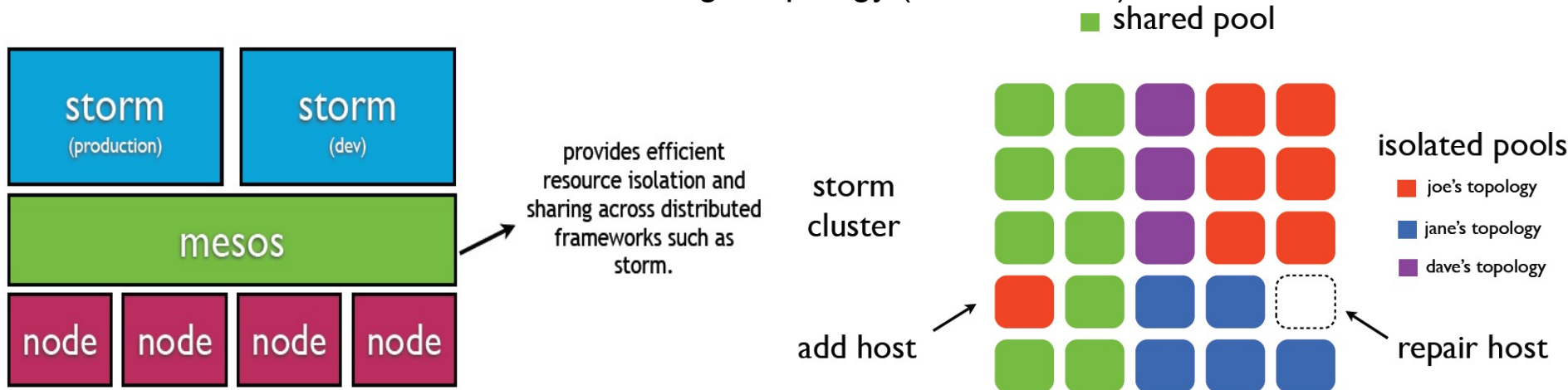
- Distribute Code and Configuration
- Robust Process Management
- Monitors Topologies and Assign Failed Tasks
- Provide Reliability by Tracking Tuple Tree
- Routing and Partitioning of Streams
- Serialization
- Fine-grained Performance Statistics/ Status of Topologies

# Comparison of Architecture: Hadoop v1 (MapReduce) vs. Storm

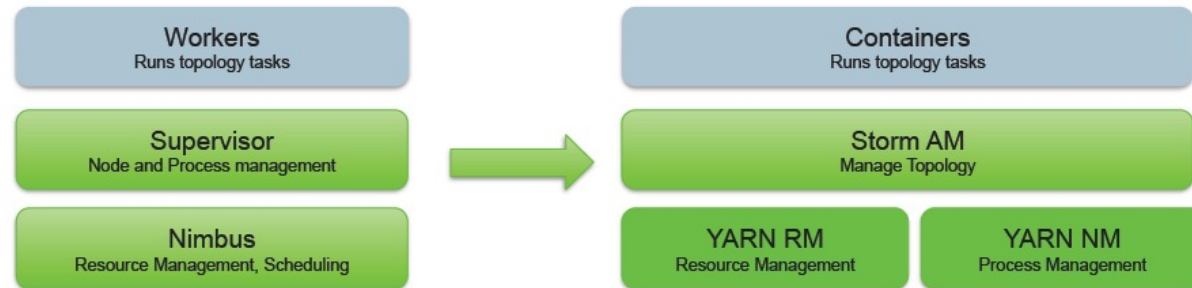
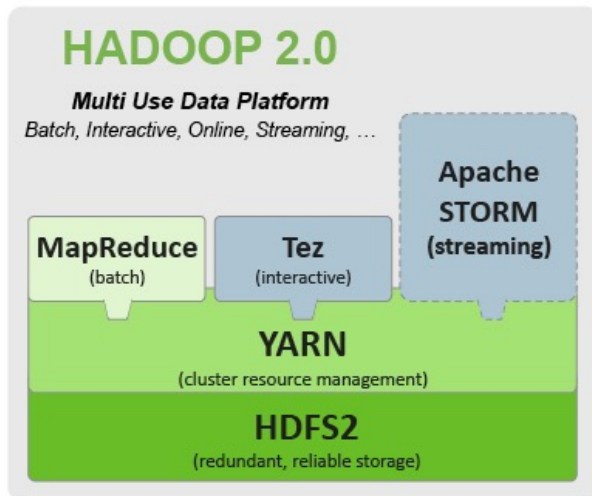
Hadoop v1	Storm	Functions in Storm
JobTracker	<b>Nimbus (only 1)</b>	<ul style="list-style-type: none"><li>▪ distributes code around cluster</li><li>▪ assigns tasks to machines/supervisors</li><li>▪ failure monitoring</li><li>▪ is fail-fast and stateless (you can "kill -9" it)</li></ul>
TaskTracker	<b>Supervisor (many)</b>	<ul style="list-style-type: none"><li>▪ listens for work assigned to its machine</li><li>▪ starts and stops worker processes as necessary based on Nimbus</li><li>▪ is fail-fast and stateless (you can "kill -9" it)</li><li>▪ shuts down worker processes with "kill -9", too</li></ul>
MR job	<b>Topology</b>	<ul style="list-style-type: none"><li>▪ processes messages forever (or until you kill it)</li><li>▪ a running topology consists of many worker processes spread across many machines</li></ul>

# Different ways to run Storm over a Cluster

- Twitter runs multiple instances of Storm over Mesos
  - Multiple Topologies can be run on the same host (Shared Pool) or
  - Dedicated Set of hosts to run a single topology (Isolated Pool)



- Storm can also be run as an application (framework) over YARN:



Mapping Storm's architecture to YARN's resource management model



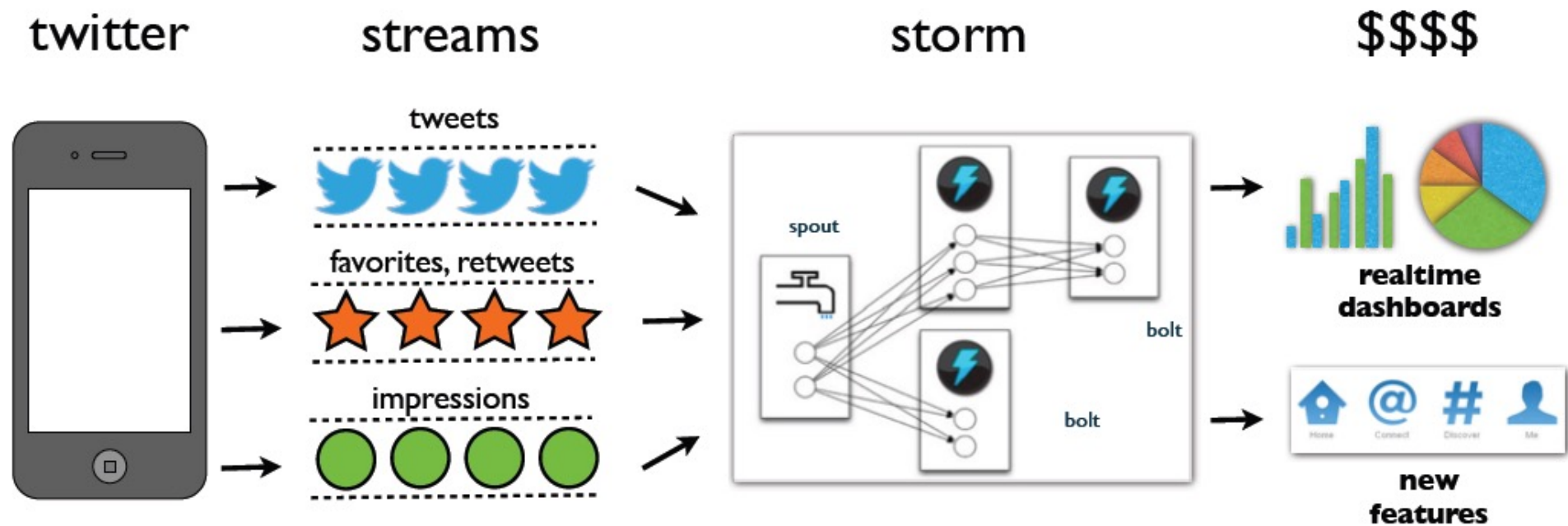
# Operating Storm

- Typical operations tasks include:
  - Monitoring topologies for Performance and Scalability (P&S) : “Don’t let our pipes blow up!”
    - Tackling P&S in Storm is a joint Ops-Dev effort.
  - Adding or removing slave nodes, i.e. nodes that run Supervisors
  - Apps management: new topologies, swapping topologies, ...
- See Ops-related references at the end of this part

# Storm security

- Original design was not created with security in mind.
- Security features are now being added, e.g. from Yahoo!'s fork.
- State of security in Storm 0.9.x:
  - No authentication, no authorization.
  - No encryption of data in transit, i.e. between workers.
  - No access restrictions on data stored in ZooKeeper.
  - Arbitrary user code can be run on nodes if Nimbus' Thrift port is not locked down.
  - This list goes on.
- Further details plus recommendations on hardening Storm:
  - <https://github.com/apache/incubator-storm/blob/master/SECURITY.md>

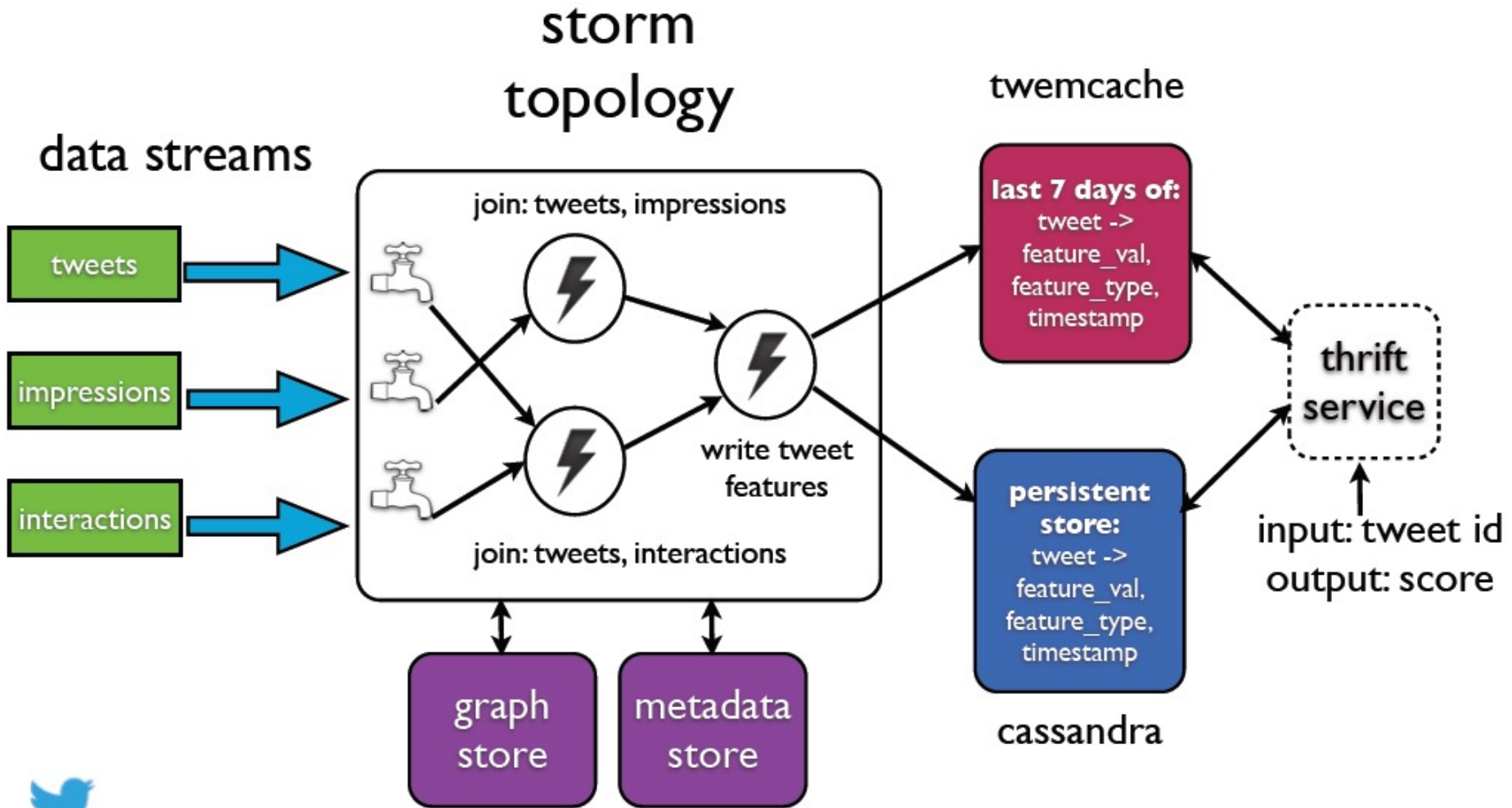
# Stream Processing Applications at Twitter



# Use Cases at Twitter

- Discovery of Emerging Topics and Stories
- Online Learning of Tweet Features for Search result Ranking
- Real-time Analytics for Ads
- Internal Log processing

# Tweet Scoring Pipeline



# Storm adoption and use cases

<https://github.com/nathanmarz/storm/wiki/Powered-By>

- **Twitter:** personalization, search, revenue optimization, ...
  - 200 nodes, 30 topos, 50B msg/day, avg latency <50ms, Jun 2013
- **Yahoo:** user events, content feeds, and application logs
  - 320 nodes (YARN), 130k msg/s, June 2013
- **Spotify:** recommendation, ads, monitoring, ...
  - v0.8.0, 22 nodes, 15+ topos, 200k msg/s, Mar 2014
- **Alibaba, Cisco, Flickr, PARC, WeatherChannel, ...**
  - Netflix is looking at Storm and Samza, too.



# More recent developments of Apache Storm

- Late 2013:
  - Storm enters Apache Incubator
- Early 2014 in Yahoo! :
  - 250-node cluster, largest topology 400 workers, 3,000 executors
- June 2014:
  - STORM-376 – Compress ZooKeeper data
  - STORM-375 – Check for changes before reading data from ZooKeeper
- Sep 2014:
  - Storm becomes an Apache Top Level Project
- Early 2015:
  - STORM-632 Better grouping for data skew
  - STORM-634 Thrift serialization for ZooKeeper data.
  - Yahoo deployed a 300-node cluster (Tested 400 nodes, 1,200 theoretical maximum)
  - Largest topology 1,500 workers, 4,000 executors
- June 2015:
  - Twitter announced the decommissioning of Storm ; replaced by Heron which adopts the same abstraction and 100% API-compatible with Storm:
    - <http://blog.acolyer.org/2015/06/15/twitter-heron-stream-processing-at-scale/>
    - Refer to the Heron paper in ACM SIGMOD 2015 for its technical details

# Open Issues of Storm



# Scheduling (Especially on Large Clusters)

- Currently round robin (We should be able to do better)
  - Take into account resource utilization (Network)
  - Locality with colocated services (Is there any advantage to running storm on the same nodes as HBase and/or Kafka?)
- What about better scheduling for Storm on Yarn?
- When is it worth it to kill a worker because a better location is available? (automatic rebalance)
  - Slow node detection (12 ms)

# Scalability Bottlenecks of Storm (cont'd)

## Storm round-robin scheduling

- $R-1/R$  % of traffic will be off rack where  $R$  is the number of racks
- $N-1/N$  % of traffic will be off node where  $N$  is the number of nodes
- Does not know when resources are full (i.e. network)

Solution: Resource & Network Topography Aware Scheduler (Storm V1.0.0, released Apr 2016) ; Also allow Pluggable Scheduler provided by user

## One slow node slows the entire topology.

Load Aware Routing (STORM-162)  
Intelligent network aware routing



## Other Notable New Features from Storm V1.0.0 Rel. Apr 2016

Automatic Backpressure, Native Streaming Window API,  
Stateful Bolts with Automatic Checkpointing, HA Nimbus

<https://storm.apache.org/2016/04/12/storm100-released.html>

# How can one grow/shrink a topology dynamically

- How to handle the different shuffles? Do we kill everything and start over or is there a better way?
- When should we grow or shrink?
- What do we do if there are no free resources and we need to grow?
- Or even more difficult can we upgrade a topology in place without killing processes?

# Resource Isolation/Utilization

- Isolation is handled currently by creating a mini-cluster (whole nodes) for a single topology (so utilization suffers).
- Can we get better utilization without letting isolation suffer? cgroups/Docker. What about predictability on heavily used vs. lightly used clusters? (12ms again)
- What about if I collocate this batch on Hadoop?

# Higher level APIs

- Streaming SQL (Spark streaming has it already)
- Pig on Storm (Some proto-type effort but...)
- Native Window-based Processing API available from Storm V1.0.0.

# Other Competing Stream Processing Systems...

- Heron (Twitter)
  - Same User Programming model and API, differ mostly in the under-the-hood system realization/ implementation, e.g.
    - Written in C++ instead of Closure
    - Better separation and scheduling of tasks, executors in JVM(s) for different components of the same/ different topologies to facilitate debugging
    - Backpressure-based congestion control of dataflow within a topology
- Google Cloud Dataflow ( <http://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf> )
  - Open Source API, BUT NOT implementation
  - Based on technologies from Google's FlumeJava & MillWheel
  - Great stream processing concepts
- Spark Streaming (BDAS of Berkeley/Databricks)
  - Micro-batch processing instead of true real-time streaming
- Microsoft's Naiad, Apache Apex (DataTorrent), Flink, Samza (LinkedIn), Amazon's Kinesis, etc

## A Summary on What Storm does

- Distribute Code and Configuration
- Robust Process Management
- Monitors Topologies and Assign Failed Tasks
- Provide Reliability by Tracking Tuple Tree
- Routing and Partitioning of Streams
- Serialization
- Fine-grained Performance Statistics/ Status of Topologies

# Additional References

- A few Storm books are already available, e.g.
  - Storm Applied by S.T. Allen et al, published by Manning, 2015
- Storm documentation
  - [https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.5/bk\\_storm-component-guide/bk\\_storm-component-guide.pdf](https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.5/bk_storm-component-guide/bk_storm-component-guide.pdf)
  - <http://storm.apache.org/releases/1.2.2/index.html>
- Storm Kafka Integration
  - <http://storm.apache.org/releases/1.2.2/storm-kafka-client.html>
- Mailing lists
  - [https://mail-archives.apache.org/mod\\_mbox/storm-user/](https://mail-archives.apache.org/mod_mbox/storm-user/)
- Related work aka tools that are similar to Storm – try them, too!
  - [Spark Streaming](#)
    - See comparison [Apache Storm vs. Apache Spark Streaming](#), by P. Taylor Goetz (Storm committer)