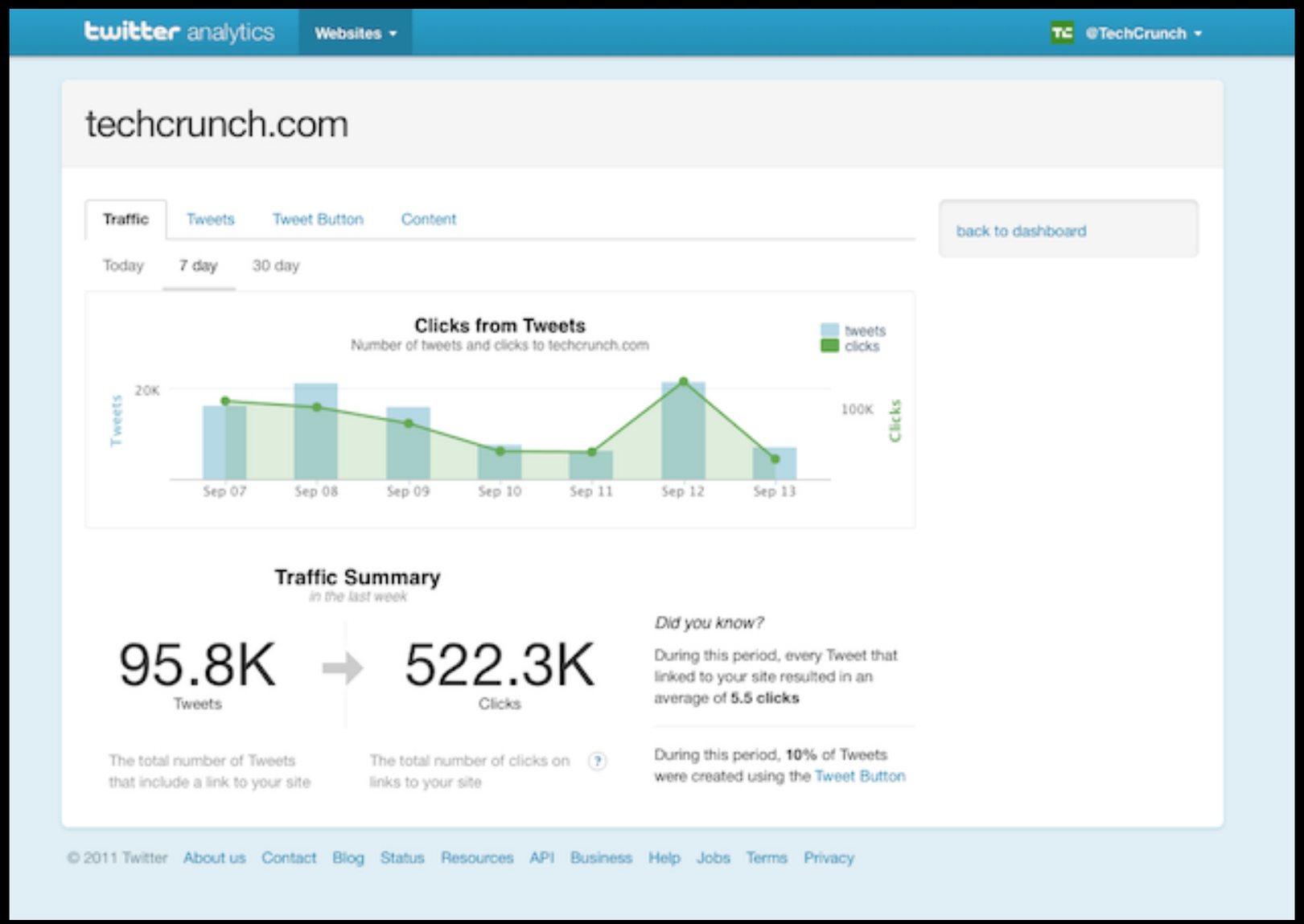
Storm

Distributed and fault-tolerant realtime computation

Nathan Marz
Twitter

Storm at Twitter



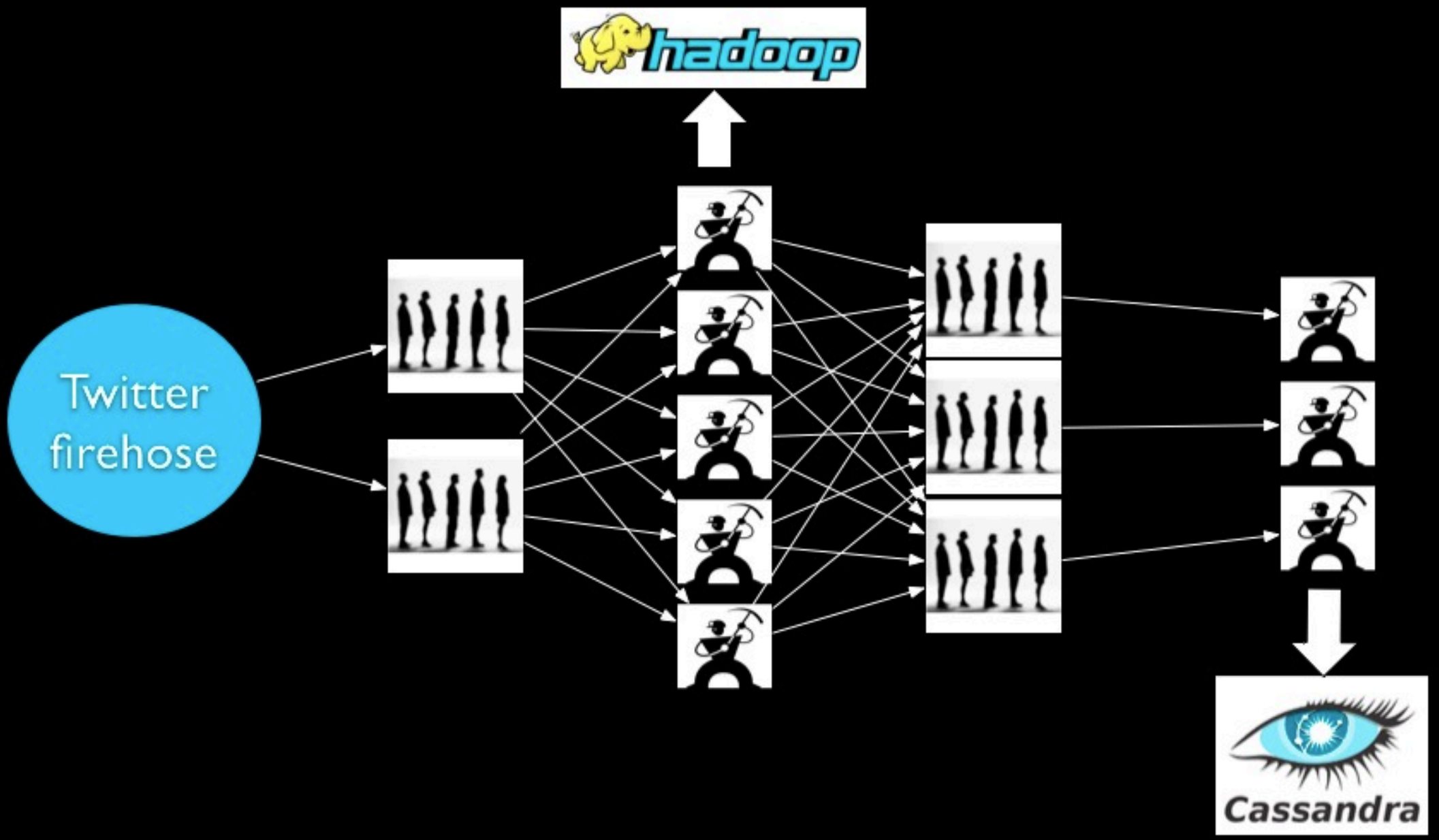
Twitter Web Analytics

Before Storm

Queues

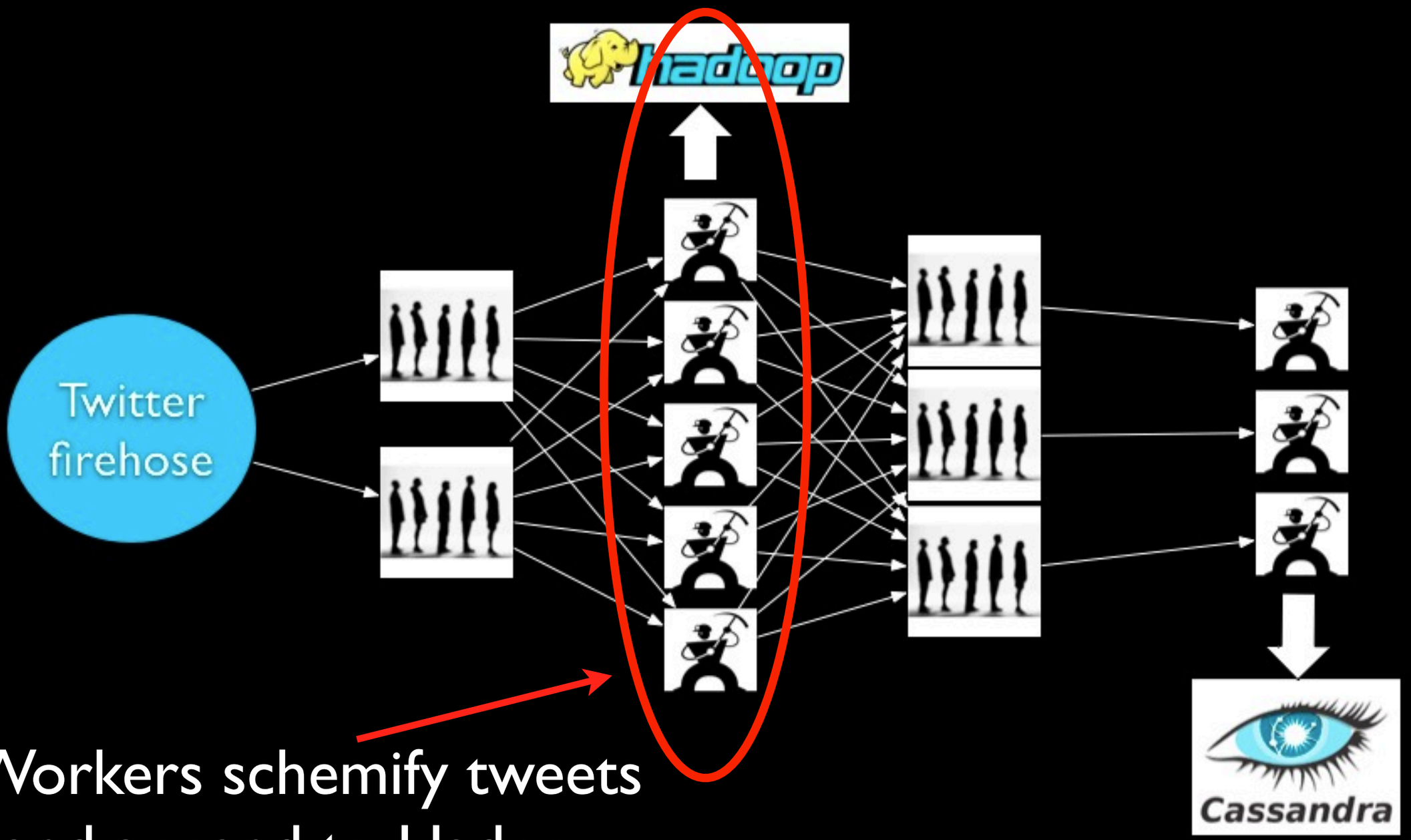
Workers

Example



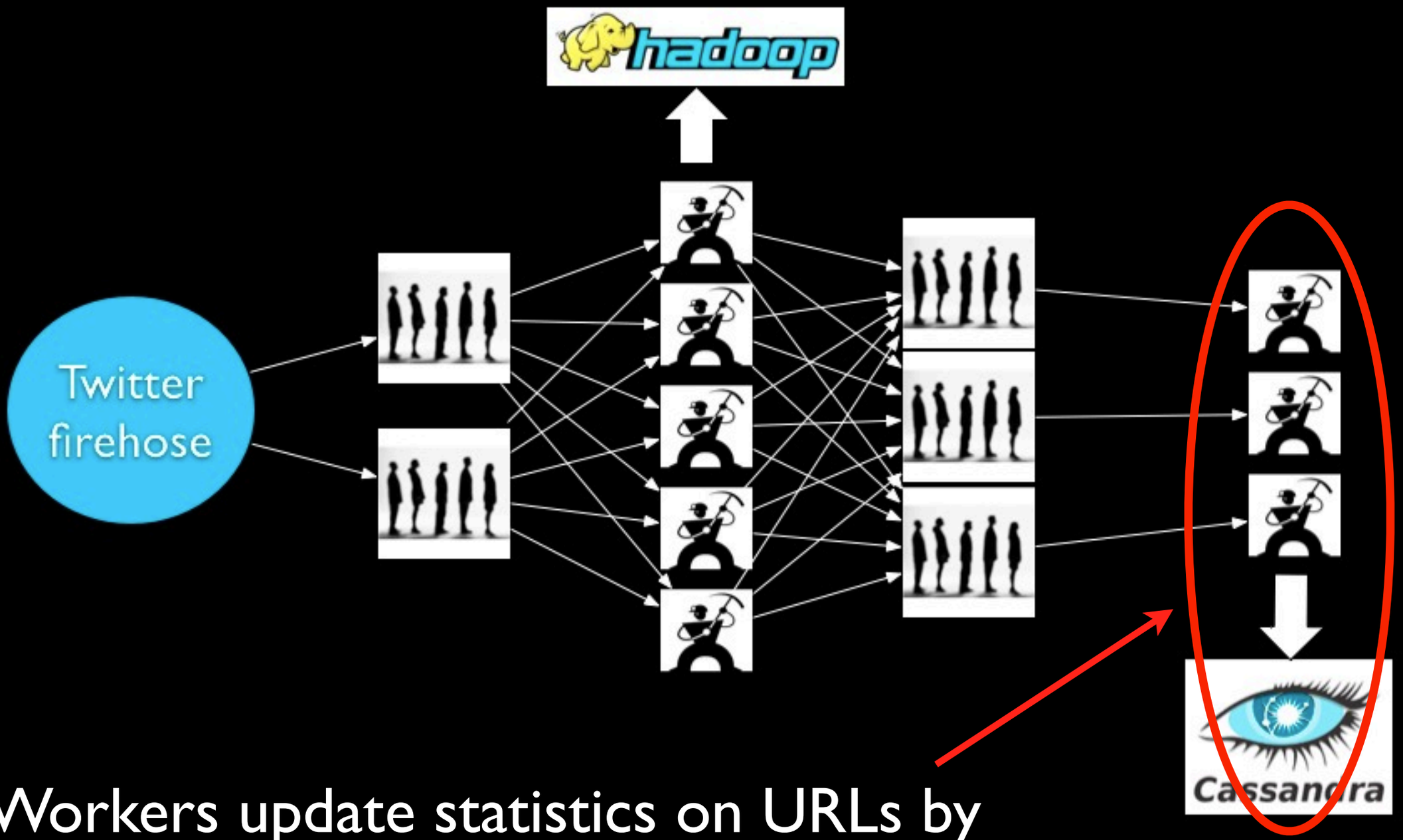
(simplified)

Example



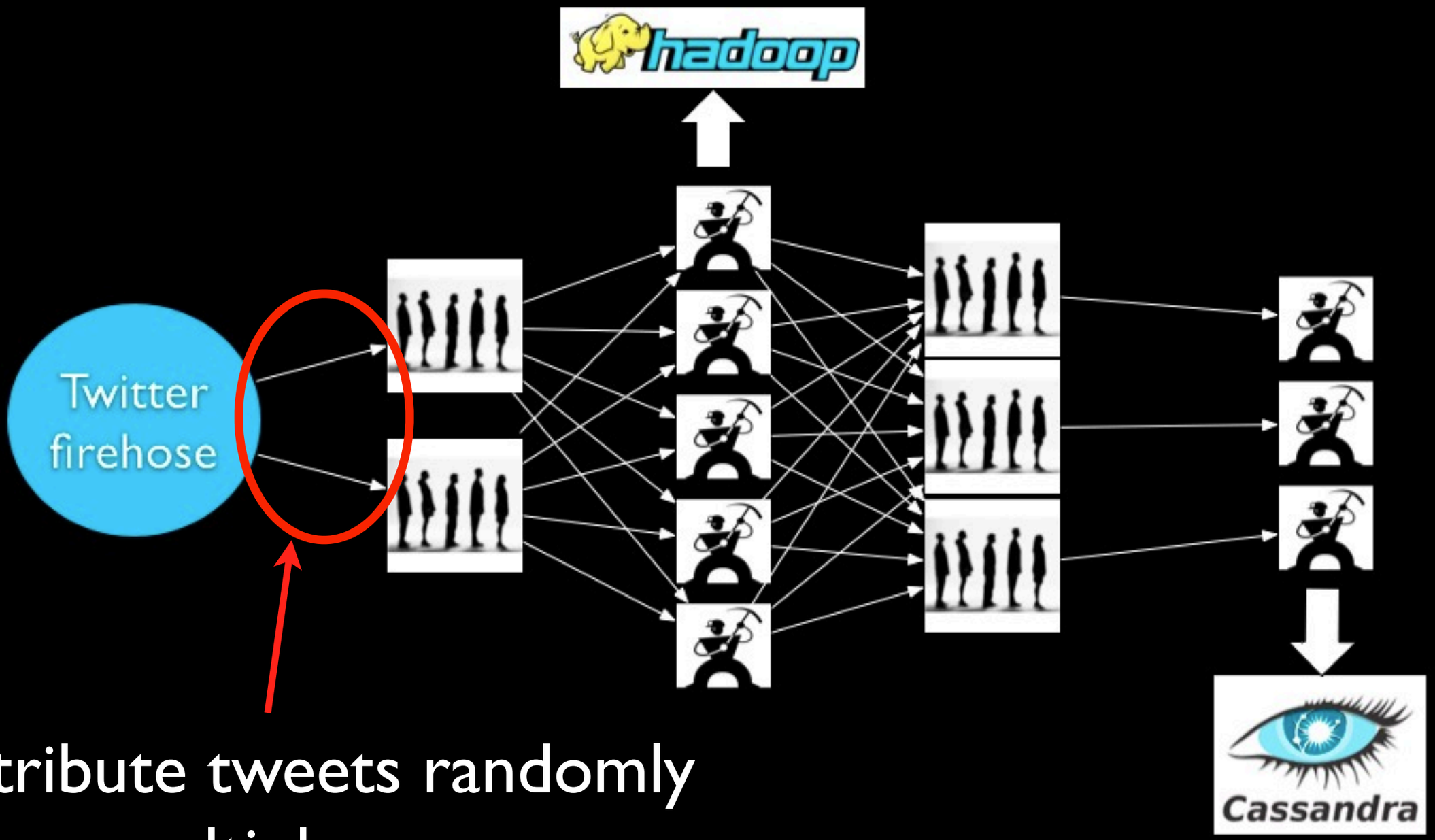
Workers schemify tweets and append to Hadoop

Example



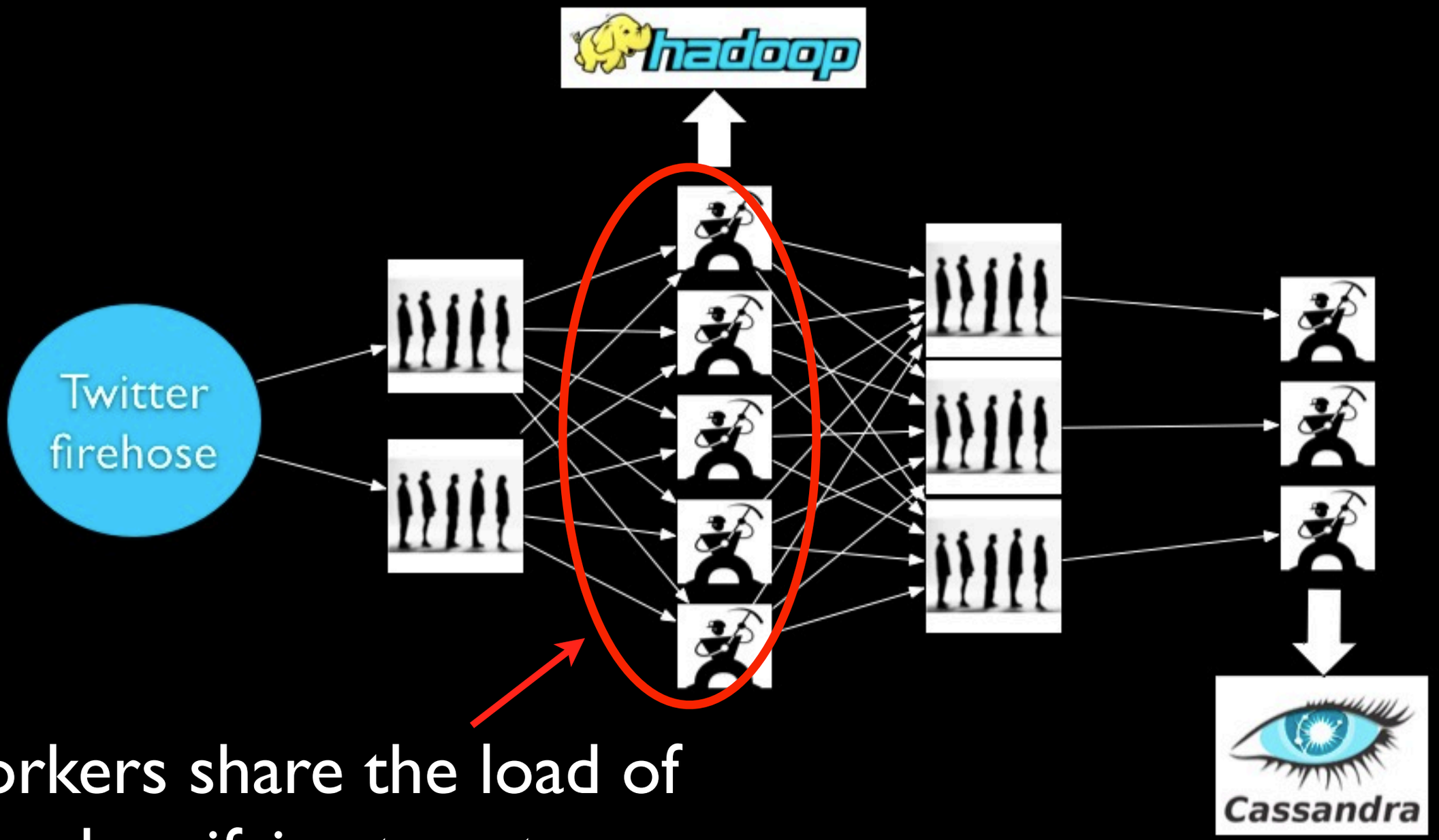
Workers update statistics on URLs by incrementing counters in Cassandra

Example



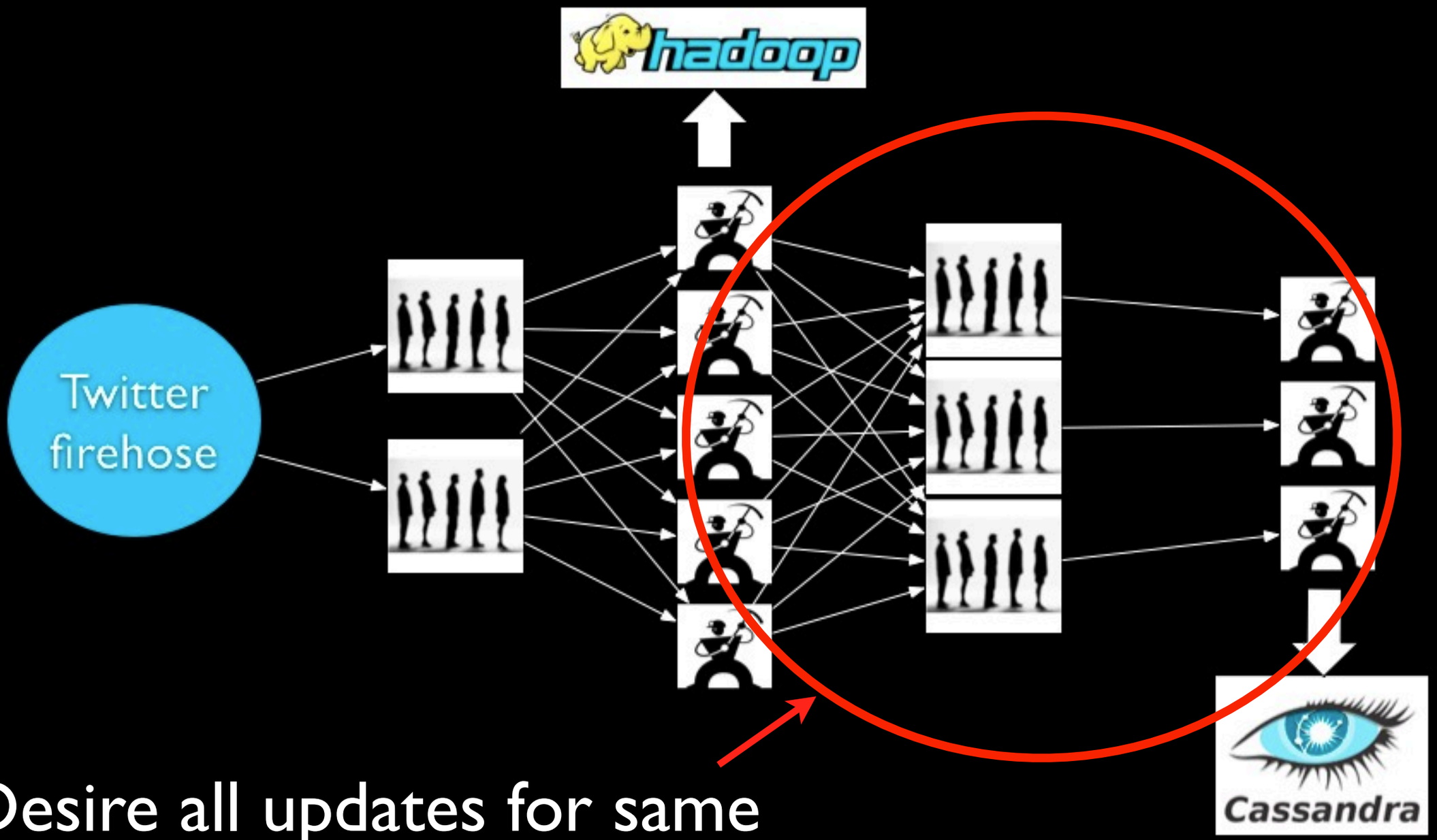
Distribute tweets randomly on multiple queues

Example



Workers share the load of schemifying tweets

Example



Desire all updates for same URL go to same worker

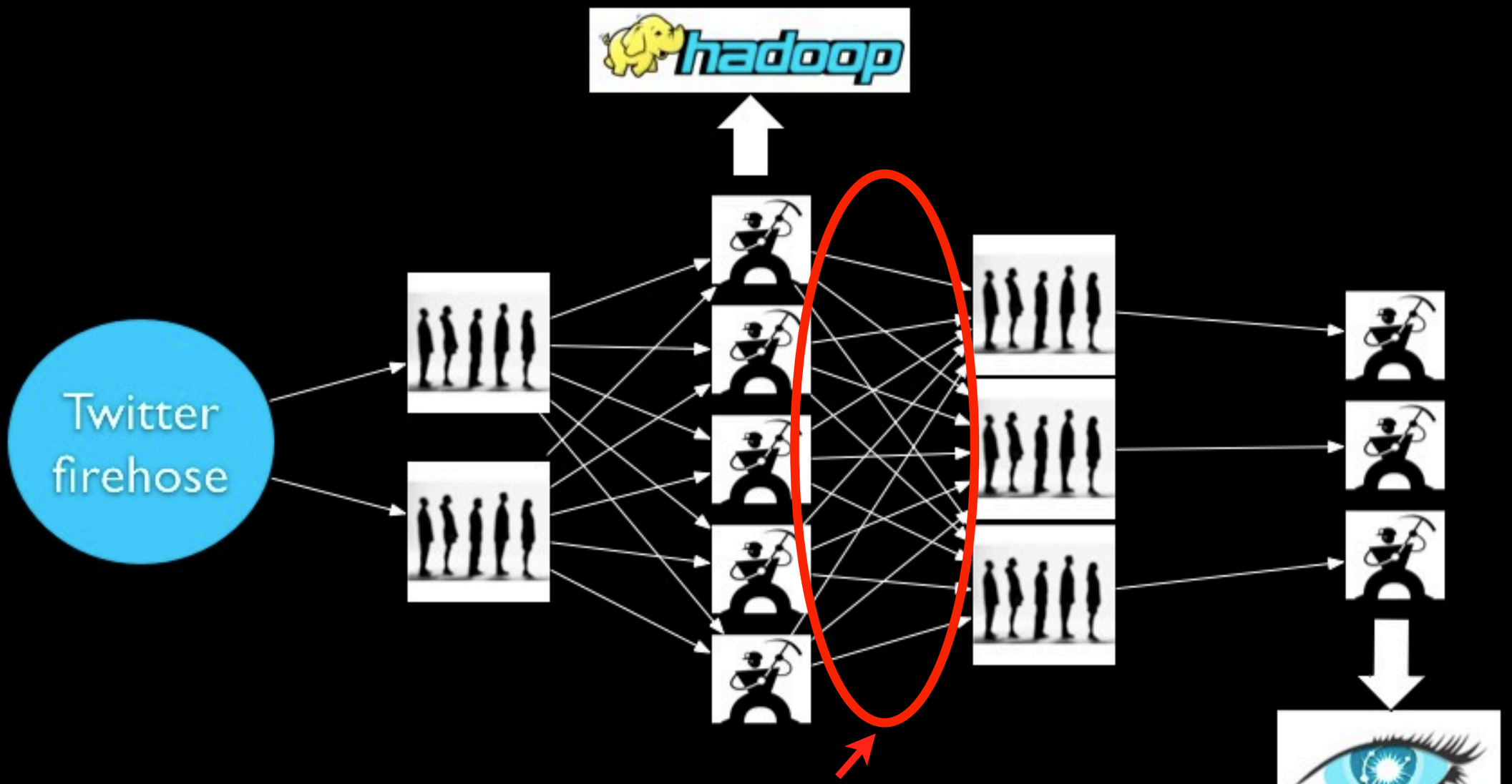
Message locality

- Because:
  - No transactions in Cassandra (and no atomic increments at the time)
  - More effective batching of updates

Implementing message locality

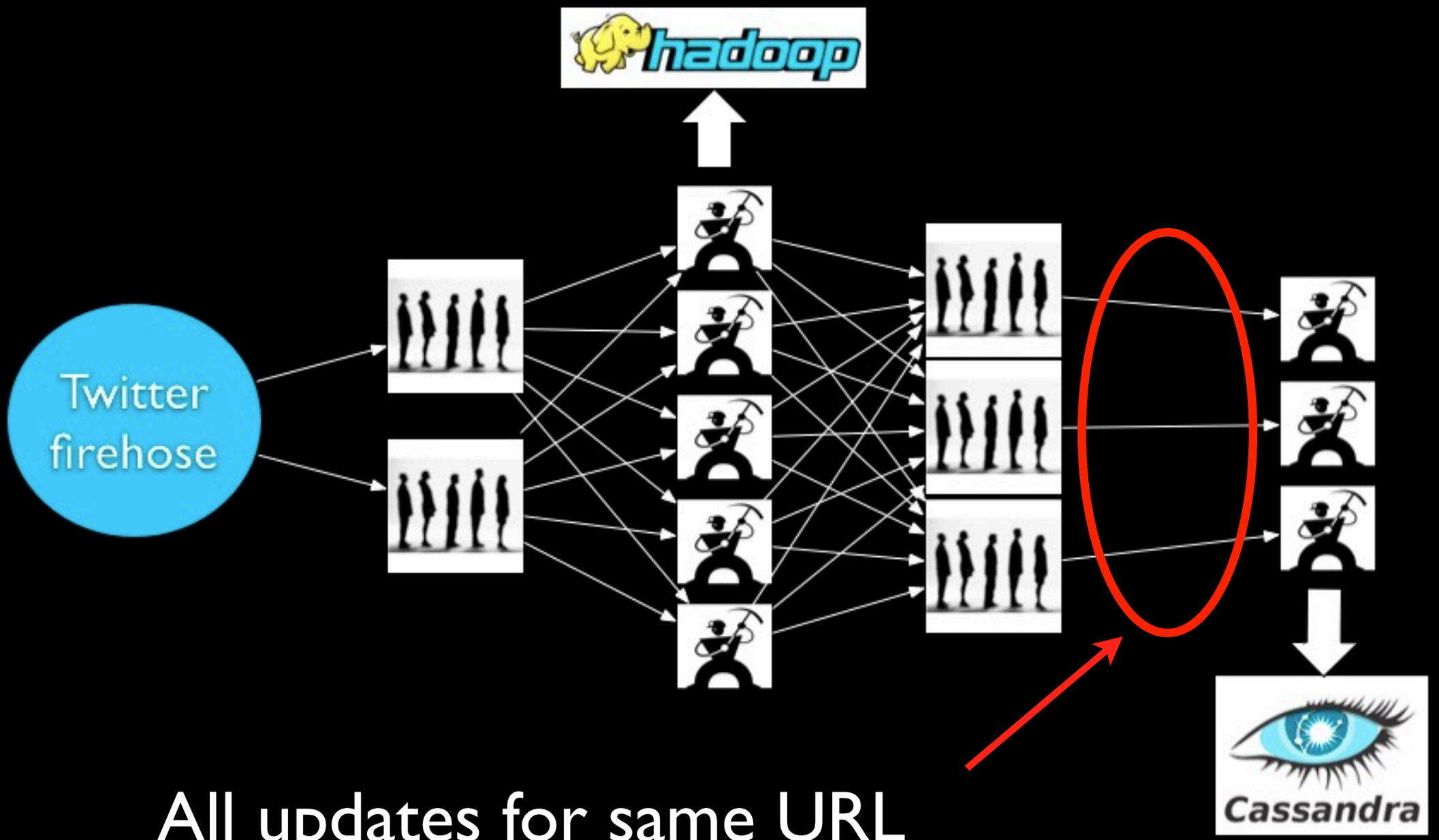
- Have a queue for each consuming worker
- Choose queue for a URL using consistent hashing

Example



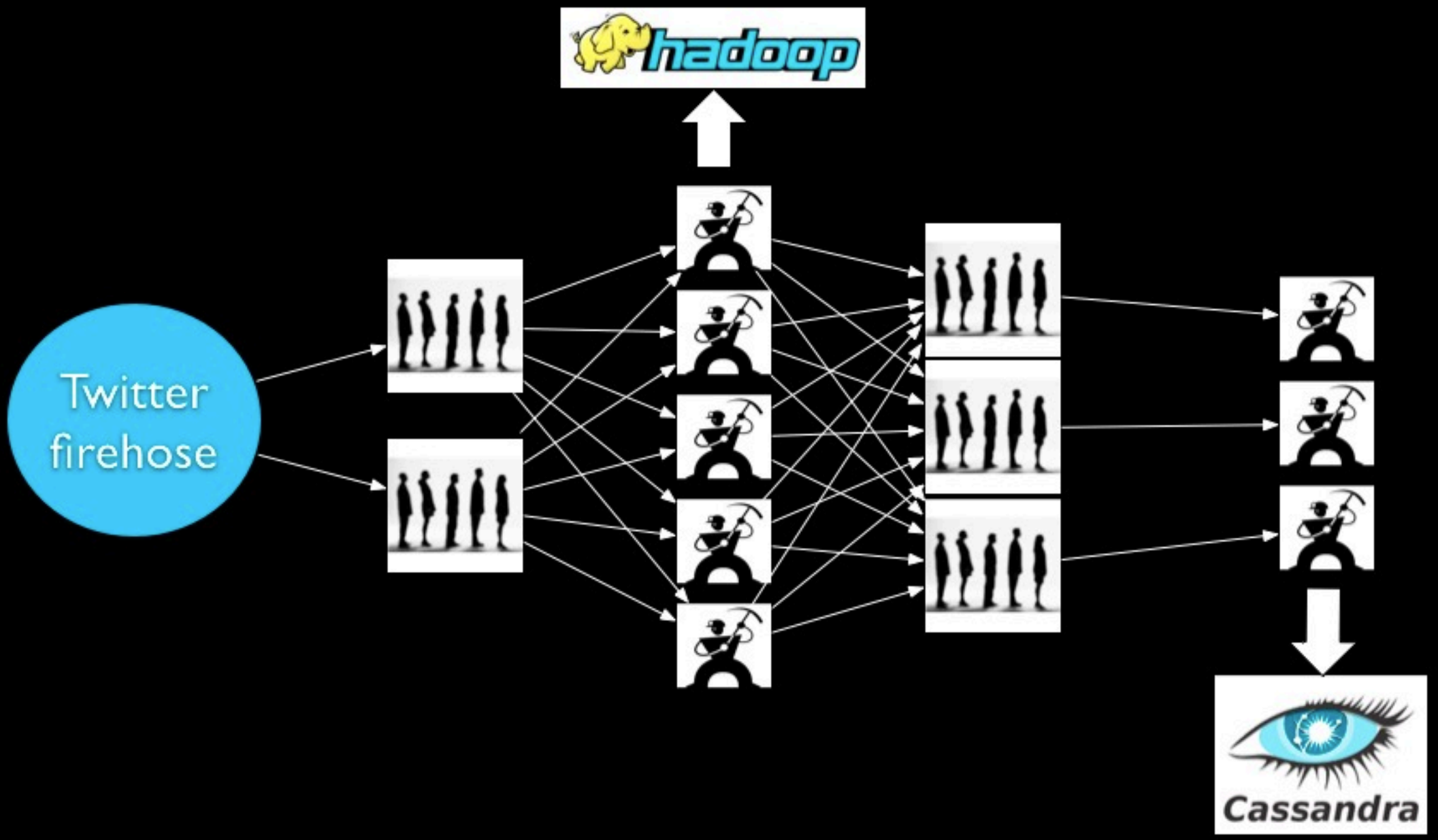
Workers choose queue to enqueue to using hash/mod of URL

Example

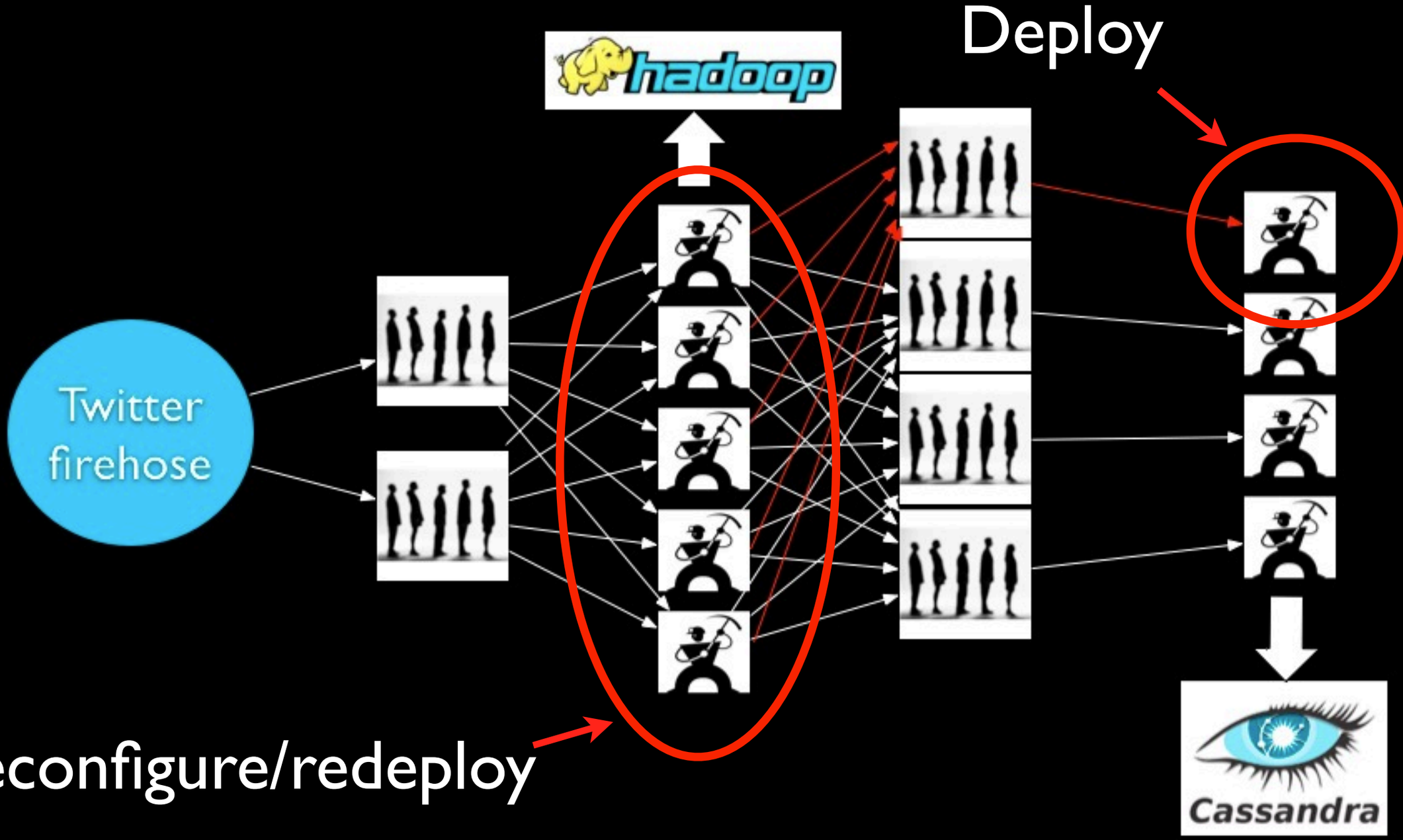


All updates for same URL guaranteed to go to same worker

Adding a worker



Adding a worker



Problems

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

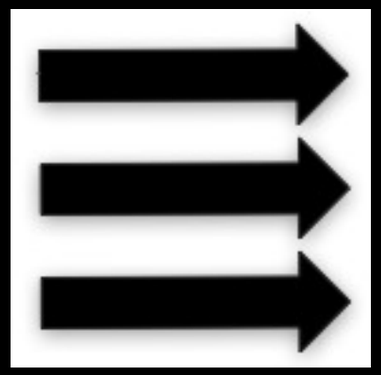
What we want

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

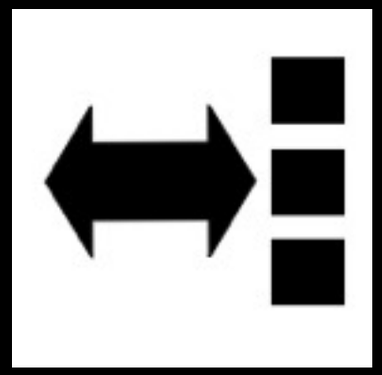
Storm

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

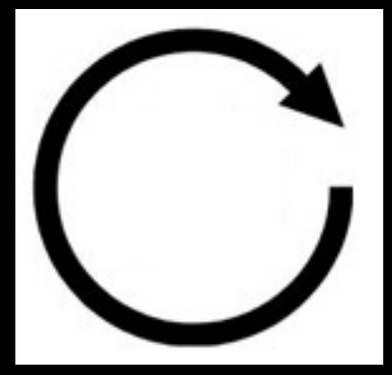
Use cases



Stream processing

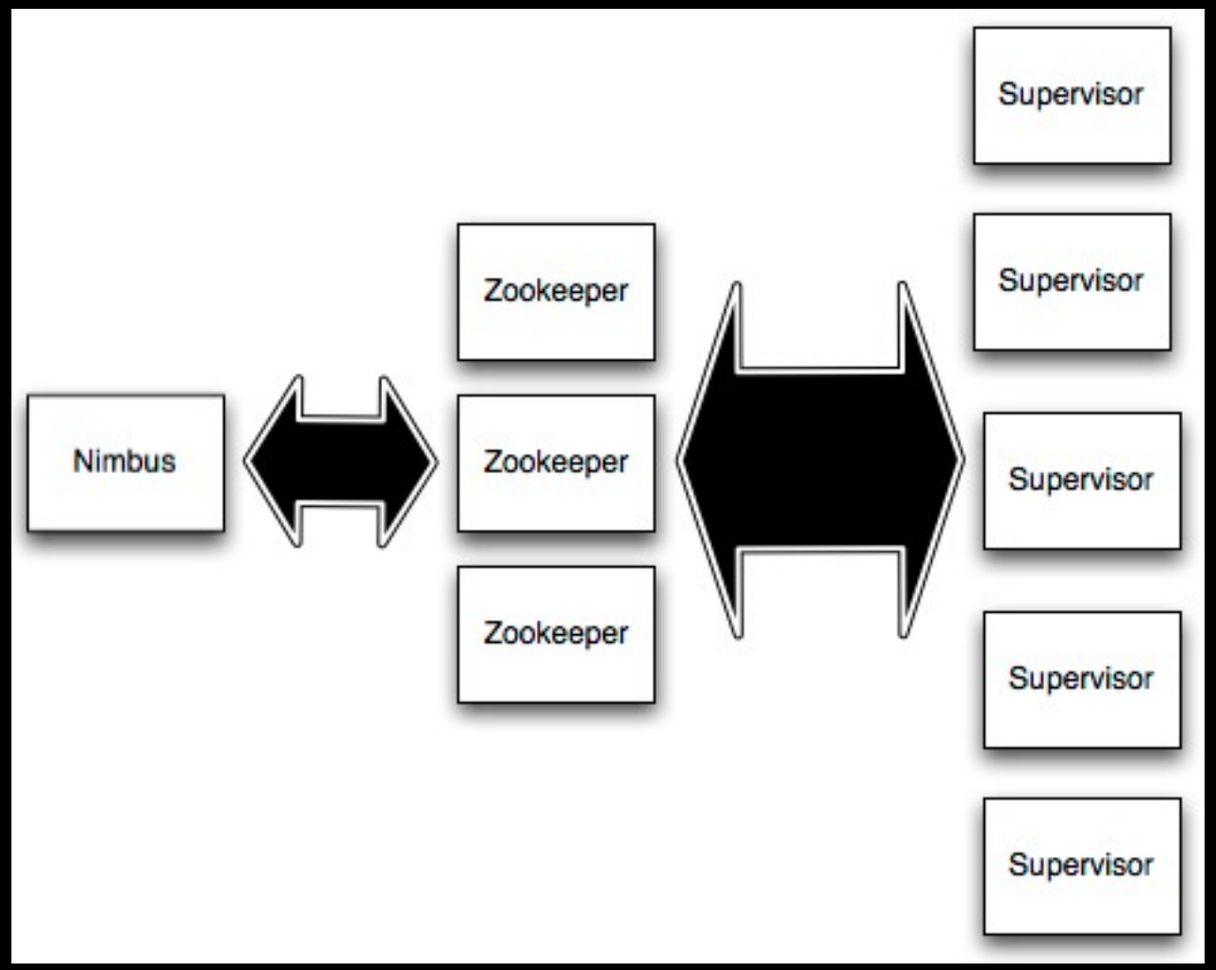


Distributed RPC

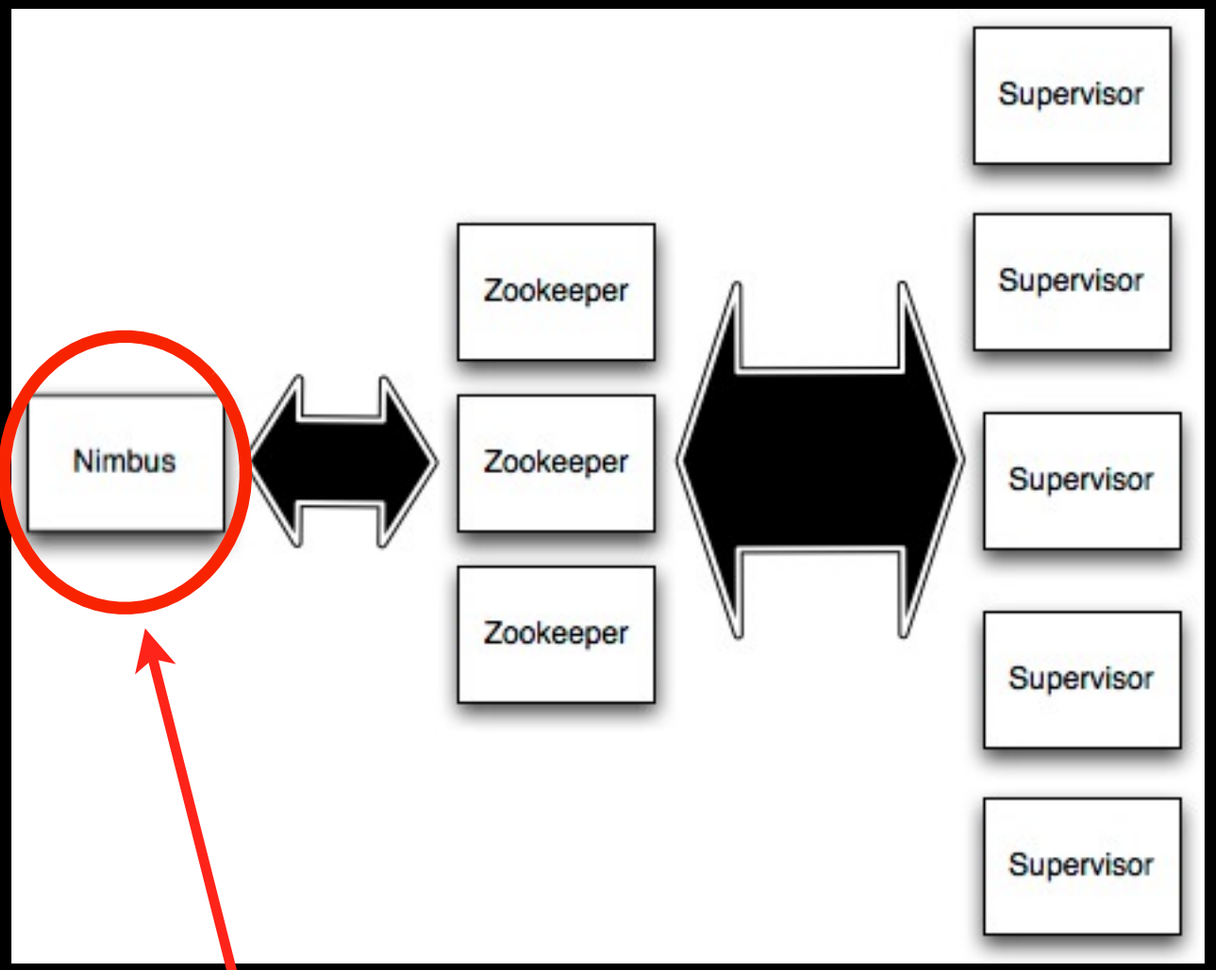


Continuous computation

Storm Cluster

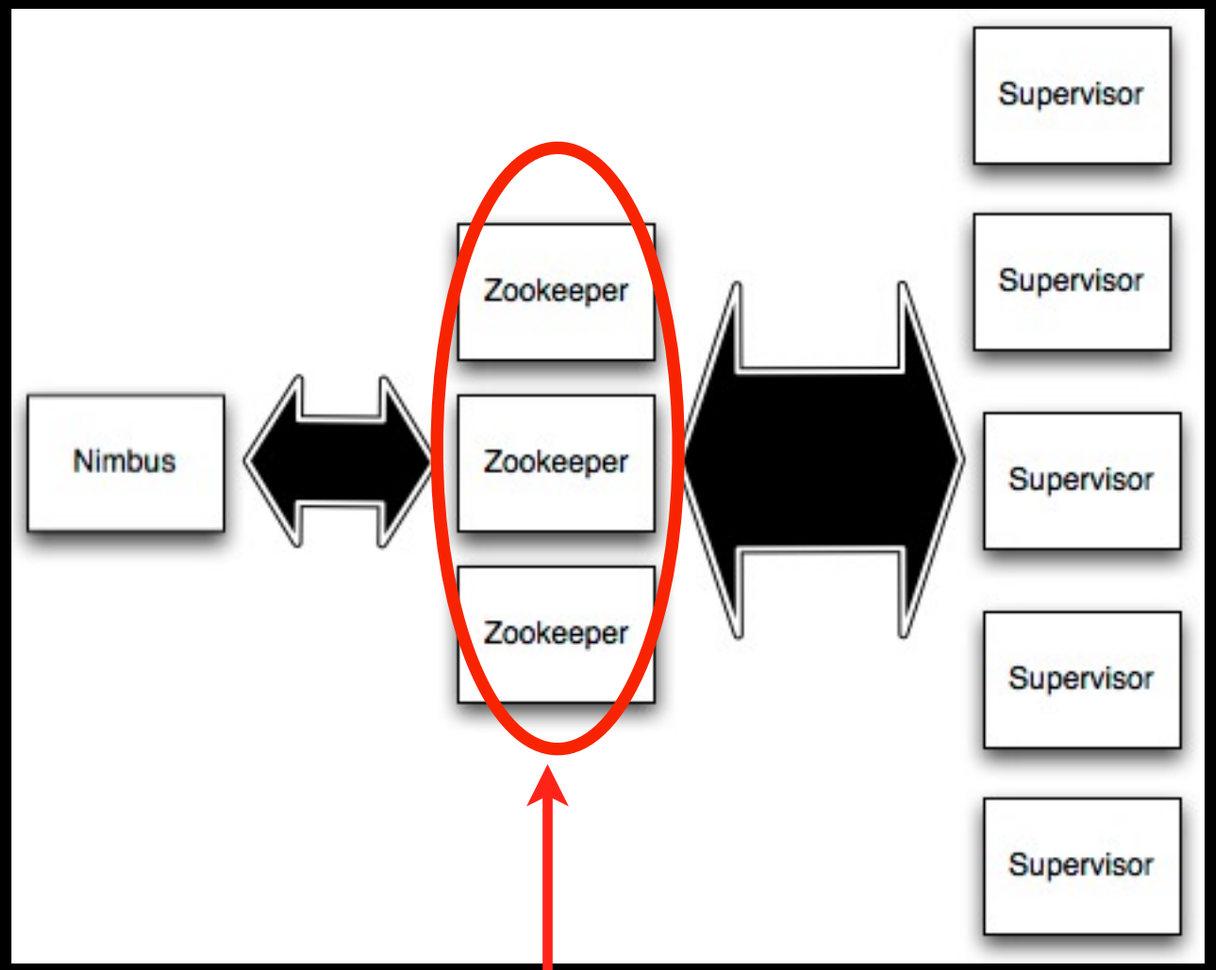


Storm Cluster



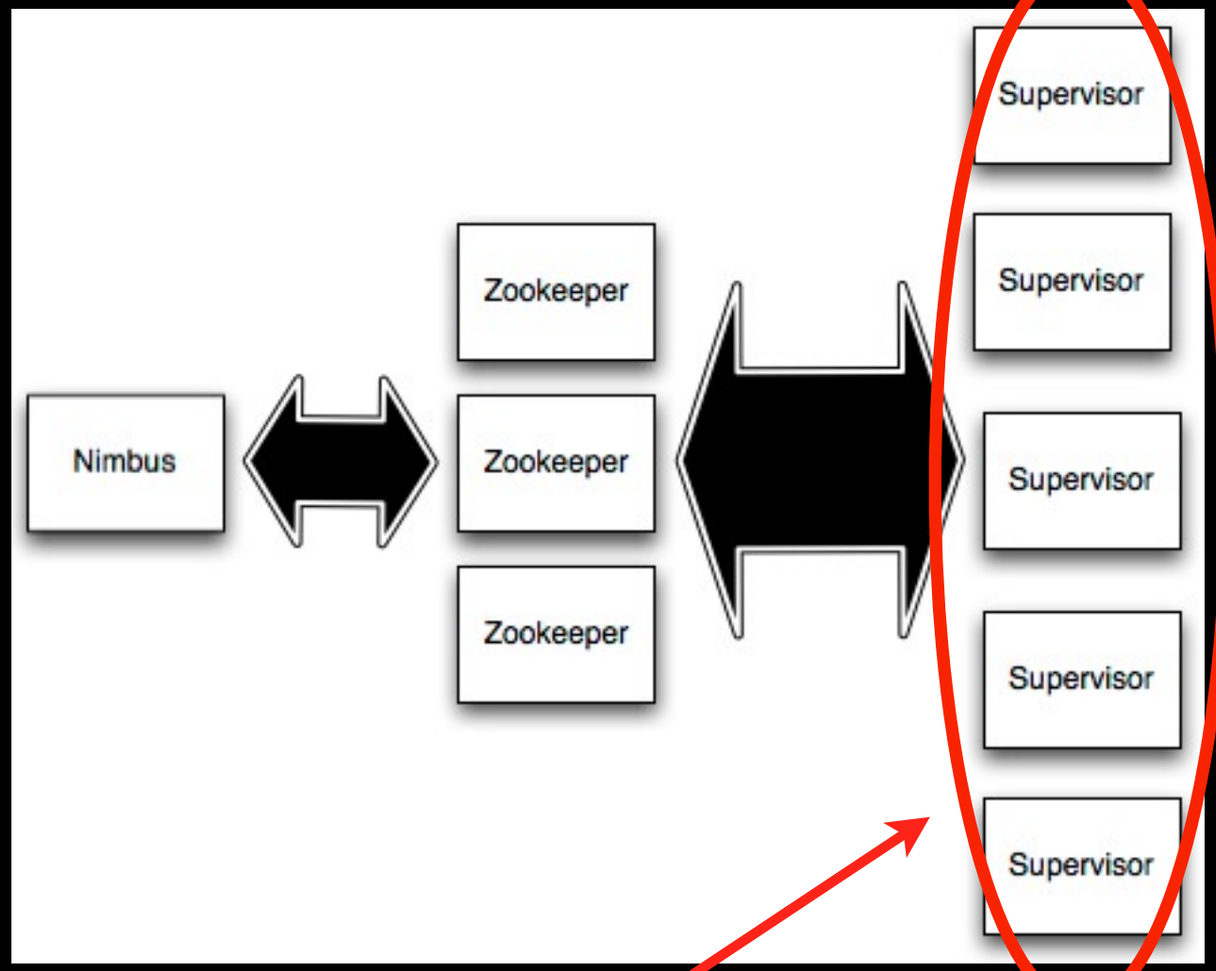
Master node (similar to Hadoop JobTracker)

Storm Cluster



Used for cluster coordination

Storm Cluster



Run worker processes

Starting a topology

```
storm jar mycode.jar twitter.storm.MyTopology demo
```

Killing a topology

```
storm kill demo
```

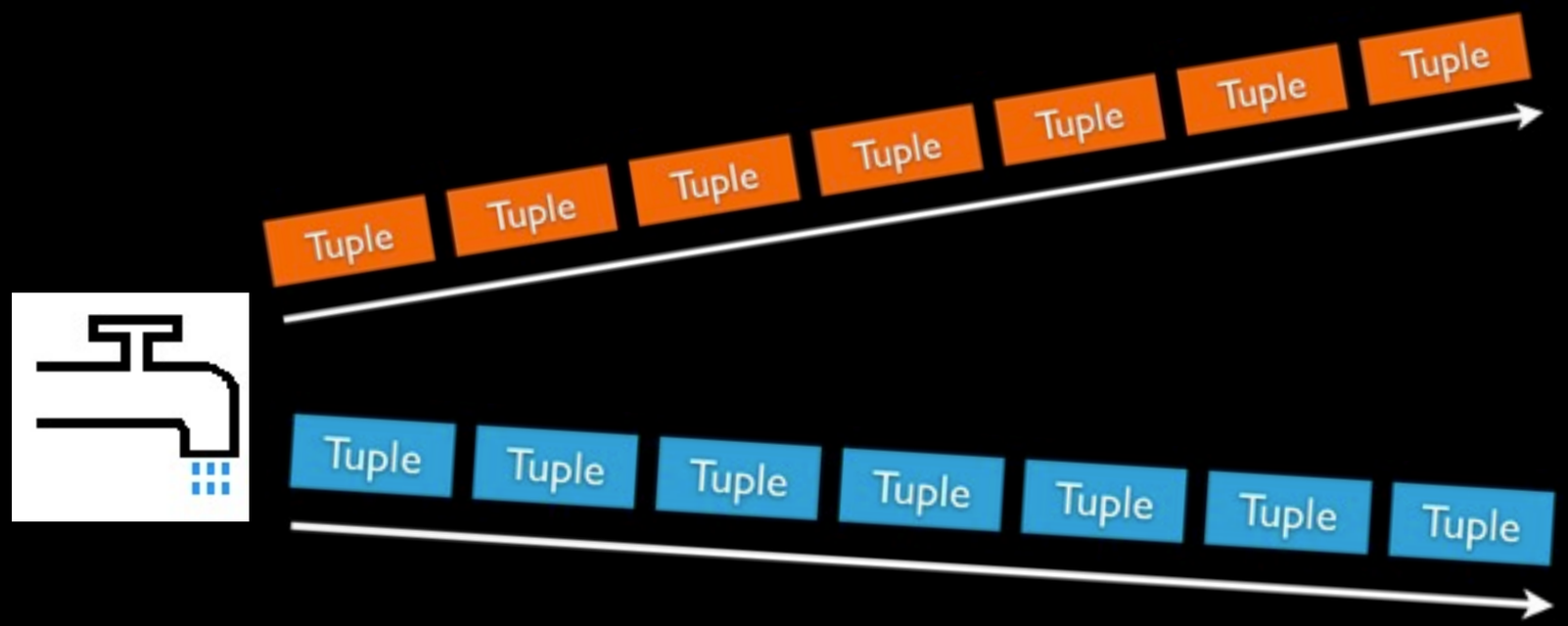
Concepts

- Streams
- Spouts
- Bolts
- Topologies

Streams

Unbounded sequence of tuples

Spouts

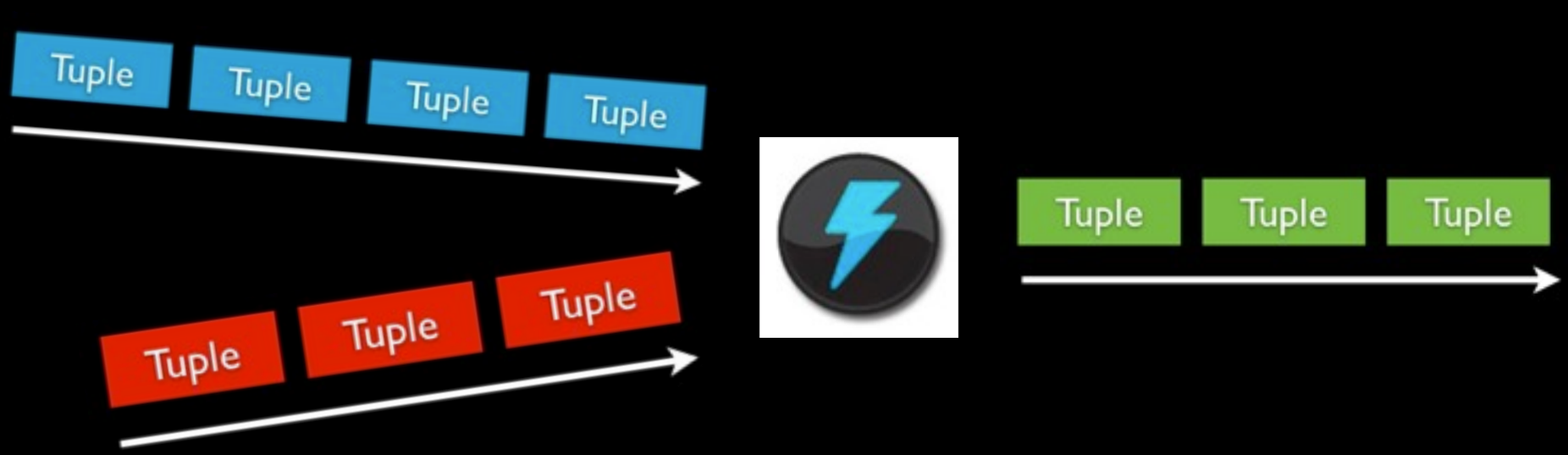


Source of streams

Spout examples

- Read from Kestrel queue
- Read from Twitter streaming API

Bolts

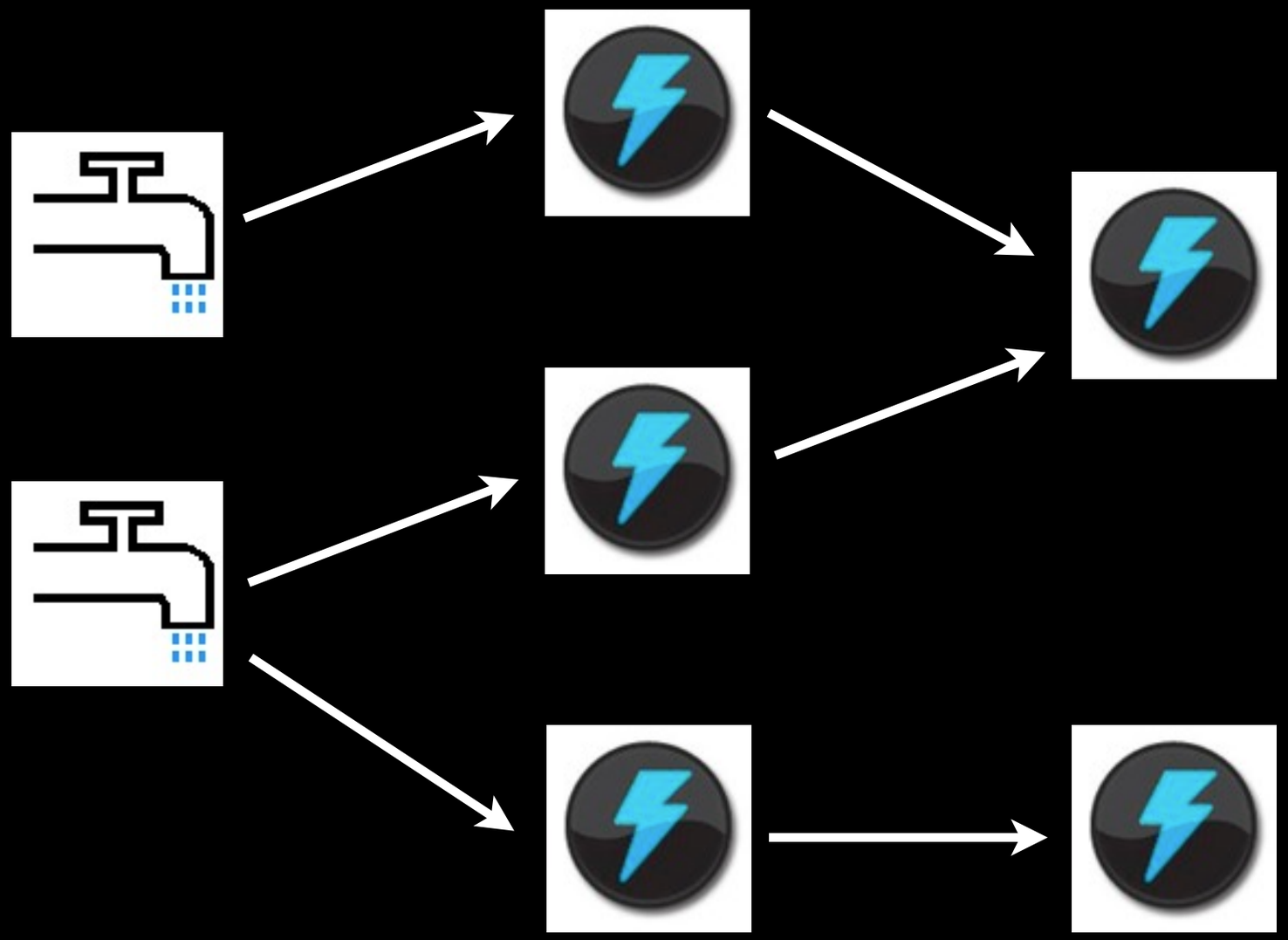


Processes input streams and produces new streams

Bolts

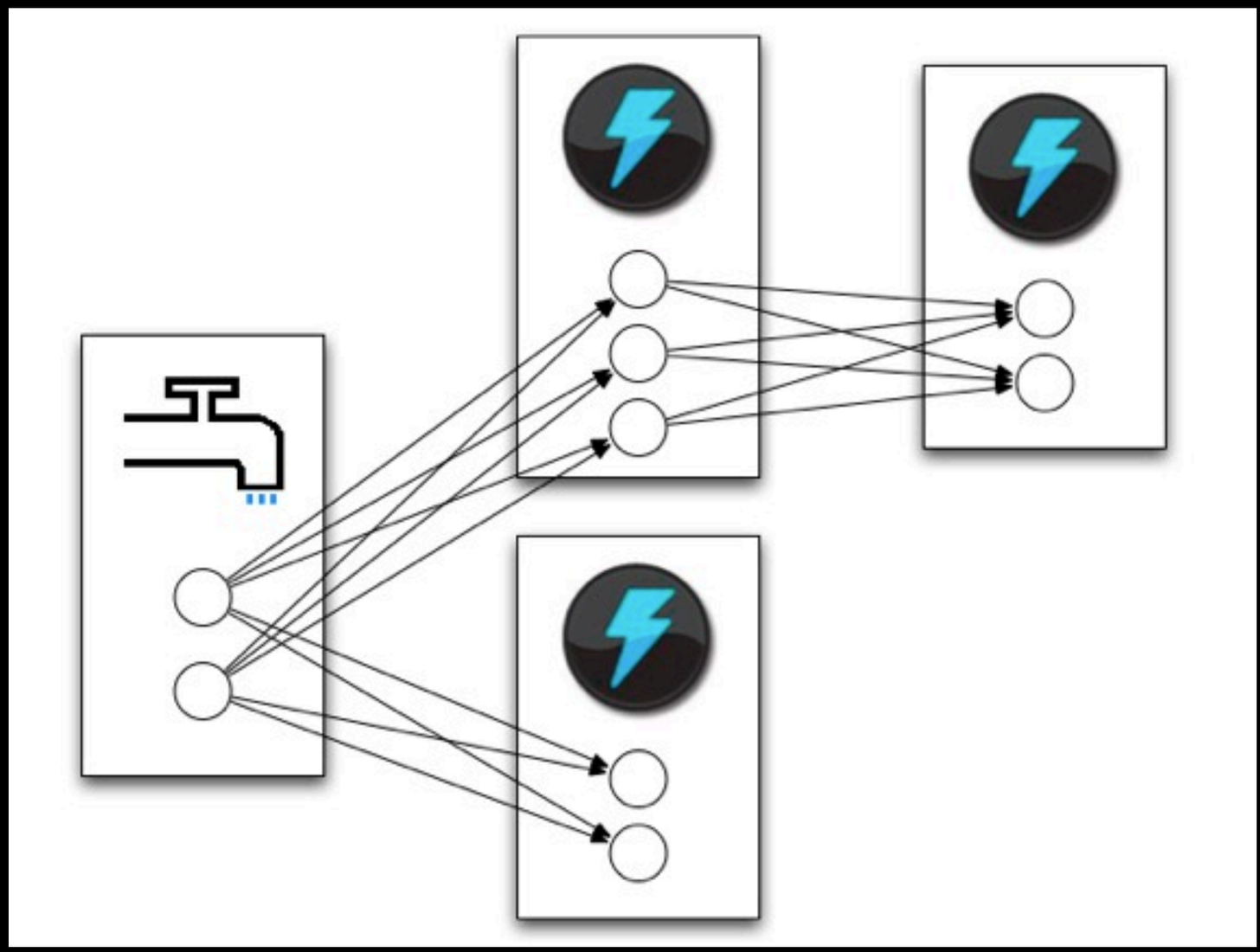
- Functions
- Filters
- Aggregation
- Joins
- Talk to databases

Topology



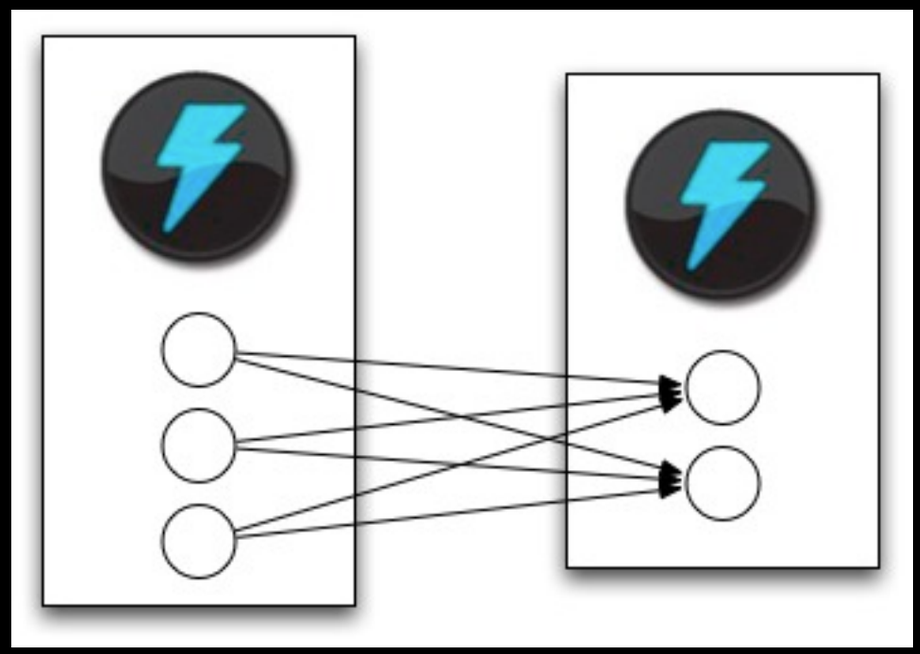
Network of spouts and bolts

Tasks



Spouts and bolts execute as
many tasks across the cluster

Stream grouping

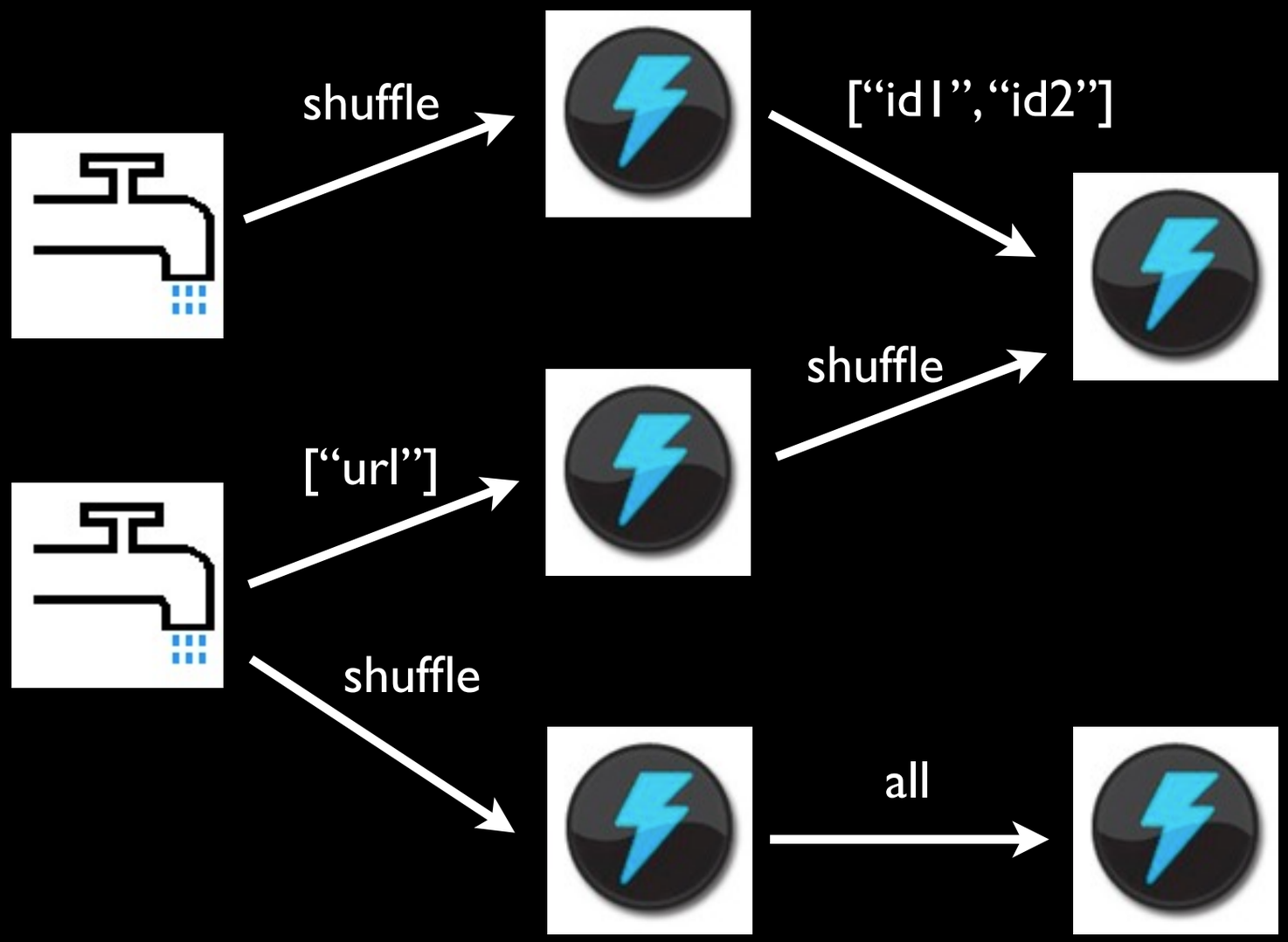


When a tuple is emitted, which task does it go to?

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

Topology



Streaming word count

```
TopologyBuilder builder = new TopologyBuilder();
```

TopologyBuilder is used to construct topologies in Java

Streaming word count

```
builder.setSpout(
  1,
  new KestrelSpout("kestrel.twitter.com",
                   22133,
                   "sentence_queue",
                   new StringScheme()),
5);
```

Define a spout in the topology with parallelism of 5 tasks

Streaming word count

```
builder.setBolt(2, new SplitSentence(), 8)
        .shuffleGrouping(1);
```

Split sentences into words with parallelism of 8 tasks

Streaming word count

```
builder.setBolt(2, new SplitSentence(), 8)
        .shuffleGrouping(1);
```

Consumer decides what data it receives and how it gets grouped

Split sentences into words with parallelism of 8 tasks

Streaming word count

```
builder.setBolt(3, new WordCount(), 12)
          .fieldsGrouping(2, new Fields("word"));
```

Create a word count stream

Streaming word count

```java
public static class SplitSentence extends ShellBolt implements IRichBolt {
    public SplitSentence() {
        super("python", "splitsentence.py");
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }
}
```

```python
import storm

class SplitSentenceBolt(storm.BasicBolt):
    def process(self, tup):
        words = tup.values[0].split(" ")
        for word in words:
          storm.emit([word])
```

splitsentence.py

Streaming word count

```
public static class WordCount implements IBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();

    public void prepare(Map conf, TopologyContext context) {
    }

    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.get(word);
        if(count==null) count = 0;
        count++;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }

    public void cleanup() {
    }

    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }
}
```

Streaming word count

```
Map conf = new HashMap();
conf.put(Config.TOPOLOGY_WORKERS, 10);

StormSubmitter.submitTopology("word-count", conf, builder.createTopology());
```

Submitting topology to a cluster

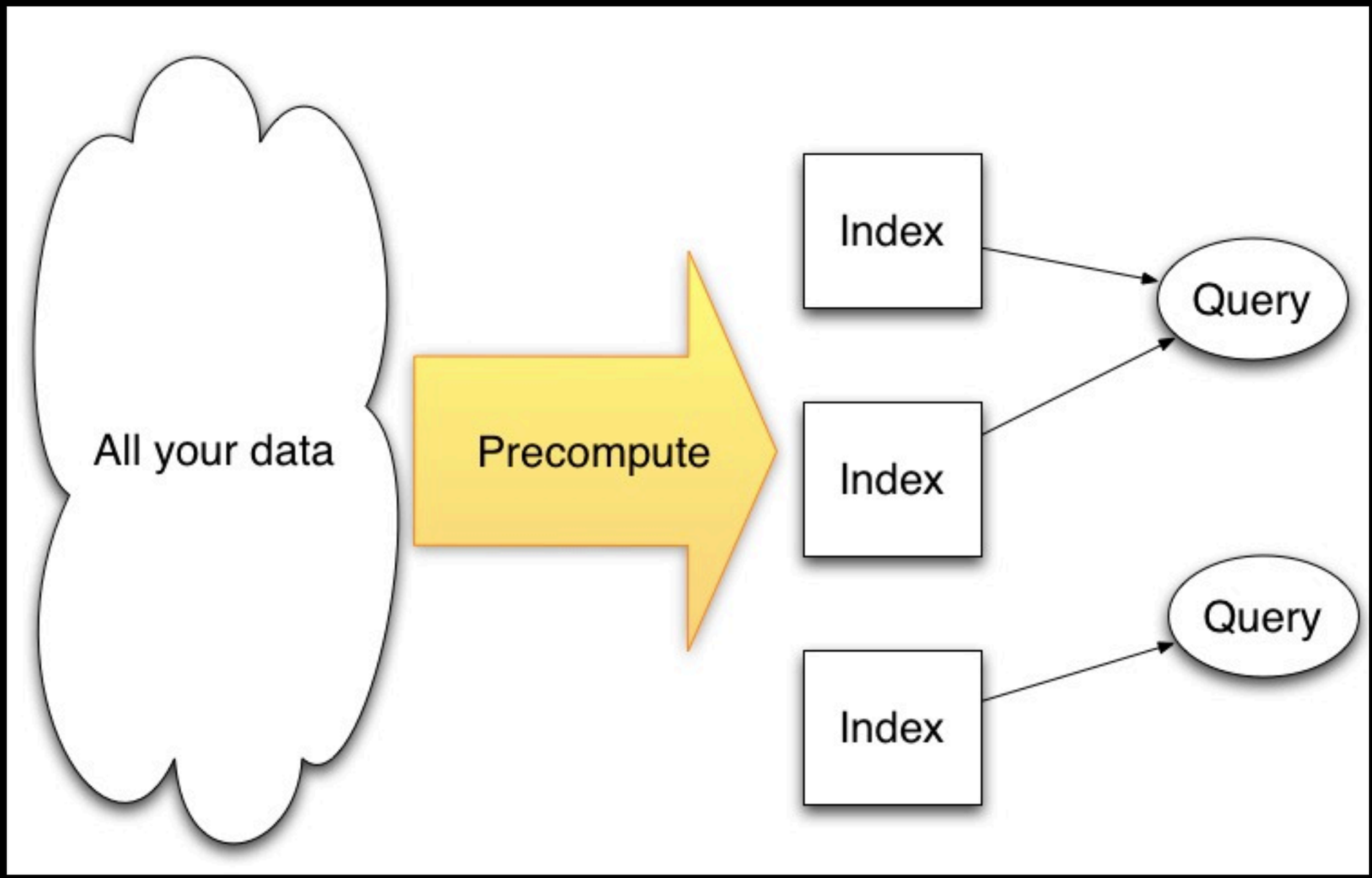
Streaming word count

```
LocalCluster cluster = new LocalCluster();

Map conf = new HashMap();
conf.put(Config.TOPOLOGY_DEBUG, true);

cluster.submitTopology("demo", conf, builder.createTopology());
```

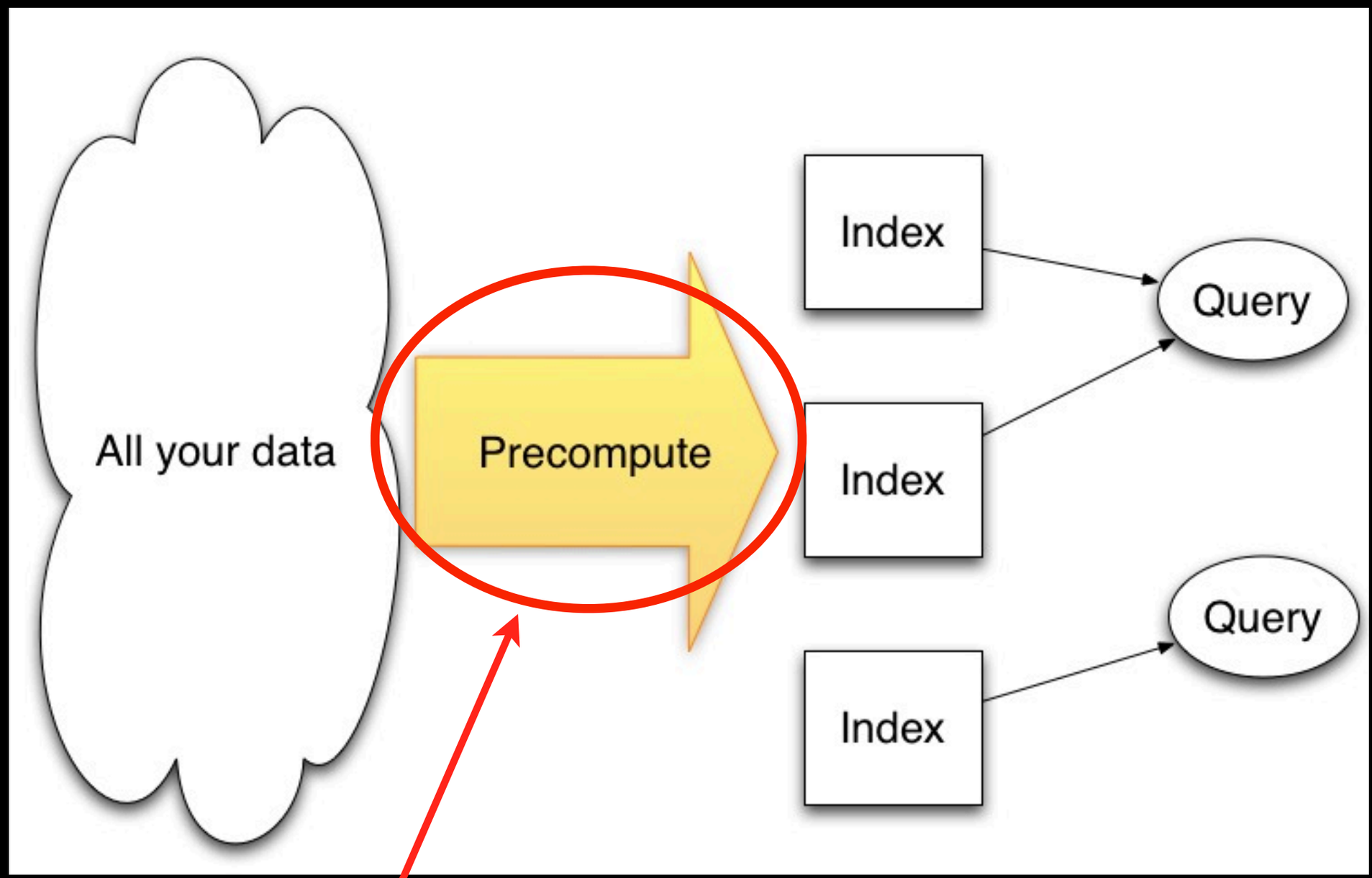
Running topology in local mode


Demo

Traditional data processing

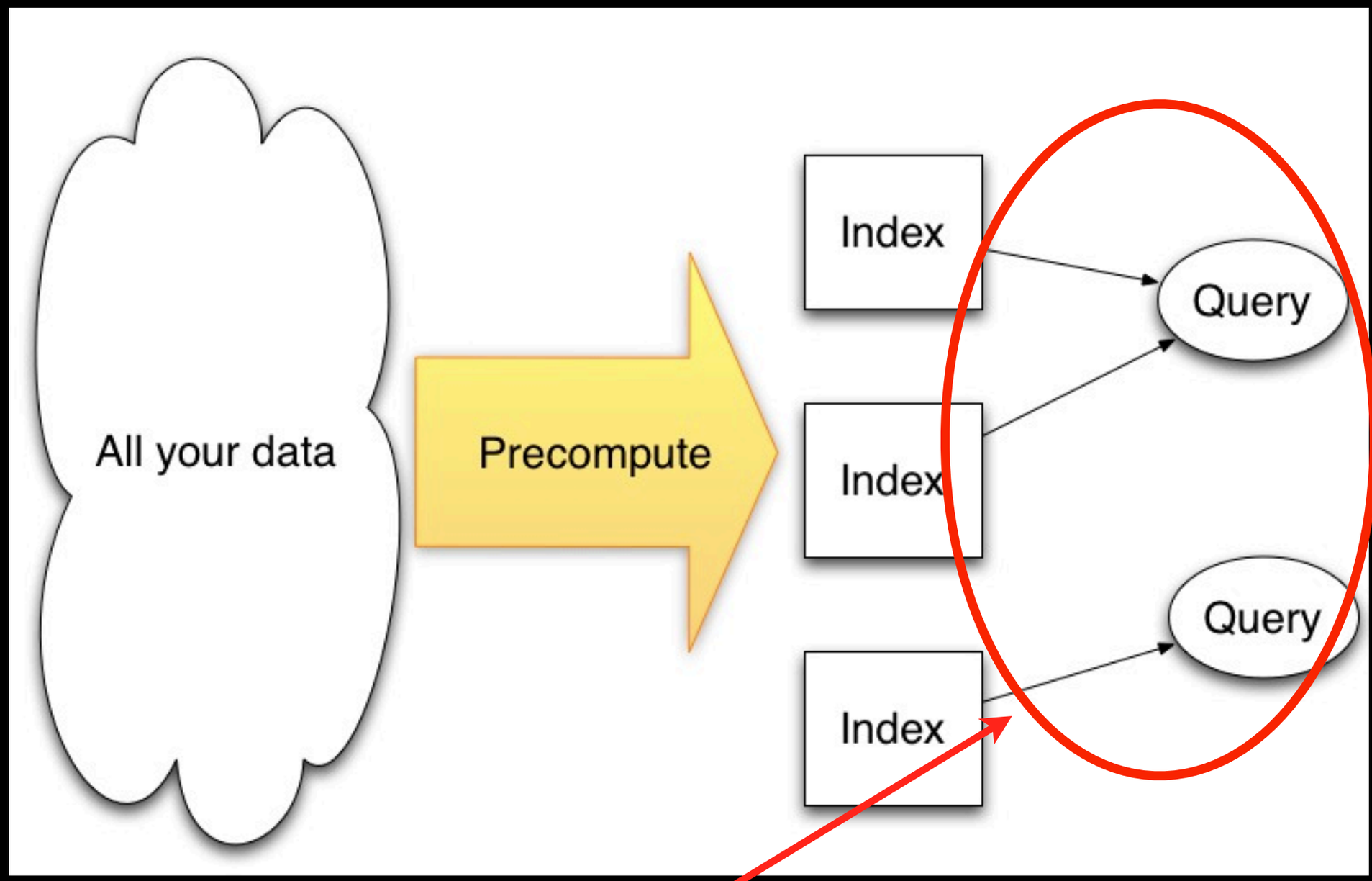


Traditional data processing



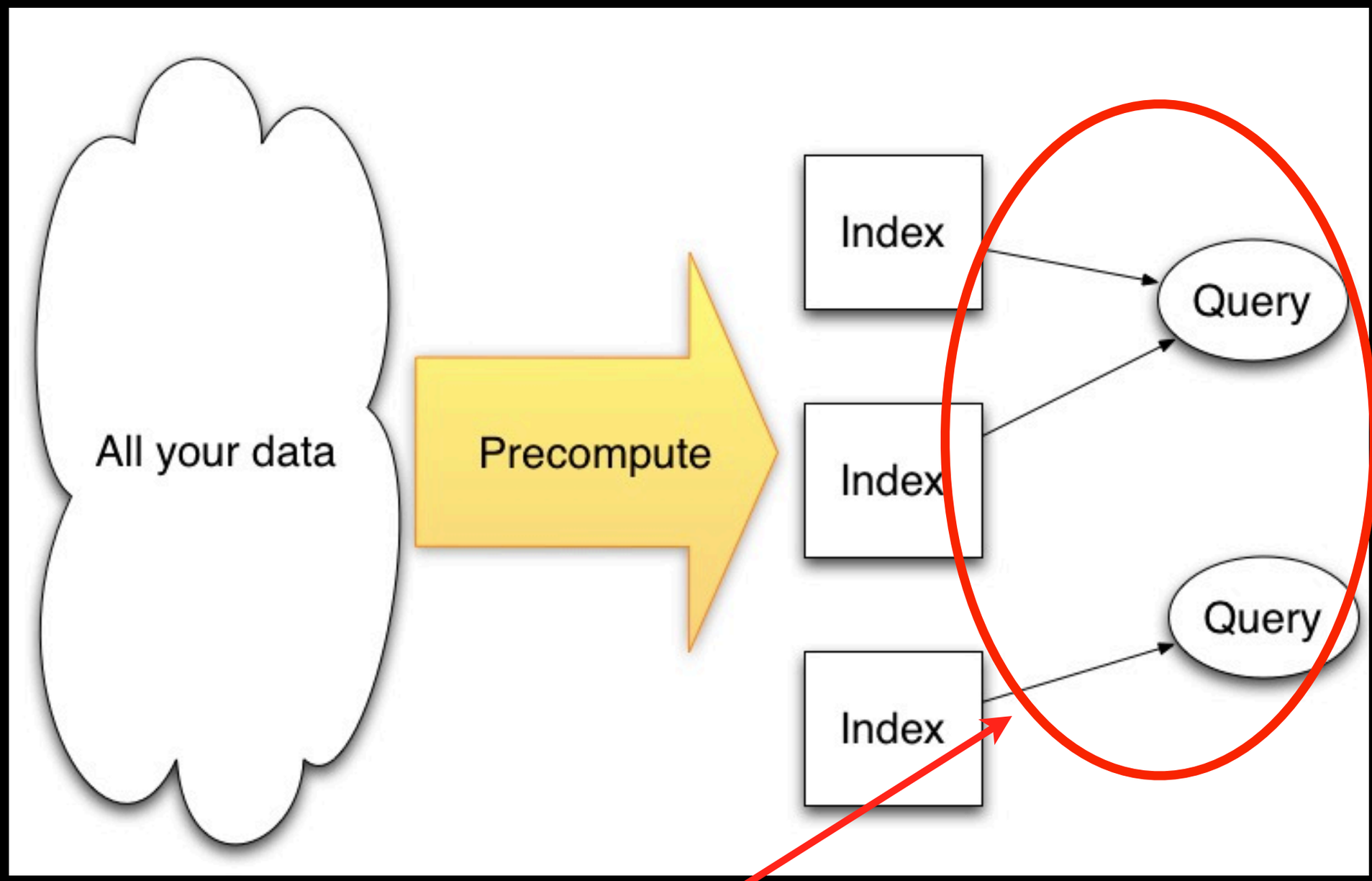
Intense processing (Hadoop, databases, etc.)

Traditional data processing



Light processing on a single machine to resolve queries

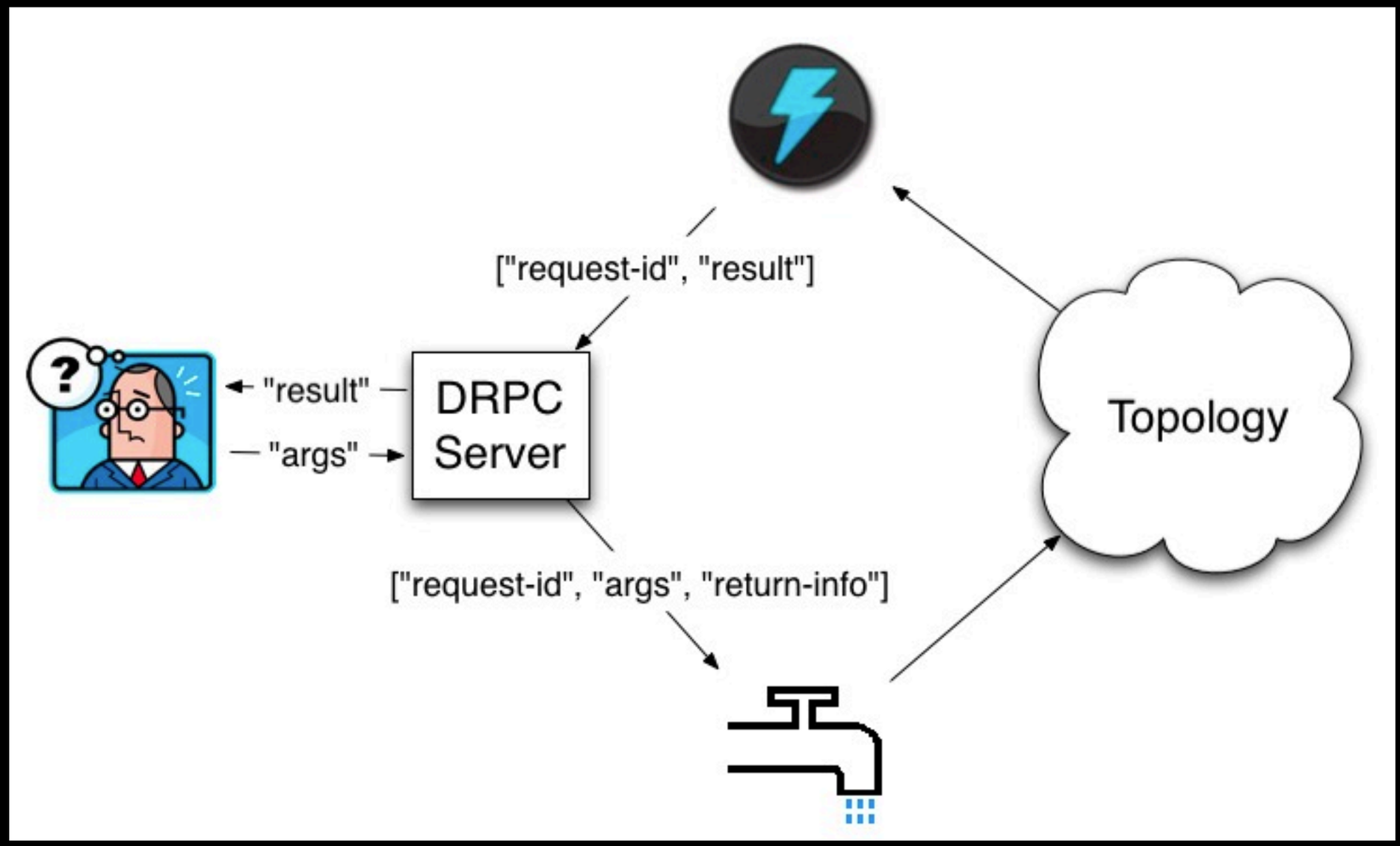
Distributed RPC



Distributed RPC lets you do intense processing at query-time

Game changer

Distributed RPC



Data flow for Distributed RPC

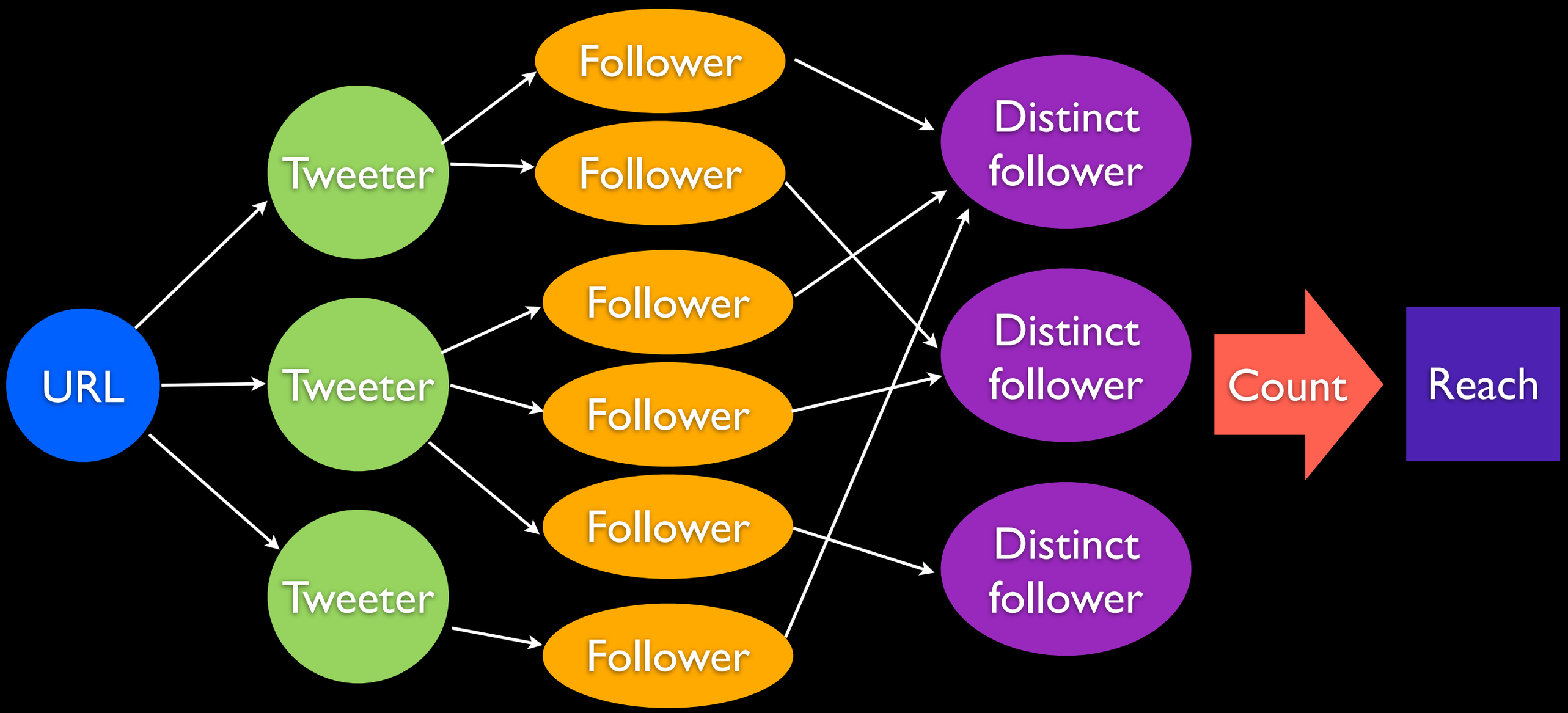
DRPC Example

Computing "reach" of a URL on the fly

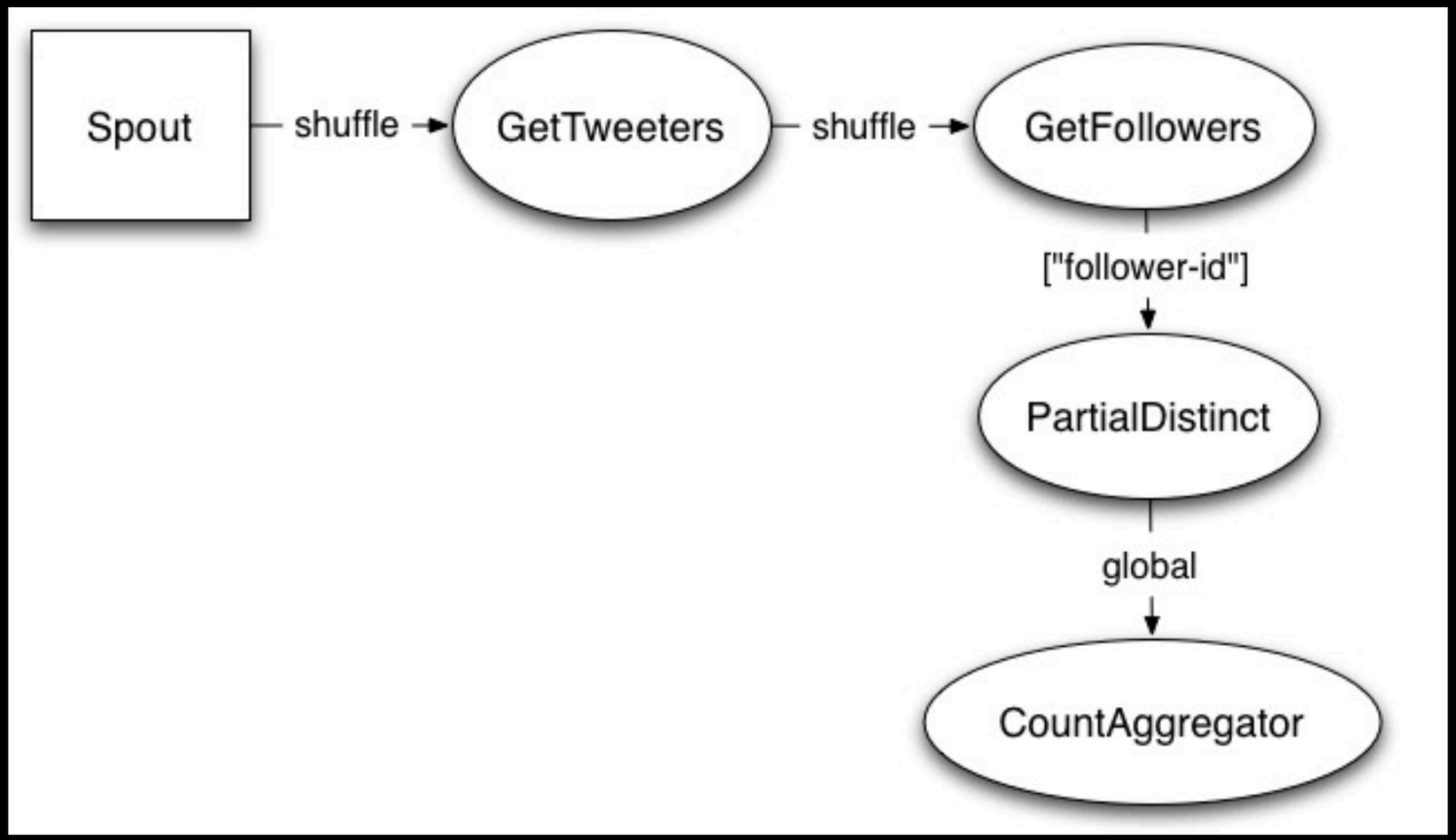
Reach

Reach is the number of unique people exposed to a URL on Twitter

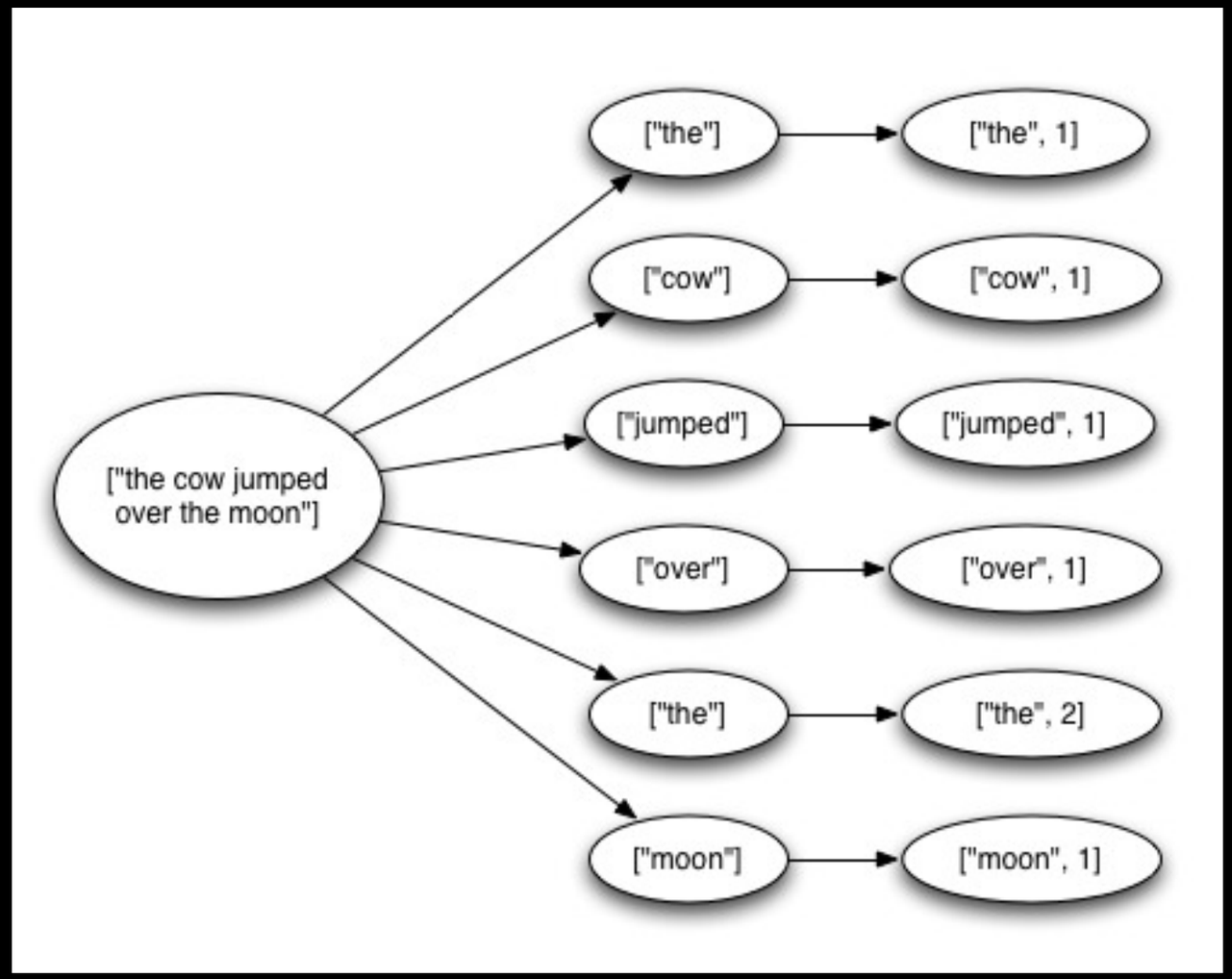
Computing reach



Reach topology



Guaranteeing message processing



"Tuple tree"

Guaranteeing message processing

- A spout tuple is not fully processed until all tuples in the tree have been completed

Guaranteeing message processing

- If the tuple tree is not completed within a specified timeout, the spout tuple is replayed

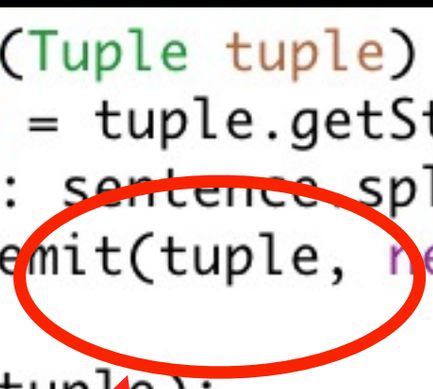
Guaranteeing message processing

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

Reliability API

Guaranteeing message processing

```
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

Guaranteeing message processing

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

Marks a single node in the tree as complete

Guaranteeing message processing

- Storm tracks tuple trees for you in an extremely efficient way

Storm UI

Storm UI

Cluster Summary

Nimbus uptime	Supervisors	Used slots	Free slots	Total slots	Running tasks
23h 44m 53s	20	80	20	100	765

Topology summary

Name	Id	Uptime	Num workers	Num tasks
poseidon	poseidon-1-1314658150	23h 4m 51s	80	765

Supervisor summary

Host	Uptime	Slots	Used slots
ip-10-32-181-48.ec2.internal	23h 24m 30s	5	4
ip-10-98-206-101.ec2.internal	23h 24m 30s	5	4
ip-10-76-89-227.ec2.internal	23h 24m 30s	5	4
ip-10-33-73-238.ec2.internal	23h 24m 30s	5	4
ip-10-79-97-116.ec2.internal	23h 24m 31s	5	4
ip-10-100-87-54.ec2.internal	23h 24m 28s	5	4
ip-10-119-7-24.ec2.internal	23h 24m 29s	5	4
ip-10-78-143-233.ec2.internal	23h 24m 29s	5	4
ip-10-76-67-223.ec2.internal	23h 24m 29s	5	4
ip-10-33-55-63.ec2.internal	23h 24m 31s	5	4
ip-10-205-25-131.ec2.internal	23h 24m 31s	5	4
ip-10-37-62-181.ec2.internal	23h 24m 37s	5	4
ip-10-196-163-95.ec2.internal	23h 24m 28s	5	4

Storm UI

Storm UI

Topology summary

Name	Id	Uptime	Num workers	Num tasks
poseidon	poseidon-1-1314658150	23h 17m 0s	80	765

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	24786020	24786000	4131.688	2338940	0
3h 0m 0s	621695800	621694600	4463.830	59353840	0
1d 0h 0m 0s	4447725560	4447716960	4278.459	438710100	0
All time	4447725560	4447716960	4278.459	438710100	0

Spouts (All time)

Id	Parallelism	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Last error
1	160	877453060	877453060	4278.459	438710100	0	

Bolts (All time)

Id	Parallelism	Emitted	Transferred	Process latency (ms)	Acked	Failed	Last error
-1	4	438716440	438716440	0.009	2223890060	0	
2	160	877451720	877451720	0.320	438725980	0	
3	160	1264258160	1264258160	5.438	438724980	0	
4	18	55946080	55946080	0.215	55946040	0	
5	18	55947280	55947280	0.121	55947280	0	
6	18	55945660	55945660	0.229	55945660	0	
7	18	55946480	55946480	0.145	55946580	0	
8	18	81512620	81512620	0.209	81512620	0	
9	30	438710060	438710060	4205.639	438710140	0	
10	80	163024580	163024580	0.194	81512200	0	

Storm UI

Storm UI

Component summary

Id	Topology	Parallelism
2	poseidon	160

Bolt stats

Window	Emitted	Transferred	Process latency (ms)	Acked	Failed
10m 0s	4640200	4640200	0.319	2320320	0
3h 0m 0s	118884360	118884360	0.308	59441960	0
1d 0h 0m 0s	877670200	877670200	0.320	438835260	0
All time	877670200	877670200	0.320	438835260	0

Input stats (All time)

Component	Stream	Process latency (ms)	Acked	Failed
1	1	0.320	438835260	0

Output stats (All time)

Stream	Emitted	Transferred
-2	438837400	438837400
1	438832800	438832800

Tasks

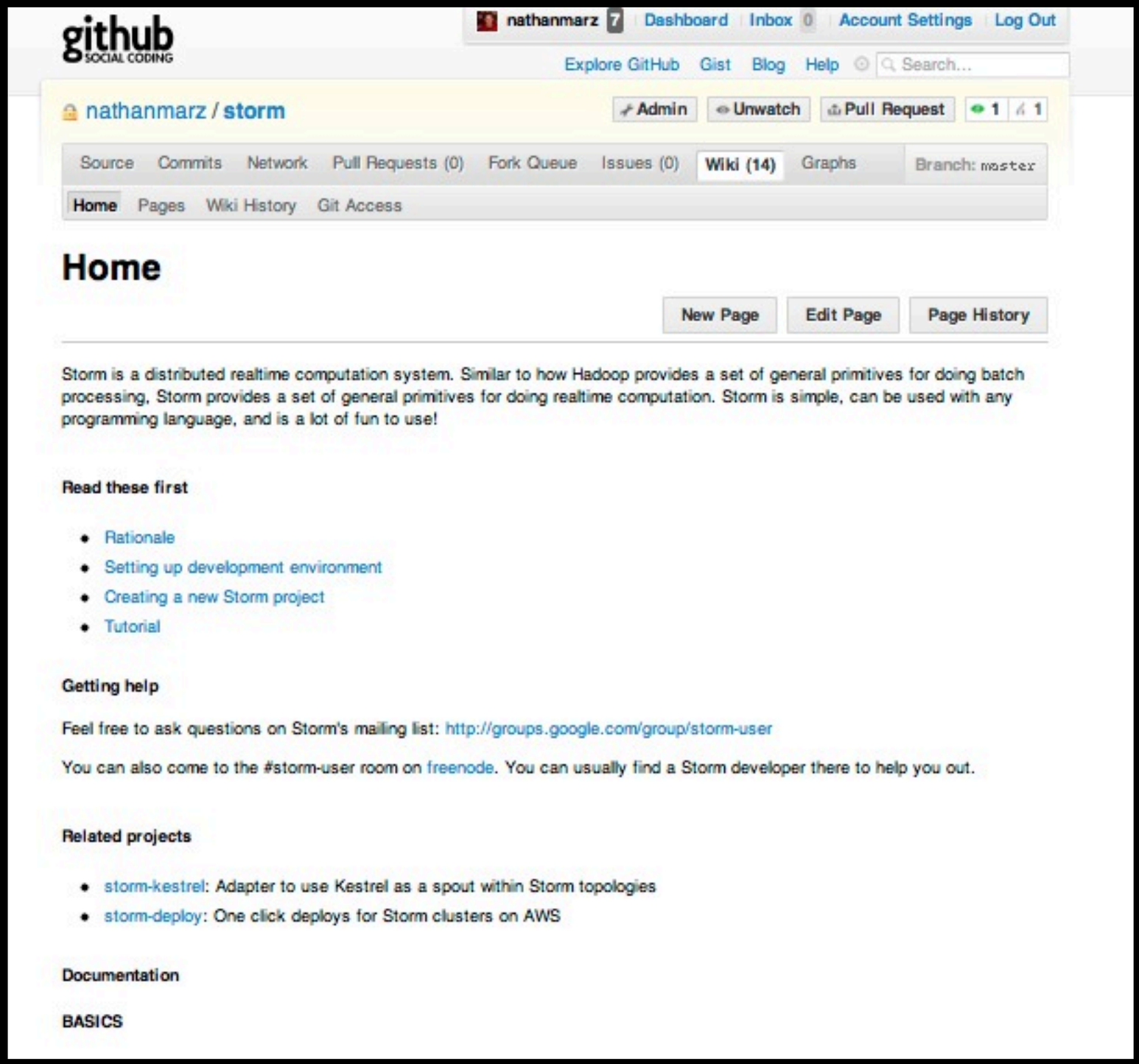
Id	Uptime	Host	Port	Emitted	Transferred	Process latency (ms)	Acked	Failed	Last error
1	23h 17m 0s	ip-10-32-181-48.ec2.internal	6700	5485420	5485420	0.311	2742720	0	
2	23h 16m 57s	ip-10-98-206-101.ec2.internal	6700	5485260	5485260	0.334	2742640	0	
3	23h 17m 9s	ip-10-76-89-227.ec2.internal	6700	5485320	5485320	0.365	2742660	0	
4	23h 17m 10s	ip-10-33-73-238.ec2.internal	6700	5485460	5485460	0.336	2742740	0	
5	23h 17m 9s	ip-10-79-97-116.ec2.internal	6700	5485460	5485460	0.336	2742740	0	

Storm on EC2

https://github.com/nathanmarz/storm-deploy

One-click deploy tool

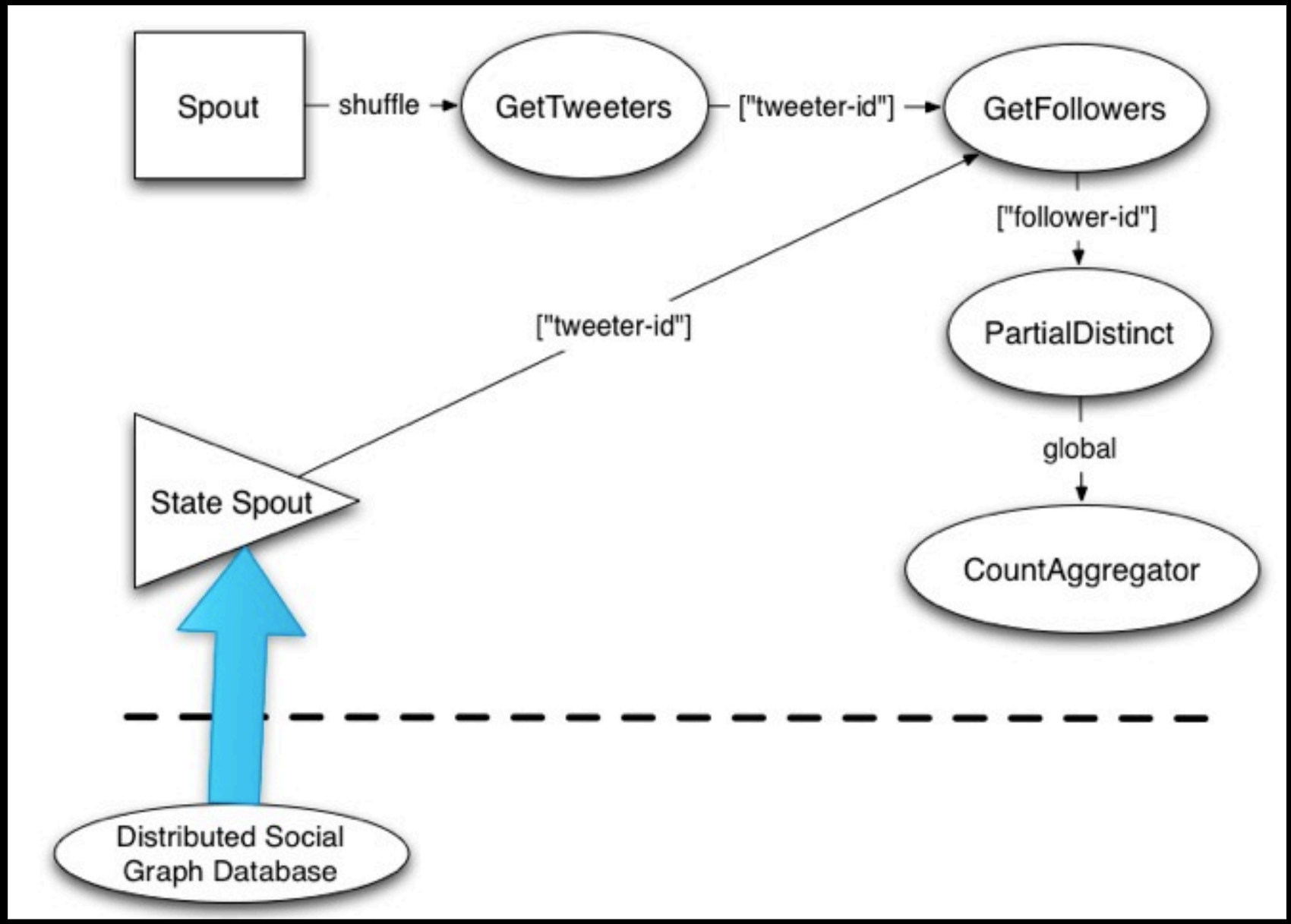
Documentation



State spout (almost done)

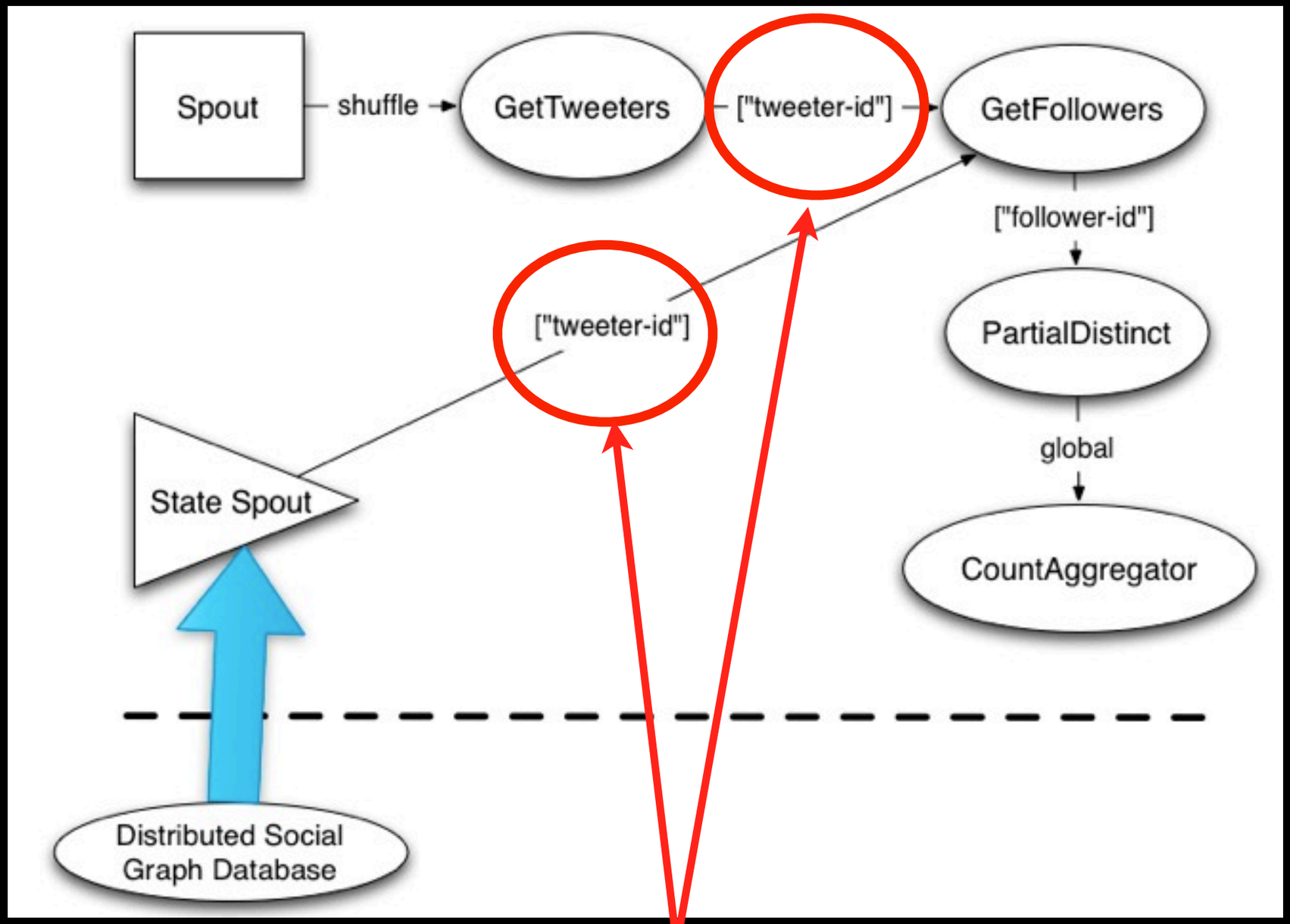
Synchronize a large amount of
frequently changing state into a topology

State spout (almost done)



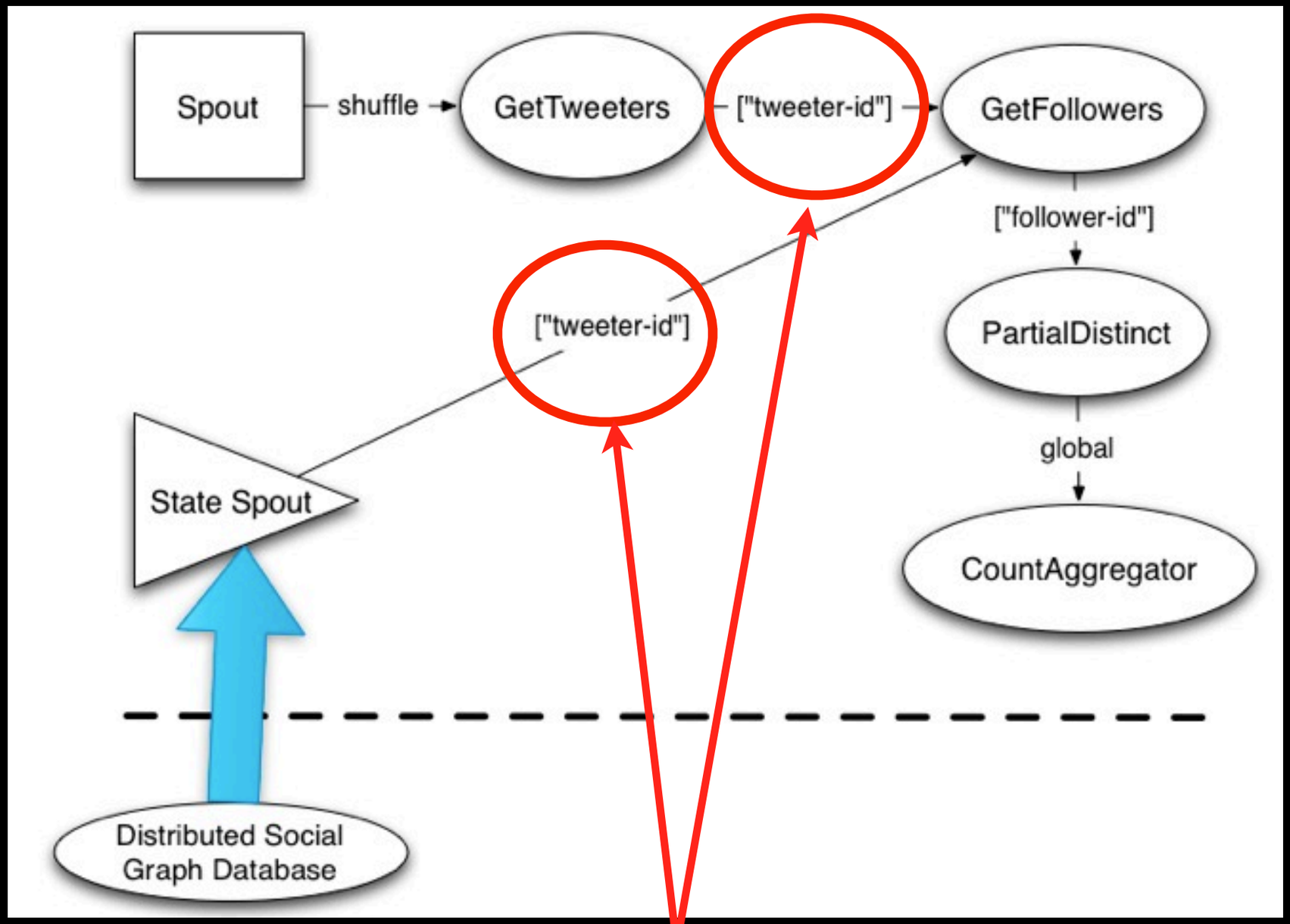
Optimizing reach topology by eliminating the database calls

State spout (almost done)



Each *GetFollowers* task keeps a synchronous cache of a subset of the social graph

State spout (almost done)



This works because *GetFollowers* repartitions the social graph the same way it partitions *GetTweeter*'s stream

Future work

- Storm on Mesos
- “Swapping”
- Auto-scaling
- Higher level abstractions

Questions?

http://github.com/nathanmarz/storm

What Storm does

- Distributes code and configurations
- Robust process management
- Monitors topologies and reassigns failed tasks
- Provides reliability by tracking tuple trees
- Routing and partitioning of streams
- Serialization
- Fine-grained performance stats of topologies