Computer Science 425
Distributed Systems

CS 425 / ECE 428

Fall 2013

Hilfi Alkaff

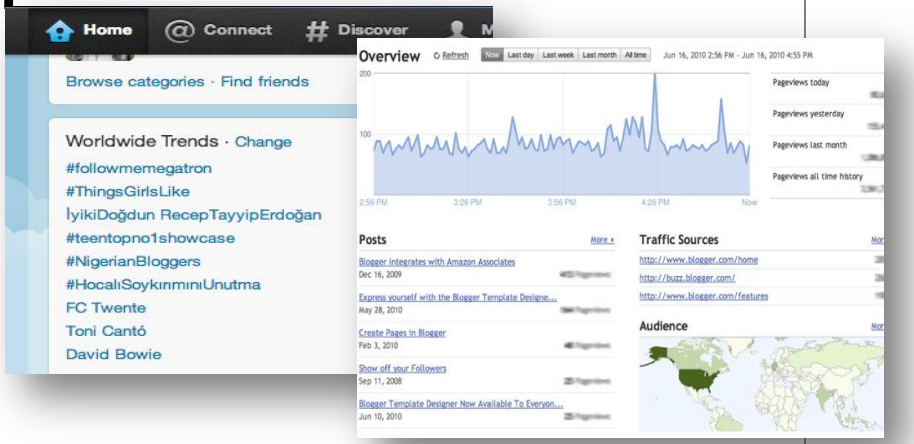
November 5, 2013

Lecture 21

Stream Processing

Motivation

- **Many important applications must process large streams of live data and provide results in near-real-time**
  - **Social network trends**
  - **Website statistics**
  - **Intrustion detection systems**
  - **etc.**

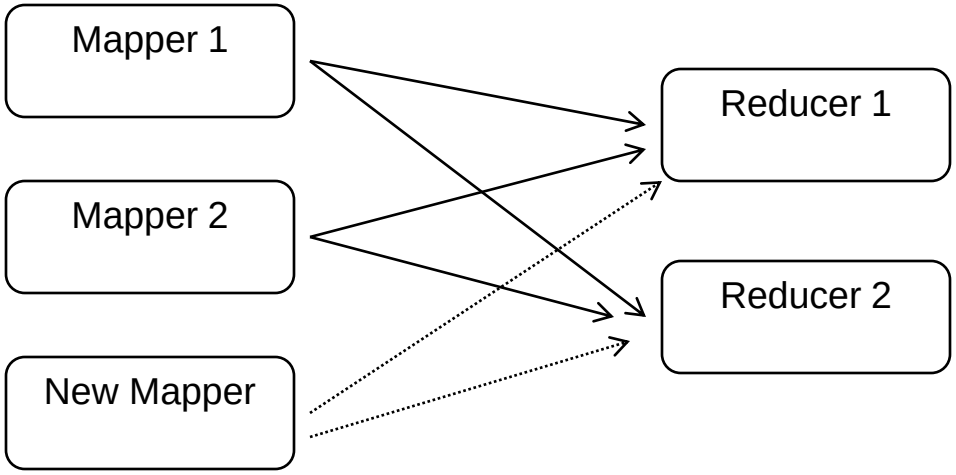


- **Require large clusters to handle workloads**

- **Require latencies of few seconds**

Traditional Solutions (Map-Reduce)

- **No partial answers**
  - **Have to wait for the entire batch to finish**
- **Hard to configure when we want to add more nodes.**



The Need for New Framework

- **Scalable**
- **Second-scale**
- **Easy to program**
- **“Just” works**
  - **Adding/removing nodes should be transparent**

Enter Storm

- Open-sourced at http://storm-project.net/
- Most watched JVM project
- Multiple languages API supported
  - Python, Ruby, Fancy
- Used by over 30 companies including
  - Twitter: For personalization, search
  - Flipboard: For generating custom feeds

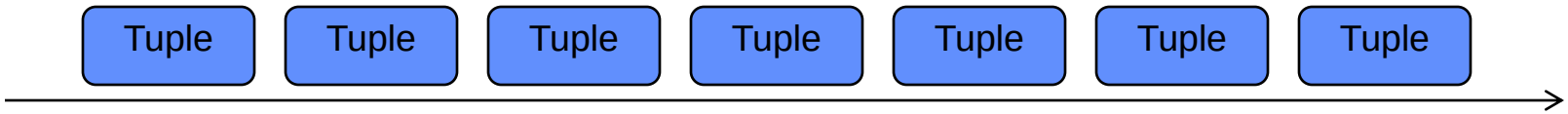
Storm's Terminology

- **Tuples**
- **Streams**
- **Spouts**
- **Bolts**
- **Topologies**

Tuples

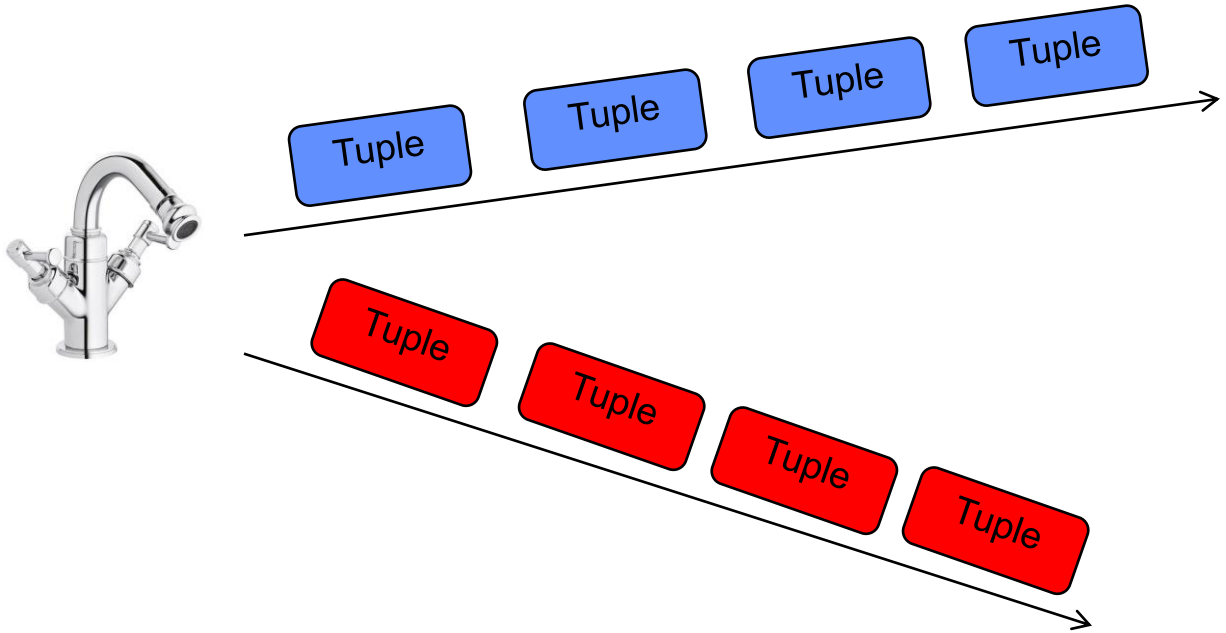
- **Ordered list of elements**
  - **E.g. [“Illinois”, “somebody@illinois.edu”]**

Streams



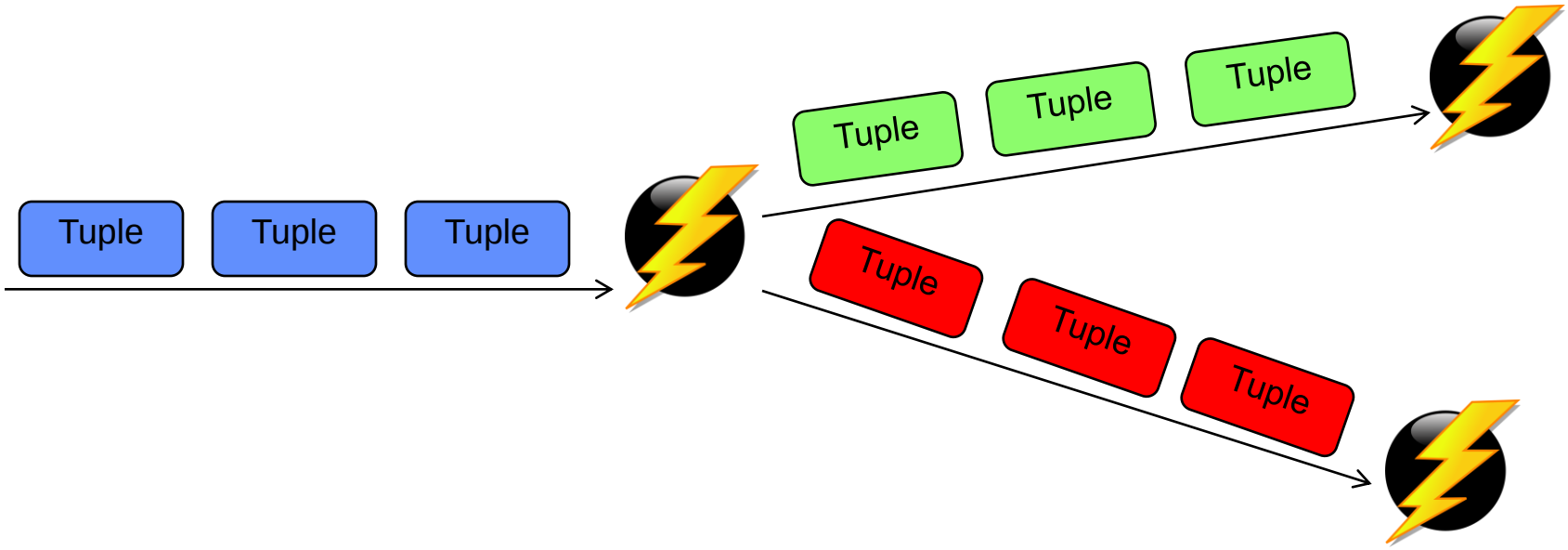
Unbounded sequence of tuples

Spouts



Source of streams (E.g. Web logs)

Bolts



Processes tuples and create new streams

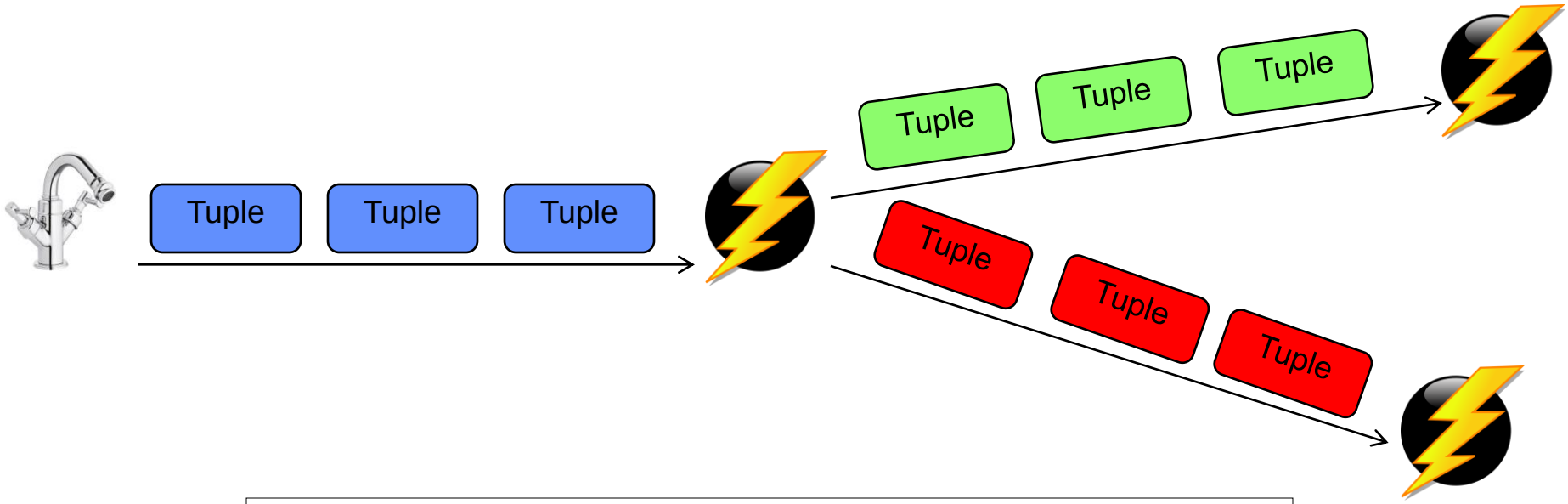
Bolts (Continued)

- **Operations that can be performed**
  - **Filter**
  - **Joins**
  - **Apply/transform functions**
  - **Access DBs**
  - **Etc**
- **Could specify amount of parallelism in each bolt**
  - **How to decide which task in bolt to route subset of the streams to?**

Streams Grouping in Bolts

- **Shuffle Grouping**
  - **Streams are randomly distributed from to the bolt's tasks**
- **Fields Grouping**
  - **Lets you group a stream by a subset of its fields (Example?)**
- **All Grouping**
  - **Stream is replicated across all the bolt's tasks (Example?)**

Topologies



A directed graph of spouts and bolts

Word Count Example (Main)

```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("spout", new RandomSentenceSpout(), 5);

builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
```

Word Count Example (Spout)

```
public class RandomSentenceSpout extends BaseRichSpout {
  SpoutOutputCollector _collector;
  Random _rand;


  @Override
  public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
    _collector = collector;
    _rand = new Random();
  }

  @Override
  public void nextTuple() {
    Utils.sleep(100);
    String[] sentences = new String[]{ "the cow jumped over the moon", "an apple a day keeps the doctor away",
        "four score and seven years ago", "snow white and the seven dwarfs", "i am at two with nature" };
    String sentence = sentences[_rand.nextInt(sentences.length)];
    _collector.emit(new Values(sentence));
  }

  @Override
  public void ack(Object id) {
  }

  @Override
  public void fail(Object id) {
  }

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

Word Count Example (Bolt)

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

  @Override
  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));
  }

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

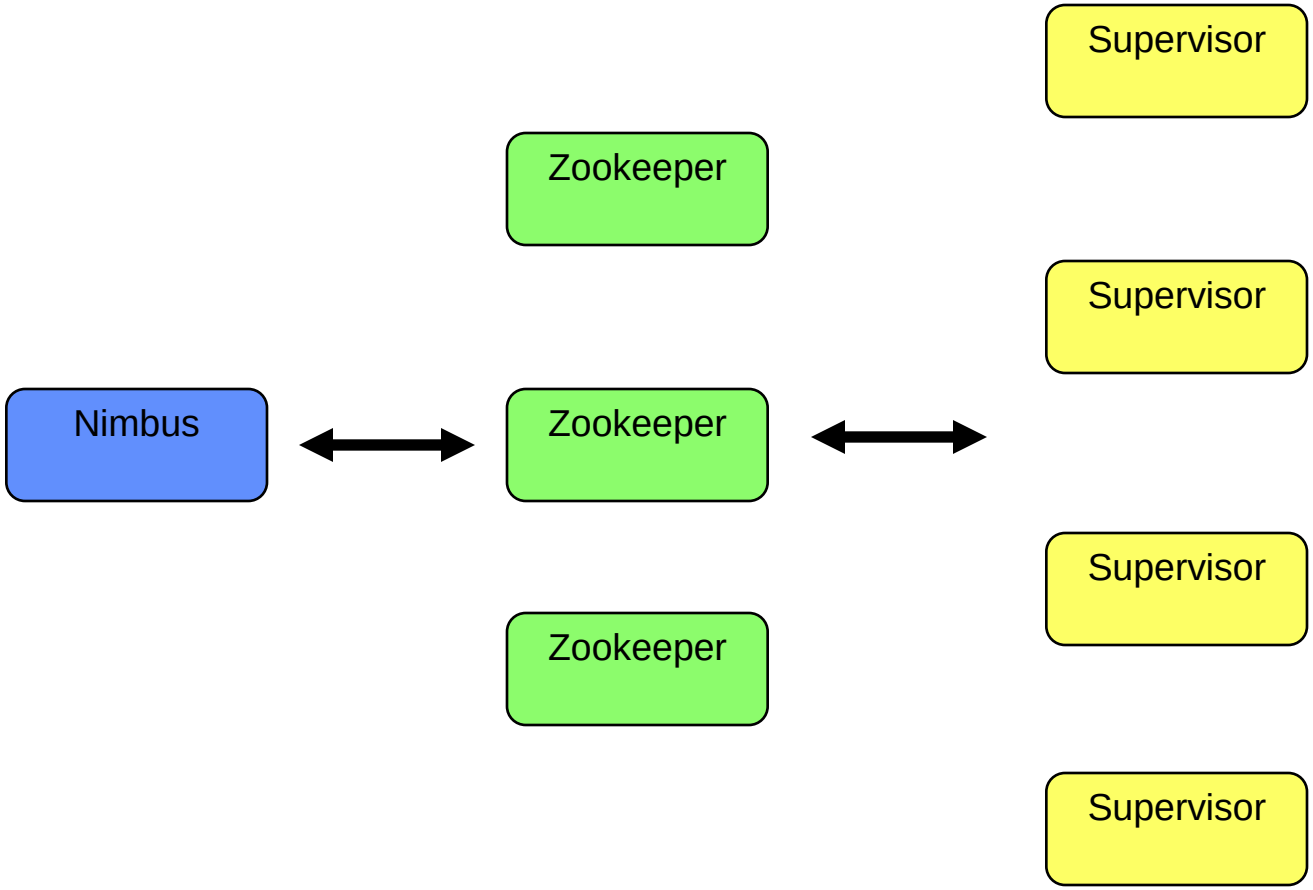
Word Count (Running)

```
LocalCluster cluster = new LocalCluster();
cluster.submitTopology("word-count", conf, builder.createTopology());
```

Storm Cluster

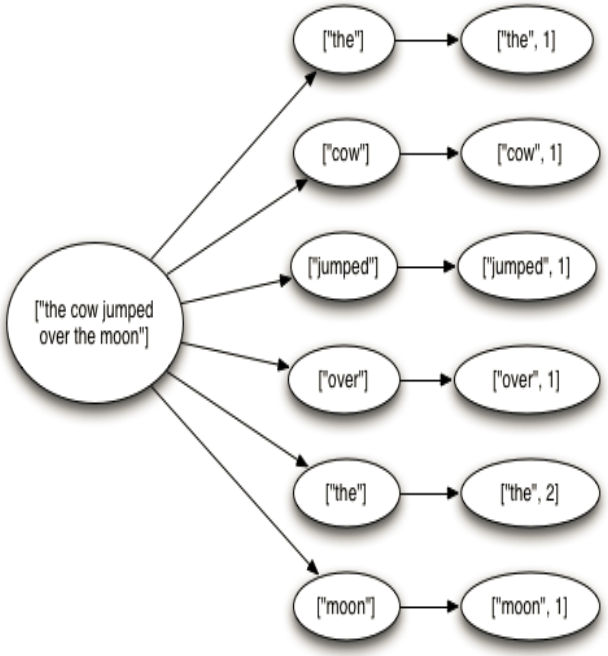
- **Master node**
  - Runs a daemon called Nimbus
  - Responsible for distributing code around cluster
  - Assigning tasks to machines
  - Monitoring for failures
- **Worker node**
  - Runs a daemon called Supervisor
  - Listens for work assigned to its machines
- **Zookeeper**
  - Coordinates Nimbus and Supervisors communication
  - All state of Supervisor and Nimbus is being kept here

Storm Cluster



Guaranteeing Message Processing

- When spout produces tuples, the followings are created:
  - Tuple for each word in the sentence
  - Tuple for the updated count for each word
- A tuple is considered failed when its tree of messages fails to be fully processed within a specified timeout



Anchoring

- **When a word is being anchored, the spout tuple at the root of the tree will be replayed later on if the word tuple failed to be processed downstream**
  - **Might not necessarily wants this feature**
- **An output can be anchored to more than one input tuple**
  - **Failure of one tuple causes multiple tuples to be replayed**

Miscellaneous for Storm's Reliability API

- **Emit(tuple, output)**
  - **Each word is anchored if you specify the input tuple as the first argument to emit**
- **Ack(tuple)**
  - **Acknowledge that you finish processing a tuple**
- **Fail(tuple)**
  - **Immediately fail the spout tuple at the root of tuple tree if there is an exception from the database, etc**
- **Must remember to ack/fail each tuple**
  - **Each tuple consume memory. Failure to do so results in memory leaks**

What's wrong with Storm?

- **Mutable state can be lost due to failure!**
  - **May update mutable state twice!**
  - **Processes each record at least once**
- **Slow nodes?**
  - **No notion of time slices**

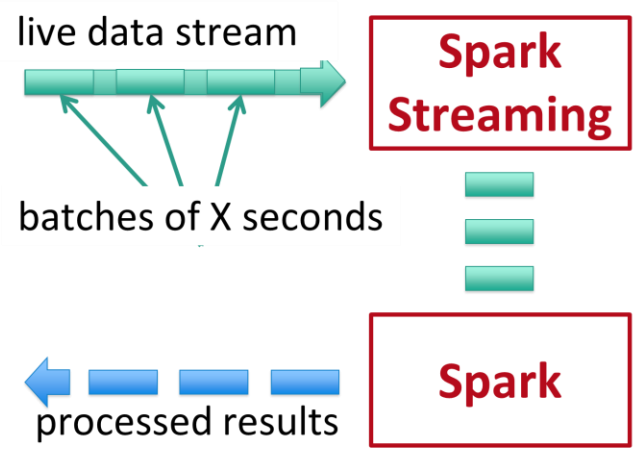
Enter Spark Streaming

- **Research project from AMPLab Group at UC Berkeley**
- **A follow-up from Spark research project**
  - **Idea: cache datasets in memory, dubbed Resilient Distributed Datasets (RDD) for repeated query**
  - **Able to perform 100 times faster than Hadoop MapReduce for iterative machine learning applications**

Discretized Stream Processing

- **Run a streaming computation as a series of very small, deterministic batch jobs**

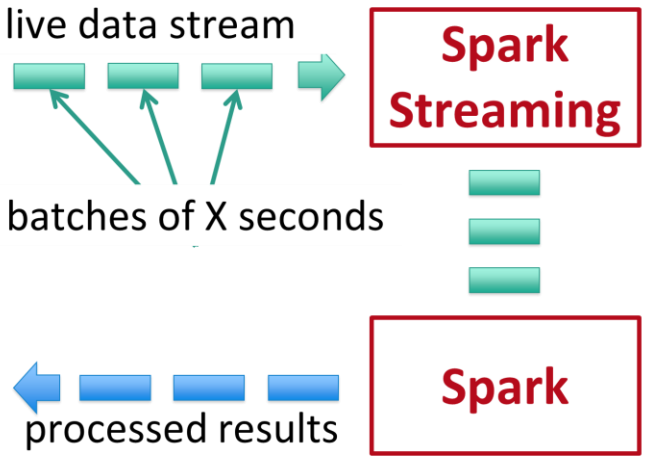
- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches



Discretized Stream Processing (Continued)

- **Run a streaming computation as a series of very small, deterministic batch jobs**

- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches

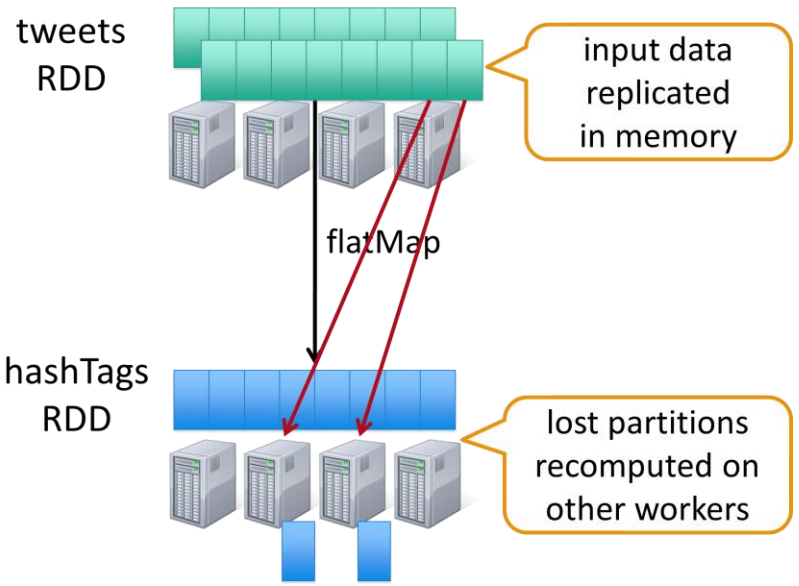


Effects of Discretization

- **(+) Easy to maintain consistency**
  - **Each time interval is processed independently**
    - » **Especially useful when you have stragglers (slow nodes)**
    - » **Not handled in Storm**
- **(+) Handle out-of-order records**
  - **Can choose to wait for a limited "slack time"**
  - **Incrementally correct late records at application level**
- **(+) Parallel recovery**
  - **Expose parallelism across both partitions of an operator and time**
- **(-) Raises minimum latency**

Fault-tolerance in Spark Streaming

- **RDDs are remember the sequence of operations that created it from the original fault-tolerant input data (Dubbed “lineage graph”)**
- **Batches of input data are replicated in memory of multiple worker nodes, therefore fault-tolerant**
- **Data lost due to worker failure, can be recomputed from input data**

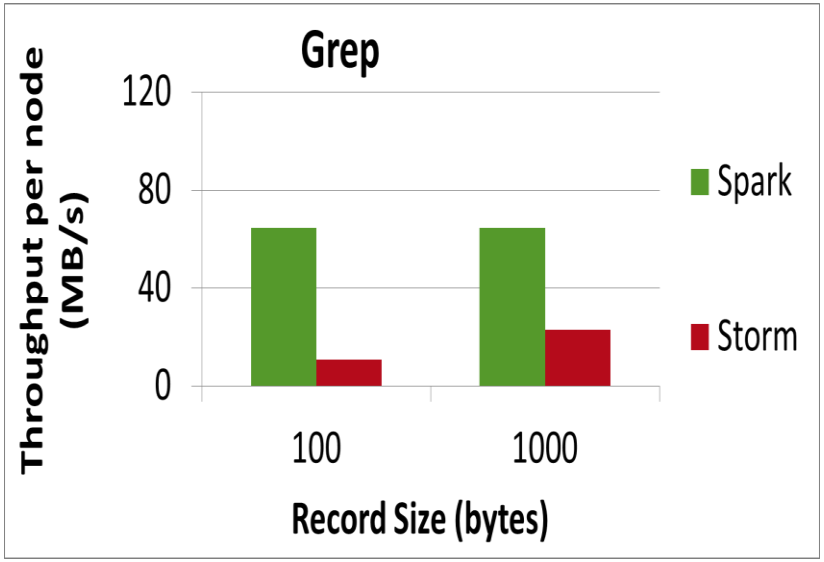


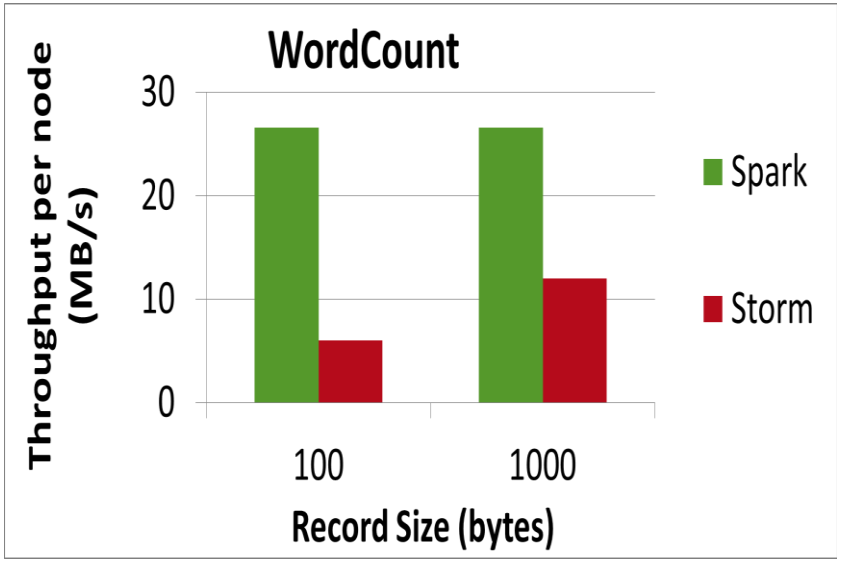
Fault-tolerant Stateful Processing

- **State data not lost even if a worker node dies**
  - **Does not change the value of your result**
- **Exactly once semantics to all transformations**
  - **No double counting!**
- **Recovers from faults/stragglers within 1 sec**

Comparison with Storm

- **Higher throughput than Storm**
  - **Spark Streaming: 670k records/second/node**
  - **Storm: 115k records/second/node**





More information

- **Storm**
  - https://github.com/nathanmarz/storm/wiki/Tutorial
  - https://github.com/nathanmarz/storm-starter
- **Spark Streaming**
  - http://spark.incubator.apache.org/docs/latest/streaming-programming-guide.html
- **Other Streaming Frameworks**
  - **Yahoo S4**
  - **LinkedIn Samza**