STREAM PROCESSING

BIG DATA PROCESSING

Félix Cuadrado

felix.cuadrado@qmul.ac.uk

Queen Mary University of London

School of Electronic Engineering and Computer Science

Contents

Queen Mary University of London

Information streams

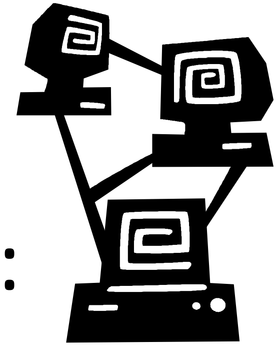
- Data is continuously generated from multiple sources
  - Messages from a social platform (e.g. Twitter)
  - Network traffic going over a switch
  - Readings from distributed sensors
  - Interactions of users with a web application
- For faster analytics, we might need to process the information the moment it is generated
  - Process the information streams

Streams – A Brave New World

- Batch processing: data stored in *finite, persistent data sets*
- Data Streams: distributed, continuous, unbounded, rapid, time varying, noisy, . . .
- Data-Stream Management: variety of modern applications
  - Network monitoring and traffic engineering
  - Sensor networks
  - Telecom call-detail records
  - Network security
  - Financial applications
  - Manufacturing processes
  - Web logs and clickstreams
  - Other massive data sets…

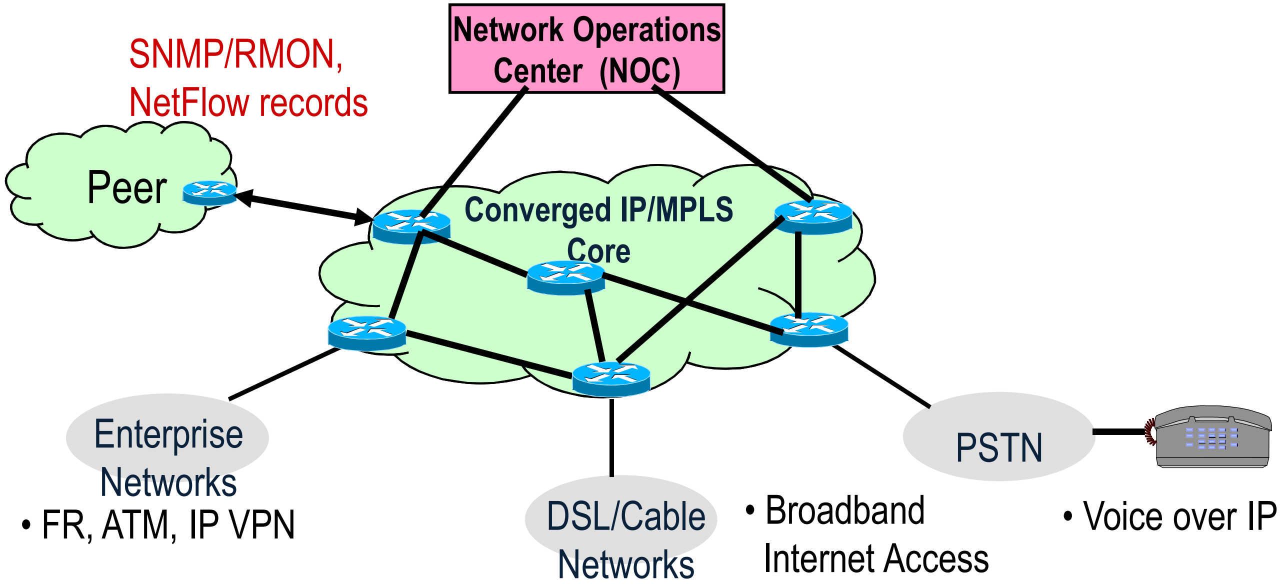
Example: IP Network Data

- Networks are sources of massive data: the metadata per hour per IP router is gigabytes
- Fundamental problem of data stream analysis: *Too much information to store or transmit*
- So process data as it arrives – *One pass, small space: the data stream approach*
- *Approximate answers* to many questions are OK, if there are guarantees of result quality



IP Network Monitoring Application

- 24x7 IP packet/flow data-streams at network elements
- Truly massive streams arriving at rapid rates
    - DEC-IX: 5Tb/sec peak traffic
- Often shipped off-site to data warehouse for off-line analysis

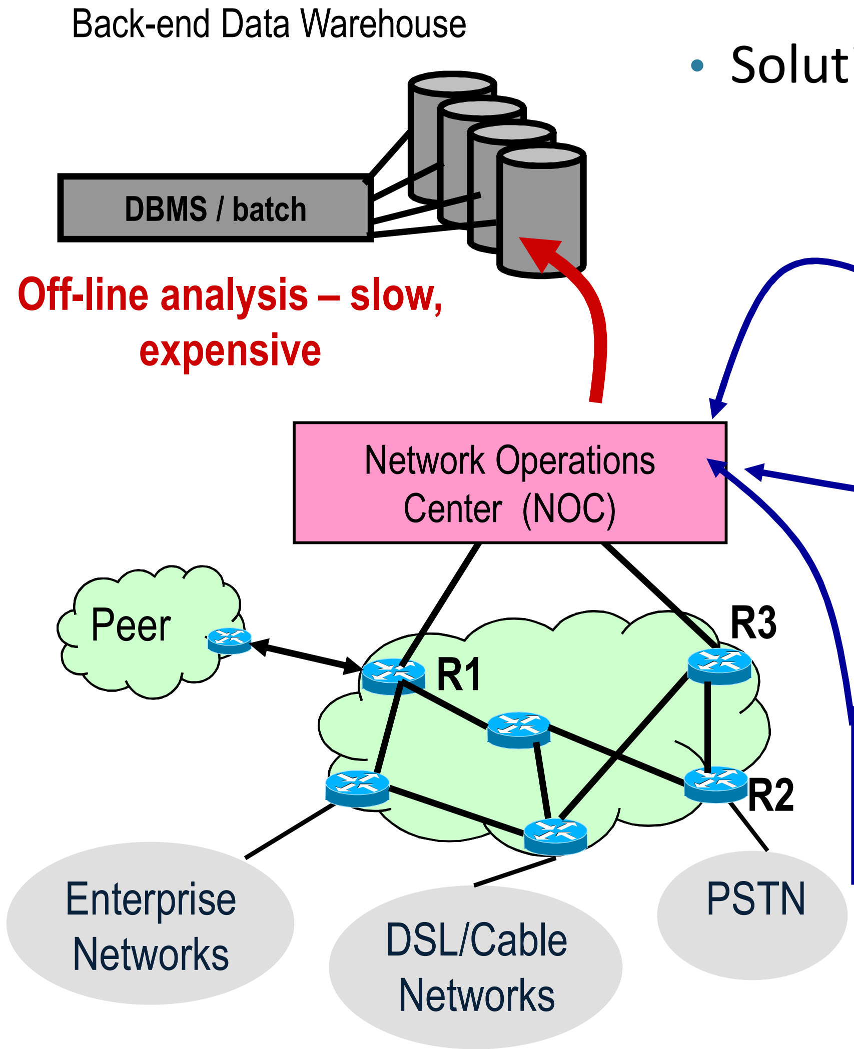


Source	Destination	Duration	Bytes	Protocol
10.1.0.2	16.2.3.7	12	20K	http
18.6.7.1	12.4.0.3	16	24K	http
13.9.4.3	11.6.8.2	15	20K	http
15.2.2.9	17.1.2.1	19	40K	http
12.4.3.8	14.8.7.4	26	58K	http
10.5.1.3	13.0.0.1	27	100K	ftp
11.1.0.6	10.3.4.5	32	300K	ftp
19.7.1.2	16.5.5.8	18	80K	ftp

Example NetFlow IP Session Data

Network Monitoring Queries

- Extra complexity comes from *limited space and time*
- Solutions exist for these and other problems



What are the top (most frequent) 1000 (source, dest) pairs seen over the last month?

How many distinct (source, dest) pairs have been seen by both R1 and R2 but not R3?

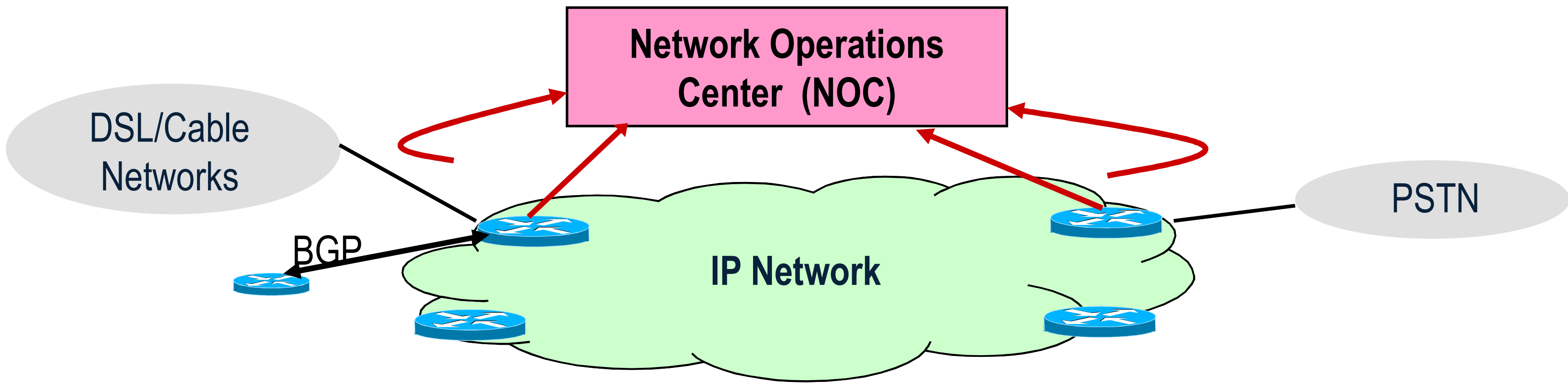
Set-Expression Query

```
SELECT COUNT (R1.source, R2.dest)
FROM  R1, R2
WHERE R1.dest = R2.source
```

SQL Join Query

Real-Time Data-Stream Analysis

- Must process network streams in *real-time* and *one pass*
- Critical NM tasks: fraud, DoS attacks, SLA violations
  - Real-time traffic engineering to improve utilization
- *Tradeoff result accuracy vs. space/time/communication*
  - Fast responses, small space/time
  - Minimize use of communication resources

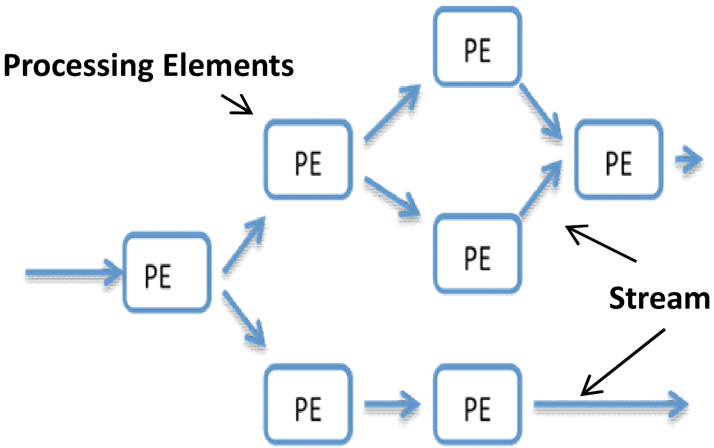


Contents

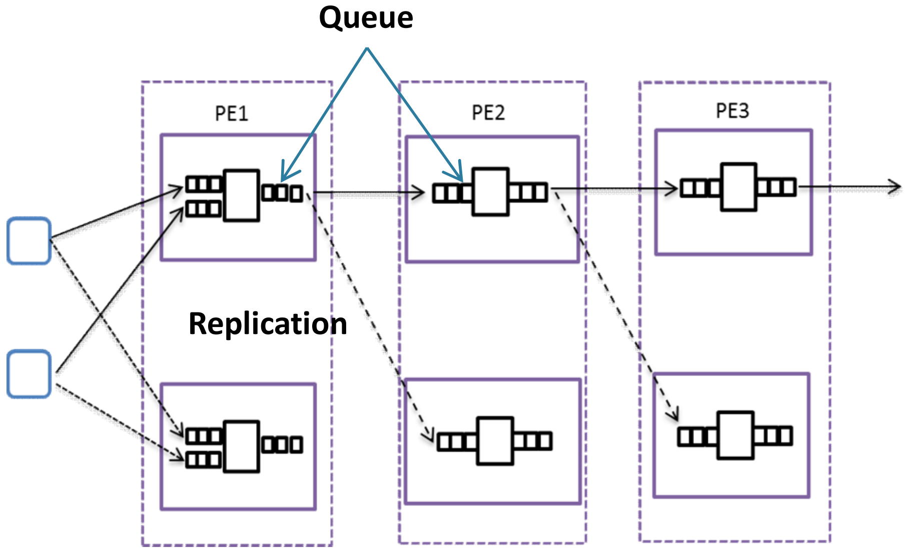
- Information Streams
- **Stream Processing - Storm**
- Micro batch processing - DStream

Stream Processing

- Stream – Sequence of unbounded tuples



Macro view



Microscopic View

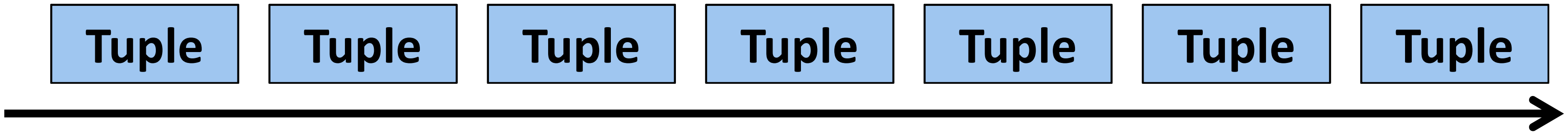
Apache Storm

- Developed by BackType which was acquired by Twitter. Now donated to Apache foundation
- Storm provides realtime computation of data streams
  - Scalable (distribution of blocks, horizontal replication)
  - Guarantees no data loss
  - Extremely robust and fault-tolerant
  - Programming language agnostic

Storm Concepts

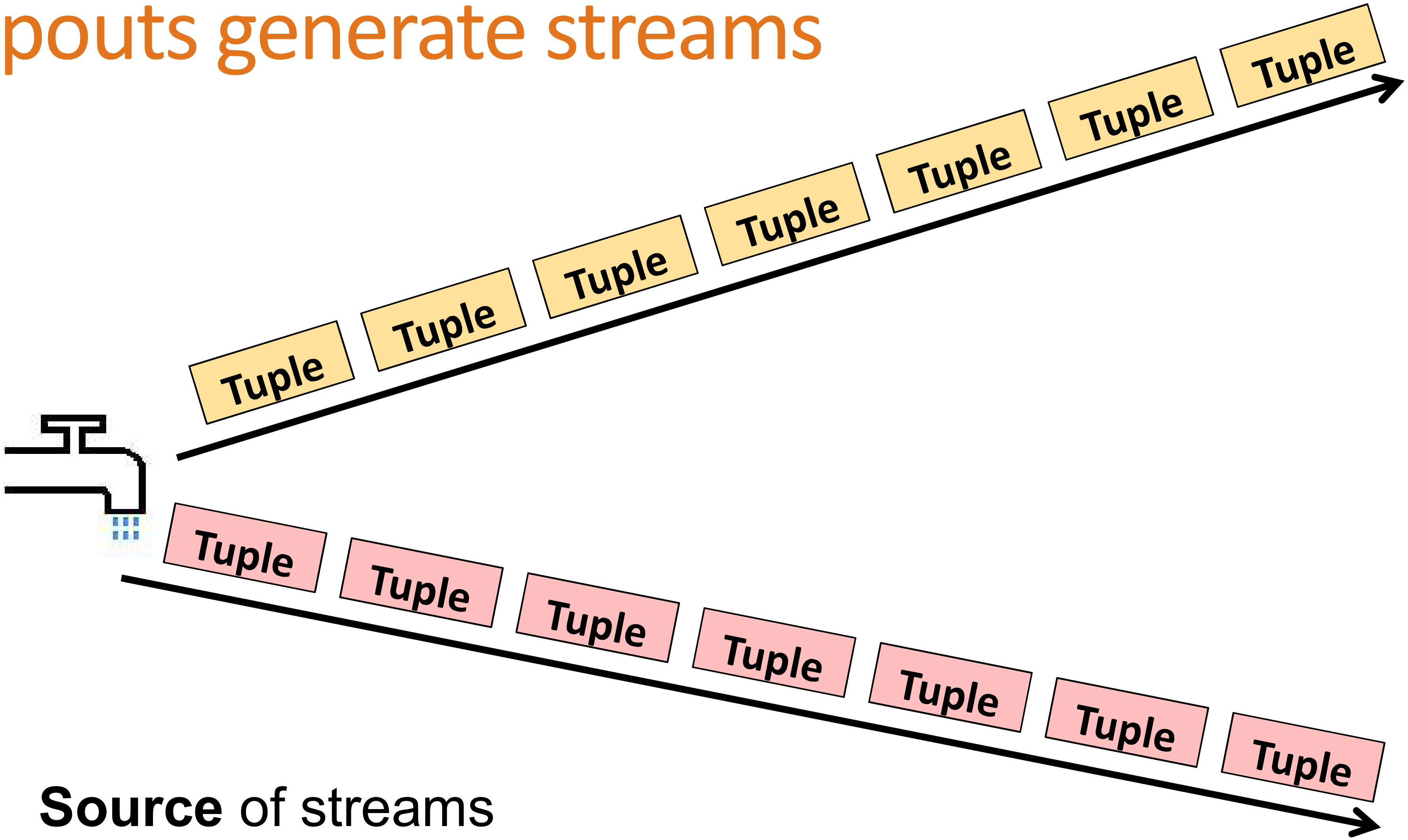
- Streams
  - Of messages arriving to bolts
- Spouts
  - Generating message streams
- Bolts
  - Consuming and generating message streams
- Topologies
  - Data flows of spouts, streams and bolts

Streams



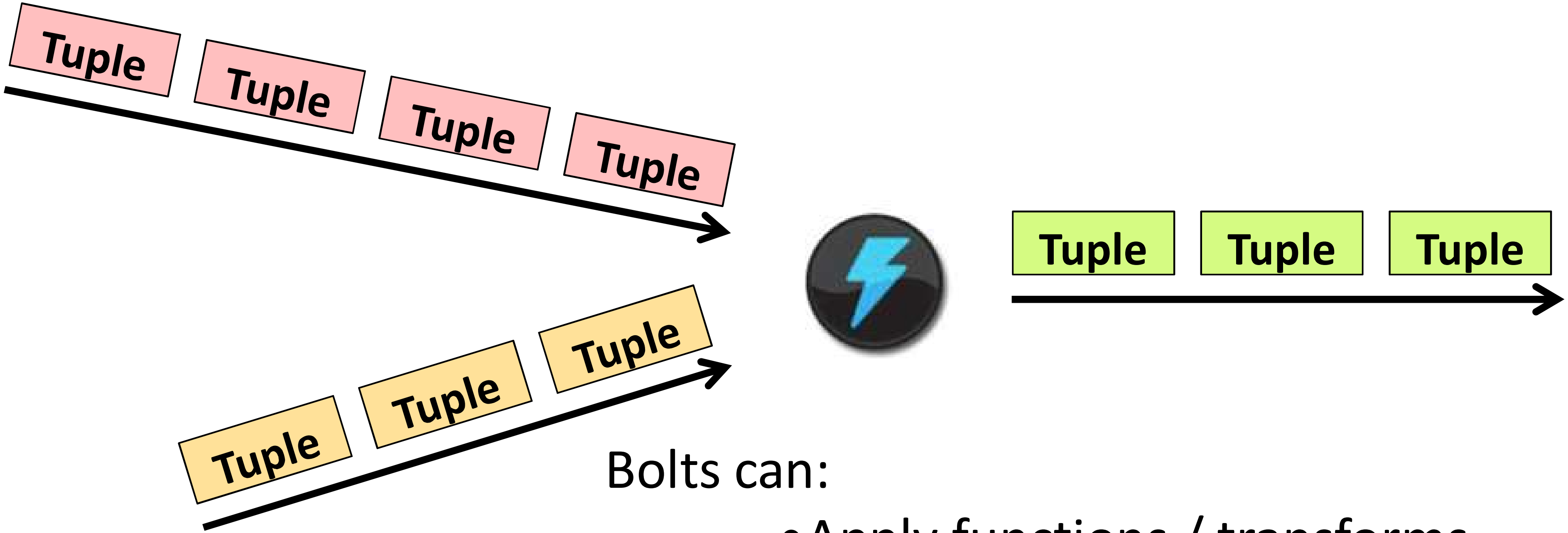
Unbounded sequence of tuples

Spouts generate streams



Source of streams

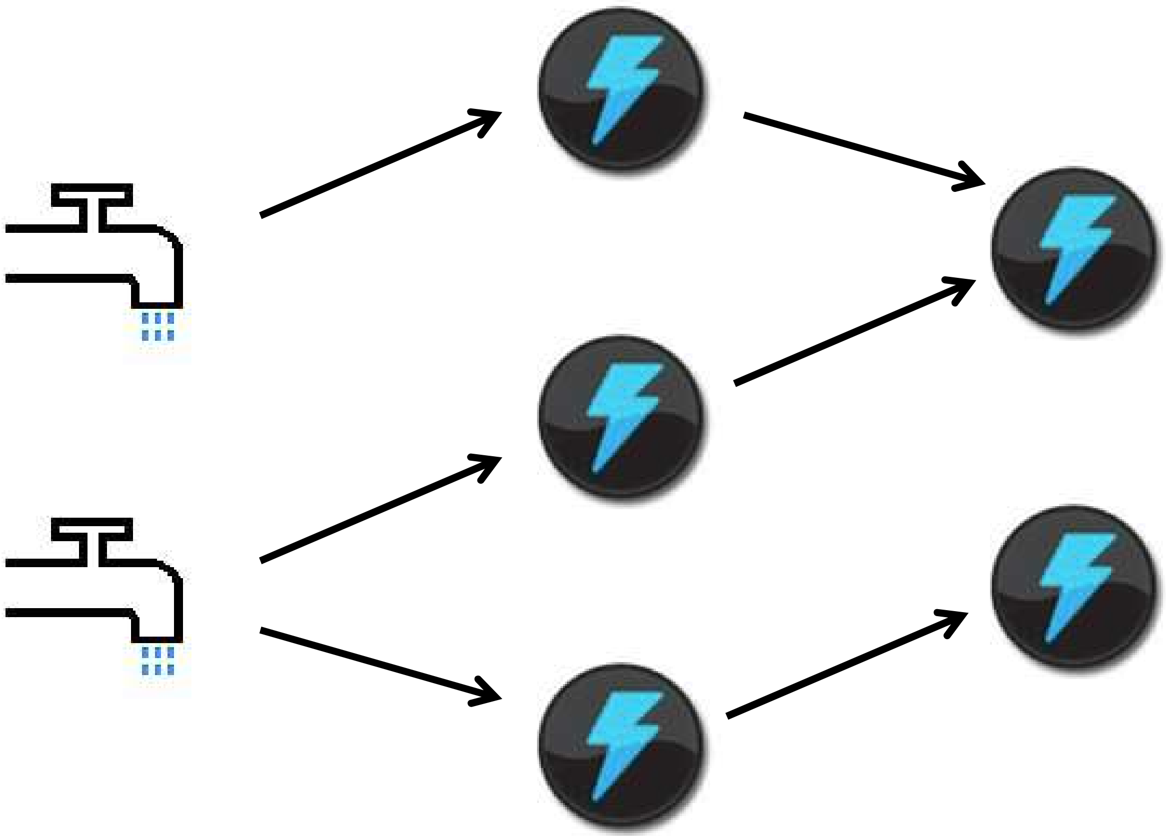
Bolts



Bolts can:

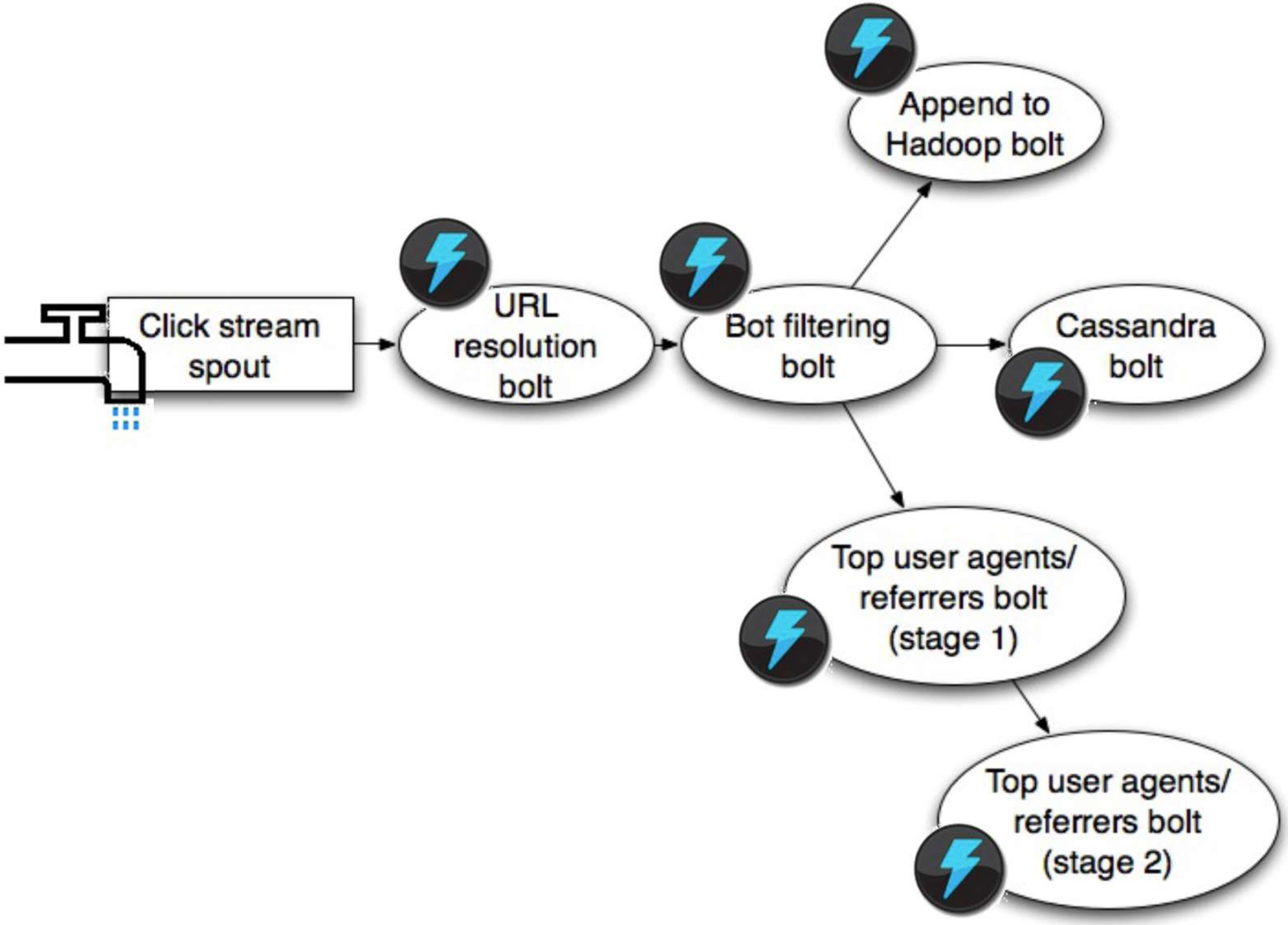
- Apply functions / transforms
- Filter
- Aggregation
- Stream joining or splitting
- Access DBs, APIs, etc

Topology



Network of spouts and bolts

Sample topology: Website click analysis

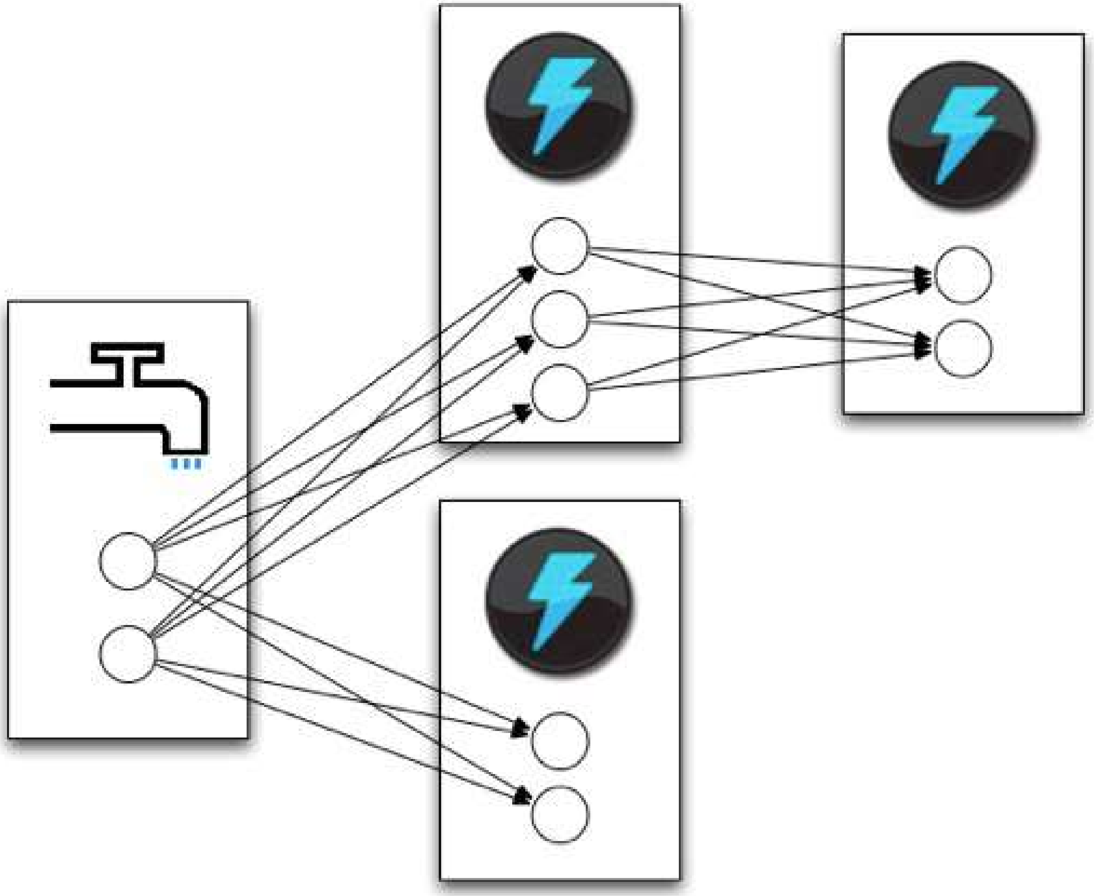


Storm scalability

- Distribute bolts and spouts in different nodes of the cluster

- Run multiple replicas of the bolt / spout
  - Messages are grouped and sent to different replicas
    - Fields grouping: hash-based partitions
    - Random grouping: purely random, balanced
  - Problem: no longer can keep global state

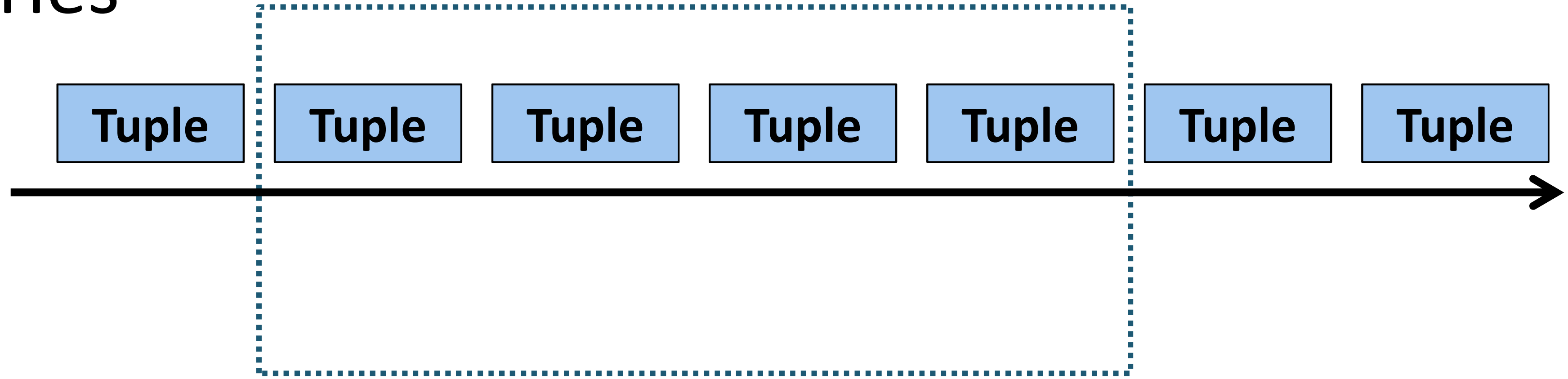
Storm Topology



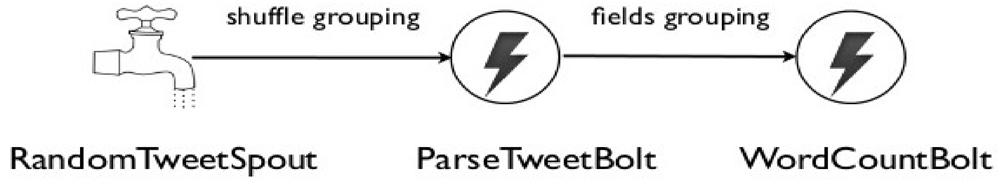
Spouts and bolts execute as
many tasks across the cluster
Horizontal scaling/parallelism

Sliding Window

- Some bolts have to access more than the current tuple in order to perform its computation
- A sliding window stores a rolling list with the latest items from the stream
- Contents change over time, replaced by new entries



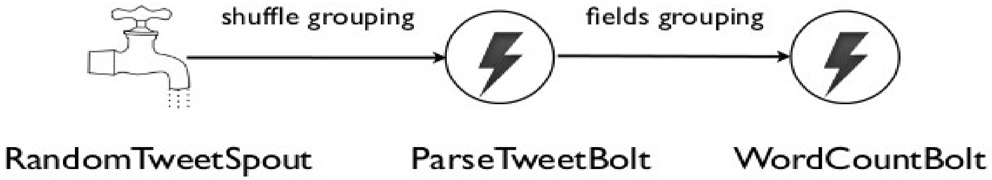
Streaming word count




```java
class ParseTweetBolt extends BaseBasicBolt {

  @Override
  public void execute(Tuple tuple, BasicOutputCollector collector) {
    String tweet = tuple.getString(0);
    for (String word : tweet.split(" ")) {
      collector.emit(new Values(word));
    }
  }

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

```java
class WordCountBolt 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);
    count = (count == null) ? 1 : count + 1;
    counts.put(word, count);
    collector.emit(new Values(word, count));
  }

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

Streaming word count




```
TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("tweet_spout", new RandomTweetSpout(), 5);
builder.setBolt("parse_bolt", new ParseTweetBolt(), 8)
    .shuffleGrouping("tweet_spout")
    .setNumTasks(2);
builder.setBolt("count_bolt", new WordCountBolt(), 12)
    .fieldsGrouping("parse_bolt", new Fields("word"));

Config config = new Config();
config.setNumWorkers(3);
StormSubmitter.submitTopology("demo", config, builder.createTopology());
```

tweet spout

```
class RandomTweetSpout extends BaseRichSpout {
  SpoutOutputCollector collector;
  Random rand;
  String[] tweets = new String[] {
    "@jkrums:There's a plane in the Hudson. I'm on the ferry to pick up people. Crazy",
    "@barackobama: Four more years. pic.twitter.com/bAJE6Vom",
    ...
  };

  ....

  @Override
  public void nextTuple() {
    Utils.sleep(100);
    String tweet = tweets[rand.nextInt(tweets.length)];
    collector.emit(new Values(tweet));
  }
}
```

Contents

What is Spark Streaming?

- Framework for large scale stream processing
  - Micro-batch processing model
    - Can achieve second scale latencies
  - Scales to 100s of nodes
  - Integrates with Spark's batch and interactive processing
  - Provides a simple batch-like API for implementing complex algorithm
  - Can absorb live data streams from Kafka, Flume, ZeroMQ, etc.

Discretized Streams

- Reuse Spark Programming model
  - Transformations on RDDs
- RDDs are created combining all the messages in a defined time interval
- A new RDD is processed at each slot
- Spark code for creating one:
  - `val streamFromMQTT = MQTTUtils.createStream(ssc, brokerUrl, topic, StorageLevel.MEMORY_ONLY_SER_2)`

Case study: Conviva, Inc.

- Real-time monitoring of online video metadata
  - HBO, ESPN, ABC, SyFy, …
- Two processing stacks

Custom-built distributed stream processing system

- 1000s complex metrics on millions of video sessions
- Requires many dozens of nodes for processing

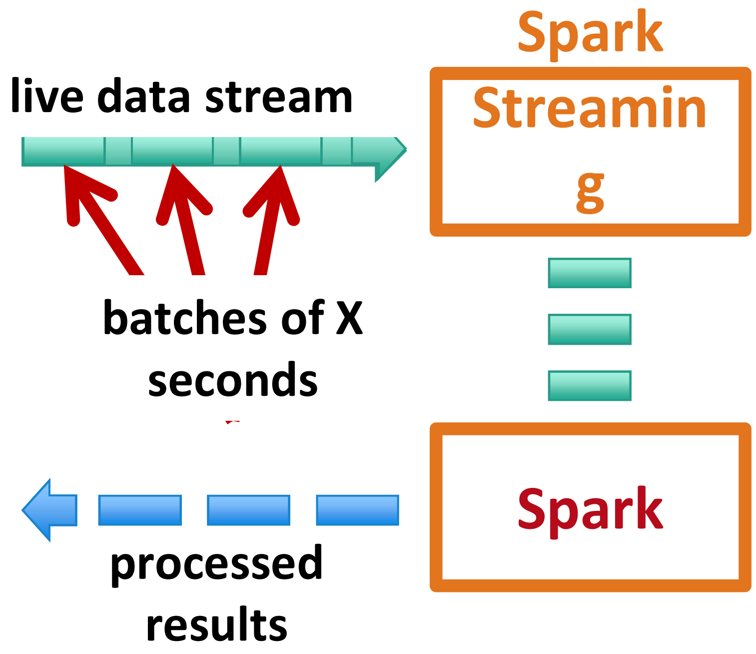
Hadoop backend for offline analysis

- Generating daily and monthly reports
- Similar computation as the streaming system

Discretized Stream Processing

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

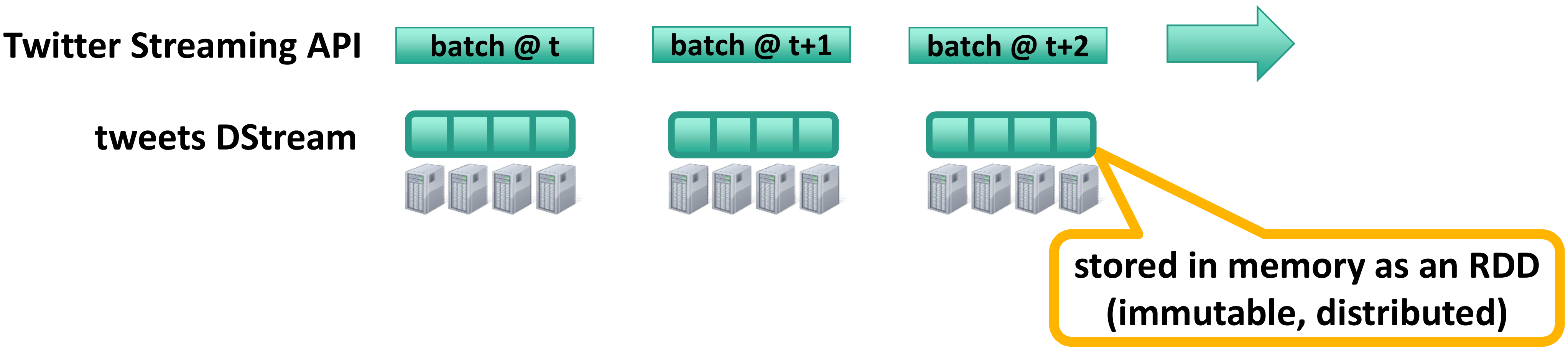
- Chop up the live stream into batches of X seconds
- Spark treats each batch of data as RDDs and processes them using RDD operations
- Finally, the processed results of the RDD operations are returned in batches



Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
```

DStream: a sequence of RDD representing a stream of data

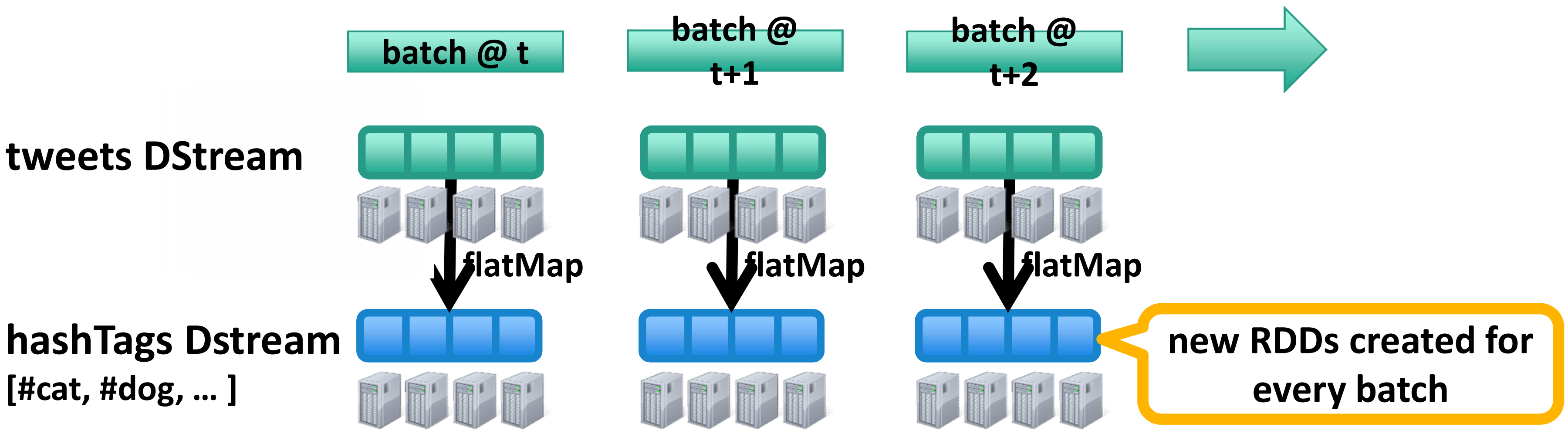


Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
```

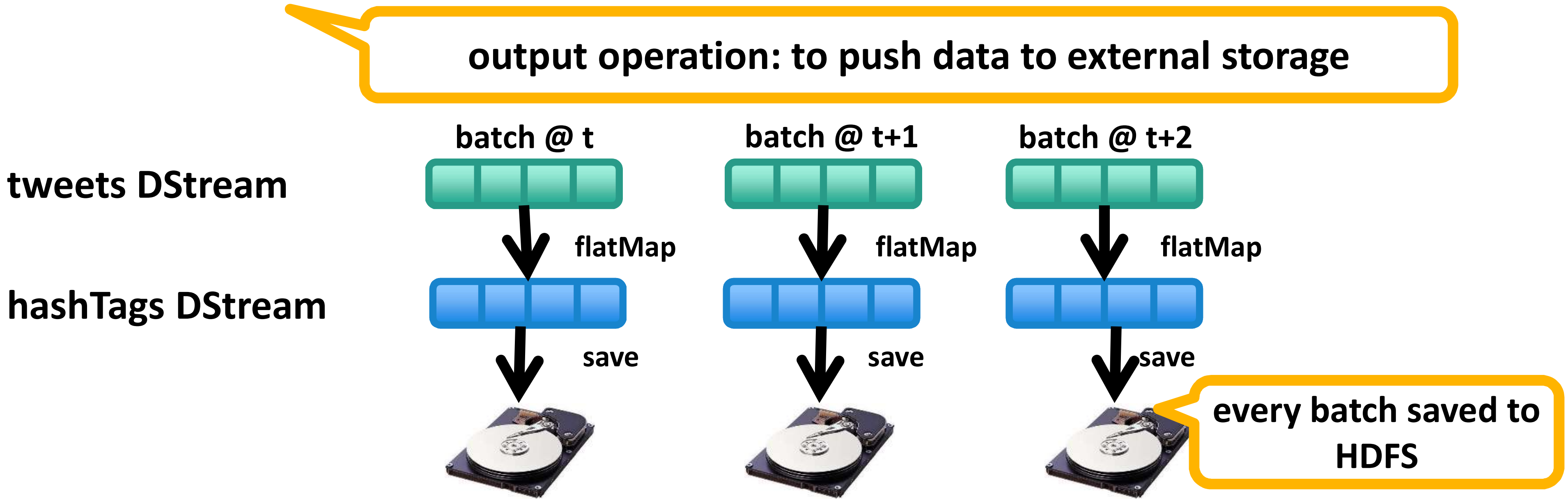
new DStream

transformation: modify data in one Dstream to create another DStream



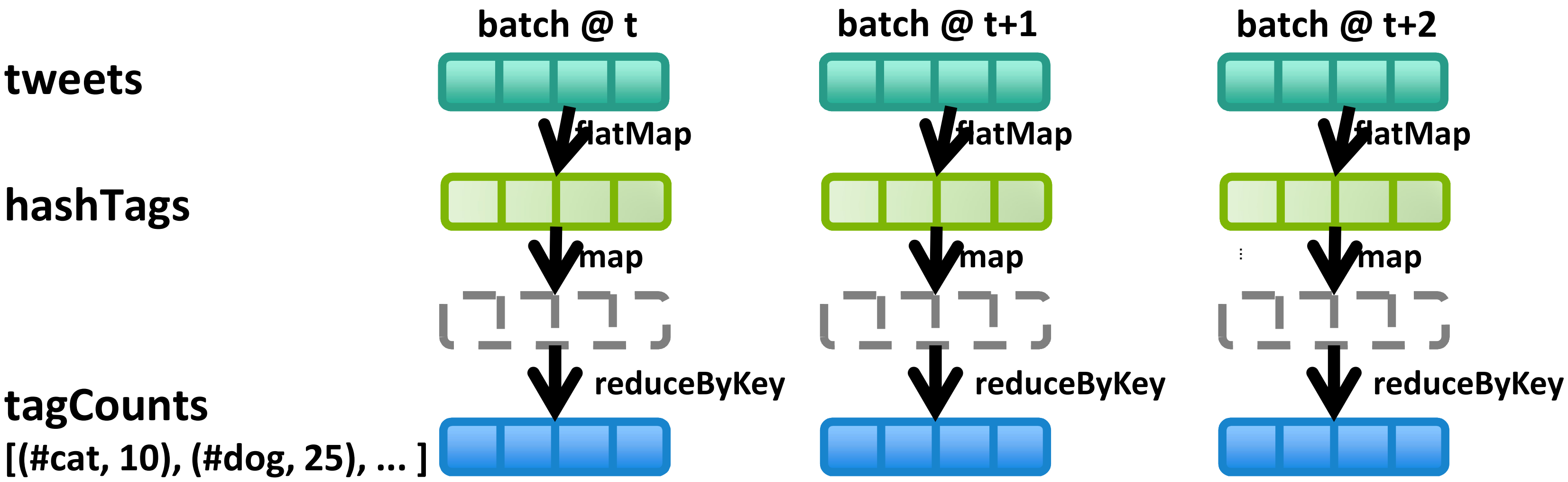
Example 1 – Get hashtags from Twitter

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```



Example 2 – Count the hashtags

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.countByValue()
```



Example 3 – Count the hashtags over last 10 mins

```
val tweets = ssc.twitterStream(<Twitter username>, <Twitter password>)
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```

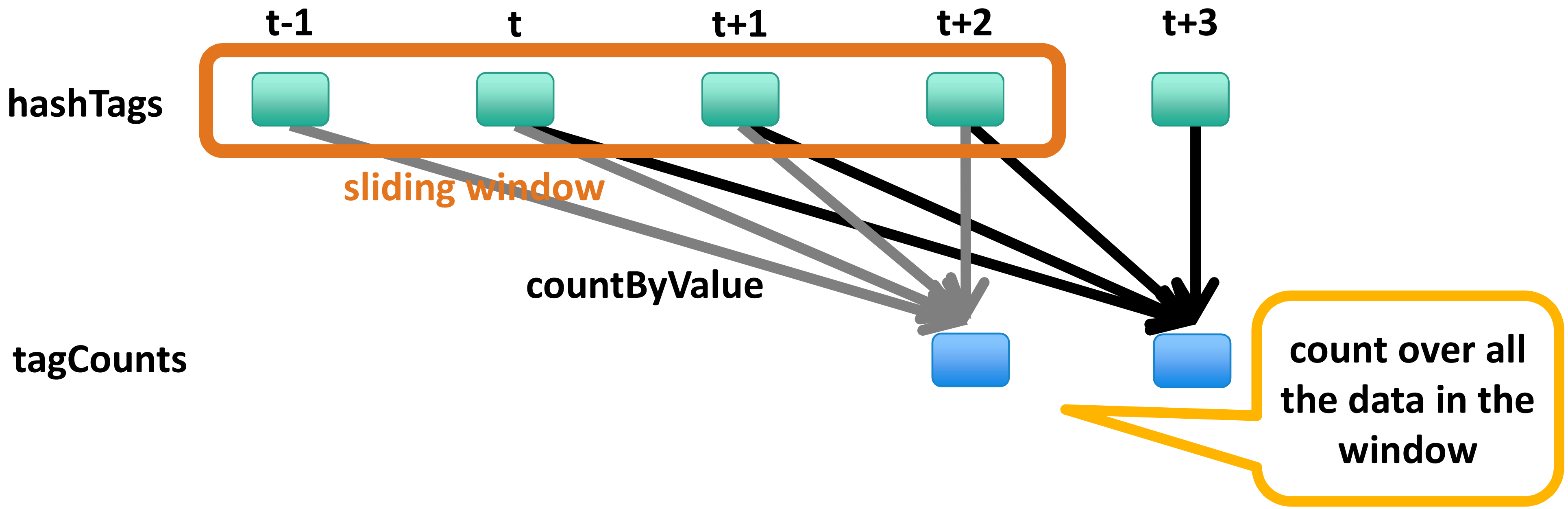
sliding window operation

window length

sliding interval

Example 3 – Counting the hashtags over last 10 mins

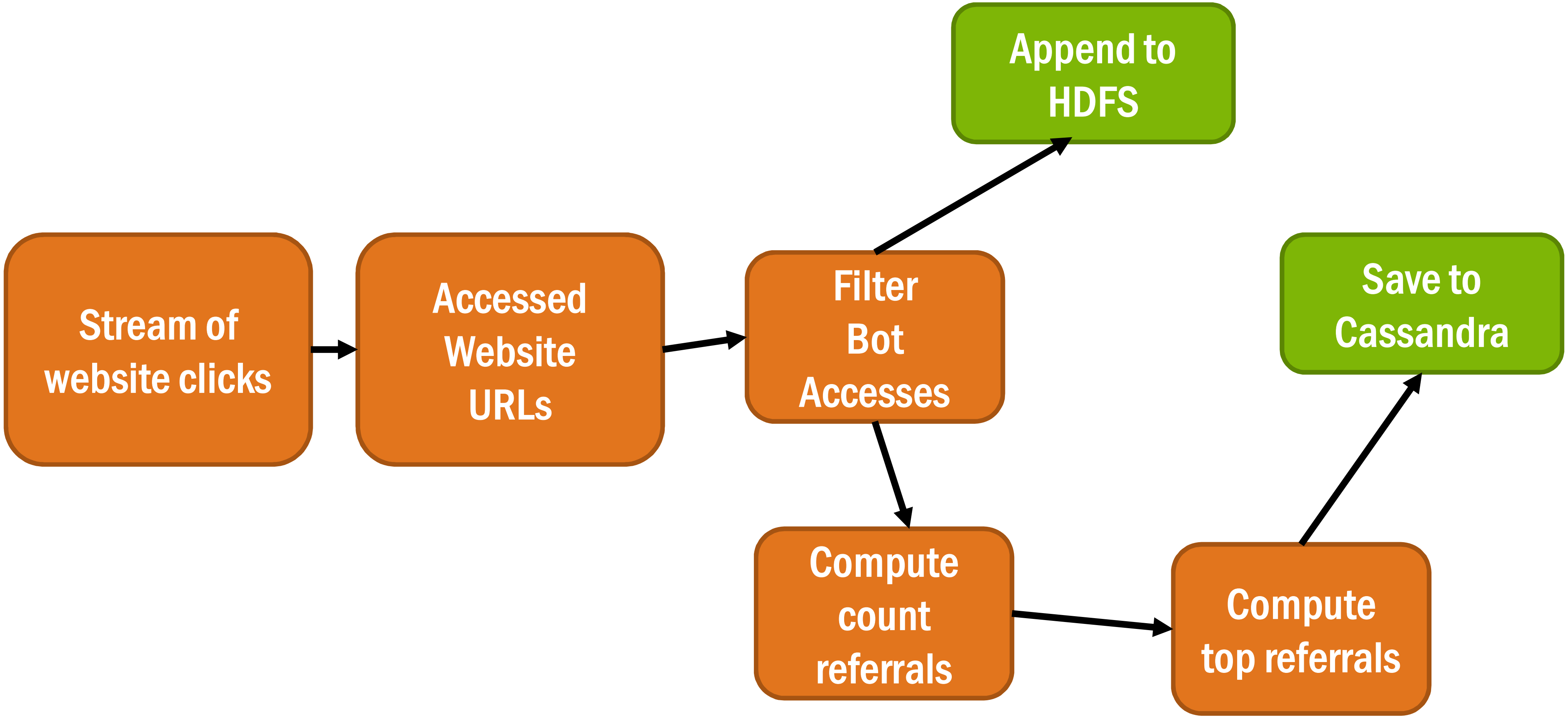
```
val tagCounts = hashTags.window(Minutes(10), Seconds(1)).countByValue()
```



D-Stream Streaming context

- Spark streaming flows are configured by creating a StreamingContext, configuring what transformations flow will be done, and the invoke the start method
  - `val ssc = new StreamingContext(sparkUrl, "Tutorial", Seconds(1), sparkHome, Seq(jarFile))`
- There must be some action collecting in some way the results of a temporal RDD

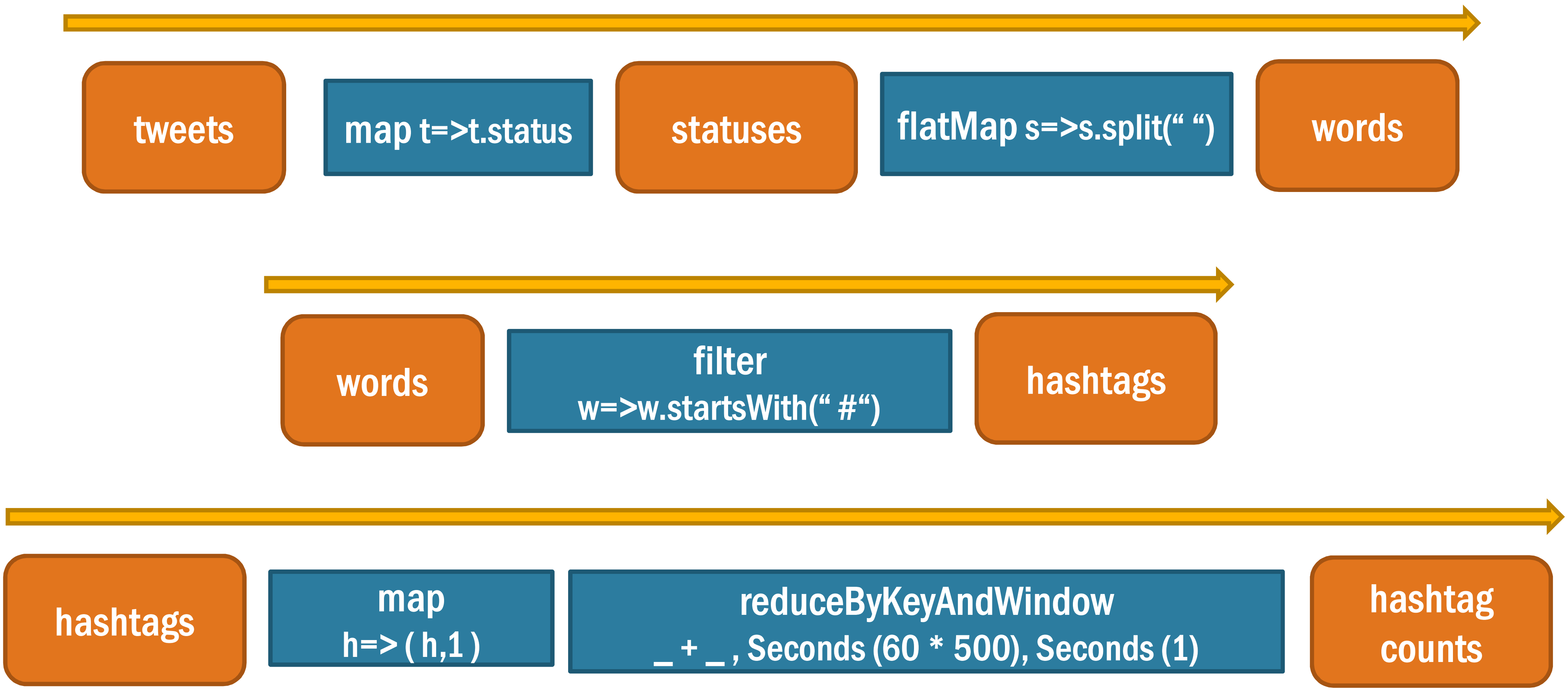
Sample topology: Website click analysis



Sliding window operations in Spark D-Stream

- D-Stream provides direct API support for specifying sliding windows
- Two parameters:
  - Size of the window (in seconds)
  - Frequency of computations (in seconds)
- E.g. process the maximum temperature over the last 60 seconds, every 5 seconds.
  -
```
reduceByWindowAndKey((a,b)=>math.max(a,b),
                     Seconds(60, Seconds(5) )
```

Sample Twitter processing stream



Sample Twitter Processing Stream

```
val ssc = new StreamingContext(sparkUrl,
"Tutorial", Seconds(1), sparkHome, Seq(jarFile))
val tweets = ssc.twitterStream()
val statuses = tweets.map(status => status.getText())
val words = statuses.flatMap(
               status => status.split(" "))
val hashtags = words.filter(
               word => word.startsWith("#"))
val hashtagCounts = hashtags.map(tag => (tag, 1)).
     reduceByKeyAndWindow(
          _ + _, Seconds(60 * 5), Seconds(1))
ssc.checkpoint(checkpointDir)
ssc.start();
```