Social Media for Emergency Management

Vimala Nunavath

vimala.nunavath@uia.no

Agenda

- Recap
- Social media
- Social media use in disasters
- What is streaming?
- Why Spark streaming?
- Spark streaming components
- Social network graphs
- Example: Real-time twitter data stream processing with Apache Spark
- Presentations by you!!

Recap

- Central Themes:
  - Introduction to Emergency management (Session 1)
  - Introduction to Big data (Session 2)
  - Big Data sources for EM (Session 3)
  - Sensors/IOT for EM (Session 4)
  - **Social media for EM (Session 5)**
  - **Natural language processing (NLP) for EM and visualization/Dashboards (Session 5)**
  - **Machine learning for EM (Session 5)**
- Essay and student programming project

Social media

Social media in general

- The term “social media” refers to Internet-based applications that enable people to communicate and share resources and information.

- e.g., 

- Huge volumes of data are generated every minute, a phenomenon commonly referred to by researchers as **big data**, **information overload** or **data deluge**.

- Evolving phenomenon
- New technologies have enabled people to interact and share information through media.

Social media in disasters

- Social media (SM) plays a vital role in disaster response and recovery by providing response information before, during and after disasters.
- Social media are changing the way people communicate not only in their day-to-day lives, but also during disasters that threaten public health.
- Engaging with and using emerging social media may well place the emergency-management community, including medical and public health professionals, in a better position to respond to disasters.
- The effectiveness of public emergency system relies on routine attention to preparedness, agility in responding to daily stresses and catastrophes, and the resilience that promotes rapid recovery. Social media can enhance each of these component efforts.

Social media in disasters

- The use of social media for emergencies and disasters on an organizational level may be conceived of as two broad categories:

  - To disseminate information and receive user feedback via incoming messages, wall posts, and polls.

  - An emergency management tool. Systematic usage might include:

    1) using the medium to conduct emergency communications and issue warnings;

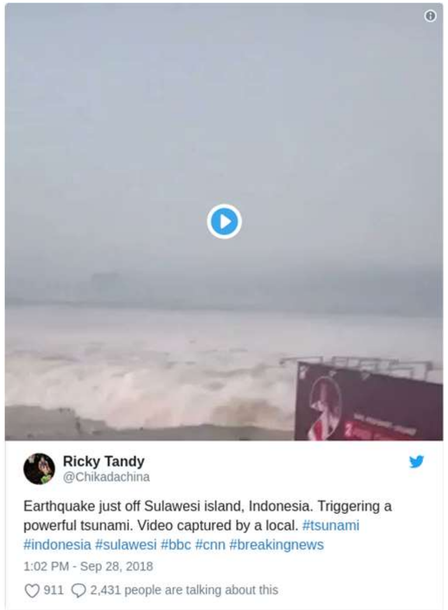
    2) using social media to receive victim requests for assistance;

    3) monitoring user activities and postings to establish situational awareness; and

    4) using uploaded images to create damage estimates, among others.

Social media in disasters

- For instance:
  - 2018 Indonesia Earthquake
  - 2012 Hurricane Sandy
  - 2012 Utøya bombing



SM for Situational Awareness

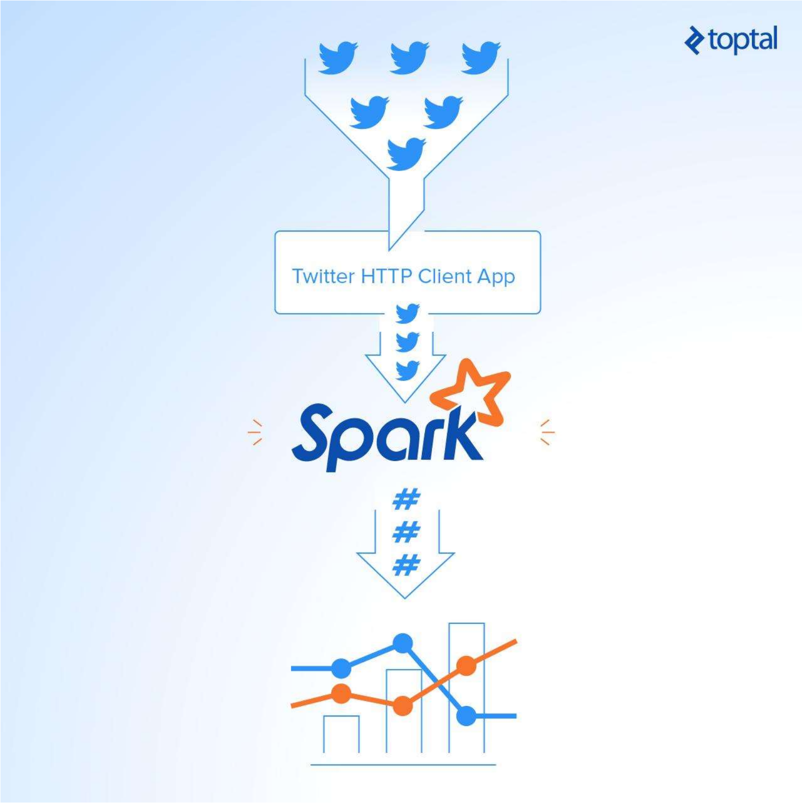
- Social media could be used to alert emergency managers and officials to certain situations by monitoring the flow of information from different sources during an incident.
- Monitoring information flows could help establish situational awareness.
- Situational awareness: the ability to identify, process, and comprehend critical elements of an incident or situation.
- Obtaining real-time information as an incident unfolds can help officials determine where people are located, assess victim needs, and alert citizens and first responders to changing conditions and new threats.

Challenges with Social media data

- Providing inaccurate and false information
  - complicate situational awareness of an incident
  - jeopardize the safety of first responders and the community
- Malicious use of social media during disasters
- Technological limitations
- Privacy issues

Accessing Social Media data

Streaming Twitter data with Spark

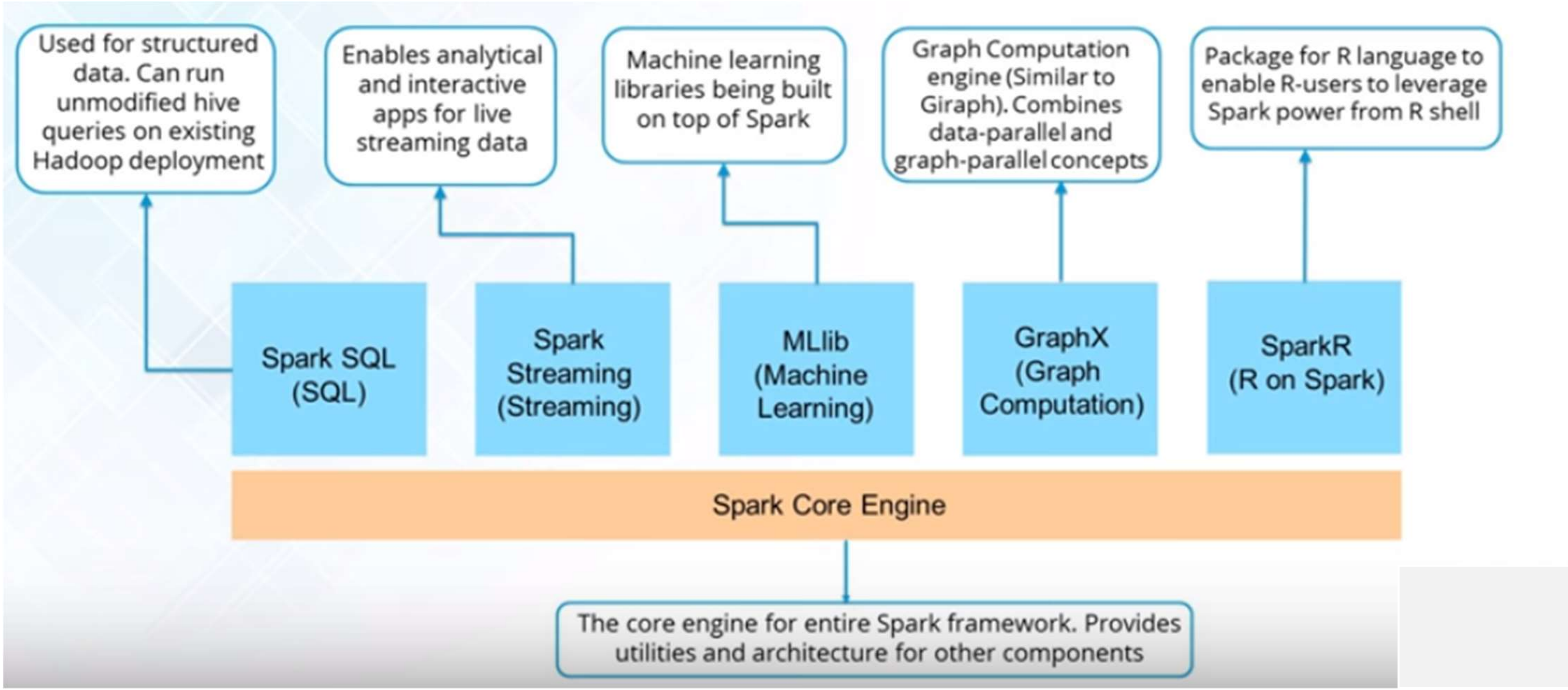


What is Streaming?

- **Data streaming** is a technique for transforming data so that it can be processed as a **steady** and **continuous** stream.

- Streaming technologies are becoming increasingly important with the growth of the internet.

Spark ecosystem



 Source: https://www.edureka.co/

Why Spark Streaming

Spark Streaming is used to stream real-time data from various sources like twitter, Facebook, and geographical systems and perform powerful analytics to help during disasters.

Spark streaming features

 Source: https://www.edureka.co/

What is Spark Streaming?

- It is an extension of the ***core Spark API*** that enables
  - Scalable, high-throughput, fault-tolerant ***stream processing of live data streams.***
  - Data can be ingested from many sources e.g., Kafka, Flume, Kinesis, or TCP sockets

Figure: Data from a variety of sources to various storage systems

  - It can be processed using complex algorithms expressed with high-level functions ***like map, reduce, join and window.***
  - processed data can be pushed out ***to filesystems, databases, and live dashboards.***

 Source: https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html

Spark streaming overview

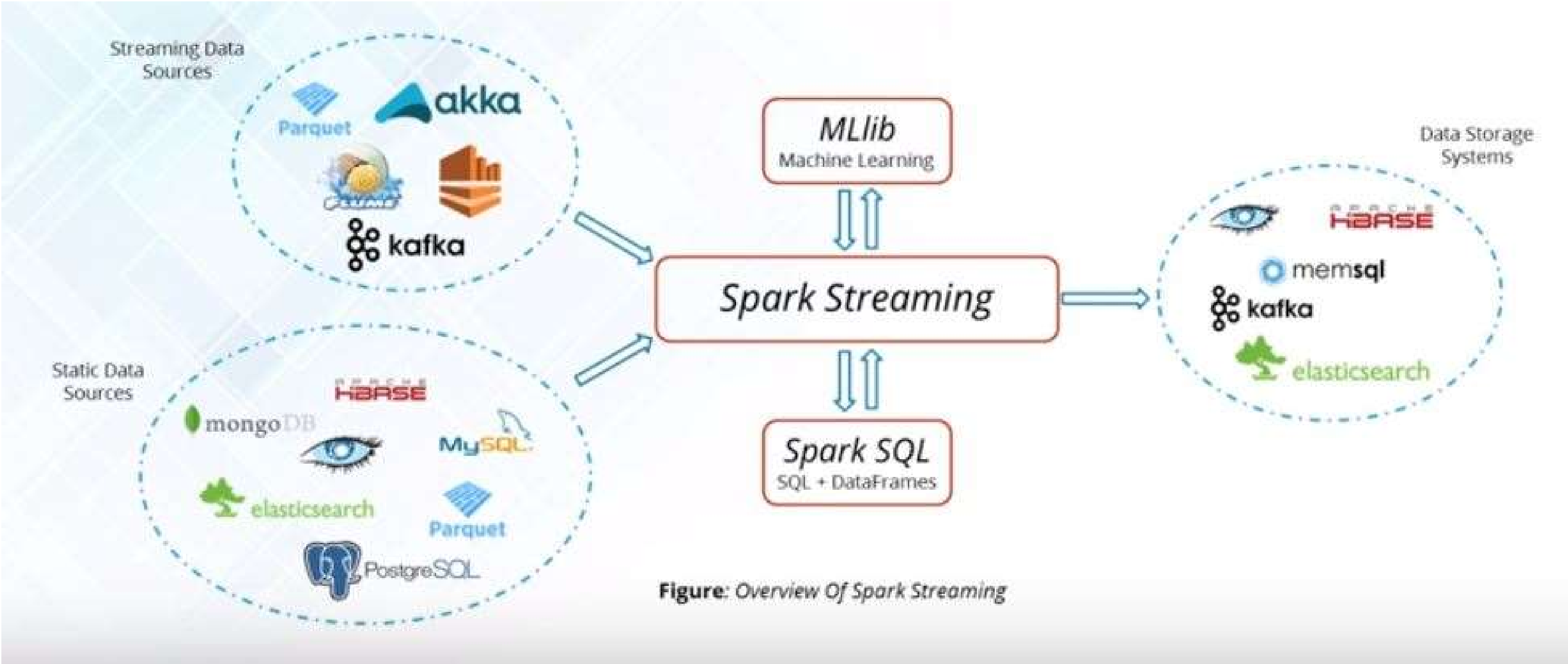


Figure: *Overview Of Spark Streaming*

Spark Streaming

- Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

- It provides a high-level abstraction called ***discretized stream* or *Dstream*.**
- We can write Spark Streaming programs in Scala, Java or Python

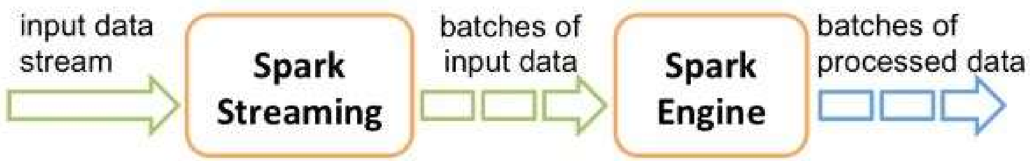
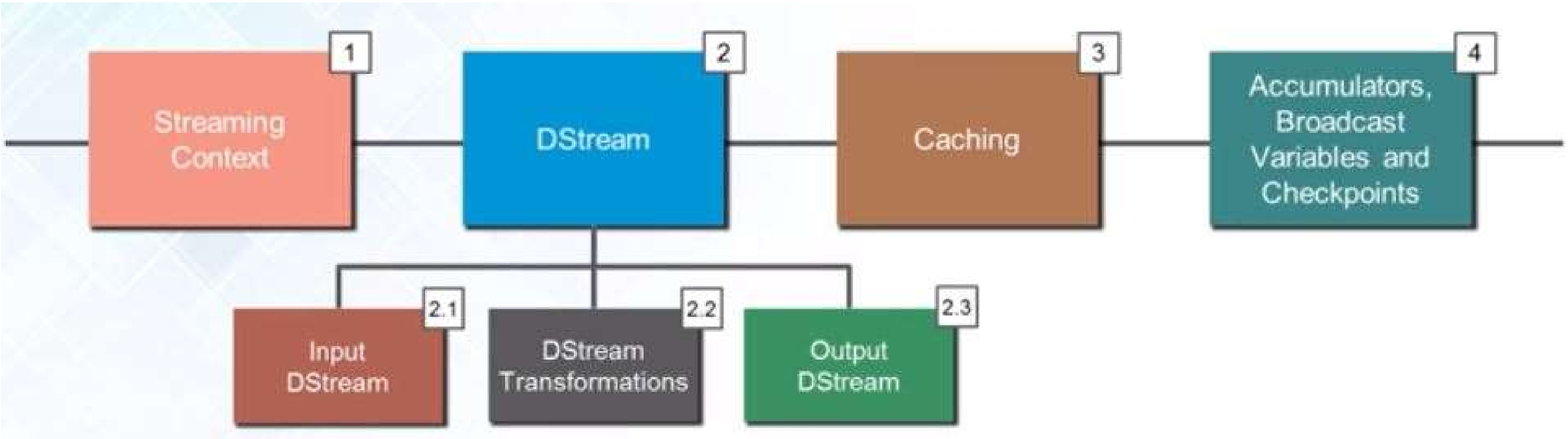


Figure: Incoming streams of data divided into batches

Source: https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html

Streaming fundamentals



 Source: https://www.edureka.co/

Streaming Context

Figure: *Spark Streaming Context*

- Consumes a stream of data in Spark.
- Registers an **InputDstream** to produce a **Reciever** object.
- It is the main entry point for Spark functionality.
- Spark provides a number of default implementations of sources like Twitter, Akka Actor, and ZeroMQ that are accessible from the context.

Figure: *Default Implementation Sources*

Streaming Context - Initialization

- A StreamingContext object can be created from a SparkContext object.
- A SparkContext represents the connection to a Sprak cluster and can be used to create RDDs, accumulators and broadcast variables on that cluster.

```
import org.apache.spark._
import org.apache.spark.streaming._
var ssc = new StreamingContext(sc,Seconds(1))
```

 Source: https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html

Dstream

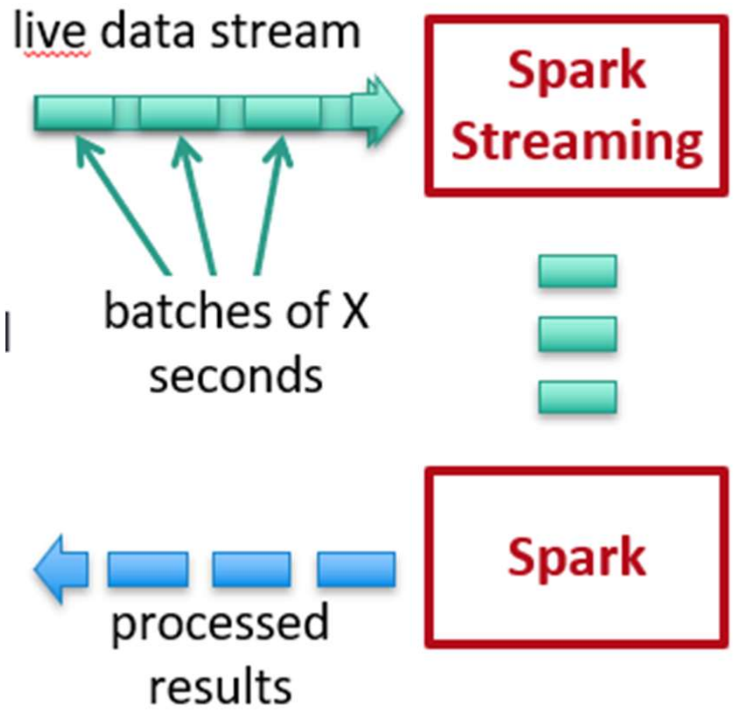
- Discretized stream (Dstream) *is the basic* abstraction provided by Spark Streaming.
- It represents a continuous stream of data.
- It is received from source or from a processed data stream generated by transforming the input stream.
- Internally, a DStream is represented by a continuous series of Resilient Distributed Datasets (RDDs). Each RDD contains data from a certain interval.

Figure: Input data streams divided into discrete chunks of data

Source: https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html

Discretized Stream Processing

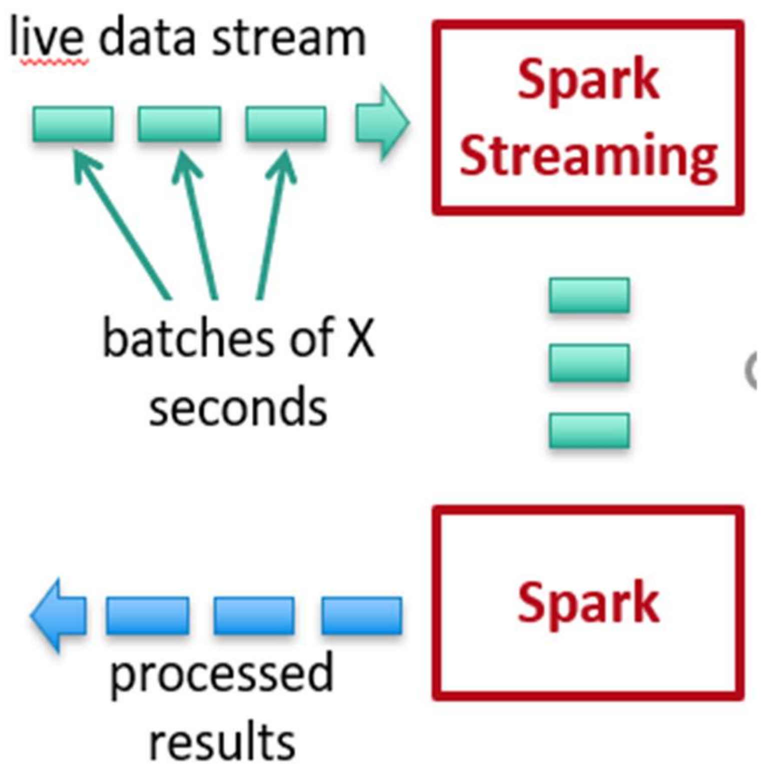
- Dstreams: Run a streaming computation as a series of very small, deterministic batch jobs
  - Chop up the live stream into batches of X seconds
  - Spark treats each batch of data as RDDs and processes them using RDD operations
  - Finally, the processed results of the RDD operations are returned in batches



 source: http://ampcamp.berkeley.edu/

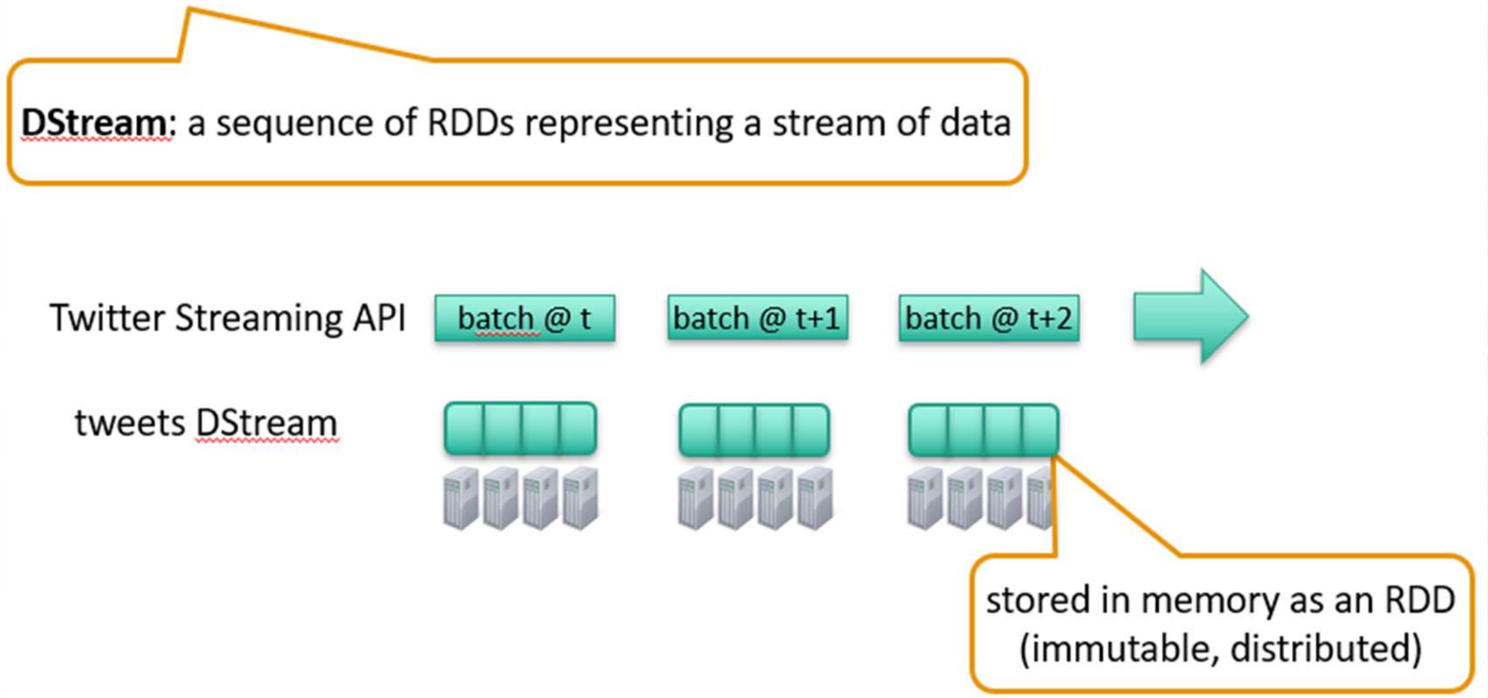
Discretized Stream Processing

- Dstreams: Run a streaming computation as a series of very small, deterministic batch jobs
  - Batch sizes as low as ½ second, latency of about 1 second
  - Potential for combining batch processing and streaming processing in the same system



Example – Get hashtags from Twitter

```
val tweets = ssc.twitterStream()
```



Dstream operation

- Any operation applied on a DStream translates to operations on the underlying RDDs.
- For example, in the example of converting a stream of lines to words, ***the flatMap operation*** is applied on each RDD in the lines DStream to generate the RDDs of the words Dstream.

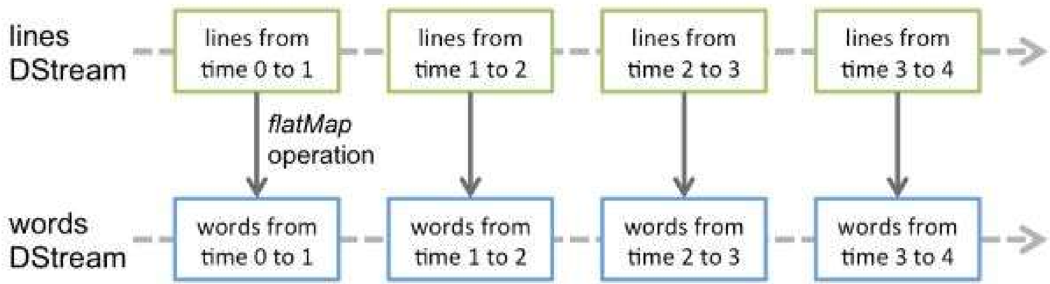
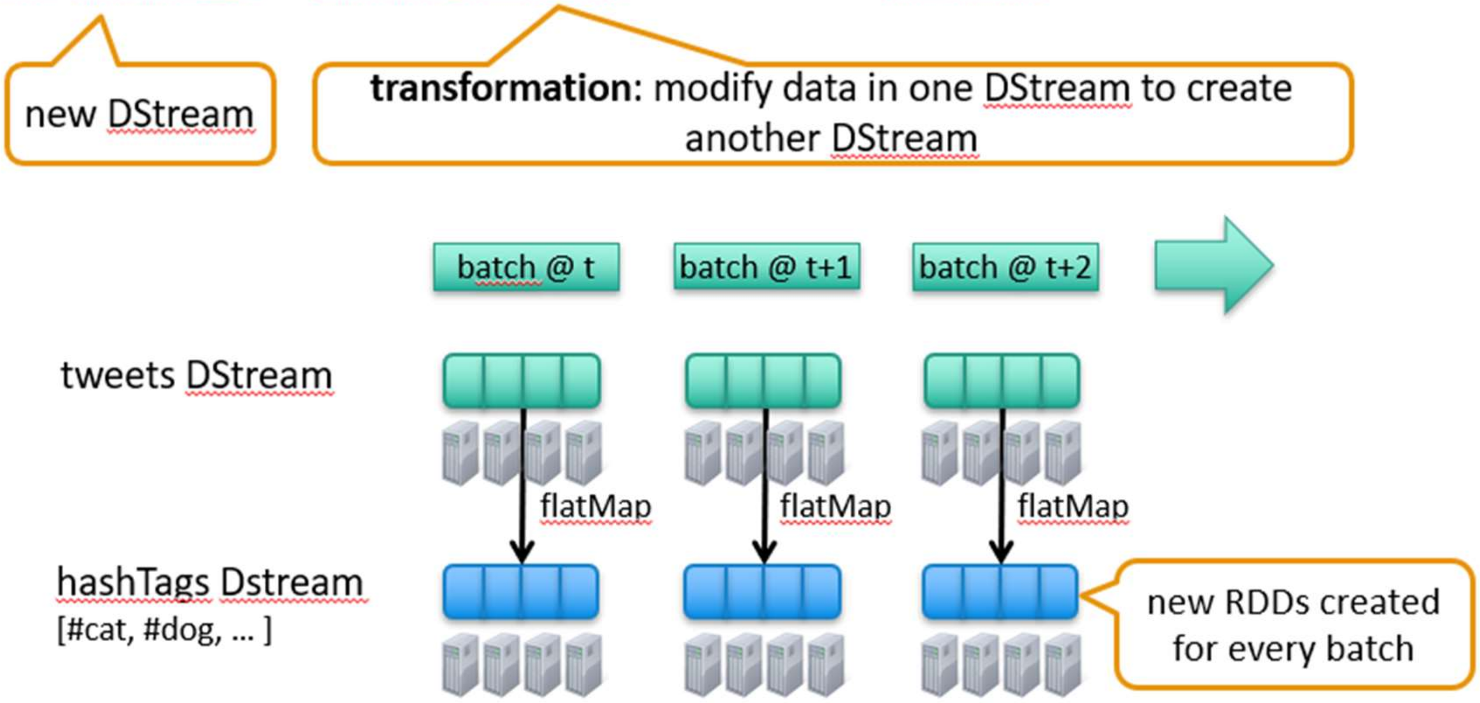


Figure: Extracting words from an Inputstream

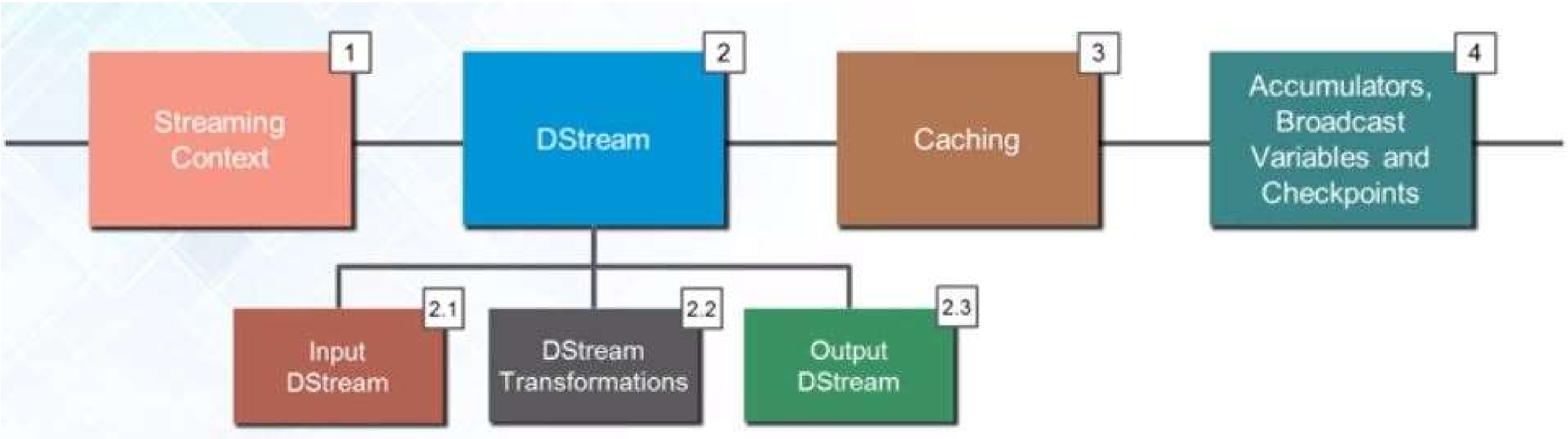
Example – Get hashtags from Twitter

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap (status => getTags(status))
```



 source: http://ampcamp.berkeley.edu/

Streaming fundamentals

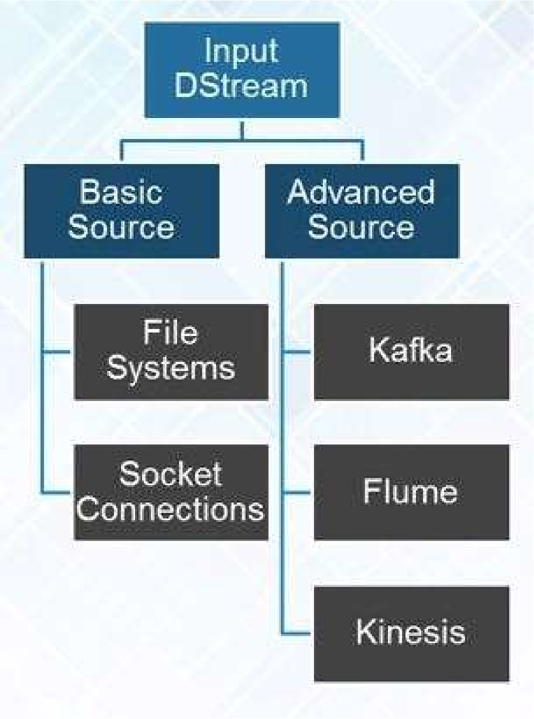


Input DStreams

- Input DStreams are DStreams representing the stream of input data received from streaming sources.

Figure: Input datastream divided into discrete chunks of data

Input DStreams



- If we want to receive multiple streams of data in parallel in our streaming application, then we can create multiple input Dstreams.
- This will create multiple receivers which will simultaneously receive multiple data streams.

Receiver

- Every input DStream is associated with a Receiver object which receives the data from a source and stores it in Spark's memory for processing.

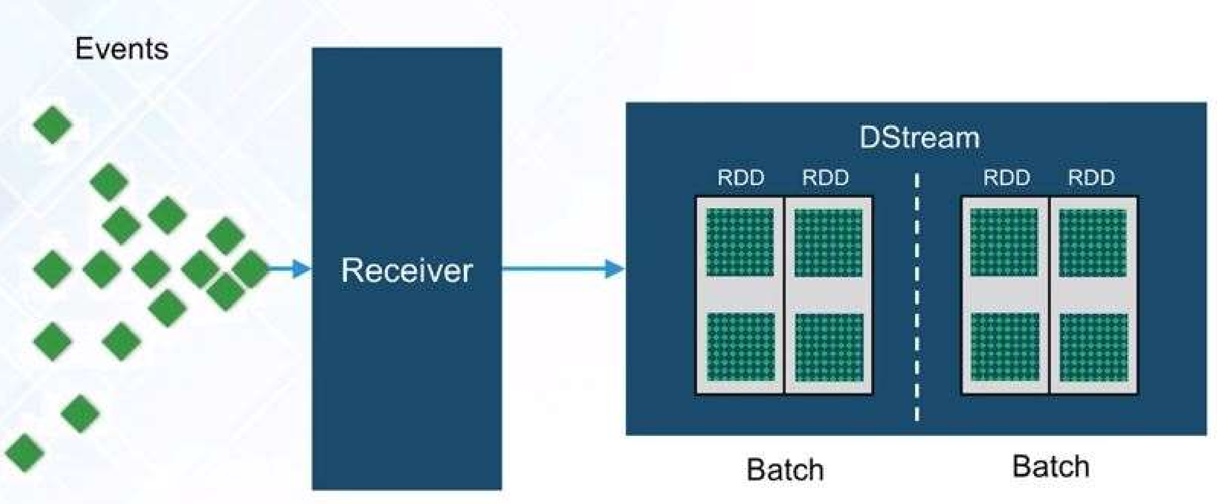


Figure: *The Receiver sends data onto the DStream where each Batch contains RDDs*

Transformations on Dstreams

- Transformations allow the data from the inputDstream to be modified similar to RDDs. Dstreams support many of the transfromations available on normal Spark RDDs.

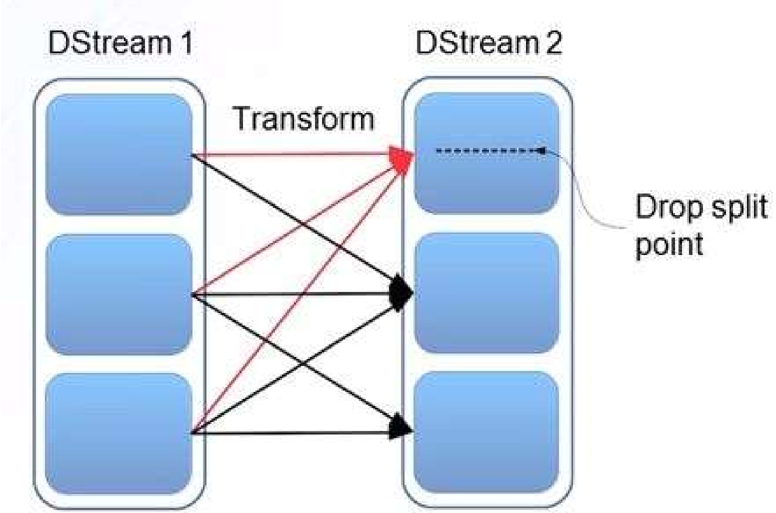


Figure: *DStream Transformations*

Transformations on Dstreams

- Map(func):

  - It returns a new Dstream by passing each element of the source Dstream through a function func.

Figure: *Input DStream being converted through map(func)*

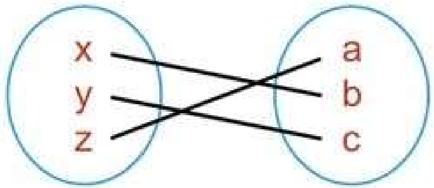


Figure: *Map Function*

Transformations on Dstreams

- flatMap(func):

  - It is similar to map(func), but each input item can be mapped to 0 or more output items and returns a new Dstream by passing each source element through a function func.

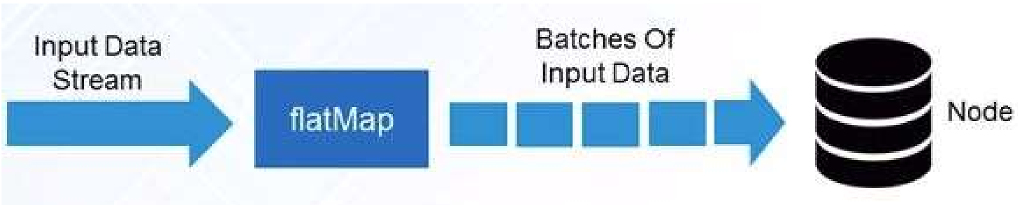


Figure: *Input DStream being converted through flatMap(func)*

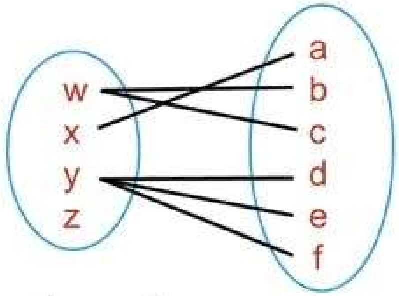


Figure: *flatMap Function*

Transformations on Dstreams

- Filter(func):
  - It returns a new Dstream by selecting only the records of the source Dstream on which func returns true.

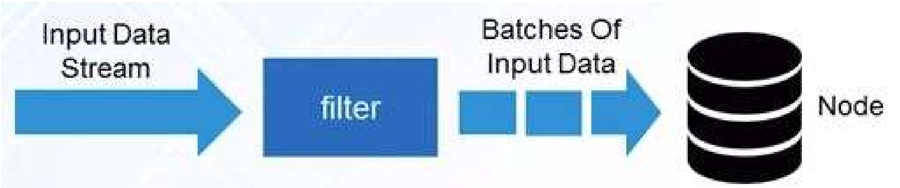


Figure: *Input DStream being converted through filter(func)*

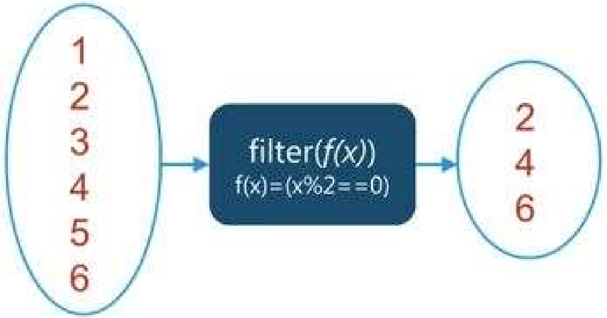


Figure: *Filter Function For Even Numbers*

Transformations on Dstreams

- <u>Reduce(func):</u>

  - It returns a new Dstream of single-element RDDs by aggregating the elements in each RDD of the source Dstream using a function func.

Figure: *Input DStream being converted through reduce(func)*

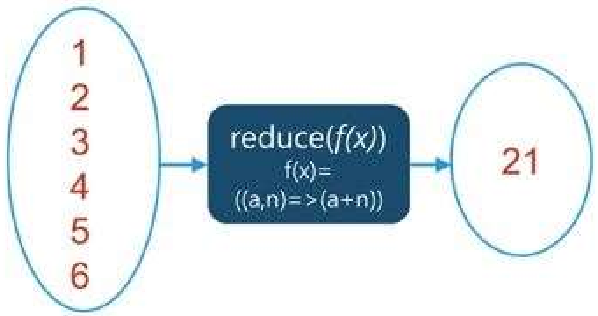


Figure: *Reduce Function To Get Cumulative Sum*

Transformations on Dstreams

- groupBy(func):
  - It returns the new RDD which basically is made up with a key and corresponding list of items of that group.

Figure: *Input DStream being converted through groupBy(func)*

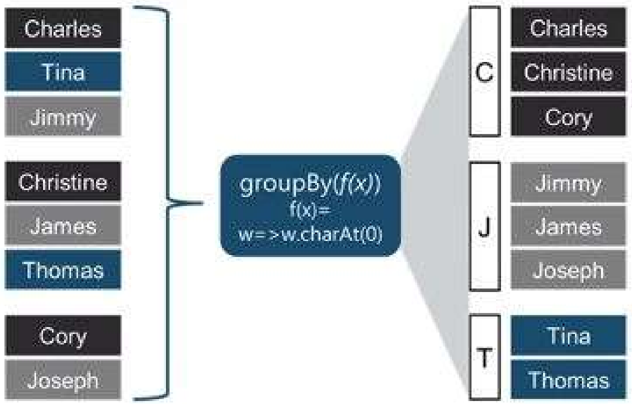


Figure: *Grouping By First Letters*

Dstream Window Operations

- Spark Streaming also provides windowed computations, which allow you to apply transformations over a sliding window of data.

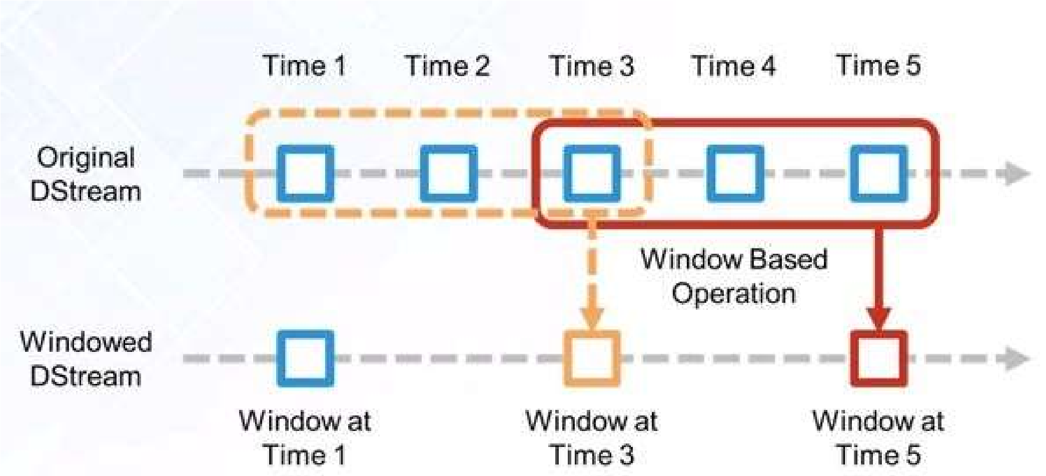
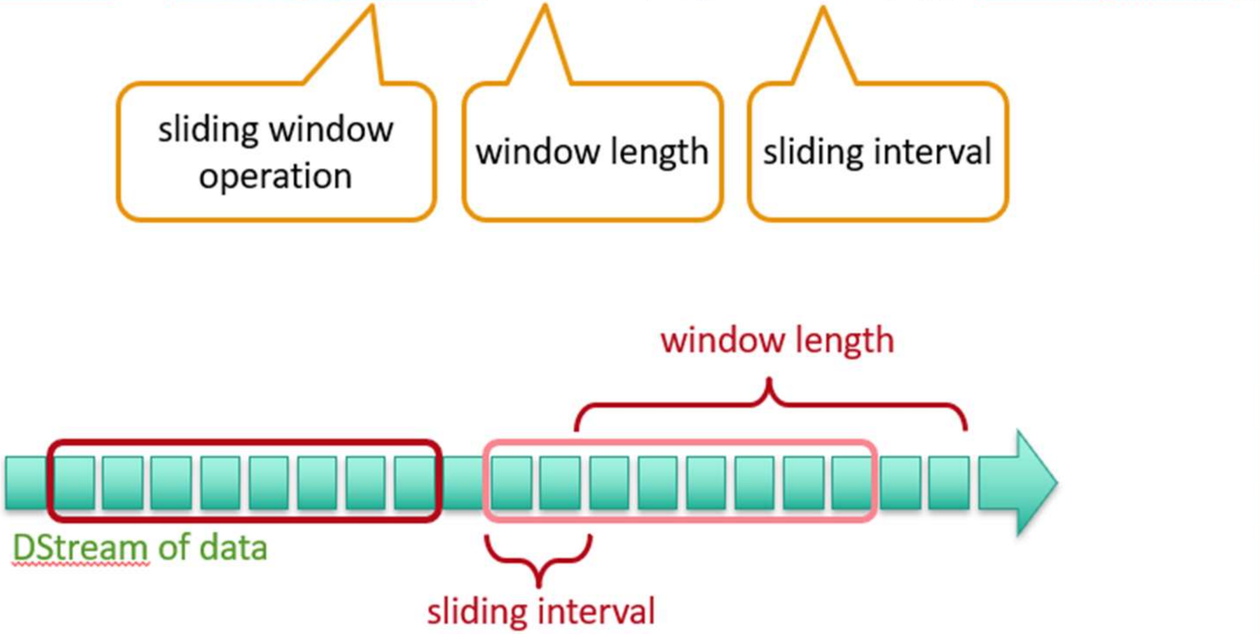


Figure: *DStream Window Transformation*

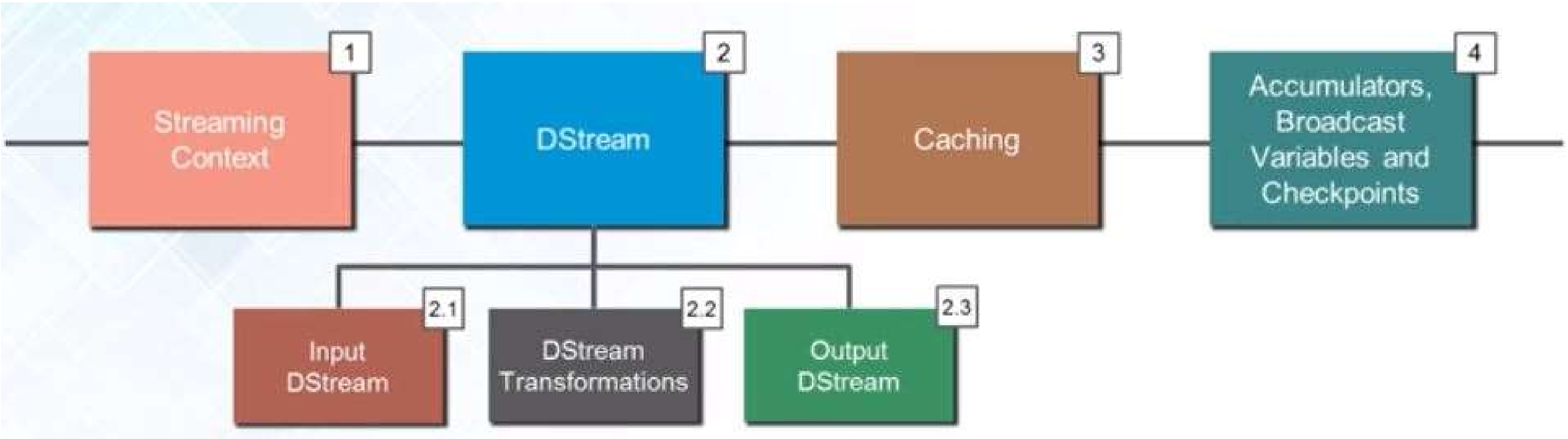
Window-based Transformations

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap (status => getTags(status))
val tagCounts = hashTags.window(Minutes(1), Seconds(5)).countByValue()
```



source: http://ampcamp.berkeley.edu/

Streaming fundamentals



Output Operations on DStreams

- It allows DStream's data to be pushed out to external systems like databases or file systems.
- Output operations trigger the actual execution of all the DStream transformations.

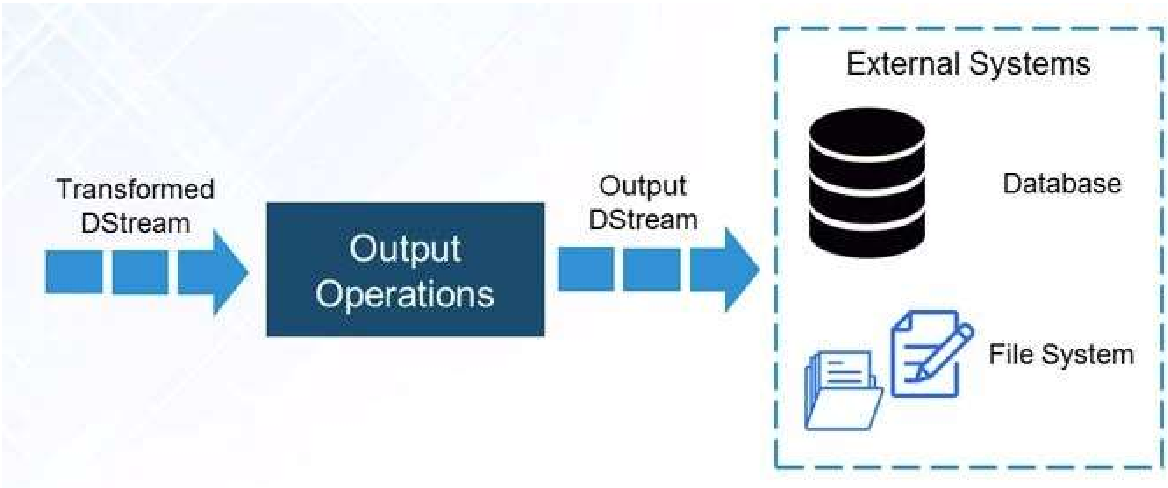
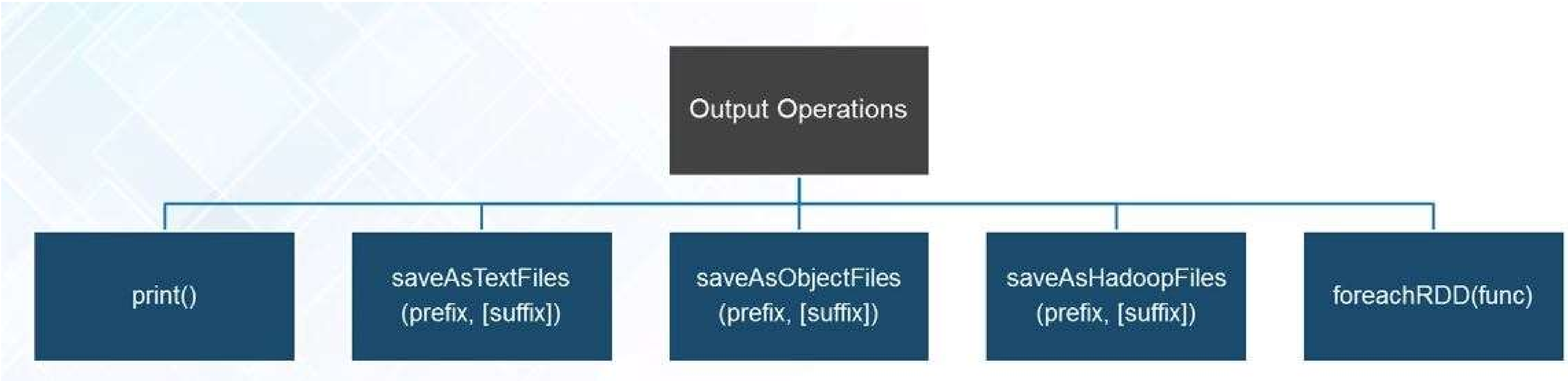


Figure: *Output Operations on DStreams*

Output Operations on DStreams

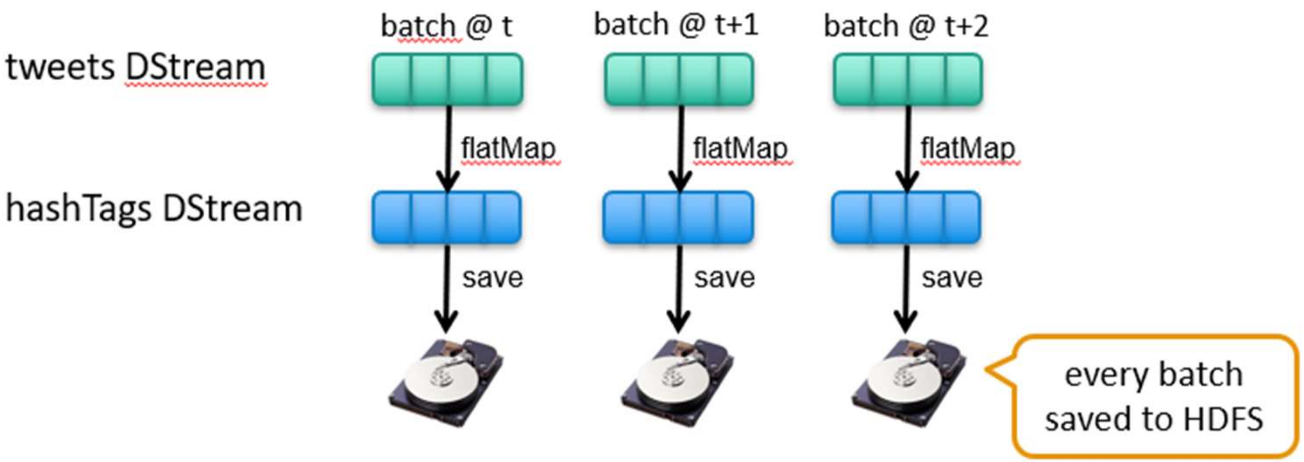
- Currently, the following output operations are defined:



Example – Get hashtags from Twitter

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```

output operation: to push data to external storage



Design Patterns for using foreachRDD

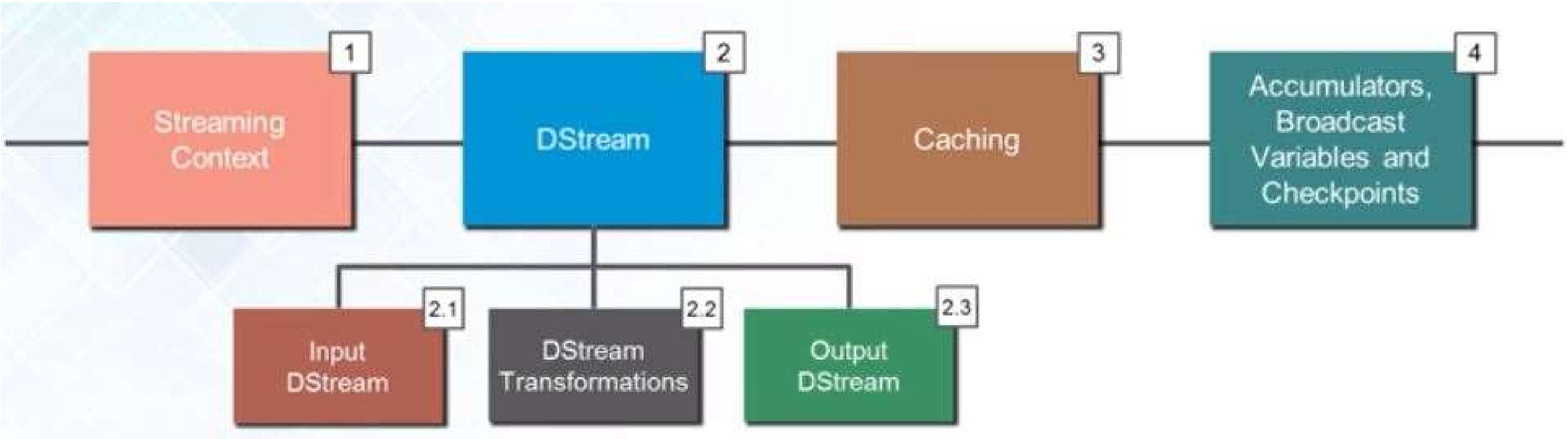
- dstream.foreachRDD is a powerful primitive that allows data to be sent out to external systems.
- The lazy evaluation achieves the most efficient transfer of data.

```
dstream.foreachRDD { rdd =>
 rdd.foreachPartition { partitionOfRecords =>

 // ConnectionPool is a static, lazily initialized pool of connections
 val connection = ConnectionPool.getConnection()
 partitionOfRecords.foreach(record => connection.send(record))

 // Return to the pool for future reuse
 ConnectionPool.returnConnection(connection)
 }
}
```

Streaming fundamentals



 Source: https://www.edureka.co/

Caching and persistent

- Dstreams allow developers to cache/persist the stream's data in memory. This is useful if the data in the Dstream will be computed multiple times.
- This can be done using the persist() method on a Dstream.
- For input streams that receive data over the network (such as Kafka, Flume, sockets, etc), the default persistence level is set to replicate the data to two nodes for fault-tolerance.

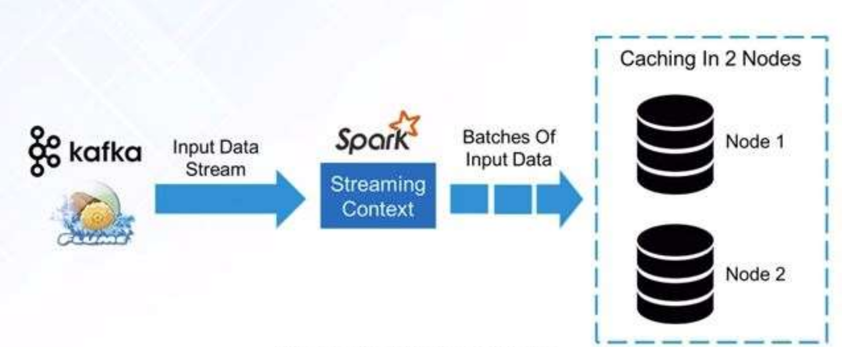
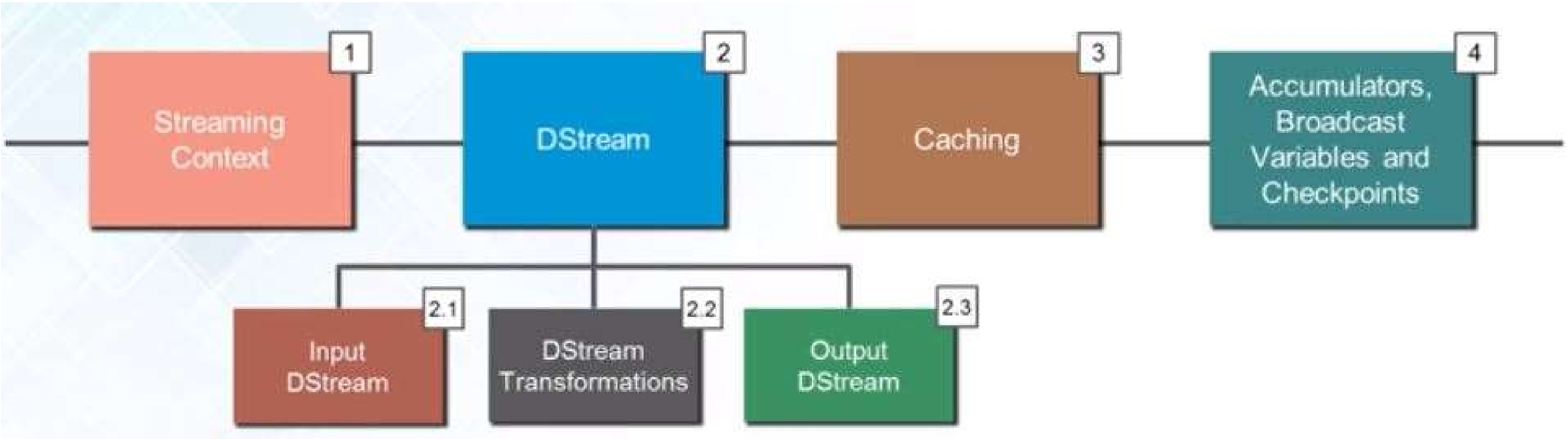


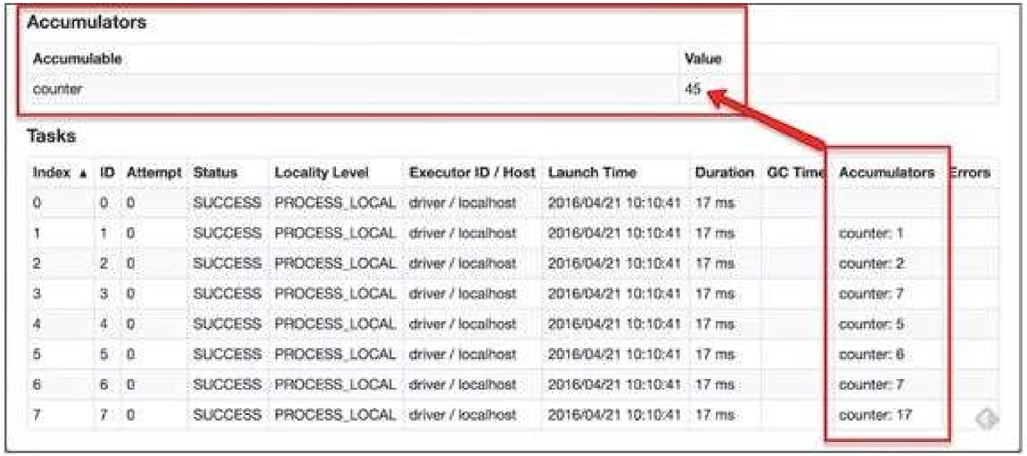
Figure: *Caching Into 2 Nodes*

Streaming fundamentals



Accumulators/Broadcast/Variables/Checkpoints

- Accumulators are variables that are only added through an associative and commutative operation.
- They are used to implement counters or sums.
- Tracking accumulators in the UI can be useful for understanding the progress of running stages.
- Spark natively supports numeric accumulators. We can create named or unnamed accumulators.



Accumulators

Accumulable	Value
counter	45

Tasks

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Accumulators	Errors
0	0	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms			
1	1	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 1	
2	2	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 2	
3	3	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 7	
4	4	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 5	
5	5	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 6	
6	6	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 7	
7	7	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2016/04/21 10:10:41	17 ms		counter: 17	

Figure: *Accumulators In Spark Streaming*

Accumulators/Broadcast/Variables/Checkpoints

- Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks.
- They can be used to give every node a copy of a large input dataset in an efficient manner.
- Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication post.

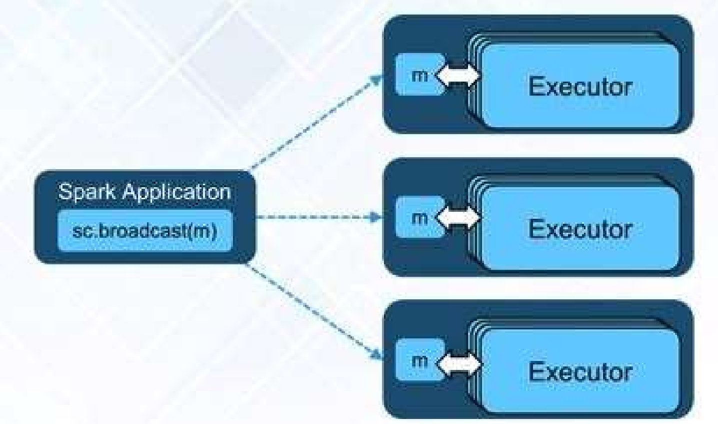


Figure: *Broadcasting A Value To Executors*

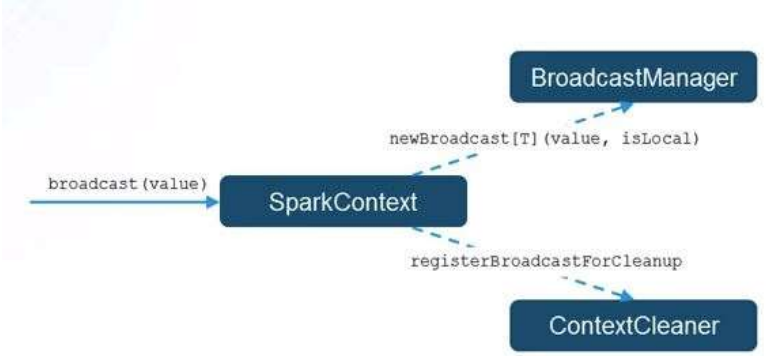


Figure: *SparkContext and Broadcasting*

Accumulators/Broadcast/Variables/Checkpoints

- Checkpoints are similar to checkpoints in gaming. They make it run 24/7 and make it resilient to failures unrelated to the application logic.

Use Case- Twitter analysis

Apache Spark

```
import org.apache.spark.SparkConf
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.Seconds
import twitter4j.conf.ConfigurationBuilder
import twitter4j.auth.OAuthAuthorization
import twitter4j.Status
import org.apache.spark.streaming.twitter.TwitterUtils
object TwitterData {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: TwitterData <ConsumerKey><ConsumerSecret><accessToken><access
        "[<filters>]")
      System.exit(1)
    }
    val appName = "TwitterData"
    val conf = new SparkConf()
    conf.setAppName(appName).setMaster("local[3]")
    val ssc = new StreamingContext(conf, Seconds(5))
    val Array(consumerKey, consumerSecret, accessToken, accessTokenSecret) = args.take(4)
    val filters = args.takeRight(args.length - 4)
    val cb = new ConfigurationBuilder
    cb.setDebugEnabled(true).setOAuthConsumerKey(consumerKey)
      .setOAuthConsumerSecret(consumerSecret)
      .setOAuthAccessToken(accessToken)
      .setOAuthAccessTokenSecret(accessTokenSecret)
    val auth = new OAuthAuthorization(cb.build)
    val tweets = TwitterUtils.createStream(ssc, Some(auth))
    val englishTweets = tweets.filter(_.getLang() == "en")
    englishTweets .saveAsTextFiles("tweets", "json")
    ssc.start()
    ssc.awaitTermination()
  }
}
```

Setting up the Spark streaming context

- We need to set the Spark streaming context as follows:

```
val ssc = new StreamingContext(conf, Seconds(5))
```

Authenticate twitter user

- val cb = new ConfigurationBuilder
- cb.setDebugEnabled(true).setOAuthConsumerKey(consumerKey)
- .setOAuthConsumerSecret(consumerSecret)
- .setOAuthAccessToken(accessToken)
- .setOAuthAccessTokenSecret(accessTokenSecret)

- Authentication:

  val auth = new OauthAuthorization(cb.build)

Starting the spark streaming

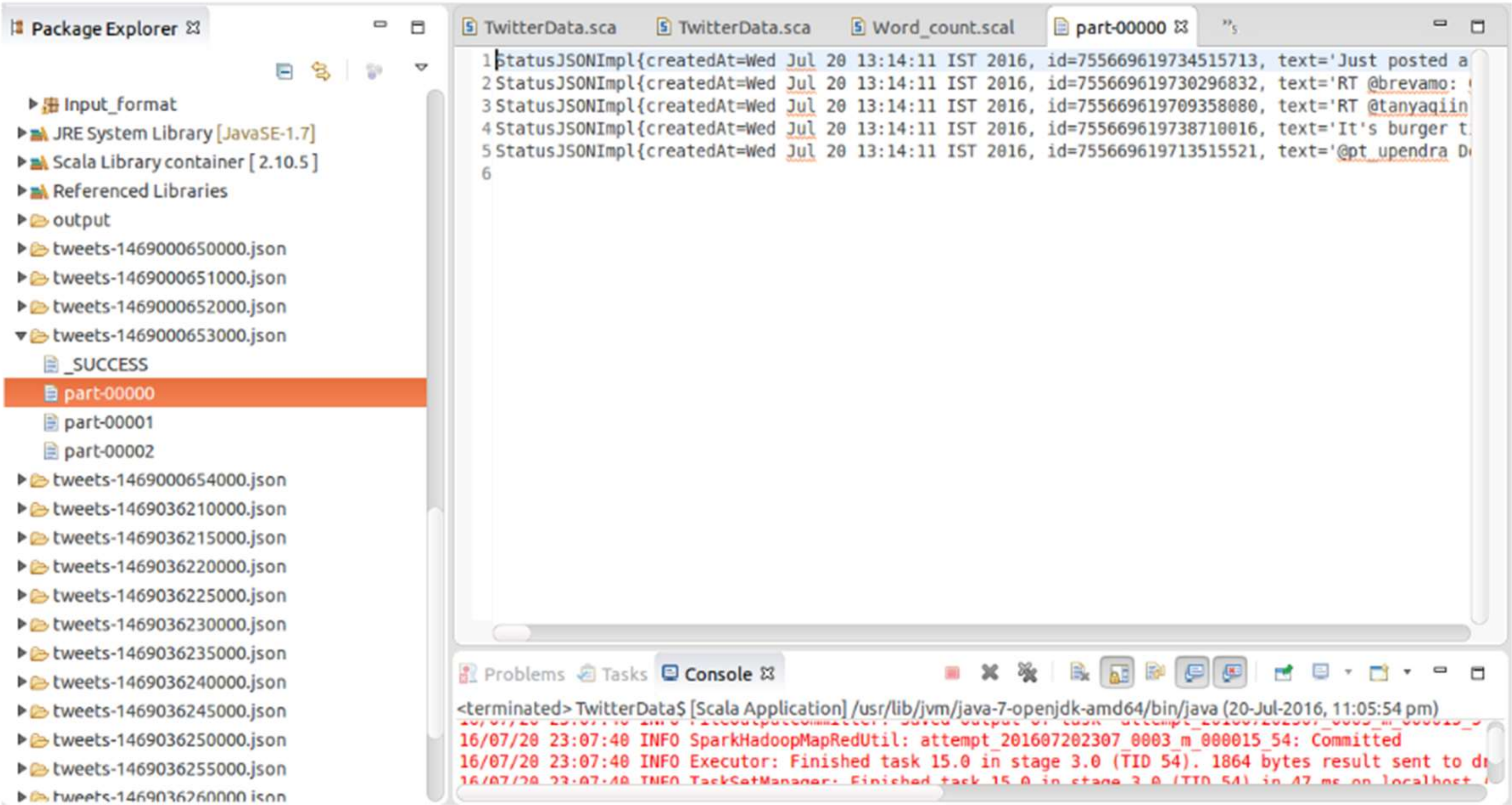
- val tweets = TwitterUtils.createStream(ssc, Some(auth))
- val englishTweets = tweets.filter(_.getLang() == "en")
- Run the class file and then console window is appeared.

Spark streaming

Problems Tasks Console

```
<terminated> TwitterData$ [Scala Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (20-Jul-2016, 11:05:54 pm)
16/07/20 23:07:40 INFO SparkHadoopMapRedUtil: attempt_201607202307_0003_m_000015_54: Committed
16/07/20 23:07:40 INFO Executor: Finished task 15.0 in stage 3.0 (TID 54). 1864 bytes result sent to driver
16/07/20 23:07:40 INFO TaskSetManager: Finished task 15.0 in stage 3.0 (TID 54) in 47 ms on localhost (16/17)
16/07/20 23:07:40 INFO BlockManager: Found block input-0-1469036259400 locally
16/07/20 23:07:40 INFO FileOutputCommitter: Saved output of task 'attempt_201607202307_0003_m_000016_55' to file:/home/kiran/workspace/Word_Count/twe
16/07/20 23:07:40 INFO SparkHadoopMapRedUtil: attempt_201607202307_0003_m_000016_55: Committed
16/07/20 23:07:40 INFO Executor: Finished task 16.0 in stage 3.0 (TID 55). 1864 bytes result sent to driver
16/07/20 23:07:40 INFO TaskSetManager: Finished task 16.0 in stage 3.0 (TID 55) in 26 ms on localhost (17/17)
16/07/20 23:07:40 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
16/07/20 23:07:40 INFO DAGScheduler: ResultStage 3 (saveAsTextFiles at TwitterData.scala:34) finished in 0.304 s
16/07/20 23:07:40 INFO DAGScheduler: Job 11 finished: saveAsTextFiles at TwitterData.scala:34, took 0.368559 s
16/07/20 23:07:40 INFO JobScheduler: Finished job streaming job 1469036260000 ms.0 from job set of time 1469036260000 ms
16/07/20 23:07:40 INFO JobScheduler: Total delay: 0.944 s for time 1469036260000 ms (execution: 0.929 s)
16/07/20 23:07:40 INFO MapPartitionsRDD: Removing RDD 29 from persistence list
16/07/20 23:07:40 INFO BlockRDD: Removing RDD 28 from persistence list
16/07/20 23:07:40 INFO BlockManager: Removing RDD 29
16/07/20 23:07:40 INFO BlockManager: Removing RDD 28
16/07/20 23:07:40 INFO TwitterInputDStream: Removing blocks of RDD BlockRDD[28] at createStream at TwitterData.scala:32 of time 1469036260000 ms
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036249800 on localhost:33404 in memory (size: 8.9 KB, free: 946.7 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036250000 on localhost:33404 in memory (size: 65.4 KB, free: 946.8 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036250200 on localhost:33404 in memory (size: 3.6 KB, free: 946.8 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036250600 on localhost:33404 in memory (size: 6.0 KB, free: 946.8 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036250800 on localhost:33404 in memory (size: 37.9 KB, free: 946.8 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036251000 on localhost:33404 in memory (size: 26.7 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036251400 on localhost:33404 in memory (size: 3.5 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036251600 on localhost:33404 in memory (size: 3.1 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036251800 on localhost:33404 in memory (size: 41.2 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036252000 on localhost:33404 in memory (size: 17.1 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036252200 on localhost:33404 in memory (size: 3.6 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036252400 on localhost:33404 in memory (size: 4.5 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036252600 on localhost:33404 in memory (size: 3.9 KB, free: 946.9 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036252800 on localhost:33404 in memory (size: 47.3 KB, free: 947.0 MB)
16/07/20 23:07:40 INFO BlockManagerInfo: Removed input-0-1469036253000 on localhost:33404 in memory (size: 51.8 KB, free: 947.0 MB)
```

Output



Package Explorer
Input_format
JRE System Library [JavaSE-1.7]
Scala Library container [2.10.5]
Referenced Libraries
output
tweets-1469000650000.json
tweets-1469000651000.json
tweets-1469000652000.json
tweets-1469000653000.json
_SUCCESS
part-00000
part-00001
part-00002
tweets-1469000654000.json
tweets-1469036210000.json
tweets-1469036215000.json
tweets-1469036220000.json
tweets-1469036225000.json
tweets-1469036230000.json
tweets-1469036235000.json
tweets-1469036240000.json
tweets-1469036245000.json
tweets-1469036250000.json
tweets-1469036255000.json
TwitterData.sca
TwitterData.sca
Word_count.scal
part-00000
StatusJSONImpl{createdAt=Wed Jul 20 13:14:11 IST 2016, id=755669619734515713, text='Just posted a
StatusJSONImpl{createdAt=Wed Jul 20 13:14:11 IST 2016, id=755669619730296832, text='RT @brevamo:
StatusJSONImpl{createdAt=Wed Jul 20 13:14:11 IST 2016, id=755669619709358080, text='RT @tanyaqiin
StatusJSONImpl{createdAt=Wed Jul 20 13:14:11 IST 2016, id=755669619738710016, text='It's burger t
StatusJSONImpl{createdAt=Wed Jul 20 13:14:11 IST 2016, id=755669619713515521, text='@pt_upendra D
Problems
Tasks
Console
<terminated> TwitterData$ [Scala Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (20-Jul-2016, 11:05:54 pm)
16/07/20 23:07:40 INFO SparkHadoopMapRedUtil: attempt_201607202307_0003_m_000015_54: Committed
16/07/20 23:07:40 INFO Executor: Finished task 15.0 in stage 3.0 (TID 54). 1864 bytes result sent to dr

Social network graphs

Social network graphs

- Social Networks as graphs

    - Social networks are naturally modeled as graphs, which we sometimes refer to as a social graph.

- What is a social network??

    - When we think of a social network, we think of Facebook, Twitter, Google+, or another website that is called a “social network,” and indeed this kind of network is representative of the broader class of networks called “social.”

Social network graphs

- In social networks, the entities are the nodes, and an edge connects two nodes if the nodes are related by the relationship that characterizes the network.
- If there is a degree associated with the relationship, this degree is represented by labeling the edges.
- Often, social graphs are undirected, as for the Facebook friends' graph. But they can be directed graphs, as for example the graphs of followers on Twitter or Google+.

An example social network

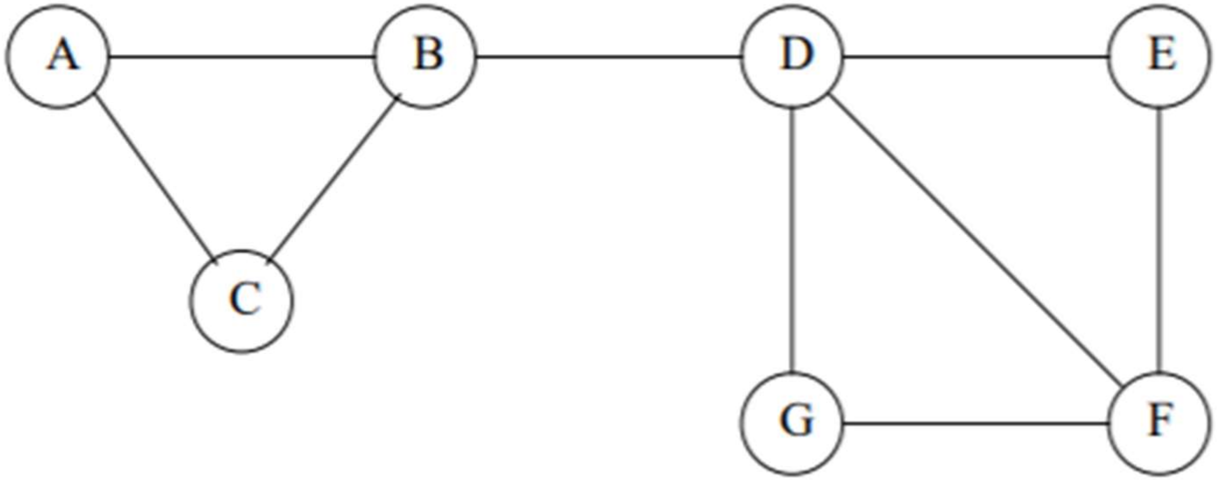
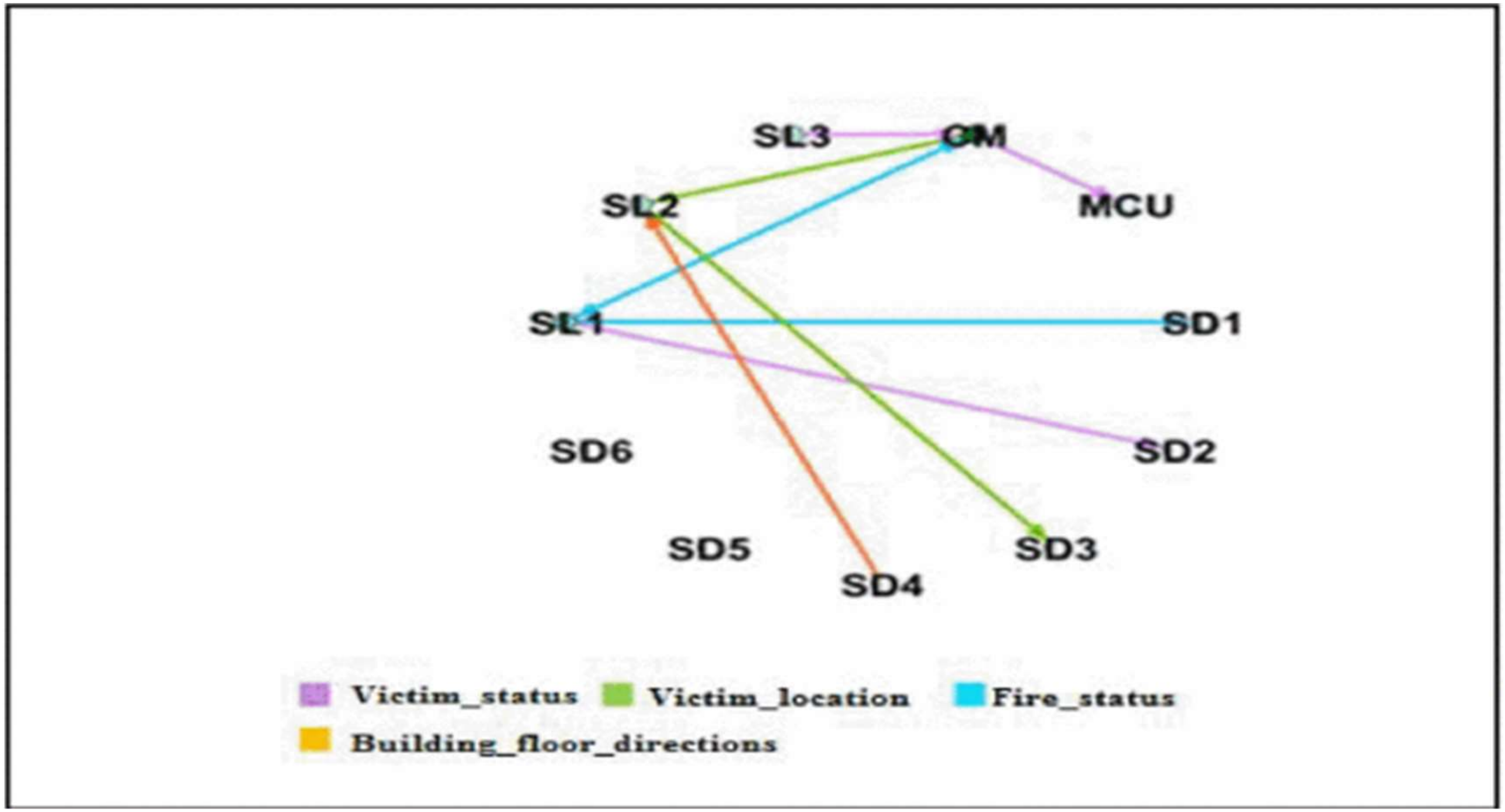


Figure 10.1: Example of a small social network

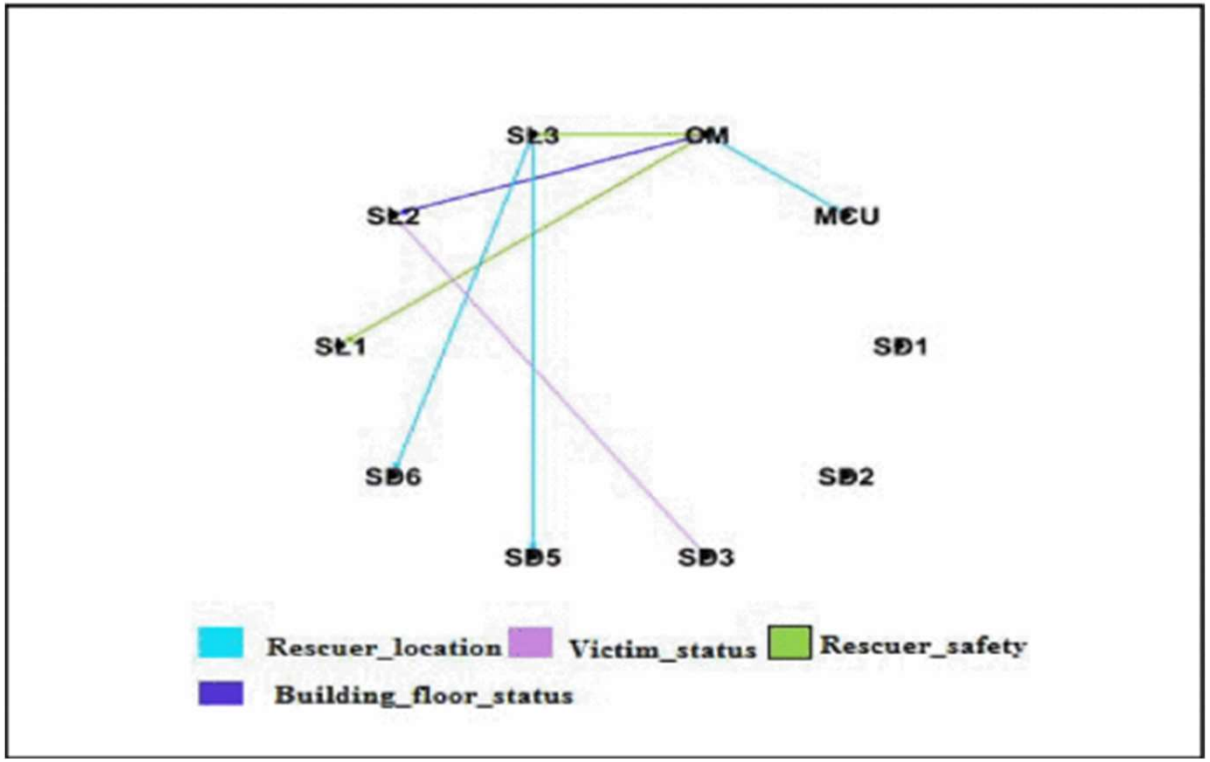
Social network analysis

- Social network analysis (SNA) is “the process of investigating social structures through the use of networks and graph theory”.
- Under the SNA model, social networks are represented by “nodes” and “edges,” or the elements (e.g. individuals, organizations, ideas) of a given social network and the various relationships that connect them.
- In the case of social networks on Twitter, nodes may represent different users while edges may represent any of the interactions that connect these individual accounts (e.g. likes, replies, retweets, mentions etc.).

SNA example 1:



SNA Example 2:



Example 3

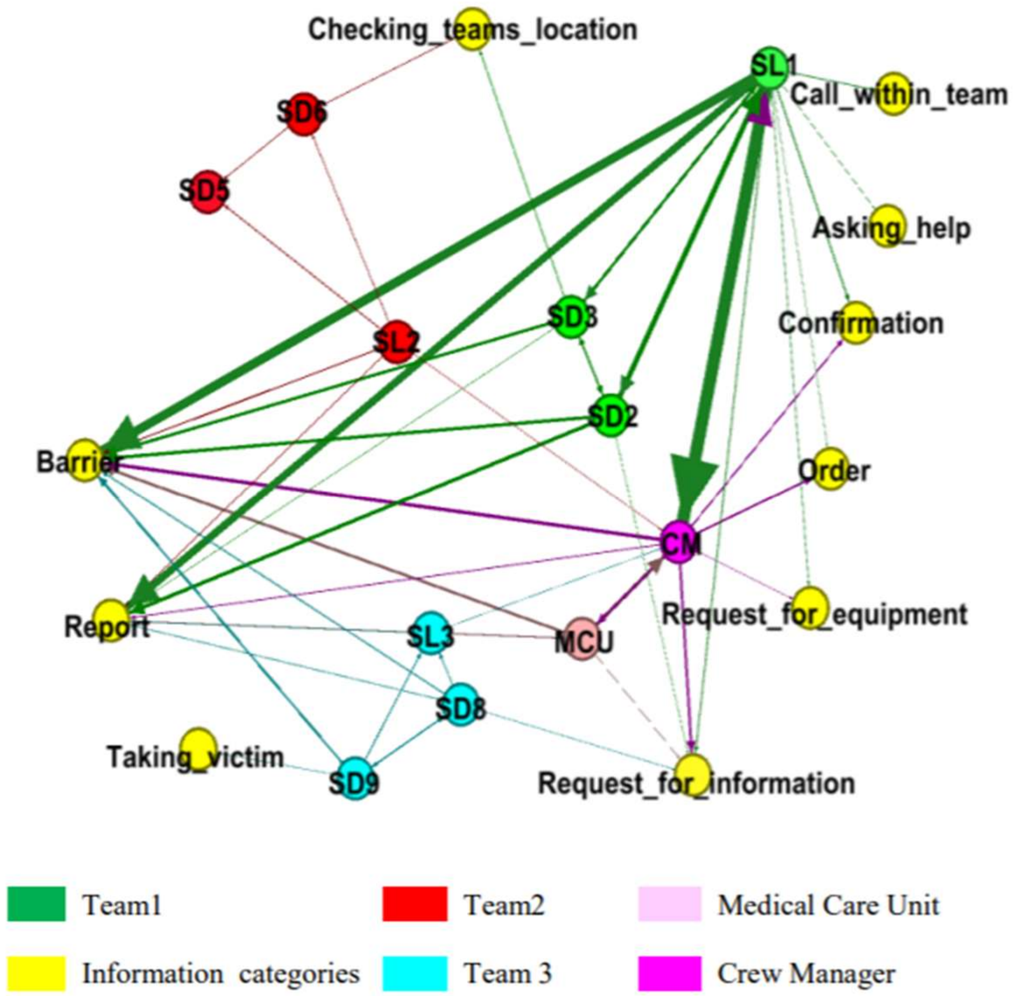
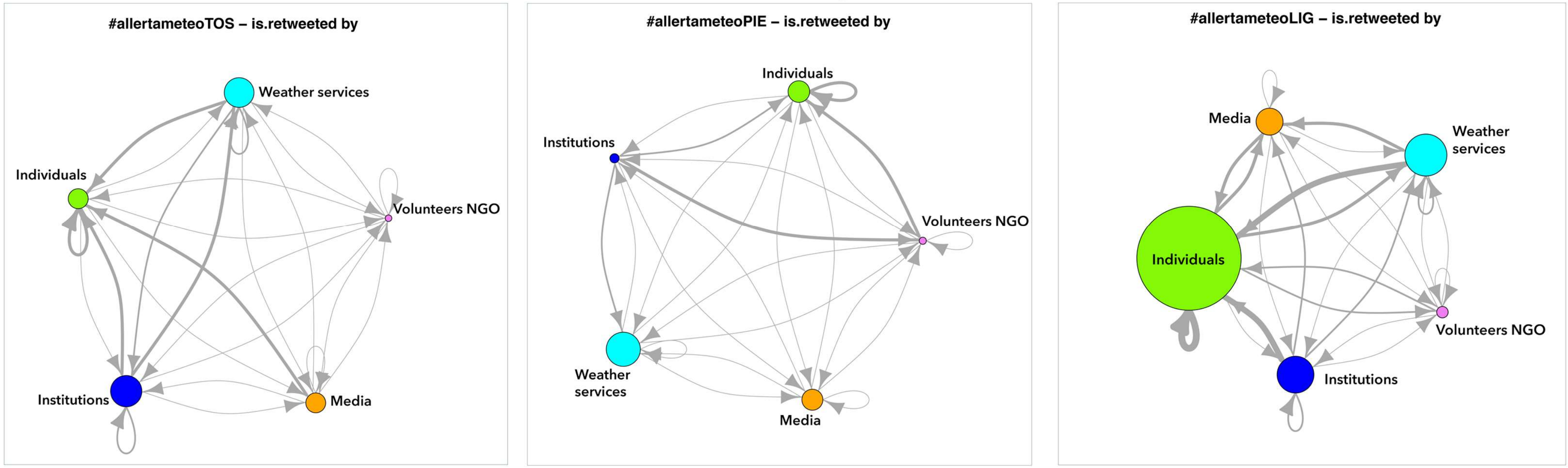


Figure 2. Information communication and tracking Network

Twitter data on SNG

Who is retweeted by whom?



Conclusions

- Social media
- Social media use in disasters
- What streaming is?
- Why Spark streaming is important?
- Different Spark streaming components
- Example: Real-time twitter data stream processing with Apache Spark