CS535 BIG DATA

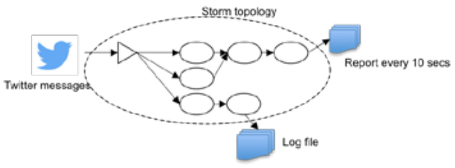
PART 2. SCALABLE FRAMEWORKS FOR REAL-TIME BIG DATA ANALYTICS

1. SPEED LAYER: APACHE STORM

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

FAQs

- Google credit available
- Assignment 2 has been posted



Today's topics

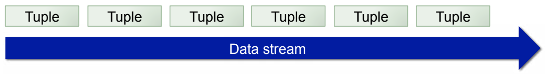
- Storm model
  - Architecture

Speed layer: Apache Storm

Storm model

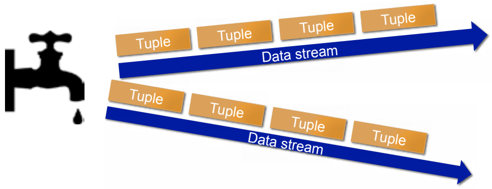
Storm Model

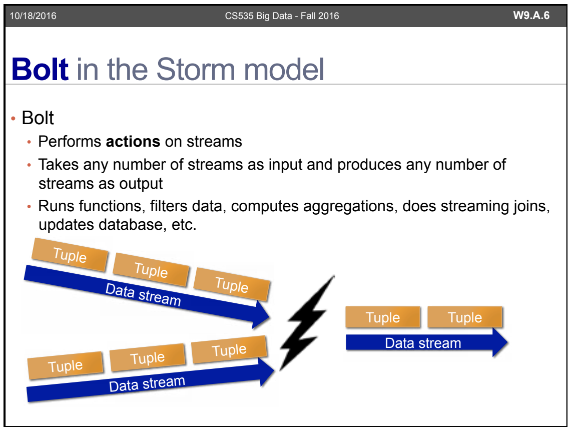
- One-at-a-time stream processing
- Represents the entire stream processing pipeline as a graph of computation called a **topology**
- A single program is deployed across a cluster
- A stream is represented an infinite sequence of **tuples**
  - **A tuple: a named list of values**



Spout in the Storm model

- Spout
  - A source of streams in a topology
  - A spout can read from a Kestrel or Kafka queue
  - Turns the data into a tuple stream
  - Timer spout could emit a tuple into its output stream every 10 seconds



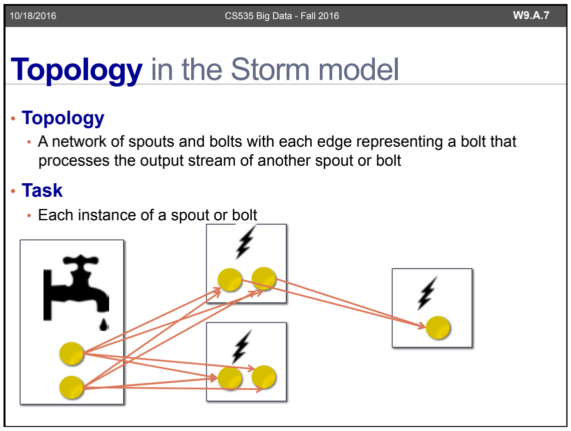


10/18/2016 CS535 Big Data - Fall 2016 W9.A.6

Bolt in the Storm model

- Bolt
  - Performs **actions** on streams
  - Takes any number of streams as input and produces any number of streams as output
  - Runs functions, filters data, computes aggregations, does streaming joins, updates database, etc.

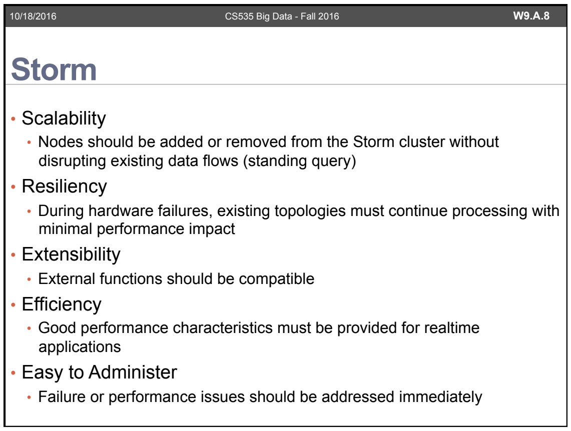
Tuple
Tuple
Tuple
Data stream
Tuple
Tuple
Tuple
Data stream
Tuple
Tuple
Data stream



10/18/2016 CS535 Big Data - Fall 2016 W9.A.7

Topology in the Storm model

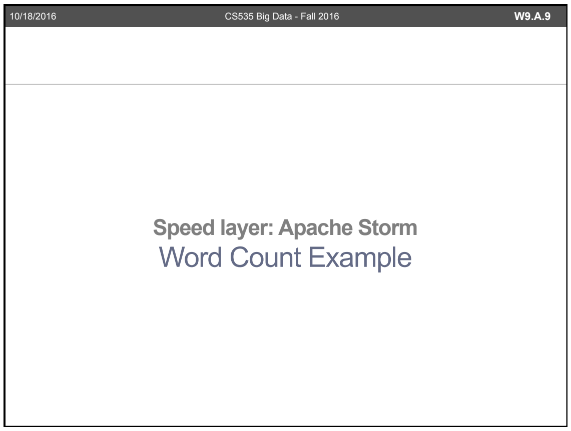
- **Topology**
  - A network of spouts and bolts with each edge representing a bolt that processes the output stream of another spout or bolt
- **Task**
  - Each instance of a spout or bolt



10/18/2016 CS535 Big Data - Fall 2016 W9.A.8

Storm

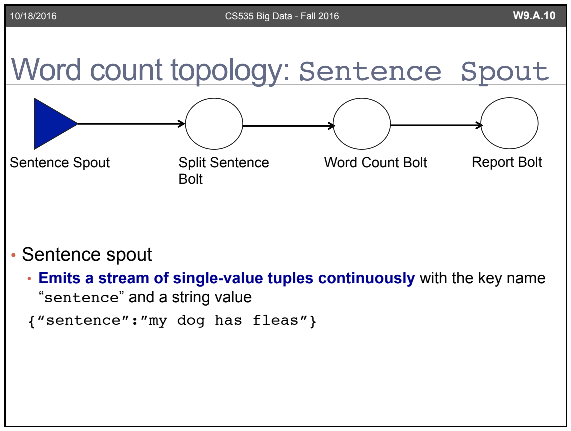
- Scalability
  - Nodes should be added or removed from the Storm cluster without disrupting existing data flows (standing query)
- Resiliency
  - During hardware failures, existing topologies must continue processing with minimal performance impact
- Extensibility
  - External functions should be compatible
- Efficiency
  - Good performance characteristics must be provided for realtime applications
- Easy to Administer
  - Failure or performance issues should be addressed immediately



10/18/2016 CS535 Big Data - Fall 2016 W9.A.9

Speed layer: Apache Storm

Word Count Example



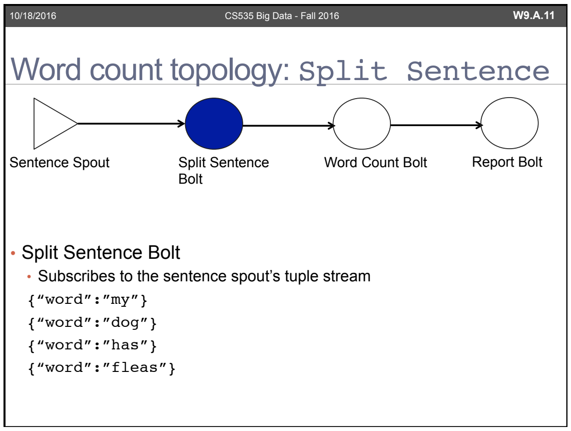
10/18/2016 CS535 Big Data - Fall 2016 W9.A.10

Word count topology: `Sentence Spout`

Sentence Spout
Split Sentence Bolt
Word Count Bolt
Report Bolt

- Sentence spout
  - **Emits a stream of single-value tuples continuously** with the key name "sentence" and a string value

```
{"sentence":"my dog has fleas"}
```



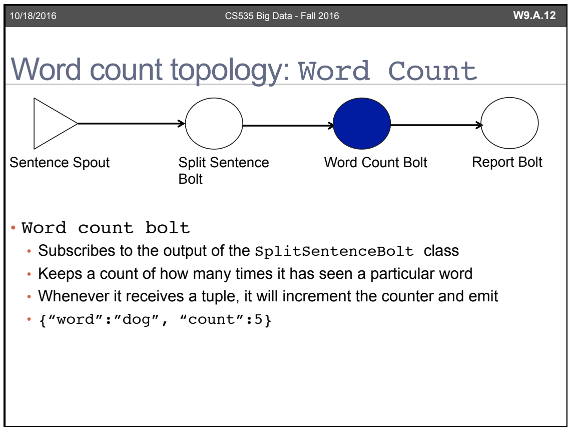
10/18/2016 CS535 Big Data - Fall 2016 W9.A.11

Word count topology: `Split Sentence`

Sentence Spout
Split Sentence Bolt
Word Count Bolt
Report Bolt

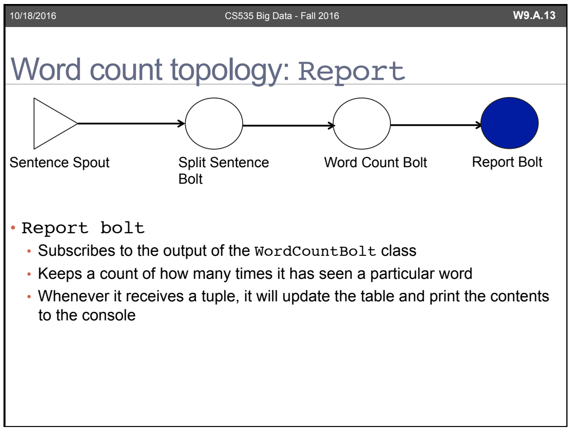
- Split Sentence Bolt
  - Subscribes to the sentence spout's tuple stream

```
{"word":"my"}
{"word":"dog"}
{"word":"has"}
{"word":"fleas"}
```


10/18/2016 CS535 Big Data - Fall 2016 W9.A.12

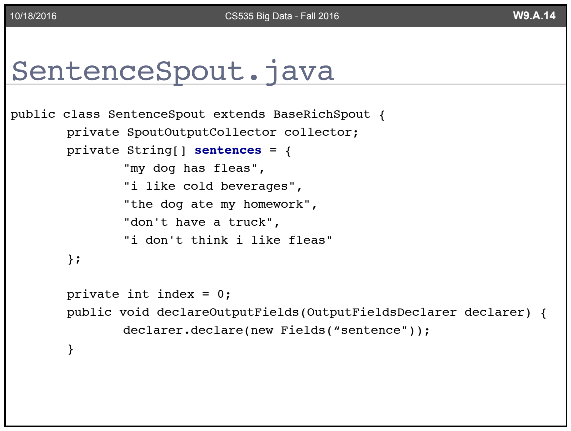
Word count topology: `Word Count`

- `Word count bolt`
  - Subscribes to the output of the `SplitSentenceBolt` class
  - Keeps a count of how many times it has seen a particular word
  - Whenever it receives a tuple, it will increment the counter and emit
  - `{“word”:”dog”, “count”:5}`


10/18/2016 CS535 Big Data - Fall 2016 W9.A.13

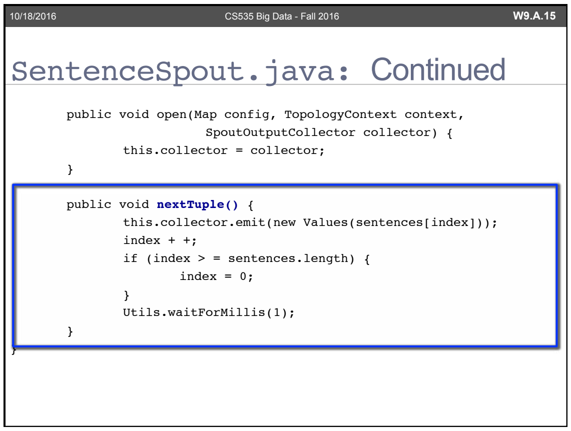
Word count topology: `Report`

- `Report bolt`
  - Subscribes to the output of the `WordCountBolt` class
  - Keeps a count of how many times it has seen a particular word
  - Whenever it receives a tuple, it will update the table and print the contents to the console


10/18/2016 CS535 Big Data - Fall 2016 W9.A.14

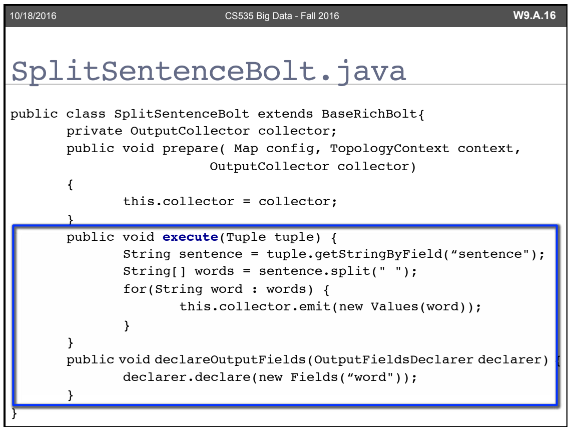
`SentenceSpout.java`

```
public class SentenceSpout extends BaseRichSpout {
        private SpoutOutputCollector collector;
        private String[] sentences = {
                "my dog has fleas",
                "i like cold beverages",
                "the dog ate my homework",
                "don't have a truck",
                "i don't think i like fleas"
        };

        private int index = 0;
        public void declareOutputFields(OutputFieldsDeclarer declarer) {
                declarer.declare(new Fields(“sentence"));
        }
```


10/18/2016 CS535 Big Data - Fall 2016 W9.A.15

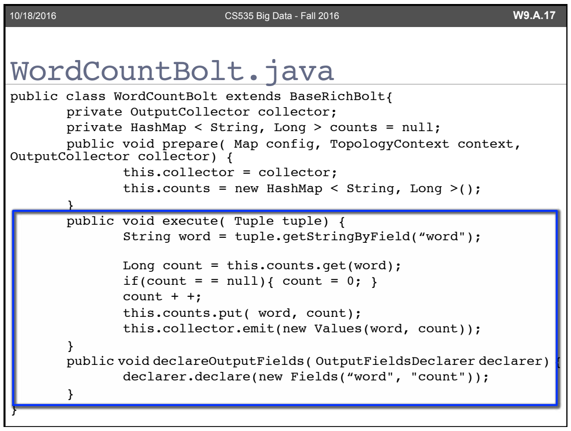
`SentenceSpout.java:` Continued

```
        public void open(Map config, TopologyContext context,
                        SpoutOutputCollector collector) {
                this.collector = collector;
        }

        public void nextTuple() {
                this.collector.emit(new Values(sentences[index]));
                index + +;
                if (index > = sentences.length) {
                        index = 0;
                }
                Utils.waitForMillis(1);
        }
}
```


10/18/2016 CS535 Big Data - Fall 2016 W9.A.16

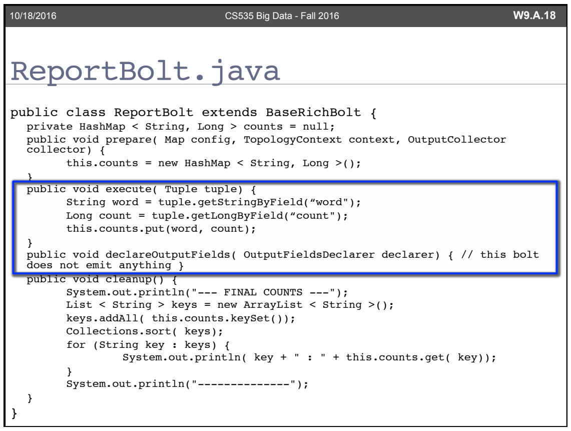
`SplitSentenceBolt.java`

```
public class SplitSentenceBolt extends BaseRichBolt{
        private OutputCollector collector;
        public void prepare( Map config, TopologyContext context,
                            OutputCollector collector)
        {
                this.collector = collector;
        }
        public void execute(Tuple tuple) {
                String sentence = tuple.getStringByField(“sentence");
                String[] words = sentence.split(" ");
                for(String word : words) {
                        this.collector.emit(new Values(word));
                }
        }
        public void declareOutputFields(OutputFieldsDeclarer declarer) {
                declarer.declare(new Fields(“word"));
        }
}
```


10/18/2016 CS535 Big Data - Fall 2016 W9.A.17

`WordCountBolt.java`

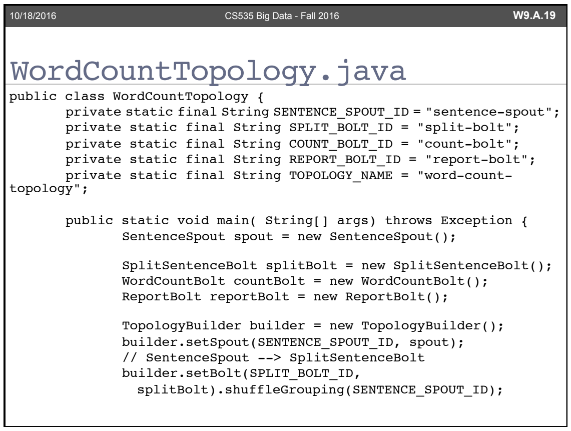
```
public class WordCountBolt extends BaseRichBolt{
        private OutputCollector collector;
        private HashMap < String, Long > counts = null;
        public void prepare( Map config, TopologyContext context,
OutputCollector collector) {
                this.collector = collector;
                this.counts = new HashMap < String, Long >();
        }
        public void execute( Tuple tuple) {
                String word = tuple.getStringByField(“word");

                Long count = this.counts.get(word);
                if(count = = null){ count = 0; }
                count + +;
                this.counts.put( word, count);
                this.collector.emit(new Values(word, count));
        }
        public void declareOutputFields( OutputFieldsDeclarer declarer) {
                declarer.declare(new Fields(“word", "count"));
        }
}
```



10/18/2016 CS535 Big Data - Fall 2016 W9.A.18

ReportBolt.java

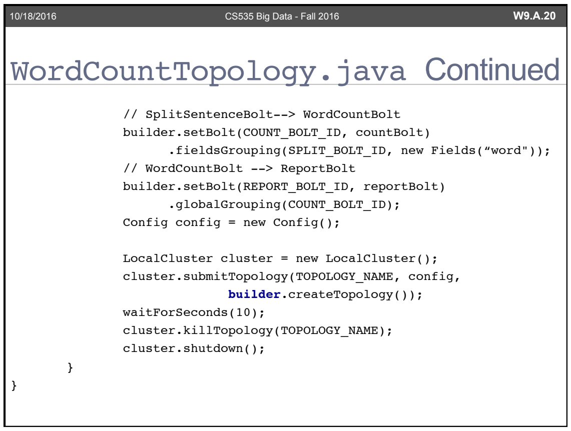
```
public class ReportBolt extends BaseRichBolt {
  private HashMap < String, Long > counts = null;
  public void prepare( Map config, TopologyContext context, OutputCollector
  collector) {
        this.counts = new HashMap < String, Long >();
  }
  public void execute( Tuple tuple) {
        String word = tuple.getStringByField(“word");
        Long count = tuple.getLongByField(“count");
        this.counts.put(word, count);
  }
  public void declareOutputFields( OutputFieldsDeclarer declarer) { // this bolt
  does not emit anything }
  public void cleanup() {
        System.out.println("--- FINAL COUNTS ---");
        List < String > keys = new ArrayList < String >();
        keys.addAll( this.counts.keySet());
        Collections.sort( keys);
        for (String key : keys) {
               System.out.println( key + " : " + this.counts.get( key));
        }
        System.out.println("--------------");
  }
}
```



10/18/2016 CS535 Big Data - Fall 2016 W9.A.19

WordCountTopology.java

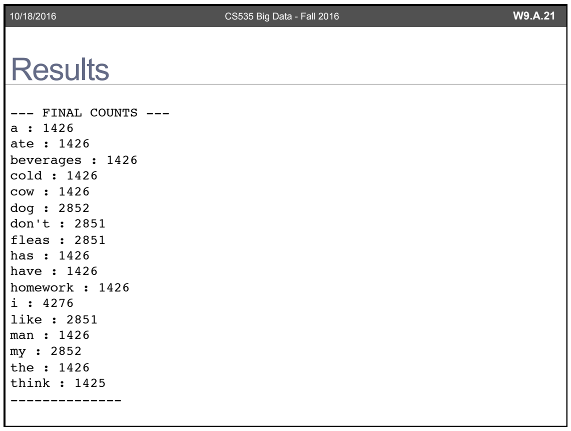
```
public class WordCountTopology {
       private static final String SENTENCE_SPOUT_ID = "sentence-spout";
       private static final String SPLIT_BOLT_ID = "split-bolt";
       private static final String COUNT_BOLT_ID = "count-bolt";
       private static final String REPORT_BOLT_ID = "report-bolt";
       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);
              // SentenceSpout --> SplitSentenceBolt
              builder.setBolt(SPLIT_BOLT_ID,
                splitBolt).shuffleGrouping(SENTENCE_SPOUT_ID);
```



10/18/2016 CS535 Big Data - Fall 2016 W9.A.20

WordCountTopology.java Continued

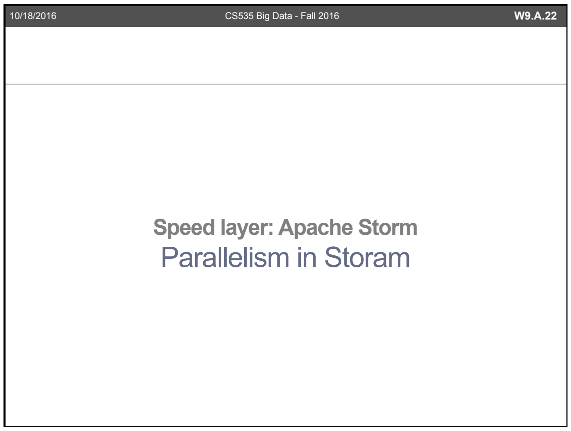
```
              // SplitSentenceBolt--> WordCountBolt
              builder.setBolt(COUNT_BOLT_ID, countBolt)
                     .fieldsGrouping(SPLIT_BOLT_ID, new Fields(“word"));
              // WordCountBolt --> ReportBolt
              builder.setBolt(REPORT_BOLT_ID, reportBolt)
                     .globalGrouping(COUNT_BOLT_ID);
              Config config = new Config();

              LocalCluster cluster = new LocalCluster();
              cluster.submitTopology(TOPOLOGY_NAME, config,
                            builder.createTopology());
              waitForSeconds(10);
              cluster.killTopology(TOPOLOGY_NAME);
              cluster.shutdown();
       }
}
```



10/18/2016 CS535 Big Data - Fall 2016 W9.A.21

Results

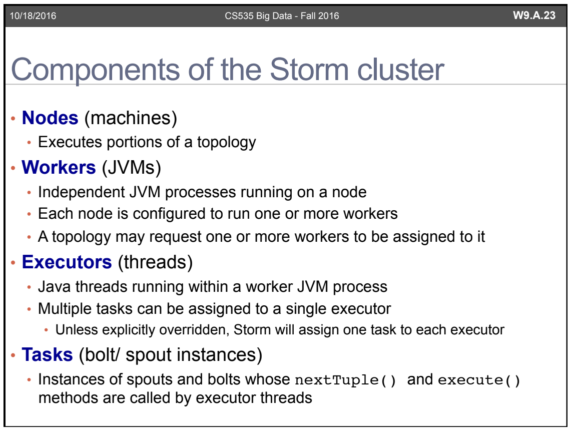
```
--- FINAL COUNTS ---
a : 1426
ate : 1426
beverages : 1426
cold : 1426
cow : 1426
dog : 2852
don't : 2851
fleas : 2851
has : 1426
have : 1426
homework : 1426
i : 4276
like : 2851
man : 1426
my : 2852
the : 1426
think : 1425
--------------
```



10/18/2016 CS535 Big Data - Fall 2016 W9.A.22

Speed layer: Apache Storm

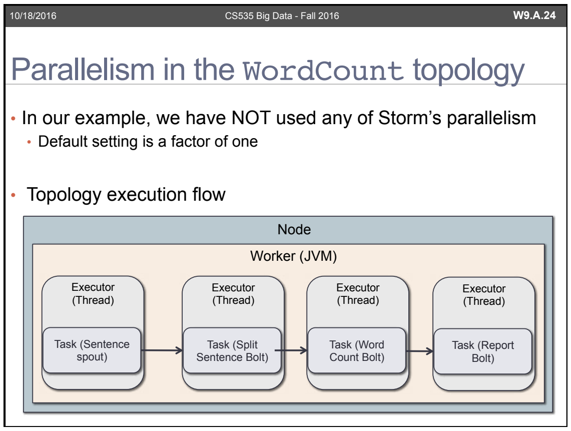
Parallelism in Storam



10/18/2016 CS535 Big Data - Fall 2016 W9.A.23

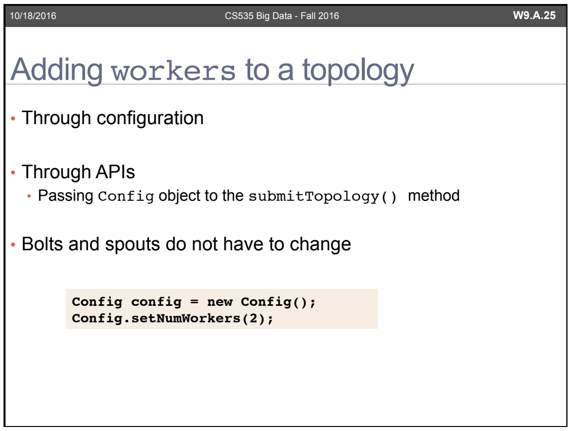
Components of the Storm cluster

- **Nodes** (machines)
  - Executes portions of a topology
- **Workers** (JVMs)
  - Independent JVM processes running on a node
  - Each node is configured to run one or more workers
  - A topology may request one or more workers to be assigned to it
- **Executors** (threads)
  - Java threads running within a worker JVM process
  - Multiple tasks can be assigned to a single executor
    - Unless explicitly overridden, Storm will assign one task to each executor
- **Tasks** (bolt/ spout instances)
  - Instances of spouts and bolts whose `nextTuple()` and `execute()` methods are called by executor threads



Parallelism in the `WordCount` topology

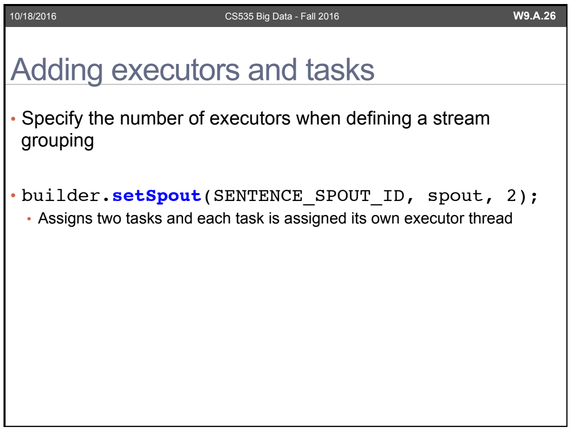
- In our example, we have NOT used any of Storm's parallelism
  - Default setting is a factor of one
- Topology execution flow



Adding `workers` to a topology

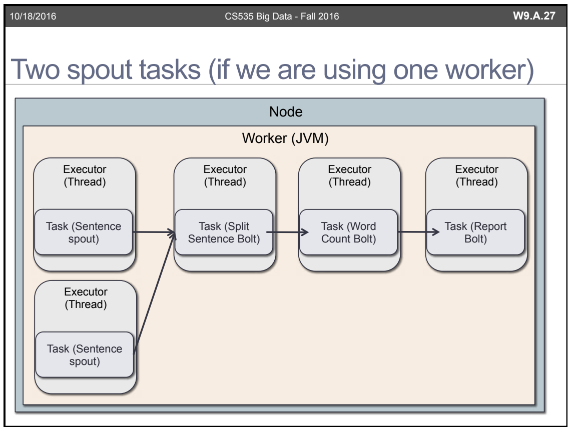
- Through configuration
- Through APIs
  - Passing `Config` object to the `submitTopology()` method
- Bolts and spouts do not have to change

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

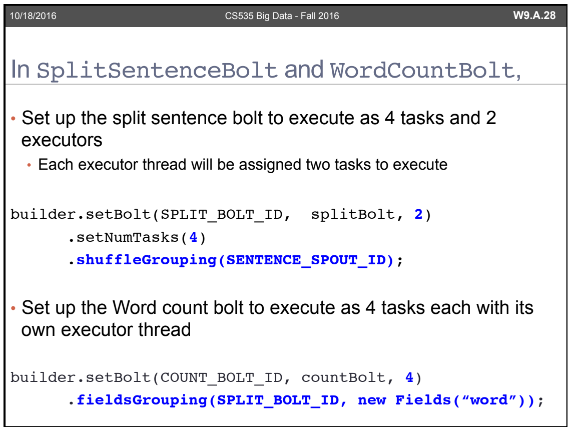


Adding executors and tasks

- Specify the number of executors when defining a stream grouping
- `builder.setSpout(SENTENCE_SPOUT_ID, spout, 2);`
  - Assigns two tasks and each task is assigned its own executor thread



Two spout tasks (if we are using one worker)



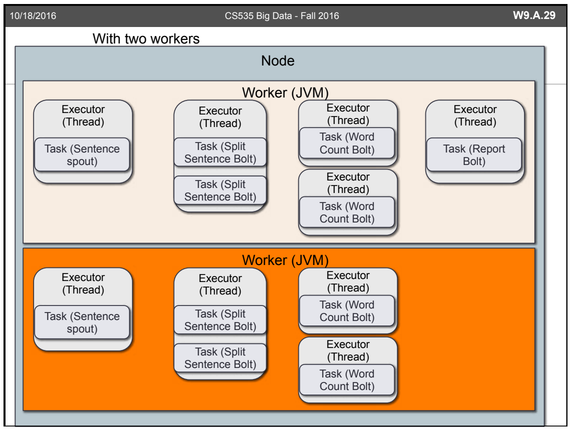
In `SplitSentenceBolt` and `WordCountBolt`,

- Set up the split sentence bolt to execute as 4 tasks and 2 executors
  - Each executor thread will be assigned two tasks to execute

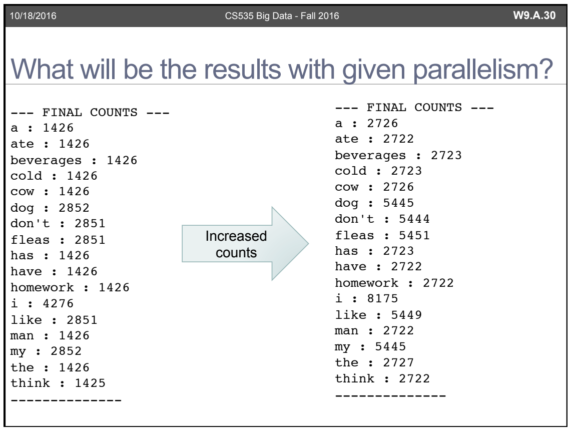
```
builder.setBolt(SPLIT_BOLT_ID,  splitBolt, 2)
      .setNumTasks(4)
      .shuffleGrouping(SENTENCE_SPOUT_ID);
```

- Set up the Word count bolt to execute as 4 tasks each with its own executor thread

```
builder.setBolt(COUNT_BOLT_ID, countBolt, 4)
      .fieldsGrouping(SPLIT_BOLT_ID, new Fields(“word”));
```



With two workers

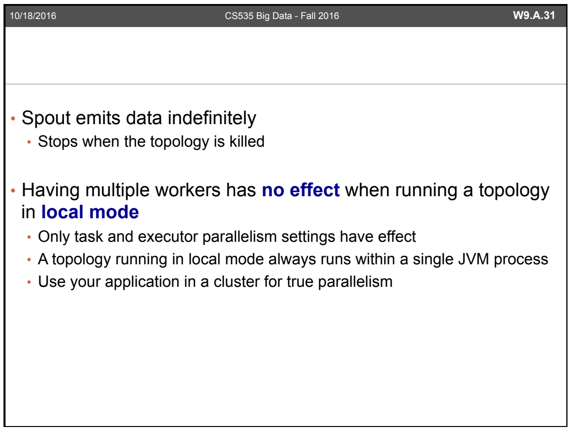

10/18/2016 CS535 Big Data - Fall 2016 W9.A.30

What will be the results with given parallelism?

```
--- FINAL COUNTS ---
a : 1426
ate : 1426
beverages : 1426
cold : 1426
cow : 1426
dog : 2852
don't : 2851
fleas : 2851
has : 1426
have : 1426
homework : 1426
i : 4276
like : 2851
man : 1426
my : 2852
the : 1426
think : 1425
--------------
```

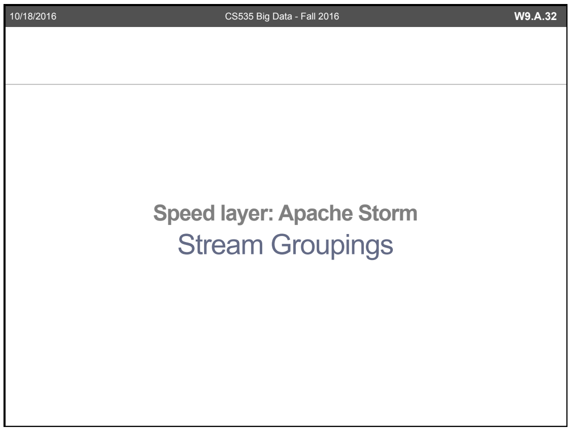
Increased counts

```
--- FINAL COUNTS ---
a : 2726
ate : 2722
beverages : 2723
cold : 2723
cow : 2726
dog : 5445
don't : 5444
fleas : 5451
has : 2723
have : 2722
homework : 2722
i : 8175
like : 5449
man : 2722
my : 5445
the : 2727
think : 2722
--------------
```


10/18/2016 CS535 Big Data - Fall 2016 W9.A.31

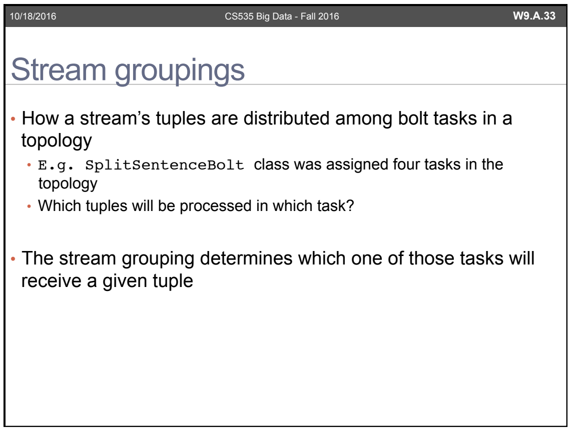
- Spout emits data indefinitely
  - Stops when the topology is killed

- Having multiple workers has **no effect** when running a topology in **local mode**
  - Only task and executor parallelism settings have effect
  - A topology running in local mode always runs within a single JVM process
  - Use your application in a cluster for true parallelism


10/18/2016 CS535 Big Data - Fall 2016 W9.A.32

Speed layer: Apache Storm

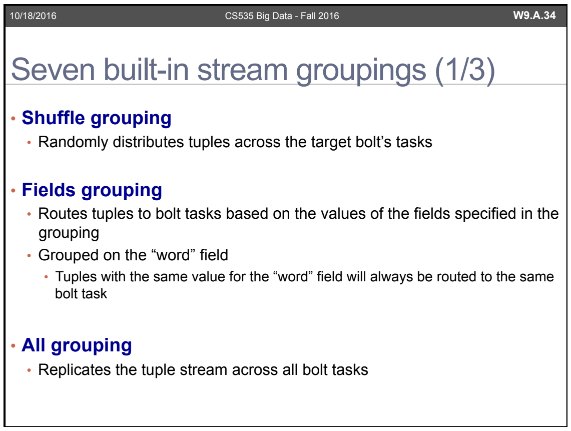
Stream Groupings


10/18/2016 CS535 Big Data - Fall 2016 W9.A.33

Stream groupings

- How a stream's tuples are distributed among bolt tasks in a topology
  - `E.g. SplitSentenceBolt` class was assigned four tasks in the topology
  - Which tuples will be processed in which task?

- The stream grouping determines which one of those tasks will receive a given tuple

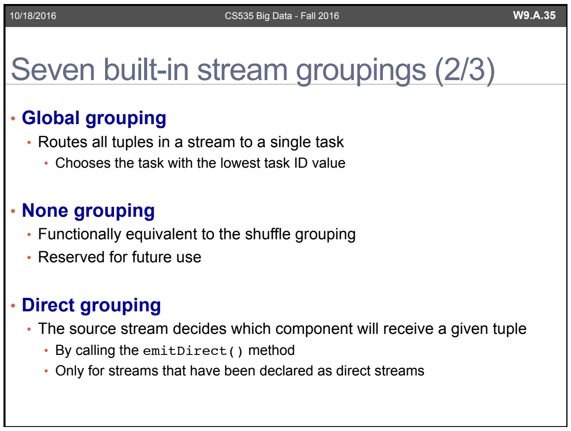

10/18/2016 CS535 Big Data - Fall 2016 W9.A.34

Seven built-in stream groupings (1/3)

- **Shuffle grouping**
  - Randomly distributes tuples across the target bolt's tasks

- **Fields grouping**
  - Routes tuples to bolt tasks based on the values of the fields specified in the grouping
  - Grouped on the "word" field
    - Tuples with the same value for the "word" field will always be routed to the same bolt task

- **All grouping**
  - Replicates the tuple stream across all bolt tasks

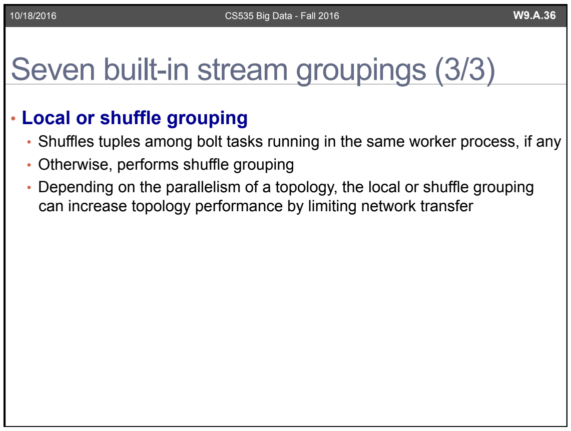

10/18/2016 CS535 Big Data - Fall 2016 W9.A.35

Seven built-in stream groupings (2/3)

- **Global grouping**
  - Routes all tuples in a stream to a single task
    - Chooses the task with the lowest task ID value

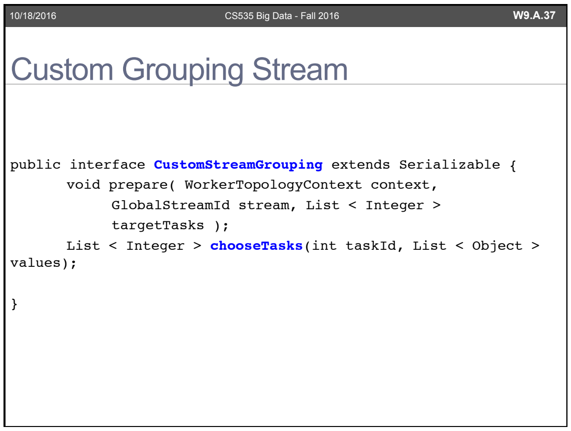
- **None grouping**
  - Functionally equivalent to the shuffle grouping
  - Reserved for future use

- **Direct grouping**
  - The source stream decides which component will receive a given tuple
    - By calling the `emitDirect()` method
    - Only for streams that have been declared as direct streams



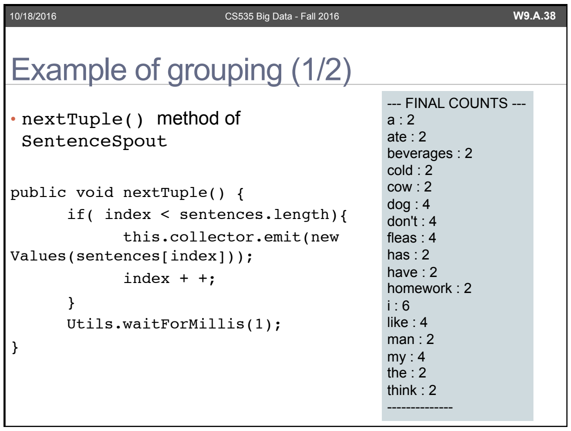
Seven built-in stream groupings (3/3)

- **Local or shuffle grouping**
  - Shuffles tuples among bolt tasks running in the same worker process, if any
  - Otherwise, performs shuffle grouping
  - Depending on the parallelism of a topology, the local or shuffle grouping can increase topology performance by limiting network transfer



Custom Grouping Stream

```
public interface CustomStreamGrouping extends Serializable {
       void prepare( WorkerTopologyContext context,
             GlobalStreamId stream, List < Integer >
             targetTasks );
       List < Integer > chooseTasks(int taskId, List < Object >
values);

}
```

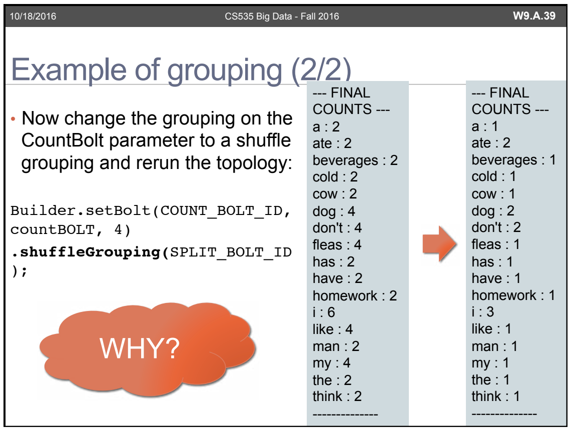


Example of grouping (1/2)

- `nextTuple()` method of `SentenceSpout`

```
public void nextTuple() {
       if( index < sentences.length){
              this.collector.emit(new
Values(sentences[index]));
              index + +;
       }
       Utils.waitForMillis(1);
}
```

```
--- FINAL COUNTS ---
a : 2
ate : 2
beverages : 2
cold : 2
cow : 2
dog : 4
don't : 4
fleas : 4
has : 2
have : 2
homework : 2
i : 6
like : 4
man : 2
my : 4
the : 2
think : 2
--------------
```



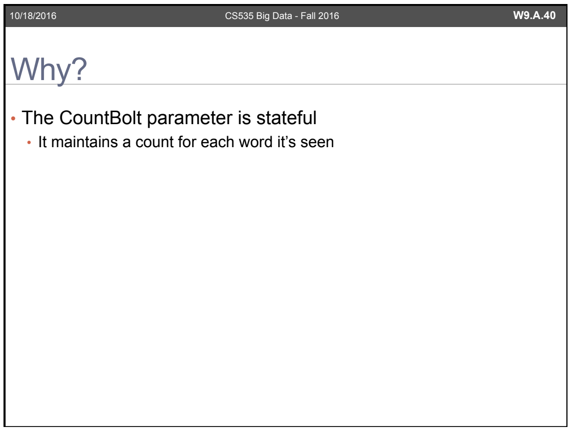
Example of grouping (2/2)

- Now change the grouping on the CountBolt parameter to a shuffle grouping and rerun the topology:

```
Builder.setBolt(COUNT_BOLT_ID,
countBOLT, 4)
.shuffleGrouping(SPLIT_BOLT_ID
);
```

```
--- FINAL
COUNTS ---
a : 2
ate : 2
beverages : 2
cold : 2
cow : 2
dog : 4
don't : 4
fleas : 4
has : 2
have : 2
homework : 2
i : 6
like : 4
man : 2
my : 4
the : 2
think : 2
--------------
```

```
--- FINAL
COUNTS ---
a : 1
ate : 2
beverages : 1
cold : 1
cow : 1
dog : 2
don't : 2
fleas : 1
has : 1
have : 1
homework : 1
i : 3
like : 1
man : 1
my : 1
the : 1
think : 1
--------------
```

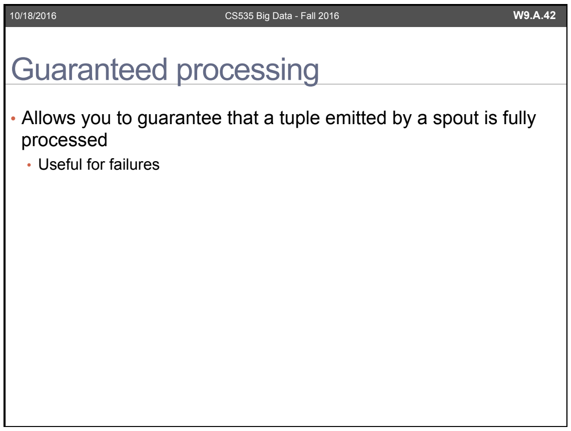


Why?

- The CountBolt parameter is stateful
  - It maintains a count for each word it's seen

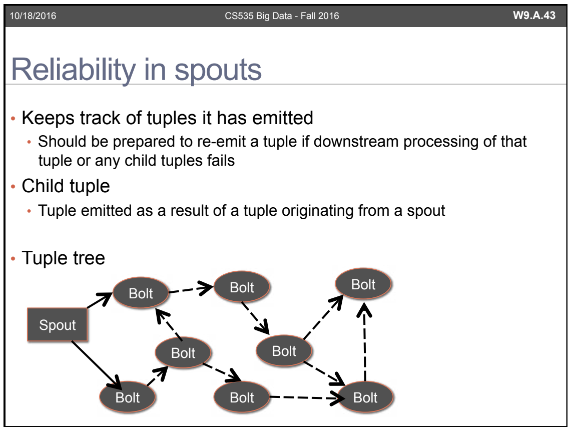
Speed layer: Apache Storm

Reliability in Storm



Guaranteed processing

- Allows you to guarantee that a tuple emitted by a spout is fully processed
  - Useful for failures



Reliability in spouts

- Keeps track of tuples it has emitted
  - Should be prepared to re-emit a tuple if downstream processing of that tuple or any child tuples fails
- Child tuple
  - Tuple emitted as a result of a tuple originating from a spout
- Tuple tree