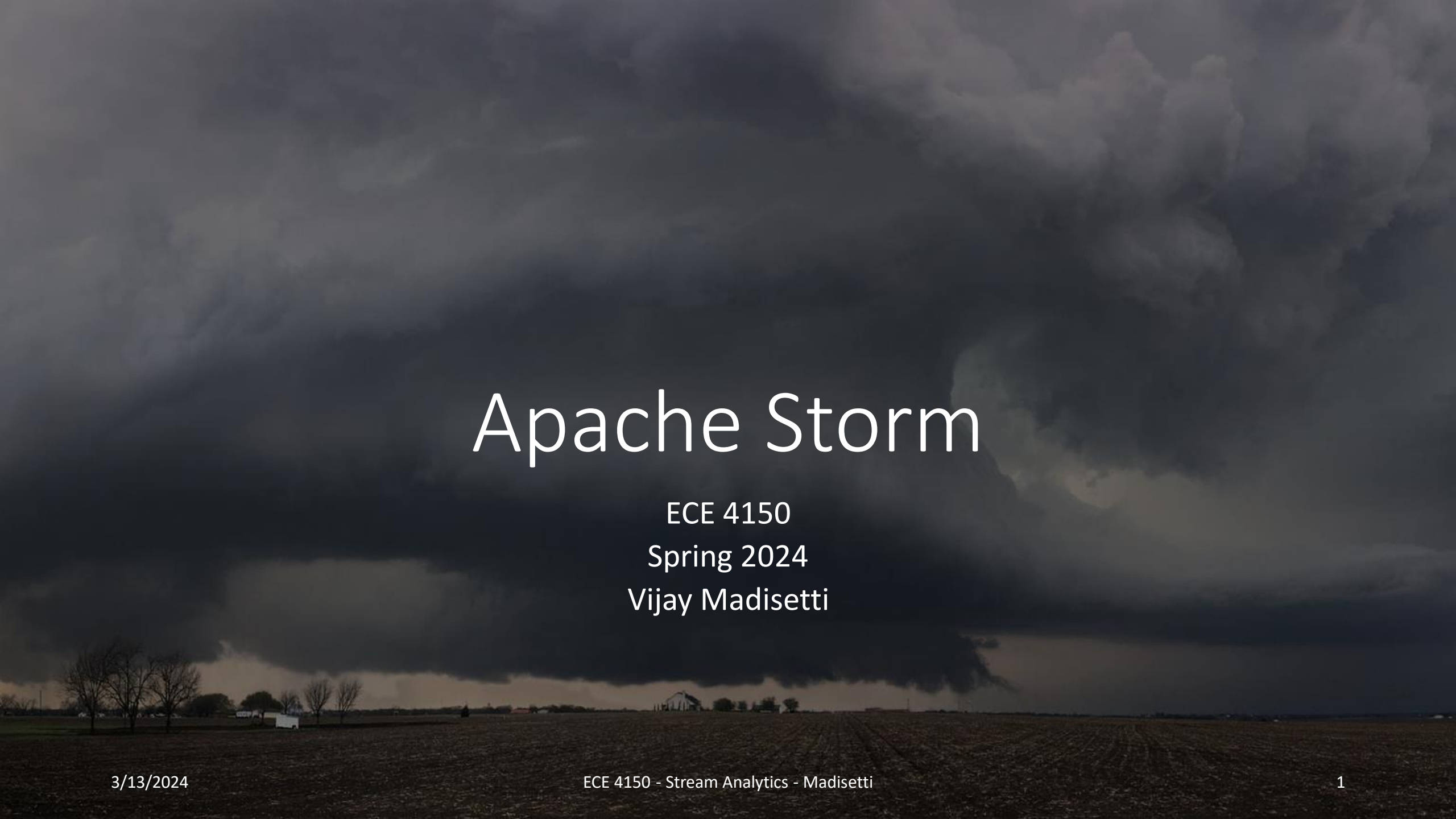


Apache Storm

ECE 4150

Spring 2024

Vijay Madisetti

Plan for Class Till End of Semester

- Next Week is Spring Break
- Total of five labs. One more lab will be released.
- March & 27: AWS EMR in Depth Lectures
- Apr 1- Lecture by Industry Visitor (Zoom likely)
- April 3 – Pre-Recorded Zoom lecture (Real-Time Container Scheduling)
- Project Plan: Short paper (use the template of a typical lab to show the various steps of your work on it) on either Spark or Storm, try out some sample code on Spark or Storm and show that you have used it somehow. Lots of tutorials available online, and I will distribute some handouts as well. Due mid April.

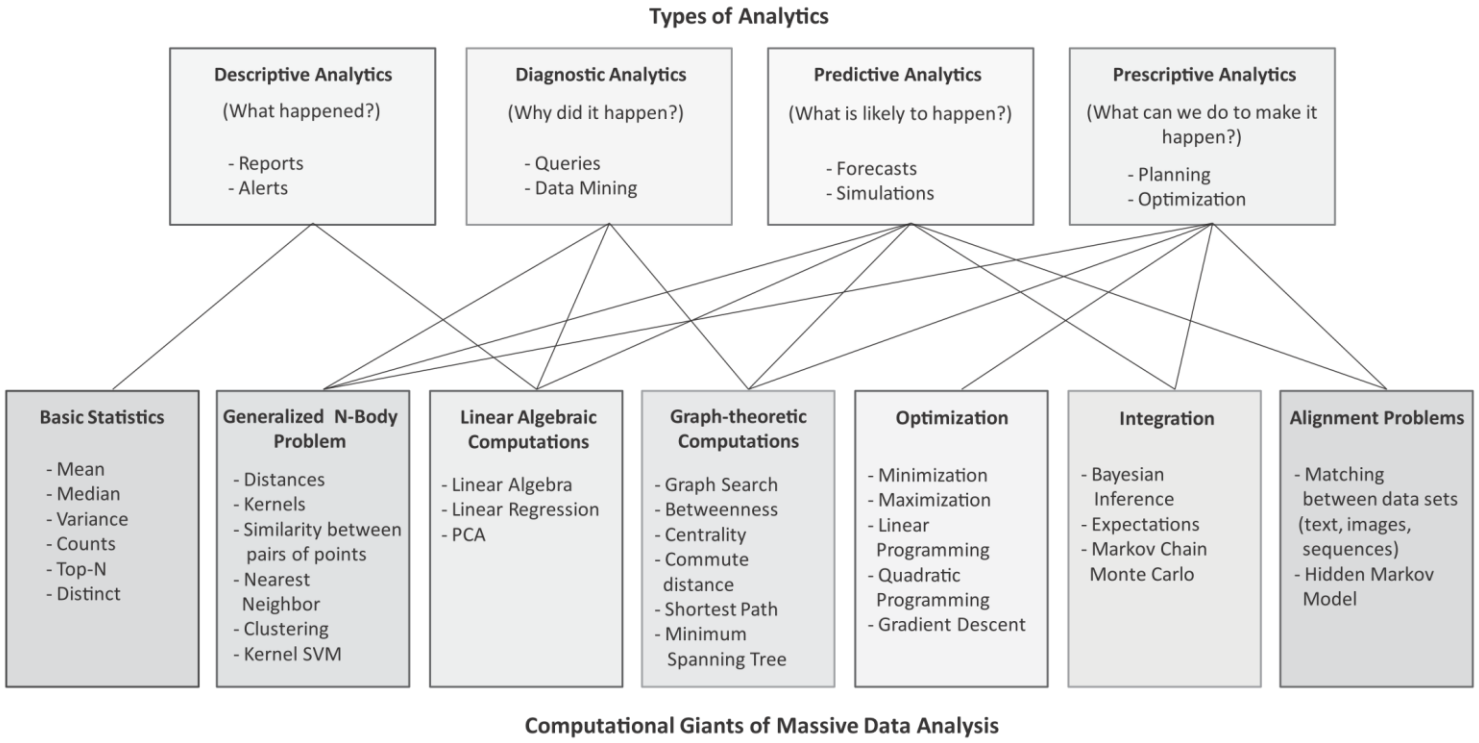
What is Analytics?

- Analytics is a broad term that encompasses the processes, technologies, frameworks and algorithms to extract meaningful insights from data.
- Raw data in itself does not have a meaning until it is contextualized and processed into useful information.
- Analytics is this process of extracting and creating information from raw data by filtering, processing, categorizing, condensing and contextualizing the data.
- This information obtained is then organized and structured to infer knowledge about the system and/or its users, its environment, and its operations and progress towards its objectives, thus making the systems smarter and more efficient.

The Seven Giants

- The National Research Council has done a characterization of computational tasks for massive data analysis (called the seven “giants").
- These computational tasks include:
  (1) Basic Statistics,
  (2) Generalized N-Body Problems,
  (3) Linear Algebraic Computations,
  (4) Graph-Theoretic Computations,
  (5) Optimization,
  (6) Integration and
  (7) Alignment Problems.
- This characterization of computational tasks aims to provide a taxonomy of tasks that have proved to be useful in data analysis and grouping them roughly according to mathematical structure and computational strategy.

Mapping between types of analytics and computational tasks or ‘giants’



Types of Analytics

- **Descriptive Analytics**
  - Descriptive analytics comprises analyzing past data to present it in a summarized form which can be easily interpreted. Descriptive analytics aims to answer - ***What has happened?*** A major portion of analytics done today is descriptive analytics through use of statistics functions such as counts, maximum, minimum, mean, top-N, percentage, for instance. These statistics help in describing patterns in the data and present the data in a summarized form.

- **Diagnostic Analytics**
  - Diagnostic analytics comprises analysis of past data to diagnose the reasons as to why certain events happened. Diagnostic analytics aims to answer - ***Why did it happen?*** While descriptive analytics can be useful for summarizing the data by computing various statistics (such as mean, minimum, maximum, variance, or top-N), diagnostic analytics can provide more insights into why certain a fault has occurred based on the patterns in the sensor data for previous faults.

Types of Analytics

- **Predictive Analytics**
  - Predictive analytics comprises predicting the occurrence of an event or the likely outcome of an event or forecasting the future values using prediction models. Predictive analytics aims to answer - ***What is likely to happen?*** Predictive Analytics is done using predictive models which are trained by existing data. These models learn patterns and trends from the existing data and predict the occurrence of an event or the likely outcome of an event (classification models) or forecast numbers (regression models).

- **Prescriptive Analytics**
  - While predictive analytics uses prediction models to predict the likely outcome of an event, prescriptive analytics uses multiple prediction models to predict various outcomes and the best course of action for each outcome. Prescriptive analytics aims to answer - ***What can we do to make it happen?*** Prescriptive Analytics can predict the possible outcomes based on the current choice of actions.

Domain Specific Examples of Big Data

- Web
- Financial
- Healthcare
- Internet of Things
- Environment
- Logistics & Transportation
- Industry
- Retail

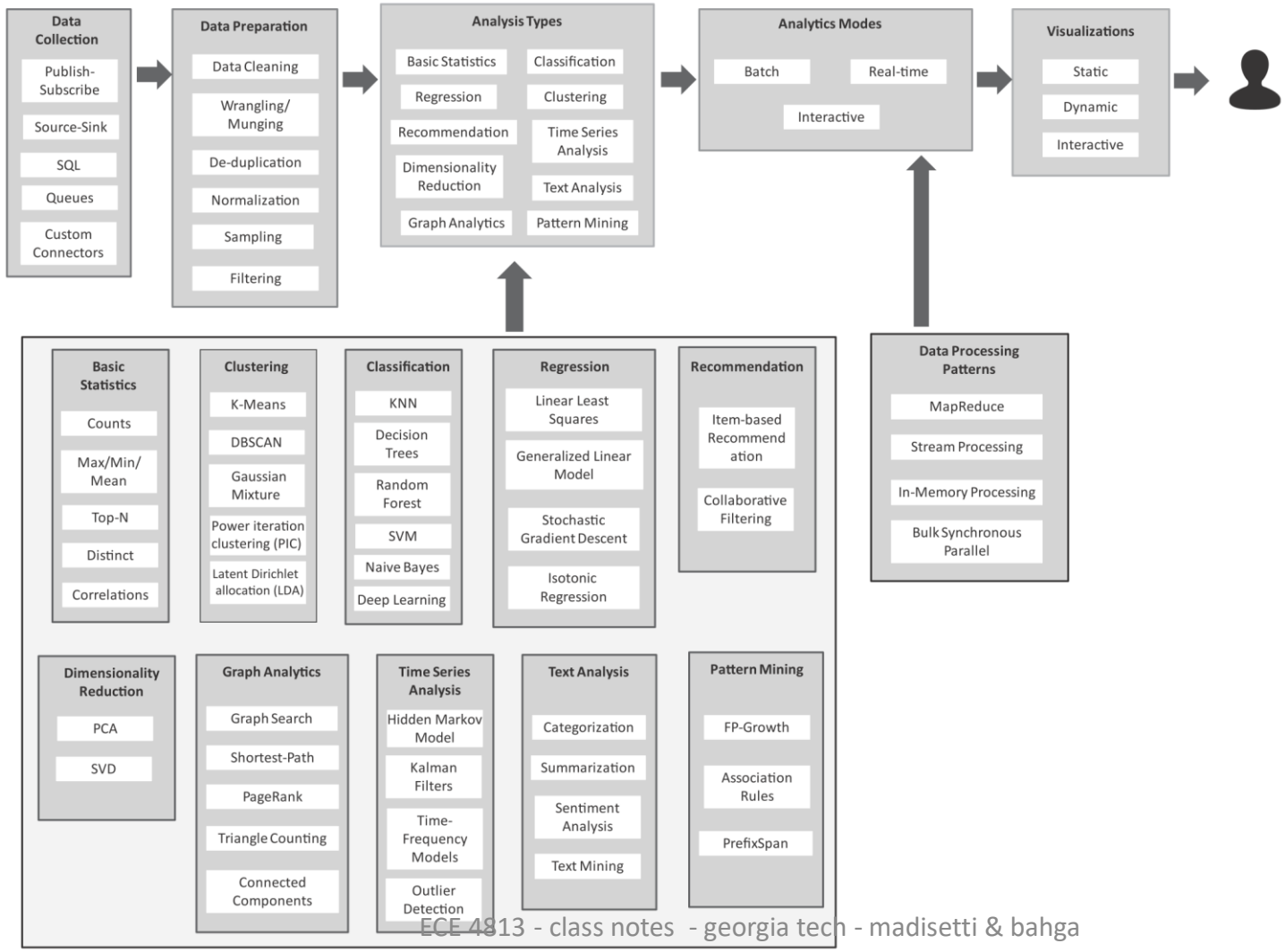
Analytics Flow for Big Data

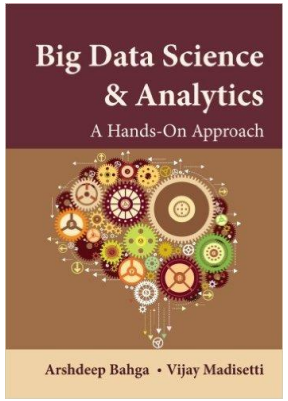
- **Data Collection**
  - Data collection is the first step for any analytics application. Before the data can be analyzed, the data must be collected and ingested into a big data stack. The choice of tools and frameworks for data collection depends on the source of data and the type of data being ingested.
- **Data Preparation**
  - Data can often be dirty and can have various issues that must be resolved before the data can be processed, such as corrupt records, missing values, duplicates, inconsistent abbreviations, inconsistent units, typos, incorrect spellings and incorrect formatting. Data preparation step involves various tasks such as data cleansing, data wrangling or munging, de-duplication, normalization, sampling and filtering.

Analytics Flow for Big Data

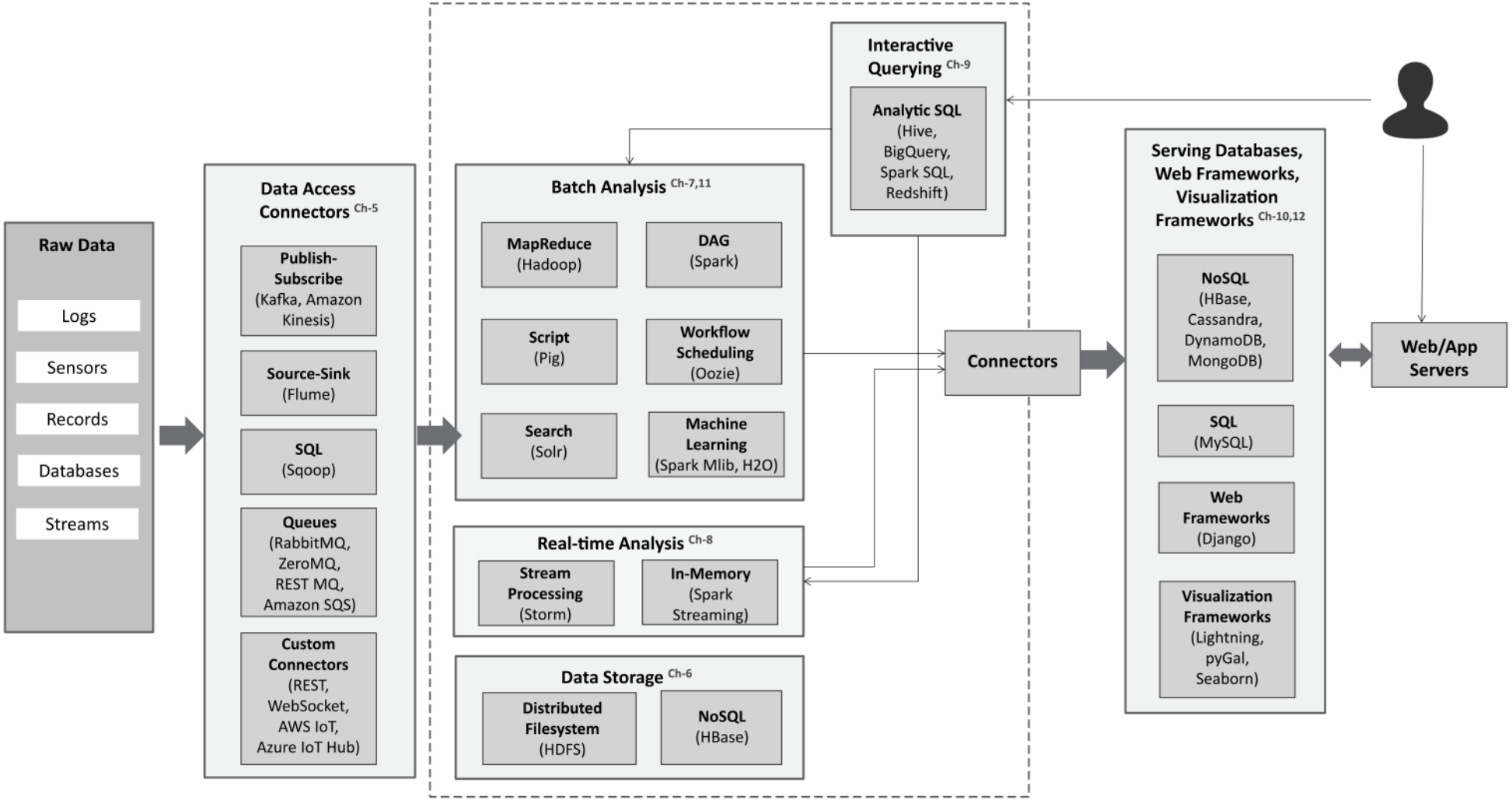
- **Analysis Types**
  - The next step in the analysis flow is to determine the analysis type for the application.
- **Analysis Modes**
  - With the analysis types selected for an application, the next step is to determine the analysis mode, which can be either batch, real-time or interactive. The choice of the mode depends on the requirements of the application.
- **Visualizations**
  - The choice of the visualization tools, serving databases and web frameworks is driven by the requirements of the application. Visualizations can be static, dynamic or interactive.

Analytics Flow for Big Data





Big Data Stack



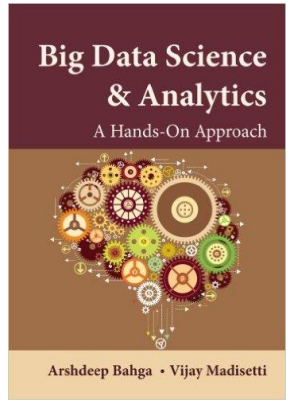
Big Data Stack

- **Raw Data Sources**
  - In any big data analytics application or platform, before the data is processed and analyzed, it must be captured from the raw data sources into the big data systems and frameworks.
- **Data Access Connectors**
  - The Data Access Connectors includes tools and frameworks for collecting and ingesting data from various sources into the big data storage and analytics frameworks. The choice of the data connector is driven by the type of the data source.
- **Data Storage**
  - The data storage block in the big data stack includes distributed filesystems and non-relational (NoSQL) databases, which store the data collected from the raw data sources using the data access connectors.

Big Data Stack

- **Batch Analytics**
  - The batch analytics block in the big data stack includes various frameworks which allow analysis of data in batches such as Hadoop-MapReduce, Pig, Oozie, Pig, Spark, Solr.
- **Real-time Analytics**
  - The real-time analytics block includes the Apache Storm and Spark Streaming frameworks.
- **Interactive Querying**
  - Interactive querying systems allow users to query data by writing statements in SQL-like languages.
- **Serving Databases, Web & Visualization Frameworks**
  - While the various analytics blocks process and analyze the data, the results are stored in serving databases for subsequent tasks of presentation and visualization. These serving databases allow the analyzed data to be queried and presented in the web applications.

Mapping Analytics Flow to Big Data Stack



Data Collection

Analysis Type	Framework (Mode)
Publish-Subscribe	Kafka, Kinesis
Source-Sink	Flume
SQL	Sqoop
Queues	SQS, RabbitMQ, ZeroMQ, RESTMQ
Custom Connectors	REST, WebSocket, MQTT

Data Preparation

Analysis Type	Framework
Data Cleaning	Open Refine
Data Wrangling	Open Refine DataWrangler
De-Duplication	Open Refine, Pig, Hive, Spark SQL
Normalization Sampling, Filtering	MapReduce, Pig, Hive, Spark SQL

Basic Statistics

Analysis Type	Framework (Mode)
Counts, Max, Min, Mean, Top-N, Distinct	Hadoop-MapReduce (Batch), Pig (Batch), Spark (Batch), Spark Streaming (Realtime), Spark SQL (Interactive), Hive (Integrative), Storm (Real-time)
Correlations	Hadoop-MapReduce (Batch), Spark Mlib (Batch)

Clustering

Analysis Type	Framework (Mode)
K-Means	Hadoop-MapReduce (Batch), Spark Mlib (Batch & Real-time) H2O (Batch)
DBSCAN	Spark (Batch)
Gaussian Mixture	Spark Mlib (Batch)
PIC	Spark Mlib (Batch)
LDA	Spark Mlib (Batch)

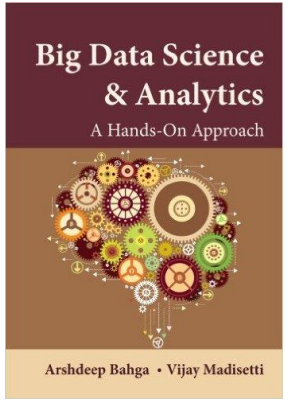
Classification

Analysis Type	Framework (Mode)
KNN	Spark Mlib (Batch , Realtime)
Decision Trees	Spark Mlib (Batch, Realtime)
Random Forest	Spark Mlib (Batch , Realtime), H2O (Batch)
SVM	Spark Mlib (Batch , Realtime)
Naïve Bayes	Spark Mlib (Batch, Realtime), H2O (Batch)
Deep Learning	H2O (Batch)

Regression

Analysis Type	Framework (Mode)
Linear Least Squares	Spark Mlib (Batch, Realtime)
Generalized Linear Model	H2O (Batch)
Stochastic Gradient Descent	Spark Mlib (Batch, Realtime)
Isotonic Regression	Spark Mlib (Batch, Realtime)

Mapping Analytics Flow to Big Data Stack



Graph Analytics

Analysis Type	Framework (Mode)
Graph Search	Spark GraphX (Batch)
Shortest-Path	Spark GraphX (Batch)
PageRank	Spark GraphX (Batch)
Triangle Counting	Spark GraphX (Batch)
Connected Components	Spark GraphX (Batch)

Time Series Analysis

Analysis Type	Framework (Mode)
Kalman Filter	Spark (Realtime)
Time Frequency Models	Spark (Realtime)
Outlier Detection	Storm (Realtime), Spark (Batch, Realtime)

Dimensionality Reduction

Analysis Type	Framework (Mode)
SVD	Spark Mlib (Batch)
PCA	Spark Mlib (Batch), H2O (Batch)

Recommendation

Analysis Type	Framework (Mode)
Item-bases Recommendation	Spark Mlib (Batch)
Collaborative Filtering	Spark Mlib (Batch)

Text Analysis

Analysis Type	Framework (Mode)
Categorization	Hadoop-MapReduce (Batch), Storm (Realtime), Spark (Batch, Realtime)
Summarization	Spark (Batch)
Sentiment Analysis	Storm (Realtime), Spark (Batch, Realtime)
Text Mining	Storm (Realtime), Spark (Batch, Realtime)

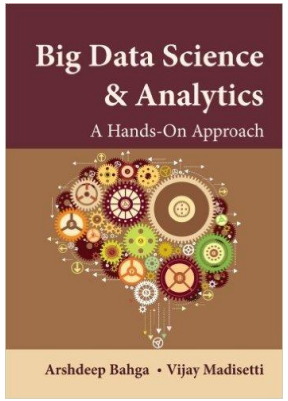
Pattern Mining

Analysis Type	Framework (Mode)
FP-Growth	Spark Mlib (Batch)
Association Rules	Spark Mlib (Batch)
PrefixSpan	Spark Mlib (Batch)

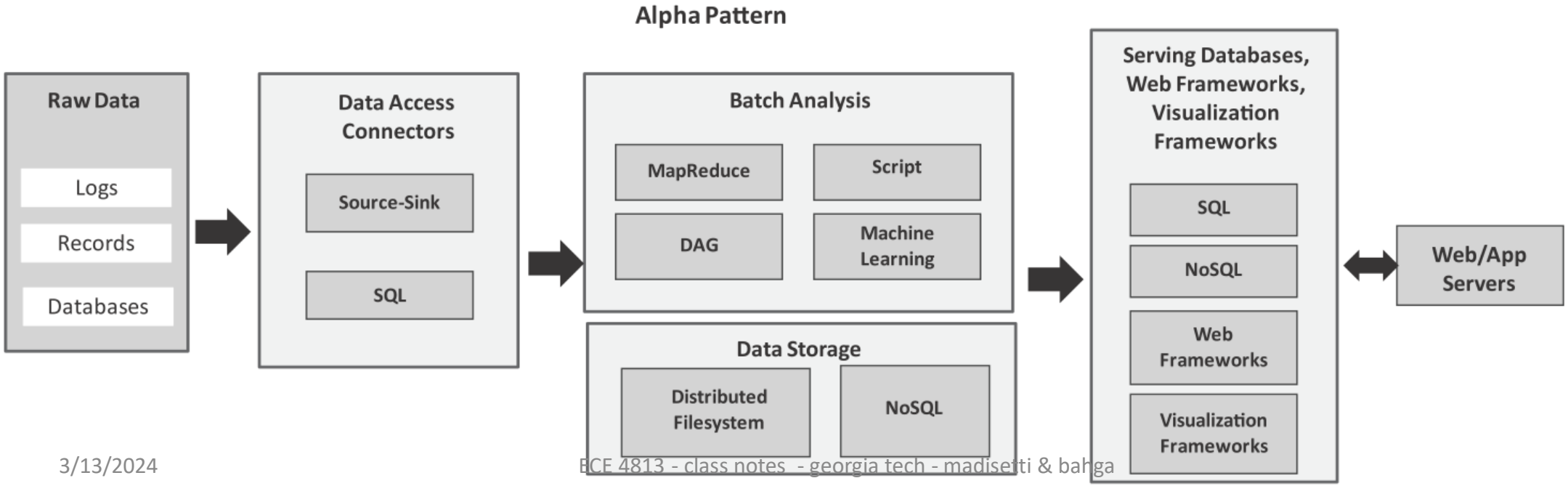
Visualization

Analysis Type	Framework (Mode)
Web Frameworks	Django, Flask
SQL Databases	MySQL
NoSQL Databases	Hbase, DynamoDB, Cassandra, MongoDB
Visualization Frameworks	Lightning, pyGal, Seaborn

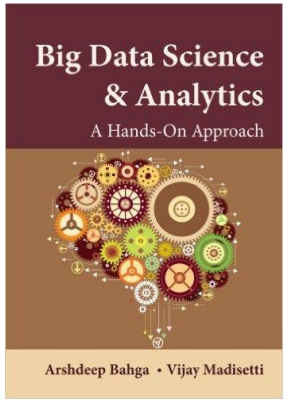
Analytics Patterns



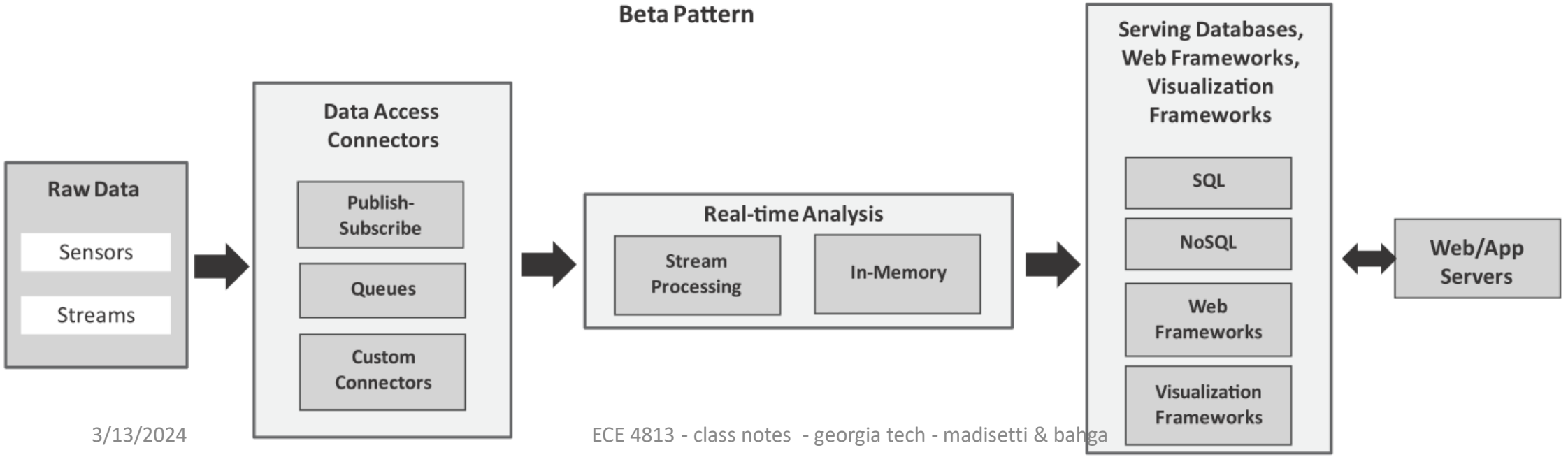
- **Alpha Pattern**
  - This pattern can be used for ingesting large volumes of data into a distributed filesystem (such as HDFS) or a NoSQL database (such as HBase) using source-sink connectors (such as Flume) and SQL connectors (such as Sqoop). After the data is moved to the stack, the data can be analyzed in batch mode with batch analysis frameworks including MapReduce (using Hadoop), scripting frameworks (such as Pig), distributed acyclic graph frameworks (such as Spark), machine learning frameworks (such as Spark MLlib). The analysis results are stored either in relational or non-relational databases.



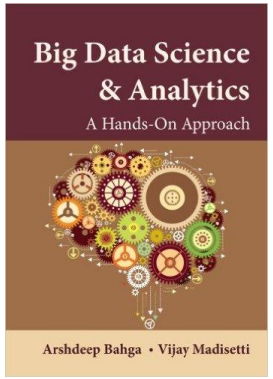
Analytics Patterns



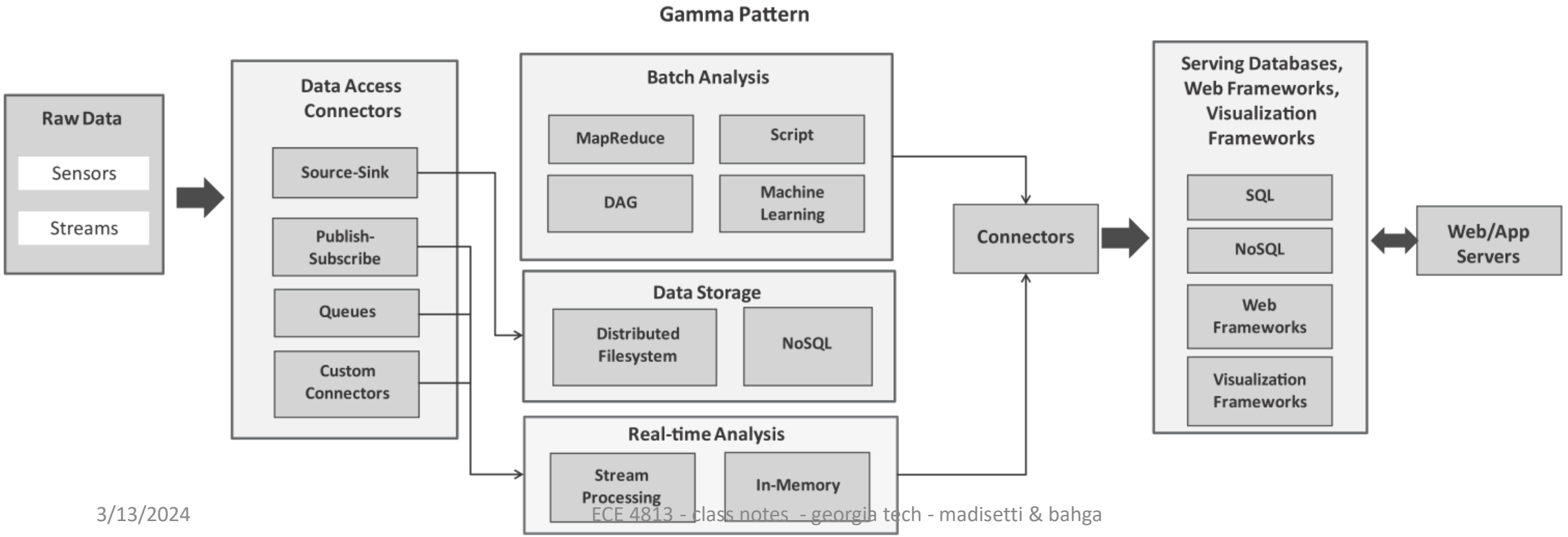
- **Beta Pattern:**
  - This pattern can be used for ingesting streaming data using publish-subscribe messaging frameworks, queues and custom connectors. For real-time analysis, we can use stream processing frameworks (such as Storm) or in-memory processing frameworks (such as Spark).



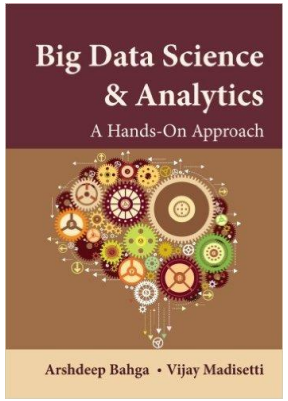
Analytics Patterns



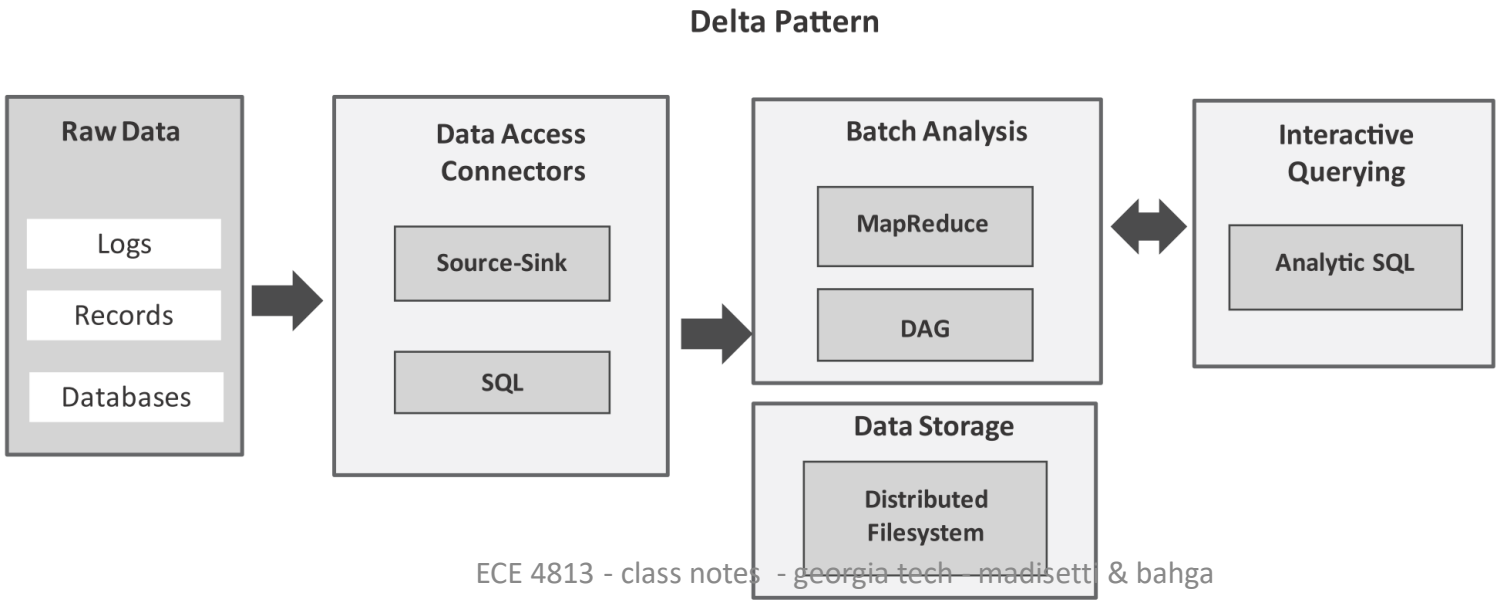
- **Gamma Pattern:**
  - This pattern is meant for ingesting streaming data into a big data stack and analyzing the data both in real-time and in batch modes.



Analytics Patterns



- **Delta Pattern:**
  - This pattern uses, source-sink connectors (such as Flume) or SQL connectors (such as Sqoop) to ingest bulk data into the big data stack. After the data is moved to a distributed filesystem, you can use interactive querying frameworks (such as Hive or Spark SQL) for querying data with SQL-like queries in an interactive mode.



Mapping of blocks in analytics patterns to AWS and Azure services

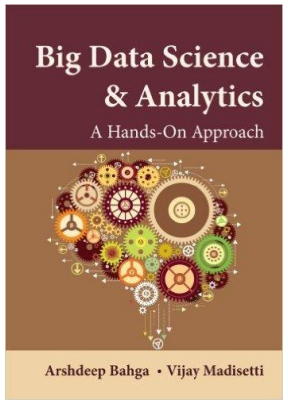
Data Access Connector	AWS Service	Azure Service
Publish-Subscribe	AWS Kinesis	Azure Event Hubs
Source-Sink	Flume on EMR	Flume on HDInsight
SQL	Sqoop on EMR	Sqoop on HDInsight
Queues	AWS SQS	Azure Queue Service
Custom Connectors	AWS IoT	Azure IoT Hub

Data Storage	AWS Service	Azure Service
Distributed Filesystem	HDFS on AWS EMR	HDFS on Azure HDInsight
NoSQL	AWS DynamoDB	Azure DocumentDB

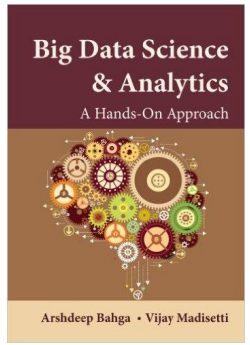
Batch Analysis	AWS Service	Azure Service
MapReduce	Hadoop on AWS EMR	Hadoop on Azure HDInsight
Script	Pig on AWS EMR	Pig on Azure HDInsight
DAG	Spark on AWS EMR	Spark on Azure HDInsight
Machine Learning	Spark MLlib on AWS EMR, AWS Machine Learning	Spark MLlib on Azure HDInsight, Azure Machine Learning

Real-time Analysis	AWS Service	Azure Service
Stream Processing	Storm cluster on AWS EC2	Storm on Azure HDInsight, Azure Stream Analytics
In-Memory Processing	Spark on AWS EMR	Spark on Azure HDInsight

Serving Databases, Web & Visualization Frameworks	AWS Service	Azure Service
SQL	AWS RDS	Azure SQL DB
NoSQL	AWS DynamoDB	Azure DocumentDB
Web Framework	Django on AWS EC2 instance	Django on Azure Virtual Machines instance
Visualization Framework	Lightning, Pygal, Seaborn, on AWS Virtual Machines instance	Lightning, Pygal, Seaborn, on Azure Virtual Machines instance, Azure Power BI



Complexity Levels for Analytics Patterns



Levels	Alpha	Beta	Gamma	Delta
Security	• **Apache Ranger**: Authorization, authentication, auditing, data encryption, security administration • **Apache Knox Gateway**: REST API gateway, LDAP and Active Directory Authentication, Federation/SSO, Authorization, Auditing			
Fault Tolerance	• Distributed, fault tolerant and highly available architectures • Data replication, automatic failovers			
Scalability	Scalable to thousands of nodes that can store and process several Petabytes of data	Scalable to thousands of nodes that can process very high throughput streaming data	Scalable to thousands of nodes that can process very high throughput streaming data	Scalable to thousands of nodes that can store and process several Petabytes of data
Performance	Process large volumes of data with within timescales of few minutes	Process streaming data on the timescales of few milliseconds to seconds	Combines batch and real-time processing for streaming data, with timescales of milliseconds to seconds for real-time, and few minutes for batch	Interactively query large volumes of data with within timescales of few milliseconds to seconds
Functionality	Batch Processing	Real time processing	Batch & Real-time processing	Interactive querying

Analytics Patterns

Nathan Marz
@nathanmarz

I'm working on a new kind of stream processing system. If this sounds interesting to you, ping me. I want to learn your use cases.

2:17 PM - Dec 14, 2010

2 13

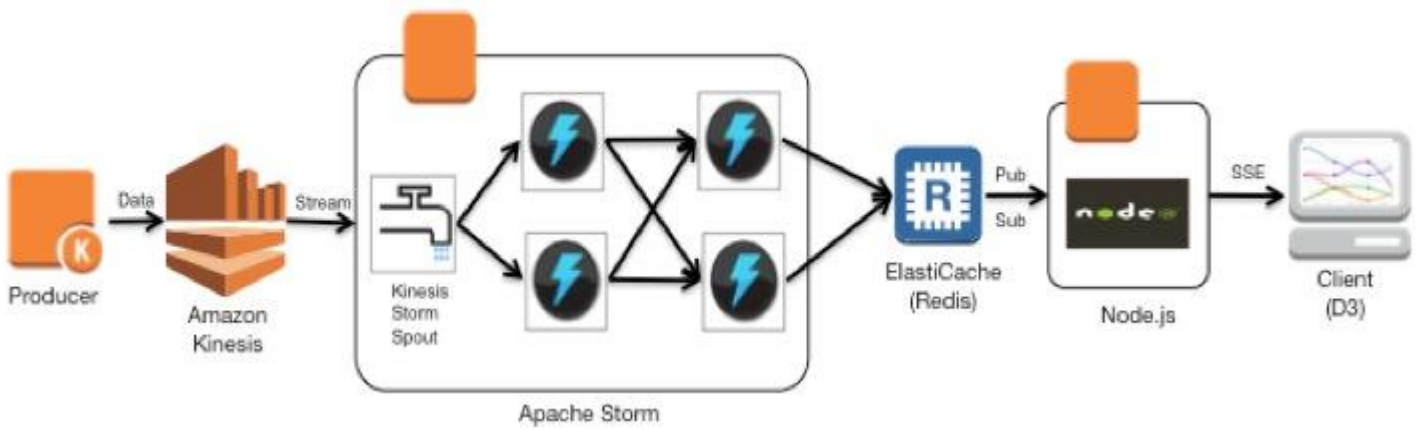
Creator of Apache Storm

Menu aws Products Solutions Pricing Getting Started Documentation Software Support Customers Partners Enterprises Startups Public Sec

RELATED POSTS

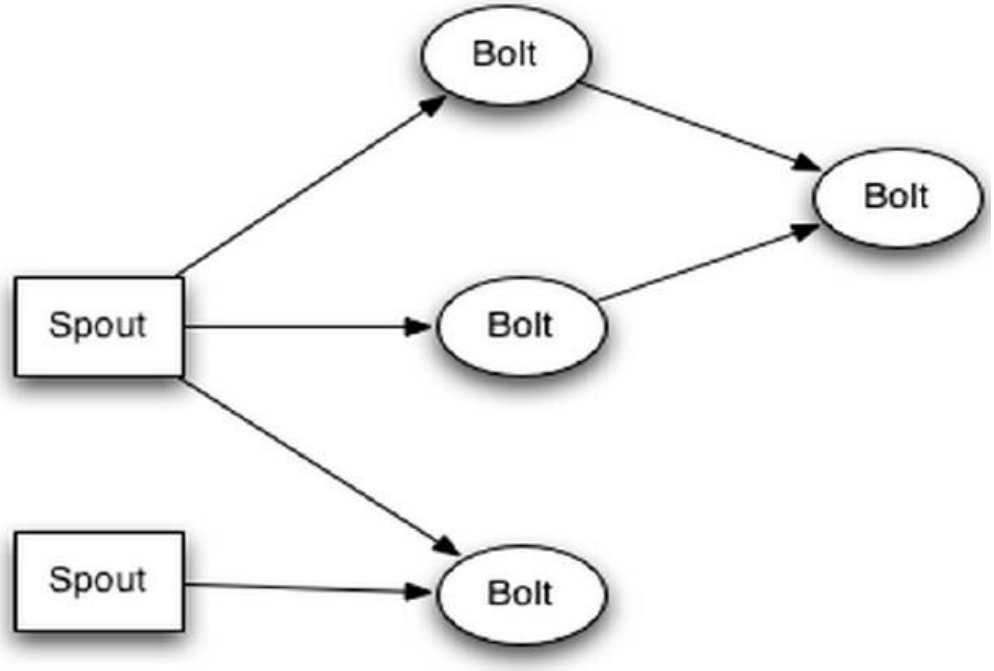
New – AWS PrivateLink for AWS Services: Kinesis, Service Catalog, EC2 Systems Manager, Amazon EC2 APIs, and ELB APIs in your VPC	AWS CloudFormation Feature Updates: Support for Amazon Athena and Coverage Updates for Amazon S3, Amazon RDS, Amazon Kinesis and Amazon CloudWatch	Preprocessing Data in Amazon Kinesis Analytics with AWS Lambda	AWS CloudFormation Supports Amazon Kinesis Analytics Applications	New: Server-Side Encryption for Amazon Kinesis Streams	Perform Streami and Am

https://aws.amazon.com/blogs/big-data/implement-a-real-time-sliding-window-application-using-amazon-kinesis-and-apache-storm/



Storm Processing Flow

- **Topology**
  - Graph of computation – can have cycles
  - Network of Spouts and Bolts
  - Spouts and bolts execute as many tasks across the cluster



http://storm.apache.org/releases/current/Tutorial.html

Use Case - Trending topics on Twitter

9:21-9:29 PM

9:30 PM

- Meryl Streep
- Moonlight
- #HiddenFences
- Casey Affleck
- Donald Glover
- #ThingsGhostsProbablyDo
- Emma Stone
- Brad Pitt
- #ILaughAboutItBecause
- #ThingsNerdsSay

Top keywords

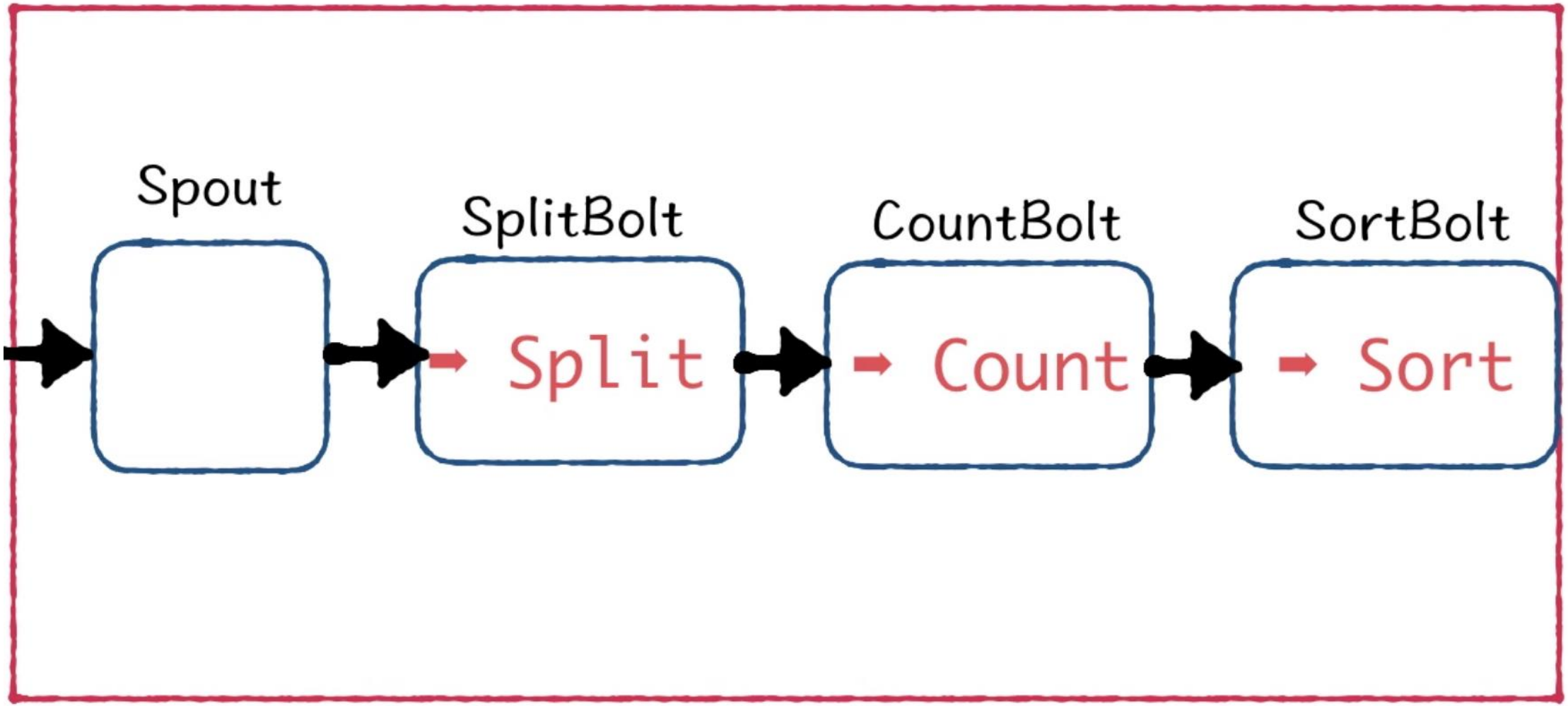
Algorithm Flow

MerylStreep 15000
GoldenGlobes 10000
CarrieFisher 2000

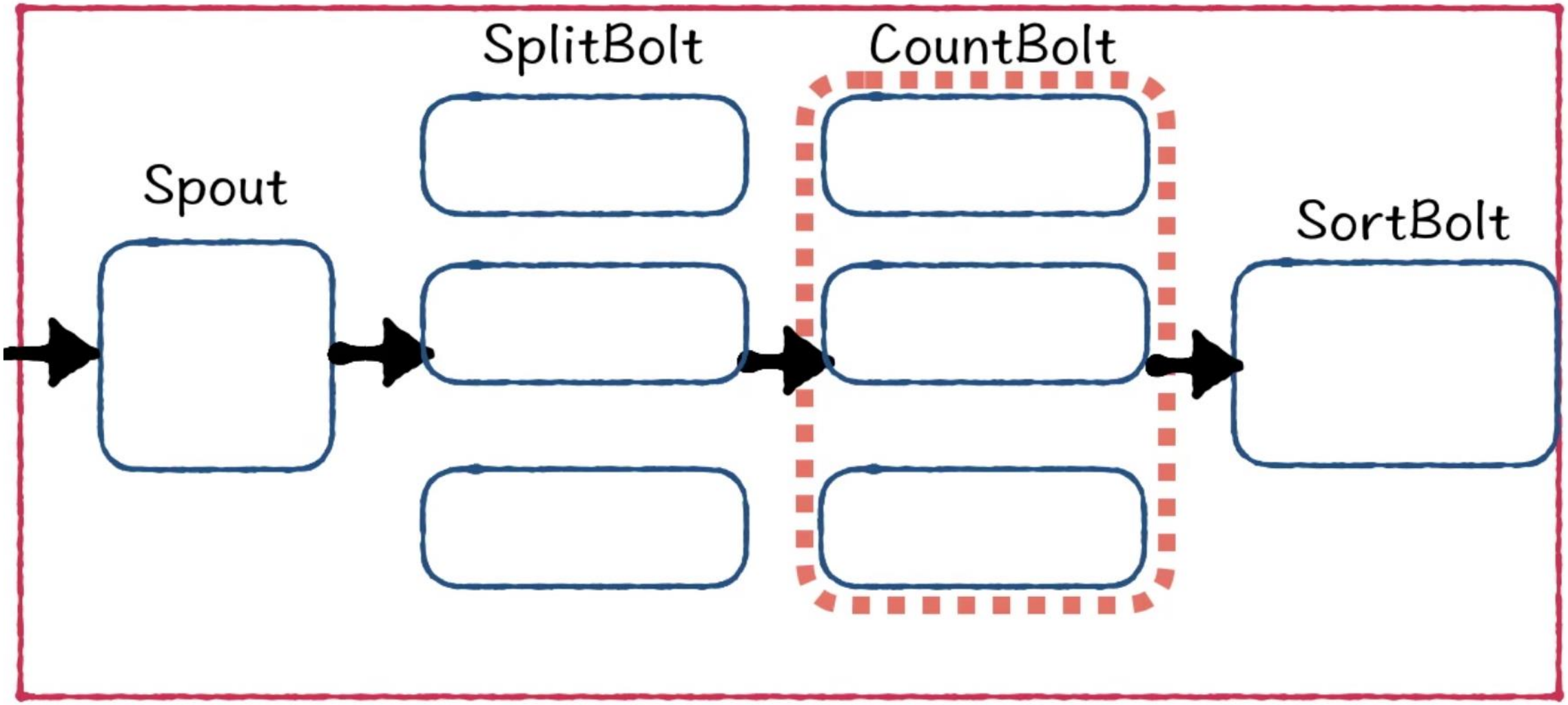
App

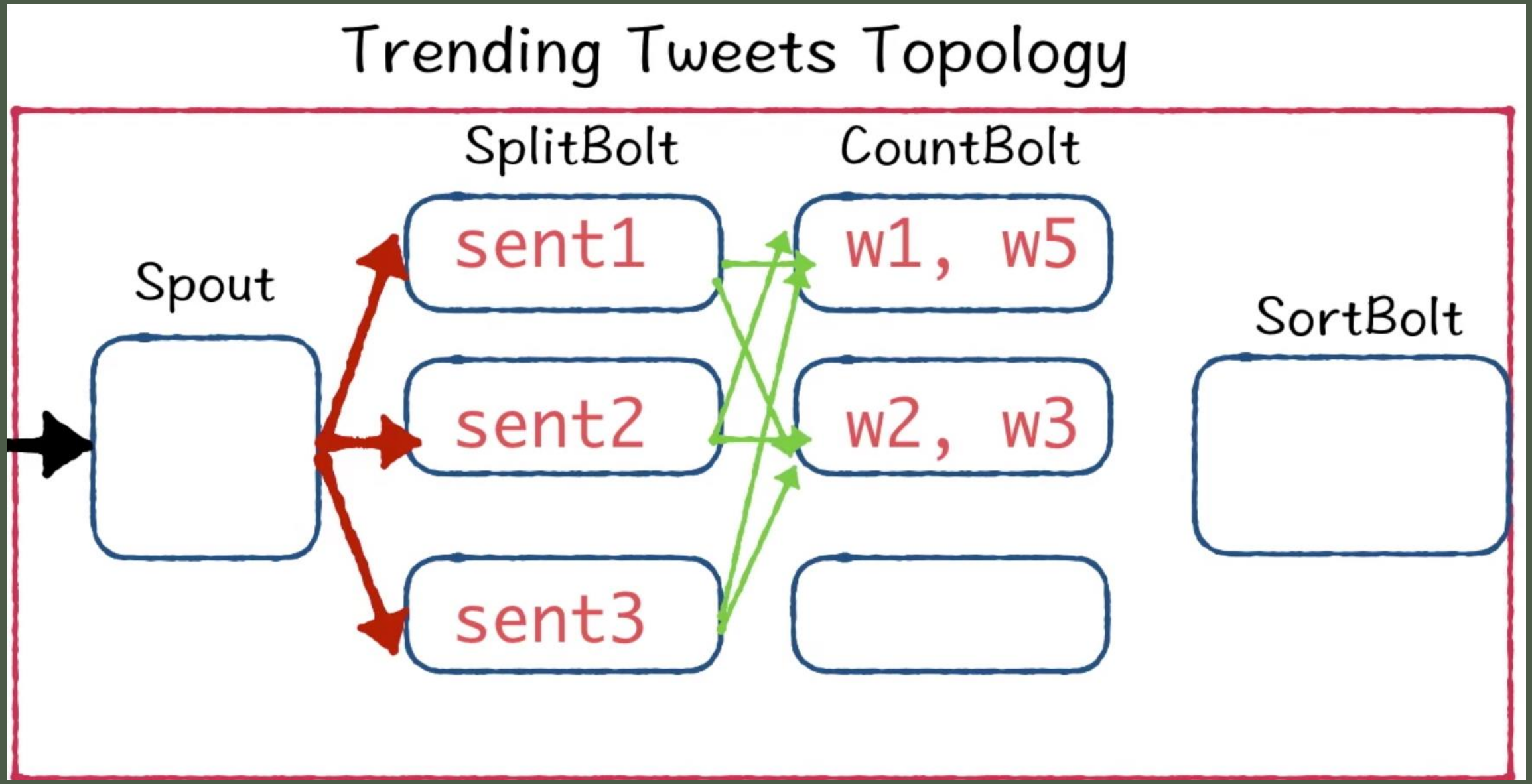
- Split tweets into words
- Find counts for each word in last 10 mins
- Sort words by count

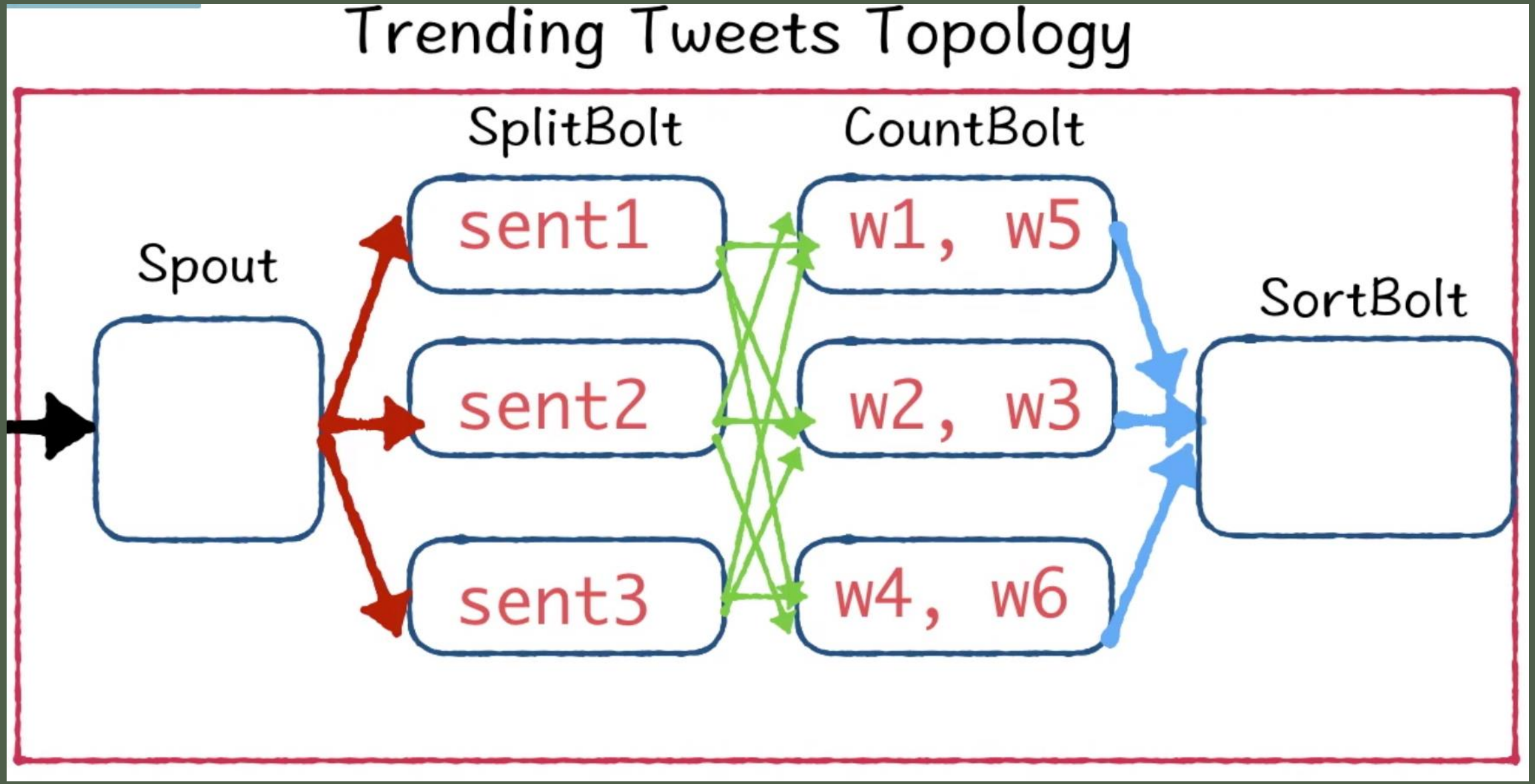
Trending Tweets Topology



Trending Tweets Topology



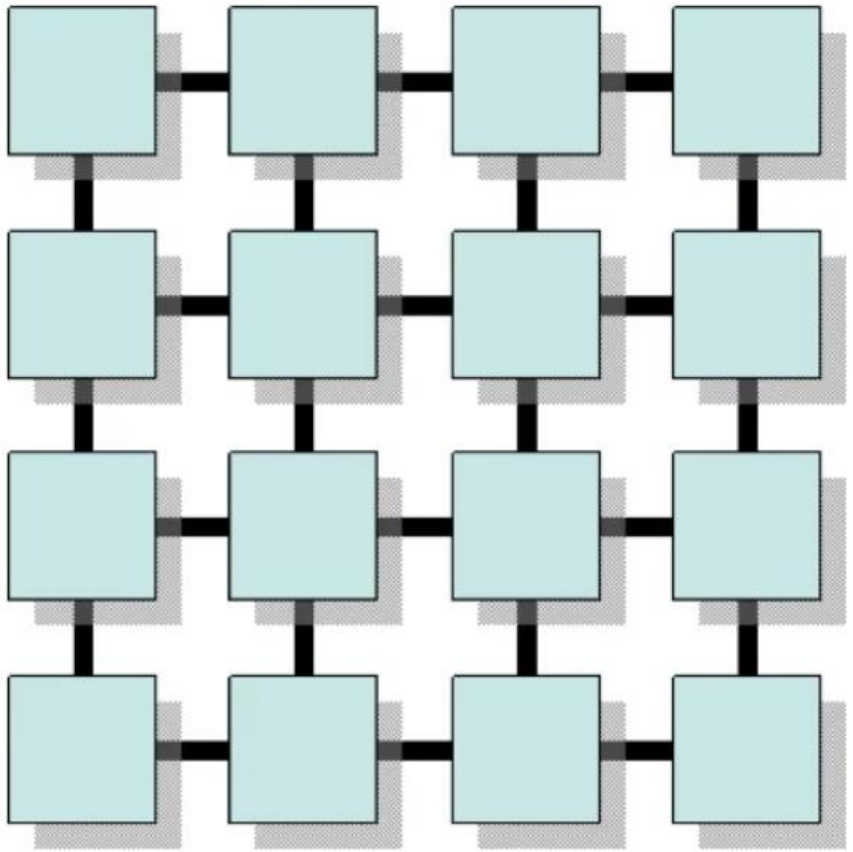

Trending Tweets Topology
SplitBolt
CountBolt
Spout
sent1
sent2
sent3
w1, w5
w2, w3
SortBolt


Trending Tweets Topology
SplitBolt
CountBolt
Spout
sent1
sent2
sent3
w1, w5
w2, w3
w4, w6
SortBolt

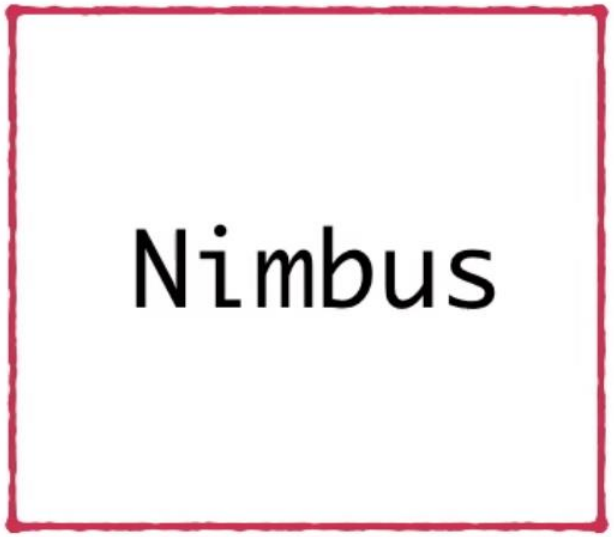
Apache Storm

The processes can run on different machines

Remote Mode

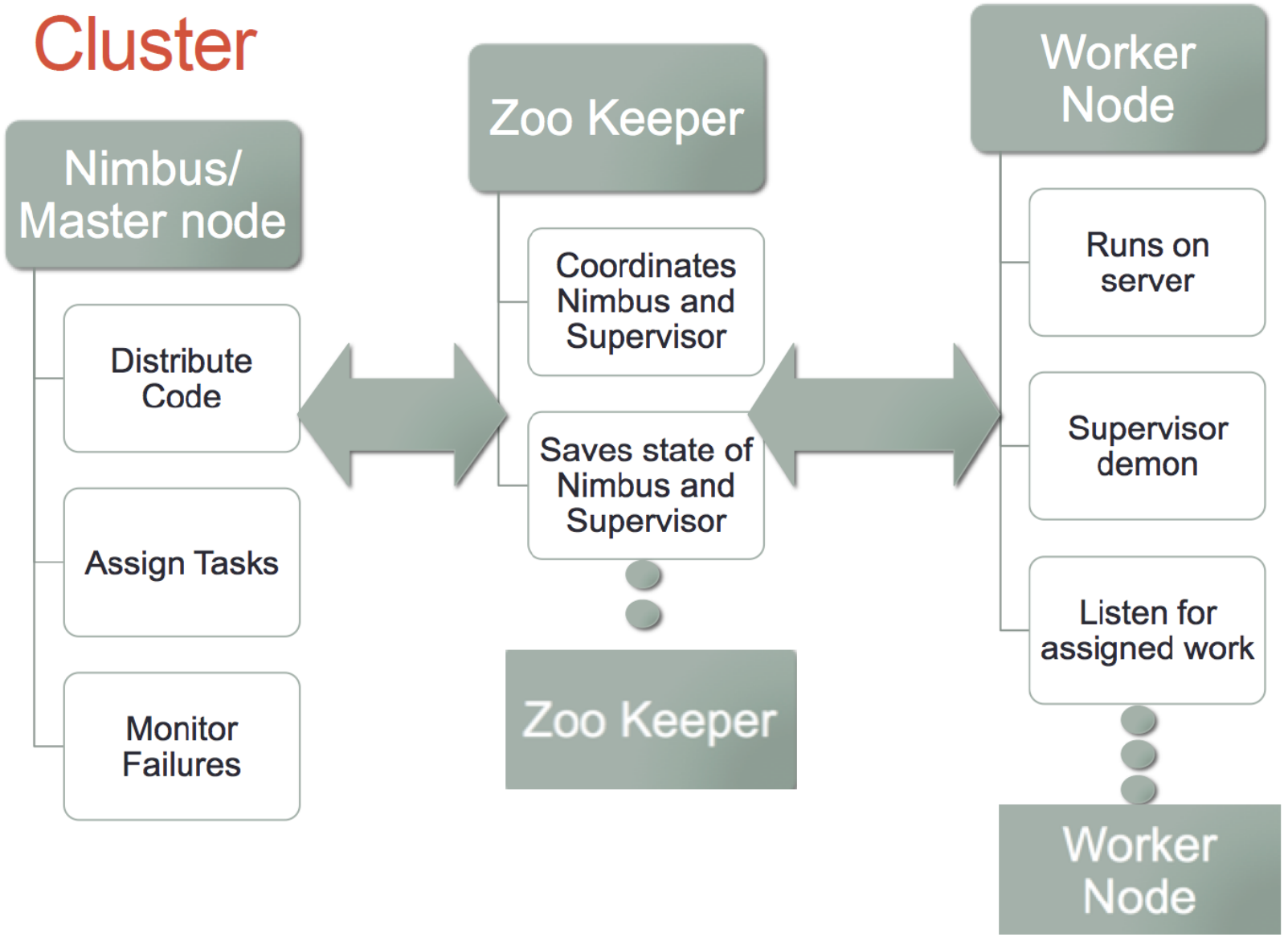


Apache Storm

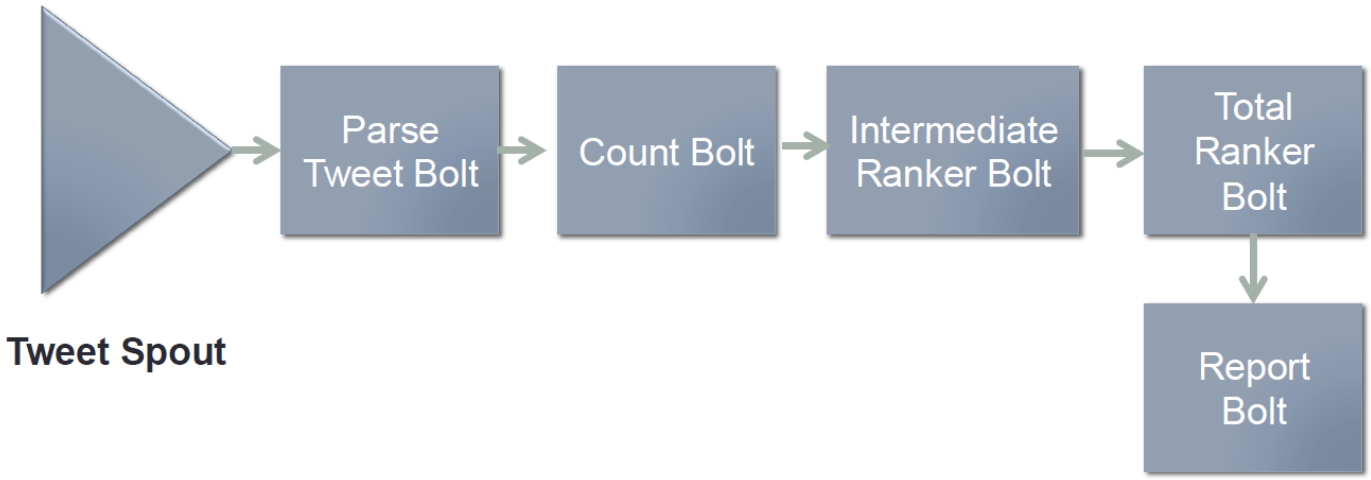


The Nimbus will start up a bunch of processes for Spouts and Bolts

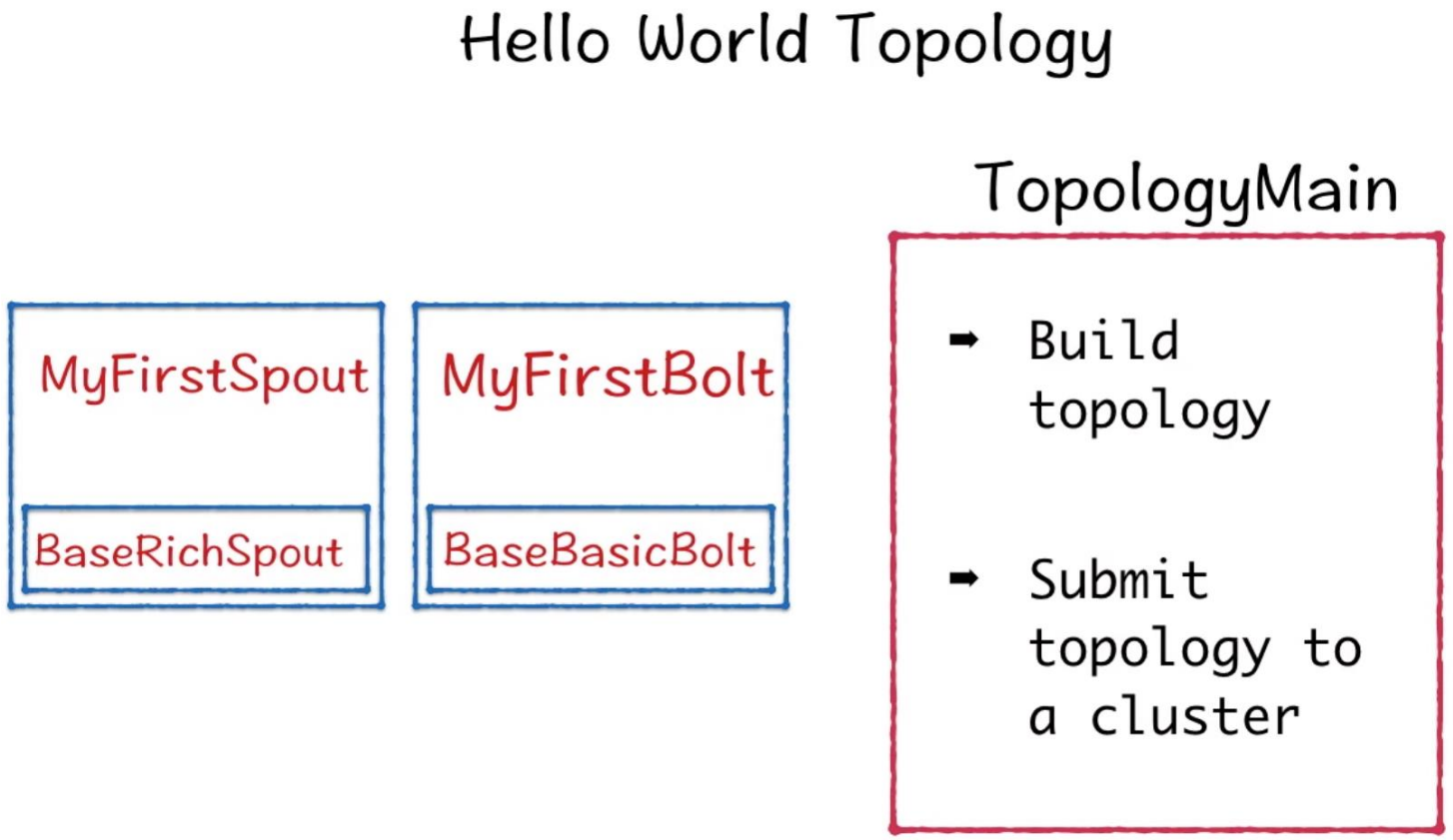
Storm Distributed Processing Architecture



Generic Flow



Use Case 1


Hello World Topology
MyFirstSpout
BaseRichSpout
MyFirstBolt
BaseBasicBolt
TopologyMain
Build topology
Submit topology to a cluster

Spout Architecture

D:\Dropbox (VijayMadisetti)\VKM-Files-Private\GT\Cloud-Computing-Fall2017\Storm\SourceCode\SourceCode1-24\src\main\java\examples\ex1_HelloWorld\myFirstSpout.java - Sublime Text

Edit Selection Find View Goto Tools Project Preferences Help

myFirstBolt.java | myFirstSpout.java | TopologyMain.java

```java
package examples.ex1_HelloWorld;


import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;


import java.util.Map;


public class myFirstSpout extend aseRichSpout {

    private SpoutOutputColl or collector;

    private Integer i ;

    public void open(Map conf, TopologyContext context,
                     SpoutOutputCollector collector) {

        this.collector = collector;
    }
    public void nextTuple() {
```

Big Five Methods

1

2

- **open** : Provides spout with an environment to execute.

  open(Map conf, TopologyContext context, SpoutOutputCollector collector)

- **nextTuple** : Emits the generated data.

  nextTuple()

- **close** : call when a spout is going to close.

  close()

- **declareOutputFields** : Declares the output schema of the tuple.

  declareOutputFields(OutputFieldsDeclarer declarer)

- **ack** : Acknowledges that a specific tuple is processed

  ack(Object msgId)

- **fail** : Specifies that a specific tuple is not processed

3

```java
    public void nextTuple() {

        this.collector.emit(new Values(this.i));
        this.i=this.i+1;

    }

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

    public void ack(Object msgId) {}


    public void fail(Object msgId) {}

```

4

5

Bolt Architecture

- Prepare : Provides the bolt with an environment to execute.
  prepare(Map conf, TopologyContext context, OutputCollector collector)
- Execute : Process a single tuple of input.
  execute(Tuple tuple)
- Cleanup : When a bolt is going to shutdown.
  cleanup()
- declareOutputFields : Schema of the tuple.
  declareOutputFields(OutputFieldsDeclarer declarer)

D:\Dropbox (VijayMadisetti)\VKM-Files-Private\GT\Cloud-Computing-Fall2017\Storm\SourceCode\SourceCode1-24\src\main\java\examples\ex1_HelloWorld\myFirstBolt.java - Sublime Text

Edit Selection Find View Goto Tools Project Preferences Help

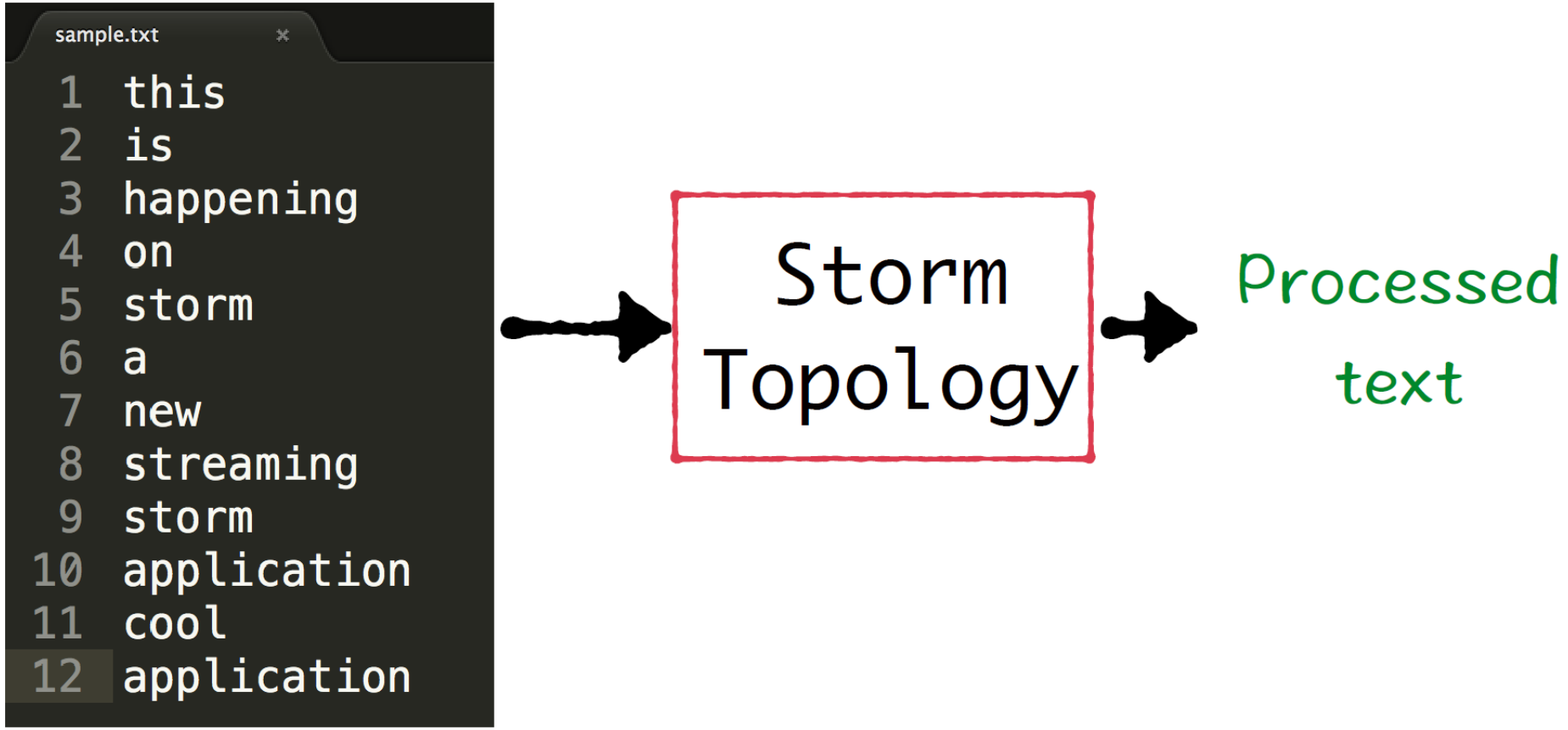
myFirstBolt.java | myFirstSpout.java | TopologyMain.java

```java
package examples.ex1_HelloWorld;


import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class myFirstBolt extends BaseBasicBolt {
    public void execute(Tuple input,BasicOutputCollector collector) {
        collector.emit(new Values(input.getInteger(0)*2));
    }

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

    public void cleanup() {}
}
```

Topology of storm

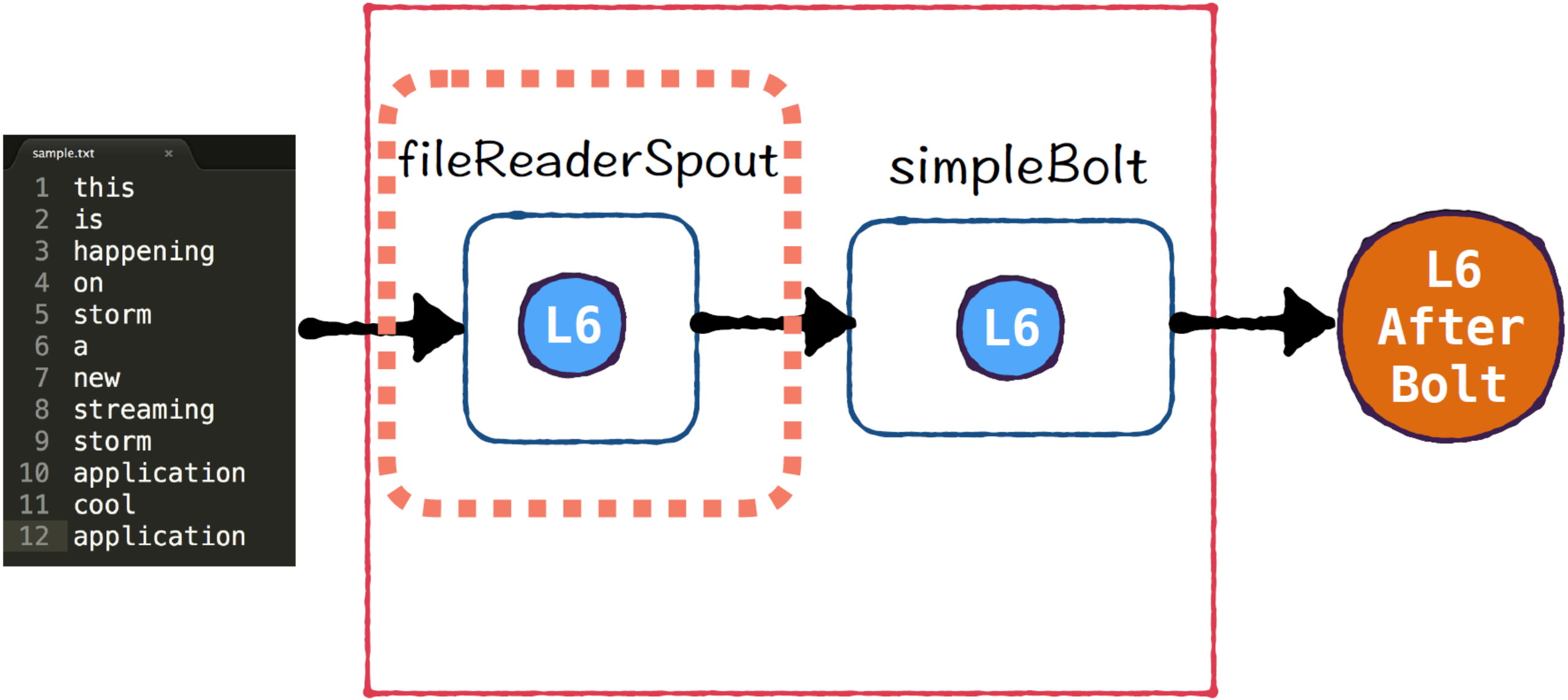
```java
package examples.ex1_HelloWorld;


import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.topology.TopologyBuilder;


public class TopologyMain {
    public static void main(String[] args) throws InterruptedException {

        //Build Topology
        TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("My-First-Spout", new myFirstSpout());
        builder.setBolt("My-First-Bolt", new myFirstBolt()).shuffleGrouping("My-First-Spout");

        //Configuration
        Config conf = new Config();
        conf.setDebug(true);


        //Submit Topology to cluster
        LocalCluster cluster = new LocalCluster();
        try{
            cluster.submitTopology("My-First-Topology", conf, builder.createTopology());
            Thread.sleep(1000);}

        finally{
            cluster.shutdown();}
    }
```

Reading from a File



Use Case 2

Reading from a File



Bolt Architecture

myFirstBolt.java | myFirstSpout.java | TopologyMain.java | fileReaderSpout.java | simpleBolt.java

```java
package examples.ex2_readFromFile;

import org.apache.storm.task.OutputCollector;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class simpleBolt extends BaseBasicBolt {

    public void cleanup() {}


    public void execute(Tuple input,BasicOutputCollector collector) {
        collector.emit(new Values(input.getString(0)+" After Bolt"));
    }

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

```java
package examples.ex2_readFromFile;


import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class fileReaderSpout extends BaseRichSpout {

    private SpoutOutputCollector collector;
    private boolean completed = false;
    private FileReader fileReader;
    private String str;
    private BufferedReader reader ;
```

```java

    public void open(Map conf, TopologyContext context,
                     SpoutOutputCollector collector) {
        try {
            this.fileReader = new FileReader(conf.get("fileToRead").toString());
        } catch (FileNotFoundException e) {
            throw new RuntimeException("Error reading file ["+conf.get("wordFile")+"]");
        }

        this.collector = collector;
        this.reader =  new BufferedReader(fileReader);


    }
```

```java

    public void nextTuple() {

        if (!completed) {
          try {

                this.str = reader.readLine();
                if (this.str != null) {

                    this.collector.emit(new Values(str));
                } else {
                    completed = true;
                    fileReader.close();;
                }

            } catch (Exception e) {
                throw new RuntimeException("Error reading tuple", e);
            }
        }
    }
```

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

    public void ack(Object msgId) {}


    public void fail(Object msgId) {}
}
```

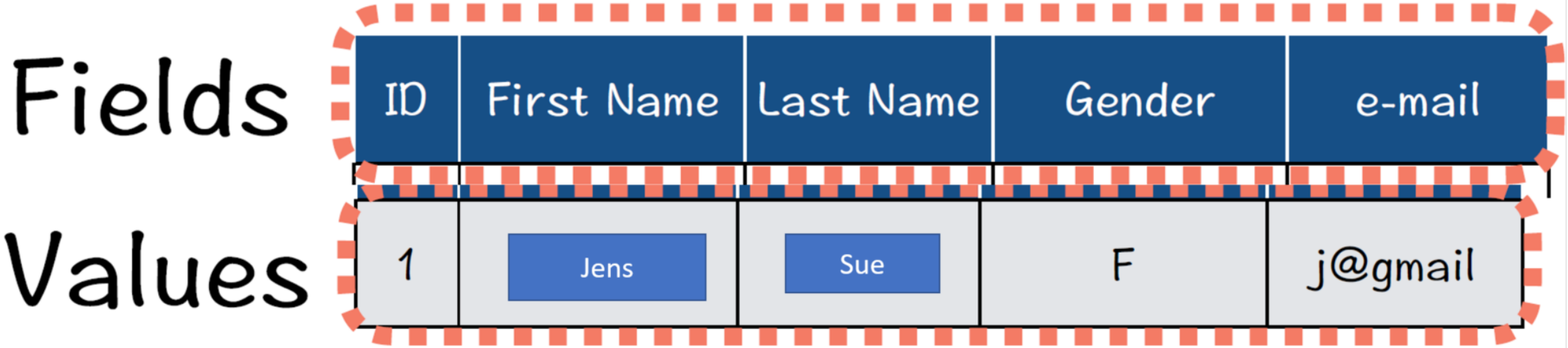
Spout Architecture

Storm Topology

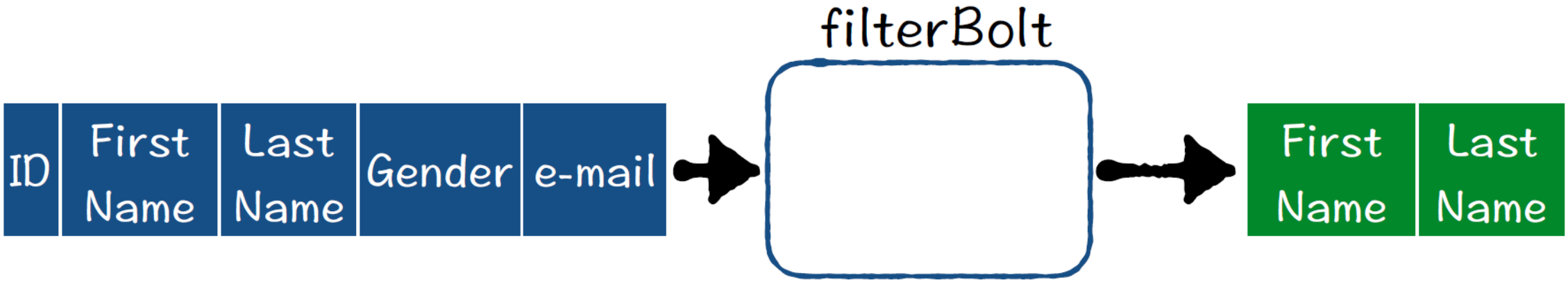
myFirstBolt.java | myFirstSpout.java | TopologyMain.java — ex1_HelloWorld | fileReaderSpout.java | simpleBolt.java | TopologyMain.java — ex2_readFromFile

```java
package examples.ex2_readFromFile;


import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.topology.TopologyBuilder;


public class TopologyMain {
    public static void main(String[] args) throws InterruptedException {

        //Topology definition
        TopologyBuilder builder = new TopologyBuilder();
        builder.setSpout("File-Reader-Spout", new fileReaderSpout());
        builder.setBolt("Simple-Bolt", new simpleBolt()).shuffleGrouping("File-Reader-Spout");

        //Configuration
        Config conf = new Config();
        conf.setDebug(true);
        conf.put("fileToRead", "/Users/vkm/Desktop/sample.txt");

        LocalCluster cluster = new LocalCluster();
        try{cluster.submitTopology("File-Reader-Topology", conf, builder.createTopology());
        Thread.sleep(1000);}
       finally{
        cluster.shutdown();}
    }
```

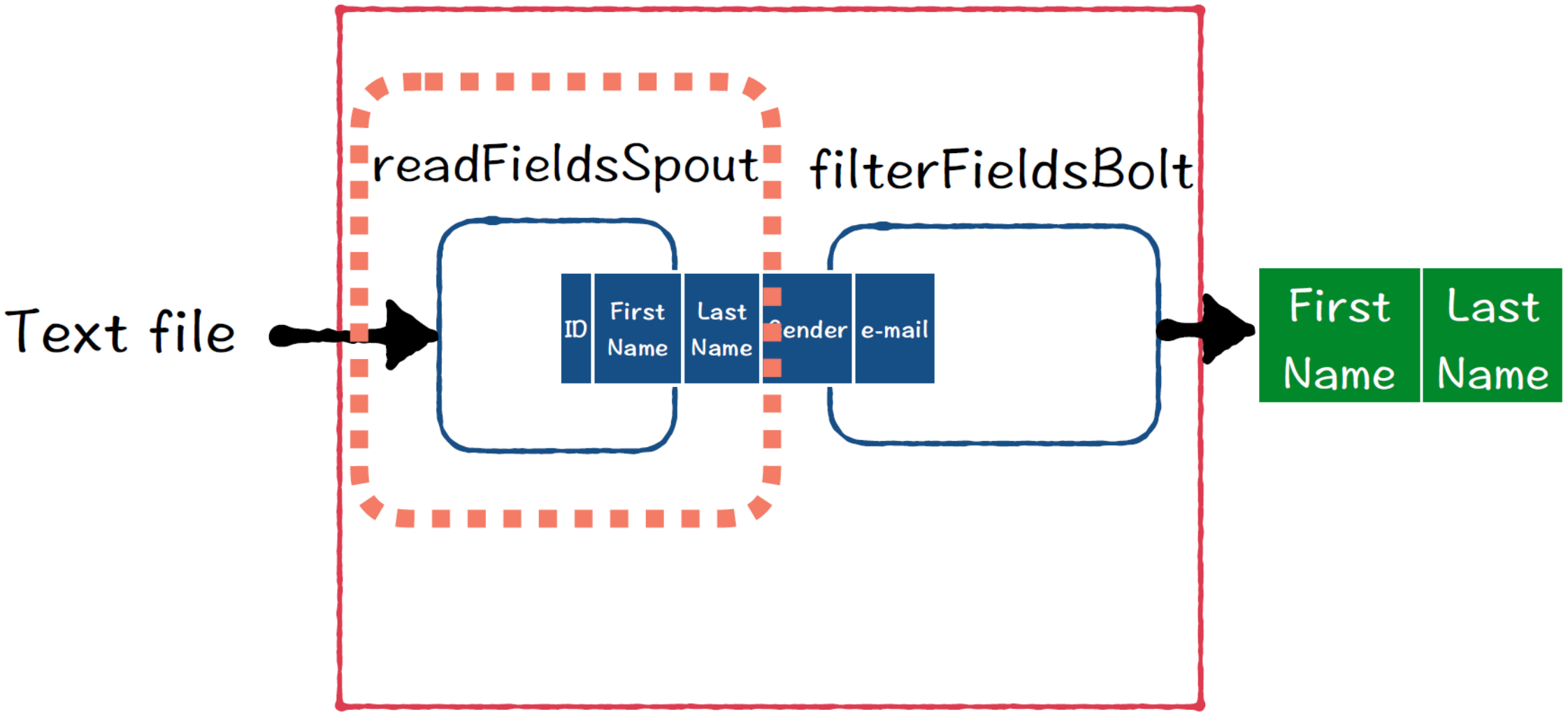
Use Case 3

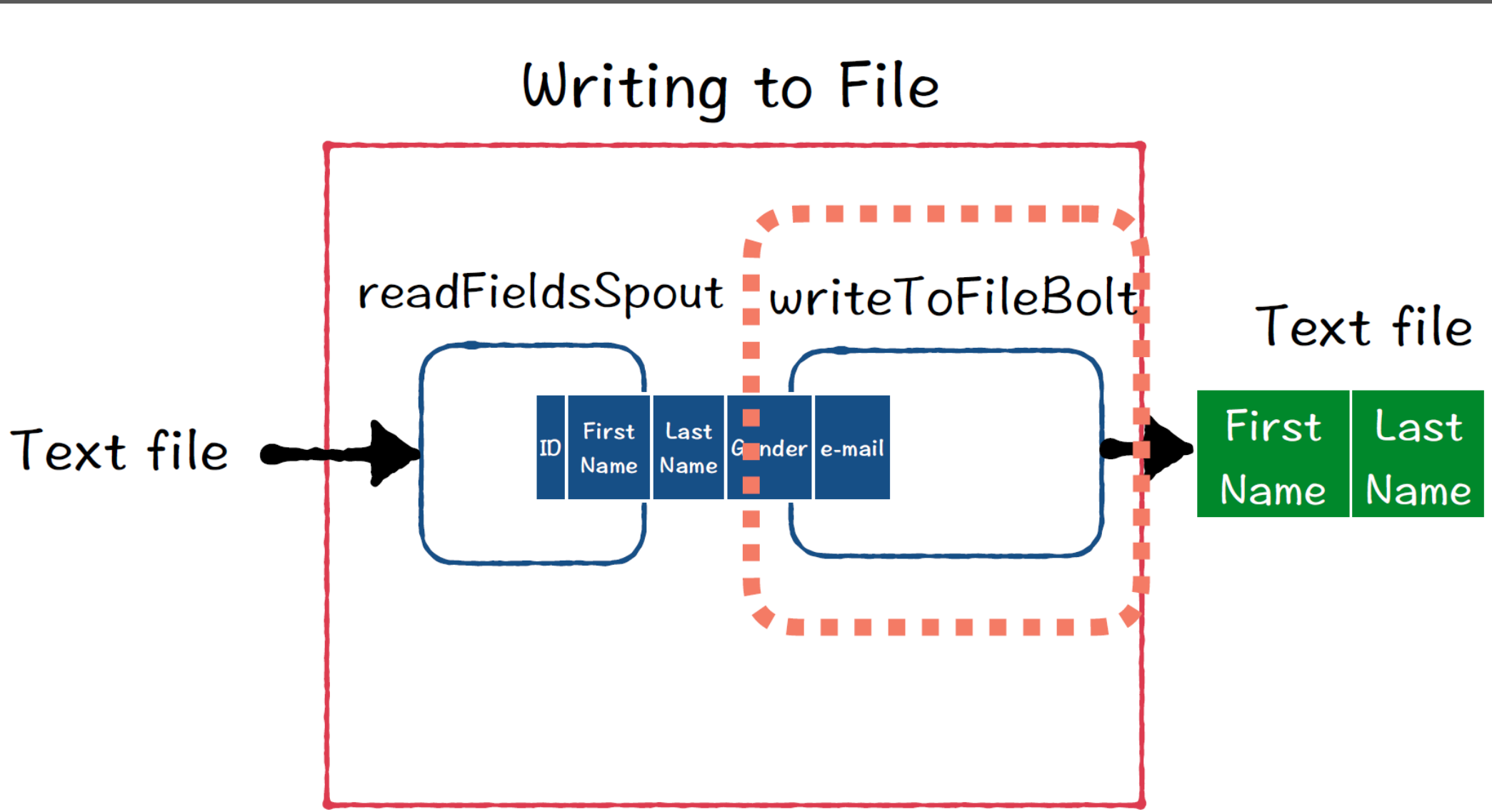

Fields
ID
First Name
Last Name
Gender
e-mail
Values
1
Jens
Sue
F
j@gmail

Output fields are fixed for a Storm Component



Extracting Fields




Writing to File
readFieldsSpout
writeToFileBolt
Text file
ID
First Name
Last Name
Gender
e-mail
Text file
First Name
Last Name

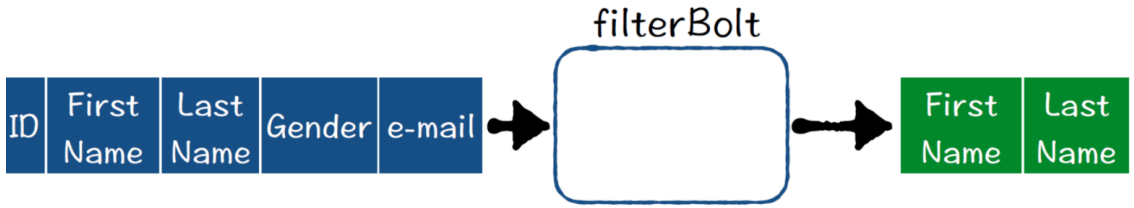
Portion of Storm Spout, readFieldsSpout, Generating Fields & Values

Two of the big Five

```
43
44      public void nextTuple() {
45
46          if (!completed) {
47              try {
48
49                  this.str = reader.readLine();
50                  if (this.str != null) {
51
52                      this.collector.emit(new Values(str.split(",")));
53                  } else {
54                      completed = true;
55                      fileReader.close();
56                  }
57
58              } catch (Exception e) {
59                  throw new RuntimeException("Error reading tuple", e);
60              }
61          }
62      }
63
64      public void declareOutputFields(OutputFieldsDeclarer declarer) {
65          declarer.declare(new Fields("id","first_name","last_name","gender","email"));
66      }
67
```

filterBolt for Filtering Fields from Input

```java
package examples.ex3_accessingFields;

import org.apache.storm.task.OutputCollector;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;

public class filterFieldsBolt extends BaseBasicBolt {

    public void cleanup() {}


    public void execute(Tuple input,BasicOutputCollector collector) {

    String firstName = input.getStringByField("first_name");
    String lastName = input.getString(2)    ;

        collector.emit(new Values(firstName,lastName));
    }

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



Apache Storm Support for Multiple Languages

See Lab 8

- http://storm.apache.org/releases/current/Multilang-protocol.html
- https://docs.microsoft.com/en-us/azure/hdinsight/storm/apache-storm-develop-python-topology

Defining Bolts in other languages

Bolts can be defined in any language. Bolts written in another language are executed as subprocesses, and Storm communicates with those subprocesses with JSON messages over stdin/stdout. The communication protocol just requires an ~100 line adapter library, and Storm ships with adapter libraries for Ruby, Python, and Fancy.

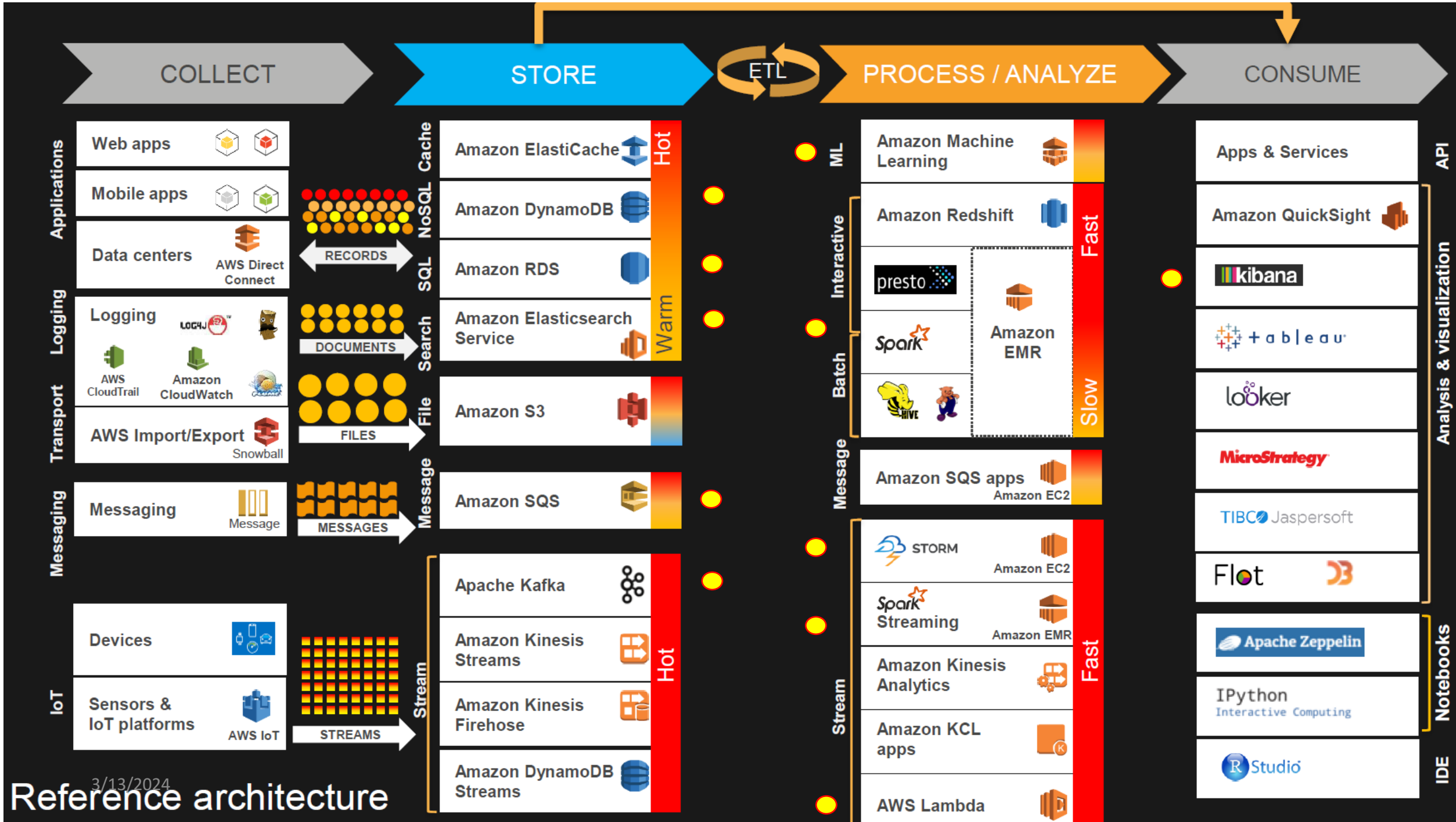
Here's the definition of the `SplitSentence` bolt from `WordCountTopology`:

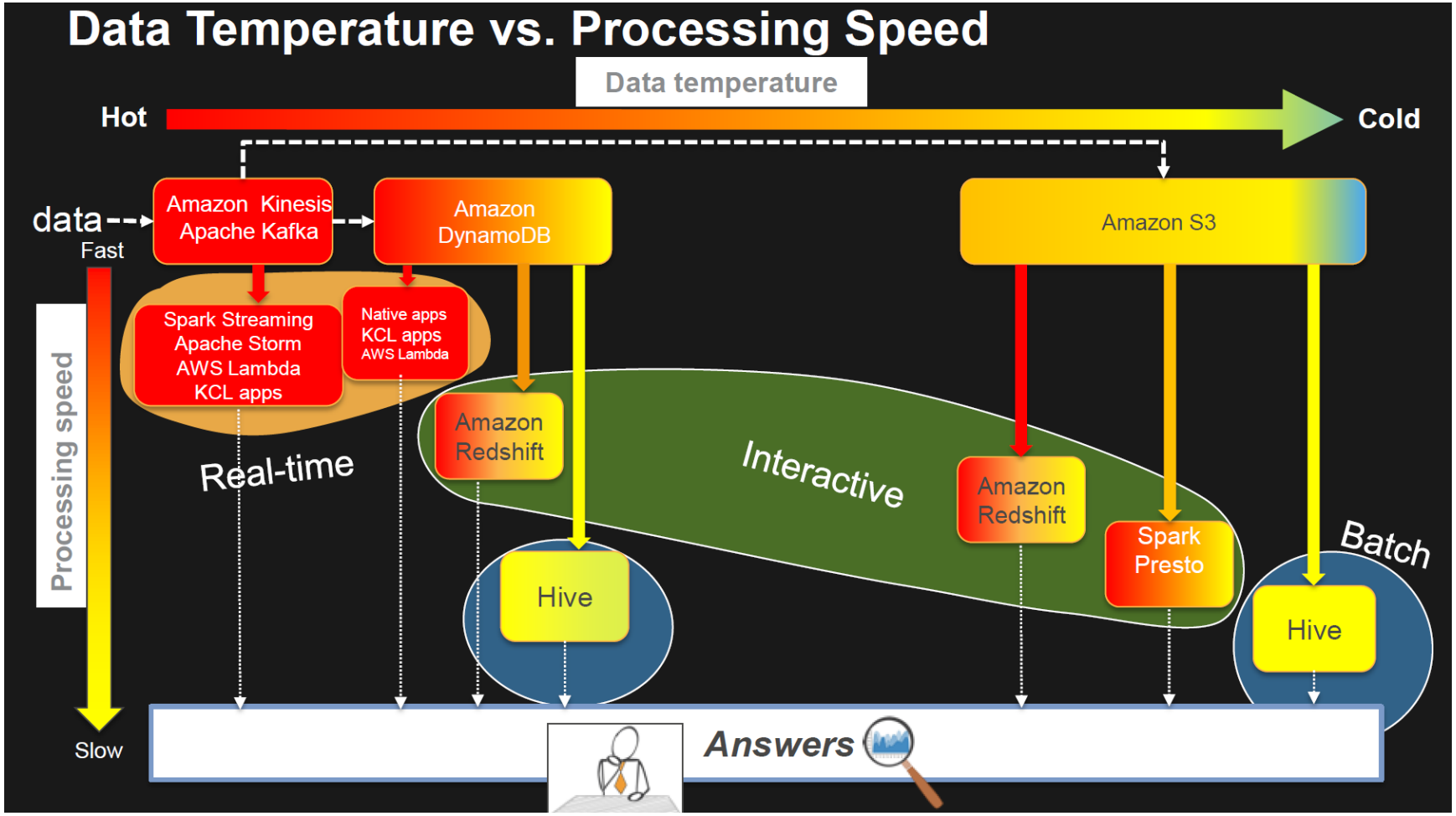
```
public static class SplitSentence extends ShellBolt implements IRichBolt {
    public SplitSentence() {
        super("python", "splitsentence.py");
    }

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

`SplitSentence` overrides `ShellBolt` and declares it as running using `python` with the arguments `splitsentence.py`. Here's the implementation of `splitsentence.py`:

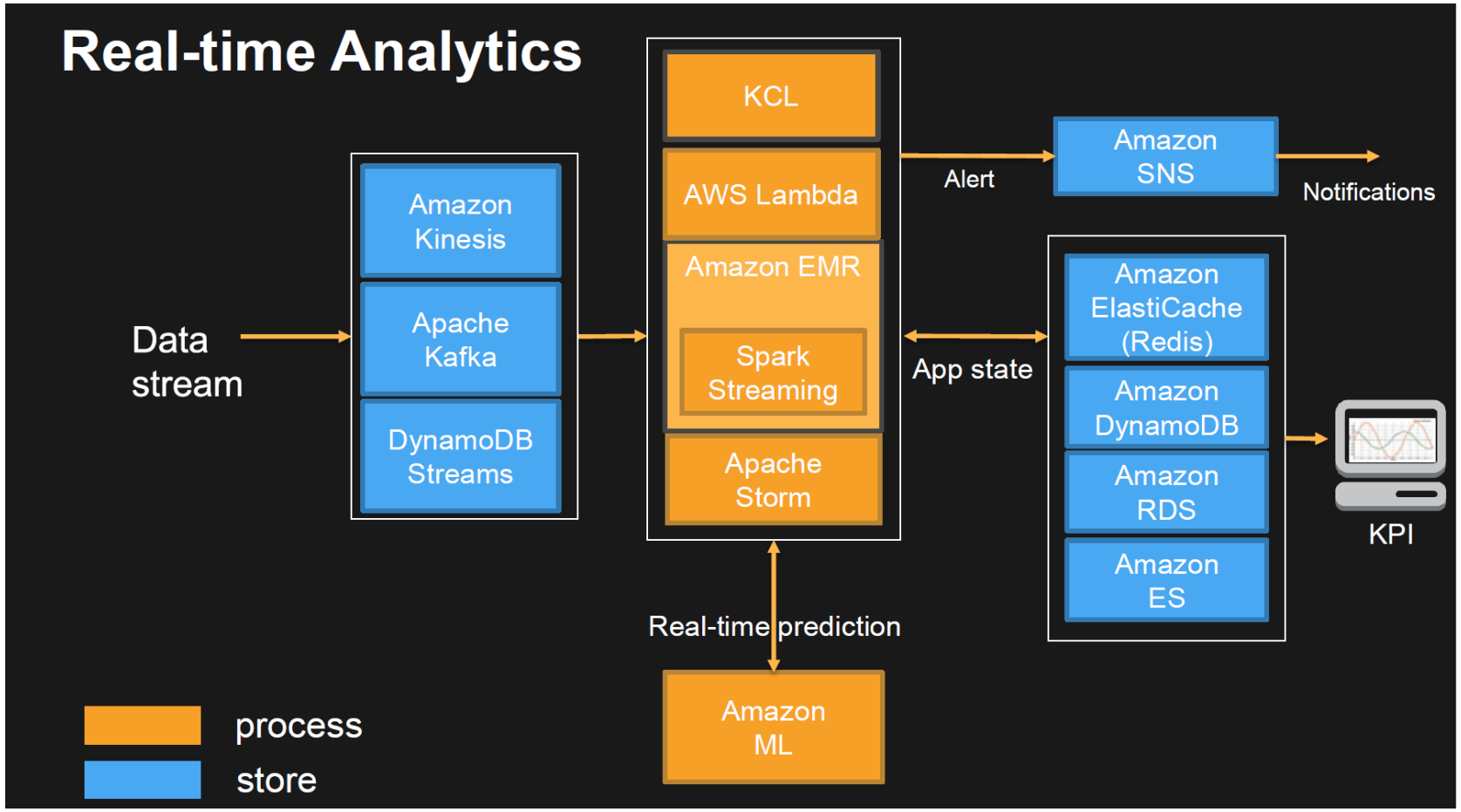
```
import storm

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

SplitSentenceBolt().run()
```

For more information on writing spouts and bolts in other languages, and to learn about how to create topologies in other languages (and avoid the JVM completely), see Using non-JVM languages with Storm.


COLLECT
STORE
ETL
PROCESS / ANALYZE
CONSUME
Applications
Web apps
Mobile apps
Data centers
AWS Direct Connect
Logging
Logging
AWS CloudTrail
Amazon CloudWatch
Transport
AWS Import/Export
Snowball
Messaging
Messaging
Message
IoT
Devices
Sensors & IoT platforms
AWS IoT
RECORDS
DOCUMENTS
FILES
MESSAGES
STREAMS
Cache
NoSQL
SQL
Search
File
Message
Stream
Hot
Warm
Hot
Amazon ElastiCache
Amazon DynamoDB
Amazon RDS
Amazon Elasticsearch Service
Amazon S3
Amazon SQS
Apache Kafka
Amazon Kinesis Streams
Amazon Kinesis Firehose
Amazon DynamoDB Streams
ML
Interactive
Batch
Message
Stream
Fast
Slow
Fast
Amazon Machine Learning
Amazon Redshift
presto
Spark
Amazon EMR
Amazon SQS apps
Amazon EC2
STORM
Amazon EC2
Spark Streaming
Amazon EMR
Amazon Kinesis Analytics
Amazon KCL apps
AWS Lambda
API
Analysis & visualization
Notebooks
IDE
Apps & Services
Amazon QuickSight
kibana
tableau
looker
MicroStrategy
TIBCO Jaspersoft
Flot
Apache Zeppelin
IPython
Interactive Computing
RStudio
3/13/2024
Reference architecture


Data Temperature vs. Processing Speed
Data temperature
Hot
Cold
data
Fast
Amazon Kinesis
Apache Kafka
Amazon
DynamoDB
Amazon S3
Processing speed
Spark Streaming
Apache Storm
AWS Lambda
KCL apps
Native apps
KCL apps
AWS Lambda
Real-time
Amazon
Redshift
Interactive
Amazon
Redshift
Spark
Presto
Batch
Hive
Hive
Slow
Answers

Summary - Streaming Implementation Architecture



Summary

- You have looked at the big data stack
- You have look at different patterns for analytics – alpha, beta, ..
- You looked at Storm for stream processing.