Dr. Ambarish G. Mohapatra

IOT & Applications

Module – 5
Class-4

Hadoop Subprojects: Pig, Hive, Hbase, YARN, Oozie, Spark, Kafka, Apache Storm

Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

Hadoop Related Subprojects

- Pig
  - High-level language for data analysis
- Hive
  - SQL-like Query language and Metastore
- HBase
  - Table storage for semi-structured data
- Zookeeper
  - Coordinating distributed applications
- Mahout
  - Machine learning
- YARN
  - Resource management and job scheduling/monitoring into separate daemons.
- Oozie
  - Server-based workflow scheduling
- Spark
  - Open-source unified analytics engine for large-scale data processing.
- Kafka
  - Data stream used to feed Hadoop BigData lakes.
- Apache Storm
  - Distributed stream processing computation framework written predominantly in the Clojure programming language.

Dr. Ambarish G. Mohapatra

Pig

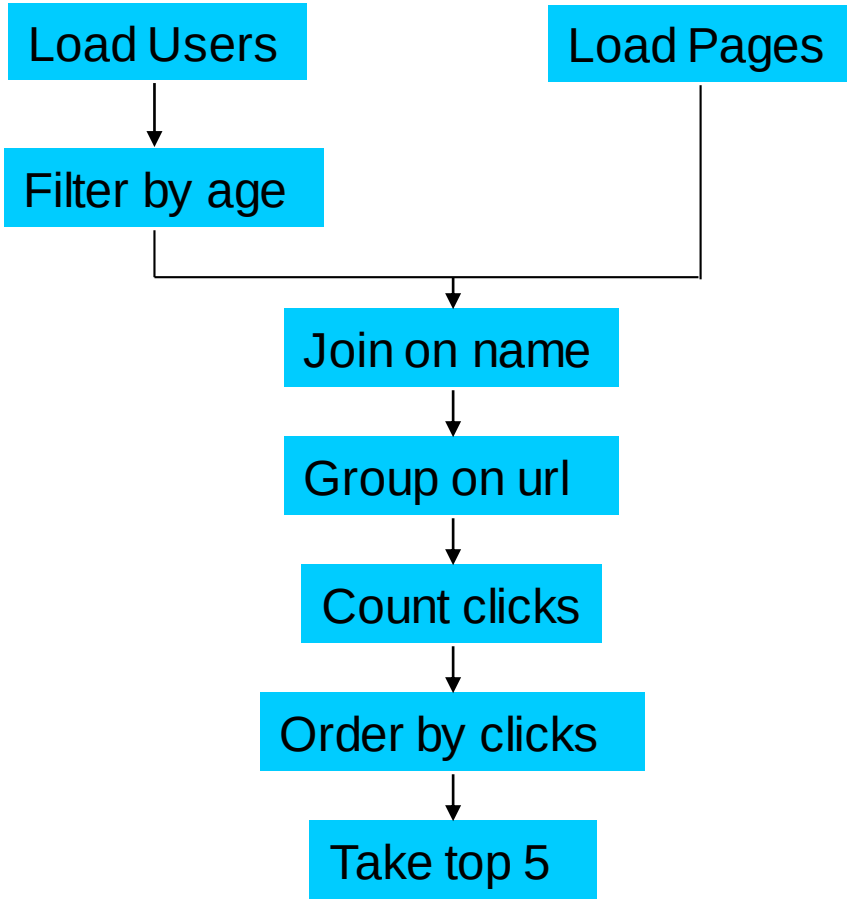
Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

Pig

- Started at Yahoo! Research
- Now runs about 30% of Yahoo!'s jobs
- Features
  - Expresses sequences of MapReduce jobs
  - Data model: nested "bags" of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc.)
  - Easy to plug in Java functions

An Example Problem

- Suppose you have user data in a file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25



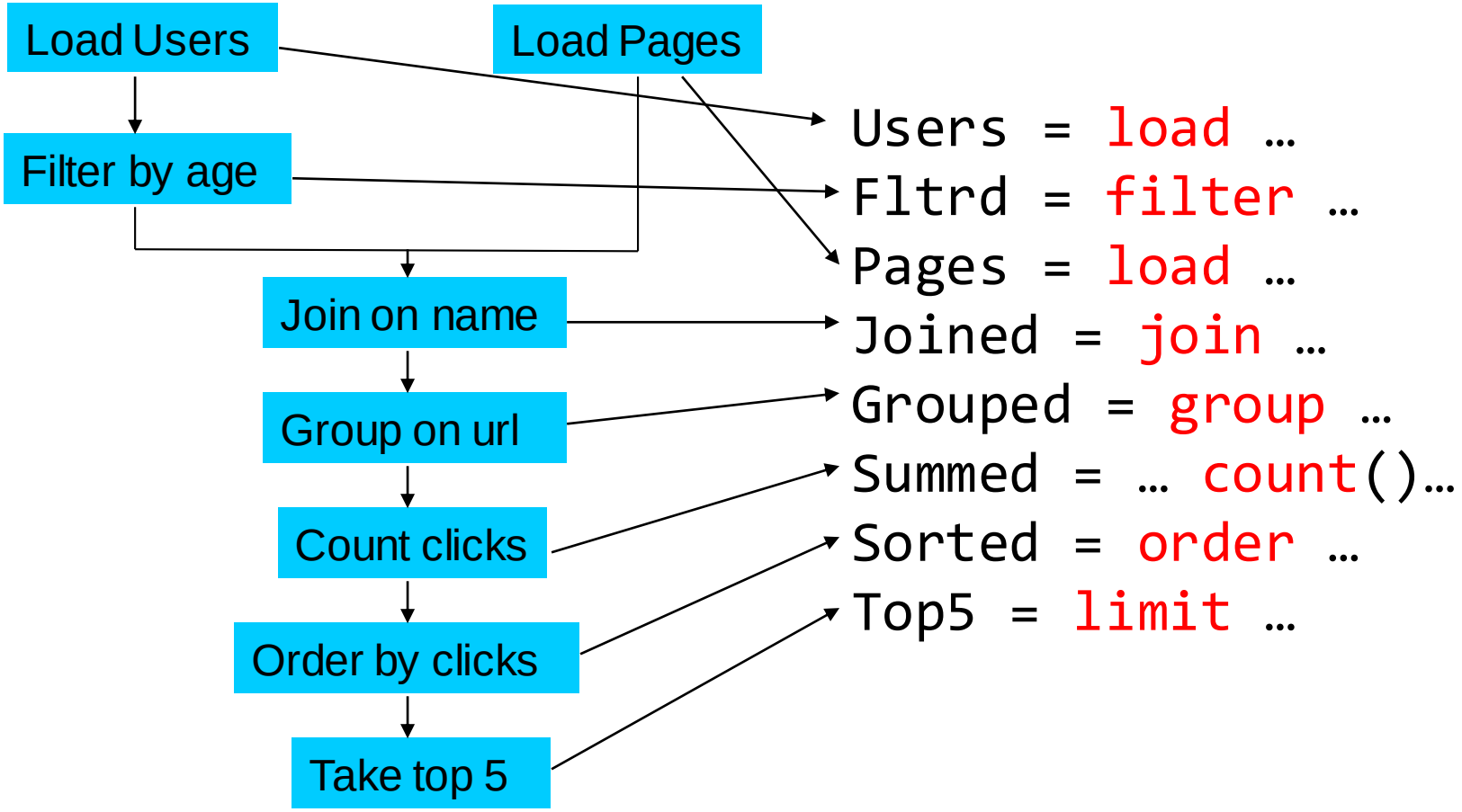
In MapReduce: Same Example Problem

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

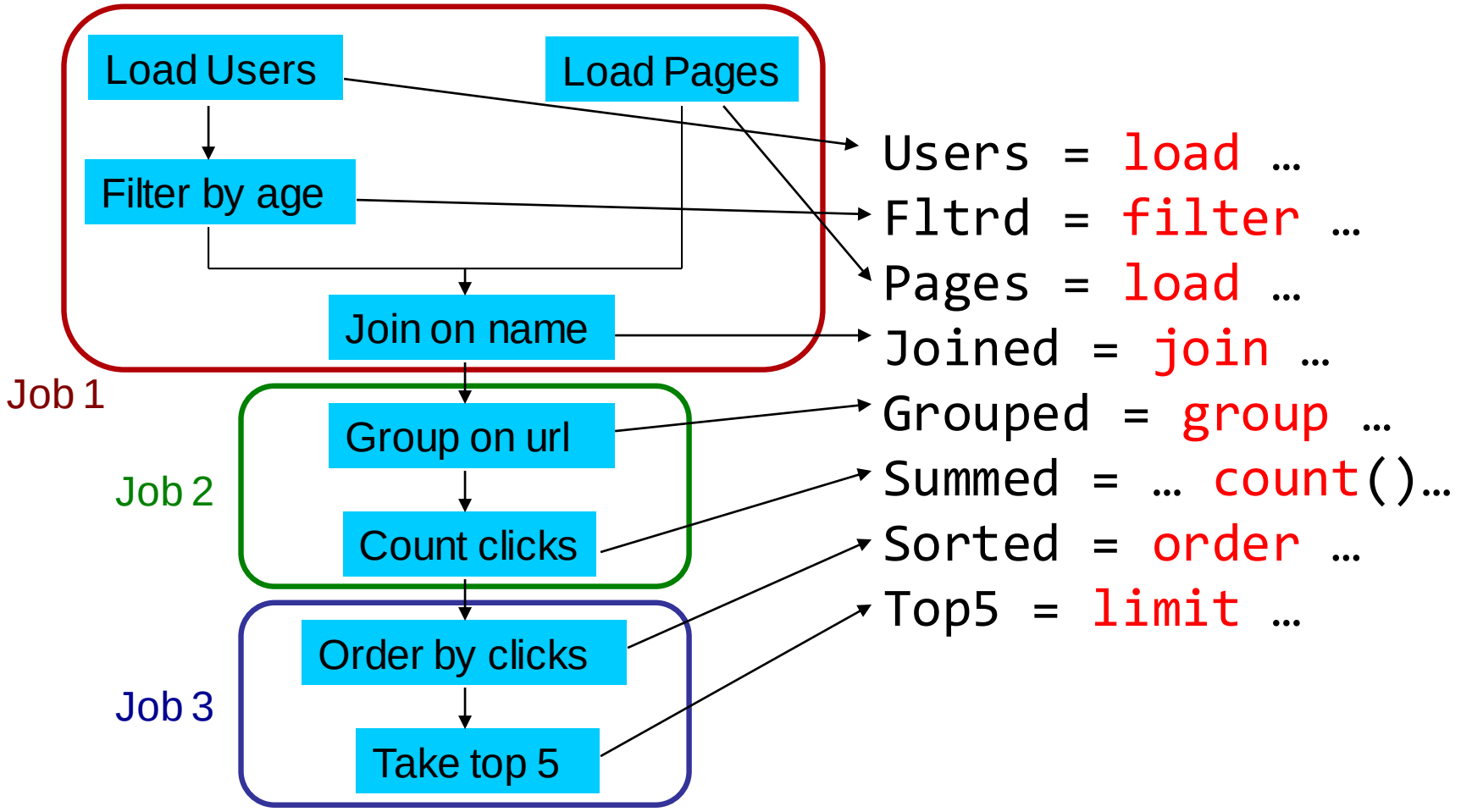
In Pig Latin: Same Example Problem

```
Users = load ‘users’ as (name, age);
Filtered = filter Users by age >= 18 and age <= 25;
Pages = load ‘pages’ as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
                 count(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;
store Top5 into ‘top5sites’;
```

Ease of Translation



Ease of Translation



Dr. Ambarish G. Mohapatra

Hive

Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

Hive

- Developed at Facebook
- Used for majority of Facebook jobs
- "Relational database" built on Hadoop
  - Maintains list of table schemas
  - SQL-like query language (HiveQL)
  - Can call Hadoop Streaming scripts from HiveQL
  - Supports table partitioning, clustering, complex data types, some optimizations

Creating a Hive Table

```
CREATE TABLE page_views(viewTime INT, userid BIGINT,
                    page_url STRING, referrer_url STRING,
                    ip STRING COMMENT 'User IP address')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

- Partitioning breaks table into separate files for each (dt, country) pair

  Ex: /hive/page_view/dt=2008-06-08,country=USA

  /hive/page_view/dt=2008-06-08,country=CA

A Simple Query

- Find all page views coming from xyz.com on March 31st:

```
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01'
AND page_views.date <= '2008-03-31'
AND page_views.referrer_url like '%xyz.com';
```

- Hive only reads partition `2008-03-01,*` instead of scanning entire table

Aggregation and Joins

- Count users who visited each page by gender:

```
SELECT pv.page_url, u.gender, COUNT(DISTINCT u.id)
FROM page_views pv JOIN user u ON (pv.userid = u.id)
GROUP BY pv.page_url, u.gender
WHERE pv.date = '2008-03-03';
```

- Sample output:

page_url	gender	count(userid)
home.php	MALE	12,141,412
home.php	FEMALE	15,431,579
photo.php	MALE	23,941,451
photo.php	FEMALE	21,231,314

Using a Hadoop Streaming Mapper Script

```
SELECT TRANSFORM(page_views.userid,
                                page_views.date)
USING 'map_script.py'
AS dt, uid CLUSTER BY dt
FROM page_views;
```

Dr. Ambarish G. Mohapatra

HBase

Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

HBase - What?

- Modeled on Google's Bigtable
- Row/column store
- Billions of rows/millions on columns
- Column-oriented - nulls are free
- Untyped - stores byte[]

HBase - Data Model

Row	Timestamp	Column family: animal:		Column family repairs:
		animal:type	animal:size	repairs:cost
enclosure1	t2	zebra		1000 EUR
	t1	lion	big	
enclosure2	…	…	…	…

HBase - Data Storage

Column family animal:

(enclosure1, t2, animal:type)	zebra
(enclosure1, t1, animal:size)	big
(enclosure1, t1, animal:type)	lion

Column family repairs:

(enclosure1, t1, repairs:cost)	1000 EUR

HBase - Code

```
HTable table = …
Text row = new Text("enclosure1");
Text col1 = new Text("animal:type");
Text col2 = new Text("animal:size");
BatchUpdate update = new BatchUpdate(row);
update.put(col1, "lion".getBytes("UTF-8"));
update.put(col2, "big".getBytes("UTF-8));
table.commit(update);

update = new BatchUpdate(row);
update.put(col1, "zebra".getBytes("UTF-8"));
table.commit(update);
```

HBase - Querying

- Retrieve a cell

  Cell = table.getRow("enclosure1").getColumn("animal:type").getValue();

- Retrieve a row

  RowResult = table.getRow("enclosure1");

- Scan through a range of rows

  Scanner s = table.getScanner(new String[] { "animal:type" });

Dr. Ambarish G. Mohapatra

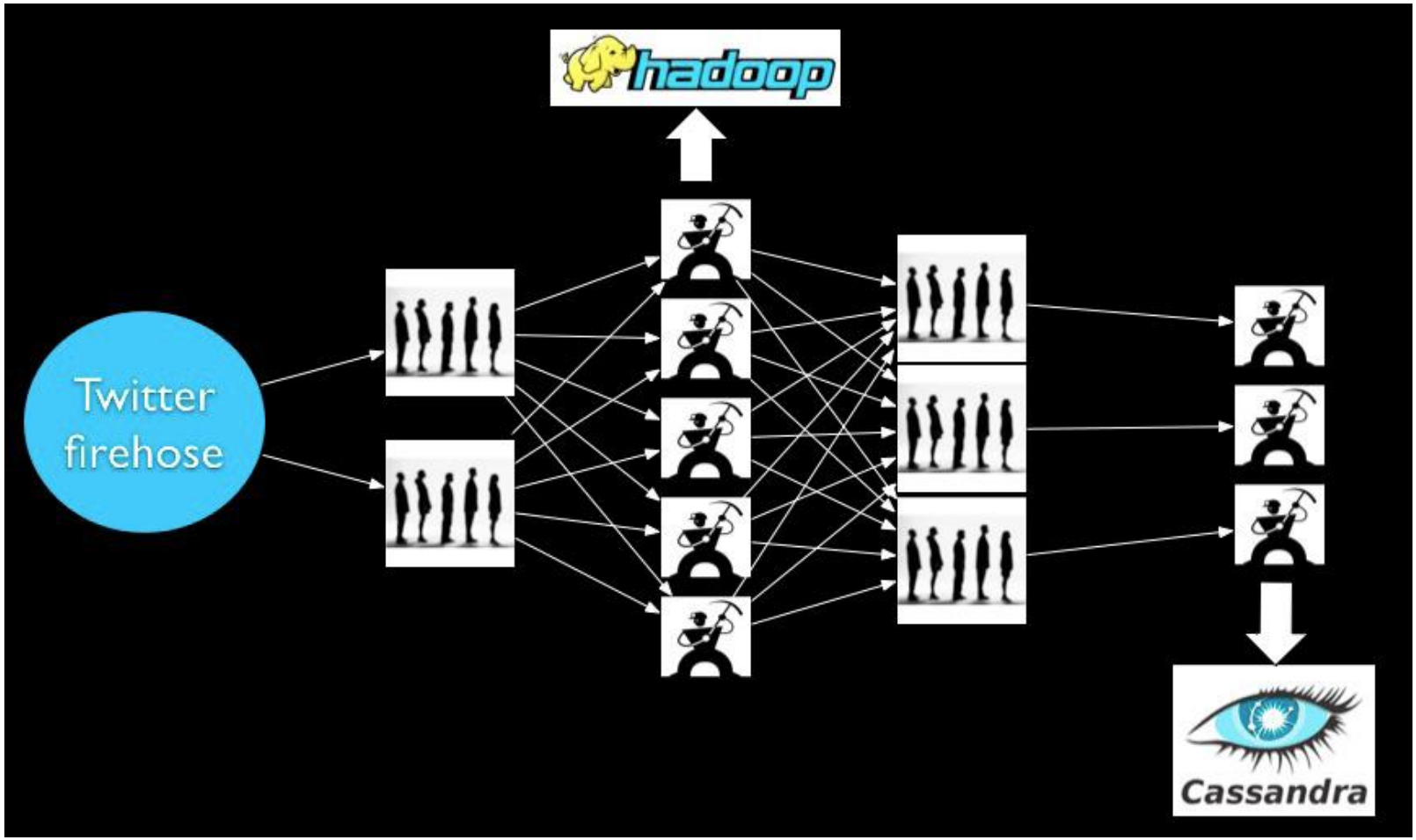
Storm

Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

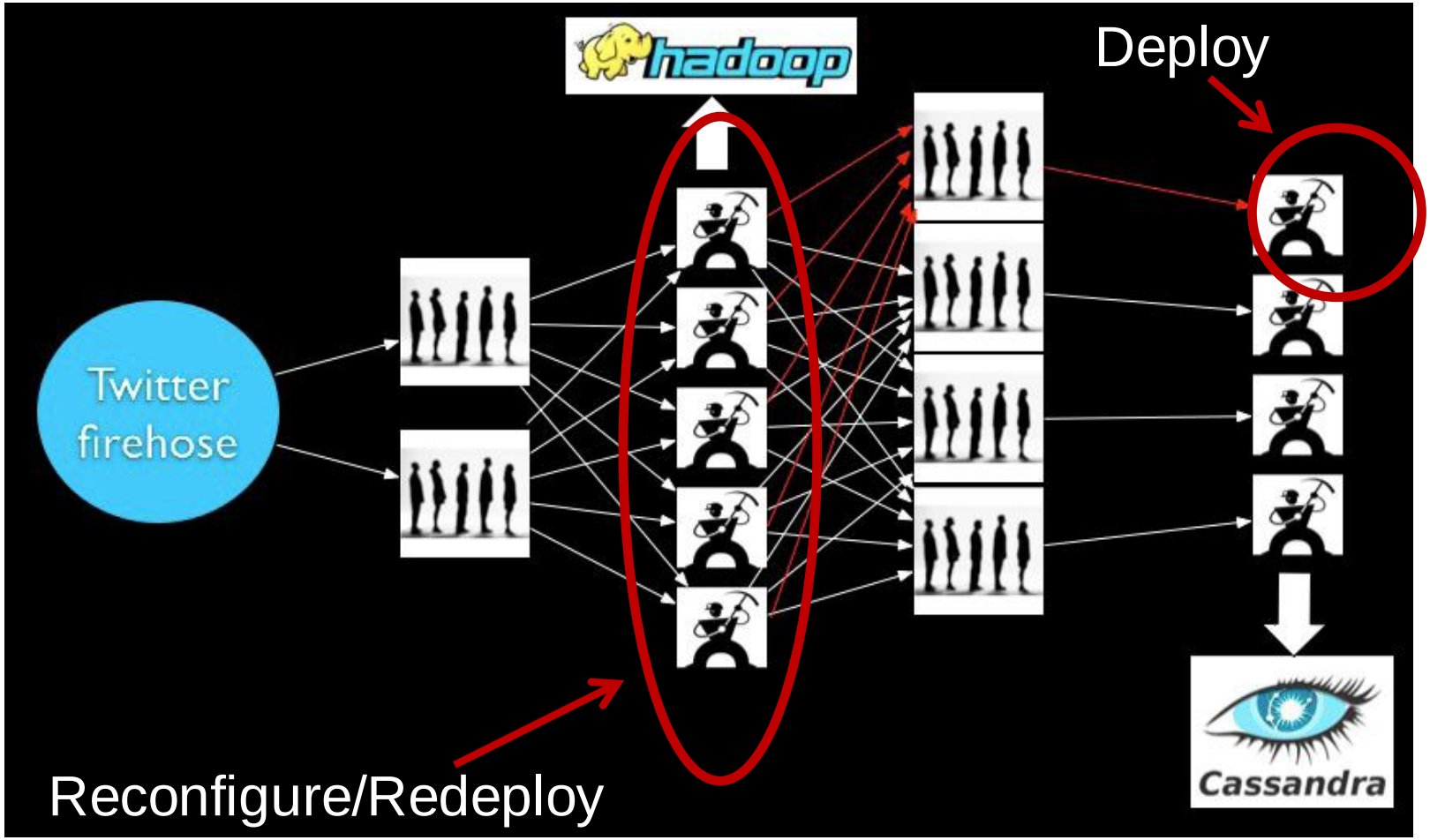
Storm

- Developed by BackType which was acquired by Twitter
- Lots of tools for data (i.e. batch) processing
  - Hadoop, Pig, HBase, Hive, …
- None of them are realtime systems which is becoming a real requirement for businesses
- Storm provides realtime computation
  - Scalable
  - Guarantees no data loss
  - Extremely robust and fault-tolerant
  - Programming language agnostic

Before Storm



Before Storm – Adding a worker



Problems

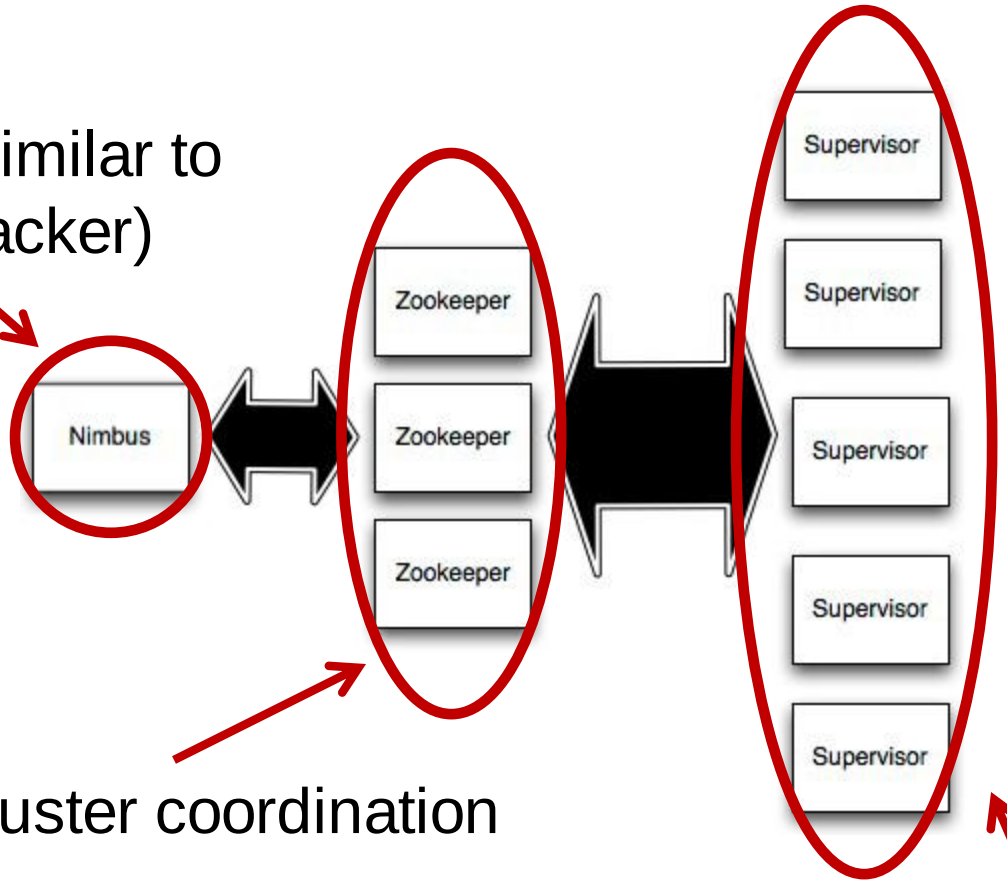
- Scaling is painful
- Poor fault-tolerance
- Coding is tedious

Storm Features

- Guaranteed data processing
- Horizontal scalability
- Fault-tolerance
- No intermediate message brokers!
- Higher level abstraction than message passing

Storm Cluster

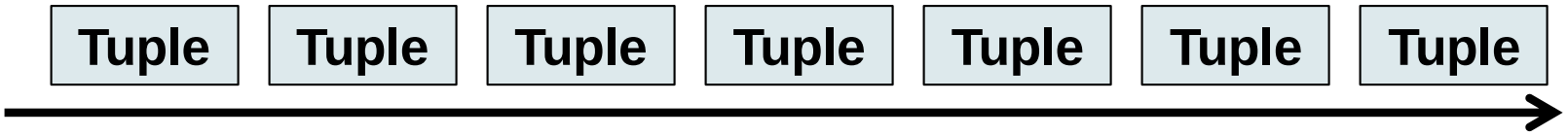
Master node (similar to Hadoop JobTracker)



Used for cluster coordination

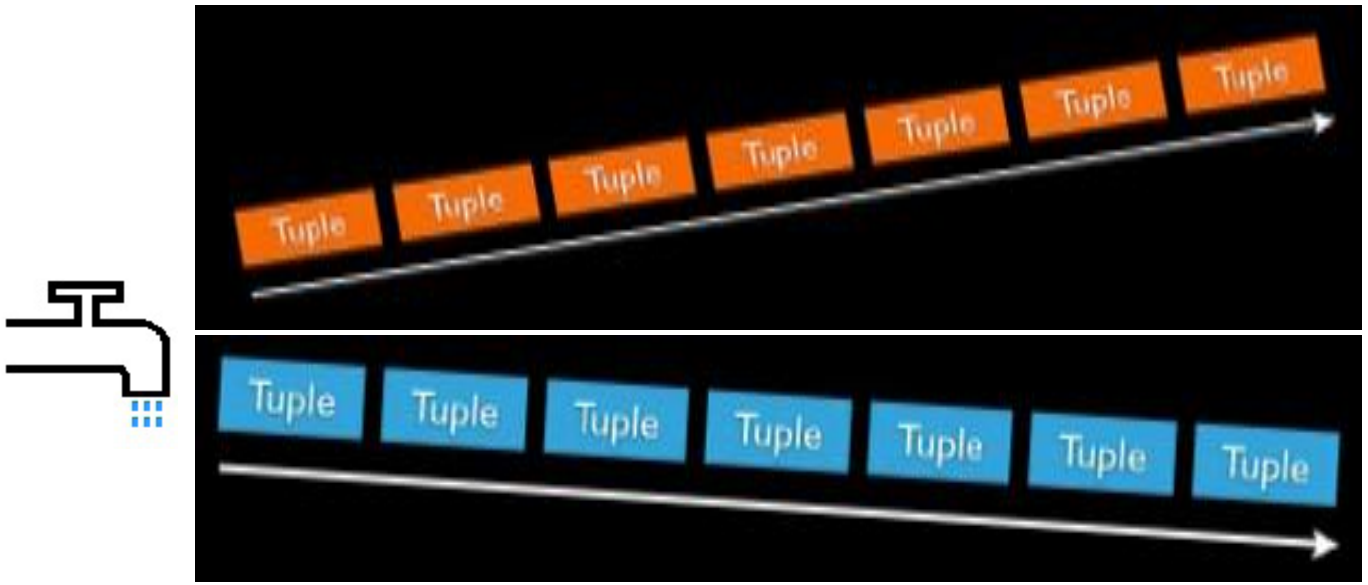
Run worker processes

Streams



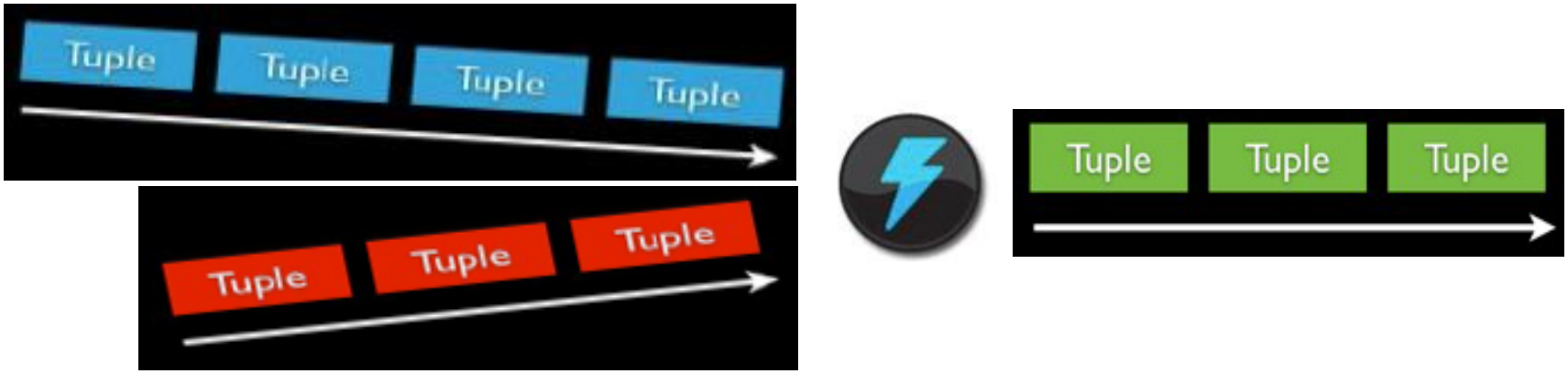
Unbounded sequence of tuples

Spouts



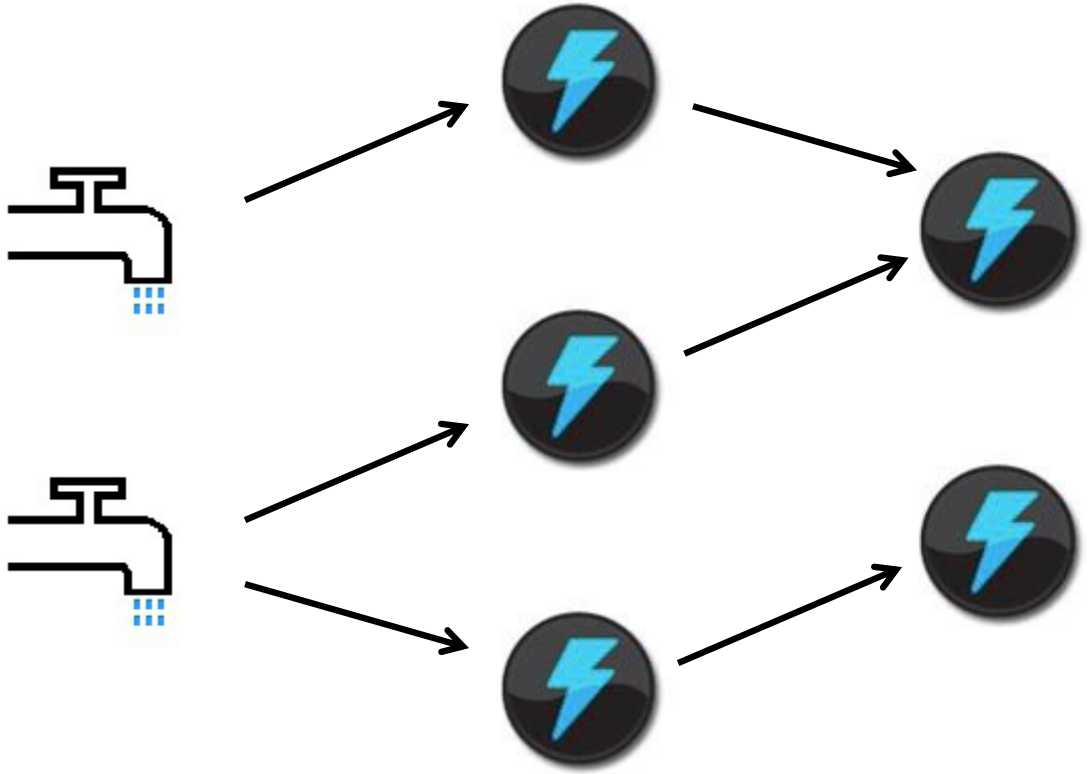
Source of streams

Bolts



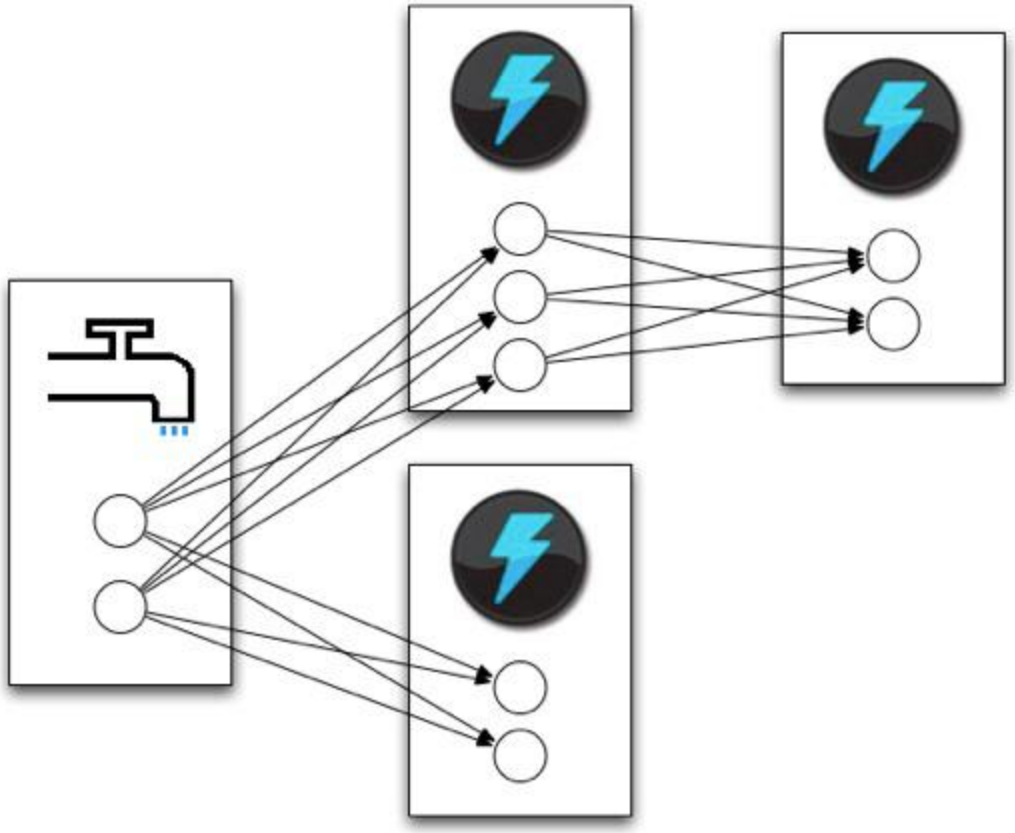
Processes input streams and produces new streams:
Can implement functions such as filters, aggregation, join, etc

Topology



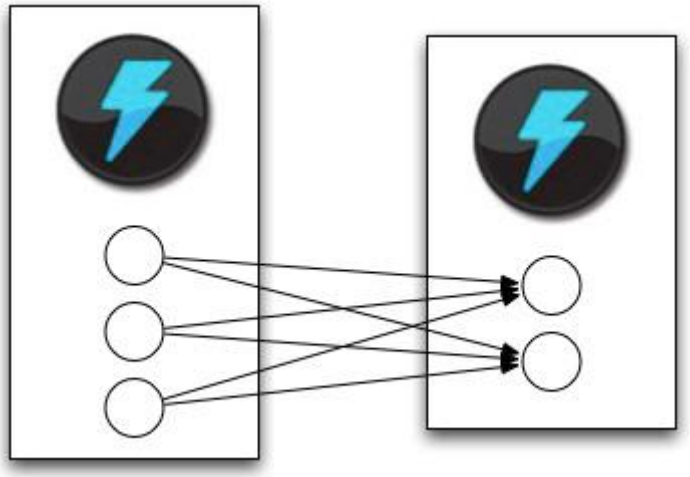
Network of spouts and bolts

Topology



Spouts and bolts execute as many tasks across the cluster

Stream Grouping



When a tuple is emitted which task does it go to?

Stream Grouping

- **Shuffle grouping**: pick a random task
- **Fields grouping**: consistent hashing on a subset of tuple fields
- **All grouping**: send to all tasks
- **Global grouping**: pick task with lowest id

Dr. Ambarish G. Mohapatra

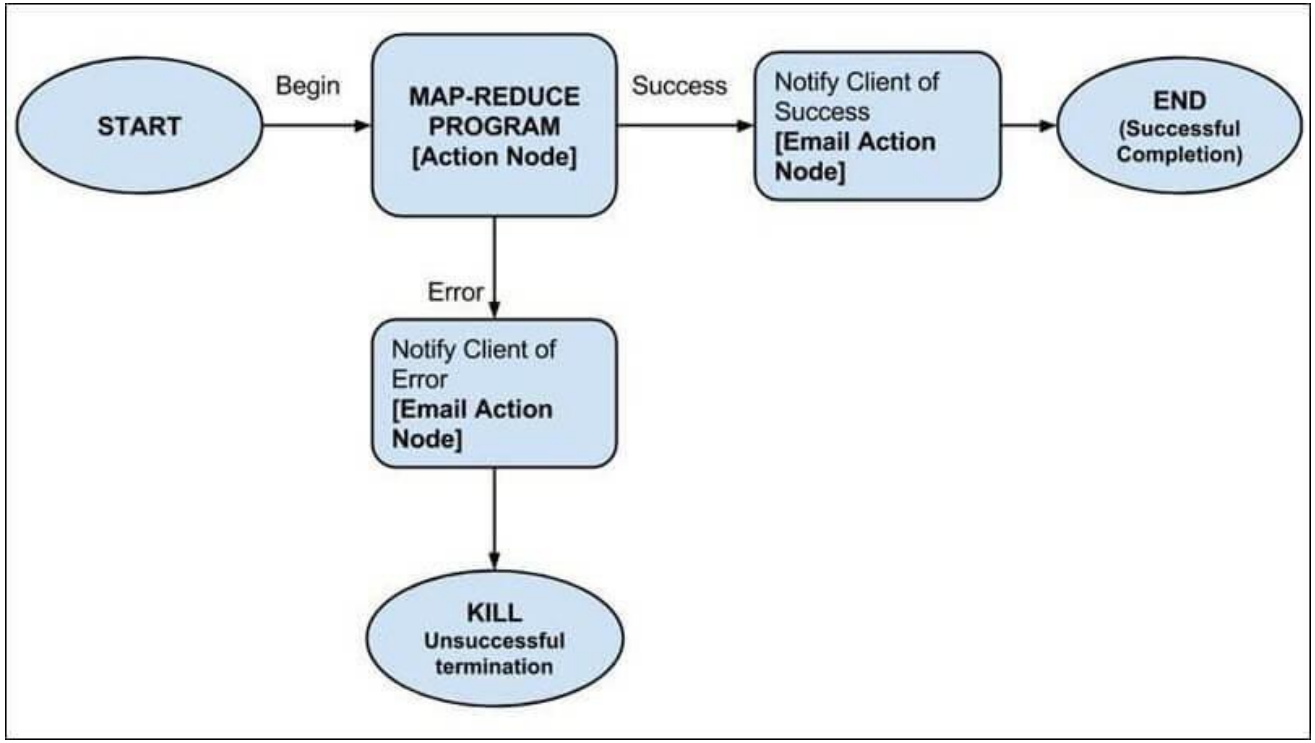
Oozie

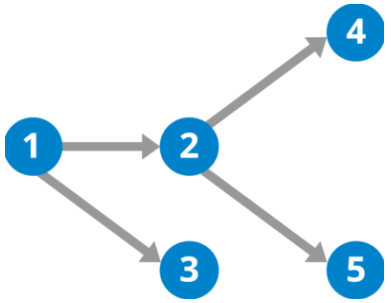
Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

What is OOZIE?

- Apache Oozie is a workflow scheduler for Hadoop. It is a system which runs the workflow of dependent jobs. Here, users are permitted to create Directed Acyclic Graphs of workflows, which can be run in parallel and sequentially in Hadoop.
- It consists of two parts:
  - **Workflow engine:** Responsibility of a workflow engine is to store and run workflows composed of Hadoop jobs e.g., MapReduce, Pig, Hive.
  - **Coordinator engine**: It runs workflow jobs based on predefined schedules and availability of data.

- Oozie is scalable and can manage the timely execution of thousands of workflows (each consisting of dozens of jobs) in a Hadoop cluster.
- Oozie is very much flexible, as well. One can easily start, stop, suspend and rerun jobs. Oozie makes it very easy to rerun failed workflows. One can easily understand how difficult it can be to catch up missed or failed jobs due to downtime or failure. It is even possible to skip a specific failed node.





Dr. Ambarish G. Mohapatra

Kafka

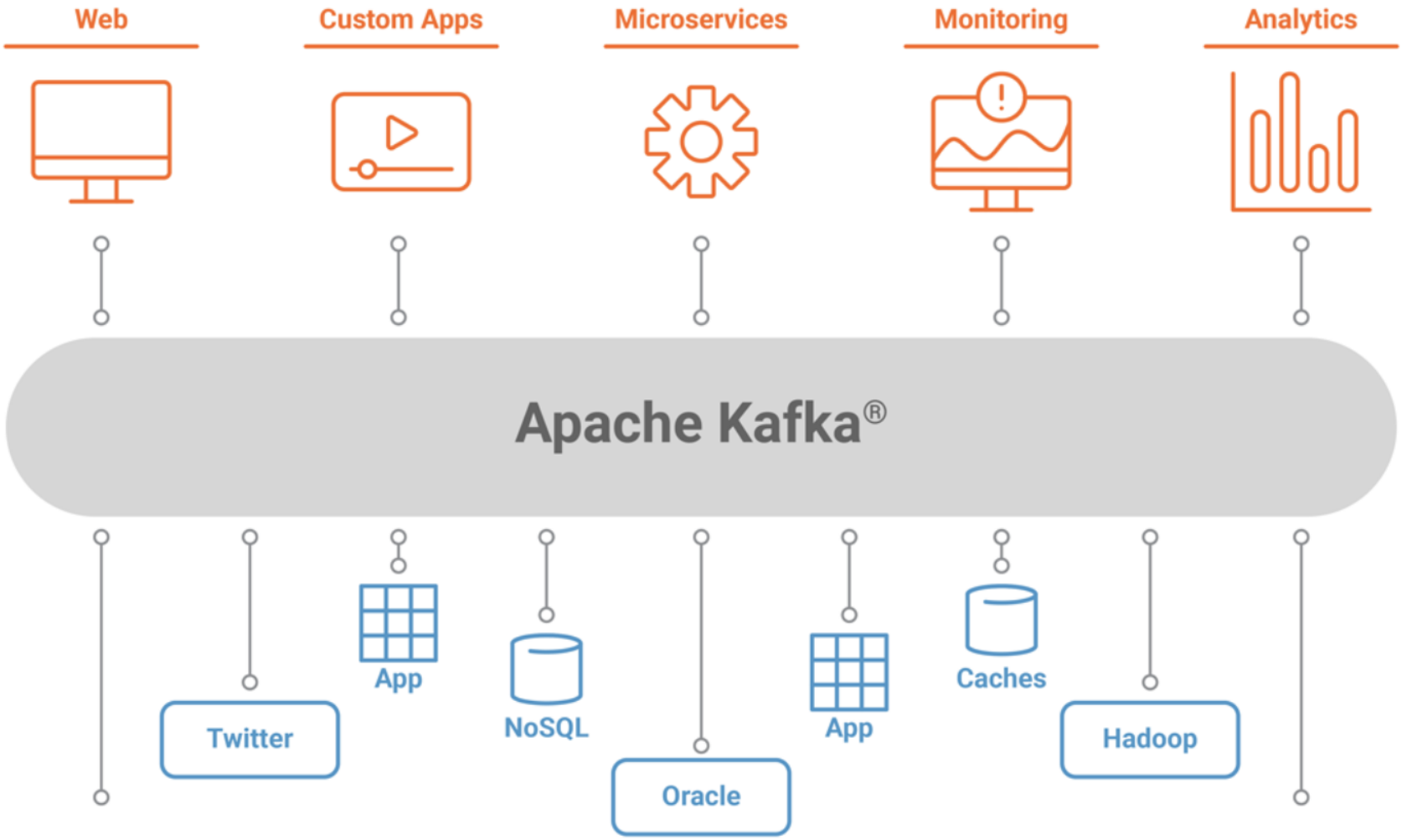
Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

Introduction to Kafka

- Apache Kafka® is a distributed streaming platform that:
  - Publishes and subscribes to streams of records, similar to a message queue or enterprise messaging system.
  - Stores streams of records in a fault-tolerant durable way.
  - Processes streams of records as they occur.
- Kafka is used for these broad classes of applications:
  - Building real-time streaming data pipelines that reliably get data between systems or applications.
  - Building real-time streaming applications that transform or react to the streams of data.

Kafka is run as a cluster on one or more servers that can span multiple datacenters. The Kafka cluster stores streams of *records* in categories called *topics*. Each record consists of a key, a value, and a timestamp.

Apache Kafka Features



Core APIs in Kafka

Kafka has these core APIs:

- Producer API
  - Applications can publish a stream of records to one or more Kafka topics.
- Consumer API
  - Applications can subscribe to topics and process the stream of records produced to them.
- Streams API
  - Applications can act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- Connector API
  - Build and run reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

In Kafka the communication between the clients and the servers is done with a simple, high-performance, language agnostic TCP protocol.

This protocol is versioned and maintains backwards compatibility with older version.

The Java client is provided for Kafka, but clients are available in many languages.

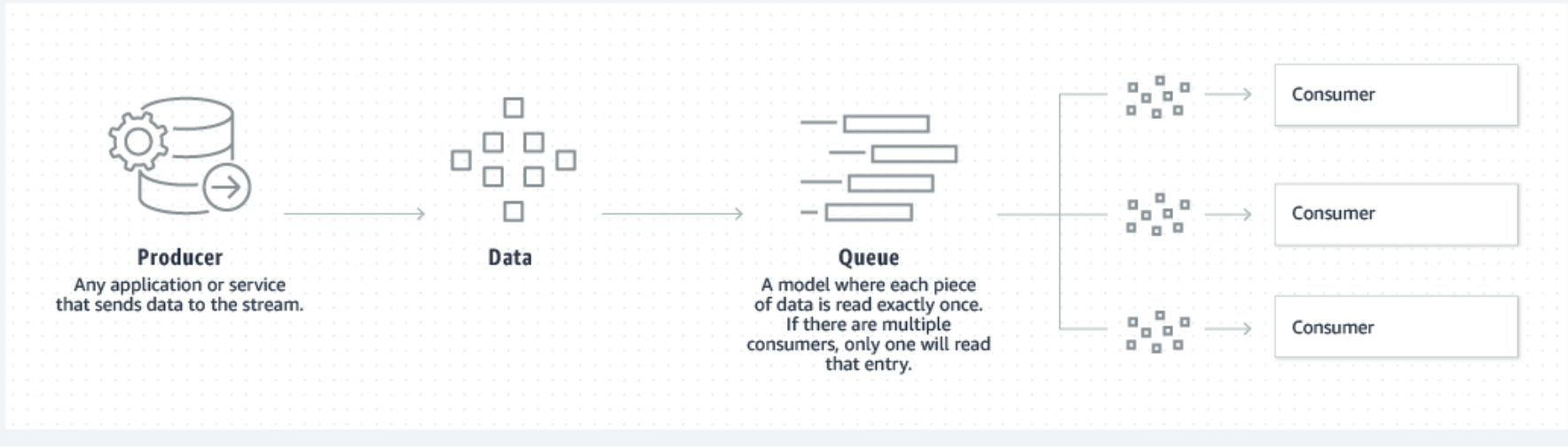
Benefits of Kafka's approach

Scalable	**Fast**	**Durable**
Kafka's partitioned log model allows data to be distributed across multiple servers, making it scalable beyond what would fit on a single server.	Kafka decouples data streams so there is very low latency, making it extremely fast.	Partitions are distributed and replicated across many servers, and the data is all written to disk. This helps protect against server failure, making the data very fault-tolerant and durable.

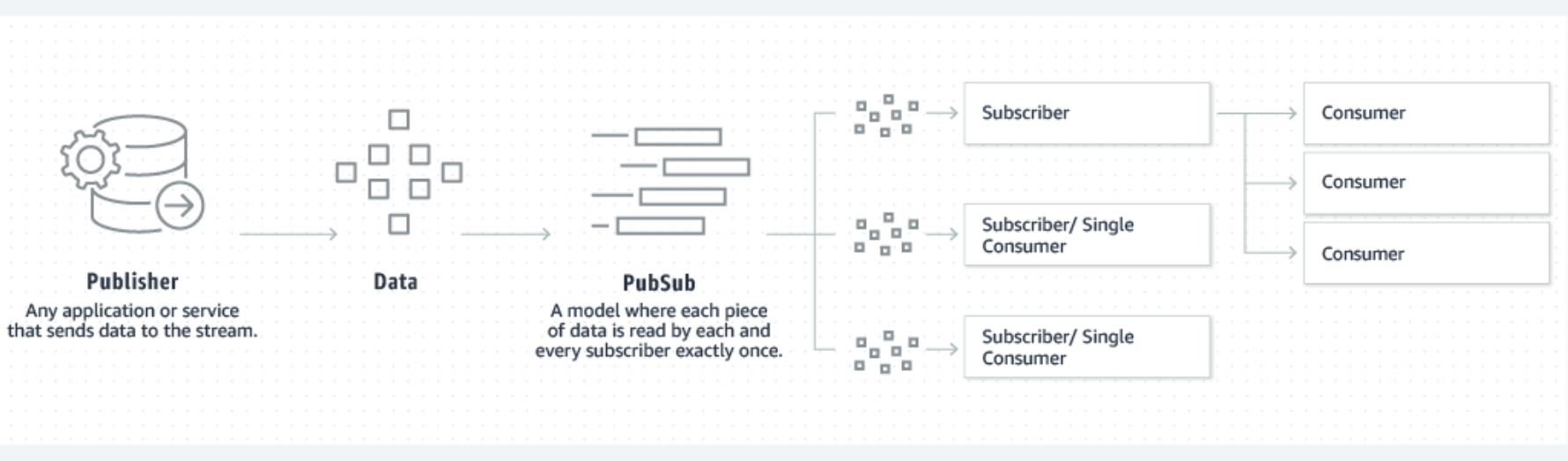
How does Kafka work?

- Kafka combines two messaging models, queuing and publish-subscribe, to provide the key benefits of each to consumers.
- Queuing allows for data processing to be distributed across many consumer instances, making it highly scalable. However, traditional queues aren't multi-subscriber.
- The publish-subscribe approach is multi-subscriber, but because every message goes to every subscriber it cannot be used to distribute work across multiple worker processes.
- Kafka uses a partitioned log model to stitch together these two solutions. A log is an ordered sequence of records, and these logs are broken up into segments, or partitions, that correspond to different subscribers.
- This means that there can be multiple subscribers to the same topic and each is assigned a partition to allow for higher scalability.
- Finally, Kafka's model provides replayability, which allows multiple independent applications reading from data streams to work independently at their own rate.

Queuing



Publish-Subscribe



Apache Kafka Characteristics

CHARACTERISTICS	APACHE KAFKA
Architecture	Kafka uses a partitioned log model, which combines messaging queue and publish subscribe approaches.
Scalability	Kafka provides scalability by allowing partitions to be distributed across different servers.
Message retention	Policy based, for example messages may be stored for one day. The user can configure this retention window.
Multiple consumers	Multiple consumers can subscribe to the same topic, because Kafka allows the same message to be replayed for a given window of time.
Replication	Topics are automatically replicated, but the user can manually configure topics to not be replicated.
Message ordering	Each consumer receives information in order because of the partitioned log architecture.
Protocols	Kafka uses a binary protocol over TCP.

Dr. Ambarish G. Mohapatra

Spark

Silicon ...beyond teaching
Silicon Institute of Technology
Bhubaneswar

Introduction to Apache Spark

- Apache Spark is a unified analytics engine for large-scale data processing.
- It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.
- It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

What is Spark?

- Apache Spark is a general-purpose & lightning fast cluster computing system. It provides a high-level API. For example, Java, Scala, Python, and R. Apache Spark is a tool for Running Spark Applications. Spark is 100 times faster than Bigdata Hadoop and 10 times faster than accessing data from disk.
- Spark is written in Scala but provides rich APIs in Scala, Java, Python, and R.
- It can be integrated with Hadoop and can process existing Hadoop HDFS data.

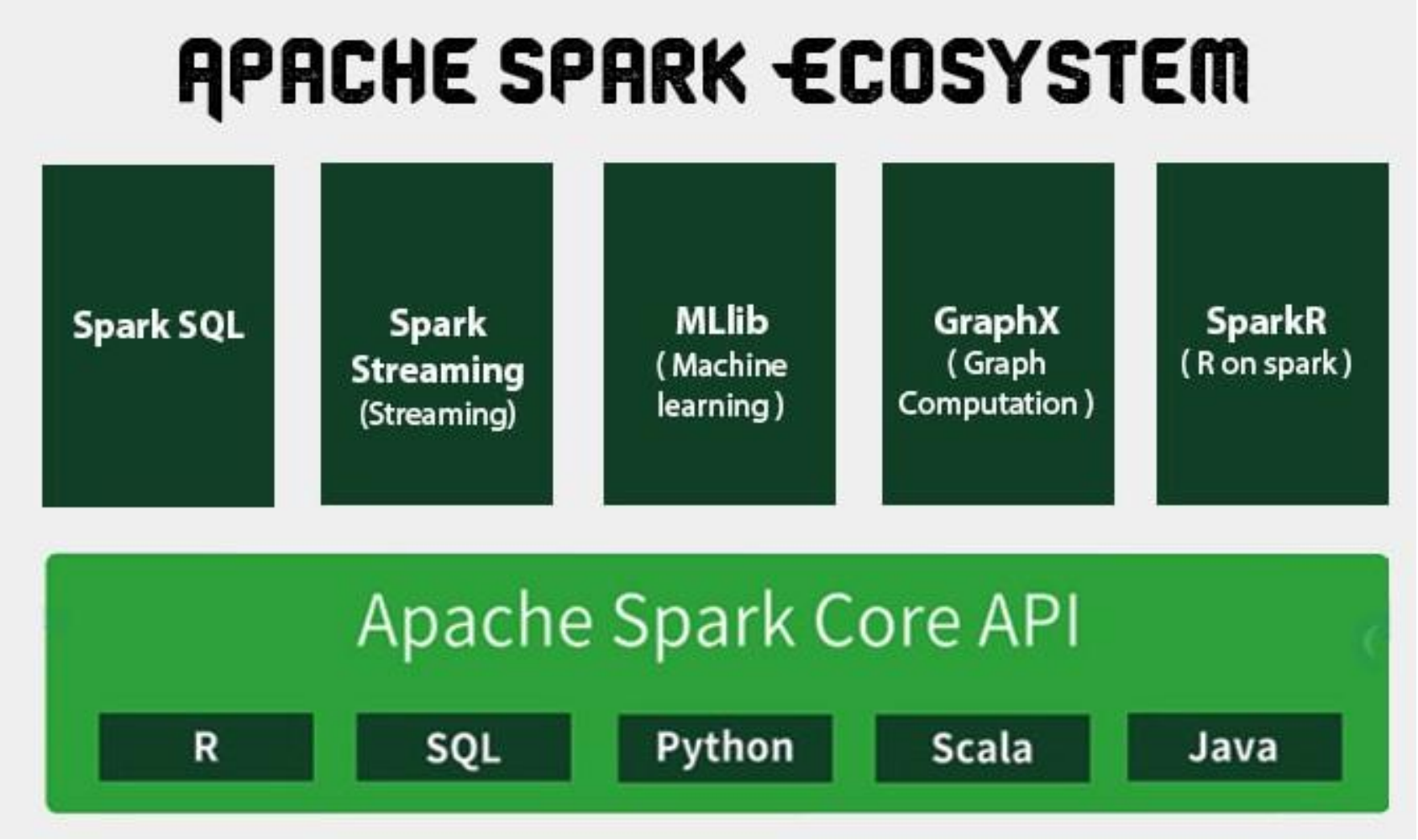
History Of Apache Spark

- Apache Spark was introduced in 2009 in the UC Berkeley R&D Lab, later it becomes AMPLab.
- It was open sourced in 2010 under Berkeley Source Distribution (BSD) license.
- In 2013 spark was donated to Apache Software Foundation where it became top-level Apache project in 2014.

Why Spark?

After studying Apache Spark introduction lets discuss, why Spark come into existence?

- In the industry, there is a need for a general-purpose cluster computing tool as:
  - Hadoop MapReduce can only perform batch processing.
  - Apache Storm / S4 can only perform stream processing.
  - Apache Impala / Apache Tez can only perform interactive processing
  - Neo4j / Apache Giraph can only perform graph processing
- Hence in the industry, there is a big demand for a powerful engine that can process the data in real-time (streaming) as well as in batch mode. There is a need for an engine that can respond in sub-second and perform in-memory processing.


APACHE SPARK ECOSYSTEM
Spark SQL
Spark Streaming (Streaming)
MLlib (Machine learning)
GraphX (Graph Computation)
SparkR (R on spark)
Apache Spark Core API
R
SQL
Python
Scala
Java

THANK YOU