Advanced Data Processing Techniques for Distributed Applications and Systems

Hong-Linh Truong
Distributed Systems Group, TU Wien

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong
@linhsolar

What this lecture is about?

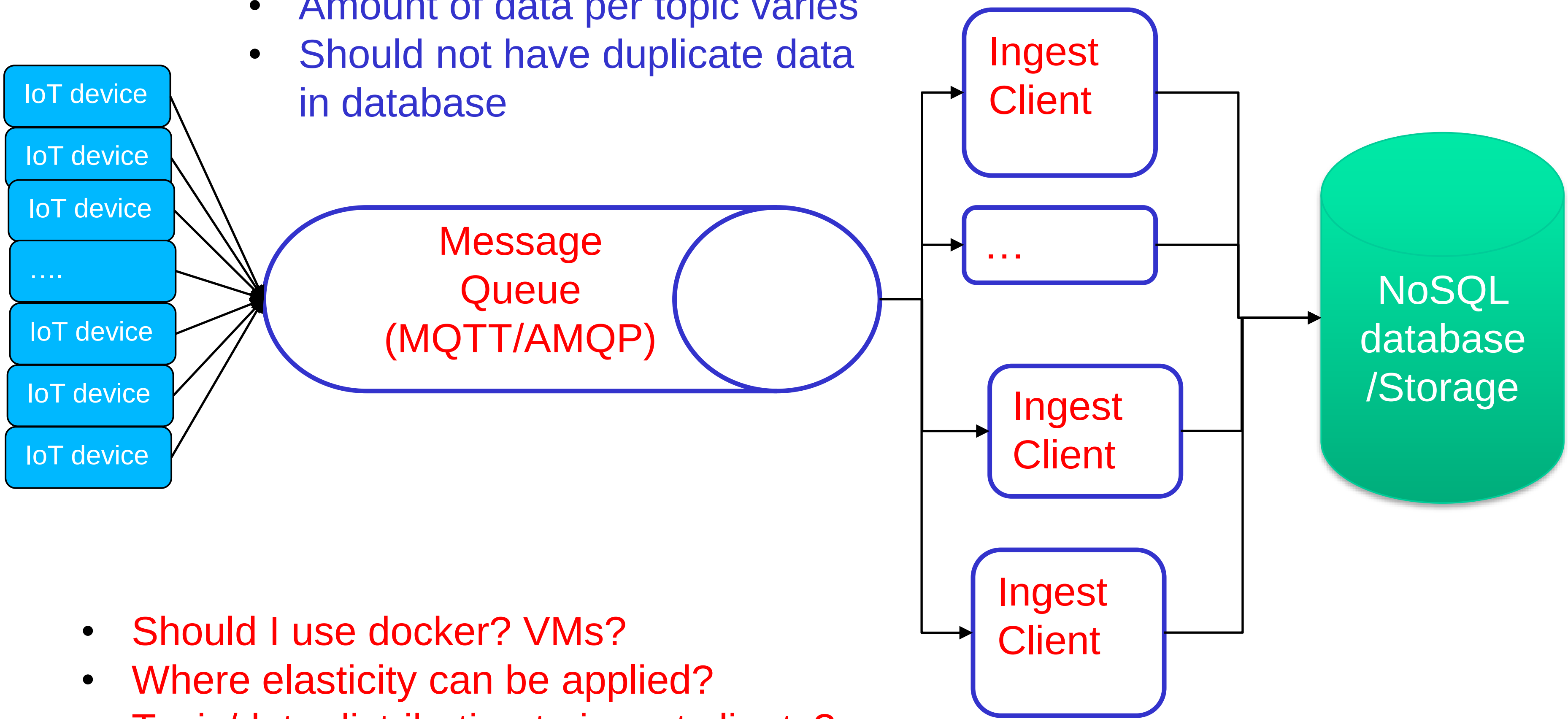
- Large-scale data analytics
- Advanced messaging
  - Apache Kafka
- Advanced data analytics with streaming data processing
  - Stream processing with Apache Apex
- Advanced data analytics with workflows
  - Data pipeline with Airflow/Beam

Large-scale data analytics

- Analytics-as-a-service
  - Understand monitoring information, logs, user activities, etc.
  - Provide insightful information for optimizing business
- Big data analytics
  - Handle and process big data at rest and in motion
- Key issues
  - Collect/produce messages from distributed application components and large-scale monitoring systems
  - Need scalable and reliable large-scale messaging broker systems
  - Require workflow and stream data processing capabilities
  - Integrate with various different types of services and data sources

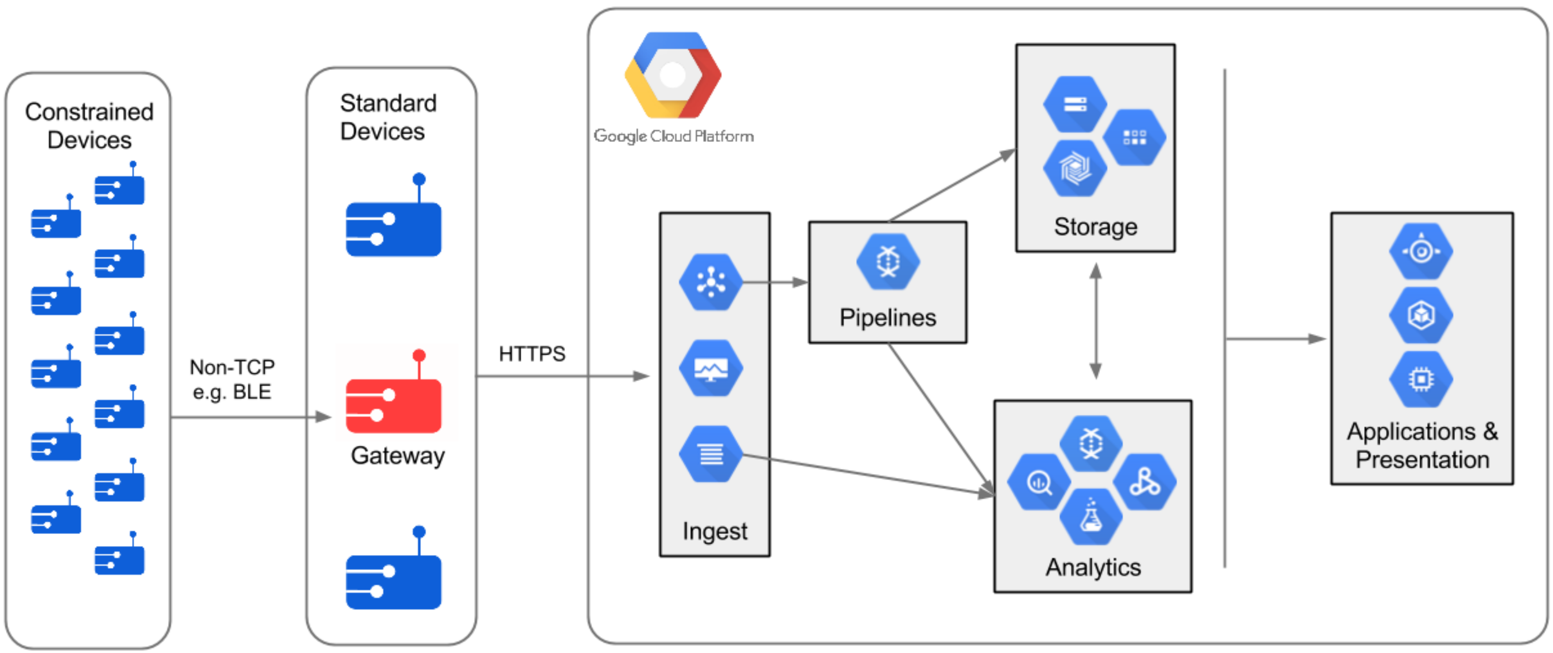
Example from Lecture 4

- Multiple topics
- Amount of data per topic varies
- Should not have duplicate data in database



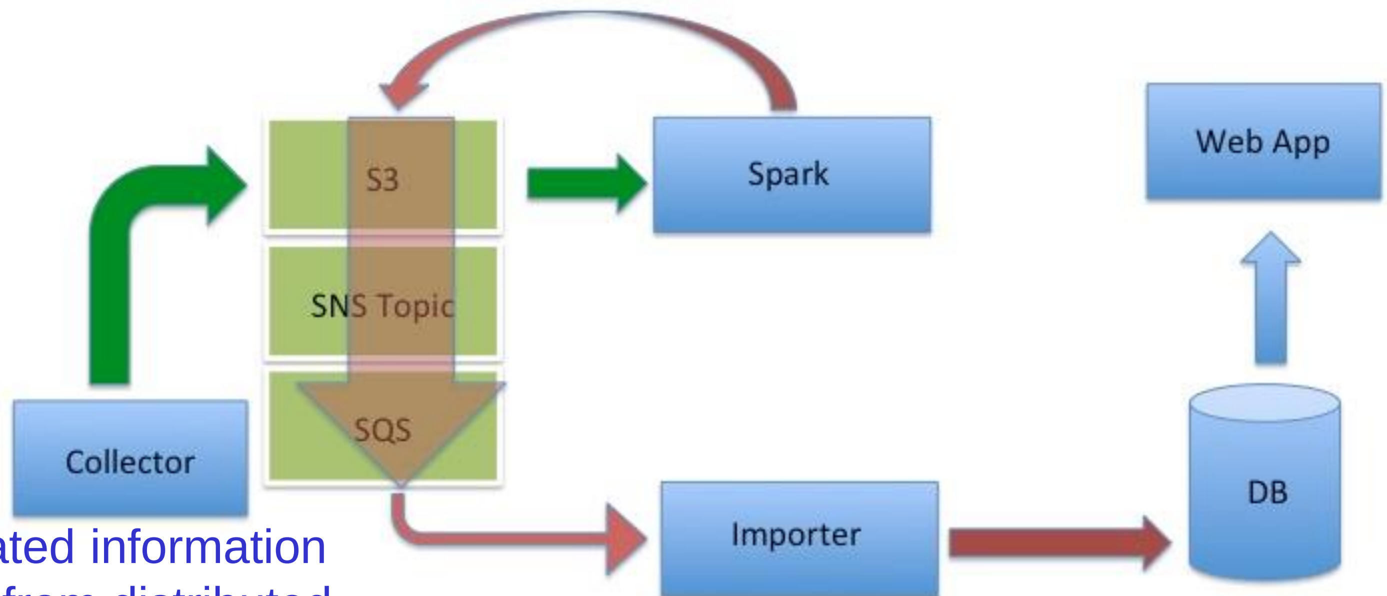
- Should I use docker? VMs?
- Where elasticity can be applied?
- Topic/data distribution to ingest clients?

Implementation atop Google cloud



Source: https://cloud.google.com/solutions/architecture/streamprocessing

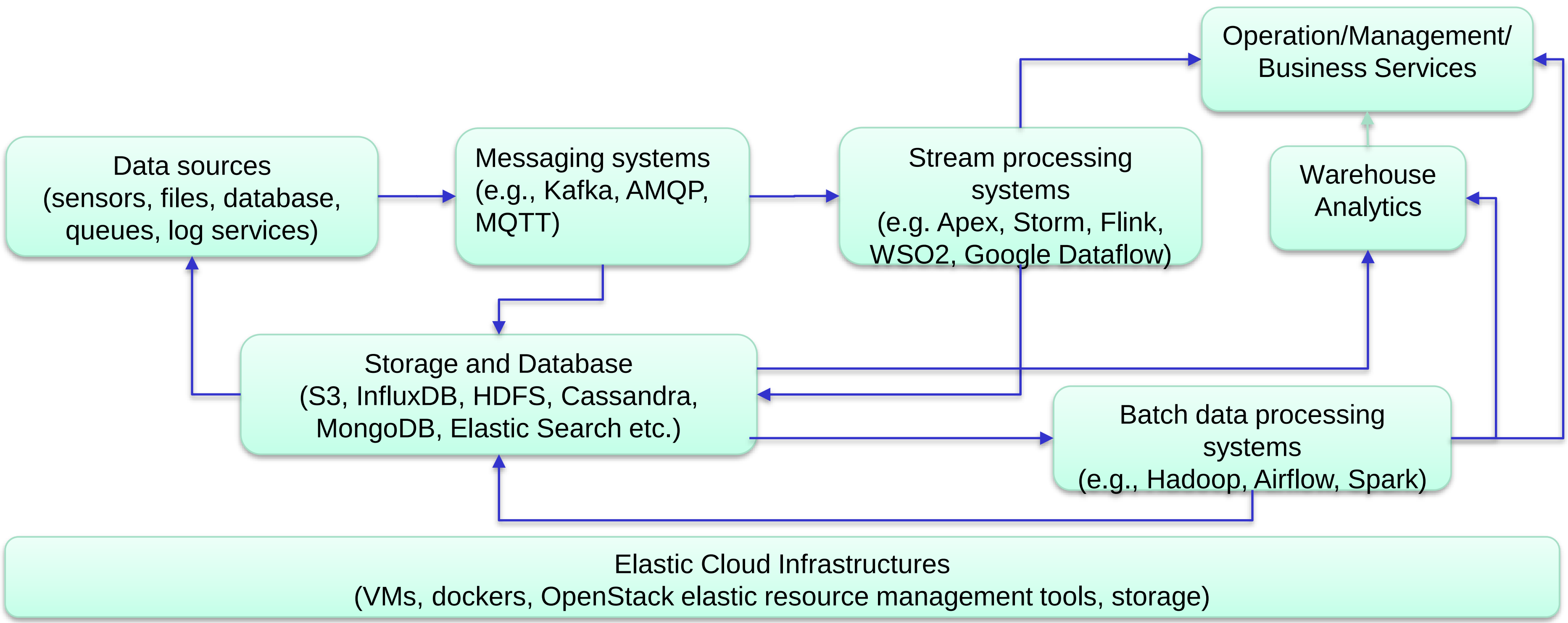
Example: monitoring and security



Security-related information and metrics from distributed customers

Source: http://highscalability.com/blog/2015/9/3/how-agari-uses-airbnbs-airflow-as-a-smarter-cron.html

Cloud services and big data analytics



Recall: Message-oriented Middleware (MOM)

- Well-supported in large-scale systems for
  - Persistent and asynchronous messages
  - Scalable message handling
- Message communication and transformation
  - publish/subscribe, routing, extraction, enrichment
- Several implementations

Amazon SQS

JMS

Apache Qpid™

stormmq

Apache Kafka

RabbitMQ
Messaging that just works

Recall: Workflow of Web services

- You learn it from the Advanced Internet Computing course

- Typically for composing Web services from different enterprises/departments for different tasks

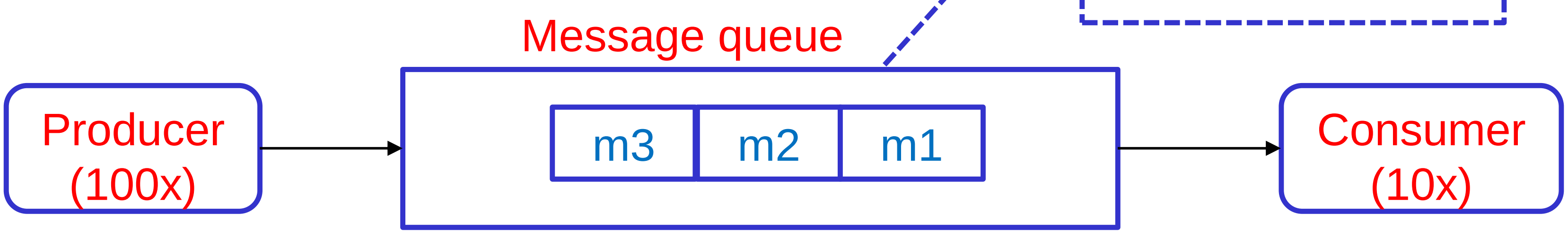
- For big data analytics and Analytics-as-a-Service
  - Tasks are not just from Web services

http://kafka.apache.org/ , originally from LinkedIn

APACHE KAFKA

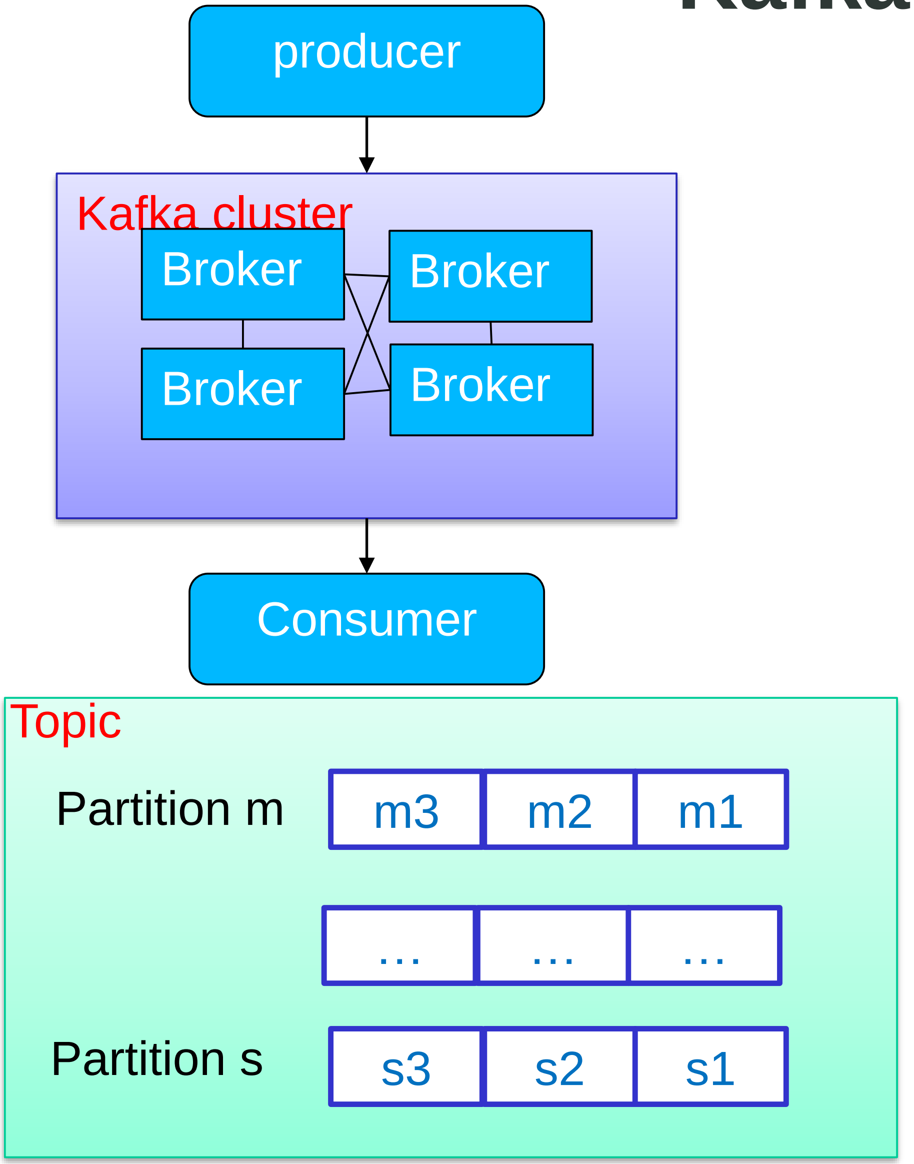
Some use cases

- Producers generate a lot of realtime events
- Producers and consumers have different processing speeds
  - E.g. activity logging



- Rich and diverse types of events
  - E.g. cloud-based logging
- Dealing with cases when consumers might be on and off (fault tolerance support)

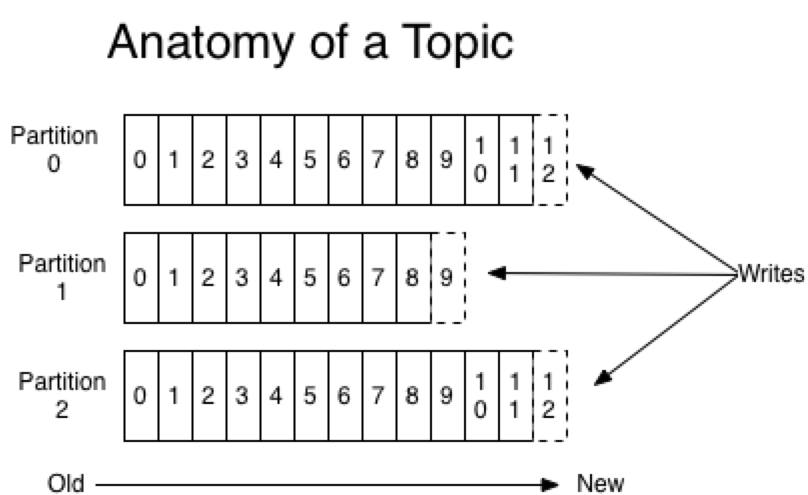
Kafka Design



- Use cluster of brokers to deliver messages
- A topic consists of different partitions
- Durable messages, ordered delivery via partitions
- Online/offline consumers
- Using filesystem heavily for message storage and caching

Messages, Topics and Partitions

- Ordered, immutable sequence of messages
- Messages are kept in a period of time (regardless of consumers or not)
- Support total order for messages within a partition
- Partitions are distributed among server



Source: http://kafka.apache.org/documentation.html

Consumers

- Consumer pulls the data
- The consumer keeps a single pointer indicating the position in a partition to keep track the offset of the next message being consumed
- Why?

→ allow customers to design their speed

→ support/optimize batching data

→ easy to implement total order over message

→ easy to implement reliable message/fault tolerance

Example of a Producer

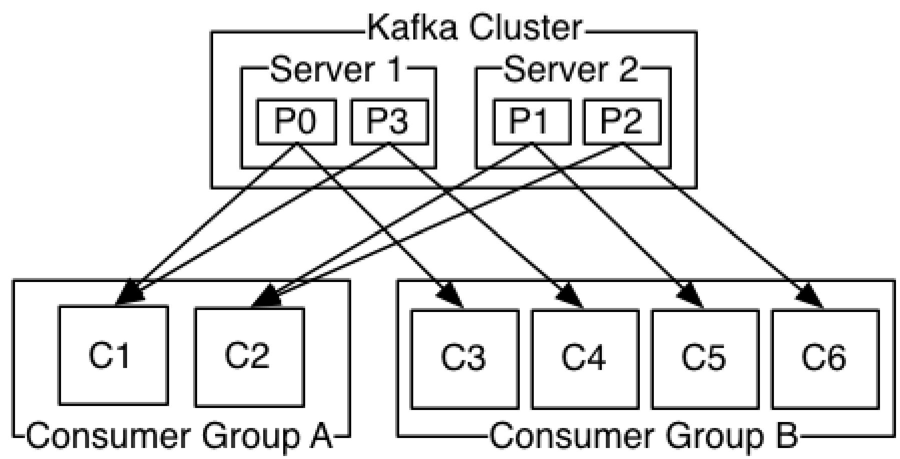
```java
public SimpleProducer( String url, String inputfile, String topic ) {
    Properties props = new Properties();
    props.put("bootstrap.servers", url);
    props.put("client.id", "rdsea.io.training.demo");
    props.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    producer = new KafkaProducer<Integer,String>(props);
    this.topic = topic;
    this.inputfile =inputfile;
}

public void run() {
    int messageNo = 1;
  //read data from file:
    try {
        Reader in = new FileReader(inputfile);
        Iterable<CSVRecord> records = CSVFormat.RFC4180.withFirstRecordAsHeader().parse(in);
        for (CSVRecord record : records) {

            JsonObject event = new JsonObject();
            event.addProperty("USERPHONE", 6645);
            event.addProperty("TIME", Long.parseLong(record.get("TIME")));

            event.addProperty("lat", Float.parseFloat(record.get("LATITUDE")));
            event.addProperty("lon", Float.parseFloat(record.get("LONGITUDE")));

            event.addProperty("GSM_BIT_ERROR_RATE", Float.parseFloat(record.get("GSM_BIT_ERROR_RATE")));
            event.addProperty("GSM_SIGNAL_STRENGTH", Float.parseFloat(record.get("GSM_SIGNAL_STRENGTH")));
            //a simple way to handle missing data is to skip the record
            if (!record.get("LOC_ACCURACY").equals("")) {
                event.addProperty("LOC_ACCURACY", Float.parseFloat(record.get("LOC_ACCURACY")));
            } else {
                continue;
            }
            if (!record.get("LOC_SPEED").equals("")) {
                event.addProperty("LOC_SPEED", Float.parseFloat(record.get("LOC_SPEED")));
            } else {
                continue;
            }

            String eventString = "{\"event\": " + event + "}";
            try {
                            producer.send(new ProducerRecord<Integer,String>(topic,messageNo,eventString)).get();
                    } catch (ExecutionException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    }
                System.out.println("Sent message: (" + messageNo + ", " + eventString + ")");
```

Example of a consumer

```java
public class SimpleConsumer {
    private final KafkaConsumer<Integer, String> consumer;
    private final String topic;
    private final int pollNr;
    public SimpleConsumer(String url, String topic, int pollNr) {

        Properties props = new Properties();
        //just use standard example configuration
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, url);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "RDSEA Simple Consumer");
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "30000");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.IntegerDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        consumer = new KafkaConsumer<Integer, String>(props);
        this.topic = topic;
        this.pollNr = pollNr;
    }

    public void readData() {
        consumer.subscribe(Collections.singletonList(this.topic));
        ConsumerRecords<Integer, String> records = consumer.poll(pollNr);
        for (ConsumerRecord<Integer, String> record : records) {
            System.out.println("Received message: (" + record.key() + ", " + record.value() + ") at offset " + record.offset());
        }
    }

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                if (args.length < 3) {
            System.out.println("Usage: SimpleProducer kafka_broker  topic nr");
            System.exit(0);
        }
                int pollNr =Integer.valueOf(args[2]);
        SimpleConsumer consumer = new SimpleConsumer(args[0], args[1], pollNr);
        consumer.readData();
        }

}
```

Scalability and Fault Tolerance

- Partitions are distributed and replicated among broker servers
- Consumers are organized into groups
- Each message is delivered to a consumer instance in a group
- One partition is assigned to one consumer



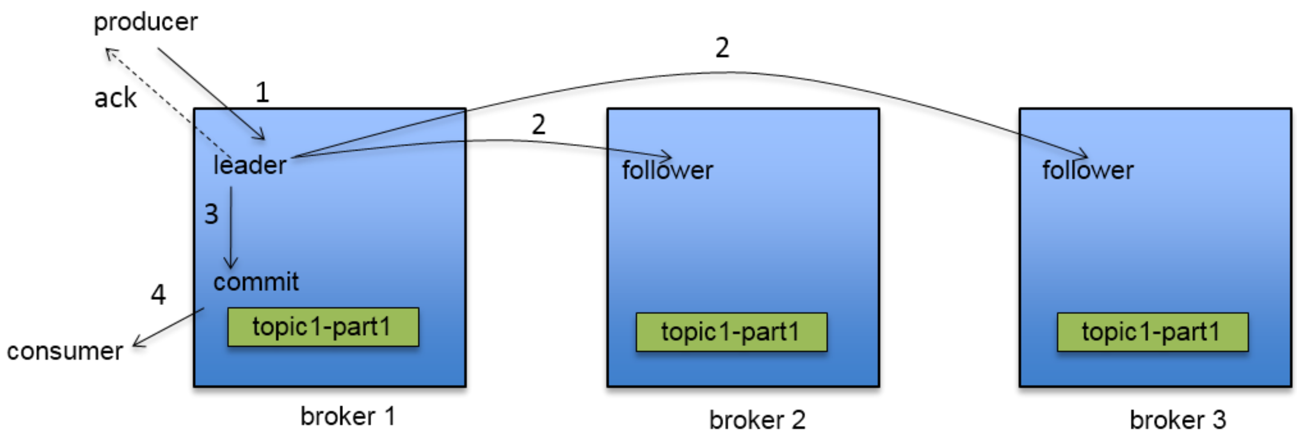
http://kafka.apache.org/documentation.html#majordesignelements

Partitions and partition replication

- Why partitions?
  - Support scalability
    - enable arbitrary data types and sizes for a topic
    - enable parallelism in producing and consuming data

- But partitions are replicated, why?
  - For fault tolerance

Partition Replication



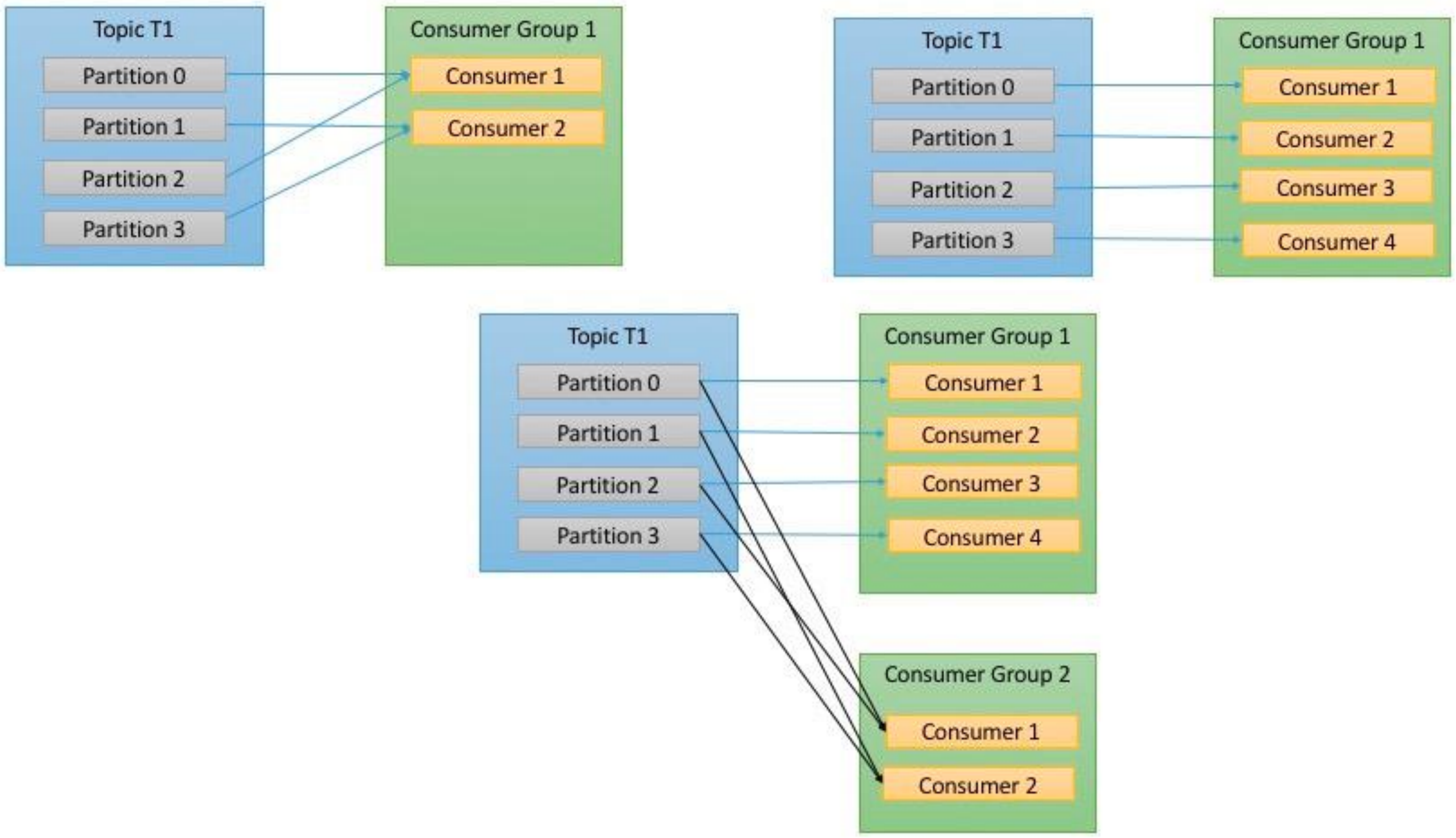
Source: http://de.slideshare.net/junrao/kafka-replication-apachecon2013

The leader handles all read and write requests

Consumer Group

- Consumer Group: a set of consumers
  - is used to support scalability and fault tolerance
  - allows multiple consumers to read a topic
- In one group: each partition is consumed by only consumer instance
  - Combine „queuing“ and „publish/subscribe“ model
- Enable different applications receive data from the same topic.
  - different consumers in different groups can retrieve the same data

Group rebalancing



Source: https://www.safaribooksonline.com/library/view/kafka-the-definitive/9781491936153/ch04.html

Key Questions/Thoughts

- Why do we need partitions per topic?

→ arbitrary data handling, ordering guarantees, load balancing

- How to deal with high volume of realtime events for online and offline consumers?

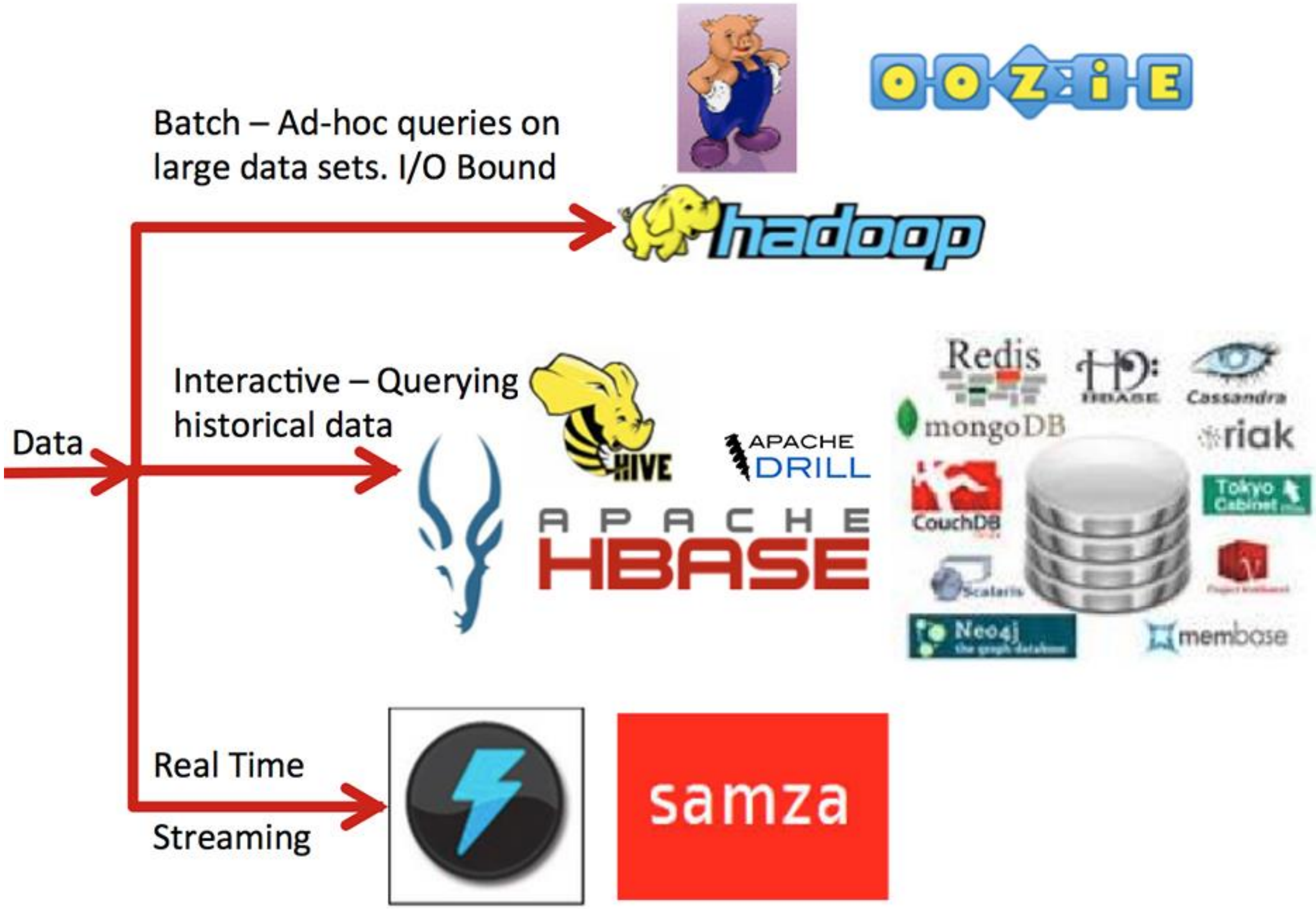
→ partition, cluster, message storage, batch retrieval, etc.

- Queuing or publish-subscribe model?

→ check how Kafka delivers messages to consumer instances/groups

STREAMING DATA PROCESSING

Batch, Stream and Interactive Analytics

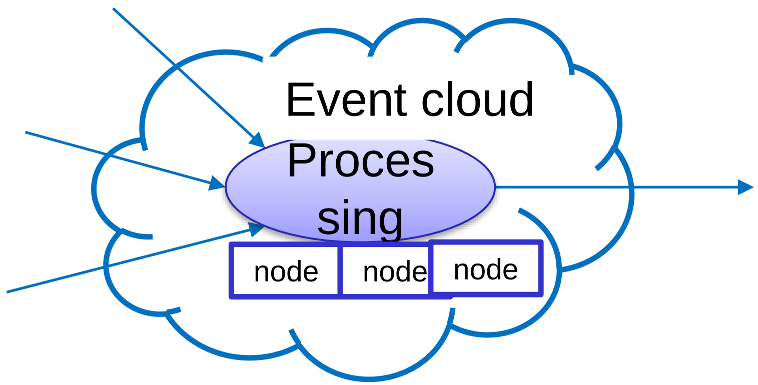


Source: https://dzone.com/refcardz/apache-spark

Recall: Centralized versus distributed processing topology

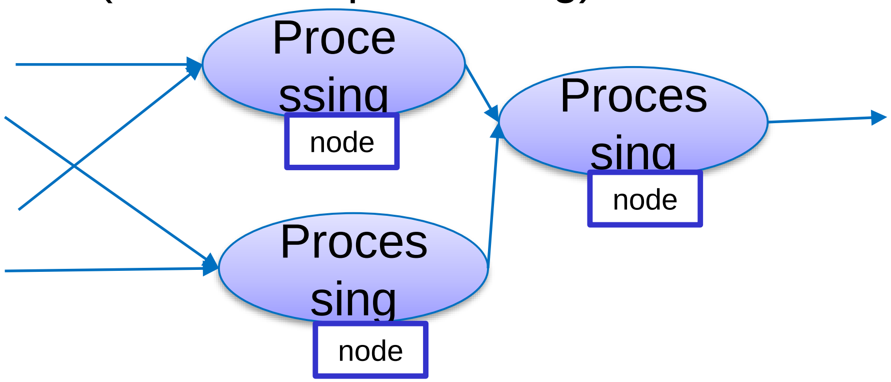
Two views: streams of events or cloud of events

Complex Event Processing
(centralized processing)



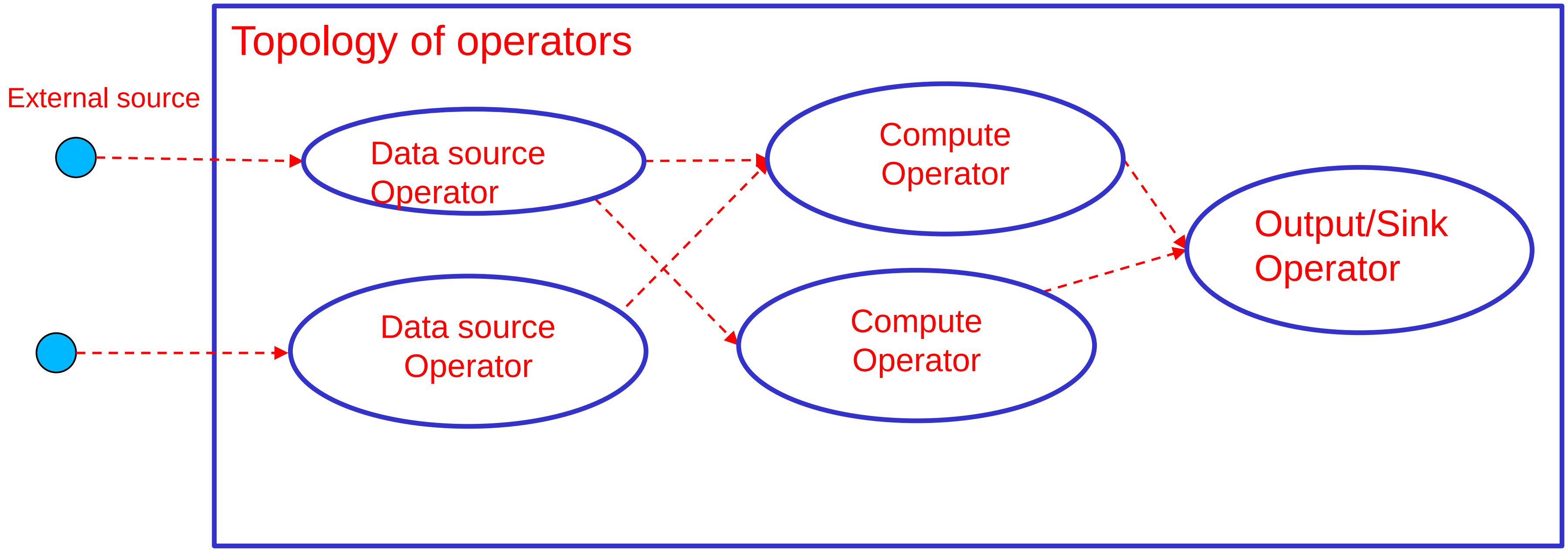
Usually only
queries/patterns are written

Streaming Data Processing
(distributed processing)



Code processing events and topologies need to be written

Structure of streaming data processing programs



- Data source operator: represents a source of streams
- Compute operators: represents processing functions
- *Native* versus *micro-batching*

Key concepts

- Structure of the data processing
  - Topology: Directed Acycle Graph (DAG) of operators
  - Data input/output operators and compute operators
  - Accepted various data sources through different connectors
- Scheduling and execution environments
  - Distributed tasks on multiple machines
  - Each machine can run multiple tasks
- Stream: connects an output port from an operator to an input port to another operator
- Stream data is sliced into windows of data for compute operators

Implementations

- Many implementation, e.g.
  - Apache Storm
    - https://storm.apache.org/
  - Apache Spark
    - https://spark.apache.org/
  - Apache Apex
    - https://apex.apache.org/

Check:
http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1

http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2

Apache Apex – Data Streams

- Data stream is the key abstraction

Recall:
Data stream: a sequence/flow of data units
Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

In Apache Apex: a stream of atomic data elements (tuples)

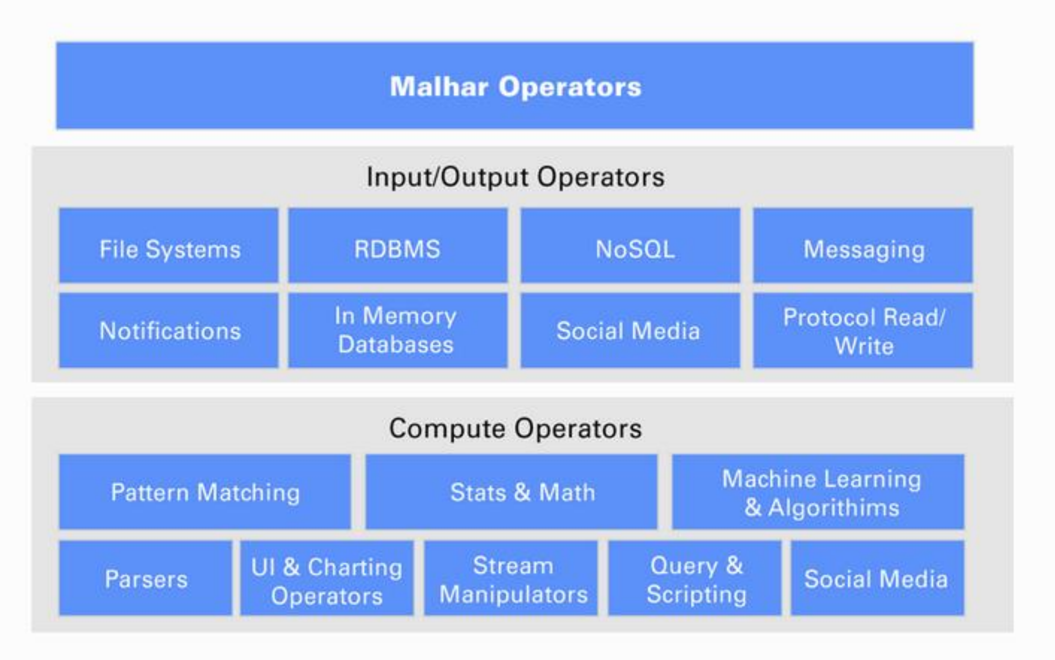
Example of an application in Java

```java
/**
 *
 * @author truong
 */
public class VietcontrolMQTTInput extends AbstractMqttInputOperator{
public final transient DefaultOutputPort<String> out;

    public VietcontrolMQTTInput() {
        this.out = new DefaultOutputPort<>();
        //out.emit("Test message");
    }
    @Override
    public void emitTuple(org.fusesource.mqtt.client.Message msg) {
        System.out.println("topic: "+msg.getTopic());
        byte[] data =msg.getPayload();
        String v = new String(data, Charset.forName("UTF-8") );
        System.out.println(v);
        out.emit(v);
    }

}
```

```java
@ApplicationAnnotation(name="MySecondApplication")
public class BTSApplication implements StreamingApplication
{
  String topic ="apextest";
  QoS qos;

    public BTSApplication() {
        this.qos = QoS.AT_MOST_ONCE;
    }
  @Override
  public void populateDAG(DAG dag, Configuration conf)
  {

      System.out.println("Start the application by connecting to MQTT");
      MqttClientConfig btsmqttConfig = new MqttClientConfig();

      btsmqttConfig.setHost("localhost");
      btsmqttConfig.setPort(1883);
      btsmqttConfig.setUserName("guest");
      btsmqttConfig.setPassword("guest");
      btsmqttConfig.setCleanSession(true);
      //creating input operator
      VietcontrolMQTTInput btsInput = dag.addOperator("input", VietcontrolMQTTInput.class);
      btsInput.setMqttClientConfig(btsmqttConfig);
      System.out.println("Subscribe topics");
      btsInput.addSubscribeTopic(topic, qos);
      //just a simple example to output the data to the console
      ConsoleOutputOperator cons = dag.addOperator("console", new ConsoleOutputOperator());
      cons.setSilent(false);
      System.out.println("Just create one single stream");
      dag.addStream("test", btsInput.out, cons.input).setLocality(Locality.CONTAINER_LOCAL);

  }
}
```

Apex - Operators

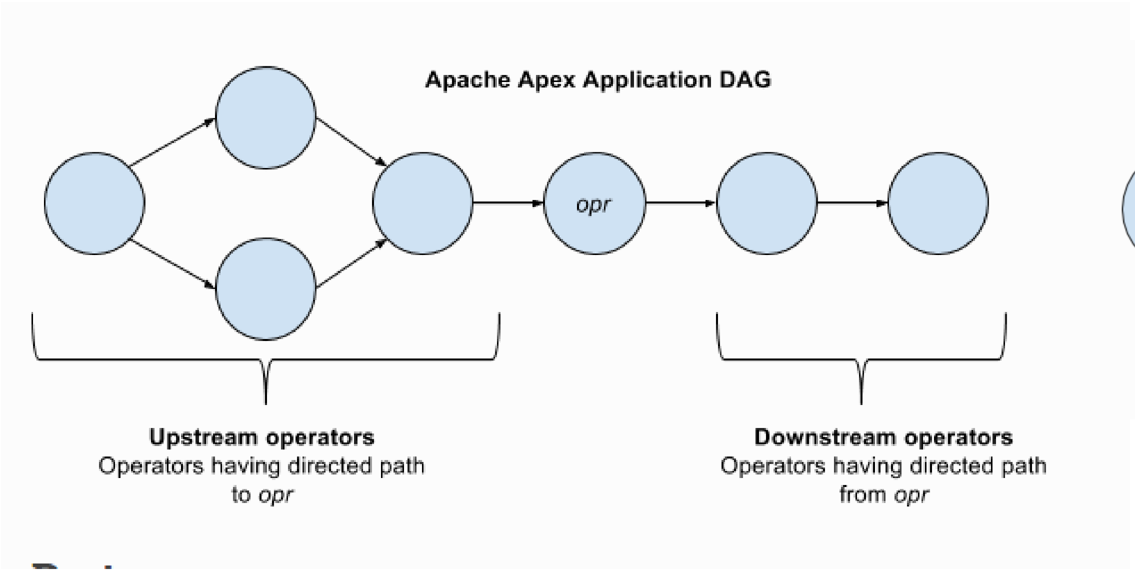
- Streaming applications are built with a set of operators: for data and computation

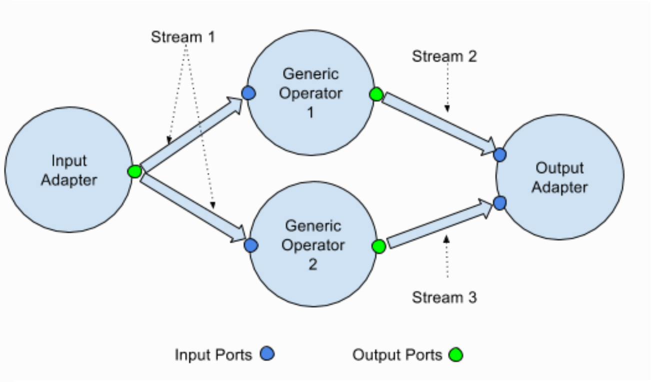


Source: https://apex.apache.org/docs/malhar/

- Some common data operators (related to other lectures)
  - MQTT
  - AMQP
  - Kafka

Apex Operators





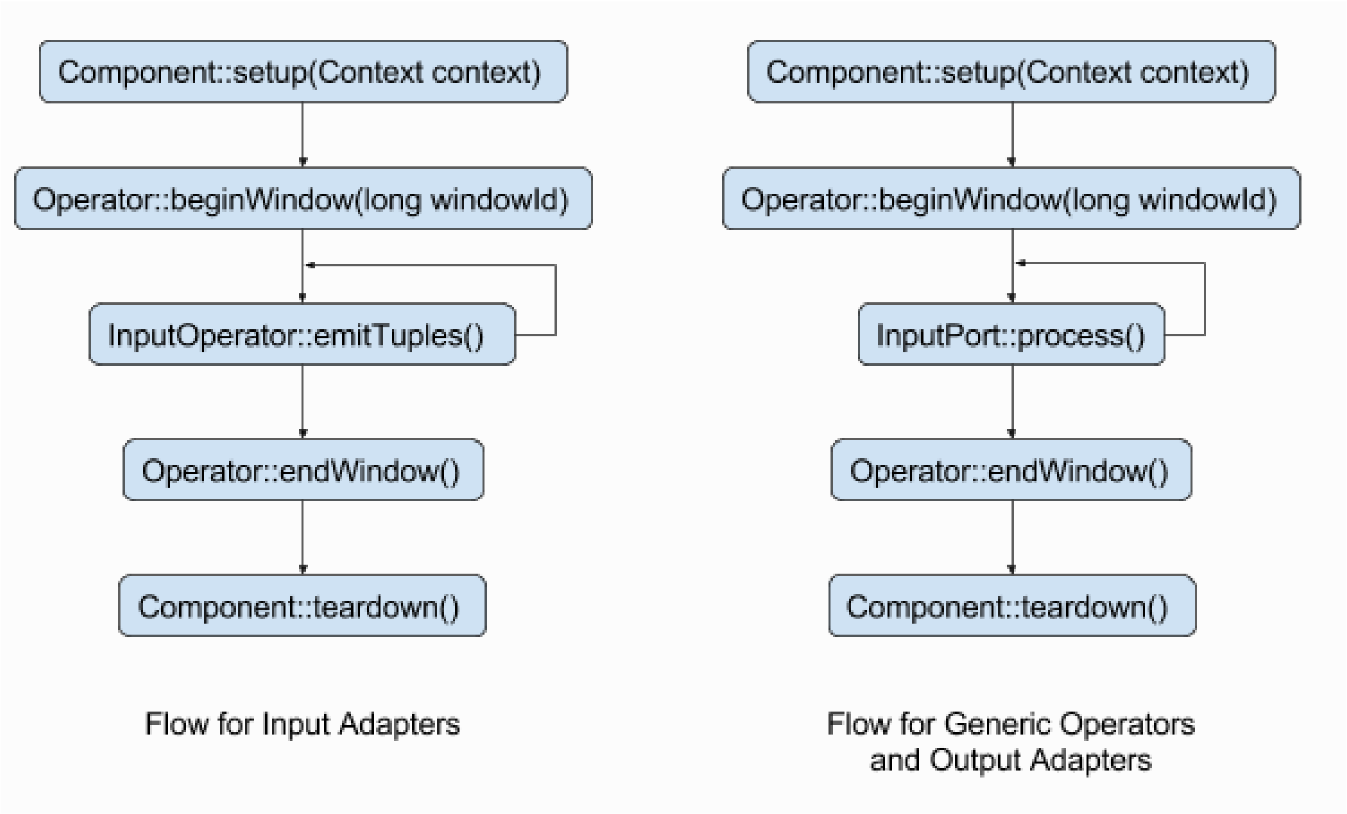
Source: https://apex.apache.org/docs/apex-3.6/operator_development/

- Ports: for input and output data
- Data in a stream: streaming windows

Distributed Systems Group

Processing data in operators

Different types of Windows: GlobalWindows, TimeWindows, SlidingTimeWindows, etc.



Source: https://apex.apache.org/docs/apex/operator_development/

Operators Fault tolerance

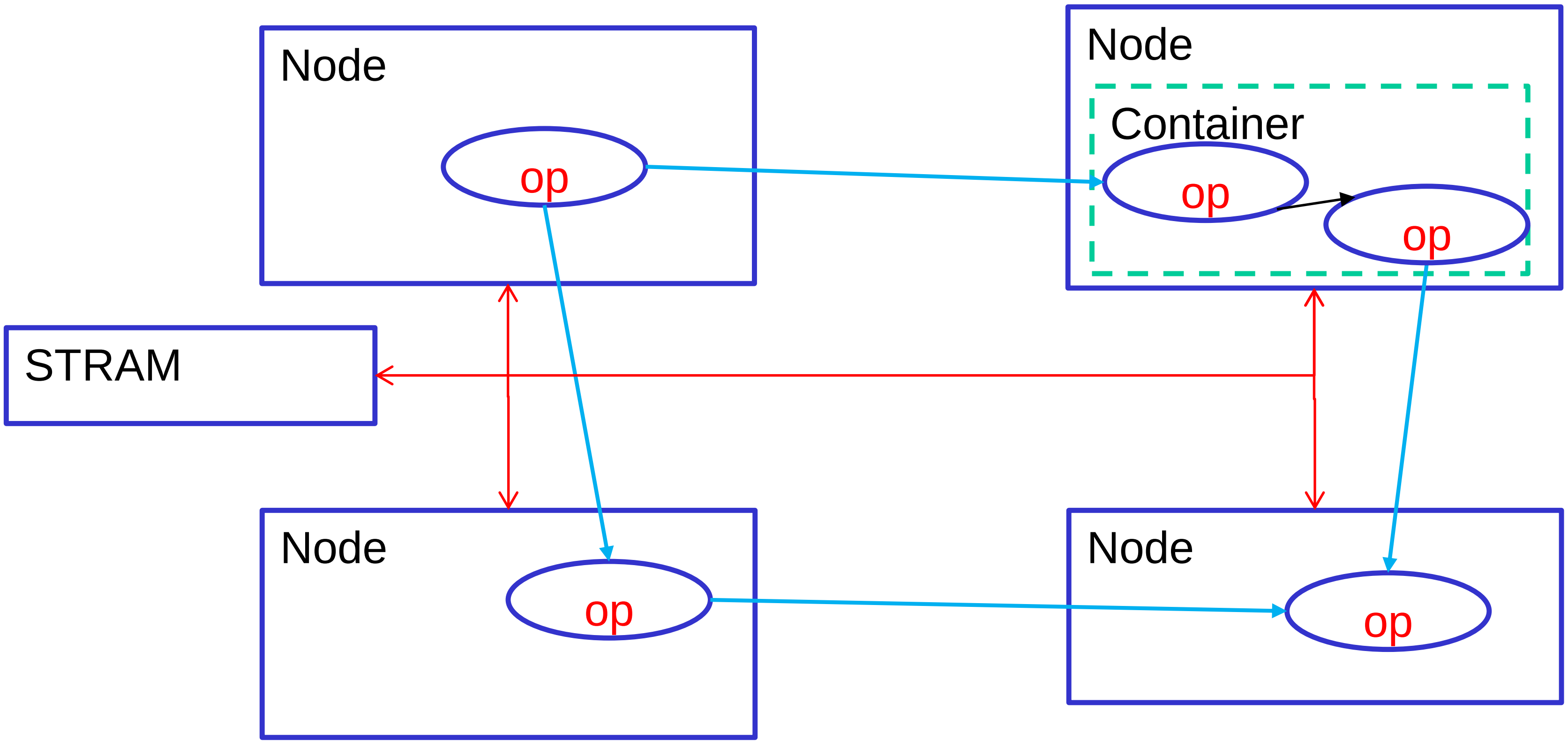
- Checkpoint of operators: save state of operators (e.g. into HDFS)
  - @Stateless no checkpoint
  - Check point interval: CHECKPOINT_WINDOW_COUNT
- Recovery
  - At least once
  - At most once
  - Exactly once

Fault tolerance - Recovery

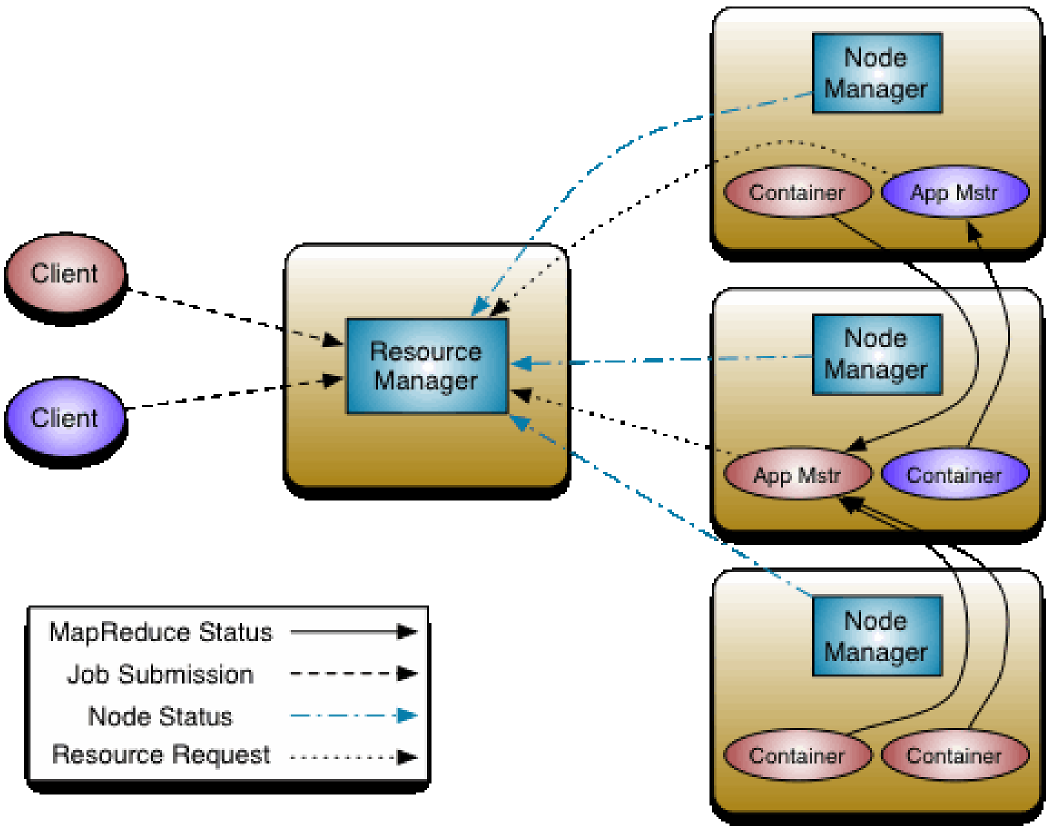
- At least once
  - Downstream operators are restarted
  - Upstream operators are replayed
- At most once
  - Assume that data can be lost: restart the operator and subscribe to new data from upstream
- Exactly once
  - https://www.datatorrent.com/blog/end-to-end-exactly-once-with-apache-apex/

Execution Management

- Using YARN for execution tasks
- Using HDFS for persistent state



Understand YARN/Hadoop to understand Apex operator execution management



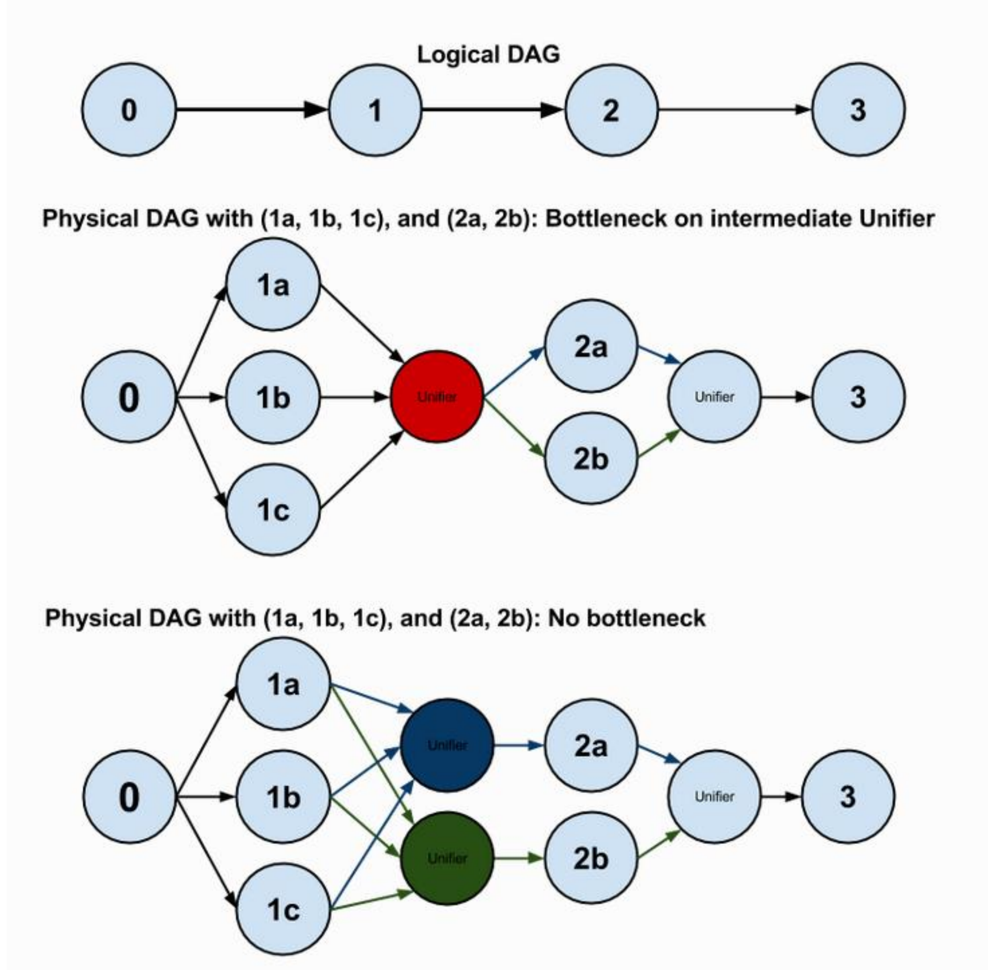
Source: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Scalability

- Locality configuration for deployment of streams and operators
- Affinity and anti-affinity rules
- Possible localities:
  - THREAD_LOCAL (intra-thread)
  - CONTAINER_LOCAL (intra-process)
  - NODE_LOCAL (inter-process but within a Hadoop node)
  - RACK_LOCAL (inter-node)

Example of Partitioning and unification

- Dynamic Partition
  - Partition operators
  - Dynamic: specifying when a partition should be done
  - Unifiers for combining results (reduce)
- StreamCodec
  - For deciding which tuples go to which partitions
  - Using hashcode and masking mechanism



Source: https://apex.apache.org/docs/apex/application_development/#partitioning

Exercise

How to make sure no duplication results when we recover End-to-End Exactly Once?

How to use hash and masking mechanism to distributed tuples?

How to deal with data between operators not in a CONTAINER_LOCAL or in THREAD_LOCAL

ADVANCED WORKFLOWS/DATA PIPELINE PROCESSING

Use cases

- Access and coordinate many different compute services, data sources, deployment services, etc, within an enterprise, for a particular goal
- Implementing complex „business logics" of your services
- Analytics-as a service: metrics, user activities analytics, testing, e.g.
  - Analytics of log files (generated by Aspects in Lecture 3)
  - Dynamic analytics of business activities

Workflow and Pipeline/data workflow

- Workflows: a set of coordinated activities
  - Generic workflows of different categories of tasks
  - Data workflows → data pipeline

    „a pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one"

    Source: https://en.wikipedia.org/wiki/Pipeline_%28computing%29
- We use a pipeline/data workflows to carry out a data processing job
- But analytics have many more than just data processing activities.

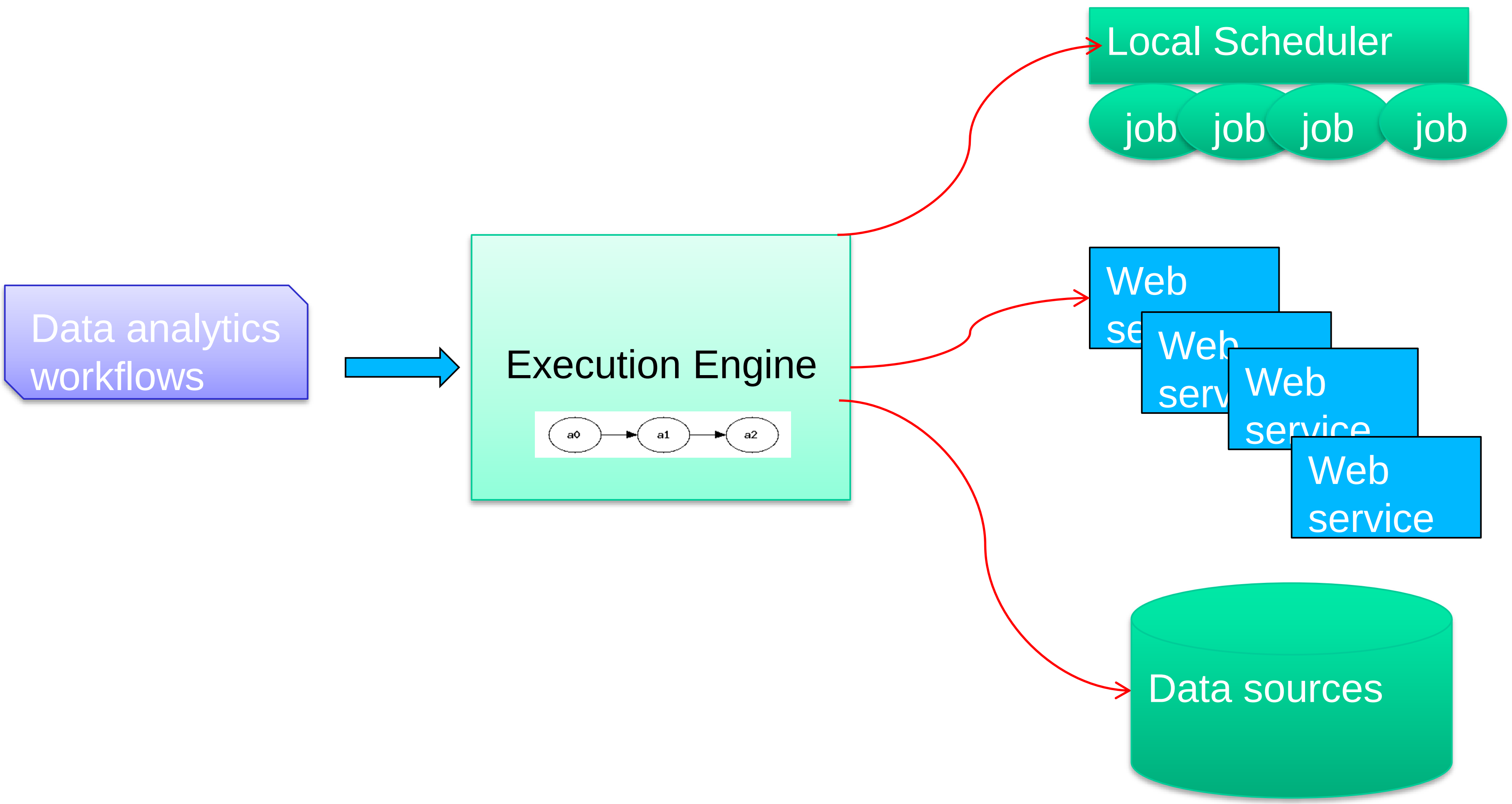
Example of Pipeline in Google Dataflow

JAVA

```
public static void main(String[] args) {
  // Create a pipeline parameterized by commandline flags.
  Pipeline p = Pipeline.create(PipelineOptionsFactory.fromArgs(arg));

  p.apply(TextIO.Read.from("gs://..."))   // Read input.
   .apply(new CountWords())               // Do some processing.
   .apply(TextIO.Write.to("gs://..."));   // Write output.

  // Run the pipeline.
  p.run();
}
```

https://cloud.google.com/dataflow/model/pipelines#a-simple-example-pipeline

Data analytics workflow execution models



Your are in a situation:

- Many underlying distributed processing frameworks
  - Apex, Spark, Flink, Google
- Work with different underlying engines
- Write only high-level pipelines
- Stick to your favour programming languages

Apache Beam

- Goal: separate from pipelines from backend engines

Read data analytics → Post-processing result → Store analysis result

Apache Apex™ | Flink | Apache Spark™ | Dataflow

Appache Beam

- https://beam.apache.org/
- Suitable for data analysis processes that can be divided into different independent tasks
  - ETL (Extract, Transform and Load)
  - Data Integration
- Execution principles:
  - Mapping tasks in the pipeline to concrete tasks that are supported by the selected back-end engine
  - Coordinating task execution like workflows.

Basic programming constructs

- Pipeline:
  - For creating a pipeline
- PCollection
  - Represent a distributed dataset
- Transform

  [Output PCollection] = [Input PCollection] | [Transform]

  - Possible transforms: ParDo, GroupByKey, Combine, etc.

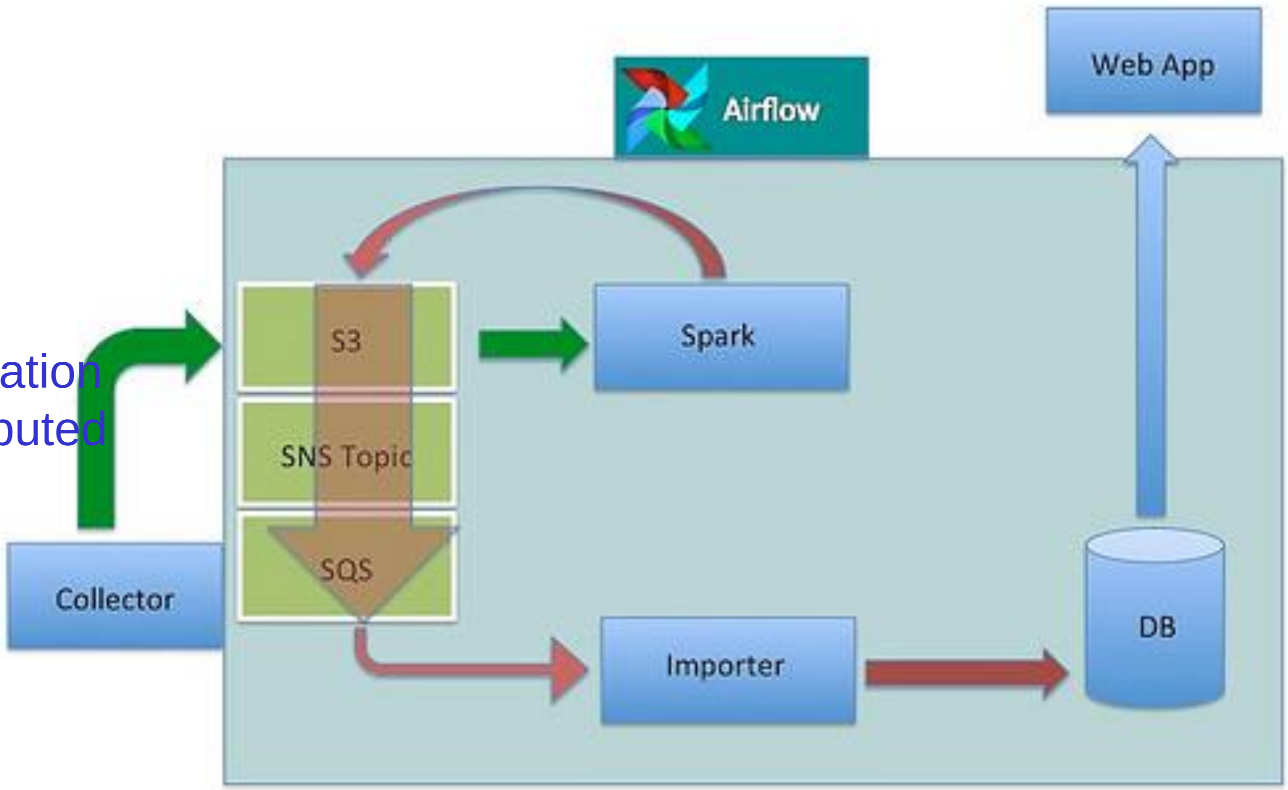
A simple example with Google Dataflow as back-end engine

```
import apache_beam as beam
from apache_beam.options.pipeline_options import PipelineOptions

p = beam.Pipeline(options=PipelineOptions())

entries = p | 'ReadHadoopResult' >> beam.io.ReadFromText('gs://.../ElectricityAlarm
/electricity_alarm_frequency-2017-05-11-00-vn.csv')
class ExtractAlarmFrequency(beam.DoFn):
  def process(self, elements):
    ....
    return ....
frequency = entries| beam.ParDo(ExtractAlarmFrequency())
frequency | 'write' >> beam.io.WriteToText('gs://.../ElectricityAlarm')
result = p.run()
result.wait_until_finish()
```

But what if you need diverse types of tasks with various back-end services?

→ Workflow systems

Example of using workflows



Security-related information and metrics from distributed customers

Source: http://highscalability.com/blog/2015/9/3/how-agari-uses-airbnbs-airflow-as-a-smarter-cron.html

Representing and programming workflows/data workflows

- Programming languages
  - General- and specific-purpose programming languages, such as Java, Python, Swift
- Descriptive languages
  - BPEL and several languages designed for specific workflow engines

Airflow from Airbnb

- http://airbnb.io/projects/airflow/

- Features
  - Dynamic, extensible, scalable workflows
  - Programmable language based workflows
    - Write workflows as programmable code

- Good and easy to study to understand concepts of workflows/data pipeline

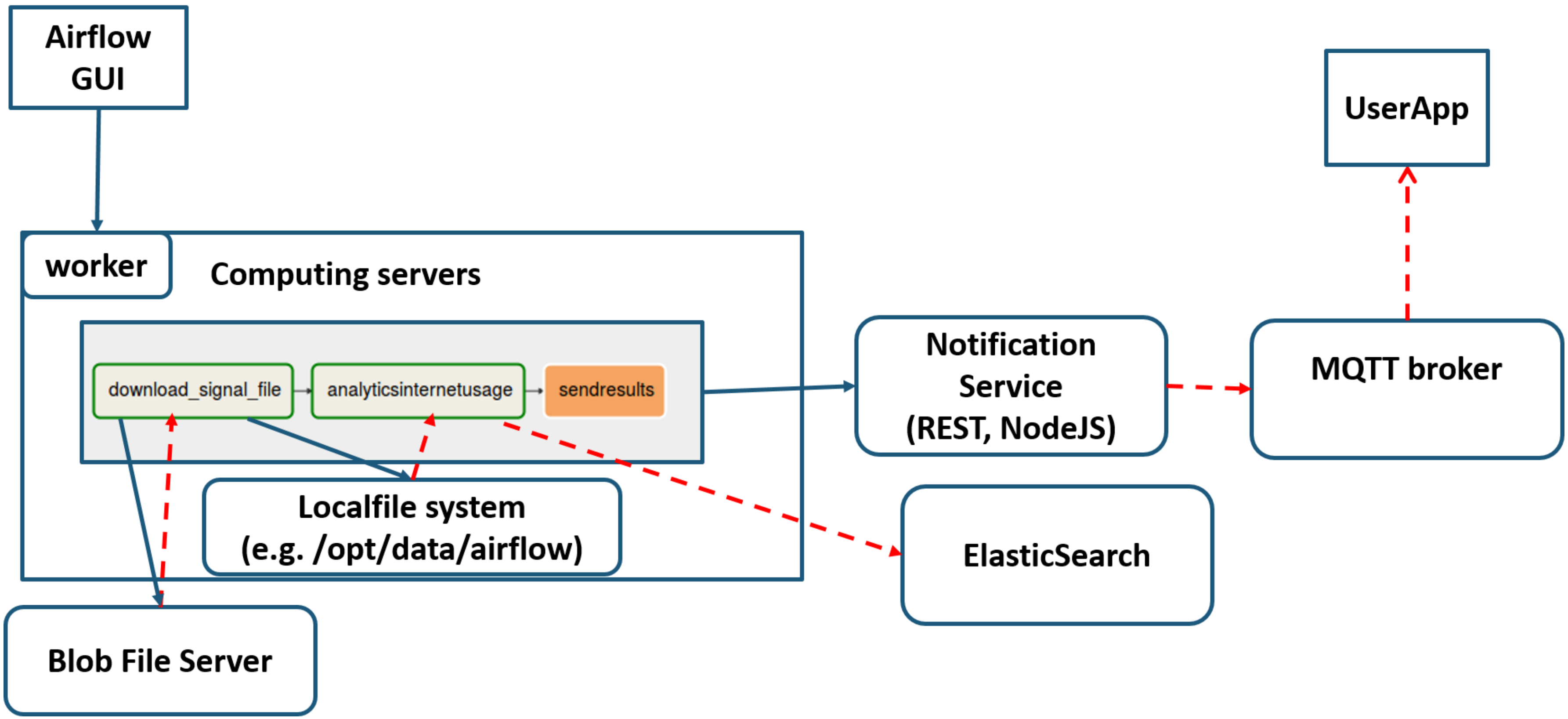
Airflow Workflow structure

- Workflow is a DAG (Direct Acyclic Graph)
  - A workflow consists of a set of activities represented in a DAG
  - Workflow and activities are programed using Python – described in code
- Workflow activities are described by Airflow operator objects
  - Tasks are created when instantiating operator objects

Airflow from Airbnb

- Rich set of operators
  - So that we can program different kinds of tasks and integrate with different systems
- Different Types of operators for workflow activities
  - BashOperator, PythonOperator, EmailOperator, HTTPOperator, SqlOperator, Sensor,
  - DockerOperator, HiveOperator, S3FileTransferOperator, PrestoToMysqlOperator, SlackOperator

Example for processing signal file



Example for processing signal file

```
DAG_NAME = 'signal_upload_file'

default_args = {
    'owner': 'hong-linh-truong',
    'depends_on_past': False,
    'start_date': datetime.now(),
}

dag = DAG(DAG_NAME, schedule_interval=None, default_args=default_args)


stations=["station1", "station2"]

def checkSituation(**kwargs):
    f = 'f'
    t = 't'
    return t

downloadlogscript="curl  file:///home/truong/myprojects/mygit/rdsea-mobifone-training/data/opensignal/sample-Oct182016.csv -o /opt/data/air

t_downloadlogtocloud=  BashOperator(
    task_id="download_signal_file",
    bash_command=downloadlogscript,
    dag = dag
    )


t_analytics=  BashOperator(
    task_id="analyticsinternetusage",
    bash_command="/usr/bin/python /home/truong/myprojects/mygit/rdsea-mobifone-training/examples/databases/elasticsearch/uploader/src/uploa
    dag = dag
    )
t_sendresult =SimpleHttpOperator(
    task_id='sendresults',
    method='POST',
    http_conn_id='station1',
    endpoint='api/update/credit',
    data=json.dumps({"userphone": "066412345","credit":10}),
    headers={"Content-Type": "application/json"},
    dag = dag
    )

t_analytics.set_upstream(t_downloadlogtocloud)
t_sendresult.set_upstream(t_analytics)
```

Example

Airflow | DAGs | Data Profiling | Browse | Admin | Docs | 15:14 UTC

DAGs

Show entries Search:

		DAG	Schedule	Owner	Recent Statuses	Links
	Off	example_bash_operator	0 0 * * *	airflow		
	Off	example_branch_dop_operator_v3	*/1 * * * *	airflow		
	Off	example_branch_operator	@daily	airflow		
	Off	example_http_operator	1 day, 0:00:00	airflow		
	Off	example_passing_params_via_test_command	*/1 * * * *	airflow		
	Off	example_python_operator	None	airflow		
	Off	example_short_circuit_operator	1 day, 0:00:00	airflow		
	Off	example_skip_dag	1 day, 0:00:00	airflow		
	Off	example_subdag_operator	@once	airflow		
	Off	example_trigger_controller_dag	@once	airflow		
	Off	example_trigger_target_dag	None	airflow		
	Off	example_twitter_dag	@daily	Ekhtiar		
	Off	example_xcom	@once	airflow		
	Off	signal_upload_file	None	hong-linh-truong		
	Off	tutorial	1 day, 0:00:00	airflow		

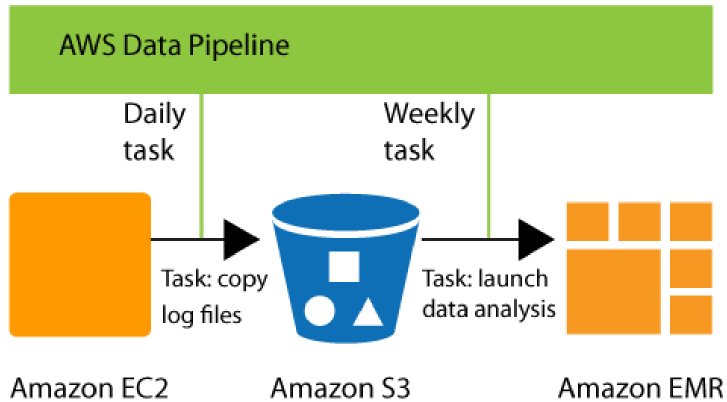
Showing 1 to 15 of 15 entries

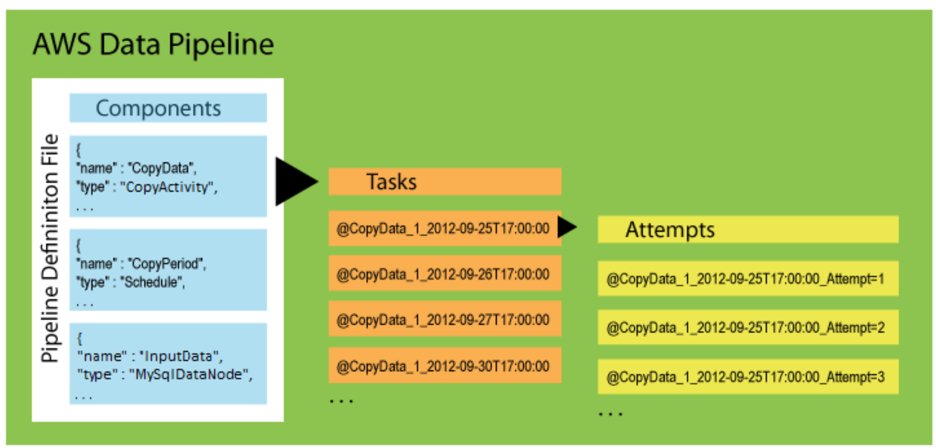
Previous 1 Next

Elasticity control for Workflows/Data Flows

- How to scale the workflows?
- Scheduling in a large resource pool (e.g., using clusters)
- Elasticity controls of virtualized resources (VMs/containers) for executing tasks
- Distributed Task Queue, e.g. Celery
    - http://docs.celeryproject.org/en/latest/getting-started/brokers/index.html
    - Job description/request sent via queues
    - Results from jobs can be stored in some back-end

Other systems, e.g., AWS Data Pipeline





http://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide

Summary

- Analytics-as-a-service for large-scale distributed applications and big data analytics require different set of tools
- Kafka, Apache Apex and Airflow are just some of the key frameworks
  - There are a lot of tools
- Need to understand common concepts and distinguishable features
- Select them based on your use cases and application functionality and performance requirements
- Exercises:
  - a small application utilizing Kafka/MQTT and Apache Apex
  - Log analytics using AOP and Kafka and Airflow

Further materials

- http://kafka.apache.org
- http://www.corejavaguru.com/bigdata/storm/stream-groupings
- https://cloud.google.com/dataflow/docs/
- http://storm.apache.org/
- https://azure.microsoft.com/en-us/documentation/articles/hdinsight-storm-iot-eventhub-documentdb/

https://storm.apache.org/

STREAMING ANALYSIS WITH APACHE STORM

Apache Storm – Key concepts

- Originally from Twitter
- Data
- Structure of the data processing
  - Topology
  - Spouts
  - Bolts
  - Stream groupings
- Scheduling and execution environments
  - Processes, Executors and Tasks

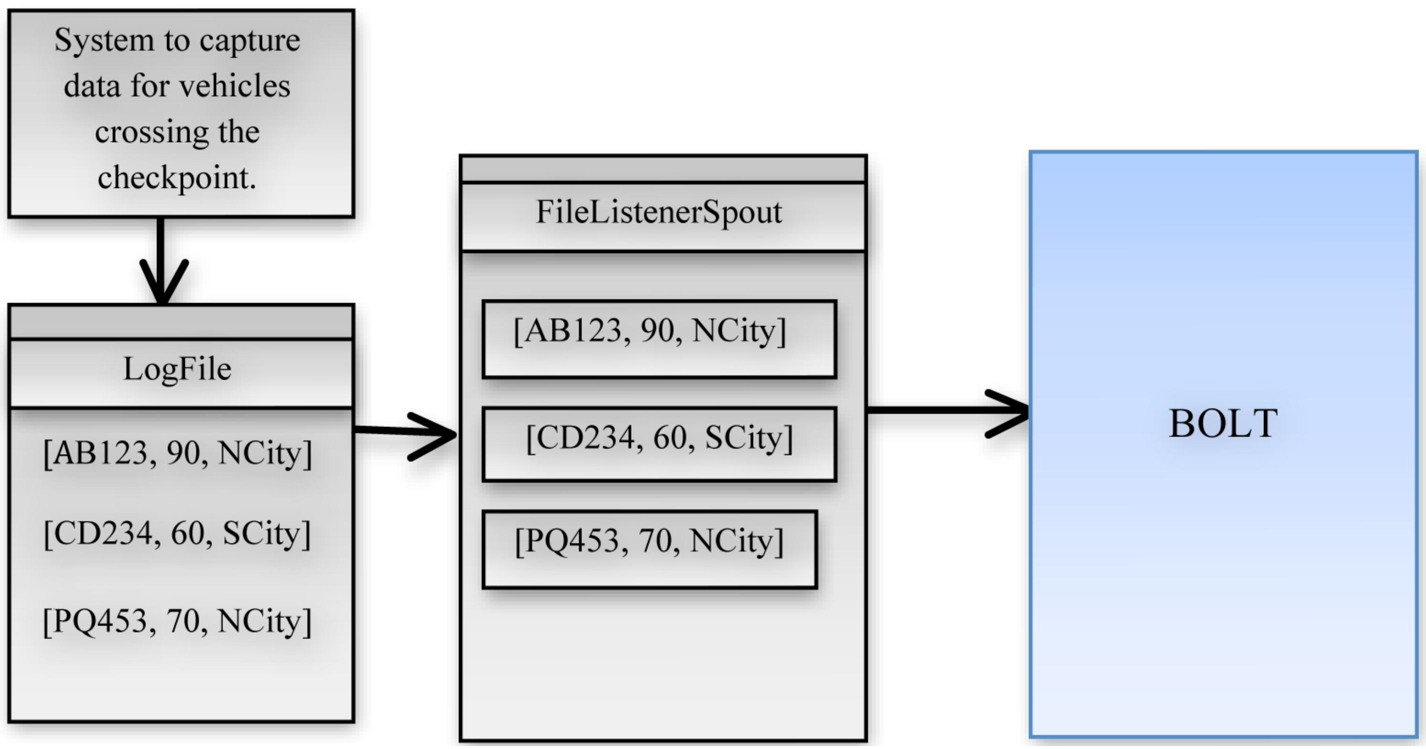
Apache Storm – Data Streams

- Data stream is the key abstraction

Recall:
Data stream: a sequence/flow of data units
Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

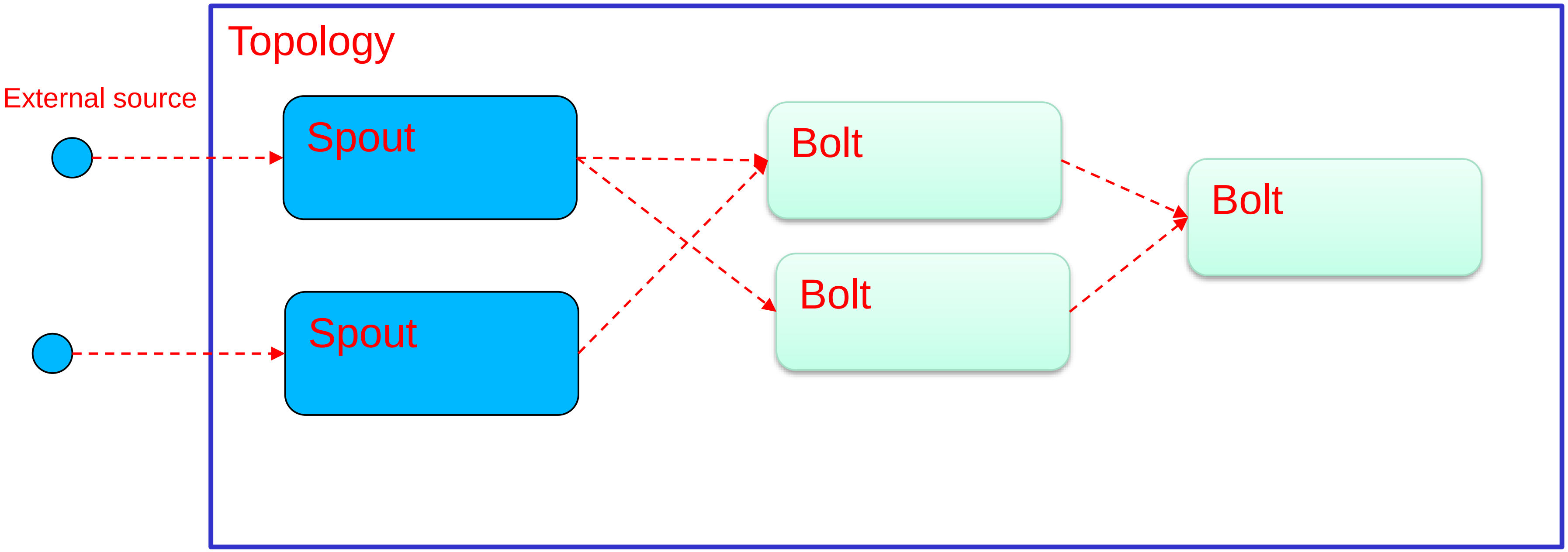
Apache Storm: a stream is „n unbounded sequence of tuples" → data units = tuples

Example of data stream



Source: http://www.drdobbs.com/open-source/easy-real-time-big-data-analysis-using-s/240143874

Structure of data processing program



- Spout: represents a source of streams
  - Read tuples from a external source and feed the tuples to the topology
- Bolt: represents processing functions

Spouts and Bolts

Spouts

- Can emit multiple streams
- unreliable/reliable
- Main APIs

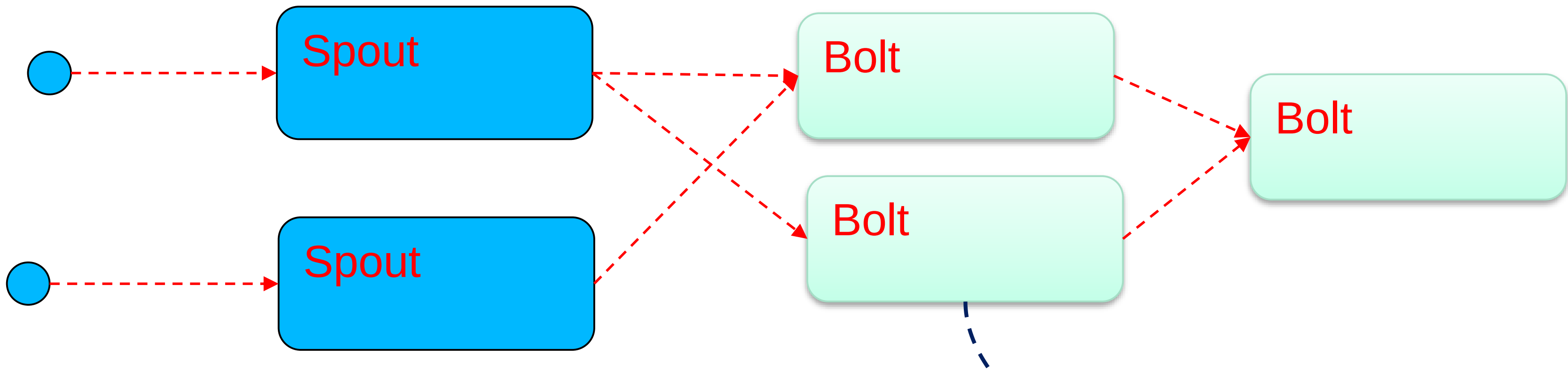
```
nextTuple()
fail(Object msgId)
ack(Object msgId)
```

Bolts

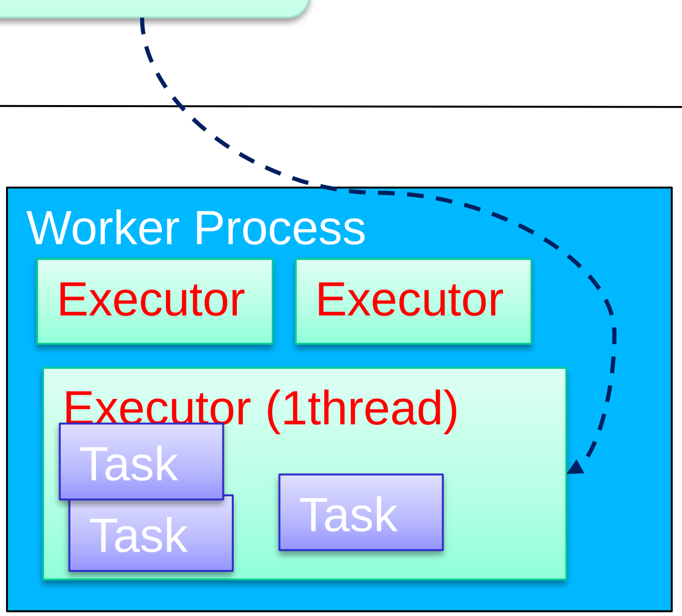
- Can emit multiple streams
- Main methods

```
execute(Tuple input)
prepare(Map stormConf,
TopologyContext
context,
OutputCollector
collector)
```

Structure of data processing program

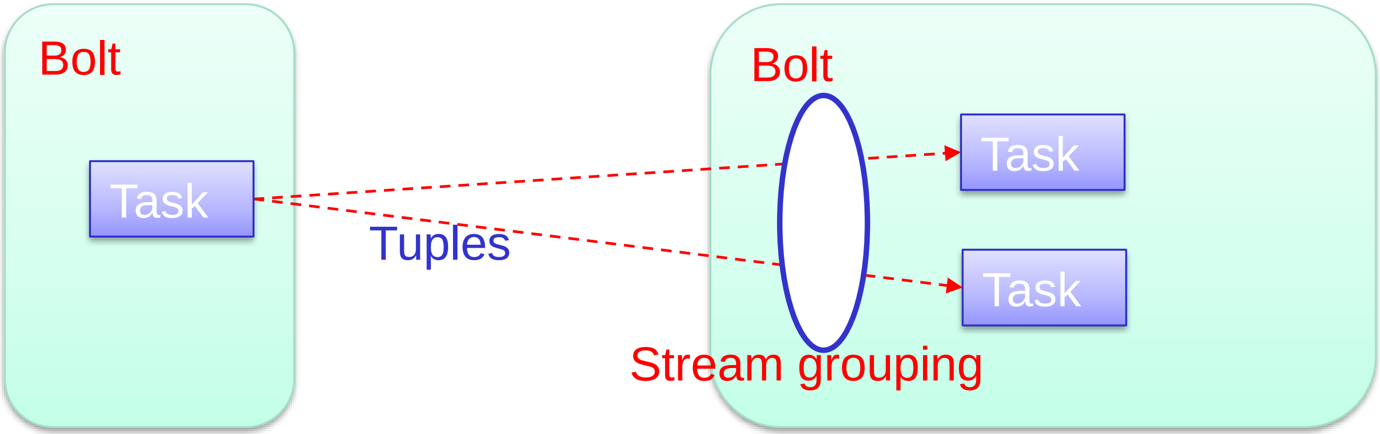



```
setSpout(String id,
   IRichSpout spout, Number
   parallelism_hint)
setBolt(String id,
   IRichBolt bolt, Number
   parallelism_hint)
```



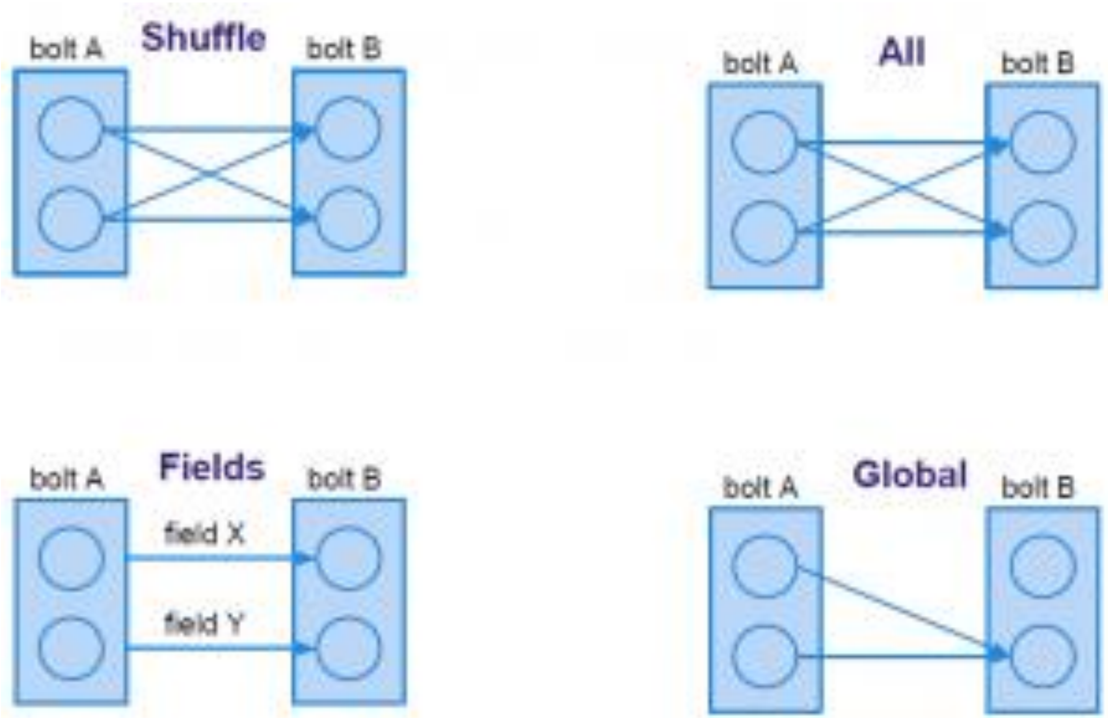
Runtime

Stream Grouping (1)



- Stream grouping defines how tuples are streamed to Tasks in Bolts
- Examples:
  - Shuffle grouping, Fields grouping, Partial Key grouping, All grouping, Global grouping, None grouping, Direct grouping, Local or shuffle grouping

Stream grouping (2)



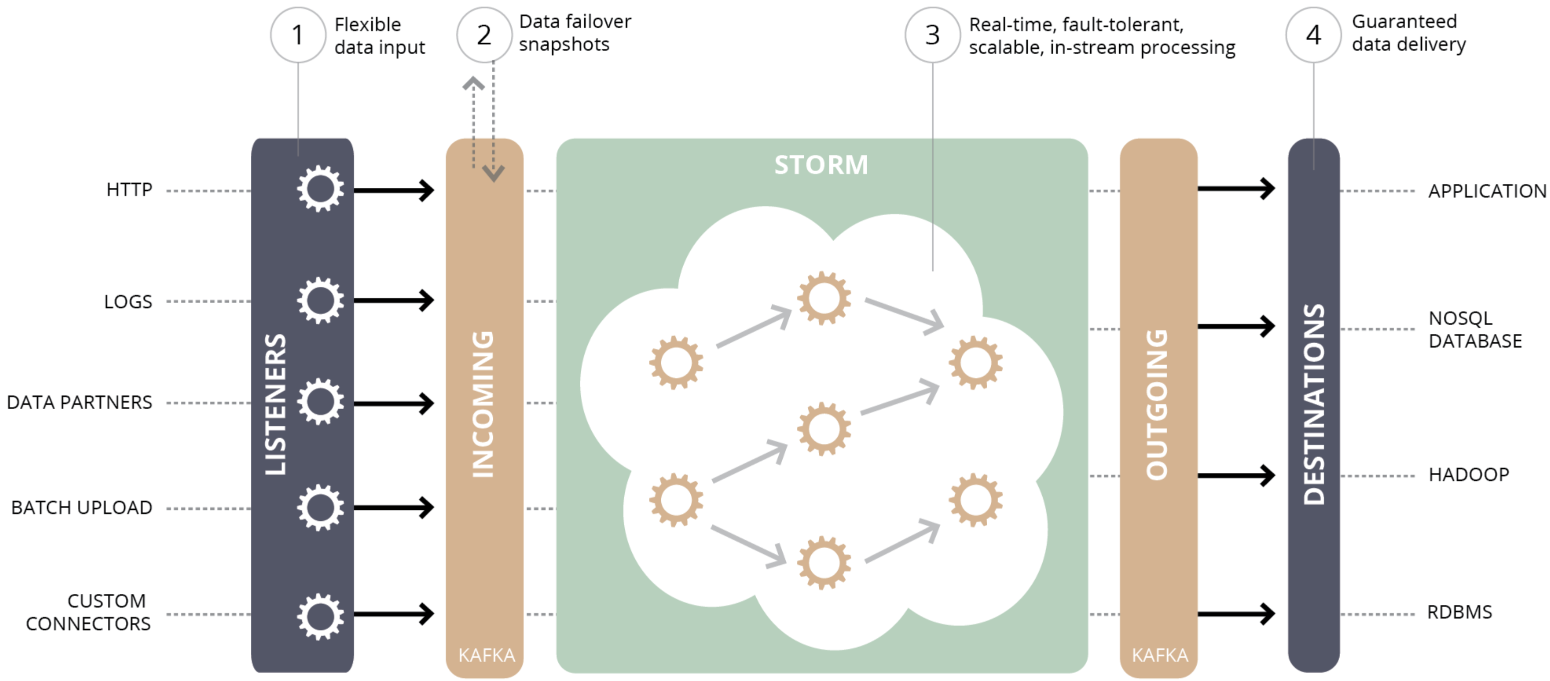
Soure: https://www.safaribooksonline.com/blog/2013/06/11/your-guide-to-storm/

Example of programming stream grouping

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("spout", new RandomSentenceSpout(),
   5);
builder.setBolt("split", new SplitSentence(),
   8).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(),
   12).fieldsGrouping("split", new Fields("word"));
```

Source: https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.3.4/bk_storm-user-guide/content/storm-stream-groupings.html

Integration example



http://blog.infochimps.com/2012/10/30/next-gen-real-time-streaming-storm-kafka-integration/

Thanks for your attention

Hong-Linh Truong
Distributed Systems Group, TU Wien
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong
@linhsolar