Building Streaming Applications with Apache Apex

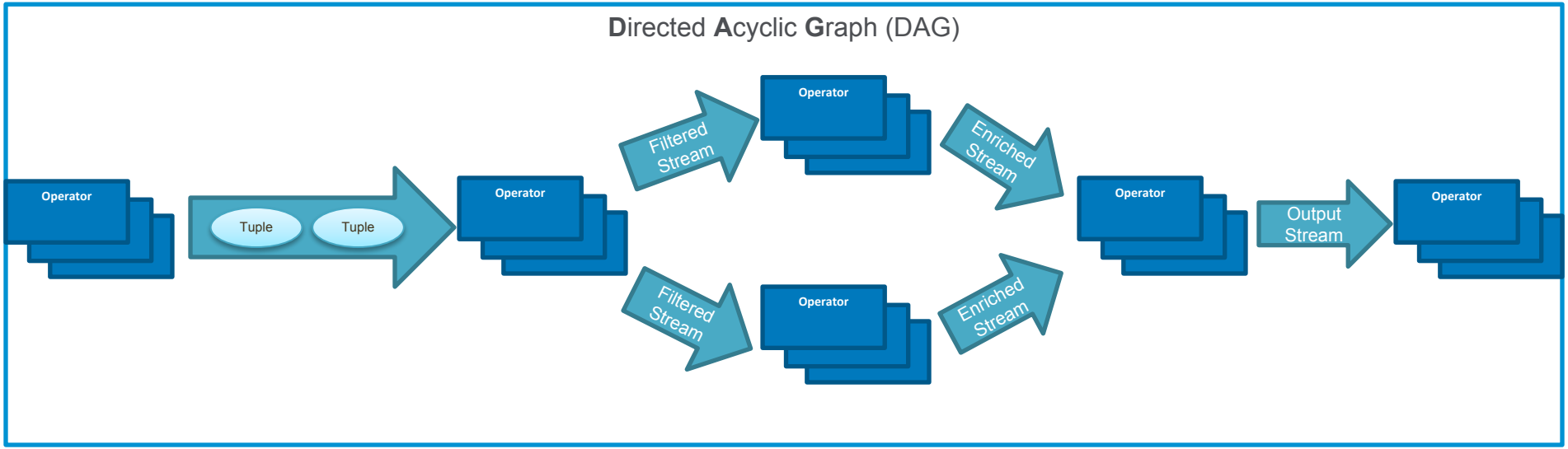
Chinmay Kolhatkar, Committer @*ApacheApex*, Engineer @*DataTorrent*
Thomas Weise, PMC Chair @*ApacheApex*, Architect @*DataTorrent*

Nov 15th 2016

Agenda

- Application Development Model
- Creating Apex Application - Project Structure
- Apex APIs
- Configuration Example
- Operator APIs
- Overview of Operator Library
- Frequently used Connectors
- Stateful Transformation & Windowing
- Scalability - Partitioning
- End-to-end Exactly Once

Application Development Model



- **Stream** is a sequence of data tuples
- **Operator** takes one or more input streams, performs computations & emits one or more output streams
  - Each Operator is YOUR custom business logic in java, or built-in operator from our open source library
  - Operator has many instances that run in parallel and each instance is single-threaded
- **Directed Acyclic Graph (DAG)** is made up of operators and streams

Creating Apex Application Project


```
chinmay@chinmay-VirtualBox:~/src$ mvn archetype:generate -DarchetypeGroupId=org.apache.apex
-DarchetypeArtifactId=apex-app-archetype -DarchetypeVersion=LATEST -DgroupId=com.example
-Dpackage=com.example.myapexapp -DartifactId=myapexapp -Dversion=1.0-SNAPSHOT
…
…
...
Confirm properties configuration:
groupId: com.example
artifactId: myapexapp
version: 1.0-SNAPSHOT
package: com.example.myapexapp
archetypeVersion: LATEST
 Y: : Y
…
…
...
[INFO] project created from Archetype in dir: /media/sf_workspace/src/myapexapp
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 13.141 s
[INFO] Finished at: 2016-11-15T14:06:56+05:30
[INFO] Final Memory: 18M/216M
[INFO] ------------------------------------------------------------------------
chinmay@chinmay-VirtualBox:~/src$
```

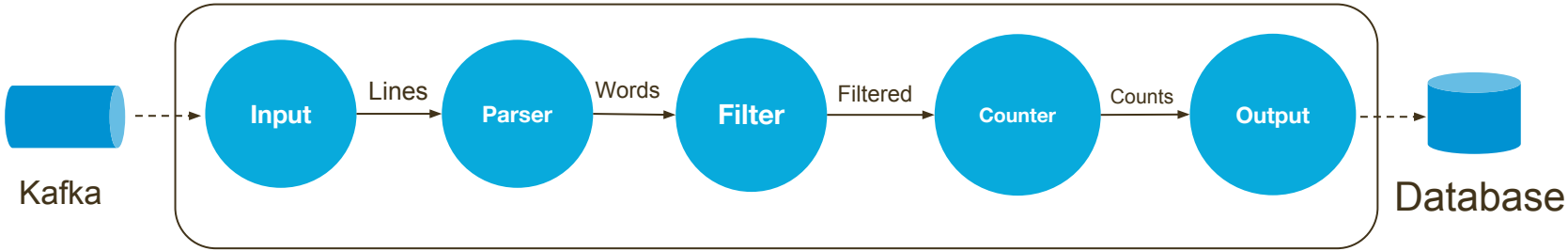
https://www.youtube.com/watch?v=z-eeh-tjQrc

Apex Application Project Structure

```
chinmay@DevEnv:~/src/myapexapp$ tree
.
├── pom.xml
├── src
│   ├── assemble
│   │   └── appPackage.xml
│   ├── main
│   │   ├── java
│   │   │   └── com
│   │   │       └── example
│   │   │           └── myapexapp
│   │   │               ├── Application.java
│   │   │               └── RandomNumberGenerator.java
│   │   └── resources
│   │       └── META-INF
│   │           └── properties.xml
│   ├── site
│   │   └── conf
│   │       └── my-app-conf1.xml
│   └── test
│       ├── java
│       │   └── com
│       │       └── example
│       │           └── myapexapp
│       │               └── ApplicationTest.java
│       └── resources
│           └── log4j.properties
└── XmlJavadocCommentsExtractor.xsl

17 directories, 9 files
chinmay@DevEnv:~/src/myapexapp$
```

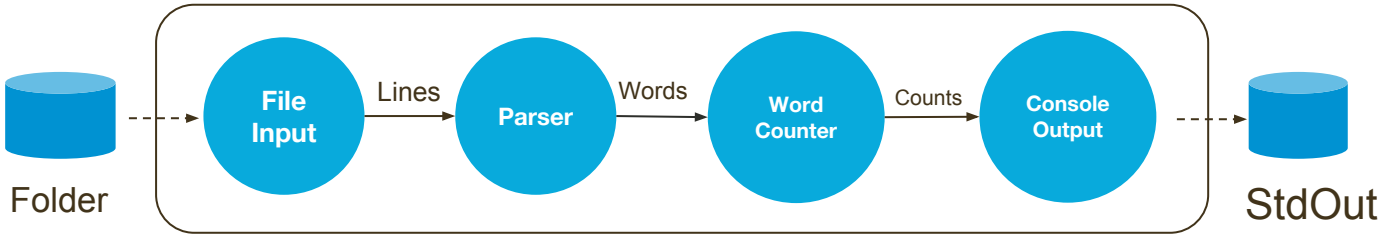
- *pom.xml*
  - Defines project structure and dependencies
- *Application.java*
  - Defines the DAG
- *RandomNumberGenerator.java*
  - Sample Operator
- *properties.xml*
  - Contains operator and application properties and attributes
- *ApplicationTest.java*
  - Sample test to test application in local mode

Apex APIs: Compositional (Low level)



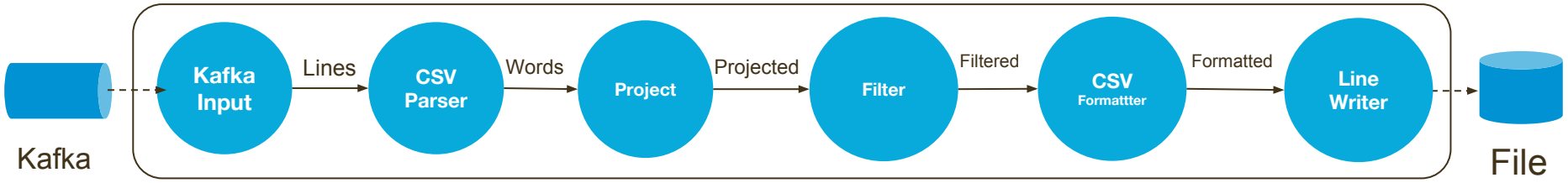

```
@ApplicationAnnotation(name="MeetupApplication")
public class Application implements StreamingApplication
{
  @Override
  public void populateDAG(DAG dag, Configuration conf)
  {
    KafkaSinglePortStringInputOperator kafkaInput
            = dag.addOperator("Input", KafkaSinglePortStringInputOperator.class);
    Parser parser = dag.addOperator("Parser", Parser.class);
    Filter filter = dag.addOperator("Filter", Filter.class);
    Counter<String> counter = dag.addOperator("Counter", Counter.class);
    JdbcOutput output = dag.addOperator("Output", JdbcOutput.class);

    dag.addStream("lines", kafkaInput.outputPort, parser.input);
    dag.addStream("words", parser.output, filter.input);
    dag.addStream("filtered", filter.output, counter.input);
    dag.addStream("counts", counter.count, output.input);
  }
}
```

Apex APIs: Declarative (High Level)




```
StreamFactory.fromFolder("/tmp")
    .flatMap(input -> Arrays.asList(input.split(" ")), name("Words"))
    .window(new WindowOption.GlobalWindow(),
            new TriggerOption().accumulatingFiredPanes().withEarlyFiringsAtEvery1))
    .countByKey(input -> new Tuple.PlainTuple<>(new KeyValPair<>(input, 1L)), name("countByKey"))
    .map(input -> input.getValue(), name("Counts"))
    .print(name("Console"))
    .populateDag(dag);
```

Apex APIs: SQL




```
SQLExecEnvironment.getEnvironment()
    .registerTable("ORDERS",
        new KafkaEndpoint(conf.get("broker"), conf.get("topic"),
            new CSVMessageFormat(conf.get("schemaInDef"))))
    .registerTable("SALES",
        new FileEndpoint(conf.get("destFolder"), conf.get("destFileName"),
            new CSVMessageFormat(conf.get("schemaOutDef"))))
    .registerFunction("APEXCONCAT", this.getClass(), "apex_concat_str")
    .executeSQL(dag,
        "INSERT INTO SALES " +
            "SELECT STREAM ROWTIME, FLOOR(ROWTIME TO DAY), APEXCONCAT('OILPAINT', SUBSTRING(PRODUCT, 6, 7) " +
            "FROM ORDERS WHERE ID > 3 AND PRODUCT LIKE 'paint%'");
```


DataTorrent

Apex APIs: Beam

- Apex Runner of Beam is available!!
- Build once run-anywhere model
- Beam Streaming applications can be run on apex runner:

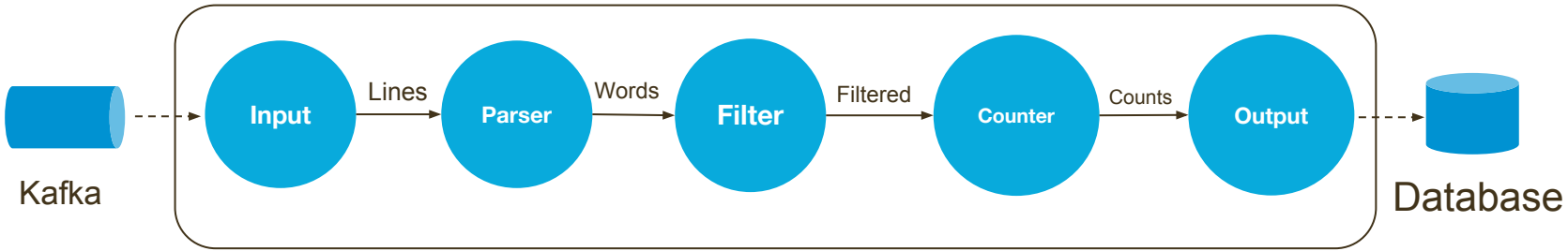
```
public static void main(String[] args) {
 Options options = PipelineOptionsFactory.fromArgs(args).withValidation().as(Options.class);

 // Run with Apex runner
 options.setRunner(ApexRunner.class);


 Pipeline p = Pipeline.create(options);
 p.apply("ReadLines", TextIO.Read.from(options.getInput()))
     .apply(new CountWords())
     .apply(MapElements.via(new FormatAsTextFr()))
     .apply("WriteCounts", TextIO.Write.to(options.getOutput()));
 .run().waitUntilFinish();
}
```

Apex APIs: SAMOA

- Build once run-anywhere model for online machine learning algorithms
- Any machine learning algorithm present in SAMOA can be run directly on Apex.
- Uses Apex Iteration Support
- Following example does classification of input data from HDFS using VHT algorithm on Apex:

```
$   bin/samoa apex ../SAMOA-Apex-0.4.0-incubating-SNAPSHOT.jar "PrequentialEvaluation
    -d /tmp/dump.csv
    -l (classifiers.trees.VerticalHoeffdingTree -p 1)
    -s (org.apache.samoa.streams.ArffFileStream
          -s HDFSFileStreamSource
          -f /tmp/user/input/covtypeNorm.arff)"
```

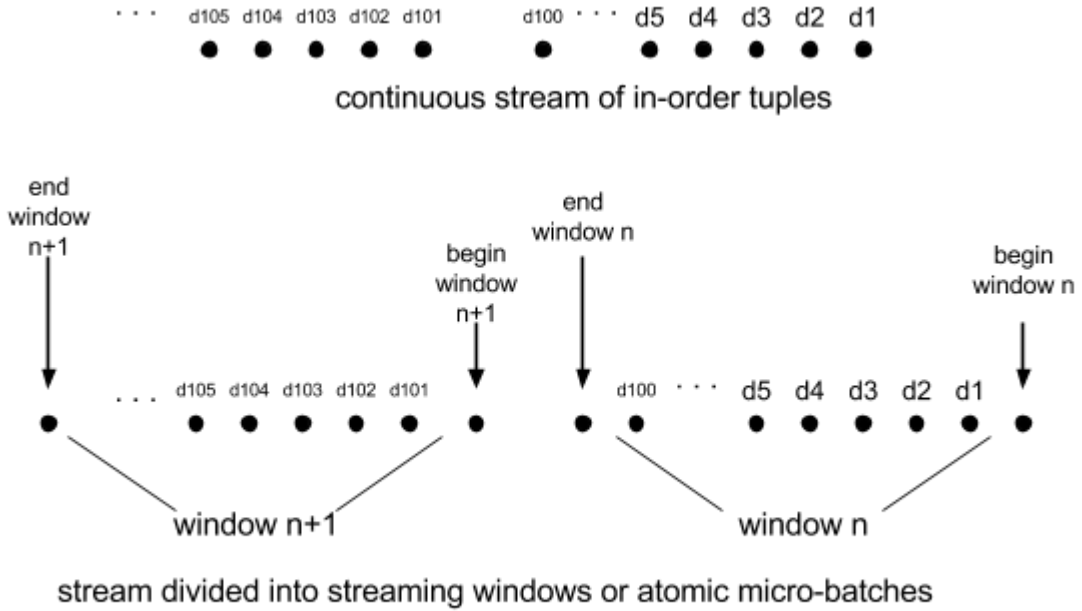
Configuration (properties.xml)




```
<property>
  <name>dt.operator.Input.prop.topic</name>
  <value>meetup</value>
</property>
<property>
  <name>dt.operator.Input.prop.zookeeper</name>
  <value>node28.morado.com:2181</value>
</property>

<property>
  <name>dt.operator.Output.prop.store.databaseDriver</name>
  <value>com.mysql.jdbc.Driver</value>
</property>
<property>
  <name>dt.operator.Output.prop.store.databaseUrl</name>
  <value>jdbc:mysql://node28.morado.com/meetup?user=meetup&amp;password=meetup</value>
</property>
```

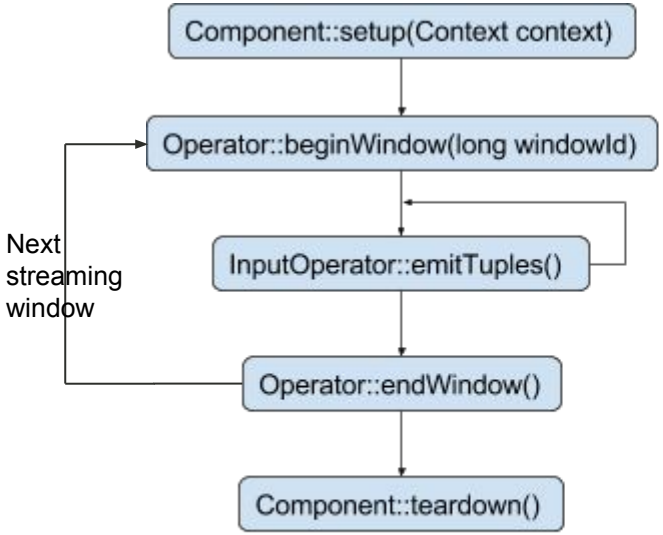
Streaming Window

Processing Time Window

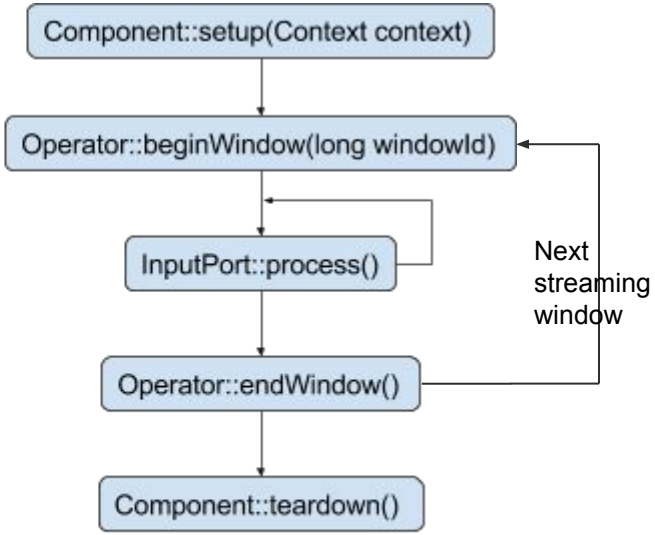


- Finite time sliced windows based on processing (event arrival) time
- Used for bookkeeping of streaming application
- Derived Windows are: *Checkpoint Windows*, *Committed Windows*

Operator APIs



Flow for Input Adapters



Flow for Generic Operators and Output Adapters

Input Adapters - Starting of the pipeline. Interacts with external system to generate stream
Generic Operators - Processing part of pipeline
Output Adapters - Last operator in pipeline. Interacts with external system to finalize the processed stream

Overview of Operator Library (Malhar)

APEX

Messaging
- Kafka
- JMS (ActiveMQ etc.)
- Kinesis, SQS
- Flume, NiFi

NoSQL
- Cassandra, HBase
- Aerospike, Accumulo
- Couchbase/ CouchDB
- Redis, MongoDB
- Geode

RDBMS
- JDBC
- MySQL
- Oracle
- MemSQL

File Systems
- HDFS/ Hive
- Local File
- S3

Parsers
- XML
- JSON
- CSV
- Avro
- Parquet

Transformations
- Filters, Expression, Enrich
- Windowing, Aggregation
- Join
- Dedup

Analytics
- Dimensional Aggregations (with state management for historical data + query)

Protocols
- HTTP
- FTP
- WebSocket
- MQTT
- SMTP

Other
- Elastic Search
- Script (JavaScript, Python, R)
- Solr
- Twitter

Frequently used Connectors

Kafka Input

	KafkaSinglePortInputOperator	KafkaSinglePortByteArrayInputOperator
Library	malhar-contrib	malhar-kafka
Kafka Consumer	0.8	0.9
Emit Type	byte[]	byte[]
Fault-Tolerance	At Least Once, Exactly Once	At Least Once, Exactly Once
Scalability	Static and Dynamic (with Kafka metadata)	Static and Dynamic (with Kafka metadata)
Multi-Cluster/Topic	Yes	Yes
Idempotent	Yes	Yes
Partition Strategy	1:1, 1:M	1:1, 1:M

Frequently used Connectors

Kafka Output

	KafkaSinglePortOutputOperator	KafkaSinglePortExactlyOnceOutputOperator
Library	malhar-contrib	malhar-kafka
Kafka Producer	0.8	0.9
Fault-Tolerance	At Least Once	At Least Once, Exactly Once
Scalability	Static and Dynamic (with Kafka metadata)	Static and Dynamic, Automatic Partitioning based on Kafka metadata
Multi-Cluster/Topic	Yes	Yes
Idempotent	Yes	Yes
Partition Strategy	1:1, 1:M	1:1, 1:M

Frequently used Connectors

File Input

- AbstractFileInputOperator
  - Used to read a file from source and emit the content of the file to downstream operator
- Operator is idempotent
- Supports Partitioning
- Few Concrete Impl
  - FileLineInputOperator
  - AvroFileInputOperator
  - ParquetFilePOJOReader
- https://www.datatorrent.com/blog/fault-tolerant-file-processing/

```
public class FileLineInputOperator extends AbstractFileInputOperator<String> {
  public final transient DefaultOutputPort<String> output = new DefaultOutputPort<>();
  protected transient BufferedReader br;

  @Override
  protected InputStream openFile(Path path) throws IOException {
    InputStream is = super.openFile(path);
    br = new BufferedReader(new InputStreamReader(is));
    return is;
  }

  @Override
  protected void closeFile(InputStream is) throws IOException {
    super.closeFile(is);
    br.close();
    br = null;
  }

  @Override
  protected String readEntity() throws IOException {
    return br.readLine();
  }

  @Override
  protected void emit(String tuple) {
    output.emit(tuple);
  }
}
```

Frequently used Connectors

File Output

- AbstractFileOutputOperator
  - Writes data to a file
- Supports Partitions
- Exactly-once results
  - Upstream operators should be idempotent
- Few Concrete Impl
  - StringFileOutputOperator
- https://www.datatorrent.com/blog/fault-tolerant-file-processing/

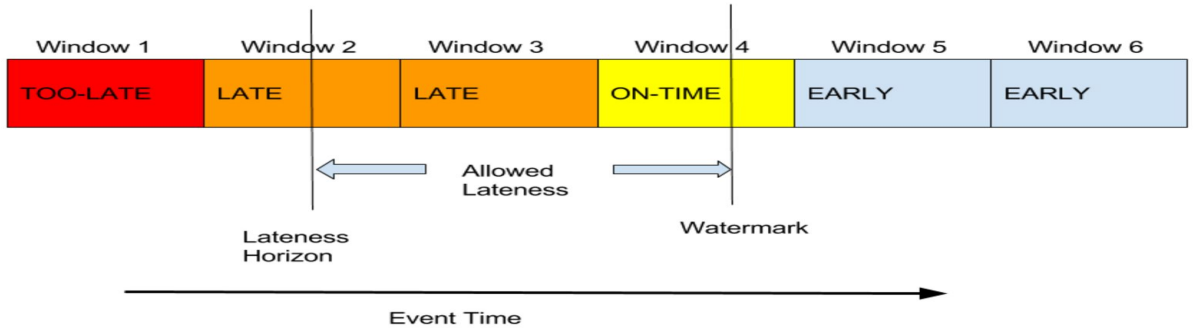
```
public class LineWriter extends AbstractFileOutputOperator<String> {
  private String fileName;

  @Override
  protected String getFileName(String tuple) {
    return fileName;
  }

  @Override
  protected byte[] getBytesForTuple(String tuple) {
    return tuple.getBytes();
  }
}
```

Windowing Support

- *Event-time Windows*
- Computation based on event-time present in the tuple
- Types of event-time windows supported:
  - *Global* : Single event-time window throughout the lifecycle of application
  - *Timed* : Tuple is assigned to single, non-overlapping, fixed width windows immediately followed by next window
  - *Sliding Time* : Tuple is can be assigned to multiple, overlapping fixed width windows.
  - *Session* : Tuple is assigned to single, variable width windows with a predefined min gap

Stateful Windowed Processing

- *WindowedOperator* from *malhar-library*
- Used to process data based on Event time as contrary to ingression time
- Supports windowing semantics of Apache Beam model
- Supported features:
  - Watermarks
  - Allowed Lateness
  - Accumulation
  - Accumulation Modes: Accumulating, Discarding, Accumulating & Retracting
  - Triggers
- Storage
  - In memory based
  - Managed State based



Stateful Windowed Processing

Compositional API

APEX

```
@Override
public void populateDAG(DAG dag, Configuration configuration)
{
 WordGenerator inputOperator =  new WordGenerator();

 KeyedWindowedOperatorImpl windowedOperator = new KeyedWindowedOperatorImpl();
 Accumulation<Long, MutableLong, Long> sum = new SumAccumulation();

 windowedOperator.setAccumulation(sum);
 windowedOperator.setDataStorage(new InMemoryWindowedKeyedStorage<String, MutableLong>());
 windowedOperator.setRetractionStorage(new InMemoryWindowedKeyedStorage<String, Long>());
 windowedOperator.setWindowStateStorage(new InMemoryWindowedStorage<WindowState>());
 windowedOperator.setWindowOption(new WindowOption.TimeWindows(Duration.standardMinutes(1)));
 windowedOperator.setTriggerOption(TriggerOption.AtWatermark()
                                                 .withEarlyFiringsAtEvery(Duration.millis(1000))
                                                 .accumulatingAndRetractingFiredPanes());
 windowedOperator.setAllowedLateness(Duration.millis(14000));

 ConsoleOutputOperator outputOperator =  new ConsoleOutputOperator();
 dag.addOperator("inputOperator", inputOperator);
 dag.addOperator("windowedOperator", windowedOperator);
 dag.addOperator("outputOperator", outputOperator);
 dag.addStream("input_windowed", inputOperator.output, windowedOperator.input);
 dag.addStream("windowed_output", windowedOperator.output, outputOperator.input);
}
```


DATATORRENT

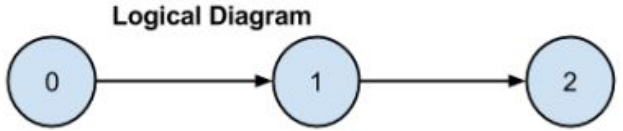
Stateful Windowed Processing

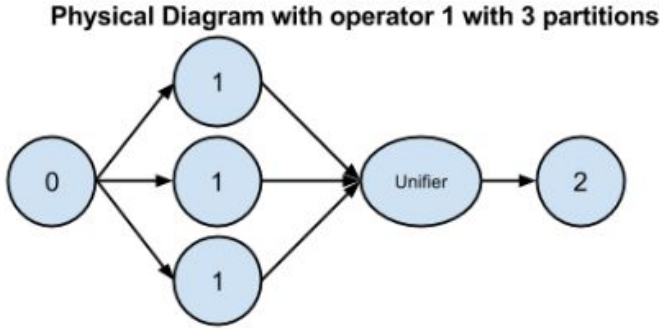
Declarative API


```
StreamFactory.fromFolder("/tmp")
   .flatMap(input -> Arrays.asList(input.split( " ")), name("ExtractWords"))
   .map(input -> new TimestampedTuple<>(System.currentTimeMillis(), input),   name("AddTimestampFn"))

   .window(new TimeWindows(Duration.standardMinutes(WINDOW_SIZE)),
           new TriggerOption().accumulatingFiredPanes().withEarlyFiringsAtEvery(1))

   .countByKey(input -> new TimestampedTuple<>(input.getTimestamp(),  new KeyValPair<>(input.getValue(),
                1L ))), name("countWords"))
   .map(new FormatAsTableRowFn(),  name("FormatAsTableRowFn"))
   .print(name("console"))
   .populateDag(dag);
```

Scalability - Partitioning

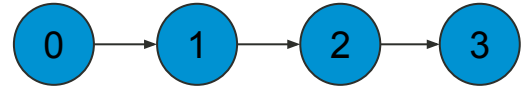
- Useful for low latency and high throughput
- Replicates (Partitions) the logic
- Configured at launch time (Application.java or properties.xml)
- *StreamCodec*
  - Used for controlling distribution of tuples to downstream partitions
- *Unifier* (combine results of partitions)
  - Passthrough unifier added by platform to merge results from upstream partitions
  - Can also be customized
- Type of partitions
  - *Static partitions* - Statically partition at launch time
  - *Dynamic partitions* - Partitions changing at runtime based on latency and/or throughput
  - *Parallel partitions* - Upstream and downstream operators using same partition scheme



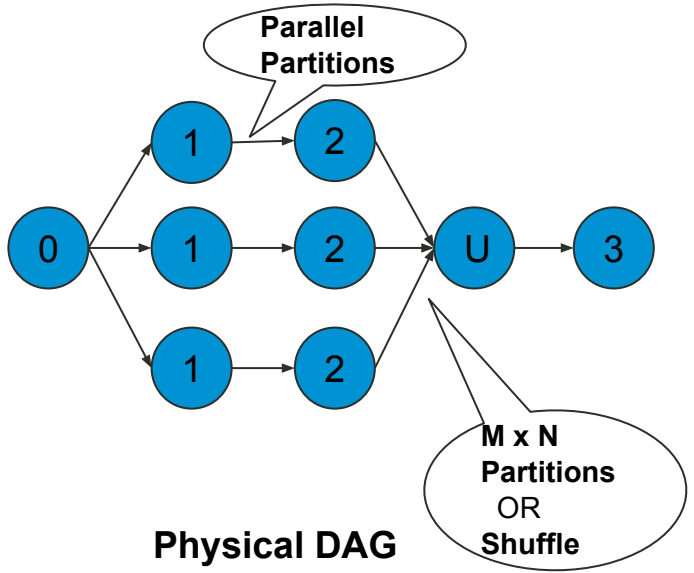


Scalability - Partitioning (contd.)

```
<configuration>

 <property>
   <name>dt.operator.1. attr.PARTITIONER</name>
   <value>com.datatorrent.common.partitioner. StatelessPartitioner:3</value>
 </property>

 <property>
   <name>dt.operator.2.port.inputPortName. attr.PARTITION_PARALLEL</name>
   <value>true</value>
 </property>

</configuration>
```

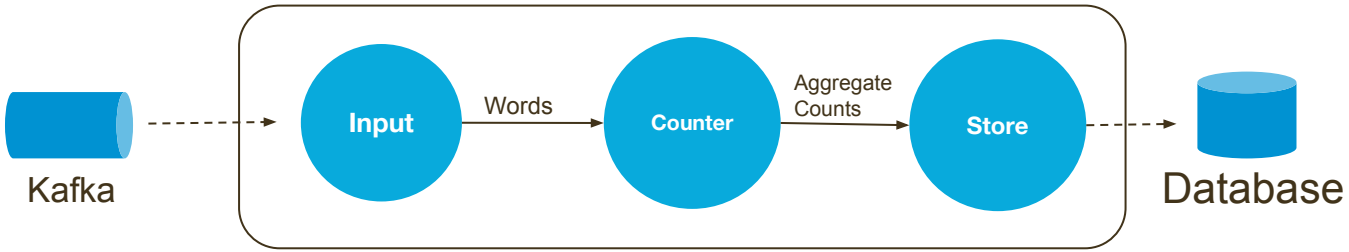


Logical DAG



Physical DAG

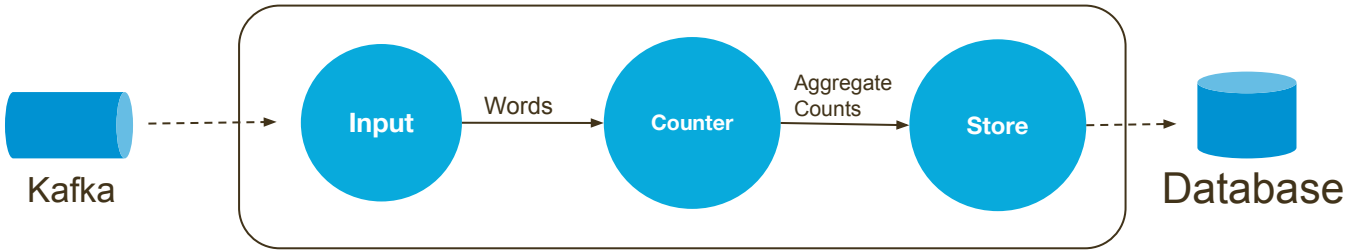
End-to-End Exactly-Once



- **Input**
  - Uses *com.datatorrent.contrib.kafka.KafkaSinglePortStringInputOperator*
  - Emits words as a stream
  - Operator is idempotent
- **Counter**
  - *com.datatorrent.lib.algo.UniqueCounter*
- **Store**
  - Uses *CountStoreOperator*
  - Inserts into JDBC
  - Exactly-once results (End-To-End Exactly-once = At-least-once + Idempotency + Consistent State)

https://github.com/DataTorrent/examples/blob/master/tutorials/exactly-once
https://www.datatorrent.com/blog/end-to-end-exactly-once-with-apache-apex/

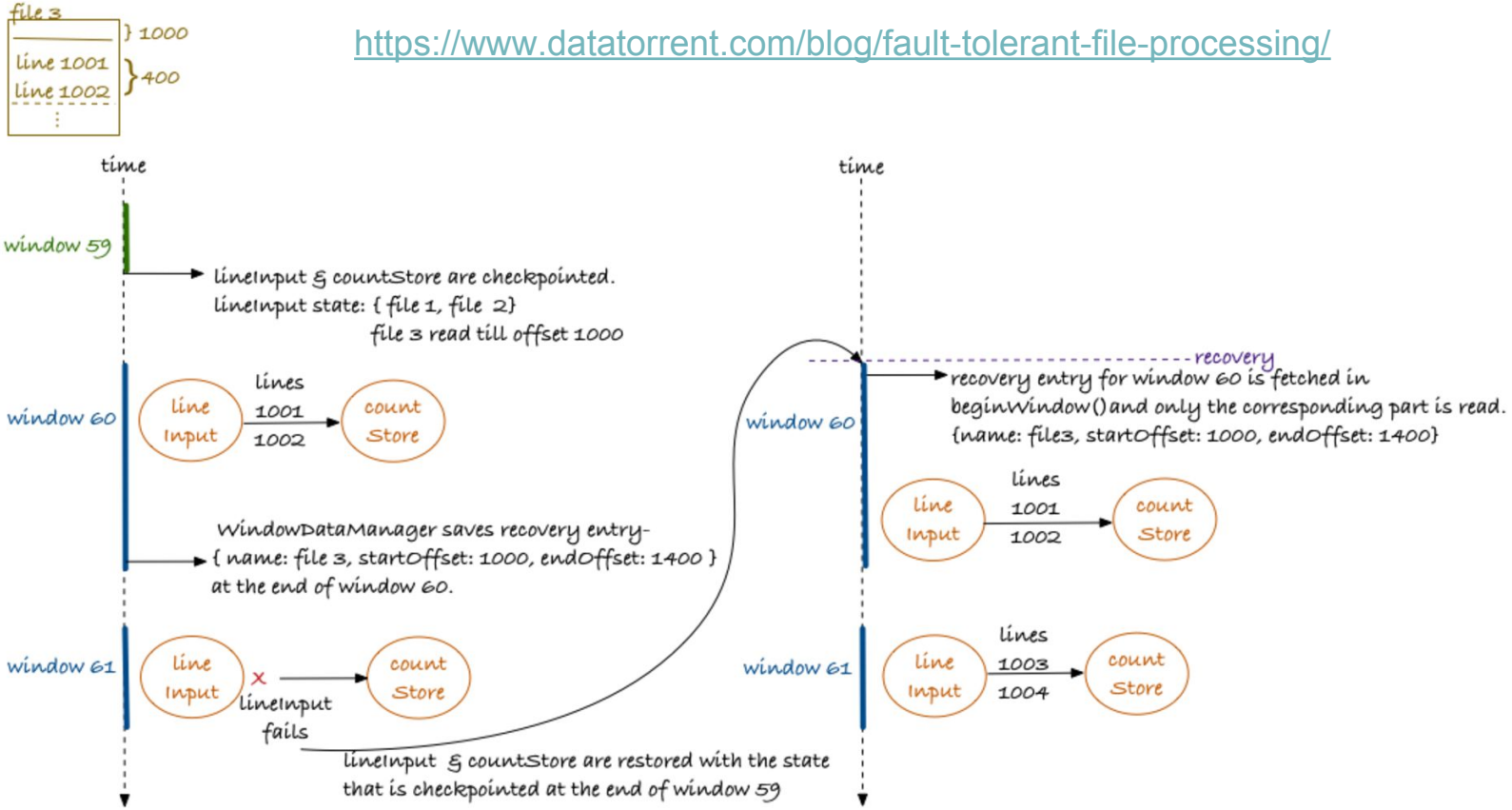
End-to-End Exactly-Once (Contd.)




```
public static class CountStoreOperator extends AbstractJdbcTransactionableOutputOperator<KeyValPair<String, Integer>>
{
 public static final String SQL =
     "MERGE INTO words USING (VALUES ?, ?) I (word, wcount)"
     + " ON (words.word=I.word)"
     + " WHEN MATCHED THEN UPDATE SET words.wcount = words.wcount + I.wcount"
     + " WHEN NOT MATCHED THEN INSERT (word, wcount) VALUES (I.word, I.wcount)";

 @Override
 protected String getUpdateCommand()
 {
   return SQL;
 }

 @Override
 protected void setStatementParameters(PreparedStatement statement, KeyValPair<String, Integer> tuple) throws SQLException
 {
   statement.setString(1, tuple.getKey());
   statement.setInt(2, tuple.getValue());
 }
}
```

End-to-End Exactly-Once (Contd.)

https://www.datatorrent.com/blog/fault-tolerant-file-processing/



Who is using Apex?

- Powered by Apex
  - http://apex.apache.org/powered-by-apex.html
  - Also using Apex? Let us know to be added: users@apex.apache.org or @ApacheApex
- Pubmatic
  - https://www.youtube.com/watch?v=JSXpgfQFcU8
- GE
  - https://www.youtube.com/watch?v=hmaSkXhHNu0
  - http://www.slideshare.net/ApacheApex/ge-iot-predix-time-series-data-ingestion-service-using-apache-apex-hadoop
- SilverSpring Networks
  - https://www.youtube.com/watch?v=8VORISKeSjI
  - http://www.slideshare.net/ApacheApex/iot-big-data-ingestion-and-processing-in-hadoop-by-silver-spring-networks

Resources

- http://apex.apache.org/
- Learn more - http://apex.apache.org/docs.html
- Subscribe - http://apex.apache.org/community.html
- Download - http://apex.apache.org/downloads.html
- Follow @ApacheApex - https://twitter.com/apacheapex
- Meetups - https://www.meetup.com/topics/apache-apex/
- Examples - https://github.com/DataTorrent/examples
- Slideshare - http://www.slideshare.net/ApacheApex/presentations
- https://www.youtube.com/results?search_query=apache+apex
- Free Enterprise License for Startups - https://www.datatorrent.com/product/startup-accelerator/

Q&A