Comparison of streaming processing framework
Distributed processing approach

• Batch Processing
  – has access to all data
  – might compute something big and complex
  – more concerned with throughput than latency
  – higher latencies

• Stream Processing
  – a one-at-a-time processing model
  – data are processed immediately upon arrival
  – computations are relatively simple and generally independent
  – sub-second latency
Stream processing approach

• Native stream processing

Native stream processing systems
continuous operator model

records processed one at a time
Stream processing approach

- Micro-batching processing

Records processed in short batches
Stream processing framework

- Storm
- Storm with Trident
- Spark
- Samza
- Flink
# Comparing table

<table>
<thead>
<tr>
<th></th>
<th>Storm</th>
<th>Trident</th>
<th>Spark</th>
<th>Samza</th>
<th>Flink</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Streaming Model</strong></td>
<td>Native</td>
<td>Micro-batching</td>
<td>Micro-batching</td>
<td>Native</td>
<td>Native</td>
</tr>
<tr>
<td><strong>API</strong></td>
<td>Compositional</td>
<td>Declarative</td>
<td>Compositional</td>
<td>Declarative</td>
<td>Declarative</td>
</tr>
<tr>
<td><strong>Guarantees</strong></td>
<td>At-least-once</td>
<td>Exactly-once</td>
<td>Exactly-once</td>
<td>At-least-once</td>
<td>Exactly-once</td>
</tr>
<tr>
<td><strong>State Management</strong></td>
<td>Not Built-in</td>
<td>Dedicated DStream</td>
<td>Dedicated DStream</td>
<td>Stateful Operation</td>
<td>Stateful Operation</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>Very Low</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>
API

• Compositional
  – Provide basic building blocks like source or operator
  – Must be tied together to create expected topology

• Declarative
  – System creates and optimize topology itself
API

• Compositional

```java
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("spout", new RandomSentenceSpout(), 5);
builder.setBolt("split", new Split(), 8).shuffleGrouping("spout");
```

• Declarative

```scala
val conf = new SparkConf().setAppName("wordcount")
val ssc = new StreamingContext(conf, Seconds(1))

val text = ...

val counts = text.flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)

counts.print()

ssc.start()
ssc.awaitTermination()
```
Guarantee

• Message Delivery Guarantees
  – At most once: data may be lost
  – At least once: data may be duplicated
  – Exactly once: data neither lost nor duplicated
## Comparing table

<table>
<thead>
<tr>
<th></th>
<th>Storm</th>
<th>Trident</th>
<th>Spark</th>
<th>Samza</th>
<th>Flink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streaming Model</td>
<td>Native</td>
<td>Micro-batching</td>
<td>Micro-batching</td>
<td>Native</td>
<td>Native</td>
</tr>
<tr>
<td>API</td>
<td>Compositional</td>
<td>Declarative</td>
<td>Compositional</td>
<td>Declarative</td>
<td>Declarative</td>
</tr>
<tr>
<td>Guarantees</td>
<td>At-least-once</td>
<td>Exactly-once</td>
<td>Exactly-once</td>
<td>At-least-once</td>
<td>Exactly-once</td>
</tr>
<tr>
<td>State Management</td>
<td>Not Built-in</td>
<td>Dedicated Operation</td>
<td>Dedicated DStream</td>
<td>Stateful Operation</td>
<td>Stateful Operation</td>
</tr>
<tr>
<td>Latency</td>
<td>Very Low</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Throughput</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>