Flink

Agenda

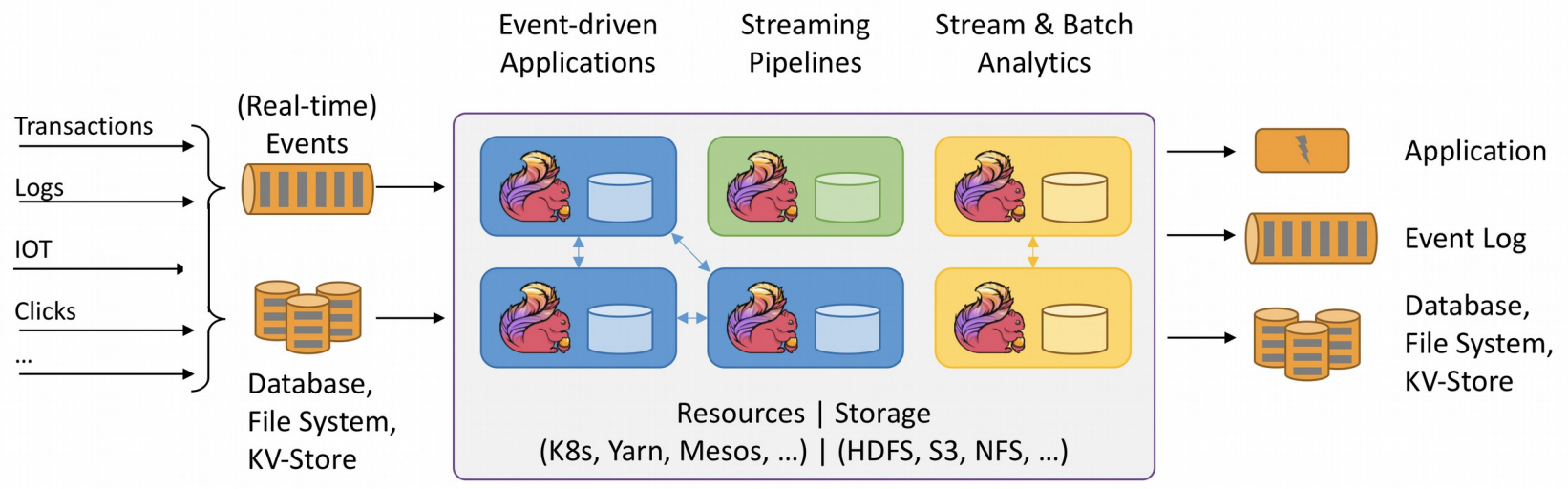
- **What is Flink?**
  - Architecture
  - Applications
  - Operations
- **Use Cases**
  - Event-Driven Applications
  - Data Analytics
  - Data Pipeline
- **Streaming API**
  - Examples
- **DataSet API**
  - Examples
- **Table API**
  - Examples
- **SQL**
  - Examples

What is Flink?

Apache Flink is:

- A framework
- A distributed processing engine
  - Unbounded data streams
  - Bounded data streams
  - All common cluster environments
  - In-memory speed computations
  - Very scalable

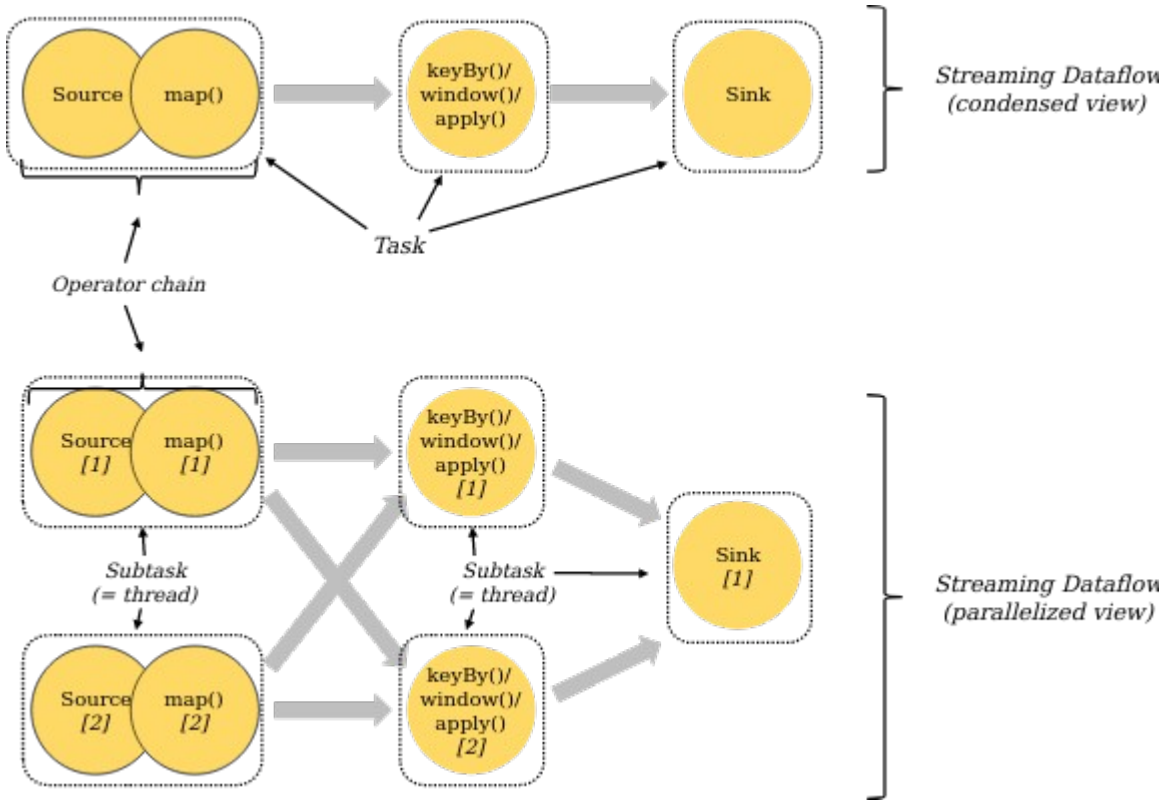
What is Flink?



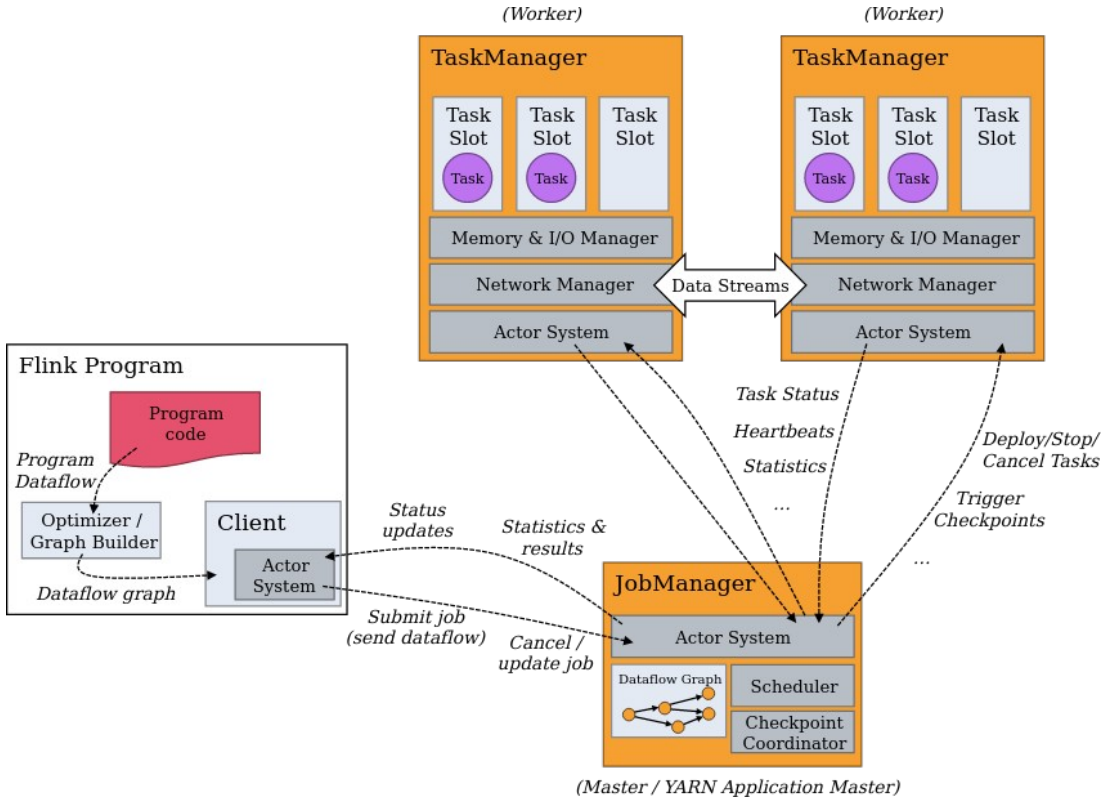
Flink Architecture

- Tasks and Operator Chains
- Job Managers, Task Managers, Clients
- Task Slots and Resources
- State Backends

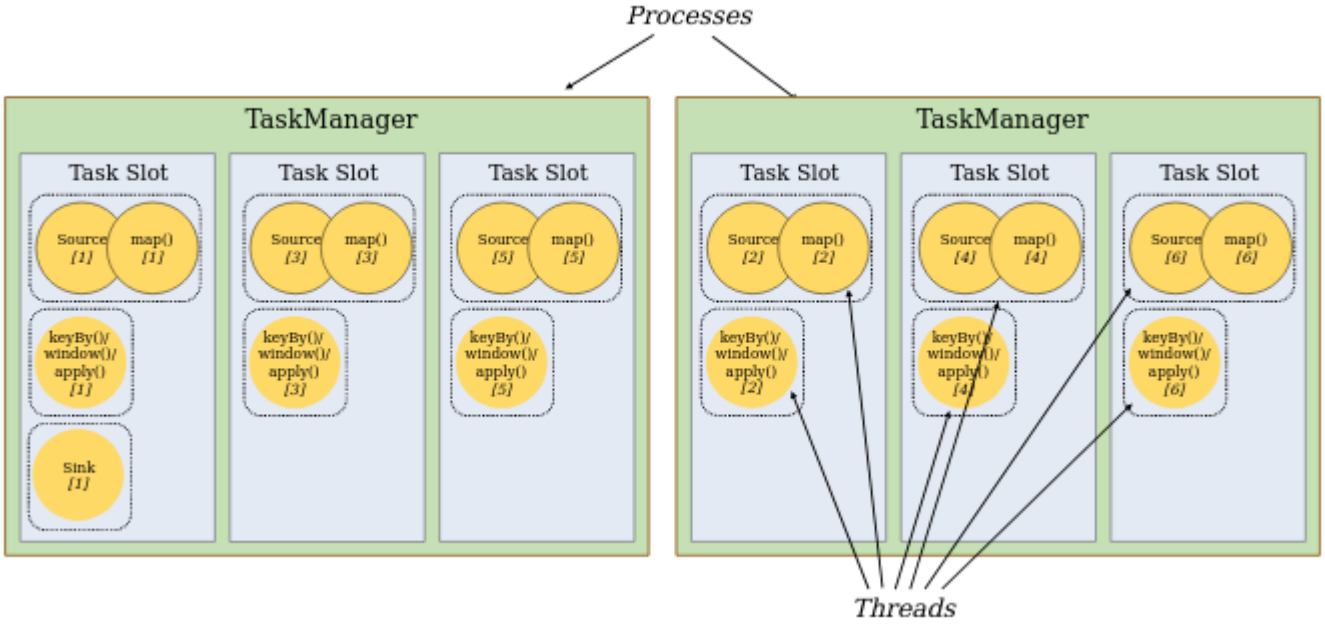
Tasks and Operator Chains



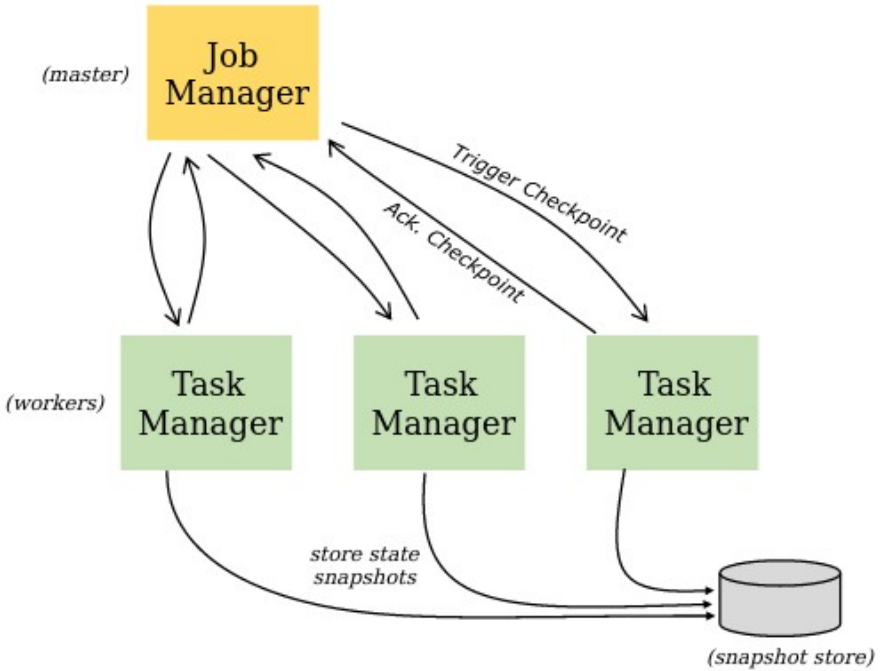
Job Managers, Task Managers, Clients



Task Slots and Resources



State Backends



Flink Applications

Types of applications that can be built with Flink (or any other stream processing framework) are defined by stream, state and time control.

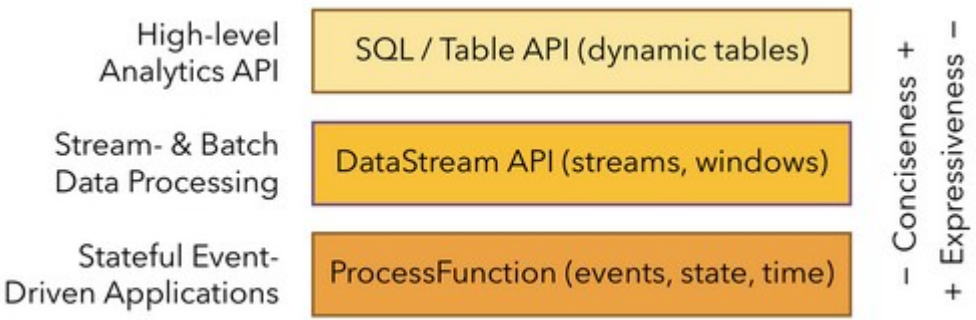
Stream Types

- **Bounded**
  - Fixed-sized data sets
- **Unbounded**
  - Unlimited/indeterminate sized data sets
- **Real-time**
  - Process as it is generated
- **Recorded**
  - Persist the stream to a storage system, process later

State Management

- **Multiple State Primitives**
  - Atomic values, lists, maps
- **Pluggable State Backends**
  - State managed in and checkpointed by a pluggable state backend
  - In-memory, file-system, RocksDB, Custom
- **Exactly-once State Consistency**
  - Guaranteed consistency of application state in case of a failure
- **Very Large State**
  - Flink can handle terabytes of state
- **Scalable Applications**
  - State can be redistributed to more or fewer workers

Layered APIs



ProcessFunction

```
public static class StartEndDuration
    extends KeyedProcessFunction<String, Tuple2<String, String>, Tuple2<String, Long>> {

  private ValueState<Long> startTime;

  @Override
  public void open(Configuration conf) {
    // obtain state handle
    startTime = getRuntimeContext()
      .getState(new ValueStateDescriptor<Long>("startTime", Long.class));
  }
```

```
  /** Called when a timer fires. */
  @Override
  public void onTimer(
      long timestamp,
      OnTimerContext ctx,
      Collector<Tuple2<String, Long>> out) {

    // Timeout interval exceeded. Cleaning up the state.
    startTime.clear();
  }
```

```
  /** Called for each processed event. */
  @Override
  public void processElement(
      Tuple2<String, String> in,
      Context ctx,
      Collector<Tuple2<String, Long>> out) throws Exception {

    switch (in.f1) {
      case "START":
        // set the start time if we receive a start event.
        startTime.update(ctx.timestamp());
        // register a timer in four hours from the start event.
        ctx.timerService()
          .registerEventTimeTimer(ctx.timestamp() + 4 * 60 * 60 * 1000);
        break;
      case "END":
        // emit the duration between start and end event
        Long sTime = startTime.value();
        if (sTime != null) {
          out.collect(Tuple2.of(in.f0, ctx.timestamp() - sTime));
          // clear the state
          startTime.clear();
        }
      default:
        // do nothing
    }
  }
```

DataStream API

```
// a stream of website clicks
DataStream<Click> clicks = ...

DataStream<Tuple2<String, Long>> result = clicks
  // project clicks to userId and add a 1 for counting
  .map(
    // define function by implementing the MapFunction interface.
    new MapFunction<Click, Tuple2<String, Long>>() {
      @Override
      public Tuple2<String, Long> map(Click click) {
        return Tuple2.of(click.userId, 1L);
      }
    })
  // key by userId (field 0)
  .keyBy(0)
  // define session window with 30 minute gap
  .window(EventTimeSessionWindows.withGap(Time.minutes(30L)))
  // count clicks per session. Define function as lambda function.
  .reduce((a, b) -> Tuple2.of(a.f0, a.f1 + b.f1));
```

SQL & Table API

```
// get a TableEnvironment
TableEnvironment tableEnv = ...; // see "Create a

// register Orders table

// scan registered Orders table
Table orders = tableEnv.scan("Orders");
// compute revenue for all customers from France
Table revenue = orders
  .filter("cCountry === 'FRANCE'")
  .groupBy("cID, cName")
  .select("cID, cName, revenue.sum AS revSum");

// emit or convert Table
// execute query
```

```
// get a TableEnvironment
TableEnvironment tableEnv = ...; // see "Create a Ta

// register Orders table

// compute revenue for all customers from France
Table revenue = tableEnv.sqlQuery(
    "SELECT cID, cName, SUM(revenue) AS revSum " +
    "FROM Orders " +
    "WHERE cCountry = 'FRANCE' " +
    "GROUP BY cID, cName"
  );

// emit or convert Table
// execute query
```

Libraries

- **Complex Event Processing (CEP)**
  - Pattern detection
    - Network intrusion detection,
    - Business process monitoring,
    - Fraud detection
- **DataSet API**
  - Batch processing
- **Gelly**
  - Graph processing & analysis
    - Label propagation, triangle enumeration, page rank

Flink Operations

- **Consistent Checkpoints**
- **Efficient Checkpoints**
  - Async & Incremental checkpoints
- **End-to-End Exactly-Once**
- **Integration with Cluster Managers**
  - Hadoop YARN, Mesos, Kubernetes
- **High-Availability Setup**
  - HA-mode based on Apache ZooKeeper

Flink Operations

- **Application Evolution**
  - Also possible to start application from an earlier point in time
- **Cluster Migration**
  - Using savepoints, applications can be migrated (or cloned) to different clusters
- **Flink Version Updates**
  - Application can be migrated using a savepoint
- **Application Scaling**
- **A/B Tests and What-If Scenarios**
- **Pause/Resume**
- **Archiving**

Flink Operations

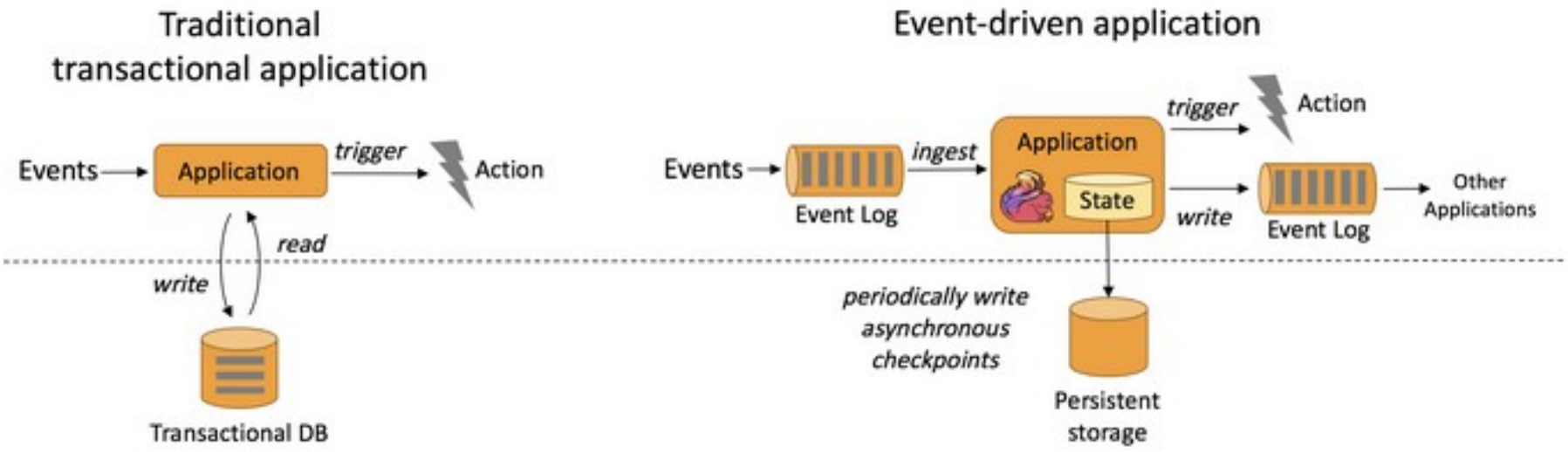
- **Web UI**
- **Logging**
- **Metrics**
  - JMX, Ganglia, Graphite, Prometheus, StatsD, Datadog
- **REST API**
  - Submit new application
  - Take savepoint of a running application
  - Cancel application
  - Meta data
  - Collected metrics

Flink Use Cases

Most common ones are

- **Event-driven Applications**
- **Data Analytics Applications**
- **Data Pipeline Applications**

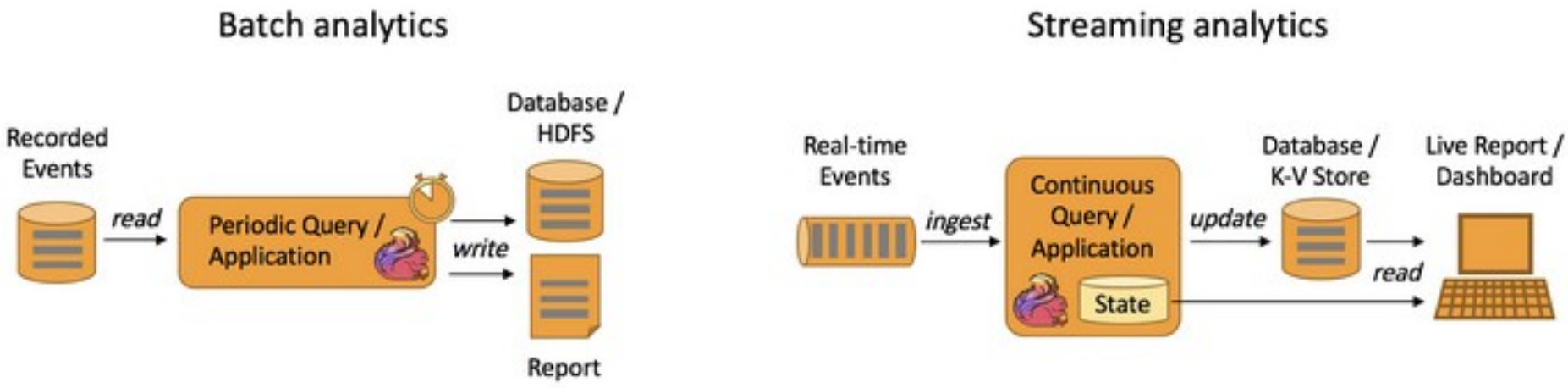
Event-driven Applications



Typical Event-driven Applications

- **Fraud detection**
- **Anomaly Detection**
- **Rule-based alerting**
- **Business process monitoring**
- **Web applications (e.g. social networks)**

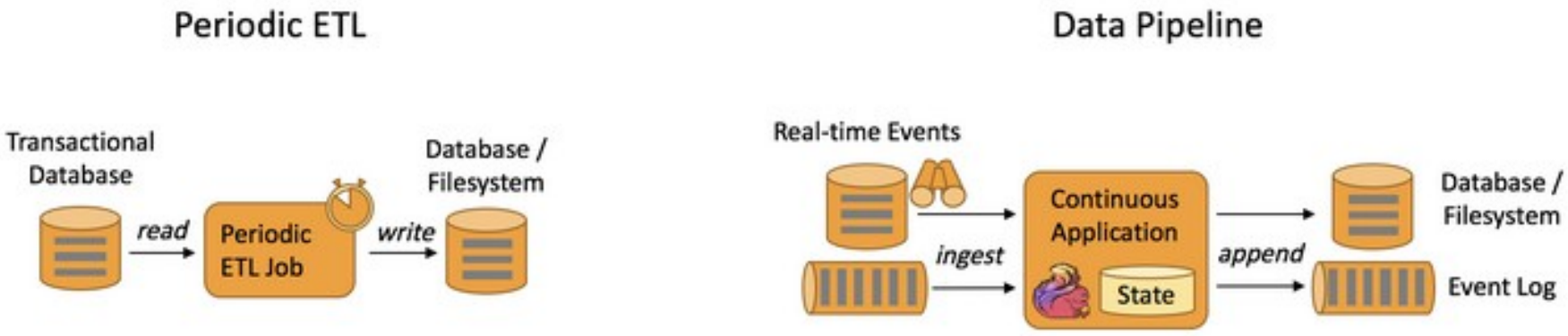
Data Analytics Applications



Typical Data Analytics Applications

- **Network quality monitoring**
- **Ad-hoc analysis of live data**
- **Large-scale graph analysis**

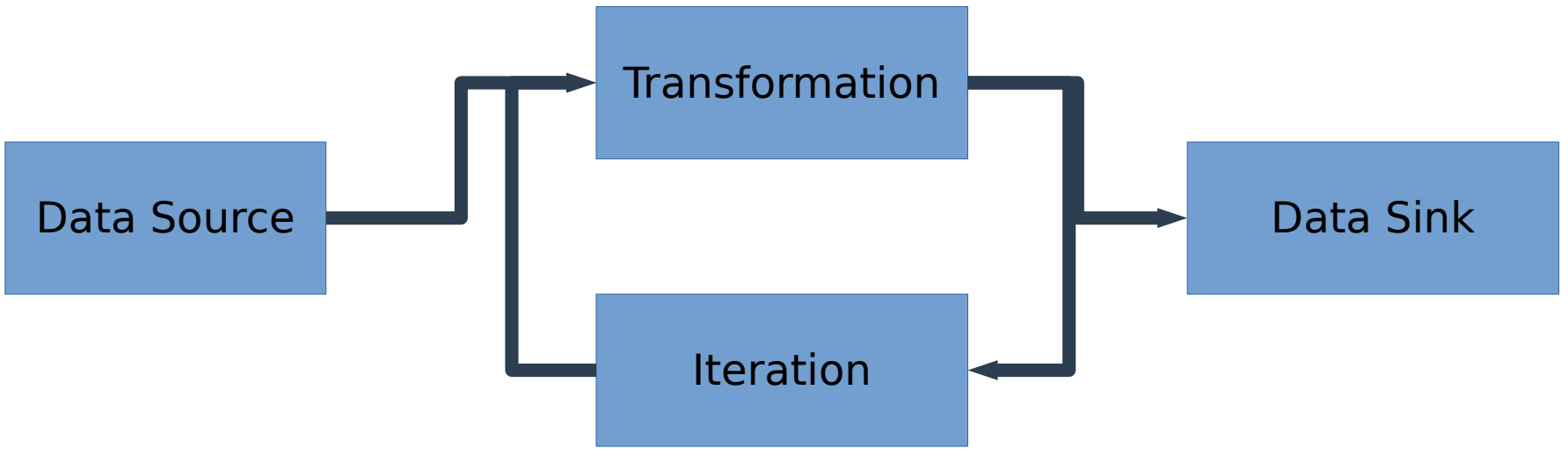
Data Pipeline Applications



- Real-time search index building
- Continuous ETL

ETL: Extract-Transform-Load

Streaming API



Example Streaming API Application

```
public class WindowWordCount {

    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        DataStream<Tuple2<String, Integer>> dataStream = env
                .socketTextStream("localhost", 9999)
                .flatMap(new Splitter())
                .keyBy(0)
                .timeWindow(Time.seconds(5))
                .sum(1);

        dataStream.print();

        env.execute("Window WordCount");
    }

    public static class Splitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
        @Override
        public void flatMap(String sentence, Collector<Tuple2<String, Integer>> out) throws Exception {
            for (String word: sentence.split(" ")) {
                out.collect(new Tuple2<String, Integer>(word, 1));
            }
        }
    }

}
```

Sources, Operators and Sinks

- **Predefined Sources:**
  - ReadTextFile
  - ReadFile
  - SocketTextStream
  - FromCollection
  - FromElements
  - FromParallelCollection
  - GenerateSequence
- **Predefined Sinks:**
  - WriteAsText
  - WriteAsCsv
  - Print / printToErr
  - WriteUsingOutputFormat
  - writeToSocket

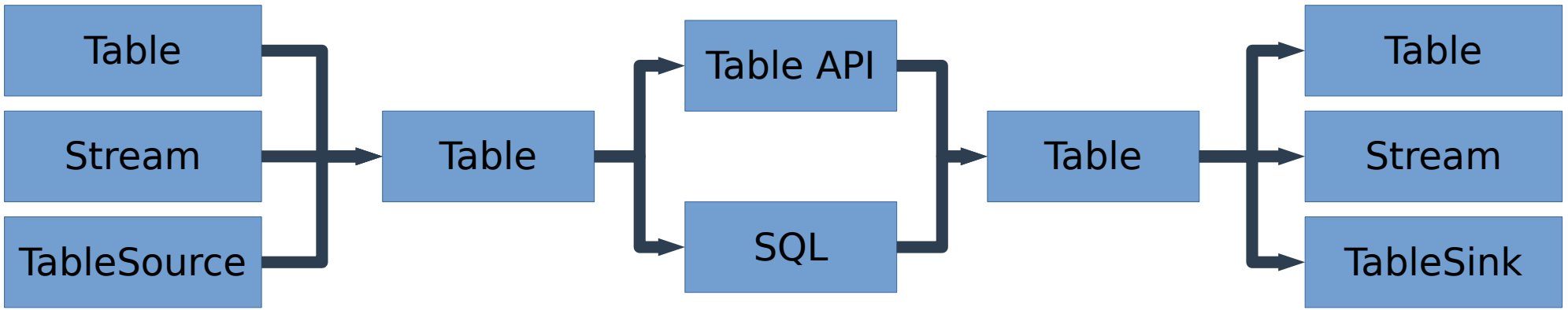
Sources, Operators and Sinks

- **Operators:**
  - Map / FlatMap
  - Filter
  - KeyBy
  - Reduce
  - Fold
  - sum/min/max/minBy/maxBy
  - Windowed versions of above operators
  - Union
  - Connect (to create ConnectedStreams)
  - Split/select
  - Iterate
  - Timestamp extraction
  - Projection
  - Partitioning Control
  - Chaining
    - Start / disable / configure slot sharing group

DataSet API

```java
public class WordCountExample {
    public static void main(String[] args) throws Exception {
        final ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

        DataSet<String> text = env.fromElements(
            "Who's there?",
            "I think I hear them. Stand, ho! Who's there?");

        DataSet<Tuple2<String, Integer>> wordCounts = text
            .flatMap(new LineSplitter())
            .groupBy(0)
            .sum(1);

        wordCounts.print();
    }

    public static class LineSplitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
        @Override
        public void flatMap(String line, Collector<Tuple2<String, Integer>> out) {
            for (String word : line.split(" ")) {
                out.collect(new Tuple2<String, Integer>(word, 1));
            }
        }
    }
}
```

SQL & Table API



- Both Table API / SQL continuous queries are translated into DataStream/DataSet programs
  - Table is emitted / SQL update query / Table is converted into a DataStream / DataSet

SQL & Table API

```
// get a TableEnvironment
TableEnvironment tableEnv = ...; // see "Create a

// register Orders table

// scan registered Orders table
Table orders = tableEnv.scan("Orders");
// compute revenue for all customers from France
Table revenue = orders
  .filter("cCountry === 'FRANCE'")
  .groupBy("cID, cName")
  .select("cID, cName, revenue.sum AS revSum");

// emit or convert Table
// execute query
```

```
// get a TableEnvironment
TableEnvironment tableEnv = ...; // see "Create a Ta

// register Orders table

// compute revenue for all customers from France
Table revenue = tableEnv.sqlQuery(
    "SELECT cID, cName, SUM(revenue) AS revSum " +
    "FROM Orders " +
    "WHERE cCountry = 'FRANCE' " +
    "GROUP BY cID, cName"
  );

// emit or convert Table
// execute query
```

Hands-on Session

Scenario

- Generate unrecognized client IP alarms from access logs
  - Compare current client IP with previous client ips for email,
  - Send an alarm if the IP is not recognized.

Thanks!