Apache flink

By pranay

Contents

❑ What is Flink?
❑ Why Flink
❑ Flink Ecosystem
❑ Sources and sinks
❑ Flink Program structure
❑ Dataset API and DataStream API
❑ Windowing in flink
❑ Fault tolerance & Availability
❑ Use cases
❑ Limitations

Big Data

- We define big data as huge amount of data.
- Structured, semi-structured and unstructured format
- This led to emergence of certain tools & frameworks that help in storage and processing of data
- Processing part - 1. Batch Processing
  2. Stream Processing

Batch processing
Vs
Stream processing

STREAM PROCESSING

Stream processing challenges

- Data is unbounded, means no start and end
- Unpredictable and inconsistent intervals of new data
- Data can be Out of order with different timestamps
- Latency factor impacts accuracy of results

What is APache FLink?

distributed-Stream processing framework

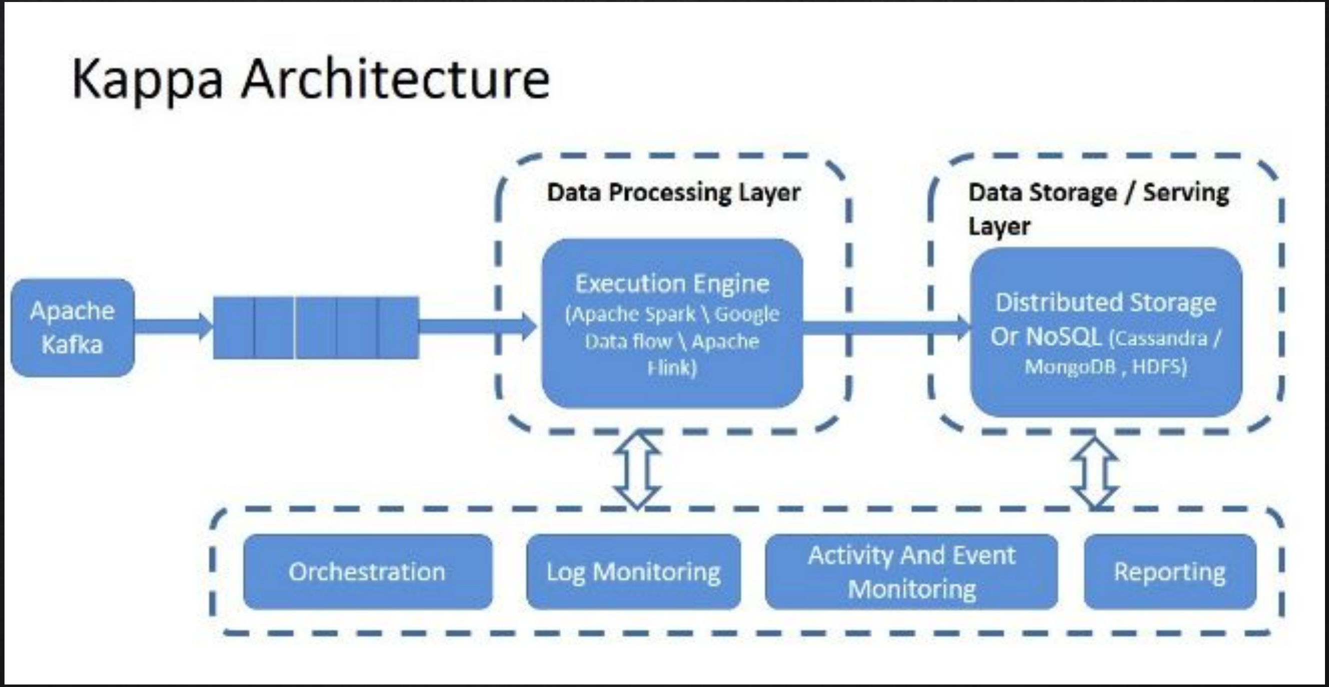
- Scalable
- High throughput
- Low latency
- Flexible windowing

- Exactly once semantics
- High availability
- Fault tolerant

MORE ABOUT FLINK

- Flink works on kappa architecture
- The key idea of this architecture is to handle both batch and real-time data through single stream processing engine.
- Hence, flink treats all data as stream and processes the data in real time.
- Batch data is special case of streaming here.

REmember last slide?



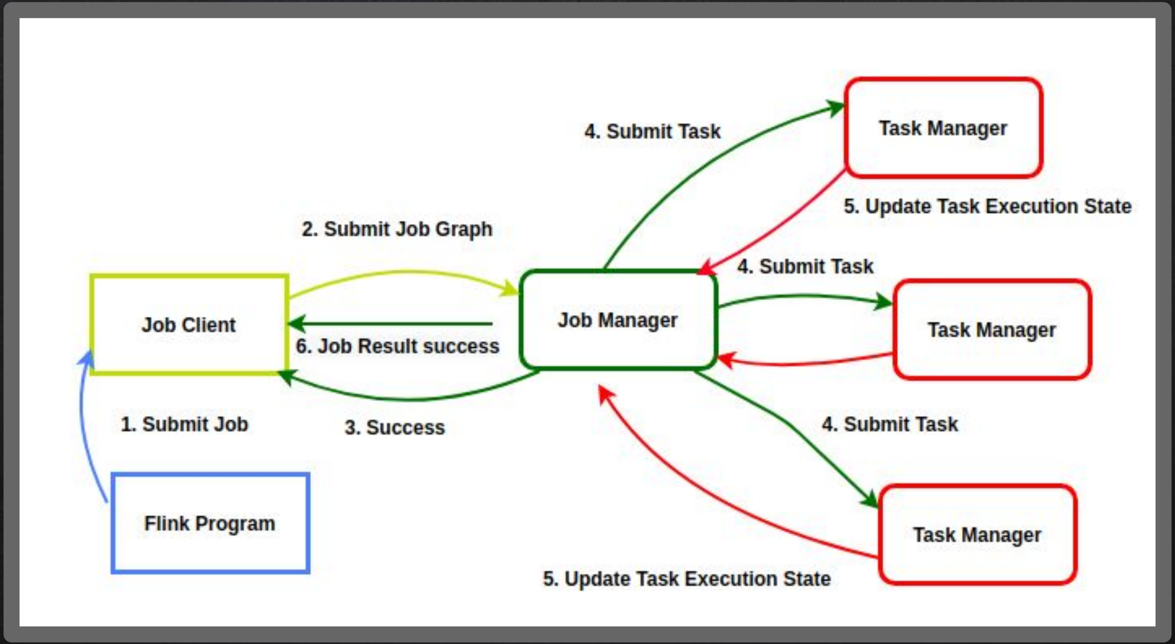
Kappa Architecture

Data Processing Layer
Execution Engine (Apache Spark \ Google Data flow \ Apache Flink)
Data Storage / Serving Layer
Distributed Storage Or NoSQL (Cassandra / MongoDB , HDFS)
Apache Kafka
Orchestration
Log Monitoring
Activity And Event Monitoring
Reporting

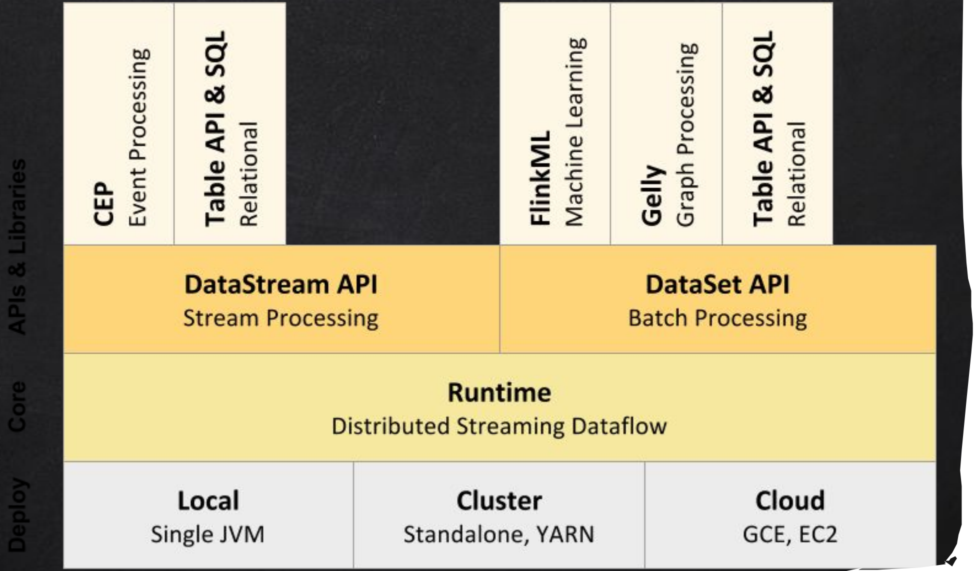
HISTORY IN SHORT

Hadoop < Spark < Flink

Flink job execution architecture



Flink Ecosystem



APIs & Libraries

CEP
Event Processing

Table API & SQL
Relational

FlinkML
Machine Learning

Gelly
Graph Processing

Table API & SQL
Relational

DataStream API
Stream Processing

DataSet API
Batch Processing

Core

Runtime
Distributed Streaming Dataflow

Deploy

Local
Single JVM

Cluster
Standalone, YARN

Cloud
GCE, EC2

Storage/streaming

- Flink is just computation engine, doesn't come with storage system
- Can read, write data from different storage systems as well as streaming systems

➢ Apache kafka(source/sink)
➢ Apache Cassandra(source/sink)
➢ Elastic search(sink)
➢ Hadoop filesystem(sink)
➢ RabbitMQ(source/sink)
➢ Amazon Kinesis streams(source/sink)
➢ Twitter streaming API(source)
➢ Google PubSub(Source/sink)

PreDefined sources/sinks

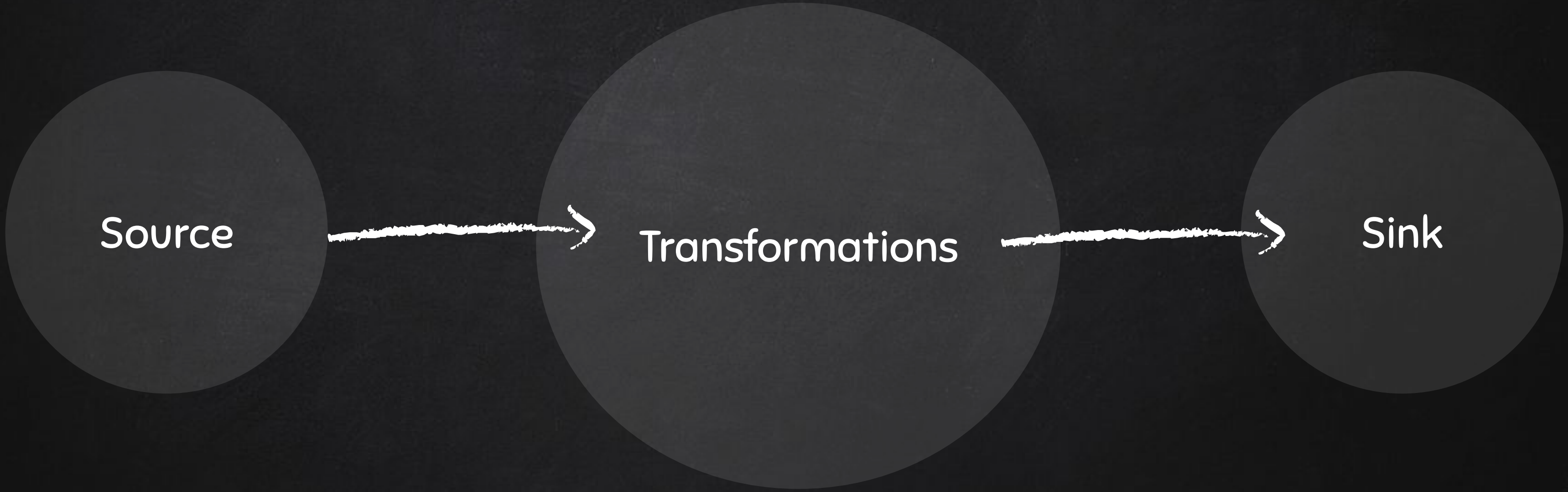
- File based
    - readTextFile(path)
    - readTextFile(fileformat,path)
- Socket-based
    - SocketTextStream
- Collection-based(java.util.Collection)
    - fromCollection(Collection)
    - Elements must be of same type
- Custom(Previous slide)
    - addSource(...)

- writeAsText() – TExt output format
- writeAsCsv() – CSV output format
- print() – standard output
- writeUsingOutputFormat() – custom file output
- WriteToSocket
- Custom(Previous slide)
    - addSink(...)

Some more information

- As you see flink is a layered system. Different layers are built on top of other raising abstraction level of the program
- *Runtime* layer receives program in the form of job graph. This is the layer where job manager sits.
- Both Datastream API and DataSet API generate jobgraphs through separate compilation processes.

Flink Program sKeleton



Source

Transformations

Sink

Flink api's

Dataset API

It is used to perform batch operations on the data over a period. Datasets are created from sources like local files and transformations like filtering, mapping, aggregating, joining are applied and the result is written to sinks.

DataStream API

It is used for handling data in continuous stream. Operations like filtering, mapping, windowing and aggregating can be applied on the datastream and can be written to sinks.

Datastream TRansformations

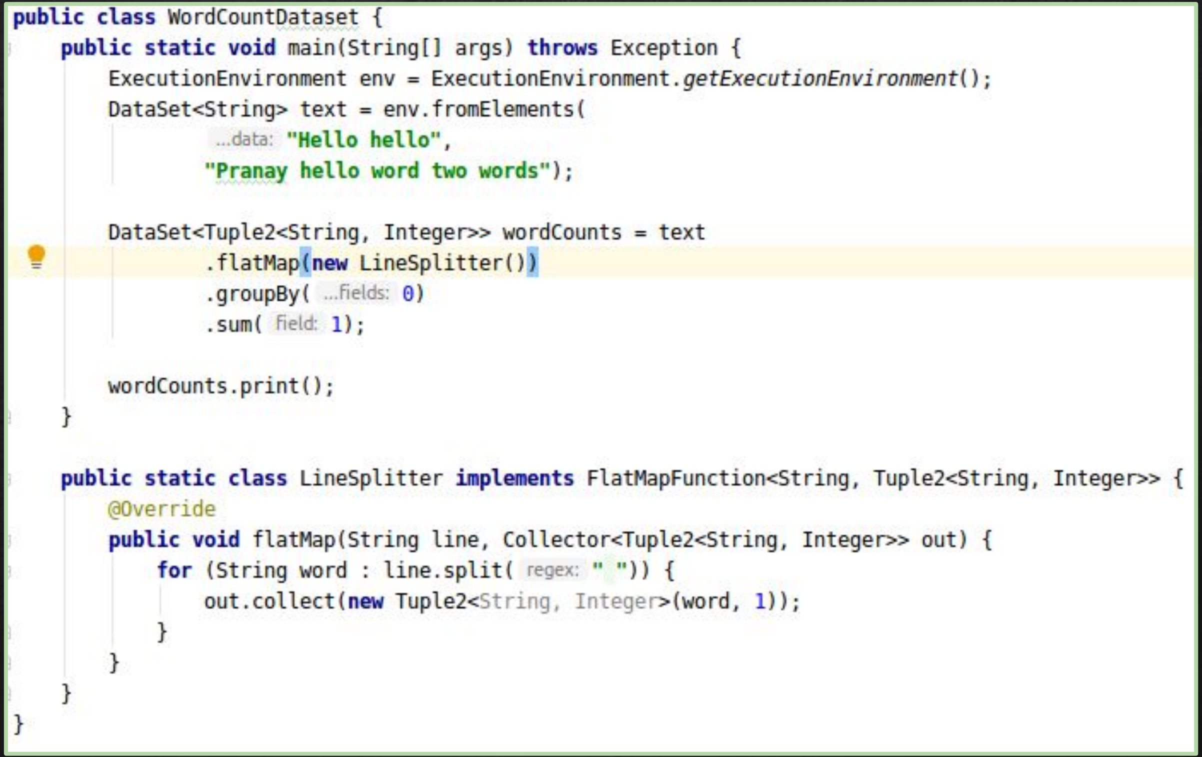
- Map
- FlatMap
- Filter
- KeyBy
- Reduce
- Fold
- Window
  - windowAll
  - Window apply
  - Window fold
  - Window reduce

- Aggregations
  - Sum
  - Min
  - Max
- Aggregate
- Select
- Split
- Iterate
- Extract Timestamps
- Shuffle

HOw it works?

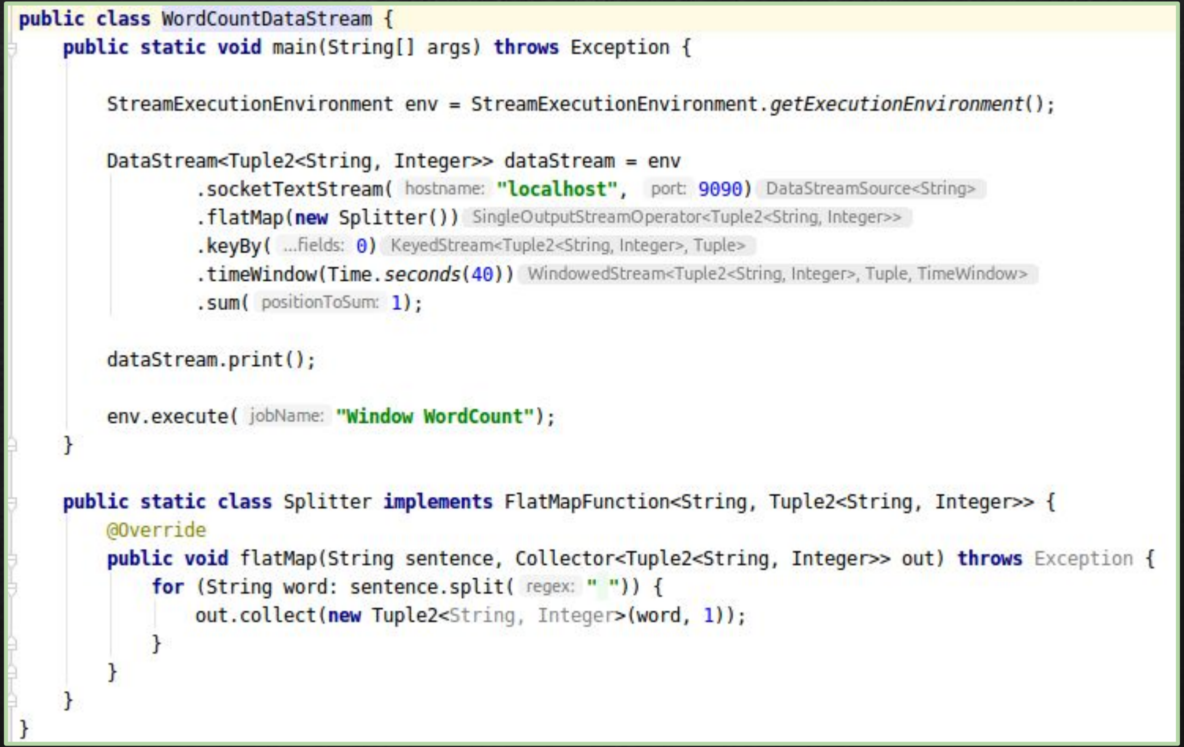
EXAMPLE PLEASE…

DATASET API EXAMPLE



```
public class WordCountDataset {
    public static void main(String[] args) throws Exception {
        ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
        DataSet<String> text = env.fromElements(
                ...data: "Hello hello",
                "Pranay hello word two words");

        DataSet<Tuple2<String, Integer>> wordCounts = text
                .flatMap(new LineSplitter())
                .groupBy( ...fields: 0)
                .sum( field: 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( regex: " ")) {
                out.collect(new Tuple2<String, Integer>(word, 1));
            }
        }
    }
}
```

DATASTREAM API EXAMPLE



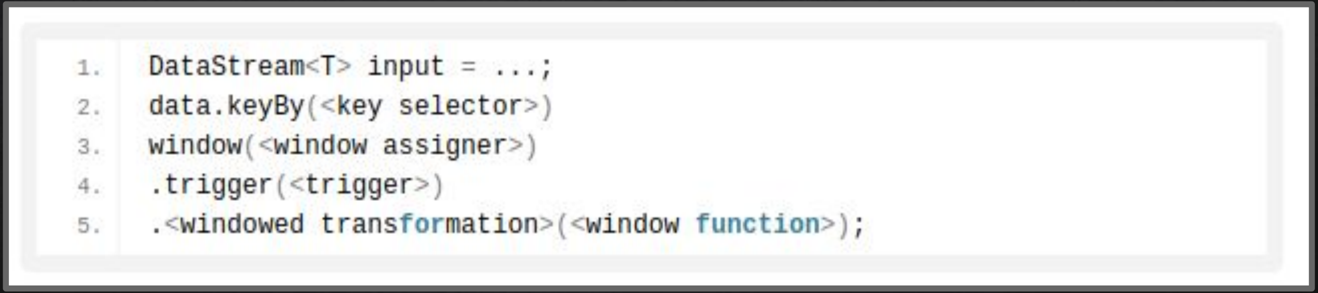
```
public class WordCountDataStream {
    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        DataStream<Tuple2<String, Integer>> dataStream = env
                .socketTextStream( hostname: "localhost",  port: 9090) DataStreamSource<String>
                .flatMap(new Splitter()) SingleOutputStreamOperator<Tuple2<String, Integer>>
                .keyBy( ...fields: 0) KeyedStream<Tuple2<String, Integer>, Tuple>
                .timeWindow(Time.seconds(40)) WindowedStream<Tuple2<String, Integer>, Tuple, TimeWindow>
                .sum( positionToSum: 1);

        dataStream.print();

        env.execute( jobName: "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( regex: " ")) {
                out.collect(new Tuple2<String, Integer>(word, 1));
            }
        }
    }
}
```

Windows In Flink

- Windows are the heart of processing infinite streams.
- Windows split the stream into fixed sized buckets over which we can apply computations.
- It is a mechanism to take snapshot of the stream based on time or other variables
- Need of windowing – in case we need some kind of aggregation on the incoming data

MORE INFO . . .

- Most of window operations are encouraged on keyed datastream.
- The basic structure of windowed transformation is as follows.



```
DataStream<T> input = ...;
data.keyBy(<key selector>)
window(<window assigner>)
.trigger(<trigger>)
.<windowed transformation>(<window function>);
```

Window Assigners . . .

A window assigner defines how elements are assigned to windows

❏ Tumbling Windows

❏ Sliding Windows

❏ Sessions Windows

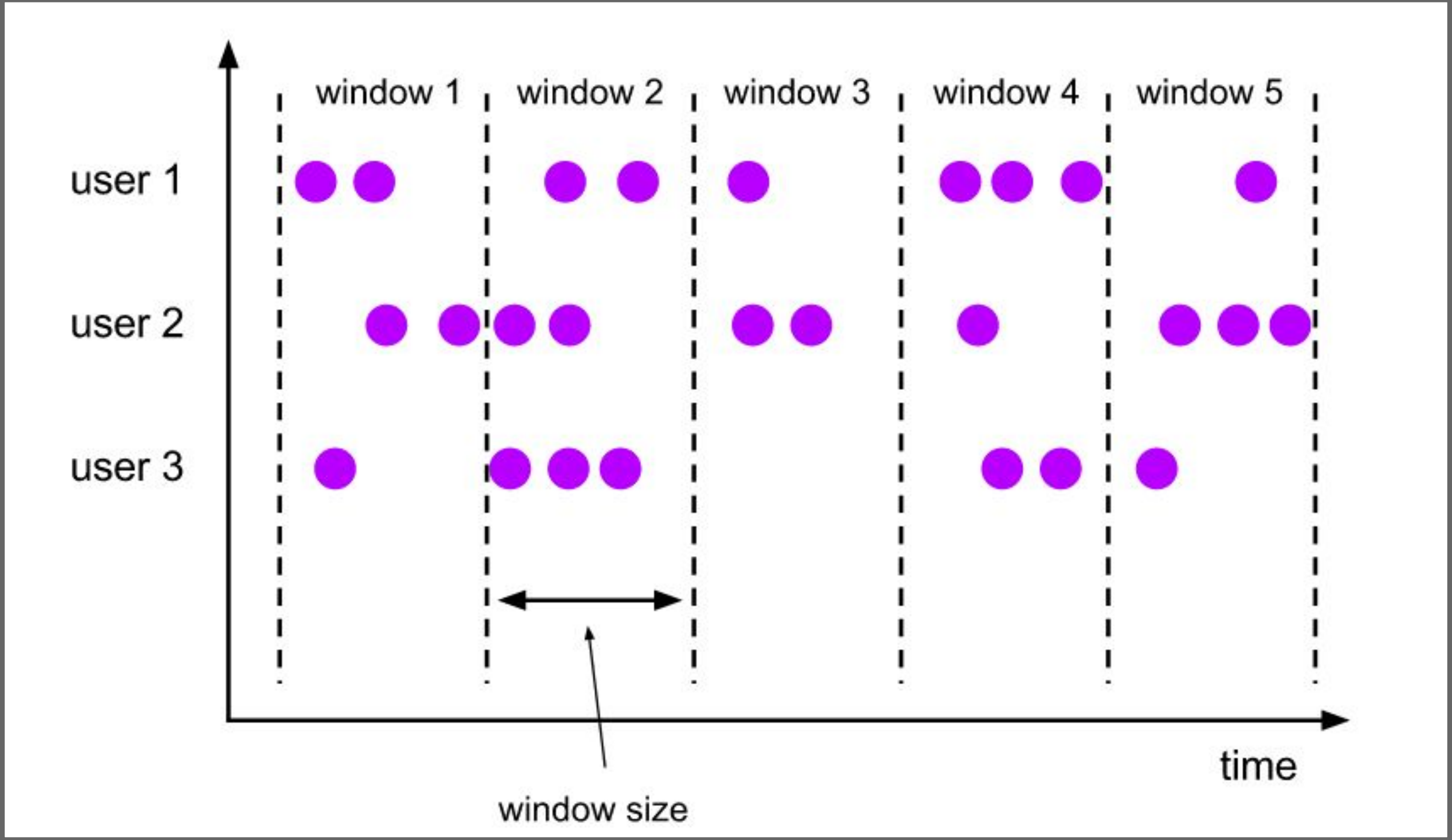
❏ Global Windows

TIME IN FLINK

- Processing Time – Assigns elements based on the current clock of the machines
- Event Time – Assigns windows based on timestamp of the elements.
- Ingestion time – Assigns wall clock timestamps as they arrive in the system and processes as event time semantics based on the attached timestamps.

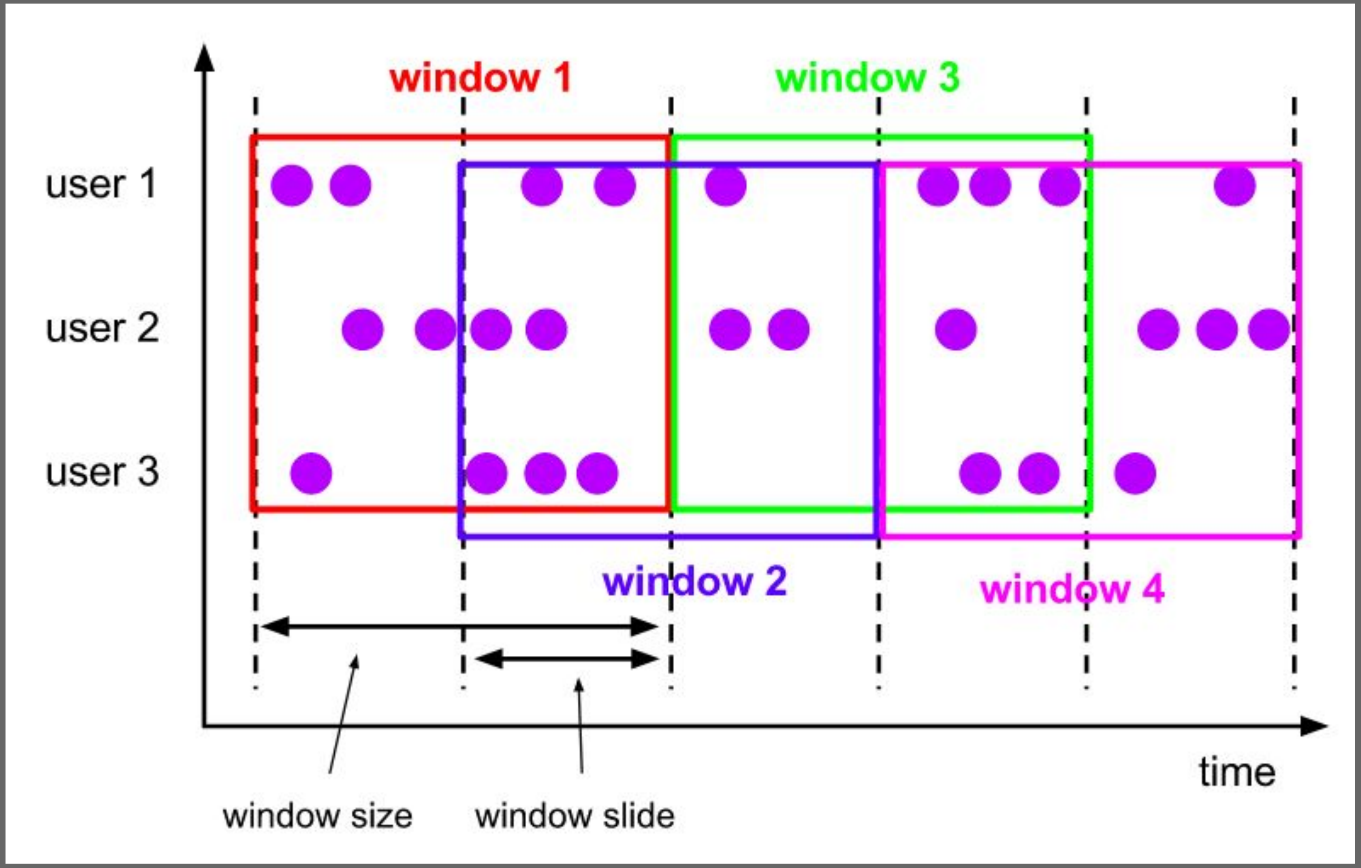
Tumbling windows

- ✘ Here window size is specified and elements are assigned to non overlapping windows.
- ✘ Eg. If window size is specified as 2 minutes, all elements in 2 minutes would come in one window for processing.
- ✘ It is useful in dividing the stream into multiple discrete batches.


window 1
window 2
window 3
window 4
window 5
user 1
user 2
user 3
time
window size

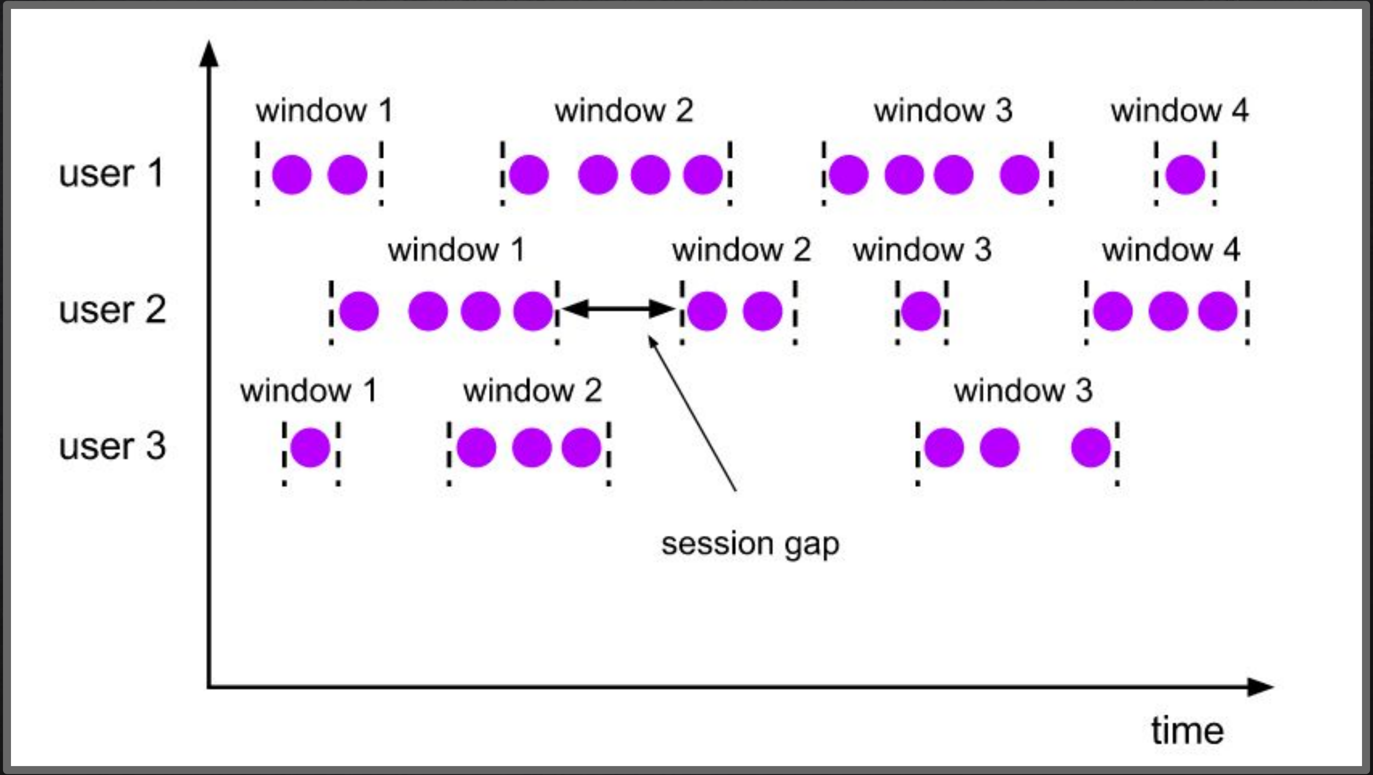
Sliding windows

- Here windows can overlap unlike tumbling windows
- Overlap is user specified. Here one element can be assigned to multiple windows
- Eg. 10 minutes window with a slide of 5 minutes.


window 1
window 3
user 1
user 2
user 3
window 2
window 4
window size
window slide
time

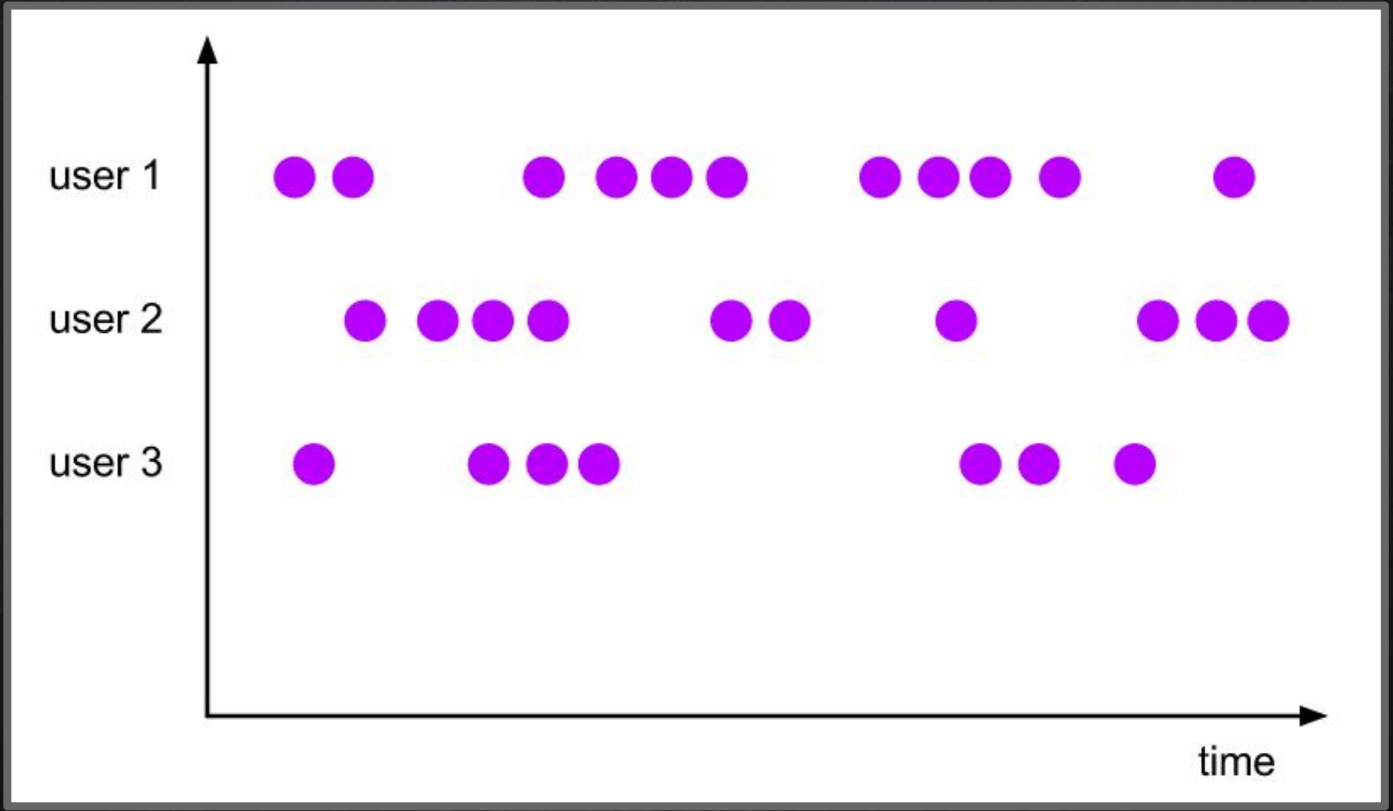
Session Windows

- This is applicable cases where window boundaries needs to be adjusted as per incoming data.
- Here we specify session gap or inactivity period before marking a session closed.
- Refer next slide for example.


window 1
window 2
window 3
window 4
user 1
window 1
window 2
window 3
window 4
user 2
window 1
window 2
window 3
user 3
session gap
time

Global Windows

- Sub-division of elements into windows is not done. Instead, we assign each element to 1 single key per global window.
- There is no end to this window, so triggers should be used if any computation needs to be done.
- Flink also has count windows. For eg. a tumbling count window of 100 will add 100 elements to the window and will evaluate upon receiving 100th element.


user 1
user 2
user 3
time

Evictors & Triggers

- A trigger determines when window is ready for processing. Except Global windows, all windows come with a default trigger.
- We can define a custom trigger which tells when to fire elements for processing .
- Evictors are used to evict some elements from the window. After the window is completed, it is purged

Fault Tolerance

- Before diving in, let’s define stateful computing. Most of the computational tasks require stateful data.
- We already came across this in previous slides.
- In word count example, word count is an output that accumulates new objects into existing word count. It is a stateful variable.
- Here fault tolerance is achieved with checkpointing. Flink maintains state locally per task.
- State is periodically checkpointed to durable storage. A checkpoint is consistent snapshot of the state of all tasks.

More on Checkpointing . . .

- ➢ Flink backs up state at a certain interval
- ➢ In case of a failure, flink recovers all tasks to the state of last checkpoint and starts running the tasks from that checkpoint.
- ➢ The application continues as the failure never happened.
- ➢ Flink guarantees exactly once semantics which means all data is processed exactly once regardless of any failures.

Stopping and resuming jobs . . .

- ➢ The running jobs must be stopped before upgrading a component and resumed after the upgrade.
- ➢ There are 2 modes to resume the jobs.
- ➢ Savepoint – unlike checkpoint which is periodically triggered by the system, savepoint is triggered by running commands.
- ➢ Even the storage format is different from checkpoints.
- ➢ External checkpoint – Extension of existing internal checkpoint. After storing internal checkpoint, it is stored specific directory.

Savepoints vs checkpoints . . .

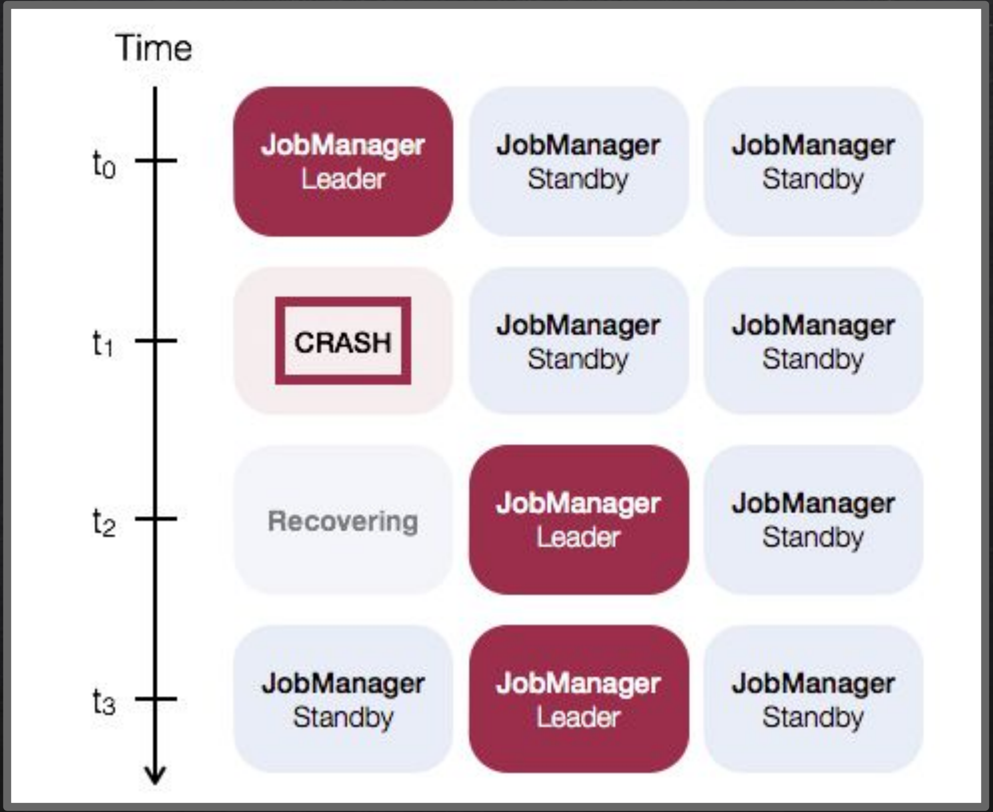
- The primary purpose of checkpoints is to provide a recovery mechanism in case of unexpected job failures. Checkpoint’s lifecycle is managed by flink. Checkpoints are dropped after job was terminated by user. They are lightweight.
- Savepoints are created, owned and deleted by user. Their use-case is for planned manual backup and resume. They are bit of expensive. Mostly used for flink version upgrade, changing job graph and for changing parallelism.

More on checkpoints. . .

- We already know state is persisted upon checkpoints to avoid data loss and recover consistently. How the state is stored internally and how it is persisted upon checkpoints depends on state backend.
- 3 state backends available
    - Memory state backend
    - Fs state backend
    - Rocksdb state backend

High Availability for standalone cluster

- ➢ Job manager coordinates flink deployment and it is responsible for scheduling the jobs and resource management.
- ➢ However, there is only one job manager per flink cluster creating a single point of failure.
- ➢ There will be a single leading job manager at any time and multiple standby job managers to take leadership incase leader fails.
- ➢ Flink uses zookeeper for distributed coordination between all running instances of job manager and leader election.



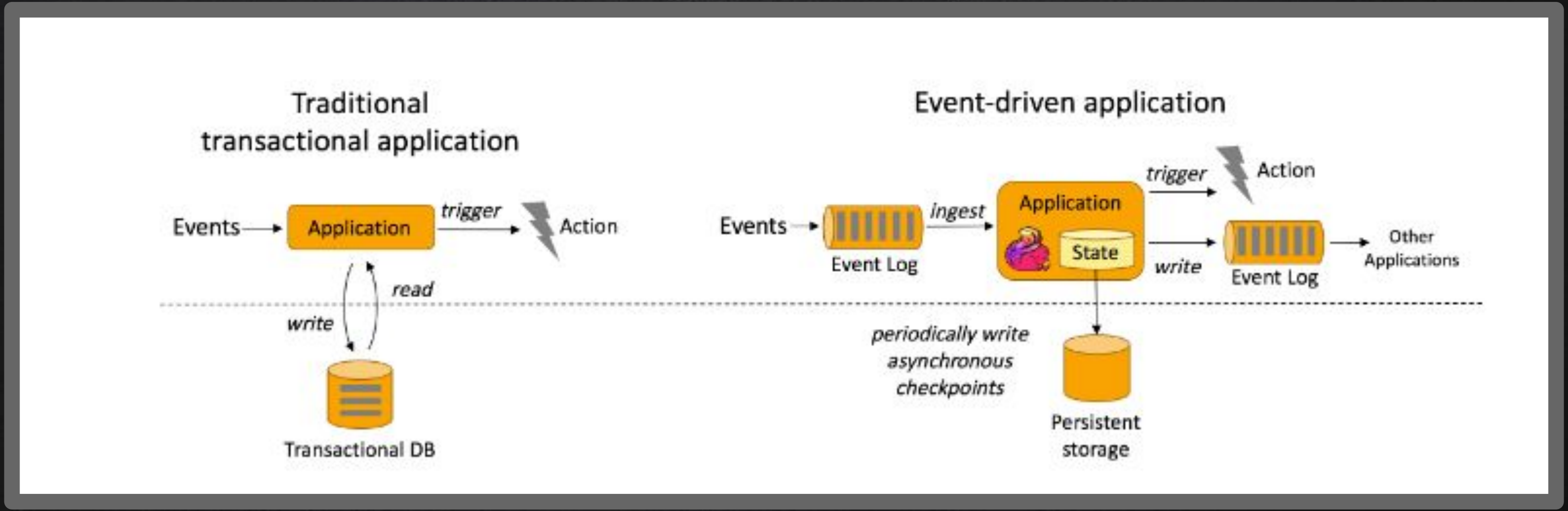
Time			
t_0	JobManager Leader	JobManager Standby	JobManager Standby
t_1	CRASH	JobManager Standby	JobManager Standby
t_2	Recovering	JobManager Leader	JobManager Standby
t_3	JobManager Standby	JobManager Leader	JobManager Standby

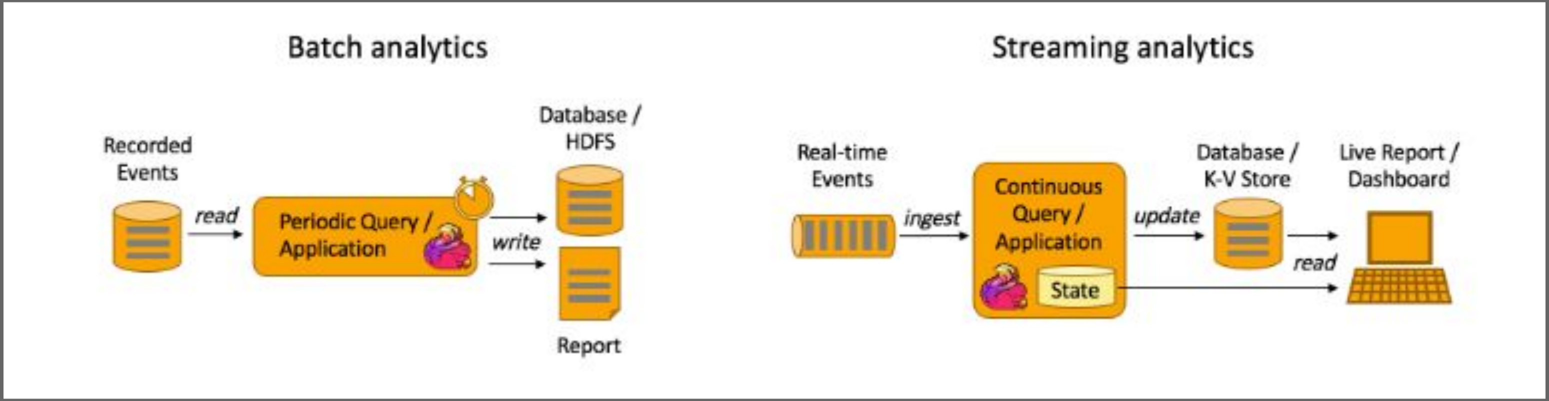
Limitations ✖

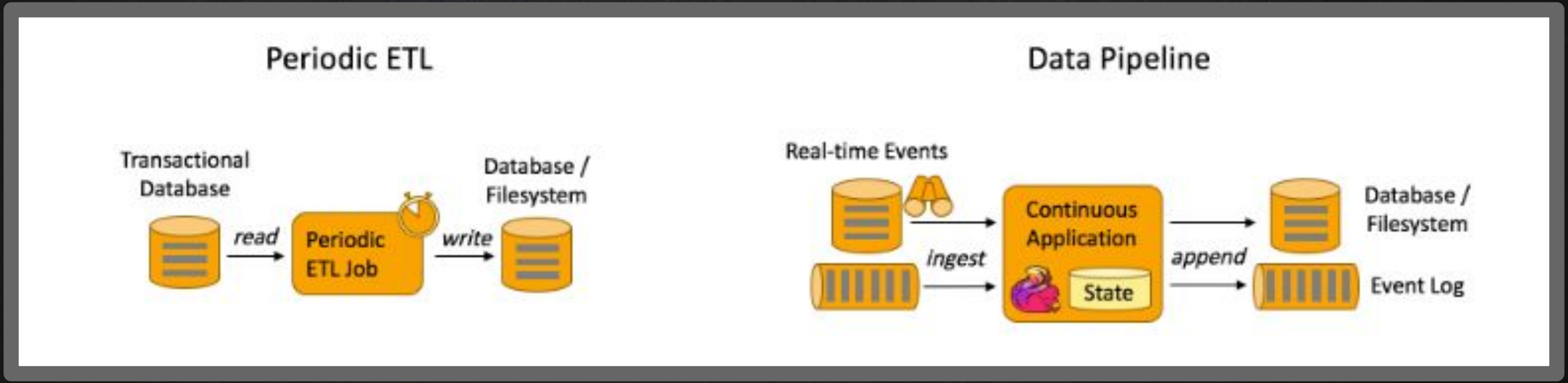
- Maturity and community support is less in the industry than its predecessors but it is growing fastly.
- No known adaptation of Flink batch, only popular for streaming

Use Cases

- ❑ Event driven Applications
- ❑ Data Analytics Applications
- ❑ Data Pipeline Applications


Traditional
transactional application
Events
Application
trigger
Action
read
write
Transactional DB
Event-driven application
Events
Event Log
ingest
Application
State
trigger
Action
write
Event Log
Other
Applications
periodically write
asynchronous
checkpoints
Persistent
storage


Batch analytics
Recorded Events
read
Periodic Query / Application
write
Database / HDFS
Report
Streaming analytics
Real-time Events
ingest
Continuous Query / Application
State
update
Database / K-V Store
read
Live Report / Dashboard


Periodic ETL
Transactional Database
read
Periodic ETL Job
write
Database / Filesystem
Data Pipeline
Real-time Events
ingest
Continuous Application
State
append
Database / Filesystem
Event Log

EXAMPLES

Event driven applications

- ➢ Fraud detection
- ➢ Anomaly detection
- ➢ Rule Based Alerting
- ➢ Business Process Monitoring

Data Analytics Applications

- ➢ Quality monitoring of Telecom networks
- ➢ Analysis of live data
- ➢ Analysis of product updates
- ➢ Large scale graph analysis

Data Pipeline Applications

- ➢ Continuous ETL in E-commerce
- ➢ Real time search index building in E-commerce

Use cases cont'd. . .

Bouygues Telecom

For real time event processing and analytics for billions of messages per day and real time customer insights.

King – candy crush creator

Real time analytics on 30 billion events generated everyday across 200 countries.

Zalando E-commerce

Uses flink for real time process monitoring, stream processing, business process monitoring.

Otto – largest online retailer

For user agent identification and identifying a search session via stream processing,

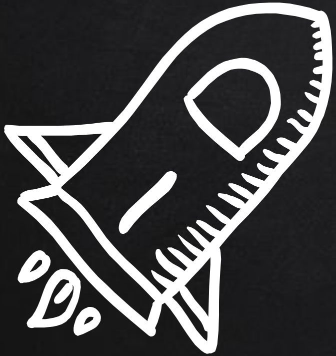
Alibaba Group

For online recommendations of the products based on the user activity. They developed Blink on top of the flink for this.

Capital One – commercial banking

Monitor customer data in real time, to detect customer issues, event correlation, anomaly identification

Thank You!



Keep Learning . . .