Apache Flink™ Training

Advanced Stream Processing

Tzu-Li (Gordon) Tai

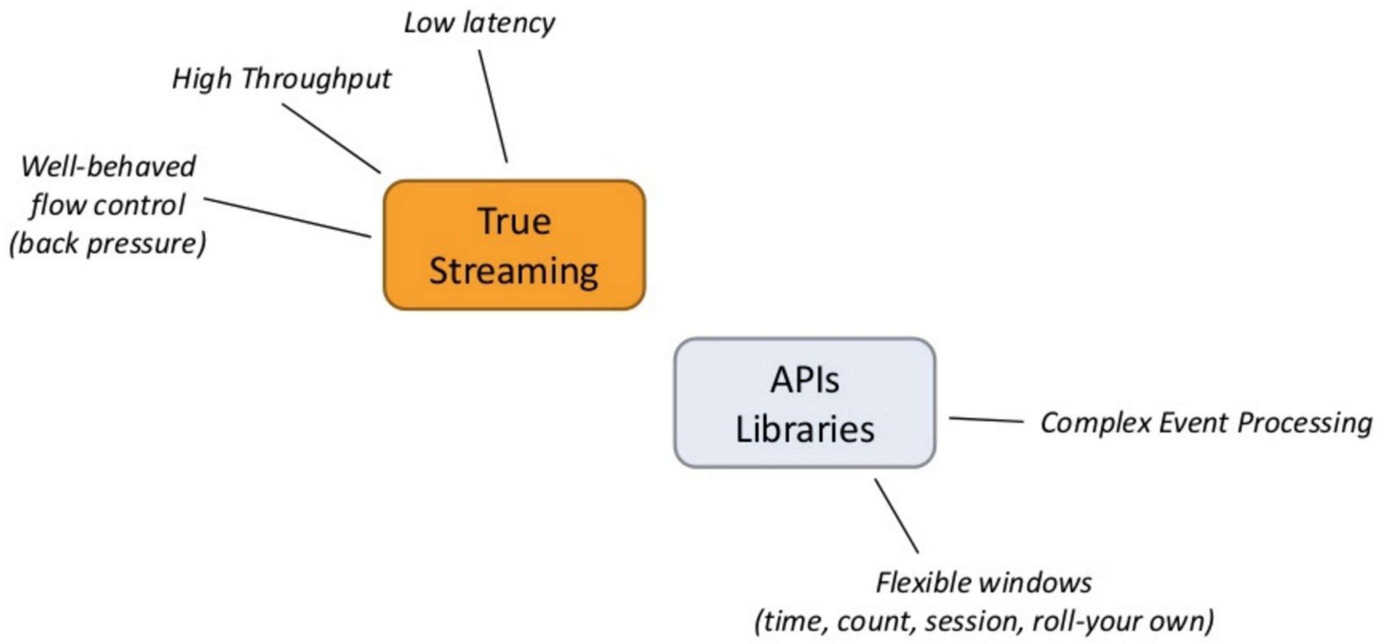
tzulitai@apache.org

@tzulitai

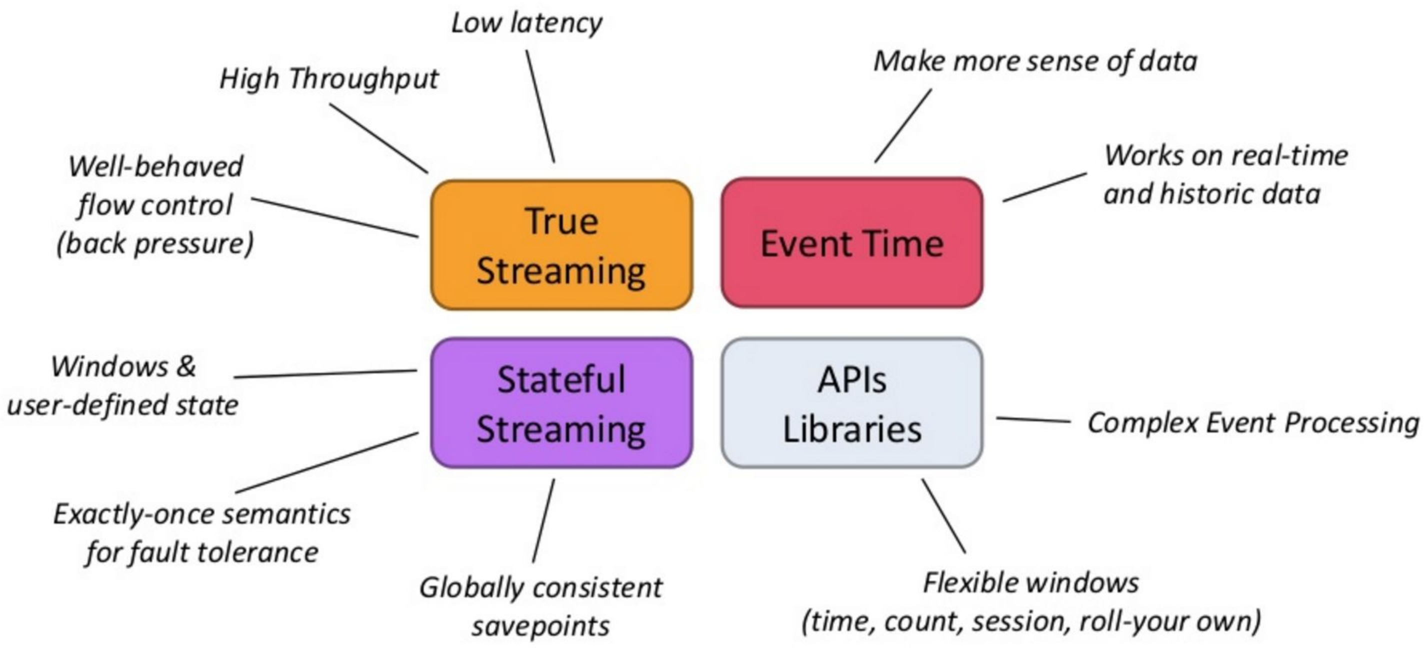
Flink.tw
Apache Flink Taiwan User Group

Sept 2016 @ HadoopCon

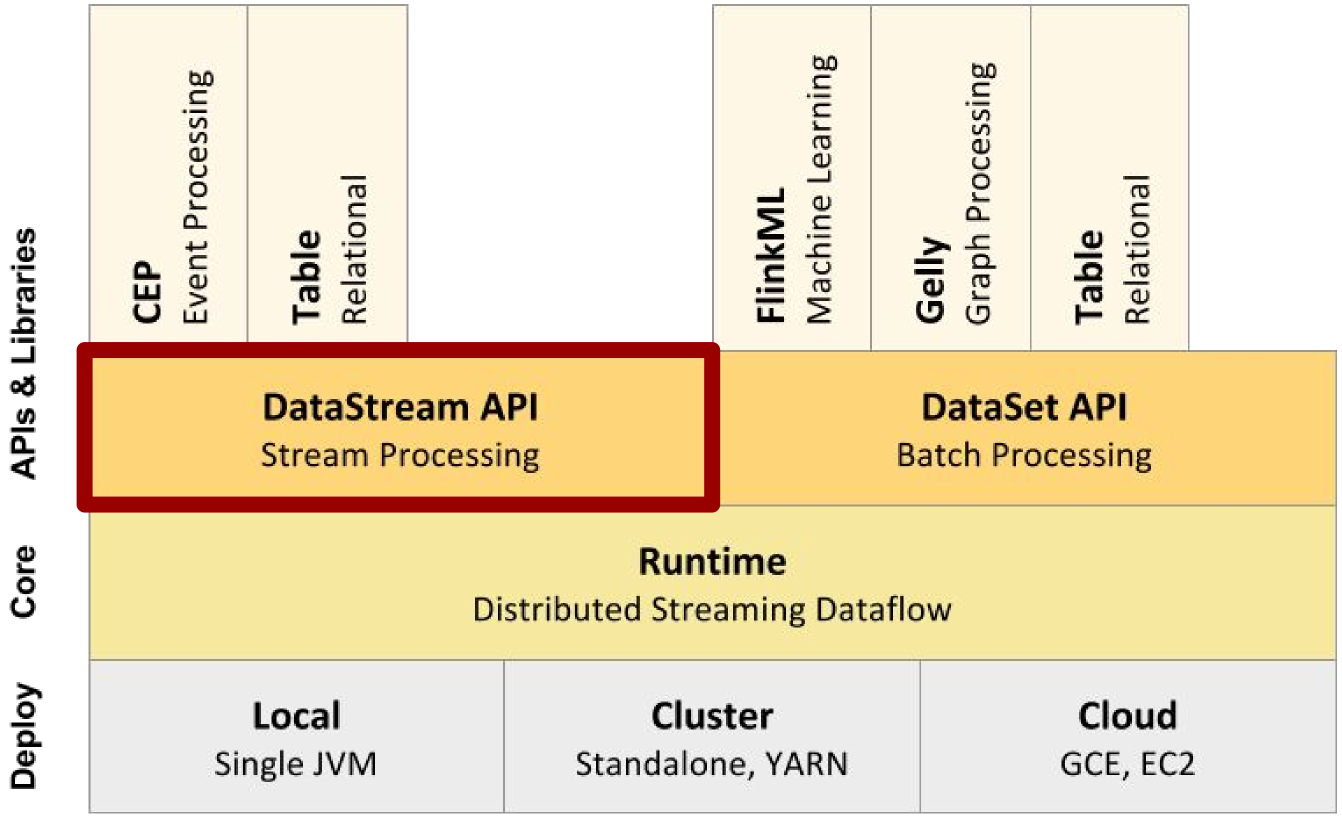
00 This session will be about ...


Low latency
High Throughput
Well-behaved flow control (back pressure)
True Streaming
APIs Libraries
Complex Event Processing
Flexible windows (time, count, session, roll-your own)

00 This session will be about ...


Low latency
High Throughput
Make more sense of data
Well-behaved flow control (back pressure)
True Streaming
Event Time
Works on real-time and historic data
Windows & user-defined state
Stateful Streaming
APIs Libraries
Complex Event Processing
Exactly-once semantics for fault tolerance
Globally consistent savepoints
Flexible windows (time, count, session, roll-your own)

00 This session will be about ...



00 This session will be about ...

- Flink’s notion of time in streaming jobs
- How *Watermarks* support Event-Time Processing

- Flink’s fault-tolerant, exactly-once streaming semantics
- Flink’s distributed snapshot checkpointing

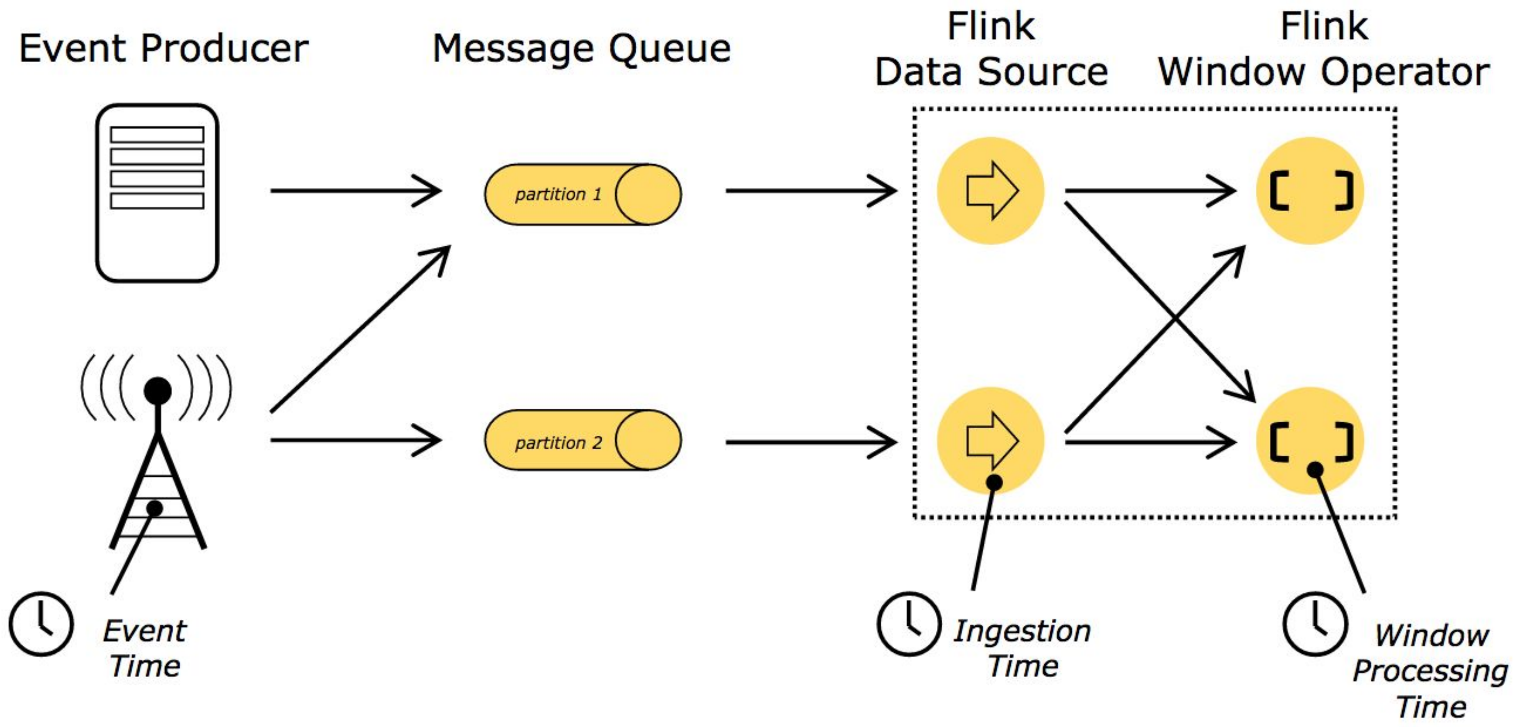
- Out-of-core streaming state backends

Flink’s Notions of Time

01 Different Kinds of “Time”

- ***Processing Time:***
  - The timestamp at which a system processes an event
  - “Wall Time”
- ***Ingestion Time:***
  - The timestamp at which a system receives an event
  - “Wall Time”
- ***Event Time:***
  - The timestamp at which an event is generated

01 Different Kinds of “Time”



02 Why Wall Time is Incorrect

- Think *Twitter hash-tag count every **5 minutes***
    - We would want the result to reflect the number of Twitter tweets actually tweeted in a 5 minute window
    - ***Not*** the number of tweet events the stream processor receives within 5 minutes

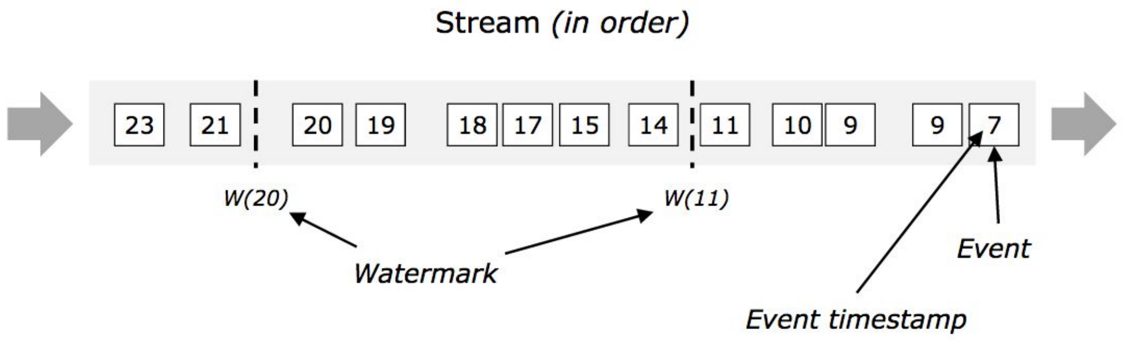
02 Why Wall Time is Incorrect

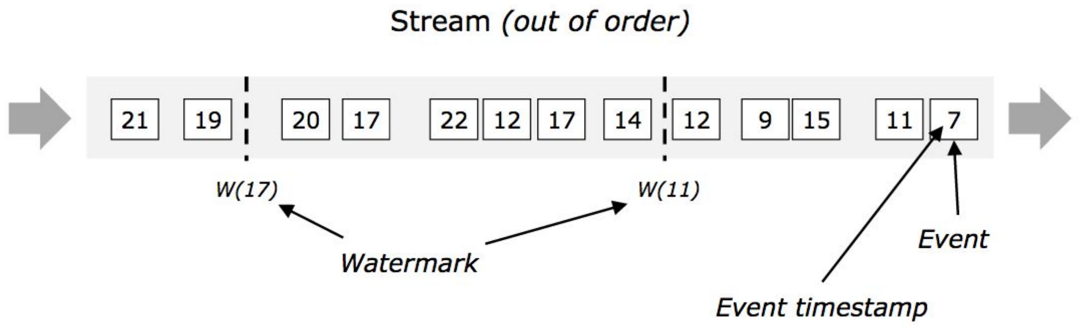
- Think *replaying a Kafka topic on a windowed streaming application …*
    - If you're replaying a queue, windows are definitely wrong if using a wall clock

03 Watermarks & Event-Time

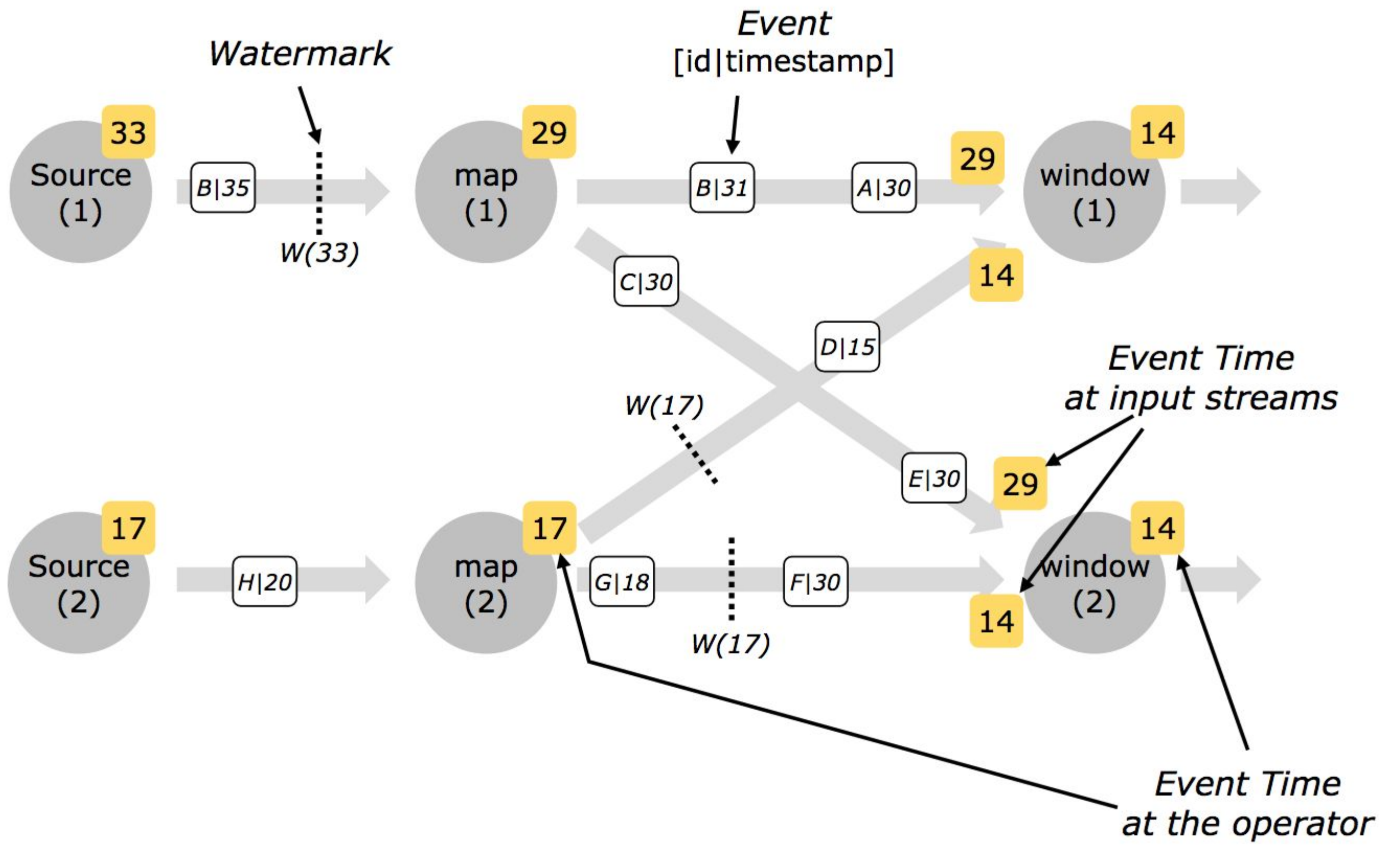
- ***Watermarks*** is a way to let Flink monitor the progress of event time
- Essentially a record that flows within the data stream
- Watermarks carry a timestamp ***`t`***. When a task receives a ***`t`*** watermark, it knows that there will be no more events with timestamp ***`t′ < t`***

04 Watermarks & Event-Time





05 Watermarks in Parallel Streams



Flink.tw
Apache Flink Taiwan User Group

06 Event-Time Processing API

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
```

Tell Flink to use “Event Time”

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)

val stream: DataStream[MyEvent] = env.readFile(
         myFormat, myFilePath, FileProcessingMode.PROCESS_CONTINUOUSLY, 100,
         FilePathFilter.createDefaultFilter());

val withTimestampsAndWatermarks: DataStream[MyEvent] = stream
        .filter( _.severity == WARNING )
        .assignTimestampsAndWatermarks(new MyTimestampsAndWatermarks())

withTimestampsAndWatermarks
        .keyBy( _.getGroup )
        .timeWindow(Time.seconds(10))
        .reduce( (a, b) => a.add(b) )
        .addSink(...)
```

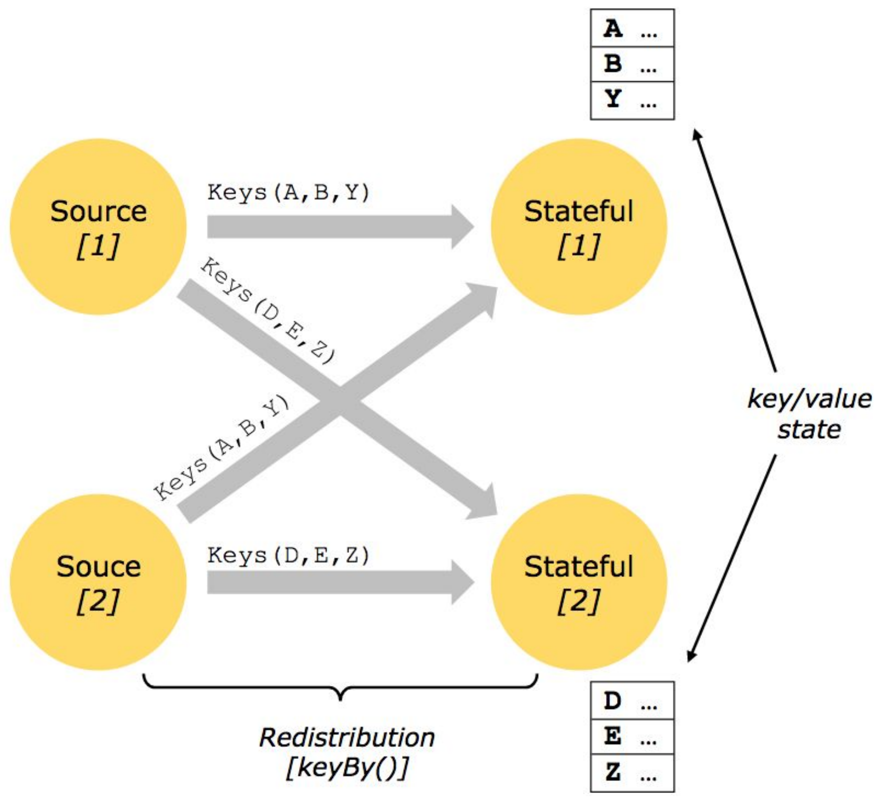
Assign event timestamps and watermarks

Exactly-Once Streaming Fault-Tolerance

07 Stateful Streaming

- ***Any*** non-trivial streaming application is stateful
- To draw insights from a stream you usually need to look beyond a single record
- Any kind of aggregation is stateful (ex. windows)

08 What "state" looks like in Flink



- Any Flink task can be stateful
- State is partitioned with the streams that are read by stateful tasks

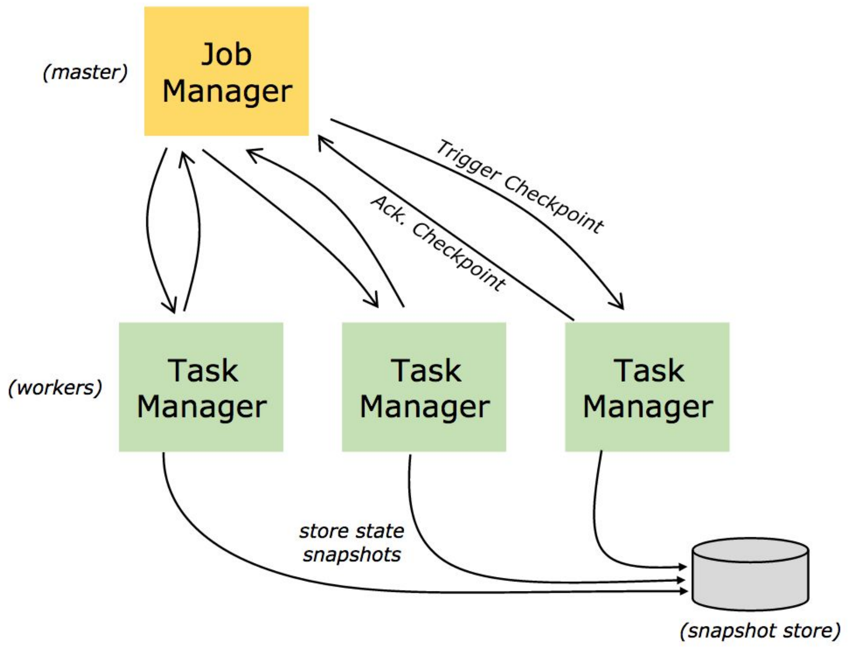
Flink.tw Apache Flink Taiwan User Group

09 Distributed Snapshots

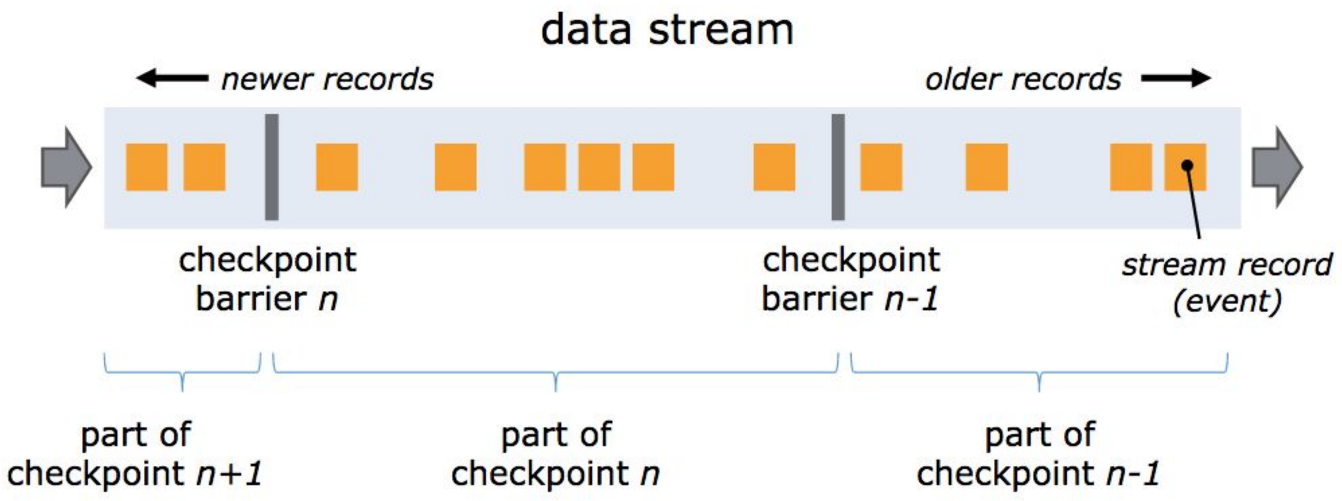
- On each checkpoint trigger, task managers tell all stateful tasks that they manage to snapshot their own state

- When complete, send checkpoint acknowledgement to JobManager

- *Chandy Lamport Distributed Snapshot Algorithm*

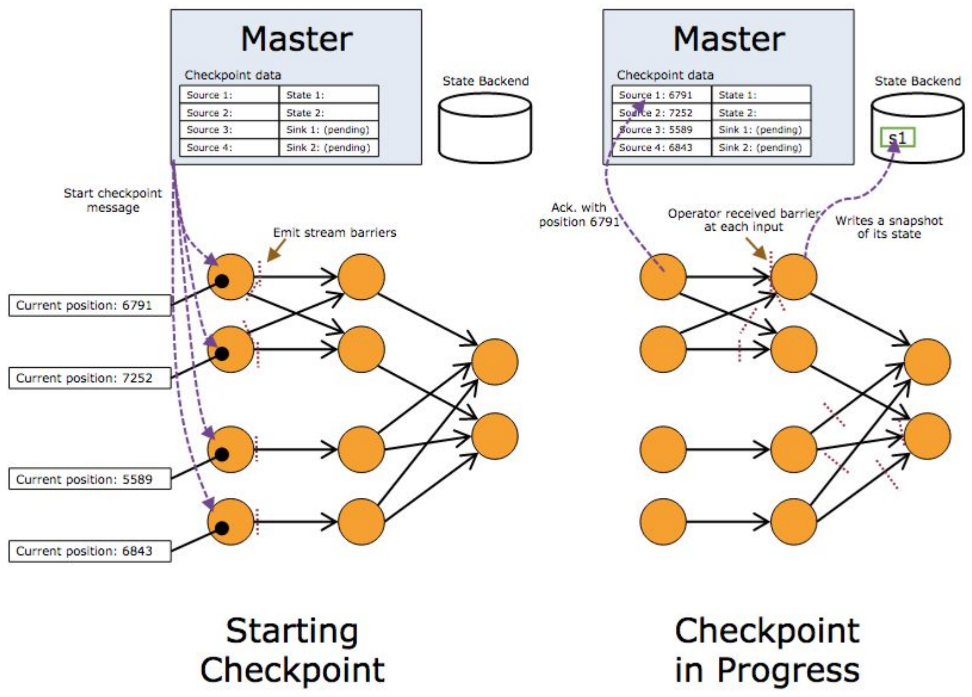


09 Distributed Snapshots



- On a checkpoint trigger by the JobManager, a ***checkpoint barrier*** is injected into the stream

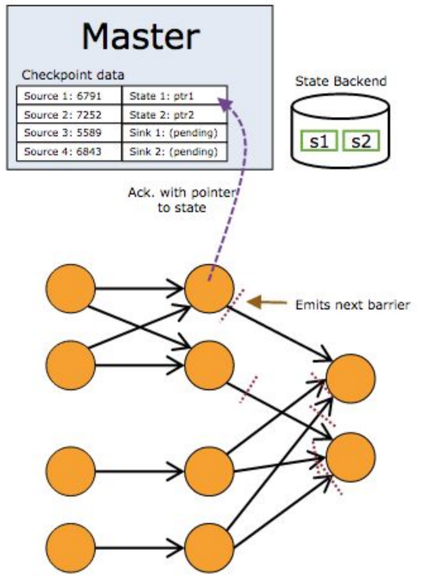
10 Distributed Snapshots



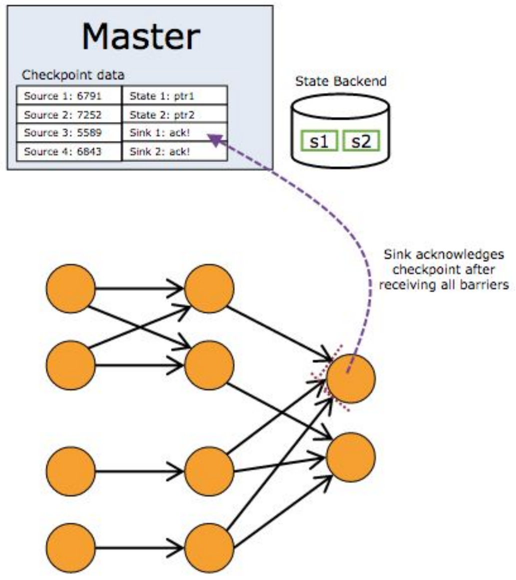
- When a task receives a checkpoint barrier, its state is checkpointed to a state backend

- A pointer value to the stored state is stored in the distributed snapshot

Distributed Snapshots



Checkpoint in Progress



Checkpoint Completed

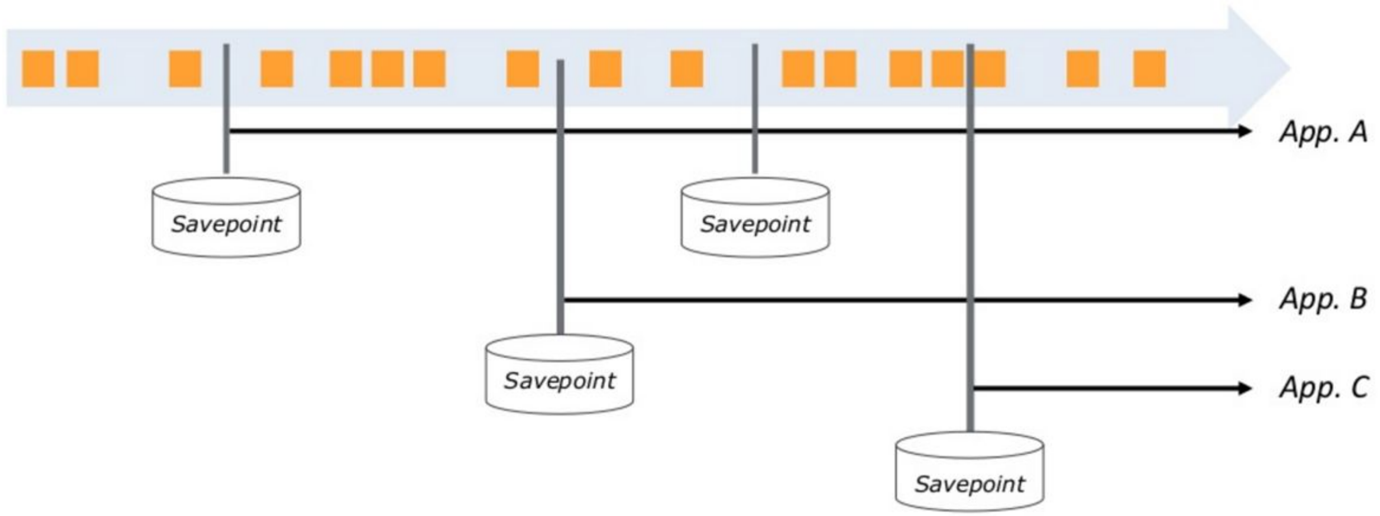
- After all stateful tasks acknowledges, the distributed snapshot is completed
- Only fully completed snapshots are used for restore on failure

12 Checkpointing API

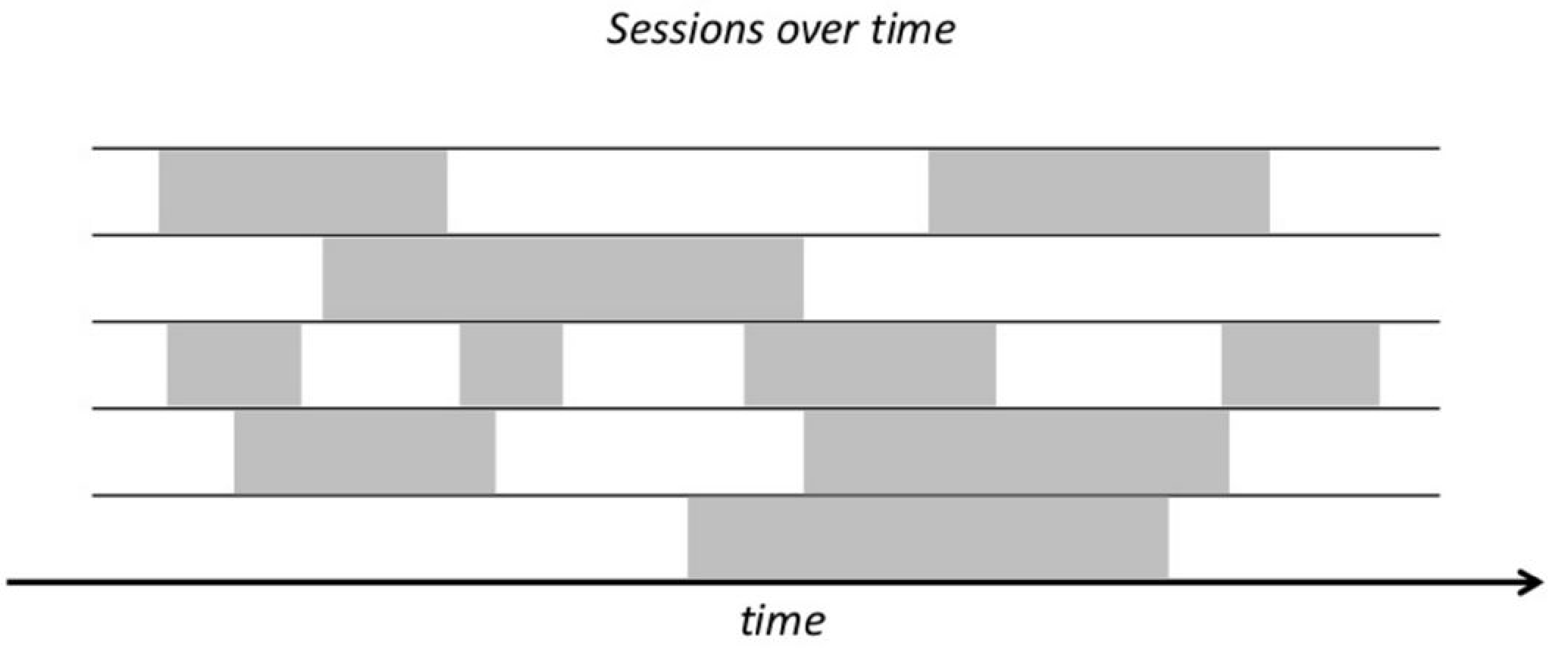
```
val env = StreamExecutionEnvironment.getExecutionEnvironment()

env.enableCheckpointing(100) // trigger checkpoint every 100ms
env.setStateBackend(new RocksDBStateBackend(...))
```

13 Flink Streaming Savepoints

- Basically, a checkpointed that is persisted in the state backend
- Allows for stream progress “versioning”

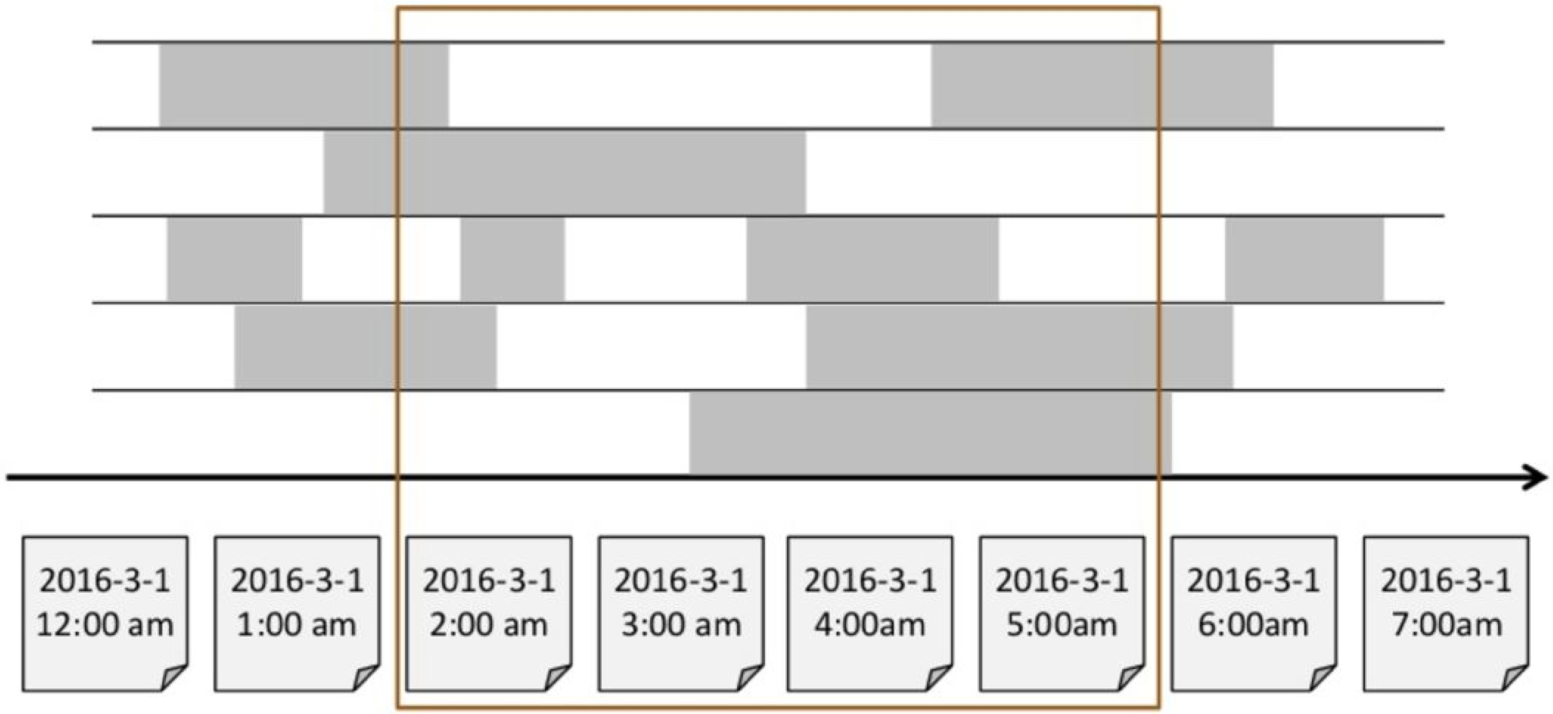


14 Power of Savepoints



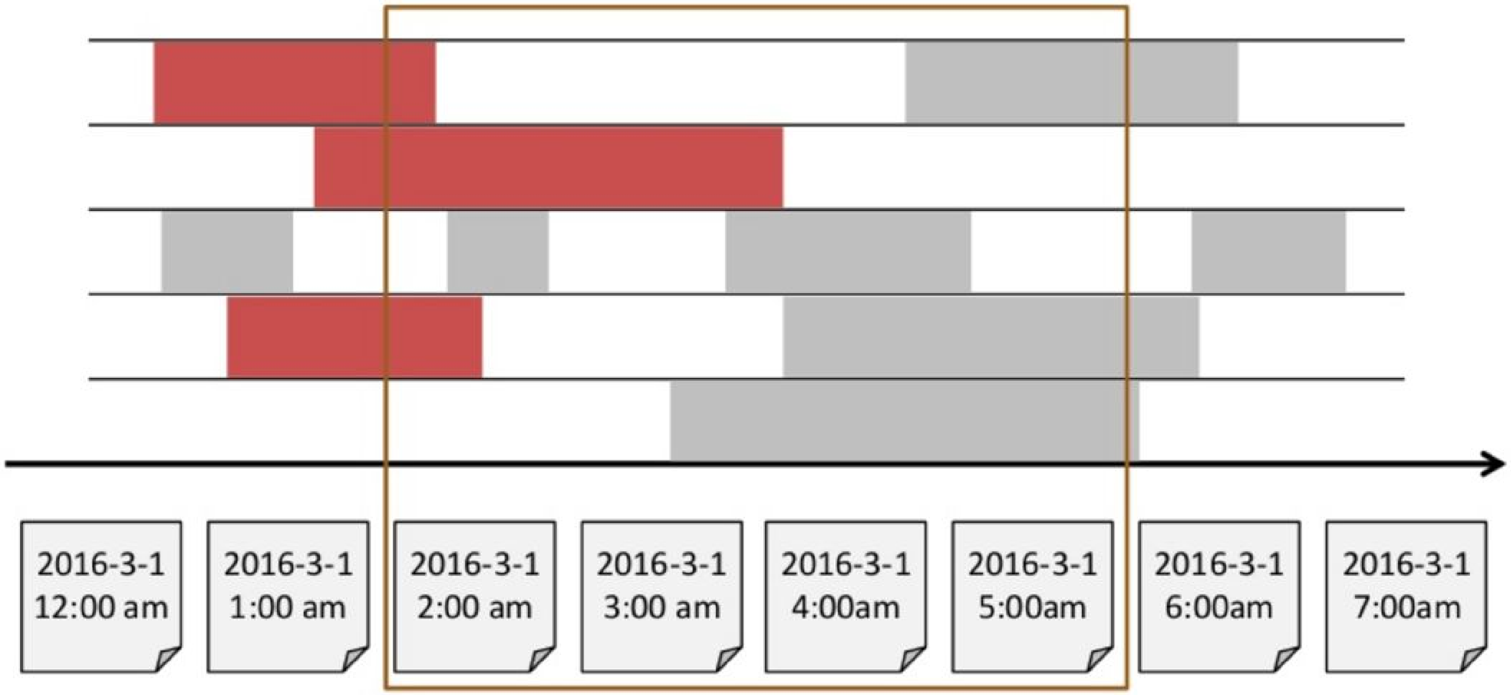
- No stateless point in time

14 Power of Savepoints



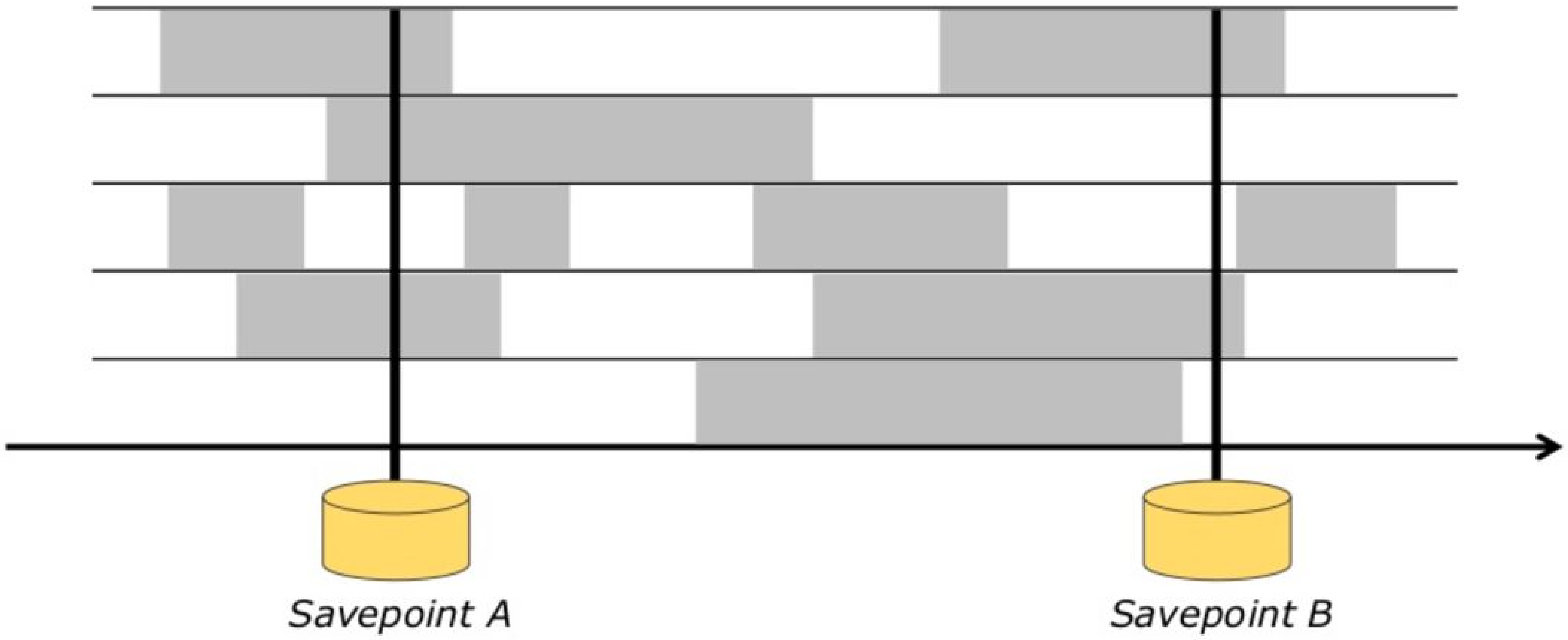
- Reprocessing as batch

14 Power of Savepoints



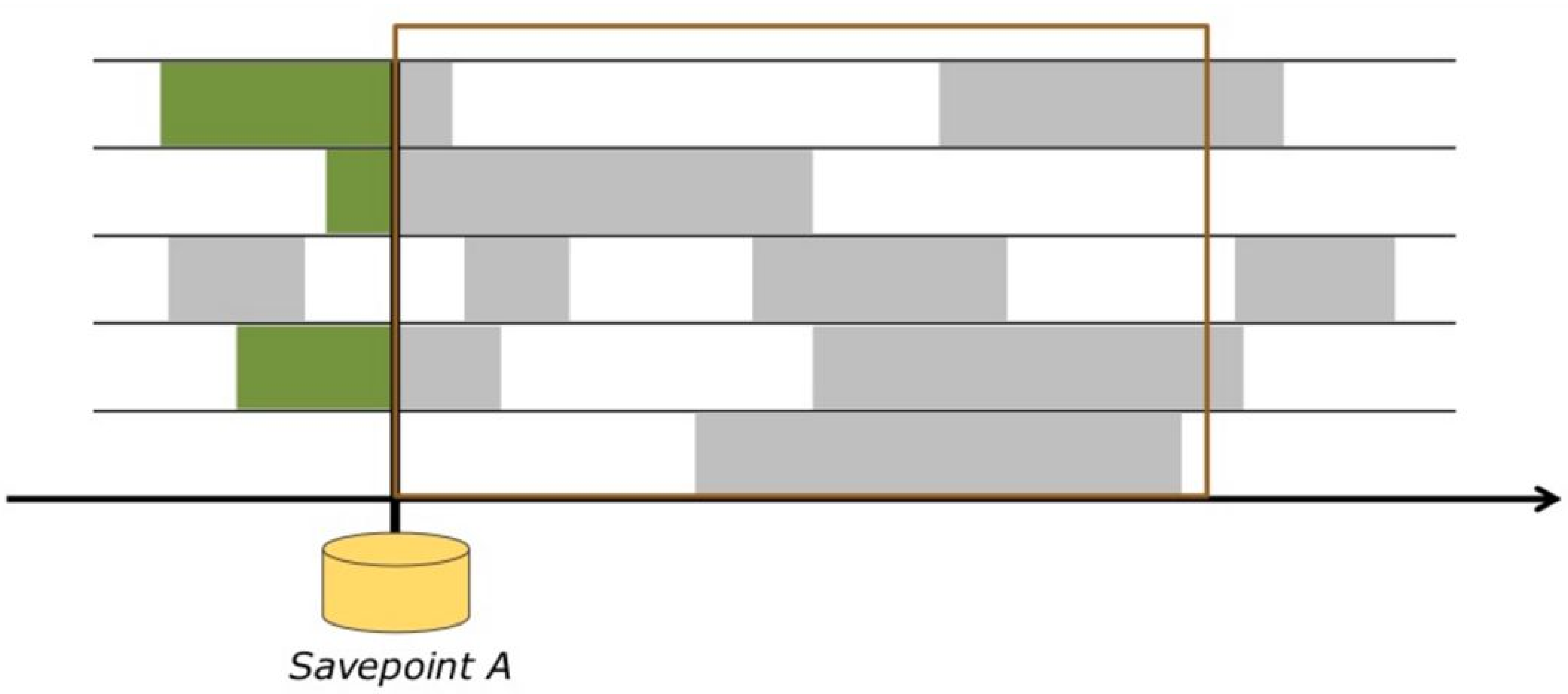
- Reprocessing as batch (*corrupt state*)

14 Power of Savepoints



- Reprocessing as streaming, starting from savepoint

15 Power of Savepoints



- Reprocessing as streaming, starting from savepoint