Stream Processing beyond streaming data – Batch, Streaming, and Applications

Timo Walther
PMC of Apache Flink
Ververica (formerly dataArtisans, now part of Alibaba Group)

Follow me: @twalthr (yes, without the e)

© 2019 Ververica

Alternative Talk Titles

"Batch is a special case of streaming"

"If all you have is a Squirrel, everything looks like a stream"

"What's taking you so long to
merge DataSet and DataStream?"

© 2019 Ververica

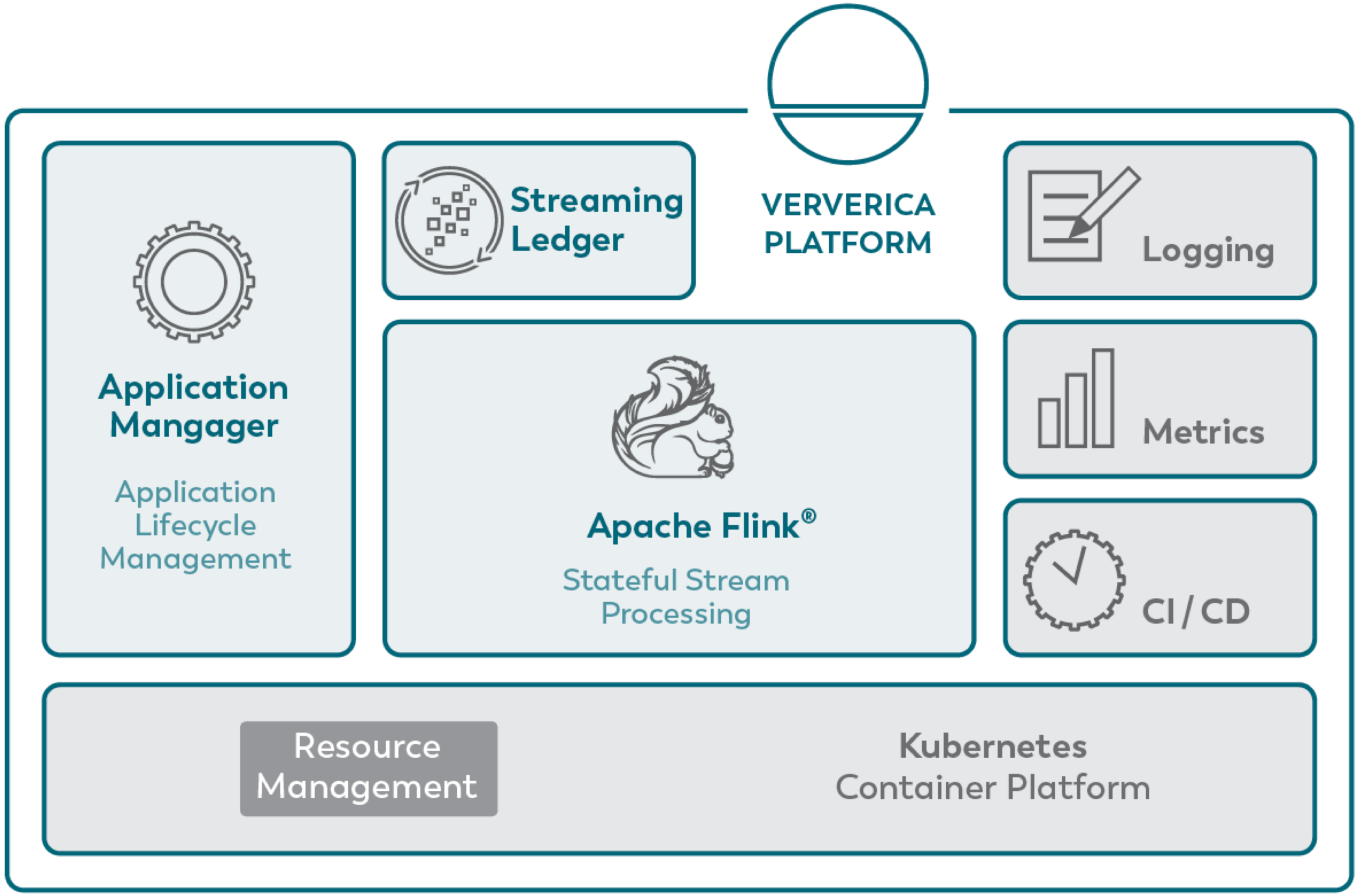
About Ververica

Original creators of Apache Flink®

Enterprise Stream Processing

© 2019 Ververica

Ververica Platform



© 2019 Ververica

Flink 101

Some Apache Flink Users

airbnb | AIRBUS DEFENCE & SPACE | Alibaba Group | aws | BetterCloud | bouygues TELECOM | Capital One

COMCAST | criteo | JOHN DEERE | DELL EMC | DiDi | ebay | ERICSSON

Goldman Sachs | HUAWEI | king | ING | lyft | MediaMath | NETFLIX | oppo | OVH.com

Pinterest | SK telecom | splunk> | stripe | Telefónica | Tencent 腾讯 | theTradeDesk

Uber | workday. | yelp

Sources: Powered by Flink, Speakers – Flink Forward San Francisco 2019, Speakers – Flink Forward Europe 2019

Stream Processing with Apache Flink at Alibaba

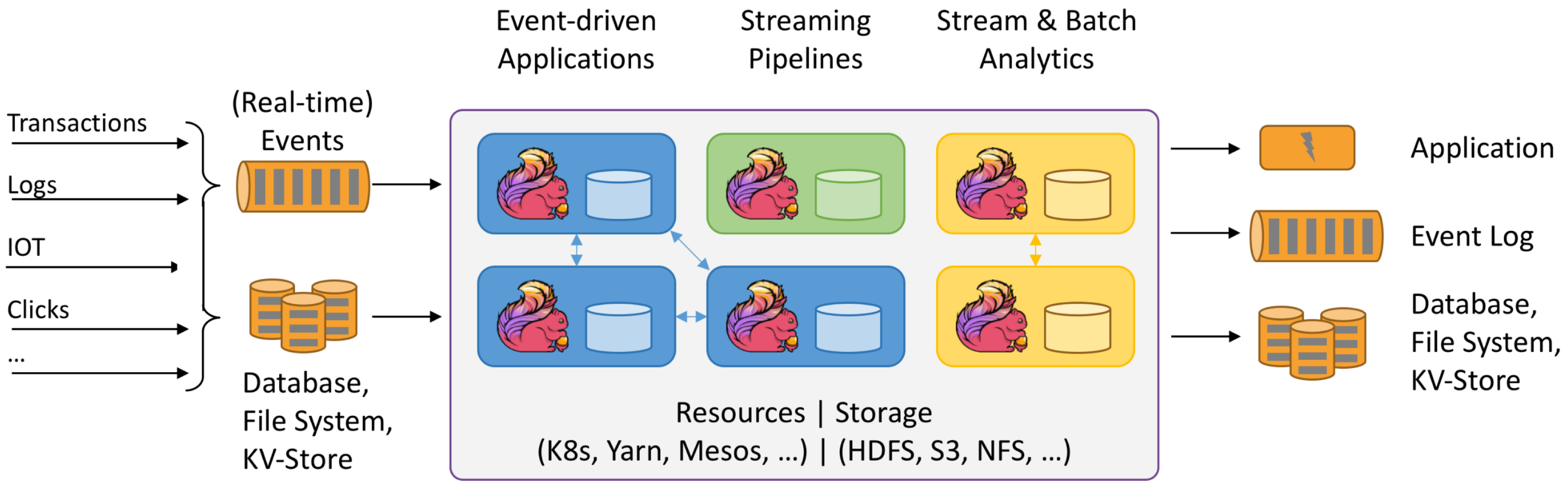
The "Singles Day“ (11/11)

machines	queries	throughput	latency	state size
10K	10K	1.7B events / sec	Sub-Second	100TB

© 2019 Ververica

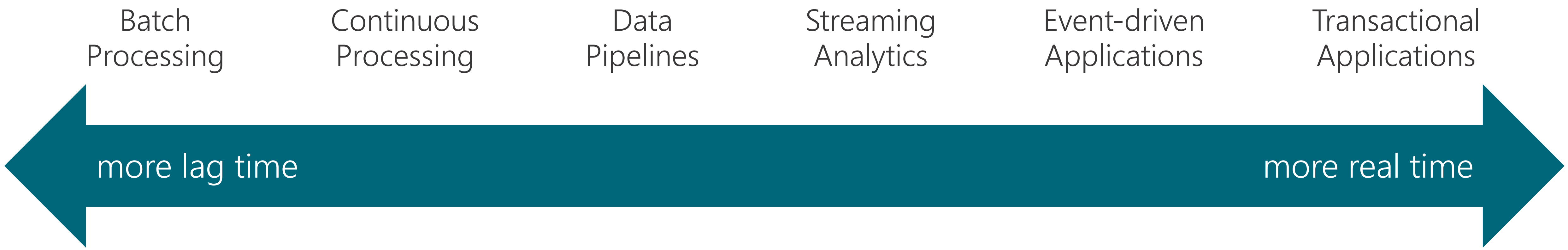
Apache Flink

Stateful Computations over Data Streams


Event-driven Applications
Streaming Pipelines
Stream & Batch Analytics
Transactions
Logs
IOT
Clicks
...
(Real-time) Events
Database, File System, KV-Store
Resources | Storage
(K8s, Yarn, Mesos, ...) | (HDFS, S3, NFS, ...)
Application
Event Log
Database, File System, KV-Store

© 2019 Ververica

The Flink Vision



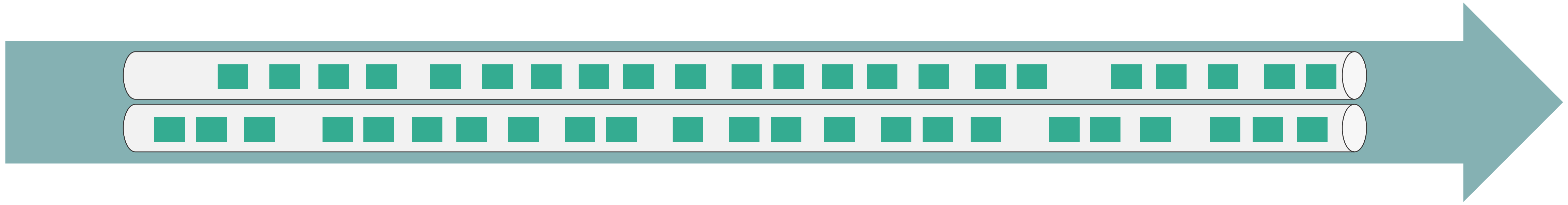
Unifying data processing based on thinking in data streams

And building the first open source system to cover that spectrum

© 2019 Ververica

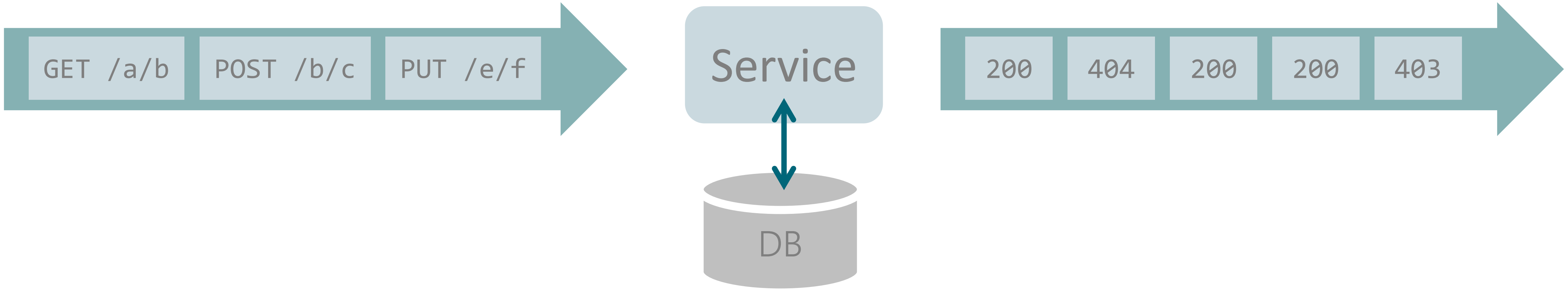
Everything is a Stream

Streams Of Records in a Log or MQ
[e.g., Apache Kafka or AWS Kinesis ...]



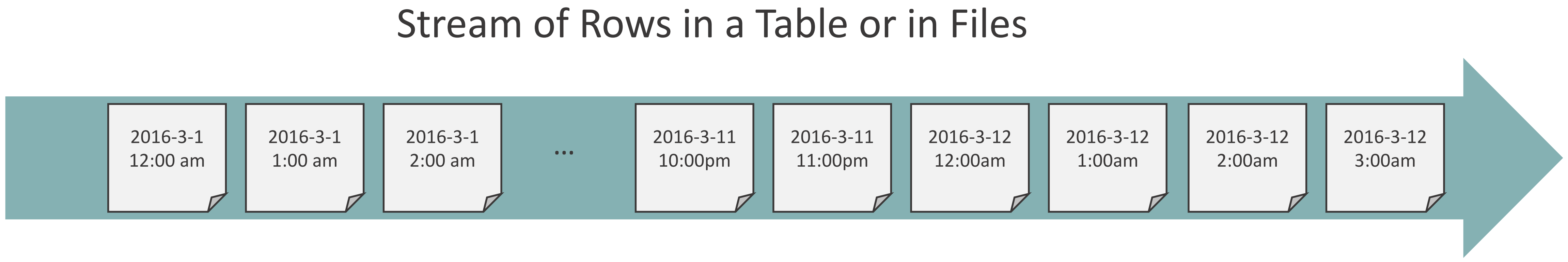
Everything is a Stream

Stream of Requests/Responses to/from Services



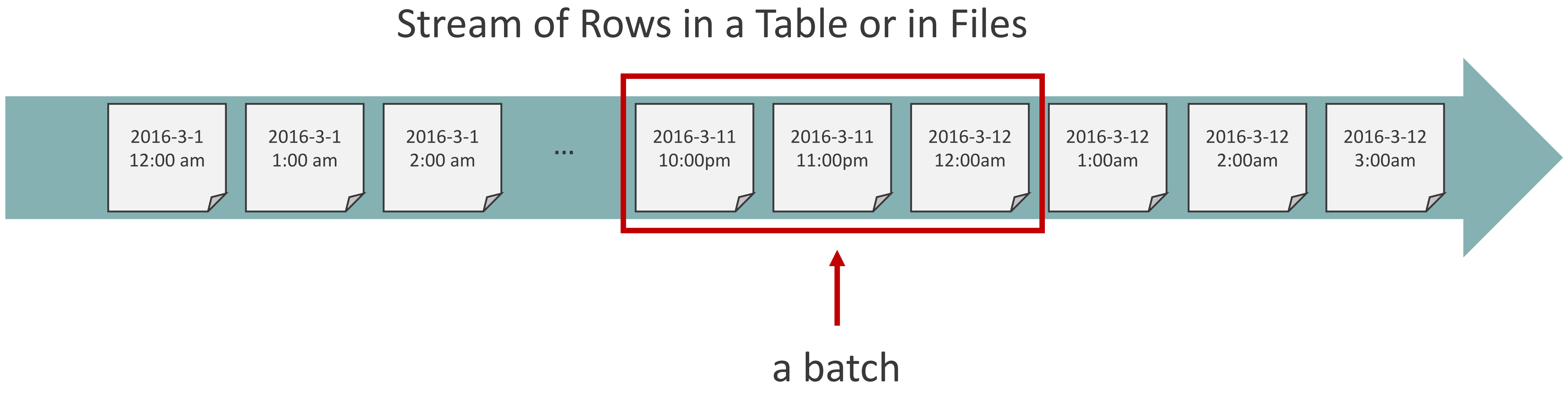
→ event sourcing architecture

Everything is a Stream



© 2019 Ververica

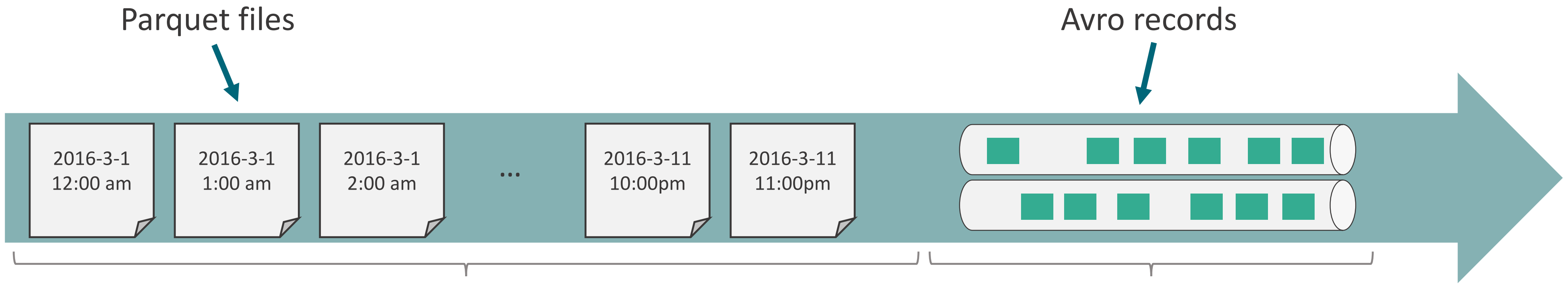
Everything is a Stream



© 2019 Ververica

Everything is a Stream

Streams may span storage systems



more distant past

(e.g., compressed files in DFS/Object Store)

recent past

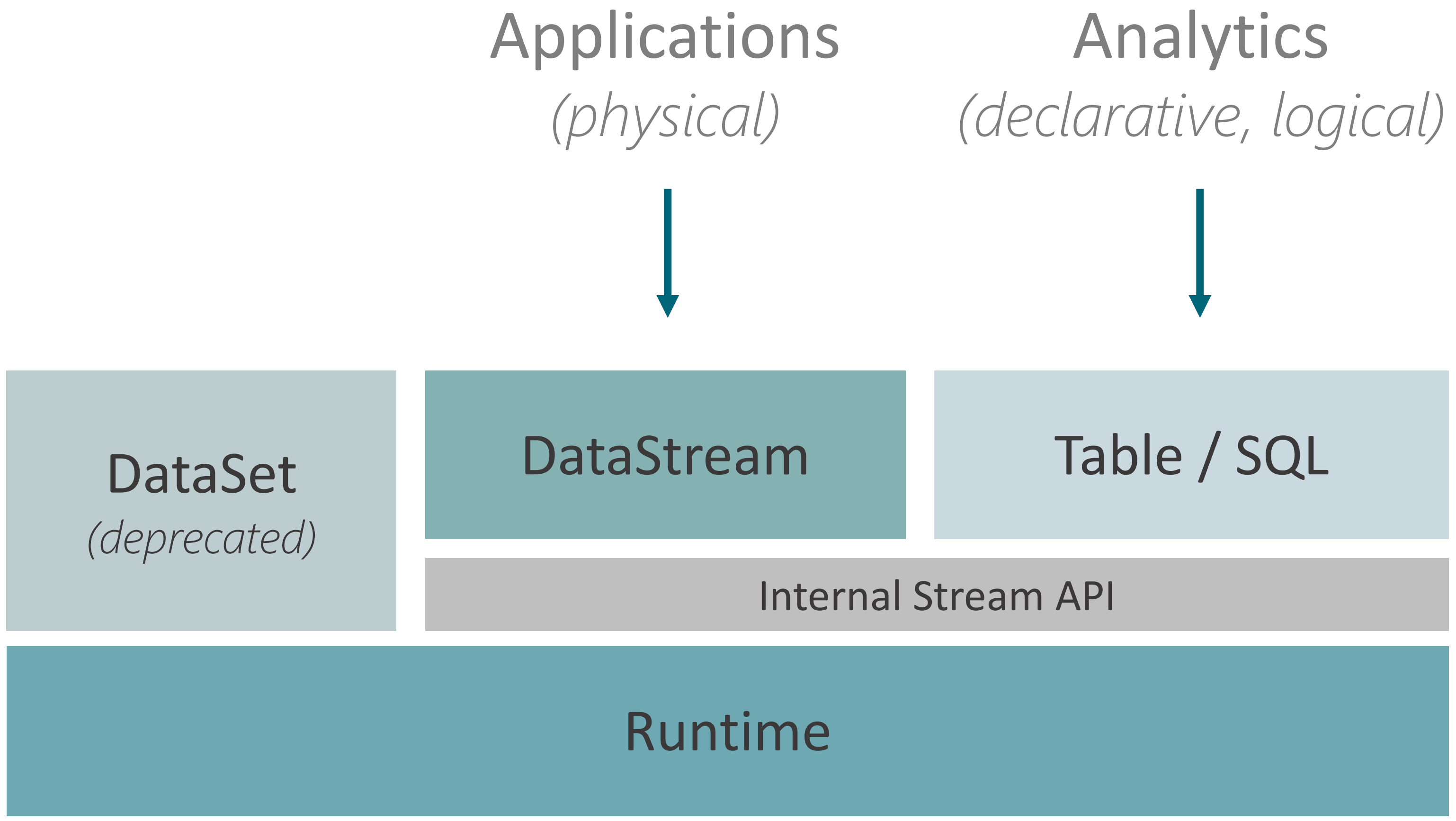
(e.g., events in MQ/Log)

Easy way to bootstrap an application with past data and then let it continue with real time data

© 2019 Ververica

APIs

APIs to Support these Use Cases



© 2019 Ververica

DataStream API (Functional Java / Scala)

```
val lines: DataStream[String] = env.addSource(new FlinkKafkaConsumer011(…))

val events: DataStream[Event] = lines.map((line) => parse(line))

val stats: DataStream[Statistic] = stream
    .keyBy("sensor")
    .timeWindow(Time.seconds(5))
    .sum(new MyAggregationFunction())

stats.addSink(new StreamingFileSink(path))
```

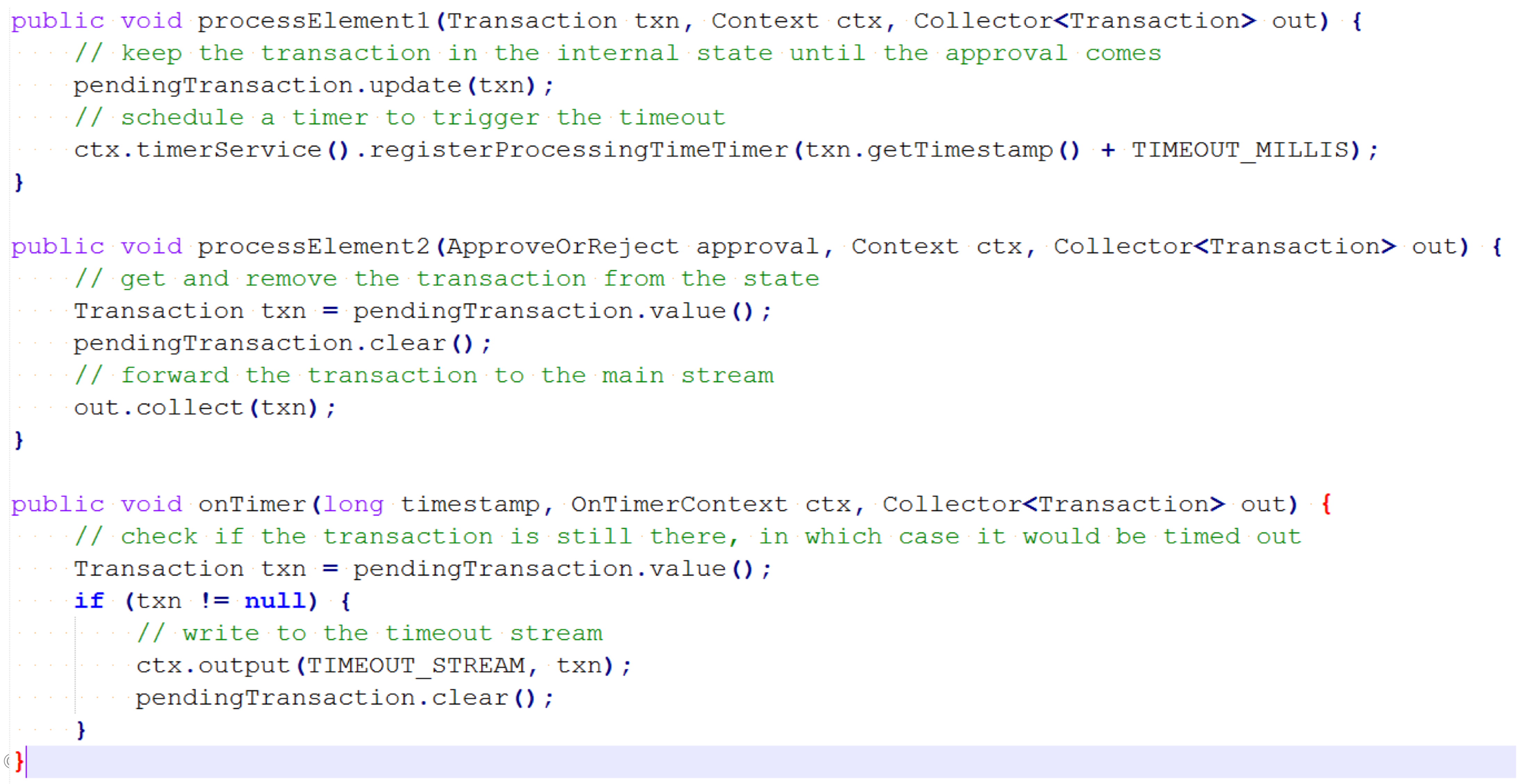
Source

Transformation

Windowed Transformation

Sink

DataStream API Process Functions

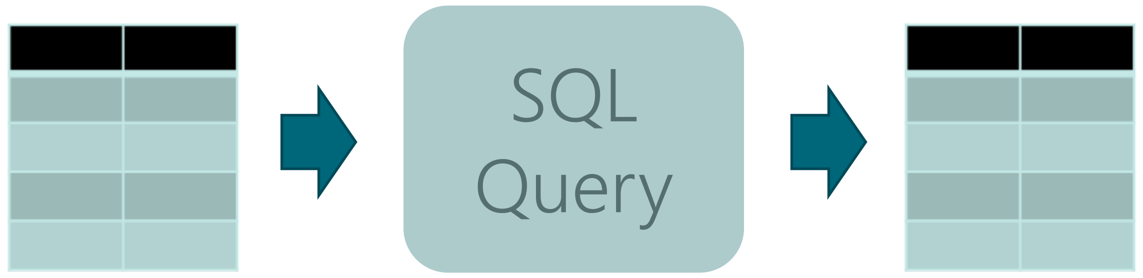


```java
public void processElement1(Transaction txn, Context ctx, Collector<Transaction> out) {
    // keep the transaction in the internal state until the approval comes
    pendingTransaction.update(txn);
    // schedule a timer to trigger the timeout
    ctx.timerService().registerProcessingTimeTimer(txn.getTimestamp() + TIMEOUT_MILLIS);
}

public void processElement2(ApproveOrReject approval, Context ctx, Collector<Transaction> out) {
    // get and remove the transaction from the state
    Transaction txn = pendingTransaction.value();
    pendingTransaction.clear();
    // forward the transaction to the main stream
    out.collect(txn);
}

public void onTimer(long timestamp, OnTimerContext ctx, Collector<Transaction> out) {
    // check if the transaction is still there, in which case it would be timed out
    Transaction txn = pendingTransaction.value();
    if (txn != null) {
        // write to the timeout stream
        ctx.output(TIMEOUT_STREAM, txn);
        pendingTransaction.clear();
    }
}
```

SQL / Table API – Batch style (fix data set as input)



Batch Query Execution

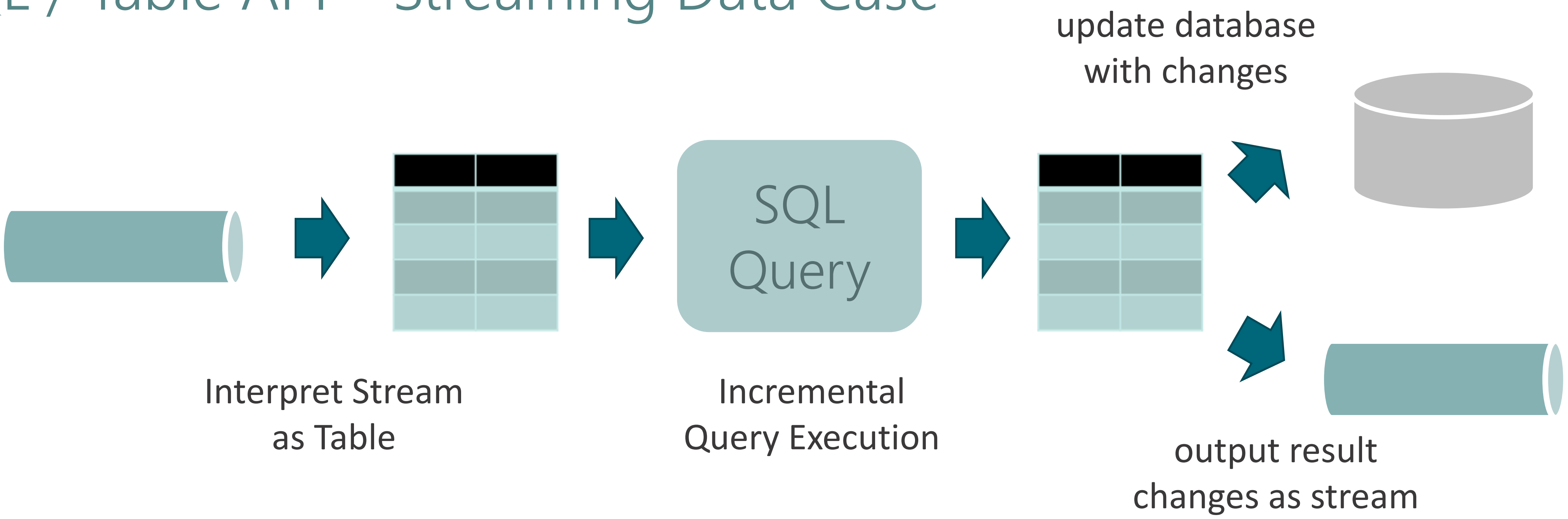
Full TPC-H support in Flink 1.9 with Blink query engine

Full TPC-DS support targeted for Flink 1.10

```
SELECT
    room,
    TUMBLE_END(rowtime, INTERVAL '1' HOUR),
    AVG(temperature)
FROM
    sensors
GROUP BY
    TUMBLE(rowtime, INTERVAL '1' HOUR), room
```

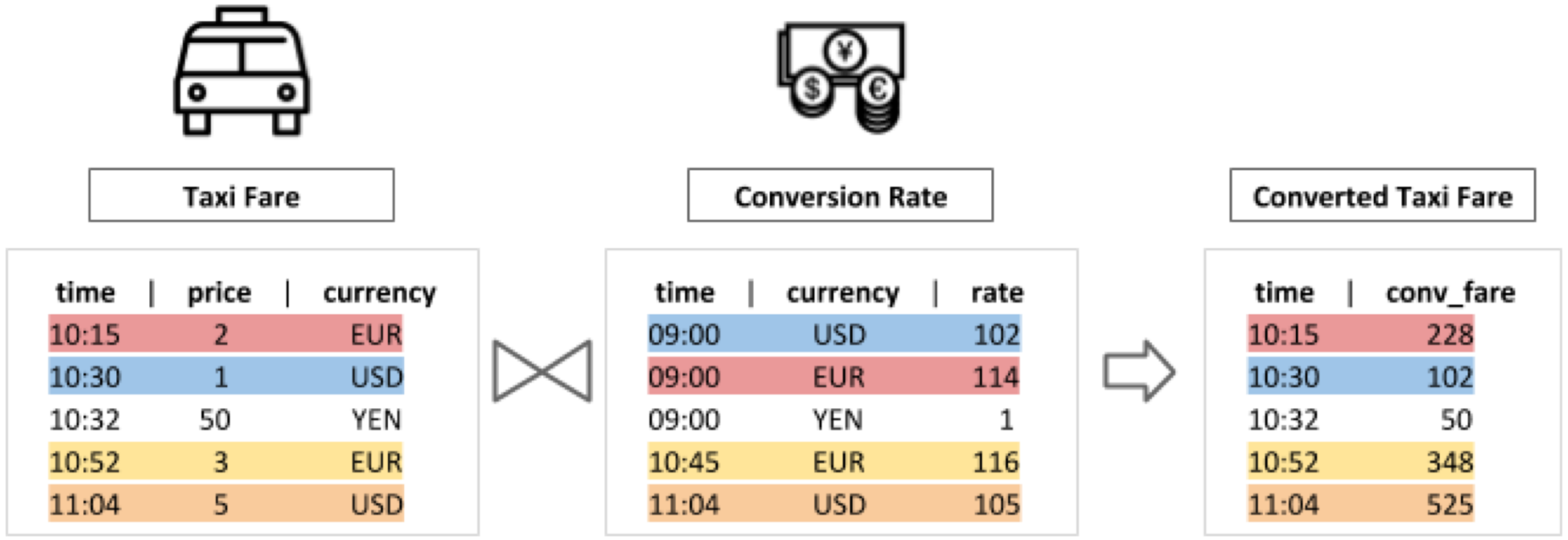
© 2019 Ververica

SQL / Table API – Streaming Data Case



```
SELECT
    room,
    TUMBLE_END(rowtime, INTERVAL '1' HOUR),
    AVG(temperature)
FROM
    sensors
GROUP BY
    TUMBLE(rowtime, INTERVAL '1' HOUR), room
```

SQL / Table API – Temporal Joins Example



Taxi Fare

time	price	currency
10:15	2	EUR
10:30	1	USD
10:32	50	YEN
10:52	3	EUR
11:04	5	USD

Conversion Rate

time	currency	rate
09:00	USD	102
09:00	EUR	114
09:00	YEN	1
10:45	EUR	116
11:04	USD	105

Converted Taxi Fare

time	conv_fare
10:15	228
10:30	102
10:32	50
10:52	348
11:04	525

```
SELECT tf.time
       tf.price * rh.rate as conv_fare

FROM taxiFare AS tf

LATERAL TABLE (Rates(tf.time)) AS rh

WHERE tf.currency = rh.currency;
```

© 2019 Ververica

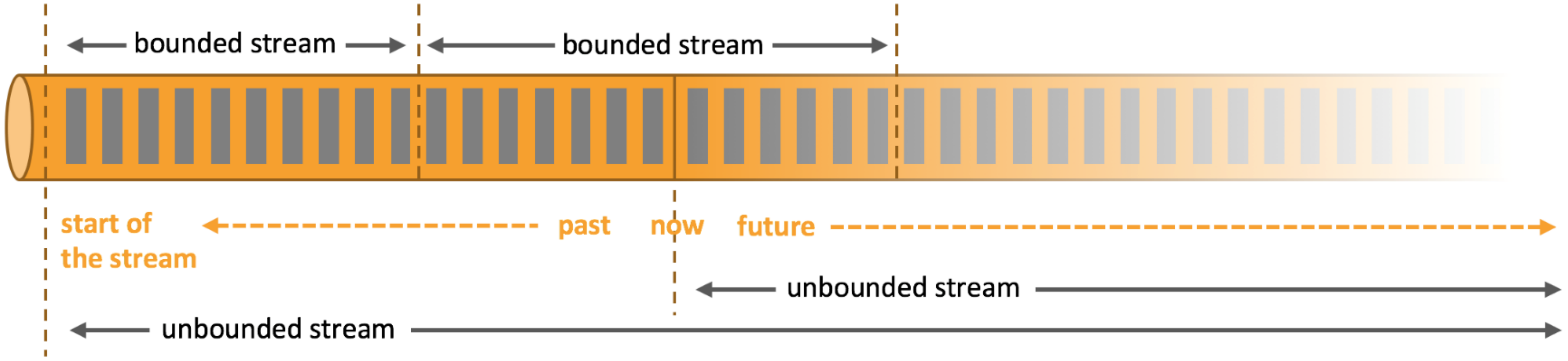
SQL / Table API – Event Pattern Matching Example

```
SELECT rideId, timeDiff(startT, endT) / 60000 AS durationMin
FROM Rides
MATCH_RECOGNIZE (
  PARTITION BY rideId
  ORDER BY rideTime
  MEASURES
    S.rideTime AS startT,
    E.rideTime AS endT
  AFTER MATCH SKIP PAST LAST ROW
  PATTERN (S E)
  DEFINE
    S AS S.isStart,
    E AS NOT E.isStart
);
```

The Relationship between Batch and Streaming

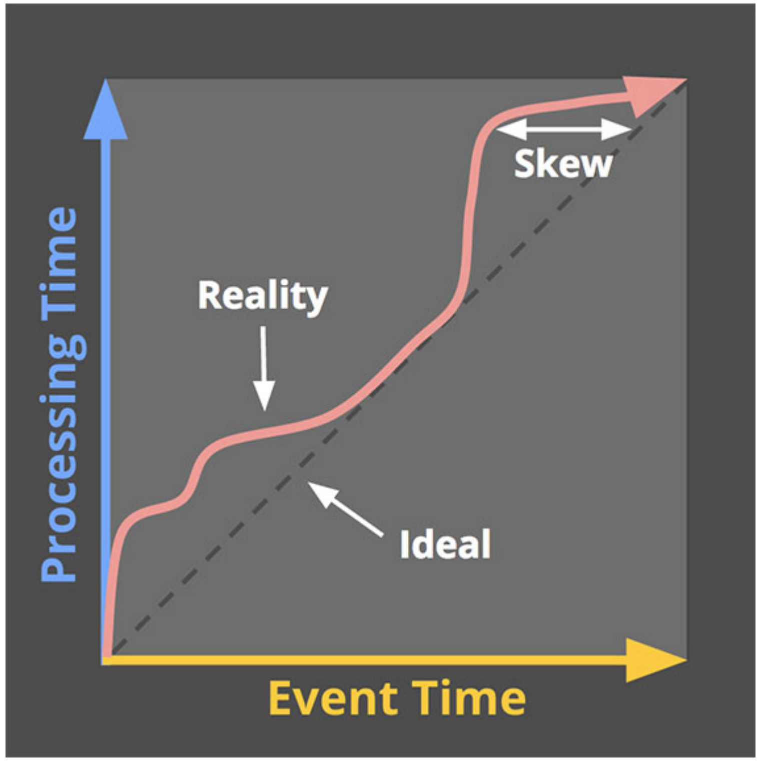
Batch Processing is a special case of Stream Processing

A batch is just a bounded stream.



That is about 60% of the truth…

The remaining 40% of the truth



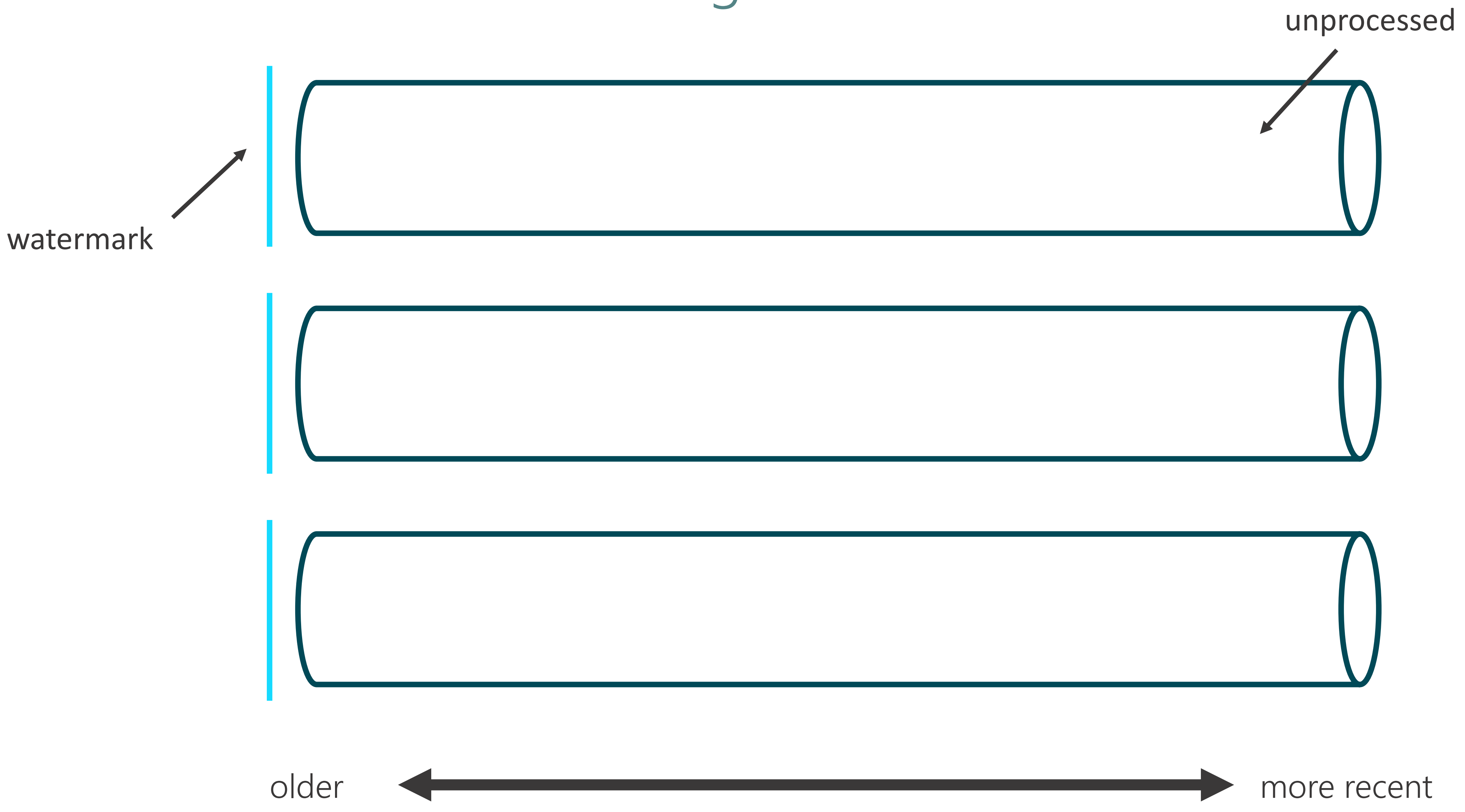
… never seen this in Batch Processing, though.

The (Event-time) Watermark

The remaining 40% of the truth

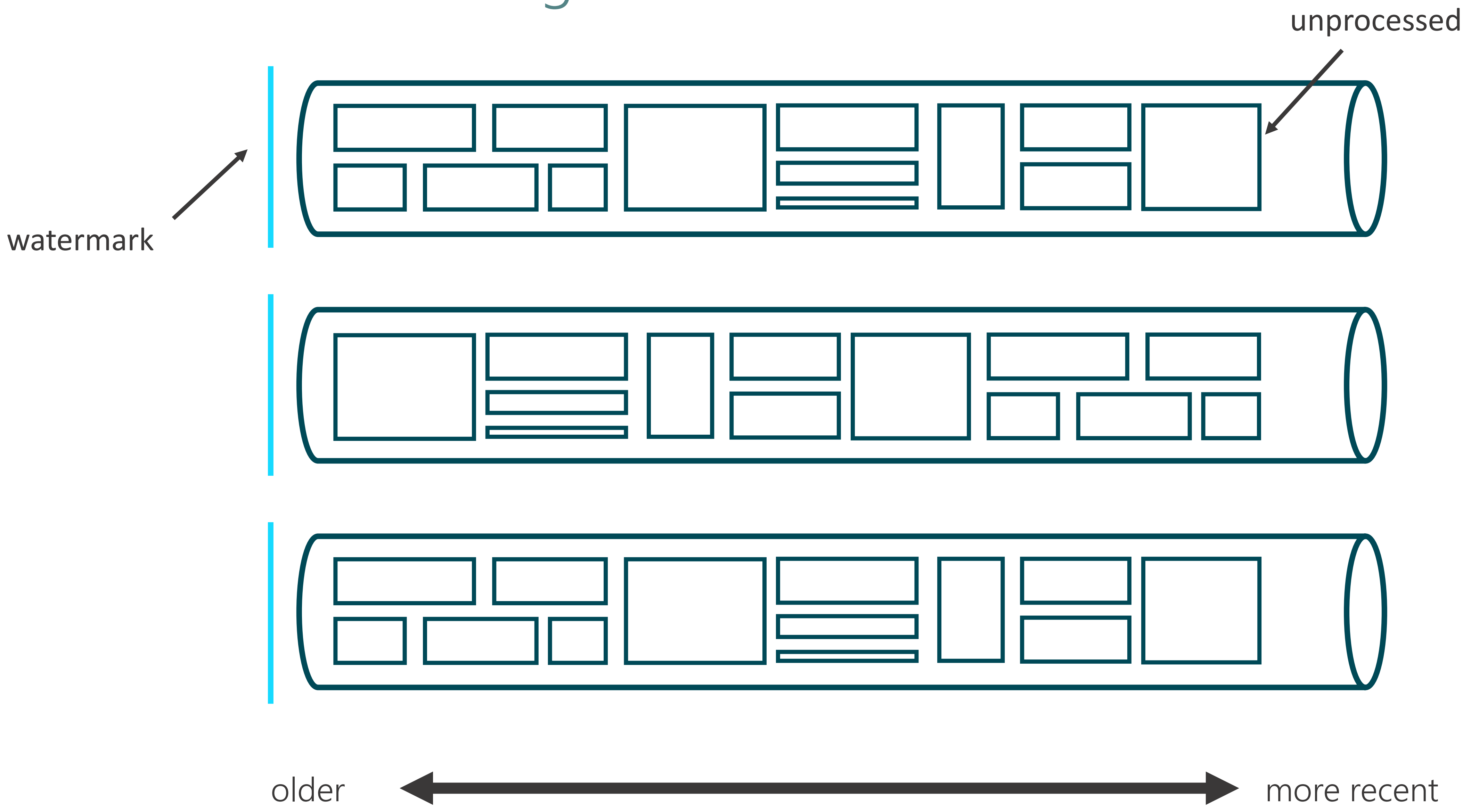
Continuous Streaming	Batch Processing
Data is incomplete	Data is as complete as it gets within the job
Latency SLAs	
Completeness and Latency is a tradeoff	No Low Latency SLAs

Stream Real-time Processing



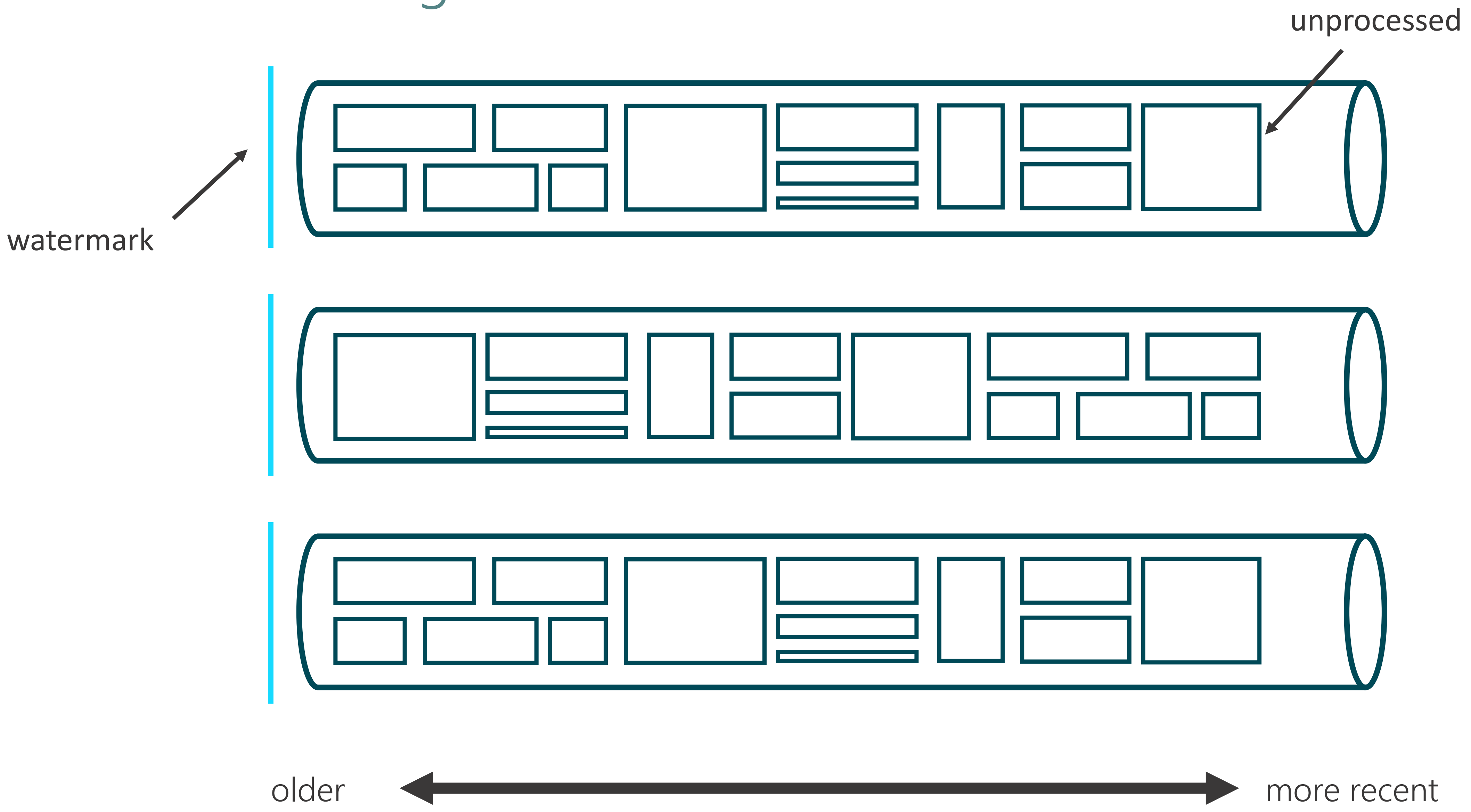
© 2019 Ververica

Stream Re-Processing



© 2019 Ververica

Batch Processing



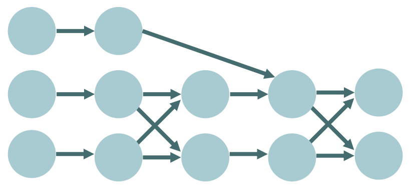
Batch vs. Stream Processing

Continuous Streaming	Batch Processing
Watermarks to model Completeness/Latency tradeoff	No Watermarks
Incremental results & Proc.-Time Timers	Results at end-of-program only
In-receive-order ingestion with low parallelism	Massively parallel out-of-order ingestion

Fast Batch Runtime in a Stream Processor

Evolution through the entire stack!

Exploiting the Batch Special Case



activates additional
optimizer choices

Planner/Optimizer

```
if (bounded && non-incremental)
```

Continuous Operators

Streaming
Scheduler Rules

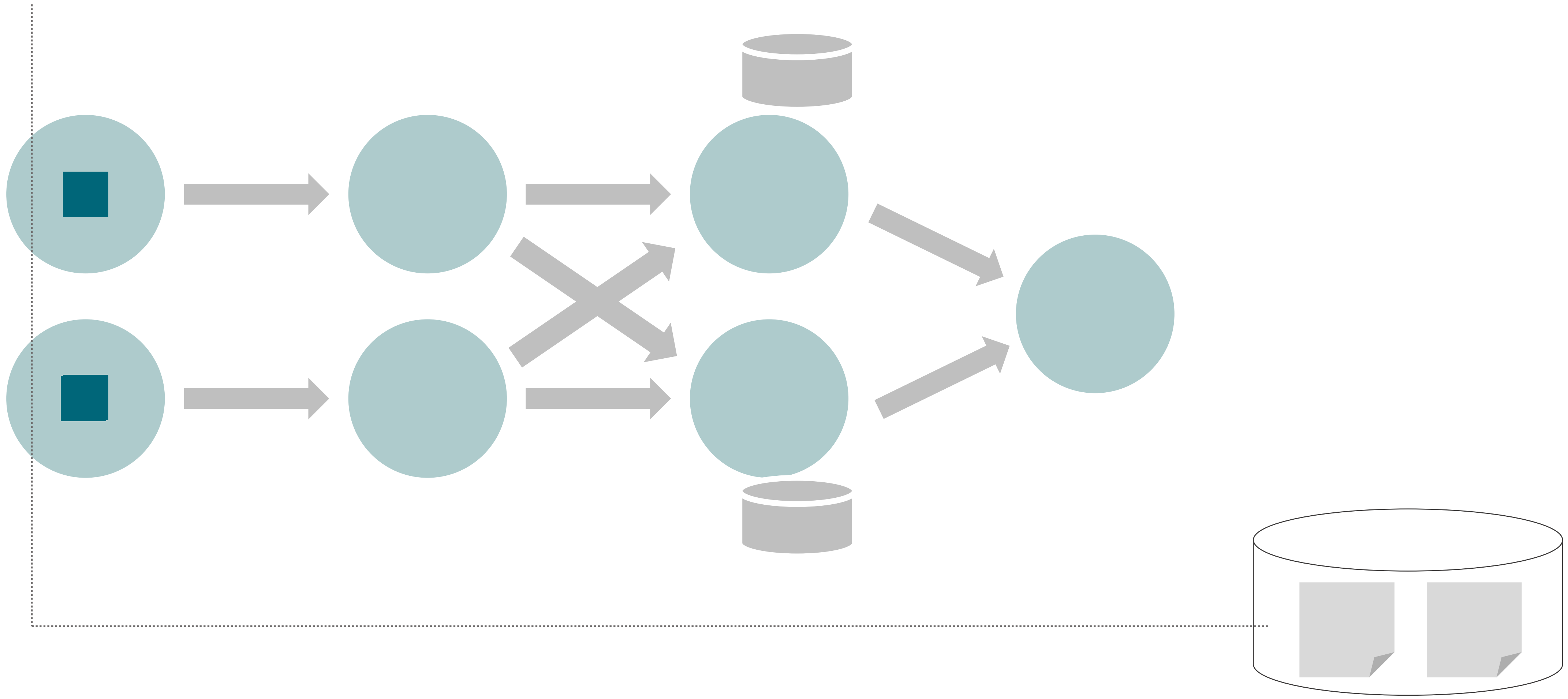
Core operators,
cover all cases

Additional Bounded
Operators

Additional
Scheduling Strategies

Optimized operators
for subset of cases

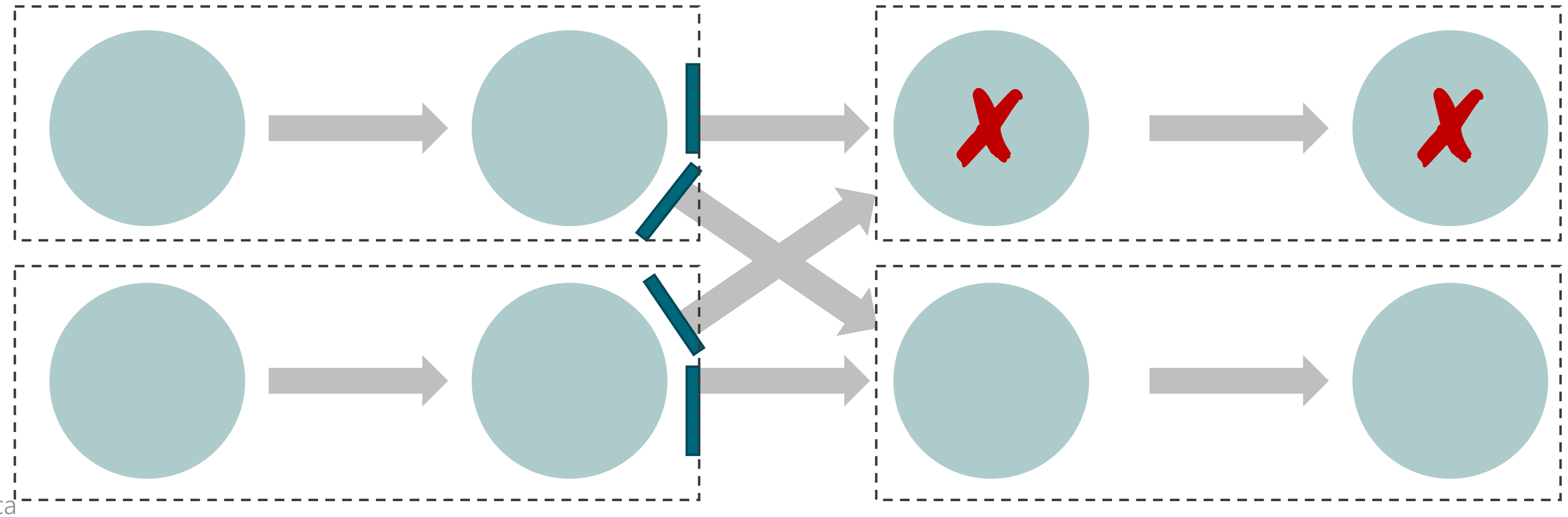
Fault tolerance without writing intermediate streams to Brokers or DFS



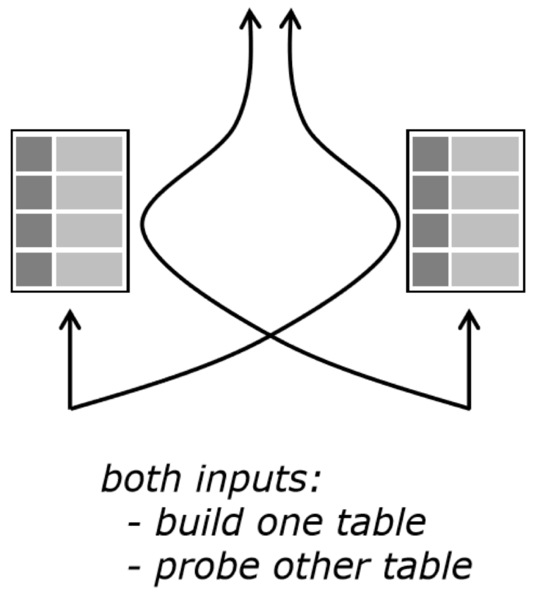
© 2019 Ververica

Scheduling Strategies

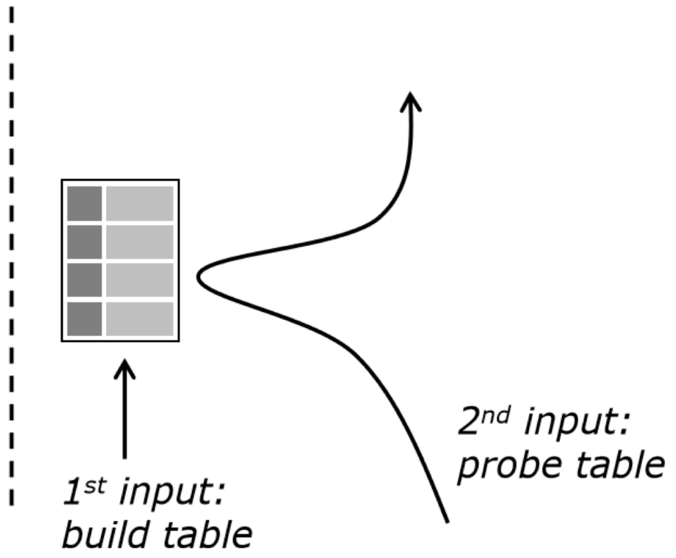
- Build pipelined regions
  - Incremental results: everything pipelines
  - Non-incremental results: break pipelines once in a while
- Recovery: Restart the pipelined region from latest checkpoint (or beginning)
  - replay input since checkpoint or beginning



Streaming versus Batch Join



Continuous Streaming Join



Batch Hash Join

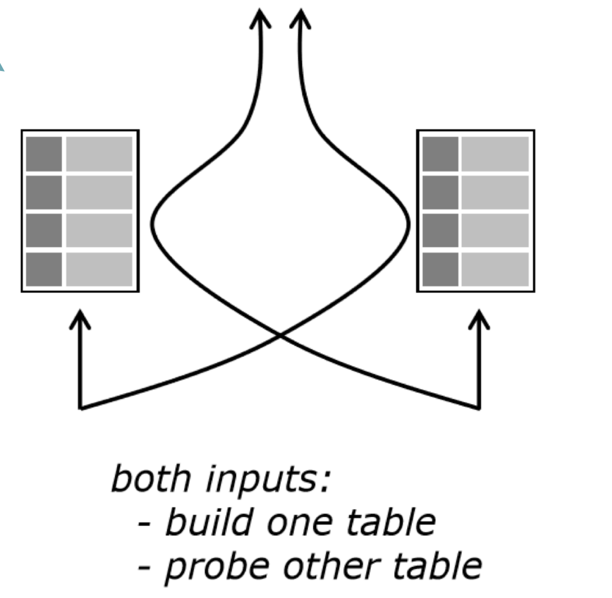
© 2019 Ververica

Streaming versus Batch Join

2x RocksDB LSM-Trees

bounded/ unbounded

incremental results



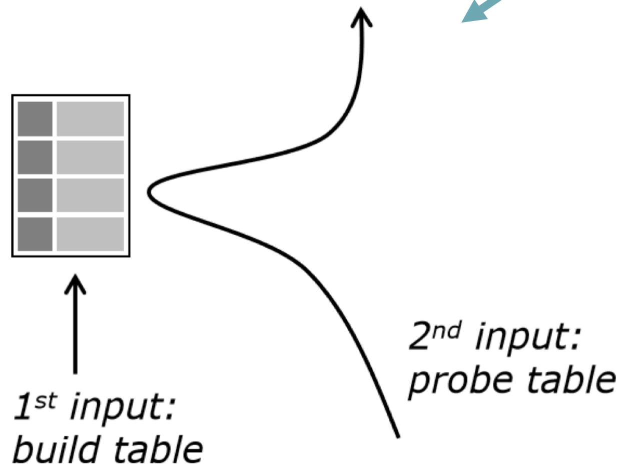
Continuous Streaming Join

1x Hybrid Hash Join

only on bounded data

batch results

no checkpoints



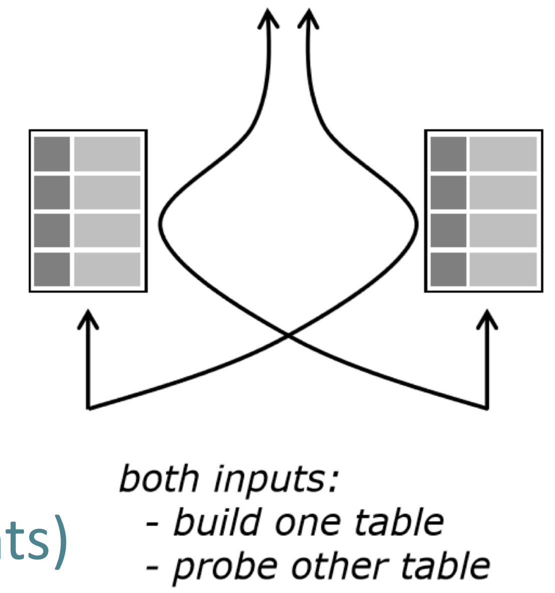
Batch Hash Join

more general

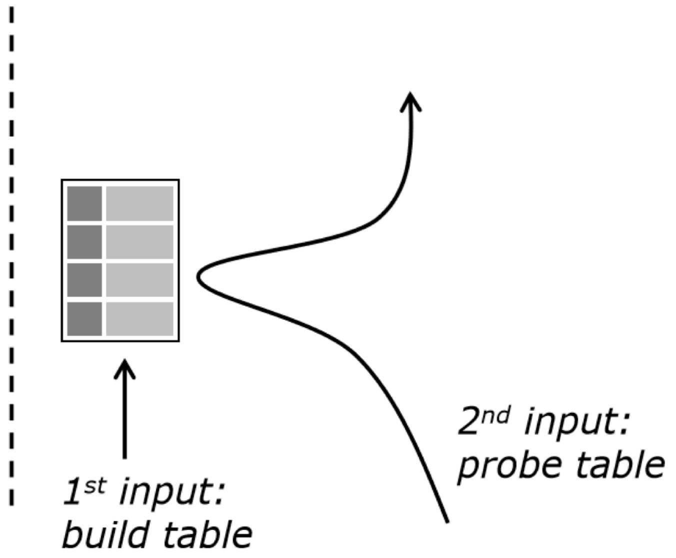
order-of-magnitude faster

Streaming versus Batch Join

push-based
(latency/checkpoints)



Continuous Streaming Join



Batch Hash Join

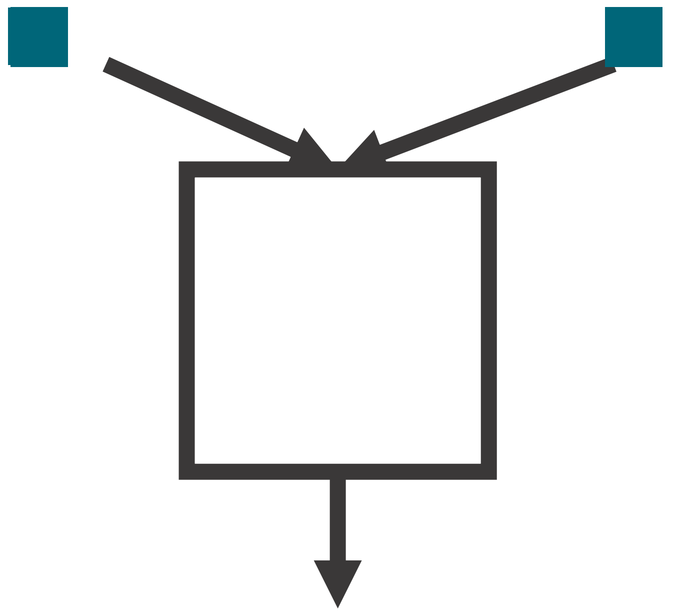
pull-based
(data flow control)

more general

order-of-magnitude faster

Push-based and Pull-based Operators

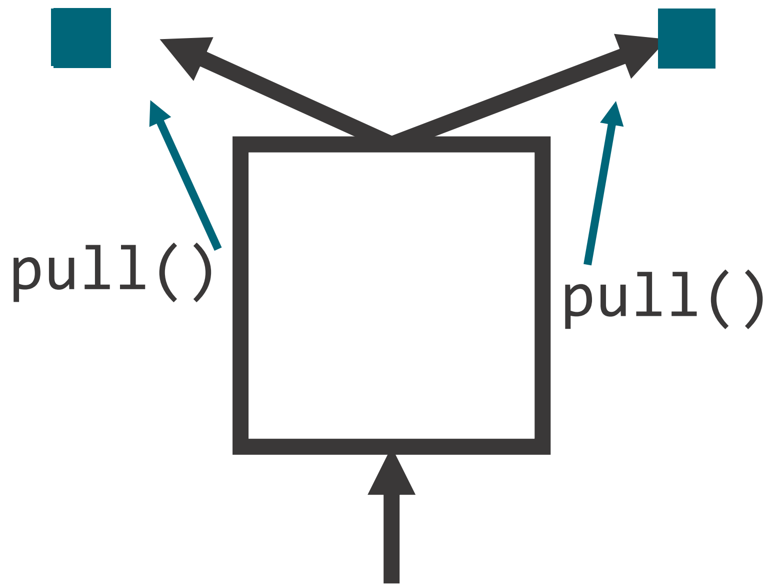
Push Operators



accept data from any input immediately
(like actor messages)

minimize latency
supports checkpoint alignment

Pull Operators

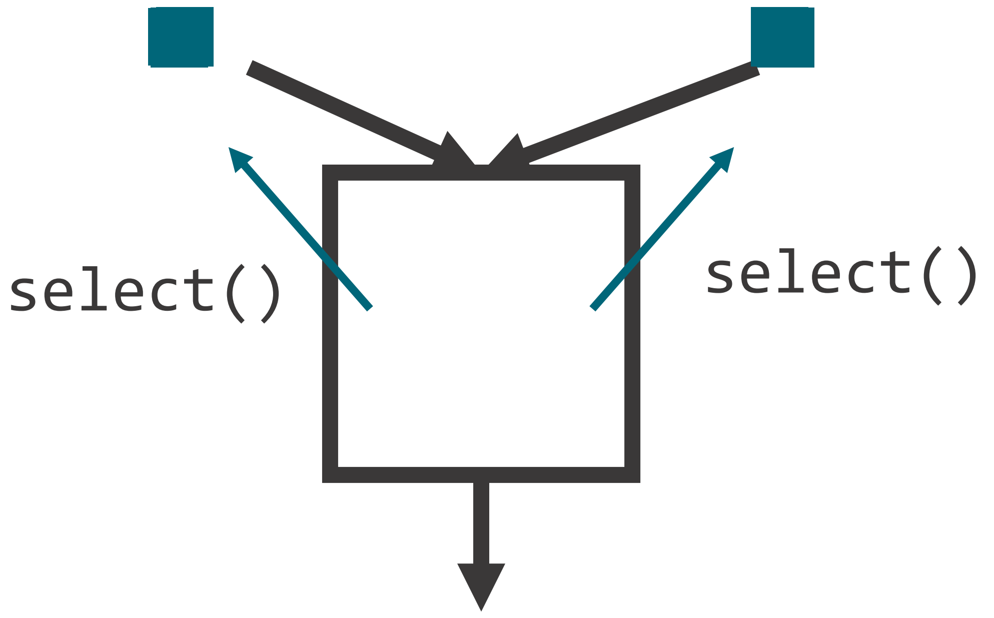


pull data from one input at a time
(like reading streams)

control over data flow,
high-latency, breaks checkpoints

© 2019 Ververica

Flink 1.9 - Selectable Push-based Operators



similar to non-blocking-I/O model

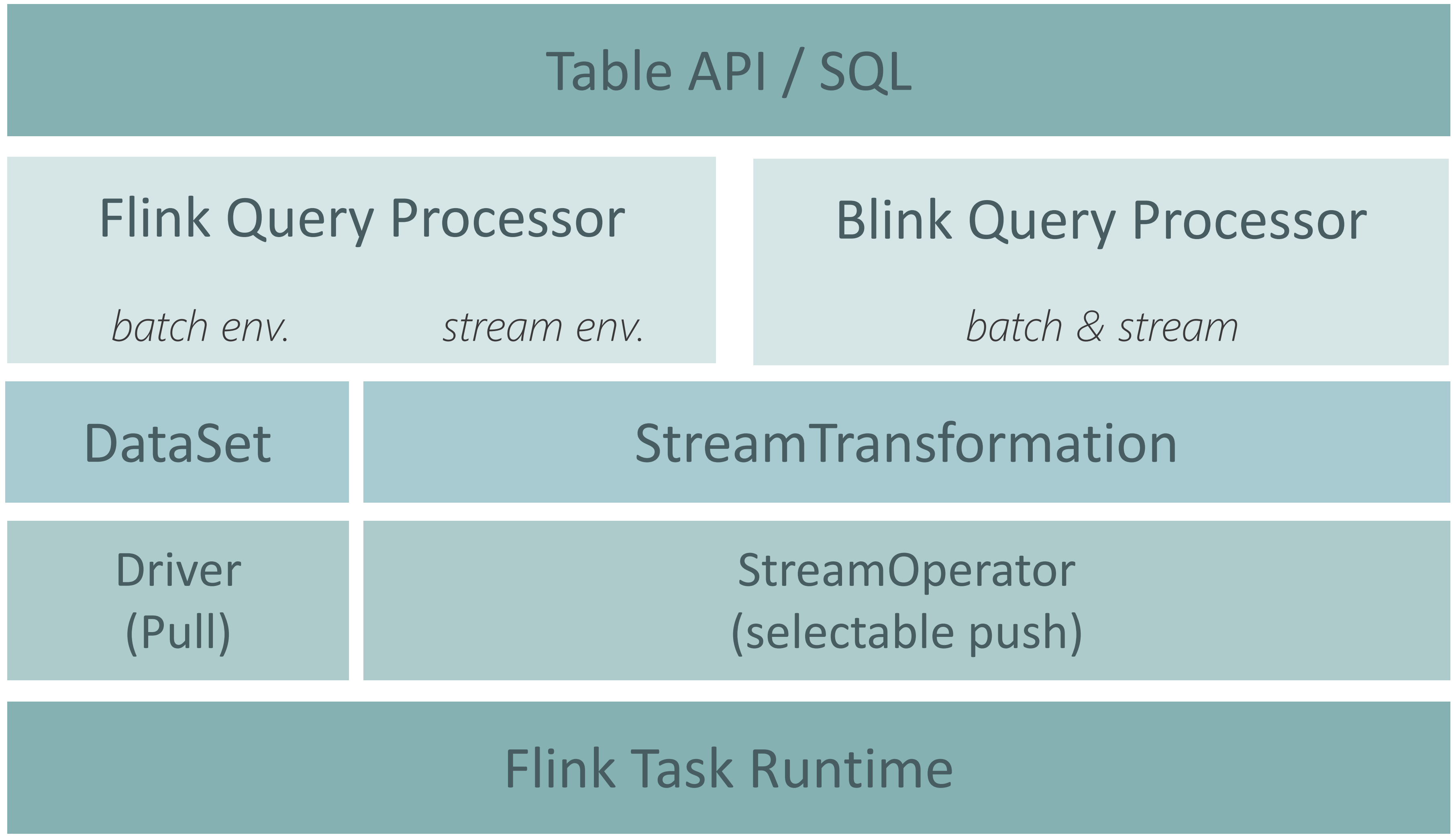
Java NIO, Linux Epoll, or Select

subscribe to inputs (select)
and receive pushed events

➔ Operators control data flow by selecting active data paths
➔ Among active data paths, fully asynchronous data flow driven by network, data sources (and timers)

The State of the Batch & Streaming Interplay in Flink

Table API / SQL in Flink 1.9



DataStream API

- DataStream is already supporting Bounded and Unbounded Streams

- Not exploiting batch optimizations so far
  - Bounded batch-style execution still faster on DataSet API

- After Flink 1.10:
  - Introduce `BoundedDataStream` and `non-incremental mode` to exploit optimizations for bounded data
  - Watermarks "jump" from **-∞** to **+∞** at end of program
  - Processing time timers deactivated or deferred (end of key)
  - ´Add same operators back batch-style SQL execution also for DataStream.

© 2019 Ververica

Wrapping Up

What else is new in Flink 1.9

Python Table API

Hive support

Analytics over Checkpoints/Savepoints

Preview of new Blink SQL Engine

Atomic stop-with-savepoint

...and lot's more

© 2019 Ververica

What else is the community working on?

Cross-Batch-Streaming Machine Learning

Python Table UDFs

Unaligned Checkpoints

Full support of Blink SQL Engine and TPC-DS coverage

DDL and Clients for Streaming SQL

New Source and Connector API

a big documentation overhaul

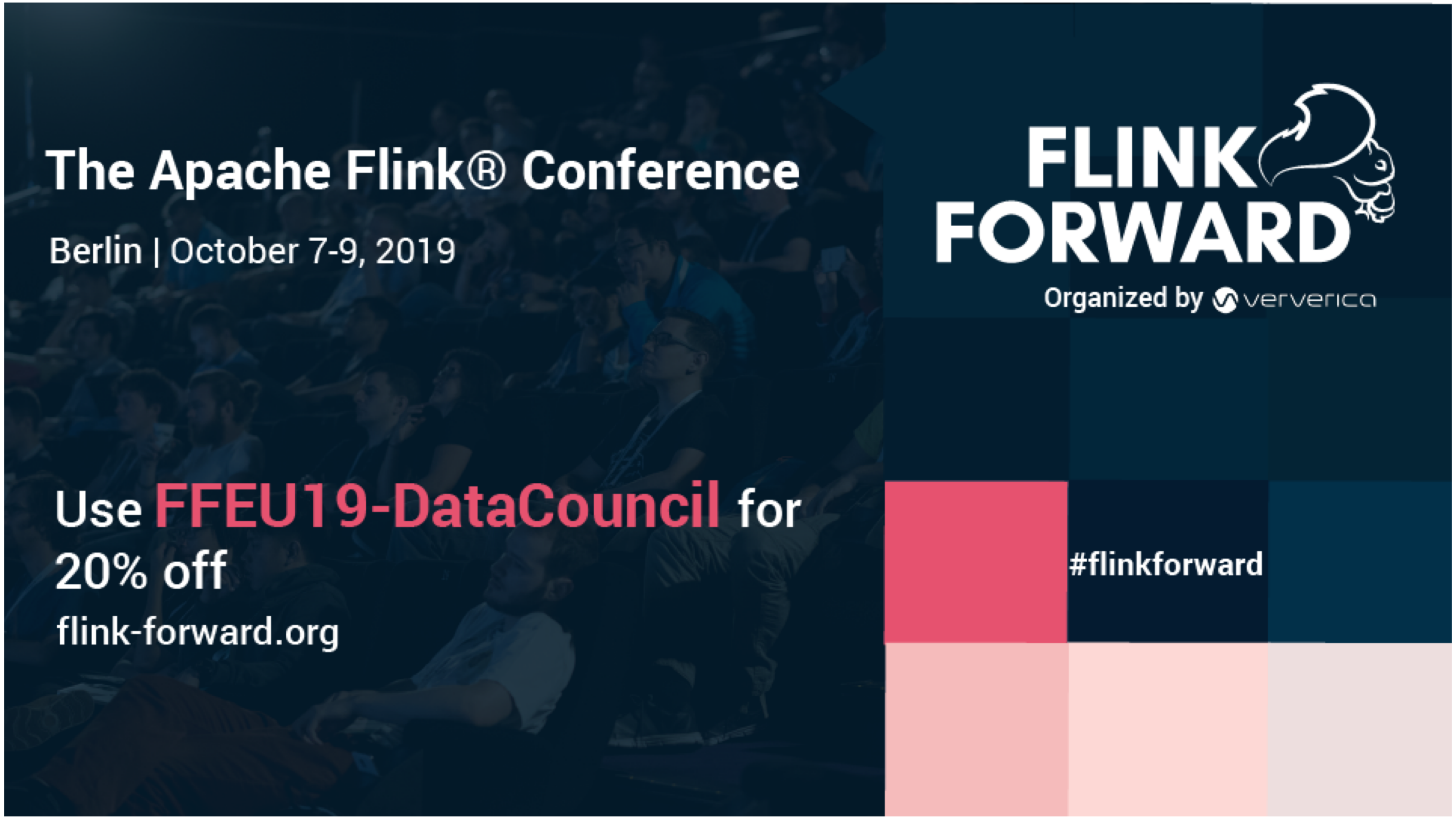
Interactive multi-job programs

...and lot's more

Learn more about Flink

Learn from the original creators of Apache Flink®
Developer Training
Barcelona, ES - **Oct 21**
Operations Training
Barcelona, ES - **Oct 24**
Register
Organized by ververica

Or another conference?



The Apache Flink® Conference

Berlin | October 7-9, 2019

Use FFEU19-DataCouncil for 20% off

flink-forward.org

FLINK FORWARD

Organized by ververica

#flinkforward

Thank you!

If you liked this, engage with the Apache Flink® community

- Try Flink and help us improve it
- Contribute docs, code, tutorials
- Share your use cases and ideas
- Join a Flink Meetup
- Come to Flink Forward `(https://www.flink-forward.org/)`

@twalthr @ApacheFlink @VervericaData

`https://flink.apache.org/`