Towards Flink 2.0 –
Unifying the Batch and Streaming Stack

Stephan Ewen
Co-creator and PMC of Apache Flink
Ververica (formerly dataArtisans, now part of Alibaba Group)

Alternative Talk Titles

Batch is a special case of something

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

Why is there still DataSet and DataStream?
What's taking you folks so long?

This is talk is based on joint work with many members of the Apache Flink community

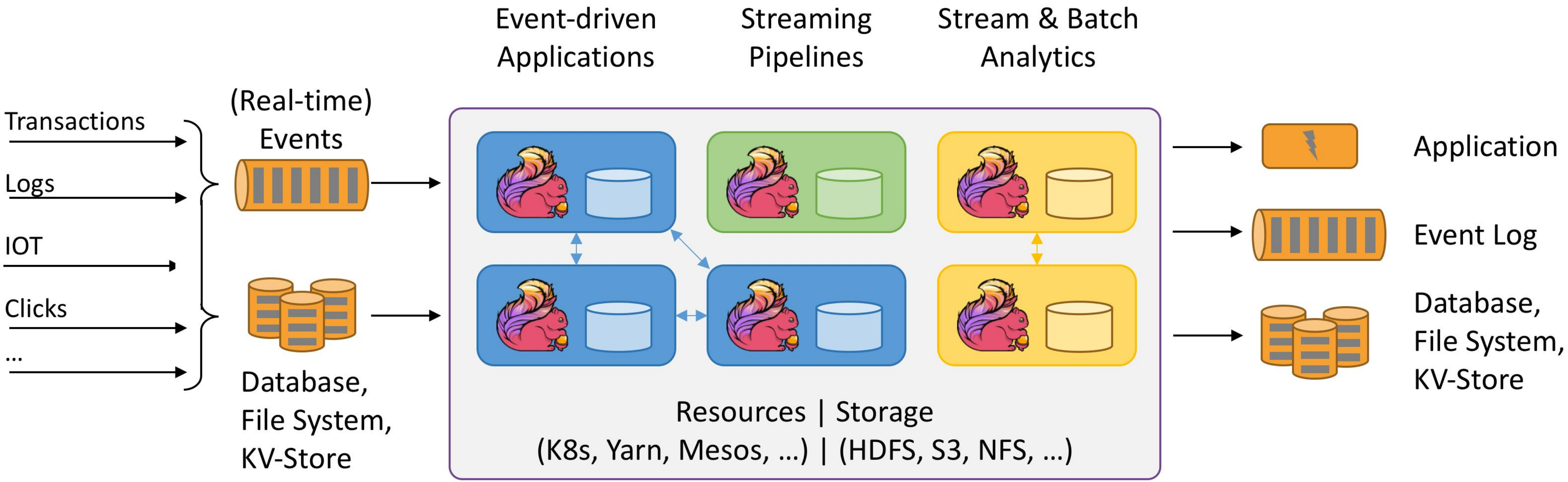
Xiaowei, Aljoscha, Timo, Dawid, Shaoxuan, Kurt, Guowei, Becket, Jincheng, Fabian, Till, Andrey, Gary, Chesnay, Piotr, Stefan, Zhijiang, Bowen, Haibo, etc.

And many others…

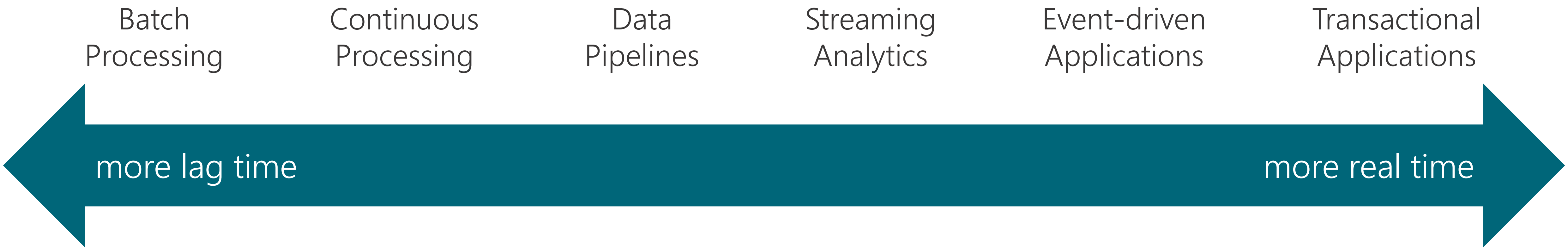
This is a snapshot of the state of design discussion and work-in-progress.
Some things may change as discussions evolve.

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

Computing over Data Streams



Batch Processing

Continuous Processing

Data Pipelines

Streaming Analytics

Event-driven Applications

Transactional Applications

Stream Processing based on Apache Flink at Alibaba

Performance during "Singles Day"

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

Some Apache Flink Users

airbnb
Alibaba Group
aws
BetterCloud
bouygues TELECOM
Capital One

COMCAST
criteo
JOHN DEERE
ebay
ERICSSON
HUAWEI
king

ING
lyft
MediaMath
NETFLIX
OVH.com
splunk>
Telefonica

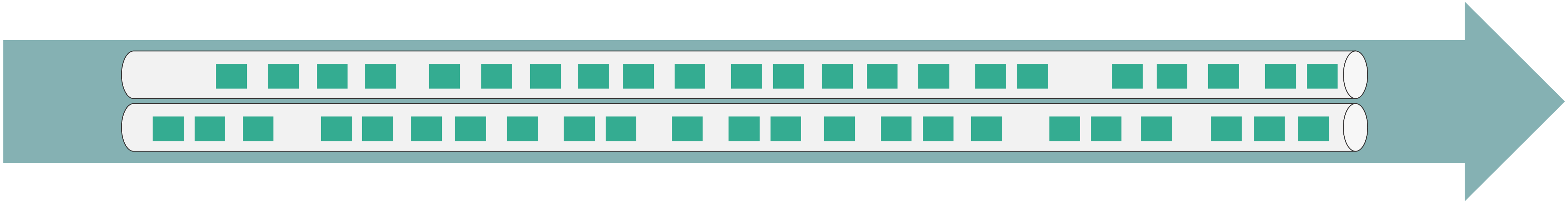
Tencent 腾讯
theTradeDesk
Uber
yelp
zalando

Source: https://flink.apache.org/poweredby.html and https://sf-2019.flink-forward.org/speakers

The Relationship between Batch and Streaming

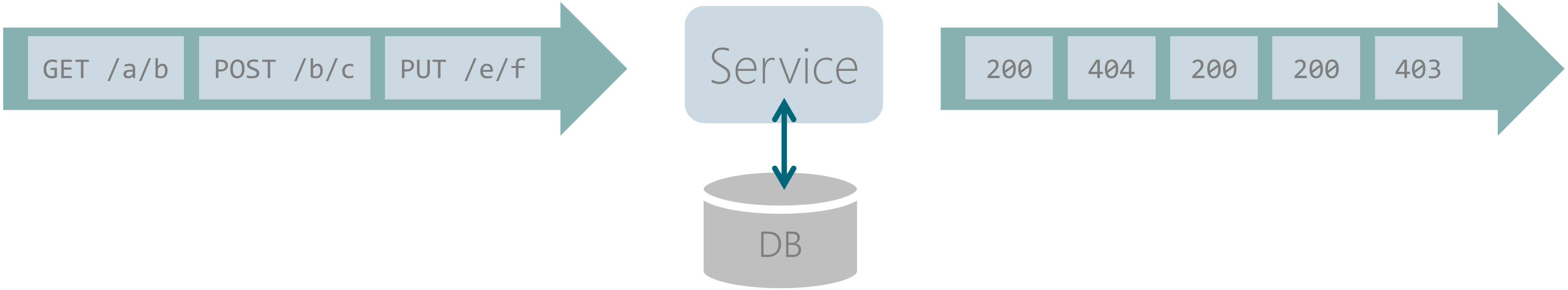
Everything is a Stream

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



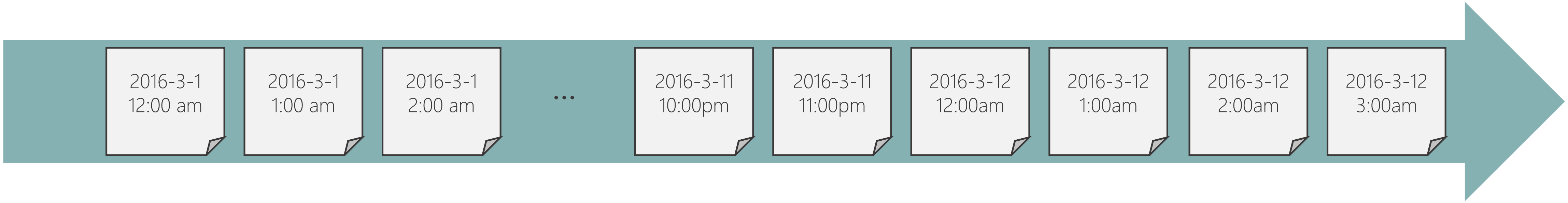
Everything is a Stream

Stream of Requests/Responses to/from Services

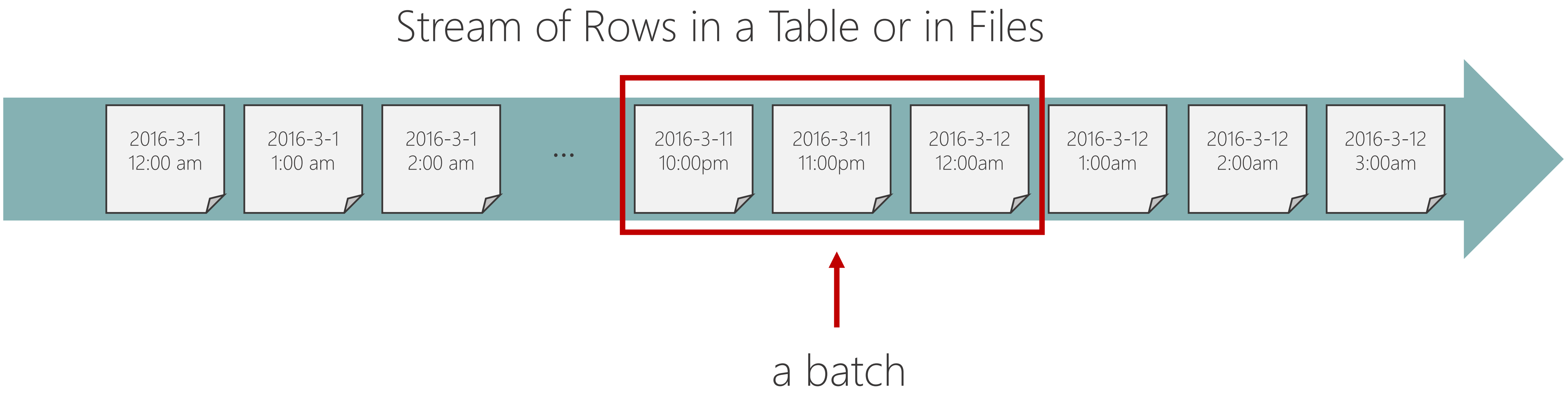


→ event sourcing architecture

Everything is a Stream


Stream of Rows in a Table or in Files

A batch is a Bounded Stream

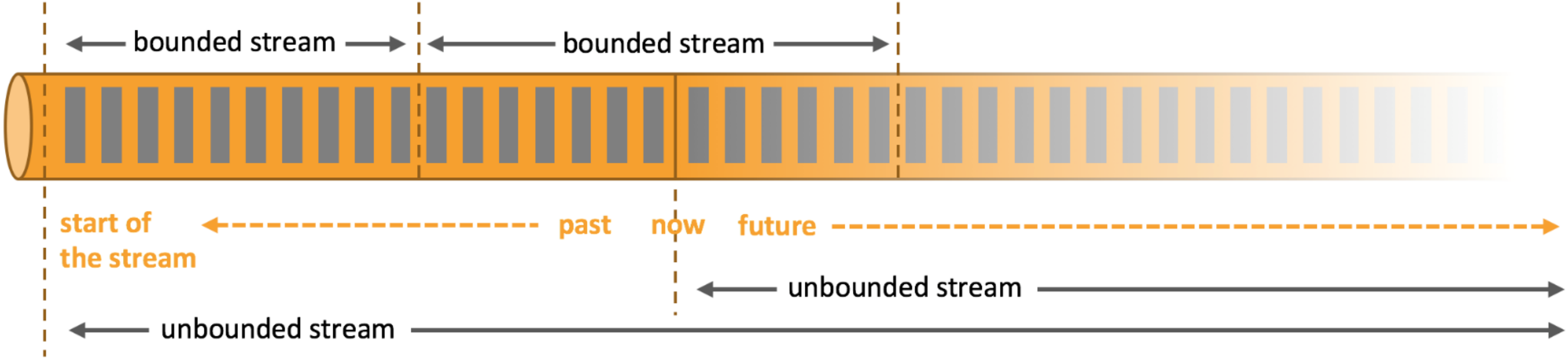


Stream of Rows in a Table or in Files

a batch

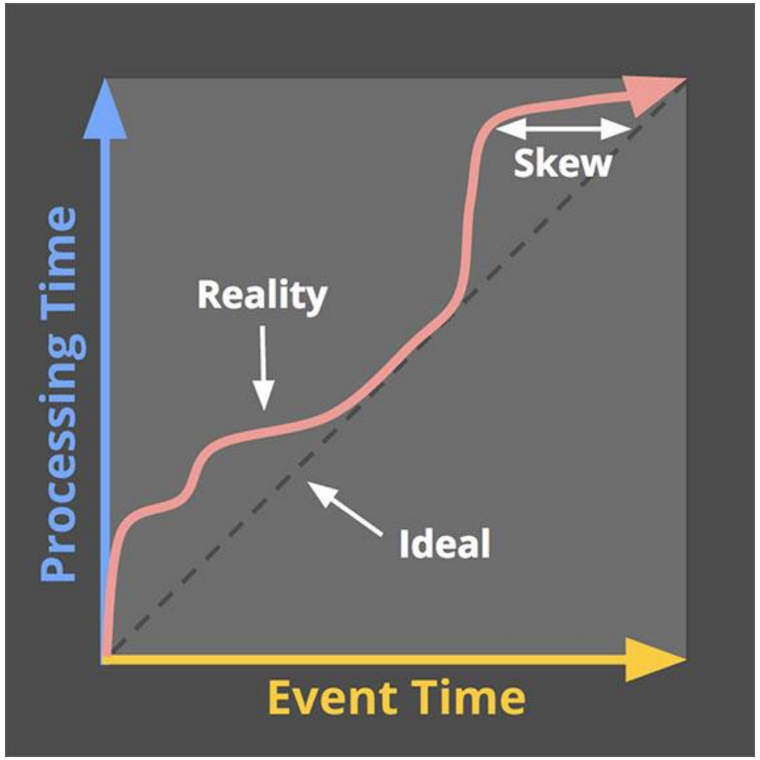
Batch Processing is a special case of Stream Processing

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 Low) Watermark

The remaining 40% of the truth

Continuous Streaming

Data is incomplete

Latency SLAs

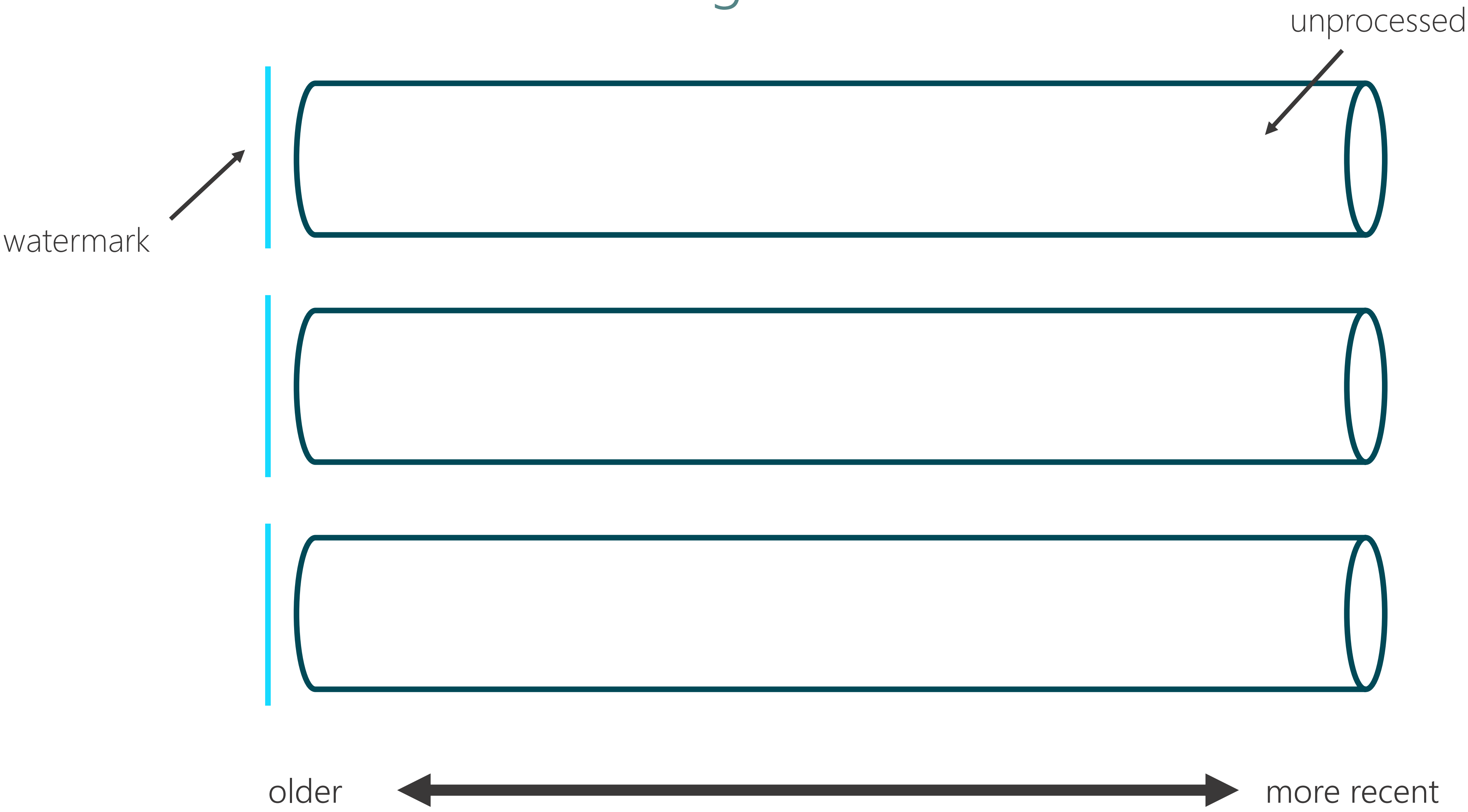
Completeness and Latency is a tradeoff

Batch Processing

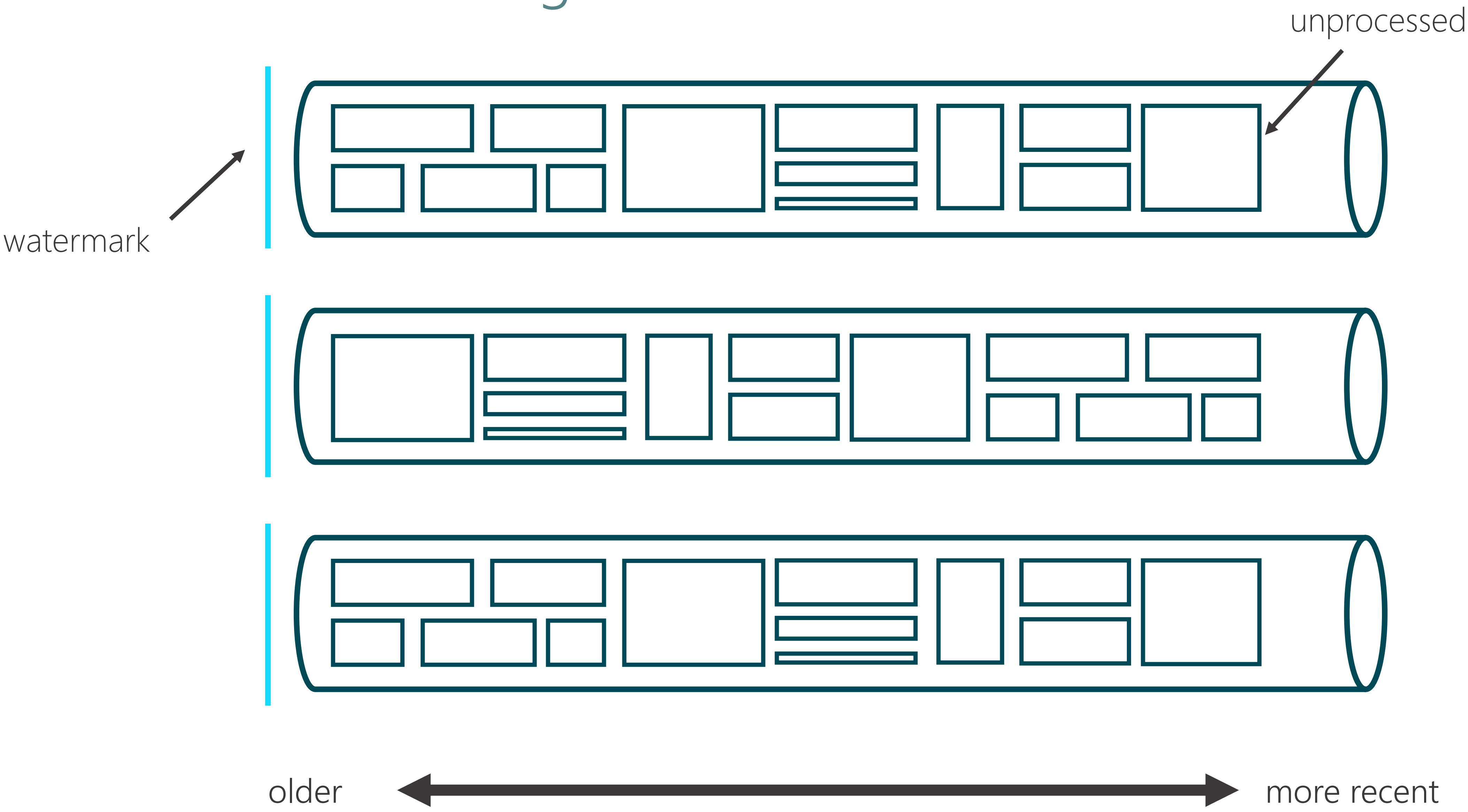
Data is as complete as it gets within the job

No Low Latency SLAs

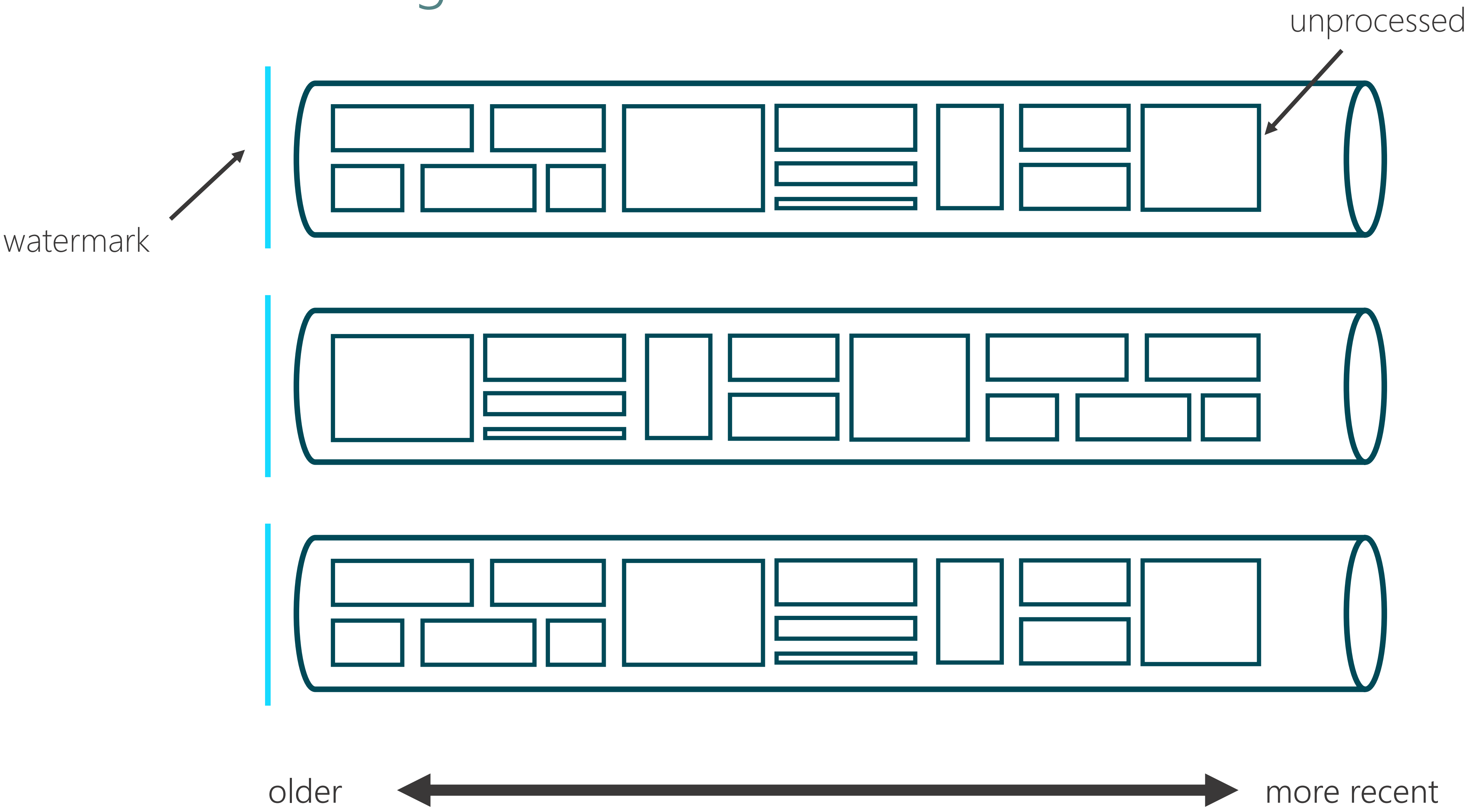
Stream Real-time Processing



Stream Re-Processing



Batch Processing



Batch vs. Stream Processing

Continuous Streaming

Watermarks to model Completeness/Latency tradeoff

Incremental results & Proc.-Time Timers

In-receive-order ingestion with low parallelism

Batch Processing

No Watermarks

Results at end-of-program only

Massively parallel out-of-order ingestion

The remainder of this talk

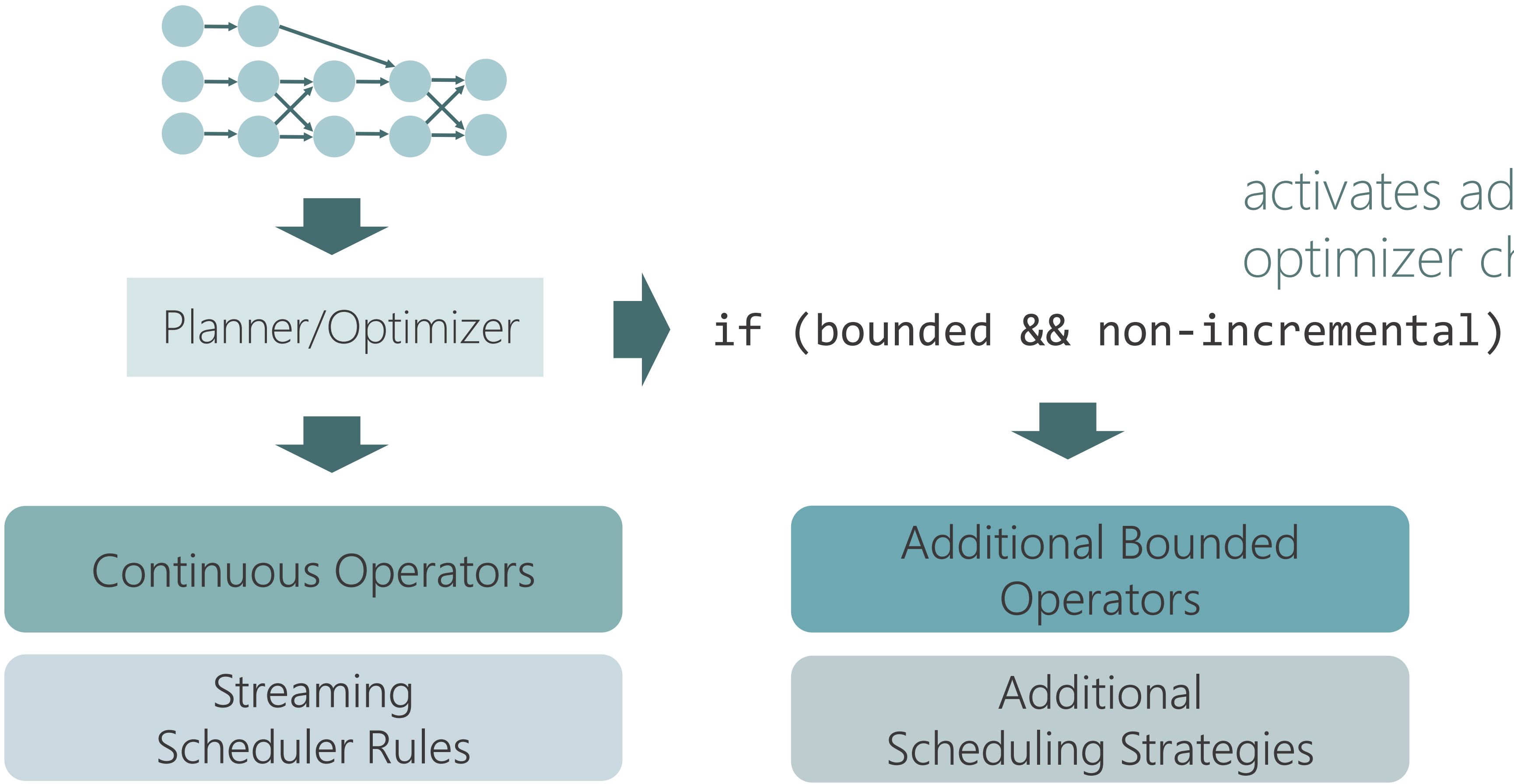
What does that mean for

(1) A unified Batch/Streaming Data Processing Runtime

(2) Unified Batch- and Streaming APIs

Stream- and Batch-Processing in the Runtime

Exploiting the Batch Special Case



Planner/Optimizer

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

activates additional optimizer choices

Continuous Operators

Streaming Scheduler Rules

Additional Bounded Operators

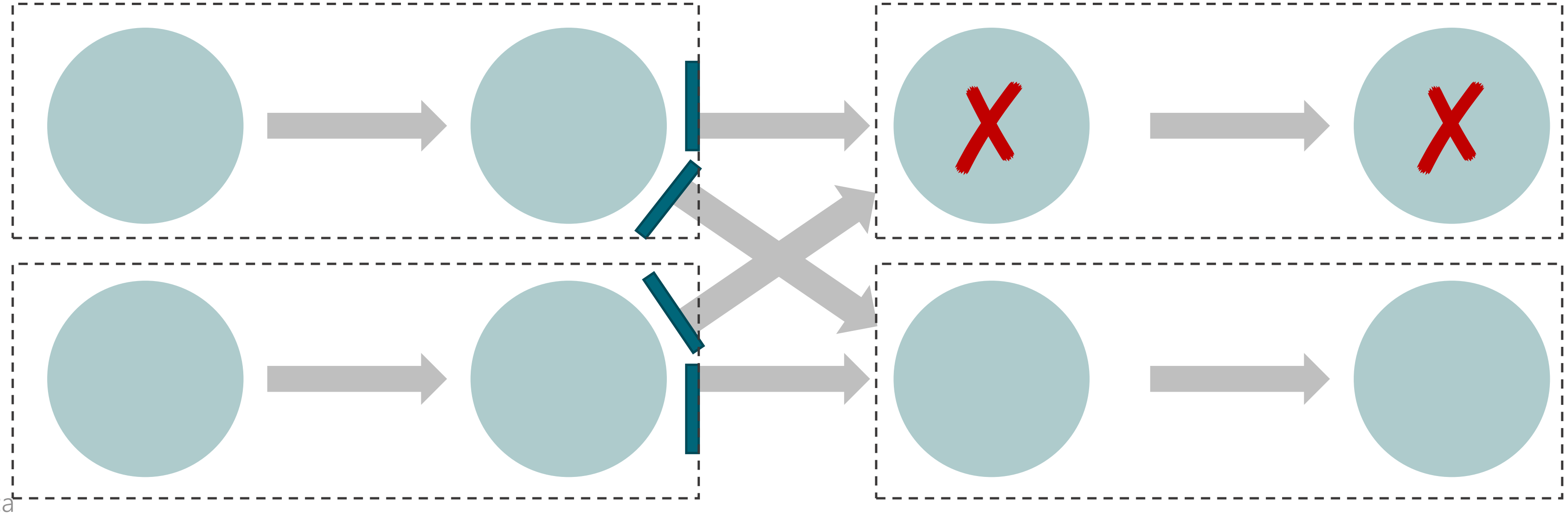
Additional Scheduling Strategies

Core operators,
cover all cases

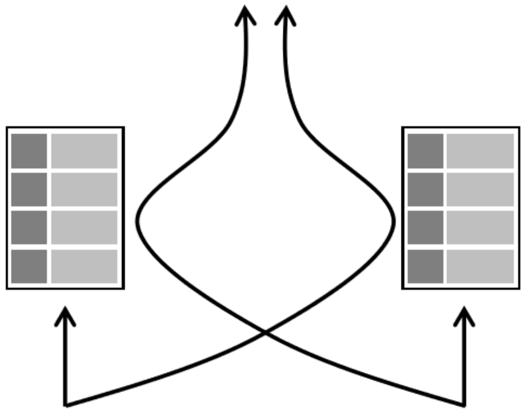
Optimized operators
for subset of cases

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

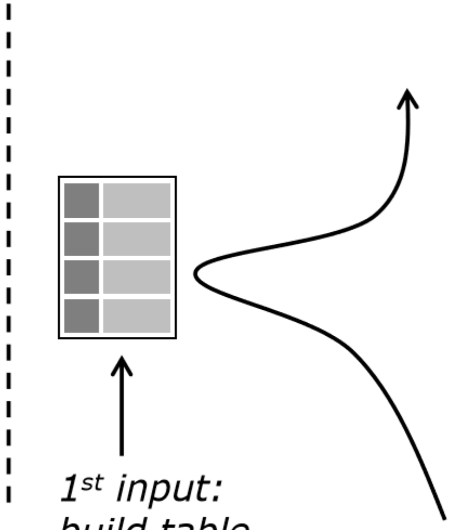


Streaming versus Batch Join



both inputs:
- build one table
- probe other table

Continuous Streaming Join



1st input:
build table

2nd input:
probe table

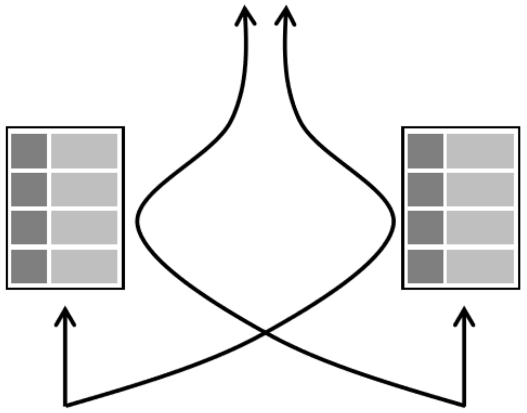
Batch Hash Join

Streaming versus Batch Join

2x RocksDB
LSM-Trees

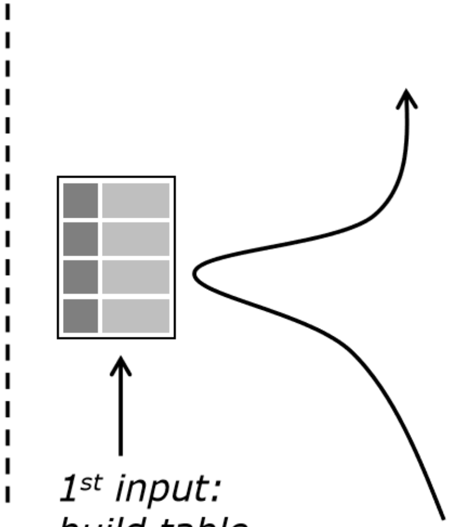
bounded/
unbounded

incremental
results



both inputs:
- build one table
- probe other table

Continuous Streaming Join



1st input:
build table

2nd input:
probe table

Batch Hash Join

1x Hybrid Hash Join

only on
bounded data

batch results

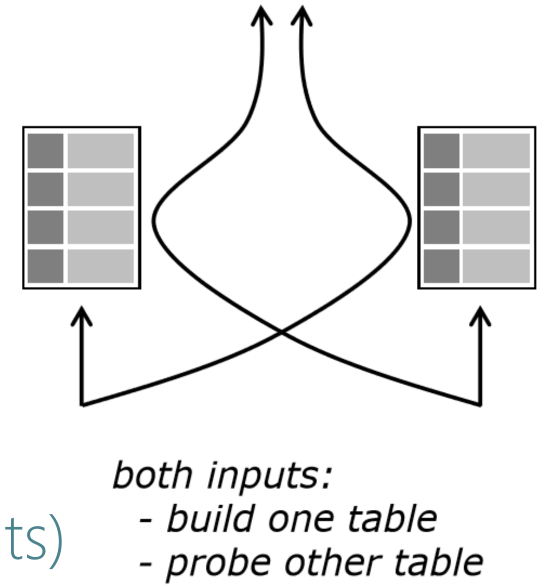
no checkpoints

more general

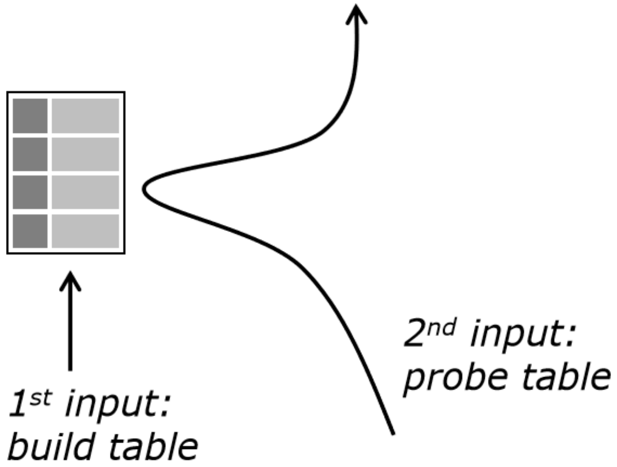
order-of-magnitude faster

Streaming versus Batch Join

push-based
(latency/checkpoints)



Continuous Streaming Join



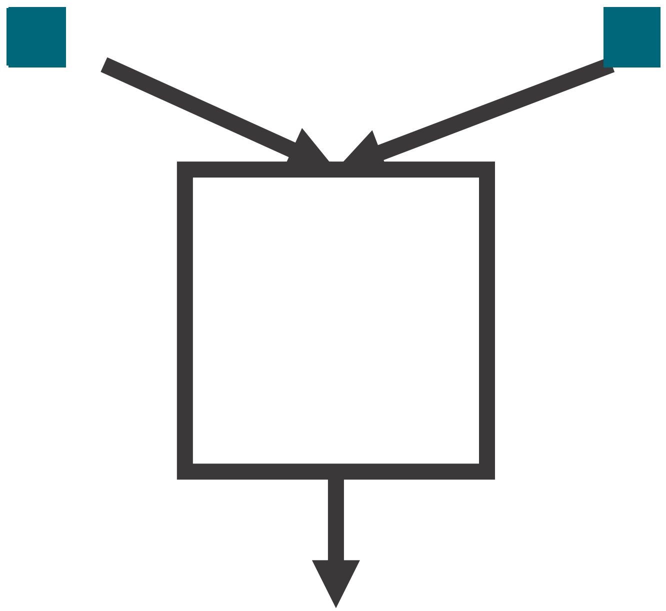
Batch Hash Join

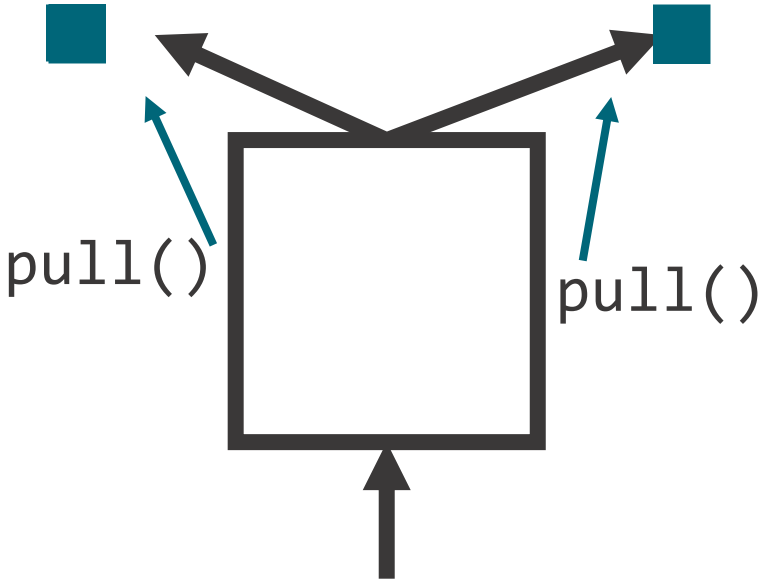
pull-based
(data flow control)

more general

order-of-magnitude faster

Push-based and Pull-based Operators





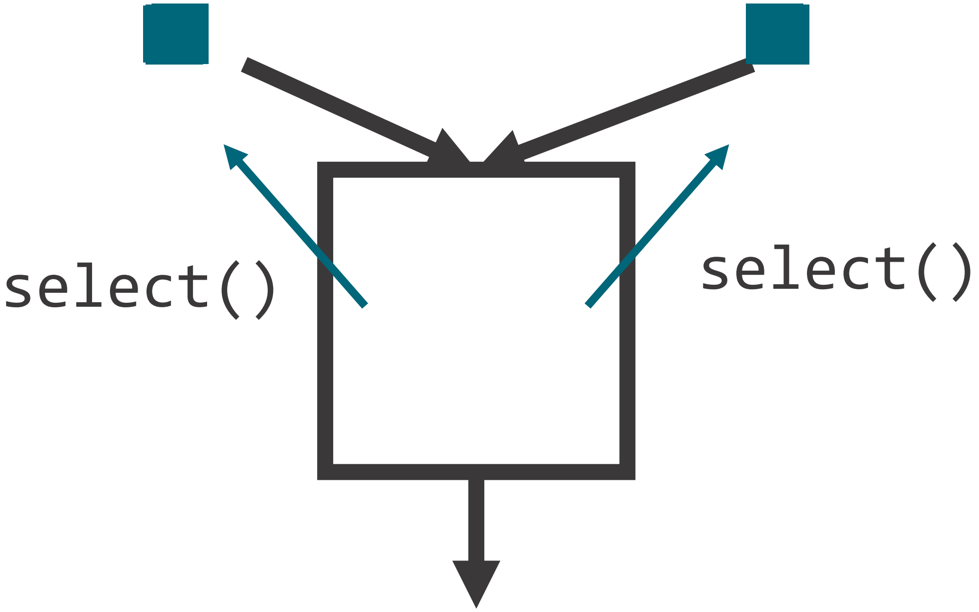
accept data from any input immediately
(like actor messages)

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

minimize latency, supports
checkpoint alignment

control over data flow,
high-latency, breaks checkpoints

Selectable Push-based Operators



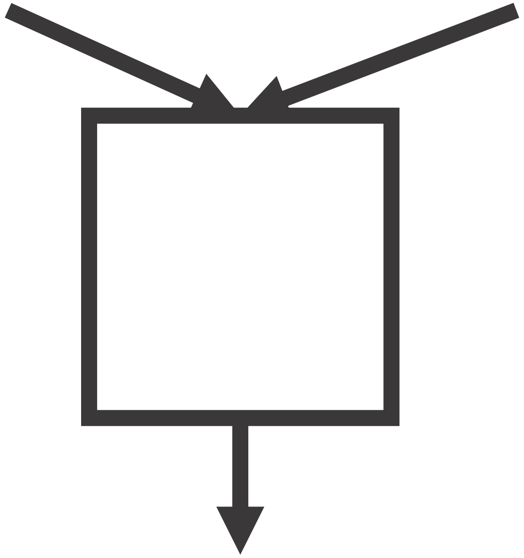
similar to non-blocking-I/O model

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)

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

➔ Input selection affects network channel credit assignment.

➔ Possible to process checkpoints through deselected channels (not yet implemented)

Flink 1.9 Table API and Merging Blink

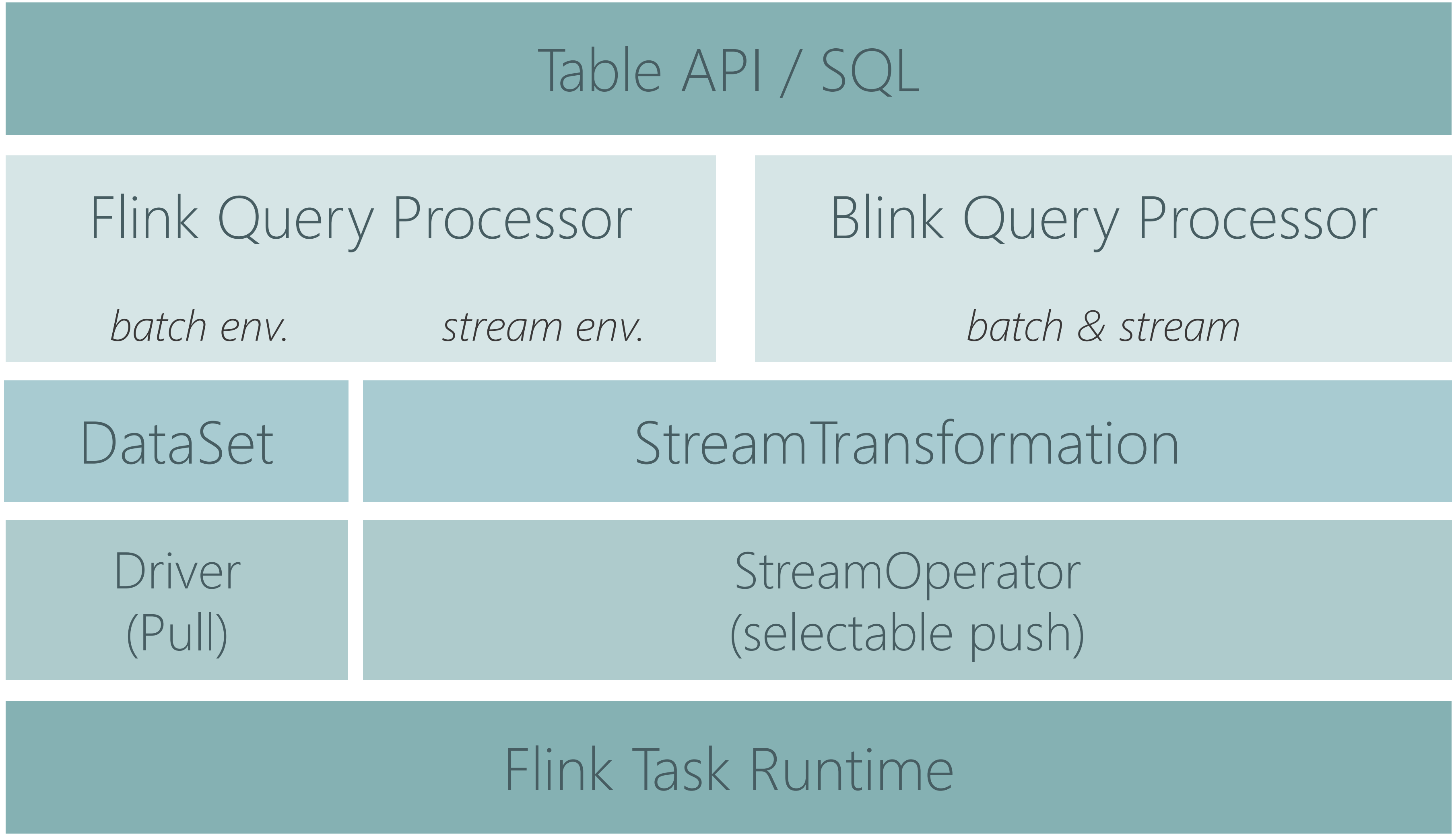


Table API / SQL

Flink Query Processor

batch env. *stream env.*

Blink Query Processor

batch & stream

DataSet

StreamTransformation

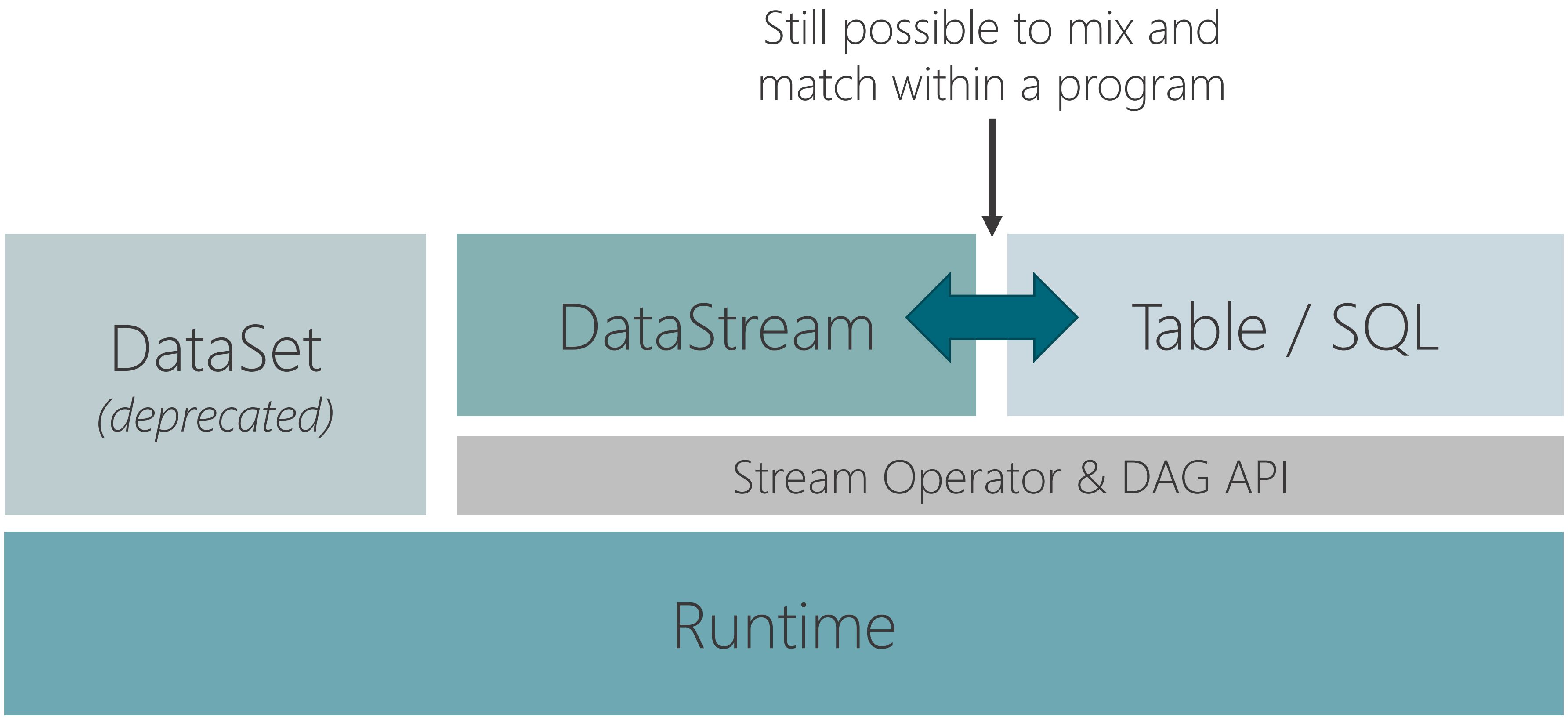
Driver
(Pull)

StreamOperator
(selectable push)

Flink Task Runtime

Stream- and Batch-Processing in the APIs

Flink's future API Stack

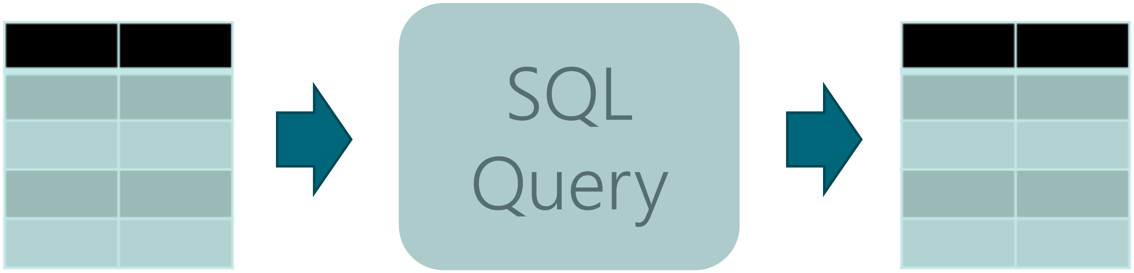


Still possible to mix and match within a program

APIs for Analytical Processing and Applications

DataStream API	SQL / Table API
Applications *(physical)*	Analytical Processing *(declarative)*
Types are Java / Scala classes	Logical Schema
Transformation Functions	Declarative Language (SQL, Table DSL)
Explicit State and Time	Automatic Optimization

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

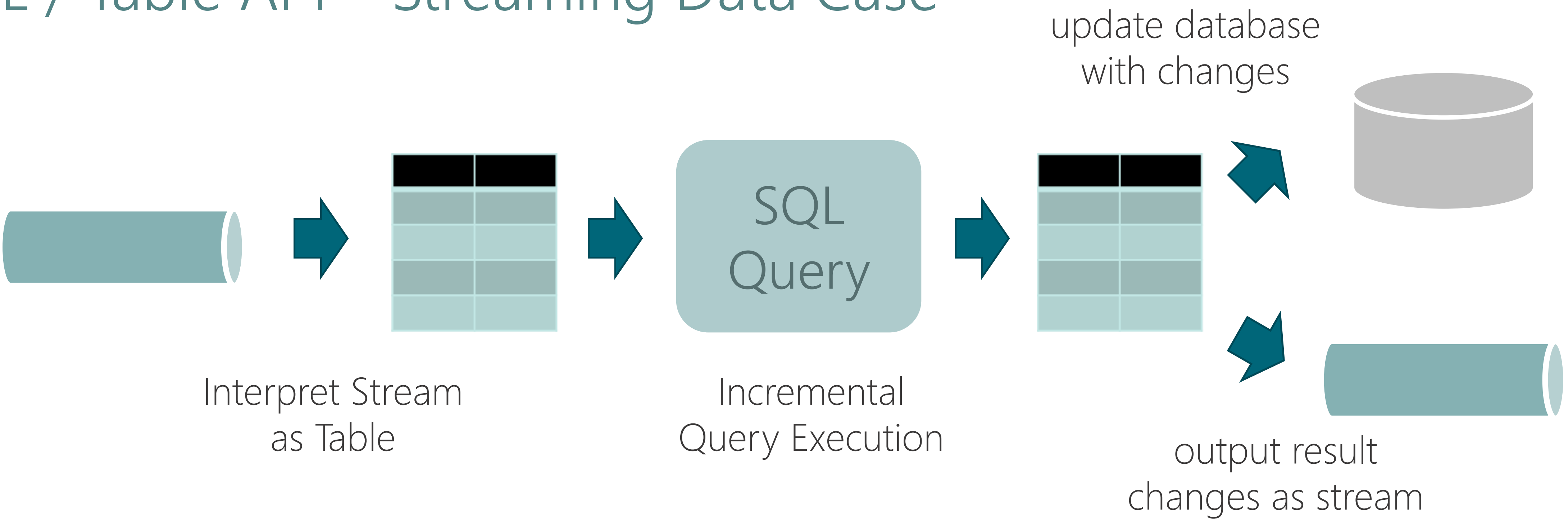


SQL Query

Batch Query Execution

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

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
```

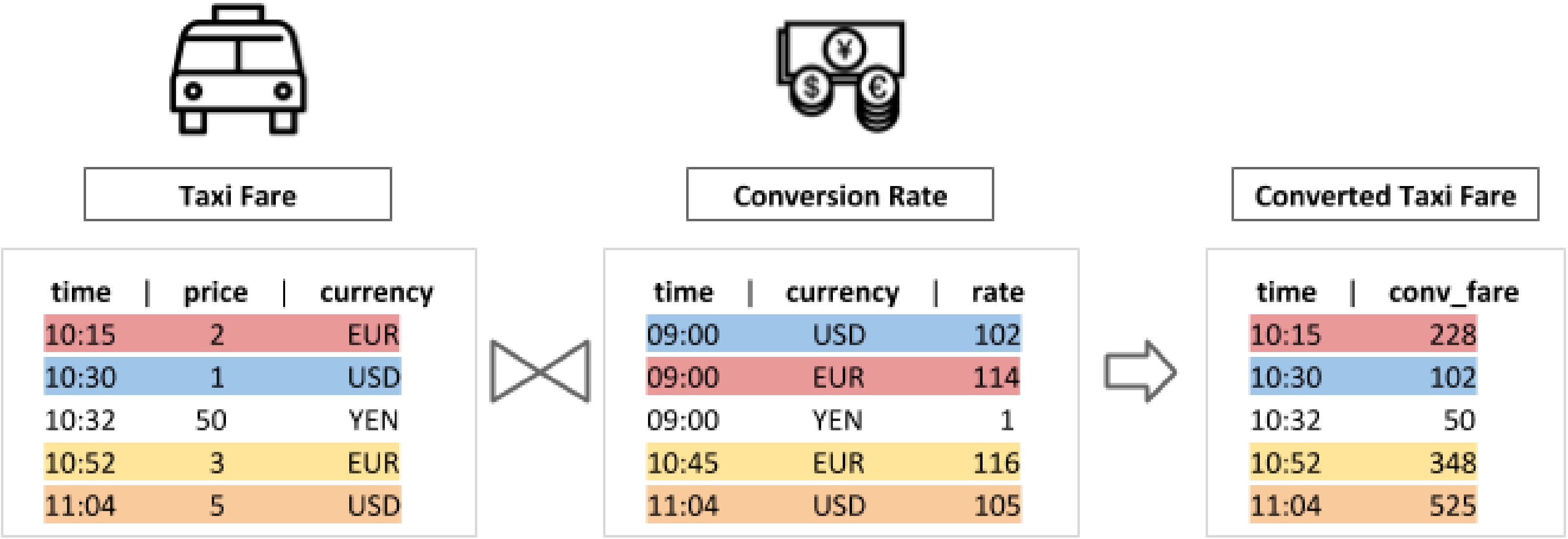
More Details also

Beam Summit Europe – Thursday June 19th

One SQL to Rule Them All – a Syntactically Idiomatic Approach to Management of Streams and Tables

Fabian Hüske, Tyler Akidau

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

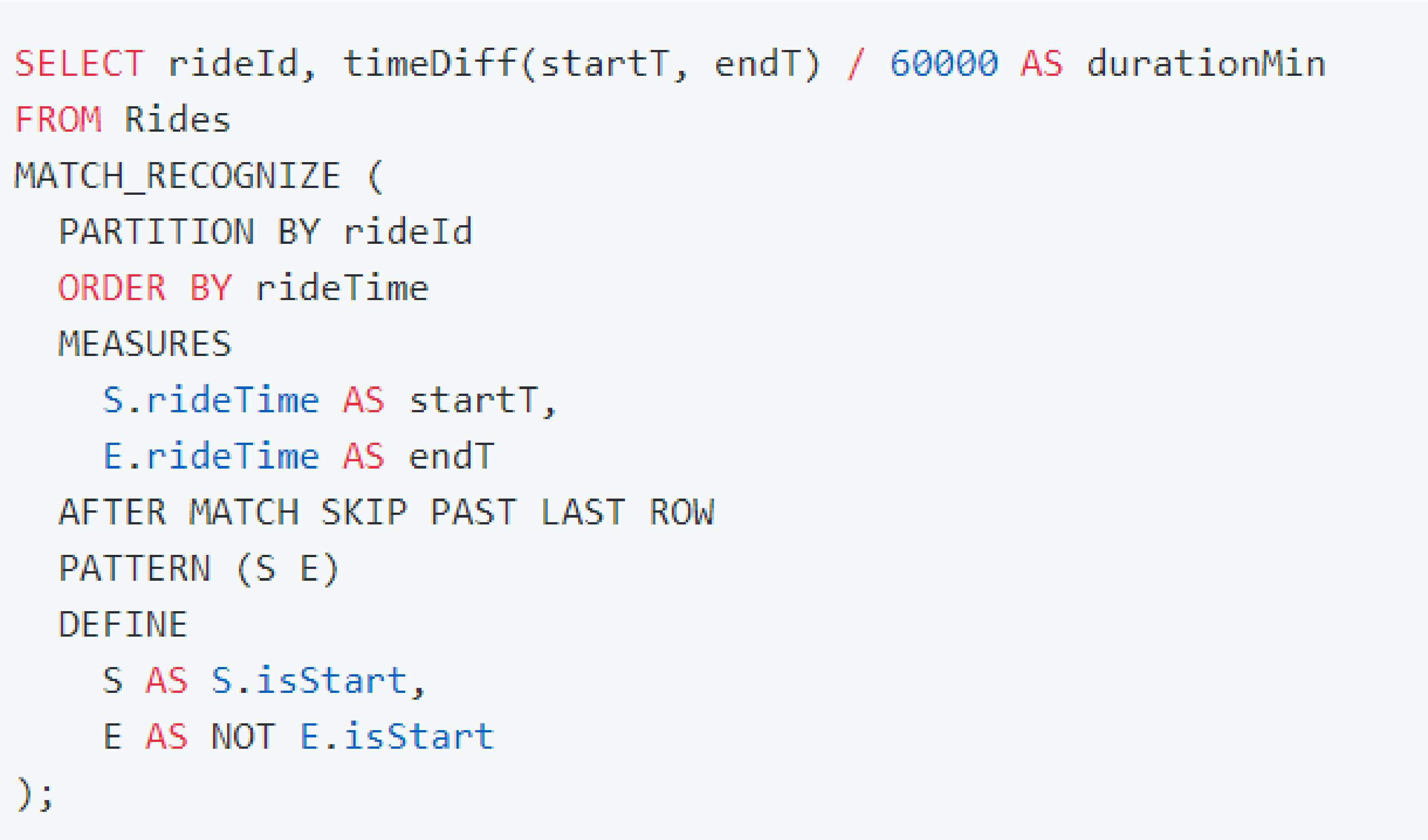
```
SELECT tf.time
       tf.price * rh.rate as conv_fare

FROM taxiFare AS tf

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

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

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
);
```

DataStream API

- DataStream is already supporting Bounded and Unbounded Streams
- 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)
- Cannot offer this mode before runtime supports batch-style execution.

This is not a final design, it is
an intermediate state of
still informal design discussions

DataStream Sources - Flink Improvement Proposal (FLIP) - 27

- Ongoing work to unify the data source API between batch and streaming
- Current draft is based on input splits and non-blocking (async) readers
- Synchronous implementations for common source threading models
- Split/partition processing in-/out-of -order
- Further goals
  - common checkpointing, per-partition watermarks, event-time idleness, event-time alignment

https://cwiki.apache.org/confluence/display/FLINK/FLIP-27%3A+Refactor+Source+Interface

What else is the Flink Community currently working on?

Hive support

Cross-Batch-Streaming
Machine Learning

Python Table API

Querying state and snapshots

More powerful incremental
streaming SQL runtime

Atomic stop-with-savepoint

Interactive multi-job programs

a big documentation overhaul

...and lot's more

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/)`

@StephanEwen @ApacheFlink @VervericaData

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