Stream Join in Flink: from Discrete to Continuous

Xingcan Cui

Shandong University, China

About Me

2017.02 – First Pull Request to Apache Flink

2018.05 – Flink Committer
(mainly on SQL/Table API)

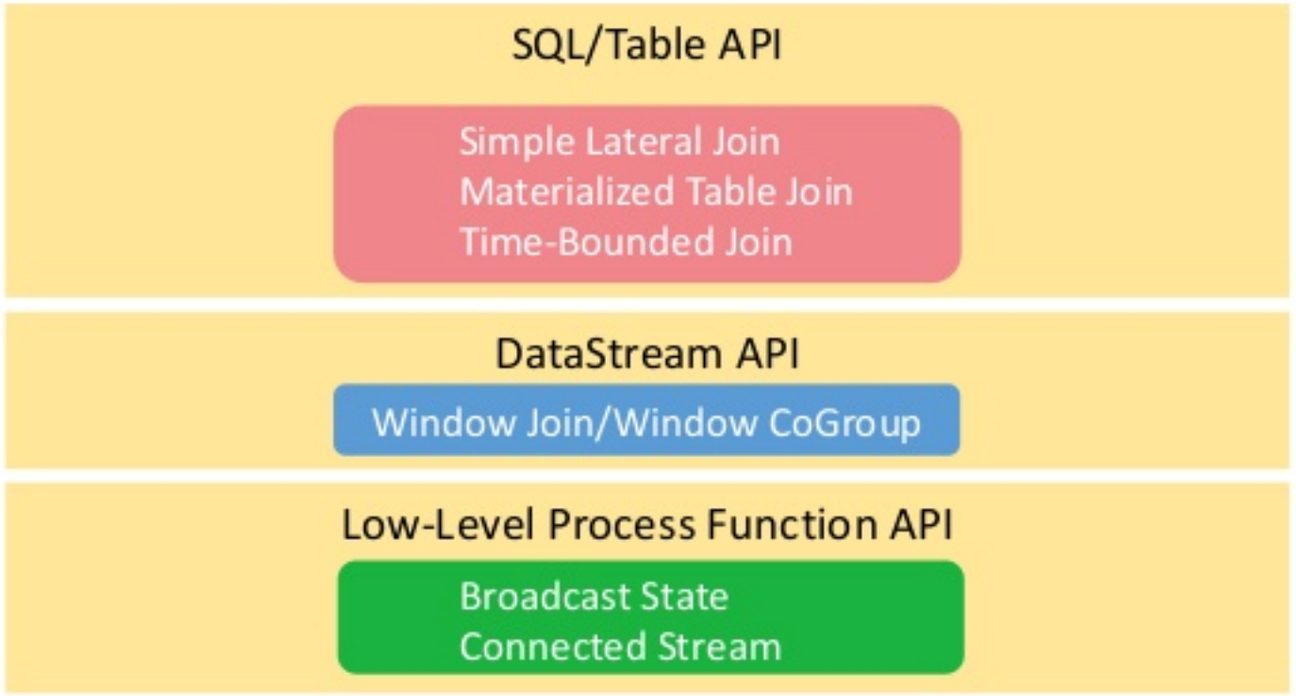
2018.06 – Ph.D. in CS from Shandong University
(supervised by Prof. Xiaohui Yu)

2018.10 – Postdoc at York University

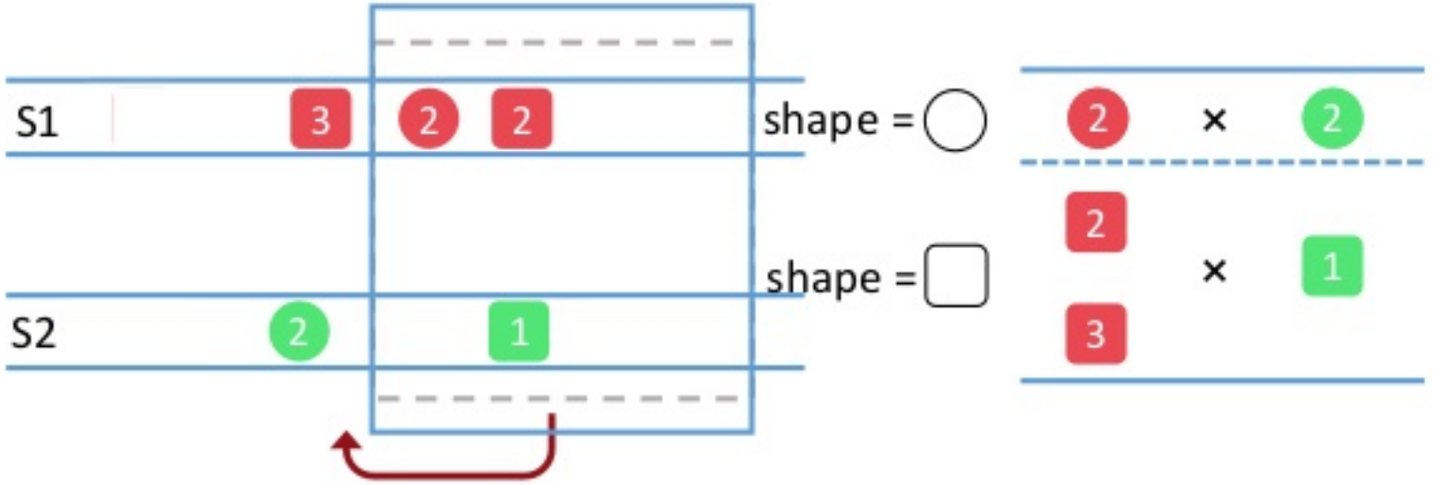
Outline

1. The Stream Join APIs
   - Window Join in DataStream API
   - Broadcast State in Low-Level API
   - Joins in SQL/Table API
2. Details about the Time-Bounded Join
   - Why Time-Bounded
   - How to Perform
   - What's More
3. Future Work

Overview of the Stream Join APIs



Window Join/Window CoGroup




```
S1.join(S2)
    .where(<shape1>).equalTo(<shape2>)
    .window(<window>)
    .apply {new JoinFunction()/FlatJoinFunction()};

S1.coGroup(S2)
    .where(<shape1>).equalTo(<shape2>)
    .window(<window>)
    .apply {new CoGroupFunction()};
```

JoinFunction

```
OUT join(
    IN1 first,
    IN2 second);
```

2 2

FlatJoinFunction

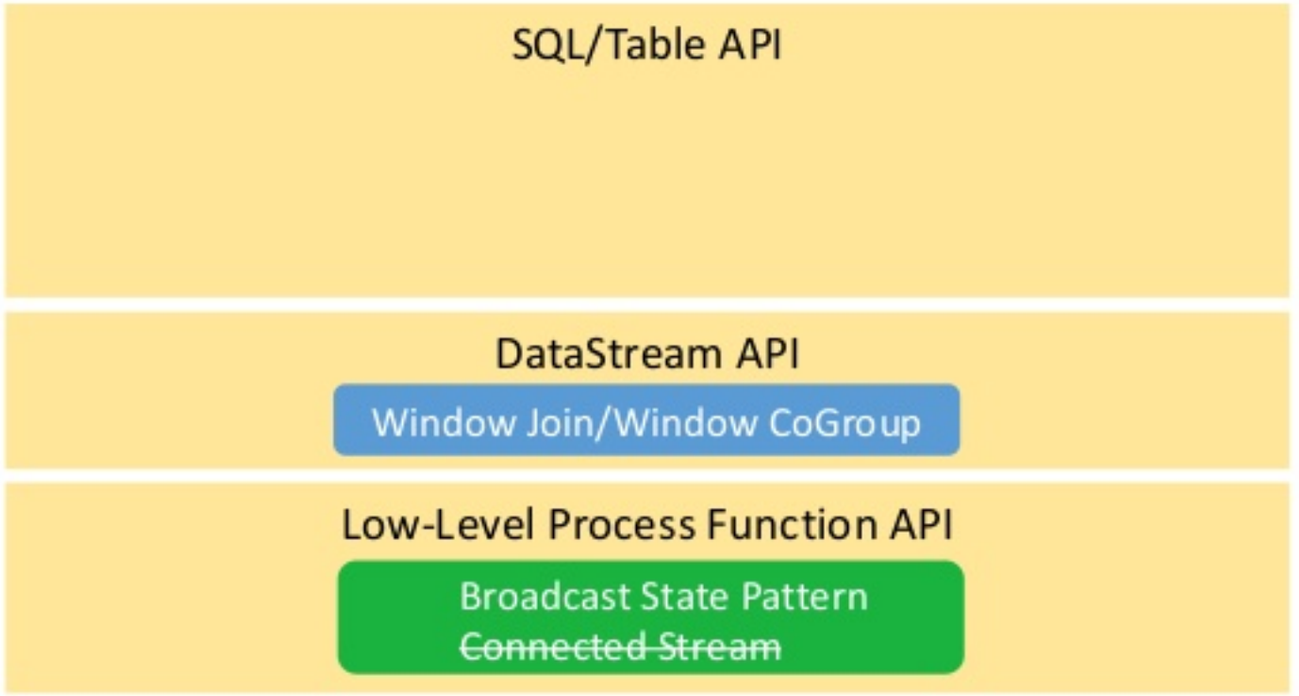
```
void join(
    IN1 first,
    IN2 second,
    Collector<OUT> out);
```

CoGroupFunction

```
void coGroup(
    Iterable<IN1> first,
    Iterable<IN2> second,
    Collector<OUT> out);
```

1

Overview of the Stream Join APIs



Broadcast State


```
MapStateDescriptor state = new MapStateDescriptor();

BroadcastStream broadcast = S2.broadcast(state);

S1.keyBy(<key selector>);
    .connect(broadcast)
    .process(new KeyedBroadcastProcessFunction());
```

```
KeyedBroadcastProcessFunction

void processElement(
    IN1 value,
    ReadOnlyContext ctx,
    Collector<OUT> out)

void processBroadcastElement(
    IN2 value,
    Context ctx,
    Collector<OUT> out)
```

r

r/w

MapState

4 1

4 2

```
processElement(...)

processBroadcastElement(...)
```

2 1

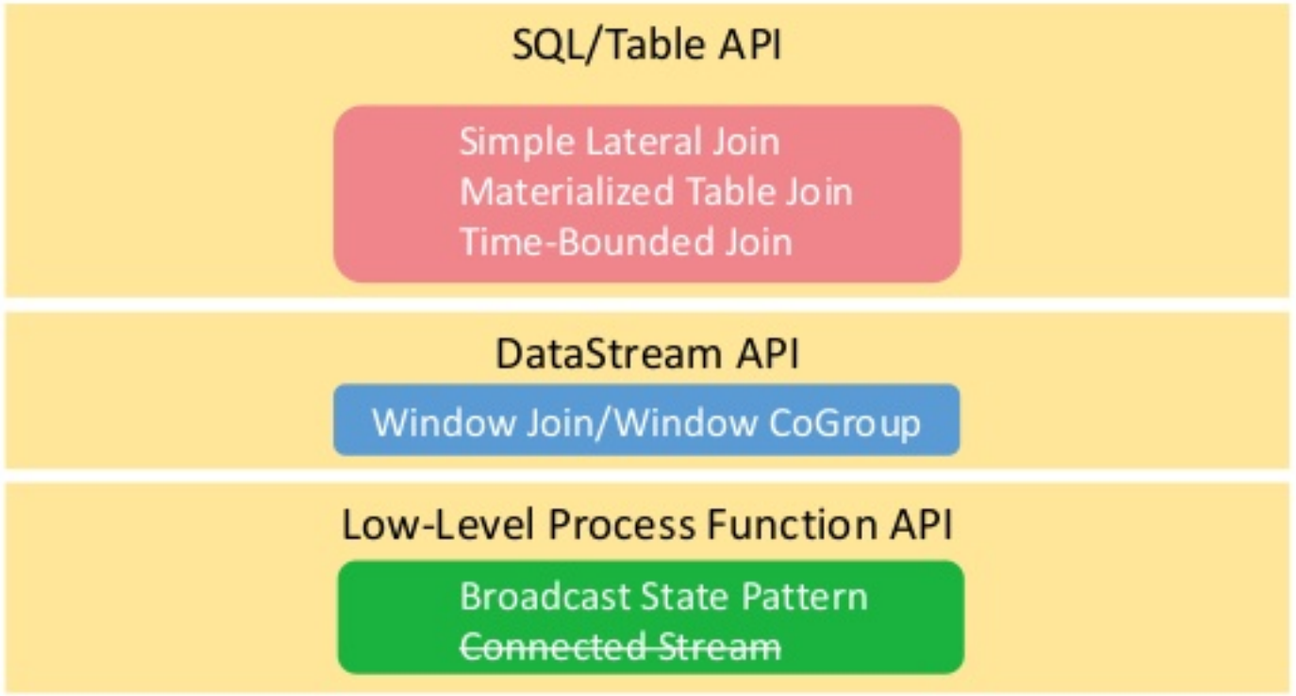
5 1

5 2

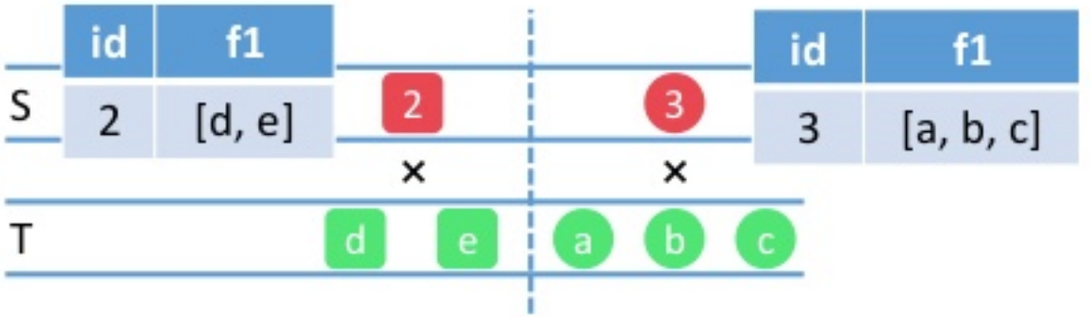
```
processElement(...)

processBroadcastElement(...)
```

3 1

Overview of the Stream Join APIs

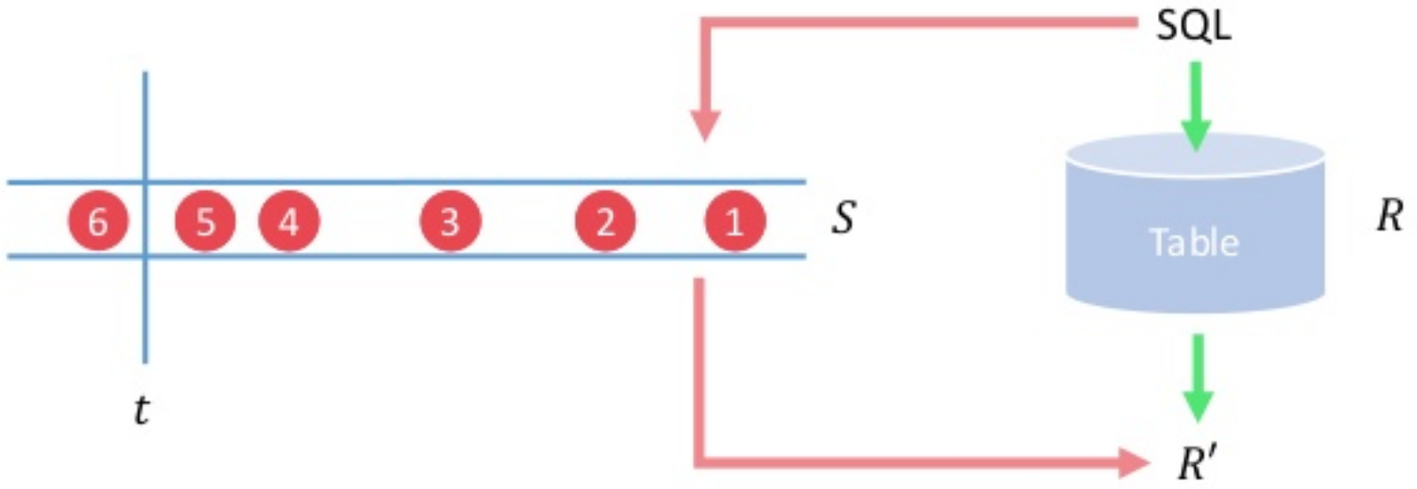


Simple Lateral Join




```
SELECT id, f2
FROM S, LATERAL TABLE(udtf(S.f1)) AS T(f2)

SELECT id, f2
FROM S, UNNEST(S1.f1) AS T(f2)
```

SQL on Streams



Stream to Table Conversion

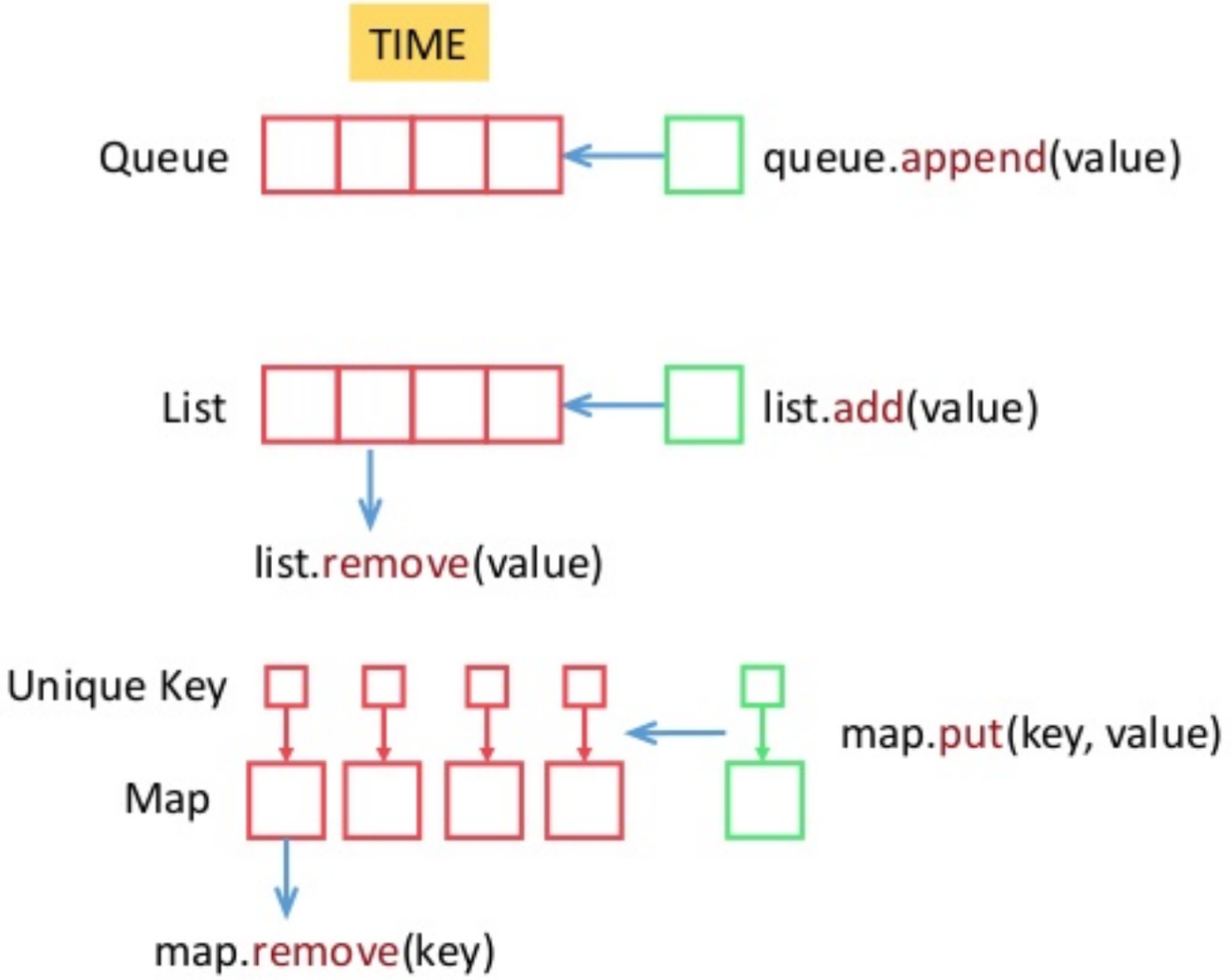
- Append Stream

Time-Bounded Join

- Retract Stream

Materialized Stream Join

- Upsert Stream



Materialized Table Join

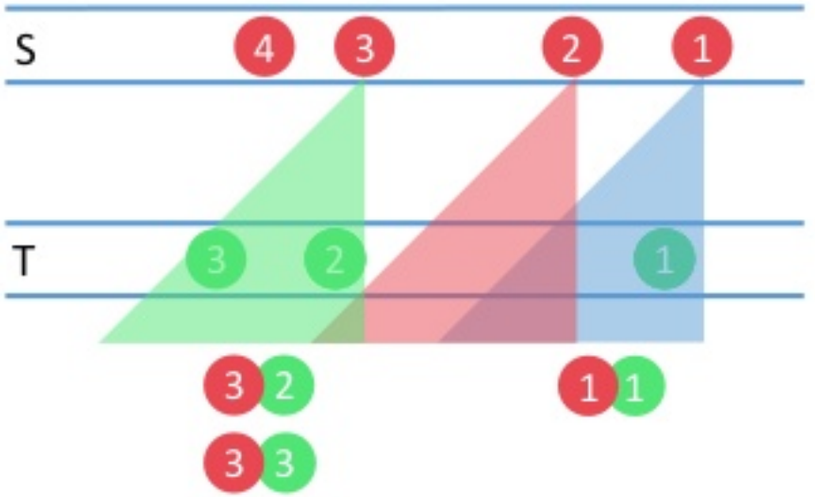

```
SELECT S.id, T.name
FROM S, T
WHERE S.id = T.id
```

State TTL Config

```
streamQueryConfig
    .withIdleStateRetentionTime(Time.hours(1), Time.hours(2));
```

1. You know the table size will never exceed the state capacity (have duplications or rows will be retracted in time).
2. The rows will be naturally expired after a period time.
3. You don't really care about the accuracy of the join result (best-effort approach).

Time-Bounded Join




```
SELECT S.id, T.time
FROM S, T
WHERE S.id = T.id AND
T.time BETWEEN S.time - INTERVAL '4' HOUR AND S.time
```

Summary of the Stream Join APIs

Name	Description	Outer Support	Non-Equi	Conti-nuous	API Level	Launched Version
Window Join/ Window CoGroup	Cut two streams into chunks and join data in the same chunk.	-/LRF	N	N	DataStream	1.0
Connected Stream	Connect two streams and jointly process them with a CoProcessFunction.				ProcessFunction	1.2
Simple Lateral Join	Join a stream with an unnested array or a user-defined table function.	L	Y	Y	SQL/Table	1.2
Time-Bounded Join	Join two append only streams with a predicate that bounds the time on both sides.	LRF	N	Y	SQL/Table	1.4
Retract Stream Join	Join two retract streams (with idle state retention time config).	LRF	N	Y	SQL/Table	1.5
Broadcast State Pattern	Join a high-throughput stream with a low-throughput stream.	LRF	Y	Y	ProcessFunction	1.5

Outline

1. The Stream Join APIs
   - Window Join in DataStream API
   - Broadcast State in Low-Level API
   - Joins in SQL/Table API
2. Details about the Time-Bounded Join
   - Why Time-Bounded
   - How to Perform
   - What's More
3. Future Work

Append Stream Join

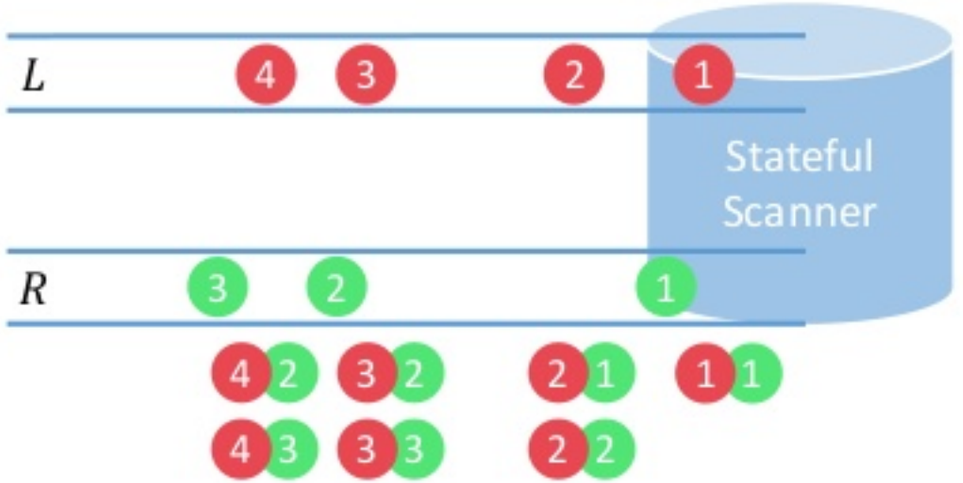
Append Stream = Sequential Scan

Time-Ordered Table

time	id	...
3	A	...
8	S	...
...	...	...

Join two time-ordered table with only one sequential scan on both sides.

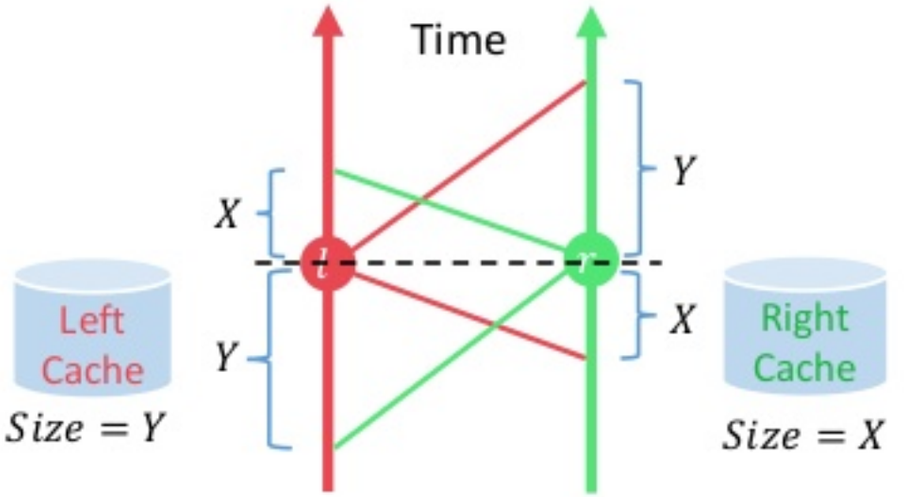
Make the Join Feasible



A sliding window that slides smoothly.

The Predicates for Time-Bounded Join

```
SELECT L.id, R.time
FROM L, R
WHERE L.id = R.id AND
R.time BETWEEN L.time − X AND L.time + Y
L.time BETWEEN R.time − Y AND R.time + X
```



1. An equi-predicate to make the join operator scalable.

2. A time-bound predicate to keep the qualified records near.

Outline

1. The Stream Join APIs
   - Window Join in DataStream API
   - Broadcast State in Low-Level API
   - Joins in SQL/Table API
2. Details about the Time-Bounded Join
   - Why Time-Bounded
   - How to Perform
   - What's More
3. Future Work

The Connected Stream

```
leftStream
    .connect(rightStream)
    .keyBy(<key selector for the equi-predicate>)
    .process(new KeyedCoProcessFunction())
```

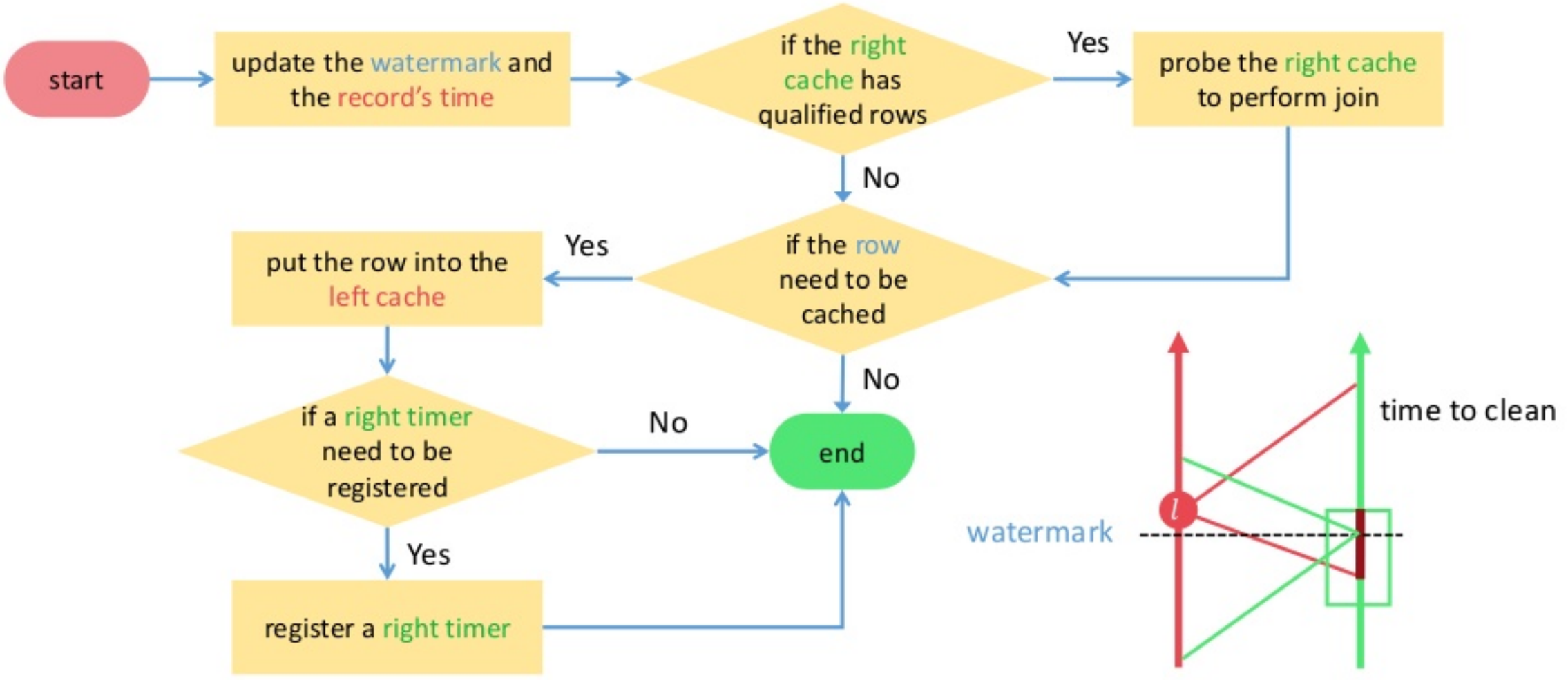
KeyedCoProcessFunction

```
void processElement1(
    IN1 value,
    Context ctx,
    Collector<OUT> out)

void processElement2(
    IN2 value,
    Context ctx,
    Collector<OUT> out)

void onTimer(
    long timestamp,
    OnTimerContext ctx,
    Collector<OUT> out)
```

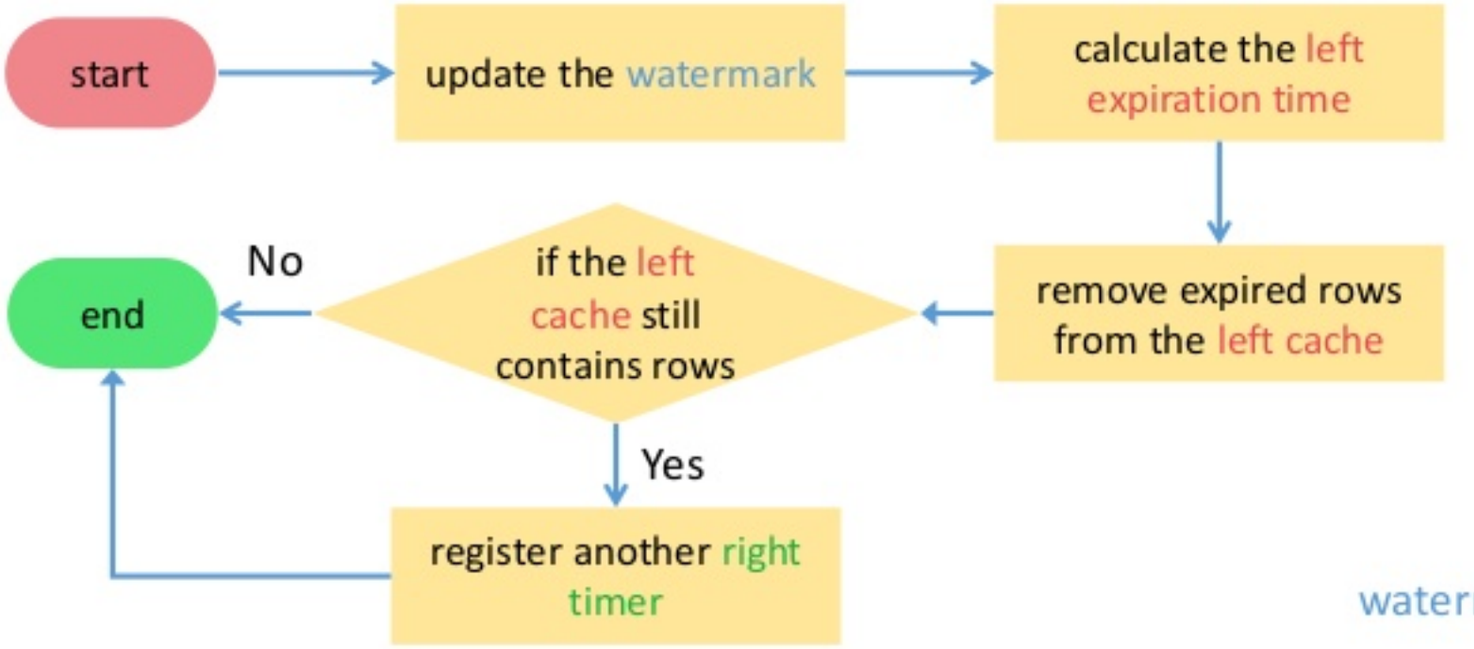
left timer Map[Long, List[Row]]

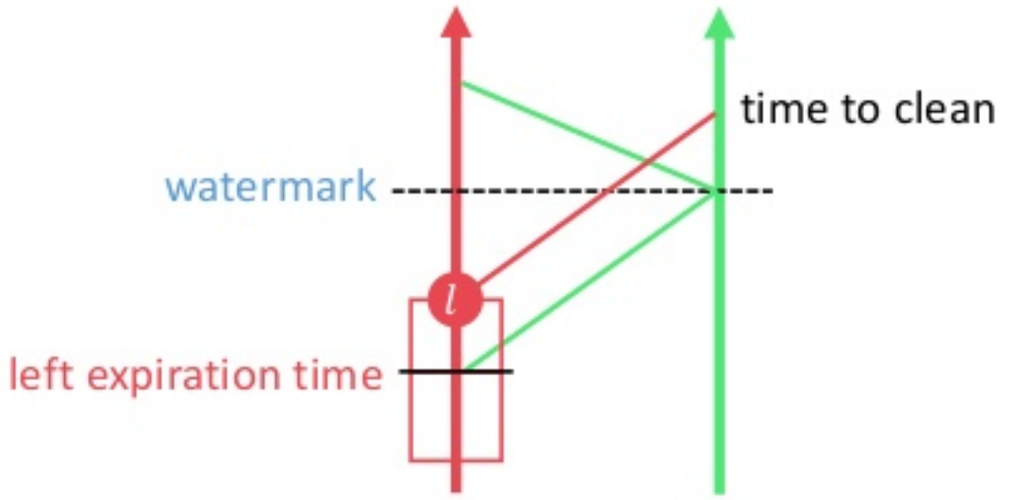
right timer time -> records of that time

The Logic for processElement()



The Logic for onTimer()





Outline

1. The Stream Join APIs
   - Window Join in DataStream API
   - Broadcast State in Low-Level API
   - Joins in SQL/Table API
2. Details about the Time-Bounded Join
   - Why Time-Bounded
   - How to Perform
   - What's More
3. Future Work

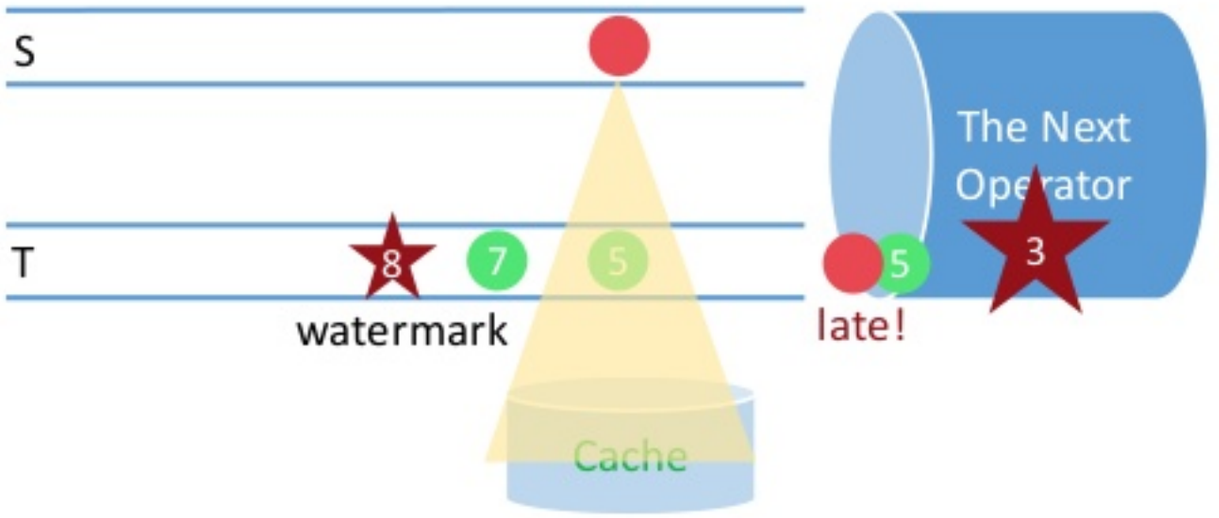
About the Join Result

1. The result for a time-bounded join is still an append stream.

2. When converting the join result to a DataStream, at most one rowtime field can be reserved.

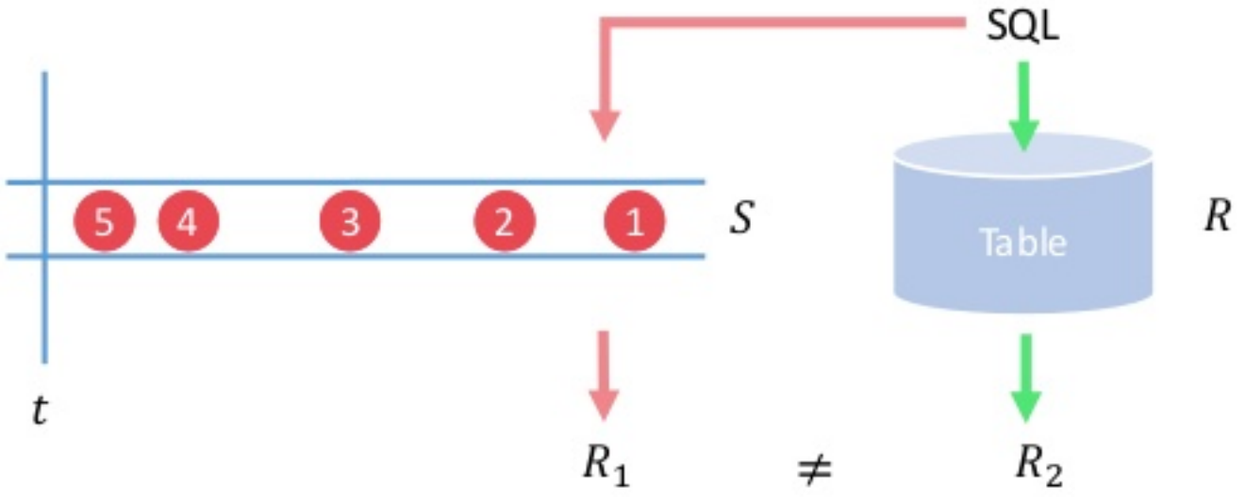
```
SELECT ltime, CAST(rtime AS TIMESTAMP), ... FROM ...
```

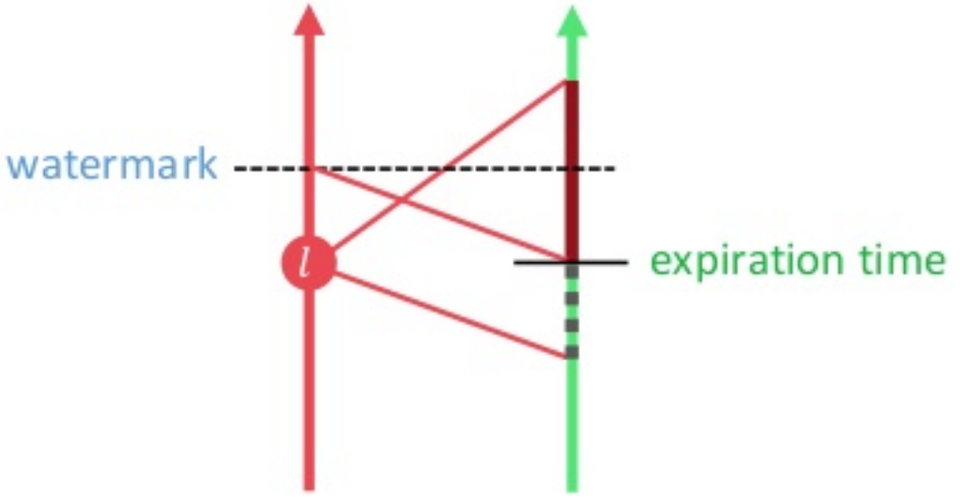
Holding Back Watermarks



Subtract a fixed value from each watermark.

The Integrity Problem





A. Drop l as if it never comes (or use side output).

B. Keep l to produce as many results as possible.

R_1 could be affected by lateness, thus may not be identical with R_2.

Outline

1. The Stream Join APIs
   - Window Join in DataStream API
   - Broadcast State in Low-Level API
   - Joins in SQL/Table API
2. Details about the Time-Bounded Join
   - Why Time-Bounded
   - How to Perform
   - What's More
3. Future Work

P1. More Kinds of Stream Join

An efficient policy to clean the cache.

1. Time Versioned Join (WIP)

2. One-to-One Join (query hints?)

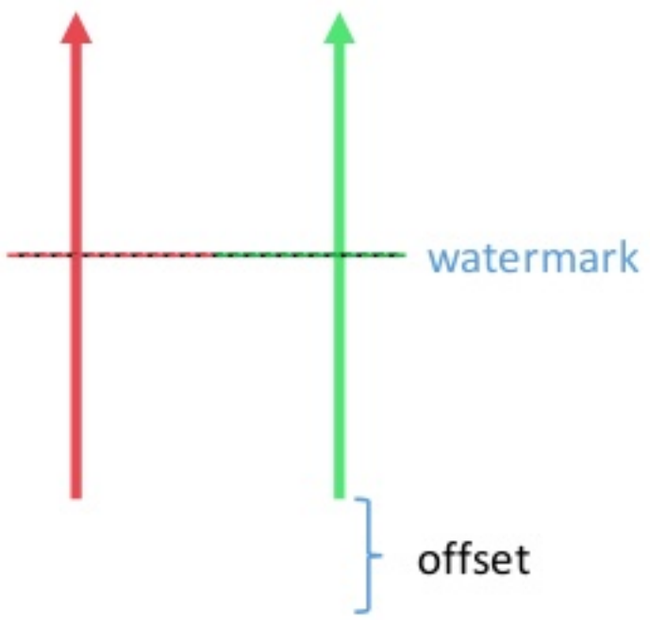
P2. Better State

1. Sorted MapState

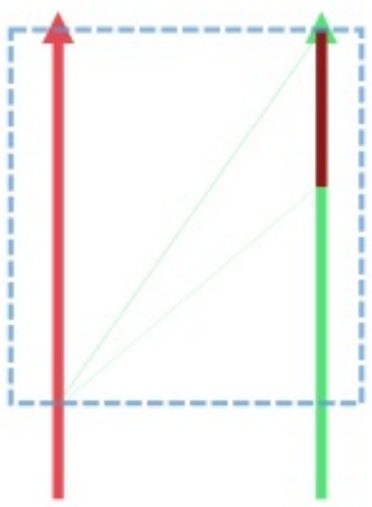
```
mapState.getByRange(start, end);

mapState.removeByRange(start, end);
```

2. RocksDB Backend for Operator State

P3. Separated Watermarks



Cache Size Optimization

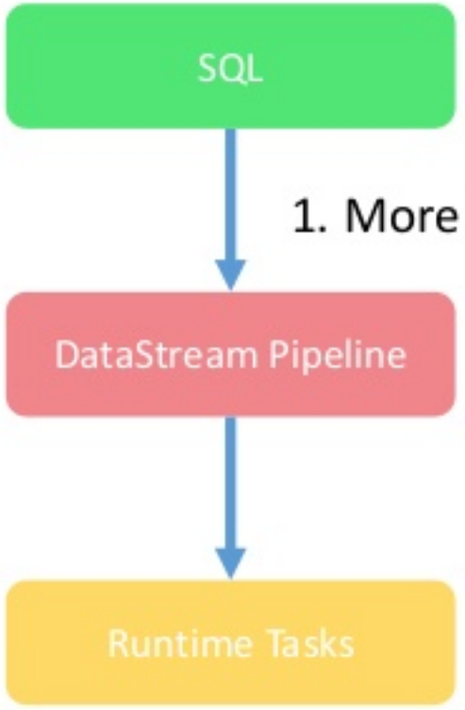


X $(RSize)$ | Y $(LSize)$

```
R.time BETWEEN L.time -(-30) MINUTE AND L.time +50 MINUTE
R.time BETWEEN L.time -(-10) MINUTE AND L.time +30 MINUTE
R.time BETWEEN L.time -(+ 0) MINUTE AND L.time +20 MINUTE
R.time BETWEEN L.time -(+ 5) MINUTE AND L.time +15 MINUTE
```

The size of the two caches can be optimized to be the minimum value, $X + Y$.

P4. Query Optimization



1. More Stream Specific Rules

2. Setting the Parallelism (hints/sampling inputs)

3. Adaptive Query Optimization

Q&A

https://www.youtube.com/watch?v=yDUfz3OFcOo

https://baike.baidu.com/item/%E5%8E%8B%E9%9D%A2%E6%9C%BA/3477203