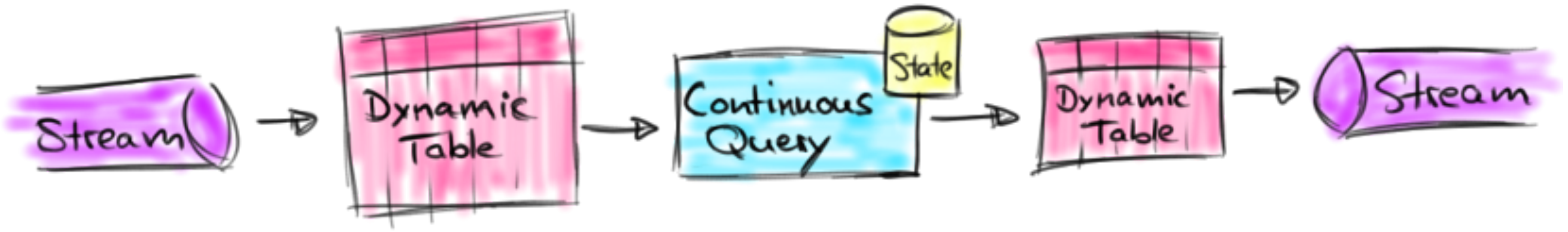
Streams & Tables

Most of the images used in this section are courtesy of the Flink Documentation

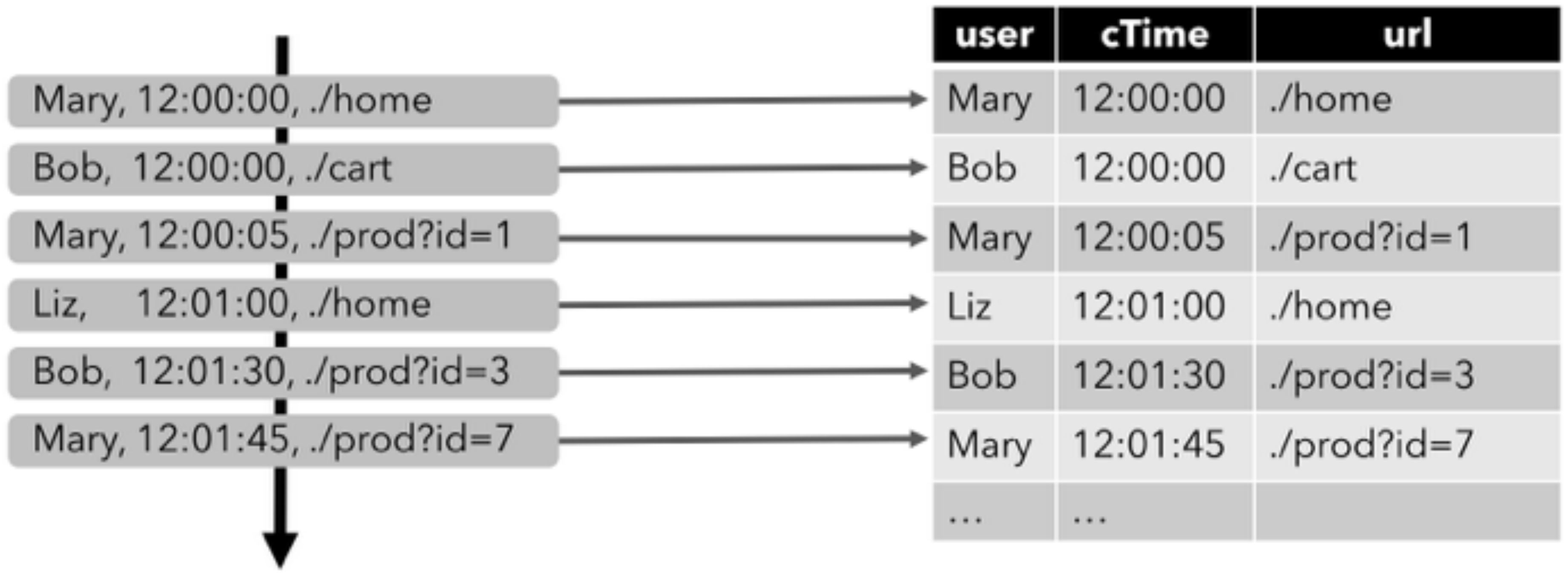
Dynamic Tables



- Tables changing over time
- S2R -> R2R -> R2S
- How to pass from one to the other?

S2R

Stream to Relation operator converts a stream into a relation (dynamic table) by appending every record to the table

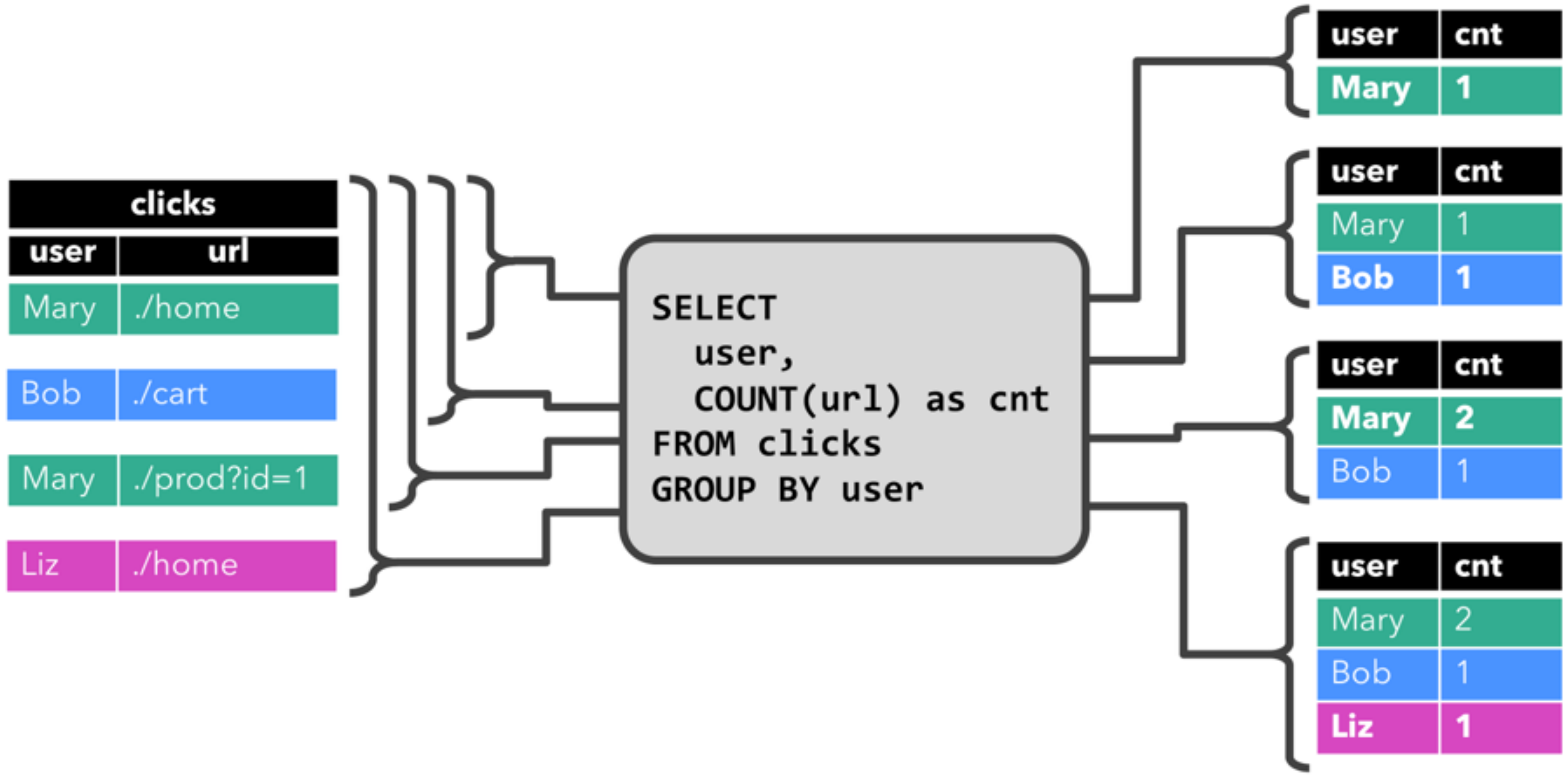

Mary, 12:00:00, ./home
Bob, 12:00:00, ./cart
Mary, 12:00:05, ./prod?id=1
Liz, 12:01:00, ./home
Bob, 12:01:30, ./prod?id=3
Mary, 12:01:45, ./prod?id=7

user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:00:05	./prod?id=1
Liz	12:01:00	./home
Bob	12:01:30	./prod?id=3
Mary	12:01:45	./prod?id=7
…	…	

R2R

- **Relation to Relation** operators are the classic SQL operators that given a table provide a table as result;
- If the input table is dynamic (because it records a stream) the SQL query becomes a continuous query;
- SQL queries on dynamic tables can involve time-dependent transformations (windows).

R2R - Continuous Query (1)



clicks	
user	url
Mary	./home
Bob	./cart
Mary	./prod?id=1
Liz	./home

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

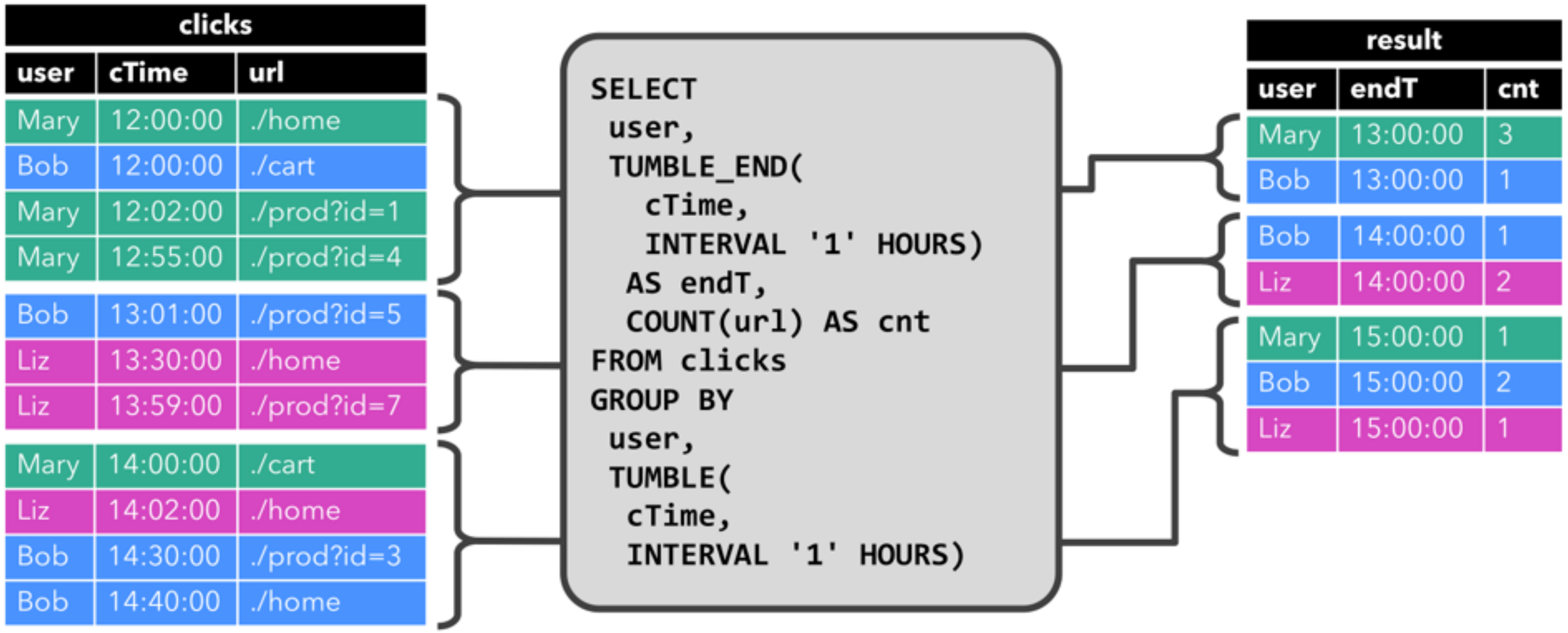
user	cnt
Mary	**1**

user	cnt
Mary	1
Bob	**1**

user	cnt
Mary	**2**
Bob	1

user	cnt
Mary	2
Bob	1
Liz	**1**

R2R - Continuous Query (2)



clicks

user	cTime	url
Mary	12:00:00	./home
Bob	12:00:00	./cart
Mary	12:02:00	./prod?id=1
Mary	12:55:00	./prod?id=4
Bob	13:01:00	./prod?id=5
Liz	13:30:00	./home
Liz	13:59:00	./prod?id=7
Mary	14:00:00	./cart
Liz	14:02:00	./home
Bob	14:30:00	./prod?id=3
Bob	14:40:00	./home

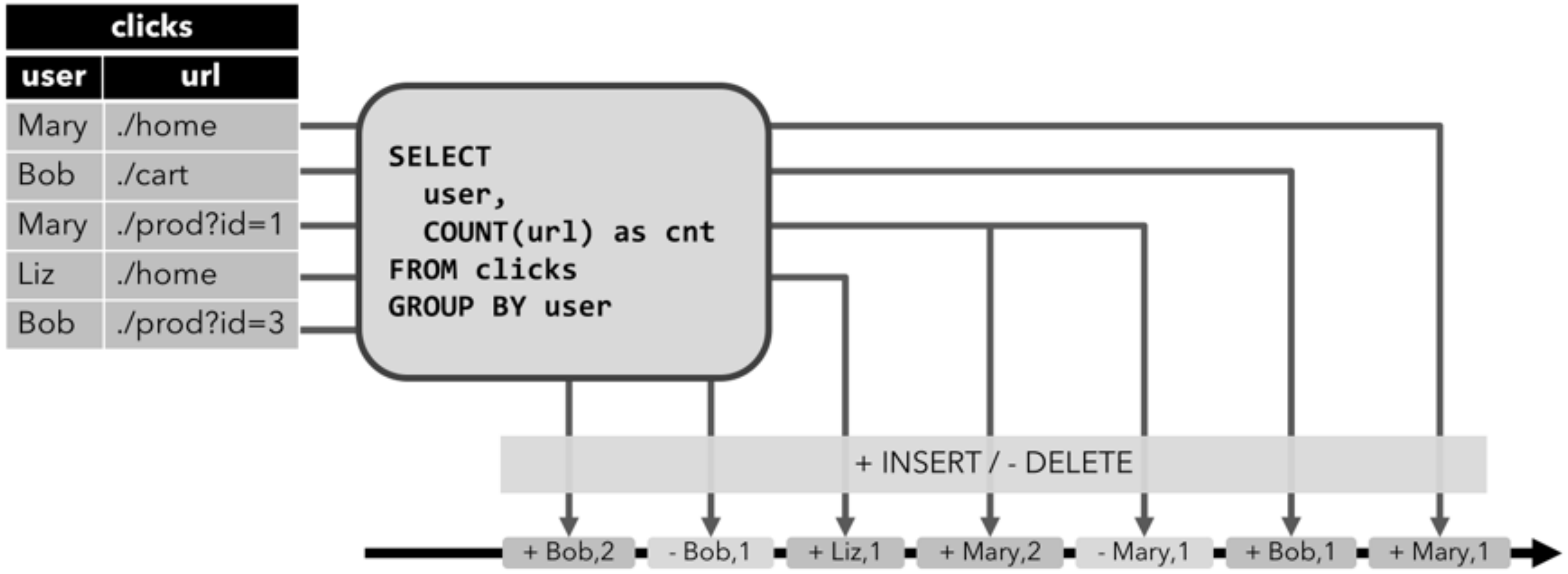
```
SELECT
 user,
 TUMBLE_END(
   cTime,
   INTERVAL '1' HOURS)
  AS endT,
  COUNT(url) AS cnt
FROM clicks
GROUP BY
 user,
 TUMBLE(
  cTime,
  INTERVAL '1' HOURS)
```

result

user	endT	cnt
Mary	13:00:00	3
Bob	13:00:00	1
Bob	14:00:00	1
Liz	14:00:00	2
Mary	15:00:00	1
Bob	15:00:00	2
Liz	15:00:00	1

R2S

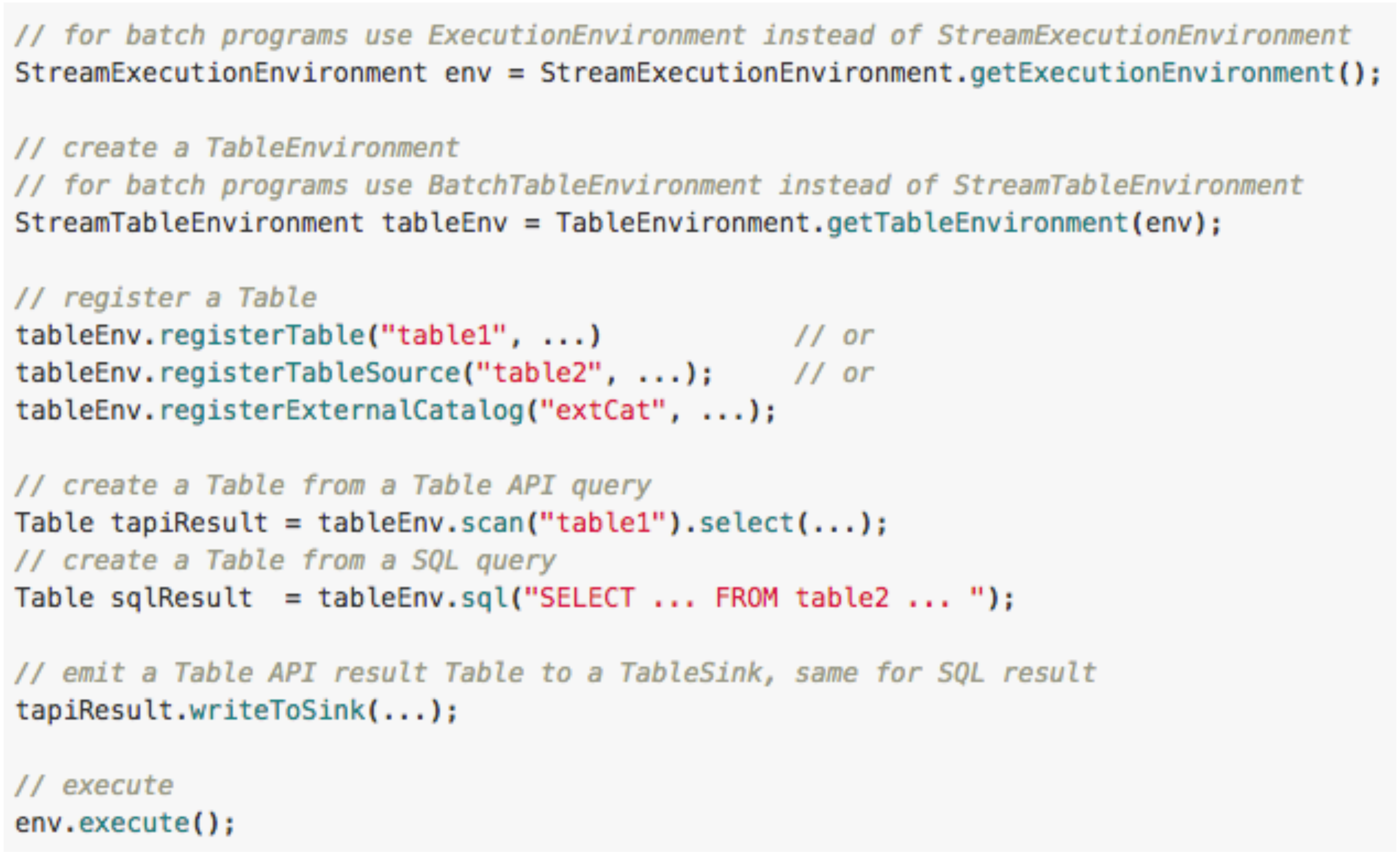
Relation to Stream convert a dynamic table to a stream as a log of updates (insert/update)



clicks	
user	**url**
Mary	./home
Bob	./cart
Mary	./prod?id=1
Liz	./home
Bob	./prod?id=3

```
SELECT
  user,
  COUNT(url) as cnt
FROM clicks
GROUP BY user
```

API



```
// for batch programs use ExecutionEnvironment instead of StreamExecutionEnvironment
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

// create a TableEnvironment
// for batch programs use BatchTableEnvironment instead of StreamTableEnvironment
StreamTableEnvironment tableEnv = TableEnvironment.getTableEnvironment(env);

// register a Table
tableEnv.registerTable("table1", ...)            // or
tableEnv.registerTableSource("table2", ...);     // or
tableEnv.registerExternalCatalog("extCat", ...);

// create a Table from a Table API query
Table tapiResult = tableEnv.scan("table1").select(...);
// create a Table from a SQL query
Table sqlResult  = tableEnv.sql("SELECT ... FROM table2 ... ");

// emit a Table API result Table to a TableSink, same for SQL result
tapiResult.writeToSink(...);

// execute
env.execute();
```

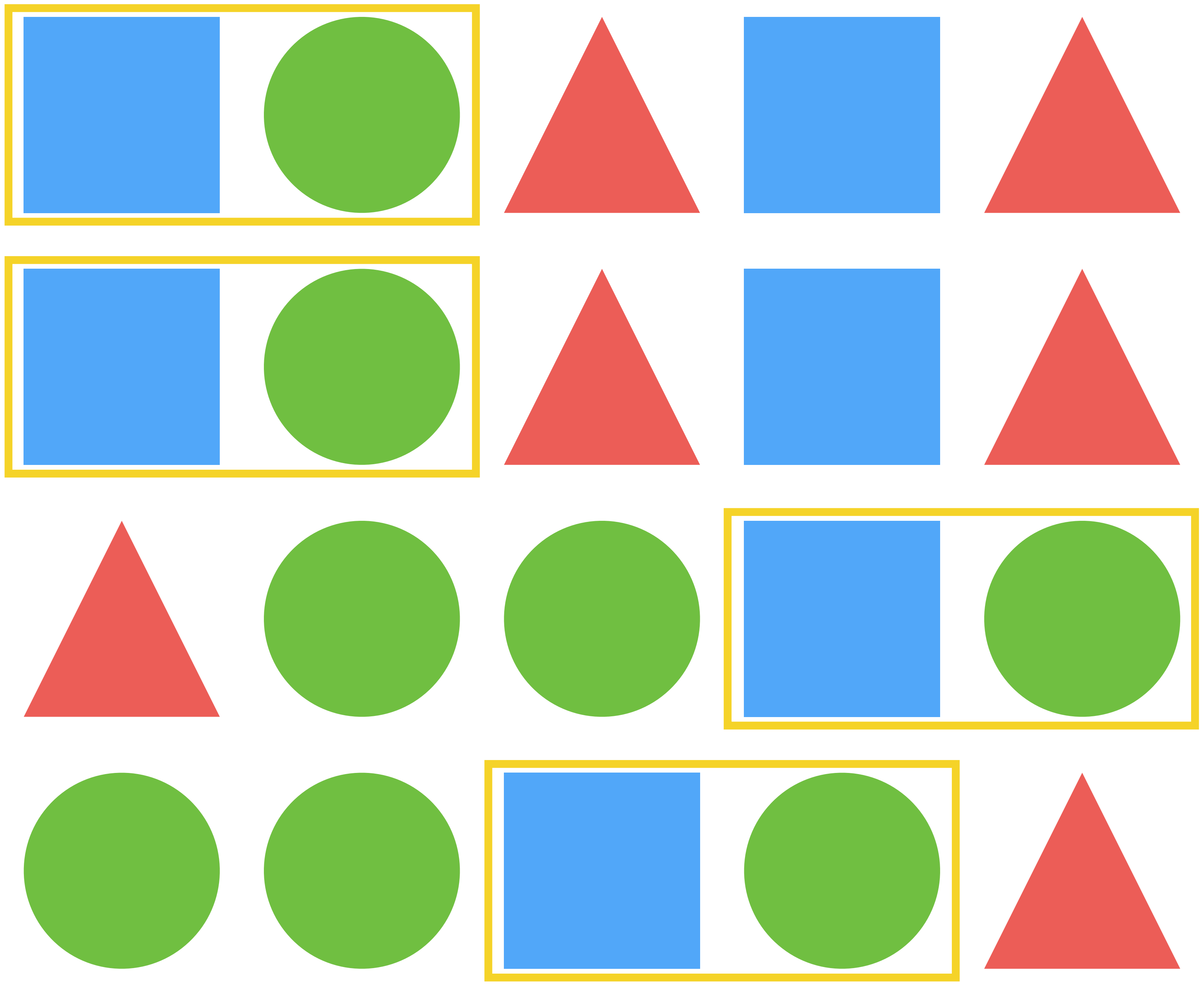
CQL

- Is the **Continuous Query Language** created by Jennifer Widom in 2003[1];
- I intentionally used **S2R**, **R2R**, and **R2S** because it is the notation that Widom used;
- The major difference between CQL and Flink Dynamic Tables lies in S2R: in CQL the stream to relation operator is the window;
- R2R is simply static SQL.

[1] CQL paper

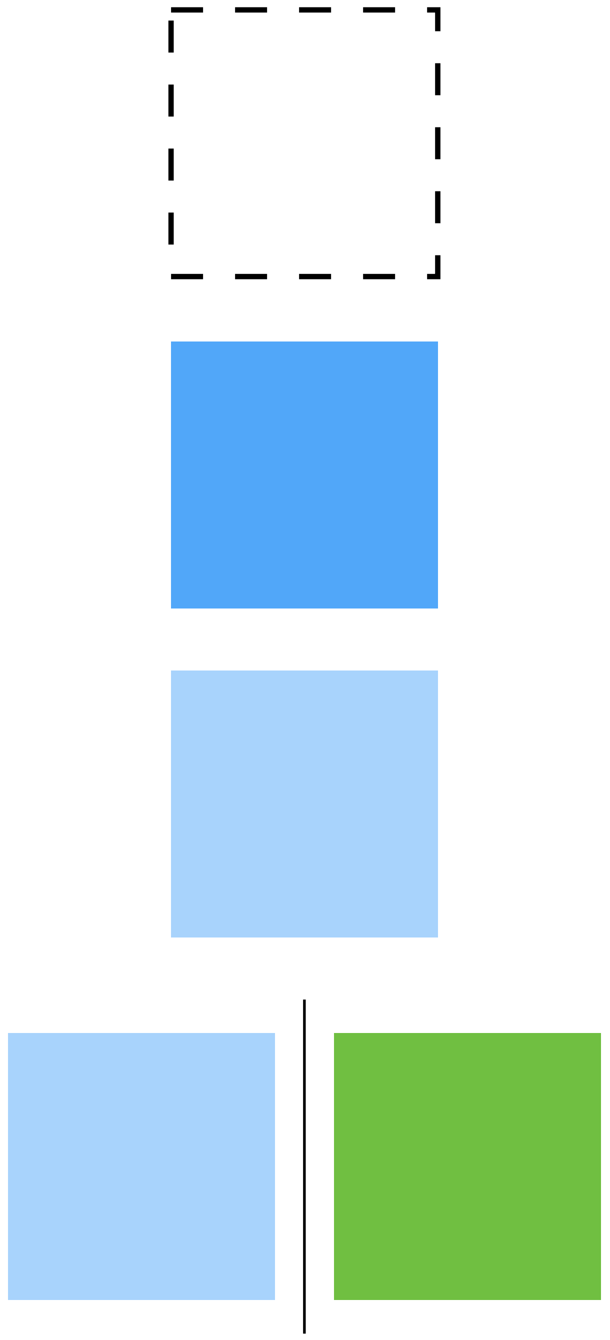
The CEP Library

Patterns



WARNING: in these slides, time flows from left to right! **—t—>**

Conditions



Shape is Square

```
p.subtype(Square.class)
```

and is blue

```
.where(
  s -> s.color() == Color.BLUE
)
```

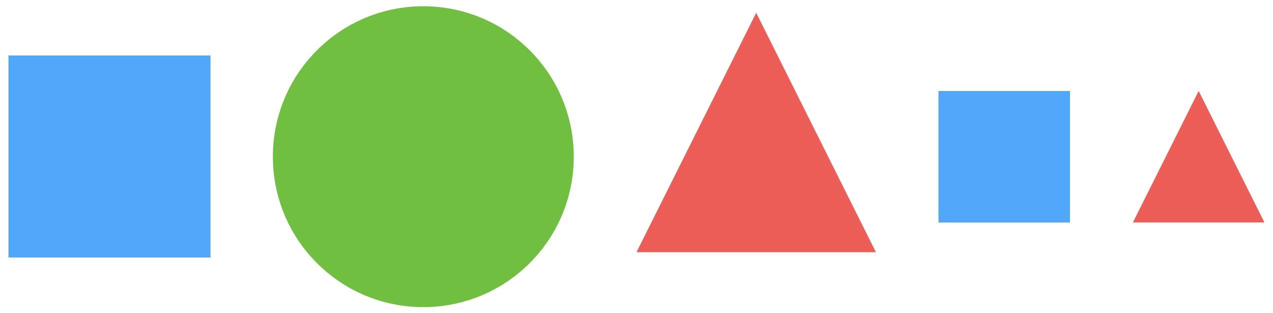
and is transparent

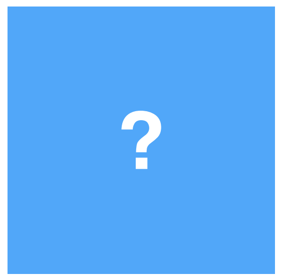
```
.where(…) // AND
```

or is green

```
.or(…) // OR
```

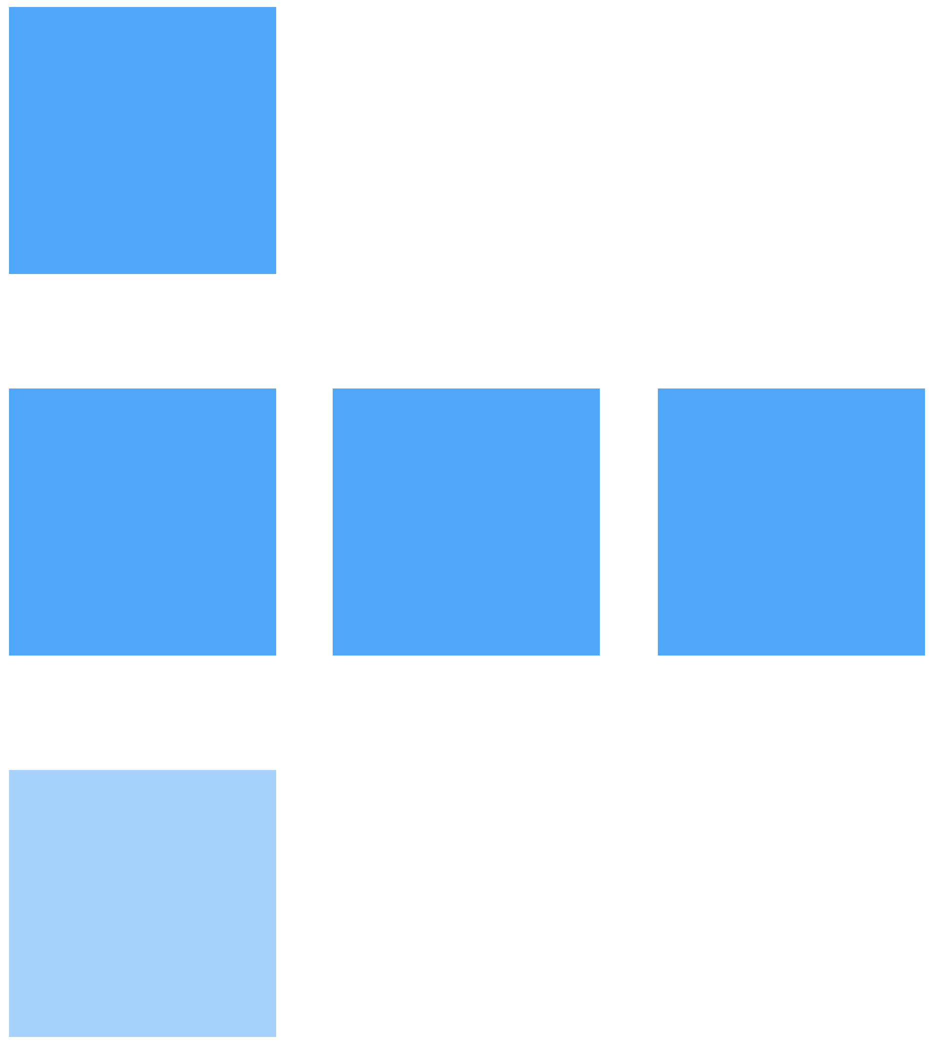
Iterative Conditions





"Accept this square if the average surface **of the last accepted shapes** is less then 10"

Quantifiers



singleton (a)

… looping (a+)

… optional (a?)

```
p.times(n)   p.oneOrMore()   p.optional()
```

Contiguity (1/2)

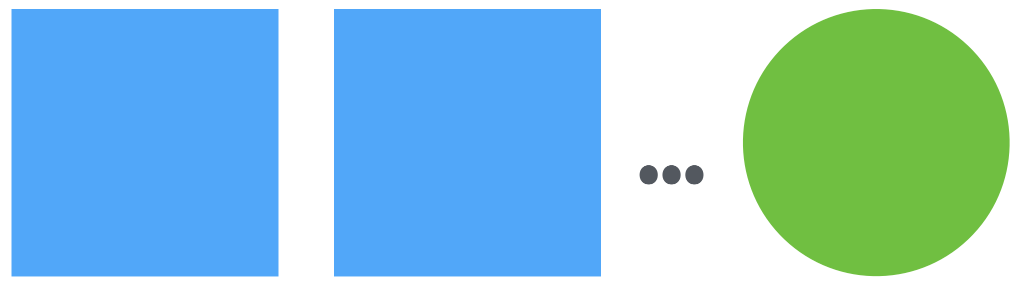
- **Strict Contiguity**: Expects all matching events to appear strictly one after the other, without any non-matching events in-between.
- **Relaxed Contiguity**: Ignores non-matching events appearing in-between the matching ones.
- **Non-Deterministic Relaxed Contiguity**: Further relaxes contiguity, allowing additional matches that ignore some matching events.

Contiguity (2/2)

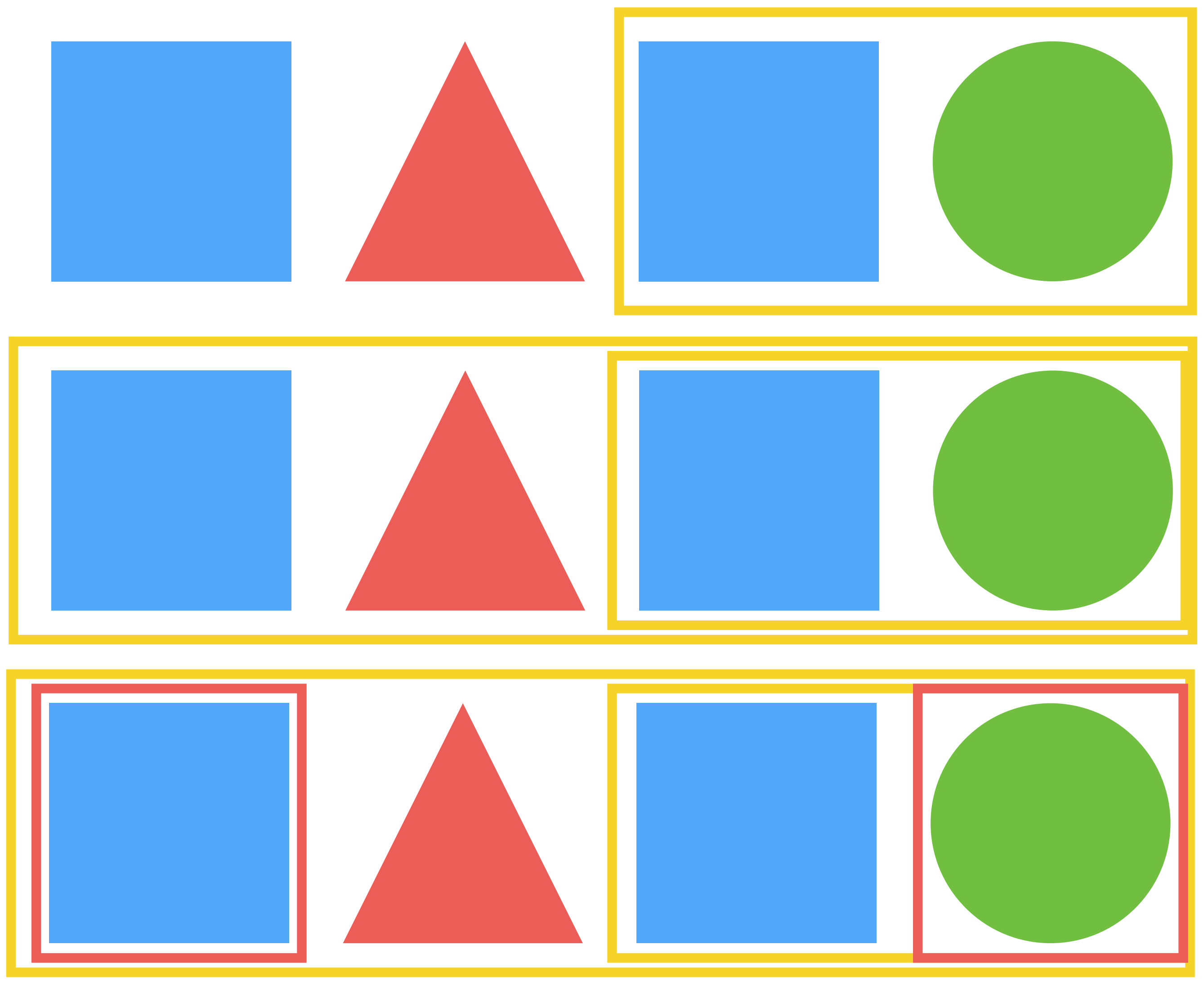
Pattern: "*a+ b*", Input: "*a1 - c - a2 - b*"

- **Strict Contiguity**: {*a2 - b*} – the *c* after *a1* causes *a1* to be discarded.
- **Relaxed Contiguity**: {*a1 - b*} and {*a1 - a2 - b*} – *c* is ignored.
- **Non-Deterministic Relaxed Contiguity**: {*a1 - b*}, {*a2 - b*}, and {*a1 - a2 - b*}.

Contiguity on Loops



`square.oneOrMore().?.circle`



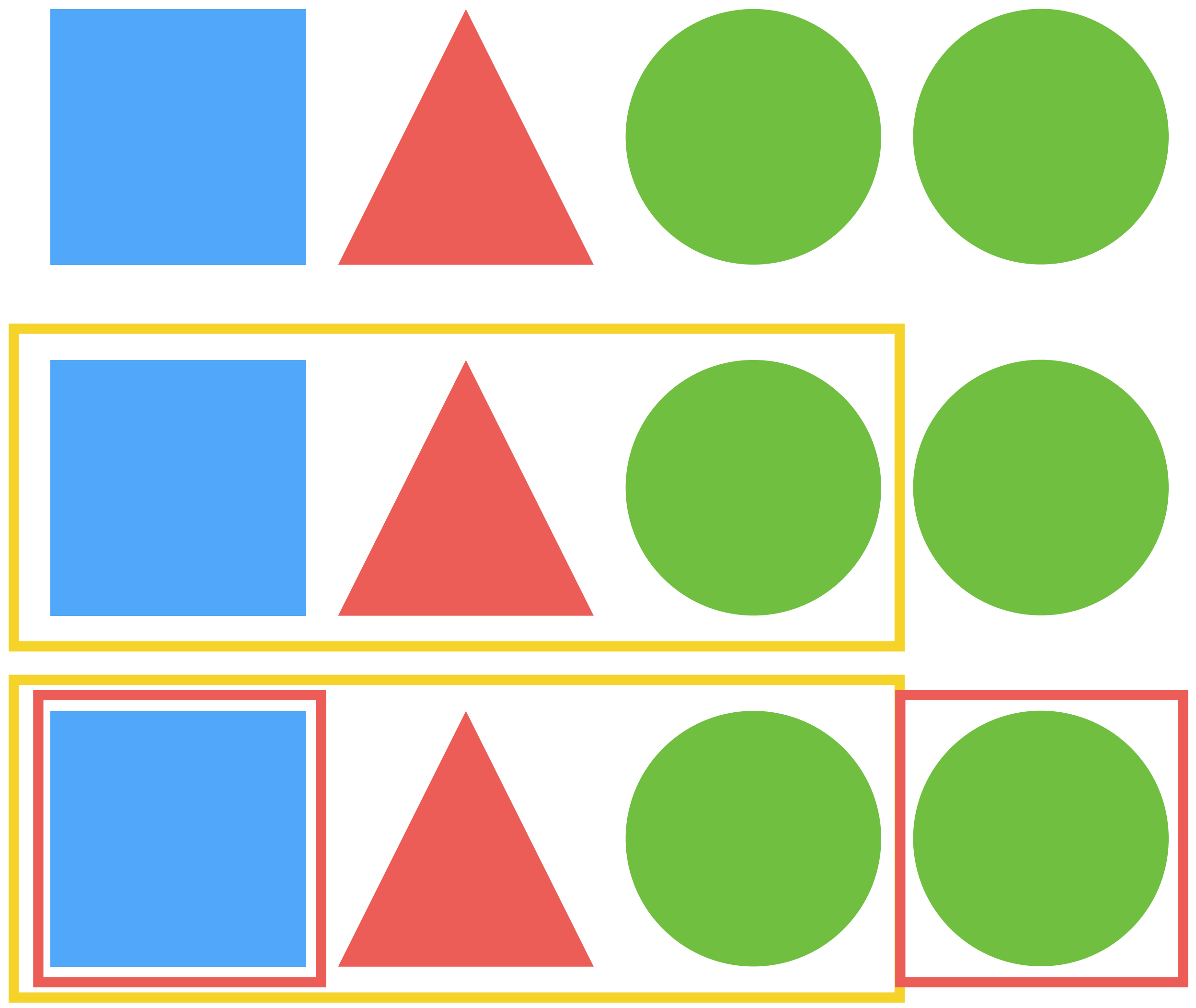
strict contiguity
`.consecutive()`

relaxed contiguity

non-deterministic contiguity
`.allowCombinations()`

Combining Patterns

```
square.oneOrMore().?.?.circle
```



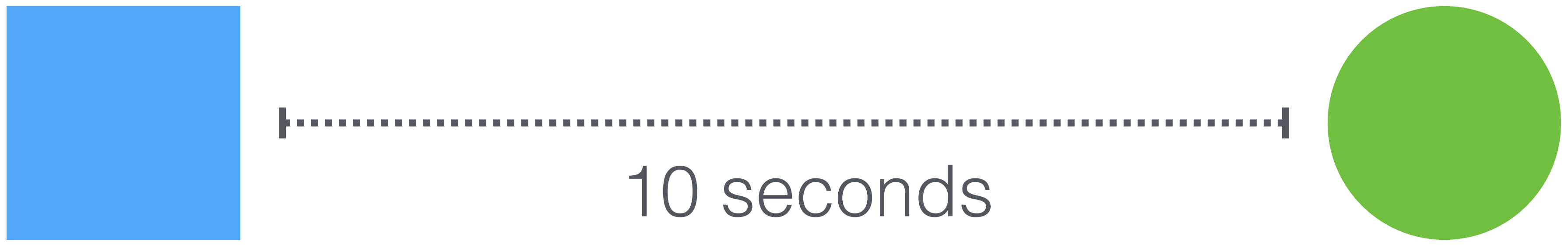
strict contiguity
`.next()`

relaxed contiguity
`.followedBy()`

NOT

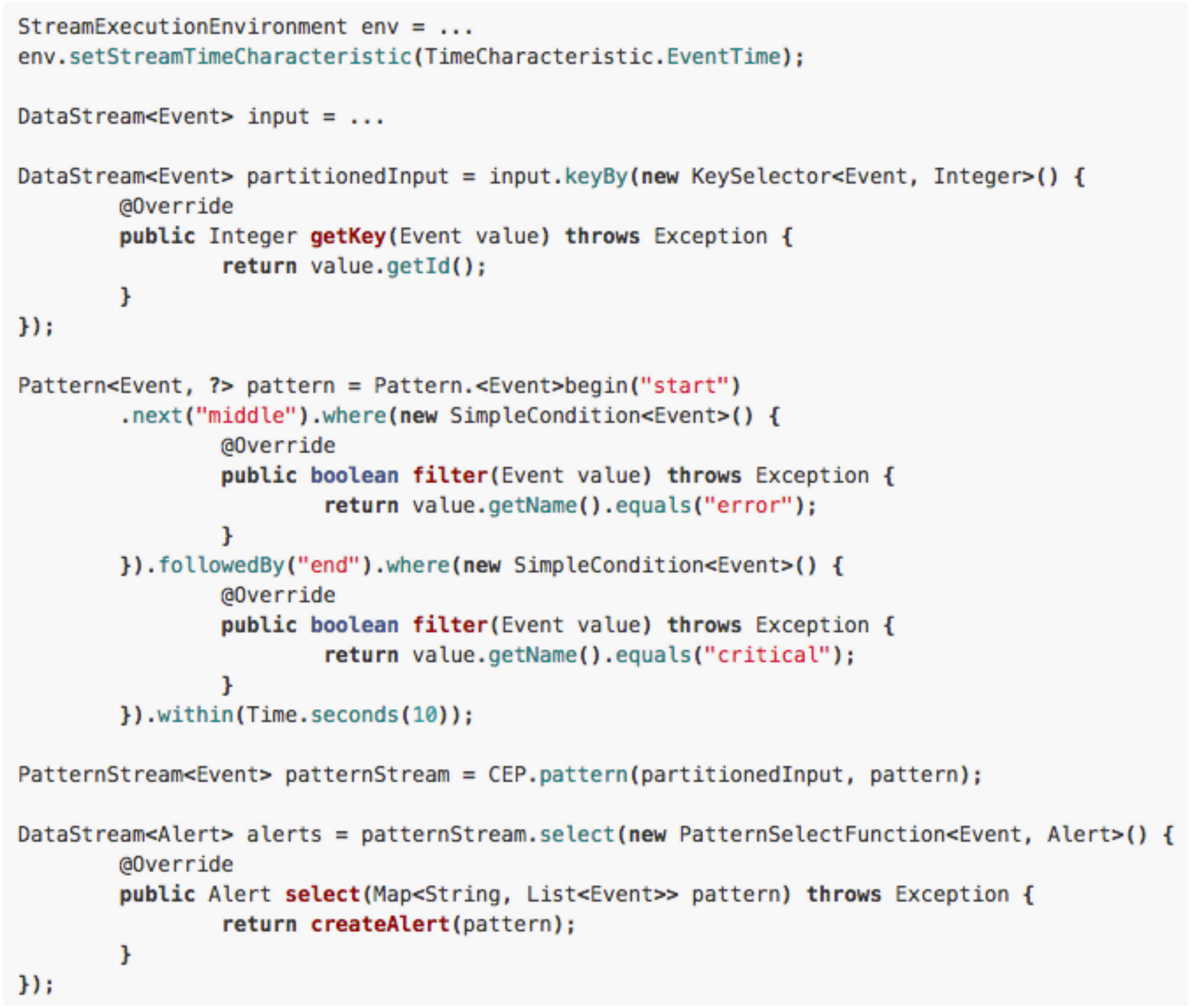
non-deterministic contiguity
`.followedByAny()`

Temporal Constraints



```
square
   .followedBy(circle)
   .within(Time.seconds(10))
```

Example



```java
StreamExecutionEnvironment env = ...
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

DataStream<Event> input = ...

DataStream<Event> partitionedInput = input.keyBy(new KeySelector<Event, Integer>() {
	@Override
	public Integer getKey(Event value) throws Exception {
		return value.getId();
	}
});

Pattern<Event, ?> pattern = Pattern.<Event>begin("start")
	.next("middle").where(new SimpleCondition<Event>() {
		@Override
		public boolean filter(Event value) throws Exception {
			return value.getName().equals("error");
		}
	}).followedBy("end").where(new SimpleCondition<Event>() {
		@Override
		public boolean filter(Event value) throws Exception {
			return value.getName().equals("critical");
		}
	}).within(Time.seconds(10));

PatternStream<Event> patternStream = CEP.pattern(partitionedInput, pattern);

DataStream<Alert> alerts = patternStream.select(new PatternSelectFunction<Event, Alert>() {
	@Override
	public Alert select(Map<String, List<Event>> pattern) throws Exception {
		return createAlert(pattern);
	}
});
```

References

- More on tables and streams
- CEP Library