Complex Event Processing with Flink The state of FlinkCEP

Kostas Kloudas
@kkloudas

dataArtisans

Flink Forward Berlin
SEPTEMBER 12, 2017

dataArtisans

dA
PLATFORM

Original creators of Apache Flink®

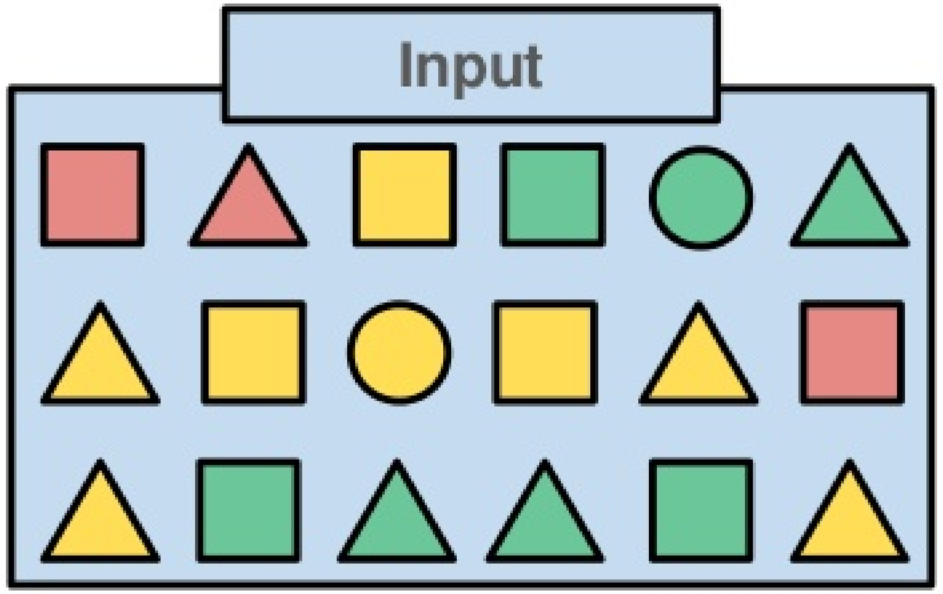
Providers of
dA Platform 2, including
open source Apache Flink +
dA Application Manager

What is CEP?

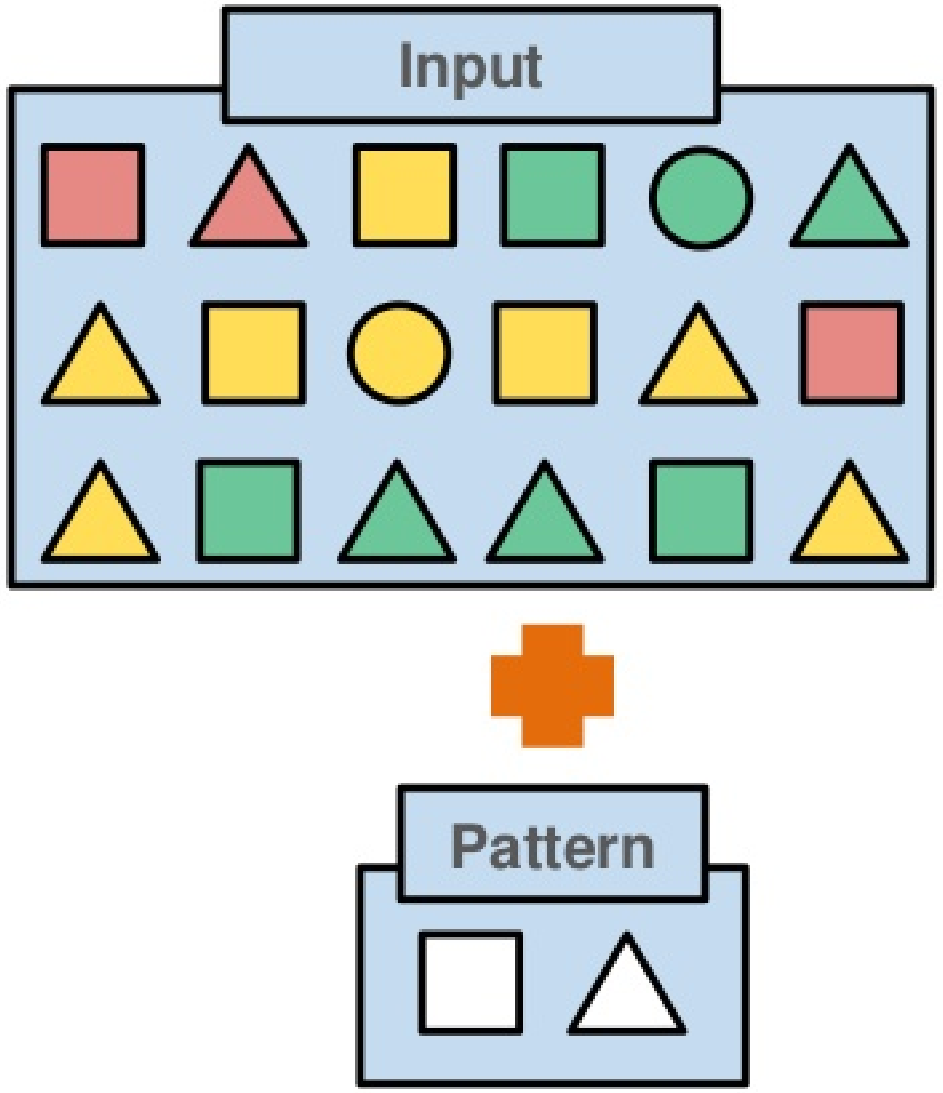
CEP: Complex Event Processing

- Detecting event patterns
- Over continuous streams of events
- Often arriving out-of-order

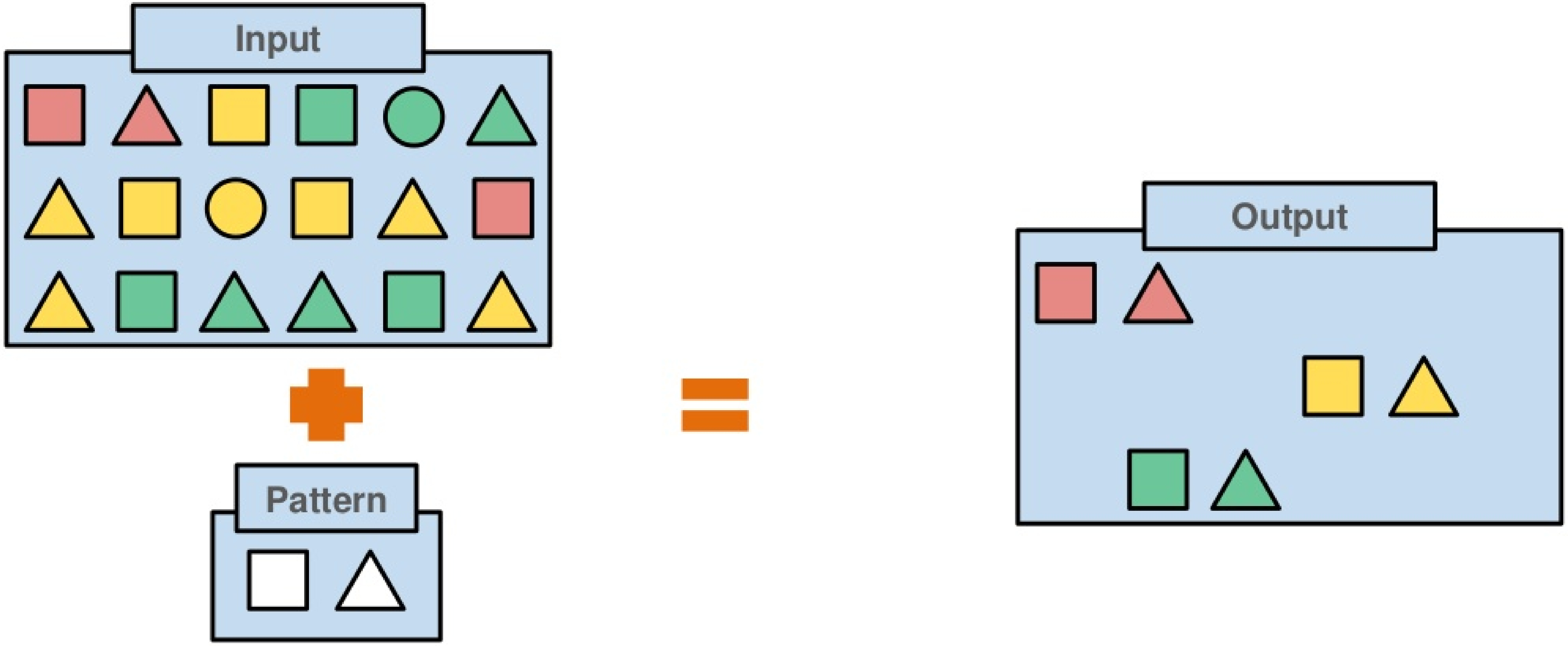
CEP: Complex Event Processing



CEP: Complex Event Processing



CEP: Complex Event Processing

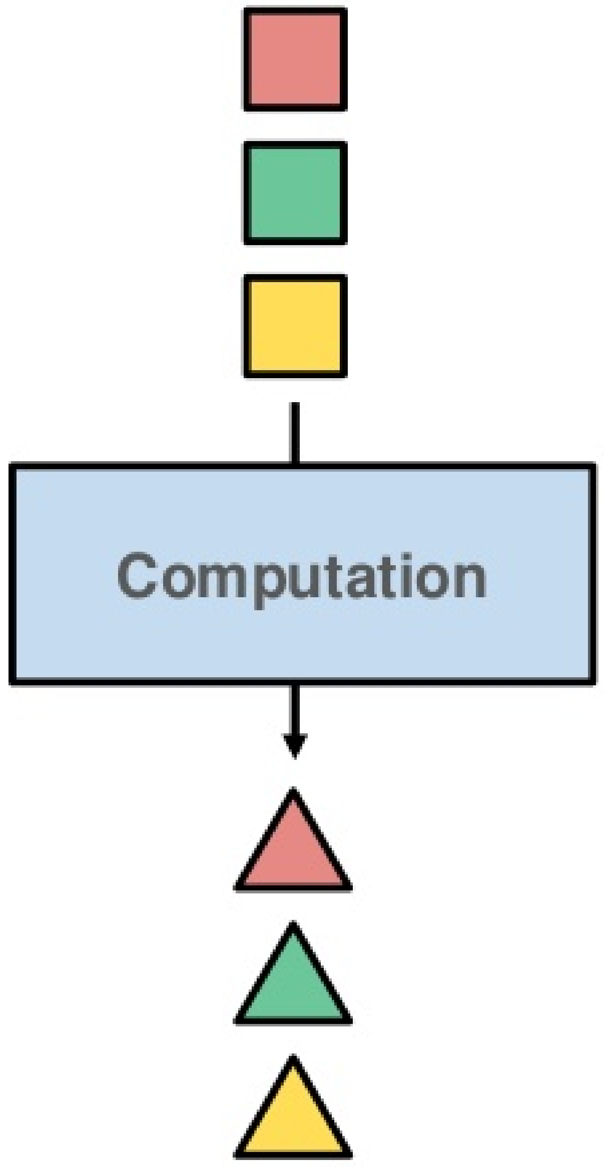


CEP: use-cases

- IoT
- Infrastructure Monitoring and Alarms
- Intrusion detection
- Inventory Management
- Click Stream Analysis
- Trend detection in financial sector
- *...yours?*

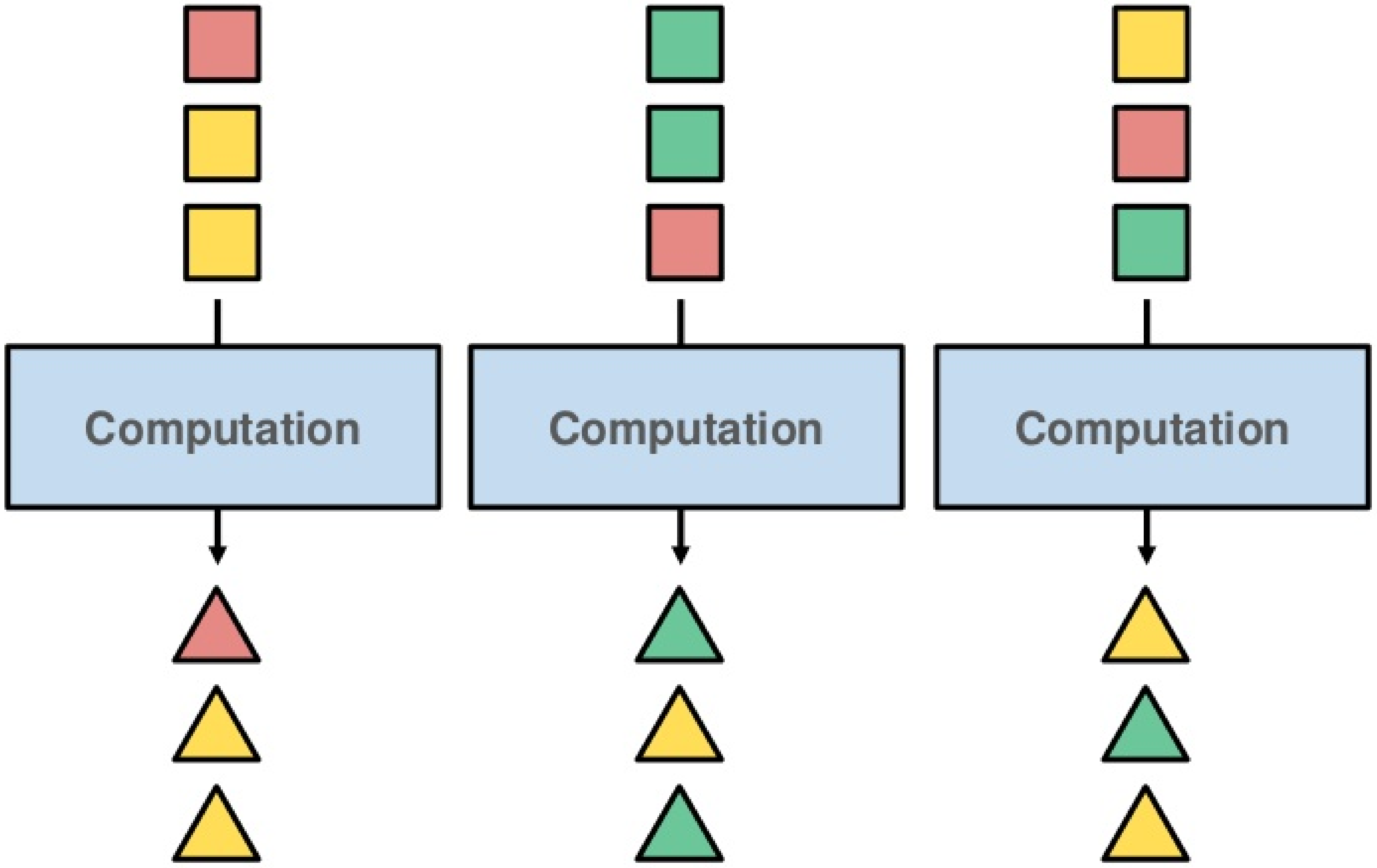
What is Stream Processing?

Stream Processing



Computations on
never-ending
“streams” of events

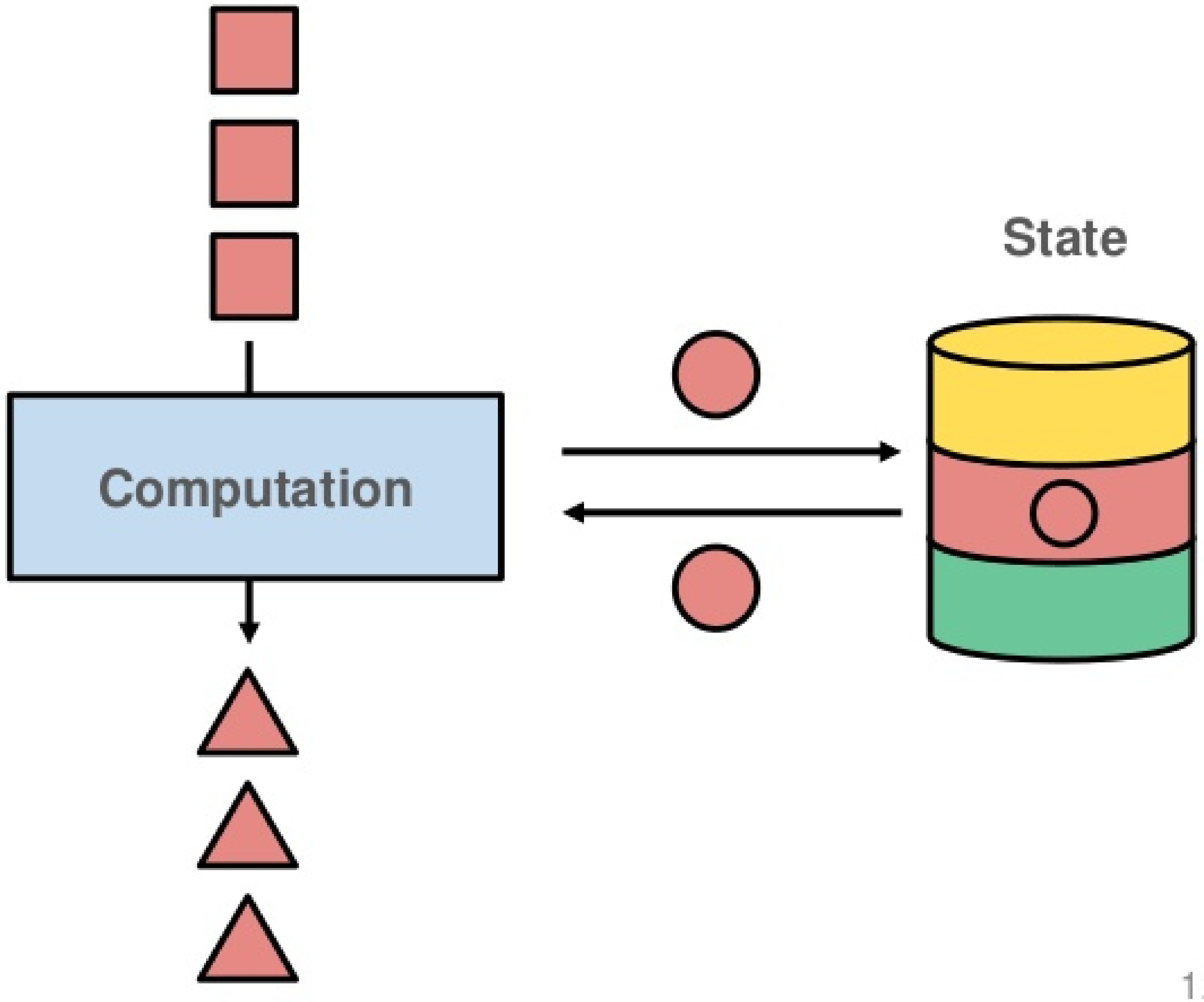
Distributed Stream Processing



Computation spread across many machines

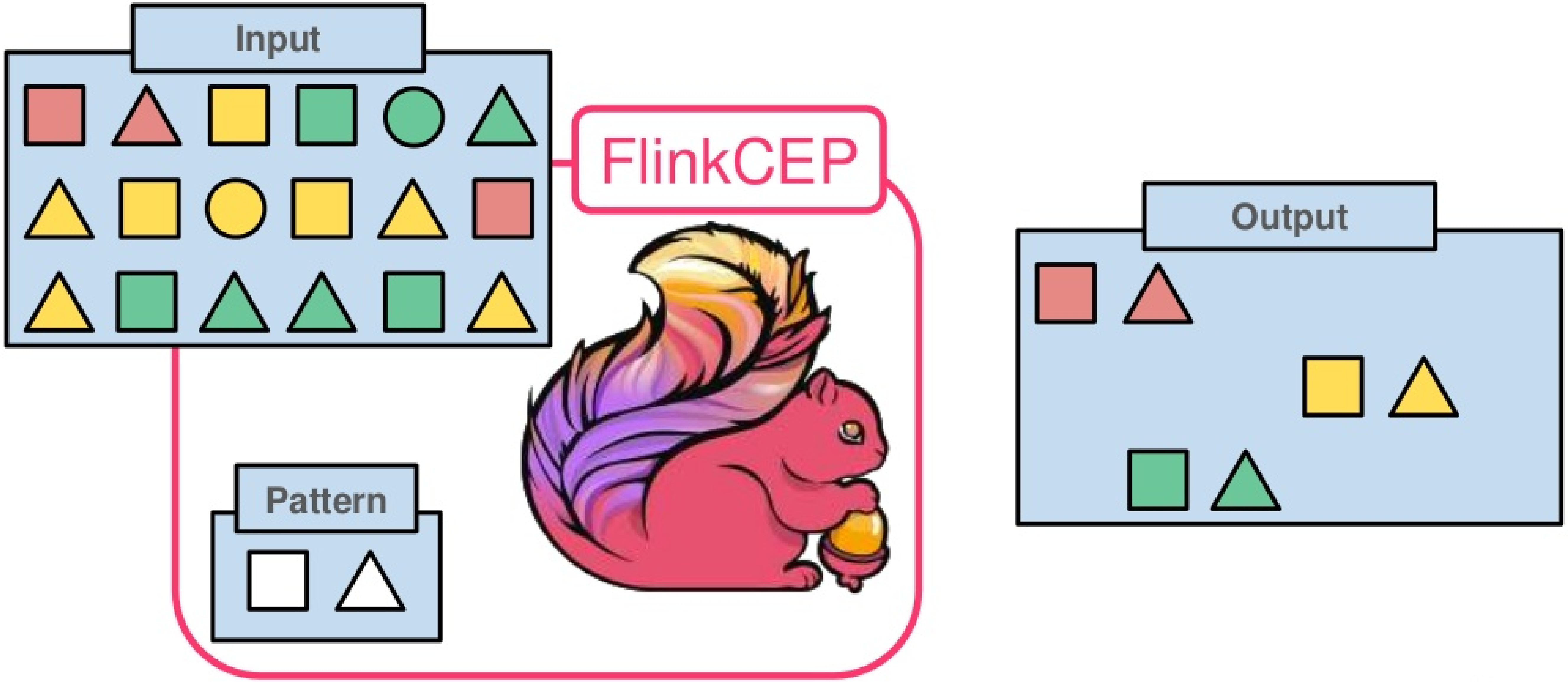
Stateful Stream Processing

Result depends on history of stream



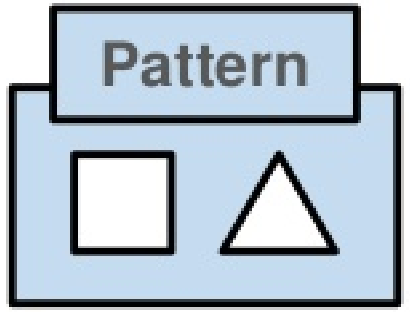
Stream Processors are a natural fit for **CEP**

FlinkCEP



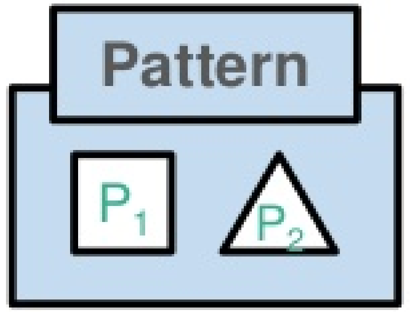
What does FlinkCEP offer?

Pattern Definition



Pattern Definition

- Composed of Individual Patterns
  - □ P_1(shape == rectangle)
  - △ P_2(shape == triangle)

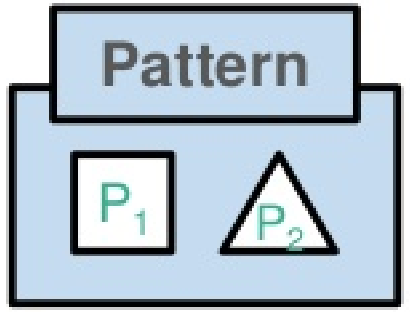


Pattern Definition

- Composed of Individual Patterns

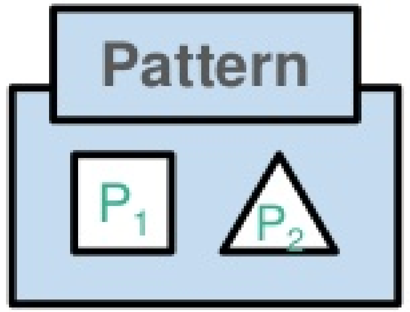
  □ P_1(shape == rectangle)

  △ P_2(shape == triangle)



- Combined by Contiguity Conditions
  - ...later

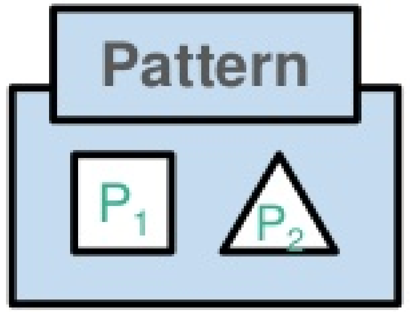
FlinkCEP Individual Patterns



- Unique Name
- Quantifiers : how many times ?
  - Looping oneOrMore(), times(from, to), greedy()
  - Optional optional()
- Condition : which elements to accept ?
  - Simple *e.g* shape == rectangle
  - Iterative *e.g* rectangle.surface < triangle.surface
  - Stop until(cond.)

FlinkCEP Complex Patterns

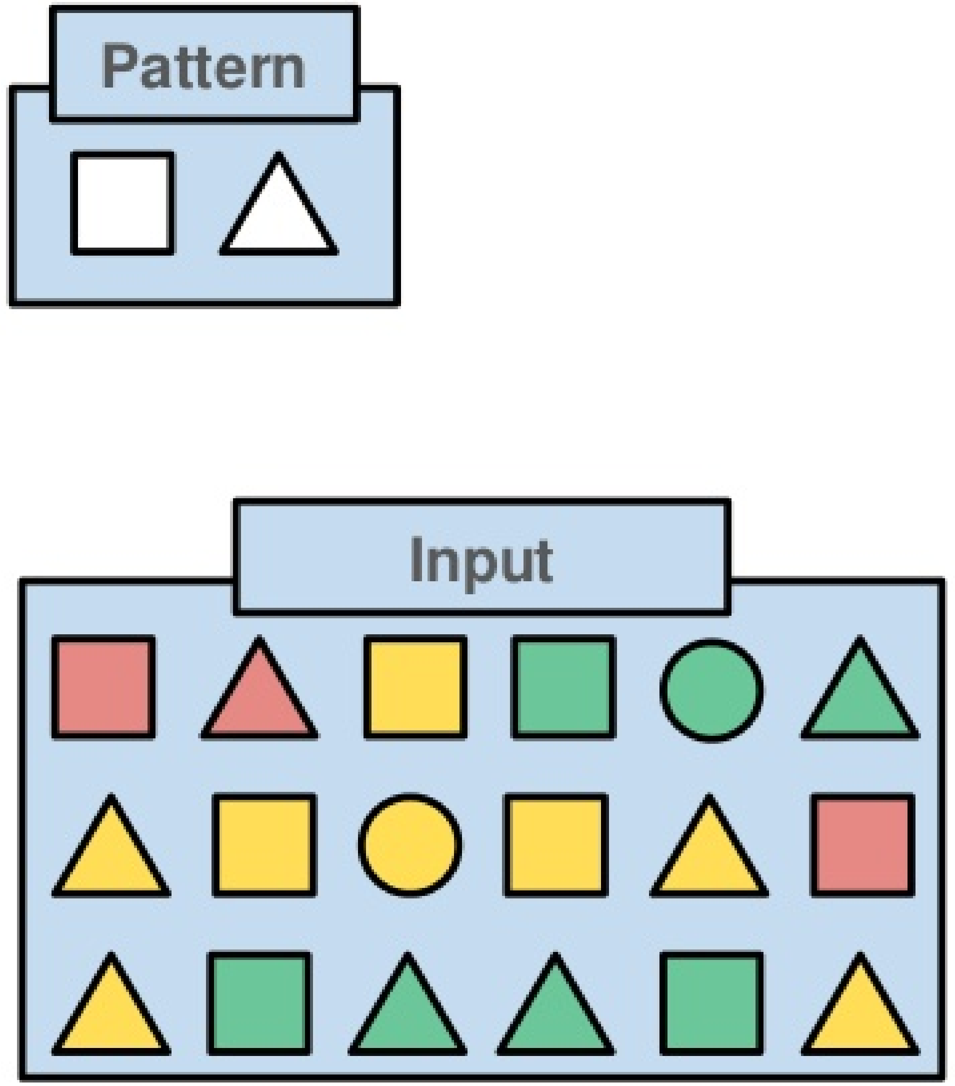
- Combine Individual Patterns



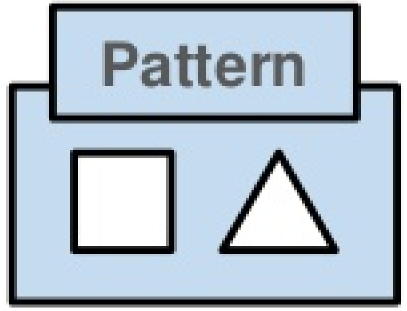
- Contiguity Conditions
  - *how to select relevant events* given an input mixing relevant and irrelevant events

- Time Constraints (event/processing time)
  - *within(time) e.g.* all events have to come within 24h

FlinkCEP Contiguity Conditions

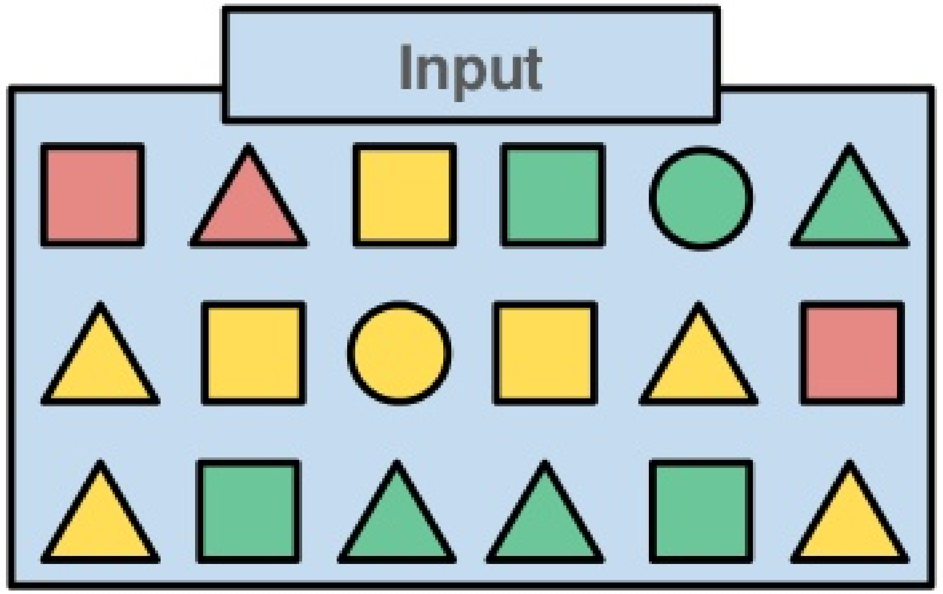


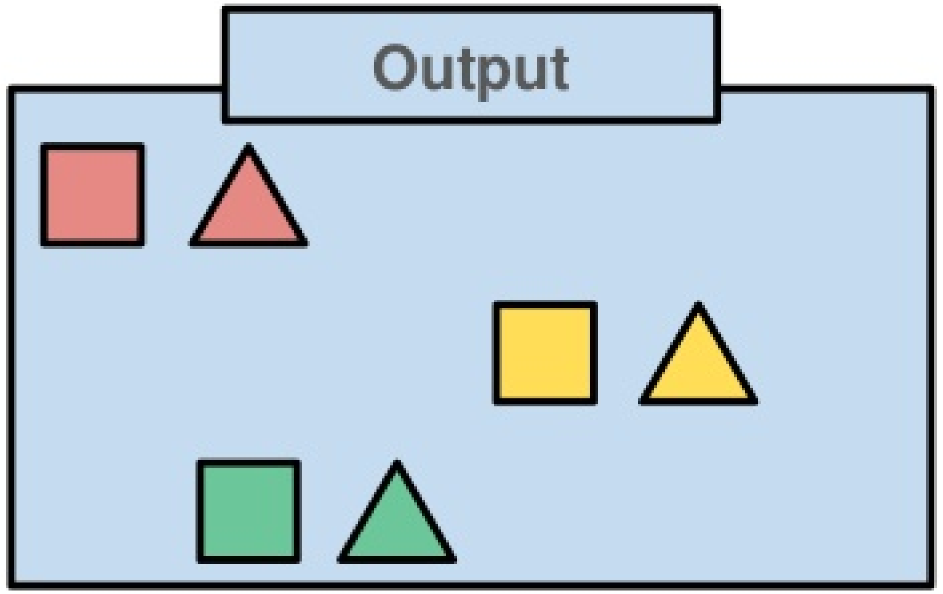
FlinkCEP Contiguity Conditions



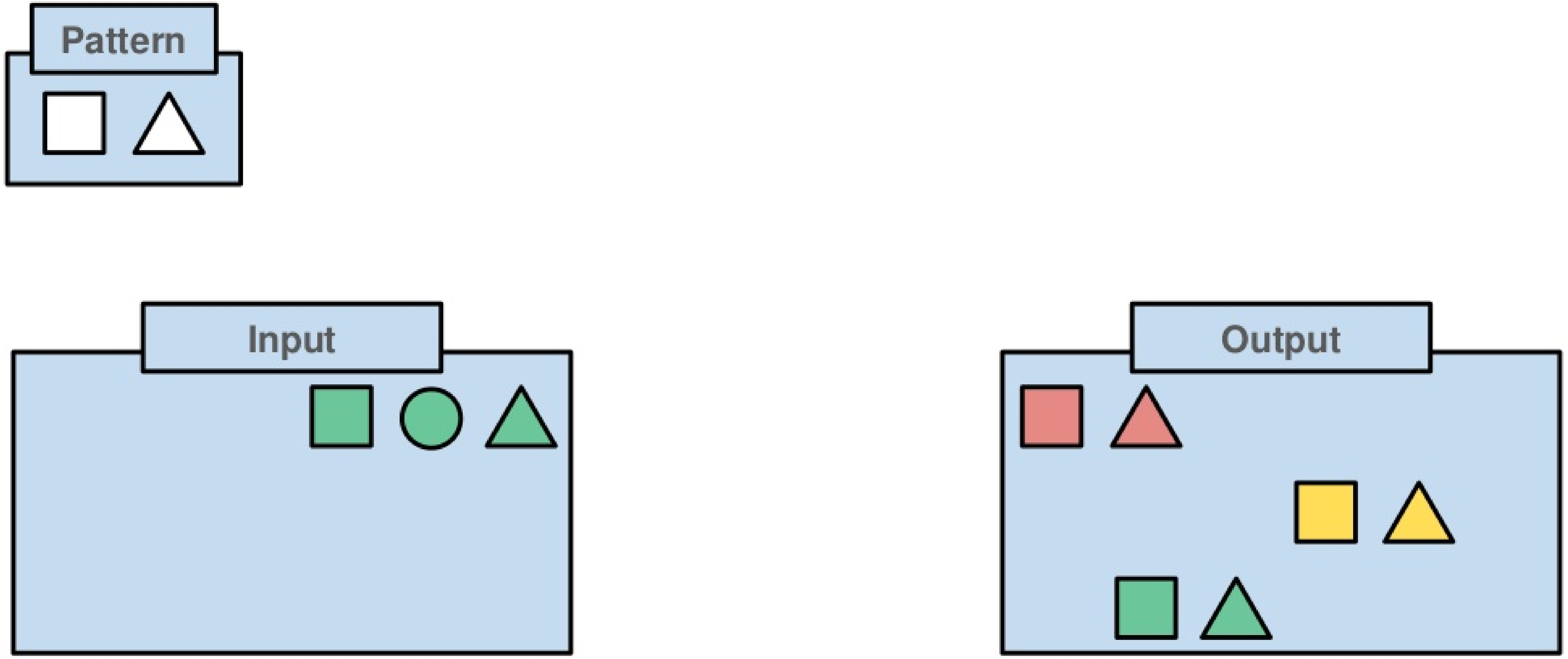
Strict Contiguity

- matching events strictly follow each other

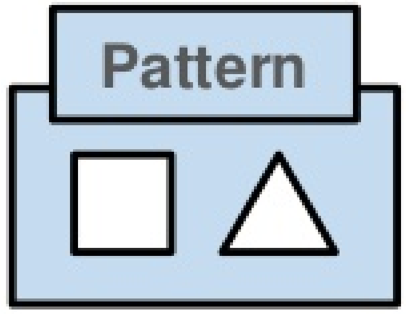




FlinkCEP Contiguity Conditions

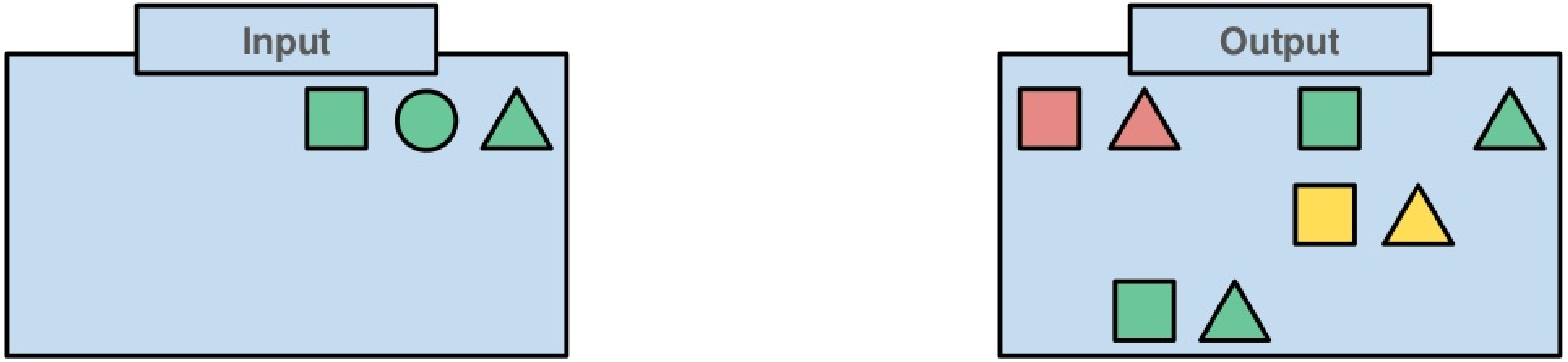


FlinkCEP Contiguity Conditions

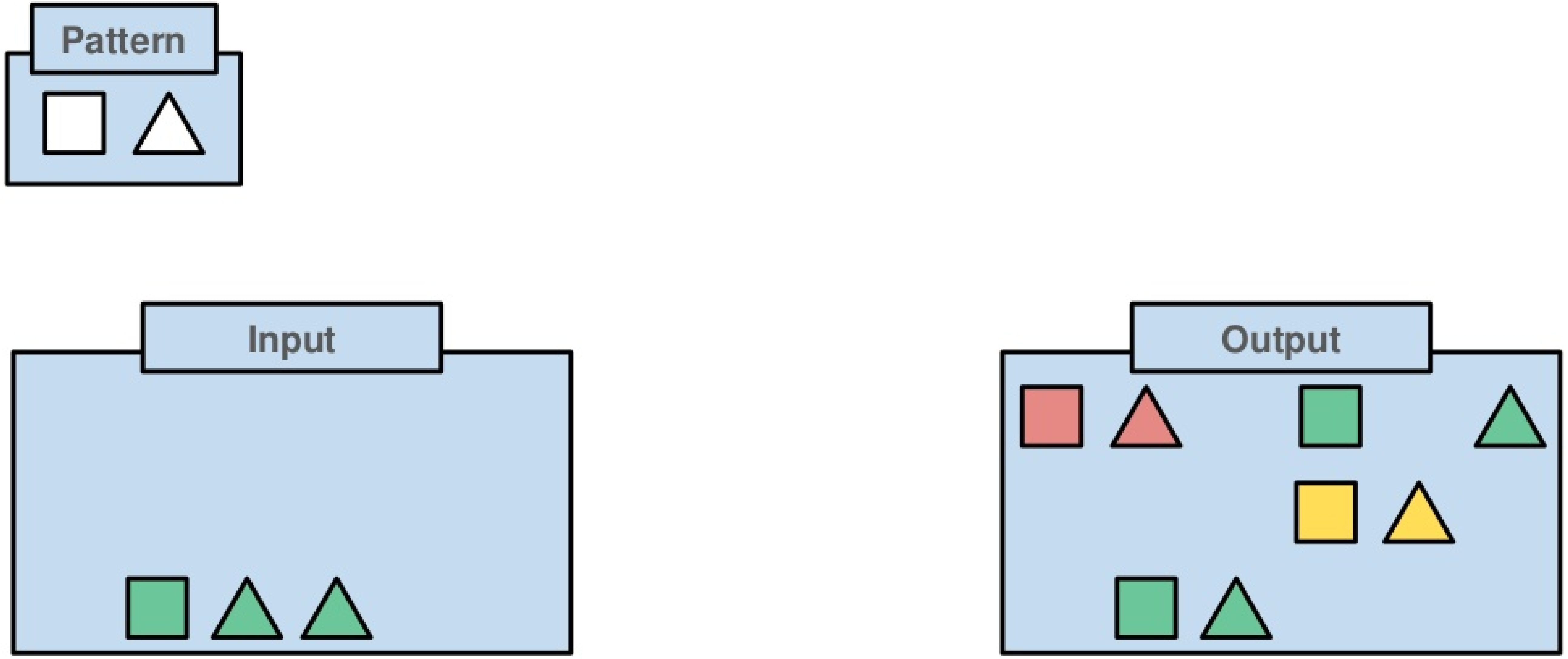


Relaxed Contiguity

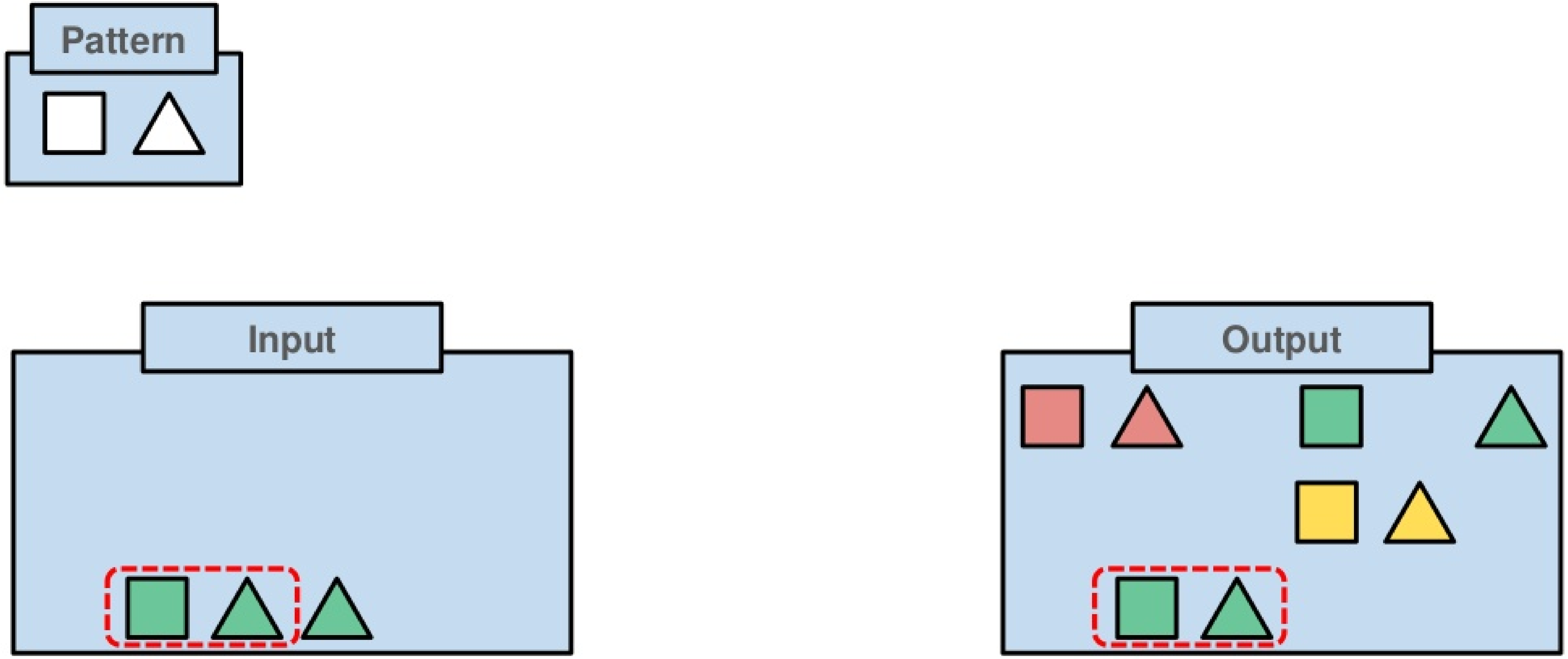
- non-matching events to simply be ignored



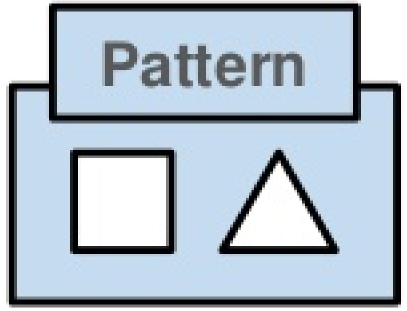
FlinkCEP Contiguity Conditions



FlinkCEP Contiguity Conditions

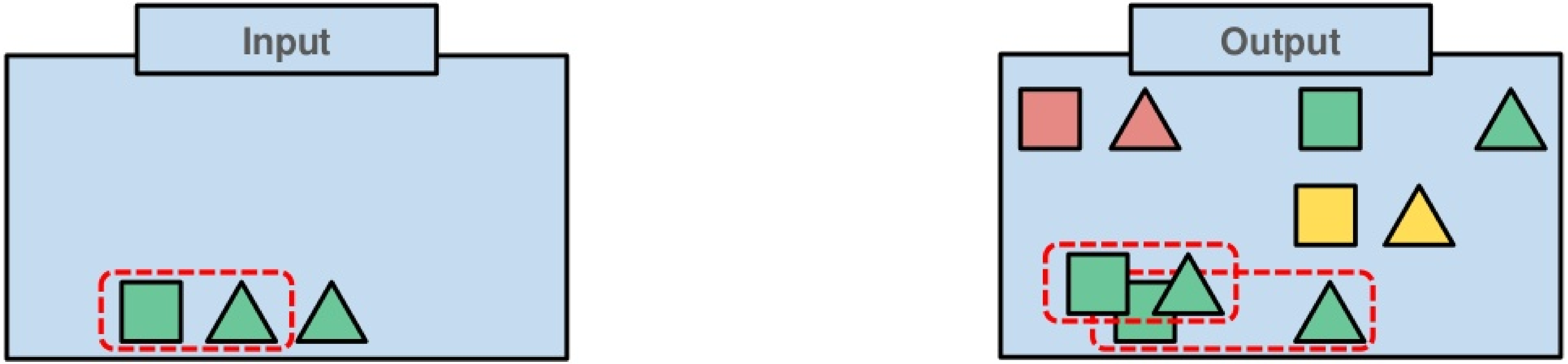


FlinkCEP Contiguity Conditions

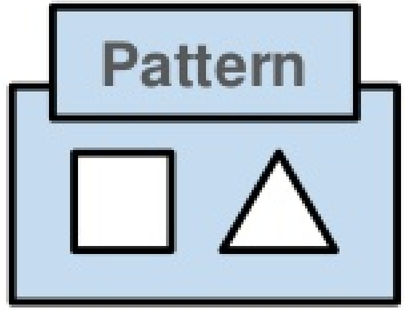


Non-Deterministic Relaxed Contiguity

- allows non-deterministic actions on relevant events

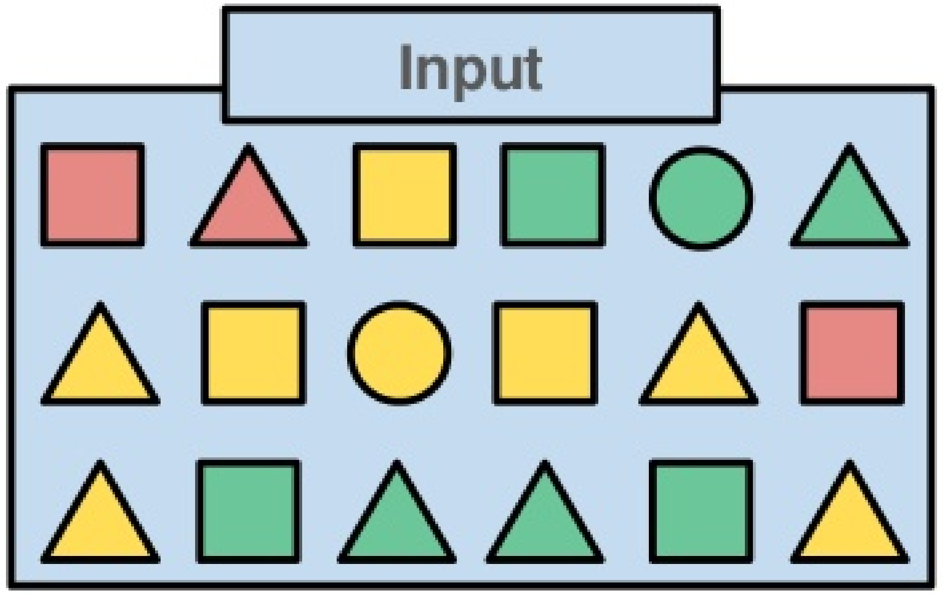


FlinkCEP Contiguity Conditions



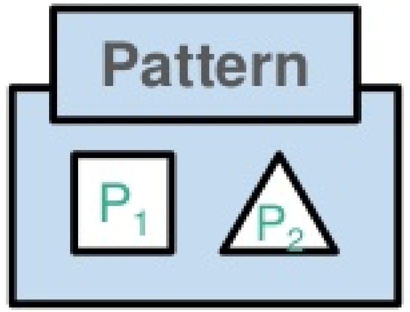
NOT patterns:

- for strict and relaxed contiguity
- for cases where an event should invalidate a match



FlinkCEP Grouping Patterns

- Define Individual Patterns



- Combine them into Complex Patterns

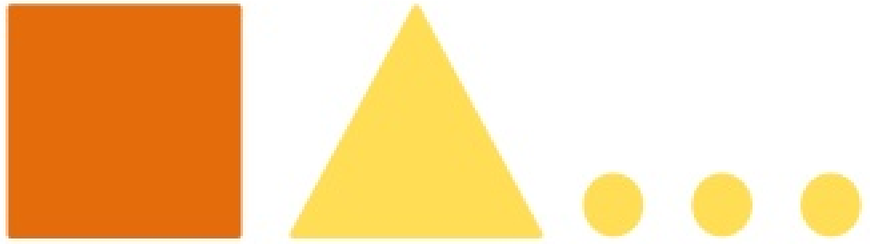
- Can we combine …Complex Patterns ?

FlinkCEP Grouping Patterns

Grouping Patterns are for CEP what parenthesis are for mathematical expressions.

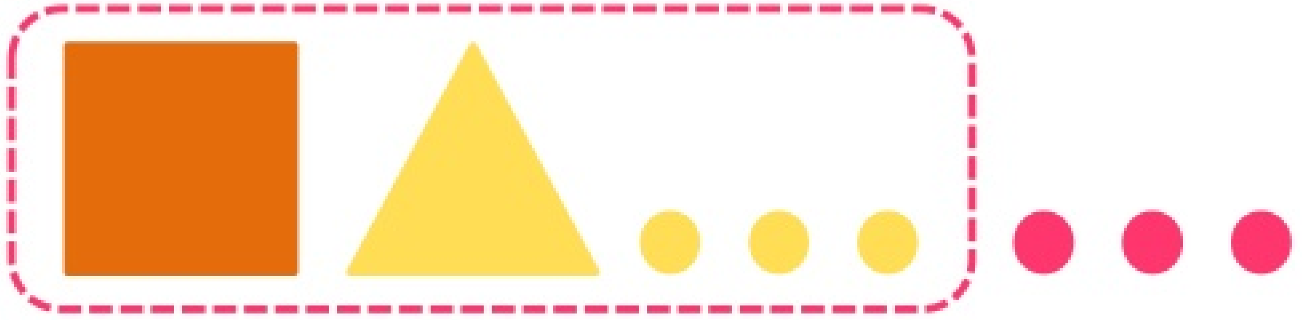
FlinkCEP Grouping Patterns

Grouping Patterns are for CEP what parenthesis are for mathematical expressions.



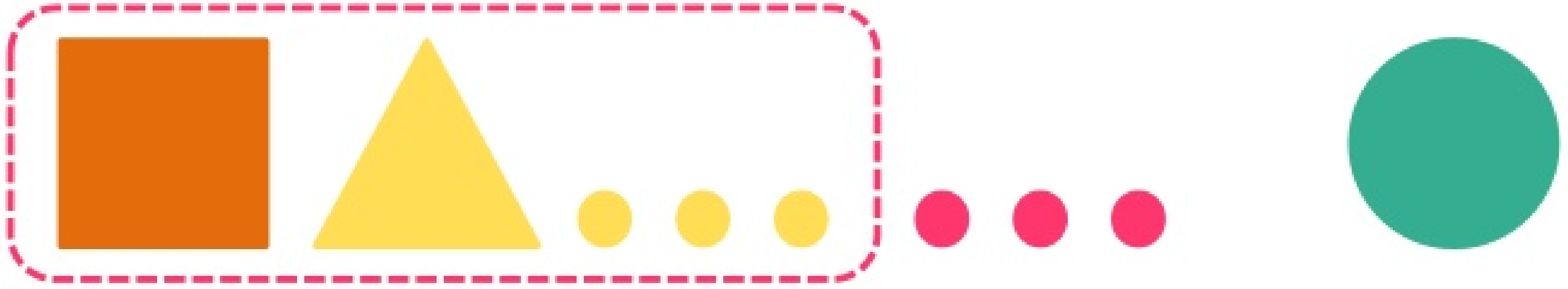
FlinkCEP Grouping Patterns

Grouping Patterns are for CEP what parenthesis are for mathematical expressions.



FlinkCEP Grouping Patterns

Grouping Patterns are for CEP what parenthesis are for mathematical expressions.



FlinkCEP Summary

- Quantifiers oneOrMore(), times(), optional()
- Conditions Simple, Iterative, Stop
- Time Constraints Event and Processing time
- Contiguity Constraints

  Strict, relaxed, non-deterministic relaxed, NOT
- Grouping Patterns

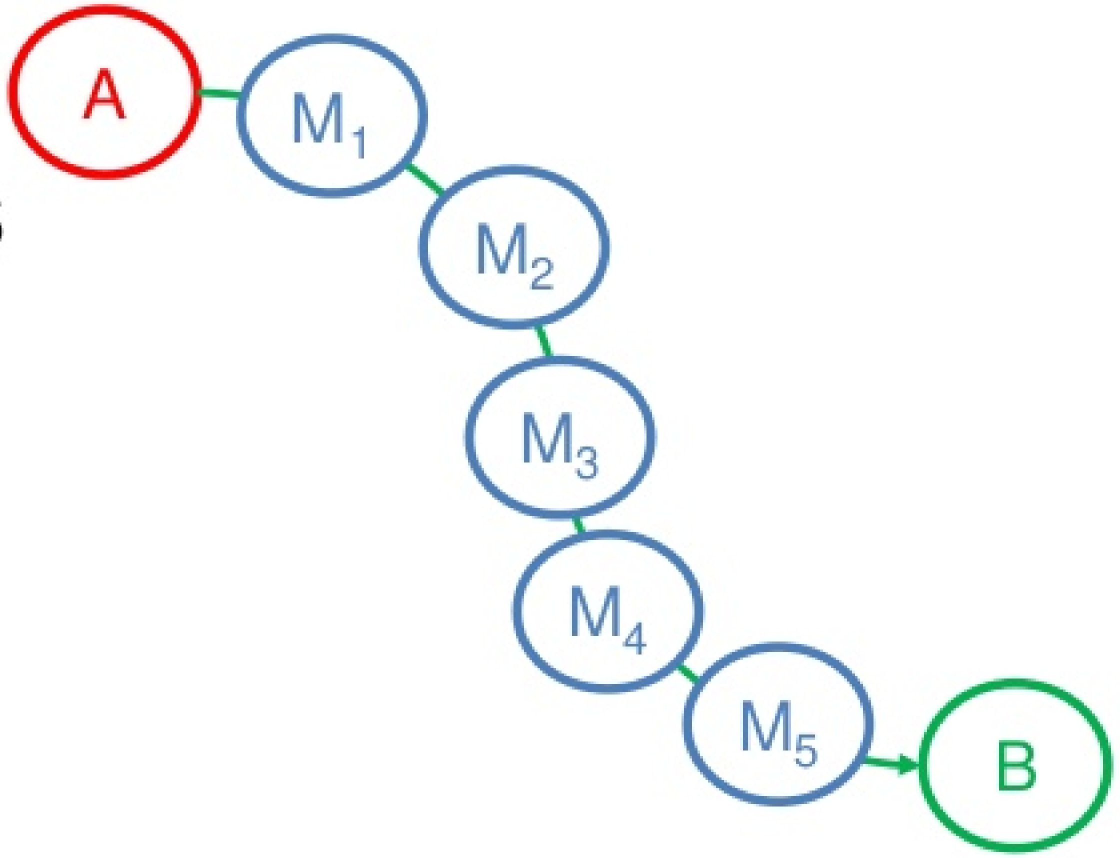
FlinkCEP Integration with SQL

- Flink already supports **SQL**:
  - match_recognize clause in SQL:2016
  - ongoing effort with a lot of interest from the community

Example

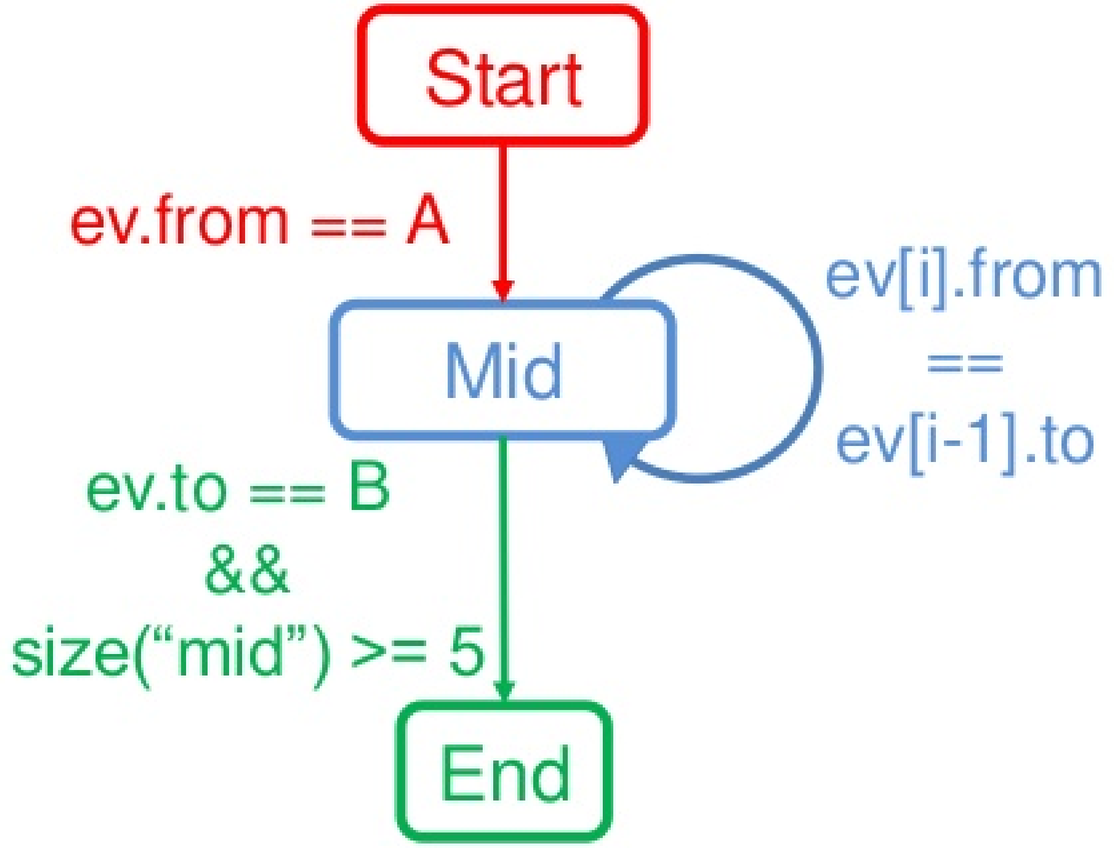
Running Example: retailer

- Trace all shipments which:
  - start at location A
  - have at least 5 stops
  - end at location B
  - within the last 24h



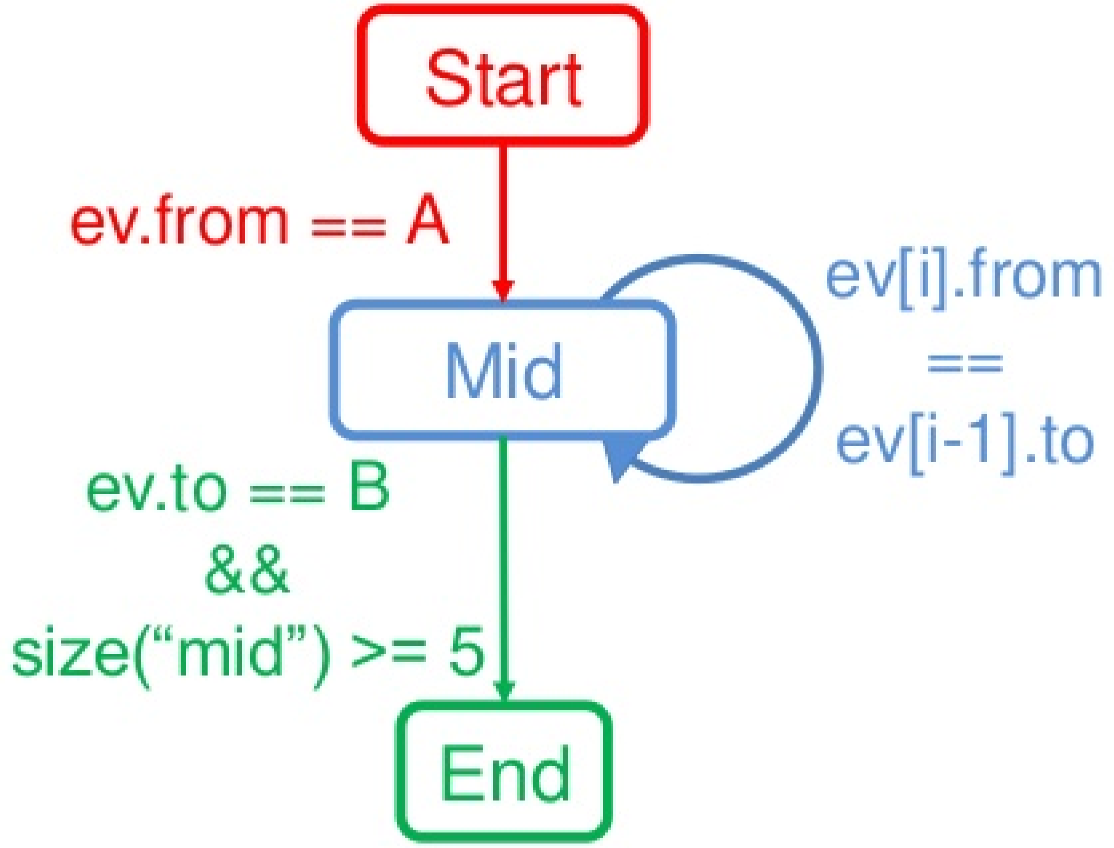
Observation A Individual Patterns

- Trace all shipments which:
  - start at location A
  - have at least 5 stops
  - end at location B
  - within the last 24h



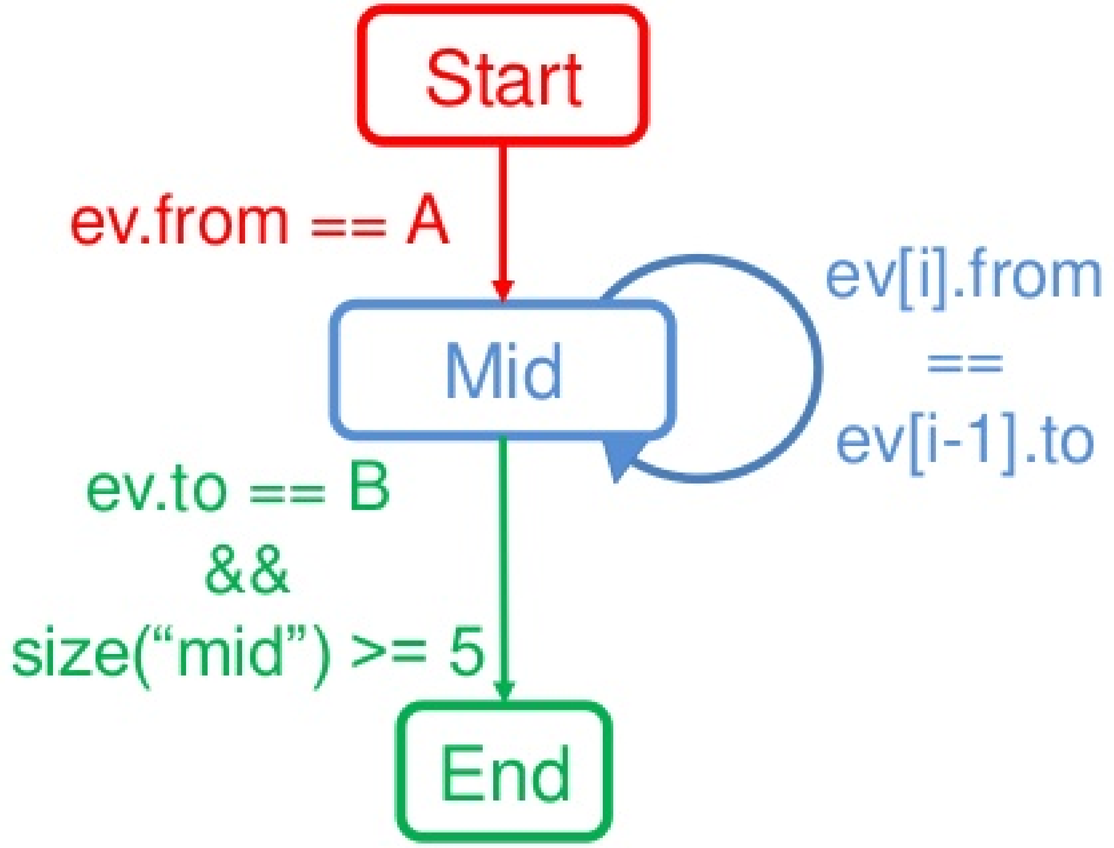
Observation B Quantifiers

- Start/End: single event
- Middle: multiple events
  - .oneOrMore()



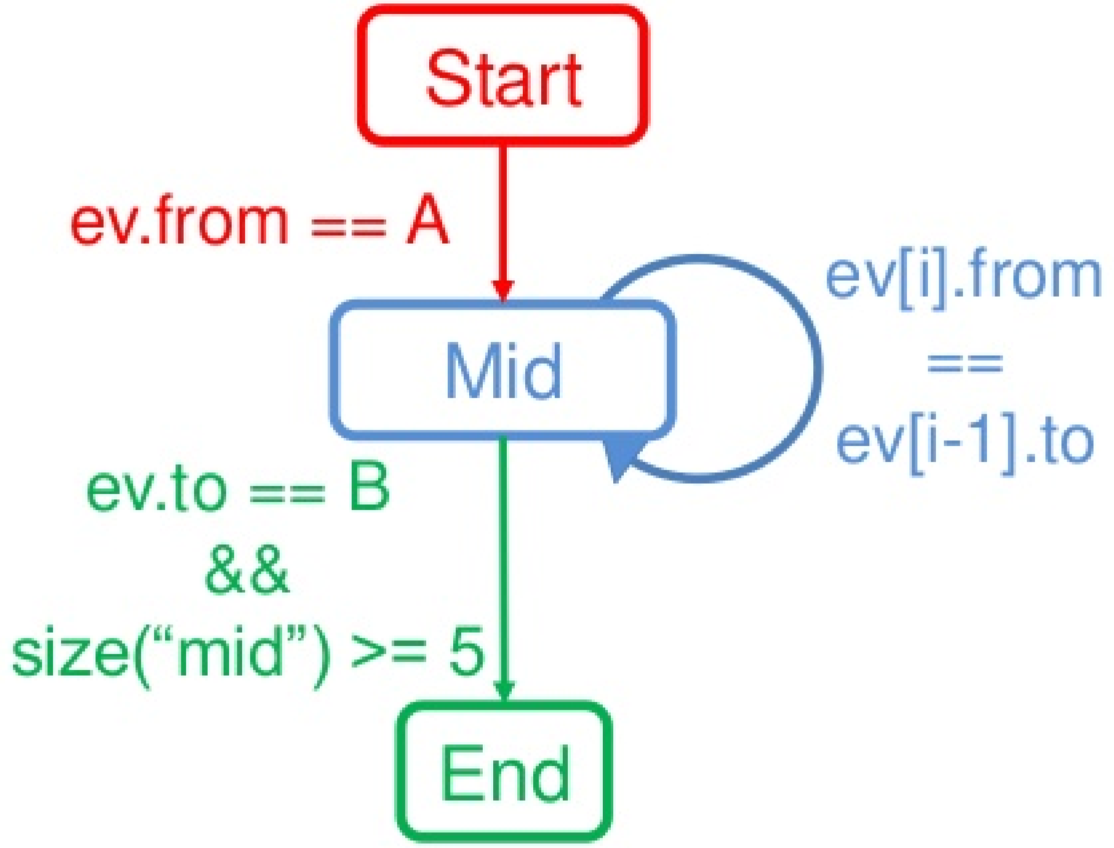
Observation C Conditions

- Start -> Simple
  - properties of the event
- Middle/End -> Iterative
  - Depend on previous events



Observation D Time Constraints

- Trace all shipments which:
  - start at location A
  - have at least 5 stops
  - end at location B
  - within the last 24h



Observation E Contiguity

- We opt for relaxed continuity

Running Example Individual Patterns

```
Pattern<Event, ?> pattern = Pattern
    .<Event>begin("start")
        .where(mySimpleCondition)
    .followedBy ("middle")
        .where(myIterativeCondition1)
        .oneOrMore()
    .followedBy ("end”)
        .where(myIterativeCondition2)
    .within(Time.hours(24))
```

Start

Middle

End

Running Example Quantifiers

```
Pattern<Event, ?> pattern = Pattern
          .<Event>begin("start")
                    .where(mySimpleCondition)
          .followedBy ("middle")
                    .where(myIterativeCondition1)
                    .oneOrMore()
          .followedBy ("end")
                    .where(myIterativeCondition2)
          .within(Time.hours(24))
```

Start

Middle

End

Running Example Conditions

```
Pattern<Event, ?> pattern = Pattern
    .<Event>begin("start")
        .where(mySimpleCondition)           Start
    .followedBy ("middle")
        .where(myIterativeCondition1)       Middle
        .oneOrMore()
    .followedBy ("end")                     End
        .where(myIterativeCondition2)
    .within(Time.hours(24))
```

Running Example Time Constraint

```
Pattern<Event, ?> pattern = Pattern
      .<Event>begin("start")
            .where(mySimpleCondition)              Start
      .followedBy ("middle")
            .where(myIterativeCondition1)          Middle
            .oneOrMore()
      .followedBy ("end”)
             .where(myIterativeCondition2)         End
      .within(Time.hours(24))
```

Running Example Pattern Integration

```
Pattern<Event, ?> pattern = ...

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

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

Running Example Pattern Integration

```
Pattern<Event, ?> pattern = ...

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

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

Running Example Pattern Integration

```
Pattern<Event, ?> pattern = ...

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

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

Documentation

- FlinkCEP documentation:

FlinkCEP 1.3: https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/libs/cep.html
FlinkCEP 1.4: https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/libs/cep.html

Thank you!

@kkloudas
@ApacheFlink
@dataArtisans

dataArtisans

We are hiring!

data-artisans.com/careers