EasyFlinkCEP: Big Event Data Analytics for Everyone

Nikos Giatrakos

Athena Research & Innovation Center,
Technical University of Crete

ngiatrakos@athenarc.gr

ngiatrakos@softnet.tuc.gr

INFORE
Interactive Extreme-Scale Analytics and Forecasting

Outline

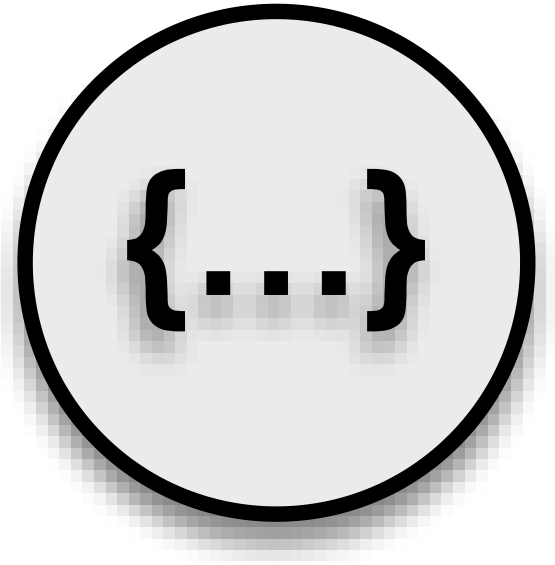
- Introduction (Streams & Apache Flink)
- CEP vs Traditional Streams
- FlinkCEP
  - Basics
  - Adoption barriers
- EasyFlinkCEP
  - EasyFlinkCEP Operator
  - EasyFlinkCEP Optimizer
- What we achieved
- Conclusions & Future Work

Outline

- Introduction (Streams & Apache Flink)
- CEP vs Traditional Streams
- FlinkCEP
  - Basics
  - Adoption barriers
- EasyFlinkCEP
  - EasyFlinkCEP Operator
  - EasyFlinkCEP Optimizer
- What we achieved
- Conclusions & Future Work

Streaming Model

- Tuple

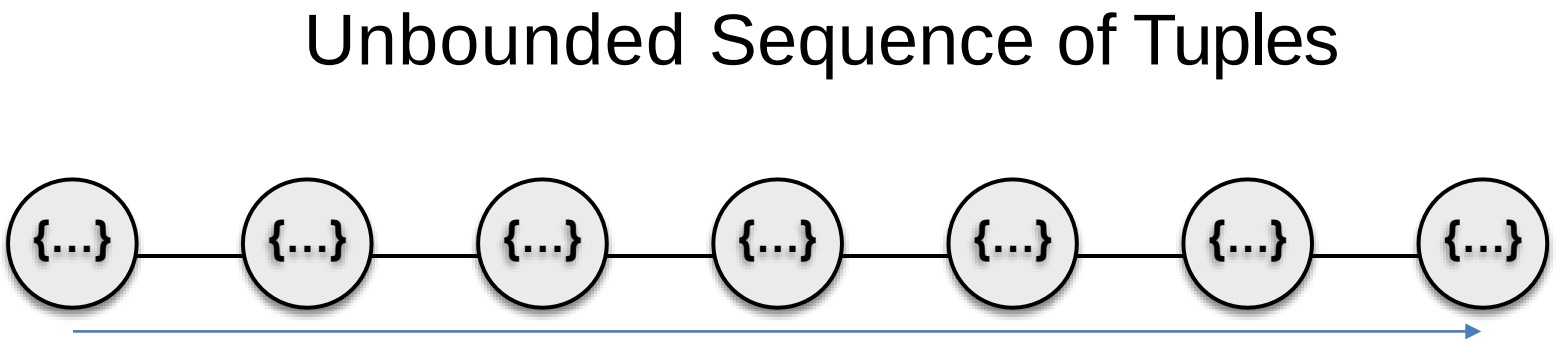


- Core Unit of Data
- Immutable Set of Key/Value Pairs
- Stock Market Example:

```
StockID Date        Time      Price  Volume
EURTRY, 01/15/2019,09:57:41, 29.21, 152
```

Streaming Model

- Stream



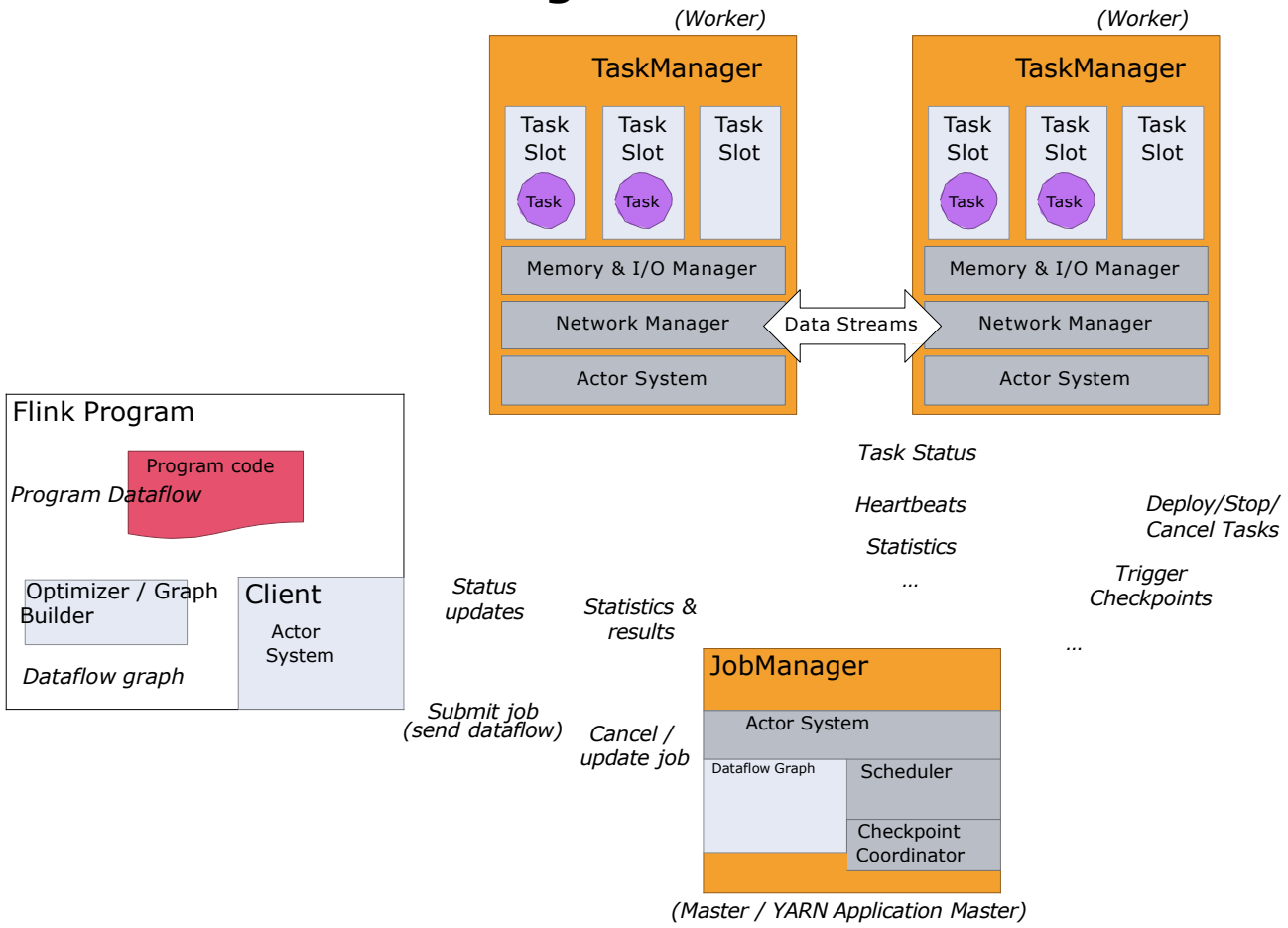

```
StockID Date       Time      Price  Volume

EURTRY, 01/15/2019,09:57:41,29.21, 152
EURTRY, 01/15/2019,09:58:45,29.2,  190
EURTRY, 01/15/2019,09:58:45,29.2,  177
…
…
```

Scalability – Real Time Analytics

https://en.wikipedia.org/wiki/Blue_Gene

Apache Flink – Basics

- Master/worker concept can be integrated into YARN
- The client (Flink Program) is an external process
- **Notable new! Task chaining**



Programming Model

- A DAG of streams applies transformations

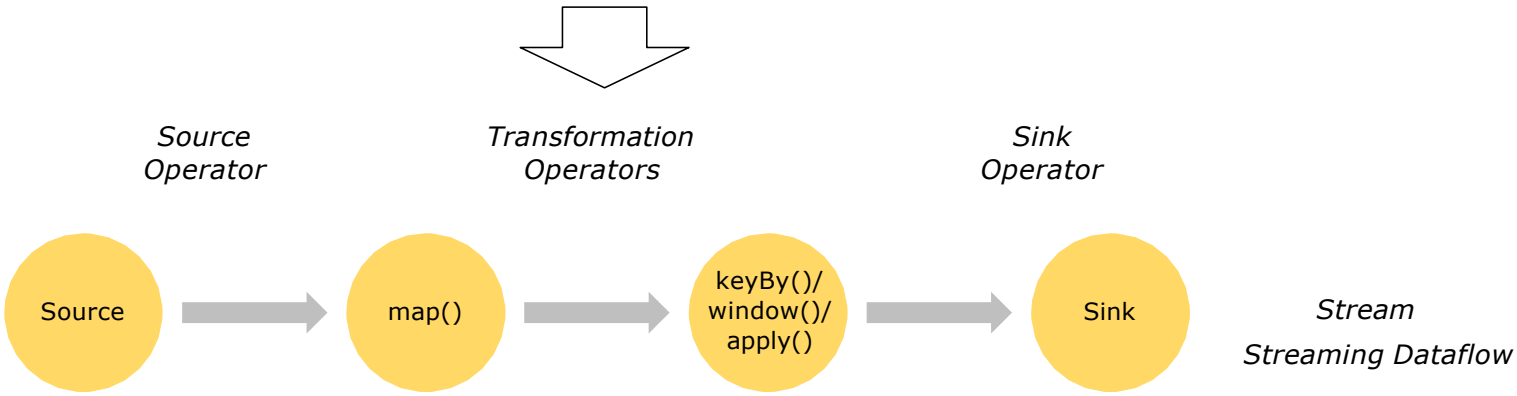
```
DataStream<String> lines = env.addSource(
                                new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

Source

Transformation

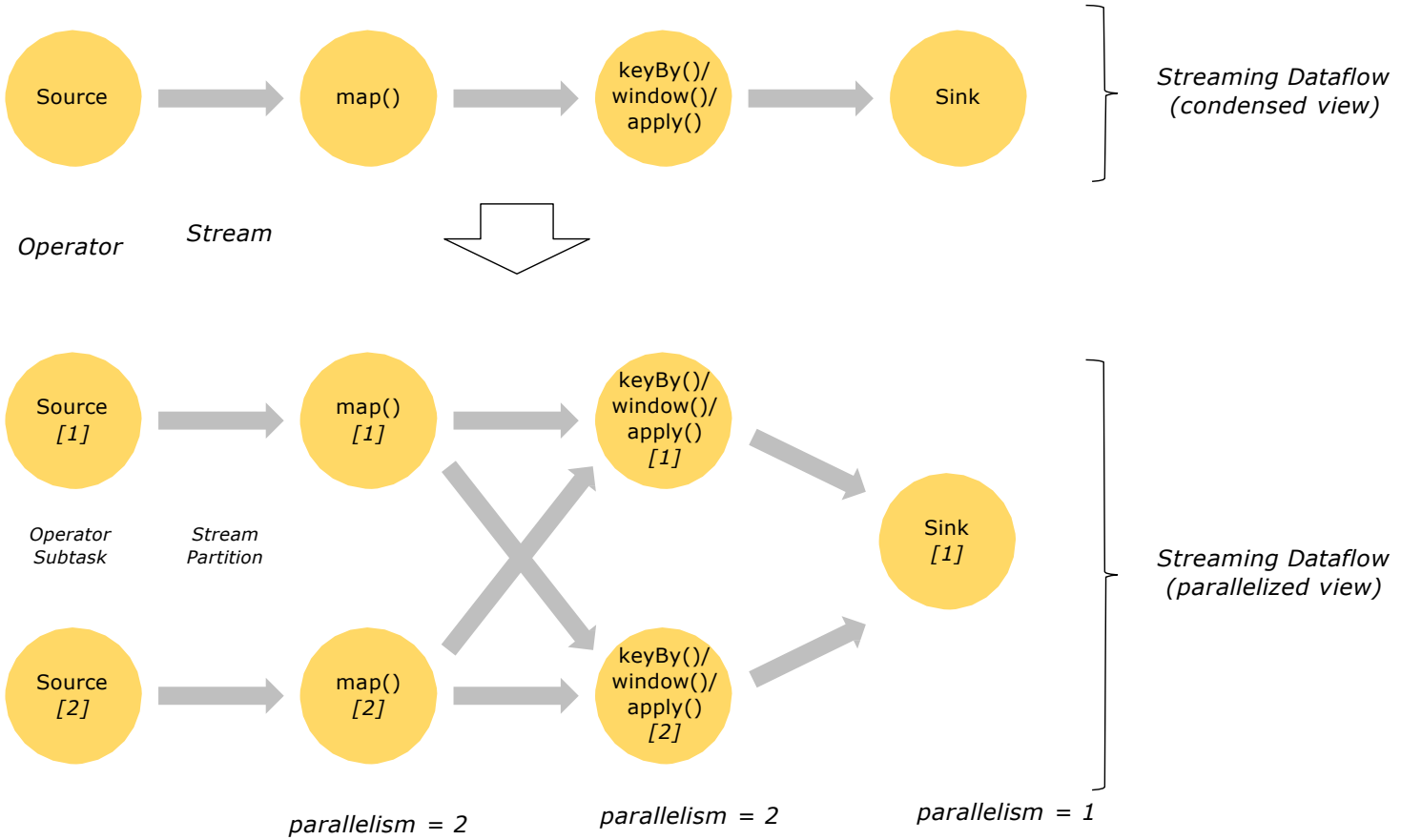
Transformation

Sink



Parallelization

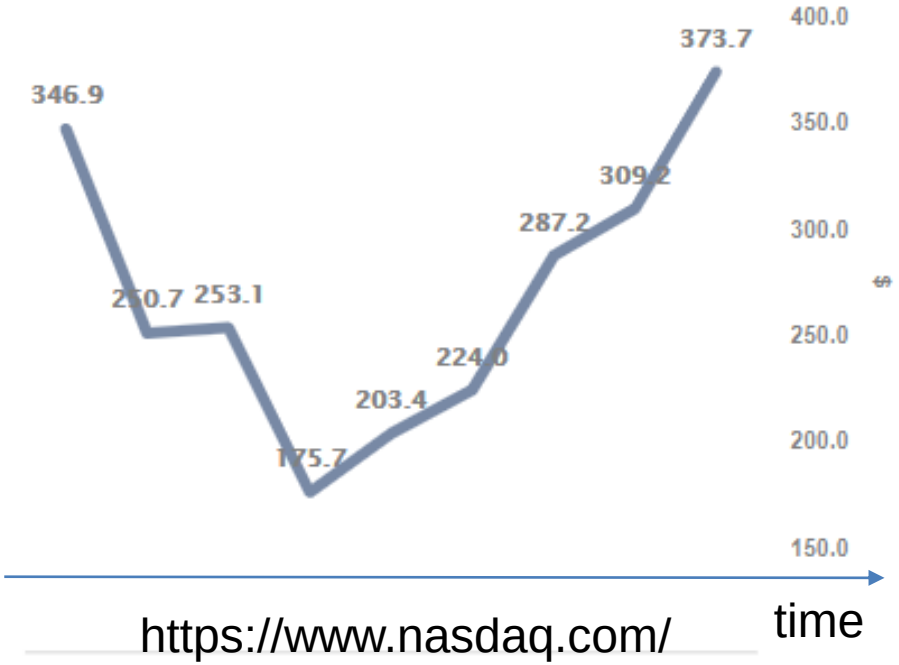
- Parallelization via stream partitions and operator subtasks
- One-to-one streams preserve the order, redistribution changes them



Outline

- Introduction (Streams & Apache Flink)
- CEP vs Traditional Streams
- FlinkCEP
  - Basics
  - Adoption barriers
- EasyFlinkCEP
  - EasyFlinkCEP Operator
  - EasyFlinkCEP Optimizer
- What we achieved
- Conclusions & Future Work

Motivating Example – CEP in Stock Market



https://www.nasdaq.com/

Motivating Example – CEP in Stock Market

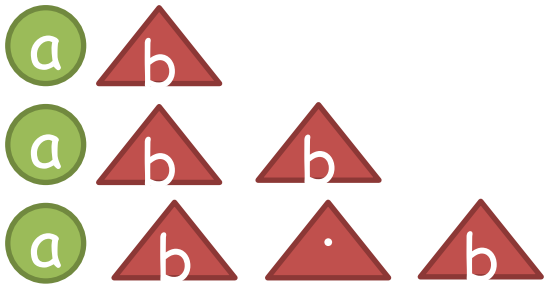
a : decreasing trend

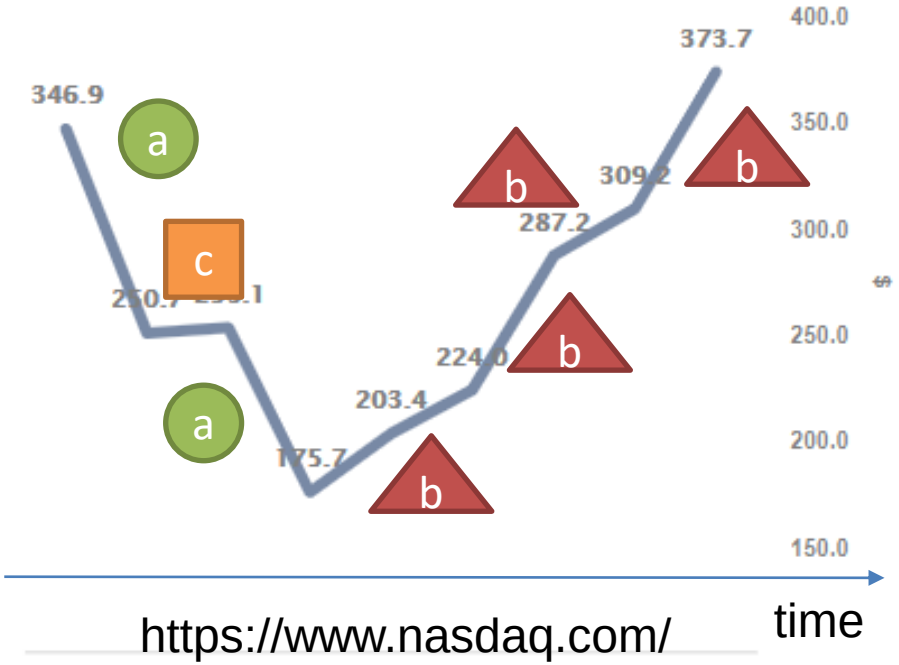
b : increasing trend

c : steady trend

d : undetermined trend

Increasing
after Decreasing
Price Trend Patterns:



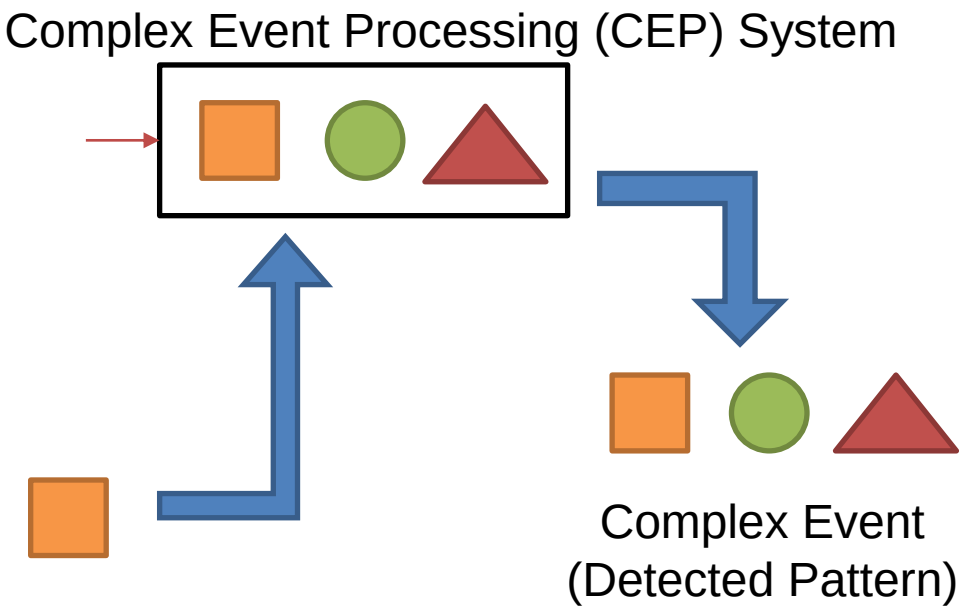


https://www.nasdaq.com/

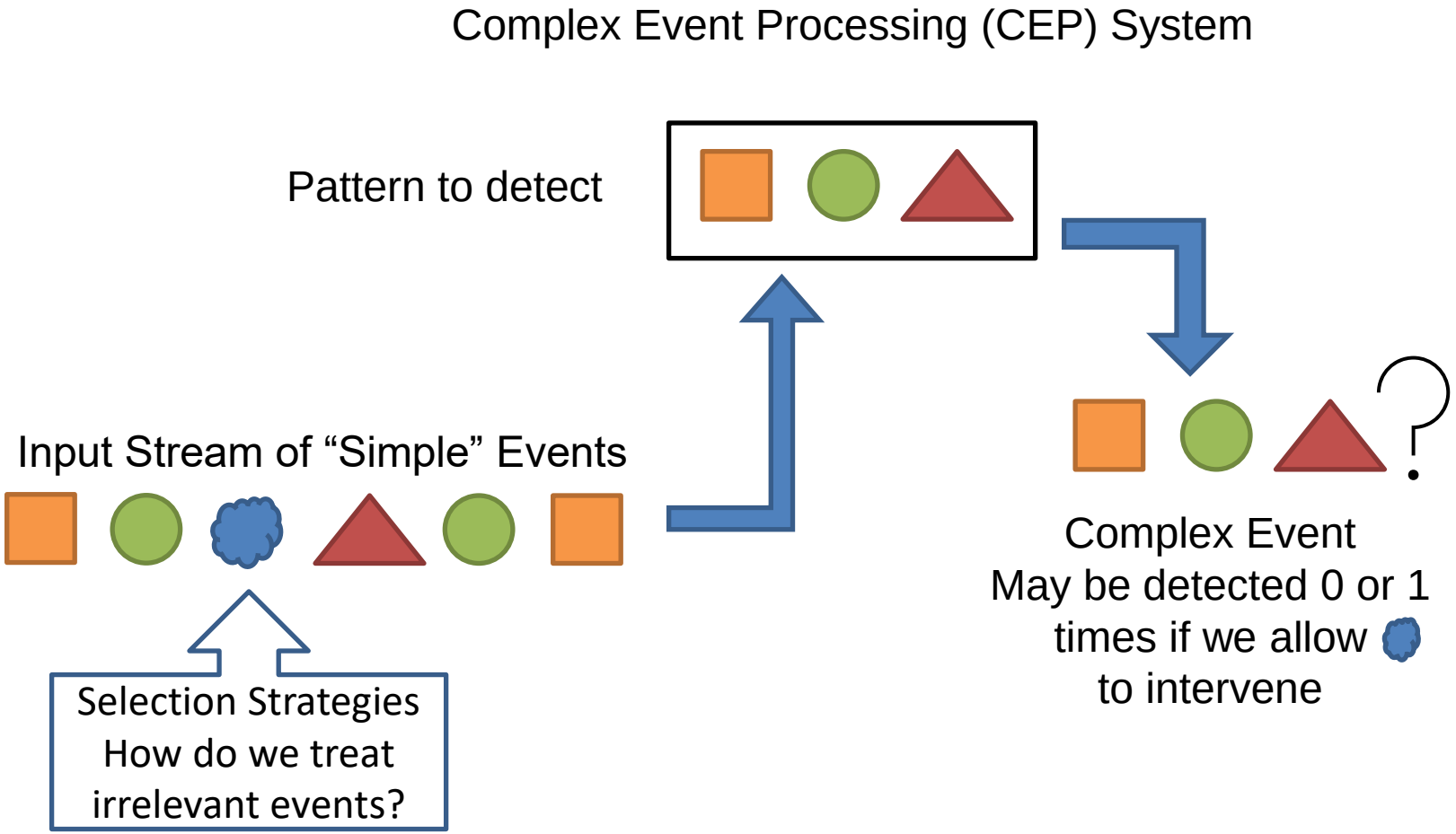
Complex Event Processing (CEP)

Query in CEP Languages
SEQ, OR, looping, use of quantifiers and more….

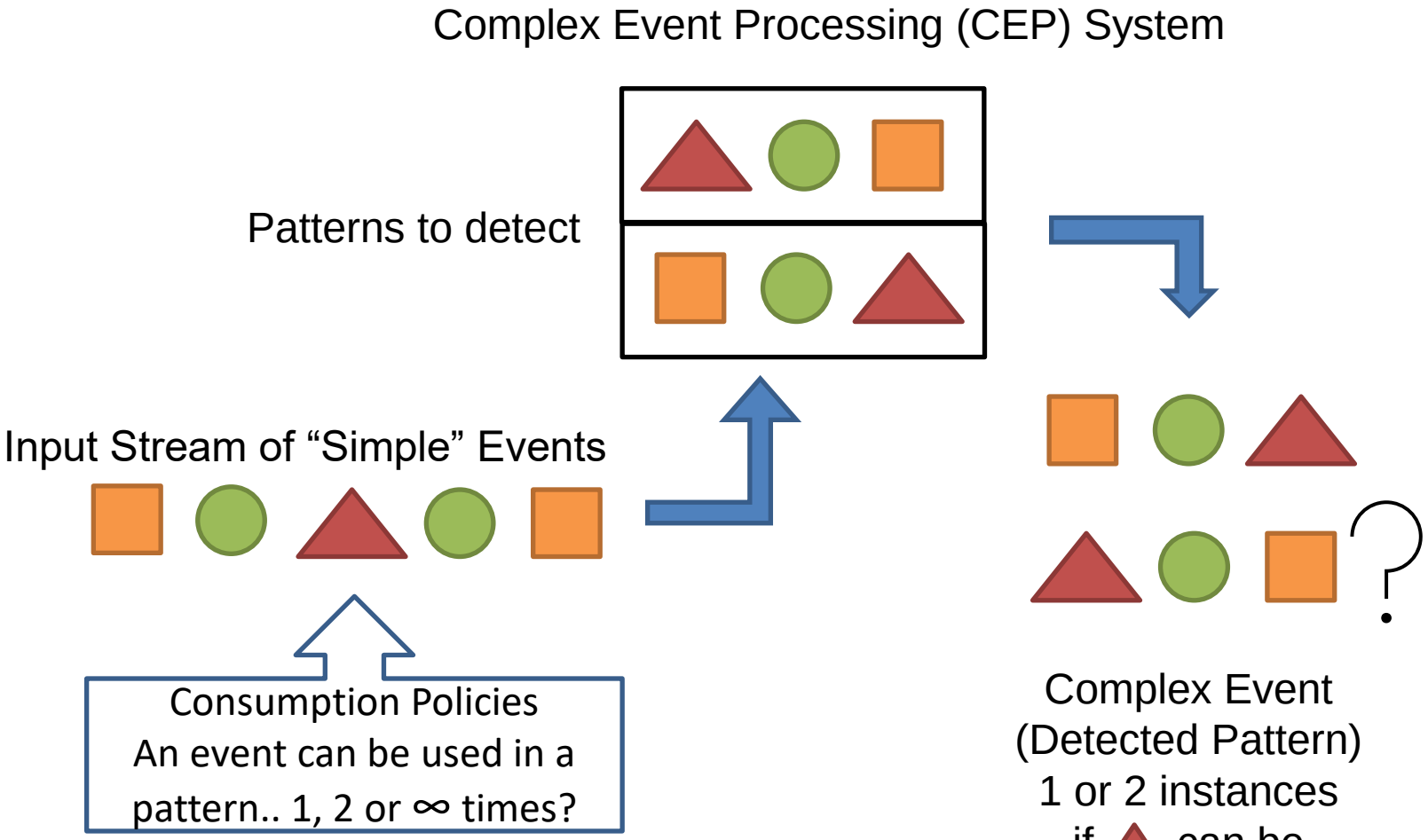
Input Stream of "Simple" Events
(can be raw stream tuples)



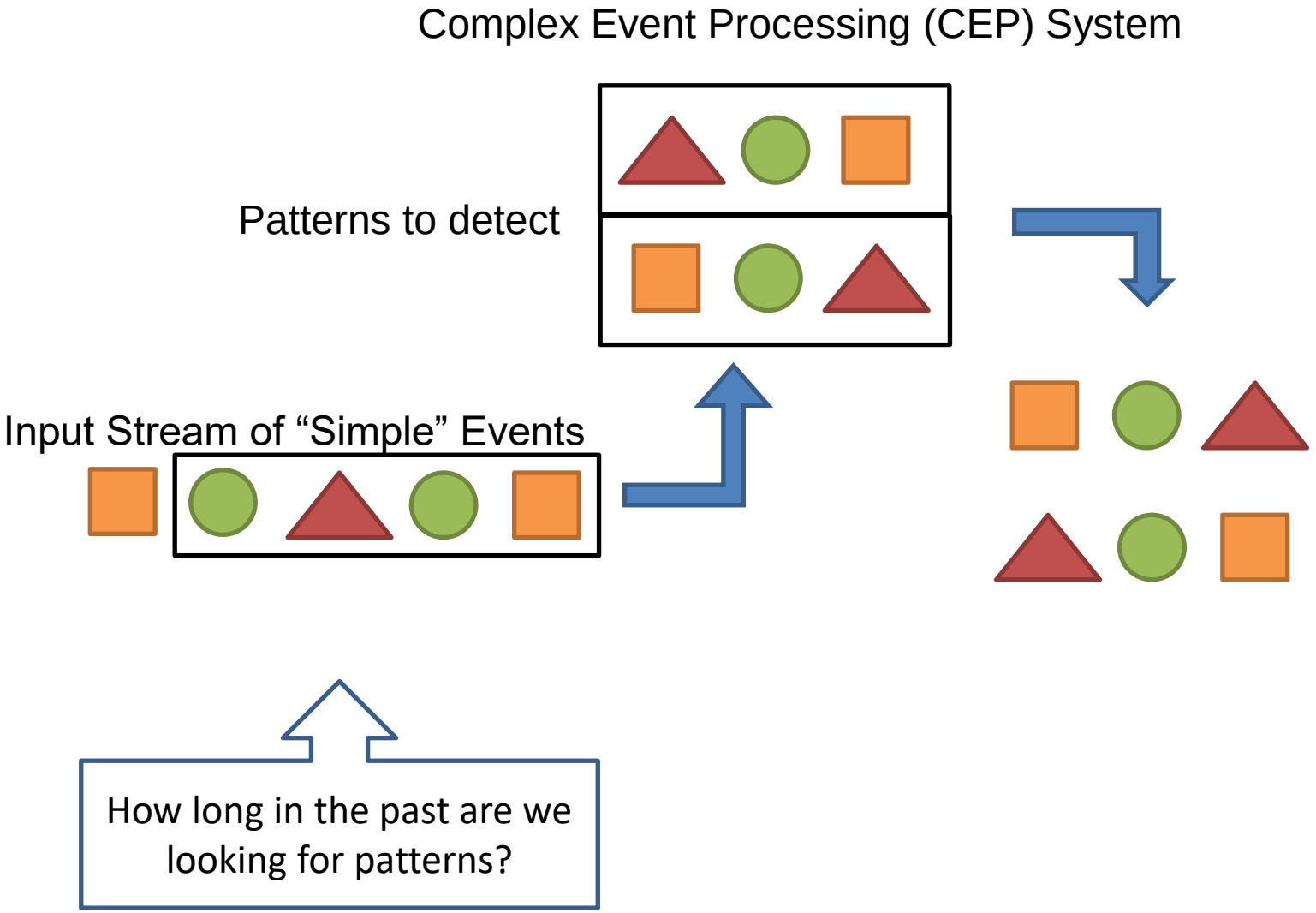
Selection Strategies



Consumption Policies



Windows

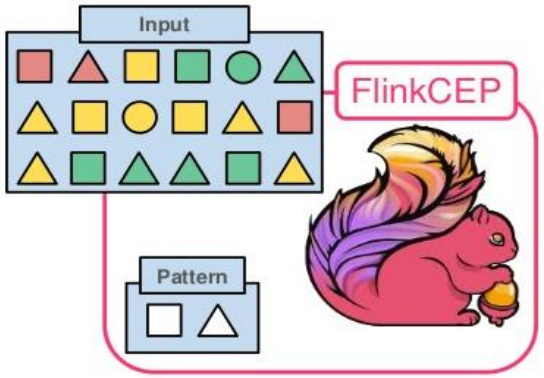

Complex Event Processing (CEP) System
Patterns to detect
Input Stream of “Simple” Events
How long in the past are we looking for patterns?

Outline

- Introduction (Streams & Apache Flink)
- CEP vs Traditional Streams
- FlinkCEP
  - Basics
  - Adoption barriers
- EasyFlinkCEP
  - EasyFlinkCEP Operator
  - EasyFlinkCEP Optimizer
- What we achieved
- Conclusions & Future Work

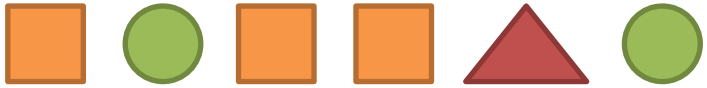
FlinkCEP: Native CEP on Apache Flink

- ✓ Parallel processing in a state-of-the-art Big Data platform
- ✓ Native API for defining CEP pipelines
- ✓ CEP language of high expressive power

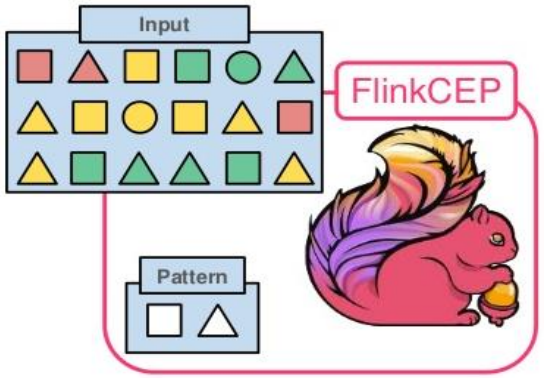


FlinkCEP: Native CEP on Apache Flink

- ✓ Selection Strategies (Example:  )
  - ✓ STRICT

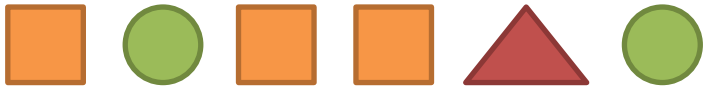
Input Stream: 

Matches: 

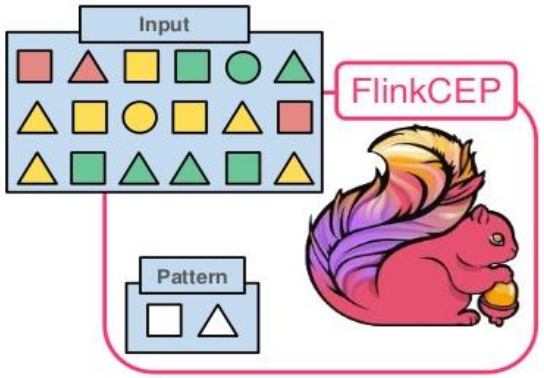


FlinkCEP: Native CEP on Apache Flink

- ✓ Selection Strategies (Example:  )
  - ✓ Relaxed

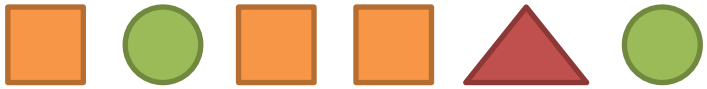
Input Stream: 

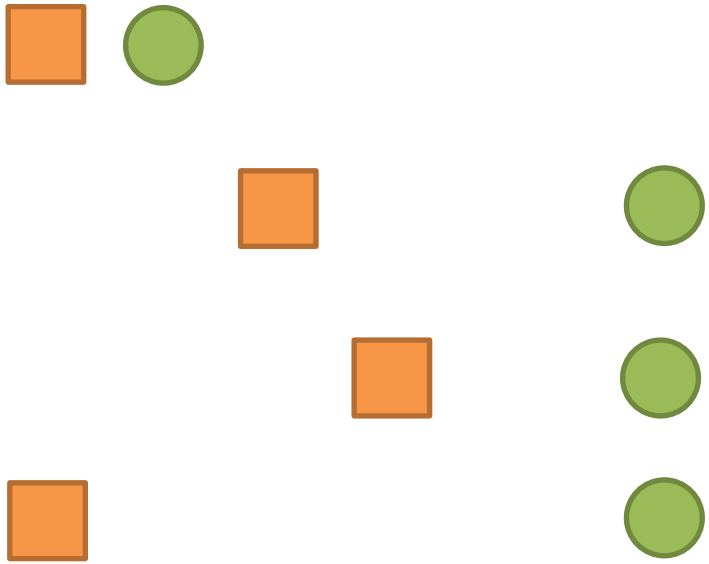
Matches: 

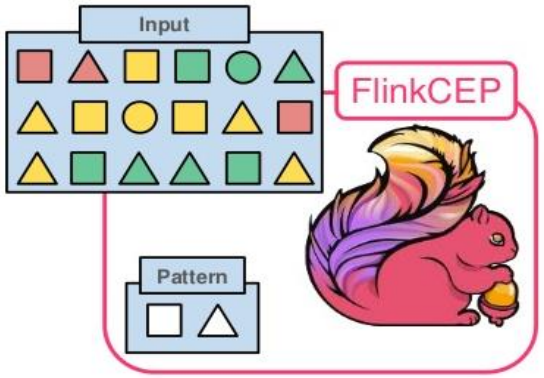


FlinkCEP: Native CEP on Apache Flink

- ✓ Selection Strategies (Example:  )
  - ✓ Non-deterministic

Input Stream: 

Matches: 

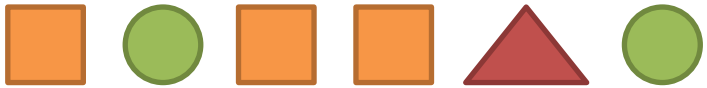


FlinkCEP: Native CEP on Apache Flink

✓ Consumption Policies (Example: )

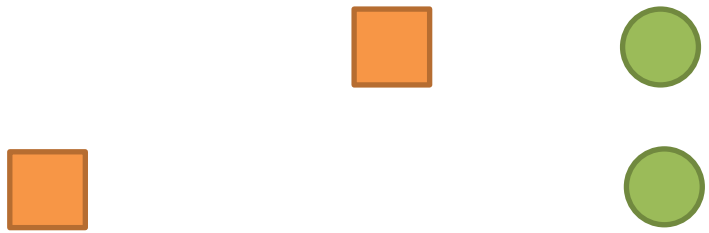
assuming Non-deterministic Selection

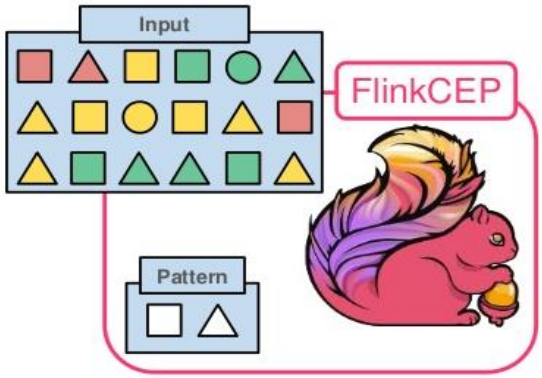
✓ NO_SKIP (Reuse everything)



Input Stream:

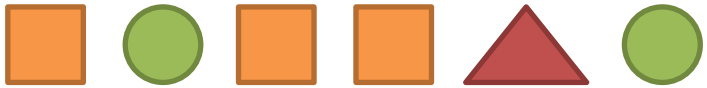
Matches:

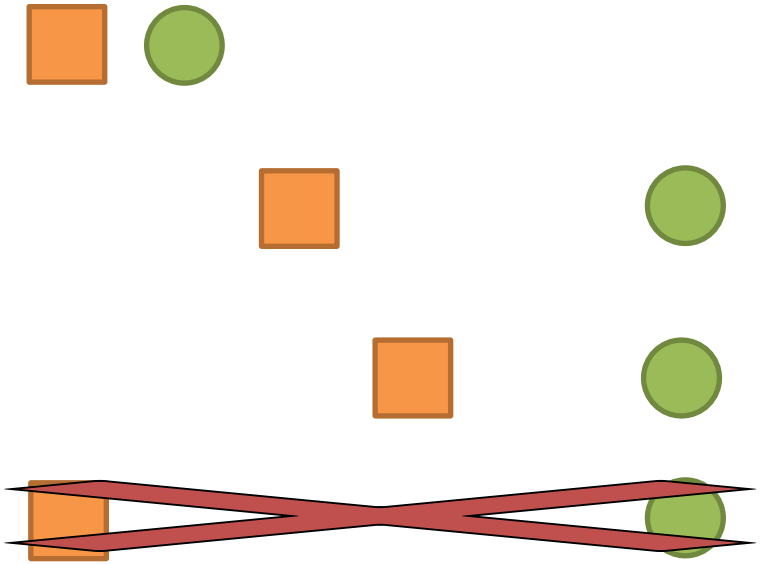


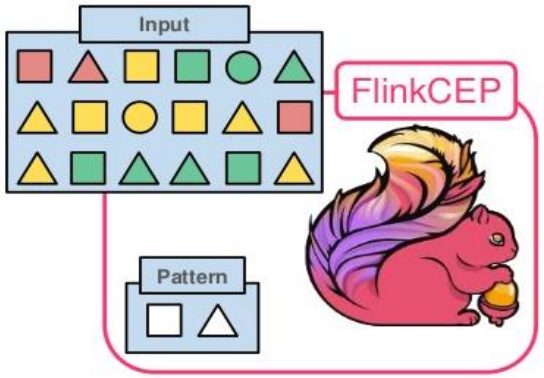


FlinkCEP: Native CEP on Apache Flink

- ✓ Consumption Policies (Example: )
  - ✓ SKIP_TO_NEXT (Uses starting event only once)

Input Stream: 

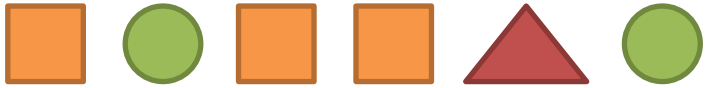
Matches: 



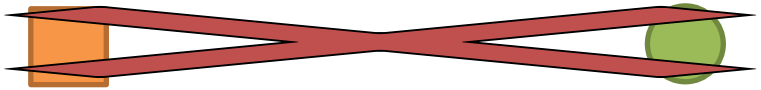
FlinkCEP: Native CEP on Apache Flink

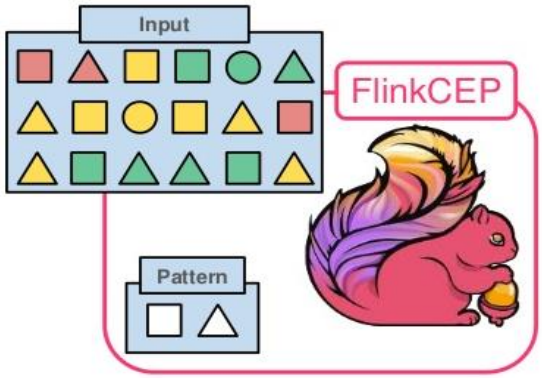
✓ Consumption Policies (Example: )

✓ SKIP_TO_LAST () (drop partial matches in-between full match)

Input Stream: 

Matches: 

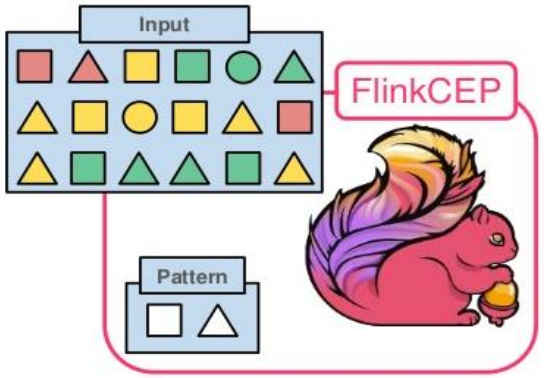




FlinkCEP: Native CEP on Apache Flink

- ✓ Parallel processing in a state-of-the-art Big Data platform
- ✓ Native API for defining CEP pipelines
- ✓ CEP language of high expressive power

- Difficult for non-programmer Business Analysts
  - Java/Scala Coding
- Cumbersome pattern parameterization
- Cluster administration decisions needed



Outline

- Introduction (Streams & Apache Flink)
- CEP vs Traditional Streams
- FlinkCEP
  - Basics
  - Adoption barriers
- EasyFlinkCEP
  - EasyFlinkCEP Operator
  - EasyFlinkCEP Optimizer
- What we achieved
- Conclusions & Future Work

Motivating Example – CEP in Stock Market

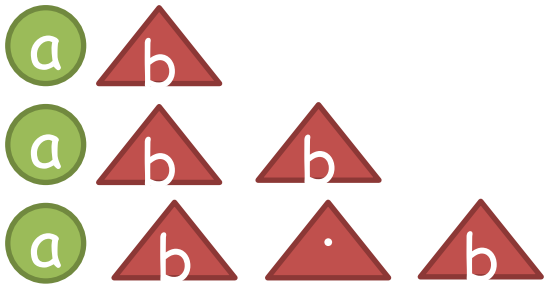
a : decreasing trend

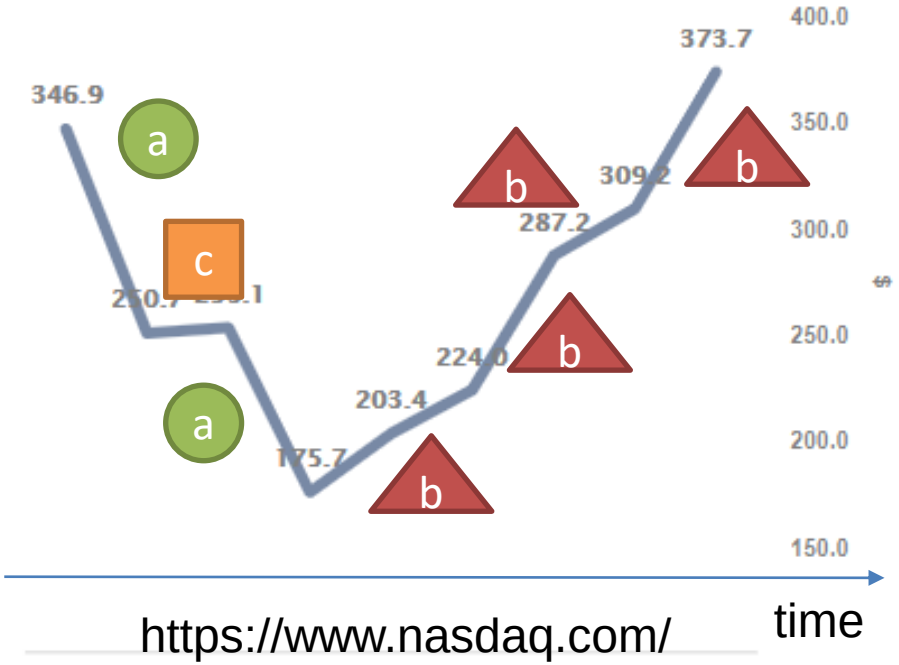
b : increasing trend

c : steady trend

d : undetermined trend

Increasing
after Decreasing
Price Trend Patterns:



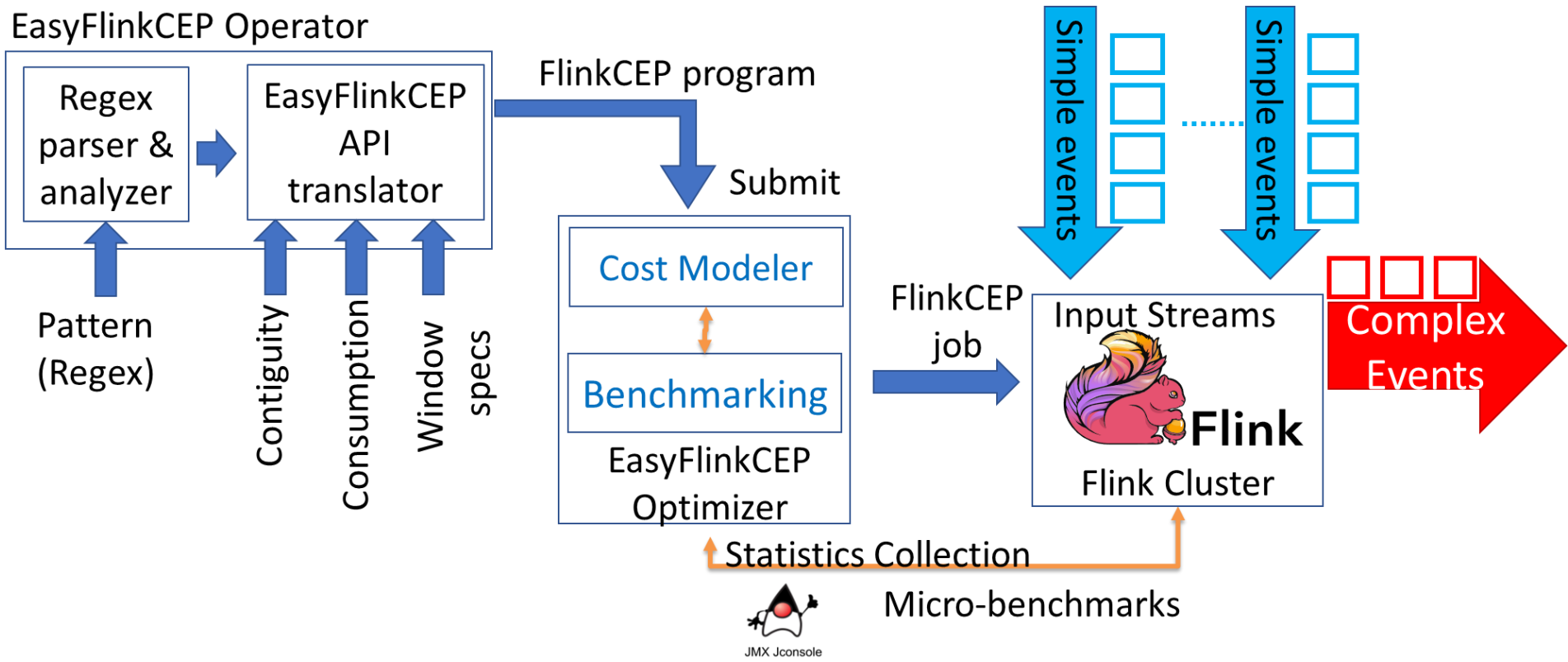


https://www.nasdaq.com/

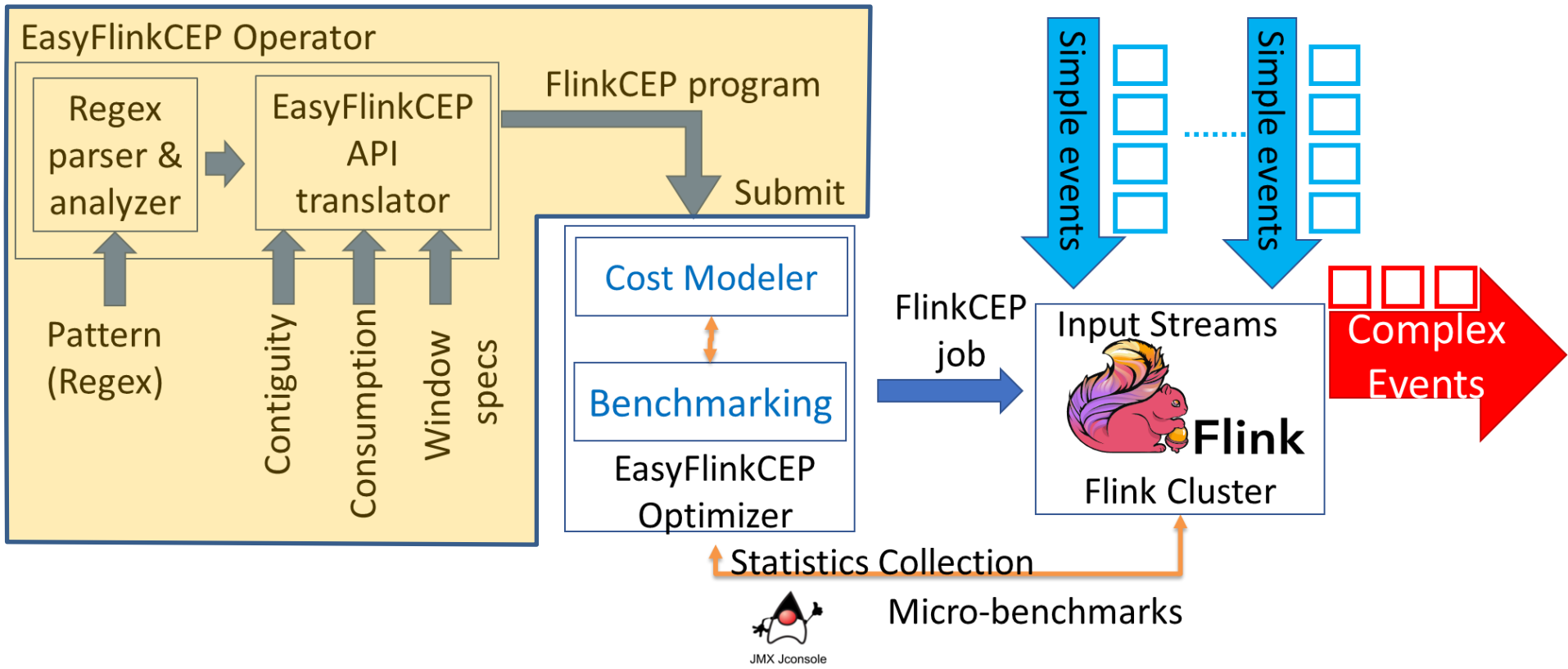
Regular Expressions Plug

- The language of the “Business Analysts”
  - Examples
    - *ab*
    - *ab+*
    - *ab**
    - *ab{1, 3}c|d*
    - *a{2, 5}b?(c|d)*

EasyFlinkCEP Architecture & Contributions



EasyFlinkCEP Operator



Motivating Example – The FlinkCEP Program

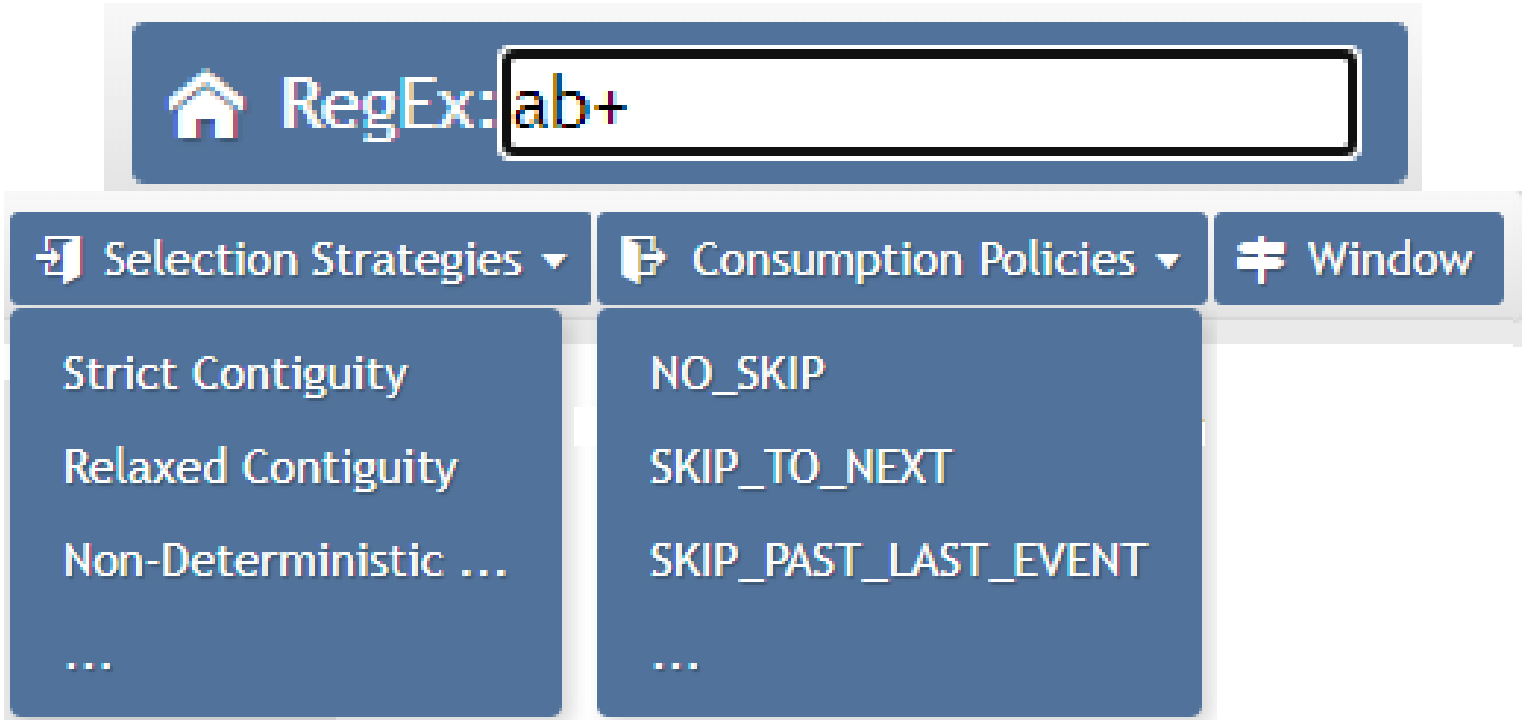
```
// ------ Create pattern "a b+" ------
Pattern<Event, ?> pattern = Pattern.<Event>begin("start", skipStrategy).where(new
SimpleCondition<>() {
    @Override
    public boolean filter (Event value){
        return value.getName().startsWith("a");}
}).followedBy("next").where(new SimpleCondition<>() { //followedBy for Relaxed Contiguity
    @Override
    public boolean filter (Event value){
        return value.getName().startsWith("b");}
}).oneOrMore().consecutive(); // oneOrMore().consecutive() for b+
PatternStream<Event> patternStream = CEP.pattern(input, pattern);
```

Motivating Example –EasyFlinkCEP Operator

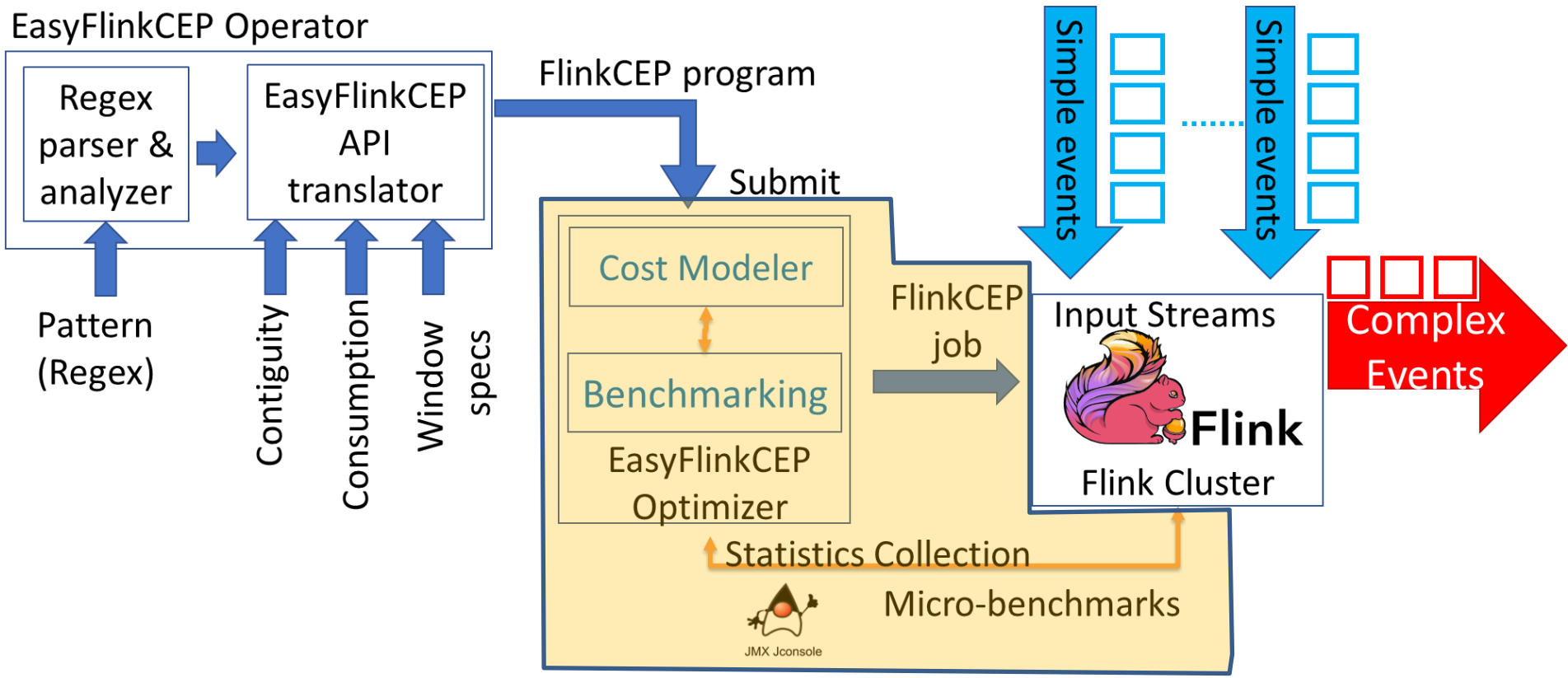
EasyFlinkCEP Operator allows to:

- ✓ Graphically Define
  RegEx, Selection, Consumption, Window

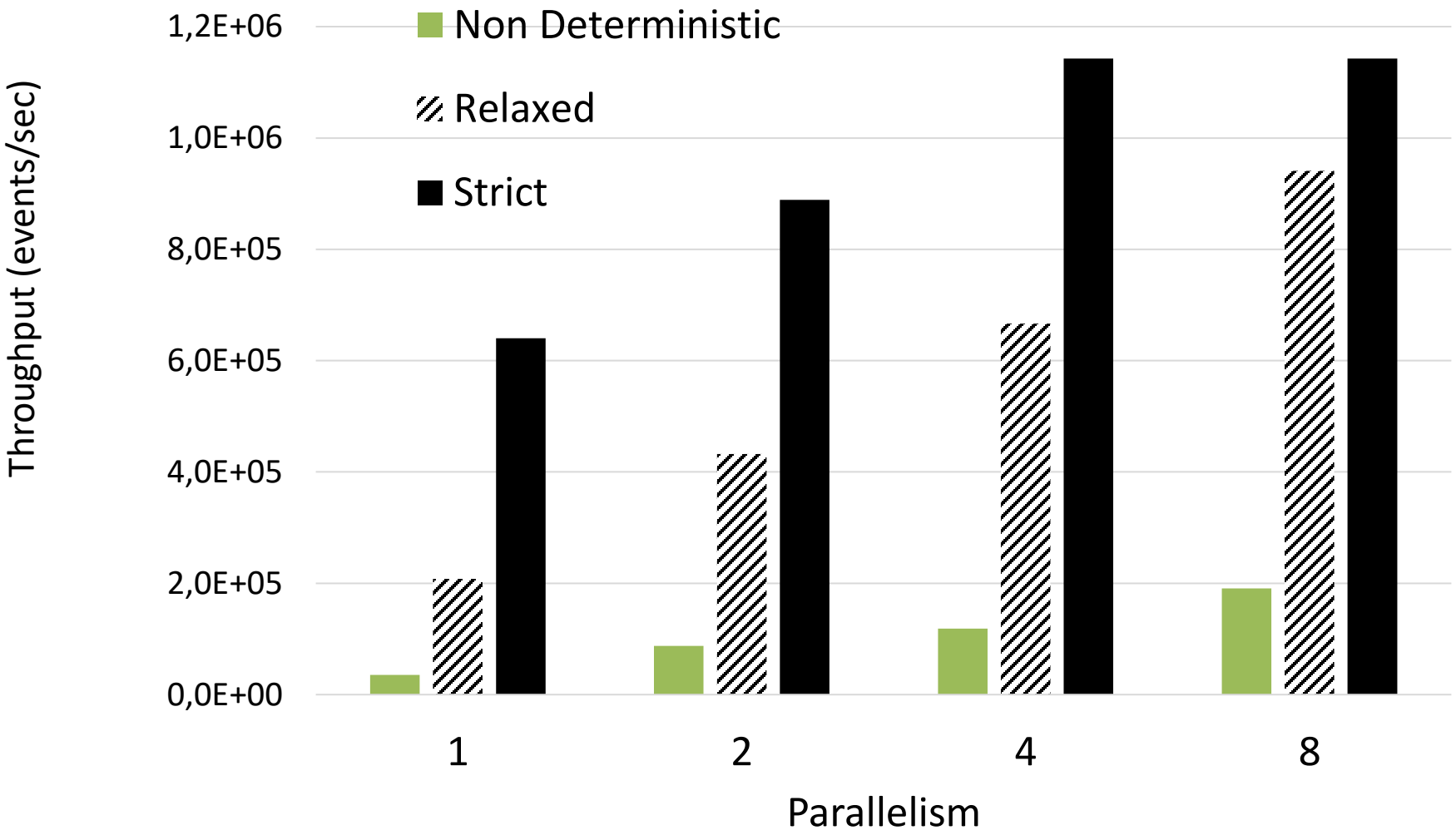
- ✓ Internally writes the FlinkCEP program



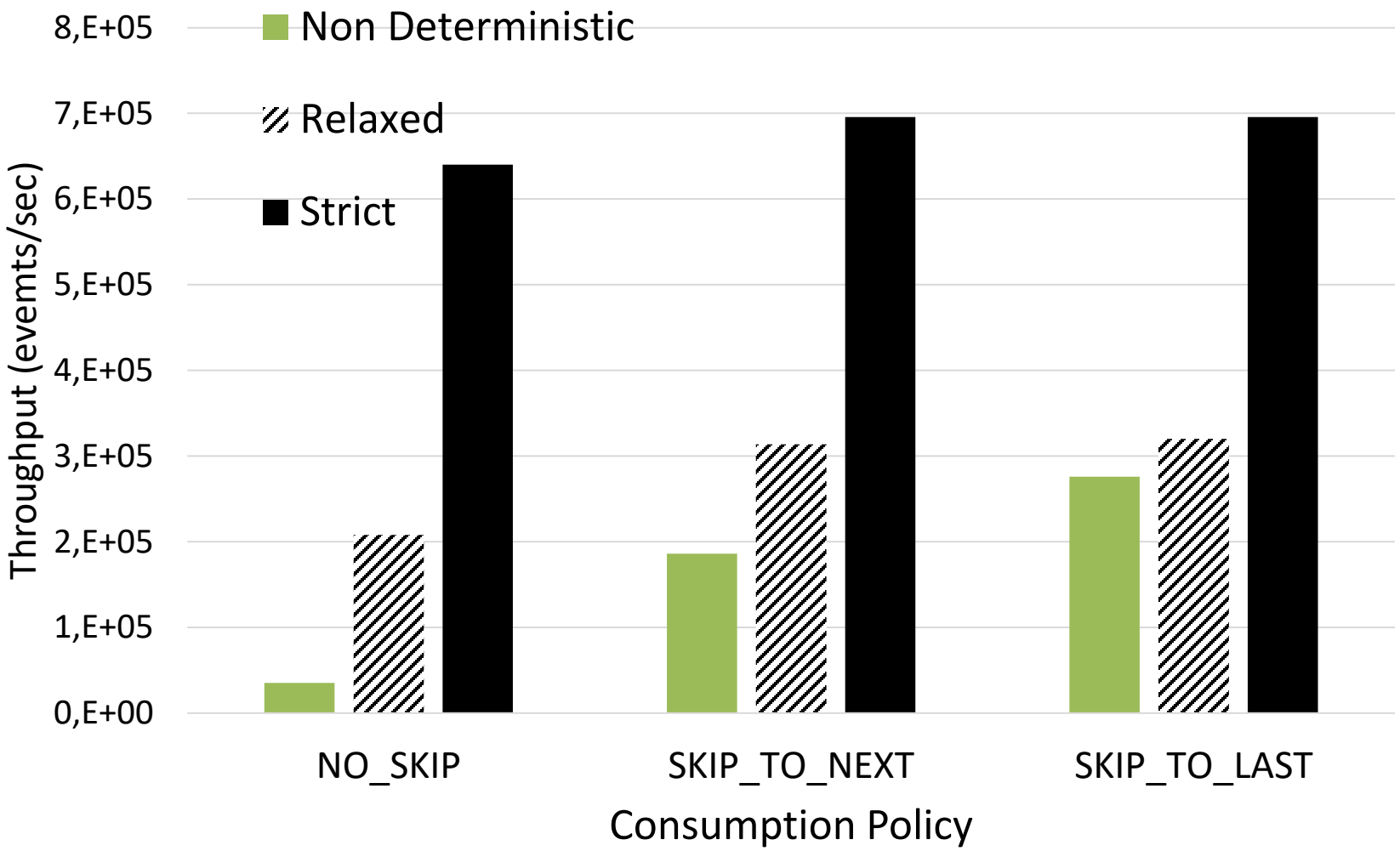
EasyFlinkCEP Optimizer



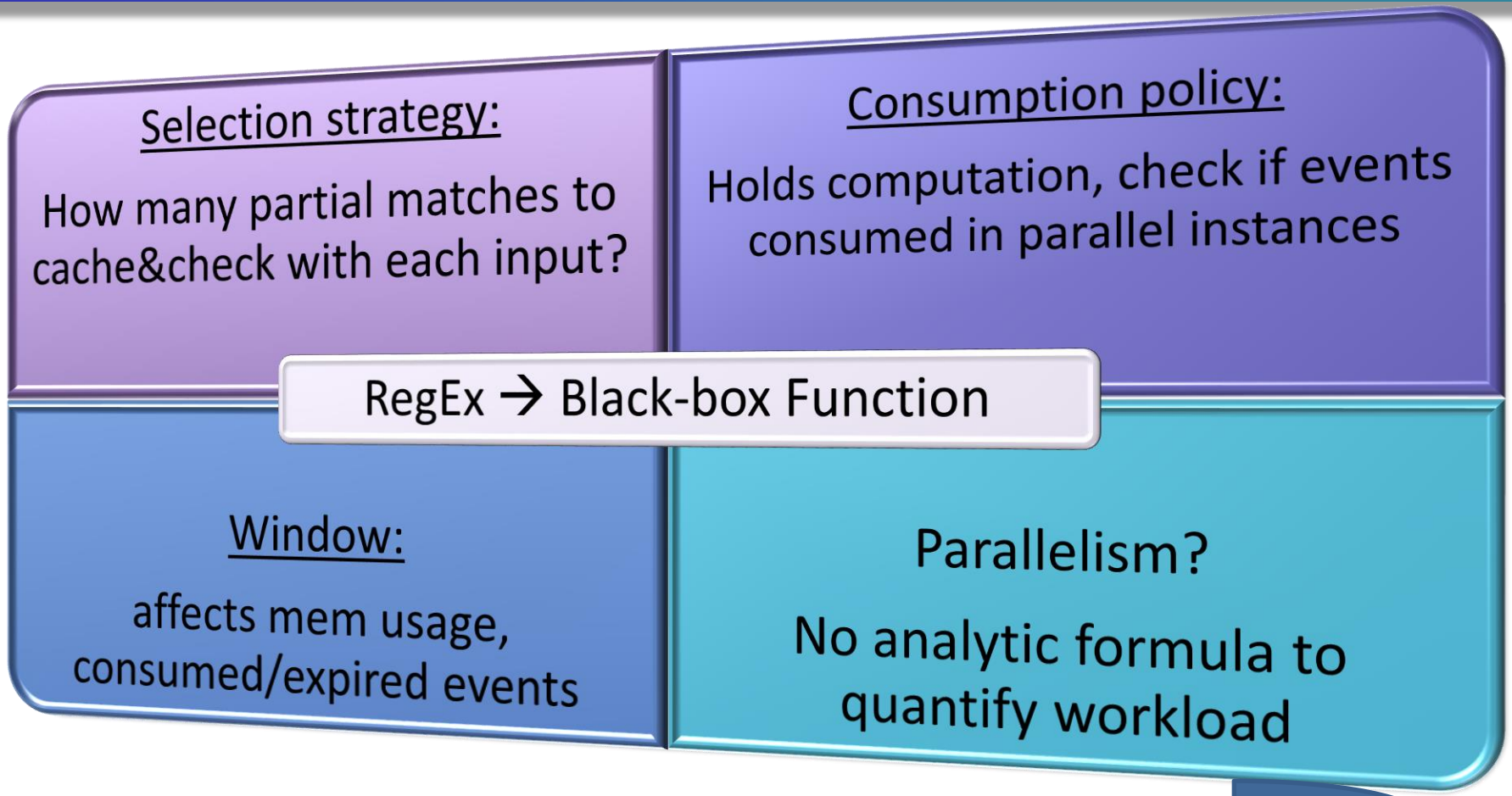
Exploring Selection Strategy Effect



Exploring Consumption Policy Effect

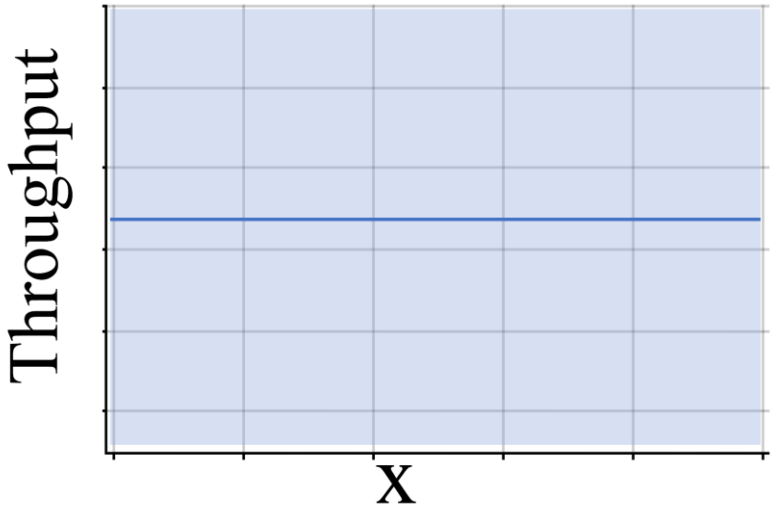


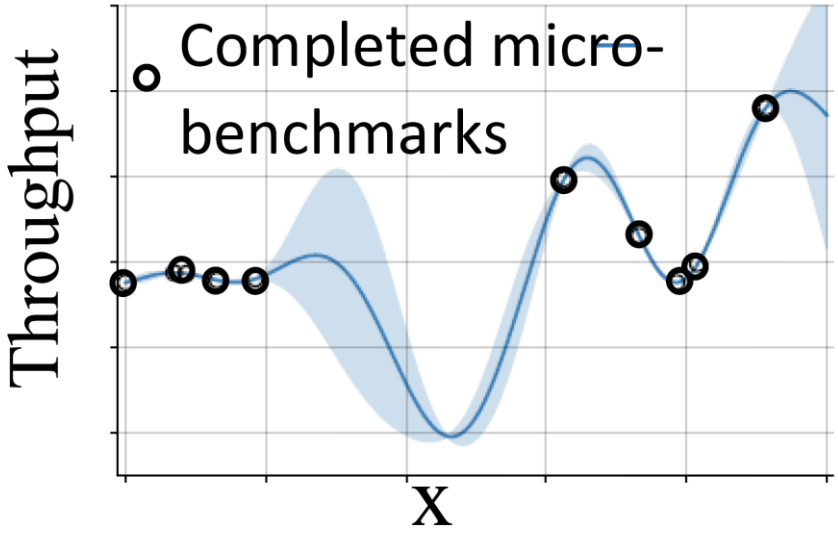
Motivation for Bayesian Optimization (BO)



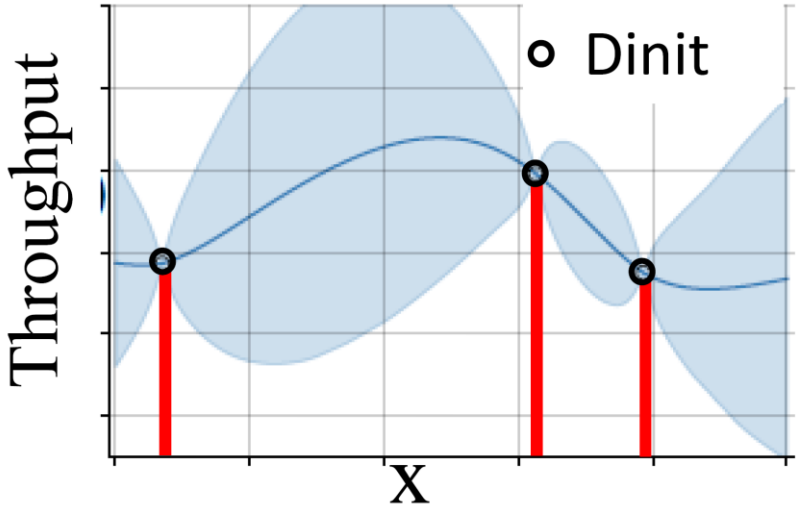
- Learn RegEx computational demands
  - Expensive function evaluations / Benchmarks
  - Time consuming

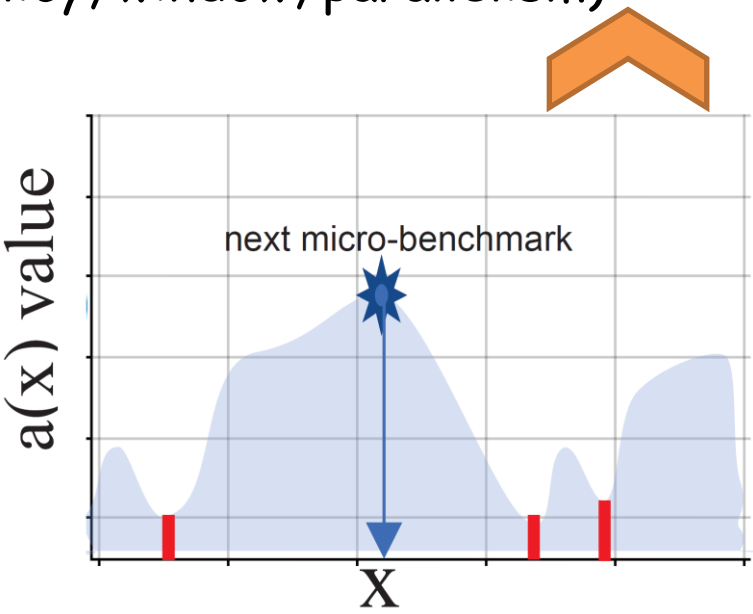
EasyFlinkCEP Benchmarking (1/2)



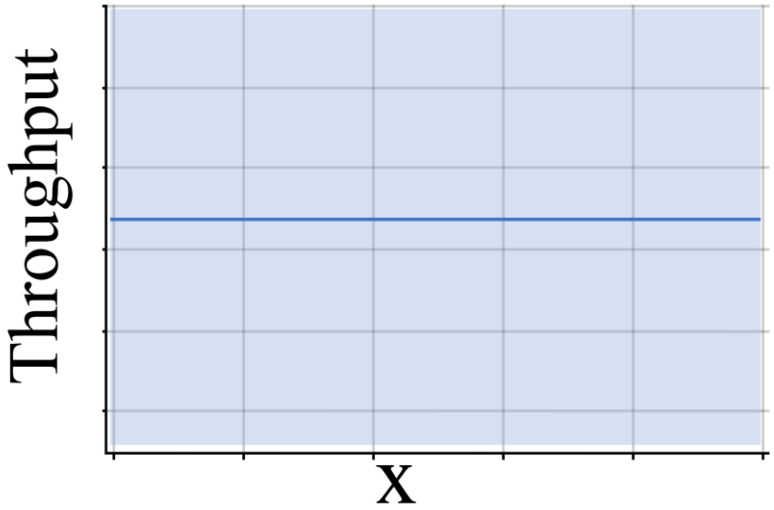


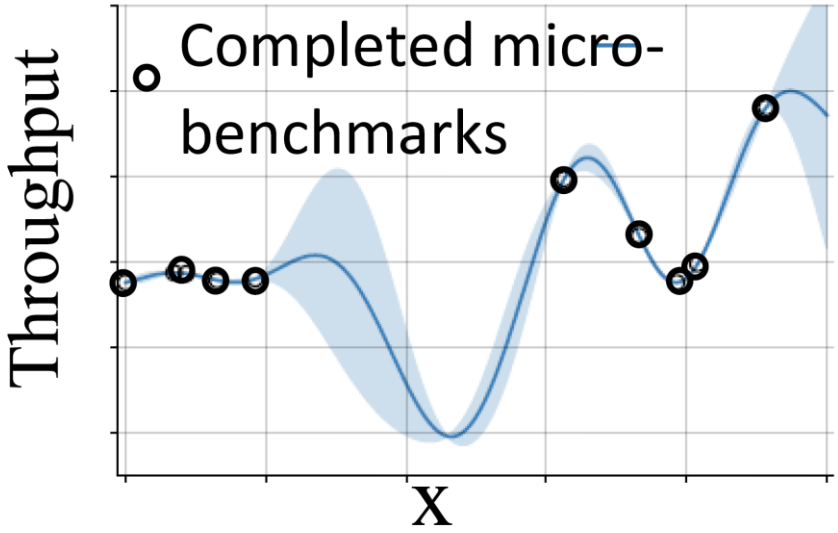
X=(selection strategy, consumption policy, window, parallelism)



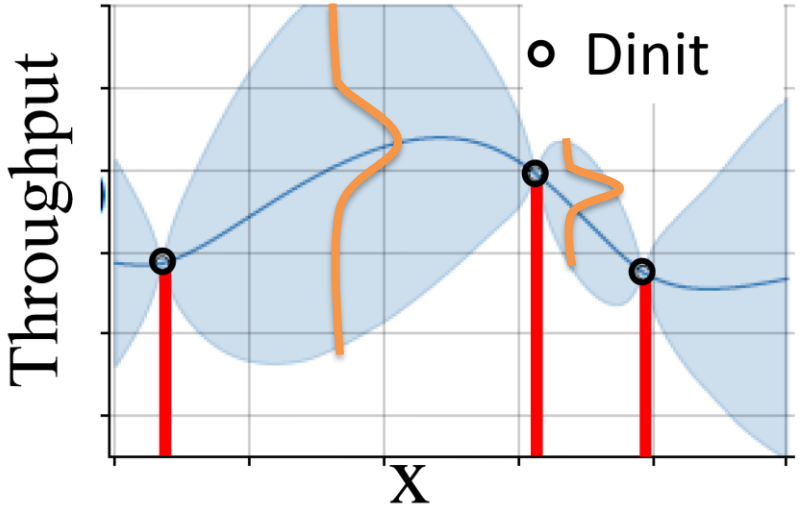


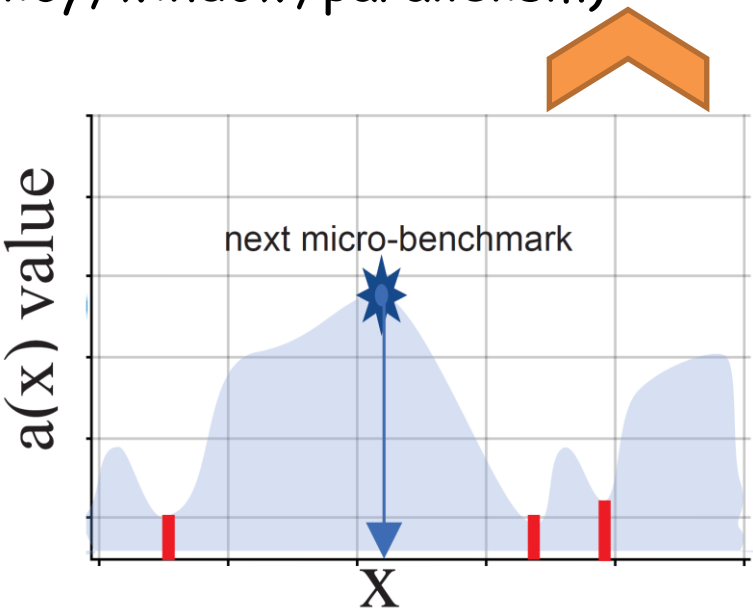
EasyFlinkCEP Benchmarking (1/2)





X=(selection strategy, consumption policy, window, parallelism)

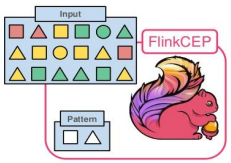




EasyFlinkCEP Benchmarking (2/2)

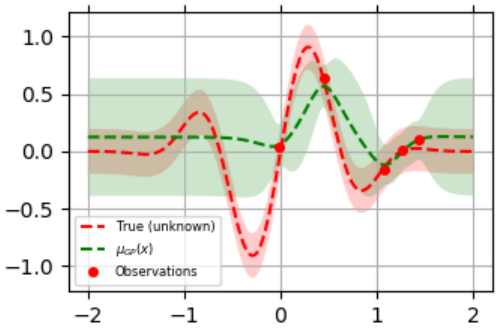
RegEx

Sample Jobs varying Selection, Consumption, Window, Parallelism

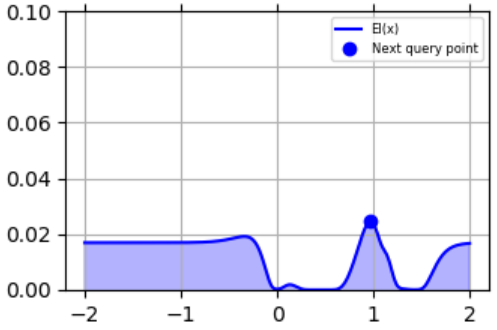


throughput

BO model



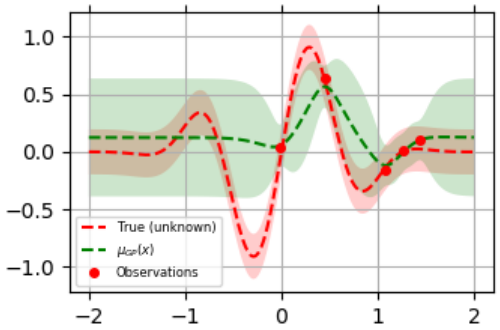
Acquisition Function



Determine next microbenchmark

EasyFlinkCEP Optimizer

Cost Modeler



What if?
Parallelism = πi

Performance Oracle
Throughput (T)

RegEx,
Selection Strategy,
Consumption Policy,
Window

Performance Oracle
under parallelism = π

Loop πi: 1..maxΠ

$\max T\pi_i / T\pi_{k,\, k<i}$
s.t. $\pi i \in \mathbb{N}$,
$1 \leq \pi i \leq \max\Pi$

Optimizer pick

Good Practices

- D_{init} proportional to 10% of all (contiguity, consumption, window, parallelism) combinations
- Use Lower Confidence Bound (LCB) as the acquisition function (higher accuracy – fewer BO iterations)
- Bucketize window value ranges of interest
- Running about 30% of all possible jobs for a few minutes suffices
- Add input distribution in the X axis if streams are highly volatile

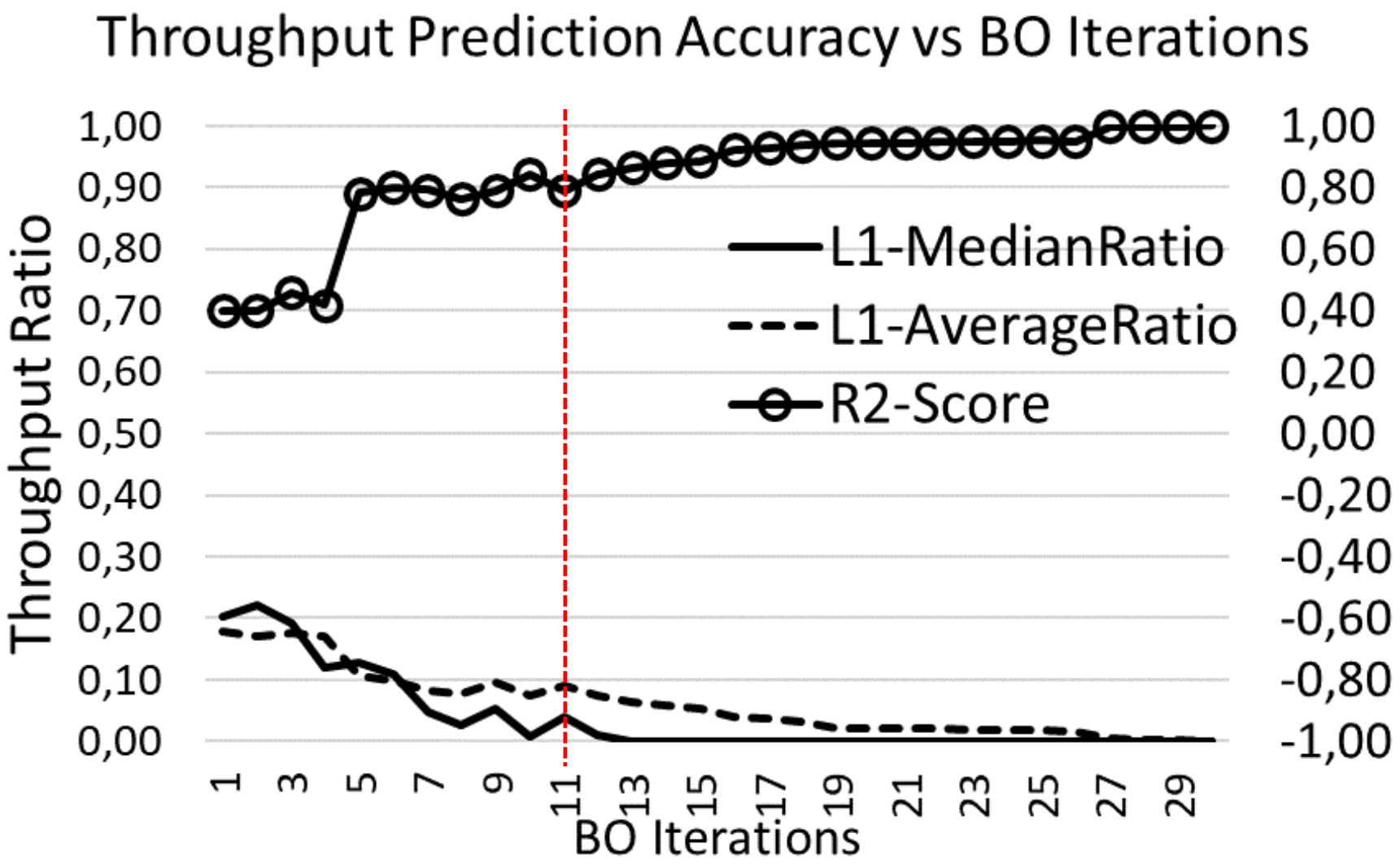
Outline

- Introduction (Streams & Apache Flink)
- CEP vs Traditional Streams
- FlinkCEP
  - Basics
  - Adoption barriers
- EasyFlinkCEP
  - EasyFlinkCEP Operator
  - EasyFlinkCEP Optimizer
- What we achieved
- Conclusions & Future Work

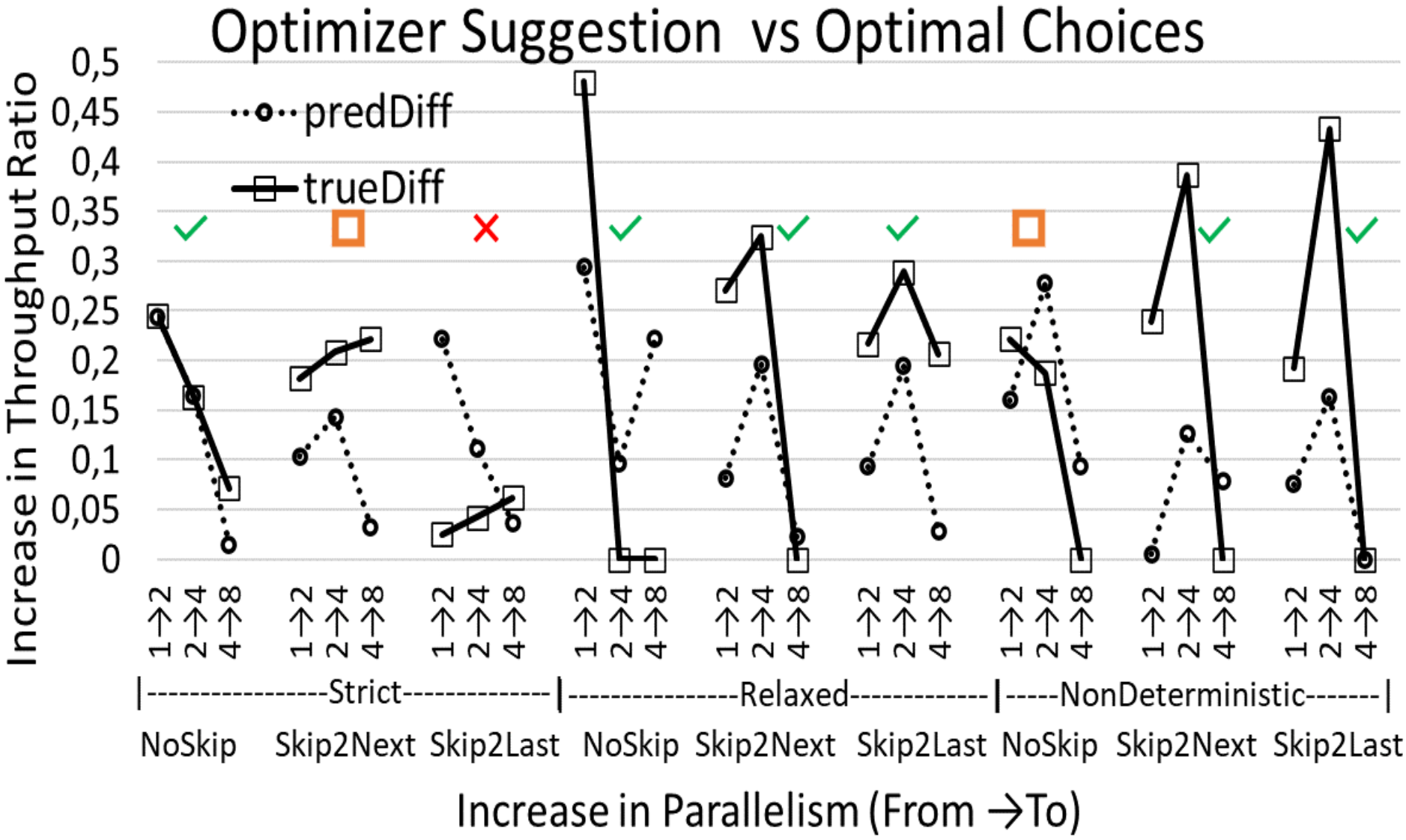
Experimental Data & Setup

- What we evaluate
  - Accurate Cost modeler?
  - …with Few benchmarks required?
  - Right Optimizer Choices?
- Datasets
  - 8 stock streams w 2M events each
  - RegEx: $ab\{1, 3\}c|d$ varying selection/consumption params
  - Window of 128 events/stream
- Cluster Setup
  - VM with 16 cores and 16GB of main memory running Ubuntu 18.04

Experimental Results



Experimental Results



Outline

- Introduction (Streams & Apache Flink)
- CEP vs Traditional Streams
- FlinkCEP
  - Basics
  - Adoption barriers
- EasyFlinkCEP
  - EasyFlinkCEP Operator
  - EasyFlinkCEP Optimizer
- What we achieved
- Conclusions & Future Work

Conclusions & Future Work

- EasyFlinkCEP
  - Putting event analysts back to business
    - No coding required
    - No cluster administration decisions
  - 1st parallel CEP Optimizer
- Future Steps:
  - Better FlinkCEP Language Support & Event Hierarchies
    - More complex selection strategy combinations
    - More complex consumption policy combinations
    - Harder to properly define windows
  - Runtime Adaptation

Few References

- EasyFlinkCEP: Big Event Data Analytics for Everyone
  Nikos Giatrakos, Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, Yannis Kotidis:. CIKM 2021

- Complex event recognition in the Big Data era: a survey
  Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, Minos Garofalakis:. VLDB J. 29(1): 313-352 (2020)

http://infore-project.eu/

EasyFlinkCEP:
Big Event Data Analytics for Everyone

Thank you!

ngiatrakos@athenarc.gr
ngiatrakos@softnet.tuc.gr