Differential Testing for Stream Processing Systems

Konstantinos Kallas, **Filip Niksic**, Caleb Stanford, Rajeev Alur

University of Pennsylvania

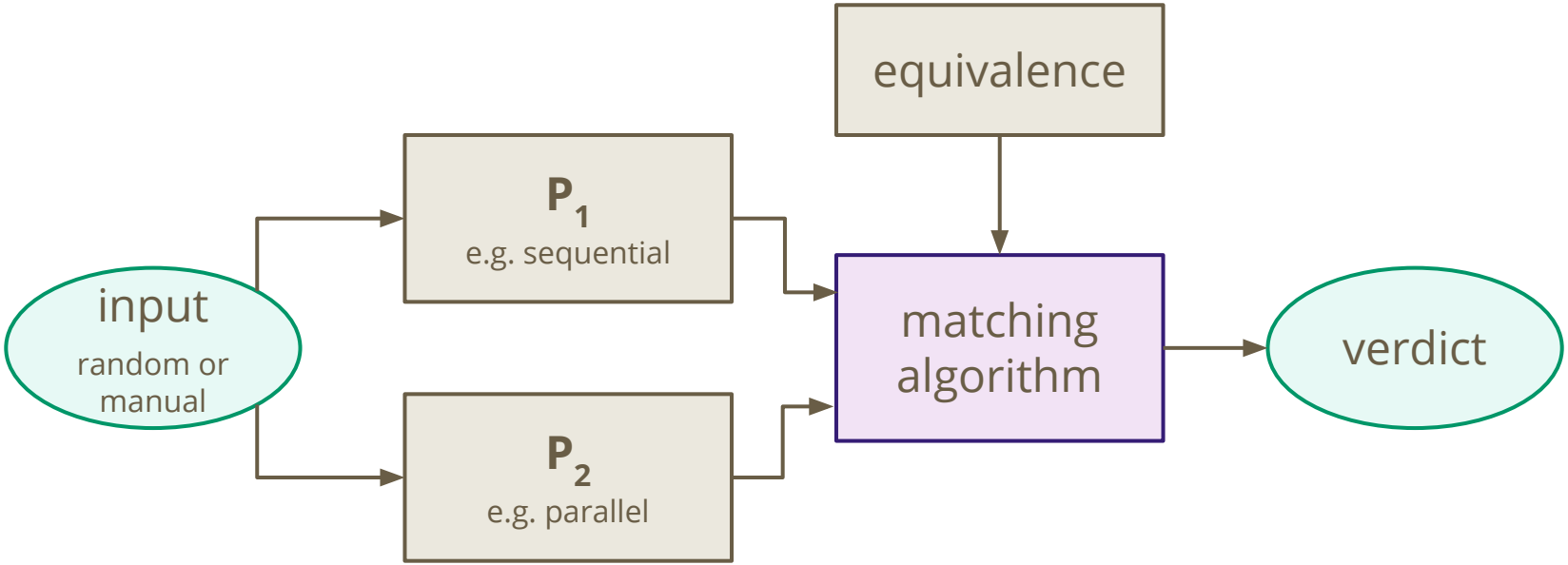
Distributed stream processing systems

Evolution of “big data” processing systems:

- **event-by-event** processing on **unbounded** streams of events
- **stateful** computations
- **parallel** processing in a **distributed** system
- **low latency, high throughput**
- fault tolerance, ...

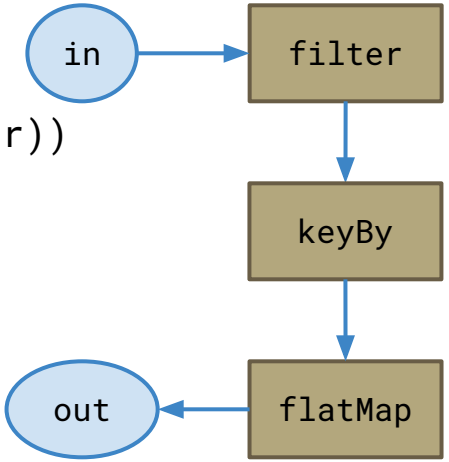
Stream processing programs are prone to errors due to data reordering

Our approach: Differential testing (for Flink)



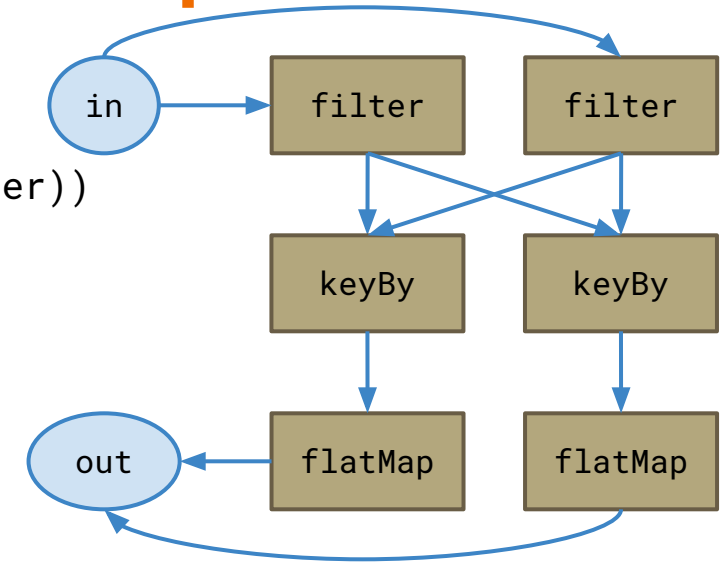
Flink program for processing stock prices

```
DataStream<Price> in = env.addSource(...);

out = in.filter(p → portfolio.contains(p.ticker))
        .keyBy(p → p.ticker)
        .flatMap(computeRateOfChange);

out.addSink(...);

env.execute();
```



Flink program for processing stock prices

```
DataStream<Price> in = env.addSource(...);

out = in.filter(p → portfolio.contains(p.ticker))
        .keyBy(p → p.ticker)
        .flatMap(computeRateOfChange);

out.addSink(...);

env.execute();
```

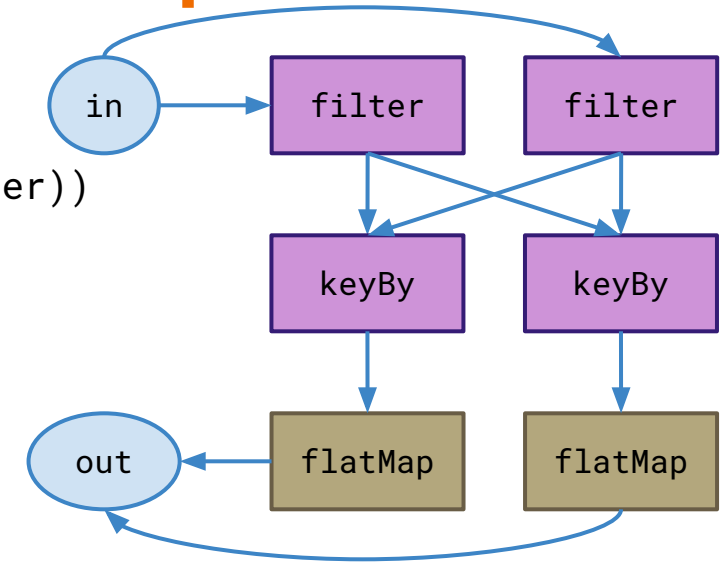


parallelism = 2

- **Prices with the same ticker should not be reordered**
- **`filter` can reorder prices!**

Flink program for processing stock prices

```
DataStream<Price> in = env.addSource(...);

out = in.filter(p → portfolio.contains(p.ticker))
        .keyBy(p → p.ticker)
        .flatMap(computeRateOfChange);

out.addSink(...);

env.execute();
```



parallelism = 2

- **Prices with the same ticker should not be reordered**
- **`filter` can reorder prices!**

Equivalence of streams

Specified as a **dependence relation** on events:

- a symmetric relation
- gives rise to a **logical ordering** of the events in the stream
- streams are equivalent iff the events have the same logical ordering (cf. Mazurkiewicz traces)

Example:

- prices ordered per ticker: `p D q` ⇔ `p.ticker == q.ticker`

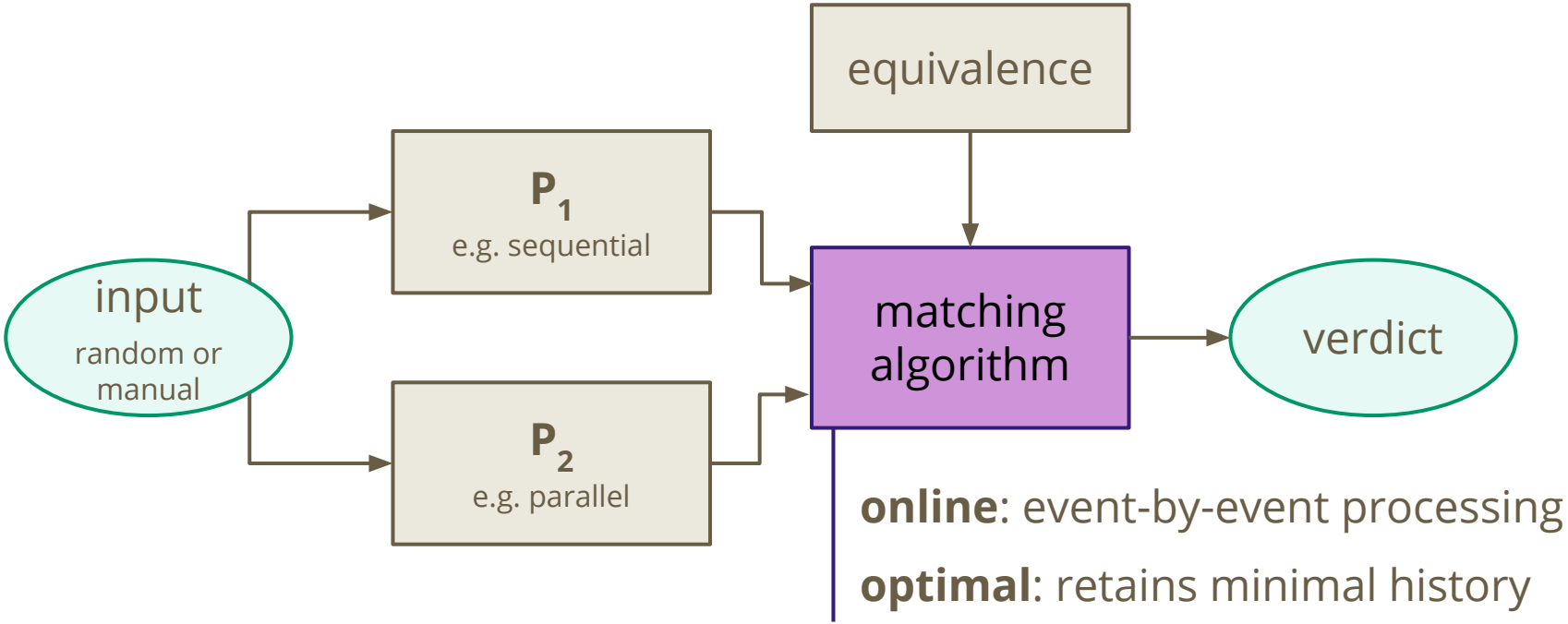
Using our testing framework

```
DataStream<Price> in = env.addSource(...);

expected = in.filter(p → portfolio.contains(p.ticker))
             .parallelism(1)
             .keyBy(p → p.ticker)
             .parallelism(1);
obtained = in.filter(p → portfolio.contains(p.ticker))
             .keyBy(p → p.ticker);

Matcher<Price> matcher =
    Matcher.create(expected, obtained, (p,q) → p.ticker == q.ticker);

env.execute();

matcher.assertEquivalence();  // Fails since filter reorders
```

Using our testing framework

```
DataStream<Price> in = env.addSource(...);

expected = in.filter(p → portfolio.contains(p.ticker))
             .parallelism(1)
             .keyBy(p → p.ticker)
             .parallelism(1);
obtained = in.filter(p → portfolio.contains(p.ticker))
             .parallelism(1)
             .keyBy(p → p.ticker);

Matcher<Price> matcher =
    Matcher.create(expected, obtained, (p,q) → p.ticker == q.ticker);

env.execute();

matcher.assertEquivalence();  // Succeeds
```

Our approach: Differential testing (for Flink)



Contributions

Online algorithm for the equivalence of streams

- Uniformly handles complex orderings of data

Differential testing framework for Flink

- Integrates with JUnit and existing Flink testing tools

Case studies

- Nontrivial parallelization task
- Real-world MapReduce programs from the literature
- Online monitoring