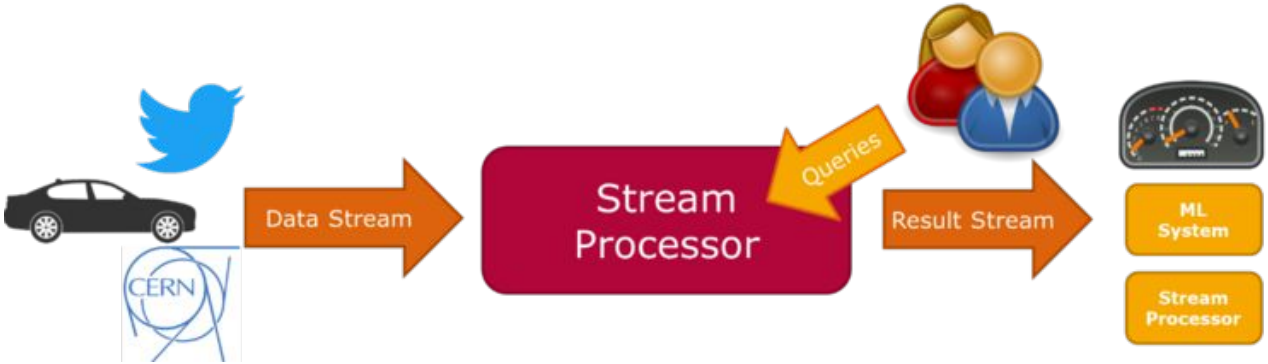


HDES: A Dynamic Stream Processing Engine

Nico Duldhardt, Torben Meyer, Marvin Thiele, Anton von Weltzien
14.04.2020
Masterprojekt WS 19/20
Data Engineering Systems

Agenda

1. Goals
2. Features
3. Architecture Overview
4. Query Transformation
5. Query Execution
6. Ad-hoc join processing
7. Benchmark Results

Goals

1. Build a **standalone prototype** of a stream processing engine that has first class support for dynamic query deployment and removal
2. Support processing **simple queries** and **streams**
3. Support **online optimizations** for efficient multi-query processing

Features

- Stream Processing Framework written in **Java 11**
- **Ad-hoc** addition and removal of arbitrary queries
- **Single** node, **multi-threaded** execution
- **Optimization** for Joins and Aggregations in **multi-query execution**
- Queries are defined in a Flink-like **dataflow language**
- Support for **Sliding-** and **Tumbling-Windows** both with **Event-** and **Processing-Time**

Dataflow API Overview

Stream type	Name	Description	Resulting type
AStream	flatMap	Maps an incoming elements to an iterable of any type.	AStream
	map	Maps an incoming element to a arbitrary type.	AStream
	filter	Retains only the elements, which comply with a given predicate.	AStream
	window	Windows the stream with a given window assigner.	WindowedAStream
	to	Writes the elements of the stream into a sink.	None
WindowedAStream	groupBy	Groups the stream by a key.	KeyedWindowedAstream
	aggregate	Aggregates the stream in a window.	AStream
	join	Performs a join with a different stream.	AStream
	ajoin	Performs an AJoin with a different stream.	AStream
KeyedWindowedAStream	aggregate	Aggregates the stream by a key in a window.	AStream

Code Example

```
JobManager jobManager = new JobManager();
jobManager.runEngine();
NetworkSource nws1 = new NetworkSource(7001, ...);
NetworkSource nws2 = new NetworkSource(7002, ...);

TopologyBuilder builder = TopologyBuilder.newQuery();

AStream<Tuple3<String,Float,Long>> s1 = builder.streamOf(nws1);
AStream<Tuple3<String,Float,Long>> s2 = builder.streamOf(nws2);

s1.window(TumblingWindow.ofEventTime(Time.seconds(5)))
    .join(s2,
        (t1, t2) -> new Tuple4<>(t1.v1,t1.v2,t2.v2,t1.v3),
        Tuple3::v1,
        Tuple3::v1,
        WatermarkGenerator.seconds(1, 1_000),
        t3 -> t3.v4
        )
    .to(new FileSink("join"));

Query joinQuery = builder.buildAsQuery();
jobManager.addQuery(joinQuery, 50, ChronoUnit.Seconds);
```

Code Example

```
JobManager jobManager = new JobManager();
jobManager.runEngine();
NetworkSource nws1 = new NetworkSource(7001, ...);
NetworkSource nws2 = new NetworkSource(7002, ...);

TopologyBuilder builder = TopologyBuilder.newQuery();

AStream<Tuple3<String,Float,Long>> s1 = builder.streamOf(nws1);
AStream<Tuple3<String,Float,Long>> s2 = builder.streamOf(nws2);

s1.window(TumblingWindow.ofEventTime(Time.seconds(5)))
    .join(s2,
        (t1, t2) -> new Tuple4<>(t1.v1,t1.v2,t2.v2,t1.v3),
        Tuple3::v1,
        Tuple3::v1,
        WatermarkGenerator.seconds(1, 1_000),
        t3 -> t3.v4
        )
    .to(new FileSink("join"));

Query joinQuery = builder.buildAsQuery();
jobManager.addQuery(joinQuery, 50, ChronoUnit.Seconds);
```

Create JobManager and start engine

Code Example

```
JobManager jobManager = new JobManager();
jobManager.runEngine();
NetworkSource nws1 = new NetworkSource(7001, ...);
NetworkSource nws2 = new NetworkSource(7002, ...);

TopologyBuilder builder = TopologyBuilder.newQuery();

AStream<Tuple3<String,Float,Long>> s1 = builder.streamOf(nws1);
AStream<Tuple3<String,Float,Long>> s2 = builder.streamOf(nws2);

s1.window(TumblingWindow.ofEventTime(Time.seconds(5)))
    .join(s2,
        (t1, t2) -> new Tuple4<>(t1.v1,t1.v2,t2.v2,t1.v3),
        Tuple3::v1,
        Tuple3::v1,
        WatermarkGenerator.seconds(1, 1_000),
        t3 -> t3.v4
        )
    .to(new FileSink("join"));

Query joinQuery = builder.buildAsQuery();
jobManager.addQuery(joinQuery, 50, ChronoUnit.Seconds);
```

Define Sources

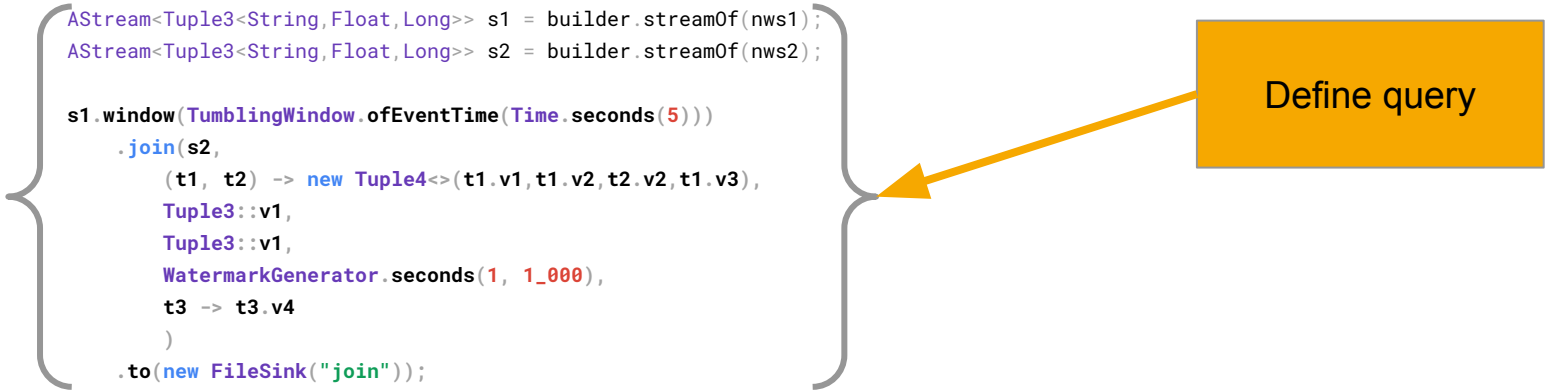
Code Example

```
JobManager jobManager = new JobManager();
jobManager.runEngine();
NetworkSource nws1 = new NetworkSource(7001, ...);
NetworkSource nws2 = new NetworkSource(7002, ...);

TopologyBuilder builder = TopologyBuilder.newQuery();

AStream<Tuple3<String,Float,Long>> s1 = builder.streamOf(nws1);
AStream<Tuple3<String,Float,Long>> s2 = builder.streamOf(nws2);

s1.window(TumblingWindow.ofEventTime(Time.seconds(5)))
    .join(s2,
        (t1, t2) -> new Tuple4<>(t1.v1,t1.v2,t2.v2,t1.v3),
        Tuple3::v1,
        Tuple3::v1,
        WatermarkGenerator.seconds(1, 1_000),
        t3 -> t3.v4
        )
    .to(new FileSink("join"));

Query joinQuery = builder.buildAsQuery();
jobManager.addQuery(joinQuery, 50, ChronoUnit.Seconds);
```

Create new query with TopologyBuilder

Code Example

```
JobManager jobManager = new JobManager();
jobManager.runEngine();
NetworkSource nws1 = new NetworkSource(7001, ...);
NetworkSource nws2 = new NetworkSource(7002, ...);

TopologyBuilder builder = TopologyBuilder.newQuery();

AStream<Tuple3<String,Float,Long>> s1 = builder.streamOf(nws1);
AStream<Tuple3<String,Float,Long>> s2 = builder.streamOf(nws2);

s1.window(TumblingWindow.ofEventTime(Time.seconds(5)))
    .join(s2,
        (t1, t2) -> new Tuple4<>(t1.v1,t1.v2,t2.v2,t1.v3),
        Tuple3::v1,
        Tuple3::v1,
        WatermarkGenerator.seconds(1, 1_000),
        t3 -> t3.v4
        )
    .to(new FileSink("join"));

Query joinQuery = builder.buildAsQuery();
jobManager.addQuery(joinQuery, 50, ChronoUnit.Seconds);
```

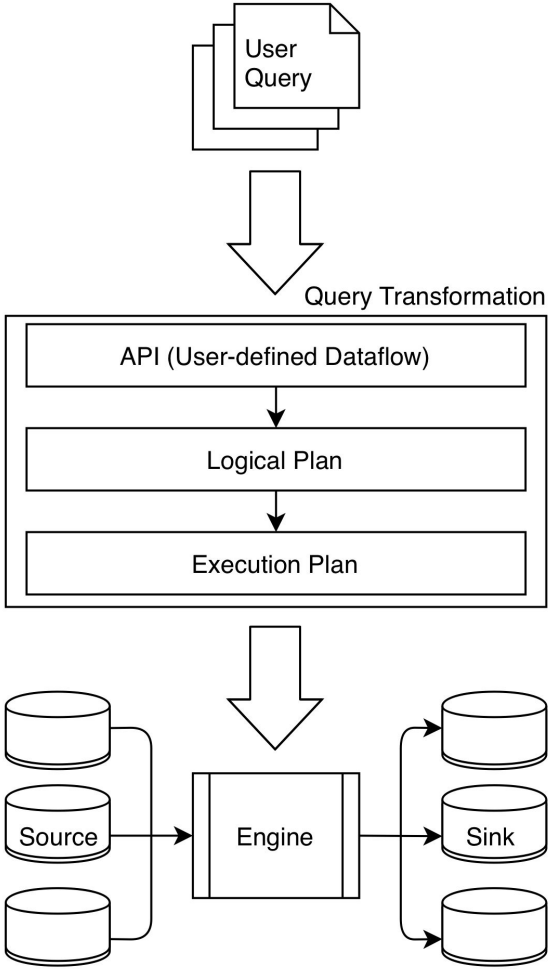


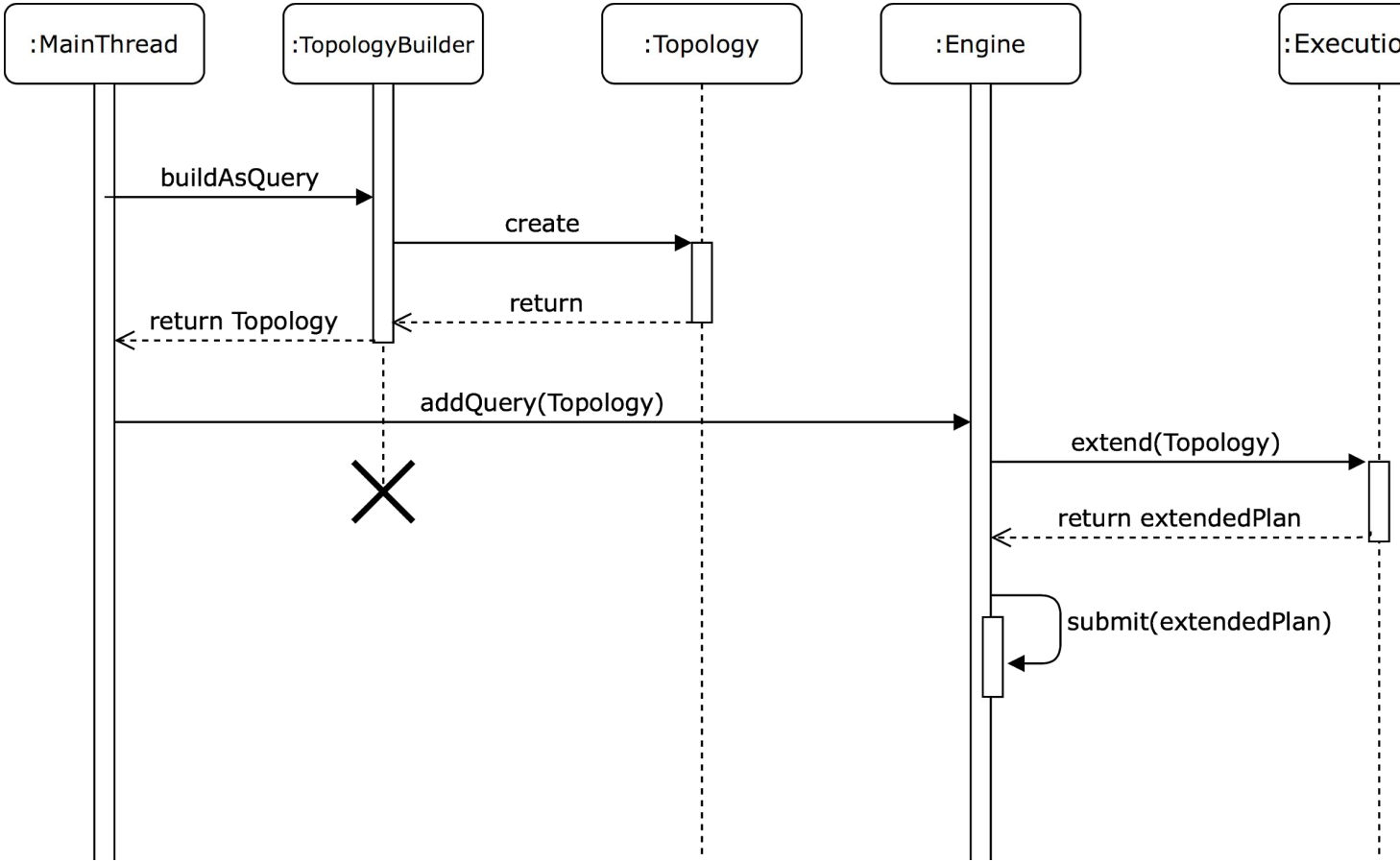
Code Example

```
JobManager jobManager = new JobManager();
jobManager.runEngine();
NetworkSource nws1 = new NetworkSource(7001, ...);
NetworkSource nws2 = new NetworkSource(7002, ...);

TopologyBuilder builder = TopologyBuilder.newQuery();

AStream<Tuple3<String,Float,Long>> s1 = builder.streamOf(nws1);
AStream<Tuple3<String,Float,Long>> s2 = builder.streamOf(nws2);

s1.window(TumblingWindow.ofEventTime(Time.seconds(5)))
    .join(s2,
        (t1, t2) -> new Tuple4<>(t1.v1,t1.v2,t2.v2,t1.v3),
        Tuple3::v1,
        Tuple3::v1,
        WatermarkGenerator.seconds(1, 1_000),
        t3 -> t3.v4
        )
    .to(new FileSink("join"));

Query joinQuery = builder.buildAsQuery();
jobManager.addQuery(joinQuery, 50, ChronoUnit.Seconds);
```

Build and submit query

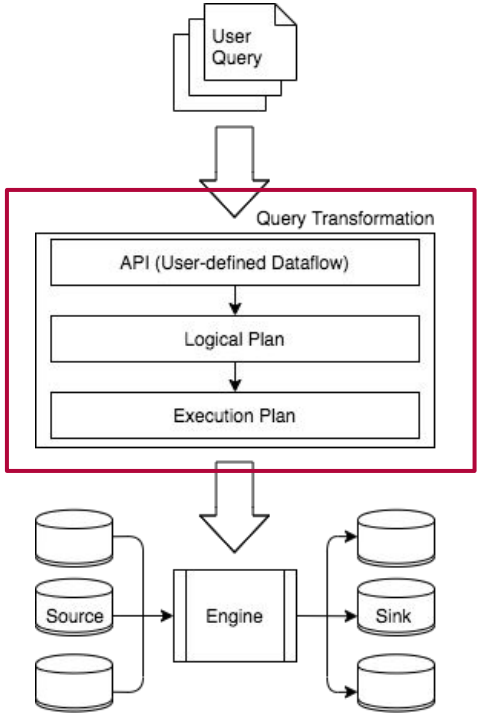
3. Architecture Overview


User
Query
Query Transformation
API (User-defined Dataflow)
Logical Plan
Execution Plan
Source
Engine
Sink


:MainThread
:TopologyBuilder
:Topology
:Engine
:ExecutionPlan
buildAsQuery
create
return
return Topology
addQuery(Topology)
extend(Topology)
return extendedPlan
submit(extendedPlan)

4. Query Transformation

Transformation Pipeline



Operators

Source:

- Read from a source
- Attaches metadata

OneInputOperator:

- Transform a single event into *n* new events

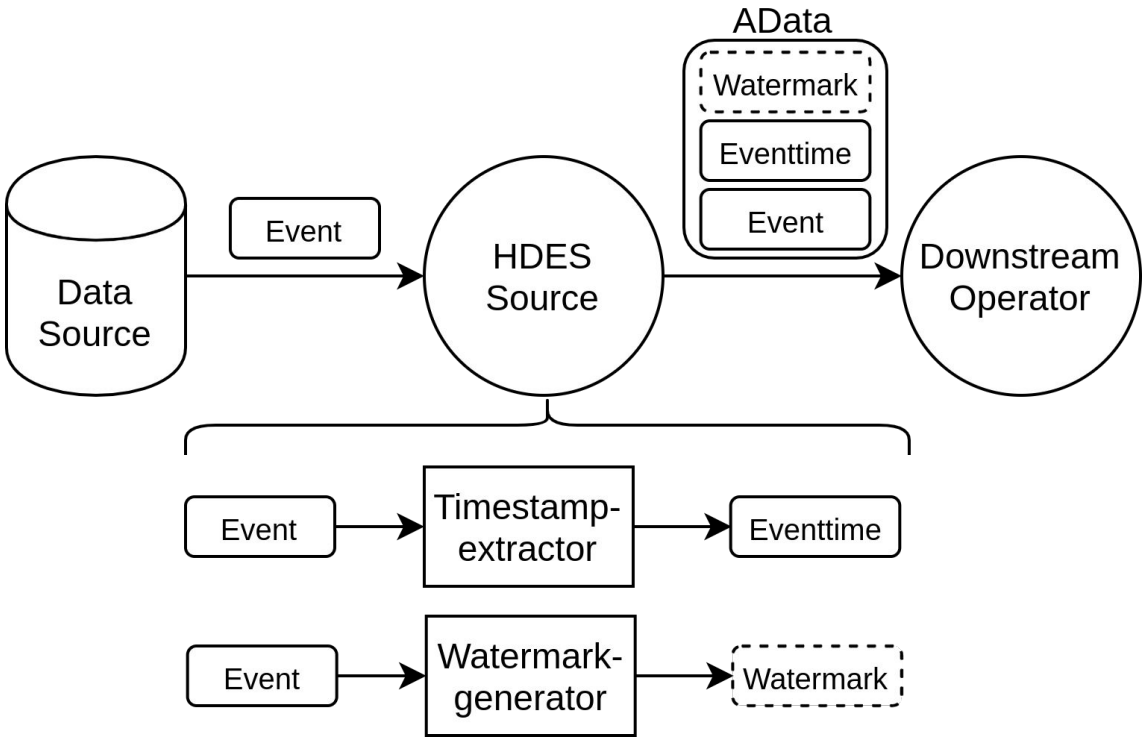
TwoInputOperator:

- Transform events from two different origins

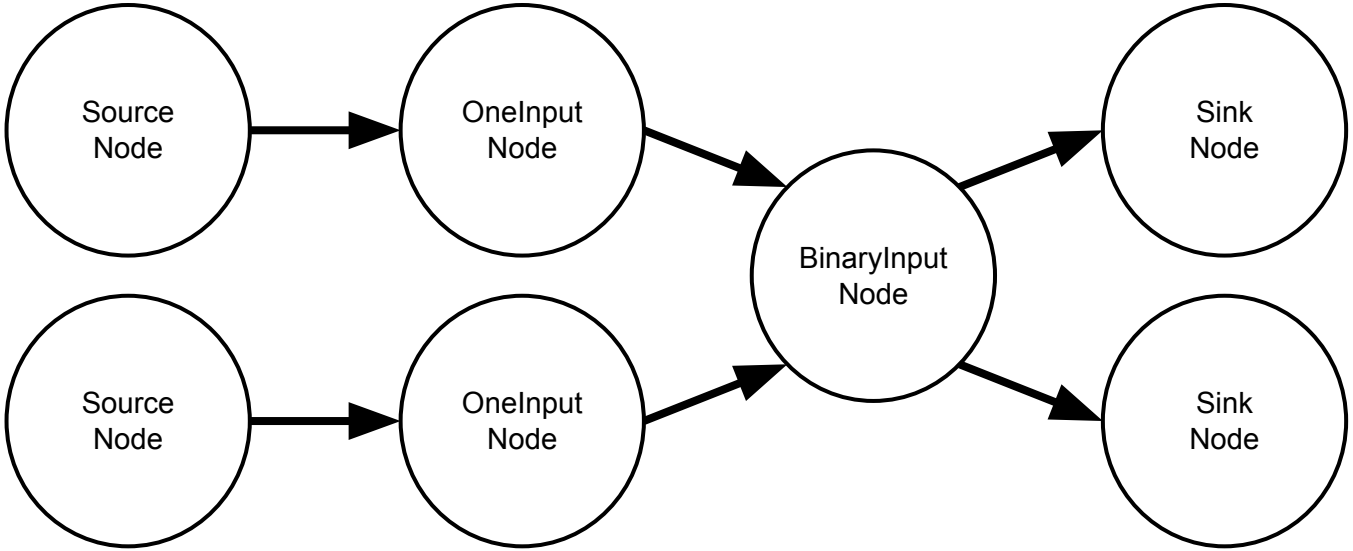
Sink:

- Write to a sink

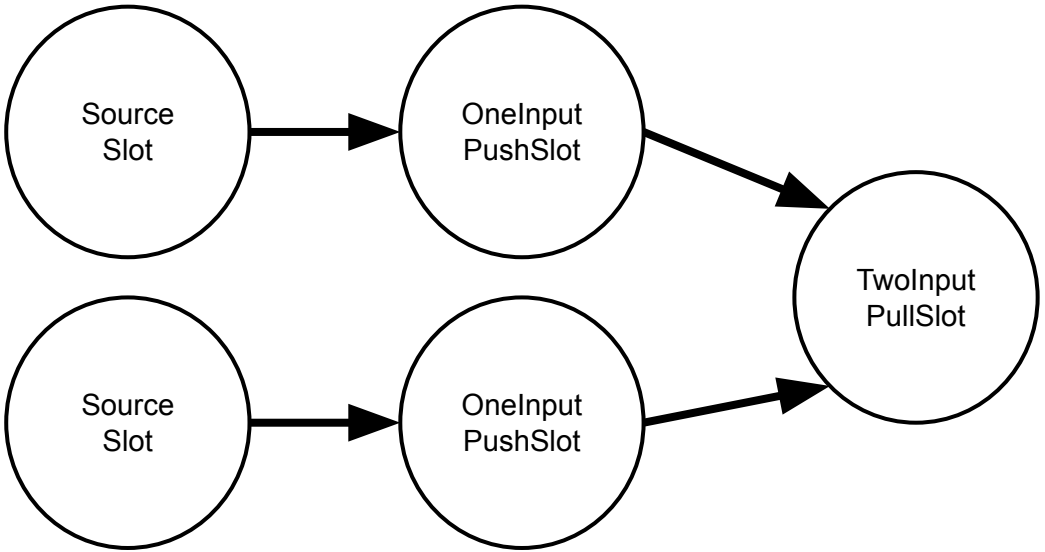
Source Operator



Logical Plan



Execution Plan



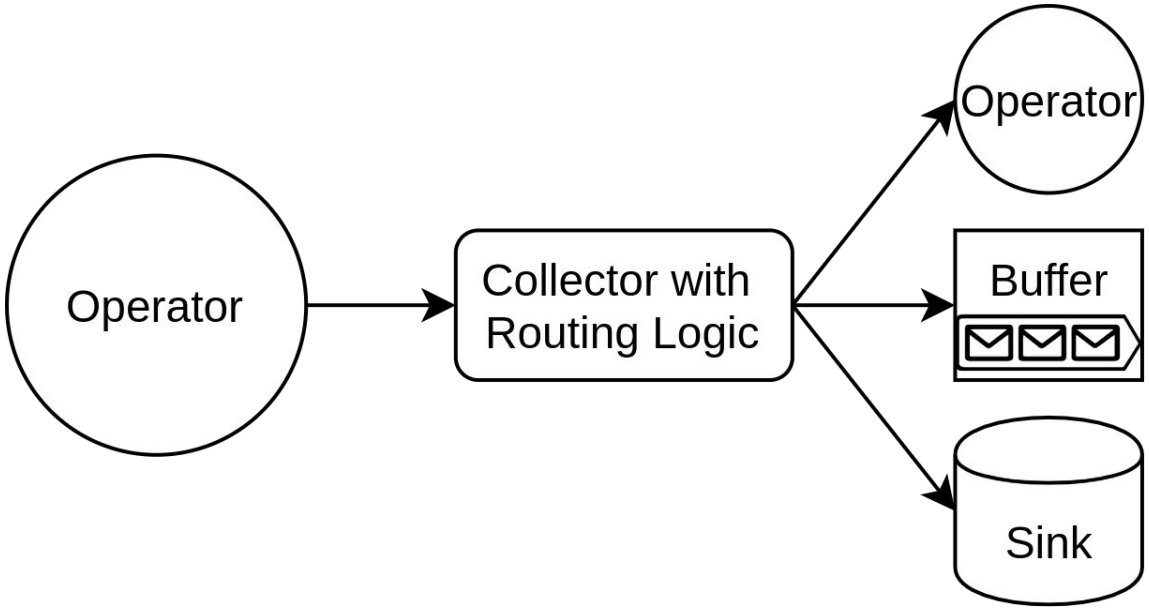
Transformation Properties

Layered architecture decouples query definition and execution.

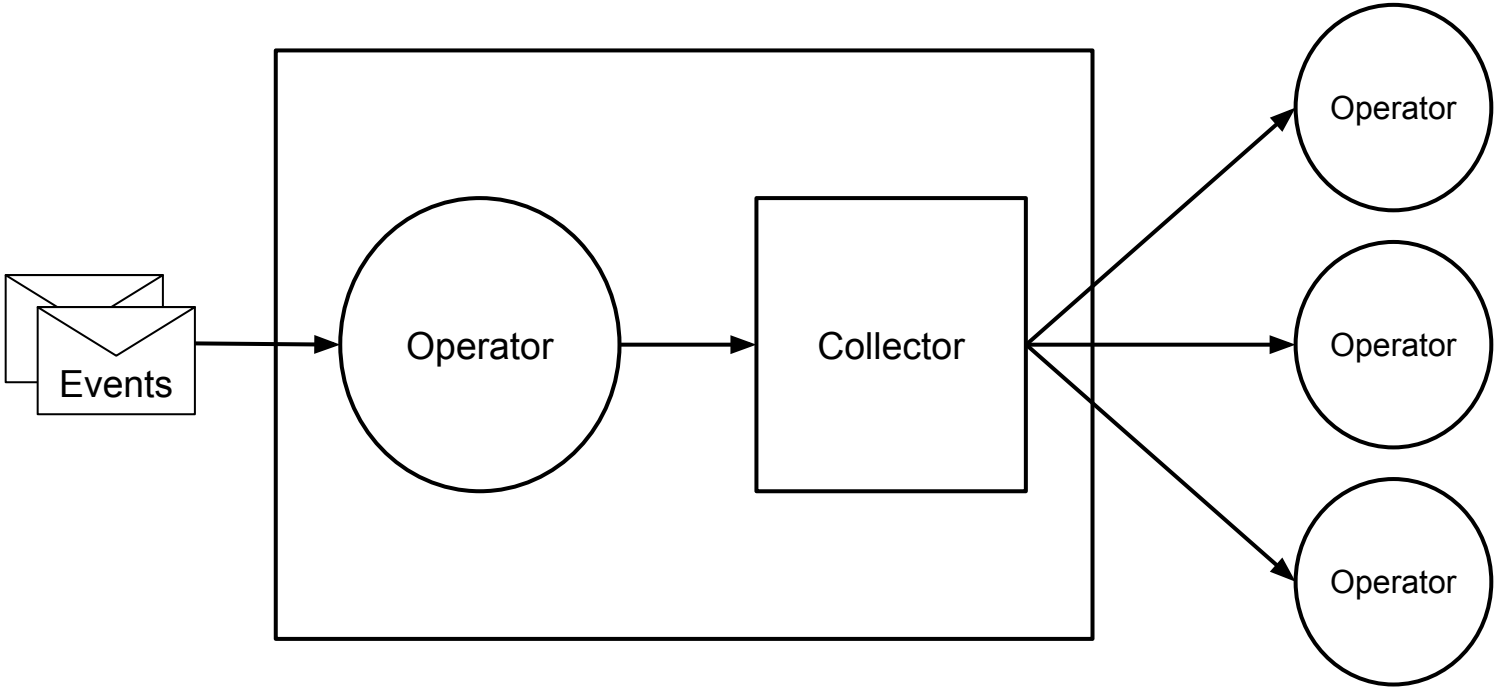
- Interchangeable query definition
- Interchangeable Execution Plan

5. Query Execution

Routing

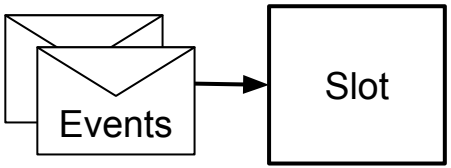

Operator
Collector with
Routing Logic
Operator
Buffer
Sink

Slot

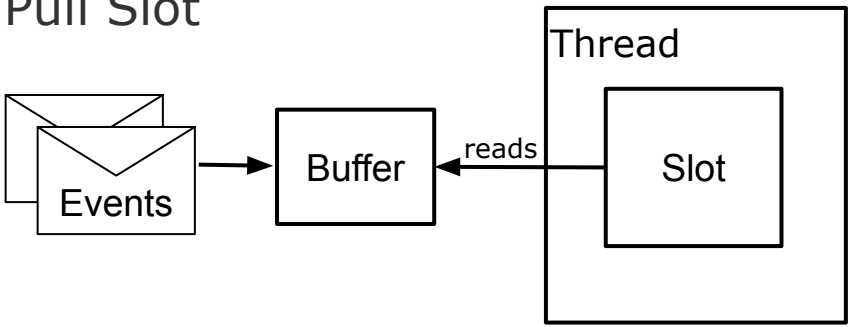


Slot Types

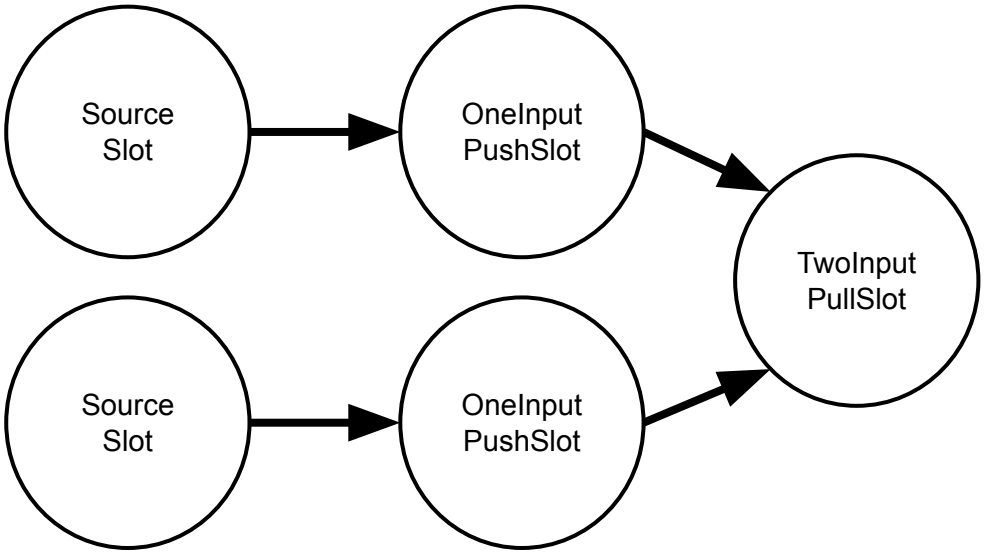
Push Slot



Pull Slot

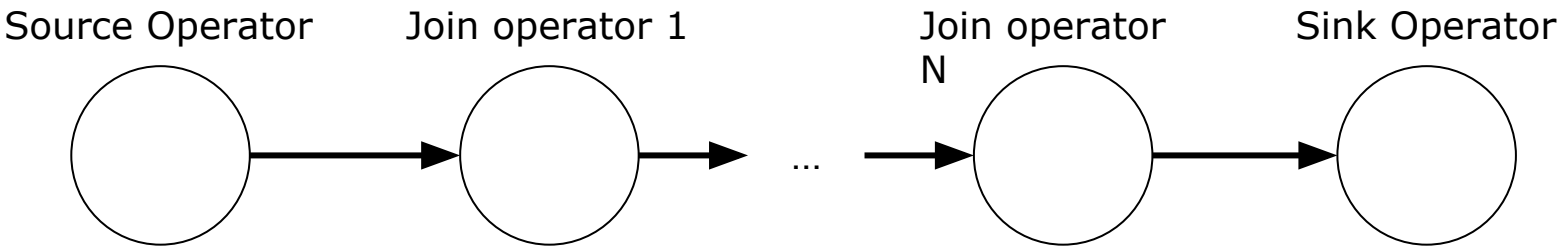


Execution Plan

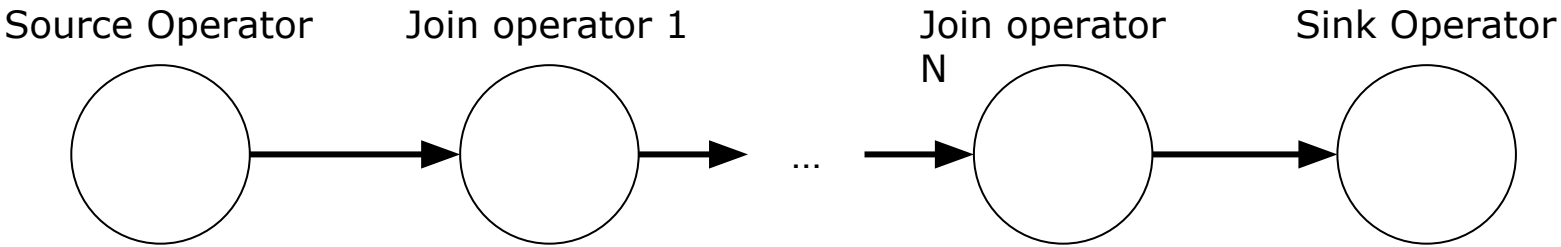


6. Ad-hoc join processing

Efficient Distributed Join Architecture

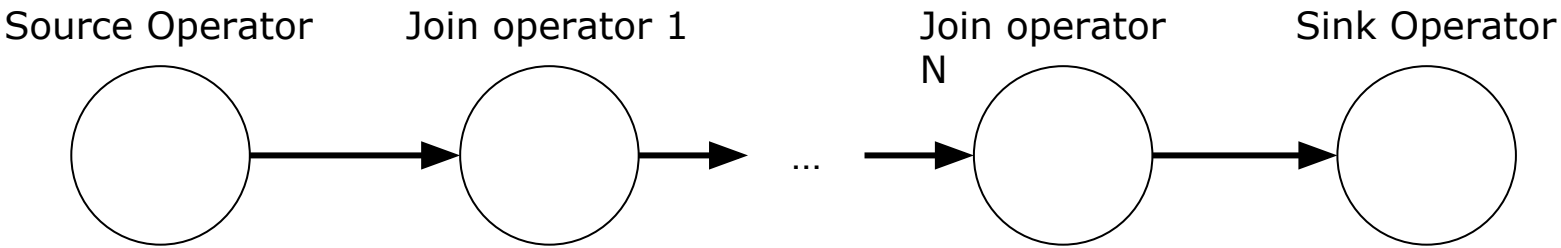


Efficient Distributed Join Architecture



- Indexing
- Windowing

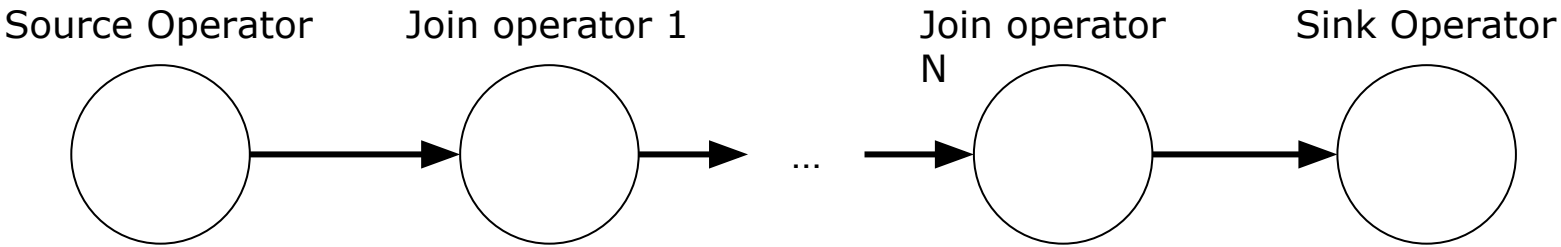
Efficient Distributed Join Architecture



- Indexing
- Windowing

- Set intersection of join index

Efficient Distributed Join Architecture

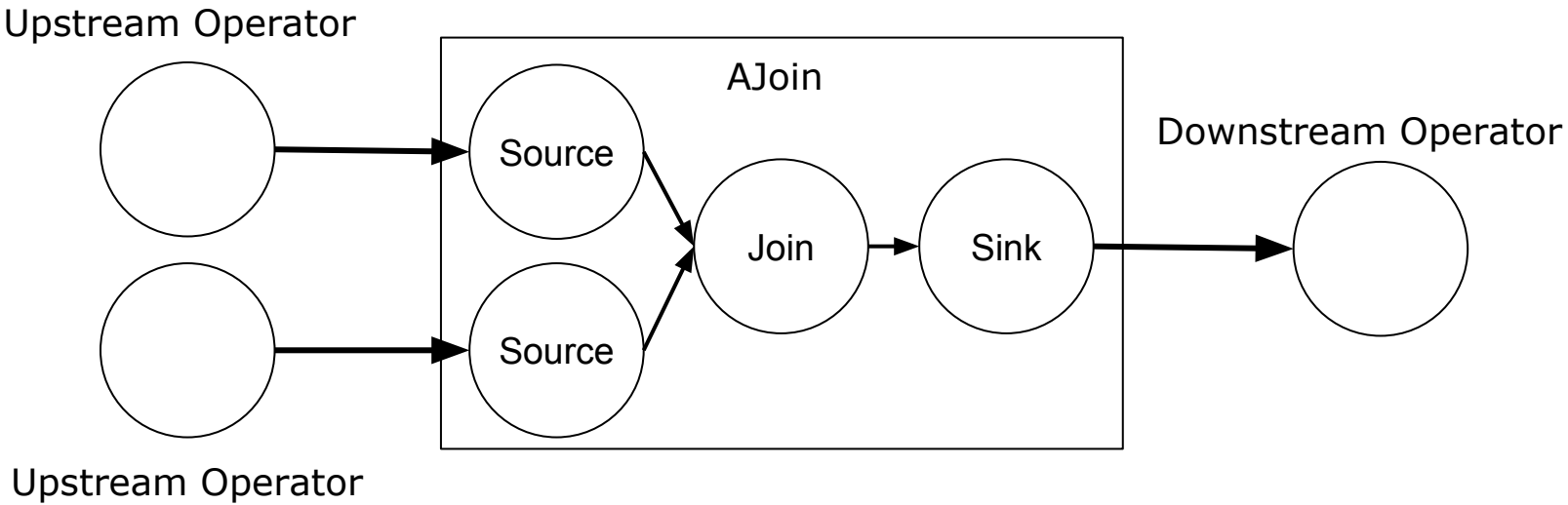


- Indexing
- Windowing

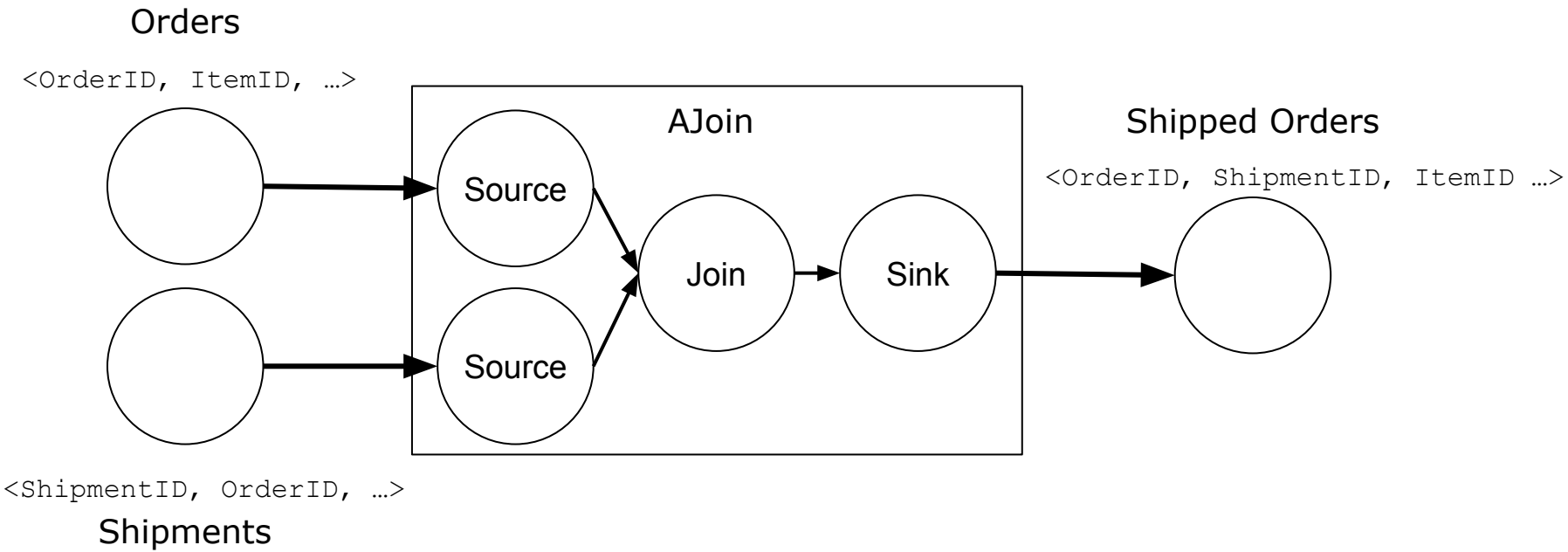
- Set intersection of join index

- Joins matching tuples
- Pushes to output channels

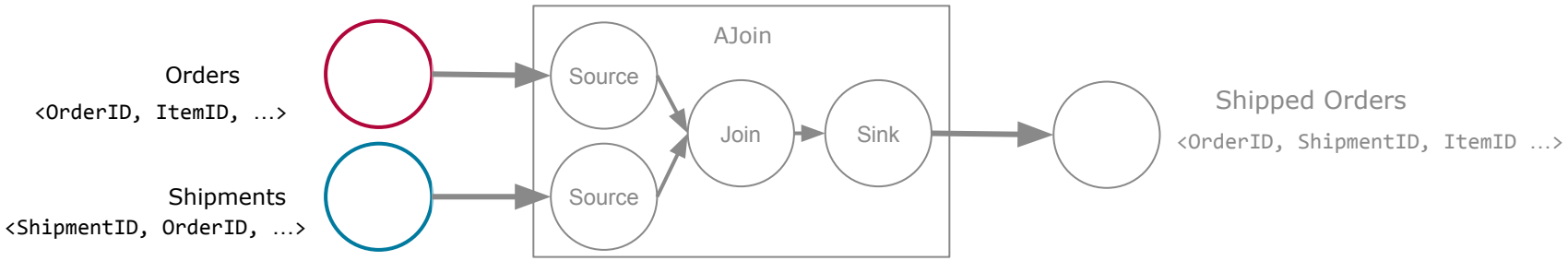
AJoin in HDES



HDES AJoin Example



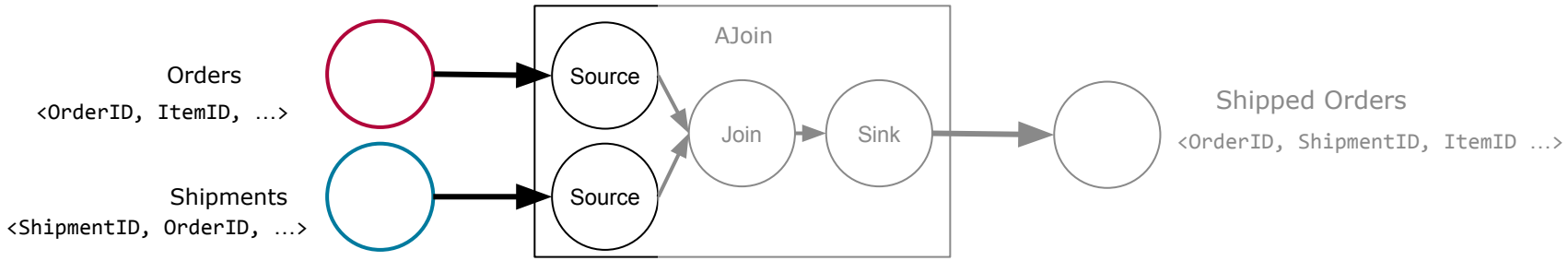
HDES AJoin Example



<1, 5,...> <1, 8,...> <4, 7,...> <5, 2,...> <5, 7,...>

<5, 7,...> <6, 8,...> <9, 1,...> <6, 4,...> <3, 5,...>

HDES AJoin Example



<1, 5,...> <1, 8,...> <4, 7,...> <5, 2,...> <5, 7,...>

<5, 7,...> <6, 8,...> <9, 1,...> <6, 4,...> <3, 5,...>

1 ← [<1, 5,...>, <1,8,...>]

4 ← [<4,7,...>]

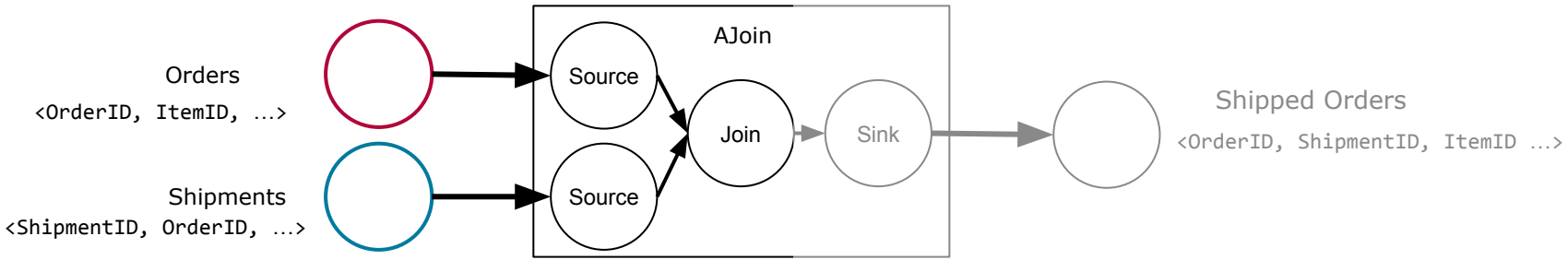
5 ← [<5,2,...>, <5,7,...>]

Orders Bucket

7 ← [<5, 7,...>]
8 ← [<6, 8,...>]
1 ← [<9, 1,...>]
4 ← [<6, 4,...>]
5 ← [<3, 5,...>]

Shipment Bucket

HDES AJoin Example



`1 ← [<1, 5,...>, <1,8,...>]|4 ← [<4,7,...>]|5 ← [<5,2,...>, <5,7,...>]`

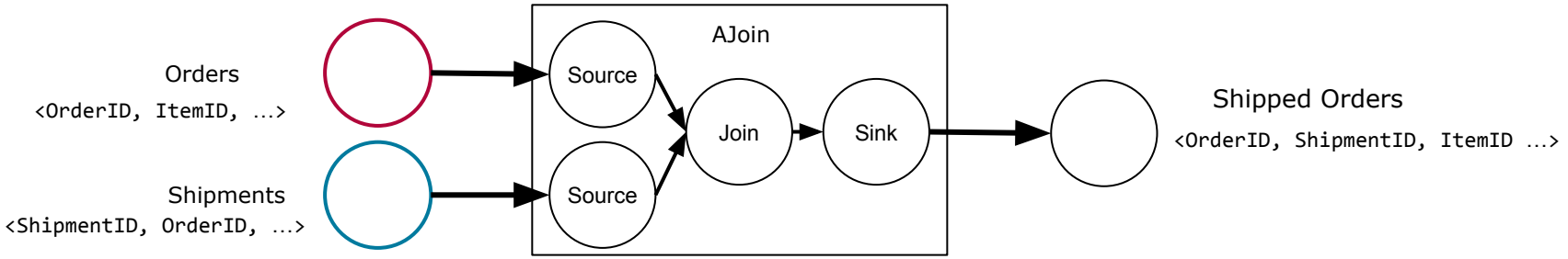
`7 ← [<5, 7,...>]|8 ← [<6, 8,...>]|1 ← [<9, 1,...>]|4 ← [<6, 4,...>]|5 ← [<3, 5,...>]`

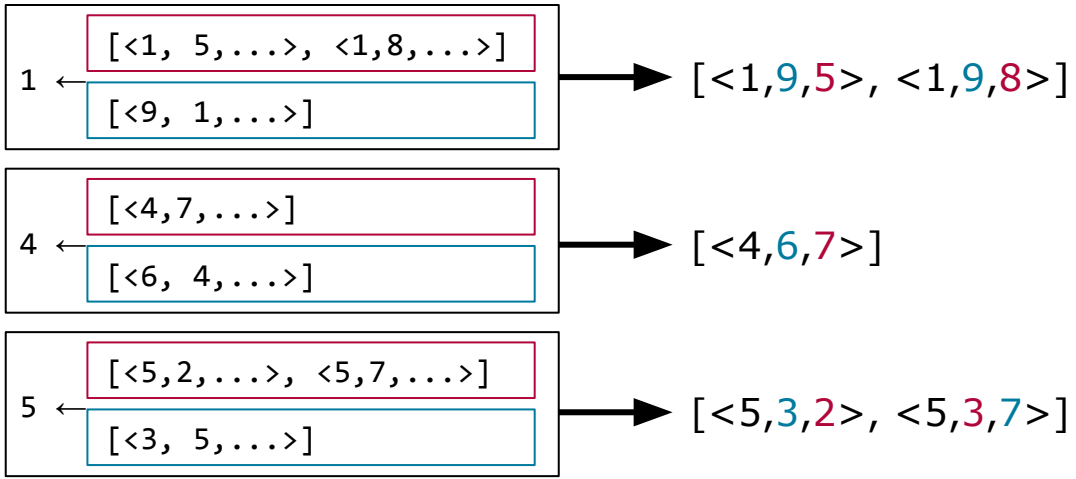
1 ←
- `[<1, 5,...>, <1,8,...>]`
- `[<9, 1,...>]`

5 ←
- `[<5,2,...>, <5,7,...>]`
- `[<3, 5,...>]`

4 ←
- `[<4,7,...>]`
- `[<6, 4,...>]`

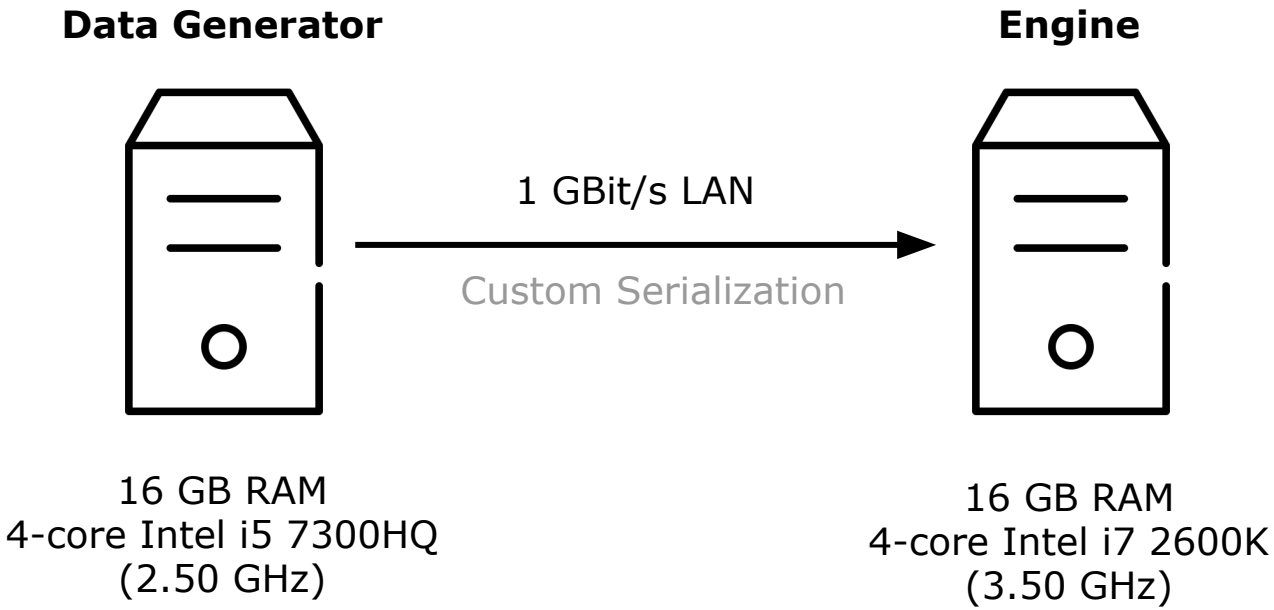
HDES AJoin Example





7. Benchmarking Results

Benchmarking Setup



Description of Metrics

Timestamps:

Event Time = The moment the tuple is **created**
Processing Time = The moment the tuple **enters** the **engine**
Ejection Time = The moment the tuple **enters** the **sink**

Latency:

Event Time Latency = Ejection Time - Event Time
Processing Time Latency = Ejection Time - Processing Time

Other metrics:

Throughput = Amount of tuples send from the data-generator per second

Data Sets - "Basic" and "Nexmark-Light"

Legend:
Long
Integer

Tuple (12 Bytes)
(EventTime, Number)

Auction (28 Bytes)
(AuctionId, Quantity, Type, Reserve, EventTime)

Bid (32 Bytes)
(BidId, AuctionId, BetterId, Price, EventTime)

Basic

Simplistic test-data

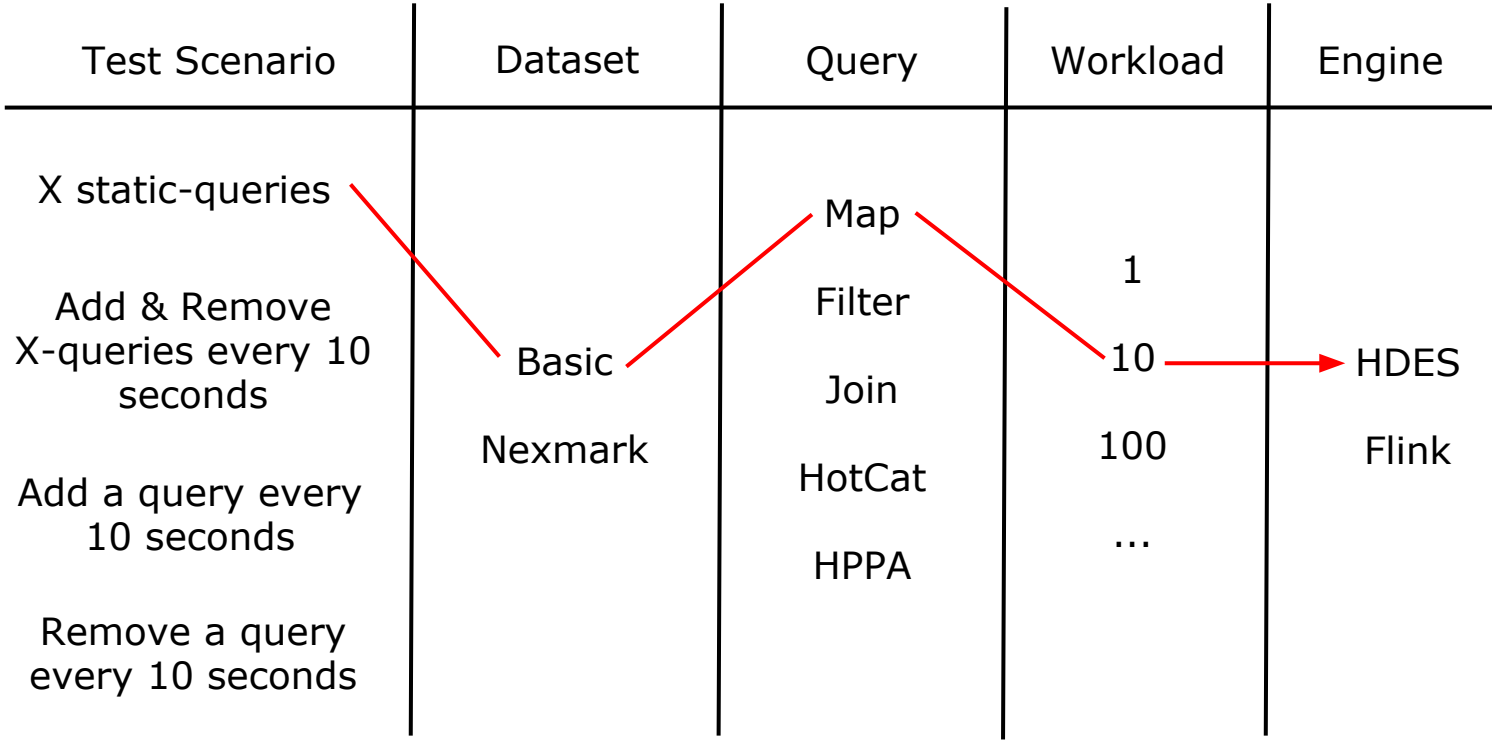
Nexmark-Light

More realistic data an auction house use-case

Benchmark Dimensions

Test Scenario	Dataset	Query	Workload	Engine
X static-queries	Basic Nexmark	Map Filter Join HotCat HPPA	1 10 100 ...	HDES Flink
Add & Remove X-queries every 10 seconds				
Add a query every 10 seconds				
Remove a query every 10 seconds				

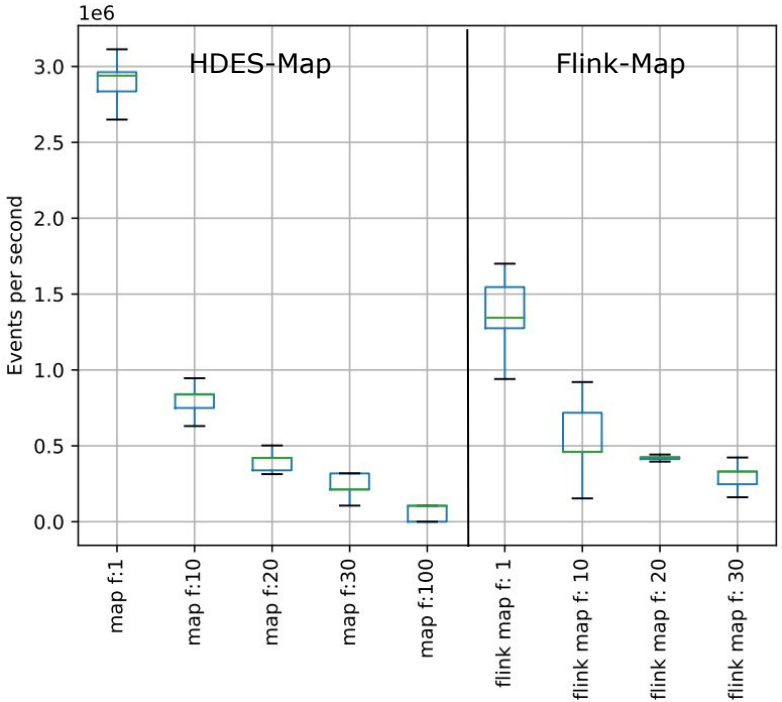
Benchmark Dimensions



Scenario: 10 fixed map-queries executed with HDES and the basic dataset

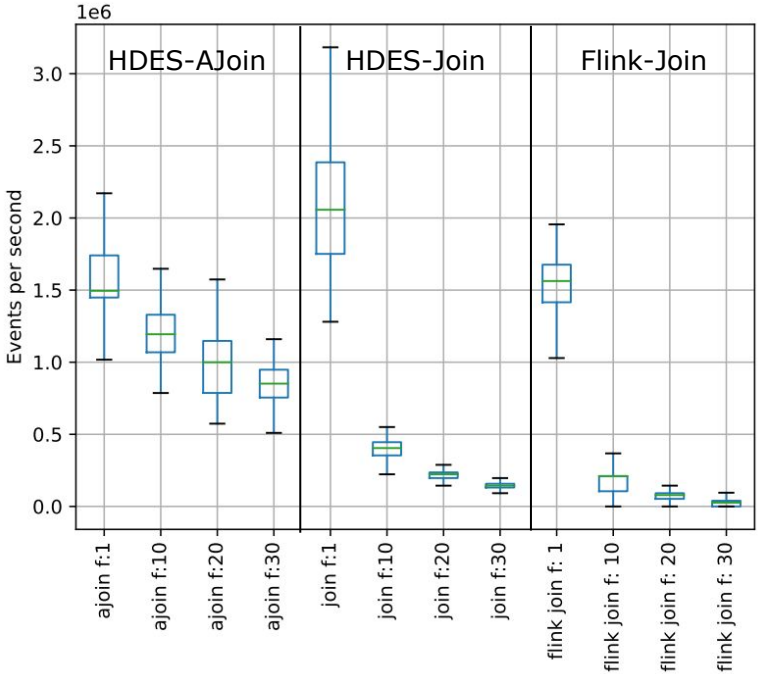
Static Query Execution

Static X-Queries - HDES Map vs. Flink Map



HDES can outperform Flink with simple operations across datasets

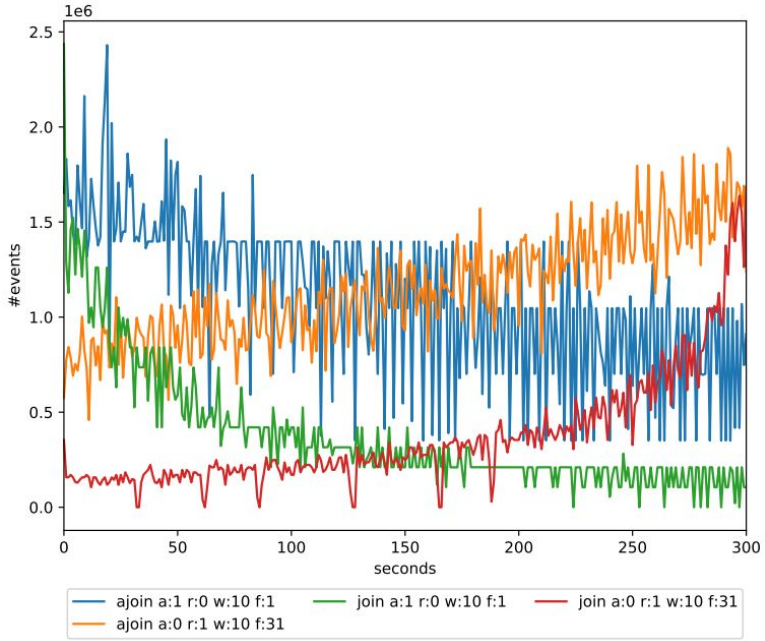
Static X-Queries - HDES (A)Join vs. Flink Join



AJoin is better than Join in multi-query situations and HDES outperforms Flink's Join in every scenario

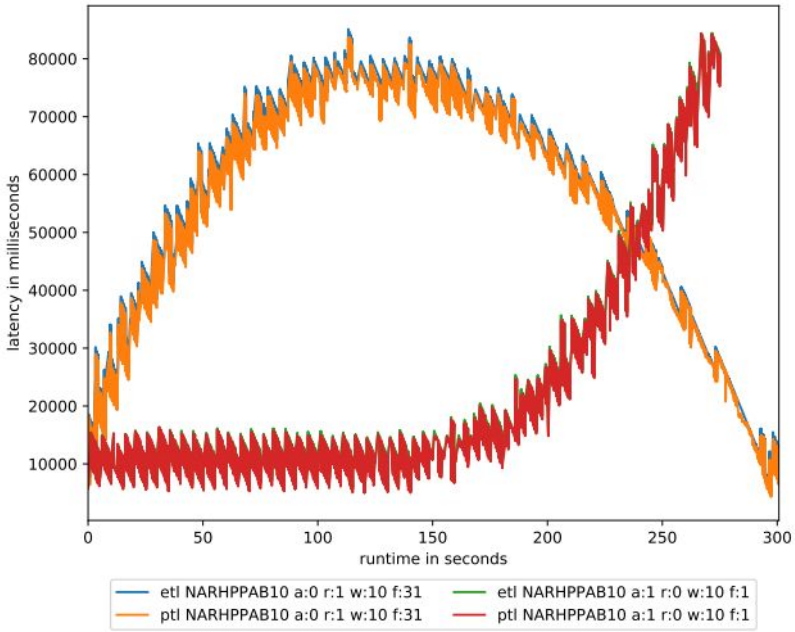
Dynamic Query Execution

Dynamic - Increased Adding & Removing



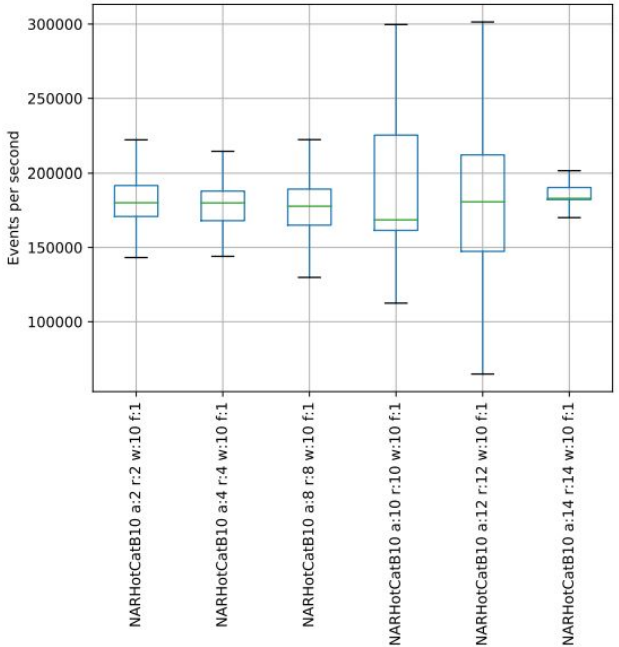
Basic - AJoin & Join

Dynamic - Increased Adding & Removing



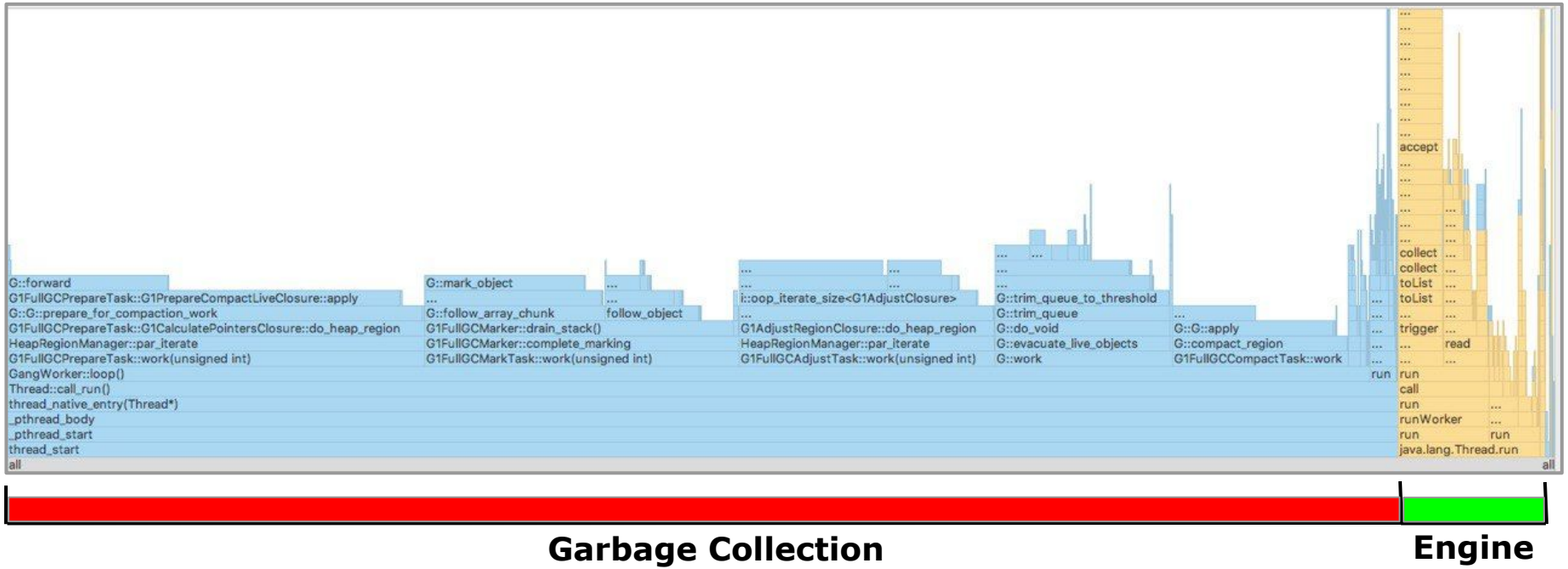
Nexmark - HPPA

Dynamic - Constant Adding & Removing

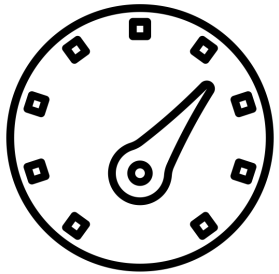


Nexmark - Hottest Category

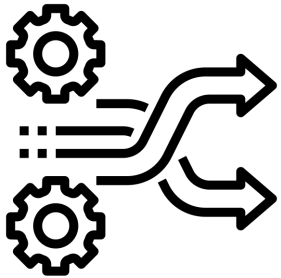
Garbage Collection



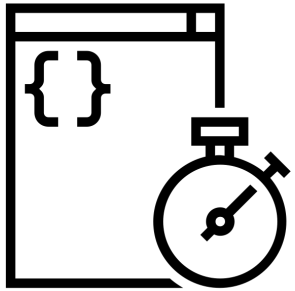
Conclusion



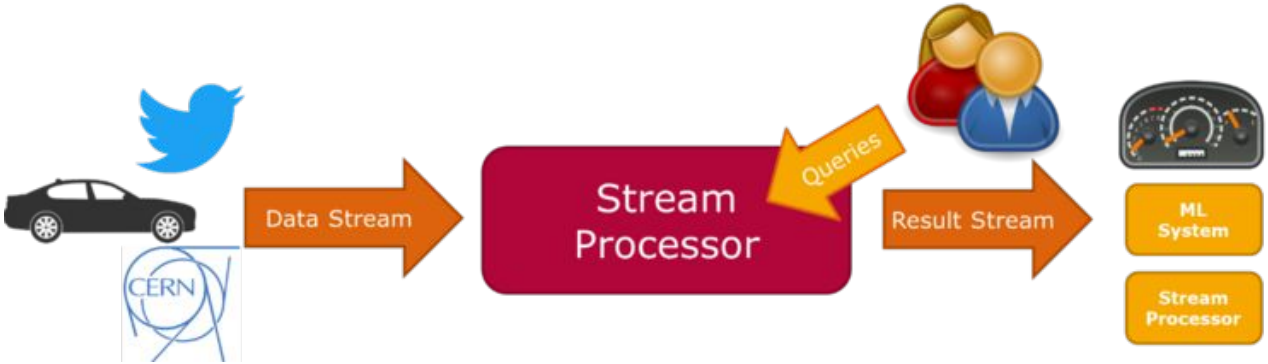
HDES **outperforms** Flink in most situations and can even compute **larger workloads**



HDES offers **ad-hoc functionality** as a first-class citizen



HDES offers integration of **optimized sharing techniques** from previous work



Thank you for your attention!

Nico Duldhardt, Torben Meyer, Marvin Thiele, Anton von Weltzien
Masterprojekt WS 19/20
Data Engineering Systems

Agenda

1. Goals (Anton) ~ 1
2. Features (Anton) ~ 2-4
3. Architecture Overview (Anton) 2-3
4. Query Transformation (Nico) 4
5. Query Execution (Nico) 3
6. Multi-Query Execution (Torben) > 7 min
7. Benchmark Results (Marvin) >7 min

Dynamic - Scalability of Join & AJoin