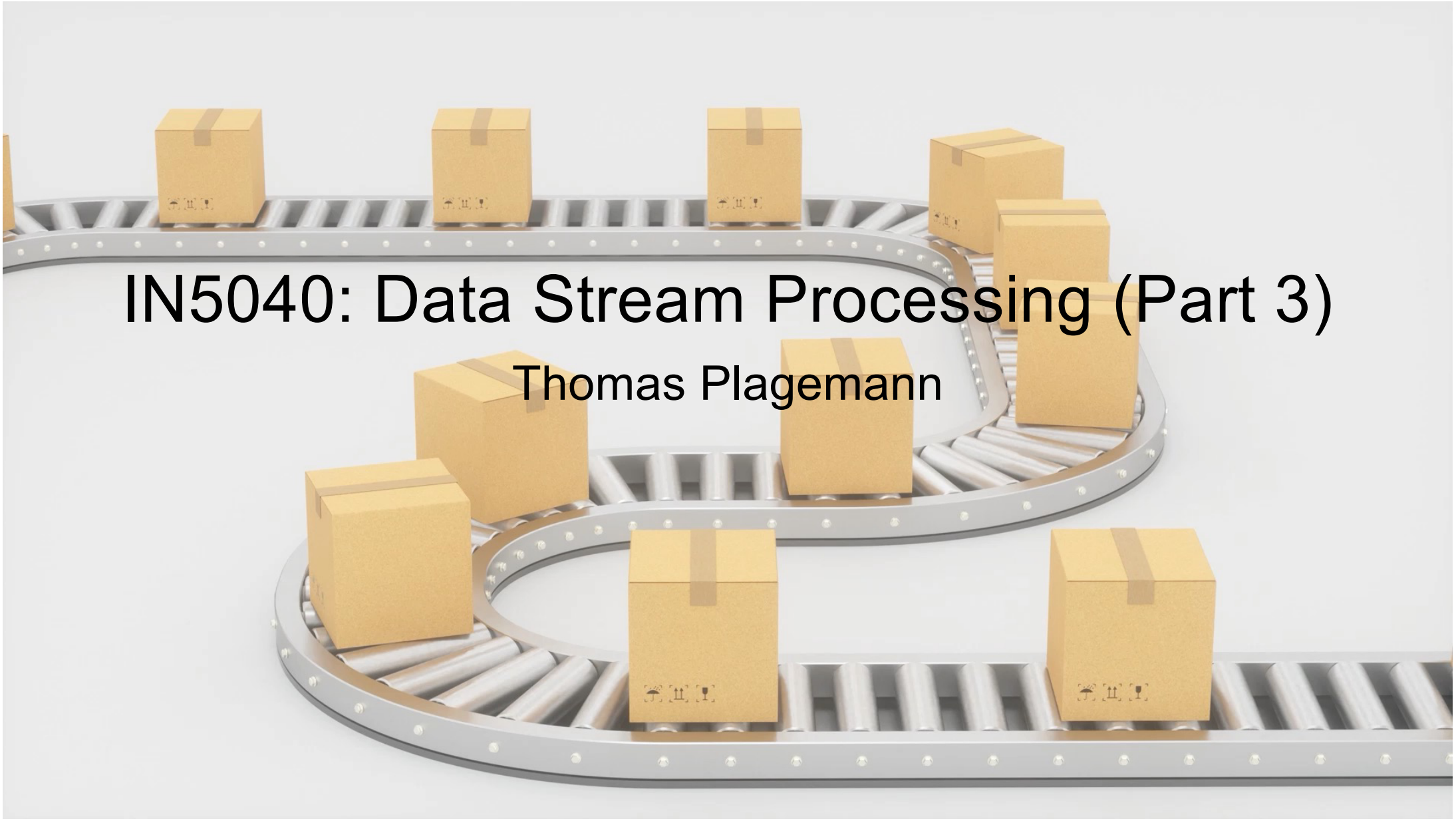


IN5040: Data Stream Processing (Part 3)

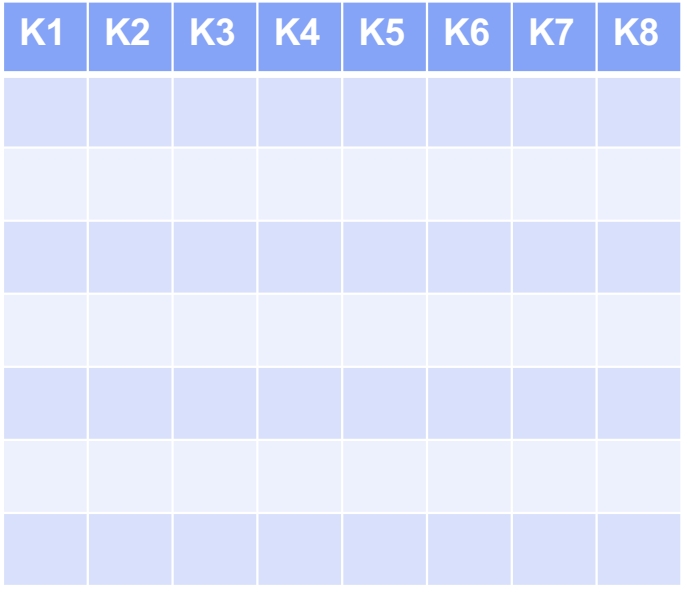
Thomas Plagemann

Apache Flink in Cloud Environments - Storage

- State-backends for Flink
  - Manage state
  - Checkpointing
- Pluggable file systems
  - Amazon S3
  - Aliyun Object Storage Service
  - Azur Data Lake Store Gen 2
  - Azur Blob Storage
  - Google Cloud Storage

Apache Flink in Cloud Environments - CPU

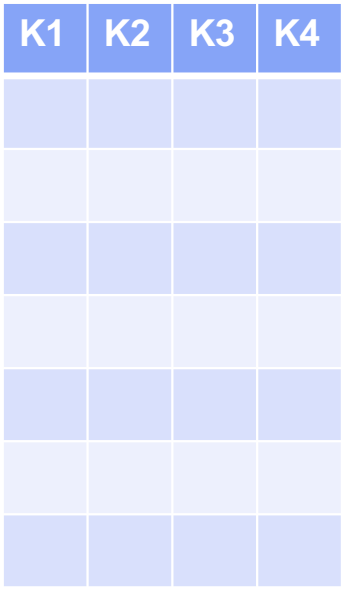
Host 1



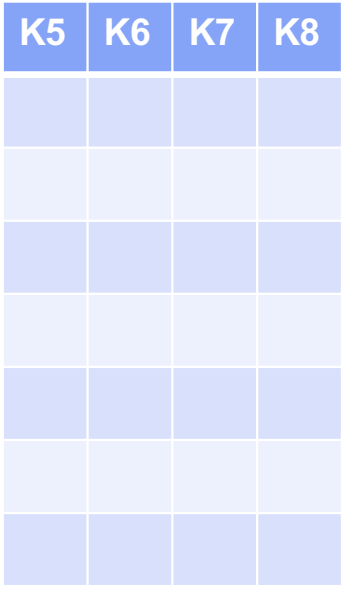
Overload

- scale out
- Extract some state
- Migrate it to other host(s)
- Update operator graph

Host 2



Host 1



UNIVERSITY OF OSLO

Semantic and utility aware operator migration for Distributed Stream Processing

Thomas Plagemann

UiO : University of Oslo

Espen Volnes

Distributed Stream Processing: Performance Evaluation and Enhanced Operator Migration

Thesis submitted for the degree of Philosophiae Doctor

Department of Informatics
Faculty of Mathematics and Natural Sciences

2023

In collaboration with Vera Goebel, Boris Koldehofe, Thomas Plagemann

Supported by the Parrot Project (Norwegian Research council, project number 311197)

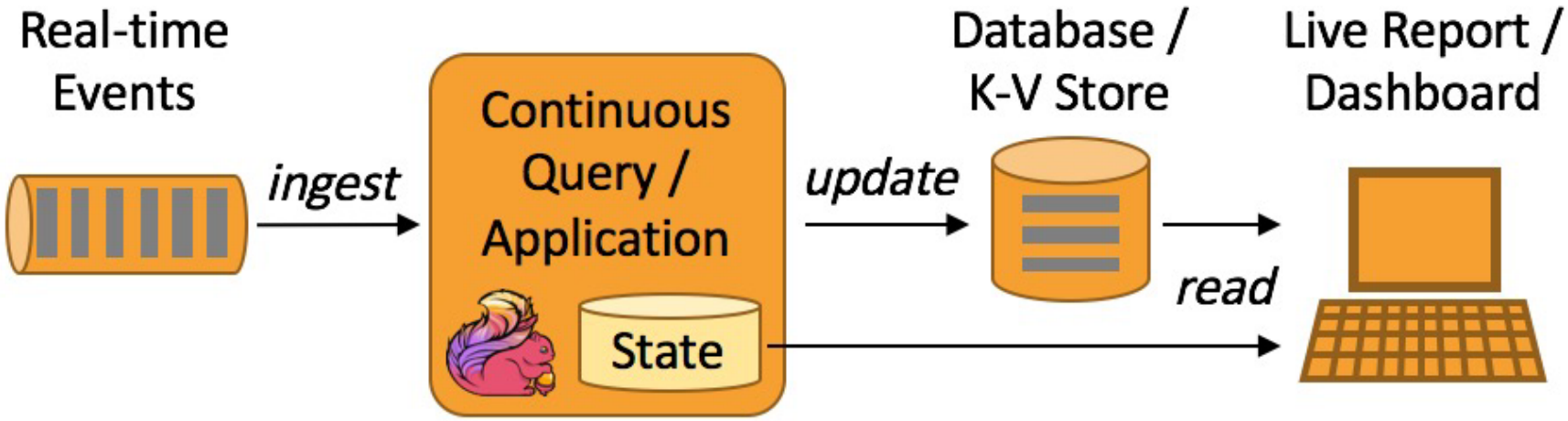
Outline

- Background
  - Stream processing
  - Operator migration
- Core idea
- Implementation
- Evaluation
- Conclusions

Data stream sources and applications

[Source: https://www.icterra.com/iot-is-coming-over-but-are-we-ready-for-it/]

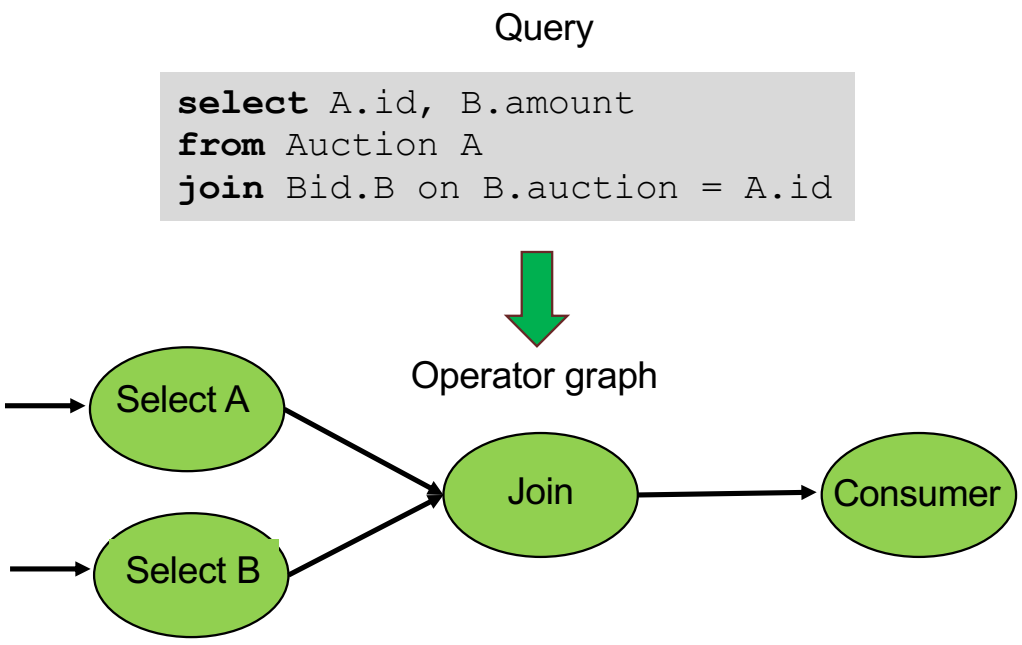
Data stream processing

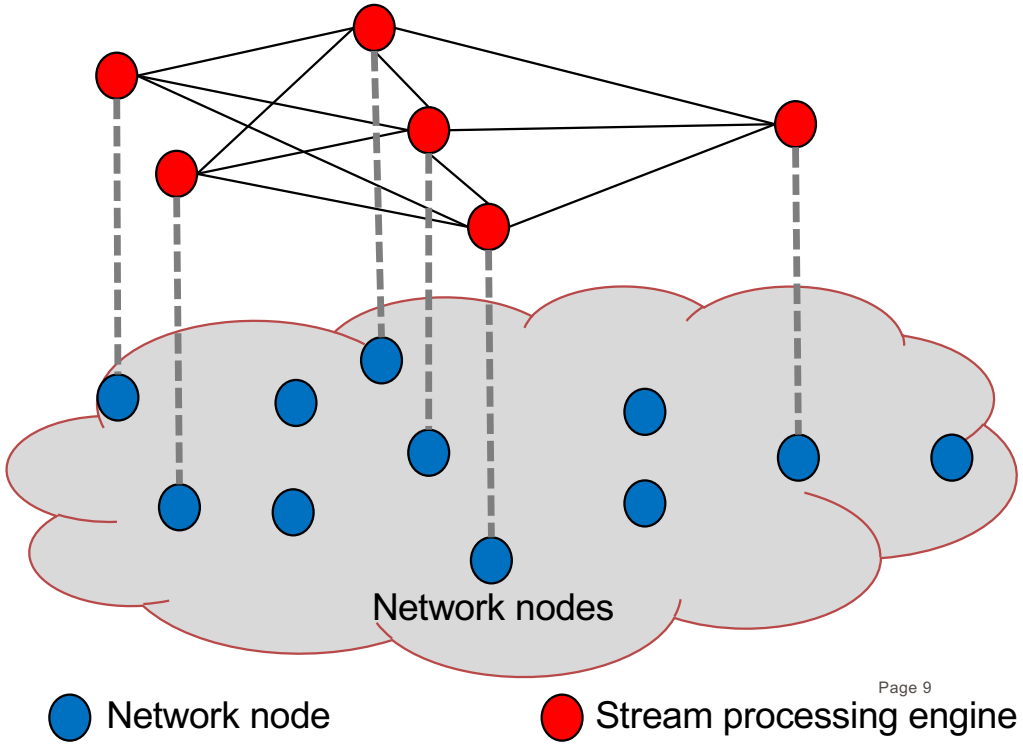


[Source: https://dz2cdn1.dzone.com/storage/temp/11630297-batch-stream-processing-apache-flink-real-time-dat.png]

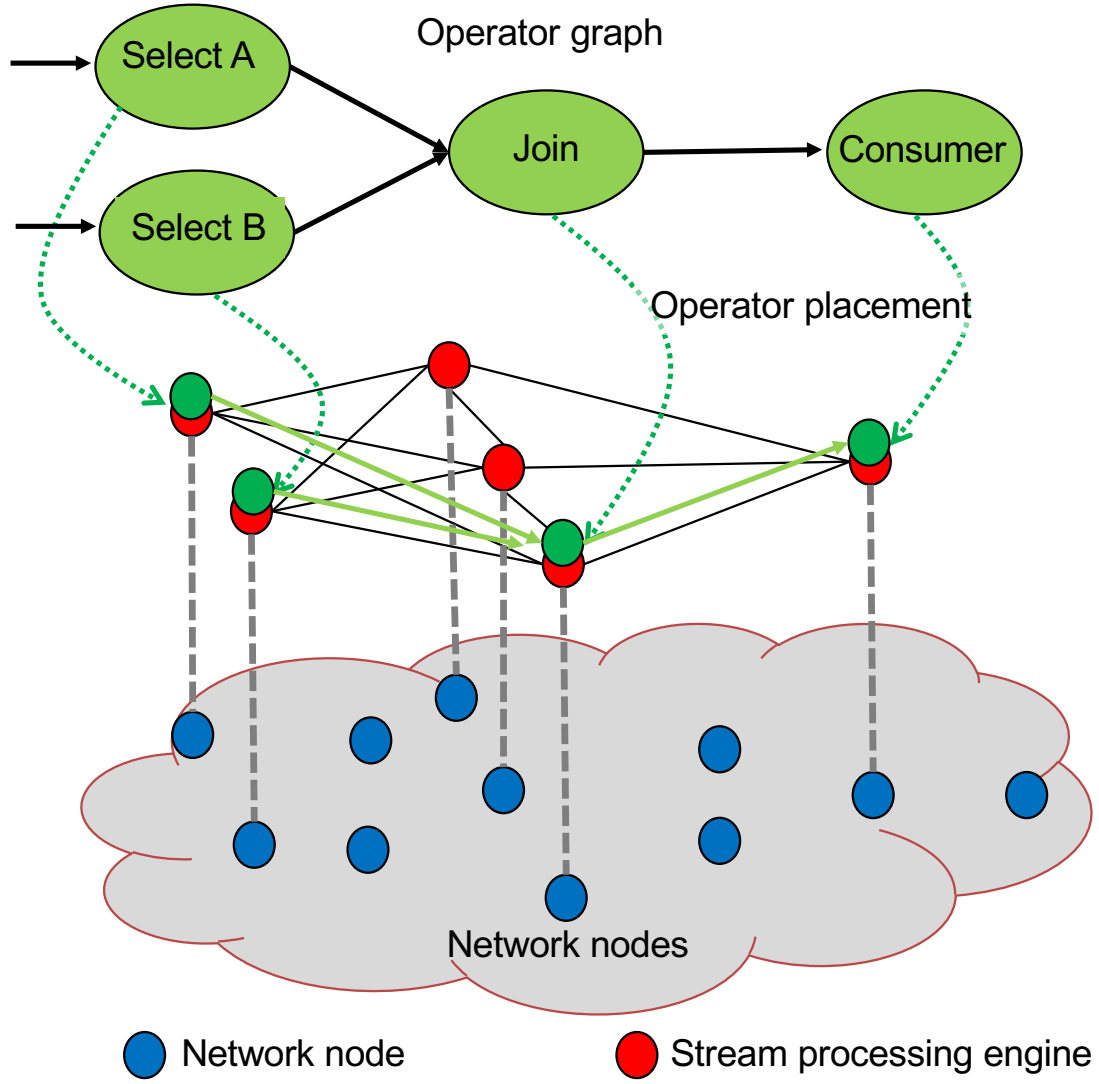
Distributed stream processing

- Distribute the load over multiple networked nodes
- Operator the basic “unit” of distribution
- Query → operator graph

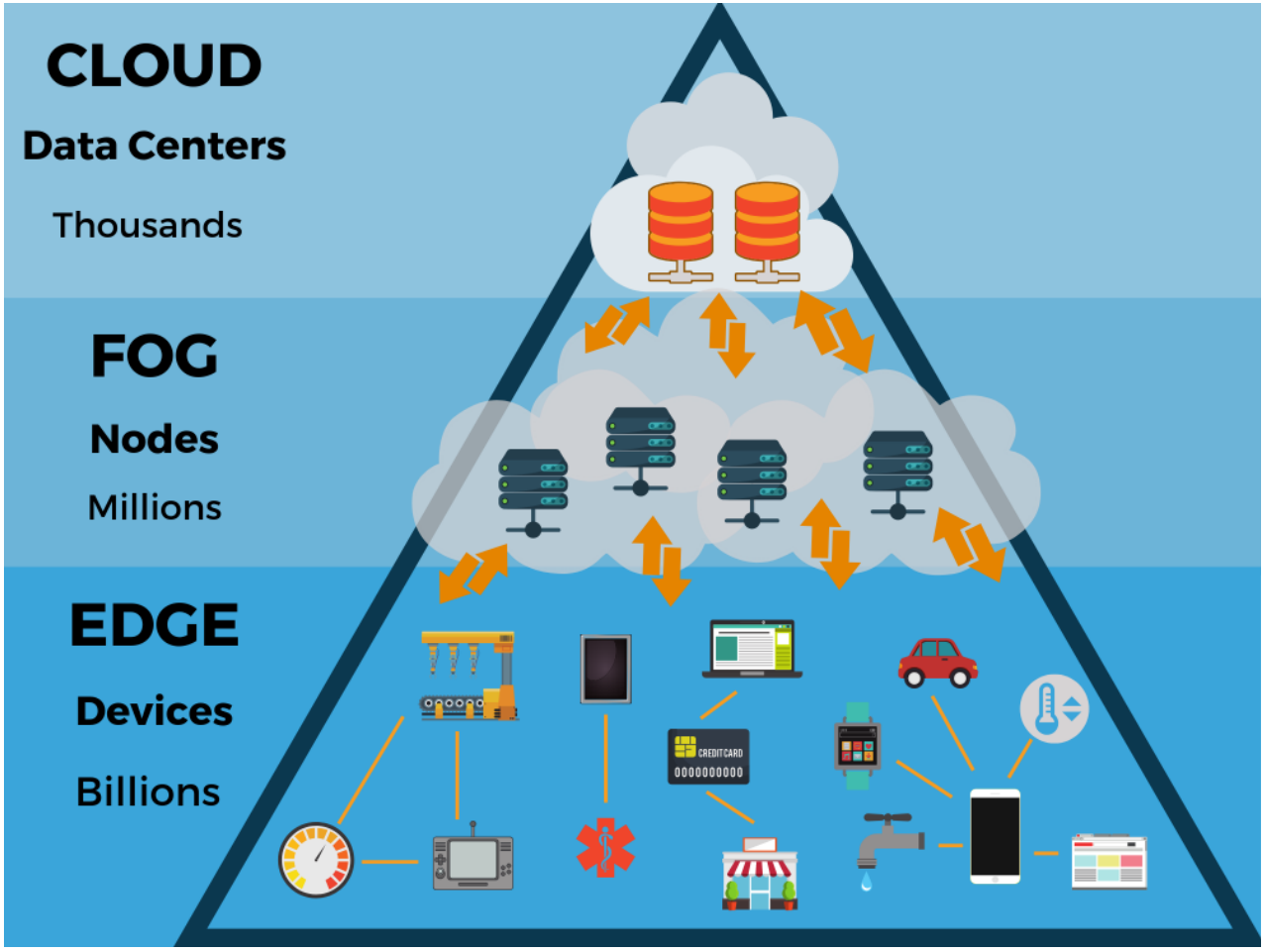




Initial placement



Processing environments



Adaptation

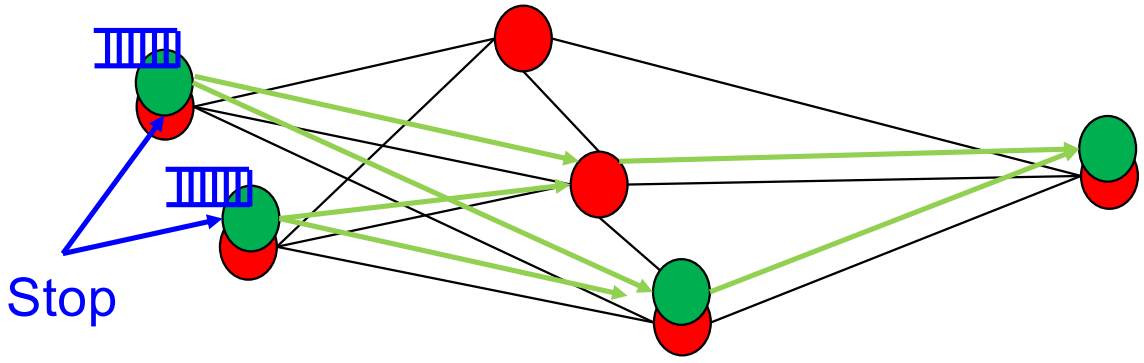
- Queries are long running
- Amount of streaming data can change
- Load of operator hosts can change
- Network QoS can change
- Mobile nodes can change the location

⇒ Adapt distributed stream processing overlay through operator migration

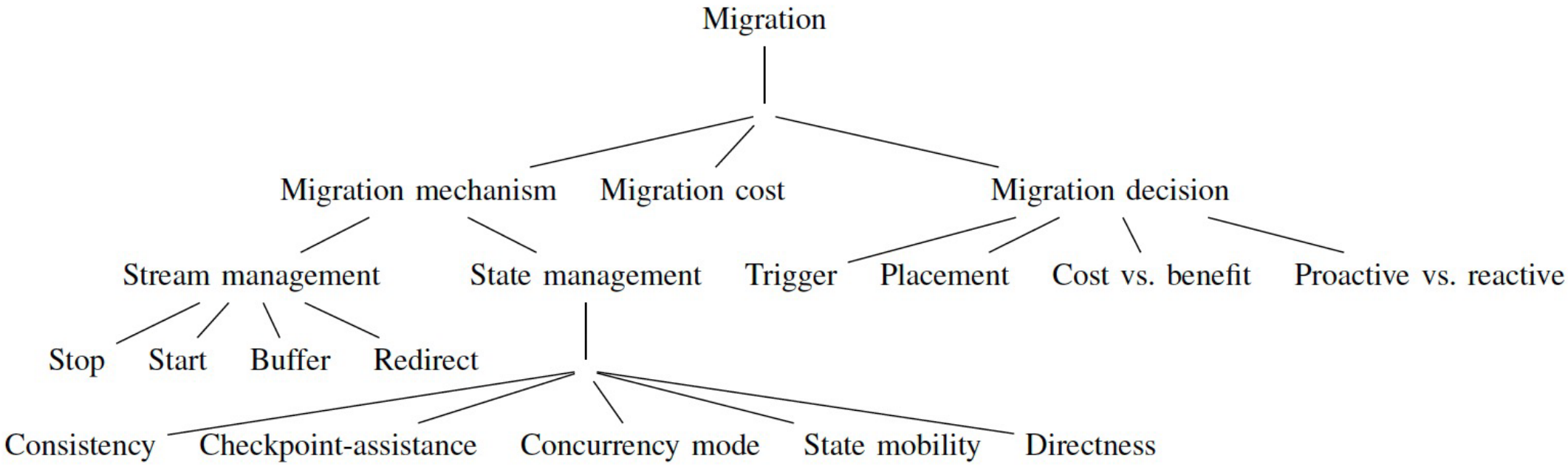
⇒ Load balancing/elasticity, fault tolerance, Quality of Service

Basic steps of all-at-once operator migration

- Stop streams from upstream nodes
- Buffer tuples at upstream nodes
- Extract operator state at old host
- Transfer operator state to new host
- Redirect streams
- Start streaming of tuples from upstream nodes



Concepts of migration



Espen Volnes, Thomas Plagemann, and Vera Goebel. 2023. To Migrate or not to Migrate: An Analysis of Operator Migration in Distributed Stream Processing. IEEE Communications Surveys & Tutorials, Volume: 26, Issue: 1, Firstquarter 2024

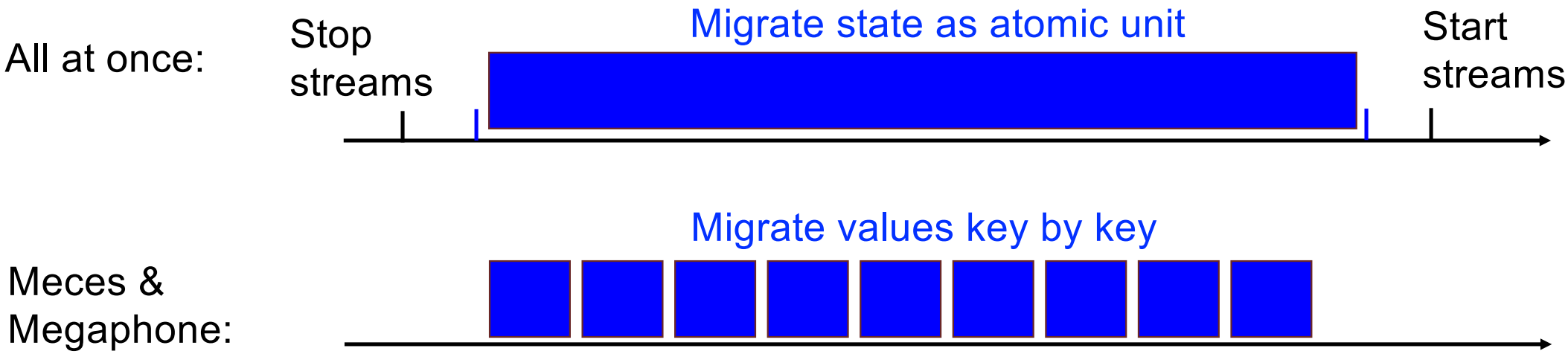
Operator state

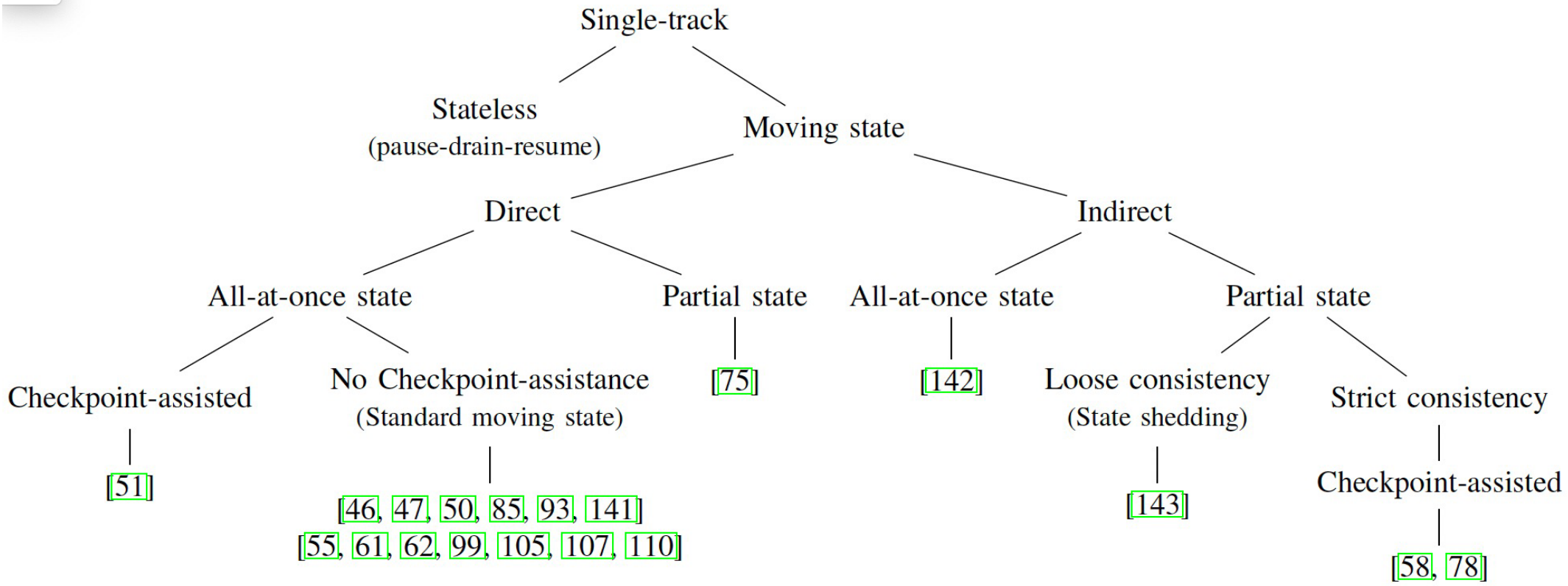
- Processing State: historical data necessary for processing incoming tuples
- Buffer State: holds output tuples that are not yet forwarded downstream
- Routing State: next operator for output tuples

- Examples of processing state:
  - Sum: all tuples that have arrived in a window or sum of all tuples that have arrived so far in the window or partial aggregate
  - Join: all tuples that have arrived in a window so far

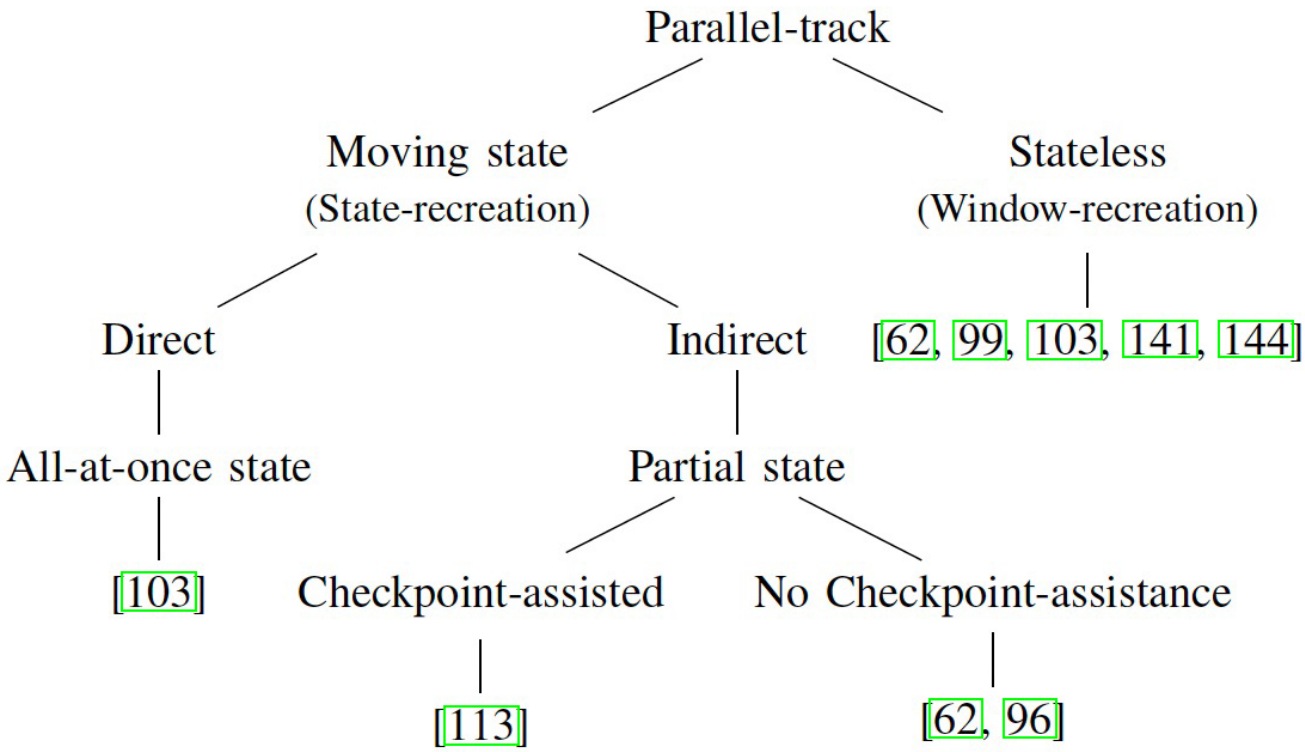
- In modern systems, like Apache Flink, state is managed in key-value stores

Timeline operator migration mechanism (simplified)





Espen Volnes, Thomas Plagemann, and Vera Goebel. 2023. To Migrate or not to Migrate: An Analysis of Operator Migration in Distributed Stream Processing. IEEE Communications Surveys & Tutorials, Volume: 26, Issue: 1, Firstquarter 2024



Espen Volnes, Thomas Plagemann, and Vera Goebel. 2023. To Migrate or not to Migrate: An Analysis of Operator Migration in Distributed Stream Processing. IEEE Communications Surveys & Tutorials, Volume: 26, Issue: 1, Firstquarter 2024

Trend in modern solutions & core ideas

- Recent solutions: state is not an atomic unit
  - Static key – value partitions
- Core ideas of Lazy Migration:
  - Fine granular static and dynamic state partitions
  - Semantic awareness: understand which part of the state is needed when
  - Utility awareness: what are the most important parts of the state

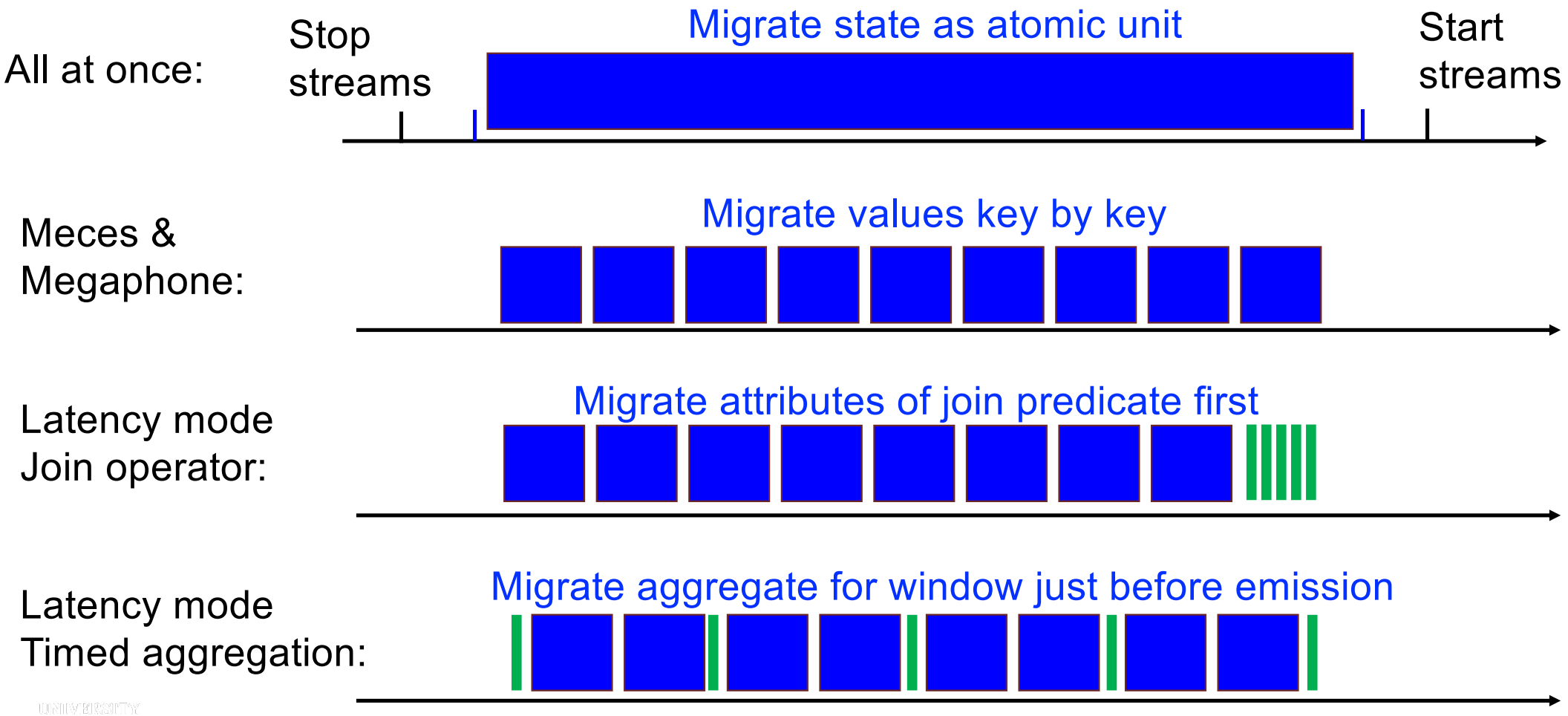
Perspective	State units	Consideration
Partial state	Tuple in join operator (op.)	Selectivity
	Sequence in pattern matching op.	Selectivity, Minimum number of events before match
	Tuple in aggregation op.	Effect on aggregated results, Emission times
	Partial window in aggregation op.	Emission time
Key-Value	Key Value	Number of Key-Value pairs, Size of each Key-Value pair
All-at-once	Entire state	State size

Lazy migration: core approach

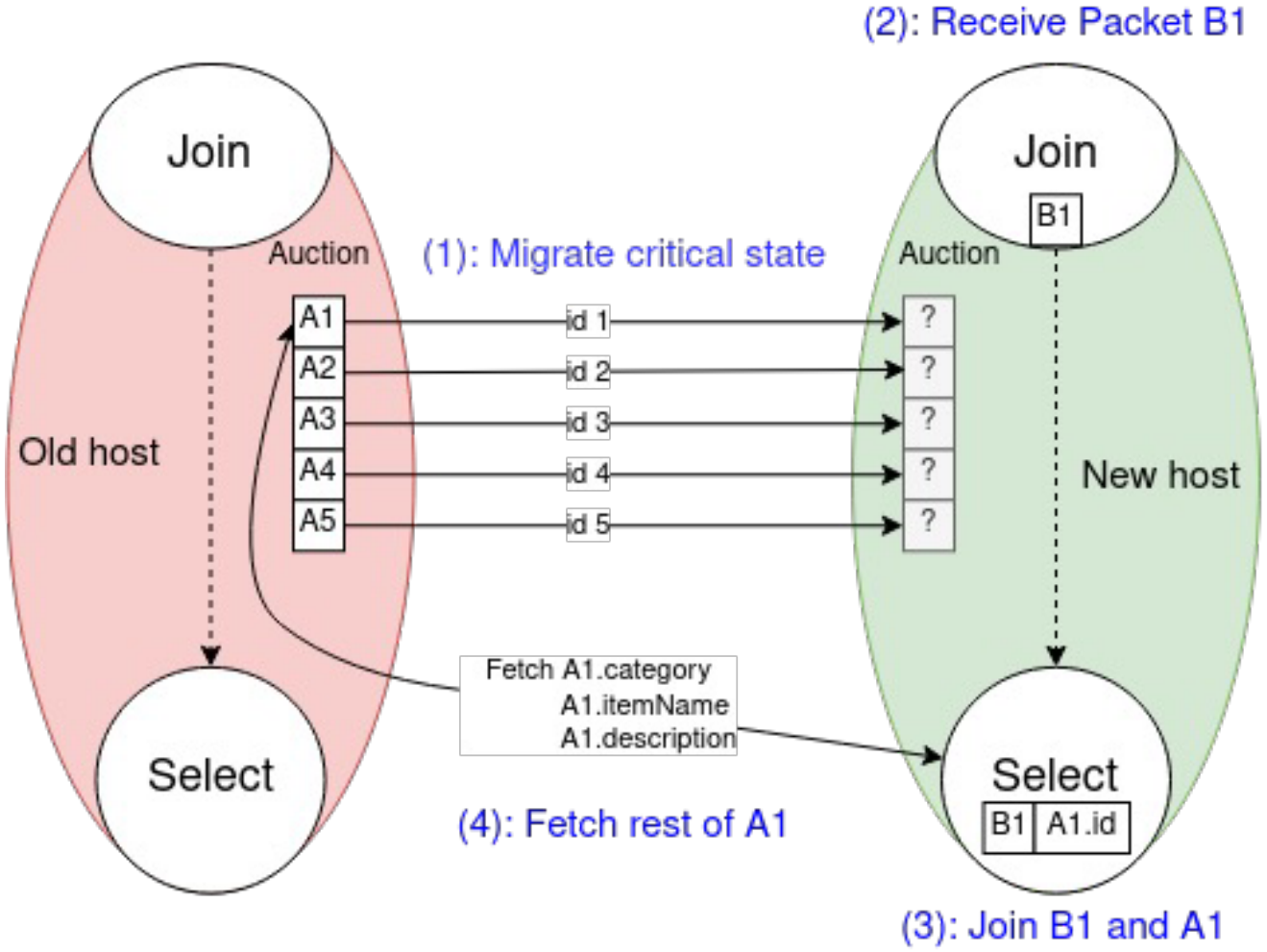
- Fine granular state partition, static & dynamic
- Transfer state partitions to the new host just before they are needed
- State partitions that are essential for processing have higher priority
- Latency mode:
  - Dynamic state partition
  - Timely scheduling
- Utility mode:
  - Essential and unessential attributes
  - User defined utility functions

Espen Volnes, Thomas Plagemann, Boris Koldehofe, and Vera Goebel. 2022. Travel light: state shedding for efficient operator migration. In Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems (DEBS '22)

Timeline operator migration mechanism (simplified)

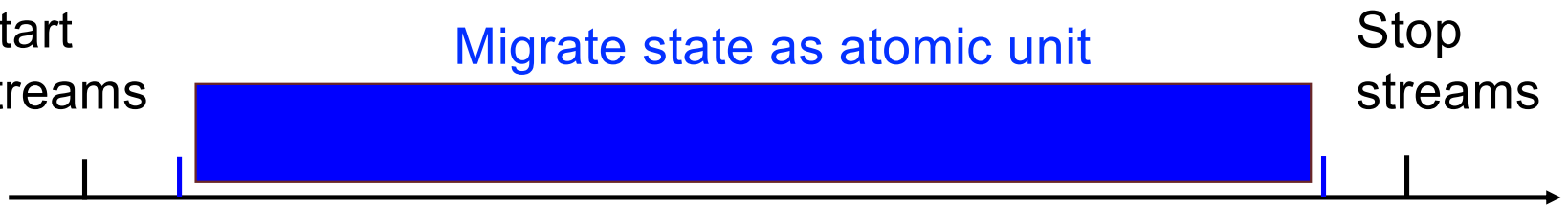


Utility mode – join operator

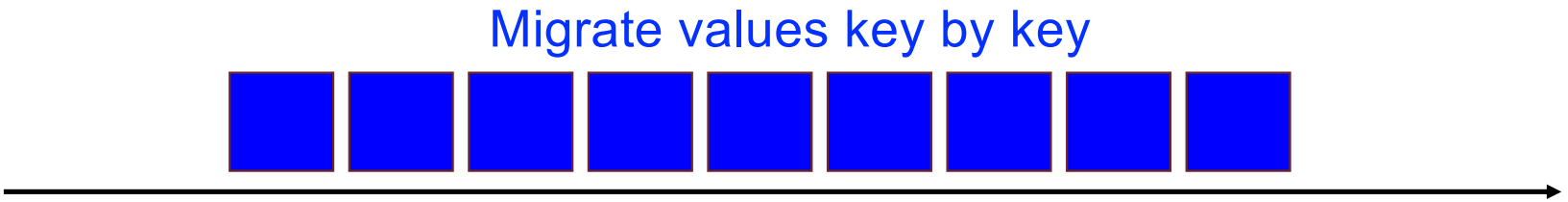


Timeline operator migration mechanism (simplified)

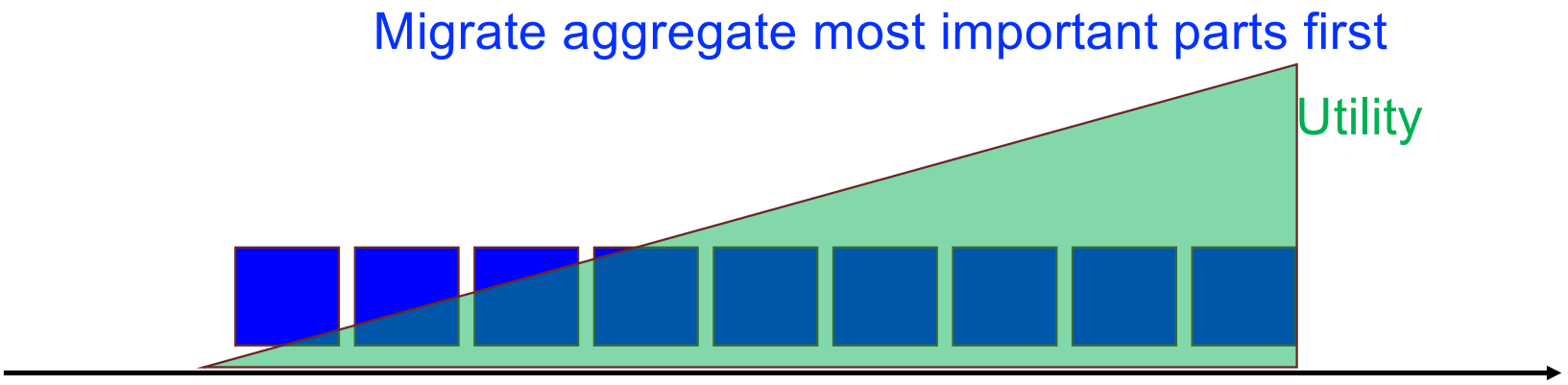
All at once:



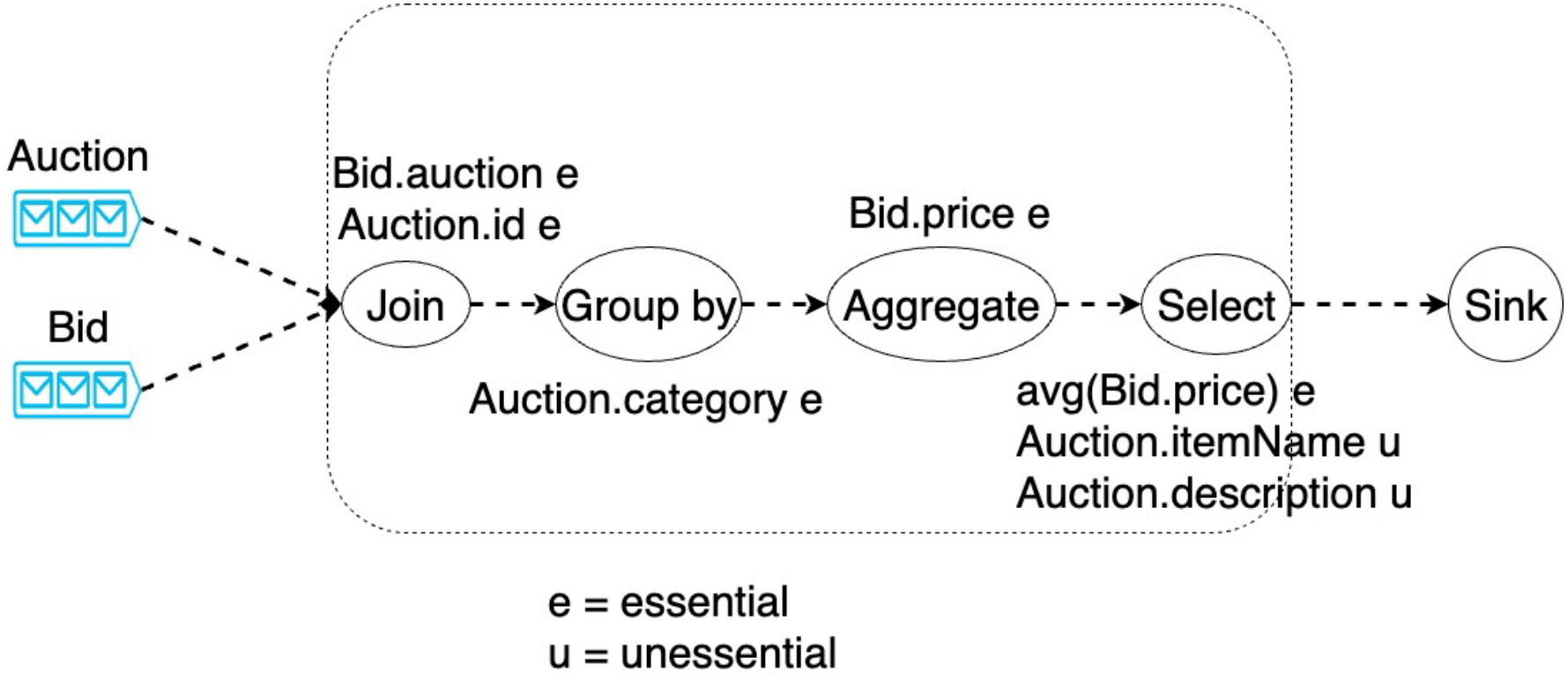
Meces &
Megaphone:



Lazy Migration
Utility mode:



Operator graph with essential and unessential attributes



How to evaluate?

- Compare performance with state-of-the art systems and baseline: all-at-once, Meces, Rhino, Megaphone
- Not all systems are available
- How to isolate the operator migration overhead from the rest of the system

⇒ Implement all operator migration mechanism in the same system

- Apache Flink???
  - Core idea of Lazy Migration is based on fine grained state partitions
  - Not (yet) supported in Flink

How to evaluate? (cont.)

- Fair comparison requires repeatable experiments
- All solutions must be evaluated under exactly the same conditions
- Hard to guarantee in real-world experiments

⇒ Implementation and simulation in DCEP-Sim

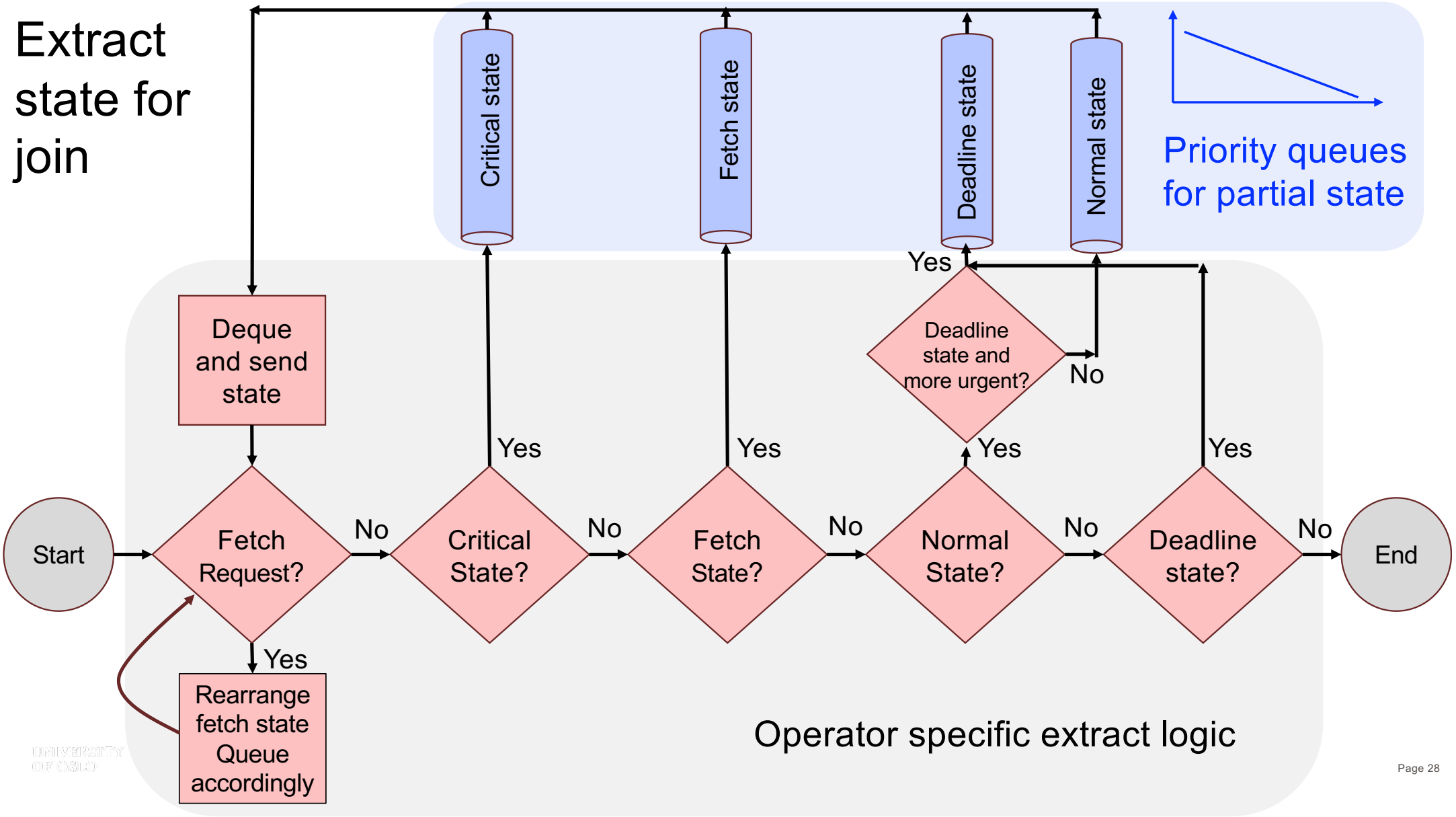
- DCEP-Sim is a distributed stream processing system in ns-3
  - Accurate models for communication behaviour exist
  - Models for processing delay
  - Migration mechanism framework

Migration mechanism framework

- Import state
- Extract state and place operator state partitions into priority queues
  - Critical state queue: if this state is not available the operator on the new host blocks
  - Fetch state queue: increase priority when it is necessary
  - Normal state queue: lowest priority
  - Deadline state queue: partitions that need to be delivered at a certain deadline
- Schedule partitions according to priority queues and deadlines

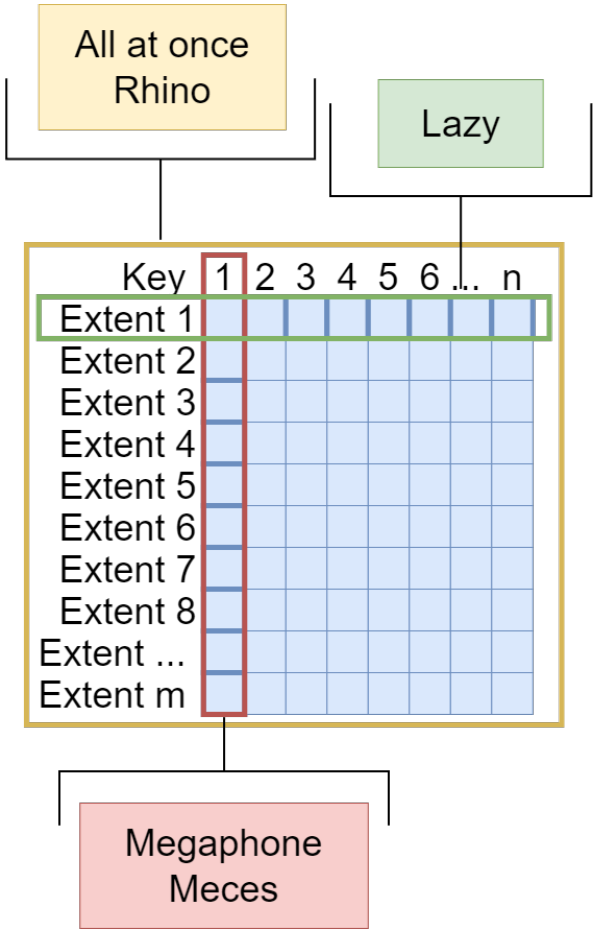
- The framework facilitated the implementation of all migration mechanisms

Extract state for join

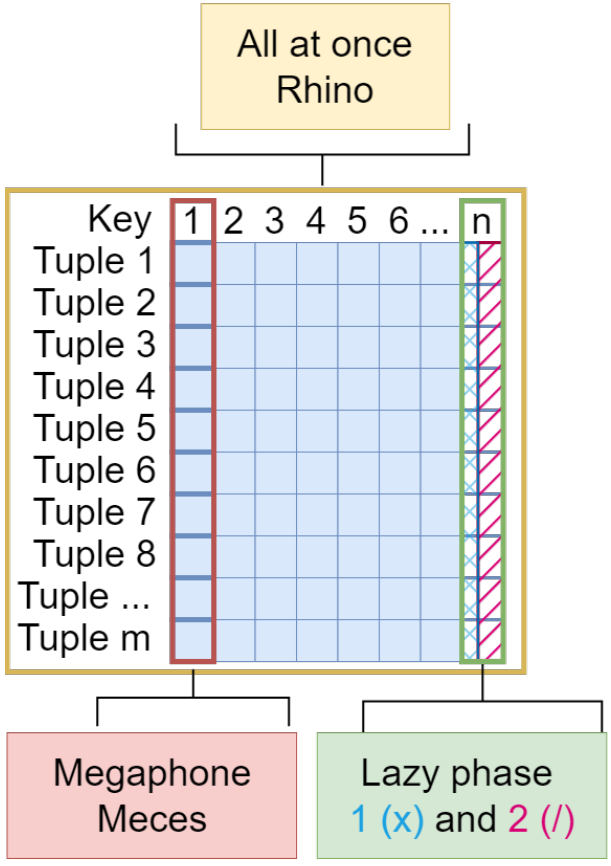


State movement priority for stateful operators

Timed aggregation



Join



Workload: NextMark benchmark

Join:

```
select A.itemName, B.price
from Auction A
join Bid B
on A.id > B.auction - 10
and A.id < B.auction + 10
```

Timed aggregartion:

```
select avg(B.price)
from Bid B
group by B.auction, hop(100 seconds, 1 second)
```

Join followed by timed aggregation:

```
select A.itemName, avg(B.price)
from Auction A
join Bid B
on A.id > B.auction - 10
and A.id < B.auction + 10
group by B.auction, hop(100 seconds, 1 second)
```

10.000 Auction tuples
1000.000 Bid tuples
Network bandwidth: 10 MB/s

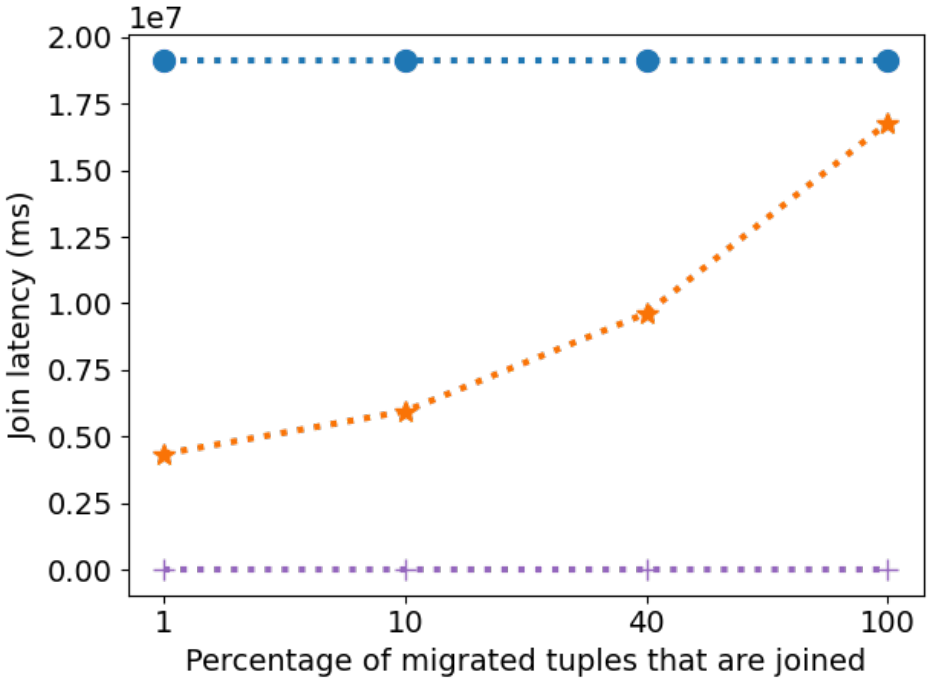
Metrics

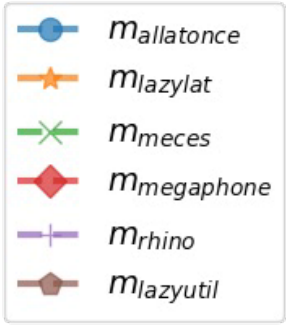
- Join latency: comparing join time with and without migration
- Aggregation latency: comparing aggregation time with and without migration
- Emission delay: how long after emission deadline a tuple is emitted
- Missed window extents: tuples that could not be processed in the window they should
- Utility: number of tuples that are joined

- Parameters:
  - Join selectivity: 1%, 10%, 40%, 100%
  - Window jump: 200 ms to 10.000 ms in ten steps
  - Operational migration period (OMP): 1ms, 1000 ms, 2000ms, …, 15.000ms

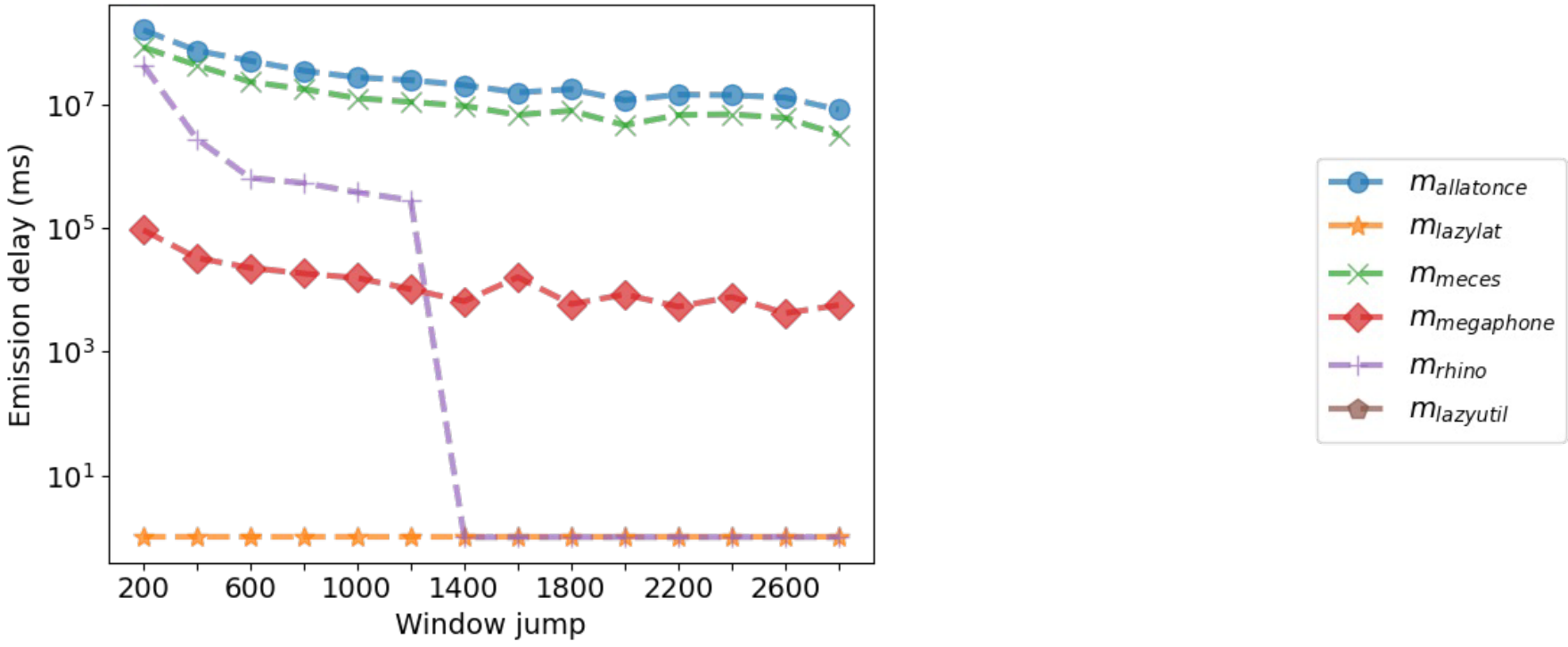
Experiment 1: Join latency for non-equijoin

migrated tuples that are joined: 1%, 10%, 40%, 100%

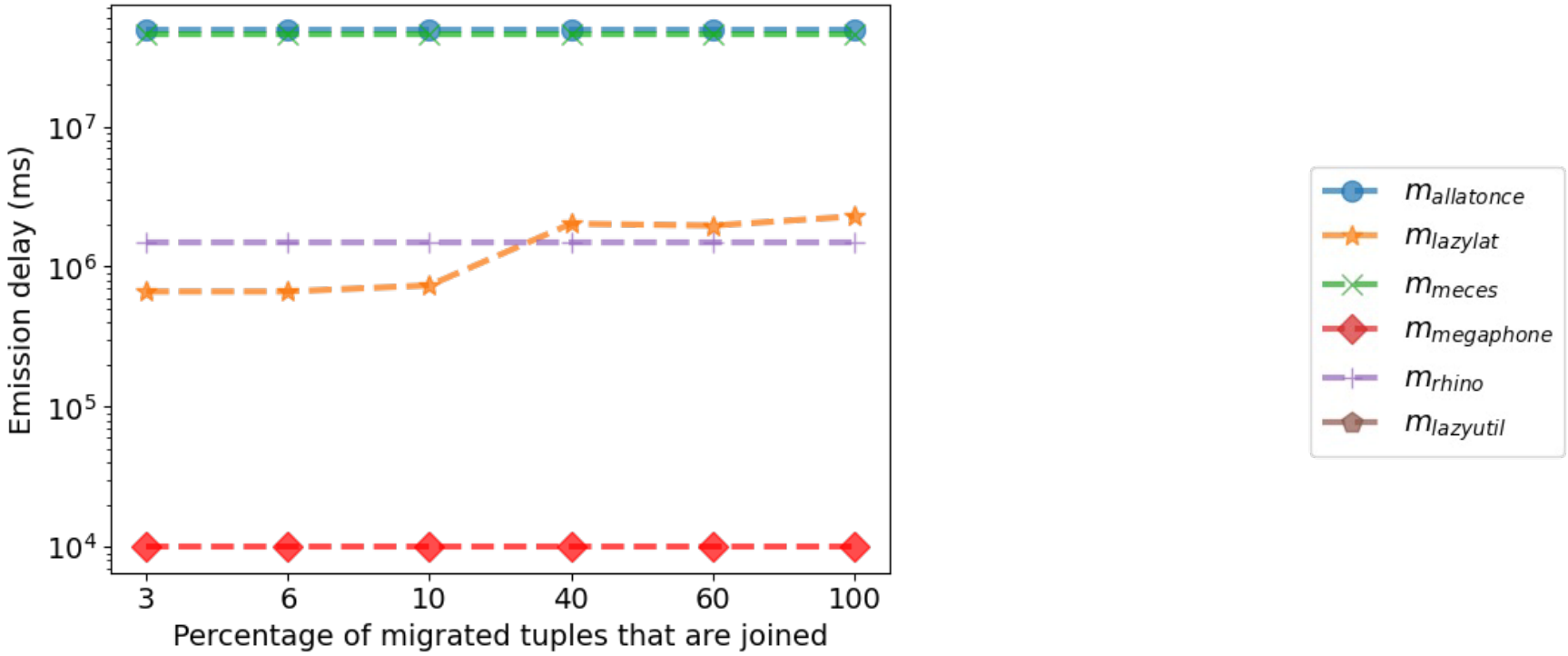




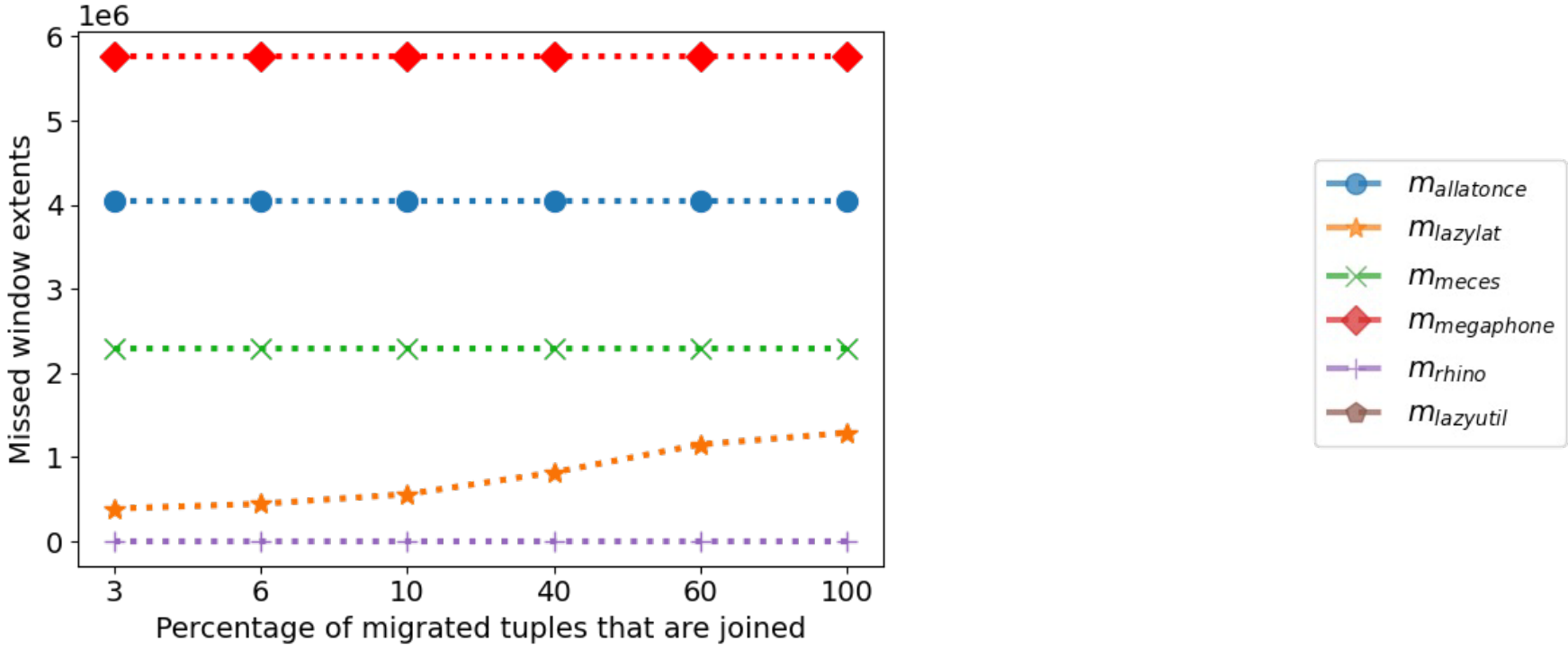
Experiment 2: Emission delay for timed aggregation



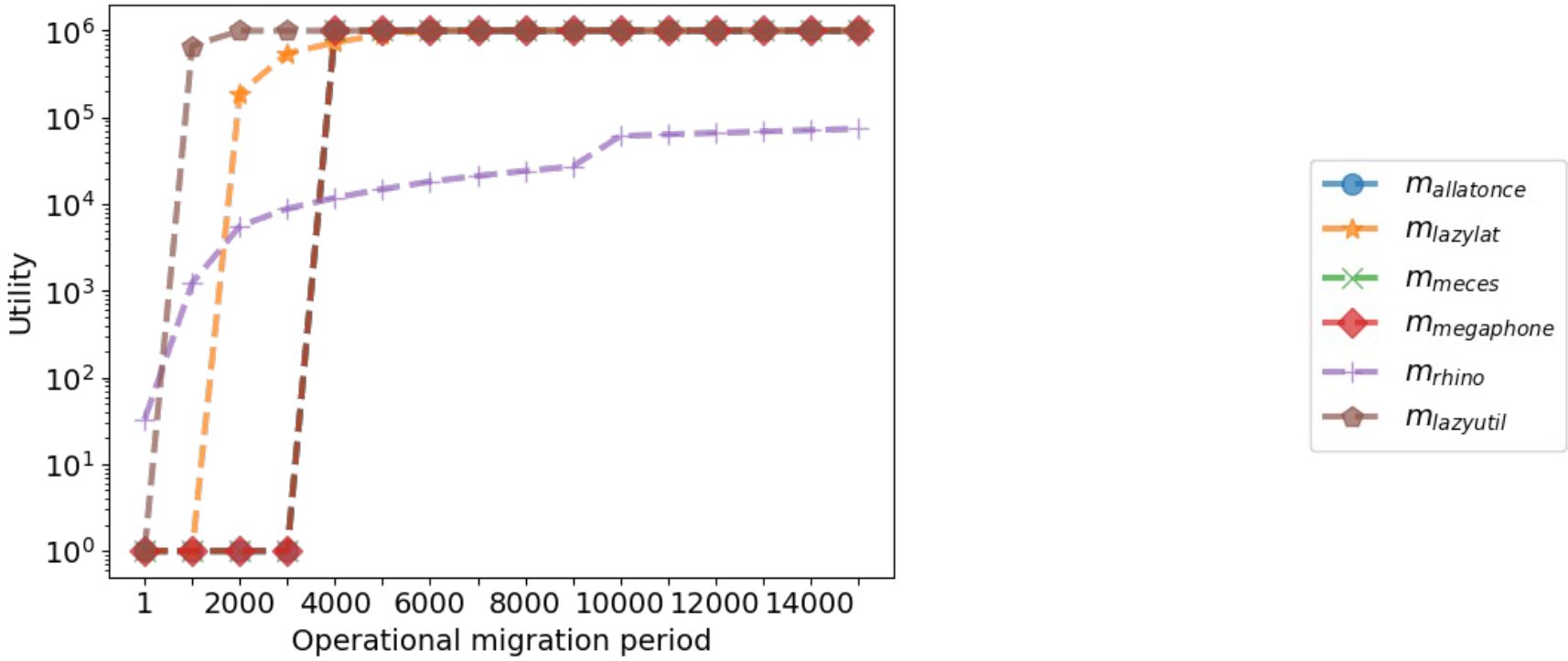
Experiment 3: Emission delay for join → timed aggregation



Experiment 3: Missed window extents for join → timed aggregation



Experiment 5: Utility for non-equijoin and different OMPs



Conclusions

- Main contributions:
  - Latency mode:
    - Superior with non-equijoin and sliding window timed aggregation
    - For word count and equijoin Meces and Megaphone are better optimized
  - Migration that is interrupted:
    - Utility mode superior
    - Latency mode works also well
  - Migration Framework

- How useful is an evaluation based on DCEP-Sim

Questions?

UNI
OF