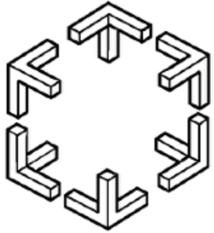


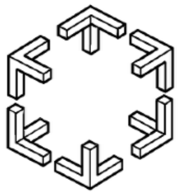
Viper: Communication-Layer Determinism and Scaling in Low-Latency Stream Processing

Ivan Walulya, Yiannis Nikolakopoulos, Vincenzo Gulisano

Marina Papatriantafilou and Philippas Tsigas

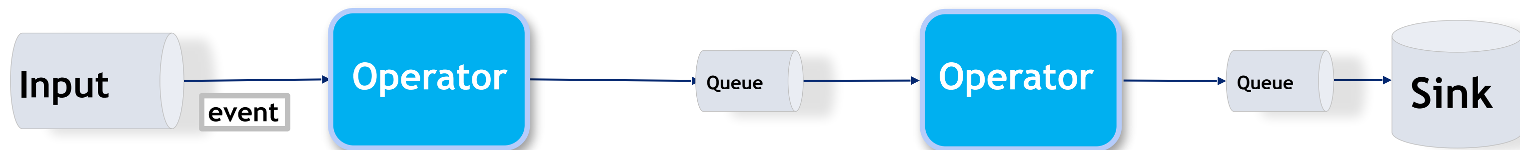
Auto-DaSP 2017

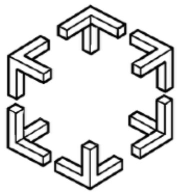
Chalmers University
of Technology
Göteborg, Sweden



Stream Processing Applications

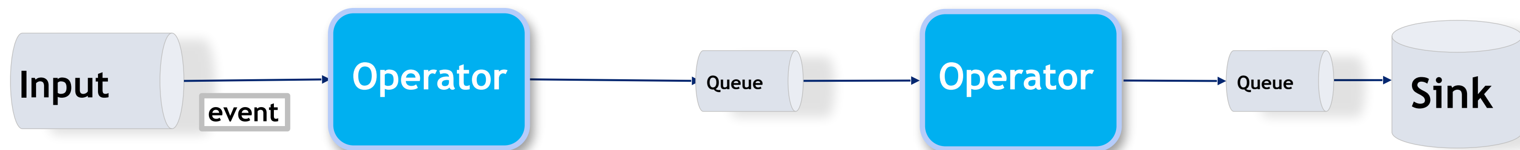
- Process infinite streams of data
  - High throughput
  - Low latency
- High resource requirements (multiple cores/nodes)

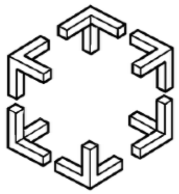




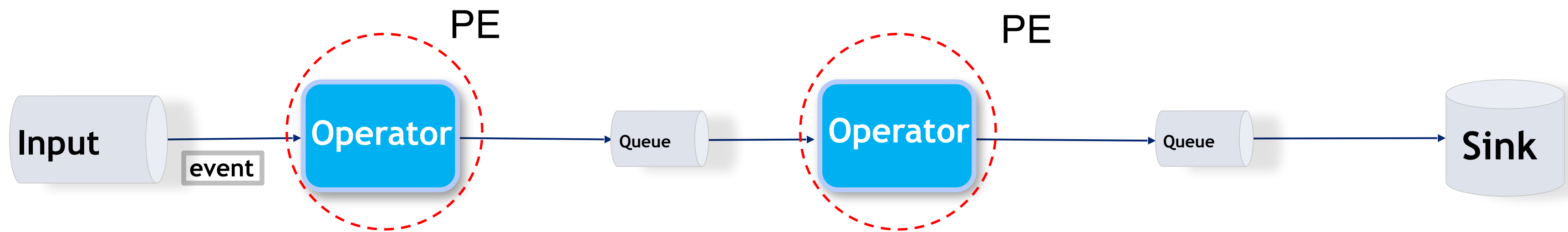
Stream Processing Applications

- Process infinite streams of data
  - High throughput
  - Low latency
- High resource requirements (multiple cores/nodes)
- Abstraction: Data-flow graphs of operators and streams
  - Expose pipeline and task parallelism

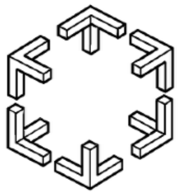




Introduction

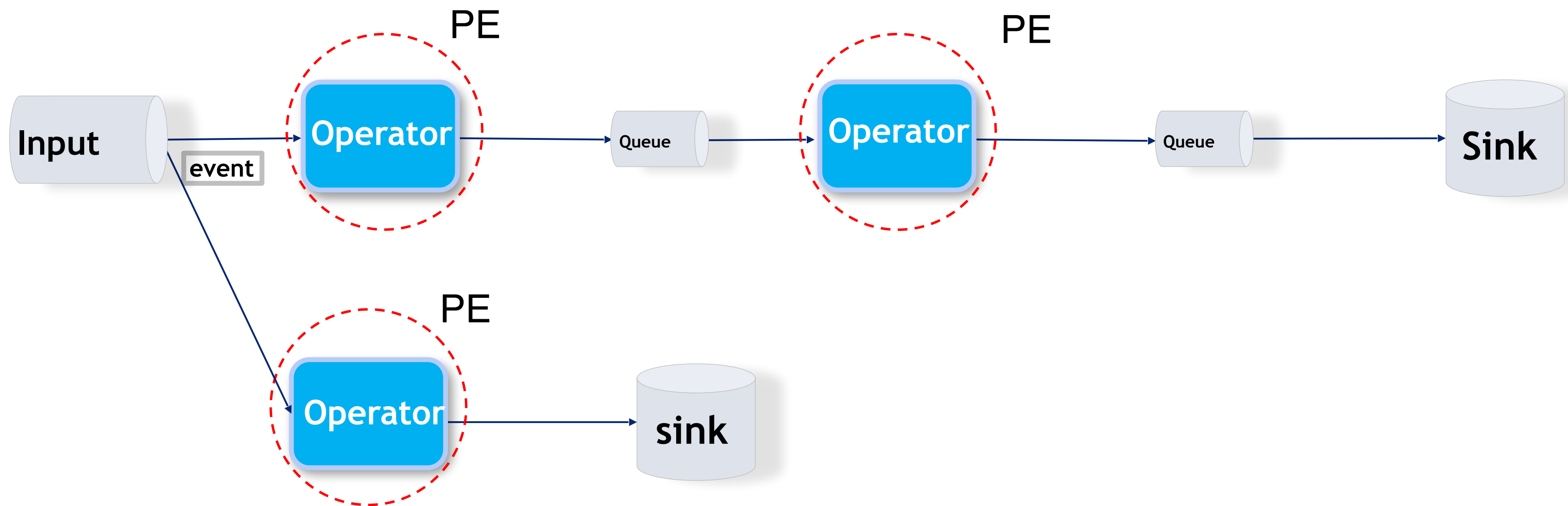


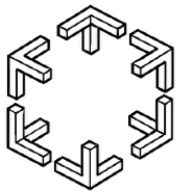
PE – Processing Element



Introduction

- Parallel processing:
  - Pipeline parallelism
  - Task parallelism



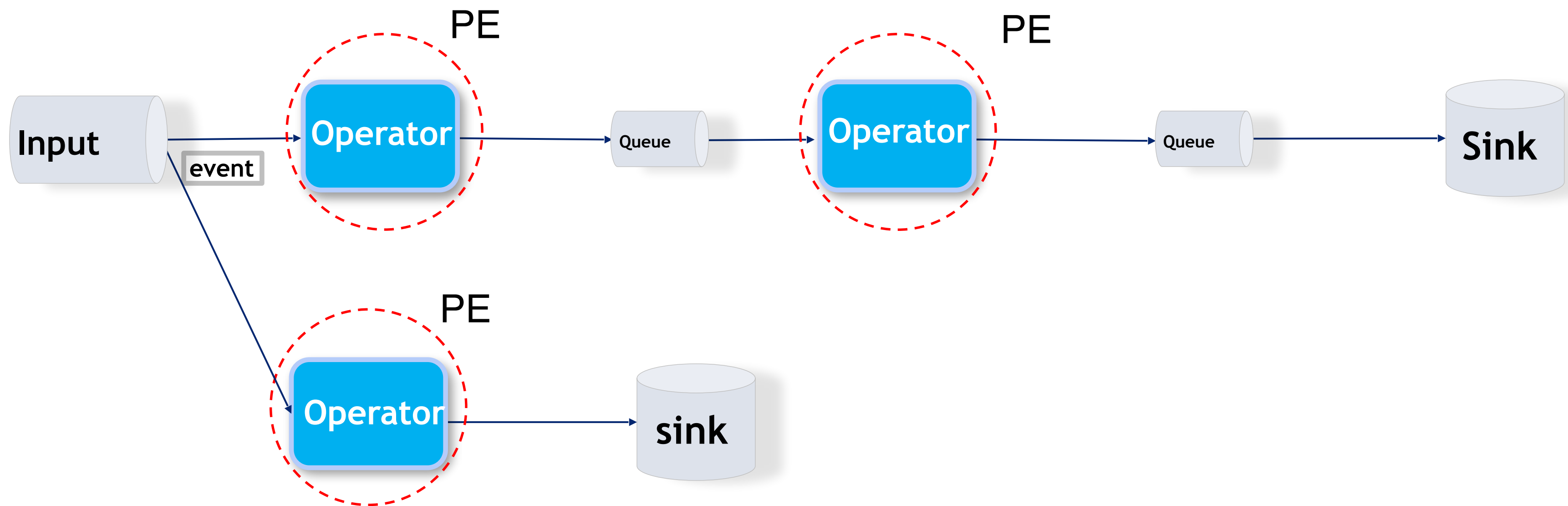


Introduction

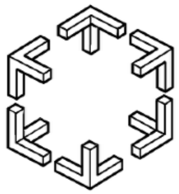
- Parallel processing:
  - Pipeline parallelism
  - Task parallelism

✓ **Compilers**
✓ **Run-time Schedulers**

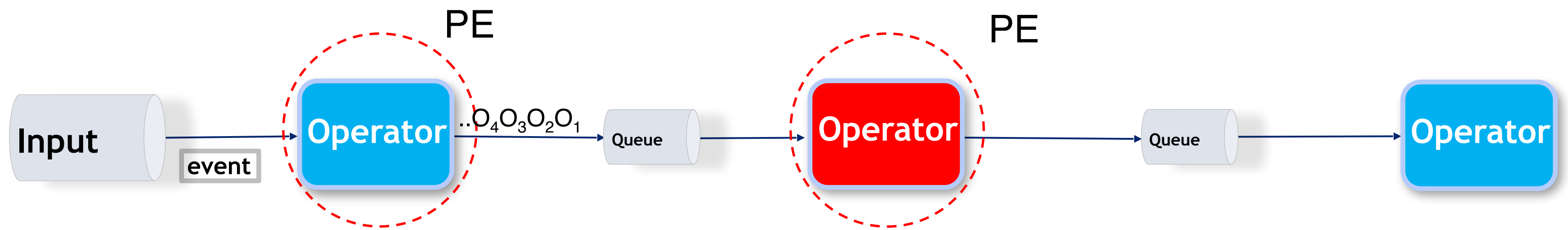
❖ **Limited by data graph**

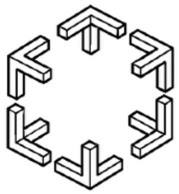


PE – Processing Element



Introduction





Introduction

- Data or operator parallelism:
  - Bottlenecks at operator level
  - Split the data and replicate the operator

PE

PE

Input

event

Operator

O_4O_1

Queue

Operator

Queue

Operator

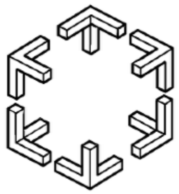
O_3O_2

Queue

Operator

Queue

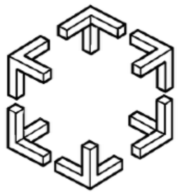
PE



Introduction

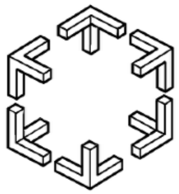
- Data or operator parallelism:
  - Bottlenecks at operator level
  - Split the data and replicate the operator

> ✓ Not limited by data-flow graph
>
> ❖ Trade latency for high throughput
>
> ❖ Preserve sequential semantics



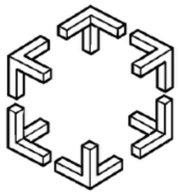
Motivation

- Determinism: preserve sequential semantics (safety)



Motivation

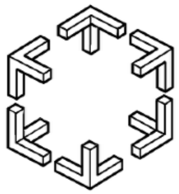
- Determinism: preserve sequential semantics (safety)
  - Merge operators
    - Enforce ordering amongst the output tuples.



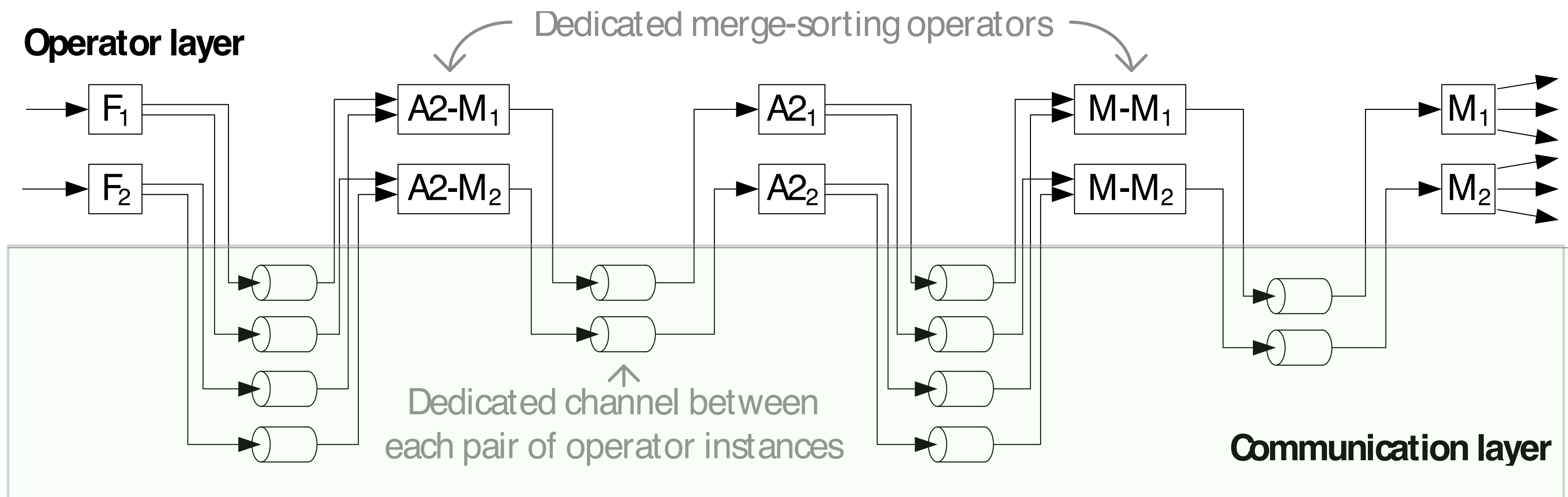
Motivation

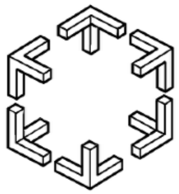
- Determinism: preserve sequential semantics (safety)
  - Merge operators
    - Enforce ordering amongst the output tuples.
    - Compiler generated [Schneider 2013].
    - Left to Application or Library developer [Apache Storm, Flink].

PE
PE
Input
event
Operator
O_4O_1
Queue
Operator
Merger
Operator
O_3O_2
Queue
Operator
PE



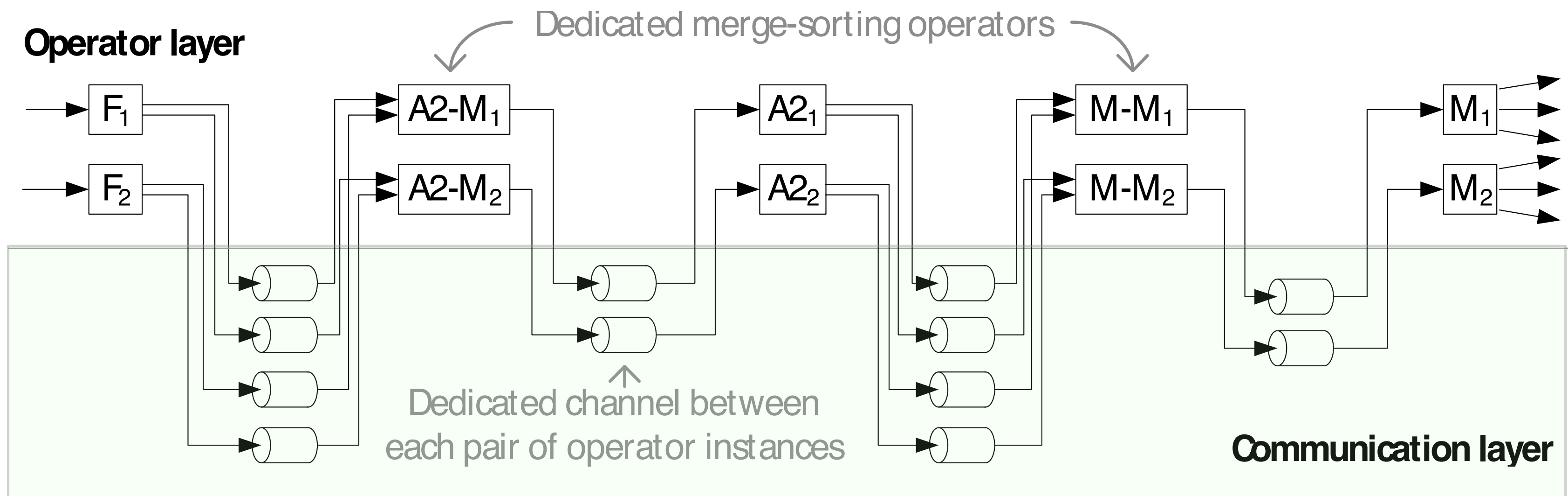
Problem Statement

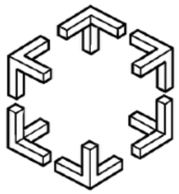




Problem Statement

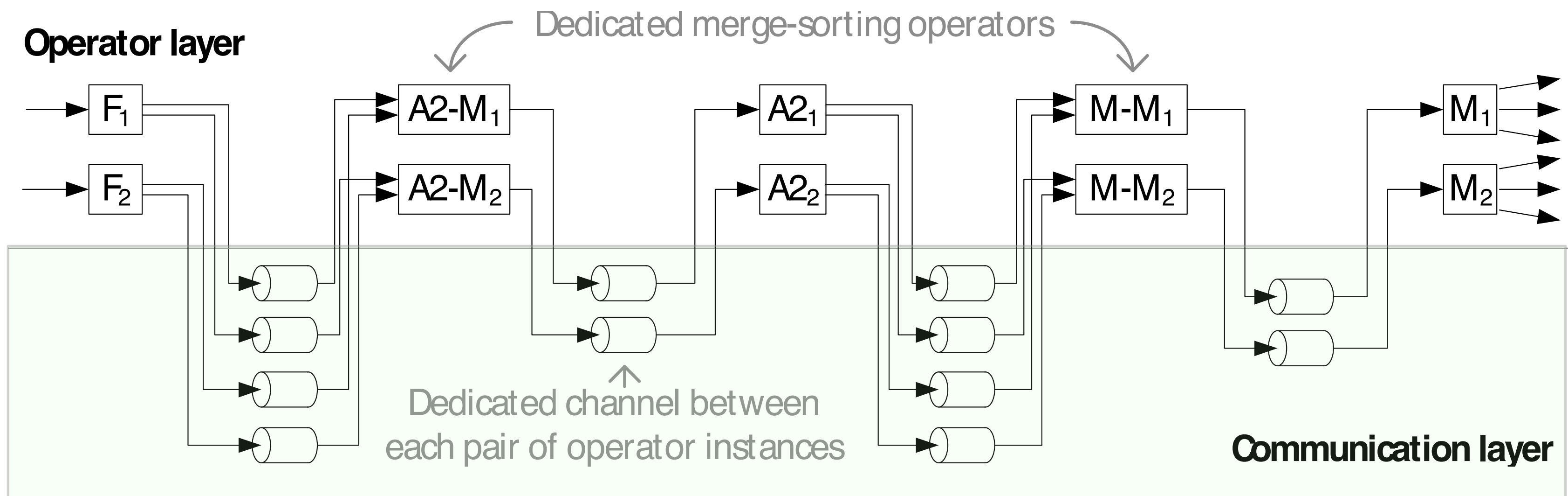
- Overheads of Merge operators:
  - Introduce computation overhead
  - Higher latency due to increase in operators
  - Become processing bottleneck
  - Considerable burden on the developers
- **Challenge**: How to apply data-parallelism transparently and safely

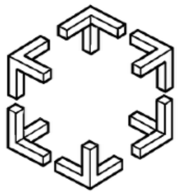




Our Solution

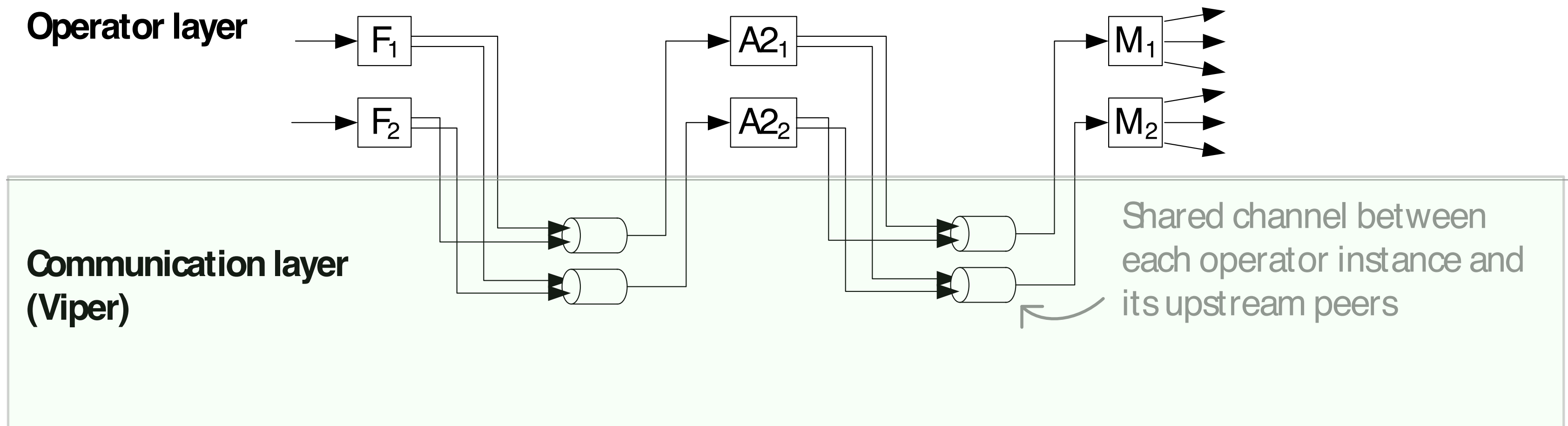
- Communication-layer determinism:
  - Leverage links to achieve both communication and determinism.
    - Shared state and synchronization

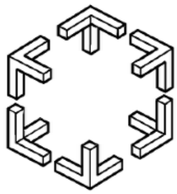




Our Solution

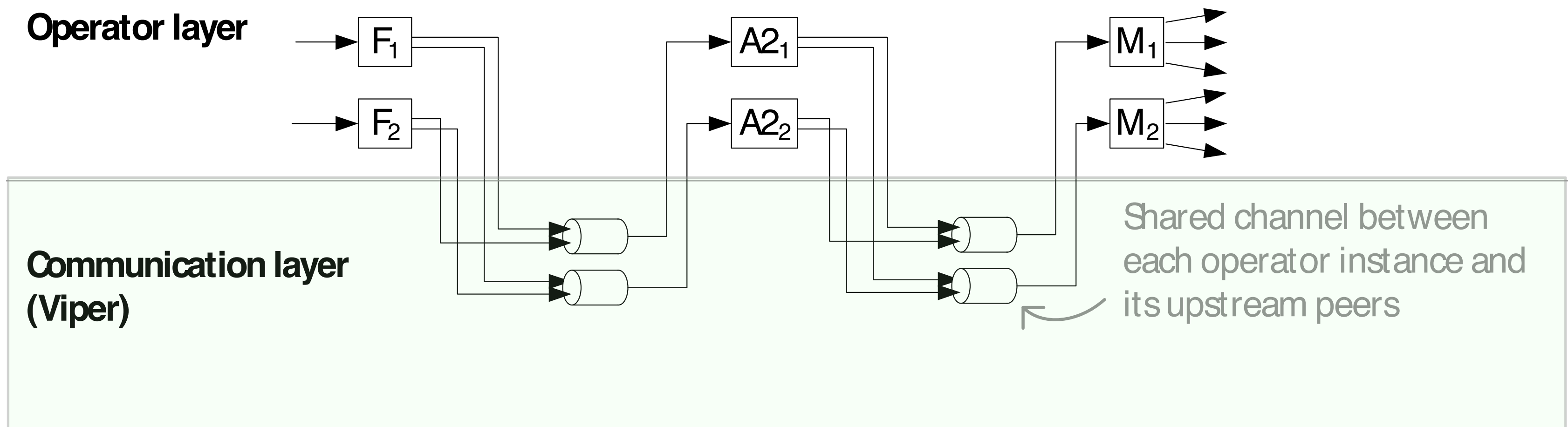
- Communication-layer determinism:
  - Leverage links to achieve both communication and determinism.
    - Shared state and synchronization

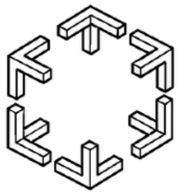




Our Solution

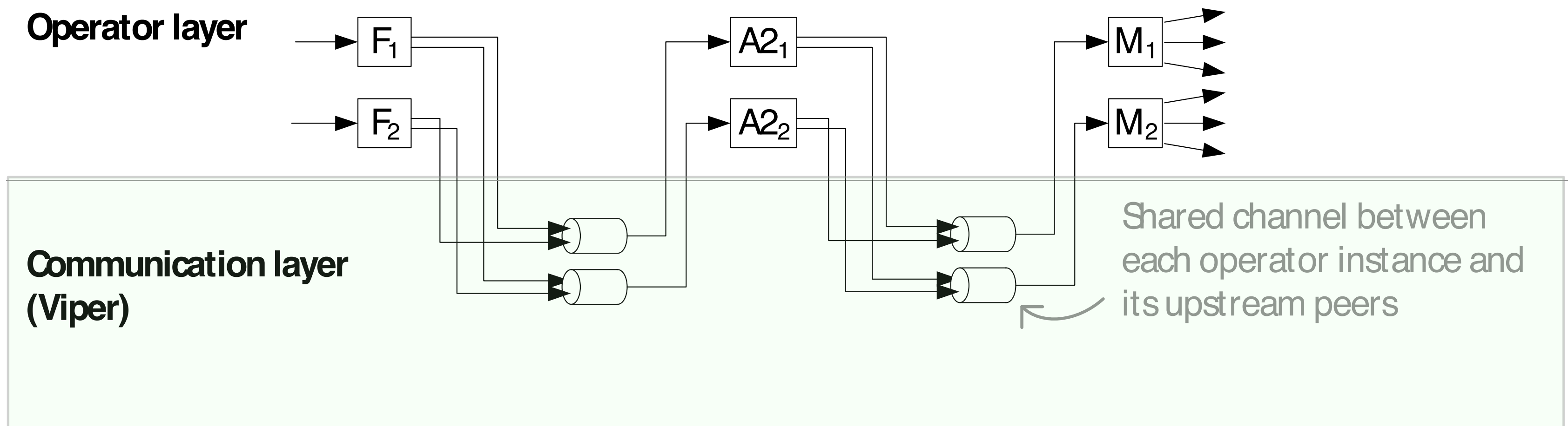
- Communication-layer determinism:
  - Leverage links to achieve both communication and determinism.
    - Shared state and synchronization
  - Extends the ScaleGate [Gulisano et. al. 2016].
  - Modularly take the logic of deterministic away from developer (***Viper***).

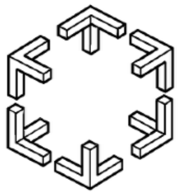




Our Solution

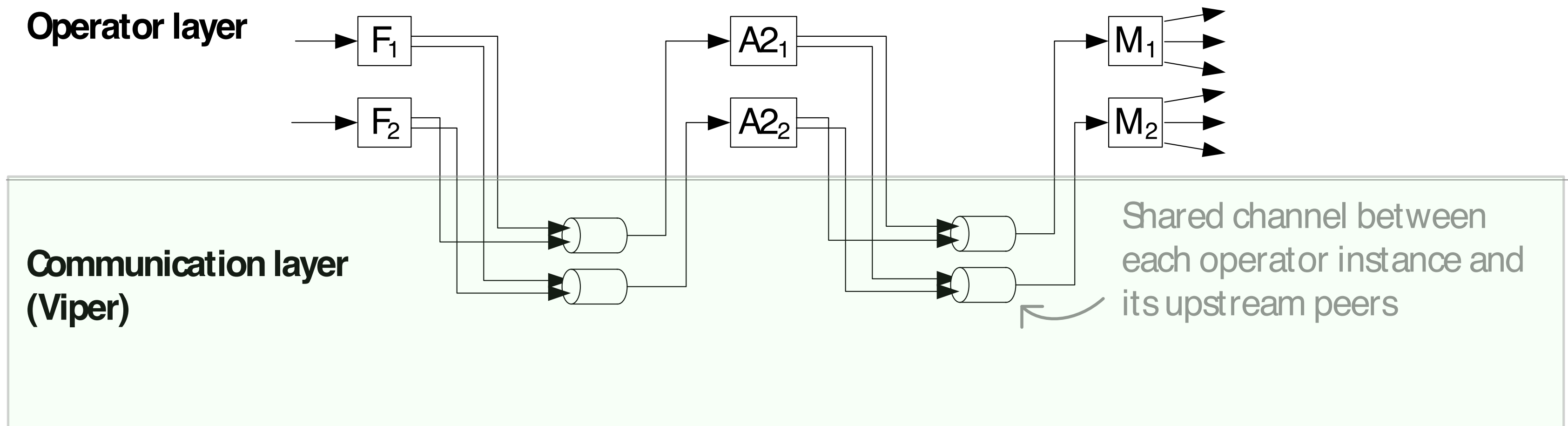
- Communication-layer determinism:
  - Leverage links to achieve both communication and determinism.
    - Shared state and synchronization
  - Extends the ScaleGate [Gulisano et. al. 2016].
  - Modularly take the logic of deterministic away from developer (***Viper***).
  - Evaluate our implementation on Apache Storm.

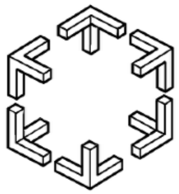




Viper Module

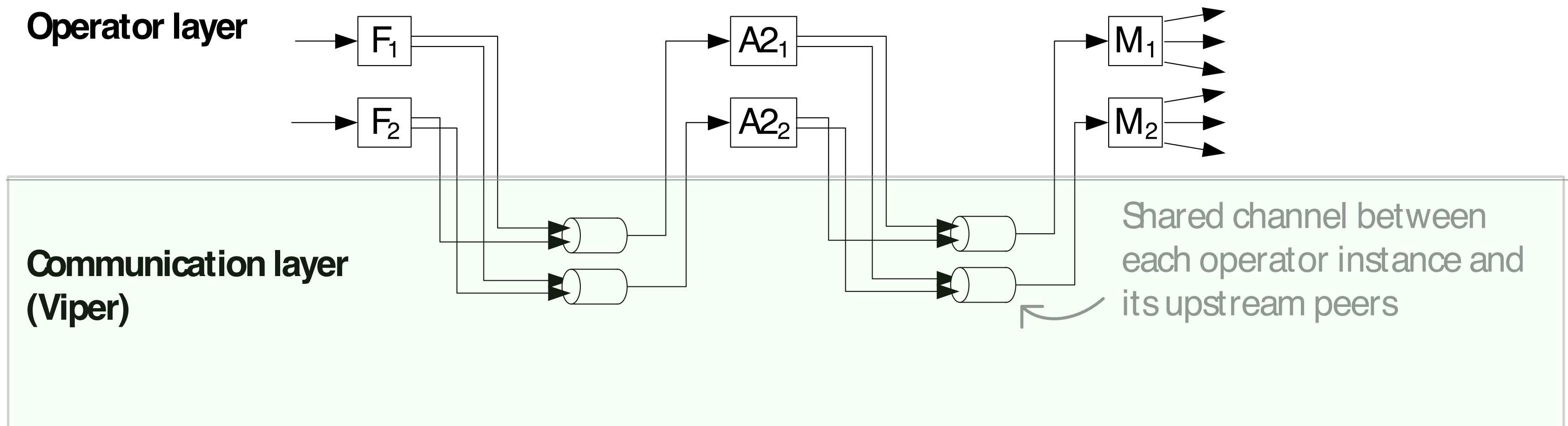
- Overall Approach
  - Replace *merge* operators and *channels* with a Viper module

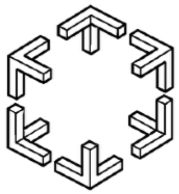




Viper Module

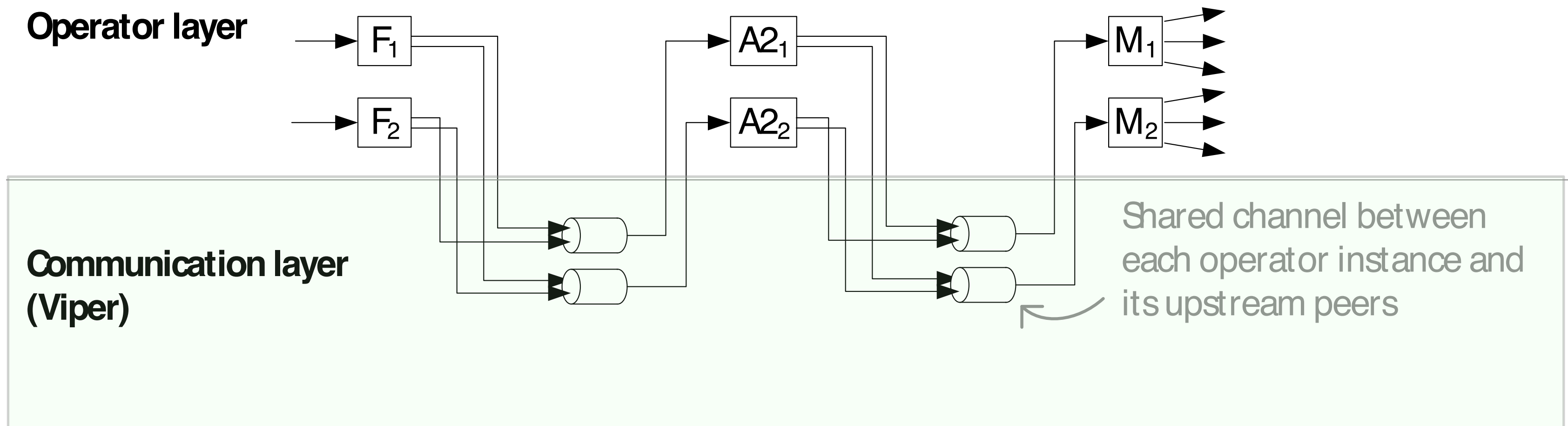
- Overall Approach
  - Replace *merge* operators and *channels* with a Viper module
  - Channel maintained for any set of source operator instances $S_1,..S_n$

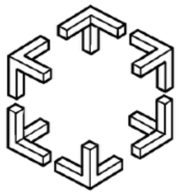




Viper Module

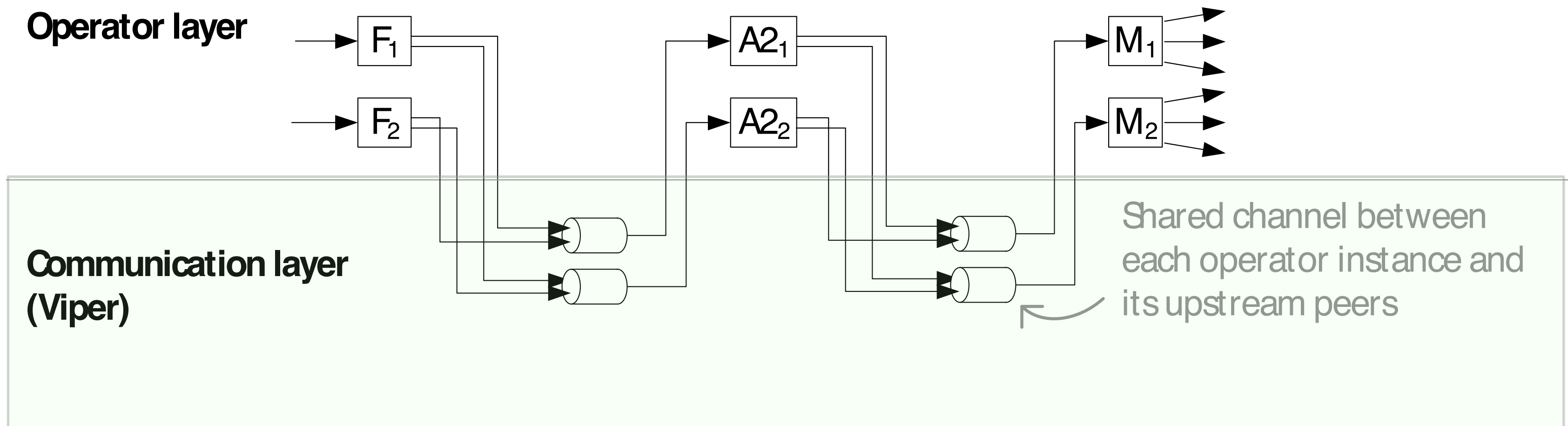
- Overall Approach
  - Replace *merge* operators and *channels* with a Viper module
  - Channel maintained for any set of source operator instances $S_1,..S_n$
  - Channel is either a thread-safe concurrent queue or a ScaleGate object

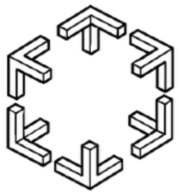




Viper Module

- Overall Approach
  - Replace *merge* operators and *channels* with a Viper module
  - Channel maintained for any set of source operator instances $S_1,..S_n$
  - Channel is either a thread-safe concurrent queue or a ScaleGate object
  - Sorting overhead shared by threads assigned to the same instance

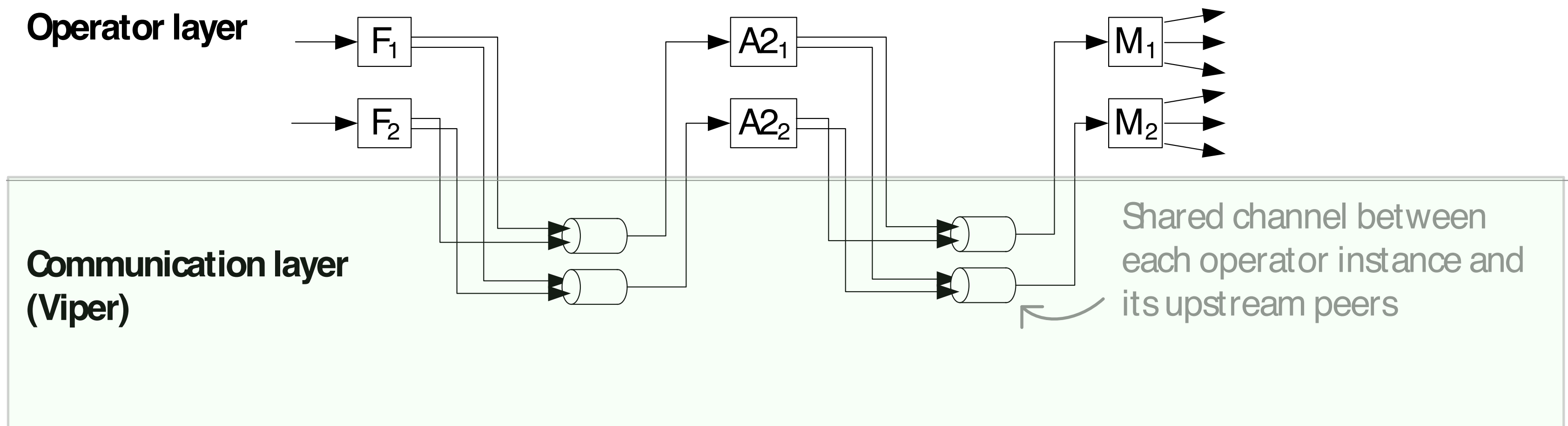


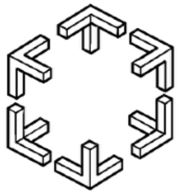


Viper Module

Overall Approach

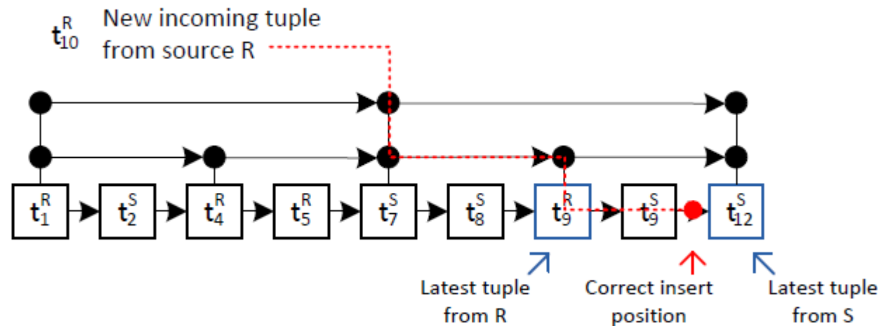
- Replace *merge* operators and *channels* with a Viper module
- Channel maintained for any set of source operator instances $S_1,..S_n$
- Channel is either a thread-safe concurrent queue or a ScaleGate object
- Sorting overhead shared by threads assigned to the same instance
- Watermarking mechanism to handle back-pressure





ScaleGate Data Structure

[Gulisano et. al. 2016].

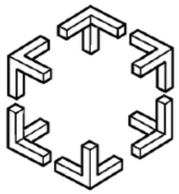


- API:

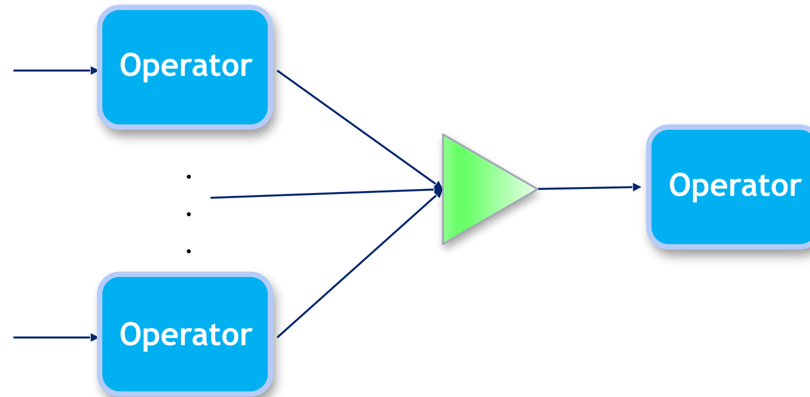
> `addTuple(tuple,sourceID)`
> allows a tuple from `sourceID` to be merged by ScaleGate in the resulting sorted stream of ready tuples.

> `getNextReadyTuple(readerID)`
> provides to `readerID` the next ***ready*** tuple that has not been yet consumed by the former.

https://github.com/dcs-chalmers/ScaleGate_Java

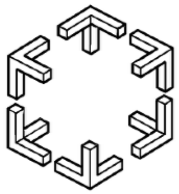


Viper API

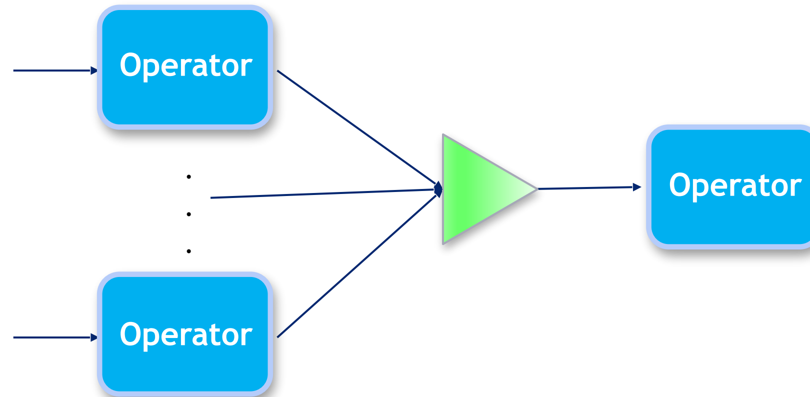



```
register(channel, sources, readers)
```

Register a new channel, specifying which sources will add tuples and which readers will get timestamp-sorted tuples from the channel.



Viper API

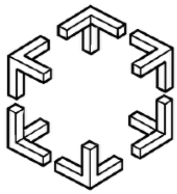



```
register(channel, sources, readers)
```

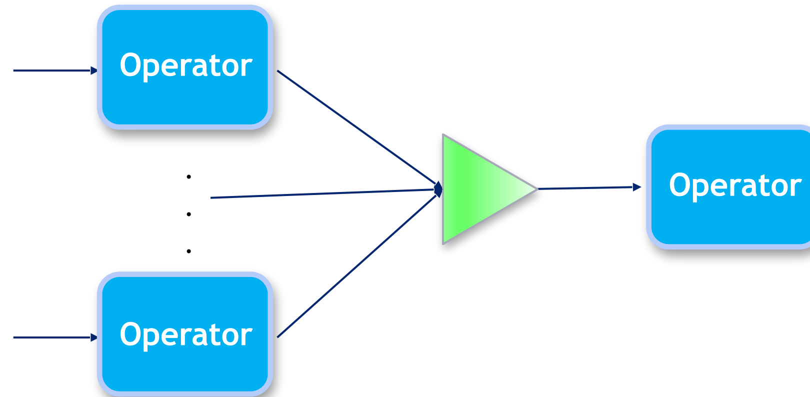
Register a new channel, specifying which sources will add tuples and which readers will get timestamp-sorted tuples from the channel.

```
addTuple(channel, sourceID, tuple)
```

Add tuple from a given source sourceID to the specified channel



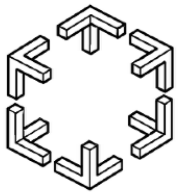
Viper API



`register(channel, sources, readers)`
Register a new channel, specifying which sources will add tuples and which readers will get timestamp-sorted tuples from the channel.

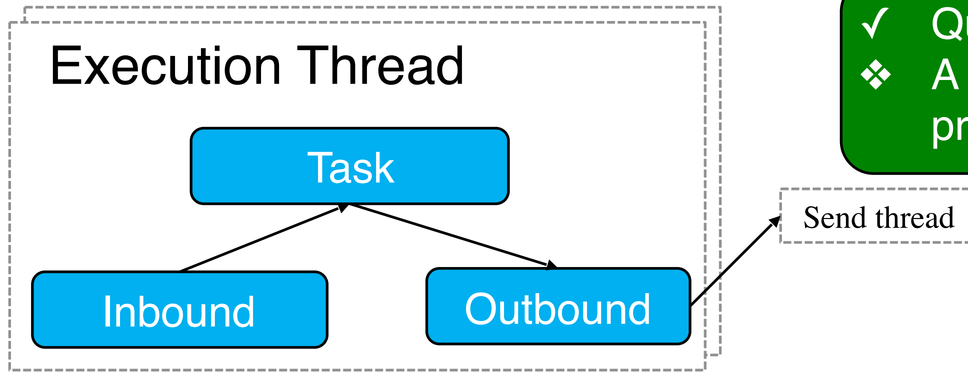
`addTuple(channel, sourceID, tuple)`
Add tuple from a given source sourceID to the specified channel

`getReadyTuple(channel, readerID)`
Retrieve the next ready tuple (if any) for the specified readerID from the channel

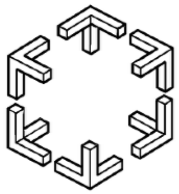


Evaluation

- Integrated Viper module in Apache Storm

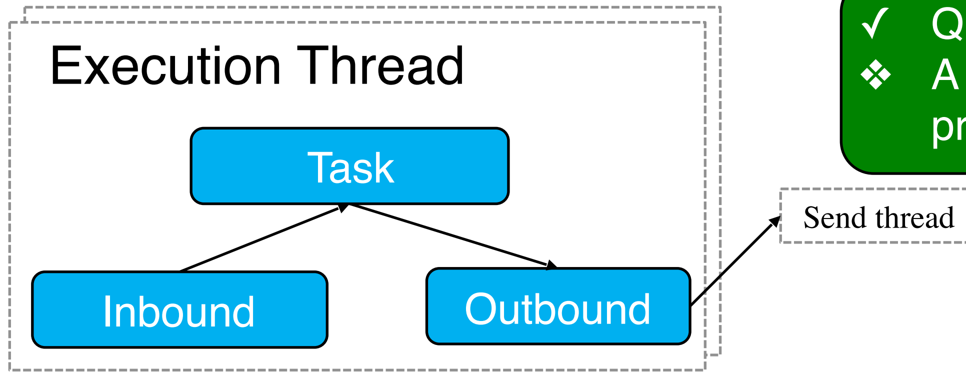


✓ Queues based on LMAX disruptor
❖ A Dedicated send thread per Executor process

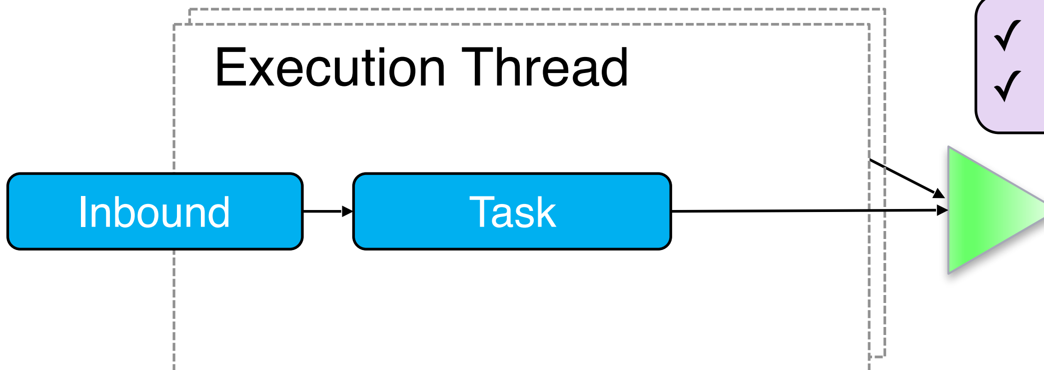


Evaluation

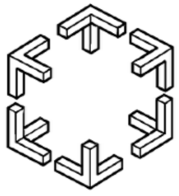
- Integrated Viper module in Apache Storm



- ✓ Queues based on LMAX disruptor
- ❖ A Dedicated send thread per Executor process

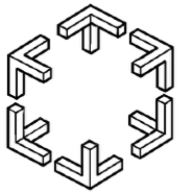


- ✓ Remove send threads
- ✓ Concurrent queues or ScaleGate



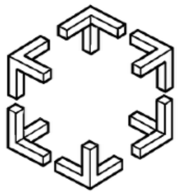
Evaluation Setup

- Linear-Road Dataset
  - Simulate vehicular traffic on a network of dynamic-toll roads
  - Variable toll dependent on congestion and accident proximity
  - Position reports and historical query requests
  - `Tuple<Type = 0, Time, VID, Spd, XWay, Lane, Dir, Seg, Pos>`



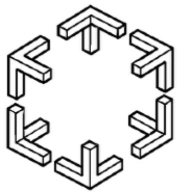
Evaluation Setup

- Linear-Road Dataset
  - Simulate vehicular traffic on a network of dynamic-toll roads
  - Variable toll dependent on congestion and accident proximity
  - Position reports and historical query requests
  - `Tuple<Type = 0, Time, VID, Spd, XWay, Lane, Dir, Seg, Pos>`
- Both `stateful` and `stateless` operators



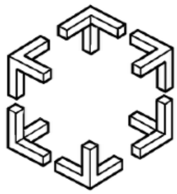
Evaluation Setup

- Linear-Road Dataset
  - Simulate vehicular traffic on a network of dynamic-toll roads
  - Variable toll dependent on congestion and accident proximity
  - Position reports and historical query requests
  - `Tuple<Type = 0, Time, VID, Spd, XWay, Lane, Dir, Seg, Pos>`
- Both `stateful` and `stateless` operators
- Metrics: Throughput, Latency and Energy

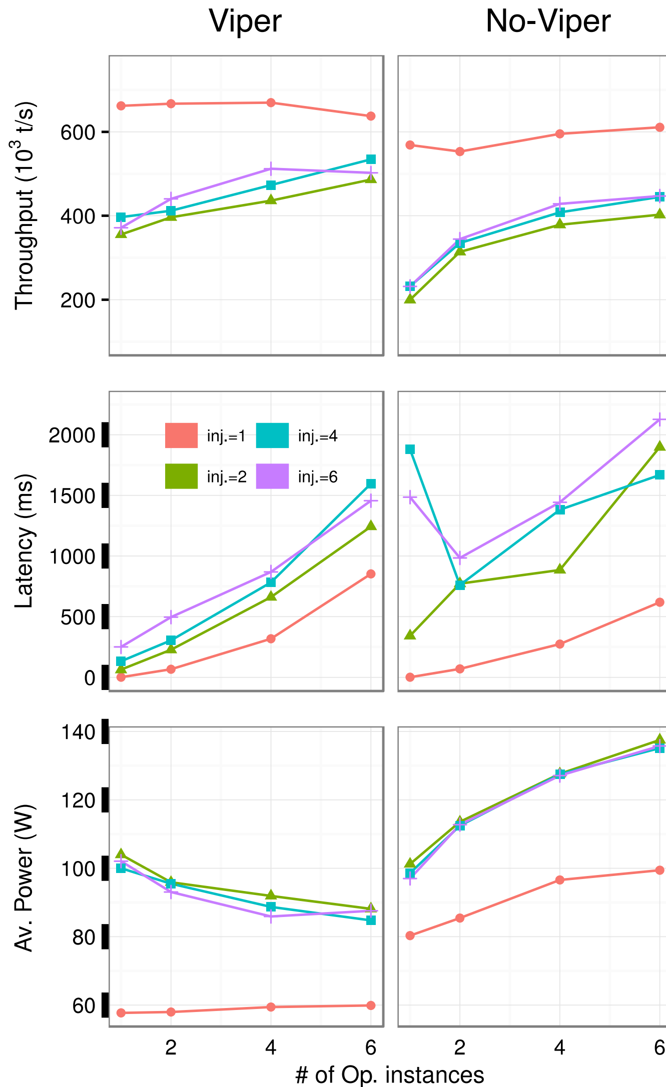


Evaluation Setup

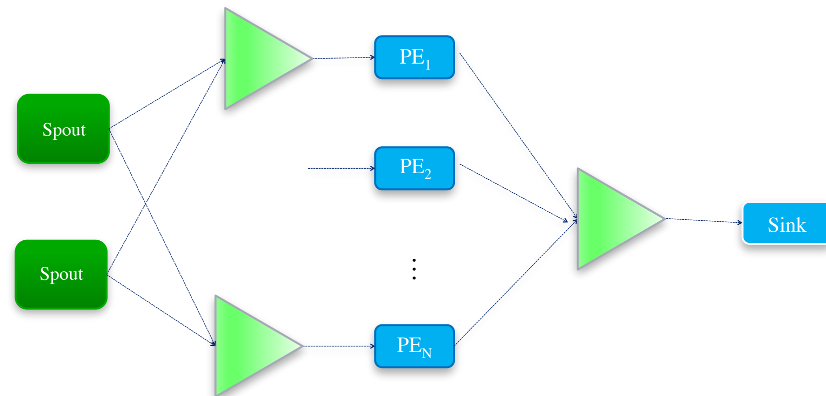
- Linear-Road Dataset
  - Simulate vehicular traffic on a network of dynamic-toll roads
  - Variable toll dependent on congestion and accident proximity
  - Position reports and historical query requests
  - `Tuple<Type = 0, Time, VID, Spd, XWay, Lane, Dir, Seg, Pos>`
- Both `stateful` and `stateless` operators
- Metrics: Throughput, Latency and Energy
- Evaluation Platform
  - Intel Xeon E5-2687W v2 3.4 GHz server (32 threads over 2 sockets), 64 GB of RAM
  - *Likwid* library to read RAPL energy counters

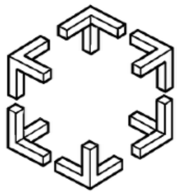


Evaluation Results

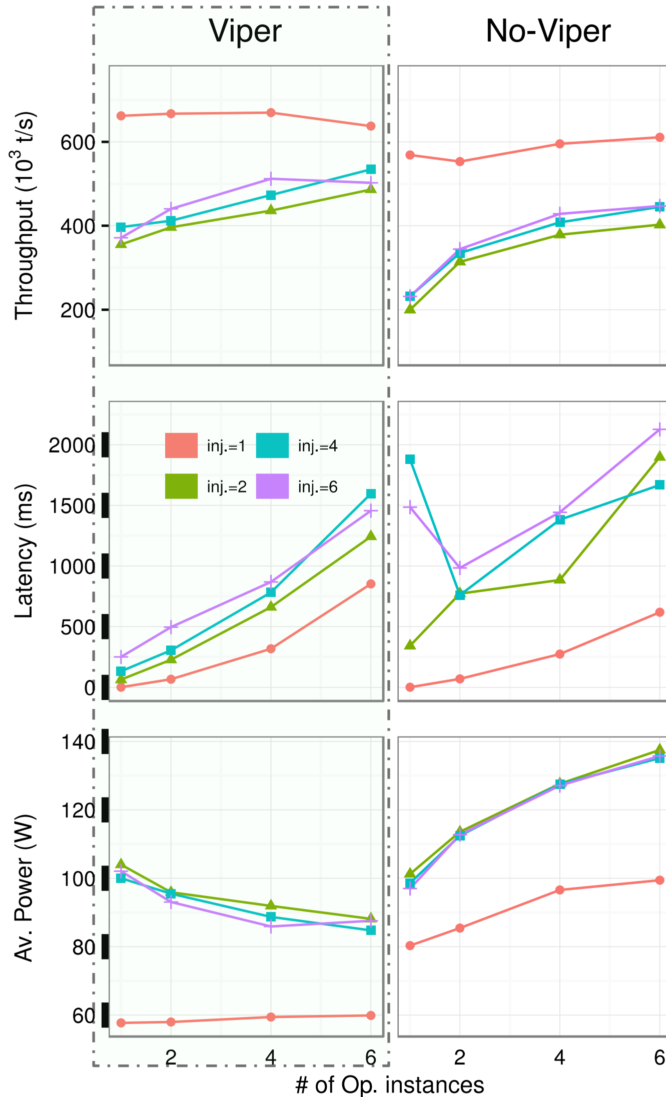


- Stateless Operator
  - Forward position reports
- Viper : Communication Layer
  - Viper Module used
- No-Viper : Operator Layer
  - Merge-Sort operator deployed
- Injection rate varied from 10,000 t/s to 1,200,000 t/s

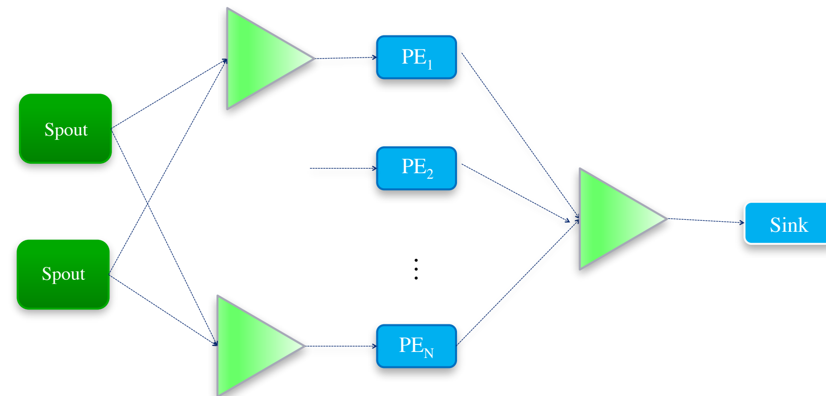


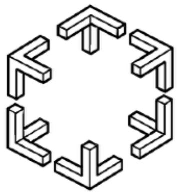


Evaluation Results

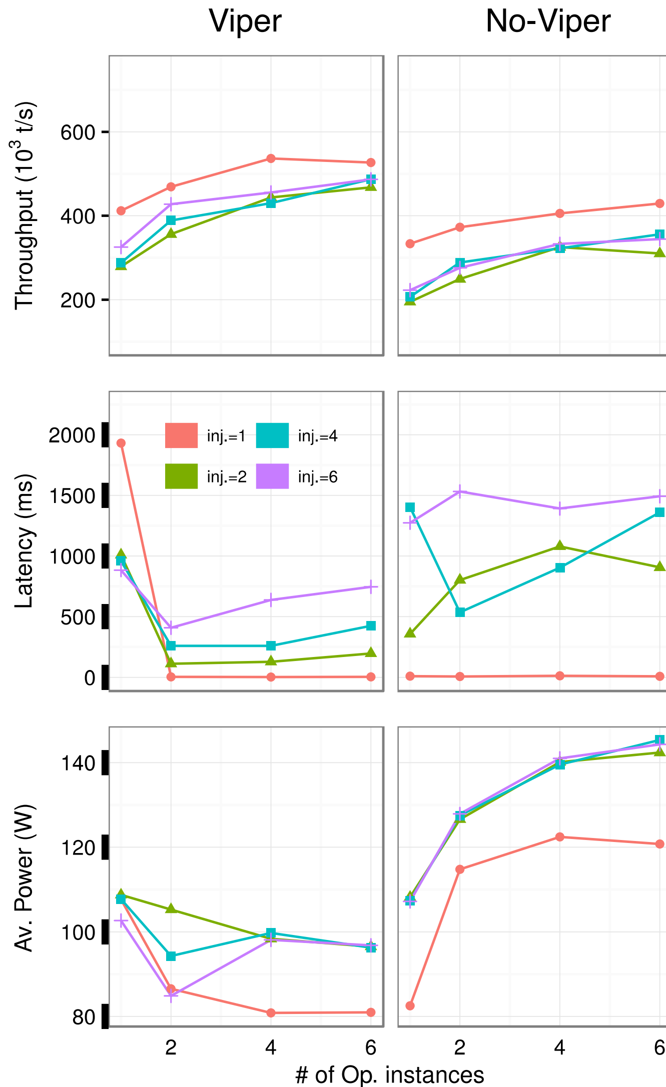


- Stateless Operator
  - Forward position reports
- Viper : Communication Layer
  - Viper Module used
- No-Viper : Operator Layer
  - Merge-Sort operator deployed
- Injection rate varied from 10,000 t/s to 1,200,000 t/s

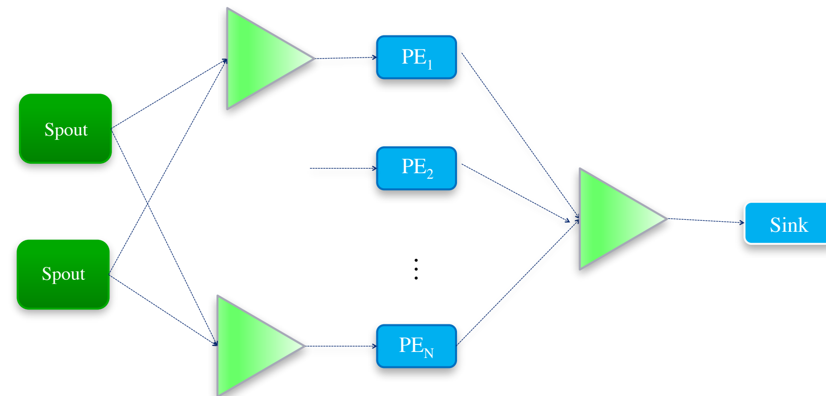


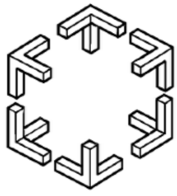


Evaluation Results



- Stateful Operator
  - Vehicle entering new Segment
- Viper : Communication Layer
  - Viper Module used
- No-Viper : Operator Layer
  - Merge-Sort operator deployed

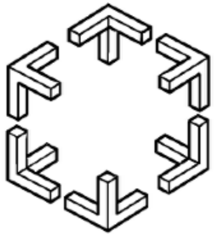




Conclusions and Future work

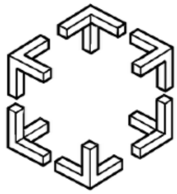
- Discussed limitations of operator layer determinism and proposed a solution to overcome these at the communication layer.
- Developed a Viper module that can be integrated in SPEs
- Evaluated the performance of the proposed module on Apache Storm

- Scale the per tuple workload in the evaluation



Viper: Communication-Layer Determinism and Scaling in Low-Latency Stream Processing

Thank You!



References

- **S. Schneider, M. Hirzel, B. Gedik and K. L. Wu** – Schneider 2015,
  "*Safe Data Parallelism for General Streaming*"
  in IEEE Transactions on Computers, 2015.
- **V. Gulisano, Y. Nikolakopoulos, M. Papatriantafilou and P. Tsigas** – Gulisano 2016,
  "*ScaleJoin: a Deterministic, Disjoint-Parallel and Skew-Resilient Stream Join*"
  in IEEE Transactions on Big Data, 2016.