# EdgeWise: A Better Stream Processing Engine for the Edge

***<u>Xinwei Fu</u>***, Talha Ghaffar, James C. Davis, Dongyoon Lee

**Department of Computer Science**

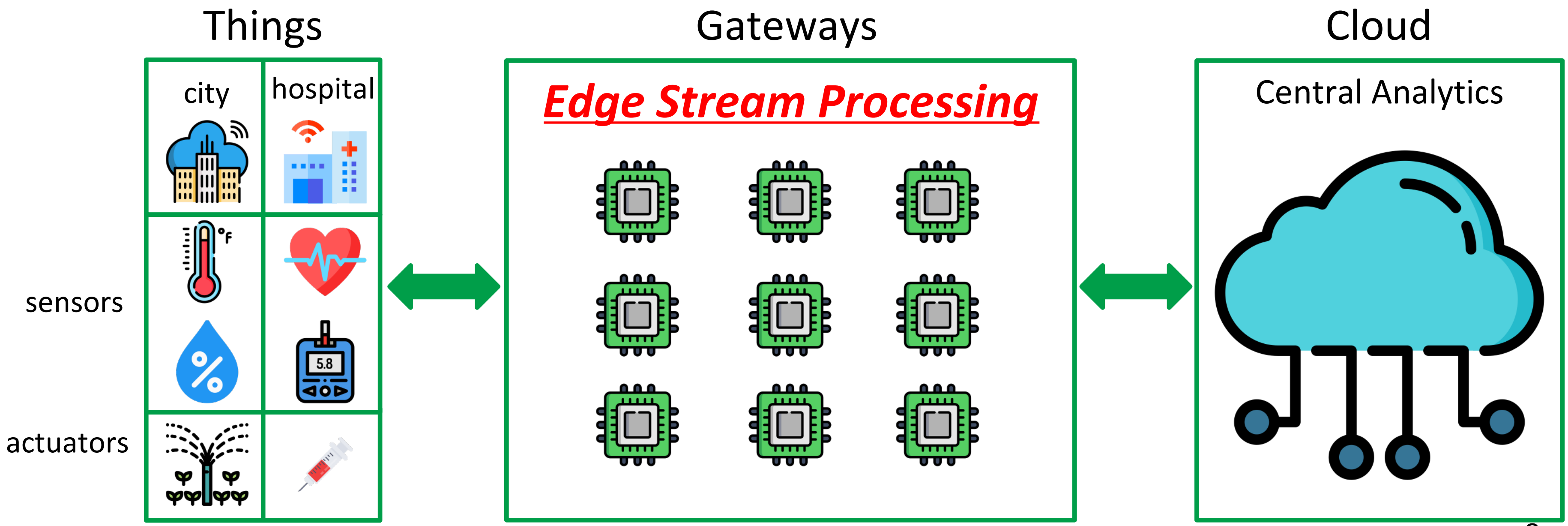
# Edge Stream Processing

Internet of Things (IoT)

- Things, Gateways and Cloud

Edge Stream Processing

- Gateways process continuous streams of data in a timely fashion.



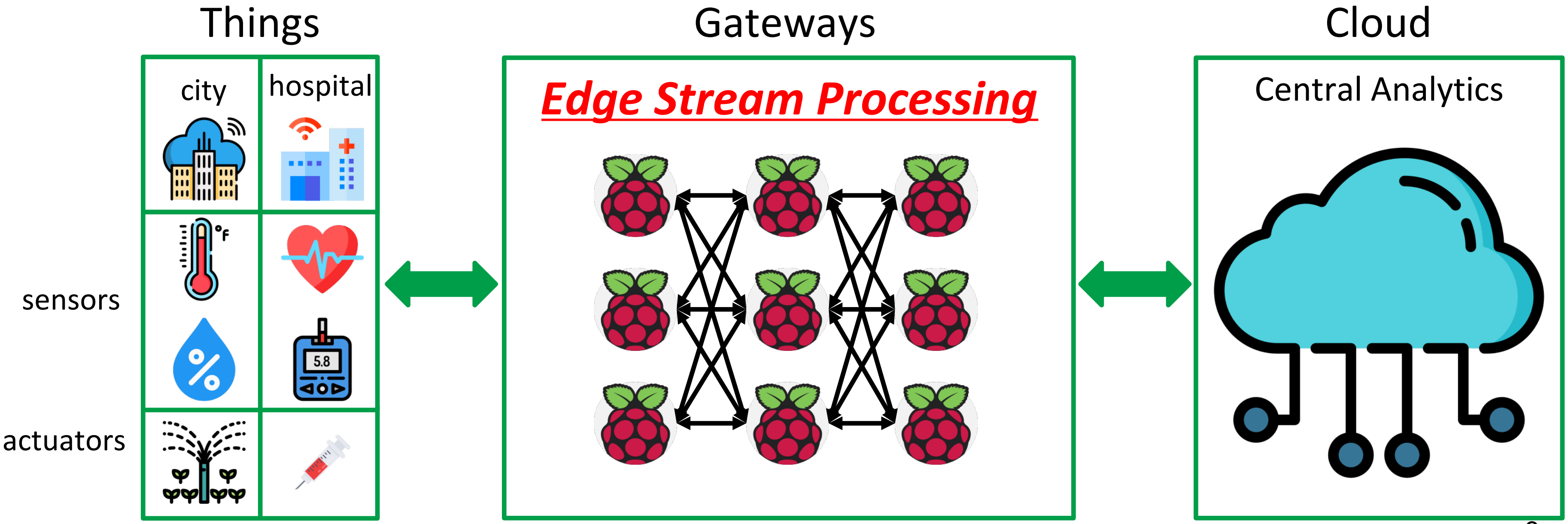
# Our Edge Model

Hardware

- Limited resources
- Well connected

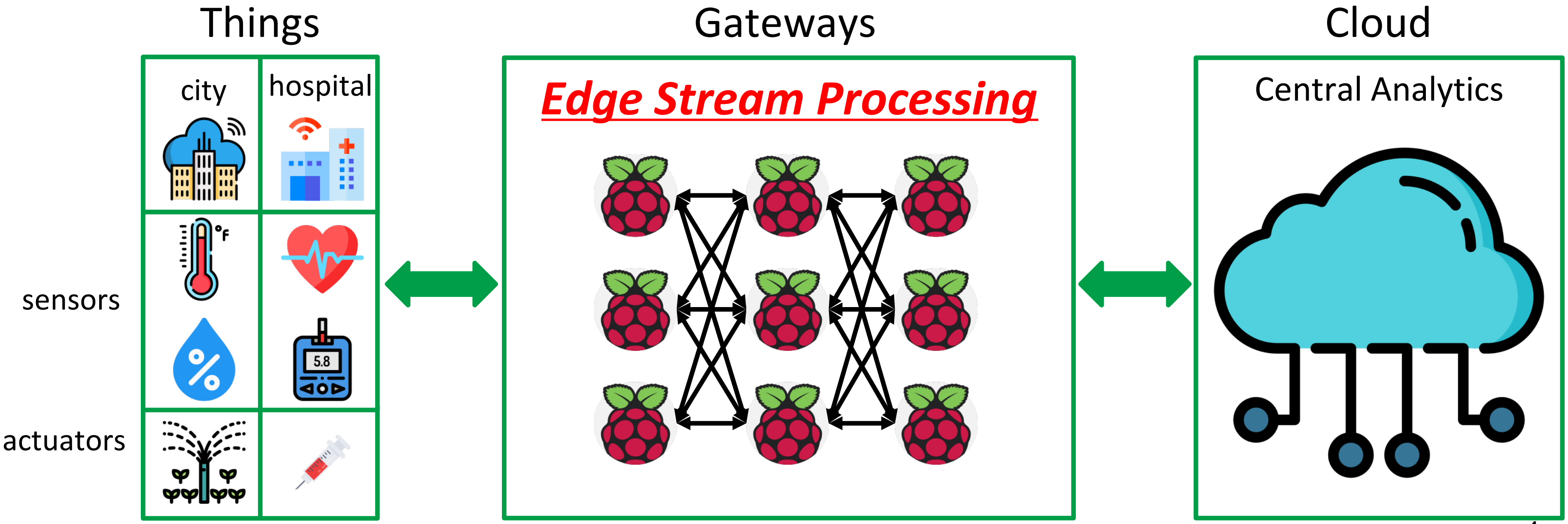
Application

- Reasonable complex operations
- For example, FarmBeats [NSDI'17]



# Edge Stream Processing Requirements

- Multiplexed - Limited resources
- No Backpressure - latency and storage
- Low Latency - Locality
- Scalable - millions of sensors



# Dataflow Programming Model

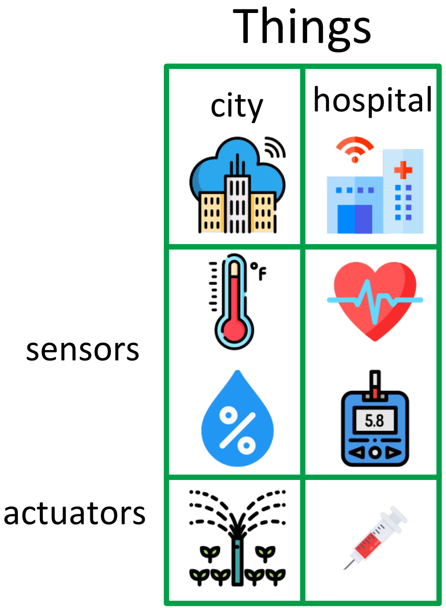
Topology - a Directed Acyclic Graph

■ source ◆ sink

○ operation → data flow

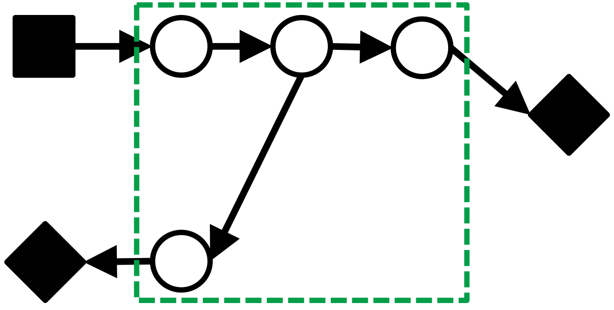
Deployment

- Describe # of instances for each operation




Gateways

*Edge Stream Processing*



Running on gateways

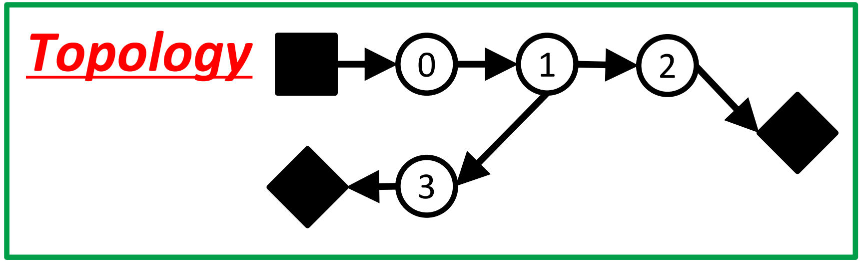
Cloud

Central Analytics

# Runtime System

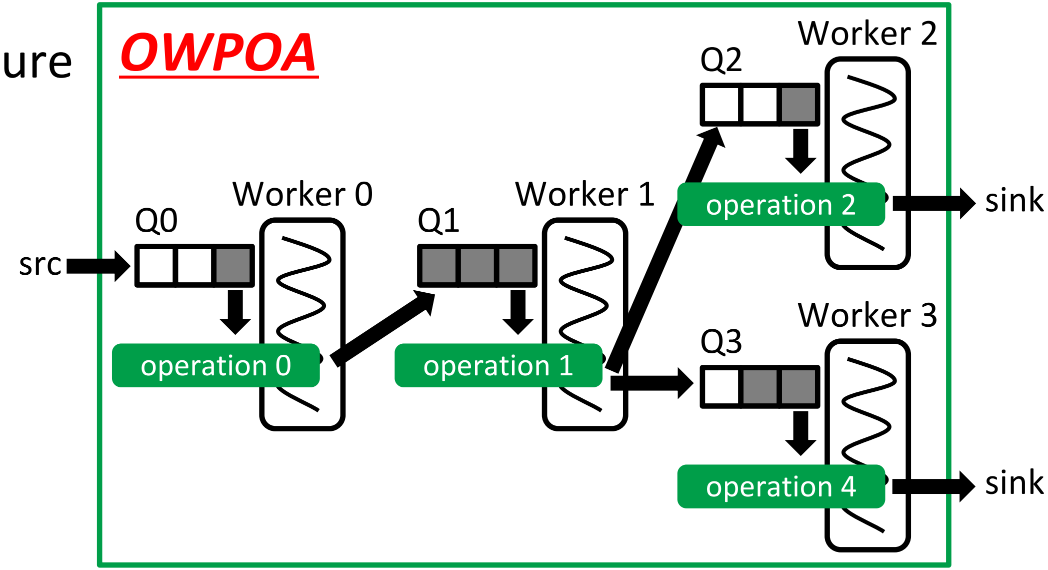
Stream Processing Engines (SPEs):

- Apache Storm
- Apache Flink
- Apache Heron




One-Worker-per-Operation-Architecture

- Queue and Worker thread
- Pipelined manner
- Backpressure
  - latency
  - storage



# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✔ ___Scalable ✘ ___Multiplexed ✘ ___Latency ✘ ___Backpressure

OWPOA SPEs

- Cloud-class resources
- OS scheduler

Edge

- Limited resources
- ***# of workers > # of CPU cores***
- ***Inefficiency in OS scheduler***

Low input rate → Most queues are empty

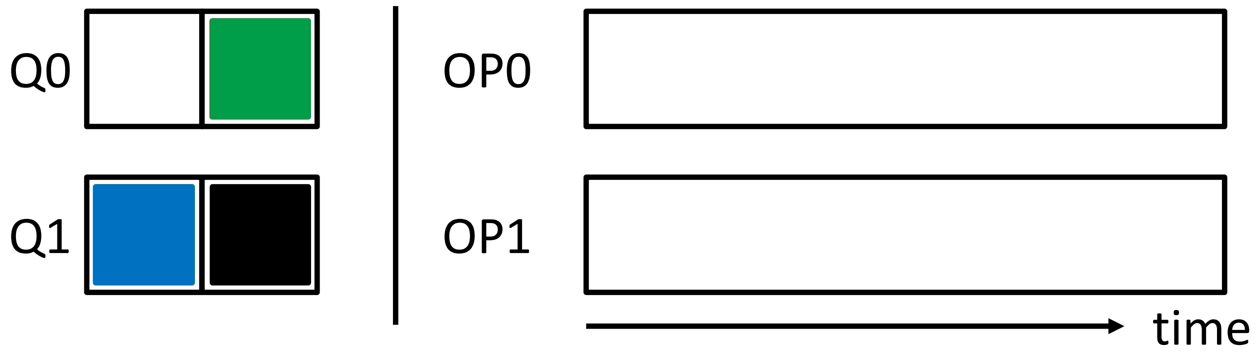
High input rate → Most or all queues contain data
→ Scheduling Inefficiency
→ Backpressure
→ Latency

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✓ ___Scalable ✗ ___Multiplexed ✗ ___Latency ✗ ___Backpressure

*Single core*
*Q0 -> Q1*



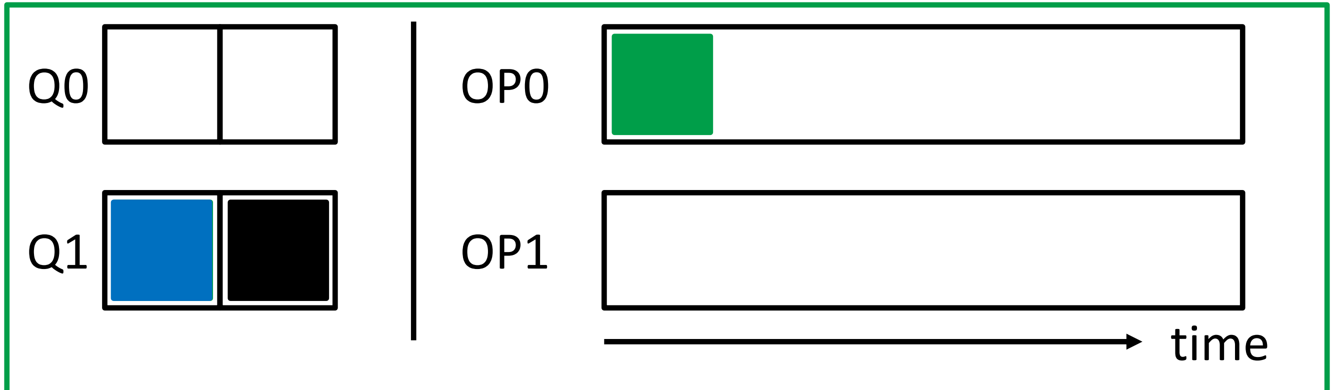

***OWPOA – Random OS Scheduler***

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✓ Scalable ✗ Multiplexed ✗ Latency ✗ Backpressure

*Single core*
*Q0 -> Q1*



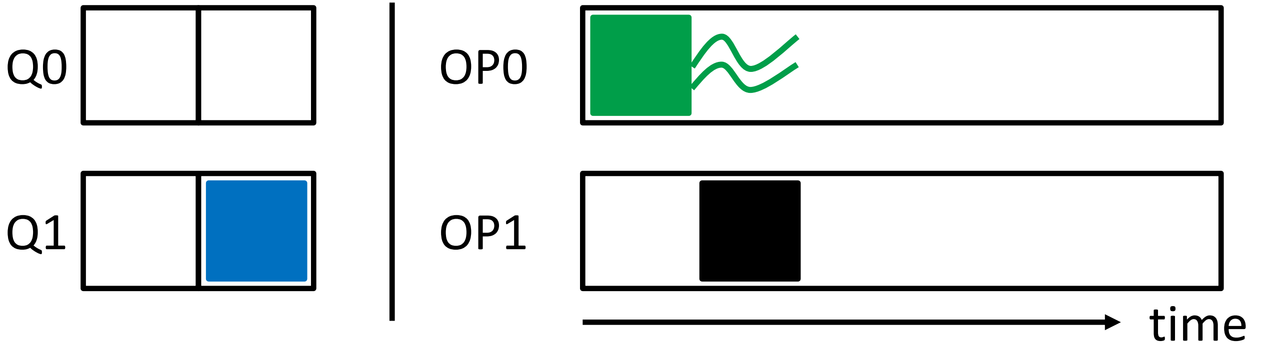

***OWPOA – Random OS Scheduler***

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✓ Scalable ✗ Multiplexed ✗ Latency ✗ Backpressure

*Single core*
*Q0 -> Q1*



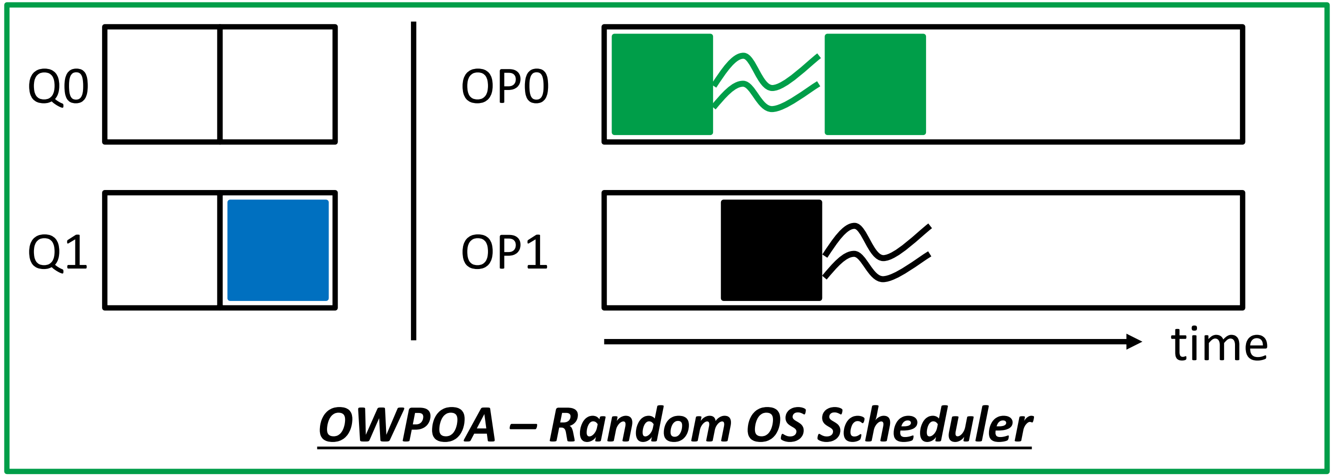

***OWPOA – Random OS Scheduler***

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✔ Scalable ✘ Multiplexed ✘ Latency ✘ Backpressure

*Single core*
*Q0 -> Q1*

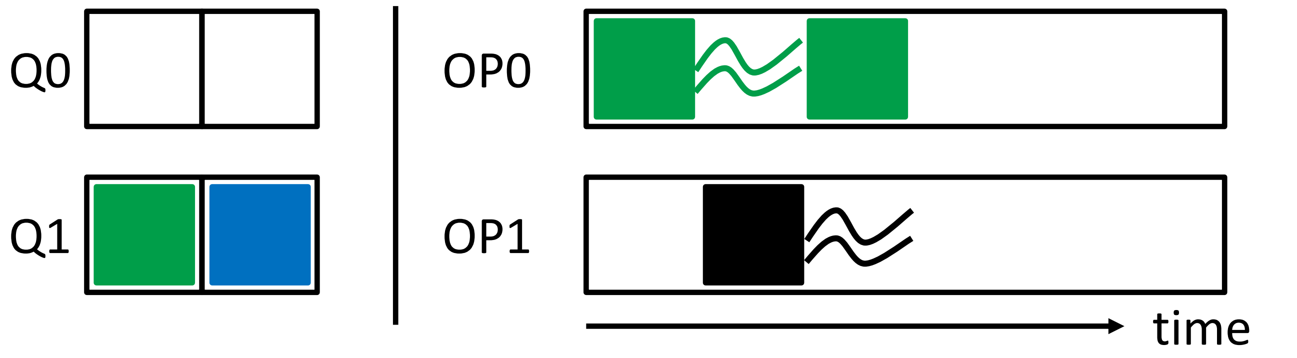


# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✔ Scalable ✘ Multiplexed ✘ Latency ✘ Backpressure

*Single core*
*Q0 -> Q1*



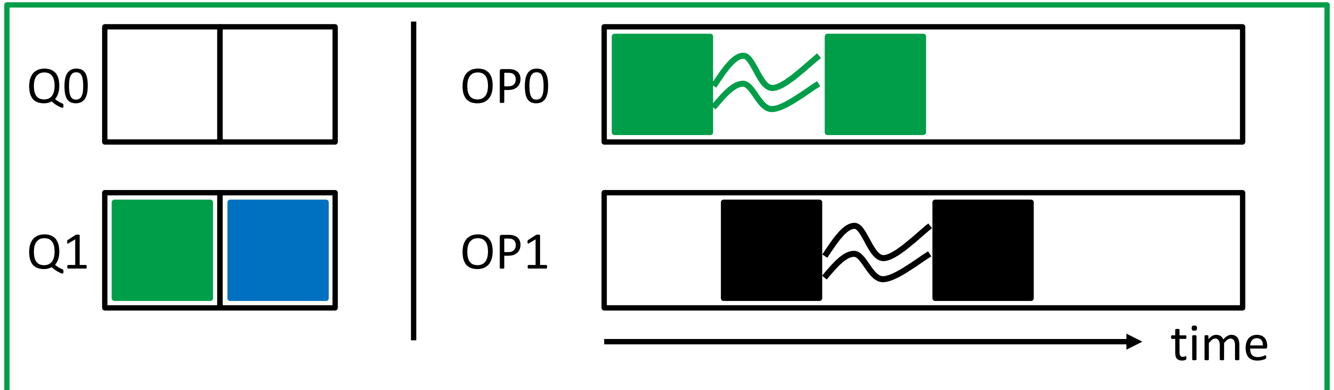

***OWPOA – Random OS Scheduler***

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✔ Scalable ✘ Multiplexed ✘ Latency ✘ Backpressure

*Single core*
*Q0 -> Q1*



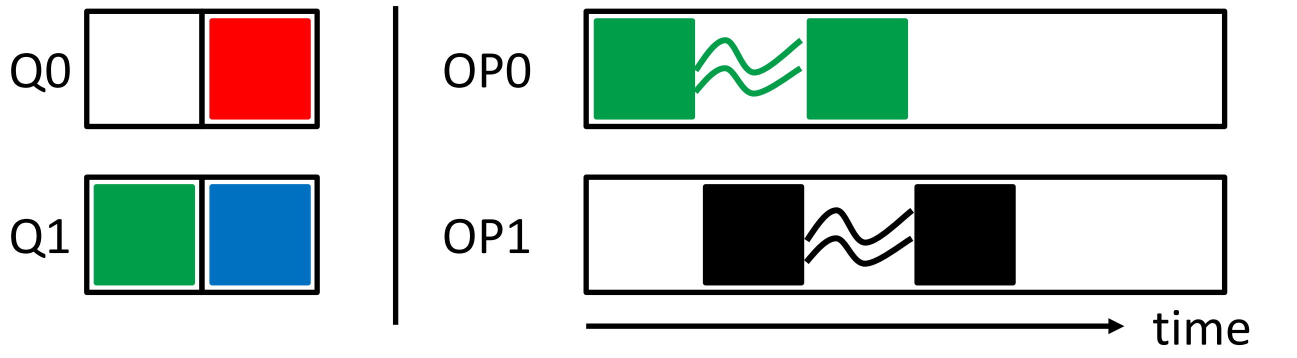

***OWPOA – Random OS Scheduler***

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✓ Scalable ✗ Multiplexed ✗ Latency ✗ Backpressure

*Single core*
*Q0 -> Q1*



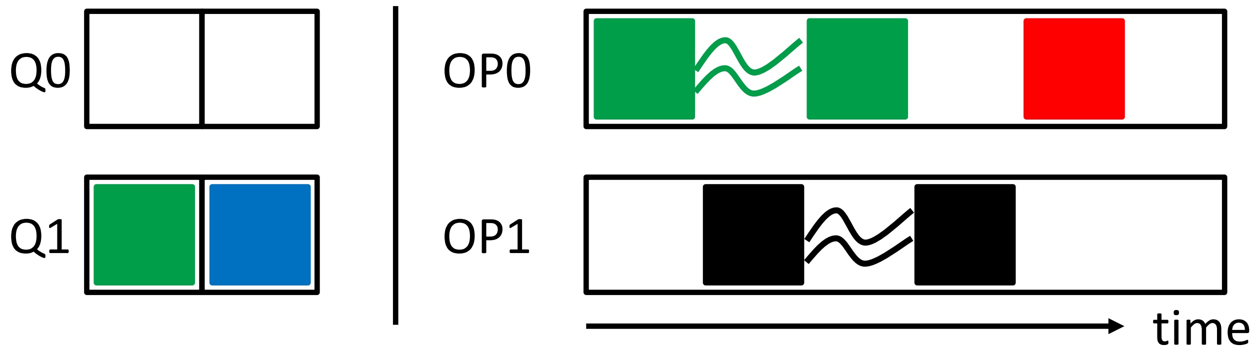

***OWPOA – Random OS Scheduler***

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

✓ ___Scalable ✗ ___Multiplexed ✗ ___Latency ✗ ___Backpressure

*Single core*
*Q0 -> Q1*



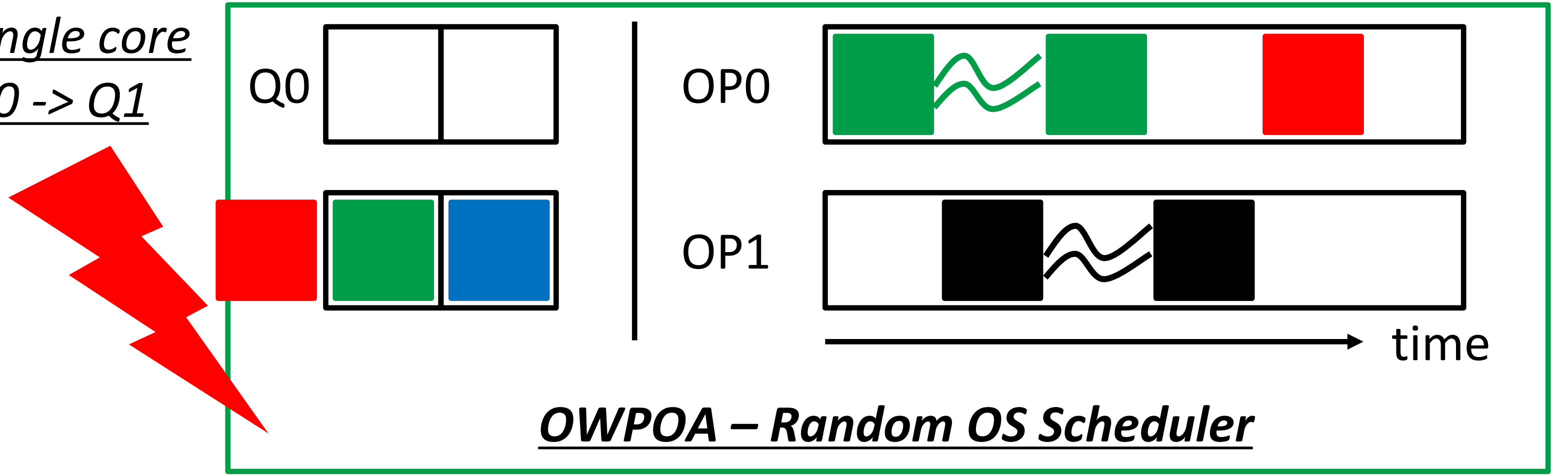

***OWPOA – Random OS Scheduler***

# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

___Scalable ___Multiplexed ___Latency ___Backpressure

*Single core*
*Q0 -> Q1*

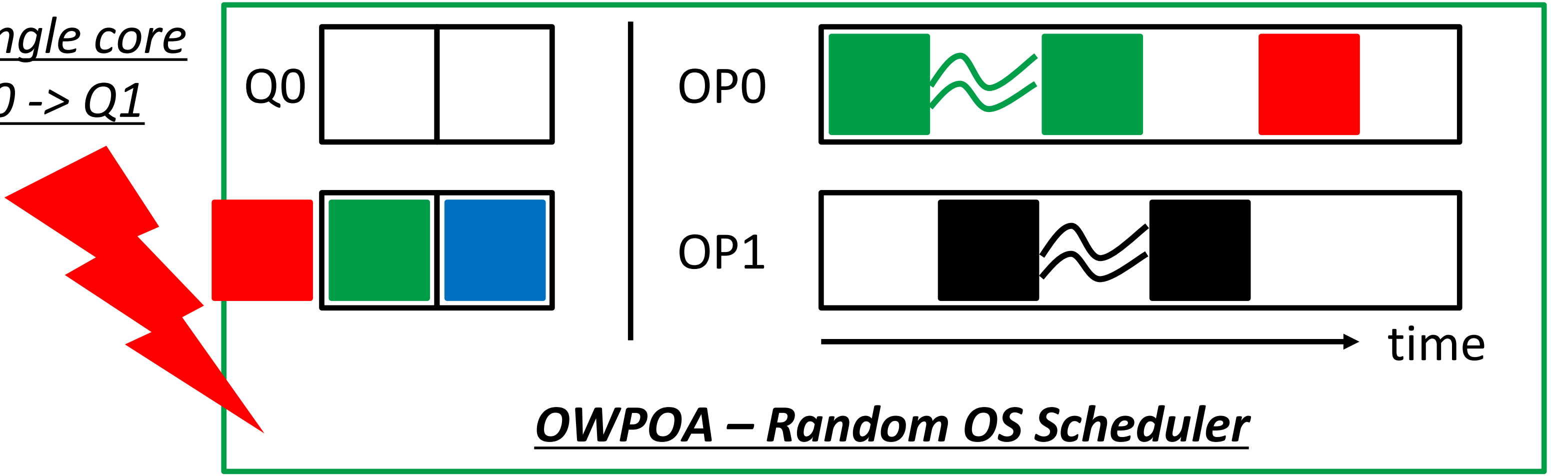


# Problem

Existing *One-Worker-per-Operation-Architecture Stream Processing Engines* are not suitable for the Edge Setting!

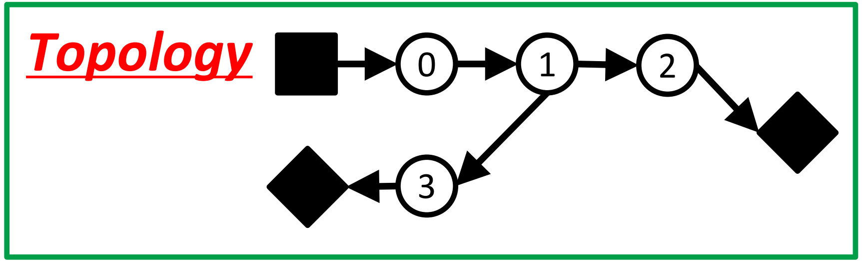
✓ Scalable ✗ Multiplexed ✗ Latency ✗ Backpressure

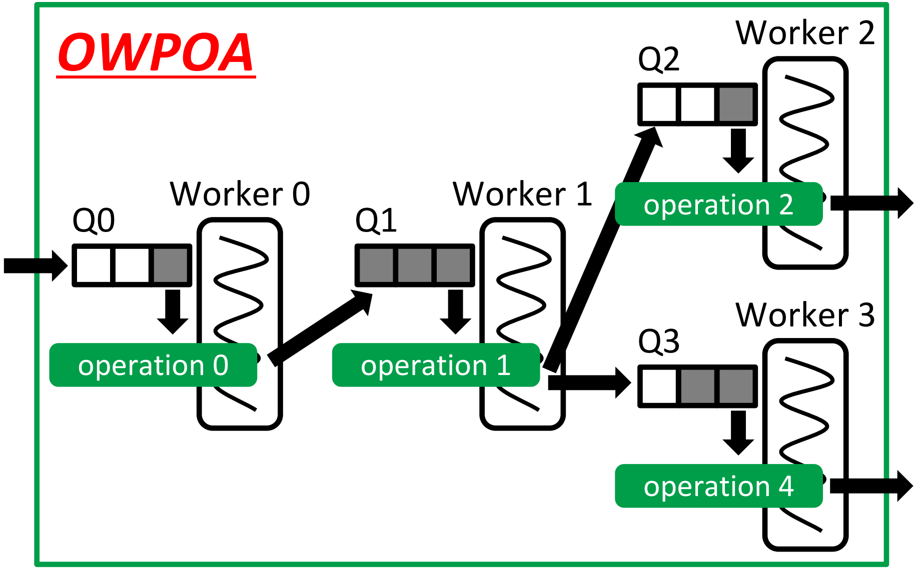
*Single core*
*Q0 -> Q1*




*OS Scheduler doesn't have engine-level knowledge.*

# EdgeWise



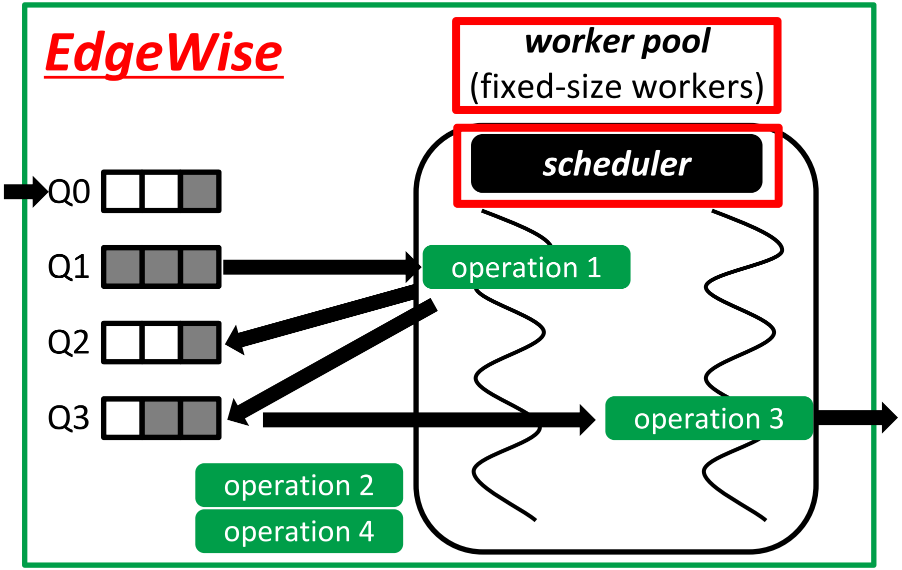

Key Ideas:

- # of workers > # of CPU cores

→ ***A fixed-sized worker pool***

- Inefficiency in OS scheduler

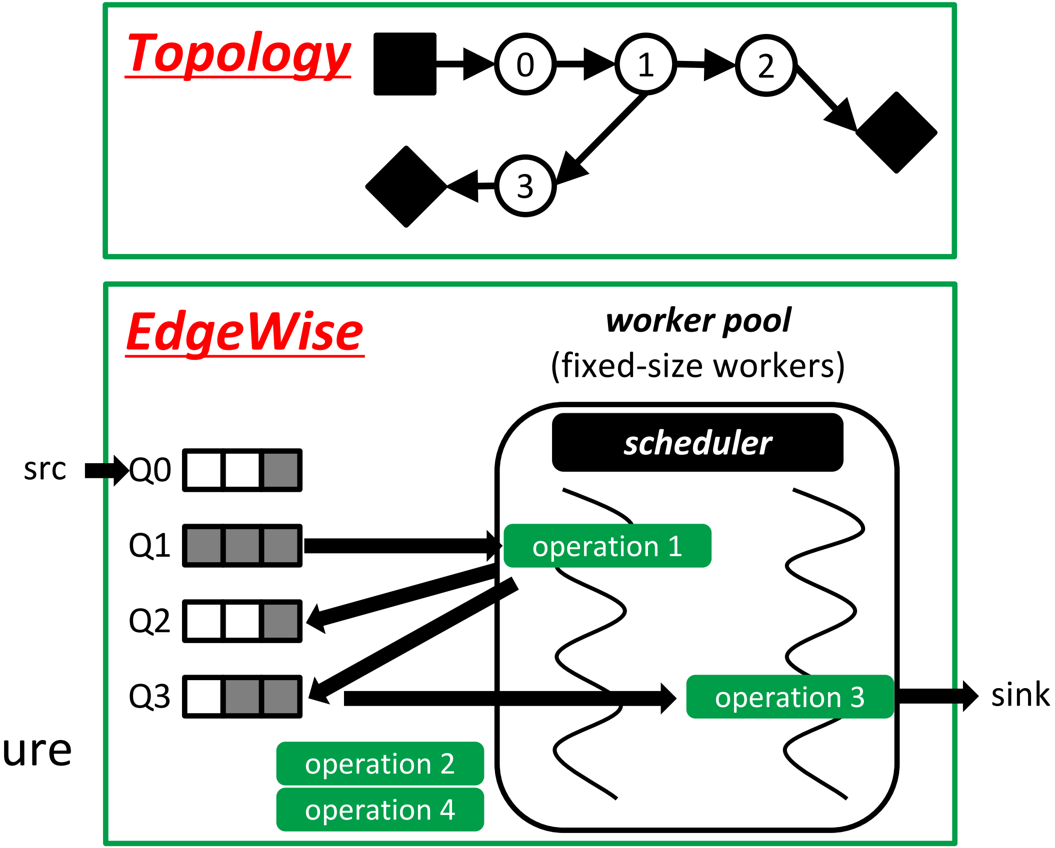
→ ***Engine-level scheduler***



# EdgeWise – Fixed-size Worker Pool

Fixed-size Worker Pool

- # of worker = # of cores
- Support an arbitrary topology on limited resources
- Reduce overhead of contending cores




✔ Scalable ✔ Multiplexed

___Latency ___Alleviate Backpressure

# EdgeWise – Engine-level Scheduler

A Lost Lesson: Operation Scheduling

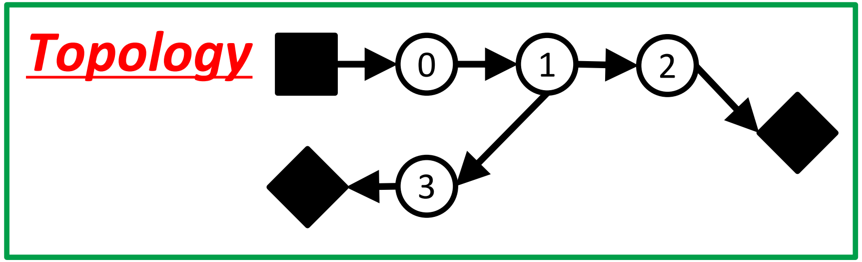
- Profiling-guided priority-based
- Multiple OPs with a single worker

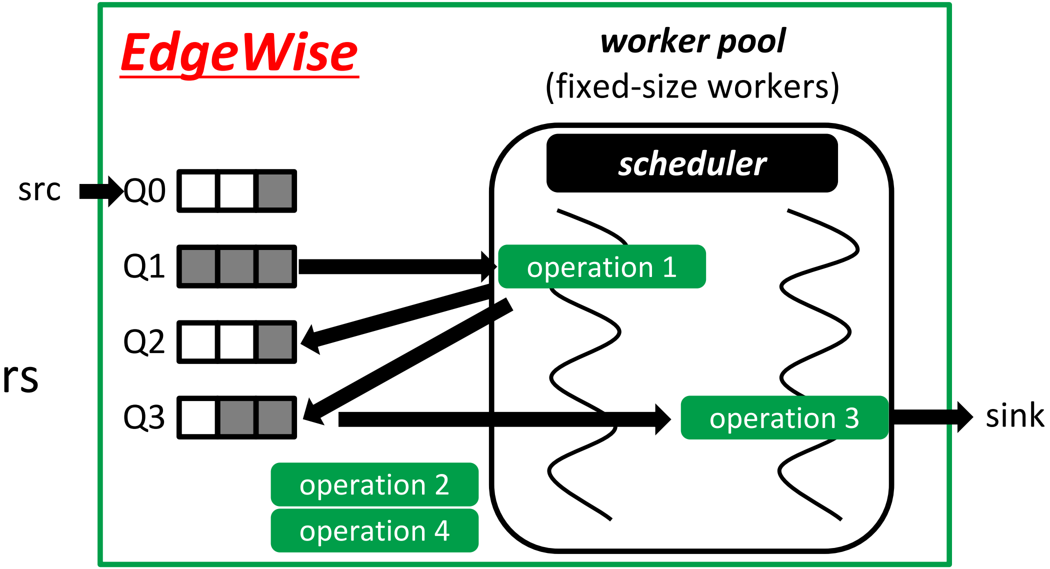
Carney [VLDB'03]

- Min-Latency Algorithm
- Higher static priority on latter OPs

Babcock [VLDB'04]

- Min-Memory Algorithm
- Higher static priority on faster filters




*We should regain the benefit of the engine-level operation scheduling!!!*

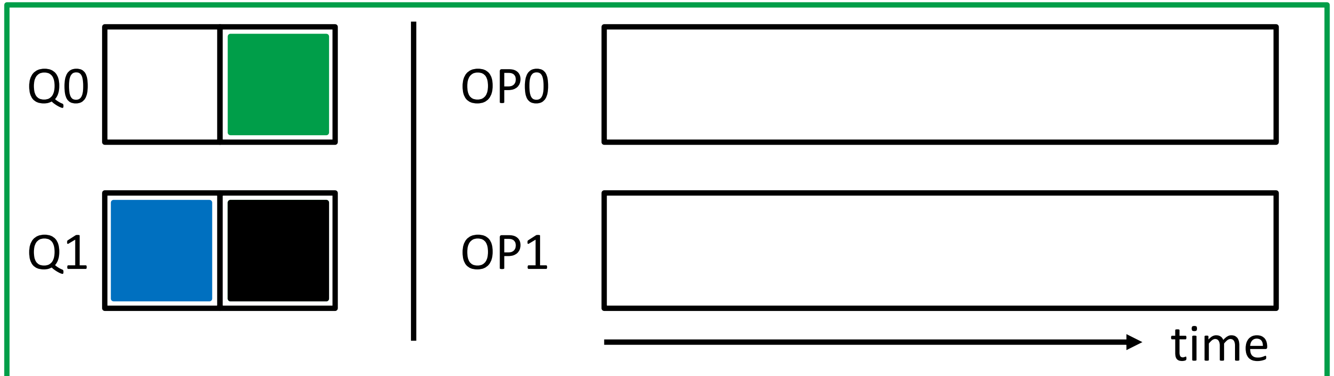
# EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

*Single core*

*Q0 -> Q1*




***EdgeWise – Congestion-Aware Scheduler***

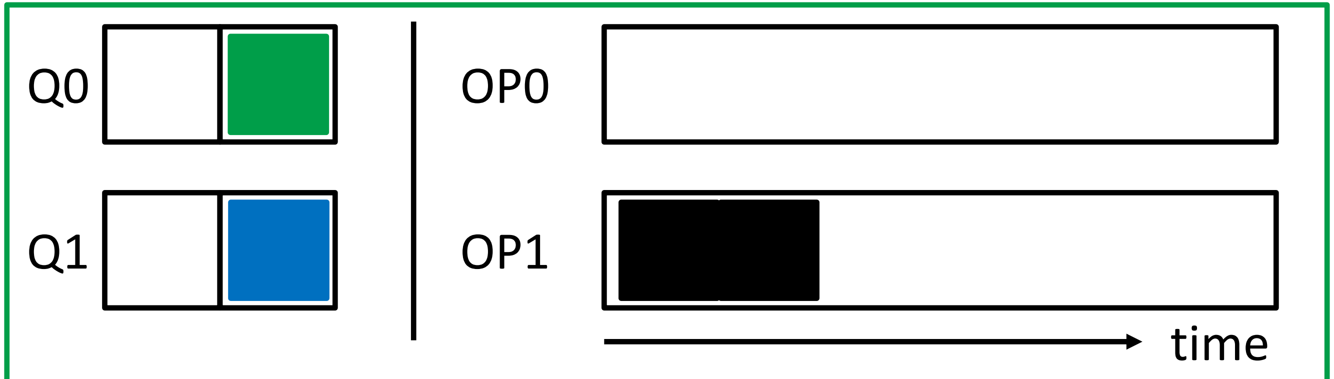
Scalable  Multiplexed  Latency  Alleviate Backpressure

# EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

*Single core*
*Q0 -> Q1*




***EdgeWise – Congestion-Aware Scheduler***

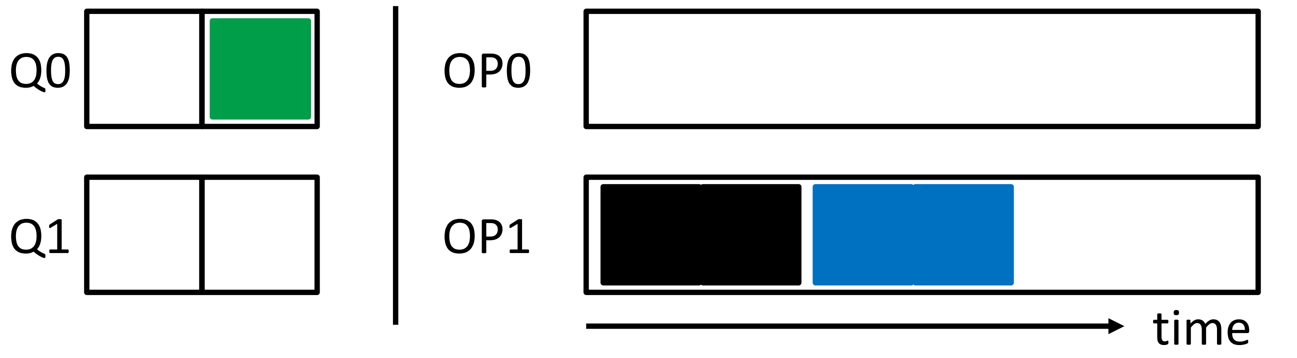
✓ Scalable ✓ Multiplexed ✓ Latency ✓ Alleviate Backpressure

# EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

*Single core*
*Q0 -> Q1*




***EdgeWise – Congestion-Aware Scheduler***

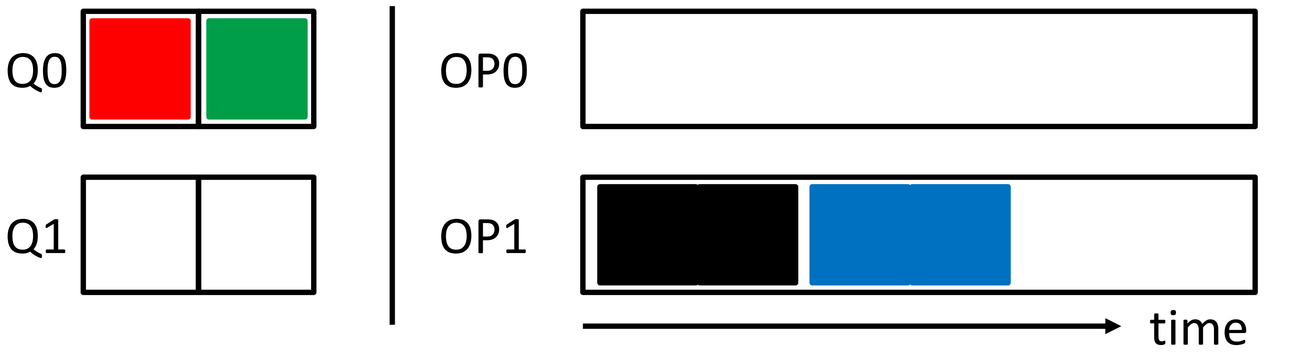
✔ Scalable ✔ Multiplexed ✔ Latency ✔ Alleviate Backpressure

# EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

*Single core*
*Q0 -> Q1*




***EdgeWise – Congestion-Aware Scheduler***

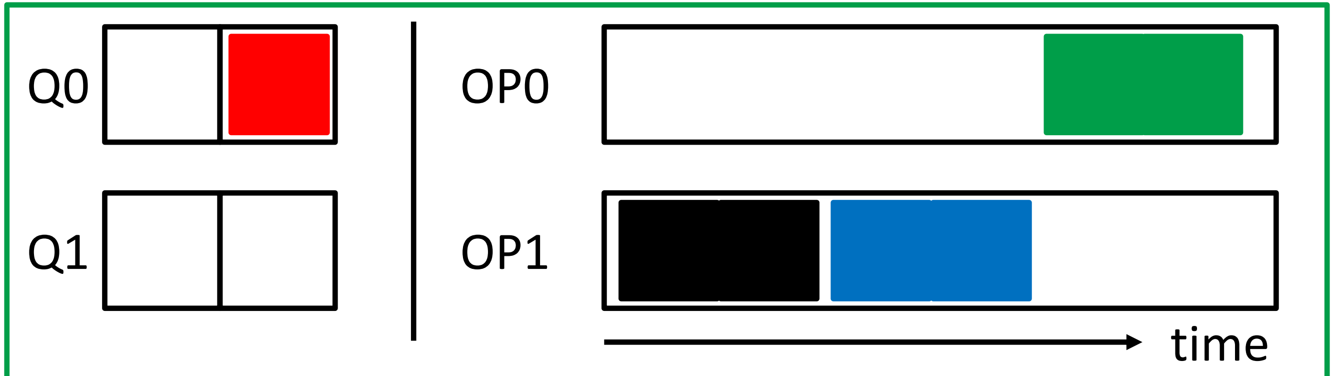
✓ Scalable ✓ Multiplexed ✓ Latency ✓ Alleviate Backpressure

# EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

*Single core*
*Q0 -> Q1*




***EdgeWise – Congestion-Aware Scheduler***

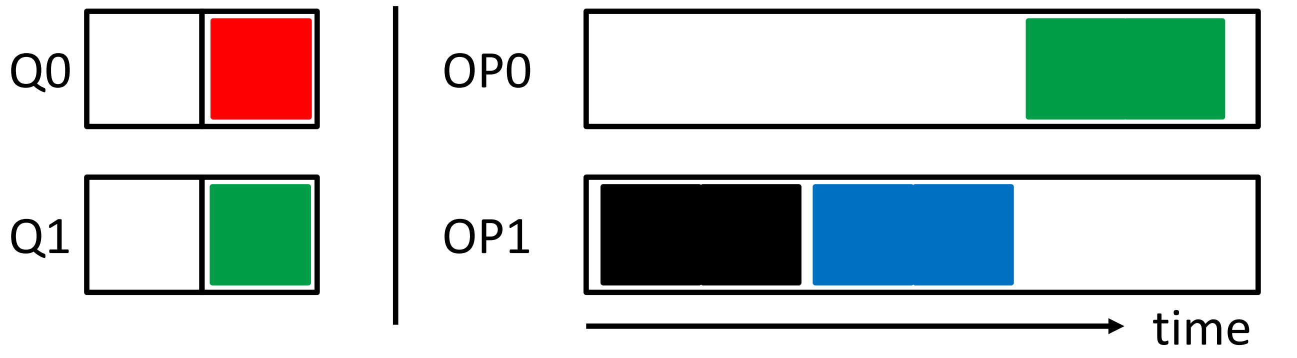
✔ Scalable ✔ Multiplexed ✔ Latency ✔ Alleviate Backpressure

# EdgeWise – Engine-level Scheduler

Congestion-Aware Scheduler

- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data.

*Single core*
*Q0 -> Q1*




***EdgeWise – Congestion-Aware Scheduler***

✓ Scalable ✓ Multiplexed ✓ Latency ✓ Alleviate Backpressure

# Performance Analysis using Queueing Theory

**Novelty:**

*To the best of our knowledge, we are the first to apply queueing theory to analyze the improved performance in the context of stream processing.*

**Conclusion 1:**

*Maximum end-to-end throughput depends on scheduling heavier operations proportionally more than lighter operations.*

**Conclusion 2:**

*Data waits longer in the queues of heavier operations.*
*The growth in wait time is non-linear.*

By balancing queue sizes, EdgeWise achieves *Higher Throughput and Lower* Latency than One-Worker-per-Operation-Architecture.

*See our paper for more details.*

# Evaluation

**Impl:** Apache Storm v1.1.0

**Hardware:** v3

**OWPOA Baseline:** Apache Storm

**Schedulers:**

- Random
- Min-Memory
- Min-Latency

**Experiment Setup:**

Focus on a single

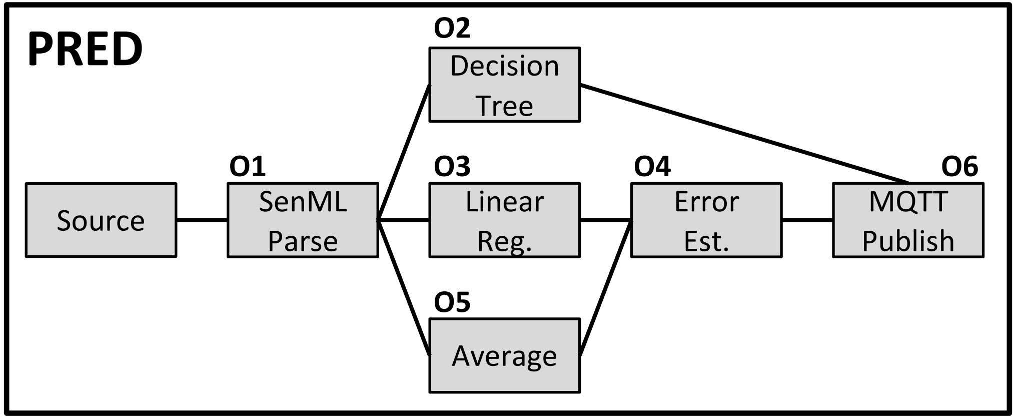
4 cores -> a pool of 4 worker threads

**Benchmarks:**

RIoTBench - a real-time IoT stream processing benchmark for Storm.

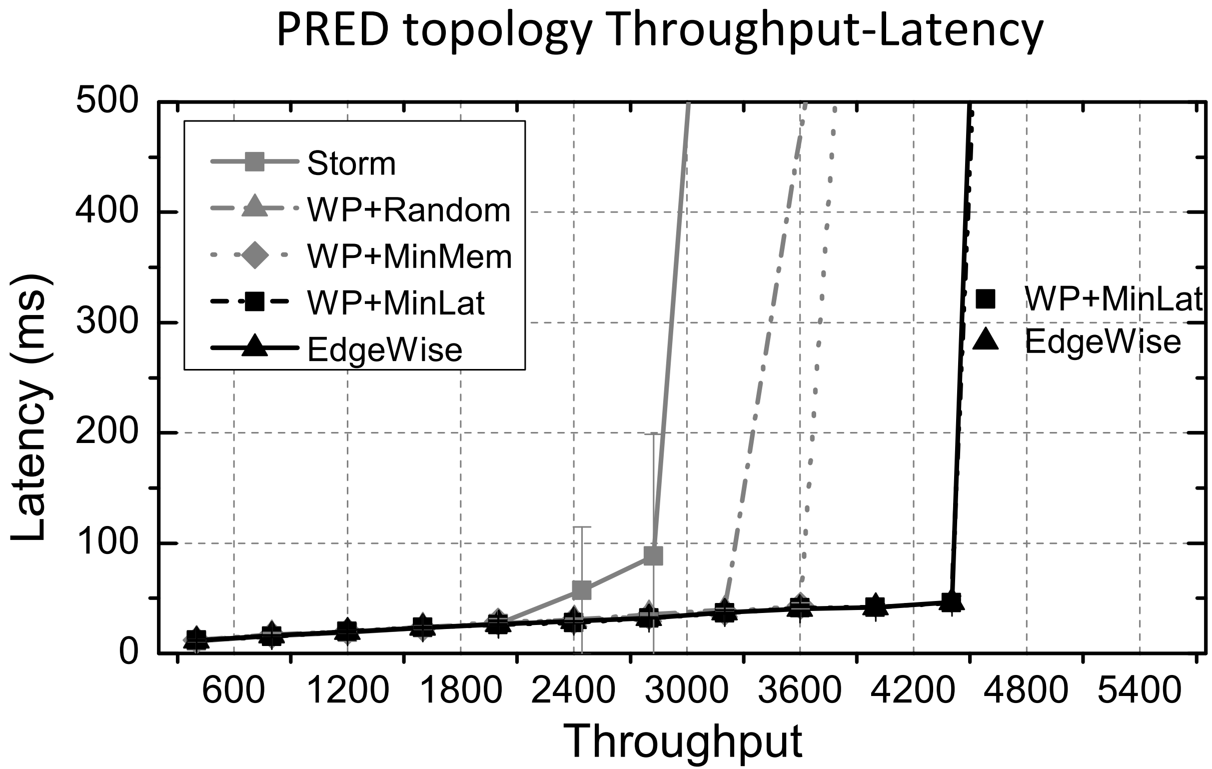
**Metrics:**

Throughput & Latency

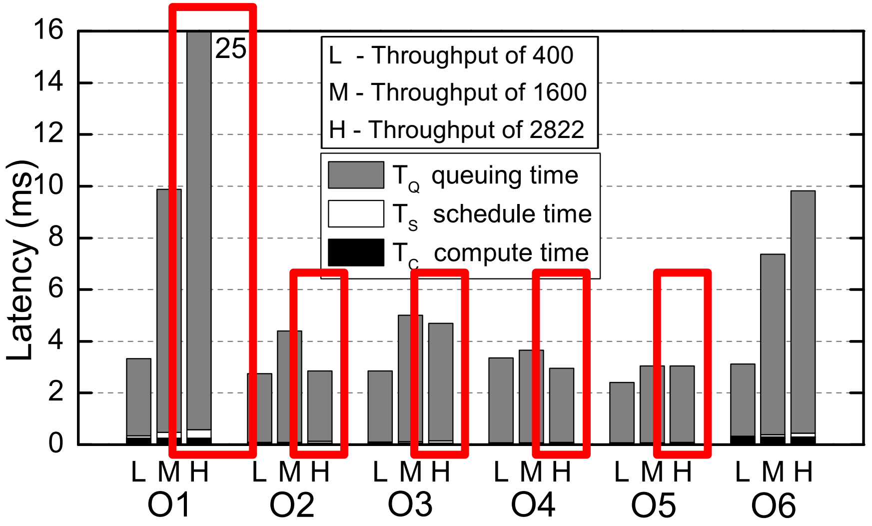



*More in the Paper.*

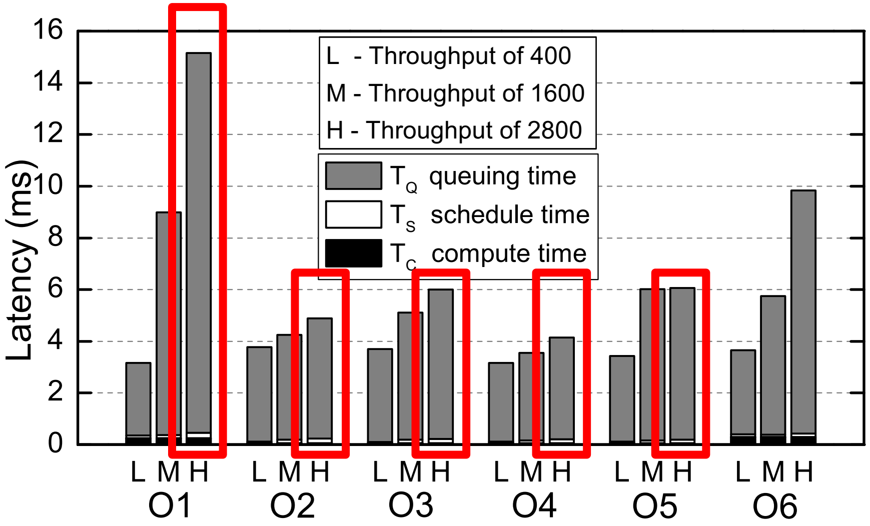
# Throughput-Latency Performance



# Fine-Grained Latency Analysis




PRED Latency breakdown in *Storm*




PRED Latency breakdown in *EdgeWise*

***This is not a zero-sum game!***

**Conclusion 2:**

*Data waits longer in the queues of heavier operations.*

*The growth in wait time is non-linear.*

# Conclusion

- Study existing SPEs and discuss their limitations in the Edge
- EdgeWise
  - Fixed-size worker pool
  - Congestion-aware scheduler
  - Lost lesson of operation scheduling
- Performance analysis of the congestion-aware scheduler using Queueing Theory
- Up to 3x improvement in throughput while keeping latency low

*Sometimes the answers in system design lie not in the future but in the past.*

# Backup Slides

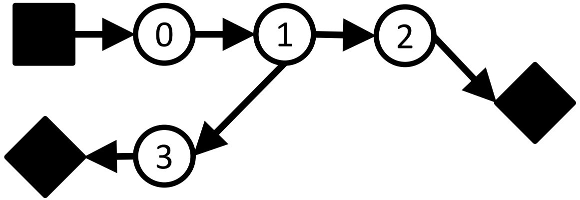
# Problem

Existing ***OWPOA SPEs*** are not suitable for the Edge Setting!

✔ ___Scalable ___Multiplexed ___Latency ___No Backpressure

# of instance of each operation can be assigned during the deployment

***Topology***



# Performance Analysis using Queueing Theory

**Novelty:**

*To the best of our knowledge, we are the first to apply queueing theory to analyze the improved performance in the context of stream processing.*

*Prior scheduling works in stream processing either provide no analysis or focus only on memory optimization.*

# Performance Analysis using Queueing Theory

**Conclusion 1:**

*Maximum end-to-end throughput depends on scheduling heavier operations proportionally more than lighter operations.*

Input rate

$$\lambda_i = q_i \cdot \lambda_0$$

Service rate

$$\mu_i = r_i \cdot \mu_0$$

Utilization

$$\rho_i = \frac{\lambda_i}{\mu_i}$$

Scheduling weight

$$\sum_i^M w_i = C$$

Effective service rate

$$\mu_i{}' = w_i \cdot \mu_i = w_i \cdot (r_i \cdot \mu_0)$$

Stable constraint

$$\forall i, \quad \rho_i{}' = \frac{\lambda_i}{\mu_i{}'} = \frac{q_i \cdot \lambda_0}{w_i \cdot r_i \cdot \mu_0} < 1$$

->

$$\forall i, \quad \lambda_0 < w_i \cdot \frac{r_i}{q_i} \cdot \mu_0$$

->

$$\text{maximize} \quad \min_i (w_i \cdot \frac{r_i}{q_i})$$

$$\text{subject to} \quad \sum_i^M w_i = C$$

$$w_1 : w_2 : \cdots : w_N = \frac{q_1}{r_1} : \frac{q_2}{r_2} : \cdots : \frac{q_M}{r_M}$$

scheduling weight -> input rate / service rate

# Performance Analysis using Queueing Theory

**Conclusion 2:**

*A data waits longer in the queues of heavier operations, and crucially the growth in wait time is non-linear.*

End-to-end latency

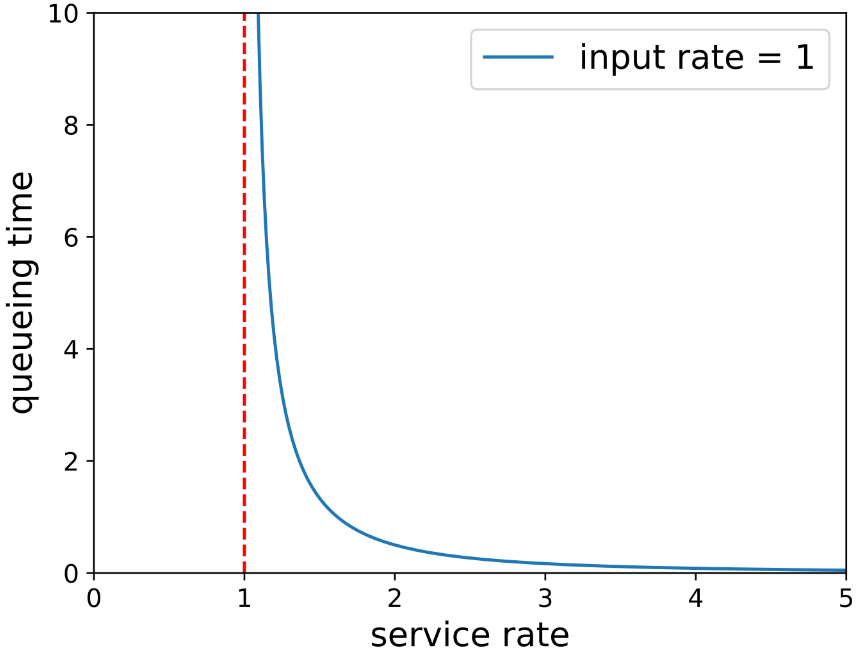
$$Latency = \sum_{i}^{M}(L_i + Comm.) \approx \sum_{i}^{M} L_i$$

Per-operation latency

$$L_i = T_Q + (T_S + T_C) \approx T_Q$$

Queueing time – waiting in the queue (using exponential distribution)

$$T_Q = \frac{\rho}{\mu - \lambda} = \frac{\lambda}{\mu(\mu - \lambda)}$$

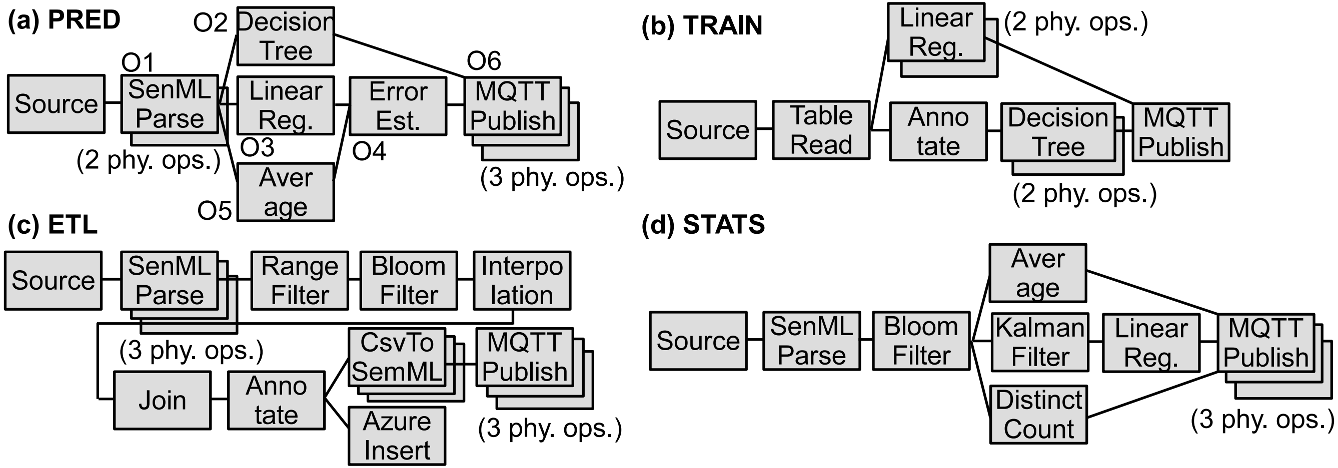


# Evaluation

**Benchmarks:**

RIoTBench - a real-time IoT stream processing benchmark for Storm

Modification: a timer-based input generator.



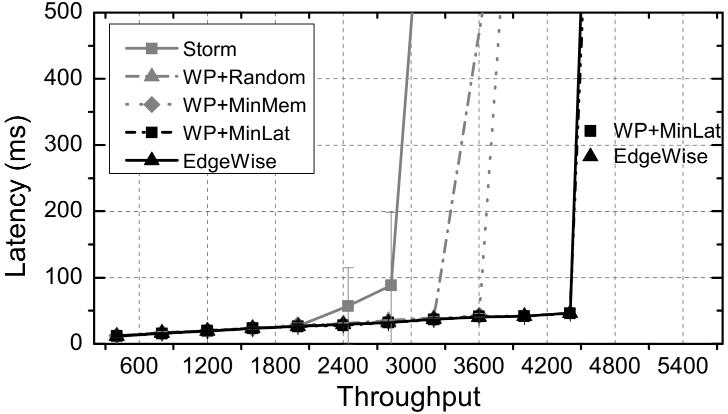

**Metrics:**

- Throughput - Latency

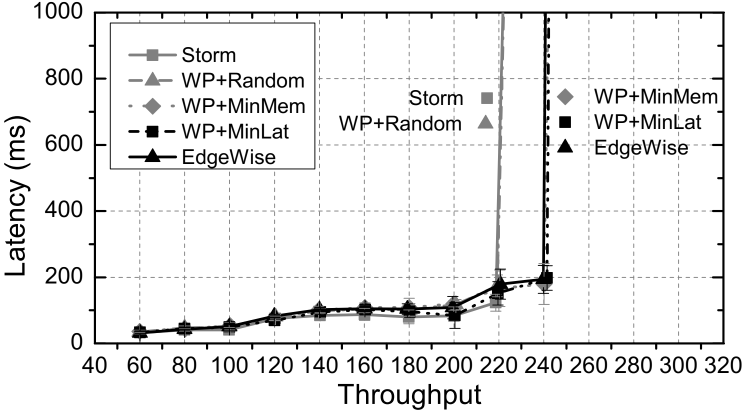
# Throughput-Latency Performance

VIRGINIA TECH

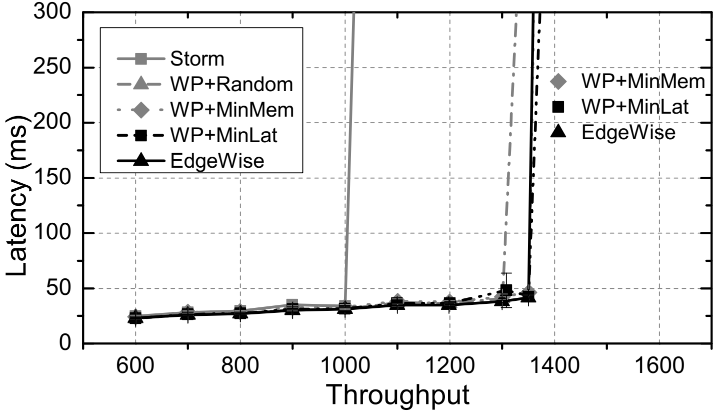
PRED



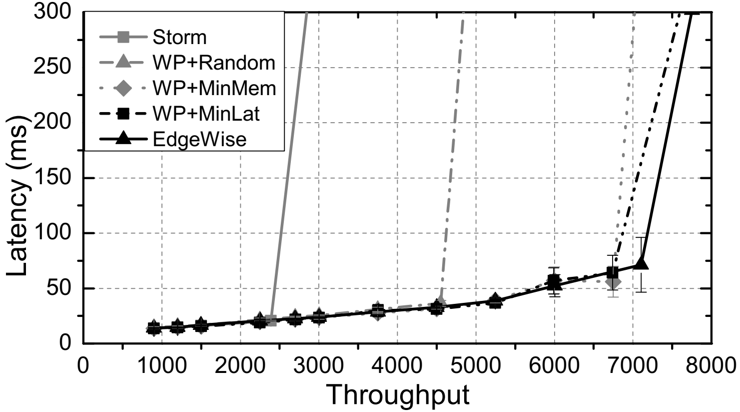

TRAIN



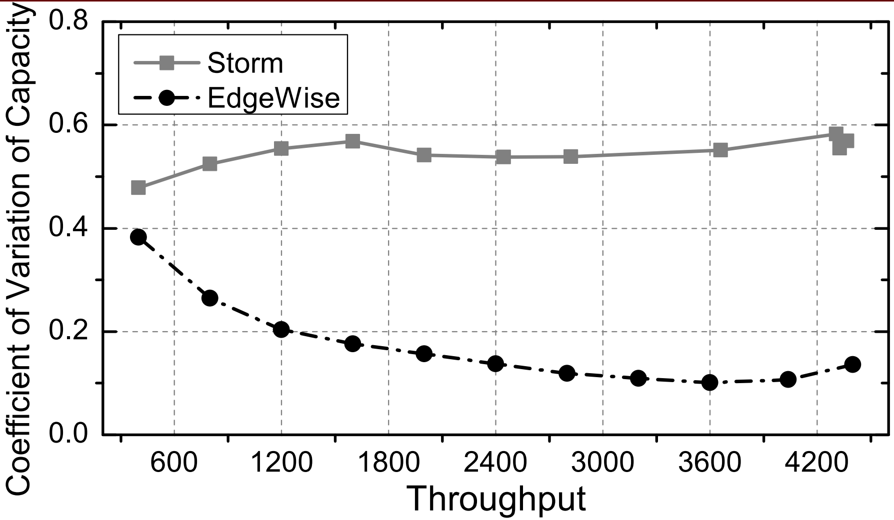

ETL




STATS



# Fine-Grained Throughput Analysis

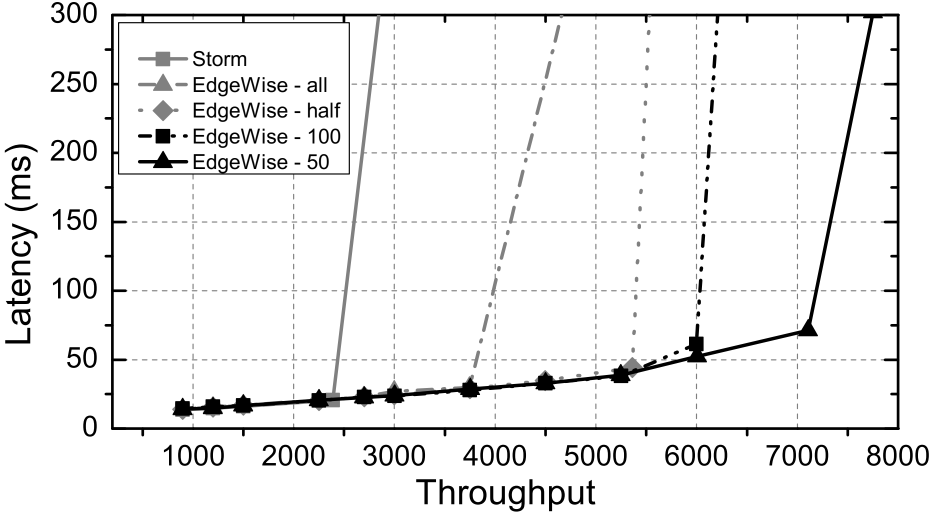



In PRED, as the input rate (throughput) increase, the *coefficient of variation (CV) of operation utilization* grows in Storm, but it decreases in EdgeWise

**Conclusion 1:**

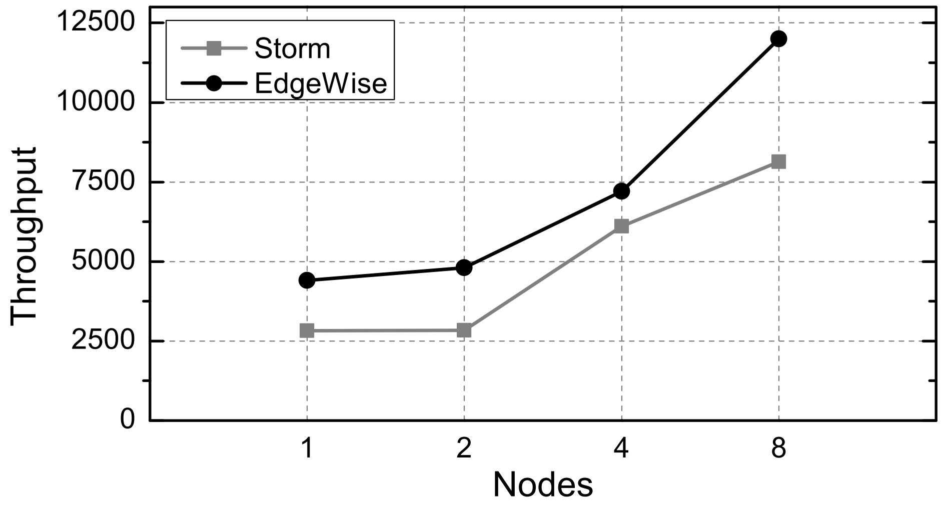
*Maximum end-to-end throughput depends on scheduling heavier operations proportionally more than lighter operations.*

# Data Consumption Policy




Sensitivity study on various consumption policies with STATS topology

# Performance on Distributed Edge




PRED: maximum throughput achieved with the *latency less than 100 ms*

# Limitations

**I/O bound computation:**

- The preferred idiom is *outer I/O, inner compute*
- Worker pool size could be tuned
- I/O could be done elsewhere, like Microsoft Bosque

# Icon Credit

Icon credit for *www.flaticon.com*