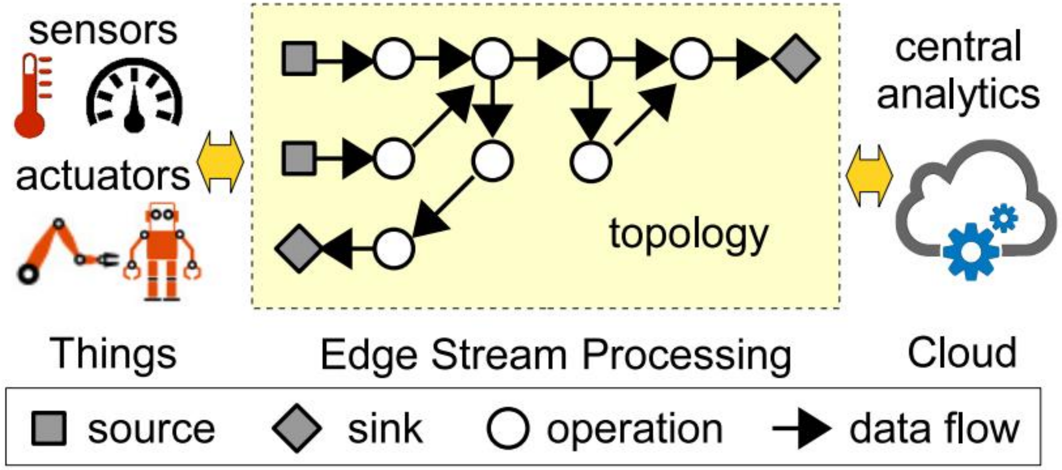
EdgeWise: A Better Stream Processing Engine for the Edge

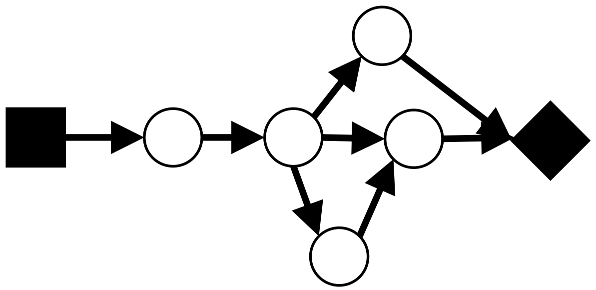
Xinwei Fu, Talha Ghaffar, James C. Davis, and Dongyoon Lee,
Virginia Tech

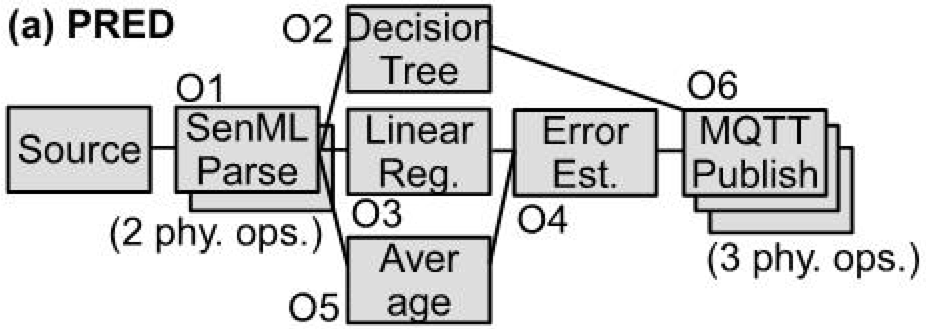
ATC’19 2023/05/03

Background : Stream Processing

- Dataflow Programming Model
  - IoT(Things, Gateways and Cloud)
- Stream Processing Engines
  - deploys the operations onto the compute node



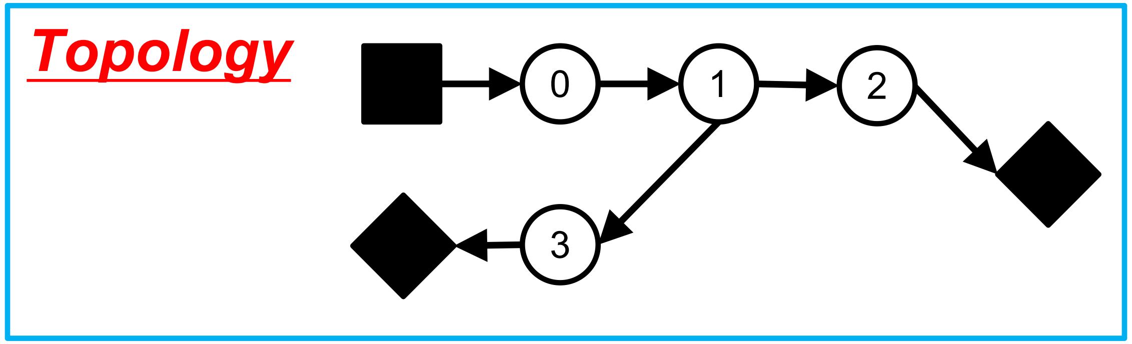


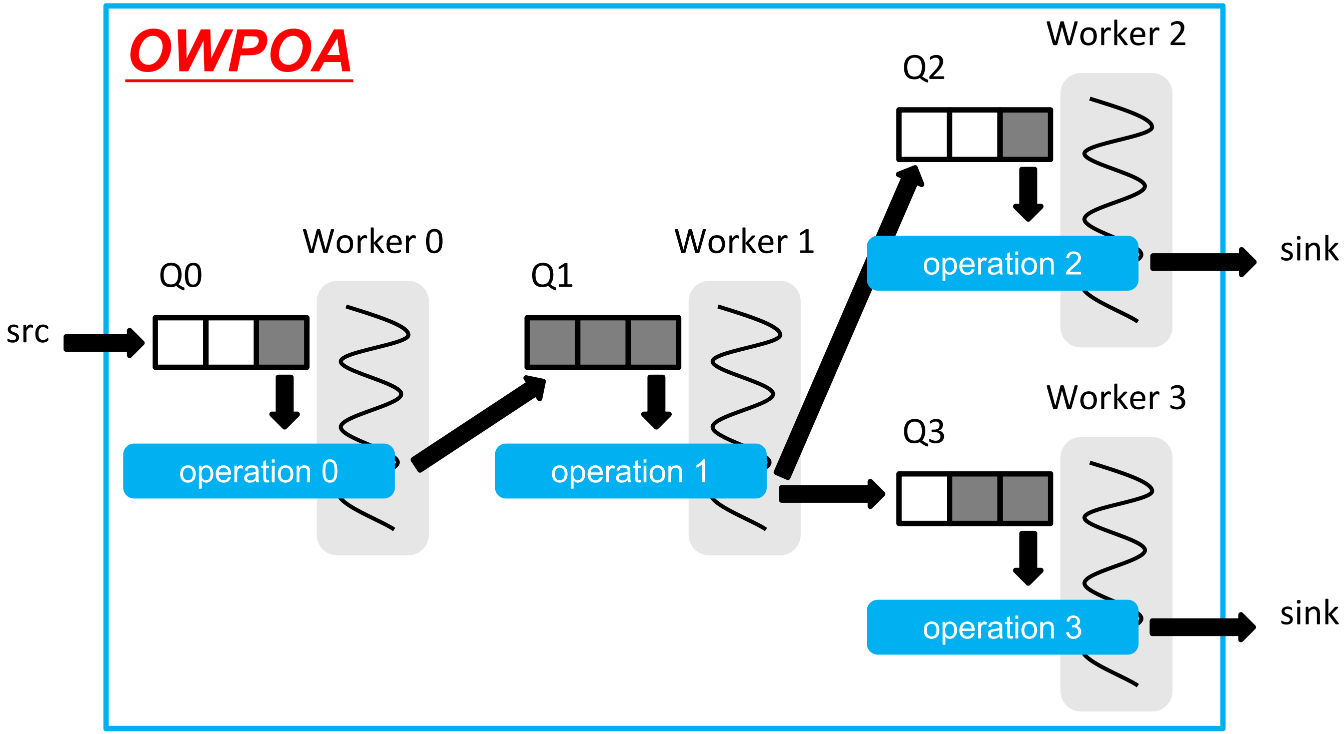


Running on PI

Modern SPEs based on : OWPOA

- One-Worker-per-Operation-Architecture(OWPOA)
  - Queue and Worker thread
  - Pipelined manner
  - Backpressure



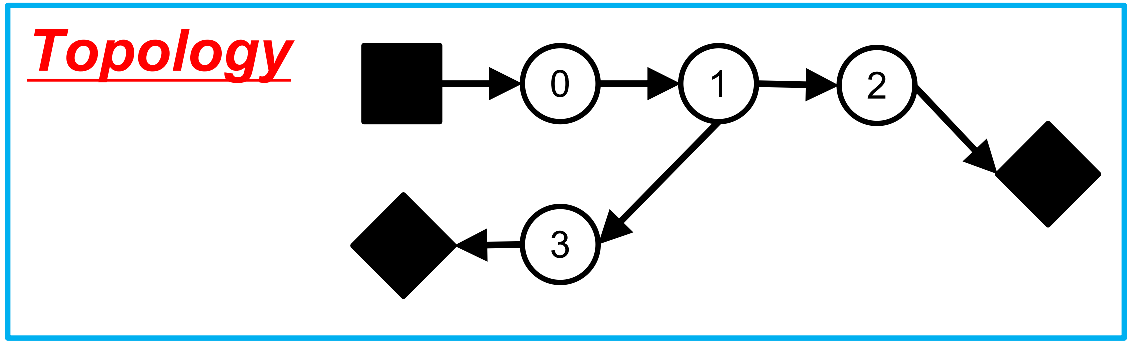


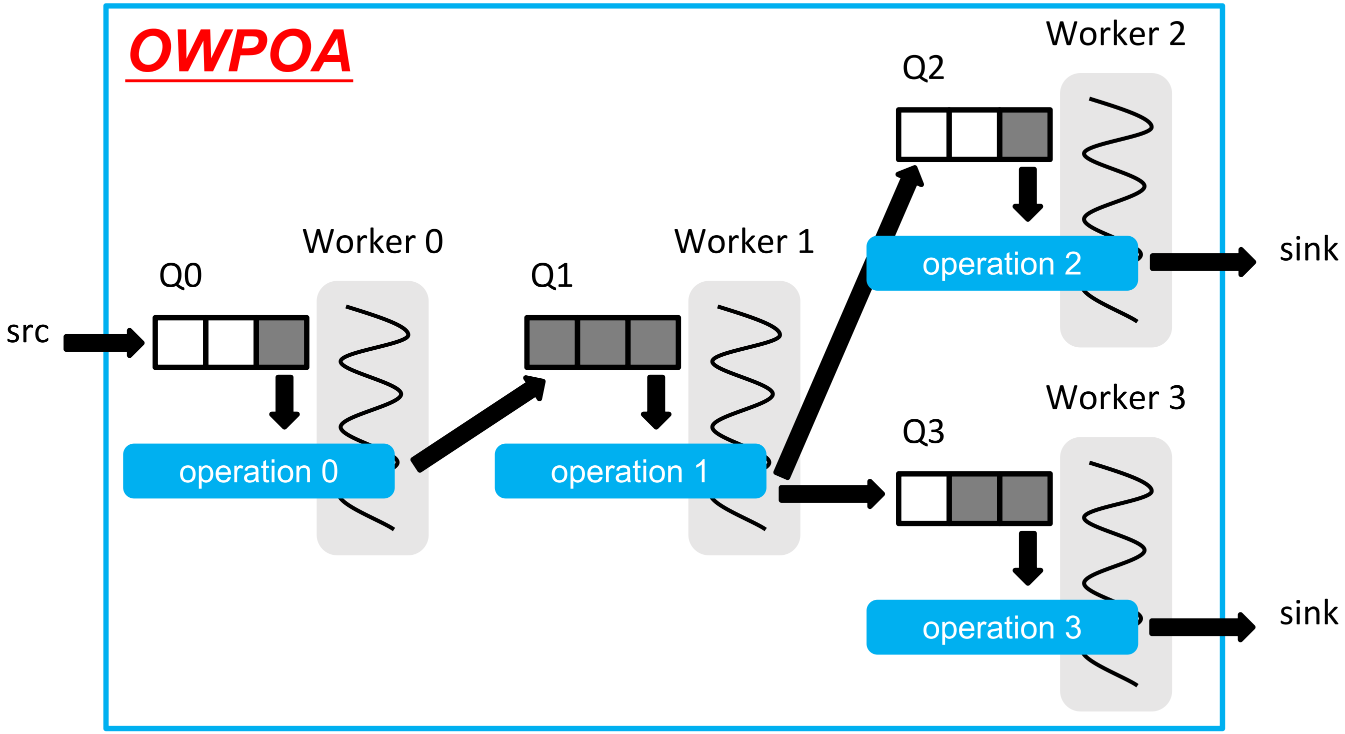
Problem : OWPOA SPEs

- Existing OWPOA Stream Processing Engines are not suitable for the Edge Setting
- OWPOA SPEs
  - Cloud-class resources
  - OS scheduler
- Edge SPEs
  - Limited resources
  - workers >> CPU cores
  - Inefficiency in OS scheduler
- database community study
  - Min-Latency, VLDB’03, Carney et al
  - Min-Memory, VLDB’04, Babcock et al

Main idea & Challenges

- Main idea
  - a new scheduling algorithm supported by a new queuing-theoretic analysis
- Challenges
  - × Multiplexed
  - × High Throughput
  - × Low Latency
  - × No Backpressure
  - × Scalable





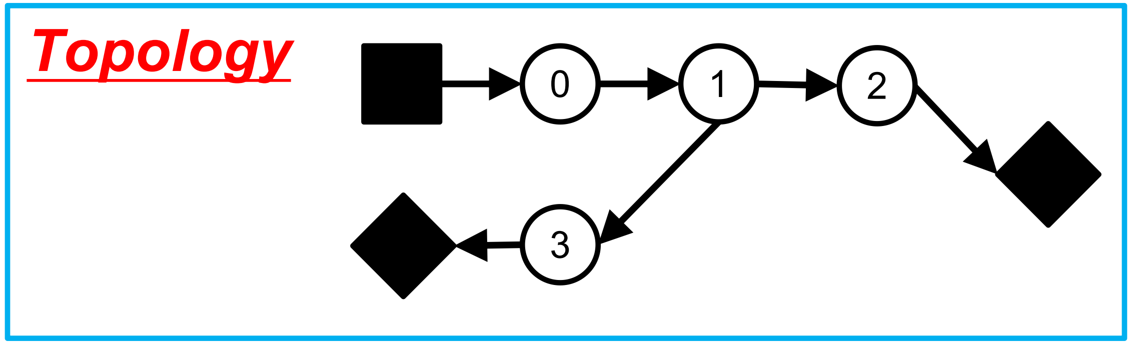
EDGEWISE DESIGN

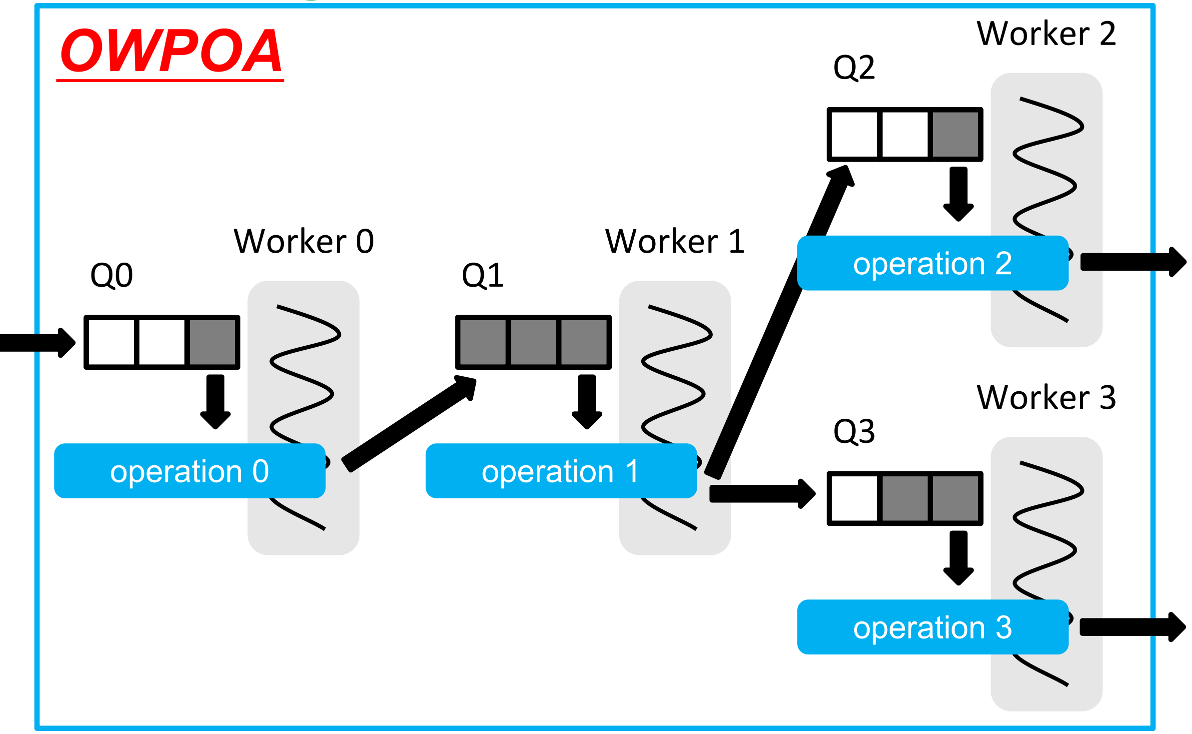
- Limited resources
- workers >> CPU cores

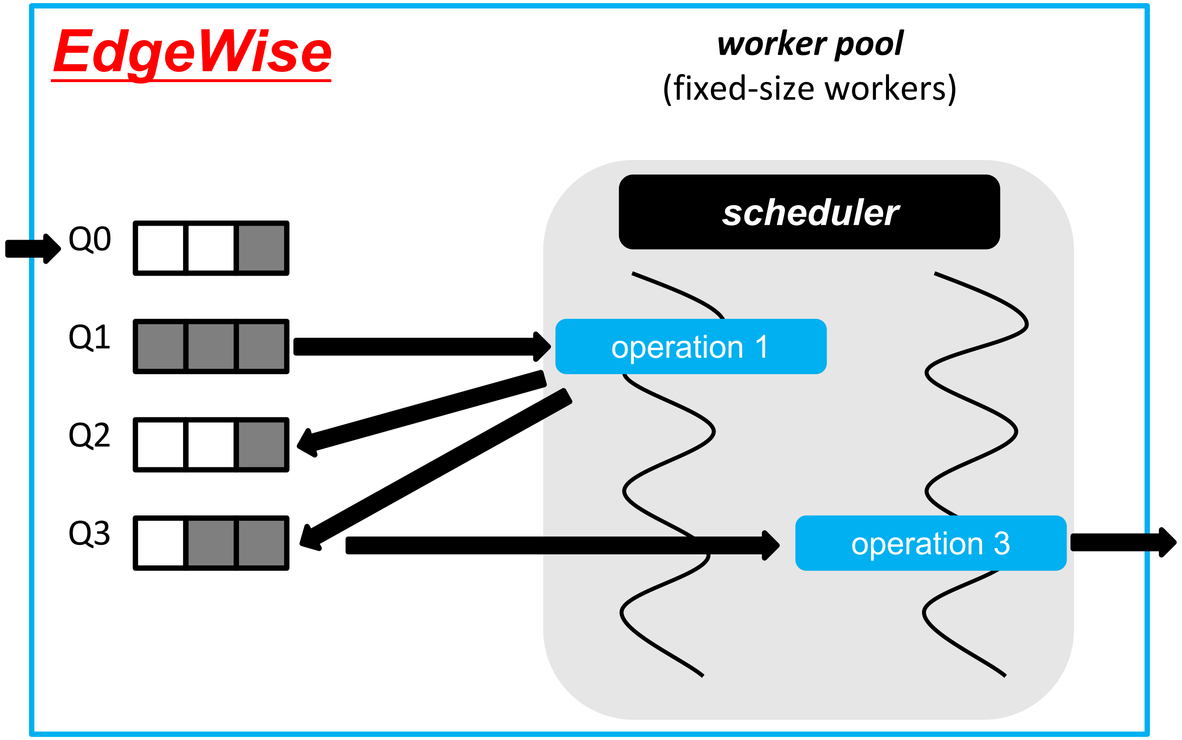
→ A fixed-sized worker pool

- Inefficiency in OS scheduler

→ Engine-level scheduler



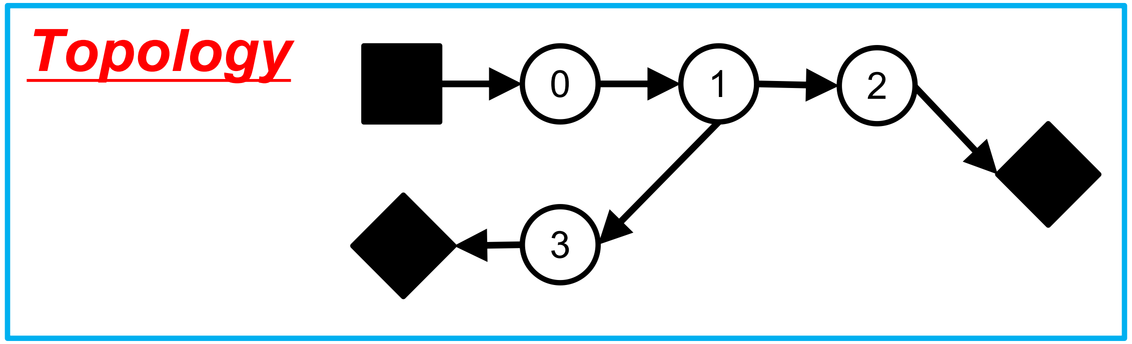




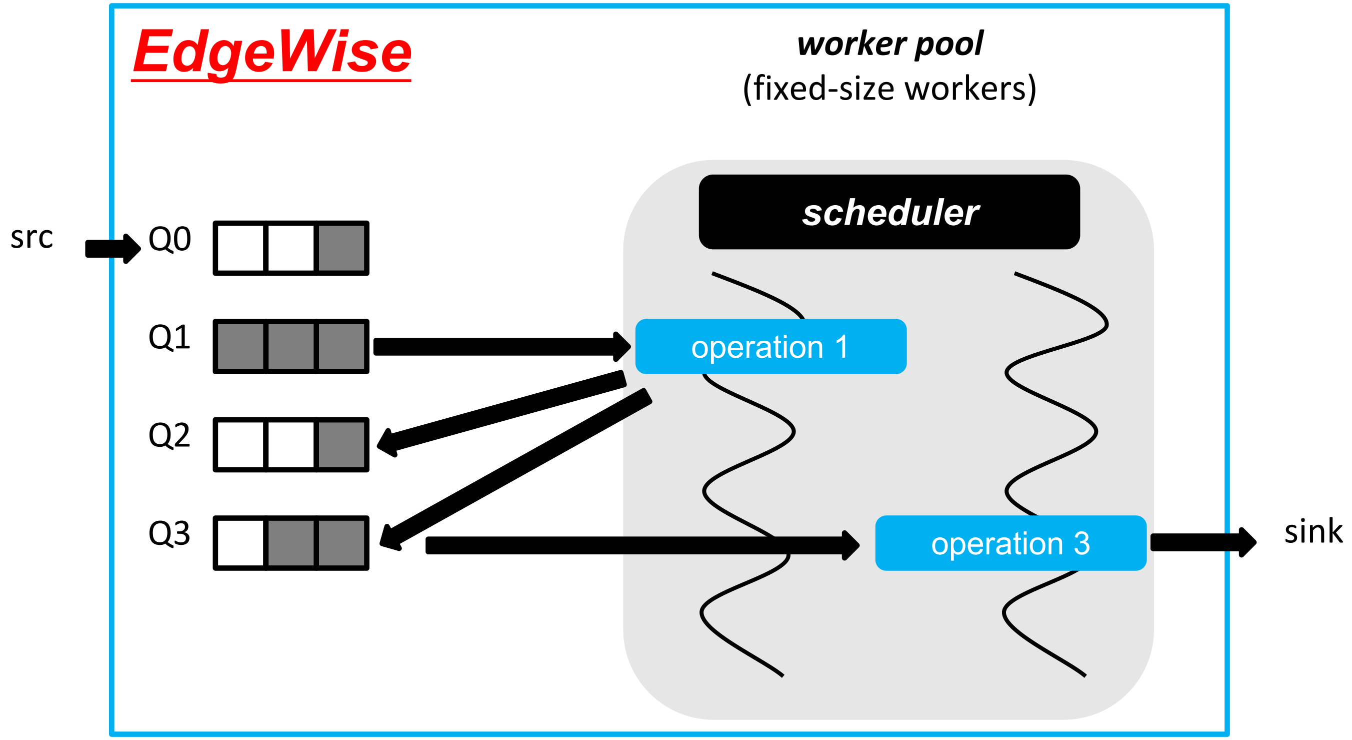
Design : Fixed-size Worker Pool

- Limited resources
- workers >> CPU cores

→ A fixed-sized worker pool



- A fixed set of workers in the worker pool
- The scheduler dynamically chooses which operation a worker should perform

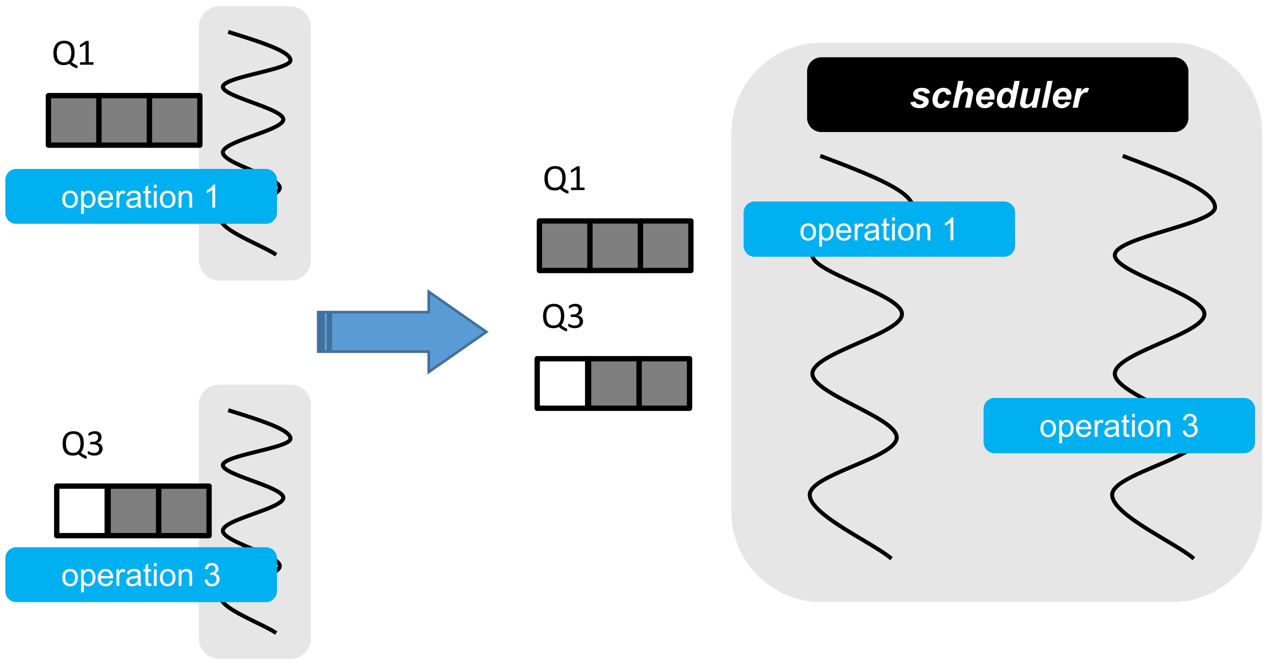


√ Multiplexed

Design : Congestion-Aware Scheduler

- Inefficiency in OS scheduler

→ Engine-level scheduler

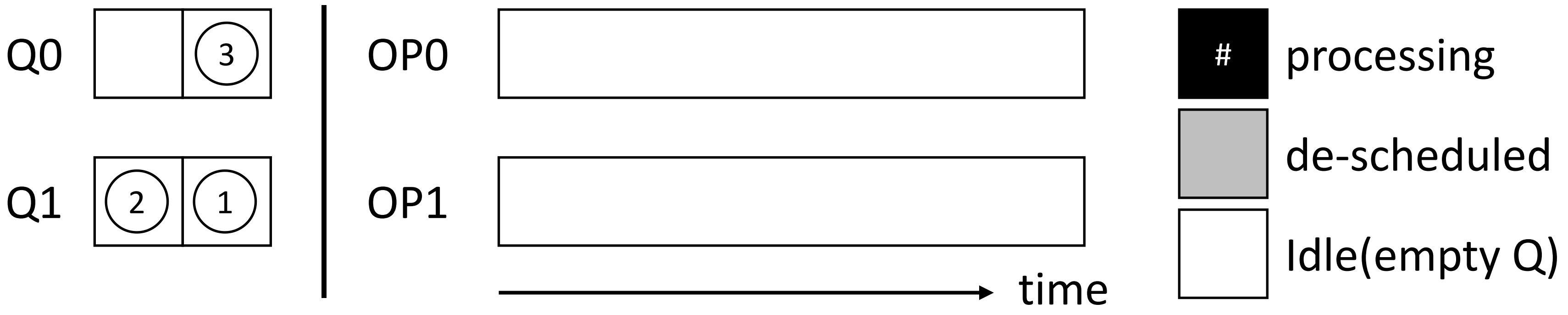


Congestion-Aware Scheduler

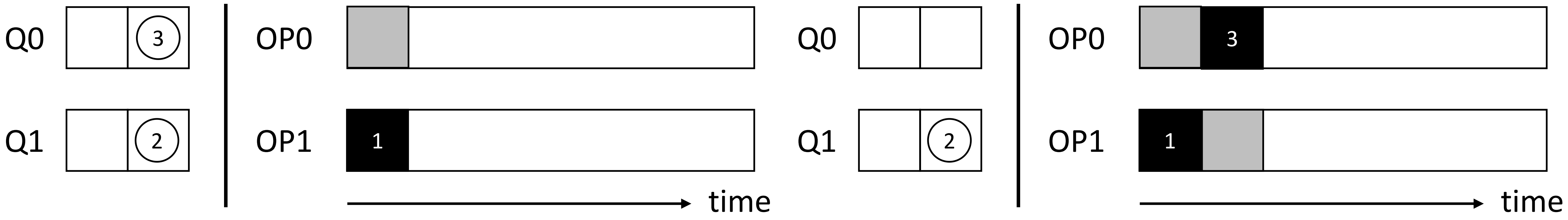
- Profiling-free dynamic solution
- Balance queue sizes
- Choose the OP with the most pending data

Design : Congestion-Aware Scheduler

- The random scheduler of the OWPOA may make the unwise choice

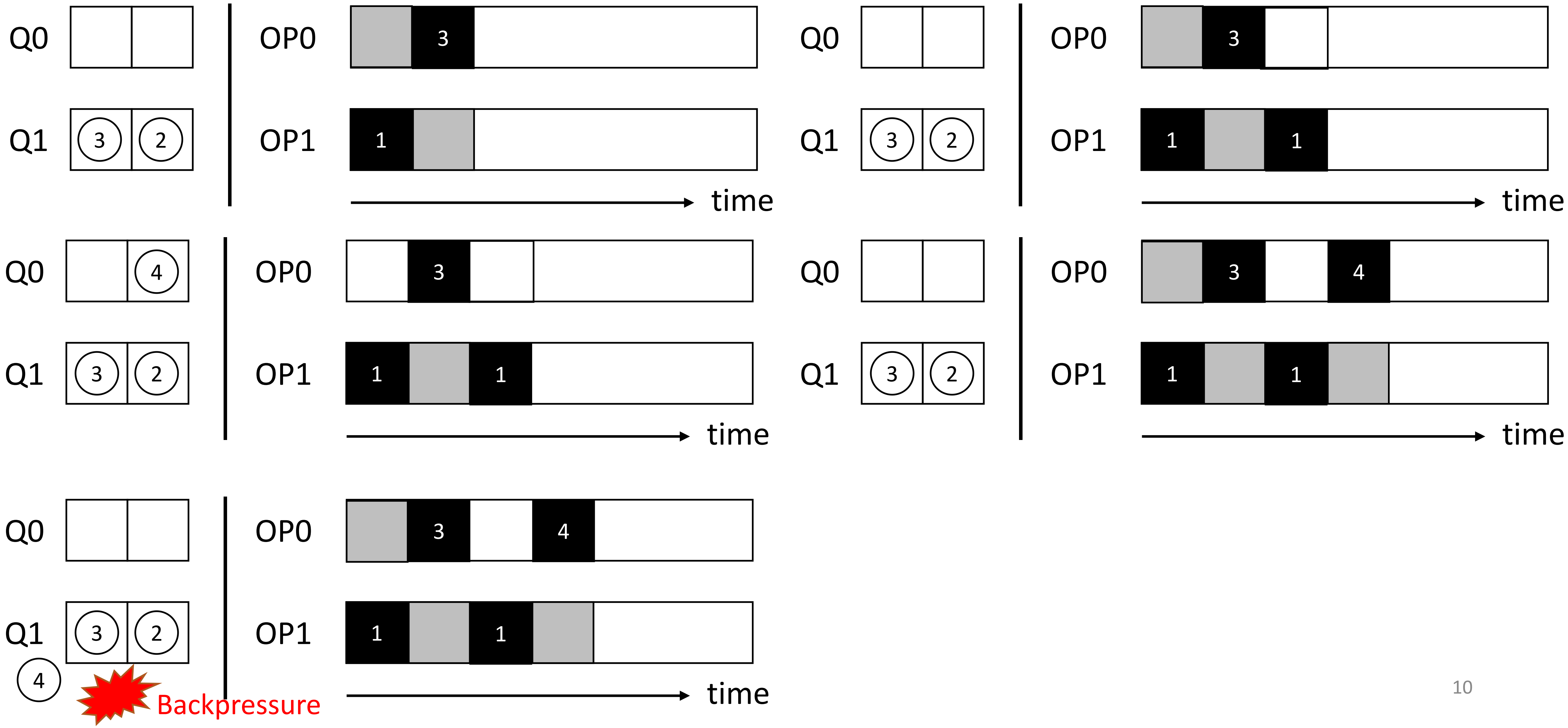


OWPOA – Random OS Scheduler



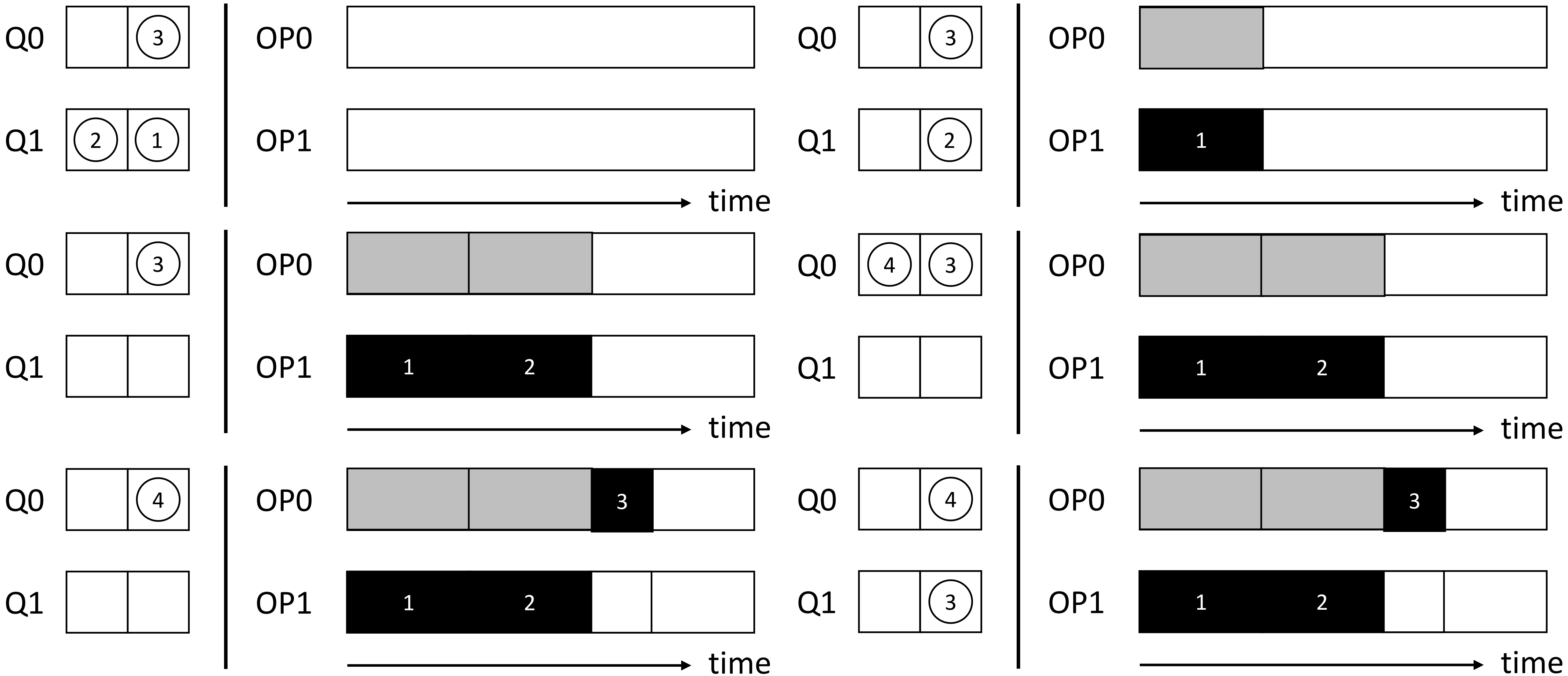
Design : Congestion-Aware Scheduler

OWPOA – Random OS Scheduler



Design : Congestion-Aware Scheduler

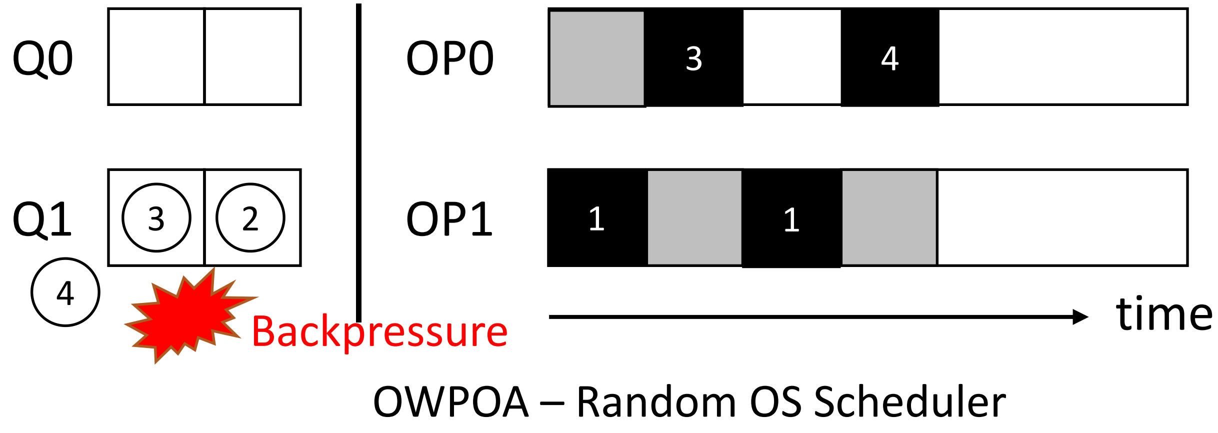
EdgeWise – Congestion-Aware Scheduler



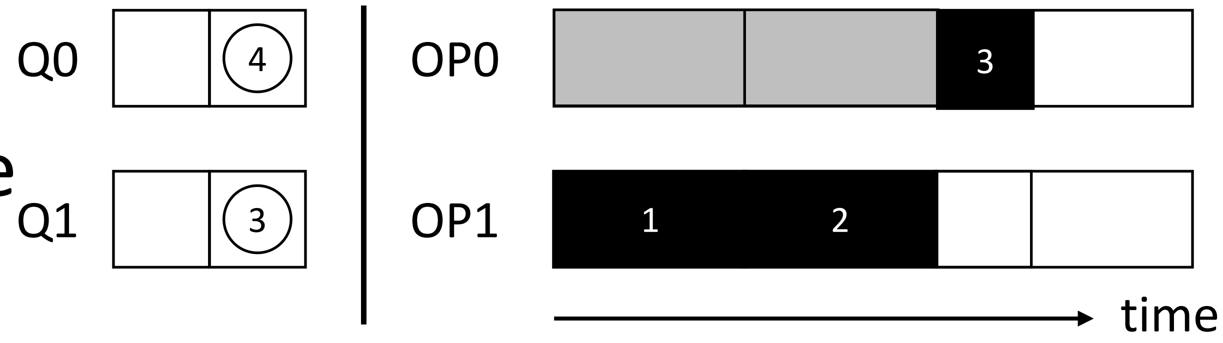
√ No Backpressure

Design : Congestion-Aware Scheduler

- Inefficiency in OS scheduler
→ Engine-level scheduler



- The random scheduler of the OWPOA may lead to backpressure (high latency)
- EDGEWISE evens out the queue lengths to avoid backpressure



√ Low Latency
√ No Backpressure

Performance Analysis : Higher Throughput

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

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

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

√ High Throughput

==>>

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

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

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

scheduling weight -> input rate / service rate

Performance Analysis : Lower Latency

The balancing queue lengths leads to an overall improvement in latency

End-to-end latency

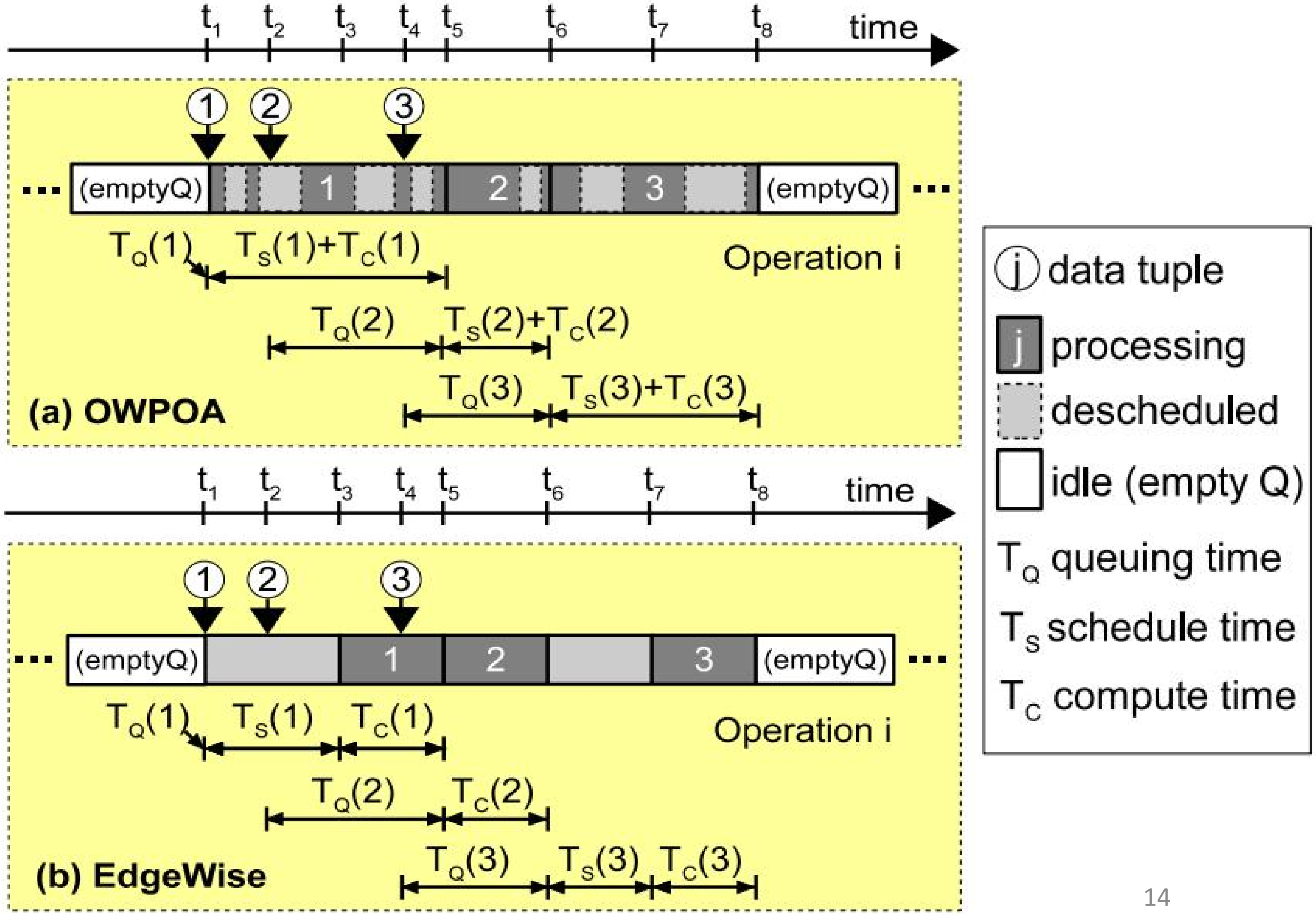
$$Latency = \sum_{i}^{M}(L_i + Comm.) \approx \sum_{i}^{M} L_i$$

Per-operation latency

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

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

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



Performance Analysis : Lower Latency

The balancing queue lengths leads to an overall improvement in latency

End-to-end latency

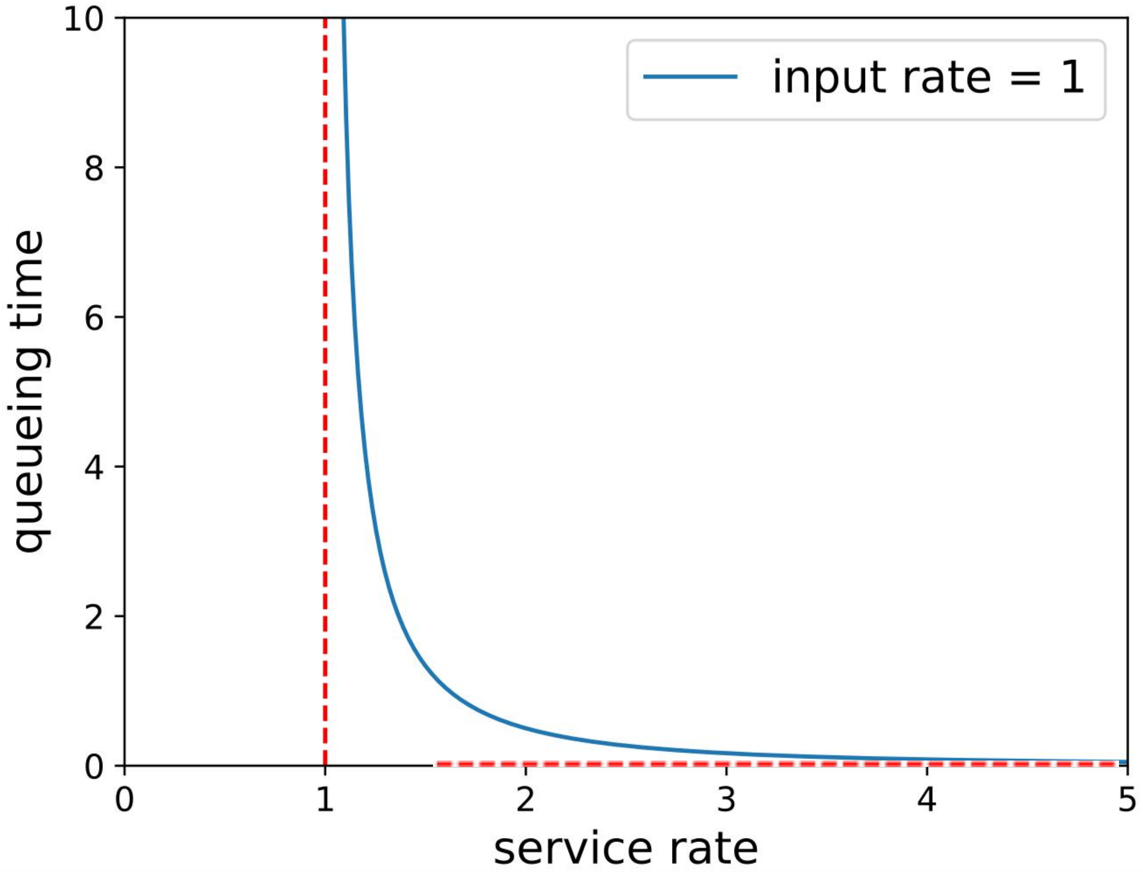
$$Latency = \sum_{i}^{M} (L_i + Comm.) \approx \sum_{i}^{M} L_i$$

Per-operation latency

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

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

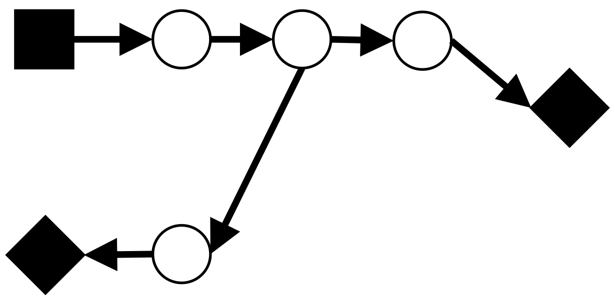
$$T_Q = \frac{\rho}{\mu - \lambda} = \frac{\lambda}{\mu(\mu - \lambda)}$$



Evaluation : Throughput-Latency Performance

- The engine-level schedulers in effect push the backpressure point to a higher input rate, allowing the SPEs to achieve higher throughput at a low latency

<u>Edge Stream Processing</u>



Running on PI

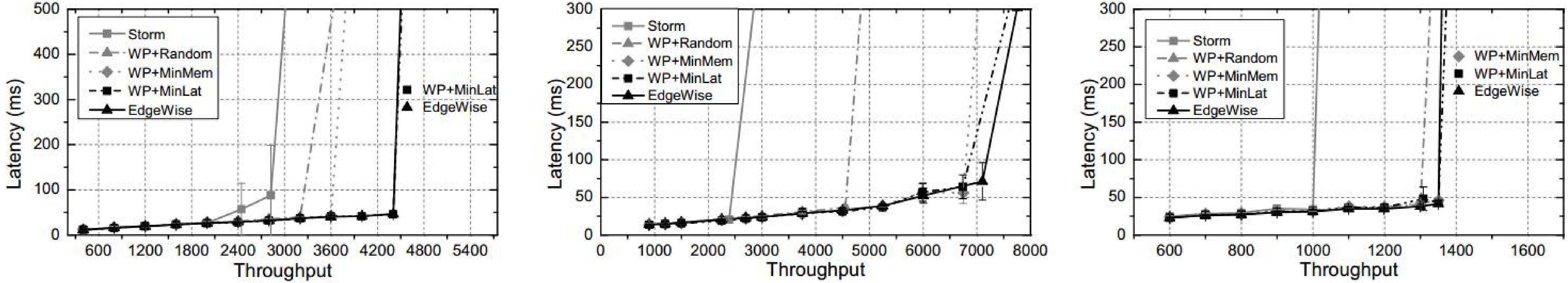


Figure 7: Throughput-latency of (a) PRED, (b) STATS, and (c) ETL topologies.

Evaluation : Detailed Performance Breakdown

Even at the cost of increased queuing times at lighter operations, would obtain an outsized improvement in latency.

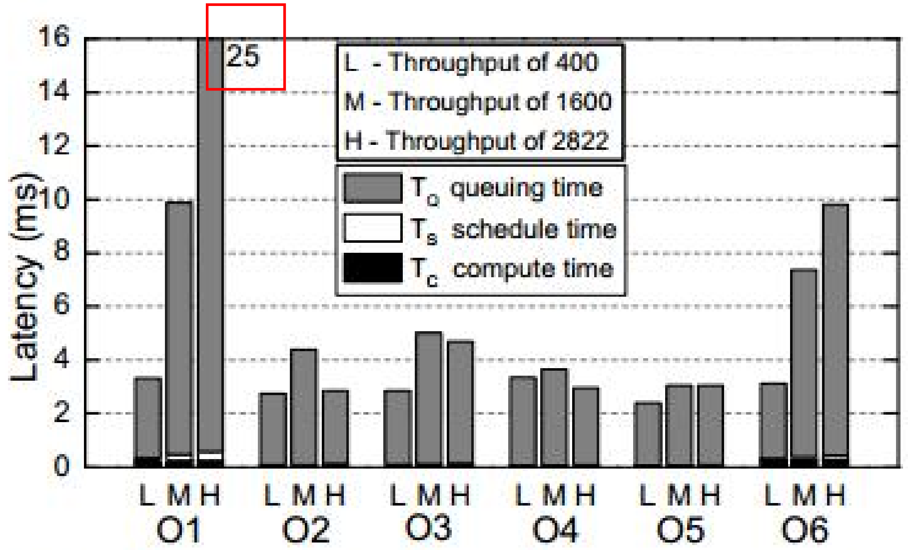


Figure 9: In Storm, as the throughput increases from L(ow) to M(edium) to H(igh), the queuing latency of heavy operations (e.g., O1 and O6) increases rapidly.

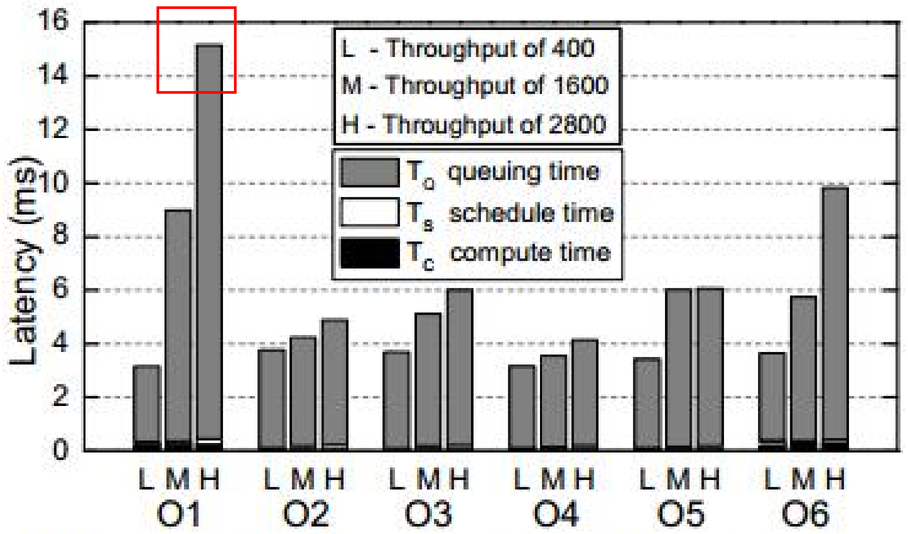


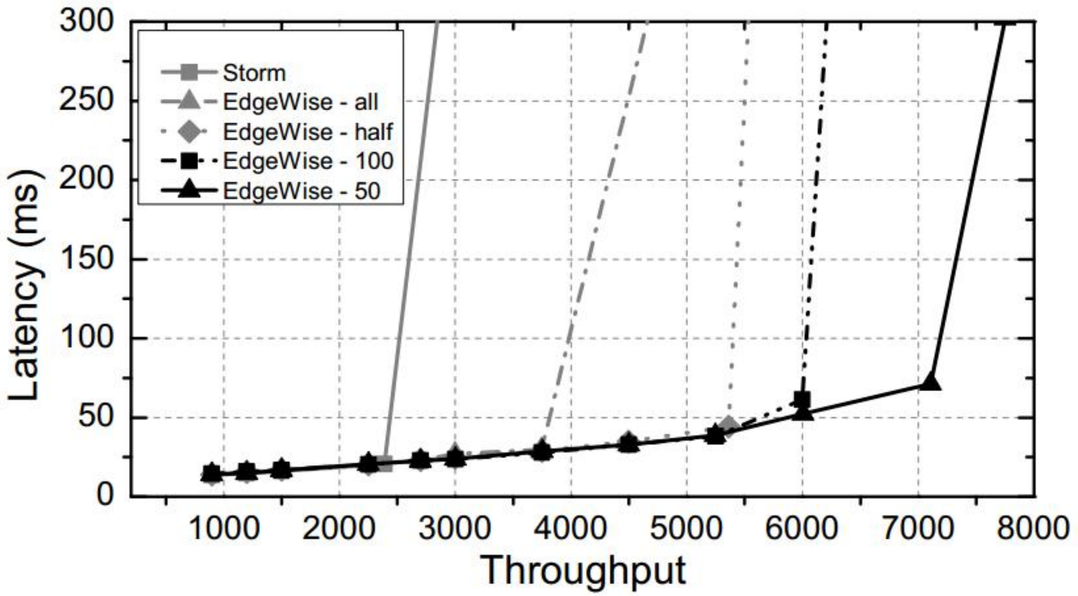
Figure 10: In EDGEWISE, as the throughput increases, the queuing latency of heavy operations (e.g., O1 and O6)

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

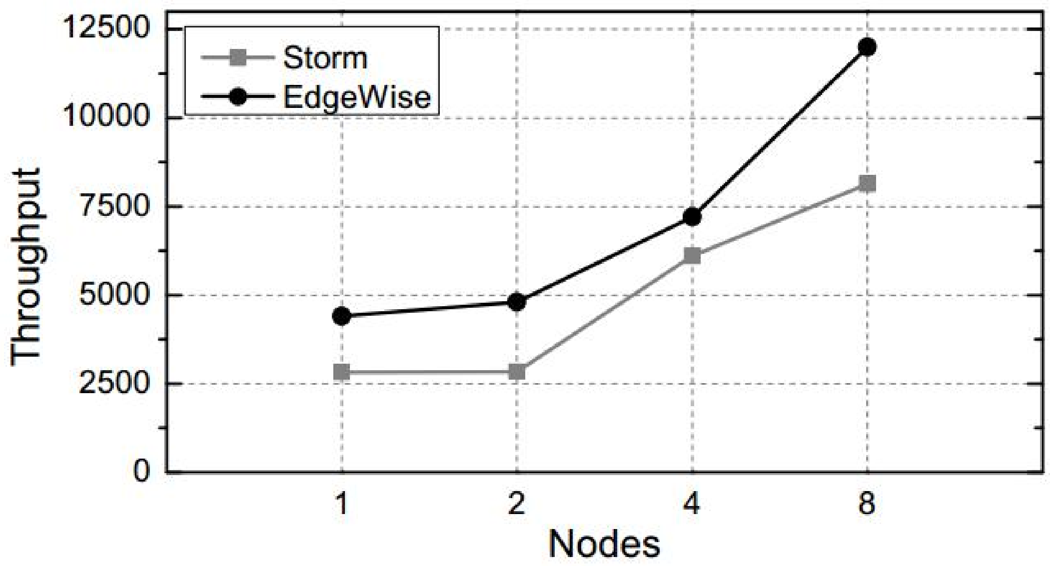
Data Consumption Policy & Performance on Distributed Edge

The constant consumption rules consistently performed well.

At-most-50 data consumption policy



EDGEWISE's intra-node optimizations benefit an internode (distributed) workload.



√ Scalable

Summary

Problem

Existing OWPOA SPEs are not suitable for the Edge Setting

main idea

a new scheduling algorithm supported by a new queuing-theoretic analysis

dessign

fixed-sized worker pool

Engine-level scheduler

About

- Why choose
  - Terminal to edge
- Help
  - Diversity of edge nodes