Sifter: An Inversion-Free and Large-Capacity Programmable Packet Scheduler

Peixuan Gao, Anthony Dalleggio, Jiajin Liu, Chen Peng, Yang Xu, H. Jonathan Chao

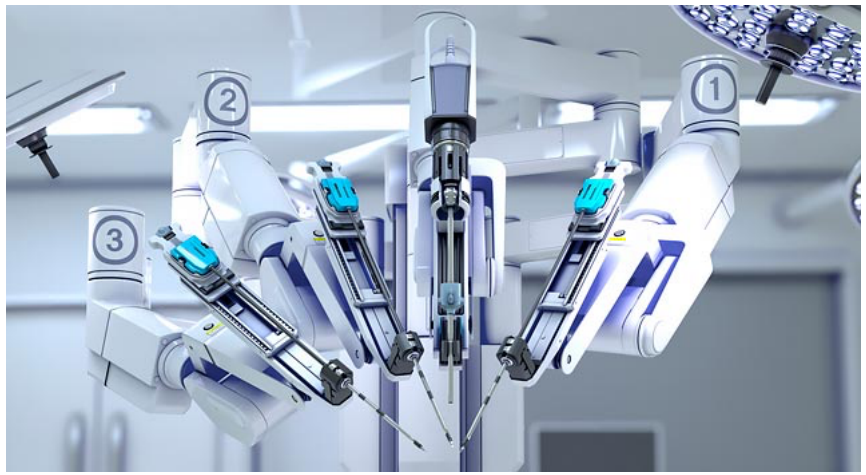
Outline

- Background and Motivation
- Observation and Insights
- Sifter: An Inversion-Free and Large-Capacity Programmable Packet Scheduler
- Evaluation and Implementation
- Conclusion

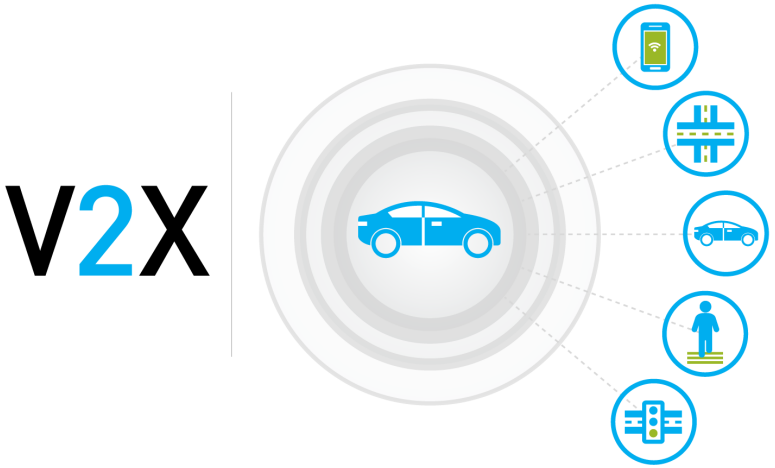
Background and Motivation

Motivation: Latency-Sensitive Application

Latency-sensitive application have hard deadlines for packets



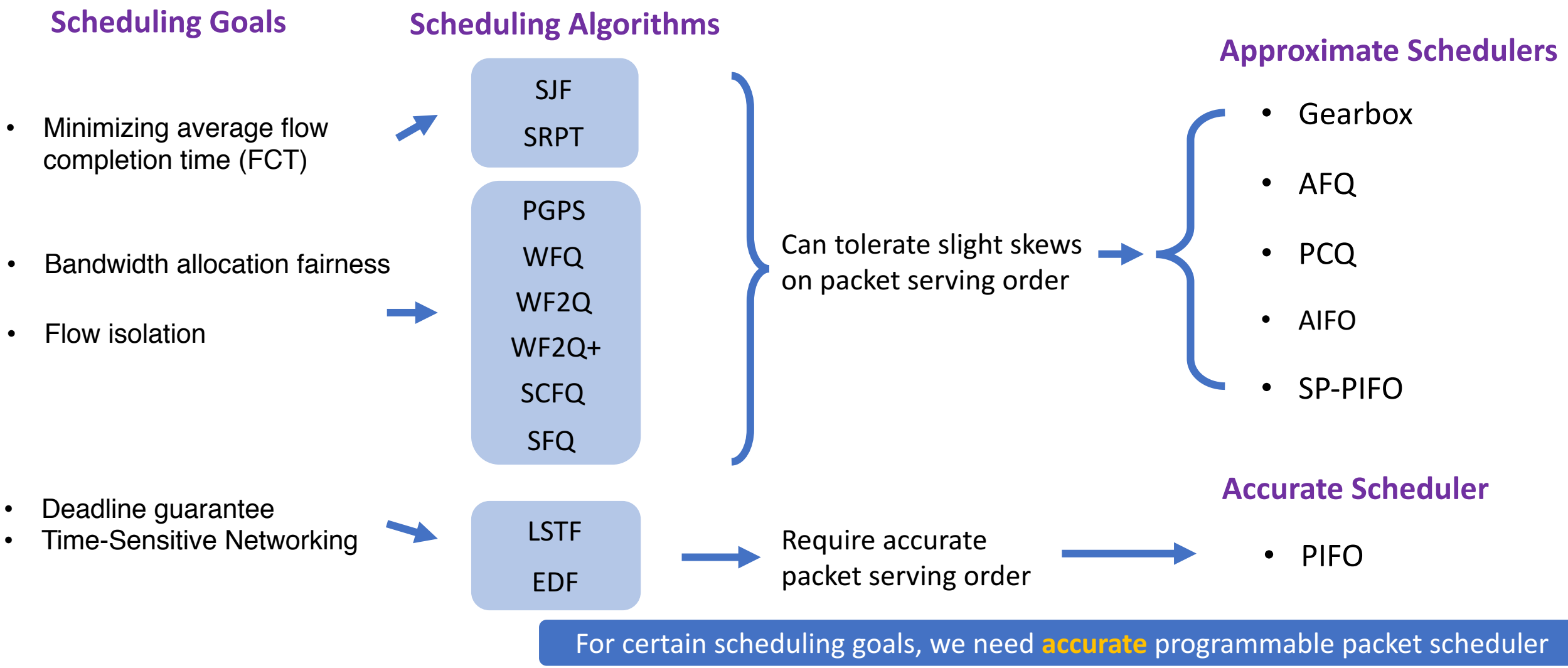
Remote surgery



V2X

These applications rely on accurate schedulers that guarantee the deadline of each packet

Motivation: Latency-Sensitive Application



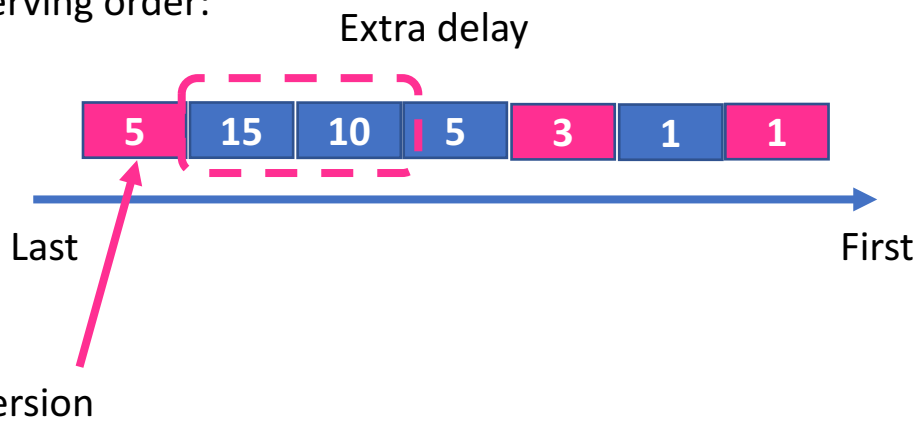
Motivation: Latency-Sensitive Application

Latency-sensitive application have hard deadlines for packets

V2X

Approximate scheduler's serving order:

Red packets: V2X packets
Sensitive to delay



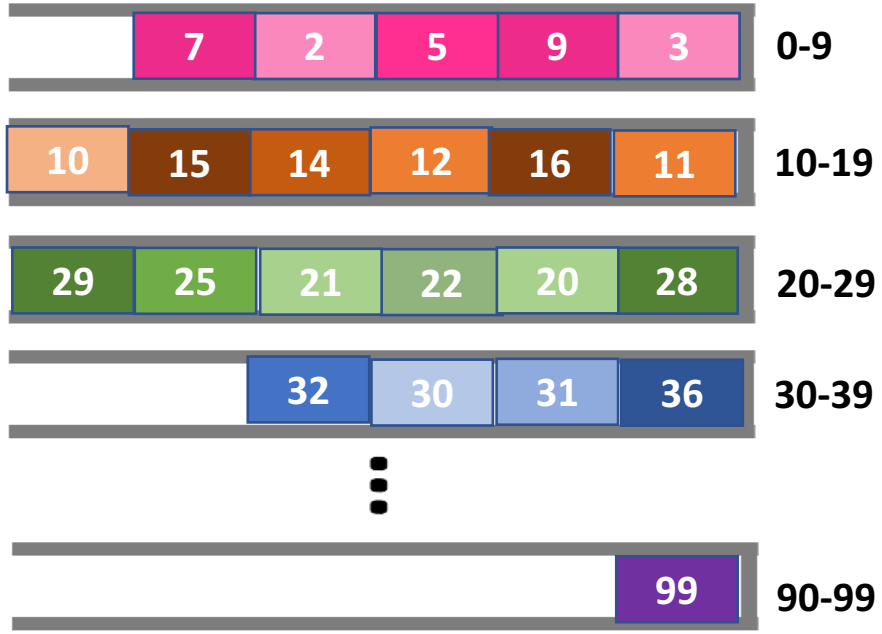
Such extra delay caused by the packet inversion can be **critical** for delay sensitive applications

Observations and Insights

Packet Inversions

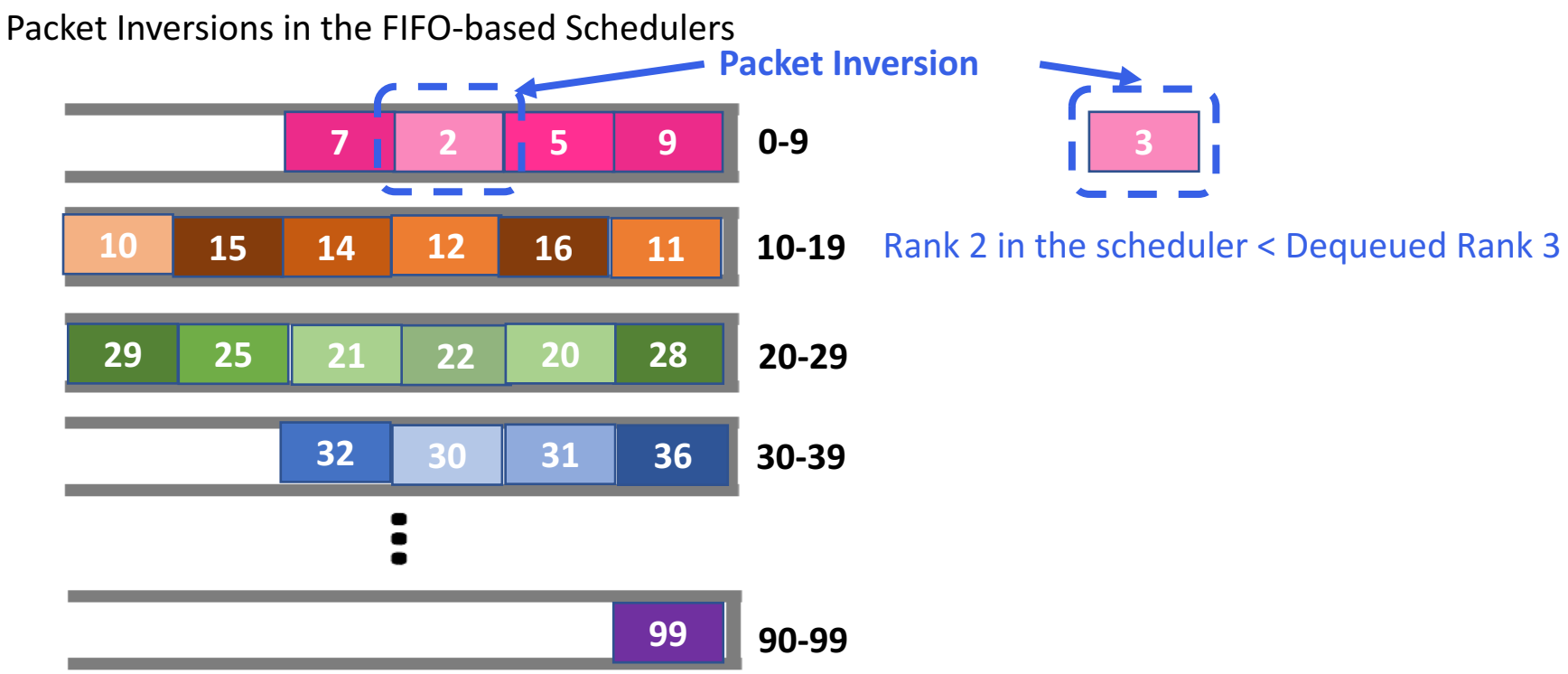
Packet Inversions: When a packet with rank r departs from the scheduler, there exists packet(s) with a smaller rank r′ (where smaller rank has higher priority, r′ < r) in the scheduler.

Packet Inversions in the FIFO-based Schedulers



Packet Inversions

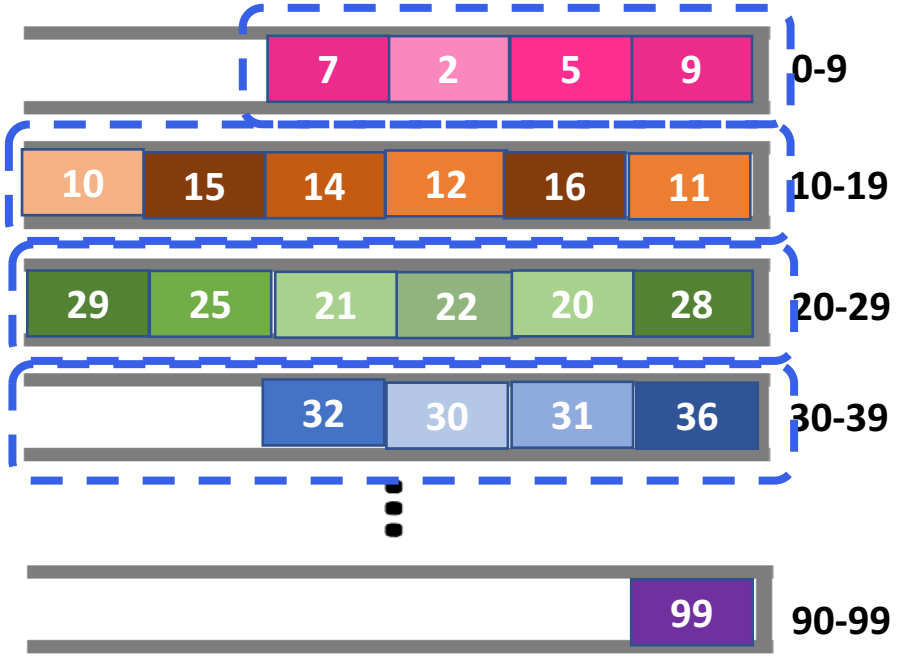
Packet Inversions: When a packet with rank r departs from the scheduler, there exists packet(s) with a smaller rank r′ (where smaller rank has higher priority, r′ < r) in the scheduler.



Packet Inversions

Packet Inversions: When a packet with rank r departs from the scheduler, there exists packet(s) with a smaller rank r′ (where smaller rank has higher priority, r′ < r) in the scheduler.

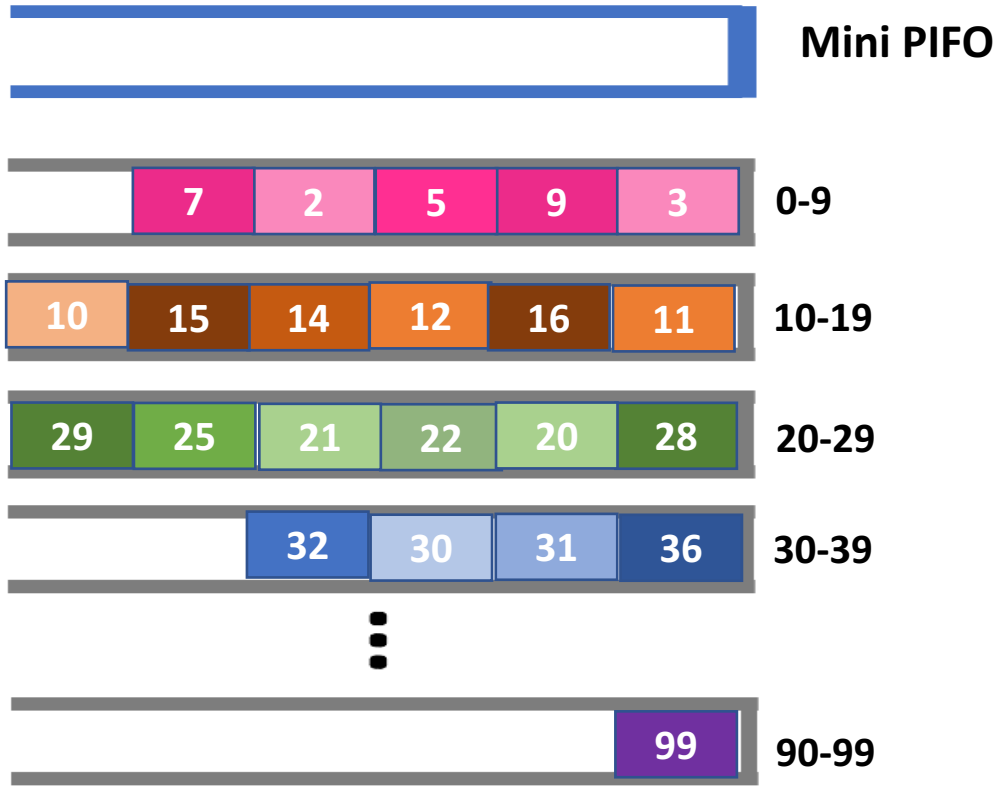
Packet Inversions in the FIFO-based Schedulers



FIFO-based schedulers are subject to packet inversion within each queue

Eliminating Packet Inversions

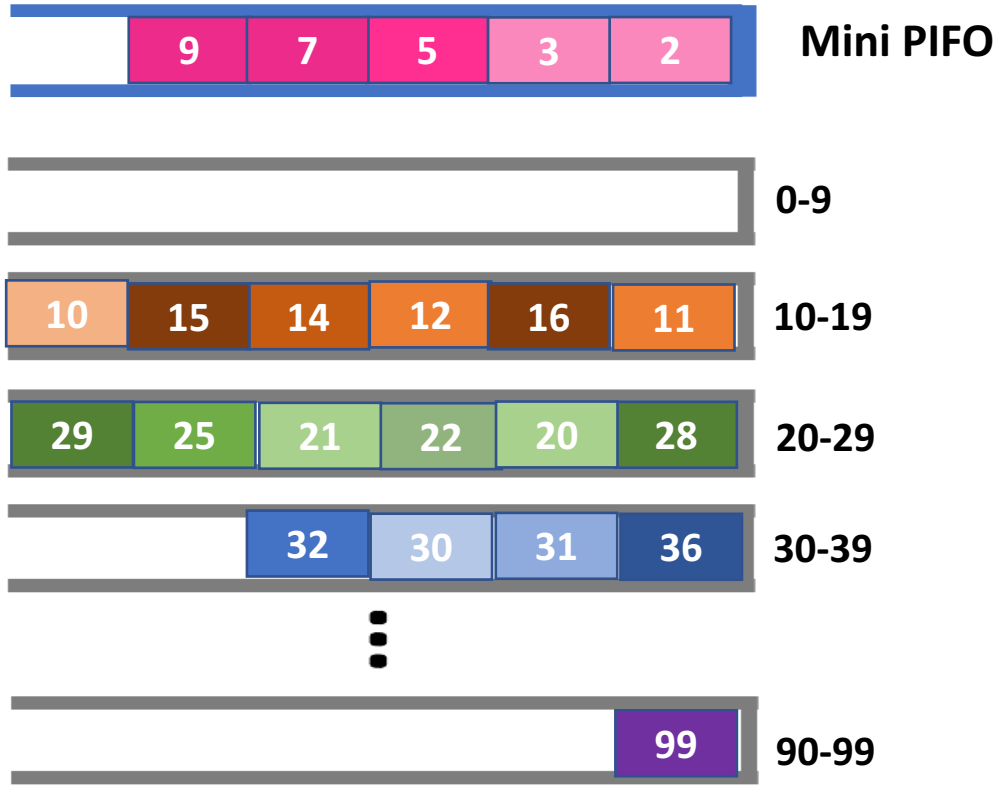
We can eliminate packet inversion if we can sort packets within each queue



Eliminating Packet Inversions

We can eliminate packet inversion if we can sort packets within each queue

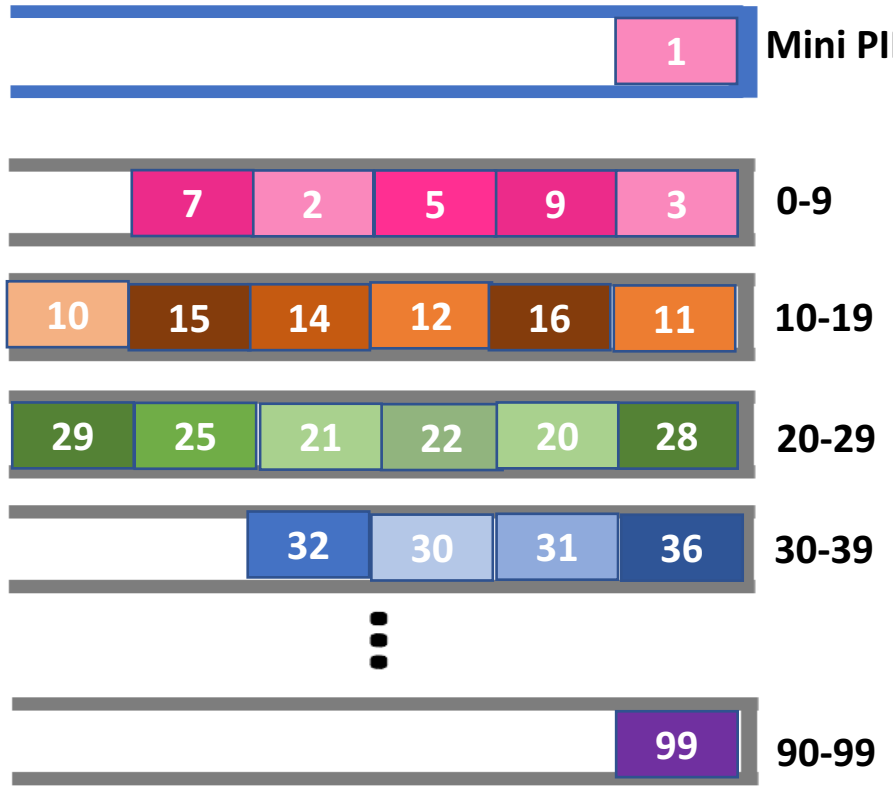
But sorting all the packets in a queue takes time



Speed-up factor

Speed-up Factor: The number of packet metadata that can be processed during the time that transmit a packet

Assume we know the packet with the smallest rank that is ready to transmit



Time to move a metadata:

$$Metadata\ size\ /\ memory\ rate = 8Bytes\ /(2GHz*32bit)$$

Time to transmit packet **1**:

$$Packet\ size\ /\ line\ rate = 128\ Bytes\ /\ 100Gbps$$

We can sort the packets in the earliest FIFO while we are transmitting the packet with the smallest rank

$$Speedup\ Factor = \frac{Time\ to\ transmit\ a\ packet}{Time\ to\ move\ a\ meta\ data}$$

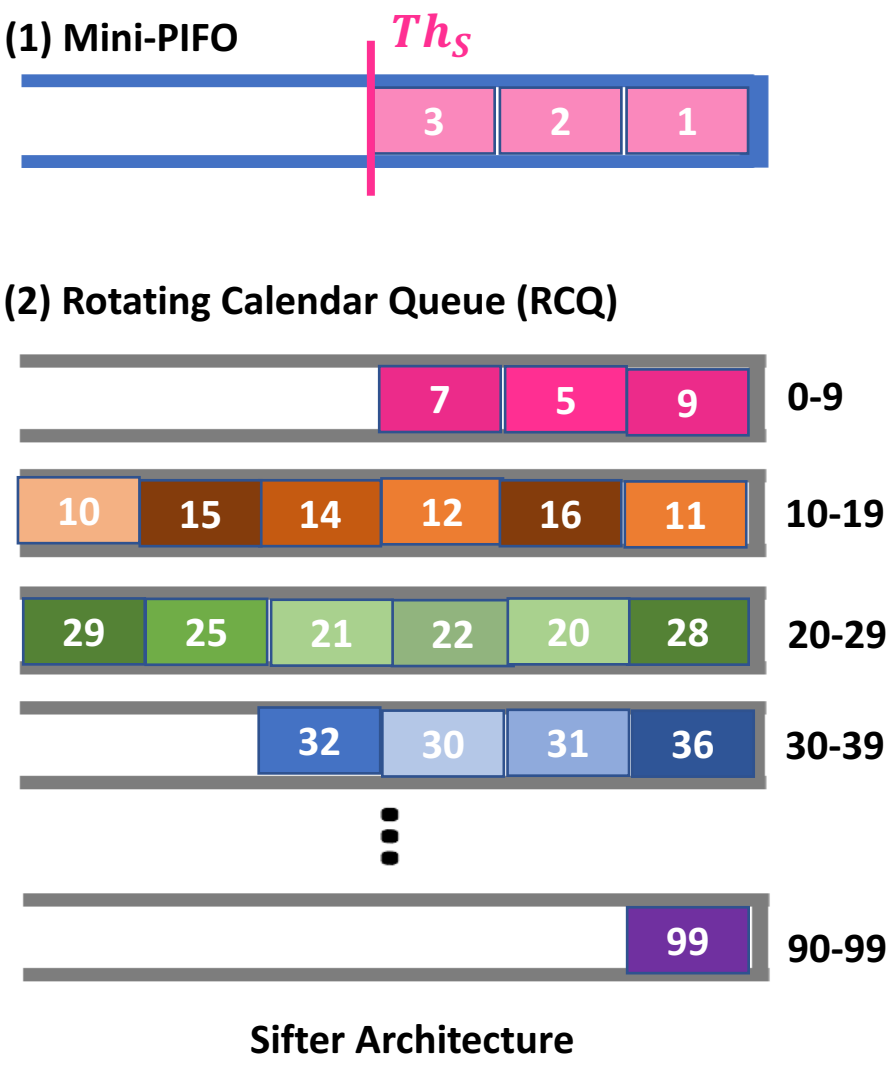
$$= \frac{128\ Bytes\ /\ 100Gbps}{8\ Bytes\ /\ 64Gbps}$$

$$= 10.24$$

This means we can process 10 metadata during the time interval of transmitting each packet

Sifter: An Inversion-Free and Large-Capacity Programmable Packet Scheduler

Sifter Architecture



Sifter Architecture

Mini-PIFO: A small PIFO stores the packets with the smallest ranks in the scheduler

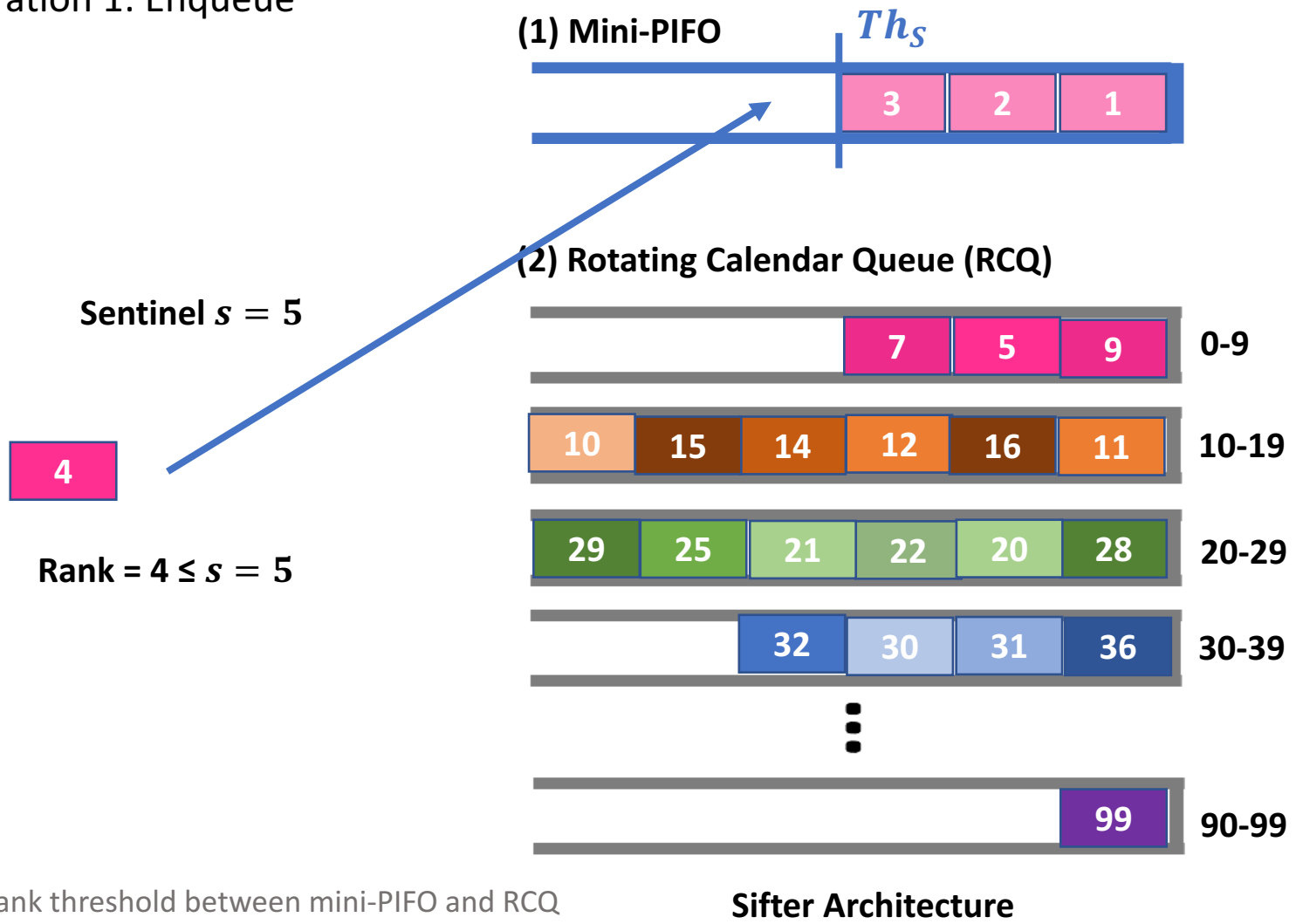
Use a mini-PIFO that **ALWAYS** store the packets with the smallest ranks in the scheduler

Set a 'Sifting Threshold Th_S' to keep the Mini-PIFO occupied by **at least** Th_S metadata

Rotating Calendar Queue (RCQ): A FIFO-based Calendar Queue stores the packets with the larger ranks in the scheduler

Sifter - Enqueue

Sifter Operation 1: Enqueue

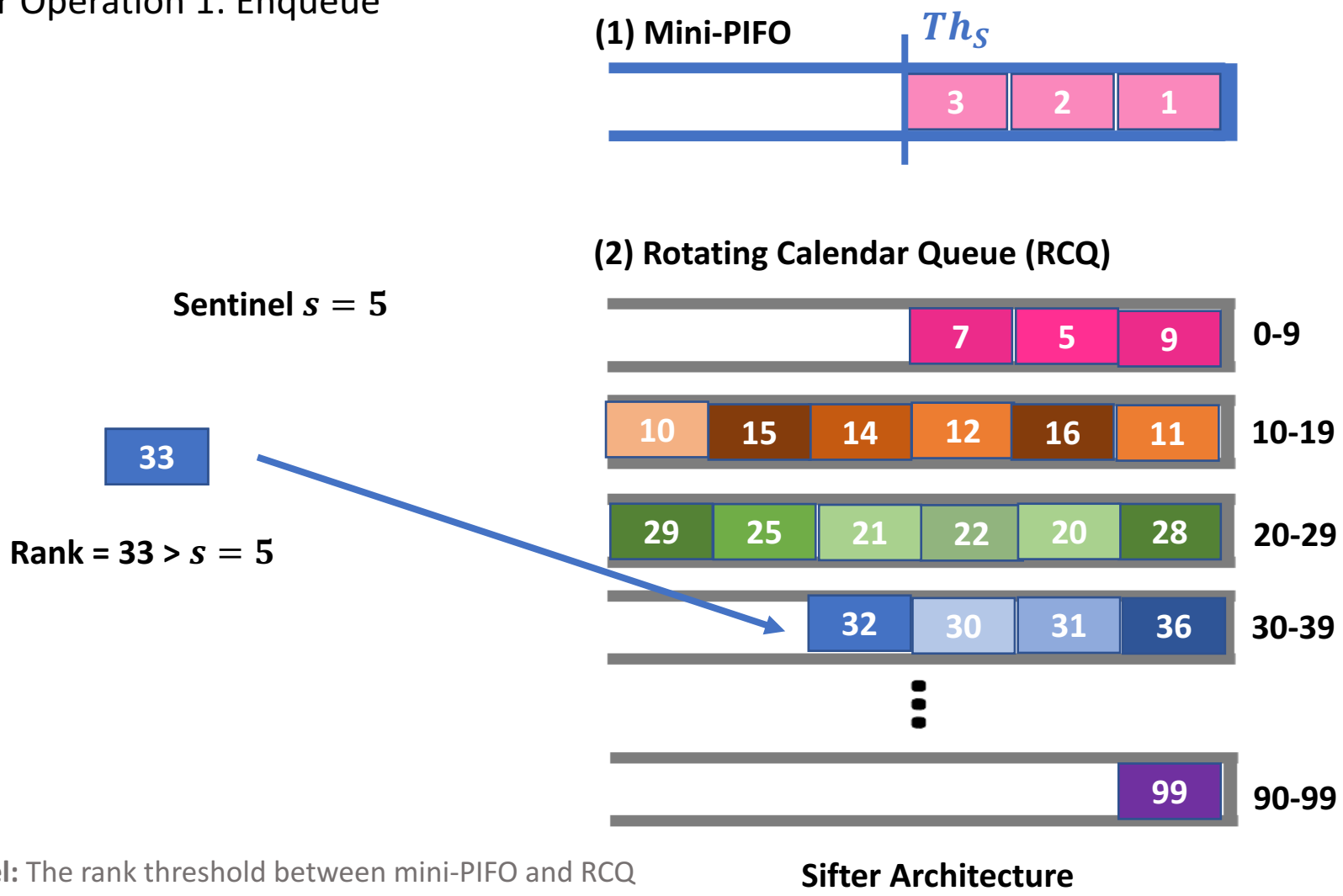


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Enqueue

Sifter Operation 1: Enqueue

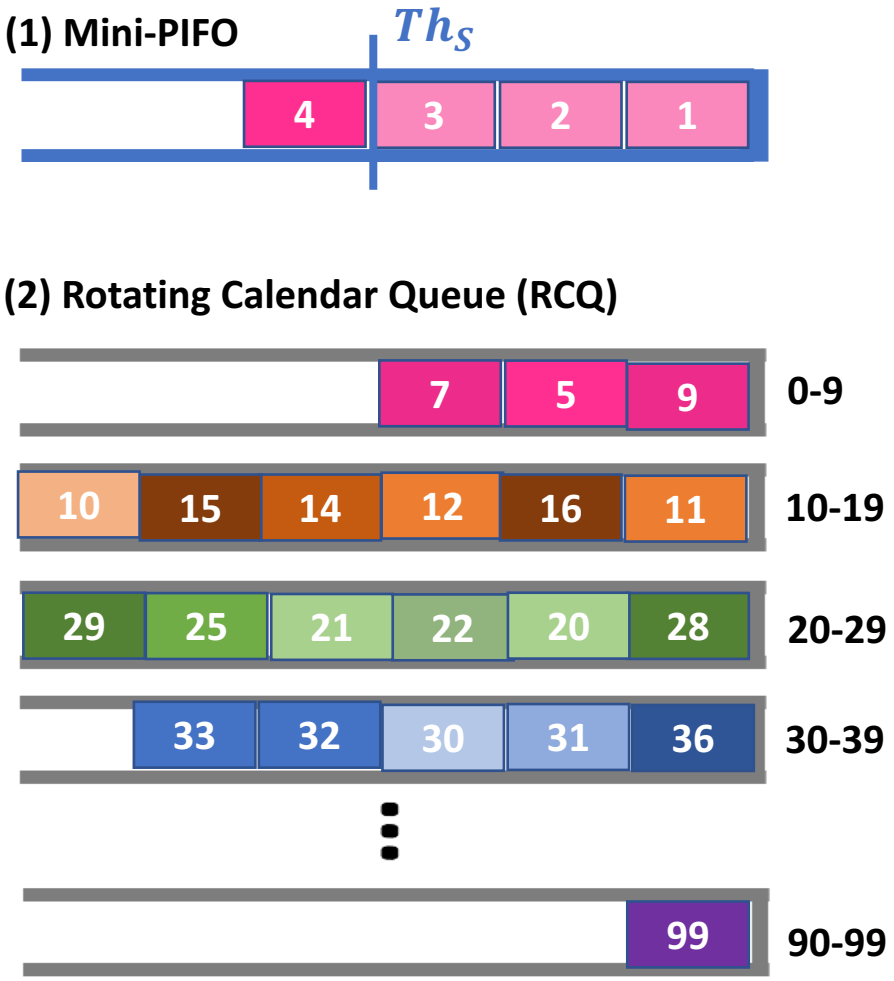


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Dequeue

Sifter Operation 2: Dequeue

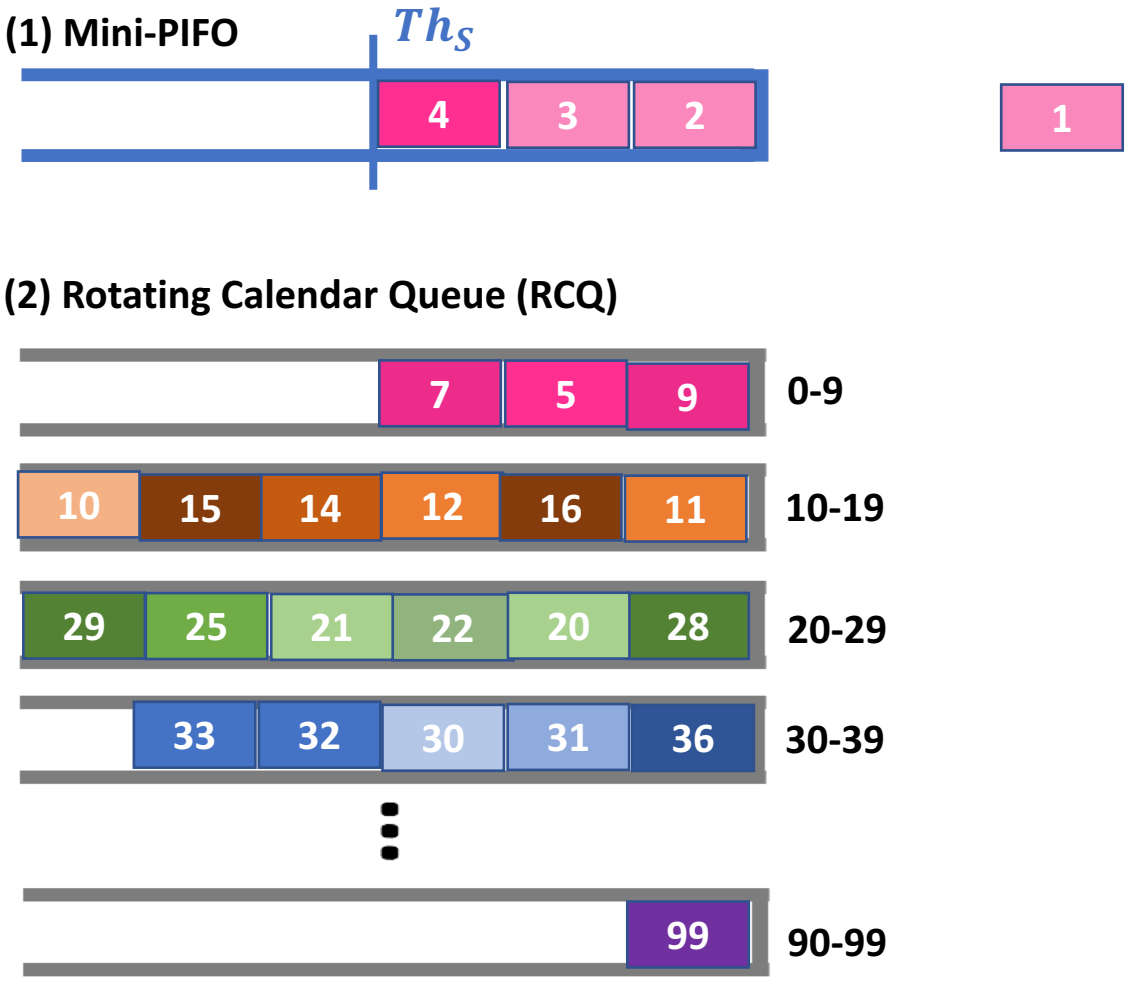


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Dequeue

Sifter Operation 2: Dequeue

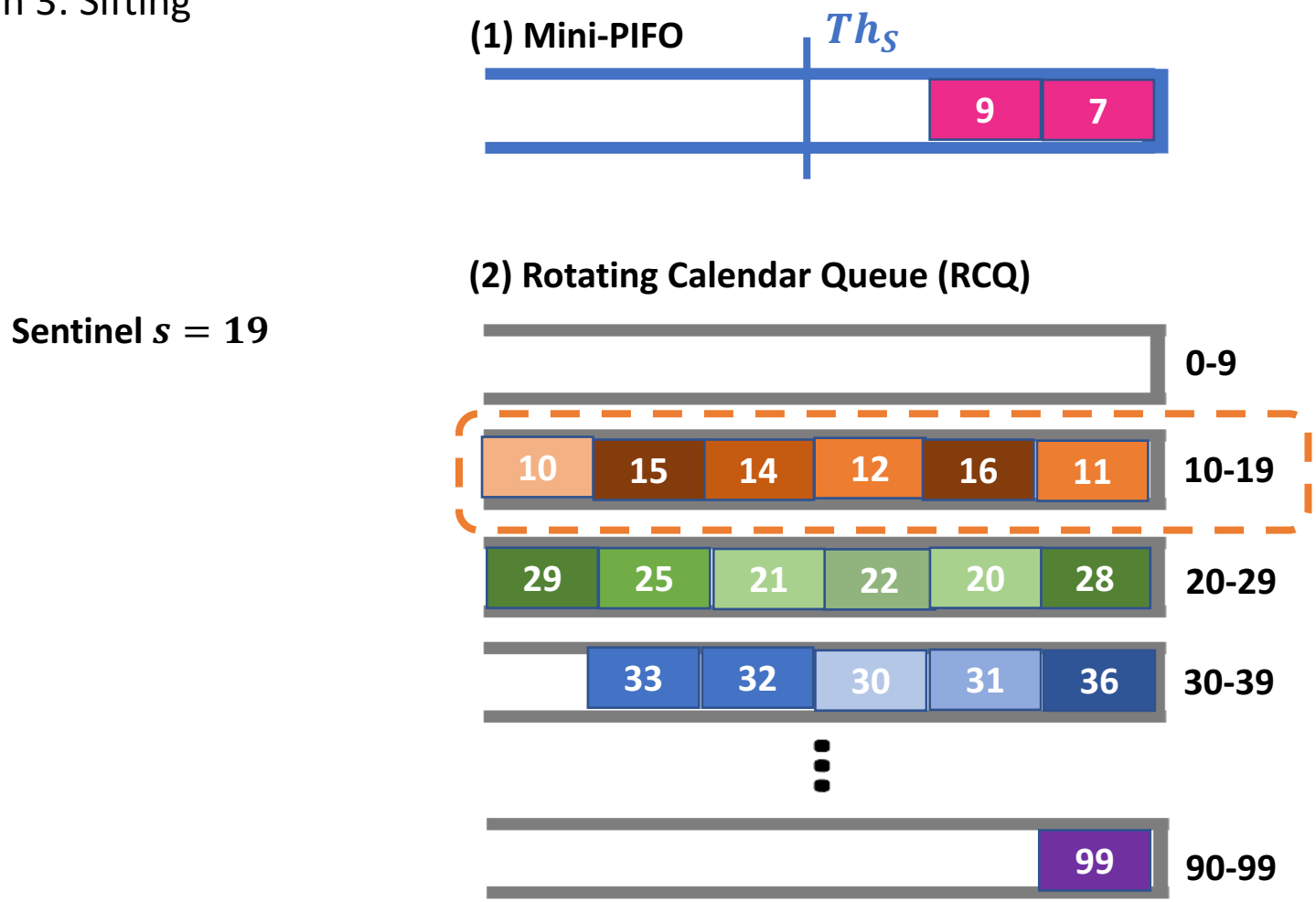


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Sifting

Sifter Operation 3: Sifting

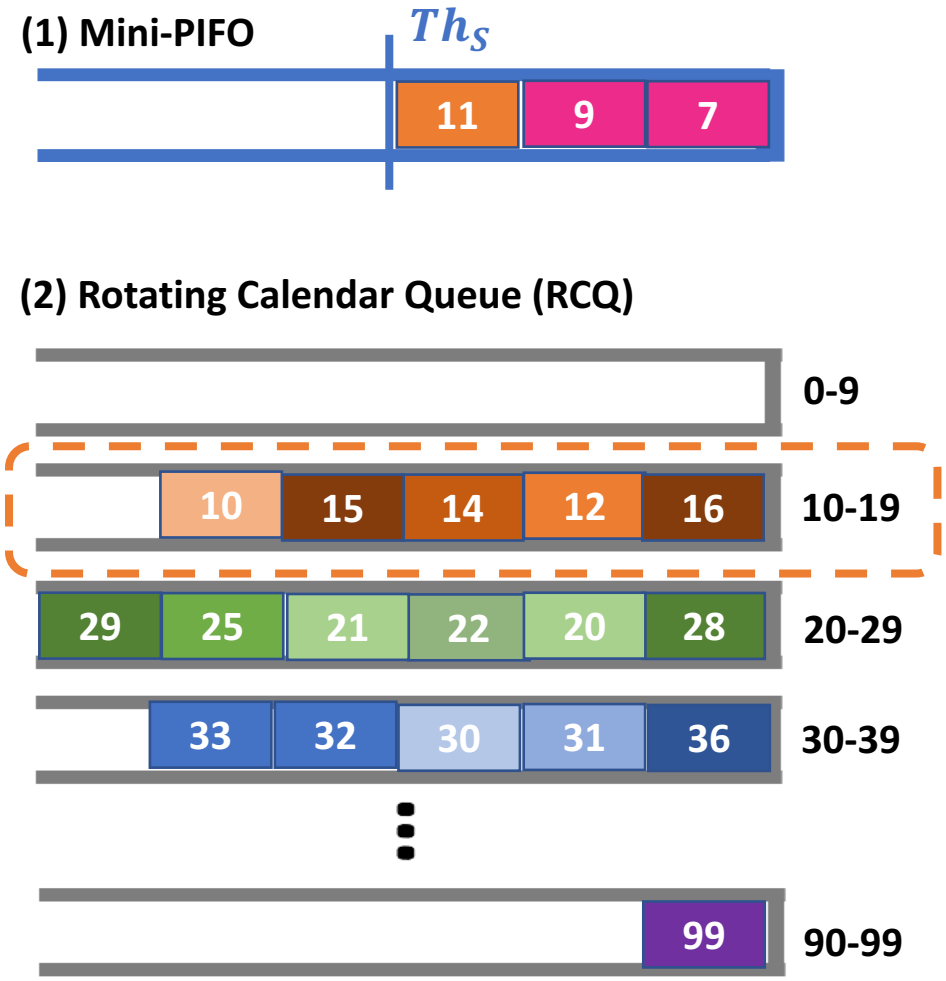


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Sifting

Sifter Operation 3: Sifting

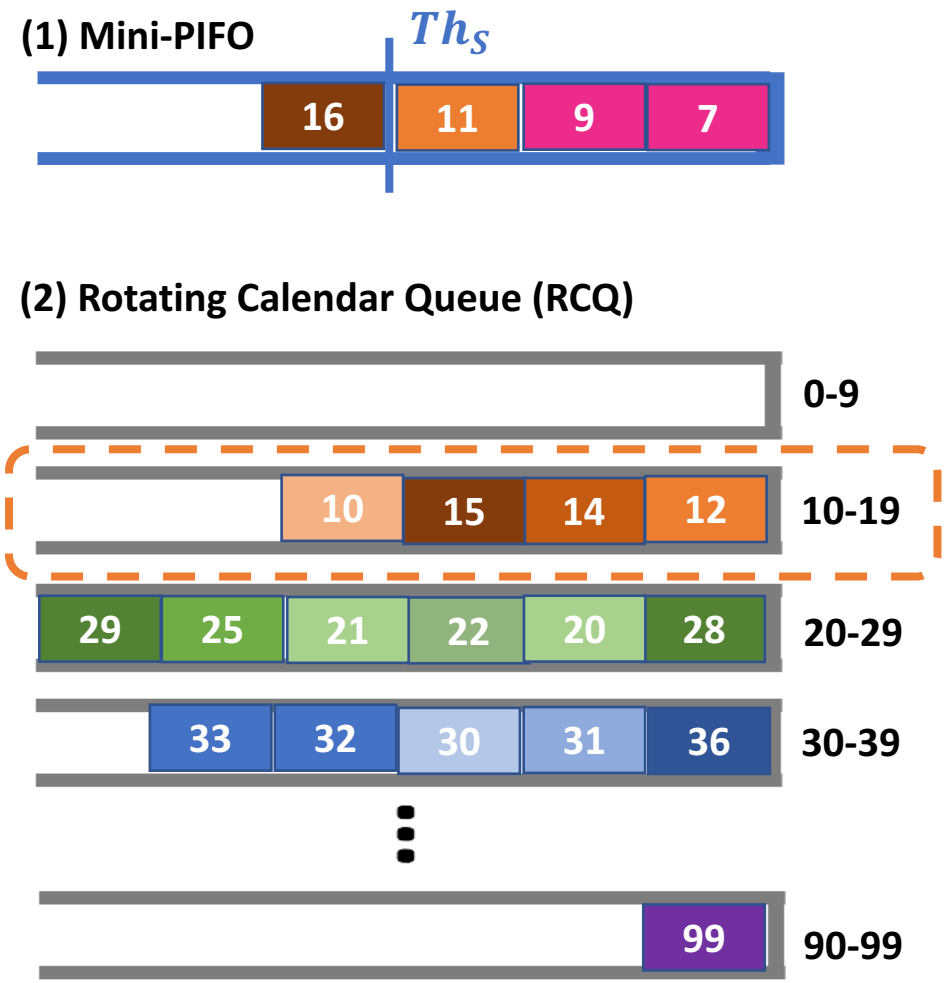


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Sifting

Sifter Operation 3: Sifting



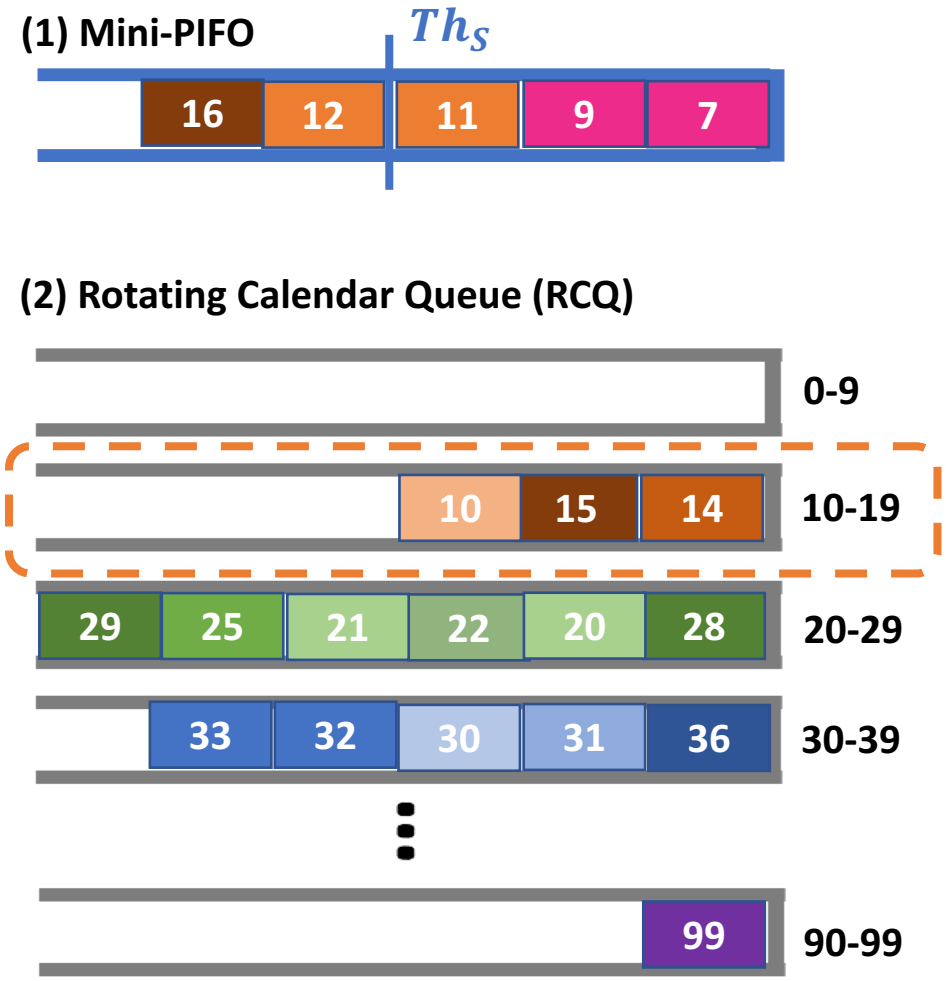
Sentinel $s = 19$

Sifter Architecture

Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter - Sifting

Sifter Operation 3: Sifting



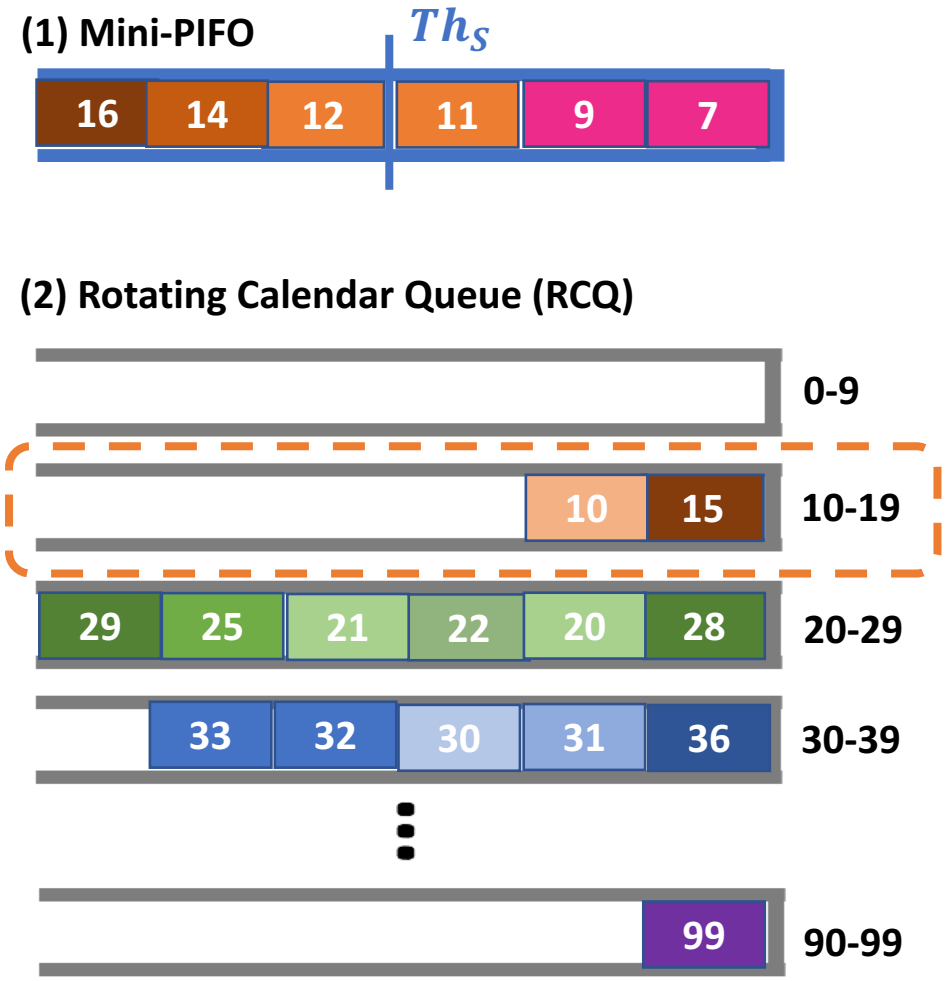
Sentinel $s = 19$

Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Sifting

Sifter Operation 3: Sifting

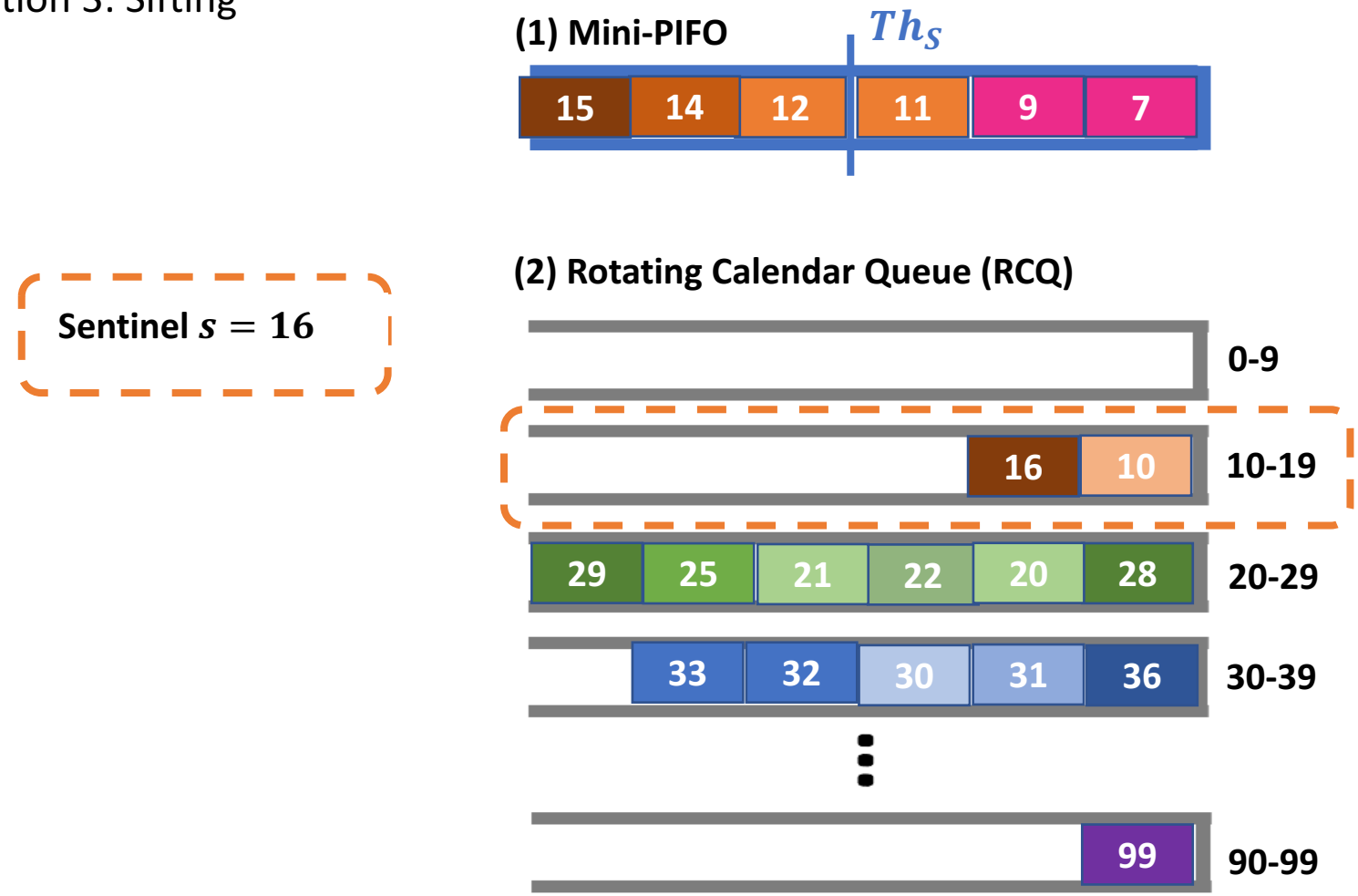


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Sifting

Sifter Operation 3: Sifting

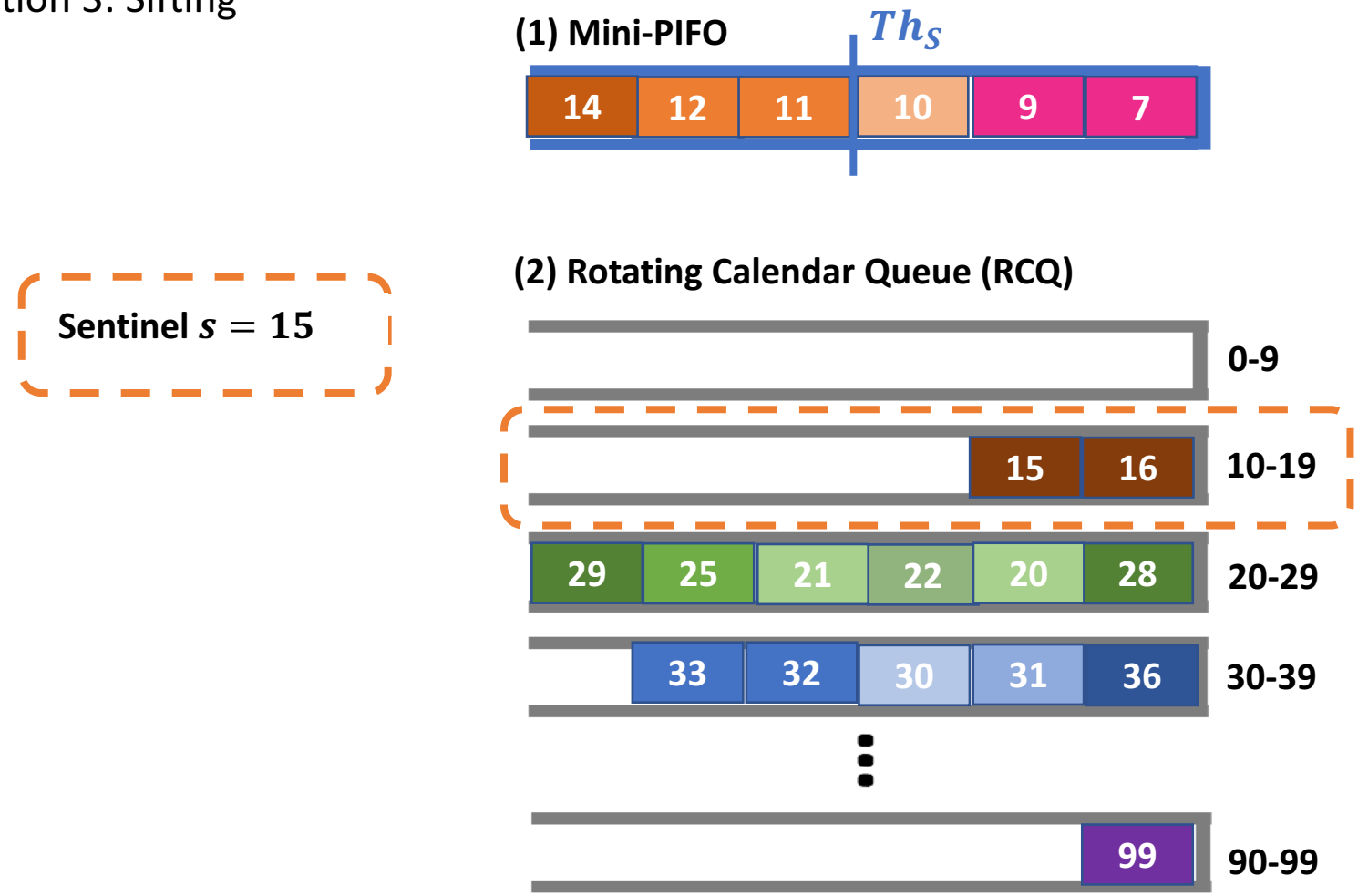


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter - Sifting

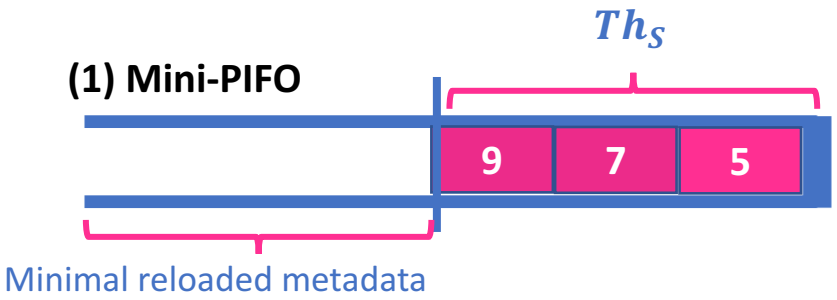
Sifter Operation 3: Sifting

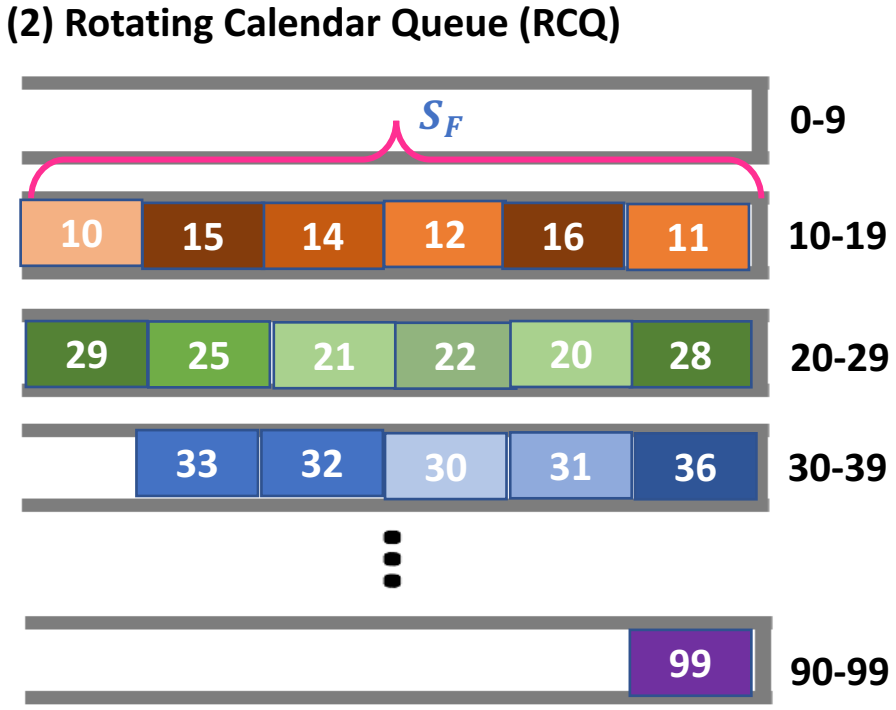


Sentinel: The rank threshold between mini-PIFO and RCQ

Sifter Architecture

Sifter – Inversion Free Condition





Condition for Inversion-free Scheduling

Metadata in the Mini PIFO should buy enough time to traverse a FIFO

(1) Time to drain Mini-PIFO ≥ Time to traverse a FIFO

$$Th_S * K \geq S_F$$

Mini PIFO should occupied by enough metadata by the end of last sifting

(2) Reloaded metadata ≥ Minimal PIFO occupancy

$$S_p - Th_S \geq Th_S \Rightarrow S_P \geq 2 * Th_S$$

Th_S : Sifting threshold S_F : FIFO size

S_P : PIFO size K : Speed-up factor

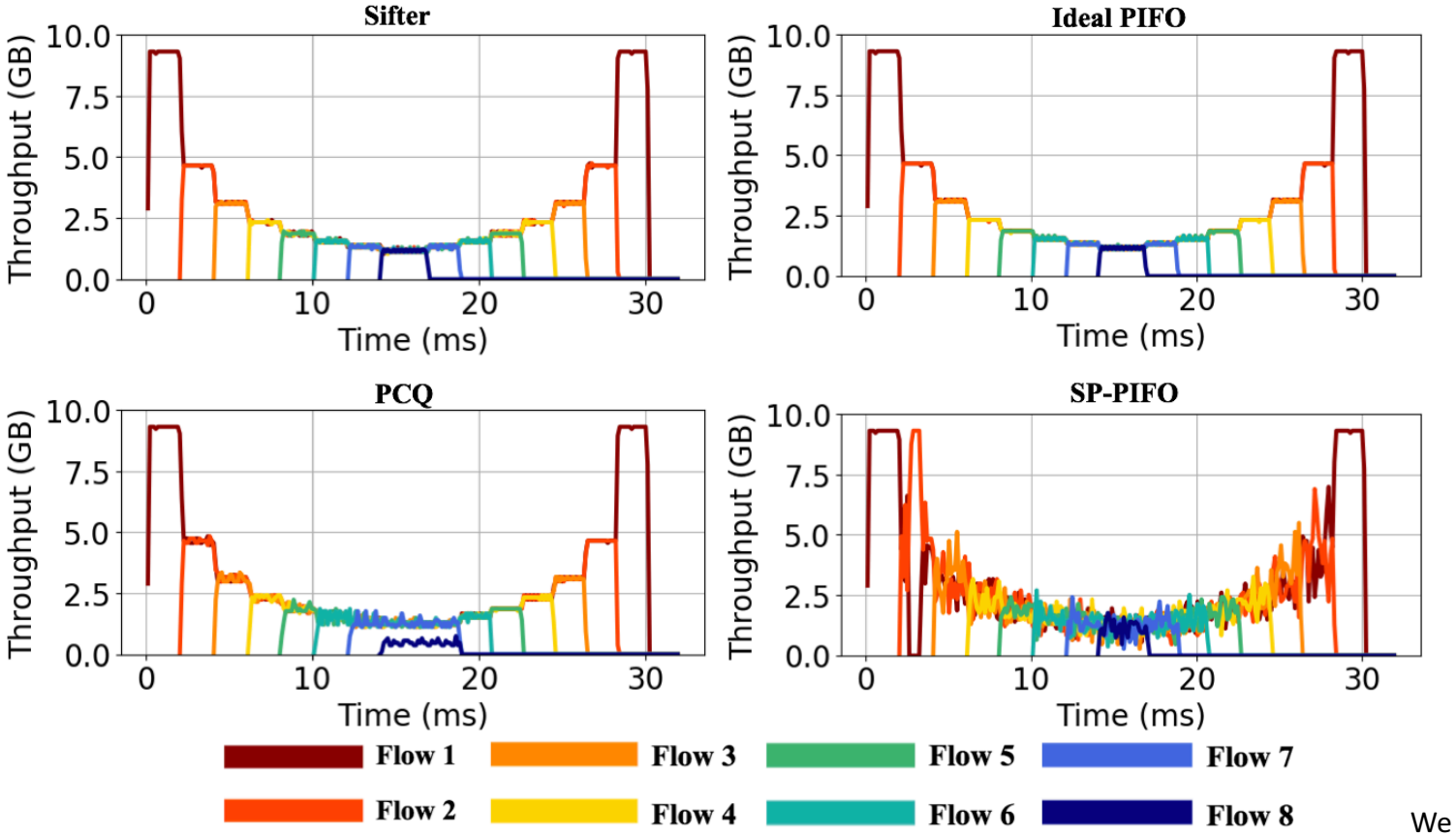
Evaluation and Implementation

Evaluation: NS3 – Single Node Flow Convergence

We use a single node topology to evaluate flow convergence on one bottleneck link:

- 8 src hosts and 1 dst host are connected by a switch with Sifter.
- All links have a bandwidth of 10 Gbps and a delay of 3µs

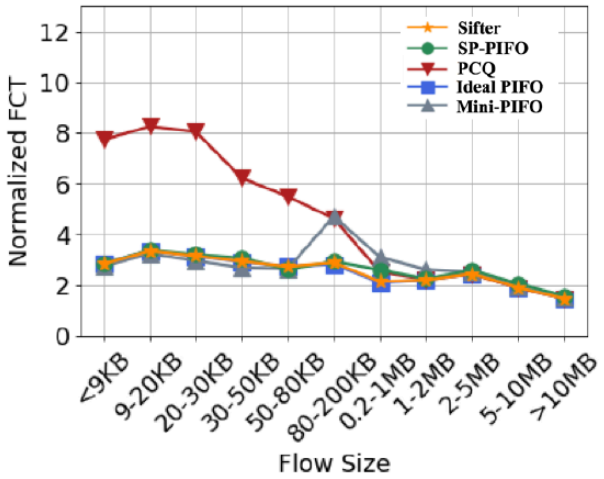
Sifter has an identical **flow convergence performance** as the ideal PIFO



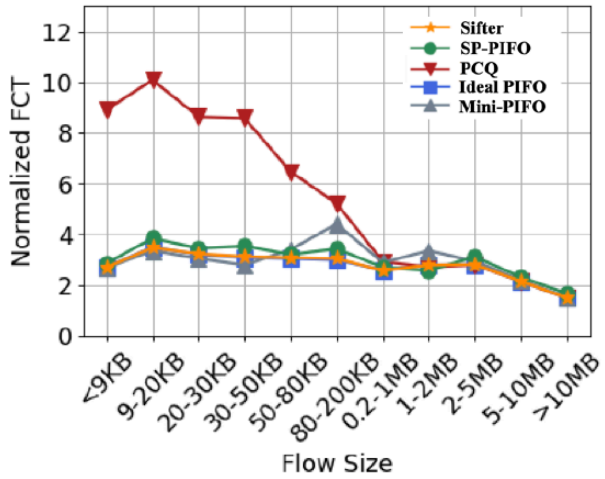
We ran STFQ on Sifter in these evaluations

Evaluation: NS3 – Fat-tree Topology

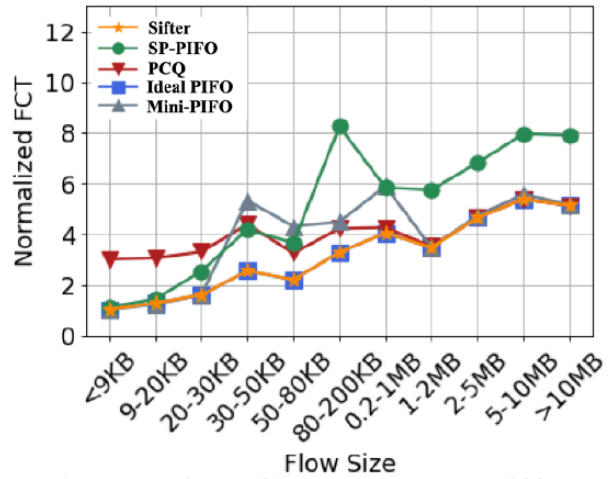
We ran STFQ on Sifter in these evaluations



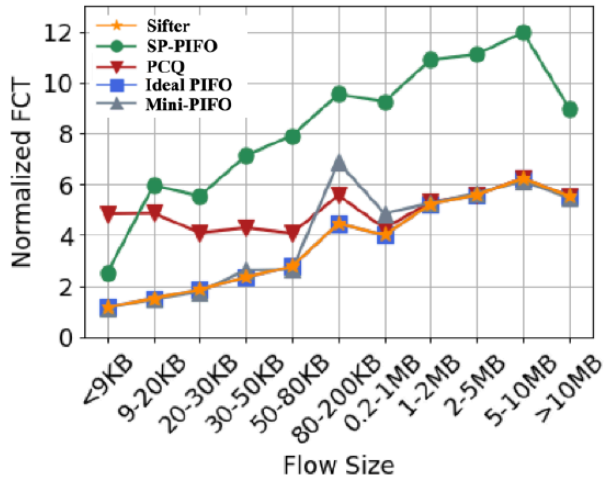
(a) Normalized FCT, Random Pattern, 70% load



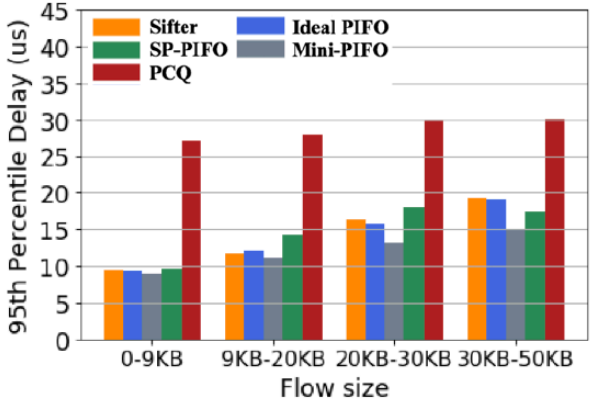
(b) Normalized FCT, Random Pattern, 90% load



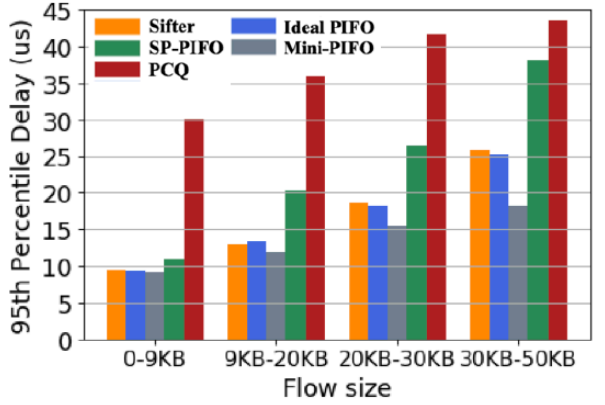
(c) Normalized FCT, In-cast Pattern, 70% load



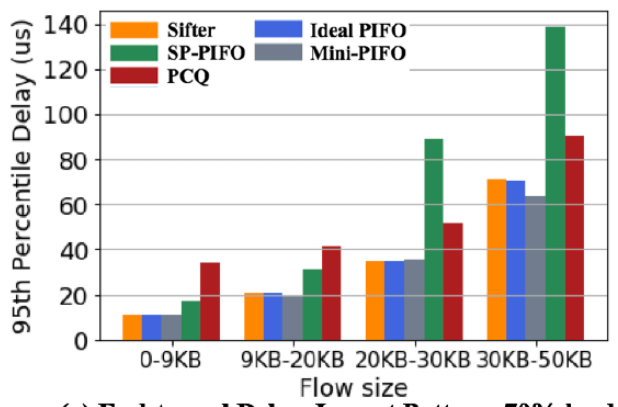
(d) Normalized FCT, In-cast Pattern, 90% load



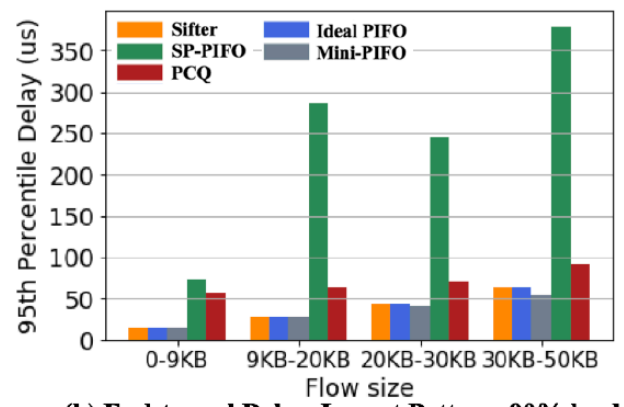
(e) End-to-end Delay, Random Pattern, 70% load



(f) End-to-end Delay, Random Pattern, 90% load



(g) End-to-end Delay, In-cast Pattern, 70% load



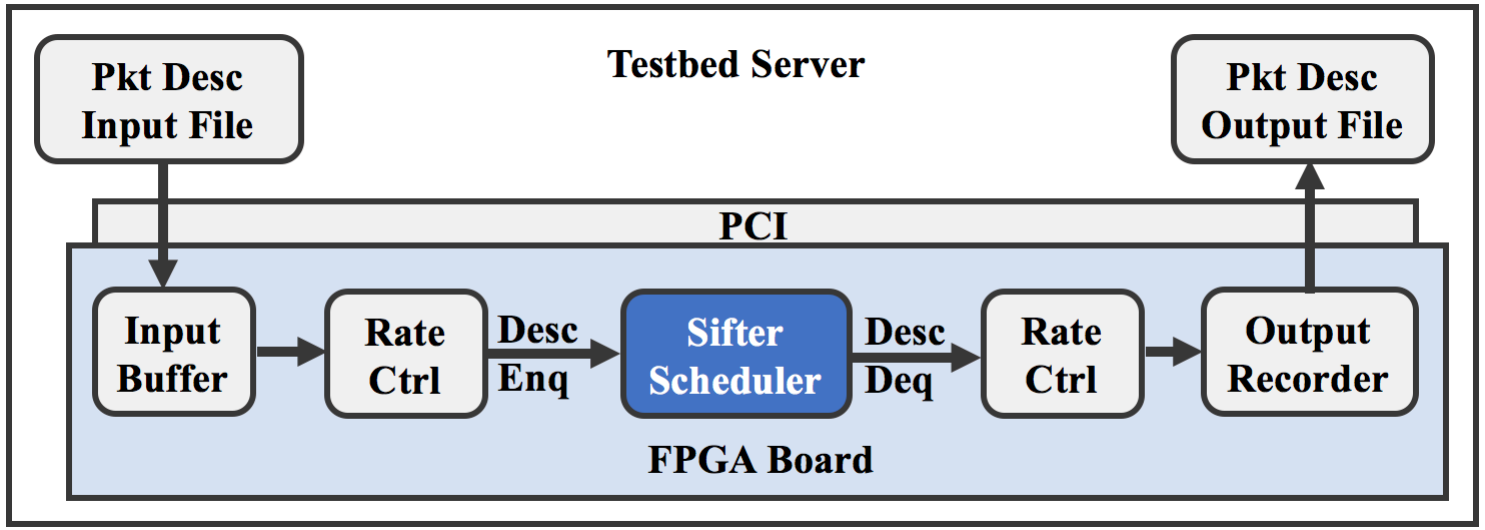
(h) End-to-end Delay, In-cast Pattern, 90% load

Sifter has an identical **flow completion time** and **tail latency performance** as the ideal PIFO

Evaluation: Hardware Testbed

We implemented Sifter in VHDL on a Xilinx Alveo U250 FPGA card with a mid-speed grade XCVU13P FPGA.

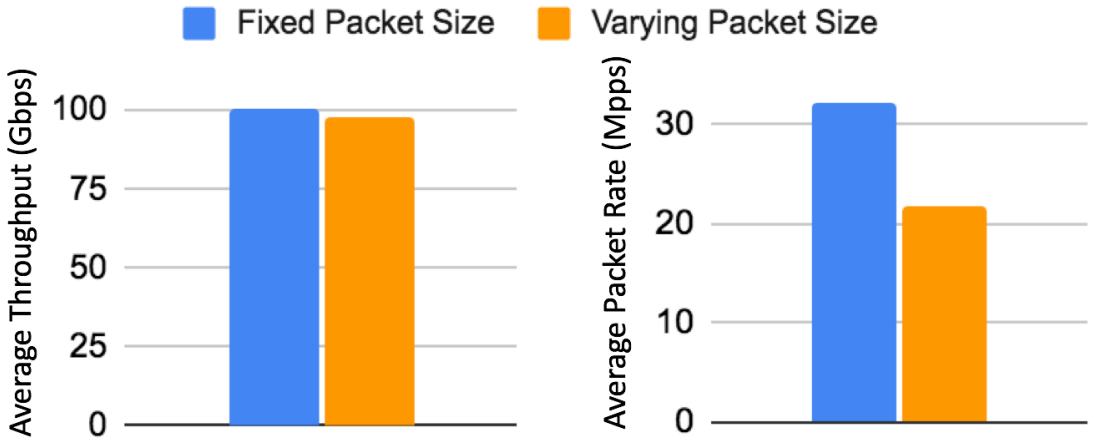
Xilinx Alveo U250 FPGA



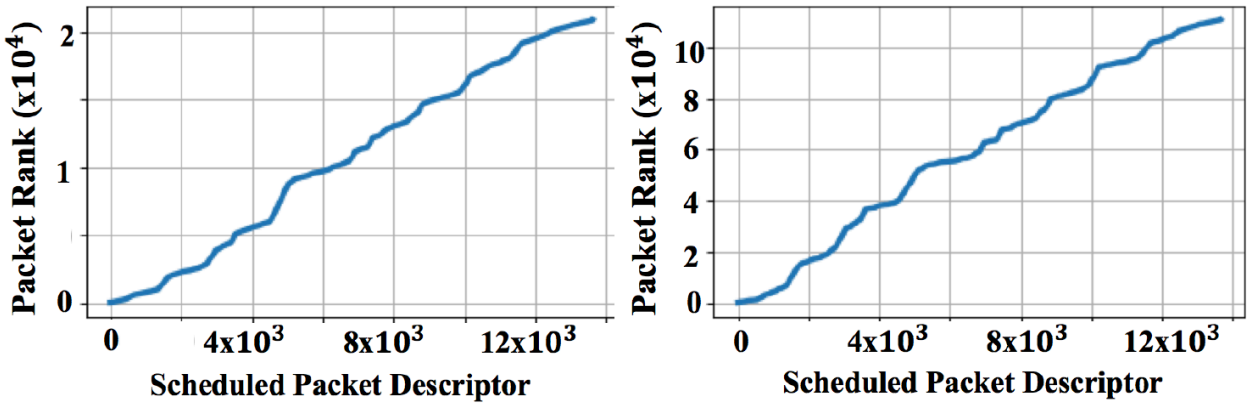
Hardware Testbed

Evaluation: Hardware Testbed

Metadata Scheduling Throughputs



Metadata Scheduling Order



(a) Scheduling Order, fixed packet size (b) Scheduling Order, varying packet size

Sifter hardware prototype supports **100Gbps line rate**

Scheduled packet ranks are monotonically increasing: **Inversion free**

We ran STFQ on Sifter in these evaluations

Max frequency: 322 MHz
Minimal packet size: 370 Bytes

Conclusion

Conclusion

Sifter: An Inversion-Free and Large-Capacity Programmable Packet Scheduler

- Generic inversion-free programmable packet scheduler
- Combines the advantages of PIFO and FIFO-based schedulers
- Two-step `Sifting Sorting` with a mini-PIFO
- Support large capacity of buffer space
- Fast scheduling process makes it able to operate at line rate

Thank you