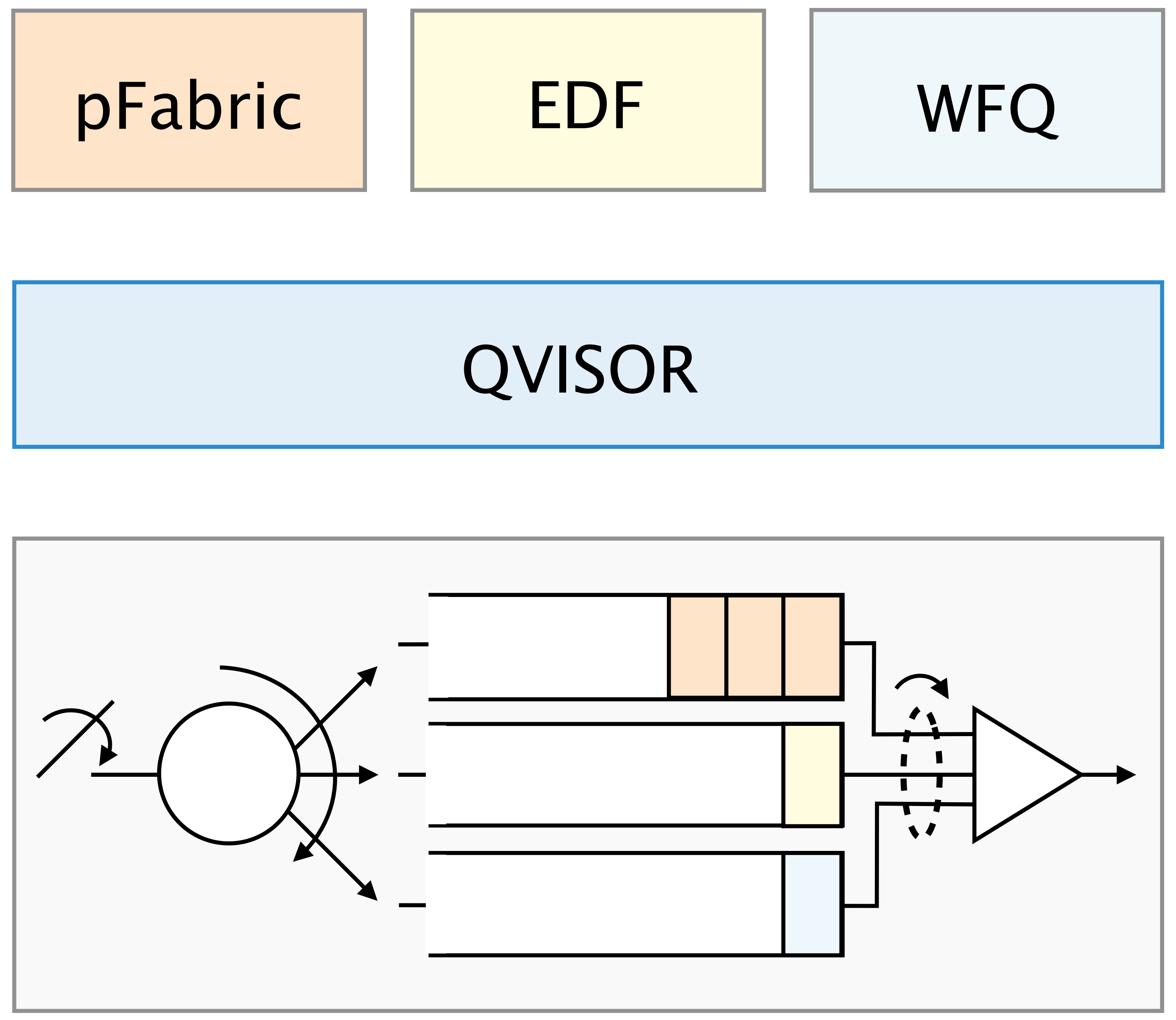
QVISOR: Virtualizing Packet Scheduling Policies



Albert Gran Alcoz

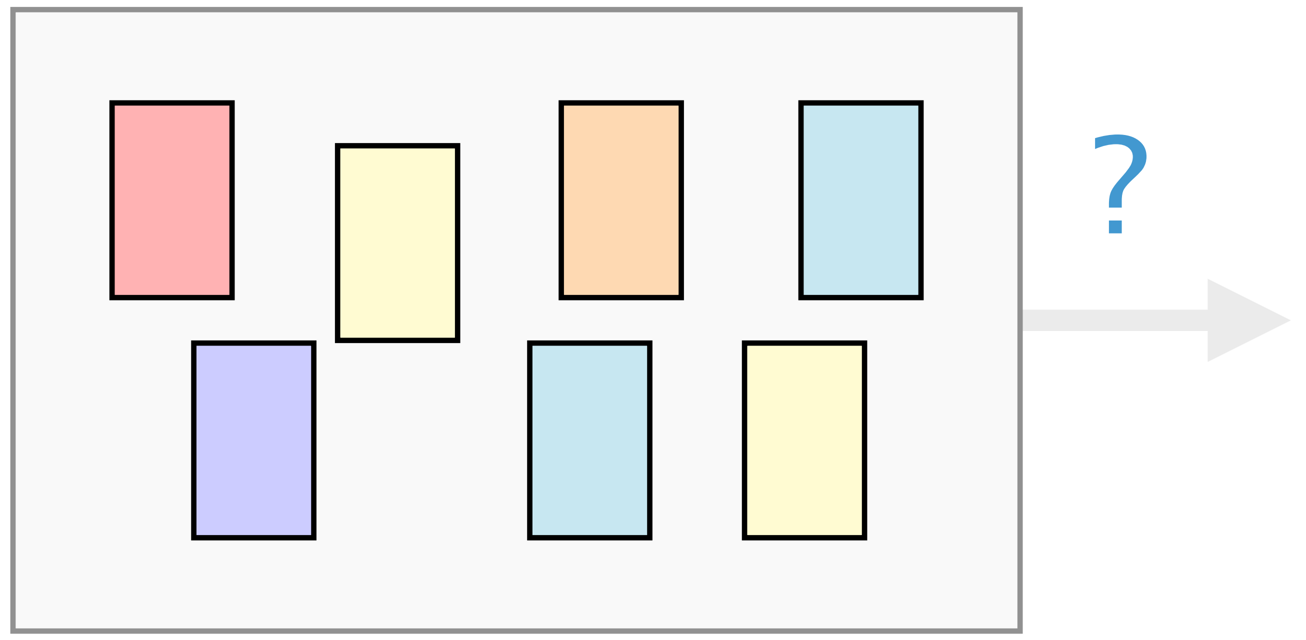
Laurent Vanbever

HotNets

November 29 2023

ETH zürich

Packet scheduling tries to answer a very simple question



What packet to send next

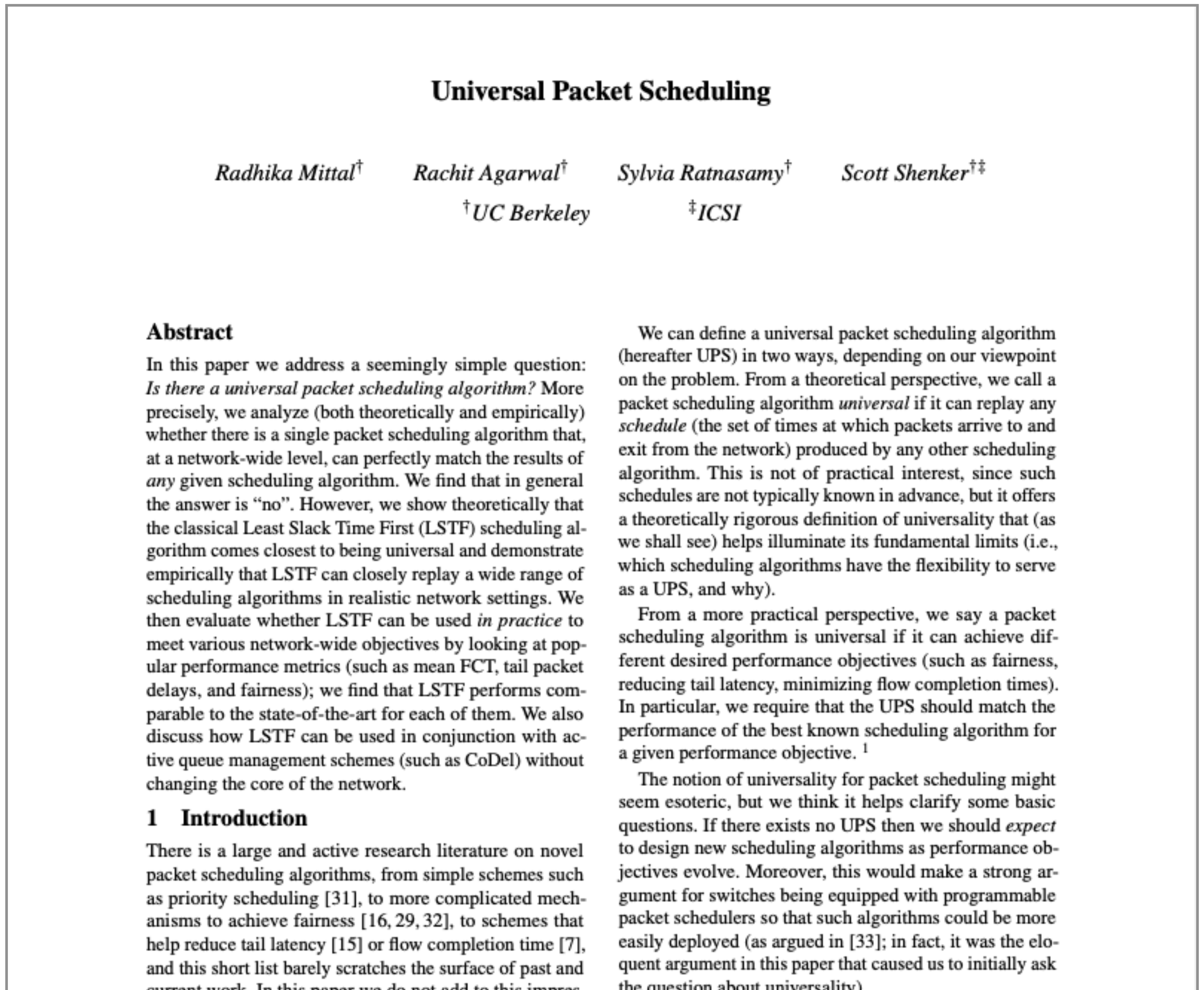
and when?

Countless packet scheduling algorithms have been proposed

Minimize tail latency	FIFO+
Prioritize packets with high slack time	
Minimize flow completion times	pFabric, PIAS
Prioritize packets from short flows	
Enforce fairness	RR, WFQ
Send one packet from each class at a time	

A universal packet scheduler does not exist

NSDI'16



Universal Packet Scheduling

Radhika Mittal[†] *Rachit Agarwal*[†] *Sylvia Ratnasamy*[†] *Scott Shenker*[†‡]

[†]*UC Berkeley* [‡]*ICSI*

Abstract

In this paper we address a seemingly simple question: *Is there a universal packet scheduling algorithm?* More precisely, we analyze (both theoretically and empirically) whether there is a single packet scheduling algorithm that, at a network-wide level, can perfectly match the results of *any* given scheduling algorithm. We find that in general the answer is "no". However, we show theoretically that the classical Least Slack Time First (LSTF) scheduling algorithm comes closest to being universal and demonstrate empirically that LSTF can closely replay a wide range of scheduling algorithms in realistic network settings. We then evaluate whether LSTF can be used *in practice* to meet various network-wide objectives by looking at popular performance metrics (such as mean FCT, tail packet delays, and fairness); we find that LSTF performs comparable to the state-of-the-art for each of them. We also discuss how LSTF can be used in conjunction with active queue management schemes (such as CoDel) without changing the core of the network.

1 Introduction

There is a large and active research literature on novel packet scheduling algorithms, from simple schemes such as priority scheduling [31], to more complicated mechanisms to achieve fairness [16, 29, 32], to schemes that help reduce tail latency [15] or flow completion time [7], and this short list barely scratches the surface of past and current work. In this paper we do not add to this impres-

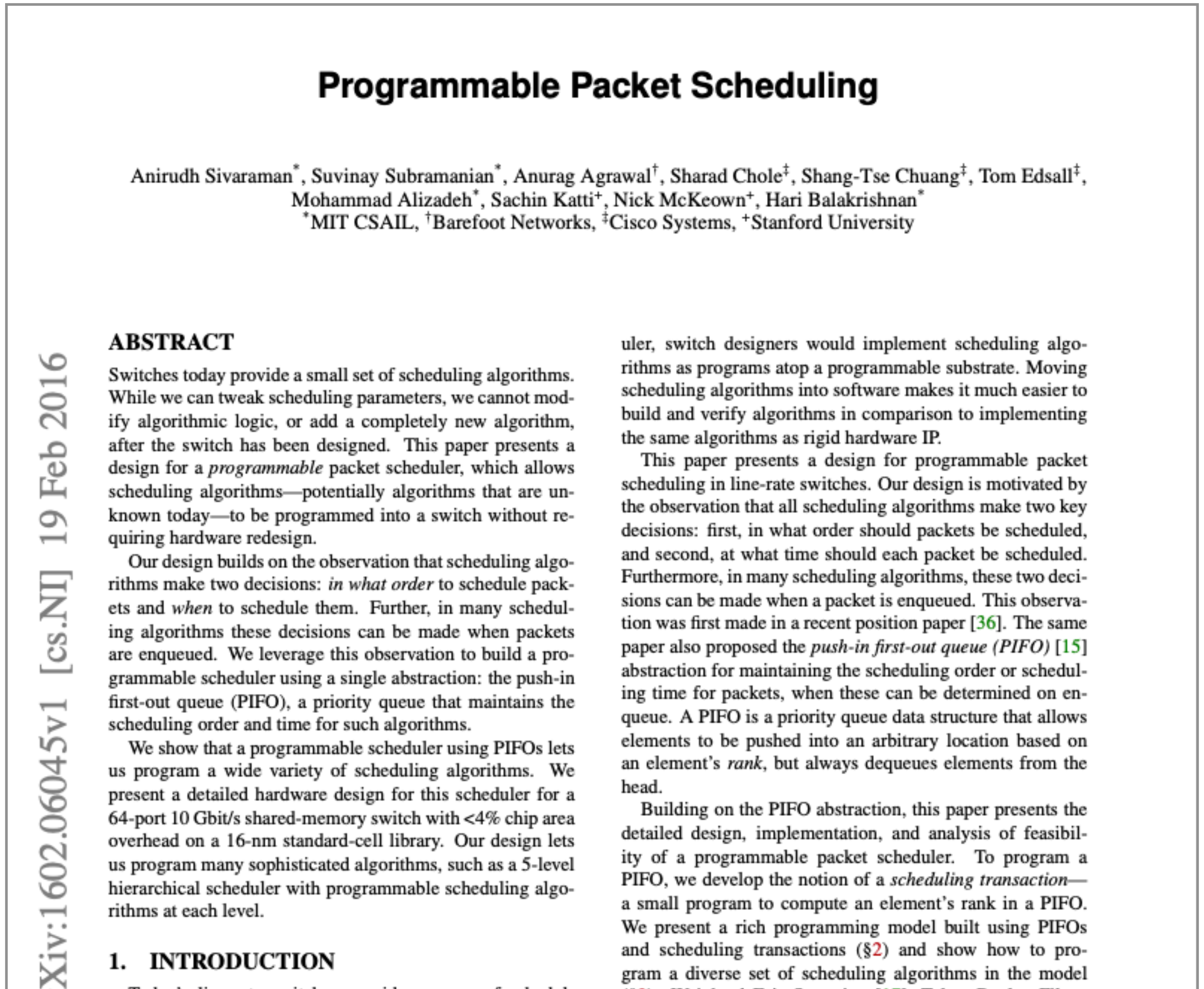
We can define a universal packet scheduling algorithm (hereafter UPS) in two ways, depending on our viewpoint on the problem. From a theoretical perspective, we call a packet scheduling algorithm *universal* if it can replay any *schedule* (the set of times at which packets arrive to and exit from the network) produced by any other scheduling algorithm. This is not of practical interest, since such schedules are not typically known in advance, but it offers a theoretically rigorous definition of universality that (as we shall see) helps illuminate its fundamental limits (i.e., which scheduling algorithms have the flexibility to serve as a UPS, and why).

From a more practical perspective, we say a packet scheduling algorithm is universal if it can achieve different desired performance objectives (such as fairness, reducing tail latency, minimizing flow completion times). In particular, we require that the UPS should match the performance of the best known scheduling algorithm for a given performance objective. [1]

The notion of universality for packet scheduling might seem esoteric, but we think it helps clarify some basic questions. If there exists no UPS then we should *expect* to design new scheduling algorithms as performance objectives evolve. Moreover, this would make a strong argument for switches being equipped with programmable packet schedulers so that such algorithms could be more easily deployed (as argued in [33]; in fact, it was the eloquent argument in this paper that caused us to initially ask the question about universality).

We should make packet scheduling programmable

SIGCOMM'16



Programmable Packet Scheduling

Anirudh Sivaraman*, Suvinay Subramanian*, Anurag Agrawal†, Sharad Chole‡, Shang-Tse Chuang‡, Tom Edsall‡, Mohammad Alizadeh*, Sachin Katti+, Nick McKeown+, Hari Balakrishnan*
*MIT CSAIL, †Barefoot Networks, ‡Cisco Systems, +Stanford University

ABSTRACT

Switches today provide a small set of scheduling algorithms. While we can tweak scheduling parameters, we cannot modify algorithmic logic, or add a completely new algorithm, after the switch has been designed. This paper presents a design for a *programmable* packet scheduler, which allows scheduling algorithms—potentially algorithms that are unknown today—to be programmed into a switch without requiring hardware redesign.

Our design builds on the observation that scheduling algorithms make two decisions: *in what order* to schedule packets and *when* to schedule them. Further, in many scheduling algorithms these decisions can be made when packets are enqueued. We leverage this observation to build a programmable scheduler using a single abstraction: the push-in first-out queue (PIFO), a priority queue that maintains the scheduling order and time for such algorithms.

We show that a programmable scheduler using PIFOs lets us program a wide variety of scheduling algorithms. We present a detailed hardware design for this scheduler for a 64-port 10 Gbit/s shared-memory switch with <4% chip area overhead on a 16-nm standard-cell library. Our design lets us program many sophisticated algorithms, such as a 5-level hierarchical scheduler with programmable scheduling algorithms at each level.

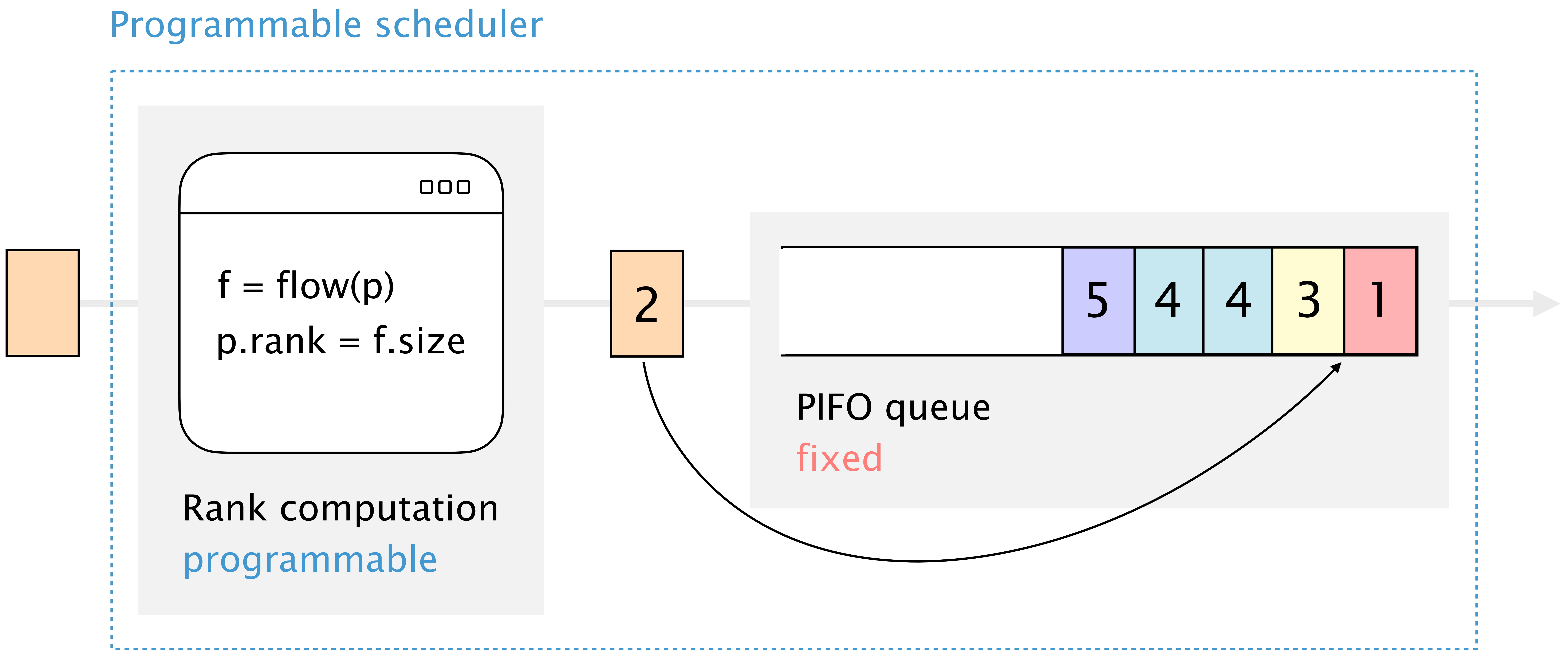
1. INTRODUCTION

uler, switch designers would implement scheduling algorithms as programs atop a programmable substrate. Moving scheduling algorithms into software makes it much easier to build and verify algorithms in comparison to implementing the same algorithms as rigid hardware IP.

This paper presents a design for programmable packet scheduling in line-rate switches. Our design is motivated by the observation that all scheduling algorithms make two key decisions: first, in what order should packets be scheduled, and second, at what time should each packet be scheduled. Furthermore, in many scheduling algorithms, these two decisions can be made when a packet is enqueued. This observation was first made in a recent position paper [36]. The same paper also proposed the *push-in first-out queue (PIFO)* [15] abstraction for maintaining the scheduling order or scheduling time for packets, when these can be determined on enqueue. A PIFO is a priority queue data structure that allows elements to be pushed into an arbitrary location based on an element's *rank*, but always dequeues elements from the head.

Building on the PIFO abstraction, this paper presents the detailed design, implementation, and analysis of feasibility of a programmable packet scheduler. To program a PIFO, we develop the notion of a *scheduling transaction*—a small program to compute an element's rank in a PIFO. We present a rich programming model built using PIFOs and scheduling transactions (§2) and show how to program a diverse set of scheduling algorithms in the model

Push-In First-Out (PIFO) queues enable programmable packet scheduling



Programmable scheduling allows deploying any one algorithm

Programmable scheduling allows deploying any one algorithm

pFabric	STFQ
FIFO+	SFQ
SFF	LSTF
PIAS	WFQ
EDF	PQ
RR	SRPT …

Which one?

Programmable scheduling allows deploying any one algorithm

pFabric	STFQ
FIFO+	SFQ
SFF	LSTF
PIAS	WFQ
EDF	PQ
RR	SRPT …

Just one?

Can we run multiple scheduling algorithms…

simultaneously,

on the same resources?

Reviewer 2: No, we can’t

Different scheduling algorithms

clash with each other

The way in which algorithms clash

changes over time

Naively merging scheduling algorithms does not work

pFabric	EDF
Minimize FCTs	Maximize satisfied deadlines
Ranks: Remaining flow size	Ranks: Flow deadline
e.g., 1000, 10^6 (bytes)	e.g., 10, 20, 40 (ms)

Naively merging scheduling algorithms does not work

pFabric

Minimize FCTs

Ranks: Remaining flow size

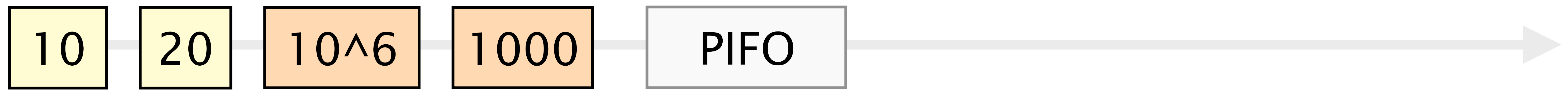
e.g., 1000, 10^6 (bytes)

EDF

Maximize satisfied deadlines

Ranks: Flow deadline

e.g., 10, 20, 40 (ms)



Naively merging scheduling algorithms does not work

pFabric

Minimize FCTs

Ranks: Remaining flow size

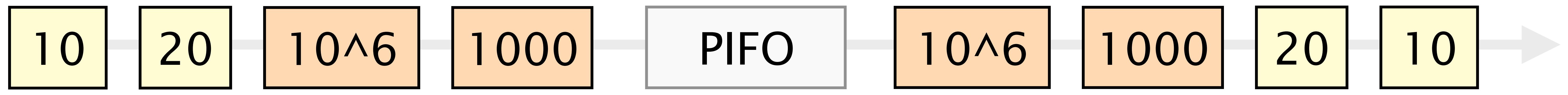
e.g., 1000, 10^6 (bytes)

EDF

Maximize satisfied deadlines

Ranks: Flow deadline

e.g., 10, 20, 40 (ms)



Information in packet ranks is relative

We can normalize and quantize policies

to compare them fairly

We can reason about worst-case performance

and update policies at runtime

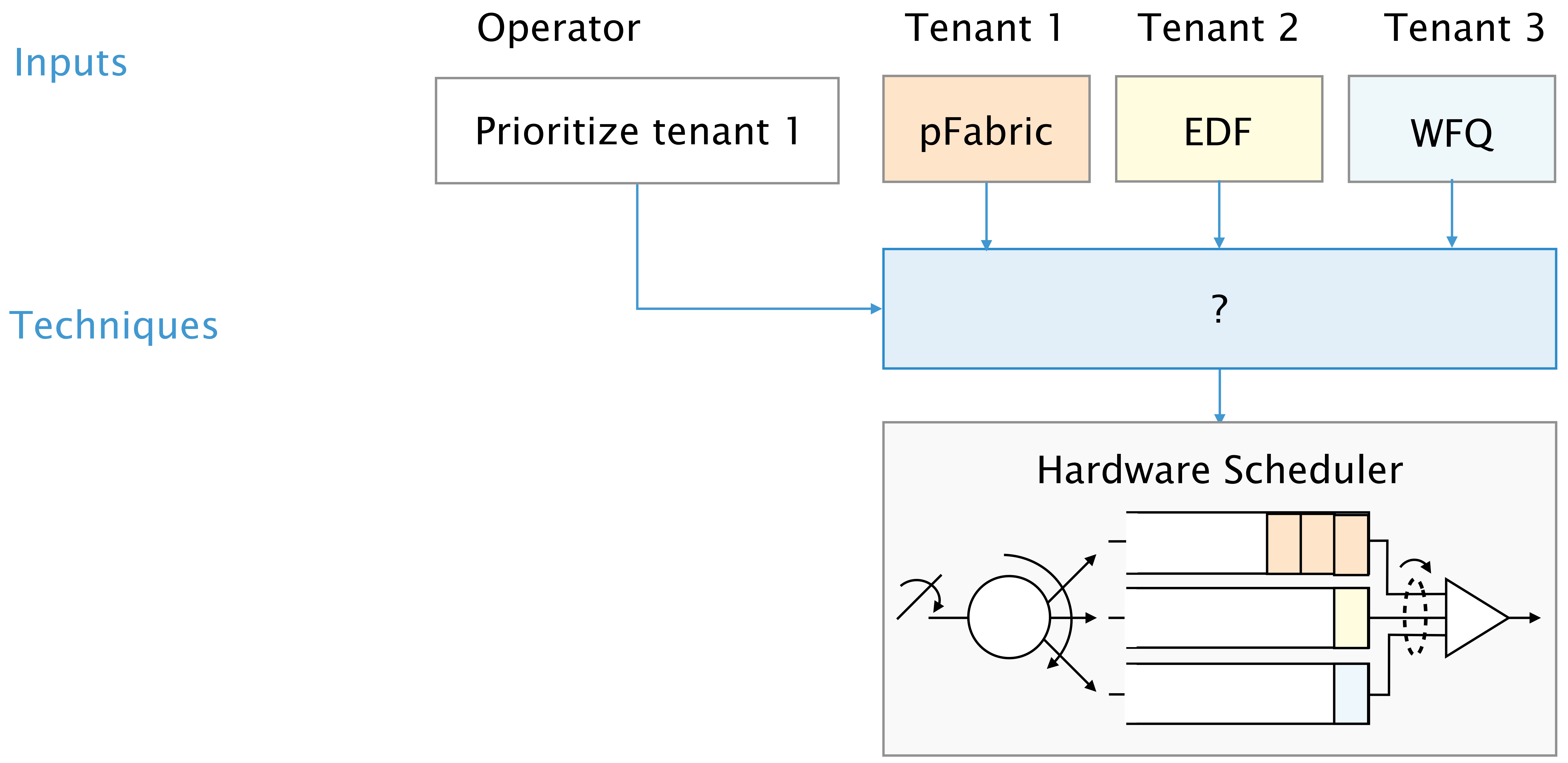
We can define high-level policies

to decide what to prioritize in case of clash

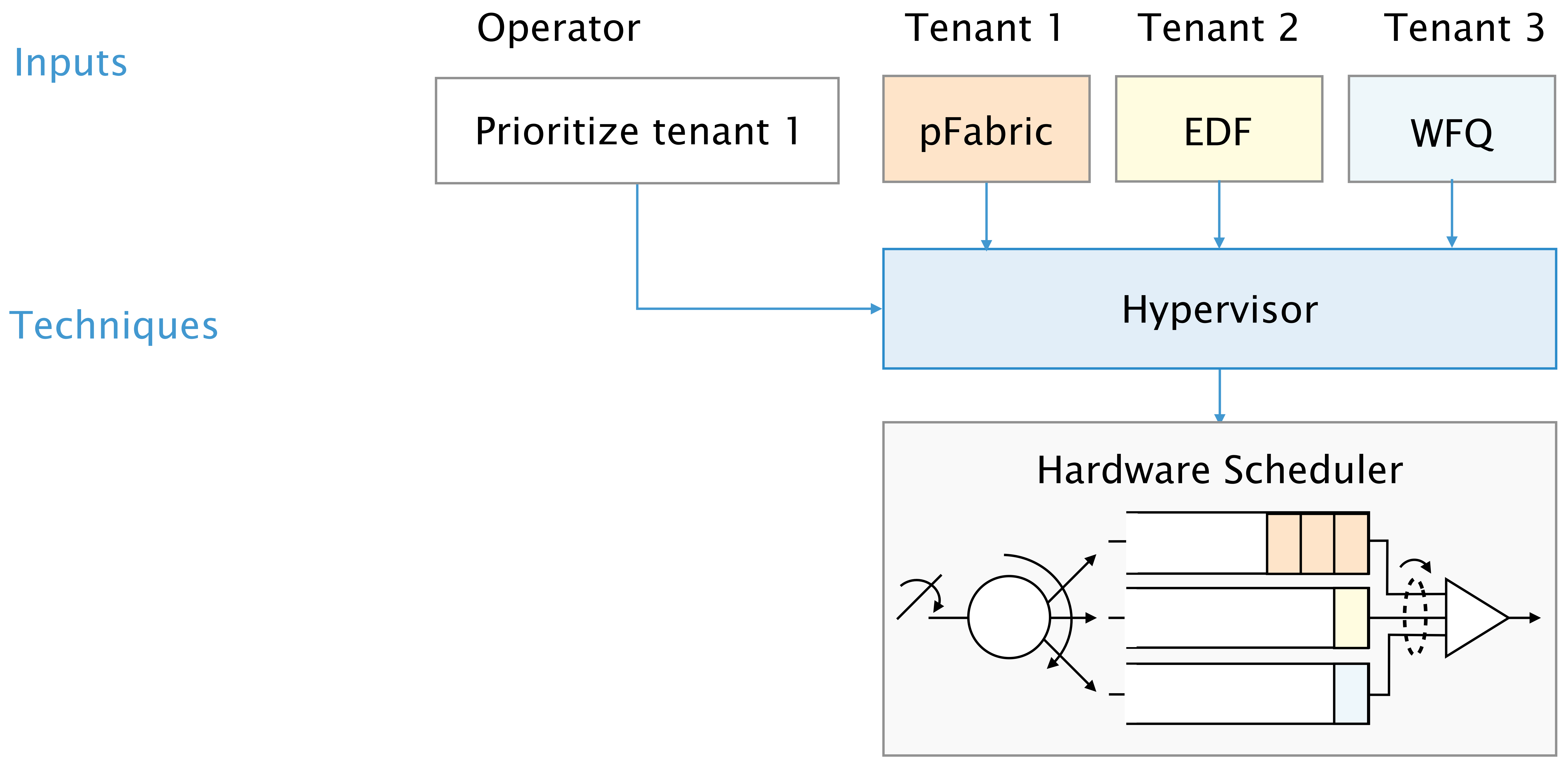
What would it take to run multiple scheduling algorithms?

	Operator	Tenant 1	Tenant 2	Tenant 3
Inputs	Prioritize tenant 1	pFabric	EDF	WFQ

What would it take to run multiple scheduling algorithms?



What would it take to run multiple scheduling algorithms?



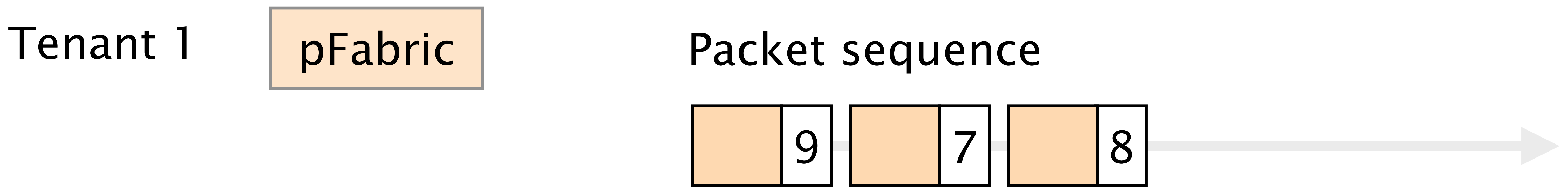
Introducing…

QVISOR

A *packet scheduling* hypervisor

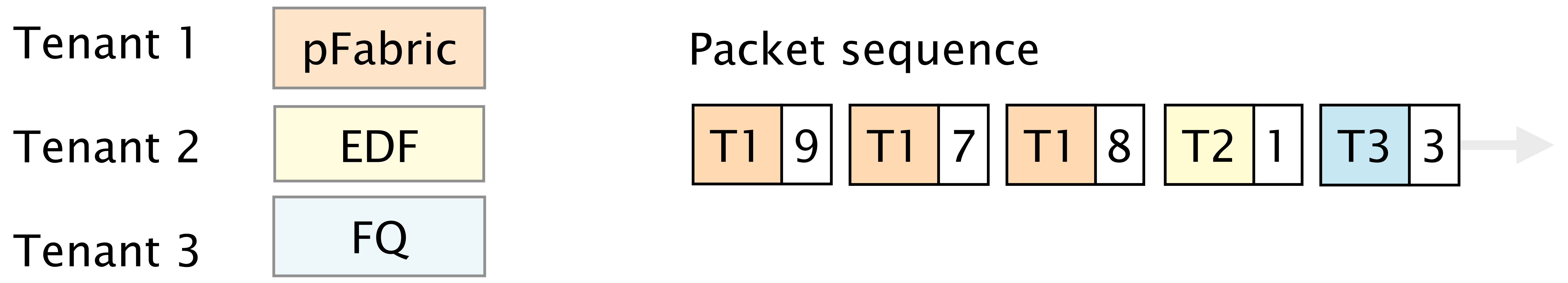
Tenants have the illusion that their traffic is scheduled by a PIFO queue

Tenants label each packet with a rank and the tenant ID



Tenants have the illusion that their traffic is scheduled by a PIFO queue

Tenants label each packet with a rank and the tenant ID



Tenants have the illusion that their traffic is scheduled by a PIFO queue

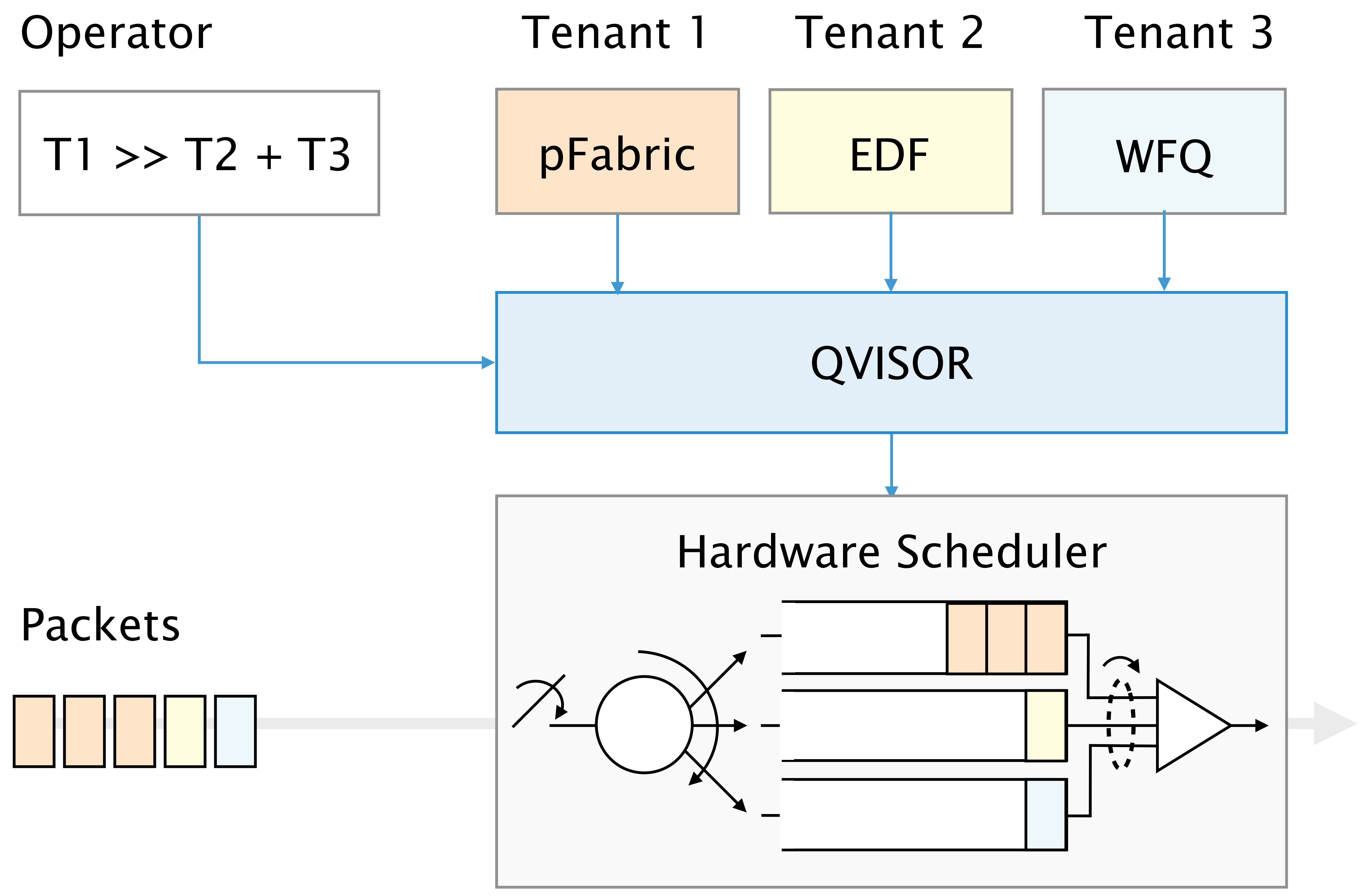
Operators define their policy with a composition language

>>	Strict priority
>	Best-effort priority
+	Sharing

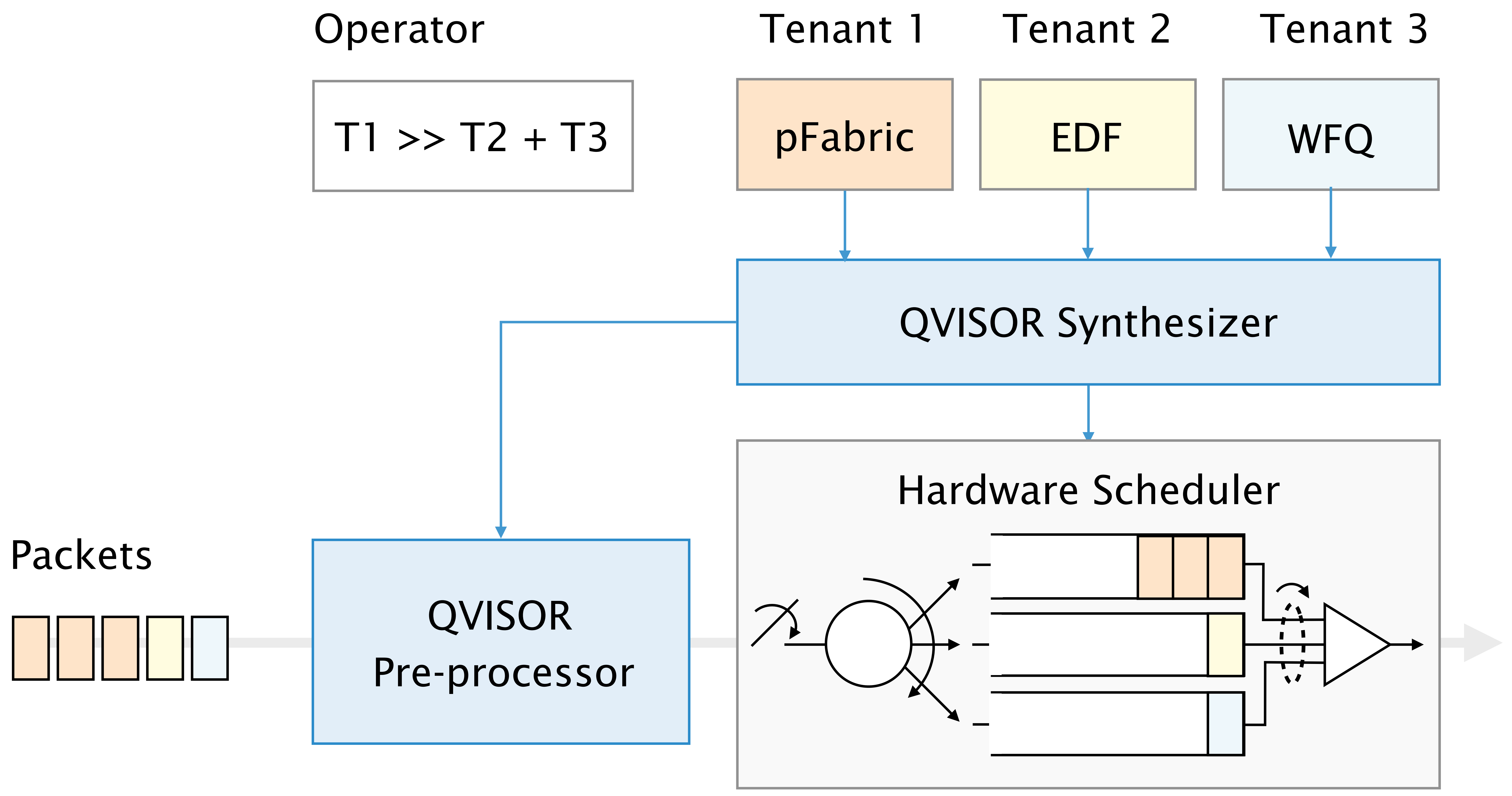
Policy:

T1 >> T2 + T3

QVISOR takes as input the policies from the tenants and the operator



QVISOR synthesizes a joint scheduling function and deploys it to hardware



QVISOR’s synthesizer generates a set of rank-transformation functions

Currently, the synthesizer supports two operation types

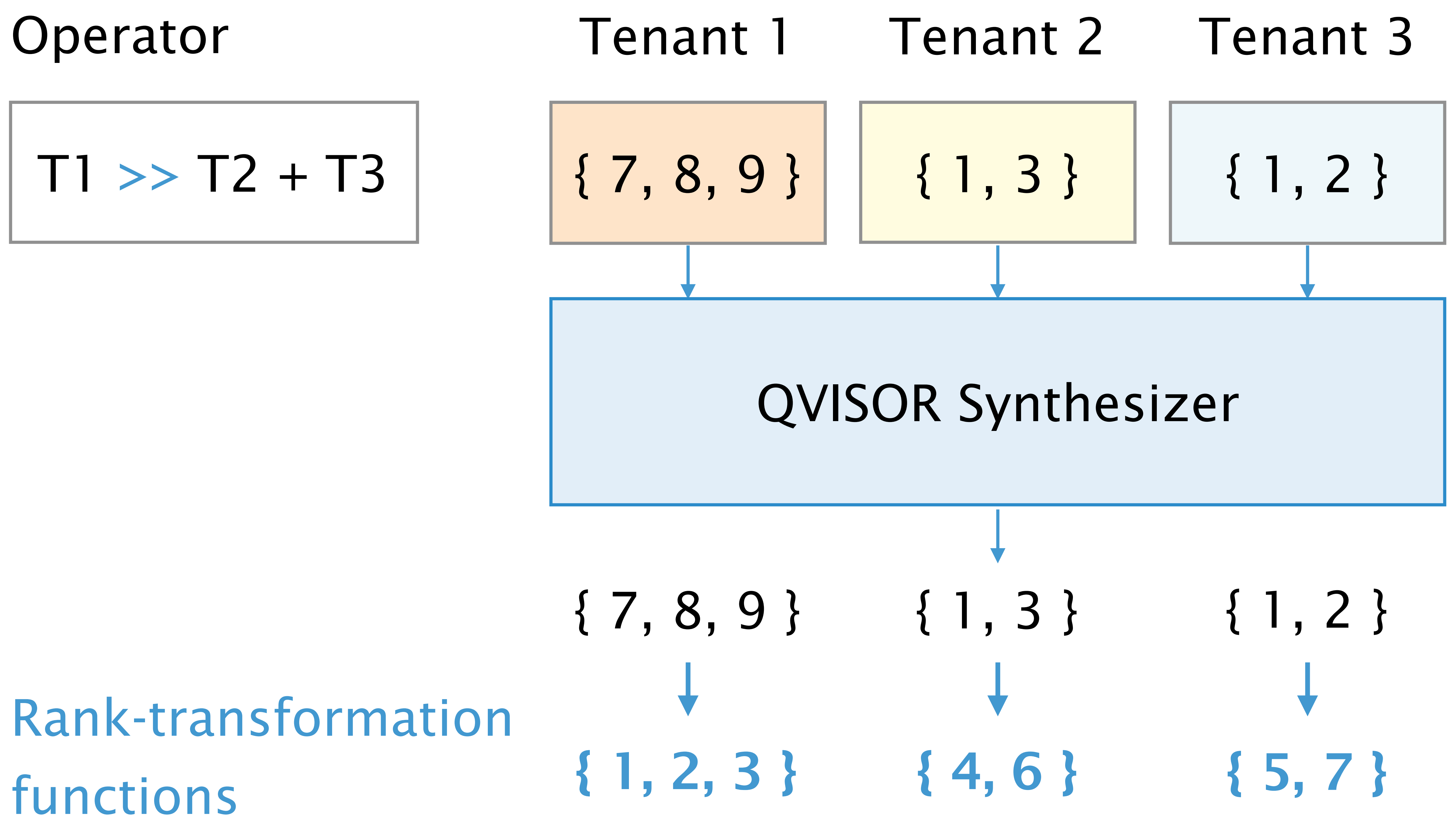
Rank normalizations

$\{ 700, 800, 900 \} \longrightarrow \{ 7, 8, 9 \}$

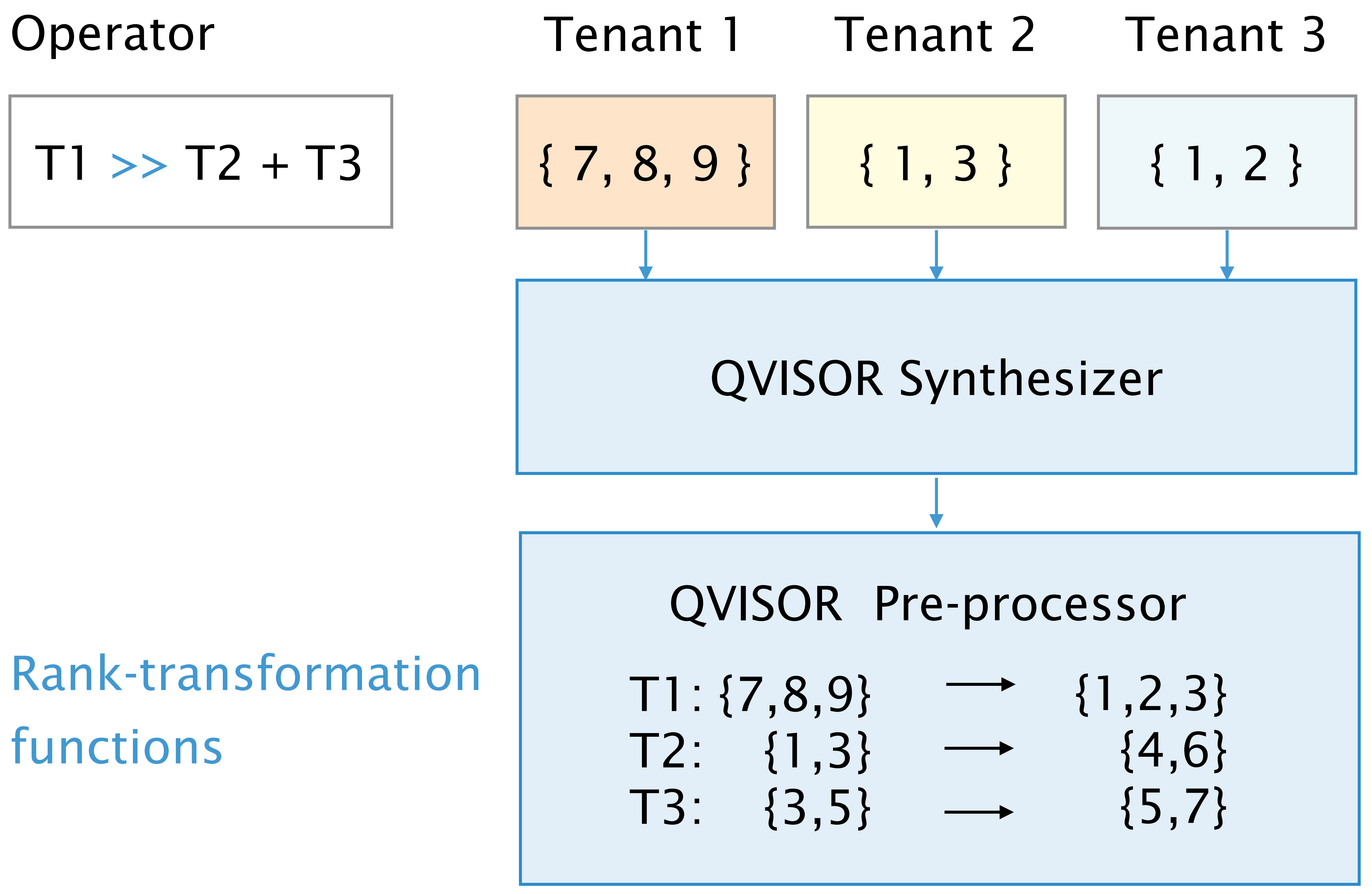
Rank shifts

$\{ 7, 8, 9 \} \longrightarrow \{ 1, 2, 3 \}$

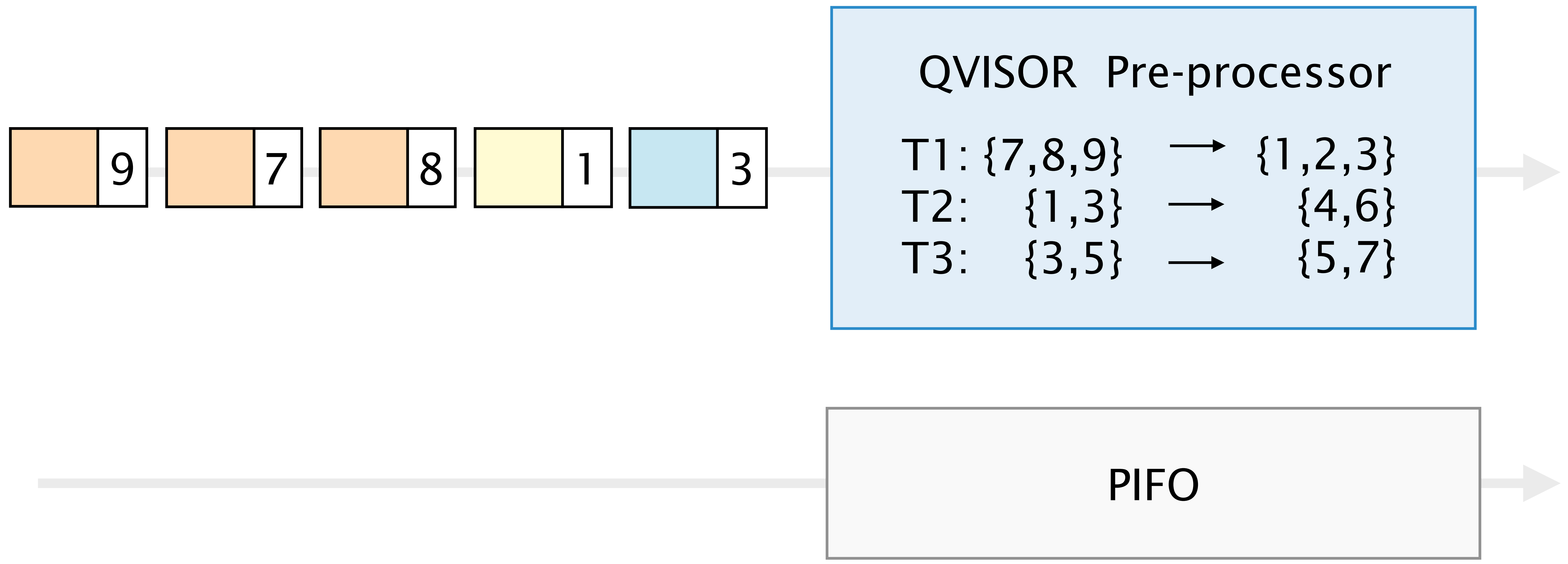
QVISOR's synthesizer generates a set of rank-transformation functions



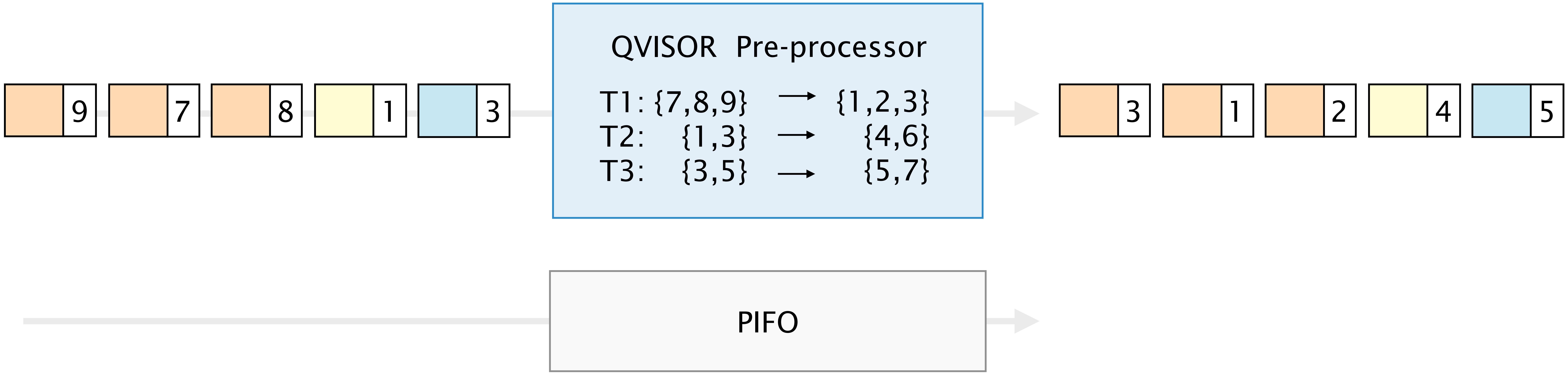
QVISOR's synthesizer generates a set of rank-transformation functions



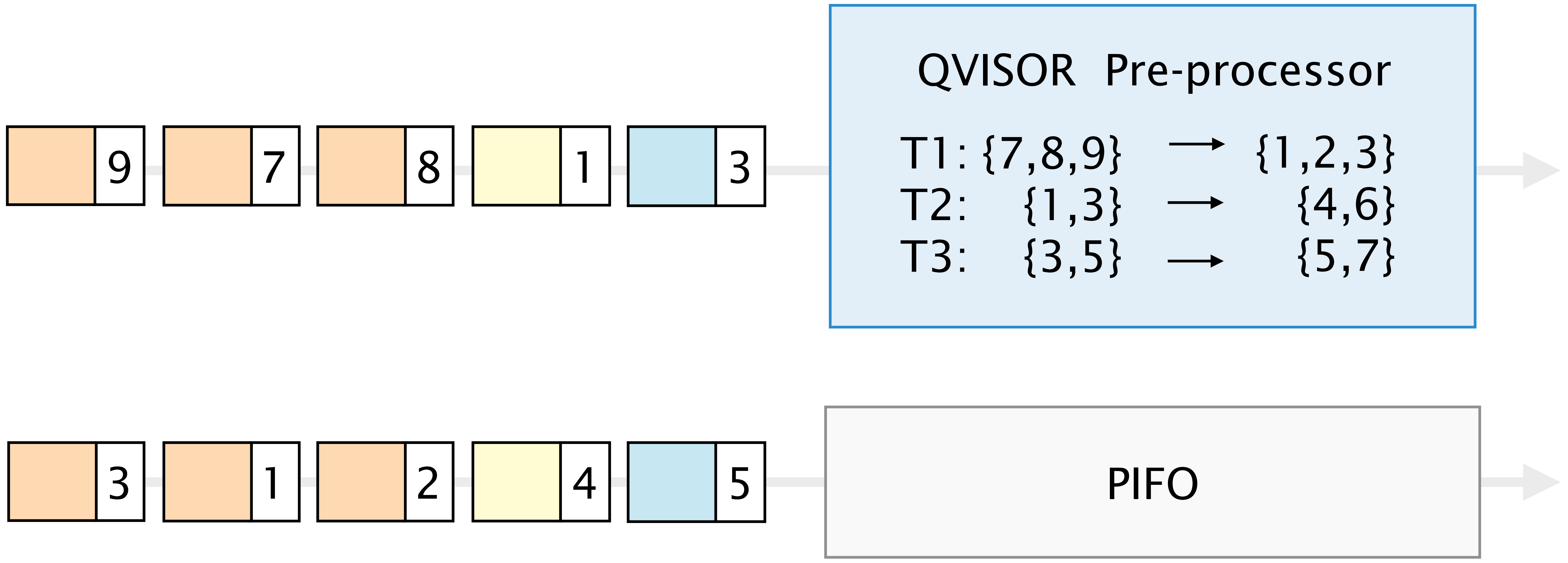
QVISOR’s pre-processor applies the transformation functions at line rate in the data plane



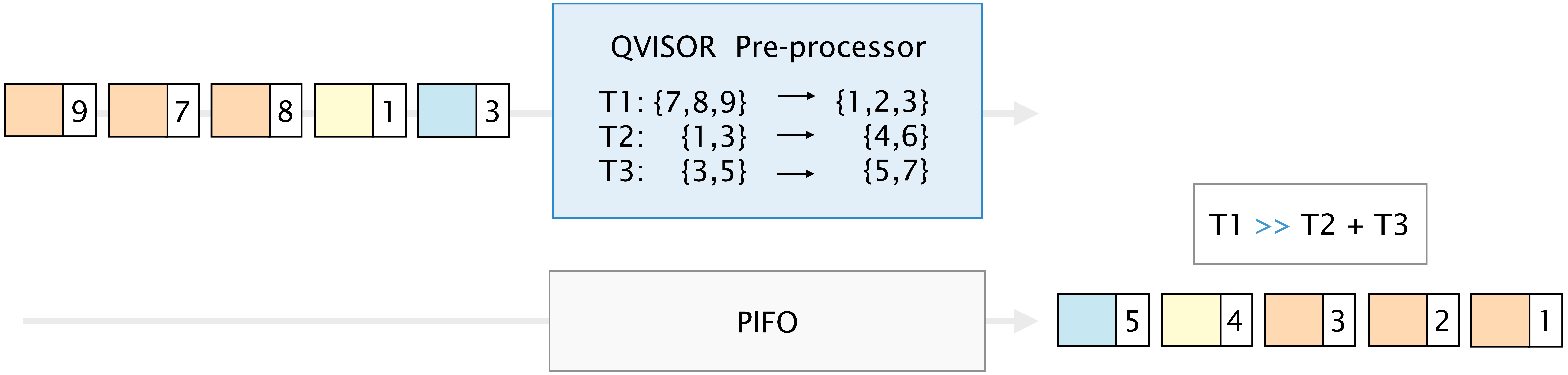
QVISOR's pre-processor applies the transformation functions at line rate in the data plane



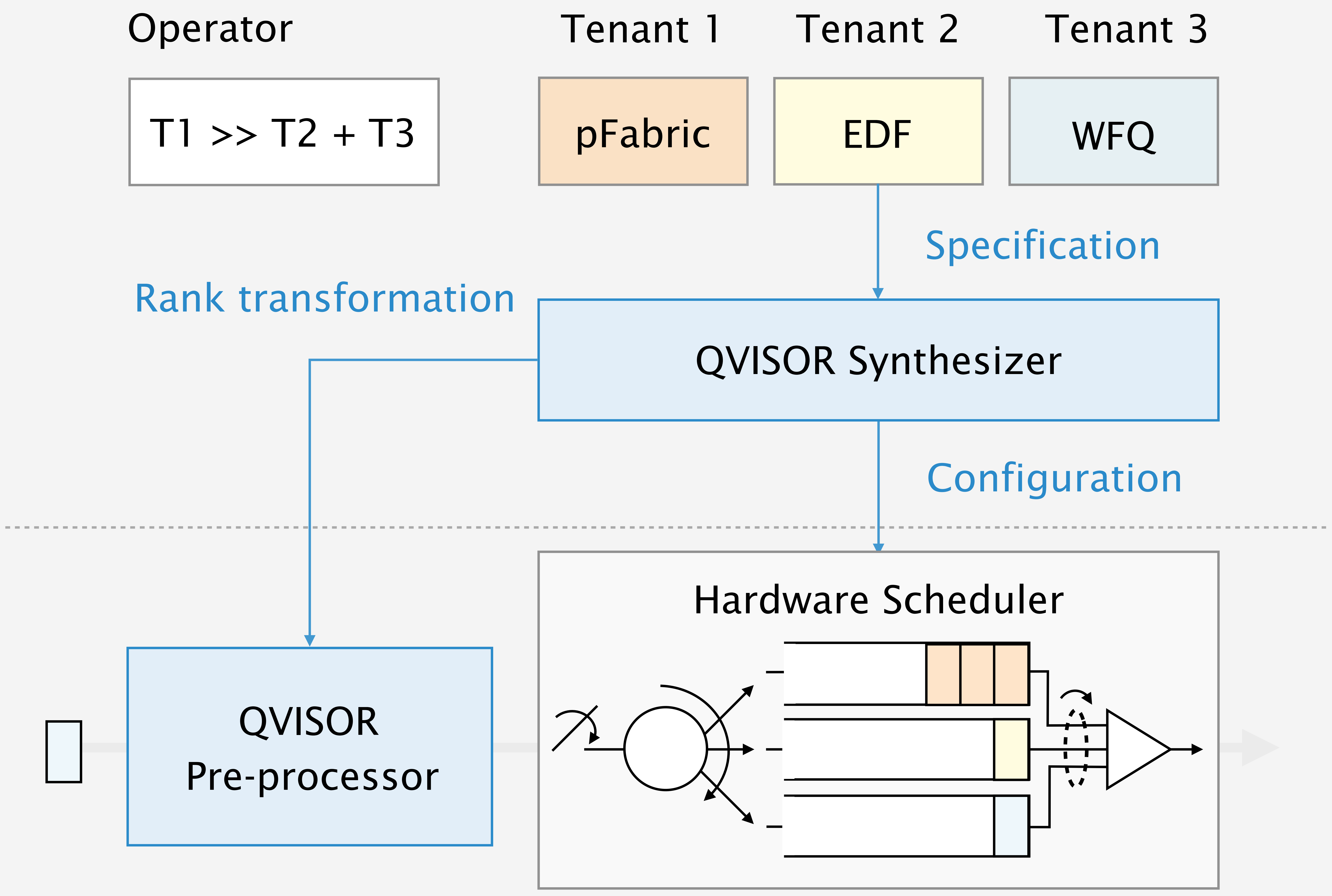
QVISOR's pre-processor applies the transformation functions at line rate in the data plane



QVISOR's pre-processor applies the transformation functions at line rate in the data plane



QVISOR


Operator
T1 >> T2 + T3
Tenant 1
pFabric
Tenant 2
EDF
Tenant 3
WFQ
Specification
Rank transformation
QVISOR Synthesizer
Configuration
Hardware Scheduler
QVISOR
Pre-processor

Check our paper!

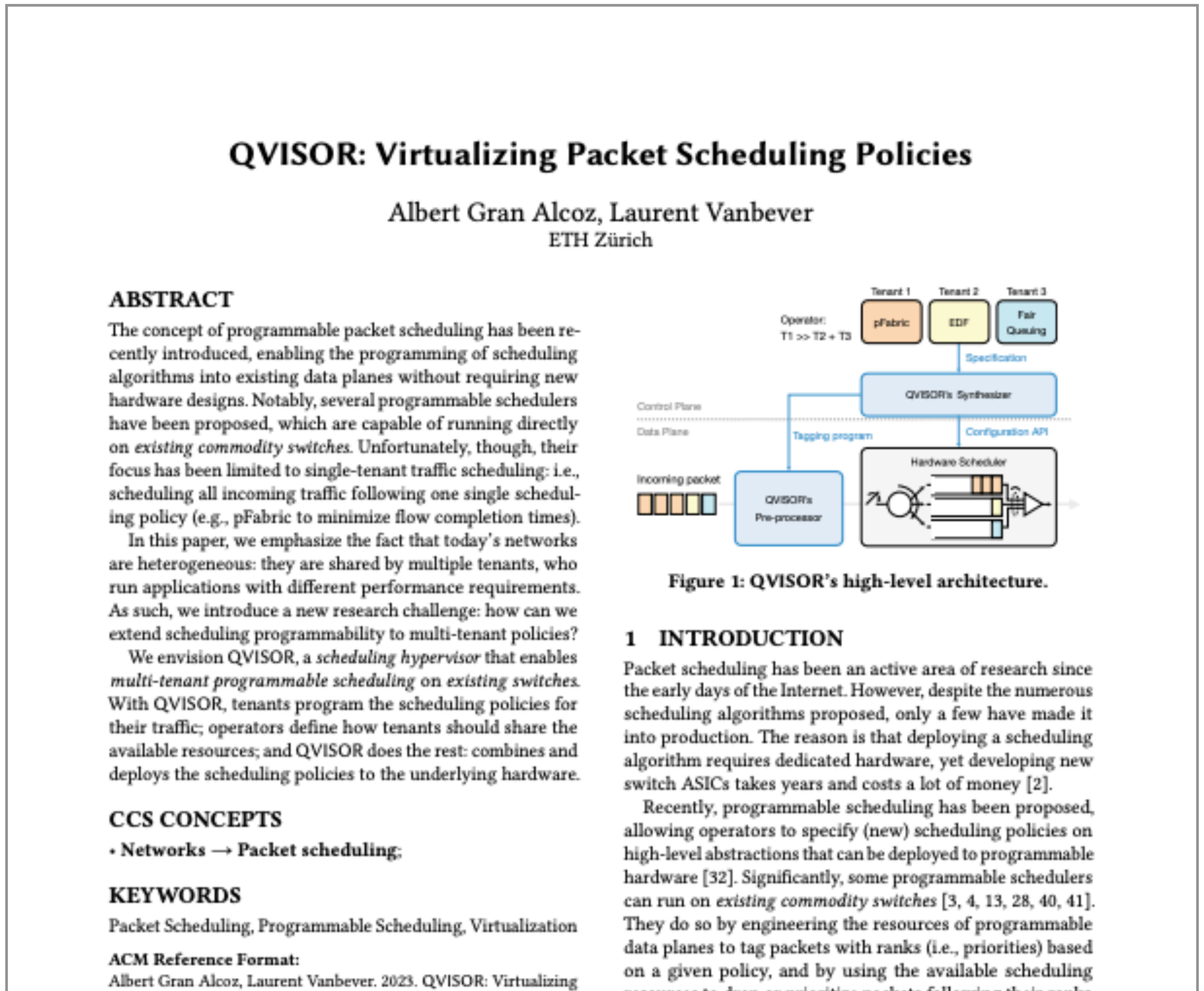
nsg.ee.ethz.ch

Evaluation

How to run on existing devices

Future research

+ more on packet scheduling!



QVISOR: Virtualizing Packet Scheduling Policies

Albert Gran Alcoz, Laurent Vanbever
ETH Zürich

ABSTRACT

The concept of programmable packet scheduling has been recently introduced, enabling the programming of scheduling algorithms into existing data planes without requiring new hardware designs. Notably, several programmable schedulers have been proposed, which are capable of running directly on *existing commodity switches*. Unfortunately, though, their focus has been limited to single-tenant traffic scheduling: i.e., scheduling all incoming traffic following one single scheduling policy (e.g., pFabric to minimize flow completion times).

In this paper, we emphasize the fact that today's networks are heterogeneous: they are shared by multiple tenants, who run applications with different performance requirements. As such, we introduce a new research challenge: how can we extend scheduling programmability to multi-tenant policies?

We envision QVISOR, a *scheduling hypervisor* that enables *multi-tenant programmable scheduling* on *existing switches*. With QVISOR, tenants program the scheduling policies for their traffic; operators define how tenants should share the available resources; and QVISOR does the rest: combines and deploys the scheduling policies to the underlying hardware.

CCS CONCEPTS

• **Networks → Packet scheduling;**

KEYWORDS

Packet Scheduling, Programmable Scheduling, Virtualization

ACM Reference Format:
Albert Gran Alcoz, Laurent Vanbever. 2023. QVISOR: Virtualizing

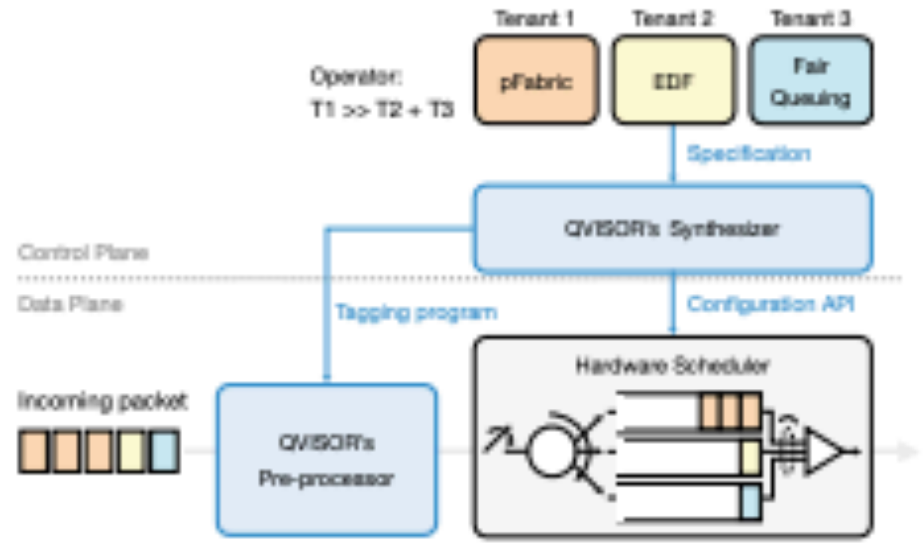


Figure 1: QVISOR's high-level architecture.

1 INTRODUCTION

Packet scheduling has been an active area of research since the early days of the Internet. However, despite the numerous scheduling algorithms proposed, only a few have made it into production. The reason is that deploying a scheduling algorithm requires dedicated hardware, yet developing new switch ASICs takes years and costs a lot of money [2].

Recently, programmable scheduling has been proposed, allowing operators to specify (new) scheduling policies on high-level abstractions that can be deployed to programmable hardware [32]. Significantly, some programmable schedulers can run on *existing commodity switches* [3, 4, 13, 28, 40, 41]. They do so by engineering the resources of programmable data planes to tag packets with ranks (i.e., priorities) based on a given policy, and by using the available scheduling