Eiffel: Efficient and Flexible Software Packet Scheduling

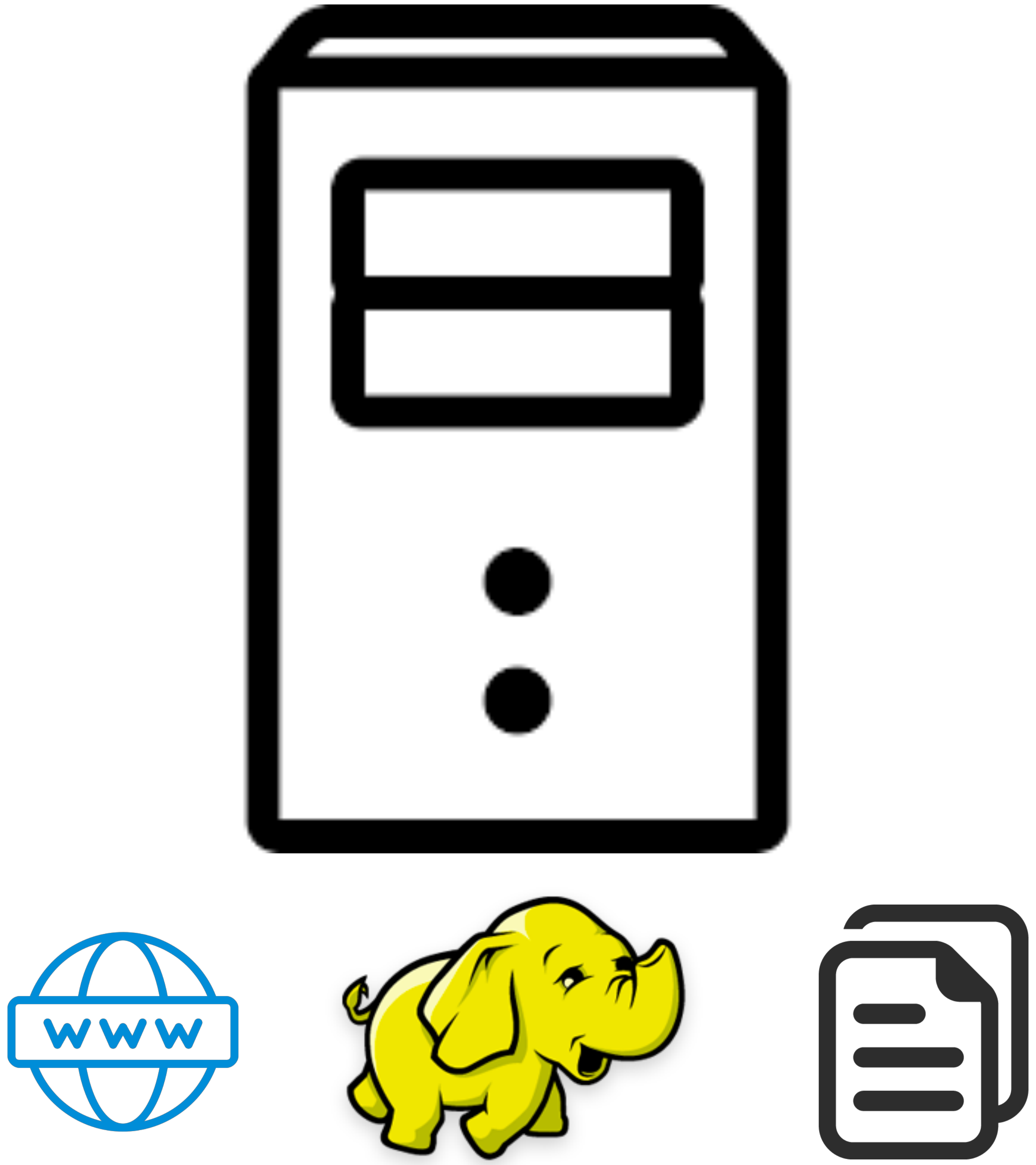
Ahmed Saeed, *Yimeng Zhao, Nandita Dukkipati,*
Mostafa Ammar, Ellen Zegura, Khaled Harras, and Amin Vahdat

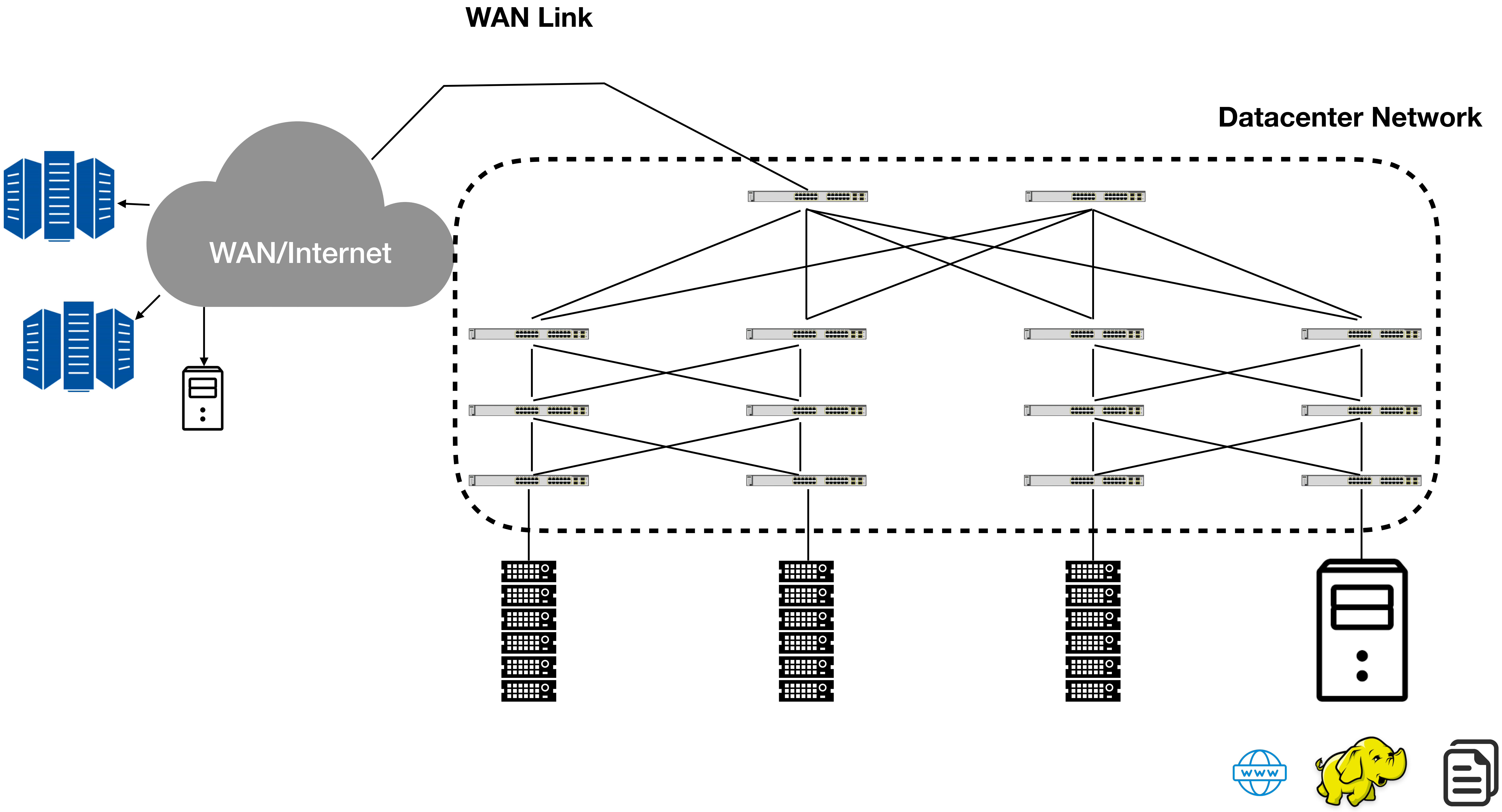
Georgia Tech

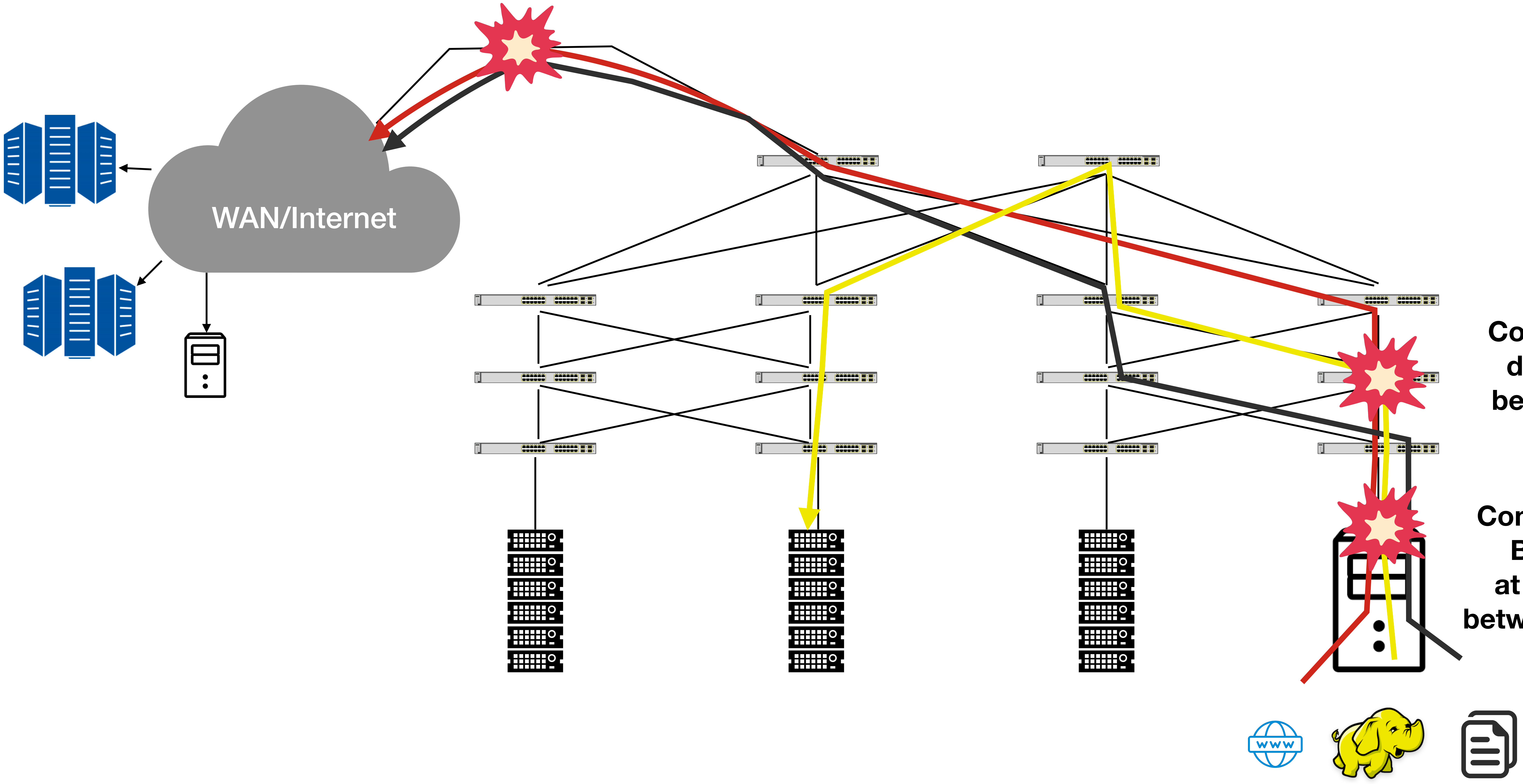
Google

Carnegie Mellon University


Google
google.com/datacenters


www

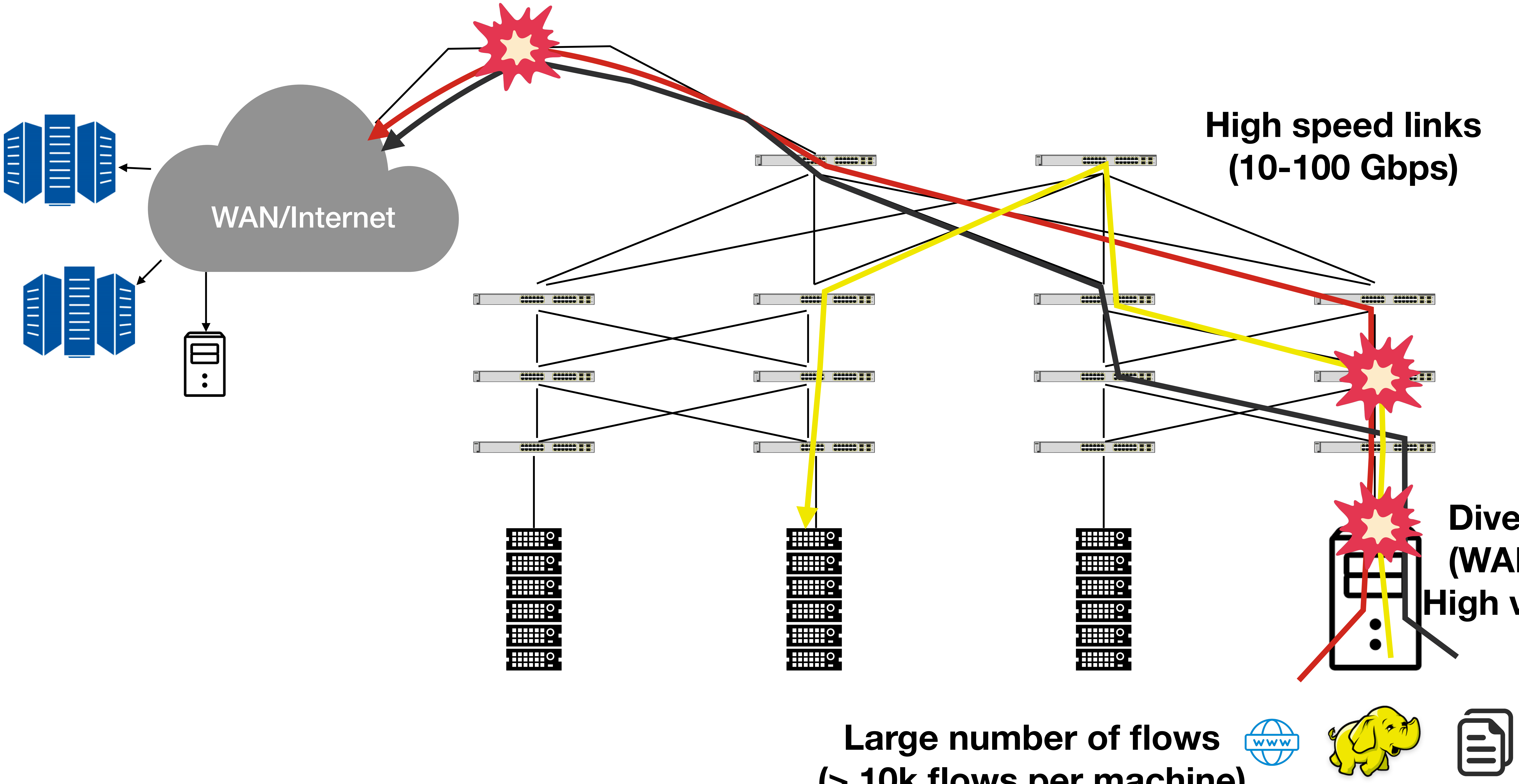

WAN Link
Datacenter Network
WAN/Internet
www



Competition for Bandwidth
at premium links between WAN traffic

Competition in the
datacenter LAN
between all traffic

Competition for
Bandwidth
at the source
between all traffic



High speed links (10-100 Gbps)

Diversity in traffic (WAN vs LAN and High vs Low priority)

Large number of flows (> 10k flows per machine)

Packet Scheduling

- Scheduling determines the relative ordering as well as transmission time of packets in a queuing data structure with respect to some ranking function

- Packet scheduling implements policies to solve such problems
    - Traffic Isolation
    - Flow Completion Time Optimization
    - Congestion Control

pFabric: Minimal Near-Optimal Datacenter Transport

Mohammad Alizadeh†‡, Shuang Yang†, Milad Sharif†, Sachin Katti†,
Nick McKeown†, Balaji Prabhakar†, and Scott Shenker§
†Stanford University ‡Insieme Networks §U.C. Berkeley / ICSI
{alizade, shyang, msharif, skatti, nickm, balaji}@stanford.edu shenker@icsi.berkeley.edu

Information-Agnostic Flow Scheduling for Commodity Data Centers

Wei Bai[1], Li Chen[1], Kai Chen[1], Dongsu Han[2], Chen Tian[3], Hao Wang[1]
[1]*SING Group @ HKUST* [2]*KAIST* [3]*Nanjing Univ.*

Silo: Predictable Message Latency in the Cloud

Keon Jang, Intel Labs, Santa Clara, CA
Justine Sherry*, UC Berkeley, Berkeley, CA
Hitesh Ballani, Microsoft Research, Cambridge, UK
Toby Moncaster*, University of Cambridge, Cambridge, UK

BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing

Alok Kumar, Nikhil Kasinadhuni, Björn Carlin, Sushant Jain, Enrique Cauich Zermeno, Mihai Amarandei-Stavila, Stephen Stuart, Uday Naik, C. Stephen Gunn, Mathieu Robin, Amin Vahdat, Anand Raghuraman, Jing Ai, Aspi Siganporia
Google Inc.
bwe-sigcomm@google.com

ABSTRACT

WAN bandwidth remains a constrained resource that is economically infeasible to substantially overprovision. Hence, it is important to allocate capacity according to service priority and based on the incremental value of additional allocation. For example, it may be the highest priority for one service to receive 10Gb/s of bandwidth but upon reaching such an allocation, incremental priority may drop sharply favoring allocation to other services. Motivated by the observation that individual flows with fixed priority may not be the ideal basis for bandwidth allocation, we present the design and implementation of Bandwidth Enforcer (BwE), a global, hierarchical bandwidth allocation infrastructure. BwE supports: i) service-level bandwidth allocation following prioritized bandwidth functions where a service can represent an arbitrary collection of flows, ii) independent allocation and delegation policies according to user-defined hierarchy, all accounting for a global view of bandwidth and failure conditions, iii) multi-path forwarding common in traffic-engineered networks, and iv) a central administrative point to override (perhaps faulty) policy during exceptional conditions. BwE has delivered more service-efficient bandwidth utilization and simpler management in production for multiple years.

Keywords

Bandwidth Allocation; Wide-Area Networks; Software-Defined Network; Max-Min Fair

1. INTRODUCTION

TCP-based bandwidth allocation to individual flows contending for bandwidth on bottleneck links has served the Internet well for decades. However, this model of bandwidth allocation assumes all flows are of equal priority and that all flows benefit equally from any incremental share of available bandwidth. It implicitly assumes a client-server communication model where a TCP flow captures the communication needs of an application communicating across the Internet.

This paper re-examines bandwidth allocation for an important, emerging trend, distributed computing running across dedicated private WANs in support of cloud computing and service providers. Thousands of simultaneous such applications run across multiple global data centers, with thousands of processes in each data center, each potentially maintaining thousands of individual active connections to remote servers. WAN traffic engineering means that site-pair communication follows different network paths, each with different bottlenecks. Individual services have vastly differ-

Queues don't matter when you can JUMP them!

Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand†, Jon Crowcroft
University of Cambridge Computer Laboratory
† now at Google, Inc.

Friends, not Foes – Synthesizing Existing Transport Strategies for Data Center Networks

Ali Munir[1]; Ghufran Baig[2], Syed M. Irteza[2], Ihsan A. Qazi[2],
Alex X. Liu[1], Fahad R. Dogar[3]
[1]Michigan State University, [2]LUMS, [3]Microsoft Research
{munirali, alexliu}@cse.msu.edu, {ghufran.baig, syed.irteza, ihsan.qazi}@lums.edu.pk,
fdogar@microsoft.com

Less is More: Trading a little Bandwidth for Ultra-Low Latency in the Data Center

Mohammad Alizadeh, Abdul Kabbani†, Tom Edsall⋆, Balaji Prabhakar,
Amin Vahdat†§, and Masato Yasuda¶
Stanford University †Google ⋆Cisco Systems §U.C. San Diego ¶NEC Corporation, Japan

End-to-end Performance Isolation through Virtual Datacenters

Sebastian Angel⋆, Hitesh Ballani, Thomas Karagiannis, Greg O'Shea, Eno Thereska
Microsoft Research ⋆*The University of Texas at Austin*

Abstract

The lack of performance isolation in multi-tenant datacenters at appliances like middleboxes and storage servers results in volatile application performance. To insulate tenants, we propose giving them the abstraction of a dedicated *virtual datacenter* (VDC). VDCs encapsulate end-to-end throughput guarantees—specified in a new metric based on virtual request cost—that hold across distributed appliances and the intervening network.

We present *Pulsar*, a system that offers tenants their own VDCs. Pulsar comprises a logically centralized controller that uses new mechanisms to estimate tenants' demands and appliance capacities, and allocates datacenter resources based on flexible policies. These allocations are enforced at end-host hypervisors through multi-resource token buckets that ensure tenants with changing workloads cannot affect others. Pulsar's design does not require changes to applications, guest OSes, or appliances. Through a prototype deployed across 113 VMs, three appliances, and a 40 Gbps network, we show that Pulsar enforces tenants' VDCs while imposing overheads of less than 2% at the data and control plane.

ter. A VDC is composed of virtual machines (VMs), and resources like virtual appliances and a virtual network that are associated with throughput guarantees. These guarantees are independent of tenants' workloads, hold across all VDC resources, and are therefore end-to-end.

Providing the VDC abstraction to tenants presents two main challenges. First, tenants can be bottlenecked at different appliances or network links, and changing workloads can cause these bottlenecks to shift over time (§2.1). Second, resources consumed by a request at an appliance can vary based on request characteristics (type, size, etc.), appliance internals, and simultaneous requests being serviced. For example, an SSD-backed filestore appliance takes disproportionately longer to serve WRITE requests than READ requests (§2.4). This behavior has two implications: (i) the *capacity*, or maximum achievable throughput, of an appliance varies depending on the workload. This is problematic because the amount of appliance resources that can be allocated to tenants becomes a moving target. (ii) Standard metrics for quantifying throughput, like requests/second or bits/second, become inadequate. For example, offering throughput guarantees in request/second, irrespective of

Better Never than Late: Meeting Deadlines in Datacenter Networks

Christo Wilson christowilson@umail.ucsb.edu
Hitesh Ballani hiballan@microsoft.com
Thomas Karagiannis thomkar@microsoft.com
Ant Rowstron antr@microsoft.com
Microsoft Research
Cambridge, UK

Finishing Flows Quickly with Preemptive Scheduling

Chi-Yao Hong, UIUC, cyhong@illinois.edu
Matthew Caesar, UIUC, caesar@illinois.edu
P. Brighten Godfrey, UIUC, pbg@illinois.edu

Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities

Behnam Montazeri, Yilong Li, Mohammad Alizadeh†, and John Ousterhout
Stanford University, †MIT

ABSTRACT

Homa is a new transport protocol for datacenter networks. It provides exceptionally low latency, especially for workloads with a high volume of very short messages, and it also supports large and high network utilization. Homa uses in-network

these conditions, so the latency they provide for short messages is far higher than the hardware potential, particularly under high network loads.

Recent years have seen numerous proposals for better transport protocols, including improvements to TCP [2, 3, 31] and

EyeQ: Practical Network Performance Isolation at the Edge

Vimalkumar Jeyakumar[1], Mohammad Alizadeh[1,2], David Mazières[1], Balaji Prabhakar[1],
Changhoon Kim[3], and Albert Greenberg[3]
[1]*Stanford University* [2]*Insieme Networks* [3]*Windows Azure*

Abstract

The datacenter network is shared among untrusted tenants in a public cloud, and hundreds of services in a private cloud. Today we lack fine-grained control over network bandwidth partitioning across tenants. In this paper we present EyeQ, a simple and practical system that provides tenants with bandwidth guarantees as if their endpoints were connected to a dedicated switch. To realize this goal, EyeQ leverages the high bisection bandwidth in a datacenter fabric and enforces admission control on traffic, regardless of the tenant transport protocol. We show that this pushes bandwidth contention to the network's edge, enabling EyeQ to support end-to-end minimum bandwidth guarantees to tenant endpoints in a simple and scalable manner at the servers. EyeQ requires no changes to applications and is deployable with support from the network available today. We evaluate EyeQ with an efficient software implementation at 10Gb/s speeds using unmodified applications and adversarial traffic patterns. Our evaluation demonstrates

Cloud" [1] with their IP addresses without interfering with other tenants. In the data plane, there has been little comparable progress.

To make comparable progress, we posit that the provider should present a simple performance abstraction of a dedicated switch connecting a tenant's endpoints [3], independent of the underlying physical topology. The endpoints may be anywhere in the datacenter, but a tenant should be able to attain full line rate for any traffic pattern between its endpoints, constrained only by endpoint capacities. Bandwidth assurances to this tenant should suffer no negative impact from the behavior and churn of other tenants in the datacenter. This abstraction has been a consistent ask of enterprise customers considering moving to the cloud, as the enterprise mission demands a high degree of infrastructure predictability [4].

Is this abstraction realizable? EyeQ described in this paper attempts to deliver this abstraction for every tenant. This requires three key components of which EyeQ provides the final missing piece.

Scheduler Implementation

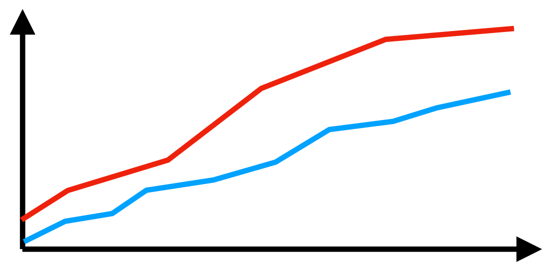
- Hardware schedulers in ASICs, FPGAs, or NPUs
  - Preprogrammed policies in switches or NICs
  - Programmable schedulers
- Software schedulers at end hosts or middleboxes
  - Kernel Queuing Disciplines (Qdiscs)
  - Userspace networking stacks

P4.org

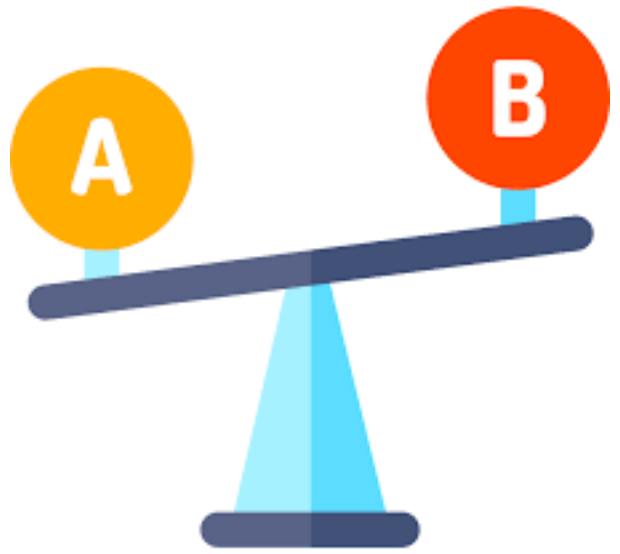
DPDK
DATA PLANE DEVELOPMENT KIT

Software vs Hardware

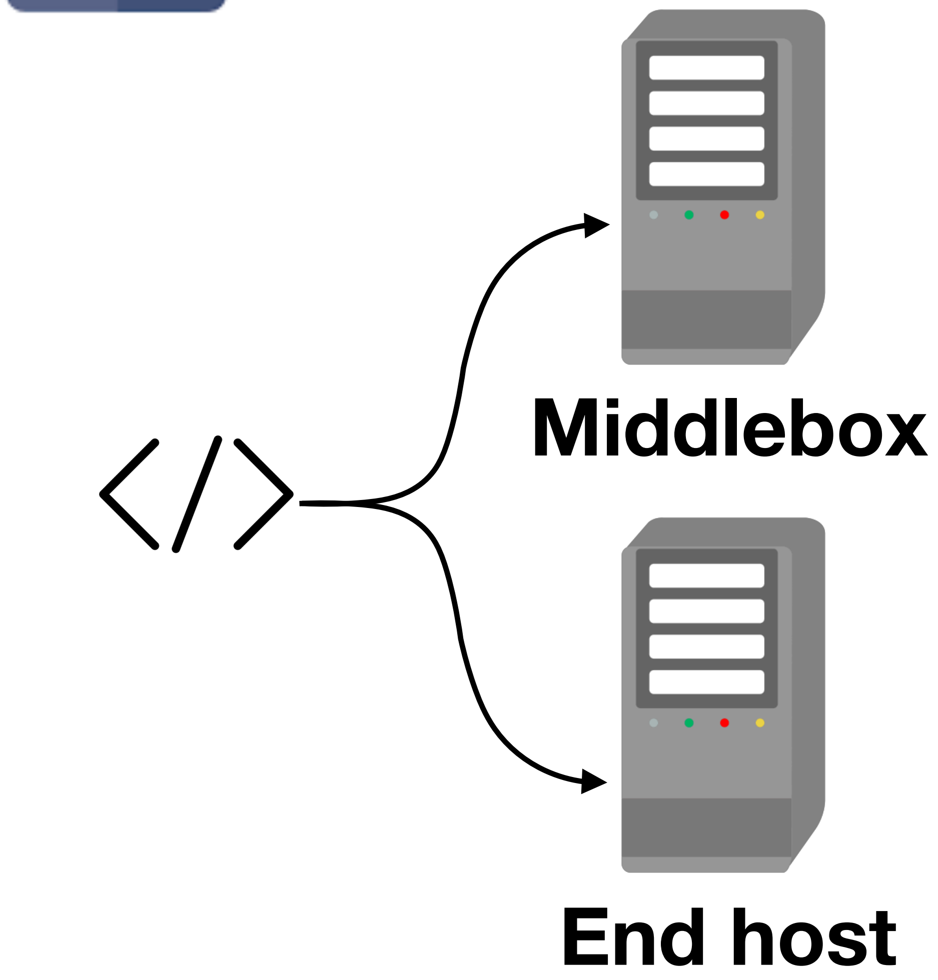
- Hardware lags behind network needs



- Software serves as a good experimental environment before hardware deployment



- Software provides a "build once, deploy many"

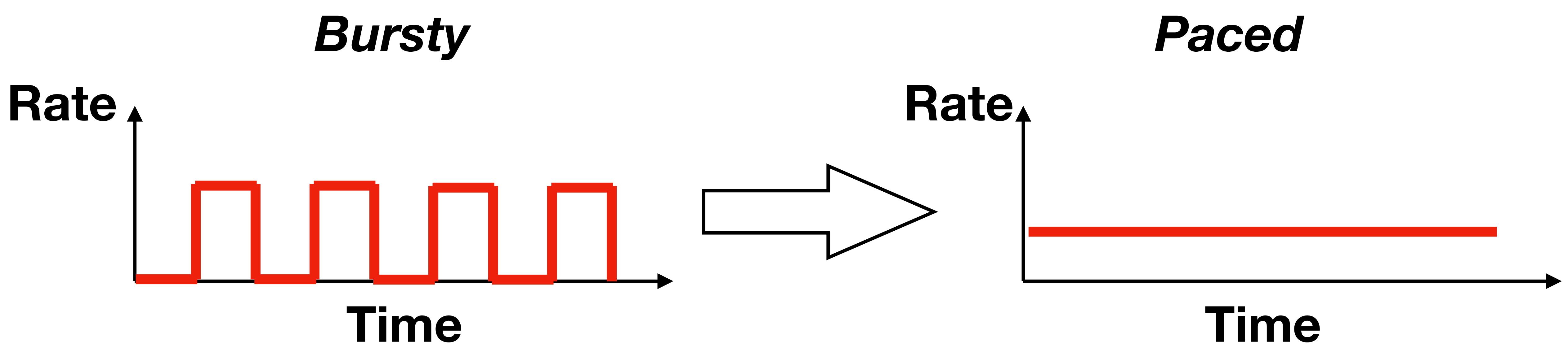


Middlebox

End host

Challenges of Network Scheduling

- Accurate scheduling



- Efficient CPU and memory implementation

O(log(n)) → ***O(1)***

- Diversity of requirements

Hierarchical Weighted Fair Queuing

Strict Priority

Rate Limiting

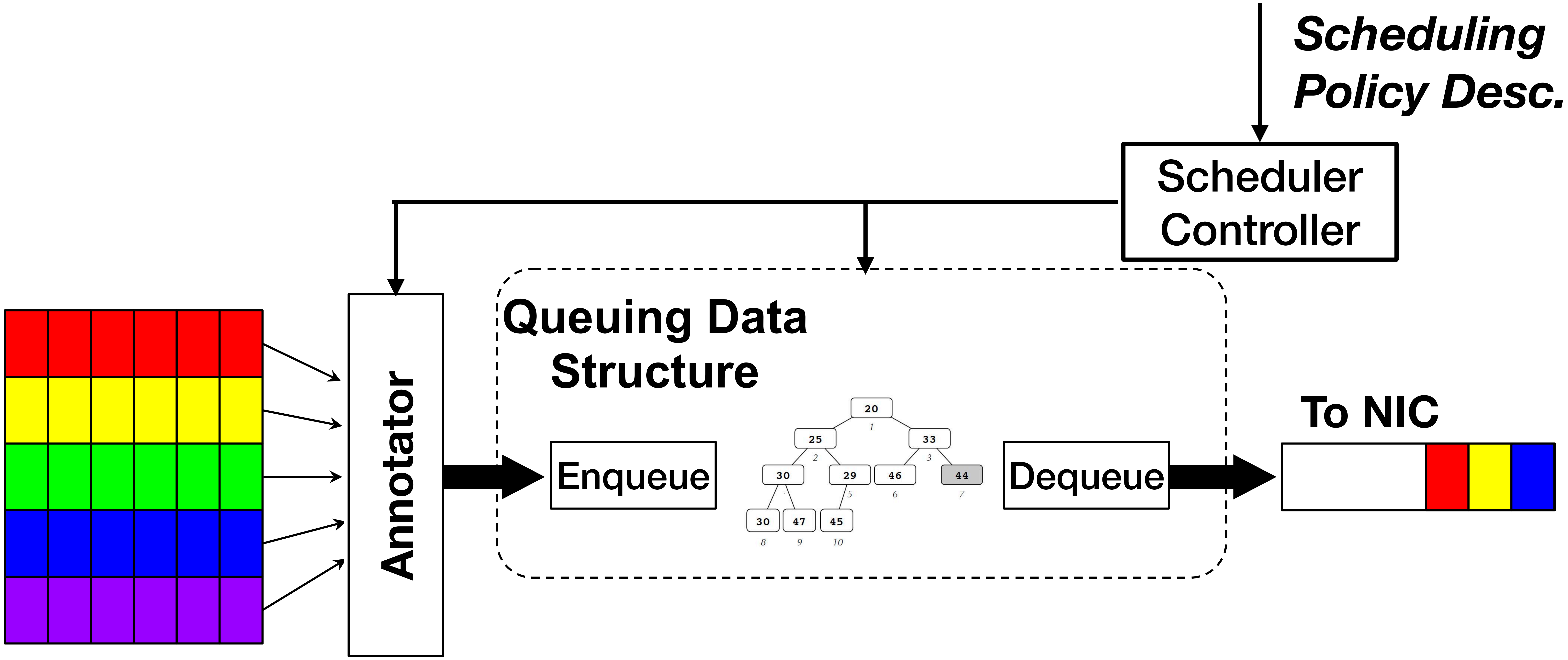
Shortest Remaining Time First

Objective: *Design an accurate, efficient, and programmable software scheduler*

Outline

- Eiffel Overview
- Characteristics of Packet Ranks
- Efficient Packet Ordering: Integer Priority Queues
- Scheduler Programmability
- Evaluation

Eiffel Overview



Scheduling Policy Desc.

Scheduler Controller

Annotator

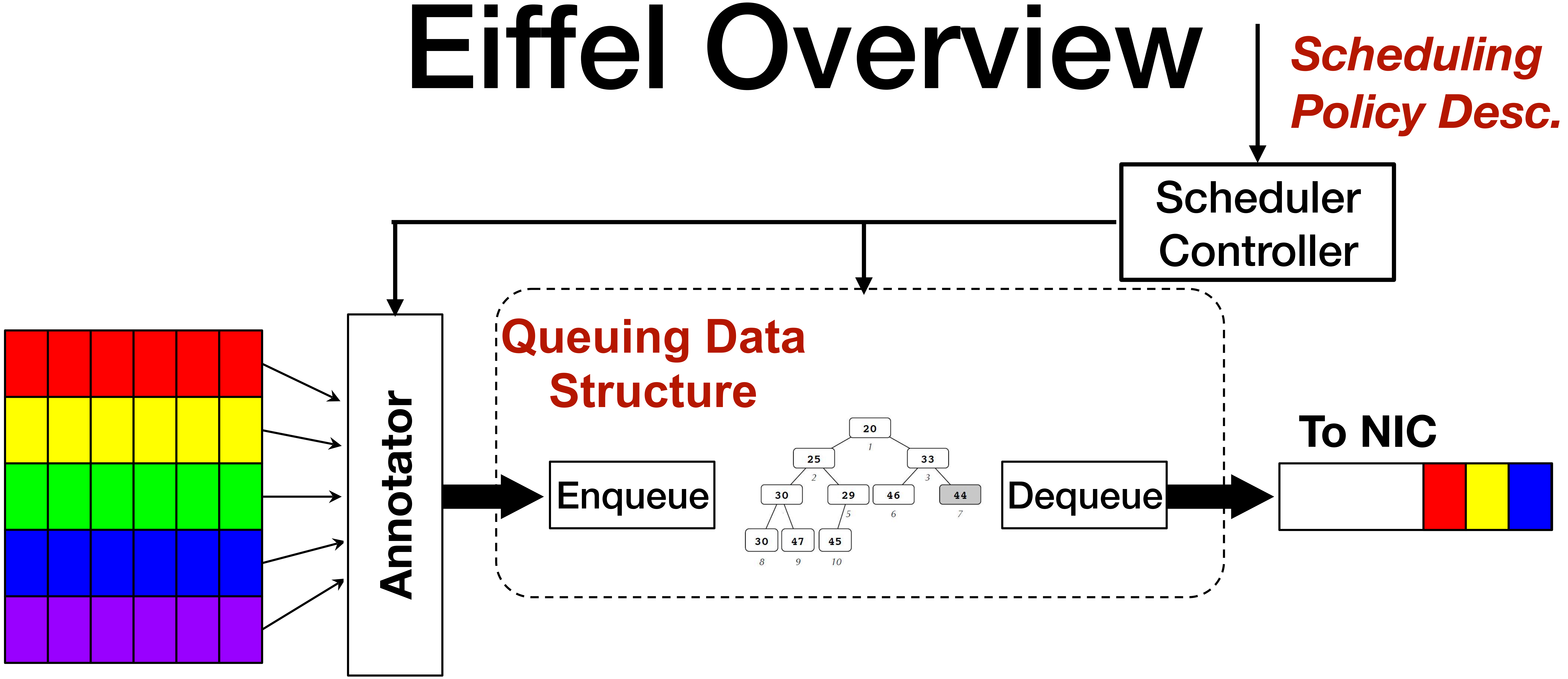
Queuing Data Structure

Enqueue

Dequeue

To NIC

Eiffel Overview


Scheduling Policy Desc.

Scheduler Controller

Annotator

Queuing Data Structure

Enqueue

Dequeue

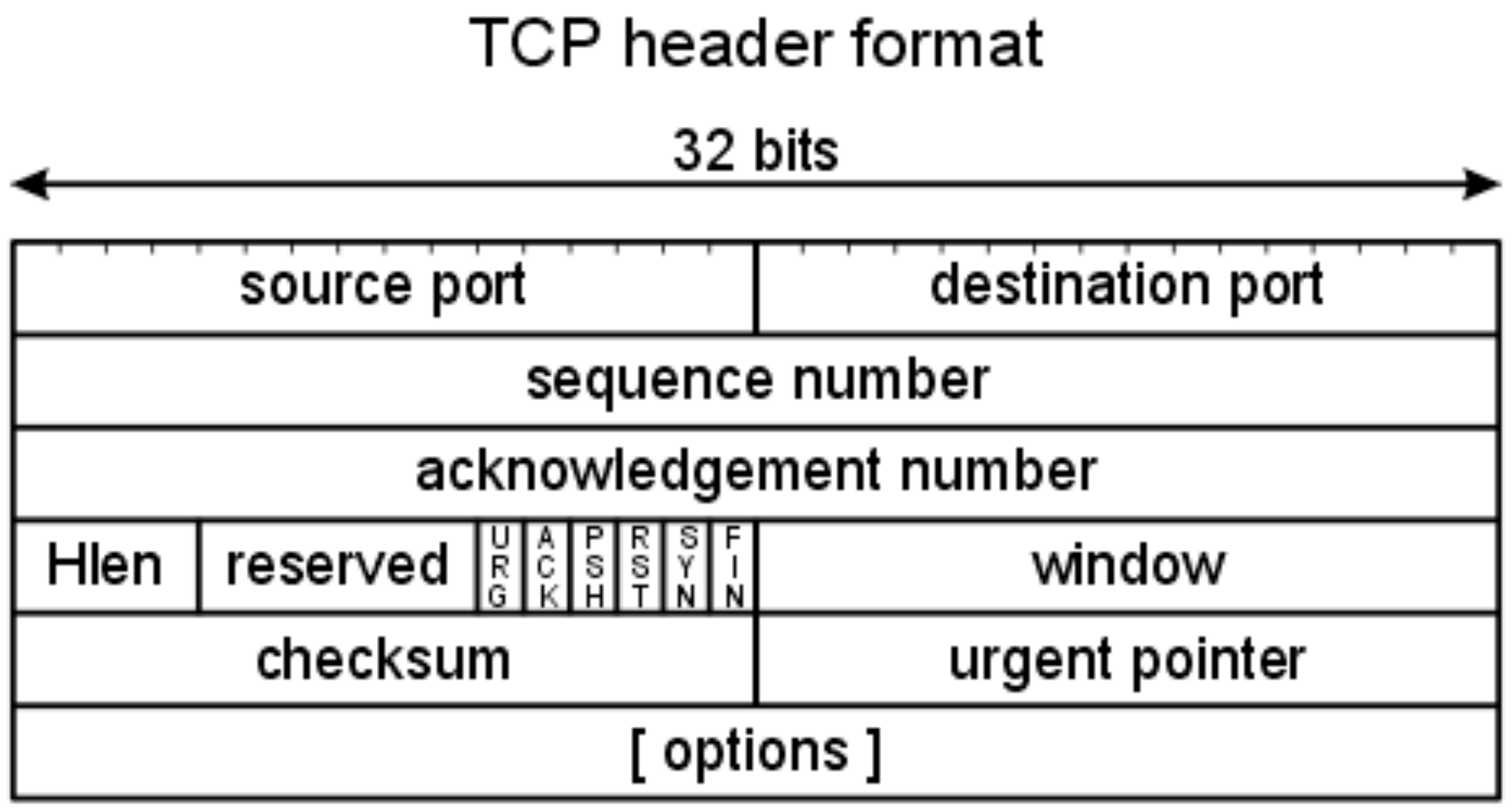
To NIC

- Efficient building block for packet sorting operating at line rate
- Expressive abstraction that can capture a wide range of policies

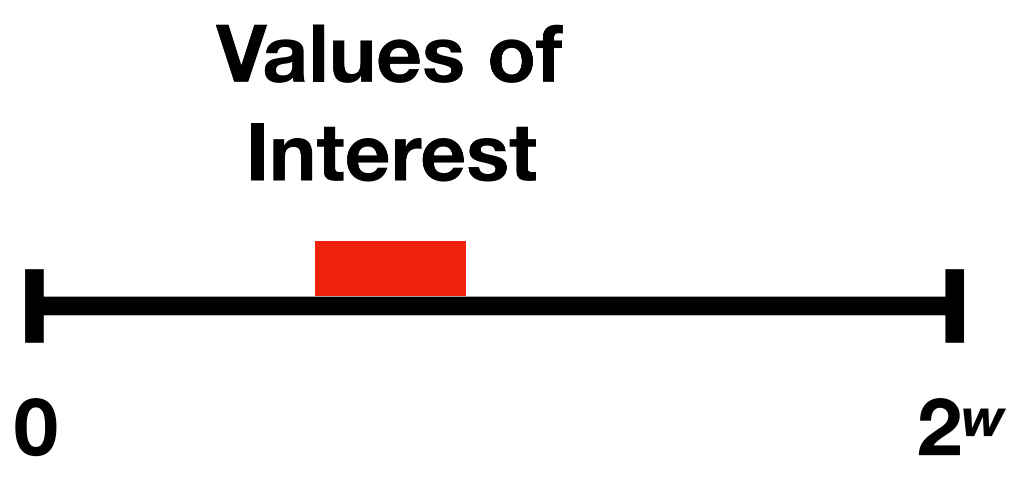
Characteristics of Packet Ranks

Ranks are Integers

- Packet carry limited precision integer priorities of width w bits
  - QoS-based priority
  - Time-based priority
  - Flow size-based priority

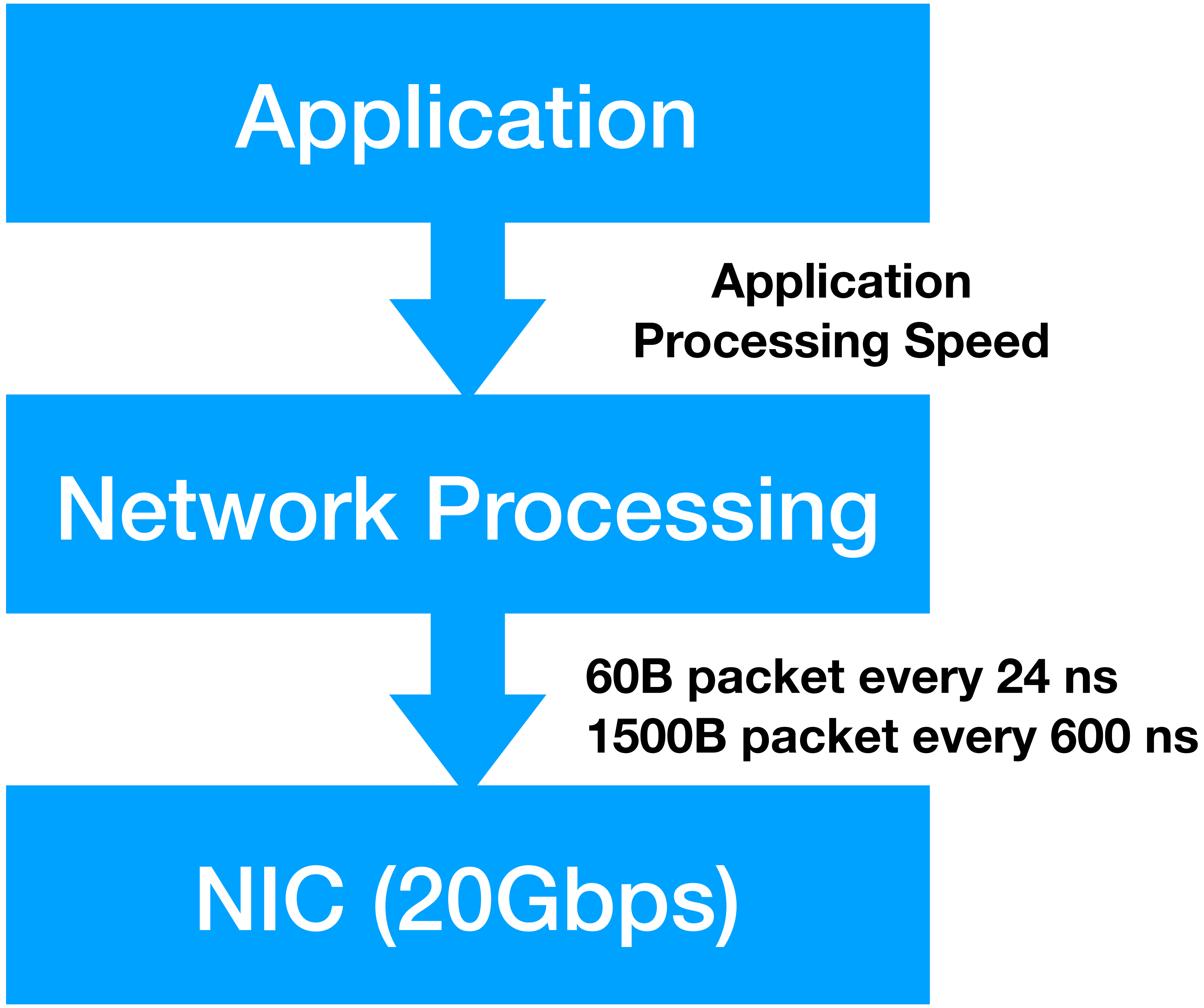


Ranks have Known Ranges



- Semantics of priority values typical have limited ranges within the whole range of integer representation
  - Time-based priorities: from now to a few seconds in the future
  - Flow size-based priorities: values are known from typical application behavior
  - Strict priority ranges: policy/network operator defined

Packets are Processed in Batches



Application

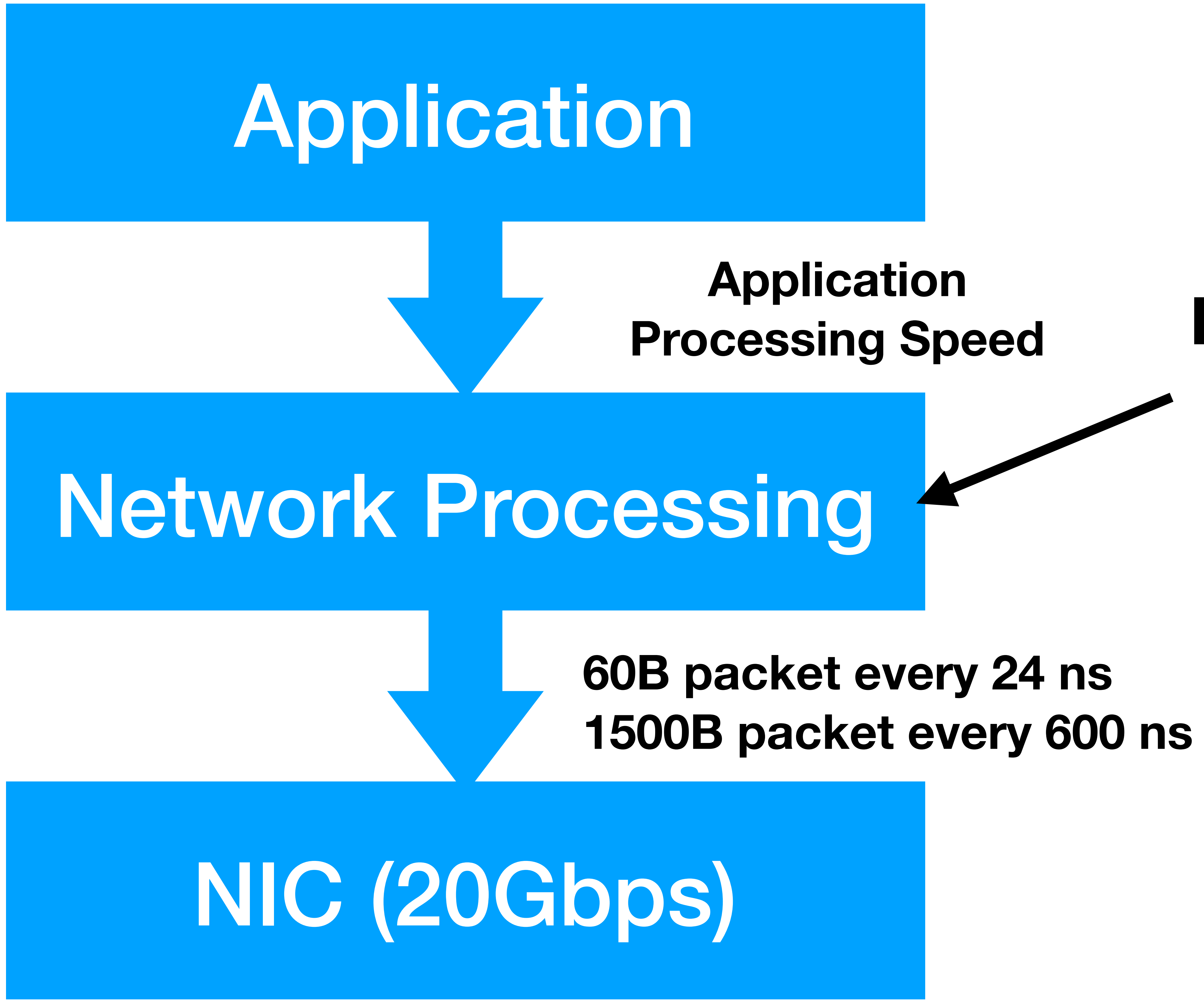
Application
Processing Speed

Network Processing

60B packet every 24 ns
1500B packet every 600 ns

NIC (20Gbps)

Packets are Processed in Batches

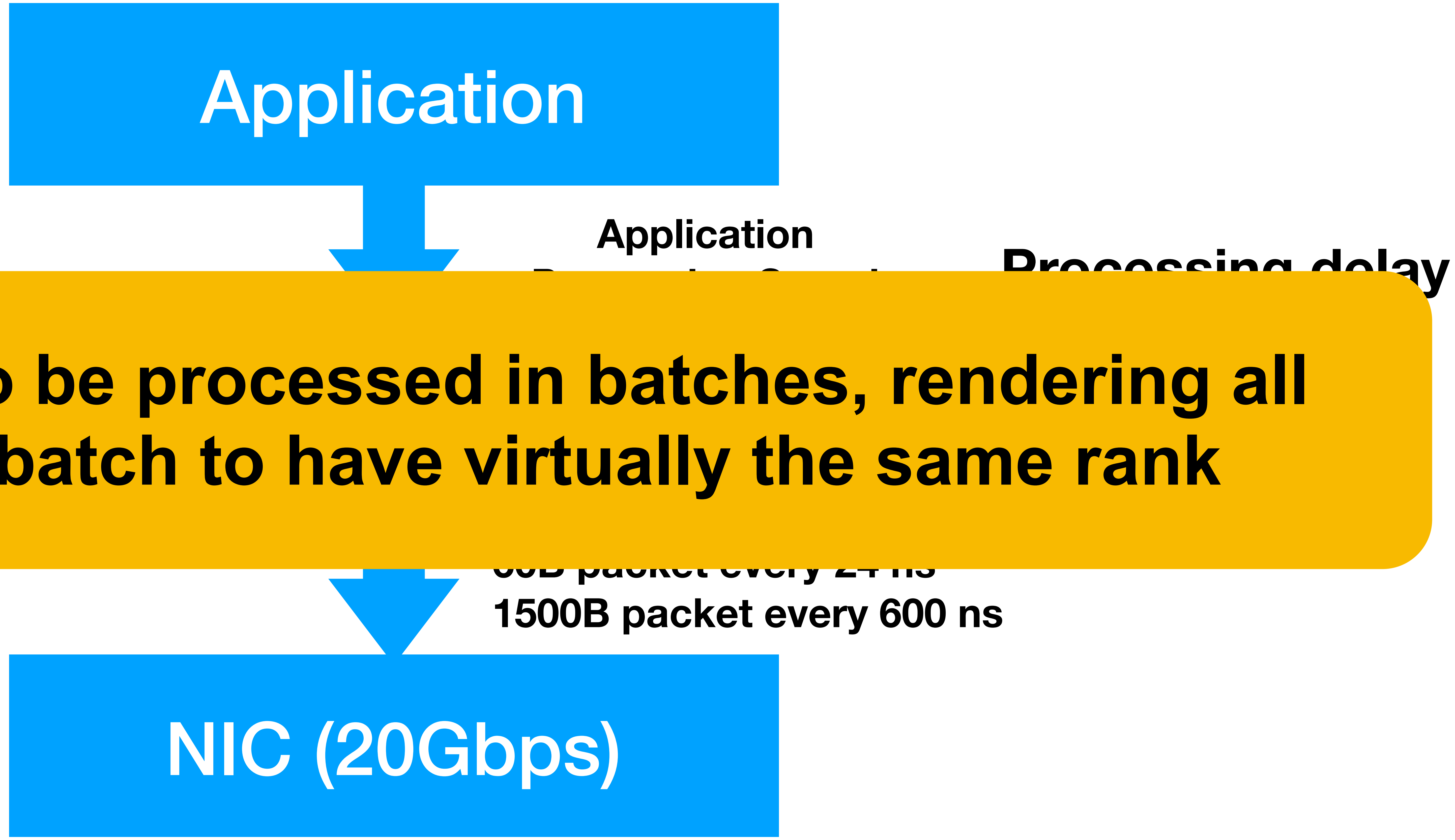


Application
Processing Speed

Processing delay
is in 100s of ns

60B packet every 24 ns
1500B packet every 600 ns

Packets are Processed in Batches



Packets have to be processed in batches, rendering all packets in a batch to have virtually the same rank

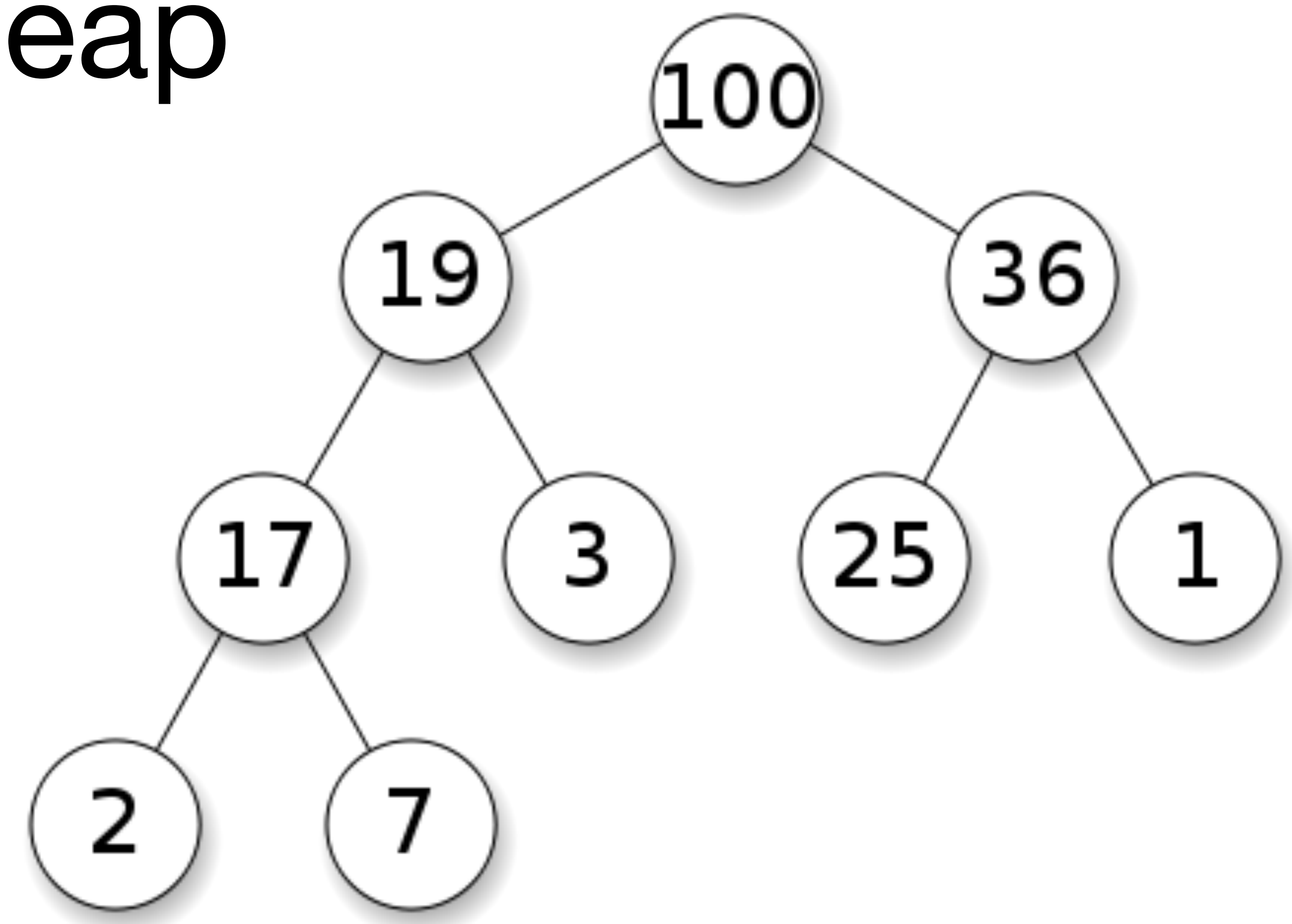
Eiffel Building Block

Bucketed Data Structure + **Limited number of buckets** + **Algorithm to find min/max non-empty bucket**

= **Integer Priority Queues**

Efficient Packet Ordering: Integer Priority Queues

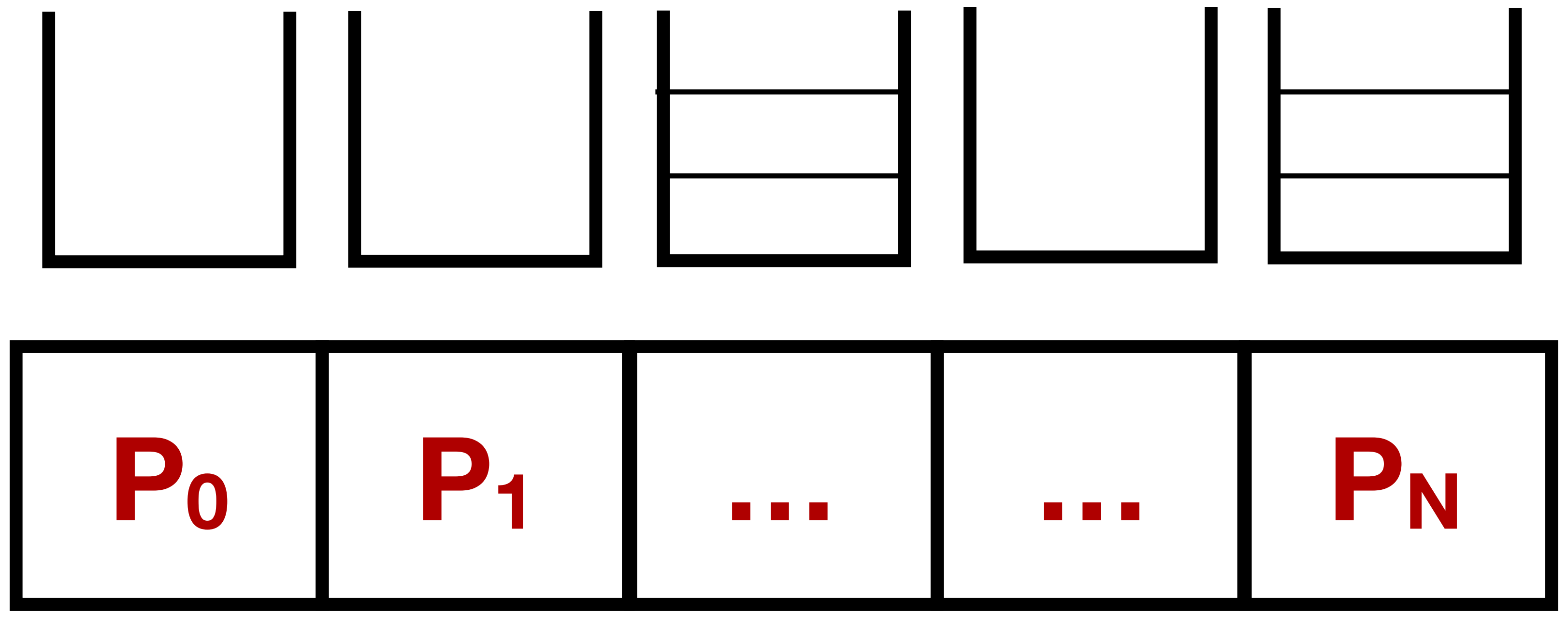
Priority Queues 101



- Binary trees, Binomial Heap, Fibonacci Heap
- Support ExtractMin/ExtractMax
- Overhead of O(log n) on insertion or extraction
- Requires definition of a comparison operator: **Comparison-based Priority Queues**

Integer Priority Queue

- Bucketed queues of N buckets
- Bucket index is the priority of elements in the bucket
- O(1) insertion and change priority
- $O(\mathrm{Log}_{w} N)$ ExtractMin/ExtractMax

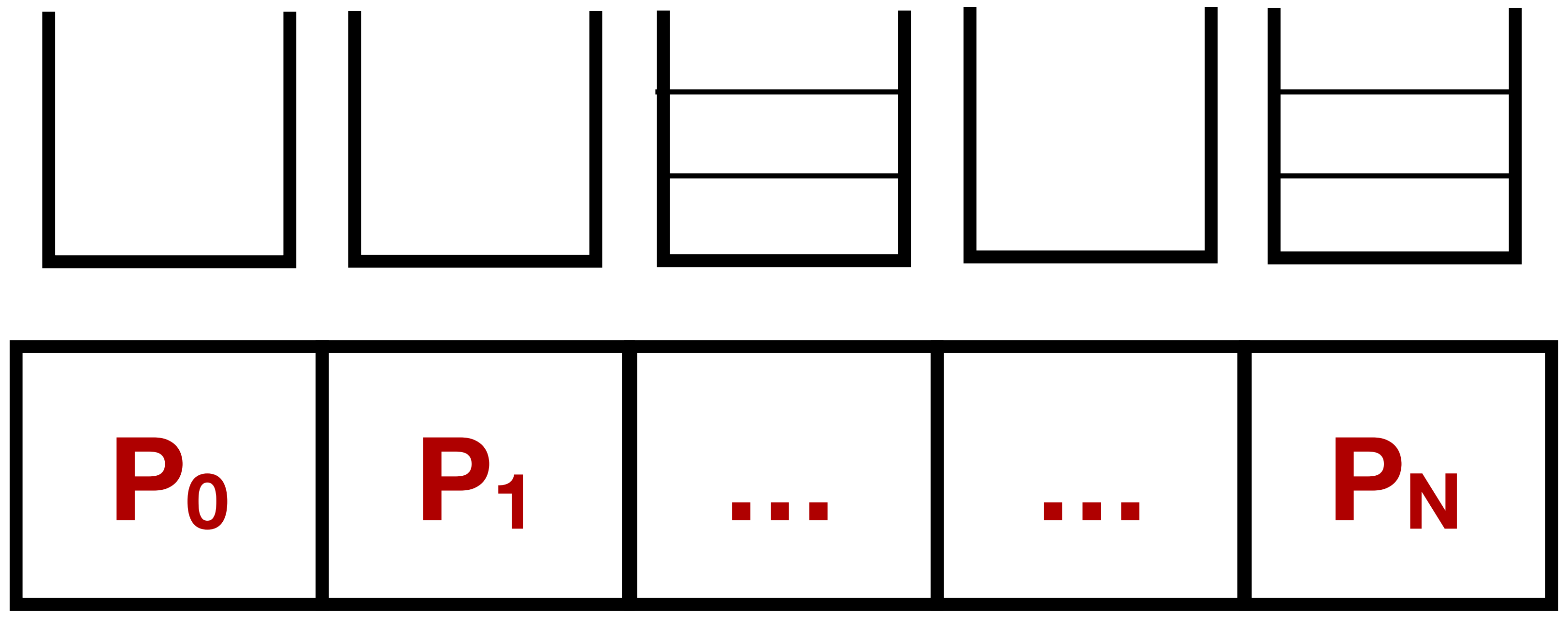


Integer Priority Queue

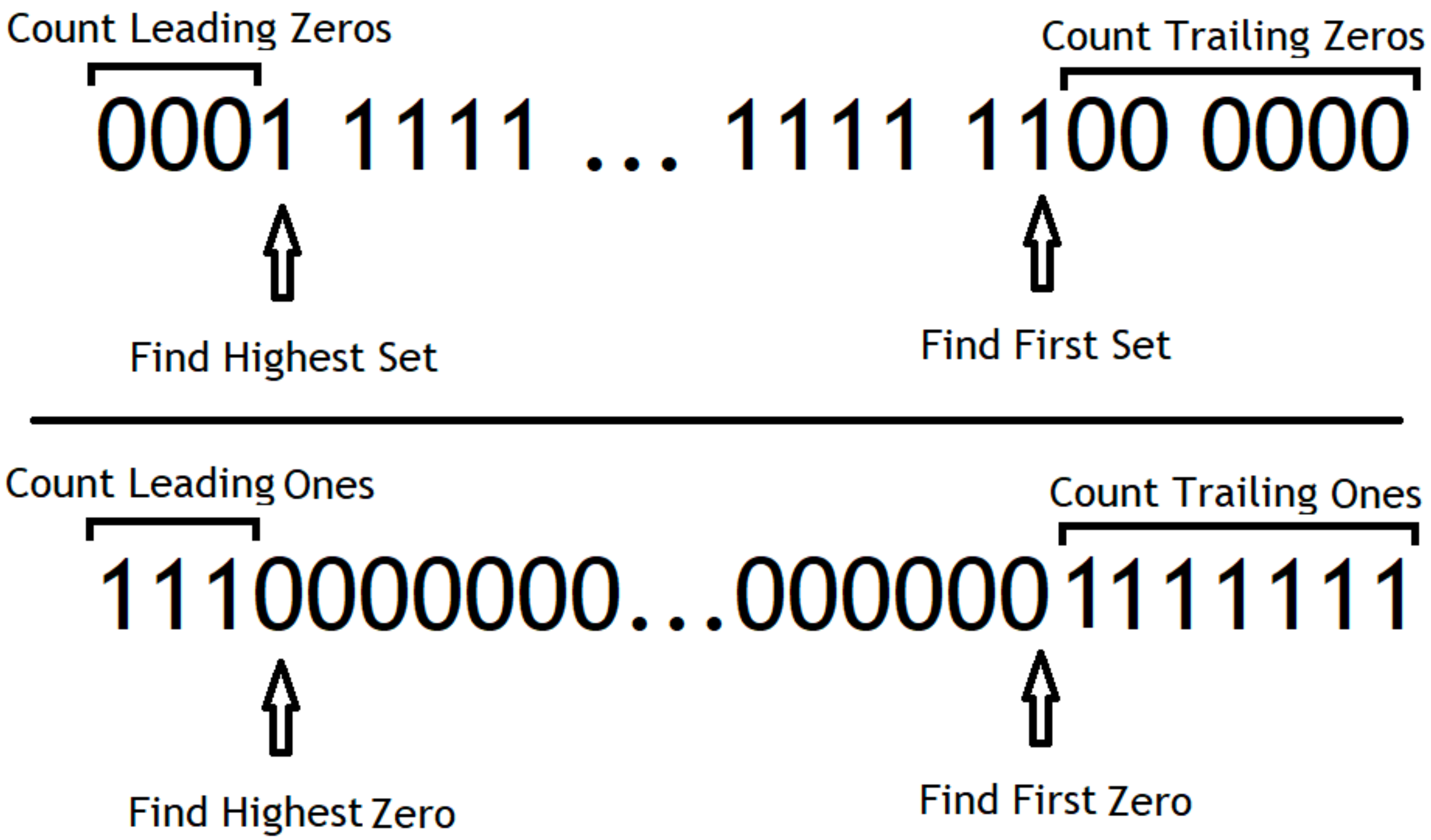
Packets have known priority range and can be grouped into coarse granularity buckets

- Bucketed queues of N buckets
- Bucket index is the priority of elements in the bucket
- O(1) insertion and change priority
- O(Log$_w$N) ExtractMin/ExtractMax

Packets have integer priority are captured in limited precision integers



FFS-based Integer Priority Queue



Count Leading Zeros

Count Trailing Zeros

0001 1111 … 1111 1100 0000

Find Highest Set

Find First Set

Count Leading Ones

Count Trailing Ones

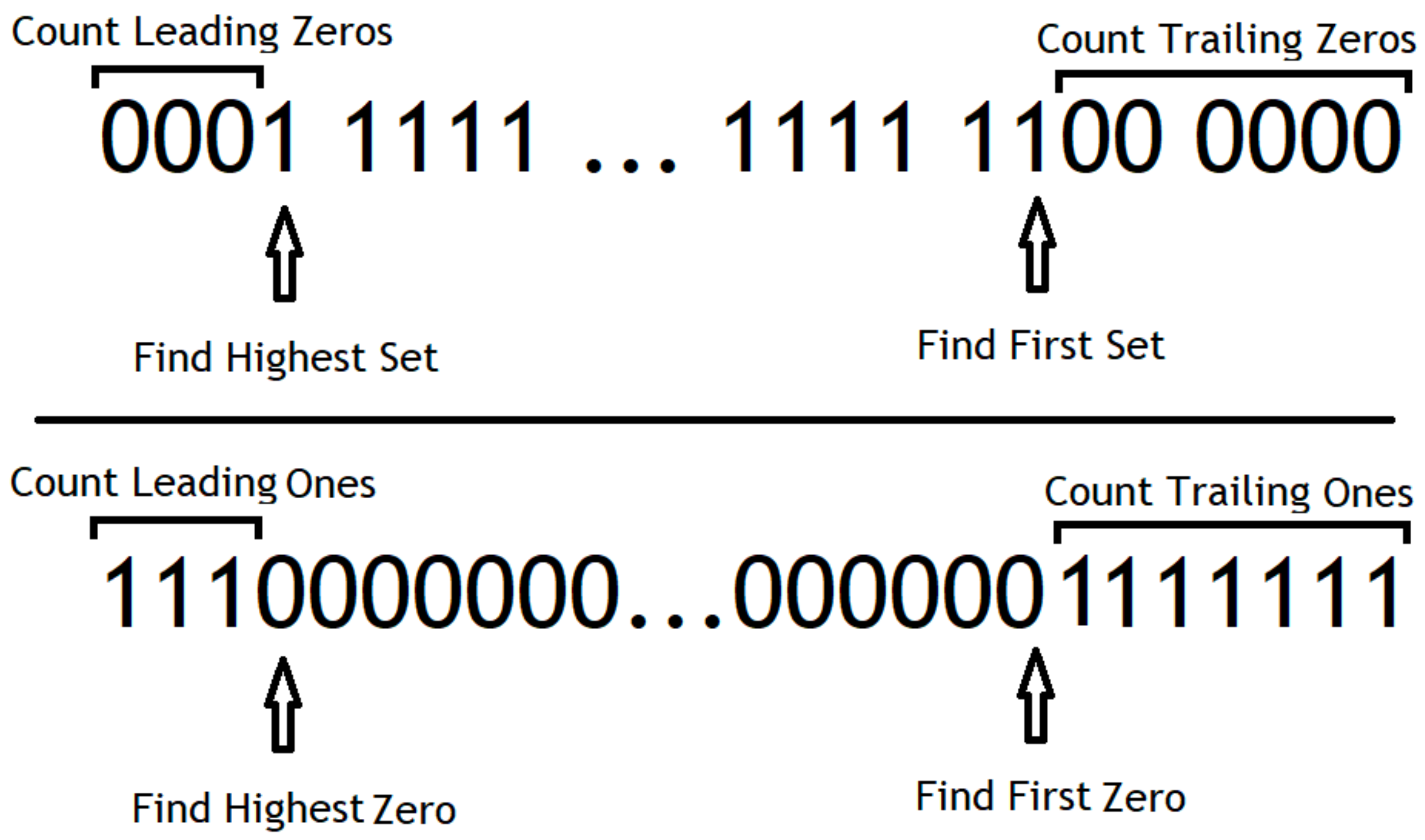
1110000000…0000001111111

Find Highest Zero

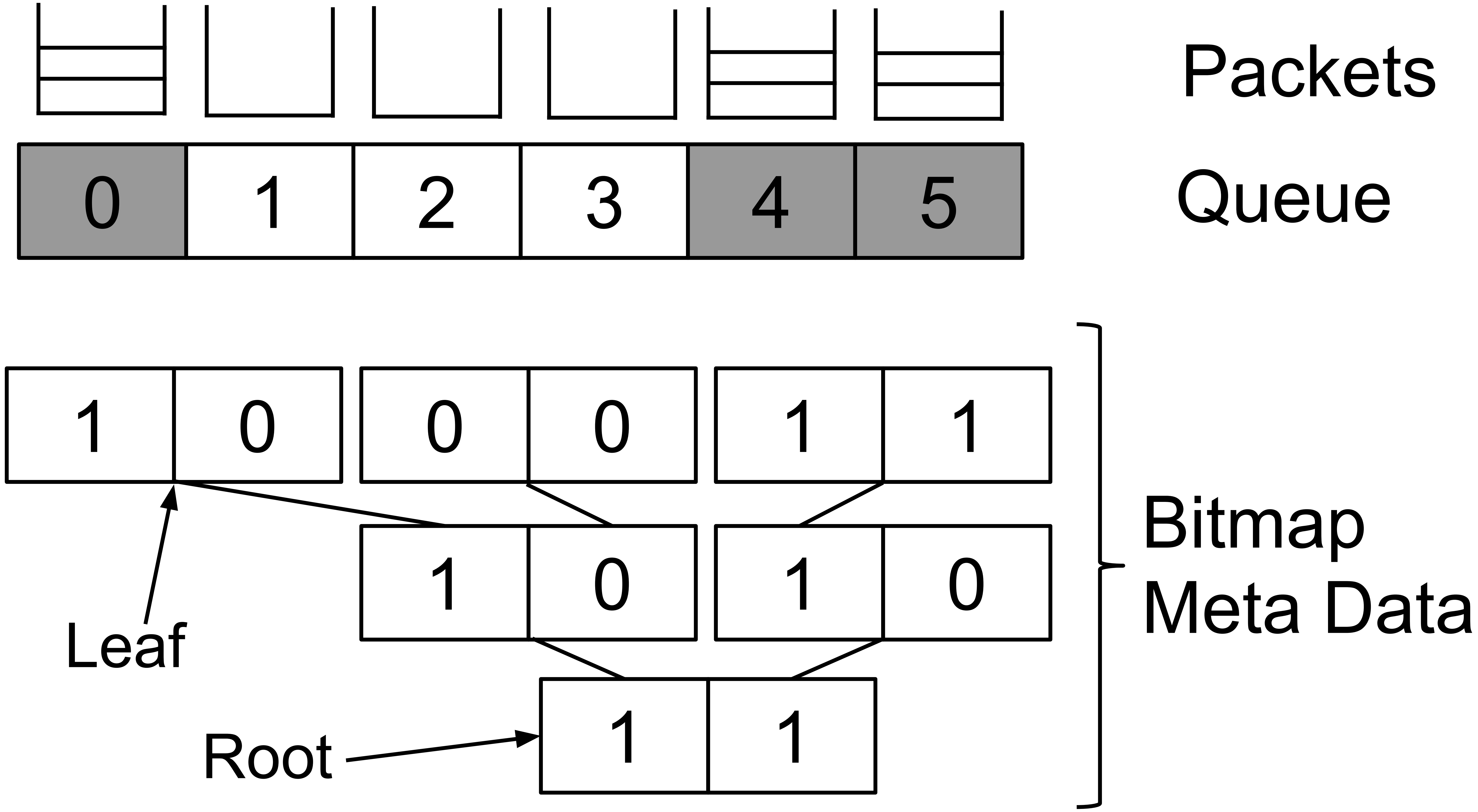
Find First Zero

FFS-based Integer Priority Queue

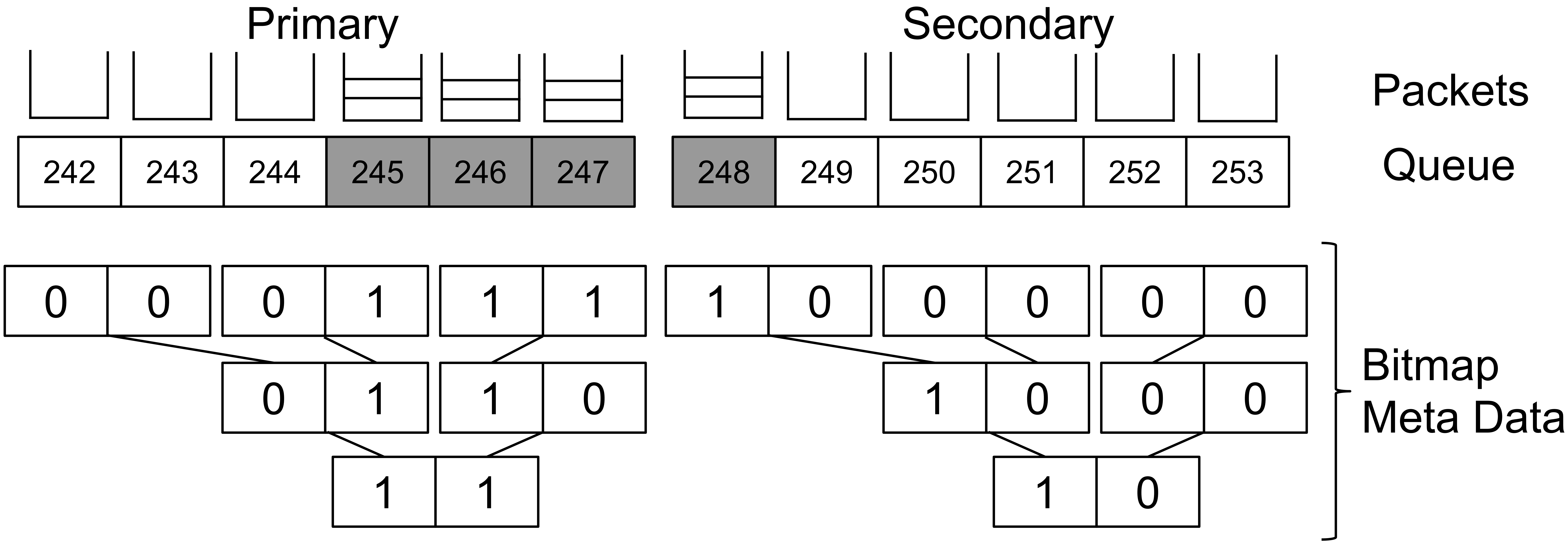
- FindFirstSet (FFS) in a 64-bit word in 3 CPU cycles
    - Every bucket is represented by a bit
    - Bit is set iff bucket is not empty
- O(1) Integer Priority Queue in for N=64
    - Linux Real Time Process Scheduler
    - Quick Fair Queuing (QFQ)
      [F. Checconi et al. INFOCOM ’13]



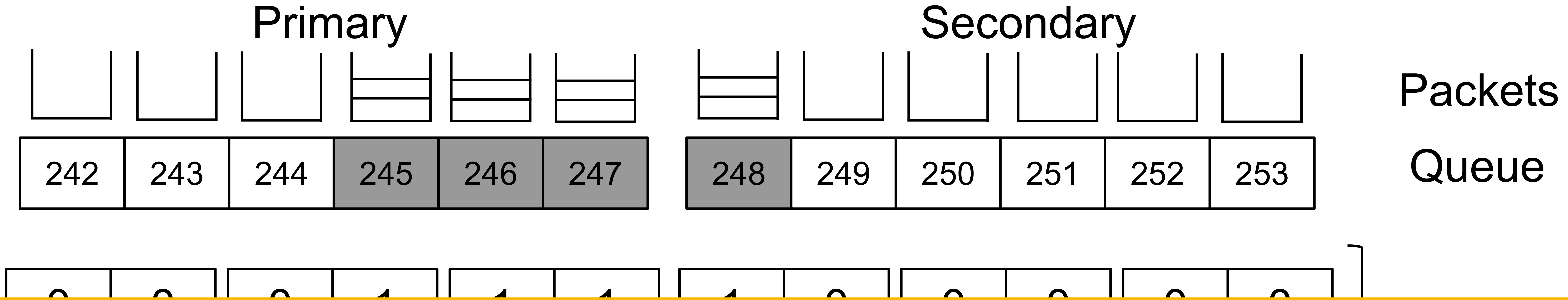
Hierarchical FFS-based Queue



Circular Hierarchical FFS-based Queue



Circular Hierarchical FFS-based Queue



cFFS-based queues has a small memory footprint and requires $O(\log_w N)$ steps for ExtractMin operating over a small N

Scheduler Programmability

PIFO Programming Model

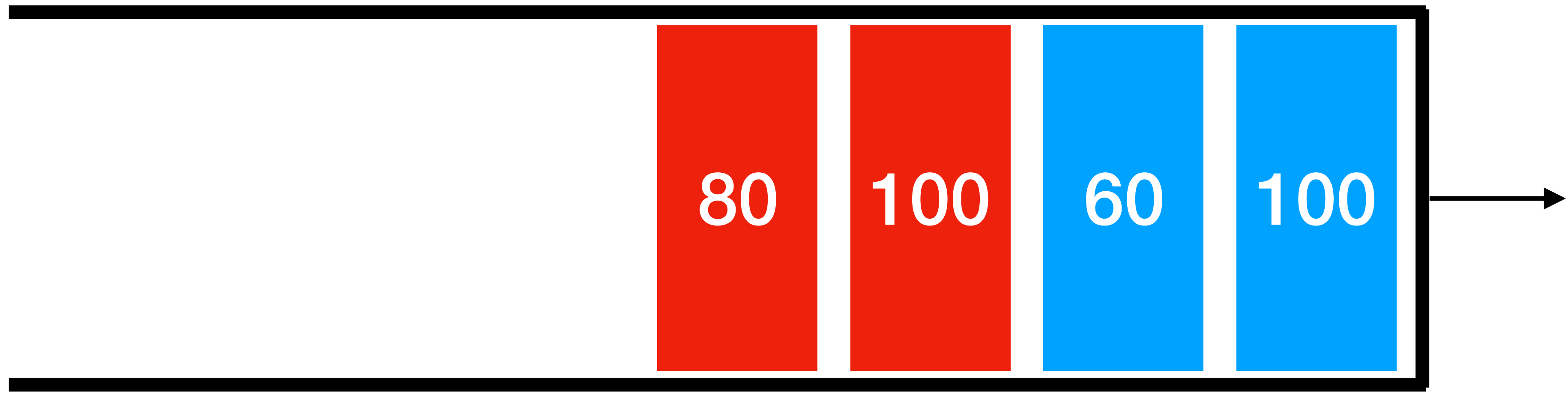
- Eiffel extends Push In First Out (PIFO) model
- PIFO model capture hierarchical policies using tress of priority queues [Sivaraman et. al SIGCOMM ’16]
  - Packet ranking is performed on enqueue
  - Scheduling and shaping are tightly coupled in a single transaction
  - Implemented in hardware through parallel comparisons

Eiffel Programming Model

- Eiffel model extends the PIFO model
    - Packets can be ordered based on flow ranking
    - Flows and packets can be ranked on enqueue and dequeue
    - Shaping and scheduling are decoupled for efficiency

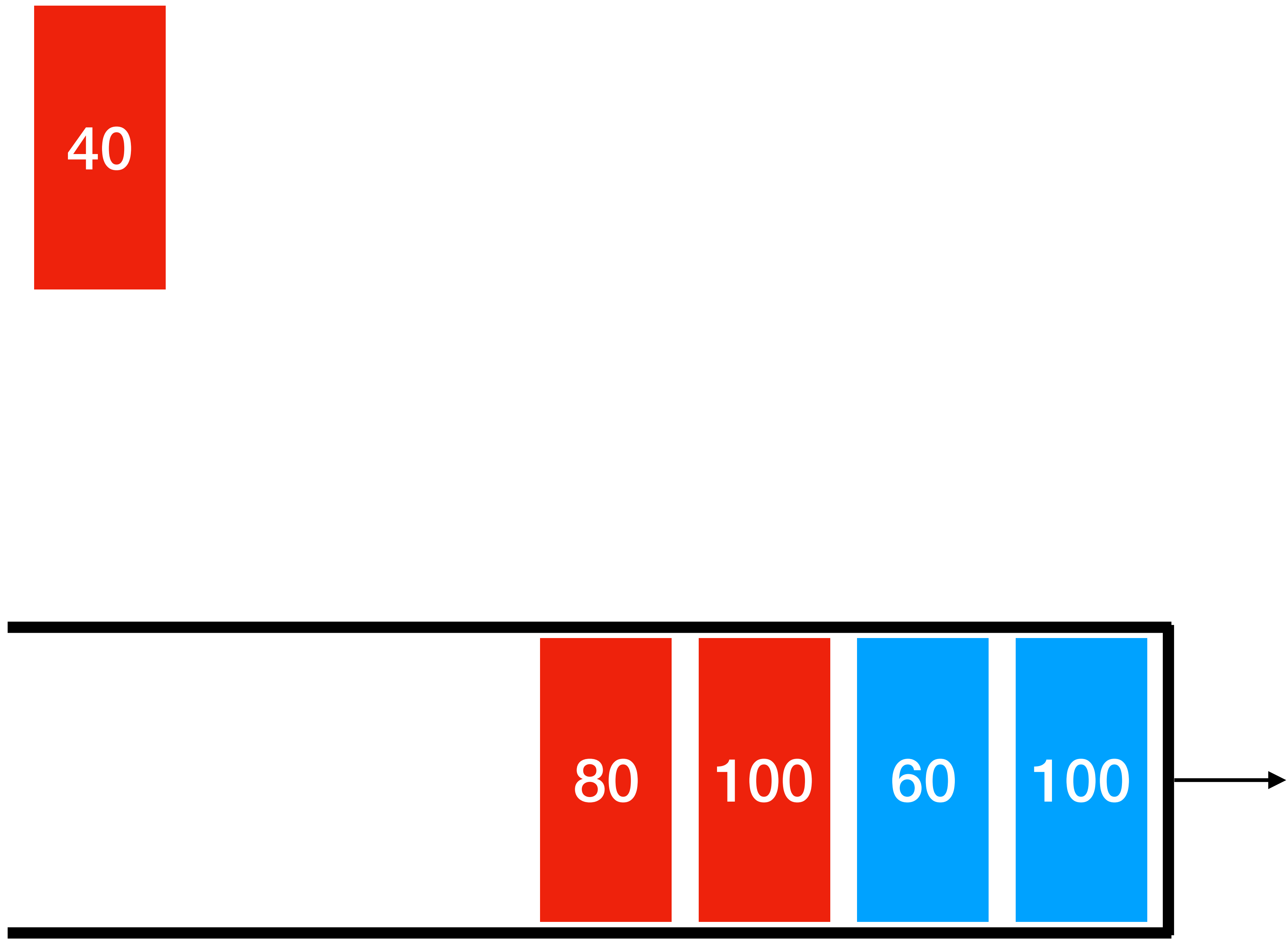
Eiffel Example: pFabric

- Each packet is tagged with Remaining Processing Time
- Packets are transmitted with *Shortest Remaining Processing Time First (SRPTF)*
- To avoid starvation, earliest packet from the highest priority flow is transmitted
- pFabric requires prioritizing flows based on ranks of packets



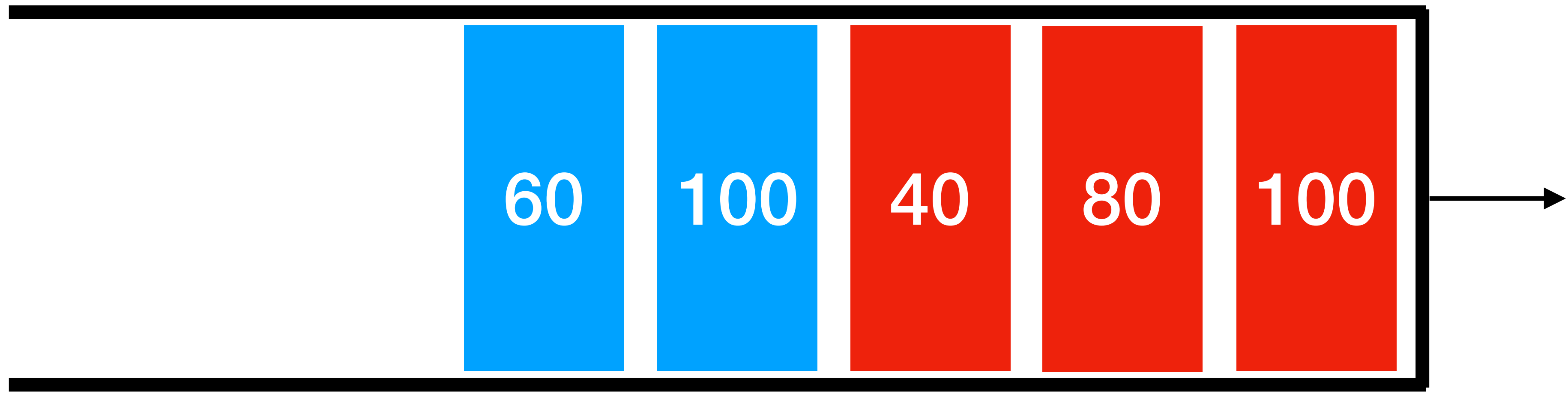
Eiffel Example: pFabric

- Each packet is tagged with Remaining Processing Time
- Packets are transmitted with *Shortest Remaining Processing Time First (SRPTF)*
- To avoid starvation, earliest packet from the highest priority flow is transmitted
- pFabric requires prioritizing flows based on ranks of packets



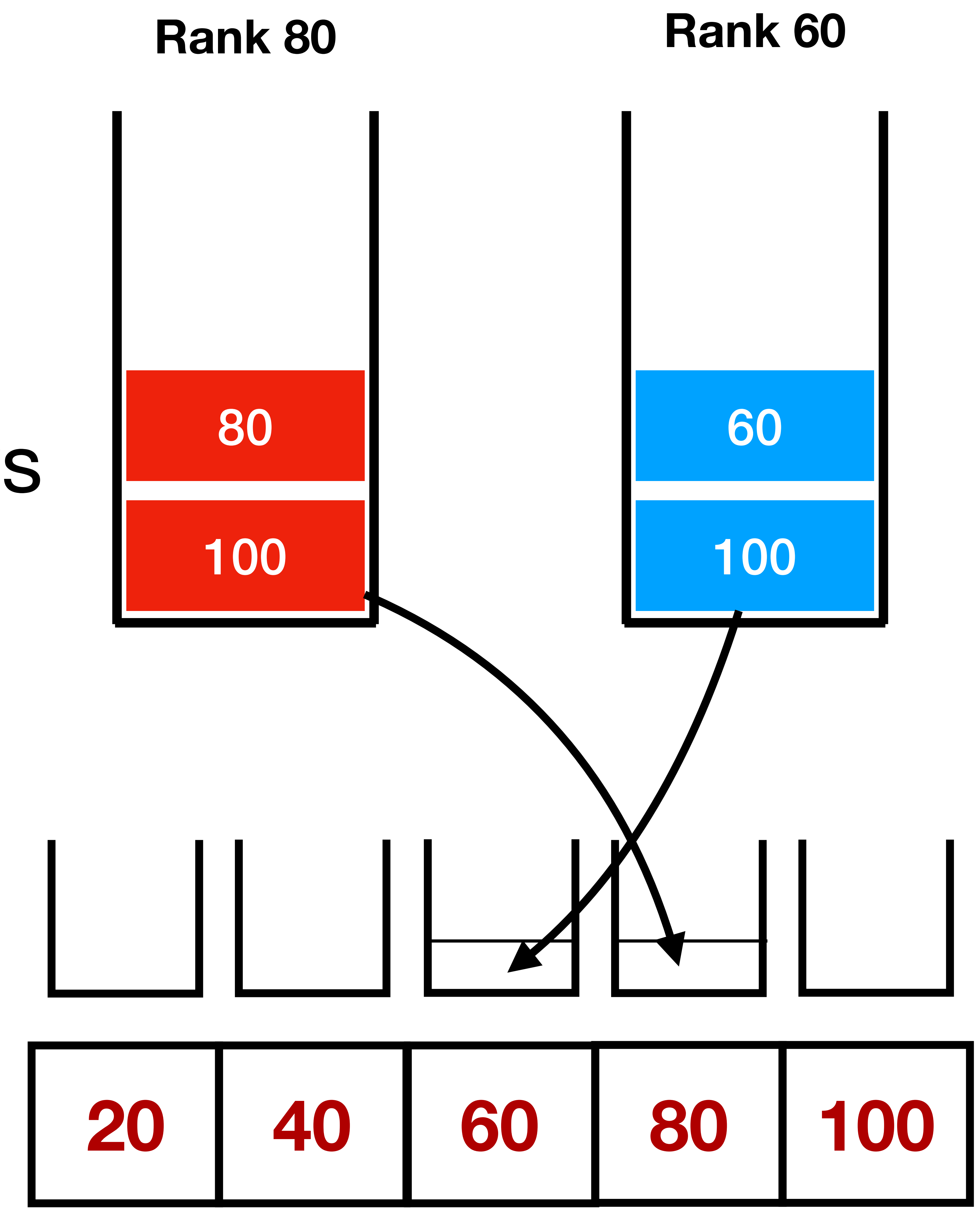
Eiffel Example: pFabric

- Each packet is tagged with Remaining Processing Time
- Packets are transmitted with *Shortest Remaining Processing Time First (SRPTF)*
- To avoid starvation, earliest packet from the highest priority flow is transmitted
- pFabric requires prioritizing flows based on ranks of packets



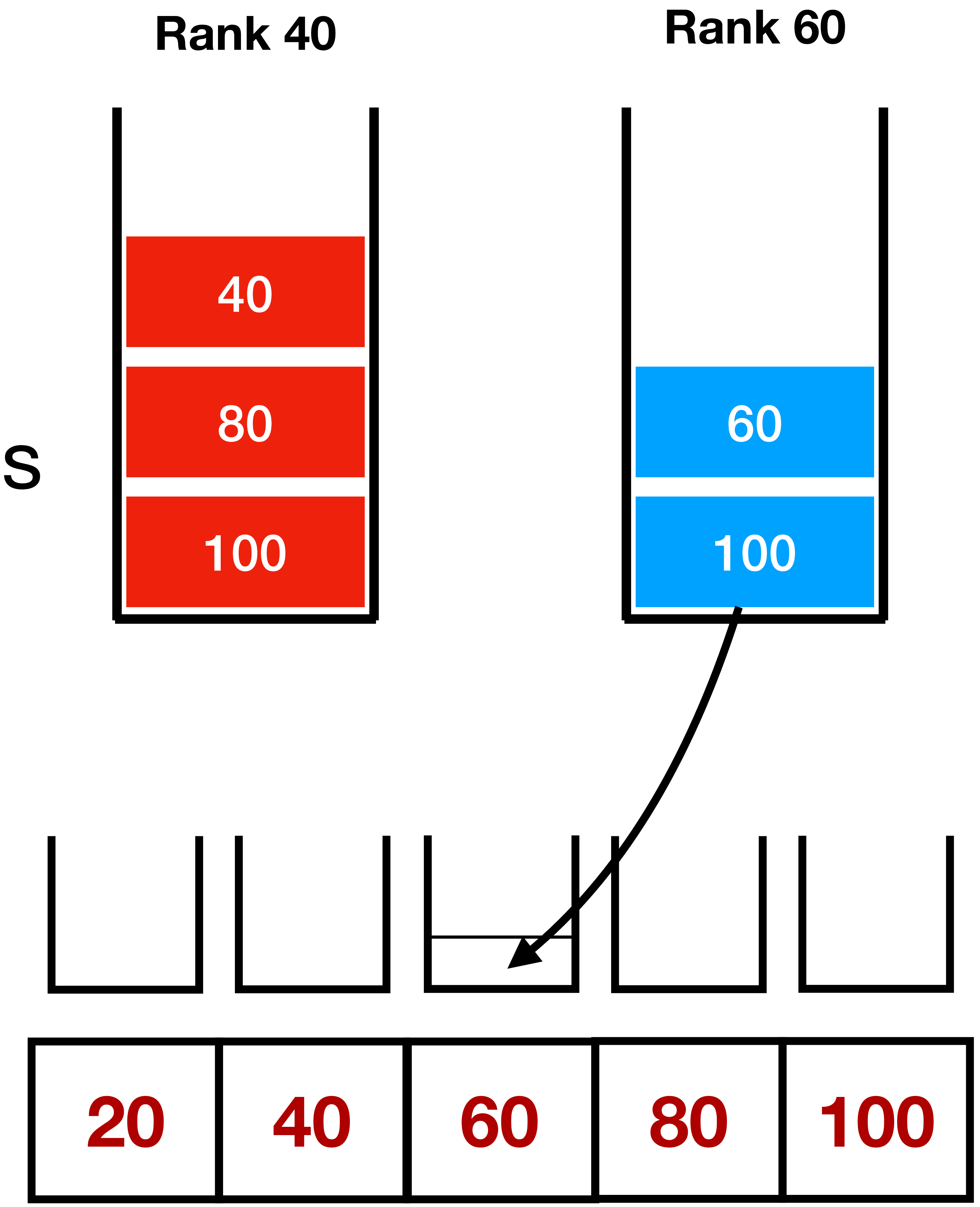
Eiffel Example: Implementation

- Data structures
  - Priority Queue per policy that ranks flows
  - FIFO queue per-flow
- On packet enqueue
  - Check packet tag and update flow rank
  - Update flow position in priority queue



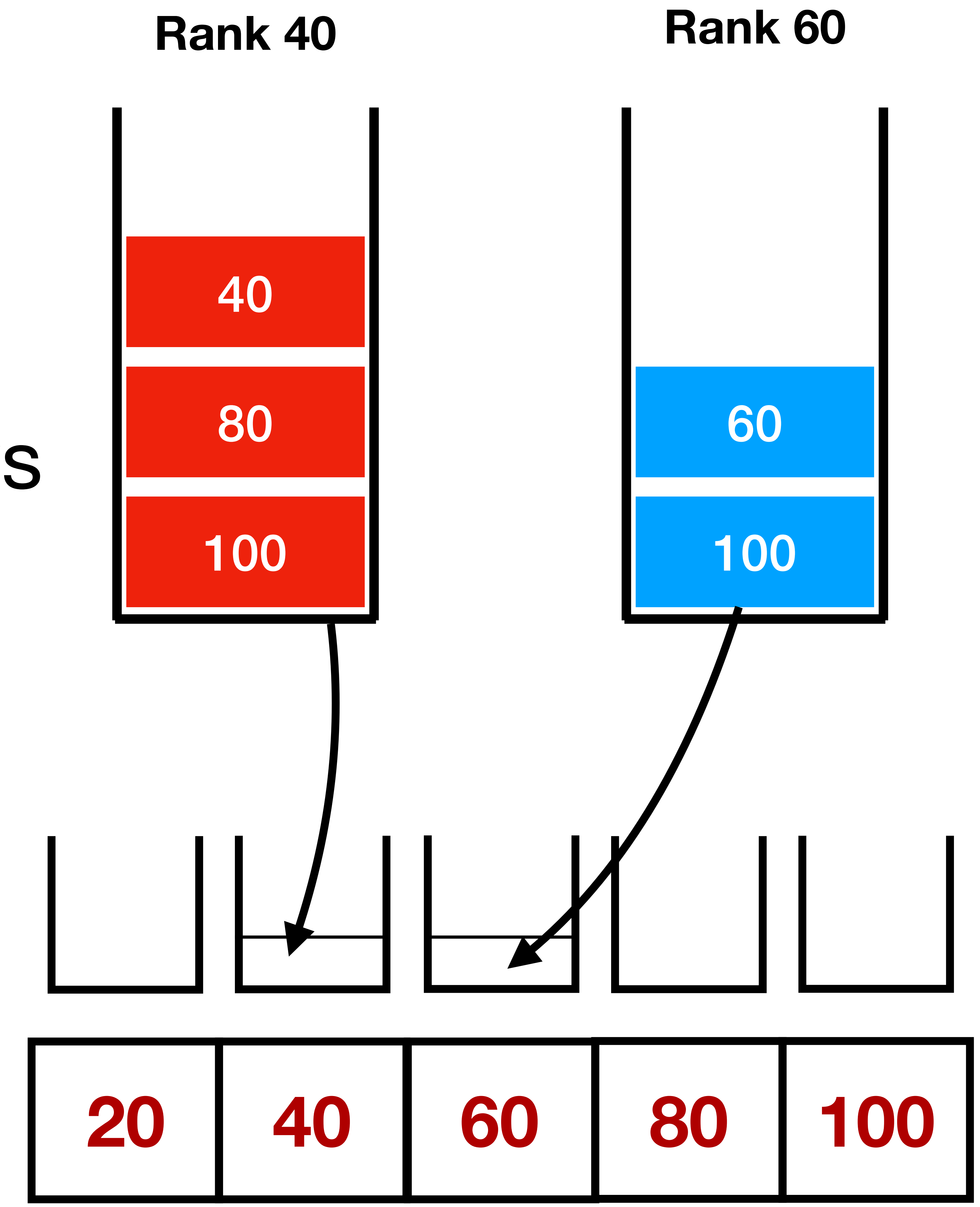
Eiffel Example: Implementation

- Data structures
  - Priority Queue per policy that ranks flows
  - FIFO queue per-flow
- On packet enqueue
  - Check packet tag and update flow rank
  - Update flow position in priority queue



Eiffel Example: Implementation

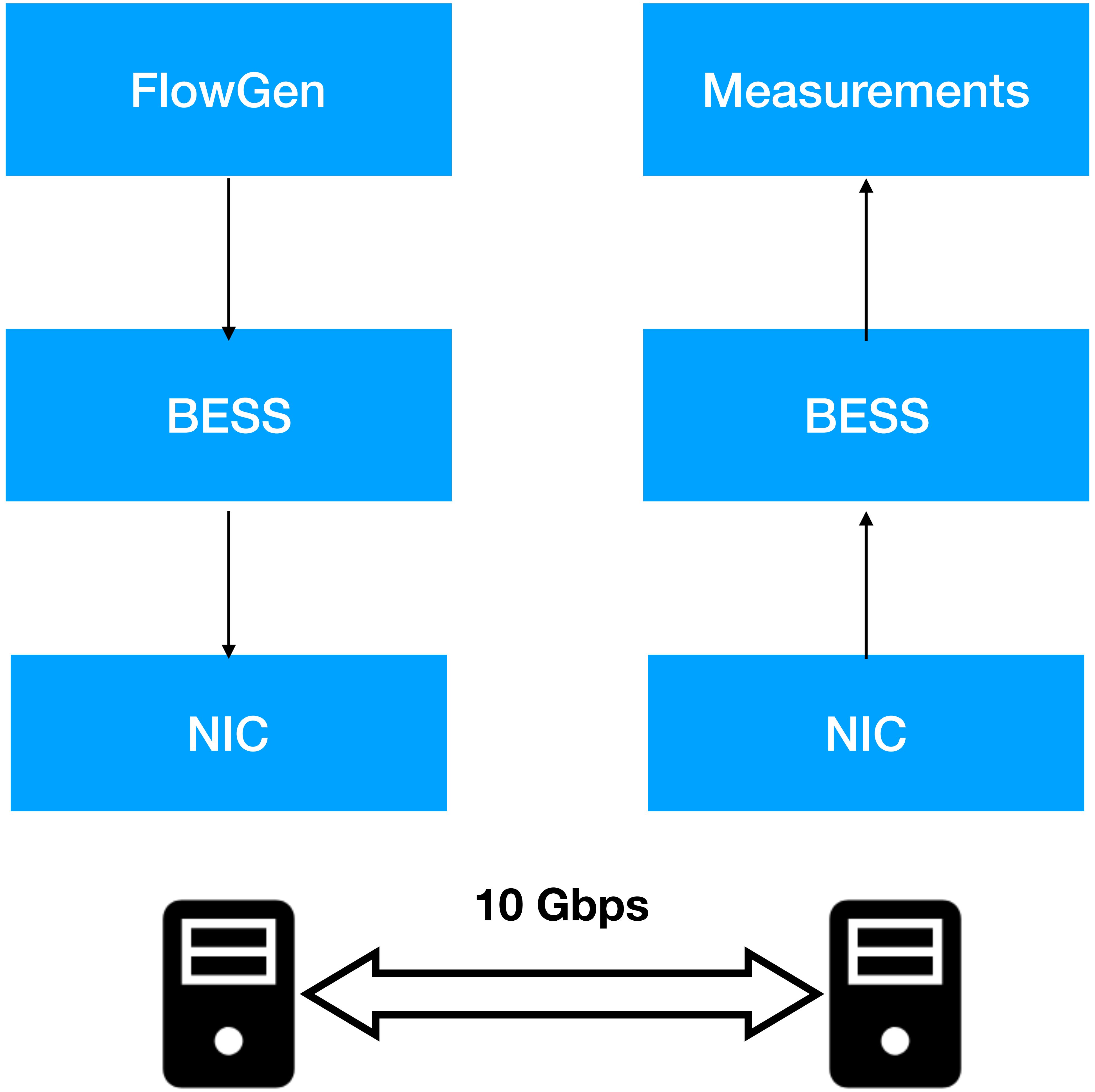
- Data structures
  - Priority Queue per policy that ranks flows
  - FIFO queue per-flow
- On packet enqueue
  - Check packet tag and update flow rank
  - Update flow position in priority queue



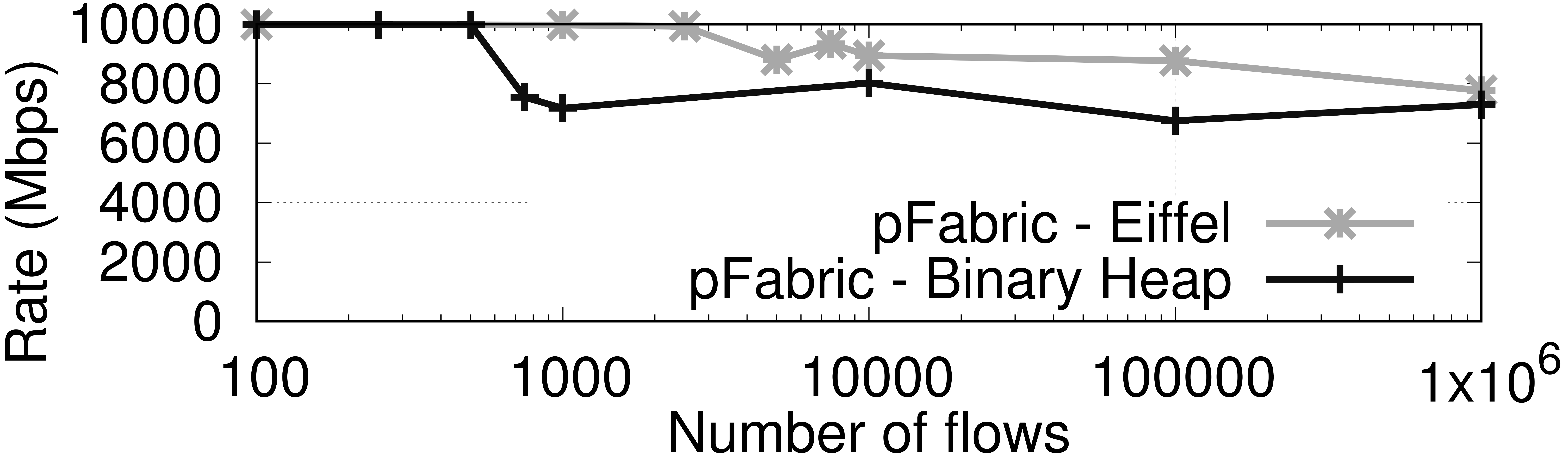
Evaluation

Evaluation Setup

- Two servers with Intel X520-SR2 dual port NICs
- Eiffel implemented in Berkeley Extensible Software Switch (BESS)
- BESS runs on a single dedicated core
- Traffic generated using BESS FlowGen with varying number of flows and fixed 1500B packets

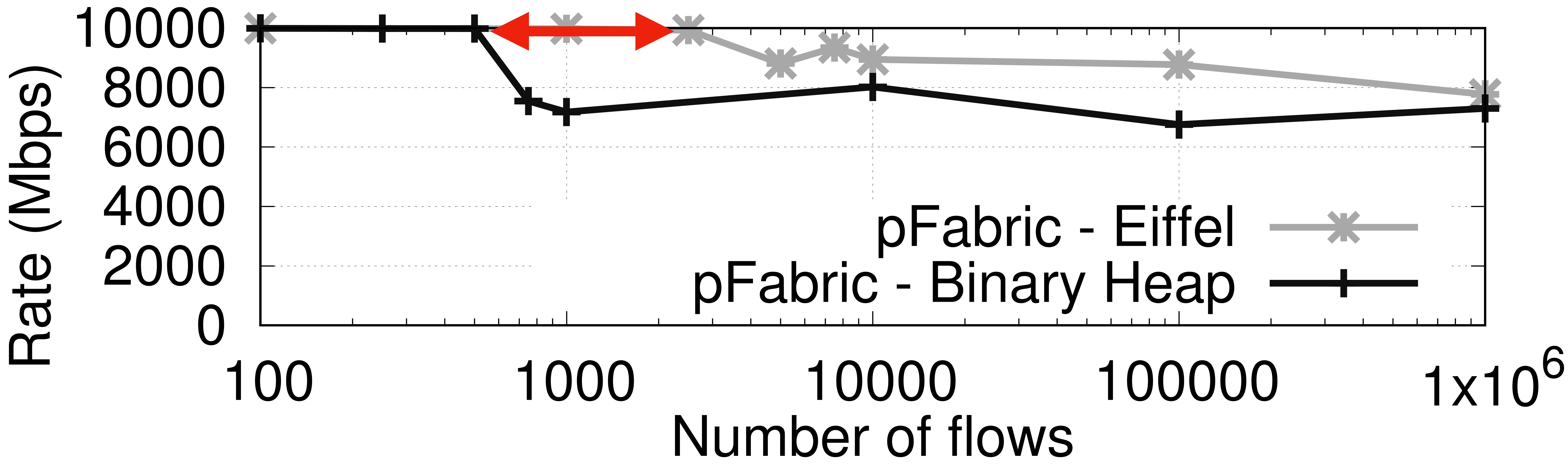


Evaluation



Rate (Mbps)
10000
8000
6000
4000
2000
0
100
1000
10000
100000
1x10^6
Number of flows
pFabric - Eiffel
pFabric - Binary Heap

Evaluation



Eiffel improves capacity by 5x in terms of number of flows that can be handled at line rate

Conclusion

- Eiffel network operators to deploy complex scheduling policies at end hosts and middle boxes
- Eiffel advantages make a strong case for rethinking the building blocks of packet in scheduling in hardware

Questions?