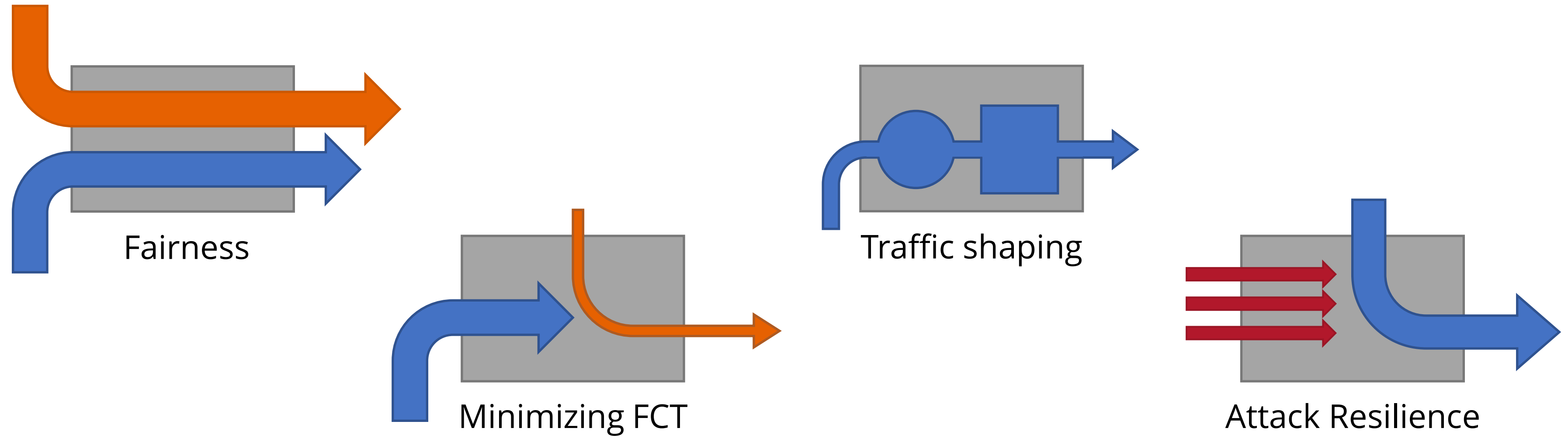
BBQ: A Fast and Scalable Integer Priority Queue for Hardware Packet Scheduling

Nirav Atre, Hugo Sadok, Justine Sherry

Carnegie Mellon University

Packet Scheduling in the Wild

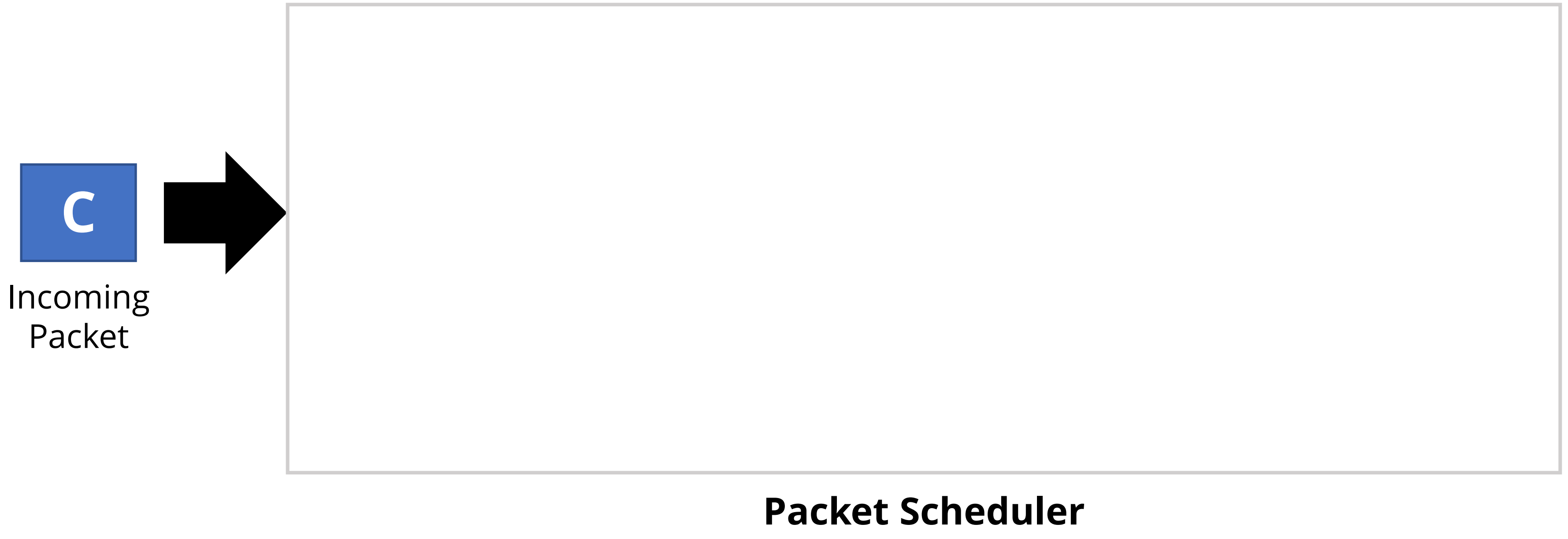
Rich literature on packet scheduling algorithms optimizing for different performance objectives in various network settings



The key to deployment is **programmable hardware packet scheduling**

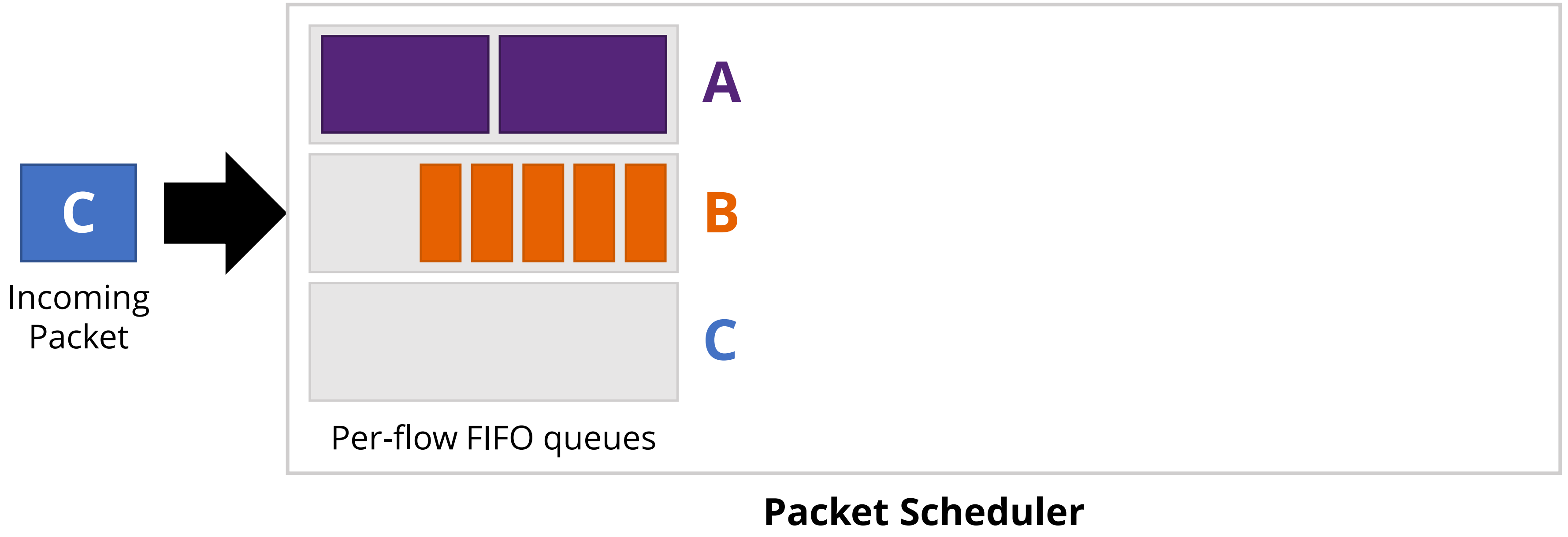
Programmable Packet Scheduling

Programmable Packet Scheduling at Line Rate [SIGCOMM ’16]
→ **Push-In First-Out (PIFO)**



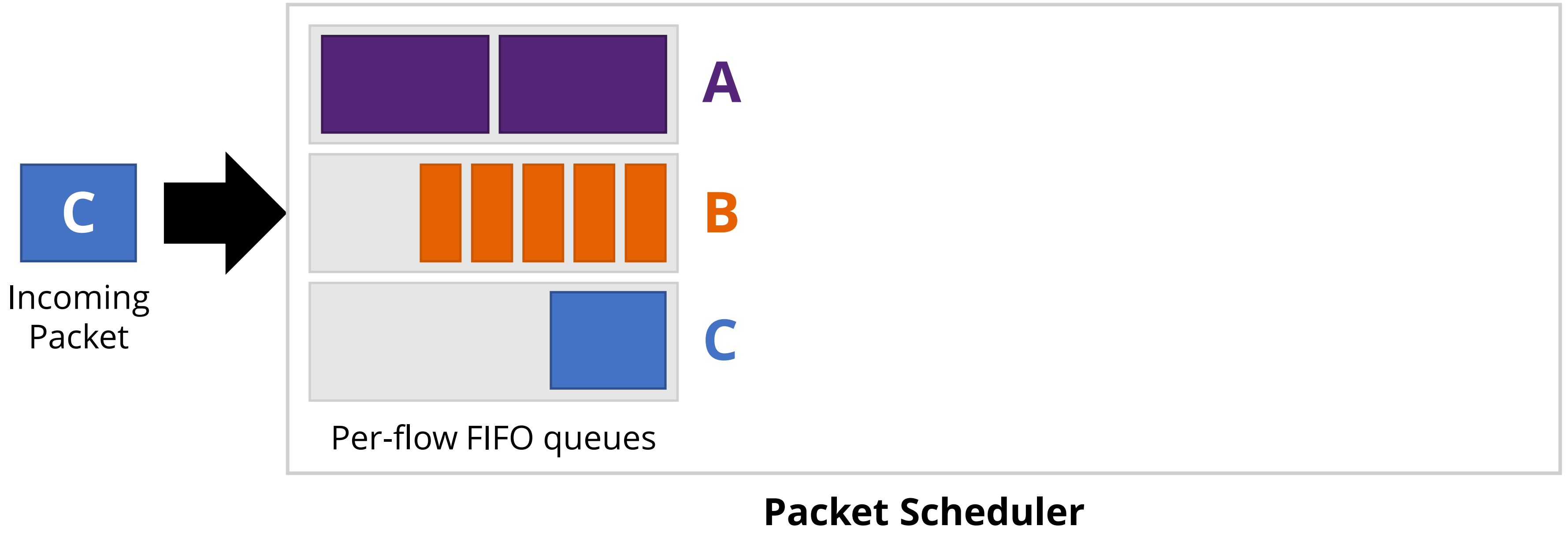
Programmable Packet Scheduling

Programmable Packet Scheduling at Line Rate [SIGCOMM ’16]
→ **Push-In First-Out (PIFO)**



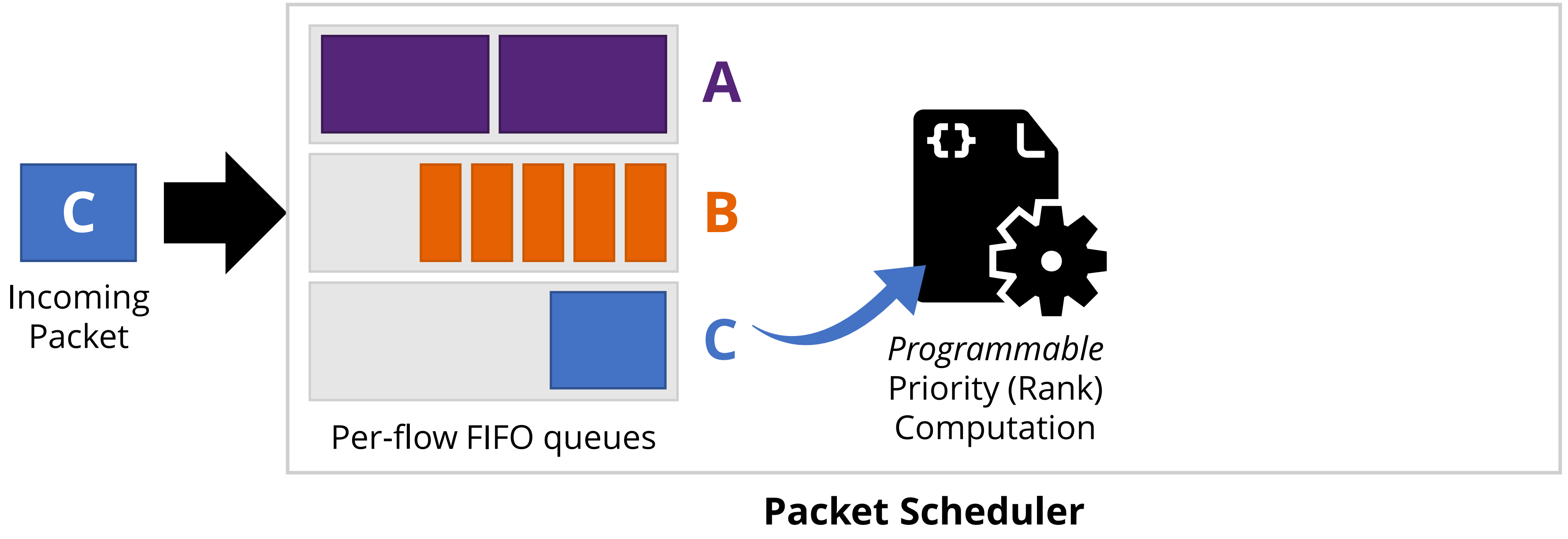
Programmable Packet Scheduling

Programmable Packet Scheduling at Line Rate [SIGCOMM '16]
→ **Push-In First-Out (PIFO)**



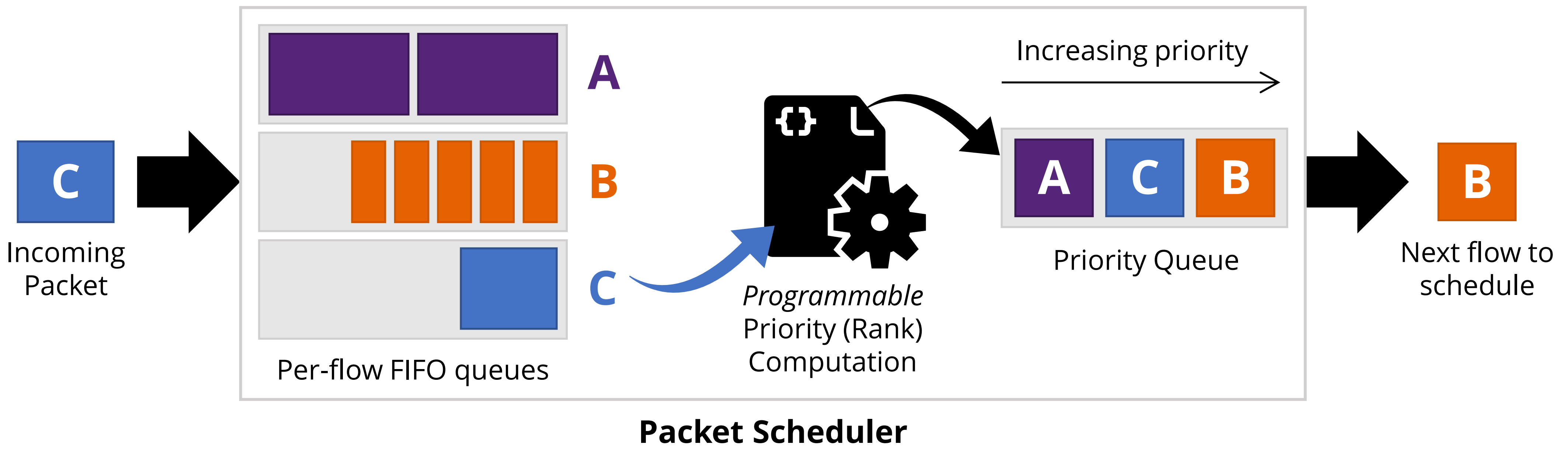
Programmable Packet Scheduling

Programmable Packet Scheduling at Line Rate [SIGCOMM '16]
→ **Push-In First-Out (PIFO)**



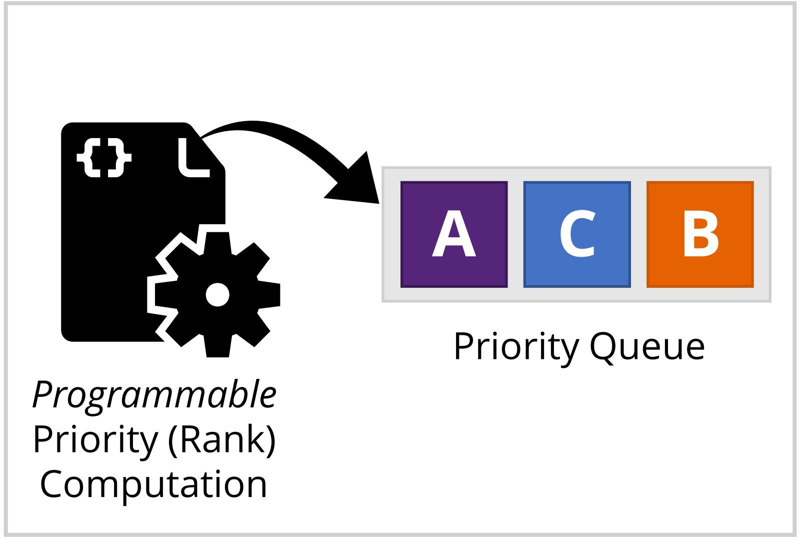
Programmable Packet Scheduling

Programmable Packet Scheduling at Line Rate [SIGCOMM '16]
→ **Push-In First-Out (PIFO)**



Programmable Packet Scheduling

Programmable Packet Scheduling at Line Rate [SIGCOMM '16]
→ **Push-In First-Out (PIFO)**

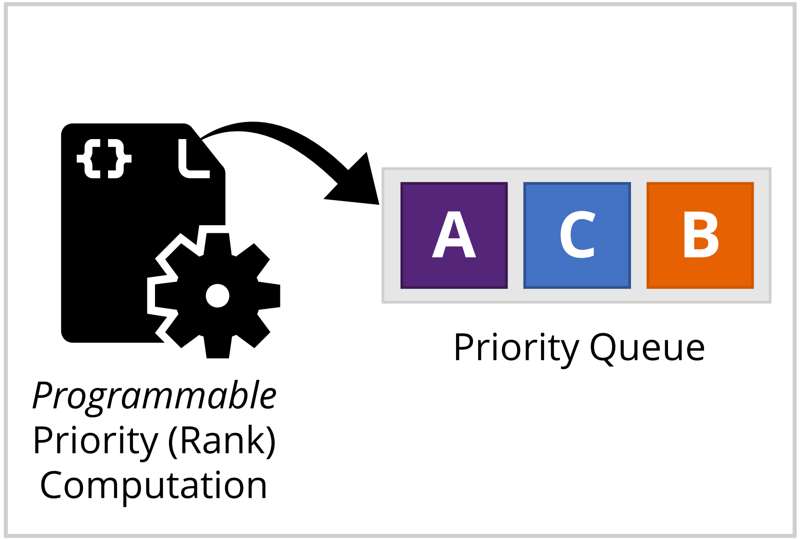


PIFO Abstraction

PIFO, proposed 8 years ago, supports practically any scheduling algorithm
Great! So today's routers must implement this abstraction, right?
...right?

Programmable Packet Scheduling

Programmable Packet Scheduling at Line Rate [SIGCOMM '16]
→ **Push-In First-Out (PIFO)**



PIFO Abstraction

At the heart of PIFO is a **hardware priority queue** that provides, at minimum, enqueue and dequeue-max functionality

PIFO's vision is hampered by throughput, scalability, and resource overhead issues associated with existing priority queue designs

This Talk

- Minimum requirements for scheduling in switches and SmartNICs
- State-of-the-art priority queue designs are infeasible
- How do we get there?
- Evaluation

This Talk

- **Minimum requirements for scheduling in switches and SmartNICs**
- State-of-the-art priority queue designs are infeasible
- How do we get there?
- Evaluation

Minimum requirements for scheduling in switches and SmartNICs

Flow Count Scalability

Support flow counts representative of modern networks: **O(100K)**

Single-Instance Performance

Sustain packet rates corresponding to today's line rates: **100Gbps+ (148.8 Mpps)**

Logical Partitioning

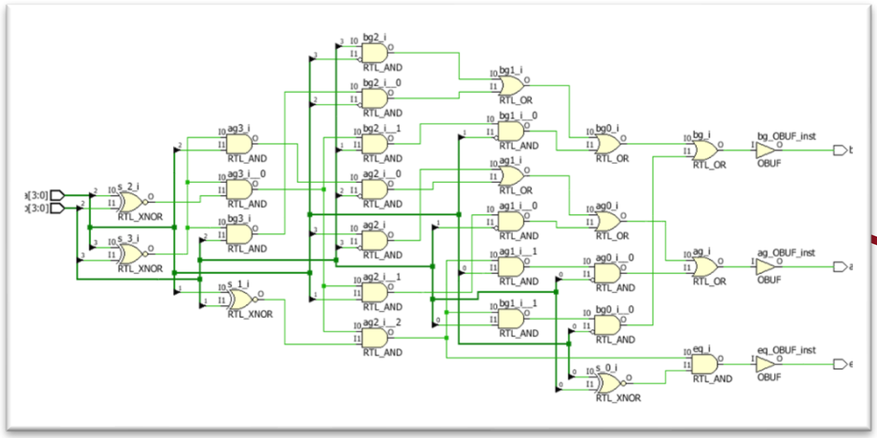
Statistically multiplex a single, **physical** priority queue between many independent **logical** priority queues

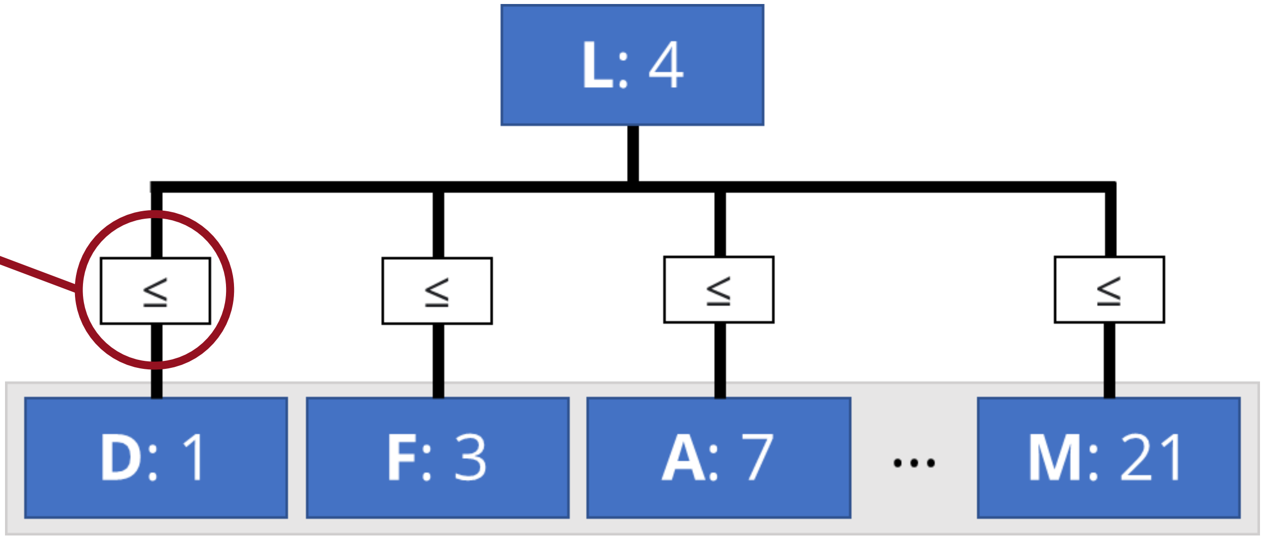
This Talk

- Minimum requirements for scheduling in switches and SmartNICs
- **State-of-the-art priority queue designs are infeasible**
- How do we get there?
- Evaluation

Existing designs are infeasible

PIFO [SIGCOMM '16]

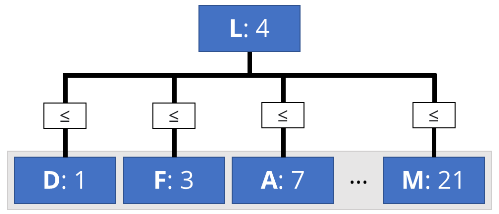




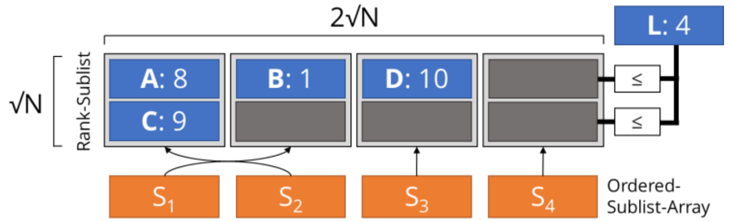
N comparators followed by priority decoding to decide where to insert the next entry → supports at most 4K flows

Poor Scalability

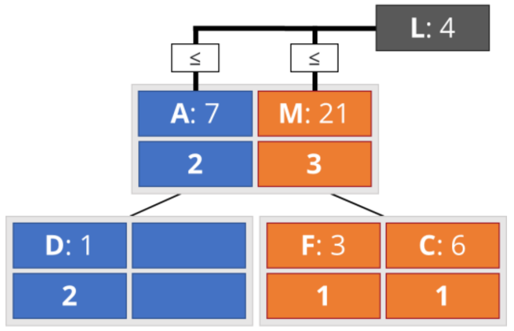
Existing designs are infeasible



PIFO [SIGCOMM '16]



PIEO [SIGCOMM '19]



BMW-Tree [SIGCOMM '23]

Poor Scalability

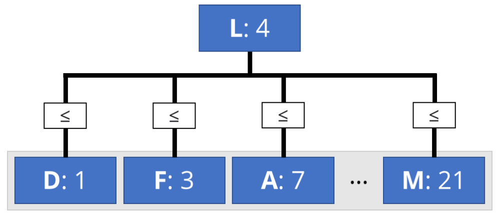
4K flows

64K flows

100K+ flows

Flow Count Scalability

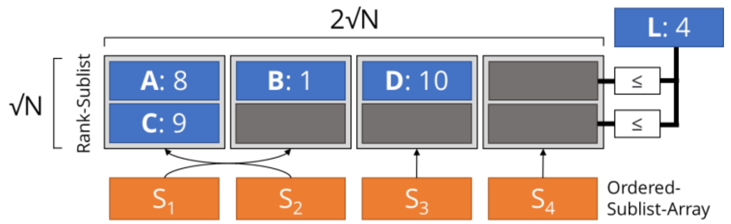
Existing designs are infeasible



PIFO [SIGCOMM '16]

Poor Scalability

4K flows

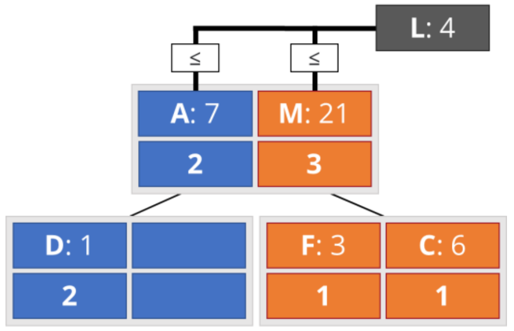


PIEO [SIGCOMM '19]

Poor Performance

64K flows

15 Mpps (10% of line rate at 100 Gbps) on an **FPGA SmartNIC**



BMW-Tree [SIGCOMM '23]

No Logical Partitioning

100K+ flows

1.5 – 6X chip area to implement on a 32-port **Switch ASIC**

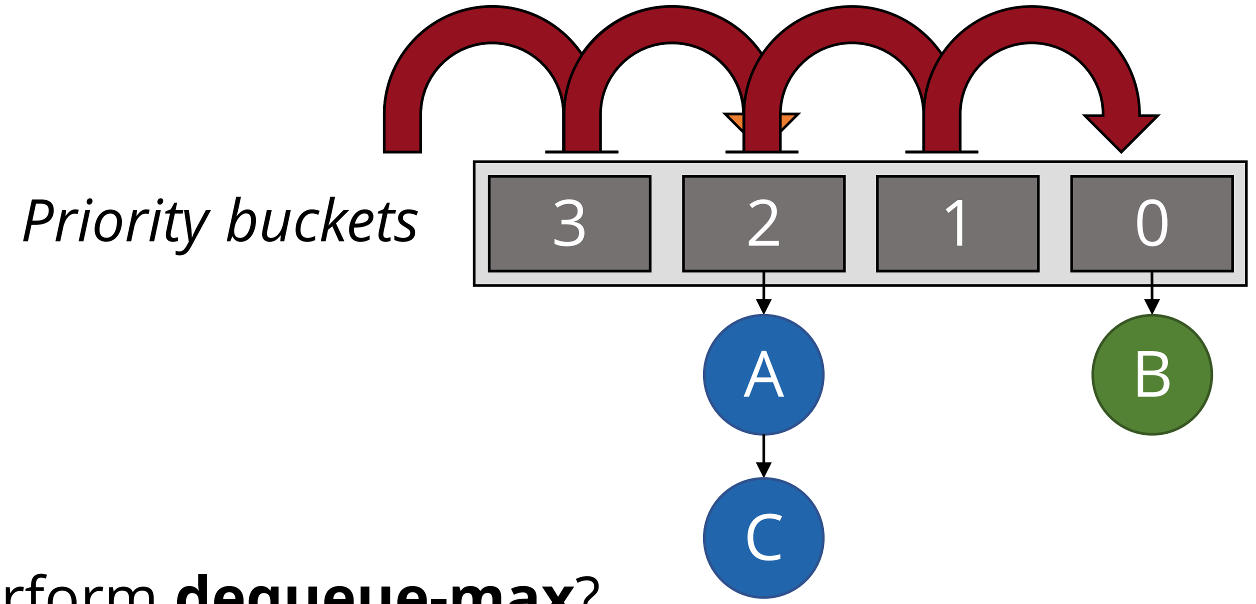
This Talk

- Minimum requirements for scheduling in switches and SmartNICs
- State-of-the-art priority queue designs are infeasible
- **How do we get there?**
- Evaluation

Key Idea

If the priority span is bounded, we can achieve all 3 properties (**scalability**, **performance**, and **logical partitioning**) using non-comparison-based sorting.

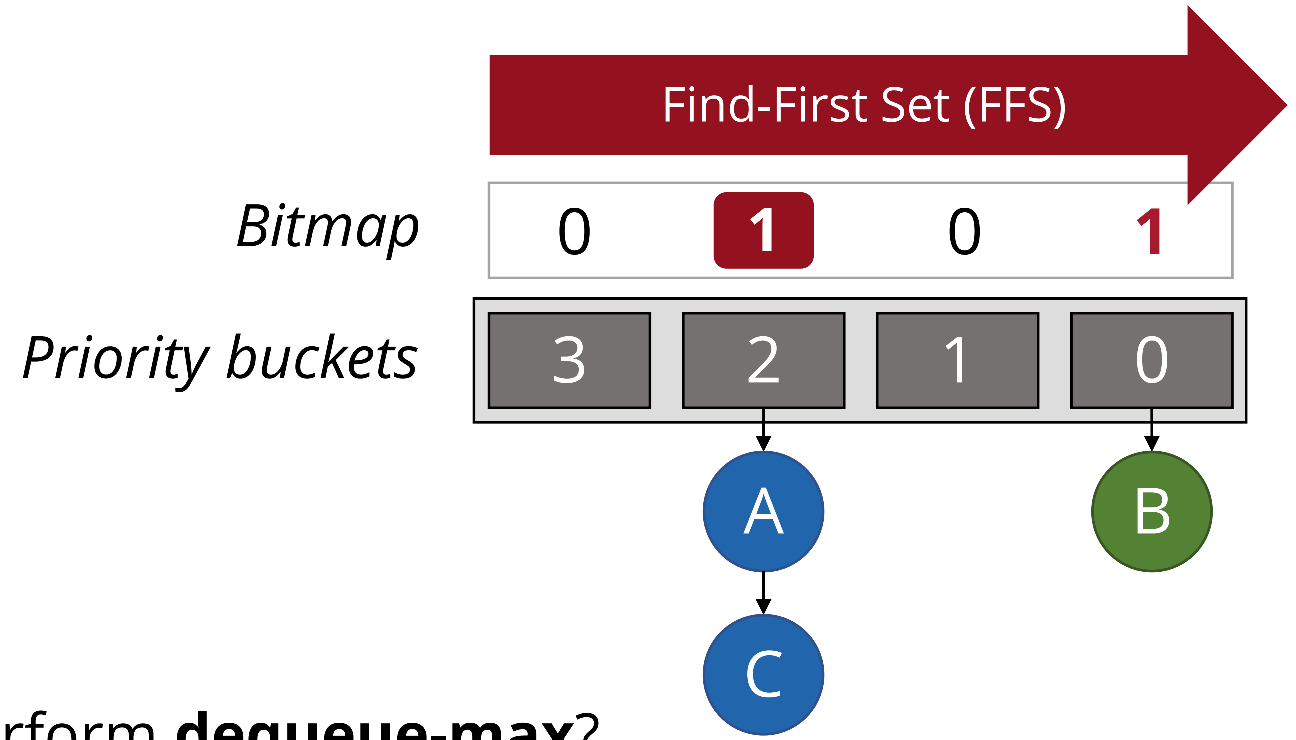
Integer Priority Queueing (IPQ)



How to perform **dequeue-max**?

- Iteratively checking each bucket is slow!

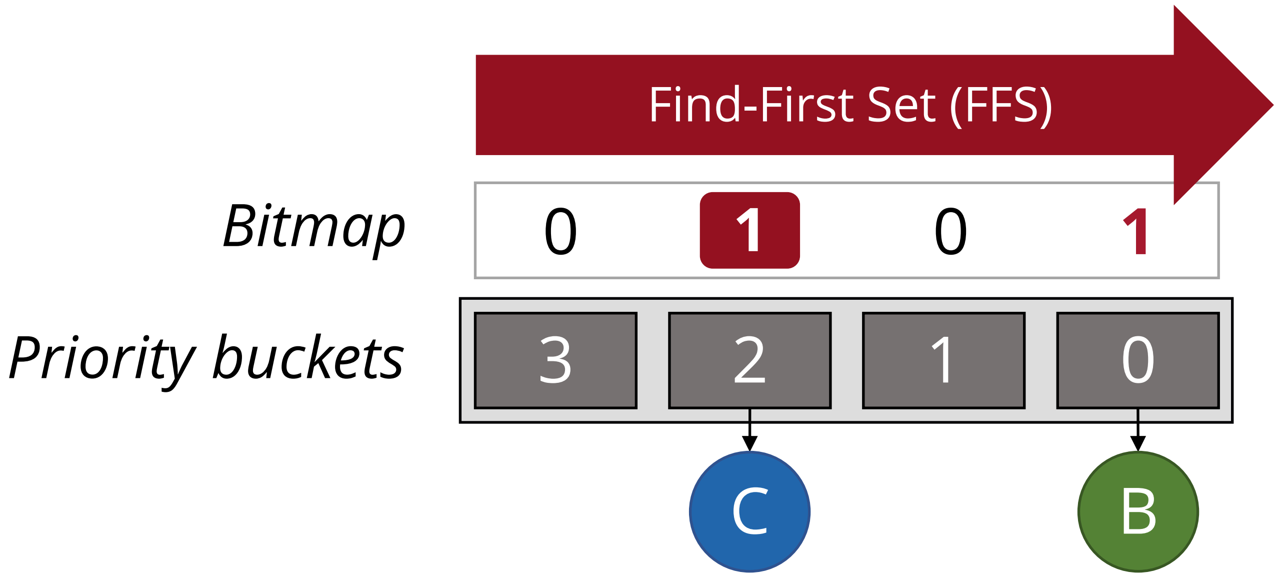
Integer Priority Queueing (IPQ)



How to perform **dequeue-max**?

- Augment with a *bitmap* encoding bucket occupancy, then use *Find-First Set*

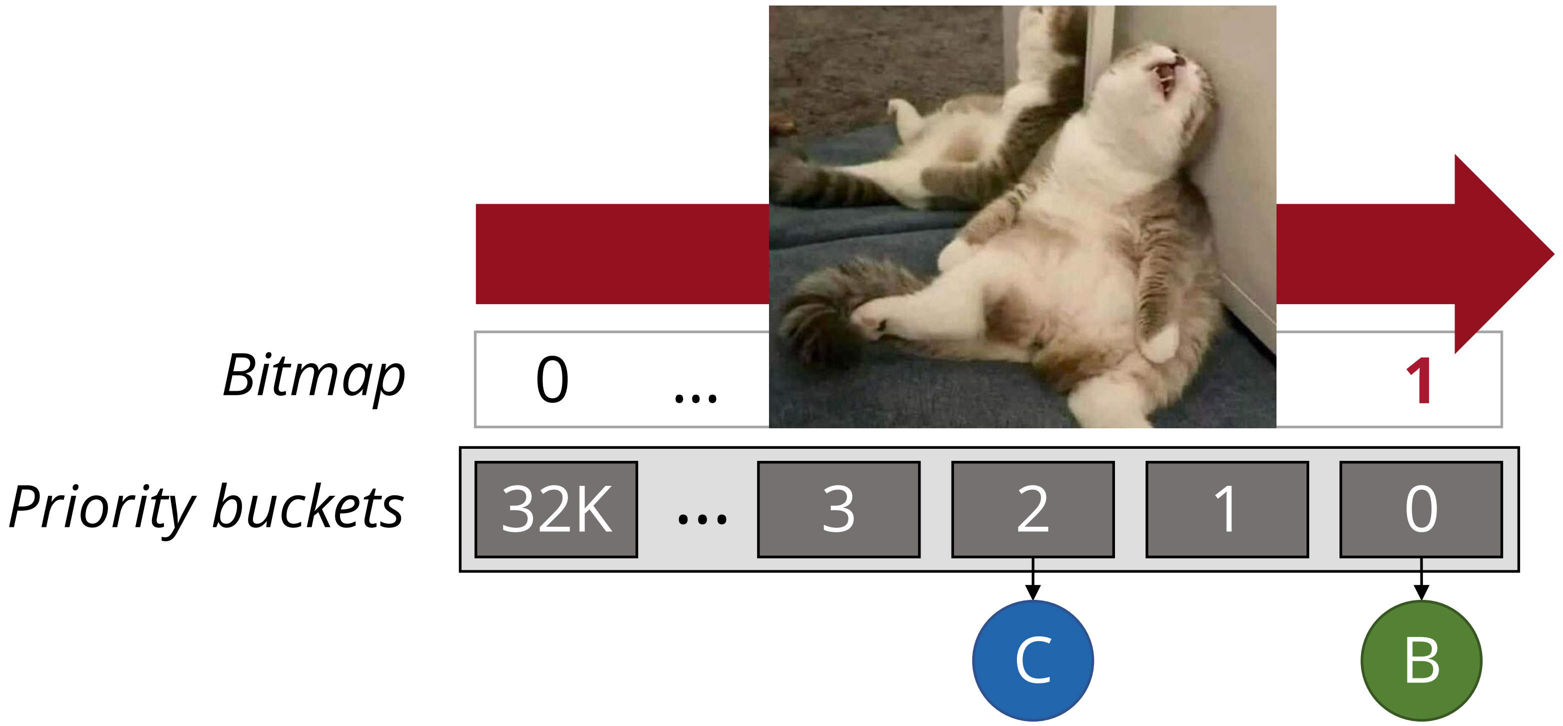
Integer Priority Queueing (IPQ)



How to perform **dequeue-max**?

- Augment with a *bitmap* encoding bucket occupancy, then use *Find-First Set*

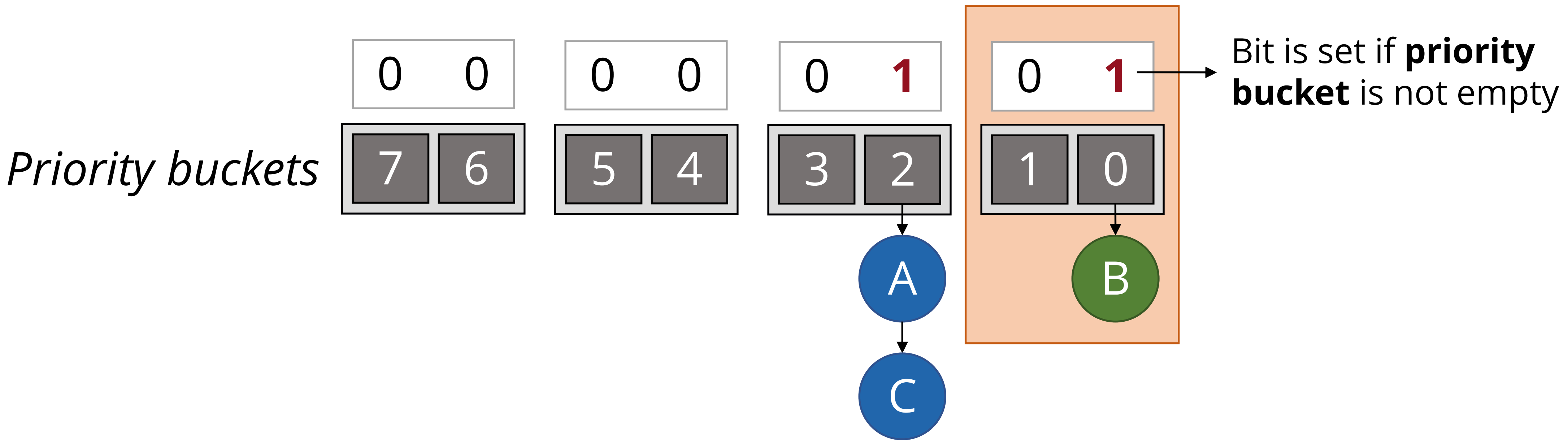
Integer Priority Queueing (IPQ)



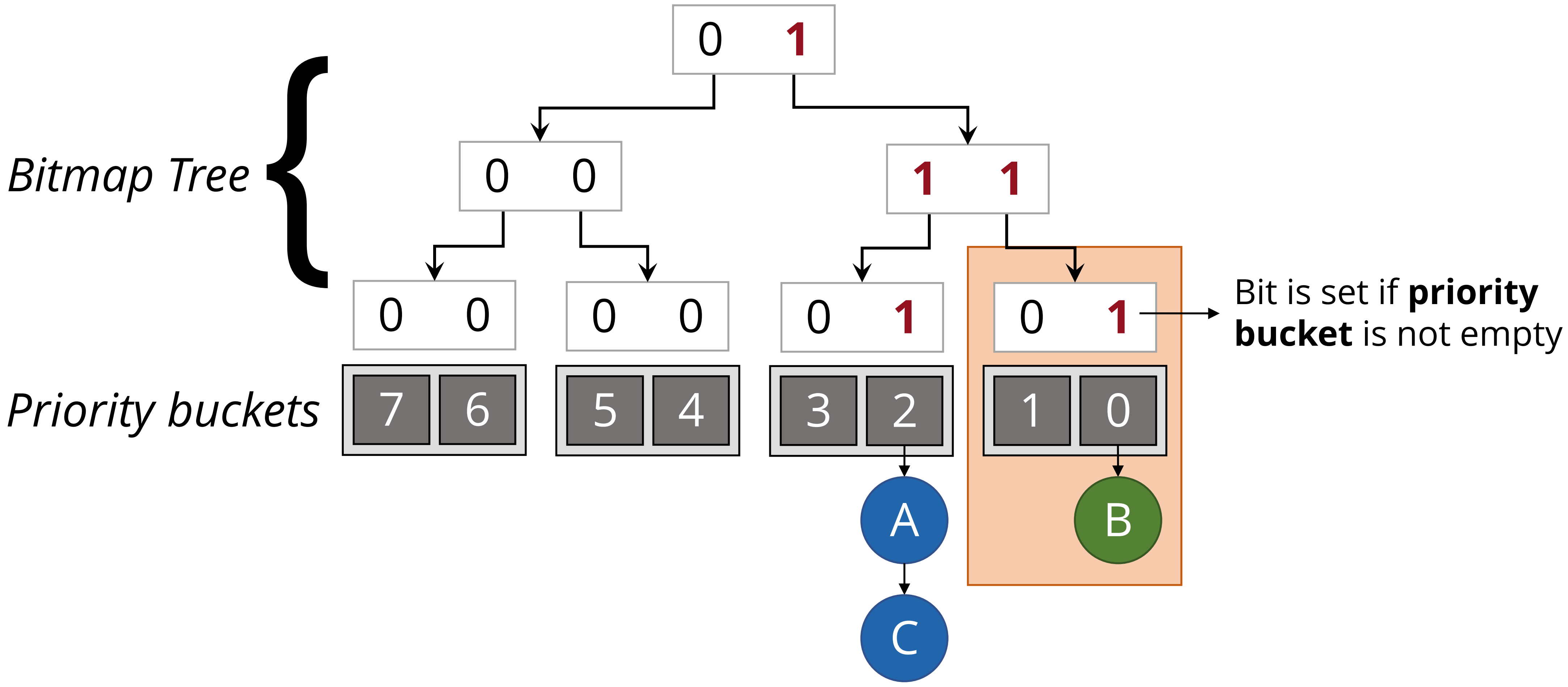
What if we need to support a **huge number of priorities** (*e.g.*, 32K)?

- Can't do FFS on a 32K-bit bitmap

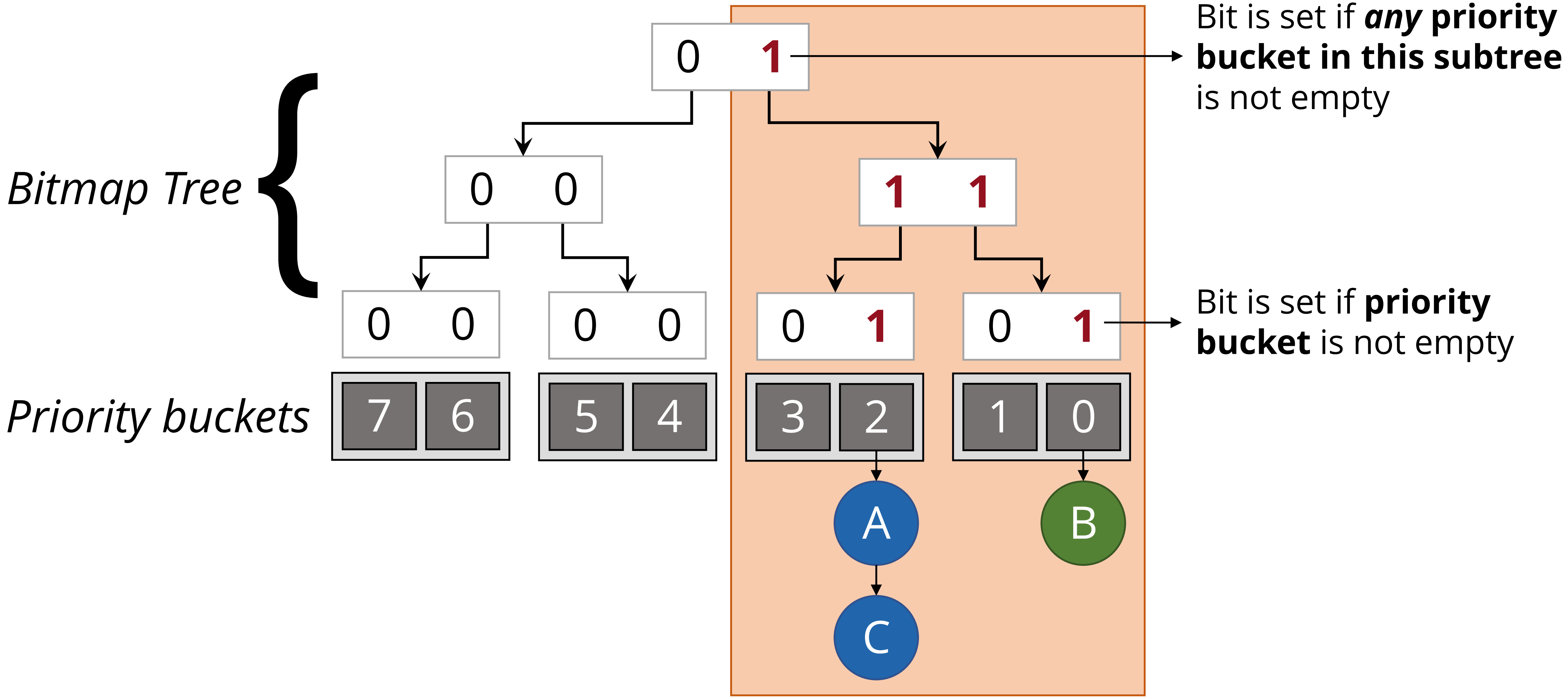
Integer Priority Queueing (IPQ)



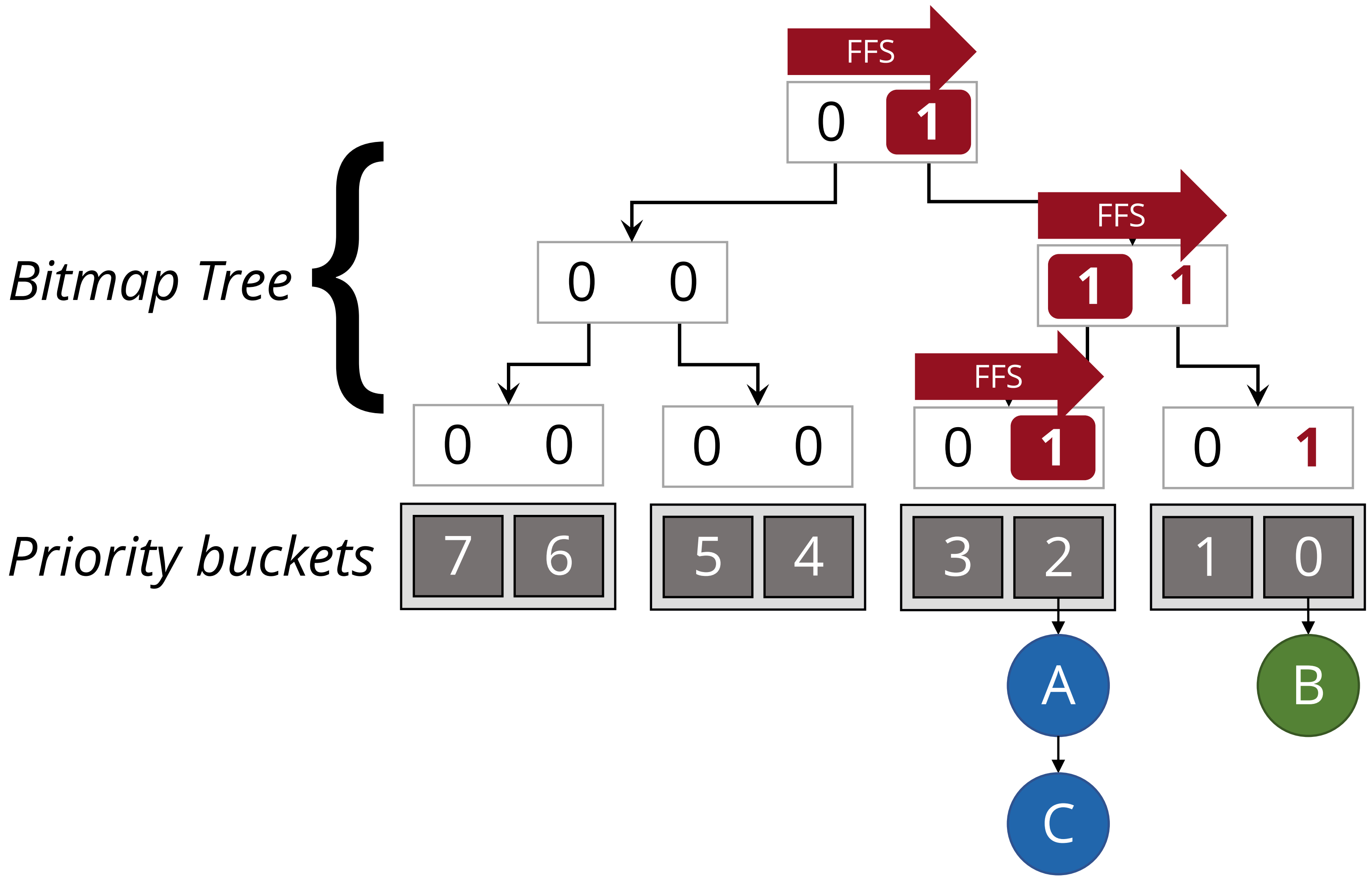
Integer Priority Queueing (IPQ)



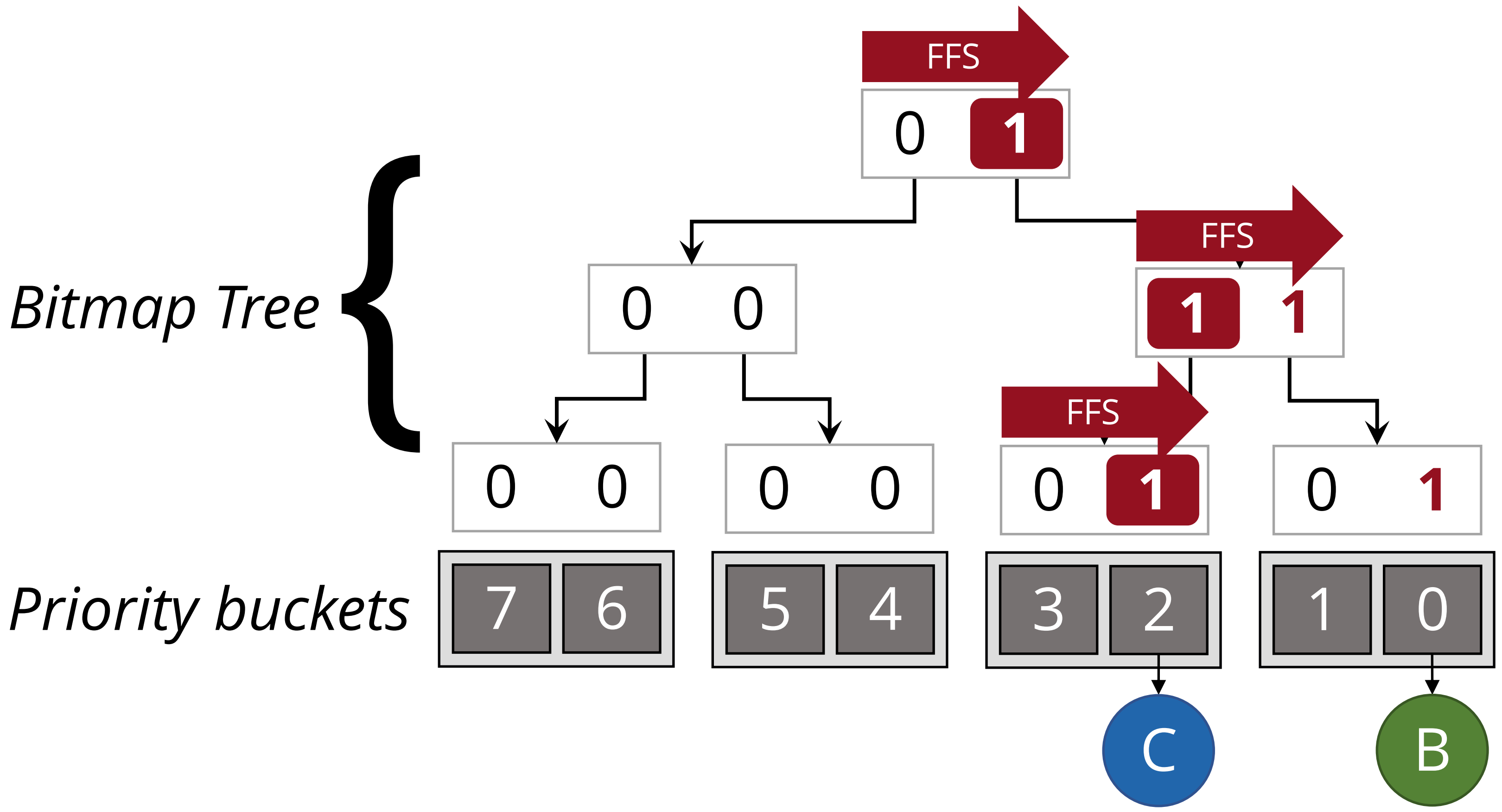
Integer Priority Queueing (IPQ)


Bit is set if ***any* priority bucket in this subtree** is not empty
Bitmap Tree
0 1
0 0
1 1
0 0
0 0
0 1
0 1
Bit is set if **priority bucket** is not empty
Priority buckets
7 6
5 4
3 2
1 0
A
B
C

Integer Priority Queueing (IPQ)



Integer Priority Queueing (IPQ)



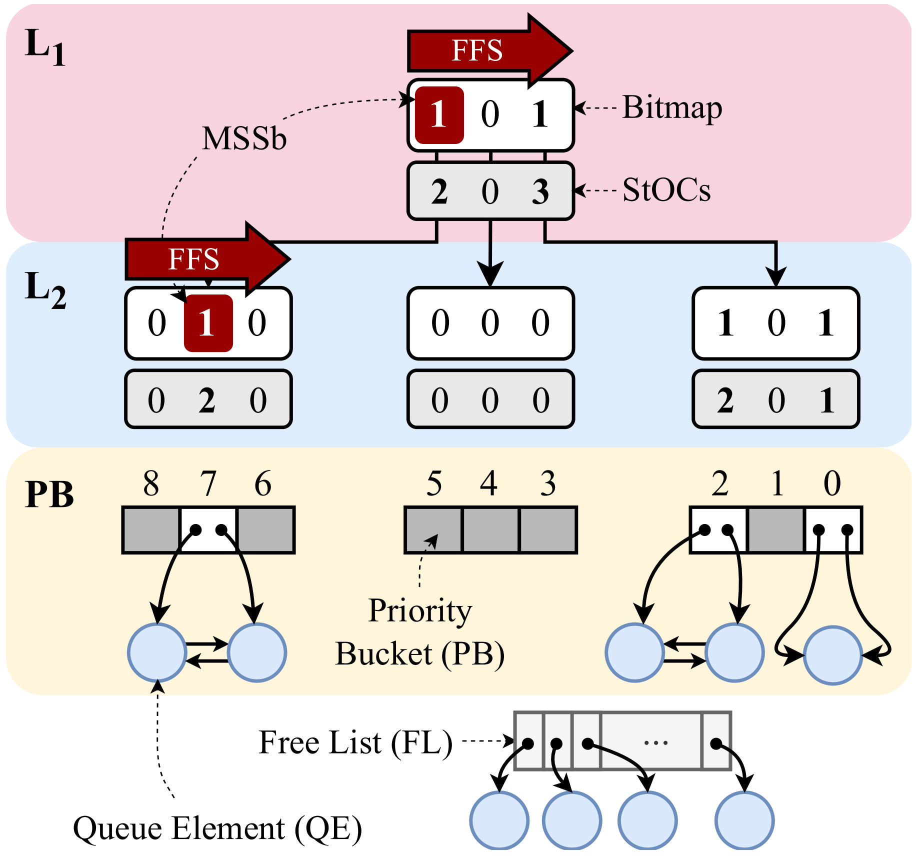
Data-structure is called a **Hierarchical Find-First Set (HFFS) Queue**

Bitmapped Bucket Queue (BBQ)

Data-structure is called a **Hierarchical Find-First Set (HFFS) Queue**

Many software systems use FFS-based priority queueing (*e.g.*, Linux scheduler for process scheduling, and Eiffel [NSDI '19] for packet scheduling).

Our insight is that this data-structure is amenable to a *high-performance, **fully-pipelined** hardware implementation*.



Scalability comes for free

BBQ uses an IPQ-based design, breaking the dependence between queue size and run-time complexity of operations.

Scalability “falls out” of this high-level design choice.

BBQ is ***highly optimized*** for performance

(1) Need a high **operating frequency** (f_{max})

(2) Need to maximize **operations-per-cycle** (OPC)

$$\boxed{\text{Performance (ops/sec)} = f_{max}\ \text{(cycles/sec)} \times \text{OPC (ops/cycle)}}$$

BBQ is *highly optimized* for performance

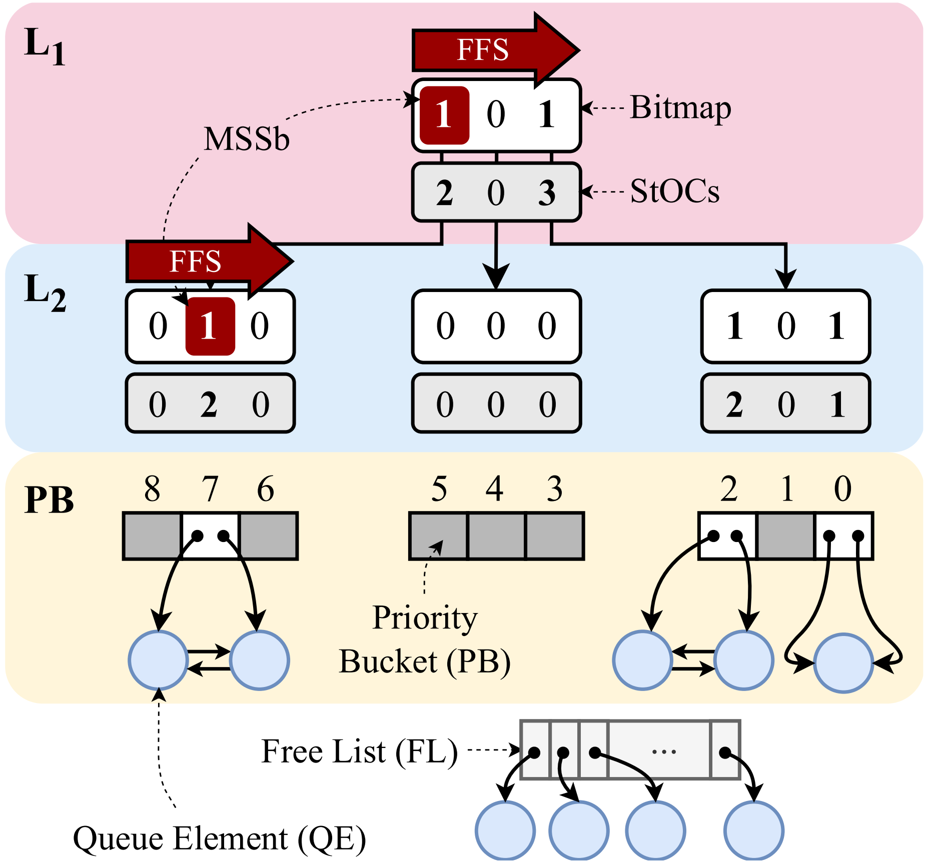
(1) Need a high **operating frequency** (f_{max})

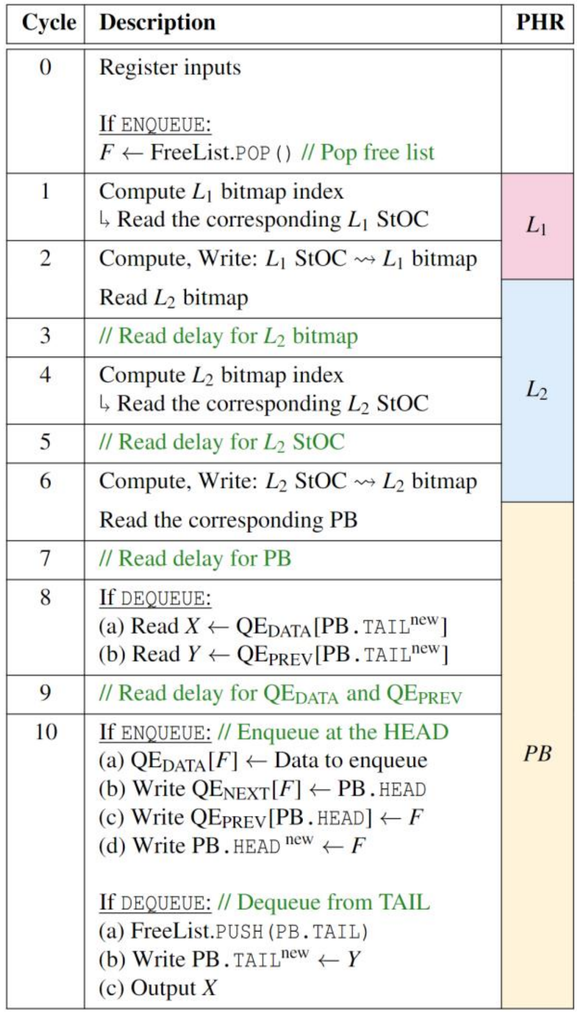
BBQ achieves this by using a deep pipeline where individual stages are designed to do both *little* and *roughly equal* amounts of work.

(2) Need to maximize **operations-per-cycle** (OPC)

$$\boxed{\text{Performance (ops/sec)} = f_{max}\ \text{(cycles/sec)} \times \text{OPC (ops/cycle)}}$$

BBQ is *highly optimized* for performance





Cycle	Description	PHR
0	Register inputs If ENQUEUE: $F \leftarrow$ FreeList.POP() // Pop free list	
1	Compute L_1 bitmap index ↳ Read the corresponding L_1 StOC	L_1
2	Compute, Write: L_1 StOC $\rightsquigarrow$ L_1 bitmap Read L_2 bitmap	
3	// Read delay for L_2 bitmap	
4	Compute L_2 bitmap index ↳ Read the corresponding L_2 StOC	L_2
5	// Read delay for L_2 StOC	
6	Compute, Write: L_2 StOC $\rightsquigarrow$ L_2 bitmap Read the corresponding PB	
7	// Read delay for PB	
8	If DEQUEUE: (a) Read $X \leftarrow \mathrm{QE}_{\mathrm{DATA}}[\mathrm{PB.TAIL}^{new}]$ (b) Read $Y \leftarrow \mathrm{QE}_{\mathrm{PREV}}[\mathrm{PB.TAIL}^{new}]$	
9	// Read delay for $\mathrm{QE}_{\mathrm{DATA}}$ and $\mathrm{QE}_{\mathrm{PREV}}$	
10	If ENQUEUE: // Enqueue at the HEAD (a) $\mathrm{QE}_{\mathrm{DATA}}[F] \leftarrow$ Data to enqueue (b) Write $\mathrm{QE}_{\mathrm{NEXT}}[F] \leftarrow$ PB.HEAD (c) Write $\mathrm{QE}_{\mathrm{PREV}}[\mathrm{PB.HEAD}] \leftarrow F$ (d) Write PB.HEAD$^{new} \leftarrow F$ If DEQUEUE: // Dequeue from TAIL (a) FreeList.PUSH(PB.TAIL) (b) Write PB.TAIL$^{new} \leftarrow Y$ (c) Output X	*PB*

$\}$ Deep (11-stage) hardware pipeline for a 2-level BBQ

BBQ is *highly optimized* for performance

(1) Need a high **operating frequency** (f_{max})

BBQ achieves this by using a deep pipeline where individual stages are designed to do both *little* and *roughly equal* amounts of work

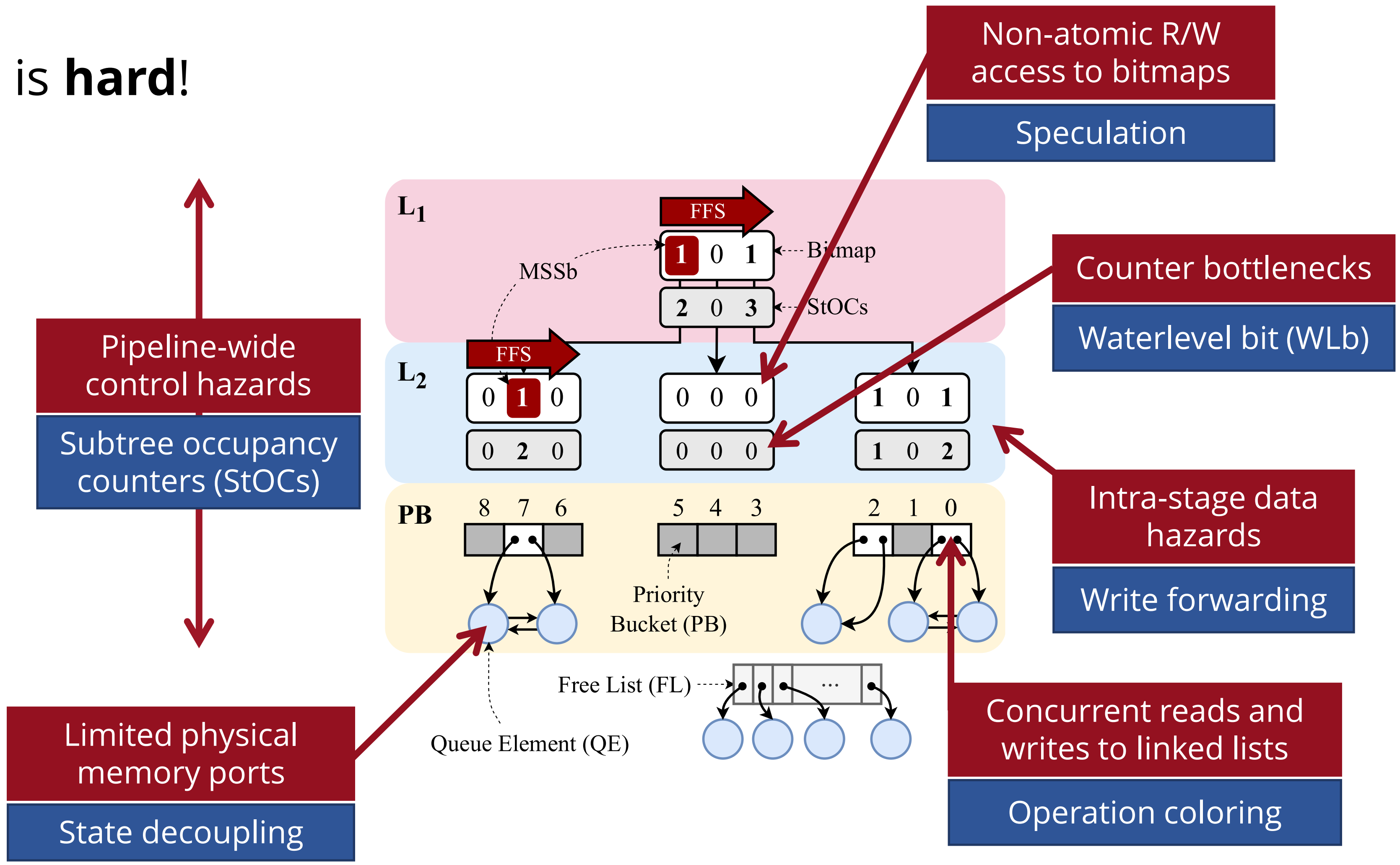
(2) Need to maximize **operations-per-cycle** (OPC)

BBQ realizes a fully-pipelined architecture (OPC of 1) agnostic of workload

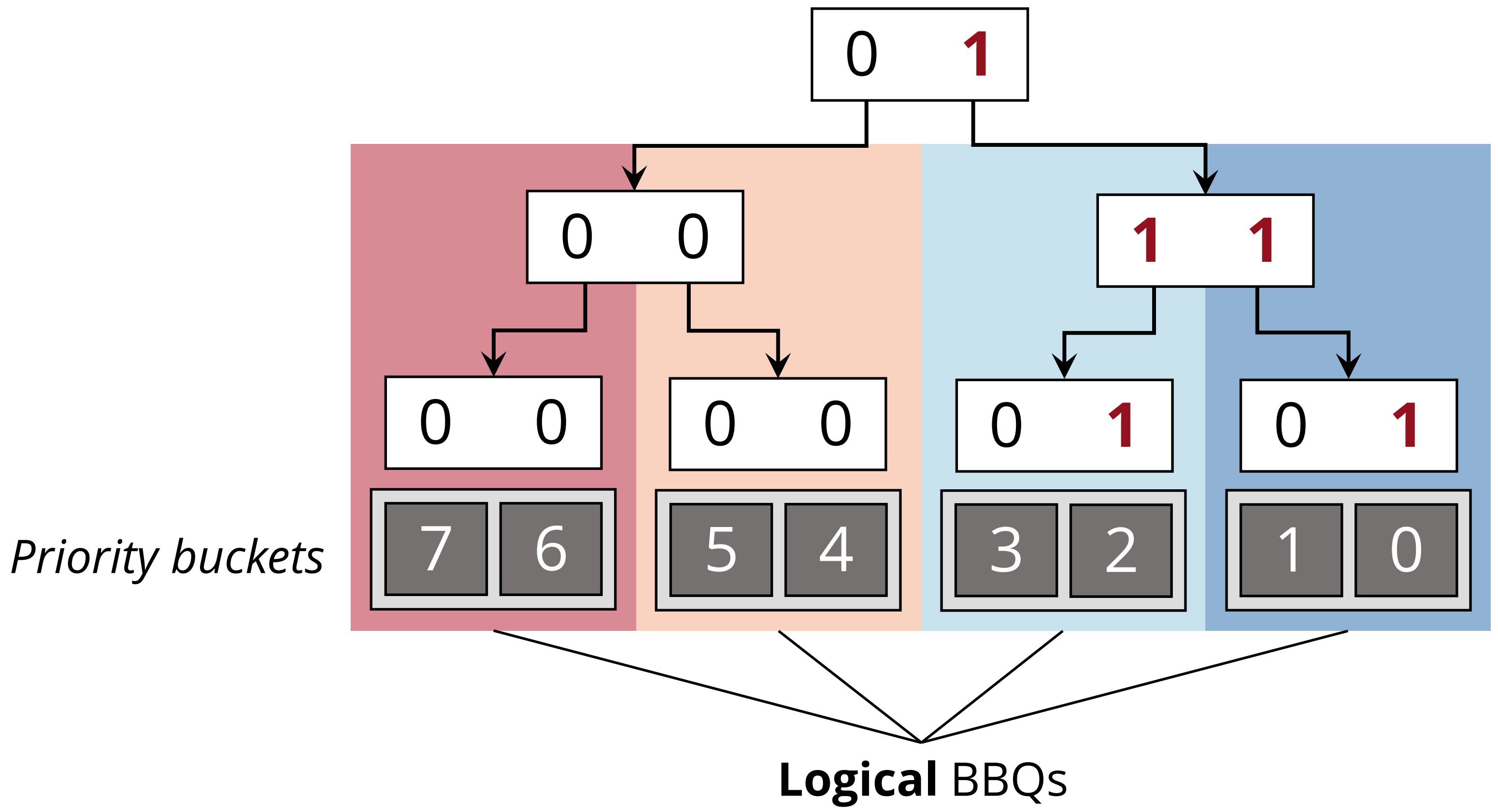
$$\boxed{\text{Performance (ops/sec)} = f_{max} \text{ (cycles/sec)} \times \text{OPC (ops/cycle)}}$$

BBQ is *highly optimized* for performance

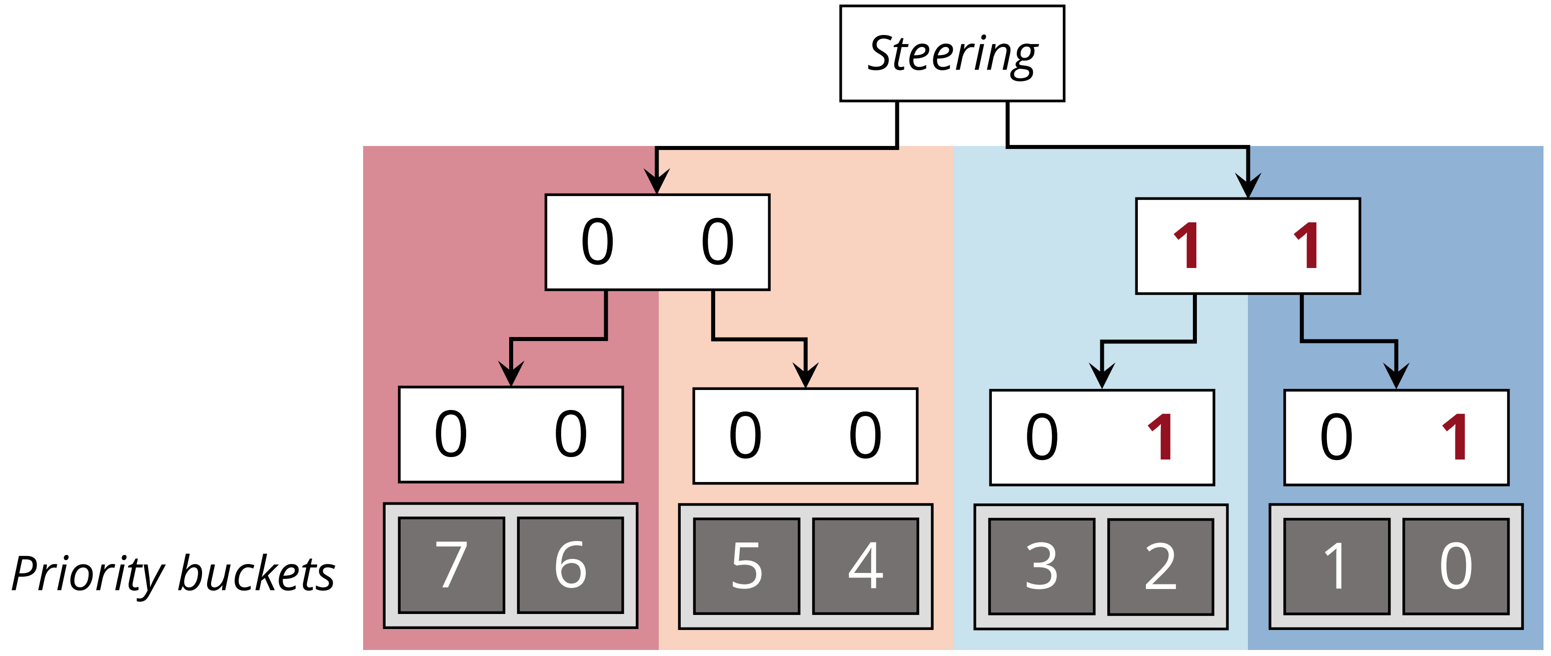
Pipelining is **hard**!



BBQ supports logical partitioning with zero fragmentation and performance overhead



BBQ supports logical partitioning with zero fragmentation and performance overhead



How does BBQ meet our requirements?

Flow Count Scalability

IPQ-based design allows scaling to O(100K) flows

Single-Instance Performance

Highly optimized, fully-pipelined design allows BBQ to support 150 Mpps (100 Gbps) on FPGAs and 1.5 Bpps (1 Tbps) on switch ASICs

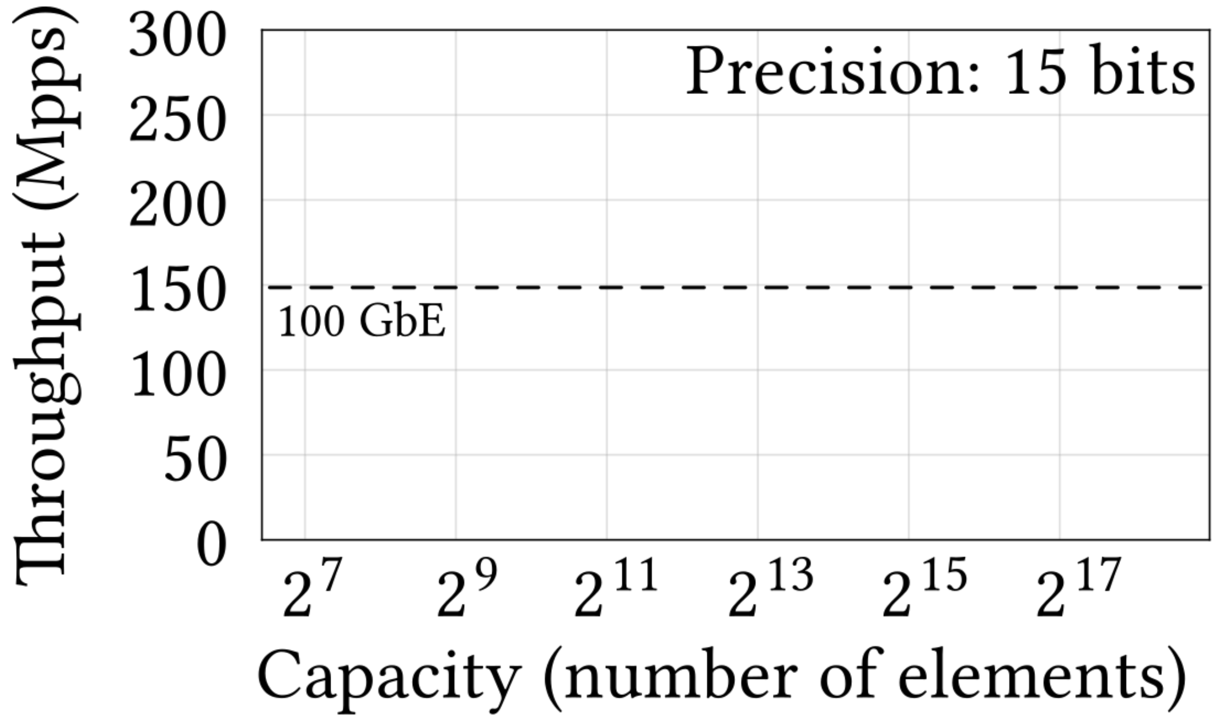
Logical Partitioning

BBQ's **unique priority index structure** allows logical partitioning with no performance overhead

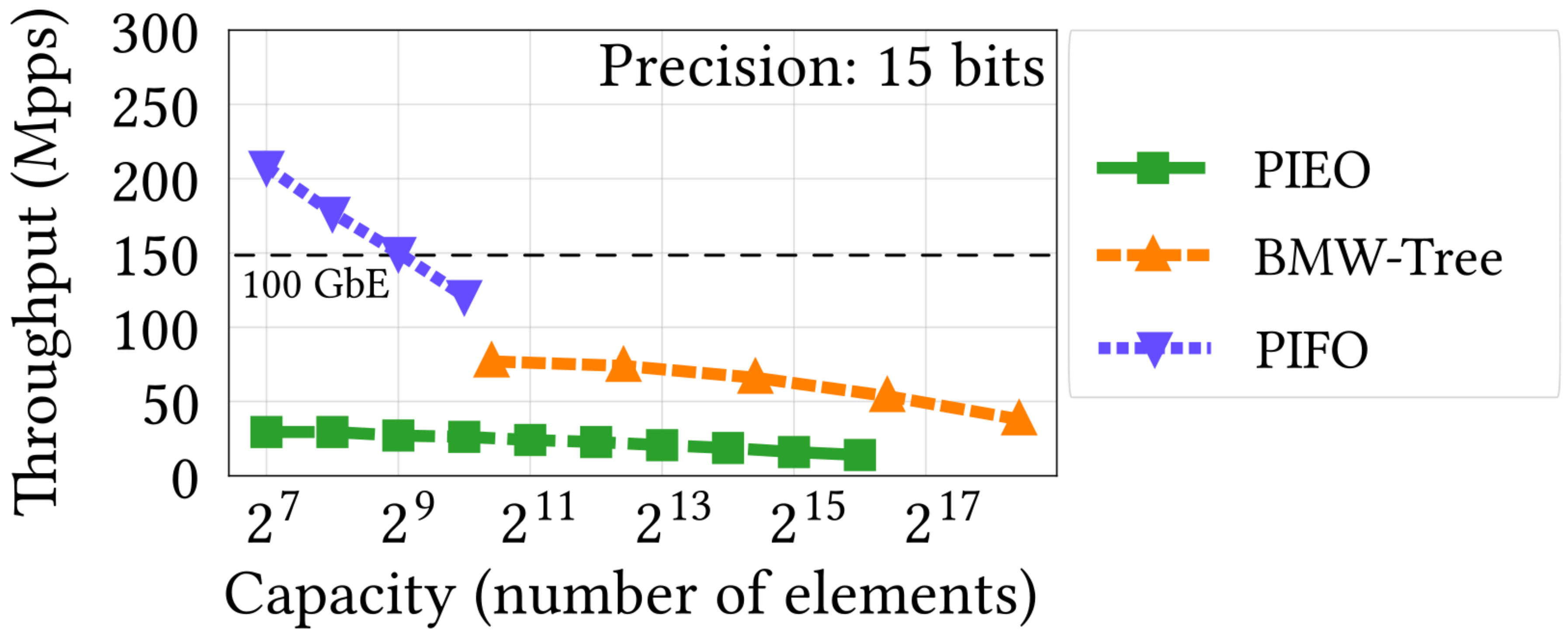
This Talk

- Minimum requirements for scheduling in switches and SmartNICs
- State-of-the-art priority queue designs are infeasible
- How do we get there?
- **Evaluation**

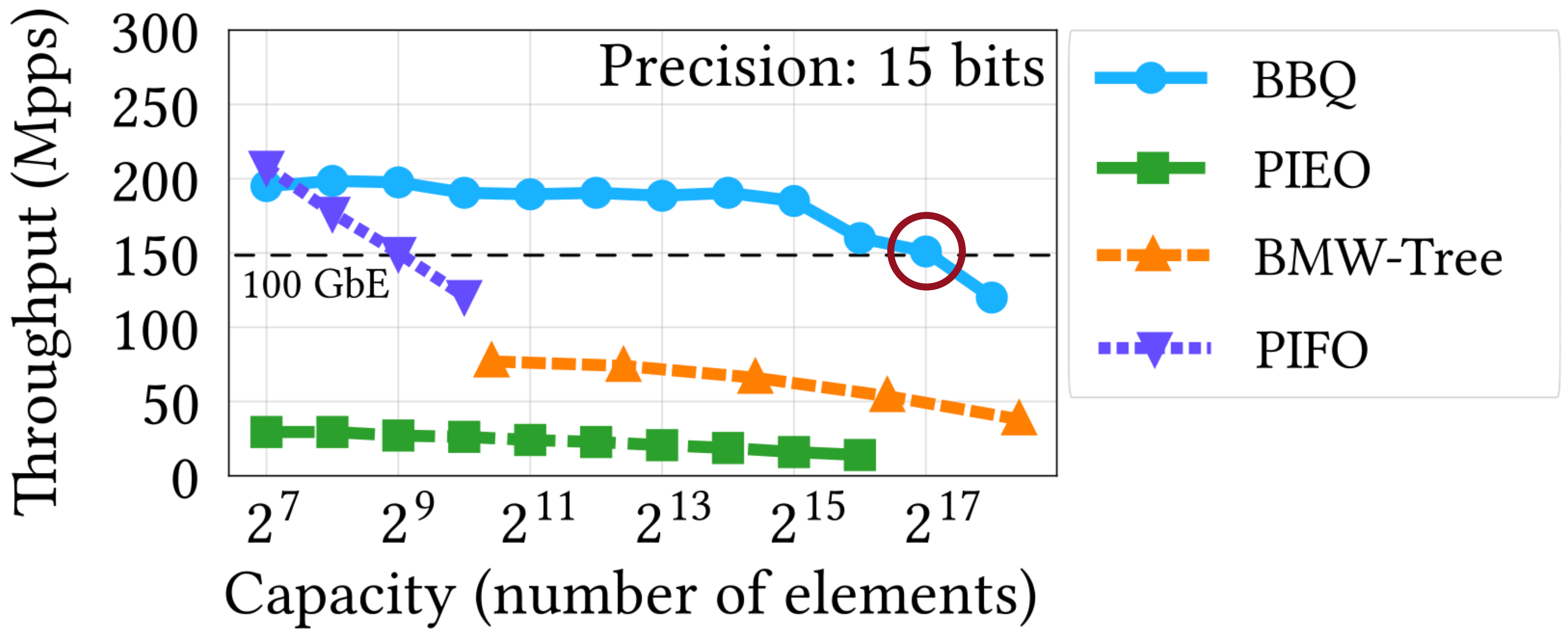
BBQ Evaluation (FPGA): Performance



BBQ Evaluation (FPGA): Performance



BBQ Evaluation (FPGA): Performance



On a Stratix 10 FPGA, BBQ sustains **100 Gbps line rate** (148.8 Mpps) with 100K+ elements and 32K priorities, 3X the packet rate of state-of-the-art designs.

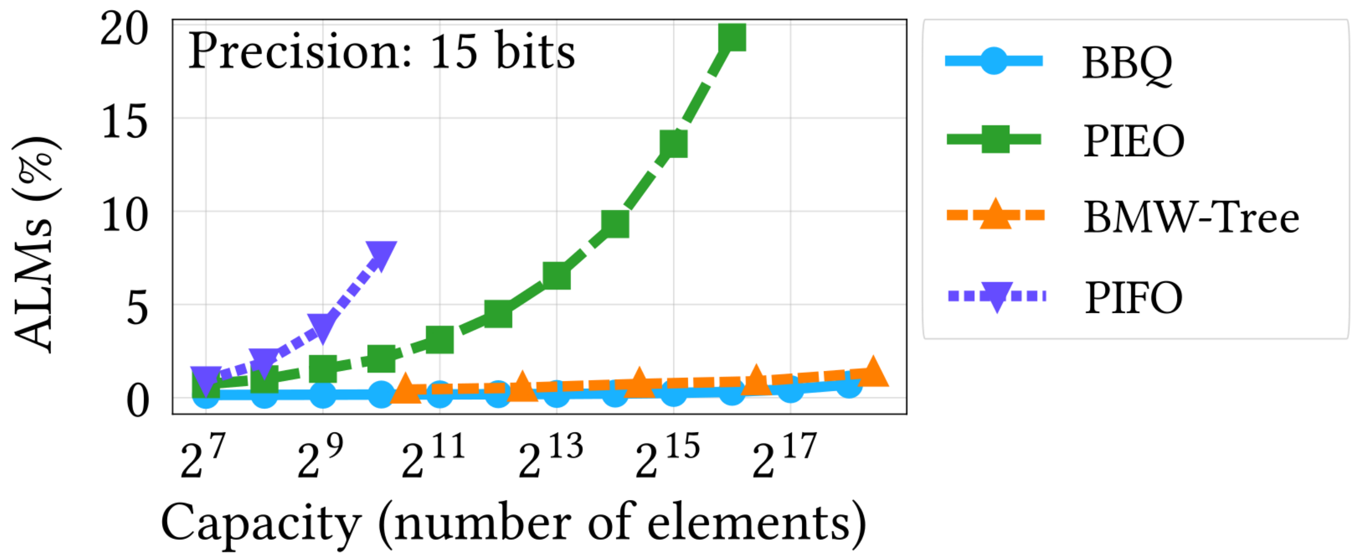
Conclusion

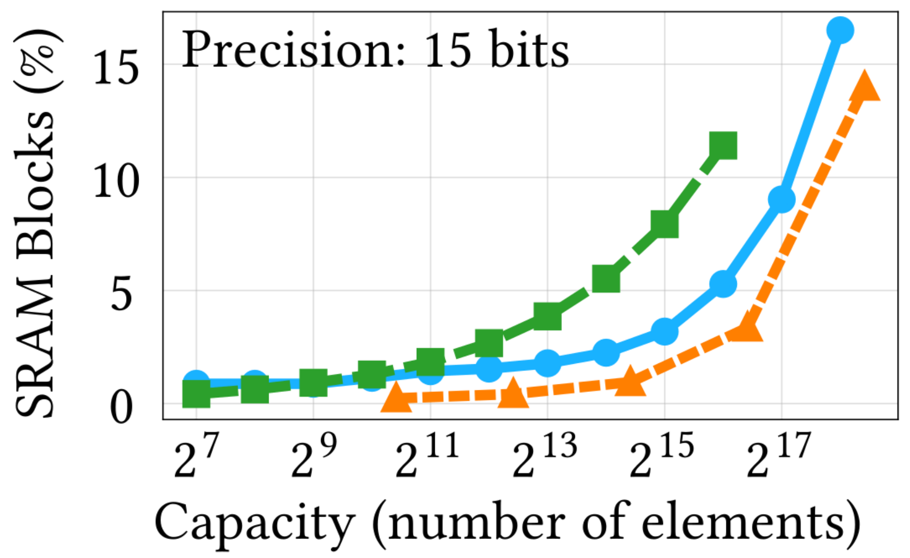
Existing hardware priority queues do not meet the stringent requirements imposed by modern schedulers. We design BBQ, an IPQ that – for the first time – makes it *feasible* to implement priority packet scheduling on line rate switches and SmartNICs.

BBQ supports 100K+ entries and 32K priorities at 100 Gbps line-rate (148.8 Mpps) on an FPGA, and 1 Tbps (1.5 Bpps) on an ASIC.

Open-source code:
https://github.com/cmu-snap/BBQ

BBQ Evaluation (FPGA): Resources





BBQ requires very few ALMs. Its SRAM usage is between PIEO and BMW-Tree (but requires fewer copies to meet the same performance target)

How does BBQ fit in the context of approximate priority queue designs?

1. BBQ is *complementary* to approaches that assume a small set of priority queues as a hardware primitive (SP-PIFO, PCQ, Sifter)
   - Accuracy improves with more strict-priority queues
   - BBQ's priority index structure (bitmap tree) is an efficient priority decoder, which is how we can scale to larger priority spans

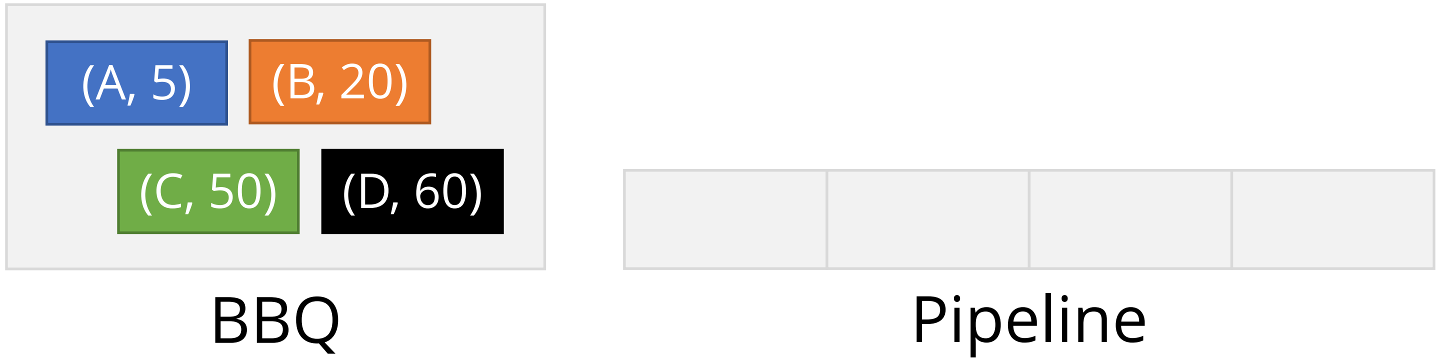
2. BBQ's design shows that is possible to scale to a large number of queue elements without sacrificing accuracy

Accuracy

Want the execution output of BBQ to be identical to an “ideal” priority queue… *unfortunately, pipelining breaks this property!*

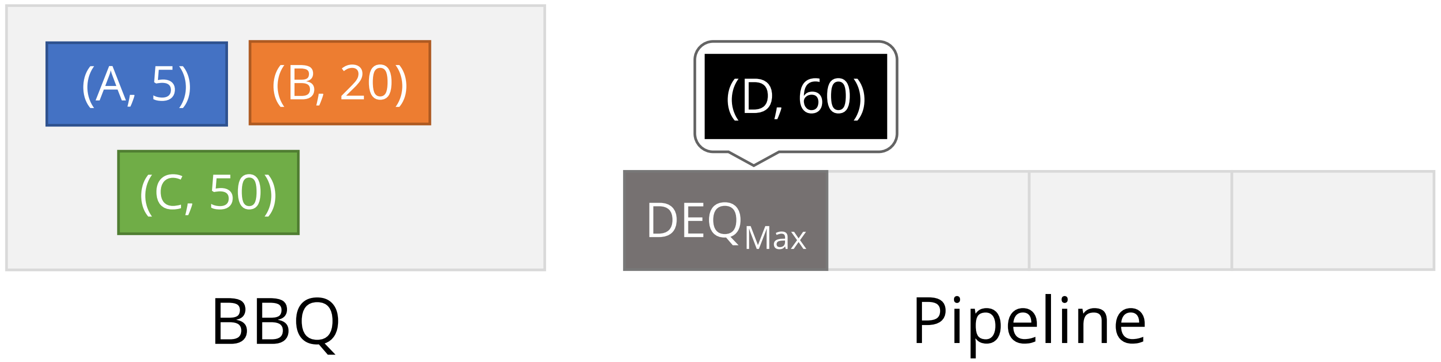
Accuracy

Want the execution output of BBQ to be identical to an "ideal" priority queue… *unfortunately, pipelining breaks this property!*



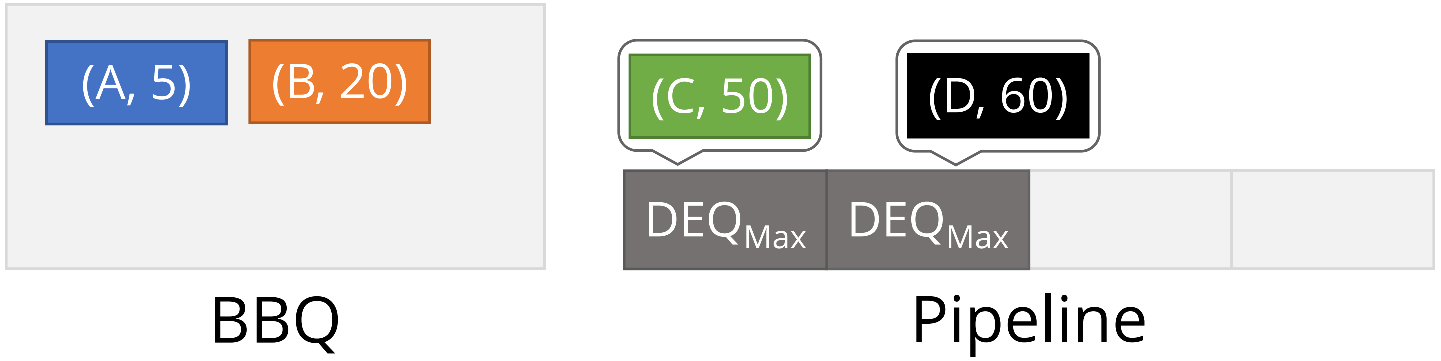
Accuracy

Want the execution output of BBQ to be identical to an "ideal" priority queue… *unfortunately, pipelining breaks this property!*



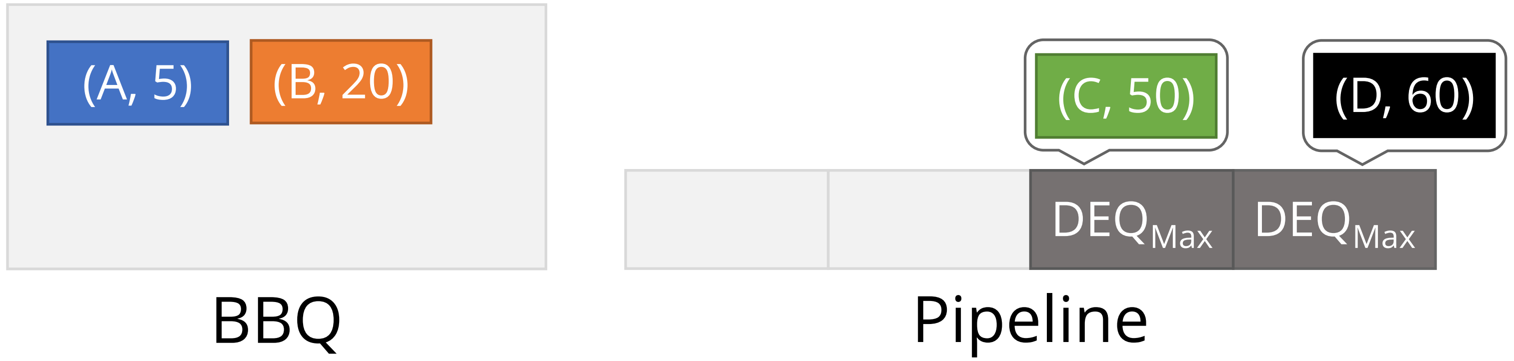
Accuracy

Want the execution output of BBQ to be identical to an “ideal” priority queue... *unfortunately, pipelining breaks this property!*



Accuracy

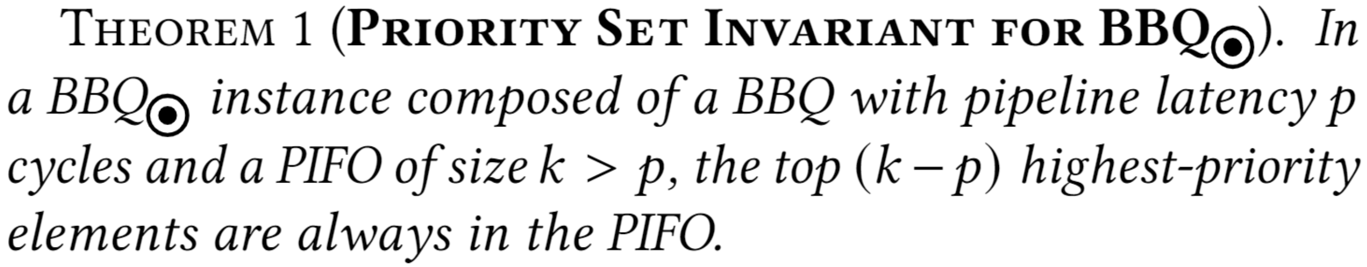
Want the execution output of BBQ to be identical to an “ideal” priority queue… *unfortunately, pipelining breaks this property!*



The highest-priority element in the system is not always in the BBQ, creating potential scheduling inaccuracies.

Accuracy

We prove that combining BBQ with a tiny PIFO recovers accuracy. The composite design has all the nice properties of BBQ, but without the pipeline latency.



THEOREM 1 (**PRIORITY SET INVARIANT FOR** $\mathbf{BBQ_{\odot}}$). *In a* $\mathit{BBQ}_{\odot}$ *instance composed of a BBQ with pipeline latency p cycles and a PIFO of size $k > p$, the top $(k - p)$ highest-priority elements are always in the PIFO.*