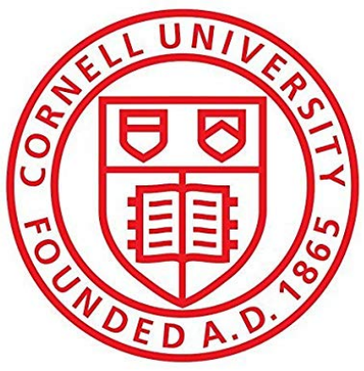
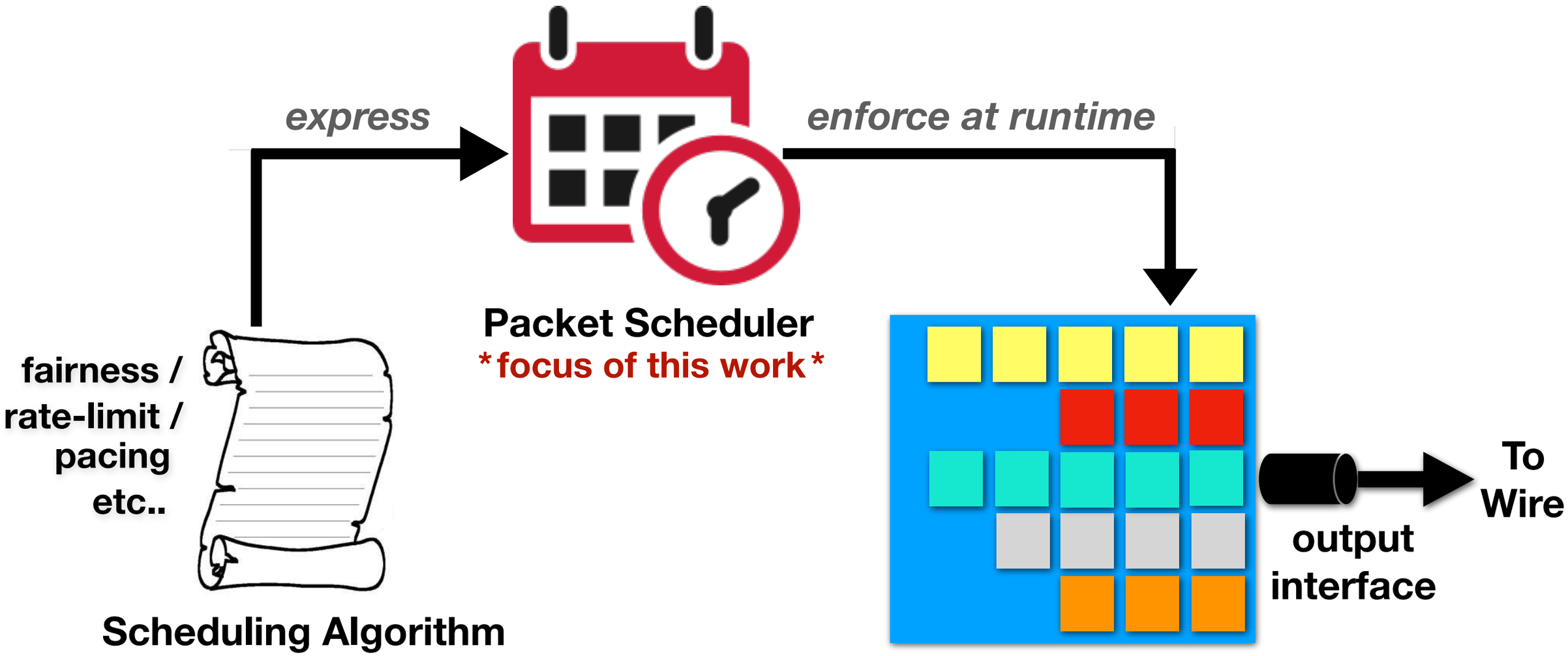
Fast, Scalable, and Programmable Packet Scheduler in Hardware

Vishal Shrivastav
Cornell University



Packet Scheduling 101



specifies when and what order
to schedule packets
onto the wire

Desirable Properties of a Packet Scheduler

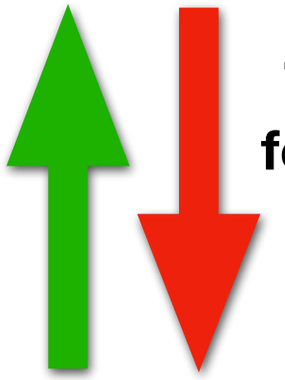
Programmability

Express wide-range of packet scheduling algorithms

Scalability

Scale to 10s of thousands of flows
[SENIC - NSDI'14] [Carousel - SIGCOMM'17]

Performance



e.g., 120 ns for MTU pkt @ 100Gbps

New transport protocols
e.g., Fastpass, Ethernet TDMA

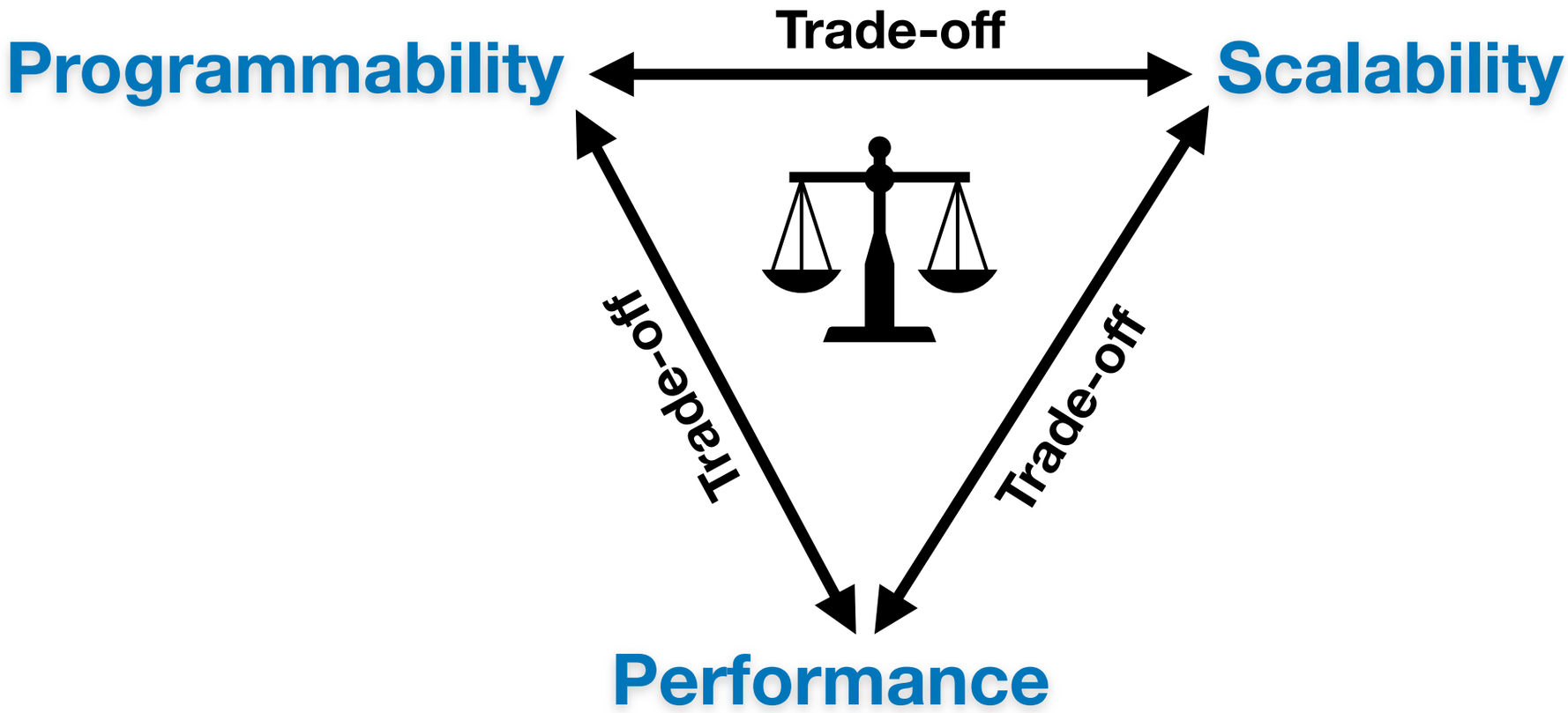
Circuit-Switched network designs
e.g., Shoal, RotorNet

↓

Transmit packets at precise times
e.g., at ns-precision in Shoal

Make scheduling decisions within deterministic 10s of nanoseconds

Desirable Properties of a Packet Scheduler



Challenging to achieve all three properties (programmability, scalability, and performance) simultaneously

State-of-the-art Packet Schedulers

	Programmability express wide range of scheduling algorithms	Scalability 10s of thousands of flows	Performance decisions within deterministic 10s of nanoseconds
Software Generality ↑ / Performance ↓ *via specialization* **Hardware**	✓	✓	✗
1. FIFO	✗	✓	✓
2. PIFO, UPS **(priority queue abstraction)**	✓*	✗	✓

*has some limitations

Can we design a packet scheduler that is simultaneously programmable, scalable, and high performance?

We present PIEO (Push-In-Extract-Out) scheduler in hardware

Abstraction

Programmable
more expressive than any state-of-the-art hardware packet scheduler

Hardware Design

Scalable
easily scales to 10s of thousands of flows

High Performance
makes scheduling decisions in O(1) time [4 clock cycles]

Can we design a packet scheduler that is simultaneously programmable, scalable, and high performance?

We present PIEO (Push-In-Extract-Out) scheduler in hardware

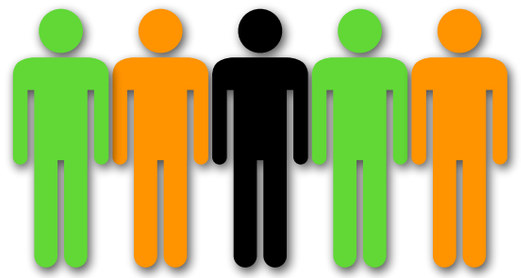
Programmable — Abstraction
more expressive than any state-of-the-art hardware packet scheduler

Scalable
easily scales to 10s of thousands of flows

High Performance
makes scheduling decisions in O(1) time [4 clock cycles]

PIEO Scheduling Abstraction

Scheduling Algorithms



<u>when</u> an element becomes eligible for scheduling?

encode using a $t_{eligible}$ value

<u>what order</u> to schedule amongst eligible elements?

encode using a $rank$ value

whenever the link is idle:

among all elements satisfying the eligibility predicate $t_{current} \geq t_{eligible}$:

schedule the smallest ranked element

PIEO Abstraction — *"schedule the smallest ranked eligible element"*

strictly more expressive than a priority queue abstraction, e.g., PIFO, UPS

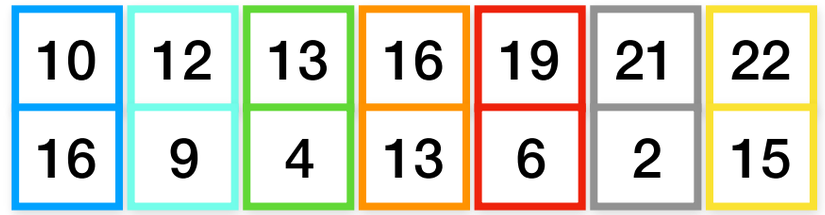
Push-In-Extract-Out Primitive

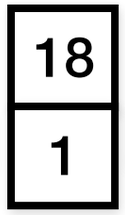
programmed based on the choice of scheduling algorithm

element

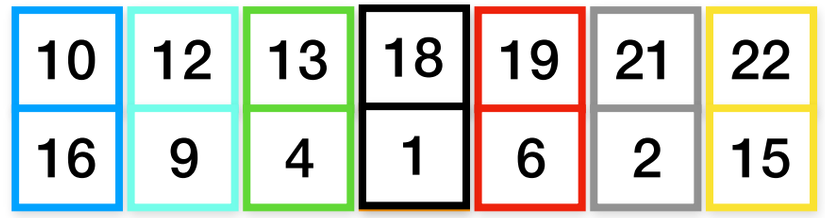
ordered list

increasing rank value



enqueue() "Push-In"

inserts element at position dictated by its rank value

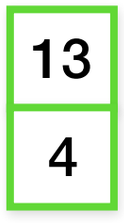


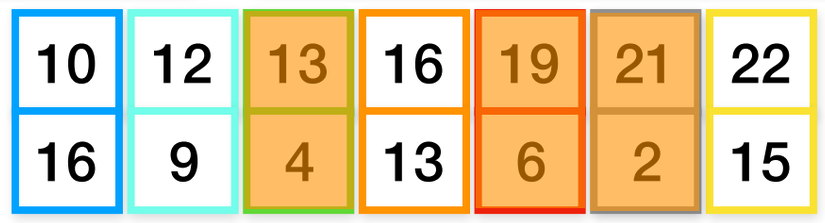
$t_{current} = 7$

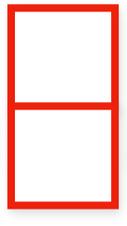
filter : $t_{current} \geq t_{eligible}$

dequeue( ) "Extract-Out"

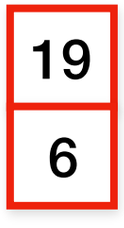
returns "smallest ranked eligible" element

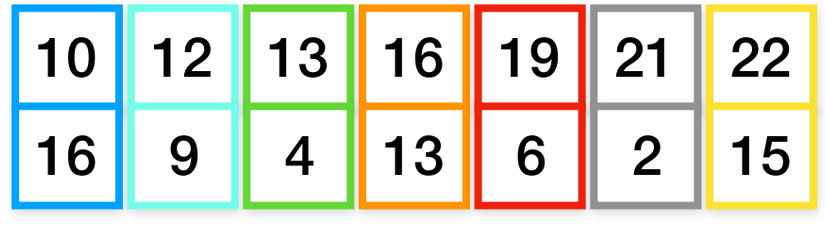




dequeue()

returns a specific element





Expressiveness of PIEO

- **Work conserving**
  - e.g., DRR, WFQ, WF^2Q
- **Non-work conserving**
  - e.g., Token Bucket, RCSP
- **Hierarchical scheduling**
  - e.g., HPFQ
- **Asynchronous scheduling**
  - e.g., Starvation avoidance, D^3
- **Priority scheduling**
  - e.g., SJF, SRTF, LSTF, EDF
- **Complex scheduling policies**
  - mixture of shaping and ordering

for each element:
calculate **start_time** and **finish_time**
at time x, all elements s.t. **virtual_time(x) >= start_time:**
schedule element with **smallest finish_time**

programming PIEO

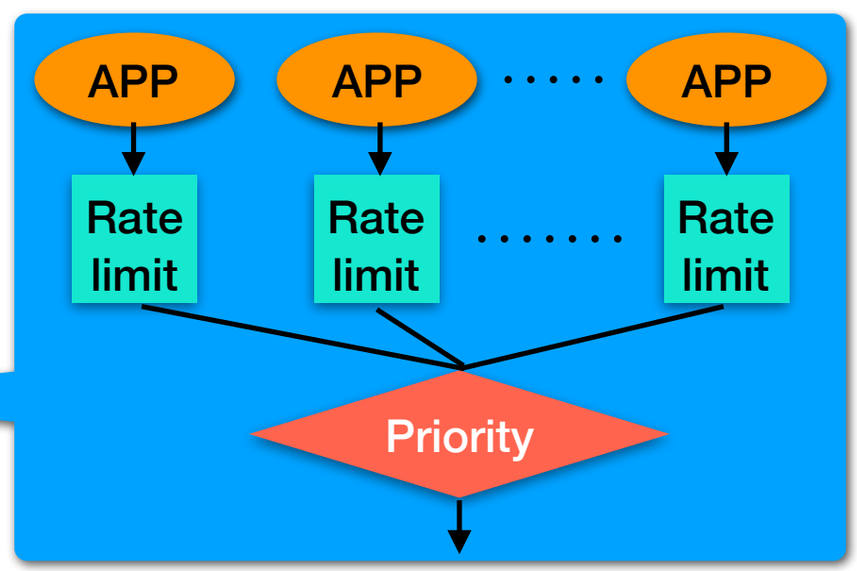
$rank = finish\_time$

$t_{eligible} = start\_time$

Predicate for filtering at dequeue at time x:

$(virtual\_time(x) \geq t_{eligible})$

e.g.



— **can not express accurately using PIFO**

Can we design a packet scheduler that is simultaneously programmable, scalable, and high performance?

We present PIEO (Push-In-Extract-Out) scheduler in hardware

Programmable

more expressive than any state-of-the-art hardware packet scheduler

Hardware Design

Scalable

easily scales to 10s of thousands of flows

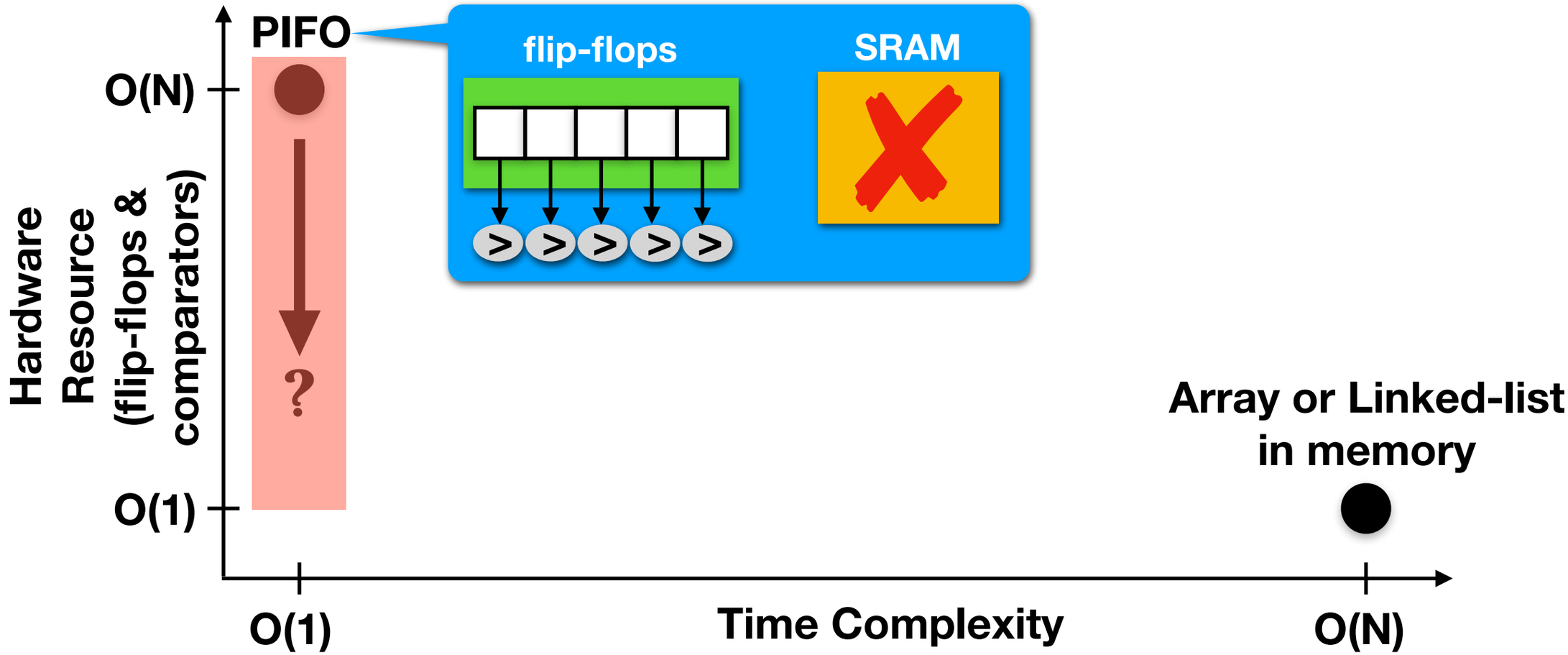
High Performance

makes scheduling decisions in O(1) time [4 clock cycles]

Hardware Design

PIEO primitive relies on an ordered list datastructure

Performance ⟵ ⟶ **Scalability**



Is it fundamentally necessary to access and compare $O(N)$ elements in parallel to maintain an (exact) ordered list (of size N) in $O(1)$ time?

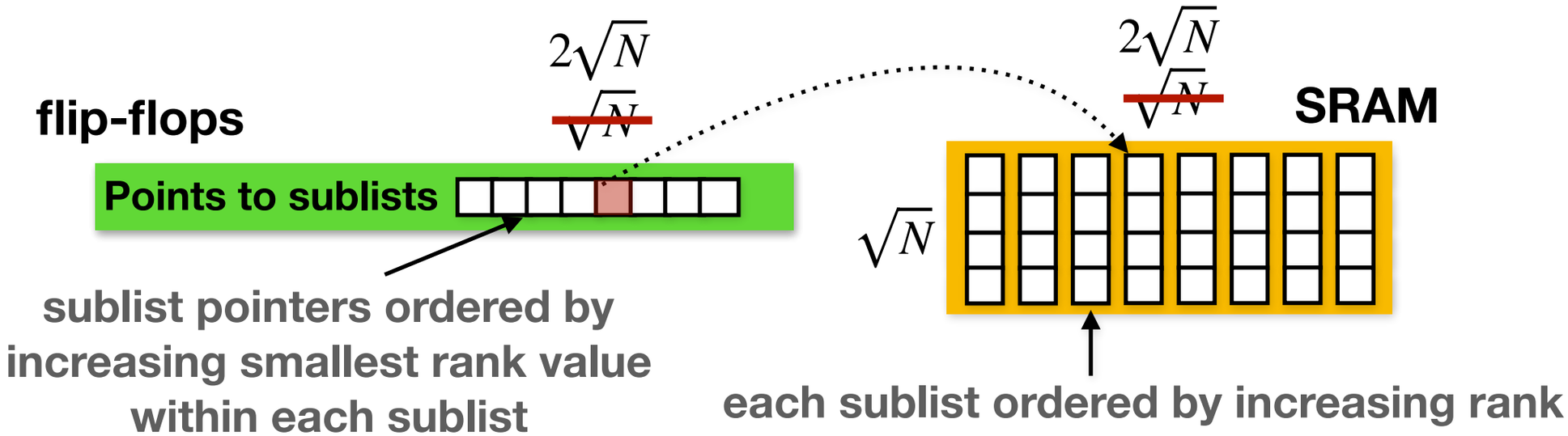
We present a design that can maintain an (exact) ordered list in $O(1)$ time, but only needs to access and compare $O(\sqrt{N})$ elements in parallel.

Key Insight

"All problems in computer science can be solved by another level of indirection"

—David Wheeler

Hardware Architecture



Q: How to zoom into the correct sublist in O(1) time?
Q: How to read/update/write an entire sublist in O(1) time?
Q: How to filter + extract-min in O(1) time?
Q: What to do when enqueue into a full sublist?

Detailed answers in the paper !!!

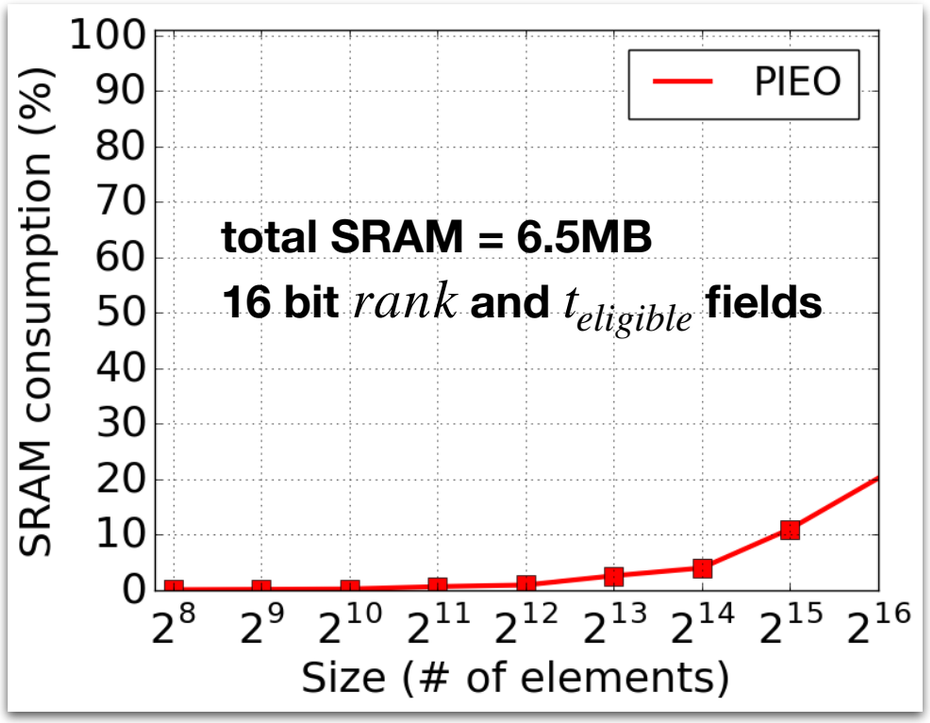
enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

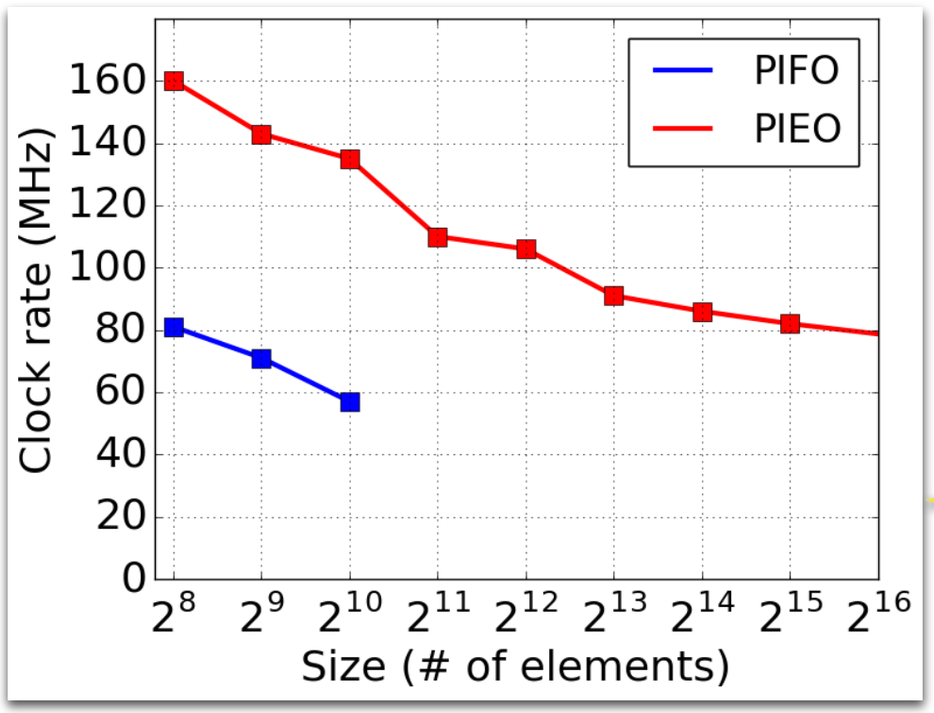
….. at the cost of 2X memory overhead

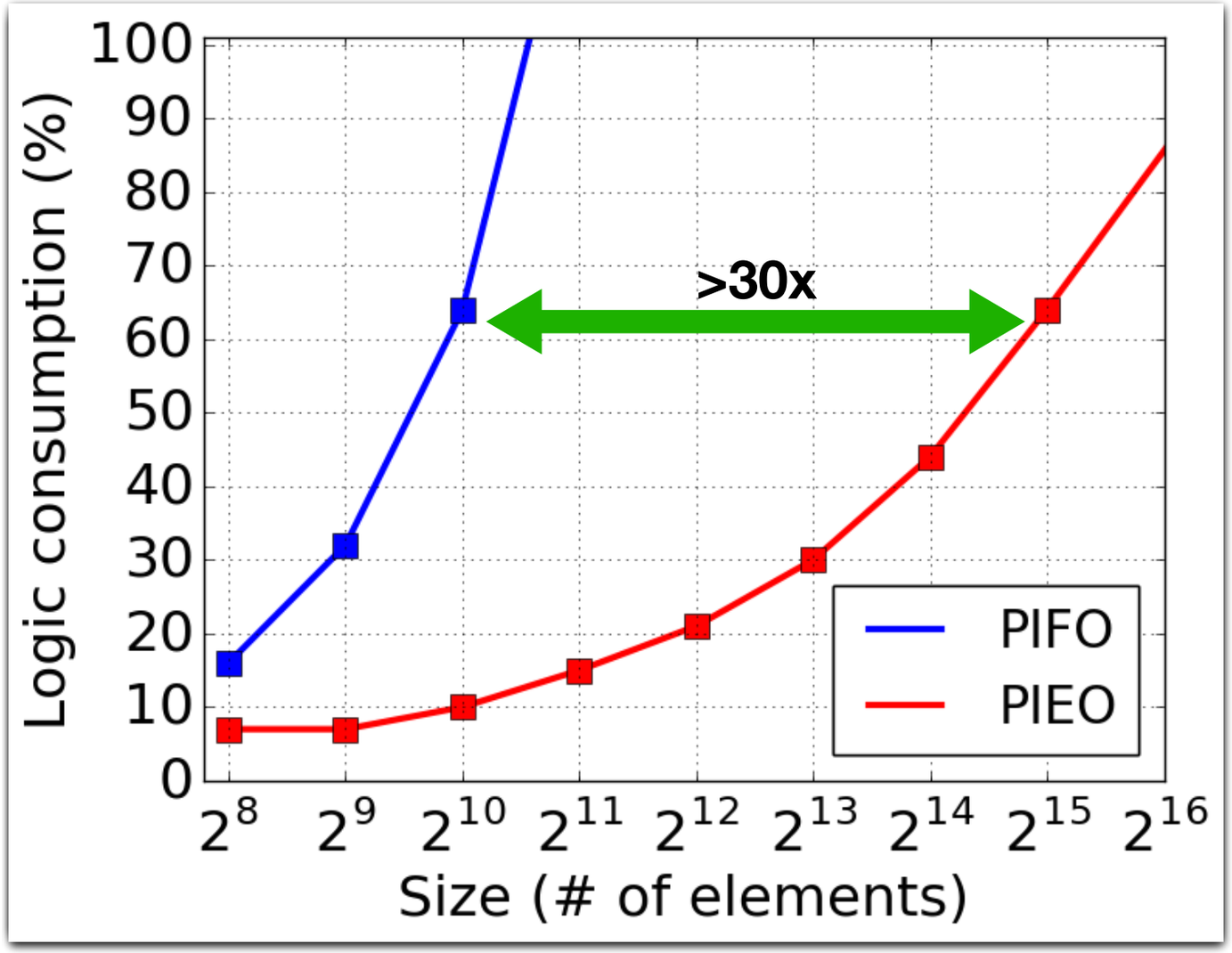
Implementation

- **Implemented PIEO scheduler on a Stratix V FPGA**
  - 234K logic modules (ALMs)
  - 6.5MB SRAM
  - 40Gbps interface bandwidth
- **~1300 LOCs in System Verilog**

Evaluation







4 cycles per primitive op, i.e., 50ns @ 80MHz
- pipelining
- ASIC target, e.g., 4ns @ 1GHz

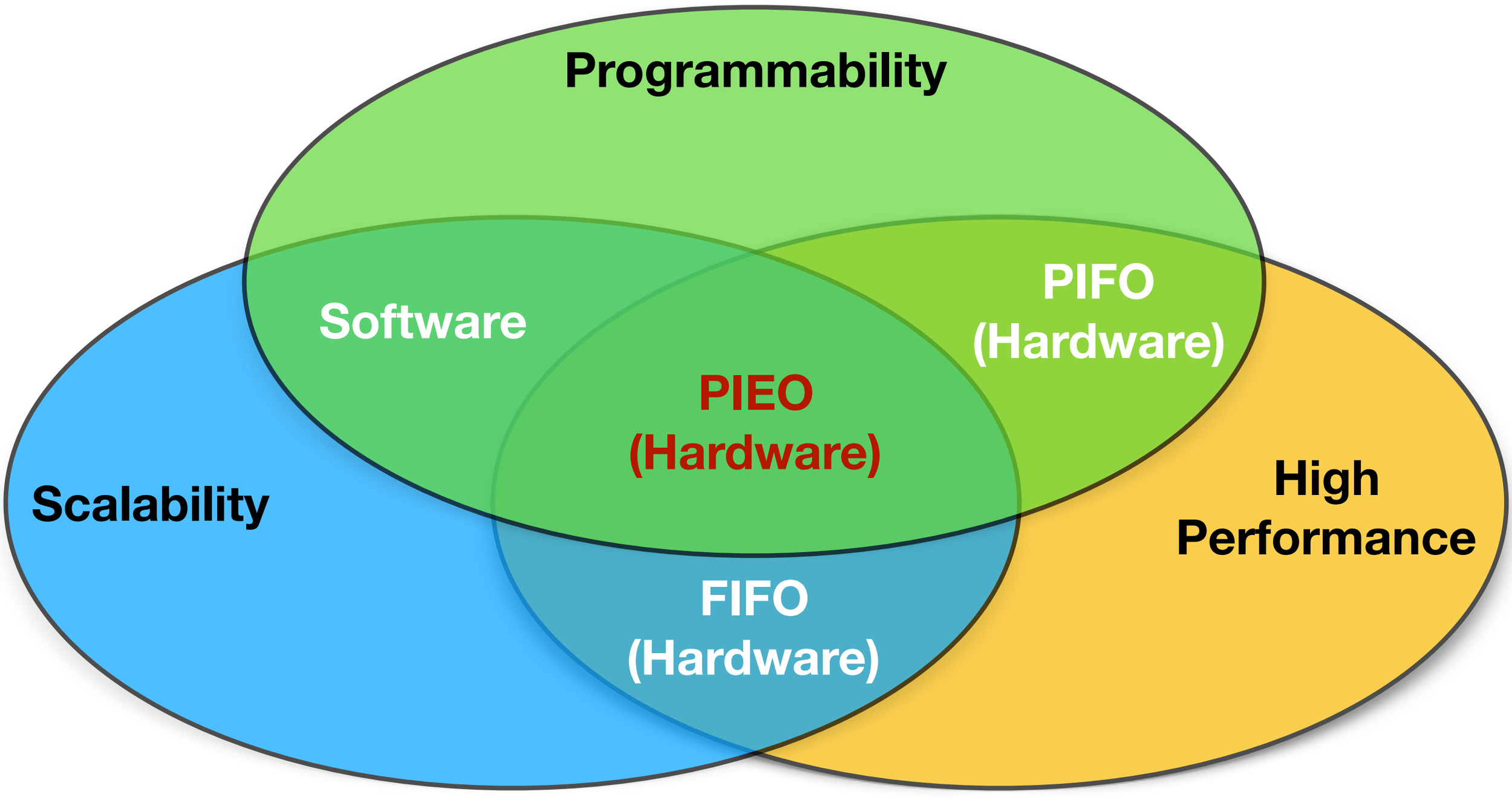
But not as fast as PIFO — 1 cycle per primitive op

Beyond Packet Scheduling

- PIEO as an O(1)-time ***generic Priority Queue***
- PIEO as an ***Abstract Dictionary Data Type***
  - act as a **(key, value)** store, indexed by keys
  - **search**, **insert**, **delete**, and **update** in O(1) time
  - efficiently do complex ops like **range filtering** over keys
  - ….. while also being reasonably scalable

PIEO as a key basic building block in the era of hardware-accelerated computing

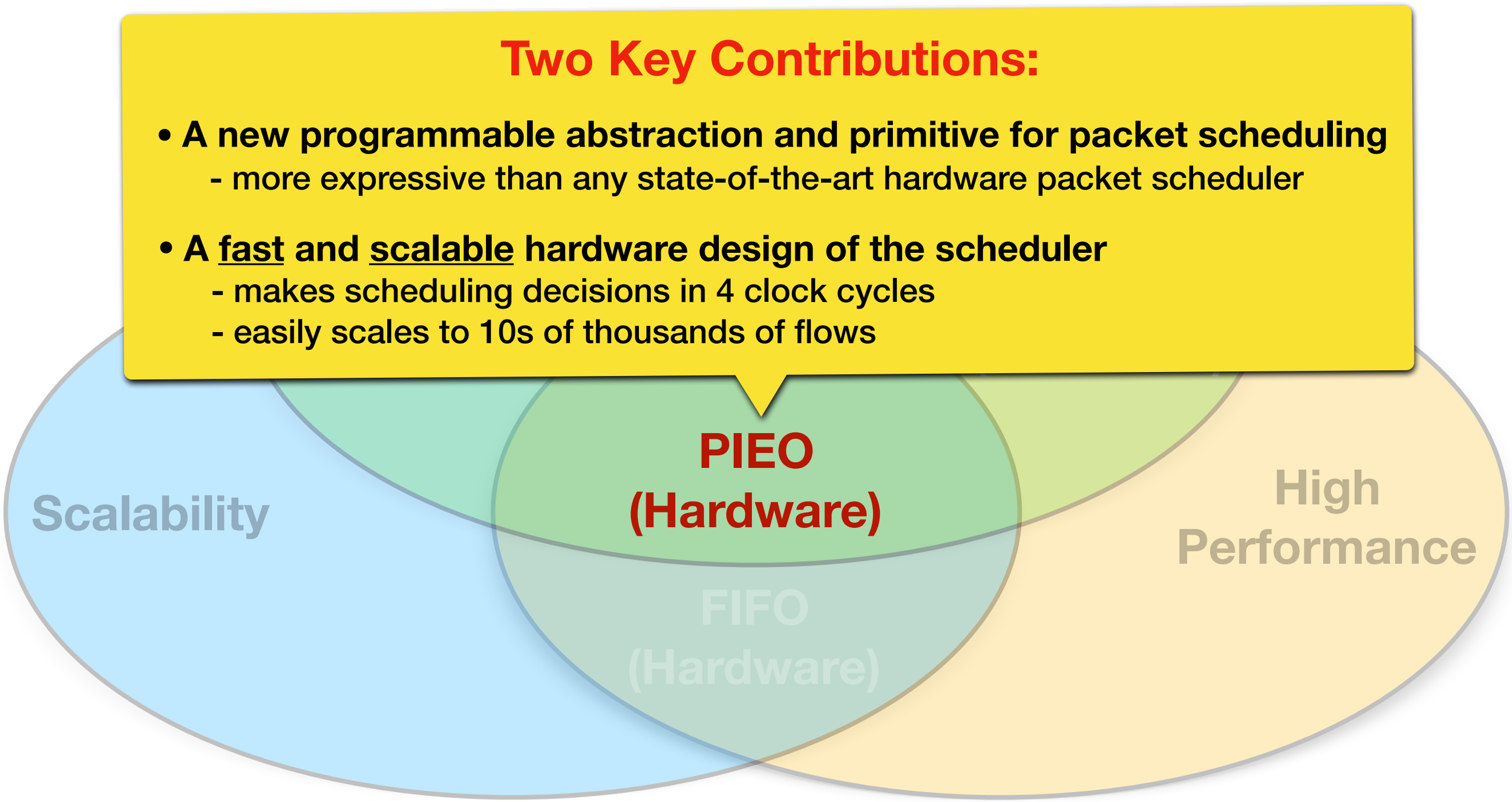
Conclusion


Programmability
Software
PIFO
(Hardware)
PIEO
(Hardware)
Scalability
High
Performance
FIFO
(Hardware)

Conclusion

Two Key Contributions:

- **A new programmable abstraction and primitive for packet scheduling**
  - more expressive than any state-of-the-art hardware packet scheduler
- **A fast and scalable hardware design of the scheduler**
  - makes scheduling decisions in 4 clock cycles
  - easily scales to 10s of thousands of flows



Thank you!

FPGA code for the implementation of PIEO scheduler is available at:
https://github.com/vishal1303/PIEO-Scheduler

Email: vishal@cs.cornell.edu
Webpage: http://www.cs.cornell.edu/~vishal/