Shared Memory Pool for Representors

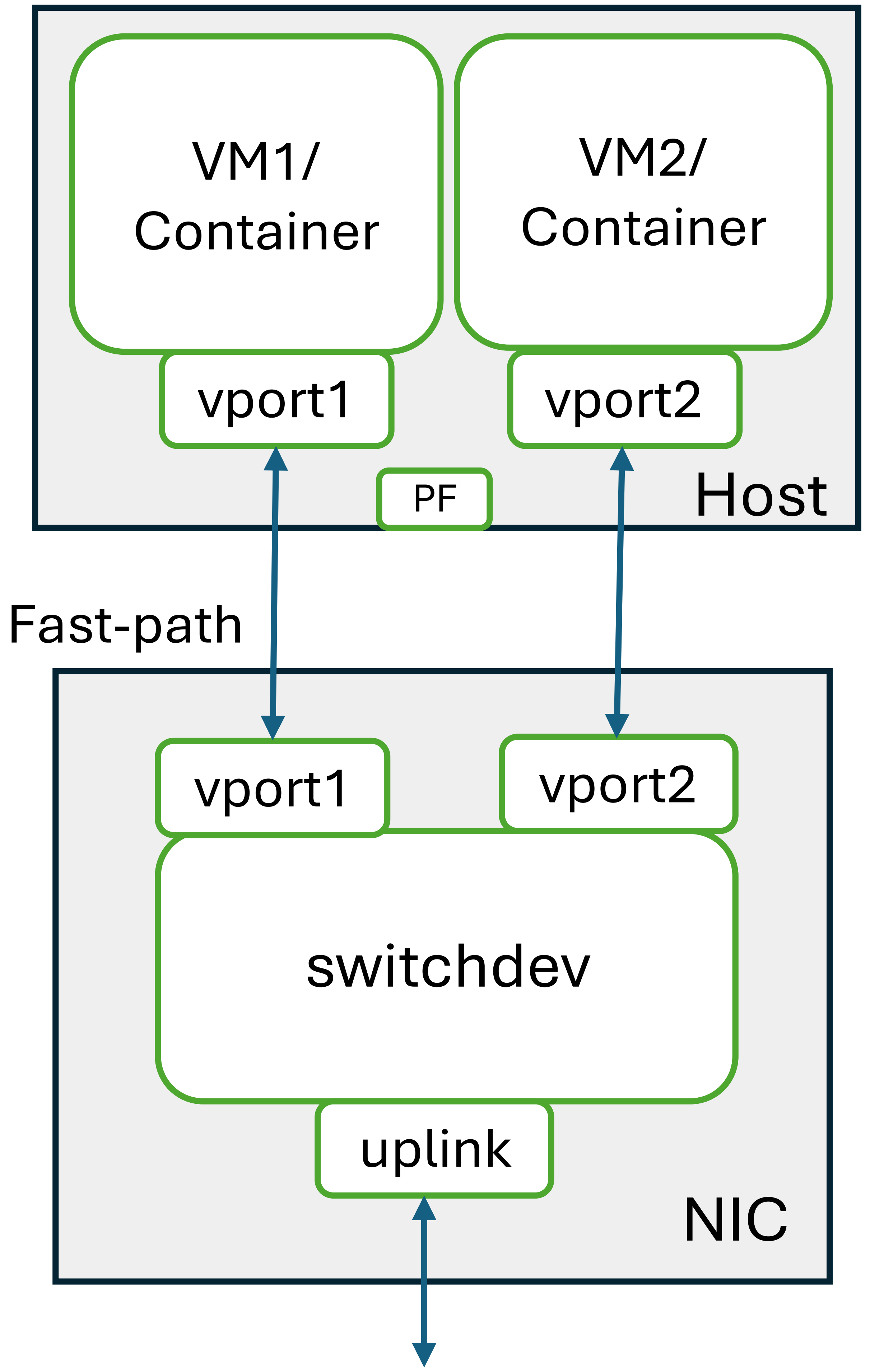
William Tu, Michal Swiatkowski, and Yossi Kuperman
Nvidia and Intel

NetDev 0x18, 2024

Switchdev Mode

Fast Path

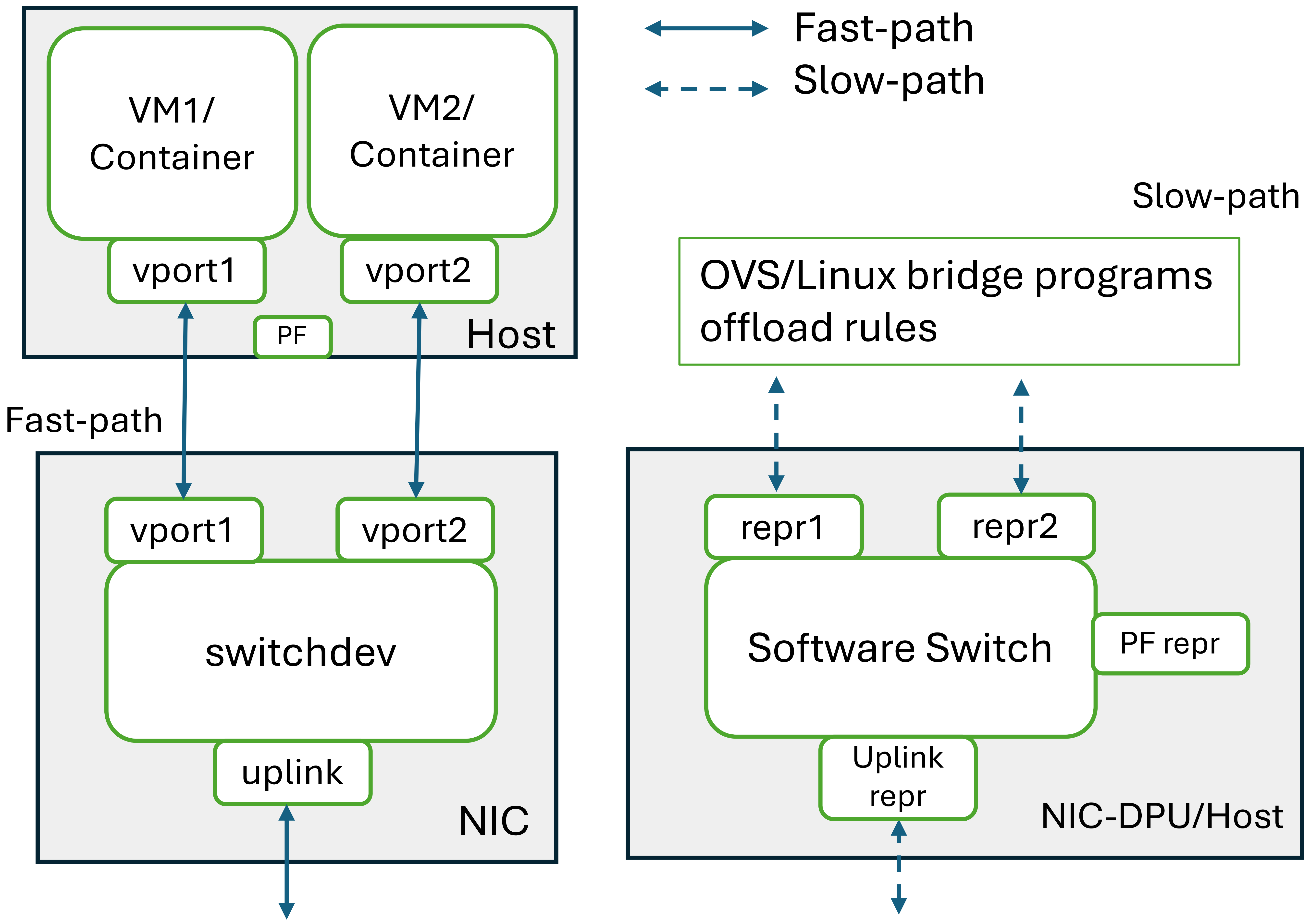
- An embedded switch in NIC
- Legacy mode supports basic L2 features (mac/vlan)
- Switchdev supports advanced hardware offloads
- Vports (VFs/SFs) are switchdev ports and connected to VM
- Handle most of the traffic in hardware



Switchdev Mode

Slow-Path

- Runs on host, or in SmartNIC embedded CPU
- Each vport has its own representor port (repr)
- Repr is the control plane of the vport (representee)
- Reprs attached to OVS or Linux bridge
- Handles first couple packets of a connection
- Insert/delete/update rules into switchdev



Slow-Path Design Challenges

When creating thousands of SFs/VFs:

- Each VF/SF has its own representor netdev, 1:1 mapping
- Each representor netdev has its own RXQs, TXQs

Challenges

- NIC does not have enough hardware queues
    - ICE supports up to 1K queues
- Consume too much memory
    - Memory is not enough on SmartNIC

🤔 Do we need a dedicated netdev for just handling slow path traffic?

Slow-Path Design Challenges

When creating thousands of SFs/VFs with representors: → Design-0: Dedicated repr netdev

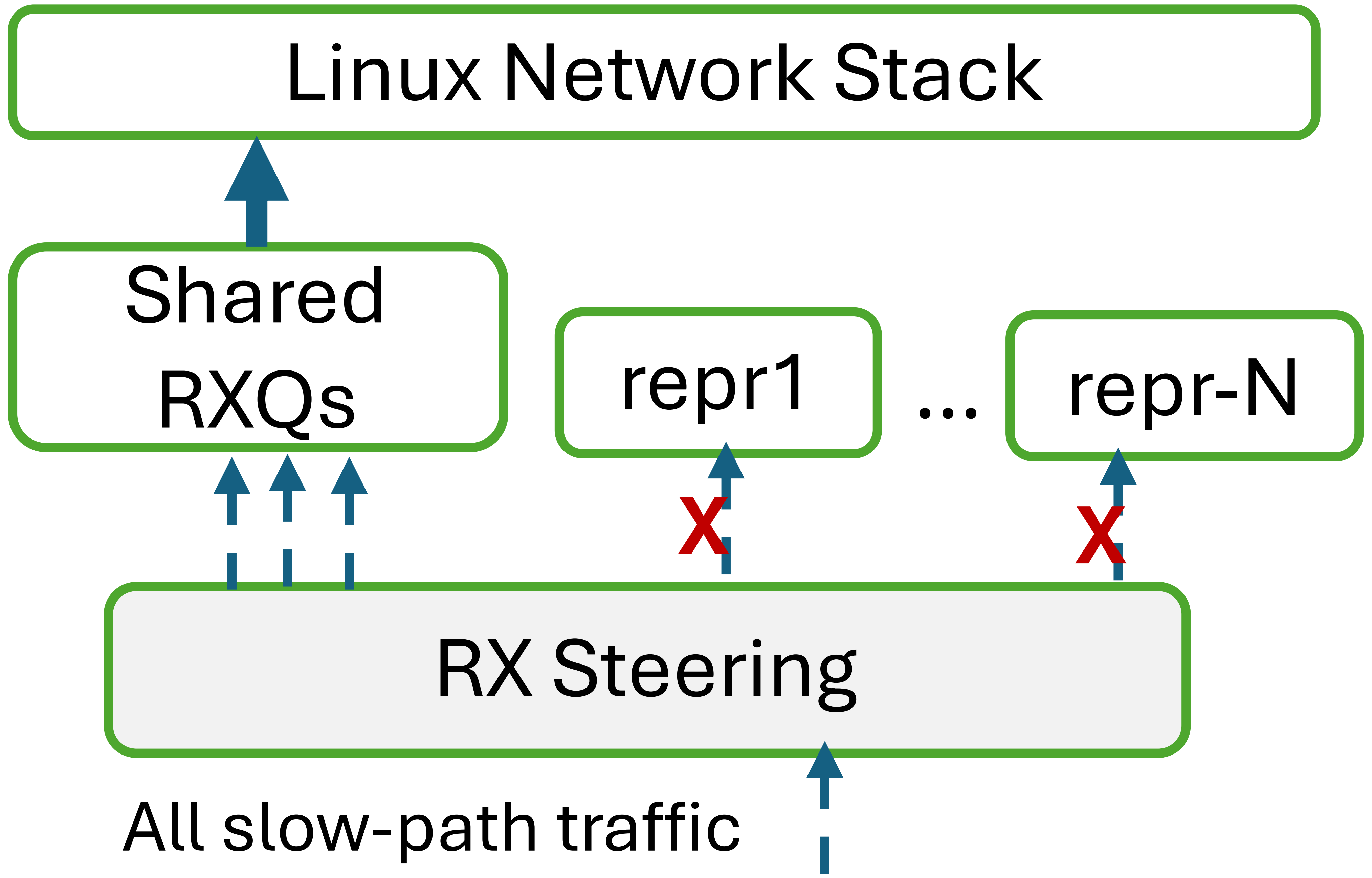
- Each VF/SF has its own representor netdev, 1:1 mapping
- Each representor netdev has its own RXQs

Challenges	Solutions
NIC does not have enough hardware queues	Design-1: Shared RXQs
Consume too much memory	Design-2: Adjustable RXQ for dedicated repr netdev

Design-1: Shared RXQ of PF

Solve the Hardware Queues Limitation

- Don’t allocate any RXQs for representors
- Shared RXQ for all representors
- RX completion metadata indicates the incoming source vport id
- TX can also use shared TXQ
- Used by ice, nfp, sfc
- **Huge memory and queue saving**

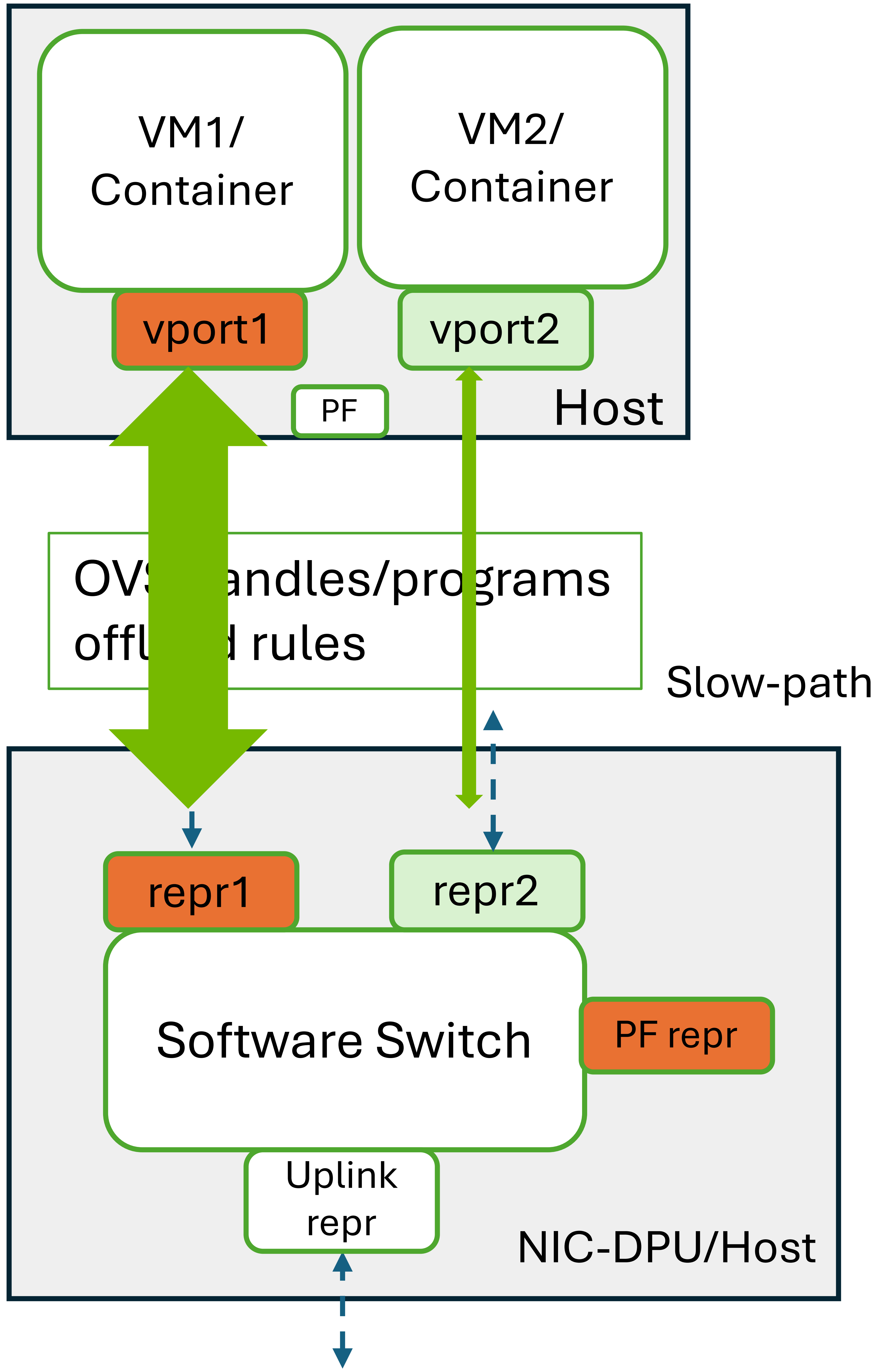


Design-1: Fairness Issue

Sharing causes Starving!

Assume traffic all goes into slow-path

- VM1 runs DPDK-pktgen
- VM2 runs ping
- All the buffers of shared RXQs are used by VM1
- VM2 get zero slow-path bandwidth ☹
- No performance isolation
- Why not use tc policing/shaping? Backpressure?



Slow-Path Design Challenges

2nd Challenge

When creating thousands of SFs/VFs with representors: → Design-0: Dedicated Repr netdev

- Each VF/SF has its own representor netdev, 1:1 mapping
- Each representor netdev has its own RXQs

Challenges	Solutions
NIC does not have enough hardware queues	Design-1: Shared RXQs
Consume too much memory	Design-2: Adjustable RXQ for dedicated repr netdev

Experiment (1/2)

How much memory a mlx5 representor netdev consumes?

- Create 200 SF-rep

```
for i in {100..200}; do
      devlink port add pci/0000:08:00.0 flavour pcisf pfnum 0 sfnum $i
done
```

- Setup RXQs and UP:

```
for i in {100..200}; do
     ethtool -L $dev combined 1  // number of channel/rxq
     ethtool -G $dev rx 1024     // RXQ depth
     ip link set dev $dev up
done
```

- Get memory differences

```
$ sar –r 1
04:51:08 PM kbmemfree   kbavail …
04:51:09 PM  31179532  31321476
```

Experiment (2/2)

How much memory a mlx5 representor netdev consumes?

- FW pages

```
 /sys/kernel/debug/mlx5/0000\:08\:00.0/pages/fw_pages_total
```

- Page pool:

```
./tools/net/ynl/cli.py --spec Documentation/netlink/specs/netdev.yaml
--dump page-pool-get
{'id': 20,
  'ifindex': 10,
  'inflight': 448, // pages
  'inflight-mem’: 1,835,008, // bytes
  'napi-id': 518},
```

- /proc/slabinfo, meminfo

MLX5 Representor Memory Consumption

- 1-RXQ, with 1024 RXQ depth: 2.86MB
- Page pool consumes 1.83MB out of 2.86MB
- 2 channels is around double: 5MB

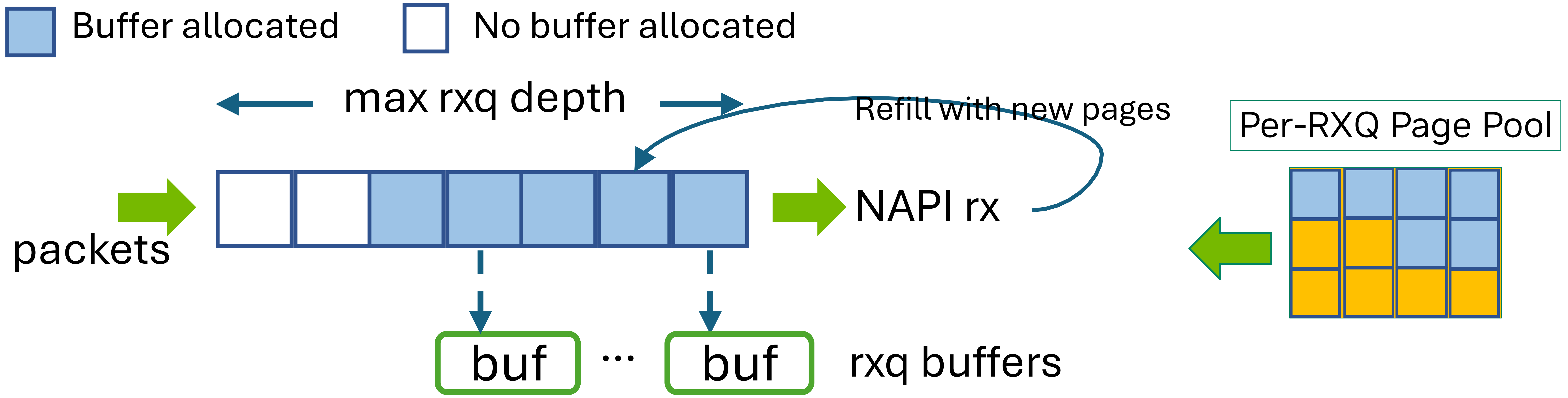
	1 RXQ total (MB)	Kernel Page Pool (MB) per-RXQ	FW (MB)	2 RXQs (MB)
Q128	1.1	0.2	0.113	2.835
Q256	1.3	0.4	0.114	2.915
Q512	1.805	0.78	0.114	2.99
Q1024	2.86	1.83	0.114	5.025
Q2048	4.935	3.93	0.115	9.25

🤔 Can we allocate less RXQ buffers?

Background: RXQ Buffer Pre-allocation

Why do we need RXQ buffers?

- A NIC has multiple RXQs (circular rings)
- Each RXQ has its own page pool, for re-allocating new pages after processing
- Pre-allocates buffers for
  - Handling burstiness, or
  - Per-packet processing jitters
  - NAPI schedule delay

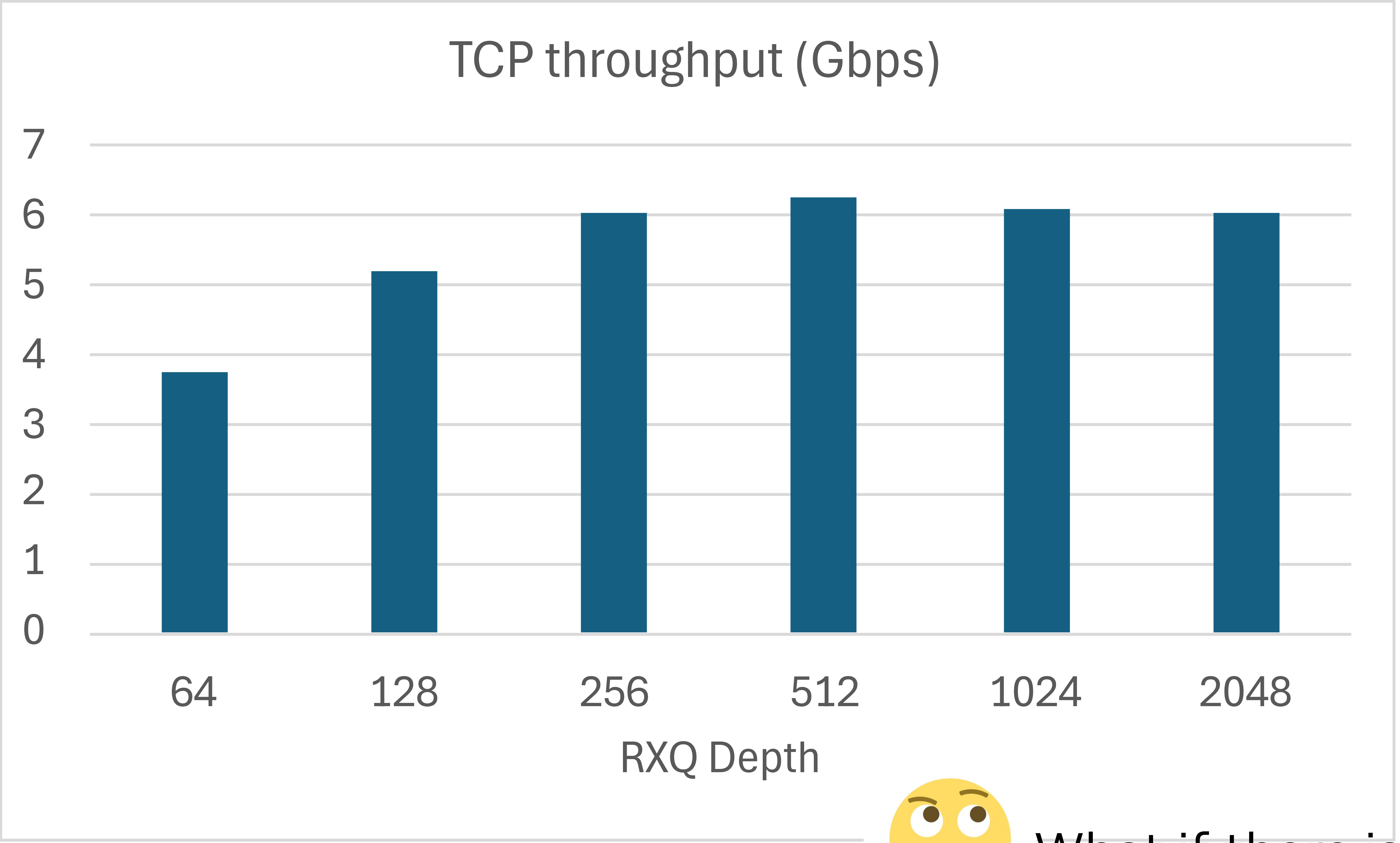


diff color

Quick Evaluation (1/2)

Fixed High Watermark

- Two servers connected back-to-back, single iperf TCP throughput
- Hardware offload disabled, all traffic go to slow-path OVS
- Statically change representor's RXQ depth (ethtool –G rx) from 64 to 2048



🤔 What if there is no / little traffic?

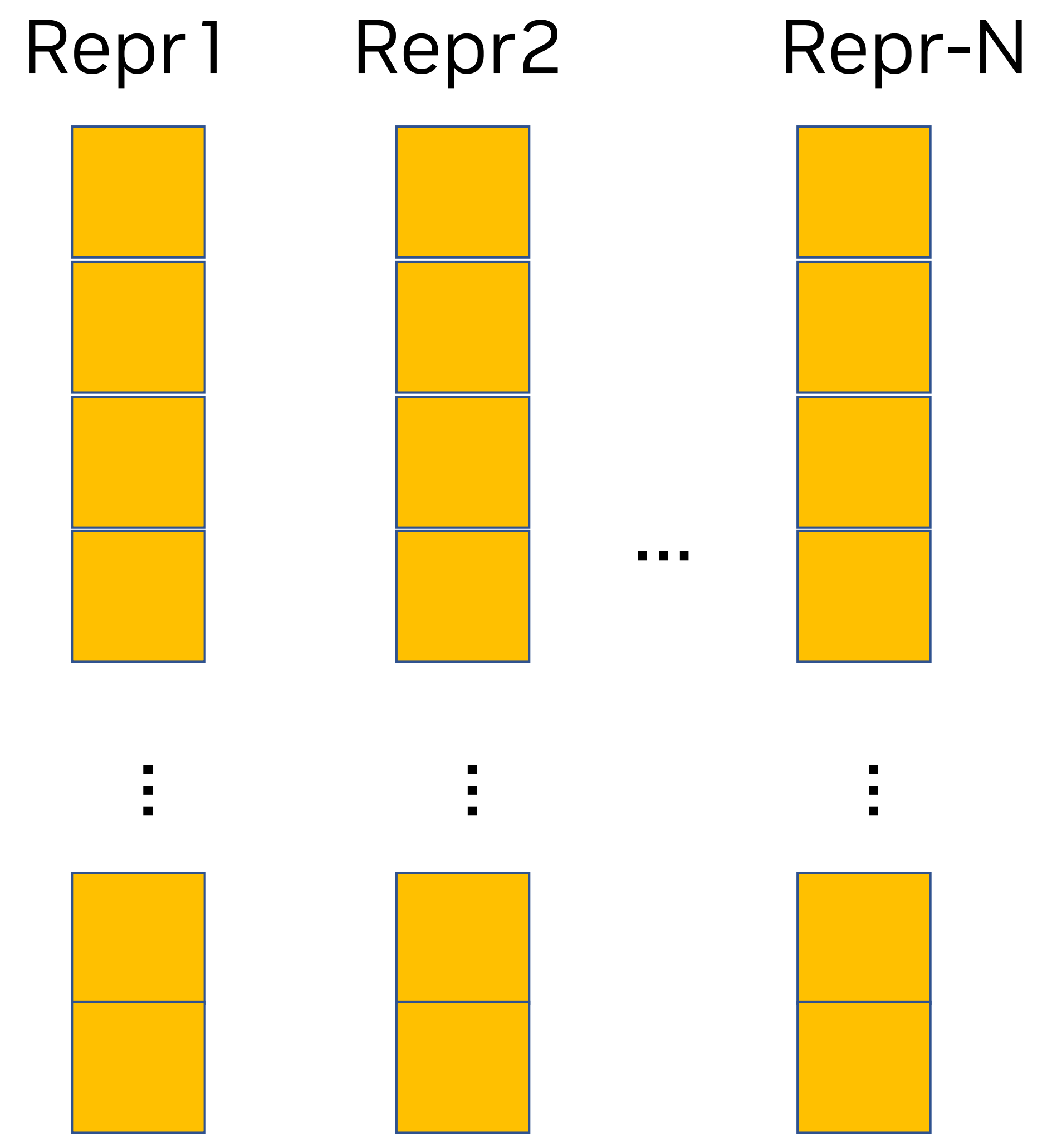
Design-2: Adjustable RXQ

save memory by dynamic allocation

Currently

- Driver always refill rxq to full, ex: default 1024 buffers
- Performance drop if rxq depth is too shallow, ex: 64
- But what if there is little traffic? Then we waste lots of memory

Current: always refill to full



Design2: Dedicated Repr netdev with Adjustable RXQ

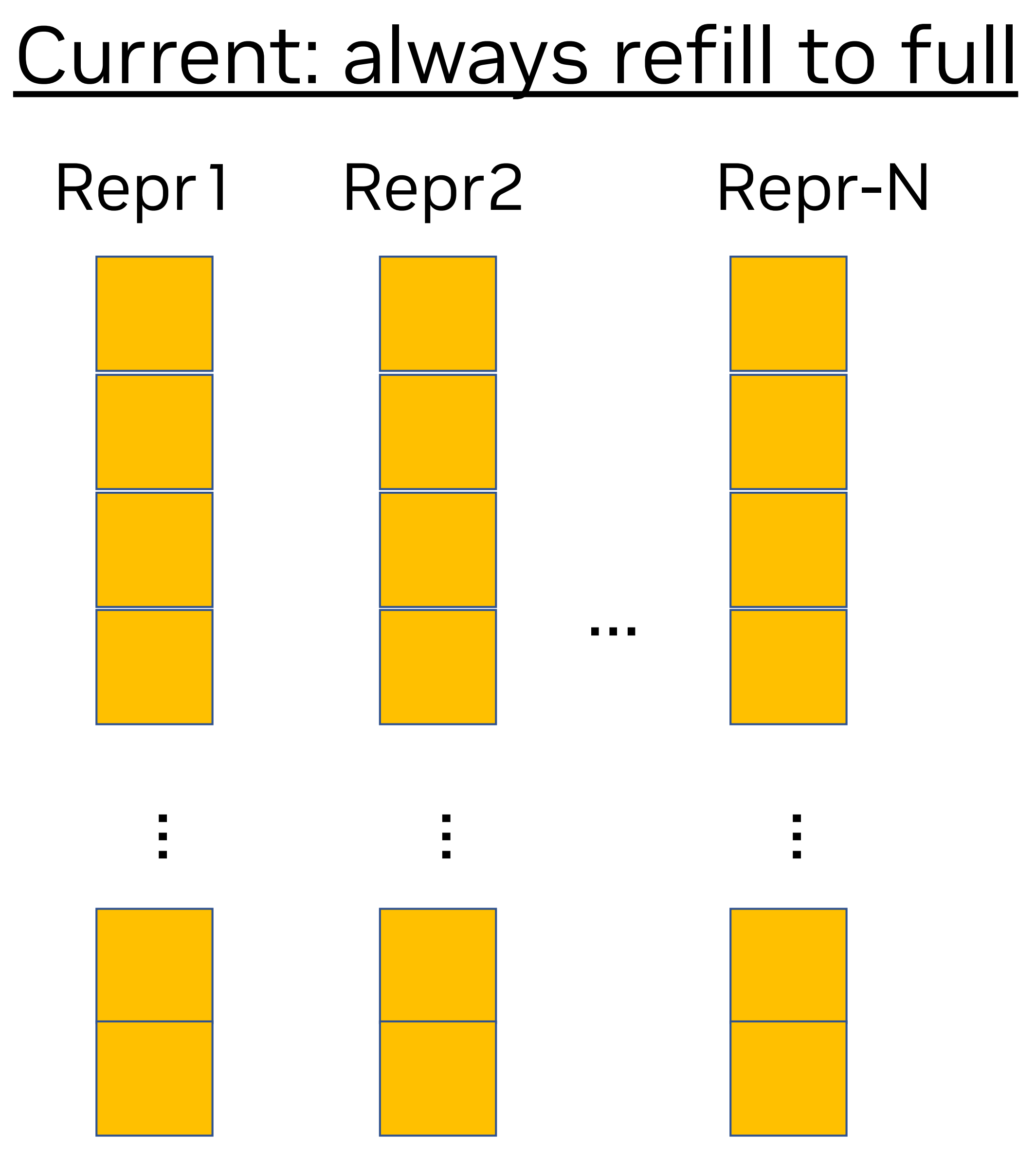
save memory by dynamic allocation

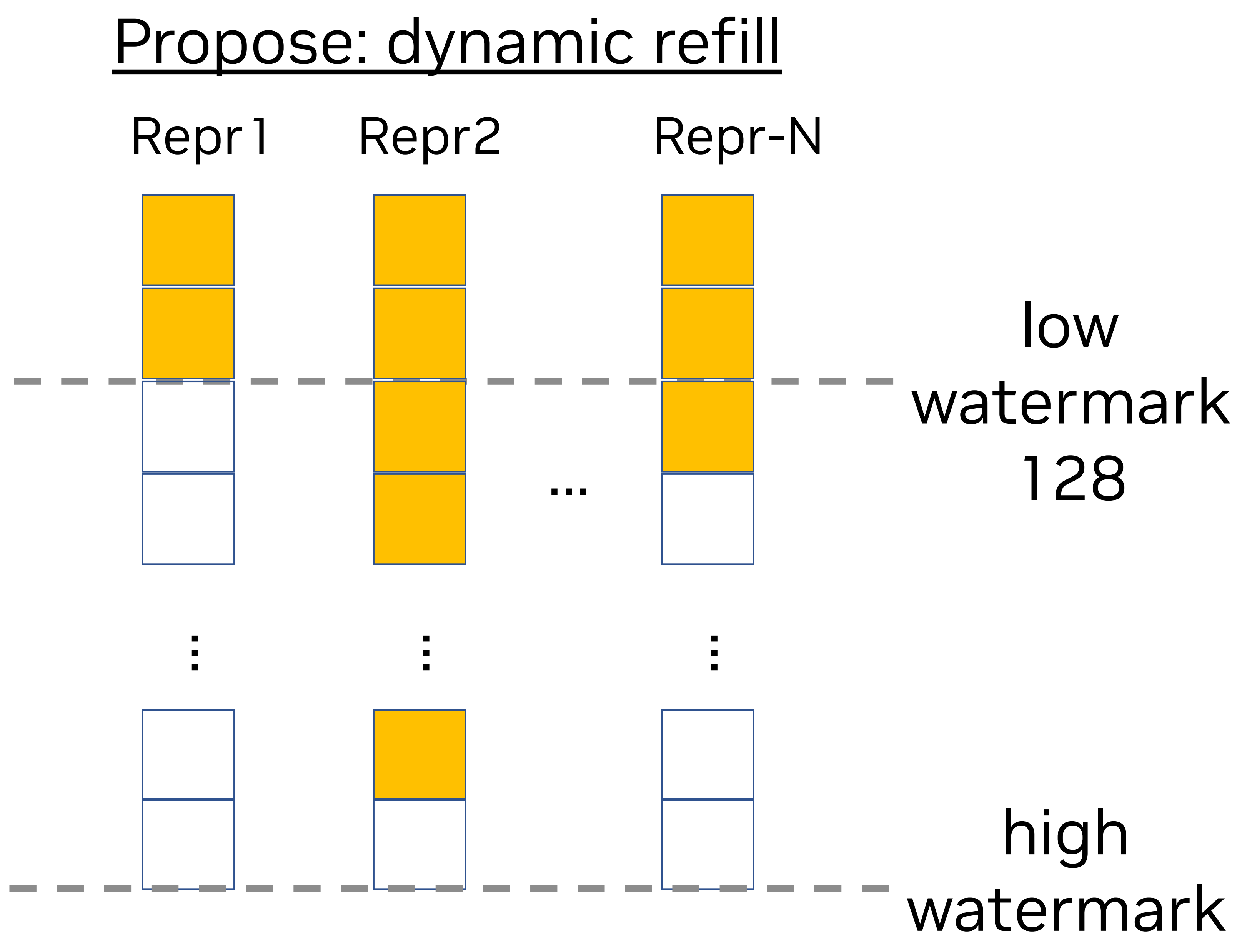
Currently

- Driver always refill rxq to full, ex: default 1024 buffers
- Performance drop if rxq depth is too shallow, ex: 64
- But what if there is little traffic? Then we waste lots of memory

Idea: Don't always allocate to full rxq size → save memory!

- Performance impact: First burst of traffic greater 128 definitely lost
- Low watermark set to fixed 128 buffers (2*NAPI_BUDGET)
- High watermark, max RXQ buffers, set by ethtool –G rx



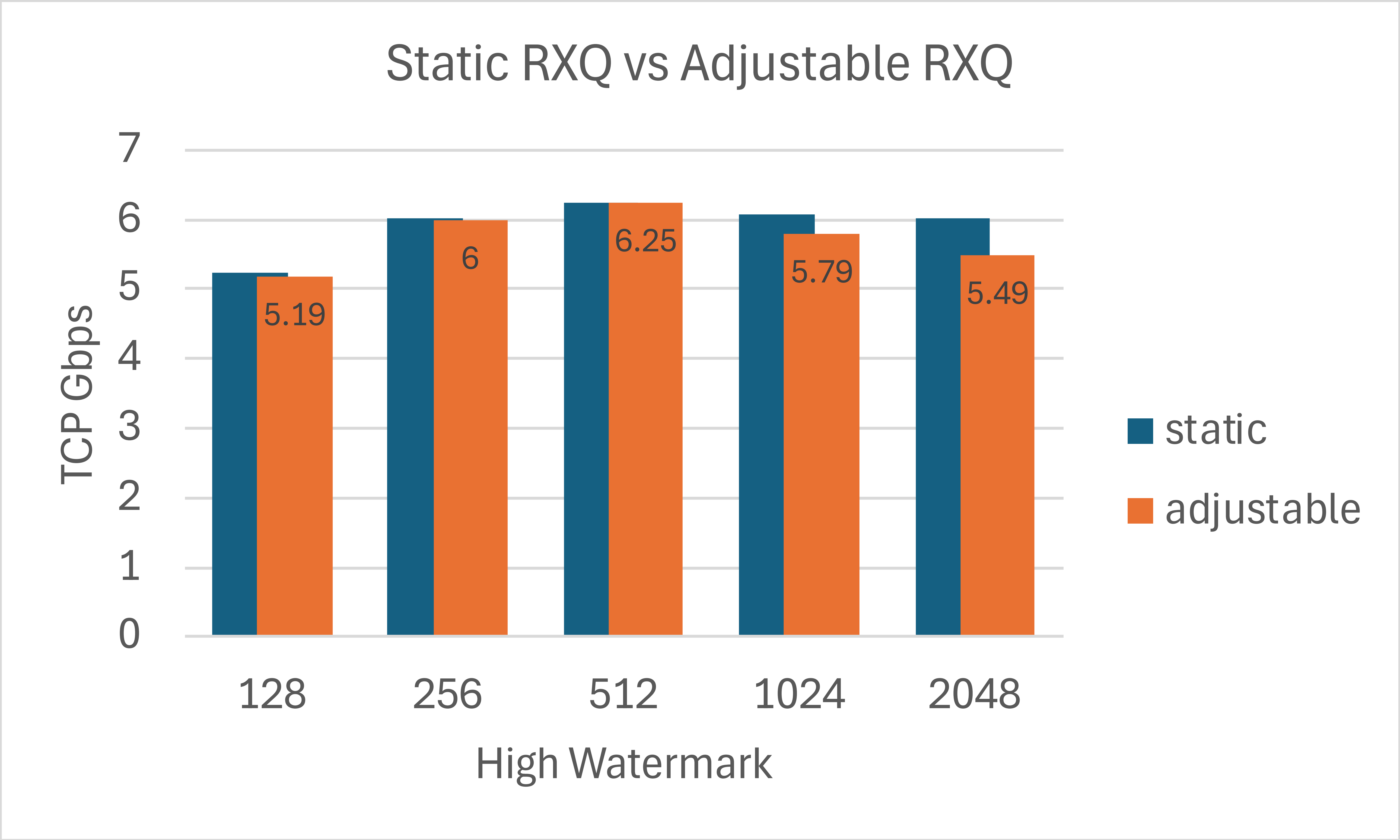


Quick Evaluation (2/2)

Dynamically Adjust the RXQ Depth

Simple Algorithm:

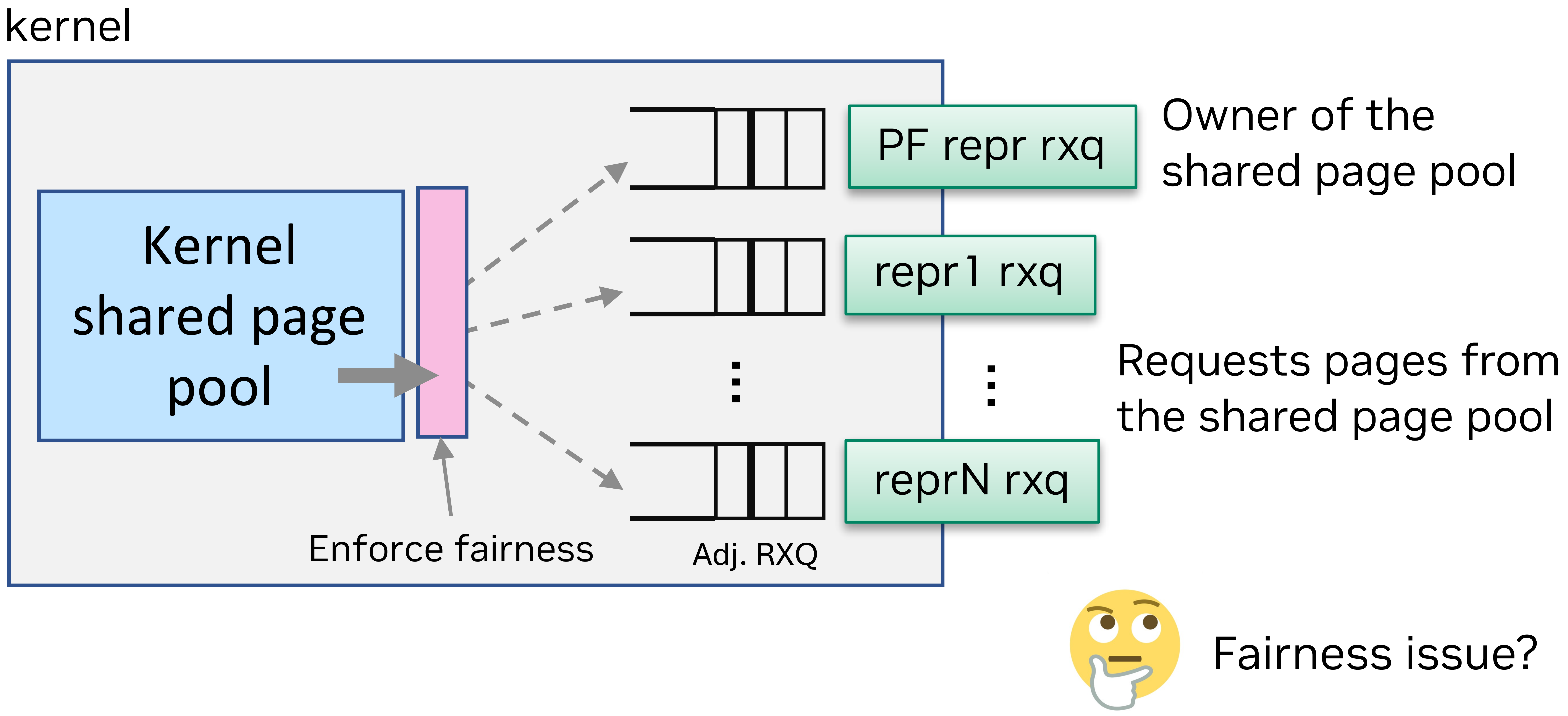
- When in NAPI-interrupt: save memory by not refill, or refill up to low_watermark
- When in NAPI-busy: fallback to default behavior, driver refill to FULL
- The first burst over 128 definitely drops, but we'll catch up



Design-2+: Adjustable RXQ with Shared Page Pool

Problem: The later-created representors might get no memory

- Current: each RXQ (NAPI) has its own page pool
- Propose: all RXQs use the same page pool
- Challenge: Need to track each RXQ usage and need lock
- Use for representors (shared single DMA device)



Fairness with Shared Page Pool

Borrow the solution from hardware switch and devlink-sb interface

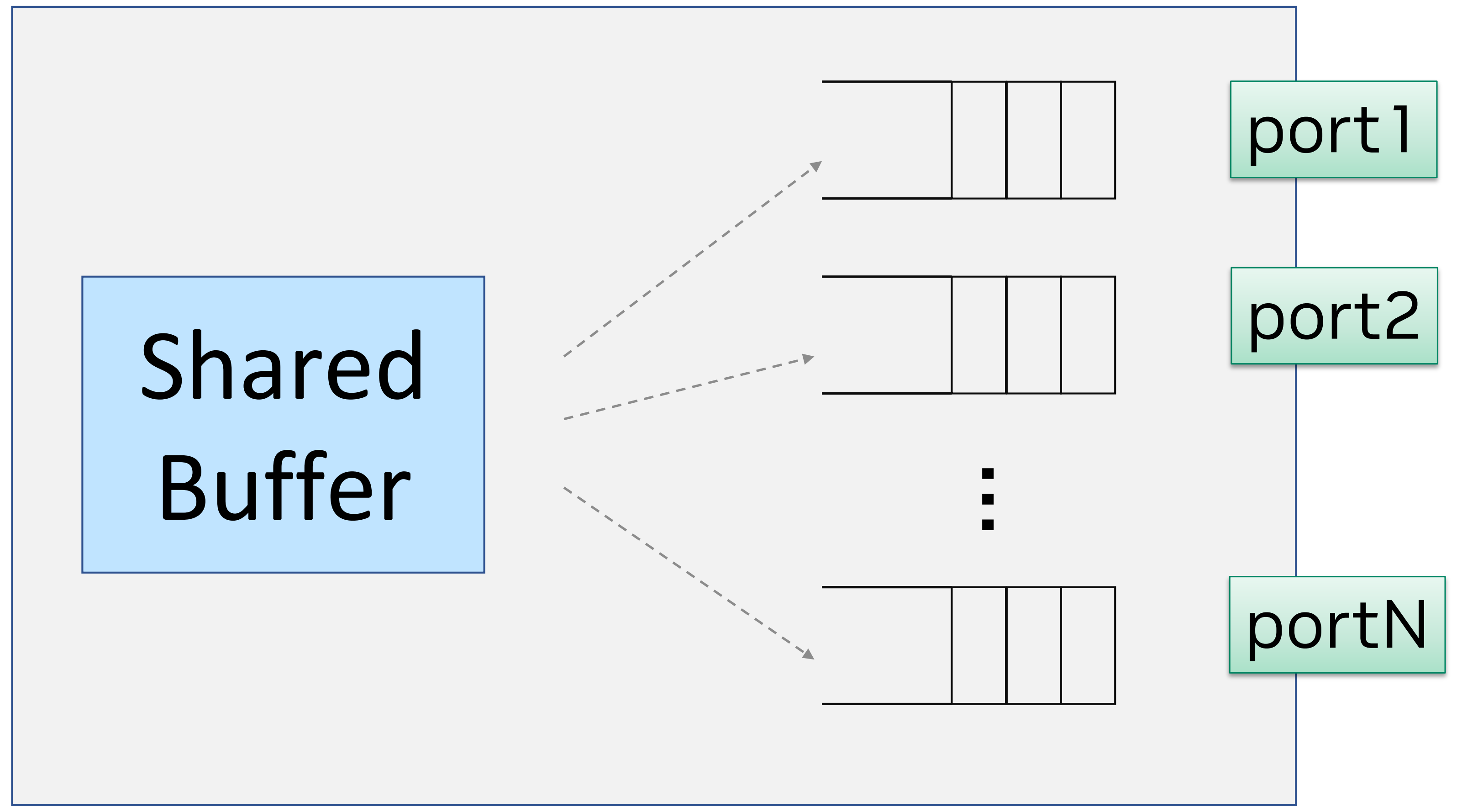
Shared Buffer in Hardware Switch

- Each output port has a logical queue
- The logical queue decides the budge/usage of the output port
- **Dynamic Threshold**: adjust queue depth based on current usage

For switchdev

- Logical queue -> RXQ
- Port -> representor netdev
- Shared Buffer -> shared page pool

Shared Memory Switch with multiple output ports



Limit a port’s shared memory usage to :

$$\text{max\_usage} = \frac{\alpha}{1+\alpha} \times \text{Free\_Buffer}$$

Dynamic Queue Length Thresholds for Shared-Memory Packet Switches
Sizing Router Buffers

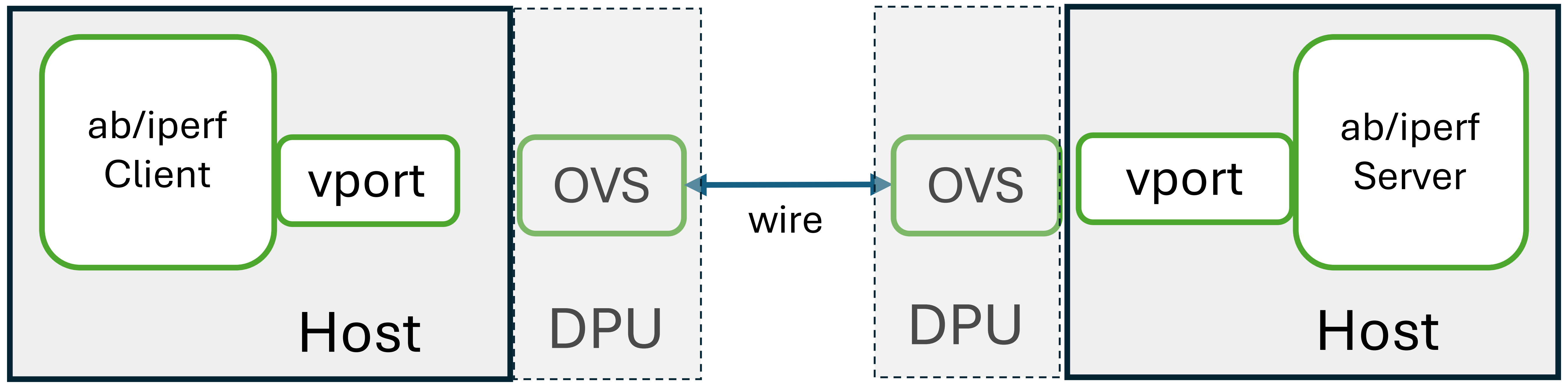
New Devlink Attribute: spool-mode

None, Basic (Shared RXQs), SPP (Shared Page Pool)

- Limited by Memory/Queue -> use shared RXQs
- Performance isolation is important -> use adjustable RXQs or dedicated

mode	Feature	Drivers
None	Design-0: Dedicated repr netdev	Octeontx2, mlx5
Basic	Design-1: Shared RXQs	ICE, BNXT, NFP, SFC
SPP	Design-2: Adjustable RXQ with shared page pool	

```
# dedicated repr netdev, ex: Octeontx2, mlx5
$ devlink dev eswitch set pci/0000:08:00.0 mode switchdev spool-mode none
# Shared RXQ with PF, ex: ICE, nfp
$ devlink dev eswitch set pci/0000:08:00.0 mode switchdev spool-mode basic
# WIP
$ devlink dev eswitch set pci/0000:08:00.0 mode switchdev spool-mode spp
```

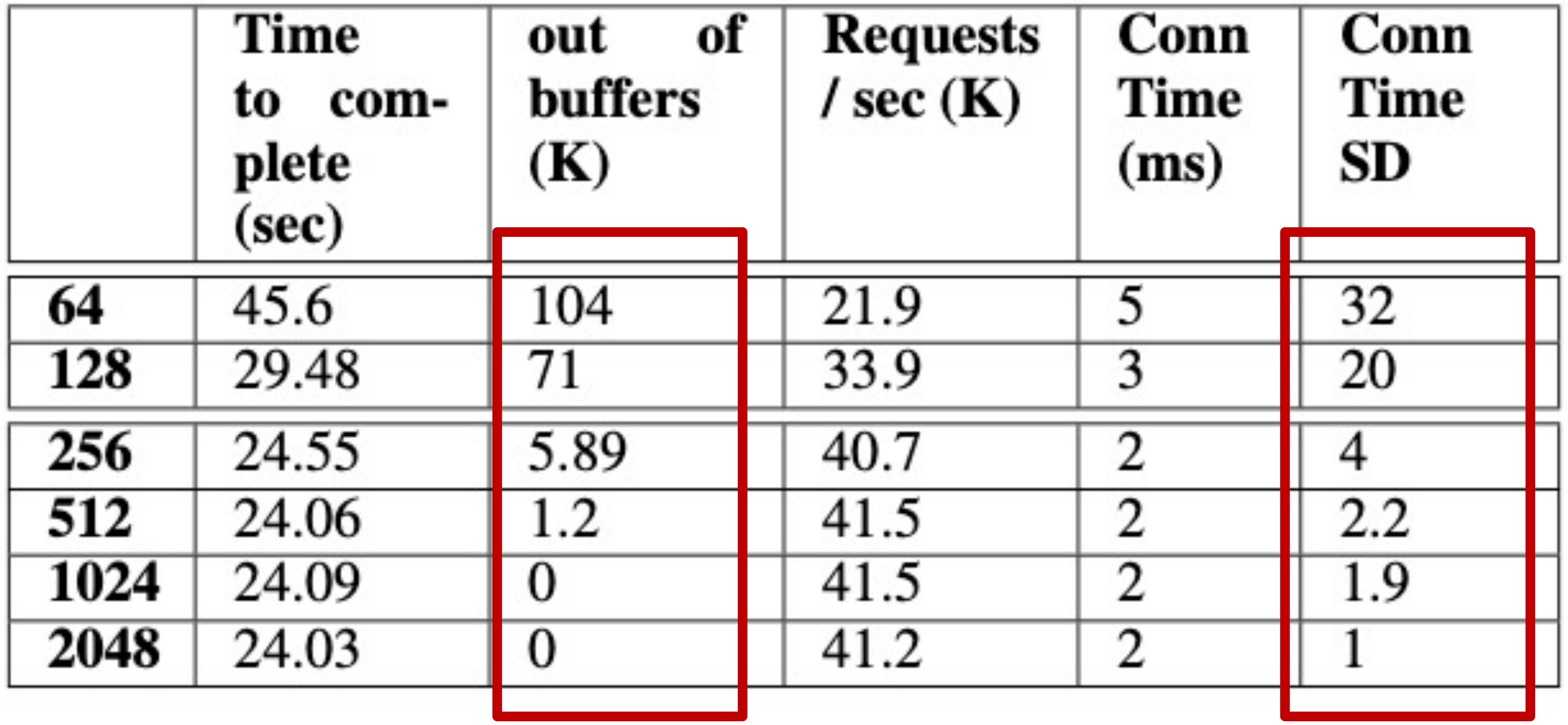


Performance Evaluation

Evaluation-1: Static RXQ, 64 - 2048

Apache ab benchmark with 1 million requests 100 concurrency, with different RXQ depth
Disable hw-offload

- Time to complete (sec): total time taken for completing the 1 million requests.
- out of buffers (K): a firmware counter, rx out of buffer, re- porting number of packets dropped due to no RXQ buffer available.
- Requests (K) / sec: average HTTP requests per seconds
- Connection Time (ms) and SD: average connection time, including connect, processing, and waiting, of the 1 million connections and their **standard deviation** (SD).



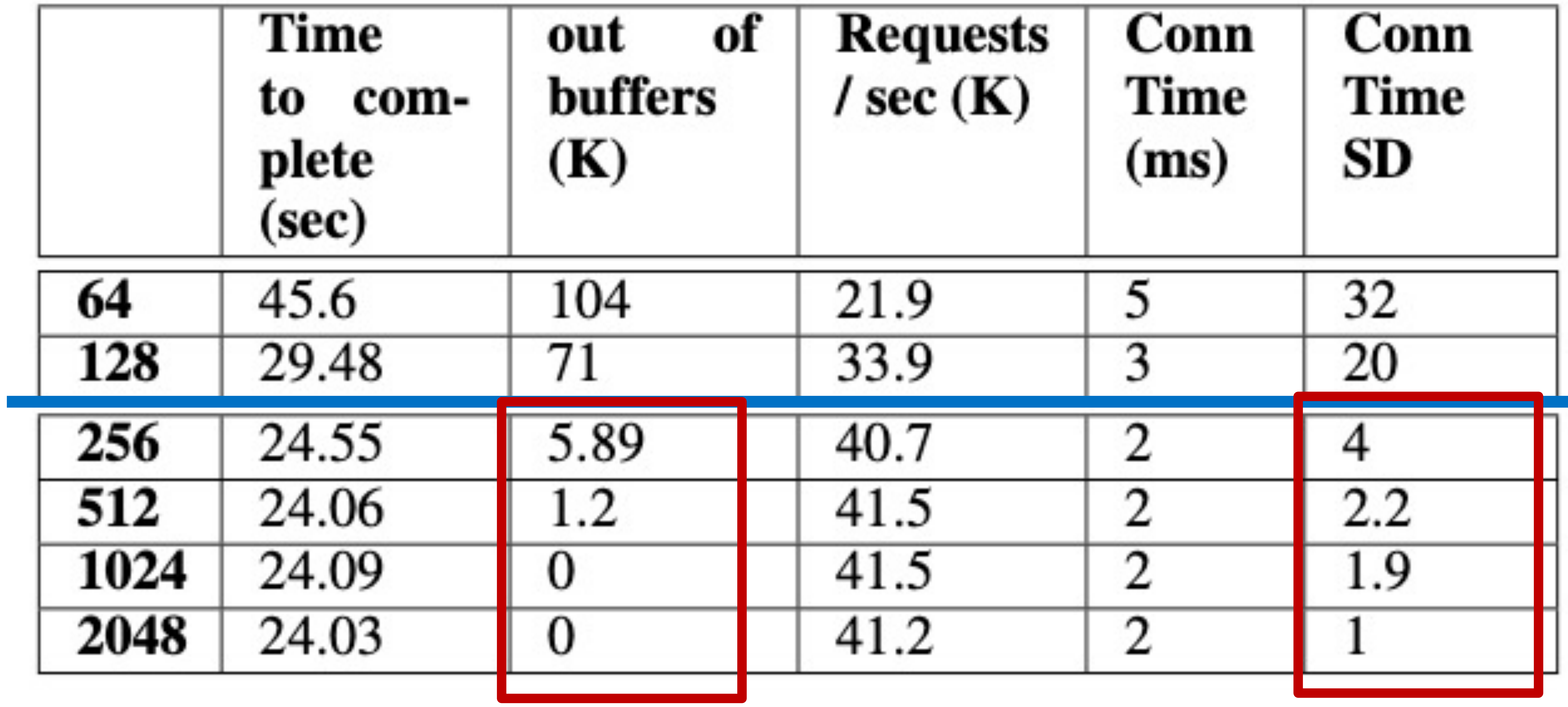
	Time to complete (sec)	out of buffers (K)	Requests / sec (K)	Conn Time (ms)	Conn Time SD
64	45.6	104	21.9	5	32
128	29.48	71	33.9	3	20
256	24.55	5.89	40.7	2	4
512	24.06	1.2	41.5	2	2.2
1024	24.09	0	41.5	2	1.9
2048	24.03	0	41.2	2	1

Evaluation-2: Static RXQ vs Adjustable RXQ

Apache ab benchmark with 1 million requests 100 concurrency, with different RXQ depth
Disable hw-offload

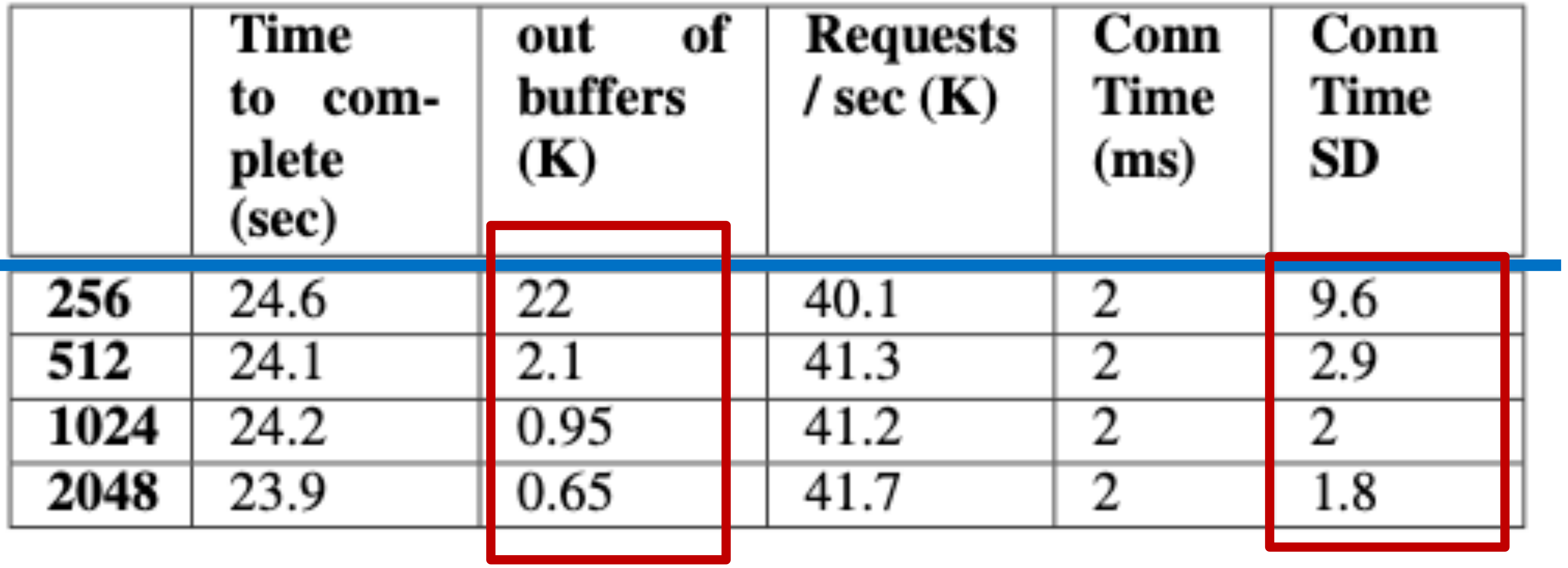
- Out of buffers showing more packets are dropped
- Higher jittering
- Average time to complete is similar

Static RXQ



	Time to complete (sec)	out of buffers (K)	Requests / sec (K)	Conn Time (ms)	Conn Time SD
64	45.6	104	21.9	5	32
128	29.48	71	33.9	3	20
256	24.55	5.89	40.7	2	4
512	24.06	1.2	41.5	2	2.2
1024	24.09	0	41.5	2	1.9
2048	24.03	0	41.2	2	1

Adjustable RXQ



	Time to complete (sec)	out of buffers (K)	Requests / sec (K)	Conn Time (ms)	Conn Time SD
256	24.6	22	40.1	2	9.6
512	24.1	2.1	41.3	2	2.9
1024	24.2	0.95	41.2	2	2
2048	23.9	0.65	41.7	2	1.8

Definitely need more benchmark strategies

Summary

Need your feedback!

- Switchdev slow-path and fast-path
- Dedicated Representor Netdev and Shared RXQs
- Adjustable RXQ (vendor drivers) with shared page pool
- Add new devlink eswitch attribute: spool-mode
- More performance number and design in paper

Discussion

- Can shared page pool used in fast-path virtual device?
- How to model the performance of adaptive RXQ?

Thank you!

Backup Slides

OFED: Uplink Rep’s RQ for all other RQs

Uplink RQ’s to service all RX packets destined for non-uplink representors (SF/VF/PF)

Control plane

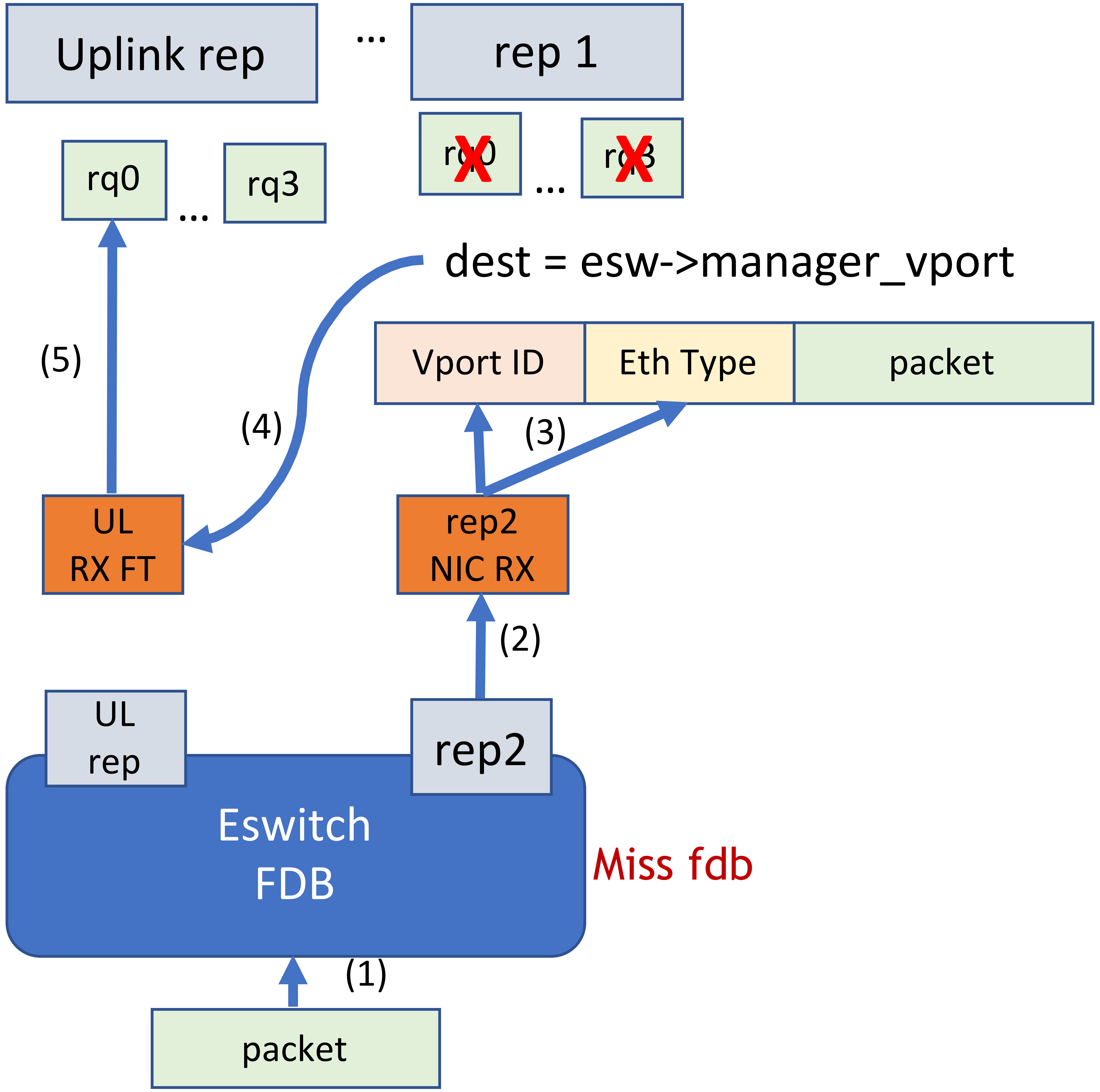
- Devlink enable at switchdev mode/ or disable
- Maintain xarray for vport id to representor netdev struct

Data plane: Steering

- Insert pet header of 8 bytes (2 bytes contains new ethertype)
- Copy 2 bytes of source vport that is stored in reg_c0
- Set uplink as destination vport

Data Plane: driver

- Get vport id from rx buffer, lookup netdev struct using vport id
- Strip the 8 bytes from SKB and patch the SKB with correct netdev



Scaling uplink REP’s rx Queues

Targeting 1K SFs

No Shared RQ

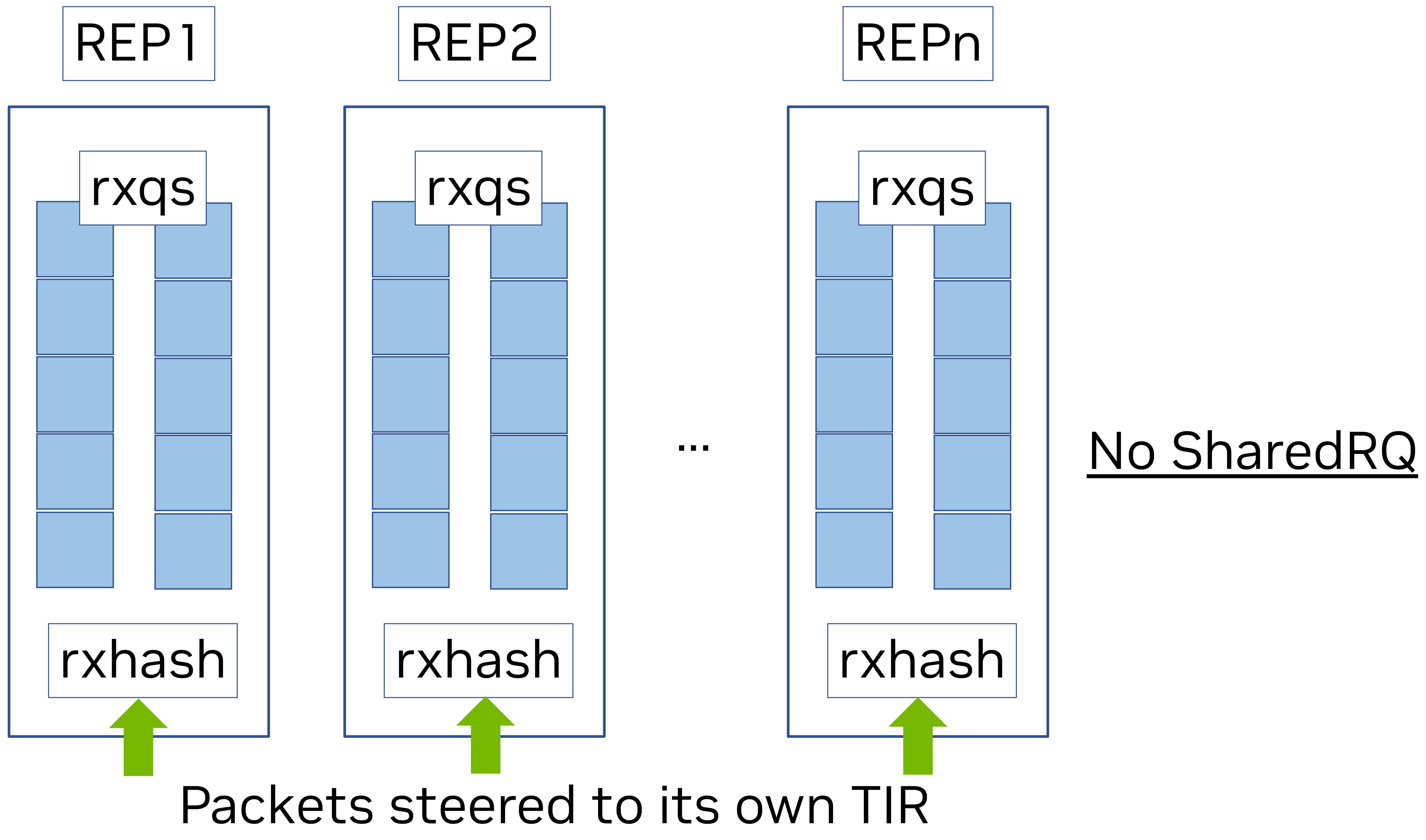
- each repr has its own rxq, ex: 2 channel/2 rxqs
- 1k representors has total 2k rxqs
- Each rep’s flow through its own rxhash

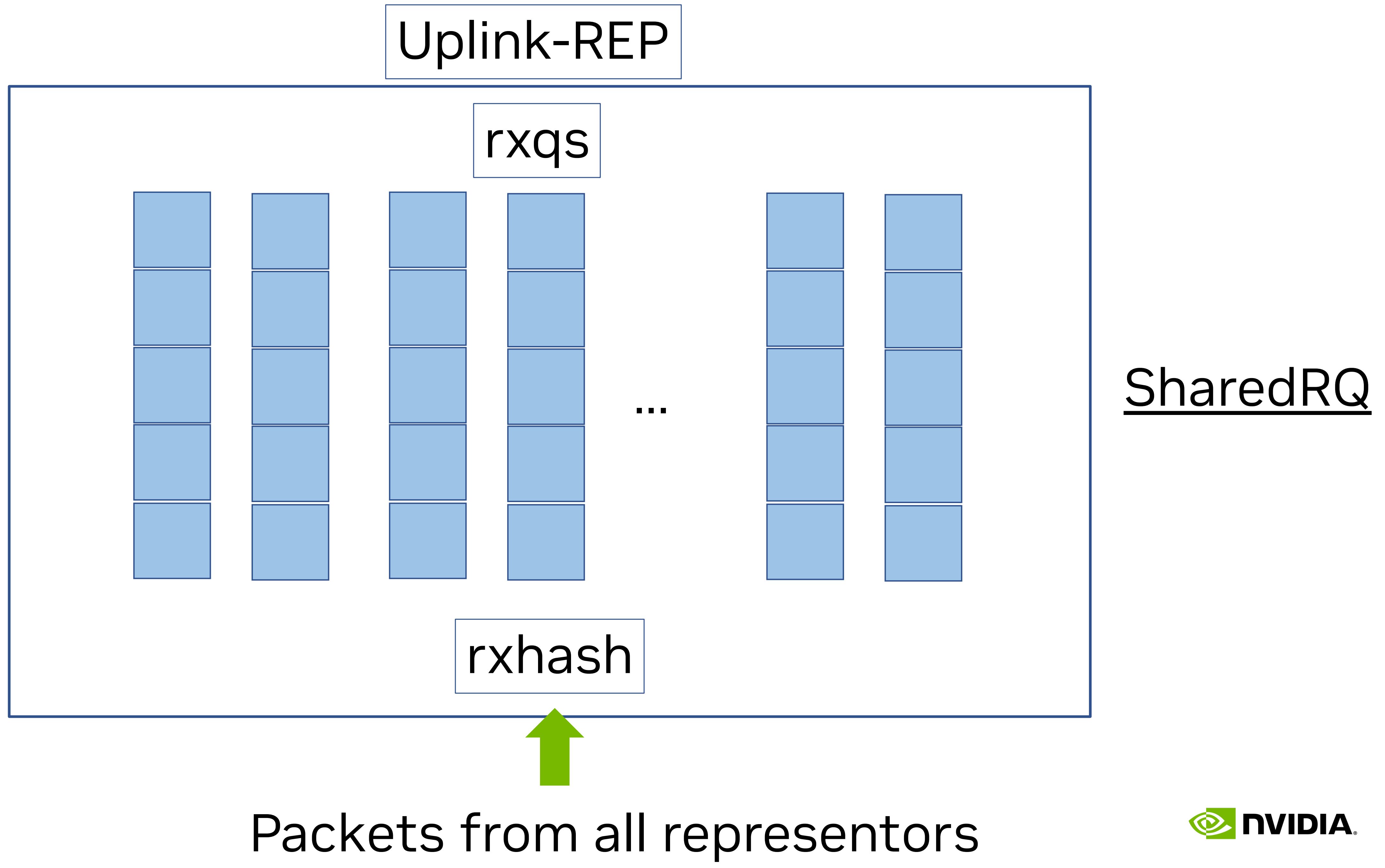
With Shared RQ on BlueField-3

- PF creates 16 rxqs (max limited by CPUs)
- Traffic from all representors uses the same rxhash and decides which rxq

Idea: increase 16 to more, ex: 128

- Lower the chance for hash collision, depth depends on NAPI scheduler
- NAPI schedule natively provides fairness for each queue
- 16 queues with 1024 entries is different than 128 queues with 64 entries!
- Can we hash based on vport_id? If yes, it’s the same as no-sharedrq
- Existing ethtool controls everything, no extra knob needed?





RFC: https://git-nbu.nvidia.com/r/c/upstream/linux/+/1099459