# Multi-PF Net Device

Tariq Toukan

Netdev 0x18 Conference
Santa Clara, California
July 2024

# Background

- Idea was originally presented by Achiad Shochat in Netdev conference 2.2.
- Matured, prioritized, implemented, and accepted upstream (v6.9).

# Agenda

- Describe problems
- Adapter-level solution
- Software model
- Design decisions
- Implementation details
- Performance numbers
- Future work

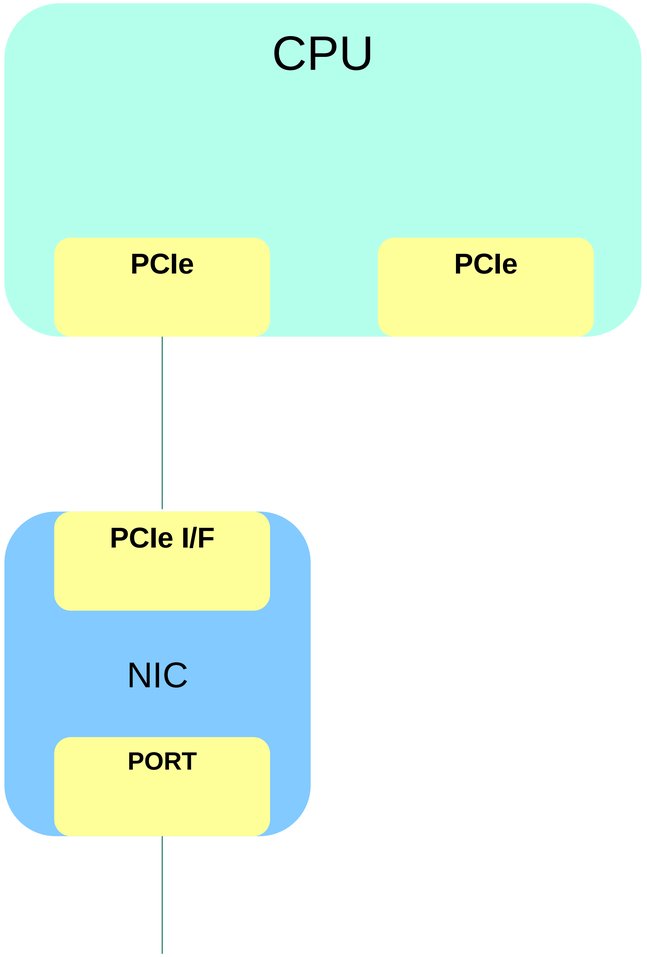
# Problem description

## Problem #1: BW mismatch PCI vs NIC

# Problem description

## Problem #1: BW mismatch

- NIC connected to host
- Many possible combinations for
  - NIC speed x PCI speed



# Problem description

## Problem #1: BW mismatch PCI vs NIC

**PCI speeds**

| | RAW bitrate (GT/s) | Link BW (Gbps) | **x16 BW (Gbps)** |
|---|---|---|---|
| **PCIe 1.0** | 2.5 | 2 | **32** |
| **PCIe 2.0** | 5 | 4 | **64** |
| **PCIe 3.0** | 8 | 8 | **128** |
| **PCIe 4.0** | 16 | 16 | **256** |
| **PCIe 5.0** | 32 | 32 | **512** |
| **PCIe 6.0** | 64 | 64 | **1024** |

**vs.**

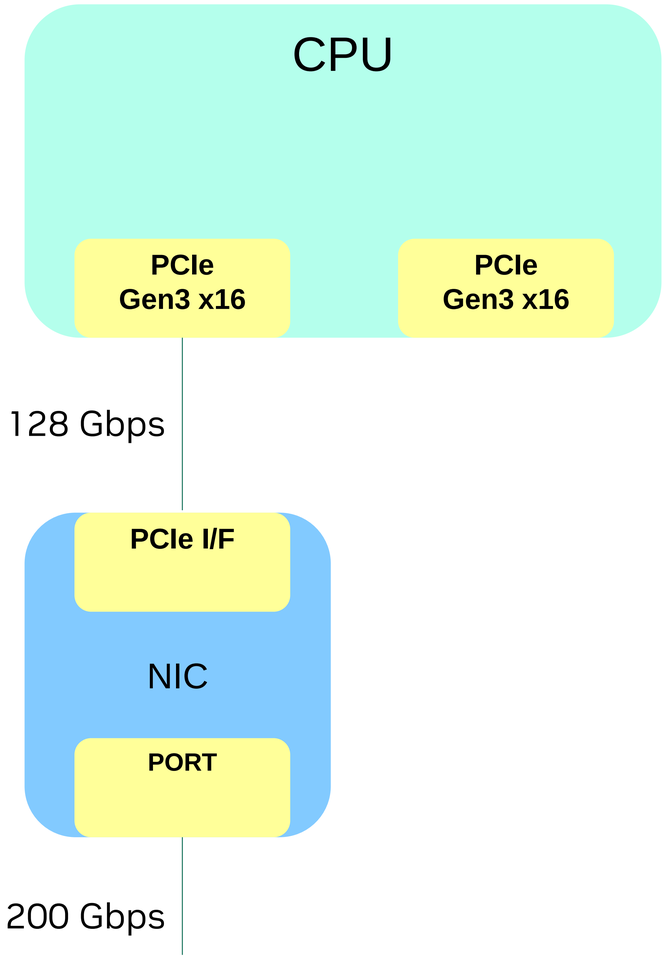
**NIC speeds**

- **200 Gbps** (ConnectX-6)
- **400 Gbps** (ConnectX-7)
- **800 Gbps** (ConnectX-8)
- …

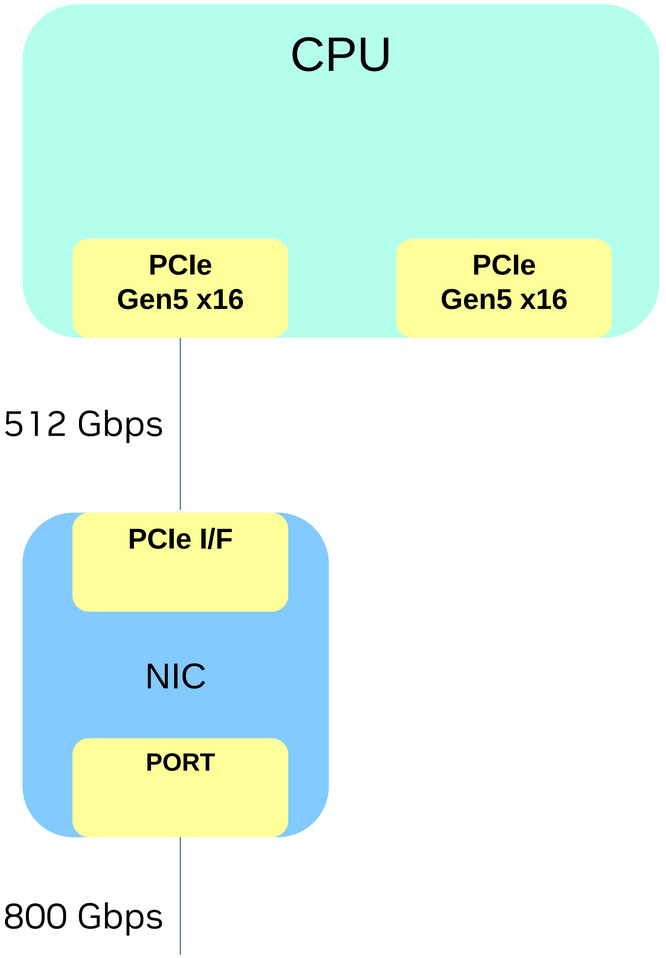
# Problem description

## Problem #1: BW mismatch

- Example 1:




- Example 2:

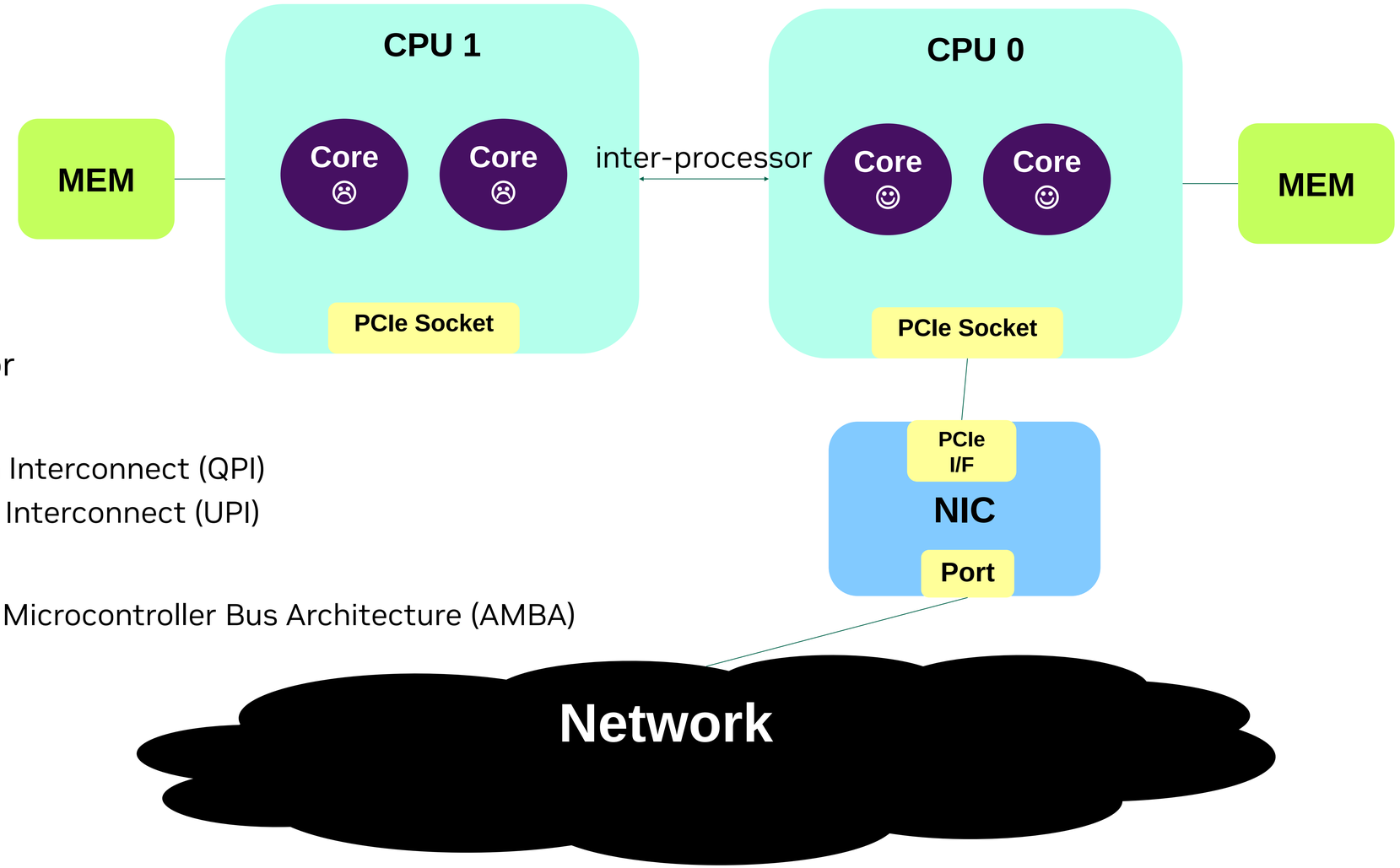


# Problem description

## Problem #2:
## CPU scaling on NUMA systems

# Problem description

## Problem #2: CPU scaling on NUMA systems




Inter-processor

- Intel:
  - QuickPath Interconnect (QPI)
  - Ultra Path Interconnect (UPI)
- ARM:
  - Advanced Microcontroller Bus Architecture (AMBA)
- ...

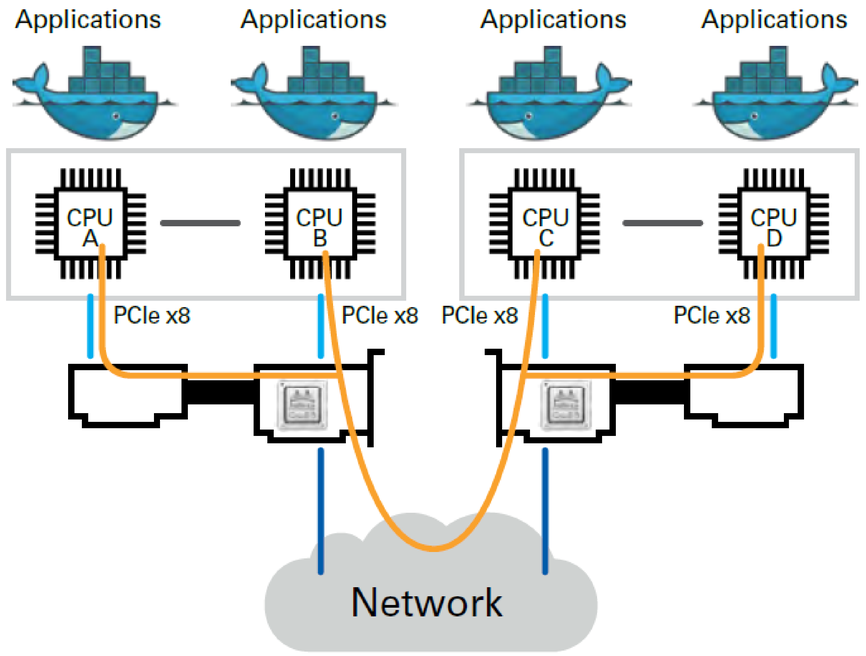
# Solution

NIC port with multiple PCI buses

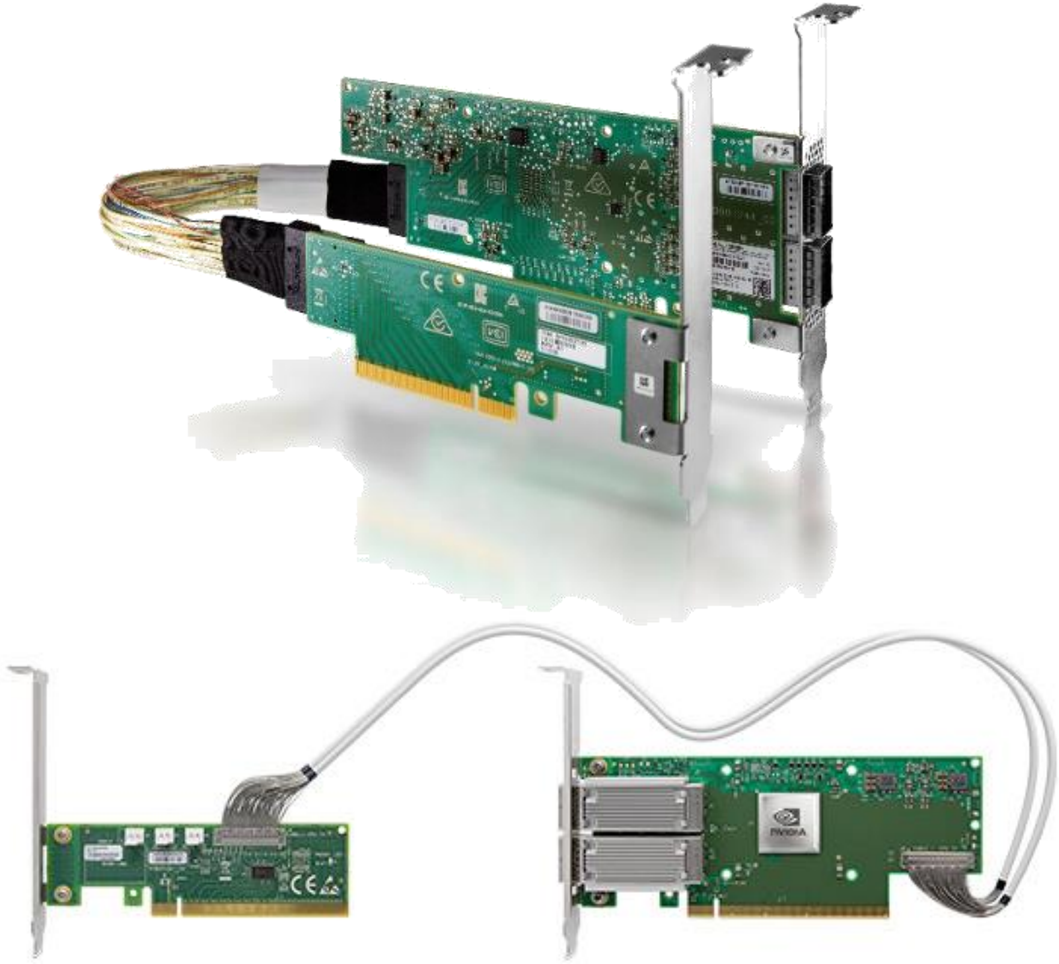
# Solution

## Adapter with multiple PCIe per port

- Adapter-level solution
- Adapter architecture that connects the NIC port to the host through multiple PCIe interfaces

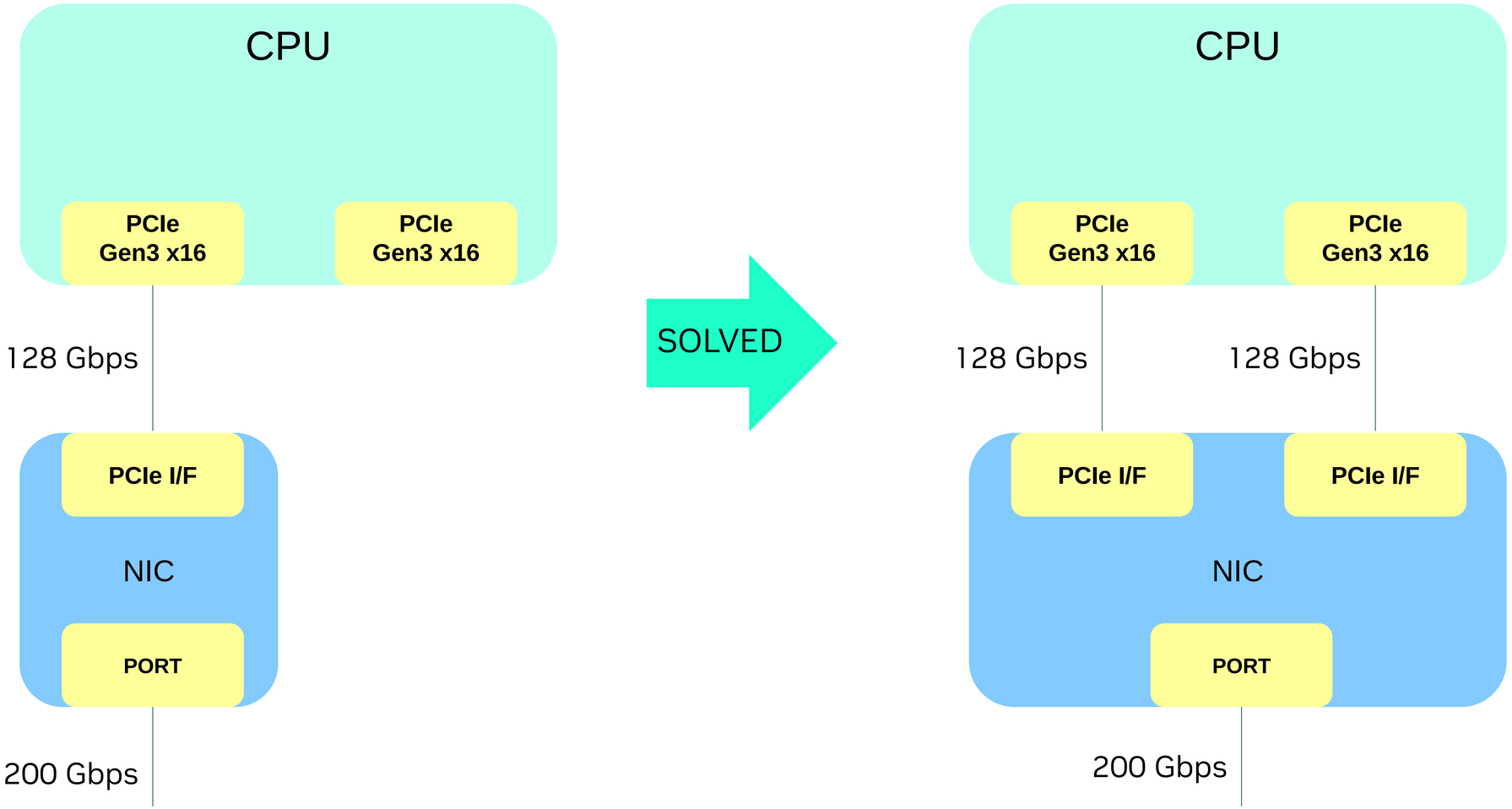



Socket Direct Card Configuration



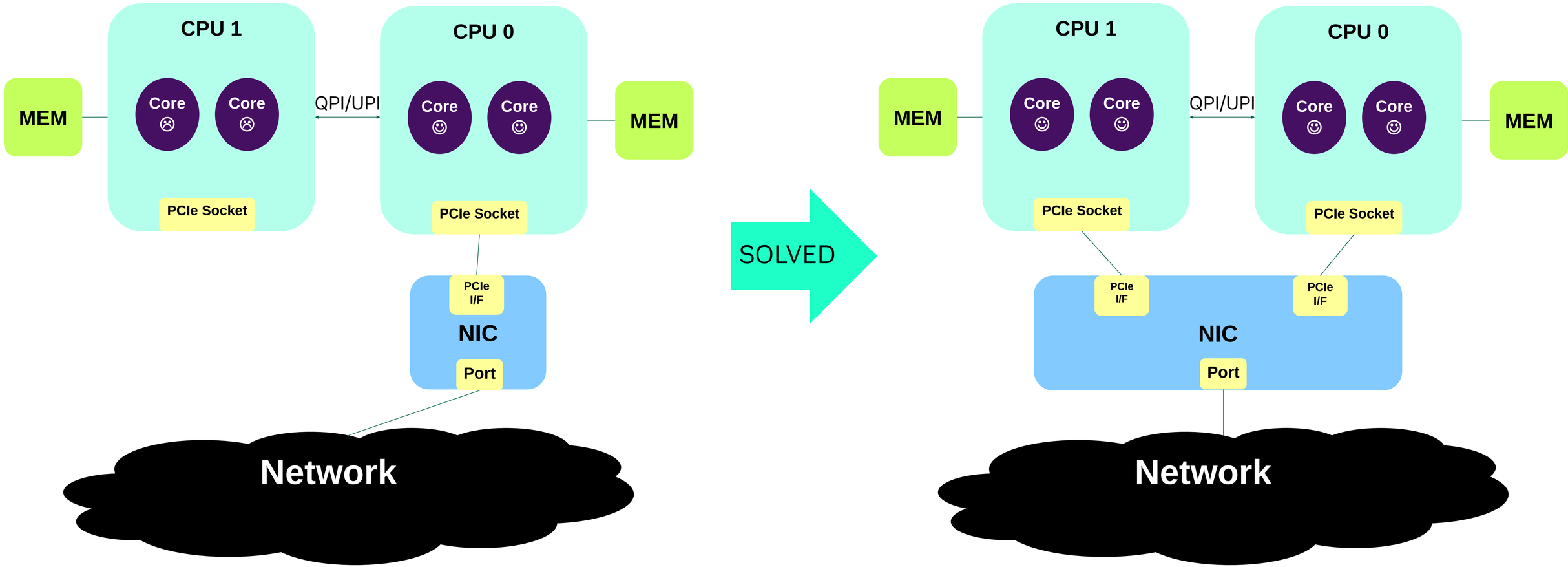
# Solution

Adapter with multiple PCIe per port
Problem #1: BW mismatch, SOLVED



# Solution

Adapter with multiple PCIe per port
Problem #2: CPU scaling on NUMA systems, SOLVED

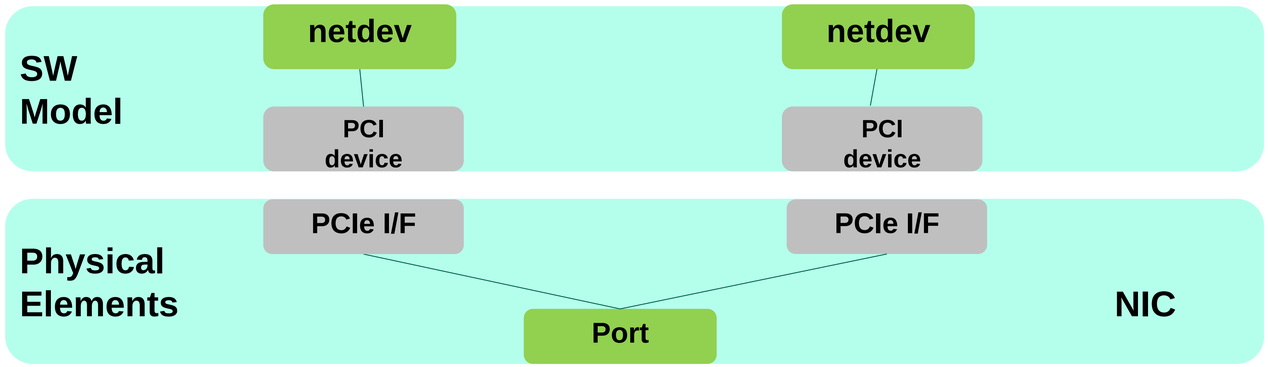


# Impact on Software

- OS is not aware of the network port sharing

- Multiple PCIe buses
  - Each creates its own netdev

- Multiple net devices
  - Multiple MAC addresses
  - Multiple IP addresses
  - ...




- Confused applications
  - Specify netdev or IP address to benefit from NUMA locality
- Totally different management

# Software Model

## Multi-PF net device

# Software Model

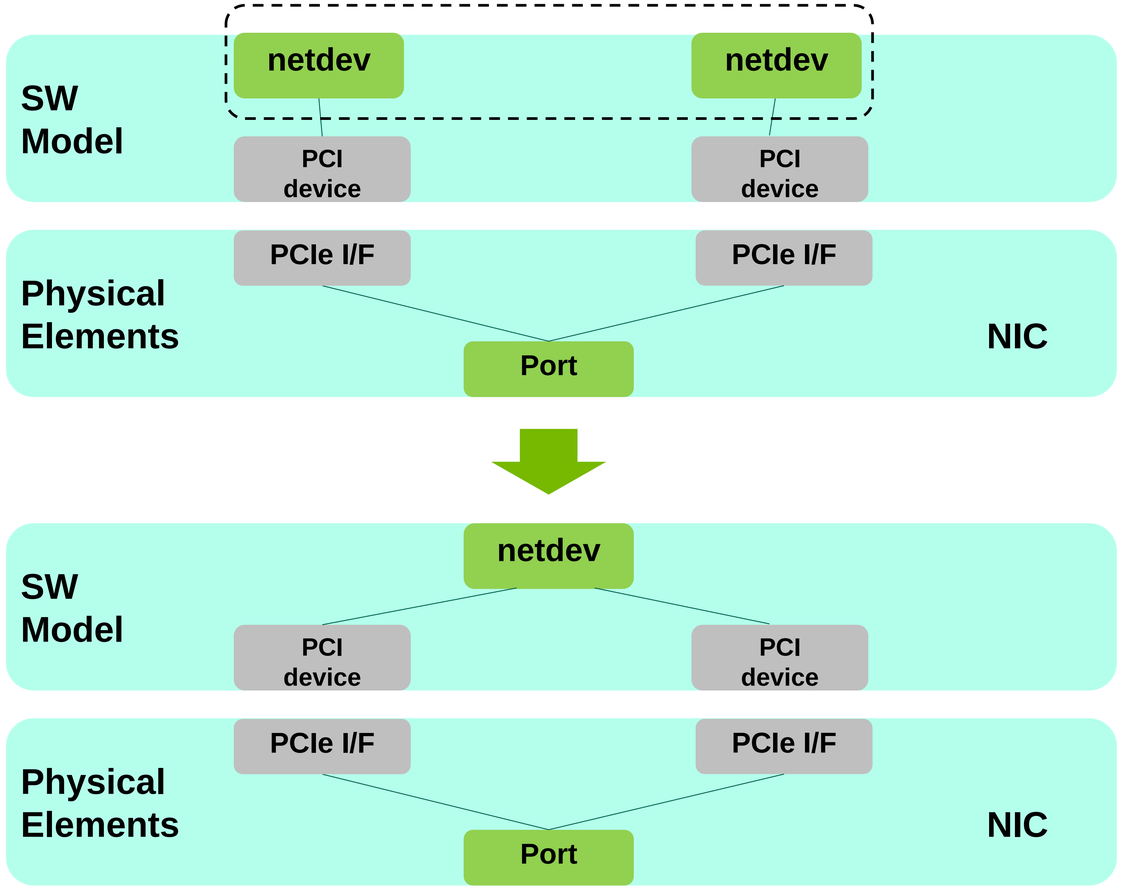
## Multi-PF net device

Idea:

- Combine the multiple net devices into one
- Abstract the aggregation logic in the vendor driver level (mlx5e)
- Expose the multi-PCI NIC port to the network stack through a single netdev

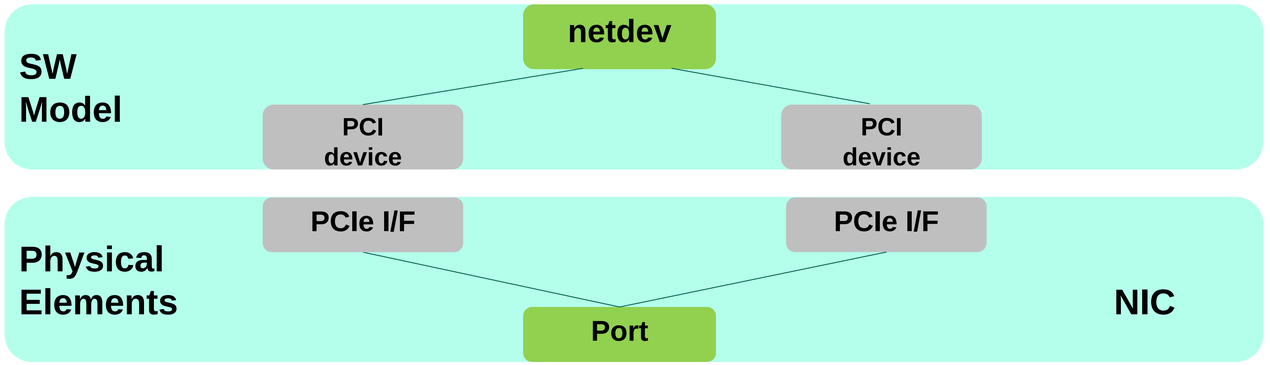
Before:

After:



# Software Model

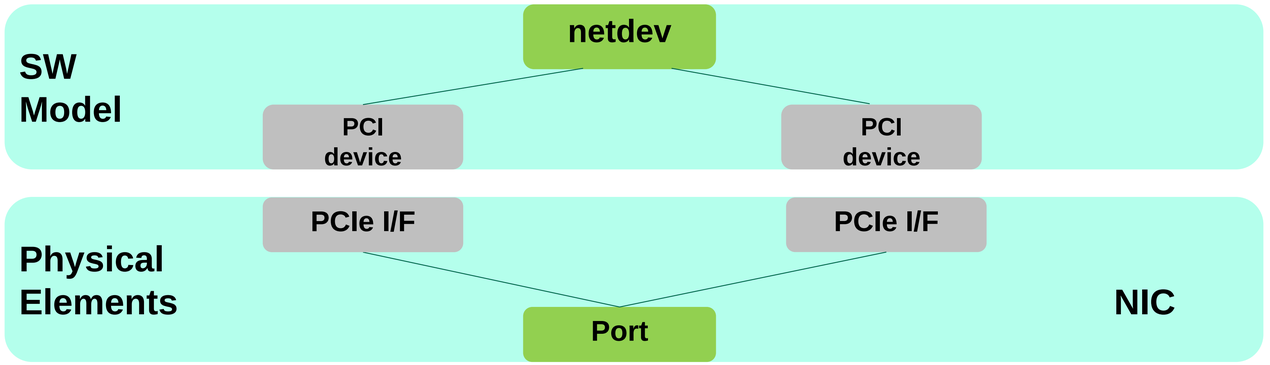
## Multi-PF net device




- Aligned with the netdev per port kernel convention
- Good layers partitioning
  - PCI subsystem is unaware (untouched) - PCI device per PCI bus
  - Net subsystem is unaware (untouched) – network device per network port
  - The whole aggregation logic is encapsulated in the network device driver (mlx5e)
- Good reflection of reality: Symmetric modeling of the physical elements
- Good Out-Of-Box experience
  - Driver takes care
  - No necessary admin configurations
- Netdev software stats are consistent with the hardware port stats

# Software Model

## Multi-PF net device




- Designed and implemented to support more than two PFs per port
- However, we do not allow untested setting
- Tested and verified on 2 PFs
  - `#define MLX5_SD_MAX_GROUP_SZ 2`
- When future hardware with more than 2 PFs per port becomes available
  - Verify functionality
  - Verify performance
  - Increase the software constant

# Design Decisions

One net device managing multiple PFs

- Device resources management
- Categorize into affined / non-affined resources

- Non-affined resources (RSS indirection table, flow steering) go through one designated bus
  - Called “primary PF”
  - All others are called “secondary PFs”

- Affined resources (TX/RX queues) go through their designated bus
- Each PF is already associated with a NUMA node
- Distribute datapath channels (TX/RX queues) equally between the PFs
- Properly set IRQ affinity and XPS

- Setting is applied by the driver in default

# Design Decisions

## Channels Distribution Policy

Channels distribution policy:

- Distribute the channels to PFs in round-robin
- Rather than distribute in ranges
- Example, for 2 PFs and 5 channels:

| Channel index | PF index |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 0 |
| 3 | 1 |
| 4 | 0 |

Advantages:

- No channels re-partition/re-shuffle when the number of channels changes
- Persistent statistics : per-ring history is still meaningful

# Design Decisions

## Channels Distribution Policy

Observability

- The relation between PF, irq, napi, and queue can be observed via netlink spec.

```
$ ./tools/net/ynl/cli.py --spec Documentation/netlink/specs/netdev.yaml \
    --dump queue-get --json='{"ifindex": 13}'
[{'id': 0, 'ifindex': 13, 'napi-id': 539, 'type': 'rx'},
 {'id': 1, 'ifindex': 13, 'napi-id': 540, 'type': 'rx'},
 {'id': 2, 'ifindex': 13, 'napi-id': 541, 'type': 'rx'},
 {'id': 3, 'ifindex': 13, 'napi-id': 542, 'type': 'rx'},
 {'id': 4, 'ifindex': 13, 'napi-id': 543, 'type': 'rx'},
 {'id': 0, 'ifindex': 13, 'napi-id': 539, 'type': 'tx'},
 {'id': 1, 'ifindex': 13, 'napi-id': 540, 'type': 'tx'},
 {'id': 2, 'ifindex': 13, 'napi-id': 541, 'type': 'tx'},
 {'id': 3, 'ifindex': 13, 'napi-id': 542, 'type': 'tx'},
 {'id': 4, 'ifindex': 13, 'napi-id': 543, 'type': 'tx'}]

$ ./tools/net/ynl/cli.py --spec Documentation/netlink/specs/netdev.yaml \
    --dump napi-get --json='{"ifindex": 13}'
[{'id': 543, 'ifindex': 13, 'irq': 42},
 {'id': 542, 'ifindex': 13, 'irq': 41},
 {'id': 541, 'ifindex': 13, 'irq': 40},
 {'id': 540, 'ifindex': 13, 'irq': 39},
 {'id': 539, 'ifindex': 13, 'irq': 36}]
```

| Channel index | PF index |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 0 |
| 3 | 1 |
| 4 | 0 |

# Design Decisions

## Channels Distribution Policy

- Here you can clearly observe our channels distribution policy:
  PF0 is at 0000:08:00.0
  PF1 is at 0000:09:00.0

| Channel index | PF index |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 0 |
| 3 | 1 |
| 4 | 0 |

```
$ ./tools/net/ynl/cli.py --spec Documentation/netlink/specs/netdev.yaml \
    --dump napi-get --json='{"ifindex": 13}'
[{'id': 543, 'ifindex': 13, 'irq': 42},
 {'id': 542, 'ifindex': 13, 'irq': 41},
 {'id': 541, 'ifindex': 13, 'irq': 40},
 {'id': 540, 'ifindex': 13, 'irq': 39},
 {'id': 539, 'ifindex': 13, 'irq': 36}]

$ ls /proc/irq/{36,39,40,41,42}/mlx5* -d -1
/proc/irq/36/mlx5_comp0@pci:0000:08:00.0
/proc/irq/39/mlx5_comp0@pci:0000:09:00.0
/proc/irq/40/mlx5_comp1@pci:0000:08:00.0
/proc/irq/41/mlx5_comp1@pci:0000:09:00.0
/proc/irq/42/mlx5_comp2@pci:0000:08:00.0
```

# Implementation details

Some implementation details:

- Init/destroy flow (probe):
  - Create netdev once all PFs are probed
  - Symmetrically, destroy netdev whenever any of the PFs is removed
- Software-level communication between PFs
  - Collaborate to make it work
- Devcom: mlx5 device driver communication infrastructure
- Use it for a “leader election” algorithm, to chose “primary PF”

# Implementation details

## Primary PF Election

- Desired attributes of leader (Primary PF) election algorithm:
  - Simple
  - Deterministic
  - Predictable
  - Persistent between reboots

- Keep same net device name
- Keep same channels indexing/distribution to PFs
- Keep same PF for RSS and RX steering

- Keep admin configuration scripts simple
- Good user experience

- Algorithm: PF with smallest ID is elected a leader

# Implementation details

## RX Steering

RX steering

- RX steering objects are not multiplied
- One instance, belongs to the primary PF
- One RSS table, the primary PF domain

- RSS table and steering rules can redirect incoming traffic to RX queues of other PFs
- At this stage, no need for PF-2-PF software communication
- netdev (and its private areas) are available by this point

- Requires hardware support

# Implementation details

## IRQ and XPS

IRQ and XPS

- Match the cpus according to the channel distribution
- Use existing cpu core distance proximity scheme
- Alternate PFs as input
- Example:

```
NUMA node(s):              2
NUMA node0 CPU(s):         0-11
NUMA node1 CPU(s):         12-23
```

PF0 on NUMA #0

PF1 on NUMA #1

| Channel index | PF index | IRQ affinity |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 12 |
| 2 | 0 | 1 |
| 3 | 1 | 13 |
| 4 | 0 | 2 |
| 5 | 1 | 14 |
| 6 | 0 | 3 |
| 7 | 1 | 15 |
| 8 | 0 | 4 |
| … | | |
| 20 | 0 | 10 |
| 21 | 1 | 22 |
| 22 | 0 | 11 |
| 23 | 1 | 23 |

```
/sys/class/net/eth2/queues/tx-0/xps_cpus:000001
/sys/class/net/eth2/queues/tx-1/xps_cpus:001000
/sys/class/net/eth2/queues/tx-2/xps_cpus:000002
/sys/class/net/eth2/queues/tx-3/xps_cpus:002000
/sys/class/net/eth2/queues/tx-4/xps_cpus:000004
/sys/class/net/eth2/queues/tx-5/xps_cpus:004000
/sys/class/net/eth2/queues/tx-6/xps_cpus:000008
/sys/class/net/eth2/queues/tx-7/xps_cpus:008000
/sys/class/net/eth2/queues/tx-8/xps_cpus:000010
                                …
/sys/class/net/eth2/queues/tx-20/xps_cpus:000400
/sys/class/net/eth2/queues/tx-21/xps_cpus:400000
/sys/class/net/eth2/queues/tx-22/xps_cpus:000800
/sys/class/net/eth2/queues/tx-23/xps_cpus:800000
```

# Implementation details

## RX / TX affinities

TX is perfectly affined by XPS

RX does hash-based RSS in default

- Cannot predict the correct PF / channel

- aRFS?
- Static RX steering rules?

# Performance

# Performance

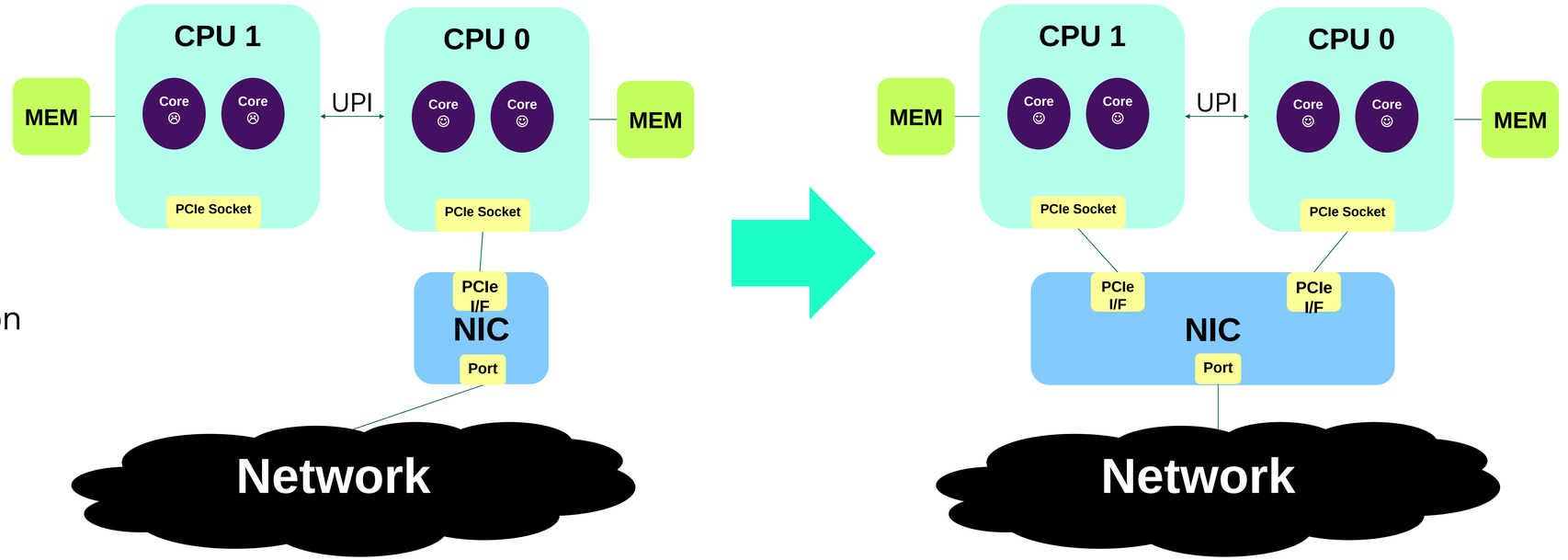
## Setting

DUT setup:

- Processor: Intel(R) Xeon(R) Platinum 8470 CPU @2.00GHz
- Before: Single PF, 200Gbps, on NUMA #0
- After: Socket-Direct system, two PFs (NUMA #0 and NUMA #1), 200Gbps port, single netdev

Setting:

- Multi-ring, multi-core, multi-stream, on 2 NUMAs.
- 1:1:1 mapping
  - Reduce number of variables
  - Performance stability
  - Apples-to-apples comparison



# Performance

## Setting

Setting:

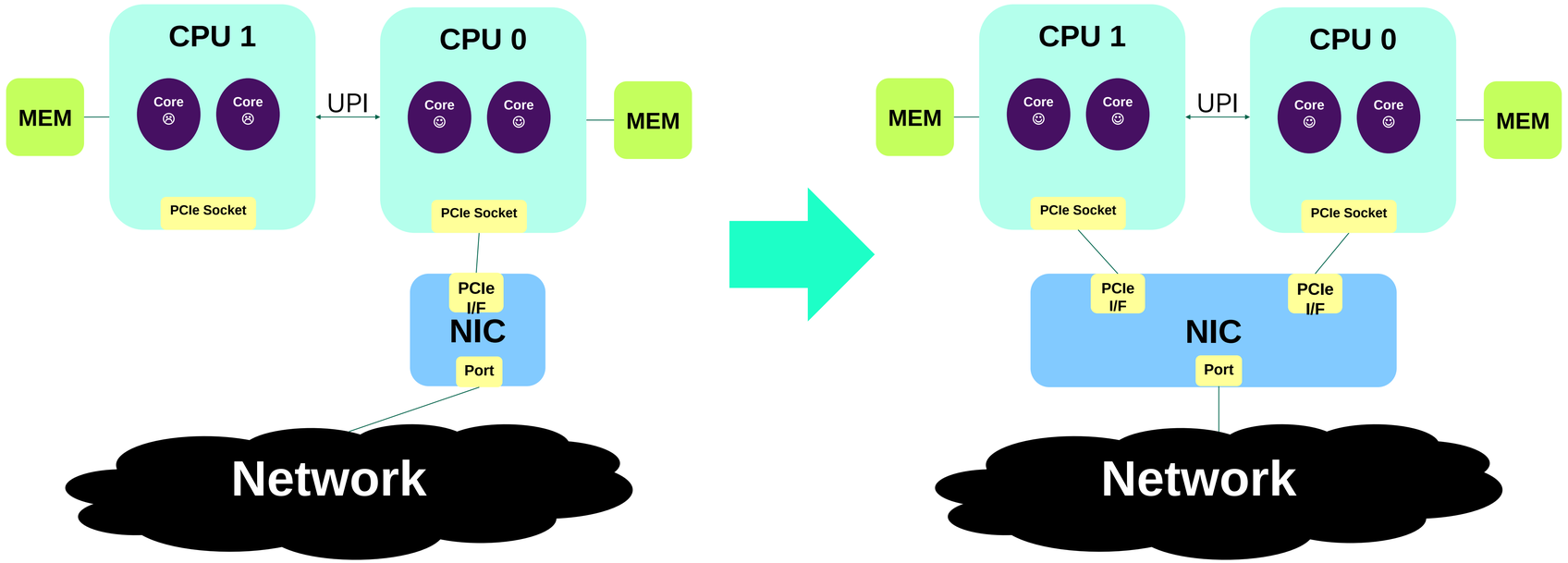
- Multi-ring, multi-core, multi-stream, on 2 NUMAs.
- 1:1:1 mapping

BW tests:

1. DUT does RX
2. DUT does TX

- Monitored:
  - Inter-processor BW (UPI)
  - Memory
  - Power

Latency test



# Performance

## BW Tests: Inter-Processor Throughput

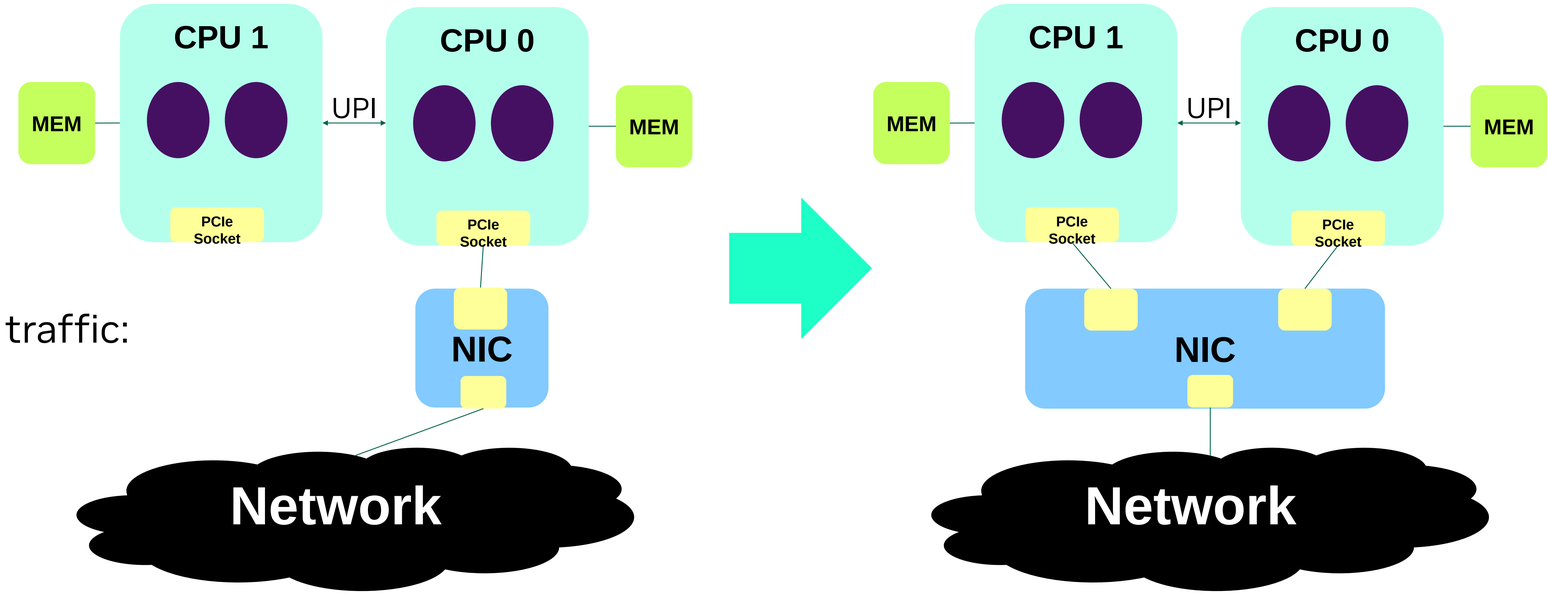
- Inter-processor throughput
  - pcm tool

RX test:

- Total UPI incoming data traffic:
  Before: 15 GBytes
  After: 1.4 Gbytes (~10 times less)
- Total UPI outgoing data and non-data traffic:
  Before: 49 GBytes
  After: 4.4GBytes (~11 times less)

TX test:

- Total UPI incoming data traffic:
  Before: 12 Gbytes
  After: 0.15 G (~80 times less)
- Total UPI outgoing data and non-data traffic:
  Before: 33 G
  After: 1.6G (~20 times less)



# Performance

## BW Tests: Memory Throughput

Memory observations:

- Remote dma writes do not go to DDIO
  - Make the processor cache the primary destination and source of I/O data
- Writes go to RAM, rather than LLC cache
- Expect high memory bandwidth on remote NUMA node
- Expect high latency in latency test

# Performance

## BW Tests: Memory Throughput

pcm-memory

RX test

- Before:

```
|---------------------------------------||---------------------------------------|
|--             Socket  0             --||--             Socket  1             --|
|---------------------------------------||---------------------------------------|
|-- NODE 0 Mem Read (MB/s) :  1207.72 --||-- NODE 1 Mem Read (MB/s) :  1803.17 --|
|-- NODE 0 Mem Write(MB/s) :  1021.13 --||-- NODE 1 Mem Write(MB/s) : 15041.88 --|
|-- NODE 0 Memory (MB/s):     2228.85 --||-- NODE 1 Memory (MB/s):    16845.04 --|
|---------------------------------------||---------------------------------------|
|---------------------------------------||---------------------------------------|
|--                System Read Throughput(MB/s):       3010.89                 --|
|--               System Write Throughput(MB/s):      16063.01                 --|
|--             System Memory Throughput(MB/s):      19073.89                 --|
|---------------------------------------||---------------------------------------|
```

- After:

```
|---------------------------------------||---------------------------------------|
|--             Socket  0             --||--             Socket  1             --|
|---------------------------------------||---------------------------------------|
|-- NODE 0 Mem Read (MB/s) :   941.00 --||-- NODE 1 Mem Read (MB/s) :   745.44 --|
|-- NODE 0 Mem Write(MB/s) :   853.43 --||-- NODE 1 Mem Write(MB/s) :   728.42 --|
|-- NODE 0 Memory (MB/s):     1794.43 --||-- NODE 1 Memory (MB/s):     1473.86 --|
|---------------------------------------||---------------------------------------|
|---------------------------------------||---------------------------------------|
|--                System Read Throughput(MB/s):       1686.44                 --|
|--               System Write Throughput(MB/s):       1581.84                 --|
|--             System Memory Throughput(MB/s):       3268.29                 --|
|---------------------------------------||---------------------------------------|
```

# Performance

## BW Tests: Memory Throughput

pcm-memory

TX test

- Before:

```
|---------------------------------------||---------------------------------------|
|--             Socket  0             --||--             Socket  1             --|
|---------------------------------------||---------------------------------------|
|-- NODE 0 Mem Read (MB/s) :   611.38 --||-- NODE 1 Mem Read (MB/s) : 12144.53 --|
|-- NODE 0 Mem Write(MB/s) :   419.18 --||-- NODE 1 Mem Write(MB/s) :   431.58 --|
|-- NODE 0 Memory (MB/s):     1030.56 --||-- NODE 1 Memory (MB/s):    12576.10 --|
|---------------------------------------||---------------------------------------|
|---------------------------------------||---------------------------------------|
|--                 System Read Throughput(MB/s):      12755.90                 --|
|--                System Write Throughput(MB/s):        850.76                 --|
|--              System Memory Throughput(MB/s):      13606.66                 --|
|---------------------------------------||---------------------------------------|
```

- After:

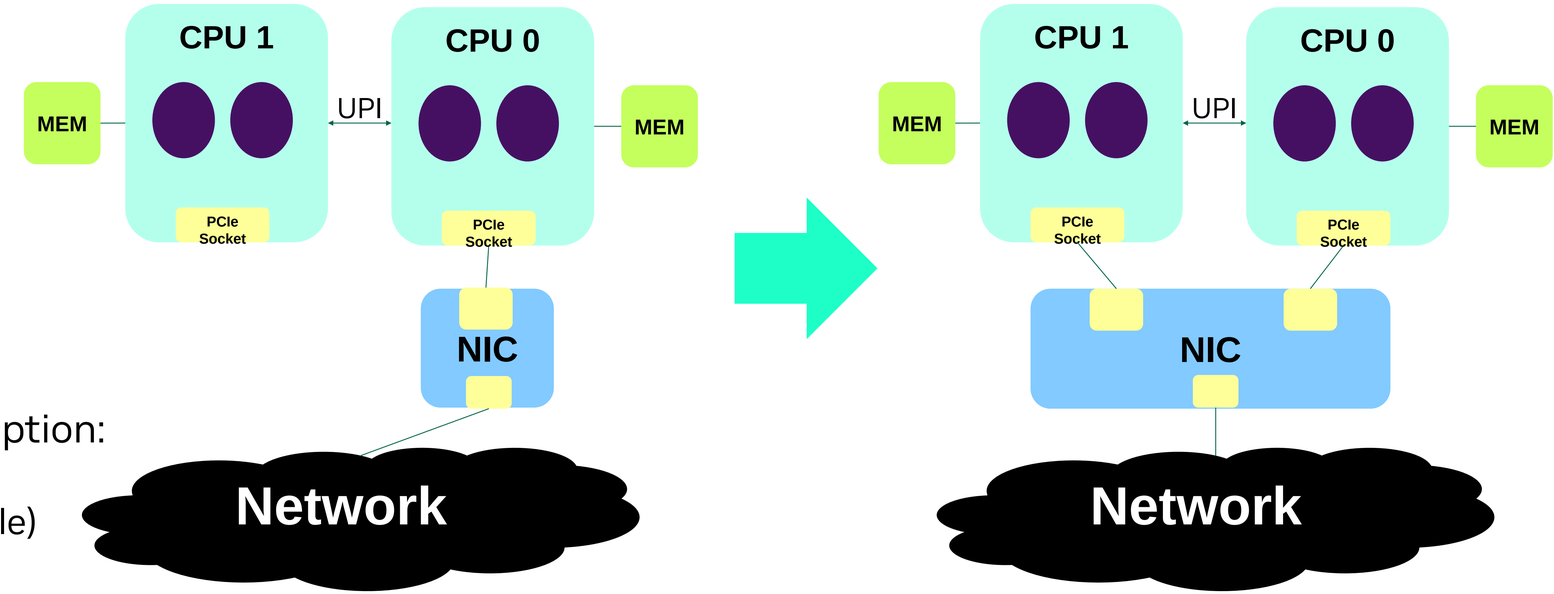
```
|---------------------------------------||---------------------------------------|
|--             Socket  0             --||--             Socket  1             --|
|---------------------------------------||---------------------------------------|
|-- NODE 0 Mem Read (MB/s) :   936.40 --||-- NODE 1 Mem Read (MB/s) :  1022.60 --|
|-- NODE 0 Mem Write(MB/s) :   651.98 --||-- NODE 1 Mem Write(MB/s) :   647.38 --|
|-- NODE 0 Memory (MB/s):     1588.37 --||-- NODE 1 Memory (MB/s):     1669.98 --|
|---------------------------------------||---------------------------------------|
|---------------------------------------||---------------------------------------|
|--                 System Read Throughput(MB/s):       1959.00                 --|
|--                System Write Throughput(MB/s):       1299.35                 --|
|--              System Memory Throughput(MB/s):       3258.35                 --|
|---------------------------------------||---------------------------------------|
```

# Performance

## BW Tests: Power Consumption

Power consumption

- Measured for the whole system through external device
- Covers
  - NIC
  - PCI
  - CPUs
  - Inter-processor
  - Memory
  - Etc…
- Measured additional power consumption:
  - additional consumption = power(during test) – power(idle)



# Performance

## BW Tests: Power Consumption

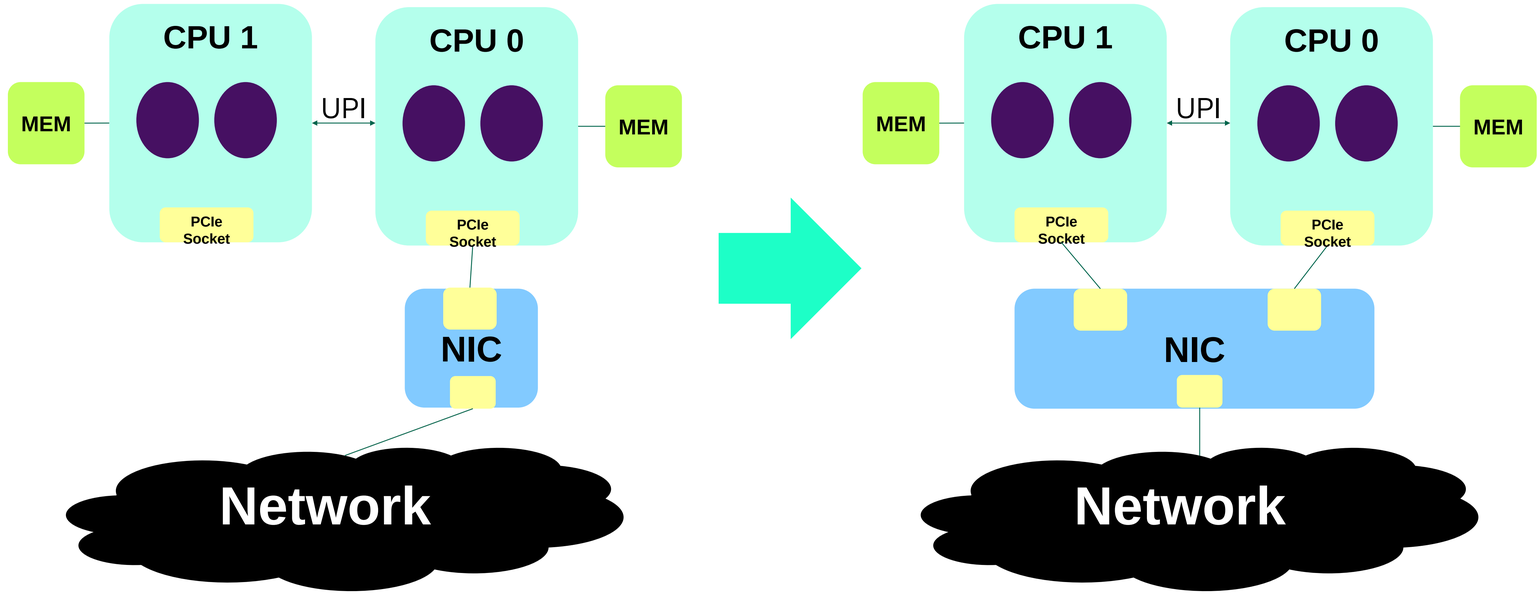
Power consumption

- additional consumption =
  power(during test) – power(idle)

- RX test
  Before: 172 Watt
  After: 164 Watt (5% saving)
- TX test
  Before: 119 Watt
  After: 112 Watt (6% saving)

- Save power
- Save money
- Save earth ??

- This speaks to everyone…



# Performance

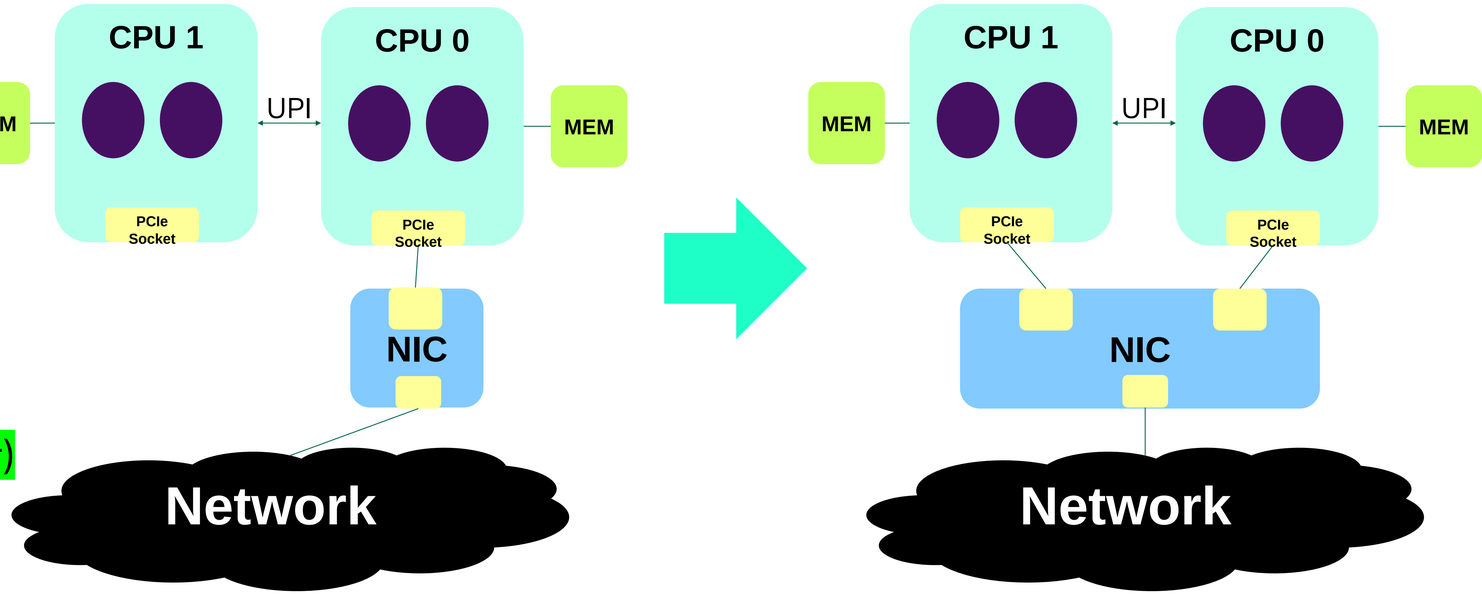
## Latency Test

- Latency test: netperf TCP_RR
- Single ring, single core (irq, napi, stack, app)

- Client side is fixed, server side is the DUT.
  - One-sided changes in a two-sided latency test.
  - Isolated improvement is even higher.

- Run on NUMA #0 core:
  Before: 52K transactions/sec
  After: 52K transactions/sec (expected)

- Run on NUMA #1 core:
  Before: 43K transactions/sec
  After: 52K transactions/sec (~20% faster)
  - Number became similar to NUMA #0 core
  - “local” once again!



# Future Work

# Future Work

- Test and extend support beyond 2 PFs for future hardware
- Possible extensions to other function types (VFs, SFs)

- Possibly add dynamic PF addition/deletion to existing netdev
  - Adds complexity
  - Real use case?

- Improve sysfs observability and control
  - Today, sysfs links netdev only to its primary PF, and vice versa

- Hope that more vendors jump in
- Generalize the logic into common netdev APIs
  - Software communication of PFs through generic API (non-mlx5)
    - drivers/base/component.c ?
  - Leader election logic
  - XPS and IRQ logic

# References


NVIDIA Socket-Direct
https://www.nvidia.com/en-us/networking/ethernet/socket-direct/

Achiad’s netdev 2.2 presentation
https://netdevconf.info//2.2/session.html?shochat-devicemgmt-talk

Netdev 0x18 presentation
https://netdevconf.info/0x18/sessions/talk/multi-pf-single-netdev.html

Kernel patches
https://lore.kernel.org/all/20240215030814.451812-1-saeed@kernel.org/

Linux Kernel Documentation
https://docs.kernel.org/networking/multi-pf-netdev.html


NVIDIA
Questions?
Thanks