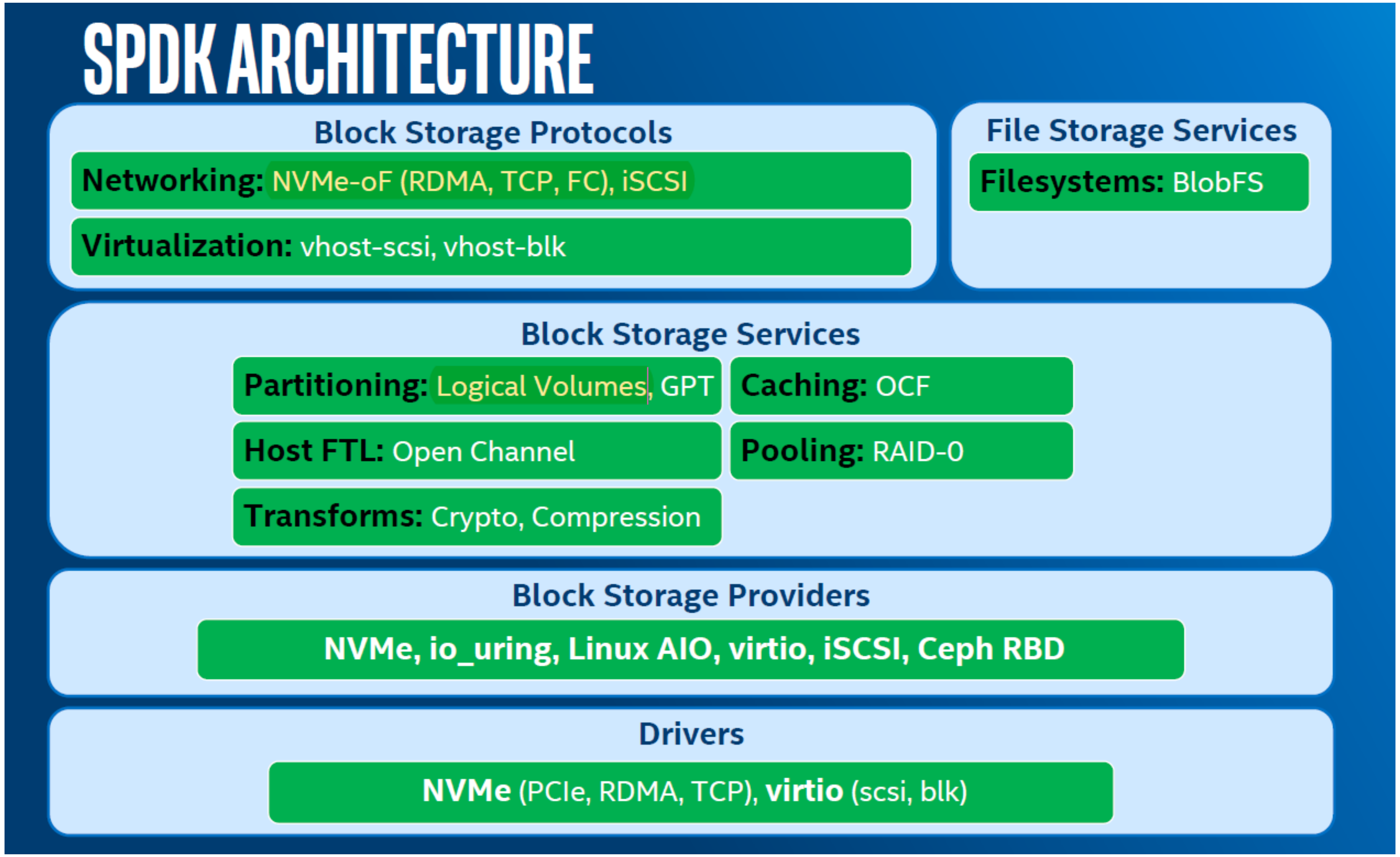
Embrace high performance storage with open arm

Richael Zhuang
arm

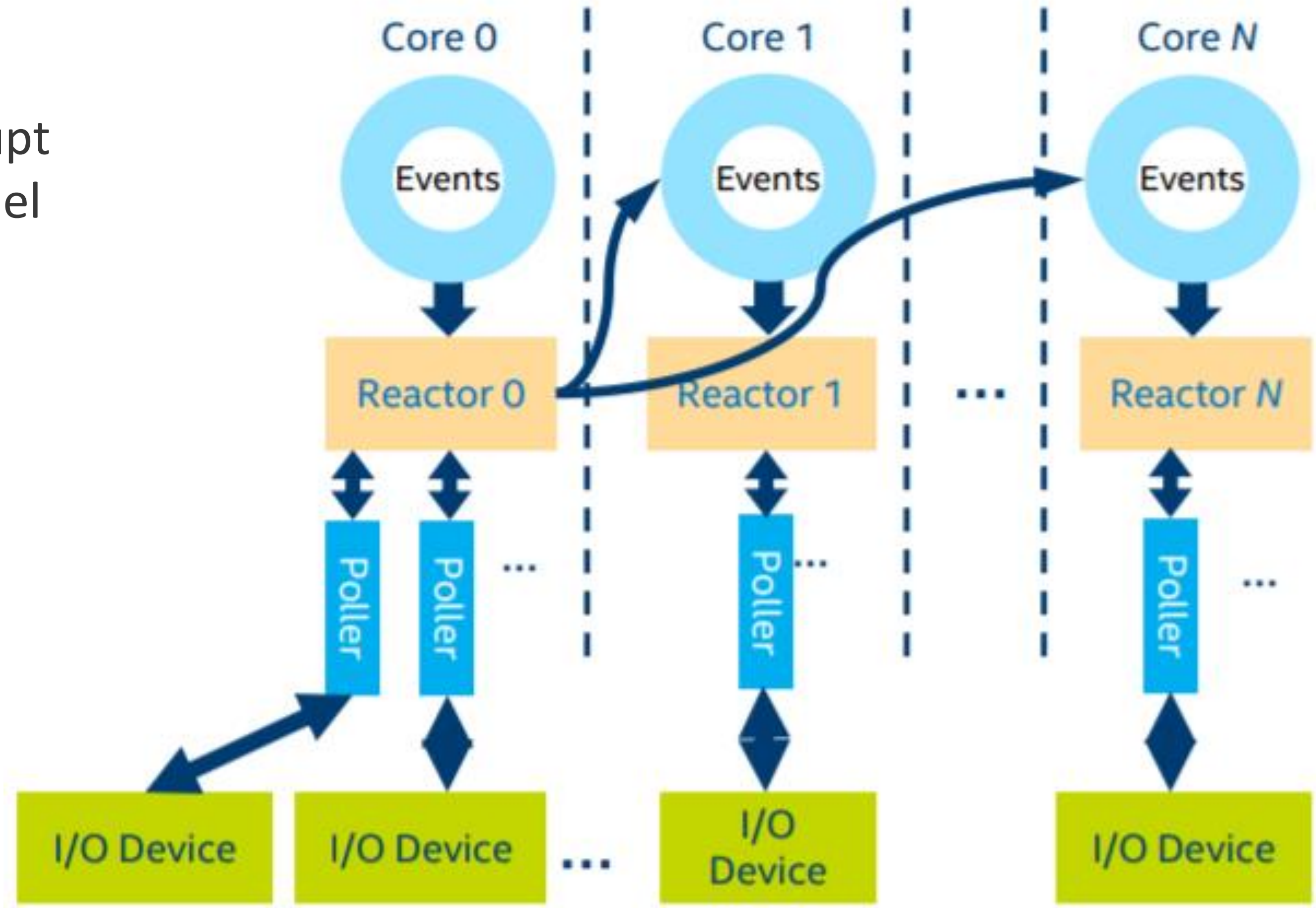
What’s SPDK

- Storage Performance Development Kit
- A set of tools and libraries to create high performance , scalable, user mode storage applications



What’s SPDK

- Key techniques
  - User mode driver(uio/vfio)
  - Poll mode instead of interrupt
  - Shared-nothing thread model



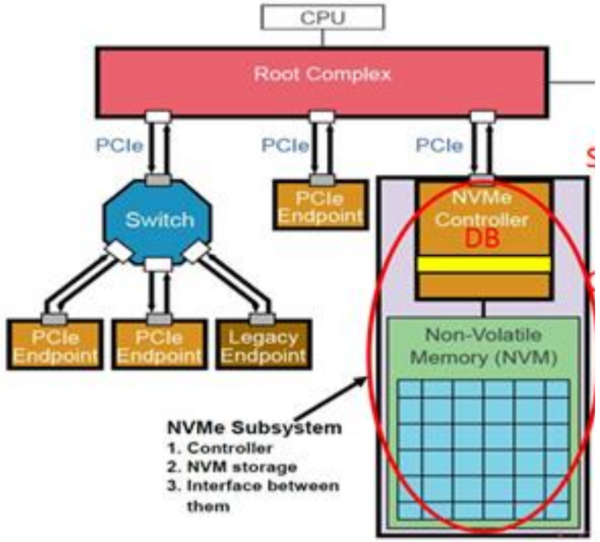
SPDK on Arm64

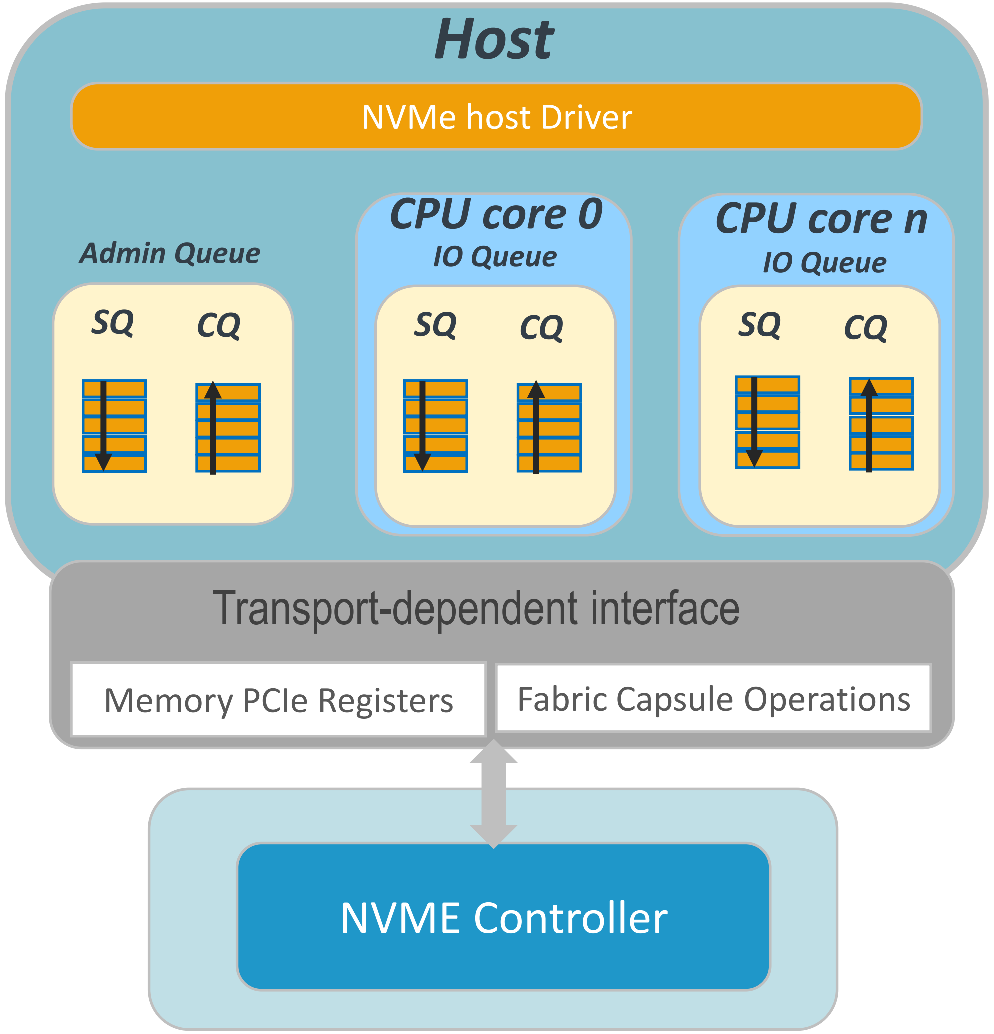
- 70+ patches to enable and optimize SPDK are merged

- SPDK NVMe over TCP
- SPDK NVMe over TCP tuning on Arm64
- SPDK NVMe over TCP optimization

NVMe over Fabrics

- Local access ：pcie (shared memory)
  - NVMe：Specification for SSD access via PCI Express (PCIe)
- Remote access： message based transport
  - fibre channel/RDMA/TCP
- NVMe three key elements
  - SQ/CQ（in host memory in general）/DB（doorbell register, in controller）
    - SQ entry： 64byte command (data: 2 PRP or 1 SGL)
    - CQ entry： 16byte status

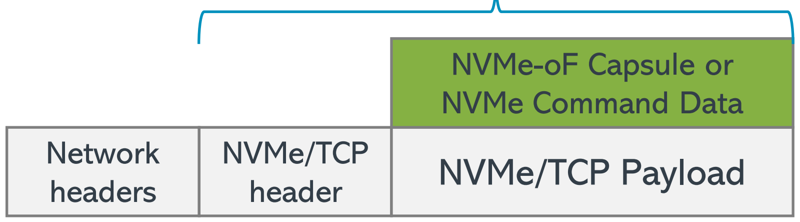


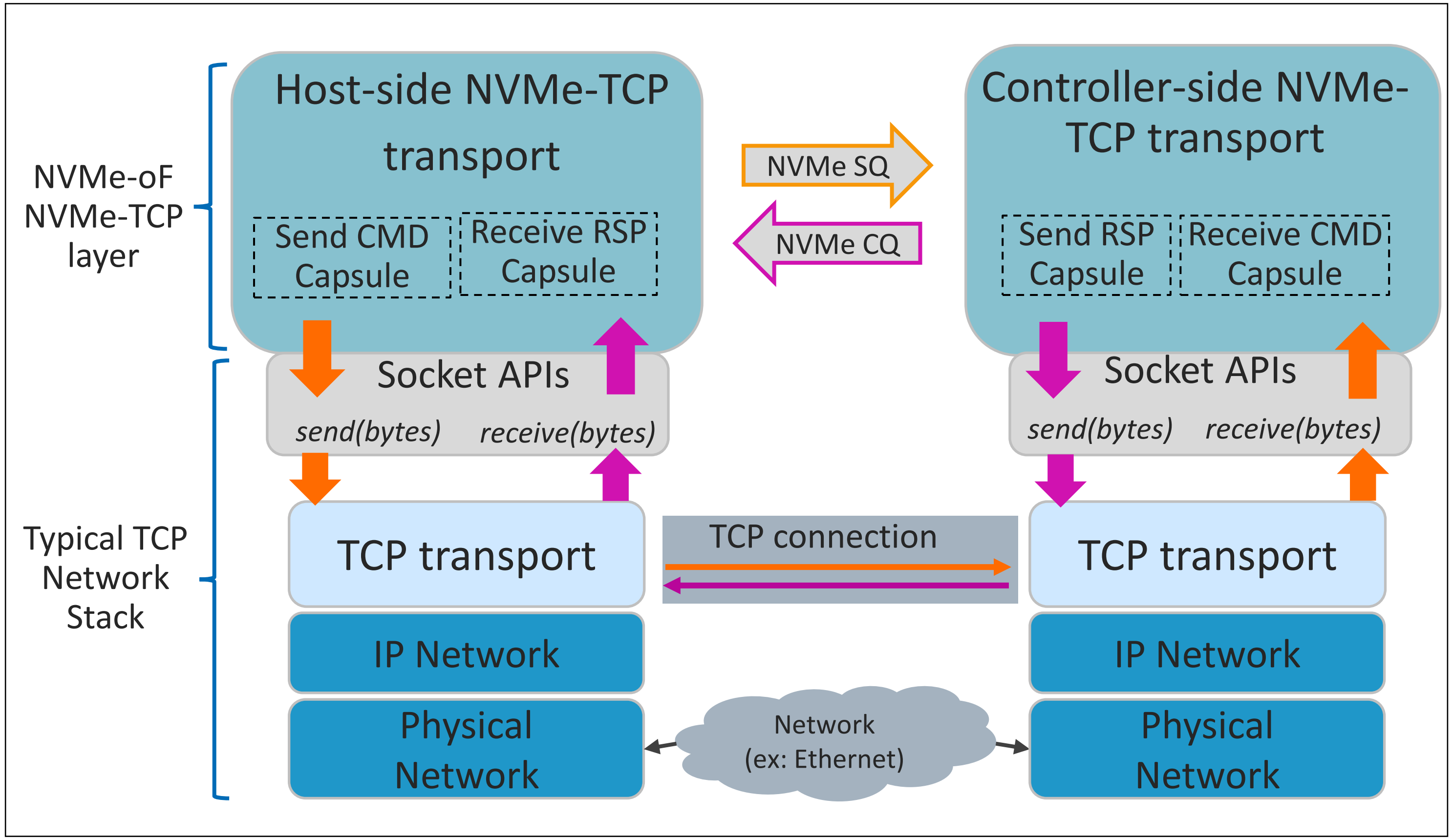


NVMe over TCP

- NVMe block storage protocol over standard TCP/IP transport
- Each NVMe queue pair（SQ+CQ）mapped to a TCP connection（SQ：CQ=1：1)
- NVMe-oF Commands and data sent over standard TCP/IP sockets

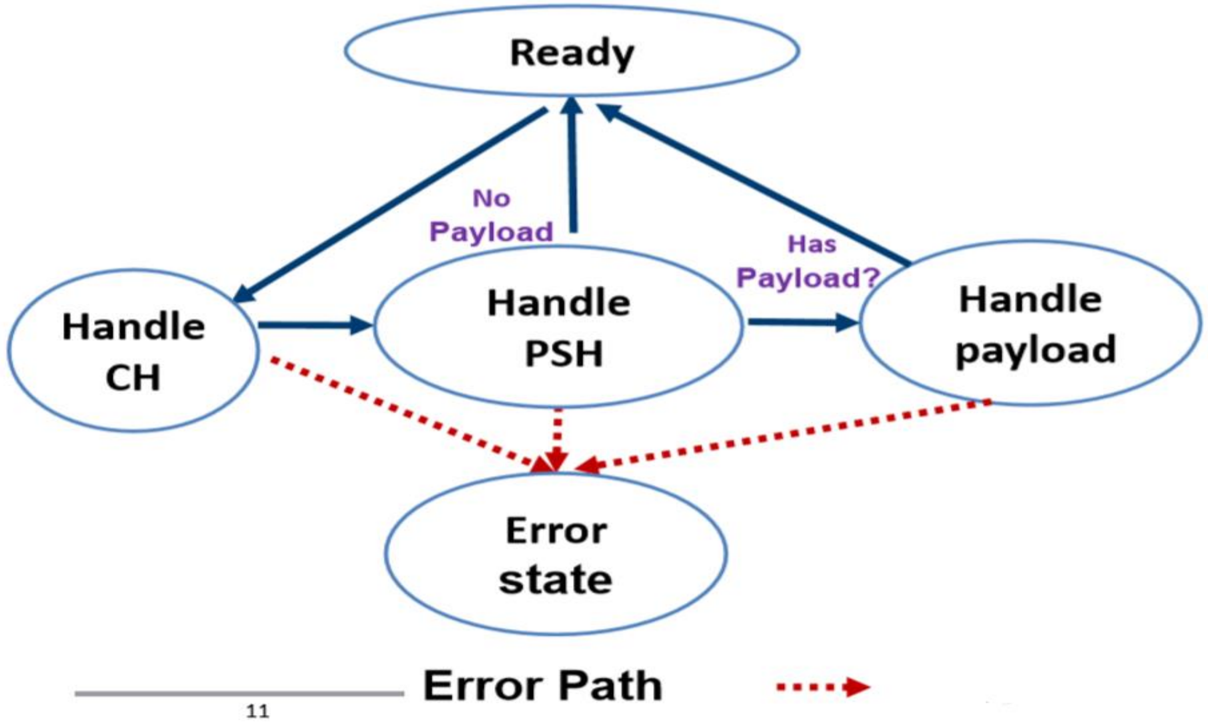
NVMe/TCP PDU

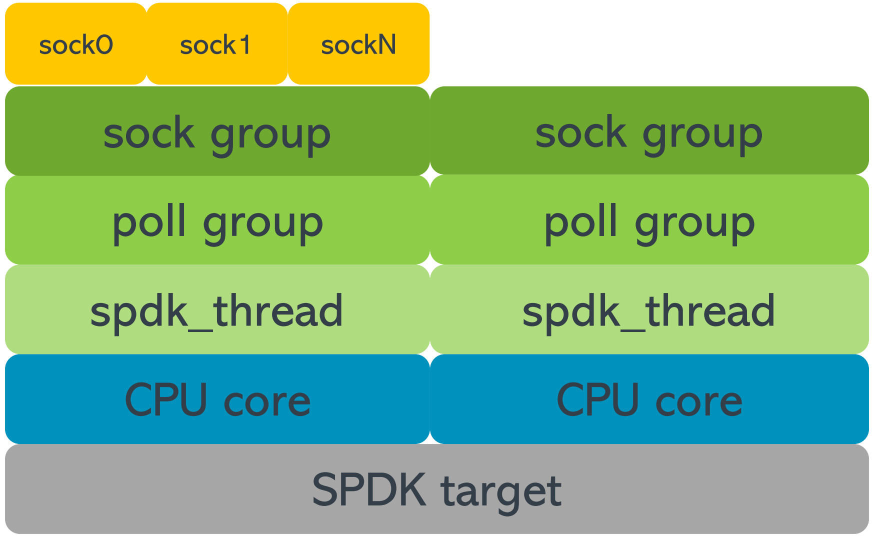


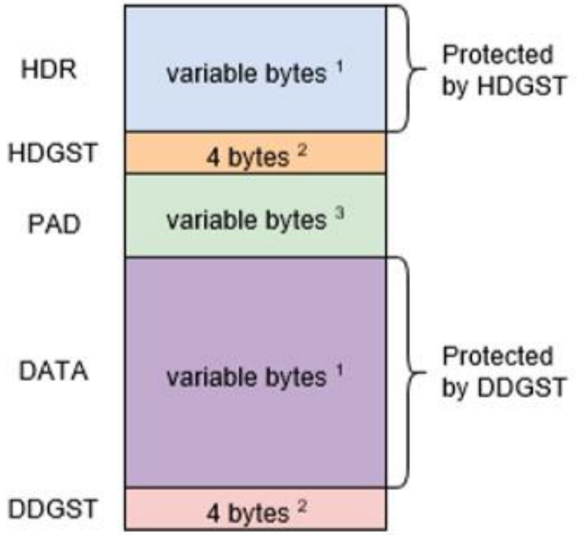


Spdk NVMe over TCP

- Epoll to get events of sockets(POSIX)
- Each TCP PDU receiving handling



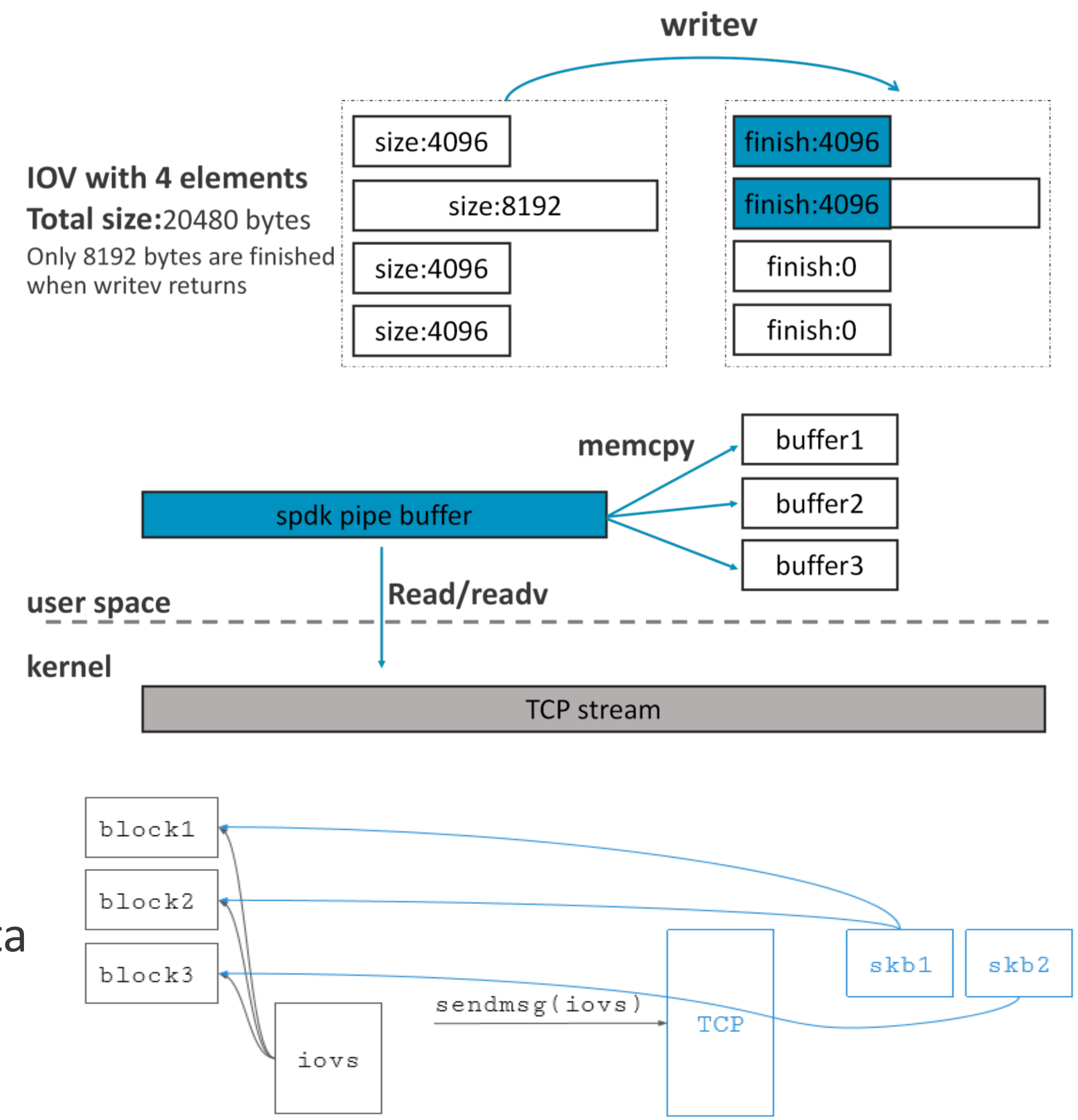




NVMe/TCP PDU

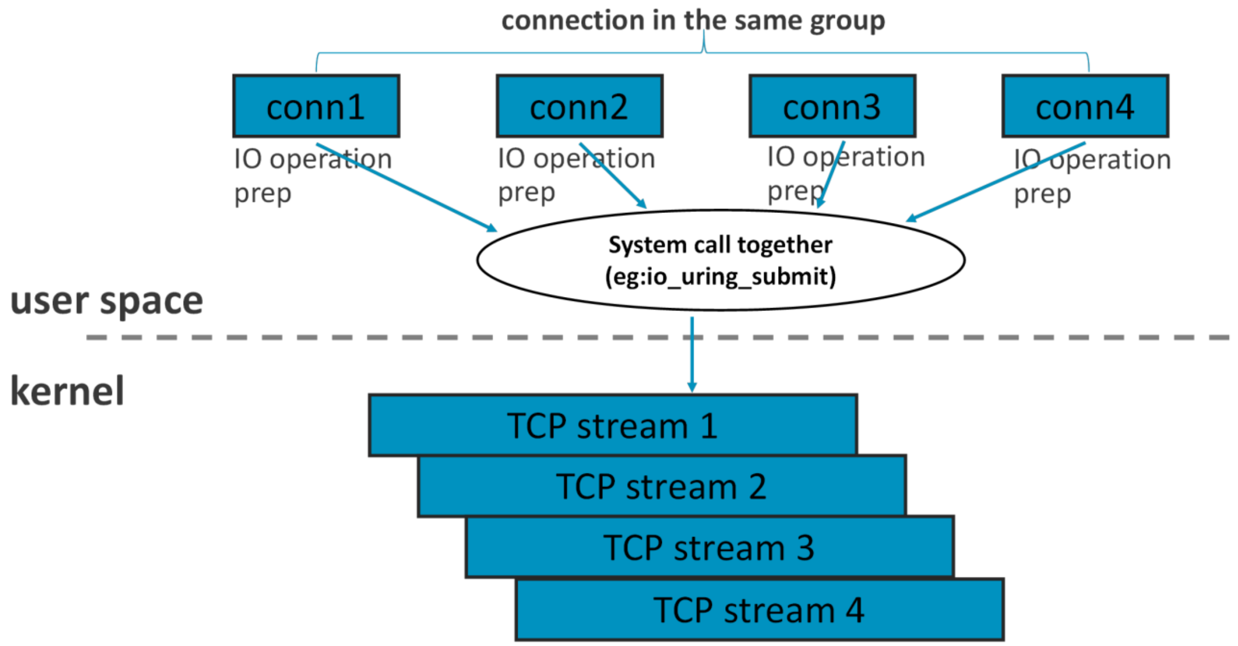
Original optimization in SPDK—single socket

- non-blocking IO   O_NONBLOCK
  - Partial write/read problem
- Pipe buffer—reduce syscall
  - Balance syscall and memcpy
- Batch write--reduce syscall
  - merge multi writev operations into a single one
- Sendmsg zerocopy
  - >= linux4.1.4, setsockopt(SO_ZEROCOPY)
  - Data sent via sendmsg(data,MSG_ZEROCOPY)
  - Packets(e.g. skbuffs in linux) keep references to the data
  - Data must remain unmodified while an skb points to it



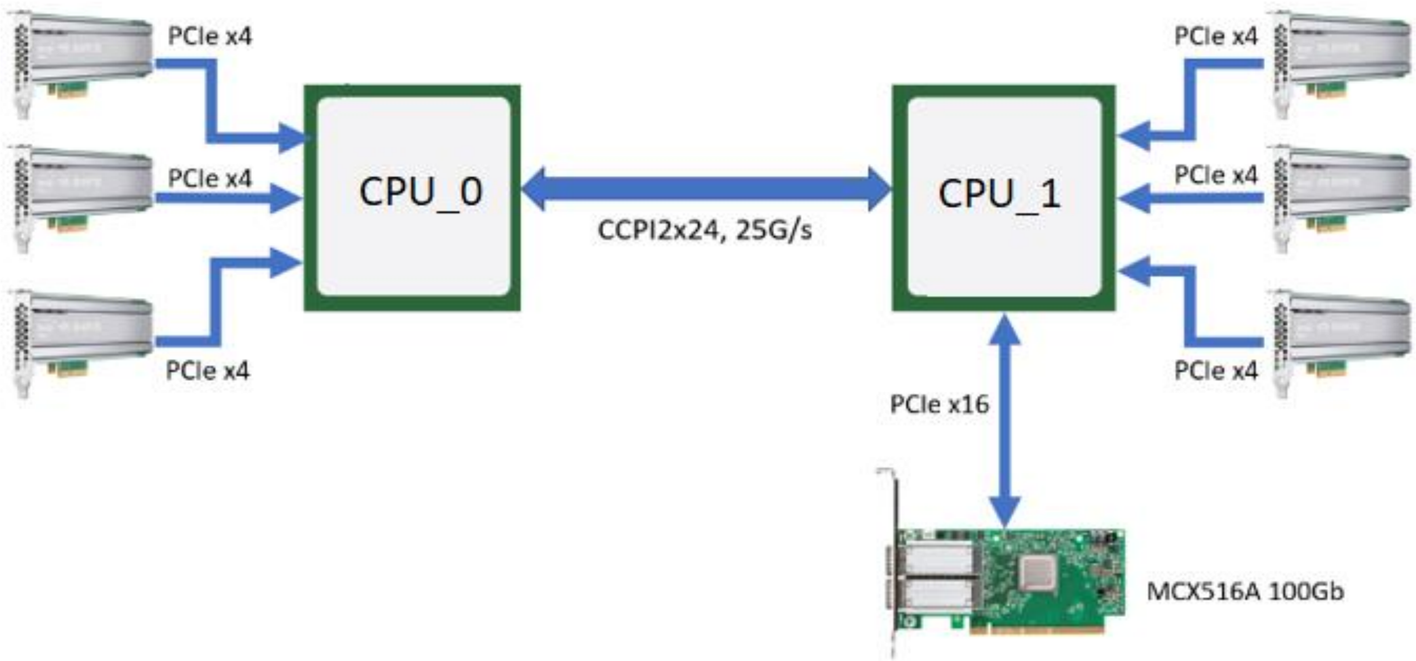
Original optimization in SPDK—multi sockets

- Each socket is managed by only one SPDK thread to avoid competition
- Epoll in Posix to reduce syscall
- In Uring sock implementation, we can submit write operations of all socket in a poll group in one time



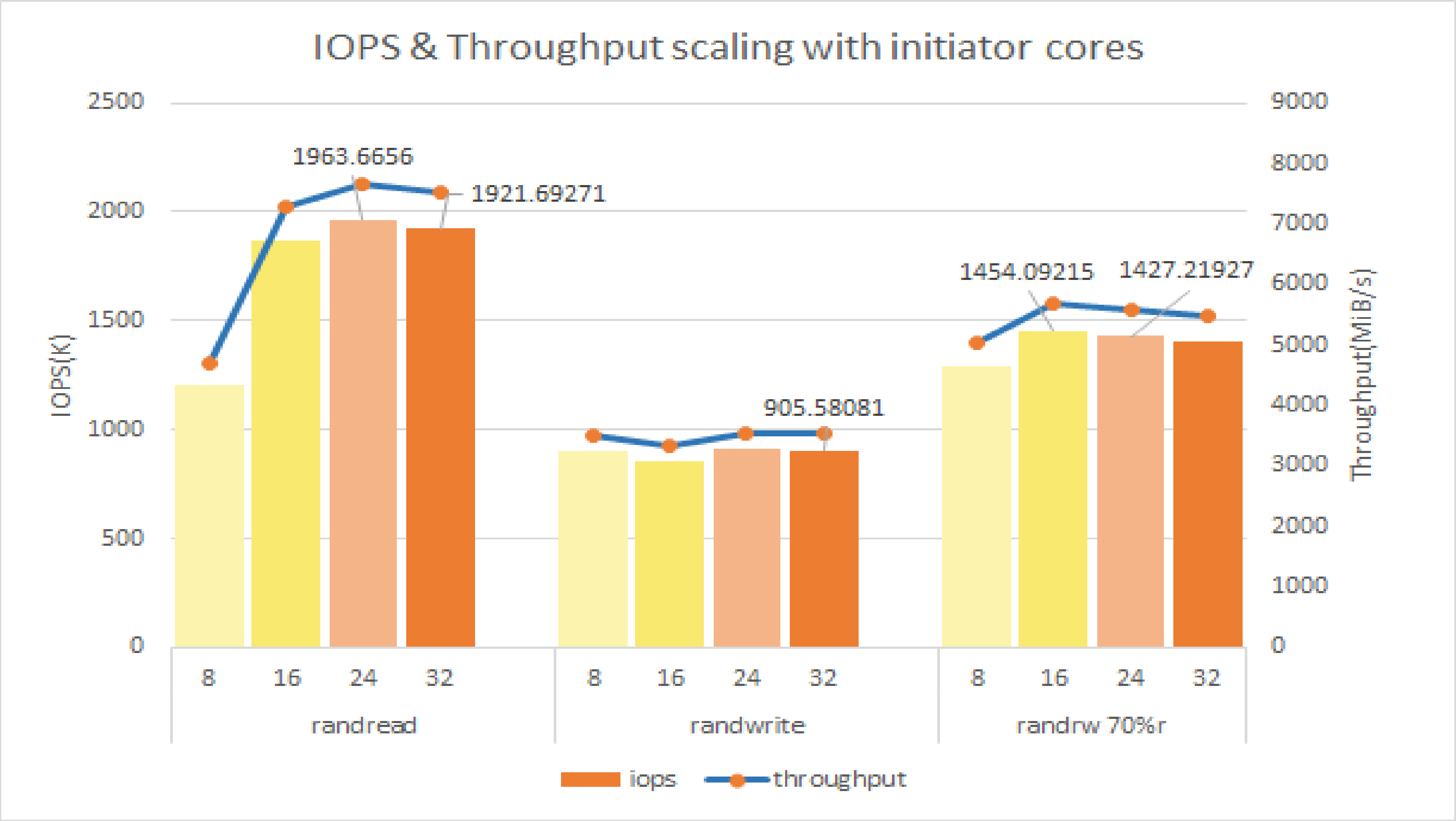
SPDK NVMe over TCP performance(posix)

- System configuration
  - 4KB kernel page size
  - Hugepage affinity : 2MB hugepagesz, and make sure there are enough hugepages(>=2GB) on the numa
  - iommu.passthrough=1
  - vfio-pci driver



SPDK NVMe over TCP performance(posix socket)

- 6 NVMe P4600 in target, target runs 8 cores
- 4KB payload size, 128 queue depth



Tuning on Arm64-1

- PCIe
  - maxpayload size(up to 4096): maximum TLP(transaction layer packet) payload size
  - MaxReadReq: default 512(up to 4096)
- NIC irq affinity/offload/irq coalescence/…
  - Combined rx/tx queues: 4;
  - Rx/tx queue depth : 1024;
  - NIC irqs bound to 4 cores on numa1(NIC on numa1); (service irqbalance stop)

  - adaptive-rx=off, adaptive-tx=off,
  - rx-usecs 64 rx-frames 128 tx-usecs 128 tx-frames 128

  - TSO tcp-segmentation-offload: on
  - GSO generic-segmentation-offload: on
  - GRO generic-receive-offload: on

Tuning on Arm64-2

- Kernel TCP parameters
  - # Set 256MB buffers
  - net.core.rmem_max = 268435456
  - net.core.wmem_max = 268435456
  - # Increase autotuning TCP buffer limits 128MB
  - # min, max and default settings
  - net.ipv4.tcp_rmem = 4096 87380 134217728
  - net.ipv4.tcp_wmem = 4096 65536 134217728

- Interrupt coalescing(soft irq)
  - net.core.netdev_budget = 300
  - net.core.netdev_budget_usecs = 8000
  - net.core.dev_weight = 64

- Queue discipline
  - ingress qdisc : net.core.netdev_max_backlog = 262144
  - outgress qdisc: txqueuelen=1000 (ifconfig)

  - default qdisc: sysctl net.core.default_qdisc = fq_codel

Tuning on Arm64-3

- SPDK parameters
  - "in_capsule_data_size": 8192,
  - "c2h_success": true,
  - "enable_recv_pipe": true,
  - "enable_zerocopy_send": true,

  …….

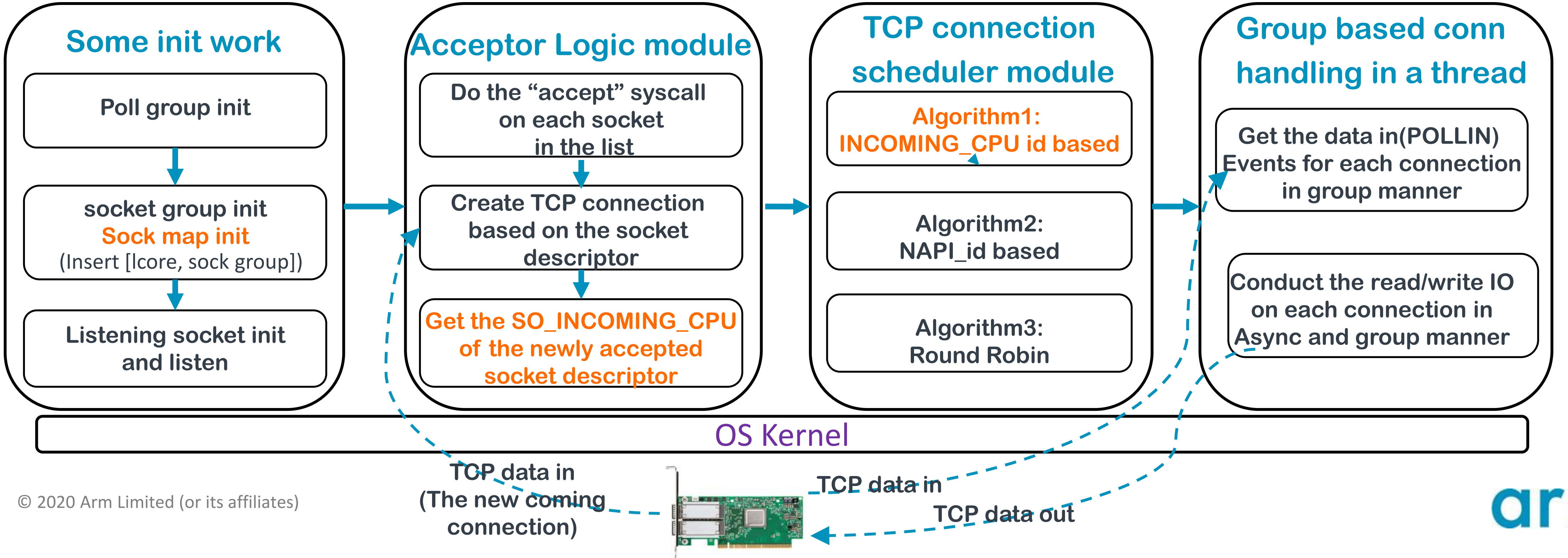
Optimization-1

- SO_INCOMING_CPU
  - gets the CPU affinity of a socket in target, and allocate the socket to the corresponding CPU to handle
  - 11%~17% randwrite performance boost for posix, and 8%~12% for uring

(test with: 6 P4600 NVMe on target,target uses 8 cores, NIC irqs are bound to these 8 cores, and initiator side uses 24 and 32 cores)

Patch link: https://review.spdk.io/gerrit/c/spdk/spdk/+/5748

Leverage INCOMING_CPU in SPDK NVMe-oF



Optimization-2

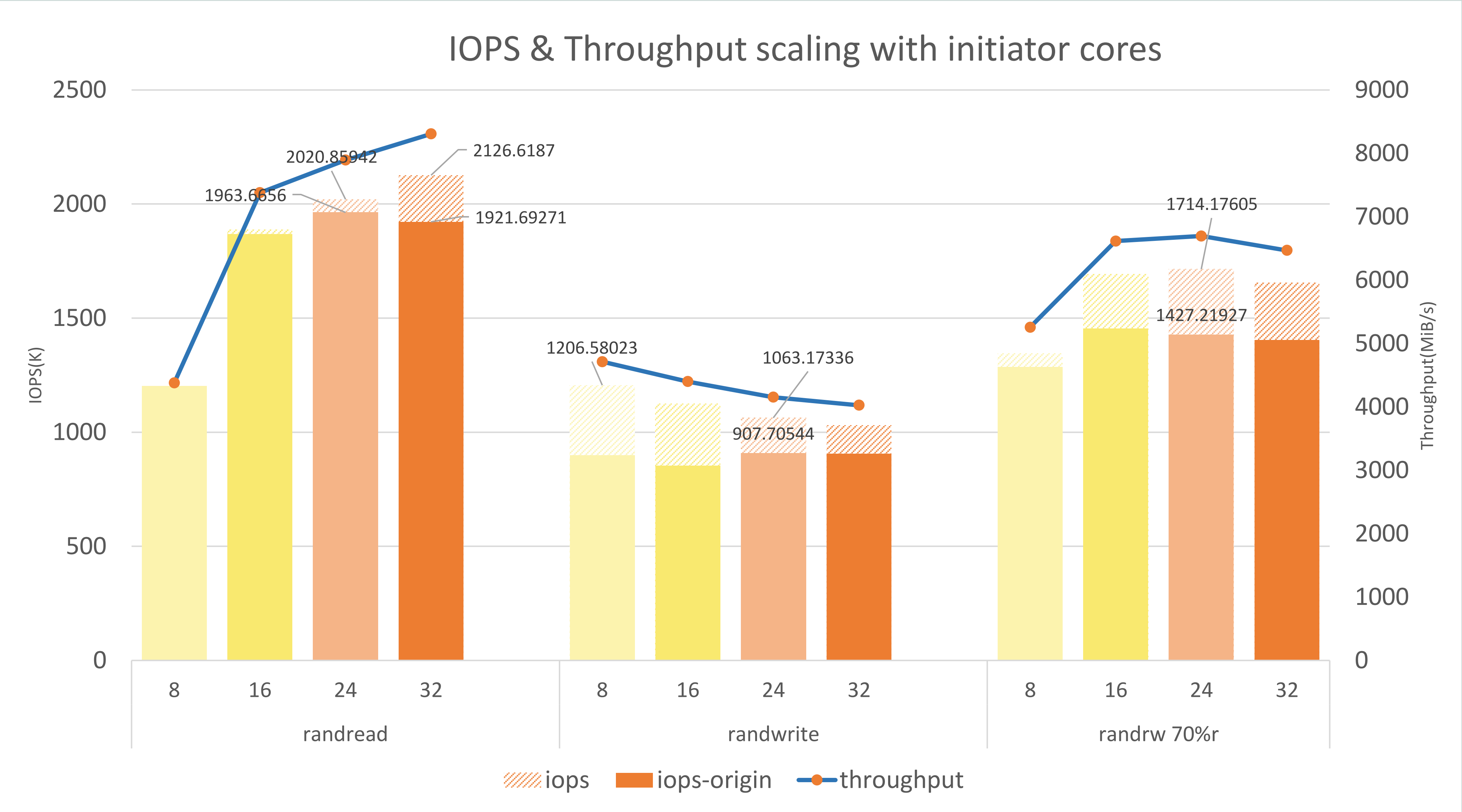
- Smooth Weighted Round Robin
- why
  - Performance with Irqs bound to 4 cores is better than irqs bound to 8 cores
  - The 4 cores that deal with irqs have more cpu overhead than other 4 cores
- algorithm
  - {current_weight, weight} , current_weight init to 0;
  - on each peer selection, increase current_weight of each eligible peer by its weight；
  - select peer with greatest current_weight and reduce its current_weight by total number of weight points distributed；
  - Eg: {a:5, b:1, c:1) { c, b, a, a, a, a, a } -> { a, a, b, a, c, a, a }

  algorithm from https://github.com/phusion/nginx/commit/27e94984486058d73157038f7950a0a36ecc6e35,

  proof from https://tenfy.cn/2018/11/12/smooth-weighted-round-robin/
- 8% ~14% randread improvement with weight "2,2,2,2,5,5,5,5" ("2" for cpus that NIC irqs are bound to)

  (test with :6 P4600 NVMe on target,target uses 8 cores,initiator runs 24 or 32 cores, NICs are set with 4 combined queues, irqs are bound to 4 cores)

SPDK NVMe over TCP performance（posix socket）



Following work

- Continue to tuning SPDK NVMe over Fabrics on Arm64
- Optimization SPDK with SVE
- …

Q&A

richael.zhuang@arm.com