

DeLiBA-K: Speeding-up Hardware-Accelerated Distributed Storage Access by Tighter Linux Kernel Integration and Use of Modern API



Babar Khan, Andreas Koch

Embedded Systems and Applications Group, TU Darmstadt, Germany



TECHNISCHE
UNIVERSITÄT
DARMSTADT



**Tenth International Workshop on Heterogeneous High-performance Reconfigurable Computing H2RC'24
SC24: The International Conference for High Performance Computing, Networking, Storage and Analysis**

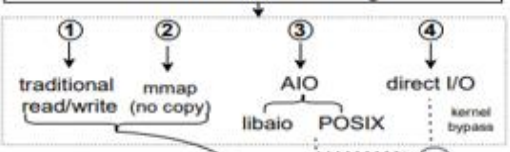
Friday, 22-November-2024, Atlanta, Georgia, USA (talk in-person/on site)

Mainly following four:

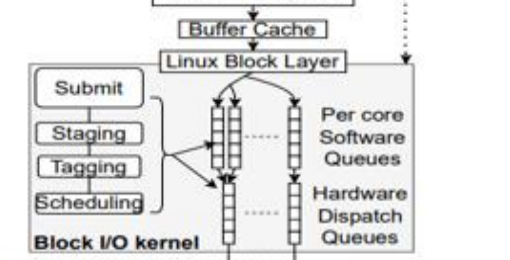
- Background (primer on I/O storage) and Research Problem
- FPGA based DeLiBA frameworks and new DeLiBA-K framework
- Hardware Evaluation and overall FPGA based speed-ups
- Conclusion and Future Work

Background and Research Problem

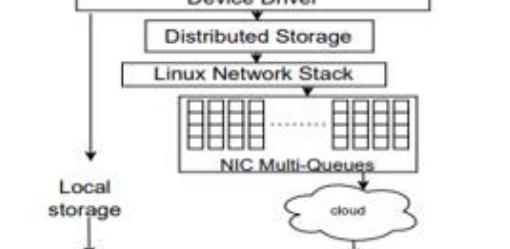
Four Main APIs in Storage



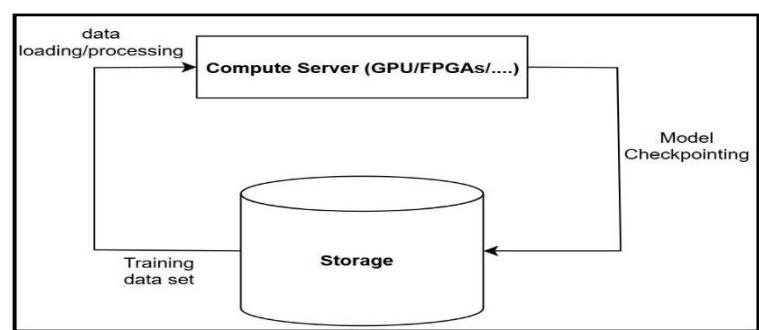
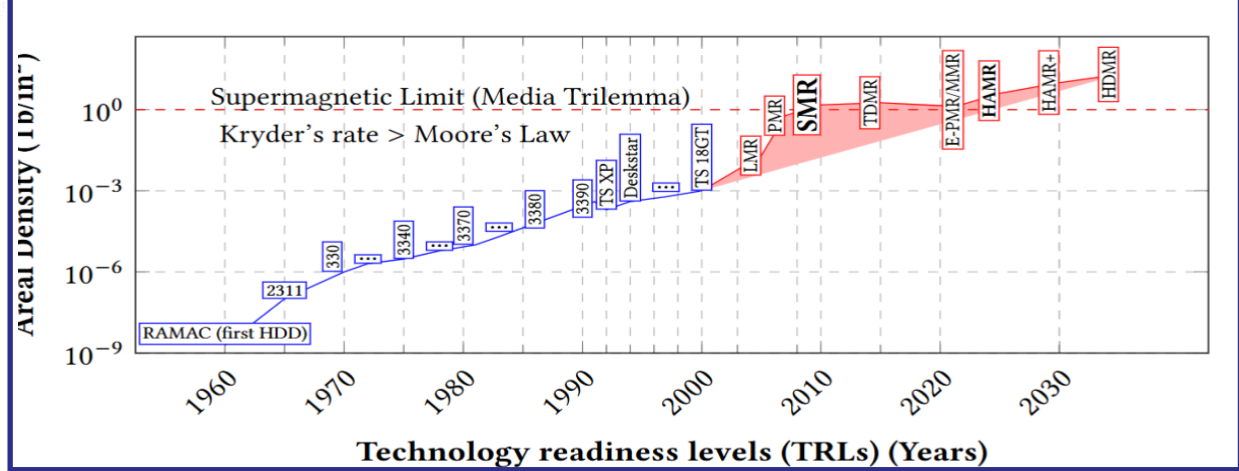
kernel



Block I/O kernel



hardware



I/O stack , AI training I/O model, disks
as of now in 2024



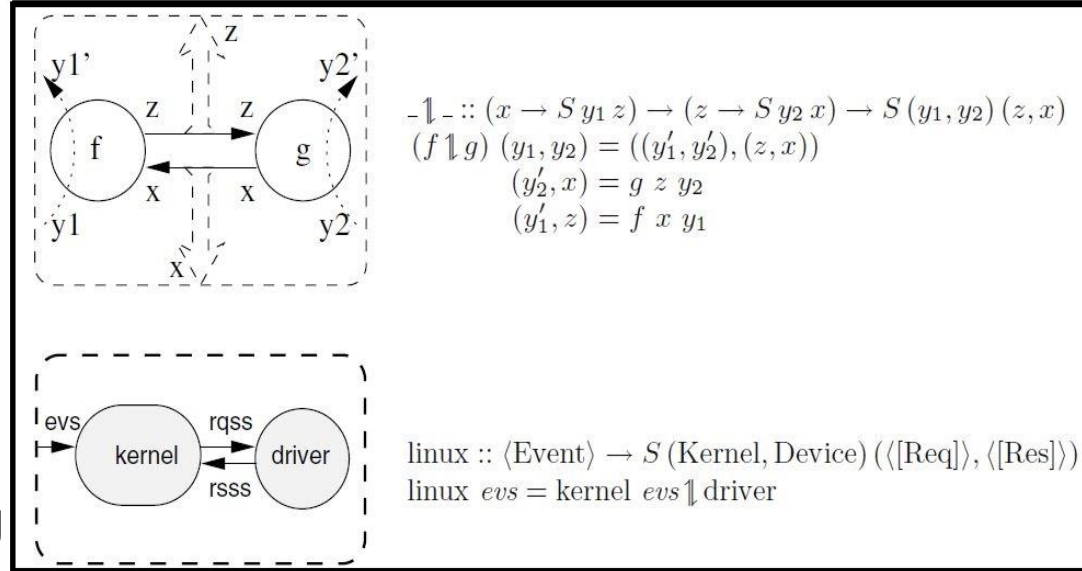
HAMR disk at Seagate booth in SC 2024

A Formal Model of Block Device: side-effecting communicating parallelism construct

- Two distinct sets

- kernel (y_1)
- Disk (y_2)

- Linux OS expressed as a **side-effecting** communicating parallel composite of **kernel core** (OS) and **driver**




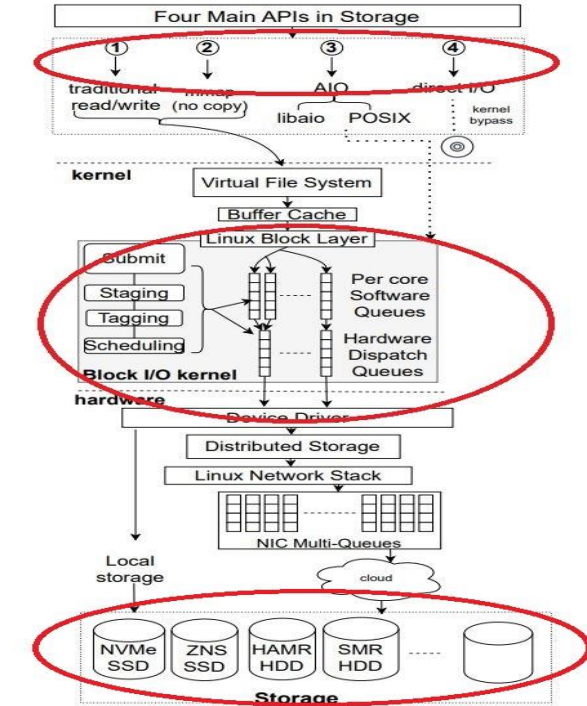
ref: Peter T. Breuer

Research Problem: I/O slow for AI Workload

Ripple down effect of monolithic block I/O model in Linux:

- **Problem 1:** Programming APIs (first circle)
 - Traditional read/write
- **Problem 2:** Performance (second circle)
 - Approx. **60%-90%** of total execution time in kernel
- **Problem 3:** Drivers mismatch with I/O layer (third circle)
 - NVMe, SMR, ZNS, HAMR, MAMR, TDMR.....

Move the needle: FPGA  storage frameworks still operate with this I/O model. Need to address all **3** problems.



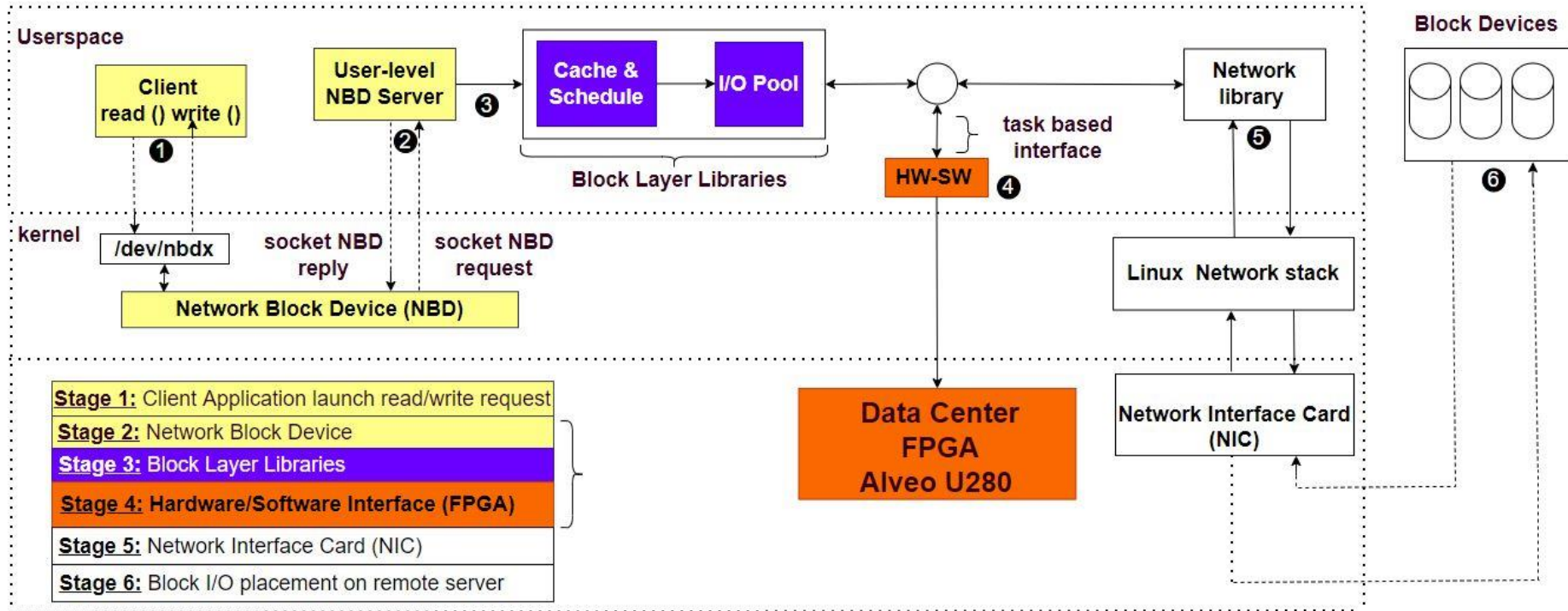
FPGA based DeLiBA-K Framework addressing 3 performance problems

DeLiBA Framework (Development of Linux Block I/O Accelerators)

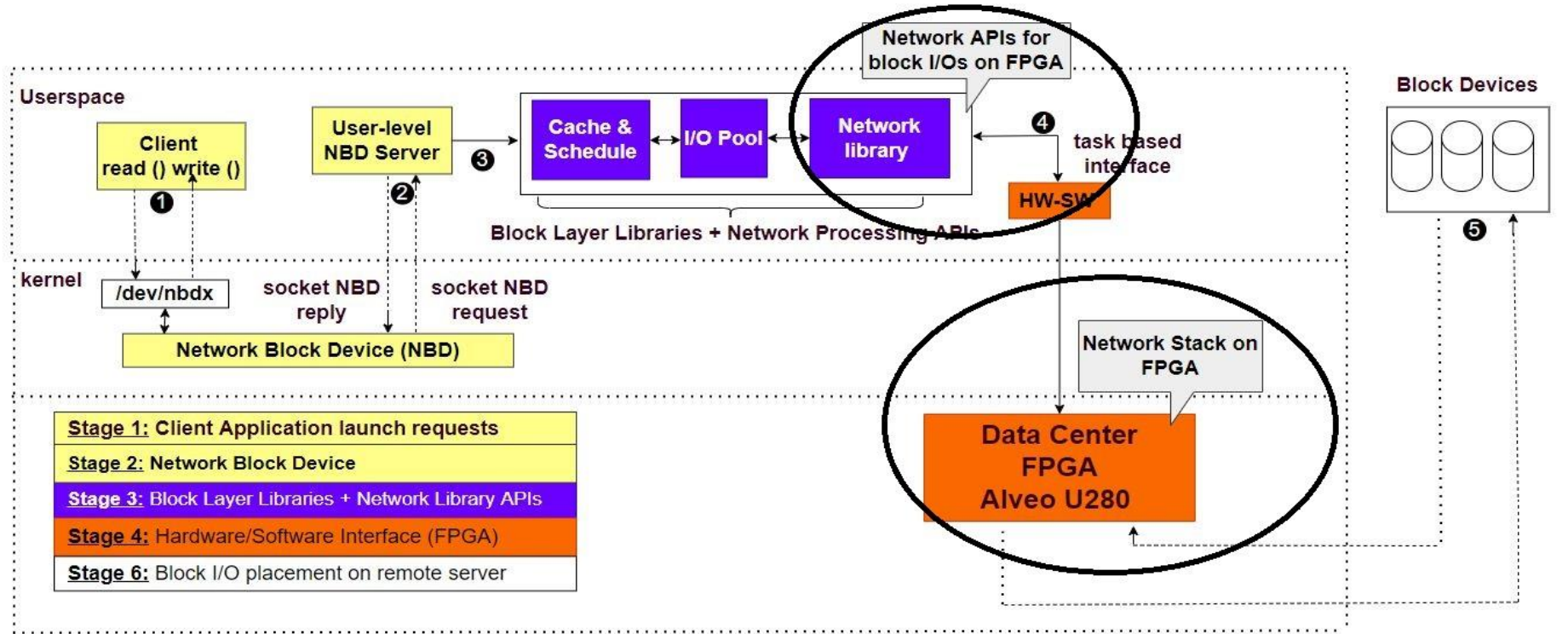
DeLiBA framework addressed **three** research problems incrementally by deploying data center 16nm FPGAs to accelerate I/O in distributed storage since year 2022:

- **DeLiBA-1 (2022):** userspace + block I/O on FPGA
Result: partially improved programming model (**Problem 1**)
- **DeLiBA-2 (2023):** userspace + block I/O & network I/O on FPGA
Result: partially improved performance (**Problem 2**)
- **DeLiBA-K (2024):** kernel + improved and extended block I/O & network I/O on FPGA
Result: addressed all three problems (**Problem 1 + Problem 2 + Problem 3**) and deployed in the research lab of industrial partner

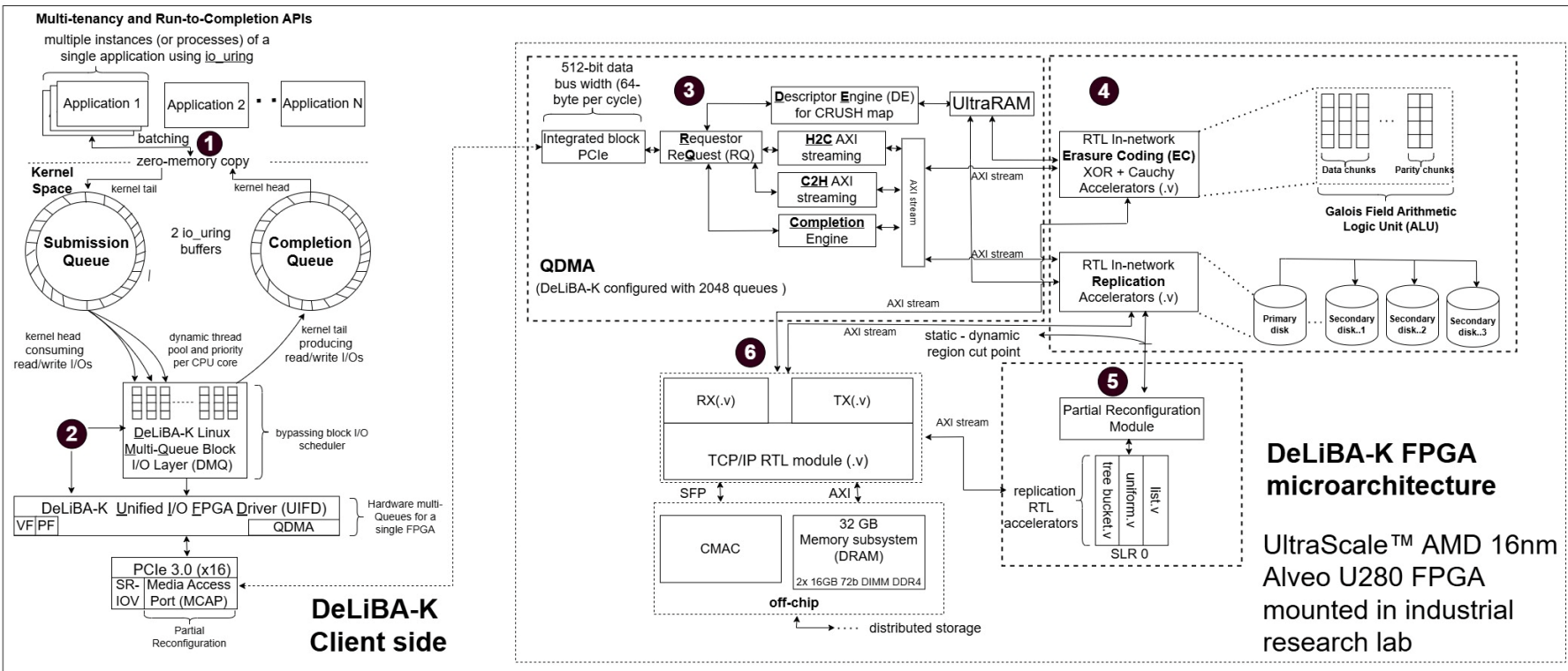
DeLiBA-1 (2022): userspace + block I/O on FPGA



DeLiBA-2 (2023): + network I/O on FPGA



DeLiBA-K: kernel + improved block I/O and network I/O on Alveo U280 FPGA



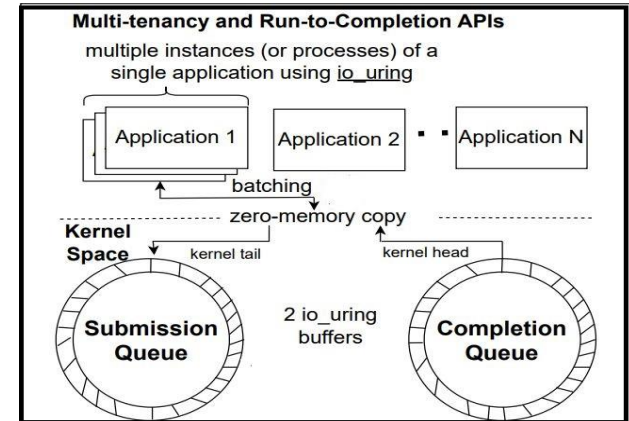
DeLiBA-K Framework three main architectural components

Next we will discuss three main architectural components of DeLiBA-K

- **First**: io_uring
- **Second**: new Linux kernel libraries and FPGA driver
- **Third**: FPGA stack with in-network storage (replication and erasure coding accelerators)

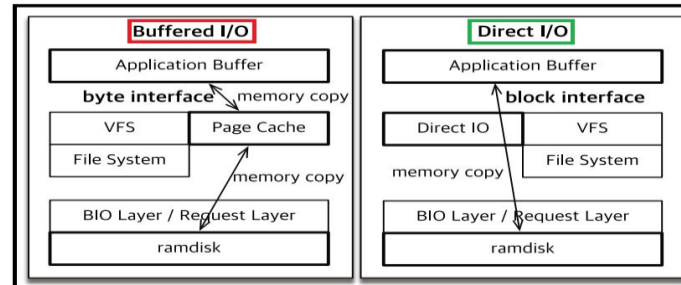
DeLiBA-K: io_uring

- DeLiBA-K implements the new AIO interface of mainline Linux kernel **io_uring** (first in version 5.1) I/O library.
- **2 ring buffers:** SQ and CQ
- **2 ring pointers per ring:** *tail* and *head*



The only AIO in Linux is **libaio**, but.....

- **works** in DIRECT I/O
- **DOES NOT** work in Buffered I/O



```
1 #include <liburing.h>
2
3 struct io_uring_sqe *sqe;
4 struct io_uring_cqe cqe;
5 struct io_uring ring;
6
7 io_uring_queue_init(8, &ring, 0);
8
9 /* get request slot, prepare request */
10 sqe = io_uring_get_sqe(&ring);
11 io_uring_prep_read(sqe, fd, buf, sizeof(buf), offset);
12
13 /* submit request(s) to the kernel */
14 io_uring_submit(&ring);
15
16 /* wait for a completion */
17 io_uring_wait_cqe(&ring, &cqe);
18 if (cqe->res < 0)
19     printf("Read error: %s\n", strerror(-cqe->res));
20 else
21     printf("Read %d from file\n", cqe->res);
22
23 /* mark cqe as seen, increments CQ ring head */
24 io_uring_cqe_seen(&ring, cqe);
```

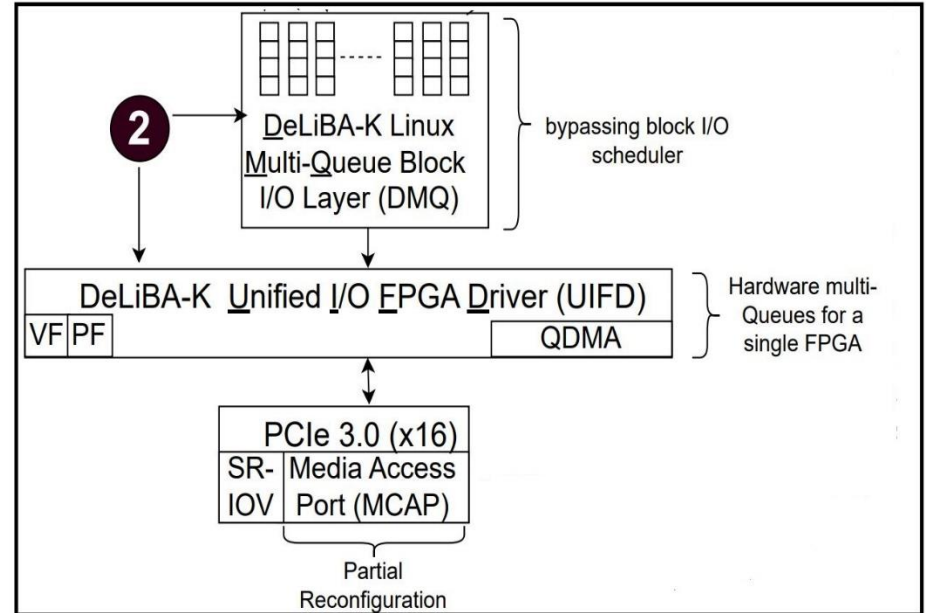
DeLiBA-K MQ (kernel library) and Unified Block I/O FPGA Driver

- **DeLiBA-K MQ:**

- changes in mainline Linux kernel
- bypass block I/O scheduler
- use MQ-block I/O layer

- **DeLiBA-K UIFD:**

- ZNS/SMR (NVMe zoned namespace and Shingled Magnetic Recording drivers)
- SR-IOV
- Ceph-RBD



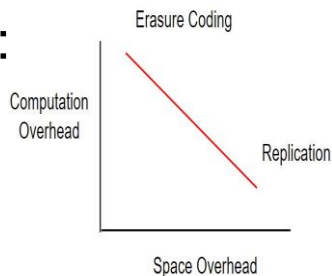
DeLiBA-K FPGA stack: Ceph I/O FPGA Accelerator

What is **Ceph**:

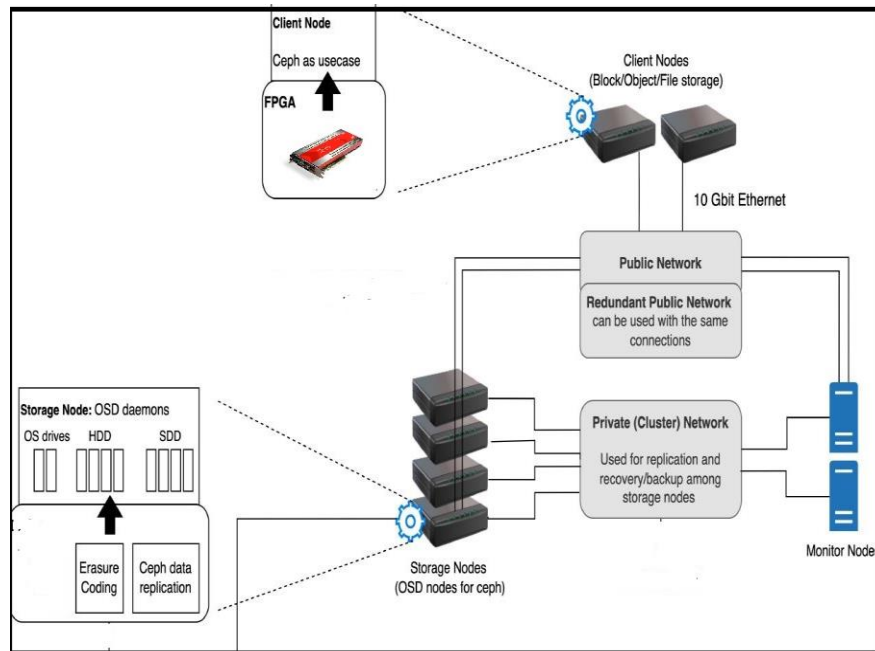
It's a software-defined distributed storage

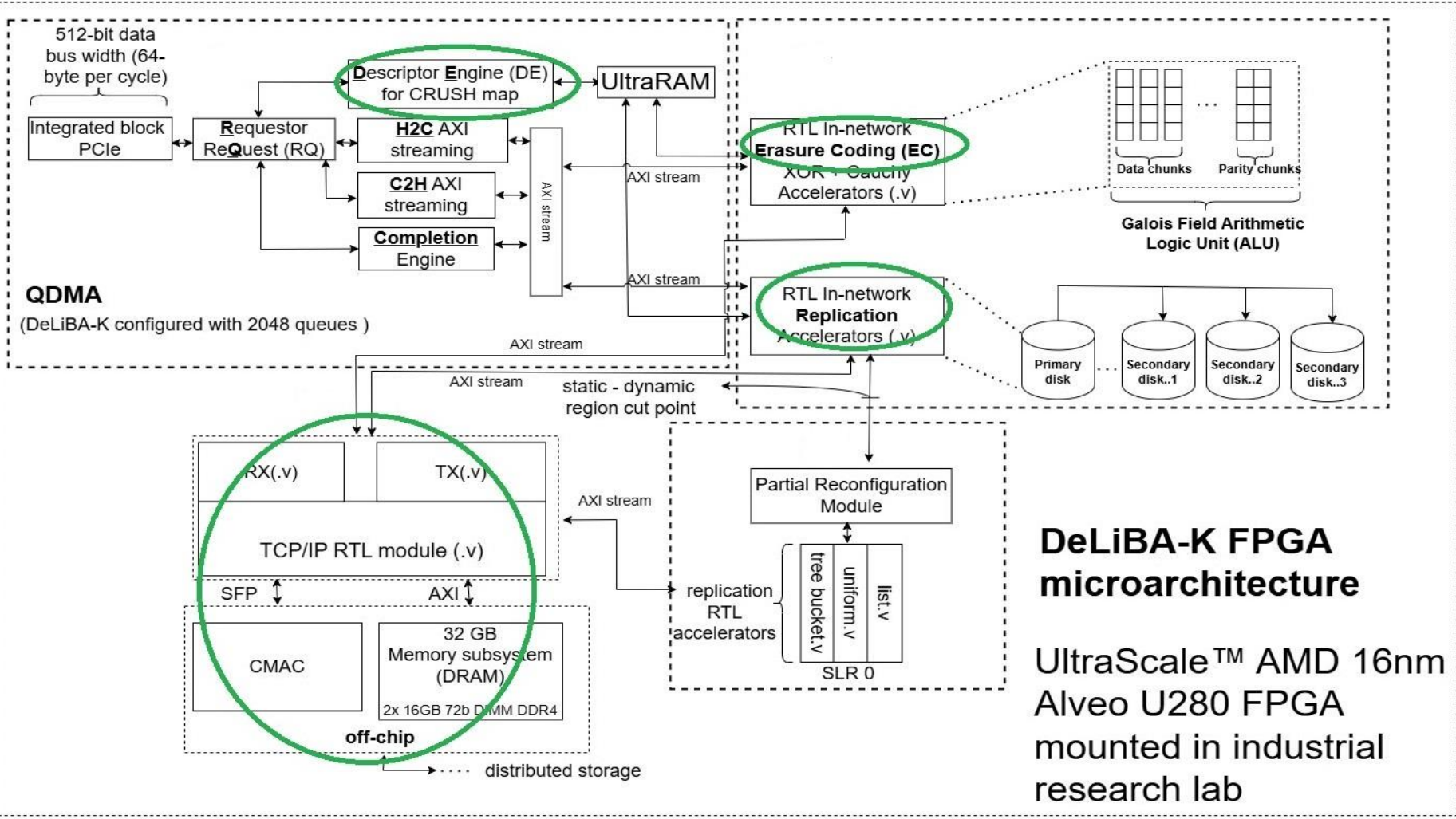
Testbed: →

Ceph's 24/7 cluster in our chair with two modes of operation:



- **Replication**
- **Erasure Coding**





DeLiBA-K FPGA microarchitecture

UltraScale™ AMD 16nm
Alveo U280 FPGA
mounted in industrial
research lab

In-network Erasure Coding (EC) FPGA Accelerator (RTL in Verilog)

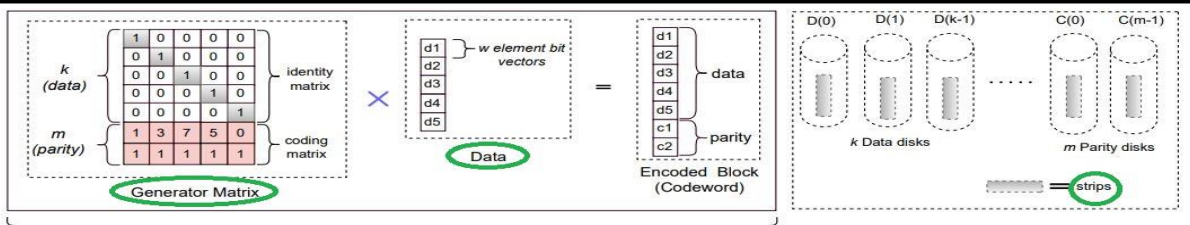
EC RTL accelerator (Galois Field ALU unit)

```

always @(posedge clk or posedge rst) begin
  if (rst) begin
    parity <= 8'b0;
  end else begin
    // XOR-based parity calculation
    parity <= data_in ^ 8'hff; // XOR with a fixed value
  end
end
endmodule
    
```

0	1	3	4	5	6	7	8
1
2
3
4
5
6
7

Addition and Multiplication table for GF(2³)



profiling workload: RS Encoder

Erasure Coding encoder using Reed-Solomon for $k=5$, $m=2$ and $GF(2^3)$

In-network Replication FPGA Accelerator (RTL in Verilog)

Replication RTL accelerator

```
always @(*) begin
    hash_value = hash_seed ^ object_id;
end

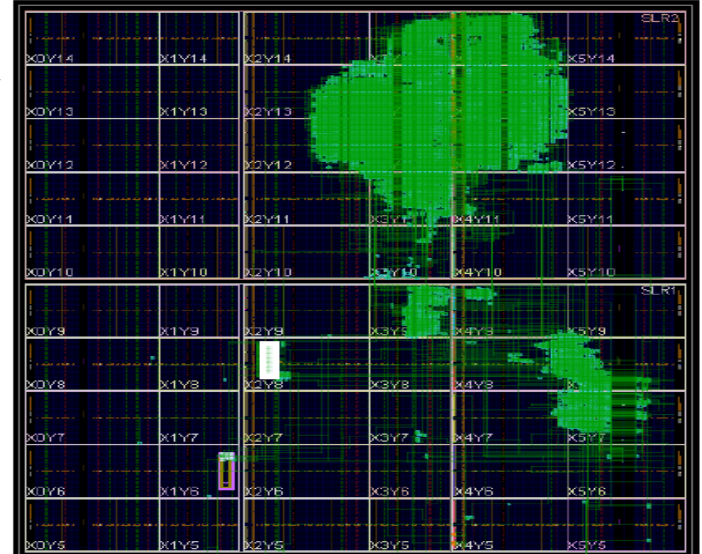
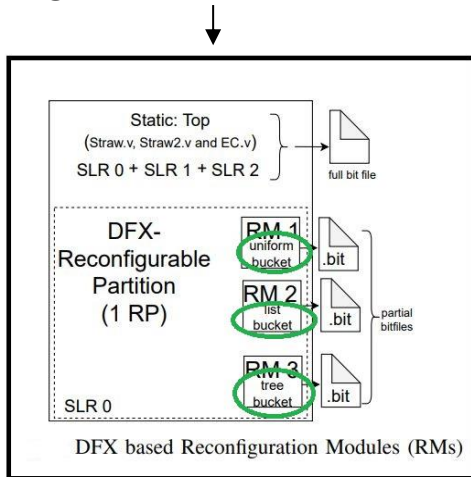
// Bucket selection logic
always @(*) begin
    if (num_buckets > 0)
        bucket_index = hash_value % num_buckets;
    else
        bucket_index = 0; // Default bucket if none exist
end

assign selected_bucket = bucket_index;
endmodule
```

Replication Kernels	Profiling SW Execution Time (Ceph-kernel)	Overall contribution to runtime	Vivado 2024 RTL Cycles (min-max)	Vivado 2024 Latency (min-max)	HW Execution on FPGA	SLOCs (C) SW Ceph-kernel	SLOCs (Verilog) HW Ceph-kernel
Straw Bucket	55 μ s	80 %	105-105	0.345 μ s - 0.355 μ s	49 μ s	256	880
Straw2 Bucket	48 μ s	80 %	155-155	0.315 μ s - 0.315 μ s	51 μ s	256	806
List bucket	35 μ s	80 %	40-40	0.161 μ s - 0.161 μ s	56 μ s	197	770
Tree Bucket	22 μ s	85 %	130-130	0.115 μ s - 0.115 μ s	31 μ s	241	780
Uniform Bucket	9 μ s	72 %	40-50	0.180 μ s - 0.180 μ s	19 μ s	237	745

Dynamic Function Exchange (DFX) ~ Partial Reconfiguration

- DeLiBA-K uses DFX through the Media Configuration Access Port (MCAP)
- 1 Reconfiguration partition and 3 Reconfiguration modules on FPGA



Hardware Evaluation on 16nm Alveo U280 FPGA in R&D lab of industrial partner



Evaluation on Hardware

Following Hardware setup:

- AMD EPYC Rome 7302P
16-core CPU with **264GB** of memory,
attached by **10 Gb/s** Ethernet to the Ceph
server.
- Xilinx Alveo U280 FPGA card attached to the client
host by **PCIe Gen3 x16** and uses a system clock
of **250 MHz**



Xilinx Alveo U280 FPGA
Card at ESA Group

Evaluation Methodology and Metrics

Methodology:

- DeLiBA-K Replication vs previous DeLiba frameworks in replication mode
- DeLiBA-K Erasure Coding vs previous DeLiba frameworks in EC mode

Metrics:

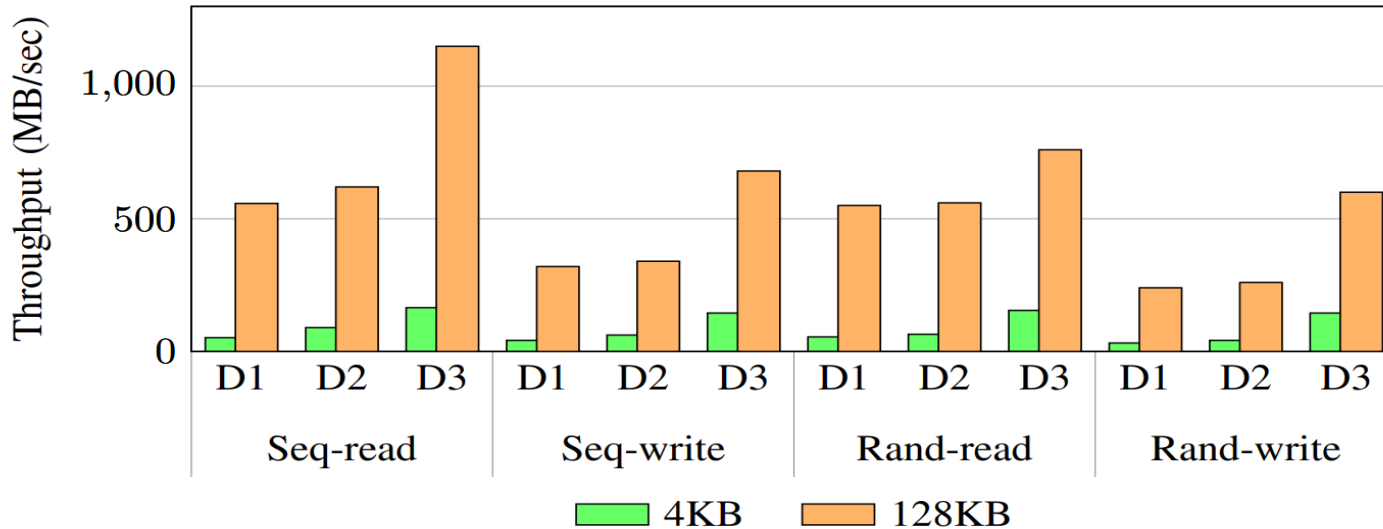
- *Throughput, Latency, IOPS, FPGA Power and Resource Utilization*

Tools and benchmark suites:

- Tools: fio, Vitis, Vivado report power, Vivado Power Analysis, AMD xbutil
- Benchmark: OLAP, and some in-house benchmark suites from our industrial partner

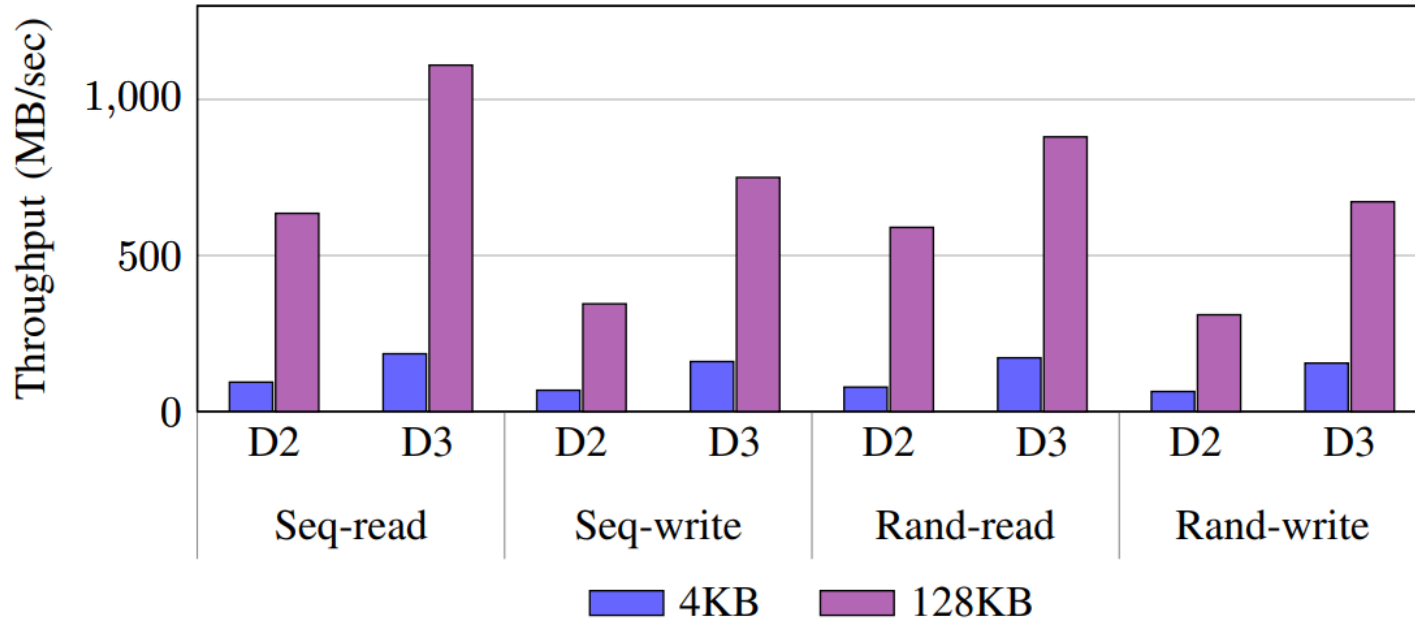
Final Hardware FPGA Evaluation in Ceph replication mode – Throughput

Here: **D1** (DeLiBA-1), **D2** (DeLiBA-2), and **D3** (DeLiBA-3)



Final Hardware FPGA Evaluation in Ceph Erasure Coding mode – Throughput

Here: **D1** (DeLiBA-1), **D2** (DeLiBA-2), and **D3** (DeLiBA-3)



Final Hardware FPGA based evaluation

Replication and Erasure Coding – I/O Latency

I/O REQUEST LATENCY IN DELiBA-K (HARDWARE) COMPARED WITH PREVIOUS FRAMEWORKS DELiBA-1 (D1) AND DELiBA-2 (D2)

Hardware (Replication) (4 kB)	Latency [μ s]			
	<i>seq-read</i>	<i>seq-write</i>	<i>rand-read</i>	<i>rand-write</i>
DeLiBA-1	65	95	130	98
DeLiBA-2	55	75	85	82
DeLiBA-K	40	52	64	68

Hardware (Erasure Coding) (4 kB)	Latency [μ s]			
	<i>seq-read</i>	<i>seq-write</i>	<i>rand-read</i>	<i>rand-write</i>
DeLiBA-2	48	70	82	75
DeLiBA-K	38	47	59	60

Evaluation Hardware – FPGA Resource Utilization on 16nm FPGA Alveo

RTL Kernel + RTL TCP/IP + CMAC + QDMA	CLB LUTs		CLB Registers		Block RAM (BRAM)		UltraRAM (URAM)		DSPs
	Count	% Usage	Count	% Usage	Count	% Usage	Count	% Usage	Count
Straw Bucket	78,555	6.2 %	224K	8.59 %	190	9.42 %	26	2.71 %	0
Straw2 Bucket	82,334	6.31%	313K	12.01 %	165	8.18 %	35	3.65 %	0
Reed-Solomon Encoder	92,355	7.08 %	582K	22.32 %	215	10.66 %	52	5.42 %	0
Partial Reconfiguration Modules (RM) in SLR 0 of U280	CLB LUTs		CLB Registers		Block RAM (BRAM)		UltraRAM (URAM)		DSPs
	Count	% Usage	Count	% Usage	Count	% Usage	Count	% Usage	Count
RM 1 List Bucket (Replication RTL Accelerator)	52,335	14.74 %	92,456	12.75 %	85	17.35 %	22	6.88 %	0
RM 2 Tree (Replication RTL Accelerator)	56,555	15.93 %	97,523	13.45 %	82	16.73 %	26	8.13 %	0
RM 3 Uniform (Replication RTL Accelerator)	62,456	17.59 %	112K	15.45 %	78	15.92 %	29	8.7%	0

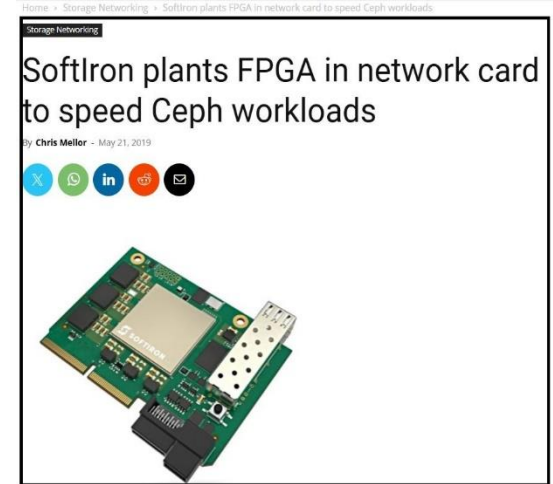
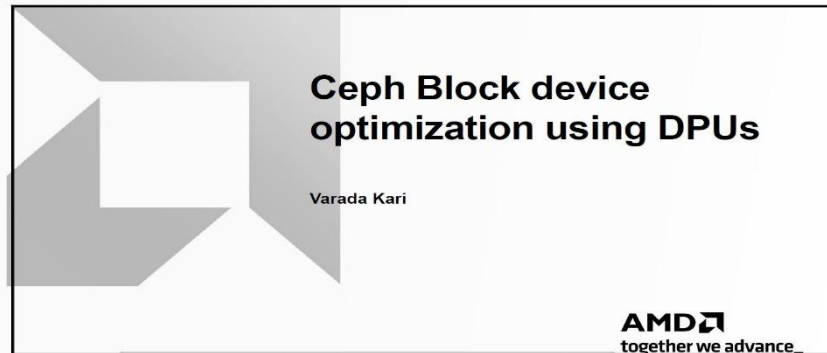
Conclusion: Final Speedups on FPGA (throughput and latency)

- Performance Gain (Speedups) throughput random-writes:
 - Random I/O: **3.45x** (4KB) & **2.5x** (8KB)
 - Sequential I/O: **2.38x** (64KB) & **2x** (128KB)
- Performance gain in terms of low-latency:
 - Random read I/O: **25%** decrease
 - Random write I/O: **17%** decrease

Contemporary Ceph I/O Accelerators vs DeLiBA-K

- **Ceph Accepherator:**
 - no publicly available benchmark numbers for Ceph Accepherator.

- **AMD Ceph DPU:** no benchmark numbers available



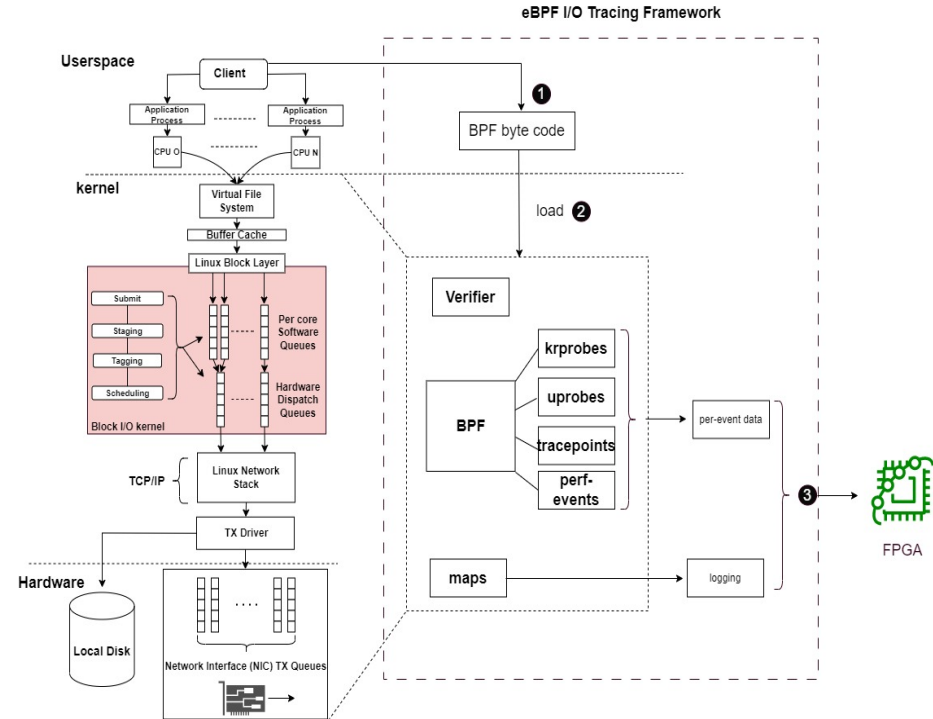
Conclusion and Future Work

Conclusion:

- DeLiBA-K is being used by our industrial partner as of now.
- First work to implement io_uring in a FPGA framework.

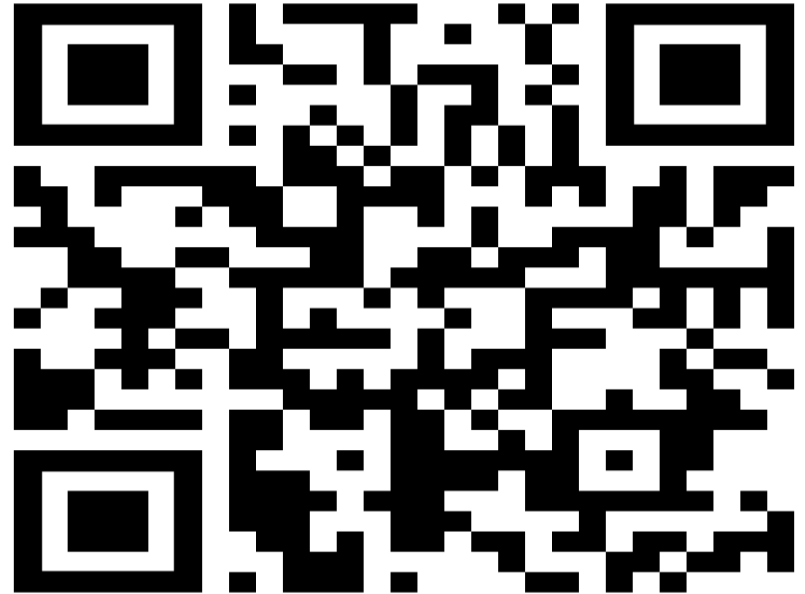
Mainly following two future work:

- We will explain in detail the profiling and tracing framework developed during DeLiBA-K
- Erasure Coding on Versal AI Engines



DeLiBA is open-source

- DeLiBA is available at our ESA github:
<https://github.com/esa-tu-darmstadt/deliba>



QR code for our DeLiBA repo

THANKS FOR YOUR ATTENTION!