# vFPIO

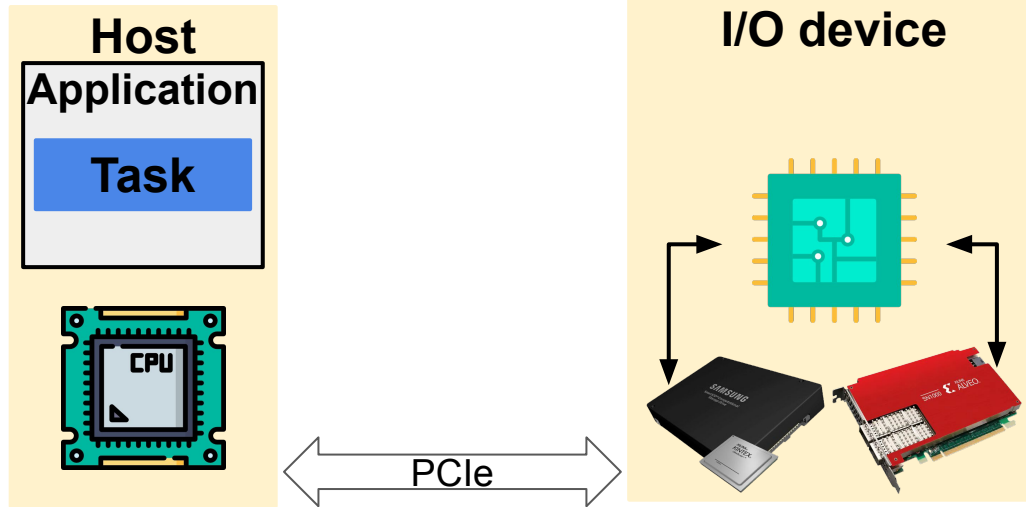## A **V**irtual I/O Abstraction for **FP**GA-accelerated **I/O** Devices

Jiyang Chen, **Harshavardhan Unnibhavi**, Atsushi Koshiba,
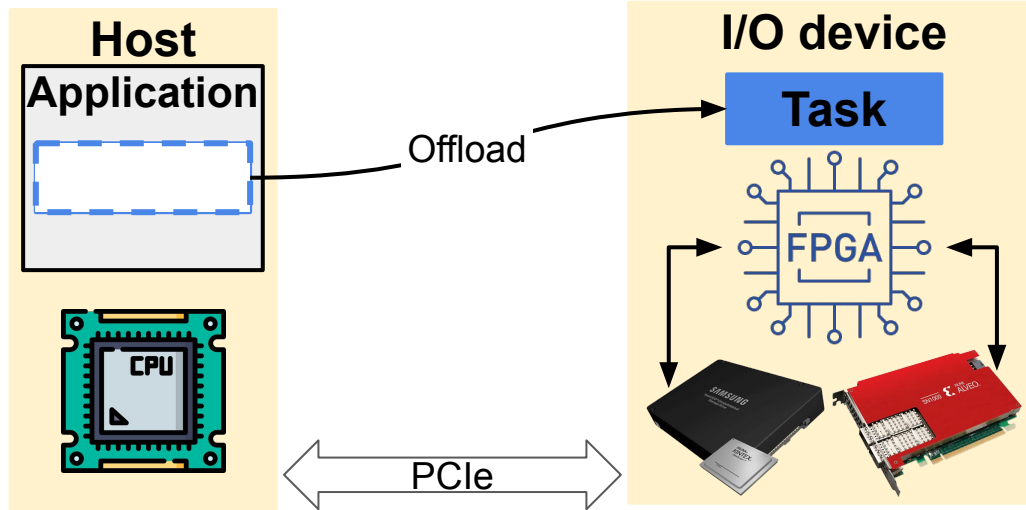Pramod Bhatotia

USENIX ATC 2024

# Context: FPGA-based I/O acceleration

**I/O-acceleration** is popular in the cloud (e.g. Azure Boost, Google Titanium)

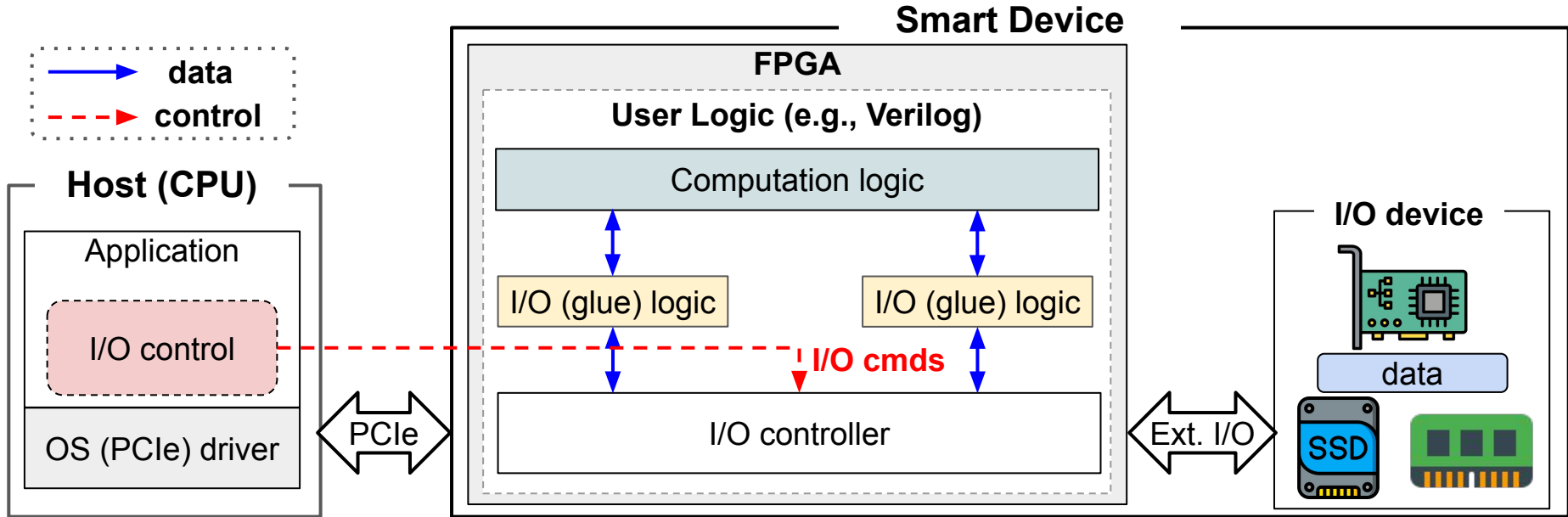# Context: FPGA-based I/O acceleration

**I/O-acceleration** is popular in the cloud (e.g. Azure Boost, Google Titanium)



FPGAs enable high-performant I/O processing in a flexible manner

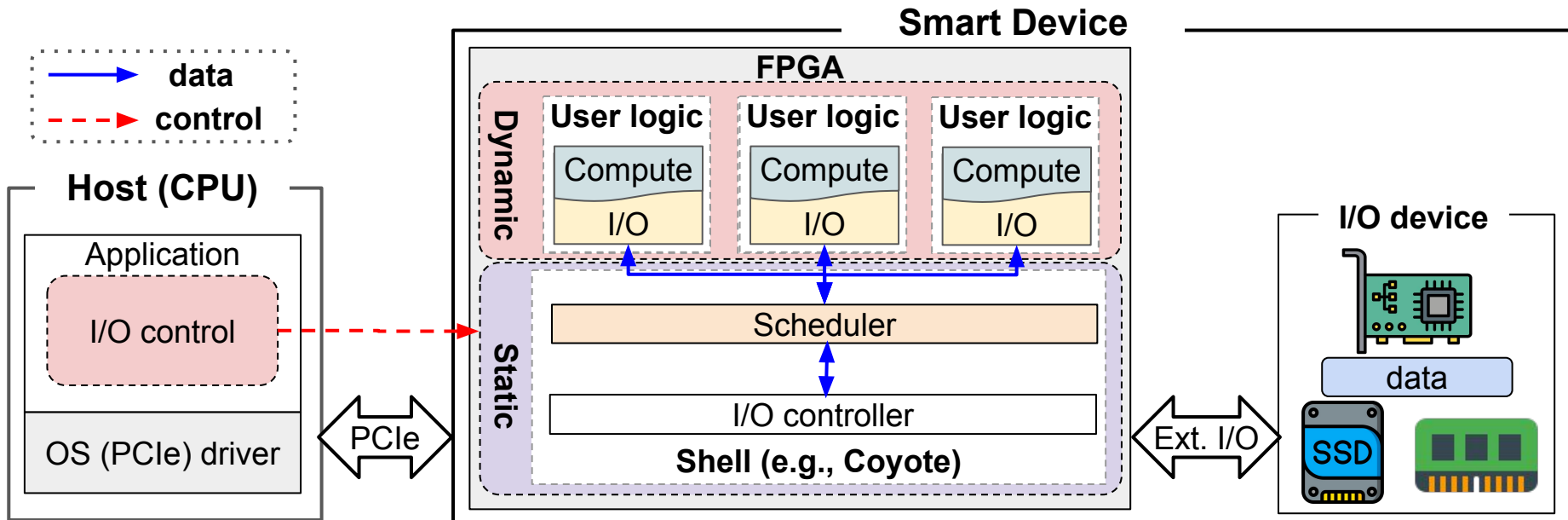# Background: Develop I/O accelerators on FPGA

FPGAs enable I/O acceleration in a flexible manner



**Programmability issues** and **no multi-tenancy**

# State-of-the-art: FPGA Shell

**Shells** (e.g., **Coyote** [OSDI'20]) enhance <u>programmability and isolation</u>



**Lack of I/O virtualization** introduces challenges in user logic design

# I/O challenges in FPGAs

- User logic that manages device I/O is hard to ***program***
  - Shells only expose low-level I/O device interfaces


- User logic is not ***portable*** across multiple I/O devices
  - Rewrite I/O logic for the target device even if computation logic is the same


- ***I/O performance*** is not ***isolated*** between user logics
  - Sharing I/O device between user logics degrades I/O throughput

# Problem statement

How to design an FPGA I/O acceleration framework that enhances **programmability, portability** and ensures **performance isolation** of user logics?
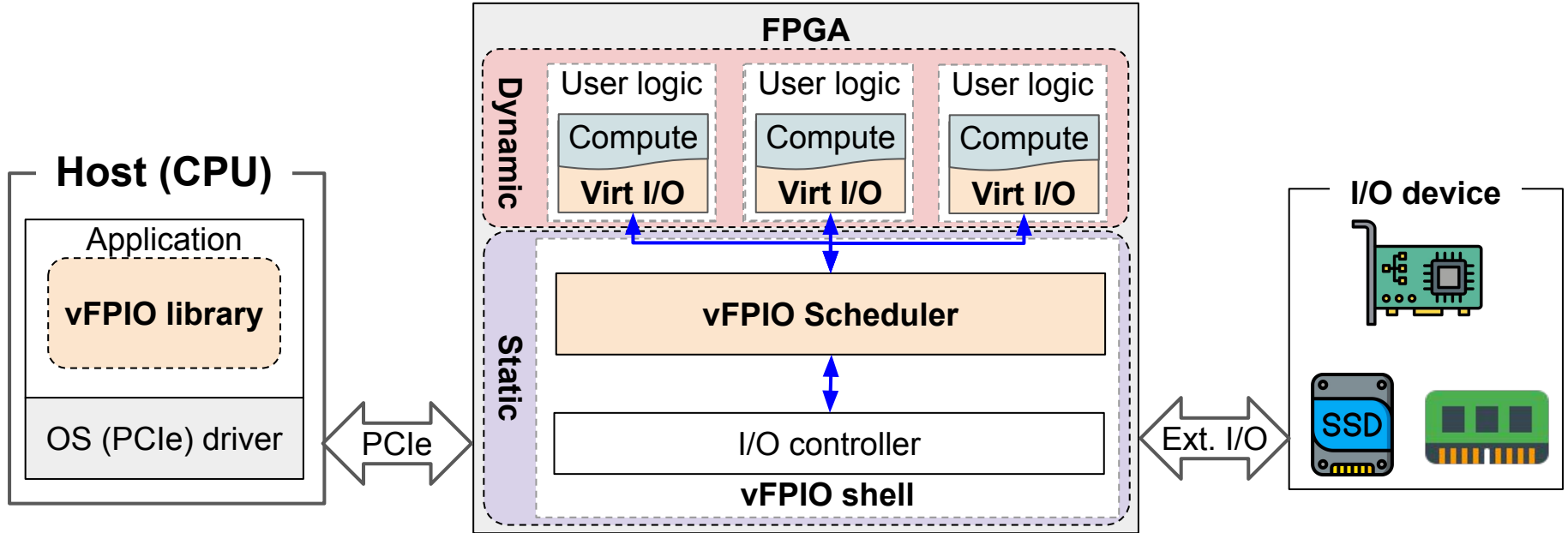
# Our proposal

**vFPIO**
A portable and easy-to-use FPGA I/O acceleration framework

**Design goals:**

- **Programmability** improvements for both host and FPGA user logic

- **Portability** across multiple I/O devices

- **Performance isolation** between multiple user logics

# vFPIO overview

# Outline

- ~~Motivation~~
- **Design and workflow**
- Evaluation

# Challenge #1: Programmability

- Shells directly expose device I/O interfaces to user logic
    - Low-level device interfaces accessible through I/O protocols (e.g. AXI)
    - Fixed number of I/O ports and I/O width


- Limitations of shell I/O interfaces
    - Programmers need to understand low-level I/O interfaces
    - User logics contain device-specific I/O logic

# Solution #1: Virtual I/O module

> **Virtual I/O module** to abstract *physical I/O ports into virtual I/O ports*

- Virtual I/O ports are device-agnostic
  - Connect user logic to any I/O device

- Dynamic I/O reconfiguration
  - I/O switching between virtual and physical ports

- Exposes control interface to the host
  - vFPIO APIs for cross-device portability



12
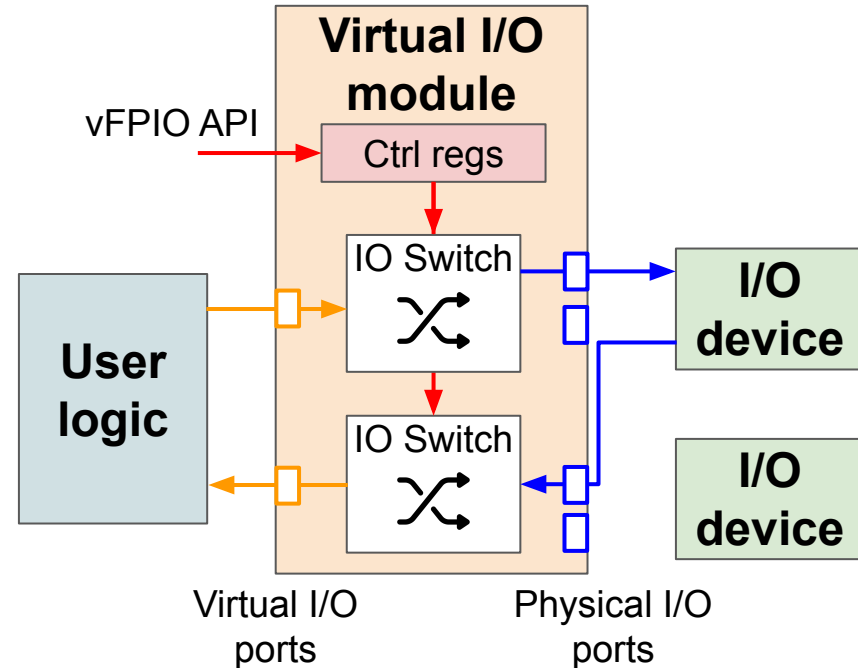
# Solution #1: Virtual I/O module

> **Virtual I/O module** to abstract *physical I/O ports into virtual I/O ports*
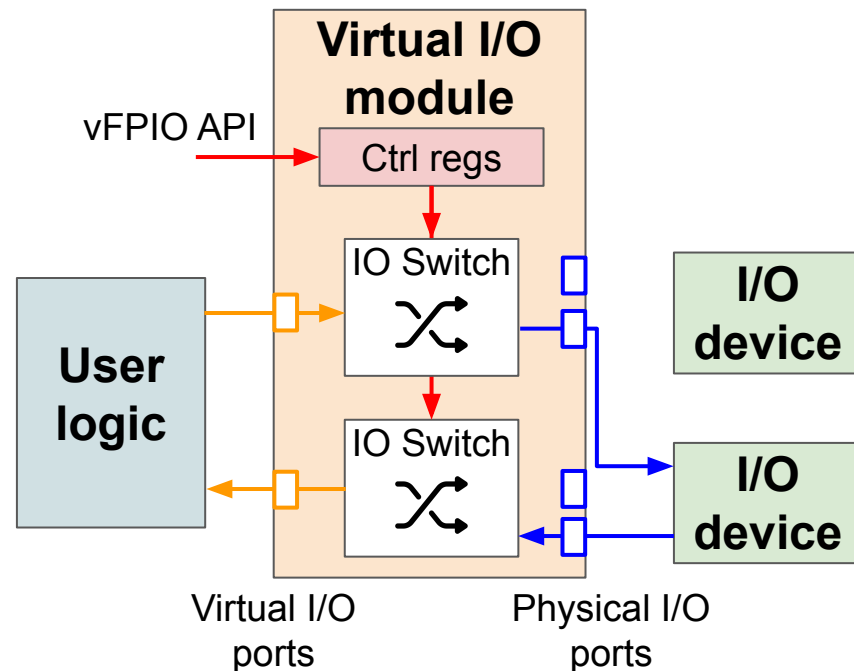
- Virtual I/O ports are device-agnostic
  - Connect user logic to any I/O device

- Dynamic I/O reconfiguration
  - I/O switching between virtual and physical ports

- Exposes control interface to the host
  - vFPIO APIs for cross-device portability

# Challenge #2: Portability

- User logics are forced to include low-level I/O interfaces
  - Tight coupling of computational and I/O logic
  - User logic developed for only one I/O device

- Portability issues across I/O devices
  - Same computational logic cannot be reused across I/O devices
  - Reprogram, compile and offload user logic ⟩ **Very expensive!**

# Solution #2: vFPIO library

> **vFPIO library** for *software-driven dynamic I/O reconfiguration*
> across user logic and I/O devices *on-demand*

- Host CPU-side software library
  - Allow host applications to configure I/O paths on the FPGA
  - I/O path reconfiguration without reconfiguring whole user logic

- POSIX-like APIs
  - I/O devices represented as files
  - Decouple computational from user logic

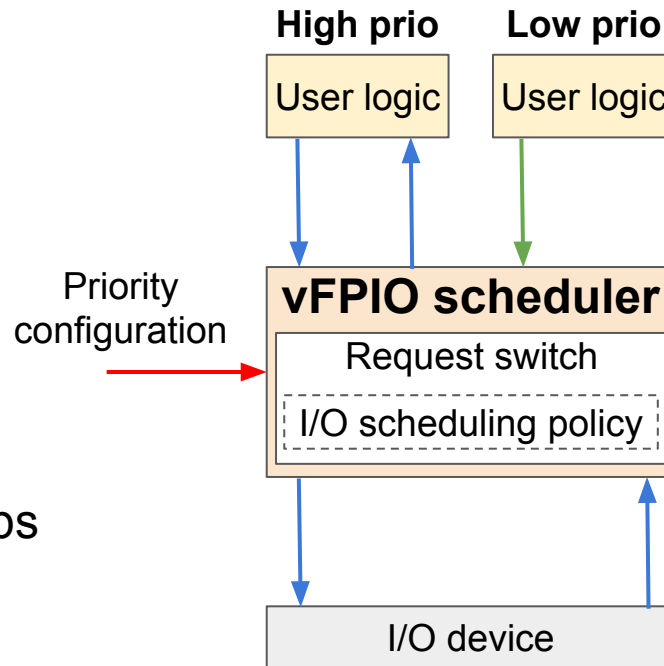| vFPIO API | Description |
|-----------|-------------|
| init() | Initialize vFPGA with bitstream |
| open() | Connect user logic to device |
| close() | Disconnect user logic from device |
| read() | Read data **from** device **to** user logic |
| write() | Write data **to** device **from** user logic |

# Challenge #3: Performance isolation

- Degrades I/O throughput when multiple user logics share an I/O device
  - FPGA shells divide I/O bandwidth equally

- Lacks priority based scheduling of I/O transactions
  - Execute critical applications first
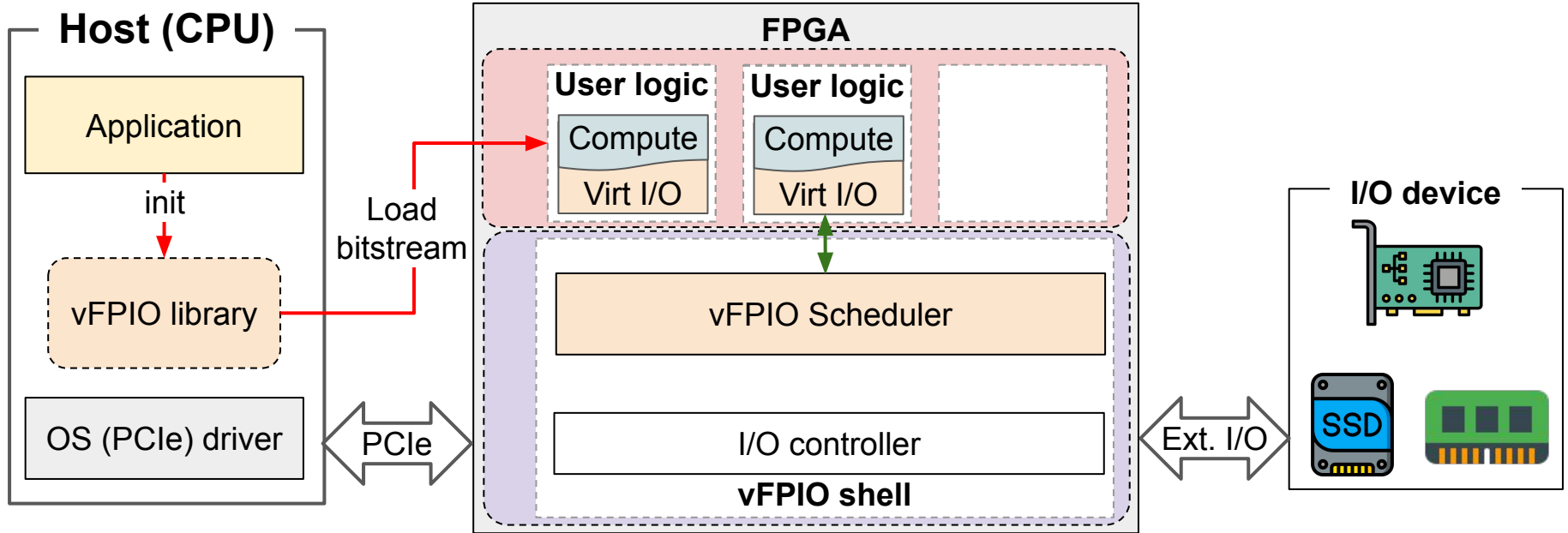  - Pause and resume low priority applications without errors

# Solution #3: vFPIO scheduler

> An **I/O scheduler** that supports *priority-based scheduling* in the vFPIO shell
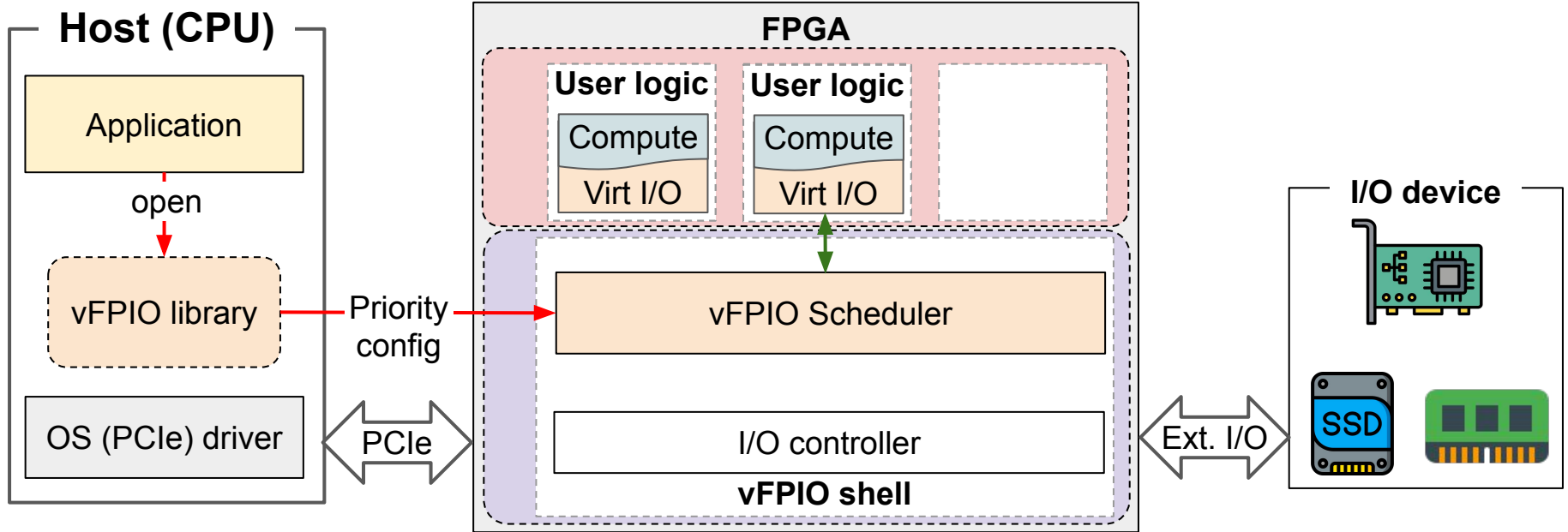
- Scheduler pair per I/O device
  - One each for read and write requests

- Multiple scheduling policies
  - Round-robin or priority-based

- Priority-based scheduling for critical apps
  - Pick I/O request with highest priority

**High prio**
User logic

**Low prio**
User logic

Priority configuration

**vFPIO scheduler**
Request switch
I/O scheduling policy

I/O device

# vFPIO workflow

**Host (CPU)**

Application

init

vFPIO library

OS (PCIe) driver

Load bitstream

**FPGA**

**User logic**

Compute

Virt I/O

**User logic**

Compute

Virt I/O

vFPIO Scheduler

I/O controller

**vFPIO shell**

PCIe

Ext. I/O

**I/O device**

SSD
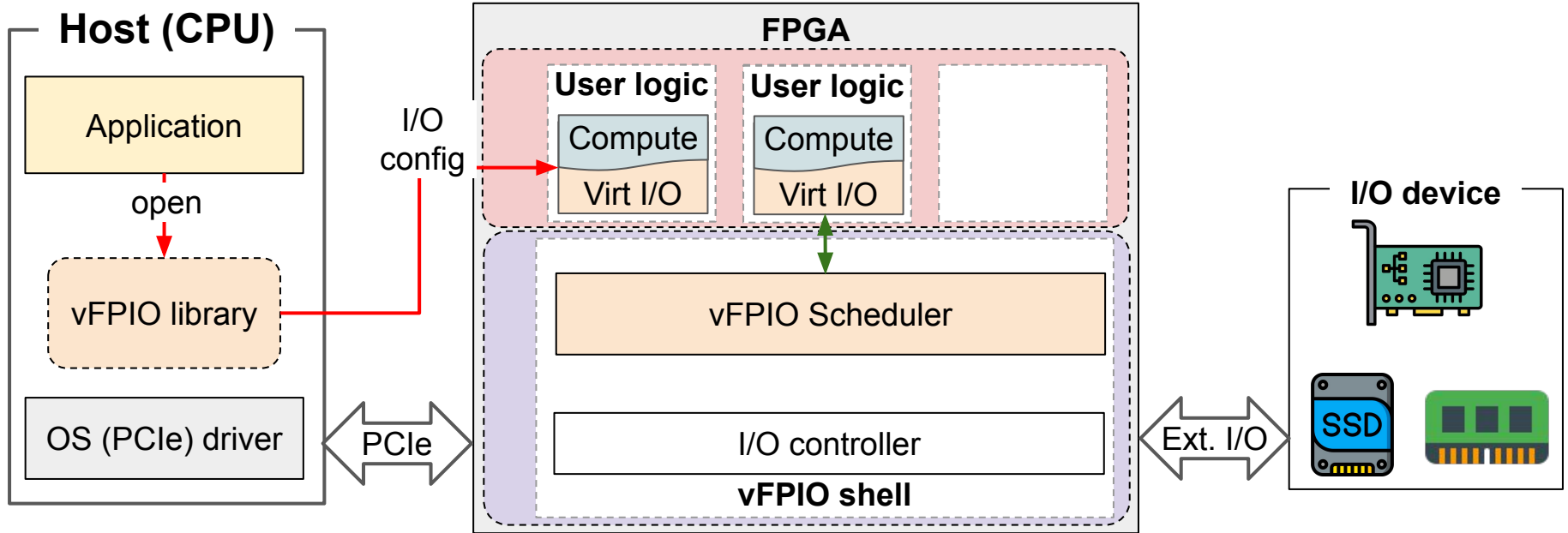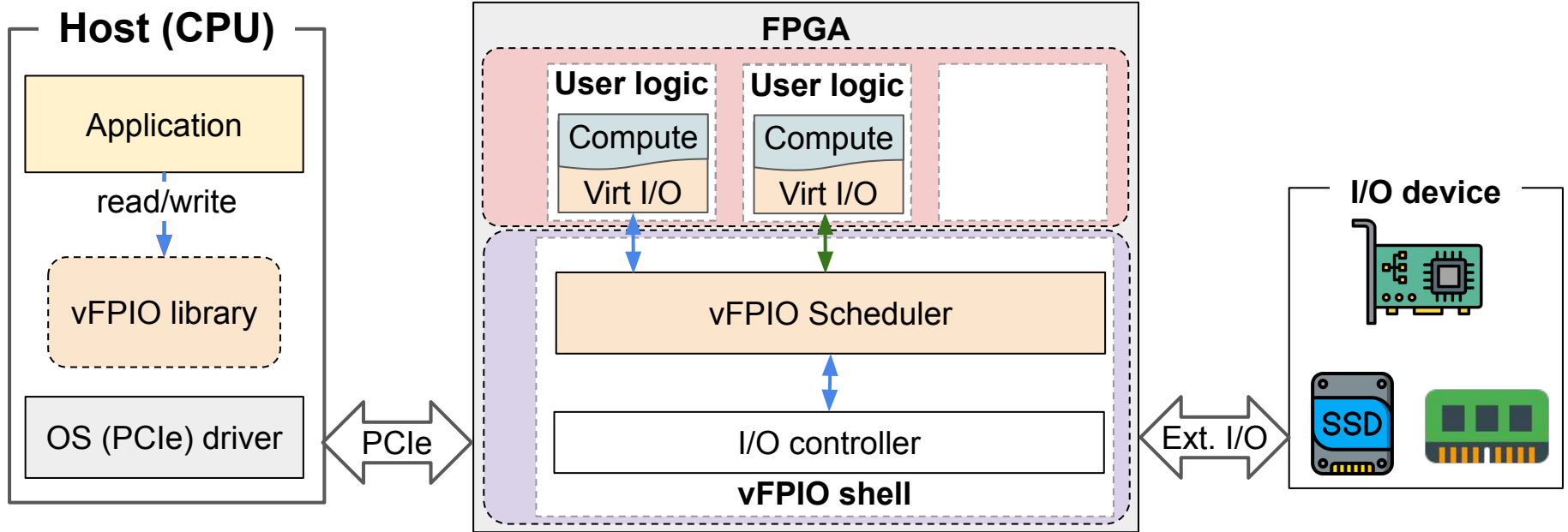
# vFPIO workflow

# vFPIO workflow

# vFPIO workflow

# Outline

- ~~Motivation~~
- ~~Design and workflow~~
- **Evaluation**

# Evaluation

- Questions
  - Performance overheads
  - I/O switching overheads
  - Performance isolation
  - Programmability benefits for both host and FPGA user logic ⎫
  - Resource consumption of vFPIO ⎭ Refer to the paper

# Evaluation

- **Experimental setup**
  - 2 x AMD(R) EPYC 7413 (24 cores)
  - 2 x Xilinx Alveo U280 (8GiB HBM)
  - 2 x QSFP28 (100GbE)

- **Baseline**
  - Coyote [OSDI'20]: Coyote shell without virtual I/O support

# Performance overheads

**vFPIO's throughput compared to Coyote**



Higher is better
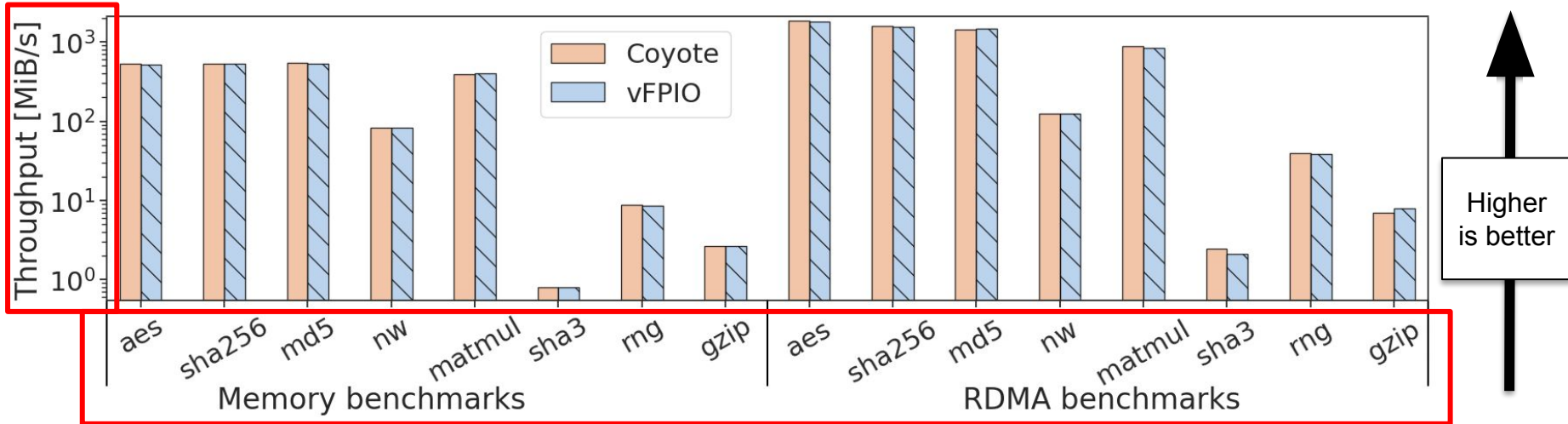
vFPIO's virtual I/O abstractions introduce low performance overheads
(**0.7% for memory and 1.1% for RDMA**)

# I/O switching overhead

**Overhead of switching I/O devices in vFPIO compared to Coyote**

| Application | Coyote [µs] | vFPIO [µs] |
|:---:|:---:|:---:|
| aes | 21K | 1.3 |
| sha256 | 20.2K | 1.3 |
| md5 | 18.8K | 1.3 |
| nw | 29.2K | 1.3 |
| matmul | 23.9K | 1.3 |

Fast I/O reconfiguration
**~20K faster**

vFPIO's virtual I/O abstractions enable **fast I/O reconfiguration** at **negligible performance overheads**

**vFPIO's performance isolation for multiple user logics**
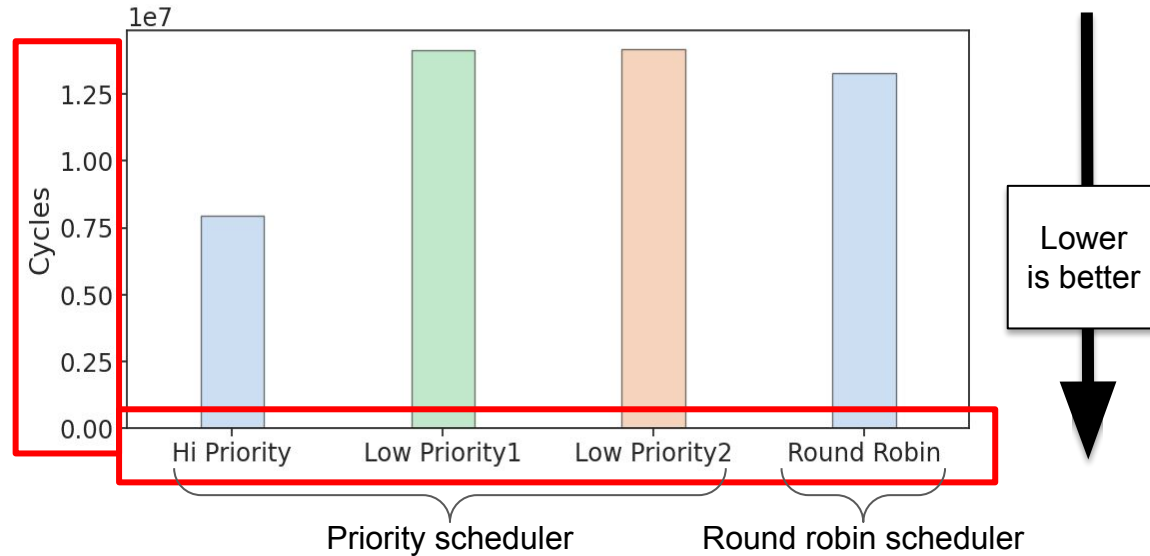


vFPIO's priority scheduler improves high-priority application's performance **by 1.7x** compared to a round-robin I/O scheduler

# Summary

For FPGA user logics on Smart I/O devices,
*How to ensure **their programmability, portability and performance isolation?***

## vFPIO: FPGA I/O acceleration framework

- **Virtual I/O module** improves *programmability* for user logic managing I/Os

- **vFPIO APIs** improves user-logic *portability* across different I/O devices

- **vFPIO scheduler** ensures *performance isolation* during I/O device access

## Try it out!
https://github.com/TUM-DSE/vFPIO

ARTIFACT EVALUATED · usenix ASSOCIATION · AVAILABLE

ARTIFACT EVALUATED · usenix ASSOCIATION · FUNCTIONAL

ARTIFACT EVALUATED · usenix ASSOCIATION · REPRODUCED