



# Coyote

## Abstractions for Modern Heterogeneous Hardware

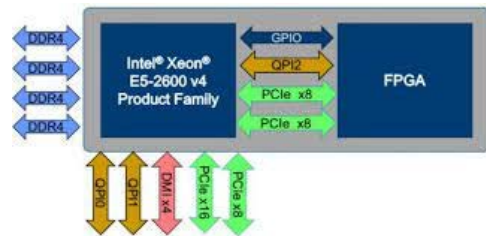
Dario Korolija, Timothy Roscoe, Gustavo Alonso

*Systems Group, Dept. of Computer Science, ETH Zurich*

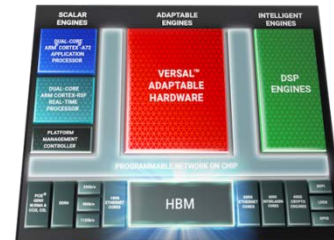


# Introduction...

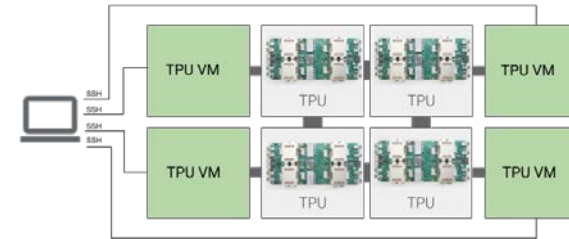
- ❖ Specialized hardware becoming a reality
  - Amazon, Microsoft, Google, Alibaba, Intel, AMD ...



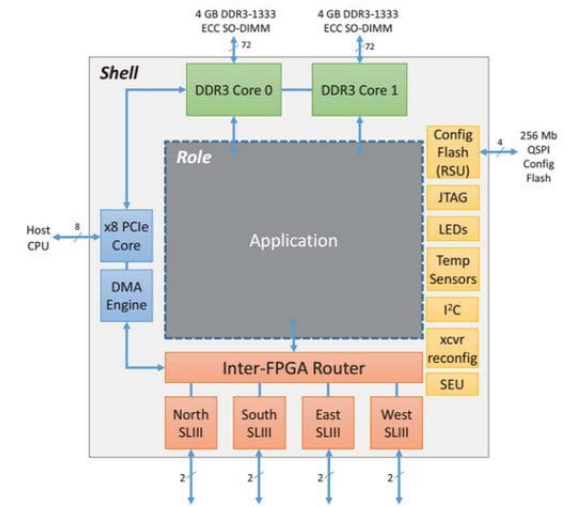
Intel HARP



AMD Versal

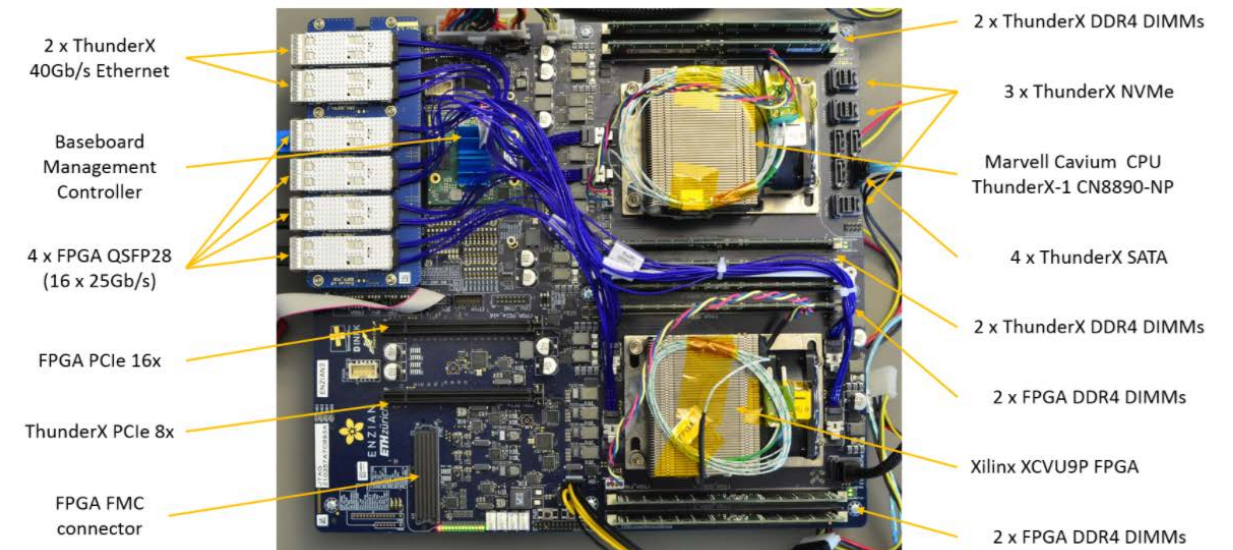


Google TPU



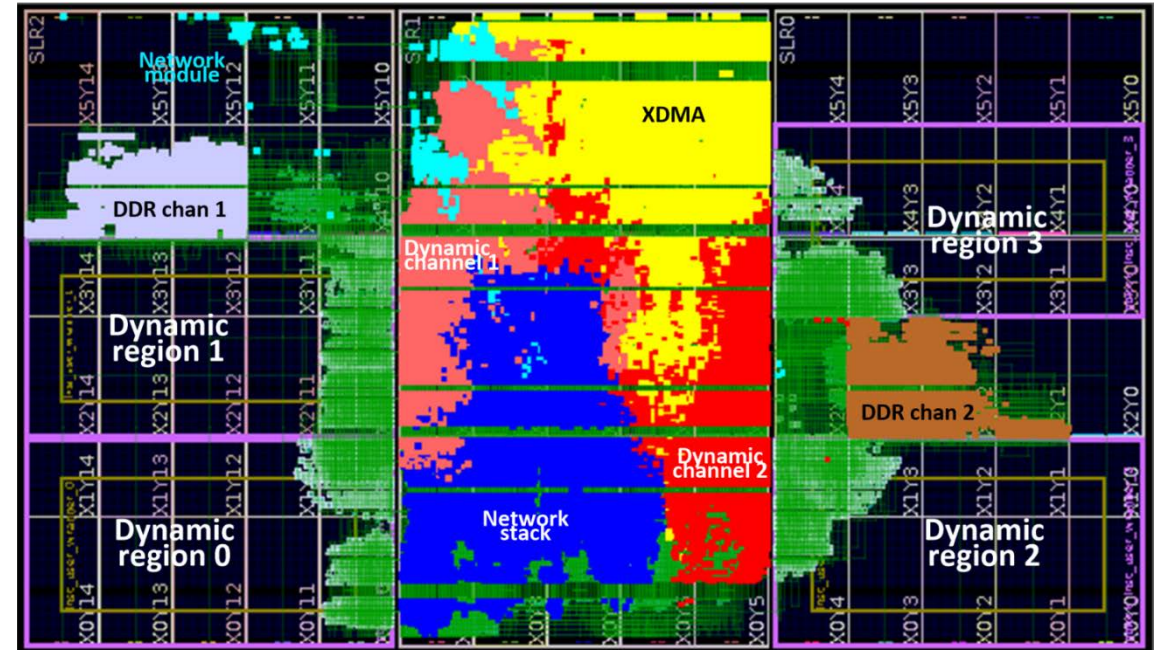
Microsoft Catapult

- ❖ One built within System group at ETH:



# Complex to program (interact with)

- ❖ I/O interaction ...
- ❖ Low level APIs
- ❖ No standard execution environment
- ❖ Lack of portability (HLS ...)
- ❖ Lack of traditional abstractions



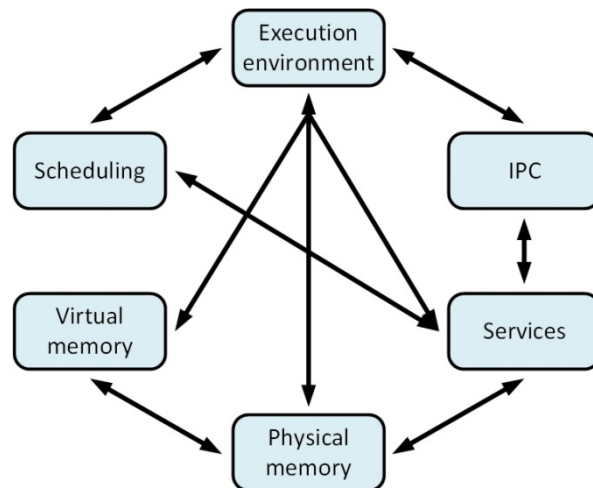
*FPGA layout*  
*PCIe, memory, storage, network...*

# Microkernel for FPGAs

## Hybrid computing system

- ❖ Plenty of research, focused on individual functionalities only
- ❖ **Coyote** provides a complete minimal core set of essential features on which further services can be used

### Interdependence



### A Hypervisor for Shared-Memory FPGA Platforms

Jiucheng Ma<sup>1</sup>, Gefei Zan<sup>1</sup>, Kevin Loughlin<sup>2</sup>, Xiaohu Cheng<sup>3</sup>, Yanqiang Lin<sup>1</sup>,  
Abel Mulgetta Eneyew<sup>1</sup>, Zhengwei Qi<sup>1</sup>, Baris Kasikci<sup>1</sup>,  
<sup>1</sup>University of Michigan, <sup>2</sup>Hong Kong University of Science and Technology,  
<sup>3</sup>Shanghai Jiao Tong University

#### Abstract

Cloud providers accelerate their FPGA offerings to meet support in cost-effective program use. Direct memory access (DMA) is a key feature in FPGA. This paper provides a minimal core set of essential features on which further services can be used.

### Sharing, Protection, and Compatibility for Reconfigurable Fabric with AMORPHOS

Ahmed Khawaja<sup>1</sup>, Joshua Landgraf<sup>1</sup>, Rohith Prakash<sup>1</sup>,  
Michael Wei<sup>1</sup>, Eric Schkafra<sup>2</sup>, Christopher J. Rossbach<sup>1</sup>,  
<sup>1</sup>The University of Texas at Austin, <sup>2</sup>VMware Research Group,  
<sup>3</sup>The University of Texas at Austin and VMware Research Group

#### Abstract

Cloud providers such as AWS began to support on-demand FPGA and hardware vendors will process. At the same time, FPGAs have greatly increased parallelism of current FPGAs. This paper provides a minimal core set of essential features on which further services can be used.

### The Feniks FPGA Operating System for Cloud Computing

Jiansong Zhang<sup>1</sup>, Yongqiang Xiong<sup>1</sup>, Ningyi Xu<sup>1</sup>, Ran Shu<sup>1</sup>, Bojie Li<sup>1</sup>,  
Peng Cheng<sup>1</sup>, Guo Chen<sup>1</sup>, Thomas Moscibroda<sup>1</sup>,  
<sup>1</sup>Microsoft Research, <sup>2</sup>Tsinghua University, <sup>3</sup>USTC,  
(jiucheng.yq.xiong.yx.nx.sshu.bj.li.peng.chen.tmoscibro@microsoft.com)

#### ABSTRACT

Driven by explosive growth of Moore's law, FPGAs are becoming an important part of server-side acceleration. In this paper, we present Feniks, a new FPGA operating system for cloud computing. Feniks is designed to be a minimal core set of essential features on which further services can be used.

### A Cloud-Scale Acceleration Architecture

Adrian M. Caulfield, Eric S. Chung, Andrew Patnam,  
Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey,  
Palmer Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kaiti Oshchakov,  
Michael Pajunaschad, Lisa Woods, Sitaram Lanka, Derek Chao, Doug Burger  
Microsoft Corporation

#### 1 INTRODUCTION

Modern hypervisor datacenters have made huge strides with improvements in networking, virtualization, energy efficiency, and infrastructure management, but still have the same basic structure as they have for years: individual servers with multicore CPUs, DRAM, and local storage, connected by the NIC through Ethernet switches to other servers. At hypervisor scale, this structure is inefficient and expensive. The key difference over previous work is that the accelerators are now integrated into the server architecture.

#### 2 BACKGROUND

Modern hypervisor datacenters have made huge strides with improvements in networking, virtualization, energy efficiency, and infrastructure management, but still have the same basic structure as they have for years: individual servers with multicore CPUs, DRAM, and local storage, connected by the NIC through Ethernet switches to other servers. At hypervisor scale, this structure is inefficient and expensive. The key difference over previous work is that the accelerators are now integrated into the server architecture.

Abstract—Hypervisor datacenters have struggled to balance the growing need for specialized hardware (efficiency) with the economic benefits of homogeneity (manageability). In this paper, we propose a new cloud architecture that uses reconfigurable logic to accelerate basic network, storage, database, and application. This Configurable Cloud architecture places a layer of reconfigurable logic (FPGAs) between the network switches and the servers, enabling acceleration of local applications running on the servers, and enabling the FPGAs to communicate directly, at datacenter scale, to harvest remote FPGAs unused by their local servers. We deployed this design over a production server farm, and show how it can be used for both server acceleration (with search ranking and network acceleration) and for data in transit at high-speeds. This architecture is much more scalable than prior work which used secondary rack-scale networks for inter-FPGA communication. By coupling to the network plane, direct FPGAs-to-FPGA messages can be achieved at comparable latency to previous work, without the secondary network. Additionally, the scale of direct inter-FPGA messaging is much larger. The average round-trip latencies observed in our measurements among 24, 1000, and 250,000 machines are under 3.5, 10, and 20 microseconds, respectively. The Configurable Cloud architecture has been deployed at hypervisor in Microsoft's production datacenters worldwide.

#### 1 INTRODUCTION

Modern hypervisor datacenters have made huge strides with improvements in networking, virtualization, energy efficiency, and infrastructure management, but still have the same basic structure as they have for years: individual servers with multicore CPUs, DRAM, and local storage, connected by the NIC through Ethernet switches to other servers. At hypervisor scale, this structure is inefficient and expensive. The key difference over previous work is that the accelerators are now integrated into the server architecture.

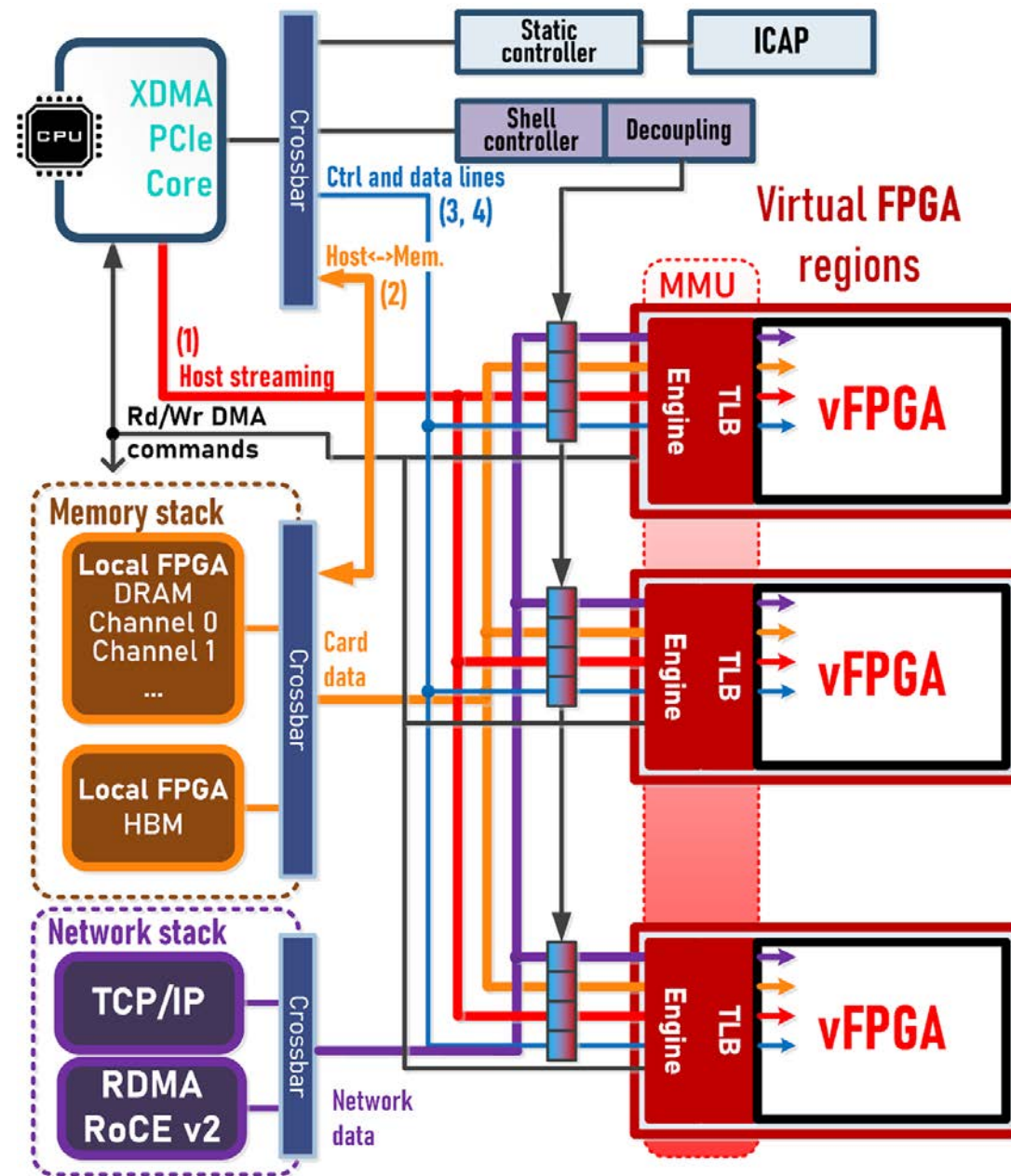
#### 2 BACKGROUND

Modern hypervisor datacenters have made huge strides with improvements in networking, virtualization, energy efficiency, and infrastructure management, but still have the same basic structure as they have for years: individual servers with multicore CPUs, DRAM, and local storage, connected by the NIC through Ethernet switches to other servers. At hypervisor scale, this structure is inefficient and expensive. The key difference over previous work is that the accelerators are now integrated into the server architecture.

# Coyote

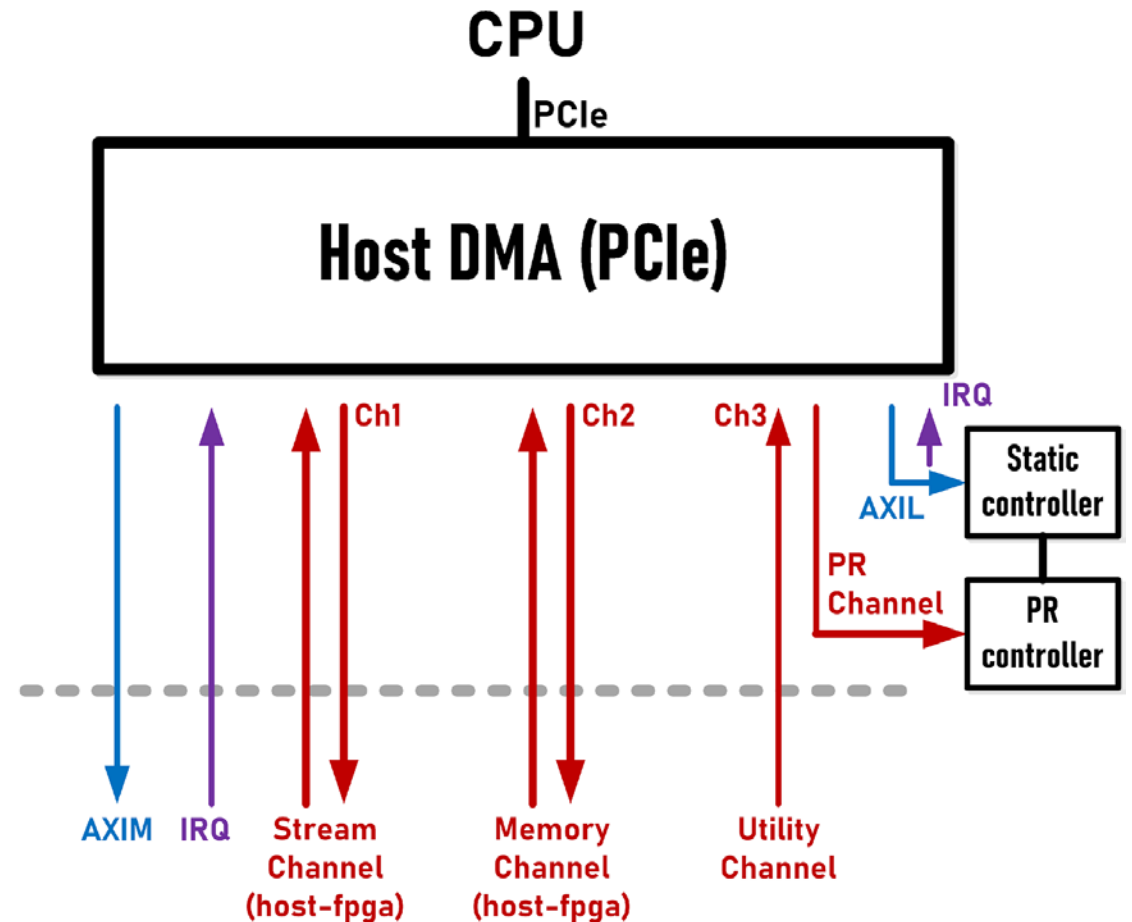
## HW Architecture

- ❖ Microkernel for FPGAs
  - Hardware split into 3 regions:
    - **Static region**
    - **Service (shell) region**
    - **Application region (further split into trusted wrapper and untrusted user app)**
- ❖ Host software consists of:
  - Kernel driver (hypervisor)
  - Runtime scheduler
  - High level API
- ❖ Runs on Alveo, Enzian



# Host <-> FPGA connection

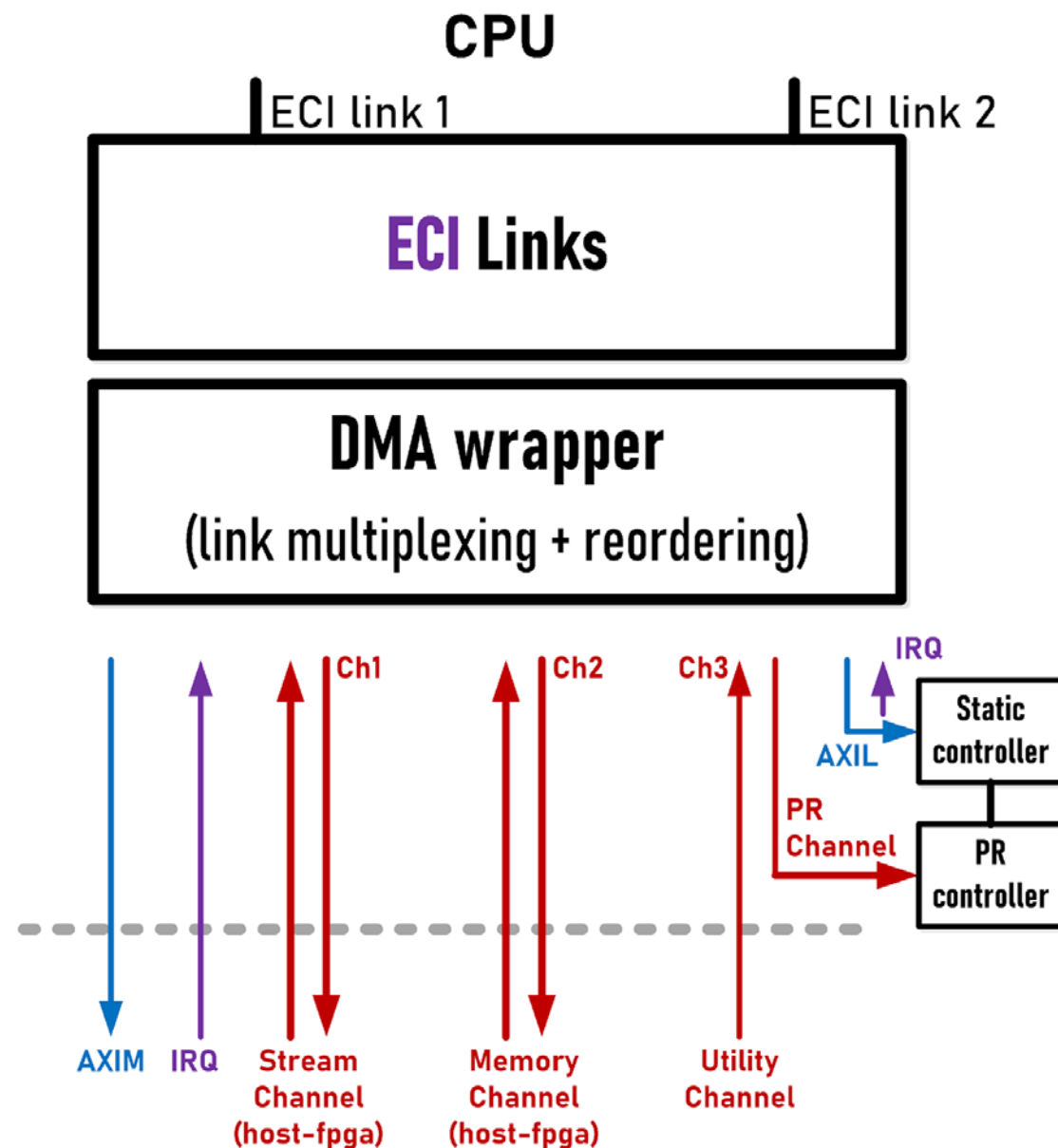
- ❖ PCIe connection (cards still at gen3 x16 ...)
- ❖ XDMA IP<sup>[1]</sup> core used with Alveo cards (trial runs with QDMA core as well)
- ❖ Very little dependency on the actual DMA core used
- ❖ Port to open source cores



[1]: <https://docs.xilinx.com/r/en-US/pg195-pcie-dma/Introduction>

# ECI – Enzian Coherency Interface

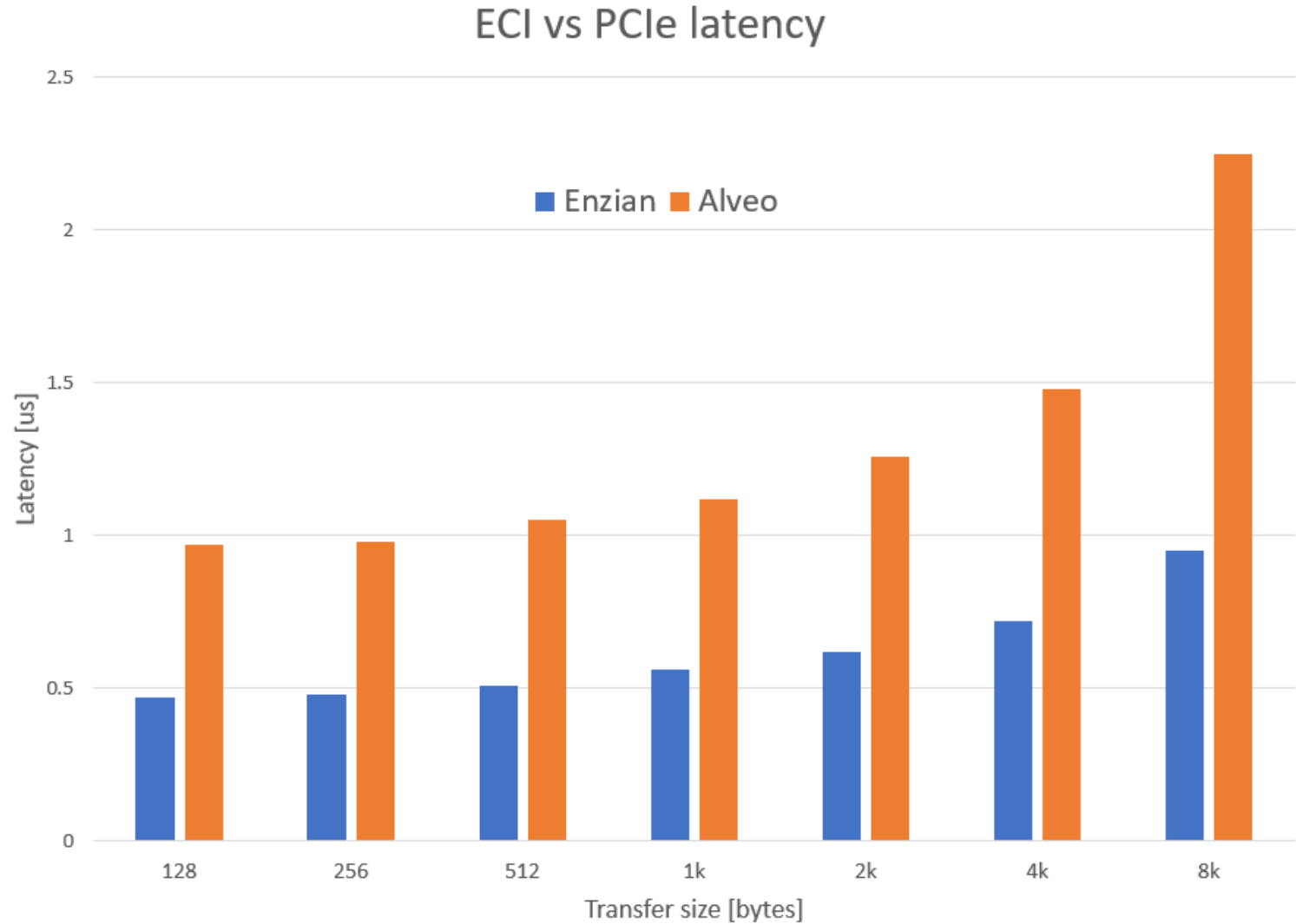
- ❖ Cache coherent interface<sup>[2]</sup>
- ❖ Coherency between CPU and FPGA side memories
- ❖ Our own DMA wrapper built on top which provides integration for Coyote interfaces



[2]: Enzian: An Open, General, CPU/FPGA Platform for Systems Software Research, ASPLOS 2022

# ECI vs XDMA

- ❖ Throughput PCIe:
  - ❖ ~12 GB/s
- ❖ Throughput ECI:
  - ❖ ~21 GB/s





# Coyote

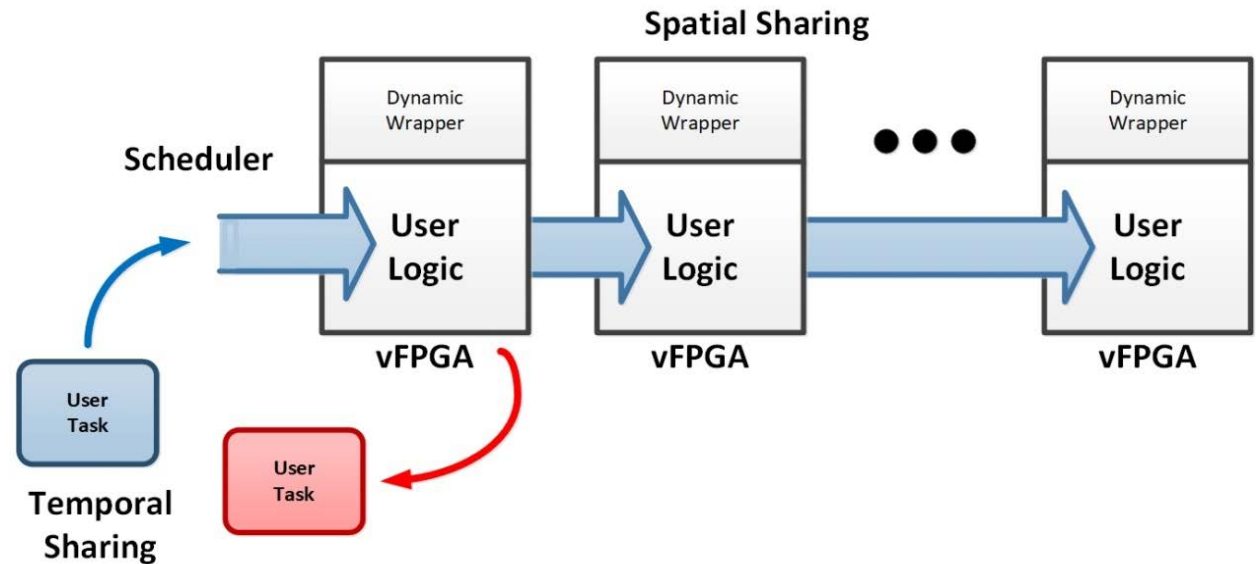
Processes, Threads, Tasks ...

## ❖ FPGAs are fundamentally different

- No CPUs or cores
- Spatial and temporal sharing

## ❖ Coyote multi-tenancy:

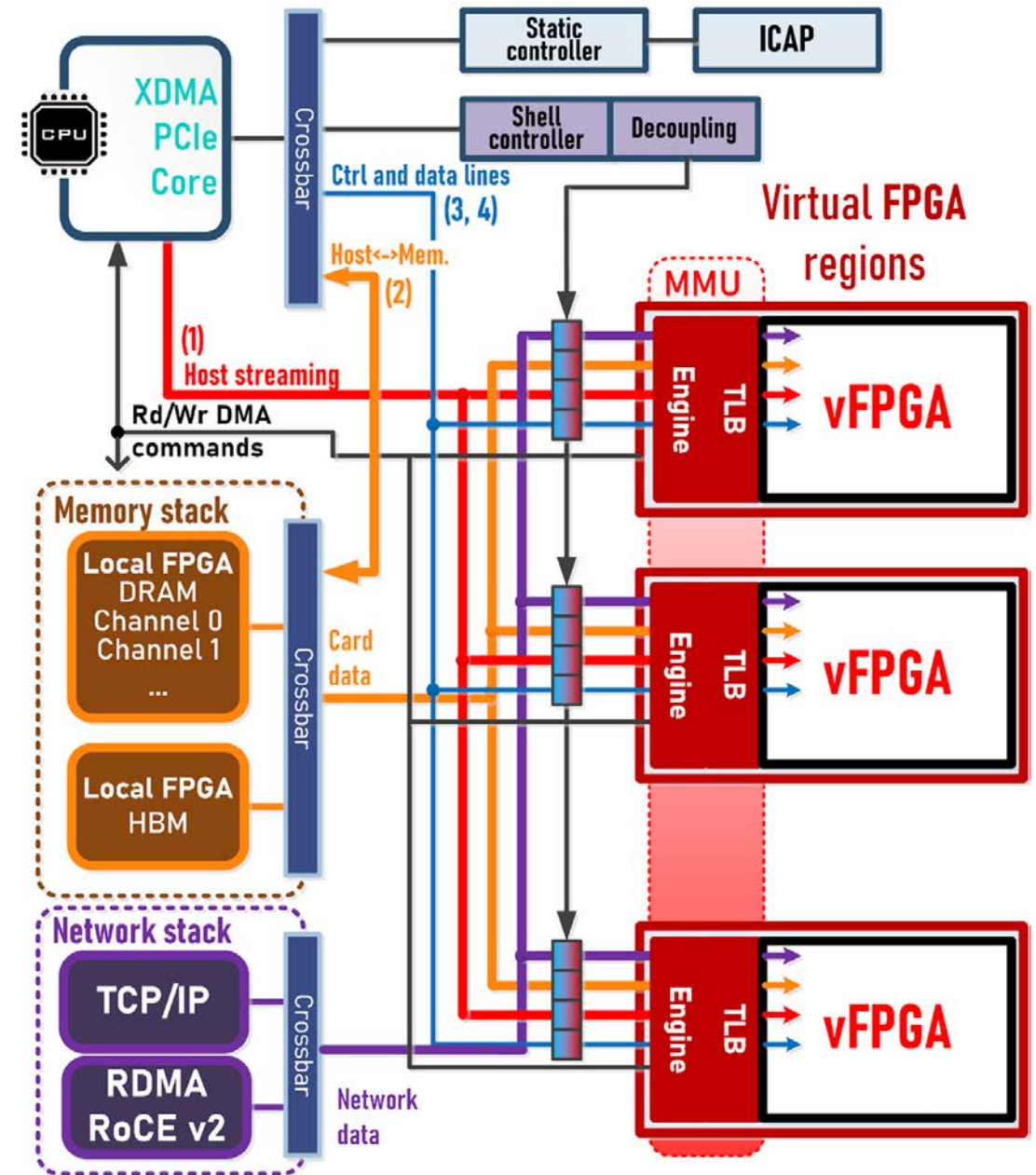
- Combines both approaches
- Multitasking abstraction for a set of independent, isolated vFPGAs
- Applications within vFPGAs are untrusted



# Coyote

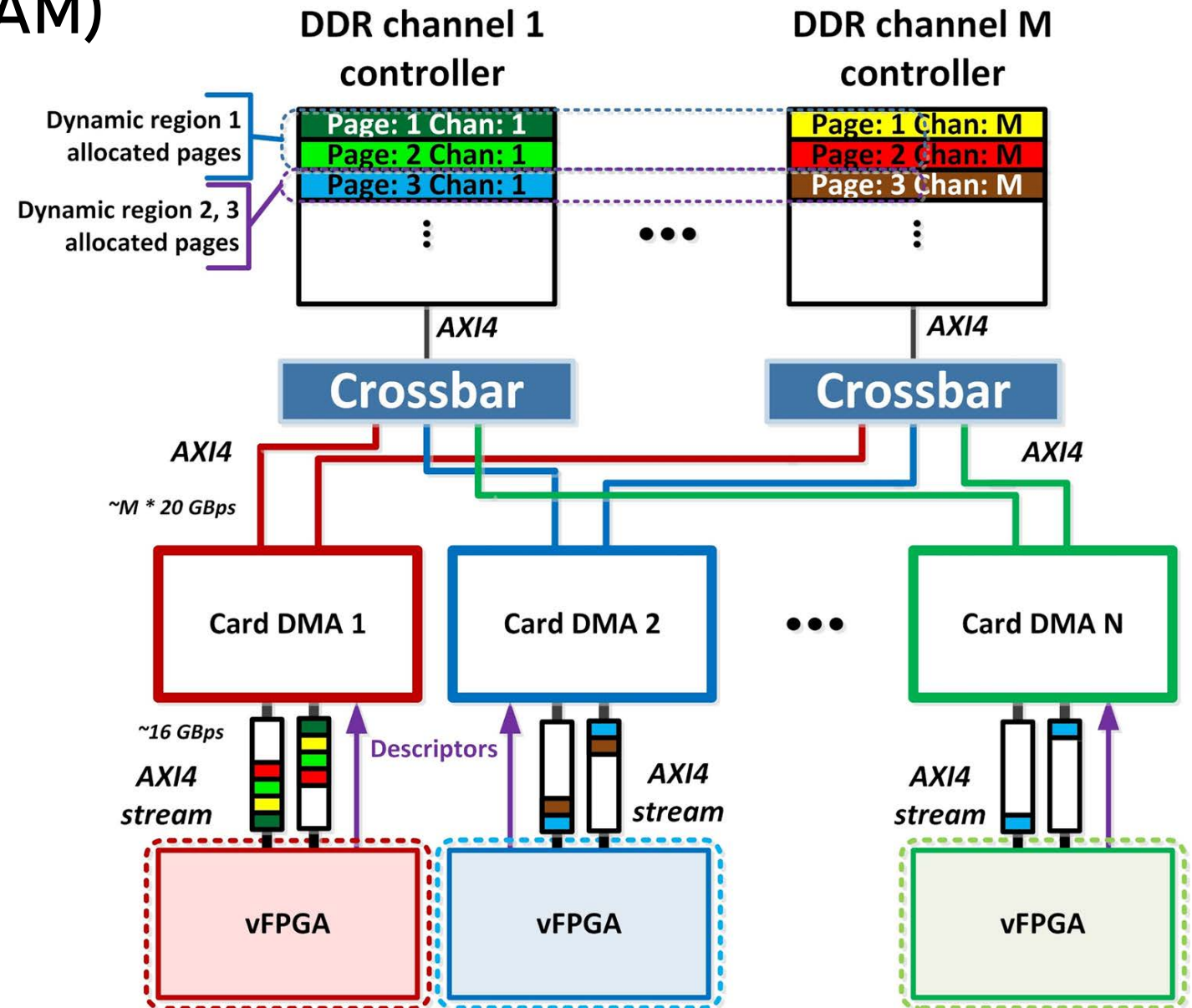
## Virtual Memory

- ❖ Important abstraction
  - Provides the illusion of unlimited memory, simplifies compilation, provide protection
- ❖ Typically missing in FPGAs !
- ❖ Dedicated MMU layer in Coyote
  - Same virtual address space between HW SW
  - Applications can master transactions (pointer chasing example ...)
  - RDMA ...
  - Virtualization of FPGA side memory
  - Additional memory models on top



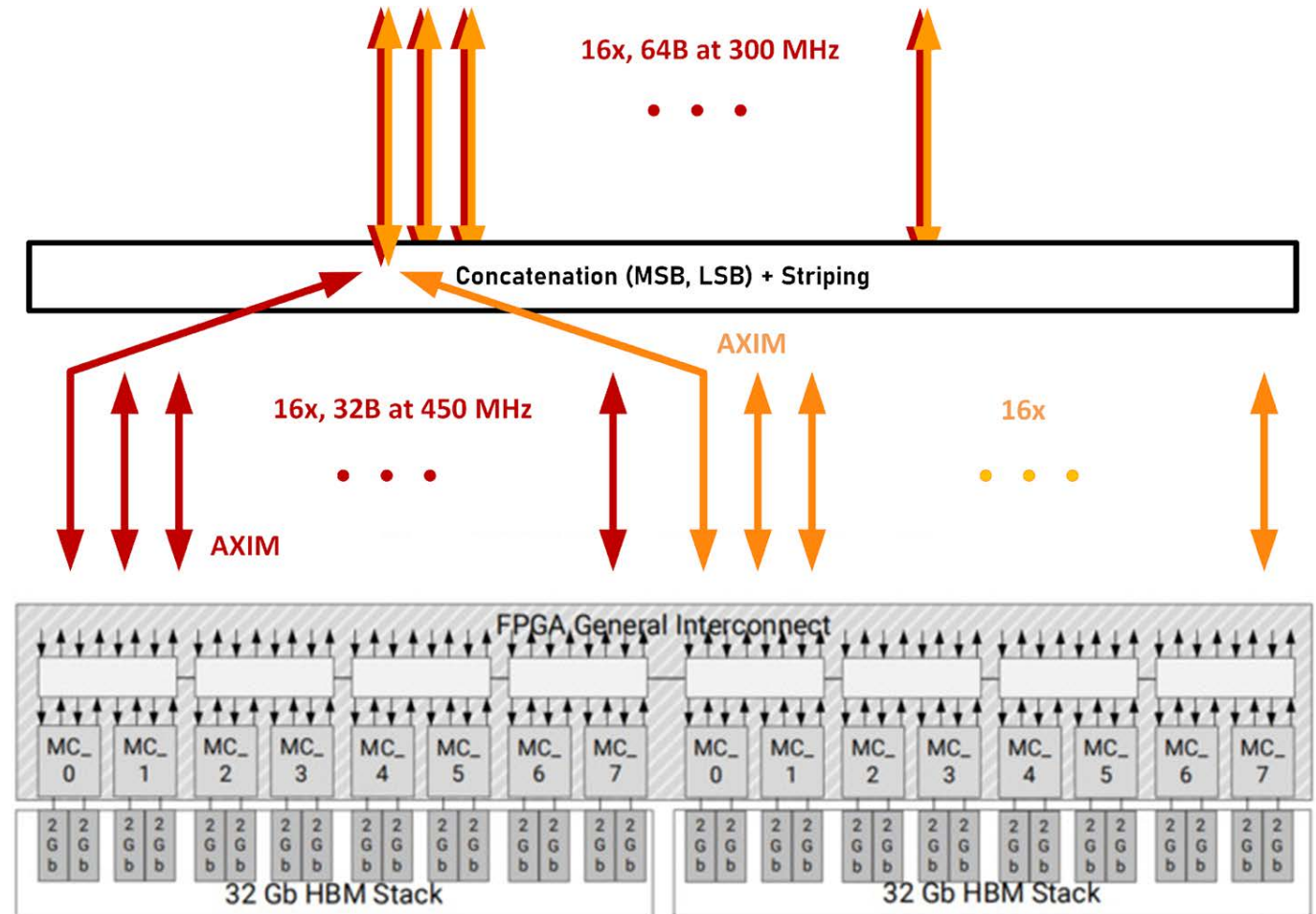
# Memory organization (DRAM)

- ❖ Dynamic allocation across all available channels
- ❖ Striping access pattern
- ❖ Same interface (DRAM, HBM, ...)
- ❖ Optimizes bandwidth distribution across vFPGAs



# Memory Organization (HBM)

- ❖ Single channel has access to the whole memory
- ❖ RAMA IP<sup>[3]</sup> core used for striping (same idea ...)
- ❖ Current HBM in Alveo cards: 32 AXIM channels at 450 MHz
- ❖ Data width conversion from 32B to 64B, max thr: ~14 GB/s
- ❖ Coyote typically runs at ~300 MHz, max thr: ~19 GB/s

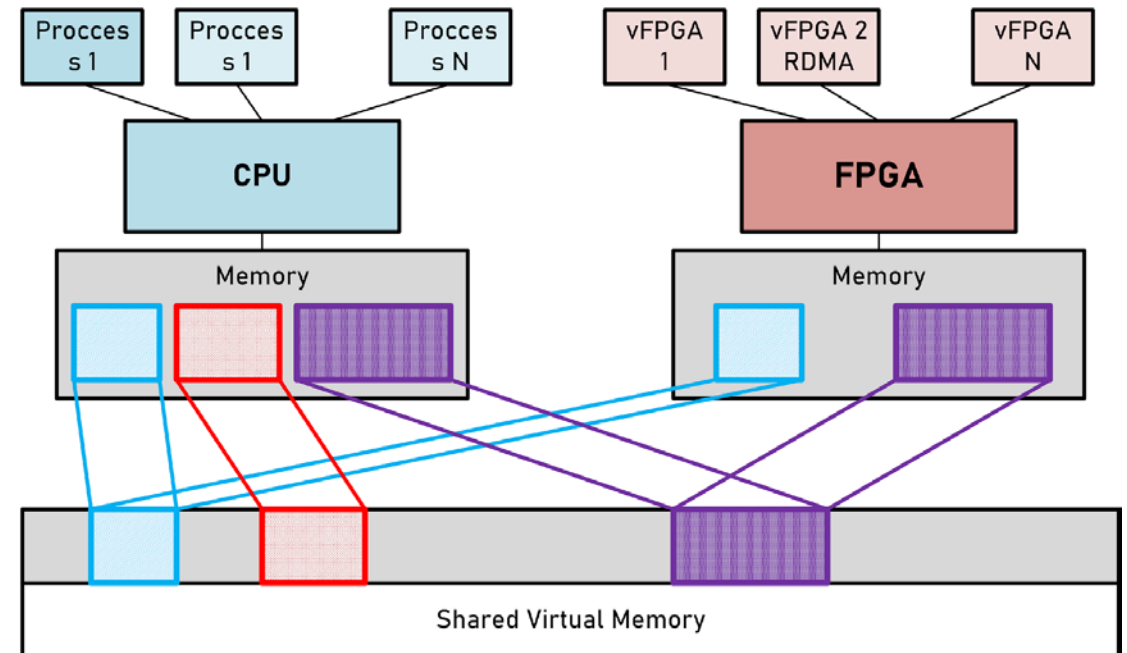


[3]: <https://www.xilinx.com/products/intellectual-property/rama.html>

# Coyote

## Shared Virtual Memory (Unified Memory)

- ❖ Programming model which allows multiple devices with independent memories to be programmed as if they share memory
- ❖ Seamless sharing of host and device side user space pointers
- ❖ Provides memory consistency between host and device accesses (Heterogeneous Memory Management API)
- ❖ Used extensively in modern GPUs



# Unified Memory Example

// NO UNIFIED MEMORY

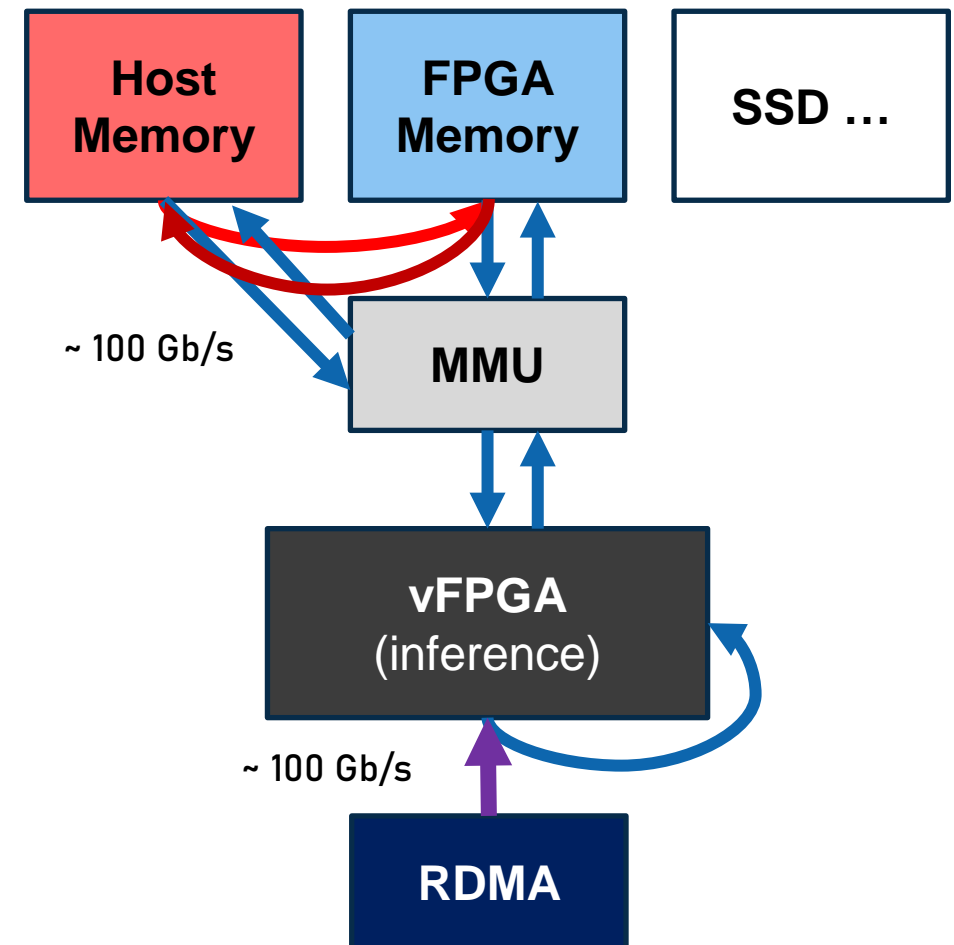
// WITH UNIFIED MEMORY

IRQ handling overhead:

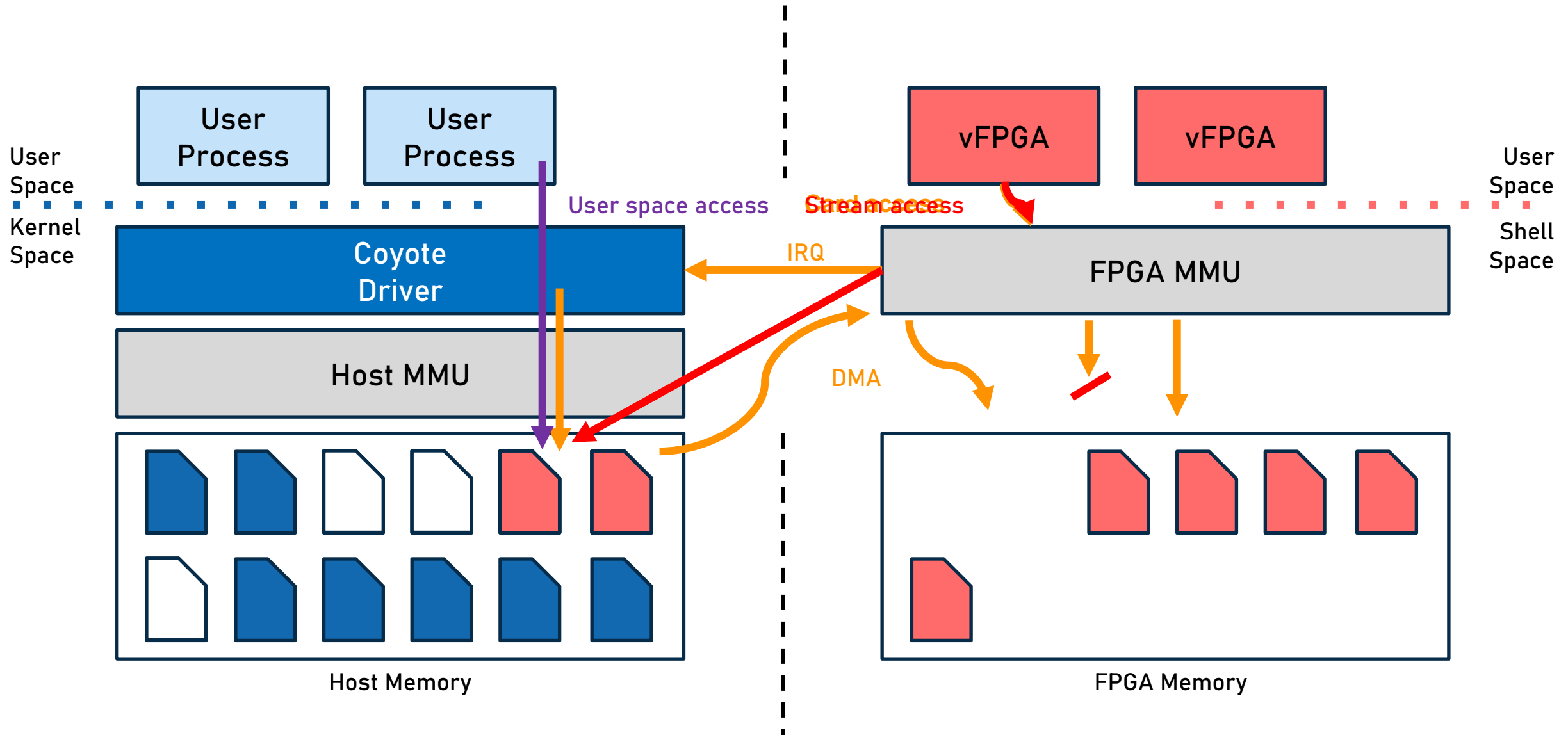
```
vm:  
mean: 2430.298  
medain: 2396.0  
std: 644.358263077304  
host:  
mean: 2470.978  
medain: 2402.0  
std: 1608.3578157599134
```

# FPGAs are not GPUs ...

- ❖ Direct access and corresponding interfaces are needed (streaming interfaces)
- ❖ Especially if network and/or other I/O comes into play ...
- ❖ Bump-in-the-wire processors
- ❖ How to integrate this within HMM(SVM)?



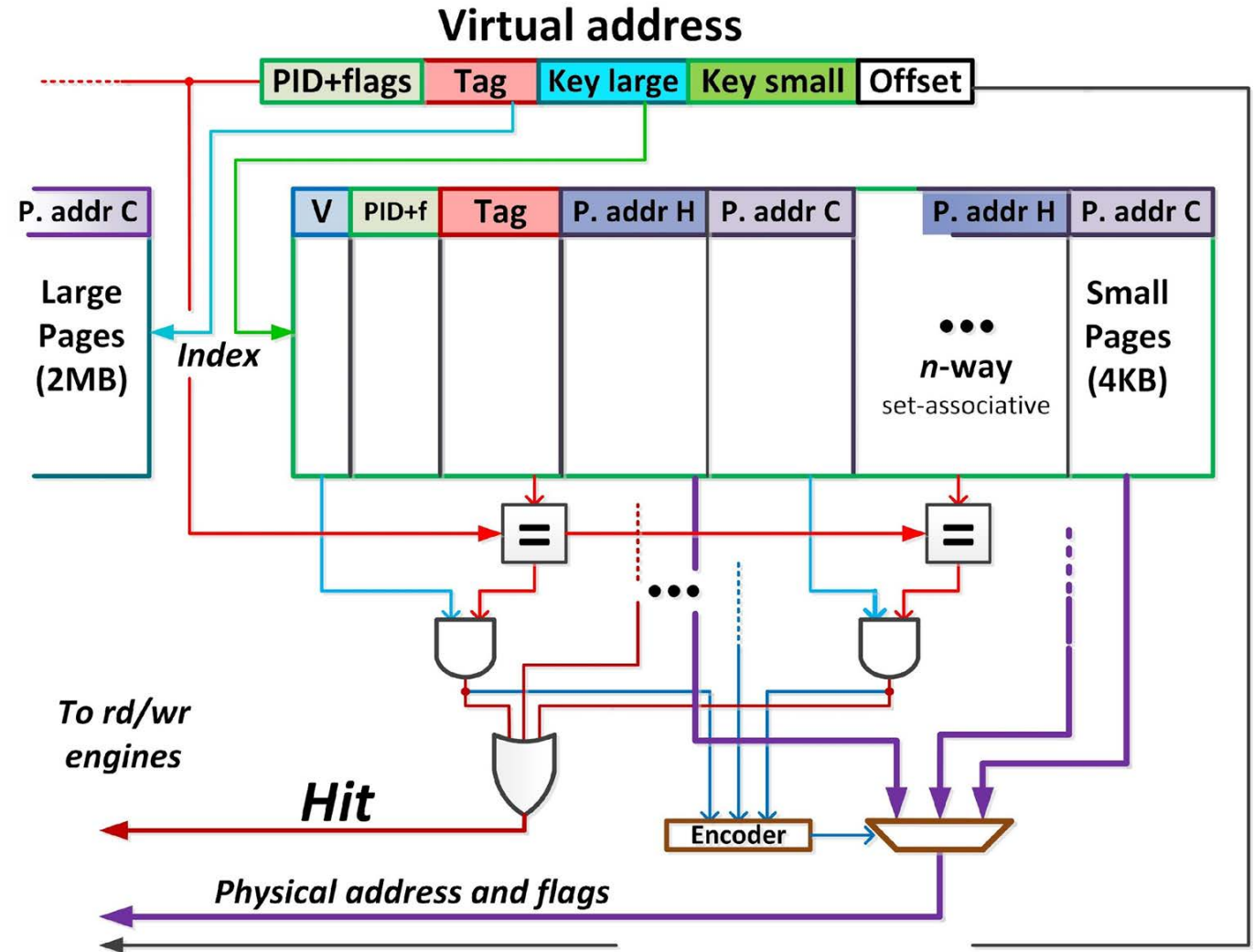
# HMM operation





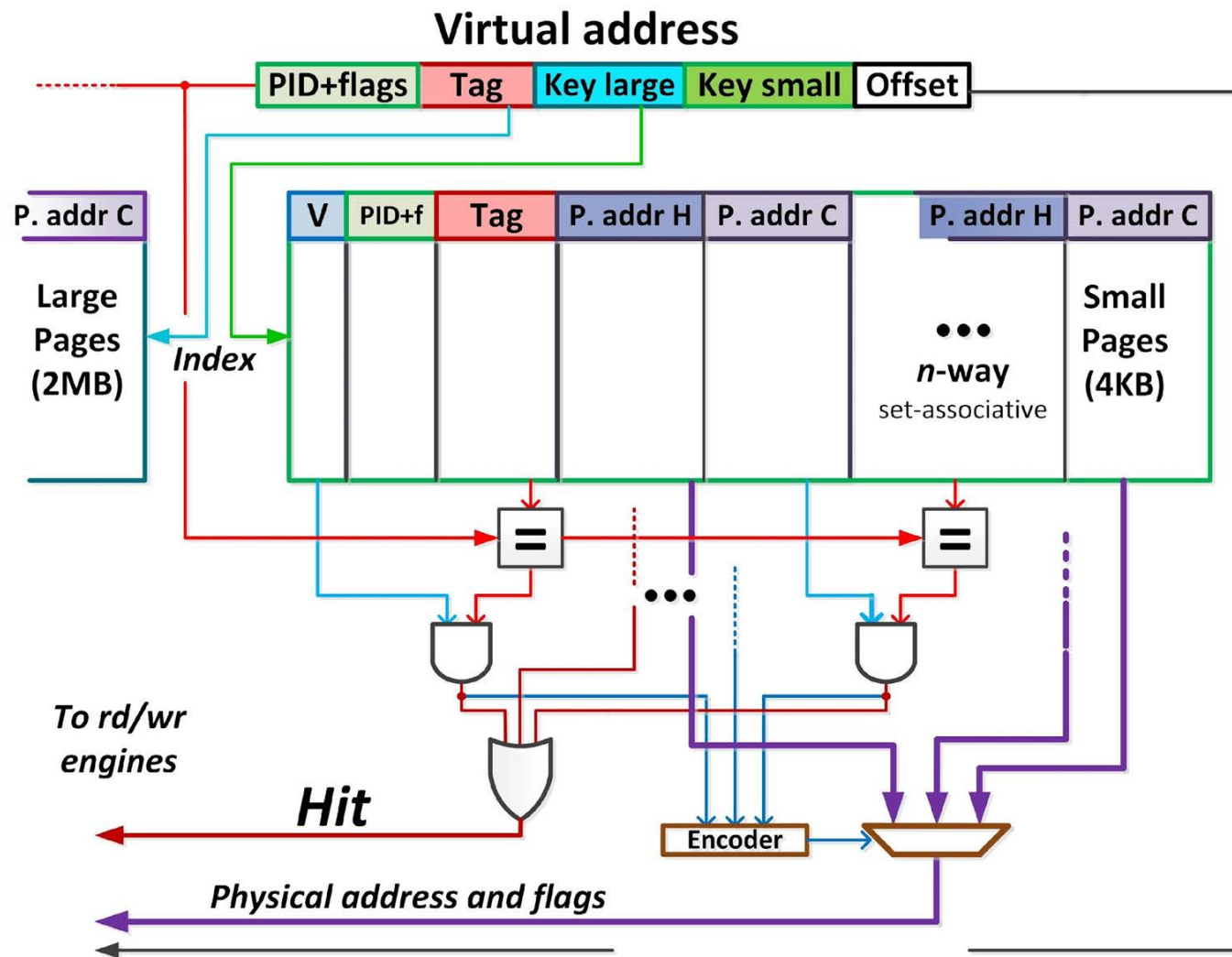
# MMU Notifiers

- ❖ Traditionally pages pinned prior to device access
- ❖ Callback functions within MMU notifiers can keep the TLBs up to date without pinning
- ❖ Invalidations need to be handled within properly



# HMM Obstacles ...

- ❖ Coverage with hugePages (hugetlbfs), variable TLB page size
- ❖ Transparent HugePages (khugepaged thread)
- ❖ API being changed ...



# Coyote

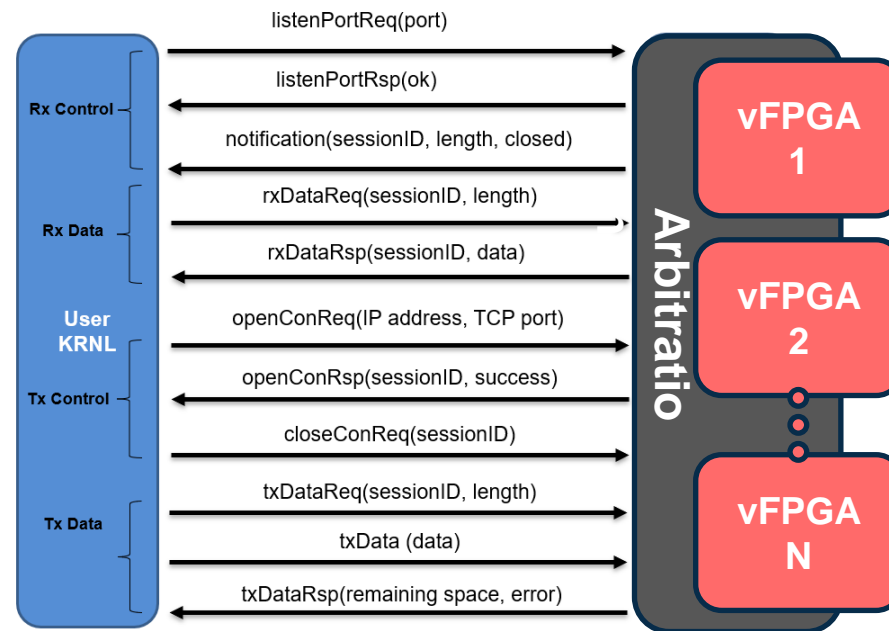
## Network Stacks

- ❖ 100G TCP/IP and RDMA (RoCE v2) network stacks<sup>[4]</sup>
- ❖ Reliable protocols (use single channel of on-board memory)
- ❖ ~20% of overall FPGA resources
- ❖ Interleaved, accessible by all tenants (vFPGAs)

[4]: <https://github.com/fpgasystems/fpga-network-stack>

# TCP stack

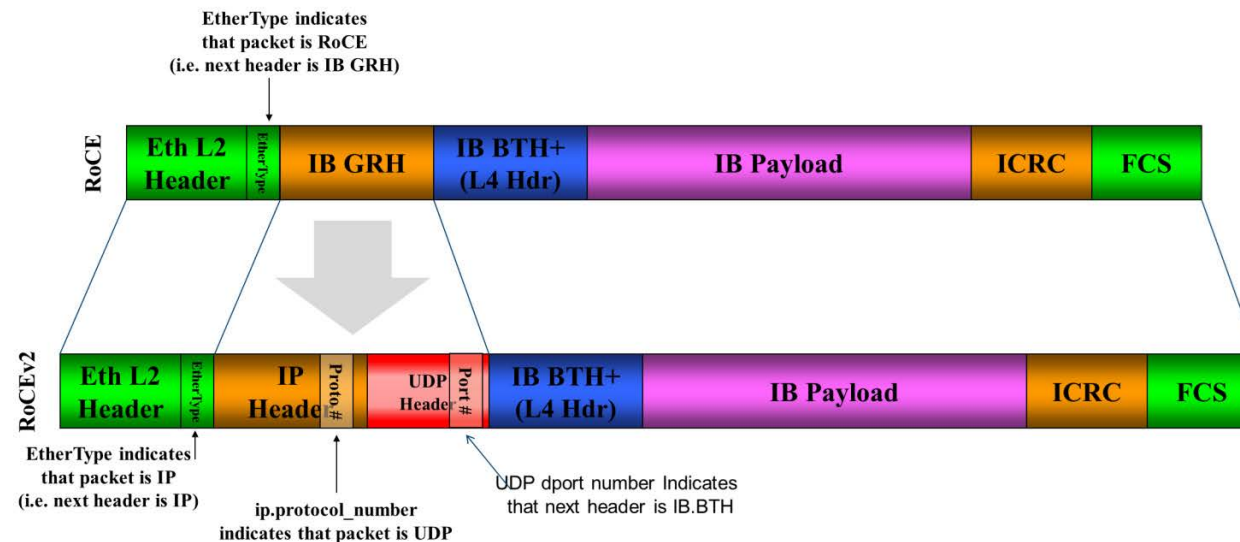
- ❖ Shared across vFPGAs
- ❖ Multiple accelerators serving clients on different ports
- ❖ AXI stream interface (opening and closing ports and connections,
- ❖ Lower level of abstraction than RDMA !



[5]: [https://github.com/fpgasystems/Vitis\\_with\\_100Gbps\\_TCP-IP/blob/vitis\\_2022\\_1/img/interface.png](https://github.com/fpgasystems/Vitis_with_100Gbps_TCP-IP/blob/vitis_2022_1/img/interface.png)

# RDMA stack

- ❖ Open source RDMA stack (UC, RC)<sup>[5]</sup>
- ❖ RDMA over Converged Ethernet (RoCE v2)
- ❖ Implemented on top of UDP/IPv4/IPv6 (far lower overhead than iWARP)
- ❖ InfiniBand (IB) transport packets over Ethernet (READ, WRITE, SEND)



<https://docs.nvidia.com/networking/display/WINOFv55053000/RoCEv2>

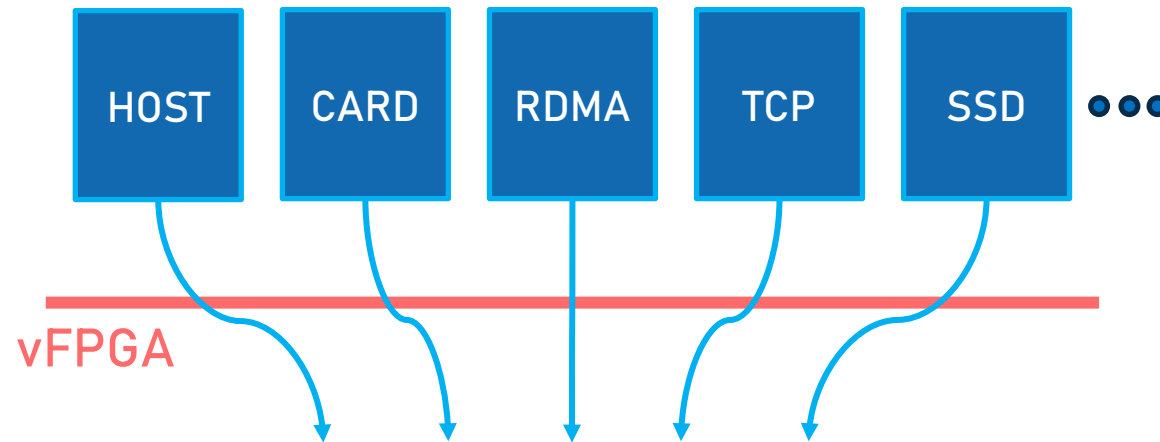
[5] *StRoM: Smart Remote Memory, EuroSys '20*

David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, Gustavo Alonso

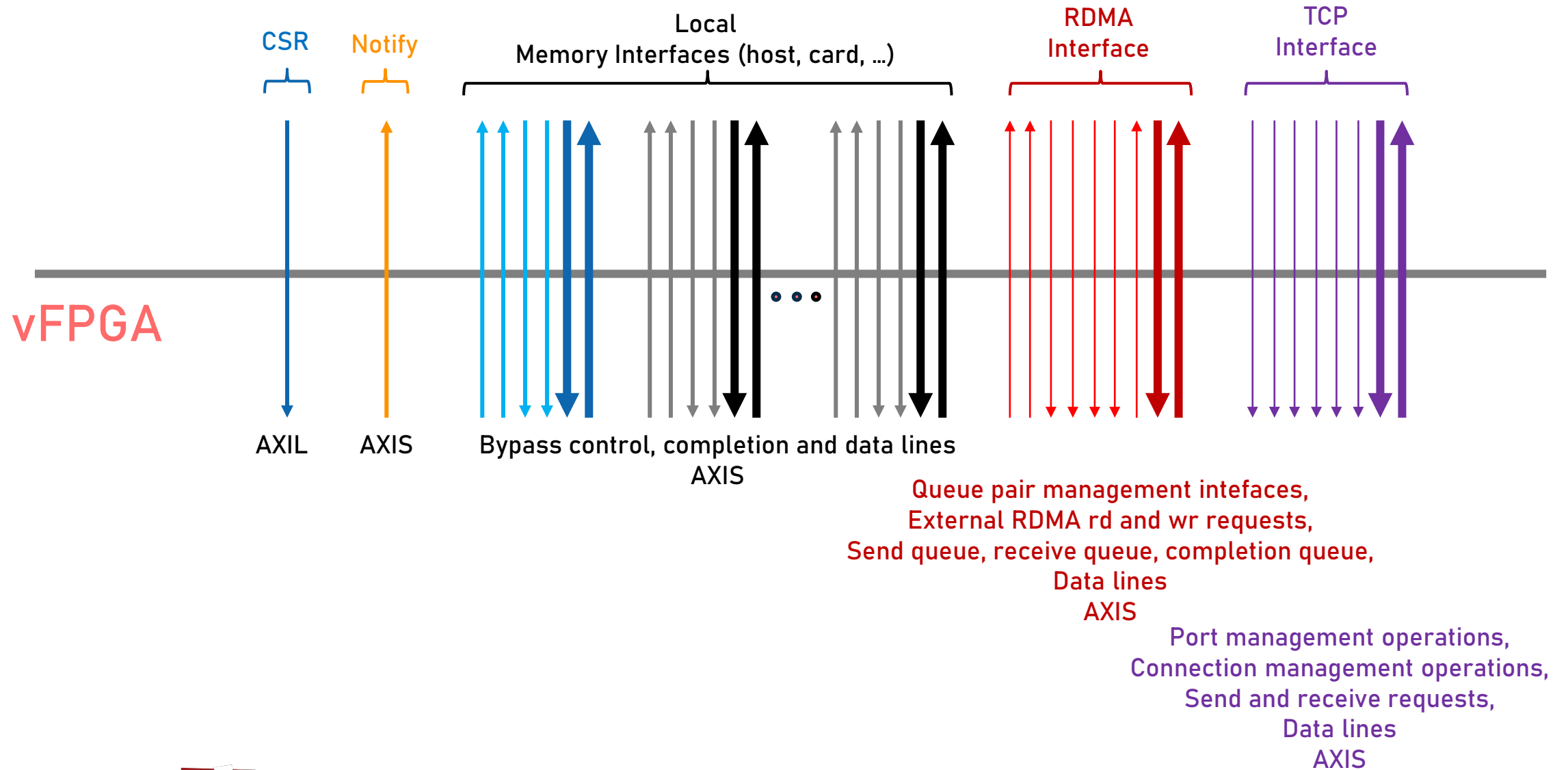
# Coyote

## Unified Logic Interface

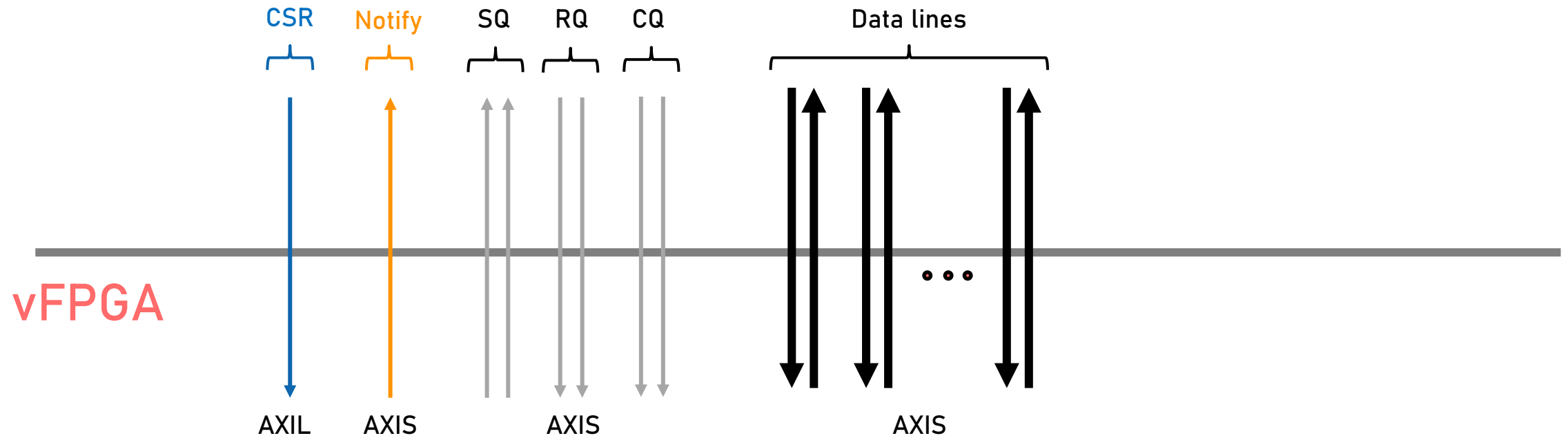
- ❖ Don't really exist across FPGA platforms ...
  - ❖ Vitis HLS
  - ❖ Much higher flexibility than other devices
- ❖ User Logic Interface
  - ❖ AXI streams (descriptor interface)



# Unified Logic Interface



# Can this be simplified?





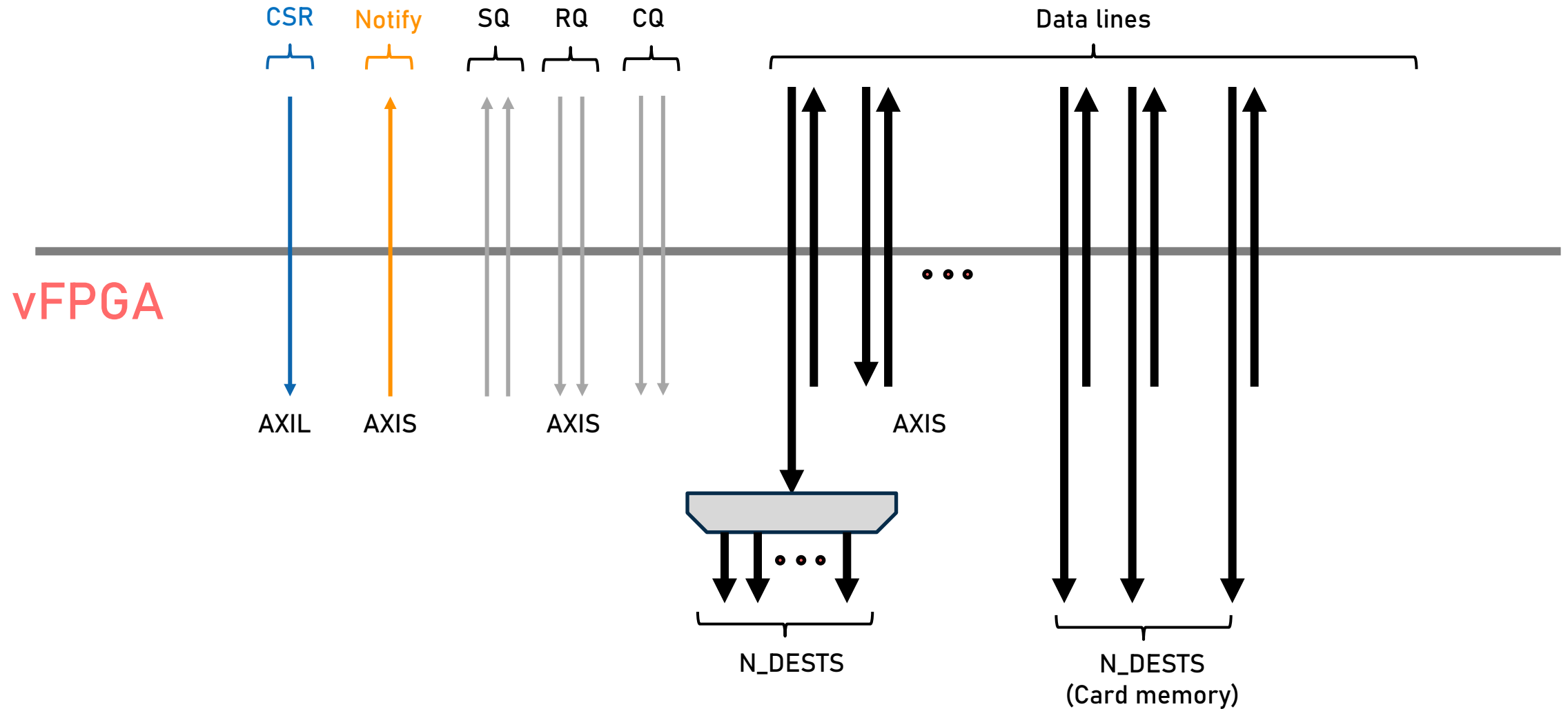
# SQ Interface

- ❖ All local(host, card, ...) and remote(RDMA, TCP/IP) operations can be invoked through SQ interface



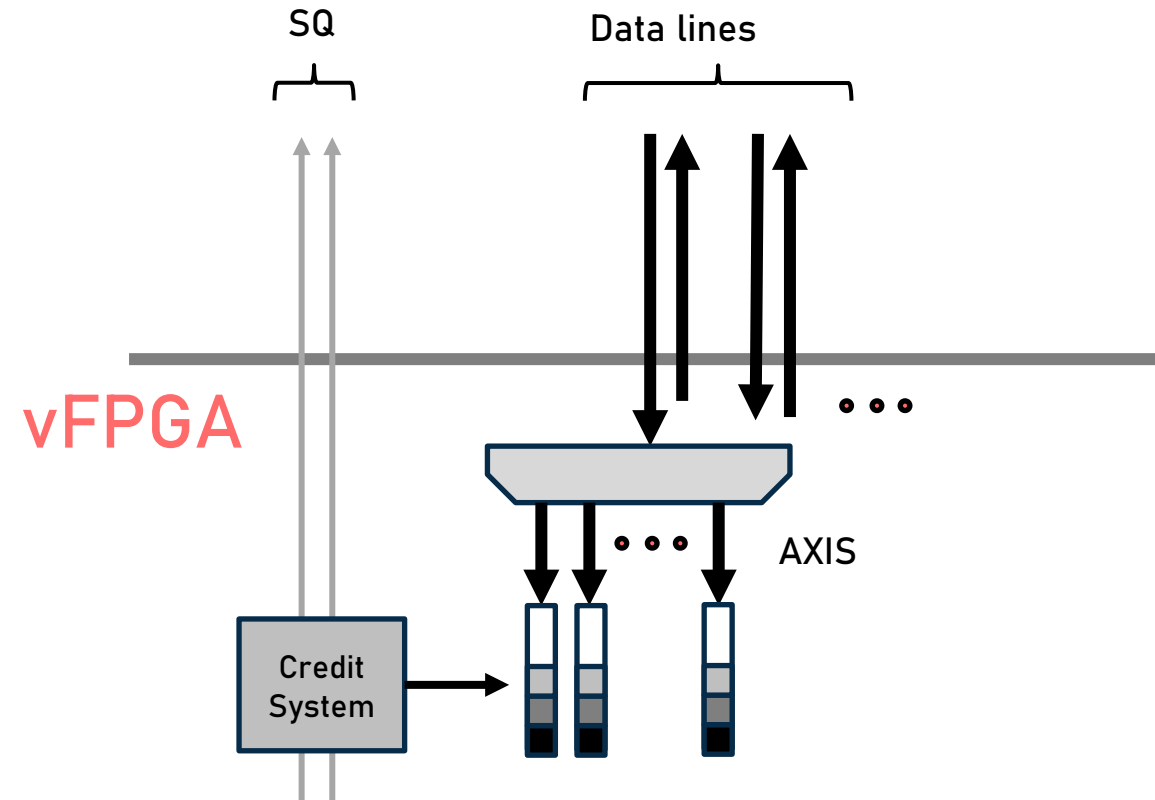
- OPCODE – Operation to be invoked
  - CPID – Host process identifier
  - VFID – vFPGA (reserved)
  - DEST – Destination queue
  - SID – Connection ID
  - VADDR – User space shared pointer
  - LEN – Transfer size
- ❖ RQ – Received remote requests (RDMA and TCP/IP)
  - ❖ CQ – Completion events

# Destination queues



# Untrusted environment

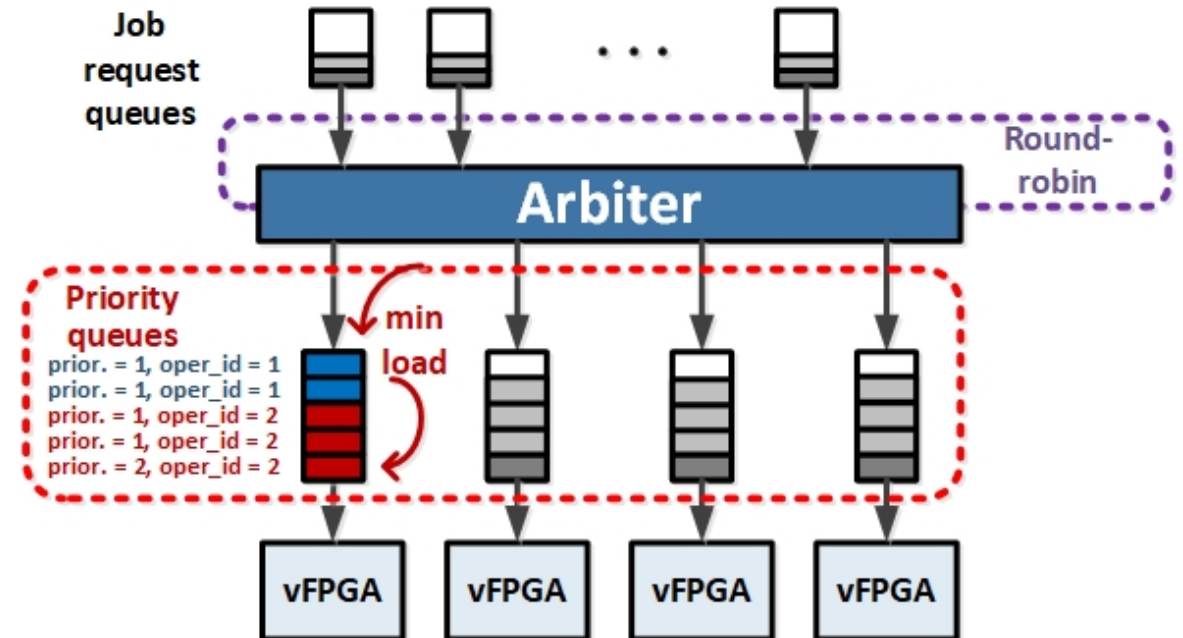
- ❖ Credit system for each destination queue:
  - Write requests are issued only if accompanying data is provided
  - Read requests are issued only if queues are ready to accept the data
- ❖ Credit system for all local and remote requests
- ❖ Can quickly overwhelm the resources ...
- ❖ For RQ, RDMA accepts packets only if vFPGA is able to accept



# Coyote

## Dynamic Reconfiguration and Scheduling

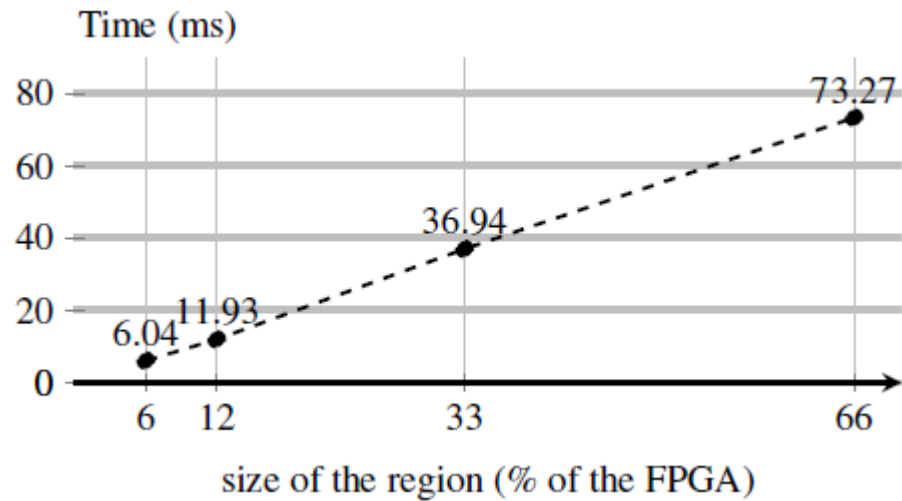
- ❖ Basic mechanisms to capture the state of the FPGA don't exist
- ❖ Non-preemptive task based approach
- ❖ Preemption?
  - User application trust
  - Requires a form of cooperation
  - Additional application complexity



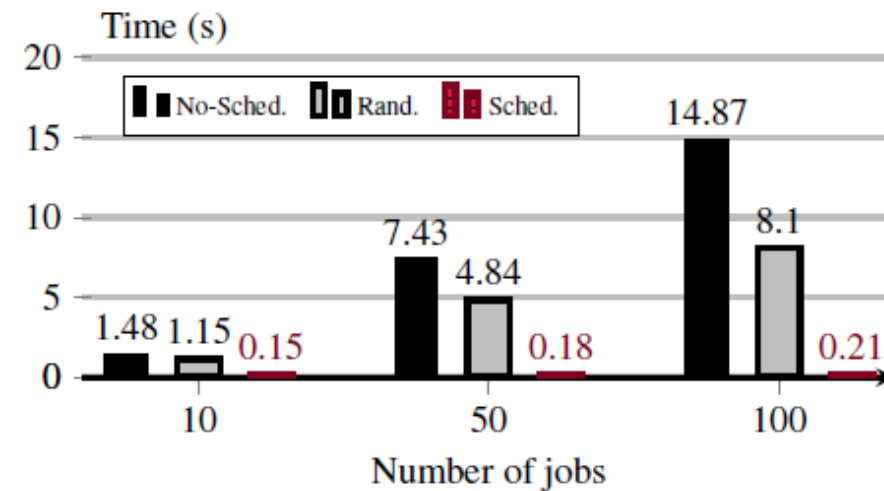
# Reconfiguration overhead and scheduling efficiency

- ❖ Penalty of partial reconfiguration is high
- ❖ Modified priority queue based scheme reduces overall execution time

## Reconfiguration times



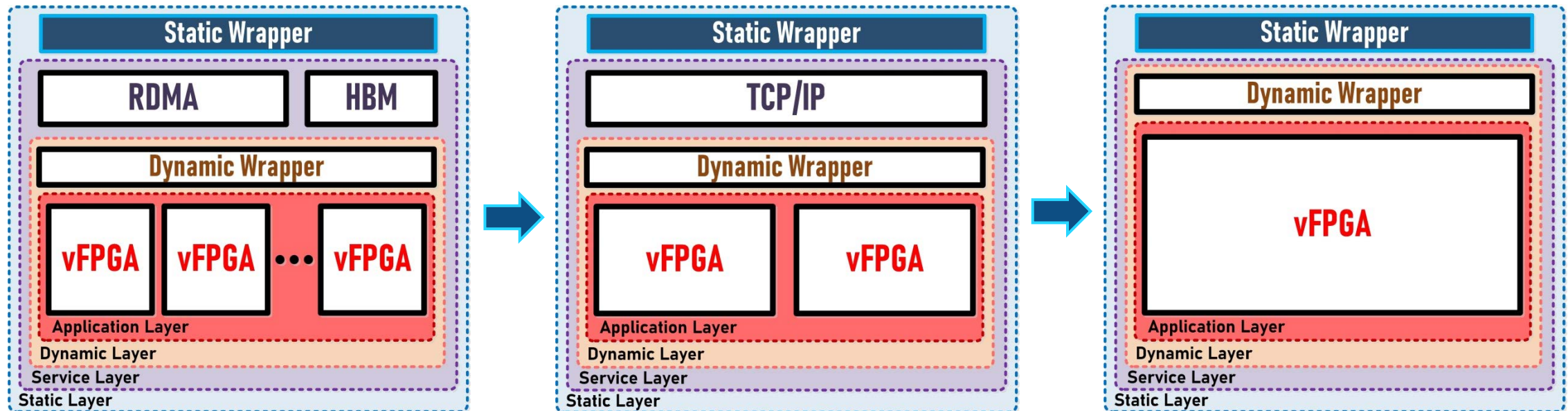
## Scheduling algorithm



# Hierarchical Reconfiguration

## ❖ Nested reconfiguration (*Nested-DFX*)

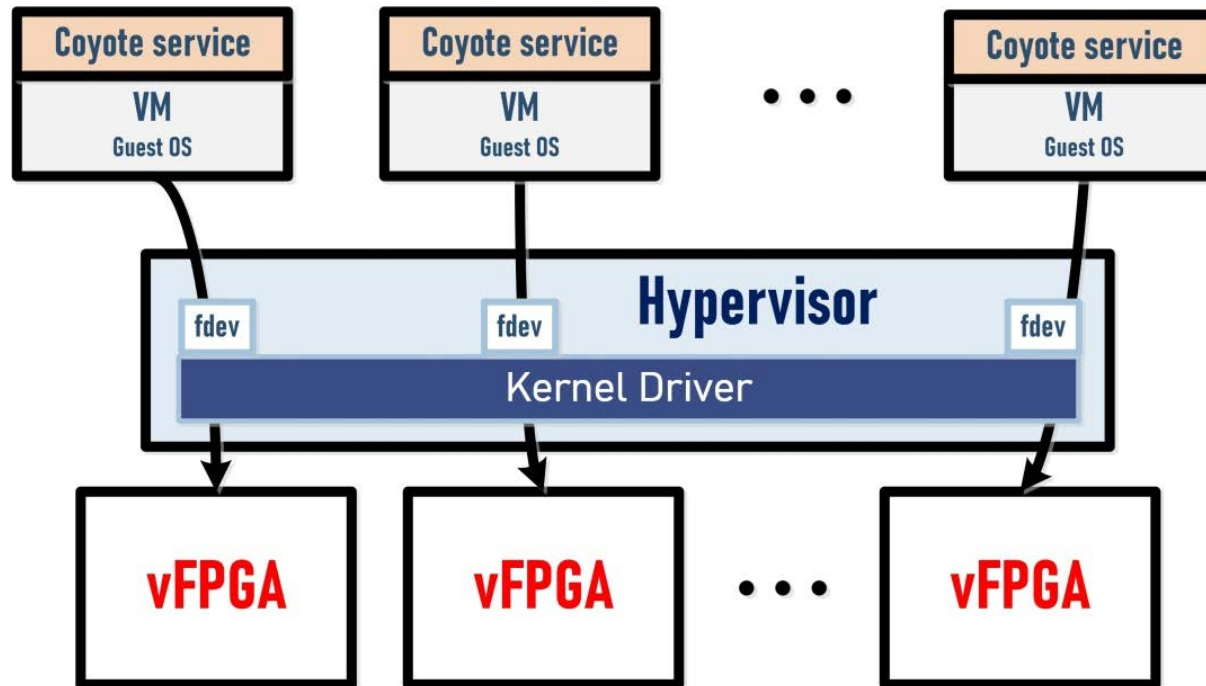
- ❑ Static layer
- ❑ Service layer
- ❑ Dynamic layer
- ❑ Application layer



# Coyote

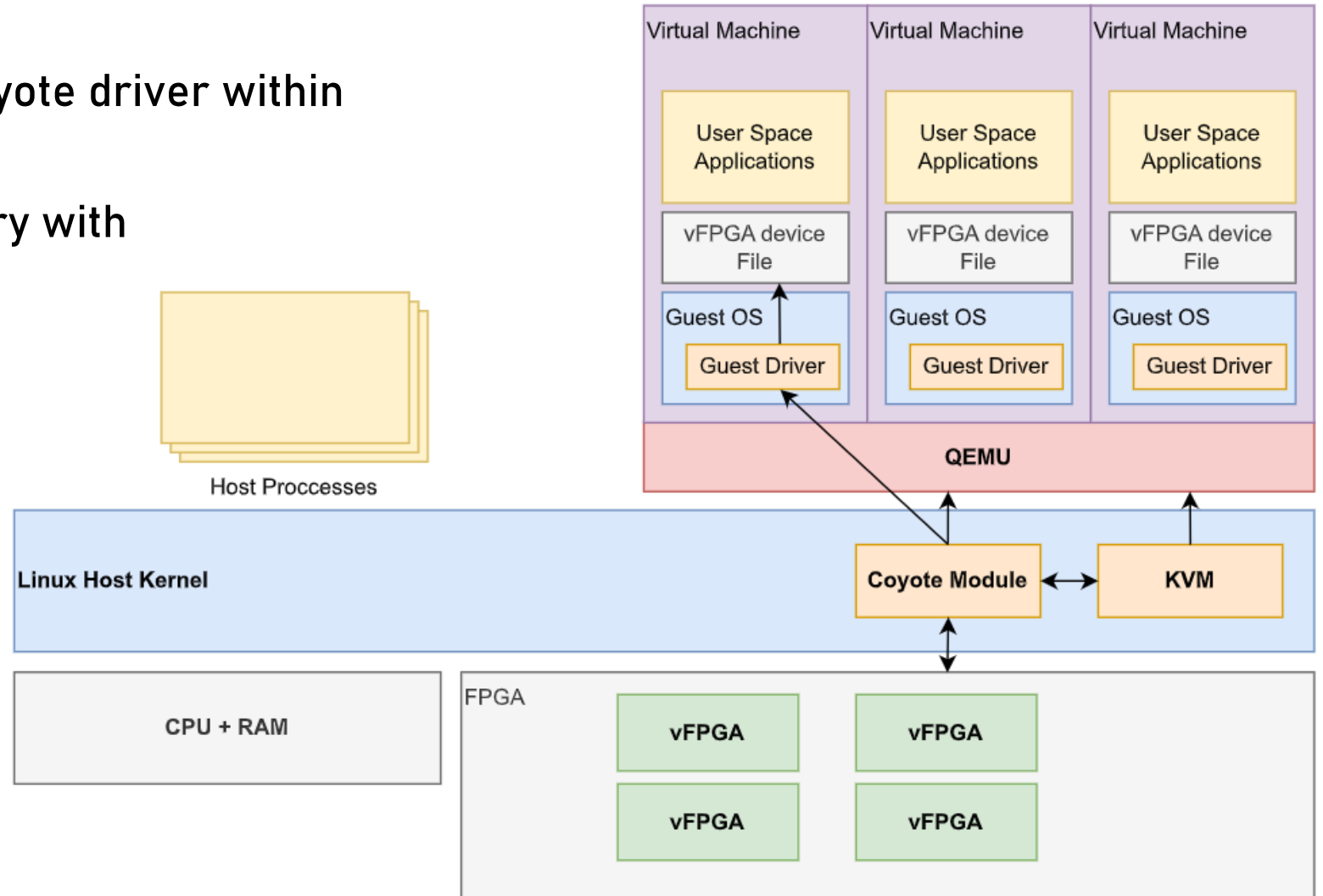
## Virtual Machines

- ❖ Virtualization –vFPGAs «pulled up» all the way to different VMs
  - Virtual Function I/O (VFIO) Mediated devices



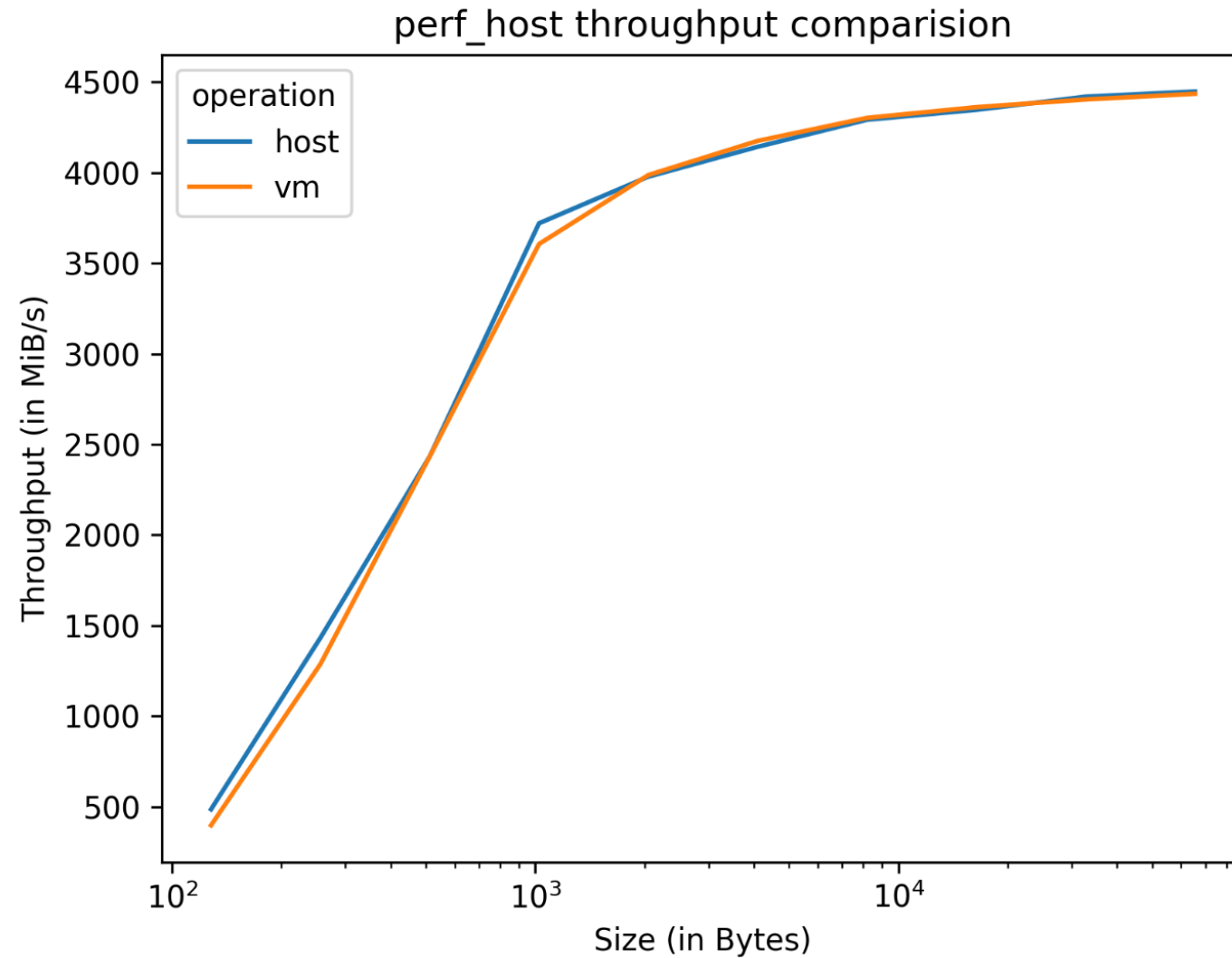
# Virtualization Architecture

- ❖ Hypervisor running on top of Coyote driver within Linux kernel
- ❖ Virtualization of CPU and memory with KVM
- ❖ For emulation of other components QEMU
- ❖ Passthrough through IOCTL interface and emulation of PCI device with VFIO + MDEV





# VM Performance



# Coyote

## SW Architecture

### ❖ Layered parallelization

### ❖ User space abstractions

- ❑ cSched - Coyote scheduler, reconfiguration controller
- ❑ cProc - Coyote process, multiple can run within a single *vFPGA*
- ❑ cThread - Coyote thread, multiple can run within a single *cProc*. Task level parallelism
- ❑ cTask - Coyote task, arbitrary user variadic function executed by *cThreads*
- ❑ cService - Coyote library daemon, background service, UDS for IPC

```
.  
. .  
. .  
/**  
 * Open a Unix Domain Socket and send a decrypt_and_compress task request  
 * This is the only place of interaction with Coyote  
 */  
cLib clib("/tmp/coyote-daemon-vfid-0");  
clib.task({opDecryptAndCompress, {mem, size, key}}); // blocking  
. . .
```

# Coyote

## Build System

- ❖ CMake incremental builds
  - Revamped for hierarchical flow
- ❖ Cores can be both RTL and HLS
  - ↳ hw/aes
    - ↳ cyt\_user\_top.sv
      - ↳ hdl
        - ↳ \*.sv, \*.svh, \*.vhd
      - ↳ hls
        - ↳ \*.cpp, \*.hpp
- ❖ make shell (shell partial bitstream)
- ❖ make app (application partial bitstreams)

```
cmake_minimum_required(VERSION 3.0)
project(test)

set(CYT_DIR ${CMAKE_SOURCE_DIR})
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${CYT_DIR}/cmake)

find_package(CoyoteHW REQUIRED)

# Configuration
set(SHELL_PROBE 1)
set(FDEV_NAME "u55c")
set(COMP_CORES 80)
set(N_REGIONS 2)
set(EN_STRM 1)
set(EN_MEM 1)
set(EN_PR 1)
set(N_CONFIG 3)
set(N_CARD_AXI 2)
set(N_HOST_AXI 2)

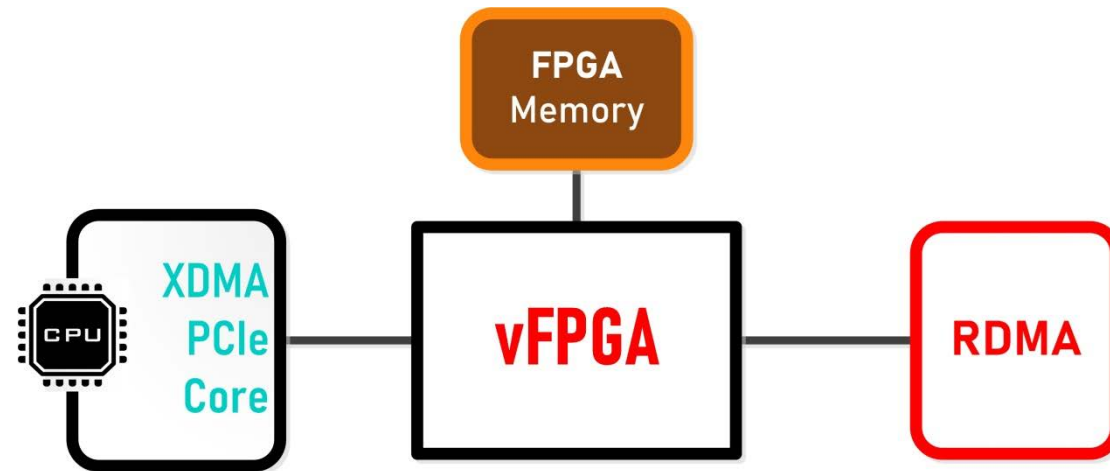
# Load applications
load_apps(
    VFPGA_C0_0 "hw/adder"
    VFPGA_C0_1 "hw/adder"
    VFPGA_C1_0 "hw/aes"
    VFPGA_C1_1 "hw/aes"
    VFPGA_C2_0 "hw/sha"
    VFPGA_C2_1 "hw/sha"
)

create_hw()
```

# Coyote

What kind of stuff can we run on top?

- ❖ ACCL project (Accl: Fpga-accelerated collectives over 100 gbps tcp-ip, *Z. He et al.*)
  - Streaming interfaces
  - Kernel invocation overhead Vitis: ~50us, Coyote: ~1-1.5us
  - RDMA



# Questions?