ETH zürich

Coyote

Abstractions for Modern Heterogeneous Hardware

Dario Korolija, Timothy Roscoe, Gustavo Alonso

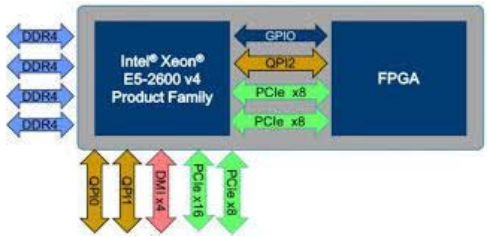
Systems Group, Dept. of Computer Science, ETH Zurich

DINFK

Systems@ETH zürich

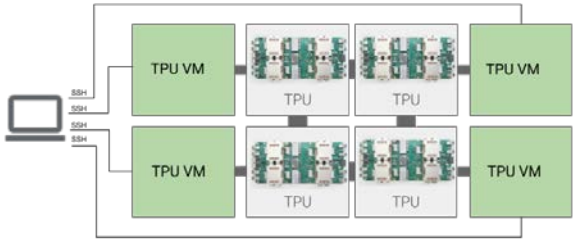
Introduction…

- Scpecialized hardware becoming a reality
  - *Amazon*, *Microsoft*, *Google*, *Alibaba*, *Intel*, *AMD* …

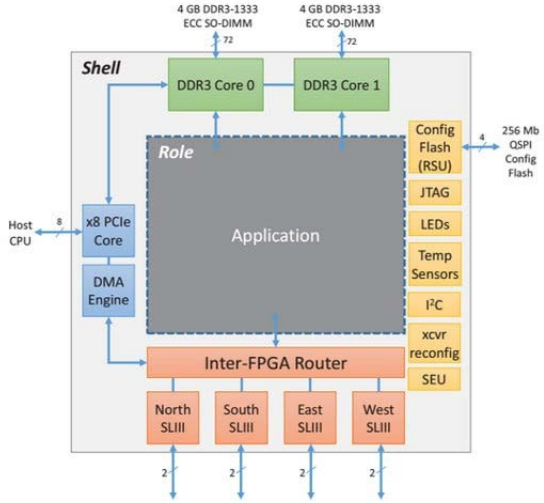


Intel
HARP

AMD
Versal



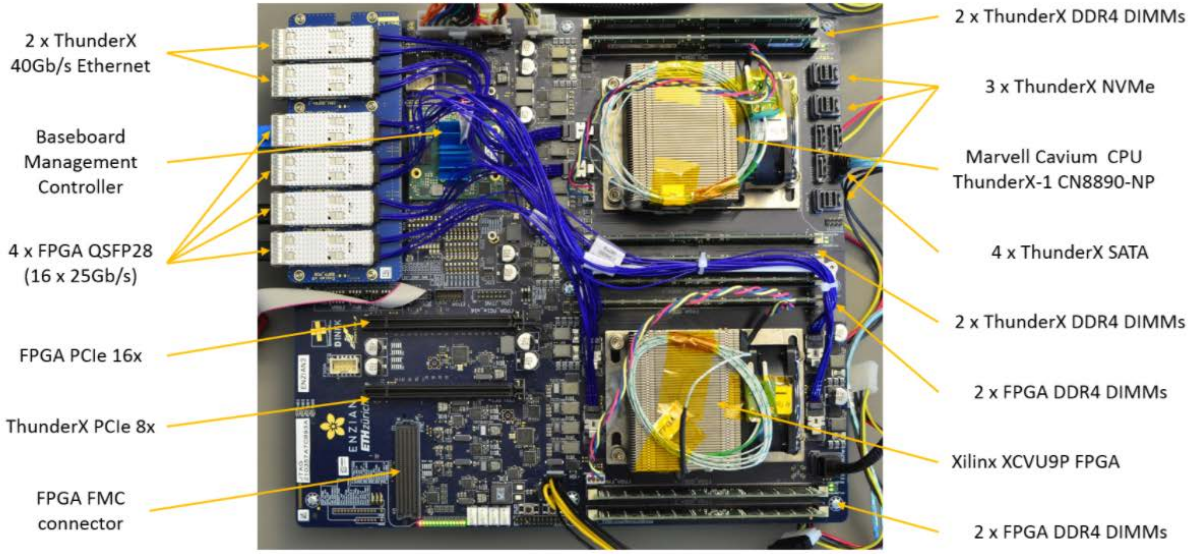
Google
TPU



Microsoft Catapult

- One built within System group at ETH:

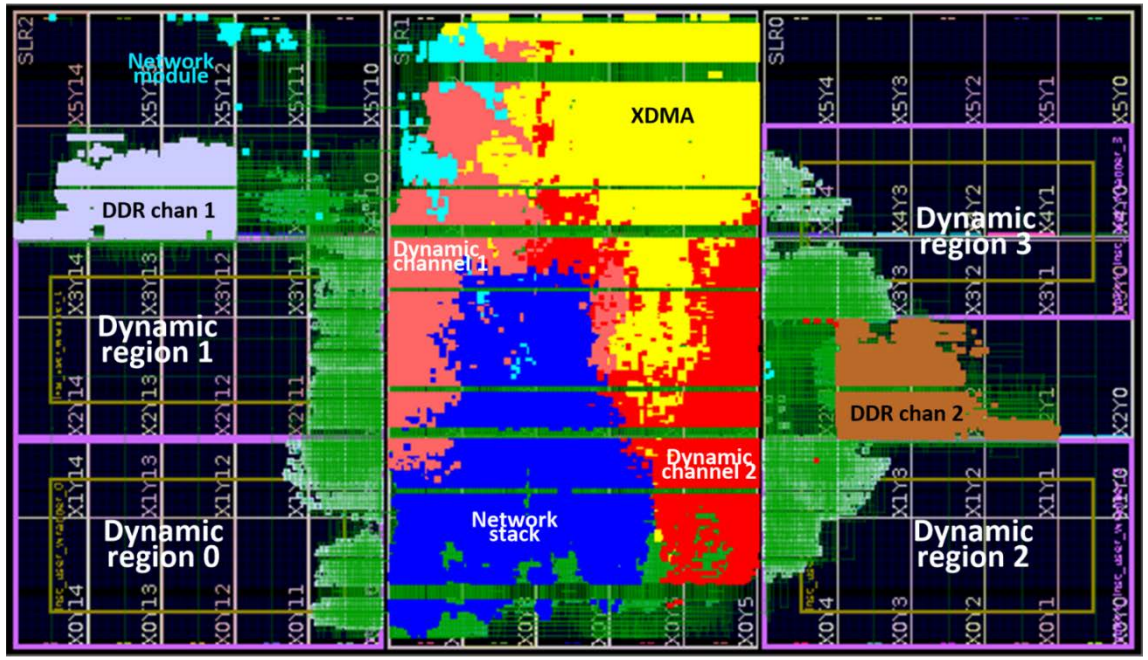
ENZIAN



ETH zürich Systems@ETH zürich D INFK

Complex to program (interact with)

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

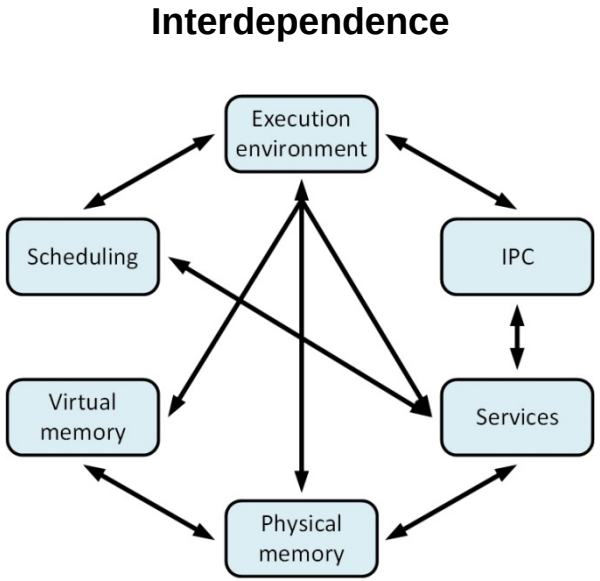


FPGA layout

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




A Hypervisor for Shared-Memory FPGA Platforms

Sharing, Protection, and Compatibility for Reconfigurable Fabric with AMORPHOS

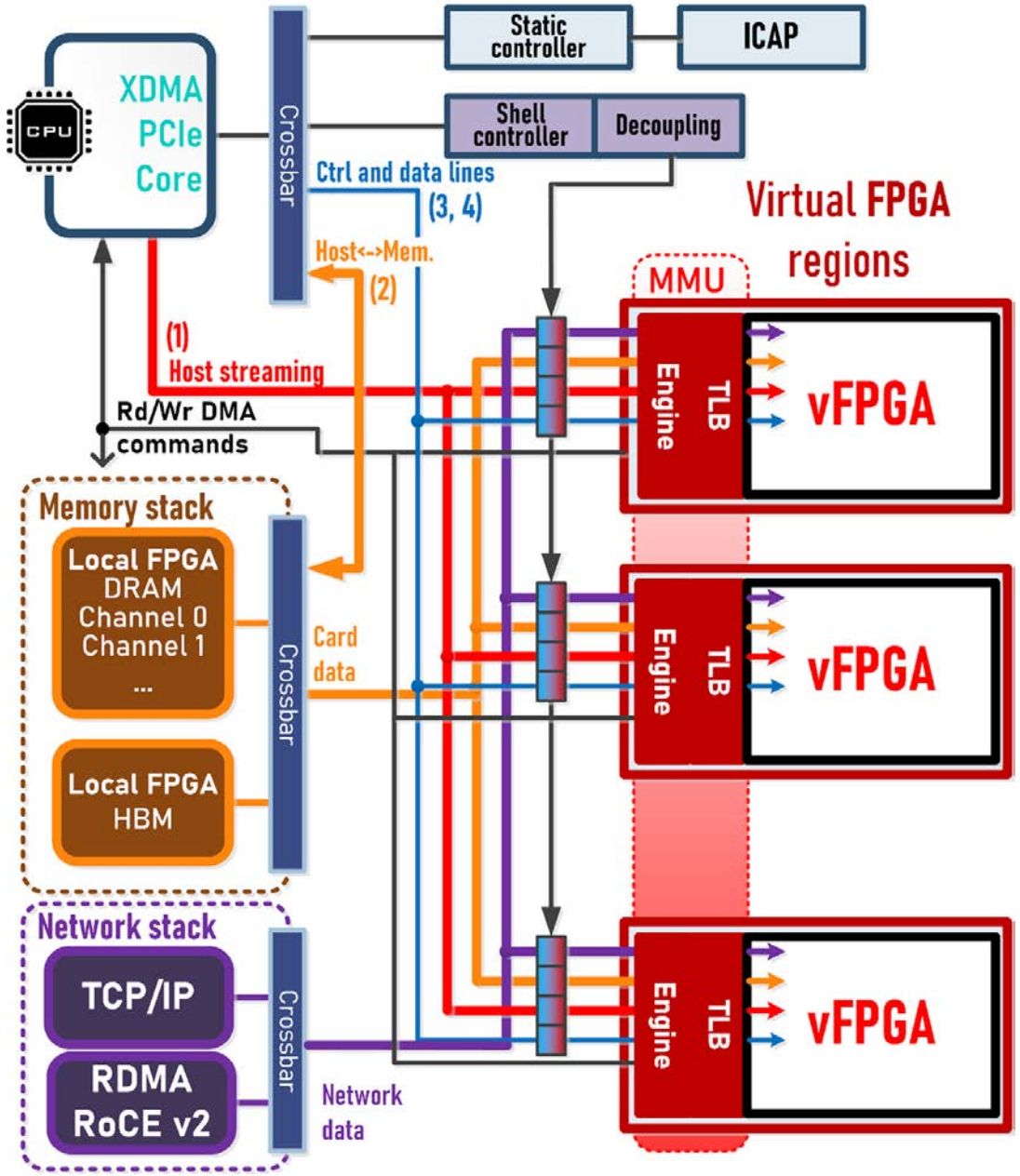
The Feniks FPGA Operating System for Cloud Computing

A Cloud-Scale Acceleration Architecture

Coyote

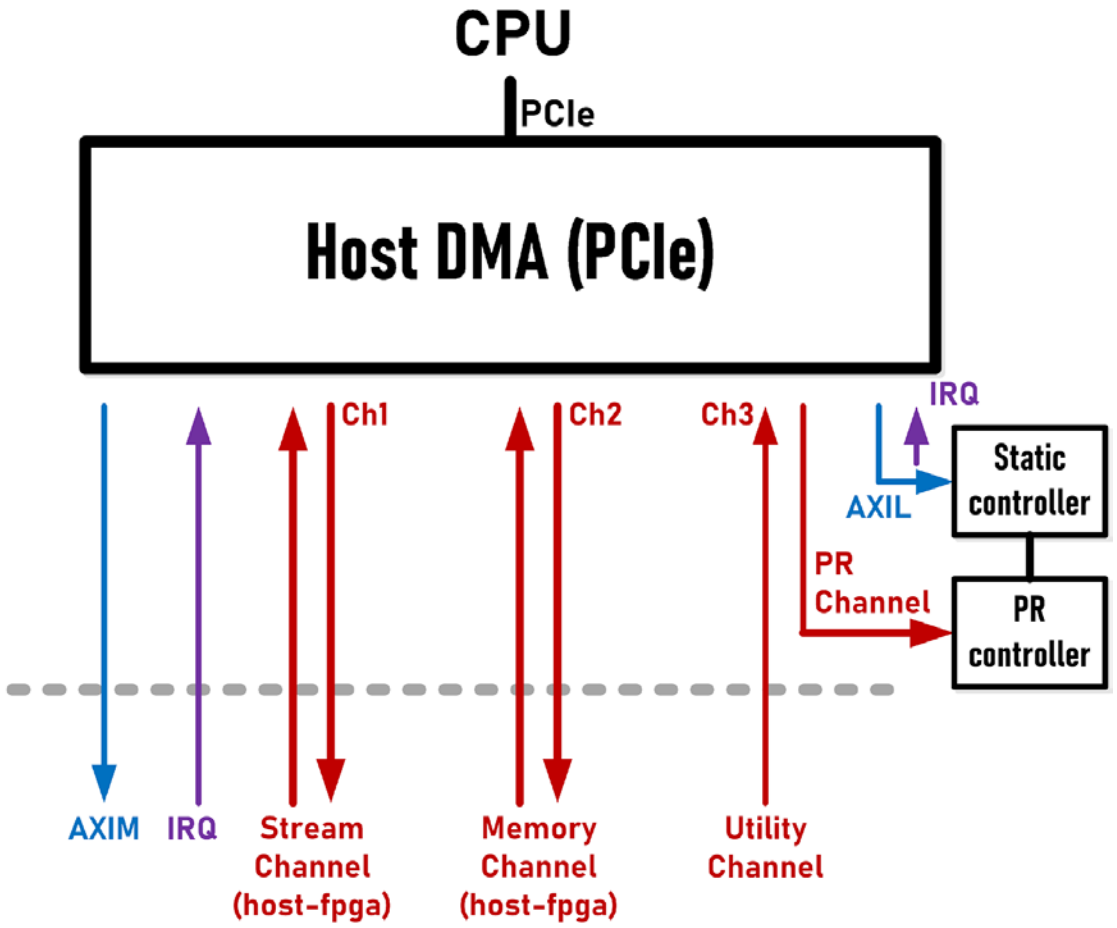
HW Architecture

- 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

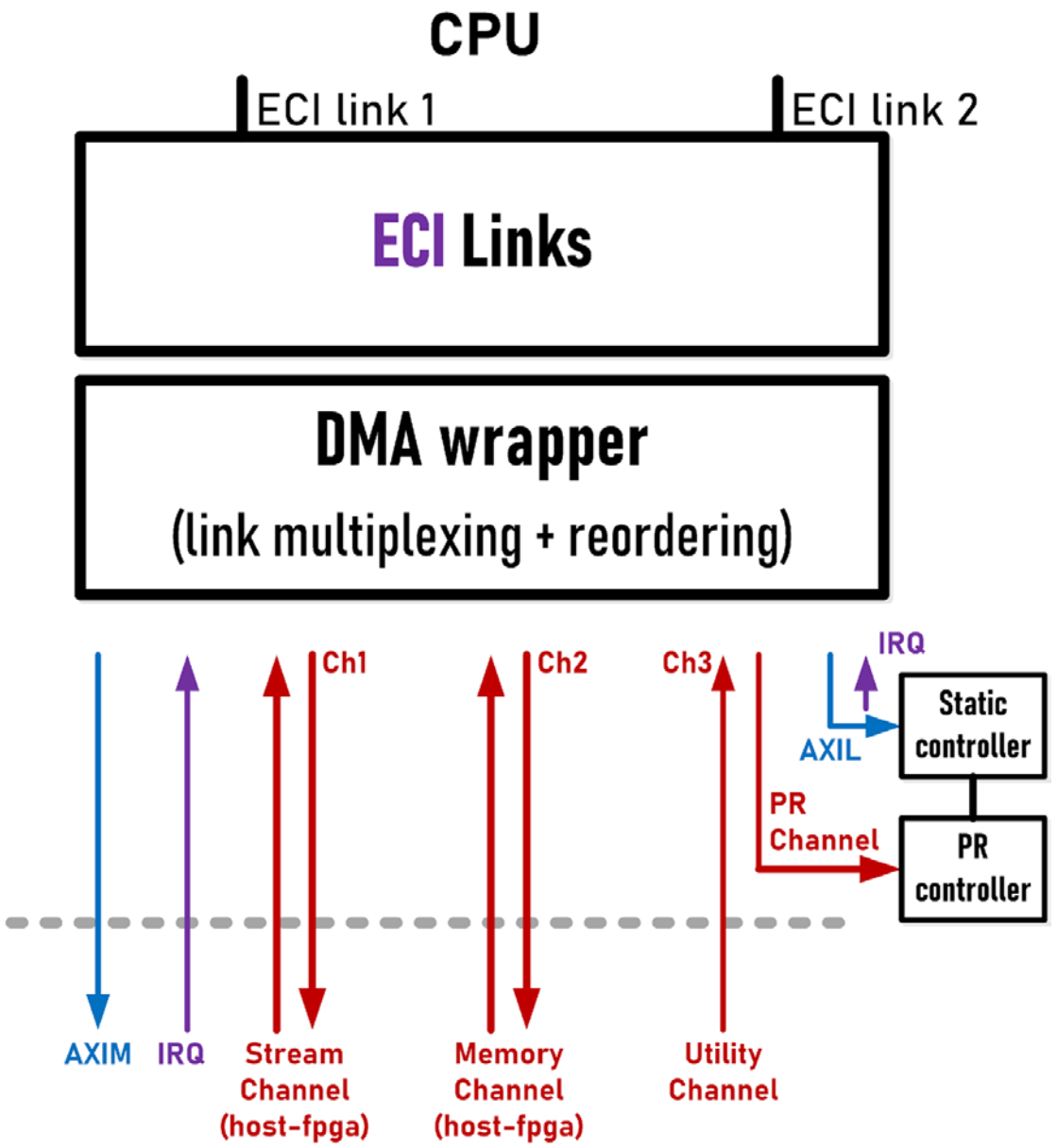
- PCIe connection (cards still at gen3 x16 …)
- XDMA IP[1] core used with Alveo cards (trial runs with QDMA core as well)
- Very little dependency on the actual DMA core used
- Port to open source cores



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

ECI – Enzian Coherency Interface

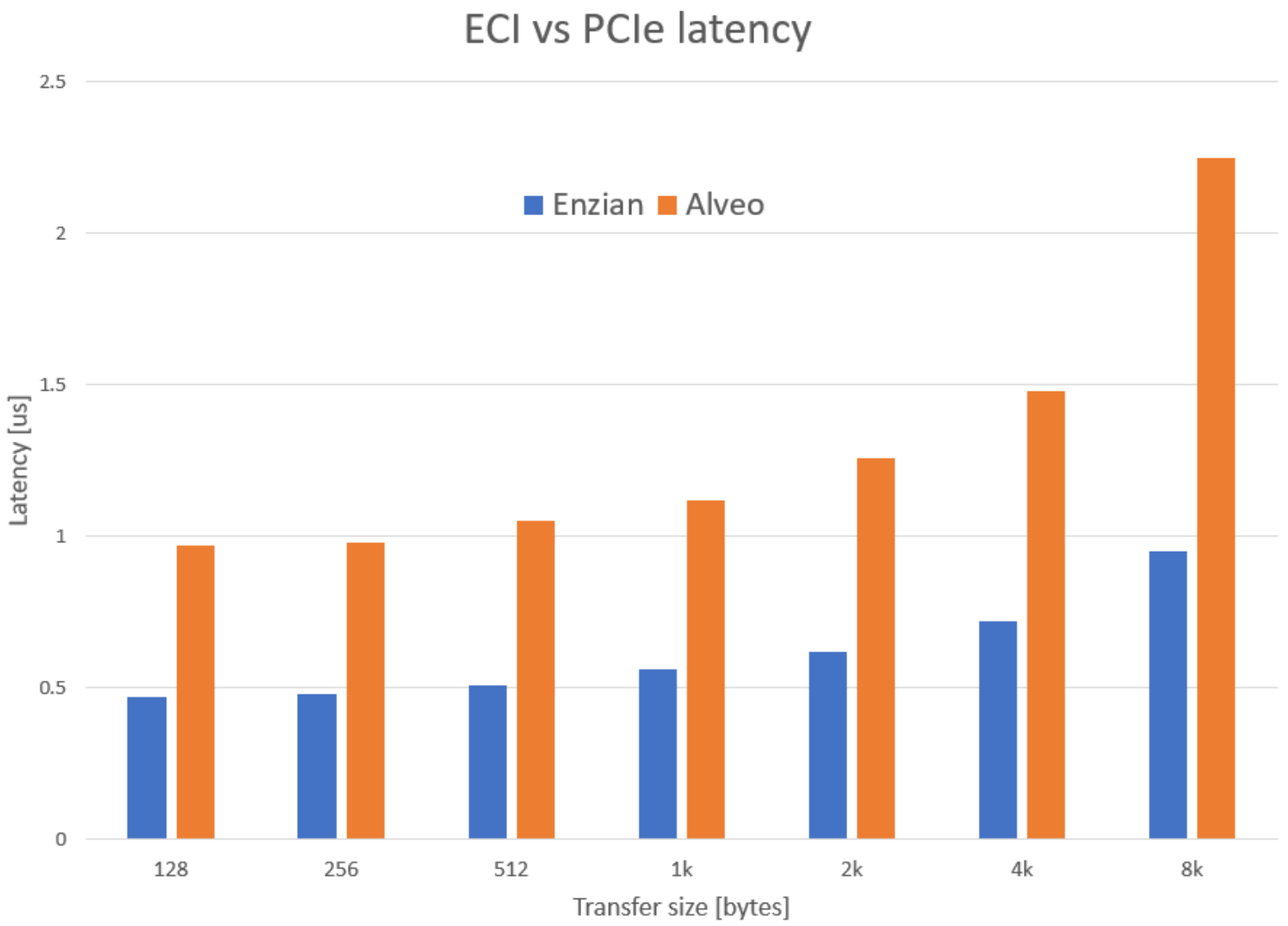
- Cache coherent interface[2]
- Coherency between CPU and FPGA side memories
- Our own DMA wrapper built on top which provides integration for Coyote interfaces



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

ECI vs XDMA

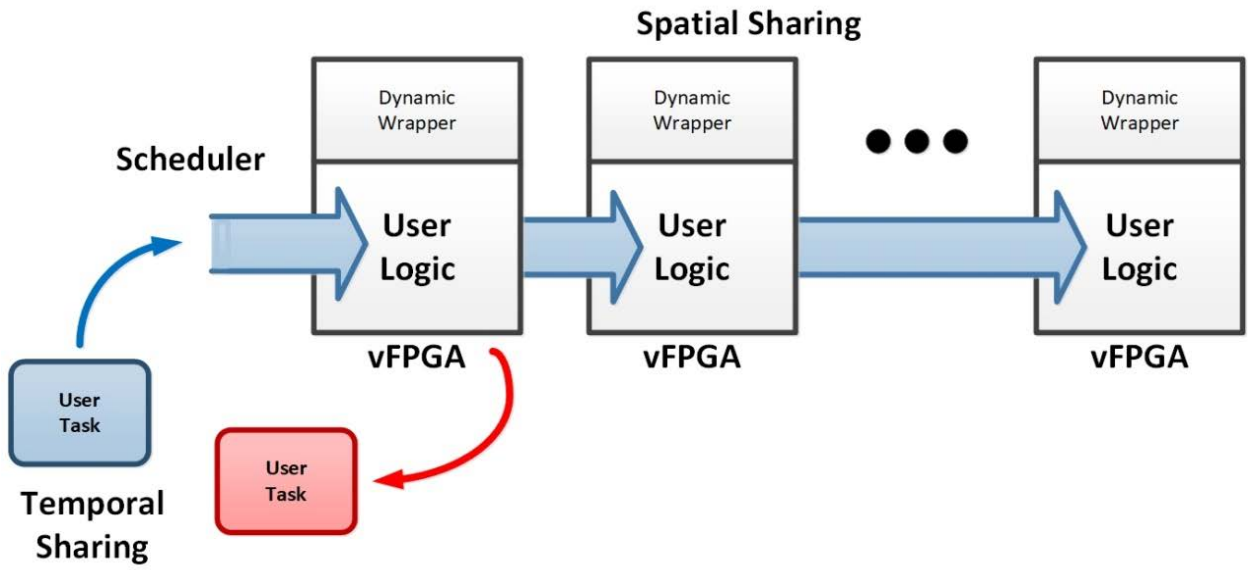
- Throughput PCIe:
  - ~12 GB/s
- Throughput ECI:
  - ~21 GB/s



Coyote

Processes, Threads, Tasks …

- FPGAs are fundamentally different
  - No CPUs or cores
  - Spatial and temporal sharing

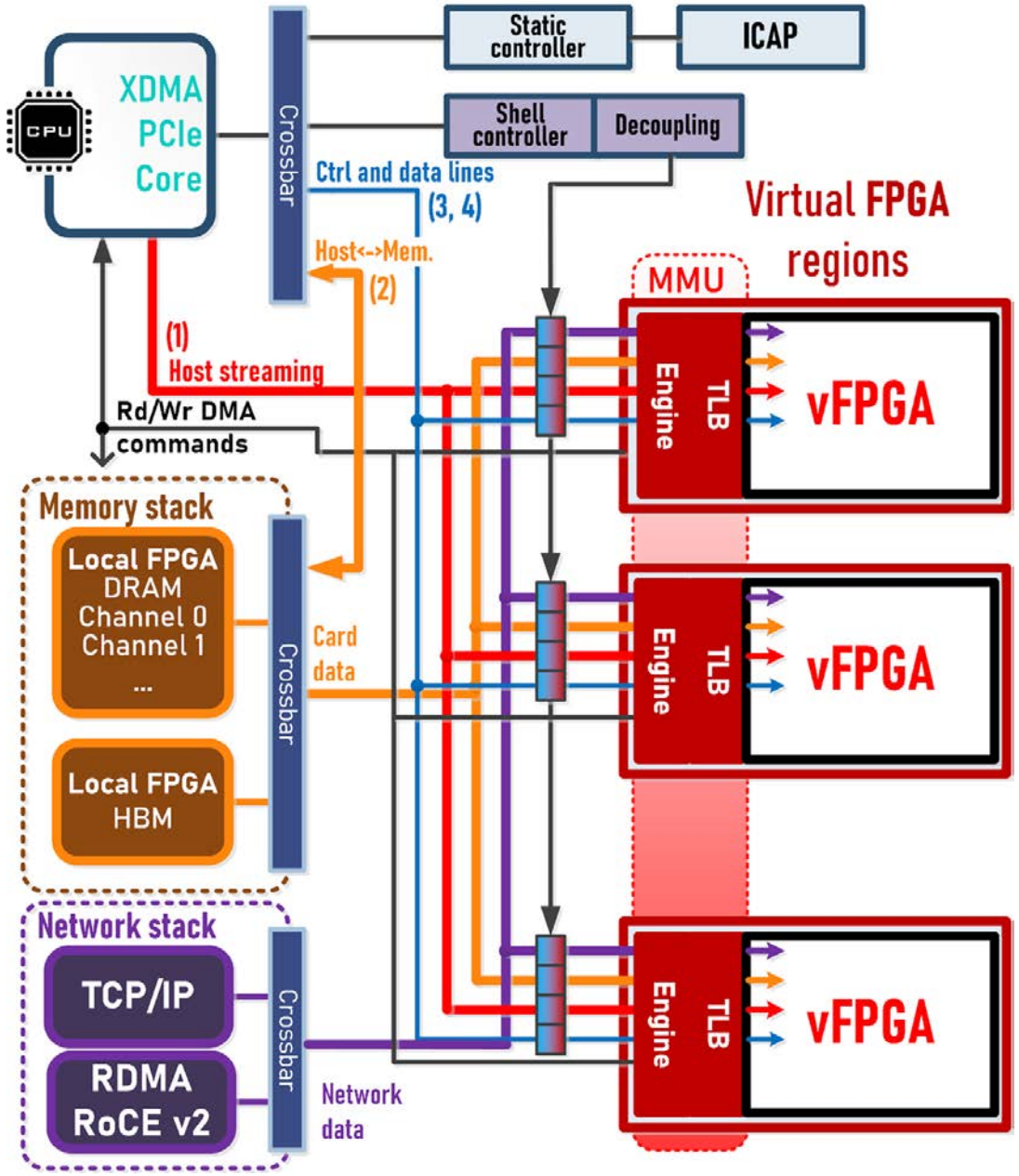


- Coyote multi-tenancy:
  - Combines both approaches
  - Multitasking abstraction for a set of independent, isolated vFPGAs
  - Applications within vFPGAs are untrusted

Coyote

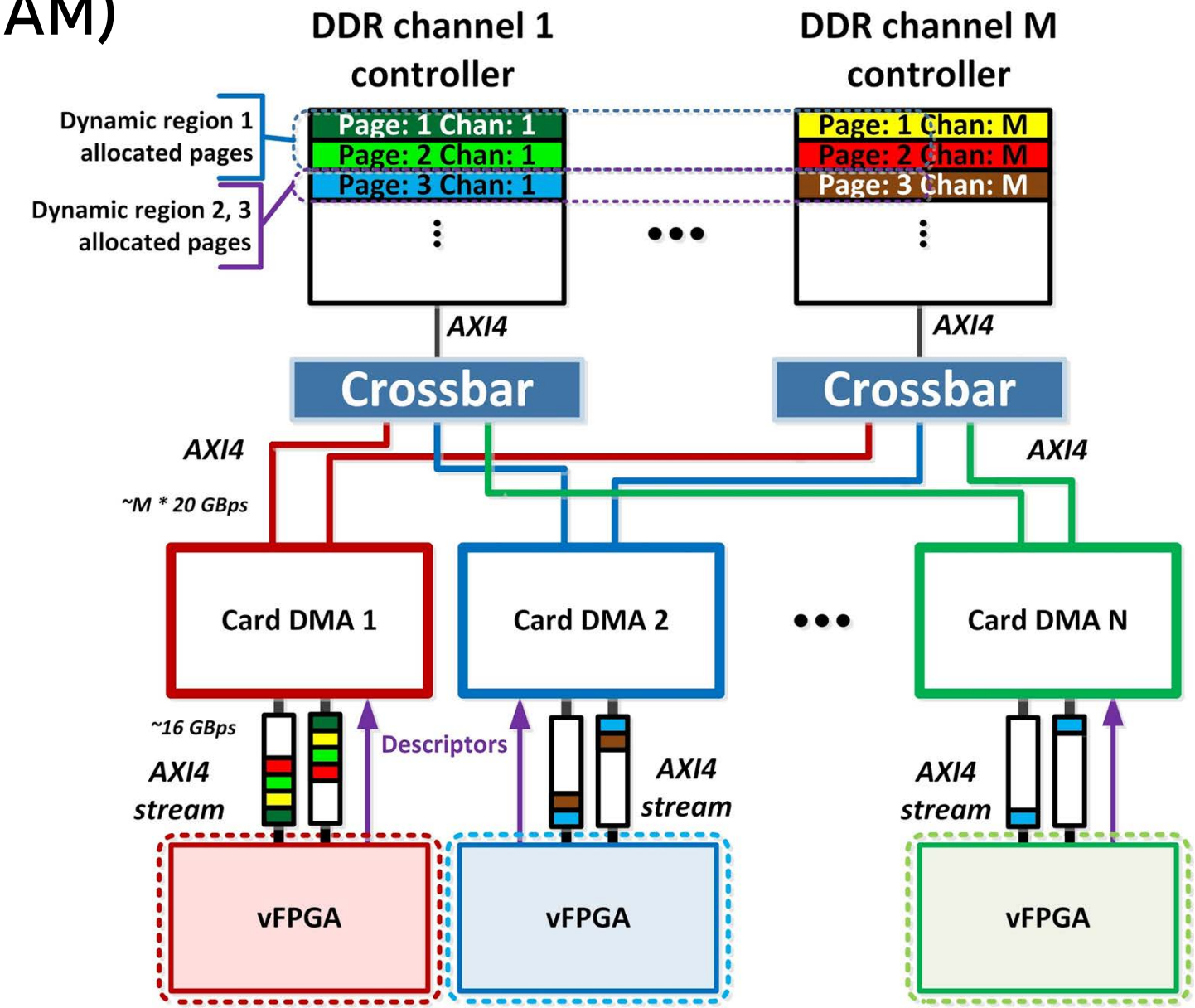
Virtual Memory

- 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



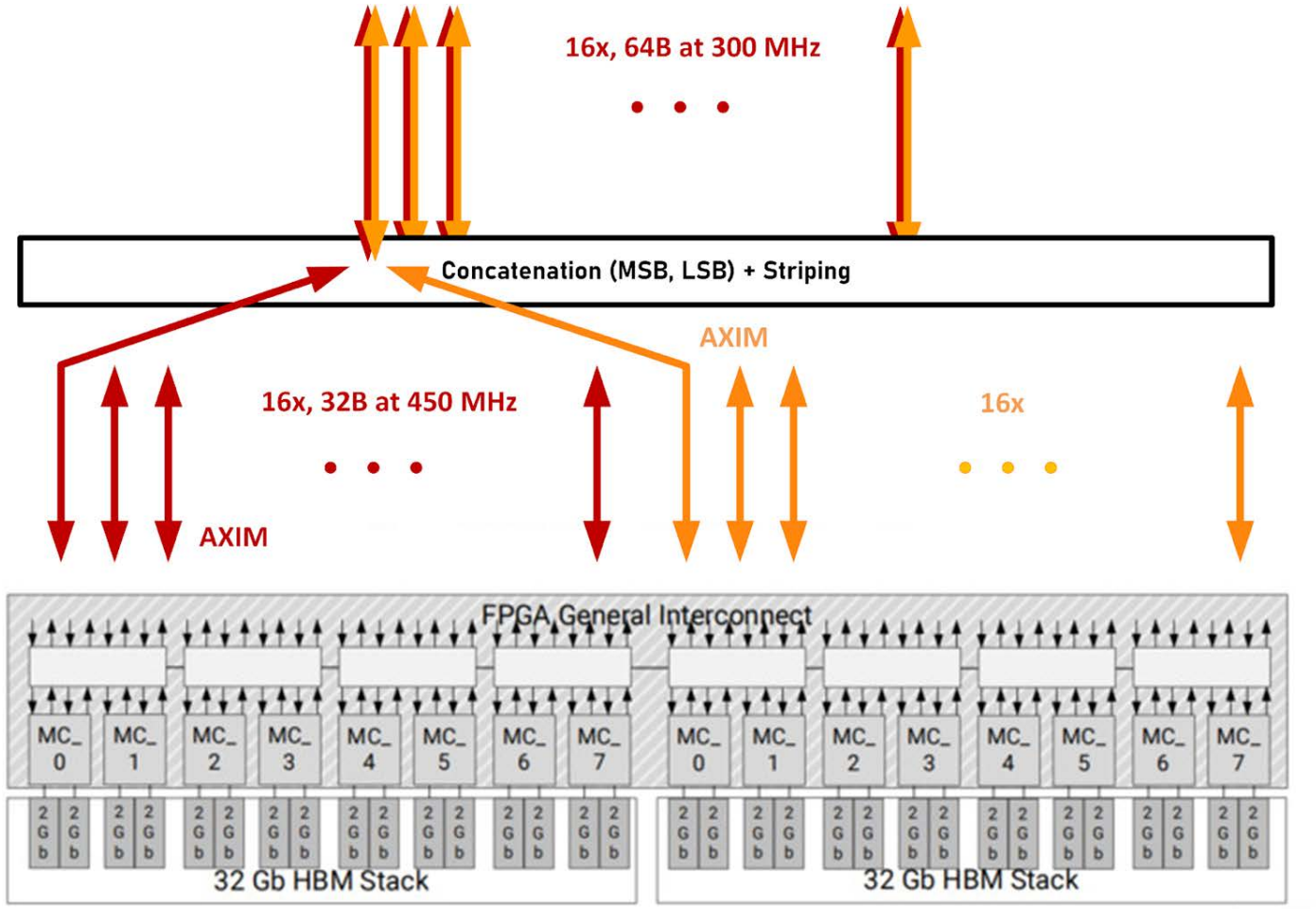
Memory organization (DRAM)

- 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[3] 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

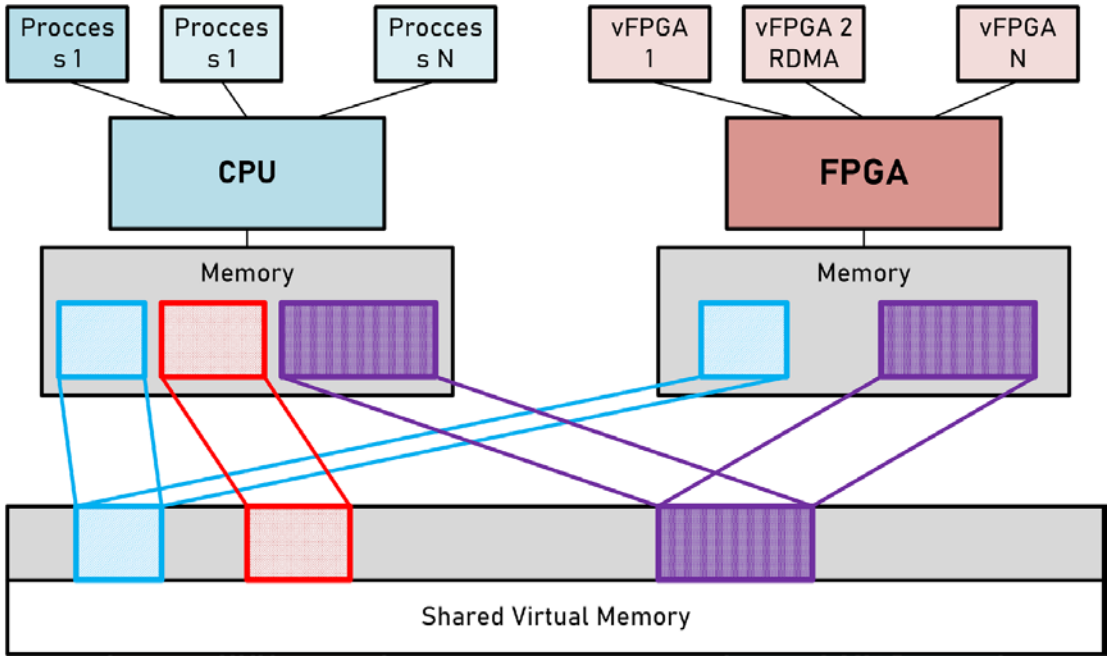


[3]: https://www.xilinx.com/products/intellectual-property/rama.html

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

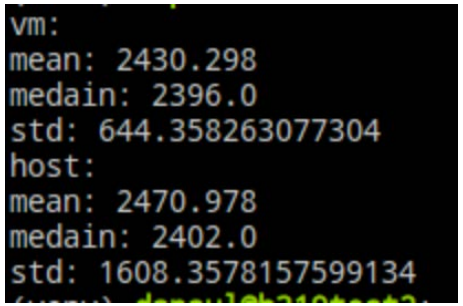


Unified Memory Example

// NO UNIFIED MEMORY

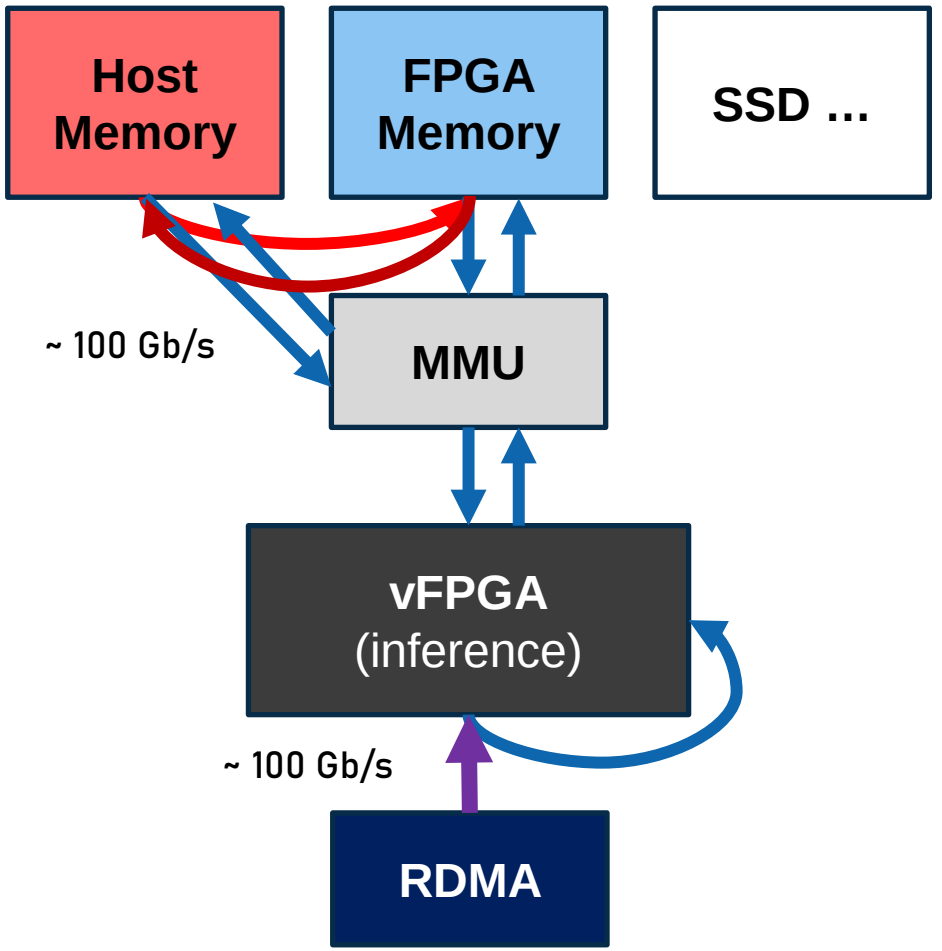
// WITH UNIFIED MEMORY

IRQ handling overhead:

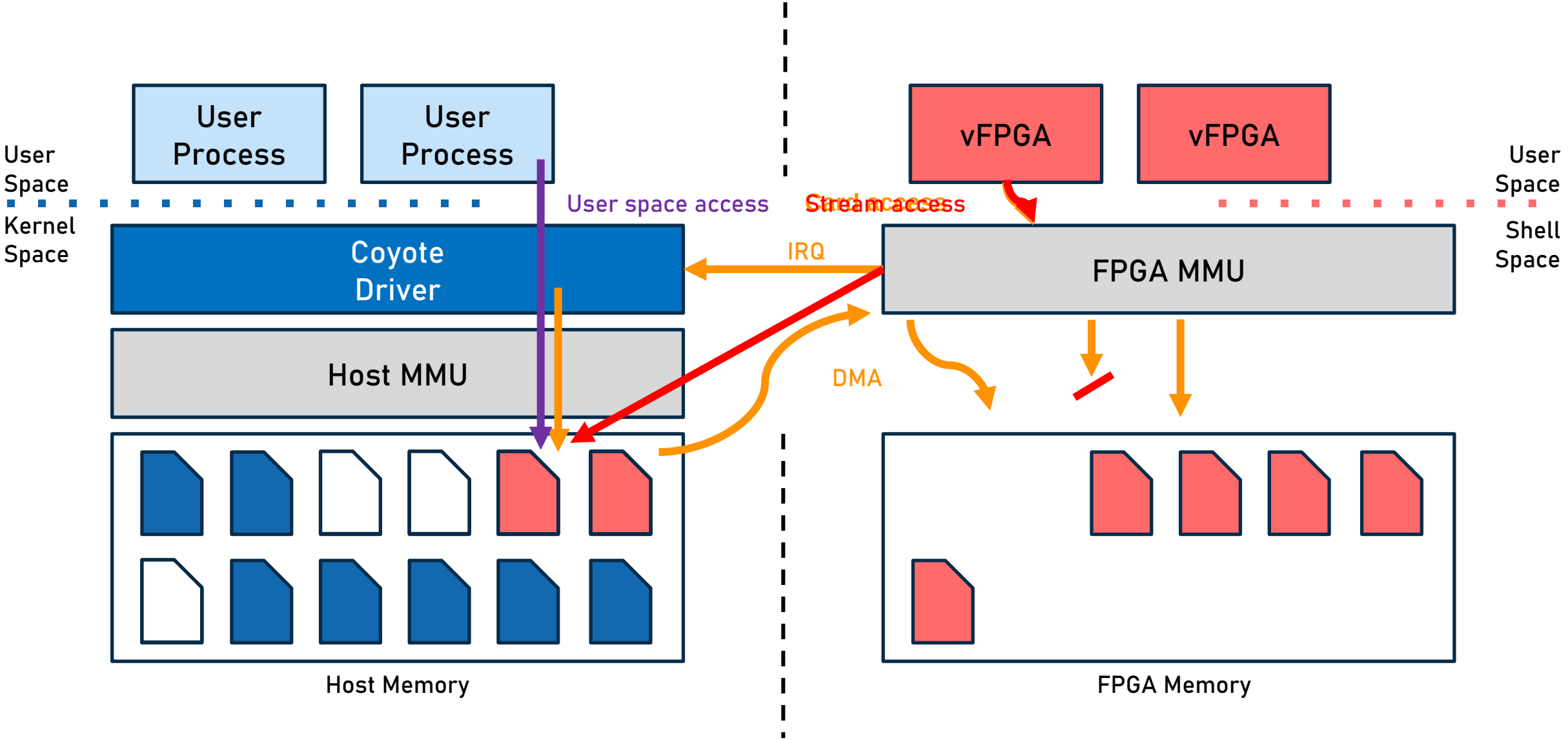


FPGAs are not GPUs …

- 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)?

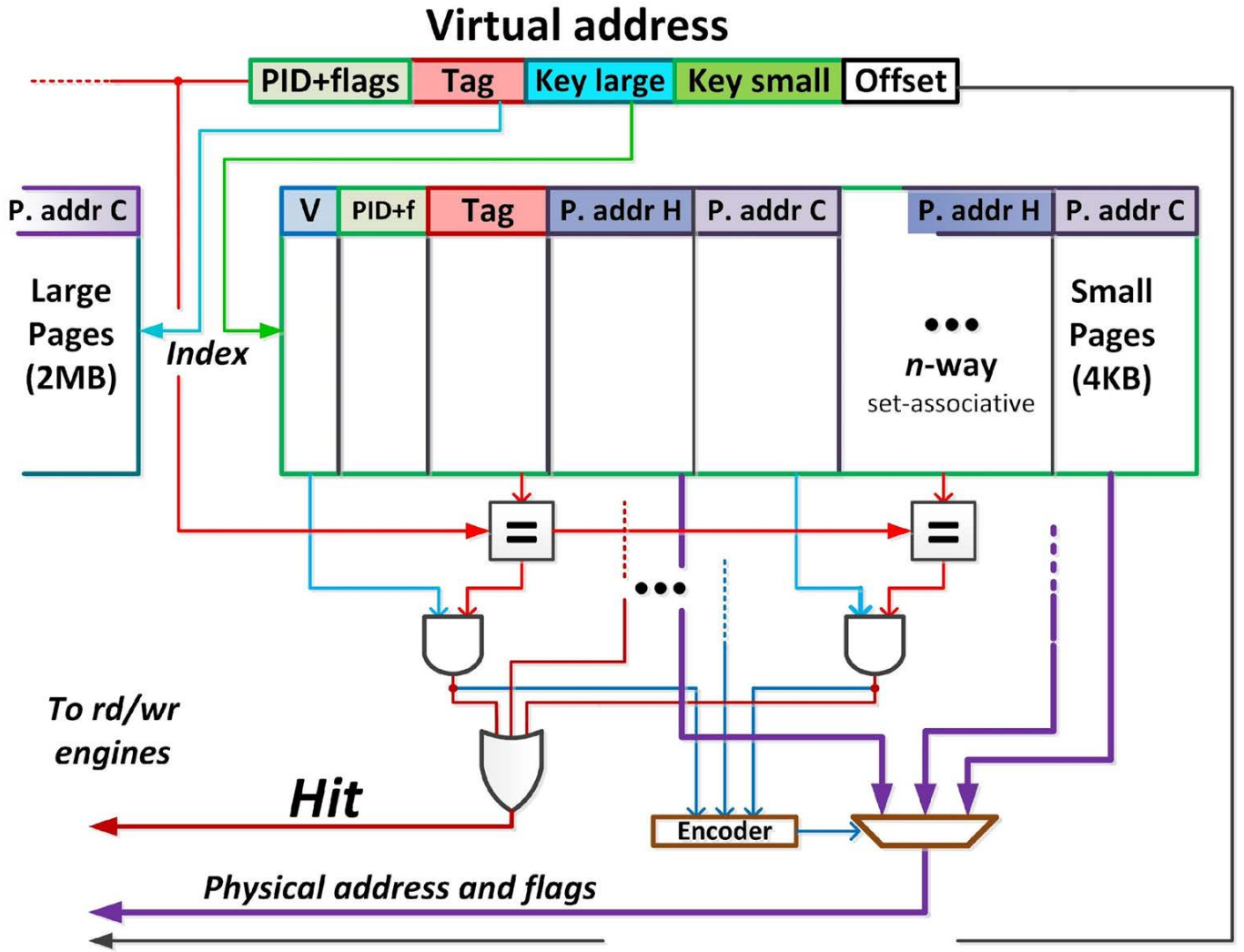


HMM operation



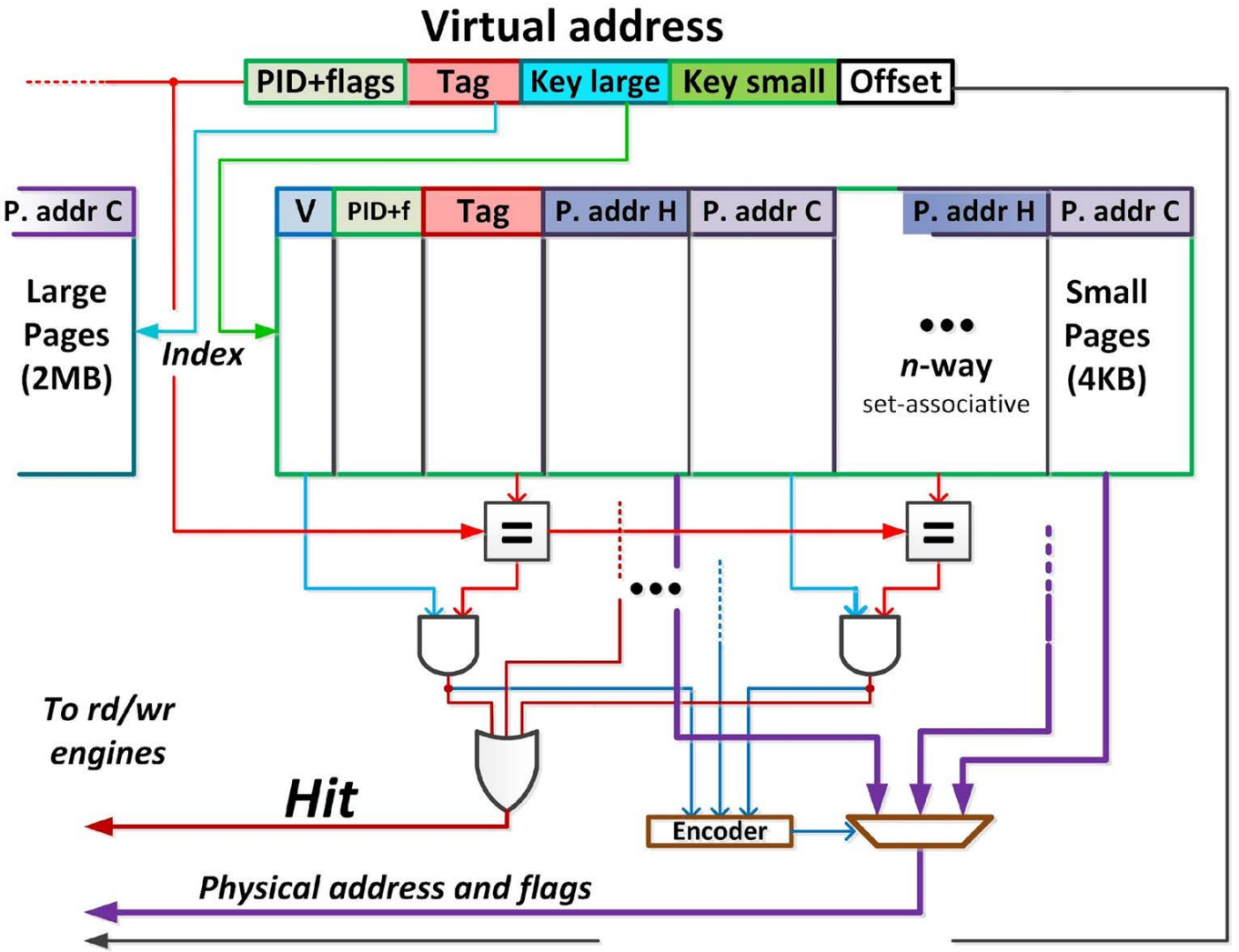
MMU Notifiers

- 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 …



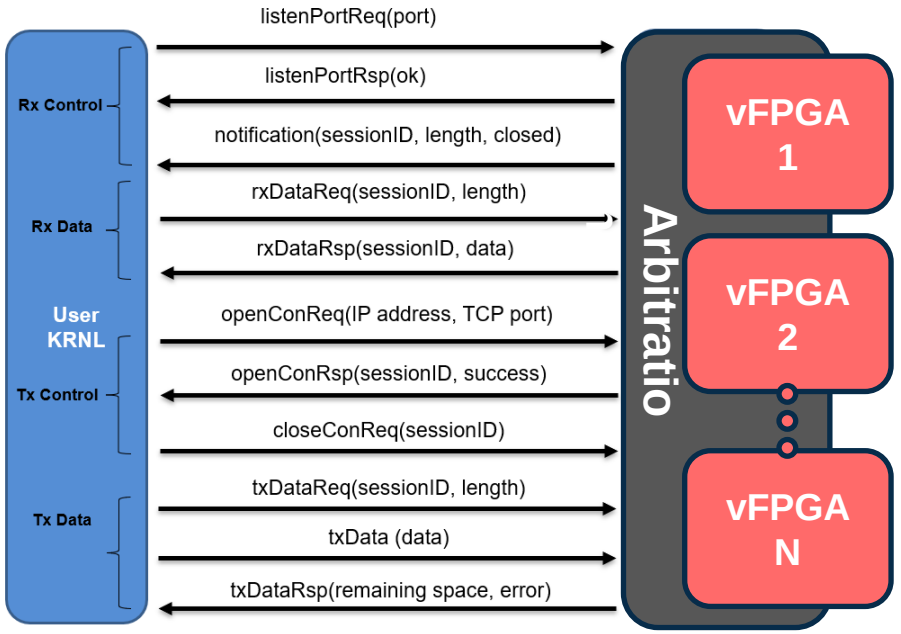
Coyote
Network Stacks

- 100G TCP/IP and RDMA (RoCE v2) network stacks[4]
- 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

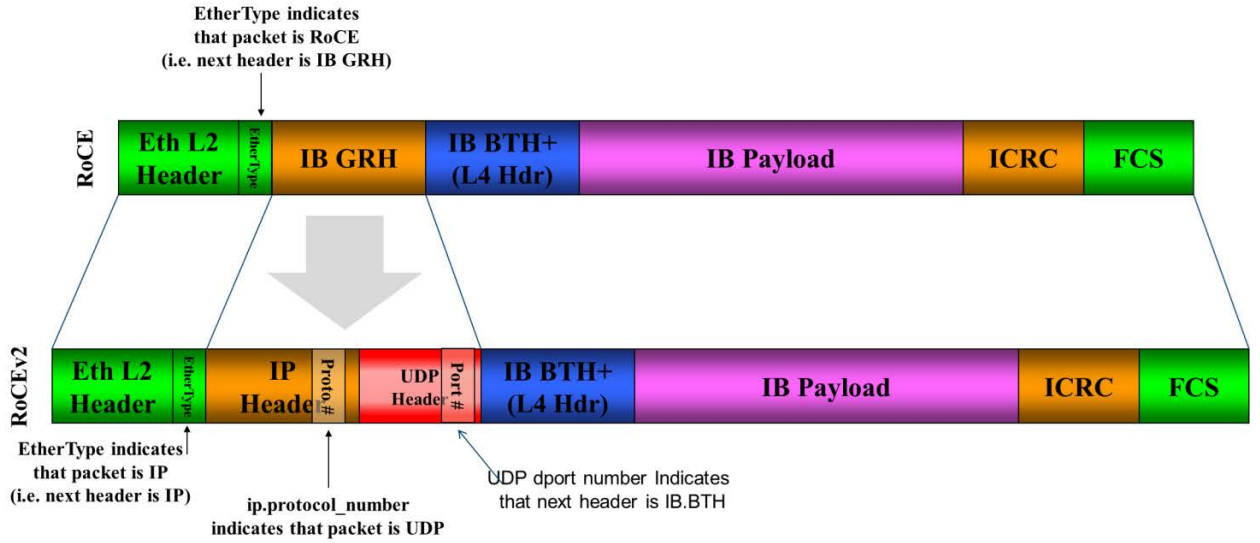
- Shared across vFPGAs
- Multiple accelerators serving clients on different ports
- AXI stream interface (opening and closing ports and connections,
- Lower level of abstraction than RDMA !



[5]: https://github.com/fpgasystems/Vitis_with_100Gbps_TCP-IP/blob/vitis_2022_1/img/interface.png

RDMA stack

- Open source RDMA stack (UC, RC)[5]
- 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

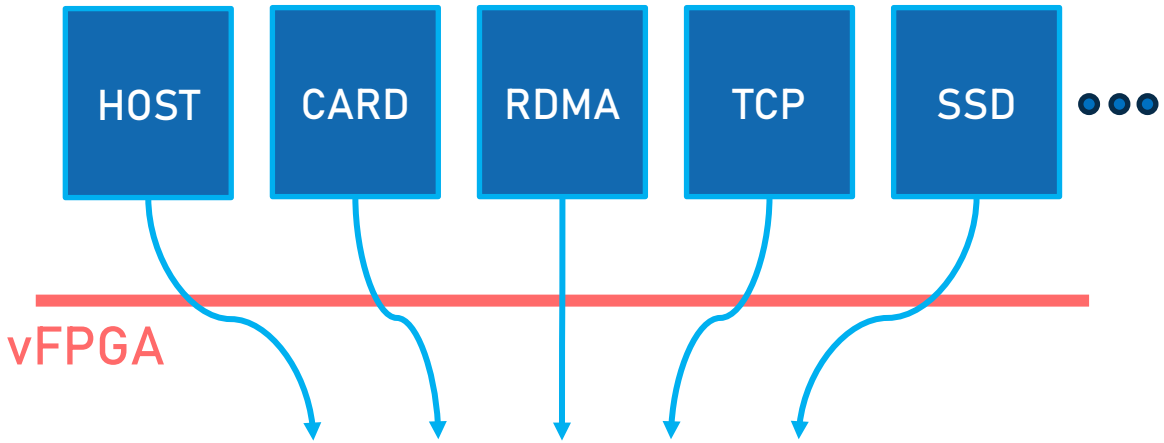
[5] ***StRoM**: Smart Remote Memory, EuroSys '20*
David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, Gustavo Alonso

Coyote

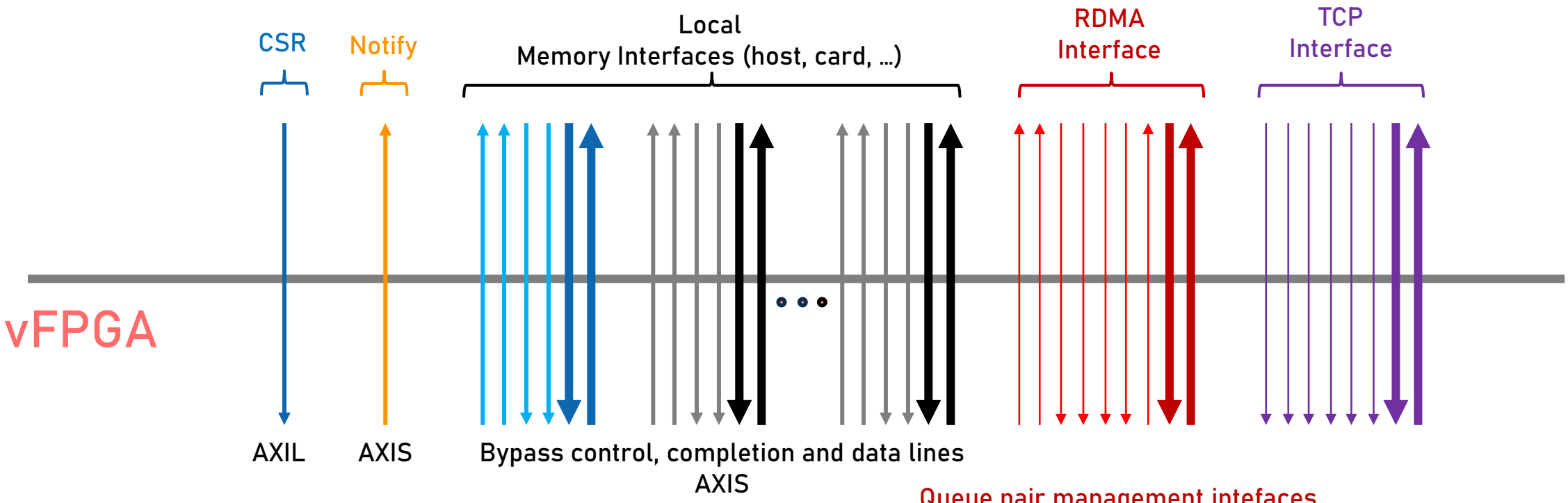
Unified Logic Interface

- Don't really exist across FPGA platforms …
  - Vitis HLS
  - Much higher flexibility than other devices

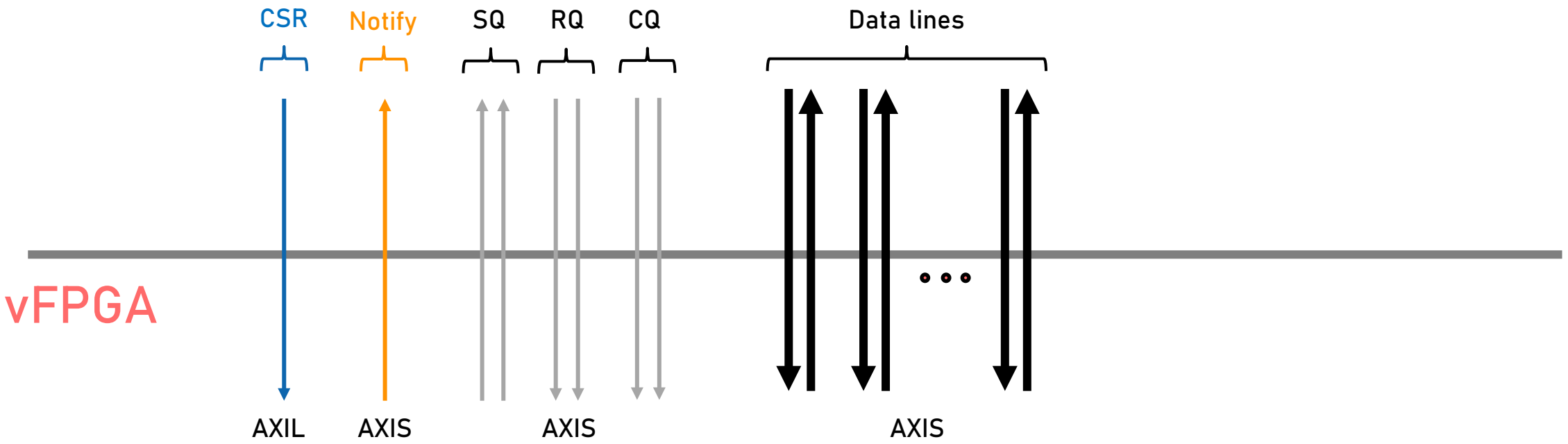
- User Logic Interface
  - AXI streams (descriptor interface)



Unified Logic Interface



Can this be simplified?

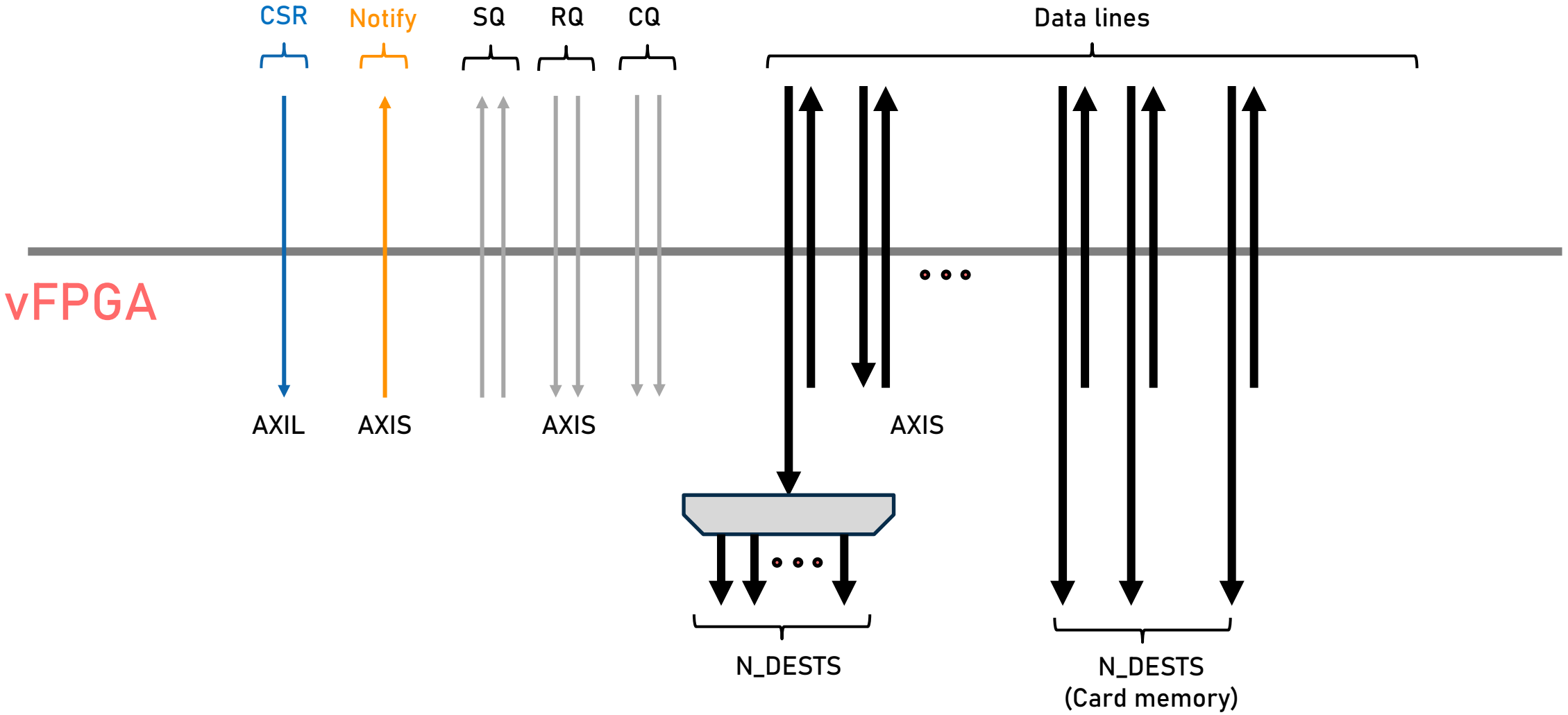


SQ Interface

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

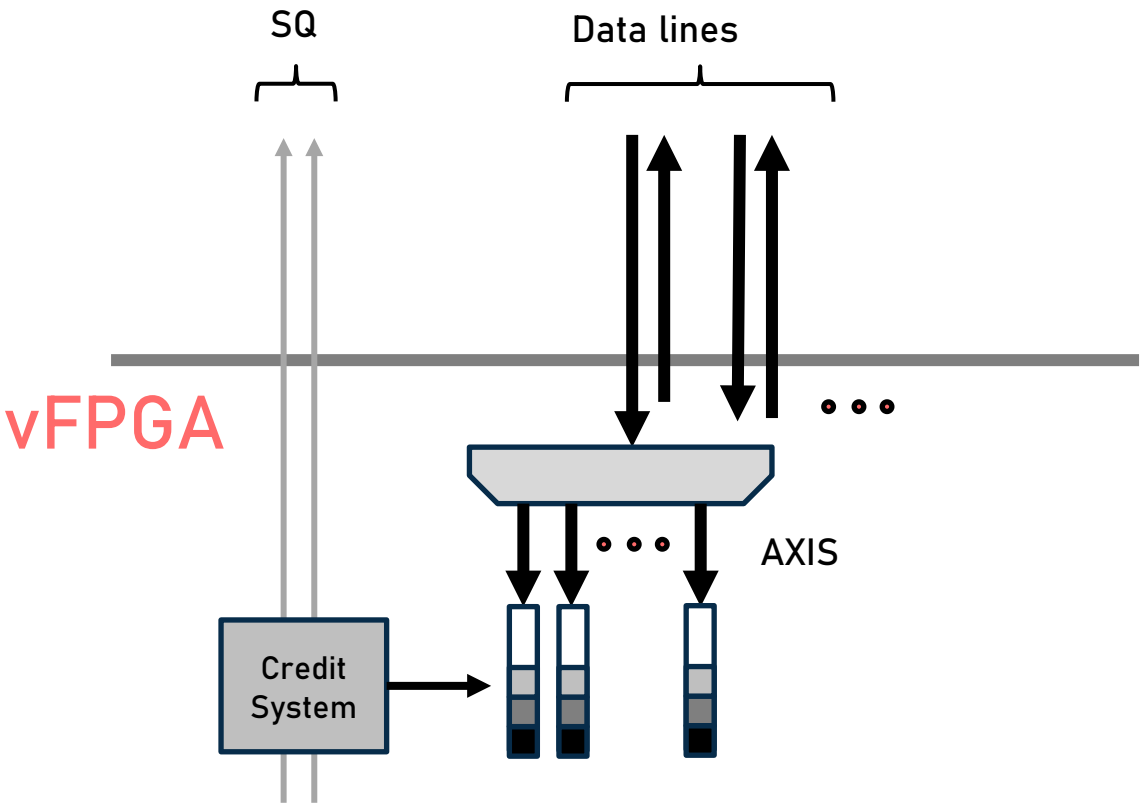
  - OPCODE – Operation to be invoked
  - CPID – Host process identifier
  - VFID – vFPGA (reserved)
  - DEST – Destination queue
  - SID – Connection ID
  - VADDR – User space shared pointer
  - LEN – Transfer size
- RQ – Received remote requests (RDMA and TCP/IP)
- CQ – Completion events

Destination queues



Untrusted environment

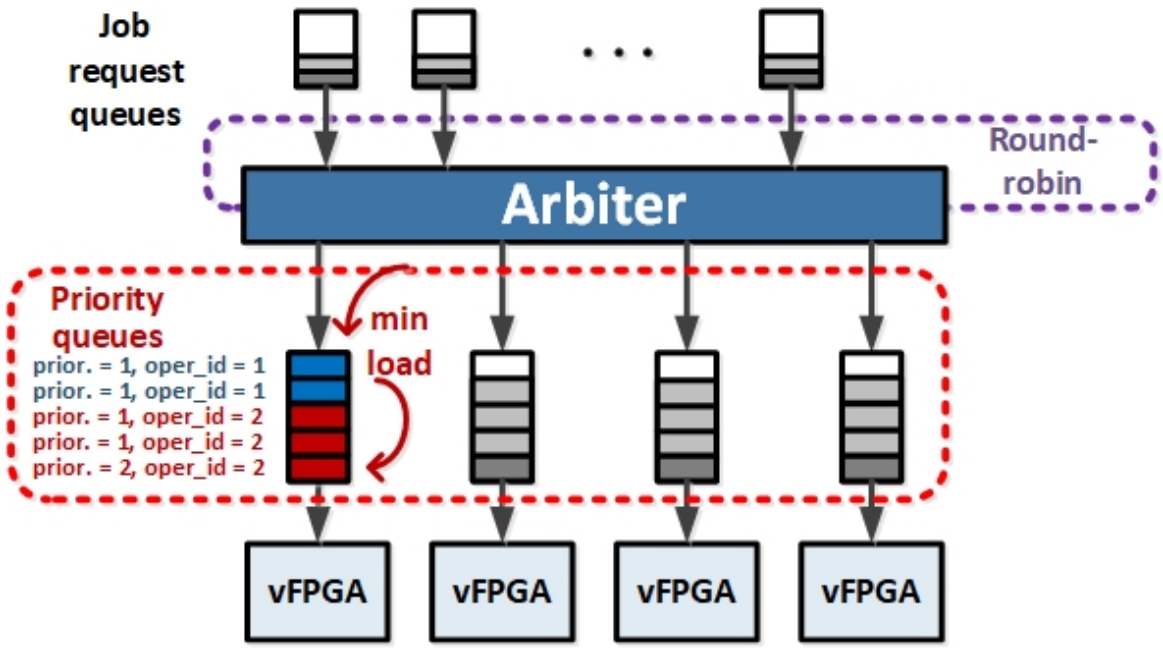
- Credit system for each destination queue:
  - Write requests are issued only if accompanying data is provided
  - Read requests are issued only if queues are ready to accept the data
- Credit system for all local and remote requests
- Can quickly overwhelm the resources …
- For RQ, RDMA accepts packets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



Coyote

Dynamic Reconfiguration and Scheduling

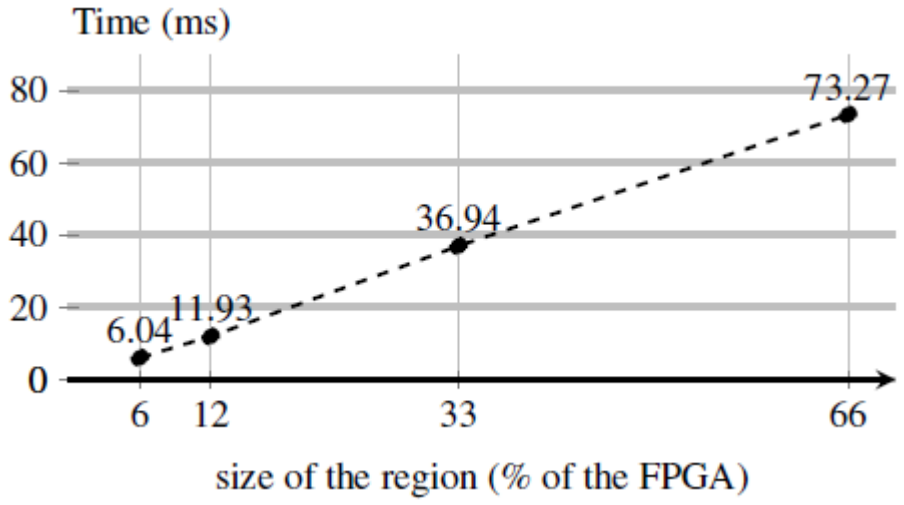
- Basic mechanisms to capture the state of the FPGA don't exist
- Non-preemptive task based approach
- Preemption?
  - User application trust
  - Requires a form of cooperation
  - Additional application complexity



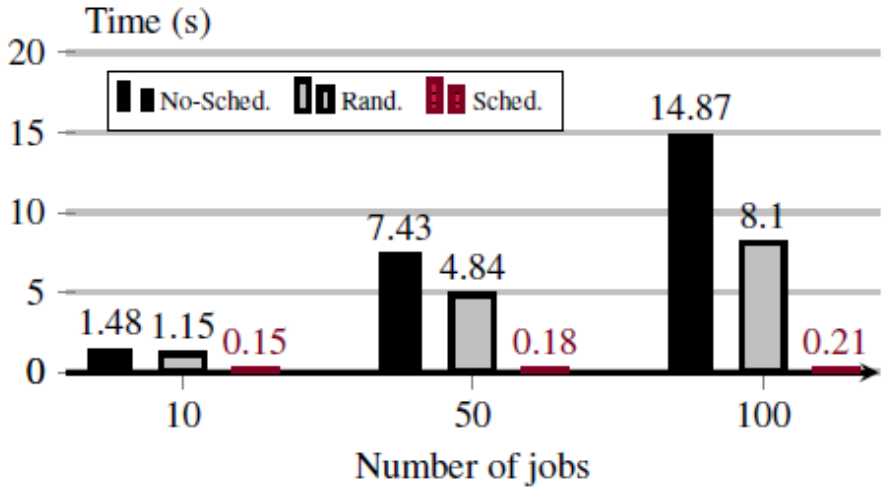
Reconfiguration overhead and scheduling efficiency

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

Reconfiguration times

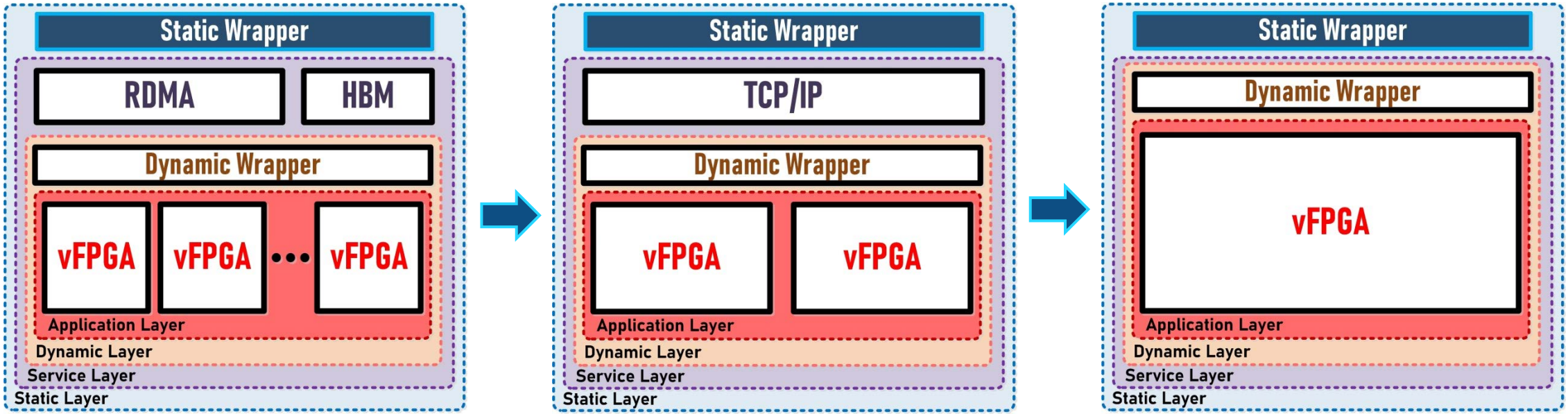


Scheduling algorithm



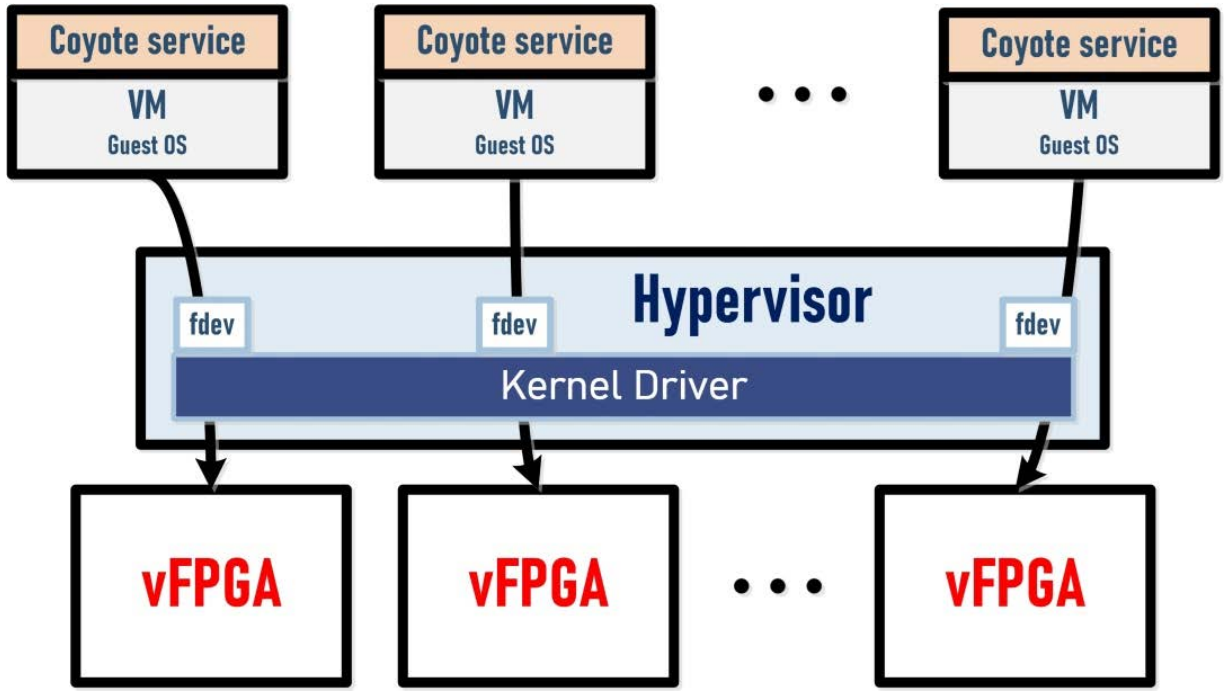
Hierarchical Reconfiguration

- Nested reconfiguration (*Nested-DFX*)
  - Static layer
  - Service layer
  - Dynamic layer
  - Application layer



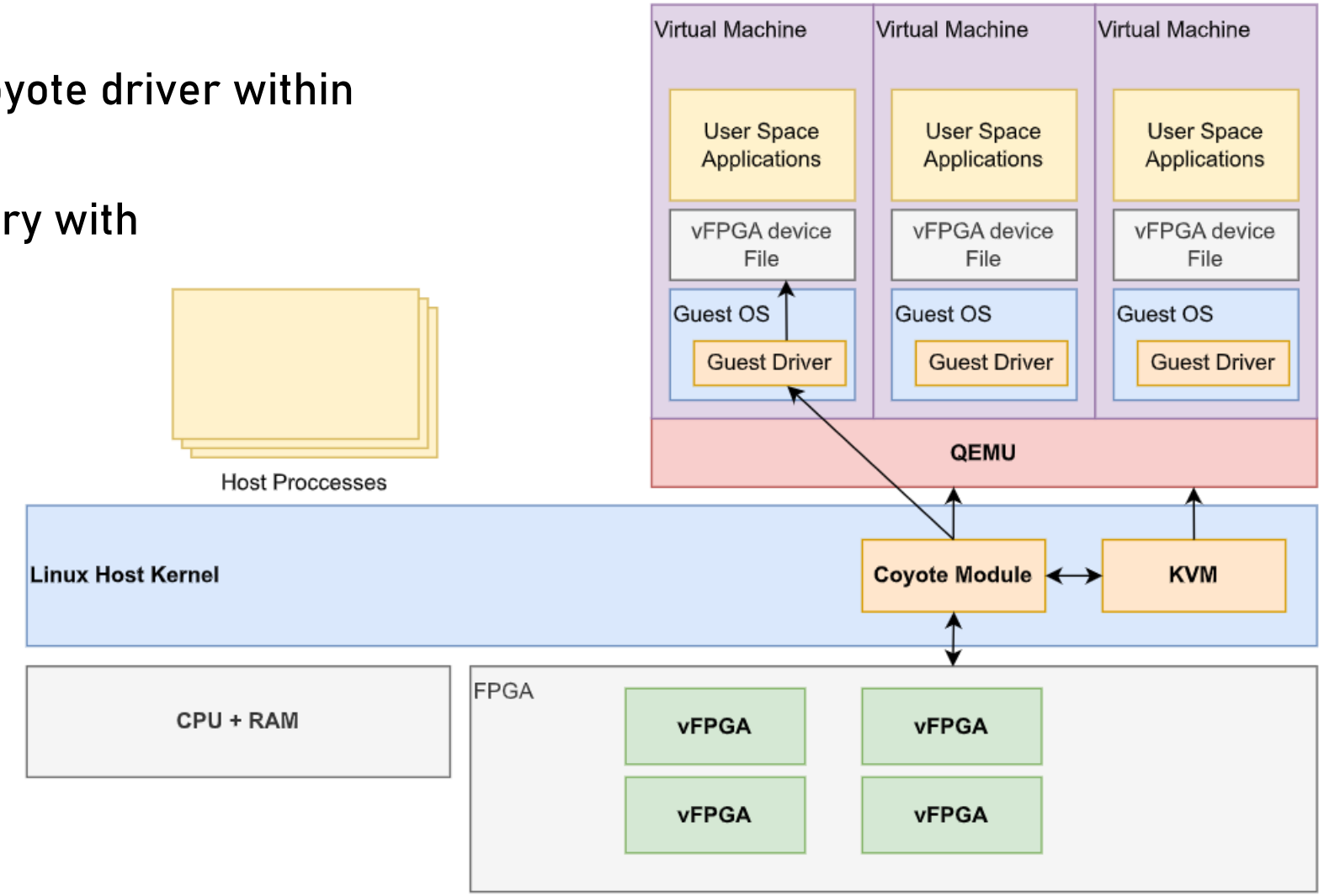
Coyote
Virtual Machines

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

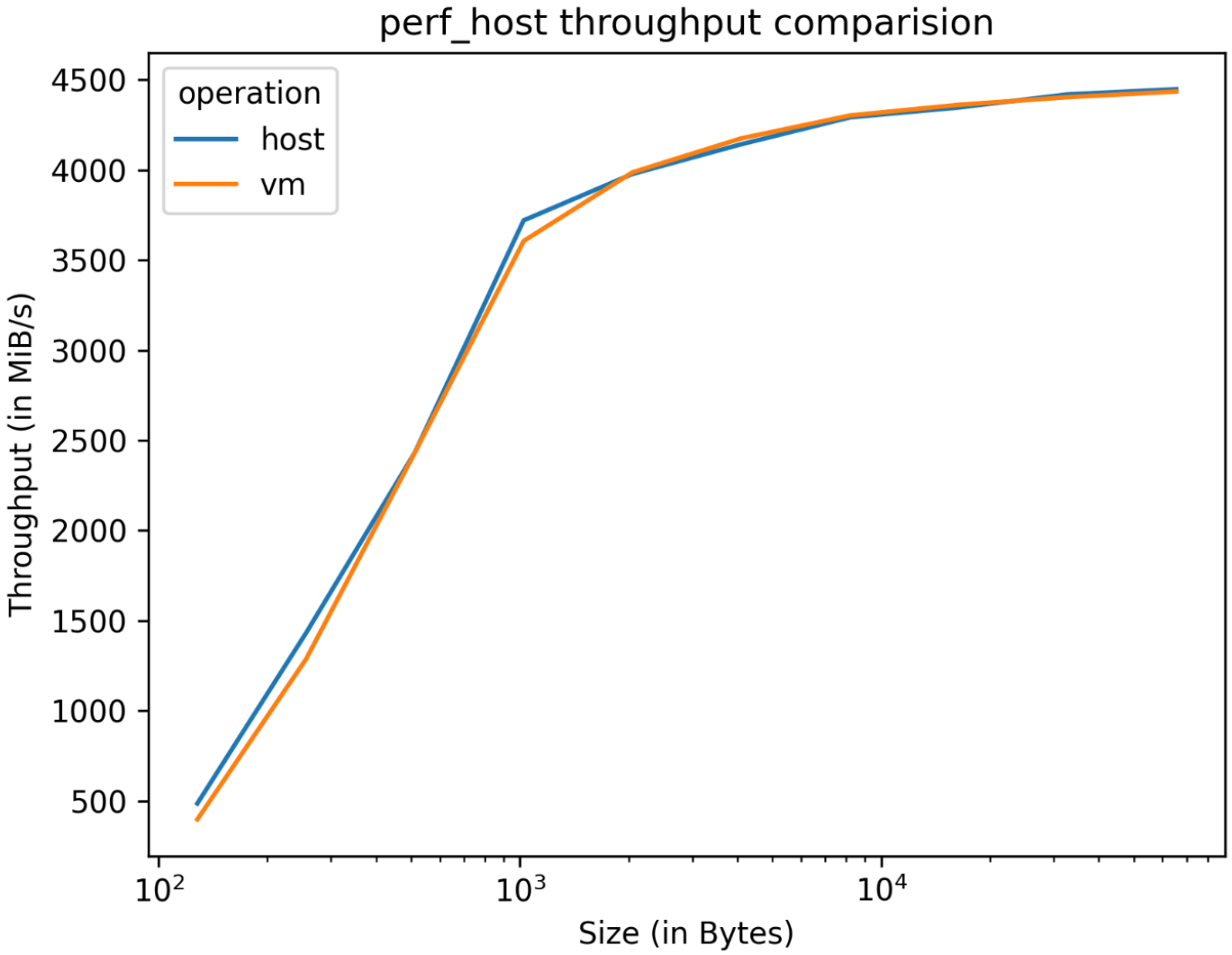


Virtualization Architecture

- 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

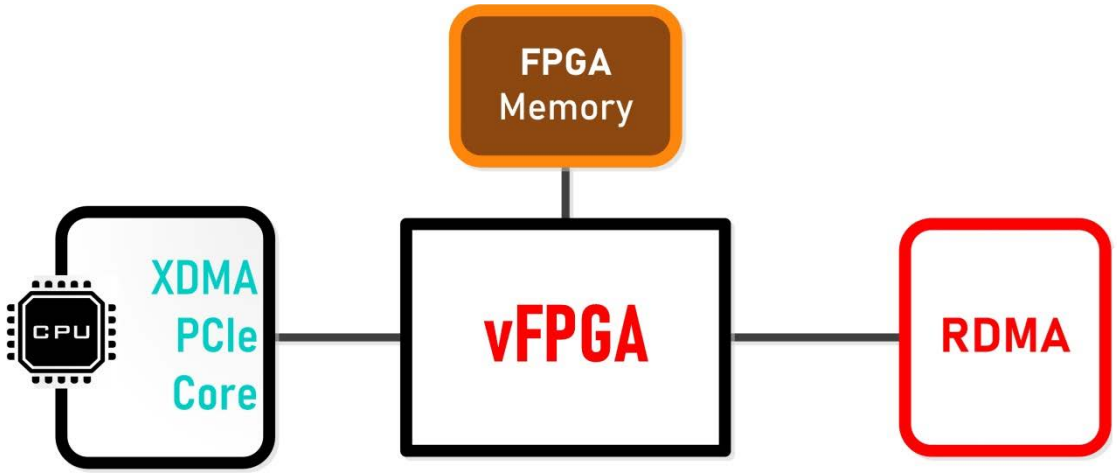
- CMake incremental builds
  - Revamped for hierarchical flow
- Cores can be both RTL and HLS
  - hw/aes
    - cyt_user_top.sv
    - hdl
      - *.sv, *.svh, *.vhd
    - hls
      - *.cpp, *.hpp
- make shell (shell partial bitstream)
- make app (application partial bitstreams)

```
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?