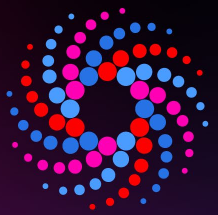


SC22
Dallas, TX | hpc accelerates.

Enabling VirtIO Driver Support on FPGAs

Sahan Bandara Zaid Tahir Martin Herbordt
Boston University

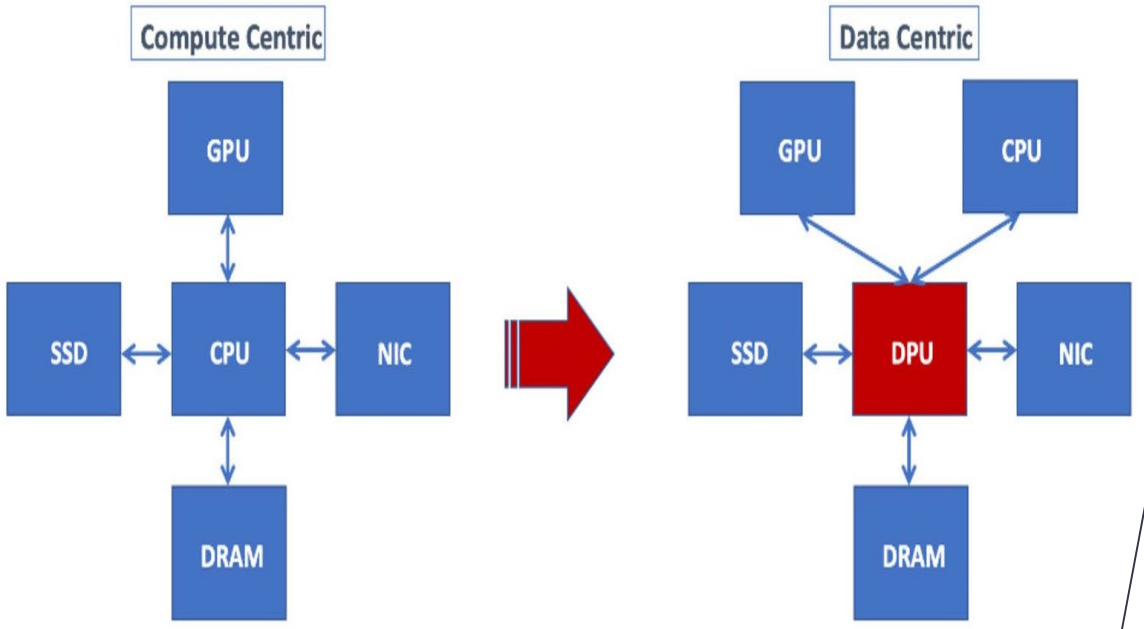
Ahmed Sanaullah Ulrich Drepper
Red Hat

November 14th, 2022

Data Centers are Evolving …

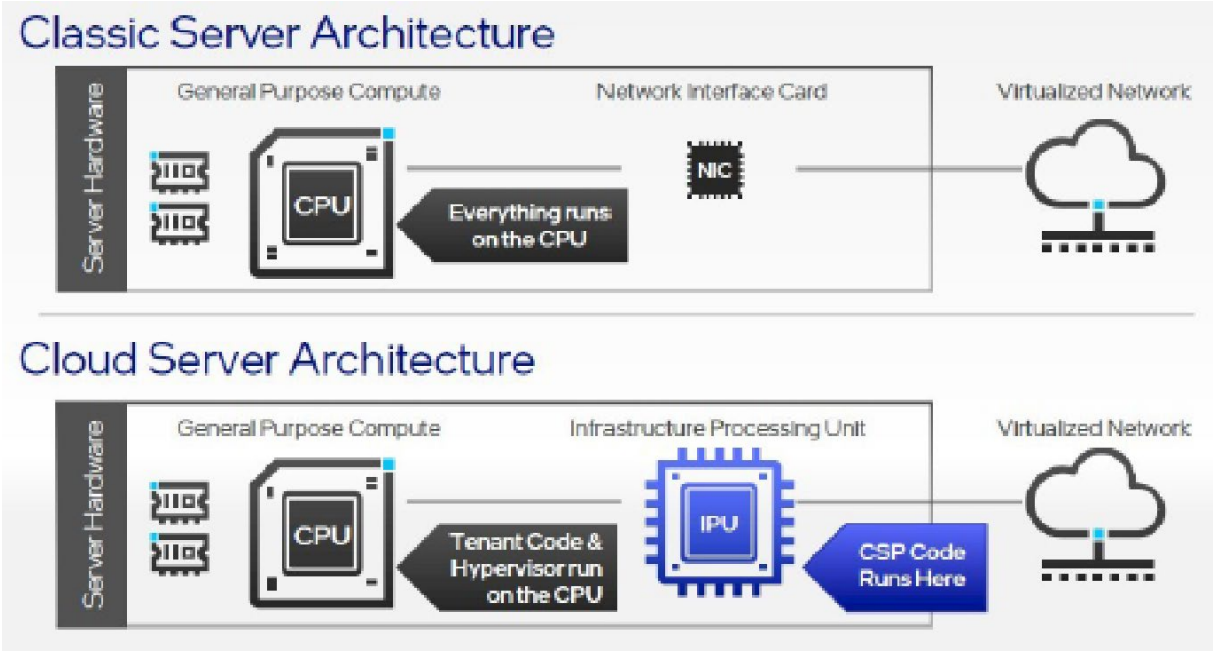
Herbert Yoshida - Hitachi

The Compute Centric data center is running out of gas. … The move to offload data management functions has already begun with the introduction of Smart NICs and **FPGAs**. Hitachi's recent announcement of new enhancements and capabilities to its hyperconverged infrastructure (HCI) portfolio …

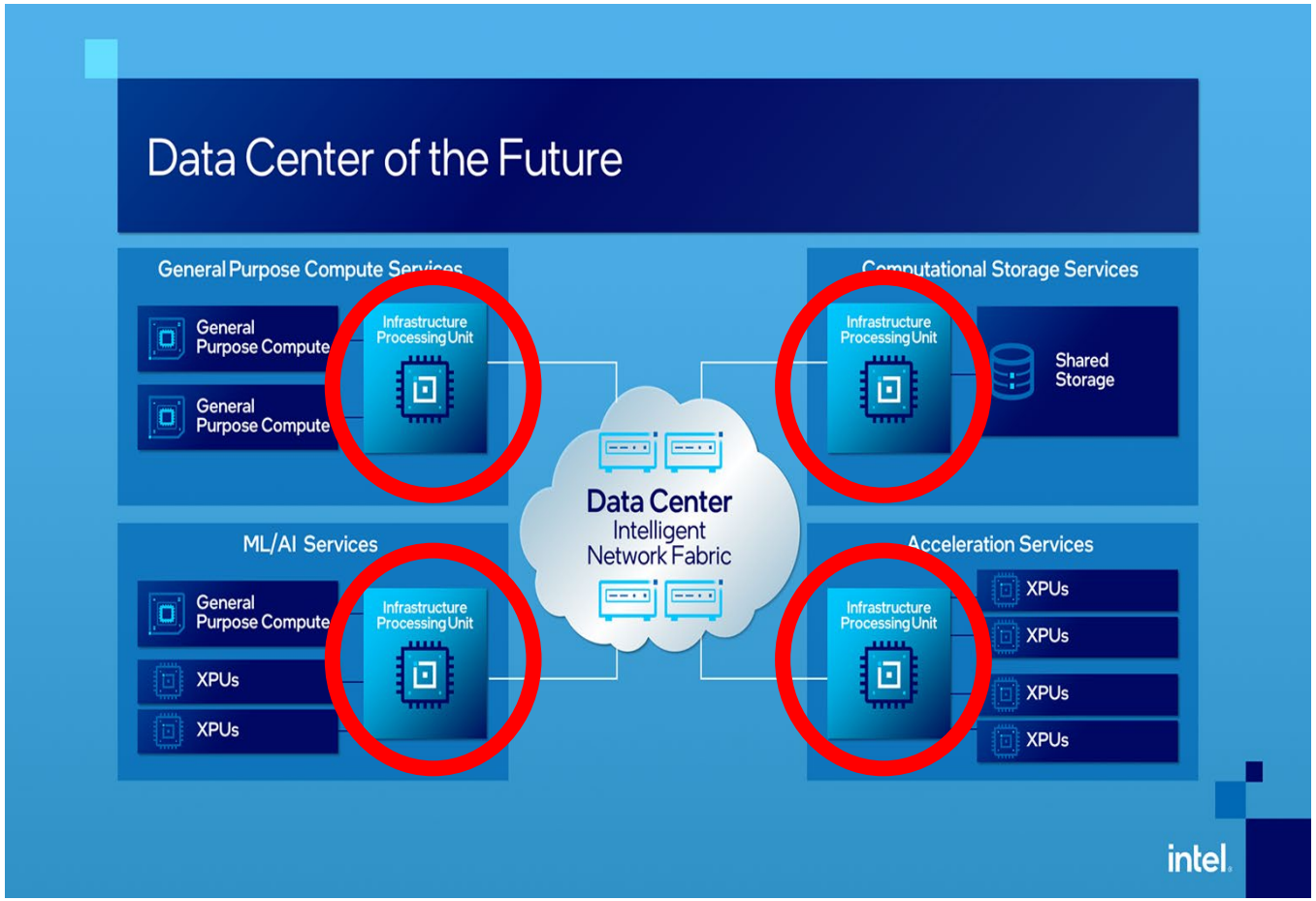


Timothy Prickett Morgan – The Next Platform

One of the reasons why **Intel** spent $16.7 billion to acquire FPGA maker Altera six years ago was because it was convinced that its onload model - where big parts of the storage and networking stack were running on CPUs - was going to go out of favor and that companies would want to offload this work to network interface cards with lots of their own cheap processing …
This is what we used to call SmartNICs …



Revolution in Data Center Architecture



Intel's vision for the datacenter

Requirements

- In-line, line-rate processing
- Complex (look-aside) line-rate processing
- Process many streams simultaneously
- Process many types of applications
- Support frequent updates, upgrades for changing protocols, standards, etc.

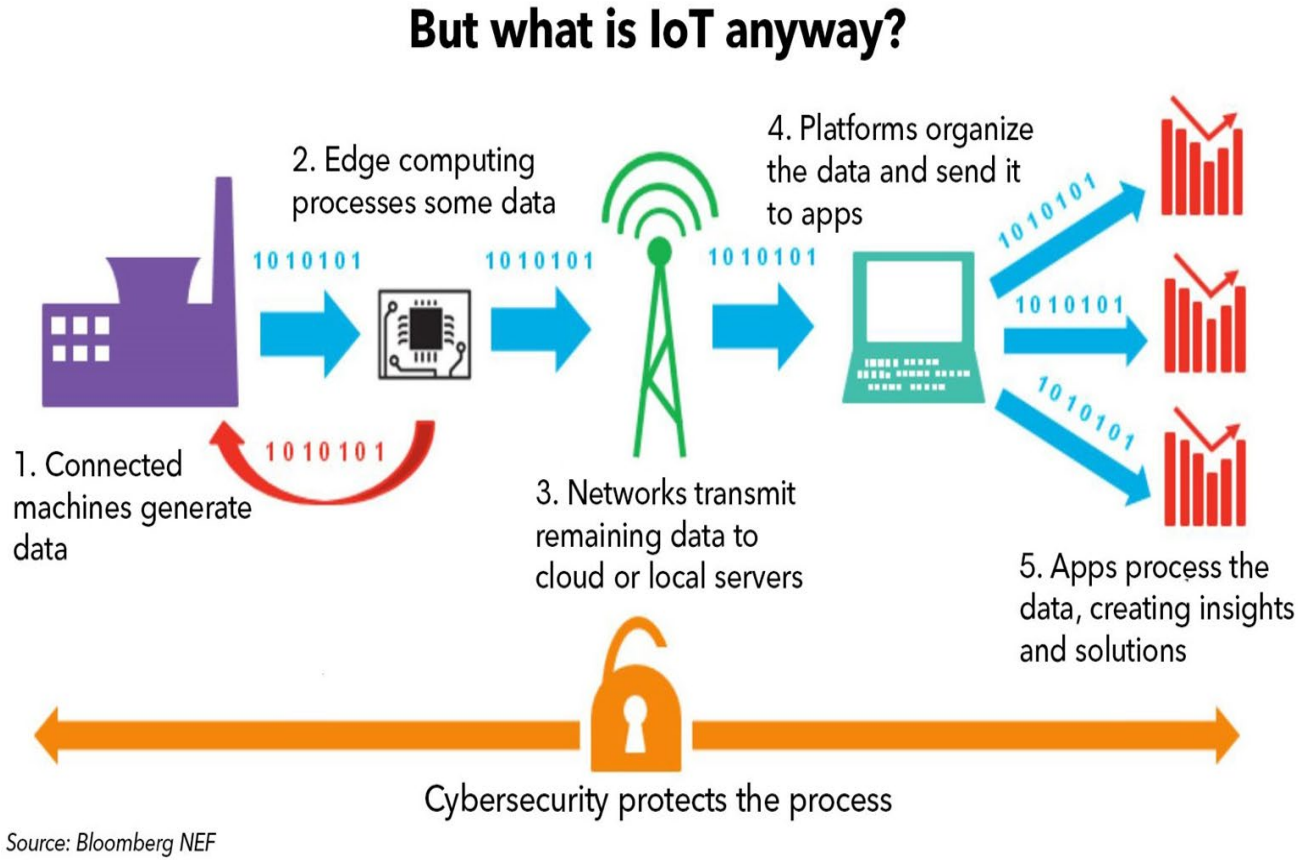
Need HW speed with SW flexibility

EDGE/IOT - same requirements as for data center

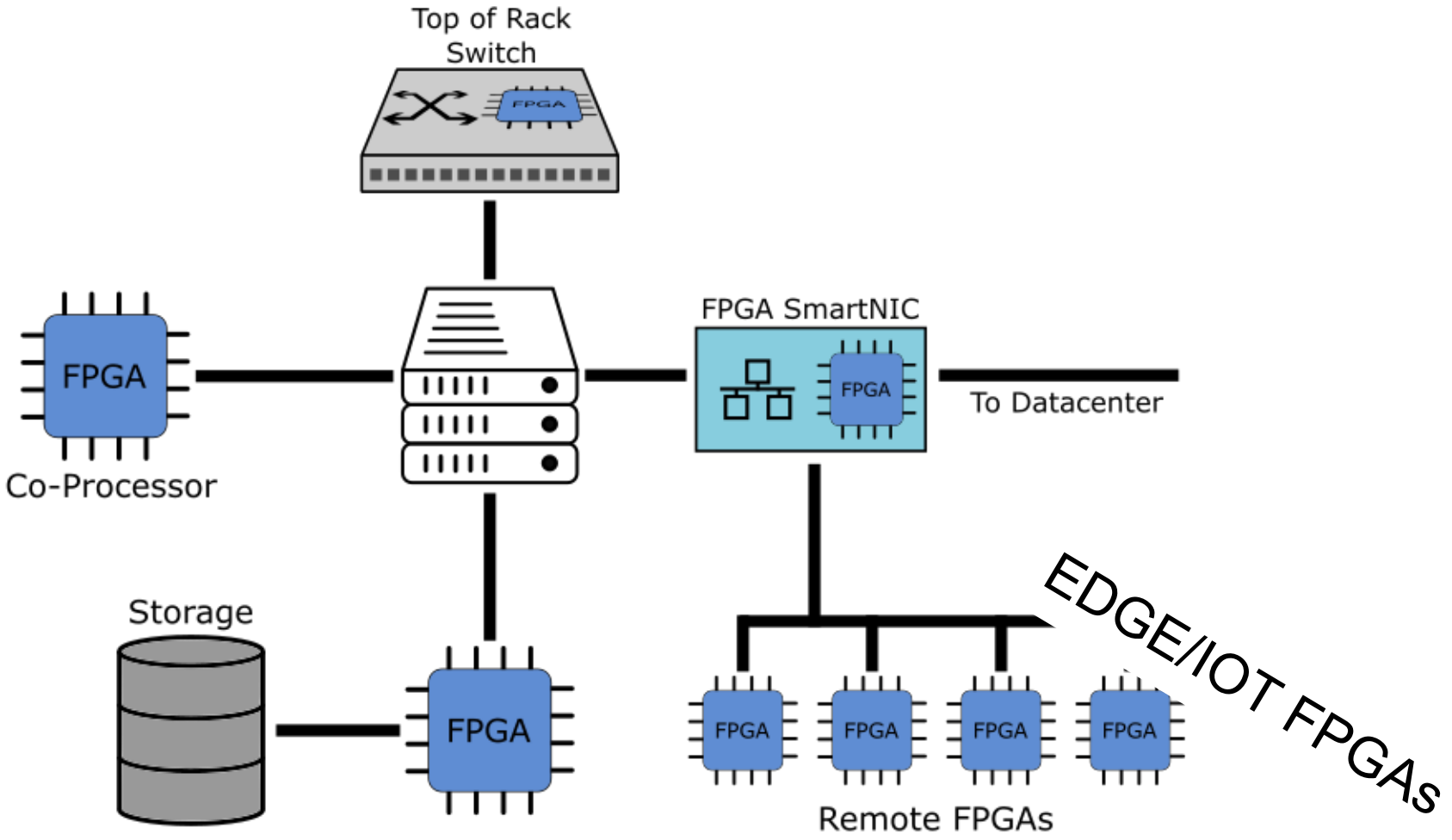
Requirements

- In-line, line-rate processing
- Complex (look-aside) line-rate processing
- Process many streams simultaneously
- Process many types of applications
- Support frequent updates, upgrades for changing protocols, standards, etc.

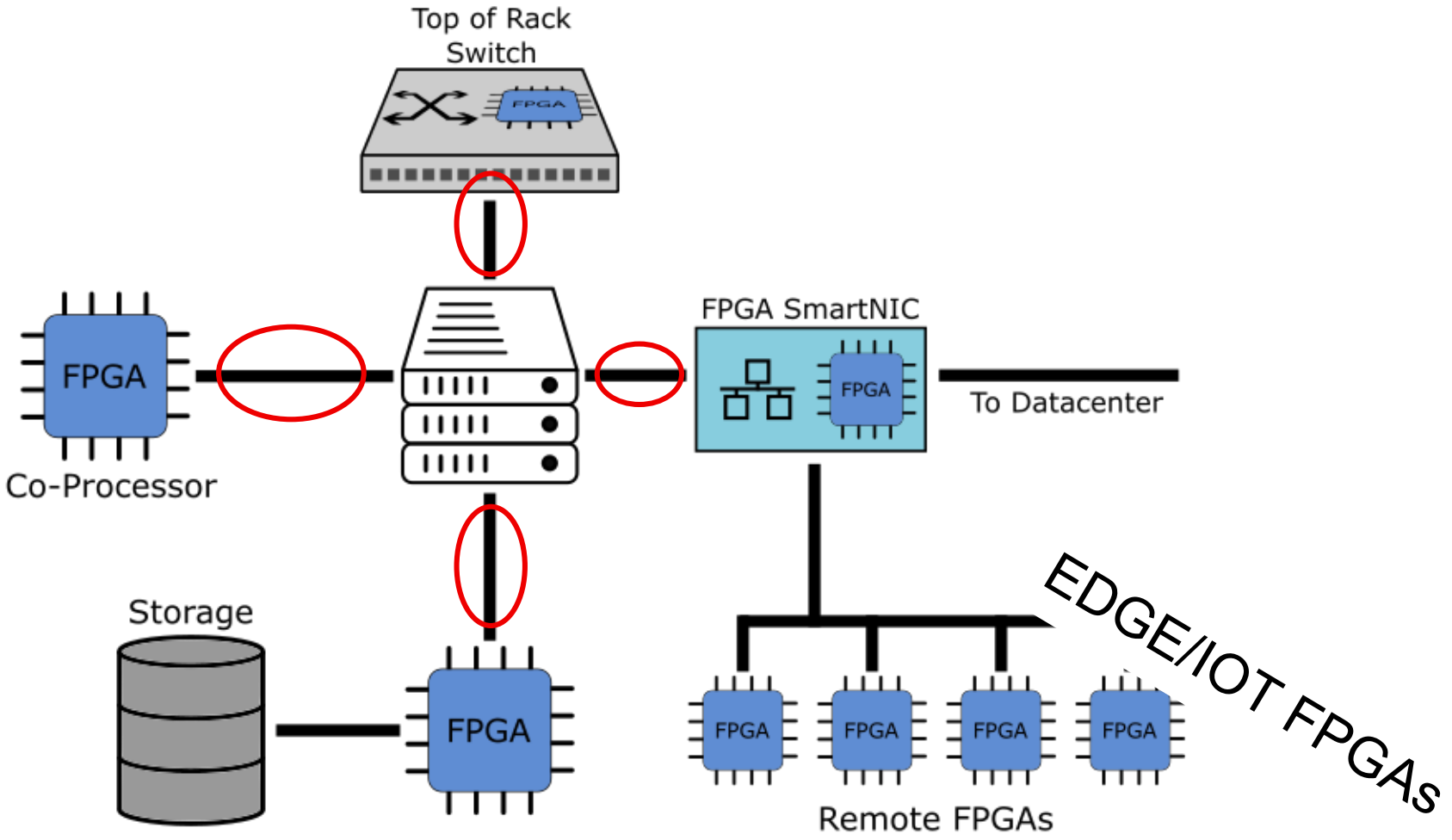
Need HW speed with SW flexibility



FPGAs in the Datacenter



FPGAs in the Datacenter

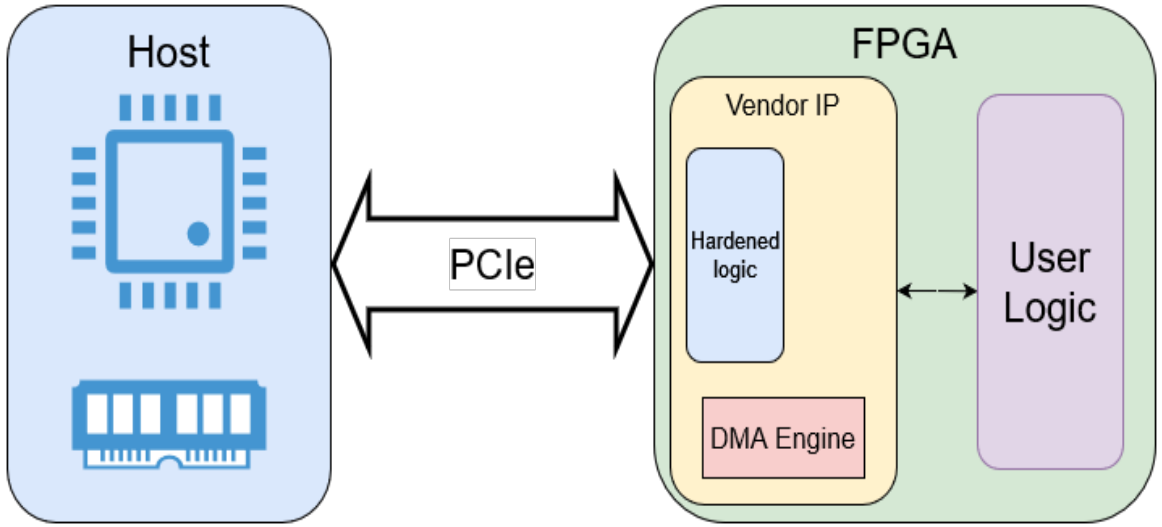


Host-FPGA Communication Interface

- To effectively leverage FPGAs
  - High Bandwidth
  - Reliable
  - Robust
  - Uniform
  - Support necessary protocols & functionality
- Ideally;
  - Portable
  - Easy to maintain software

Host-FPGA Communication (Current Status)

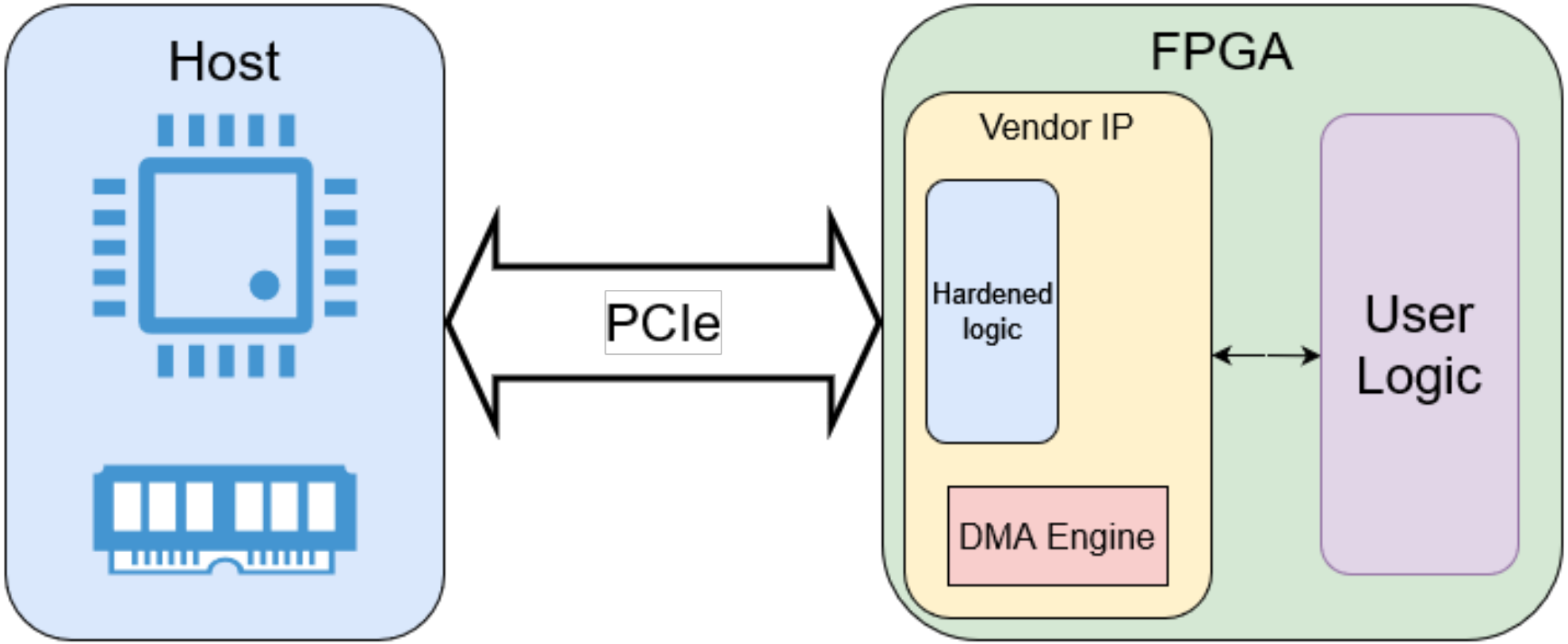
- PCIe is the most used
- Vendor provided IP blocks used on the device side
  - Even third-party custom IPs are bound to integrated blocks on the FPGA
- Device drivers on the host side
  - Provided by FPGA vendors
  - Written by end user
  - Provided by third parties
- Limitations
  - Lack of portability
  - Difficult to maintain



Host-FPGA PCIe Communication (Limitations)

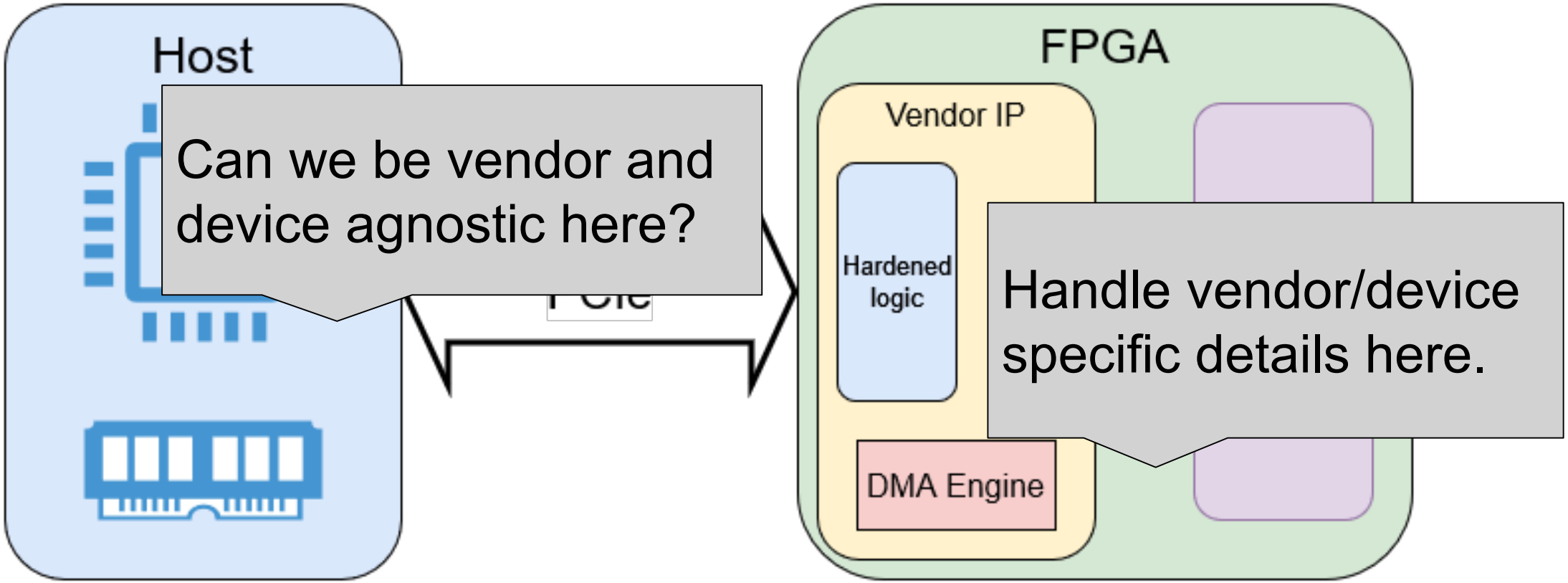
- Lack of portability
  - Device side hardware is vendor- and device- specific
    - Different resource types/amounts across FPGAs
    - IP capabilities and interfaces exposed
  - Device drivers are designed to handle the device specific details
    - Specific to vendor, device, and even IP core
- Maintaining the drivers – must modify whenever
  - Host kernel updates
  - Deprecated dependencies
  - Newer versions of vendor IPs

Host - FPGA PCIe Communication



Device driver (usually) needs to match specifics of vendor IP
Ideally – same mechanism for accessing the device from VM
Vitio drivers?

Host - FPGA PCIe Communication



Device driver (usually) needs to match specifics of vendor IP
Ideally – same mechanism for accessing the device from VM
Vitio drivers?

The Big Picture

- This is not an issue if:
  - You have infinite resources to write and maintain all your drivers
  - Or you always use the same device
  - And the kernel never updates

- For everyone else: – Desire: Never need to do anything ever again.
  - Don’t write device drivers for FPGAs
  - No need to update the drivers with kernel updates
  - No need to update the drivers with IP updates
  - Use the same driver with all the FPGAs
  - Design portability across devices
  - Don’t implement PCIe controllers, etc. on the FPGA
  - Just focus on the application logic

Current Efforts Towards a Solution

- Open Programmable Acceleration Engine (OPAE)
  - Software framework for managing and accessing Intel programmable accelerators (FPGAs)
  - Linux device driver and SDK (user level libraries and tools)
  - User kernel implemented in the reconfigurable region inside the FPGA shell
  - Only works with **some** Intel FPGAs
- Xilinx Runtime Library (XRT)
  - Standardized software interface that facilitates communication between the application code and the accelerated-kernels deployed on the reconfigurable portion of PCIe based FPGAs
  - User space library and kernel drivers
  - Works with a FPGA shell that implements PCIe interface, DMA, etc.
  - Only works with **some** Xilinx FPGAs
- Device drivers and IPs from FPGA vendors and third parties
  - Just to implement PCIe communication
  - Open-source and proprietary, paid and free

Our Observations – life of the FPGA user:

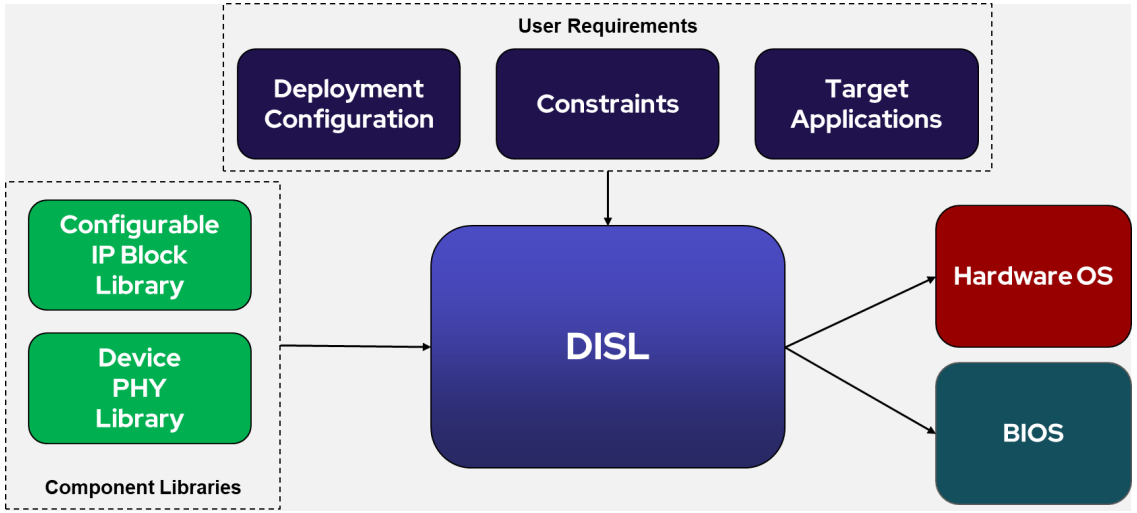
- FPGA shells + runtime libraries + drivers:
  - Allow the user to focus on the application logic
  - No need to implement PCIe, DMA, etc. on FPGA side or drivers on host side
  - But, extremely limited in devices supported
  - Specific to both the device and the development board
  - Shell implementations do not have good separation between device/board specific and generic logic limiting portability
- Device driver + IP
  - Implementation effort is higher than when using FPGA shells
  - User does not have to write the drivers
  - The user might have to keep the drivers up to date
  - Vendor provided: Usually, spotty maintenance for drivers which are not for the latest and greatest FPGAs
  - Third-party provided: Only a limited set of devices/boards supported

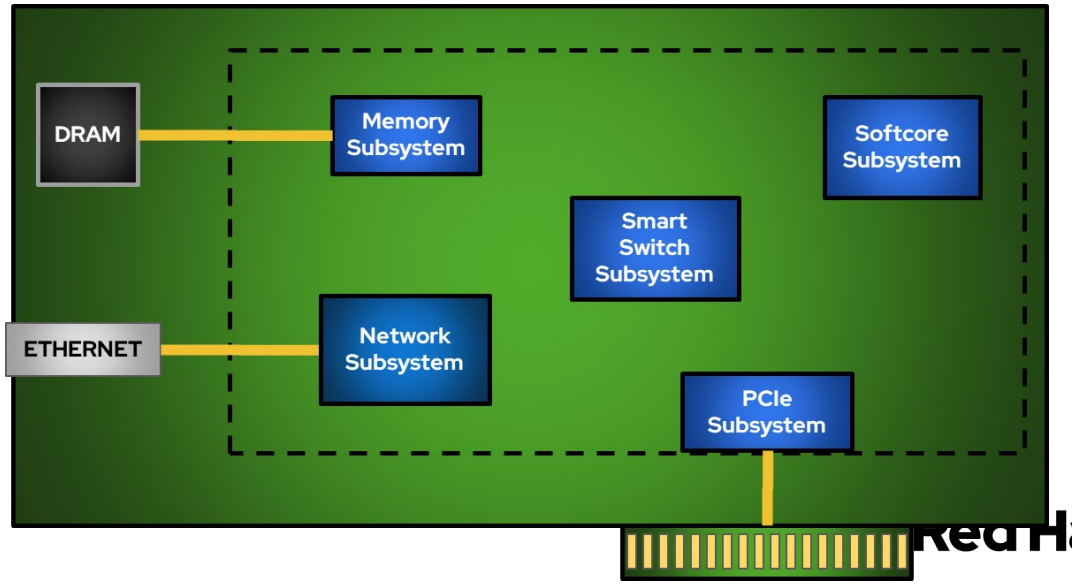
What We Like to See in a Solution

- FPGA side:
  - Clear separation between device specific and generic logic in the shell
  - Standard interfaces exposed to user logic
  - Application portability between devices/boards
  - Limit the use of vendor provided IPs to the integrated ASIC blocks and replace rest of the logic with generic open-source components
- Host side
  - Same driver across all devices
    - Driver and device should be able to negotiate supported features
  - Drivers are part of the operating system and get updated with the kernel updates
    - and not as a reaction to a kernel update

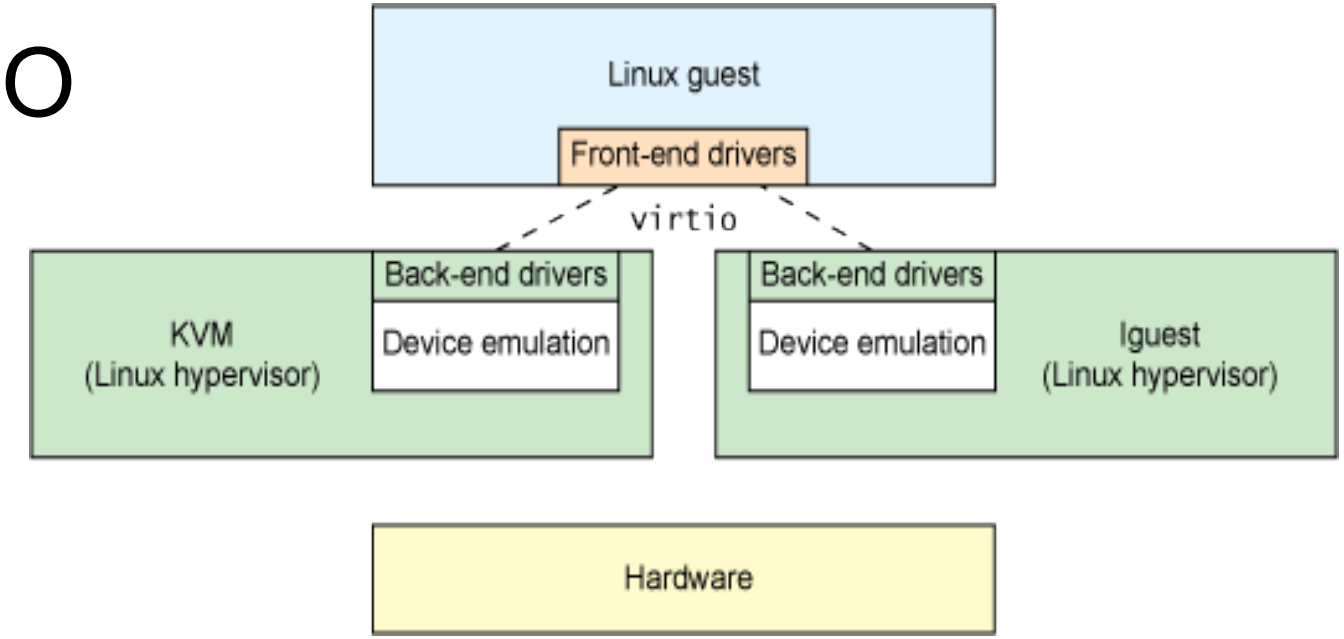
Aside - This work is part of a bigger project (DISL)

- Dynamic Infrastructure Services Layer for Reconfigurable Hardware (DISL)
  - Operating system like abstractions on the FPGA
  - Generator framework to automatically generate BIOS and OS layers using a component library and user requirements as the input
  - OS layer provides services to the user application through standard interfaces
    - Making applications portable
  - BIOS layer implements all the device specific logic making the OS layer common across devices
  - Use standard in-kernel drivers for host-FPGA communication
    - Our choice is VirtIO drivers





I/O Virtualization and VirtIO



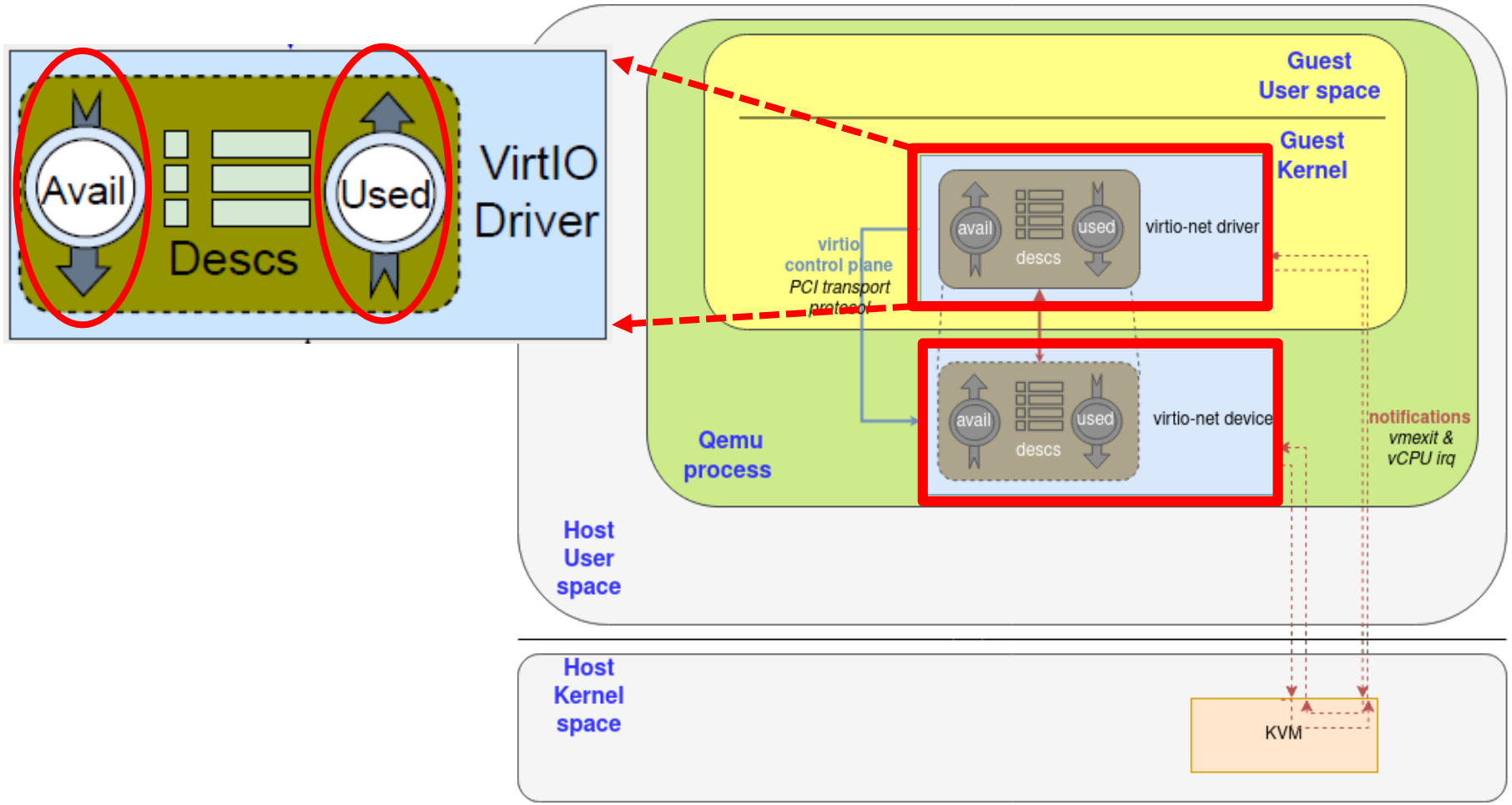
- Guest processes interact with devices emulated by a hypervisor
- Full virtualization:
  - Guest OS is unaware of the virtualization
  - Hypervisor traps I/O requests and emulates the behavior of real hardware
- Paravirtualization:
  - Guest OS is aware of the virtualization and uses appropriate front-end drivers
  - The hypervisor implements the back-end drivers for the particular device emulation
- VirtIO is an abstraction for a set of common emulated devices in a ***paravirtualized*** hypervisor

https://developer.ibm.com/articles/l-virtio/

VirtIO Drivers

- Industry standard for I/O virtualization
- Native Support on host operating systems
  - No need to write/maintain an additional driver
- APIs are relatively consistent
- Allows feature negotiation between driver and device
- Works from both host and guest OSs
- Potential to define new device types to match different FPGA use cases
- VirtIO devices:
  - Virtual devices found in virtual environments
  - Can use normal bus mechanisms for device discovery, interrupts, DMA, etc.

VirtIO Drivers – Basic Model

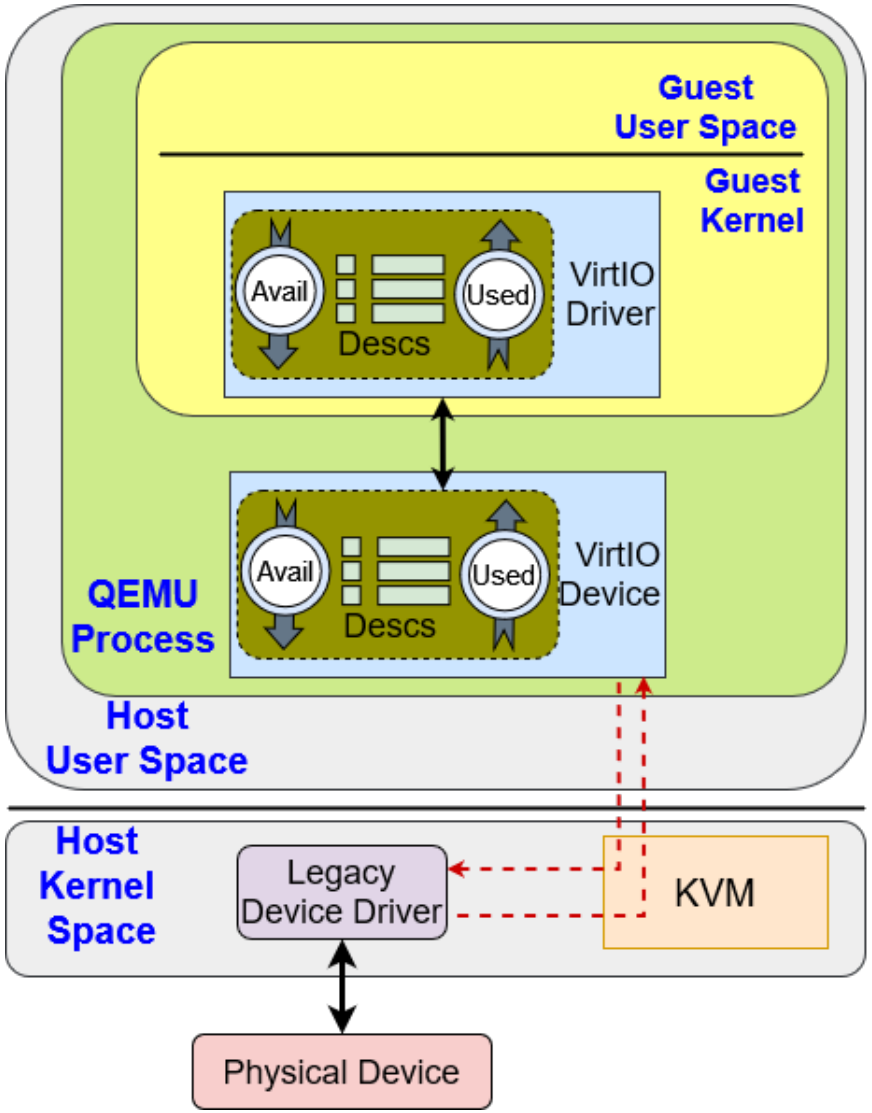


3 key components

QEMU = an open-source machine emulator

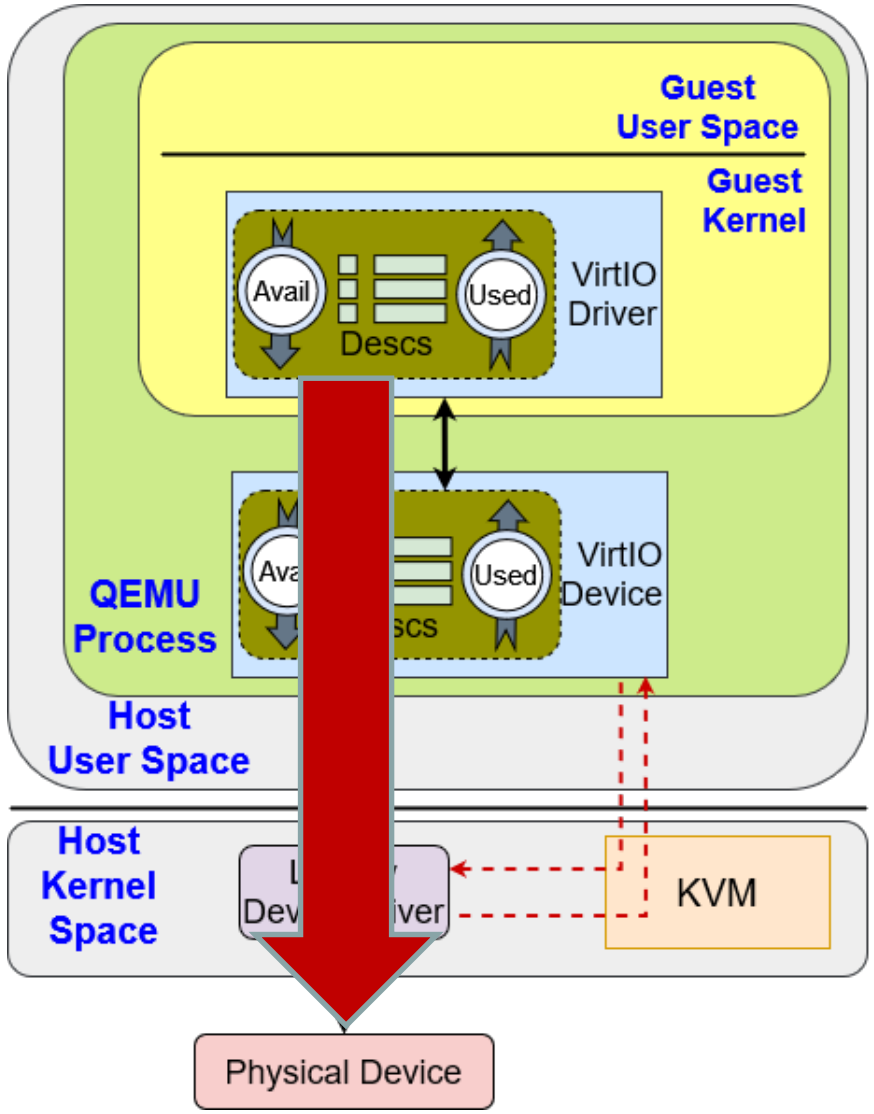
https://www.redhat.com/en/blog/virtqueues-and-virtio-ring-how-data-travels

VirtIO Drivers – paravirtualization w/ physical device



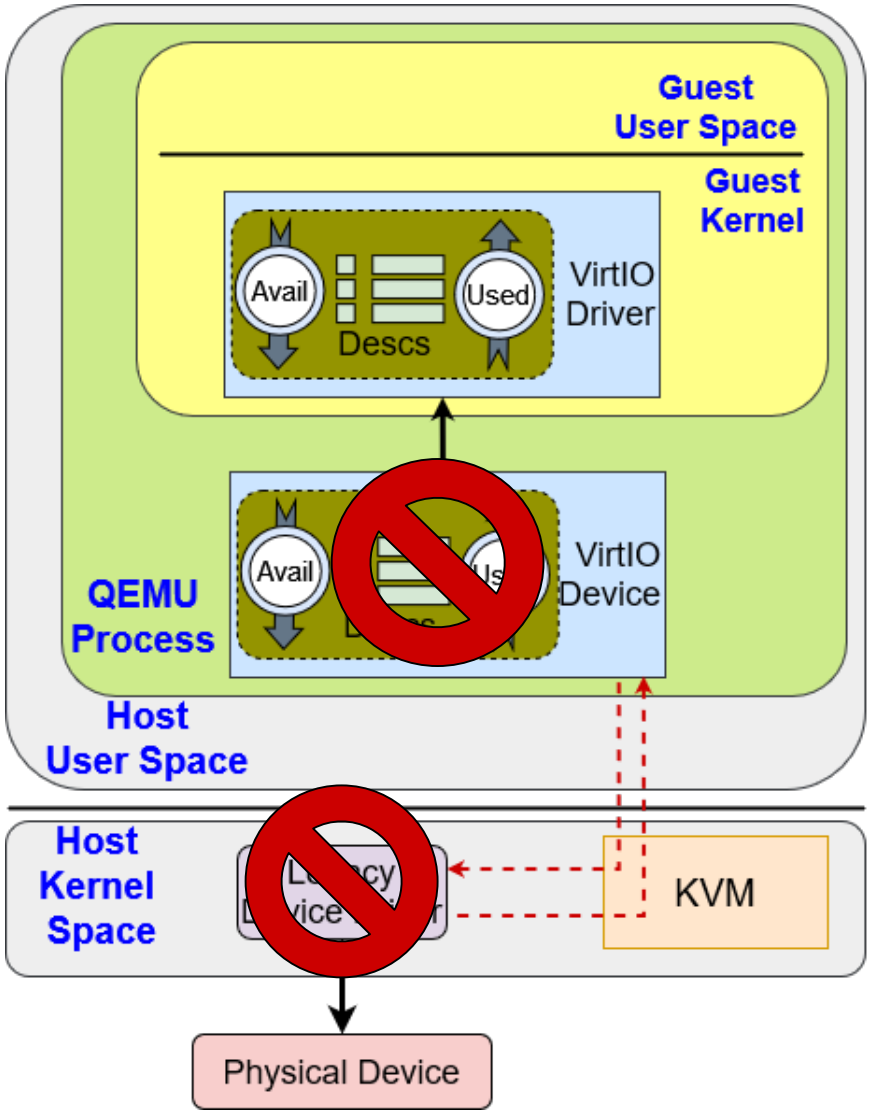
Host uses legacy driver

VirtIO Drivers – bypass layers?



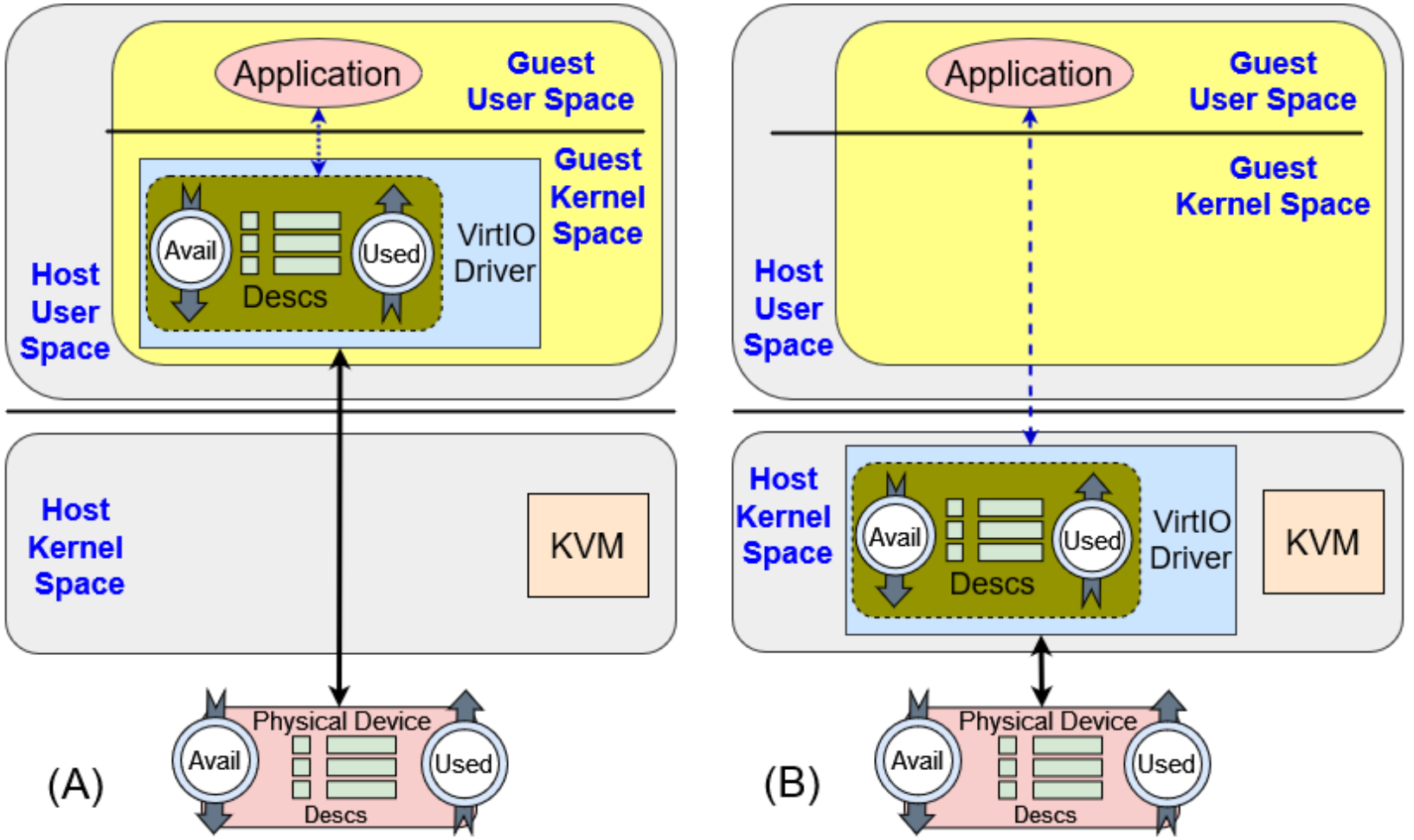
Get rid of copies

Contribution: VirtIO Support on FPGAs



- Virtio functionality directly on the FPGA
- Neither device specific driver is now needed
- Or the emulated back end

New Use Models



No logical difference btw guest or host virtio driver accessing the FPGA

New models
a) Virtio driver runs in guest kernels space – PCI passthrough?
b) User application communicates directly with virtio driver in host kernel space

Challenges on the FPGA Side

- The FPGA must present a VirtIO compliant interface to the host
  - Implementing appropriate data structures and state machines
- The vendor-provided IP blocks may not support some of the functionality
- Our approach:
  - Building a subset of hardware blocks using generic RTL
  - Leverage existing IP blocks for device specific parts

Implementing VirtIO Functionality (on FPGA)

For compliant interface →

1 Device discovery and initialization

- Add VirtIO capabilities to the device's PCIe capability list
- VirtIO data structures

2 Data movement

- Virtqueue control state machines
- More data structures

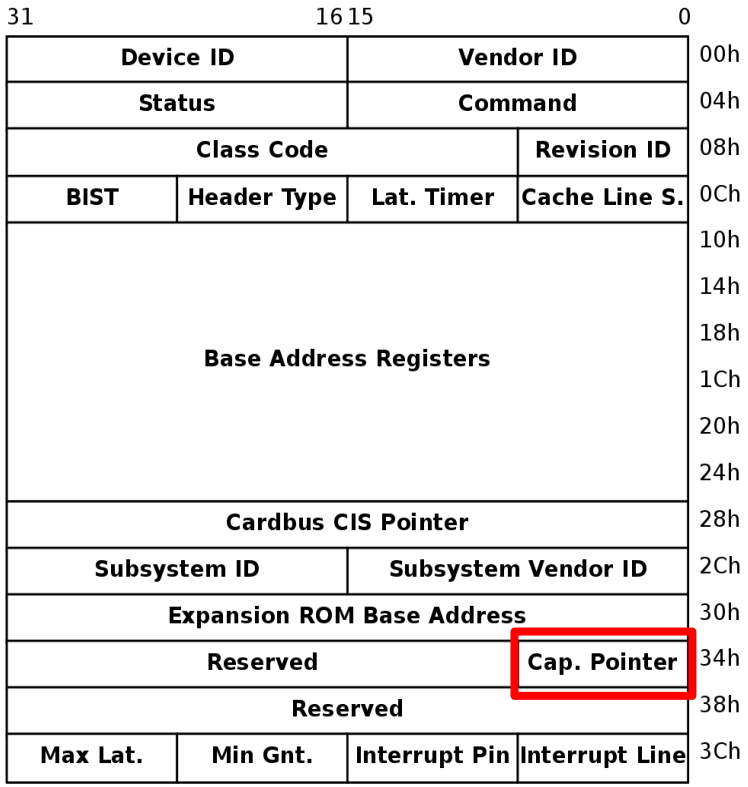
VirtIO drivers are agnostic to device specific details

- Need to control DMA engine from the device side
  - Potentially replace IP or DMA engine

Implementing VirtIO Functionality (In Detail)

To add virtio capabilities to the device's capability list →

- Capabilities are part of the device's PCI configuration space
- Formed as a linked list
- VirtIO needs five capabilities
  - Common Configuration
  - Notification
  - ISR status
  - Device specific configuration (optional)
  - PCI configuration access
- Configuration space is part of the hardened PCIe block (in our case)
  - This may not be true for all devices –
  - Which would make it simpler to modify IP



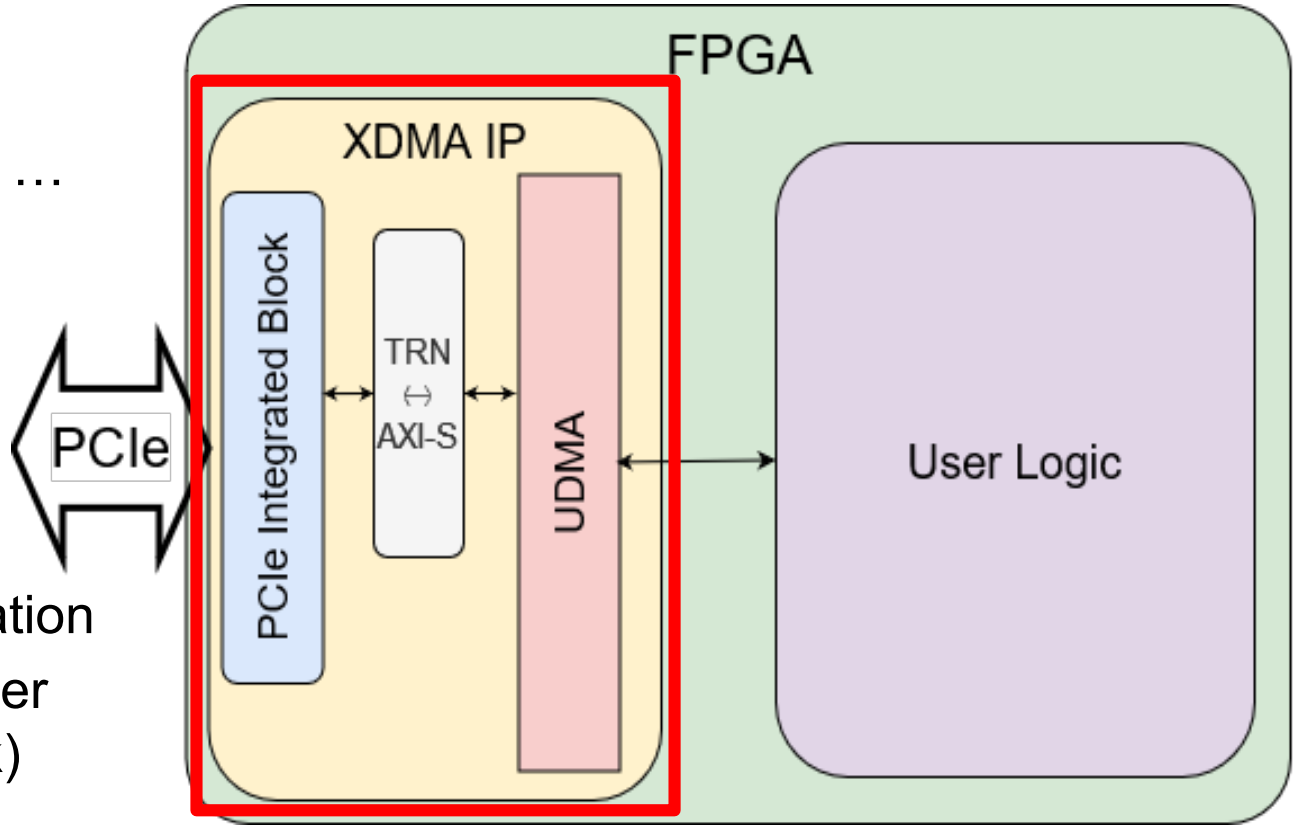
By Vijay Kumar Vijaykumar - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=3181779

Implementing VirtIO Functionality (In Detail)

- PCIe block allows the configuration space accesses to be forwarded – but …
- PCI configuration space accesses are different from regular memory accesses over PCI
  - Different packet type
  - Must be routed out of the integrated block
  - Need user logic to implement extra configuration space registers
- Config. space access forwarding feature not available for the DMA IP
  - Need to modify IP source files to enable this feature
- DMA IP removes packet header information before it reaches user logic
  - Implement extra configuration space registers inside the IP
- Cannot set the *next pointer* of capabilities implemented by the integrated block
  - So, modify IP source files

Look at the Internals of the XDMA IP *(for hacking)*

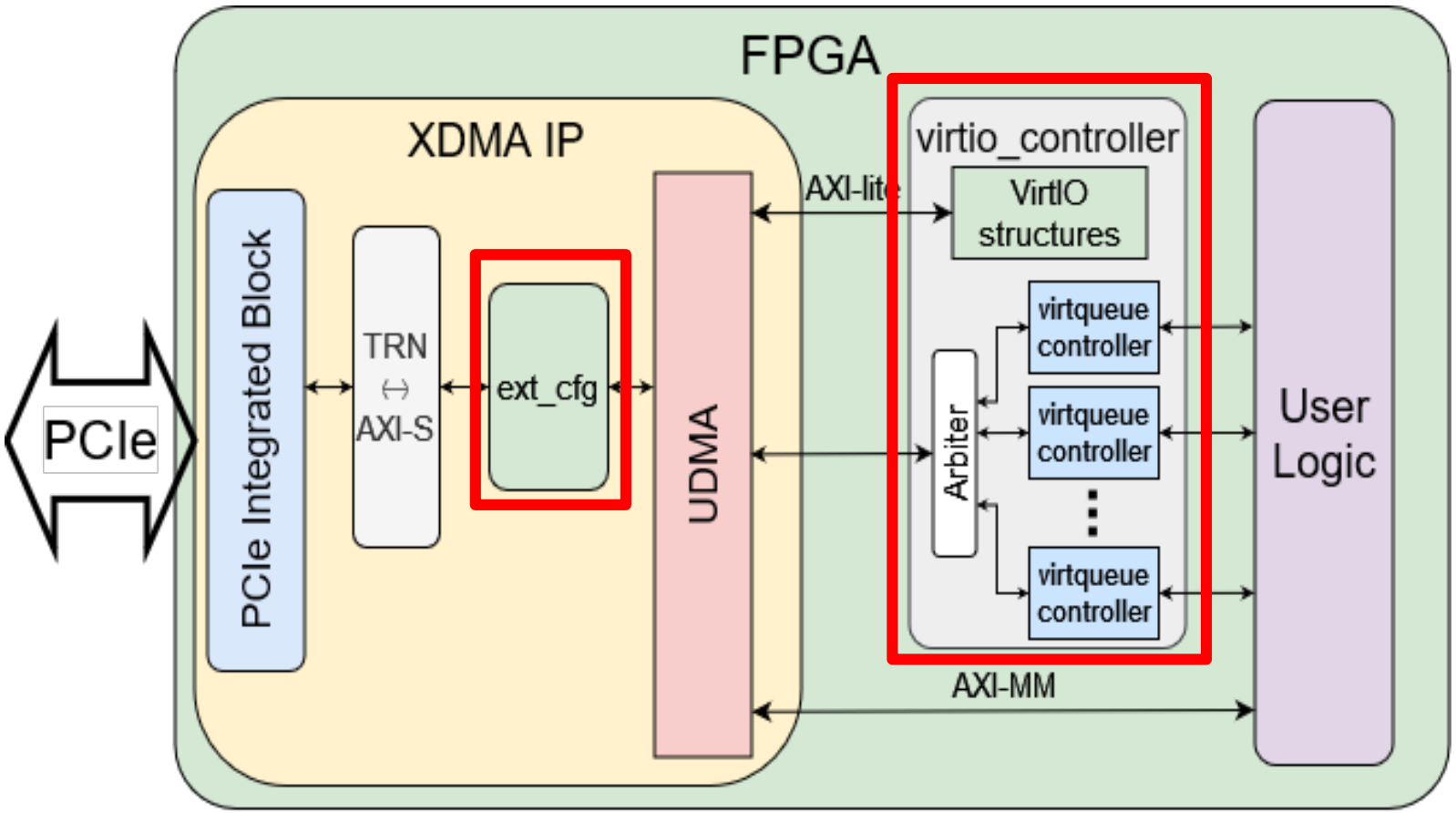
- PCIe integrated block includes hardened and …
  - Physical and link layer
  - Configuration space
- … reconfigurable logic
  - DMA engine
  - Operate at the transaction layer
- User logic uses AXI interfaces for communication
- Additional interfaces optionally exposed to user logic to control the DMA engine (UDMA block)
- Integrated block instantiation parameters:
  - Forward configuration space accesses
  - Set next pointers for capability list



Implementing Virtqueues

- Virtqueues move data between the driver and the device
- For VirtIO over PCIe, the device is responsible for controlling the data movement using DMA
- Need support for multiple queues
  - Individual controllers for each queue
- Queues have to share the DMA engine
  - Implement necessary arbitration logic
- VirtIO data structures
  - required for device initialization and data movement
  - Pointed to by the capabilities we added
  - Implemented in reconfigurable logic

Implementing VirtIO Functionality



Virtio controller block needs data structures and state machines

To add capabilities, modify vendor IP:
*Intercept and responds to transactions btw hardened PCIe block and DMA engine
*We demonstrate that hardware should have necessary capabilities to implement virtio support
*Potential limiting factor – FPGA tools not exposing features to user and encrypted IP which prevents modification by user

Challenges

- Not all the capabilities of the hardware are made available to the user
  - Because those are advanced features?
  - Improper use can prevent both the FPGA and the host machine from working properly
- Lack of documentation on how to use the features
- Some capabilities of the integrated blocks are not properly utilized by the vendor IPs
  - Configuration space access forwarding is disabled in the XDMA IP although the PCIe integrated block has the capability (Sahan added this to IP source)

Results - QED

Virtio console device implemented and identified by unmodified virtio drivers

```
02:00.0 Serial controller: Red Hat, Inc. Virtio console (rev 01) (prog-if 01 [16450])
        Subsystem: Red Hat, Inc. Virtio console
        Physical Slot: 3
        Flags: bus master, fast devsel, latency 0, IRQ 16
        Memory at df110000 (32-bit, non-prefetchable) [size=4K]
        Memory at df100000 (32-bit, non-prefetchable) [size=64K]
        Capabilities: [40] Power Management version 3
        Capabilities: [48] MSI: Enable- Count=1/4 Maskable- 64bit+
        Capabilities: [60] Express Endpoint, MSI 00
        Capabilities: [9c] MSI-X: Enable+ Count=31 Masked-
        Capabilities: [a8] Vendor Specific Information: VirtIO: CommonCfg
        Capabilities: [b8] Vendor Specific Information: VirtIO: Notify
        Capabilities: [cc] Vendor Specific Information: VirtIO: ISR
        Capabilities: [dc] Vendor Specific Information: VirtIO: <unknown>
        Capabilities: [100] Device Serial Number 00-00-00-00-00-00-00-00
        Kernel driver in use: virtio-pci
```

Virtio capabilities added to the device's capabilities list

We have communicated with user logic on the FPGA

Future Work

- Implement other VirtIO device types
- Define new VirtIO device types that correctly represent different FPGA use cases
- Detailed performance analysis
  - Against vendor and third-party IPs and drivers
  - Host vs guest OS communicating with the device
  - For different VirtIO device types

Thank You!