2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

VIRTIO BASED GPU MODEL

Pratik Parvati
Lead Engineer
Vayavya
Labs Pvt. Ltd.

accellera
SYSTEMS INITIATIVE

CONTENT

01 The Problem

02 Host GPU accelerated Model

03 The Solution – Virtio

- Devices, Drivers.
- VirtIO transport layer.

04 VirtIO-GPU

- Devices, Drivers

05 Device Operation

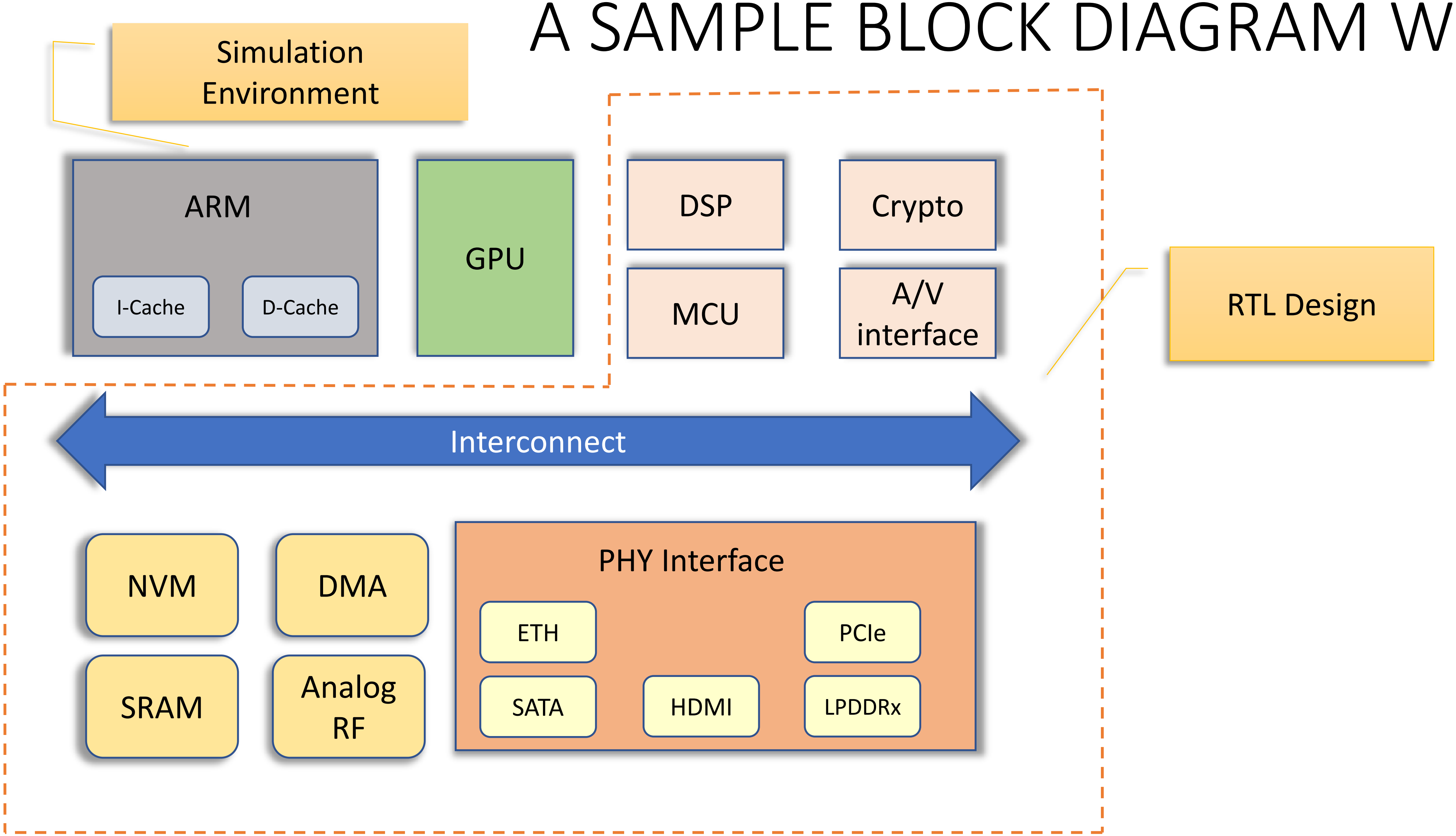
- Controlq, cursorq
- Virglrenderer.

06 Demo

WHAT IS THE PROBLEM WE ARE TRYING TO ADDRESS?

- GPUs are essential components of modern computer systems.
  - GPUs are widely used in various fields like Scientific Computing, Image Processing, Data Analysis etc.
- Unlike CPU, GPU is optimized for Parallel instruction operation.
- Challenges involved in modeling a full blown GPU.

A SAMPLE BLOCK DIAGRAM WITH GPU



HOW DO WE MODEL GPU

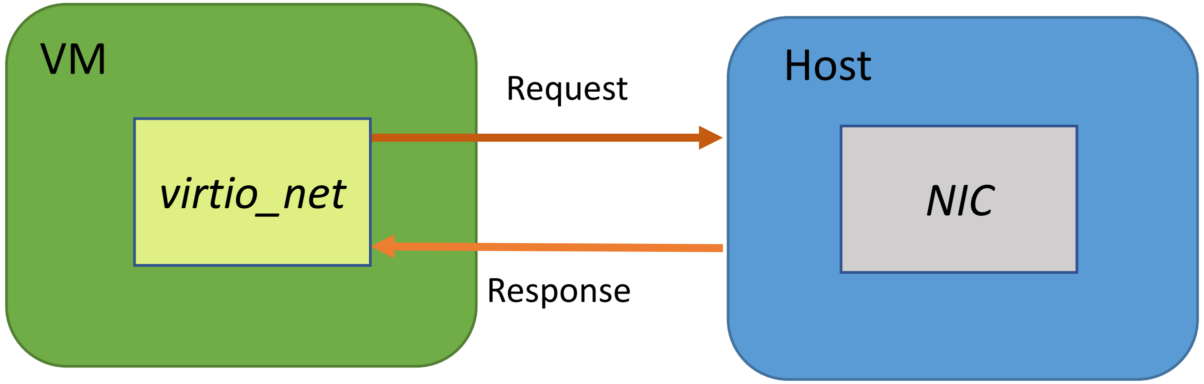
- Cons with Full-blown SystemC model
    - GPUs requires data structures that are more rigid than on conventional processors (CPU).
        - Designed exclusively for high performance computing applications.
    - GPUs have hundreds to thousands of processing elements.
        - AMDs Radeon HD 6000 series of GPUs contain more than 1000 processing elements on a single GPU die.
    - Demands parallel simulators as sequential simulators are slow.
    - Synchronization overhead by simulating the parallel components of the GPU architecture independently using multiple simulation threads.

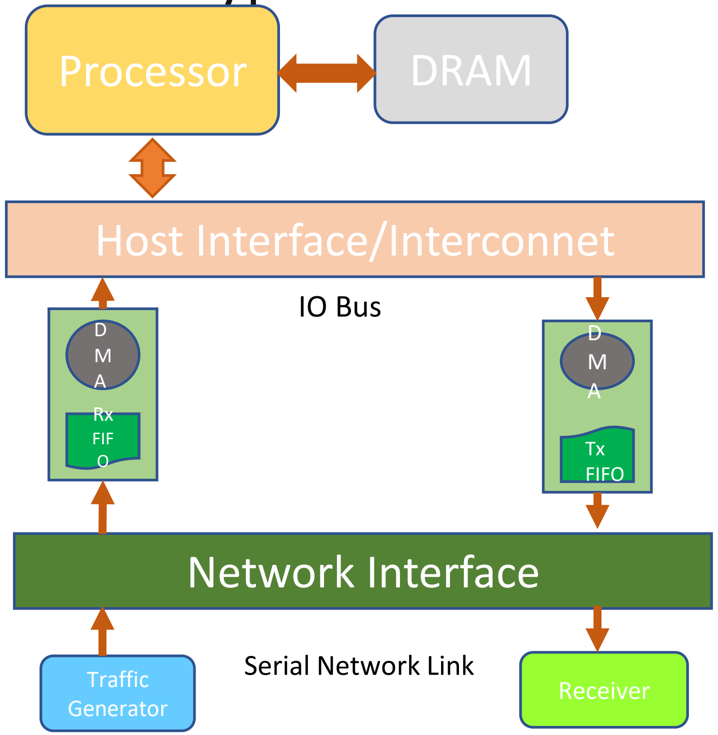
SOME GRAPHICS TERMS

- *Pixels* - Rectangular grid, arranged in rows and columns on the screen.
- *Vertices* - Co-ordinates of an objects like lines, curves and polygon.
- *Primitives* - Building blocks containing lines, curves and polygon, which can be combined to create more complex graphical images.
- *Shader* – Program that rests on GPU, that transforms set of inputs to output as per an algorithm.
- *Texture Mapping* - Texture mapping applies an image to the faces of our geometry and adds realism to the scene.
- *OpenGL* - Software interface to the graphic hardware
- *MESA drivers* – Graphics library, is an open source implementation of OpenGL, Vulkan, and other graphics API specifications

INTRODUCTION TO VIRTIO

- VirtIO stands for virtual input & output and was developed by Rusty Russell.
- VirtIO is an abstraction layer over a host's devices in para-virtualized hypervisor.



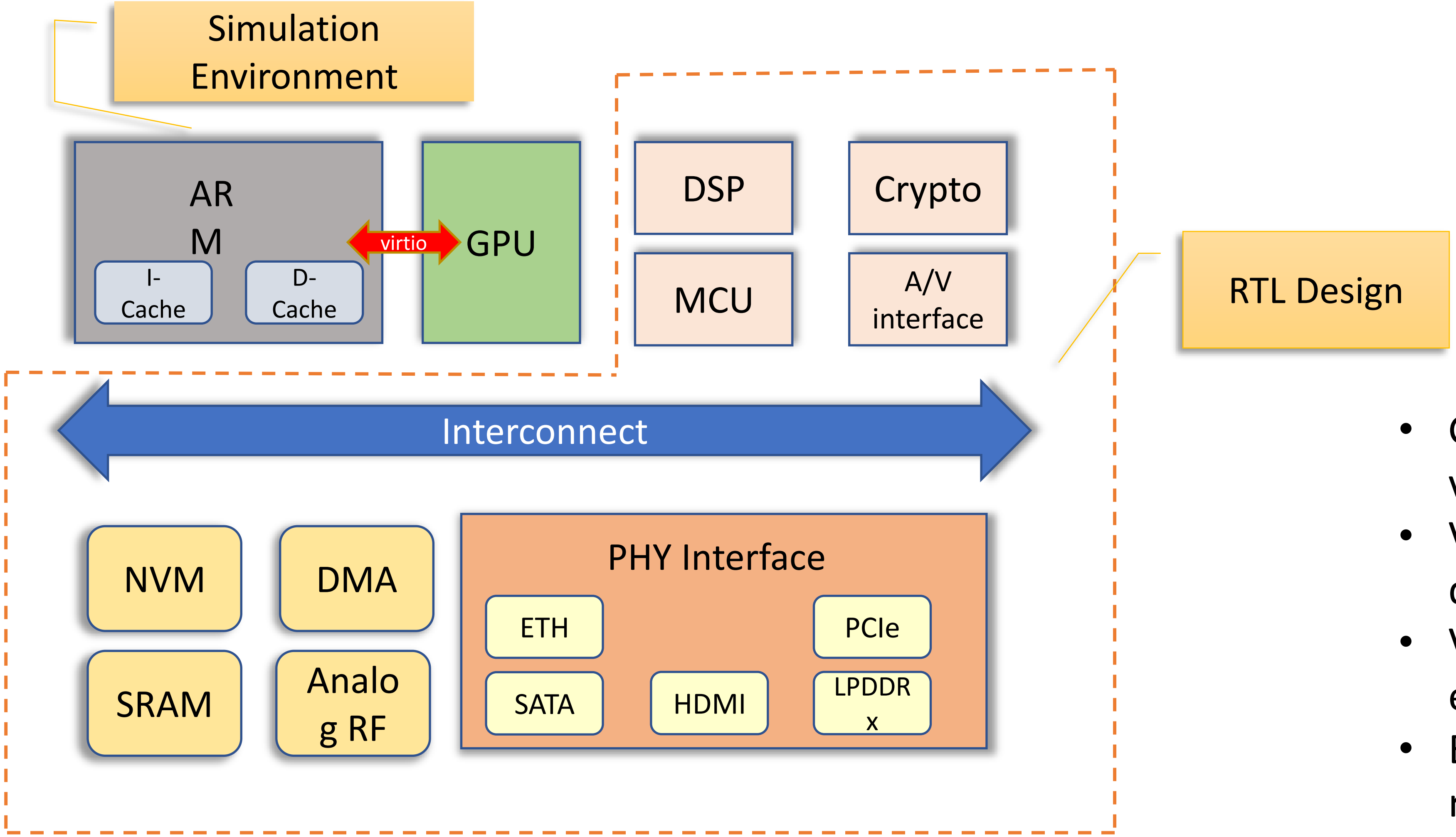


Traditional NIC Modeling approach

- Offloading the majority of the work to the host.
  - Speeds up VM operation over more traditional "emulated" devices.

- VirtIO is a HSI standardized interface

HOST-GPU ACCELERATED MODEL



- GPU is modelled on top of vitio interface.
- Virtio-gpu driver compatible device.
- Virtio-gpu driver is treated as embedded software on CPU.
- Exploits Host CPU and GPU resources.

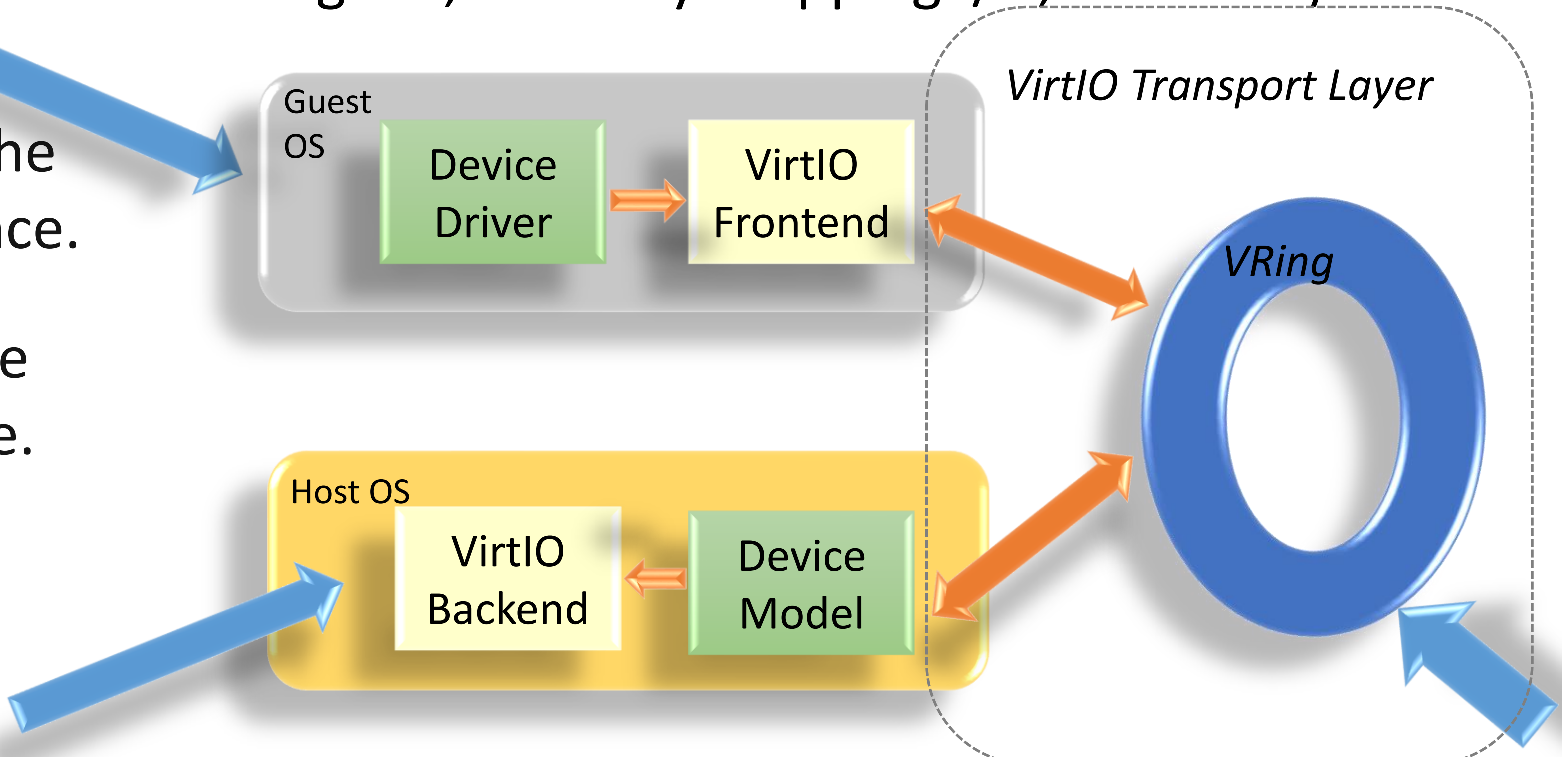
Why VirtIO?

- **Straightforward**: VirtIO devices use normal bus mechanisms of interrupts and DMA.
- **Efficient:** VirtIO devices consist of rings of descriptors for both input and output.
- **Standard:** VirtIO makes no assumptions about the environment in which it operates.
- **Extensible:** VirtIO devices contain feature bits which are acknowledged by the guest operating system during device setup.
- Improved host and guest performance.
- Exports a common set of emulated devices and make them available through common API.

VirtIO Devices

- Support different kinds of devices (network, block, video, GPU...)
- Exposed to the emulated environment using PCI, Memory Mapping I/O, Channel I/O.

- The frontend component is the guest side of the virtio interface.
- The backend component is the host side of the virtio interface.
- VirtIO Transport Layer is a channel between front -end and back -end



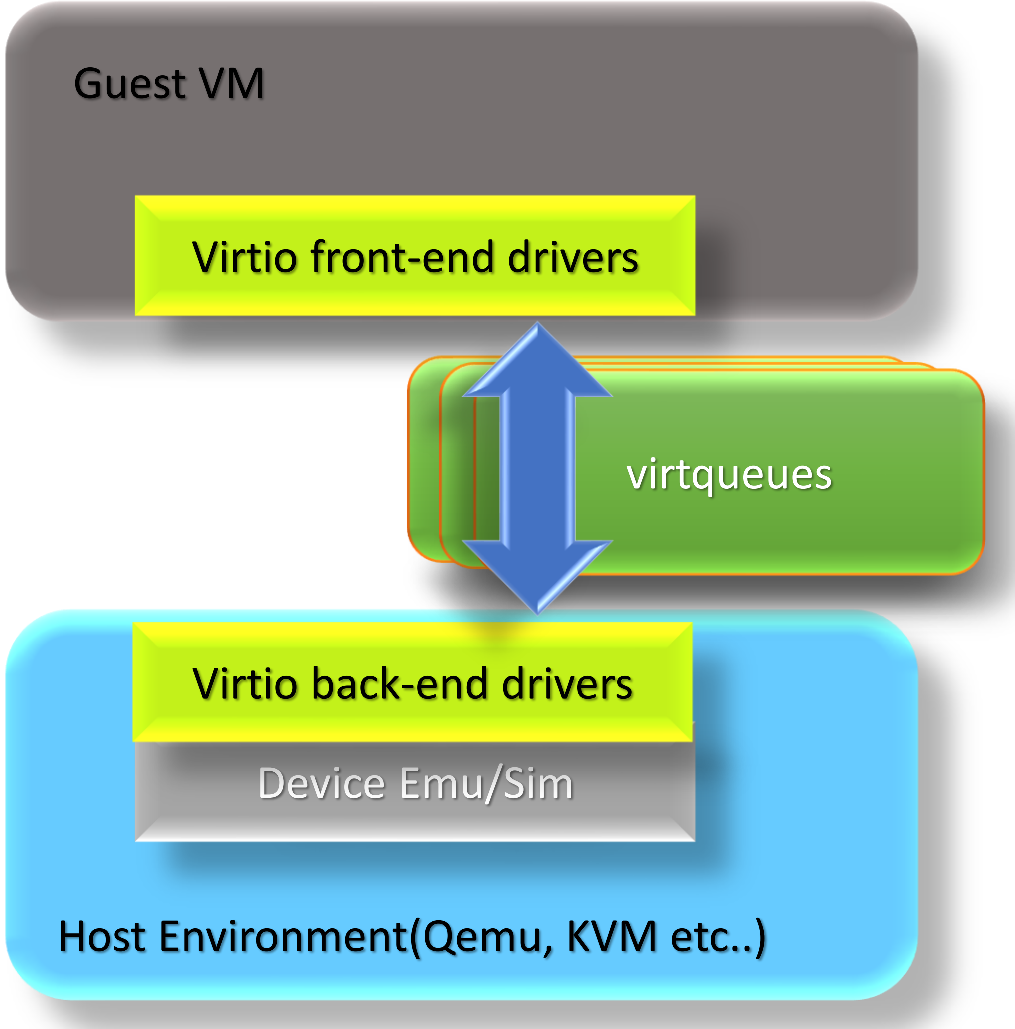
VirtIO Devices Cont..

- Device IDs are used to identify different types of virtio devices.
- All VirtIO devices have a Vendor ID of 0x1AF4, and have a DeviceID between 0x1000 and 0x103F.
- All devices have a common "header" block of registers.
- The Guest Features register is used by the guest VM to communicate the features that the guest VM driver supports.
- The Device Status field is used by the guest VM to communicate the current state of the guest VM driver.

Offset (Hex)	**Name**
00	Device Features
04	Guest Features
08	Queue Address
0C	Queue Size
0E	Queue Select
10	Queue Notify
12	Device Status
13	ISR Status

VirtIO Drivers

- The front-end driver is the device driver installed in the guest OS.
- Accepts I/O requests from the user process and transfer I/O requests to back-end driver.
- The back-end driver resides in the hypervisor and is responsible for accessing the physical device.
- Accepts I/O requests from front - end driver and perform I/O operation via physical device.



VirtIO Transport Layer: VirtQueue

- Virtqueue is a queue of guest’s buffer that host consumes, either by reading them or writing to them.
- Virtqueues are shared in guest physical memory - driver and device access the same page in RAM.
- The descriptors/buffer can be chained.
- Driver to device notifications via doorbell method.
- Device to driver notification via interrupt.
- Virtqueue interface -
  - *add_buf*: expose buffer to other end.
  - *kick*: update after *add_buf*.
  - *get_buf*: get the next used buffer.

VirtIO Transport Layer: VRing

- Vring is a memory mapped region between Host process (Device model) and guest OS.
- Vring is the memory layout of the VQs abstraction.
- Holds the actual data being transferred.
- A virtio device contains one or more VQs.
- VQs has three types of VRings (or areas):
  - Descriptor ring (descriptor area)
  - Available ring (driver area)
  - Used ring (device area)

VirtIO Transport Layer: Desc Area

- Virtio Buffers: Guest drivers (front-end) communicate with hypervisor (back-end) drivers through buffer.
- Guest provides one or more buffers representing the request.

```
struct Buffers[QueueSize]
{
  uint64_t Address; // 64-bit address of the buffer on the guest machine.
  uint32_t Length;  // 32-bit length of the buffer.
  uint16_t Flags;   // 1:linked buffer index;  2: Buffer is write-only.
                    // 4: Buffer contains additional buffer addresses.
  uint16_t Next;    // If flag is set, contains index of next buffer in chain.
}
```

- These buffers are added to virtual queues in memory.

Virtio Transport Layer: Avail and Used Area

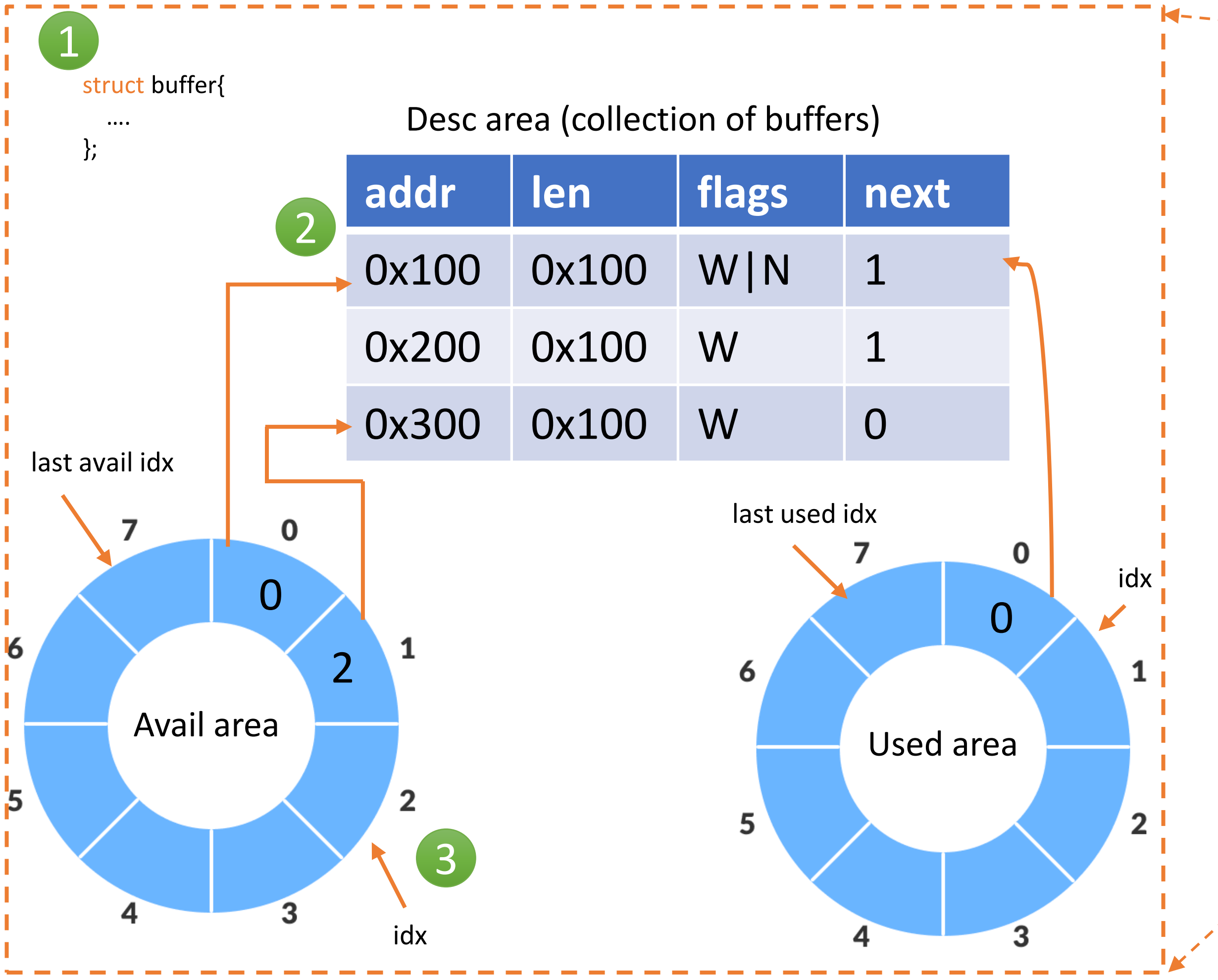
- Avail Area: References to available descriptors in the descriptor ring.

```
struct Available
{
  uint16_t Flags;             // 1: Do not trigger interrupts.
  uint16_t Index;             // Index of the next ring index to be used.
  uint16_t Ring[QueueSize];   // List of available buffer indexes from the Buffers array
}
```

- Used Area: References to *used* descriptor entries on the descriptor ring.

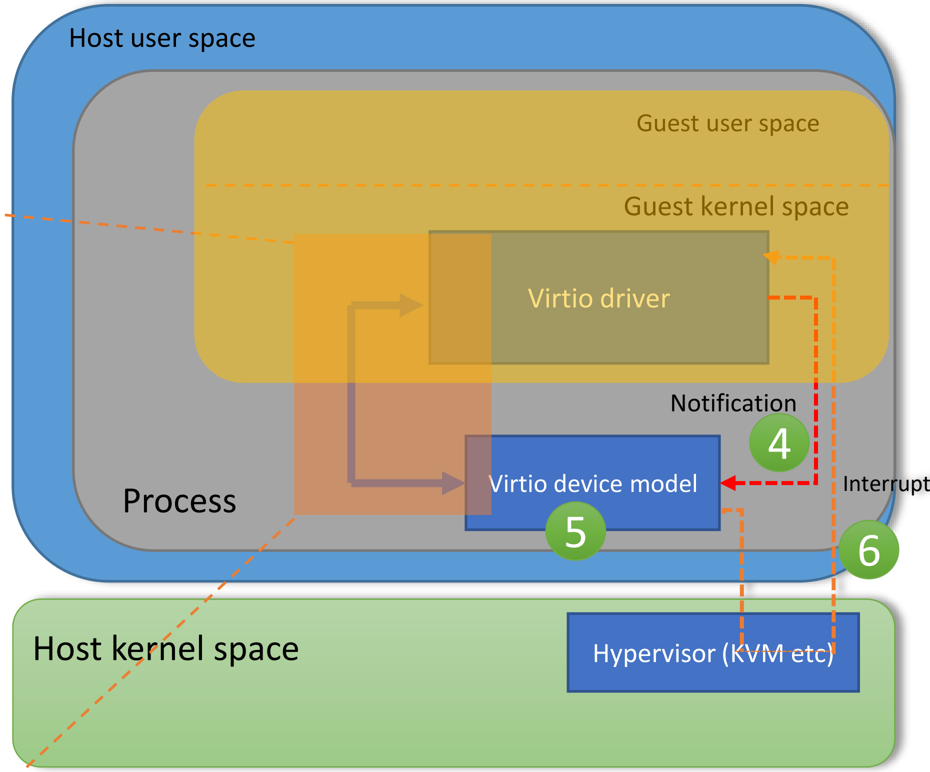
```
struct Used
{
  uint16_t Flags;            // 1: Do not notify device when buffers are added to available ring.
  uint16_t Index;            // Index of the next ring index to be used.  (Last used ring buffer index+1)
  struct Ring[QueueSize]
  {
    uint32_t Index;          // Index of the used buffer in the Buffers array above.
    uint32_t Length;         // Total bytes written to buffer.
  }
  uint16_t AvailEvent;       // Only used if VIRTIO_F_EVENT_IDX was negotiated
}
```

VirtIO Transport Layer:Data Exchange



Desc area (collection of buffers)

addr	len	flags	next
0x100	0x100	W|N	1
0x200	0x100	W	1
0x300	0x100	W	0



1. Allocate/create buffer.
2. Populate descriptor entry.
3. Update avail index.
4. Send notification to device.
5. Process the descriptors.
6. Send interrupt to driver.

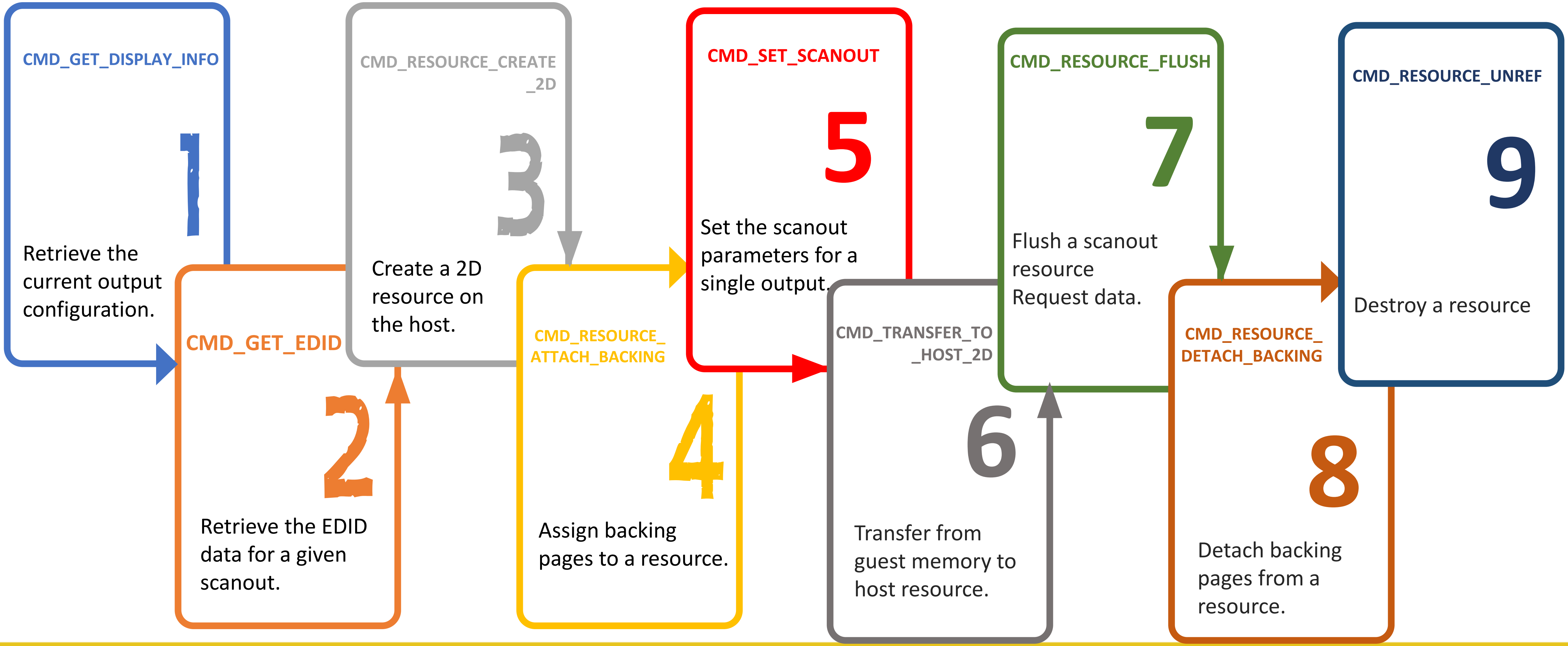
Virtio-GPU

- Operate in 2D mode and in 3D (virgl) mode.
- 3D mode will offload rendering ops to the host gpu.
- Supports two VQs
  - *Controlq:* queue for sending control commands.
  - *Cursorq:* queue for sending cursor updates.
- Feature bits
  - *VIRTIO_GPU_F_VIRGL (0):* virgl 3D mode is supported.
  - *VIRTIO_GPU_F_EDID (1):* EDID is supported.
- Configuration layout
  - **events_read** signals pending events to the driver.
  - **events_clear** clears pending events in the device.
  - **num_scanouts** specifies the maximum number of scanouts supported by the device.

Virtio-GPU - Device Operation

- Create a framebuffer and configure scanout
  - Create a host resource using VIRTIO_GPU_CMD_RESOURCE_CREATE_2D.
  - Allocate a framebuffer from guest ram, and attach it as backing storage to the resource just created, using VIRTIO_GPU_CMD_RESOURCE_ATTACH_BACKING.
  - Use VIRTIO_GPU_CMD_SET_SCANOUT to link the framebuffer to a display scanout.
- Update a framebuffer scanout
  - Use VIRTIO_GPU_CMD_TRANSFER_TO_HOST_2D to update the host resource from guest memory.
  - Use VIRTIO_GPU_CMD_RESOURCE_FLUSH to flush the updated resource to the display.

Device Operation: controlq



CMD_GET_DISPLAY_INFO

1

Retrieve the current output configuration.

CMD_GET_EDID

2

Retrieve the EDID data for a given scanout.

CMD_RESOURCE_CREATE_2D

3

Create a 2D resource on the host.

CMD_RESOURCE_ATTACH_BACKING

4

Assign backing pages to a resource.

CMD_SET_SCANOUT

5

Set the scanout parameters for a single output.

CMD_TRANSFER_TO_HOST_2D

6

Transfer from guest memory to host resource.

CMD_RESOURCE_FLUSH

7

Flush a scanout resource Request data.

CMD_RESOURCE_DETACH_BACKING

8

Detach backing pages from a resource.

CMD_RESOURCE_UNREF

9

Destroy a resource

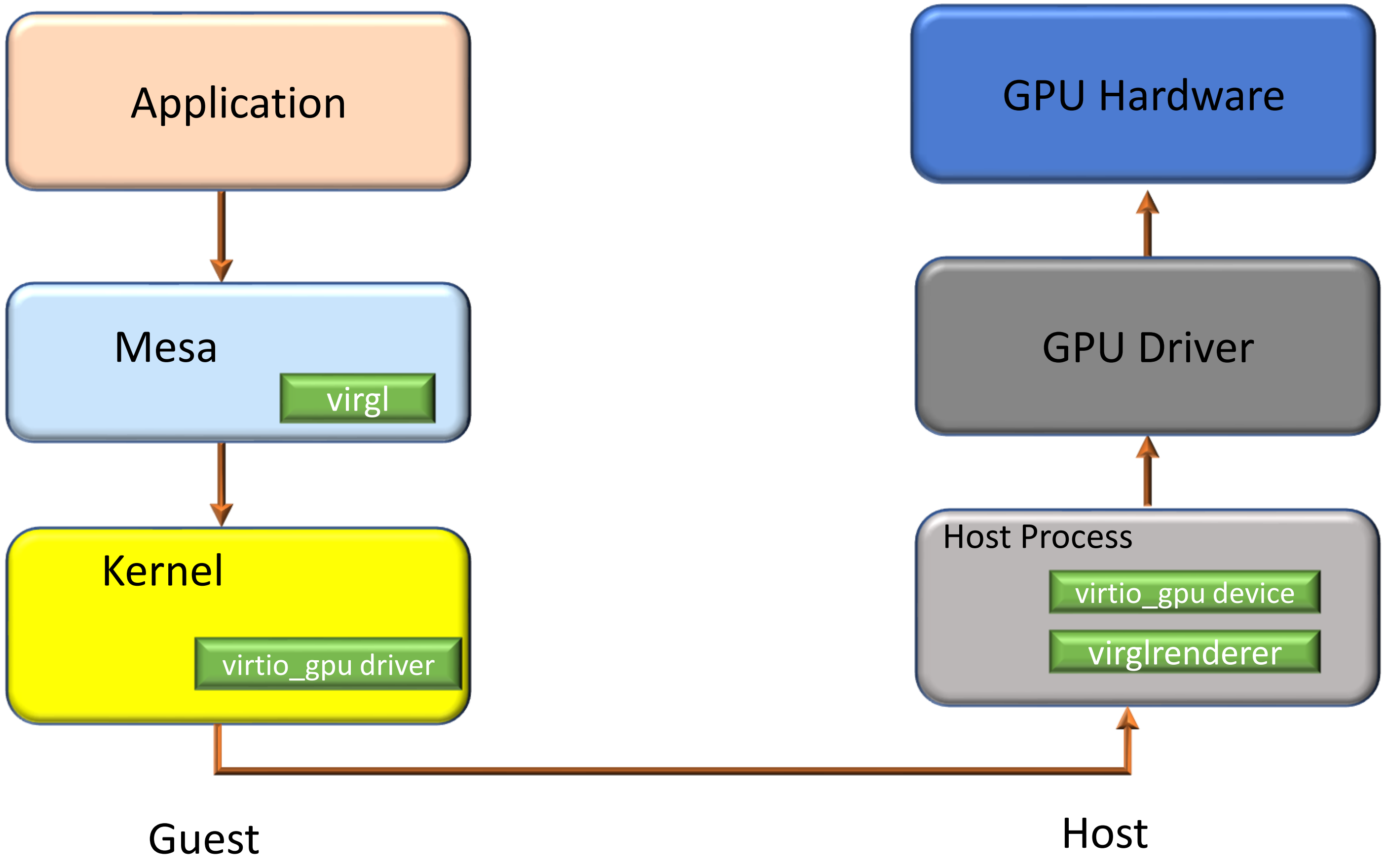
Device Operation: cursorq

- *VIRTIO_GPU_CMD_UPDATE_CURSOR :* Update cursor.
- *VIRTIO_GPU_CMD_MOVE_CURSOR:* Move cursor.

Virglrenderer

- Virglrenderer is a virtual 3D GPU library that
  - enables a virtualized operating system to use the host GPU to accelerate 3D rendering.
- Mesa handing commands are channeled through virtio-gpu on the guest to the host.
- The host gets the raw state (Gallium state) and translates it into an OpenGL form using virglrenderer.
- It is then run as regular OpenGL on the host system.

Virglrenderer Contd..



DEMO

THANK YOU