CSE 511: Operating Systems Design

Lecture 14

Virtualization

Containers

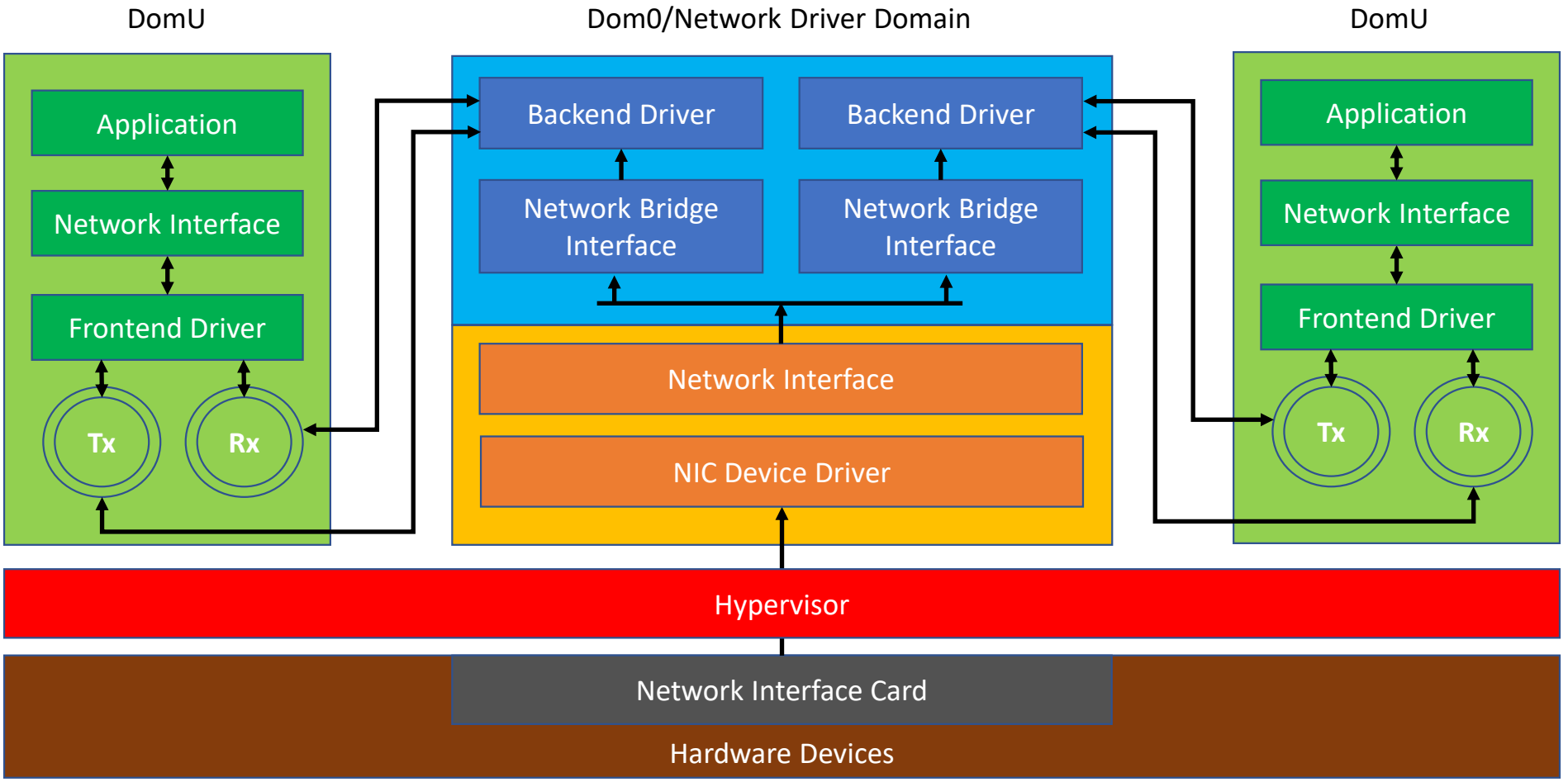
I/O Virtualization

- A hypervisor needs to organize device sharing
- Device drivers are usually executed in
  - a host OS (Type II hypervisors)
  - specialized driver domains
  - a dedicated privileged domain (e.g., Dom0 in case of Xen)
- The privileged domain has unrestricted access to hardware

I/O Virtualization

- All other guest domains
  - Complete emulation of devices
    - the privileged domain will emulate devices and guests will access them as if devices were some real piece of hardware
  - A split driver model
    - drivers consist of two parts that interact with each other using inter-domain communication
  - Direct I/O: device is disabled in the privileged domain and reassigned to a specified guest

I/O Virtualization



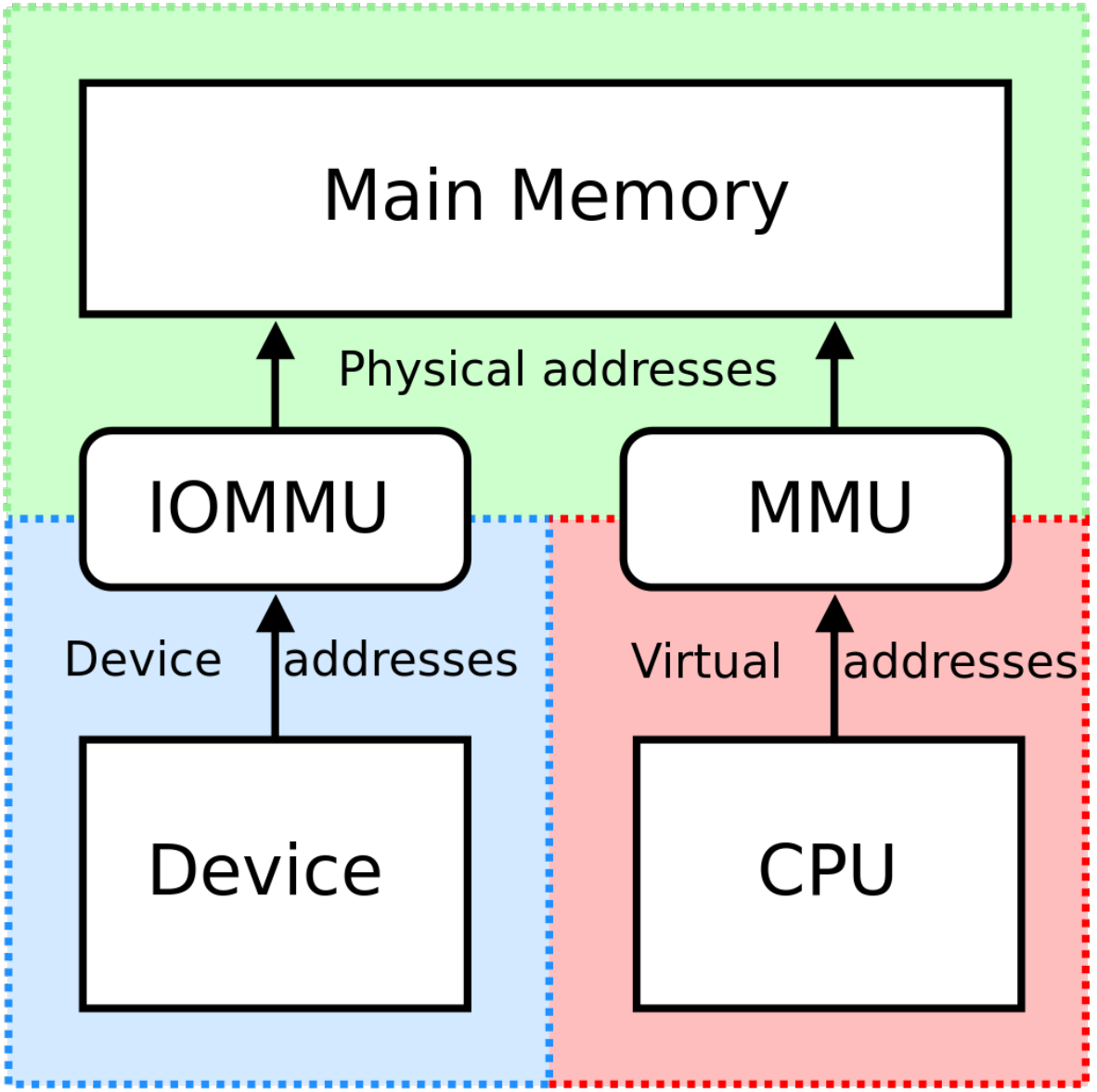
I/O Virtualization

- Drivers can run in the privileged domain (Dom0) or in dedicated driver domains
  - Backend drivers run in Dom0 or driver domains
  - Frontend drivers run in other (guest) domains
- Direct I/O: giving a direct access to a device, “PCI passthrough”
  - Xen/KVM can safely assign access to devices on a PCI bus to guests other than the privileged domain
  - To avoid collisions, the privileged domain excludes those PCI addresses from the I/O space it manages

I/O Virtualization

- To make PCI passthrough safe, the physical presence of an Input/Output memory management unit (IOMMU) is required
  - Also the hypervisor should support that
  - Specific implementation of IOMMU are Intel VT-d, AMD-Vi
- An IOMMU remaps and protects addresses and interrupts used by memory-mapped I/O devices
  - It thus protects from devices and drivers that might make improper use of DMA or interrupts

I/O Virtualization



* The picture is taken from
https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit

Summary: What Do Hypervisors Manage?

- ***Scheduling***: Each guest OS (domain) has its own virtual CPUs (vCPUs)
  - Similar to scheduling of tasks/threads in operating systems
- ***Memory***: Each guest OS has to get a slice of physical memory
- ***Devices***: Full Emulation, Special Drivers, or Direct I/O
- In summary, a hypervisor can be considered as a special “OS” that schedules guests, allocates memory, gives access to devices
  - The difference is that the level of abstraction is much low-level

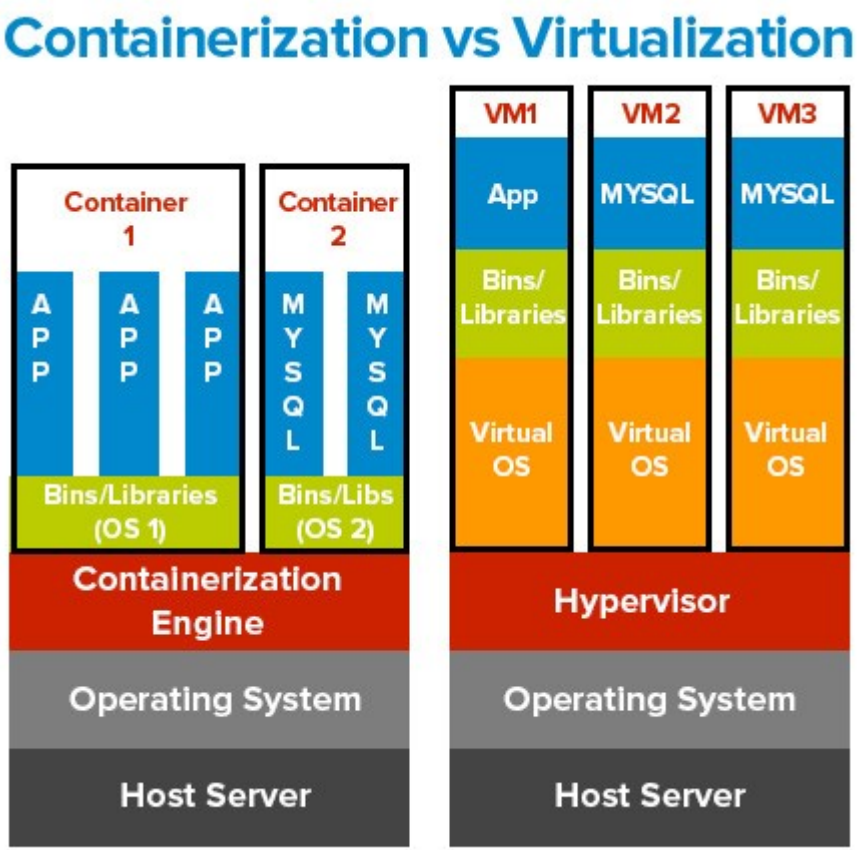
Hypervisor API

- It depends on the hypervisor
  - Xen supports “hypercalls”, which are similar to system calls but used in the hypervisor context
  - A guest kernel can issue hypercalls
    - Avoids trapping and emulation, requests can be batched together to reduce costs, etc
- CPU support
  - Intel VT-x: vmcall, AMD-V: vmmcall
- VT-x and AMD-V diverge but provide comparable capabilities
  - Xen supports both (known as VMX: Intel and SVM: AMD)

Containers

- Containerization is a feature that enables hosting multiple user-space instances
  - They appear as separate OS instances
  - Got some attention recently, especially with Docker
  - Docker, LxC (Linux Containers), etc – are examples of modern containers
- Not a completely new idea, has been around for a while in one or the other form
  - ***Examples***: FreeBSD jails, chroot jails

Containerization vs. Virtualization



* Taken from
https://www.smartfile.com/blog/what-is-containerization-and-has-it-killed-virtualization/

Advantages

- Instances are lightweight and easy to deploy
  - They also mostly have everything including an “operating system” environment
  - Fast to launch
- Presumably low overhead
- Resource allocation
  - An application will see only allocated resources
- No need for a hypervisor and an extra layer of virtualization
  - Using regular system calls
  - Just one level of scheduling
  - No virtual machines, virtual CPUs, etc

Disadvantages

- Lower flexibility
  - Cannot host a different operating system
    - Has to use the same kernel
- Lower isolation
  - Only isolating user-space environments
  - What if a device driver in the kernel fails?
- Security implications
  - A reduced number of protection domains (e.g., the Meltdown bug did not affect VMs)
  - Lower isolation also typically means lower security

Features

- File system isolation, disk quotas, etc
- File-level copy-on-write (CoW) support
  - Some rely on CoW file systems such as ZFS
- I/O rate limiting
- Memory limits
- CPU quotas
- Network isolation
- Root privilege isolation

Adoption

- Example: Kubernetes – running containers on a cluster of Amazon EC2 compute instances
- Lightweightness and easiness were the biggest selling points of containers
- But see recent papers, e.g., “My VM is Lighter (and Safer) than your Container” (SOSP’17)

Unikernels vs. Containers

- Both are lightweight and both enable faster reboots
- Containers can run many application in an ordinary operating system
- Containers depend on the existing OS, only specialize the user-space portion
- Unikernels may offer better isolation and better flexibility
  - But they are limited to a single application

Microvisors

- See “The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors”, APSys 2010
- Microkernels and hypervisors provide similar abstractions
  - Requirements for both can be met with a single set of abstractions
  - The OKL4 “microvisor” is effectively a Type-I hypervisor, which a variant of the L4 microkernel
  - Has effectively its own variant of “paravirtualization”
  - OKL4 is deployed on over 2 billion mobile phones

Microkernels: L4Linux

- Running Linux on top of the L4 microkernel
  - Using the L4 API

* Taken from https://l4linux.org/