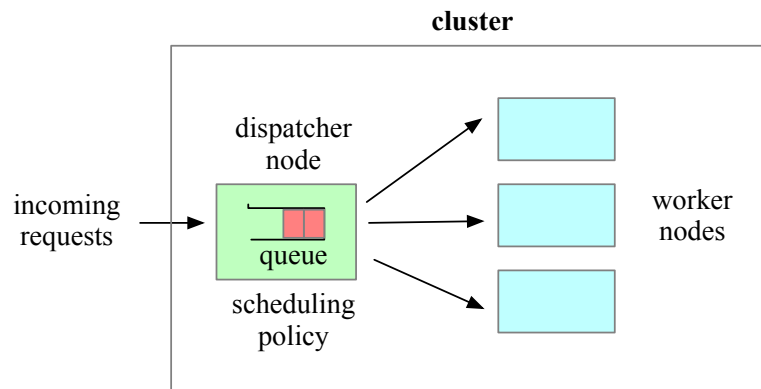


Part 3: Cluster Scheduling

- Scheduling tasks on to a cluster of servers
 - Machines are cheap, no need to rely on idle PCs anymore
 - Use a cluster of powerful servers to run tasks
 - User requests sent to the cluster (rather than a idle PC)
- **Interactive** applications
 - Web servers use a cluster of servers
 - “Job” is a single HTTP request; optimize for response time
- **Batch** applications
 - Job is a long running computation; optimize for throughput

Typical Cluster Scheduler

- Dispatcher node assigns queued requests to worker nodes as per a scheduling policy



Scheduling in Clustered Web Servers

- Distributed scheduling in large web servers
 - N nodes, one node acts as load balancer/dispatcher
 - other nodes are replica worker nodes (“server pool”)
- Requests arrive into queue at load balancer node
 - Dispatcher schedules request onto an worker node
- How to decide which node to choose?
 - Scheduling policies: least loaded, round robin
 - Weighted round robin when servers are heterogeneous
- Session-level versus request-level load balancing
 - Web server maintain session state for client (e.g., shopping cart)
 - Perform load balancing at session granularity
 - All requests from client session sent to same worker

Scheduling Batch Jobs

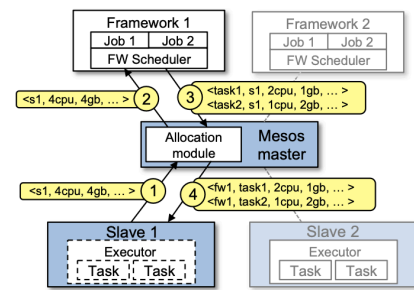
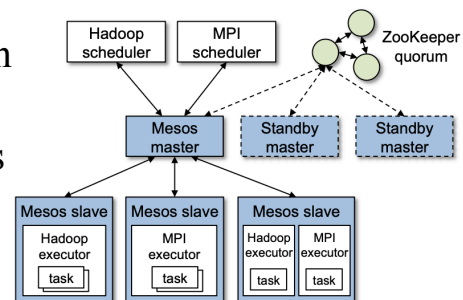
- Batch jobs are non-interactive tasks
 - ML training, data processing tasks, simulations
- Batch scheduling in a server cluster
 - Users submit job to a queue, dispatcher schedules jobs
- SLURM: Simple Linux Utility for Resource Management
 - Linux batch scheduler; runs on > 50% supercomputers
 - Nodes partitioned into groups; each group has job queue
 - Specify size, time limits, user groups for each queue
 - Example: short queue, long queue
 - Many policies: FCFS, priority, gang scheduling
 - Exclusive or shared access to nodes (e.g., MPI jobs)
- Others: SunGridEngine, DQS, Load Leveler, IBM LSF

Mesos Scheduler

- Mesos: Cluster manager and scheduler for multiple frameworks
 - Cluster typically runs multiple frameworks: batch, Spark, ...
 - Statically partition cluster, each managed by a scheduler
 - Mesos: fine-grain server sharing between frameworks
- Two-level approach: allocate resources to frameworks, framework allocates resources to tasks
- **Resource Offers**: bundle of resources offered to framework
 - Framework can accept or reject offer
 - Higher-level policy (e.g., fair share) governs allocation; resource offers used to offer resources
 - Framework-specific scheduling policy allocates to tasks
 - Framework can not ask for resources; only accept/reject resource offers (Paper shows this is sufficient).

Mesos Scheduler

- Four components: **coordinator**, Mesos **worker**, framework **scheduler**, **executor** on server nodes
- Step 1: worker node (6 core, 6GB) becomes idle, reports to coordinator
- Step 2: Coordinator invokes policy, decides to allocate to Framework 1. Sends resource offer
- Step 3: Framework accepts, scheduler assigns task 1 (2C, 2GB) and task 2 (2C, 3GB)
- Step 4: Coordinator sends tasks to executor on node
- Unused resources (2C, 1GB): new offer

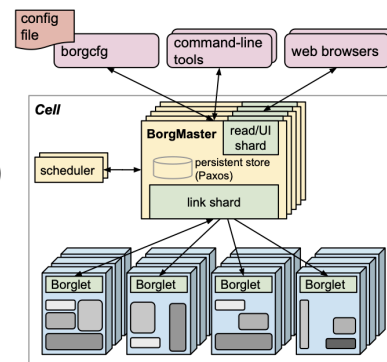


Borg Scheduler

- Google's cluster scheduler: scheduling at very large scales
 - run hundreds of thousands of concurrent jobs onto tens of thousands of server
 - Borg's ideas later influenced *kubernetes*
- Design Goals:
 - hide details of resource management and failures from apps
 - Operate with high reliability (manages gmail, web search, ..)
 - Scale to very large clusters
- Designed to run two classes: interactive and batch
 - Long running interactive jobs (prod job) given priority
 - Batch jobs (non-prod jobs) given lower priority
 - % of interactive and batch jobs will vary over time

Borg Scheduler

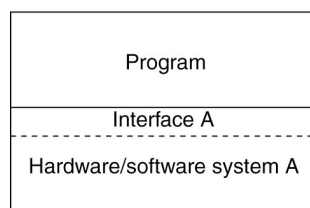
- Cell: group of machines in a cluster (~10K servers)
- Borg: matches jobs to cells
 - jobs specify resource needs
 - Borg finds a cell/machine to run a job
 - job needs can change (e.g., ask for more)
- Use resource reservations (“alloc”)
 - alloc set: reservations across machines
 - Schedule job onto alloc set
- Preemption: higher priority job can preempt a lower priority job if there are insufficient resources
- Borg Master coordinator: replicated 5 times, uses paxos to
- Priority queue to schedule jobs: uses best-fit, worst-fit



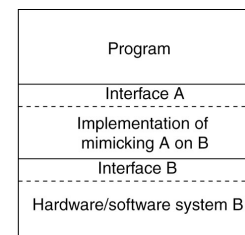
Virtualization

- Part 1: Basics of virtualization
- Part 2: Hypervisors
- Part 3: Virtualizing Resources

Part 1: Virtualization



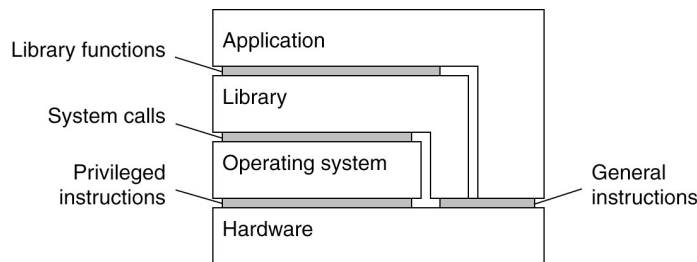
(a)



(b)

- Virtualization: extend or replace an existing interface to mimic the behavior of another system.
 - Introduced in 1970s: run legacy software on newer mainframe hardware
- Handle platform diversity by running apps in VMs
 - Portability and flexibility

Types of Interfaces



- Different types of interfaces
 - Assembly instructions
 - System calls
 - APIs
- Depending on what is replaced /mimiced, we obtain different forms of virtualization

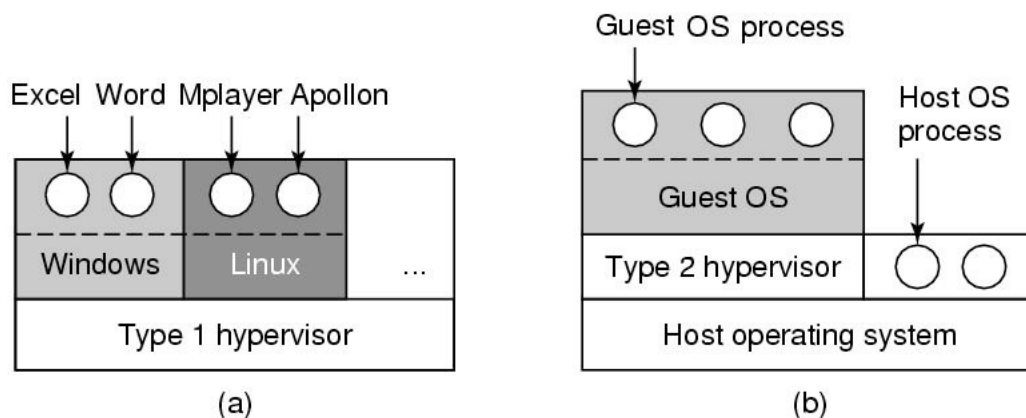
Types of Virtualization

- Emulation
 - VM emulates/simulates complete hardware
 - Unmodified guest OS for a different PC can be run
 - Bochs, VirtualPC for Mac, QEMU
- Full/native Virtualization
 - VM simulates “enough” hardware to allow an unmodified guest OS to be run in isolation
 - Same hardware CPU
 - IBM VM family, VMWare Workstation, Parallels, VirtualBox

Types of virtualization

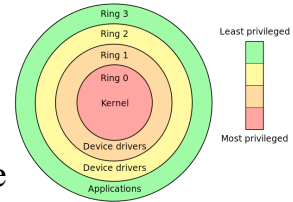
- Para-virtualization
 - VM does not simulate hardware
 - Use special API that a modified guest OS must use
 - Hypercalls trapped by the Hypervisor and serviced
 - Xen, VMWare ESX Server
- OS-level virtualization
 - OS allows multiple secure virtual servers to be run
 - Guest OS is the same as the host OS, but appears isolated
 - apps see an isolated OS
 - Solaris Containers, BSD Jails, Linux Vserver, Linux containers, Docker
- Application level virtualization
 - Application is gives its own copy of components that are not shared
 - (E.g., own registry files, global objects) - VE prevents conflicts
 - JVM, Rosetta on Mac (also emulation), WINE

Part 2: Hypervisors



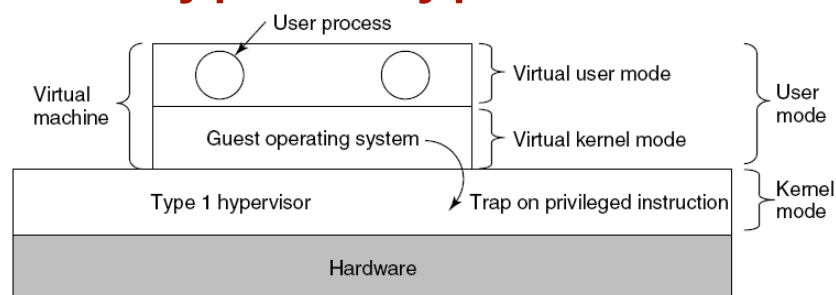
- Type 1: hypervisor runs on “bare metal”
- Type 2: hypervisor runs on a host OS
 - Guest OS runs inside hypervisor
- Both VM types act like real hardware

How Virtualization works?



- CPU supports kernel and user mode (ring0, ring3)
 - Set of instructions that can only be executed in kernel mode
 - I/O, change MMU settings etc -- *sensitive instructions*
 - Privileged instructions: cause a trap when executed in kernel mode
- Result: type 1 virtualization feasible if sensitive instruction subset of privileged instructions
- Intel 386: ignores sensitive instructions in user mode
 - Can not support type 1 virtualization
- Recent Intel/AMD CPUs have hardware support
 - Intel VT, AMD SVM
 - Create containers where a VM and guest can run
 - Hypervisor uses hardware bitmap to specify which inst should trap
 - Sensitive inst in guest traps to hypervisor

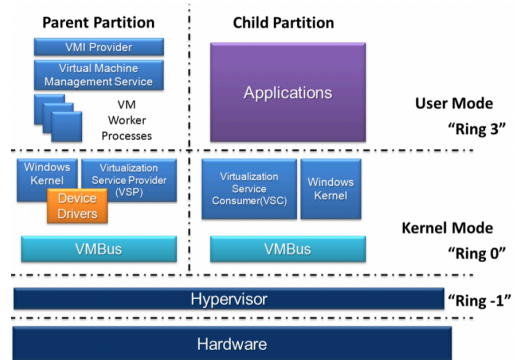
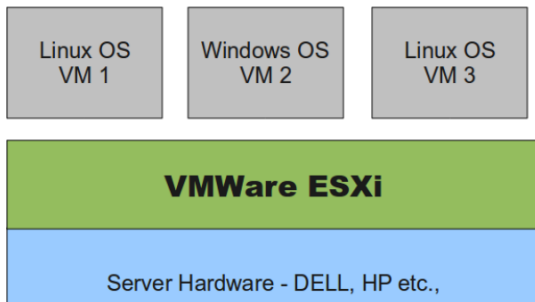
Type 1 hypervisor



- Unmodified OS is running in user mode (or ring 1)
 - But it thinks it is running in kernel mode (*virtual kernel mode*)
 - privileged instructions trap; sensitive inst-> use VT to trap
 - Hypervisor is the “real kernel”
 - Upon trap, executes privileged operations
 - Or emulates what the hardware would do

Type 1 Hypervisors Examples

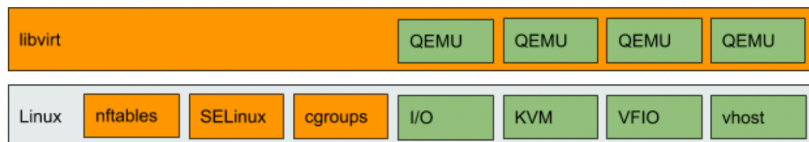
- VMWare ESX Server
 - Specialized OS kernel designed to run virtual machines on bare metal
- Hyper-V Windows hypervisor
 - parent partition runs windows server
 - child partitions run VMs



<https://en.wikipedia.org/wiki/Hyper-V>

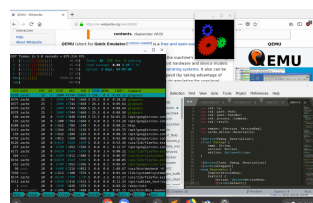
Type 1 Hypervisors Examples

- Linux KVM (“kernel virtual machine”)
 - Kernel infrastructure (driver) for range of VMMs
 - One example: QEMU (vmm) + libvirt on top of lvm



<https://www.redhat.com/en/blog/all-you-need-know-about-kvm-userspace>

- another example: crosvm for Chrome OS to run linux apps.

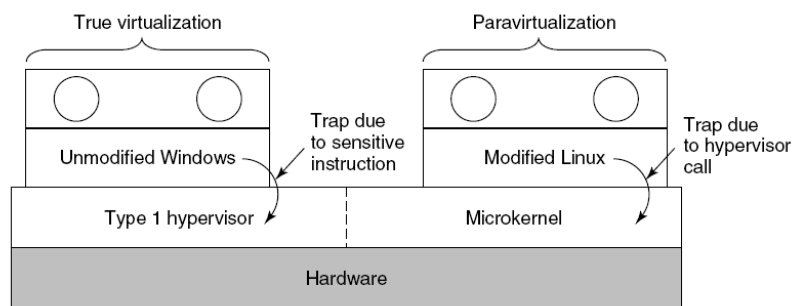


https://chromium.googlesource.com/chromiumos/docs/+master/containers_and_vms.md#Crostini

Type 2 Hypervisor

- VMWare example
 - Upon loading program: scans code for basic blocks
 - If sensitive instructions, replace by Vmware procedure
 - Binary translation
 - Cache modified basic block in VMWare cache
 - Execute; load next basic block etc.
- Type 2 hypervisors work without VT support
 - Sensitive instructions replaced by procedures that emulate them.
- Examples: VirtualBox, Vmware workstation/fusion, Parallels Desktop

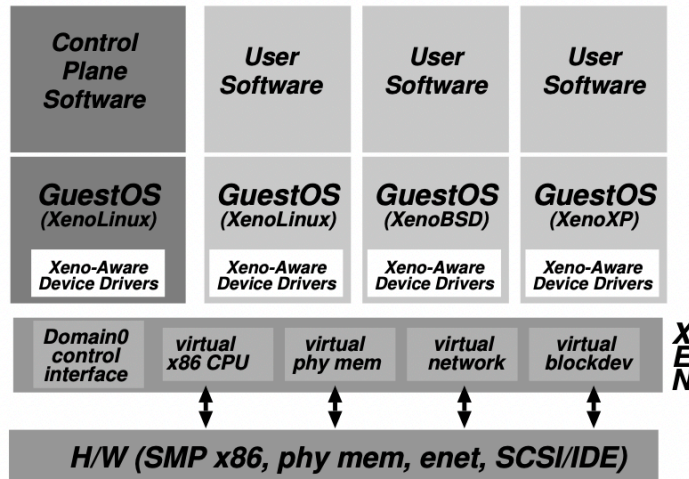
Paravirtualization



- Both type 1 and 2 hypervisors work on unmodified OS
- Paravirtualization: modify OS kernel to replace all sensitive instructions with hypercalls
 - OS behaves like a user program making system calls
 - Hypervisor executes the privileged operation invoked by hypercall.

Xen Hypervisor

- Linux Type 1 hypervisor with no special hardware support
 - Requires modified kernel, but can run unmodified apps
 - Dom-0 runs control plane; each guestOS runs in its own domain/VM



See Paper:
Xen and art
of virtualization

UMassAmherst

Lecture 7, page 21

Part 3: Virtualizing Other Resources Memory virtualization

- OS manages page tables
 - Create new pagetable is sensitive -> traps to hypervisor
- hypervisor manages multiple OS
 - Need a second shadow page table
 - OS: VM virtual pages to VM's physical pages
 - Hypervisor maps to actual page in shadow page table
 - Two level mapping
 - Need to catch changes to page table (not privileged)
 - Change PT to read-only - page fault
 - Paravirtualized - use hypercalls to inform

UMassAmherst

CS677: Distributed OS

Lecture 7, page 22

I/O Virtualization

- Each guest OS thinks it “owns” the disk
- Hypervisor creates “virtual disks”
 - Large empty files on the physical disk that appear as “disks” to the guest OS
 - Hypervisor converts block # to file offset for I/O
 - DMA need physical addresses
 - Hypervisor needs to translate
- Stored as virtual disk or vmdk files

Virtual Appliances & Multi-Core

- Virtual appliance: pre-configured VM with OS/ apps pre-installed
 - Just download and run (no need to install/configure)
 - Software distribution using appliances
- Multi-core CPUs
 - Run multiple VMs on multi-core systems
 - Each VM assigned one or more vCPU
 - Mapping from vCPUs to physical CPUs
- Today: Virtual appliances have evolved into docker containers

Use of Virtualization Today

- Data centers:
 - server consolidation: pack multiple virtual servers onto a smaller number of physical server
 - saves hardware costs, power and cooling costs
- Cloud computing: rent virtual servers
 - cloud provider controls physical machines and mapping of virtual servers to physical hosts
 - User gets root access on virtual server
- Desktop computing:
 - Multi-platform software development
 - Testing machines
 - Run apps from another platform