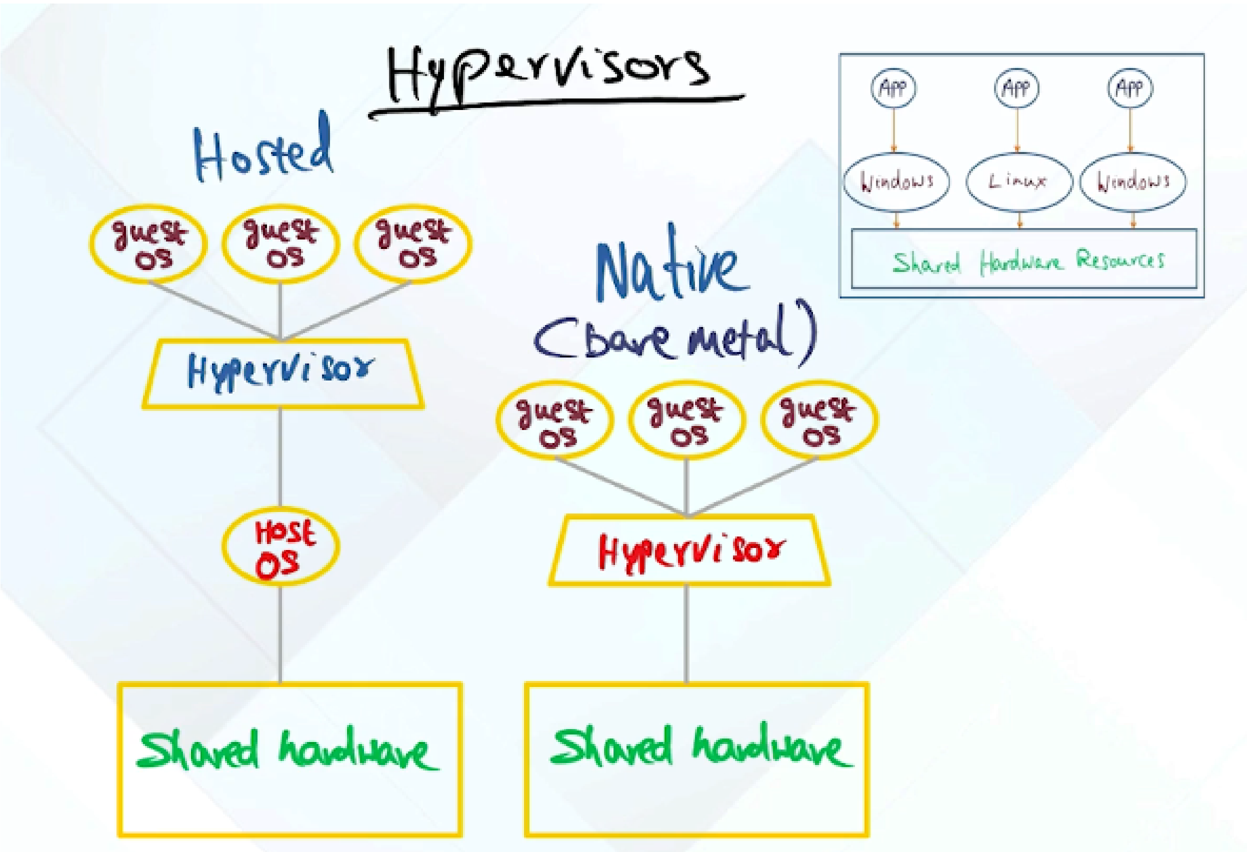
Introduction to Virtualization Technologies

Introduction

1. Virtualization dates back to IBM systems in the 60s and 70s
    - Resurrected in the 80s and 90s at universities due to a need for resource consolidation
    - Taken a new form under cloud computing

Virtualization Technologies

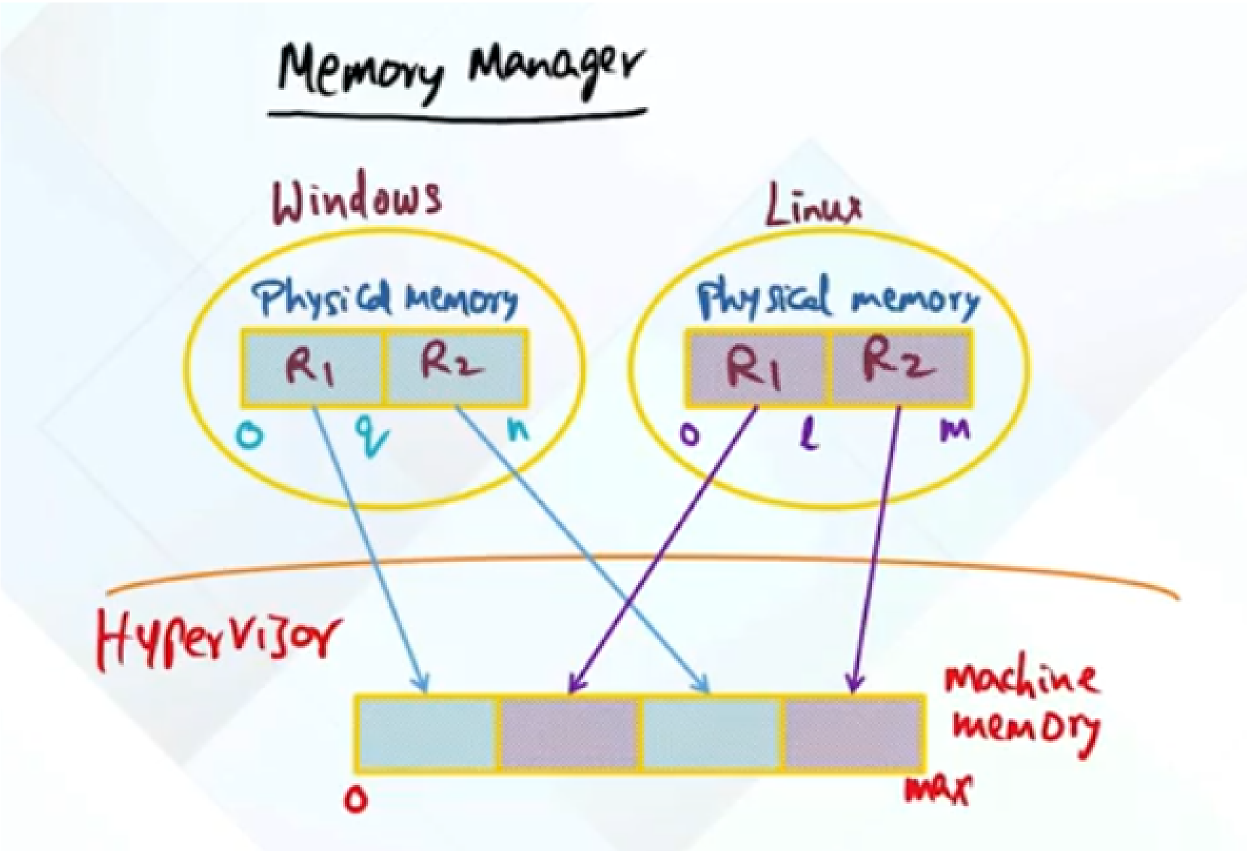
1. Origins of Virtualization
    - IBM VM 370 (70s)
    - Microkernels (late 80 and 90s)
    - Extensibility of OS (90s)
    - SIMOS (late 90s)
        - Simulate processor design in the context of the OS
    - Xen and VMWare (early 2000s)
    - Now…
        - Accountability for usage in data centers
2. Hypervisors
    - Run multiple OS on same hardware resources
    - Hypervisor: Thin layer that sits between guest OS and hardware resources to moderate the usage of these resources
    - Bare metal hypervisor: Running directly on bare metal
    - Hosted hypervisor: Hypervisor runs as a process on top of the host OS
    - Full virtualization: Hypervisor runs an unchanged OS binary
        - Pro: Vendors don't have to change their software
        - Con: Performance disadvantages
    - Para virtualization: Hypervisor runs a modified OS binary that knows it is virtualized (modifications 1-2% of codebase)
        - Pro: Can alleviate performance concerns
        - Con: Requires vendors to participate
3. Big Picture
    - What needs to be done?
        - Virtualize hardware
            * Memory
            * CPU
            * Devices
        - Effect data and control between guests and hypervisor



Hypervisors

Memory Virtualization

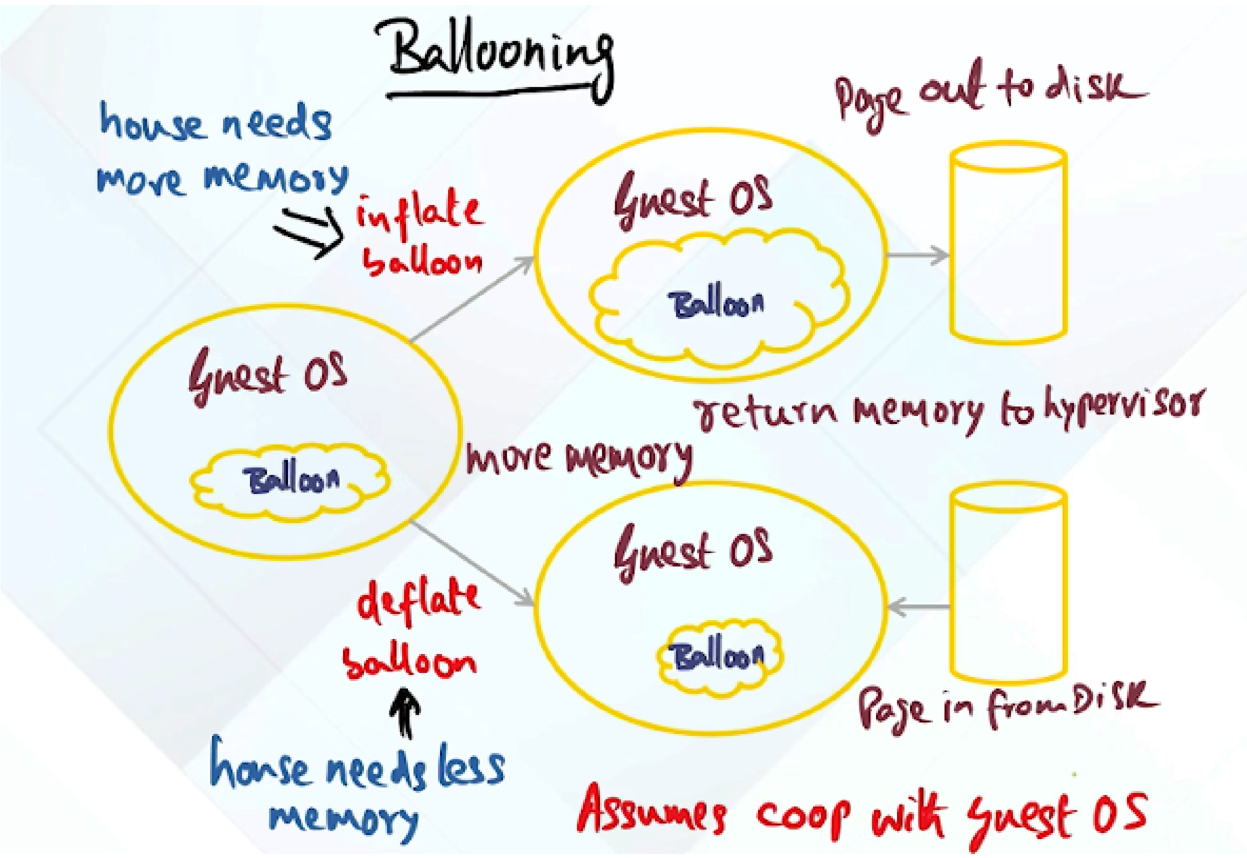
1. Memory manager
    - Every OS supports virtual memory for processes running in the OS
        - Physical memory backs up the virtual memory
        - In a virtualized context, physical memory is also an illusion
        - "Machine memory" in a virtualized world
    - Machine memory is contiguous, physical memory is not
        - OS still wants the illusion of contiguous memory
2. Dynamically increasing memory
    - Guest OS might need more physical memory to handle processes that are currently running within it
    - Guest OS goes to the hypervisor and requests more memory
        - If hypervisor has machine memory, it can give it to the guest OS, but it is also possible that there is no additional memory available
            * Can't necessarily take memory from another OS



Memory Virtualization

Ballooning

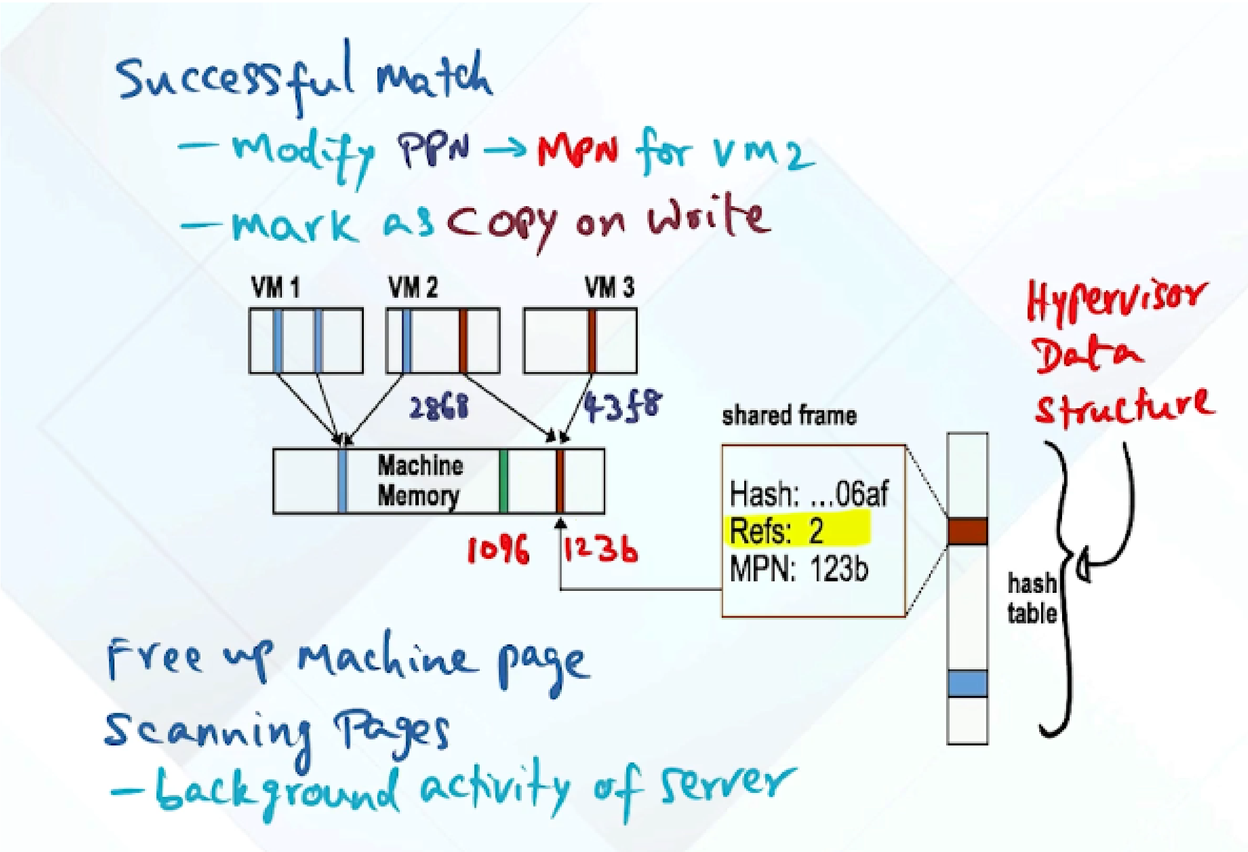
1. Understanding between hypervisor and guest OS in either full or para virtualized environment
   - Use a technique called ballooning to manage memory
   - Guest OS installs a device driver to run on the hypervisor
     - Device driver inflates the balloon when the hypervisor needs more memory
     - This returns memory to the hypervisor, which can give it to the other guest OS
       * Some data may need to be paged out to disk
     - Can also deflate the balloon when the hypervisor needs less memory
       * Can page in from disk
   - Balloon driver assumes cooperation between hypervisor and guest OS
   - Hypervisor has policies to determine where to take memory from



Ballooning

Sharing Memory Across VMs

1. The same running process in two different VMs can share memory
   - Page tables are internal to guest OS, of which the hypervisor has no knowledge
2. VM Oblivious Page Sharing
   - Look at page coming from disk and hash the contents
   - Hypervisor maintains a hint frame (hash table)
     - Hash
     - VM
     - PPN
     - MPN
   - If a page fault occurs and the hash matches a hint frame, we might be able to map it to physical memory that already exists
     - Do a full comparison of the pages
   - Mark page entries in VM as copy on write to prevent sharing modified pages
   - Don't have to do anything with respect to the VMs themselves
   - Scanning pages takes some cycles, so it's done as a background activity on the server



Sharing Memory Across VMs

Memory Allocation Policies

1. Pure share based approach
2. Working-set based approach
3. Dynamic idle-adjusted shares approach
    - Tax idle pages more than active pages
4. Reclaim most idle memory
    - Allow for sudden working set increases

CPU Virtualization

1. Hypervisor wants to provide the illusion that each guest OS owns the CPU despite it being a shared hardware resource
2. Illusion of ownership of CPU for each VM
    - Proportional share: Proportional to the negotiated SLAs
    - Fair share
3. Delivering events to parent guest OS
    - Address translation happens on every memory access
    - Process on guest OS makes a system call
    - Page fault occurs
    - Excecption (divide by zero, segmentation fault)
    - External interrupts
    - Events are delivered as software interrupts
        - In a para virtualized environment, hypervisor can provide APIs that the guest OS can use

        * Guest OS knows it isn't in control of the hardware
    - If guest OS wants to switch from user mode to kernel mode, it can't because, as far as the hypervisor is concerned, it is a process running in user mode
        * Trap into the hypervisor, hypervisor can emulate what the guest OS wants to do

Device Virtualization

1. Full virtualization
    - Trap and emulate: Guest OS thinks it has privilege to do whatever it wants, but it doesn't. Traps into hypervisor which emulates the event
2. Para virtualization
    - More opportunity for virtualization
    - Can make specific requests to the hypervisor
    - Shared buffers between guest OS and hypervisor

Control Transfer

1. Full virtualization
    - Implicit (traps) guest OS -> hypervisor
    - Software interrupts (events) hypervisor -> guest
2. Para virtualization
    - Explicit (hypercalls) guest OS -> hypervisor
    - Software interrupts (events) hypervisor -> guest
        - Guest has control via hypercalls on when event notifications are delivered
            * Can improve performance

Data Transfer

1. Full virtualization
    - Data transfer is implicit
        - In OS, an I/O transfer might need to be copied between user and kernel space
    - Guest OS thinks it can do this itself, but it can't, so it traps
2. Para virtualization
    - Data transfer is explicit -> opportunity to innovate
    - Xen's asynchronous I/O rings are used to move data between the guest OS and hypervisor
        - Request producer and consumer hold separate pointers to the same I/O ring buffer
        - Response producer and consumer are also separate pointers

Containers

1. Containers
    - Light form of resource virtualization
        - In contrast to a full blown virtual machine of a hypervisor
    - Multiple containers on top of the same kernel
        - Illusion to each container that they are the only one using the hardware resources
        - Kernel services (e.g., I/O subsystems) not replicated in each container
2. Advantages of Containers
    - Fault isolation
        - No leaks across containers
    - Resource isolation
        - Performance guarantees for each container
    - Security isolation
        - Configuration and namespace independence
3. Containers vs VMs
    - Performance

    - Order of magnitude faster to boot up container compared to spinning up VMs
  - More manageable
    - Fewer OSes to manage
    - Easier patching of OS and apps
    - Improved resource utilization

VM Migration

1. VM Migration
   - Why?
     - Rebalancing resources
     - Upgrades
   - Challenges
     - Minimizing downtime
     - Avoiding disruption of active services
   - Migration options
     - Push phase
     - Stop and copy phase
     - Pull phase
2. Generic steps in migration
   - Pre-migration
     - Reservation
   - Iterative pre-copy
     - Reduce the downtime
   - Stop and copy
     - Suspend VM on source machine
     - Set up routing redirection
     - Sync all remaining VM state
   - Commitment
     - Release source machine resources
   - Activation
     - Start VM on destination resources

Conclusion

1. Technology pertaining to virtualizating data center resources is more involved than can be described in a single lecture