Virtual Machine Monitor

CS 4230

Jay Urbain, Ph.D.

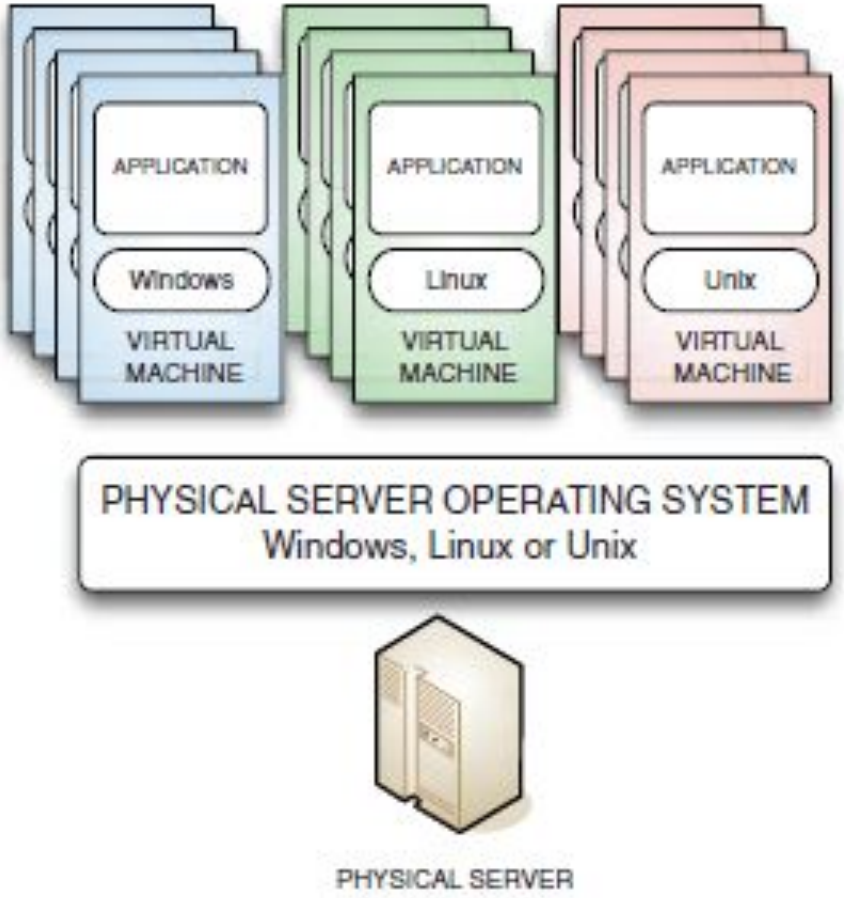
Credits:

- Zen and the Art of Virtualization, Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfiled.
- Virtualization in education". IBM. October 2007. Retrieved 6 July 2010.
- Goldberg, Robert P. (February 1973) (PDF). *Architectural Principles for Virtual Computer Systems*. Harvard University.

Virtual Machine Monitor (VMM)

- *Virtual Machine Monitor (VMM)* allows multiple operating systems to run concurrently on a host computer.
- VMM is also called *hypervisor* (above supervisor) was first used in 1965, referring to software available for the IBM 360/65.
- Each of the operating systems being hosted is referred to as a *guest*.

Virtual Machine Monitor (VMM)



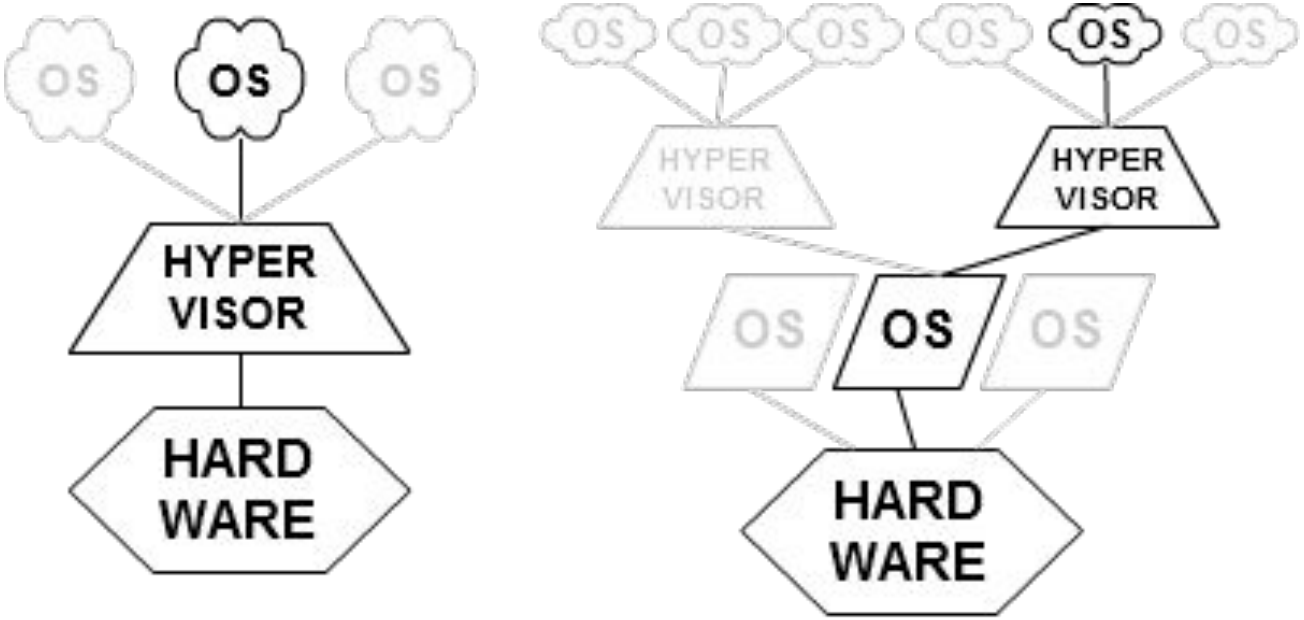
Virtual server model

Virtual Machine Monitor (VMM)

- The *hypervisor* presents to the *guest* operating systems a *virtual operating platform,* and *monitors* the execution of the guest operating systems.
- Multiple instances of a variety of OS's may share the same server's *virtualized hardware resources*.
- *Hypervisor* is often used to describe the interface provided by the specific *cloud computing* functionality:
  - *Infrastructure as a service(IaaS).*

Types of Hypervisor

Robert P. Goldberg classifies two types of *hypervisor*:



VMWare

VirtualBox

Hypervisor Type 1

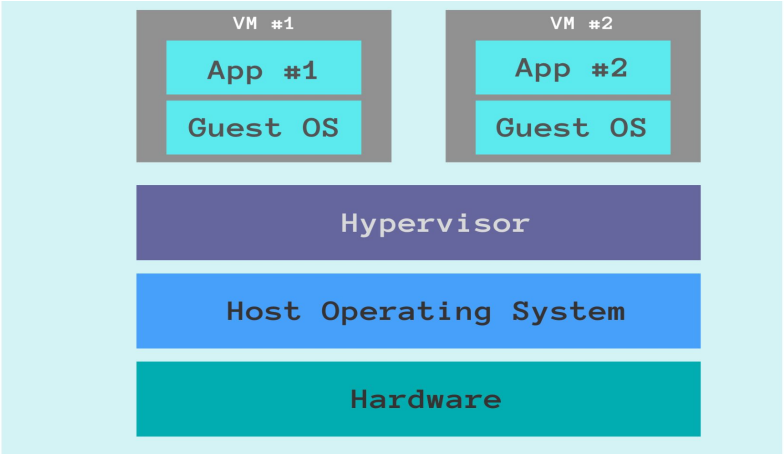
- ***Type 1*** (or *native*, *bare metal*) *hypervisors* run directly on the host's hardware to control the hardware and to monitor guest operating systems.
  - A guest operating system thus runs on another level above the *hypervisor*.
  - Represents the classic implementation of virtual machine architectures; the original hypervisor was CP/CMS, developed at IBM in the 1960s, ancestor of IBM's z/VM.
  - Modern equivalents of this is the VMware *ESXi*, Citrix *XenServer,* or Microsoft *Hyper-V*.

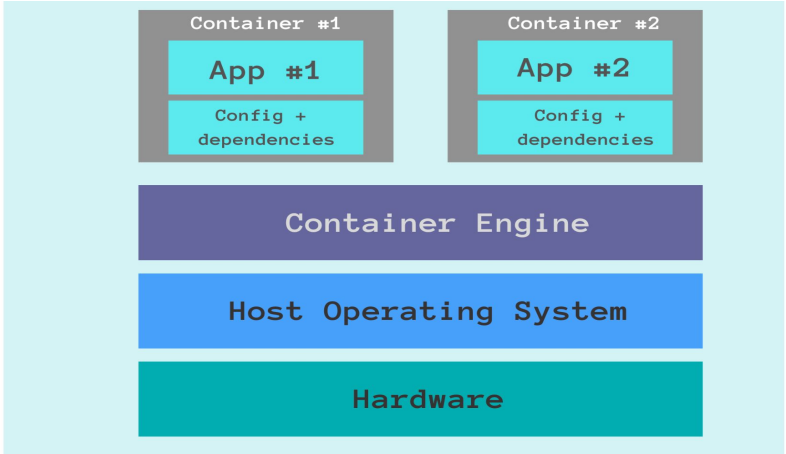
Hypervisor Type 2

- *HypervisorType 2* (or *hosted*) hypervisors run within a conventional operating system environment.
  - The *hypervisor* layer has a distinct *second software level*, guest operating systems run at the third level above the hardware.
    - *I.e., Type 1 hypervisor runs directly on the hardware; a Type 2 hypervisor runs on another operating system, such as Linux.*
  - Example: VirtualBox

Note: Docker vs. VM

- Docker and VMs are both technologies used for virtualization, but they work in different ways and serve different purposes.
- Virtual machines emulate the hardware of a physical computer and allow multiple operating systems to run on a single physical host.
- Docker, on the other hand, is a containerization platform that allows applications to be packaged with their dependencies and run in an isolated environment.
- Docker containers share the same kernel as the host operating system, but have their own file system, networking, and process space.





A little Virtualization History

- The first *hypervisor* providing *full virtualization* was IBM's CP-40 system.
- Deployed in 1967 as the first version of IBM's CP/CMS operating system.
- CP/CMS formed part of IBM's attempt to build robust time-sharing systems for its mainframe computers.
- The hypervisor increased system robustness and stability: If one OS crashed, the others would continue working.
- CP/CMS VM time-sharing OS lost out to MVS batch processing OS.

Disco

- Resurgence of virtualization began in earnest in the mid-to-late 1990's.
- Led by a group of researchers at Stanford headed by Professor Mendel Rosenblum.
- Work on Disco, a VMM for the MIPS processor revived interest in VMMs.
- Led that group to the founding of *VMware* – market leader in virtualization technology.
- Key enabler for running Windows software on Unix servers.

UNIX and Linux Servers

Resurgence in using virtualization technology occurred with *UNIX* and *Linux* server vendors:

- Expanding hardware capabilities, allowing each single machine to do more simultaneous work.
- Efforts to control costs and to simplify management through consolidation of servers.
- Need to control large multiprocessor and cluster installations, e.g., in *server farms* and *render farms*.
- Improved security, reliability, and device independence possible from hypervisor architectures.
- The ability to run complex, OS-dependent applications in different hardware or OS environments.

Virtualization Platforms

- Major *UNIX* vendors, including Sun Microsystems (now part of Oracle), HP, IBM, and SGI (Cray Research), have been selling virtualized hardware since before 2000.
- Similar trends have occurred with x86/x86_64 server platforms, where *open-source* projects such as *Xen* have led virtualization efforts.

Virtualization Platforms

- Interest in the high-profit server-hardware market sector has led to the development of hypervisors for machines using the Intel x86 instruction set, including for traditional desktop PCs.
- *VMware*, debuted in 1998.
- *Parallels, Inc*. introduced *Parallels Workstation* in 2005, which was originally used on PCs, and *Parallels Desktop for Mac*, in 2006, which runs on Mac OS X (10.4 for Intel or higher).

Approaches to Virtualizing Guest OS's

- Hypervisor
- Paravirtualization

Hypervisor

Hypervisor

- Software structure used to *fully emulate computer hardware in software*.
- Software layer creates *virtual CPUs* and *virtual memory* by intercepting the I/O from guest OSes before it reaches the physical hardware.
- *Virtual Hard Disk Drives* are created by mapping disk read and writes to a single file or LUN (logical unit number as in disk storage).
- *Read* and *writes to the networking software* stack are intercepted and redirected to virtual network adapters.

- *Key factor is that guest OS does not know that it runs an abstracted environment.*
  - *System administrator should not perceive any significant difference between an OS running on a hypervisor or on a dedicated physical server.*

Paravirtualization

Paravirtualization

- Similar to hypervisor principle.
- A software hypervisor is installed on a physical server and a guest OS is installed into the environment.
- Presents a software interface to virtual machines that is similar to, but not identical to that of the underlying hardware – **means guest OS's must be modified!**

- *The difference is that the guest OS needs to "know" that it is virtualized to take advantage of the functions.*
  - *Operating systems require extensions to make API calls to the hypervisor.*

Virtualization Notes

- *Modern Virtualization engines are hardware-assisted* – relying on hardware support (HVM) of the CPU (in the form of Instruction Sets).
- Intel and AMD processors now have Instruction Sets called VT and AMD-V respectively which are required by Hyper-V, VMware, and Xen which allows OS's to run unmodified.
- Intercept supervisor calls.

Embedded Systems

- In 2009 virtual machines started to appear in embedded systems, such as mobile phones.
- Provides a high-level operating-system interface for application programming, such as *Linux* or *Microsoft Windows*, while at the same time maintaining traditional *real-time operating system* (RTOS) APIs.
- Obtain priority preemptive scheduling.

Virtualizing the CPU

- Basic technique used is *limited direct execution*.
  - Run as much as possible directly on CPU, but with "limitations."
- Same basic technique that an OS uses to virtualize the CPU.
- To boot a new OS on top of the VMM, set PC (program counter) to the address of the first instruction, and let the OS begin running.
- Similar to an OS switching between running processes (*context switch*).
- VMM acts like OS and schedule different virtual machines to run.

Privileged Instructions

Unlike OS switching between running process, VMM must have way of dealing with hardware *privileged instructions*:

- Privileged instruction: processor opcode (assembler instruction) which can only be executed in "supervisor" (or Ring-0) mode.
- These types of instructions tend to be used to access I/O devices, deal with timers and scheduling, and protect data structures used by the kernel.
- Regular programs execute in "user mode" (Ring-3) which disallows direct access to I/O devices, etc.

Virtualizing the CPU

- UNIX-based systems *open(char *path, int flags, mode t mode)*.
- First, the arguments get pushed onto the stack (mode, flags, path), then a 5 gets pushed onto the stack (OS convention).
- Then int 80h is called, which transfers control to the kernel.

```
open:
    push    dword mode
    push    dword flags
    push    dword path
    mov     eax, 5
    push    eax
    int     80h
```

Virtualizing the CPU

Assume we are running on single processor, and we wish to multiplex between two virtual machines, i.e., two OSes and their respective applications.

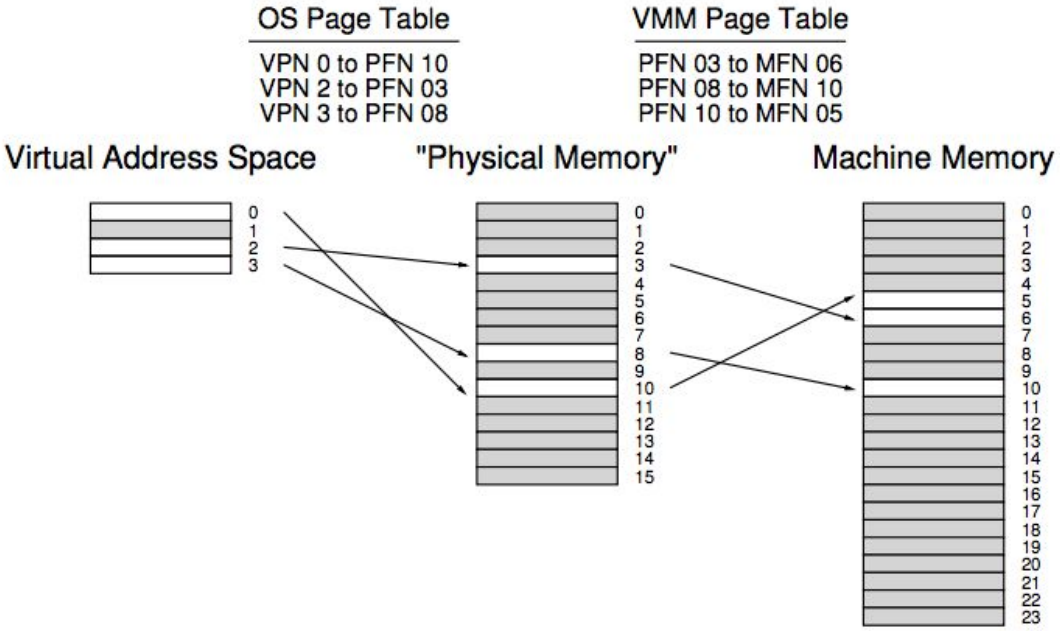
- Similar to an OS switching between running processes (*context switch*).
- VMM must perform a machine switch between running virtual machines.
- When performing a switch:
  - VMM must save the entire machine state of one OS: registers, PC, and unlike in a context switch, any privileged hardware state.
  - Restore the machine state of the to-be-run VM, and then jump to the PC of the to-be-run VM.

Virtualizing Memory

- Each OS normally thinks of physical memory as a linear array of pages, and assigns each page to itself or a user process.
- The OS already virtualizes memory for its running processes, such that each process has the illusion of its own private address space.
- VMM needs to add another layer of virtualization, so that multiple OSes can share the actual physical memory.
- Additional layer of indirection:
  - Each OS maps virtual-to-physical addresses via its per-process page tables
  - The VMM maps the resulting physical mappings to underlying machine addresses via its per-OS page tables.

Virtualizing Memory

- Additional layer of indirection:
  - Each OS maps virtual-to-physical addresses via its per-process page tables
  - The VMM maps the resulting physical mappings to underlying machine addresses via its per-OS page tables.



Xen

Xen is a VMM for IA-32, x86-64, Itanium, and ARM architectures.

- Allows several guest operating systems to execute on the same computer hardware concurrently.
- Intended to scale to ~100+ virtual machines running industry standard applications and services.
- The University of Cambridge Computer Laboratory developed the first versions of Xen.
- The Xen community develops and maintains Xen as free software, licensed under the GNU General Public License (GPLv2).
- Most Linux distributions include Xen packages to interact with the Xen hypervisor and start additional domains.

Xen: Issues with full virtualization

- In traditional VMM (hypervisor) , the virtual HW exposed is functionally identical to the underlying machine.
- Full virtualization has the benefit of hosting unmodified OS's.
- Also some drawbacks – especially with x86 architecture since full virtualization was never part of its design.
  - Certain supervisor instructions when executed *with insufficient privilege* fail rather than causing a trap – which is needed when VMM intervention is required.
- Can be solved with increased complexity and reduced performance.
  - E.g., VMware dynamically rewrites portions of the hosted machine code to insert traps.
- Also helpful to provide both real and virtual time.
  - E.g. handle network timeouts.

Xen: Approach

- Xen avoids drawbacks of full virtualization by presenting a virtual machine abstraction that is similar, but not identical to the underlying HW – *paravirtualization*.
- Provides improved performance, although it requires modification to the guest operating system.
  - Note: no changes to the ABI (application binary interface), i.e., guest applications.

Xen: Design Principles

1. *Support for unmodified application binaries* is essential, or users will not transition to Xen.
   - Virtualize all architectural features. Required by existing standard ABIs.
2. *Support full multi-application OS's*.
   - Allows complex server configurations to be virtualized with single guest OS instance.
3. *Paravirtualization* is necessary to obtain *high performance* and strong resource isolation on uncooperative machine architectures.
   - x86.
4. *Do not completely hide resource virtualization*.
   - Even on cooperative machine architectures, completely hiding the effects of resource virtualization from guest OS's risks both correctness and performance.

Skip to borrowed time

VM Interface – Memory Management

Memory management

- *TLB – Translational Look-aside Buffer*
  - CPU cache that memory management hardware uses to improve virtual address translation speed.
  - Mapping of virtual and physical address spaces
- Easier if software-managed, but much *slower*.
- X86 serviced directly by the processor.

Xen:

- Guest OSes responsible for managing their own hardware page tables.
- Xen exists in 64MB section at the top of every address space to avoid flushing the cache. Not accessible to guest OSes.

Procedure:

- Each time a guest OS requires a new page table, e.g., process is created, guest OS allocates and initializes a page from its own memory and registers it with Xen.
- Relinquish direct write privileges to page-table memory.
- All subsequent updates must be validated by Xen.

VM Interface – CPU

CPU

- Insertion of hypervisor below the operating system *violates the usual assumption that the OS is the most privileged entity in the system*.
- To protect the hypervisor (and other domains) from OS misbehavior, *guest OS's must be modified to run at a lower privilege level*.
- Most OS's have two privilege levels - X86 has four privilege levels.
- *OS code executes in ring 0 (kernel mode)* – no other ring can execute privileges instructions. 2 & 3 not typically used. 4 is for applications.
- Xen runs guest OS's in ring 2. Prevents access to privileged instructions (i.e., IO, state, protected memory), protects them from applications running in ring 4.

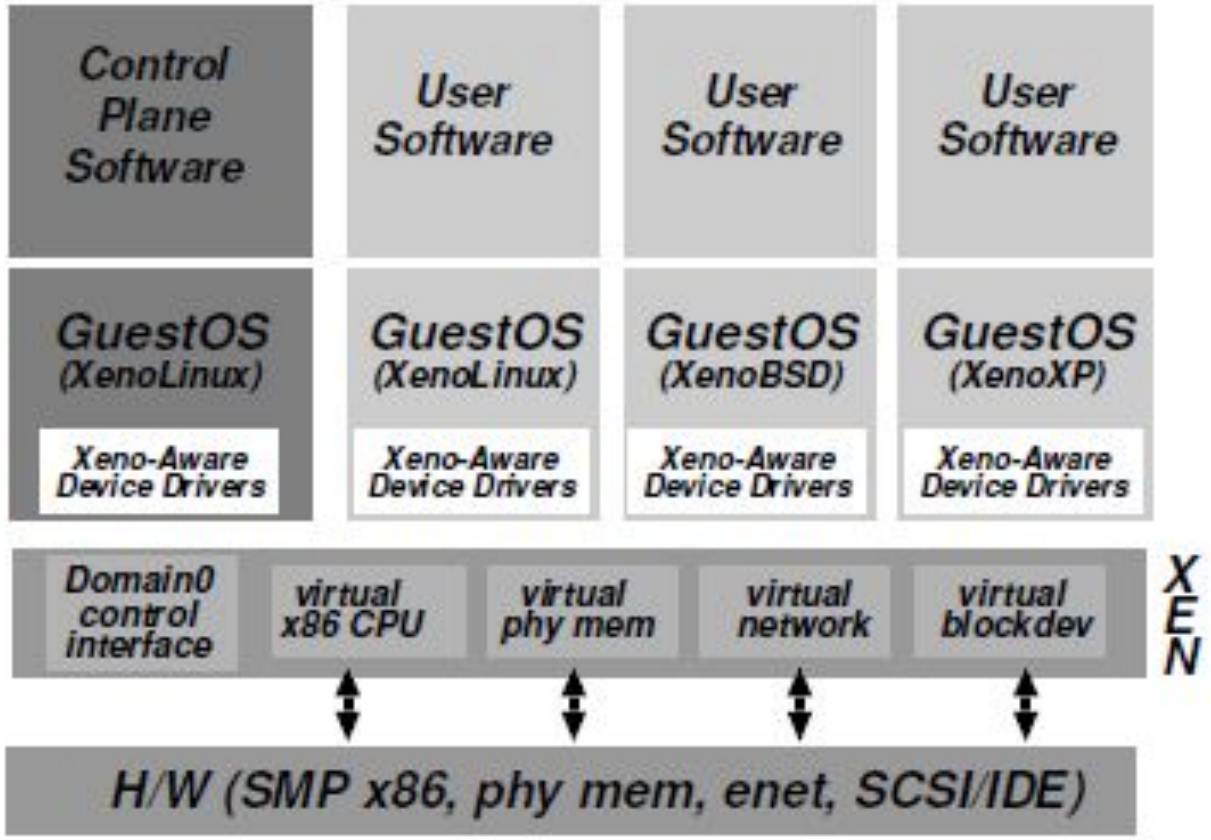
VM Interface – Device IO

Device I/O

- Xen exposes a clean and simple device abstraction.
- Data is transferred to/from each domain via Xen, using shared memory with asynchronous buffering.
- Also supports light-weight event delivery, similar to interrupts.

Control & Management

- Structure of a machine running Xen hypervisor
- Emphasis is on hypervisor providing only basic control operations.



Control & Management

- Domain created at boot time.
- Initial domain, Domain0, hosts the application-level management software.
- Control interface provides ability to create and terminate other domains and to control their associated scheduling parameters, physical memory allocations and the access to disks and network devices

Borrowed-Virtual-Time (BVT)

- Systems need to run a larger and more diverse set of applications, from *real-time* to *interactive* to *batch*.
- Uniprocessor and multiprocessor platforms.
- Most schedulers limit their applicability to general-purpose systems. Either:
  - do not address latency requirements (priority-weighted time sharing), or
  - are specialized to complex real-time paradigms (priority preemptive).

BVT Scheduling

- Provides low-latency for realtime *and* interactive applications.
- Weighted sharing of the CPU across applications according to a system policy.
- Makes minimal demands on application developers.
- Can be used with a control module for hard real-time app's.

BVT

- Each thread's execution time is monitored in terms of *virtual time, dispatching the runnable thread with* **the earliest *effective virtual time (EVT).***
- However, ***a latency sensitive*** thread is allowed to *warp back in virtual time to* make it appear earlier and thereby ***gain dispatch preference.***
- It then effectively ***borrows virtual time from its future CPU*** allocation and thus does not disrupt long-term CPU sharing.
- Hence the name, *borrowed virtual time scheduling.*

Summary

- Virtualization is fundamental for achieving cloud computing.
- Users and admins need to run multiple OSes on the same machine at the same time.
- Key is that VMMs generally provide this service transparently: the OS has little clue that it is not actually controlling the HW of the machine.
- The key method that VMMs use to do so is to extend the notion of limited direct execution by setting up the HW to enable the VMM to interpose on key events (such as traps).
- The VMM can completely control how machine resources are allocated while preserving the illusion that the OS requires.