Service Oriented Computing (4)

Chan-Hyun YOUN

Dept of Electrical Engineering, KAIST

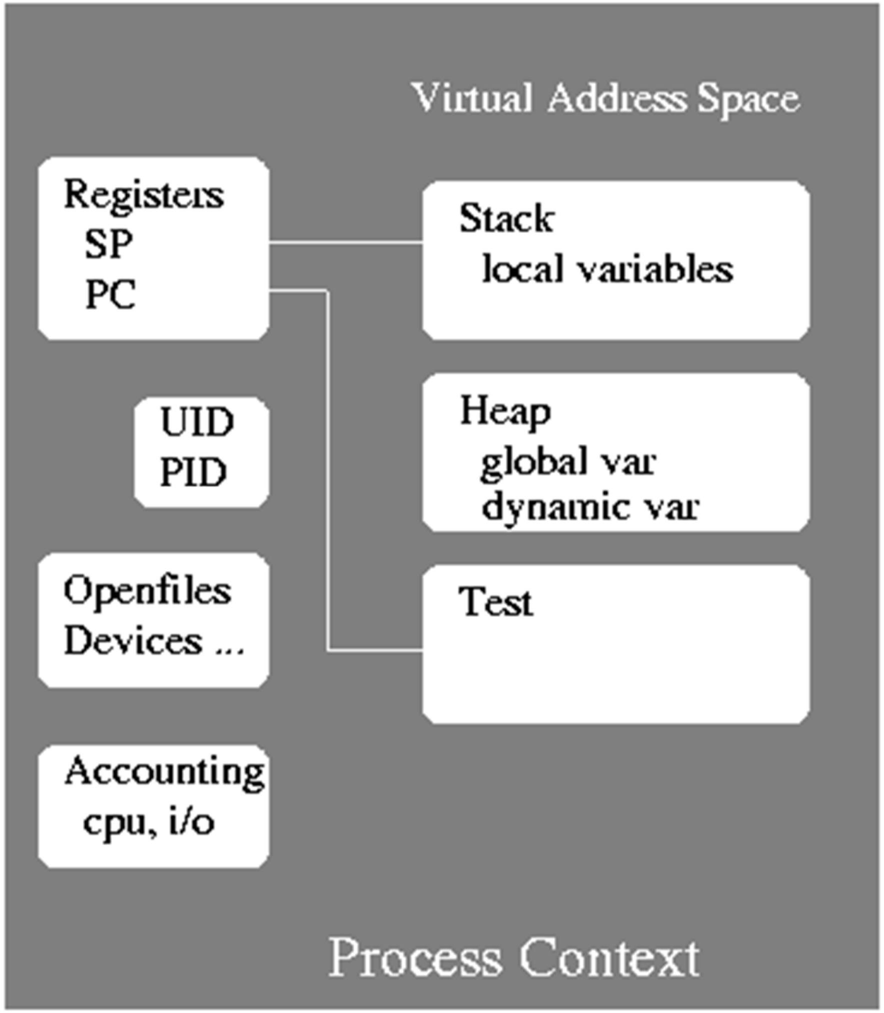
Process and Thread

Outline

- Process and Thread
  - Abstraction of execution entity
- Software vs HW Implementation
  - User-level and Kernel-level
  - Hyper-threading and multi-core

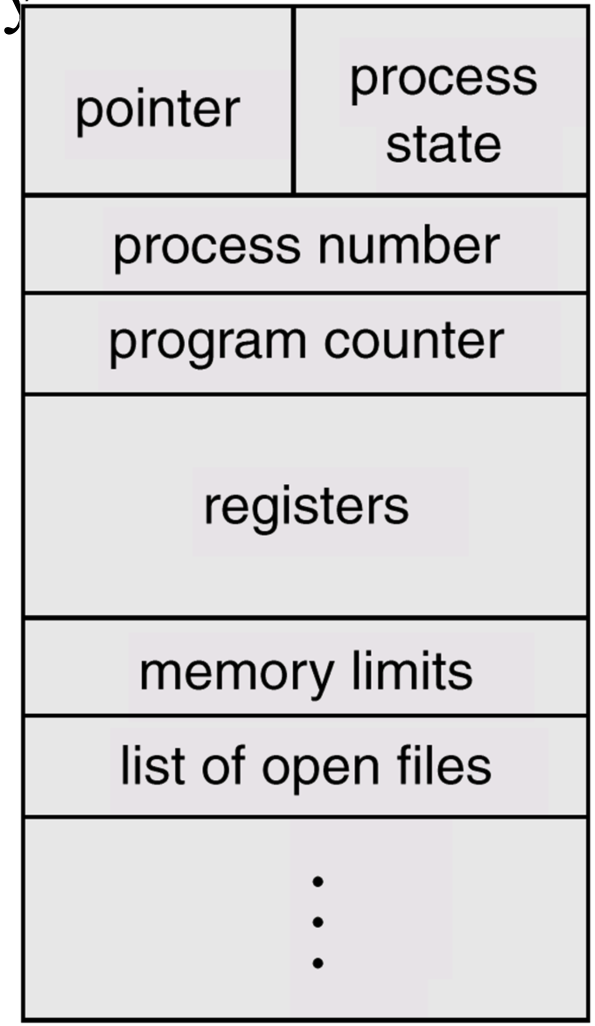
Process Model

- Process: a program in execution
  - IE to web browsing, outlook to read email, word to write reports
- Process is a full blown virtual machine, defining its own address space and maintains running state
  - Address space: the set of memory locations that can be generated and accessed directly by a program; enforced by hw for protection
  - I/O part of the machine is accessed through OS calls.



Under the Hood

- Process Control Block (PCB): a key data structure that maintains info associated with each process:
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management info
  - Accounting information
  - I/O status information



pointer

process state

process number

program counter

registers

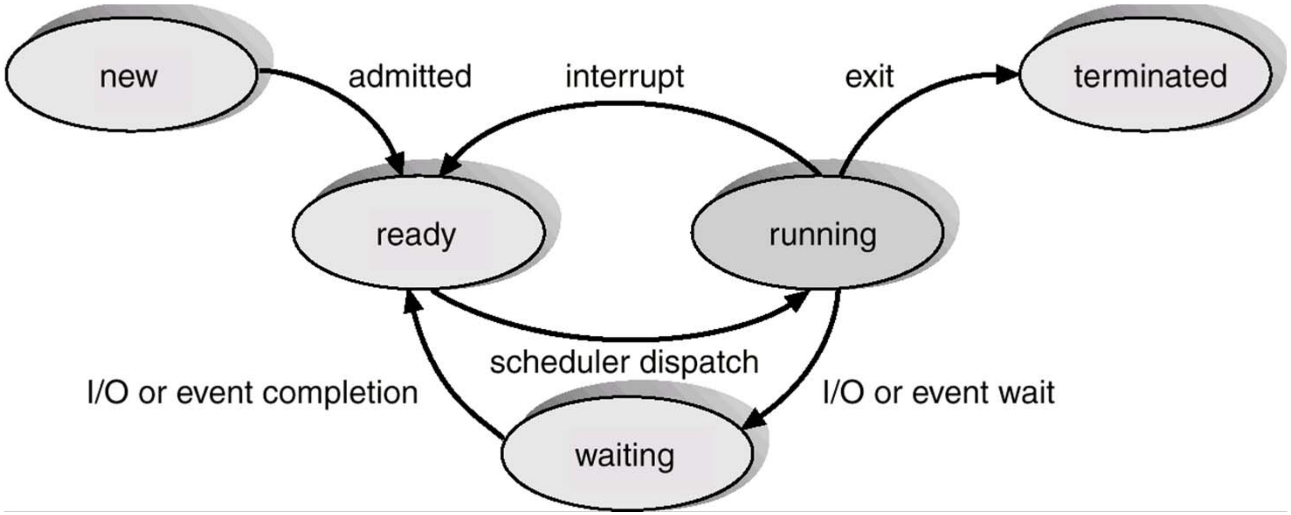
memory limits

list of open files

•
•
•

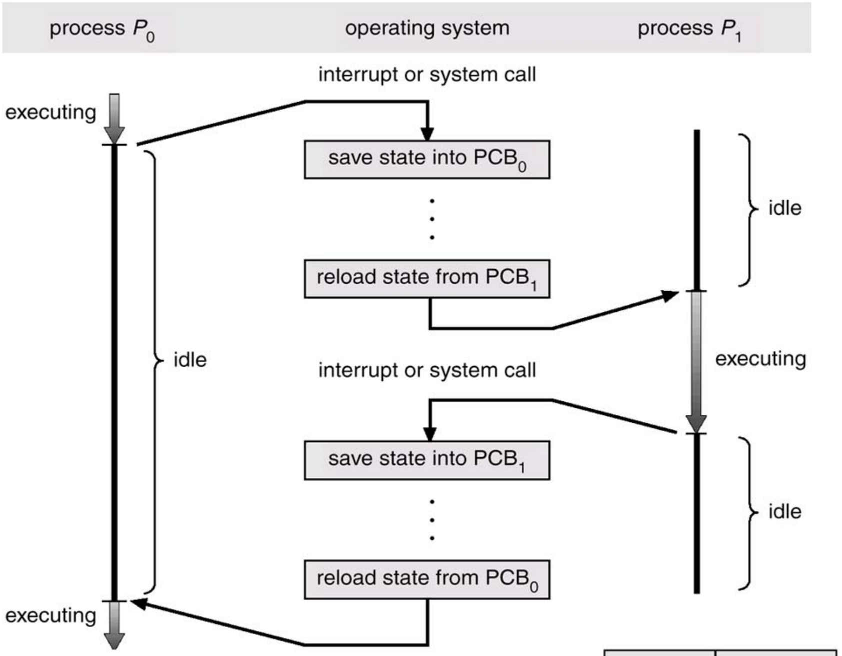
Process State

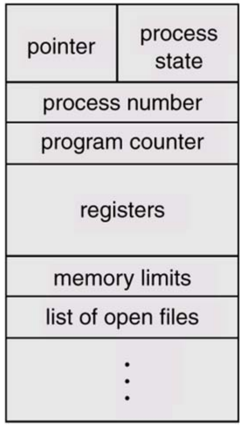
- As a process executes, it changes *state*
  - new: The process is being created.
  - running: Instructions are being executed.
  - waiting: The process is waiting for some event to occur.
  - ready: The process is waiting to be assigned to a process.
  - terminated: The process has finished execution.



Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

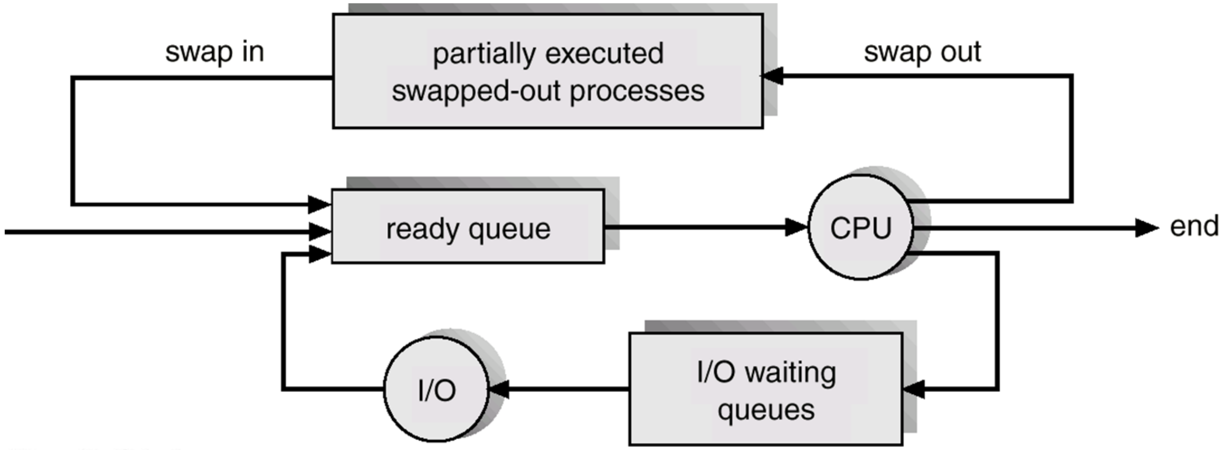




PCB

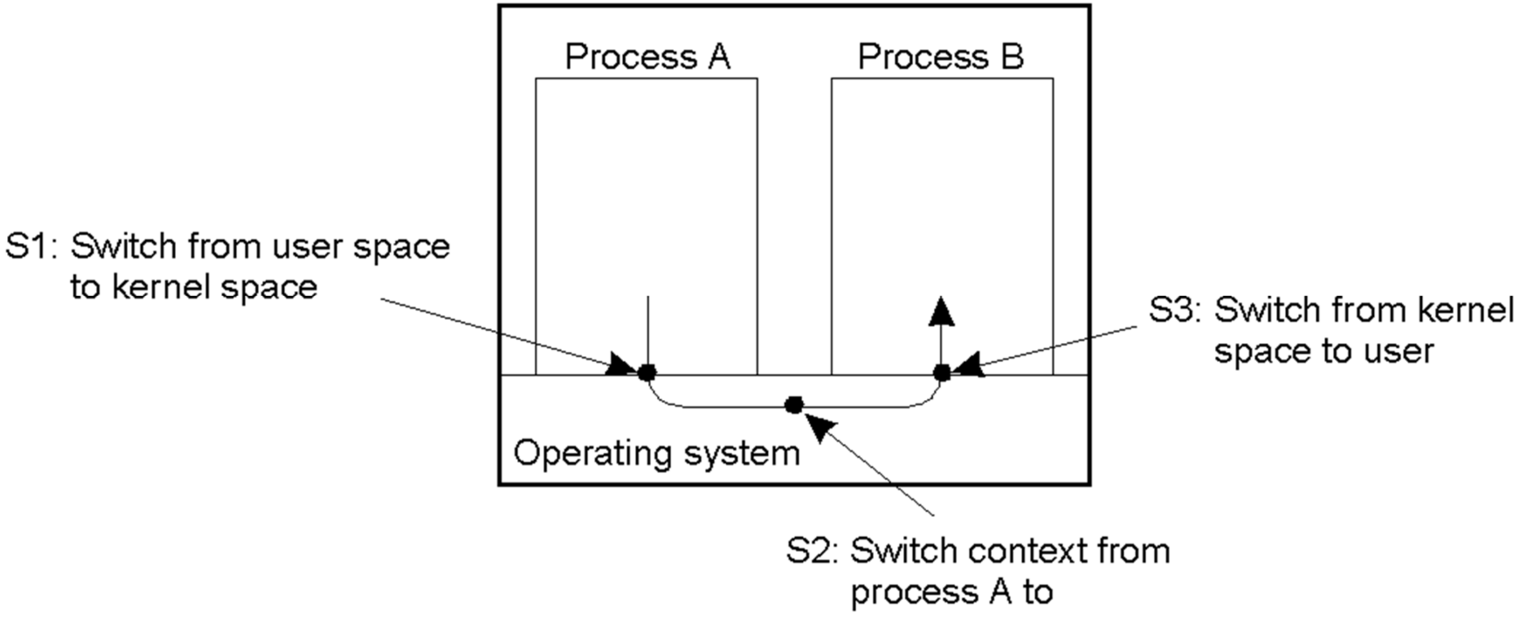
Process Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
  - Invoke infrequently (seconds, minutes)
  - Control the degree of concurrency
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.
  - Invoke frequently (millisecond) enough to provide a perception of concurrent execution on single core system; must be fast
  - Optimize throughput, average response time, slowdown, revenue, etc
  - Slowdown is a normalized queuing delay w.r.t. exec time
  - On multiprocessor or multi-core system, real concurrent execution



Interprocess Communication (IPC)

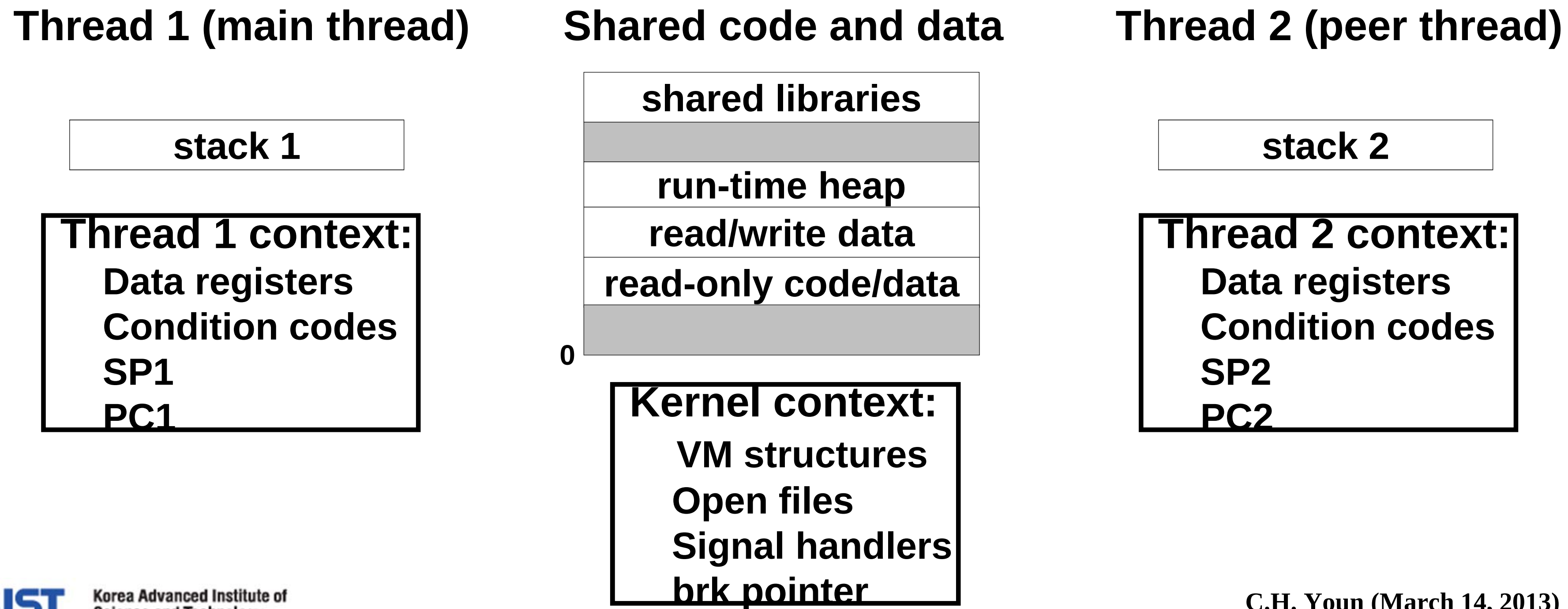
- Processes are usually communicate by sending msgs back and forth via the OS


Process A
Process B
S1: Switch from user space to kernel space
S3: Switch from kernel space to user
Operating system
S2: Switch context from process A to

- Processes can share a segment of physical memory by mapping it to their address spaces
  - data in the segment can be accessed by processes for faster IPC

Thread and Multithreaded Process

- A thread of control is a seq of instructions being executed within a process context. Each has its own logical control flow (pc)
- A multithreaded process has two or more threads within the same context. They share code, data in heap, and kernel context(open files, timers, etc)
- Each thread has its own id.



Thread 1 (main thread)

stack 1

Thread 1 context:
Data registers
Condition codes
SP1
PC1

Shared code and data

shared libraries

run-time heap

read/write data

read-only code/data

0

Kernel context:
VM structures
Open files
Signal handlers
brk pointer

Thread 2 (peer thread)

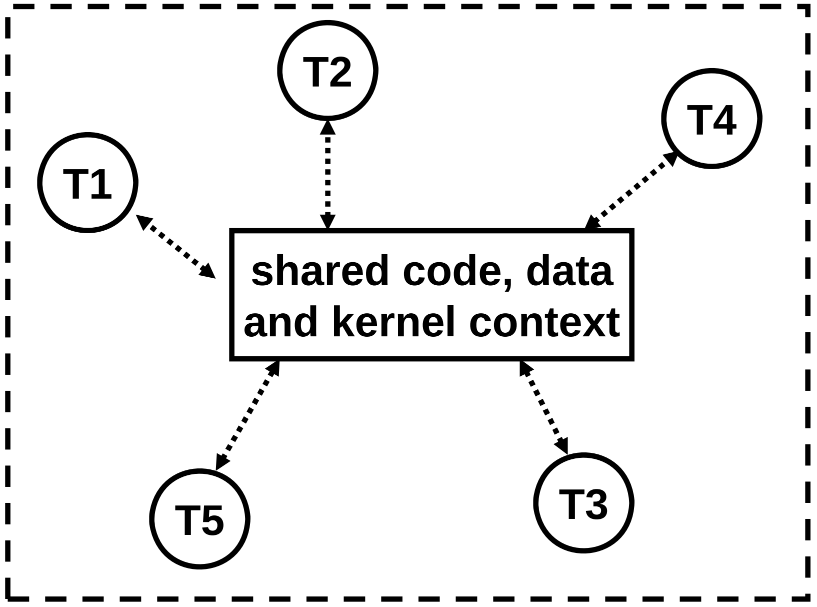
stack 2

Thread 2 context:
Data registers
Condition codes
SP2
PC2

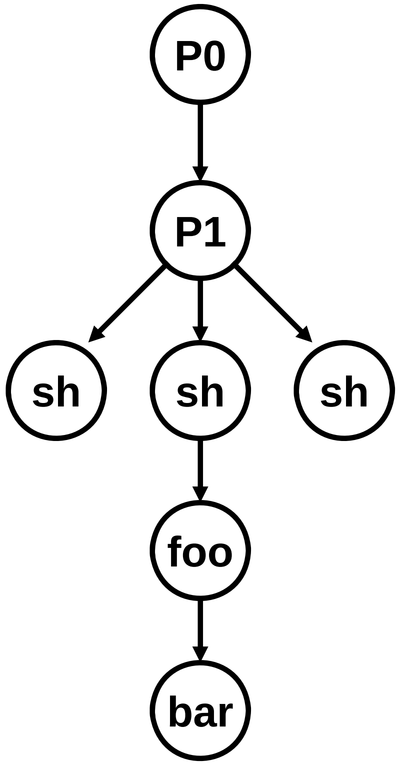
Logical View of Threads

- Threads associated with a process form a pool of peers.
  - unlike processes which form a tree hierarchy

Threads associated with process foo



Process hierarchy



Process vs Threads

- Processes are typically independent, while threads exist as subsets of a process
- Processes carry considerable state info, whereas multiple threads within a process share state as well as memory and other resources
- Processes have separate address space, whereas threads share their address space
- Processes interact through system-provided IPC
- Context switching between threads is an order of magnitude faster than context switching between processes

Why Multithreading

- Improve program structure
  - e.g. producer-consumer problem
- Efficiency
  - Creating a process calls into OS, which duplicates entire address space
  - Synchronizing processes need to trap into the OS, too.
  - Threads could be created in user-space
  - Threads could be synchronized by monitoring shared variables
- Match multi-core and multiprocessor arch
- Improve application responsiveness
  - overlapping I/O
    - e.g. Multithreaded clients, fast file read/write
  - asynchronous event handling
    - e.g. network-centric server , GUI

Outline

- Process and Thread
  - Abstraction of execution entity
- Software vs HW Implementation
  - User-level and Kernel-level
  - Hyper-threading and multi-core
- Virtual Machines
  - Another layer of abstraction of exec entity
- Scheduling Policies/Disciplines

Design Issues for Multithreading

- Thread management:
  - thread creation and termination
- Thread synchronization:
  - mutex, condition variable
  - semaphore, reader/writer
  - monitor
- Thread scheduling
  - in a way similar to process scheduling

Implementation

- User space implementation (green threads)
  - Transparent to OS kernel
  - Runtime system manages threads in userspace
  - E.g. Java Runtime Enviroment
- Kernel space implementation
  - Thread becomes a basic unit of OS's resource management
  - e.g. Window, Solaris, Linux
- Hardware implementation: Simultaneous Multithreading (SMT)
  - Duplicate certain CPU sections (arch. states) to make a processor appear as multiple "logical" processors
  - E.g. Intel's Hyper-Threading Technology (HTT)

User Space Implementation

- Runtime system is a collection of procedures that manage threads
  – create/terminate threads, synchronization, schedule
  – RTS maintains a table per process with each entry per thread
    - thread's registers, state, priority, etc.
  – Switch threads when a thread be suspended
    - switch register values
    - switch stack pointer and program counter

Kernel Space Implementation

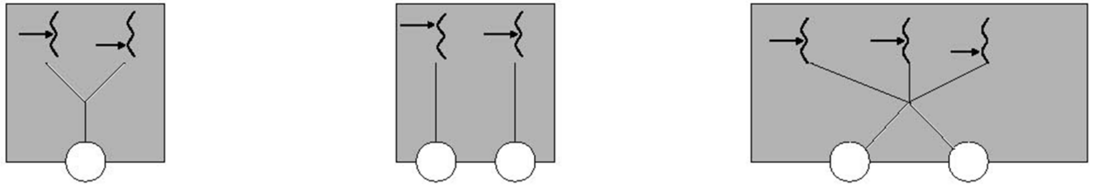
- Threads are managed by kernel
  - creation/termination are through system calls.
  - Thread table is maintained in the kernel space
- Threads are scheduled as processes
  - when a thread is blocked, the kernel run another thread from the same proc or a different proc
- but, relatively heavier
  - Solution: thread recycling (create more kernel level threads than available processors); But
  - Idle kernel threads, priority problems, deadlock introduced by blocked kernel threads
  - Kernel doesn’t know which threads to take away

Pros and Cons of User Threads

- Lightweight: an order of magnitude faster than kernel trapping (procedure call ~10ns vs system call ~5us)
- flexible in scheduling threads of a process in userspace
- Cons:
  - how to implement blocking system calls
    - A blocking call may stop all threads of a process
    - Wrap a blocking call with a jacket: e.g. SELECT, MPI_Test()
    - No true parallelism without support from kernel thread
  - Information barrier between library and kernel
    - A thread holding lock could be scheduled out by kernel
    - A thread of high priority could be scheduled out by kernel
- Hard for time-slicing scheduling
  - no clock interrupts to threads
  - Java thread scheduling is left up to implementation
    - round-robin in Solaris vs time-slicing in Windows
    - robust code: yield or sleep()

Hybrid User/kernel-Space Impl.

- M-1 model (user-level): all app-level threads map to a single kernel-level scheduled entity. portable, easy to programming, but no concurrency
- 1:1 model (kernel-level): user-level threads are in 1-1 correspondence with schedulable entities in the kernel. Each user level thread is known to the kernel and all threads can access the kernel at the same time but, hard to prog



M:1

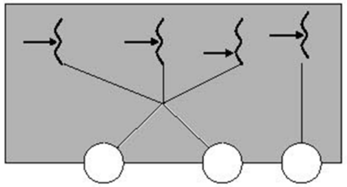
HP-UX 10.20
(via DCE)
Green Threads

1:1

Win32, OS/2, AIX 4.0

M:M (strict)

Linux, Solaris 5.9 and later, NetBSD5, FreeBSD 8

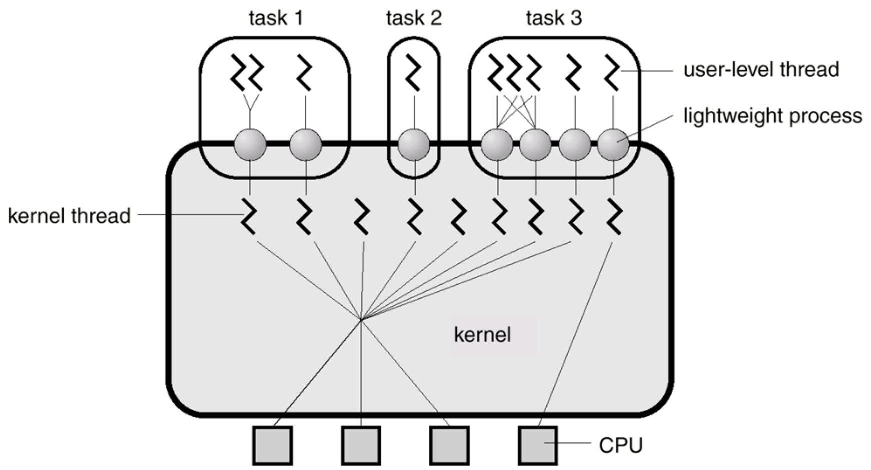


2-Level (aka: M:M)
Solaris, DEC, IRIX, HP-UX 10.30, AIX 4.1

Solaris 5.2 to 5.9, NetBSD2 to 4, FreeBSD 5 and 6

N-to-M Model:

- M:M model (2-level model) to minimizesprogramming effort while reducing the cost and weight of each thread.
- Threads, Lightweight Process, Processor
  - user threads over lightweight processes(lwp)
  - LWP, supported by kernel-thread, over CPU
- A program can create any number of threads. It relies on user-level threads library for scheduling. Kernel only needs to manage currently active threads.
- But, complexity, priority inversion, suboptimal scheduling

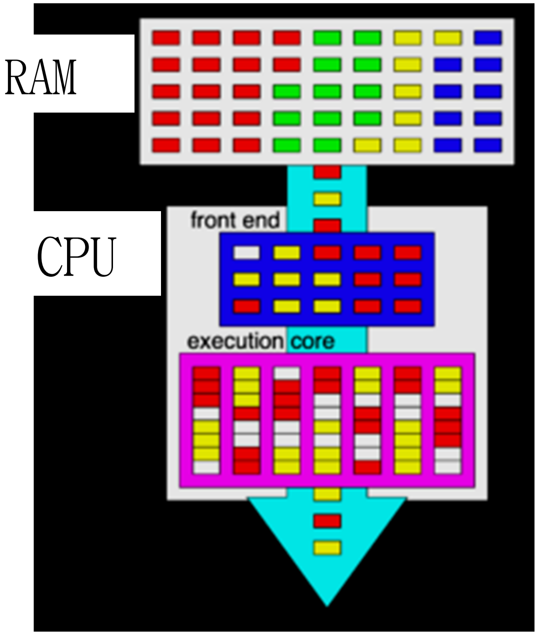


M-to-M Thread Model

- Solaris 5.2 to 5.8, NetBSD 2 to Net BSD 4, FreeBSD 5 to 7
- Create user-level threads via library *libthread*
- User can specify how many LWPs should run these user-level threads
  - *thr_setconcurrency*(new_level) and *thr_getconcurrency*()
- User can bind threads to LWPs, or leave it to the library to schedule
  - unbound vs bound
- User level thread library *libthread* schedules the threads on the LWPs
  - preemptive scheduling: thr_setprio() and thr_getprio()
- Kernel schedules the LWPs on the available CPUs
  - each LWP has a unique interval timer and alarm

Simultaneous Multithreading

- SMT (Hyper-threading in Pentium 4): duplicate certain sections of the processor, mainly those for storing the architectural states, but not duplicating the main execution resources
  - Make it appear to OS as multiple "logical" processors
  - When the execution part is not used due to memory stall, another task can be scheduled for execution
- Transparent to OS and programs. But SMP support needs to be enabled to take advantage of HTT
- Cons: not energy efficient



Multicore Architecture

- Combine 2 or more independent cores (normally CPU) into a single package
- Support multitasking and multithreading in a single physical package

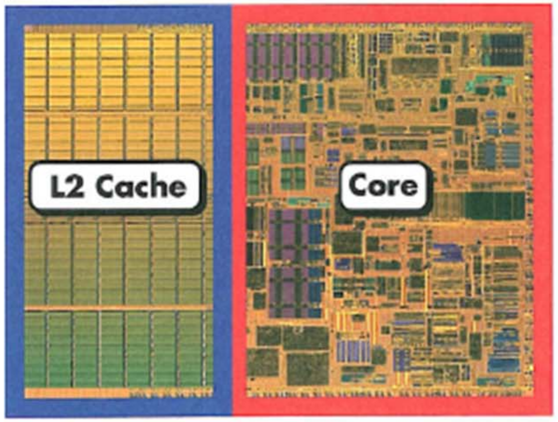


Figure 12-6: The floor plan of an Intel Pentium M processor

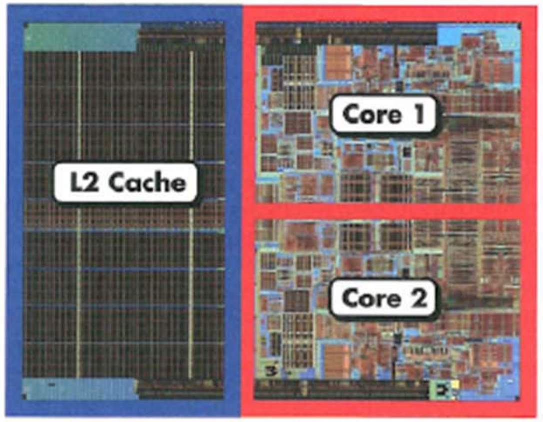
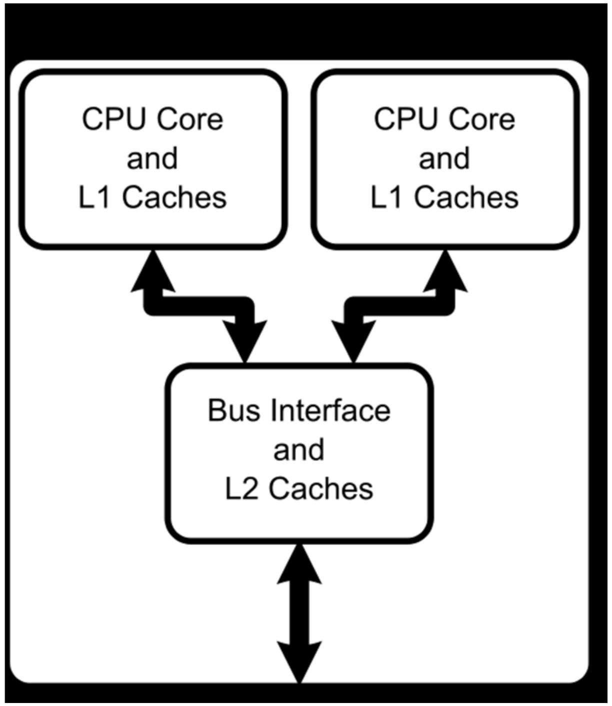
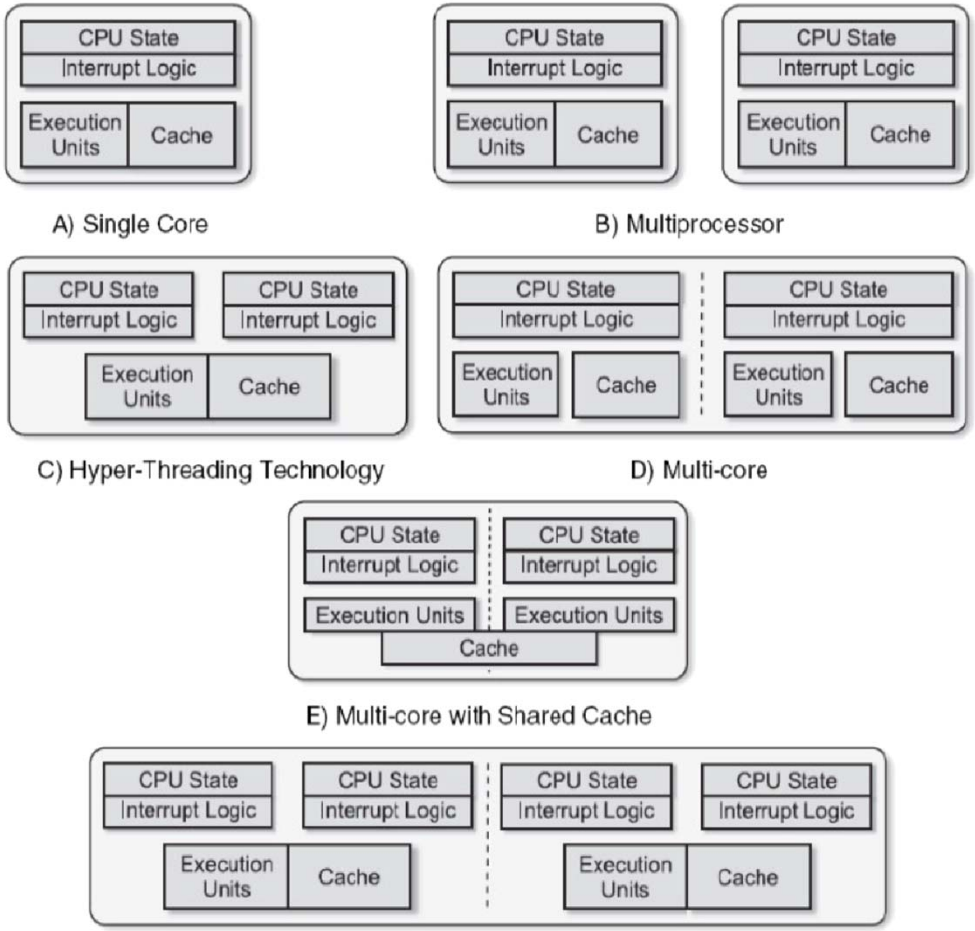


Figure 12-9: The floor plan of an Intel Core Duo processor



Hyperthreading vs Multicore



A) Single Core

B) Multiprocessor

C) Hyper-Threading Technology

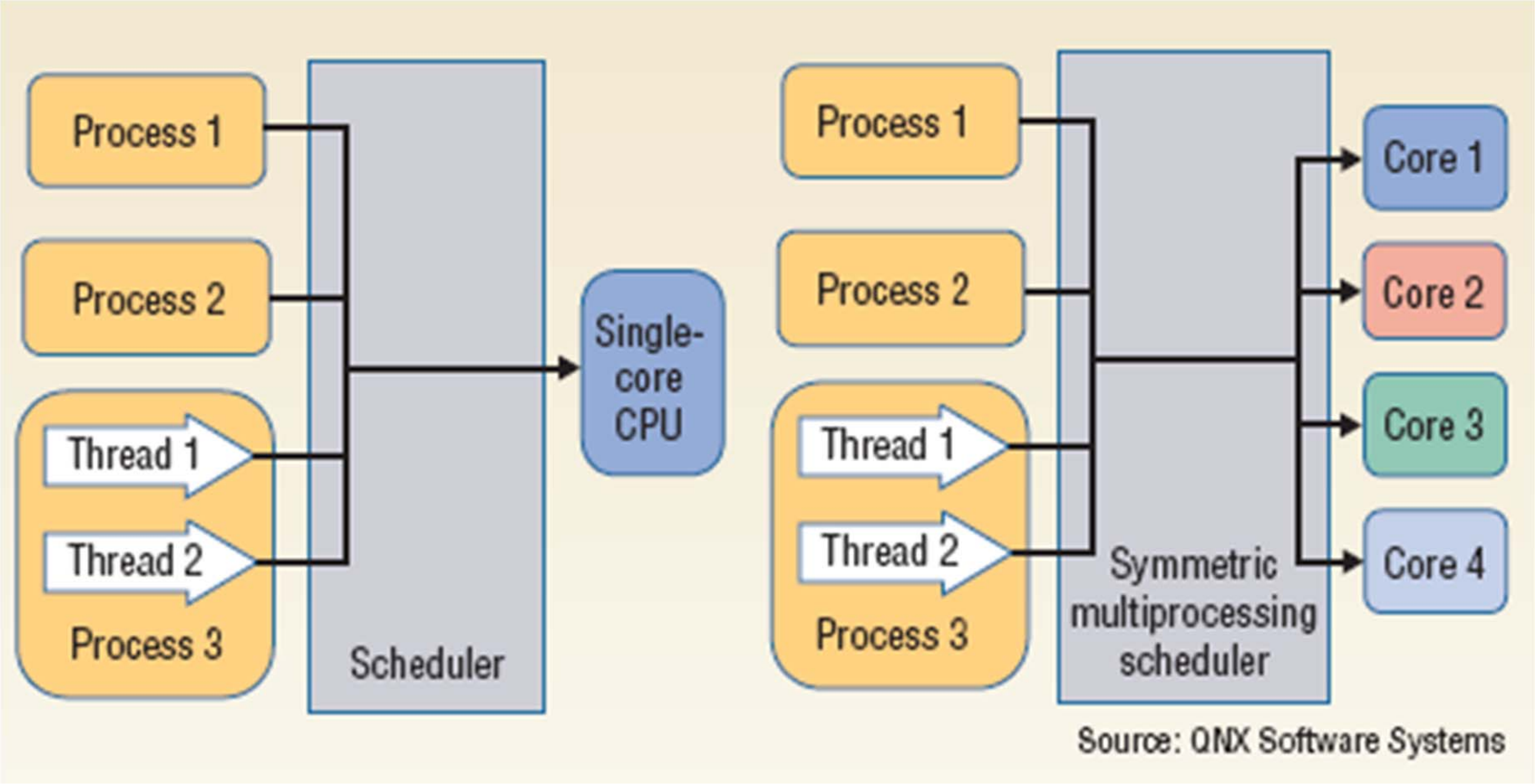
D) Multi-core

E) Multi-core with Shared Cache

F) Multi-core with Hyper-Threading Technology

Figure 1.4 Simple Comparison of Single-core, Multi-processor, and Multi-Core Architectures

Multithreading on multi-core



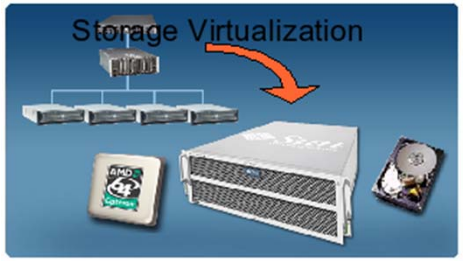
Source: QNX Software Systems

David Geer, IEEE Computer, 2007

Virtualization

♣ Lecture Notes are composed with Prof. S Park’s presentation at Sogang Univ.

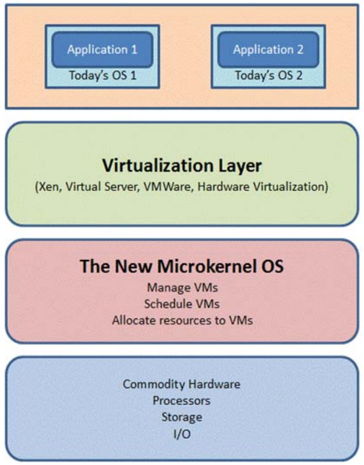
Virtualization



Storage Virtualization

Application Virtualization

Server virtualization



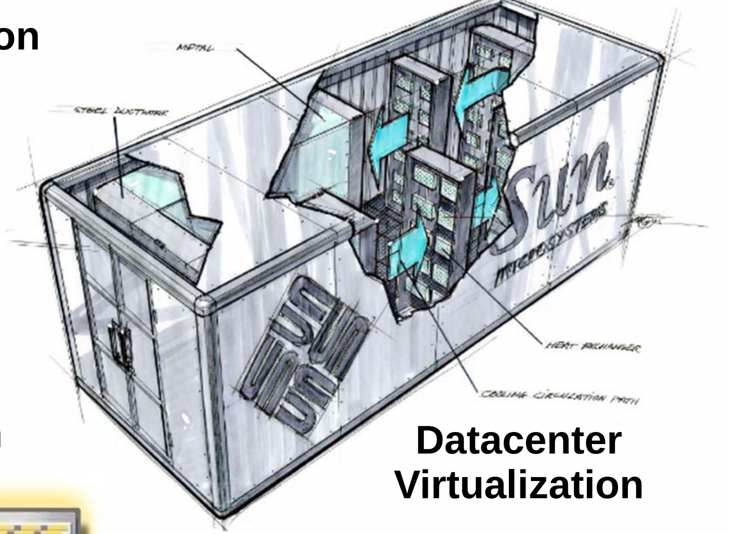
Application 1
Today's OS 1

Application 2
Today's OS 2

Virtualization Layer
(Xen, Virtual Server, VMWare, Hardware Virtualization)

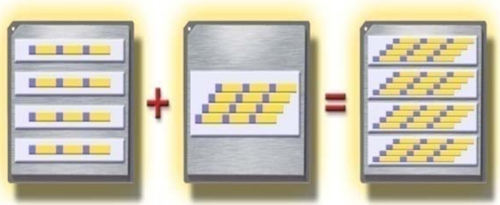
The New Microkernel OS
Manage VMs
Schedule VMs
Allocate resources to VMs

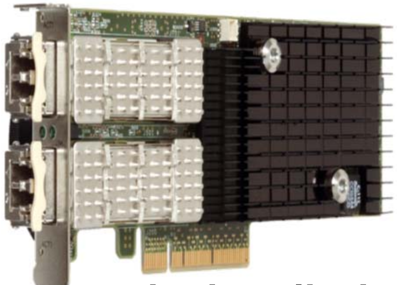
Commodity Hardware
Processors
Storage
I/O



Datacenter Virtualization

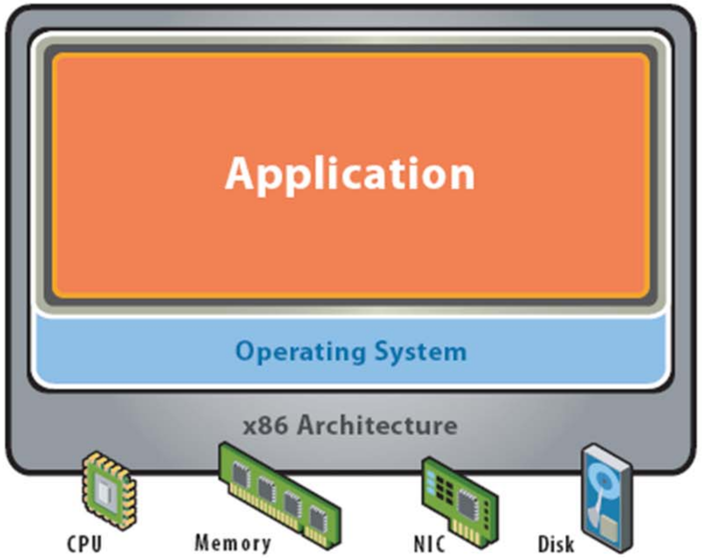
CPU Virtualization

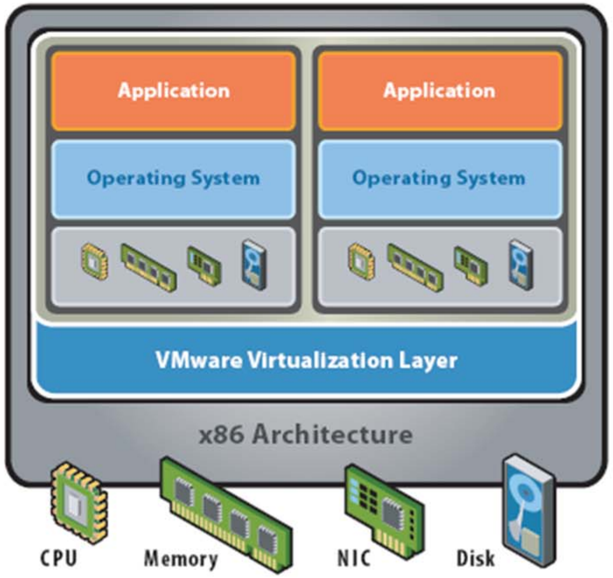




Network Virtualizaiton

Platform Virtualization





Before Virtualization:

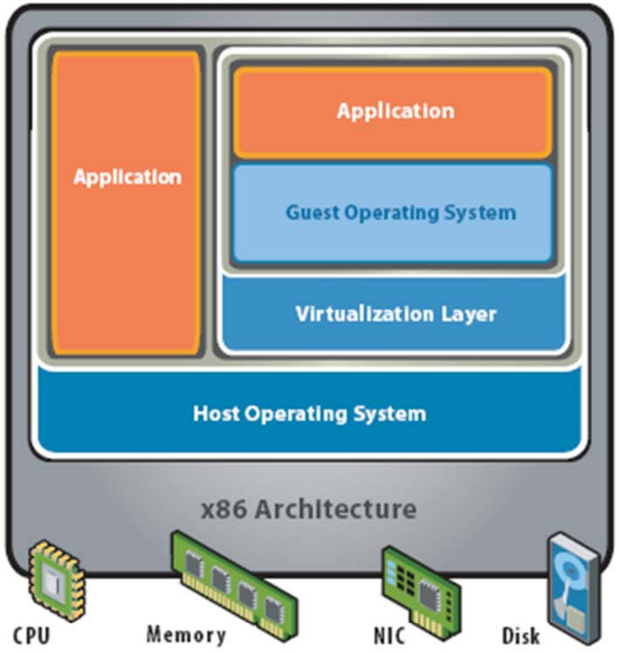
- Single OS image per machine
- Software and hardware tightly coupled
- Running multiple applications on same machine often creates conflict
- Underutilized resources
- Inflexible and costly infrastructure

After Virtualization:

- Hardware-independence of operating system and applications
- Virtual machines can be provisioned to any system
- Can manage OS and application as a single unit by encapsulating them into virtual machines

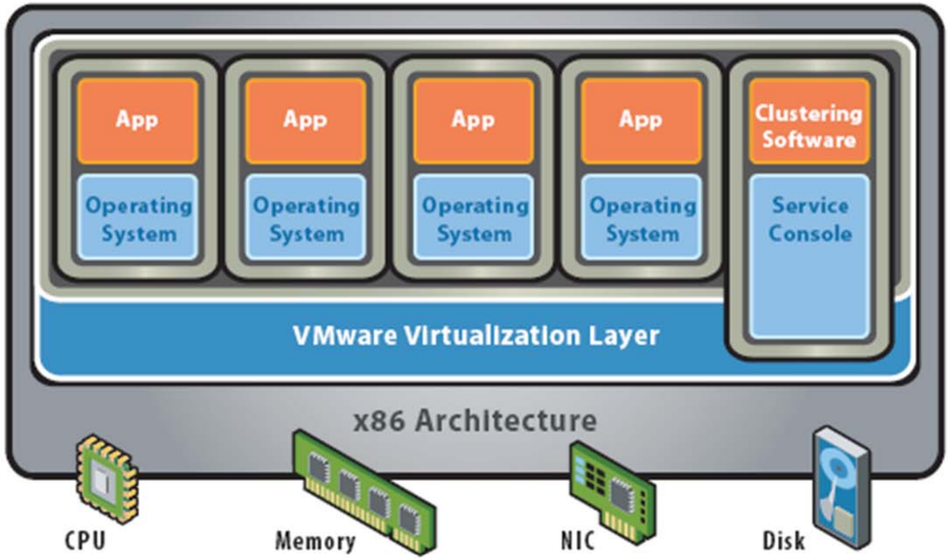
Platform Virtualization

: “Multiple OS” on a Single Machine



Hosted Architecture

- Installs and runs as an application
- Relies on host OS for device support and physical resource management

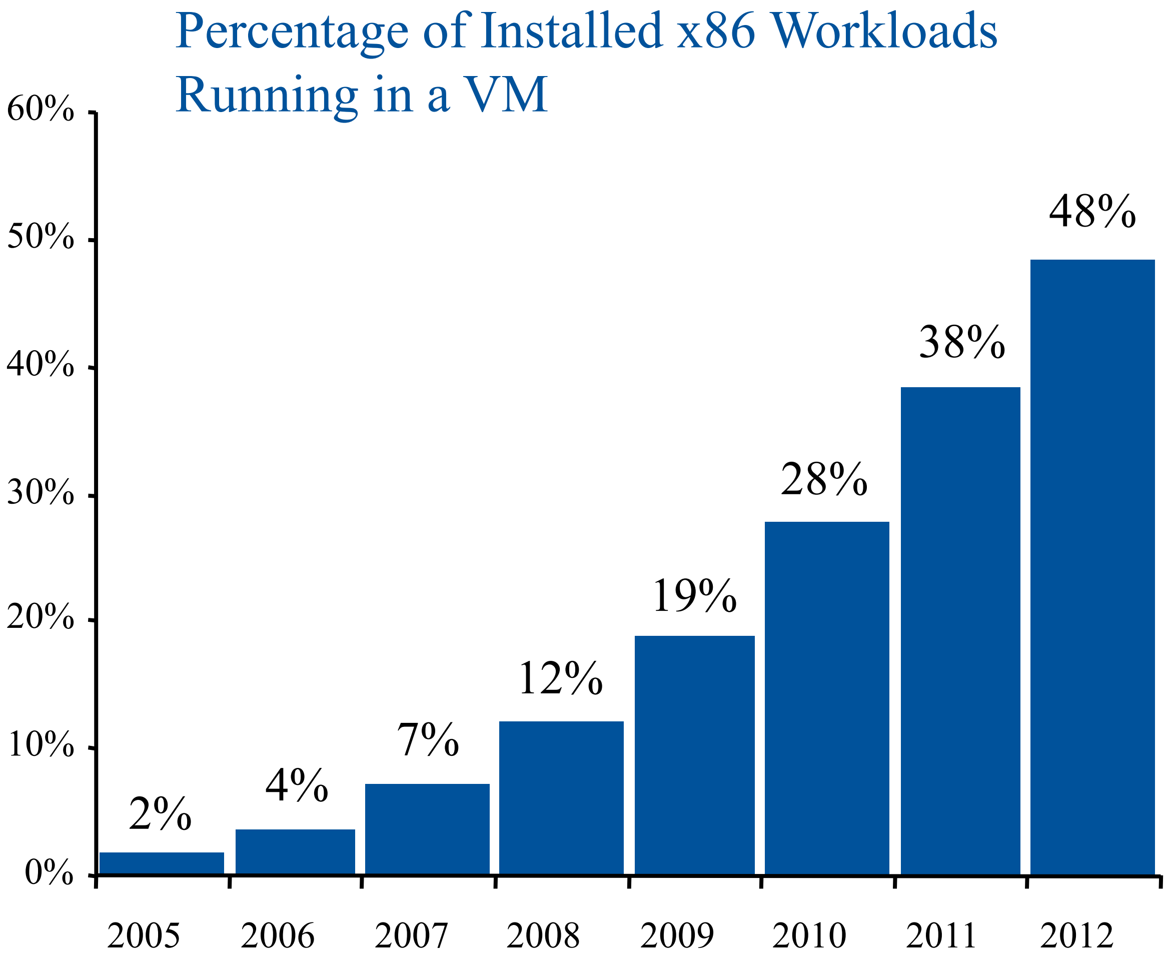


Bare-Metal (Hypervisor) Architecture

- Lean virtualization-centric kernel
- Service Console for agents and helper applications

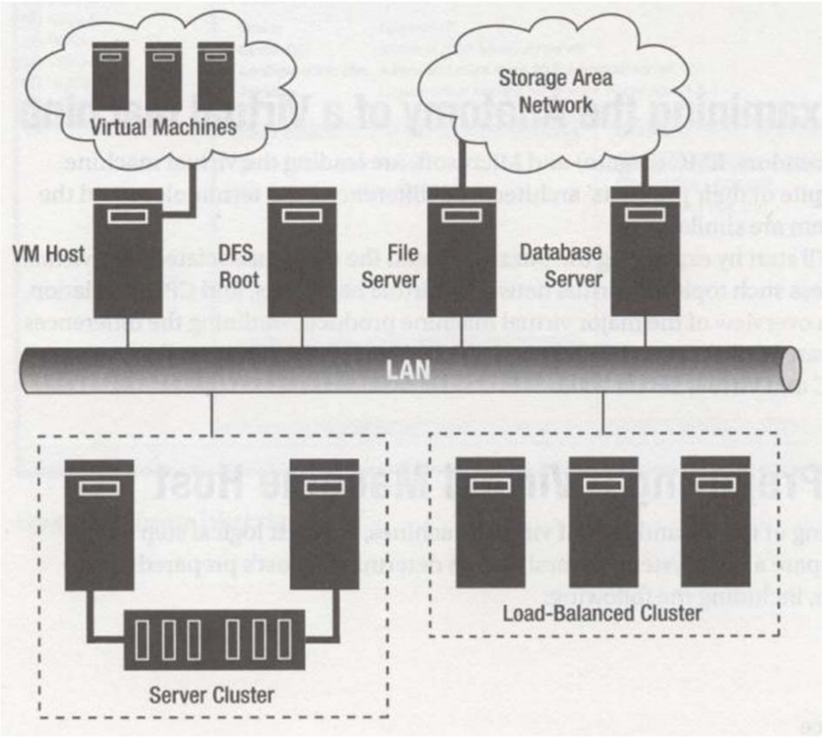
Virtual Machines: Moving From Niche to Mainstream

- Large enterprises started sooner — Global 500 (G500) are perhaps 25% virtualized
- Small or midsize businesses (SMBs) started later, and tend to be less virtualized
- SMBs are virtualizing very fast — will exceed G500 penetration in 2009 or 2010



What is Virtualization?

- "**The abstraction of computer resources**"
  - A technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources.

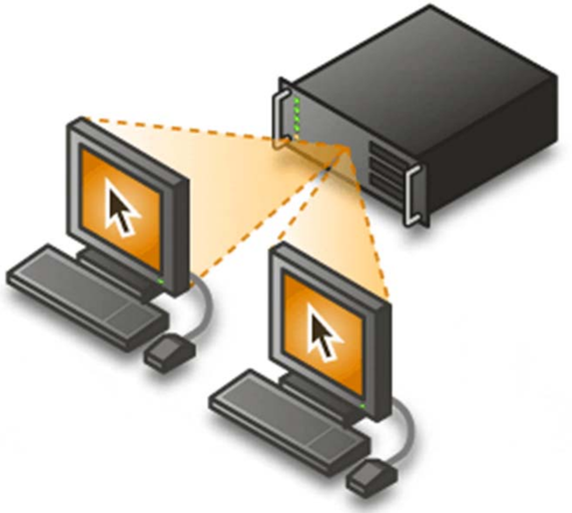


Benefits

- **Partitioning**
  - Multiple applications and OSes can be supported within a single physical system.
  - Servers can be consolidated into VMs on either a scale-up (scale vertically) or scale-out (scale horizontally) architecture.
  - Computing resources are treated as a uniform pool to be allocated to VMs in a controlled manner.
- **Isolation**
  - VMs are completely isolated from the host machine and other VMs. If a VM crashes, all others are unaffected.
  - Data does not leak across VMs and applications can only communicate over configured network connections.
- **Encapsulation**
  - Complete VM environment is saved as a single file; easy to backup, move and copy.
  - Standardized virtualized H/W is presented to the application. – guaranteeing compatibility.

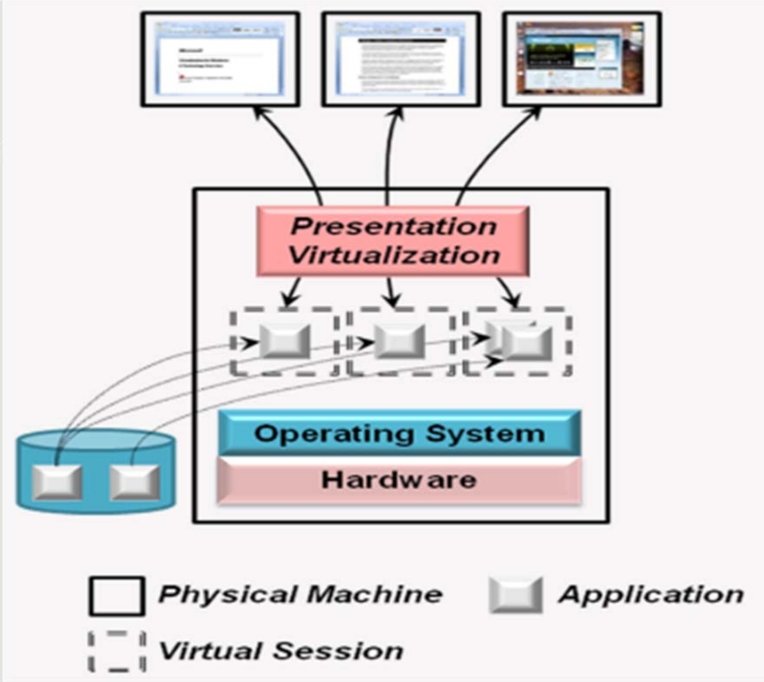
Desktop Virtualization

- The ability to display a graphical desktop from one computer system on another computer system or smart display device
- Presentation Virtualization
  - Virtual sessions
    - Executing project their user interfaces remotely
    - Each session might run only a single application, or it might present its user with a complete desktop offering multiple applications.



Desktop Virtualization

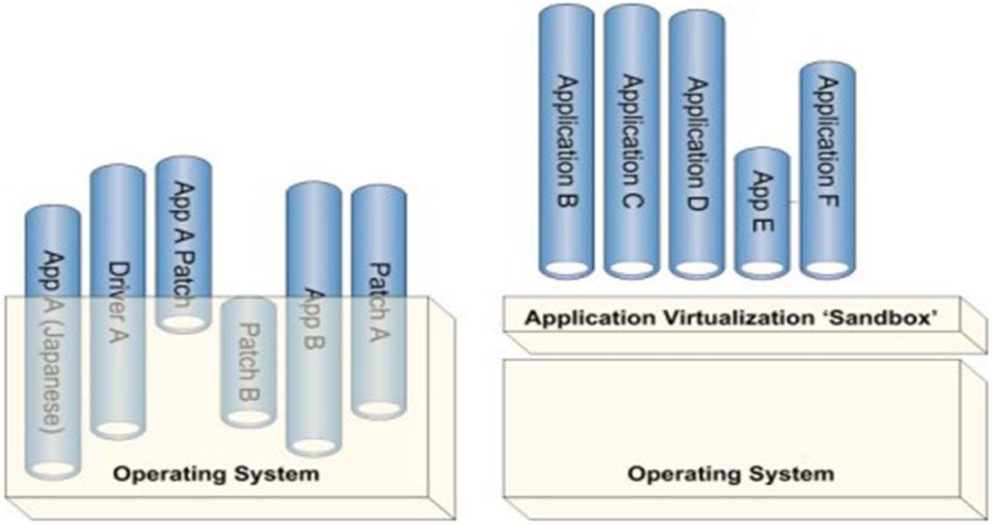
- Advantages
  - Centralized Data
    - Storing safely on a central serv
    - Security improvement
      - Reducing the application managing cost
      - Organizations need no longer worry about incompatibilities between an application and a desktop OS.



- Examples
  - Microsoft's Remote Desktop (Thin Client)
    - http://www.microsoft.com/windowsxp/using/mobility/getstarted/remoteintro.mspx#EIB

Application Virtualization

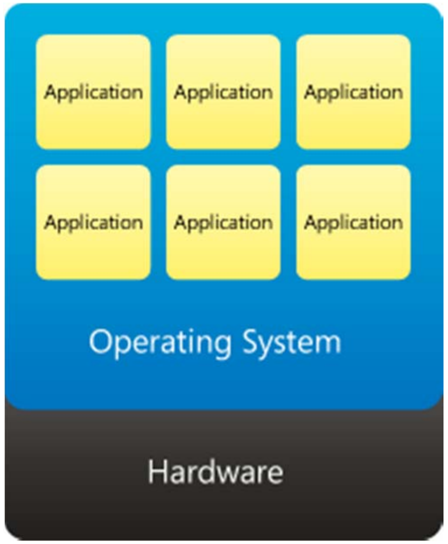
- Separating the application configuration layer from the OS
  - It enables applications to run on clients without being installed, and to be administered from a central location.



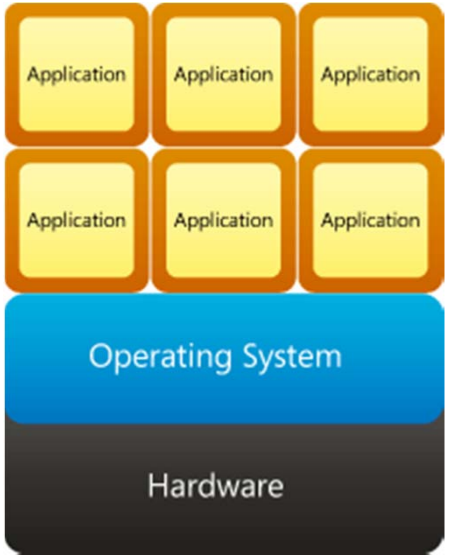
- Application virtualization makes deployment significantly easier.

Application Virtualization

- In a normal computing environment
  - Be installed directly into the OS
  - Since they all write to shared system files, applications will often conflict with one another.



- With application virtualization
  - Run in its own protective runtime environment, isolating them from each other and the underlying OS



Application Virtualization

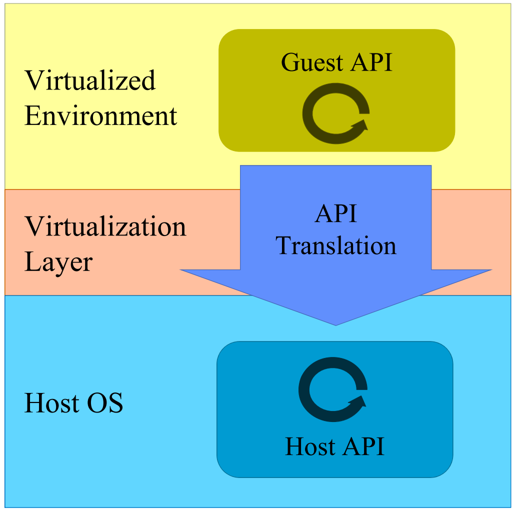
- Criterion
  - Virtualization Target
- Library Virtualization
- High-level Language Virtualization

Application Virtualization

– Library Virtualization

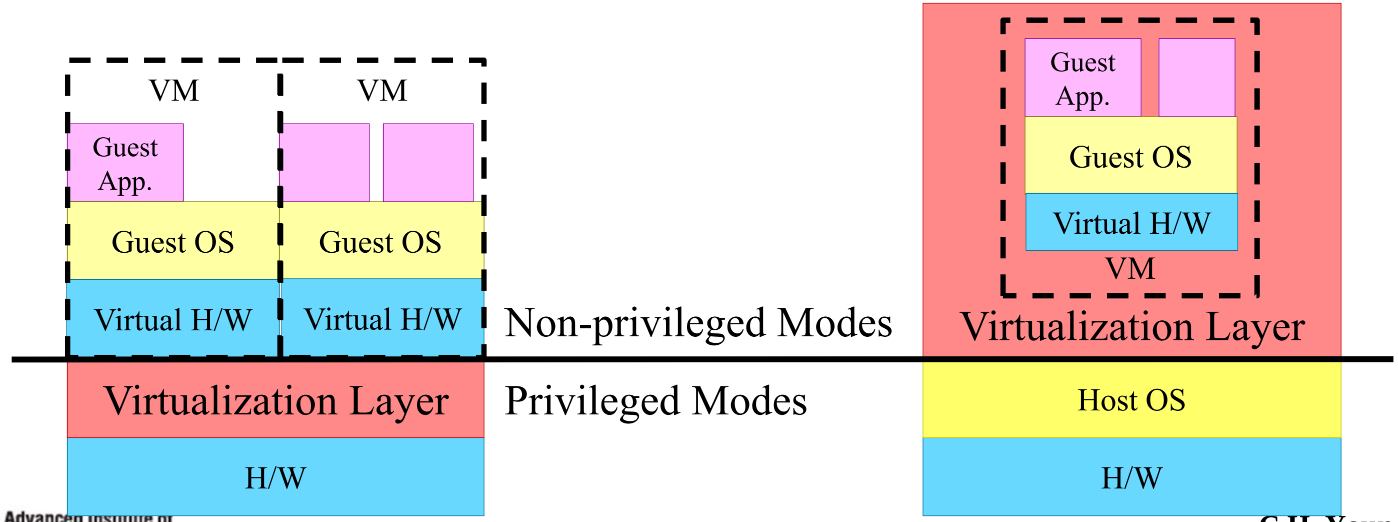
- Providing some environments of the other OS
  - Mapping Guest’s API to Host’s API
  - Not binary translation

- Example
  - cygwin
    - MS Windows → Linux
    - MS Windows ← Linux
    - http://www.cygwin.com/
  - Wine
    - Linux → MS Windows
    - http://www.winehq.org/



System Virtualization

- The ability to run an entire VM with its own (guest) OS on another OS or on a bare-machine
  - Allows multiple VMs, with heterogeneous guest OSes to run in isolation, side-by-side on the same physical machine.
  - Each VM has its own set of virtual H/W upon which a guest OS and guest applications are loaded.
  - The guest OS sees a consistent, normalized set of H/W regardless of the actual physical H/W components.



System Virtualization

- Purpose
  - Several virtual servers share a single set of H/W.
    - Better resource utilization
    - H/W and support costs are lowered.
  - Make easier to provision and reallocate servers
    - Set up a server using a pre-existing template
    - Shift server images from one physical server to another to balance workloads or improve efficiency
  - Provide a secure environment
    - Each servers are isolated from the others.

System Virtualization

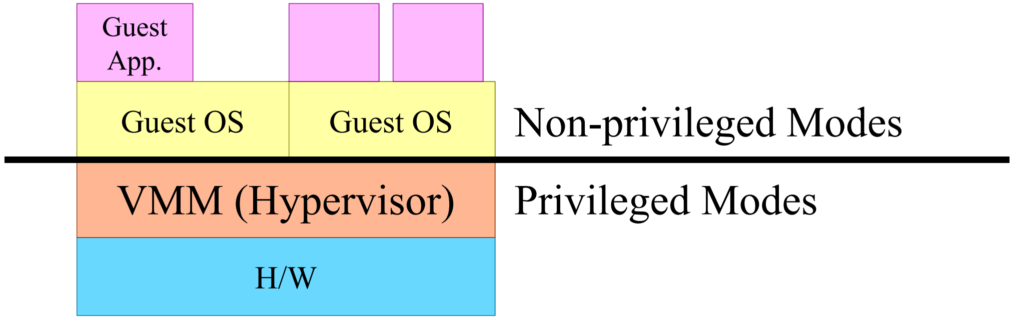
- Criterion
  - Implementation level of the VMM
    - VMM can be implemented and run in an application-level, OS/Kernel level, or hypervisor level.
    - Virtual Machine Monitor (VMM)
      - Create and manage logically separated virtual systems, which can run all OSes or components on the native H/W
      - Hypervisor
        - On the bare hardware

- Native VM System (hypervisor level)
- Hosted VM System (application level)
- OS Extension VM System (OS/Kernel level)

System Virtualization

- Native VM System

- A VM system in which the VMM operates in a privilege mode higher than the mode of the guest VMs.
  - VMM (Hypervisor)
    - Executes in the highest privilege level
    - Installed on the bare H/W
  - Guest OS
    - Installed on top of the VMM
    - Run in levels of privilege lower than that of the VMM
    - The privileged instructions of the guest OS are emulated by the VMM.



System Virtualization

- Native VM System

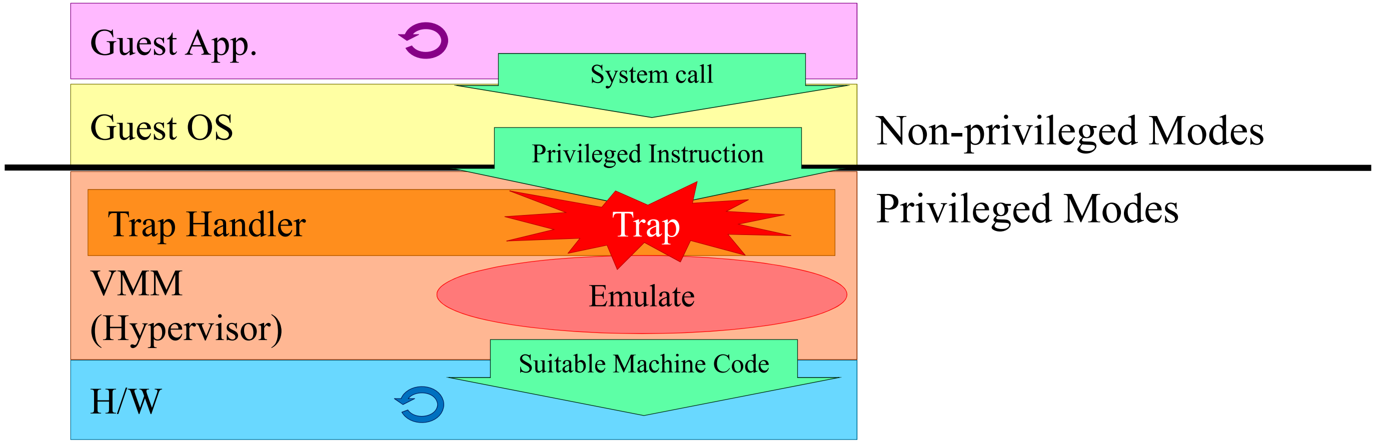
- Criterion
  - Hardware simulation method
    - How handles sensitive and privileged instructions to virtualization

- Full Virtualization
  - Run unmodified guest OS
- Para-virtualization
  - Guest OS should be modified
- H/W Assisted Virtualization
  - Use hardware supports for virtualization
  - Also called, Hardware Virtual Machine (HVM)

Native VM System

- Full Virtualization

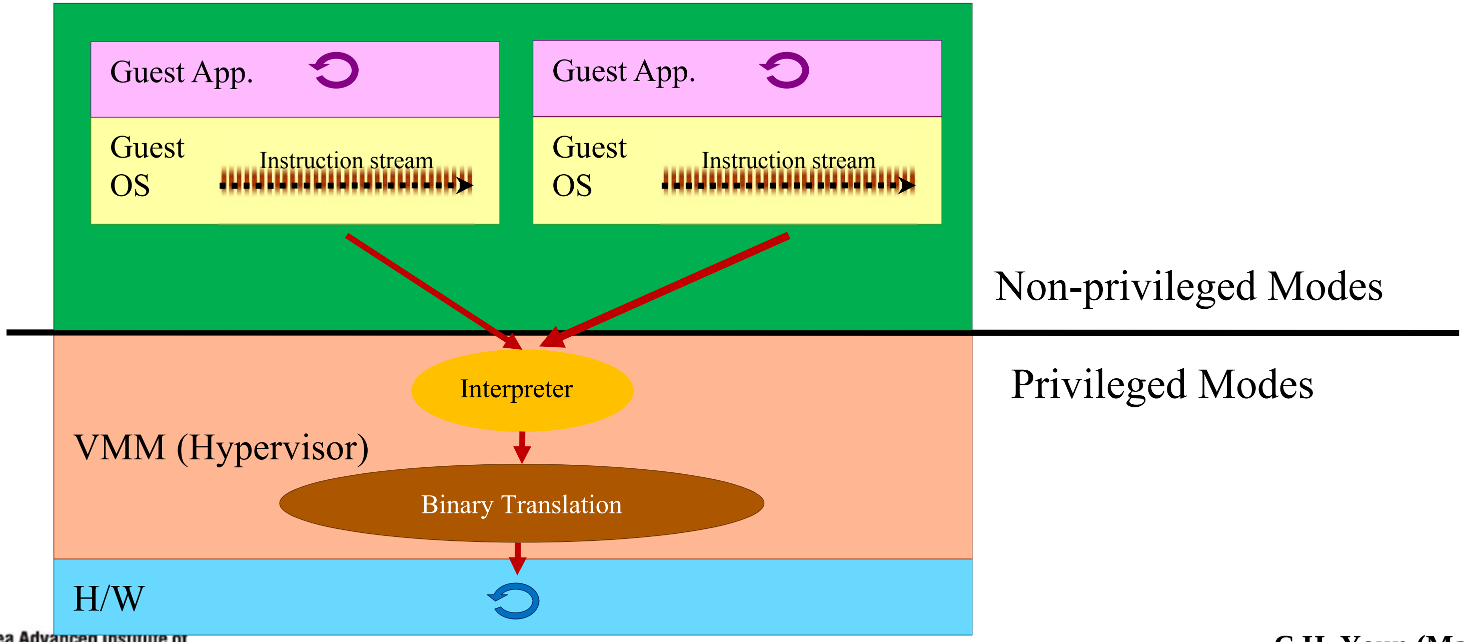
- Classical Virtualization
  - Trap and Emulate
    - All instructions that read or write privileged state can be made to trap when executed in an unprivileged context.
    - The VMM intercepts traps from the de-privileged guest, and emulates the trapping instructions.
    - x86 architectures are not fully *virtualizable* since some privileged instructions do not generate traps when running at user-level.



Native VM System

- Full Virtualization

- Software Virtualization
  - Binary Translation
    - Guest executes on an interpreter instead of directly on a physical CPU
    - Translating the privileged instruction code of the guest to non-privileged instruction or emulating



Native VM System

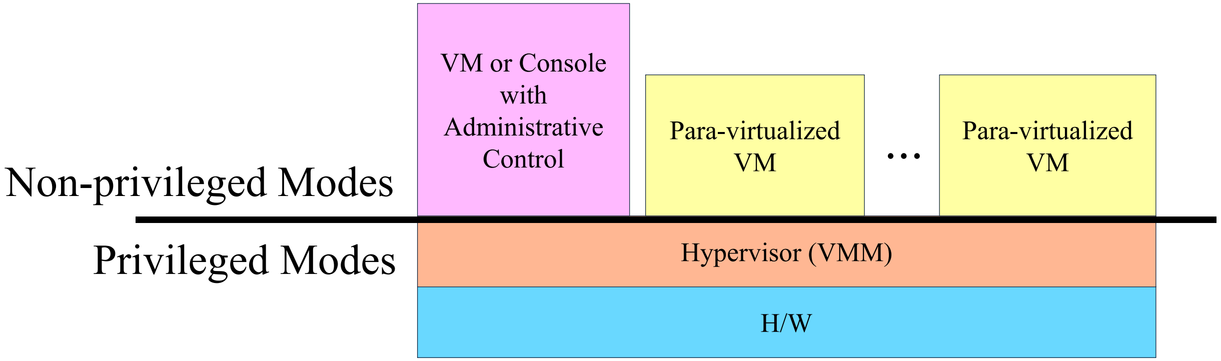
- Full Virtualization

- Enabling unmodified OSes to run on top of the hypervisor
- Performance degradation
  - Caused by trap and emulate
  - Caused by binary translation
  - Compared with Para-virtualization
- Example
  - VMWare ESX Server
    - https://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=%2Feicaz%2Feicazxbcesx.htm
    - http://www.youtube.com/watch?v=JS7u3gGKR3E
    - http://www.trainsignal.com/blog/what-is-vmware-esx-server-and-why-you-need-it

Native VM System

- Para-virtualization

- No trapping
  - Hypercall
    - Modified Guest OS
      - The source code of the OS running in a VM may need to be modified to communicate with the hypervisor using hypercalls.
    - Hypervisor
      - On top of a machine’s H/W
      - Handle queuing, dispatching, and returning the results of H/W requests from VMs

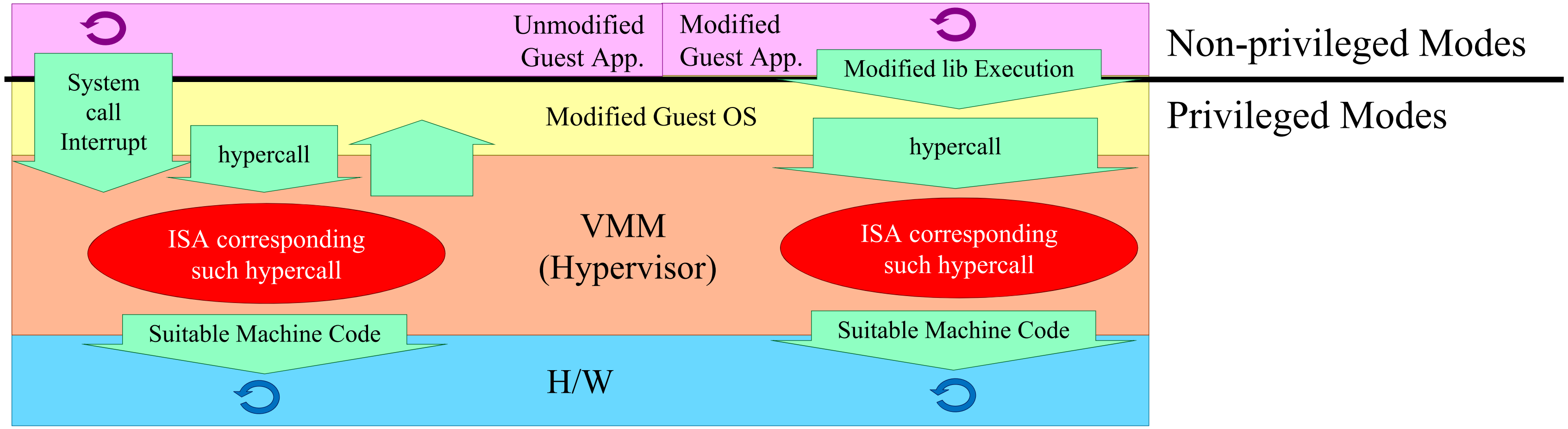


Native VM System

- Para-virtualization

- Administrative OS
  - Runs on top of the hypervisor, as do the VMs themselves
  - Communicate with the hypervisor and be used to manage the VM
- VMs
  - Compiled for the same H/W and instruction set as the physical machine

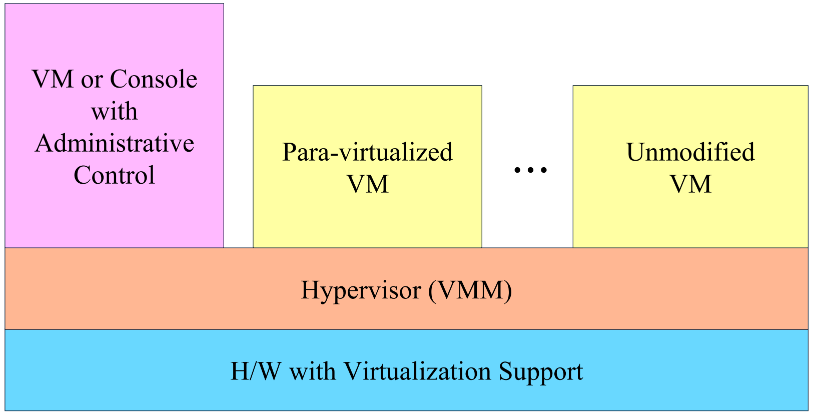
- Example
  - Xen
    - http://xen.org/



Native VM System

- H/W assisted Virtualization

- To avoid
  - Modifying guest OS of para-virtualization
  - The complexity and performance problems of full virtualization
- Support virtualization in H/W
  - Intel VT (aka Vanderpool) and AMD-V (aka Pacifica) processors


VM or Console with Administrative Control
Para-virtualized VM
…
Unmodified VM
Hypervisor (VMM)
H/W with Virtualization Support

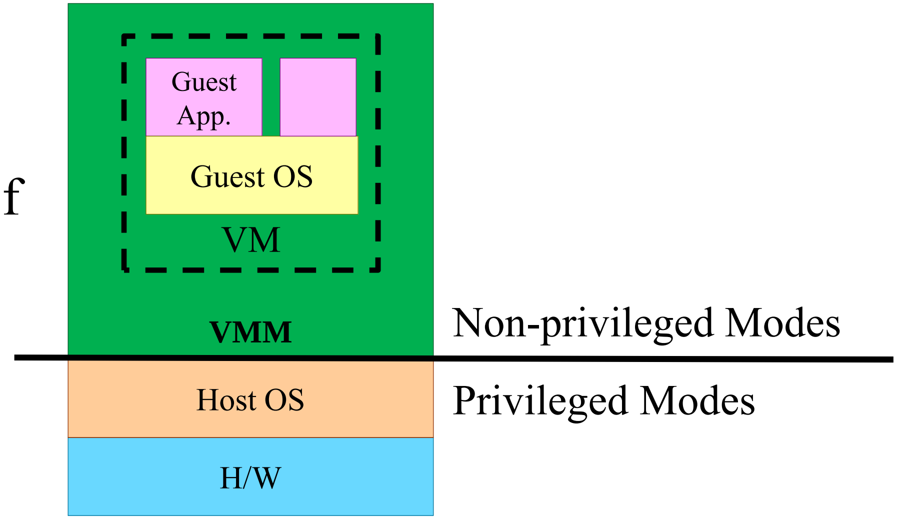
- Supported by
  - VMWare ESX Server
  - Xen 3.0

System Virtualization

- Hosted VM System

- For user convenience and implementation simplicity, it is often advantageous to install a VM system on a host platform that is already running an existing OS.

- VMM (Virtualization Application)
  - Utilize the functions already available on the host OS to control
  - Manage resources desired by each of the VMs
- Can be used when
  - Source code is unavailable
  - Guest OS needs licensing agreements



System Virtualization

- Hosted VM System

- Criterion
  - Hardware simulation method
    - How handles sensitive and privileged instructions to virtualization
- Full Virtualization
  - Run unmodified guest Os
- Para-virtualization
  - Guest OS should be modified
- H/W Assisted Virtualization
  - Use hardware supports for virtualization
  - Also called, Hardware Virtual Machine (HVM)

Hosted VM System

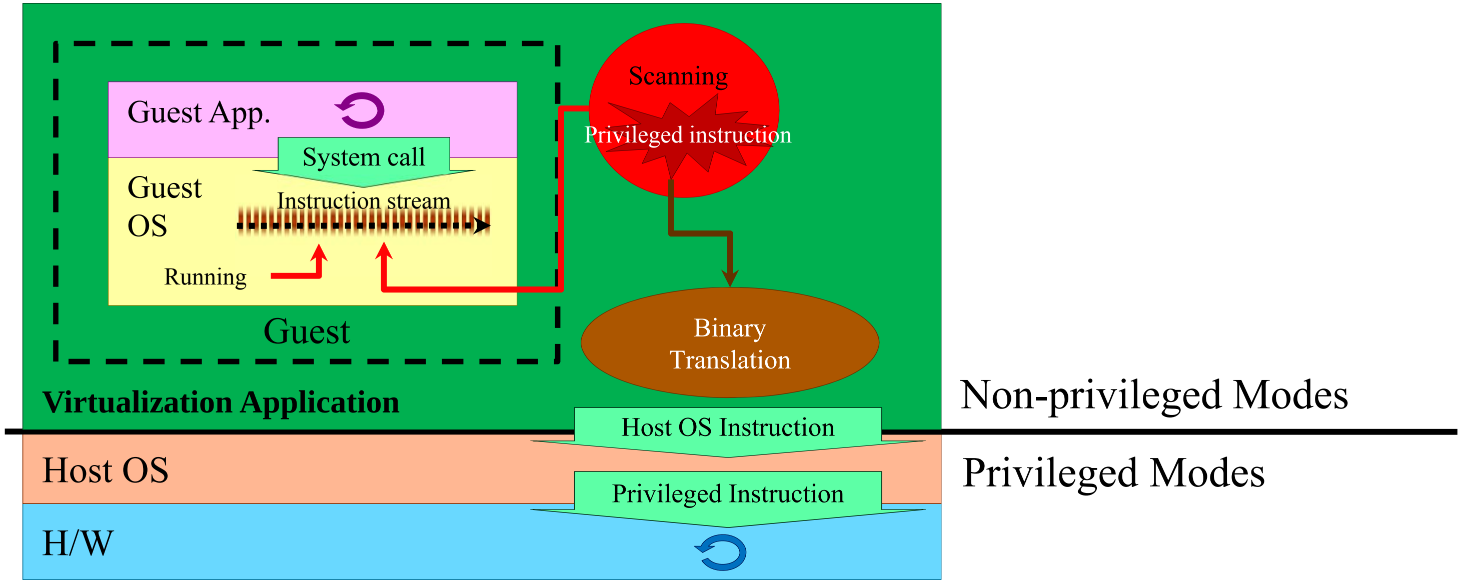
- Full Virtualization

- All processes in guest OS are scheduled as one process by host OS.
- VMs run within a virtualization application.
  - Virtualization Application
    - Manages the VMs
    - Mediates access to the H/W resources on the physical host system
    - Intercepts and handles any privileged instruction issued by the VMs
  - Guest OS
    - Compile for the same type of processor and instruction set as the physical machine
    - If the virtualization application can perform instruction set translation or emulation, it can be compiled for other processors

Hosted VM System

- Full Virtualization

- Binary Translation
  - Scanning the running instruction stream
  - For non-trapping,
    - Translating the privileged instruction code of the guest to non-privileged instruction or emulating



Hosted VM System

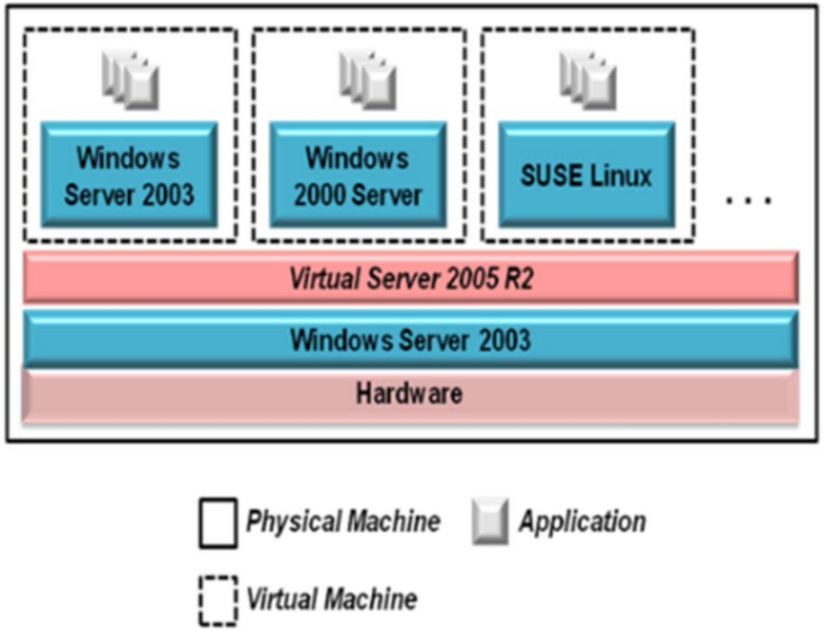
- Full Virtualization

- Criterion
  - Emulating whether machine dependent or independent
- Machine dependent emulation
- Machine independent emulation

Full Virtualization

- Machine dependent emulation

- The virtualization application do not emulate the requests of each VMs.
- Compile for the same type of processor and instruction set as the physical machine
- Example
  - VMWare Workstation
    - http://www.vmware.com/products/ws/overview.html
  - Microsoft Virtual Server
    - http://www.microsoft.com/windowsserversystem/virtualserver/
  - VirtualBox
    - http://www.virtualbox.org



Full Virtualization

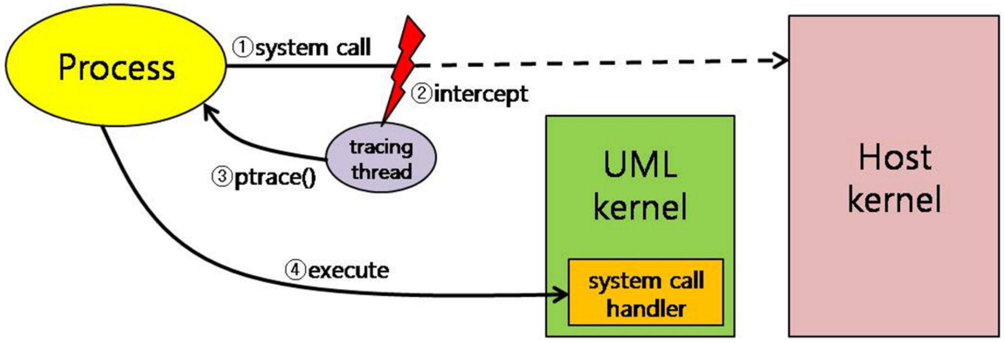
- Machine independent emulation

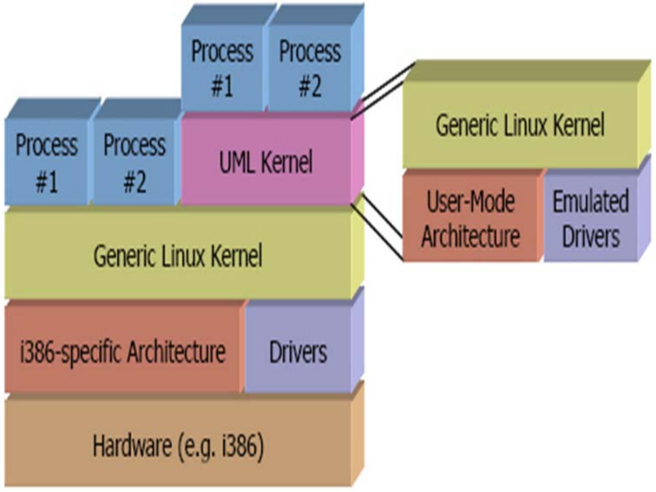
- Providing the functionality of the target H/W completely in S/W
  - Dynamic translation
  - Emulating an architecture using a completely different architecture
- Example
  - QEMU
    - http://fabrice.bellard.free.fr/qemu/
    - Emulates x86, x86_64, PowerPC, SPARC, ARM, MIPS
    - Runs on Linux, Windows, Mac OS X
    - Runs Linux, Solaris, Microsoft Windows, DOS, and BSD
  - Bochs
    - http://bochs.sourceforge.net/
    - Emulates an x86 PC
    - Runs on UNIX, Linux, Windows, Mac OS X, BeOS, OS/2, etc.
    - Runs Linux, DOS, Windows (95, NT), MacOS X

Hosted VM System

- Para-virtualization

- All processes in host OS and guest OSes are scheduled as processes by host OS.
  - But processes running in guest have to be handled by each guest OS.
- The guest OS kernel traps and manages the requests created by its own processes.
- Example
  - UML
    - http://user-mode-linux.sourceforge.net/
    - http://wiki.kldp.org/wiki.php/UserModeLinux





Hosted VM System

- H/W Assisted Virtualization

- Example
  - KVM
  - VMWare Workstation
  - Virtual PC
  - MS Virtual Server

System Virtualization

- OS Extension VM System

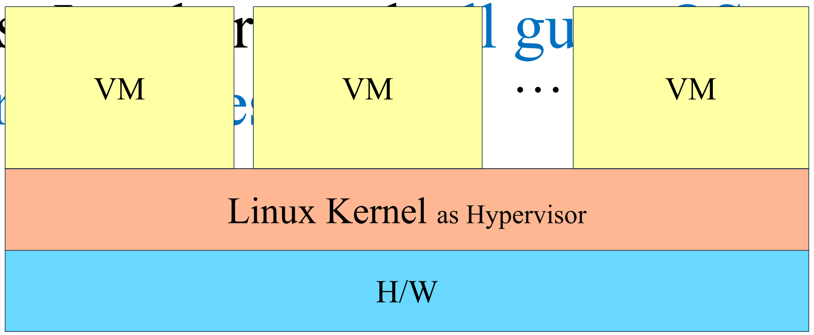
- VMM is implemented as part of a host OS.

- Criterion
  - Whether Shared kernel or not
- Kernel Virtualization
- OS-level Virtualization

OS Extension VM System

- Kernel Virtualization

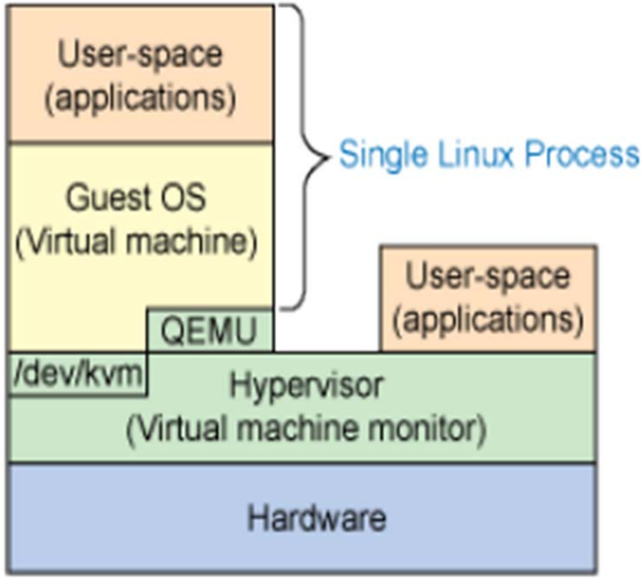
- The Linux kernel runs the VMs, just like any other user-space process.
  - Host Linux kernel
    - Runs a separate version of the Linux Kernel
    - In point of host OS's view, all processes in a guest OS are just one user process r l gu OS are scheduled as regular e
  - VM
    - Has an its own OS
    - Compiled for the same H/W and instruction set as the Host Linux kernel



OS Extension VM System

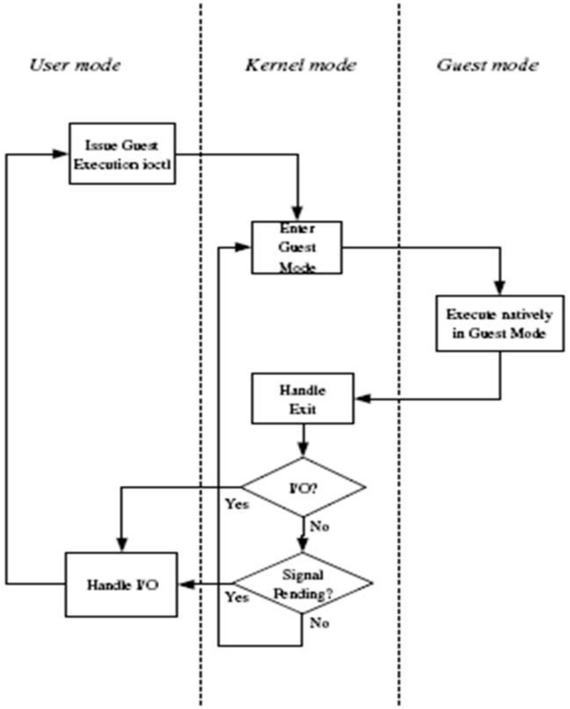
- Kernel Virtualization

- Example
  - KVM (Kernel Virtual Machine)



m/

ı/cc



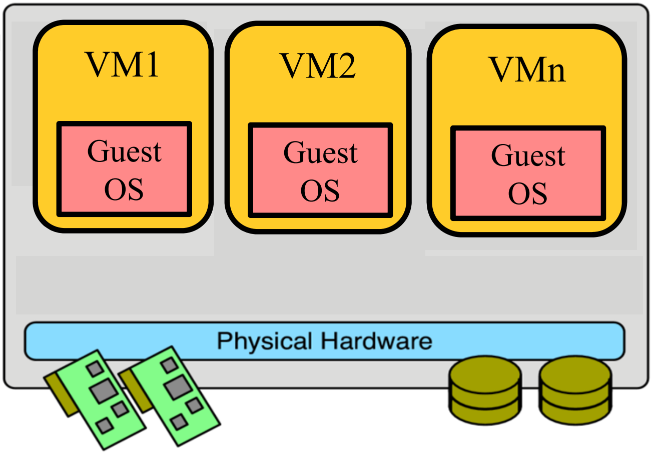
OS Extension VM System

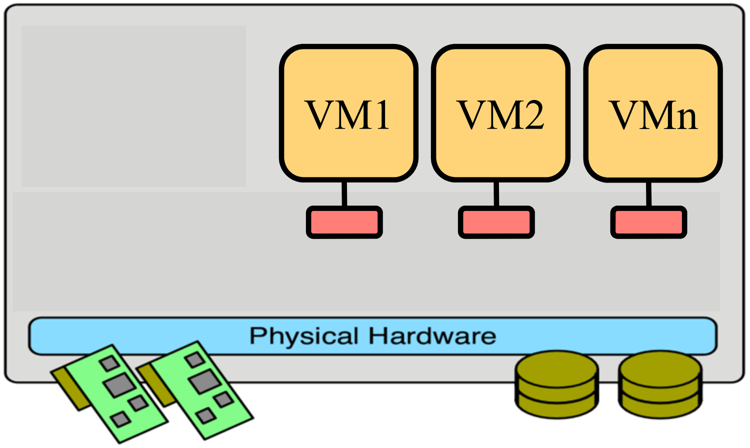
- OS-level Virtualization

- The kernel of an OS allows for multiple isolated user-space instances (instead of just one).
  - From the point of view of its owner, such instances (called containers) look and feel like a real server.
  - chroot (change root) mechanism

sk ro

child

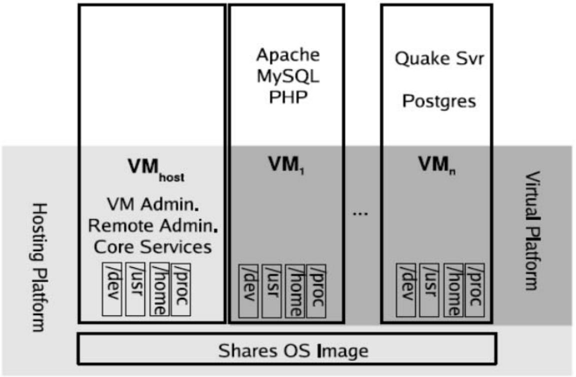




OS Extension VM System

- OS-level Virtualization

- Example
  - Linux-VServer
    - http://linux-vserver.org/
    - ttp://www.linux.com/feature/59



  - OpenVZ/Virtuozzo
    - http://openvz.org/
    - http://www.parallels.com/en/products/virtuozzo/

Hypervisor for Client Devices

- Security, Manageability and Supportability
  - Virtual Appliances pre-installed with various applications
    - Virtual appliances are a subset of the broader class of software appliances -> aimed to eliminate the installation, configuration, and maintenance costs associated with running complex stacks of software (e.g., IDS, Malware detection, remote access, backup, etc) (http://www.vmware.com/appliances)
    - Can solve Grid computing problem -> appliances for Grid computing (http://www.grid-appliance.org)
  - Building multi-level secure systems
    - Run multiple guest VMs with very controlled information flow
      - Enable Bring-Your-Own-PC model
      - Corporate VM; VM for web browsing; VM for banking
      - Migration of VMs between datacenter and laptops for office use

From Laptops and Mobiles

- Smart phones and PDAs
  - Smart phones now suffer from many of the same problems as PCs
  - Xen ARM

- Simple restricted use cases:
  - Three VMs running on one CPU:
    - Real time VM for controlling the radio
    - VM for vendor/operator – supplied S/W
    - VM for user-downloaded software

Hypervisor for Servers

- Computing clouds (Cloud computing)
  - Refers to computing resources being accessed which are typically owned and operated by a third-party provider on a consolidated basis in datacenter locations.
  - Consumers of cloud computing services purchase computing capacity on-demand.
  - Virtualization provides dynamic infrastructure for Software as a service (Saas).
  - Example: Amazon Elastic Compute Clouds (EC2)
- Green IT computing
  - Virtualization is one of the solutions to reduce the power consumption in the datacenter environments (via consolidation or migration).
    - Green Grid (http://www.thegreengrid.org/home)
    - Climate Savers Computing Initiative (http://climatesaverscomputing.org)

Hypervisor for Network Virtualization

- PlanetLab (http://www.planet-lab.org)
  - A group of computers available as a test-bed for computer networking and distributed systems research.
    - As of June, there are 880 nodes at 460 sites worldwide.
    - Each research project has a “slice” or virtual machine.
    - Linux V-Server is used for the slice.

- GENI – Global Environment for Network Innovations (http://www.geni.net)
  - Enhance experimental research in networking and distributed systems, and to accelerate the transition of this research into products and services -> Future Internet.
  - Use (initially) Xen for slice.