

<Slides download>

<https://www.pf.is.s.u-tokyo.ac.jp/classes>

# Advanced Operating Systems

## #7

Hiroyuki Chishiro

Project Lecturer

Department of Computer Science

Graduate School of Information Science and Technology

The University of Tokyo

# Course Plan

- Multi-core Resource Management
- Many-core Resource Management
- GPU Resource Management
- Virtual Machines
- Distributed File Systems
- High-performance Networking
- Memory Management
- Network on a Chip
- Embedded Real-time OS
- Device Drivers
- Linux Kernel

# Schedule

1. 2018.9.28 Introduction + Linux Kernel (Kato)
2. 2018.10.5 Linux Kernel (Chishiro)
3. 2018.10.12 Linux Kernel (Kato)
4. 2018.10.19 Linux Kernel (Kato)
5. 2018.10.26 Linux Kernel (Kato)
6. 2018.11.2 Advanced Research (Chishiro)
7. 2018.11.9 Advanced Research (Chishiro)
8. 2018.11.16 (No Class)
9. 2018.11.23 (Holiday)
10. 2018.11.30 Advanced Research (Kato)
11. 2018.12.7 Advanced Research (Kato)
12. 2018.12.14 Advanced Research (Chishiro)
13. 2018.12.21 Linux Kernel
14. 2019.1.11 Linux Kernel
15. 2019.1.18 (No Class)
16. 2019.1.25 (No Class)

# Virtual Machines and Distributed File Systems

Introducing virtual machine and distributed file systems

/\* The cases for VMware Workstation and Xen in virtual machines \*/

/\* The case for NFS in distributed file systems \*/

*Acknowledgement:*

*Prof. Kenji Kono, Virtualization Technology of OS, Keio*

*Prof. Andrew S. Tanenbaum and Prof. Maarten Van Steen,  
Distributed Systems: Principles and Paradigms, Vrije Universiteit*

# Outline

- Virtual Machine
- VMWare Workstation
- Xen
- Distributed File Systems

# Outline

- Virtual Machine
- VMWare Workstation
- Xen
- Distributed File Systems

# Why Virtual Machine?

- 1960-1970s: shares expensive mainframe easily (IBM TSS/360 costs \$50 million)
  - runs several OSES on same machine.
  - runs single-user single-tasking OS in time-sharing system.
  - NOTE: IBM VM System/370 is the first virtual machine in 1967.
- 1980-1990s: goes out of fashion.
  - Mainframes become cheaper and cheaper.
- 2005: becomes hot topic in industry and academia [Rosenblum 05]
  - Industry (Intel, AMD, Sun Microsystems, and IBM) develops virtualization technologies.
  - Academia solves mobility, security, and manageability problems.

[Rosenblum 05] Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors: Current technology and future trends. IEEE Comp., Vol. 38, No. 5, pp. 39–47, 2005.

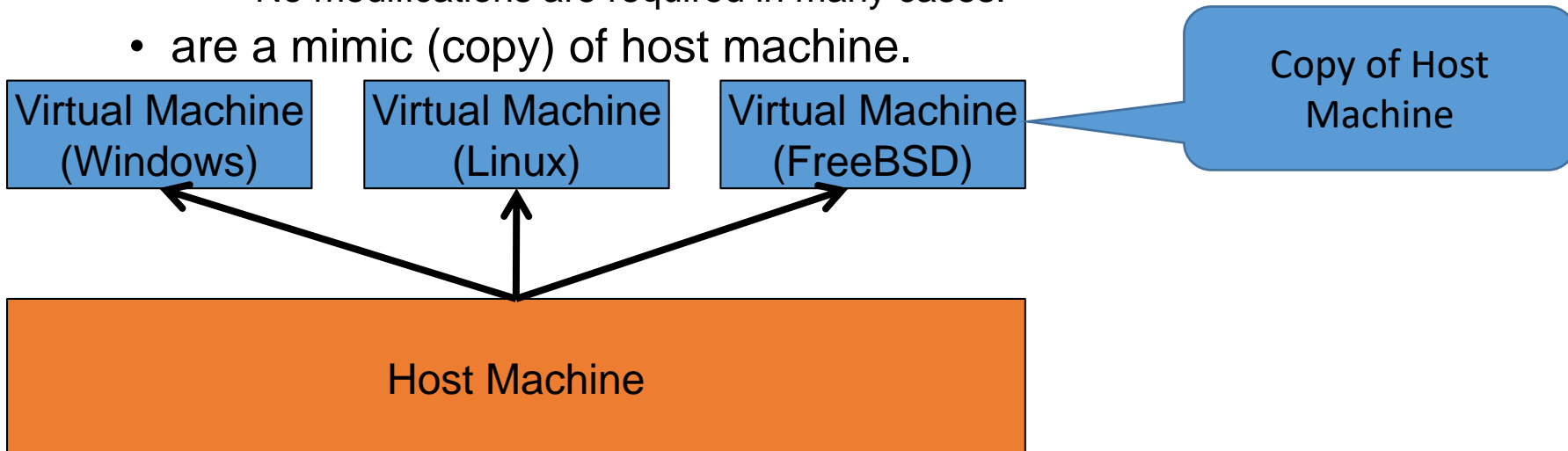
# Types of Virtual Machines

- System Virtual Machines:
  - provide the platform for *operating system* by Instruction Set Architecture (ISA) Level Virtualization.
  - E.g, IBM 370, VMWare
- Process Virtual Machines:
  - provide the platform for *single program (process)* by Application Binary Interface (ABI) Level Virtualization.
  - E.g.: Java VM



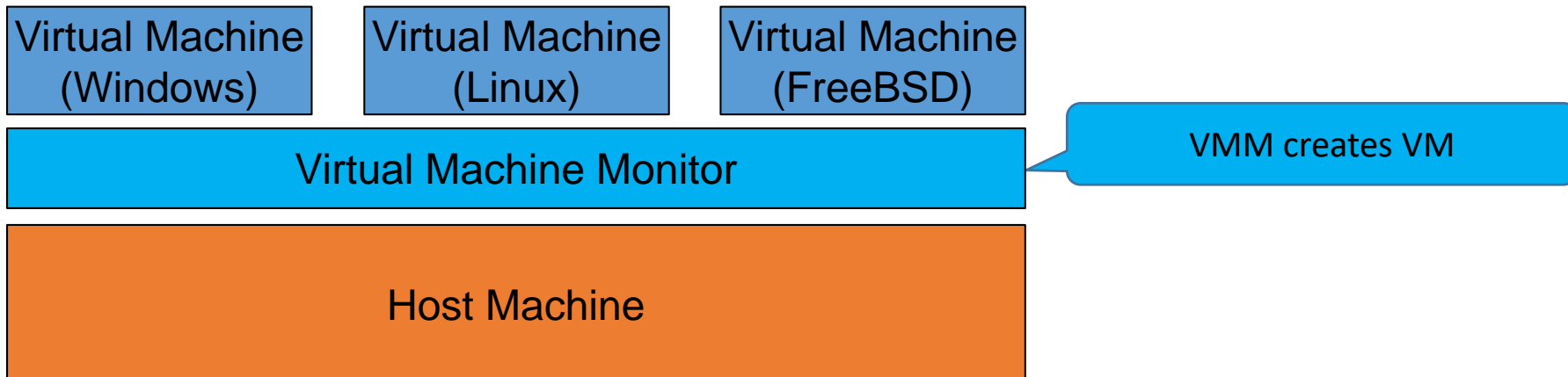
# Virtual Machine Technology

- can use a single machine as multiple machines.
- can run multiple operating systems on a single machine.
- Virtual Machines:
  - are software that runs operating systems and applications like physical computers.
    - No modifications are required in many cases.
  - are a mimic (copy) of host machine.

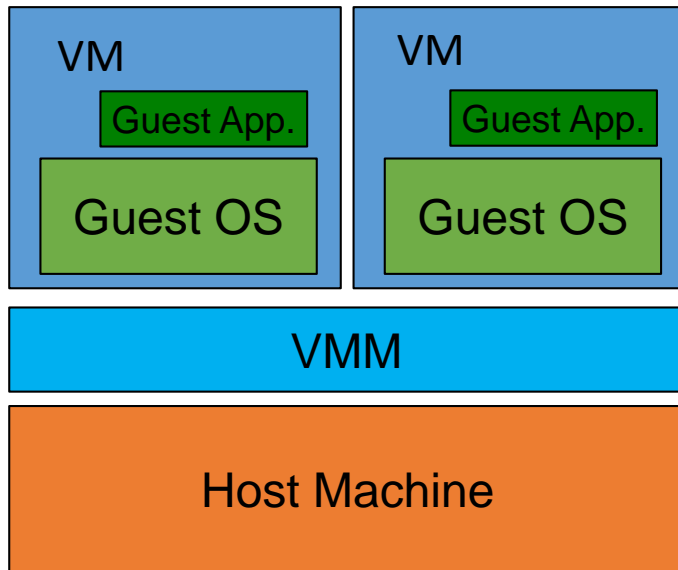


# Virtual Machine Monitor (Hypervisor)

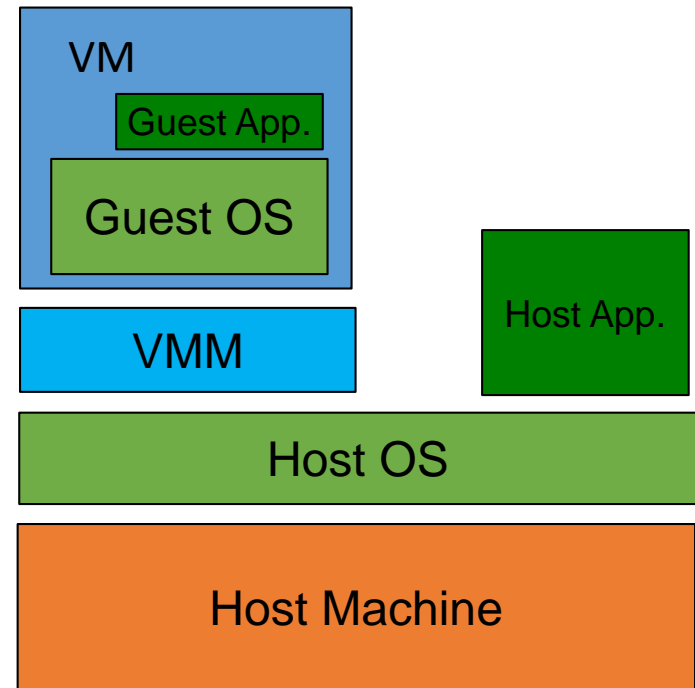
- is software to create virtual machines
  - Virtual Machine (VM) != Virtual Machine Monitor (VMM)
- Guest Application: Software on VM
- Guest OS: Software on VMM



# Categorization of VMM



**Type 1 VMM:** Native (Bare Metal) VMM  
VMM running on **Host Machine**  
E.g.: Xen, VMWare ESXi



**Type 2 VMM:** Hosted VMM  
VMM running on **Host OS**  
E.g.: VMWare Workstation, Virtual Box, Qemu

# Categorization of Virtualization

- Full Virtualization
  - performs guest OS without modification.
- Paravirtualization
  - performs guest OS with modification.
- Software Virtualization
  - achieves virtualization by only software.
- Hardware Virtualization
  - achieves virtualization by hardware capabilities.

# Examples of Virtualization

- Full Virtualization + Software Virtualization
  - VMWare Workstation, Qemu
- Paravirtualization + Software Virtualization
  - Xen
- Hardware Virtualization
  - Intel VT
  - AMD-V

# Popek and Goldberg Virtualization Requirements [Popek 74]

- Equivalence/Fidelity
  - Any program running under VMM should be identical as well as running on an equivalent machine directly.
- Efficiency/Performance
  - VMM must control the virtualized resources completely.
- Resource Control/Safety
  - Machine instructions must be executed without VMM intervention (interference).

[Popek 74] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. Communications of ACM, Vol. 17, No. 7, pp. 412-421, July 1974.

# CPU Virtualization

- issues the instructions in the guest OS safely:
  - including privileged instructions.
- does not interference VMM and other VMs.
- If CPU time is directly assigned to VM:
  - Guest OS issues the privileged instructions.
  - Since guest OS intends to isolate the host machine, it interferences with VMM and other VMs.

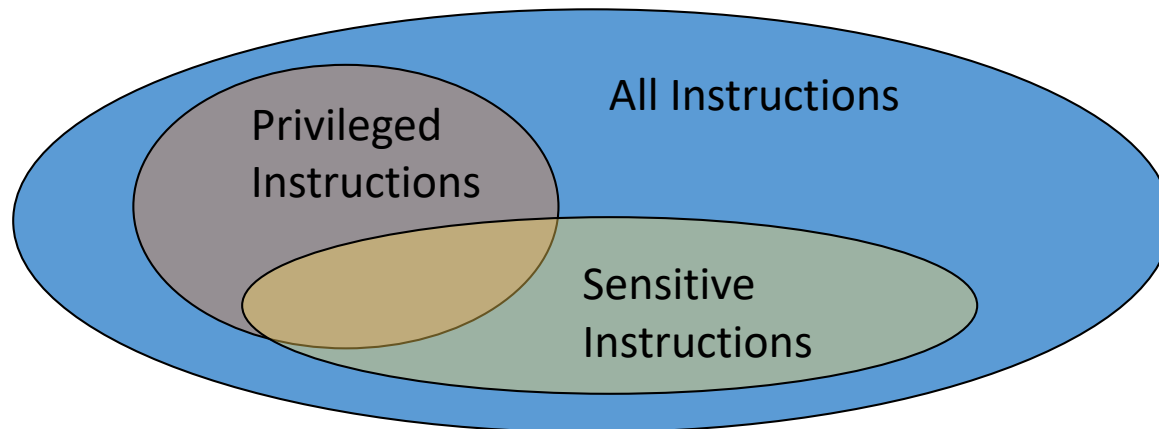
# De-Privileging

- runs guest OS at user-level to protect VMM and other VMs.
  - generates protection exception if privileged instructions are issued.
    - VMM handles this exception.
  - emulates privileged instructions in VMM.
    - VM misunderstands if privileged instructions are issued successfully.
- Example of emulation of privileged instruction:
  - A VM issues *cli (interrupt disable)* instruction.
  - VMM generates protection exception and handles this exception.
  - VMM waits interrupts to the VM with interrupt disable.
    - NOTE: Real CPU does not disable interrupts because other VMs may wait them.



# Sensitive Instructions

- must not be issued by guest OS.
  - Interference with VMM and guest VM
  - E.g.: I/O instruction (*in/out* instructions)
- Non-sensitive instructions are **innocuous instructions**.



# Examples of Sensitive Instructions (1/2)

- Control-Sensitive Instructions:
  - get/set the status of system.
  - Example 1: instruction to disable interrupt
    - A guest OS must not control the interrupts in the overall system.
  - Example 2: instruction to get the processor mode
- Problem of Example 2 is as follows.

```
m = get_processor_mode();  
If (m != privileged) {  
    /* not work well because the guest OS must not issue sensitive instructions.*/  
    panic();  
}
```

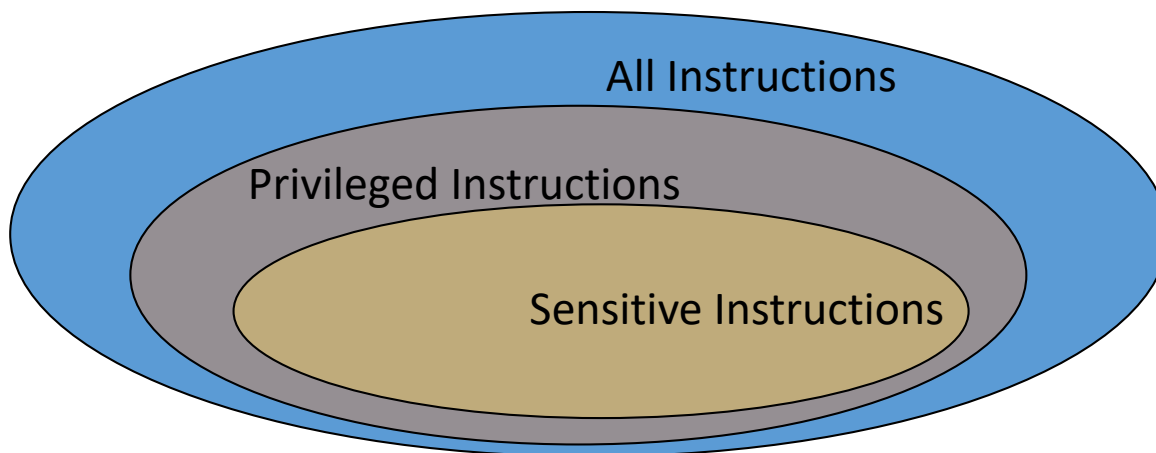
# Examples of Sensitive Instructions (2/2)

- Behavior-sensitive instructions:
  - change the behavior depending on processor mode.
    - E.g.: POPF instruction in Intel x86
      - pops top of stack into lower 16bits of EFLAGS.
      - NOTE: Interrupt enable flag is in EFLAGS.
    - If interrupt enable flag is changed by using POPF...
      - do the change of interrupt enable flag in privileged mode.
      - ignore the change of interrupt enable flag in non-privileged mode.
        - No protection exception occurs.
- Problem of issuing POPF in guest OS:

```
/* Value with interrupt disable flag is in the top of stack */  
POPF /* try to set interrupt disable flag */  
/* Interrupt disable region (but interrupt is enable  
due to ignoring the change of interrupt enable flag) */
```

# (Typical) Virtualizable CPU

- Virtualizable CPU means that sensitive instructions are subset of privileged instructions.



- If the above inclusion relation holds:
  - Guest OS runs in non-privileged mode.
  - Guest OS cannot issue sensitive instructions.
  - If guest OS issues sensitive instructions, protection exception occurs.

# Is Intel x86 Virtualizable?

- No (Typical Meaning)
- There are 17 sensitive but non-privileged instructions.
  - SGDT, SIDT, SLDT
  - SMSW
  - PUSHF, POPF
  - LAR, LSL, VERR, VERW
  - ...
- Challenge is how to virtualize Intel x86.

# Trap and Emulate

- are implementation techniques for typical VMM.
  - VMM runs in privileged mode and guest OS runs in non-privileged mode.
- If guest OS and applications run:
  - Innocuous instructions are issued directly.
  - Sensitive instructions are trapped (generate software interrupt) and VMM handles the software interrupt.
- VMM emulates the sensitive instructions trapped as expected for guest OS.

# Memory Virtualization

- Each VM misunderstands to isolate physical memory, i.e.:
  - Address of physical memory starts 0 and is continuous.
- Actually, all VMs share the physical memory in host machines.
  - The address of physical memory does not always start 0 and is not always continuous.
- However, VMM would like each VM to misunderstand that the address of physical memory starts 0 and is continuous.

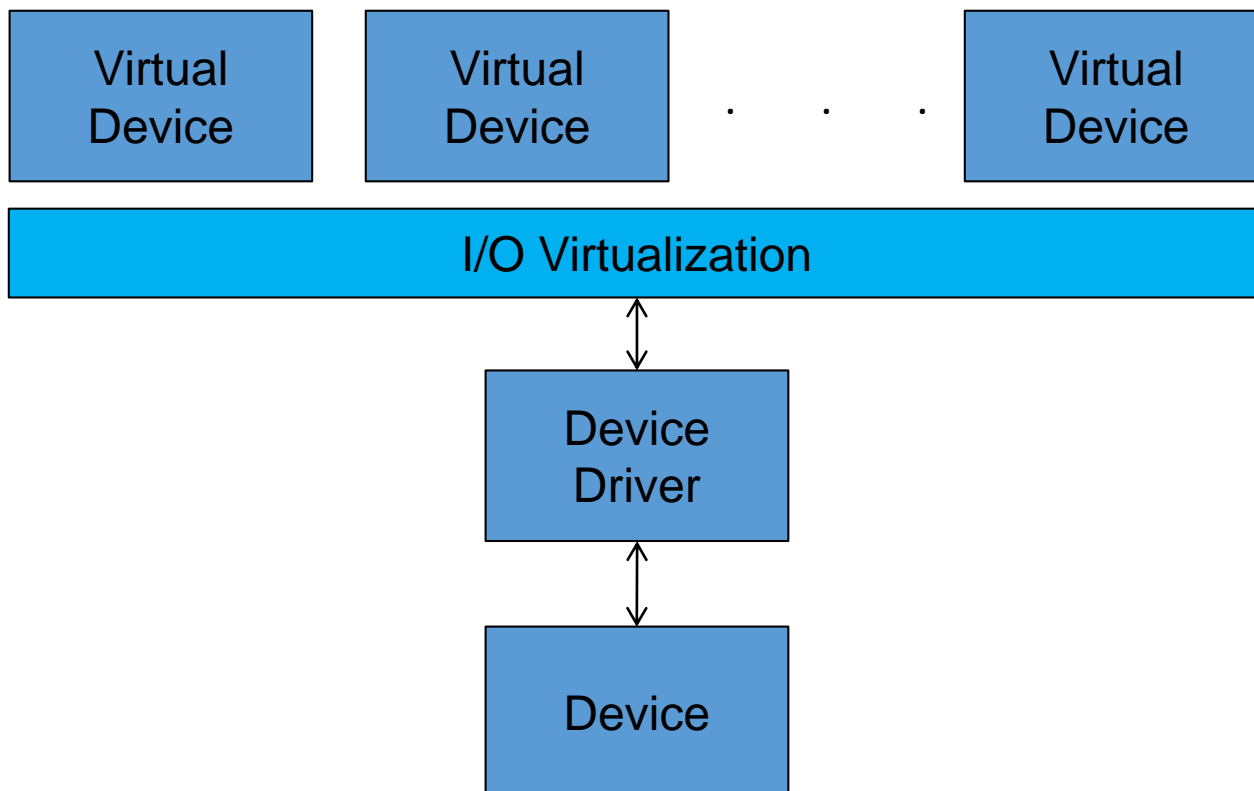
# Second Level Address Translation (SLAT)

- is required to achieve memory virtualization.
  - First Level Address Translation by Guest OS
    - Virtual Address -> Guest Physical Address
  - Second Level Address Translation by VMM
    - Guest Physical Address -> Host Physical Address
- Terms:
  - Virtual Address (guest virtual addr, gVA)
    - Virtual Address in VM
  - Guest Physical Address (guest physical addr, gPA)
    - Physical Address in VM
  - Host Physical Address (host physical addr, hPA)
    - Physical Address in Host Machine
- NOTE: In general, MMU cannot perform SLAT.
- Requirement: SLAT by using MMU.



# I/O Virtualization

- Guest OS misunderstands to isolate I/O devices.
  - I/O devices must be multiplex.
  - Devices accessed by guest OS are virtual devices.



# Primary Registers in Intel x86

- General Purpose Registers
  - EAX, EBX, ECX, EDX, ESI, EDI
  - EBP: Base Pointer
  - ESP: Stack Pointer
- Segment Registers
  - CS: Code Segment
  - DS: Data Segment
  - SS: Stack Segment
  - ES, FS, GS: Extra Segment
- Control Registers
  - CR0, CR4

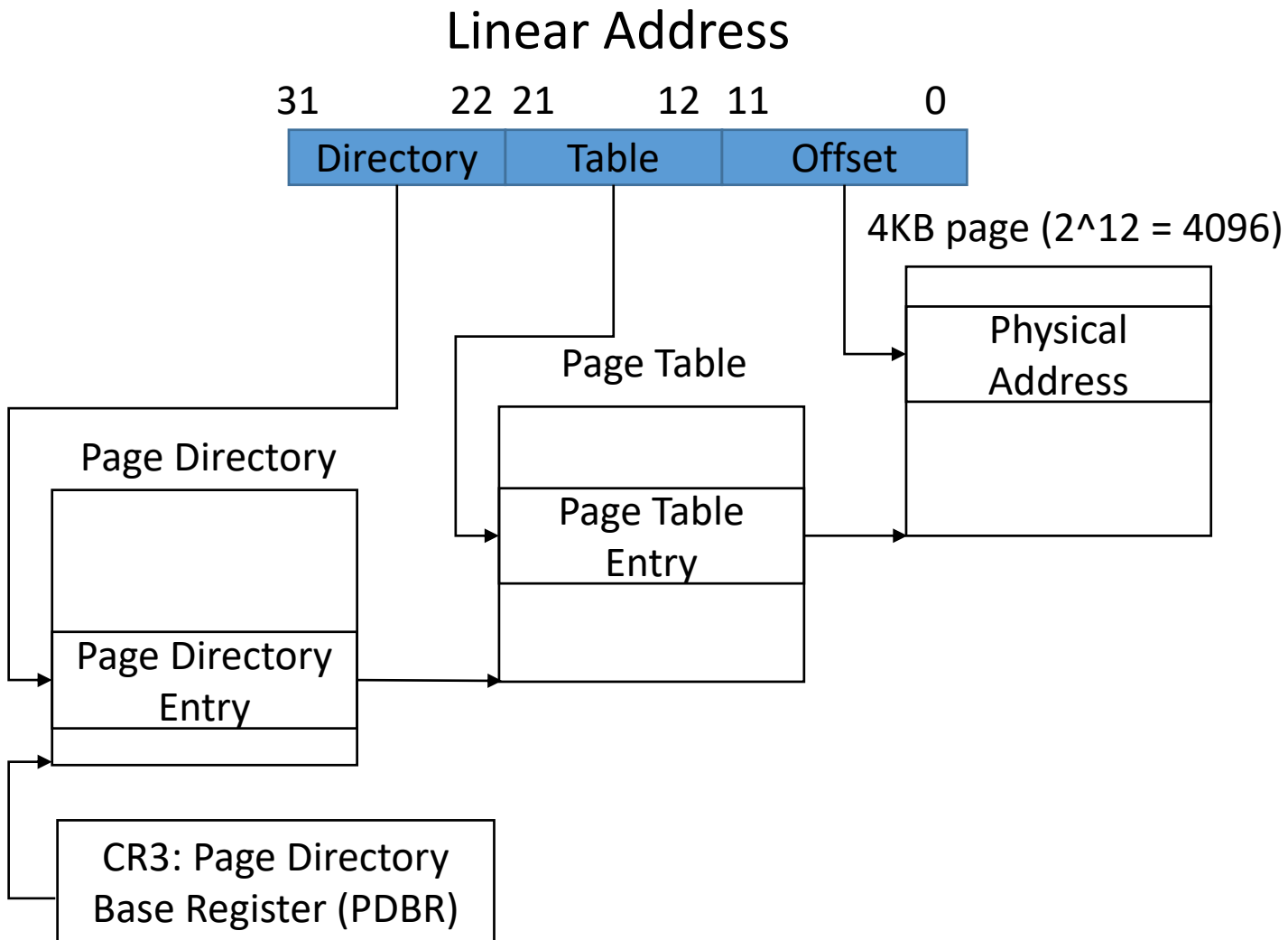
# Segmentation in Intel x86

- SLAT
  - Translation by Segmentation
    - Virtual Address -> Linear Address
  - Translation by Paging
    - Linear Address -> Physical Address
- Structure of Segmentation
  - specifies segment descriptor by segment selectors.
    - Segment descriptor includes segment selectors.
  - Segment descriptor:
    - has start address and length.
    - cannot access the upper address of segment.
    - also has Descriptor Privilege Level (DPL) about access right etc.
  - Calculation of linear address is:
    - Linear address = start address of segment + effective address.

# Intel Terms

- Interrupt Descriptor Table (IDT)
  - has a table of code segment for interrupt handler and offset of segment.
- Interrupt Descriptor Table Register (IDTR)
  - registers to specify the start address of IDT.
- Page Table
  - CR3: Register to specify the head of page table
  - Intel x86 (32bits) has second level page tables.
    - 32-22bits: Page Directory (PD)
    - 21-12bits: Page Table (PT)
    - 11-0bits: Offset
  - Hardware-walk and not software TLB
  - No TLB tag

# Linear Address Translation



# Linux Namespaces/Cgroups

## [namespaces,cgroups]

- They are **not virtual machines**.
  - No ISA/ABI level virtualizations
- They are container virtualization for processes.
  - Operating system level virtualization
- Namespaces control the access of resources.
  - E.g.: Mount, Network, PID, etc.
  - mainlined since Linux 2.4.19 (August 2002)
- Cgroups control the amount of resources.
  - E.g.: CPU, Memory, I/O, etc.
  - mainlined since Linux 2.6.24 (January 2008)

[namespaces] [http://doc.cat-v.org/plan\\_9/4th\\_edition/papers/names](http://doc.cat-v.org/plan_9/4th_edition/papers/names)

[cgroups] <https://lwn.net/Articles/236038/> (named cgroups as process container)

# Outline

- Virtual Machine
- VMWare Workstation
- Xen
- Distributed File Systems

# Contribution of VMware Workstation

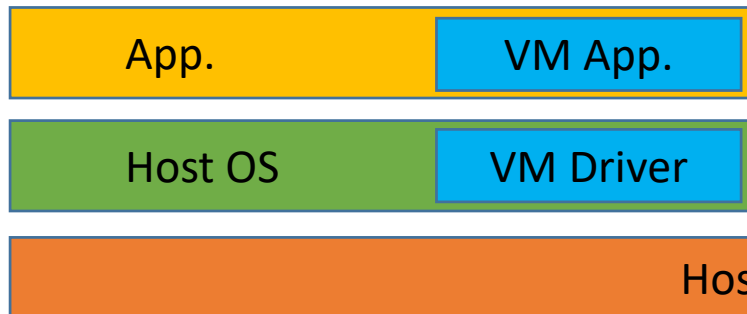
- achieves virtualization for commodity PC (Full Virtualization).
- Challenging to Virtualization
  - Intel x86 is not (typical) virtualizable.
  - Intel x86 has sensitive but non-privileged instructions.
  - Diversity of PC hardware
    - There are many kinds of PC devices.
    - VMM cannot have all device drivers.
  - Many users would like to use preinstall software (OS and applications) continuously.



# VMware Workstation Architecture: Hybrid of Types 1 and 2 [Sugerman 01]

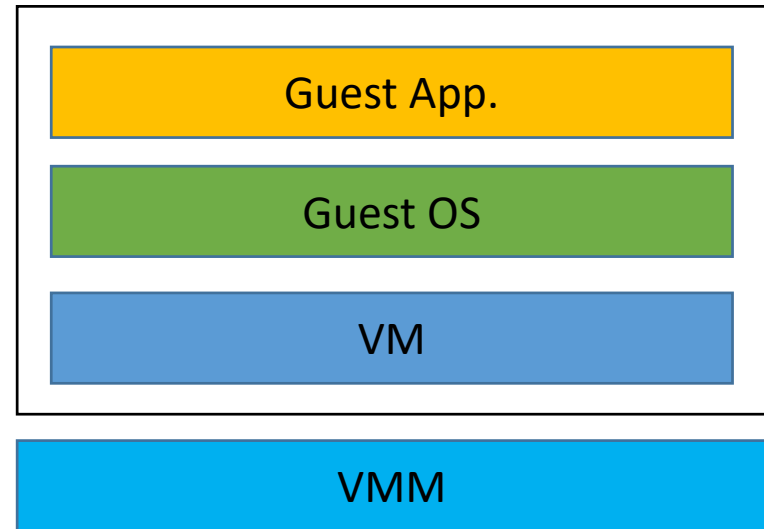
OS and applications in Host World are as they are.

## Host World



VMM requests Host OS to access I/O devices in VMM world.

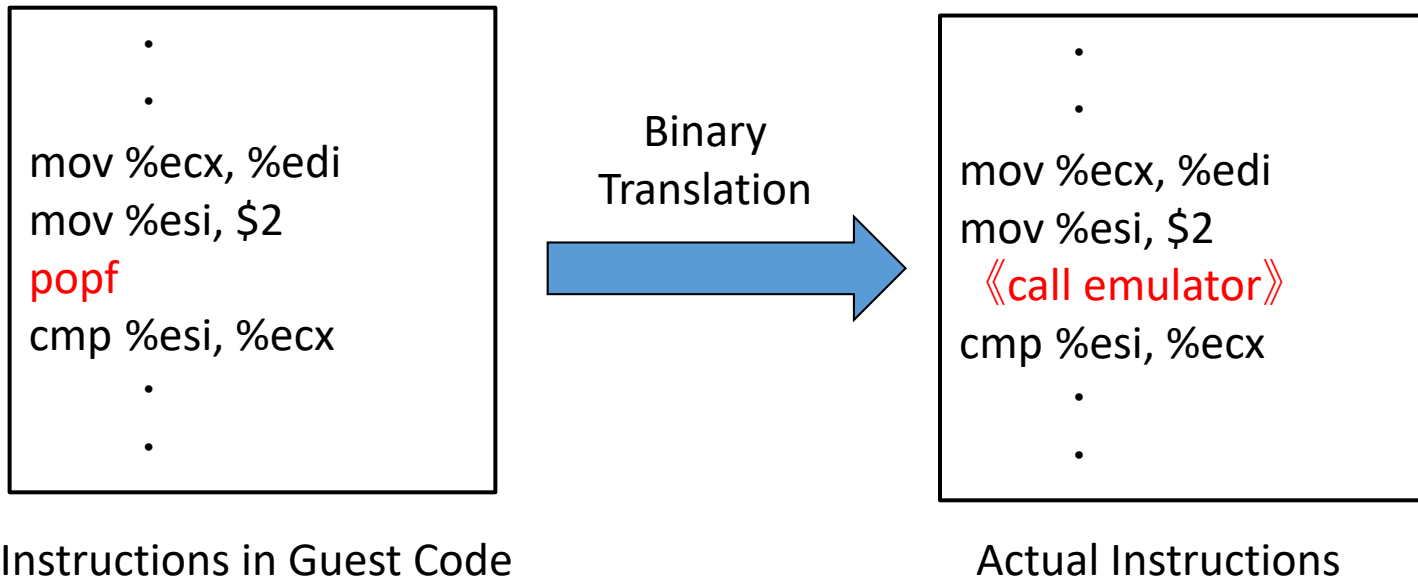
## VMM World



[Sugerman 01] Jeremy Sugerman, Ganesh Venkitachalam and Beng-Hong Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. Proceedings of the 2001 USENIX Annual Technical Conference, pp. 1-14, June 2001.

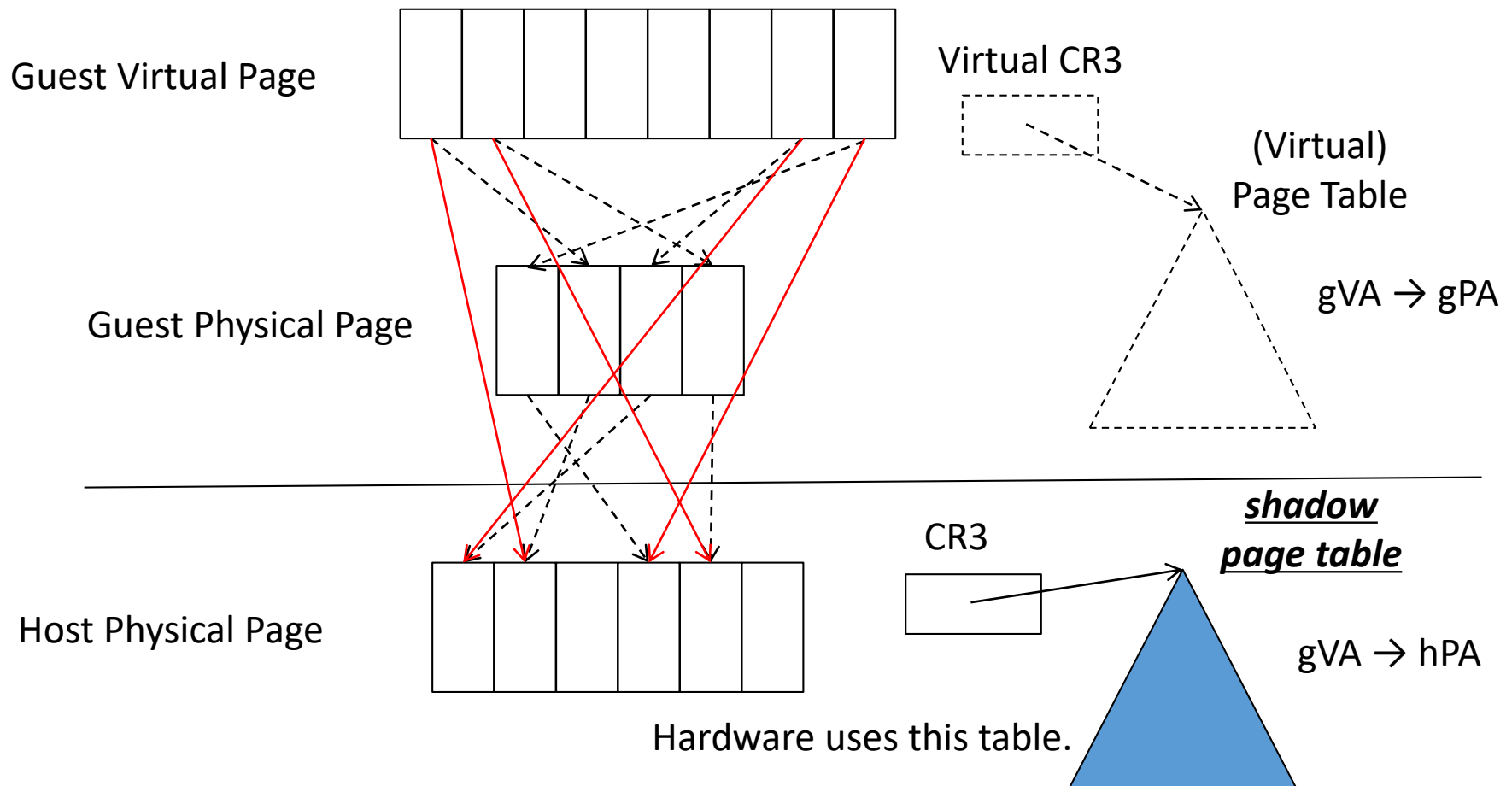
# CPU Virtualization by Binary Translation (BT)

- Innocuous instructions are issued directly.
- Sensitive (either privileged or non-privileged) instructions are emulated.



# Memory Virtualization by Shadow Page Table

- Table for address translation by hardware



# I/O Virtualization

- VMM does not have device drivers and requests host OS to access I/O devices.
  - To reduce the development cost due to multiple kinds of devices.
- Standard PC devices are virtualized.
  - PS/2 keyboard
  - PS/2 mouse
  - Floppy
  - IDE/ATA
  - ATAPI CD-ROM
  - ...

# Outline

- Virtual Machine
- VMWare Workstation
- Xen
- Distributed File Systems

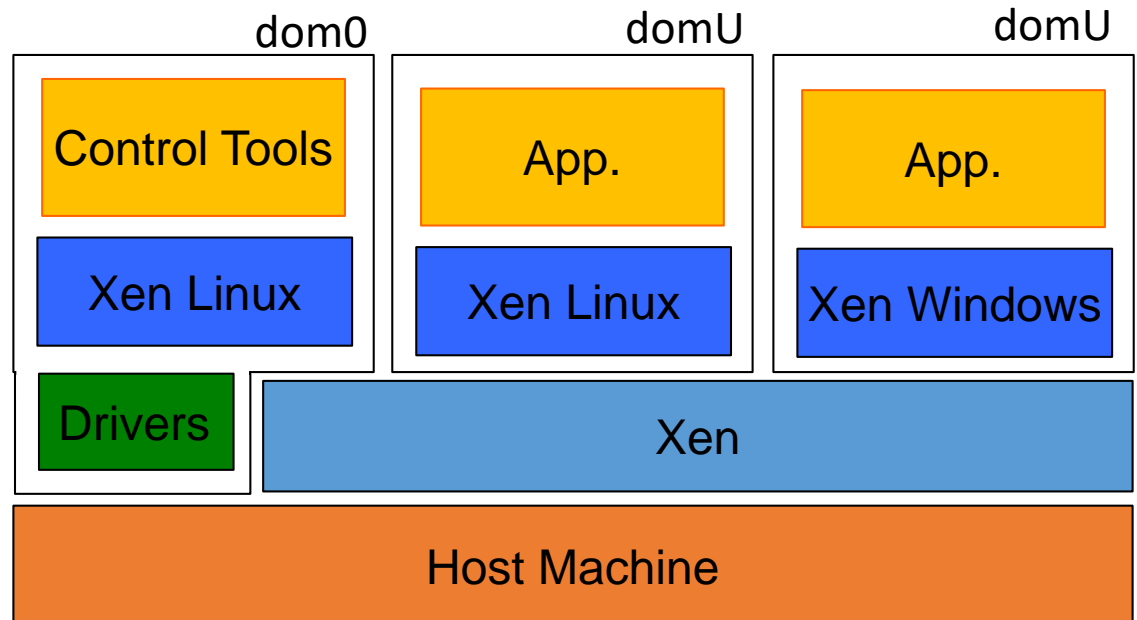
# Xen [Barham 03]

- achieves paravirtualization:
  - modifies guest OS to run on Xen.
  - E.g.: XenoLinux, XenoBSD, XenoXP
  - does not need to modify guest applications.
- is protected from guest OS.
  - Xen can run if guest OS has bugs/malicious programs.
- isolates VMs.
  - If a guest OS have bugs/malicious programs, other guest OSes can run normally.
- Compared to full virtualization (E.g.: VMWare Workstation)
  - The performance is improved in many cases.

[Barham 03] Barham et al. Xen and the art of virtualization. Proceedings of the nineteenth ACM Symposium on Operating Systems Principles, pp. 164-177, October 2003.

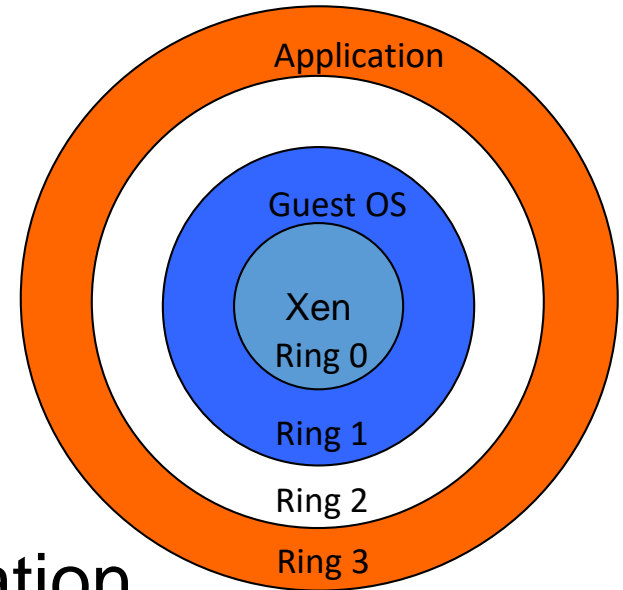
# domain: Xen's VM

- dom0: Privileged domain (driver main)
  - controls other domains.
  - controls physical devices.
  - runs by Xen if Xen starts to run.
- domU: General domain
  - is created by user.



# Protection of Xen

- uses the ring protection of x86.
  - De-privileging
    - Xen runs in ring0.
    - Guest OS runs in ring1.
- Memory protection by segmentation
  - Xen runs in the upper 64MB of memory.
  - Guest OS is modified not to use the memory region of Xen.





# Hypercall

- calls Xen from guest OS.
- is similar with system calls from processes to OS.
  - Low privileged codes call high privileged codes.
- translates sensitive instructions to hypercall.
  - One hypercall processes multiple calls.
  - `mutlicall(void *call_list, int nr_calls);`
- Full virtualization VMM:
  - calls VMM by trap.
  - inserts code to call VMM by binary translation.
- Paravirtualization VMM:
  - calls VMM explicitly by the modification of guest OS.
  - E.g.: if guest OS modifies page table:
    - call `mmu_update();`

# CPU Virtualization

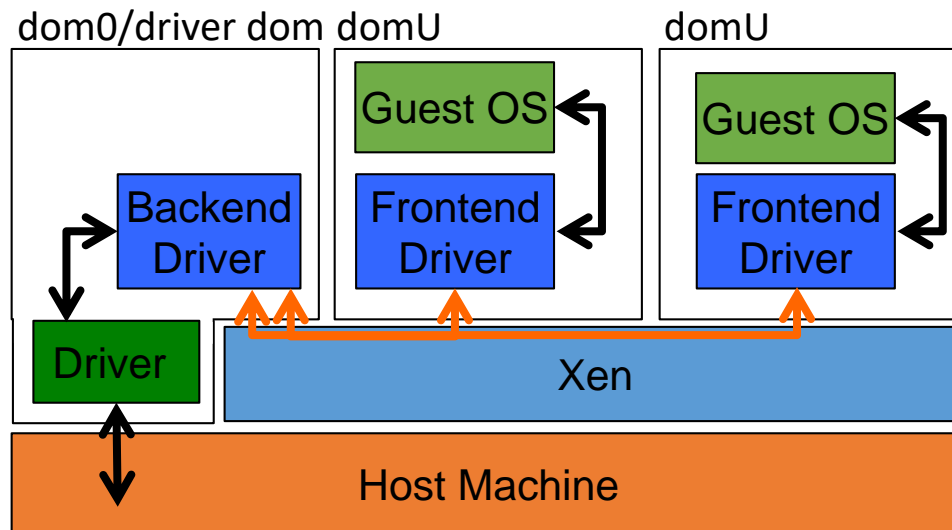
- translates sensitive instructions to hypercall.
  - E.g.: switch page tables
    - update privileged register CR3
    - Since a guest OS cannot update CR3, request Xen to update CR3 by hypercall.
- Shared Info Page
  - shares the status of virtual CPUs
    - Shared memory between guest OS and Xen.
  - shares CPU time and interrupt status.

# Memory Virtualization

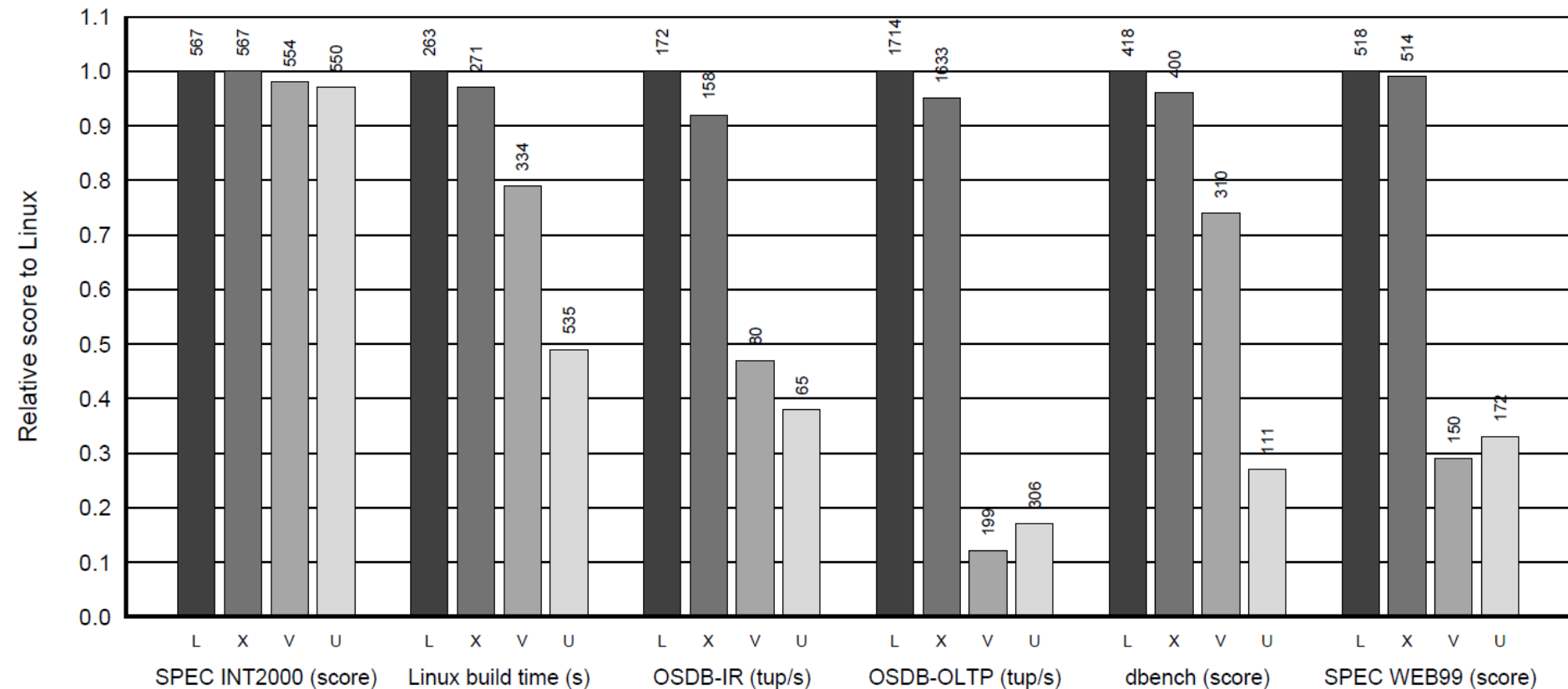
- Basic Approach
  - Minimized support for safety
  - Guest OS manages real page table by itself.
    - manages page table used by host address translation.
    - i.e., manages gVA -> hPA.
- Guest OS
  - makes host page table.
  - allocates physical page for page table.
    - select one of reserved physical pages for specified domain.
  - initializes and registers host page table as page table in Xen.
    - After that, page table is mapped into guest OS read only.
- Xen verifies the update of page table.
  - not to crush Xen or domain.

# I/O Virtualization

- Xen Device Model: Split device driver architecture
  - Paravirtualized device driver
    - Since guest OS is paravirtualized, Xen is more challenge than VMware.
  - Designated I/O domain (dom0), called driver domain, accesses physical devices.
    - Driver domain has drivers to access physical devices.



# Relative Performance of Native Linux (L), XenoLinux (X), VMware Workstation 3.2 (V), and User-Mode Linux (U)



Dell 2650 dual processor 2.4GHz Xeon server with 2GB RAM, a Broadcom Tigon 3 Gigabit Ethernet NIC, and a single Hitachi DK32EJ 146GB 10k RPM SCSI disk, Linux 2.4.21 (i686)

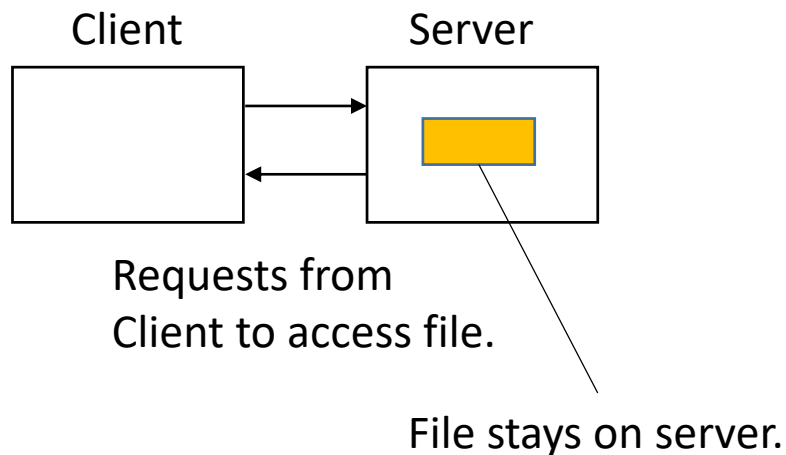
# Outline

- Virtual Machine
- VMWare Workstation
- Xen
- Distributed File Systems

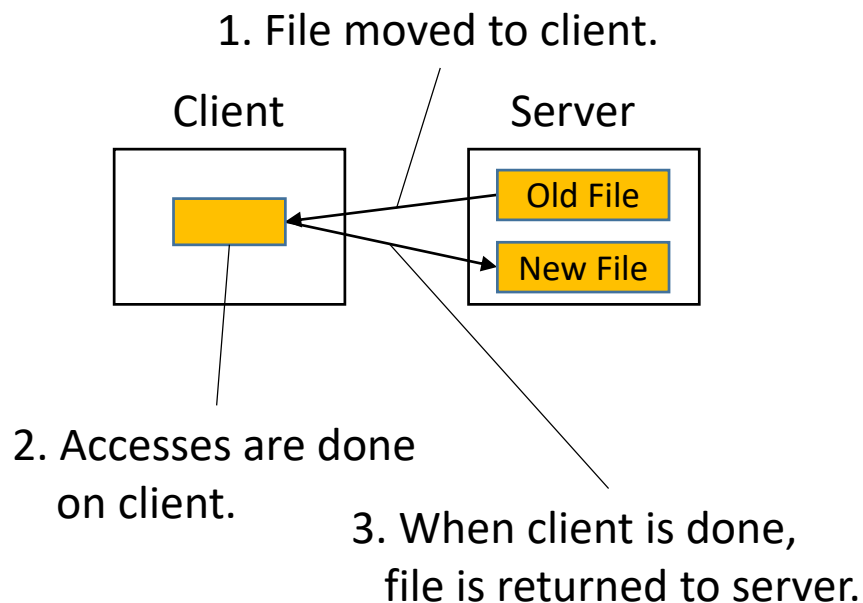
# Why Distributed File Systems?

- share data in distributed systems.
- access data transparently (as if they are local).

## Remote Access Model



## Upload/Download Model



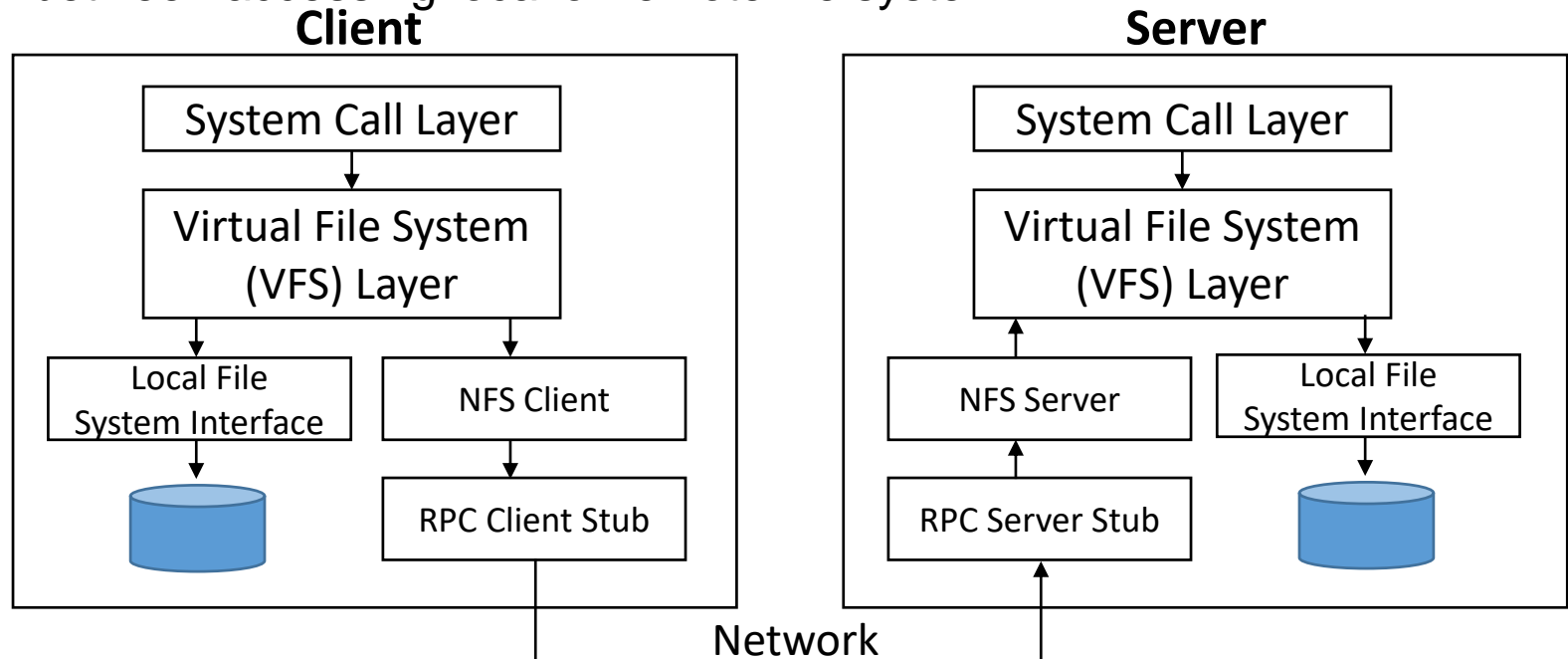
# History of Network File Systems (NFS)

- 19YY: NFSv1 by Sun Microsystems
- 1984: NFSv2
  - 1985: implemented in SunOS 2.0
  - 1989: (defined) in RFC 1094
- 1995: NFSv3 in RFC 1813
- 2000: NFSv4 in RFC 3010
  - 2003: Revised in RFC 3530
  - 2015: Revised again in RFC 7530
- 2010: NFSv4.1 in RFC 5661
- 2016: NFSv4.2 in RFC 7862



# NFS Architecture


- is implemented using the virtual file system.
- is a de facto standard for interfacing to different (distributed) file systems.
- provides standard file system interface and allows to hide difference between accessing local or remote file system.



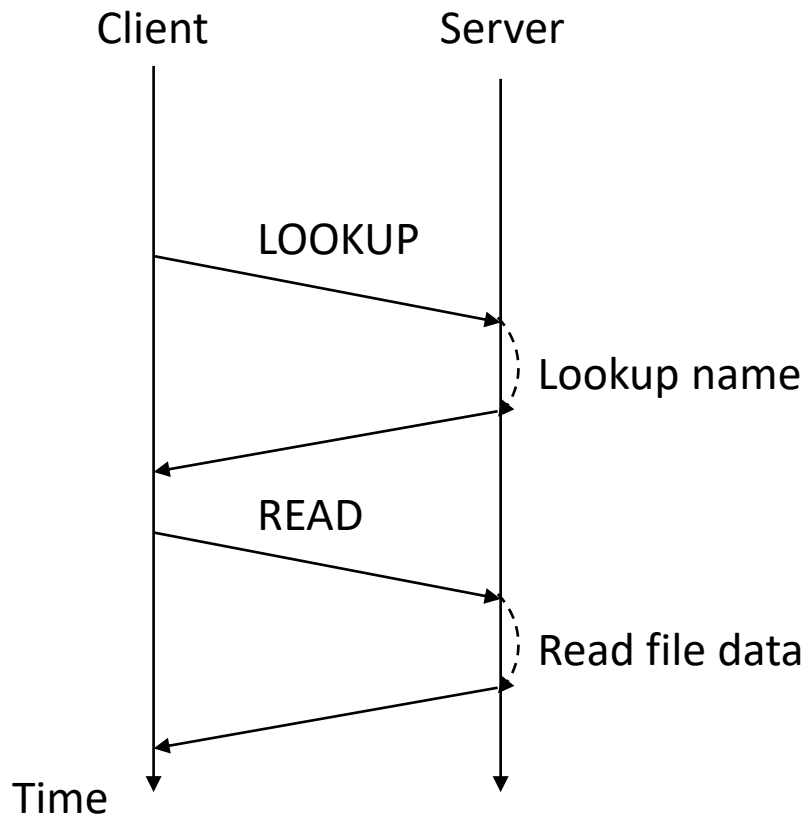
**Basic NFS Architecture for UNIX Systems**

# NFS File Operations

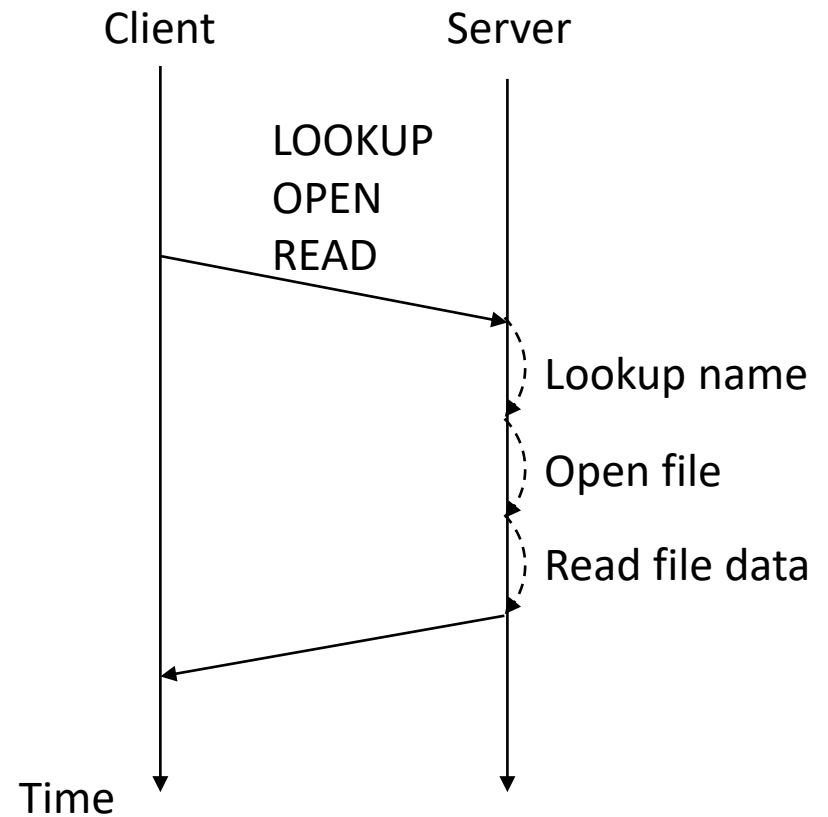
Operation	V3	V4	Description
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory in a given directory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Remove	Yes	Yes	Remove a file from a file system
Rmdir	Yes	No	Remove an empty subdirectory from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Get the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file



# Communication by RPC in NFS



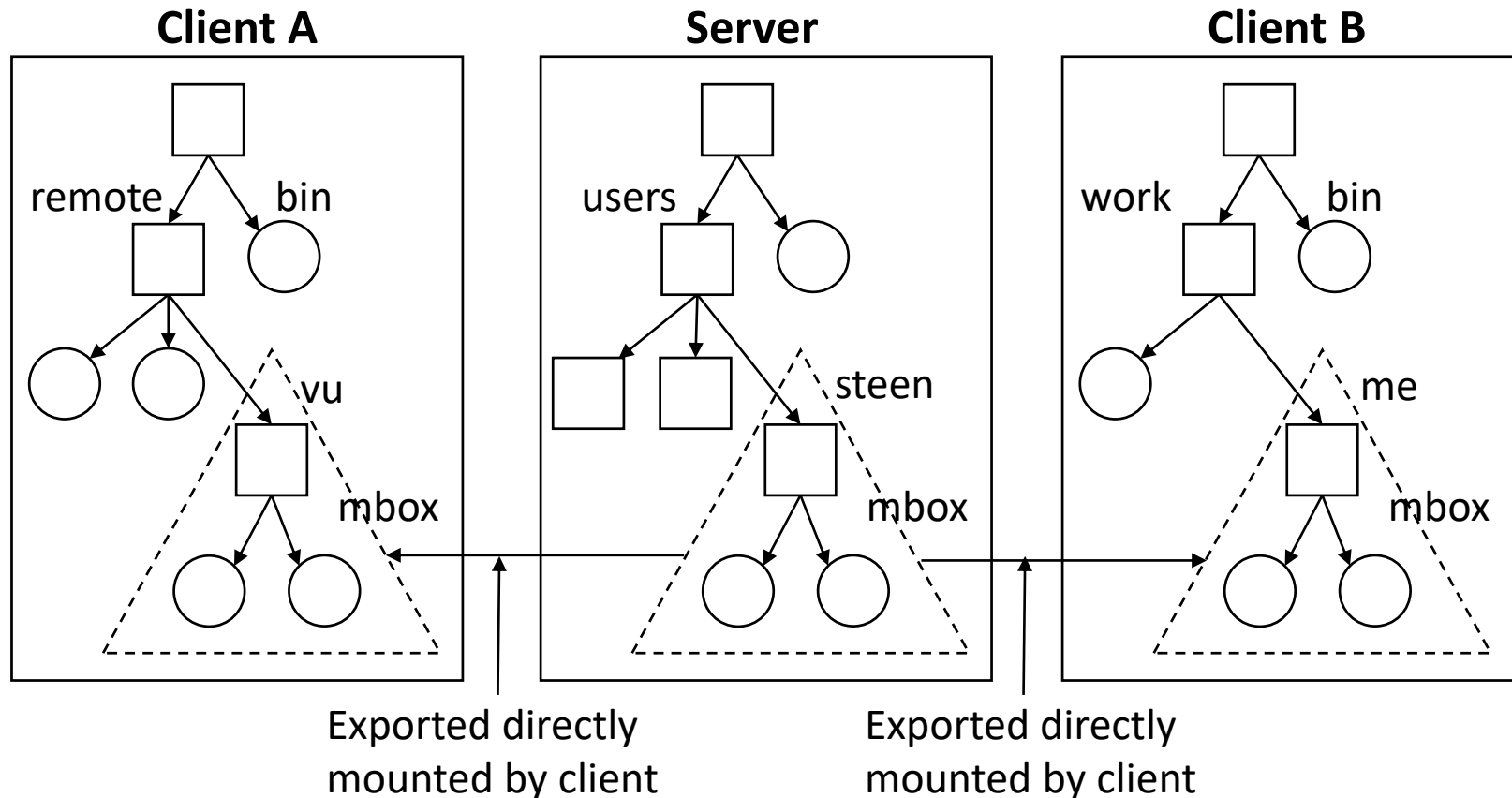
Reading data from a file in NFSv3



Reading data using a compound procedure  
In NFSv4

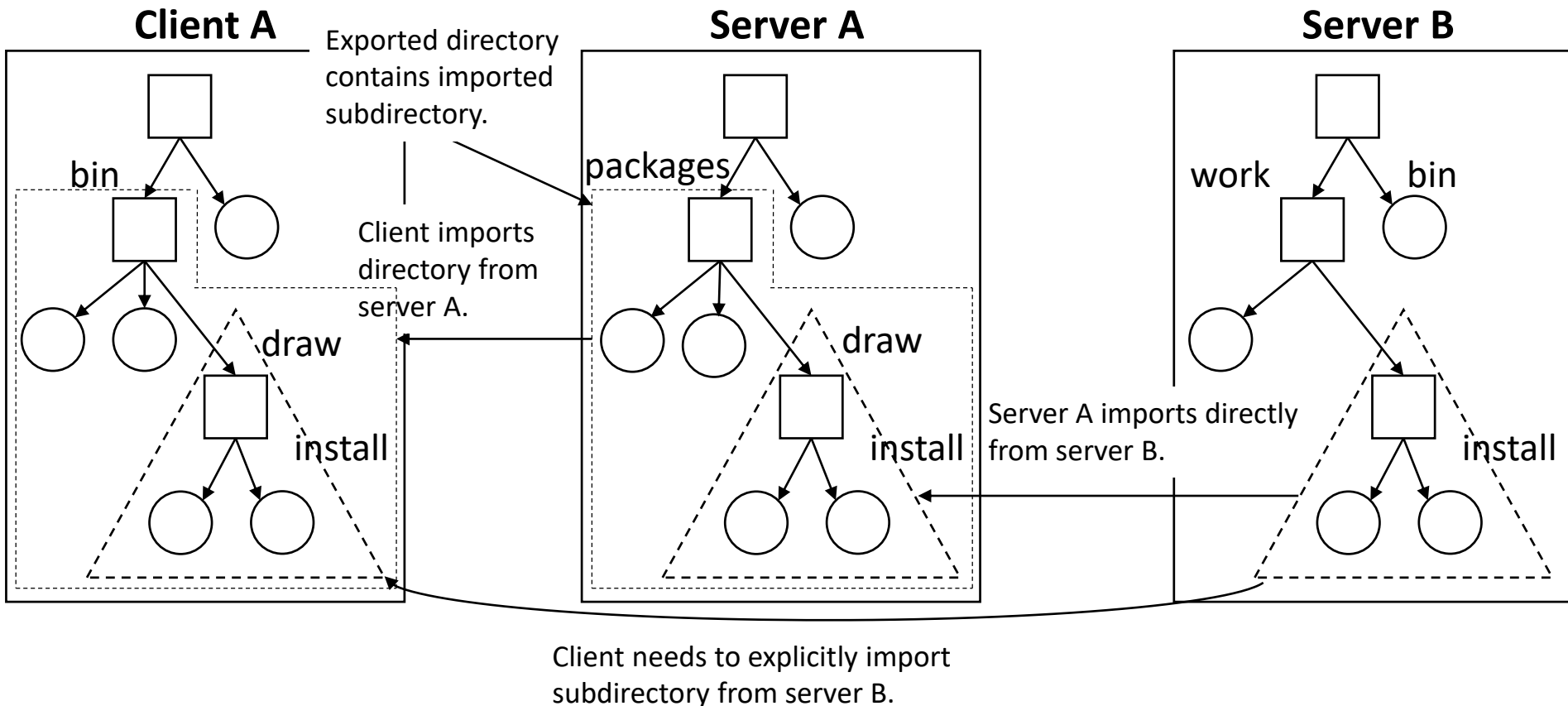
# Mounting (Part of) Remote File System in NFS

- NFS provides clients with the complete transparent access to a remote file system as maintained by a server.



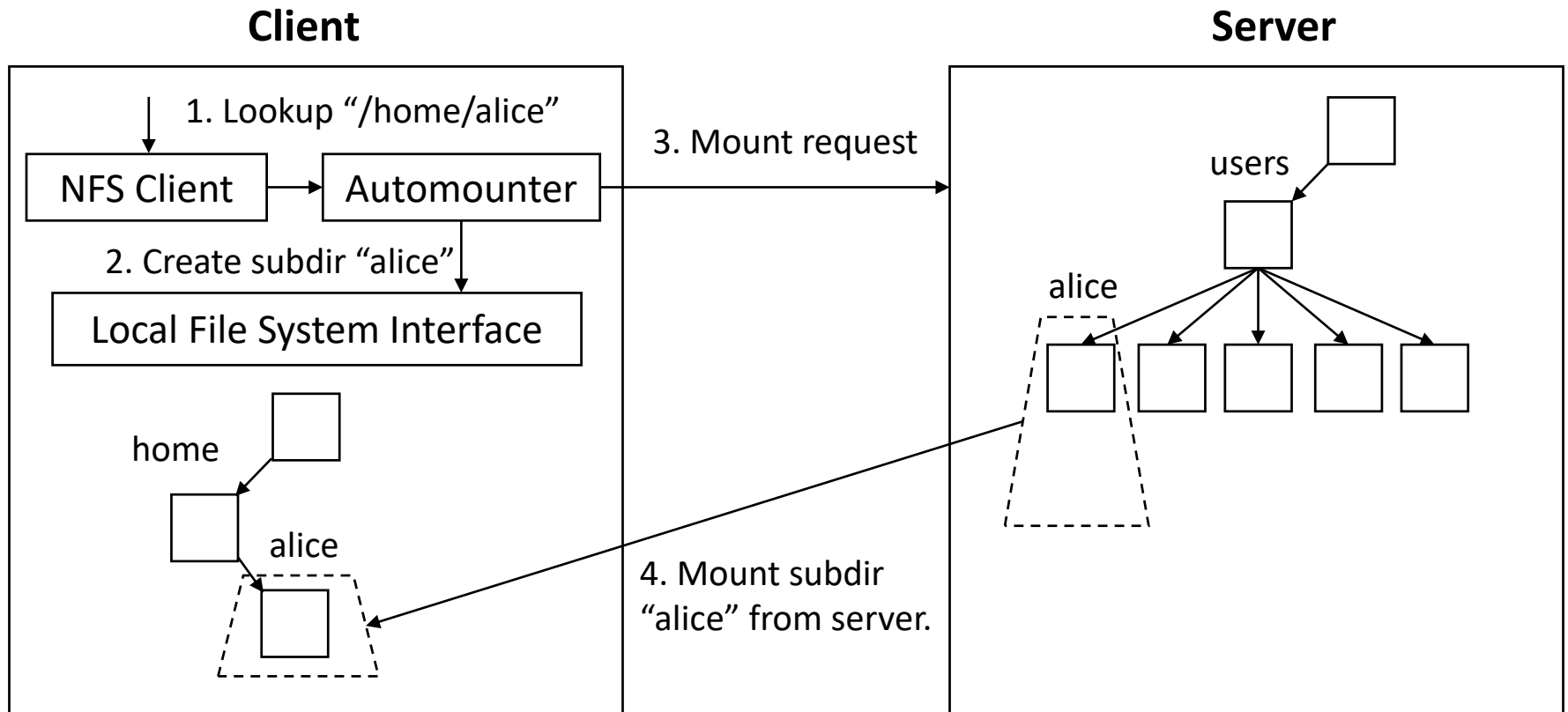
# Mounting Nested Directories from Multiple Servers in NFS

- NFSv3 requires three separate calls to resolve a name such as */bin/draw/install*.
- In NFSv4, a client can pass a complete path name to a server and request the server to resolve it.



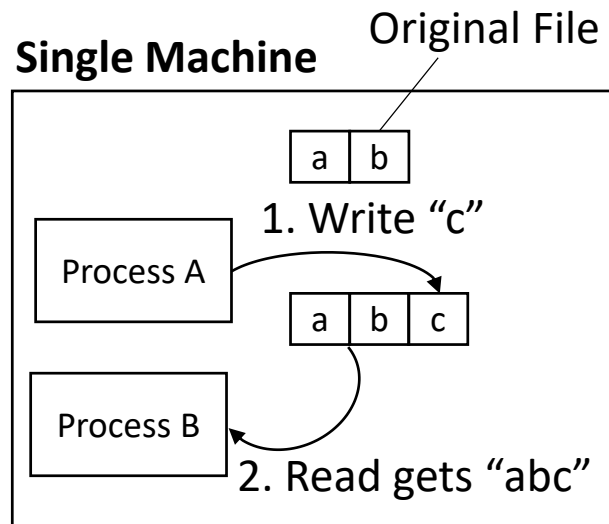
# Automounting

- mounts a remote file system on demand.



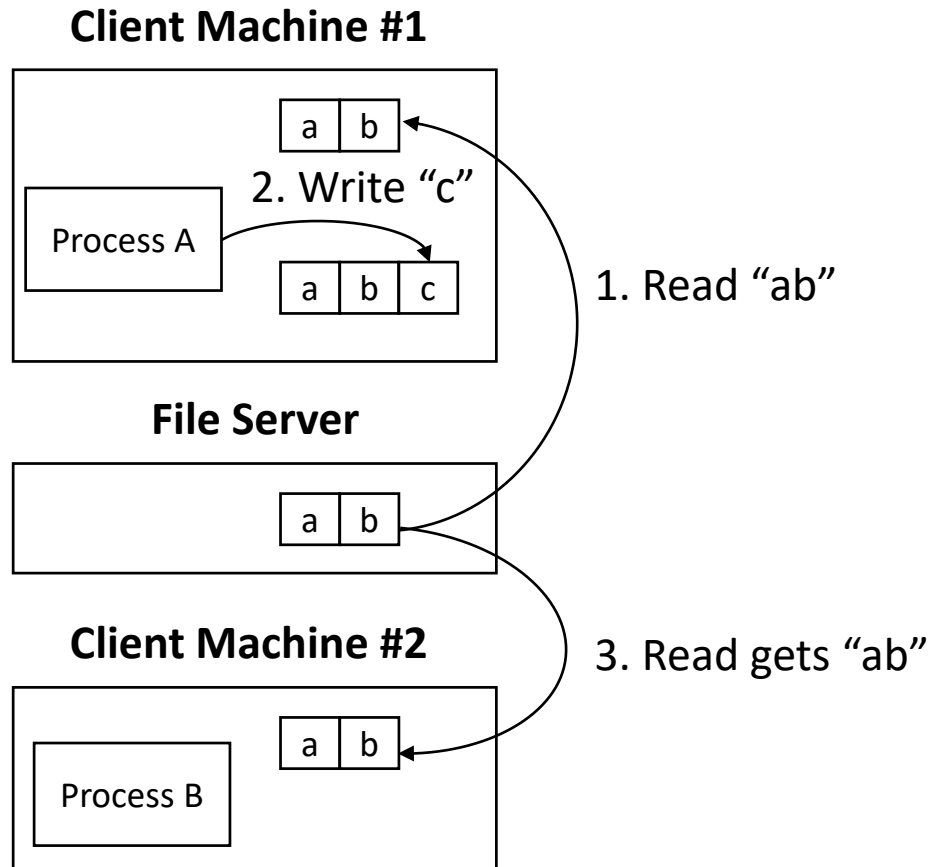
# Semantics of File Sharing (1/2)

- When two or more users share the same file at the same time, it is necessary to define the semantics of reading and writing precisely to avoid problems.



On a single processor, when a *read* follows a *write*, the value returned by the *read* is the value just written.

# Semantics of File Sharing (2/2)



In a distributed system with caching, obsolete (old) values may be returned.



# Four Ways of Dealing with the Shared Files in a Distributed System

Method	Comment
UNIX Semantics	Every operation on a file is instantly visible to all processes
Session Semantics	No changes are visible to other processes until the file is closed
Immutable Files	No updates are possible; simplifies sharing and replication
Transactions	All changes occur atomically

# NFSv4 Operations related to File Locking

Operation	Description
Lock	Create a lock for a range of bytes
Lockt	Test whether a conflicting lock has been granted
Locku	Remove a lock from a range of bytes
Renew	Renew the lease on a specified lock

- NOTE: Locking was not part of NFS until version 3.

# Result of *open* Operation with Shared Reservations in NFS.

- (a) When the client requests shared access given the current file denial state.  
(b) When the client requests a denial state given the current file access state.

(a) Current File Denial State

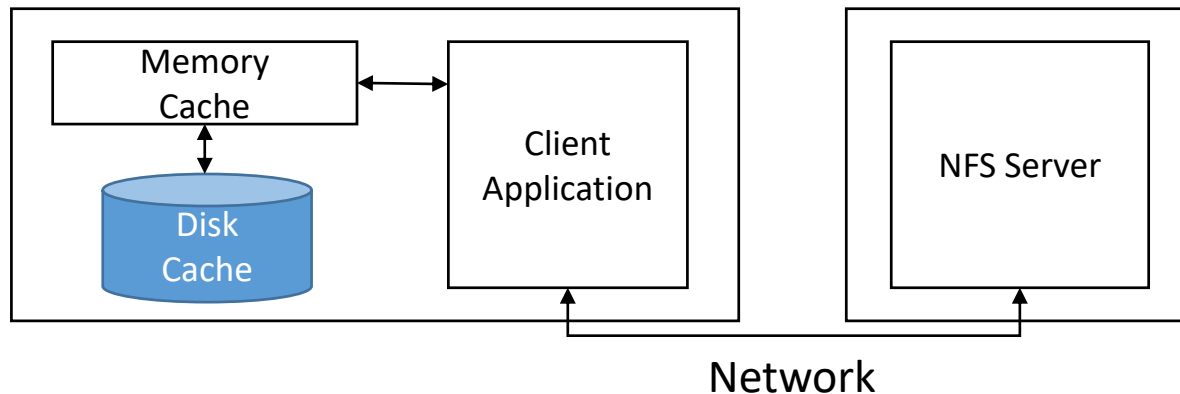
Request Access		NONE	READ	WRITE	BOTH
	READ	Succeed	Fail	Succeed	Fail
	WRITE	Succeed	Succeed	Fail	Fail
	BOTH	Succeed	Fail	Fail	Fail

(b) Requested File Denial State

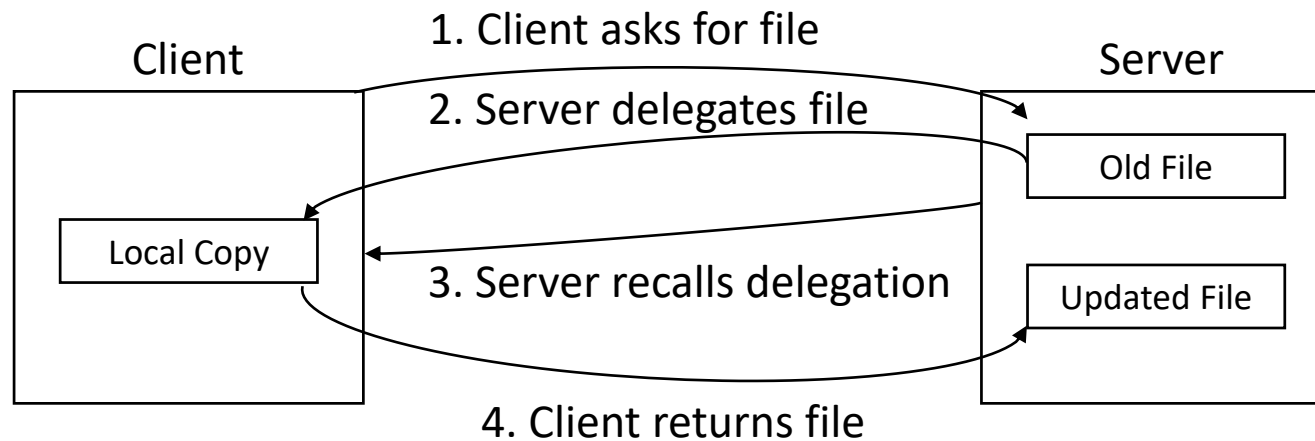
Current Access State		NONE	READ	WRITE	BOTH
	READ	Succeed	Fail	Succeed	Fail
	WRITE	Succeed	Succeed	Fail	Fail
	BOTH	Succeed	Fail	Fail	Fail

# Consistency and Replication

- NFSv3
  - has been mainly left outside of the protocol.
  - does not guarantee consistency.
  - allows cache data to be stale for 30 seconds without the client knowing.
- NFSv4
  - solves some of consistency problems by client-caching in NFS.
    - If modifications have taken place, the cached data must be flushed back to the server.



# Open Delegation by RPC



**Using NFSv4 callback mechanism to recall file delegation**

# Performance Evaluations of NFSv3 and NFSv4

Configuration	Find (m:ss)	Remove (m:ss)
local filesystem	0:01	0:03
NFSv3 noatime,nfsvers=3	9:44	2:36
NFSv3 noatime,nfsvers=3,async	0:31	0:10
NFSv4 noatime	9:52	2:27
NFSv4 noatime,async	0:40	0:08

Extract the linux-2.6.25.4.tar uncompressed Linux kernel source tarball and delete the extracted sources.

No clear performance advantage to moving from NFSv3 to NFSv4.

# Other Distributed File Systems

- Coda File System [Kistler 92]
- Ivy File System [Muthitacharoen 02]
- Google File System [Ghemawat 03]

[Kistler 92] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda File System. ACM Transactions on Computer Systems, Vol. 10, No. 1, pp. 3-25, February 1992.

[Muthitacharoen 02] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A Read/Write Peer-to-Peer File System. In Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation, pp. 31-43, December 2002.

[Ghemawat 03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp. 29-43, October, 2003.