A Survey on Virtualization Technologies

Virtualization is "HOT"

- Microsoft acquires Connectix Corp.
- EMC acquires VMware
- Veritas acquires Ejascent
- IBM, already a pioneer
- Sun working hard on it
- HP picking up

➔Virtualization is HOT!!!

Veritas/Ejascent

- **Veritas Cluster Server**
  - **Integrates the Ejascent's Application Virtualization software**
  - **Enables cluster server users to move data seamlessly across applications without disrupting the transaction state**

Virtualization: What is it, really?

- Real vs. Virtual
  - Similar essence, effect
  - *"Formally"* different
- A framework that combines or divides [computing] resources to present a *transparent* view of one or more environments
  - Hardware/software partitioning (or aggregation)
  - Partial or complete machine simulation
  - Emulation (again, can be partial or complete)
  - Time-sharing (in fact, sharing in general)
  - In general, can be M-to-N mapping (M "real" resources, N "virtual" resources)
  - Examples: VM (M-N), Grid Computing (M-1) , Multitasking (1-N)

Virtualization: Why?

- Server consolidation
- Application Consolidation
- Sandboxing
- Multiple execution environments
- Virtual hardware
- Debugging
- Software migration (Mobility)
- Appliance (software)
- Testing/Quality Assurance

Virtual Machine Implementation: Issues

- Only one "bare" machine interface
- Virtualizable Architecture

  "A virtualizable architecture allows any instruction inspecting/modifying machine state to be trapped when executed in any but the most privileged mode"

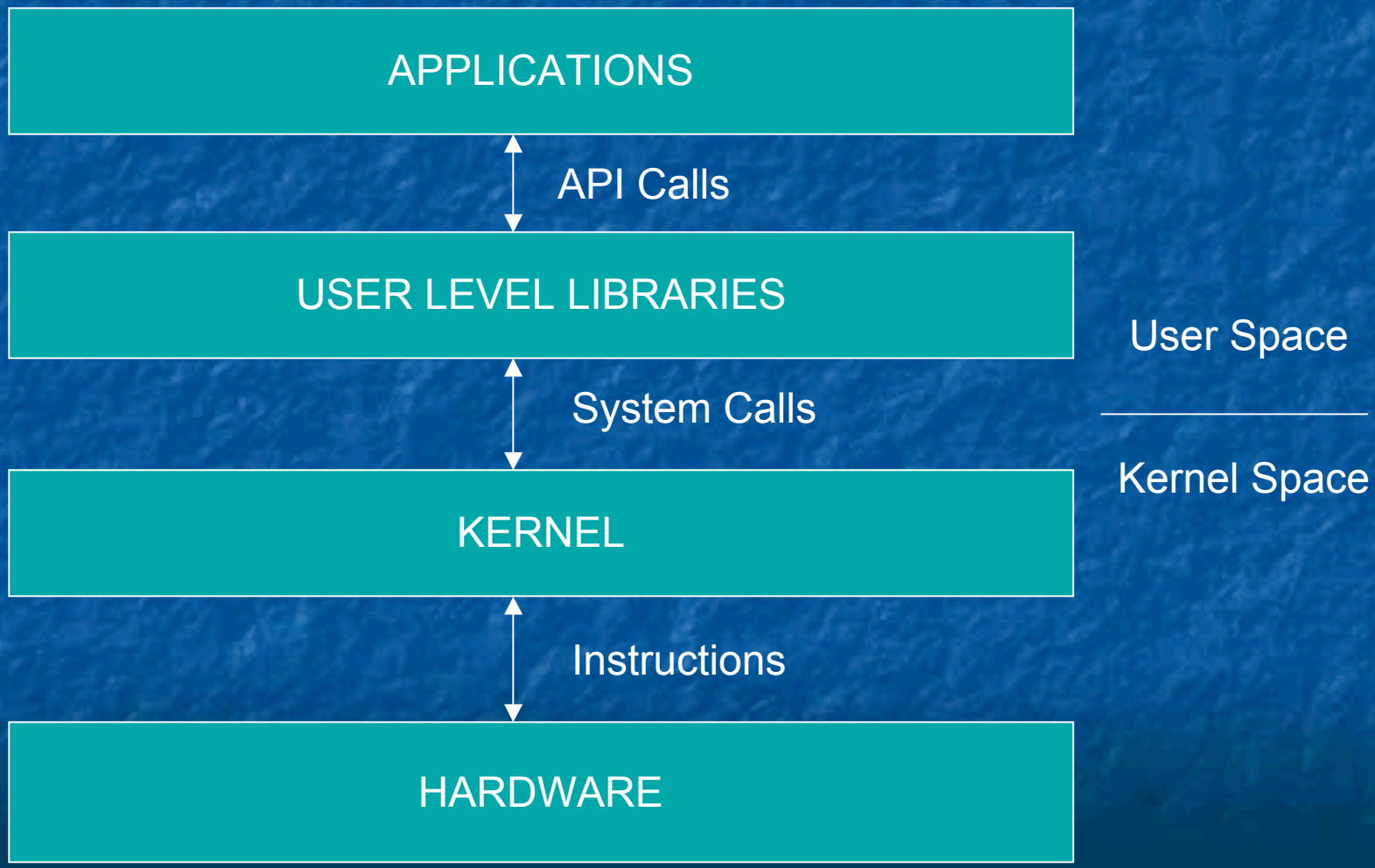
  - Popek & Goldberg (1974)
- X86 is not virtualizable (Vanderpool??)
- Hard to optimize [from below]
  - Unused memory pages
  - Idle CPU
- Difficult to know what NOT to do
  - Example: Page faults (VMM), System Calls (OS level)

Example

- X86 Instruction: STR (gets security state)
  - Value retrieved has the Requester Privilege Level
  - Thus, behavior depends on the privilege level

  ➔Problematic
- X86 has at least 17 such instructions

Machines: Stacked Architecture



Possible Abstraction Levels

- Instruction Set Architecture
  - Emulate the ISA in software
    - Interprets, translates to host ISA (if required)
    - Device abstractions implemented in software
    - Inefficient
  - Optimizations: Caching? Code reorganization?
  - Applications: Debugging, Teaching, multiple OS
- Hardware Abstraction Layer (HAL)
  - Ring Compression
  - Between "real machine" and "emulator" (maps to real hardware)
  - Handling non-virtualizable architectures (scan, insert code?)
  - Applications: Fast and usable, virtual hardware (in above too), consolidation, migration

Possible Abstraction Levels cont'd

- Operating System Level
  - Virtualized SysCall Interface (may be same)
  - May or may not provide all the device abstractions
  - Easy to manipulate (create, configure, destroy)
- Library (user-level API) Level
  - Presents a different subsystem API to application
  - Complex implementation, if kernel API is limited
  - User-level device drivers
- Application (Programming Language) Level
  - Virtual architecture (ISA, registers, memory, …)
  - Platform-independence (→ highly portable)
  - Less control on the system (extremely high-level)

Overall Picture

	ISA	HAL	OS	Library	PL
Performance	*	****	****	***	**
Flexibility	****	***	**	**	**
Ease of Impl	**	*	***	**	**
Degree of Isolation	***	****	**	**	***

(more stars are better)

Instruction Set Architecture Level Virtualization

- Technologies
  - Emulation: Translates guest ISA to native ISA
  - Emulates h/w specific IN/OUT instructions to mimic a device
  - Translation Cache: Optimizes emulation by making use of similar recent instructions
  - Code rearrangement
  - Speculative scheduling (alias hardware)
- Issues
  - Efficient Exception handling
  - Self-modifying code

ISA Level Virtualization: Examples

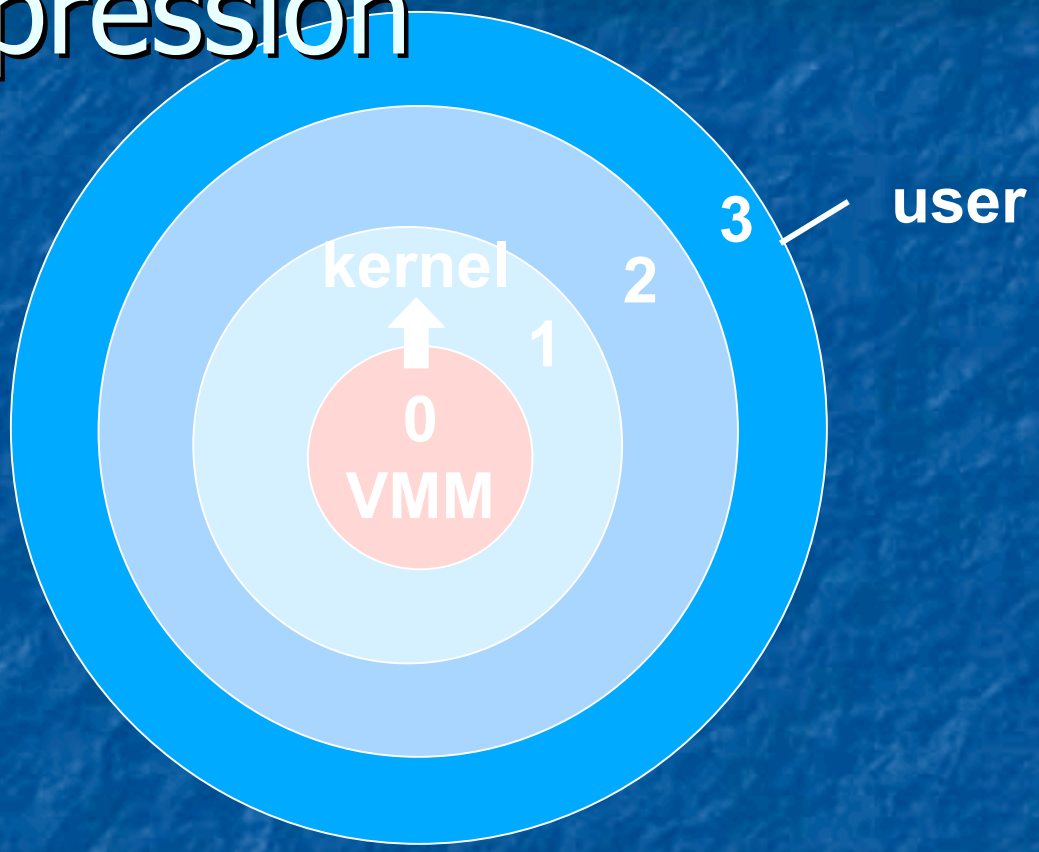
- Bochs: Open source x86 emulator
  - Emulates whole PC environment
    - x86 processor and most of the hardware (VGA, disk, keyboard, mouse, …)
    - Custom BIOS, emulation of power-up, reboot
    - Host ISAs: x86, PowerPC, Alpha, Sun, and MIPS
- Crusoe (Transmeta)
  - "Code morphing engine" – dynamic x86 emulator on VLIW processor
  - 16 MB "translation cache"
  - Shadow registers: Enables easy exception handling
- QEMU:
  - Full Implementation
    - Multiple target ISAs: x86, ARM, PowerPC, Sparc
    - Supports self-modifying code
    - Full-software and simulated (using mmap()) MMU
  - User-space only: Useful for Cross-compilation and cross-debugging

Virtualization through Ring Compression

Virtual Machine Monitor (VMM) runs at ring 0

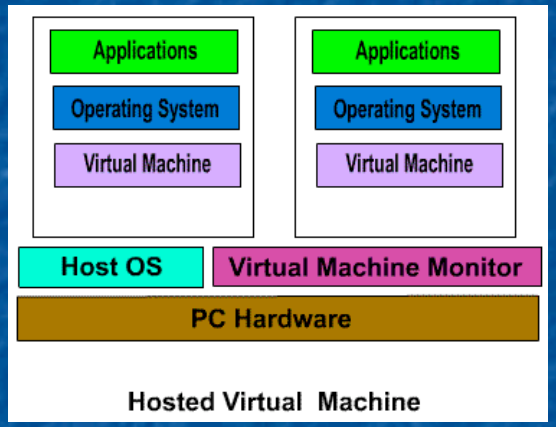
Kernel(s) run at ring 1

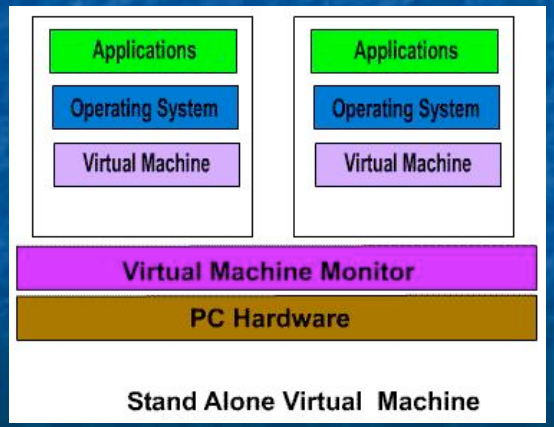
Requires that CPU is virtualizable



HAL Virtualization Techniques

- Standalone vs. Hosted
  - Drivers
  - Host and VMM worlds
  - I/O
- Protection Rings
  - Multilevel privilege domains
- Handling "silent" fails
  - Scan code and insert/replace artificial traps
  - Cache results to optimize

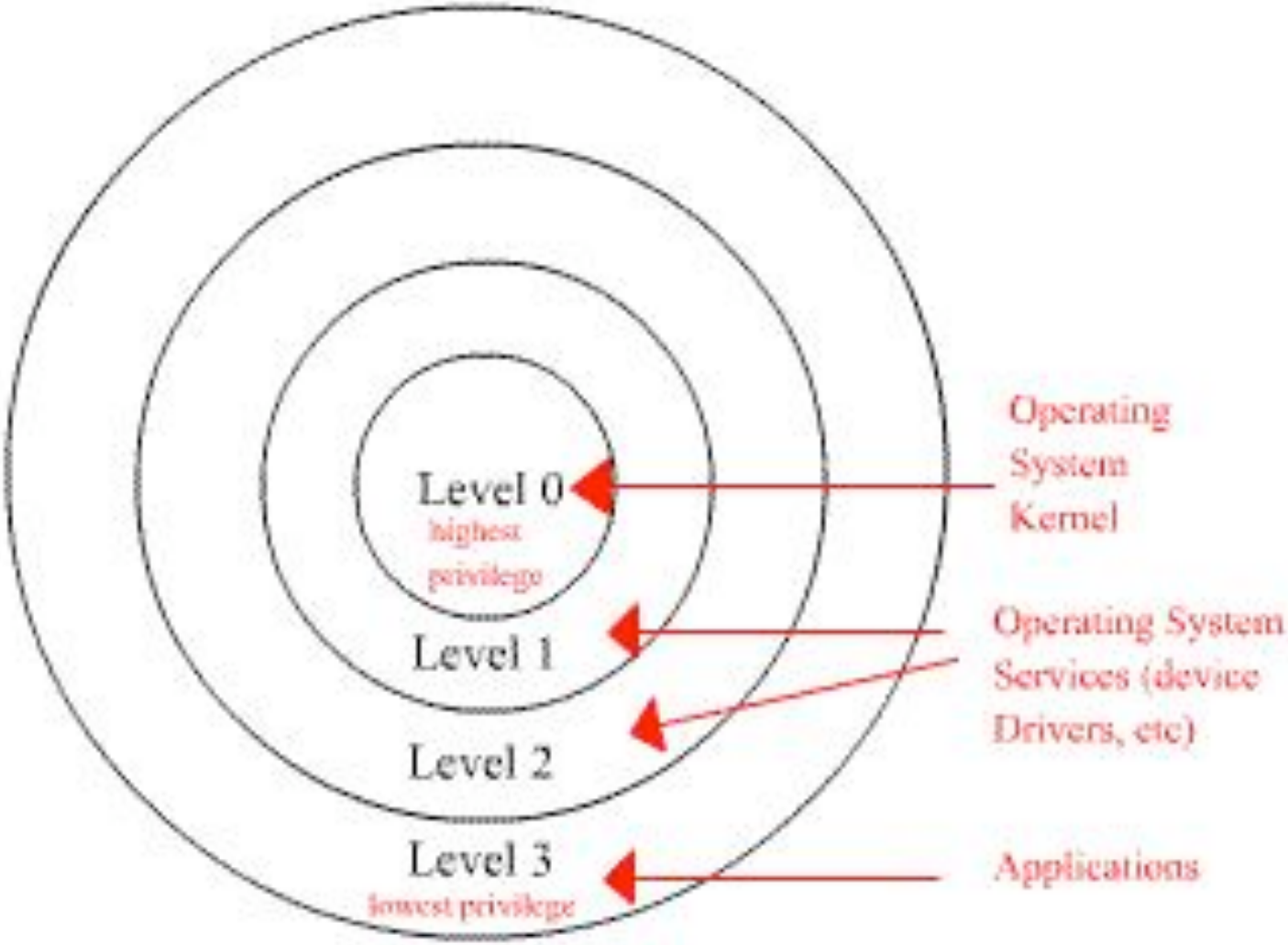




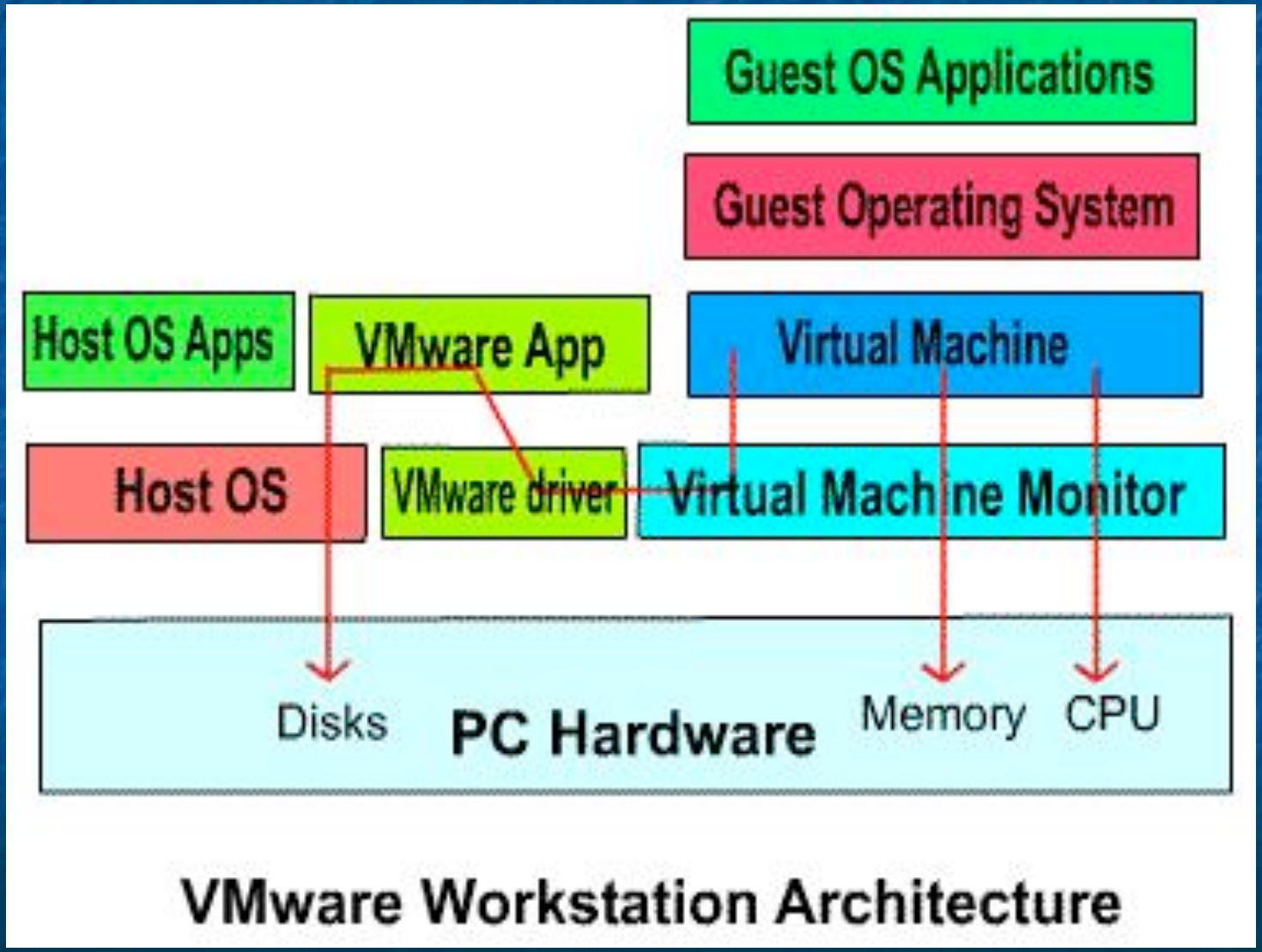
Classification of processor architectures

- **Strictly virtualizable** processor architectures
  - Can build a VMM based on trap emulation exclusively
    - No software running inside the VM cannot determine the presence of the VMM (short of ~~timing attacks~~)
  - Examples: IBM S/390, DEC Compaq Intel Alpha, PowerPC
- **(Non-strictly) virtualizable** processor architectures
  - Trap emulation alone is not sufficient and/or not complete
    - E.g. instructions have different semantics at various levels (sufficient)
    - E.g Some software sequences can determine the presence of the VMM (complete)
  - Examples: IA-32, IA-64
- Non virtualizable processor architectures
  - Basic component missing (e.g. MMU, …)

Intel IA32 Protection Rings

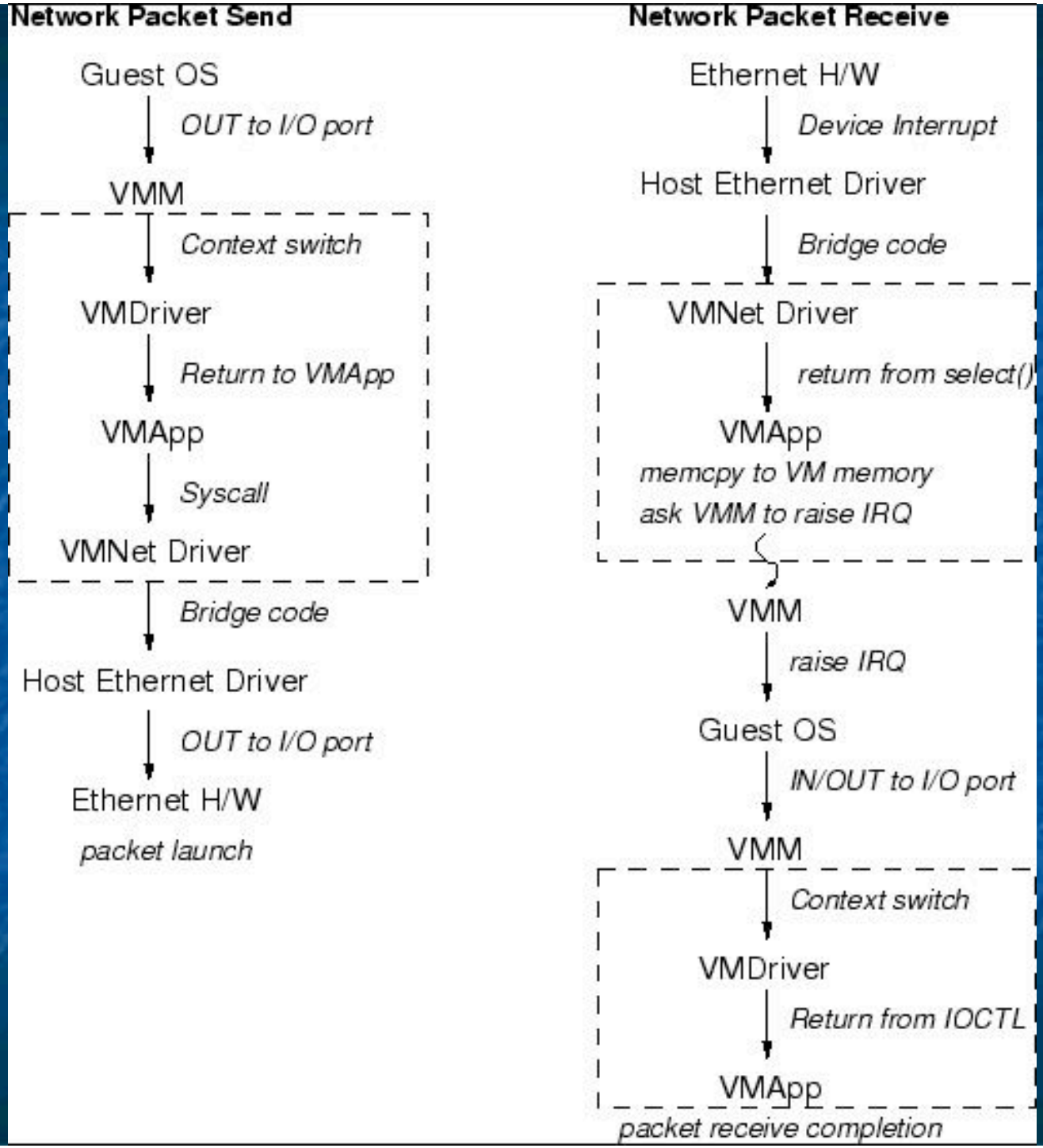


VMware Architecture



VMware: I/O Virtualization

- VMM does not have access to I/O
- I/O in "host world"
  - Low level I/O instructions (issued by guest OS) are merged to high-level I/O system calls
  - VM Application executes I/O SysCalls
- VM Driver works as the communication link between VMM and VM Application
- World switch needs to "save" and "restore" machine state
- Additional techniques to increase efficiency


Network Packet Send
Guest OS
OUT to I/O port
VMM
Context switch
VMDriver
Return to VMApp
VMApp
Syscall
VMNet Driver
Bridge code
Host Ethernet Driver
OUT to I/O port
Ethernet H/W
packet launch
Network Packet Receive
Ethernet H/W
Device Interrupt
Host Ethernet Driver
Bridge code
VMNet Driver
return from select()
VMApp
memcpy to VM memory
ask VMM to raise IRQ
VMM
raise IRQ
Guest OS
IN/OUT to I/O port
VMM
Context switch
VMDriver
Return from IOCTL
VMApp
packet receive completion