Virtual Machine

How to virtualize the entire machine?

By Matthew Tancreti
For COM S 352
Iowa State University

Virtual Machine

We have seen virtualization of processing and memory

What is achieved by virtualizing entire machine?

Applications may require different OS environments, therefore desire to have multiple OSes on single machine

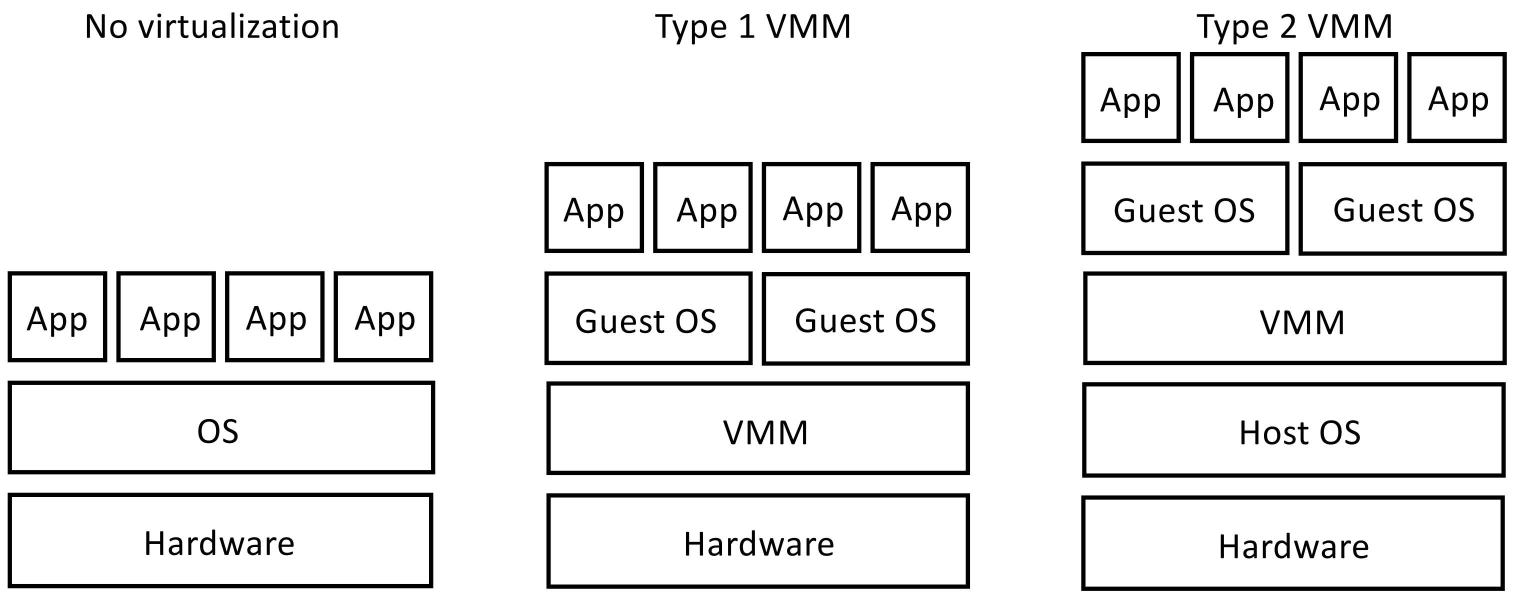
How to virtualize a machine?

The Treachery of Images, René Magritte [source]

Virtual Machine Monitor

Virtual Machine Monitor (**VMM**) (also called **hypervisor**) creates illusion of being hardware to a OS above it

Don't want to modify the OS to run on VMM, **transparency** is major goal of VMM



Motivations for VMM

Consolidate multiple OSes onto fewer hardware platforms, lowering cost and ease administration

Accesses to applications that only run on a different platform, e.g., run Linux application on Windows host

Isolated and reproducable environment for testing and debugging

Limited Direct Execution

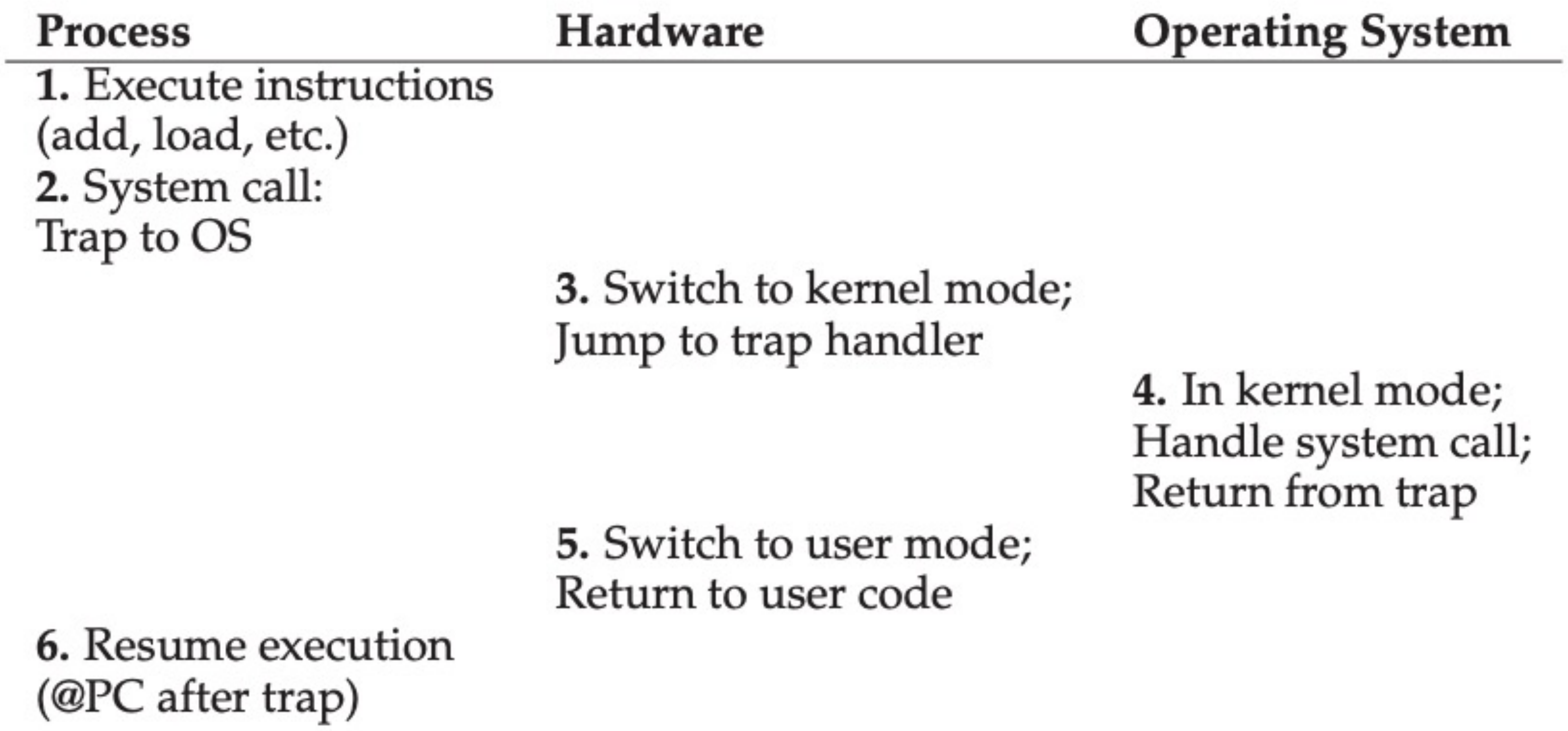
OS achieves virtualization through limited direct execution, e.g., context switching between processes

VMM also operates on limited direct execution, need to context switch (or “machine switch”) between virtual machines

To achieve “machine switch”, VMM must save entire machine state of one OS and restore state for state being switched into

A problem is that OS normally operates in kernel mode allowing it to execute all privileged instructions

Standard System Call



Process	Hardware	Operating System
1. Execute instructions (add, load, etc.)		
2. System call: Trap to OS		
	3. Switch to kernel mode; Jump to trap handler	
		4. In kernel mode; Handle system call; Return from trap
	5. Switch to user mode; Return to user code	
6. Resume execution (@PC after trap)		

Trap Handler

Standard system call:

application in user mode executes privileged instruction

which causes trap

which switches processor to kernel mode

In standard configuration, it is safe for trap to execute in kernel mode because the **OS installed the trap handlers at boot time**

The difference on virtual machine is the **VMM installed the trap handlers**

How does VMM know what to do with trap?

When the guest OS boots it tries to install trap handlers by writing them into specified memory locations

VMM know where in memory the guest OS trap handers are located

When VMM trap handler executes it calls the guest OS trap handler

When guest OS trap handler returns from trap, it returns into the VMM trap handler which performs the actual return-from-trap into the application

Reduced Privilege

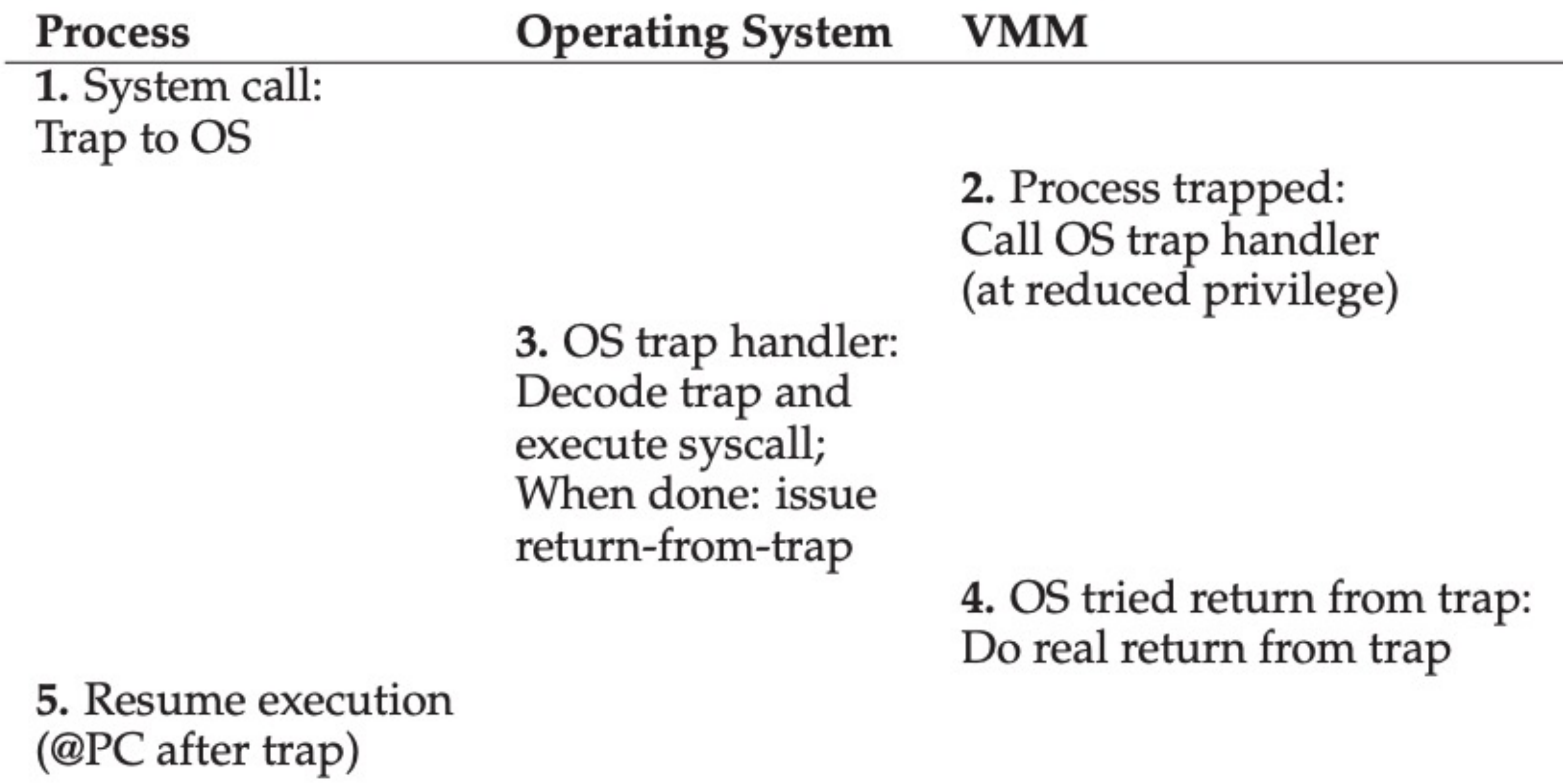
But what stops the guest OS trap handler from executing instructions that overwrite the VMM instructions?

Before calling guest OS trap handler, VMM reduces the privilege mode (some processors have an additional **supervisor mode**)

When OS executes privileged instruction, a trap occurs giving VMM control

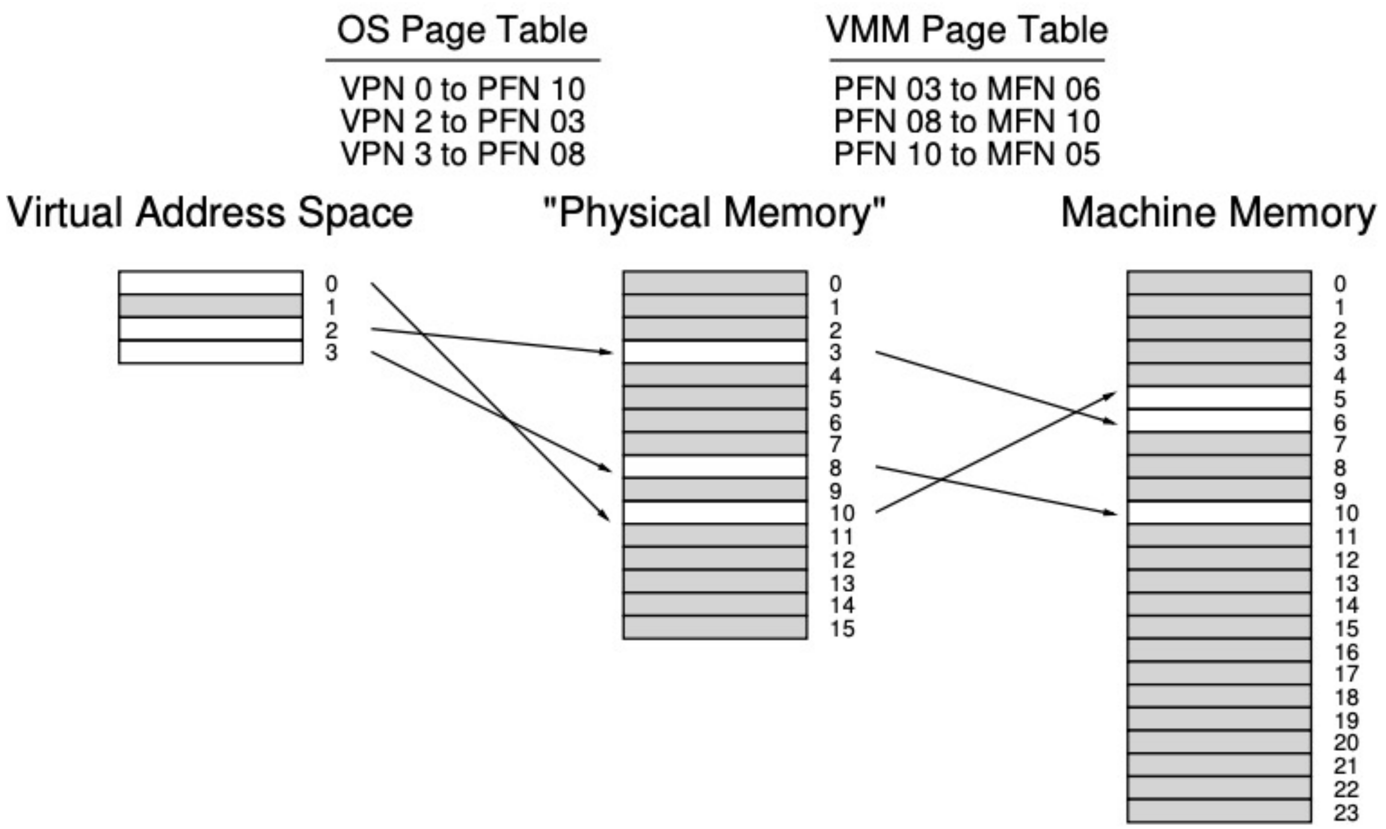
Guest OS executing return-from-trap is also a privileged instruction, which is how control returns to VMM trap handler

System Call with VMM



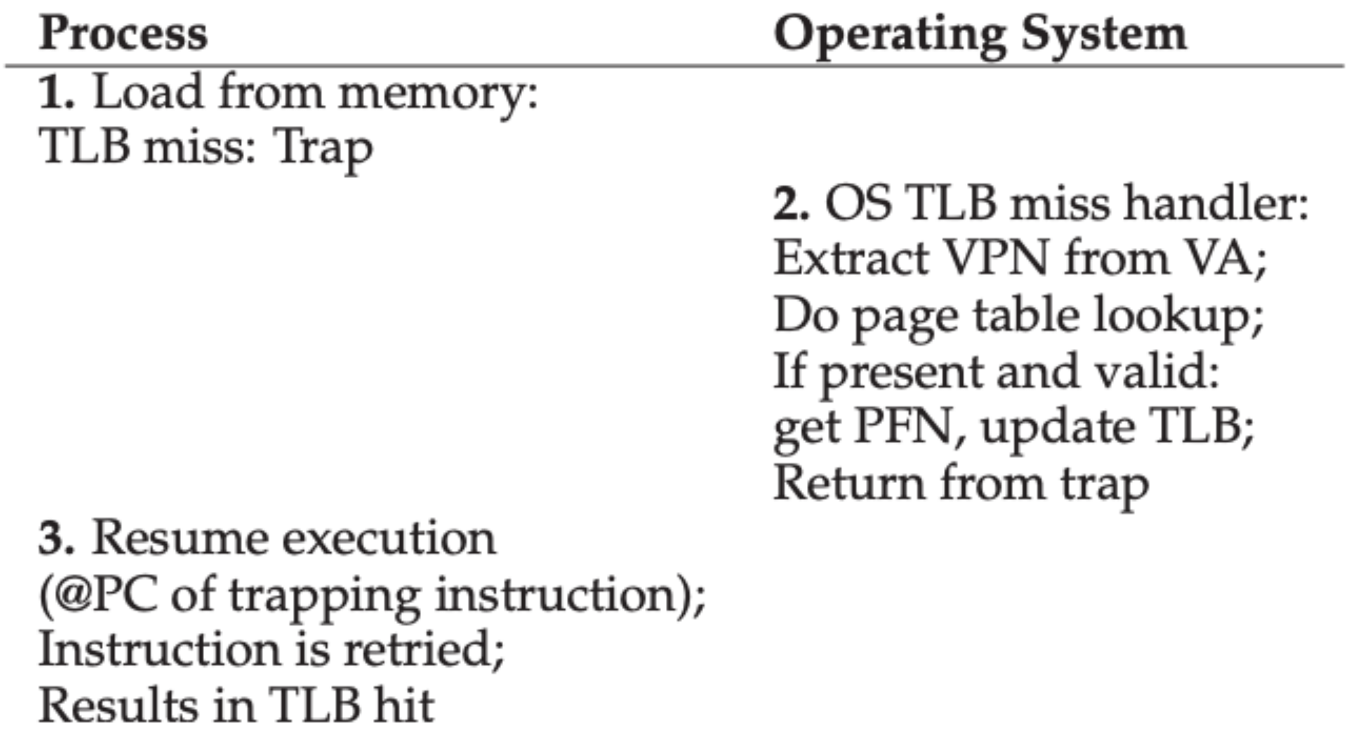
Process	Operating System	VMM
1. System call: Trap to OS		
		2. Process trapped: Call OS trap handler (at reduced privilege)
	3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap	
		4. OS tried return from trap: Do real return from trap
5. Resume execution (@PC after trap)		

Memory Virtualization Virtualization



Standard Address Translation Flow on TLB Miss

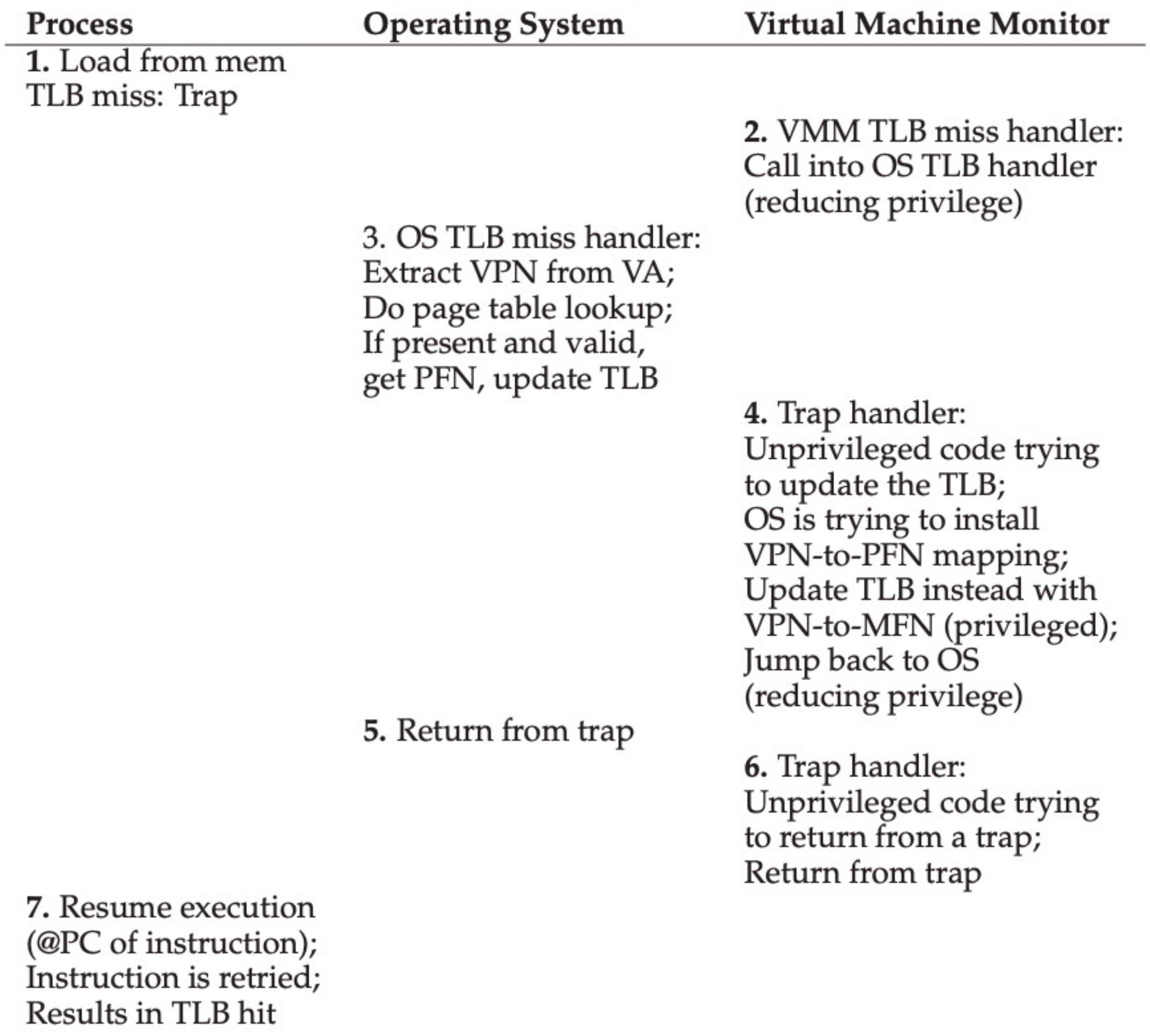
Assume software managed page tables, i.e., in memory and managed by the OS
VA = Virtual Address



Process	Operating System
1. Load from memory: TLB miss: Trap	
	2. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid: get PFN, update TLB; Return from trap

3. Resume execution
(@PC of trapping instruction);
Instruction is retried;
Results in TLB hit	

TLB Miss with VMM



Process	Operating System	Virtual Machine Monitor
1. Load from mem TLB miss: Trap		
		2. VMM TLB miss handler: Call into OS TLB handler (reducing privilege)
	3. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid, get PFN, update TLB	
		4. Trap handler: Unprivileged code trying to update the TLB; OS is trying to install VPN-to-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)
	5. Return from trap	
		6. Trap handler: Unprivileged code trying to return from a trap; Return from trap

7. Resume execution
(@PC of instruction);
Instruction is retried;
Results in TLB hit		

Information Gap

Because of transparency, VMM doesn’t know what guest OS is trying to achieve

There is an **information gap** between OS and VMM that can lead to significant inefficiency

For example, when OS has no useful work (i.e., no runnable processes) it will spin in its scheduler loop

```
while (1)
  ; // the idle loop
```

Potential solution breaks transparency, in **para-virtualization** guest OS has small modifications to operate more effectively in virtualized environment