VM

Starts from IBM VM/370
definition: "an efficient, isolated duplicate of a real machine"
Old purpose: share the resource of the expensive mainframe machine
New purpose: fault tolerance and security; run multiple different OSes on the same desktop; server consolidation and performance isolation, etc.

type 1 vmm: vmm running on bare hardware
type 2 vmm: vmm running on host os

Virtualization tasks

(isolate, secure, fast, transparent …)

1. cpu virtualization

How to make sure privileged instructions (e.g., enable/disable interrupts; change critical registers) will not be executed directly

2. memory virtualization

OS thought they can control the whole physical memory. No!

3. I/O virtualization

OS thought I/O devices are all under its own control. No!

Problems of X86

1. Some privileged instructions do not trap; they just fail/change-semantic silently.
2. TLB miss will be handled by hardware instead of software

Categories of X86 virtualization

Full-virtualization (VMWare)

No change to guest OS
Use binary translation to change some instructions (privilege instructions or all OS code) on the fly

Para-virtualization (Xen)

Change guest OS to improve performance
Rewrite OS; replace privileged instructions with hypercalls
Many other changes

Disco

On MIPS
DISCO directly runs on multi-processor machine; Oses run upon DISCO
Goal: use commodity OSes to leverage many-core cluster

VMM runs in kernel mode
VM OS runs in supervisor mode
VM user runs in user mode
Privileged instructions will trap to VMM and be emulated

CPU virtualization
No special challenge
(seemed that all privilege instructions will trap)
VMM maintains a process-table-entry-like data structure for each VM (virtual PC) (including registers, states, and TLB content (used for emulation …)).

Memory virtualization

virtual address --> physical address (the one in guest OS page table) --> machine address (the "real" address pointing to physical memory location)
tlb: virtual --> machine
tlb miss: interrupt to VMM
VMM --> guest OS: guest OS goes through page table, finds out the physical page number, tries to fill TLB
(TLB-filling is a privileged instruction; guest OS executing it from less-privileged level will lead to interrupt)
trap to VMM: VMM takes the physical page number, looks up "pmap", find the matching machine page number, get the <virtual-page-number, machine-page-number> entry, fill the TLB

- pmap maps physical page numbers to machine page numbers, one per virtual machine

- TLB is flushed at every context switch, because …

- to speed up, VMM keeps software TLB, so that it does not need to ask guest OS for physical page number every time during TLB miss.

The memory virtualization challenge on x86:
hardware handles TLB misses and looks up page table.
so, there has to be a page table that contains virtual-to-machine translation for hardware to look up
solution:
1. keep a shadow page table that keeps virtual-to-machien translation; keep the OS page table read-only, so that every OS updates to page table will trap to VMM and VMM can update shadow page table (VMware solution)
2. modify OS so that OS does not directly update its page table; instead, OS calls hyper-call and asks Xen to update page table (Xen's para-vritualization solution)