Virtual Machines Disco and Xen (Lecture 10, cs262a)

Ion Stoica,
UC Berkeley
September 28, 2016

Today's Papers

Disco: Running Commodity Operating Systems on Scalable Multiprocessors, Edouard Bugnion, Scott Devine, and Mendel Rosenblum, SOSP'97 (https://amplab.github.io/cs262a-fall2016/notes/Disco.pdf)

Xen and the Art of Virtualization, P. Barham, B. Dragovic, K Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, SOSP'03 (www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf)

What is Virtual Machine?

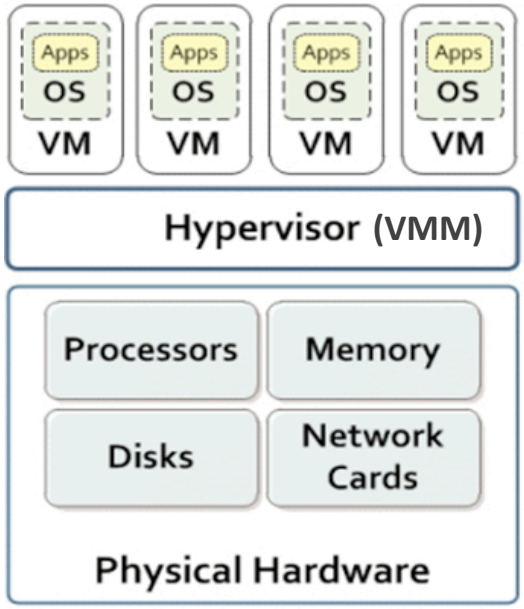
Virtual Machine:

- *"A fully protected and isolated copy of the underlying physical machine's hardware."* -- definition by IBM

Virtual Machine Monitor (aka Hypervisor):

- *"A thin layer of software that's between the hardware and the Operating system, virtualizing and managing all hardware resources"*

VM Classification: Type 1 vs Type 2



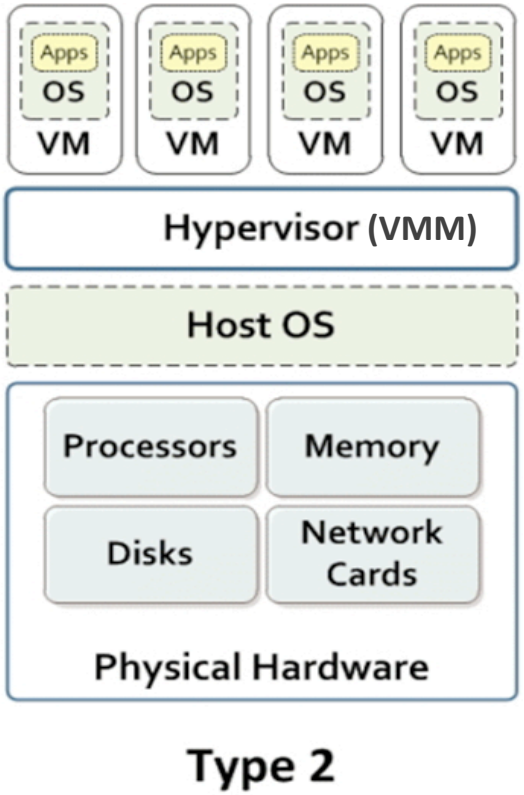
Type 1

- VMM implemented directly on physical hardware
- VMM performs scheduling and allocation of system's resources
- E.g., IBM VM/370, Disco, VMware's ESX Server, Xen

(http://www.ibm.com/developerworks/library/mw-1608-dejesus-bluemix-trs/index.html)

VM Classification: Type 1 vs Type 2

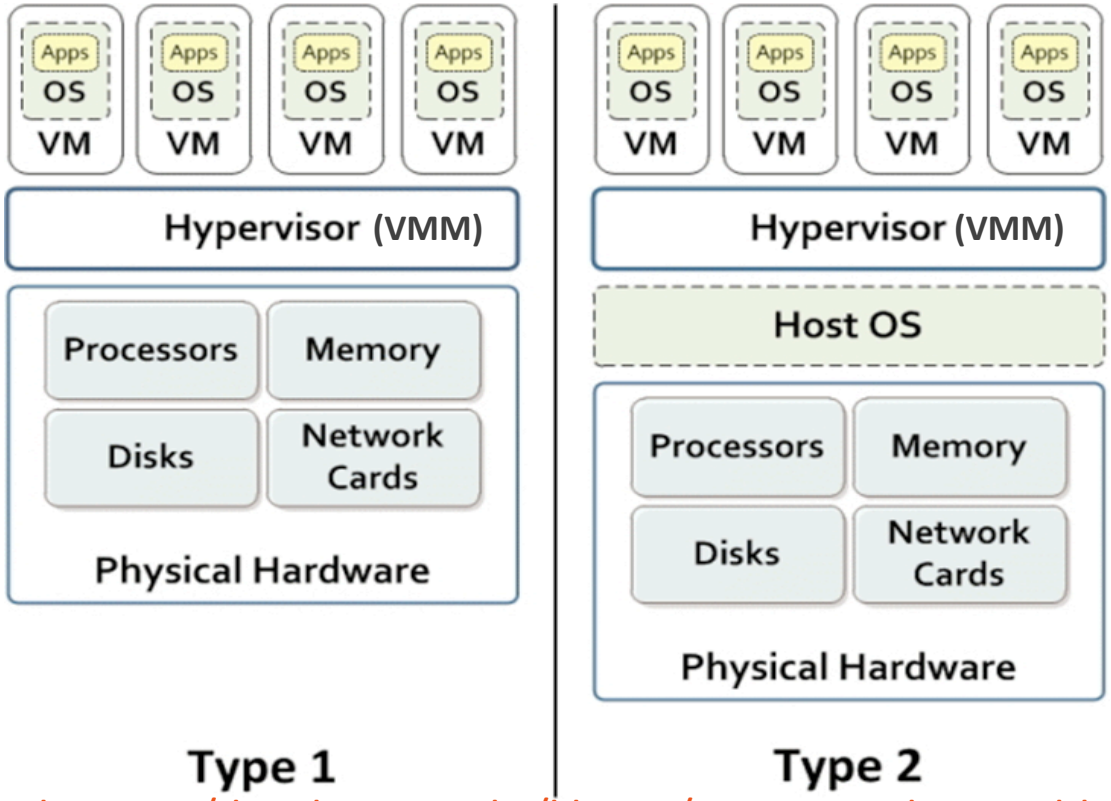
- VMMs built completely on top of a host OS
- Host OS provides resource allocation and standard execution environment to each “guest OS”
- User-mode Linux (UML), UMLinux



Type 2

(http://www.ibm.com/developerworks/library/mw-1608-dejesus-bluemix-trs/index.html)

VM Classification: Type 1 vs Type 2



(http://www.ibm.com/developerworks/library/mw-1608-dejesus-bluemix-trs/index.html)

Virtual Machine – An Old Idea

CP/CMS (*Control Program/Cambridge Monitor System*), 1968

- Time sharing providing each user with a single-user OS
- Allow users to concurrently share a computer

IBM System/370, 1972

- Run multiple OSes

Hot research topic in 60s-70s; entire conferences devoted to VMs

Why Virtual Machine?

IBM System/360

Multiplex an expensive machine

Ability to run multiple OSes targeted for different workloads

Ability to run new OS versions side by side with a stable older version

- No problem if new OS breaks compatibility for existing apps; just run them in a VM on old OS

What Happened?

Interest died out in 80s (why?)

More powerful, cheaper machines

- Could deploy new OS on different machine

More powerful OSes (e.g., Unix running on Workstations)

- No need to use VM to provide multi-user support

Sun Workstation, late 80s

New Motivation (1990s)

Multiprocessor in the market

- Innovative Hardware

Hardware development faster than system software

- Customized OS are late, incompatible, and possibly bug

Commodity OSes not suited for multiprocessors

- Do not scale due to lock contention, memory architecture
- Do not isolate/contain faults; more processors, more failures

Two Approaches

Modify OS to handle multiprocessor machines:

- Hive , Hurricane, Cellular-IRIX
- Innovative, single system image
- But large effort, can take a long time

Hard-partition machine into independent failure units: OS-light

- Sun Enterprise10000 machine
- Partial single system image
- Cannot dynamically adapt the partitioning

Disco Solution

Virtual Machine Monitor between hardware and OS running commercial OS

Virtualization:

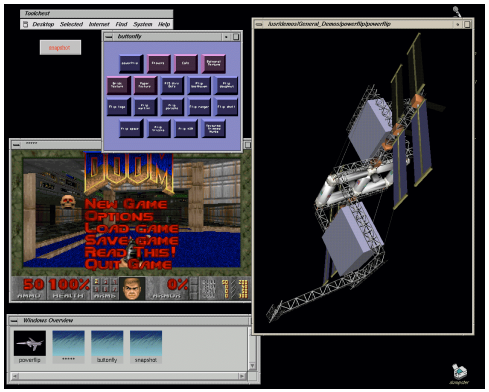
- Used to be: make a single resource appear as multiple resources
- Disco: make multiple resources appear like a single resource

Trading off between performance and development costs

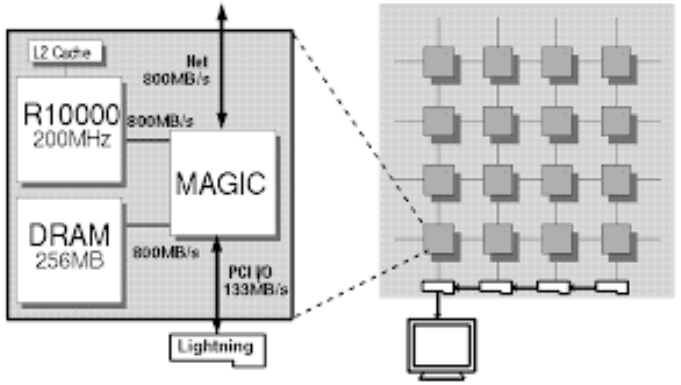
Disco

Extend modern OS to run efficiently on shared memory multiprocessors with minimal OS changes

A VMM built to run multiple copies of Silicon Graphics IRIX operating system on Stanford Flash multiprocessor

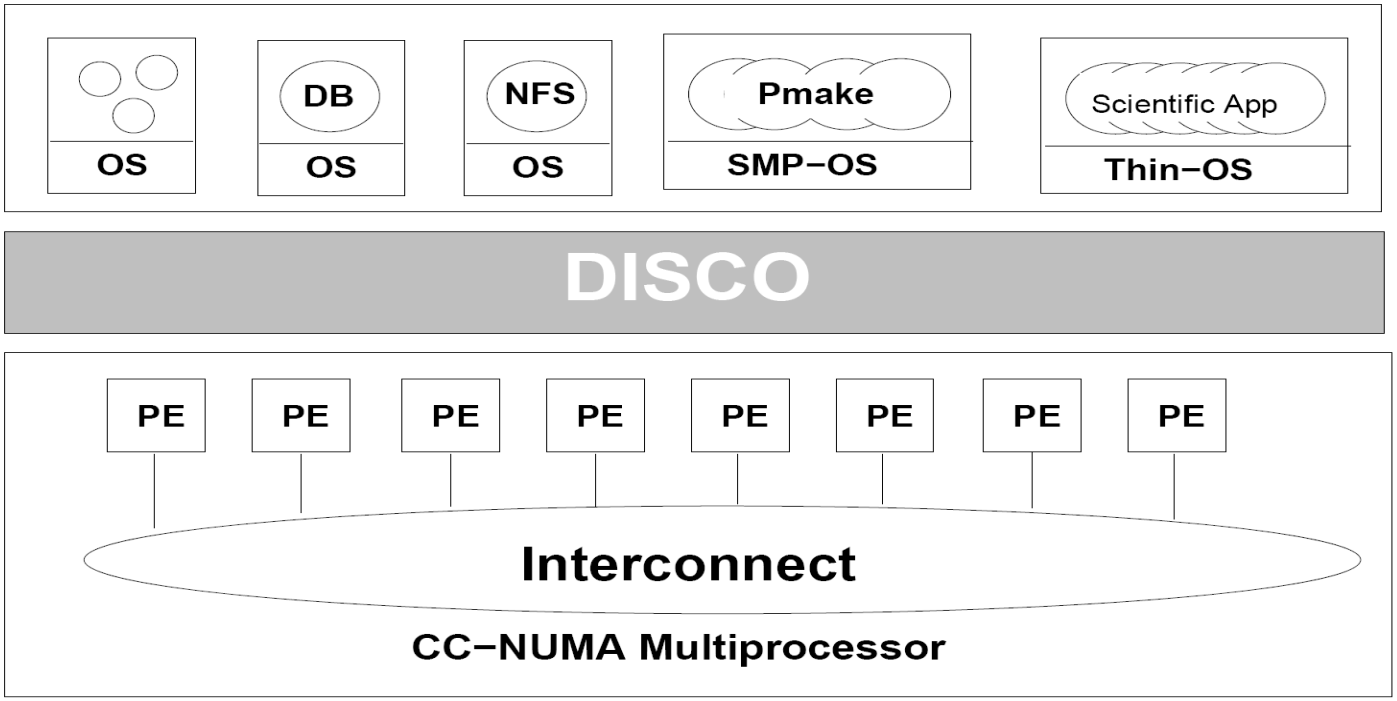


IRIX Unix based OS



Stanford FLAS: cache coherent NUMA

Disco Architecture



Challenges Facing Virtual Machines

Overhead

- Trap and emulate privileged instructions of guest OS
- Access to I/O devices
- Replication of memory in each VM

Resource Management

- Lack of information to make good policy decisions

Communication and Sharing

- Hard to communicate between stand alone VMs

Disco's Interface

Processors

- MIPS R10000 processor: emulates all instructions, MMU, trap architecture
- Extension to support common processor operations
  - Enabling/disabling interrupts, accessing privileged registers

Physical memory

- Contiguous, starting at address 0

I/O devices

- Virtualize devices like I/O, disks
- Physical devices multiplexed by Disco
- Special abstractions for SCSI disks and network interfaces
  - Virtual disks for VMs
  - Virtual subnet across all virtual machines

Disco Implementation

Multi threaded shared memory program

Attention to NUMA memory placement, cache aware data structures and IPC patterns

Disco code copied to each flash processor

Communicate using shared memory

Virtual CPUs

Direct execution on real CPU:

- Set real CPU registers to those of virtual CPU
- Jump to current PC of virtual CPU

Maintains data structure for each virtual CPU for trap emulation

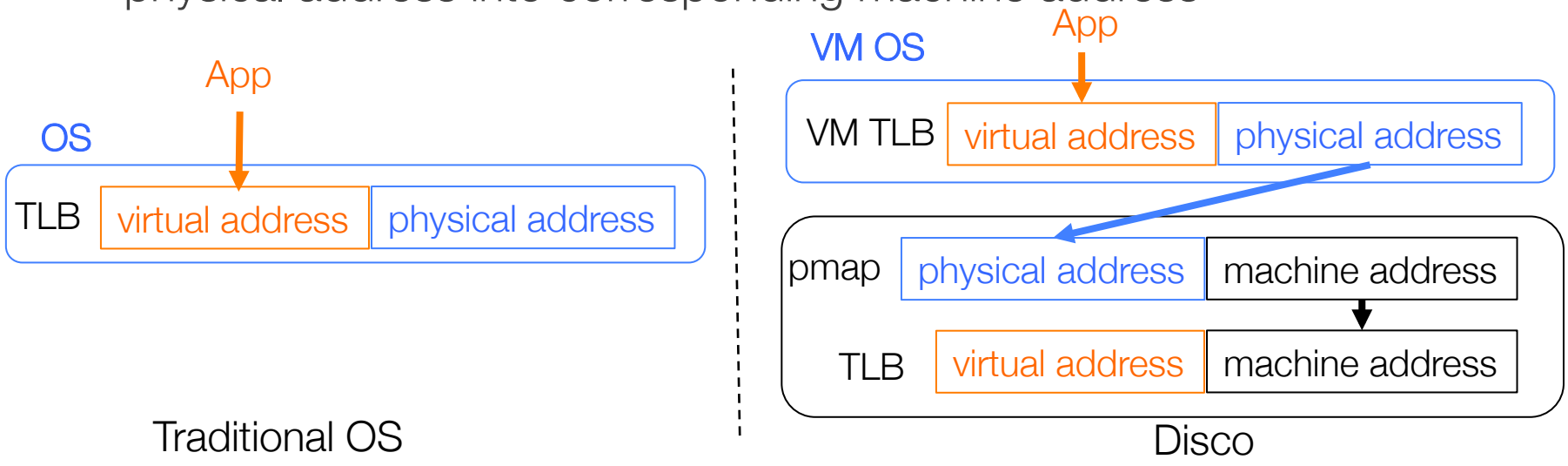
- Trap examples: page faults, system calls, bus errors

Scheduler multiplexes virtual CPU on real processor

Virtual Physical Memory

Extra level of indirection: physical-to-machine address mappings

- VM sees contiguous physical addresses starting from 0
- Map physical addresses to the 40 bit machine addresses used by FLASH
- When OS inserts a virtual-to-physical mapping into TLB, translates the physical address into corresponding machine address



Virtual Physical Memory

Extra level of indirection: physical-to-machine address mappings

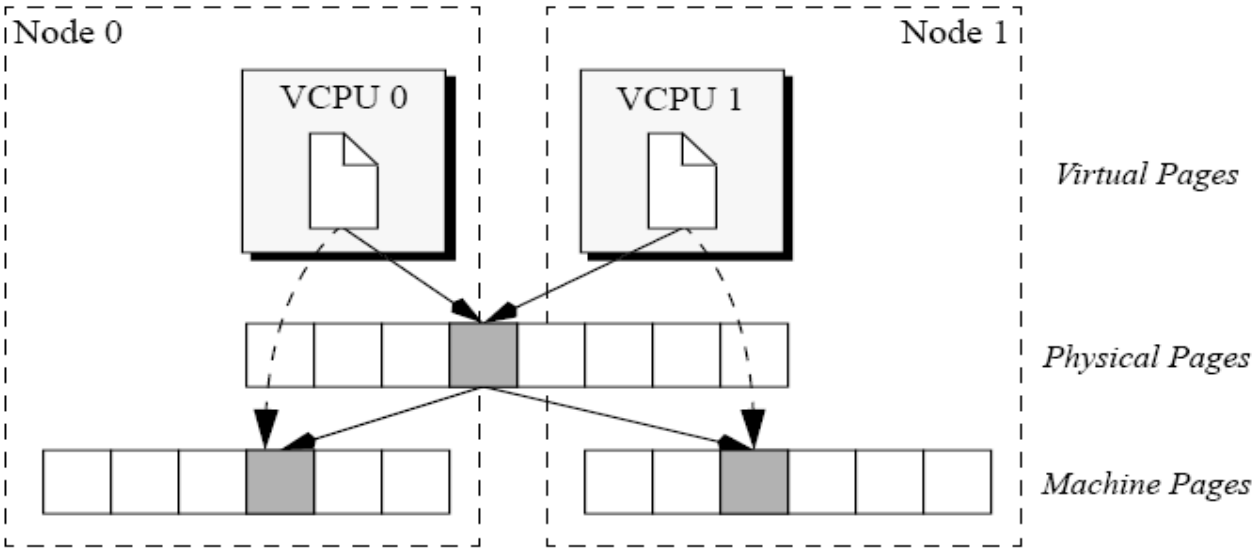
- VM sees contiguous physical addresses starting from 0
- Map physical addresses to the 40 bit machine addresses used by FLASH
- When OS inserts a virtual-to-physical mapping into TLB, translates the physical address into corresponding machine address
- To quickly compute corrected TLB entry, Disco keeps a per VM pmap data structure that contains one entry per VM physical page
- Each entry in TLB tagged with address space identifier (ASID) to avoid flushing TLB on MMU context switches
  - TLB flushed when scheduling a different virtual CPU on a physical processor
- Software TLB to cache recent virtual-to-machine translations

NUMA Memory Management

Dynamic Page Migration and Replication

- Pages frequently accessed by one node are migrated
- Read-shared pages are replicated among all nodes
- Write-shared are not moved, since maintaining consistency requires remote access anyway
- Migration and replacement policy is driven by cache-miss-counting facility provided by the FLASH hardware

Transparent Page Replication



1. Two different virtual processors of same virtual machine logically read-share same physical page, but each virtual processor accesses a local copy
2. *memmap* maintains an entry for each machine page that contains which virtual page reference it; used during TLB shootdown*

*Processors flushing their TLBs when another processor restrict access to a shared page.

Disco Memory Management

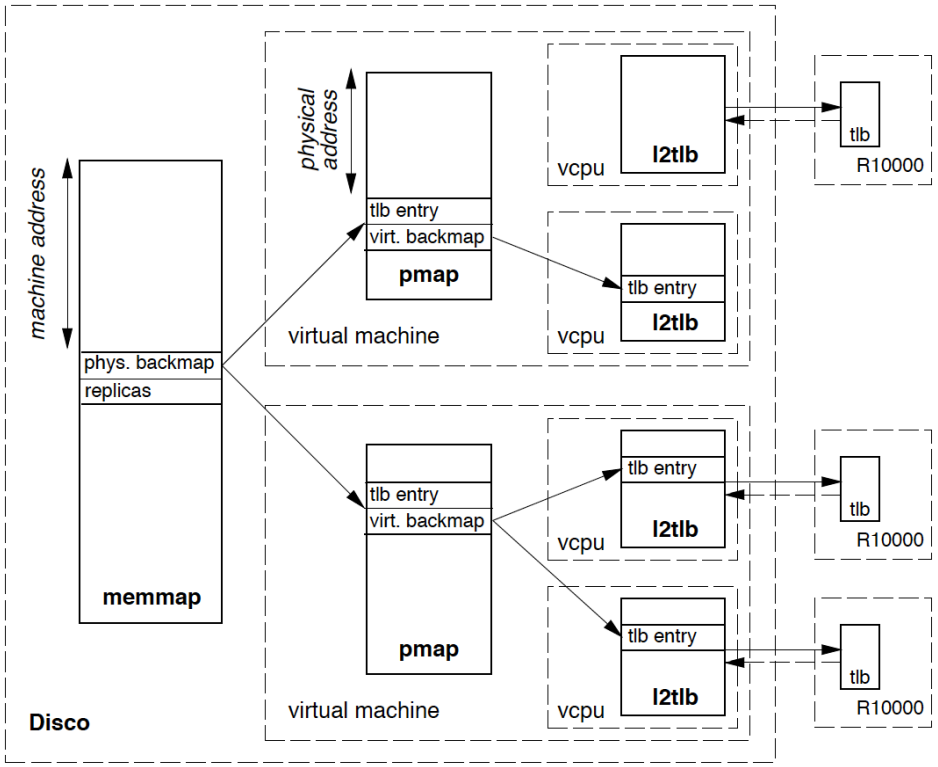
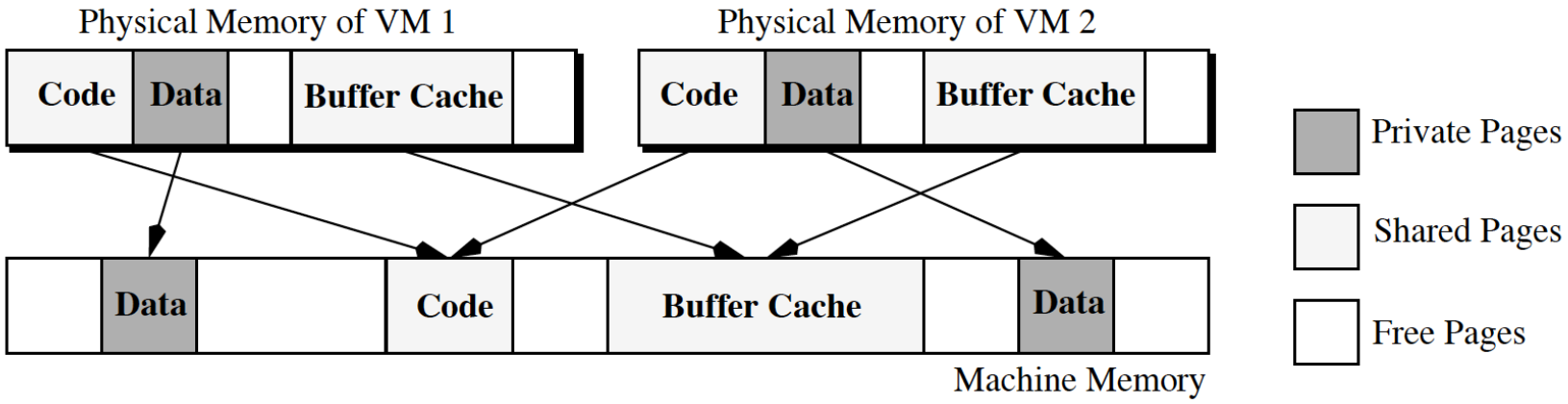


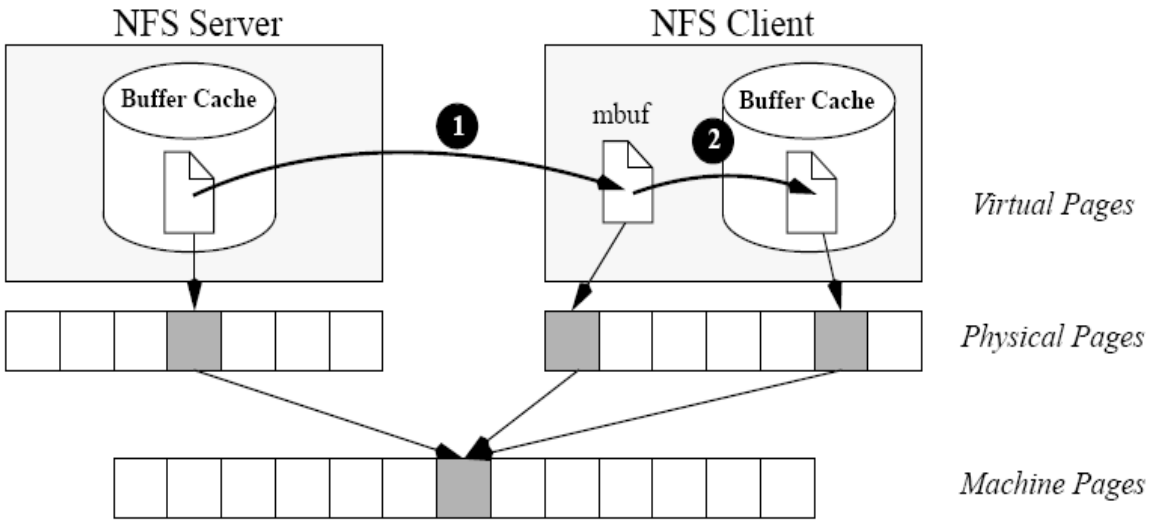
Fig. 3. Major Data Structures of Disco

Transparent Page Sharing



Global buffer cache that can be transparently shared between virtual machines

Transparent Page Sharing over NFS



1. The monitor's networking device remaps data page from source's machine address to destination's
2. The monitor remaps data page from driver's to client's buffer cache

Impact

Revived VMs for the next 20 years

- Now VMs are commodity
- Every cloud provider, and virtually every enterprise uses VMs today

VMWare successful commercialization of this work

- Founded by authors of Disco in 1998, $6B revenue today
- Initially targeted developers
- Killer application: workload consolidation and infrastructure management in enterprises

Summary

Disco VMM hides NUMA-ness from non-NUMA aware OS

Disco VMM is low effort

- Only 13K LoC

Moderate overhead due to virtualization

- Only 16% overhead for uniprocessor workloads
- System with eight virtual machines can run some workloads 40% faster

Xen

Xen Motivation

Performance

- Goal to run 100 simultaneously

Security: resource isolation

Functionality

Virtualization vs Paravirtualization

Full Virtualization: expose “hardware” as being functionally identical to the physical hardware (e.g., Disco, VMWare)

- Cannot access the hardware
- Challenging to emulate some privileged instructions (e.g., traps)
- Hard to provide real-time guarantees

Paravirtualization: just expose a “similar” architecture, not 100% identical to physical hardware (e.g., Xen)

- Better performance
- Need modifications to the OS
- Still no modifications to applications

The Cost of Porting an OS to Xen

Privileged instructions

Page table access

Network driver

Block device driver

<2% of code-base

Xen Terminology

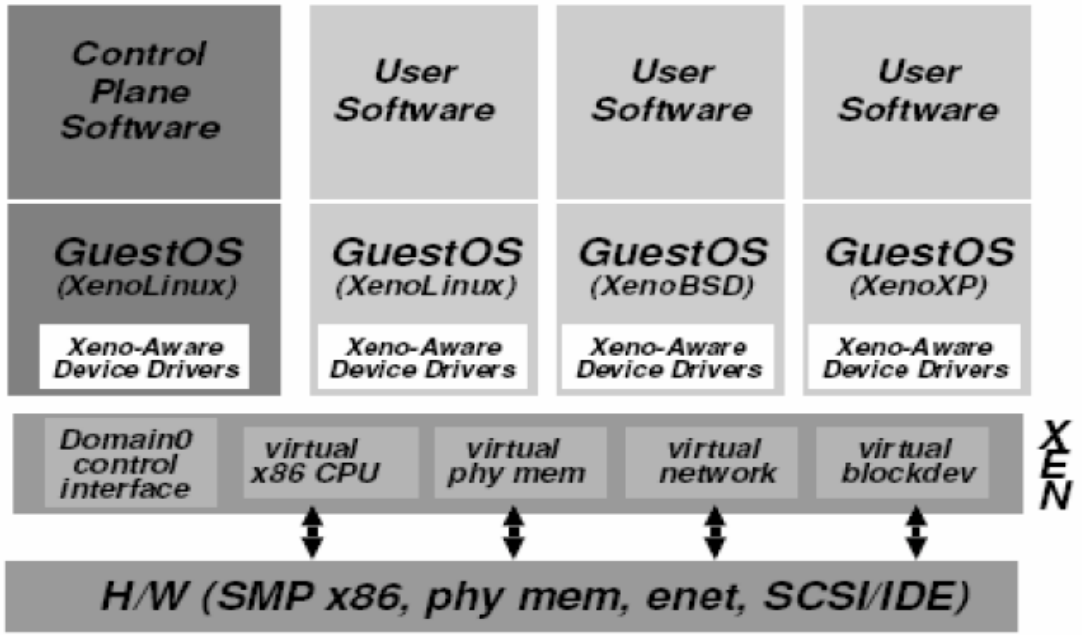
Guest OS: OS hosted by VM

Domain: VM within which a guest operating system executes

- Guest OS and domain analogous to program and process

Hypervisor: VMM

Xen's Architecture



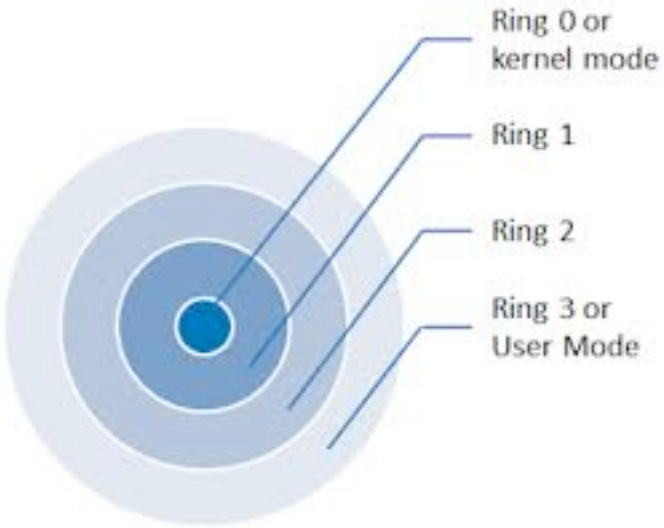
Domain0 hosts application-level management software

- E.g., creation/deletion of virtual network interfaces and block devices

CPU

X86 supports 4 privilege levels

- 0 for OS, and 3 for applications
- Xen downgrades OS to level 1, and it runs on level 2
- System-call and page-fault handlers registered to Xen
- “fast handlers” for most exceptions, doesn’t involve Xen



CPU Scheduling: Borrowed Virtual Time Scheduling

Fair sharing scheduler: effective isolation between domains

However, allows temporary violations of fair sharing to favor recently-woken domains

- Reduce wake-up latency → improves interactivity

Times and Timers

Times:

- Real time since machine boot: always advances regardless of the executing domain
- Virtual time: time that only advances within the context of the domain
- Wall-clock time

Each guest OS can program timers for both:

- Real time
- Virtual time

Control Transfer: Hypercalls and Events

Hypercalls: synchronous calls from a domain to Xen

- Allows domains to perform privileged operation via software traps
- Similar to system calls

Events: asynchronous notifications from Xen to domains

- Replace device interrupts

Exceptions

Memory faults and software traps

- Virtualized through Xen's event handler

Most frequent exceptions: system calls and page faults

- Use 'fast' handler bypassing hypervisor: direct calls from an application into its guest OS avoiding indirection through Xen

Memory

Physical memory

- Reserved at domain creation time
- Statically partitioned among domains

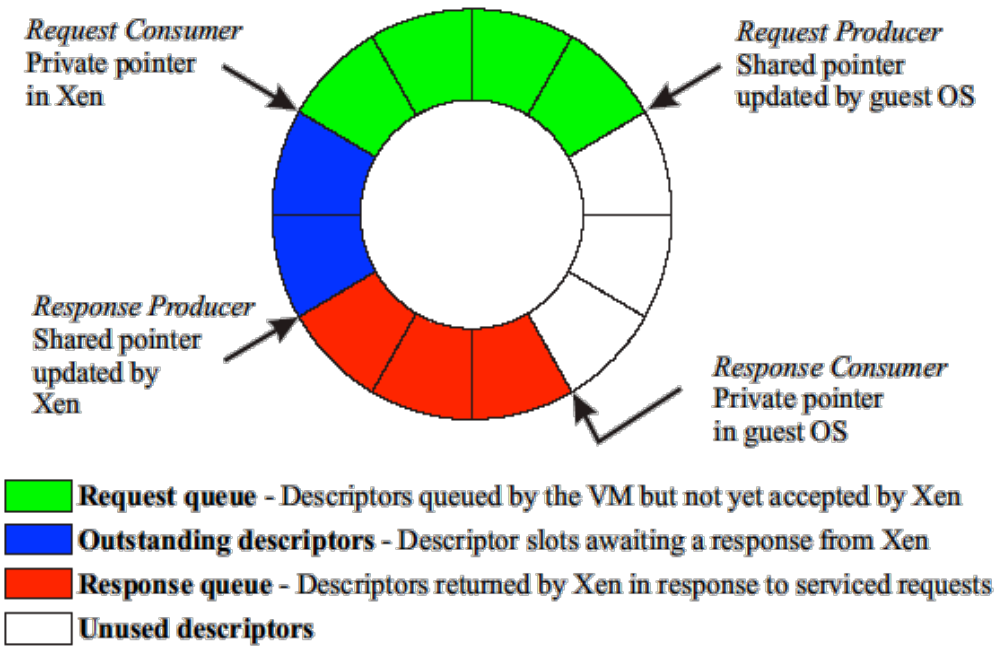
Virtual memory:

- OS allocates a page table from its own reservation and registers it with Xen
- OS gives up all direct privileges on page table to Xen
- All subsequent updates to page table must be validated by Xen
- Guest OS typically batch updates to amortize hypervisor calls

Data Transfer: I/O Rings

Zero-copy semantics

Data is transferred to/from domains via Xen through a buffer ring



History and Impact

Released in 2003 (Cambridge University)

- Authors founded XenSource, acquired by Citrix for $500M in 2007

Widely used today:

- Linux supports dom0 in Linux's mainline kernel
- AWS EC2 based on Xen

Paravirtualization of the MMU