Virtualization

the art of decoupling the logical from the physical

Virtual Machines

- Data centers virtualize hardware by running ***virtual machine monitors (VMM***) or ***hypervisors***

- Hypervisors support ***virtual machines*** on top of hardware

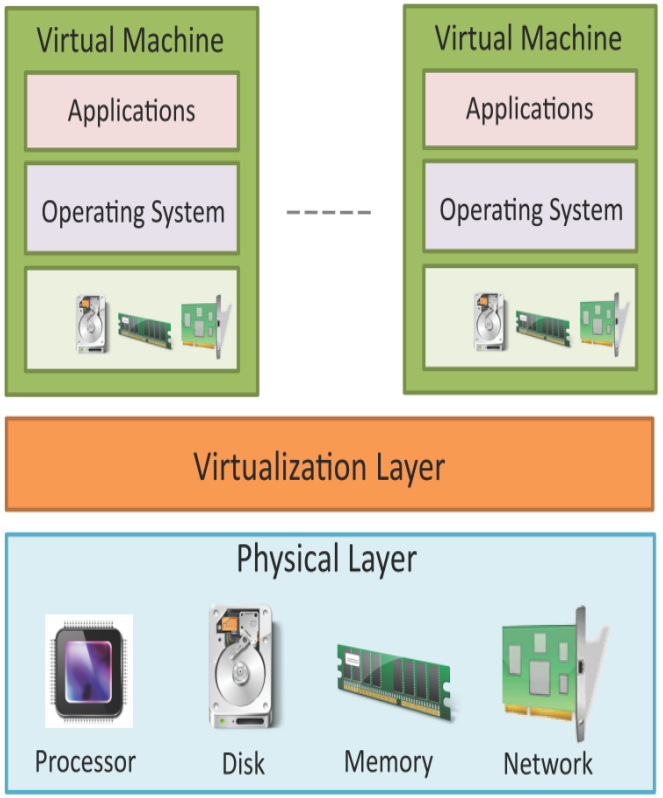
- **virtual machine**
  - a software realization of the underlying hardware
  - Supports implementation of the full instruction set

- Hypervisors allow multiple(100s) of VMs to run simultaneously on the same physical machine.

Virtual Machines

- Users request hypervisor to create and run virtual machines on the data center hardware

- Users can install a **guest OS and applications** on a VM
  - installation typically as simple as installing the OS and applications on a real machine.



History of VM

- IBM initiated VM idea to support legacy binary code
  - Support an old machine's on a newer machine (e.g., CP-67 on System 360/67)
  - Later supported multiple OS's on one machine (System 370)
- Apple's Rosetta ran old PowerPC apps on newer x86 Macs
- MAME is an emulator for old arcade games (5800+ games!!)
  - Actually executes the game code straight from a ROM image
- Modern VM research started with Stanford's Disco project
  - Ran multiple VM's on large shared-memory multiprocessor (since normal OS's couldn't scale well to lots of CPUs)
- VMware (founded by Disco creators):
  - Customer support (many variations/versions on one PC using a VM for each)
  - Web and app hosting (host many independent low-utilization servers on one machine – "server consolidation")

Why Virtualize?

- Consolidate machines
  - Huge energy, maintenance, and management savings
- Isolate performance, security, and configuration
  - Stronger than process-based
- Stay flexible
  - Rapid provisioning of new services
  - Easy failure/disaster recovery (when used with data replication)
- Cloud Computing
  - Huge economies of scale from multiple tenants in large datacenters
  - Savings on mgmt, networking, power, maintenance, purchase costs

How to Virtualize?

- Many ways in which data center resources can be virtualized and shared across multiple tenants.
  - Resources – memory, CPU, I/O, network
- Modern OS already support virtual memory, and are multiuser systems.
- So…

Approach 1 : Let OS handle it

- **Simplest approach is to install a standard operating system on a data center and allow multiple processes to run on this OS**

- **Disadvantages:**
  - Process isolation not achievable (in terms of scheduling priority, memory demand, network traffic, disk access)
  - Process isolation is usually required in systems (think conflicting processes running on the same machine)
  - System administration becomes difficult

- **Solution**

  Retrofit isolation to OS → resource containers

Approach 2 : Resource Containers!

- **Resource containers monitor resource usage and account it to the correct process**

- **Disadvantages**
  - Complex: Resource usage need to be correctly labeled to the 'resource using' process. Not to obvious how it can be done. For example, administrative tasks by the OS use resources. This should be charged to which process?
  - No Flexibility: Only one kernel. If its windows, you can run only windows applications.
  - Security Concerns: Processes running on the same machine introduces security requirements. Checks will be required for monitoring malicious processes.
- **Solution**

  Hypervisors..

Hypervisors

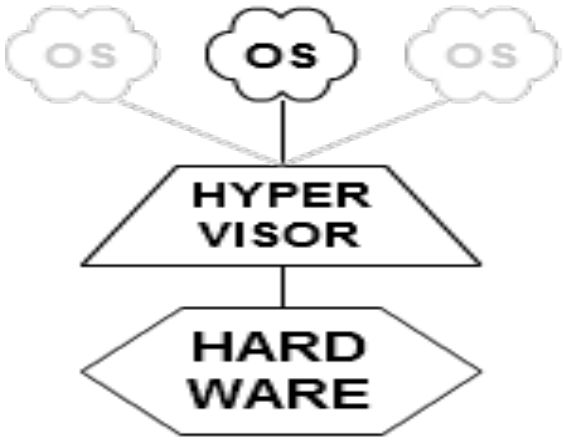
Hypervisors multiplex at a level lower than the processes, i.e. at the **OS level**.

- This provides:
  - Isolation: Each OS on its own! OS responsible for isolation amongst applications. Hypervisor ensures isolation amongst guest OS.
  - Security: Same reasoning as above, each OS is separate.
  - Flexibility: Multiple Operating Systems can be run and they can be different ones

- Hypervisors have solutions to the problems faced by resource containers, but multiplexing OS affects performance. (booting, communication, privileged instruction execution, etc)

- But experience show that advantages outweigh the performance hits (which, if engineered well can be kept quite low)!

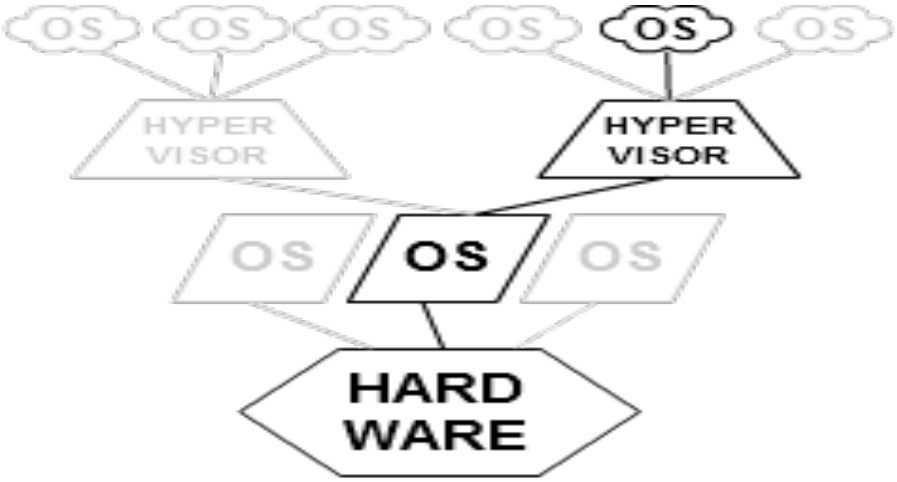
Types of Hypervisors



TYPE 1

native
(bare metal)

These hypervisors run directly on th[illegible] ha[illegible] operating systems. A guest operating system runs as a process on the host.



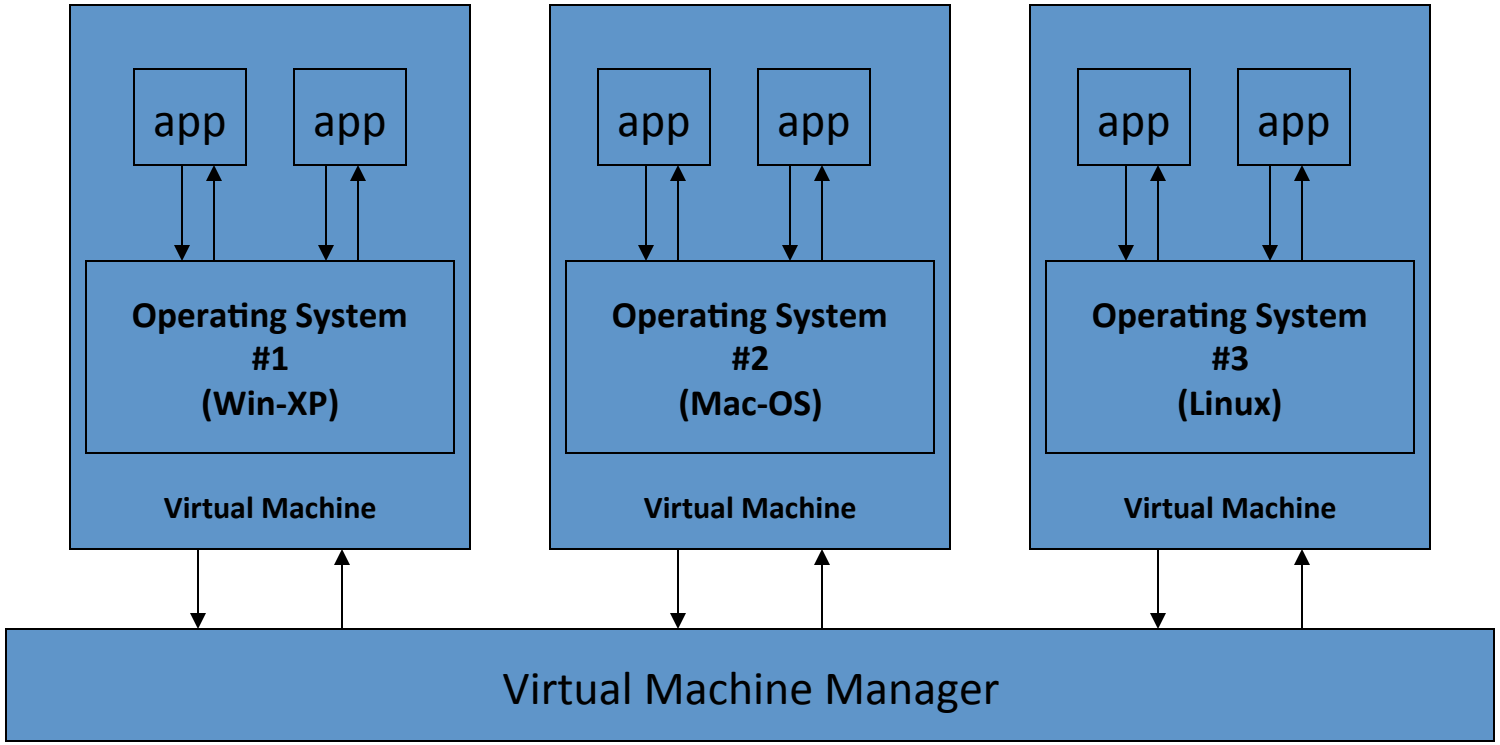
TYPE 2

hosted

These hypervisors run on a [illegible]g system [illegible]r programs do. Type-2 hypervisors abstract guest operating systems from the host operating system.

We will focus on Type 1Hypervisors.

Challenge in designing Hypervisor – loss of control by the OS!



Each operating system was designed to be in total control of the system, which makes it impossible for two or more operating systems to be executing concurrently on the same platform – 'total control' is taken away from them by a new layer of control-software: the VMM

VMM Challenge (2)

- Now the real master is no longer the OS but the "Virtual Machine Monitor" (VMM) or "Hypervisor"
- OS no longer runs in most privileged mode (reserved for VMM)
  - x86 has four privilege "rings" with ring 0 having full access,
    VMM = ring 0, OS = ring 1, app = ring 3
  - Some architectures have only two privilege levels, ring 0 having full access
    VMM = ring 0, OS and app = share ring 1
- But OS thinks it is running in priveleged mode and issues those instructions?
  - Ideally if those instructions cause traps, the VMM can emulate the privileged instructions.
  - Not always possible since one some machines (e.g., old x86) some privileged instructions fail silently (different outcomes when performed from privileged mode versus user mode).
- Virtualizing memory management can be very expensive (we will see)
  - Simple idea of trapping every virtual memory reference by Guest OS and doing translation at the hypervisor will kill performance.

VMM Approaches

- Full virtualization
  - almost complete simulation of the actual hardware to allow software, which typically consists of a guest operating system, to run unmodified. (Disco/Vmware/IBM)

- Paravirtualization
  - an exact hardware environment is not simulated; however, the guest programs are executed in their own isolated domains, as if they are running on a separate system. Guest programs need to be specifically modified to run in this environment. Modifications limited to very small part of the OS and usually not application code (Denali, Xen)

  - Moving to a paravirtual machine is like porting software to a very similar machine.

VMM Approaches (Examples)

- Full virtualization – runs unmodified OSs and applications
  - Use software emulation to shadow system data structures (DISCO),
  - Dynamic binary rewriting of OS code that modifies system structures, (VMWare)
  - Hardware virtualization support (IBM)

- Paravirtualization : slight modification to the guest program
  - Change OS but not applications – support the full Application Binary Interface (ABI) (Xen)

Why Paravirtualize?

- For some hardware (e.g., x86) full virtualization is difficult to realize
  - Some supervisor instructions that MUST run be run by VMM (in privileged mode) do not cause a trap when executed with insufficient privilege.
  - Solutions around it result in severe performance hit.
- Efficiently emulating the MMU is difficult -- causes reduced performance.
- For certain operations, Guest OS should be exposed to both virtual as well as real resources.
  - E.g., guest OS need to know real time for TCP timeouts.

Xen Approach: minimal changes to the Guest OS to make VM simpler and higher performance, but NO changes to application program

Design Principles for VMM

- While designing a VMM there are a few design principles that should be kept in mind
  - Support for unmodified application binaries
  - Support for multi- application operating systems
  - None or minimal changes to operating system
  - In case of Para-virtualization: not to completely hide resources to obtain high performance and strong resource isolation

What do VMM need to virtualize?

- All the subsystems of the physical machine
  - *CPU*
  - *Physical memory*
  - *Network*
  - *Disk*
  - *Time and timers*

- Essentially all underlying hardware components that the guest OS needs.

- The guest OS must be able to run all its services correctly on the virtualized hardware
  - *E.g., virtual memory address translation, privileged instructions, file read/write, network communication, …*

Virtualizing CPU

Let us look at how Disco virtualized the R10000 processor

- Simulate all instructions:
  - Most are directly executed
  - Privileged instructions must be emulated, since we won't run the OS in privileged mode
  - Disco runs privileged, OS runs supervisor mode, apps in user mode

- An OS privileged instruction causes a trap which causes Disco to emulated the intended instruction

- Map VCPUs onto real CPU: registers, hidden registers

- Virtualizing R10000 CPU was easy since privileged instructions running in user mode caused traps!

Virtualizing CPU

Now let us consider how Xen virtualized X86

- x86 harder to virtualize than Mips (as in Disco):
  - Some privileged instructions fail silently rather than fault
  - Xen fixed this by modifyng the guest OS to make transfer control to hypervisor when executing such instructions (**hypercalls**)
  - VMWare fixed it by binary code rewrite

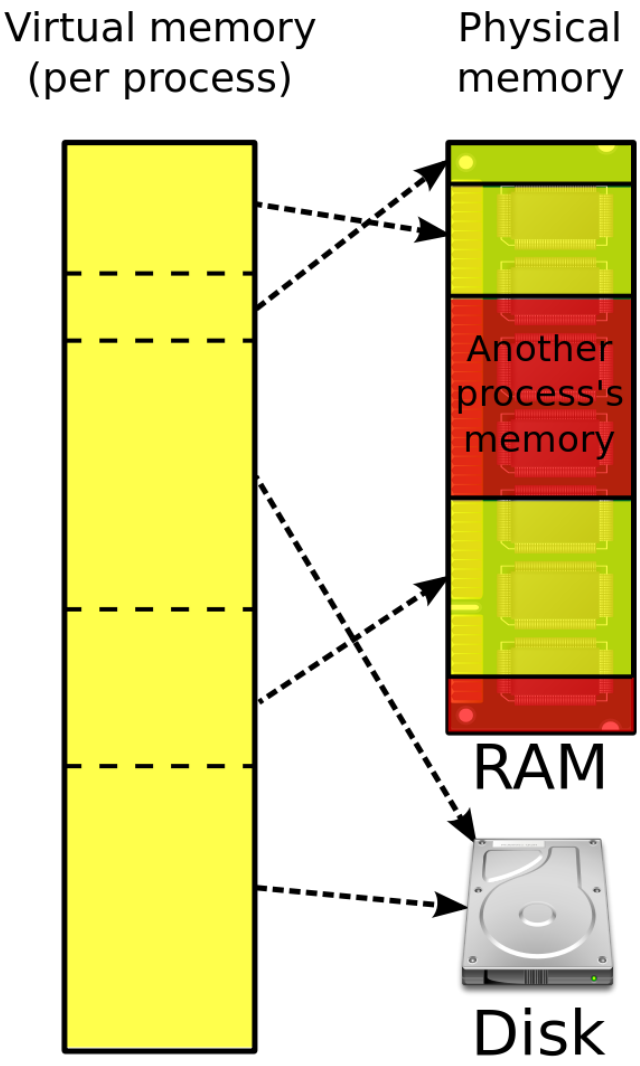
Virtualizing Physical Memory (1)

Let us first review virtual memory concept

- **Virtual memory**
  - Enables applications to share physical memory without the burden of memory management
  - provides abstraction of a contiguous address space even though the corresponding pages may be fragmented in physical memory

- **Virtual memory address mapped to physical memory using an in-memory page table**
  - If page fault occurs, virtual page in disk and OS brings the page into memory

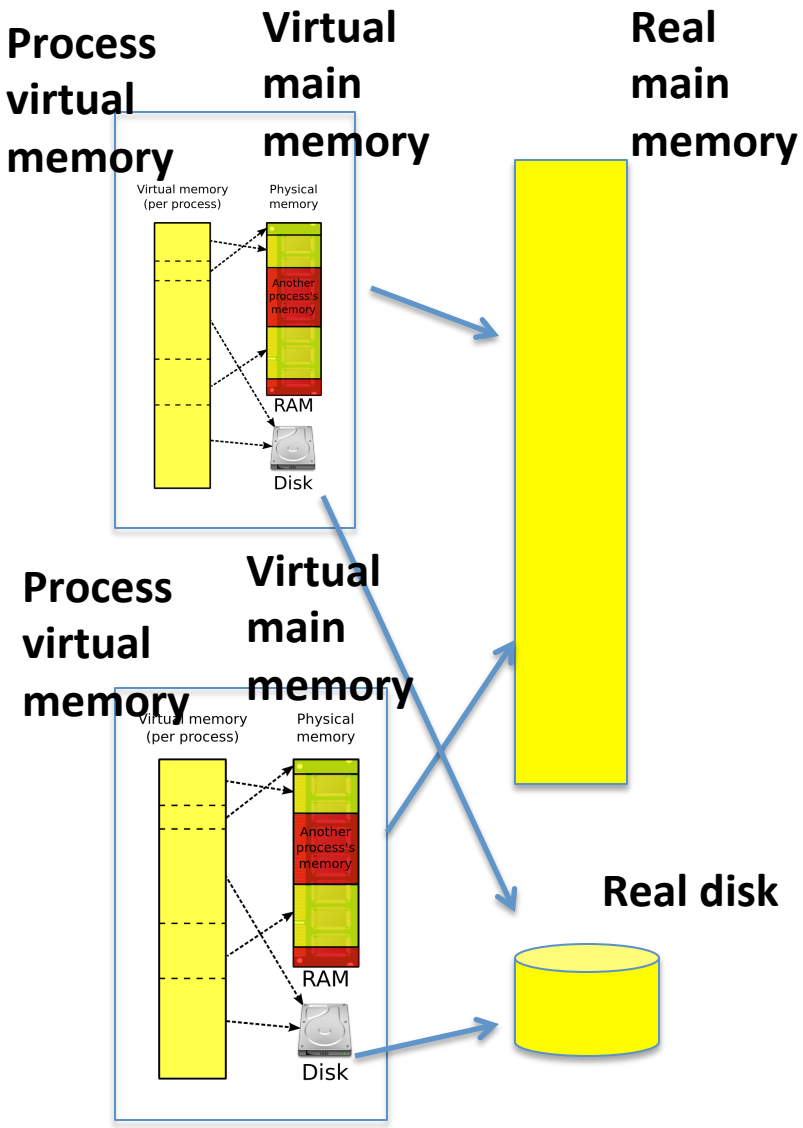
- **TLB serve as associative cache of page table.**
  - TLB could be in hardware or software



Virtualizing Physical Memory (2)

- In a VM, the hypervisor provides an abstraction of a virtualized physical memory (VPM) to the guest OS.
  - Recall guest OS supports virtual memory for its applications.

- To support VPM, hypervisor **adds a level of address translation** and **maintain** physical to machine address mapping.



Virtualizing Physical Memory (3)

Disco running on R10000

- Hypervisor maintains a **pmap** data structure for each VM.
  - pmap stores entry for each virtual physical page for the VM
  - Maps the physical page into the corresponding to page on real main memory on the machine.
- When guest OS tries to insert a virtual → physical mapping into the TLB, the hypervisor emulates the operation, translates the physical to machine address (using pmap), and inserts the correct entry into TLB.
- Subsequent reference through this mapping translated with no overhead.

Virtualizing Physical Memory (4)

- Many details:
  - On MIPS user-mode memory reference goes through TLB but OS level reference may access physical memory directly.
  - So above solution may not work.
  - Disco implements workarounds
- Slower compared to running on a real machine
  - E.g., TLB flush whenever context switched to a different VM.
- Virtualizing Physical Memory more tricky in Xen
  - Designed for x86 that had hardware TLB and not software TLB.

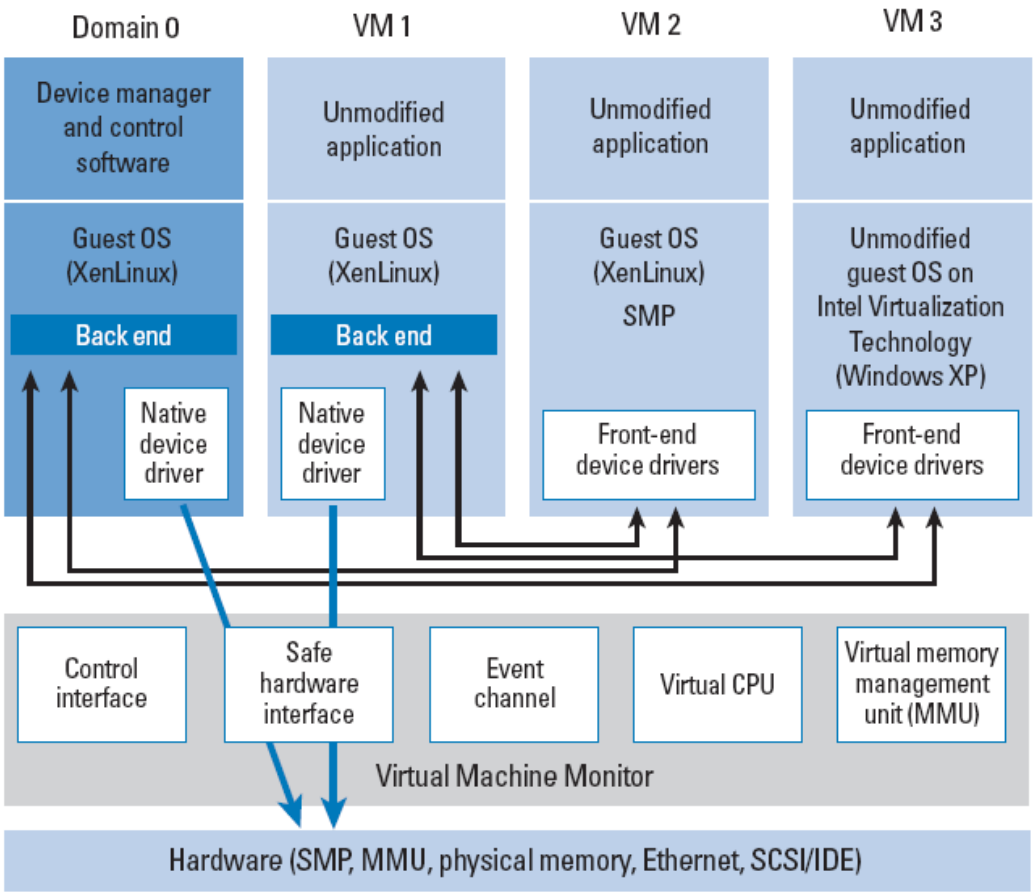
Additional Issues..

- We only focused on challenges in virtualizing physical memory …
- And that too partially with many issues not discussed
  - How to partition machine's main memory amongst virtual machines?
  - Can we reclaim part of the memory from one VM (in case it is not using it), and give to another VM (balooning)
  - When to reclaim, and to assign to who?

- Did not cover challenges in virtualizing I/O devices, networks, …
  - Hopefully we covered enough for you to explore the papers.

One key idea in Xen implementation Domain 0 (dom0) – separating policy from mechanism

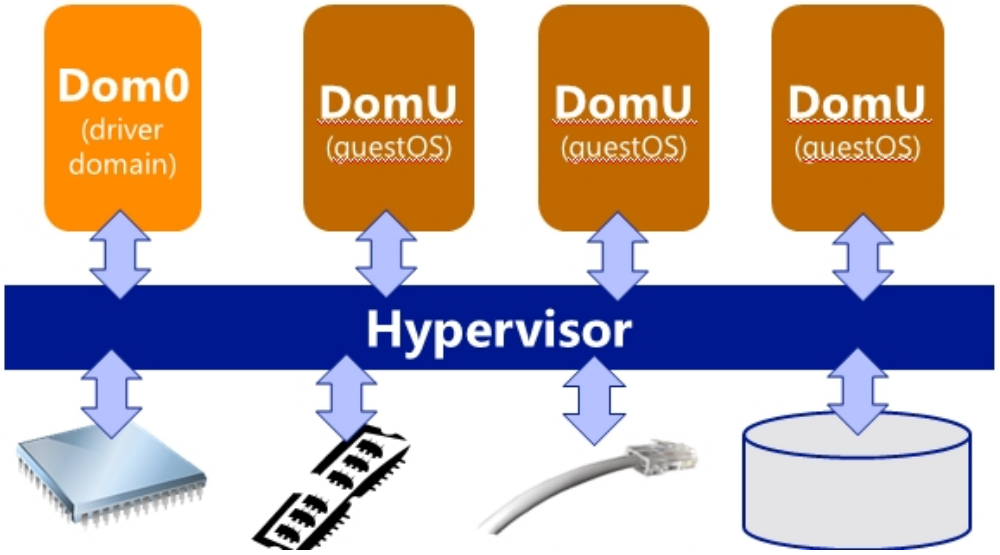
- Nice idea: run the VMM management at user level
  - Given special access to control interface for platform management
  - Has back-end device drivers
- Much easier to debug a user-level process than an OS
- Narrow hypercall API and checks can catch potential errors



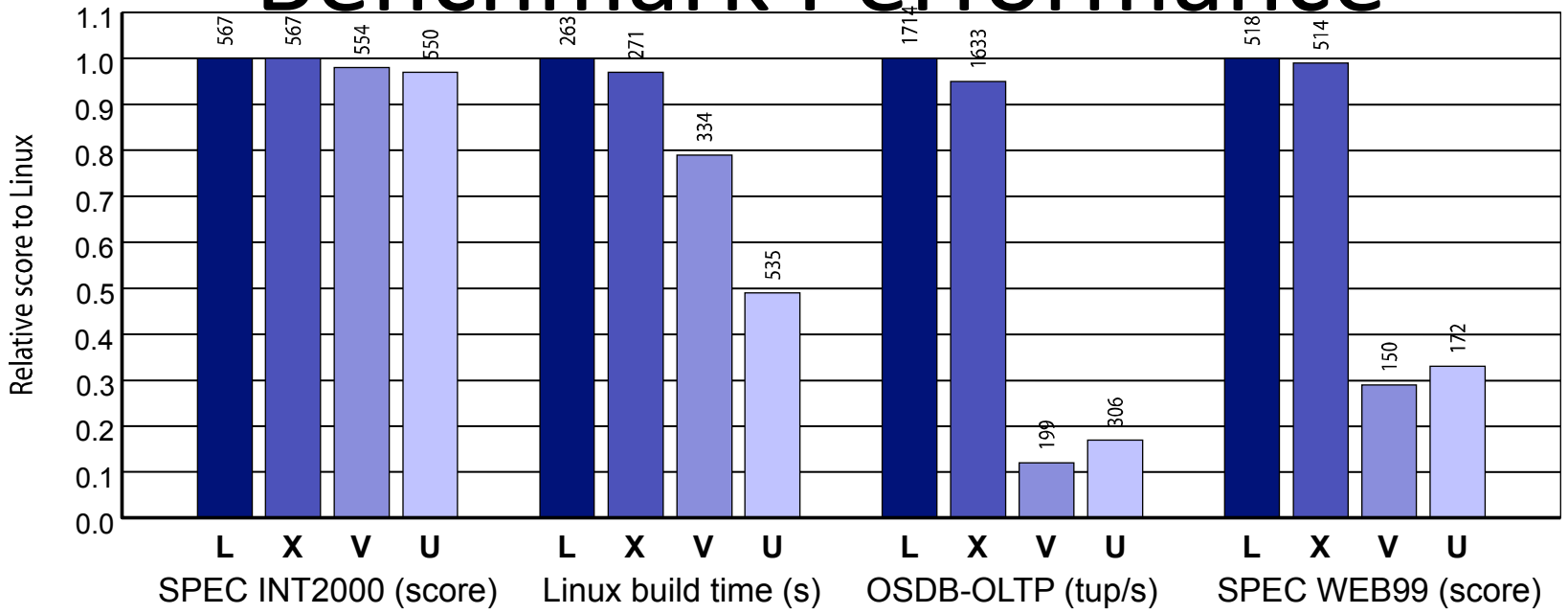
Xen Design : Separate policy from mechanism

Example: Hypervisor needs to schedule CPU between VM's but no need for it to know or be involved in how the CPU is to be shared

- Resultant architecture:
  - Hypervisor provides basic control operations
  - Export these controls through an interface to an authorized domain
  - Complex policy decisions can be made by management software running in ONE authorized domain (termed as domain0 in Xen)



Benchmark Performance



Benchmark suite running on Linux (L), Xen (X), VMware Workstation (V), and UML (U)

- Benchmarks
  - Spec INT200: compute intensive workload
  - Linux build time: extensive file I/O, scheduling, memory management
  - OSBD-OLTP: transaction processing workload, extensive synchronous disk I/O
  - Spec WEB99: web-like workload (file and network traffic)

Xen Overview

- Performance overhead of only 2-5%
- Available as open source but owned by Citrix since 2007
  - Modified version of Xen powers Amazon EC2
  - Widely used by web hosting companies
- Many security benefits
  - Multiplexes physical resources with performance isolation across OS instances
  - Hypervisor can isolate/contain OS security vulnerabilities
  - Hypervisor has smaller attack surface
    - Simpler API than OS – narrow interfaces ➔ tractable security
    - Less overall code than an OS
- BUT hypervisor vulnerabilities compromise everything…

Virtual Machines Migration

Live Migration of Virtual Machines

C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield. Appears in *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation* (NSDI), 2005

Overview

- Data centers support ability for VMs to migrate from one machine to another.
  - In the same rack
  - To a different rack in the same data center
  - Across the internet to another data center.
- Key questions
  - Why (support migration)?
  - When (should we migrate)?
  - Which (VM to migrate)?
  - How (to migrate efficiently)?
- We will focus on the how for the most part.

Why Migration is Useful

- **Load balancing for long-lived jobs** (why not short lived?)
  - Better compliance of SLAs, improved performance, ..
- **Ease of management**: controlled maintenance windows
  - System maintenance without service disruption.
- **Fault tolerance:** move job away from flaky (but not yet broken hardware)
- **Energy efficiency:** rearrange loads to reduce A/C needs

Why is Migrating at VM level better than at the process level

- OS level migration does not have to deal with residual dependencies like open file descriptors
- OS instances are easier to migrate compared to process-level migration
  - Consistent in-memory state transfer, no need to restart apps or re-establish network connections
  - OS level access, e.g. root permissions and internal kernel state, not required to accomplish migration
- Seperation of concerns between users and operators of datacenter or cluster.

Background – Process-based Migration

- Typically move the process and leave some support for it back on the original machine
  - E.g., old host handles local disk access, forwards network traffic
  - these are "residual dependencies" – old host must remain up and in use
- Hard to move exactly the right data for a process – which bits of the OS must move?
  - E.g., hard to move TCP state of an active connection for a process

VMM Migration

- Move the whole OS as a unit – don't need to understand the OS or its state
- Can move apps for which you have no source code (and are not trusted by the owner)
- Can avoid residual dependencies in data center thanks to global names

Implementation Challenges

- **Minimize machine downtime** so that there is no perceivable service disruption
- **Minimize the total migration time** so that the time for which the state must be maintained on both the machines is small
- Should have **no adverse affect on the network utilization** of other machines in the vicinity
- Must be done such that **SLAs are adhered** to even while the migration is in progress
- Should be carefully planned so that **new hotspots are not created** once the migration process completes

How to migrate VM efficiently?

Migration Schemes

Migration involves movement of memory, storage and network connections

- **Migration over a LAN**
  - Involves movement of only in memory data
  - All persistent state is stored on network attached storage and does not need to be moved during the migration
  - No explicit transfer of network connections necessary
- **Migration over a WAN**
  - Migration over a WAN includes movement of both memory and persistent state
  - Network connection migration requires setting up of IP tunnels for the period of time that migration is in progress

Migration over a LAN

Design: Migrating memory

- **Goal:**
  - Minimize downtime
  - Minimize total migration time
- **Most solutions use one or a combination of the following phases**
  - Push phase
  - Stop and copy phase
  - Pull phase

Push Phase: Migrating memory

Memory from the source VM is copied to the destination VM while the source is running.

- **Advantages**:
  - Since memory is transferred in advance, the downtime is low.

- **Disadvantages**:
  - Memory pages modified during the push phase will have to be re-sent. May hog the network between source and destination VM.
  - Some part of the memory will always be modified faster than it can be transferred. This phase alone cannot handle transfer of all the memory.

Stop & Copy Phase: Migrating memory

The source VM is stopped, memory is copied to the destination VM, then started.

- **Advantages**:
  - Memory pages only need to be sent once, unlike push phase
  - No residual dependency on the source VM. Source VM can be shut down after migration occurs

- **Disadvantages**:
  - Downtime is high, both VM's will have to be stopped while the stop and copy completes

Pull based Design: Migrating memory

The new VM is started right after copying only the essential kernel pages. If the new VM accesses a page that has not been copied, a page fault occurs and memory is copied from the source VM.

- **Advantages:**
  - Downtime is low, the destination VM can be started after just copying a minimal number of pages
- **Disadvantages**:
  - There are residual dependency after the migration occurs, the source VM cannot be turned off until all the pages have faulted over.

Design: Migrating memory

Solution**: Pre-copy migration**

- **Iterative push phase**: pre-copying occurs in rounds, bandwidth increased each round until the bandwidth limit is reached
  - In the first round copy all pages
  - In the subsequent rounds, copy only the pages that have been dirtied in previous round, but not dirtied again this round

- **Very short stop-and-copy phase**: when the bandwidth limit is reached, pages are being dirtied as fast as we can copy them. The source VM is shut down, so that those pages can be copied to the destination VM.

Iterative Pre-copy Details (I)

- Allocate resources at the destination (to ensure it can receive the domain)
- Iteratively copy memory pages to the destination host
  - Service continues to run at this time on the source host
  - Any page that gets written will have to be moved again
  - Iterate until a) only small amount remains, or b) not making much forward progress
  - Can increase bandwidth used for later iterations to reduce the time during which pages are dirtied
- Stop and copy the remaining (dirty) state
  - Service is down during this interval
  - At end of the copy, the source and destination domains are identical and either one could be restarted
  - Once copy is acknowledged, the migration is *committed* in the transactional sense

Live Migration Approach (II)

- Update IP address to MAC address translation using “gratuitous ARP” packet
  - Service packets starting coming to the new host
  - May lose some packets, but this could have happened anyway and TCP will recover
- Restart service on the new host
- Delete domain from the source host (no residual dependencies)

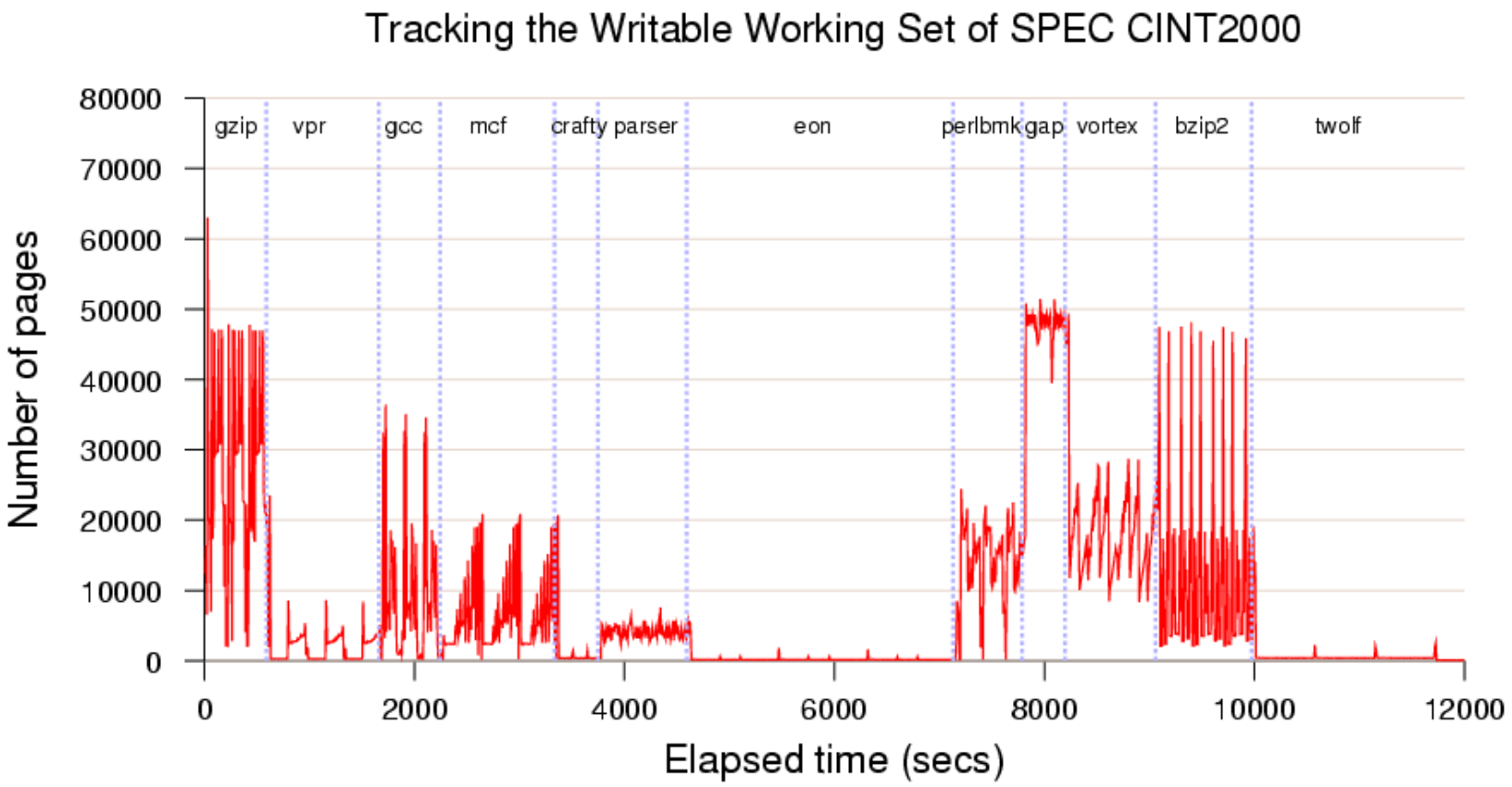
Writable working set

- **How many times should we pre-copy?**
  - If VM does not write memory, a single pre-copy suffices
  - If VM continuously dirties pages faster than copy, then all pre-copy is in vain
- **Reality somewhat in between**
  - Most pages do not change too fast so ideal for pre-copy
  - A small set written often and should be transferred during stop and copy phase.
- **Writabe working set -- set of pages that are updated frequently.**
  - not suitable to be transferred in the pre-copy phase.
  - Must be copied in the stop-and-copy phase.
- **VM downtime proportional to WSS**
- **So how big is WSS usually?**

Tracking the Writable Working Set

- Xen inserts shadow pages under the guest OS, populated using guest OS's page tables
- The shadow pages are marked read-only
- If OS tries to write to a page, the resulting page fault is trapped by Xen
- Xen checks the OS's original page table and forwards the appropriate write permission
- If the page is not read-only in the OS's PTE, Xen marks the page as dirty

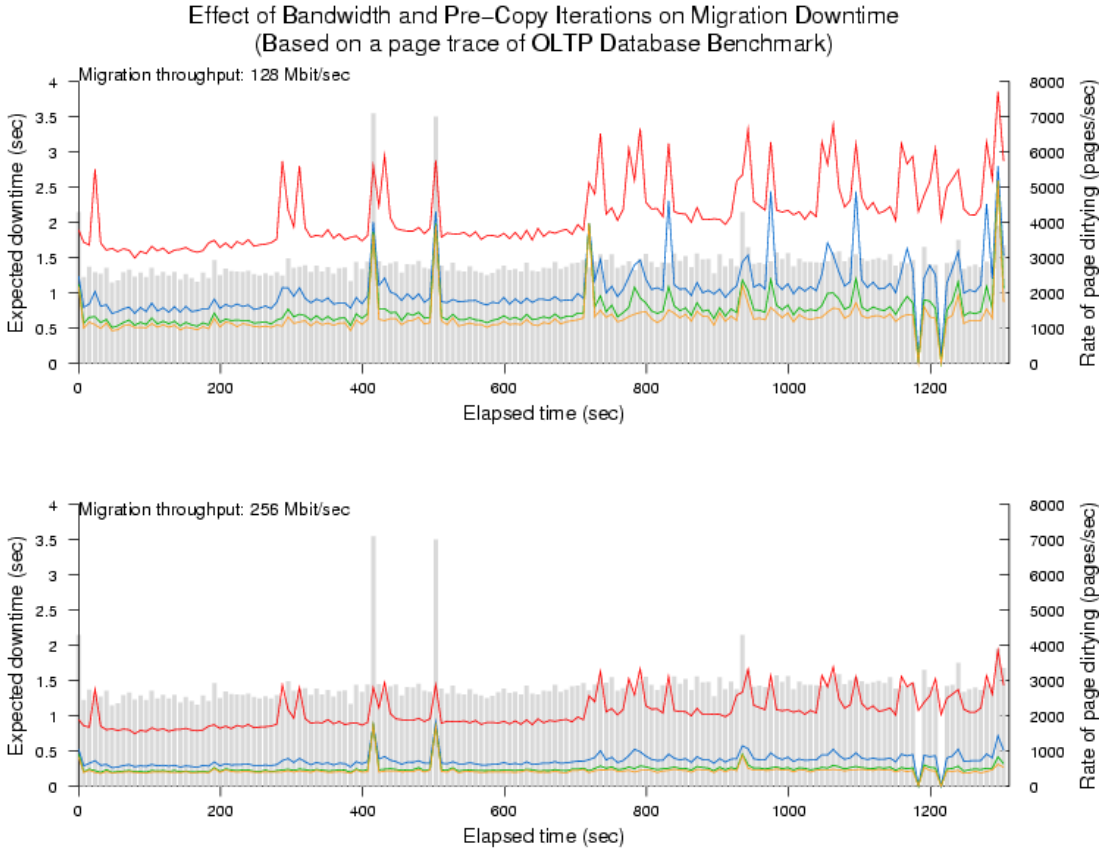
Writable Working Set



Number of pages dirtied every 8 second interval for a set of benchmarks. Lots of variation. For some benchmark it is a small portion of the actual working dataset

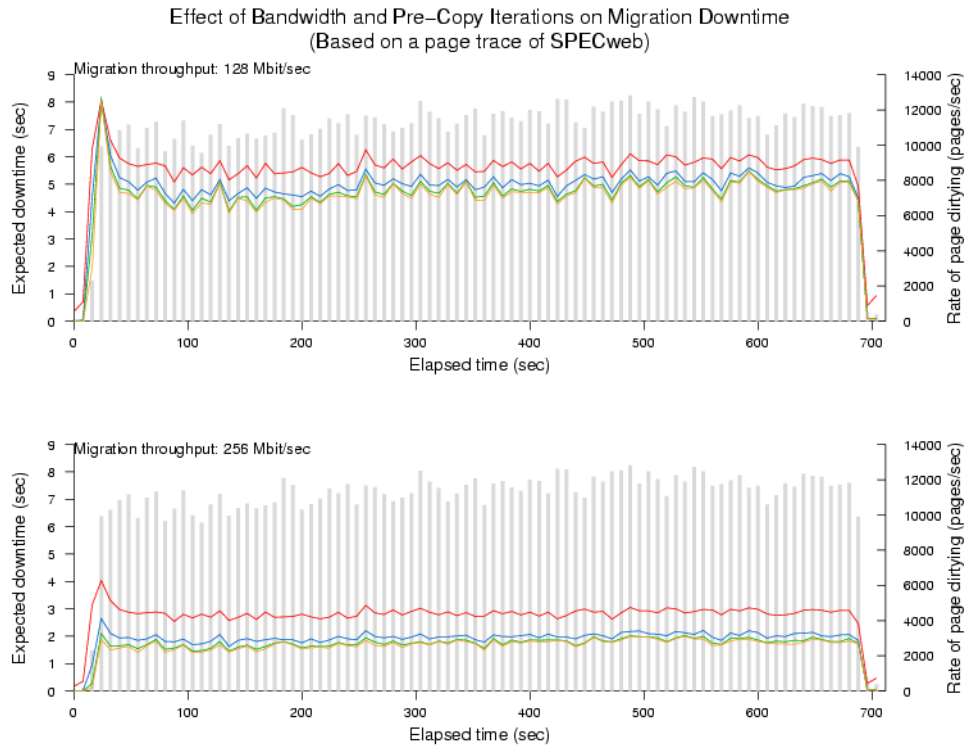
Downtime comparison: OLTP Database

- Compare with stop-and-copy:
  - 32 seconds (128Mbit/sec) or 16seconds (256Mbit/sec)



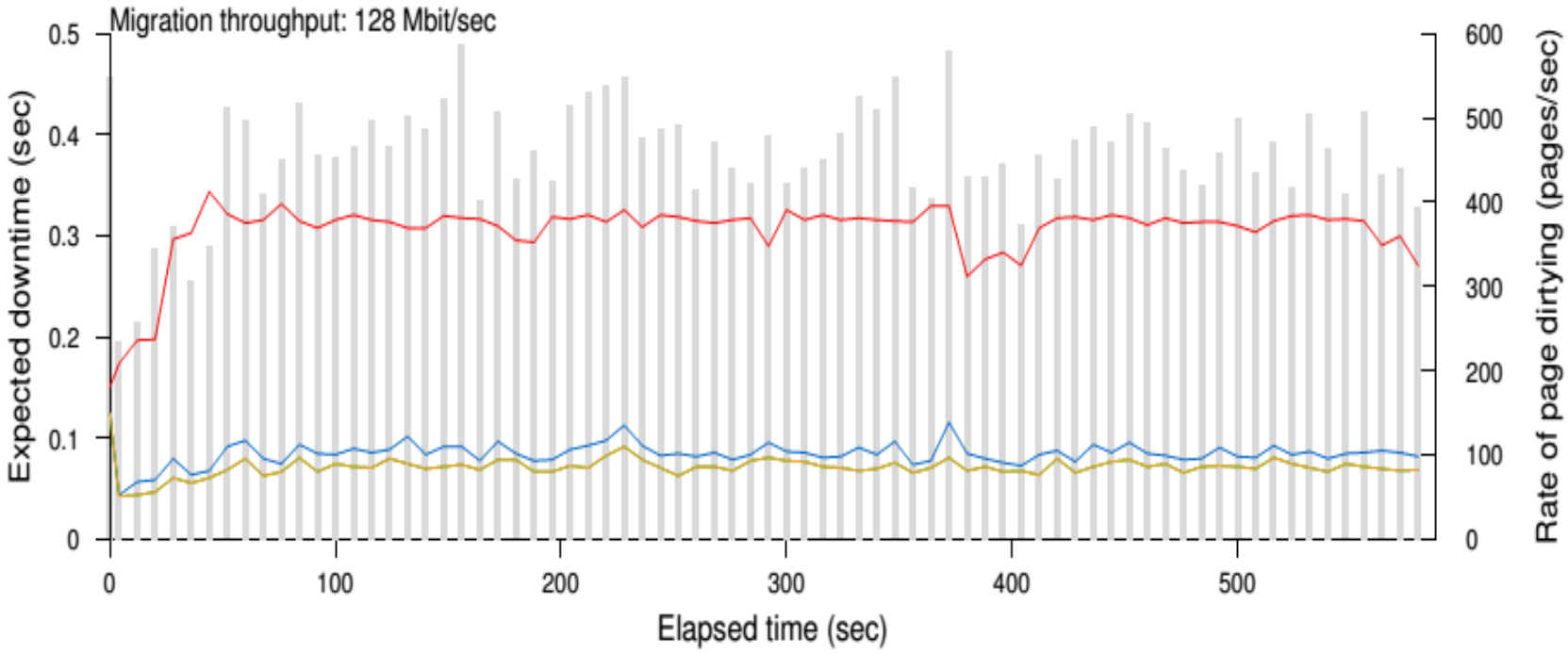
Downtime Comparison: SPECweb

- Compare with stop-and-copy:
  - 32 seconds (128Mbit/sec) or 16seconds (256Mbit/sec)



Estimating Downtime

Expected downtime due to last-round memory copy on traced page dirtying of a Quake 3 server

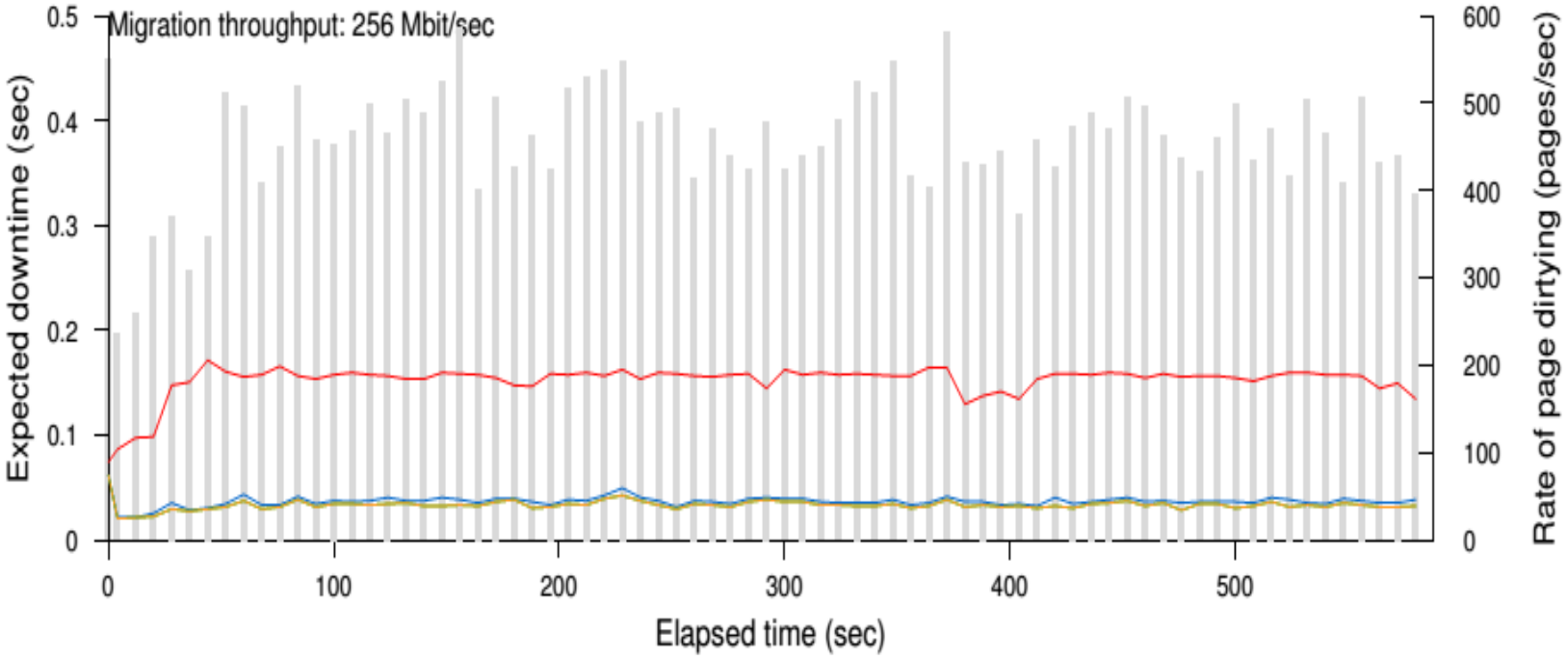


Estimated downtime with 1 iteration of pre-copy (sec)
Estimated downtime with 2 iterations of pre-copy (sec)
Estimated downtime with 3 iterations of pre-copy (sec)
Estimated downtime with 4 iterations of pre-copy (sec)

Histogram: Rate of page dirtying (pages/sec)

Estimating Downtime

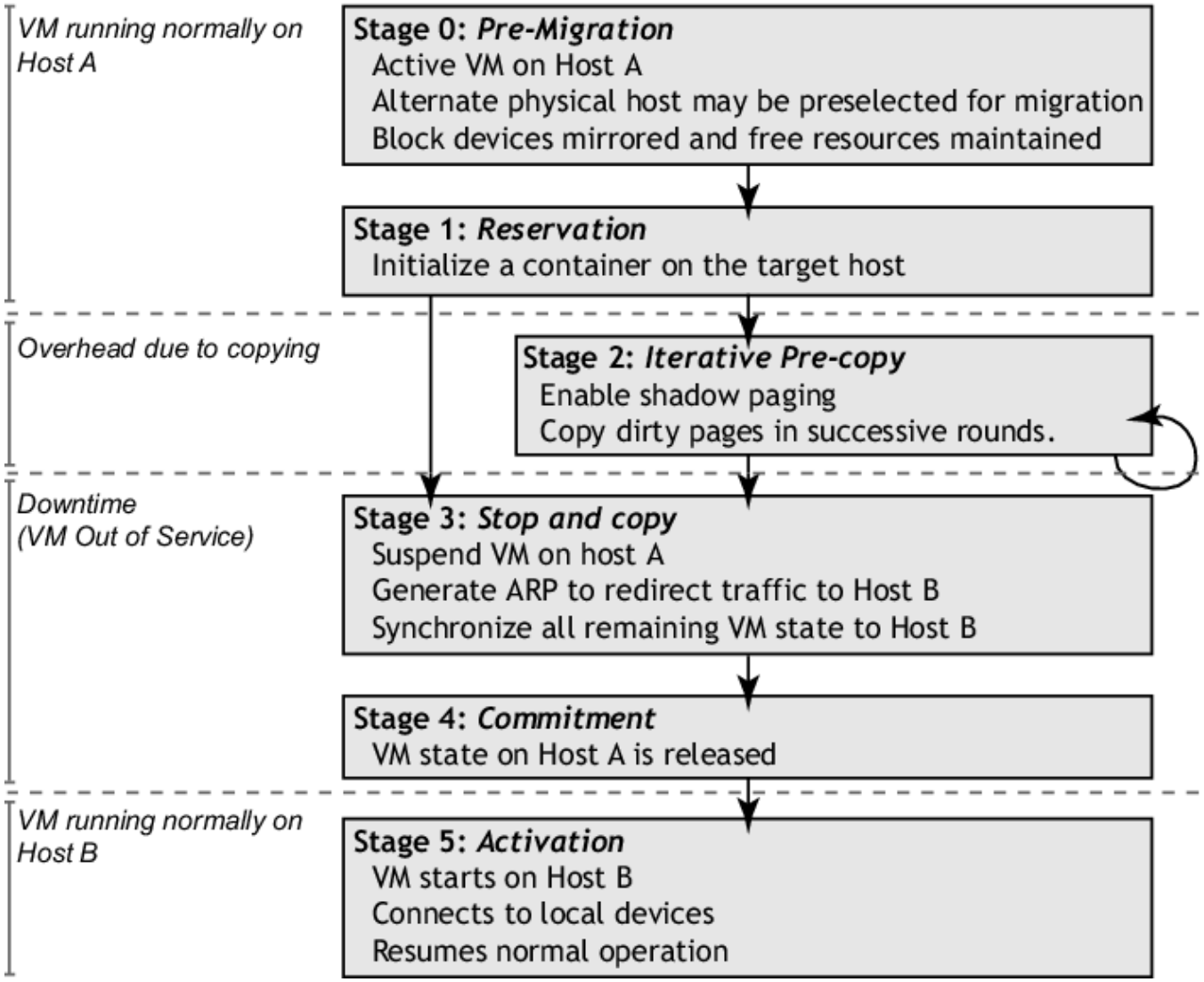
Expected downtime due to last-round memory copy on traced page dirtying of a Quake 3 server



Estimated downtime with 1 iteration of pre-copy (sec)
Estimated downtime with 2 iterations of pre-copy (sec)
Estimated downtime with 3 iterations of pre-copy (sec)
Estimated downtime with 4 iterations of pre-copy (sec)

Histogram: Rate of page dirtying (pages/sec)

Design Overview



Other Issues…

- We only covered issue of live migration partially.
- Lots of other things to consider..
- Rogue Processes -- what if some processes running on OS dirty memory too quickly ?
  - Stun such processes, slow them down temporarily
- What if some memory pages are unused? Do we need to migrate them?
  - No move such pages out of the VM prior to migrating
- Live migration interferes with the network bandwidth and hence SLAs?
  - Adjust network bandwidth at different phases of migration to minimize impact.

- Details of these + other tricks – read the paper!

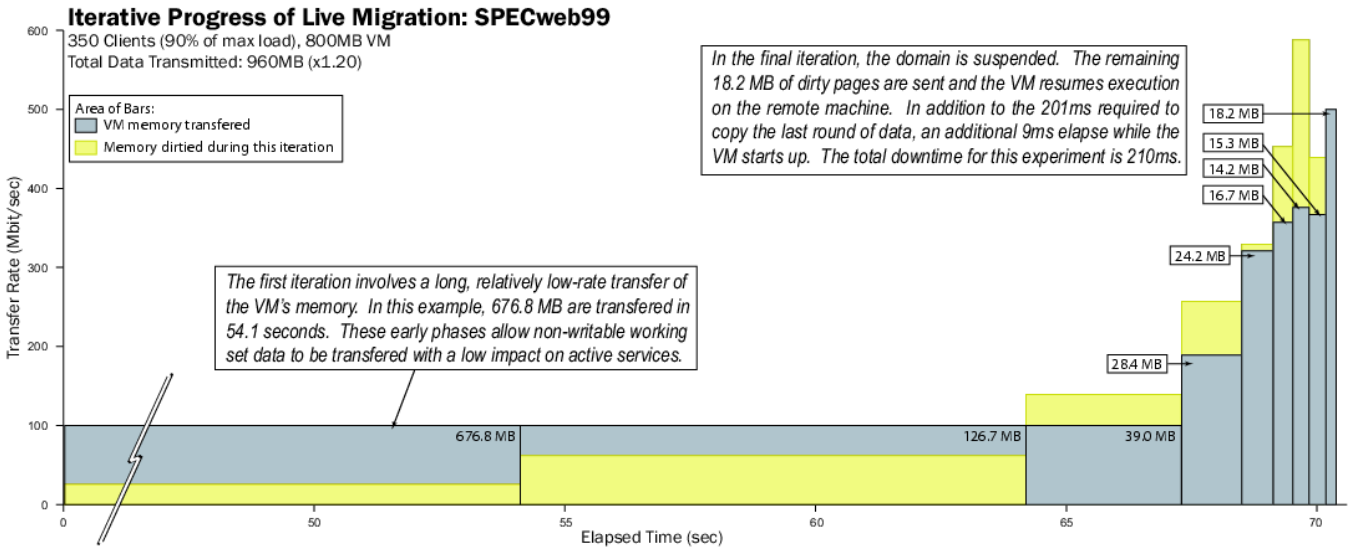
Types of Live Migration

- Managed migration: move the OS without its participation
- Managed migration with some paravirtualization
  - Stun rogue processes that dirty memory too quickly
  - Move unused pages out of the domain so they don't need to be copied
- Self migration: OS participates in the migration (paravirtualization)
  - Harder to get a consistent OS snapshot since the OS is running!

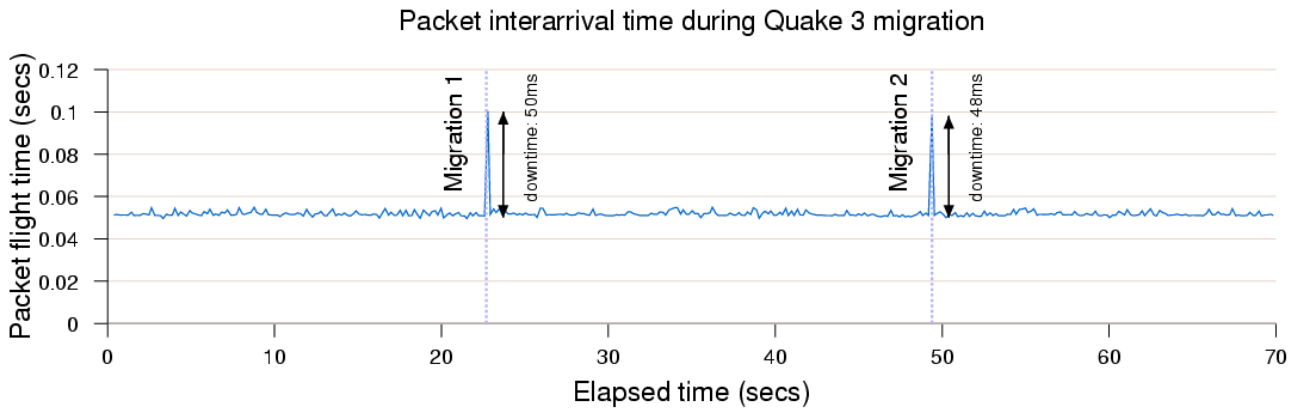
Types of Live Migration

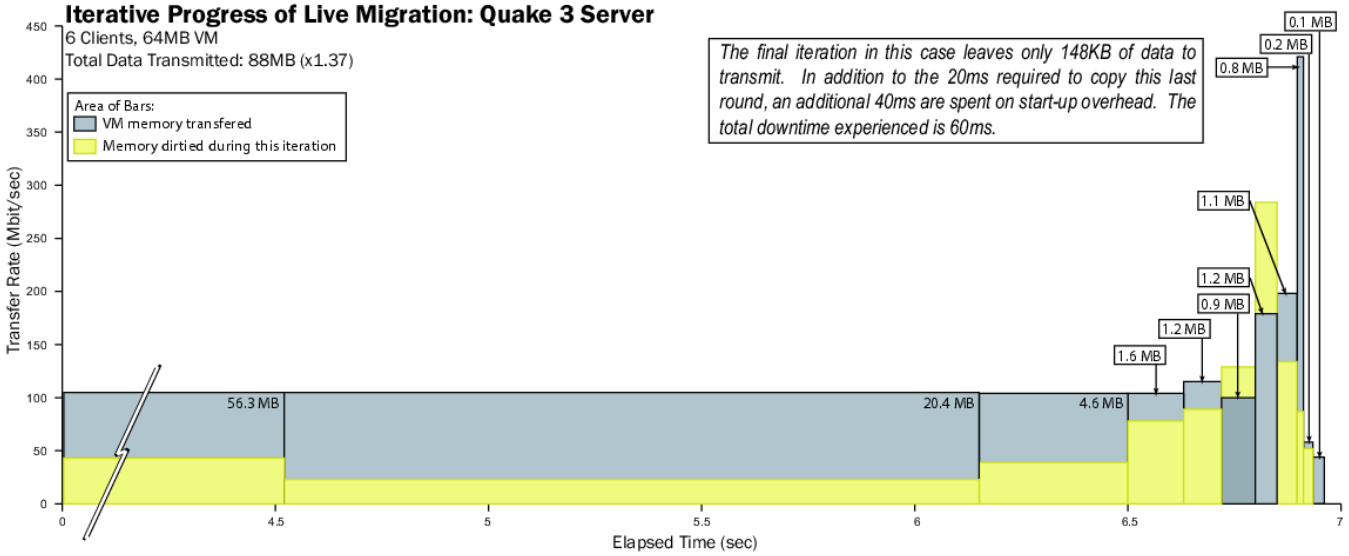
- Managed migration: move the OS without its participation
- Managed migration with some paravirtualization
  - Stun rogue processes that dirty memory too quickly
  - Move unused pages out of the domain so they don't need to be copied
- Self migration: OS participates in the migration (paravirtualization)
  - Harder to get a consistent OS snapshot since the OS is running!

Complex Web Workload: SPECweb99



Low-Latency Server: Quake 3





Last Thoughts on Migration…

- Our focus was on how to efficiently migrate.
- Did not cover, when to migrate and who to migrate, and where to migrate.
- The Sandpiper system automates the decision making behind the ‘when’ and ‘where’ of VM migration
- It collects usage statistics from across the entire network, analyzes them and decides which VMs need to be migrated
- It also identifies the destination of the VMs being migrated and initiates the migration at appropriate time.

The key is appropriate Statistic Collection…

- Coarse grained -- Black box
  - Does not require any changes to individual VMs
  - Collects statistics that are aggregate of all VMs running on a physical machine
  - Aggregate resource usage is determined by monitoring events inside the management VM of the Xen hypervisor
- Finer grained -- Grey box
  - Requires a light weight daemon running inside each VM that is being monitored
  - This allows collection of more fine grained per process level statistics
  - Also enables detection of SLA violations