UNIT III

Cloud Resource virtualization: Virtualization, layering and virtualization, virtual machine monitors, virtual machines, virtualization- full and para, performance and security isolation, hardware support for virtualization, Case Study: Xen, vBlades, Cloud Resource Management and Scheduling: Policies and Mechanisms, Applications of control theory to task scheduling, Stability of a two-level resource allocation architecture, feedback control based on dynamic thresholds, coordination, resource bundling, scheduling algorithms, fair queuing, start time fair queuing, cloud scheduling subject to deadlines, Scheduling Map Reduce applications, Resource management and dynamic application scaling.

Cloud Resource virtualization:

Virtualization:

Virtualization is the "creation of a virtual (rather than actual) version of something, such as a server, a desktop, a storage device, an operating system or network resources". Virtualization is a technique, which allows to share a single physical instance of a resource or an application among multiple customers and organizations. It does by assigning a logical name to a physical storage and providing a pointer to that physical resource when demanded.

- Virtualization is a basic principle of cloud computing – that simplifies some of the resource management tasks.
- Users can run multiple operating systems such as Windows, Linux, on a single physical machine at the same time.
- Virtual machine software can run programs and operating systems, store data, connect to networks, and do other computing functions, and requires maintenance such as updates and system monitoring.

Virtualization simulates the interface to a physical object by:

1. ***Multiplexing.*** Create multiple virtual objects from one instance of a physical object. For example, a processor is multiplexed among a number of processes or threads.
2. ***Aggregation.*** Create one virtual object from multiple physical objects. For example, a number of physical disks are aggregated into a RAID disk.
3. ***Emulation.*** Construct a virtual object from a different type of physical object. For example, a physical disk emulates a random access memory.

4. ***Multiplexing and emulation.*** Examples: Virtual memory with paging multiplexes real memory and disk, and a Virtual address emulates a real address.

Types of Virtualization:

1. Hardware Virtualization.
2. Operating system Virtualization.
3. Server Virtualization.
4. Storage Virtualization.

1) Hardware Virtualization:

When the virtual machine software or virtual machine manager *(VMM) is directly installed on the hardware system* is known as hardware virtualization.

The main job of hypervisor is to control and monitoring the processor, memory and other hardware resources.

After virtualization of hardware system we can install different operating system on it and run different applications on those OS.

Usage:

Hardware virtualization is mainly done for the server platforms, because controlling virtual machines is much easier than controlling a physical server.

2) Operating System Virtualization:

When the virtual machine software or virtual machine manager *(VMM) is installed on the Host operating system* instead of directly on the hardware system is known as operating system virtualization.

Usage: Operating System Virtualization is mainly used for testing the applications on different platforms of OS.

3) Server Virtualization:

When the virtual machine software or virtual machine manager (VMM) is directly installed on the Server system is known as server virtualization.

Usage: Server virtualization is done because a single physical server can be divided into multiple servers on the demand basis and for balancing the load.

4) Storage Virtualization:

Storage virtualization is the process of grouping the physical storage from multiple network storage devices so that it looks like a single storage device.

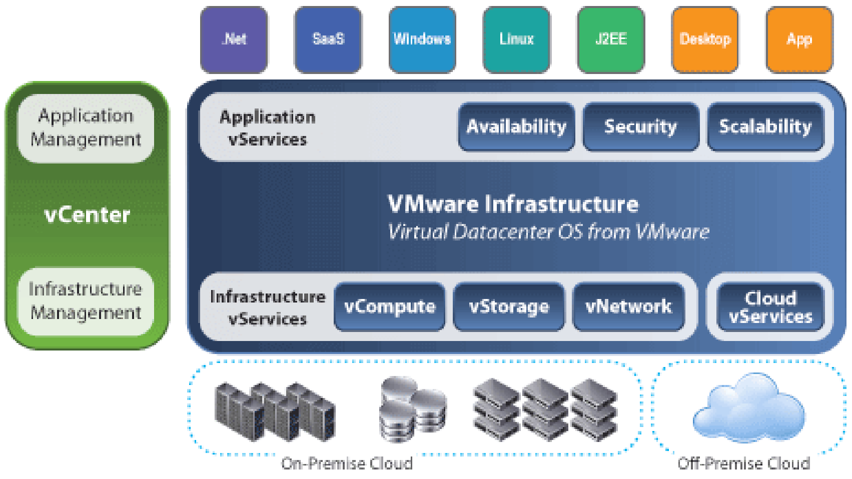
Storage virtualization is also implemented by using software applications.

Usage: Storage virtualization is mainly done for back-up and recovery purposes.

Virtualization is a critical aspect of cloud computing, equally important to the providers and consumers of cloud services, and plays an important role in:

1. System security because it allows isolation of services running on the same hardware.
2. Performance and reliability because it allows applications to migrate from one platform to another.
3. The development and management of services offered by a provider.
4. Performance isolation.

User convenience is a major advantage of a VM architecture over a traditional operating system. For example, a user of the Amazon Web Services (AWS) could submit an Amazon Machine Image (AMI) containing the applications, libraries, data, and associated configuration settings. The user could choose the operating system for the application, then start, terminate, and monitor as many instances of the AMI as needed, using the Web Service APIs and the performance monitoring and management tools provided by the AWS.



The **main usage of Virtualization Technology** is to provide the applications with the standard versions to their cloud users, suppose if the next version of that application is released, then cloud provider has to provide the latest version to their cloud users and practically it is possible because it is more expensive.

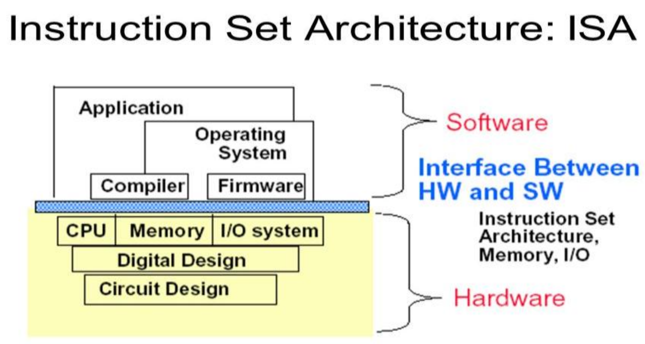
Layering and virtualization

A common approach to managing system complexity is to identify a set of layers with well-defined interfaces among them.

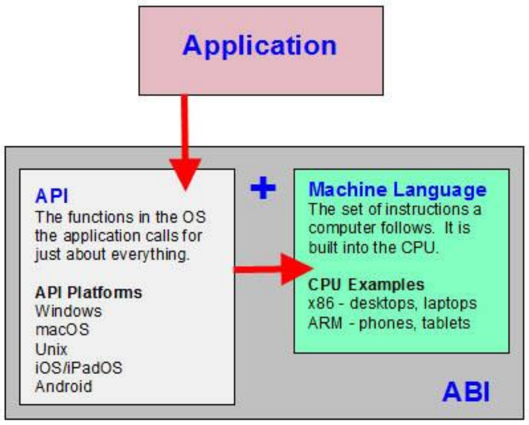
- The interfaces separate different levels of abstraction. Layering minimizes the interactions among the subsystems and simplifies the description of the subsystems.
- Each subsystem is abstracted through its interfaces with the other subsystems. Thus, we are able to design, implement, and modify the individual subsystems independently.

Interfaces:

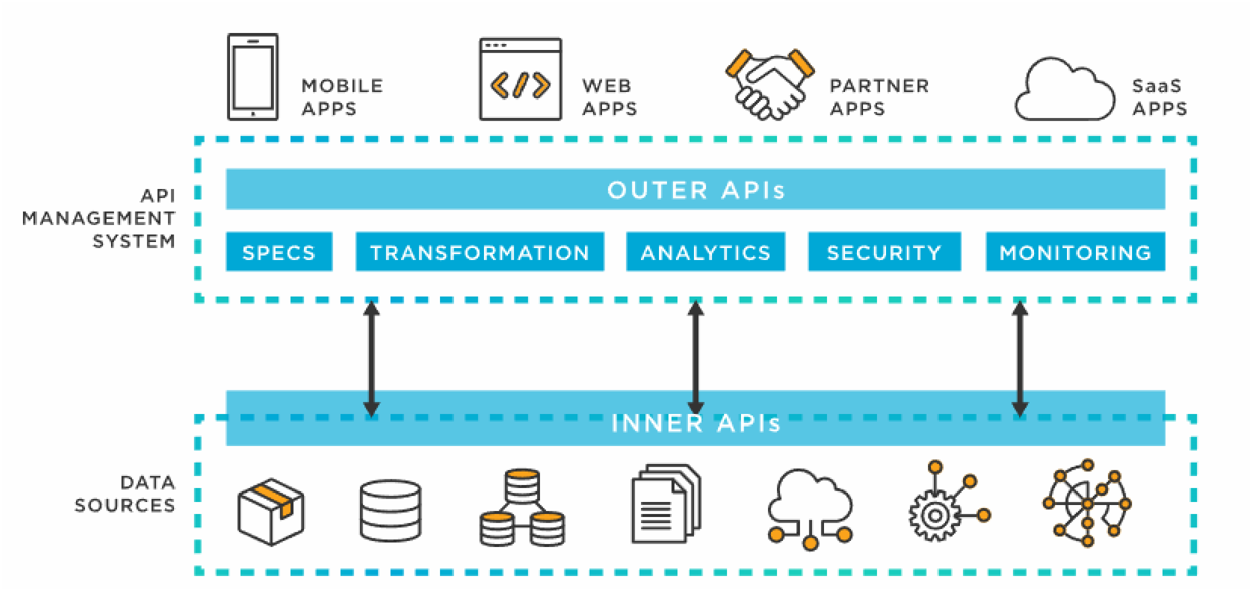
Instruction Set Architecture (ISA) – at the boundary between hardware and software. The instruction set architecture (ISA) defines a processor's set of instructions. For example, the Intel architecture is represented by the x86-32 and x86-64 instruction sets for systems supporting 32-bit addressing and 64-bit addressing, respectively. The hardware supports two execution modes, a privileged, or kernel, mode and a user mode.

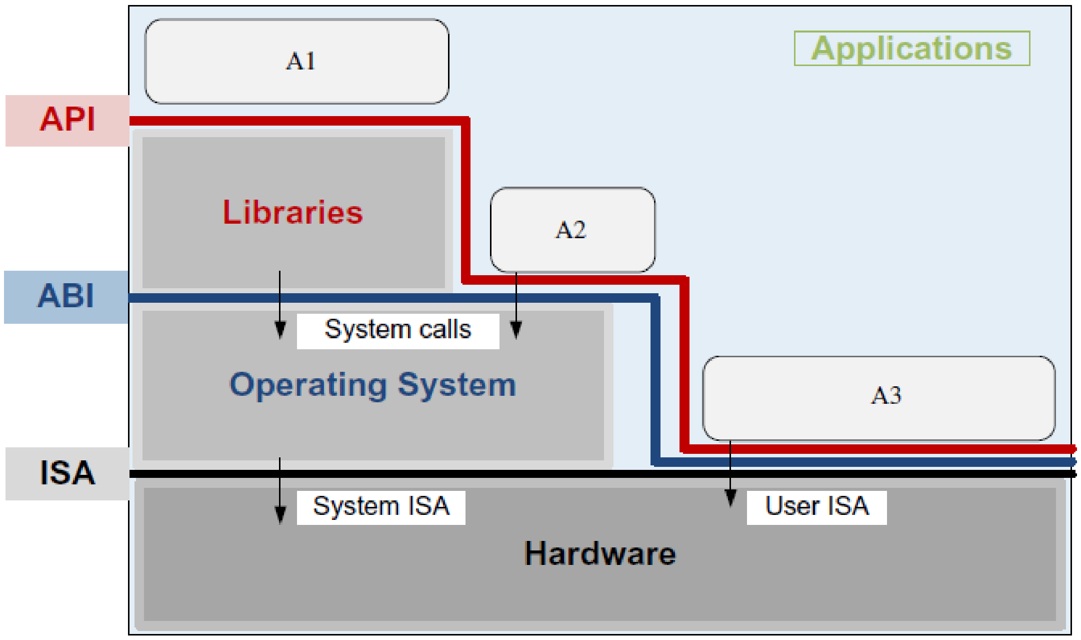


Application Binary Interface (ABI) – allows the ensemble consisting of the application and the library modules to access the hardware; the ABI does not include privileged system instructions, instead it invokes system calls.

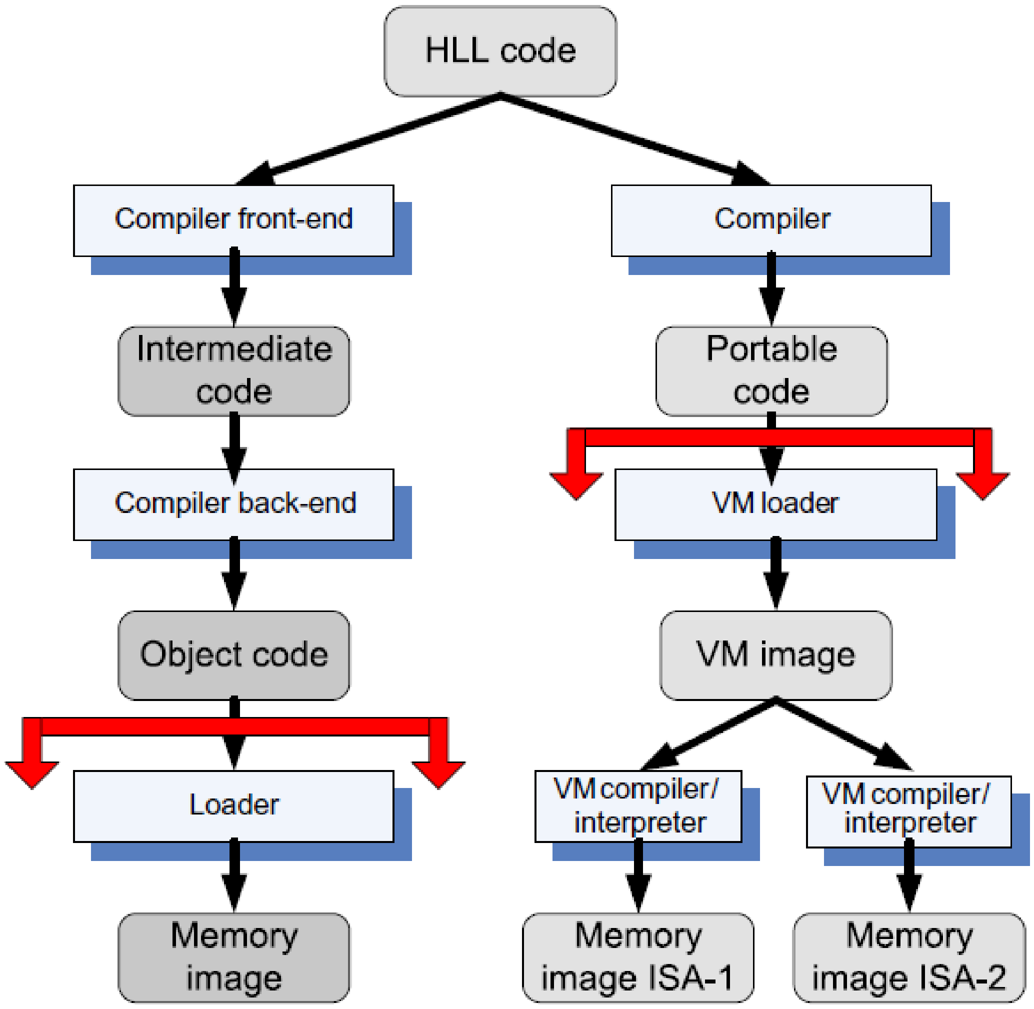

Application
API
The functions in the OS the application calls for just about everything.
API Platforms
Windows
macOS
Unix
iOS/iPadOS
Android
+
Machine Language
The set of instructions a computer follows. It is built into the CPU.
CPU Examples
x86 - desktops, laptops
ARM - phones, tablets
ABI

Application Program Interface (API) - defines the set of instructions the hardware was designed to execute and gives the application access to the ISA; it includes HLL library calls which often invoke system calls.

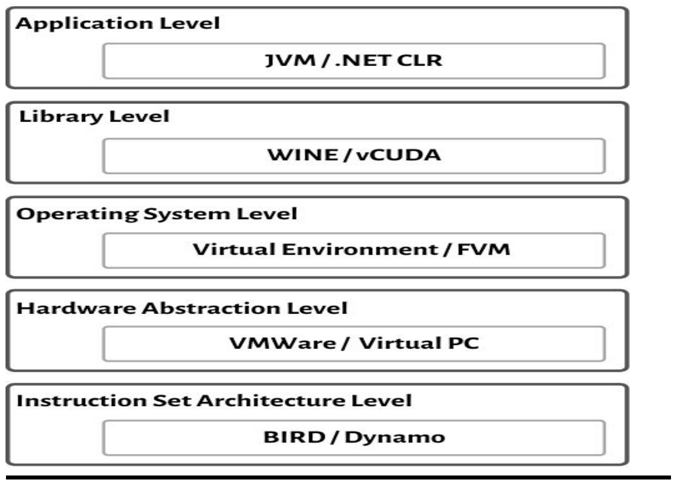




Such code cannot run on a computer with a different ISA or on computers with the same ISA but different operating systems. However, it is possible to compile an HLL program for a VM environment, where portable code is produced and distributed and then converted by binary translators to the ISA of the host system. A dynamic binary translation converts blocks of guest instructions from the portable code to the host instruction and leads to a significant performance improvement as such blocks are cached and reused.



High-level language (HLL) code can be translated for a specific architecture and operating system. HLL code can also be compiled into portable code and then the portable code translated for systems with different ISAs. The code that is shared/distributed is the object code in the first case and the portable code in the second case.



Virtual machine monitors

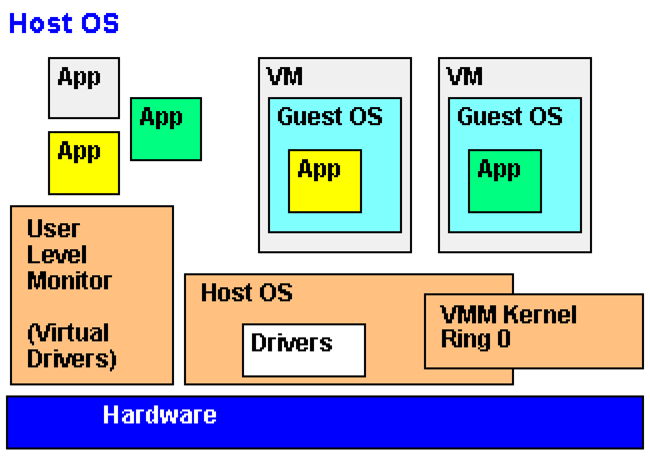
A virtual machine monitor (VMM), also called a hypervisor, is the software that securely partitions the resources of a computer system into one or more virtual machines.

- A guest operating system is an operating system that runs under the control of a VMM rather than directly on the hardware.
- The VMM runs in kernel mode, whereas a guest OS runs in user mode. Sometimes the hardware supports a third mode of execution for the guest OS.

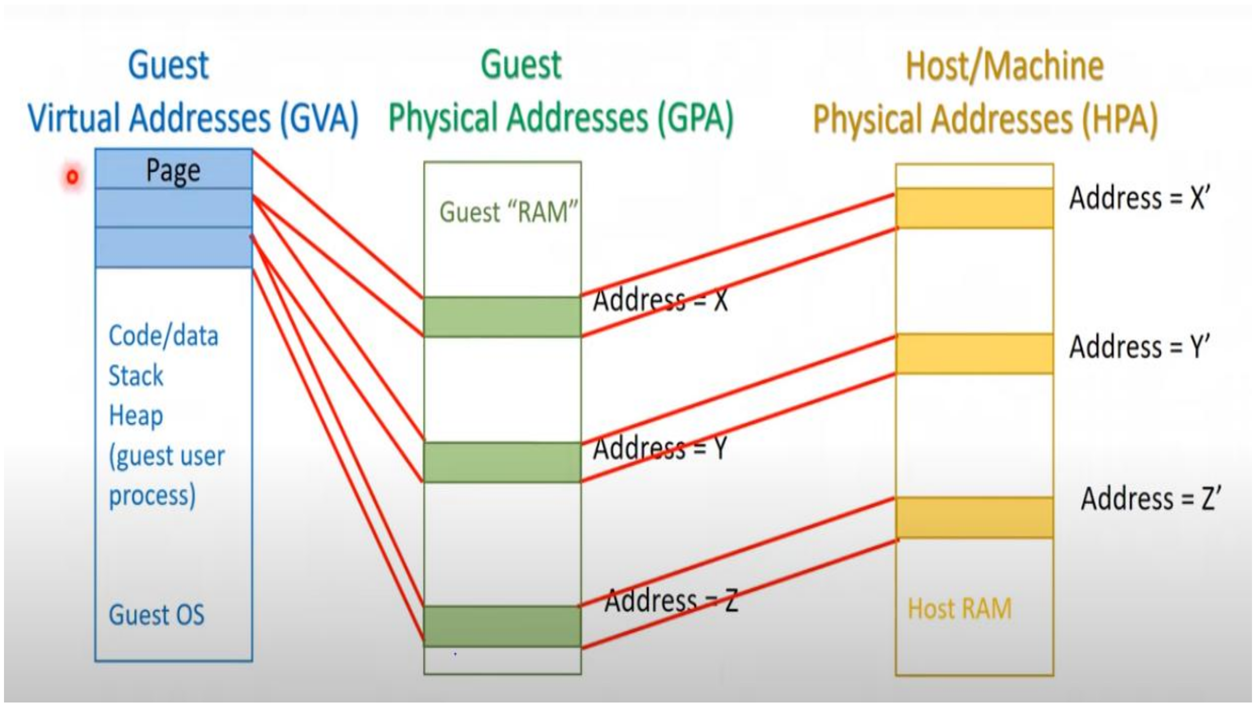
VMMs allow several operating systems to run concurrently on a single hardware platform; at the same time, VMMs enforce isolation among these systems, thus enhancing security. A VMM controls how the guest operating system uses the hardware resources. The events occurring in one VM do not affect any other VM running under the same VMM. At the same time, the VMM enables:

- Multiple services to share the same platform.
- The movement of a server from one platform to another, the so-called live migration.
- System modification while maintaining backward compatibility with the original system.

When a guest OS attempts to execute a privileged instruction, the VMM traps the operation and enforces the correctness and safety of the operation. The VMM guarantees the isolation of the individual VMs, and thus ensures security and encapsulation, a major concern in cloud computing.

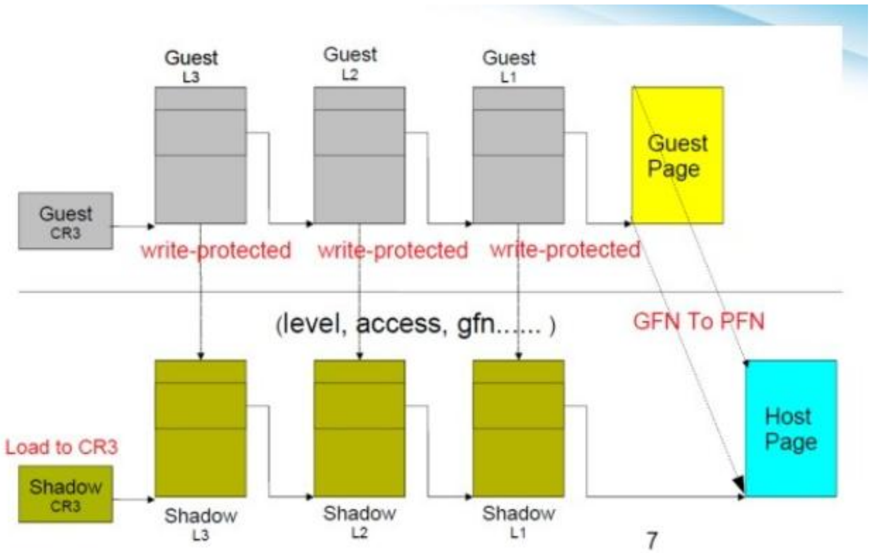


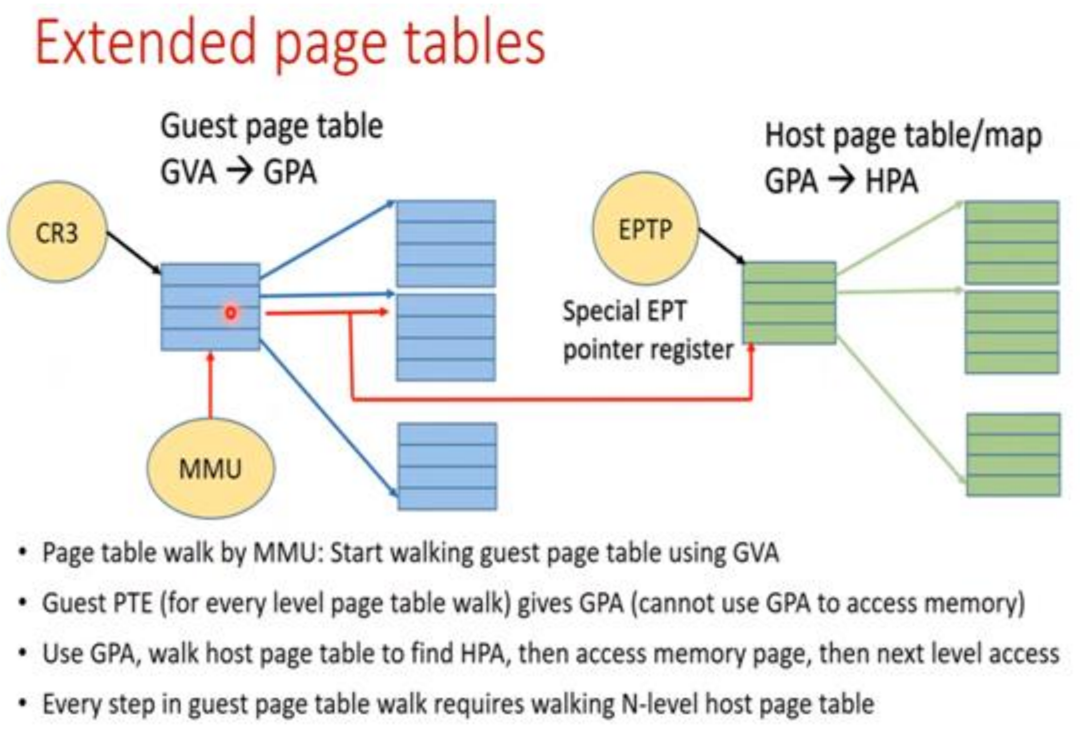
A VMM virtualizes the CPU and memory. For example, the VMM traps interrupts and dispatches them to the individual guest operating systems. If a guest OS disables interrupts, the VMM buffers such interrupts until the guest OS enables them.



The VMM maintains a shadow page table for each guest OS and replicates any modification made by the guest OS in its own shadow page table. This shadow page table points to the actual page frame and is used by the hardware component called the memory management unit (MMU) for dynamic address translation. VMM manages the backend operation of these VMs by allocating the necessary computing, memory and other input/output (I/O) resources.

The shadow page table is **a data structure that is actively maintained and re-filled by the VMM**. The shadow page table mirrors what the guest is doing in terms of its own page tables and in terms of what the VMM translates the guest physical address to the host physical address.





Extended page tables

- Page table walk by MMU: Start walking guest page table using GVA
- Guest PTE (for every level page table walk) gives GPA (cannot use GPA to access memory)
- Use GPA, walk host page table to find HPA, then access memory page, then next level access
- Every step in guest page table walk requires walking N-level host page table

Virtual Machine (VM)

A ***virtual machine (VM)*** is an isolated environment that appears to be a whole computer but actually only has access to a portion of the computer resources. Each VM appears to be running on the bare hardware, giving the appearance of multiple instances of the same computer, though all are supported by a single physical system.

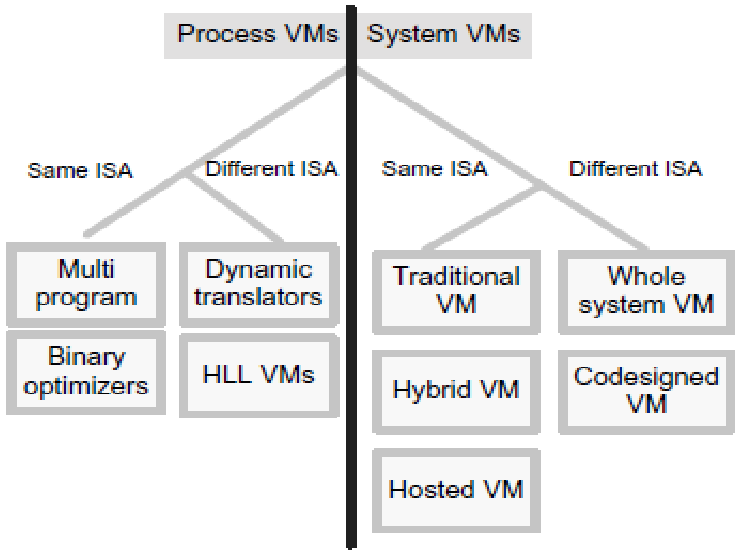
Two types of VM:

Process VMs and System VMs:

A process VM is a virtual platform created for an individual process and destroyed once the process terminates. Virtually all operating systems provide a process VM for each one of the applications running, but the more interesting process VMs are those that support binaries compiled on a different instruction set.

A system VM supports an operating system together with many user processes. When the VM runs under the control of a normal OS and provides a platform-independent host for a single application, we have an application virtual machine (e.g., Java Virtual Machine [JVM]).

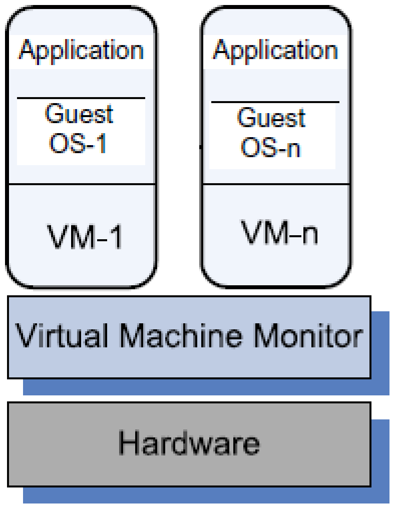
A System virtual machine provides a complete system; each VM can run its own OS, which in turn can run multiple applications. Systems such as Linux Vserver [214], OpenVZ (Open VirtualiZation), FreeBSD Jails , and Solaris Zones , based on Linux, FreeBSD, and Solaris.

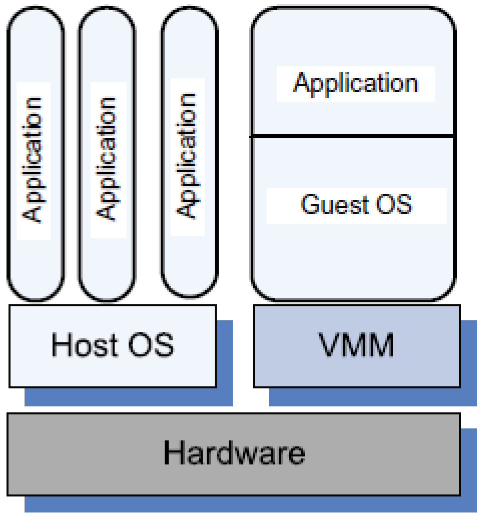


VMM allows several virtual machines to share a system. Several organizations of the software stack are possible:

Traditional VM also called a "bare metal" VMM. A thin software layer that runs directly on the host machine hardware; its main advantage is performance. Examples: VMWare ESX, ESXi Servers, Xen, OS370, and Denali.

Hybrid VM. The VMMshares the hardware with the existing OS [see Figure 5.3(c)]. Example: VMWare Workstation.





Traditional VM **Hybrid VM**

Hosted VM. The VM runs on top of an existing OS. The main advantage of this approach is that the VM is easier to build and install. Another advantage of this solution is that the VMM could use several components of the host OS, such as the scheduler, the pager, and the I/O drivers, rather than providing its own.

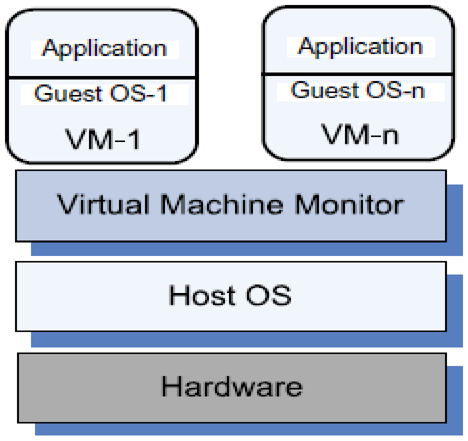


Table 5.1 A nonexhaustive inventory of system virtual machines. The host ISA refers to the instruction set of the hardware; the guest ISA refers to the instruction set supported by the virtual machine. The VM could run under a host OS, directly on the hardware, or under a VMM. The guest OS is the operating system running under the control of a VM, which in turn may run under the control of the VMM.

Name	Host ISA	Guest ISA	Host OS	Guest OS	Company
Integrity VM	*x86-64*	*x86-64*	HP-*Unix*	*Linux, Windows HP Unix*	HP
Power VM	Power	Power	No host OS	*Linux*, AIX	IBM
z/VM	z-ISA	z-ISA	No host OS	*Linux* on z-ISA	IBM
Lynx Secure	*x86*	*x86*	No host OS	*Linux, Windows*	LinuxWorks
Hyper-V Server	*x86-64*	*x86-64*	*Windows*	*Windows*	Microsoft
Oracle VM	*x86, x86-64*	*x86, x86-64*	No host OS	*Linux, Windows*	Oracle
RTS Hypervisor	*x86*	*x86*	No host OS	*Linux, Windows*	Real Time Systems
SUN xVM	*x86*, SPARC	same as host	No host OS	*Linux, Windows*	SUN
VMware EX Server	*x86, x86-64*	*x86, x86-64*	No host OS	*Linux, Windows, Solaris, FreeBSD*	VMware
VMware Fusion	*x86, x86-64*	*x86, x86-64*	*Mac OS x86*	*Linux, Windows, Solaris, FreeBSD*	VMware
VMware Server	*x86, x86-64*	*x86, x86-64*	*Linux, Windows*	*Linux, Windows, Solaris, FreeBSD*	VMware
VMware Workstation	*x86, x86-64*	*x86, x86-64*	*Linux, Windows*	*Linux, Windows, Solaris, FreeBSD*	VMware
VMware Player	*x86, x86-64*	*x86, x86-64*	*Linux, Windows*	*Linux, Windows, Solaris, FreeBSD*	VMware
Denali	*x86*	*x86*	Denali	*ILVACO, NetBSD*	University of Washington
Xen	*x86, x86-64*	*x86, x86-64*	*Linux Solaris*	*Linux, Solaris NetBSD*	University of Cambridge

Full virtualization and Para virtualization

VMware can virtualized any x86 operating system using a combination of binary translation and direct execution techniques.

Binary translation, The VMM monitors the execution of guest operating systems; nonvirtualizable instructions executed by a guest operating system are replaced with other instructions.

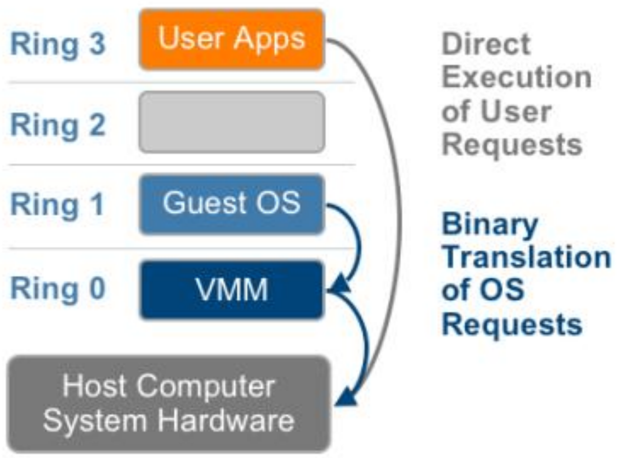
Direct execution techniques, User level code is directly executed on the processor for high performance virtualization. Each virtual machine monitor provides each Virtual Machine with all the services of the physical system, including a virtual BIOS.

1.Full virtualization, in which each virtual machine runs on an exact copy of the actual hardware.

2.Paravirtualization, in which each virtual machine runs on a slightly modified copy of the actual hardware.

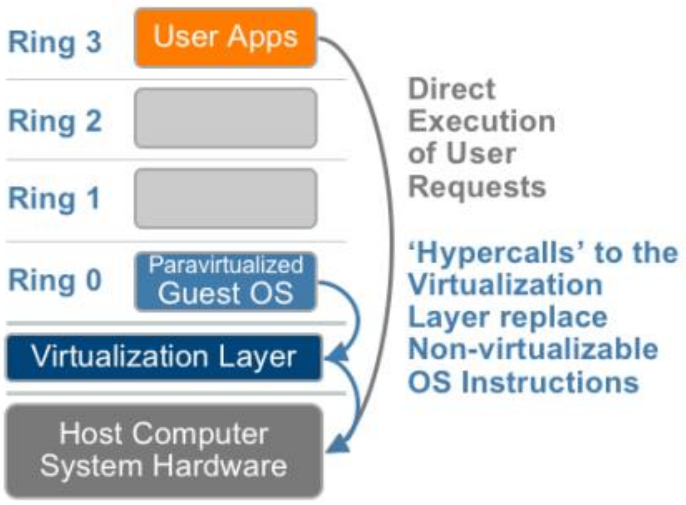
1. Full Virtualization:

Full Virtualization was introduced by IBM in the year 1966. It is the first software solution for server virtualization and uses binary translation and direct approach techniques. In full virtualization, guest OS is completely isolated by the virtual machine from the virtualization layer and hardware. Microsoft and Parallels systems are examples of full virtualization.


Ring 3
User Apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host Computer System Hardware
Direct Execution of User Requests
Binary Translation of OS Requests

2. Paravirtualization:

Paravirtualization is the category of CPU virtualization which uses hypercalls for operations to handle instructions at compile time. In paravirtualization, guest OS is not completely isolated but it is partially isolated by the virtual machine from the virtualization layer and hardware. VMware and Xen are some examples of paravirtualization.


Ring 3
User Apps
Ring 2
Ring 1
Ring 0
Paravirtualized Guest OS
Virtualization Layer
Host Computer System Hardware
Direct Execution of User Requests
'Hypercalls' to the Virtualization Layer replace Non-virtualizable OS Instructions

S.No.	Full Virtualization	Paravirtualization
1.	In Full virtualization, virtual machines permit the execution of the instructions with the running of unmodified OS in an entirely isolated way.	In paravirtualization, a virtual machine does not implement full isolation of OS but rather provides a different API which is utilized when OS is subjected to alteration.
2.	Full Virtualization is less secure.	While the Paravirtualization is more secure than the Full Virtualization.
3.	Full Virtualization uses binary translation and a direct approach as a technique for operations.	While Paravirtualization uses hypercalls at compile time for operations.
4.	Full Virtualization is slow than paravirtualization in operation.	Paravirtualization is faster in operation as compared to full virtualization.
5.	Full Virtualization is more portable and compatible.	Paravirtualization is less portable and compatible.
6.	Examples of full virtualization are Microsoft and Parallels systems.	Examples of paravirtualization are Microsoft Hyper-V, Citrix Xen, etc.

Performance and security isolation

Performance isolation is a critical condition for quality-of-service (QoS) guarantees in shared computing environments.

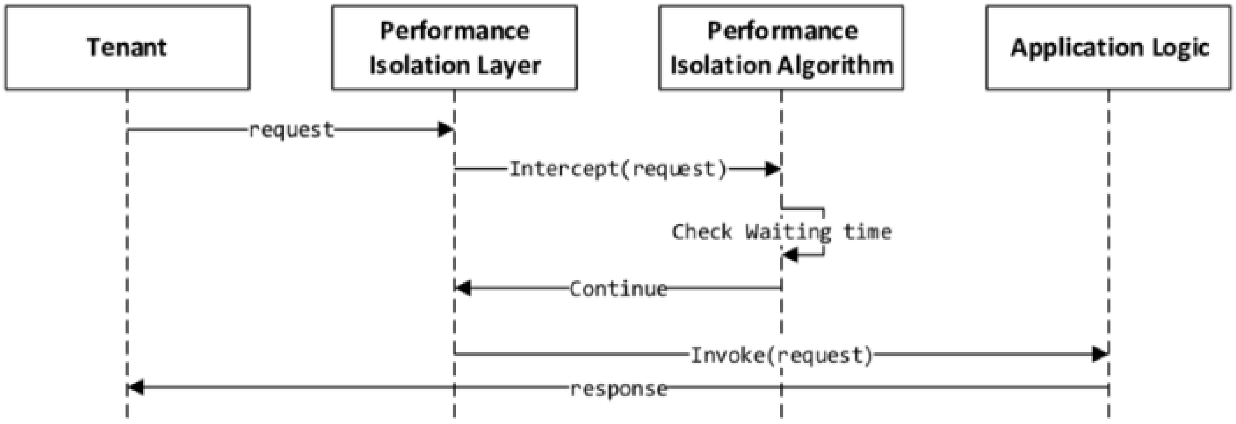
Indeed, if the run-time behavior of an application is affected by other applications running concurrently and, thus, is competing for CPU cycles, cache, main memory, and disk and network access, it is rather difficult to predict the completion time. Moreover, it is equally difficult to optimize the application.

Processor virtualization presents multiple copies of the same processor or core on multicore systems. The code is executed directly by the hardware, whereas processor emulation presents a model of another hardware system in which instructions are "emulated" in software more slowly than virtualization.

A VMM is a much simpler and better specified system than a traditional operating system. For example, the Xen VMM discussed in Section 5.8 has approximately 60,000 lines of code, whereas the Denali VMM has only about half that, or 30,000 lines of code.

The security vulnerability of VMMs is considerably reduced because the systems expose a much smaller number of privileged functions.

For example, the Xen VMM can be accessed through 28 hypercalls, whereas a standard Linux allows hundreds (e.g., Linux 2.6.11 allows 289 system calls).



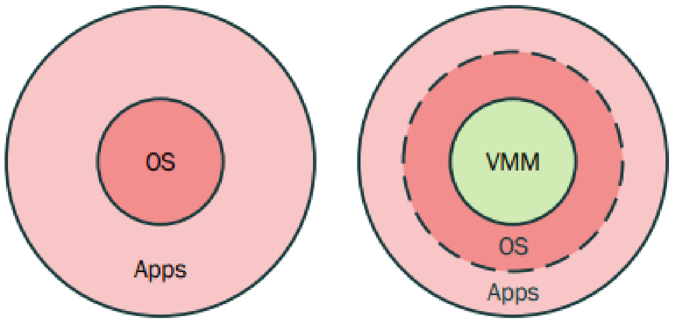
Hardware support for virtualization

- Intel and AMD started work on the first-generation virtualization extensions of the *x86 architecture.*
- The VMware Workstation is a VM software suite for x86 and x86-64 computers.
- In 2005 Intel released two Pentium 4 models supporting VT-x, and in 2006 AMD announced Pacifica and then several Athlon 64 models.

Software solutions at that time addressed some of the challenges, but hardware solutions could improve not only performance but also security and, at the same time, simplify the software systems.

The problems faced by virtualization of the x86 architecture:

Ring deprivileging. This means that a VMM forces the guest software, the operating system, and the applications to run at a privilege level greater than 0. Recall that the x86 architecture provides four protection rings at levels 0–3. Two solutions are then possible: (a) The (0/1/3) mode, in which the VMM, the OS, and the application run at privilege levels 0, 1, and 3, respectively; or (b) the (0,3,3) mode, in which the VMM, a guest OS, and applications run at privilege levels 0, 3, and 3, respectively.



Ring aliasing. Problems created when a guest OS is forced to run at a privilege level other than that it was originally designed for. For example, when the CR register is PUSHed, the current privilege level is also stored on the stack

Address space compression. A VMM uses parts of the guest address space to store several system data structures, such as the interrupt-descriptor table and the global-descriptor table.

Nonfaulting access to privileged state. Several instructions, LGDT, SIDT, SLDT, and LTR that load the registers GDTR, IDTR, LDTR, and TR, can only be executed by software running at privilege level 0, because these instructions point to data structures that control the CPU operation.

Guest system calls. Two instructions, SYSENTER and SYSEXIT, support low-latency system calls. The first causes a transition to privilege level 0, whereas the second causes a transition from privilege level 0 and fails if executed at a level higher than 0.

Interrupt virtualization. In response to a physical interrupt, the VMM generates a "virtual interrupt" and delivers it later to the target guest OS. But every OS has the ability to mask interrupts.

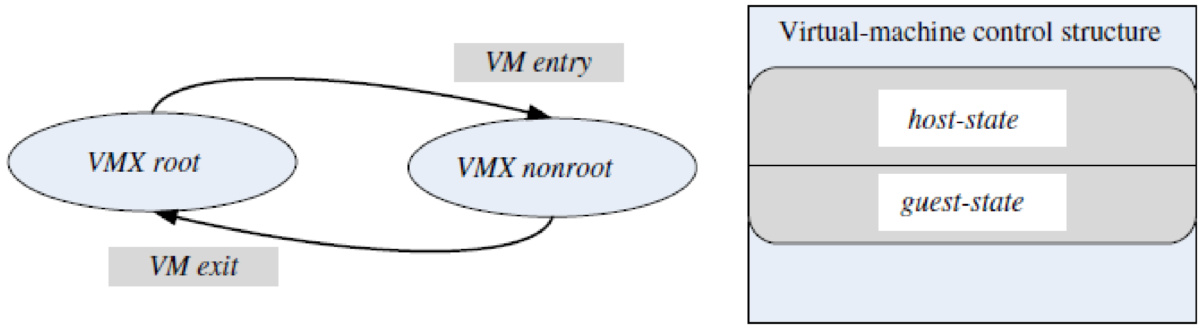
Access to hidden state. Elements of the system state (e.g., descriptor caches for segment registers) are hidden; there is no mechanism for saving and restoring the hidden components when there is a context switch from one VM to another.

Ring compression. Paging and segmentation are the two mechanisms to protect VMM code from being overwritten by a guest OS and applications. Systems running in 64-bit mode can only use paging, but paging does not distinguish among privilege levels 0, 1, and 2, so the guest OS must run at privilege level 3, the so-called (0/3/3) mode. Privilege levels 1 and 2 cannot be used; thus the name ring compression.

Frequent access to privileged resources increases VMM overhead. The task-priority register (TPR) is frequently used by a guest OS.

A major architectural enhancement provided by the VT-x is the support for two modes of operations and a new data structure called the virtual machine control structure (VMCS), including host-state and guest-state areas.

- VMX root. Intended for VMM operations and very close to the x86 without VT-x.
- VMX nonroot. Intended to support a VM.

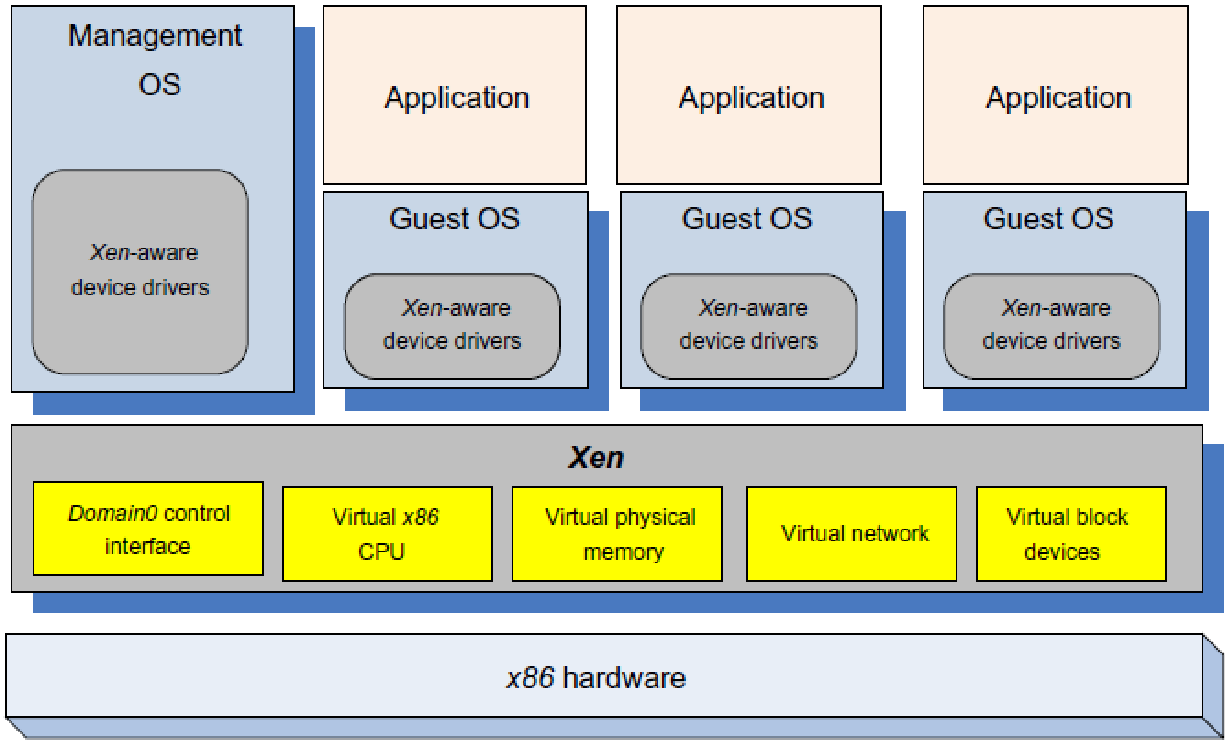


When executing a VM entry operation, the processor state is loaded from the guest-state of the VM scheduled to run; then the control is transferred from theVMMto the VM.AVMexit saves the processor state in the guest-state area of the running VM; then it loads the processor state from the host-state area and finally transfers control to the VMM. Note that all VM exit operations use a common entry point to the VMM.

Xen (VMM based on paravirtualization)

Xen is a VMM or hypervisor developed by the Computing Laboratory at the University of Cambridge, United Kingdom, in 2003. The hypervisor supports hardware level virtualization on bare metal devices like CPU, memory, disk and network interfaces.

- The hypervisor software sits directly between the physical hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor.
- Xen is an open source hypervisor program developed by Cambridge University. Xen is a microkernel hypervisor, which separates the policy from the mechanism.
- The goal of the Cambridge group - design a VMM capable of scaling to about 100 VMs running standard applications and services without any modifications to the Application Binary Interface (ABI).
- The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0. Figure 2.9 shows architecture of Xen hypervisor. Xen does not include any device drivers natively. It just provides a mechanism by which guests OS can have direct access to the physical devices.
- As a result, the size of the Xen hypervisor is kept rather small. Xen provides a virtual environment located between the hardware and the OS. The core components of a Xen system are the hypervisor, kernel, and applications.
- The organization of the three components is important. Like other virtualization systems, many guest OSes can run on top of the hypervisor. However, not all guest OSes are created equal, and one in particular controls the others.
- The guest OS, which has control ability, is called Domain 0, and the others are called Domain U (a user domain).
- Domain 0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being available.
- Domain 0 is designed to access hardware directly and manage devices. Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).
- Domain 0, behaving as a VMM, allows users to create, copy, save, read, modify, share, migrate and roll back VMs as easily as manipulating a file, which flexibly provides tremendous benefits for users.



Xen implementation on x86 architecture:

- Xen runs at privilege Level 0, the guest OS at Level 1, and applications at Level 3.
- The x86 architecture does not support either the tagging of TLBv entries or the software management of the TLB. Thus, address space switching, when the VMM activates a different OS, requires a complete TLB flush; this has a negative impact on the performance.
- Solution - load Xen in a 64 MB segment at the top of each addressv space and delegate the management of hardware page tables to the guest OS with minimal intervention from Xen. This region is not accessible or re-mappable by the guest OS.
- Xen schedules individual domains using the Borrowed Virtual Timev (BVT) scheduling algorithm. A guest OS must register with Xen a description table with thev addresses of exception handlers for validation.

Dom0 components:

XenStore – a Dom0 process.

- Supports a system-wide registry and naming service.

- Implemented as a hierarchical key-value storage.
- A watch function informs listeners of changes of the key in storage they have subscribed to. Communicates with guest VMs via shared memory using Dom0 privileges.

Toolstack - responsible for creating, destroying, and managing the ν resources and privileges of VMs.

- To create a new VM, a user provides a configuration file describing memory and CPU allocations and device configurations.
- Toolstack parses this file and writes this information in XenStore.
- Takes advantage of Dom0 privileges to map guest memory, to load a kernel and virtual BIOS and to set up initial communication channels with XenStore and with the virtual console when a new VM is created.

Xen abstractions for networking and I/O:

- Each domain has one or more Virtual Network Interfaces (VIFs) which support the functionality of a network interface card. A VIF is attached to a Virtual Firewall-Router (VFR).
- Split drivers have a front-end in the DomU and the back-end in Dom0; the two communicate via a ring in shared memory.
- Ring - a circular queue of descriptors allocated by a domain and accessible within Xen. Descriptors do not contain data, the data buffers are allocated off-band by the guest OS.
- Two rings of buffer descriptors, one for packet sending and one for packet receiving, are supported.

To transmit a packet:

- a guest OS enqueues a buffer descriptor to the send ring,
- then Xen copies the descriptor and checks safety,
- copies only the packet header, not the payload, and
- executes the matching rules.

Xen zero-copy semantics for data transfer using I/O rings:

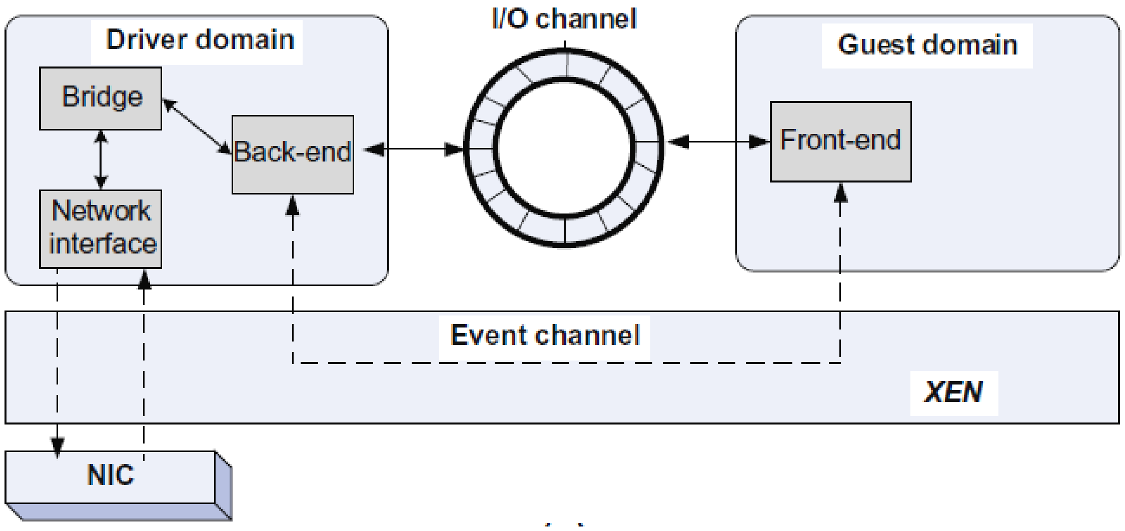


Fig. The communication between a guest domain and the driver domain over an I/O and an event channel; NIC is the Network Interface Controller.

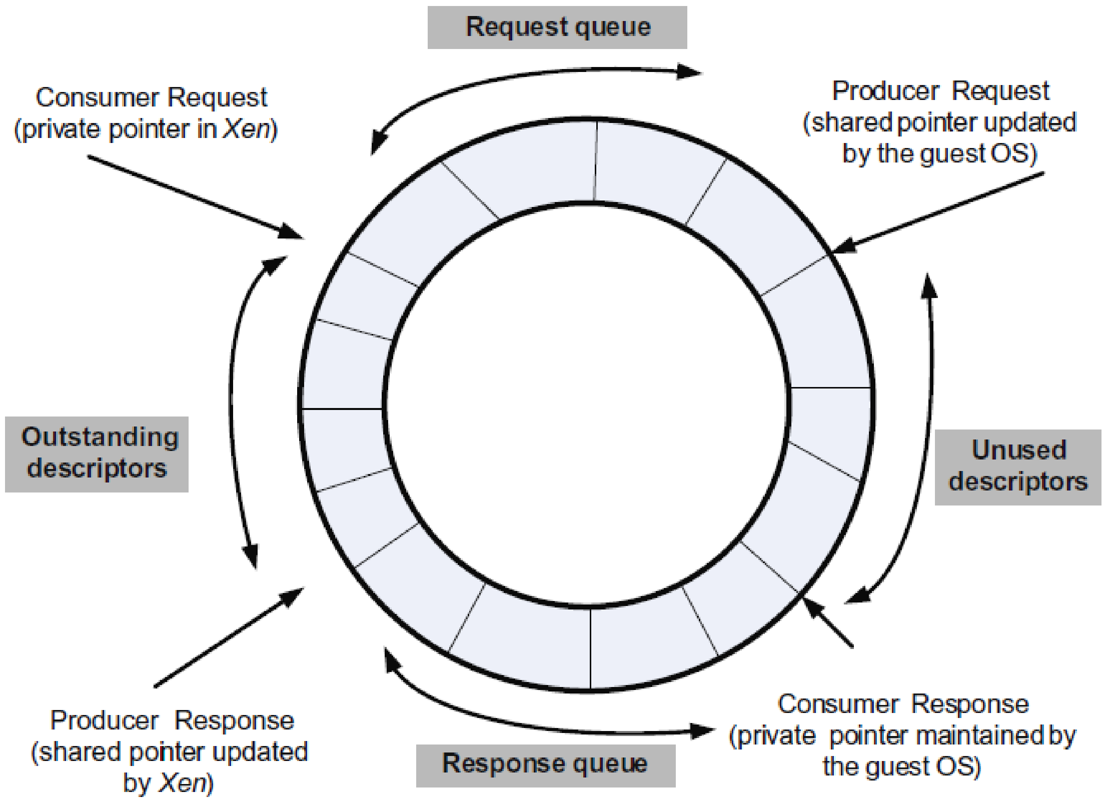

Request queue
Consumer Request (private pointer in Xen)
Producer Request (shared pointer updated by the guest OS)
Outstanding descriptors
Unused descriptors
Producer Response (shared pointer updated by Xen)
Response queue
Consumer Response (private pointer maintained by the guest OS)

Fig.The circular ring of buffers.

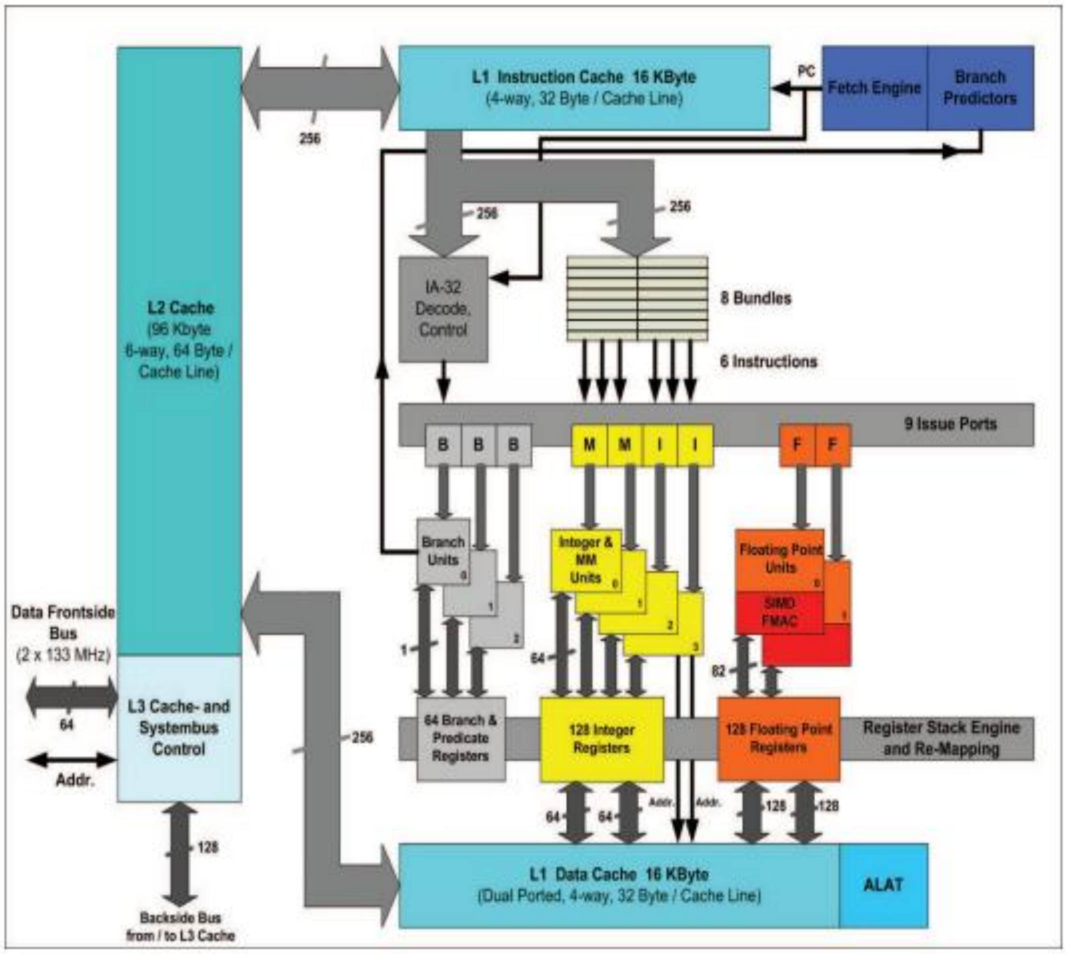
vBlades - paravirtualization targeting a x86-64 Itanium processor

To understand the impact of the computer architecture on the ability to virtualize efficiently a given architecture we discuss some of the findings of the vBlades project at HPLaboratories.

The goal of the vBlades project was to create a VMM for the Itanium family of x86-64 Intel processors47 capable of supporting the execution of multiple operating systems in isolated protection domains with security and privacy enforced by the hardware.

Itanium was selected because of its multiple functional units and multi-threading support. The VMM was also expected to support optimal server utilization and allow comprehensive measurement and monitoring for detailed performance analysis.

Fully virtualizable the ISA of a processor must conform to a set of requirements but, unfortunately, the x86-64 architecture does not meet these requirements and this made the vBlades project more challenging.


L1 Instruction Cache 16 KByte
(4-way, 32 Byte / Cache Line)
PC
Fetch Engine
Branch Predictors
256
256
256
IA-32 Decode, Control
8 Bundles
6 Instructions
L2 Cache (96 Kbyte 6-way, 64 Byte / Cache Line)
9 Issue Ports
B B B
M M I I
F F
Branch Units
Integer & MM Units
Floating Point Units
SIMD FMAC
Data Frontside Bus (2 x 133 MHz)
64
Addr.
L3 Cache- and Systembus Control
64 Branch & Predicate Registers
128 Integer Registers
128 Floating Point Registers
Register Stack Engine and Re-Mapping
256
82
64
128
Backside Bus from / to L3 Cache
L1 Data Cache 16 KByte
(Dual Ported, 4-way, 32 Byte / Cache Line)
ALAT

We first review the features of the Itanium processor important for virtualization and start with the observation that the hardware supports four privilege rings, PL0, PL1, PL2, and PL3.

Privileged instructions can only be executed by the kernel running at PL0, while applications run at level PL3 and can only execute non-privileged instructions; PL2 and PL4 rings are generally not used. The VMM uses ring compression and runs itself at PL0 and PL1 while forcing a guest OS to run at PL2.

A first problem called privilege leaking is that several non-privileged instructions allow an application to determine the Current Privilege Level (CPL); thus, a guest OS may not accept to boot or run or may itself attempt to make use of all four privilege rings.

The Itanium processor supports isolation of the address spaces of different processes with eight privileged region registers; the Processor Abstraction Layer (PAL) firmware allows the caller to set the values in the region register.

The VMM intercepts the privileged instruction issued by the guest OS to its PAL and partitions the set of address spaces among the guests OS to ensure isolation.

Cloud Resource Management and Scheduling

Resources management is a core function of any man-made system thus, an important element of cloud computing. A cloud is a complex system with a very large number of shared resources subject to unpredictable requests and affected by external events it cannot control.

Scheduling in a computing system is deciding how to allocate resources of a system, such as CPU cycles, memory, secondary storage space, I/O and network bandwidth, between users and tasks.

Policies and mechanisms for resource allocation.
- Policy-> principles guiding decisions.
- Mechanisms-> the means to implement policies.

Centralized control cannot provide adequate solutions to the host of cloud management policies that have to be enforced.

Cloud resource management policies can be loosely grouped into five classes:

1. Admission control.
2. Capacity allocation.
3. Load balancing.
4. Energy optimization.
5. Quality of service (QoS) guarantees.

1. **Admission control** - The explicit goal of an admission control policy is to prevent the system from accepting workloads in violation of high-level system policies; for example, a system may not accept an additional workload that would prevent it from completing work already in progress or contracted.

2. Capacity allocation - Capacity allocation means to allocate resources for individual instances; an instance is an activation of a service.

3. Load balancing - distribute the workload evenly among the servers. Load balancing and energy optimization can be done locally, but global load-balancing and energy optimization policies encounter the same difficulties as the one.

4. Energy optimization - minimization of energy consumption.

5. Quality of service (QoS) guarantees - ability to satisfy timing or other conditions specified by a Service Level Agreement.

Mechanisms for the implementation of resource management policies:

- Control theory - uses the feedback to guarantee system stability and predict transient behavior.
- Machine learning - does not need a performance model of the system.
- Utility-based - require a performance model and a mechanism to correlate user-level performance with cost.
- Market-oriented/economic - do not require a model of the system, e.g., combinatorial auctions for bundles of resources.

Applications of control theory to task scheduling on a cloud

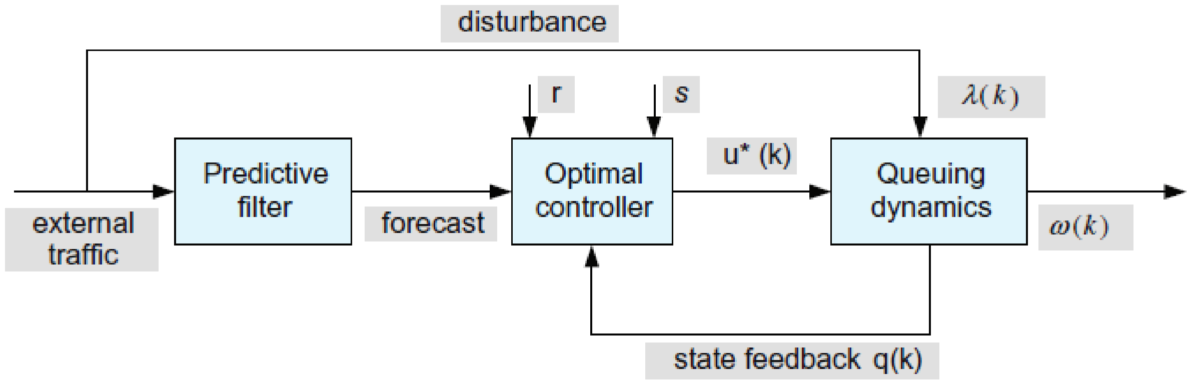
Control theory has been used to design adaptive resource management for many classes of applications, including power management , task scheduling , QoS adaptation in Web servers, and load balancing.

The classical feedback control methods are used in all these cases to regulate the key operating parameters of the system based on measurement of the system output; the feedback control in these methods assumes a linear time-invariant system model and a closed-loop controller.

Control Theory Principles:

Control theory principles one could use for optimal resource allocation. Optimal control generates a sequence of control inputs over a look-ahead horizon while estimating changes in operating conditions.

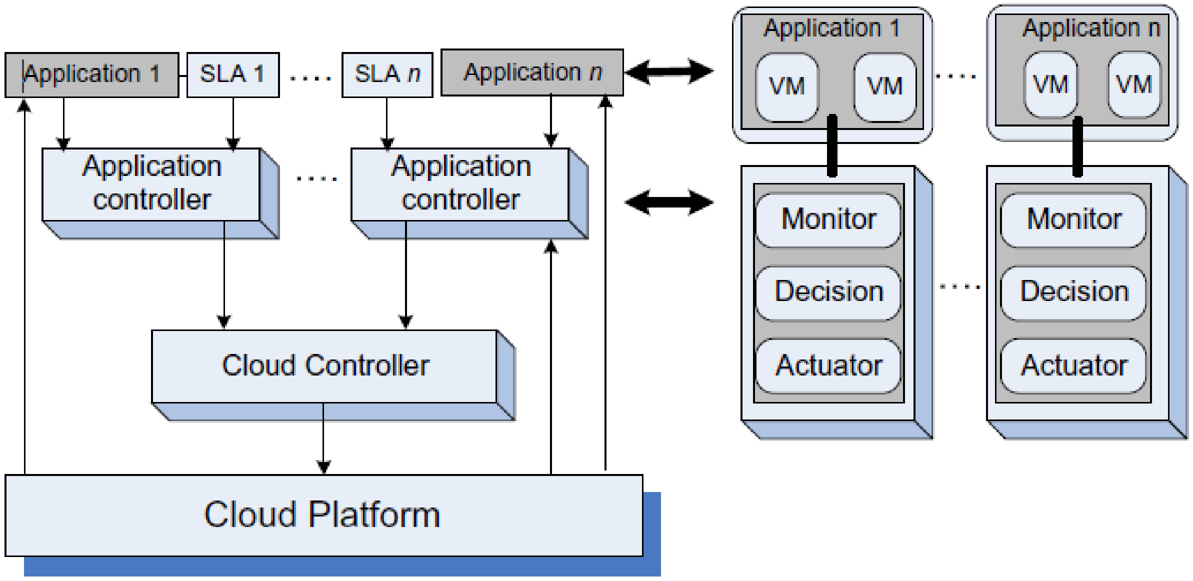
The controller uses the feedback regarding the current state as well as the estimation of the future disturbance due to environment to compute the optimal inputs over a finite horizon. The two parameters r and s are the weighting factors of the performance index.



Stability of a two-level resource allocation architecture

A server with a closed-loop control system and can apply control theory principles to resource allocation.

Allocation architecture based on control theory concepts for the entire cloud. The automatic resource management is based on two levels of controllers, one for the service provider and one for the application.



The main components of a control system are the inputs, the control system components, and the outputs. The inputs in such models are the offered workload and the policies for admission control, the capacity allocation, the load balancing, the energy optimization, and the QoS guarantees in the cloud.

The system components are sensors used to estimate relevant measures of performance and controllers that implement various policies; the output is the resource allocations to the individual applications.

Feedback and Stability:

Control granularity - the level of detail of the information used to control the system.

- **Fine control** - very detailed information about the parameters controlling the system state is used.
- **Coarse control** - the accuracy of these parameters is traded for the efficiency of implementation.

The controllers use the feedback provided by sensors to stabilize the system. Stability is related to the change of the output.

Sources of instability in any control system:

- The delay in getting the system reaction after a control action.
- The granularity of the control, the fact that a small change enacted by the controllers leads to very large changes of the output.
- Oscillations, when the changes of the input are too large and the control is too weak, such that the changes of the input propagate directly to the output.

Feedback control based on dynamic thresholds

The elements involved in a control system are sensors, monitors, and actuators. The sensors measure the parameter(s) of interest, then transmit the measured values to a monitor, which determines whether the system behavior must be changed, and, if so, it requests that the actuators carry out the necessary actions.

The implementation of such a policy is challenging.

• First, due to the very large number of servers and to the fact that the load changes rapidly in time, the estimation of the current system load is likely to be inaccurate.

• Second, the ratio of average to maximal resource requirements of individual users specified in a service-level agreement is typically very high.

Thresholds: A threshold is the value of a parameter related to the state of a system that triggers a change in the system behavior. Thresholds are used in control theory to keep critical parameters of a system in a predefined range.

The threshold could be static , defined once and for all, or it could be dynamic . A dynamic threshold could be based on an average of measurements carried out over a time interval, a so-called integral control.

The dynamic threshold could also be a function of the values of multiple parameters at a given time or a mix of the two. To maintain the system parameters in a given range, **a high and a low** threshold are often defined.

The two thresholds determine different actions; for example, a high threshold could force the system to limit its activities and a low threshold could encourage additional activities.

Proportional Thresholding:

There are two types of controllers,
(1) application controllers that determine whether additional resources are needed and
(2) cloud controllers that arbitrate requests for resources and allocate the physical resources?

The essence of the proportional thresholding is captured by the following algorithm:

1. Compute the integral value of the high and the low thresholds as averages of the maximum and, respectively, the minimum of the processor utilization over the process history.

2. Request additional VMs when the average value of the CPU utilization over the current time slice exceeds the high threshold.

3. Release a VM when the average value of the CPU utilization over the current time slice falls below the low threshold.

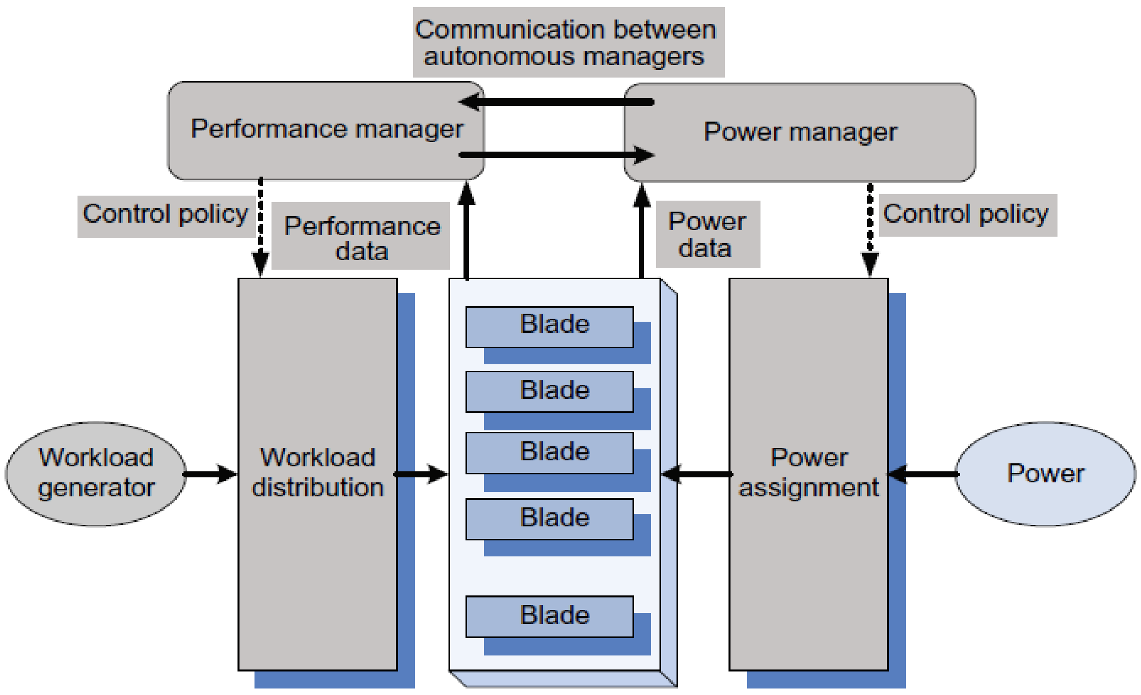
4. The conclusions reached based on experiments with three VMs are as follows:
   (a) Dynamic thresholds perform better than static ones and
   (b) Two thresholds are better than one.

Coordination of specialized autonomic performance managers:

It Can specialized autonomic performance managers cooperate to optimize power consumption. The reports on actual experiments carried out on a set of blades mounted on a chassis.

Virtually all modern processors support dynamic voltage scaling (DVS) as a mechanism for energy saving. Indeed, the energy dissipation scales quadratically with the supply voltage.

The management controls the CPU frequency and, thus, the rate of instruction execution. For some compute-intensive workloads the performance decreases linearly with the CPU clock frequency, whereas for others the effect of lower clock frequency is less noticeable or nonexistent. The clock frequency of individual blades/servers is controlled by a power manager, typically implemented in the firmware; it adjusts the clock frequency several times a second.



The approach to coordinating power and performance management in is based on several ideas:

- Use a joint utility function for power and performance. The joint performance-power utility function, $U_{pp}(R, P)$, is a function of the response time, R, and the power, P, and it can be of the form

$$U_{pp}(R, P) = U(R) - \epsilon \times P \quad \text{or} \quad U_{pp}(R, P) = \frac{U(R)}{P},$$

With U(R) the utility function based on response time only and a parameter to weight the influence of the two factors, response time and power.
- Identify a minimal set of parameters to be exchanged between the two managers.

- Set up a power cap for individual systems based on the utility-optimized power management policy.
- Use a standard performance manager modified only to accept input from the power manager regarding the frequency determined according to the power management policy.

Resource bundling

Combinatorial auctions for cloud resources:

Resources in a cloud are allocated in bundles, allowing users get maximum benefit from a specific combination of resources. Indeed, along with CPU cycles, an application needs specific amounts of main memory, disk space, network bandwidth, and so on.

Resource bundling complicates traditional resource allocation models and has generated interest in economic models and, in particular, auction algorithms.

Combinatorial Auctions.:
Auctions in which participants can bid on combinations of items, or pack- ages , are called combinatorial auctions. Such auctions provide a relatively simple, scalable, and solution to cloud resource allocation.

Two recent combinatorial auction algorithms are the **simultaneous clock auction and clock proxy auction.**

Pricing and Allocation Algorithms:

A pricing and allocation algorithm partitions the set of users into two disjoint sets, winners and losers, denoted as W and , respectively.

The algorithm should:

1. Be computationally tractable. Traditional combinatorial auction algorithms such as Vickey-Clarke- Groves (VLG) fail this criteria, because they are not computationally tractable.

2. Scale well. Given the scale of the system and the number of requests for service, scalability is a necessary condition.

3. Be objective. Partitioning in winners and losers should only be based on the price p u of a user's bid. If the price exceeds the threshold, the user is a winner; otherwise the user is a loser.

4. Be fair. Make sure that the prices are uniform . All winners within a given resource pool pay the same price.

5. Indicate clearly at the end of the auction the unit prices for each resource pool.

6. Indicate clearly to all participants the relationship between the supply and the demand in the system.

The function to be maximized is

$$\max_{x,p} f(x, p).$$

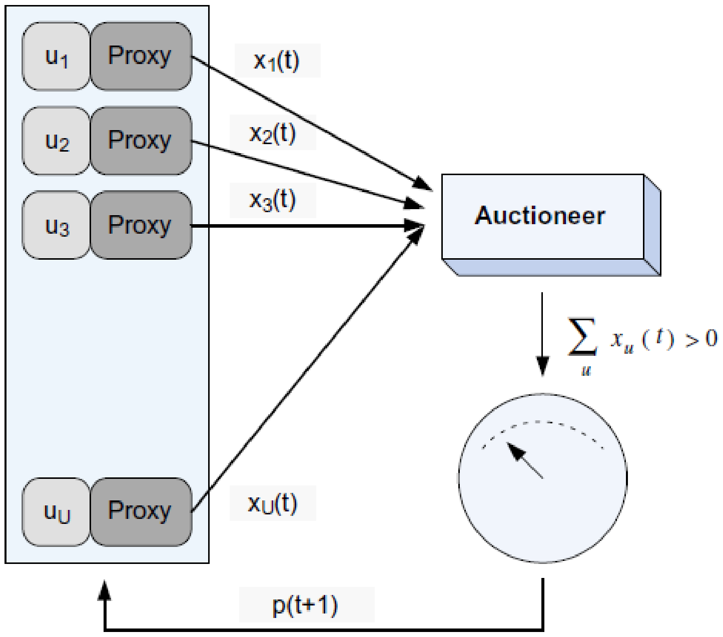
The ASCA Combinatorial Auction Algorithm:

Informally, in the ASCA algorithm the participants at the auction specify the resource and the quantities of that resource offered or desired at the price listed for that time slot. Then the excess vector

$$z(t) = \sum_{u} x_u(t)$$

is computed. If all its components are negative, the auction stops; negative components mean that the demand does not exceed the offer. If the demand is larger than the offer, $z(t) \geq 0$, the auctioneer increases the price for items with a positive excess demand and solicits bids at the new price. There is a slight complication as the algorithm involves user bidding in multiple rounds. To address this problem the user proxies automatically adjust their demands on behalf of the actual bidders.

```
1: set $t = 0$, $p(0) = \bar{p}$
2: loop
3:    collect bids $x_u(t) = \mathcal{G}_u(p(t)), \forall u$
4:    calculate excess demand $z(t) = \sum_u x_u(t)$
5:    if $z(t) < 0$ then
6:       break
7:       else
8:         update prices $p(t+1) = p(t) + g(x(t), p(t))$
9:         $t \leftarrow t + 1$
10:   end if
11: end loop
```



The schematics of the ASCA algorithm. To allow for a single round, auction users are represented by proxies that place the bids x_u(t). The auctioneer determines whether there is an excess demand and, in that case, raises the price of resources for which the demand exceeds the supply and requests new bids.

Scheduling algorithms for computing clouds

Scheduling is a critical component of cloud resource management. Scheduling is responsible for resource sharing/multiplexing at several levels.

A server can be shared among several virtual machines, each virtual machine could support several applications, and each application may consist of multiple threads.

CPU scheduling supports the virtualization of a processor, the individual threads acting as virtual processors; a communication link can be multiplexed among a number of virtual channels, one for each flow.

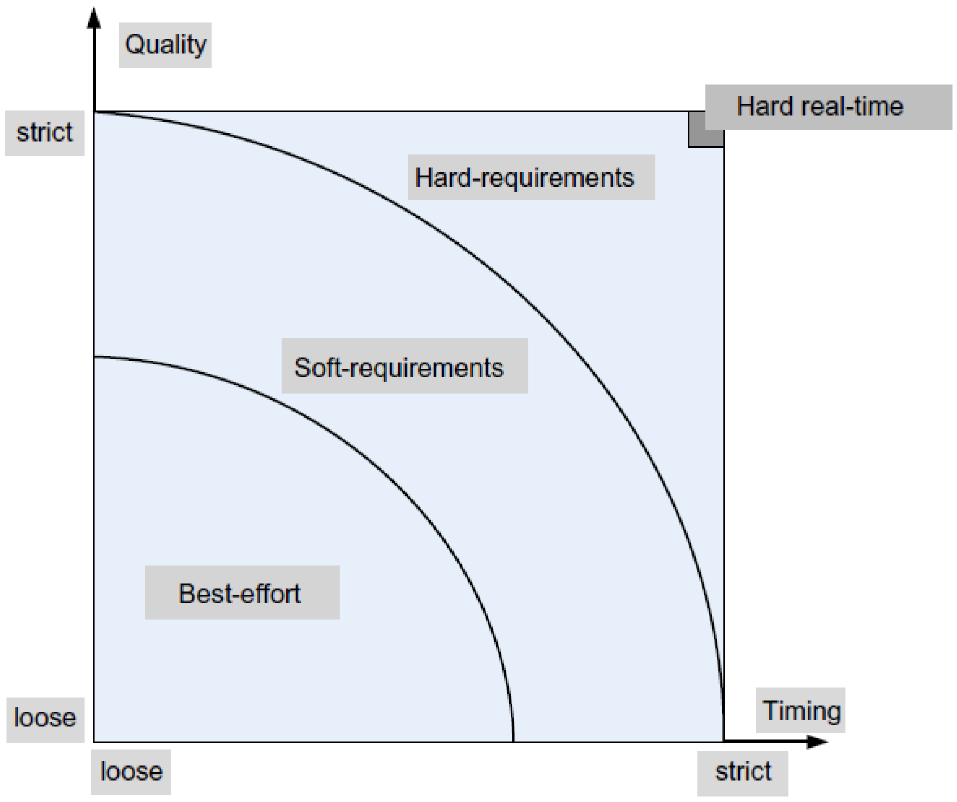
Scheduling algorithm should be efficient, fair, and starvation-free. The objectives of a scheduler for a batch system are to maximize the throughput and to minimize the turnaround time submission and its completion.

Two distinct dimensions of resource management must be addressed by a scheduling policy:

(a) The amount or quantity of resources allocated and
(b) The timing when access to resources is granted.

There are multiple definitions of a fair scheduling algorithm. First, we discuss the max-min fairness criterion. Consider a resource with bandwidth B shared among n users who have equal rights. Each user requests an amount bi and receives B_i . Then, according to the max-min criterion, the following conditions must be satisfied by a fair allocation:

C_1. The amount received by any user is not larger than the amount requested, $B_i \leqslant b_i$.
C_2. If the minimum allocation of any user is B_{min} no allocation satisfying condition C_1 has a higher B_{min} than the current allocation.
C_3. When we remove the user receiving the minimum allocation B_{min} and then reduce the total amount of the resource available from B to $(B - B_{min})$, the condition C_2 remains recursively true.



Round-robin, FCFS, shortest-job-first (SJF), and priority algorithms are among the most common scheduling algorithms for best-effort applications. Each thread is given control of the CPU for a definite period of time, called a time-slice , in a circular fashion in the case of round-robin scheduling.

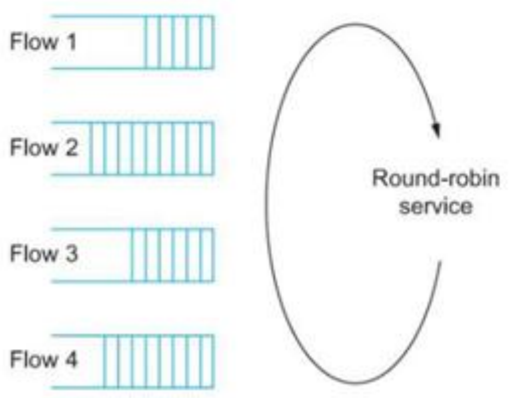
The algorithm is fair and starvation-free. The threads are allowed to use the CPU in the order in which they arrive in the case of the FCFS algorithms and in the order of their running time in the case of SJF algorithms. Earliest deadline first (EDF) and rate monotonic algorithms (RMA) are used for real-time applications.

Fair queuing

Computing and communication on a cloud are intimately related. Therefore, it should be no surprise that the first algorithm we discuss can be used for scheduling packet transmission as well as threads.

Interconnection networks allow cloud servers to communicate with one another and with users. These networks consist of communication links of limited bandwidth and switches/routers/gateways of limited capacity. When the load exceeds its capacity, a switch starts dropping packets because it has limited input buffers for the switching fabric and for the outgoing links, as well as limited CPU cycles.

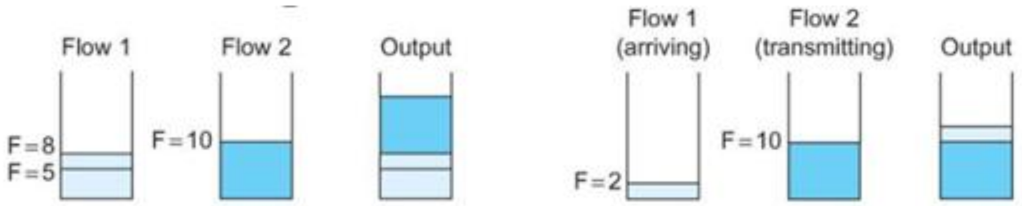
A fair queuing algorithm proposed in requires that separate queues, one per flow, be maintained by a switch and that the queues be serviced in a round-robin manner. This algorithm guarantees the fairness of buffer space management, but does not guarantee fairness of bandwidth allocation. Indeed, a flow transporting large packets will benefit from a larger bandwidth.



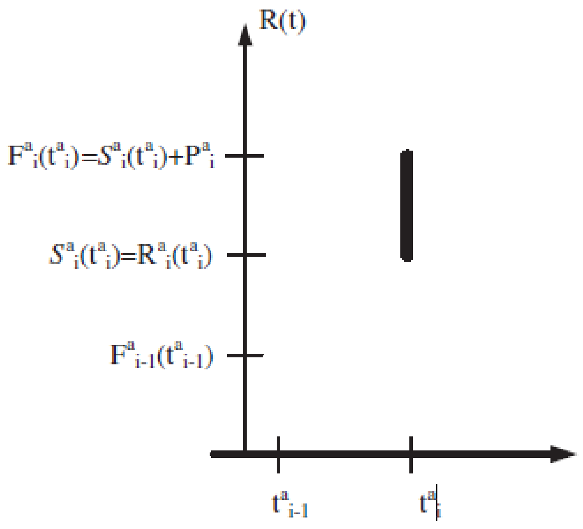
The fair queuing (FQ) algorithm, First, it introduces a bit-by-bit round-robin (BR) strategy; as the name implies, in this rather impractical scheme a single bit from each queue is transmitted and the queues are visited in a round-robin fashion.

Let R(t) be the number of rounds of the BR algorithm up to time t and $N_{active}(t)$ be the number of active flows through the switch. Call t^a_i the time when the packet i of flow a, of size P^a_i bits arrives, and call S^a_i and F^a_i the values of R(t) when the first and the last bit, respectively, of the packet i of flow a are transmitted.

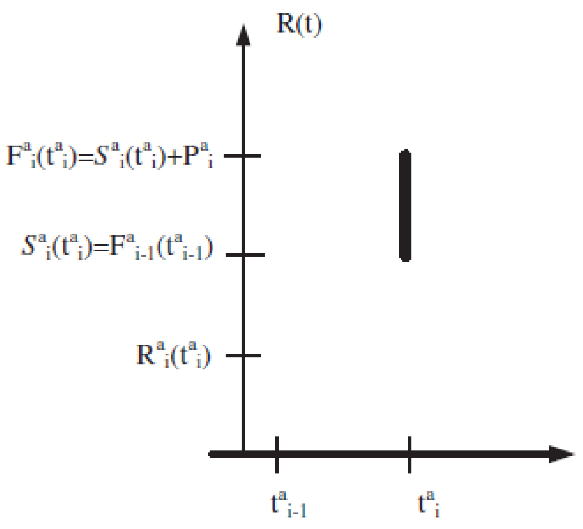
$$F_i^a = S_i^a + P_i^a \quad \text{and} \quad S_i^a = \max\left[F_{i-1}^a, R(t_i^a)\right].$$



Example of fair queuing in action:
(a) packets with earlier finishing times are sent first;
(b) sending of a packet already in progress is completed


R(t)
$F^a_i(t^a_i)=S^a_i(t^a_i)+P^a_i$
$S^a_i(t^a_i)=R^a_i(t^a_i)$
$F^a_{i-1}(t^a_{i-1})$
t^a_{i-1}
t^a_i

(a)


R(t)
$F^a_i(t^a_i)=S^a_i(t^a_i)+P^a_i$
$S^a_i(t^a_i)=F^a_{i-1}(t^a_{i-1})$
$R^a_i(t^a_i)$
t^a_{i-1}
t^a_i

(b)

The transmission of packet i of a flow can only start after the packet is available and the transmission of the previous packet has finished. (a) The new packet arrives after the previous has finished. (b) The new packet arrives before the previous one was finished.

Afair allocation of the bandwidth does not have an effect on the timing of the transmission. A possible strategy is to allow less delay for the flows using less than their fair share of the bandwidth.

$$B_i^a = P_i^a + \max\left[F_{i-1}^a, \left(R\left(t_i^a\right) - \delta\right)\right]$$

The properties of the FQ algorithm, as well as the implementation of a non preemptive version of the algorithms.

<u>Start-time fair queuing</u>

A hierarchical CPU scheduler for multimedia operating systems was proposed in. The basic idea of the start-time fair queuing (SFQ) algorithm is to organize the consumers of the CPU bandwidth in a tree structure; the root node is the processor and the leaves of this tree are the threads of each application. A scheduler acts at each level of the hierarchy. The fraction of the processor bandwidth.

$$\frac{B_i}{B} = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

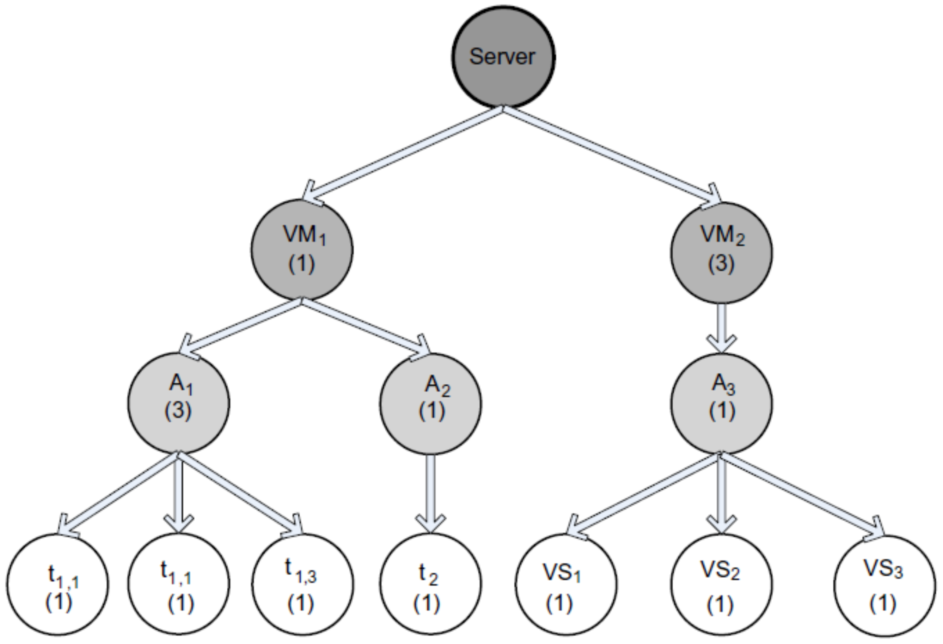
with wj , 1 $\leq$ j $\leq$ n, the weight of the n children of node i

An SFQ scheduler follows several rules:

R1. The threads are serviced in the order of their virtual start-up time; ties are broken arbitrarily.

R2. The virtual startup time of the *i* -th activation of thread *x* is

$$S_x^i(t) = \max\left[v\left(\tau^j\right), F_x^{(i-1)}(t)\right] \quad \text{and} \quad S_x^0 = 0.$$



The SFQ tree for scheduling when two virtual machines, VM_1 and VM_2, run on a powerful server. VM_1 runs two best-effort applications A_1, with three threads $t_{1,1}$, $t_{1,2}$, and $t_{1,3}$, and A_2 with a single thread, t_2. VM_2 runs a video-streaming application, A_3, with three threads vs_1, vs_2, and vs_3. The weights of virtual machines, applications, and individual threads are shown in parenthesis.

R3. The virtual finish time of the i -th activation of thread x is

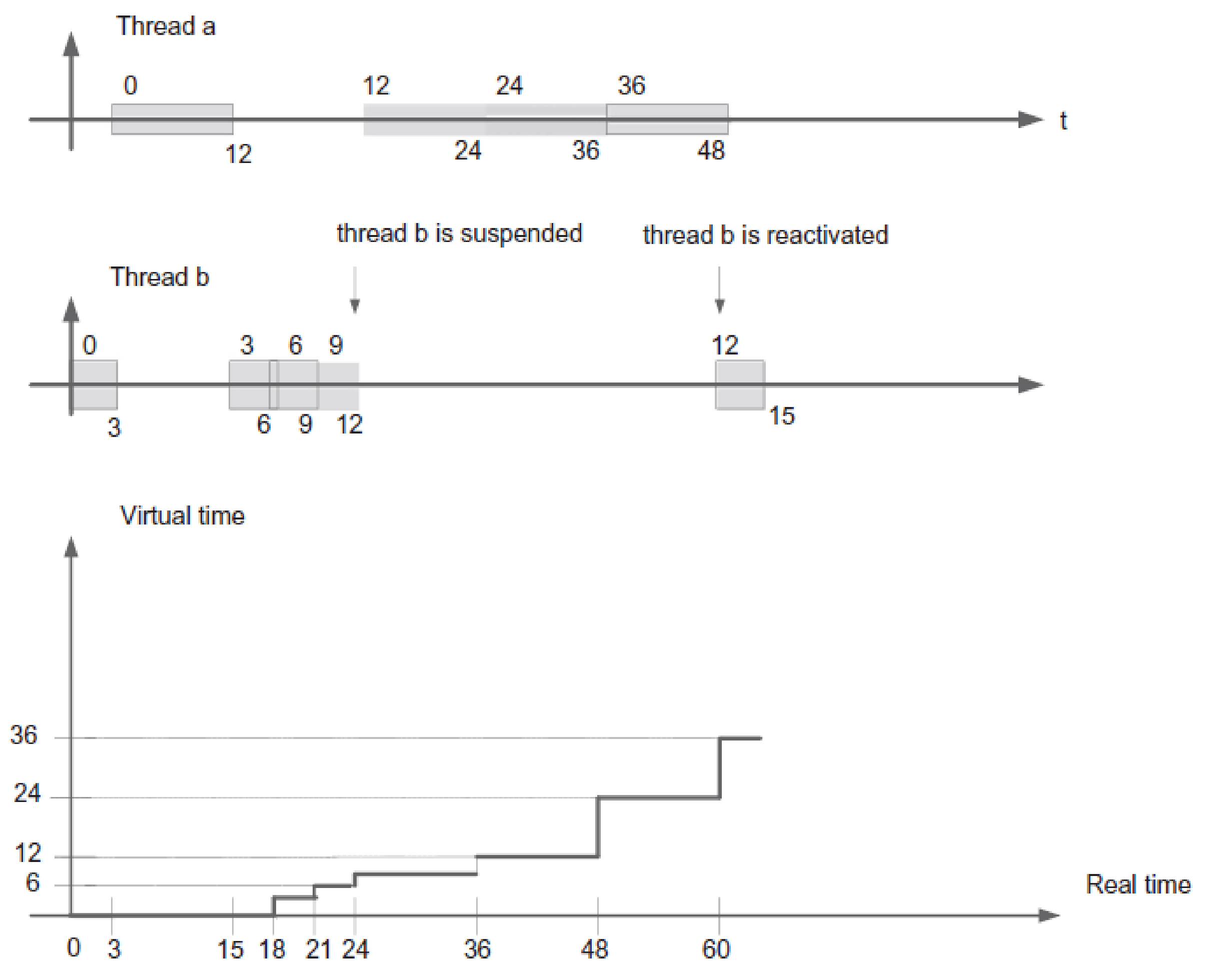
$$F_x^i(t) = S_x^i(t) + \frac{q}{w_x}.$$

A thread is stopped when its time quantum has expired; its time quantum is the time quantum of the scheduler divided by the weight of the thread.

R4. The virtual time of all threads is initially zero, $v^0_x = 0$. The virtual time v(t) at real time t is computed as follows:

$$v(t) = \begin{cases} \text{Virtual start time of the thread in service at time } t, & \text{if CPU is busy} \\ \text{Maximum finish virtual time of any thread,} & \text{if CPU is idle.} \end{cases}$$

Example. The following example illustrates the application of the SFQ algorithm when there are two threads with the weights $w_a = 1$ and $w_b = 4$ and the time quantum is $q = 12$



Cloud scheduling subject to deadlines

An SLA specifies the time when the results of computations done on the cloud should be available. This motivates us to examine cloud scheduling subject to deadlines, a topic drawing on a vast body of literature devoted to real-time applications.

Hard deadlines

- if the task is not completed by the deadline, other tasks which depend on it may be affected and there are penalties; a hard deadline is strict and expressed precisely as milliseconds, or possibly seconds.

Soft deadlines

- more of a guideline and, in general, there are no penalties; soft deadlines can be missed by fractions of the units used to express them, e.g., minutes if the deadline is expressed in hours, or hours if the deadlines is expressed in days.

System Model:

1. The order of execution of the tasks Π_i.
2. The workload partitioning and the task mapping to worker nodes.

Scheduling Policy:

- First in, first out (FIFO). The tasks are scheduled for execution in the order of their arrival.
- Earliest deadline first (EDF). The task with the earliest deadline is scheduled first.
- Maximum workload derivative first (MWF).

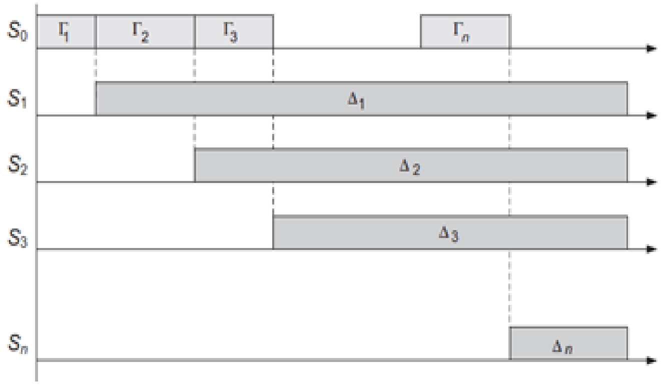
Task Partitioning:

1. OPR (Optimal Partitioning Rule)
2. EPR (Equal Partitioning Rule)

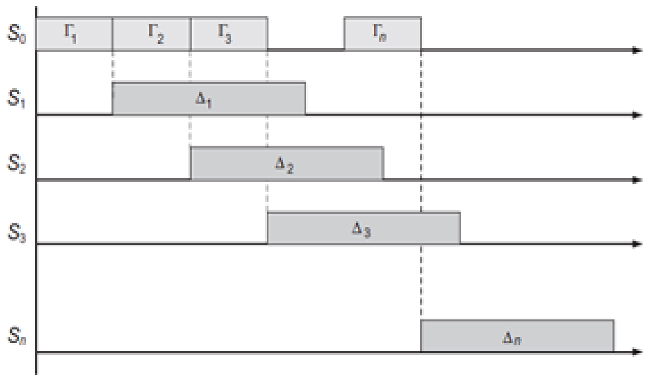
OPR is based on DLT and hence all the nodes allocated to a particular task complete execution at the same time.

EPR divides a task into a number of subtasks of equal size and they are assigned to different nodes for execution.

OPR (Optimal Partitioning Rule)

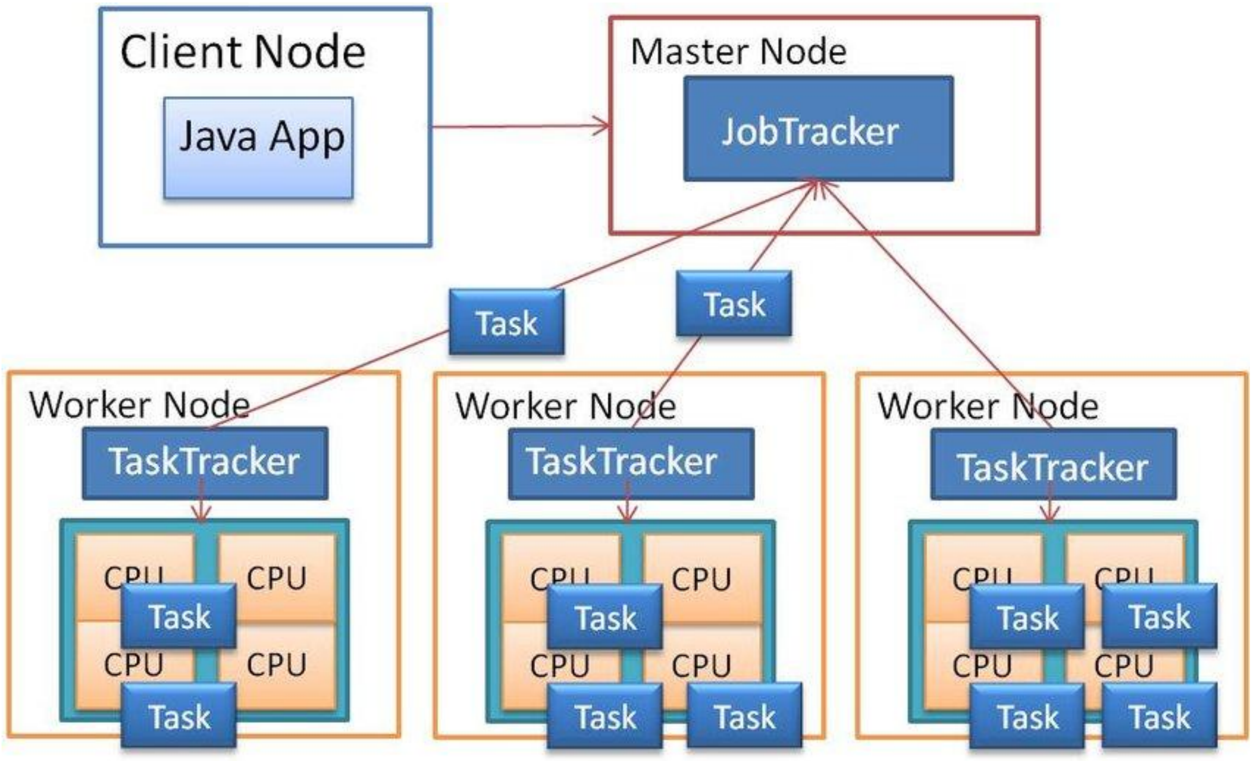


EPR (Equal Partitioning Rule)



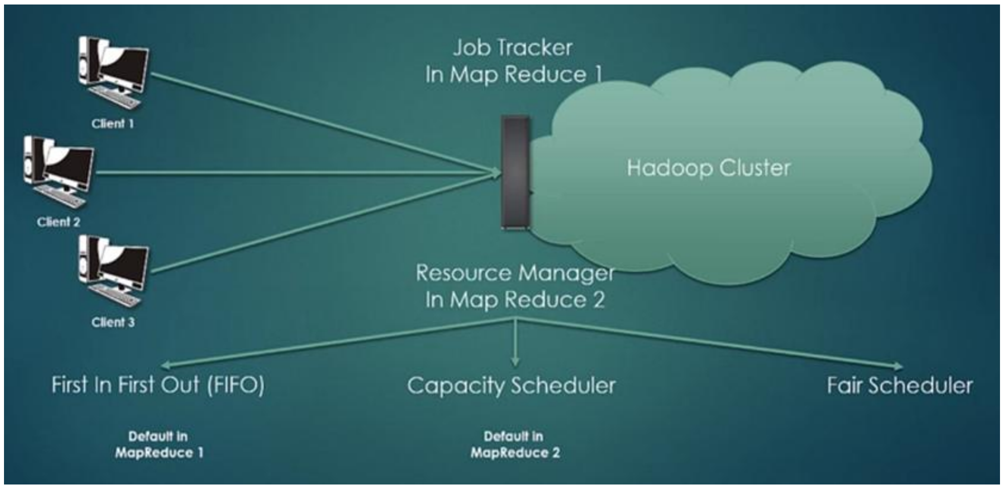
Scheduling MapReduce applications subject to deadlines

Many MapReduce applications come with strict deadlines and the fair scheduler doesn't guarantee that the job will be completed within the given deadline.



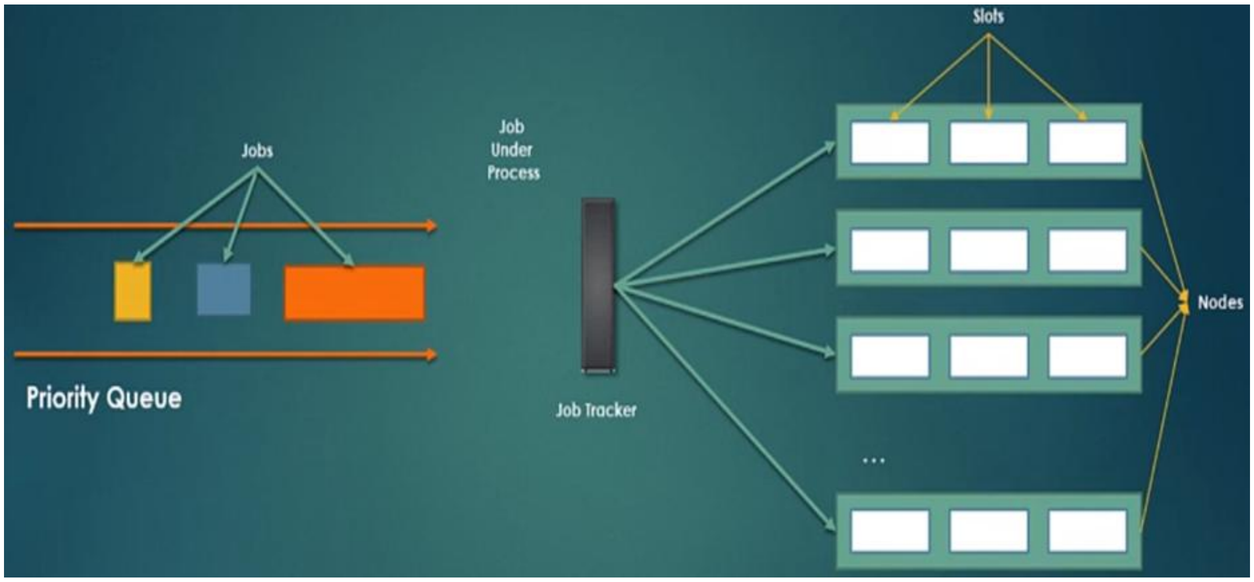
Several options for scheduling Apache Hadoop, an open-source implementation of the MapReduce algorithm

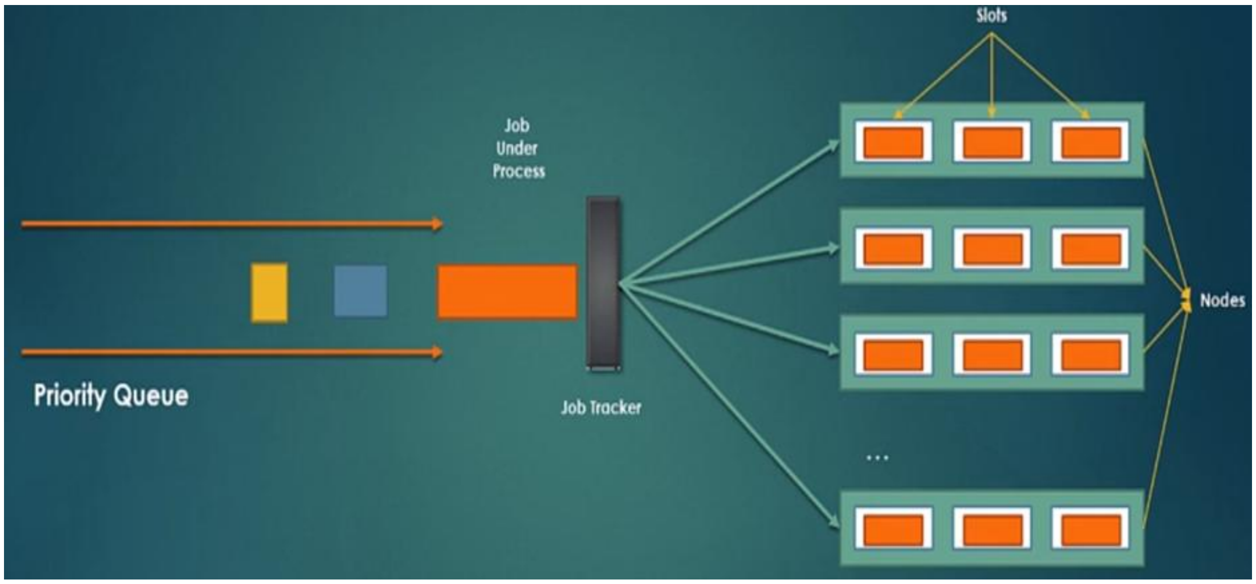
1. The default FIFO schedule.
2. The Fair Scheduler.
3. The Capacity Scheduler.
4. The Dynamic Proportional Scheduler.



FIFO Schedule:

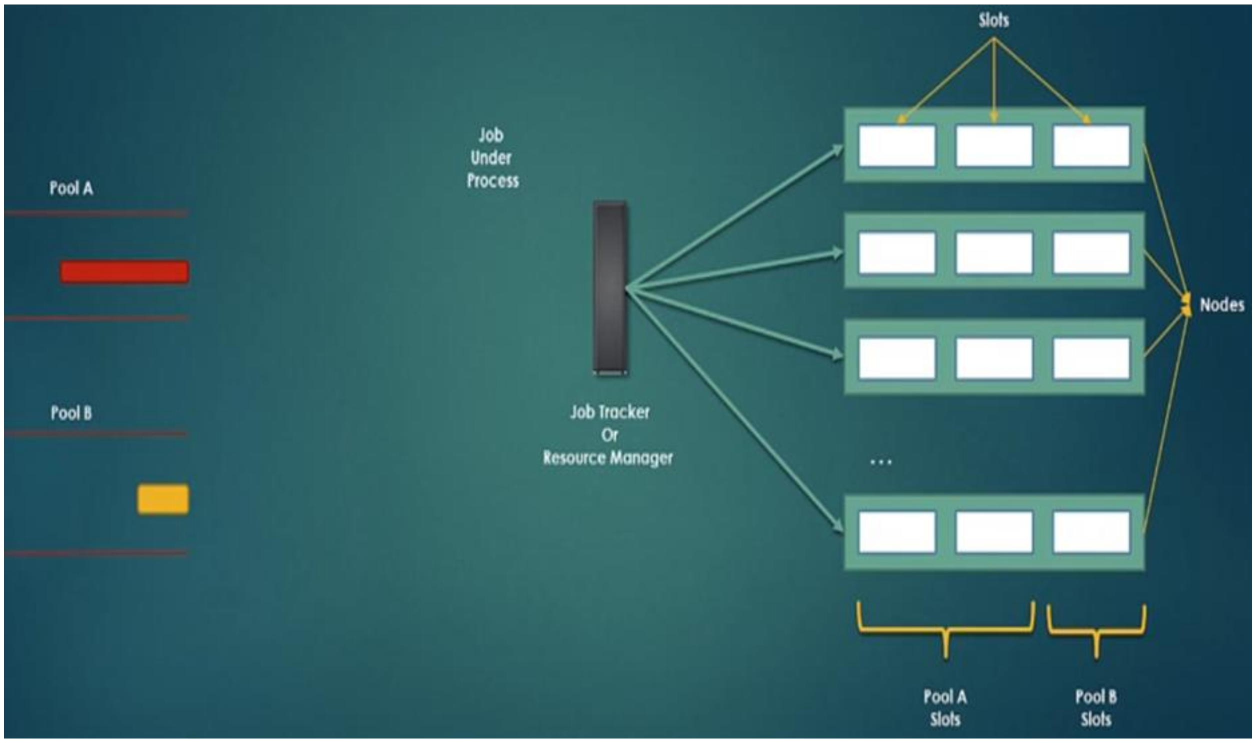
First In First Out, so the tasks or application that comes first will be served first. This is the default Scheduler we use in Hadoop. The tasks are placed in a queue and the tasks are performed in their submission order. In this method, once the job is scheduled, no intervention is allowed. So sometimes the high-priority process has to wait for a long time since the priority of the task does not matter in this method.

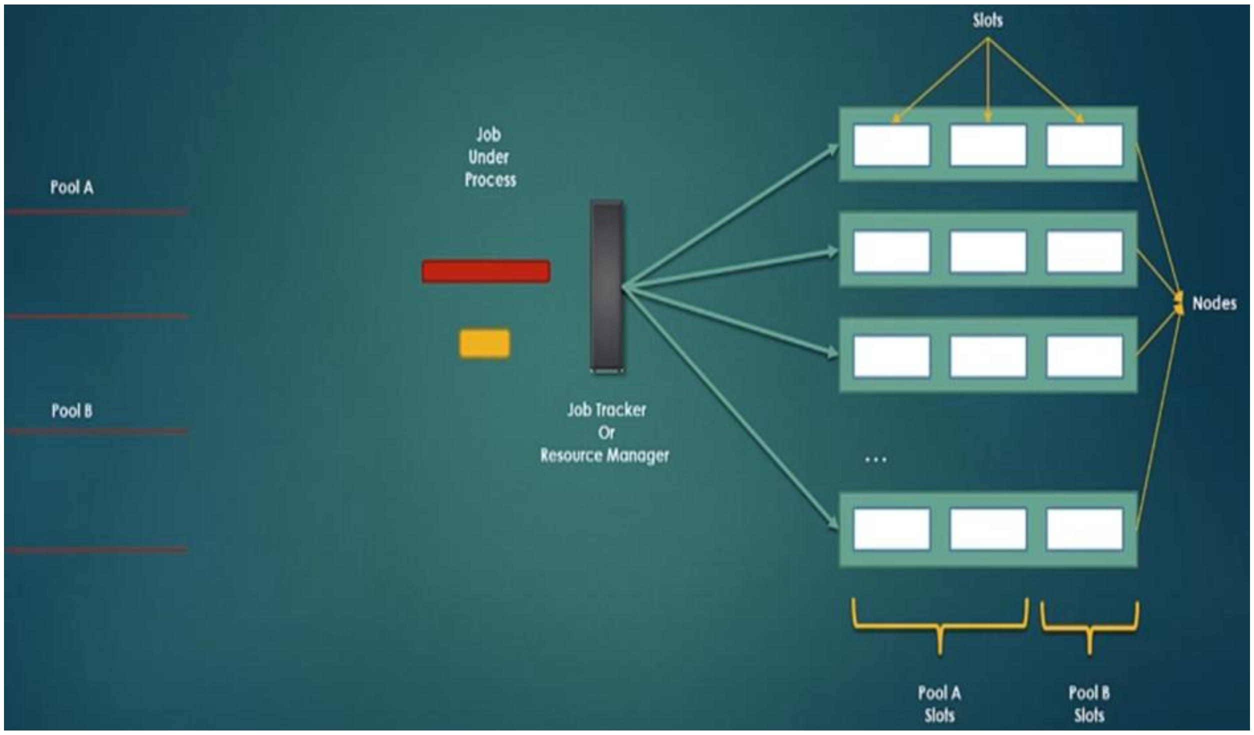


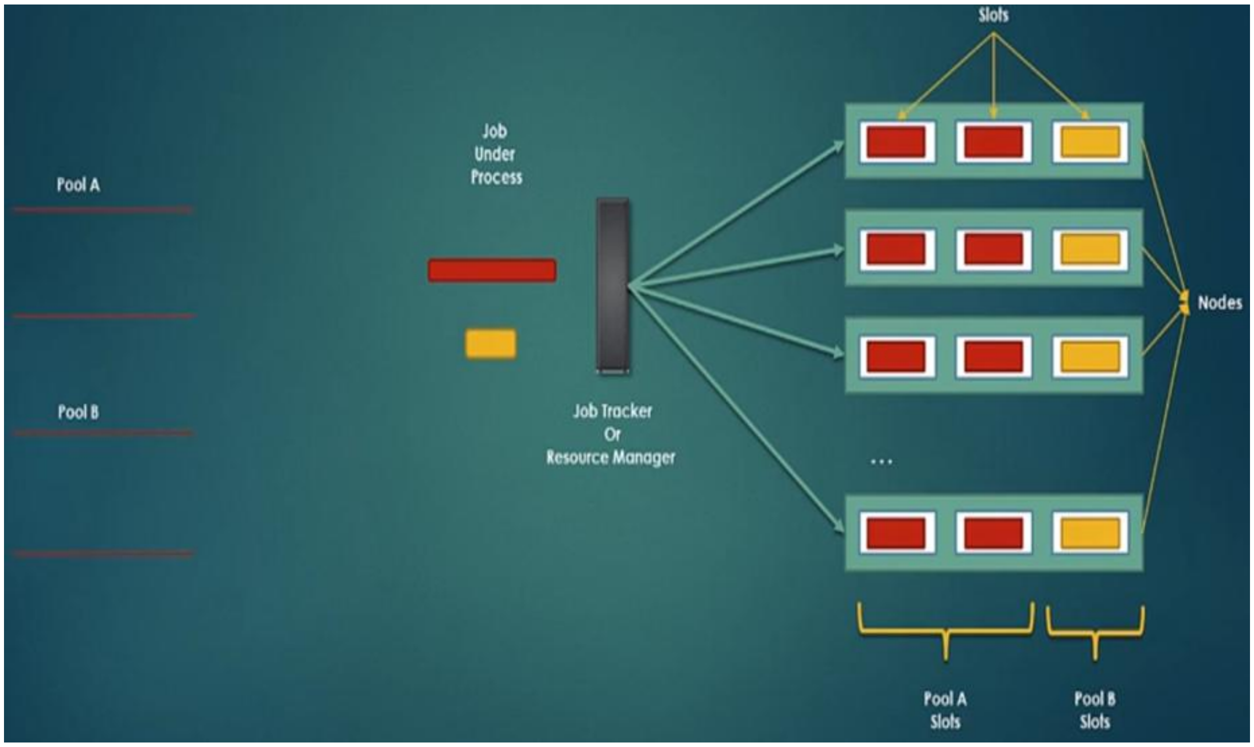


Fair Scheduler:

The Fair Scheduler is very much similar to that of the capacity scheduler. The priority of the job is kept in consideration. With the help of Fair Scheduler, the YARN applications can share the resources in the large Hadoop Cluster and these resources are maintained dynamically so no need for prior capacity.

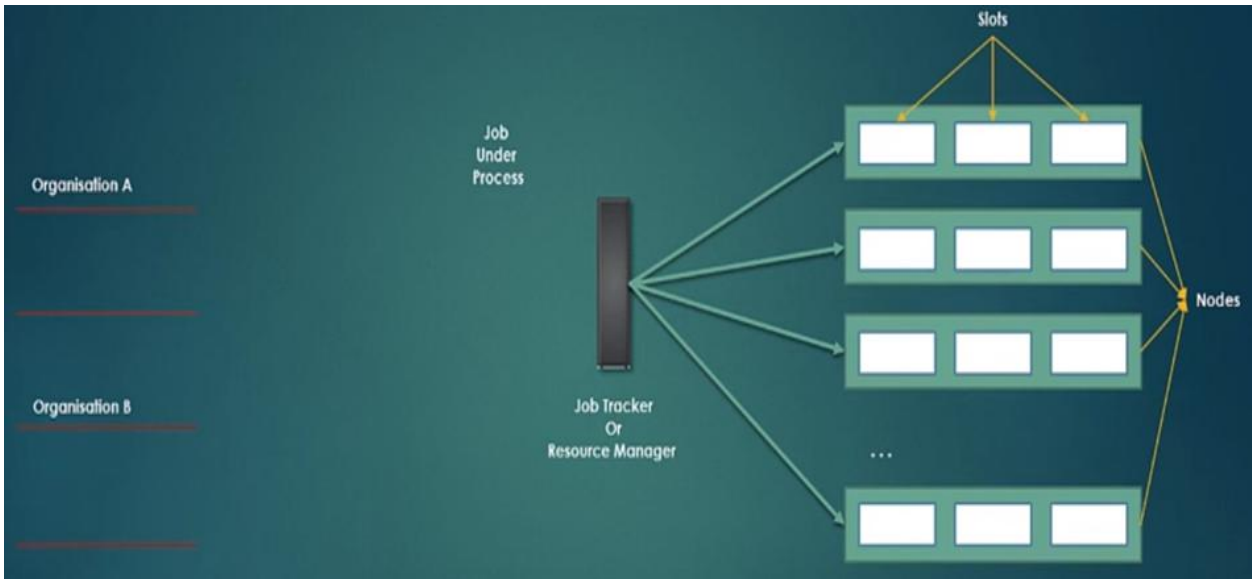


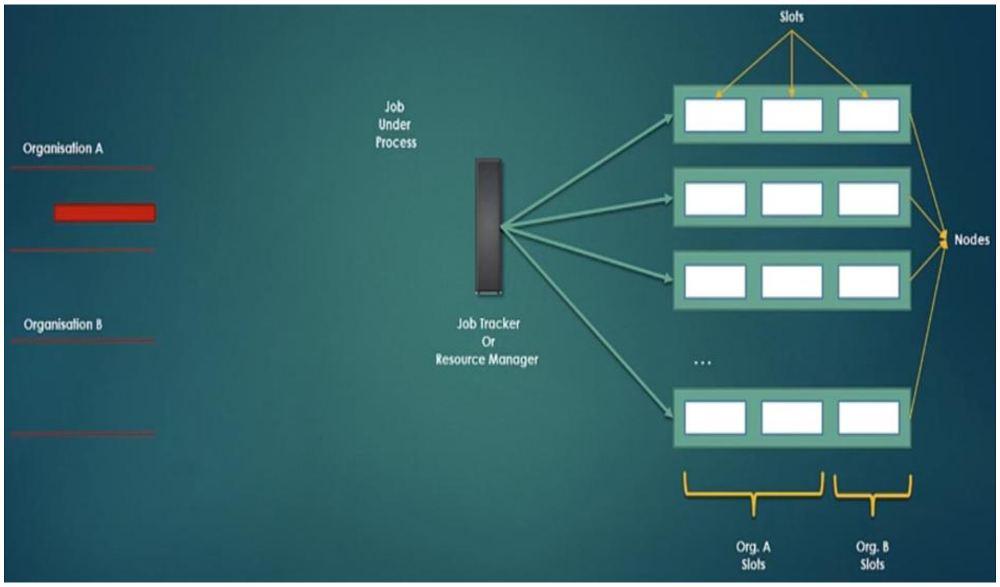


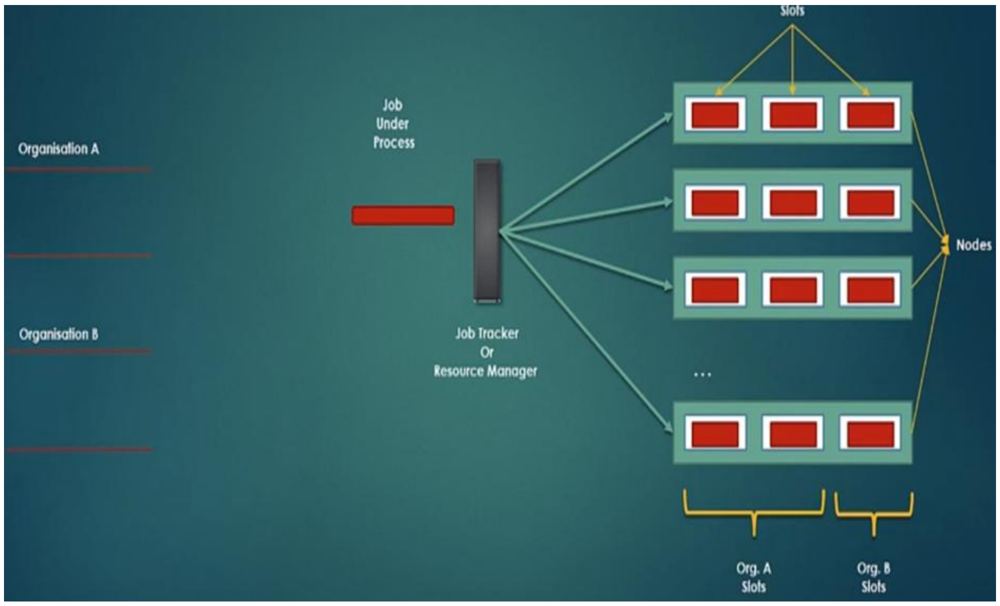


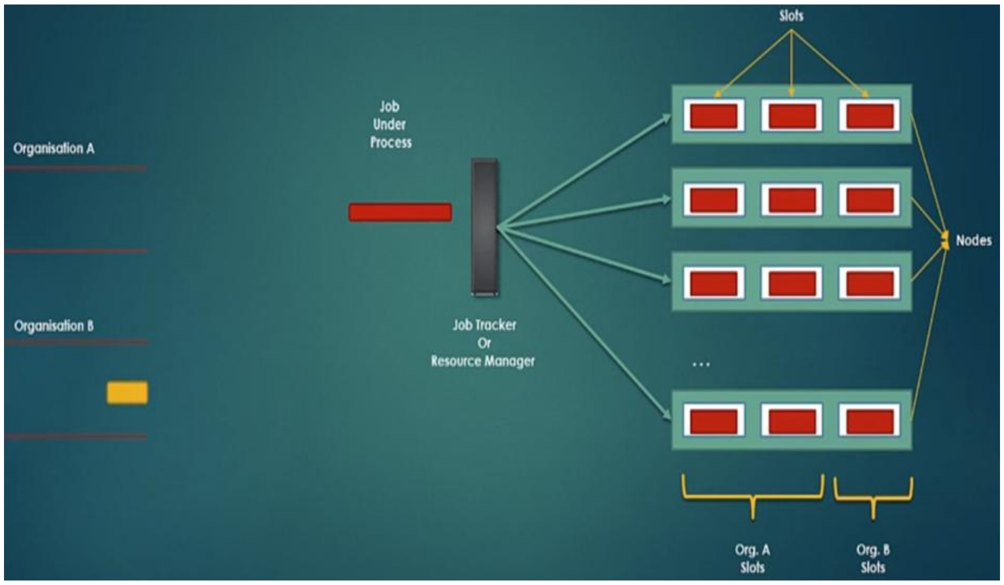
Capacity Scheduler:

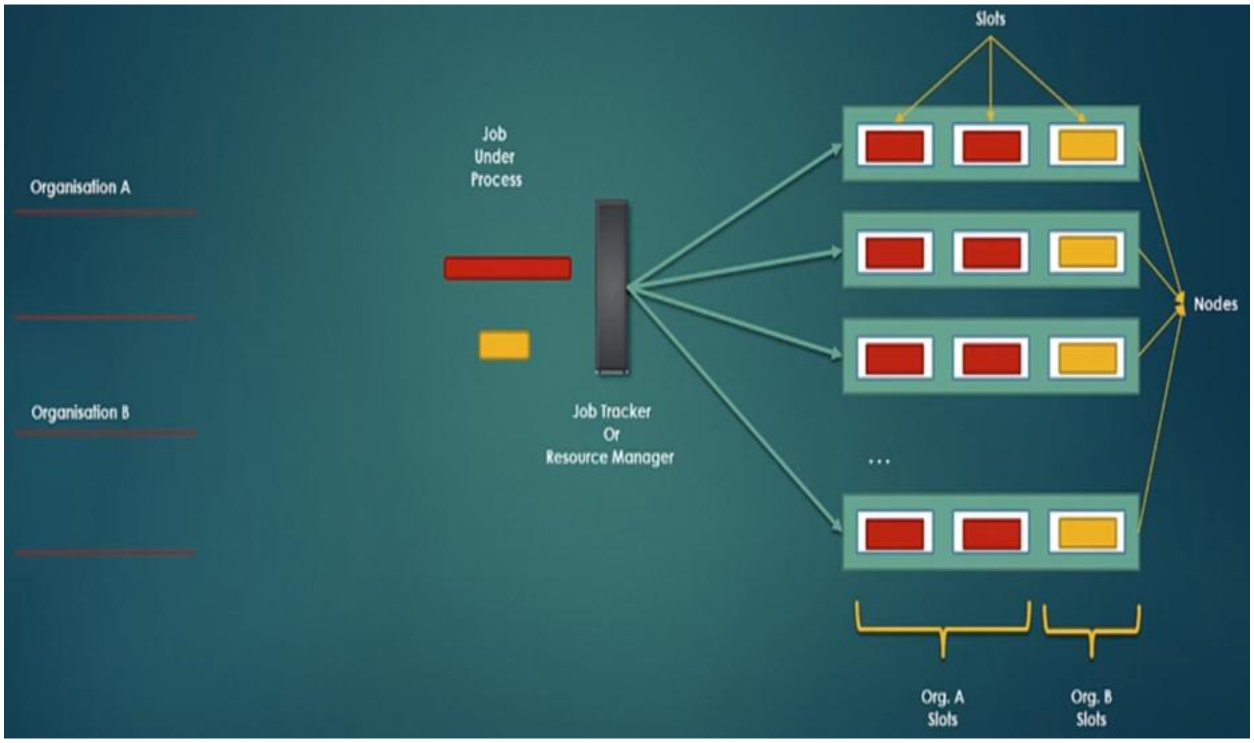
In Capacity Scheduler we have multiple job queues for scheduling our tasks. The Capacity Scheduler allows multiple occupants to share a large size Hadoop cluster. In Capacity Scheduler corresponding for each job queue, we provide some slots or cluster resources for performing job operation. Each job queue has it's own slots to perform its task.




Slots
Job
Under
Process
Organisation A
Organisation B
Job Tracker
Or
Resource Manager
Nodes
...
Org. A
Slots
Org. B
Slots


Slots
Job
Under
Process
Organisation A
Organisation B
Job Tracker
Or
Resource Manager
Nodes
...
Org. A
Slots
Org. B
Slots


Slots
Job
Under
Process
Organisation A
Organisation B
Job Tracker
Or
Resource Manager
Nodes
...
Org. A
Slots
Org. B
Slots



Resource management and dynamic application scaling

The demand for computing resources, such as CPU cycles, primary and secondary storage, and network bandwidth, depends heavily on the volume of data processed by an application. The demand for resources can be a function of the time of day, can monotonically increase or decrease in time, or can experience predictable or unpredictable peaks.

We distinguish two scaling modes:

- Vertical and horizontal.

Vertical scaling keeps the number of VMs of an application constant, but increases the amount of resources allocated to each one of them. This can be done either by migrating the VMs to more powerful servers or by keeping the VMs on the same servers but increasing their share of the CPU time.

Horizontal scaling is the most common mode of scaling on a cloud; it is done by increasing the number of VMs as the load increases and reducing the number of VMs when the load decreases. Often, this leads to an increase in communication bandwidth consumed by the application. Load balancing among the running VMs is critical to this mode of operation.