

HEADING:	:	Dark Purple 1, Light Gold 2	20
TYPES:		Green	18
SUBTYPES:		Yellow	16
Important Points:		Orange	16
Important Points:		Gold	16
Important Points:		Lime	16

# UNIT 1

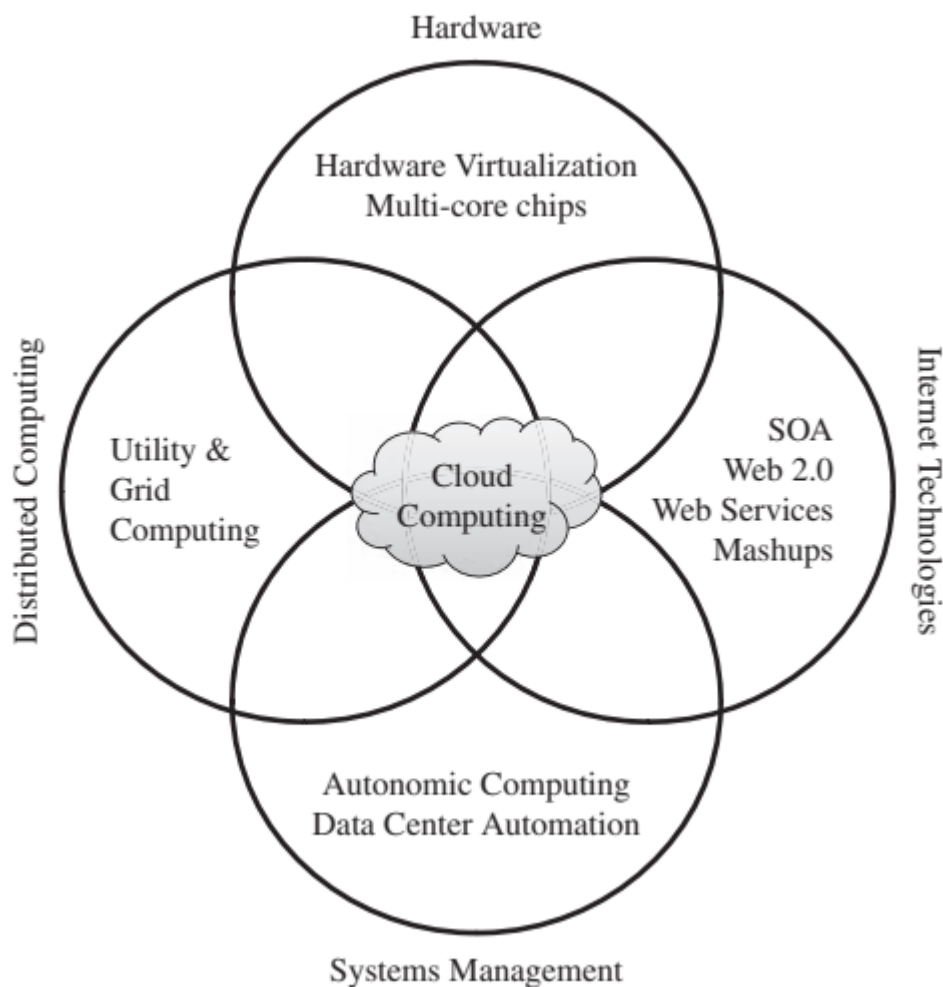
## **Distributed System Models & Enabling technology:**

Scalable computing over the internet,  
Technologies for network-based system,  
System models for distributed & cloud,  
Software environments for Distributed & Cloud,  
Performance security and Energy efficiency

# UNIT 2

Introduction to Cloud Computing:  
Cloud Computing in a Nutshell,  
System Model for Distributed and Cloud Computing,  
**Roots of Cloud Computing,**  
Grid and Cloud,  
**Layers and Types of Clouds,**  
**Desired Features of a Cloud,**  
Basic Principles of Cloud Computing,  
Challenges and Risks,  
Service Models

## Roots of CC



# GRID Computing

Grid computing is a distributed computing paradigm that aims to aggregate resources from multiple sources and provide transparent access to them. It has been primarily used in scientific applications such as climate modeling, drug design, and protein analysis, where vast computational power is required.

Key points about grid computing include:

**Resource Sharing:** Grid computing allows for the sharing of compute and storage resources distributed across different administrative domains. Projects like TeraGrid and EGEE focus on leveraging these resources to accelerate scientific research.

**Standardization:** The development of standard Web services-based protocols, such as the Open Grid Services Architecture (OGSA), has been crucial for enabling interoperability and seamless management of distributed resources in grid systems.

**Middleware:** Middleware solutions like the Globus Toolkit implement standard Grid services and facilitate the deployment of service-oriented Grid infrastructures and applications. Grid brokers help manage user interactions with multiple middleware and enforce quality of service (QoS) policies.

**Challenges:** Despite the theoretical promise of delivering on-demand computing services over the Internet, ensuring QoS in grid environments has been challenging. Issues such as lack of performance isolation, resource diversity in terms of software configurations, and portability barriers have hindered widespread adoption.

**Virtualization:** Virtualization technology has emerged as a solution to address some of the challenges associated with grid computing. Projects like Globus Virtual Workspaces aim to enhance grids by introducing a layer of virtualization for computation, storage, and network resources, thereby improving flexibility and resource utilization.

## Utility Computing

Utility computing represents a paradigm shift in resource allocation, where users assign a "utility" value to their jobs based on Quality of Service (QoS) constraints such as deadlines, importance, and satisfaction. This utility value reflects the amount users are willing to pay a service provider to meet their demands. In turn, service providers aim to maximize their own utility, which may be directly related to their profit.

## Hardware Virtualization

Hardware virtualization is a technology that allows for the virtualization of a computer system's resources, including processors, memory, and I/O devices. This enables running multiple operating systems and software stacks on a single physical platform. Key points

**Virtual Machine Monitor (VMM):** Also known as a hypervisor, the VMM is a software layer that mediates access to the physical hardware. It presents each guest operating system with a virtual machine (VM), which consists of a set of virtual platform interfaces.

**Technological Advances:** Innovative technologies such as multi-core chips, paravirtualization, hardware-assisted virtualization, and live migration of VMs have contributed to the increased adoption of virtualization on server systems.

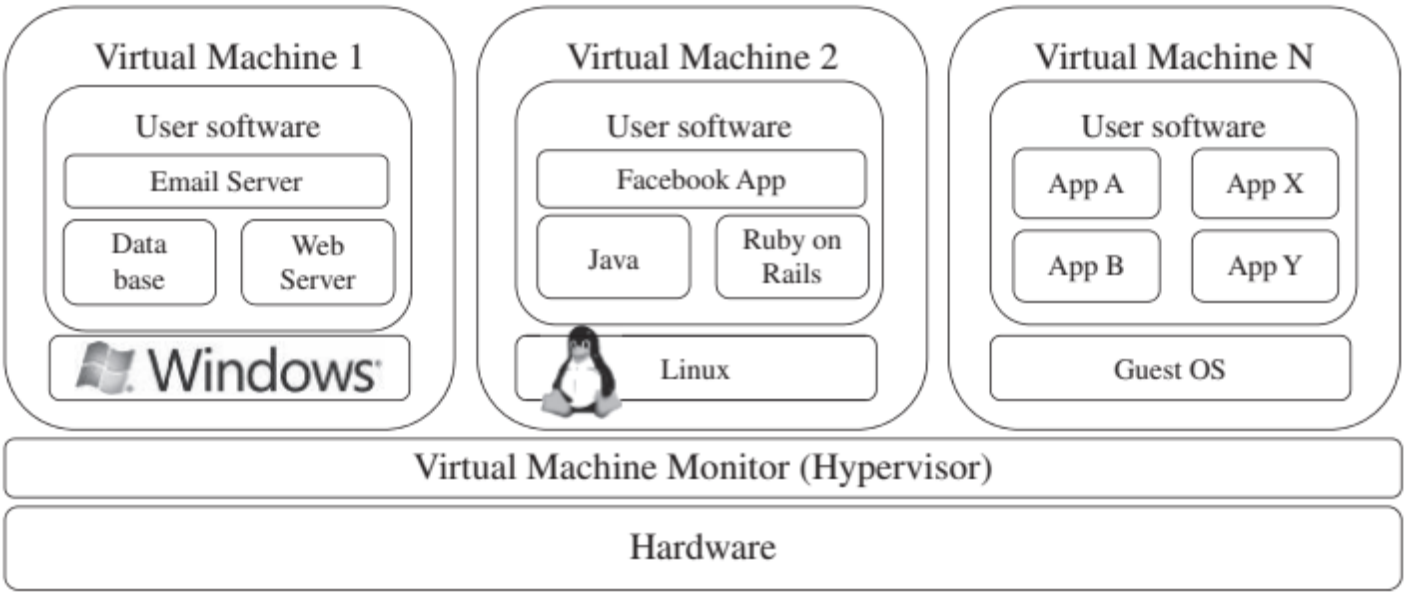
**Perceived Benefits:** Traditionally, the benefits of hardware virtualization include improvements in resource sharing, utilization, manageability, and reliability. More recently, emphasis has been placed on three basic capabilities: workload isolation, consolidation, and migration.

**Workload Isolation:** All program instructions are fully confined inside a VM, leading to improvements in security and reliability. Software failures inside one VM do not affect others

**Consolidation:** Hosting multiple individual and heterogeneous workloads on a single physical platform improves system utilization. It also helps overcome potential software and hardware incompatibilities during upgrades by allowing legacy and new operating systems to run concurrently.

**Workload Migration:** Also known as application mobility, workload migration facilitates hardware maintenance, load balancing, and disaster recovery. It involves encapsulating a guest OS state within a VM, allowing it to be suspended, migrated to a different platform, and resumed immediately or preserved for later restoration.

**Virtual Machine Platforms:** Various VMM platforms exist, serving as the basis for utility or cloud computing environments. Notable platforms include VMWare, Xen, and KVM, each offering different features and capabilities for virtualization.



**FIGURE 1.2.** A hardware virtualized server hosting three virtual machines, each one running distinct operating system and user level software stack.

# TYPES OF CLOUD

Cloud computing services are categorized into three classes based on the abstraction level of the capability provided:

## Infrastructure as a Service (IaaS):

- Providers offer virtualized computing resources over the internet.
- Users can access and manage these resources,
- Servers, Storage, Networking, as needed
- Eg: Amazon Web Services (AWS) EC2 and Microsoft Azure VM

## Platform as a Service (PaaS):

- PaaS providers offer platforms and tools that developers can use to build, deploy, and manage applications without worrying about the underlying infrastructure.
- Abstracts away the Hardware and OS
- Allows developers to focus on application development.
- Eg: Google App Engine and Heroku




## Software as a Service (SaaS):

- SaaS providers deliver software applications over the internet on a subscription basis.
- Users can access these applications through a web browser without needing to install or maintain software locally.
- Eg: Google Workspace (G Suite), Microsoft Office 365, Salesforce.

These abstraction levels form a layered architecture where services of a higher layer can be composed from services of the underlying layer.

The core middleware manages physical resources and virtual machines deployed on top of them, providing features such as accounting and billing for multi-tenant pay-as-you-go services.

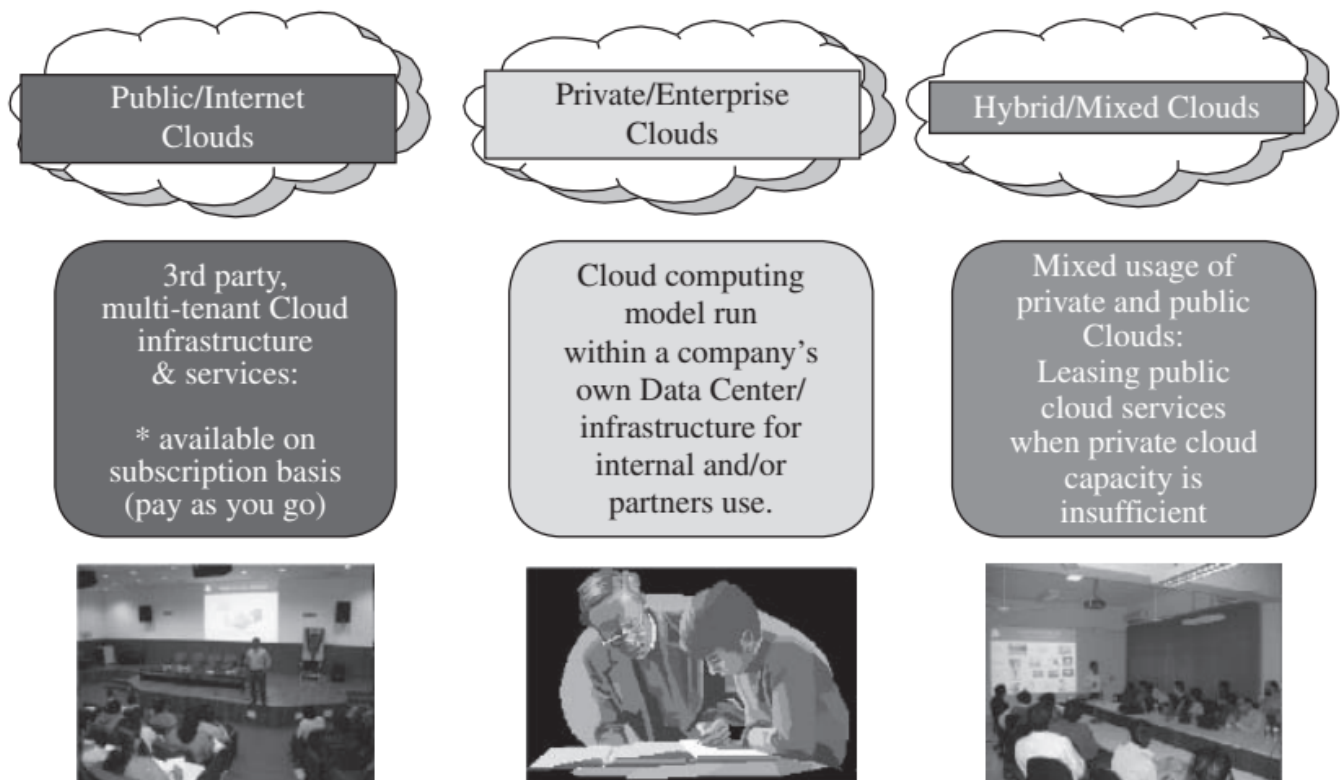
Cloud development environments are built on top of infrastructure services to offer application development and deployment capabilities, enabling the creation of various business, web, and scientific applications. Once deployed in the cloud, these applications can be consumed by end users through web browsers or other client interfaces.

Service Class	Main Access & Management Tool	Service content
 SaaS	Web Browser	<b>Cloud Applications</b> Social networks, Office suites, CRM, Video processing
 PaaS	Cloud Development Environment	<b>Cloud Platform</b> Programming languages, Frameworks, Mashups editors, Structured data
 IaaS	Virtual Infrastructure Manager	<b>Cloud Infrastructure</b> Compute Servers, Data Storage, Firewall, Load Balancer

17

**FIGURE 1.3.** The cloud computing stack.

## Deployment Models



**FIGURE 1.4.** Types of clouds based on deployment models.



## Desired Features of a Cloud

- 1. \*\*Self-Service\*\*:** Cloud consumers expect on-demand access to resources without the need for human intervention. Clouds must support self-service access, allowing customers to request, customize, pay for, and use services autonomously.
- 2. \*\*Per-Usage Metering and Billing\*\*:** Cloud computing enables users to request and use resources as needed, without upfront commitments. Services should be priced based on short-term usage (e.g., by the hour), allowing users to release resources when they are no longer required. Clouds must implement efficient pricing, accounting, and billing mechanisms to support this pay-as-you-go model, with transparent metering for different types of services.
- 3. \*\*Elasticity\*\*:** Cloud computing offers the illusion of infinite computing resources available on demand. Users expect clouds to rapidly provision resources in any quantity at any time. Clouds should be able to automatically scale resources up or down based on application load, provisioning additional resources as needed and releasing them when load decreases.
- 4. \*\*Customization\*\*:** In a multi-tenant cloud environment, users may have diverse needs. Cloud resources should be highly customizable to accommodate these varying requirements. For infrastructure services, this means allowing users to deploy specialized virtual appliances and granting privileged access to virtual servers. Other service classes like Platform as a Service (PaaS) and Software as a Service (SaaS) may offer less flexibility but should still provide some level of customization to meet user needs.

# Cloud Infrastructure Management - VIM

Managing physical and virtual resources, such as servers, storage, and networks, in a unified manner is a significant challenge for Infrastructure as a Service (IaaS) providers

This orchestration of resources must be performed rapidly and dynamically to provision resources to applications as needed. The software toolkit responsible for this orchestration is called a **Virtual Infrastructure Manager (VIM)**

**Virtualization Support:** Virtualization allows for the partitioning of a data center among multiple tenants by creating virtualized resources that can be dynamically sized and resized to meet varying requirements. This technology is ideal for accommodating diverse customer needs within a single hardware infrastructure.

**Self-Service, On-Demand Resource Provisioning:** Cloud users expect nearly instant access to resources without the need for human intervention. A Virtual Infrastructure Manager (VIM) provides a self-service interface, enabling users to request and customize services such as server creation and software configuration autonomously.

**Multiple Backend Hypervisors:** VIMs often support multiple virtualization technologies, providing a uniform management layer regardless of the underlying hypervisor. This flexibility, particularly evident in open-source VIMs, ensures compatibility with various virtualization models and tools.

**Storage Virtualization:** Abstracting logical storage from physical devices allows for the creation of virtual disks independent of device and location. While storage virtualization support is common in commercial products, some VIMs offer ways to pool and manage storage devices, albeit with varying levels of sophistication.

**Interface to Public Clouds:** VIMs may facilitate hybrid cloud setups by allowing users to manage leased resources from external cloud providers through a unified interface. This integration ensures transparency to applications utilizing resources from both local and remote environments.

**Virtual Networking:** VIMs support the creation and configuration of virtual networks, enabling the isolation of traffic and secure communication between virtual machines (VMs) across physical infrastructure boundaries.

**Dynamic Resource Allocation:** VIMs feature dynamic resource allocation capabilities to optimize resource utilization and meet application demands in real-time. This involves continuously monitoring resource utilization and reallocating resources among VMs as needed to maintain performance and energy efficiency.

**Virtual Clusters:** VIMs can manage groups of VMs to provision computing clusters on demand, facilitating the deployment of multi-tier Internet applications.

**Reservation and Negotiation Mechanism:** Advanced reservation mechanisms allow users to lease resources for specific time periods, while negotiation mechanisms enable flexible resource allocation to accommodate complex requests and ensure fairness in resource allocation.

**High Availability and Data Recovery:** VIMs provide high availability features to minimize application downtime, including failover mechanisms to detect and recover from physical and virtual server failures. Data recovery mechanisms ensure the protection and backup of VM images with minimal impact on system performance.

## IaaS – AMAZON

- \* Geographic Presence
- \* User Interface to access servers
- \* Advance Reservation of capacity
- \* Auto Scaling and Load Balancing
- \* Service Level Agreement for QoS
- \* Hypervisor and OS choice

Amazon Web Services (AWS) is a leading provider in the cloud computing market, offering a range of services including

- storage (S3),
- virtual servers (EC2),
- content delivery (Cloudfront),
- video streaming (Cloudfront Streaming),
- structured data storage (SimpleDB),
- relational databases (RDS),
- data processing (Elastic MapReduce).

The Elastic Compute Cloud (EC2) enables users to create Xen-based virtual servers known as instances from Amazon Machine Images (AMIs). These instances come in various sizes, operating systems, architectures, and prices, with CPU capacity measured in Amazon Compute Units. Users can attach nonpersistent disk space to instances, with the option of using Elastic Block Storage for persistent disk needs up to 1TB.

AWS achieves elasticity through features like CloudWatch, Auto Scaling, and Elastic Load Balancing, allowing for automatic scaling of instances based on customizable rules and distribution of traffic across instances.

Key features of Amazon EC2 include multiple data centers in the United States and Europe, various user interfaces including CLI, Web services, and web-based consoles, SSH and Remote Desktop access to instances, advanced reservation options for capacity, a 99.5% availability SLA, per-hour pricing, support for Linux and Windows operating systems, automatic scaling, and load balancing.

# PaaS

- \* Programming Models, Languages, Frameworks

P and J in GAE, .NET in MA

- \* Persistence Options

Allows to recover program failure

Store data

# UNIT 3

(295)

Service Oriented Architecture for Distributed Computing:  
Services & SOA,  
Message Oriented Middleware,  
Workflow in SOA.

Cloud Programming & Software Environments:  
Features of Cloud & Grid,  
Parallel & Distributed programming paradigms,  
Programming support of Google Cloud, Amazon AWS & Azure

## SERVICE ORIENTED ARCHITECTURE

Building systems in terms of services  
It has become a core idea of most distributed systems

### Definition:

- SOA focuses on designing software systems that utilize services
- Services: Published interfaces from applications
- Its goal is to enhance service interoperability
- It promotes loose coupling, published interfaces, and standard communication models
- HTTP, XML, REST

Distributed Systems Architecture with characteristics:

- Logical View: SOA provides abstracted, logical view of programs
- Message Orientation: abstracts implementation, interoperability
- Descriptive Orientation: service is described by metadata
- Granularity and Platform neutral (XML)

# REST (Systems of Systems)

- Representational State Transfer
- REST is a software architecture style for distributed systems
- Particularly distributed hypermedia systems, like World Wide Web

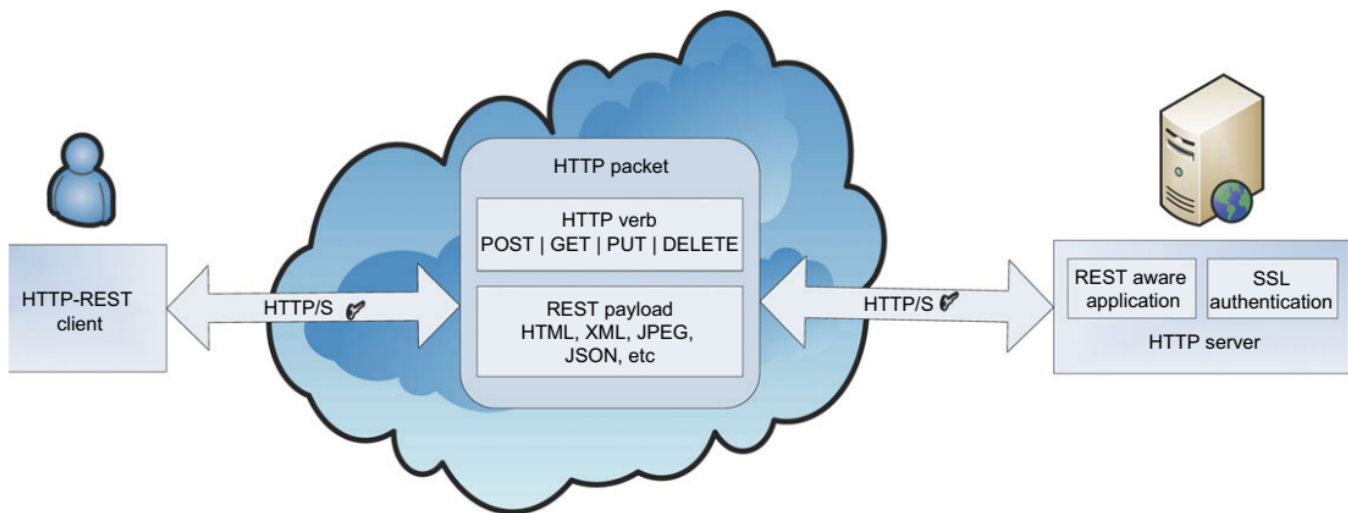


FIGURE 5.1

A simple REST interaction between user and server in HTTP specification.

(Courtesy of Thomas Fielding [2])

## Resource Identification through URIs

## Uniform, Constrained Interface: CRUD

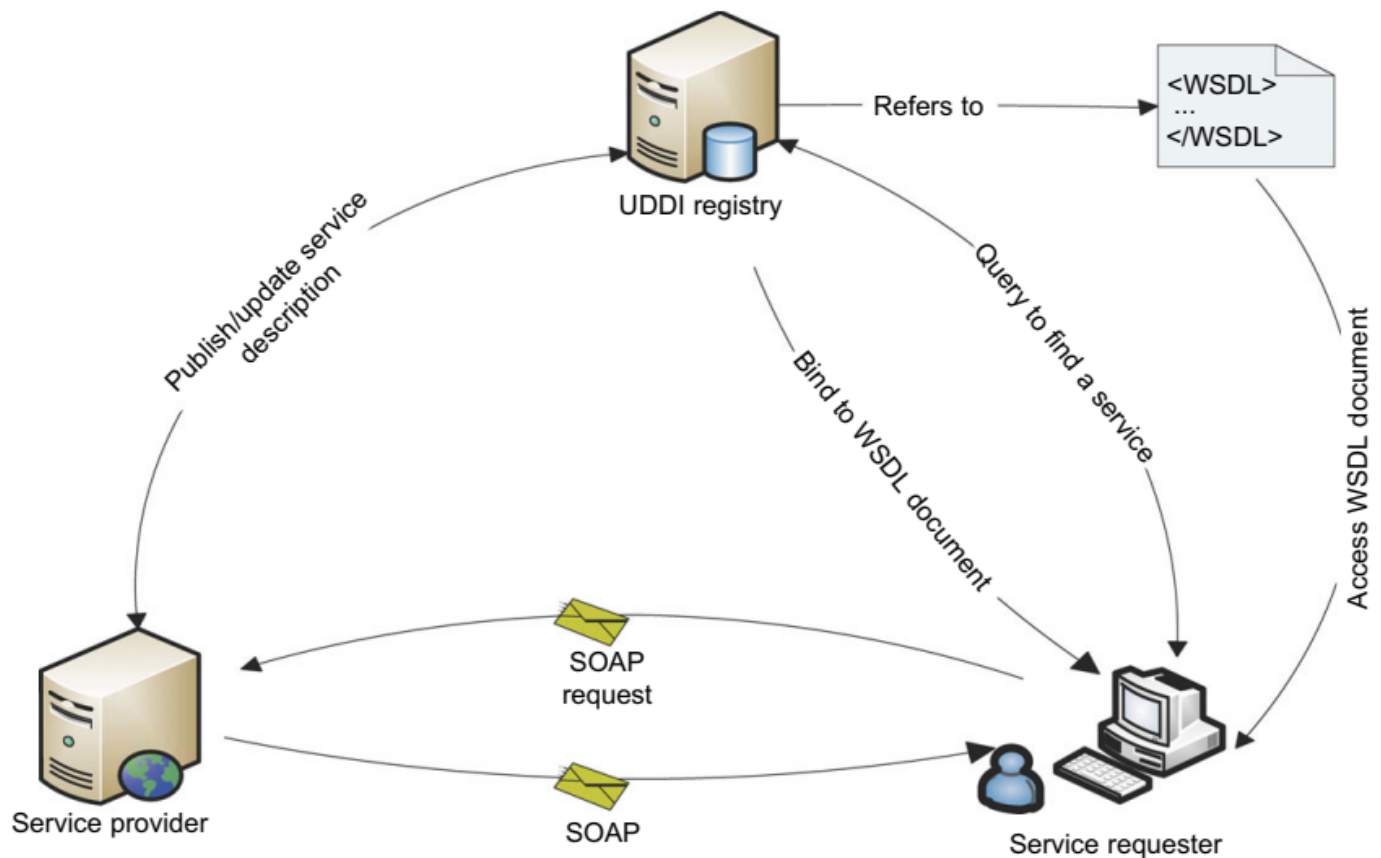
## Self-Descriptive Message:

## Stateless Interactions

Table 5.2 Sample REST Request-Response for Creating an S3 Bucket	
REST Request	REST Response
PUT/[bucket-name] HTTP/1.0 Date: Wed, 15 Mar 2011 14:45:15 GMT Authorization:AWS [aws-access-key-id]: [header-signature] Host: s3.amazonaws.com	HTTP/1.1 200 OK x-amz-id-2: VjzdTviQorQtSjcgLshzCZSzN+7CnewvHA +6sNxR3VRcUPyO5fmSmo8bWnlS52qa x-amz-request-id: 91A8CC60F9FC49E7 Date: Wed, 15 Mar 2010 14:45:20 GMT Location: /[bucket-name] Content-Length: 0 Connection: keep-alive

# SOA and Web Service

The term “web service” is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web



**FIGURE 5.2**

A simple web service interaction among provider, user, and the UDDI registry.

## Simple Object Access Protocol (SOAP)

SOAP provides a standard packaging structure for transmission of XML documents over various Internet protocols

Middleware systems can be interoperable  
SOAP message: envelope, header, body

## Web Services Description Language (WSDL)

WSDL describes the interface, a set of operations supported by a web service in a standard format. It enables disparate clients to automatically understand how to interact with a web service.

## Universal Description, Discovery, and Integration (UDDI)

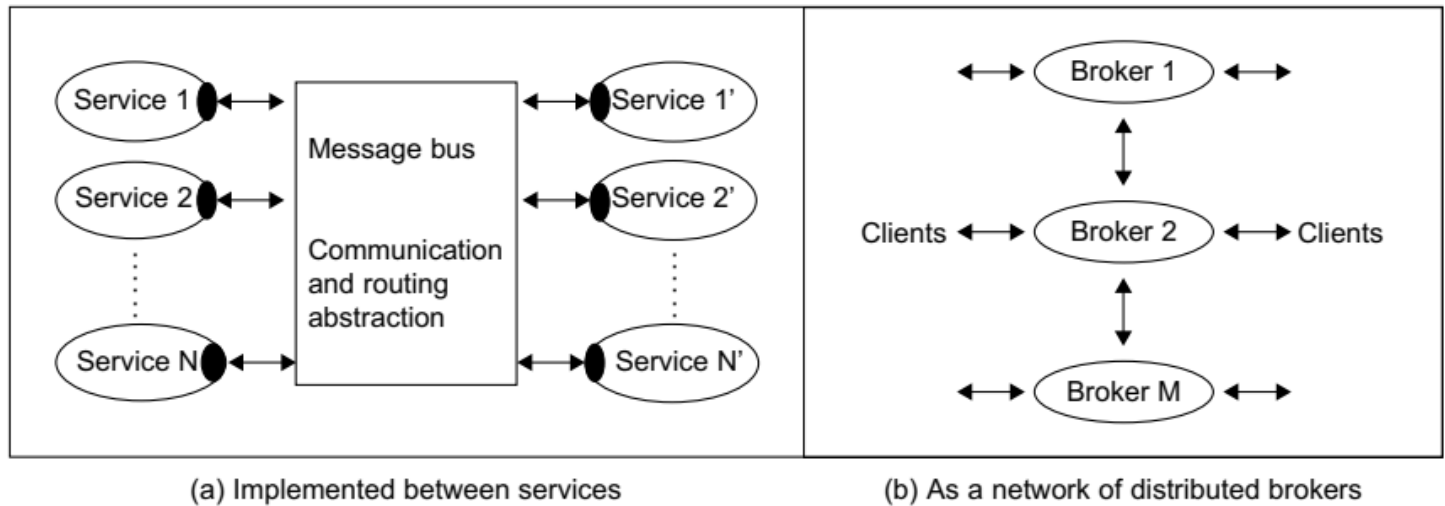
UDDI provides a global registry for advertising and discovery of web services, by searching for names, identifier



# Message Oriented Middleware

- enterprise bus
- publish-subscribe model
- queuing
- messaging systems

## Enterprise Bus



**FIGURE 5.6**

Two message bus implementations between services or using a broker network.

SOA services interact with messages of different formats (APIs, wire protocols, and transport mechanisms).

It is attractive to abstract the communication mechanism of services.

ESB refers to the case where the bus supports the convenient integration of many components, often in different styles.

One does not open a channel between source and destination, but rather injects a message into the bus with enough information to allow it to be delivered correctly

Brokers: Message Queues

## Publish Subscribe

It describes a model for linking source and dest for a message bus.

the producer of the message (publisher)

labels the message in some fashion (by topic names from vocabulary)

the receivers of the message (subscriber) will specify the topics for which they wish to receive associated messages.

The use of topic is termed message filtering.

There is many-to-many relationship between publishers and subscribers.

Publish-subscribe messaging middleware allows straightforward implementation of **notification** or **event**-based programming models.

## Queueing and Messaging systems

The best known is the Java Message Service (JMS) which specifies a set of interfaces outlining the communication semantics in pub/sub and queueing systems.

Advanced Message Queuing Protocol (AMQP) specifies the set of wire formats for communications; unlike APIs, wire formats are cross-platform.

In the web service arena, there are competing standards, WS-Eventing and WS-Notification, but neither has developed a strong following.

Key features:

Security approach, guarantees, mechanisms for message delivery.

Time-decoupled delivery

Fault tolerance

# Features of Cloud and Grid PLATFORMS

## Cloud Capabilities

## Grid Features

**Table 6.1** Important Cloud Platform Capabilities

Capability	Description
Physical or virtual computing platform	The cloud environment consists of some physical or virtual platforms. Virtual platforms have unique capabilities to provide isolated environments for different applications and users.
Massive data storage service, distributed file system	With large data sets, cloud data storage services provide large disk capacity and the service interfaces that allow users to put and get data. The distributed file system offers massive data storage service. It can provide similar interfaces as local file systems.
Massive database storage service	Some distributed file systems are sufficient to provide the underlying storage service application developers need to save data in a more semantic way. Just like DBMS in the traditional software stack, massive database storage services are needed in the cloud.
Massive data processing method and programming model	Cloud infrastructure provides thousands of computing nodes for even a very simple application. Programmers need to be able to harness the power of these machines without considering tedious infrastructure management issues such as handling network failure or scaling the running code to use all the computing facilities provided by the platforms.
Workflow and data query language support	The programming model offers abstraction of the cloud infrastructure. Similar to the SQL language used for database systems, in cloud computing, providers have built some workflow language as well as data query language to support better application logic.
Programming interface and service deployment	Web interfaces or special APIs are required for cloud applications: J2EE, PHP, ASP, or Rails. Cloud applications can use Ajax technologies to improve the user experience while using web browsers to access the functions provided. Each cloud provider opens its programming interface for accessing the data stored in massive storage.
Runtime support	Runtime support is transparent to users and their applications. Support includes distributed monitoring services, a distributed task scheduler, as well as distributed locking and other services. They are critical in running cloud applications.
Support services	Important support services include data and computing services. For example, clouds offer rich data services and interesting data parallel execution models like MapReduce.

# MAP REDUCE

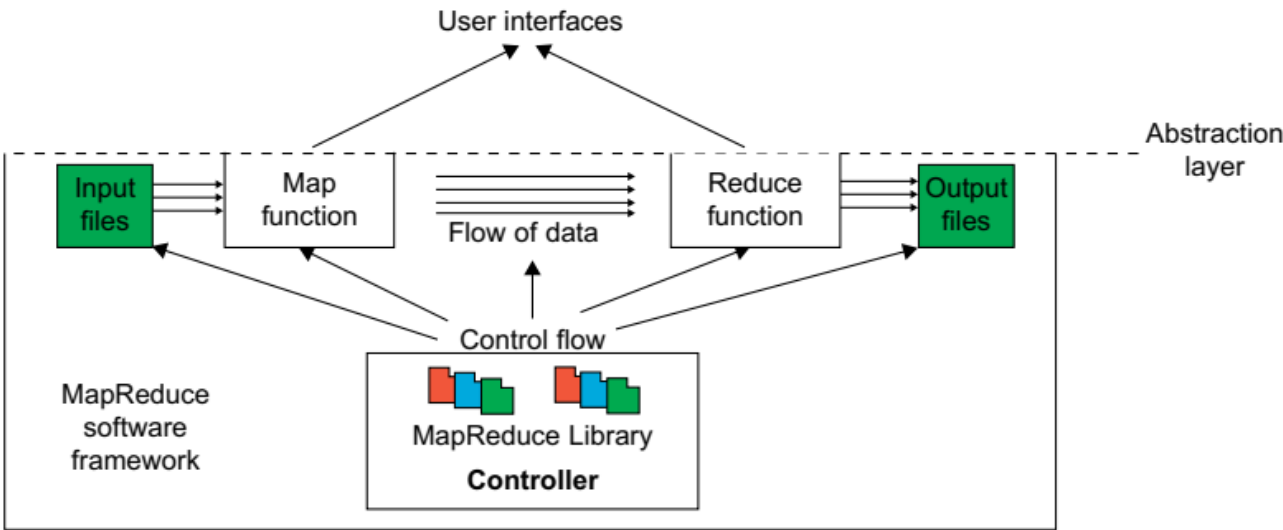


FIGURE 6.1

MapReduce framework: Input data flows through the Map and Reduce functions to generate the output result under the control flow using MapReduce software library. Special user interfaces are used to access the Map and Reduce resources.

### 6.2.2.3 Formal Notation of MapReduce Data Flow

The *Map* function is applied in parallel to every input (key, value) pair, and produces new set of intermediate (key, value) pairs [37] as follows:

$$(key_1, val_1) \xrightarrow{\text{Map Function}} \text{List}(key_2, val_2) \tag{6.1}$$

Then the MapReduce library collects all the produced intermediate (key, value) pairs from all input (key, value) pairs, and sorts them based on the “key” part. It then groups the values of all occurrences of the same key. Finally, the *Reduce* function is applied in parallel to each group producing the collection of values as output as illustrated here:

$$(key_2, \text{List}(val_2)) \xrightarrow{\text{Reduce Function}} \text{List}(val_2) \tag{6.2}$$

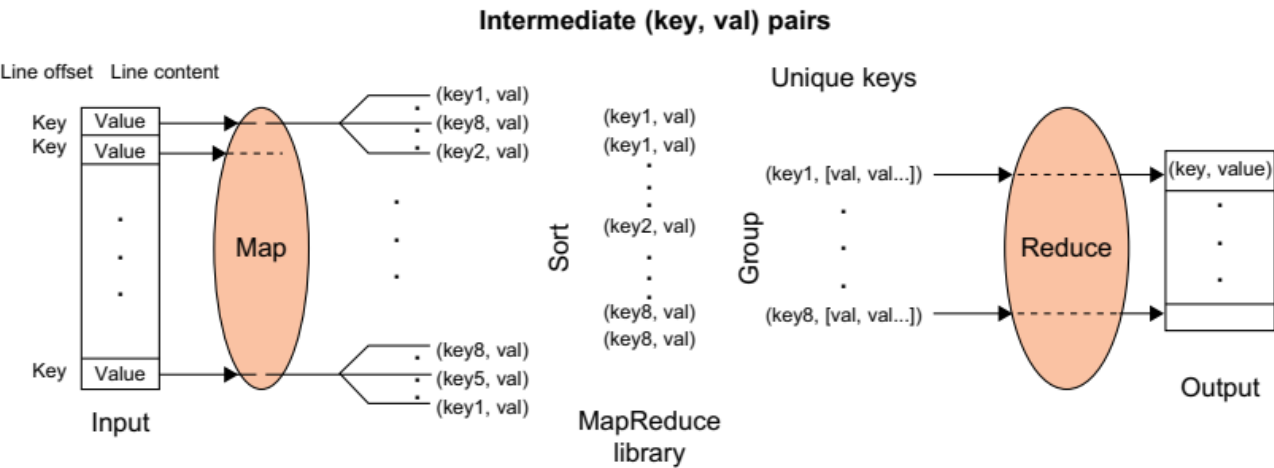
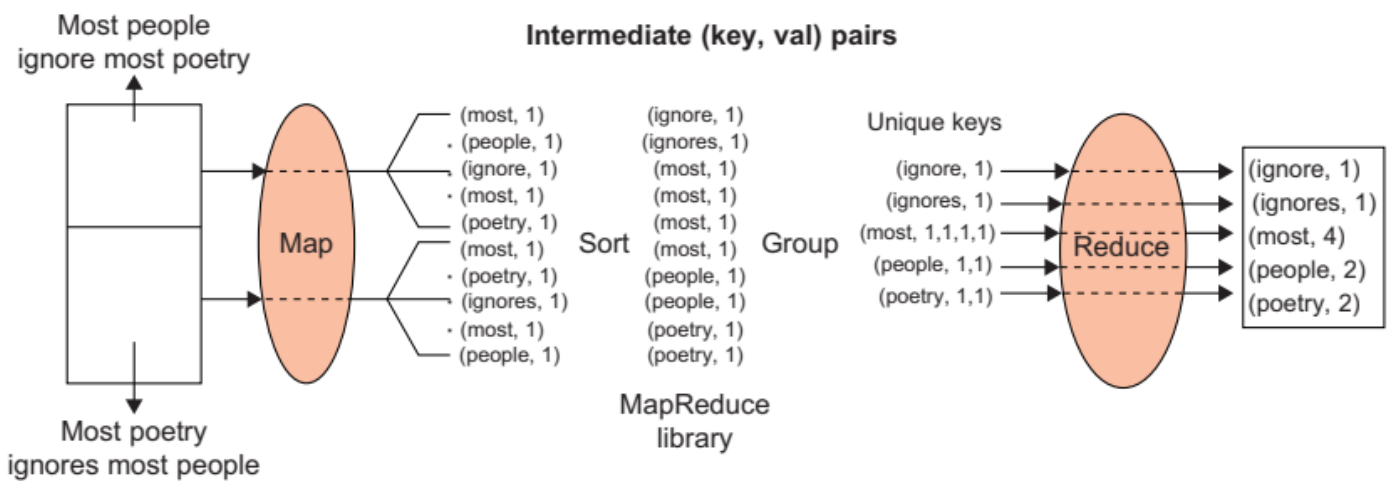


FIGURE 6.2

MapReduce logical data flow in 5 processing stages over successive (key, value) pairs.



- 1. \*\*Data Partitioning\*\*:** Input data stored in the Google File System (GFS) is split into M pieces, corresponding to the number of map tasks.
- 2. \*\*Computation Partitioning\*\*:** Users write programs in the form of Map and Reduce functions. The MapReduce library distributes copies of the user program containing these functions to available computation engines.
- 3. \*\*Master and Workers\*\*:** The MapReduce architecture follows a master-worker model. One copy of the user program becomes the master, while the rest become workers. The master assigns map and reduce tasks to idle workers.
- 4. \*\*Reading Input Data\*\*:** Each map worker reads its portion of the input data and sends it to its Map function.
- 5. \*\*Map Function\*\*:** Map functions process input data splits and produce intermediate (key, value) pairs.
- 6. \*\*Combiner Function\*\*:** An optional function within the map worker that merges local data before sending it over the network, reducing communication costs.
- 7. \*\*Partitioning Function\*\*:** Intermediate (key, value) pairs are grouped based on keys to ensure all pairs with identical keys are processed by the same Reduce function. These pairs are partitioned into R regions by the Partitioning function.

**8. \*\*Synchronization\*\*:** MapReduce coordinates map workers with reduce workers through a simple synchronization policy, where communication starts after all map tasks finish.

Here are steps 9, 10, and 11 of the communication and processing flow in the MapReduce framework as described in the text:

**9. \*\*Communication\*\*:**

**10. \*\*Sorting and Grouping\*\*:**

**11. \*\*Reduce Function\*\*:**

The reduce worker iterates over the grouped (key, value) pairs. For each unique key, it sends the key and corresponding values to the Reduce function. The Reduce function processes the input data and stores the output results in predetermined files in the user's program.

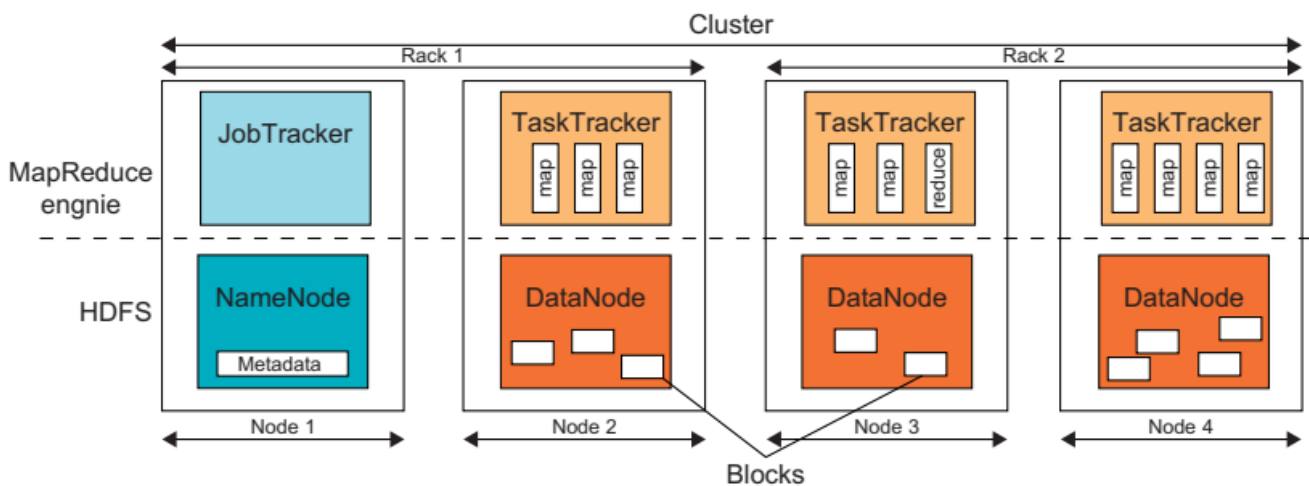
# HADOOP

- By Apache, in Java
- Open Source implementation of Map Reduce

- 1) Map Reduce Engine
- 2) HDFS as File System instead of GFS

## 6.2.3.1 Architecture of MapReduce in Hadoop

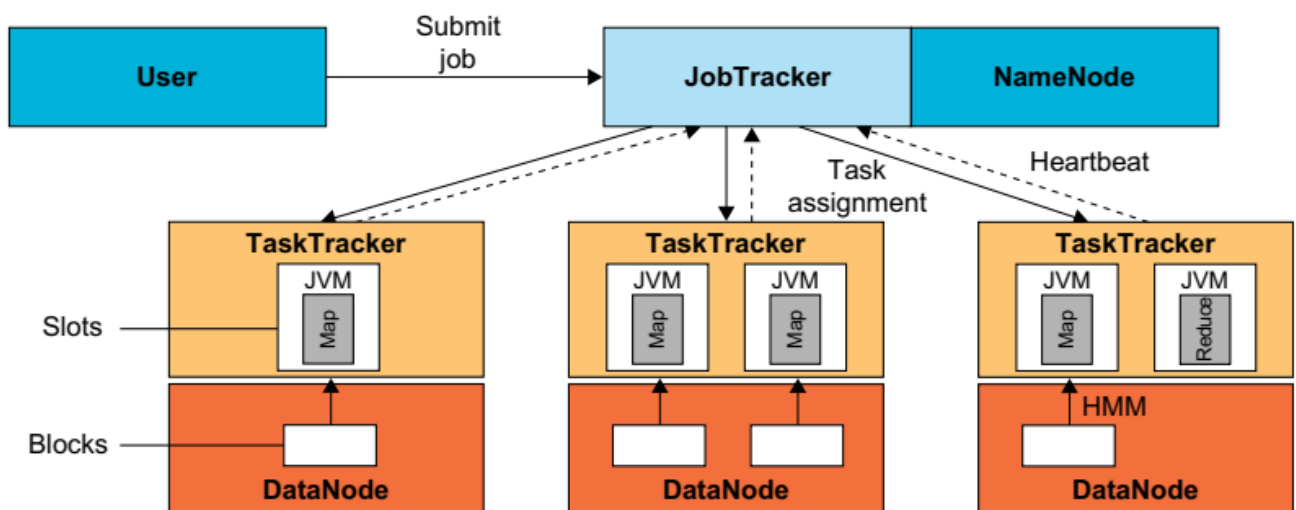
The topmost layer of Hadoop is the MapReduce engine that manages the data flow and control flow of MapReduce jobs over distributed computing systems. [Figure 6.11](#) shows the MapReduce engine architecture cooperating with HDFS. Similar to HDFS, the MapReduce engine also



**FIGURE 6.11**

HDFS and MapReduce architecture in Hadoop where boxes with different shadings refer to different functional nodes applied to different blocks of data.

- Master Slave architecture

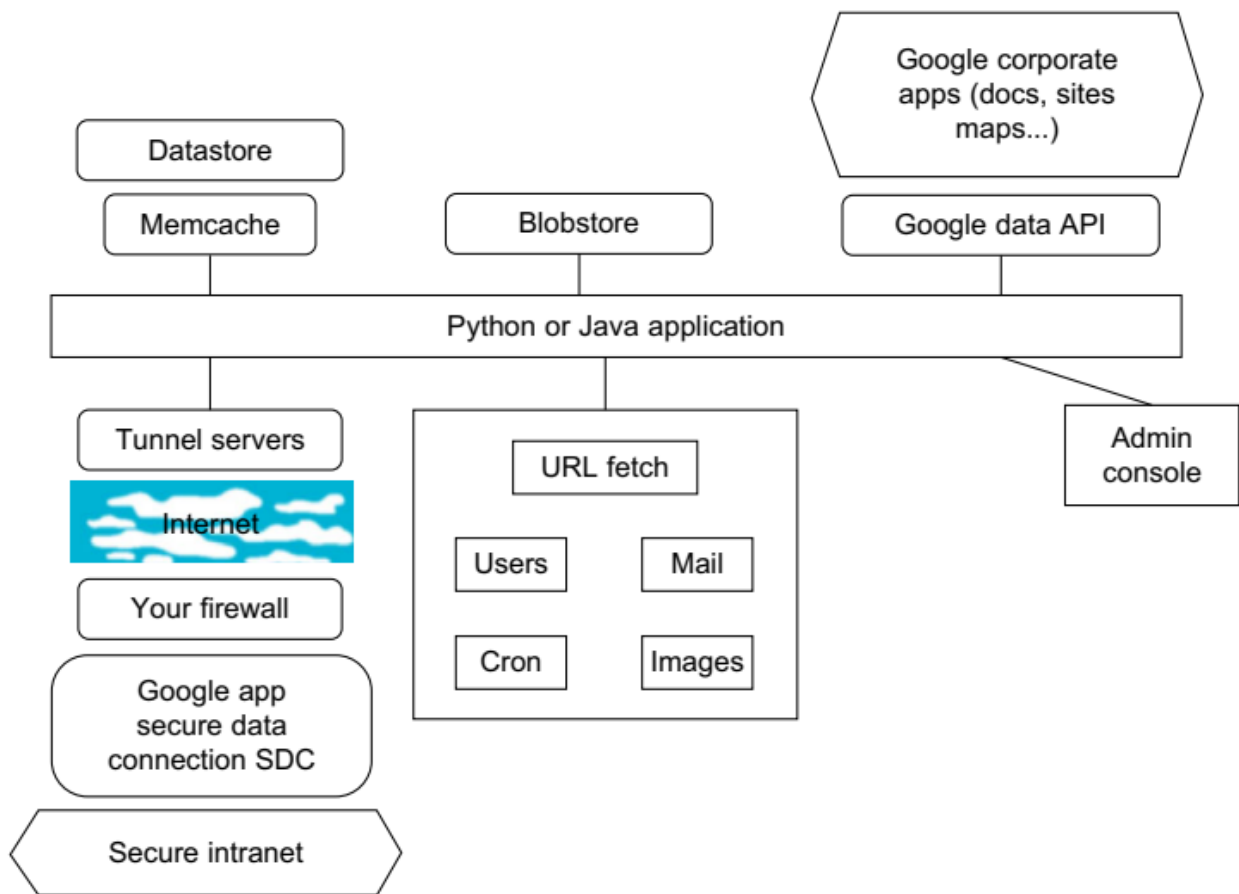


**FIGURE 6.12**

Data flow in running a MapReduce job at various task trackers using the Hadoop library.

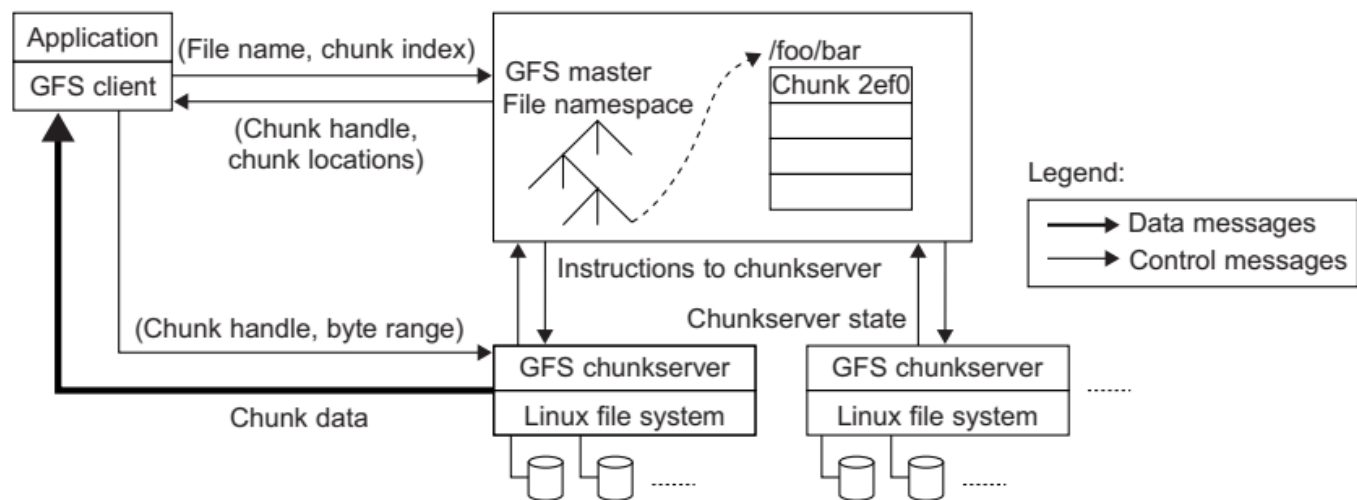
# PROGRAMMING SUPPORT

## GOOGLE APP ENGINE



**FIGURE 6.17**  
Programming environment for Google AppEngine.

## GFS



**FIGURE 6.18**  
Architecture of Google File System (GFS).



1. **Purpose and Background**: GFS was developed as the primary storage service for Google's search engine to handle massive amounts of web data. Traditional file systems couldn't meet the requirements of redundancy and scalability needed by Google.

2. **Assumptions**: GFS is designed to handle high component failure rates common in cloud computing hardware. It stores large files, often over 100 MB in size, with special I/O patterns including mostly appending operations and large streaming reads.

3. **Design Decisions**:

- **Block Size**: GFS uses a 64 MB block size instead of the typical 4 KB in traditional file systems.

- **Reliability**: Data is replicated across multiple chunk servers to ensure reliability.

- **Single Master**: A single master node coordinates access and stores metadata, simplifying cluster management.

4. **API and Interface**: GFS provides a POSIX-like file system interface with some differences, such as allowing applications to see the physical location of file blocks. The customized API includes operations like snapshot and record append to support Google applications.

5. **Architecture**: The GFS architecture consists of a single master node for metadata and chunk servers for data storage. The master manages the file system namespace and communicates with chunk servers for tasks like load balancing and fail recovery.

6. **Single Point of Failure Mitigation**: To address the potential weakness of a single master being a performance bottleneck or point of failure, Google uses a shadow master to replicate master data. Data operations are performed directly between clients and chunk servers, reducing reliance on the master for data transfer.

7. **Scalability**: Despite the single master design, GFS can handle clusters of more than 1,000 nodes with the current quality of commodity servers.

6.4.1 Programming on Amazon EC2

Amazon was the first company to introduce VMs in application hosting. Customers can rent VMs instead of physical machines to run their own applications. By using VMs, customers can load any software of their choice. The elastic feature of such a service is that a customer can create, launch, and terminate server instances as needed, paying by the hour for active servers. Amazon provides several types of preinstalled VMs. Instances are often called *Amazon Machine Images (AMIs)* which are preconfigured with operating systems based on Linux or Windows, and additional software.

Table 6.12 defines three types of AMI. Figure 6.24 shows an execution environment. AMIs are the templates for instances, which are running VMs. The workflow to create a VM is

Create an AMI → Create Key Pair → Configure Firewall → Launch

(6.3)

This sequence is supported by public, private, and paid AMIs shown in Figure 6.24. The AMIs are formed from the virtualized compute, storage, and server resources shown at the bottom of Figure 6.23.

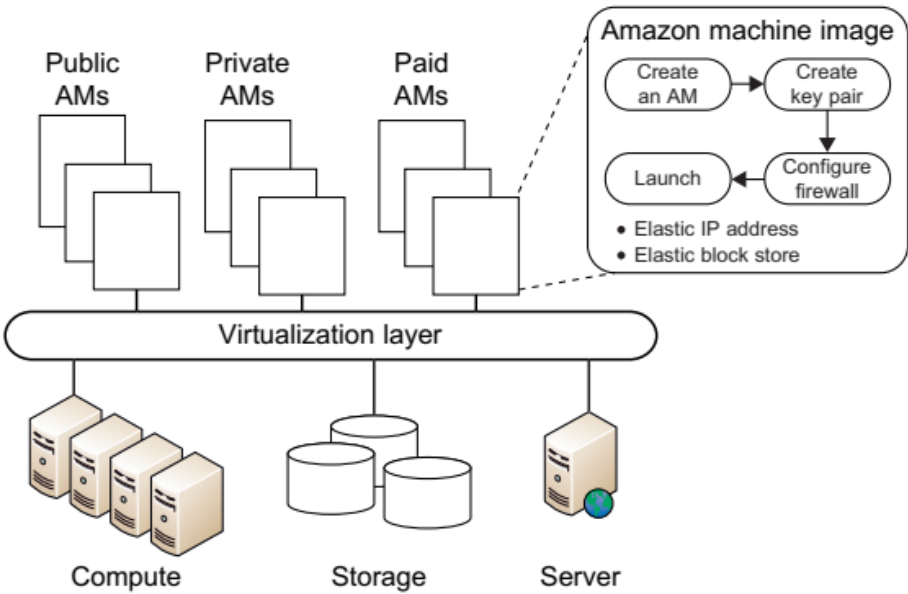


FIGURE 6.23  
Amazon EC2 execution environment.

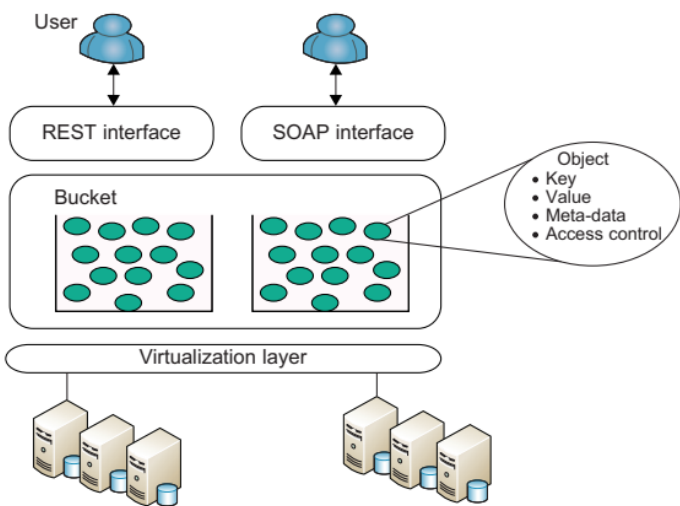
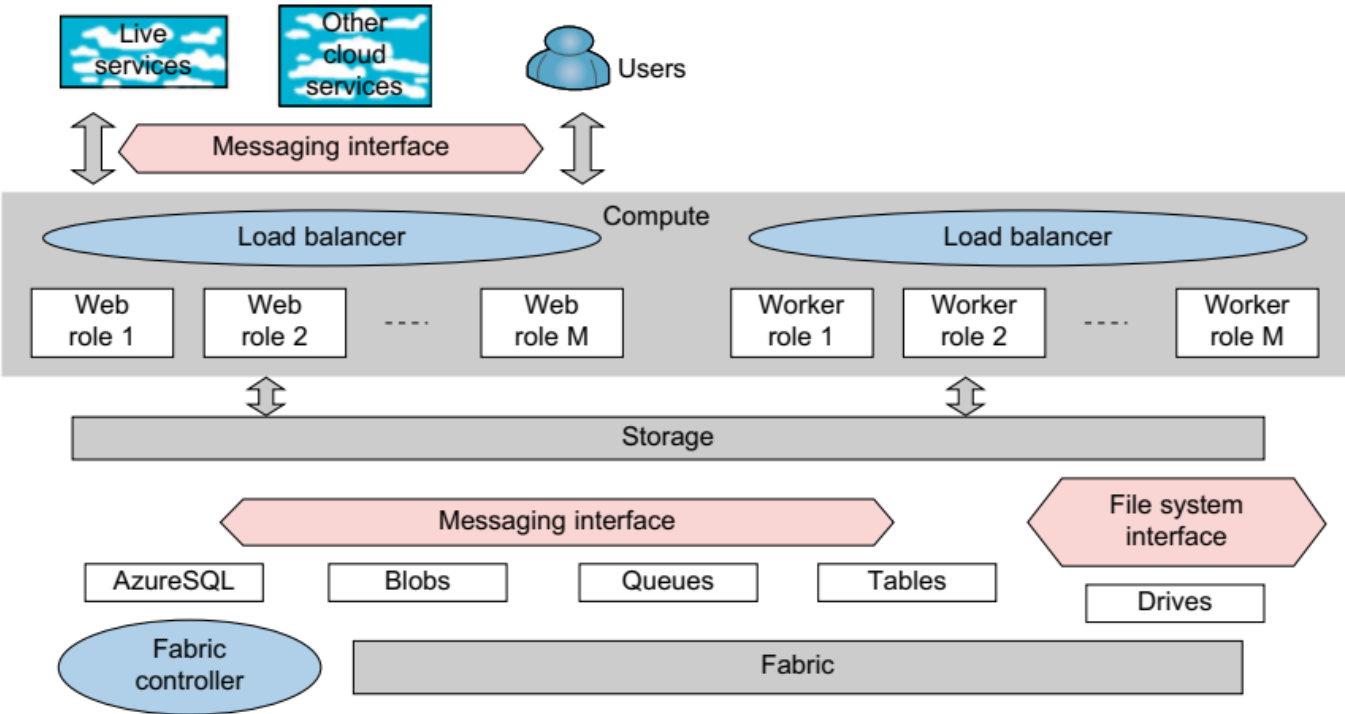


FIGURE 6.24  
Amazon S3 execution environment.

# Microsoft Azure



**FIGURE 6.25**

Features of the Azure cloud platform.

# UNIT 4

(152) Virtual Machines and Virtualization of Cluster and Data Centres

Levels of Virtualization,

Virtualization structures/Tools and Mechanism,

Virtualization of CPU, Memory and I/O Devices,

Virtual Clusters and Resources Management,

Virtualization Data-Centre Automation,

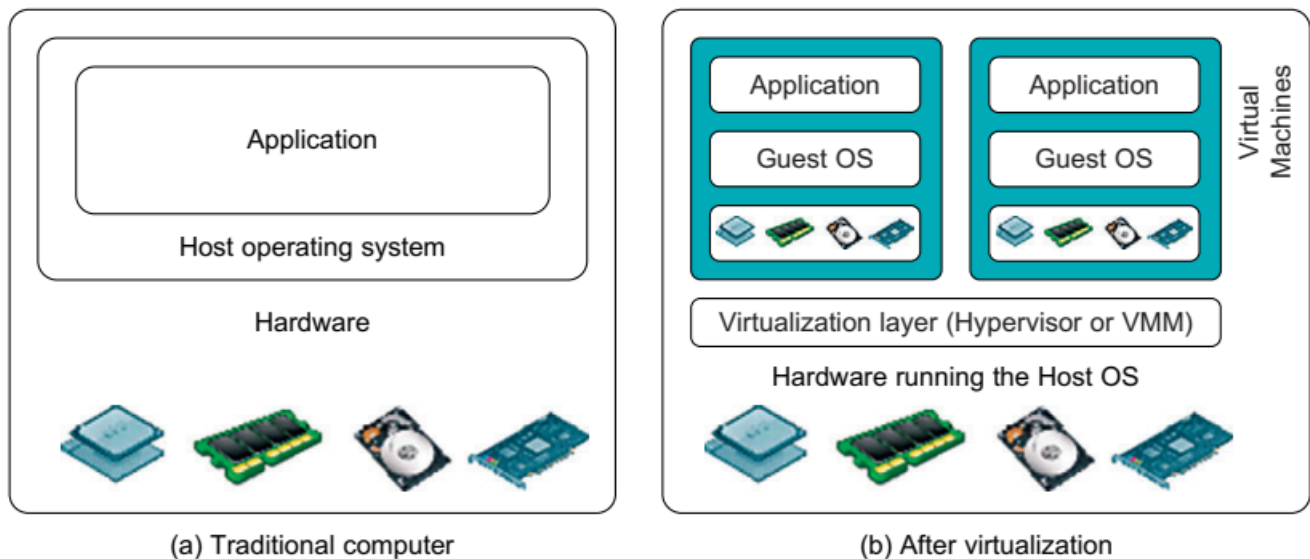
Virtual Machine Migration Services,

VM Provisioning and Migration in Action

## VIRTUALIZATION

- Computer Architecture technology
  - Multiple VMs can be multiplexed in the same hardware machine
  - Enhance resource sharing by many users
  - Hardware resources (CPU, Memory, I/O devices)
  - Software resources (OS and Software Libraries)
- can be virtualized in various functional layers

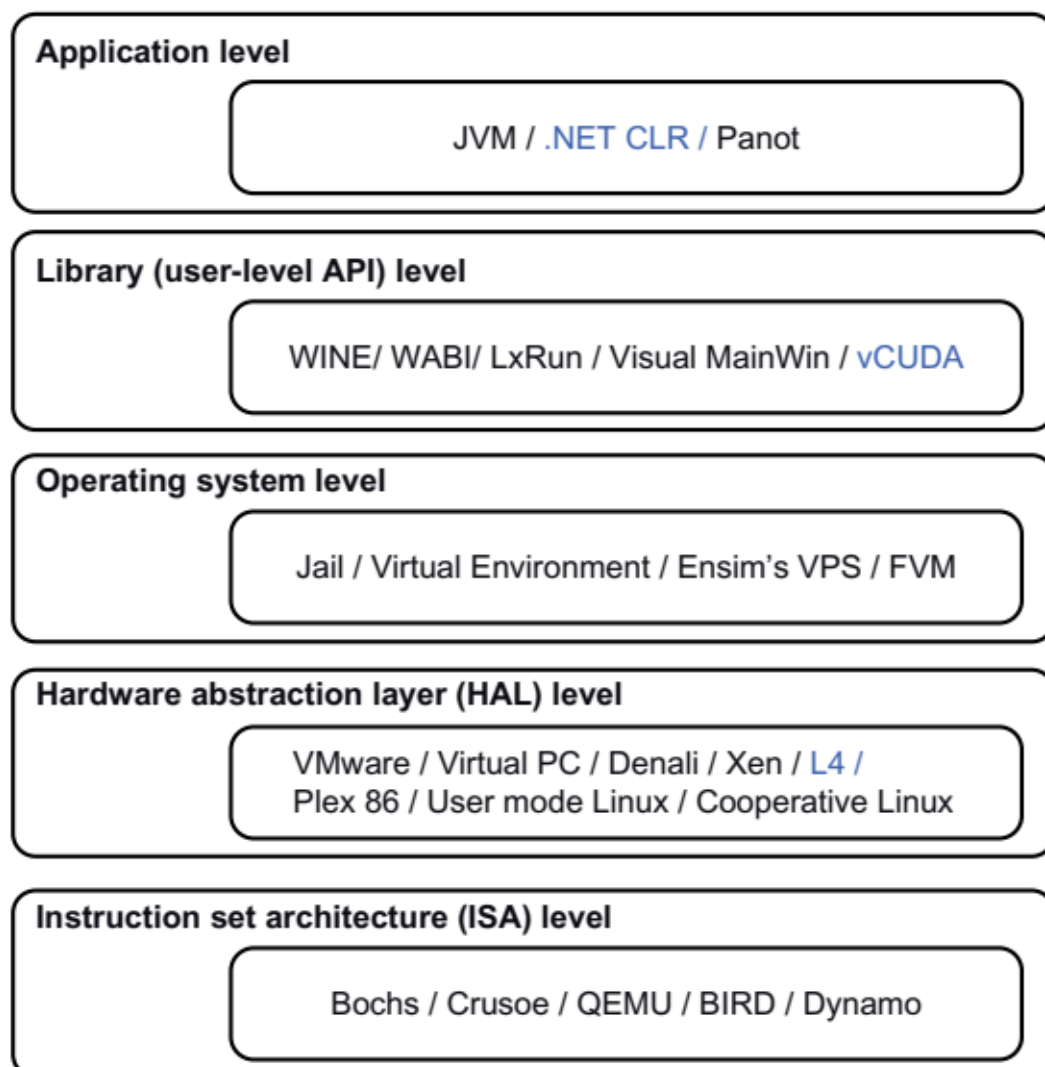
# Levels of VIRTUALIZATION - Implementation



**FIGURE 3.1**

The architecture of a computer system before and after virtualization, where VMM stands for virtual machine monitor.

The main function of the virtualization layer is to virtualize the physical hardware of a host machine into virtual resources for VMs



## 1) ISA Level

- V is performed by emulating a given ISA by the ISA of the host
- MIPS binary code can run on an x86-based host machine
- Run many legacy binary code written for various processors
- The basic emulation method is through **code interpretation**
- Program interprets the source instructions to target instructions
- **Dynamic Binary Translation**: Blocks of SI to TI

## 2) HAL

- Performed right on top of bare hardware
- Generates virtual hardware environment for VM
- Idea is to virtualize computer resources (CPU, M, IO)
- Eg: XEN Hypervisor, virtualizes x86 based machines

## 3) OSL

- Abstraction between traditional OS and Application
- Creates isolated "containers" on a single physical server
- Containers act like a real server
- Used to create virtual hosting environments

## 4) LSL

- Most applications use APIs exported by user-level libraries
- Rather than using lengthy system calls by the OS
- Most systems: provide well documented API
- They are used for virtualization
- Eg: WINE, vCUDA

## 5) UAL

- Virtualizes an application as a VM
- Also called Process level Virtualization
- Approach: Deploy HLL VMs
- Eg: JVM, .NET CLR
- Application sandboxing

**Table 3.1** Relative Merits of Virtualization at Various Levels (More "X"'s Means Higher Merit, with a Maximum of 5 X's)

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

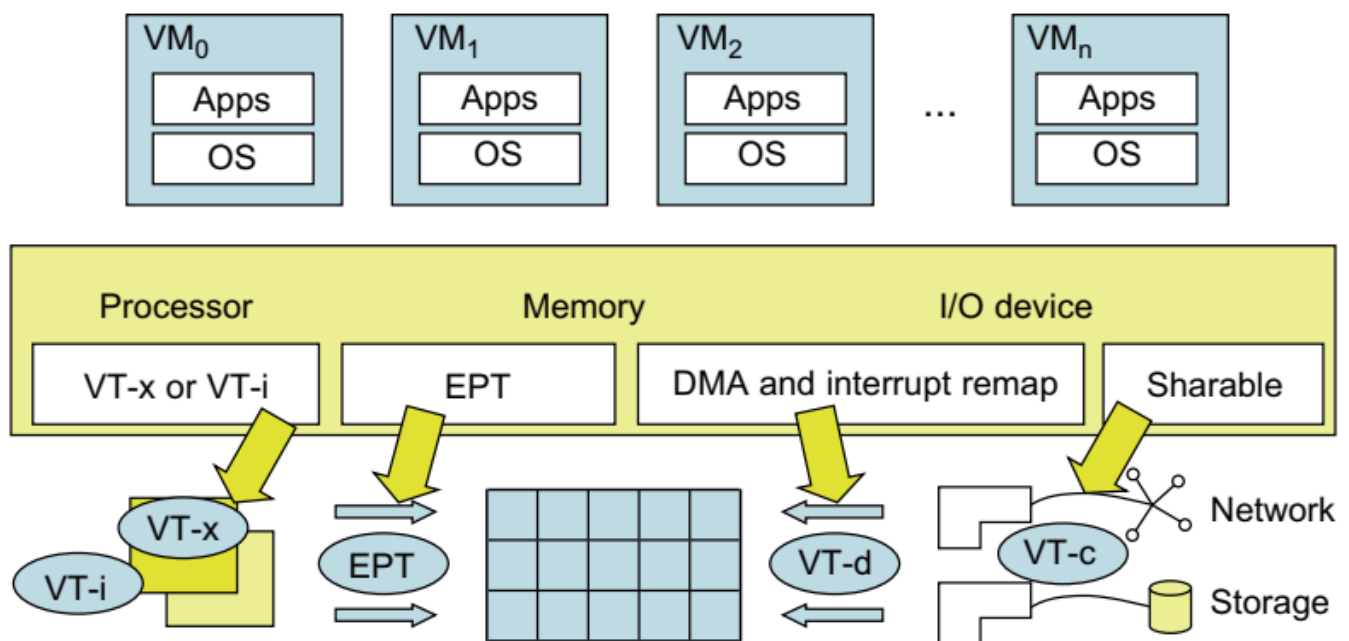
# VIRTUALIZATION of CPU

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization

Modern processors permit multiple processes to run simultaneously. If there is no protection, all instructions from different processes can cause a system crash.

Therefore, all processors have at least two modes,

- user mode: privileged instructions
  - supervisor mode: unprivileged
- to ensure controlled access of critical hardware.



For processor virtualization, Intel offers the VT-x or VT-i technique.

- VT-x adds a privileged mode
- This enhancement traps all sensitive instructions in the VMM

For memory virtualization, Intel offers the EPT,

- which translates virtual address to machine's physical addresses
- this improves performance.

For I/O virtualization, Intel implements VT-d and VT-c to support this.

# CPU Virtualization

- Unprivileged instructions of VMs are executed directly on the host
- Critical Instructions
  - Privileged: execute in special mode
  - Control sensitive: change configuration of resources
  - Behaviour sensitive: different behaviour based on configuration

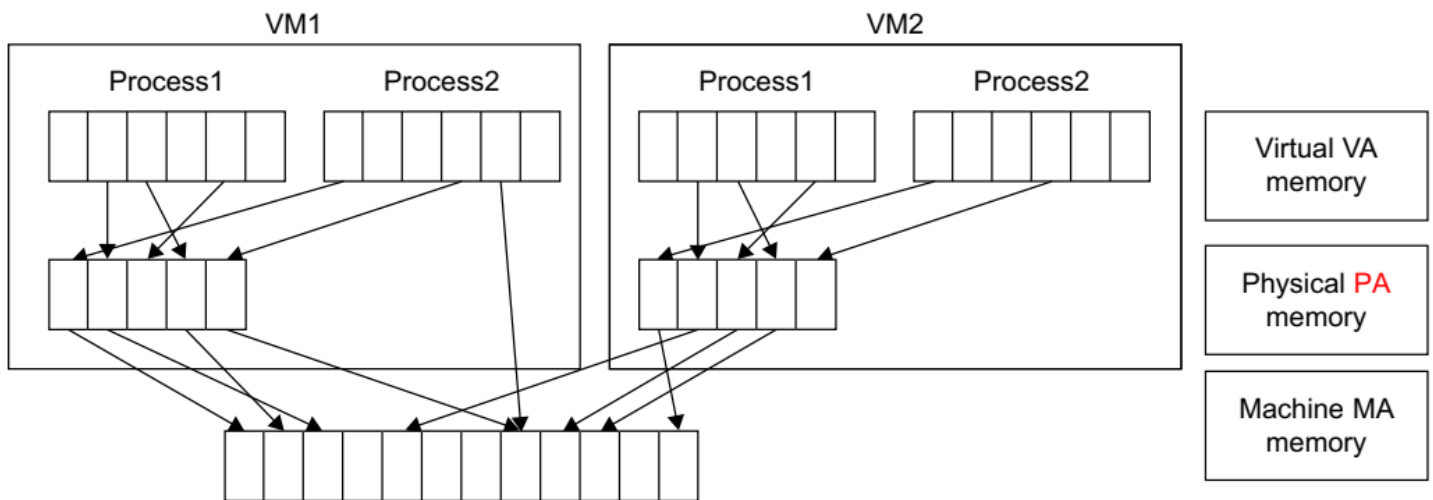
Some CPU architecture are full virtualizable

- VMM runs in supervisor mode
- Critical instructions when run are trapped in VMM
- VMM acts as mediator from different VMs
- RISC \_/ - x86 X
- Paravirtualization (XEN)
  - 80h interrupt in guest OS, and 82h in hypervisor
  - Control is passed to hypervisor to execute the instruction
- Hardware assisted CPU V
  - Privilege Mode – Ring-1 to x86 processor
  - OS in Ring-0
  - Hypervisor in Ring-1



## MEMORY Virtualization

- Similar to memory virtualization support by OS
- Mapping of virtual memory to machine memory using page tables
- One step mapping process
- Two stage mapping process by guest OS and VMM
- Virtual Memory -> Physical Memory
- Physical Memory -> Machine Memory

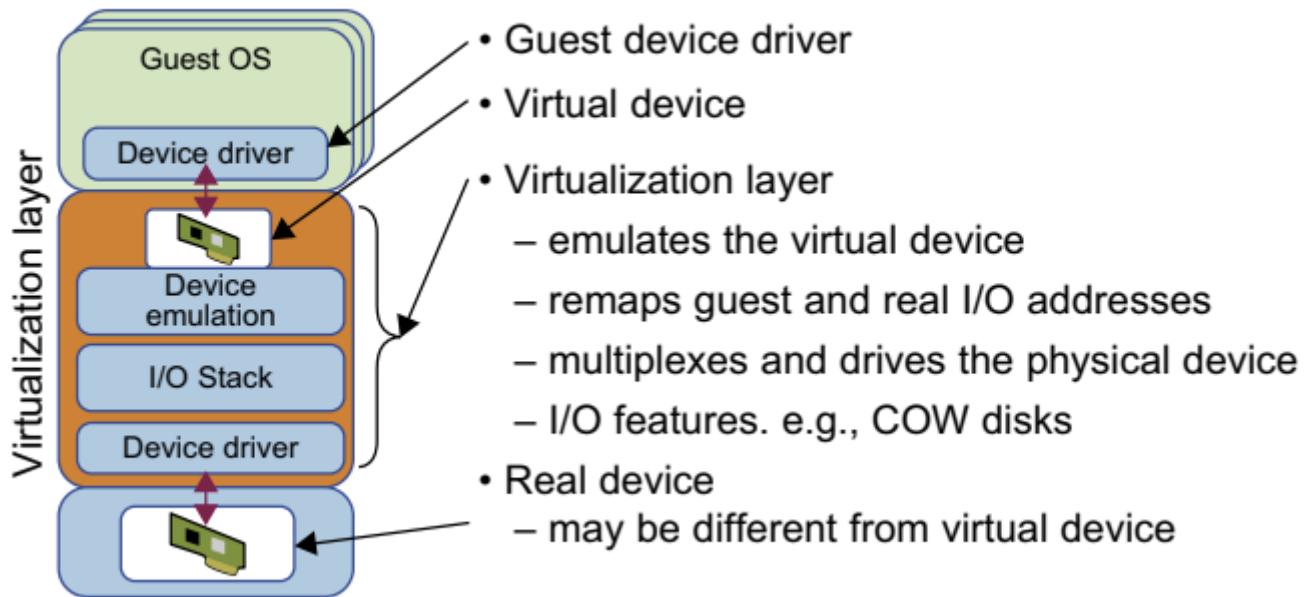


**FIGURE 3.12**

Two-level memory mapping procedure.

# I/O Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware



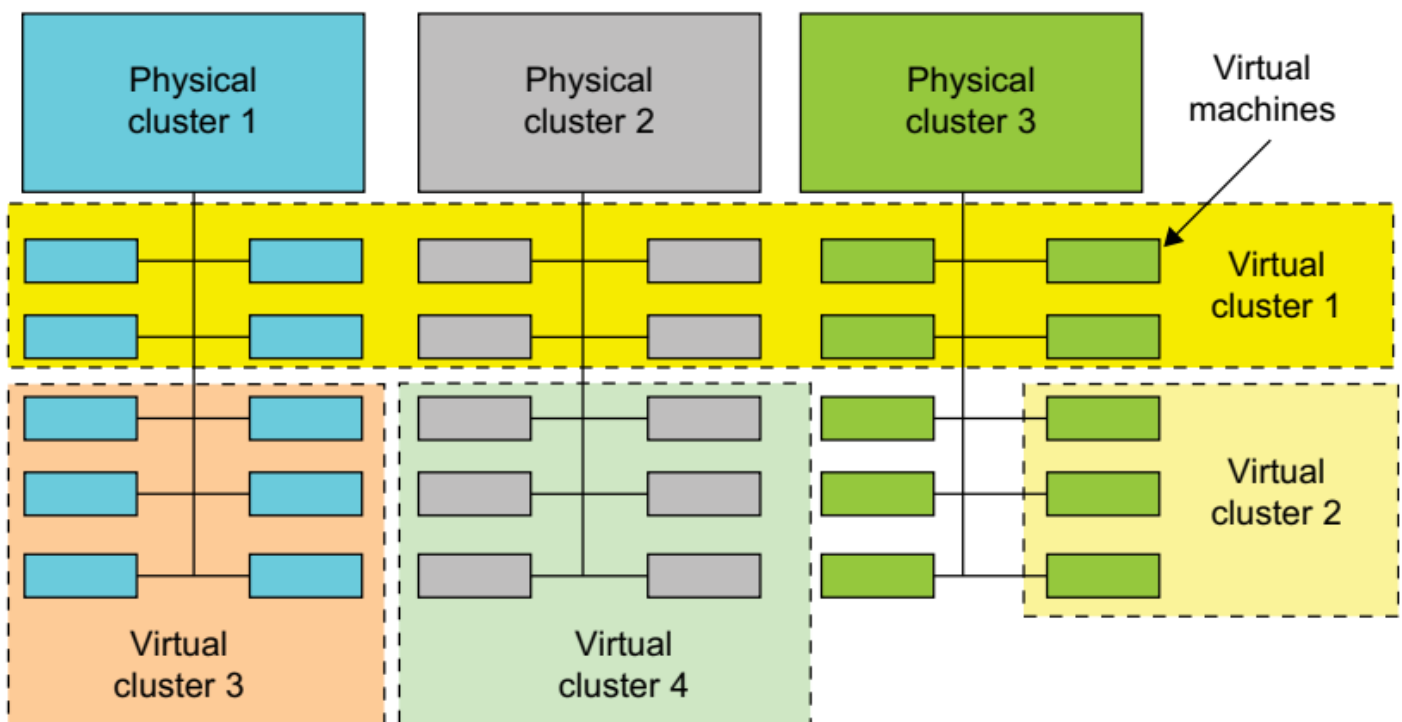
# VIRTUAL Clusters and Resource Management

Physical cluster is a collection of servers (physical machines) interconnected by a physical network such as a LAN

When a VM is initialized, the administrator needs to manually write configuration information. When more VMs join a network, an inefficient configuration always causes problems.

Amazon EC2, XenServer support BRIDGING mode

Through this, VMs can communicate with one another



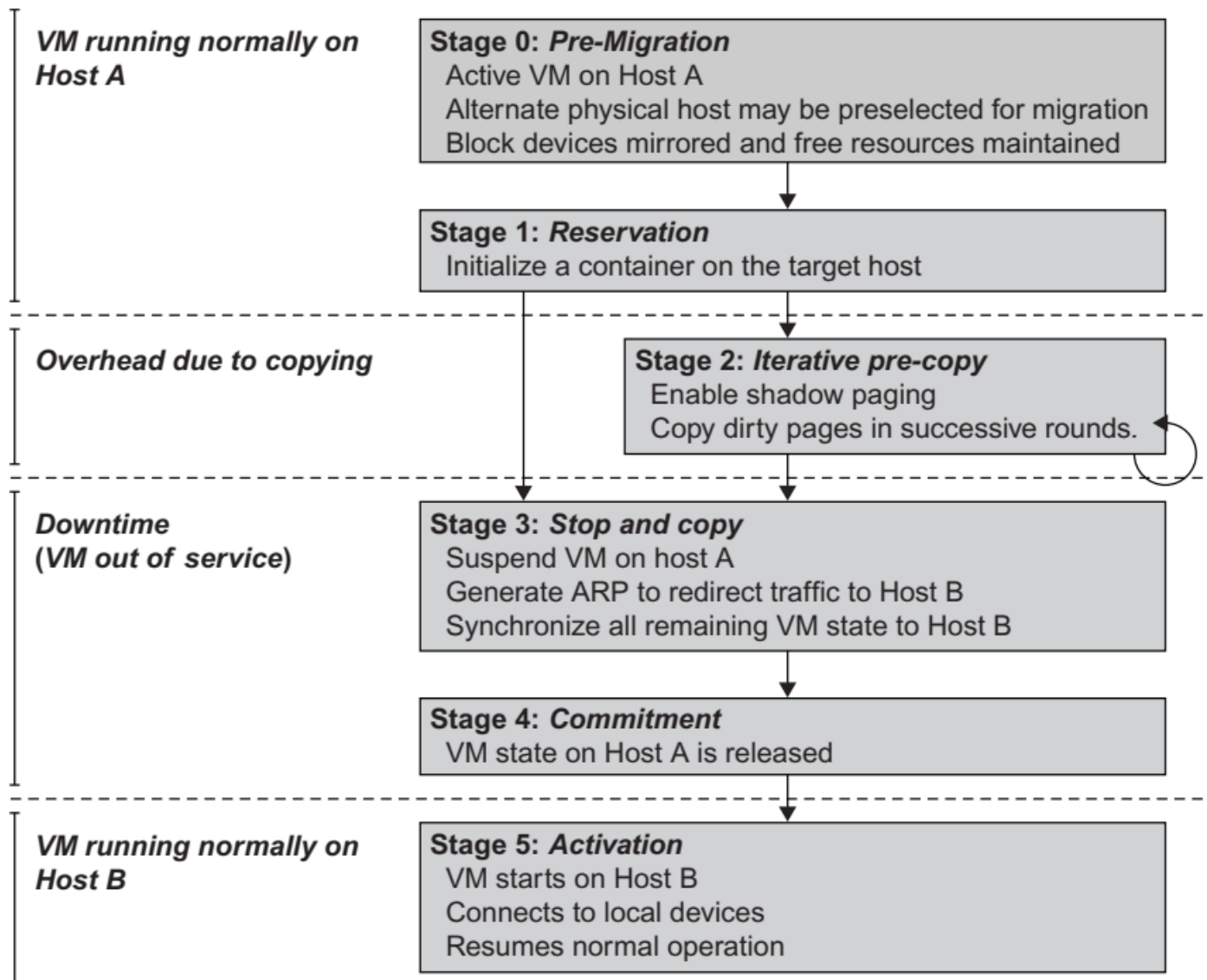
3 Design problems:

Live Migration

Memory, File System, Network Migration

Dynamic Deployment of Clusters

# 1) Live Migration



In a cluster, all operations are to be run on physical machine. When a VM fails, its role could be replaced by another VM on a different node

VMs can be live-migrated from one physical machine to another. This gives enhanced failover flexibility

Inactive state: V not enabled

Active state: V is instantiated, to perform a task

Paused state: instantiated but disabled to perform a task

Suspended state: if its files are stored back to disk

## 1) Start Migration

- Prepares for migration
- Identifies the source VM and destination VM
- Manual or Automatic (by load balancer)

## 2) Transfer the Memory

- Execution state of VM is in memory
- It is transferred, which provides continuity of services
- All memory data, then memory data that was changed
- This is iteratively copied until the dirty bit is small
- While copying, execution of program is not interrupted

## 3) Suspend the VM and Copy the last portion of the data

- VM is stopped, applications won't be run
- Downtime
- Last portion of the data, CPU and Network states are transferred

## 4) Commit and Activate the new host

- VM reloads the State
- Recovers the execution of the program
- Service provided by the VM continues

## Memory Migration

- When there's power, space, leasing, cost overhead to be considered.
- Memory instance of VM from one host to another
- ISR (internet suspend-resume) exploits temporal locality
- Tree of files
- When downtime is not an issue

## Files System Migration

- Each VM with a virtual disk which contains file system
- Global file system across all VMs
- Smart copying, exploits spatial locality

## Network Migration

- Should maintain network connections when migrated
- VM has its own IP, MAC address

# **VIRTUALIZATION for DATA-CENTER Automation**

## **Server Consolidation in Data Centers**

- Data Centers have varying workloads
- SC is a strategy to improve hardware resource utilization
- by reducing number of physical servers
- Virtualization help achieve it
- It also reduces total cost, and improves availability

# UNIT 5

## **DESIGNING DISTRIBUTED SYSTEMS: GOOGLE CASE STUDY:**

Introducing the case study:

Google Overall architecture and design philosophy

Underlying communication paradigms,

Data storage and coordination services

Distributed computation services

- Overall architecture of Search Engine
- Different Communication paradigm (discuss)
- GFS Architecture and requirements
- MapReduce and Sawzall program execution
- Summary of Distributed Computing design choices

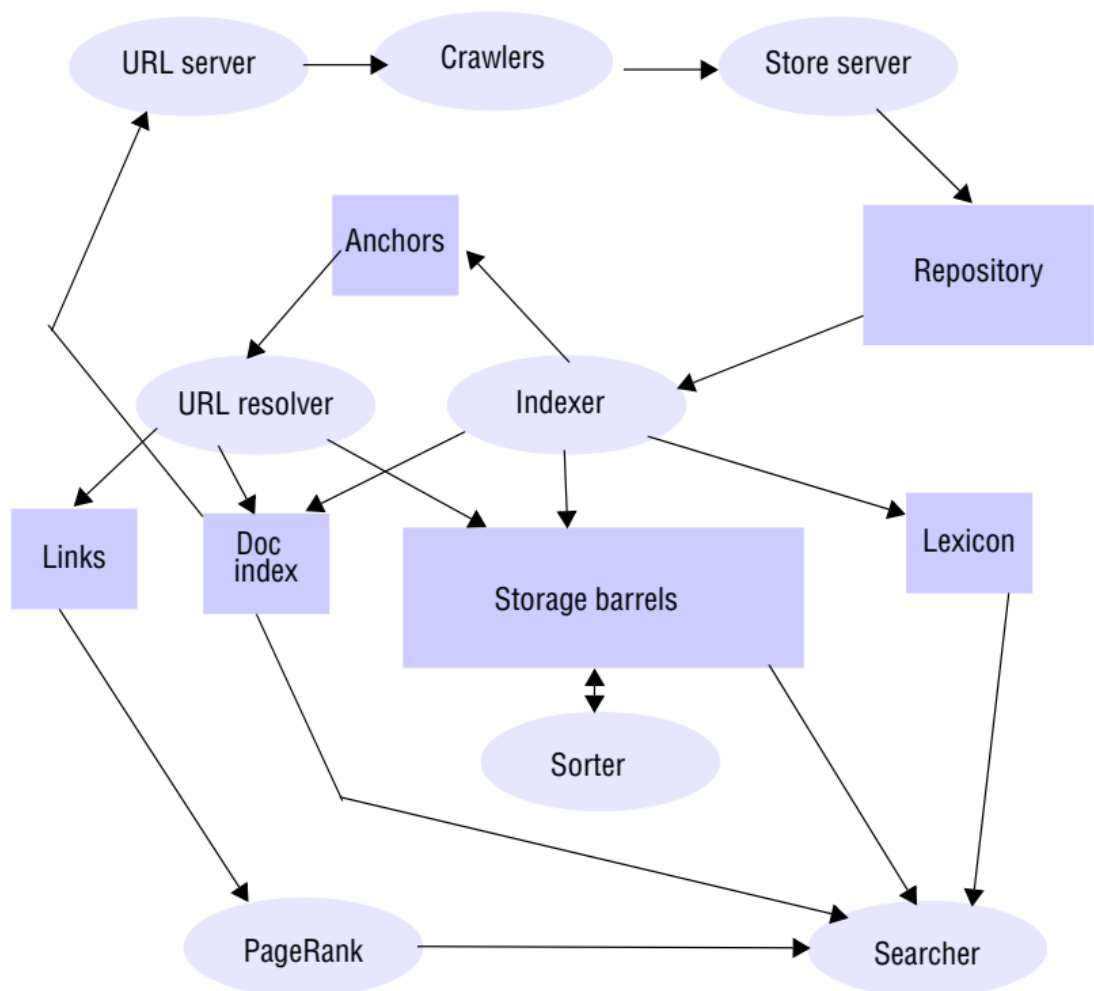
# GOOGLE

- MV, CA, USA
- googol
- Stanford research project, 1998
- "To organize the world's information, and make it accessible"
- Ranking Algorithm in its search engine (Internet Search Market)
- Diversified into Cloud Computing realms

## GOOGLE Search Engine

- Crawler
- Indexer
- Ranking

**Figure 21.1** Outline architecture of the original Google search engine [Brin and Page 1998]





# GOOGLE as Cloud Provider

## Software as a Service

- Web Applications
- Application level software over the internet
- Google Apps: set of web-based applications
- Gmail, Google Docs, Google Sites, Google Talk and Google Calendar

## Platform as a Service

- Distributed systems APIs as service over the internet
- Supports development and hosting of web application
- Google App Engine

## Overall Architecture and Design Philosophy

### PHYSICAL Architecture

**Figure 21.3** Organization of the Google physical infrastructure

