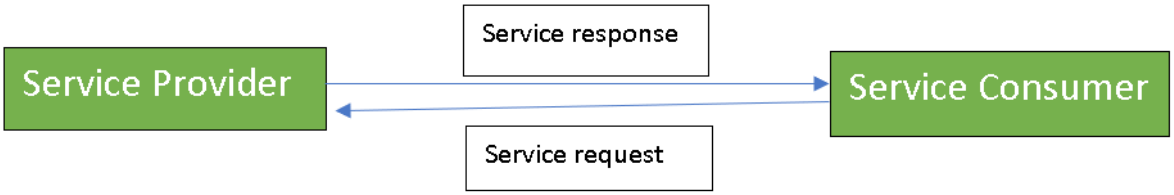
Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural approach in which applications make use of services available in the network. In this architecture, services are provided to form applications, through a communication call over the internet.

- SOA allows users to combine a large number of facilities from existing services to form applications.
- SOA encompasses a set of design principles that structure system development and provide means for integrating components into a coherent and decentralized system.
- SOA based computing packages functionalities into a set of interoperable services, which can be integrated into different software systems belonging to separate business domains.

There are two major roles within Service-oriented Architecture:

1. **Service provider:** The service provider is the maintainer of the service and the organization that makes available one or more services for others to use. To advertise services, the provider can publish them in a registry, together with a service contract that specifies the nature of the service, how to use it, the requirements for the service, and the fees charged.
2. **Service consumer:** The service consumer can locate the service metadata in the registry and develop the required client components to bind and use the service.



Services might aggregate information and data retrieved from other services or create workflows of services to satisfy the request of a given service consumer. This practice is known as service orchestration Another important interaction pattern is service choreography, which is the coordinated interaction of services without a single point of control.

Guiding Principles of SOA:

1. **Standardized service contract:** Specified through one or more service description documents.
2. **Loose coupling:** Services are designed as self-contained components, maintain relationships that minimize dependencies on other services.
3. **Abstraction:** A service is completely defined by service contracts and description documents. They hide their logic, which is encapsulated within their implementation.

4. **Reusability:** Designed as components, services can be reused more effectively, thus reducing development time and the associated costs.
5. **Autonomy:** Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation.
6. **Discoverability:** Services are defined by description documents that constitute supplemental metadata through which they can be effectively discovered. Service discovery provides an effective means for utilizing third-party resources.
7. **Composability:** Using services as building blocks, sophisticated and complex operations can be implemented. Service orchestration and choreography provide a solid support for composing services and achieving business goals.

Advantages of SOA:

- **Service reusability:** In SOA, applications are made from existing services.Thus, services can be reused to make many applications.
- **Easy maintenance:** As services are independent of each other they can be updated and modified easily without affecting other services.
- **Platform independant:** SOA allows making a complex application by combining services picked from different sources, independent of the platform.
- **Availability:** SOA facilities are easily available to anyone on request.
- **Reliability:** SOA applications are more reliable because it is easy to debug small services rather than huge codes
- **Scalability:** Services can run on different servers within an environment, this increases scalability

Disadvantages of SOA:

- **High overhead:** A validation of input parameters of services is done whenever services interact this decreases performance as it increases load and response time.
- **High investment:** A huge initial investment is required for SOA.
- **Complex service management:** When services interact they exchange messages to tasks. the number of messages may go in millions. It becomes a cumbersome task to handle a large number of messages.

Practical applications of SOA: SOA is used in many ways around us whether it is mentioned or not.

1. SOA infrastructure is used by many armies and air force to deploy situational awareness systems.
2. SOA is used to improve the healthcare delivery.
3. Nowadays many apps are games and they use inbuilt functions to run. For example, an app might need GPS so it uses inbuilt GPS functions of the device. This is SOA in mobile solutions.
4. SOA helps maintain museums a virtualized storage pool for their information and content

REST API (Introduction)

REpresentational **S**tate **T**ransfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. **REST API** is a way of accessing the web services in a simple and flexible way without having any processing.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses the less bandwidth, simple and flexible making it more suitable for internet usage. It's used to fetch or give some information from a web services. All communication done via REST API used only HTTP request.

Working

A request is send from client to server in the form of web URL as HTTP GET or POST or PUT or DELETE. After that a response is come back from server in the form of resource which can be anything like HTML, XML, Image or JSON. But now JSON is the most popular format being used in Web Services.

In **HTTP** there are five methods which are commonly used in a REST based Architecture i.e., POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations respectively. There are other methods which are less frequently used like OPTIONS and HEAD

Using HTTP Methods for RESTful Services

The HTTP verbs comprise a major portion of our "uniform interface" constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Below is a table summarizing recommended return values of the primary HTTP methods in combination with the resource URIs:

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link	404 (Not Found), 409 (Conflict)

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
		to /customers/{id} containing new ID.	if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/ Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/ Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Unit -2

Service Oriented Architecture

Rest & Systems of System

Web Services

Publish & Subscribe model

-Basics of Virtualization

Types of virtualization

Virtualization Structures

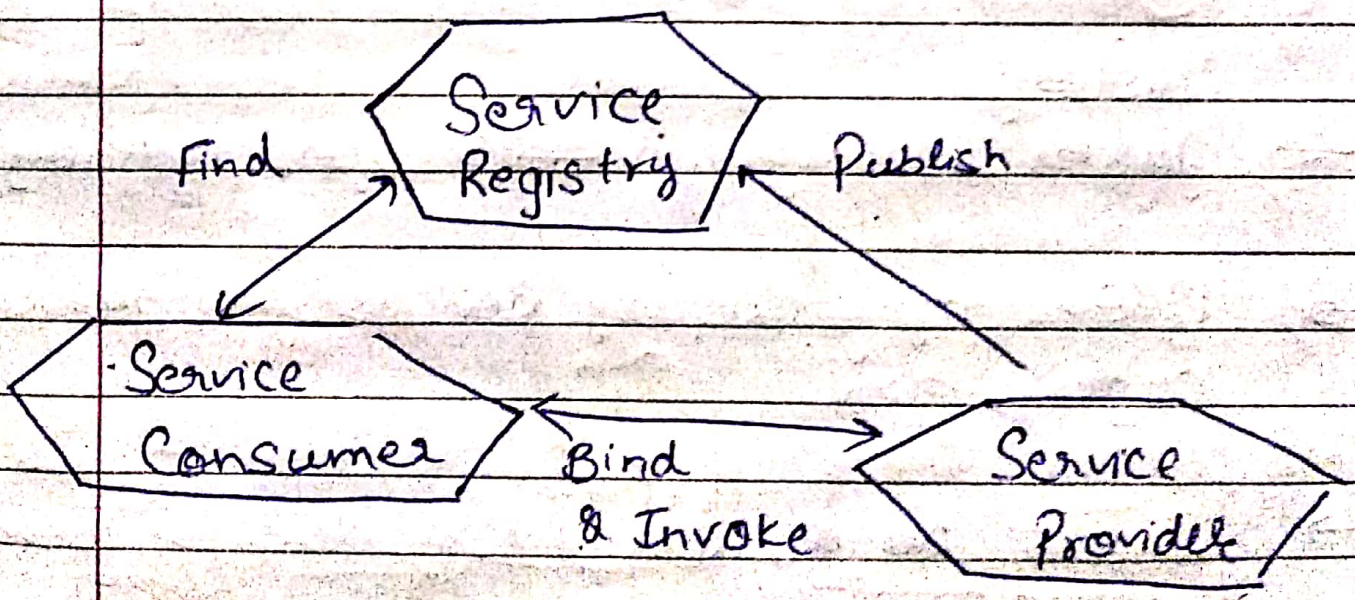
Tools & Mechanisms

Virtualization of CPU

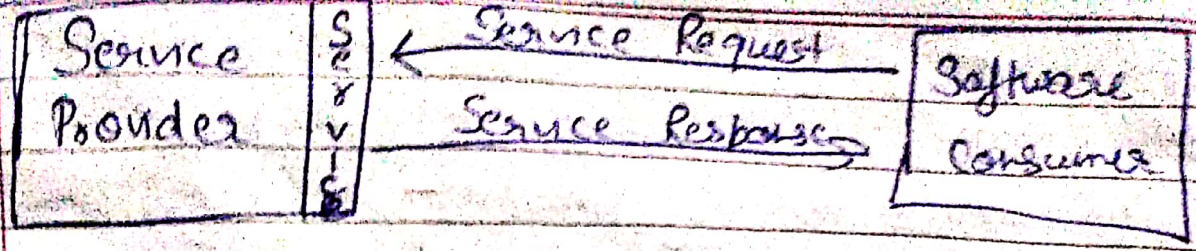
Memory I/O Devices

Visualization Support & Disaster Recovery

→ Service Oriented Architecture



Its a Style of Software design where Services of Software are provided to other components via Communication protocal over a network



SOA is a solution for making two Software communicate to each other

Service is a well defined function that doesn't depend on the State of other Services

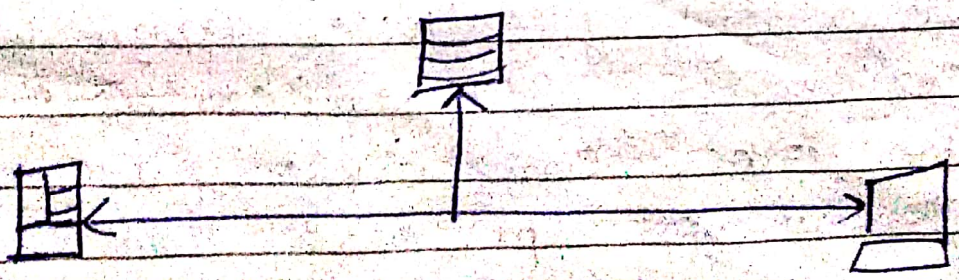
Consumer needs to know how to Call the Service & what to expect in response

History

→ 1980s mainframes → one for each organisation

→ 1990s many computing devices for each organisation

→ 2000s No of computing devices grow exponentially



System can't communicate with other even if they do it was very difficult to make interfaces for each other

Standardized Messages

XML was introduced

↳ Describes Data

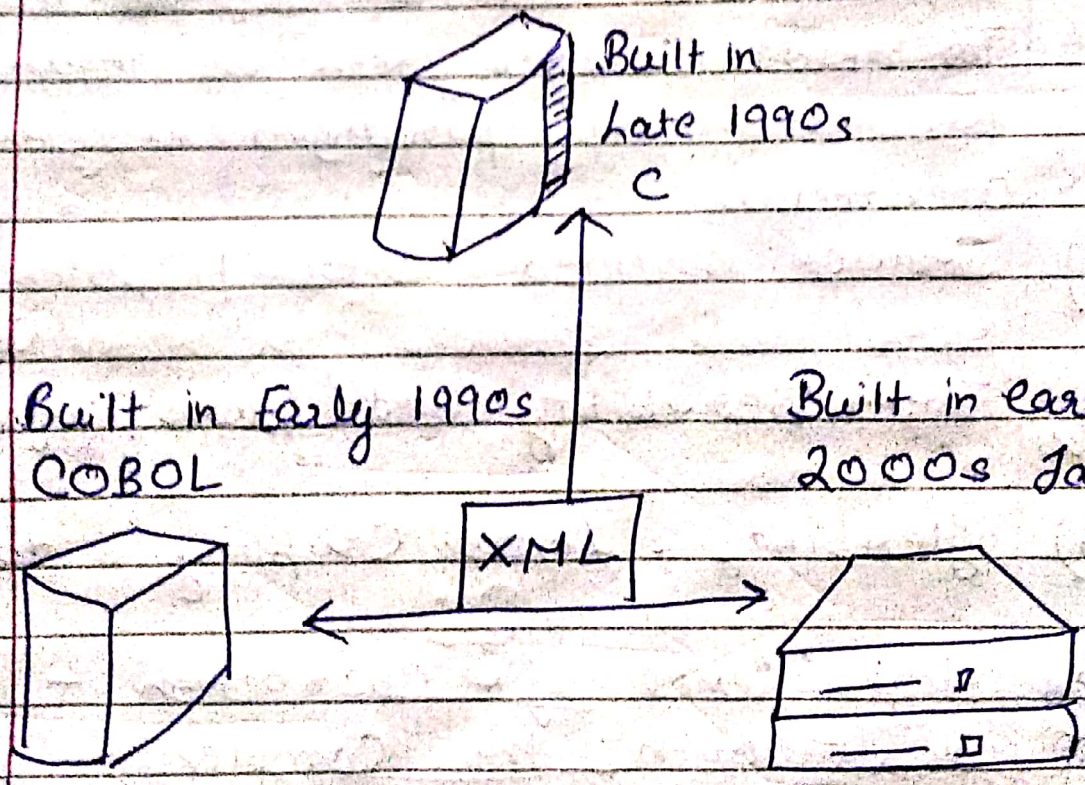
↳ Shared electronically

↳ Share info in Consistent Manner

Benifits → ~~Interplotency~~

Interoperability

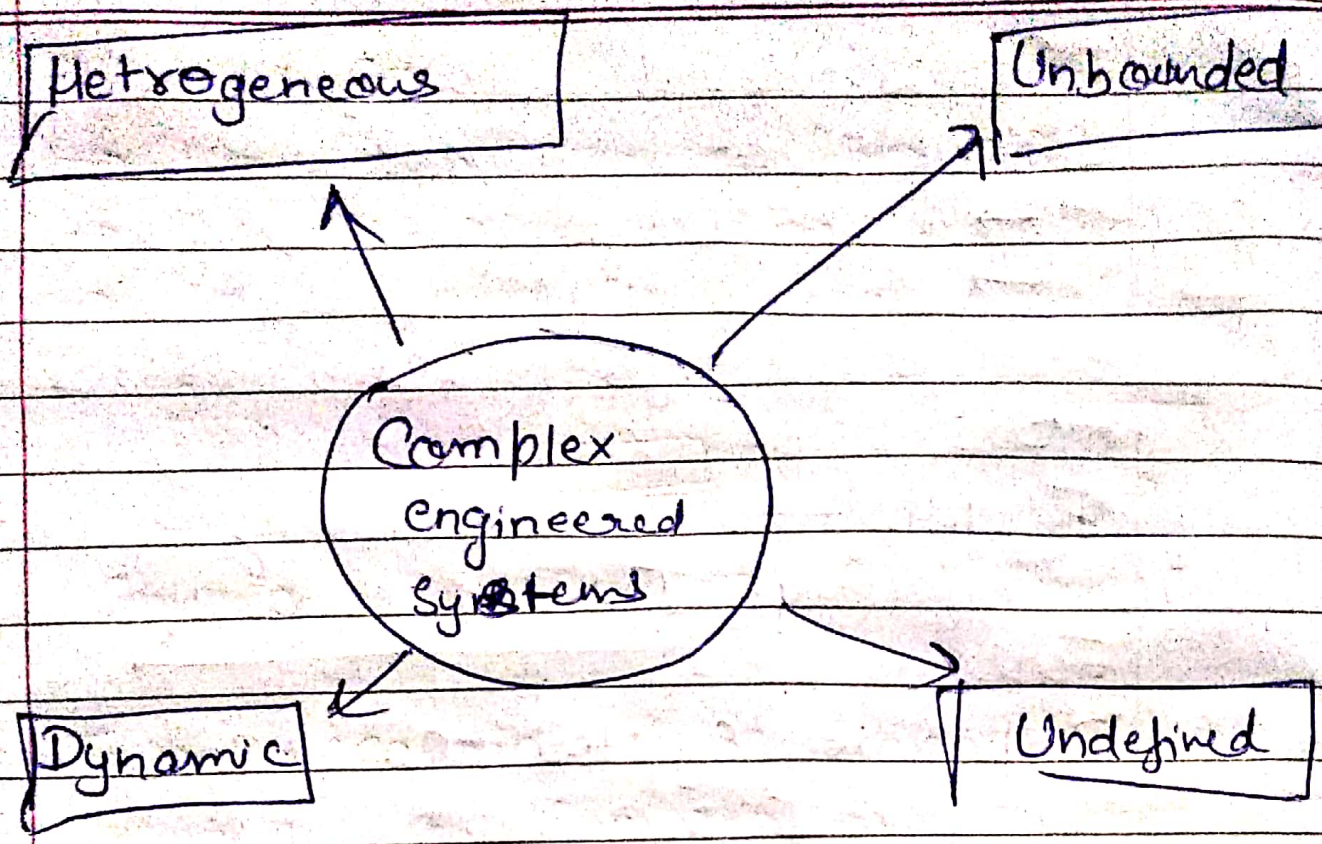
Disparate System can Communicate



SOA support Horizontal Scalability

SOA support Reusability.

↳ So whenever a new system is made there is no need to develop from Scratch. We can quickly develop new services from ~~new~~ existing services



To model these system we require an alternative paradigm in system architecture -

1) Service oriented over properties of Components

2) Interoperability and Cross platform

3) flexibility, Loose Coupling & distributed

4) High level of abstraction :- To deal with the complexity of the systems

Over past few decade a new paradigm has emerged in IT to deal with complex engineered system

SOA - It is a response adapted to distributed & hetrogeneous environment.

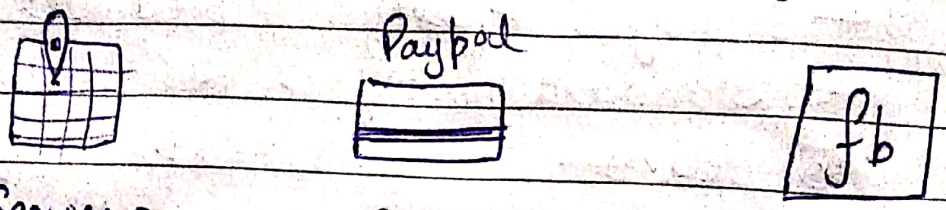
— approach for distributed architecture
— Employes loosely coupled services,
— Standard interface & protocals to deliver seemless cross platform integration
— Service is provided by set of interface and protocals so that customer can communicate with the services with the service bus

Ex: Problem: To make a Software that helps to users to pay their parking ticket online

Approach 1: I can develop a subsystem that functions as a street map than another subsystem dealing with the payment, yet another for login and user authentication
It can take many years

CLASSTIME | Page No.
Date / /

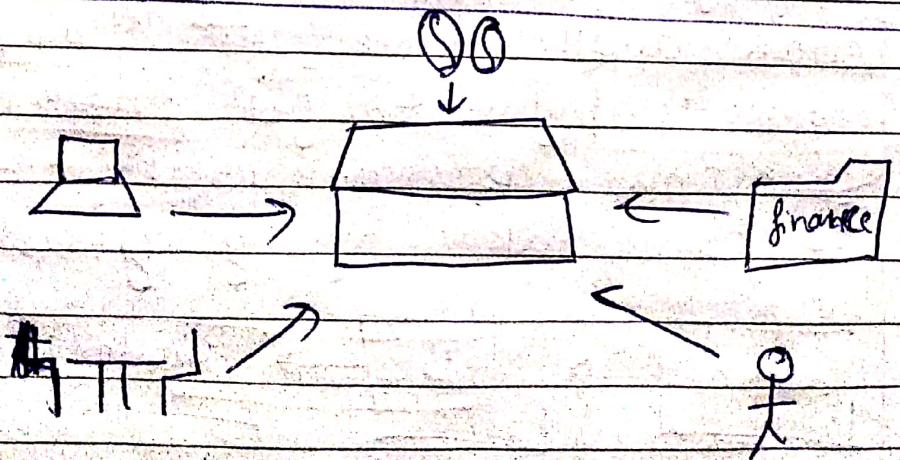
Approach 2:
→ Use login service from FB
→ Use Map service from Google Maps
→ Use payment service from paypal



Then integrate these services by creating a common process that guides to use these service to deliver a desired functionality.

Instead of starting from Scratch I used their services to make my app. This is called Service oriented architecture

Ex 2



Services are independent, abstract we are not interested in internal functioning of the components we just need to know how to interact ie interface

LINK FOR MY PREZI PPT (SOA)
https://prezi.com/view/xN9EULyaJhu7Ppr1ZoqT/

Unit 2

Web services

Contents

What are Web Services?

There is more than one way to answer, “What is a web service?” But, essentially, web services include any software, application, or cloud technology that provides standardized web protocols (HTTP or HTTPS) to interoperate, communicate, and exchange data messaging – usually XML (Extensible Markup Language) – throughout the internet.

In other words, web services are XML-centered data exchange systems that use the internet for A2A (application-to-application) communication and interfacing. These processes involve programs, messages, documents, and/or objects.

A key feature of web services is that applications can be written in various languages and are still able to communicate by exchanging data with one another via a web service between clients and

servers. A client summons a web service by sending a request via XML, and the service then responses with an XML response. Web services are also often associated with SOA (Service-Oriented Architecture).

To break that down, a web service comprises these essential functions:

- Available over the internet or intranet networks
- Standardized XML messaging system
- Independent of a single operating system or programming language
- Self-describing via standard XML language
- Discoverable through a simple location method

A web service supports communication among numerous apps with HTML, XML, WSDL, SOAP, and other open standards. XML tags the data, SOAP transfers the message, and WSDL describes the service's accessibility.

Here's an instance of how it works: A web service sits between two sets of java, .net, or PHP apps providing a way for these applications to communicate over a network. On one side, for example, a java app interacts with the java, .net, and PHP apps on the other end by way of the web service communicating an independent language.

Web services offer different benefits across business operations. The technology helps IT pros and web architects streamline connectivity by minimizing development time. And with this simplified infrastructure, company executives begin to see higher ROI (return on investment). In a B2B operation where both parties understand how the process works, web services provide efficient technology distribution throughout an entire network.

History of web services

- Structured Programming
- Object-Oriented Programming
- Distributed Programming

- Electronic data exchange
- World wide web
- Web Services

First of all structure programming came in to existence then object oriented and so on. When the world wide web came then requirement of web services need. Because due to platform independent any application communicate with each other. Firstly HP developed it after Microsoft came in to the web services in popular mode.

Advantages of web services

- **Open, test-based standards**
- **Modular approach**
- **Inexpensive to implement**
- **Reduce the cost of enterprise application integration**
- **Incremental implementation**

What are the Different Types of Web Services?

There are a few central types of web services: XML-RPC, UDDI, SOAP, and REST:

XML-RPC (Remote Procedure Call) is the most basic XML protocol to exchange data between a wide variety of devices on a network. It uses HTTP to quickly and easily transfer data and communication other information from client to server.

UDDI (Universal Description, Discovery, and Integration) is an XML-based standard for detailing, publishing, and discovering web services. It's basically an internet registry for businesses around the world. The goal is to streamline digital transactions and e-commerce among company systems.

SOAP, which will be described in detail later in the blog, is an XML-based Web service protocol to exchange data and documents over HTTP or SMTP (Simple Mail Transfer Protocol). It allows independent processes operating on disparate systems to communicate using XML.

REST, which will also be described in great detail later in the blog, provides communication and connectivity between devices and the internet for API-based tasks. Most RESTful services use HTTP as the supporting protocol.

Here are some well-known web services that use markup languages:

- Web template
- JSON-RPC
- JSON-WSP
- Web Services Description Language (WSDL)
- Web Services Conversation Language (WSCL)
- Web Services Flow Language (WSFL)
- Web Services Metadata Exchange (WS-MetadataExchange)
- XML Interface for Network Services (XINS)

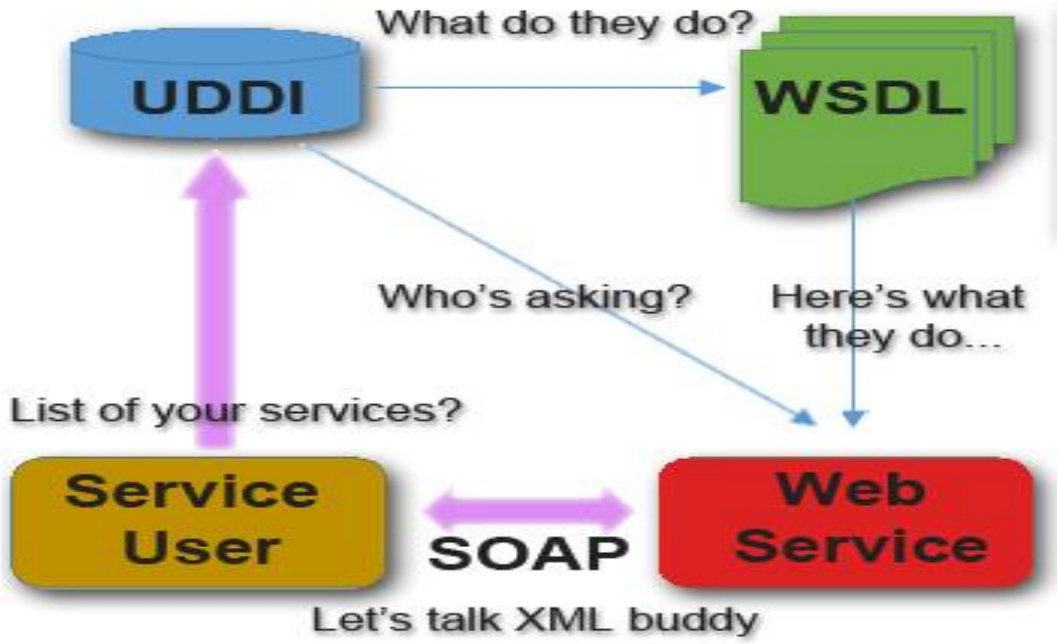


Figure1

Figure 1 shows the working of web services in which first service user(client) send the request to access the server or cloud application , so it goes to UDDI then UDDI send the request to exactly WSDL then a path will become after that web services are communicated with client and server using SOAP protocols.

SOAP vs. REST Web Services

For years, IT pros and web developers have debated over which web service is better and why. Well, there isn't a clear-cut winner – it all just depends. RESTful web services and SOAP offer different variations. For example, a REST web service is generally a better choice when time is a factor, but SOAP wins out when building a service with multiple, non-CRUD methods. A company's specific requirements determine which type of web service a partner will implement unless already decided by the WS provider.

Then there are times when both options are the right answer. That's the case for two of the world's biggest e-commerce companies: Amazon and eBay use web services for both REST and

SOAP. And as more organizations become service-focused and explore more functionality, they will have no choice but to support both types.

But what is the difference between REST web services and SOAP web services? Let's break down each option while exploring some pros and cons.

RESTful Web Services

What is a RESTful web service? The acronym REST, or sometimes ReST, stands for Representational State Transfer and is an architectural style, meaning each unique URL represents an individual object of some sort. A REST web service uses HTTP and supports/repurposes several HTTP methods: GET, POST, PUT or DELETE. It also offers simple CRUD-oriented services. Fun fact: The original RESTful architecture was designed by one of the leading authors of HTTP, Roy Fielding.

Pros: Lightweight, human readable, easier to build

Cons: Point-to-point communication, lack of standards

SOAP Web Services

SOAP is defined as Simple Object Access Protocol. This web service protocol exchanges structured data using XML and generally HTTP and SMTP for transmission. SOAP also uses WSDL (Web Services Description Language) documents to distribute a web service description model. This describes how the SOAP requests (client-side) and responses (server-side) must appear. Additionally, SOAP web Services have standards for security and addressing.

Pros: Usually easier to consume, more standards (WSDL, etc.), distributed computing

Cons: Difficult set-up, more convoluted coding, harder to develop

API vs. Web Services

Web services and APIs are often mistaken for each other, which isn't all that surprising since there is some distinct common ground.

Most web services provide an API, which, with its set of commands and functions, is used to retrieve data. Here's one example: Twitter delivers an API that authorizes a developer access tweets from a server and then collects data in JSON format.

But here's something to keep in mind: All web services can be APIs, but not all APIs can be web services. Now, if that syllogism makes your head spin, maybe these distinctions will clear up the API vs. web services confusion:

Differences between APIs and Web Services

1. APIs can be hosted within an app or IIS (Internet Information Services), but a web service can only be hosted on IIS.
2. Web services are not an open source and are used to understand JSON (JavaScript Object Notation) or XML, whereas APIs are an open source and only used for XML.
3. API is a light-weight architecture (best for limited bandwidth devices (e.g. smartphone). Web services are not lightweight architectures since they require SOAP to send and receive network data.
4. APIs can use any form of communication, but a Web service only uses SOAP, REST, and XML-RPC.
5. APIs support URL, request/response headers, caching, versioning, content formats. Web services only support HTTP.

Similarities between APIs and Web Services

1. Both are accessed through HTTP/HTTPS to enable communication between services providers and customers.
2. Both call a function, process data, and receive a response

Modernized web services have changed the digital landscape with evolved system integration and interoperability. Before advancements to web services, limited and burdensome

integration prevented streamlined data exchange among various technologies, formats, vendors, and B2B operations. Now, web services offer a level of modern functionality and less complexity.

Publish Subscribe Model

Content:

- **What is Request response Model**
- **Working of RR Model**
- **What is Public Subscribe Model?**
- **Working of Pub/Sub Model**
- **Advantages of Pub/Sub Model**
- **Disadvantages of Pub/Sub Model**

Request Response Model

Request–response, or request–reply, is one of the basic methods computers use to communicate with each other, in which the first computer sends a request for some data and the second responds to the request. Usually, there is a series of such interchanges until the complete message is sent; browsing a web page is an example of request–response communication. Request–response can be seen as a telephone call, in which someone is called and they answer the call.

Request–response is a message exchange pattern in which a requestor sends a request message to a replier system which receives and processes the request, ultimately returning a message in response. This is a simple, but powerful messaging pattern which allows two applications to have a two-way conversation with one another over a channel. This pattern is especially common in client–server architectures.

Pub/Sub Model

DefinitionPub/sub is shorthand for publish/subscribe messaging, an asynchronous communication method in which messages are exchanged between applications without knowing the identity of the sender or recipient.

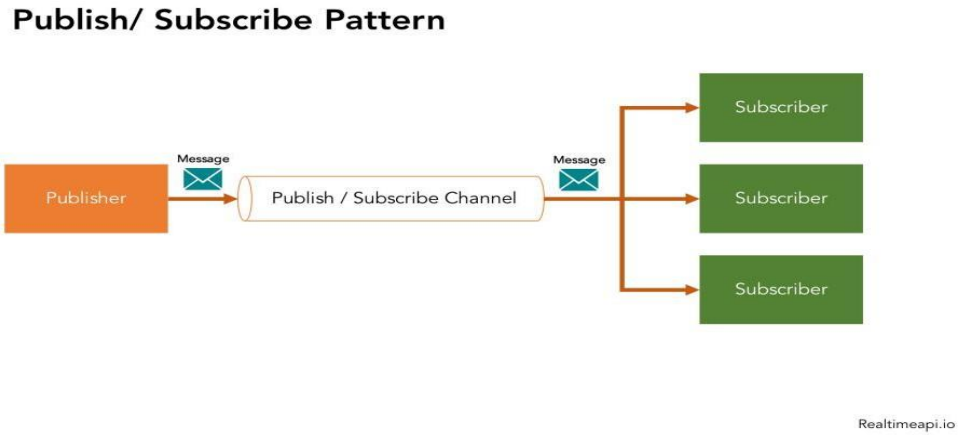
Terminology used in Pub/Sub

Four core concepts make up the pub/sub model:

1. **Topic** – An intermediary channel that maintains a list of subscribers to relay messages to that are received from publishers
2. **Message** – Serialized messages sent to a topic by a publisher which has no knowledge of the subscribers
3. **Publisher** – The application that publishes a message to a topic
4. **Subscriber** – An application that registers itself with the desired topic in order to receive the appropriate messages

Working of Pub/Sub model

- **A Publish Subscribe Architecture is a messaging pattern where the publishers broadcast messages, with no knowledge of the subscribers.**
- **Similarly the subscribers 'listen' out for messages regarding topic/categories that they are interested in without any knowledge of who the publishers are.**
- **The event bus transfers the messages from the publishers to the subscribers.**
- **Each subscriber only receives a subset of the messages that have been sent by the publisher they only receive the message topics or categories they have subscribed to.**



In the above figure publishers send message to topic and then topic relay the message to subscriber and subscriber recognize

the desire message.

Advantages and disadvantages of pub/sub

As with all technology, using pub/sub messaging comes with advantages and disadvantages. The two primary advantages are loose coupling and scalability.

Loose coupling

Publishers are never aware of the existence of subscribers so that both systems can operate independently of each other. This methodology removes service dependencies that are present in traditional coupling. For example, a client generally cannot send a message to a server if the server process is not running. With pub/sub, the client is no longer concerned whether or not processes are running on the server.

Scalability

Pub/sub messaging can scale to volumes beyond the capability of a single traditional data center. This level of scalability is primarily due to parallel operations, message caching, tree-based routing, and multiple other features built into the pub/sub model.

Scalability does have a limit though. Increasing the number of nodes and messages also increases the chances of experiencing a load surge or slowdown. On top of that, the advantages of the pub/sub model can sometimes be overshadowed by the message delivery issues it experiences, such as:

- A publisher may only deliver messages for a certain period of time regardless of whether the message was received or not.

- Since the publisher does not have a window into the subscriber it will always assume that the appropriate subscriber is listening. If the subscriber isn't listening and misses an important message it can be disastrous for production systems.

How Pub/Sub Works

In the overview we covered how a publisher sends a message to a topic and how the topic forwards the message to the appropriate subscriber. From a topology point of view it is a simple process.

When it comes to coding the publish or the subscribe process the model can be a bit more confusing. Consider the following Java code which is used to create a topic.

```
Topic createTopic(String topicName) throws IOException { String topic = getTopic(topicName); // adds project name and resource type Pubsub.Projects.Topics topics = pubsub.projects().topics(); ListTopicsResponse list = topics.list(project).execute(); if (list.getTopics() == null || !list.getTopics().contains(new Topic().setName(topic))) { return topics.create(topic, new Topic()).execute(); } else { return new Topic().setName(topic); } }
```

Cloud or edge providers often simplify this code. Google Cloud, for example, has simplified topic creation into a single line of code.

```
gcloud beta pubsub topics create topicName
```

Examples of Pub/Sub

Publish/subscribe messaging has a multitude of use cases, some of which include:

- Balancing workloads
- Asynchronous workflows
- Event notifications
- Data streaming

Faye

Faye is an open source system based on pub/sub messaging. The code below shows you how to start a server, create a client, and send messages using Faye.

```
var http = require('http'), faye = require('faye'); var server = http.createServer(), bayeux = new faye.NodeAdapter({mount: '/'}); bayeux.attach(server); server.listen(8000); var client = new Faye.Client('http://localhost:8000/'); client.subscribe('/messages', function(message) { alert('Got a message: ' + message.text); }); client.publish('/messages', { text: 'Hello world' })
```

Faye is much more straightforward than standard JavaScript and it was created specifically for Node.js and Ruby servers. It's often used for online instant messaging which is a use case for pub/sub that most people experience on a daily basis.

Pub/sub on the edge

As with a lot of edge benefits pub/sub thrives with the speed that comes with being on the edge. A publisher must send a message to a topic, sometimes located far away in the physical world, that a subscriber is listening to. To travel across a room, a message may need to travel halfway around the world through Internet exchange points and physical distance always creates latency. On the edge, and particularly with StackPath's 45 edge locations, that message can travel two to four times faster across the room or even across the world.

Key Takeaways

- Publish/subscribe messaging is when a publisher sends a message to a topic and the message is forwarded to a subscriber.
- The concept of pub/sub is easy to understand but every coding and programming language handles it differently, making it a little more challenging to learn across all platforms.
- On the edge, message delivery times can be two to four times faster by using a network backbone and multiple points of presence.

Topic: Virtualization and its types

What is VIRTUALIZATION?

Virtualization is a technique of how to separate a service from the underlying physical delivery of that service. It is the process of creating a virtual version of something like computer hardware. It was initially developed during the mainframe era. It involves using specialized software to create a virtual or software-created version of a computing resource rather than the actual version of the same resource. With the help of Virtualization, multiple operating systems and applications can run on same machine and its same hardware at the same time, increasing the utilization and flexibility of hardware.

In other words, one of the main cost effective, hardware reducing, and energy saving techniques used by cloud providers is virtualization. Virtualization allows sharing a single physical instance of a resource or an application among multiple customers and organizations at one time. It does this by assigning a logical name to a physical storage and providing a pointer to that physical resource on demand. The term virtualization is often synonymous with hardware virtualization, which plays a fundamental role in efficiently delivering Infrastructure-as-a-Service (IaaS) solutions for cloud computing. Moreover, virtualization technologies provide a virtual environment for not only executing applications but also for storage, memory, and networking.

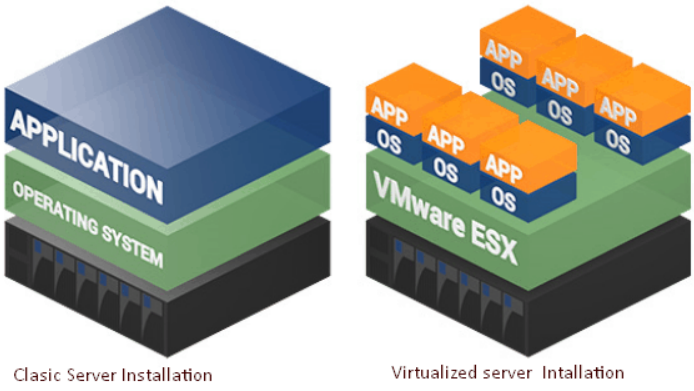


Fig 1: Concept of virtualization

Benefits of Virtualization:

1.More flexible and efficient allocation of resources.
2. Enhance development productivity.
3. It lowers the cost of IT infrastructure.
4. Remote access and rapid scalability.
5. High availability and disaster recovery.
6. Pay per use of the IT infrastructure on demand.
7. Enables running multiple operating system.

Types of Virtualization:

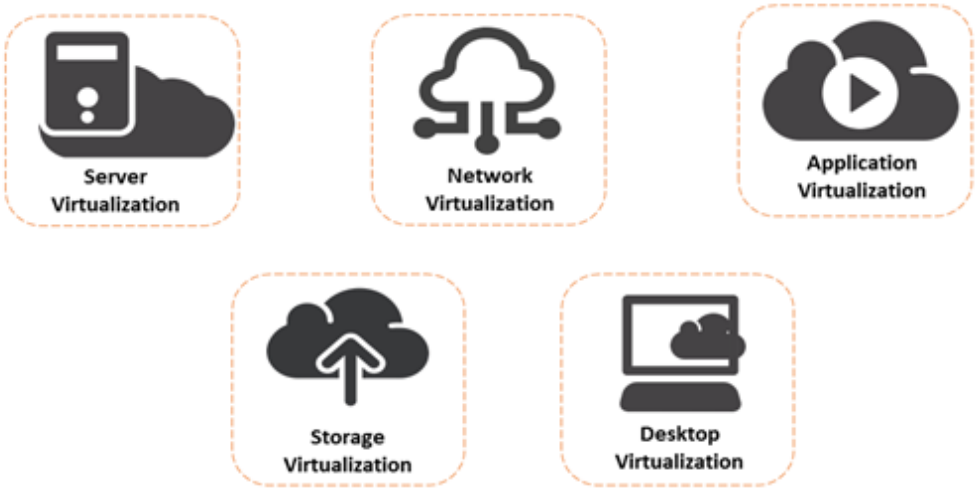


Fig 2: Types of Virtualization

1. **Application Virtualization:**
   Application virtualization helps a user to have a remote access of an application from a server. The server stores all personal information and other characteristics of the application but can still run on a local workstation through internet. Example of this would be a user who needs to run two different versions of the same software. Technologies that use application virtualization are hosted applications and packaged applications.

2. **Network Virtualization:**
   The ability to run multiple virtual networks with each has a separate control and data plan. It co-exists together on top of one physical network. It can be managed by individual parties that potentially confidential to each other.
   Network virtualization provides a facility to create and provision virtual networks—logical switches routers, firewalls, load balancer, Virtual Private Network (VPN), and workload security within days or even in weeks.

3. **Desktop Virtualization:**
   Desktop virtualization allows the users' OS to be remotely stored on a server in the data center. It allows the user to access their desktop virtually, from any location by different machine. Users who want specific operating systems other than Windows Server will need to have a virtual desktop. Main benefits of desktop virtualization are user mobility, portability, easy management of software installation, updates and patches.

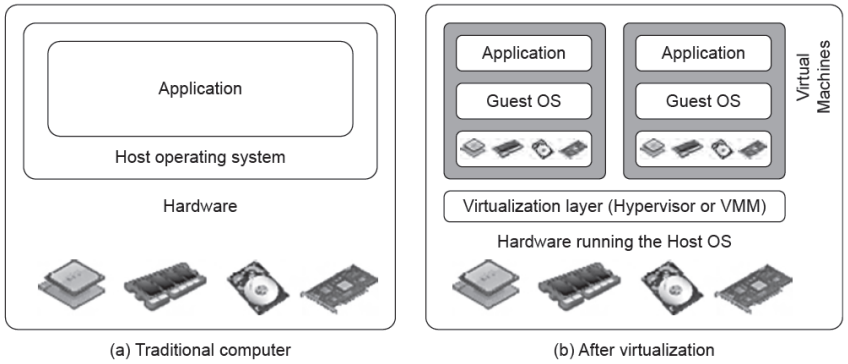
4. **Storage Virtualization:**
   Storage virtualization is an array of servers that are managed by a virtual storage system. The servers aren't aware of exactly where their data is stored, and instead function more like worker bees in a hive. It makes managing storage from multiple sources to be managed and utilized as a

single repository. Storage virtualization software maintains smooth operations, consistent performance and a continuous suite of advanced functions despite changes, break down and differences in the underlying equipment.

Implementation Levels of Virtualization

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility. The idea is to separate the hardware from the software to yield better system efficiency.

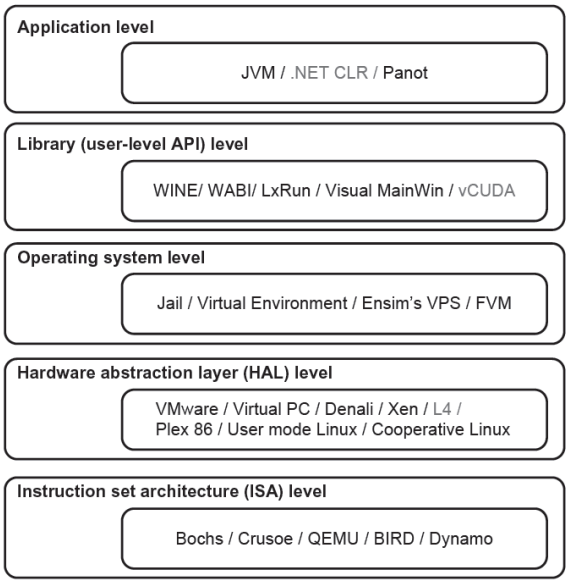
A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in Figure. After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer as shown in Figure.



(a) Traditional computer (b) After virtualization

Virtualization can be implemented at various operational levels, as given below:

- Instruction set architecture (ISA) level
- Hardware level
- Operating system level
- Library support level
- Application level



Instruction Set Architecture Level

At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. One source instruction may require tens or hundreds of native target instructions to perform its function. Obviously, this process is relatively slow. For better performance, dynamic binary translation is desired. This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level

It is performed right on top of the bare hardware and generates a virtual hardware environment for a VM. On the other hand, the process manages the underlying hardware through virtualization. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices so as hardware utilization rate by multiple users concurrently may be upgraded

Operating System Level

OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in data centers. The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users.

Library Support Level

Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration.

User-Application Level

On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL) VMs. In this scenario, the virtualization layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition. Any program written in the HLL and compiled for this VM will be able to run on it. The Microsoft .NET CLR and Java Virtual Machine (JVM) are two good examples of this class of VM. Other forms of application-level virtualization are known as application isolation, application sandboxing, or application streaming. The process involves wrapping the application in a layer that is isolated from the host OS and other applications. The result is an application that is much easier to distribute and remove from user workstations.

Virtualization Support at the OS Level

It is slow to initialize a hardware-level VM because each VM creates its own image from scratch and storage of such images are also slow. OS-level virtualization provides a feasible solution for these hardware-level virtualization issues. OS virtualization inserts a virtualization layer inside an operating system to partition a machine's physical resources. It enables multiple isolated VMs within a single operating system kernel. This kind of VM is often called a virtual execution environment (VE). This VE has its own set of processes, file system, user accounts, network interfaces with IP addresses, routing tables, firewall rules, and other personal settings.

Advantages:

- VMs at the operating system level have minimal startup/shutdown costs, low resource requirements, and high Scalability
- It is possible for a VM and its host environment to synchronize state changes when necessary

Virtualization Structures/Tools and Mechanisms

Before virtualization, the operating system manages the hardware. After virtualization, a virtualization layer is inserted between the hardware and the OS. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware. Depending on the position of the virtualization layer, there are several classes of VM architectures, namely

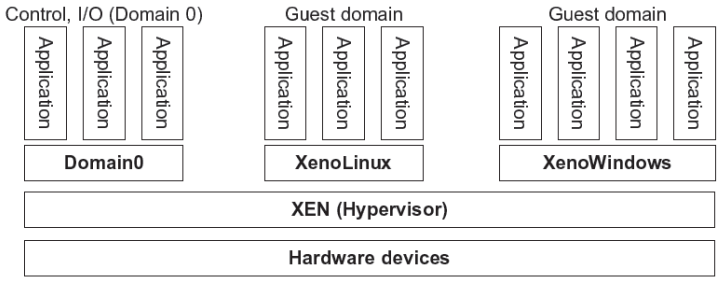
- Hypervisor architecture,
- Paravirtualization
- host-based virtualization.

Hypervisor and Xen Architecture

Depending on the functionality, a hypervisor can assume a micro-kernel architecture or a monolithic hypervisor architecture. A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor. A monolithic hypervisor implements all the aforementioned functions, including those of the device drivers. Therefore, the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor.

Xen Architecture

Xen is an open source hypervisor program developed by Cambridge University. Xen is a microkernel hypervisor, which separates the policy from the mechanism. It implementsall the mechanisms, leaving the policy to be handled by Domain 0, as shown in Figure. Xen does not include any device drivers natively. It just provides a mechanism by which a guest OS can have direct access to the physical devices.



Like other virtualization systems, many guest OSes can run on top of the hypervisor. The guest OS (privileged guest OS), which has control ability, is called Domain 0, and the others are called Domain U. 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.

Binary Translation with Full Virtualization

Depending on implementation technologies, hardware virtualization can be classified into two categories: full virtualization and host-based virtualization.

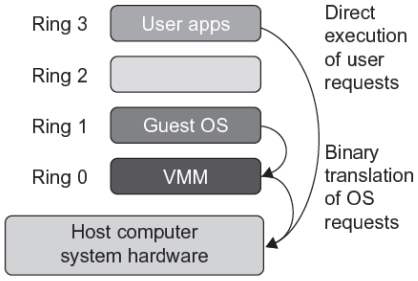
Full Virtualization

With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software.

Both the hypervisor and VMM approaches are considered full virtualization. Noncritical instructions do not control hardware or threaten the security of the system, but critical instructions do. Therefore, running noncritical instructions on hardware not only can promote efficiency, but also can ensure system security.

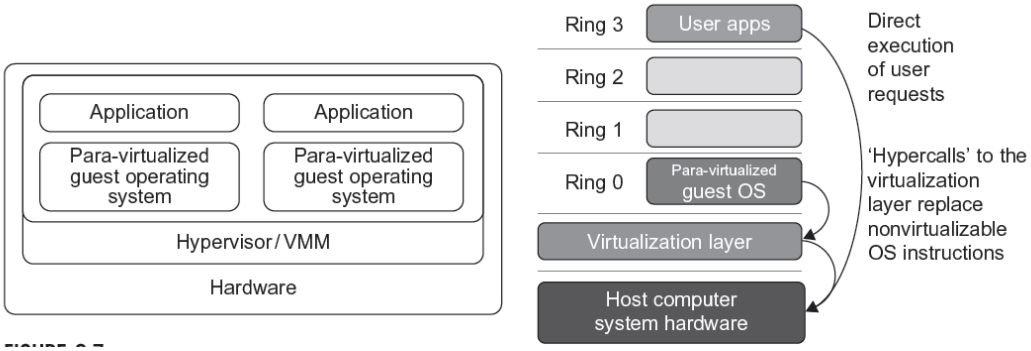
Host-Based Virtualization

An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware. The guest OSes are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications can also run with the host OS directly. This host based architecture has some distinct advantages, as enumerated next. First, the user can install this VM architecture without modifying the host OS. Second, the host-based approach appeals to many host machine configurations.



Para-Virtualization

It needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system. Figure illustrates the concept of a para-virtualized VM architecture. The guest OS are para-virtualized. They are assisted by an intelligent compiler to replace the non virtualizable OS instructions by hypercalls. The traditional x86 processor offers four instruction execution rings: Rings 0, 1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed. The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3.



Although para-virtualization reduces the overhead, it has incurred problems like compatibility and portability, because it must support the unmodified OS as well. Second, the cost is high, because they may require deep OS kernel modifications. Finally, the performance advantage of para-virtualization varies greatly due to workload variations.

Virtualization of CPU, Memory, And I/O Devices

To support virtualization, processors such as the x86 employ a special running mode and instructions, known as hardware-assisted virtualization. In this way, the VMM and guest OS run in different modes and all sensitive instructions of the guest OS and its applications are trapped in the VMM. To save processor states, mode switching is completed by hardware.

Hardware Support for Virtualization

Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions. In a virtualized environment, it is more difficult to make OSes and applications run correctly because there are more layers in the machine stack. Figure shows the hardware support by Intel.

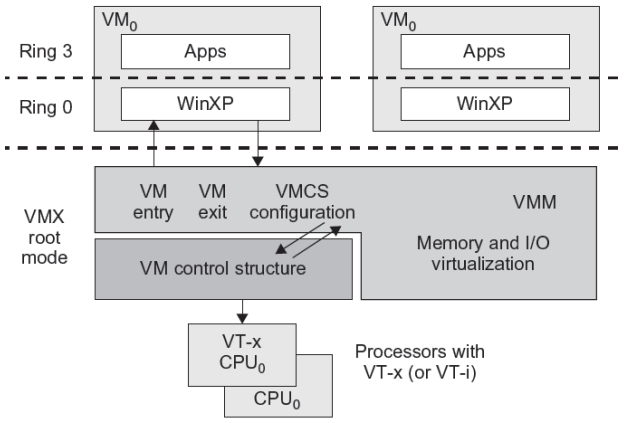
CPU Virtualization

Unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories: privileged instructions, controls sensitive instructions, and behavior-sensitive instructions. Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode. Control-sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control- and behavior-sensitive instructions of a VM are executed, they are trapped in the VMM. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions. On the contrary, x86 CPU architectures are not primarily designed to support virtualization.

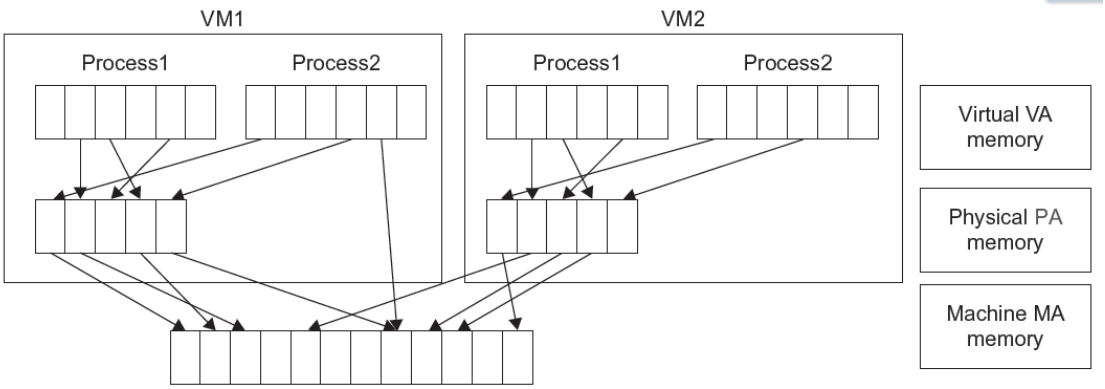
Hardware-Assisted CPU Virtualization

This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification



Memory Virtualization

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional environment, the OS maintains page table for mappings of virtual memory to machine memory, which is a one-stage mapping. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance. However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs. A two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory in guest OS.



Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table. VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup.

I/O virtualization

It involves managing the routing of I/O requests between virtual devices and the shared physical hardware. There are three ways to implement I/O virtualization:

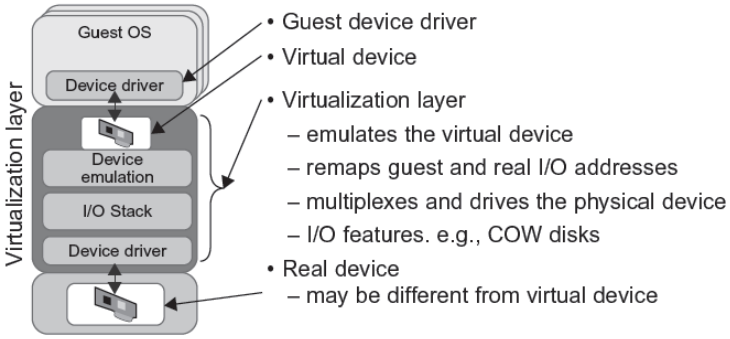
Full device emulation

Para-virtualization

Direct I/O.

Full device emulation

All the functions of a device like device enumeration, identification, interrupts, and DMA, are replicated in software and it is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices.



Para-virtualization

It is a split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

Direct I/O virtualization

It lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs. However, current direct I/O virtualization implementations focus on networking for mainframes.

Another way to help I/O virtualization is via self-virtualized I/O (SV-IO). The key idea is to harness the rich resources of a multicore processor. All tasks associated with virtualizing an I/O device are encapsulated in SV-IO. SV-IO defines one virtual interface (VIF) for every kind of virtualized I/O device, such as virtual network interfaces, virtual block devices (disk), virtual camera devices, and others. The guest OS interacts with the VIFs via VIF device drivers. Each VIF consists of two message queues. One is for outgoing messages to the devices and the other is for incoming messages from the devices. In addition, each VIF has a unique ID for identifying it in SV-IO.

Virtualization in Multi-Core Processors

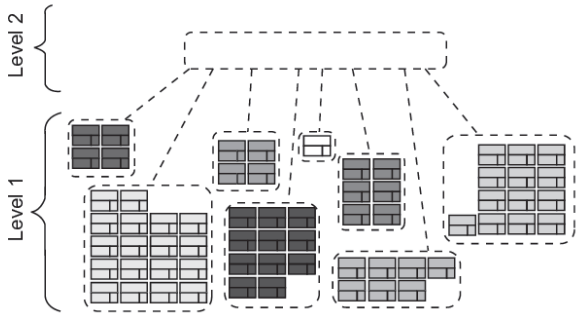
Multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualiuzation has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem. Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier.

The second challenge has spawned research involving scheduling algorithms and resource management policies

Virtual Hierarchy

A virtual hierarchy is a cache hierarchy that can adapt to fit the workload or mix of workloads. The hierarchy's first level locates data blocks close to the cores needing them for faster access, establishes a shared-cache domain, and establishes a point of coherence for faster communication. The first level can also provide isolation between independent workloads. A miss at the L1 cache can invoke the L2 access.

The following figure illustrates a logical view of such a virtual cluster hierarchy in two



(b) Multiple virtual clusters assigned to various workloads

levels. Each VM operates in a isolated fashion at the first level which minimize both miss access time and performance interference with other workloads or VMs. The second level maintains a globally shared memory facilitates dynamically repartitioning resources without costly cache flushes.

Virtualization Support and Disaster Recovery

4 Ways Virtualization helps with Disaster Recovery

- **Recover to any hardware**

  By using a virtualized environment you don't have to worry about having completely redundant hardware. Instead you can use almost any x86 platform as a backup solution, this allows you to save money by repurposing existing hardware and also gives your company more agility when it comes to hardware failure as almost any virtual server can be restarted on different hardware.

- **Backup and restore full images**

  By having your system completely virtualized each of your server's files are encapsulated in a single image file. An image is basically a single file that contains all of server's files, including system files, programs, and data; all in one location. By having these images it makes managing your systems easy and backups become as simple as duplicating the image file and restores are simplified to simply mounting the image on a new server.

- **Run other workloads on standby hardware**

  A key benefit to virtualization is reducing the hardware needed by utilizing your existing hardware more efficiently. This frees up systems that can now be used to run other tasks or be used as a hardware redundancy. This mixed with features like VMware's High Availability, which restarts a virtual machine on a different server when the original hardware fails, or for a more robust disaster recovery plan you can use Fault Tolerance, which keeps both servers in sync with each other leading to zero downtime if a server should fail.

- **Easily copy system data to recovery site**

  Having an offsite backup is a huge advantage if something were to happen to your specific location, whether it be a natural disaster, a power outage, or a water pipe bursting, it is nice to have all your information at an offsite location. Virtualization makes this easy by easily copying each virtual machines image to the offsite location and with the easy customizable automation process, it doesn't add any more strain or man hours to the IT department.