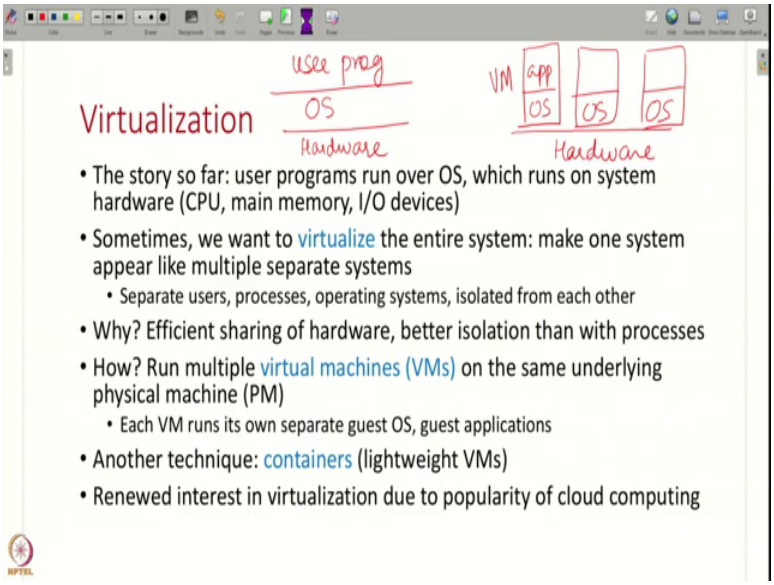
Design and Engineering of Computer Systems
Professor Mythilli Vutukuru
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture 10
Virtual Machines and Containers

Hello, everyone, welcome to the 10th lecture in the course design and engineering of computer systems. In this lecture, we are going to study the basics of what are virtual machines and containers. Let us, get started.

(Refer Slide Time: 00:27)



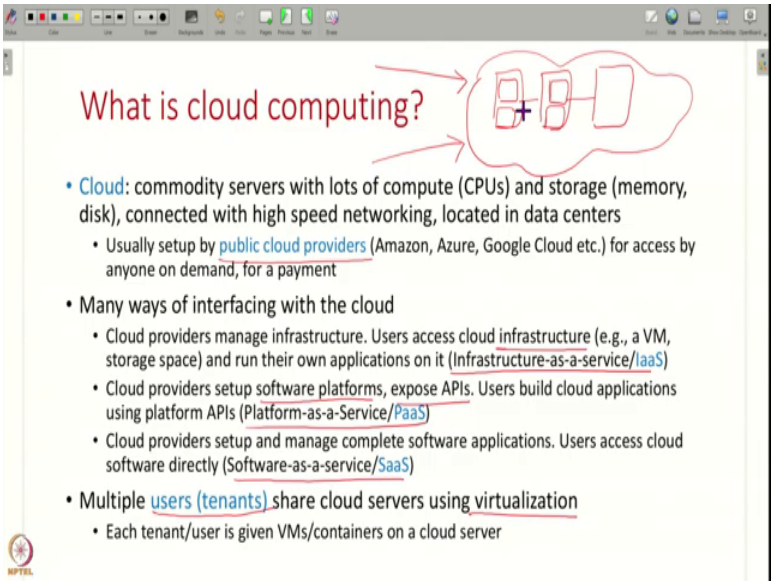
So, the story that we have seen so far, the mental model we had is that there are user programmes, which run on top of a system software like the operating system, which manages all the underlying hardware, like the CPU, main memory, IO devices, and so on. But sometimes what we want to do is we want to virtualize the system even more. That is, we want to have the underlying hardware, but we want to run multiple copies of the entire system stack on top of it.

That is we want to run different operating systems, each with their own applications and all of these operating systems are sharing the same underlying hardware. So, such instances of systems that we run are called virtual machines. So, this is a common requirement, we want to run multiple virtual machines, each with its own operating system on the underlying hardware or the underlying physical machine.

So, why do you want to do this? You want to do this to utilise the hardware better, if your CPU is lying idle due to one system, you might want to use multiple systems on the same underlying hardware to utilise it better, at the same time have better isolation from each other. If you just run multiple threads to use the hardware better, then you are not isolated from each other.

So, you want good sharing of hardware at the same time good isolation, more than what you just get with processes or threats. So, one way to do it is run multiple virtual machines, each with their own underlying guest operating system. Another way to do it is use what are called containers, which are lightweight virtual machines. And there is a renewed interest in virtual machines and containers today, due to what is called cloud computing, because these things form the building block of cloud computing.

(Refer Slide Time: 02:29)



So, let us see what is cloud computing? So, a cloud is nothing but a large data centre or a warehouse that has multiple pieces of hardware, like servers, and storage, all of these are connected by a high speed network. So, a cloud is nothing but a large set up of a lot of hardware, like CPUs. And this is usually set up by public cloud providers like Amazon, Microsoft, Google; all of them have their cloud offerings.

This is a public cloud that is set up by one of these big players and this is made available to other users for a payment on demand. So, you as a user can use this hardware after making a payment.

So, this is called cloud computing. And there are many ways in which a user can interface with the cloud. So, these cloud providers can just give you access to the hardware like they can give you like a virtual machine that is running on one of the cloud servers and on that hardware, on that infrastructure, you can install your own applications.

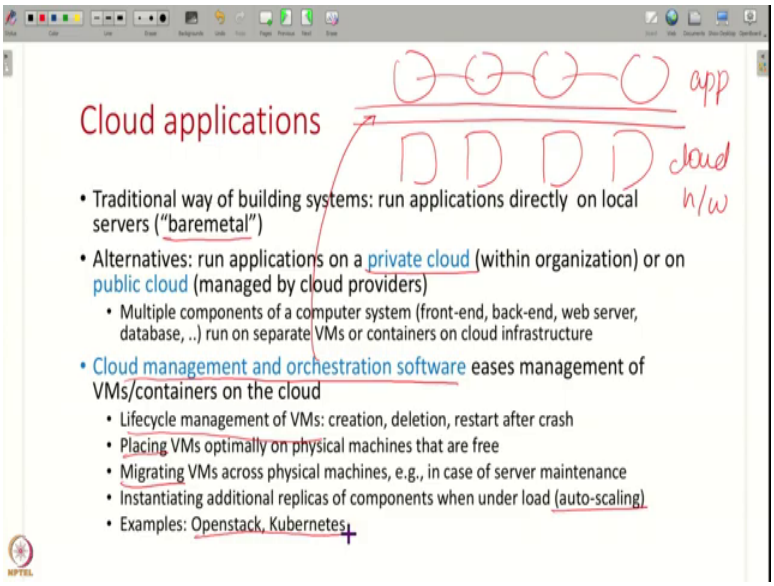
So, this model is called infrastructure as a service, where the cloud provider is only giving you access to the hardware and you are running your software on it. A different model is the cloud provider will set up both the hardware as well as a software platform. For example, if you want to run a web application, then the cloud provider might give you a software framework using which you can build web applications.

And using the API's, you can build your application and run it on the cloud. So, the cloud provider is now giving you hardware plus a software platform. But still you build your application using the API's, this is called platform as a service. Another thing is going even further, the cloud provider is giving you the hardware, as well as the complete software not just API is using which you build the software.

But the complete software is also provided by the cloud provider that is called software as a service. For example, if you use Google documents on the cloud. The software is running on the cloud is set up and managed by the cloud provider and you are using it. So, you can just use the hardware. You can use the hardware plus customise the software using the API's or you can log in and use the full software plus hardware bundle on the cloud.

These are different ways in which users can use a public cloud. And these multiple users also called tenants who share the cloud server that sharing is done via virtualization. For example, if different users are running their cloud applications, you want to separate them, isolate them as much as possible and so each user is given a VM or a container on the cloud.

(Refer Slide Time: 05:26)



So, there are the traditional way of building applications is, your application, all the different components, you run them directly on the hardware servers, that model is called the bare metal model. You run your applications on bare metal directly on the hardware, a different way that is being done today is to run applications on the cloud.

So, a cloud can either be a private cloud, that is, you have set up some hardware, a lot of servers, different servers, each with a lot of CPU cores, and memory, and all of that you have set it up, that is a private cloud, within your organisation, or it can be a public cloud, that somebody else has set up the hardware for you and you are accessing that.

So, applications can run either on a private cloud or on a public cloud. And on this cloud hardware, you can have multiple components of your system, your front end, back end, web server, database, all of these components can run on this underlying cloud hardware, your applications running on the cloud hardware.

So, what is the advantage you might ask I mean, instead of, why do not I just run bare metal? Why run it on the cloud? That is because usually, this cloud hardware is managed by a cloud management or orchestration software. So, there is another layer of software running here, which is the cloud orchestration software, which actually does take care of a lot of things for you, that makes it easier for you to run your application.

For example, if you are running your application inside a VM and this VM crashes, the lifecycle of the VM, creating it, deleting it, restarting it after crash, all of that is managed by this orchestration software, then if you want to move your component to another machine, finding a physical machine that is free and moving your VM to that physical machine, that these cloud software will do it for you, placing the VMMs on the underlying physical machines.

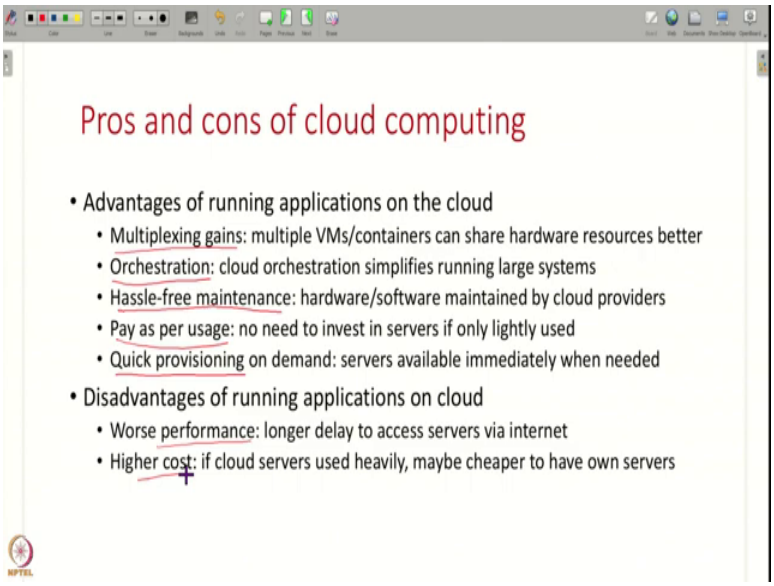
So, if you are running an application bare metal, and if your hardware resources are exhausted, then you have to find which machine is free and move your software there. But these cloud management software's, they take care of all of these things for you.

Then another example is migrating across machines, suppose your hardware, this hardware server has to go down for maintenance or repair, then you have to move whatever application is running on it to another location. But if you are running it on the cloud, the cloud management software will take care of moving these components to other machines.

Similarly, instantiating, extra replicas if your web server is overloaded; having another web server to share the load that is called auto scaling, scale up your system when under load, that is also done by cloud management software.

So, there are several examples of such software like OpenStack and Kubernetes, which, if you were running the application, bare metal, a lot of things that you would have to do yourself, these software's do it for you, they make it easy for you to run a large software application over multiple servers. So, this is the primary reason why people are preferring to run applications as cloud applications rather than as bare metal applications.

(Refer Slide Time: 09:00)



So, to summarise, here are some of the pros and cons of using applications on the cloud. Which is, of course, there are multiplexing gains, multiple VMMs can share the hardware resources better with better isolation, then you have this orchestration, moving software across different servers, all of this is automatically taken care of by cloud orchestration software.

Then, if you are in fact, using a public cloud and not a private cloud, you do not even have to worry about the hardware. The hardware is also taken care of by the cloud providers. It is all completely hassle free maintenance for you. The other advantages pay as you go model. If you only need few servers, you only pay for few servers in a public cloud. If you need more servers, you pay for more servers.

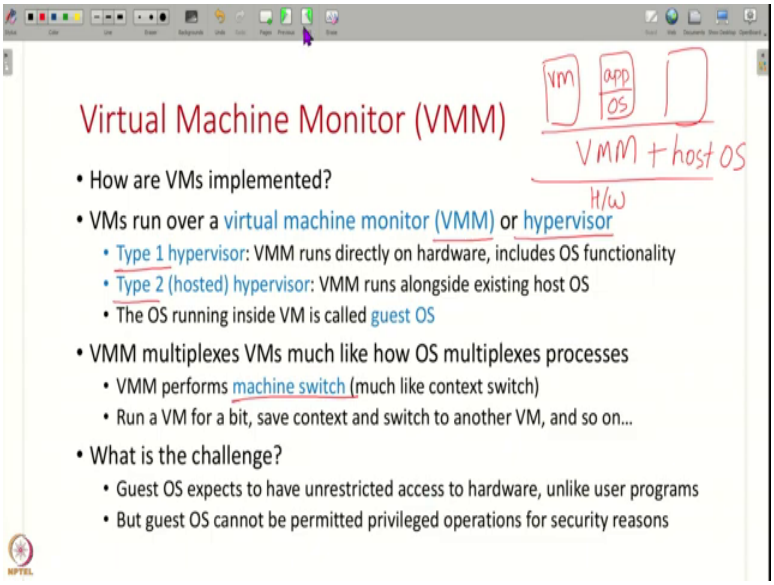
If your usage needs are less you can reduce your cost which you cannot do bare metal once you have bought all the servers, what do you do, they are with you and you are incurring the cost of their maintenance. Another advantage is quick provisioning. Now, suddenly the demand for your application has increased and suddenly you want more servers instead of you buying the server yourself, you can just pay for more and get more servers on demand on the cloud.

So, all of these are different reasons why people are moving their applications to the cloud. And of course, there are certain disadvantages also, which is performance for cloud applications is

usually more tricky because you are sharing the server, and you have to access a server that is far away, there are some delays.

So, performance tuning is a little bit harder. And also, the cost may also come out to be higher. If you are heavily using the cloud, you may end up paying a lot, and it might be cheaper to just have your own setup locally. So, these are some of the pros and cons of cloud computing. But despite some of these disadvantages, cloud computing is fairly popular today to run a lot of applications and a lot of computer systems you use in your daily life are actually hosted on either a private cloud or a public cloud.

(Refer Slide Time: 10:02)



So now, let us try to understand briefly the building block for cloud computing, which is virtualization. So, how are virtual machines implemented? So, virtual machines are managed by a software called a Hypervisor or a virtual machine monitor. That is you have the virtual machine monitor, on top of it, you will host multiple virtual machines, each with their own underlying operating system and applications.

Now, this hypervisor can either directly run on top of the hardware and do the functionality of an OS also, or it can run along with an existing operating system called the host operating system. So, a type one hypervisor runs directly on the hardware, a type two or a hosted hypervisor,

instead of re implementing the OS functionality, it will reuse an existing operating system and run along with it.

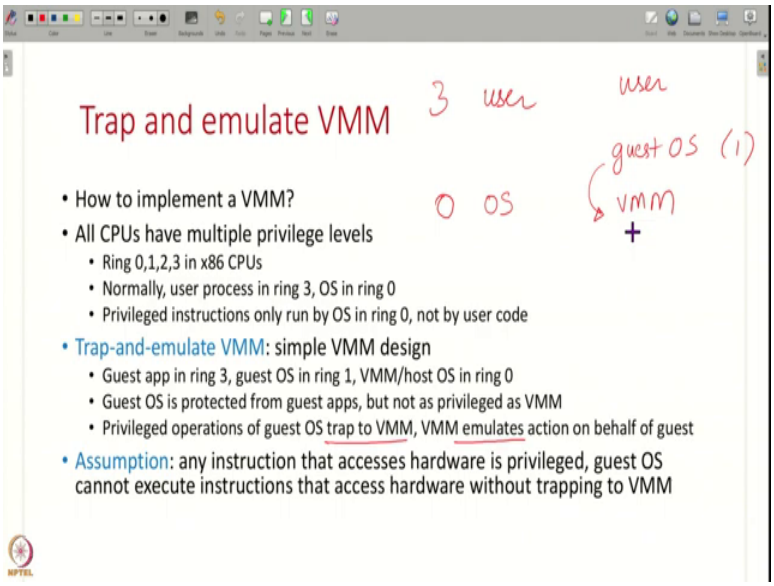
So, whatever whether it is type 1, type 2, the underlying concept is, of course, the same of a hypervisor, which is a hypervisor will multiplex multiple VMMs on the hardware, just like how an OS multiplexes processes if you realise it is the same thing, we also said an OS virtualizes the CPU for different processes.

Similarly, the VMM virtualizes, the CPU and memory and everything for different VMMs by doing machine switch, it will run a VM for some time, save it state, switch to another VM, save it state switch to another VM and so on. Just like how an OS multiplexes the CPU and memory across processes a VMM also does the same thing across VMMs.

But an important difference is that a guest OS expects full control of the hardware, whereas the user programme does not. Therefore, it is a little bit more tricky to multiplex hardware across multiple guest OS's, each of which expects to have full control of the hardware at the same time, you cannot give full control to any one guest OSs.

Why? Because you want to isolate these VMMs from each other, you do not want one user in a cloud to have full access to the hardware and deprive another user of the hardware, or do something bad, we do not want that. So, we want to restrict access to a piece of software that expects unrestricted access that is the challenge here with designing VMMs.

(Refer Slide Time: 13:18)



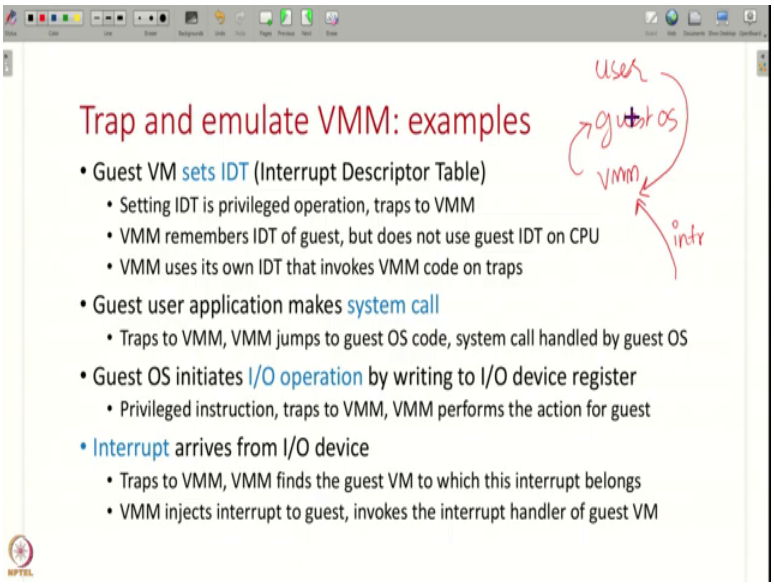
So, the standard technique to build VMMs is what is called trap and emulate that is you will just create another layer in between, another privilege level in between will be used for the guest operating system. So, normally what you do you have ring 0, you have your OS running and then like an x86 CPUs have multiple privilege levels or rings and ring 0 you have OS and ring 3 you have user applications.

With virtualization, what you can do is you can run the VMM or the host OS in ring 0 and the guest OS in an intermediate ring like say ring 1 or something and then the user applications can run in ring 3. So, you are able to protect the guest OS from the user applications. At the same time, the guest OS is not very powerful, any privileged operation it does will trap to the VMM.

So, the privileged operations trapped to the VMM and the VMM will emulate the action, trapped VMM and emulate the action on behalf of the guest OS. If the guest OS wants to do any privileged operation, it will trap to the VMM, VMM will do the operation on behalf of the guest. So, for this to work, of course, the assumption is that any instruction that accesses hardware, if whenever the guest OS tries to access the hardware, it will always trap to the VMM.

It will be a privileged operation that will trap to the VMM. This will ensure that the VMM can restrict the access of the guest.

(Refer Slide Time: 14:58)



So, some examples of how trap and emulator VMM work anytime the guest VM tries to do any privileged operation. For example, it tries to set up the interrupt descriptor table we have seen the interrupt descriptor table tells the CPU which OS core to invoke for a certain event like a interrupt or a system call.

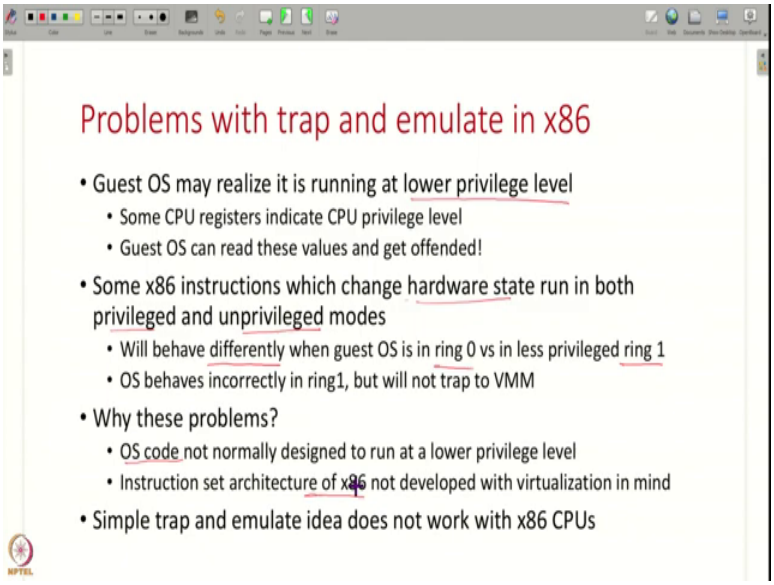
So, whenever the guest VM tries to set this up this privileged operation it will trap to the VMM and the VMM will set up its own identity. Similarly, when the guest OS tries to do any IO operation, it will trap to the VMM and the VMM will do the IO operation on behalf of the guest. So, any hardware access that the guest tries to do it will trap and the VMM will do it on behalf of the guest.

Similarly, whenever there is a system call, it will once again trap to the VMM, VMM will invoke the guest OS code. Any interrupt arrives that will trap to the VMM, VMM will find which guest code to run to handle this interrupt, which guest VM to deliver this interrupt to and which interrupt handler to run the VMM will take care of it.

So, between the user and the guest OS, all communication happens via VMM, you will first come to VMM. Then VMM, which is the most privileged software will take you to the guest OS. Similarly, any hardware interrupt will come to the VMM, VMM will take you to the guest OS. In

this way everything is trapping to the VMM and the VMM is emulating it, this is the trap and emulate idea.

(Refer Slide Time: 16:32)



But of course, this is not a straightforward to implement in modern CPUs. There are multiple reasons which we need not discuss in a lot of detail in this course. But some of the high level reasons are that some guest operating systems they do not run well at a low privilege level. A guest OS is built to run at the highest privilege.

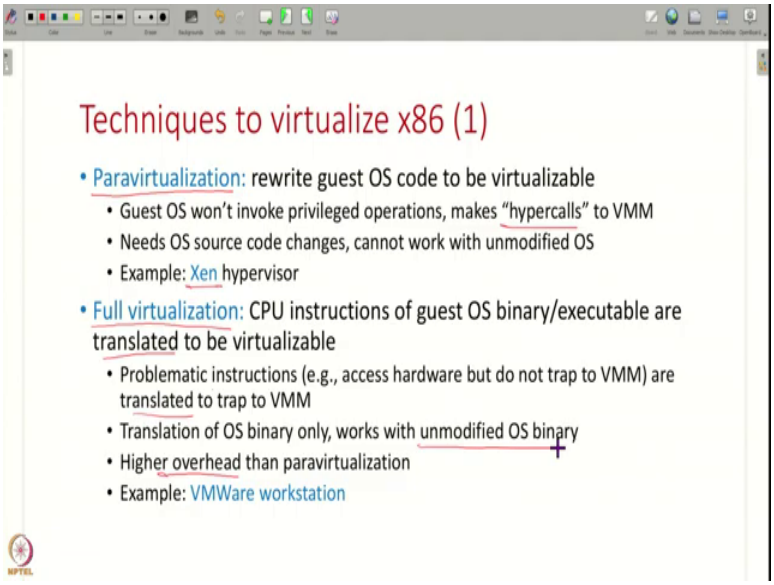
For example, some CPU registers, you can read and find out that oh, I am not ring 0, I am ring 1 and the guest OS may not run correctly. There are some instructions, which actually do not trap to the VMM instructions, which accesses the hardware state, but still need not always trap to the VMM they will run both in privileged and un-privileged modes, which is not what we want.

Whenever the guest OS is trying to access hardware, we wanted to trap to the VMM so that the VMM can restrict hardware access, but some x86 instructions, what will they do, they will access hardware, they will run in privileged mode and un-privileged mode and they will behave differently when running in a privileged mode an non privileged mode, because of which what will happen your operating system will run incorrectly in ring 1 it will not trap the VMM to access hardware the way you want it to it will run incorrectly in ring 1.

So, the summary is that guest operating systems are not designed to run at a lower privilege level, they are always built to run at ring 0 and if you run them at a lower privilege level either, you know they might detect it and you know not do their work properly or there might be some incorrect behaviour on part of the guest OS.

So, therefore, for these reasons, whether it is operating systems or x86 architecture, they are not built with this virtualization in mind and therefore, they will not behave correctly all the time. Therefore, virtualizing x86 building a trap and emulate VMM with x86 is actually quite challenging and modern systems have, of course overcome this challenge in multiple ways.

(Refer Slide Time: 18:36)



So, one technique. Now, given all of these challenges, how do you build a VMM with x86? There are many techniques and one of the techniques is what is called para-virtualization. That is you say okay, fine, let me just go ahead and change the guest operating system and rewrite it so that it does not invoke these privileged operations, and it does not have all of these issues.

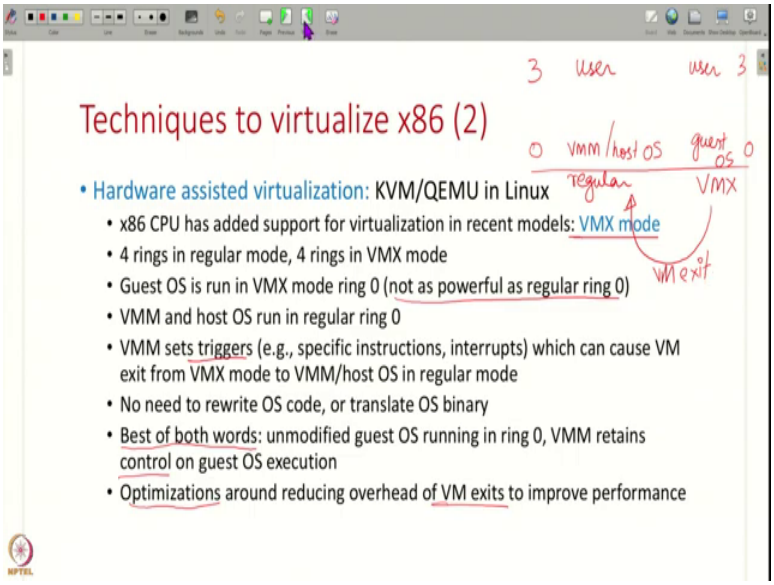
So, that is called para-virtualization where you go and change the guest OS so that it runs in an unprivileged mode. So, such guest operating systems will not invoke privileged operations, they know that they are unprivileged, and they will make hyper calls, like system calls a guest OS will make hyper calls to the VMM to get some work done. So, this is one technique.

This is used by the Xen hypervisor. But of course, the problem with this is that you have to rewrite the guest OS code. Another way if you do not want to change the guest OS code is what is called full virtualization, which is you take your guest OS binary and you translate it. If there are some problematic instructions that access the hardware but do not trap to the VMM or instructions that are checking the CPU privilege level all of these things you want to hide from the OS that you are running at a lower privilege level.

So, all those problematic instructions will be translated, whenever the instruction is running. You will translate it to something else, and then you will run it. This is of course higher overhead, whenever every CPU instruction, somebody has to translate it as and when it is running. Therefore, this is higher overhead than para-virtualization.

But you can work with unmodified operating systems; you do not have to rewrite your operating system code that has the advantage of full virtualization. So, VMware was the first company that pioneered this approach and the earlier VMware hypervisors use this full virtualization idea.

(Refer Slide Time: 20:41)



And the third technique that is most commonly used today is what is called hardware assisted virtualization. That is, the CPU makers like Intel x86 CPUs have realised that okay, there are certain problems with the CPU instructions, which is making it harder to virtualize and they have changed their CPUs to make virtualization easier.

So, what modern CPUs have done is they have added a separate mode for virtualization, it is called the VMX mode in x86. So, what happens with the VMX mode now in a regular mode of the CPU, you have 4 privilege level say ring 3 all the way to ring 0, your VMM, host OS is running in ring 0, and your user applications are running in ring 3, then what happens is this VMX mode creates another set of 4 rings like this.

You again have in VMX mode, you have ring 0 to ring 3, then your guest OS can run in this special VMX mode ring 0 and your guest user applications can run in VMX mode, ring 3. So, what is this doing? This ring 0, this VMX mode ring 0 is not as powerful as your regular ring 0. So, the guest OS wants to run in ring 0, you give it a separate ring 0, which is not as powerful as the regular ring 0.

So, now the guest OS is happy, that is it is running in ring 0, all of its instructions, privileged instructions will run fine. But the VMM still retains control VMM can set triggers, which it can say that when certain instructions happen when certain events happen, that I do not trust, the guest OS to do on its own. For such events, you can exit back into the VMM.

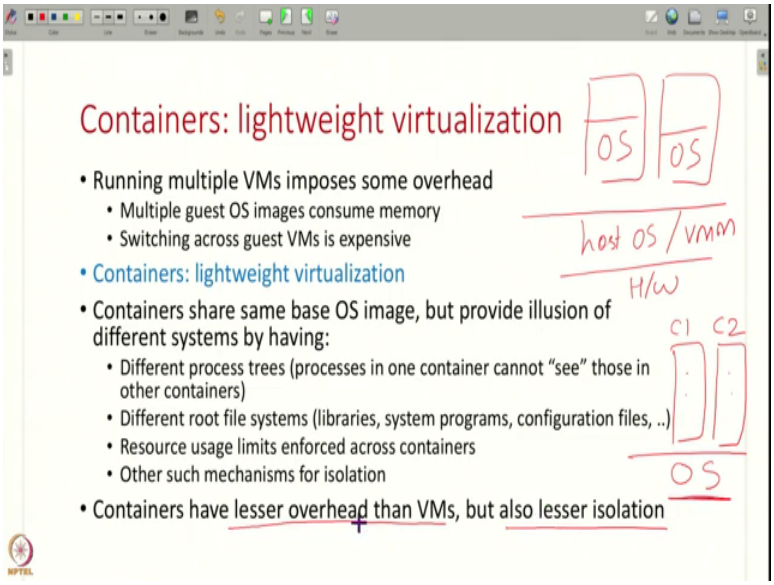
So, this ring 0 is not as powerful as the regular ring 0, the VMM can still set triggers that cause a VM exit back into the regular ring 0 and the VMM can handle these events. So, in some sense, this is the best of both worlds. You have an unmodified guest OS which is running in ring 0, the OS is happy, everything is going fine, you do not have to rewrite the operating system to run at a lower privilege level, you do not have to translate the operating system instructions, which are problematic, you do not have to do any of that you have your regular OS that is running in ring 0.

But at the same time, this VMX ring 0 is not as powerful as the regular ring 0. So, the VMM the virtual machine monitor retains control, it can prevent the guest OSs from having unrestricted access to the hardware. So, hardware assisted virtualization, of course, understanding this you have to take a full course on virtualization.

We cannot cover more detail than this in this course, but it helps you to know that there is a way to run virtual machines today with as low overhead as possible, but at the same time providing all the isolation guarantees that we need using hardware support. And of course, there are also some optimizations you may have to do to improve performance.

For example, if you are seeing a lot of VM exits, every time you are going switching back and forth, this VM exit is an expensive operation, like a context switch or switching across processes. So, you might want to minimise these VM exits to improve performance.

(Refer Slide Time: 24:11)



So the next concept that we look at is containers, which is a way to do lightweight virtualization. So, with VMMs, what do you have? You have multiple guest operating systems, all of them residing in memory and all of them sharing the underlying hardware via say another, host OS or VMM and so on.

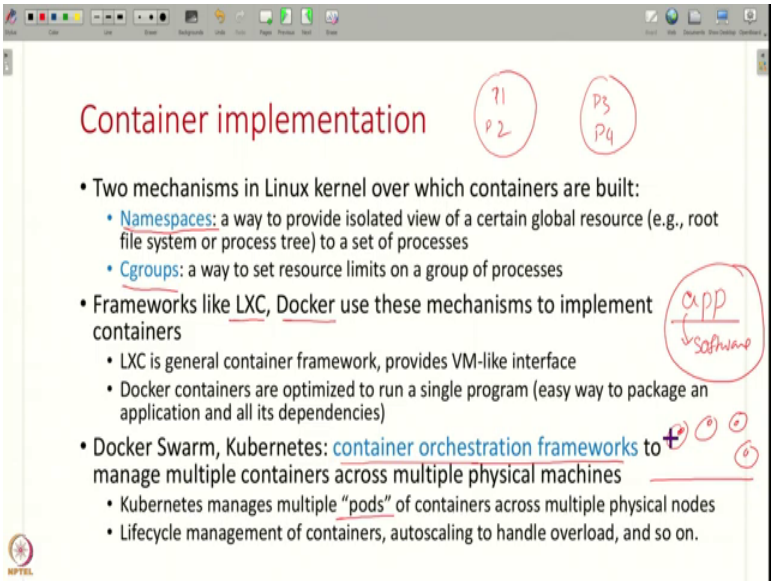
So, you have multiple pieces of operating system code and overall your system memory consumption becomes very high and switching across this VMMs is also very expensive, a VM exit, it is like a context switch, but it is much more expensive, because you have to store more context, you have to restore more context and so on. So, running multiple VMMs in general is there is some overhead associated with it.

So, containers get rid of that overhead. So, containers is a way to do lightweight virtualization containers have lesser overhead than VMMs but of course also there is a cost you have lesser isolation, but also lesser overhead. So, what are containers? Containers is basically you have the same underlying operating system, but you have different views of this underlying operating system.

That is, there is the same underlying OS but the process tree the list of processes or the list of files, the file system, the libraries, all of these things you can separate out in different containers. So, a container C1 and a container C2 can have a different view of what processes are there in the system, the processes and C1 will not see the processes in C2 you can have different libraries you can have different files, the applications in C1 will not see the files of C2 and so on. You can provide some basic isolation while sharing the underlying OS image.

And you can also ensure resource usage limits that is you can say this container can only consume this much CPU, this container can only consume this much memory that let you also do some basic isolation across containers. But at the same time, of course, since the underlying operating system is shared, you do not have full isolation like you have with VMMs, but you have lesser isolation and lesser overhead also compared to virtual machines.

(Refer Slide Time: 26:29)



So, these containers are implemented using two mechanisms in the Linux kernel. So, the Linux kernel today provides what are called namespaces. That is you can isolate certain resources for different processes, you can say, these set of processes will only see this process tree or this file system and another set of processes will see, you know, P1, P2 will see only certain resources of the system and other processes P3, P4 will see only a different slice of the system.

In this way you can isolate processes into namespaces in Linux today. And you can also set limits on how much of hardware resources each of these processes will use. You can say this group of processes can only use 2 CPU cores or 1 GB of memory, you can set such resource limits. So, you can isolate their view, and you can limit their resource usage.

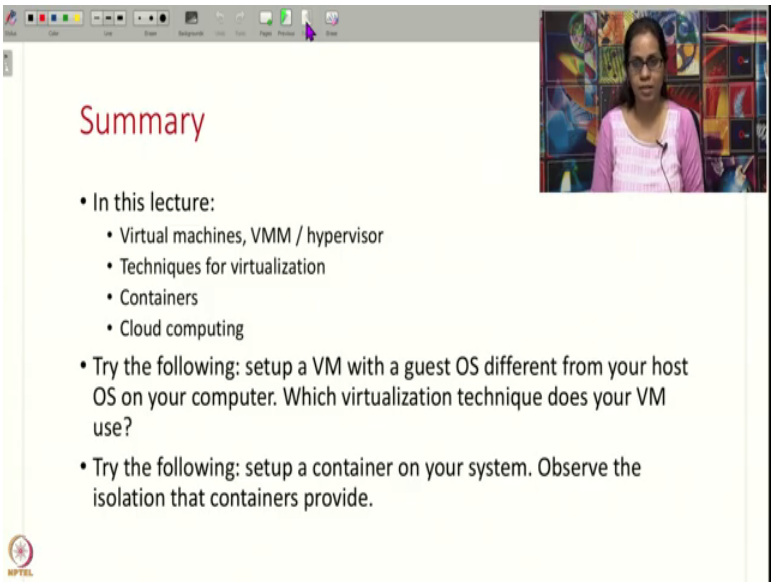
Using these two mechanisms, you can build a container framework. So, there exists already implementations of containers like LXC and Docker, which use these mechanisms, which use the techniques available in the Linux kernel to do this isolation and limit resource usage. So, there are different container frameworks have different purposes, they are built for different ways of using them. For example, LXC is a general container framework.

It gives you a VM like shell interface, whereas our Docker container is for running a single application. So, inside a Docker container a single application and all of its files, dependencies, libraries, everything can be packaged, all that the application needs, all the system software that the application needs, can be packaged into one Docker container.

And this can run on the underlying operating system. So, different containers are used for different purposes. And of course, you also have container orchestration frameworks, just like for VMMs, there are software like Docker Swarm, or Kubernetes, which let you manage, you have multiple containers on multiple machines and these frameworks will let you manage all of these containers. Like start containers on machines move containers across machines.

So, Kubernetes has a concept of what are called pods. That is you place these pods on the underlying hardware and you can move these pods around and manage the lifecycle of these pods scale these pods when there is overload. So, these container orchestration frameworks make it easy for you to run a large application with many components on an underlying hardware.

(Refer Slide Time: 29:12)



So, that is all I have for today's lecture. In today's lecture, we have seen what are VMMs, what is a hypervisor, and what are containers and how these techniques of virtualization VMMs and containers are used for cloud computing. In a cloud you have a large pool of hardware resources, which are shared using VMMs and containers.

And cloud computing makes it easy for you to run your applications because the cloud orchestration software actually manages the multiple different components of your application on the underlying hardware. So, as an exercise, please try to set up a VM or a container on your machine. There are many resources available online and many free open source frameworks available for VMMs and containers.

You can install them on your machine and understand what is this isolation that VMMs and containers provide and what are some of the overheads of them, if you can just play around with them that will help you understand the concepts in today's lecture better. Thank you very much.