

Slide 1

Part 6: Real-Time OS & Virtualization

- **What is a Real-Time OS (RTOS)?**
- Real-Time Process Specification
- Real-Time CPU Scheduling
- Virtualization

Operating Systems1.1Part 1 Introduction to Operating Systems

Slide 2

used to be called embedded systems

Cyber-Physical Systems

- Physical/Engineered systems whose operations are monitored, coordinated and controlled by a reliable computing and communication core
 - Automotive Systems (Autonomous driving, Parking assist, Airbag controls)
 - Avionics (Flight navigation & control)
 - Manufacturing Systems (Robotics, Process controls)
 - Medical Systems (Robotic surgery, devices)
 - ...

Operating Systems1.2Part 1 Introduction to Operating Systems

Trend 1: Proliferation of sensor devices. As these devices include embedded processors that are increasingly becoming smaller and also more powerful and plentiful; that is, they are everywhere and can support complex functionalities. They range from simple environmental monitoring, to medical devices, to cargo ships, to factories, to smart spaces that are for assisted living for elderly or disabled.

Trend 2: The second trend is that these systems are increasingly being integrated as system of systems. These integrations can range from simple interconnection to exchange sensor data, to inter-operations to support remote command and control, to total integration to serve as a coherent, large-scale complex system.

Relevance of Real-Time

- Common to many application examples we saw in the previous slide:
 - Collect data from various sensing devices
 - Execute control law(s) to determine response
 - Send actuator commands in a **reasonable amount of time**

Pacemaker timing diagram

Pacemaker

Collision avoidance and braking

Collision Warning with Auto Brake

Operating Systems 1.3 Part 1 Introduction to Operating Systems

*we all
agree physical
systems have
all these
features, but
most of them*

What is a Reasonable Time?

- What is the functionality?
 - Collision avoidance in automotive (milliseconds)
 - Pacemaker (up to a second)
 - Robotic surgery (varies greatly depending on the target)
- What are the environment constraints?
 - Available computing and communication resources
 - Timing characteristics of sensors/actuators/operations
- Failure-mitigation strategies?
 - Time to detect and recover from failures
 - Example: execution replication for redundancy

Operating Systems 1.4 Part 1 Introduction to Operating Systems

*what can
the system*

Common requirements among several of the CPS applications is that of real-time. Meaning the applications are required to process inputs and generate outputs within a pre-defined amount of time. We can call this time the application deadline. Not only does the application have to produce a functionally correct output, but it must do so within this deadline. So, for such applications, both functional/logical correctness as well as temporal correctness or timeliness are equally important.

Couple of examples of such real-time CPS applications are given here. One is that of a pace-maker, which is a medical device embedded in a human body. It monitors the electrical pulses generated by the brain to stimulate the heart muscles. Whenever the electrical pulses do not arrive in time from the brain, the pace-maker provides a replacement pulse to stimulate the appropriate chamber of the heart. This is a very critical device with some extremely precise real-time requirements, usually in the high milliseconds range.

Another example is that of a collision detection and avoidance system in autonomous driving. Sensors, usually cameras and lidars, provide information about the environment. This data must be perceived and a decision taken to control the speed of the ego vehicle depending on whether its path is clear or has other objects which can lead to potential collisions. Again a very critical system with real-time requirements, and depending on the speed of the vehicles involved, the real-time requirements can be below 100 milliseconds.

Although several CPS applications have hard real-time requirements (deadlines), the notion of these requirements are dependent on various factors.

It depends on the type of functionality. The notion of real-time could range from a few milliseconds to potentially even few minutes. Longer time durations would usually then not be considered as real-time requirements, but instead would be regarded as design-time planning.

It also depends on the constraints imposed by the surrounding environment, such as available computing and communication resources, the timing characteristics of the mechanical devices and processes, etc.


Finally, it can also depend on some design considerations such as techniques for failure mitigation. This may need to consider the timing overhead of strategies for detecting and recovering from failures. An example could replication of computation for redundancy.

Slide 5

not used in OS like Windows, Mac, Linux, iOS, Android, etc.
 rule on avg case not worst case
 ↑

Common Misconception

- Real-Time \neq Fast
- Real-Time = Predictable even in the worst-case



"Man drowned in a river with average depth 20 cms"

Operating Systems
1.5
Part 1 Introduction to Operating Systems

A common misconception is that real-time requirements imply fast execution on an average or simply fast response times. This is not true at all.

Real-time requirements imply guarantees on the worst-case response times, AND NOT average case response times. So, in the worst-case, the execution of the application must be completed within the pre-specified deadline.

Slide 6

Real-Time CPS / Real-Time OS

- **Definition:** System whose correctness depends not only on the logical/functional aspects, but also on the temporal aspects
 - Application has deadlines that must be met
 - A real-time OS (RTOS) provides OS services to such systems (e.g., FreeRTOS, MicriumOS, ...)
- **Key performance measure for RTOS**
 - Timeliness/Predictability on timing constraints (deadlines) all for guarantee it will work in worst case
 - Significance of worst-case over average-case
 - Deadlines are a function of application requirements

CE2005/CZ2005 Operating Systems
1.6
Part 1 Introduction to Operating Systems

In summary, the key focus of real-time CPS, and hence real-time OS, is that of ensuring predictability over the timing constraints. In the worst-case, the timing requirements of the application must be met. This worst-case must take into account the impact from mechanical devices, computation (both software and hardware), communication, etc.

Part 5: Real-Time OS & Virtualization

- What is a Real-Time OS (RTOS)?
- **Real-Time Process Specification**
- Real-Time CPU Scheduling
- Virtualization

Operating Systems 1.7 Part 1 Introduction to Operating Systems

R : process release time
 D : relative deadline \rightarrow time from release to when it should finish ($R+D$)
 C : execution requirements \rightarrow CPU burst length
 \rightarrow to make sure process finish by $R+D$
depends on input, instruction stream, state of data

RTOS (Real-Time) Process

- **Definition:** A real-time process is specified as $\langle R, C, D \rangle$, where R is process release time, C is execution requirement and D is relative deadline
 - Requires C time units of CPU in the interval $[R, R+D]$
 - How does one determine these parameters?
- Example: $P_1 \langle 0, 5, 10 \rangle$, $P_2 \langle 5, 10, 35 \rangle$, $P_3 \langle 20, 10, 30 \rangle$

P_1 release at 0, P_2 release at 5, P_3 release at 20.
 P_1 deadline at 10, P_2 deadline at 35, P_3 deadline at 30.

n/s In this lecture, we assume processes only have a single CPU burst; C is the duration of this burst

CE2005/CZ2005 Operating Systems 1.8 Part 1 Introduction to Operating Systems

We now consider how to model a real-time RTOS process. We need to know when the process is released for execution in the system. We need to know how much CPU cycles it needs in the worst-case. We also need to know by when it requires these CPU cycles. Only with these parameters can the RTOS manage the processes to ensure that their real-time requirements (deadlines) can be met.

It is an interesting question to understand how one could determine these process parameters. R and D are straightforward. Usually they are derived depending on the application requirements. How often or when the application needs to run and by when should it complete. However, C is challenging to determine. It is the worst-case execution requirement, meaning the maximum CPU cycles that the process will ever require. This is usually very hard to determine.

In this lecture, to keep things simple, we ignore I/O bursts and assume that all processes only have a single CPU burst.

Recurrent Real-Time Process

- **Nature of real-time processes**
 - Collect data from sensing devices, execute control laws to determine responses, and send actuator commands in reasonable time
 - **Repeat the above steps regularly**
 - Examples: airbag control, flight control, collision avoidance, pacemaker, etc.
- **A recurrent real-time process**
 - **Executes some function repeatedly over time**
 - Each instance of execution is a real-time process $\langle R, C, D \rangle$

CE2005/CZ2005 Operating Systems 1.9 Part 1 Introduction to Operating Systems

In many real-time applications, the nature of a real-time process is repetitive. Meaning it needs to execute the real-time function repeatedly over time. Each instance of such a recurrent real-time process is a real-time process with parameters $\langle R, C, D \rangle$.

Depending on the nature of repetition, there are two broad types of recurrent real-time processes. We will look at these in the next couple of slides.

Periodic Real-Time Process

- **Definition:** A process that **repeats periodically**
 - Processes generated by a **time-triggered phenomena** (sensor sending data periodically) *timer triggered*
 - Example: Perception function for collision detection
- A periodic process is specified as $\langle T, C, D \rangle$, where T is process period, C & D are as defined earlier
 - Real-time processes are released at $R=0, T, 2T, \dots$
 - Example: Periodic process $P \langle 10, 5, 7 \rangle$

CE2005/CZ2005 Operating Systems 1.10 Part 1 Introduction to Operating Systems

One type of recurrence is the periodic recurrence, in which real-time processes are released in the system periodically at fixed time intervals. Several time-triggered functions fall in this category.

Sporadic Real-Time Process

• **Definition:** A process that **repeats sporadically with a minimum gap between releases**

- Processes generated by an **event-triggered phenomena**
- Example: Anti-lock braking function in automotive

• A sporadic process is specified as $\langle T, C, D \rangle$, where T is minimum release-separation time

- Real-time processes are released with a min. gap of T
- Example: Sporadic process $P \langle 10, 5, 7 \rangle$

CE2005/CZ2005 Operating Systems 1.11 Part 1 Introduction to Operating Systems

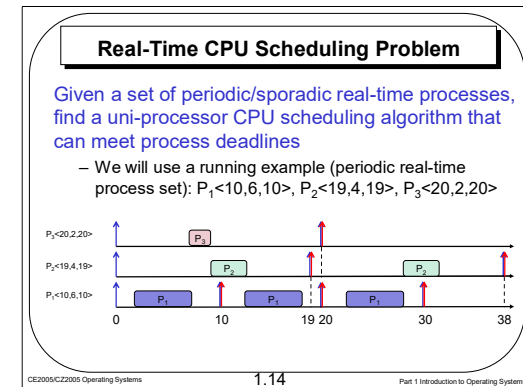
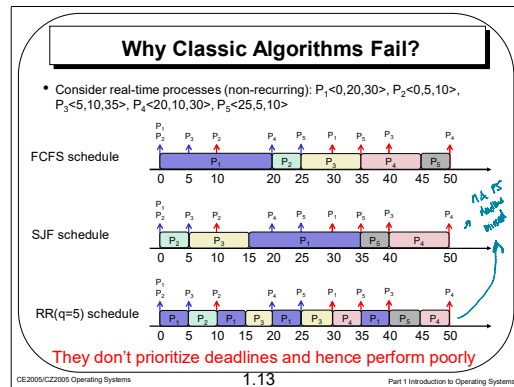
Another type of recurrence is the sporadic recurrence, in which real-time processes are released in the system sporadically with a minimum guaranteed separation between successive releases. Event-triggered functions fall in this category. They are induced by humans in many cases (examples include driver pressing brakes or pressing some activation button).

Part 5: Real-Time OS & Virtualization

- What is a Real-Time OS (RTOS)?
- Real-Time Process Specification
- **Real-Time CPU Scheduling (short-term scheduler)**
 - **Fixed-priority scheduling**
 - **Dynamic-priority scheduling**
- Virtualization

Operating Systems 1.12 Part 1 Introduction to Operating Systems

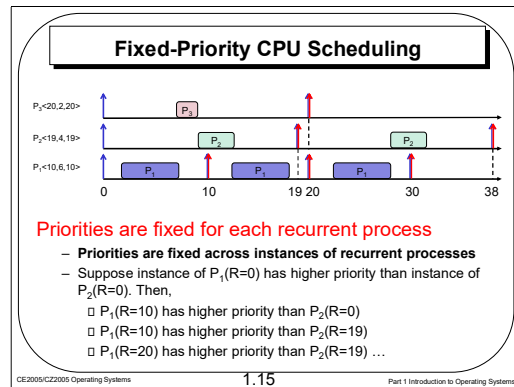
In this section we focus on the problem of CPU scheduling, that is the short-term scheduler from Week 3. We see how the algorithms we studied in Week 3 perform for real-time processes and what new algorithms we can consider for this specific application domain.



Lets take a closer look at how the classic CPU scheduling algorithms that we learned in Week 3 fair for the real-time process scheduling problem. Are they effective in meeting process deadlines?

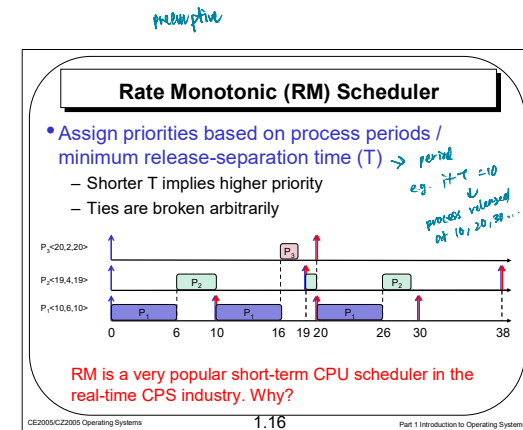
None of them do well in terms of the performance measure for real-time processes, that is meeting deadlines. This is expected, because none of them focus/prioritize on deadlines.

In this section we will focus on the short-term CPU scheduler (ready queue scheduler) for a set of periodic real-time processes scheduled on a uni-processor CPU. Although example focuses on periodic processes, the same techniques can be applied to sporadic processes as well.



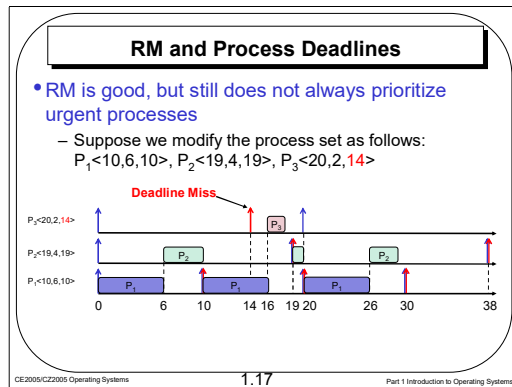
Fixed-priority CPU scheduling is a class of schedulers in which priorities are fixed at the level of each periodic/sporadic process. So priorities don't change across process instances.

We will look at two examples of this class of schedulers.

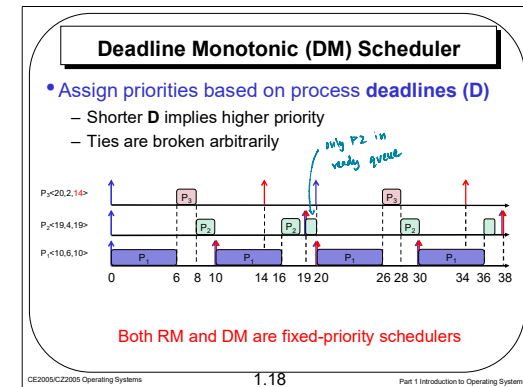


Under RM scheduling, priorities are assigned based on the process periods or minimum release-separation times. Smaller this value, higher is the priority. Ties are broken arbitrarily.

RM is a popular choice in the real-time CPS industry. For high-priority periodic/sporadic processes it gives a very predictable schedule. So important processes can be given high priority under RM.

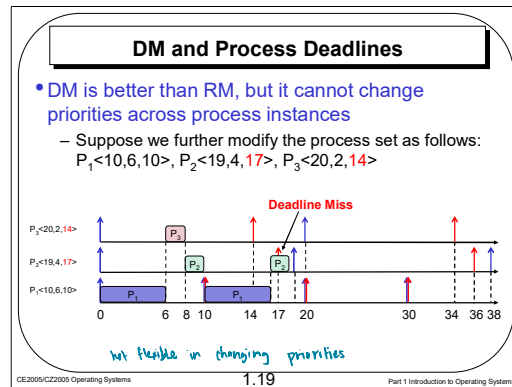


Since RM ignores the relative deadline parameter D , it can potentially miss deadlines for some process sets as shown in this slide.



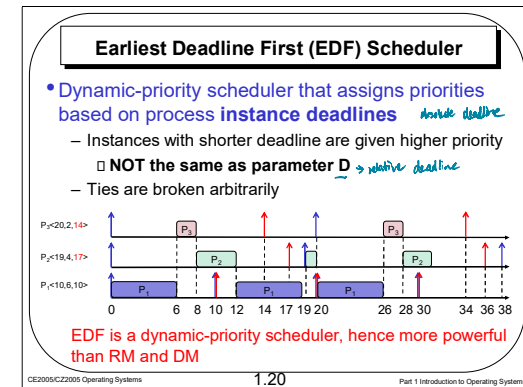
A simple extension of RM called, Deadline Monotonic or DM scheduler, solves this problem. It assigns priorities based on the relative deadline parameter D for periodic/sporadic processes instead of the parameter T .

Slide 19



Since DM is unable to adapt priorities at the level of process instances (that is, priorities are fixed for each recurrent process), it can potentially miss deadlines for some process sets. Here is a simple example, extending from the previous example.

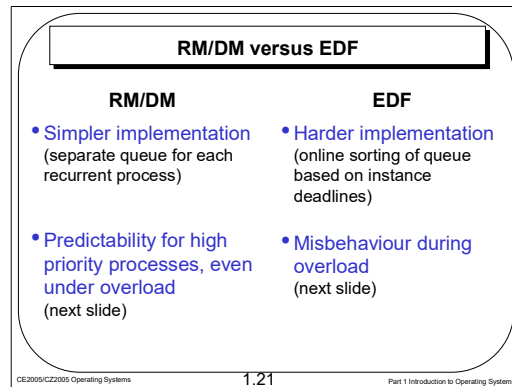
Slide 20



Best for single- and real-time OS

EDF is a dynamic-priority scheduler that changes priorities across process instances based on their actual deadlines. This allows a lot of flexibility to EDF and hence it is able to perform really well in terms of meeting deadlines.

Then, why is RM more popular in the real-time CPS industry? Lets take a closer look at the advantages and dis-advantages of these two classes of schedulers.

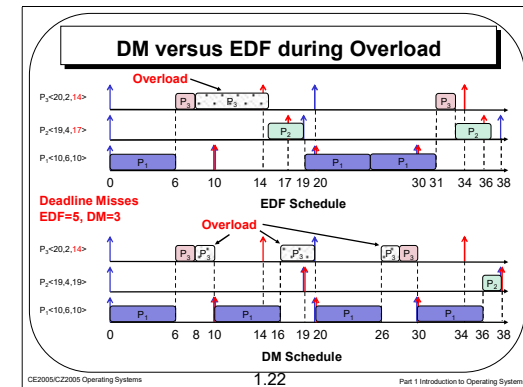


RM/DM are easy to implement. A simple priority-based queue data structure would suffice. There can be one queue per recurrent process. In fact, efficient $O(1)$ time complexity implementations are available for RM/DM using bitmaps.

EDF implementation requires an online sorting strategy, because process instance deadlines are only known once the processes are released in the system (true for sporadic processes). Further it is not possible to separate the instances into separate queues offline, like in RM/DM. The best known time complexity for implementation of EDF is $O(\log n)$, where n is the number of periodic/sporadic processes in the system.

Besides, RM/DM has better predictability for high-priority processes, even when there is overload. Lets see this in the next slide.

If you are further interested in this topic, I would encourage you to read this paper: [G. Buttazzo, "Rate monotonic vs. EDF: Judgement Day", Real-Time Systems Journal, 2005.](#)



Suppose the first instance of recurrent process P_3 overloads, meaning it executes more than its planned C . Then, the impact on subsequent deadlines is very different under fixed-priority (DM) scheduling and dynamic-priority (EDF) scheduling. This is shown in this slide.

Under EDF such overload can affect all subsequent deadlines across all recurrent processes.

Under DM such overload has no impact on recurrent processes that have higher priority than the overloading process. In this example, P_3 has no impact on P_1 because P_1 has higher priority than P_3 . So all deadlines of P_1 are met.

Part 5: Real-Time OS & Virtualization

- What is a Real-Time OS (RTOS)?
- Real-Time Process Specification
- Real-Time CPU Scheduling
- **Virtualization**

Operating Systems
1.23
Part 1 Introduction to Operating Systems

In this section we focus on the concept of virtualization. We understand what this concept means, what are the different types of virtualization, and some examples from the real-world for such virtualization techniques. Note virtualization is a very important concept in edge/cloud computing, which is very popular as of today.

For an excellent overview of virtualization, please have a look here:
<https://www.ibm.com/sg-en/cloud/learn/virtualization-a-complete-guide>

What is Virtualization?

- **Definition:** It is a technique that uses software, called **Hypervisor (virtual machine manager or VMM)**, to create abstraction of hardware
 - Hardware is divided into multiple virtual computers, called **Virtual Machines (VMs)**
 - Each VM runs its own OS, called **Guest OS**, and behaves like an independent computer
 - Application processes can run on the guest OS as if it is an independent computer
 - Each VM is using only a portion of the actual hardware

Is a general OS also using virtualization?

CE2005/CZ2005 Operating Systems
1.24
Part 1 Introduction to Operating Systems

Virtualization is a generic technology/concept in which specialized software is used to abstract computer hardware components, so that other software (guest OS, applications) running on this virtualized hardware only see an independent computer. In reality though, this independent computer (called a virtual machine) is actually sharing the underlying hardware with other virtual machines. The interactions between these virtual machines and how they share the actual hardware is managed by the specialized software called Hypervisor (more on this later).

Interesting question is whether a general purpose OS (what we saw so far) is also using virtualization? The answer is yes and no. As a concept it is indeed also virtualizing the underlying hardware for application processes. But, the term virtualization is commonly referring to a specific technology that enables this abstract view of the hardware as virtual computers in which independent guest OSes can then be deployed.

Functions of a Hypervisor

- **Creation and management of VMs**
 - Allocating hardware resources for VMs
 - Executing instructions on the hardware on behalf of VMs (VMs may be using different guest OSes)
- **Communication between VMs**
 - Mechanisms for VMs hosted on the same hardware to be able to communicate securely with each other
- **VM Migrations**
 - Migrating VMs from one hypervisor/hardware to another, almost instantaneously (at runtime)
 - Gives a lot of flexibility and portability

CE2005/CZ2005 Operating Systems
1.25
Part 1 Introduction to Operating Systems

Here we present some of the key functions of a Hypervisor. The basic function is to be able to create/import VMs and manage their execution requirements. Heterogeneity among VMs must be given special considerations.

VMs, although independent, may want to communicate with each other for instance to exchange data/information. This usually happens in designs where one VM is extracting data from sensors or other data sources which is then used by several other VMs. This communication can only happen through the hypervisor and it must facilitate the process.

Finally, VMs act like plug-and-play modules. It should be easy to migrate a VM from one hypervisor+hardware to another (compatible one). This migration may even happen at runtime due to hardware utilization considerations.

Why do we need Virtualization?

- **Allows for more efficient utilization of hardware**
 - Cost-effective hardware deployment & sharing
 - Low latency and agile execution-environment deployments (VM creation and flexibility)
 - Failure mitigation (VM independence and migrations)
- **Key technology that is driving cloud computing**
 - Cloud providers can dynamically and cost-effectively scale hardware allocations based on user requirements
 - Enables the concept of platform as a service – rent hardware & services as and when needed

CE2005/CZ2005 Operating Systems
1.26
Part 1 Introduction to Operating Systems

Here we discuss some of the main advantages of virtualization and hypervisors. They provide efficient hardware utilization and sharing across multiple applications.

They provide a low overhead solution for dynamically creating and managing varied execution environments for different application requirements.

They provide a safe and secure execution environment with some in-built failure mitigation mechanisms. VMs are independent of each other and can be configured to operate in isolation of other VMs. Thus they are protected from the failures in other VMs. Additionally, VM migrations also enable mitigation against hardware failures.

It is a key technology that drives much of the edge and cloud computing frameworks of today. The ability to operate cloud infra. as a service oriented platform (PaaS), is largely thanks to virtualization. Essentially, end-users rent out hardware as and when they need it, and the underlying virtualization framework makes this feasible and efficient (cost, time, etc.). It also supports efficient hardware scalability.

Virtualization Challenges

- **Requires management layer: Hypervisor**
 - Hypervisor is usually two orders of magnitude smaller than general purpose OS
 - Requires more disk space and RAM
 - Must ensure that instructions can be executed on H/W
- **Real-Time CPS require different solutions than server virtualization**
 - Real-time (worst-case timing predictability)
 - Minimal memory footprint & minimal overhead (highly resource constrained)

CE2005/CZ2005 Operating Systems
1.27
Part 1 Introduction to Operating Systems

Here we discuss some of the main challenges in developing a hypervisor. They consume resources, both memory and disk space, as well as computational resources. Hence, this overhead needs to be minimized.

Additionally, they need to ensure compatibility with H/W. Different VMs using different guest OSes could be running on the hypervisor at any point in time. There must be support to ensure that all instructions can be executed on the H/W.

Finally, for real-time CPS, the requirements on timing deadlines and resource utilization are even more stringent; worst-case guarantees on timing are required and the available H/W resources are usually very limited in such applications.

Type-1 Virtualization

Hypervisor interacts directly with hardware

- Also called bare-metal hypervisor
- Highly secure and has low latency
- Popular in industry (KVM, Microsoft hyper-v, VMware esxi, Xen)

```

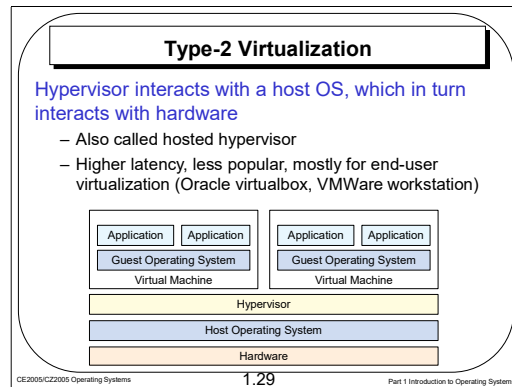
graph TD
    subgraph VMs [Virtual Machines]
        direction TB
        subgraph VM1 [Virtual Machine]
            direction TB
            App1[Application]
            App2[Application]
            OS1[Guest Operating System]
        end
        subgraph VM2 [Virtual Machine]
            direction TB
            App3[Application]
            App4[Application]
            OS2[Guest Operating System]
        end
    end
    VMs --- Hyp[Hypervisor]
    Hyp --- Hardware[Hardware]
            
```

CE2005/CZ2005 Operating Systems
1.28
Part 1 Introduction to Operating Systems

Type-1 virtualization has the hypervisor sitting directly on top of computer hardware. Hence it is also called bare-metal virtualization.

This is usually more efficient (than the other type in next slide), lower latencies, more secure, This is mainly because the hypervisor is directly interacting with the hardware.

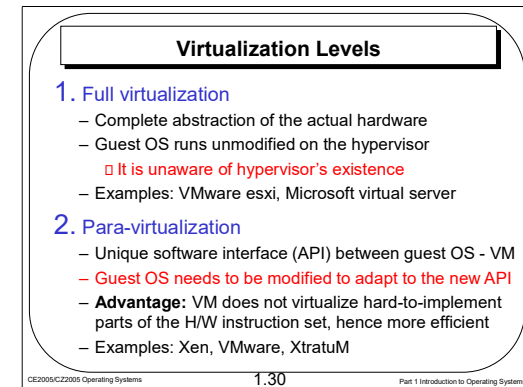
Very popular in the industry. Almost all cloud technologies use this virtualization as the foundational backend. Some examples are provided in this slide.



Type-2 virtualization has the hypervisor sitting on top of a host OS, which in turn interacts with the computer hardware. Hence, it is also called hosted virtualization.

This is usually less efficient (than the other type in previous slide), higher latencies, This is mainly because the hypervisor is indirectly interacting with the hardware through a host OS.

Not that popular in the industry and is mainly used for end-user virtualization problems like the examples shown in this slide.



Orthogonal to virtualization types, we have virtualization level, which denotes the extent to which hardware is virtualized by the hypervisor.

In full virtualization, the hardware is completely abstracted, meaning the guest OS is not aware of the existence of hypervisor. The advantage of this is that any existing OS can be used as a guest OS off-the-shelf.

Full virtualization is slow however, because some H/W instructions are hard to virtualize. An alternate to this is para-virtualization, where the hard-to-virtualize parts of the instruction set are left untouched and handled only by the hypervisor. This means the guest OS must now be modified to interact with the hypervisor, because some services and features are not available at the guest OS level.