Lecture 13: Virtualization
Operating Systems – EDA093/DIT401

Vincenzo Gulisano
vincenzo.gulisano@chalmers.se

UNIVERSITY OF
GOTHENBURG

Reading instructions

- Chapter 7, up to 7.11 (included)

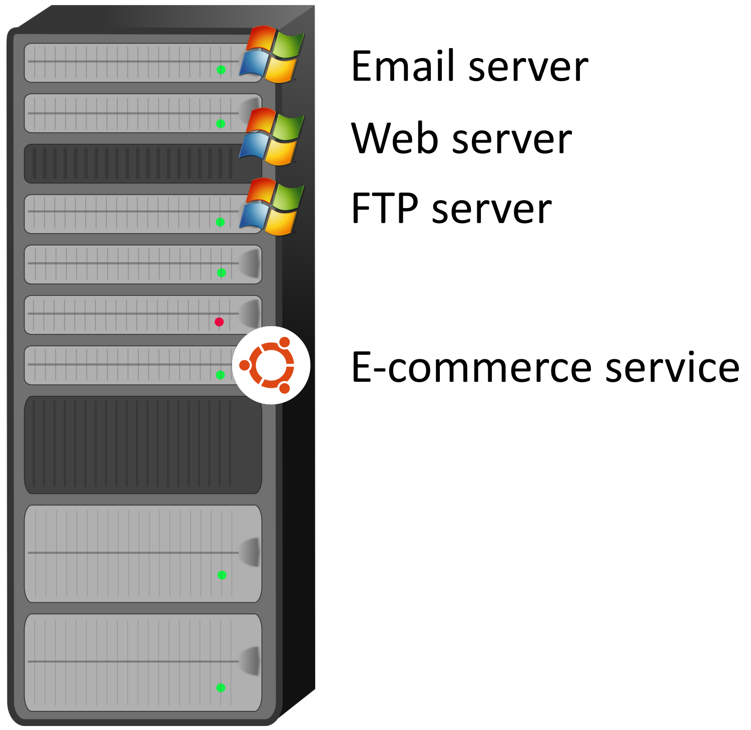
Agenda

- Introduction
- Requirement for virtualization
- Type 1 and type 2 hypervisors
- Techniques for Efficient Virtualization
- Memory virtualization
- I/O virtualization
- Other benefits of virtualization
- Clouds

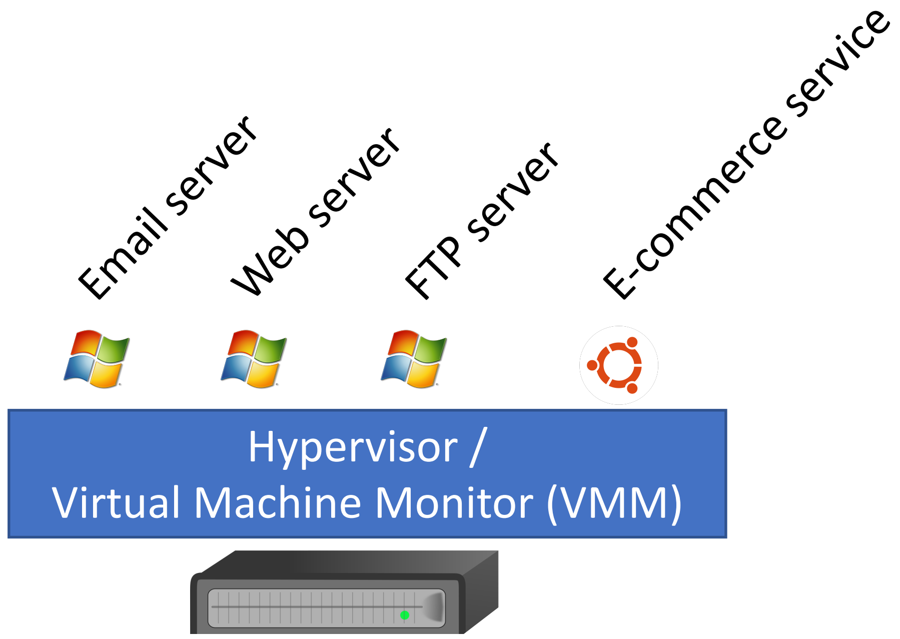
Agenda

- Introduction
- Requirement for virtualization
- Type 1 and type 2 hypervisors
- Techniques for Efficient Virtualization
- Memory virtualization
- I/O virtualization
- Other benefits of virtualization
- Clouds

Without virtualization



With virtualization
(discussed since the 60s…)



	Without virtualization	With virtualization
PROS	• If one machine cannot handle the entire load, it works • Reliability (1 server crash does not imply all services are down) • Better for security (sandboxing) – if an intruder compromises web server, does not have access to mails • Run different OSes • …	• If the failures are at software/OS level, still reliable • Run different OSes • Hypervisor code (which need to be the one you can really trust) is smaller than OS and easier to maintain • Costs less • Cheaper to try out applications/services/… (good in software development, to test with different OSes) and provide **virtual appliances** • Legacy applications • Easier to migrate services/OSes
CONS	• Costs money and time	• Hardware failure can have catastrophic consequences

As you can imagine, the Cloud starts from this idea and takes it to that of using a server located somewhere else, which further reduces costs…

Agenda

Aside from “acting like a physical machine”, what are actually the requirements?

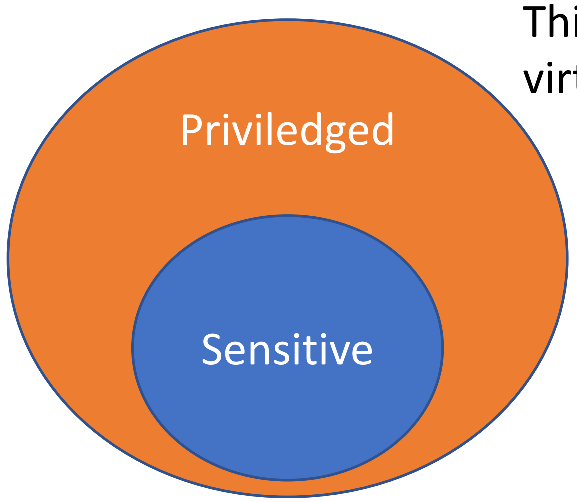
- Safety (hypervisor should have full control of the virtualized resources)
- Fidelity (program running on bare hardware = program running on a VM)
- Efficiency

Why can’t we ”just” use an interpreter?

- Wrap each instruction
  - Execute if safe (e.g., INC instruction)
  - Simulate otherwise (e.g., disable interrupts)
- Theoretically it works, but
  - Safety ✓
  - Fidelity ✓
  - Efficiency ✗
- VMMs should execute most of the code directly…

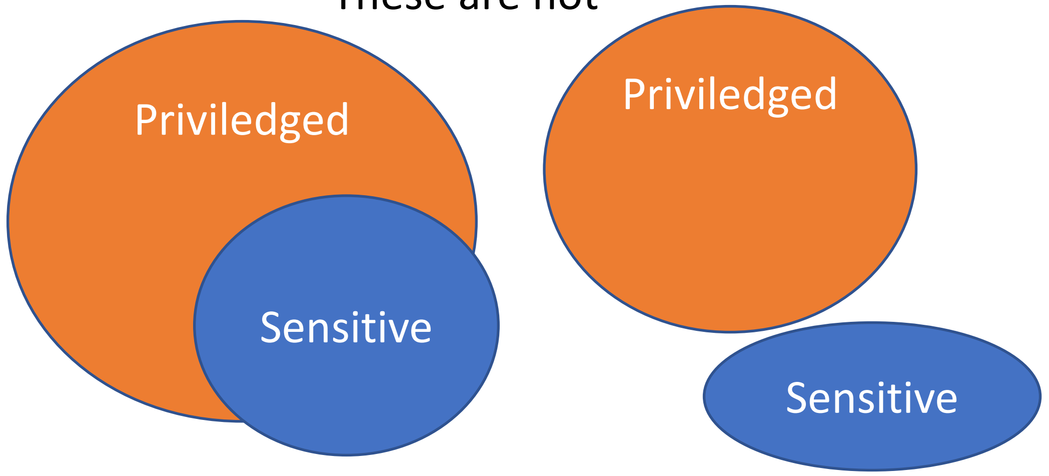
What is required for VMMs to run code directly?

- Sensitive instructions: can be run both in user mode and kernel mode
  - Their behavior might change depending on the case
- Priviledged instructions: will result in an interrupt if not in kernel mode



This machine is virtualizable

These are not



This has been taken into account since 2005

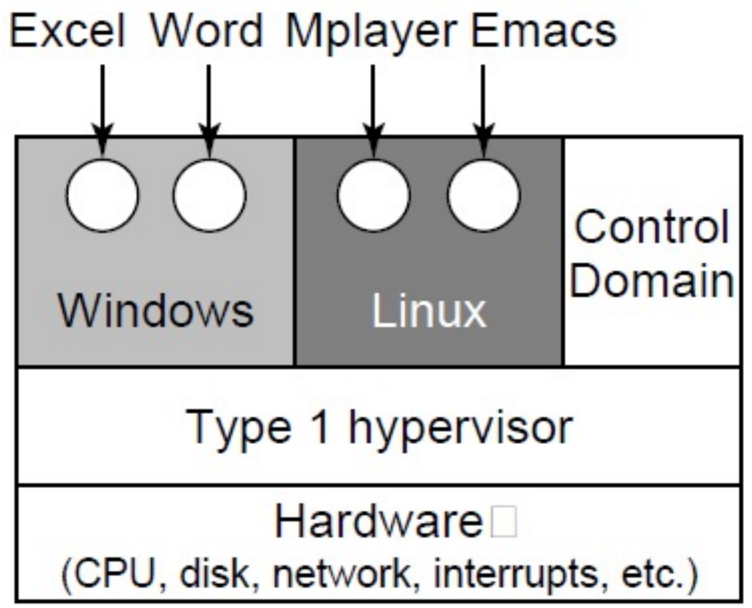
Hardware and OSes collaborating, again…

- Both from Intel and AMD
- VT technology:
  - The guest OS runs directly on the hardware until it generates an interrupt
  - Which is delivered to the hypervisor
- The hypervisor sets a hardware bitmap to specify all interrupts it wants to be alerted about (trap-and-emulate approach)
- The technique used before that was binary translation:
  - Simply put: replace all unsafe I/O with a trap

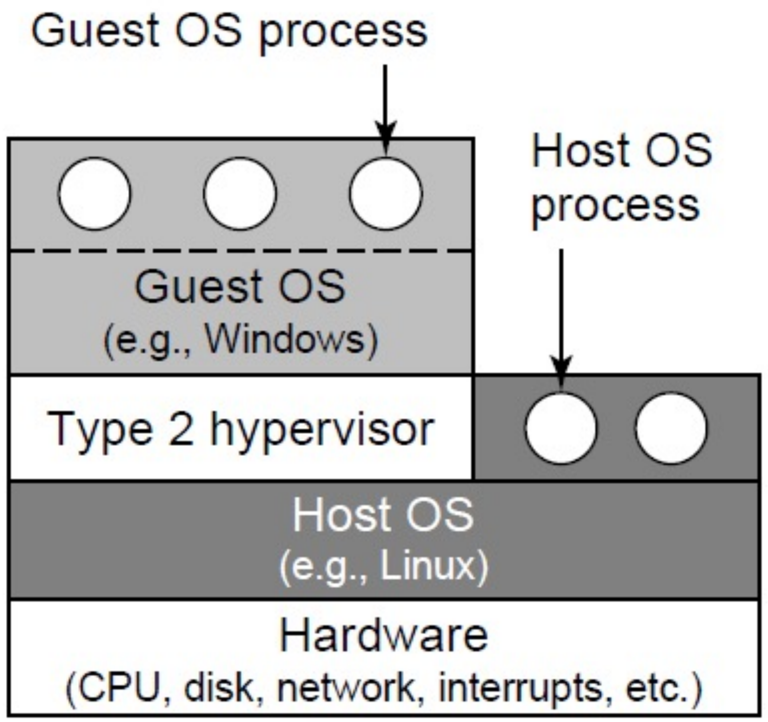
Agenda

- Introduction
- Requirement for virtualization
- Type 1 and type 2 hypervisors
- Techniques for Efficient Virtualization
- Memory virtualization
- I/O virtualization
- Other benefits of virtualization
- Clouds

Type 1 and Type 2 Hypervisors



(a)

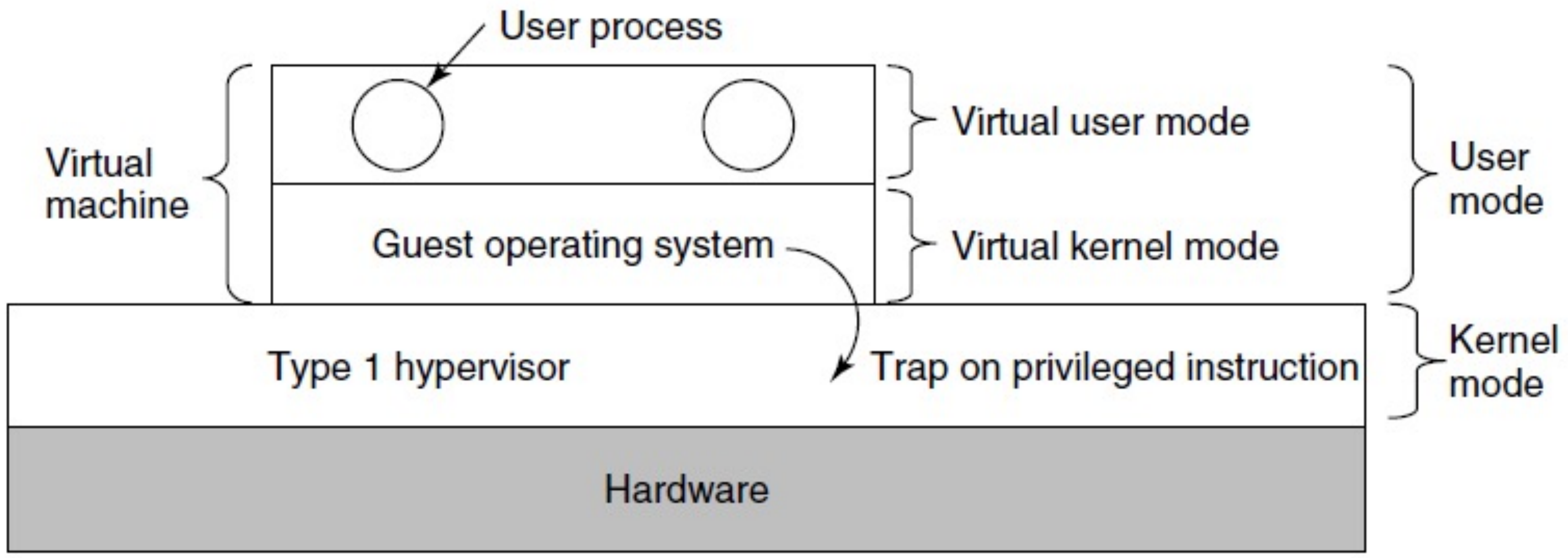


(b)

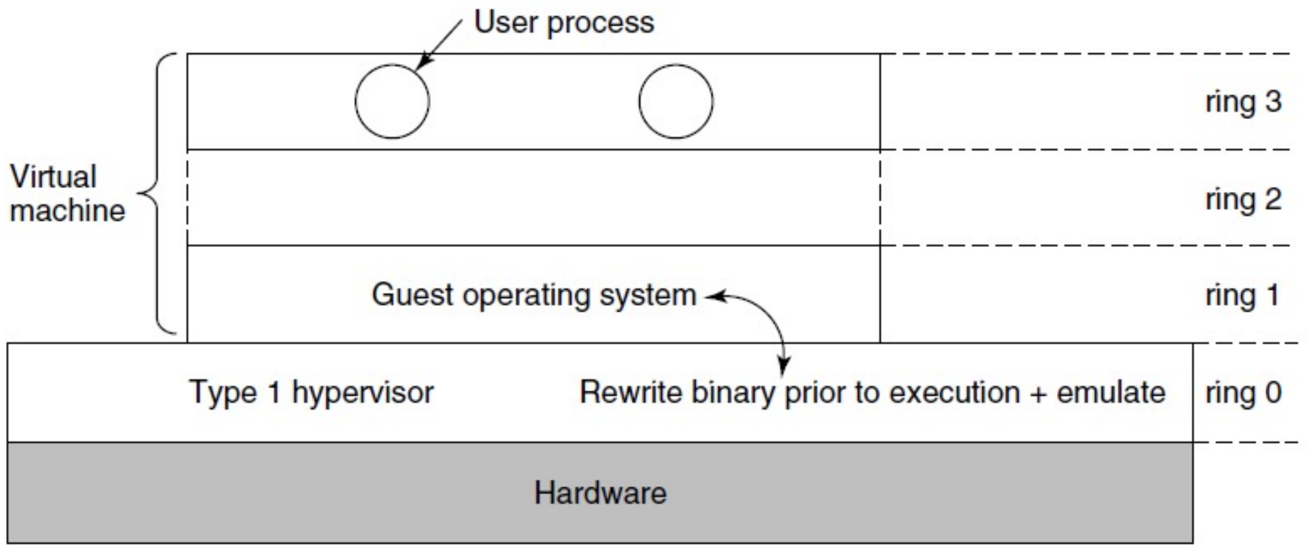
Agenda

- Introduction
- Requirement for virtualization
- Type 1 and type 2 hypervisors
- Techniques for Efficient Virtualization
- Memory virtualization
- I/O virtualization
- Other benefits of virtualization
- Clouds

Hypervisor type 1, with VT

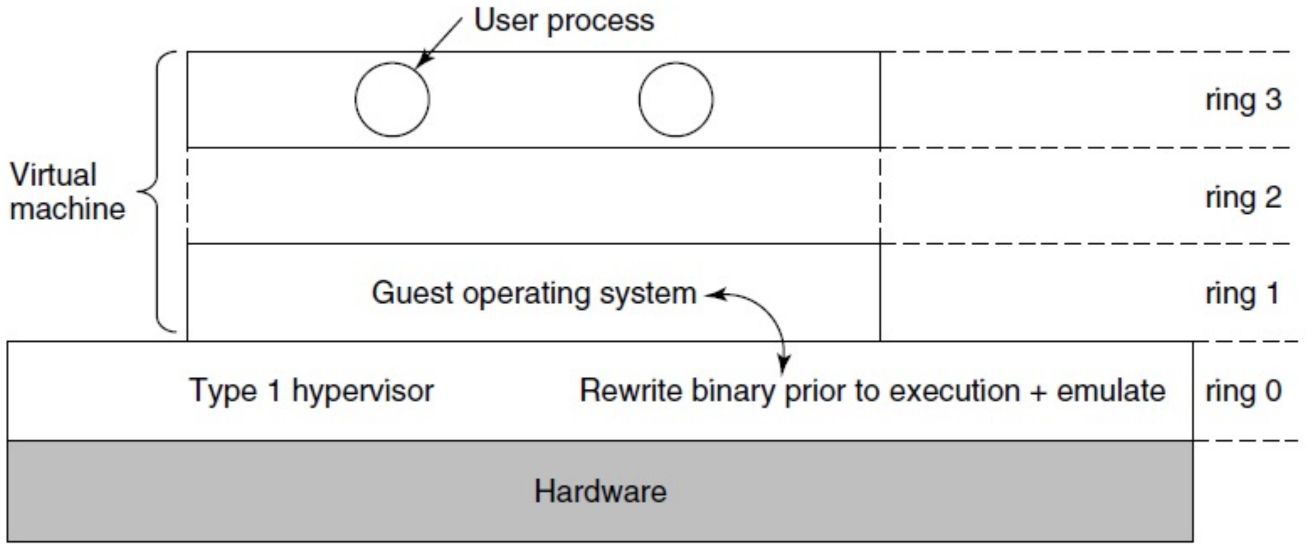


Hypervisor type 1, before/without VT



- Before VT, x86 architectures supported 4 protection modes/rings.
- Without virtualization, OS runs in ring 0, user processes in ring 3
- With virtualization, as shown in figure
- OS privileged instructions will trap, but what about sensitive ones?

Hypervisor type 1, before/without VT



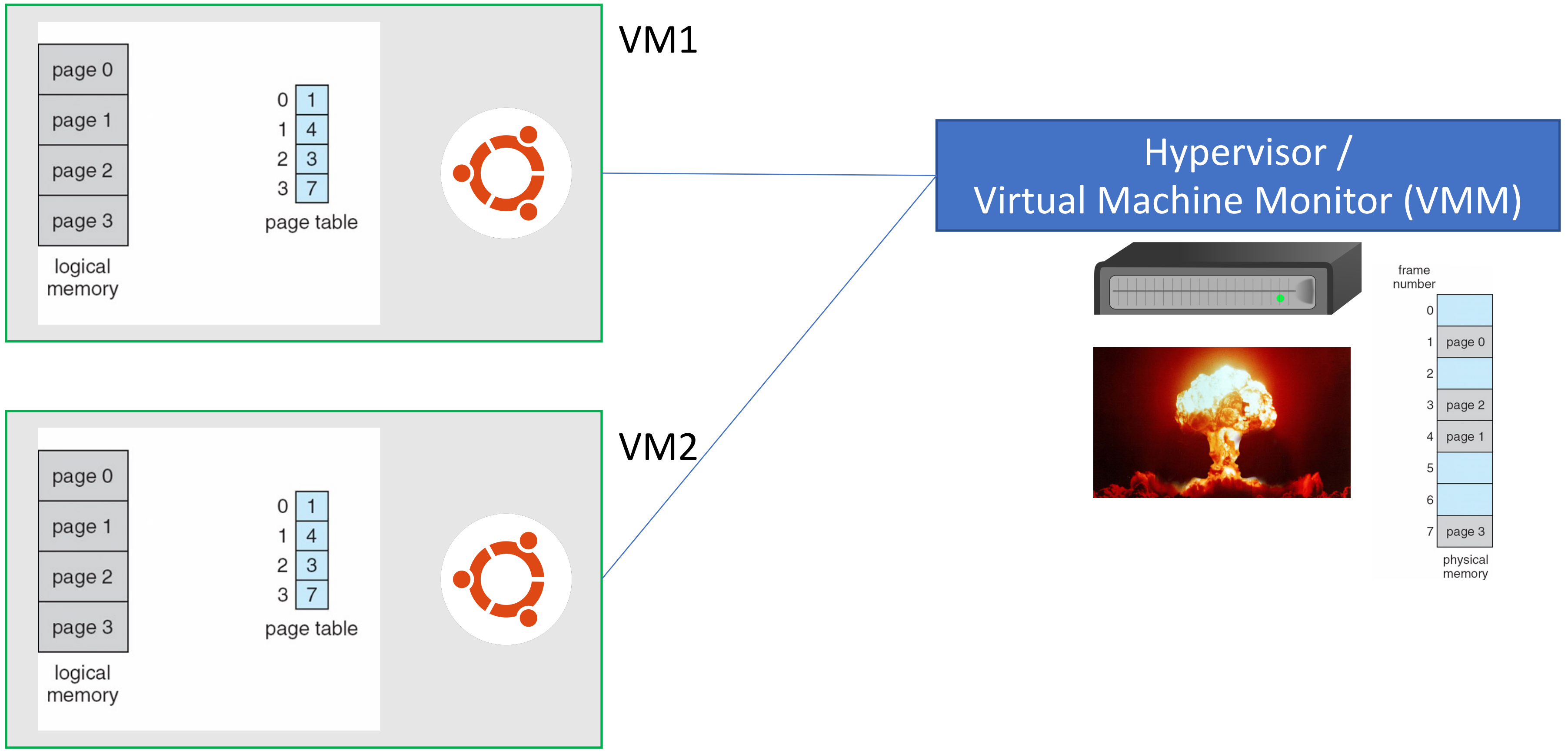
- Sensitive instructions are removed by rewriting the code
- Before executing each **basic block** (a straight sequence of instructions that ends with a branch), the hypervisor scans the block and replaces sensitive instructions

The cost of virtualization

- Traps are expensive
- Because of that, sometimes VT + trap-and-emulate performs worst than binary translation
- In fact, some hypervisor might use binary translation even for privileged instructions
- A small example: Suppose you want to disable interrupts for a while
  - VT: set registers in the hardware
  - Binary translation: rewrite code with an extra if statement

Agenda

Why do we need to virtualize memory too?



Shadow page table

- The hypervisor maintains a **shadow page table**, to translate each VM’s believed “frame” into the real physical frame
- It is complicated, though:
  - Guest OS page tables and the Hypervisor shadow page table must always be in synch,
  - But the OS can change its page table by simply writing to memory…
  - So, how does the Hypervisor detect a change?
    - Option 1: mark as read-only the VM pages that contain the page table
    - Option 2: do nothing and wait for page faults to happen
      - but this requires extra care of mappings that are removed by the guest OS, because those do not generate page-faults

Page faults

- Guest-induced page faults
  - This are detected by the hypervisor
  - But need to be reinjected to the guest OS
    - Example: OS trying to access a page that has been swapped out of RAM
- Hypervisor-induced page faults
  - Used to keep the shadow page table in synch with the guest OS page table

- When a page faults results in the hypervisor gaining control, we use the term **VM exit**

Reclaiming memory

- If the hypervisor “lies” to the guest OSes pretending it has way more frames it has,
- and then needs to reclaim some of them from OS A and give them to OS B,
- how can it do that?

- If it simply takes them away, it needs to update the OS page tables, but, where are they? When can that be done safely?...

Ballooning

- A small module is loaded in each VM (a balloon) as a pseudo device driver connected to the hypervisor
- The hypervisor can inflate is requested space to force the guest OS in releasing pages from other processes and give them to this module
  - So that the hypervisor can then use them…

Agenda

- Introduction
- Requirement for virtualization
- Type 1 and type 2 hypervisors
- Techniques for Efficient Virtualization
- Memory virtualization
- I/O virtualization
- Other benefits of virtualization
- Clouds

I/O virtualization

- The hypervisor probes devices at startup and
- reports them to the guest OSes.
- When the latter try to use them,
- they will interrupt and the hypervisor can intervene
- BUT: each OS cannot really hold the entire device (e.g., a disk)
- Possible solutions
  - Emulate a disk by means of a file managed by the Hypervisor
  - Emulate network by having the Hypervisor actually behaving as a switch

Single Root I/O Virtualization (SR-IOV)

- For performance, a hypervisor-in-the-middle virtualization approach might not always be optimal
- Modern hardware supports virtualization, which allows for devices to be directly controlled by a guest VM (bypassing the hypervisor)
- Devices with SR-IOV usually have
  - Independent memory space, interrupts and DMA streams for each VM
  - Each VM can map in is own memory its corresponding memory areas

Agenda

Other benefits of virtualization - 1

- When a multi-CPU / multi-core machine is used,
- each VM can be assigned a specific subset of cores
- This has an important implication:
  - You could be able to write programs that have a parallelism degree (i.e., number of threads or co-operating processes) that fits the actual number of physical cores the VM you are going to use has access to

Other benefits of virtualization - 2

- Different VMs share memory, and they could potentially be duplicating a lot of (read-only) information
  - Think for instance at having many VMs running the same OS in parallel…
- Deduplication techniques (also referred to as transparent page sharing or content-based page sharing) which are usually used for disks, can be used for memory too!
  - It should not come as a surprise, the physical / logical separation behind VM that is beneficial for running several processes at the same time efficiently can be also beneficial to run several OSes at the same time
  - At the end of the day, the OS runs a collection of processes too

Agenda

- Introduction
- Requirement for virtualization
- Type 1 and type 2 hypervisors
- Techniques for Efficient Virtualization
- Memory virtualization
- I/O virtualization
- Other benefits of virtualization
- Clouds

What exactly is the cloud?

- Public vs private…
- Access to physical hardware, access to virtual OSes, access to specific software
- Big machines / small machines
- Desired characteristics
  - On-demand self-service
  - Broad network access (available via network with standard mechanisms)
  - Resource pooling
  - Elasticity
  - Measured service (also in connection to $)

Types of services

- IAAS (Infrastructure as a Service)
    - Direct access to a specific VM, the user does what (s)he wants
- PASS (Platform as a Service)
    - Specific environment, with given OS, DataBase, WebServer, …
- SAAS (Software as a Service)
    - Offers specific software

Virtual Machine Migration

- What is the server running a bunch of VMs needs maintenance?
- Intuition: move the VM to a different server…
- Simple approach / but not liked by customers who pay
  - Turn off the VM, move the data to a different server, turn on the VM
  - Long downtime… also, turn off when? Ask the user?
- Live migration
  - Keep copy pages in the back, making the transition from one server to another not noticeable (seamless live migration)
  - As you probably imagine, this is everything but simple…

Checkpointing/Snapshotting

- An advantage of VMs is that they can be paused and copied
- If something bad happens, you can then revert the VM to the previous save
- Incremental snapshotting with mechanisms like copy on write can then allow for lightweight checkpointing
  - Which can be also used in combination with migration