Compute services

Boris TEABE
boris.teabe@inp-toulouse.fr

Goals

1. **Bare Metals**
2. Virtualization
3. Containers
4. Unikernels/MicroVM

Compute Services (IaaS)

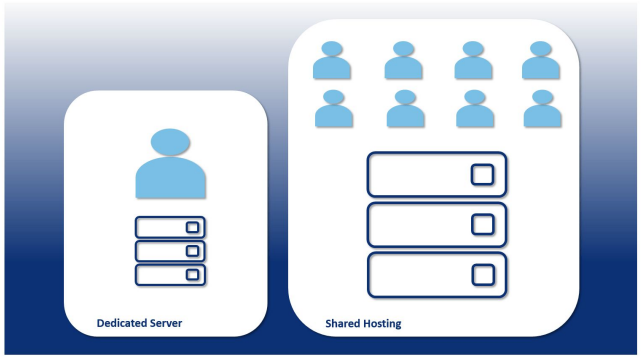
- **An infrastructure is proposed to the User**
  - Number of CPUs
  - Amount of Memory
  - Network Bandwidth
  - Storage capacity

Compute Services (IaaS)

- Several way to provide an infrastructure
  - **BareMetal**
    - Fully dedicated server
  - **Virtualization**
    - Abstraction of a hardware and software isolation
  - **Containers**
    - Lightweight virtualization, less isolated
  - **Unikernel**
    - The future, Strong isolation with good performance

Compute Services (IaaS)

- **Bare Metal (Bare Metal As A Service)**
  - Fully dedicated servers allocated to the clients
  - The client has access fully to a server
  - Can act on the hardware, e.g access hardware features
  - Useful for sensitive workload with lot of computation
  - But, very expensive
  - Provided by AWS, GCP etc.

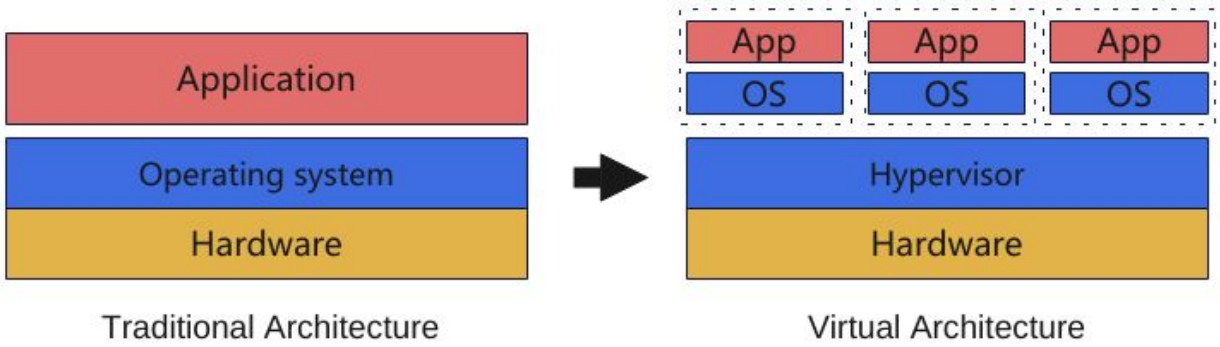


Goals

1. Bare Metals
2. **Virtualization**
3. Containers
4. Unikernels/MicroVM

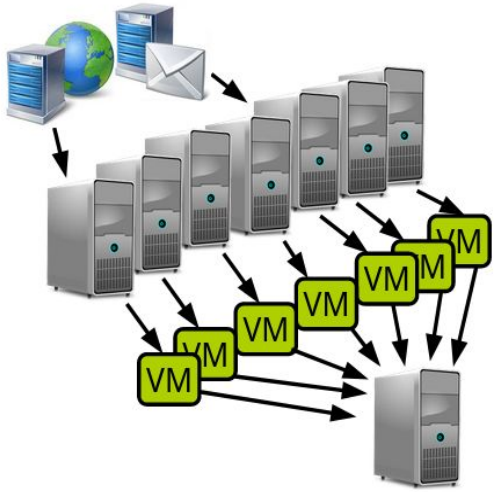
Compute Services (IaaS)

- **Virtualization**
  - Set of techniques, hardware and software, which allow managing simultaneously on a single machine several operating systems (called virtual machines (VMs)). Examples: Xen, VMWare, KVM, HyperV, etc;



Virtualization: motivations

- **Consolidation**
  - Creating X virtual machines from X physical ones and running them on Y physical hosts
    - **With Y < X**
    - Historical motivation for developing virtualization technologies
    - Gives (most of) the benefits of multi-computer systems without the $/management costs:
      - Software dependencies
      - Reliability
      - Security



Virtualization: motivations

- **Use Cases: Software Development**
  - **Flexible OS diversity: different OS on the same machine**
    - e.g VirtualBox with linux for kernel development
  - **Rapid and cost-efficient provisioning**
    - Way faster that Physical machine, we will see during lab works
  - **VMs are self-contained**
    - Practical way to “pack” an application with all its software dependencies
      - Model and version of the OS, libraries, etc.
      - Useful for development, automated testing, and even deployment

Virtualization: motivations

- **Migration and Checkpoint/Restart**
  - VMs are self-contained and can be migrated between hosts
    - **Live migration**: moving VMs from one host to another
    - Freeing resources for maintenance or when a fault is expected
    - Increased performance
    - Distributed resources scheduling: load balancing, consolidation for power savings, etc.
  - Checkpoint/restart for long-running jobs
    - Dump the VM state to disk in order to be able to resume it later
    - Both techniques are straightforward for VM as opposed to processes

Virtualization: motivations

- **Use Cases: Cloud computing**
  - Virtualization central technology in cloud computing
    - Allows the providers to securely share their computing infrastructure between clients (tenants)
    - Offloading local tasks to remote computing resources
      - Rent a VM to put a web server (IaaS)
      - Offload mail server online to Gmail/Outlook (SaaS)
      - To save on management, infrastructure, development maintenance costs

Virtualization: motivations

- **Security**
  - Virtualization provides **very strong isolation** between guests
    - **Sandboxing**
      - Cloud, virus/malware analysis, honeypots, process/task level isolation through virtualization
    - **VM introspection**
      - Analysis of the guest behavior from a privileged level higher than the OS's Guest OS cannot be trusted
        - E.g. Cloudvisor-D

Virtualization: virtual machines

- A complete compute environment with its own isolated processing capabilities, memory and communication channels.

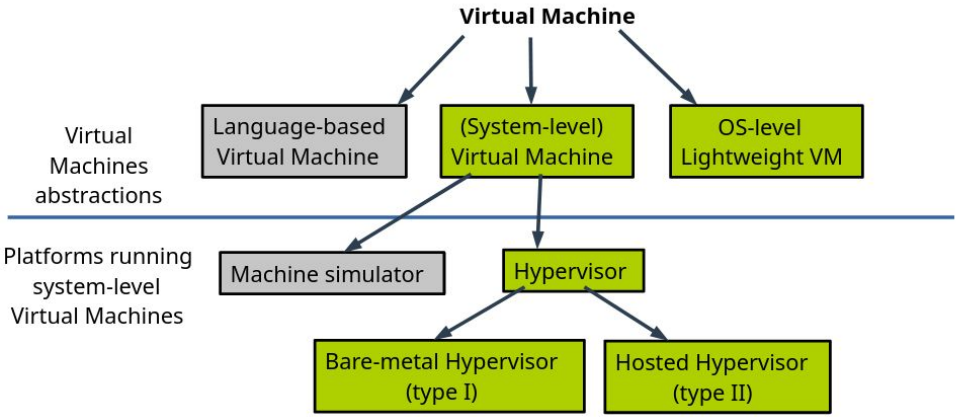
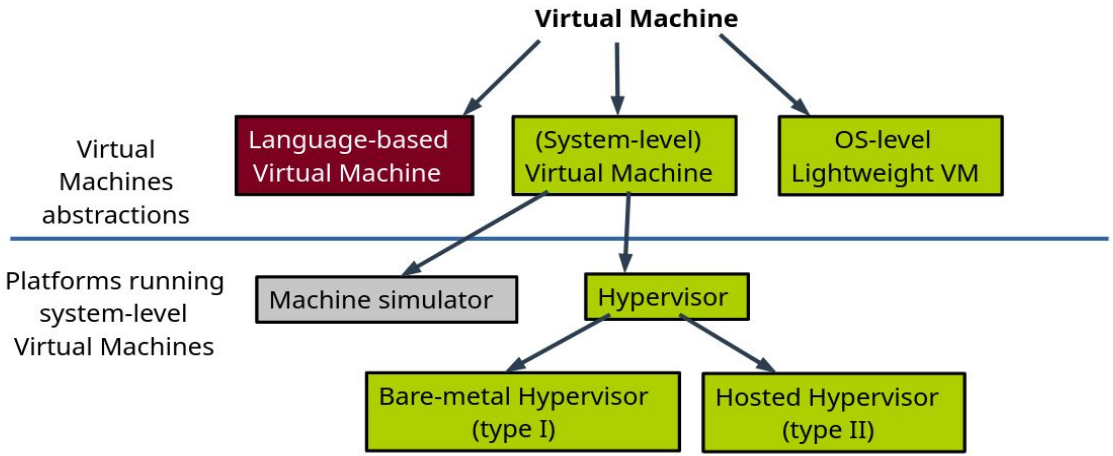


Figure from https://olivierpierre.github.io/virt-101/

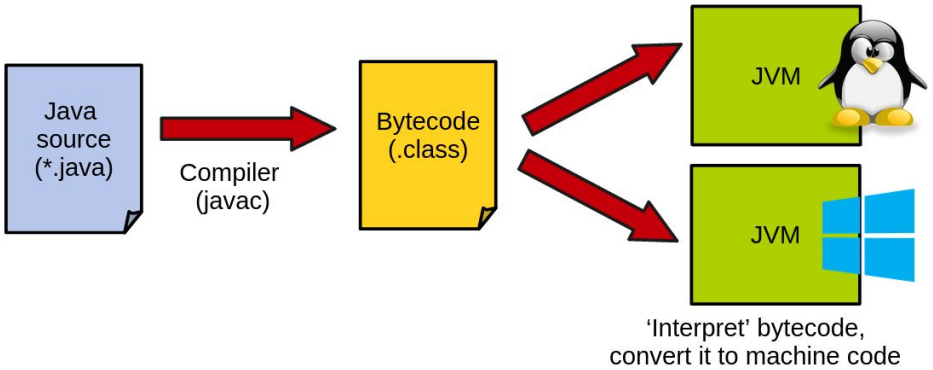
Virtualization: virtual machines



From https://olivierpierre.github.io/virt-101/

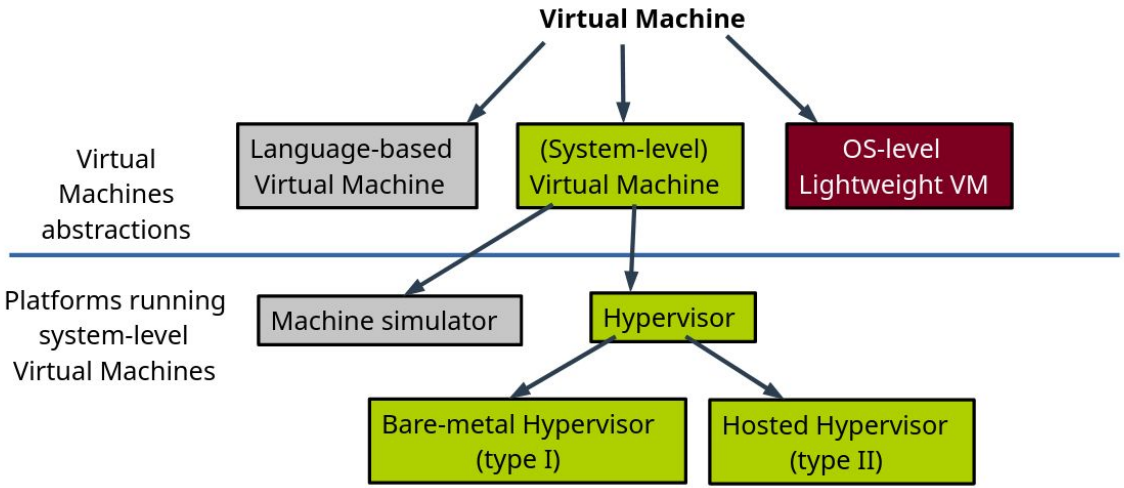
Virtualization: virtual machines

- Java Virtual Machine, JavaScript engines, Microsoft Common Language Runtime
  - Designed to run single applications



From https://olivierpierre.github.io/virt-101/

Virtualization: virtual machines

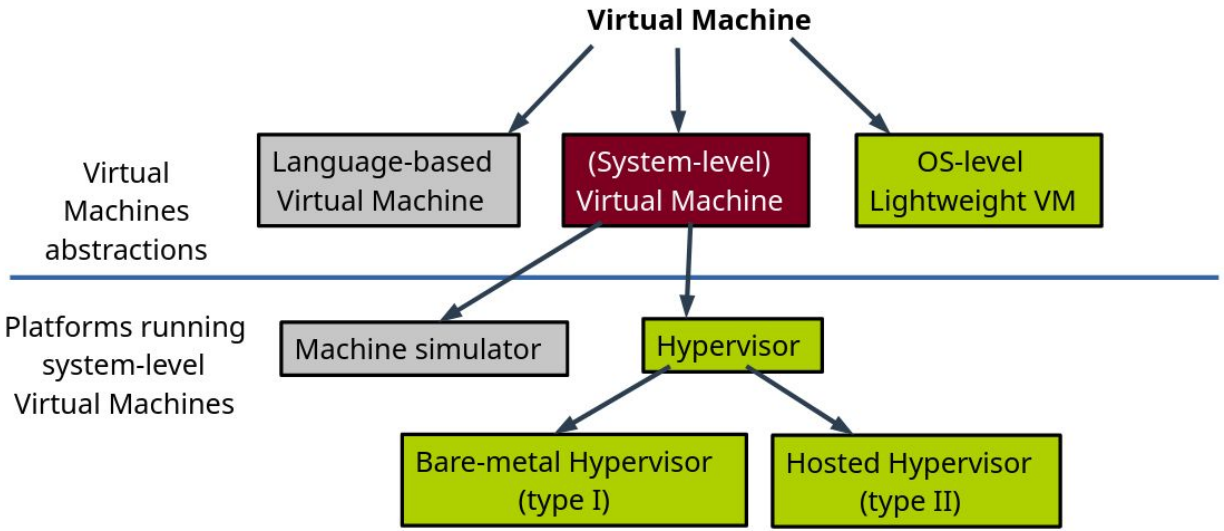


From https://olivierpierre.github.io/virt-101/

Virtualization: virtual machines

- **OS-level Lightweight VMs (discussed in detail later)**
  - Sandboxing: Isolation of native code running directly on the CPU though hardware/software mechanisms. Through OS mechanisms offering more isolation guarantees than regular process-based isolation
  - No attempt is mode to virtualize the hardware: Isolation enforced by the OS/framework level (breaks backward compatibility in some cases)
    - Cannot run unmodified OS
  - E.g. Containers, Google Native Client, LibOSes

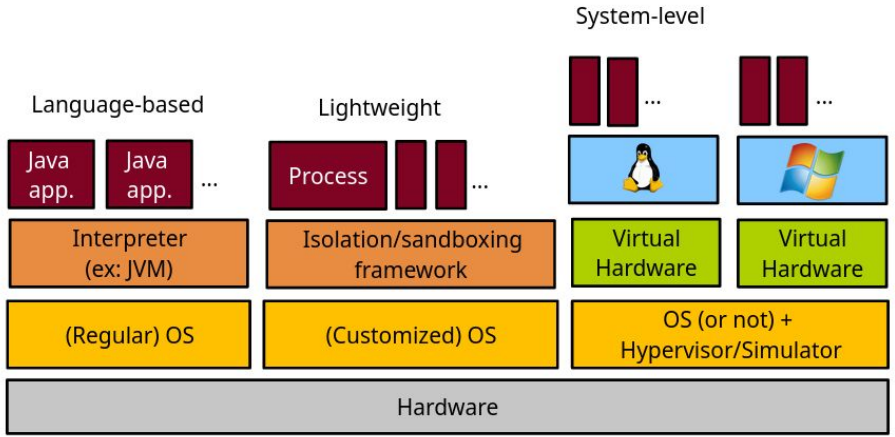
Virtualization: virtual machines



From https://olivierpierre.github.io/virt-101/

Virtualization: virtual machines

- **System-level Virtual Machines**
  - Creates a model of the hardware for a (mostly) unmodified operating system to run on top of it
    - Each VM running on the computer has its own copy of the virtualized hardware



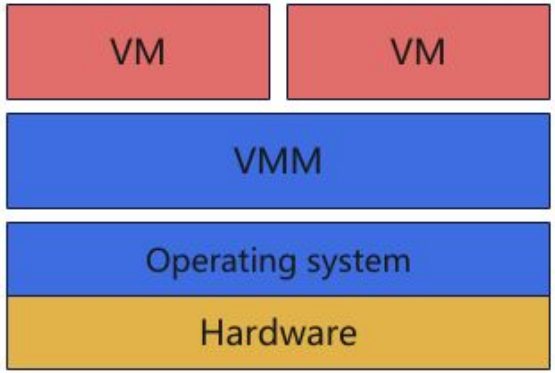
Virtualization

- **System-level Virtual Machines**
  - Full virtualization
  - Para-virtualization
  - Hardware assisted virtualization

Virtualization

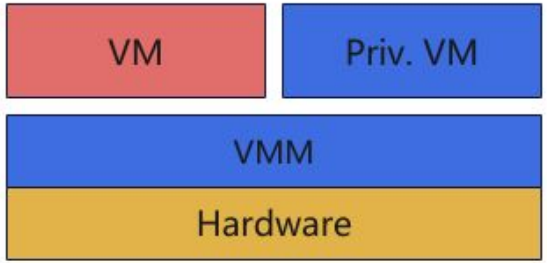
- **Full virtualization**:
  - An OS executes applications at user level. The VMM is one such application.
    Every instruction from VM is emulated by the VMM. The OS executed on a VM is not modified and can be of any type (Linux, Windows, etc.). Ex: virtualBox

- Concretely:
  - It’s hardware simulation (slow)
  - Supports several OS types



Virtualization

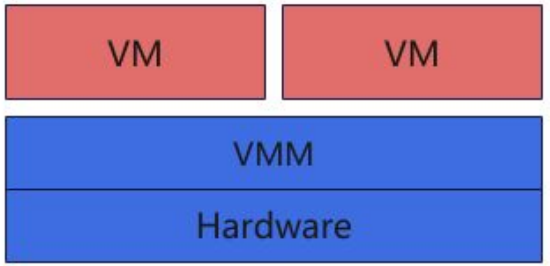
- **Para-virtualization (PV):**
  - The VMM replaces the host OS and behaves as a proxy for accessing the hardware. The host OS is considered as a VM (privileged) and it is used by the VMM to perform particular tasks. Constraint: VMs' OSes have to be modified (to invoke the VMM for privileged operations). Ex: Xen, VMware, etc.



- Concretely:
  - A mix between simulation and native. Several OS types
  - Each VM is allocated hardware resources and the VMM controls access to hardware
  - Hypercalls (a modified OS invokes the VMM)

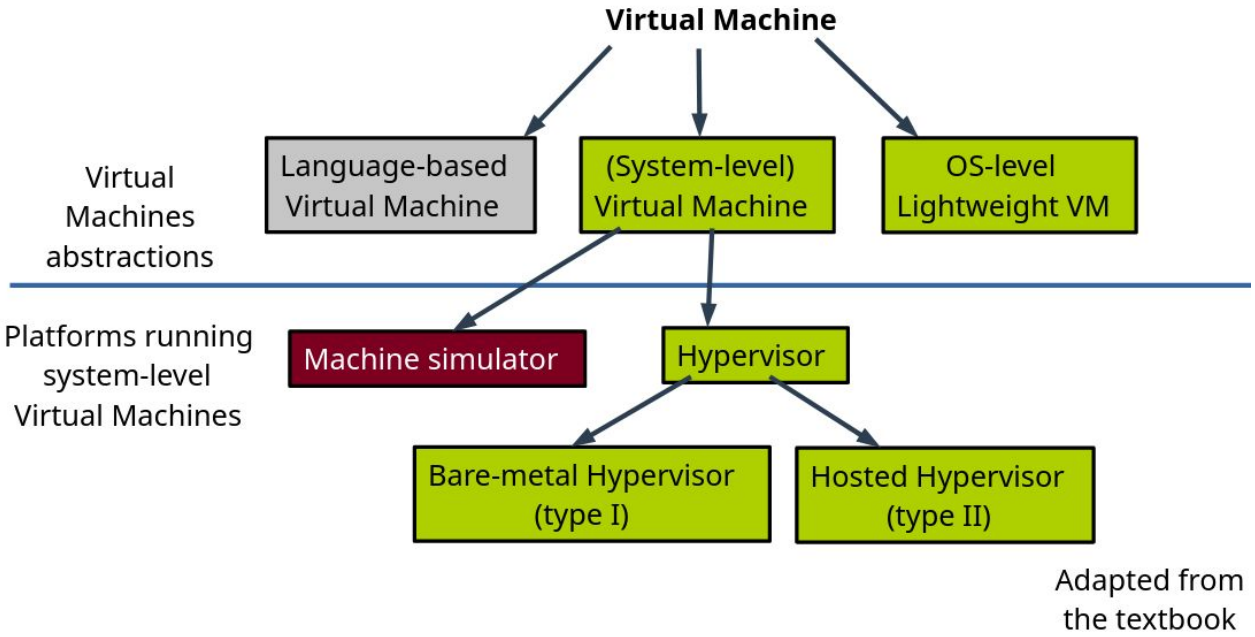
Virtualization

- **Hardware assisted virtualization (HVM):**
  - The hardware is aware of virtualization. VMs' OSes don't have to be modified. Ex: Xen, KVM, VMWare etc.



- Concretely:
  - Native, but without OS modification
  - The hardware is able to manage several VMs (MMU, network device …)
  - Sometimes, para-virtualization is faster or more flexible.

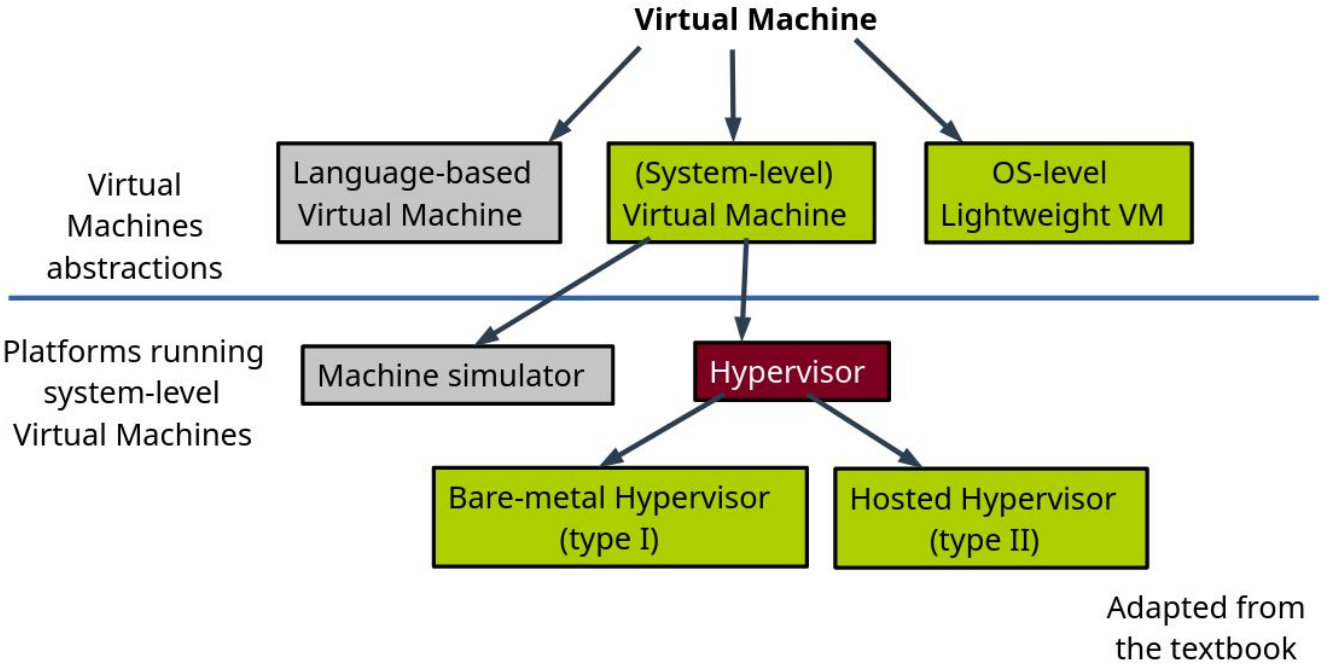
Virtualization



Virtualization: Machine simulation

- **Full virtualization**
- Cross ISA emulation: for usage as substitute
  - Compatible with legacy applications
    - e.g: Qemu
- Architecture emulation
  - Help simulate a hardware architecture for study purpose
    - e.g: Android systems
- Each guest instruction is interpreted in software

Virtualization



Adapted from the textbook

Virtualization: hypervisor

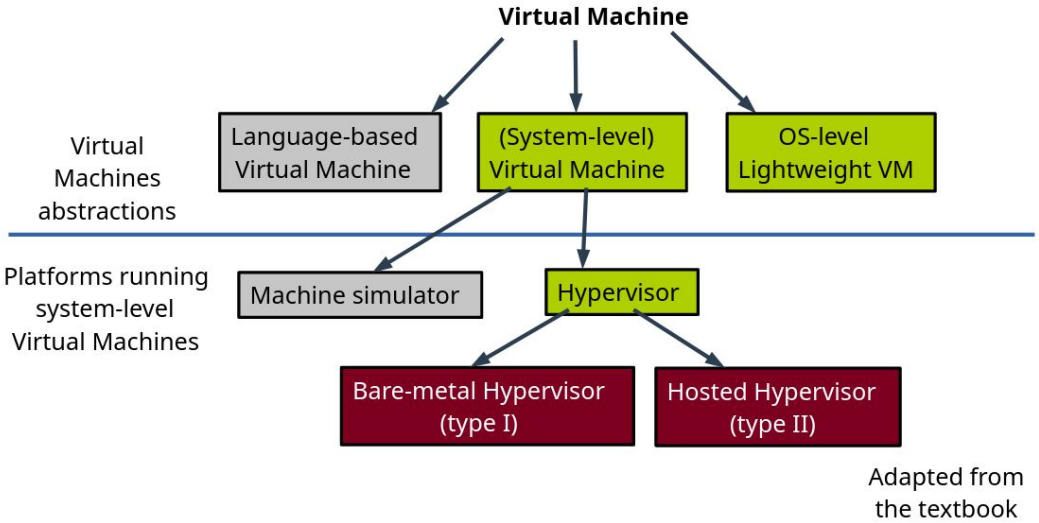
- **Hypervisor or Virtual machine monitor (VMM)**
  - Relies on direct execution for performance reasons (close to native)
  - VM code executes directly on the physical CPU, at a lower privilege level than the hypervisor
  - Privileged instructions trap to the hypervisor
    - Switch to the hypervisor which determines what to do with that instruction: trap-and-emulate model
    - Then back to the VM execution
  - E.g Xen, Linux KVM, VMware ESXi, MS Hyper-V etc.

Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Multiplexes physical resources between VMs-Run VMs while minimizing virtualization overheads
  - Tries to get as close as possible to native performance
  - Ensure isolation between VMs and between VMs and the hypervisor
    - Isolates physical resources: for example memory/address spaces
    - Isolate performance

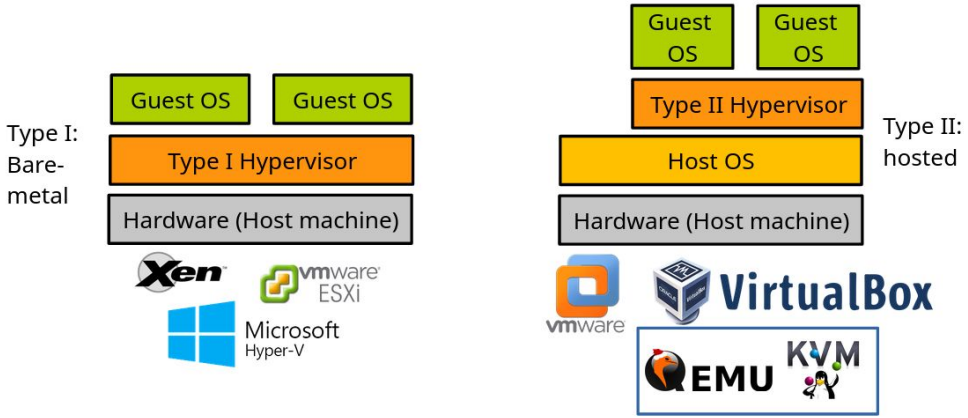
Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**



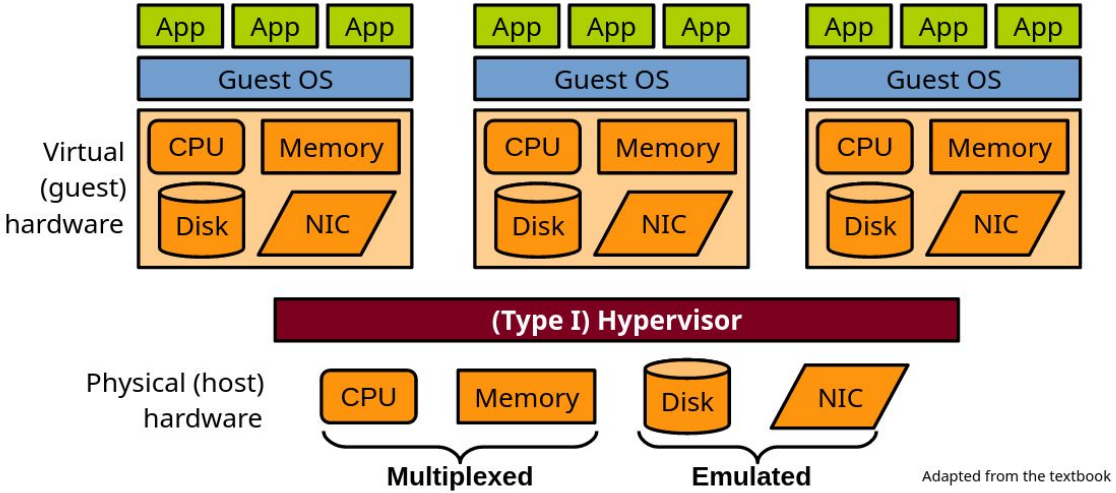
Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Resources allocation and scheduling
    - Type I: done by the hypervisor
    - Type II: more involvement from the Host OS



Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Simplified Hypervisor Sketch

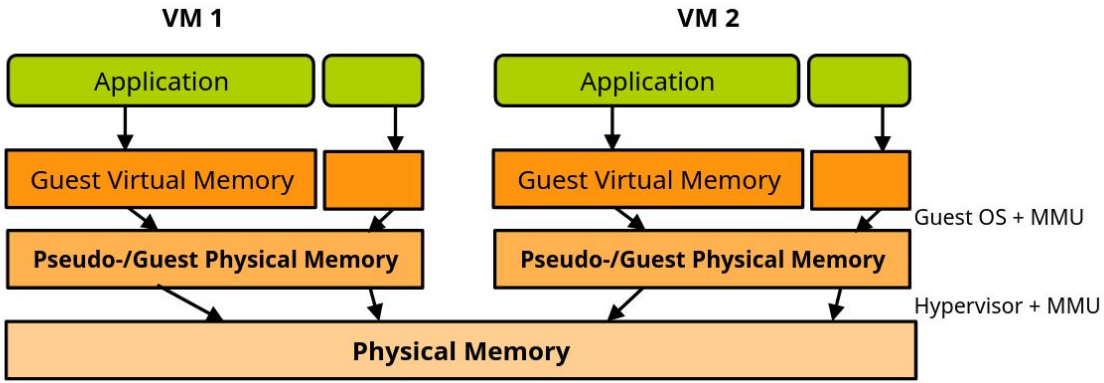


Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Multiplexing CPU and Memory
    - Virtual CPU runs with reduced privileges, cannot execute privileged instructions (that could allow to escape isolation)
    - Traps to the hypervisor on such instructions (virtualization overhead)
    - Run the hypervisor in kernel mode and the VM in user mode.

Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Multiplexing CPU and Memory
    - Another level of translation added, taken care of by the hypervisor: (Guest) virtual memory -> (Guest) pseudo-physical memory -> (host) physical memory.



Virtualization: hypervisor

- **Hypervisor or Virtual machine monitor (VMM)**
  - Emulating I/Os
    - I/O mostly emulated for compatibility reasons
    - I/O devices have well defined interfaces, for example: send a set of network packets, read 128K from disk from secto X, etc.
    - The hypervisor emulate simple virtual device (disk/NIC) that can be accessed with commonly implemented drivers (ex IDE/SCSI): **front-ent**
    - Hypervisor redirects I/O to actual devices or other abstraction, ex: disk or file: **back-end**.