Introduction to Systems Virtualization

Florent Glück - florent.gluck@hesge.ch

February 17, 2025

ISC - HEPIA

Thank you note

Thanks to Prof. Ada Gavrilovska at Georgia Institute of Technology for allowing me to use some of her “Introduction to Operating Systems” course’s content

Introduction to virtualization

What is virtualization?

- Process of creating a virtual representation of something through **abstractions**, such as hardware platforms, storage devices, or network resources
- **Not new**, originated in the 60s:
  - platform virtualization invented to logically divide hardware resources between different operating systems (OS)
  - language-based virtualization invented to give machine independence to a programming language (machine code)

What is a virtual machine?

Two types of virtual machines (VM) in computing:

- **System VM**:
  - an efficient, isolated, duplicate of a real computer machine
  - provide the functionalities required to execute an entire OS
- **Process VM**:
  - designed to execute a program independently of the hardware platform and OS

Types of virtualization

Common types of virtualization (non-exhaustive):

Types of virtualization

Common types of virtualization (non-exhaustive):

(1) Language-based virtualization

Types of virtualization

Common types of virtualization (non-exhaustive):

(1) Language-based virtualization

(2) Storage virtualization

Types of virtualization

Common types of virtualization (non-exhaustive):

(1) Language-based virtualization

(2) Storage virtualization

(3) Network virtualization

Types of virtualization

Common types of virtualization (non-exhaustive):

(1) Language-based virtualization

(2) Storage virtualization

(3) Network virtualization

(4) Platform virtualization

Types of virtualization

Common types of virtualization (non-exhaustive):

(1) Language-based virtualization

(2) Storage virtualization

(3) Network virtualization

(4) Platform virtualization

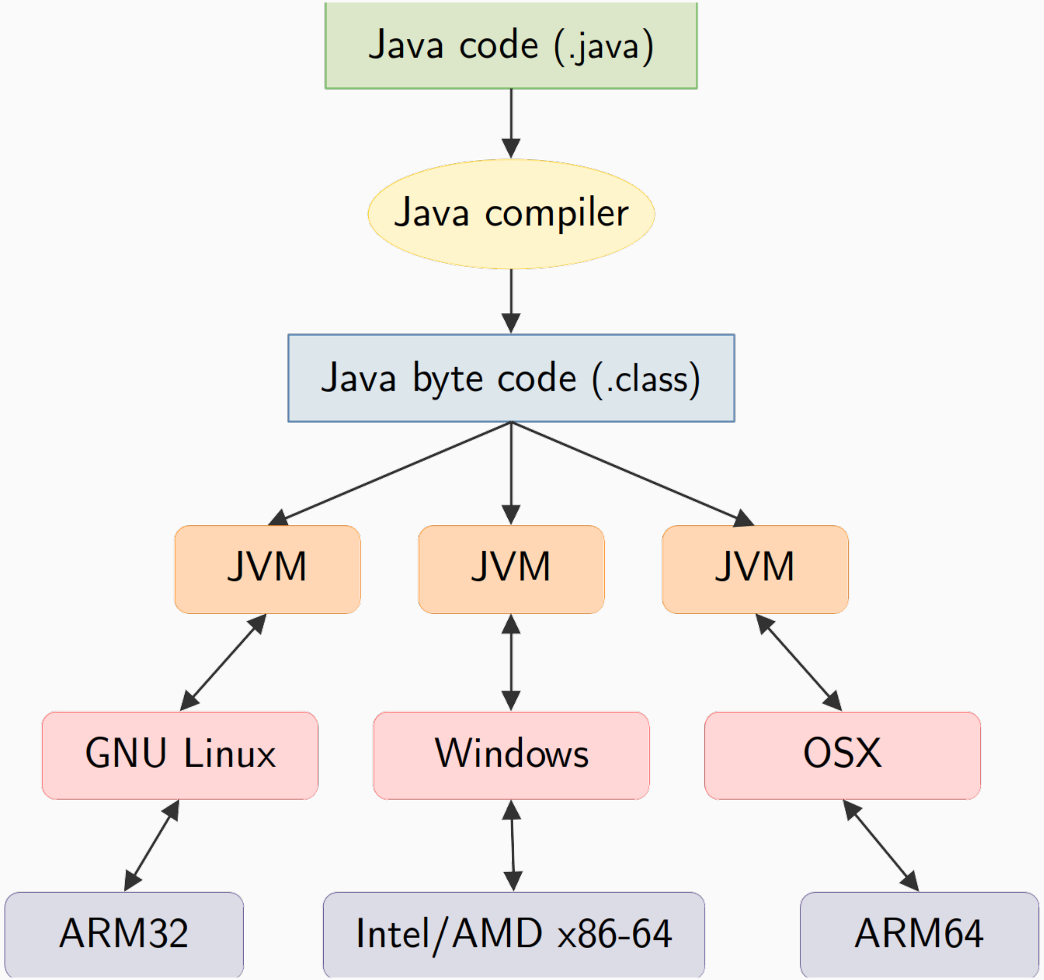
(5) OS virtualization

(1) Language-based virtualization (1/2)

- Also called *Process VM*, *Application VM*, or *Managed Runtime Environment*
- Language compiled into machine instructions targeting a **platform that does not physically exists**
- **Abstracts** the virtual platform from the physical one
  - virtual platform exposed through a **managed runtime**
- VM created when process is started and destroyed when it terminates
- Originated in the late 60s with the O-code VM, invented to give machine independence to the BCPL language[1]

[1] BCPL the ancestor of B, which is the ancestor of C

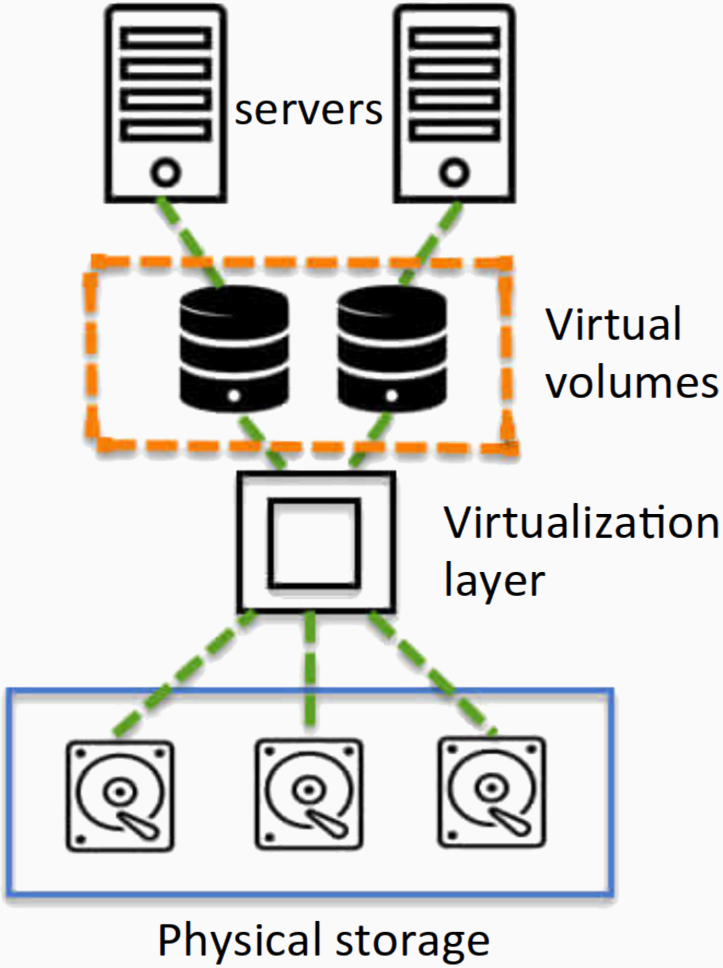
(1) Language-based virtualization (2/2)



- **Benefit**: **architecture and OS independ-ence** → provides application portability (binary) between different platforms
- **Downside**: requires a VM **for each** platform to support
- Examples: java, python, js, lua, .net, smalltalk, etc.

(2) Storage virtualization

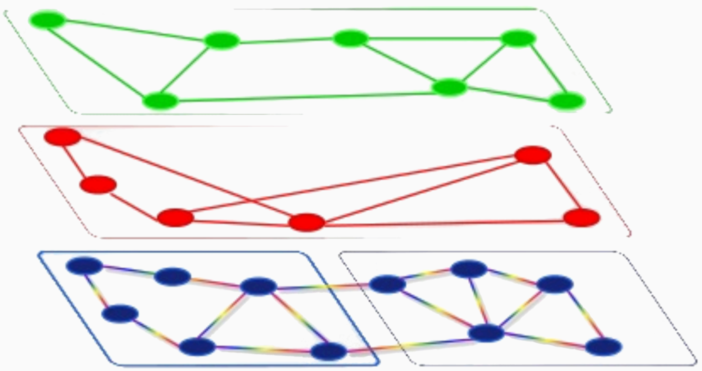
- **Abstracts** the storage-management software from the underlying hardware infrastructure
- **Benefits**:
  - flexibility
  - scalability
  - fast provisioning
- **Downside**:
  - added complexity



(3) Network virtualization

- Network virtualization **abstracts** physical networking resources so that the same set of physical resources can be shared by multiple tenants in a secure and isolated manner

- Examples:
  - software-defined network (SDN)
  - virtual LAN (VLAN)



(4) Platform virtualization

- Originated in the 60s at IBM when computing was based on few large mainframe computers shared through terminals by many users
- Each OS runs on top of an **abstraction layer** that exposes the underlying hardware in **virtual form**
- Also called "hardware virtualization"

- First virtual machine OS
- Could run 4 OS on top of the same physical hardware

(5) OS virtualization

- **Abstraction layer** that enables the OS kernel to create/manage multiple isolated user-space OS instances called containers
- Also called “containerization”
- Provides **isolation** (also called *sandboxing*)
- **Benefits**:
  - security
  - flexibility

Why virtualize?

Physical infrastucture limitations

On a physical system, unfortunate **coupling** between hardware resources and OS:

- Requires **static**, up-front **provisioning** of machine resources
- Cumbersome to **adjust hardware** resources to system needs $\longrightarrow$ requires physical access/changes
- Hard to run **multiple OS** on the same machine
- Difficult to **transfer software setups** to another machine, unless identical or nearly identical hardware

OS limitations

Lack of true **isolation** between multiple applications:

- OS "**leak**" information between processes through filesystem and other channels
- Multiple applications may require **specific and conflicting software** packages to run
- Certain applications may have very specific OS configuration and tuning requirements
- Software vendors *may not provide support* if their application runs alongside **anything** else!

Why virtualize?

- The general concept of virtualization is to provide an **abstraction layer** on top of the hardware infrastructure
- This usually leads to the following **benefits**:
  - flexibility
  - scalability
  - security

Benefits of platform virtualization (1/2)

- **Security**: bugs and faults **isolation**
- **Reliability & availability**
  - OS + apps **decoupled** from physical hardware $\longrightarrow$ **live migration** of VMs from one physical machine to another
  - increase services availability, greater reliability
- **Cost**: ability to run **multiple VMs** on a single host $\longrightarrow$ fewer machines required since used more efficiently
- **Functionality**: ability to run a native app for a **different OS**

Benefits of platform virtualization (2/2)

- **Manageability & flexibility**:
  - easy and fast **provisioning** of specific resources to VM (CPU, RAM, disk space, etc.)
  - can easily replicate an entire machine image in order to duplicate it or move it to a different host
- **Development**: kernel development without affecting host machine, developing for different architectures
- **Support**: legacy OS & applications that cannot run on the host OS

Simulation vs emulation
(in the context of hardware)

Simulation

- A **simulator** is a software that **accurately simulates the behavior** of a given hardware
- Models the full underlying state of the hardware
- Simulation is **very accurate, but very slow**
- Typically used to:
  - develop software for a particular or expensive type of hardware
  - analyze, test and validate printed circuit boards (PCBs)
  - design and optimize circuits
- Examples: Logisim, Altium Designer, Wind River Simics, etc.

Emulation

- A **emulator** is a software that **approximately simulates the behavior** of a given hardware
- Does not model the full underlying state of the hardware
- Emulation is **not always accurate**, but **good enough**
- Many **shortcuts** are taken to achieve **better performance**:
  - code designed to run correctly on real hardware executes "pretty well"
  - code not designed to run correctly on real hardware exhibits wildly divergent behavior

Full-system simulation and emulation

- A **full-system** simulator or emulator provides an environment for running a full unmodified software stack including everything from the firmware and bootloader to the OS and user applications
- In full-system emulation:
  - typically CPU & memory subsystems are emulated, but buses are not
  - just enough of the system hardware components are emulated to create an accurate "user experience"
- In full-system simulation, everything is simulated

Application emulation

- Full-system emulation emulates the **whole system**, including hardware
- Application emulation **only emulates** the programming interfaces (API) used by an application compiled for a given OS, so it can run on another OS with different programming interfaces

Resources

- *Operating Systems: Three Easy Pieces* ; Remzi H. and Andrea C. Arpaci-Dusseau ; Arpaci-Dusseau Books (free PDF version) https://pages.cs.wisc.edu/~remzi/OSTEP/
- *Virtual Machines: Versatile Platforms for Systems and Processes* ; J. Smith, R. Nair ; Morgan Kaufmann, 2005
- *Introduction to Operating Systems* ; Prof. Ada Gavrilovska, Georgia Institute of Technology