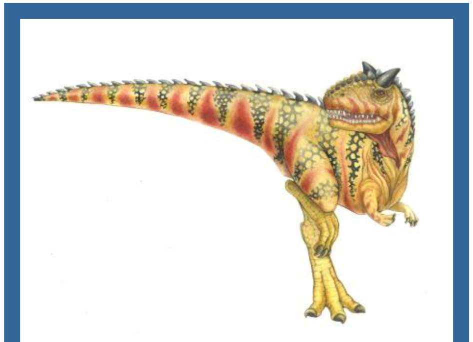
Chapter 18: Virtual Machines



Chapter 18: Virtual Machines

- Overview
- History
- Benefits and Features
- Building Blocks
- Types of Virtual Machines and Their Implementations
- Virtualization and Operating-System Components
- Examples

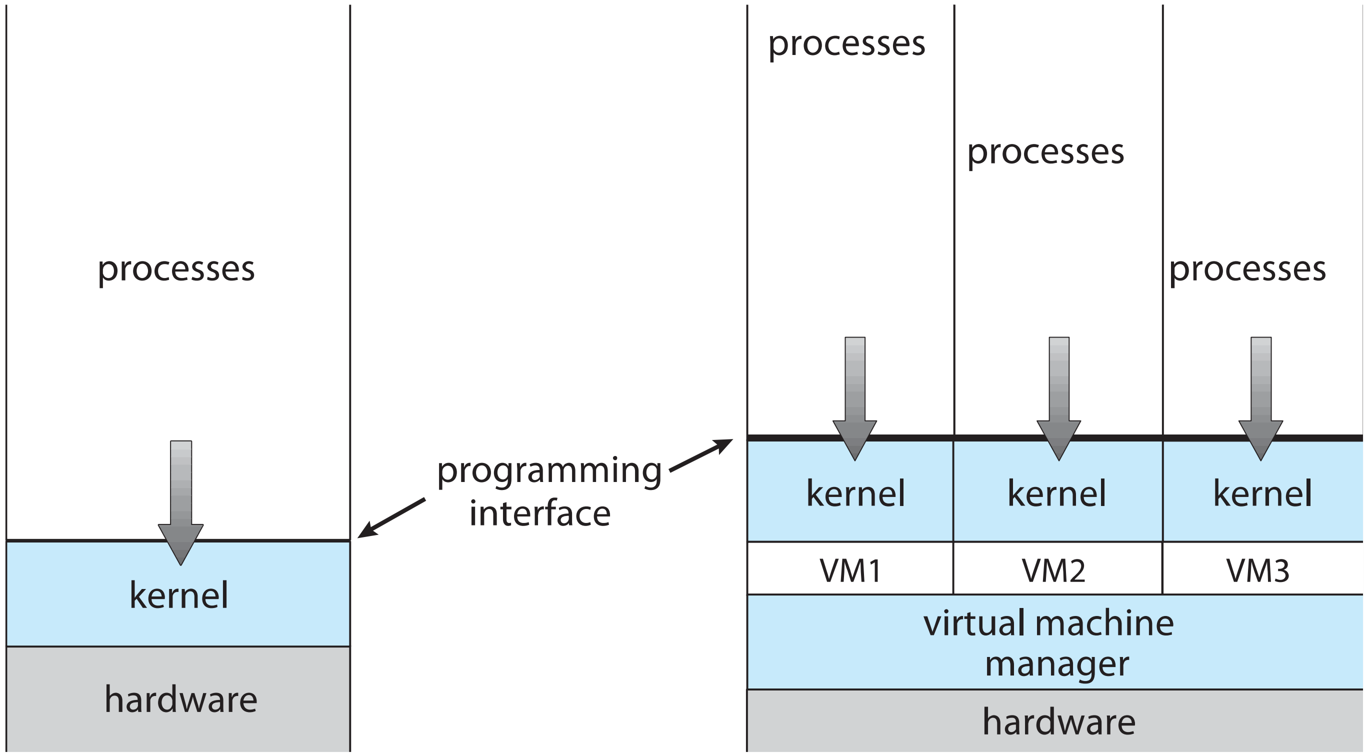
Chapter Objectives

- Explore the history and benefits of virtual machines
- Discuss the various virtual machine technologies
- Describe the methods used to implement virtualization
- Show the most common hardware features that support virtualization and explain how they are used by operating-system modules
- Discuss current virtualization research areas

Overview

- Fundamental idea – abstract hardware of a single computer into several different execution environments
  - Similar to layered approach
  - But layer creates virtual system (**virtual machine**, or **VM**) on which operating systems or applications can run
- Several components
  - **Host** – underlying hardware system
  - **Virtual machine manager** (**VMM**) or **hypervisor** – creates and runs virtual machines by providing interface that is ***identical*** to the host
    - (Except in the case of paravirtualization)
  - **Guest** – process provided with virtual copy of the host
    - Usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

System Models



Non-virtual machine

Virtual machine

Implementation of VMMs

- Vary greatly, with options including:
  - **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
    - IBM LPARs and Oracle LDOMs are examples
  - **Type 1 hypervisors** - Operating-system-like software built to provide virtualization
    - Including VMware ESX, Joyent SmartOS, and Citrix XenServer
  - **Type 1 hypervisors** – Also includes general-purpose operating systems that provide standard functions as well as VMM functions
    - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
  - **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
    - **Including** VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

Implementation of VMMs (Cont.)

- Other variations include:
  - **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
  - **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
    - Used by Oracle Java and Microsoft.Net
  - **Emulators** – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU

Implementation of VMMs (Cont.)

- **Application containment** - Not virtualization at all but rather provides virtualization-like features by segregating applications from the operating system, making them more secure, manageable
  - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs

- Much variation due to breadth, depth and importance of virtualization in modern computing
  - Data-center operations
  - Efficient application development
  - Software testing

History

- First appeared in IBM mainframes in 1972
- Allowed multiple users to share a batch-oriented system
- Formal definition of virtualization helped move it beyond IBM
  1. *Fidelity*: a VMM provides an environment for programs that is essentially identical to the original machine
  2. *Performance*: programs running within that environment show only minor performance decreases
  3. *Safety*: the VMM is in complete control of system resources
- In late 1990s Intel CPUs became fast enough for researchers to try virtualizing on general purpose PCs
  - **Xen** (Type-1, open source, maintained by the Linux Foundation now) and **VMware** created technologies, still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs

Benefits and Features

- Host system protected from VMs, VMs protected from each other
  - i.e., A virus less likely to spread
  - Sharing is provided though via shared file system volume, network communication
- Freeze, **suspend**, a running VM
  - Then can move or copy it somewhere else and **resume**
  - Snapshot of a given state, able to restore back to that state
    - Some VMMs allow multiple snapshots per VM
  - **Clone** by creating copy and running both original and copy
- Great for OS research, better system development efficiency
- Run multiple, different OSes on a single machine (data-center uses)
  - **Consolidation** - combining many lightly used systems into a more heavily used system

Benefits and Features (Cont.)

- **Templating** – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination – this allows few system administrators to manage a large number of systems
- **Live migration** – move a running VM from one host to another!
  - No interruption of user access
  - Used for load balancing and repairing/upgrading host hardware
- Applications can be deployed with a tuned/customized OS in a VM, making app management easier, though it’s best when the format of VMs can be standardized
- All those features taken together -> **cloud computing**
  - Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs
  - Virtual desktops can increase security as no data are stored locally

Building Blocks

- Generally difficult to provide an ***exact*** duplicate of underlying machine
  - Especially if only dual-mode operation available on CPU
  - But getting easier over time as CPU features and support for VMM improve
  - Most VMMs implement **virtual CPU** (**VCPU**) to represent state of CPU per guest as guest believes it to be
    - When guest context switched onto CPU by VMM, information from VCPU is loaded and stored, similar to PCB
  - Several techniques, as described in next slides

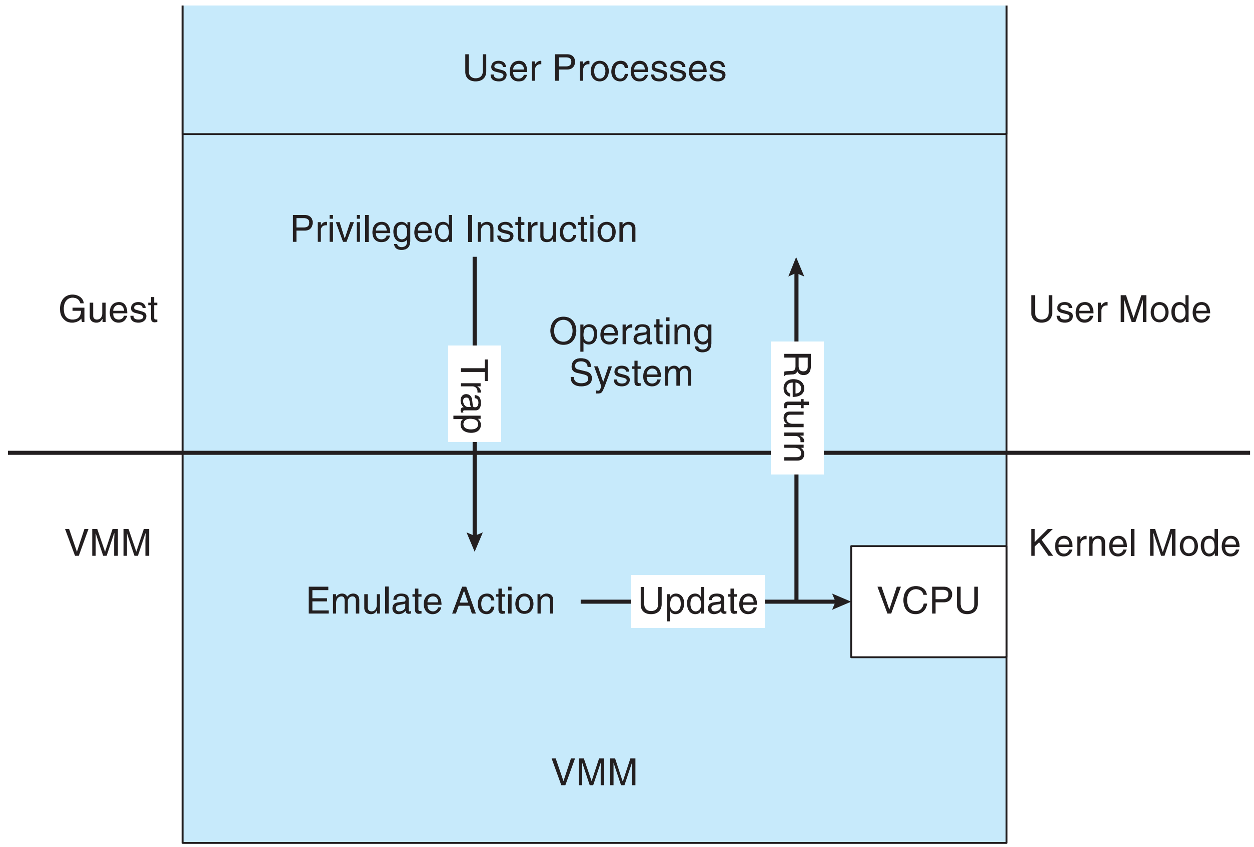
Building Block – Trap and Emulate

- Dual mode CPU means guest executes in user mode
  - Kernel runs in kernel mode
  - Not safe to let guest kernel run in kernel mode too
  - So VM needs two modes – **virtual user mode** and **virtual kernel mode**
    - Both of which run in **real user mode**
  - Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode

Trap-and-Emulate (Cont.)

- How does switch from virtual user mode to virtual kernel mode occur?
  - Attempting a privileged instruction in user mode causes an error -> trap
  - VMM gains control, analyzes error, executes operation as attempted by guest
  - Returns control to guest in user mode
  - Known as **trap-and-emulate**
  - Most virtualization products use this at least in part
- User mode code in guest runs at same speed as if not a guest
- But kernel mode privilege mode code runs slower due to trap-and-emulate
  - Especially a problem when multiple guests running, each needing trap-and-emulate
- CPUs adding hardware support, more CPU modes to improve virtualization performance

Trap-and-Emulate Virtualization Implementation



Building Block – Binary Translation

- Some CPUs don't have clean separation between privileged and nonprivileged instructions
  - Earlier Intel x86 CPUs are among them
    - Earliest Intel CPU (4004 in 1971) designed for a calculator
    - No thought given to running virtualization when first designed
  - Backward compatibility means difficult to improve
  - Consider Intel x86 **`popf`** instruction
    - Loads CPU flags register from contents of the stack
    - If CPU in privileged mode -> all flags replaced
    - If CPU in user mode -> only some flags replaced
      - No trap is generated, so trap-and-emulate won't work

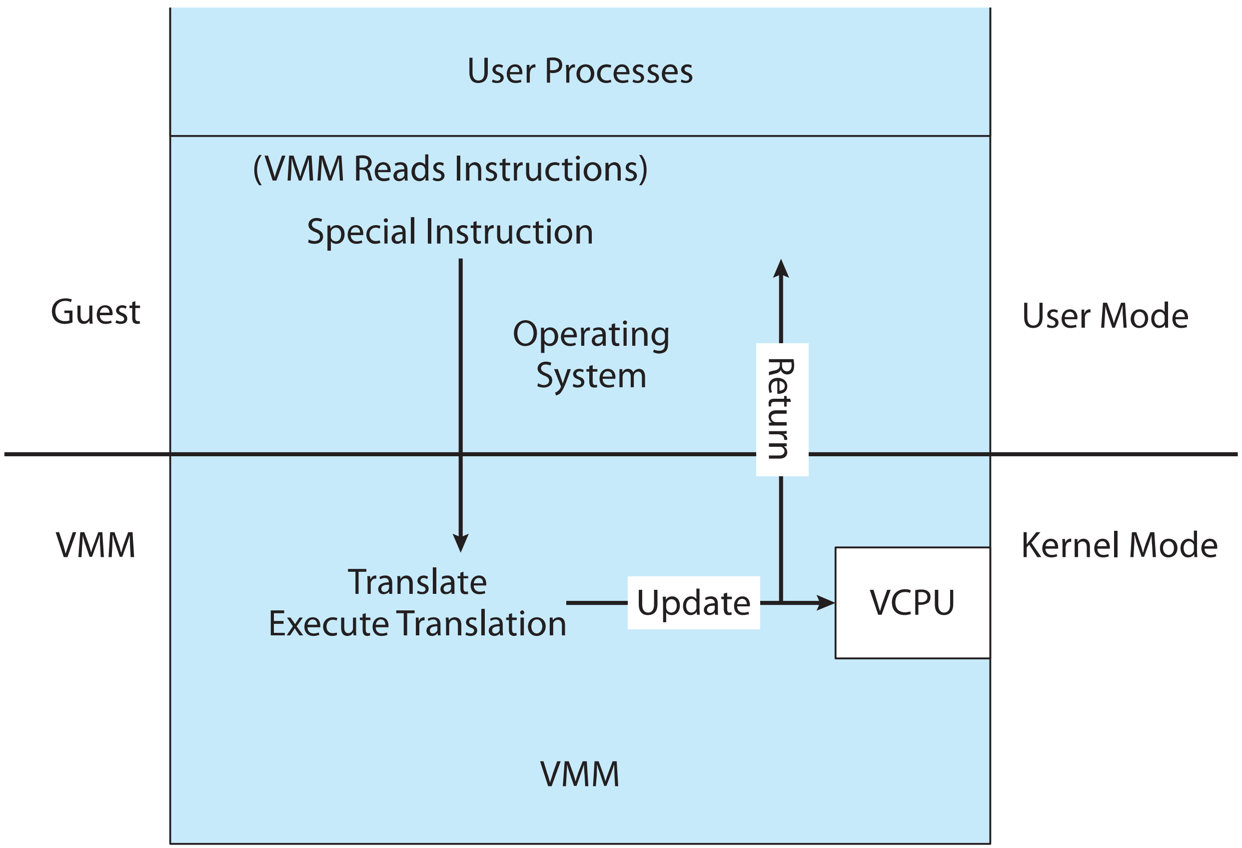
Binary Translation (Cont.)

- Other similar problem instructions we will call ***special instructions***
  - Caused trap-and-emulate method considered impossible until 1998
- Binary translation solves the problem
  1. Basics are simple, but implementation very complex
  2. If guest VCPU is in user mode, guest can run instructions natively
  3. If guest VCPU in kernel mode (guest believes it is in kernel mode)
     a) VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
     b) Non-special-instructions run natively
     c) Special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)

Binary Translation (Cont.)

- Implemented by translation of code within VMM
- Code reads native instructions dynamically from guest, on demand, generates native binary code that executes in place of original code
- Performance of this method would be poor without optimizations
  - Products like VMware use caching
    - Translate once, and when guest executes code containing special instruction cached translation used instead of translating again
    - Testing showed booting Windows XP as guest caused 950,000 translations, at 3 microseconds each, or 3 second (5 %) slowdown over native

Binary Translation Virtualization Implementation



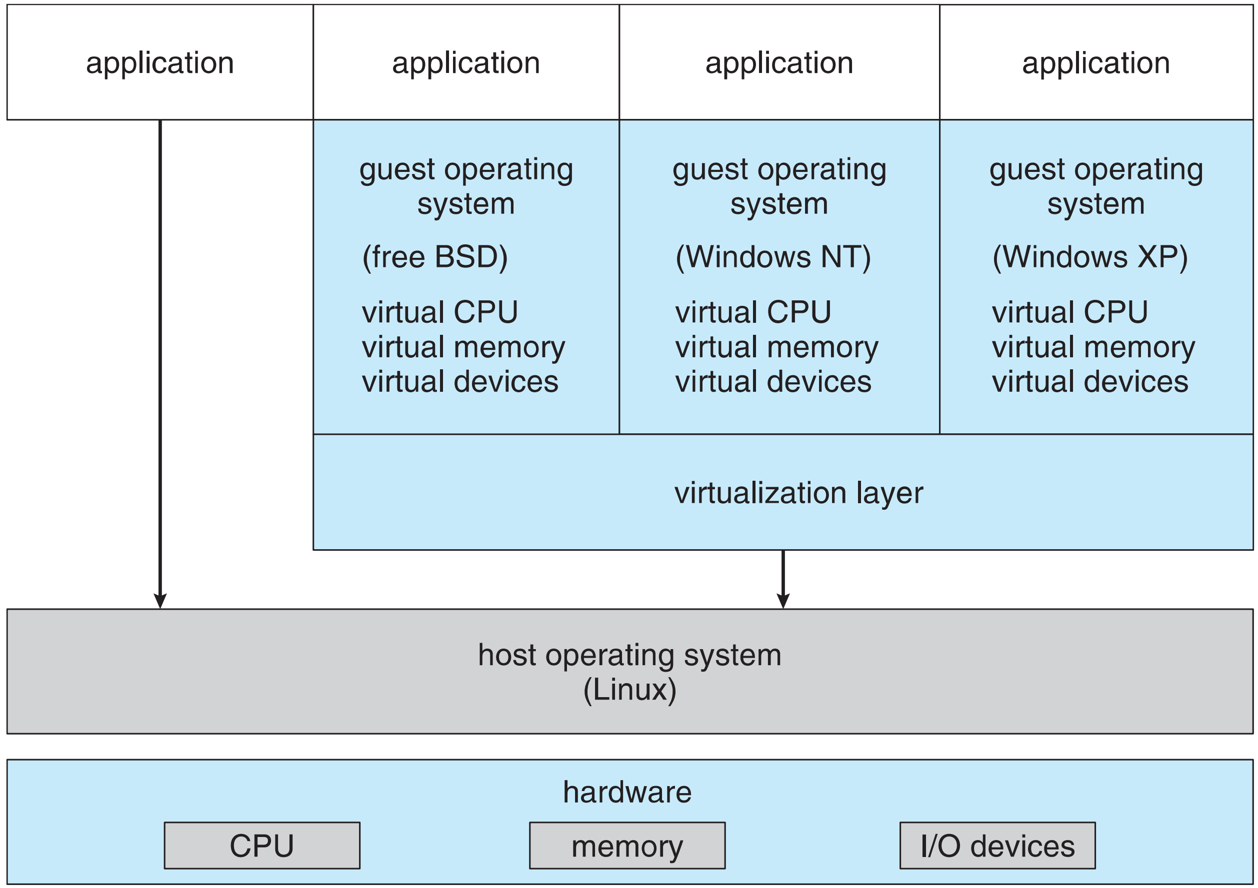
Building Blocks – Hardware Assistance

- All virtualization needs some HW support
- More support -> more feature rich, stable, better performance of guests
- Intel added new **VT-x** instructions in 2005 and AMD the **AMD-V** instructions in 2006
  - CPUs with these instructions remove need for binary translation
  - Generally define more CPU modes – “guest” and “host”
  - VMM can enable host mode, define characteristics of each guest VM, switch to guest mode, passing control to guest OSes
  - In guest mode, guest OS thinks it is running natively, sees devices (as defined by VMM for that guest)
    - Access to virtualized device, privileged instructions cause trap to VMM
    - CPU maintains VCPU, context switches it as needed
- HW support for Nested Page Tables, DMA, interrupts remapping as well over time

Examples - VMware

- VMware Workstation runs on x86, provides VMM for guests
- Runs as application on other native, installed host operating system – i.e., Type 2
- Lots of guests possible, including Windows, Linux, etc. all runnable concurrently (as resources allow)
- Virtualization layer abstracts underlying HW, providing guest with its own virtual CPUs, memory, disk drives, network interfaces, etc.
- Physical disks the guest owns and manages are just files within host file system – can be easily copied to create an identical guest

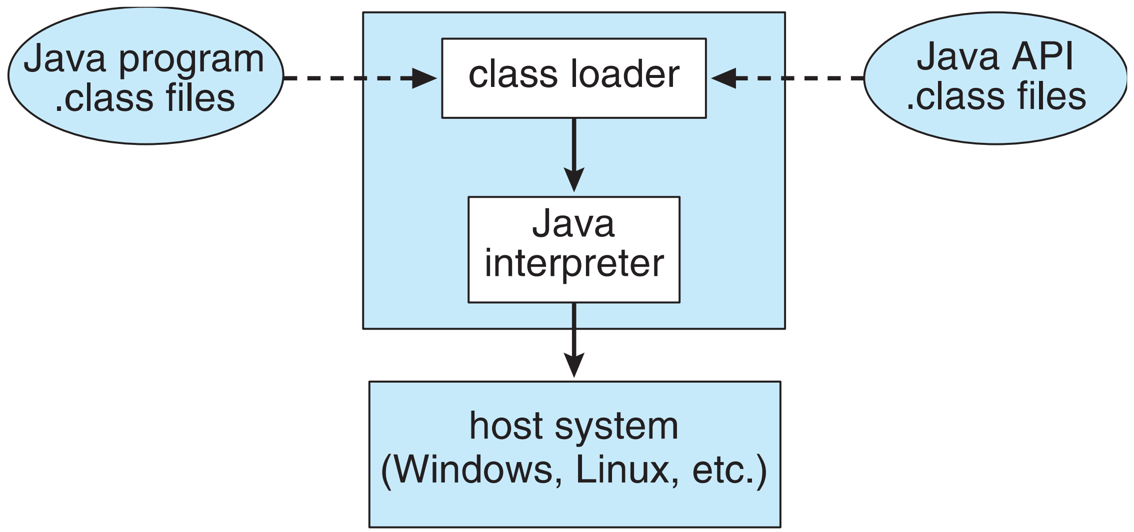
VMware Workstation Architecture



Examples – Java Virtual Machine

- Example of programming-environment virtualization
- Very popular language / application environment invented by Sun Microsystems in 1995
- Write once, run anywhere
- Includes language specification (Java), API library, Java virtual machine (JVM)
- Java objects specified by class construct, Java program is one or more objects
- Each Java object compiled into architecture-neutral **bytecode** output (`.class`) which JVM **class loader** loads
- JVM compiled per architecture, reads and executes bytecode
- Includes **garbage collection** to reclaim memory no longer in use
- Made faster by **just-in-time** (**JIT**) compiler that turns bytecode into native code and caches them

The Java Virtual Machine



End of Chapter 18