A virtual machine (VM) is an isolated environment that appears to be a whole computer but actually

only has access to a portion of the computer resources. Each VM appears to be running on the bare

hardware, giving the appearance of multiple instances of the same computer, though all are supported by

a single physical system. Virtual machines have been around since the early 1970s, when IBM released

its VM/370 operating system.

We distinguish two types of VM: process and system VMs [see Figure 5.3(a)]. A process VM is a

virtual platform created for an individual process and destroyed once the process terminates. Virtually

all operating systems provide a process VM for each one of the applications running, but the more

interesting process VMs are those that support binaries compiled on a different instruction set. A system

VM supports an operating system together with many user processes. When the VM runs under the

control of a normal OS and provides a platform-independent host for a single application, we have an

application virtual machine (e.g., Java Virtual Machine [JVM]).

A literature search reveals the existence of some 60 different virtual machines, many created by the

large software companies; Table 5.1 lists a subset of them.

A system virtual machine provides a complete system; each VM can run its ownOS, which in turn can

run multiple applications. Systems such as Linux Vserver [214], OpenVZ (Open VirtualiZation) [274],

FreeBSD Jails [124], and Solaris Zones [296], based on Linux, FreeBSD, and Solaris, respectively,

implement operating system-level virtualization technologies.

Operating system-level virtualization allows a physical server to run multiple isolated operating

system instances, subject to several constraints; the instances are known as containers, virtual private

servers (VPSs), or virtual environments (VEs). For example, OpenVZ requires both the host and the

guest OS to be Linux distributions. These systems claim performance advantages over the systems based on a VMM such as Xen or VMware; according to [274], there is only a 1% to 3% performance penalty

for OpenVZ compared to a stand-alone Linux server. OpenVZ is licensed under the GPL version 2.

Recall that a VMM allows several virtual machines to share a system. Several organizations of the

software stack are possible:

• Traditional. VM also called a “bare metal” VMM. A thin software layer that runs directly on the

host machine hardware; its main advantage is performance [see Figure 5.3(b)]. Examples: VMWare

ESX, ESXi Servers, Xen, OS370, and Denali.

• Hybrid. The VMM shares the hardware with the existing OS [see Figure 5.3(c)]. Example: VMWare

Workstation.

• Hosted. The VM runs on top of an existing OS [see Figure 5.3(d)]. The main advantage of this

approach is that the VM is easier to build and install. Another advantage of this solution is that the

VMM could use several components of the host OS, such as the scheduler, the pager, and the I/O

drivers, rather than providing its own. A price to pay for this simplicity is the increased overhead and

associated performance penalty; indeed, the I/O operations, page faults, and scheduling requests

from a guest OS are not handled directly by the VMM. Instead, they are passed to the host OS.

Performance as well as the challenges to support complete isolation of VMs make this solution less

attractive for servers in a cloud computing environment. Example: User-mode Linux.

A semantic gap exists between the added services and the virtual machine. As pointed out in [79],

services provided by the virtual machine “operate below the abstractions provided by the guest operating

system . . . . It is difficult to provide a service that checks file system integrity without the knowledge

of on-disk structure.”

The VMMs discussed next manage the resource sharing among the VMs sharing a physical system.