Operating Systems
CS-C3140, Lecture 10
Distributed Computing and Virtualization

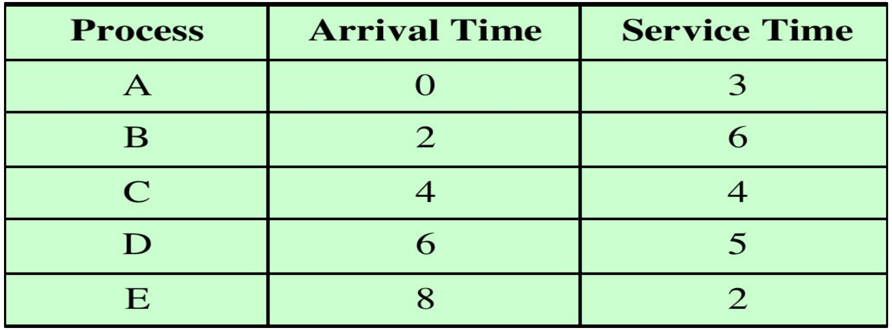
Alexandru Paler

including slides from Mario Di Francesco

Announcements

- This is the last lecture. Next weeks no lecture.
- Next week the Q&A sessions are for asking questions from the lectures
- There will be a bonus assignment
- Exam will not include C programming exercises, but system calls and shell commands might be part of questions
-

Process Scheduling Example



Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

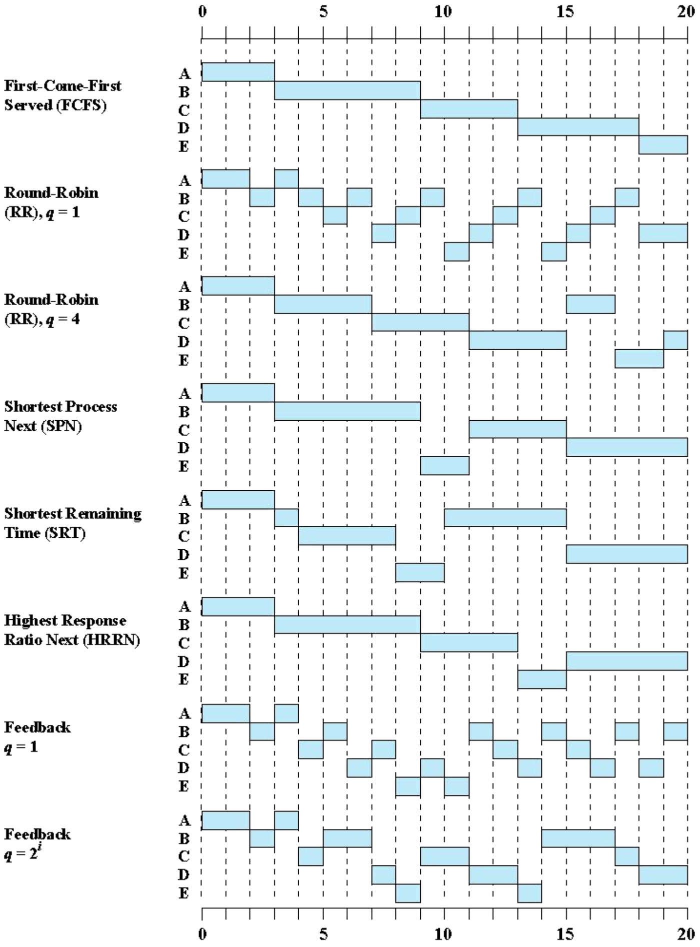
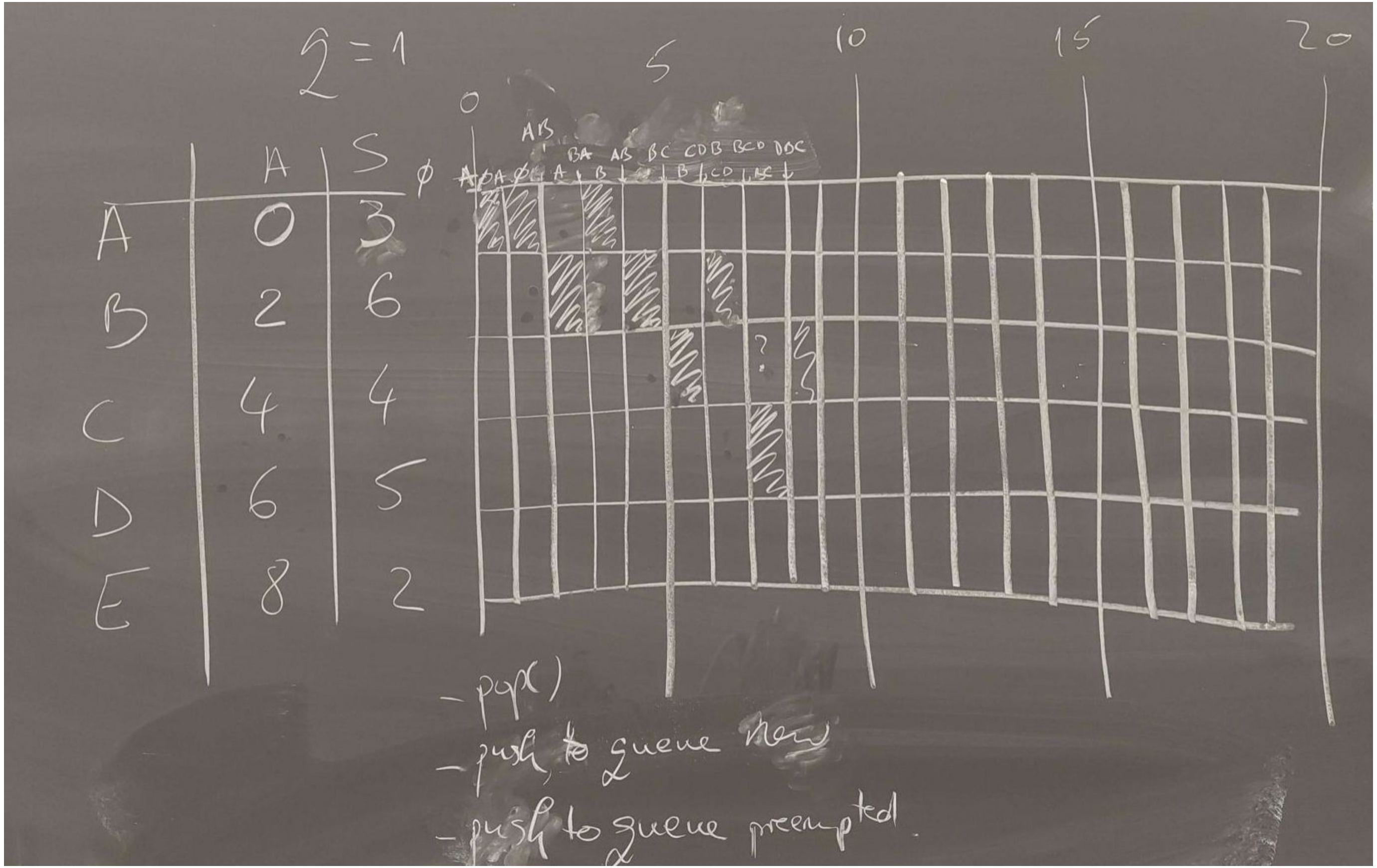


Figure 9.5 A Comparison of Scheduling Policies

Process Scheduling Example: Round Robin



q = 1

	A	S
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

- pop()
- push to queue new
- push to queue preempted.

Networks and Distributed Computing

What is the Internet?

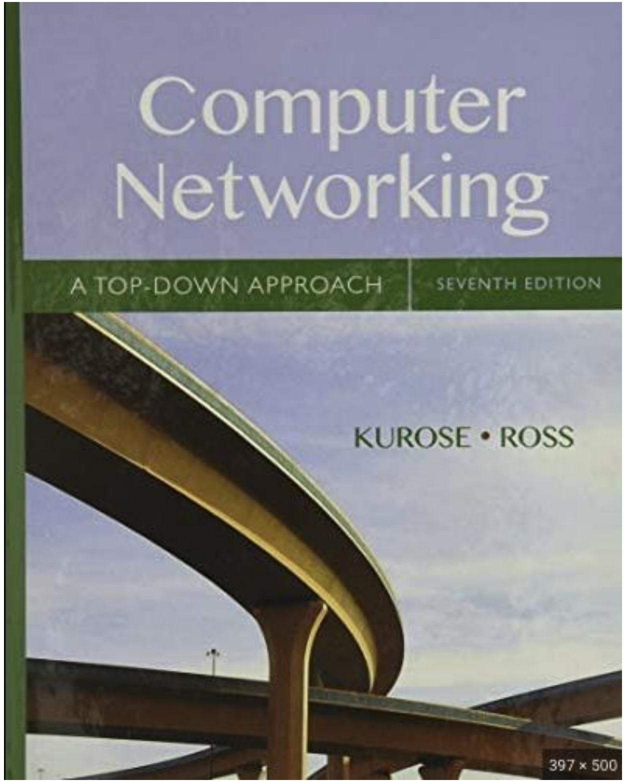
The Internet in a nutshell

- a global-scale communication infrastructure
- consisting of communication links and network elements
  - fiber optics, satellite radio, twisted-pair copper wire
  - routers, access points, switches
- senders and receivers exchange messages
  - route (or path): sequence of links / elements traversed by a message

The Internet as a black box

- the exact details on how messages traverse do not matter
- as long as the sender and the receiver can be identified

Source: James Kurose and Keith W. Ross, “Computer Networking: A Top-Down Approach”, 6th edition, Pearson Education, 2013, Chapter 1



Networks applications and services

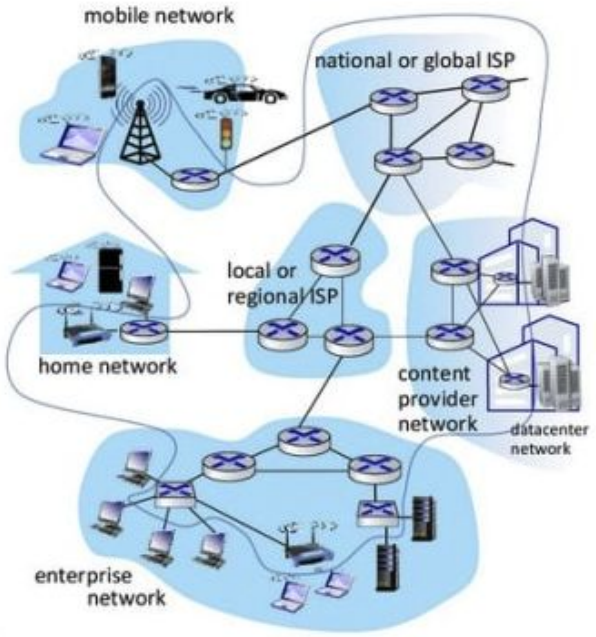
The Internet: a service view

- an infrastructure that provides services to applications
- applications are distributed
- interactions through Application Programming Interfaces (APIs)

Protocol

- specific messages and associated actions in response to them
- a more precise definition
    - a specification of the format and the order of messages exchanged between two or more communicating entities
    - the actions taken on the transmission and (or) receipt of a message or other event

Source: James Kurose and Keith W. Ross, “Computer Networking: A Top-Down Approach”, 6th edition, Pearson Education, 2013, Chapter 1



Applications over the Internet

Network applications

- programs running on different systems
- communicating over the network

Client-server architecture

- client: resource-constrained device
  - desktop,laptop,personal/mobile device (e.g.,smartphone or tablet)
  - limited energy (i.e., battery) and computing capabilities
- server: very powerful hardware
  - machine with plenty of computing, bandwidth and storage resources
  - generally located in a data center
- collectively referred to as hosts or end systems

Source: James Kurose and Keith W. Ross, "Computer Networking: A Top-Down Approach", 6th edition, Pearson Education, 2013, Chapter 2

Clients and servers

- operating system processes in a communication session
- client: initiates the communication
- server: waits to be contacted to start a session

Addressing processes

- an Internet host through an IP address (e.g., 130.233.192.1)
- or by a hostname
  - human-friendly name (e.g., www.aalto.fi)
  - Domain Name System (DNS): directory service translating hostnames to IP addresses
- the receiving process on that host through a port (e.g., 80)

Internet services for different applications

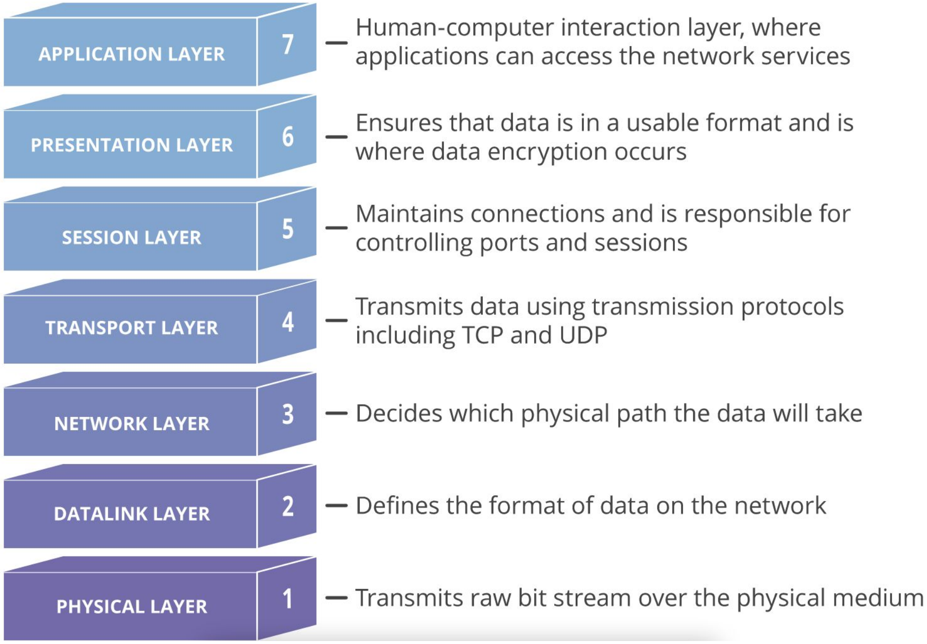
Transport-layer services

- logical communication between application processes running on different hosts
- abstraction from underlying physical infrastructure, as processes were on same machine
- realized by transport protocols

Major transport protocols

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

Source: James Kurose and Keith W. Ross, “Computer Networking: A Top-Down Approach”, 6th edition, Pearson Education, 2013, Chapter 3, Section 3.1


APPLICATION LAYER
7
— Human-computer interaction layer, where applications can access the network services
PRESENTATION LAYER
6
— Ensures that data is in a usable format and is where data encryption occurs
SESSION LAYER
5
— Maintains connections and is responsible for controlling ports and sessions
TRANSPORT LAYER
4
— Transmits data using transmission protocols including TCP and UDP
NETWORK LAYER
3
— Decides which physical path the data will take
DATALINK LAYER
2
— Defines the format of data on the network
PHYSICAL LAYER
1
— Transmits raw bit stream over the physical medium

https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/

Transmission Control Protocol vs. User Datagram Protocol

TCP - Connection-oriented service

- client and servers need to exchange control information before sending and receiving actual application messages
- such handshaking process creates a TCP connection
- full-duplex channel: messages are sent / received over the connection at the same time

Reliable transport service

- recovers from errors and lost messages
- ensures correct message ordering and no duplicates
- adapts to available bandwidth between sender and receiver

Vinton Cerf and Robert Kahn, Turing prize 2004

UDP - Connectionless service

- very lightweight solution
- no handshaking

Unreliable transport service

- no guarantee that messages will be correctly delivered, if at all
  - errors and message loss are not handled
  - there could be duplicate messages
  - the order of messages may not be preserved
- does not consider network congestion
- application could still implement needed features on top of UDP
  - Domain Name System (DNS)
  - Streaming media applications such as movies
  - Online multiplayer games
  - Voice over IP (VoIP)
  - Trivial File Transfer Protocol (TFTP)

(Indirect) Communication between processes - lecture 7

Shared resources and synchronization

- locks, semaphores and monitors
- shared address space between processes
- communication through shared data

Message passing and data transfer

- use two communication primitives: send and receive
- possibly work over multiple machines, even with different operating systems
- implicit synchronization, no need for mutual exclusion

Pipe

- an output stream connected to an input stream through an in-memory queue uni-directional communication
- send is non-blocking, receive is blocking
- the operating system takes care of buffering

Mailbox

- allow many-to-one or one-to-many communication
- still uni-directional

Pipes in UNIX

Anonymous pipes

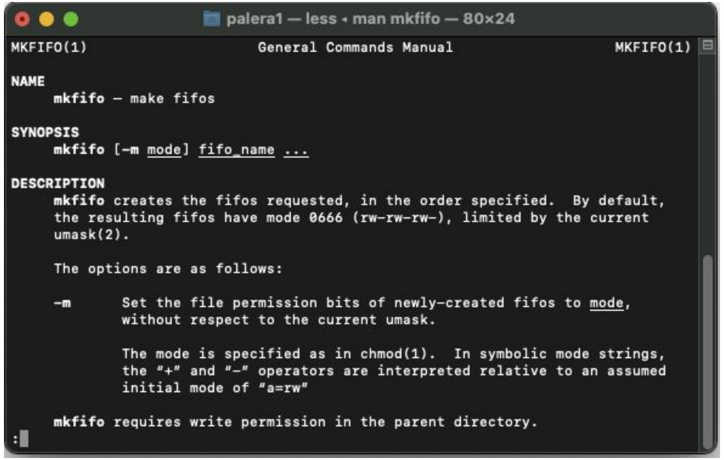
- output of one process becomes input of another process
- indicated with a vertical bar (i.e., |) in the command line
  - example:

```
echo "Operating systems" | sed "s/systems/machines/g"
```

Named pipes

- entities in a filesystem, subject to related permissions
- also called FIFOs as they use the First In First Out policy
- created with the mkfifo command / function

Sources: Computerphile, “Unix Pipeline (Brian Kernighan)” and David A. Rusling, “The Linux Kernel”, 1999, Section 5.2



```
MKFIFO(1)                General Commands Manual                MKFIFO(1)

NAME
     mkfifo – make fifos

SYNOPSIS
     mkfifo [-m mode] fifo_name ...

DESCRIPTION
     mkfifo creates the fifos requested, in the order specified.  By default,
     the resulting fifos have mode 0666 (rw-rw-rw-), limited by the current
     umask(2).

     The options are as follows:

     -m      Set the file permission bits of newly-created fifos to mode,
             without respect to the current umask.

             The mode is specified as in chmod(1).  In symbolic mode strings,
             the “+” and “-” operators are interpreted relative to an assumed
             initial mode of “a=rw”

     mkfifo requires write permission in the parent directory.
:
```

Sockets

Definition

- bi-directional mechanism
- for communication both locally and over a network
- same application programming interface

UNIX domain sockets

- similar to Internet sockets but limited to the same machine
- both named and unnamed
- generally preferred to named pipes to implement inter-process communication for important services

Source: David A. Wheeler, “Secure Programming HOWTO”, Version 3.72 (September 19, 2015) □ Section 3.4

Network sockets

Features

- transport services for different application requirements
- client-server paradigm

Socket API

- creation / deletion

```
sid = socket(af, type, protocol); se = close(sid);
```

- connection setup

```
se = bind(sid, localaddr, addrlength);
```

```
listen(sid, length);
```

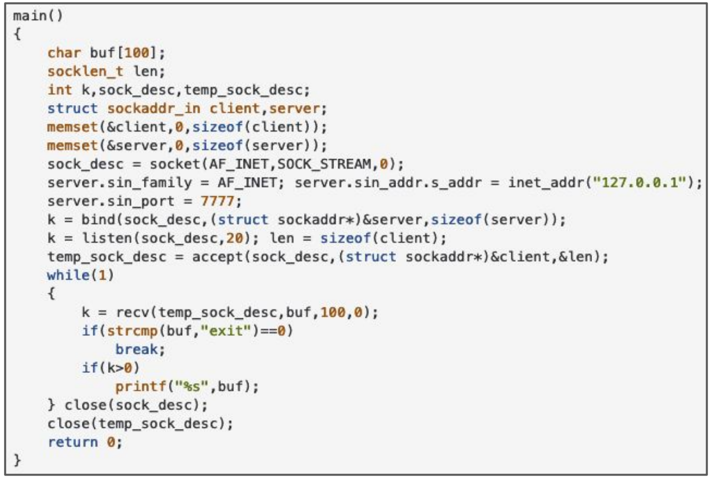
```
accept(sid, addr, length);
```

```
connect(sid, destaddr, addrlength);
```

- message passing

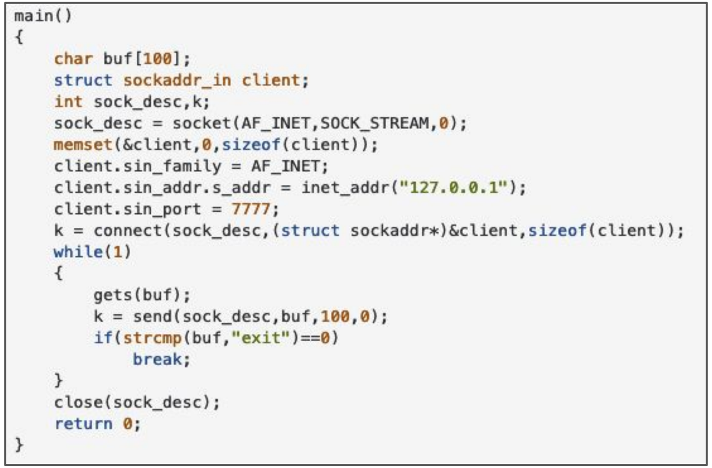
```
send(sid, buf, size, flags);
```

```
recv(sid, buf, size, flags);
```



```
main()
{
    char buf[100];
    socklen_t len;
    int k,sock_desc,temp_sock_desc;
    struct sockaddr_in client,server;
    memset(&client,0,sizeof(client));
    memset(&server,0,sizeof(server));
    sock_desc = socket(AF_INET,SOCK_STREAM,0);
    server.sin_family = AF_INET; server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = 7777;
    k = bind(sock_desc,(struct sockaddr*)&server,sizeof(server));
    k = listen(sock_desc,20); len = sizeof(client);
    temp_sock_desc = accept(sock_desc,(struct sockaddr*)&client,&len);
    while(1)
    {
        k = recv(temp_sock_desc,buf,100,0);
        if(strcmp(buf,"exit")==0)
            break;
        if(k>0)
            printf("%s",buf);
    } close(sock_desc);
    close(temp_sock_desc);
    return 0;
}
```

SERVER



```
main()
{
    char buf[100];
    struct sockaddr_in client;
    int sock_desc,k;
    sock_desc = socket(AF_INET,SOCK_STREAM,0);
    memset(&client,0,sizeof(client));
    client.sin_family = AF_INET;
    client.sin_addr.s_addr = inet_addr("127.0.0.1");
    client.sin_port = 7777;
    k = connect(sock_desc,(struct sockaddr*)&client,sizeof(client));
    while(1)
    {
        gets(buf);
        k = send(sock_desc,buf,100,0);
        if(strcmp(buf,"exit")==0)
            break;
    }
    close(sock_desc);
    return 0;
}
```

CLIENT

https://stackoverflow.com/questions/9198608/simple-socket-programming-code-working

Virtualization

Virtualization

Definition

- the creation of a virtual version of a computing resource
- not physically existing as such but made by software to appear to do so

History

- fundamental research already in the early 70s
- only recently realized in practice due to availability of resources and advances in computer systems architecture

Source: Oxford Dictionaries (https://en.oxforddictionaries.com/definition/virtualize)

Key Reasons for Using Virtualization

- We can summarize the key reasons the organizations use virtualization as follows:
- Legacy hardware
  - Applications built for legacy hardware can still be run by virtualizing the legacy hardware, enabling the retirement of the old hardware
- Rapid deployment
  - A new VM may be deployed in a matter of minutes
- Versatility
  - Hardware usage can be optimized by maximizing the number of kinds of applications that a single computer can handle

- Consolidation
  - A large-capacity or high-speed resource can be used more efficiently by sharing the resource among multiple applications simultaneously
- Aggregating
  - Virtualization makes it easy to combine multiple resources in to one virtual resource, such as in the case of storage virtualization
- Dynamics
  - Hardware resources can be easily allocated in a dynamic fashion, enhancing load balancing and fault tolerance
- Ease of management
  - Virtual machines facilitate deployment and testing of software
- Increased availability
  - Virtual machine hosts are clustered together to form pools of compute resources

Application virtual machine

Process virtual machine

- a runtime software encapsulates a process
- it appears to the operating system as a native process

High-level language virtual machines

- translate a certain programming language into machine-independent bytecode
- bytecode is converted into machine code for execution on a given hardware platform
  - at runtime by interpreters or just-in-time compilers
  - entirely beforehand by ahead-of-time compilers
- examples: Java VM and Android Runtime (ART)

Virtualization taxonomy

Platform virtualization

- also called system or server virtualization
- software layer encapsulating an operating system to replicate behavior of a physical machine

Operating system-level virtualization

- also called container-based virtualization
- shares the resources of the underlying operating system

Desktop virtualization

- also called virtual desktop infrastructure
- based on thin-client remote desktop solutions

Source: M. Pearce, S. Zeadally, and R. Hunt, “Virtualization: Issues, security threats, and solutions”, ACM Computing Surveys 45(2):17, February 2013

From workstations to data centers

Conventional computing systems

- process as primary computing abstraction
- operating systems allow process multiplexing and hardware management through a high-level interface
- privileged software (kernel space), regular programs only employ system calls (user space)
- one operating system per physical machine

Modern computing systems

- several cores, many threads
- very powerful specialized hardware
- may run multiple operating system at once

Data Center vs Cloud Computing

Definition

- a building with multiple interconnected and co-located servers
- due to different requirements, including physical security and ease of maintenance

What does a data center need?

- space
  - warehouse-scale computers: an extremely large number of servers, together with their software platforms
- power
- cooling

Source: Luiz André Barroso et al., The Datacenter as a Computer, Morgan & Claypool, 2013

Definition

- a model for enabling ubiquitous, [...] on-demand network access
- to a shared pool of configurable computing resources
  - e.g., networks, servers, storage, applications, and services
- that can be rapidly provisioned and released
- with minimal management effort or service provider interaction

Virtualization as key enabling technology

- isolation
  - applications in a virtual environment cannot access other applications running on the same physical machine
- resource control
  - applications in a virtual environment cannot use more than the resources they have been assigned

Source: P. Mell and T. Grance, “The NIST Definition of Cloud Computing”, September 2011

Platform Virtualization

Platform Virtualization: Basics

Virtual machine monitor

- also called hypervisor
- provides an environment that looks like the physical system but decoupled from the hardware state

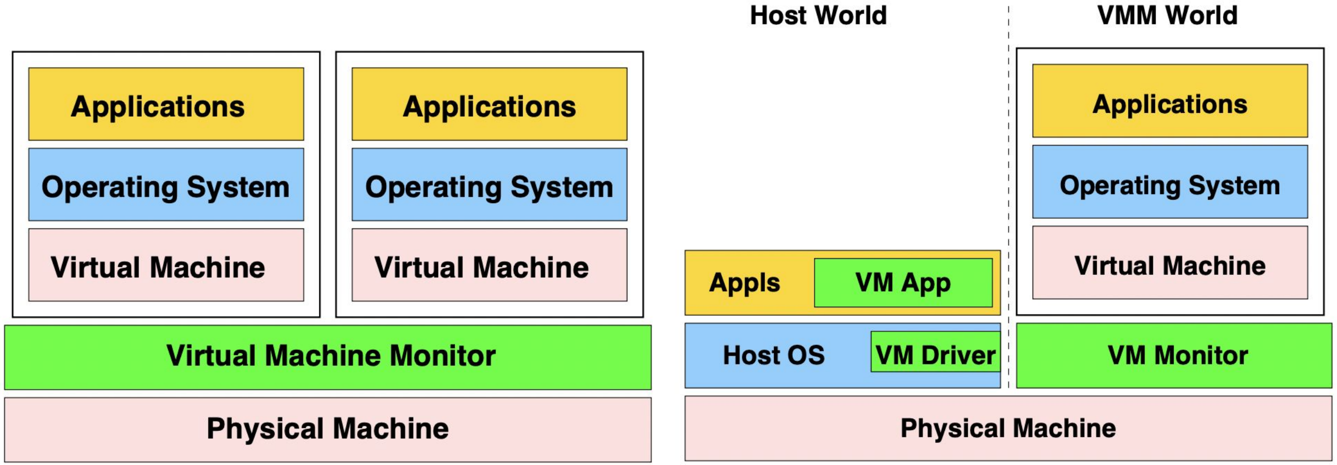
Virtual machine

- virtual environment in platform virtualization
- runs an operating system

Terminology

- host: the physical machine running the hypervisor
- guest: the operating system running within a VM

Types of virtual machine monitors



Bare metal or **Type-I**

Hosted or **Type-II**

Key properties and benefits

Isolation

- virtual machines cannot interfere with each other
- not even aware that others are running on the same machine

Resource sharing

- most servers run at a low utilization (typically less than 15%)
- share one physical machine between different operating systems
- consolidate load on less active servers

Hardware independence

- virtual machines run on any cloud infrastructure
- easy to move them from one data center to another

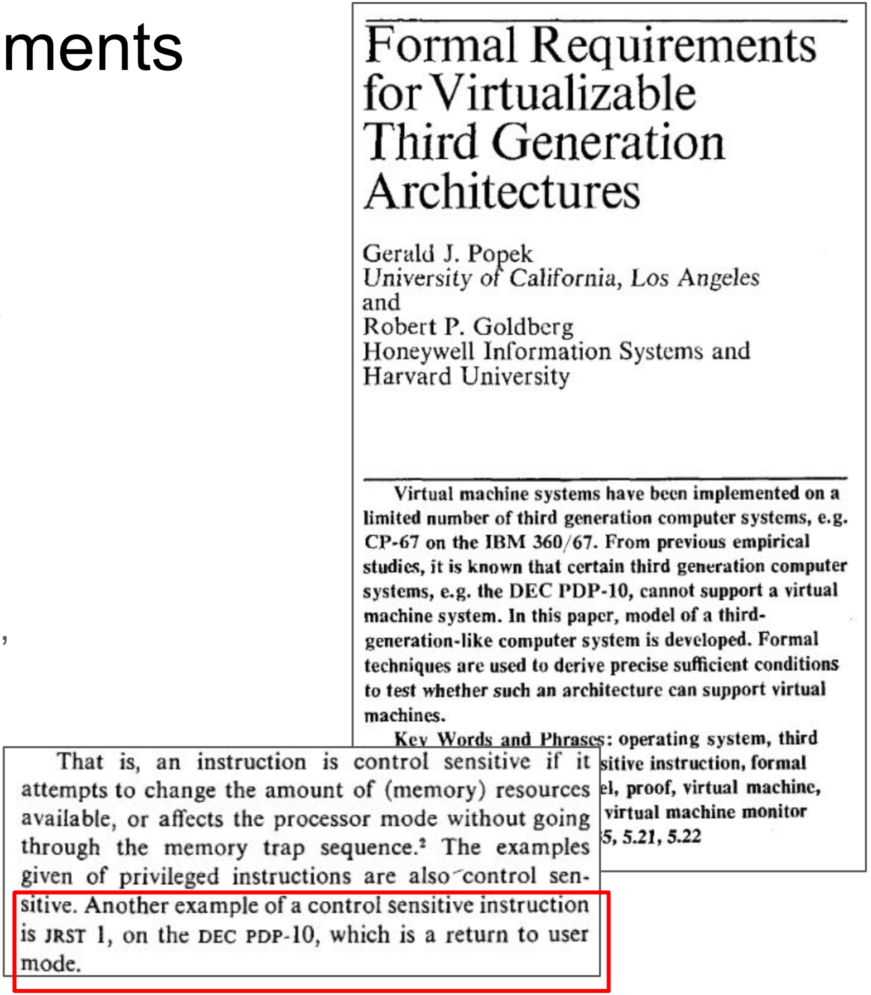
Platform virtualization: requirements

Not all systems can be virtualized

- it depends on the specific hardware architecture
- general criteria exist to assess virtualization
- what if a certain architecture cannot be virtualized?
  - construct a hybrid virtual machine

Classical virtualization

- according to the criteria in [Popek and Goldberg, CACM 1974]
- widely used still today
- based on two classes of properties
  - for instructions
  - for the virtualized architecture



Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

Virtual machine systems have been implemented on a limited number of third generation computer systems, e.g. CP-67 on the IBM 360/67. From previous empirical studies, it is known that certain third generation computer systems, e.g. the DEC PDP-10, cannot support a virtual machine system. In this paper, model of a third-generation-like computer system is developed. Formal techniques are used to derive precise sufficient conditions to test whether such an architecture can support virtual machines.

Key Words and Phrases: operating system, third
sitive instruction, formal
el, proof, virtual machine,
virtual machine monitor
5, 5.21, 5.22

That is, an instruction is control sensitive if it attempts to change the amount of (memory) resources available, or affects the processor mode without going through the memory trap sequence.[2] The examples given of privileged instructions are also control sensitive. Another example of a control sensitive instruction is JRST 1, on the DEC PDP-10, which is a return to user mode.

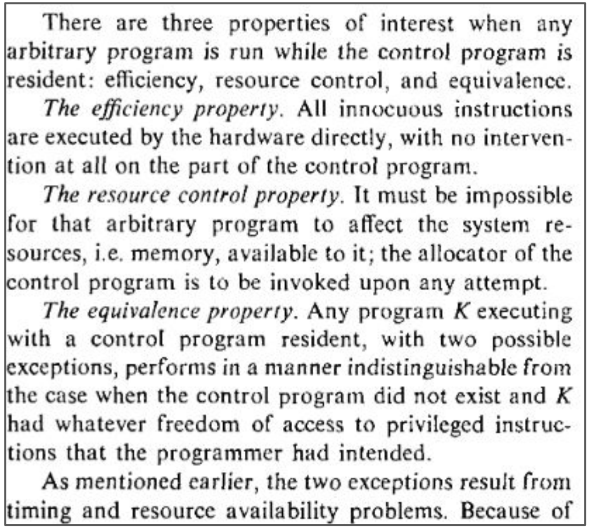
Properties of classical virtualization

Instruction properties

- Privilege: a privileged instruction allows a user space process to run a kernel space function, for instance, system calls invoking a software interrupt (trap)
- Sensitivity: a sensitive instruction may interfere with the hypervisor

Virtualized architecture properties

- Efficiency: hypervisor runs non-sensitive instructions directly
- **Control**: guest cannot bypass the hypervisor to access or manipulate resources
- Equivalence: hypervisor behaves the same as a physical machine, except for timing and resource availability



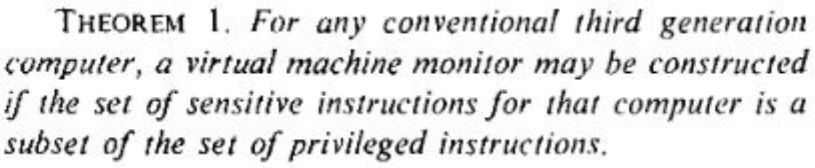
There are three properties of interest when any arbitrary program is run while the control program is resident: efficiency, resource control, and equivalence.

The efficiency property. All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program.

The resource control property. It must be impossible for that arbitrary program to affect the system resources, i.e. memory, available to it; the allocator of the control program is to be invoked upon any attempt.

The equivalence property. Any program *K* executing with a control program resident, with two possible exceptions, performs in a manner indistinguishable from the case when the control program did not exist and *K* had whatever freedom of access to privileged instructions that the programmer had intended.

As mentioned earlier, the two exceptions result from timing and resource availability problems. Because of

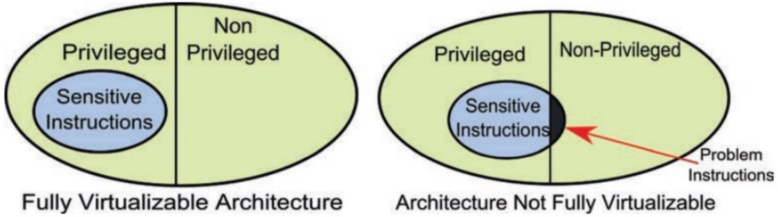


THEOREM 1. *For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.*

Classically virtualizable architectures

Constraints for full virtualization

- "an architecture is fully virtualizable if the set of sensitive instructions [...] is a subset of the set of privileged instructions"
- the x86 architecture is not fully virtualizable



Source: M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions", ACM Computing Surveys 45(2):17, February 2013

Non-virtualizable architectures - Approach

- construct a hybrid virtual machine
- special handling of critical instructions (i.e., non-privileged but sensitive)

Comparison with emulation

- emulator handles all instructions
- hypervisor only handles sensitive instructions
- non-sensitive instructions run on the hardware
  - consequence of the efficiency property

Paravirtualization and Hardware-assisted virtualization

Main idea

- the virtual machine abstraction is similar to (but not the same as) the real hardware

Approach

- modify operating system
- replace critical instructions with hypervisor calls
- fast but requires access to the source code
- first realized by Xen

Architecture extensions

- special functions added to hardware platforms to support virtualization
- define the structure of a hardware-based hypervisor
- I/O Memory Management Unit (IOMMU)

Hardware virtualization technologies

- AMD Virtualization (AMD-V)
- Intel Virtualization Technologies
    - CPU virtualization (VT-x)
    - memory virtualization (VT-d)
- ARM Virtualization Extensions

Virtualization software

VirtualBox

- cross-platform virtualization application, desktop-oriented
- hosted hypervisor

XEN

- open-source bare-metal hypervisor
- initially provided para-virtualization only, currently supports hardware-assisted virtualization too

KVM

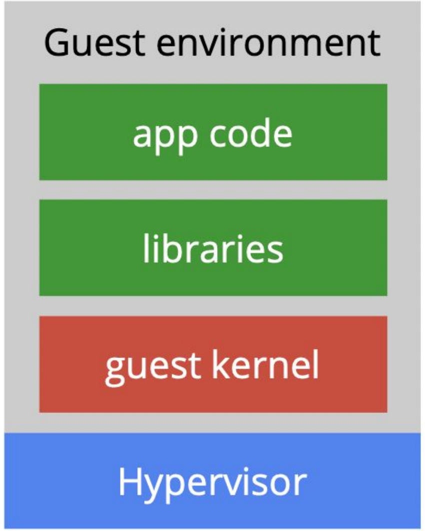
- full open-source virtualization solution for the Linux kernel
- hosted hypervisor
- hardware-assisted virtualization only

Containers and Docker

Issues with virtual machines (1/2)

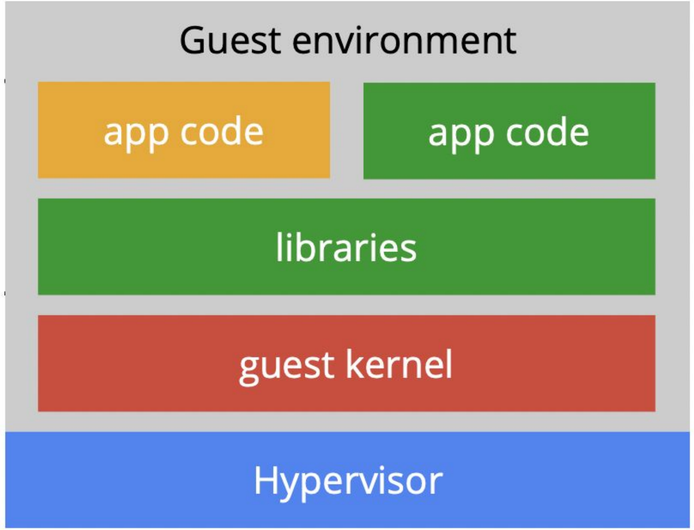
Portability

- platform-bound image



Isolation

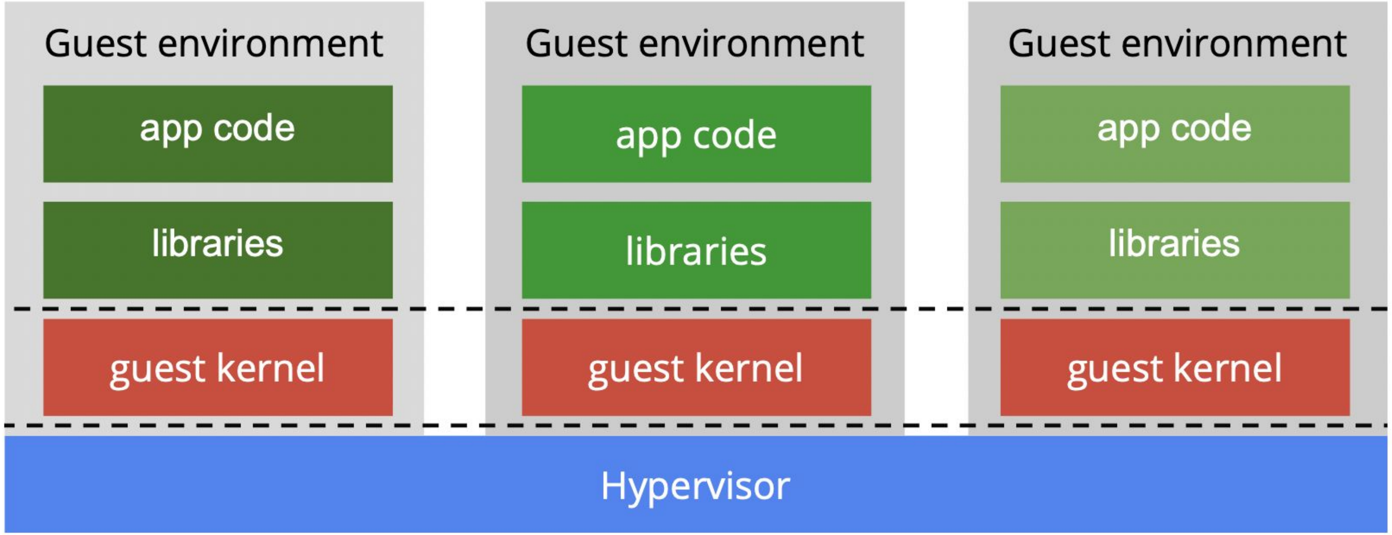
- dependencies



Issues with virtual machines (2/2)

Redundancy

- limited component reuse
- multiple unnecessary guest operating systems



Containers as virtualization technology

Operating system-level virtualization

- also called container-based virtualization
- shares the resources of the underlying operating system
- additional mechanisms for isolation between processes

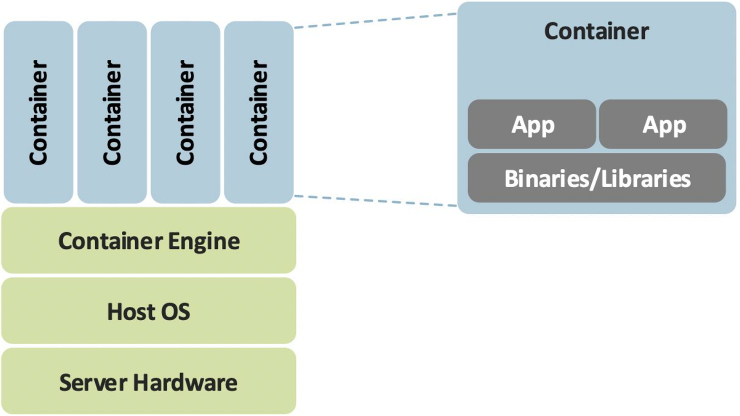
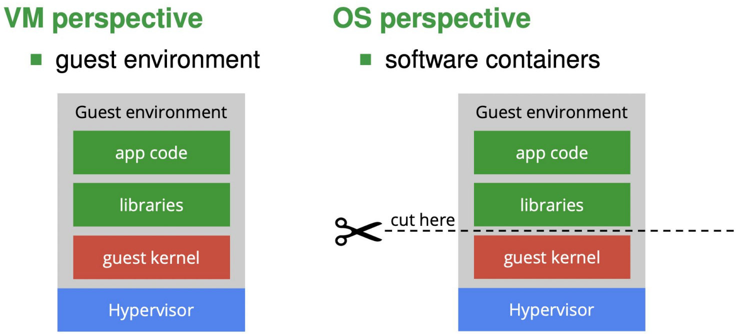


Image source: Asad Javed, “Container-based IoT Sensor Node on Raspberry Pi and the Kubernetes Cluster Framework”, Master thesis, Aalto University, 2016



Isolation in UNIX and FreeBSD, Linux

Change root

- chroot command
- changes the root of the filesystem
- does not attempt to enforce strict isolation

Jails

- also referred to as chroot jail
- facility to partition the operating system environment
- keeps the traditional UNIX security model based on root user
- processes in the jail see local resources only

Control groups

- also known as cgroups
- fine-grained resource allocation and limitation
- hierarchical structure

Namespaces

- system resources wrapped in abstraction layers
- appear to processes as isolated instances
- separate namespaces for different subsystems
    - mount (filesystems), UNIX Time-sharing Subsystem UTS (node and domain names), interprocess communication (POSIX queues), process IDs, network, users

Containers beyond virtualization

Packaging

- self-contained images
  - similar to an application package
- stored in a repository
  - most widely used: Docker Hub

Management

- automation
  - continuous integration and delivery
- container orchestration
  - examples: Docker Swarm, Kubernetes

There are several software platforms for creating, shipping, and running containers. Docker is so prevalent in the industry (and beyond) that Docker containers and software containers are commonly used as synonyms.

Separation of Concerns

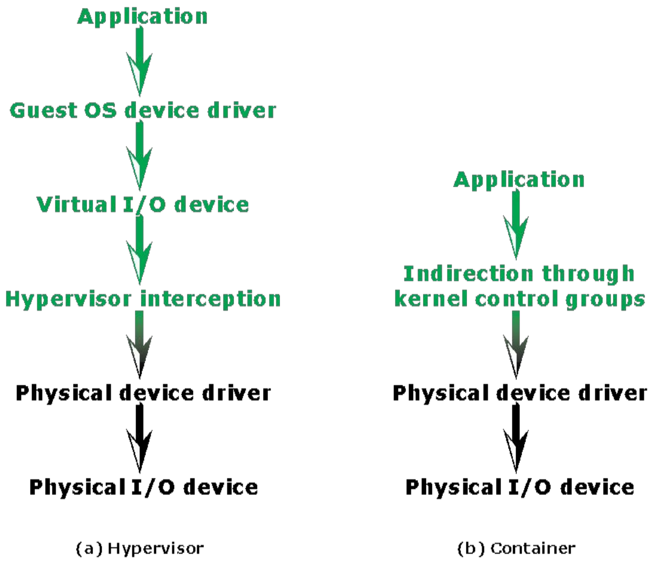
Developer view

- what is inside a container
- code, libraries, applications, data
- base environment: Linux server

System administrator view

- what is outside a container
- logging, monitoring, networking
- consistent interface for lifecycle management

Data Flow for I/O Operation via Hypervisor and Container



To compare virtual machines with containers, consider I/O operation in during an application with process P in virtualized environment.

- In classical system virtualization environment (with no hardware support), process P would be executed inside a guest virtual machine
  - I/O operation is routed through guest OS stack to emulated guest I/O device
  - I/O call is further intercepted by hypervisor that forward it through host OS stack to the physical device
- The container is primarily based on indirection mechanism provided by container framework extensions that have been incorporated into mainstream kernel
  - a single kernel is shared between multiple containers
  - in comparison with individual OS kernel in system virtual machines

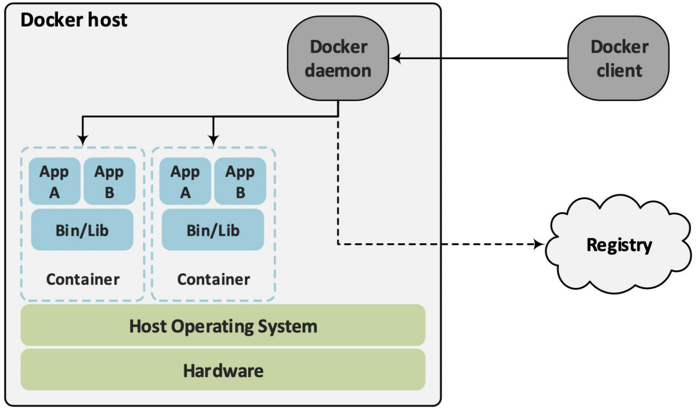
The two noteworthy characteristics of containers are the following:

1. There is no need for a guest OS in the container environment. Therefore, containers are lightweight and have less overhead compared to virtual machines.
2. Container management software simplifies the procedure for container creation and management.

Docker and its Components

Overview

- open-source engine for deployment of containerized applications
- client-server architecture with a few core components



Daemon

- server with an application programming interface
- manages containers

Client

- command-line interface to interact with the daemon

Host

- the machine running the containers
  - with an OS supporting container-based virtualization

Registry

- public or private image repository
  - examples: Docker Hub, Google Container Registry

Docker and containers

Management

- docker command-line application
- similar to standard tools for dealing with UNIX processes
  - `docker ps, docker kill, docker exec`

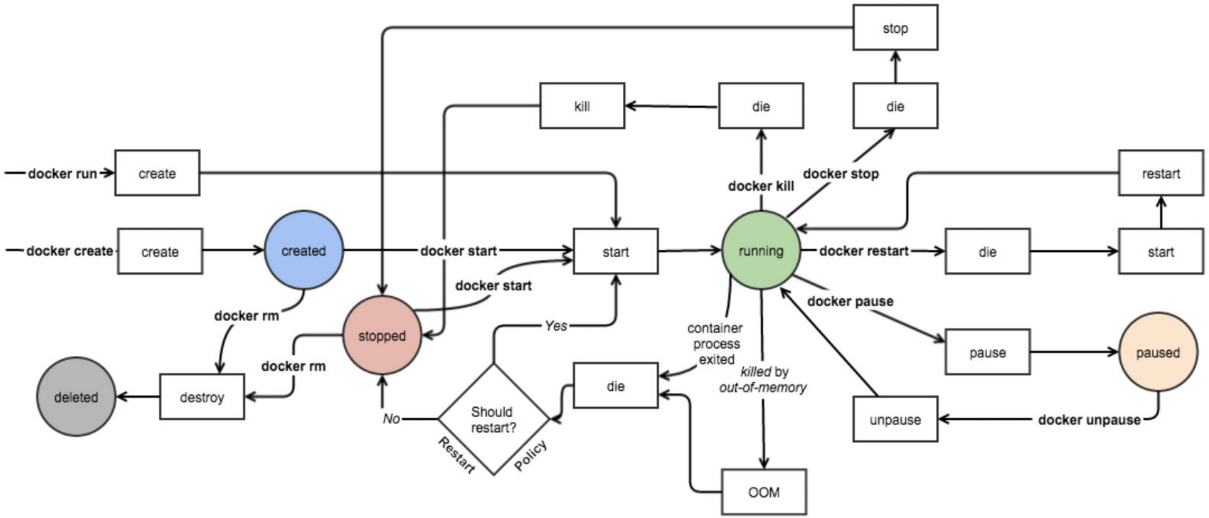
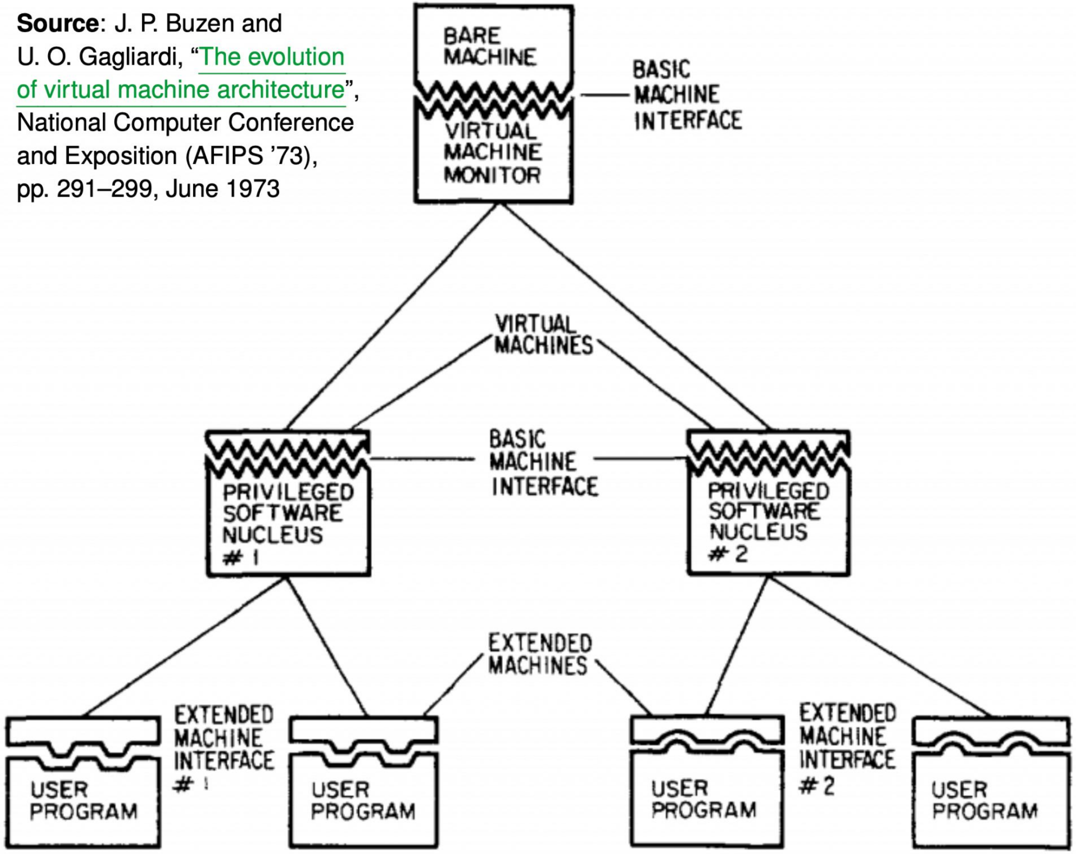


Image source: Docker Saigon, “Docker Internals – A Deep Dive Into Docker For Engineers Interested In The Gritty Details”

Virtualization



Source: J. P. Buzen and U. O. Gagliardi, “The evolution of virtual machine architecture”, National Computer Conference and Exposition (AFIPS ’73), pp. 291–299, June 1973