Operating Systems*

*Throughout the course we will use overheads that were adapted from those distributed from the textbook website. Slides are from the book authors, modified and selected by Jean Mayo, Shuai Wang and C-K Shene.

It takes a really bad school to ruin a good student
and
a really fantastic school to rescue a bad student.

Dennis J. Frailey

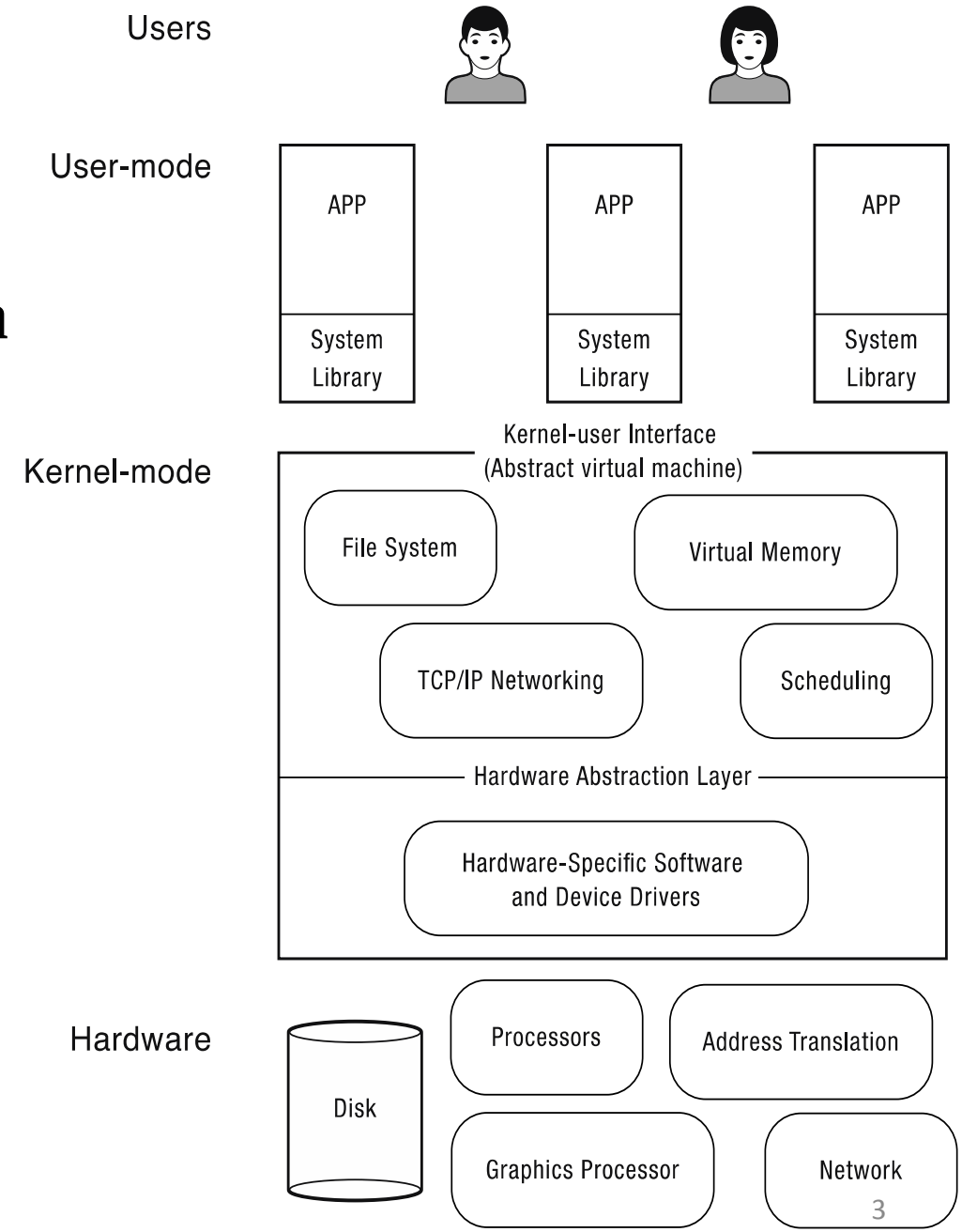
Spring 2019

Overview

- **Operating system definition**
  - **Software to manage a computer's resources for its users and applications**
- **OS challenges**
  - **Reliability, security, responsiveness, portability, …**

What is an Operating System?

- Software to manage a computer's resources for its users and applications



Operating System Roles

- ***Referee***:
  - Resource allocation among users, applications
  - Isolation of different users, applications from each other
  - Communication between users, applications
- ***Illusionist***:
  - Each application appears to have the entire machine to itself
  - Infinite number of processors, (near) infinite amount of memory, reliable storage, reliable network transport
- ***Glue***:
  - Libraries, user interface widgets, …

Example: File Systems

- ***Referee***:
  - Prevent users from accessing each other's files without permission
  - Even after a file is deleted and its space re-used
- ***Illusionist***:
  - Files can grow (nearly) arbitrarily large
  - Files persist even when the machine crashes in the middle of a save
- ***Glue***:
  - Named directories, `printf`, …

Question

- **What (hardware, software) do you need to be able to run an untrustworthy application?**

Question

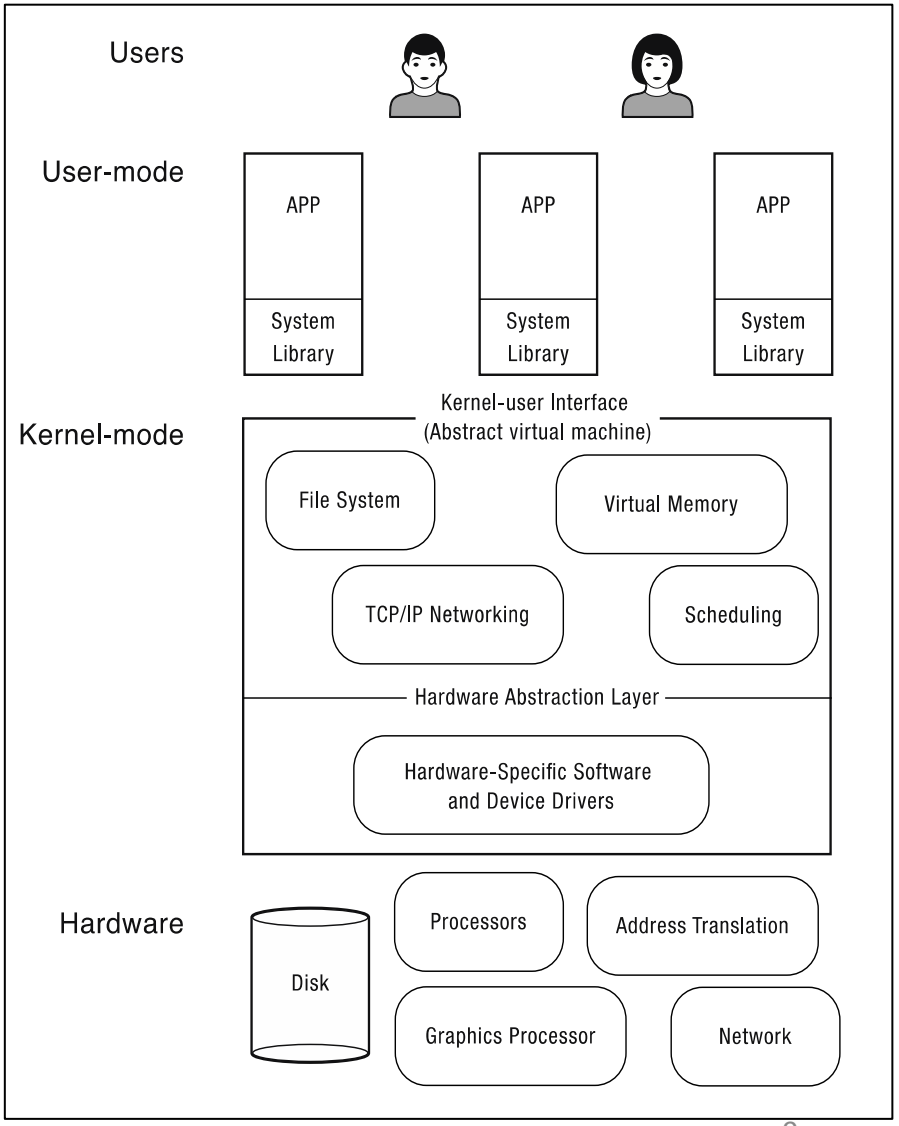
- How should an operating system allocate processing time between competing uses?
  - Give the CPU to the first to arrive?
  - To the one that needs the least resources to complete? To the one that needs the most resources?

OS Challenges

1. **Reliability**
   - Does the system do what it was designed to do?
2. **Availability**
   - What portion of the time is the system working?
   - Mean Time To Failure (MTTF), Mean Time to Repair
3. **Security**
   - Can the system be compromised by an attacker?
4. **Privacy**
   - Data is accessible only to authorized users

OS Challenges

- **Portability**
  - **For programs**:
    - ✓ Application programming interface (API)
    - ✓ Abstract virtual machine (AVM)
  - **For the operating system**
    - Hardware abstraction layer



OS Challenges

- **Performance**
  - **Latency/response time**
    - ✓How long does an operation take to complete?
  - **Throughput**
    - ✓How many operations can be done per unit of time?
  - **Overhead**
    - ✓How much extra work is done by the OS?
  - **Fairness**
    - ✓How equal is the performance received by different users?
  - **Predictability**
    - ✓How consistent is the performance over time?

Computer Performance Over Time

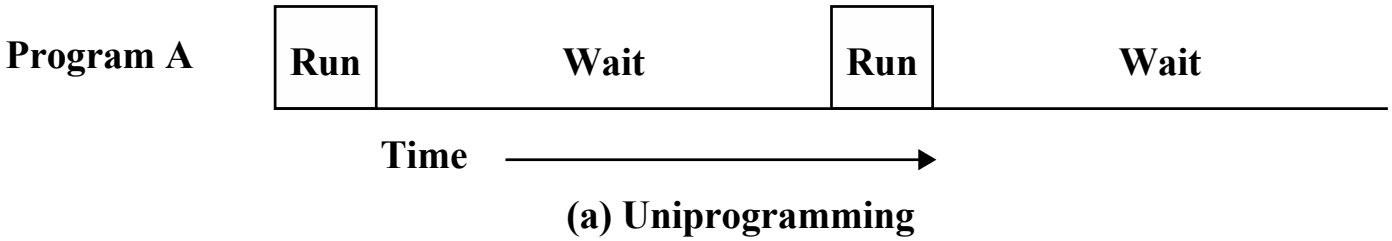
	1981	1997	2014	Factor (2014/1981)
Uniprocessor speed (MIPS)	1	200	2500	2.5K
CPUs per computer	1	1	10+	10+
Processor MIPS/$	$100K	$25	$0.20	500K
DRAM Capacity (MiB)/$	0.002	2	1K	500K
Disk Capacity (GiB)/$	0.003	7	25K	10M
Home Internet	300 bps	256 Kbps	20 Mbps	100K
Machine room network	10 Mbps (shared)	100 Mbps (switched)	10 Gbps (switched)	1000
Ratio of users to computers	100:1	1:1	1:several	100+

Early Operating Systems: Computers Very Expensive

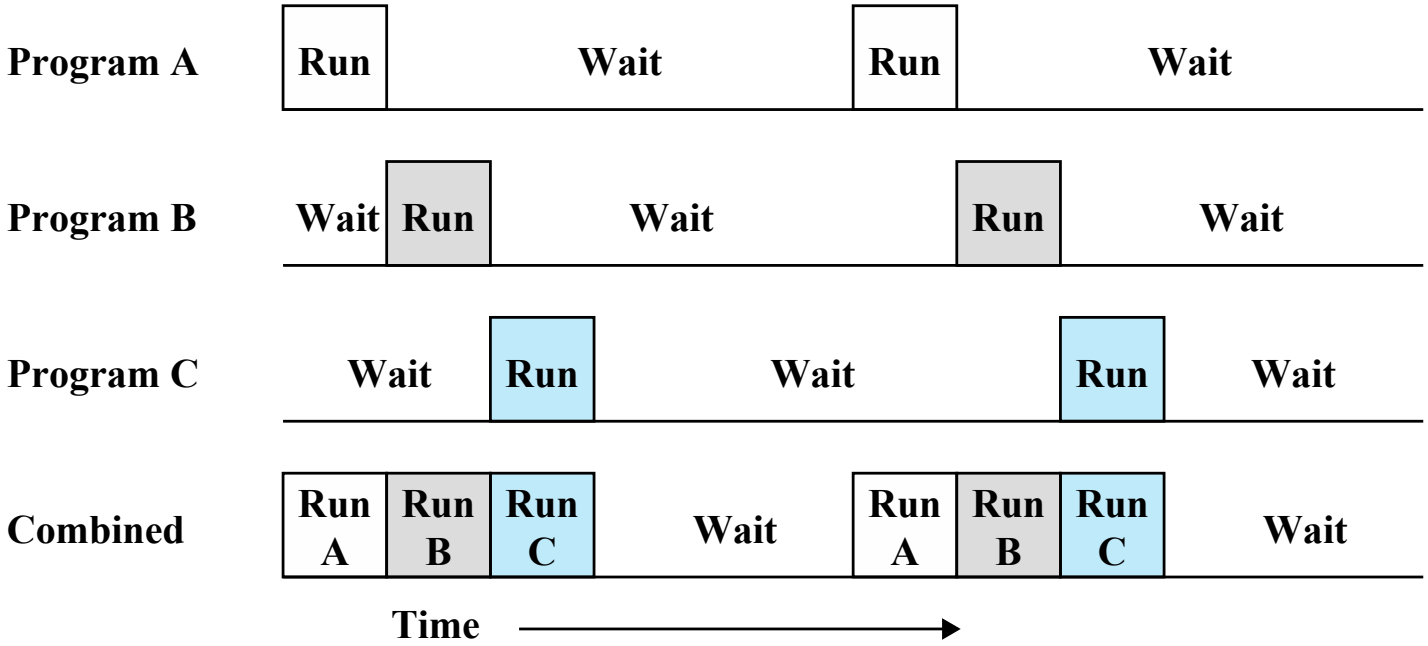
- One application at a time
  - Had complete control of hardware
  - OS was runtime library
  - Users would stand in line to use the computer
- Batch systems
  - Keep CPU busy by having a queue of jobs
  - OS would load next job while current one runs
  - Users would submit jobs, and wait, and wait, and

Time-Sharing Operating Systems: Computers and People Expensive

- Multiple users on computer at same time
  - Multiprogramming: run multiple programs at same time
  - Interactive performance: try to complete everyone's tasks quickly
  - As computers became cheaper, more important to optimize for user time, not computer time



(a) Uniprogramming



(c) Multiprogramming with three programs

Today's Operating Systems: Computers Cheap

- Smartphones
- Embedded systems
- Laptops
- Tablets
- Virtual machines
- Data center servers

Tomorrow's Operating Systems

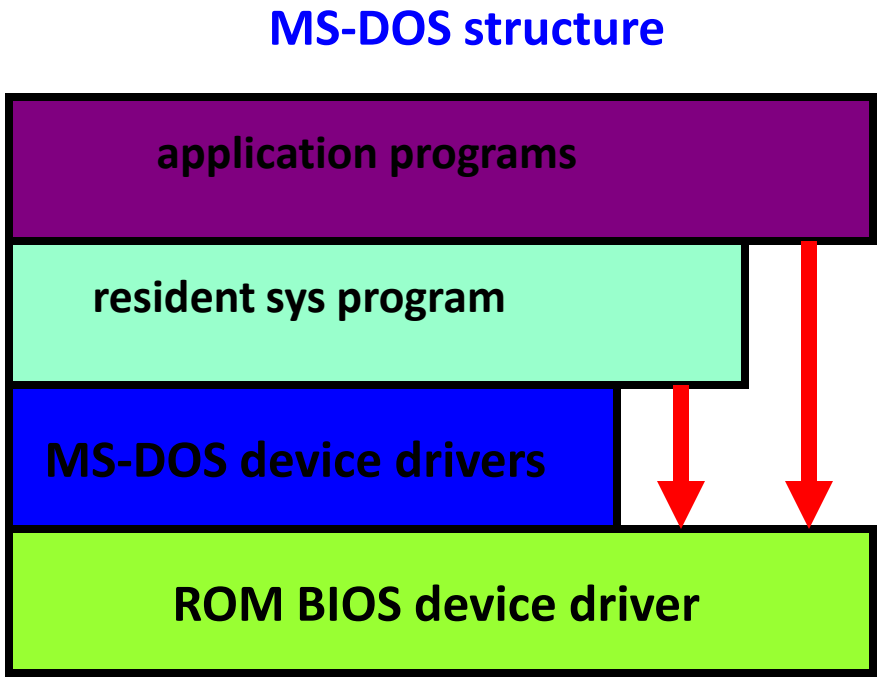
- Giant-scale data centers
- Increasing numbers of processors per computer
- Increasing numbers of computers per user
- Very large scale storage

Operating-System Structures

- Simple Structure
- Layered Approach
- Microkernels
- Modules

Simple Structure

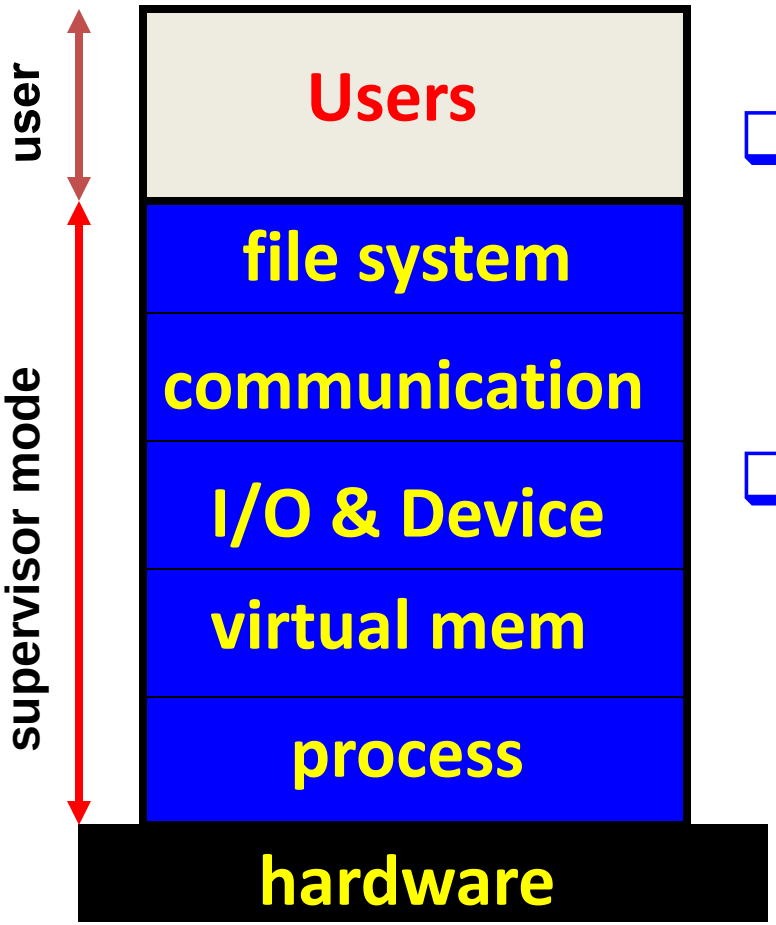
- Simple structure systems do not have well-defined structures
- Unix only had a limited structure: kernel and system programs
- Everything between the system call interface and physical hardware is the kernel.



Layered Approach: 1/4

- The operating system is broken up into a number of layers (or levels), each on top of lower layers.
- Each layer is an implementation of an abstract object that is the encapsulation of data and operations that can manipulate these data.
- The bottom layer (layer 0) is the hardware, and the highest one is the user interface.
- The main advantage of layered approach is *modularity*.

Layered Approach: 2/4



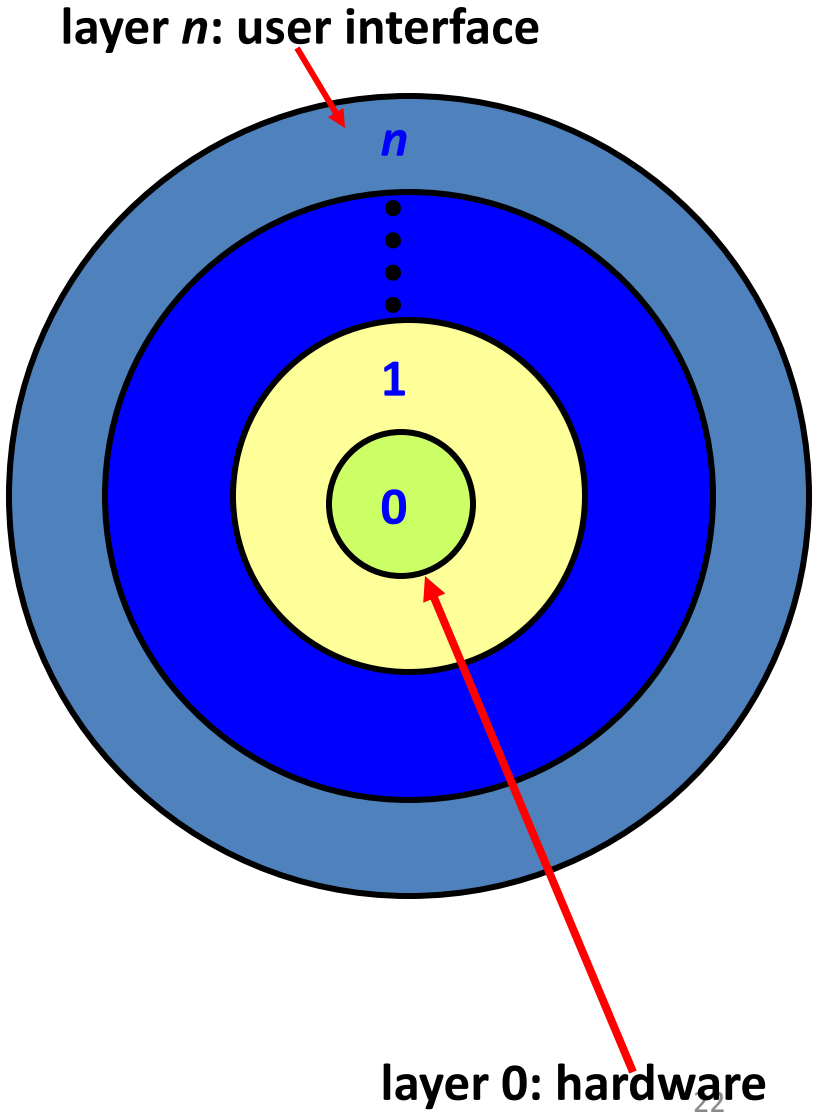
- The lowest layer is *process management*.
- Each layer *only uses the operations provided by lower layers* and does not have to know their implementation.
- Each layer hides the existence of certain data structures, operations and hardware from higher-level layers. *Think about OO.*

Layered Approach Probs: 3/4

- It is *difficult to organize* the system in layers, because a layer can use only layers below it. Example: virtual memory (lower layer) uses disk I/O (upper layer).
- Layered implementations tend to be *less efficient* than other types. Example: there may be too many calls going down the layers: user to I/O layer to memory layer to process scheduling layer.

Layered Approach Probs: 4|4

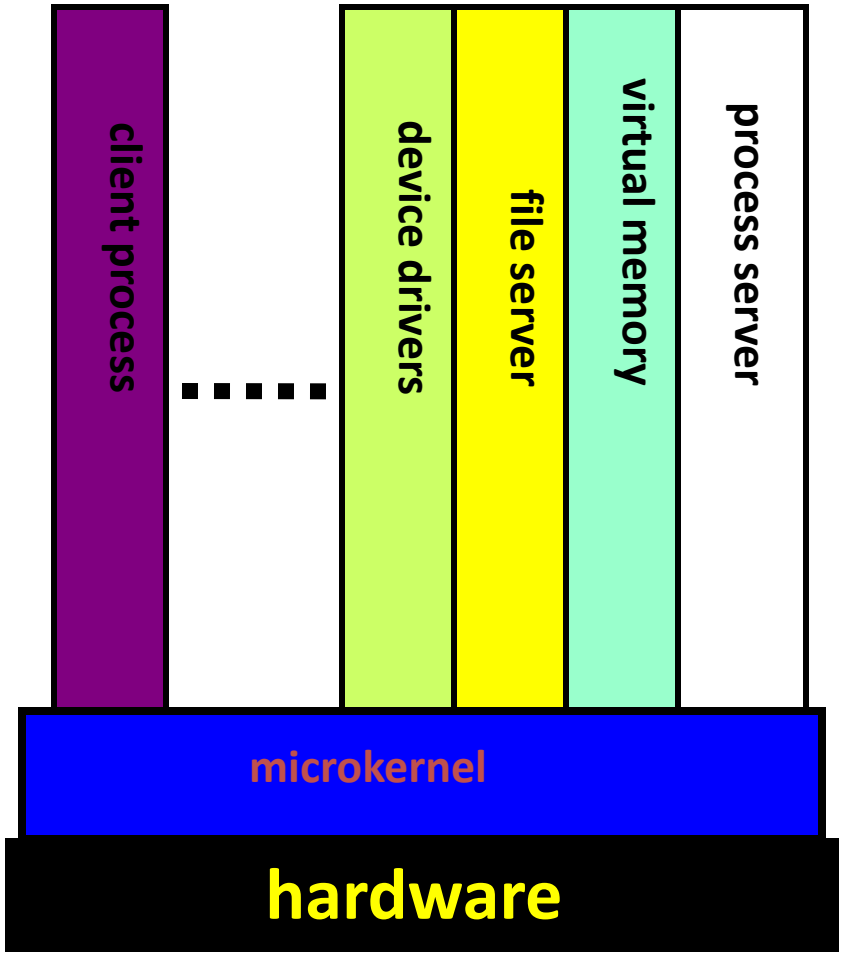
- The layered approach may also be represented as a set of concentric rings.
- The first OS based on the layered approach was THE, developed by E. Dijkstra.



Microkernels: 1/5

- **Only *absolutely essential core* OS functions should be in the kernel.**
- **Less essential services and applications are built on the kernel and *run in user mode*.**
- **Many functions that were in a traditional OS become external subsystems that interact with the kernel and with each other.**

Microkernels: 2/5



- The main function of the microkernel is to provide **communication** facility between the client programs and various services.
- Communication is provided by ***message passing***.

Microkernels: 3/5

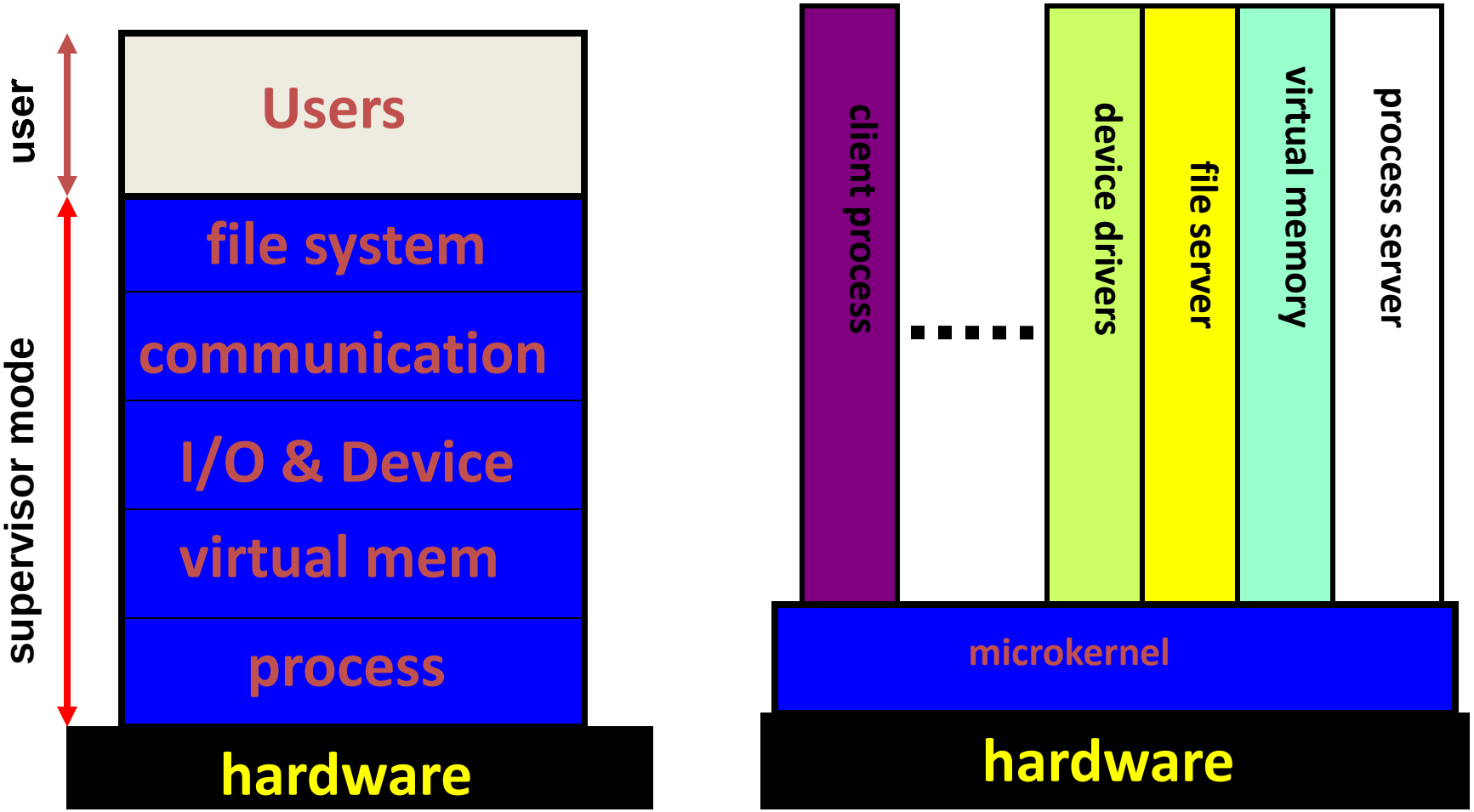
- **Uniform interfaces: message passing**
- **Extensibility: adding new services is easy**
- **Flexibility: existing services can be taken out easily to produce a smaller and more efficient implementation**
- **Portability: all or at least most of the processor specific code is in the small kernel.**
- **Reliability: A small and modular designed kernel can be tested easily**
- **Distributed system support: client and service processes can run on networked systems.**

Microkernels: 4/5

- **But, microkernels do have a problem:**
  - **As the number of system functions increases, overhead increases and performance reduces.**
  - **Most microkernel systems took a hybrid approach, a combination of microkernel and something else (*e.g.*, layered).**

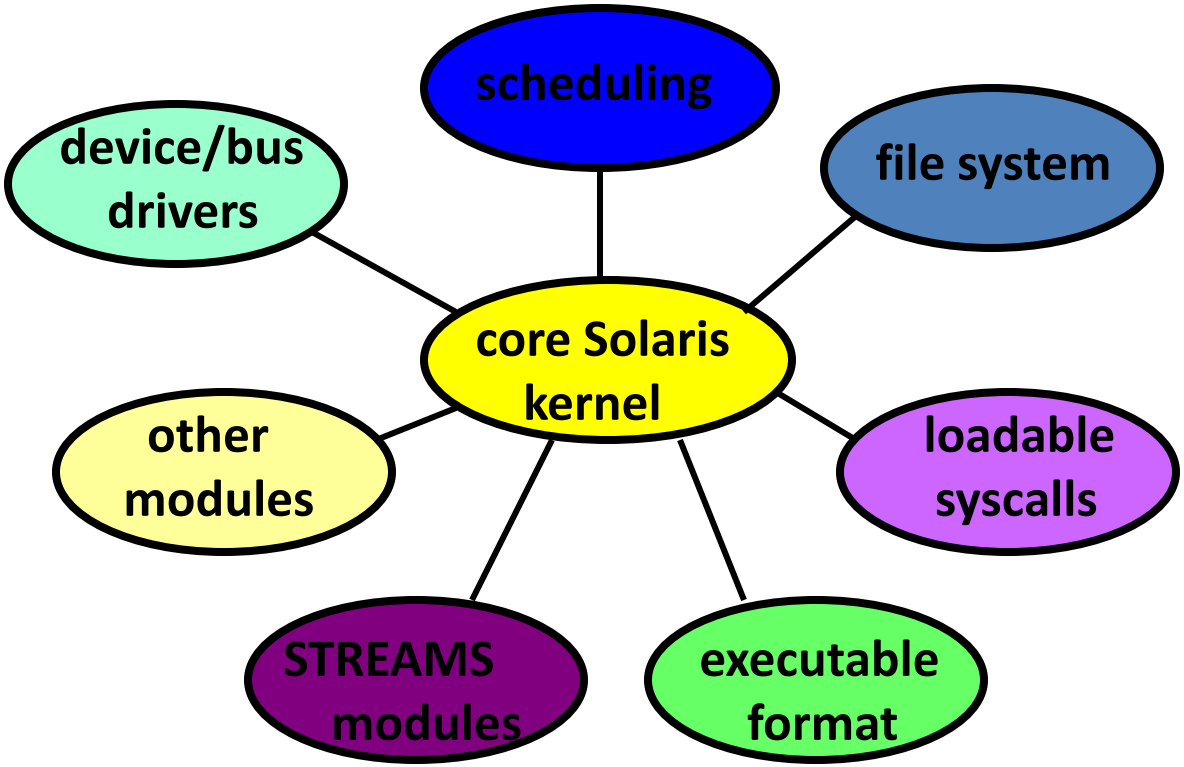
Microkernels: 5/5

Microkernel vs. Layered Approach



Modules: 1/2

- The OO technology may be used to create a modular kernel.
- The kernel has a set of core components and dynamically links in additional services either during boot time or during run time.



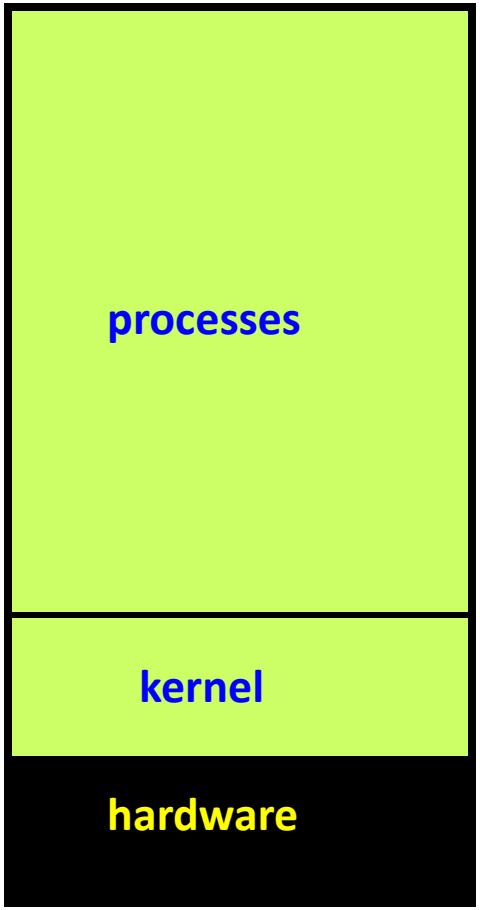
Module: 2/2

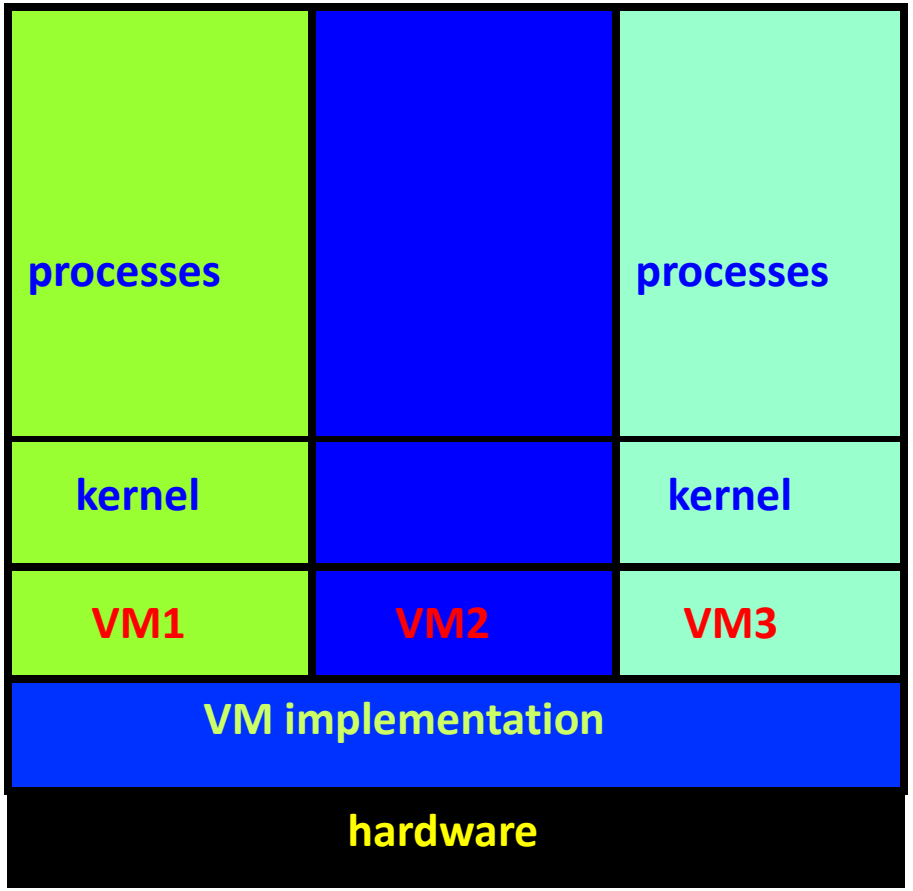
- The module approach looks like the layered approach as each module has a precise definition; however, the module approach is more flexible in that any module can call any other module.
- The module approach is also like the microkernel approach because the core module only includes the core functions. However, the module approach is more efficient because no message passing is required.

Virtual Machines: 1/5

- A virtual machine, VM, is a software between the kernel and hardware.
- A VM provides all functionalities of a CPU with software simulation.
- A user has the illusion that s/he has a real processor that can run a kernel.

Virtual Machines: 2/5





Virtual Machines: 3/5

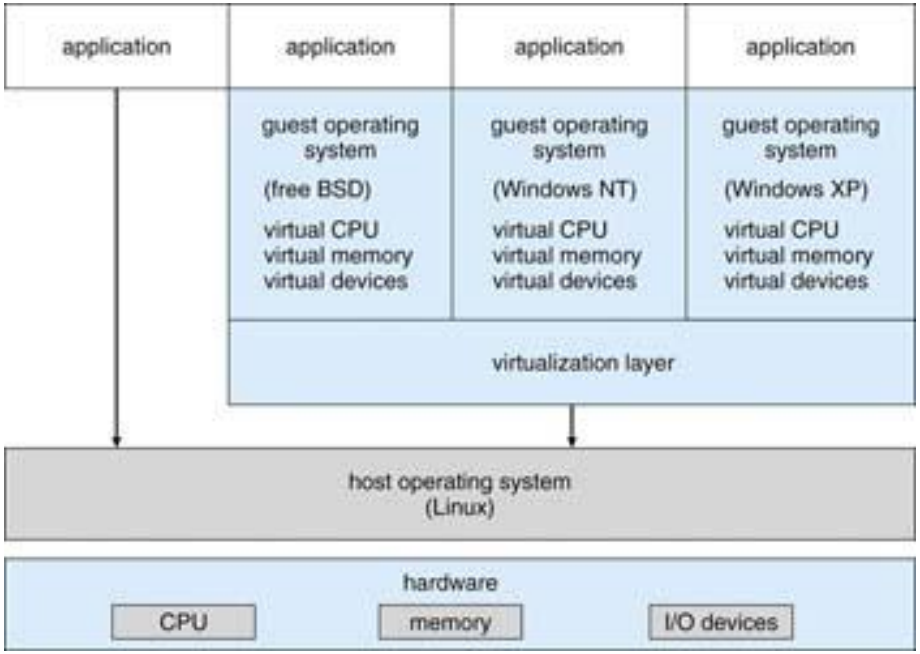
- **Self-Virtualized VM: the VM is identical to the hardware. Example: IBM's VM/370 and VMware (creating a VM under Linux to run Windows).**
- **Non-Self-Virtualized VM: the VM is not identical to the hardware. Example: Java Virtual Machine JVM and SoftWindow.**
- **It can be proved that all third-generation CPUs can be virtualized.**

Virtual Machines: 4/5

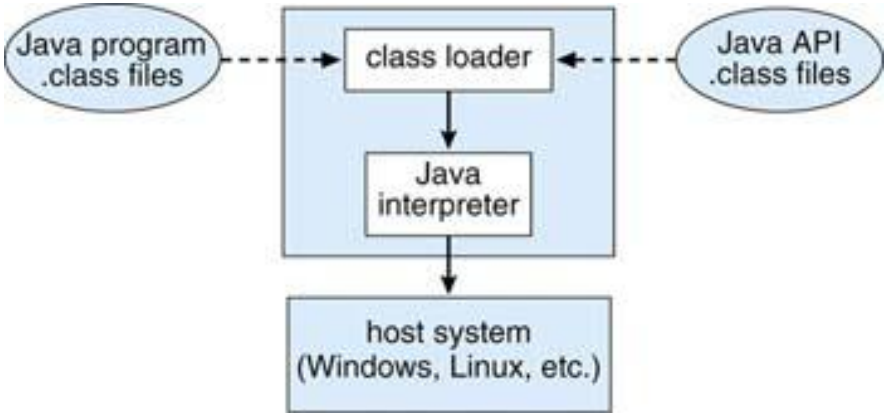
- VM's are difficult to implement because they must duplicate all hardware functions.
- Benefits:
  - VM provides a robust level of security
  - VM permits system development to be done without disrupting normal system operation.
  - VM allows multiple operating systems to run on the same machine at the same time.
  - VM can make system transition/upgrade much easier

Virtual Machines: 5/5

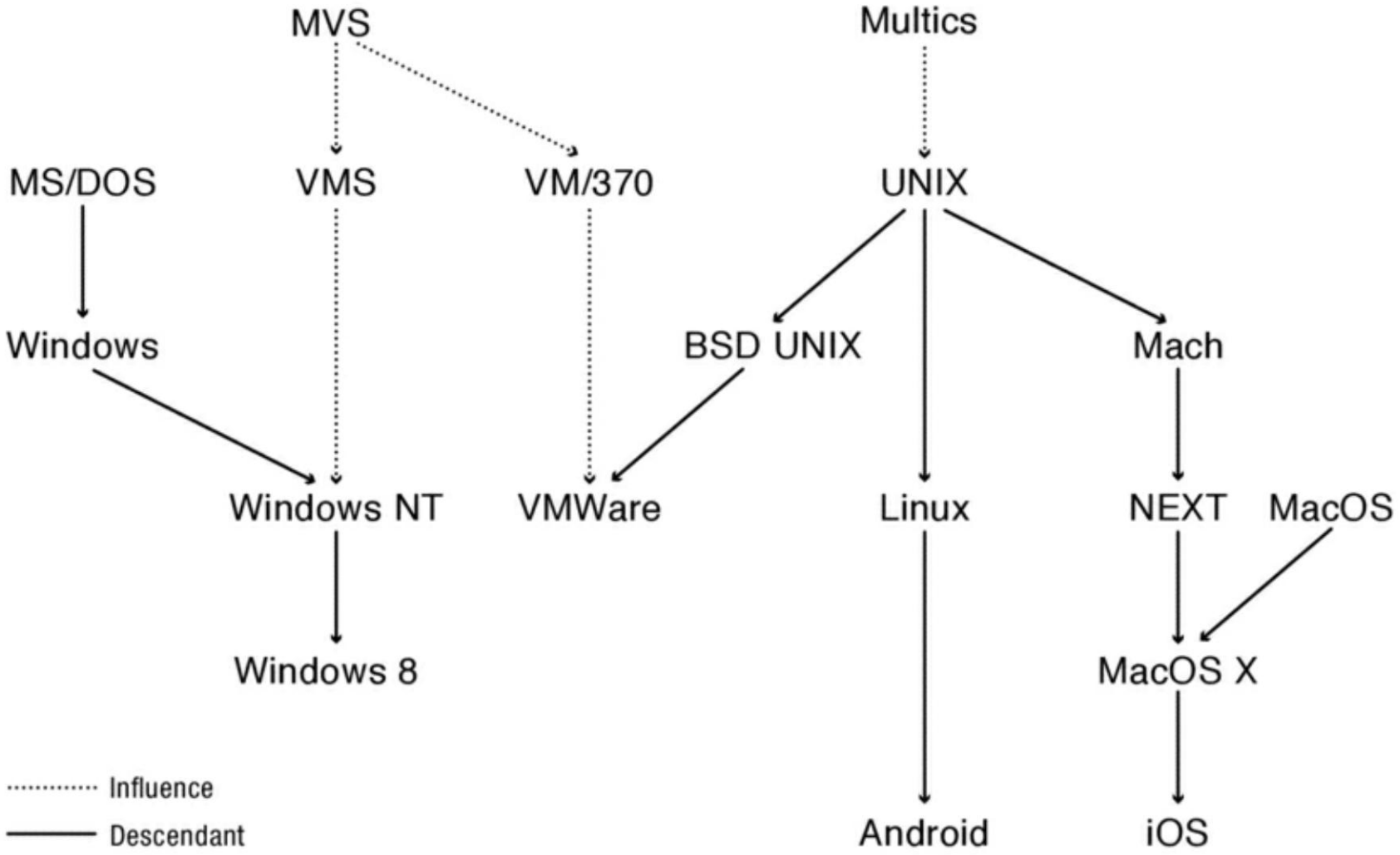
VMware



Java VM



Genealogy of Several Modern Operating Systems



The End