Part I Overview

Chapter 2: Operating System Structures

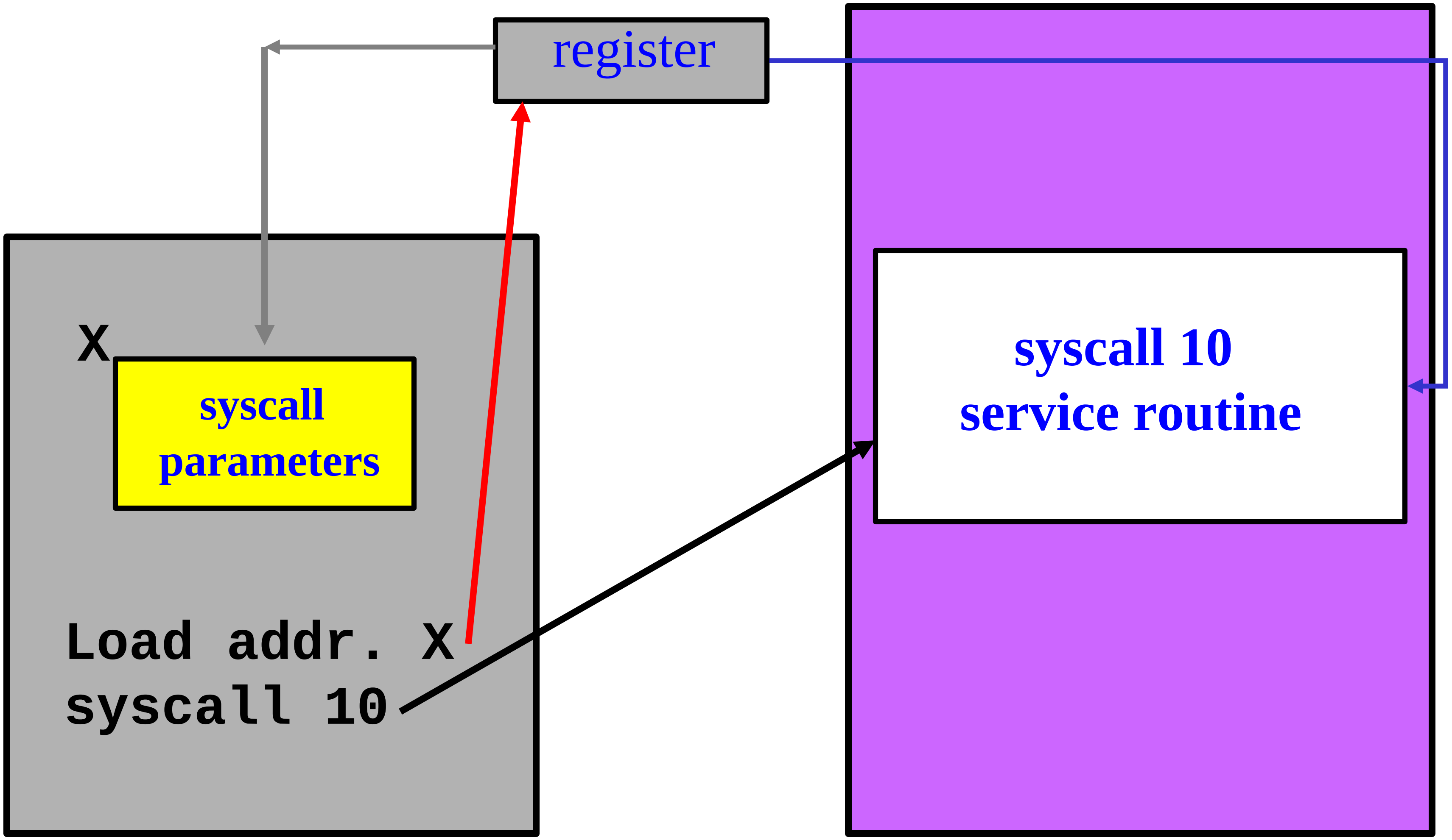
Operating System Services

- **User Interface (*e.g.*, command processor –shell and GUI – Graphical User Interface)**
- **Program execution**
- **I/O operations**
- **File-system manipulation**
- **Communications**
- **Error detection**
- **Resource allocation**
- **Accounting**
- **Protection**

System Calls

- System calls provide an interface to the services made available by an operating system.
- Type of system calls:
  - ❑Process control (e.g., create and destroy processes)
  - ❑File management (e.g., open and close files)
  - ❑Device management (e.g., read and write operations)
  - ❑Information maintenance (e.g., get time or date)
  - ❑Communication (e.g., send and receive messages)

System Call Mechanism: 1/2



register

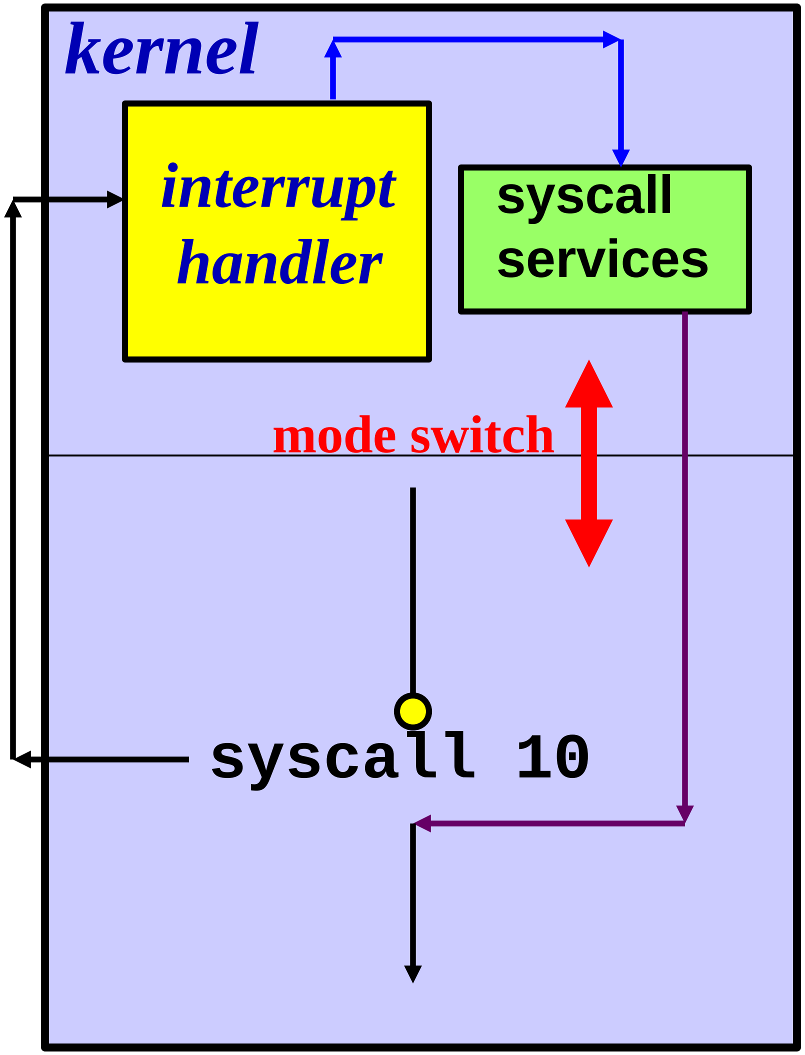
X

syscall parameters

```
Load addr. X
syscall 10
```

syscall 10 service routine

System Call Mechanism: 2/2



- A system call generates am interrupt, actually a *trap*.
- The executing program is suspended.
- Control is transferred to the OS.
- Program continues when the system call service completes.

System Programs

- **File management (*e.g.*, create, delete files)**
- **Status management (*e.g.*, date/time, available memory and disk space)**
- **File modification (*e.g.*, editors)**
- **Programming language support (*e.g.*, assemblers, compilers, interpreters)**
- **Program loading and execution (*e.g.*, loaders, linkage editors)**
- **Communications (*e.g.*, connections between processes, sending/receiving e-mails)**

Operating-System Design and Implementation: 1/5

- Design Goals
- Mechanisms and Polices
- Implementation

Operating-System Design and Implementation: 2/5

- **Design Goals**
  - ❑ **Goals and specifications will be affected by hardware and the type of the system (*e.g.*, batch, time-sharing, real-time, etc)**
  - ❑ **Requirements: *user* goals and *system* goals.**
    - ❖ **User goals: easy of use, reliable, safe, fast**
    - ❖ **System goals: easy to design, implement and maintain, and flexible, reliable, efficient, etc.**
  - ❑ **There is no unique way to achieve all goals, and some compromises must be taken.**

Operating-System Design and Implementation: 3/5

- **Mechanisms and Policies: 1/2**
  - ❑ ***Mechanisms*** **determine how to do something.**
  - ❑ ***Policies*** **determine what will be done.**
  - ❑ **Policies and mechanisms are usually separated:** ***the separation of mechanism and policy principle*****.**

Operating-System Design and Implementation: 4/5

- **Mechanisms and Policies: 2/2**
  - **The separation of mechanism and policy:**
    - **It is for *efficiency* purpose. Policies are likely to change over time.**
    - **If a more general mechanism is available, the impact of changing policy on mechanism is reduced.**
    - **Microkernel-based systems implement a basic set of building blocks, which are almost policy free. Thus, advanced policies and mechanisms can be added to user-created kernel modules.**

Operating-System Design and Implementation: 5/5

- **Implementation**
  - ❑ **The assembly vs. high-level language issue**
    - ❖ **Advantages of high-level languages: easy to use and maintain, and more available programming tools**
    - ❖ **Disadvantages of high-level languages: slower and more space**
  - ❑ **Many systems were written in both with a small percentage (*e.g.*, 10%) of assembly language for the most speed demanding low level portion.**
  - ❑ **Better data structures and algorithms are more important than tweaking assembly instructions.**

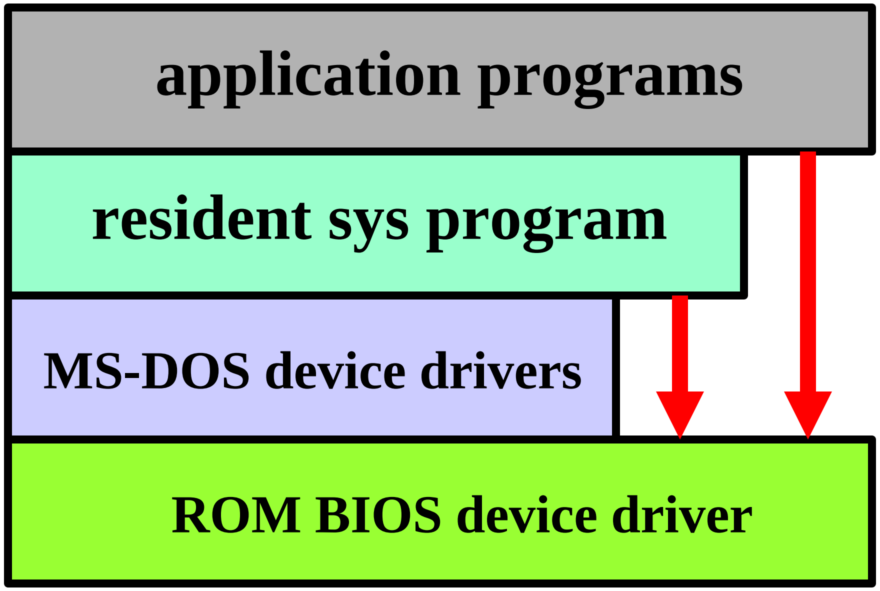
Operating-System Structures

- **Simple Structure**
- **Layered Approach**
- **Microkernels**
- **Modules**

Simple Structure

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

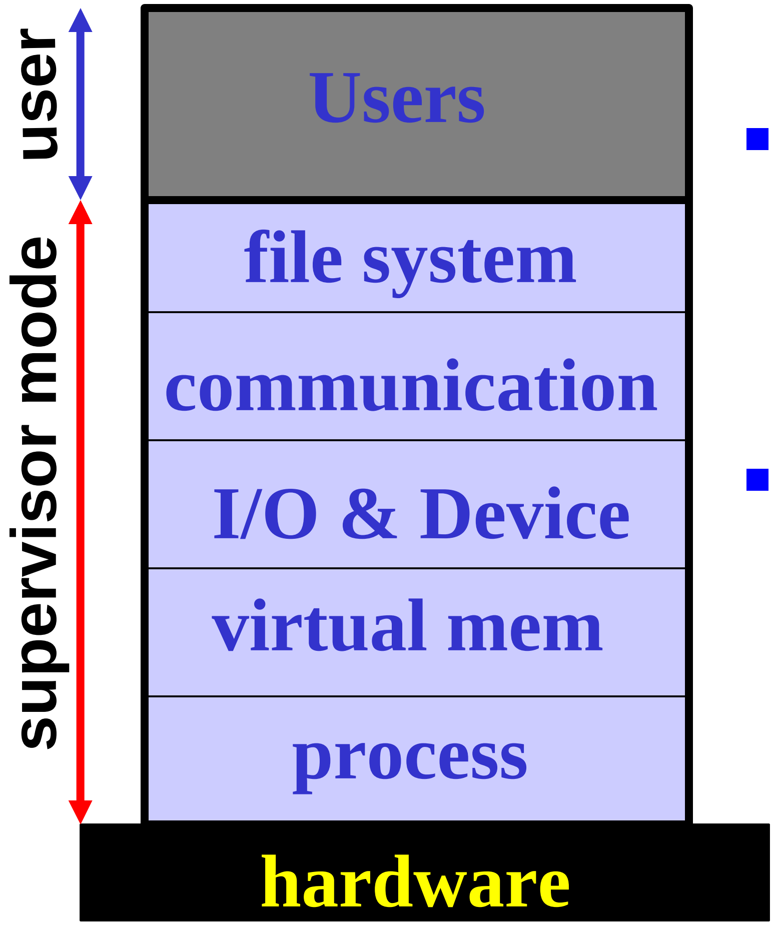
MS-DOS structure



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.**
- **The main advantage of layered approach is *modularity*.**

Layered Approach: 2/4



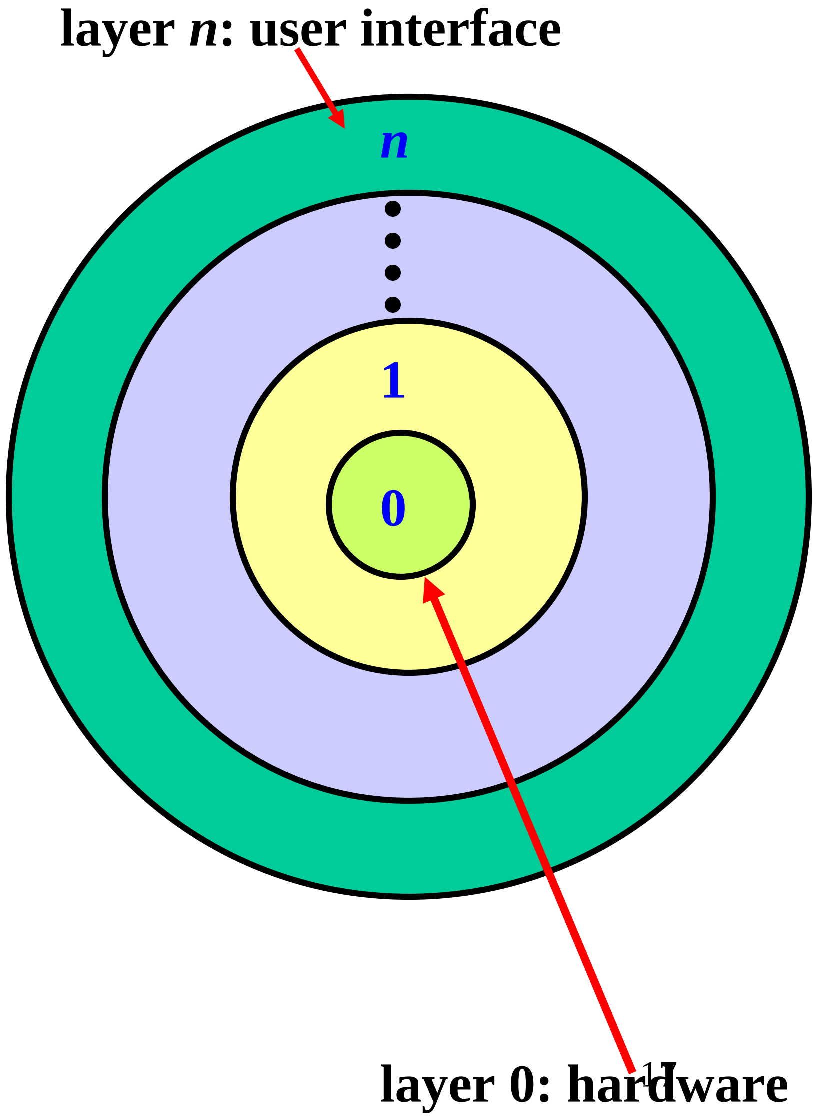
- The lowest layer is ***process management***.
- Each layer ***only uses the operations provided by lower layers*** and does not have to know their implementation.
- Each layer hides the existence of certain data structures, operations and hardware from higher-level layers. ***Think about OO.***

Layered Approach Problems: 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 Problems: 4/4

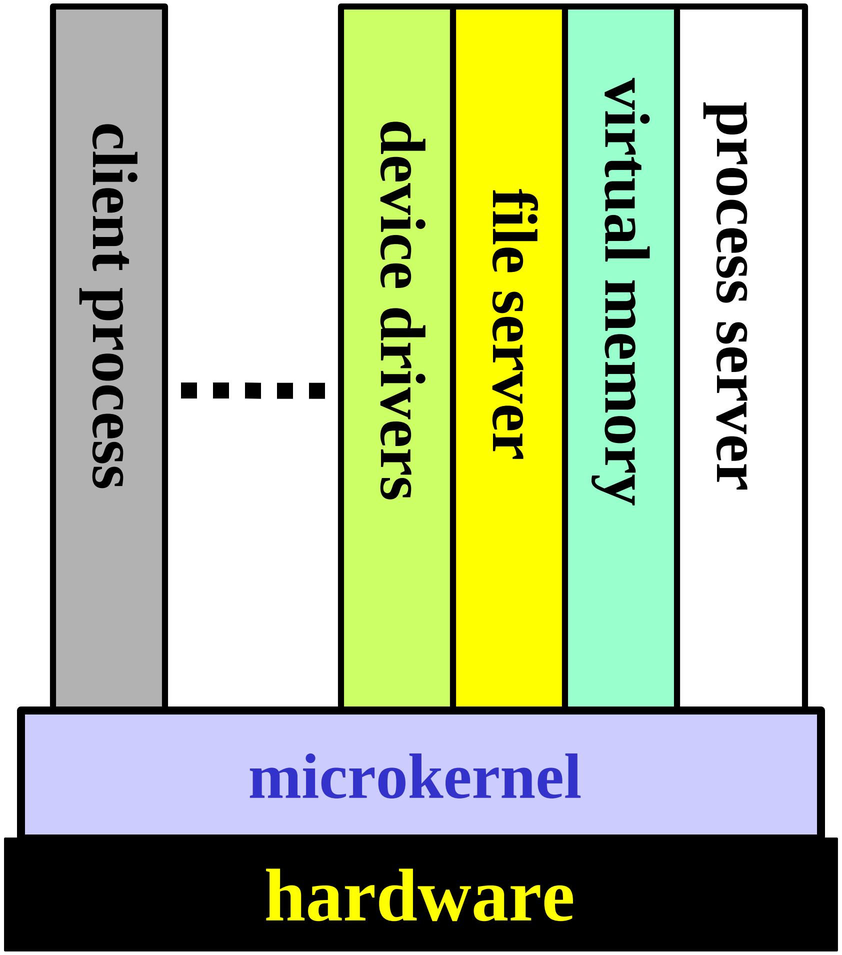
- The layered approach may also represented as a set of concentric rings.
- The first OS based on the layered approach was THE, developed by E. Dijkstra.



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 program and various services.
- Communication is provided by *message passing*.

Microkernels: 3/5

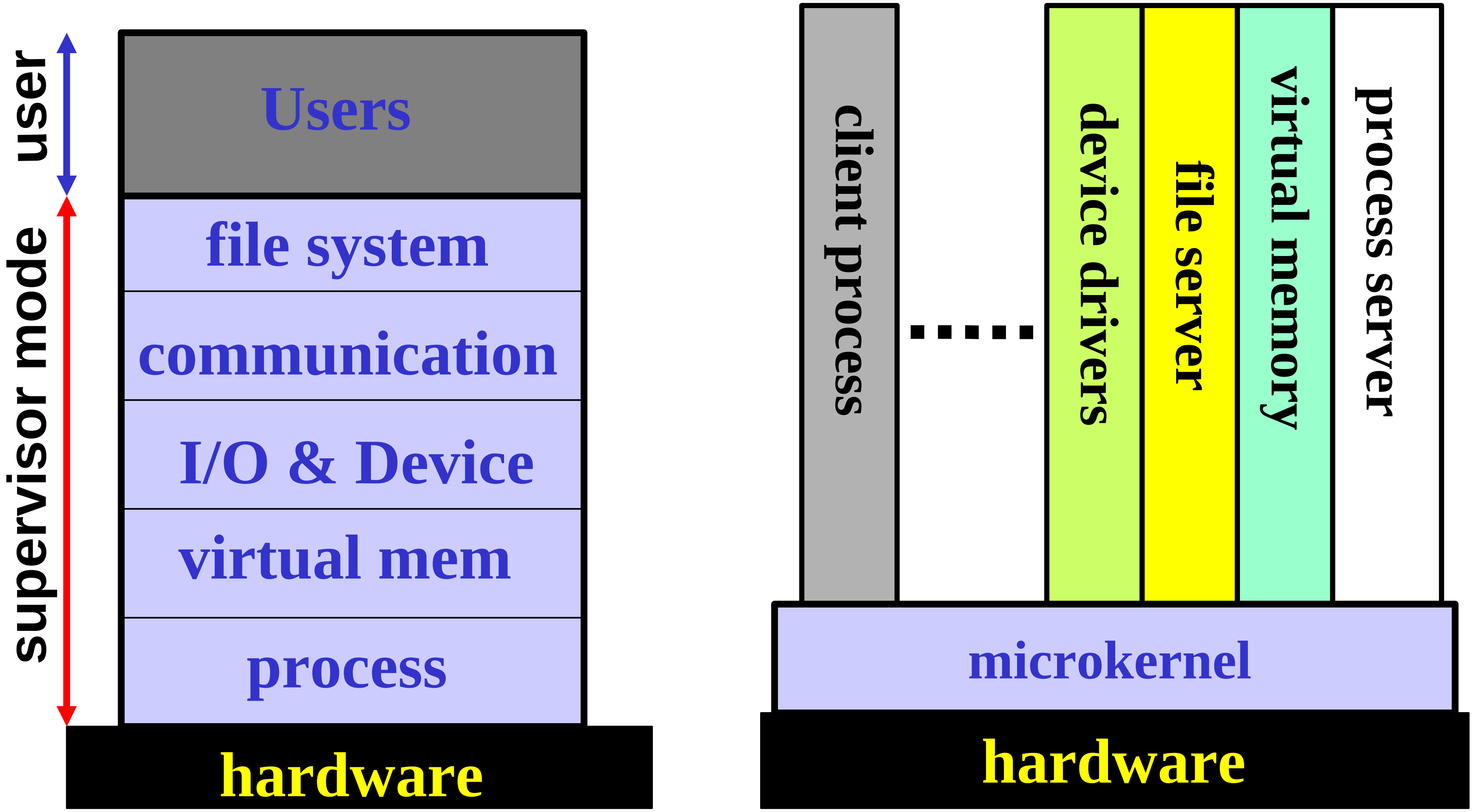
- Uniform interfaces: message passing
- Extensibility: adding new services is easy
- Flexibility: existing services can be taken out easily to produce a smaller and more efficient implementation
- Portability: all or at least much of the processor specific code is in the small kernel.
- Reliability: 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).**

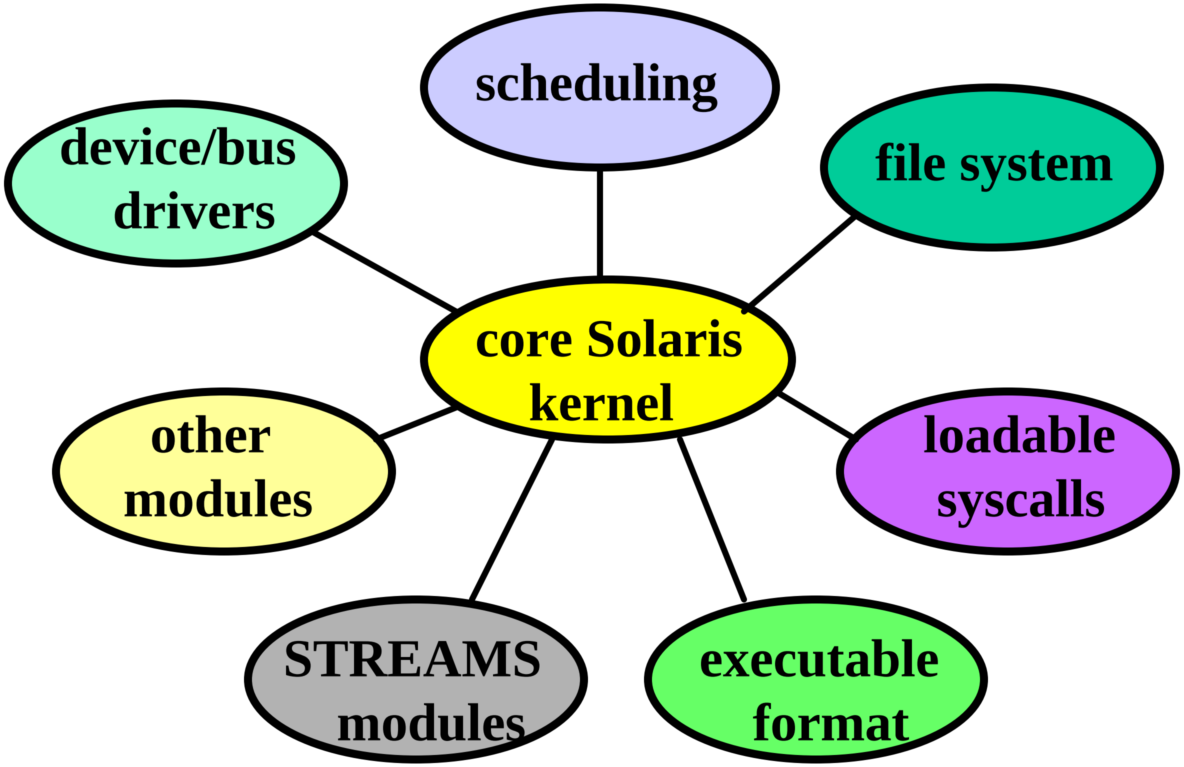
Microkernels: 5/5

Microkernel vs. Layered Approach



Modules: 1/2

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



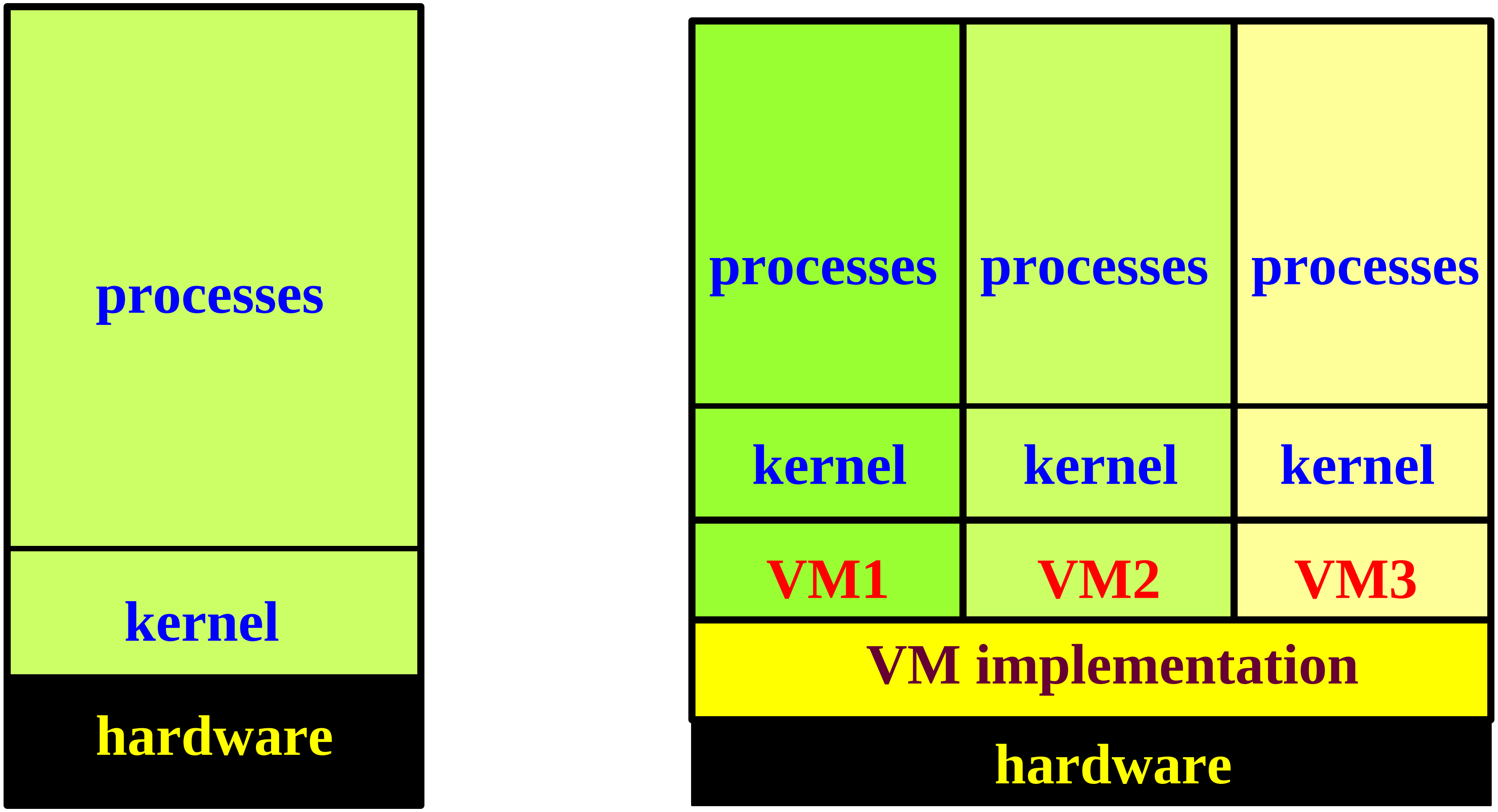
Module: 2/2

- The module approach looks like the layered approach as each module has a precise definition; however, the former 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: 1/4

- **A virtual machine, VM, is a software between the kernel and hardware.**
- **Thus, 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: 2/4



Virtual Machines: 3/4

- **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: 4/4

- **VM 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**

Bootstrap or IPL

- **When the system is powered on, a small program, usually stored in ROM (*i.e.*, read only memory), will be executed. This initial program is usually referred to as a *bootstrap program* or an *Initial Program Loader* (IPL).**
- **The bootstrap program reads in and initializes the operating system. The execution is then transferred to the OS**