

Operating System Security and Memory Protection

Vítor Francisco Fonte, vff@di.uminho.pt, University of Minho, 2025

Operating Systems

- Important security goals:
 - *separation and access control to resources*
 - first line of defence against unwanted behaviour
 - fundamental controller of all system resources
 - compromising it means loss of control to resources
- Important functions:
 - access control to resources
 - identification, authentication and credential management
 - information flow and synchronisation
 - audit and integrity protection

Operating Systems

- General boot sequence:
 1. primitive functions and device drivers
 2. process controllers
 3. file and memory management
 4. services
 5. user interface

Operating Systems

- Protection from malware:
 - must be running before an attack takes place
 - but often an add-on, subject to delayed initialisation
-
- If a malware manages to exploit an OS vulnerability:
 - run undetected and as privileged user (e.g rootkit)

Operating Systems

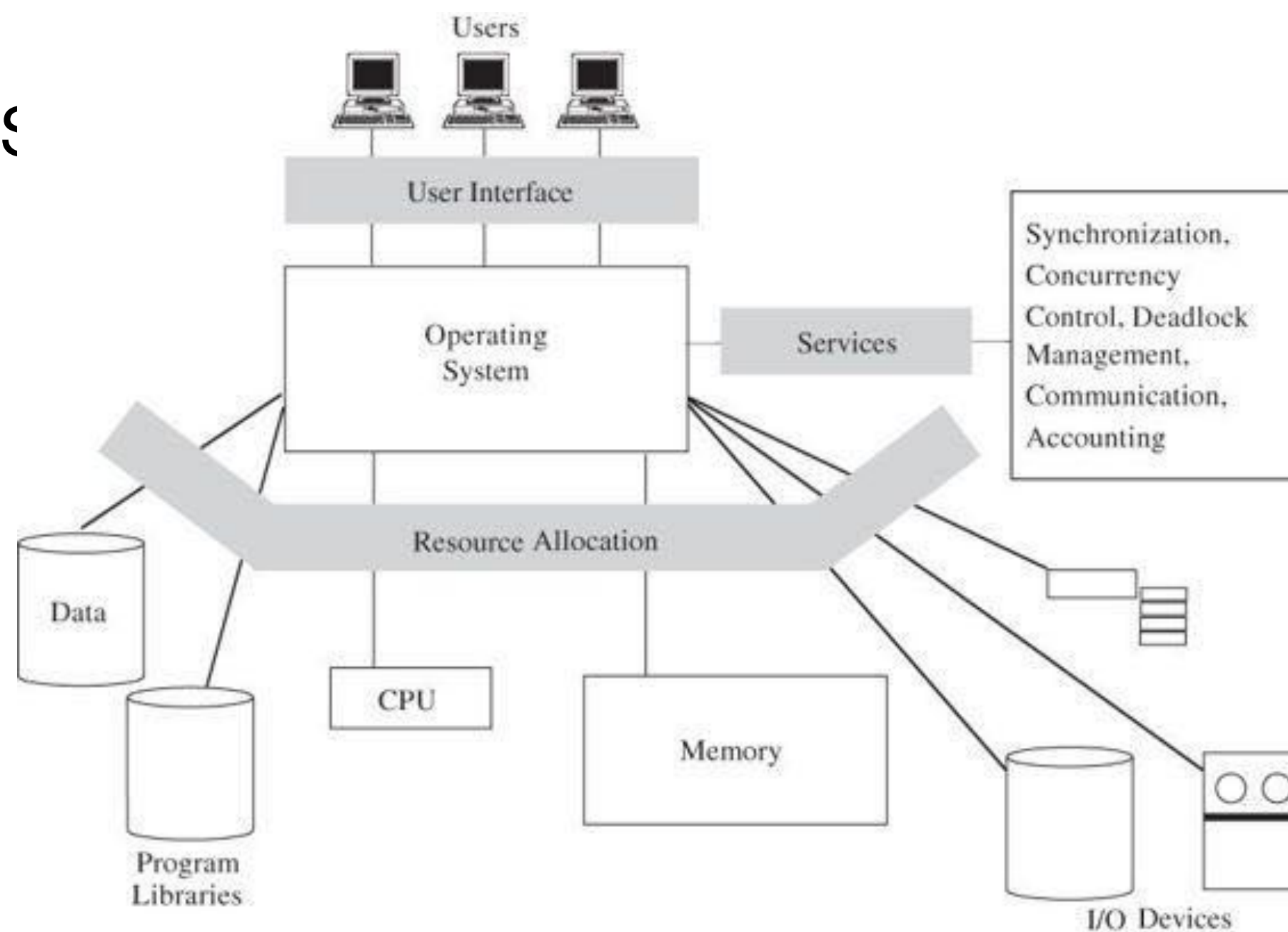
- Different hardware:
 - personal computers and mainframes
 - dedicated devices
 - automobiles and avionics
 - smartphones, tablets, web appliances
 - network appliances
- Adequate fit to the:
 - complexity of the device
 - degree of control it must exercise
 - amount of interaction to support (humans, devices)

Operating Systems

- Characteristics:
 - single- vs. multi-user
 - single- vs. multi-program (cooperative scheduling or on I/O)
 - single- vs. multi-task (preemptive scheduling)
 - single- vs. multi-threaded
- From the security standpoint:
 - interest in the OS's control of resources
 - which users are allowed which access to which objects

Operating Systems

- Operating System



Primitive OS functions:

- aka kernel functions
- base for enforcing security and other higher-level OS functions

Some primitive OS functions:

- enforced sharing
- interprocess communication and synchronisation
- protection of critical OS data
- guaranteed fair service
- interface to hardware
- user authentication
- memory protection
- file and I/O access control
- allocation and access control to general objects

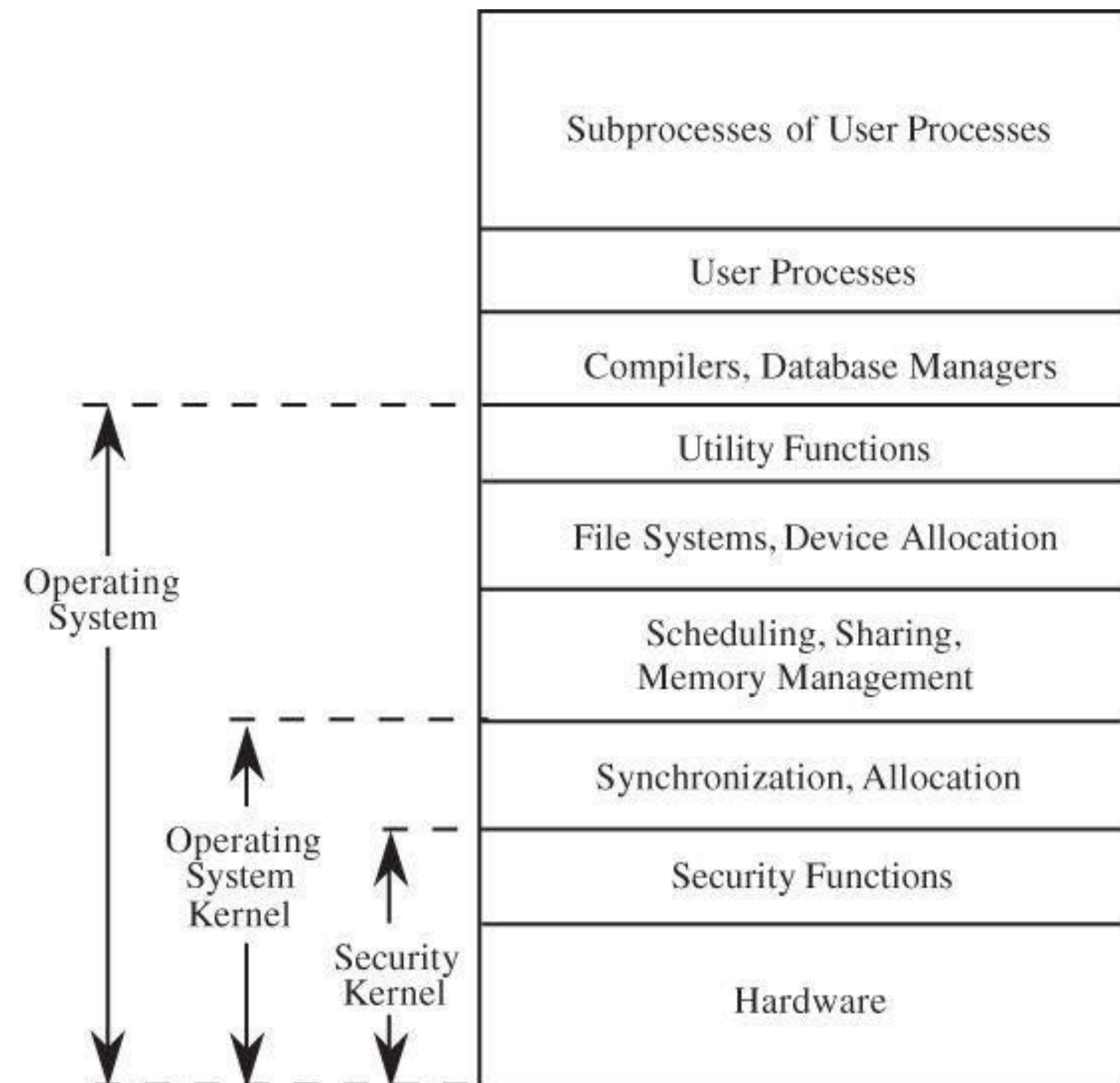
Operating System

Resource Protection

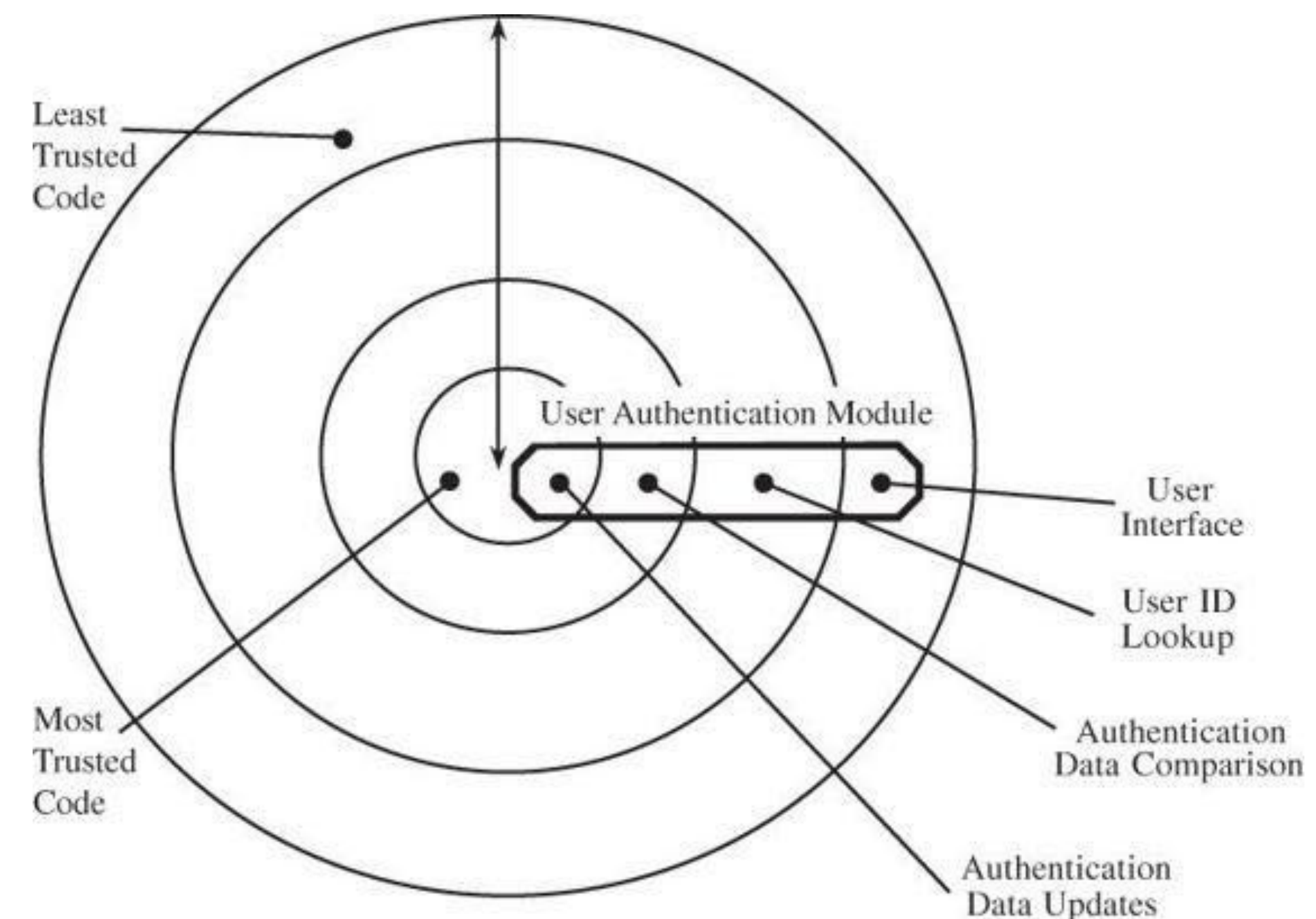
- Protected Objects:
 - memory
 - sharable I/O devices (e.g disk)
 - serially reusable I/O devices (e.g printer)
 - sharable programs and libraries
 - networks
 - sharable data
- But:
 - how to protect these objects?
 - how to tackle the need for sharing some of these objects between users?
 - and at which resource granularity?

Operating Systems As a Layered System

- Layered Operating System



Functions span across layers



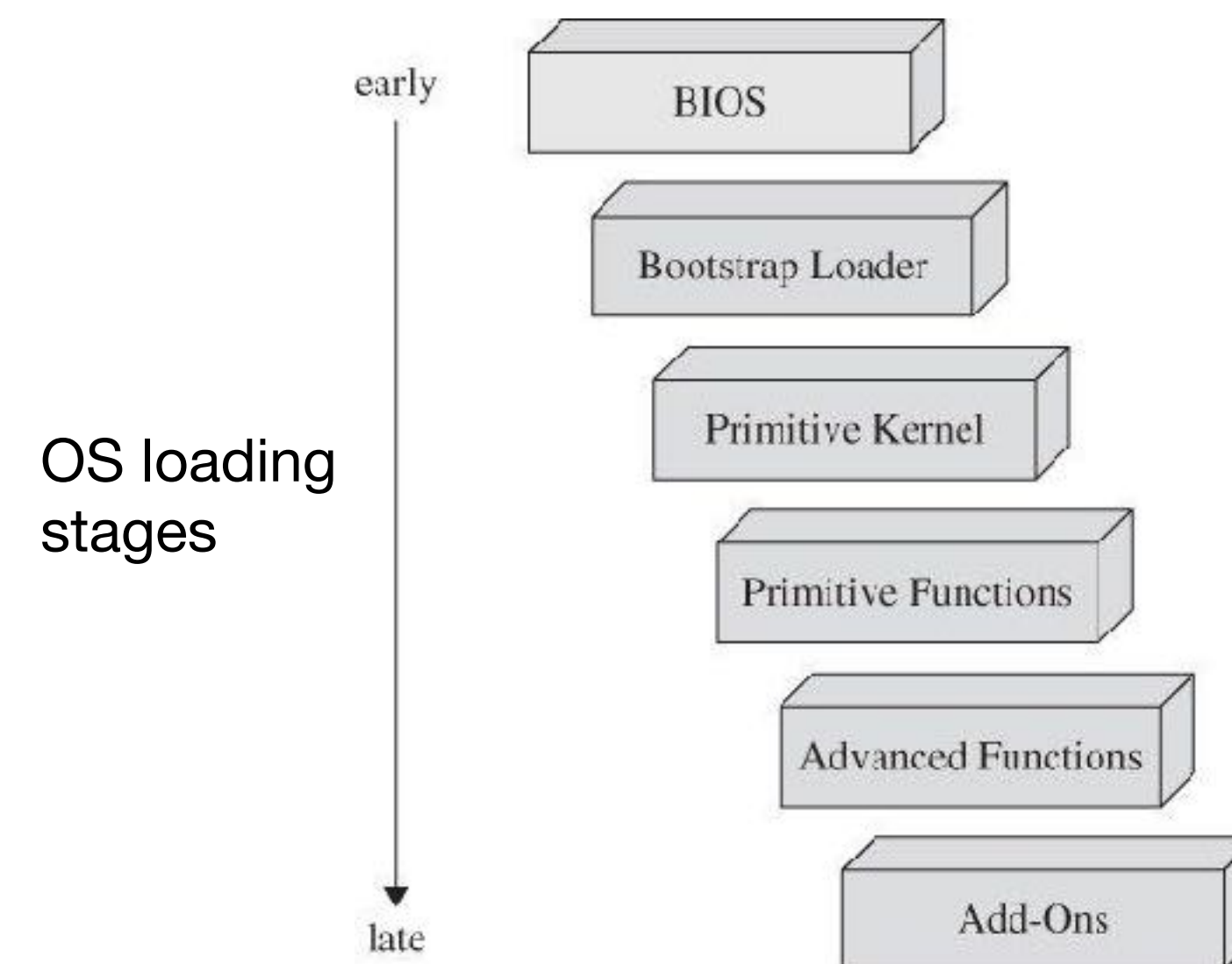
Operating Systems

Design for Self-Protection

- An OS must be designed for self-protection
- Against compromise from:
 - malicious user programs
 - incorporated modules
- Challenges:
 - timing, coordination, hand-off
 - changing requirements and functionality
 - compatibility, consistency

Limited knowledge:

- who can it trust
- for what capabilities



Operating Systems

Tools to Implement Security

- Access control paradigm:
 - Reference monitor: a subject is permitted to access an object in a particular mode, and only such authorised accessed is allowed
 - Access control techniques: access control list (ACL), privilege list, and capabilities
 - OS needs to implement both the underlying tables supporting access control and the mechanism that checks for acceptable uses

Operating Systems

Tools to Implement Security

- Audit:
 - log: who, what, when, how
 - tool for reacting after a security breach (not preventing it)
 - what information was compromised, by whom and when
 - help in preventing future incidents
 - adequate level of logging
 - useful only if can be analysed (info overload)

Operating Systems

Tools to Implement Security

- Virtualisation:
 - availability of one set of resources by using a different set
 - only the set of resources the user is entitled to access
 - e.g. virtual memory
 - e.g. virtual machine, sandbox, honeypot
 - useful for flexible resource allocation

Operating Systems

Tools to Implement Security

- Hypervisor
 - virtual machine monitor
 - implements a virtual machine
 - receives all user access requests
 - passes along those that apply to real resources the user is allowed to access
 - redirects other requests to the virtualised resources
 - separation and overlap of resources
 - multiple operating systems and multiple hardware

Operating Systems

Tools to Implement Security

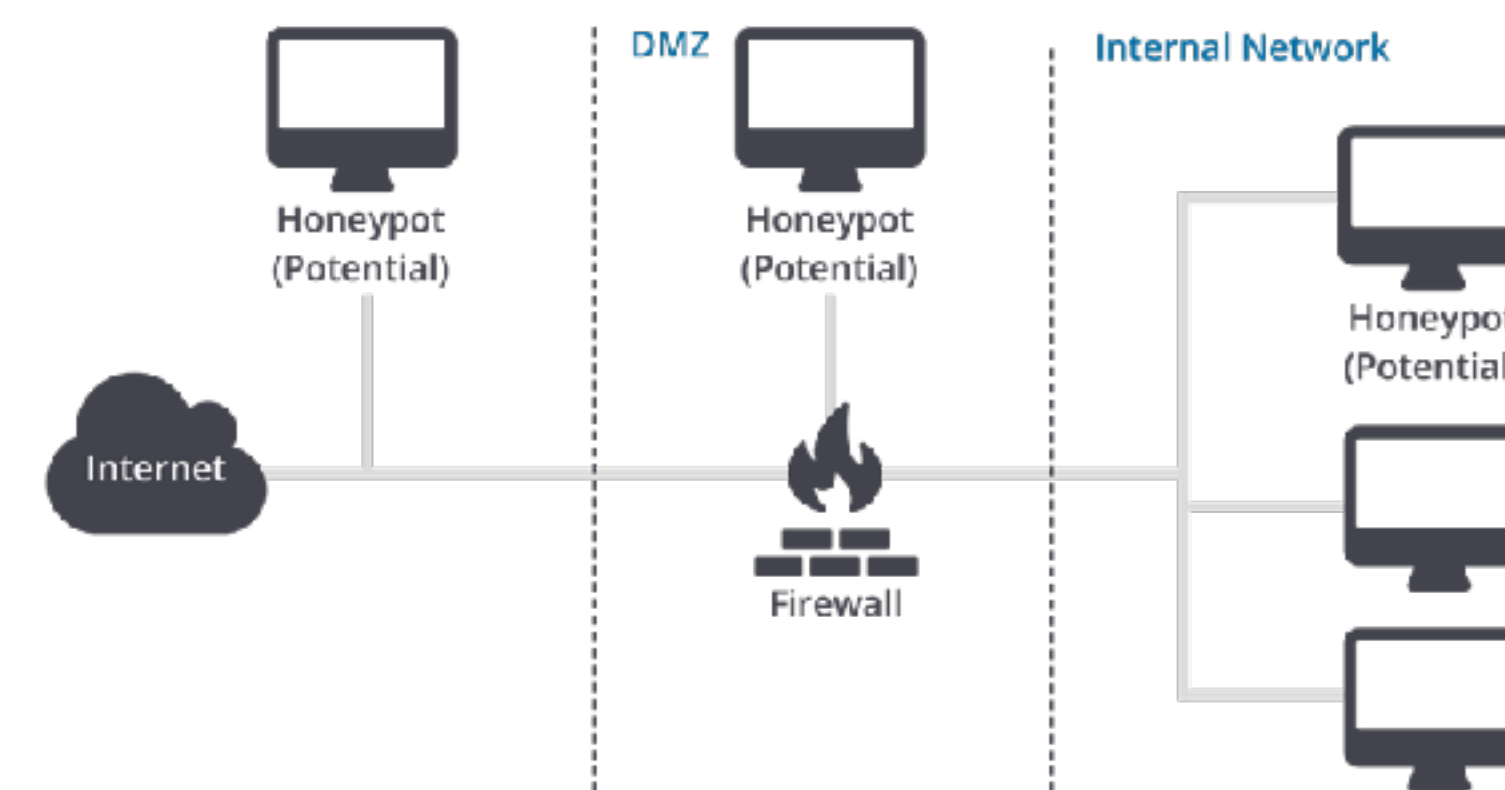
- Sandbox
- protected execution environment
- isolated from each other
- often virtualised resources
- e.g. Docker containers
 - each container has its own software, libraries and configurations
 - communicate with each other through well-defined channels
 - all containers run by a single OS kernel
 - more lightweight than virtual machines

Operating Systems

Tools to Implement Security

- Example: Honeypot
- vm or application based
- fake, monitored, network-accessible environment intended to lure an attacker
 - detect and/or deflect an attack
 - study attack techniques and objectives
- shows limited (safe) set of resources for the attacker
 - may suggest it is running well-known vulnerable services

- types: pure, low and high-interaction



Example deployment of honeypots in a network infrastructure

Operating Systems

Tools to Implement Security

- **Pure honeypots:**
 - full-fledged production systems
 - attacker is monitored using a bug tap installed on the honeypot's link to the network
 - useful but stealthiness can be ensured by a more controlled mechanism
- **Low-interaction honeypots:**
 - simulate only the services frequently requested by attackers.
 - relatively few resources, short response time, less code to implement, reduced complexity of the virtual system's security
- **High-interaction honeypots:**
 - imitate the activities of the production systems hosting a variety of services
 - an attacker may be allowed a lot of services to waste their time (e.g Honeynet)
 - difficult to detect, but expensive to maintain

Operating Systems

Tools to Implement Security

- Separation:
 - keep one user's objects separate from other users
- Types of Separation:
 - **physical**: different processes use different physical objects according to security requirements
 - **temporal**: processes executed at different times according to security requirements
 - **logical**: processes executed as if no other processes and objects exist outside each permitted domain
 - **cryptographic**: process data and computation are concealed and meaningless to other processes

Operating Systems

Tools to Implement Security

- Remarks on separation:
 - different types of separation can be used together
 - increasing order of complexity and decreasing order of security: physical, temporal, logic
 - too inefficient: physical, temporal
 - efficiency favours shifting the burden of protection to the OS
- Sharing:
 - separation is only half the answer
 - need for sharing of objects between users

Operating Systems

Tools to Implement Security

- Different kinds of separation and sharing:
- Do not protect. Appropriate when sensitive procedures are run at separate times.
- Isolate. Processes running concurrently are unaware of the presence of each other.
- Share all or nothing. Owner declares an object to be public (available to all) or private (available to the owner).
- Share but limited access. OS as guard between users and objects, ensuring that only authorised access occurs.
- Limit use of an object. Not only limits access to objects but also the use made of that object after it has been accessed (e.g. view but not copy, or copy but not print).
- Remark: increasing order of complexity to implement and of granularity

Reference Monitor

- **Reference Monitor:** An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects.
- **Security Kernel:** The hardware, firmware, and software elements of a trusted computing base that implements the reference monitor concept. It must mediate all accesses, be protected from modification, and be verifiable as correct.
- **Trusted Computing Base (TCB):** The totality of protection mechanisms within a computer system – including hardware, firmware, and software – the combination of which is responsible for enforcing a security policy. A TCB consists of one or more components that together enforce a unified security policy over a product or system. The ability of the TCB to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel or parameters (e.g. a user's clearance) related to the security policy.

Reference Monitor

- The reference validation mechanism must be **tamper-resistant** (there is no such thing as “tamper-proof”).
- The reference validation mechanism must **always be invoked**.
- The reference validation mechanism must be small enough to be **subject to analysis and tests to be sure that it is correct**.

Reference Monitor Placement

- **Options in a computer system:**
 - In hardware
 - In the operating system kernel
 - In the operating system
 - In the service layer
 - In the application layer
- **Options regarding a program:**
 - OS, an interpreter, within a program

Operating Systems

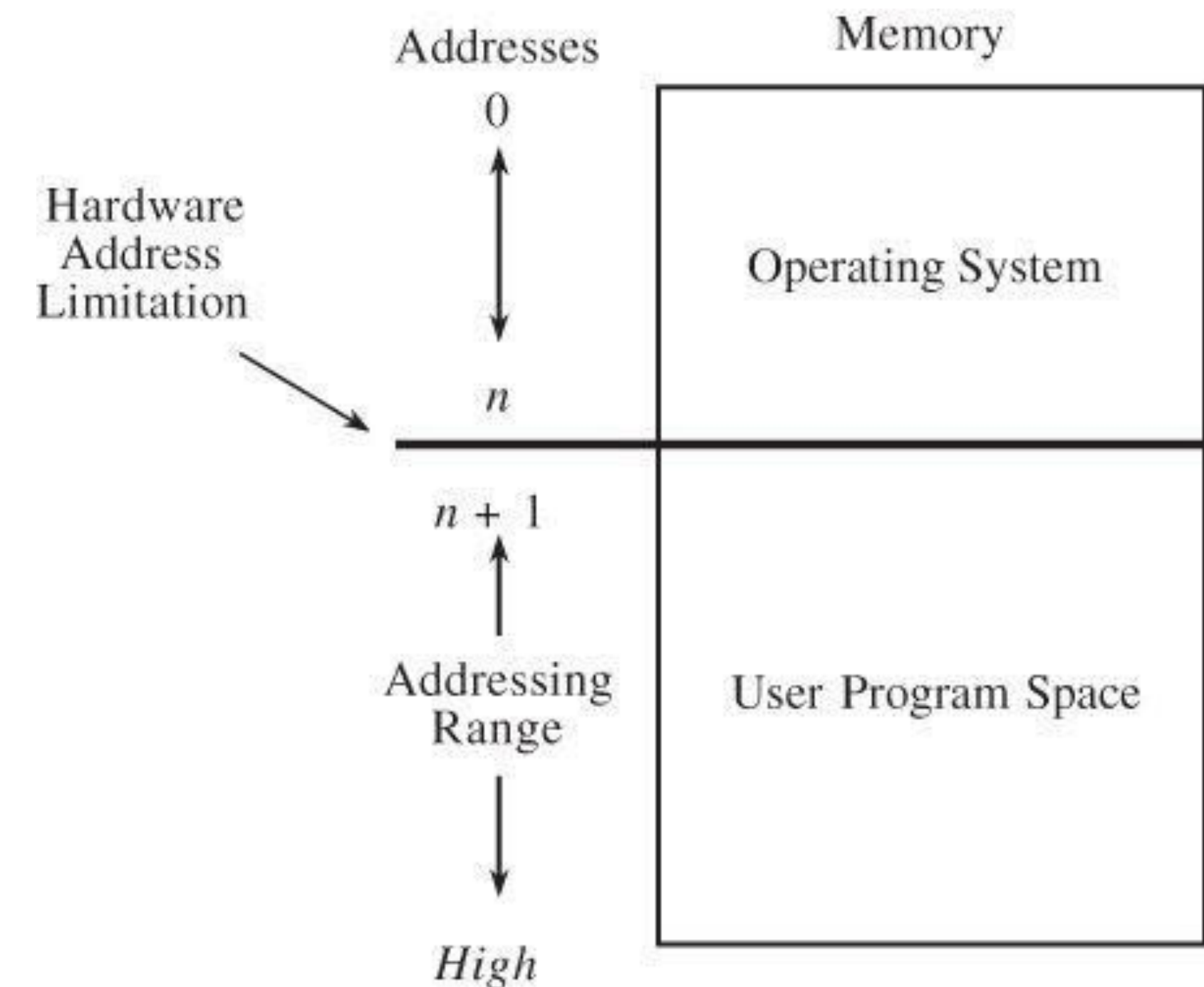
Tools to Implement Security

- **Hardware protection of memory**
- separation and sharing
- sharing of parts of memory between processes and users
- coexistence of the operating system and user processes
- mechanisms can be hard to implement
- Partially supported by hardware
- sharing can be efficient and resistant to tampering
- e.g.: fence, base/bounds, tagged, segmented, paginated

Operating Systems

Memory Protection: Fence

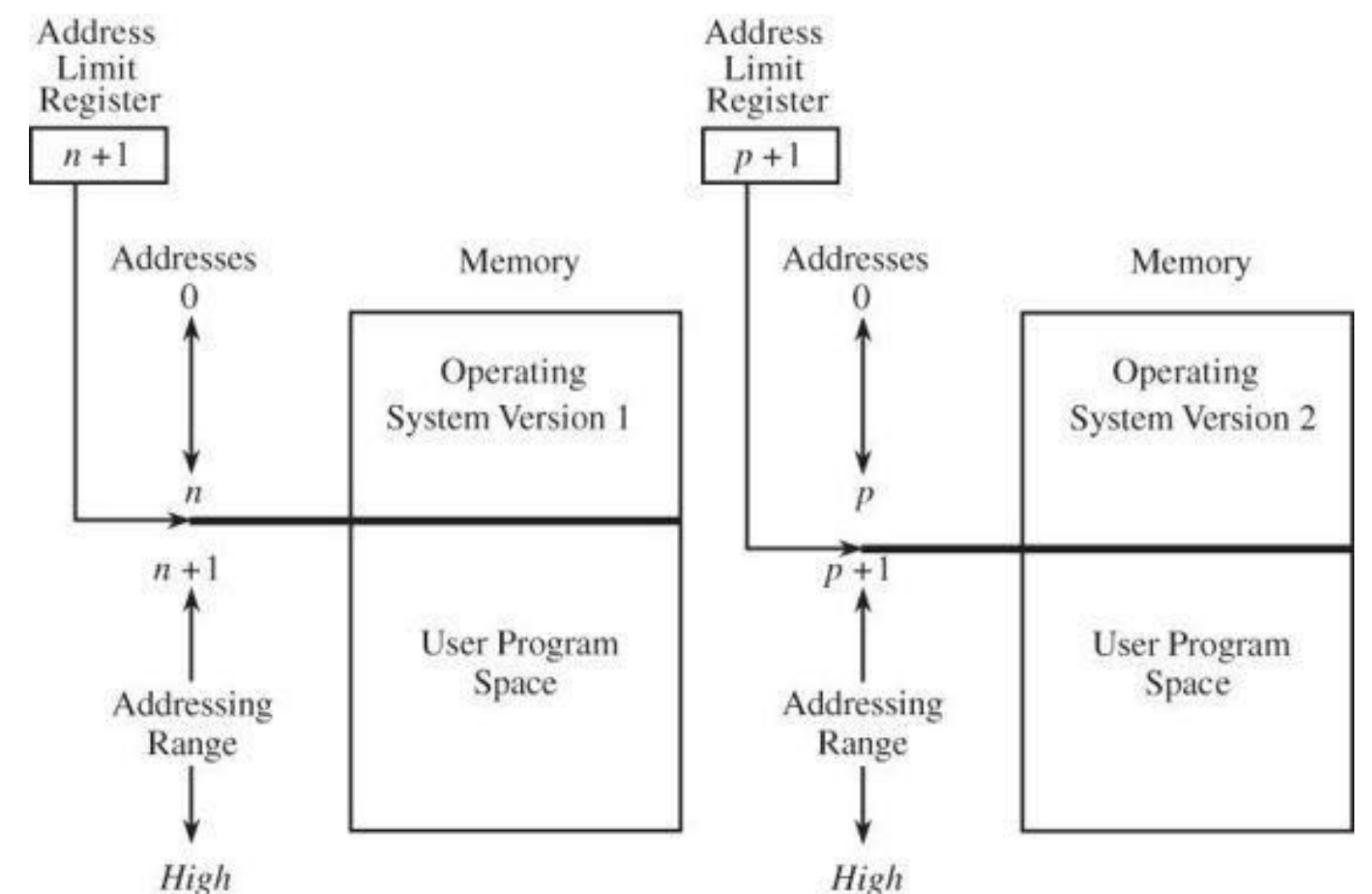
- **Simplest form of memory protection:**
 - introduced in single-user operating systems
 - a method to confine users to one side of a boundary
- **Fence as predefined memory address:**
 - a predefined amount of space reserved for OS
 - very restrictive



Operating Systems

Memory Protection: Fence

- **Fence set in hardware register:**
 - amount of space for the OS can be changed
 - access to memory addresses is automatically compared to fence register
 - if greater, then the access was granted, if lower, it is denied



Operating Systems

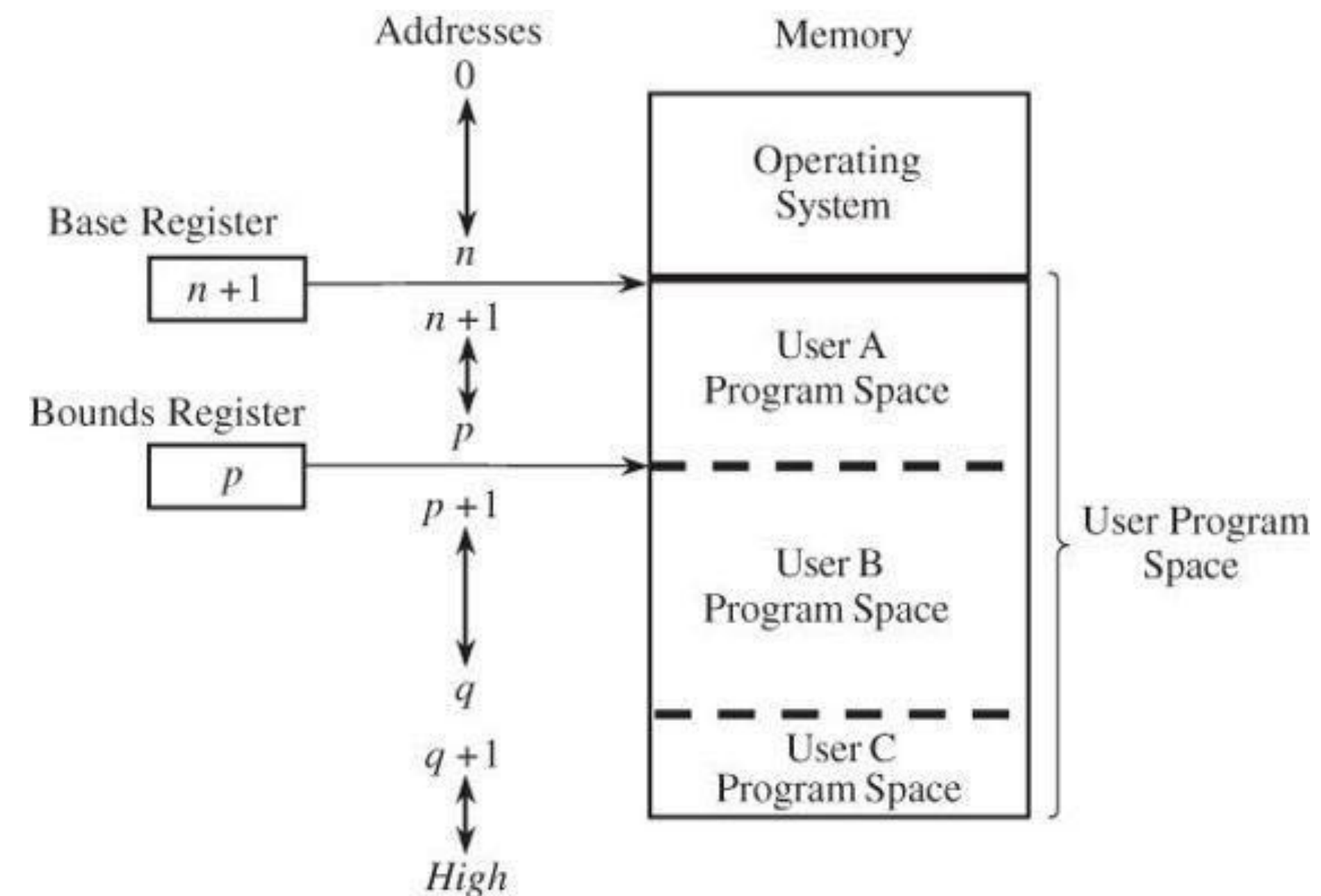
Memory Protection: Fence

- **Protects in one direction only:**
 - the operating system can be protected from the users
 - user programs cannot be protected from each other
 - user programs cannot protect areas of the program:
 - e.g. define areas as non-writable or non-executable

Operating Systems

Memory Protection: Base/Bounds Registers

- **Base/bounds registers:**
 - surrounds a program address space
 - specific to each user program
 - updated on context switching
- **User program addresses:**
 - always added to a base register
 - always checked against a bounds register
- **Protection as well as relocation:**
 - important for multiprogramming systems

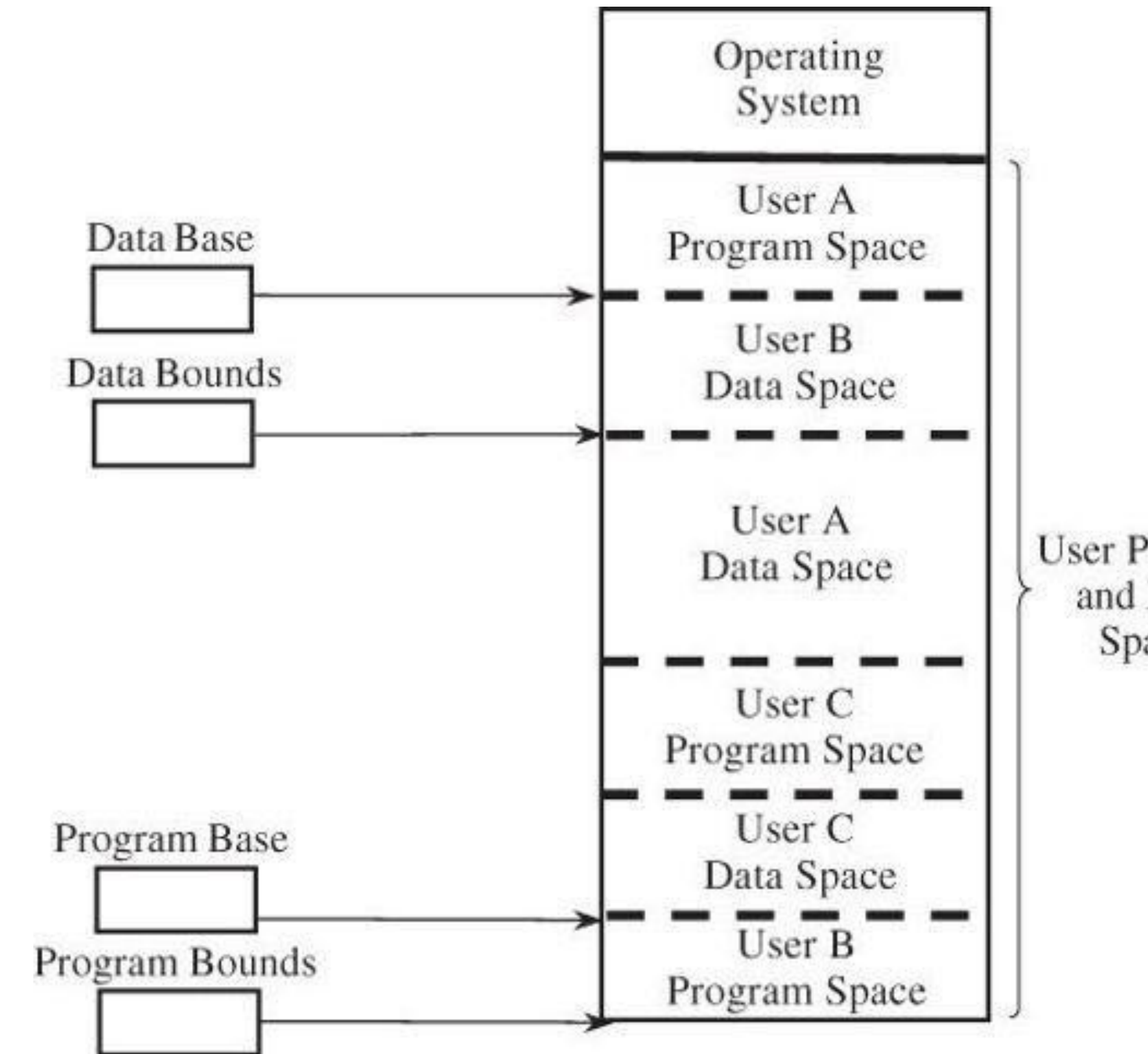


Operating Systems

Memory Protection: Base/Bound Registers

- Base/bounds registers can be used to **protect specific memory areas** of user programs:
 - e.g. code and data
- Could be extended to support more than two areas

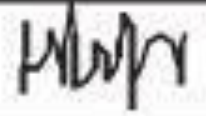
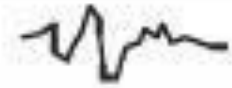
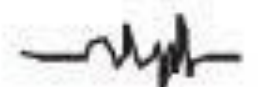
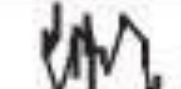
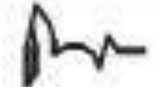
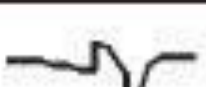
- but practical limit is two register pairs only



Operating Systems

Memory Protection: Tagged Architecture

- **Tagged architecture:**
 - every word has **extra bits identifying the access rights to that word** (or range of words)
 - extra bits are set only by privileged instructions (OS)
 - these bits are checked on each access to the word
 - for practical reasons the number of bits was always small
 - not popular due to compatibility challenges

Tag	Memory Word
R	0001
RW	0137
R	0099
X	
X	
X	
X	
X	
X	
R	4091
RW	0002

Code: R = Read-only RW = Read/Write
X = Execute-only

Operating Systems

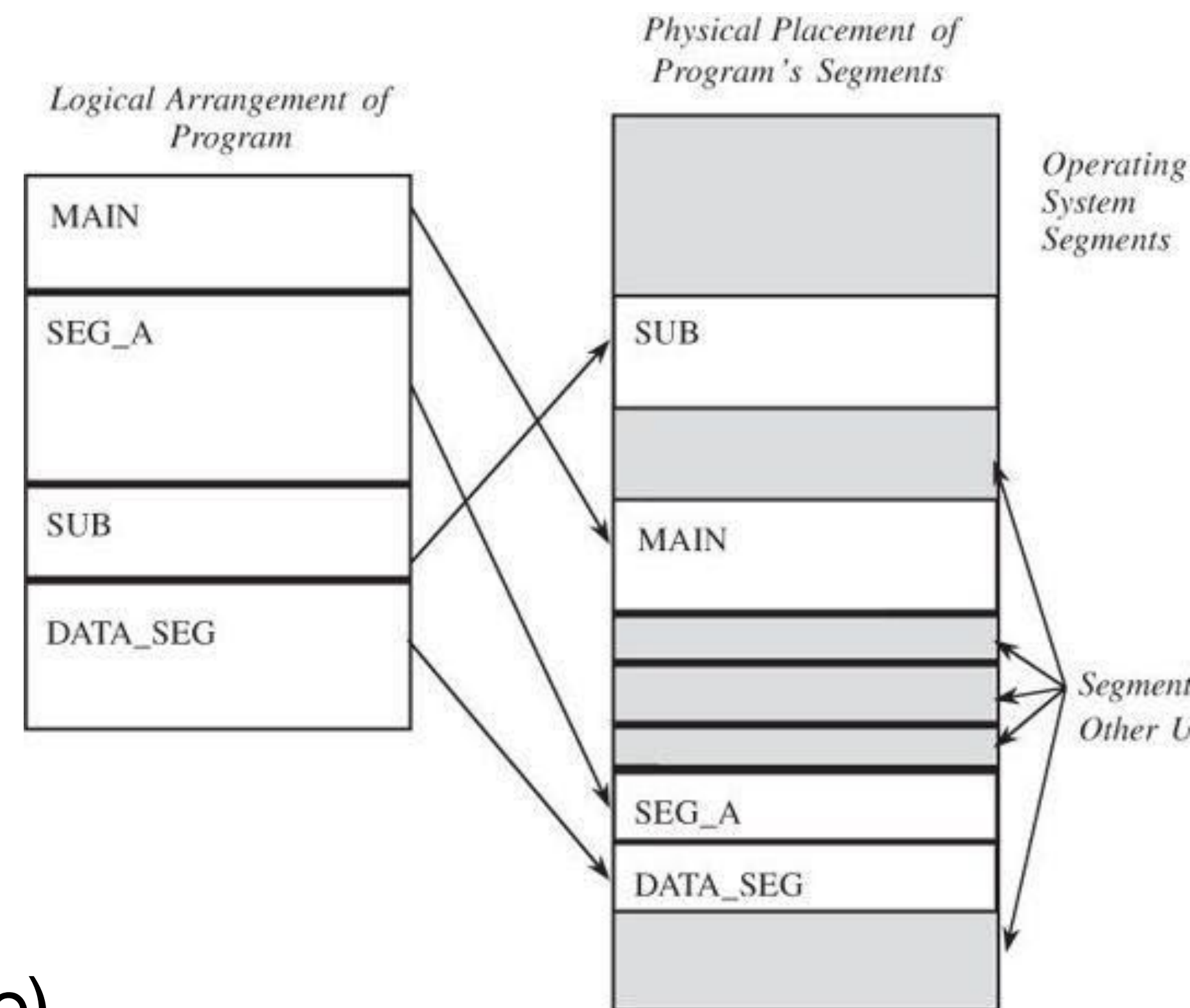
Memory Protection: Virtual Memory

- Virtual memory:
 - hardware support generally (e.g. MMU)
 - used in most general purpose OSes
 - advantages in addressing and protection
 - e.g. segmentation, paging, paging with segmentation

Operating Systems

Memory Protection: Segmentation

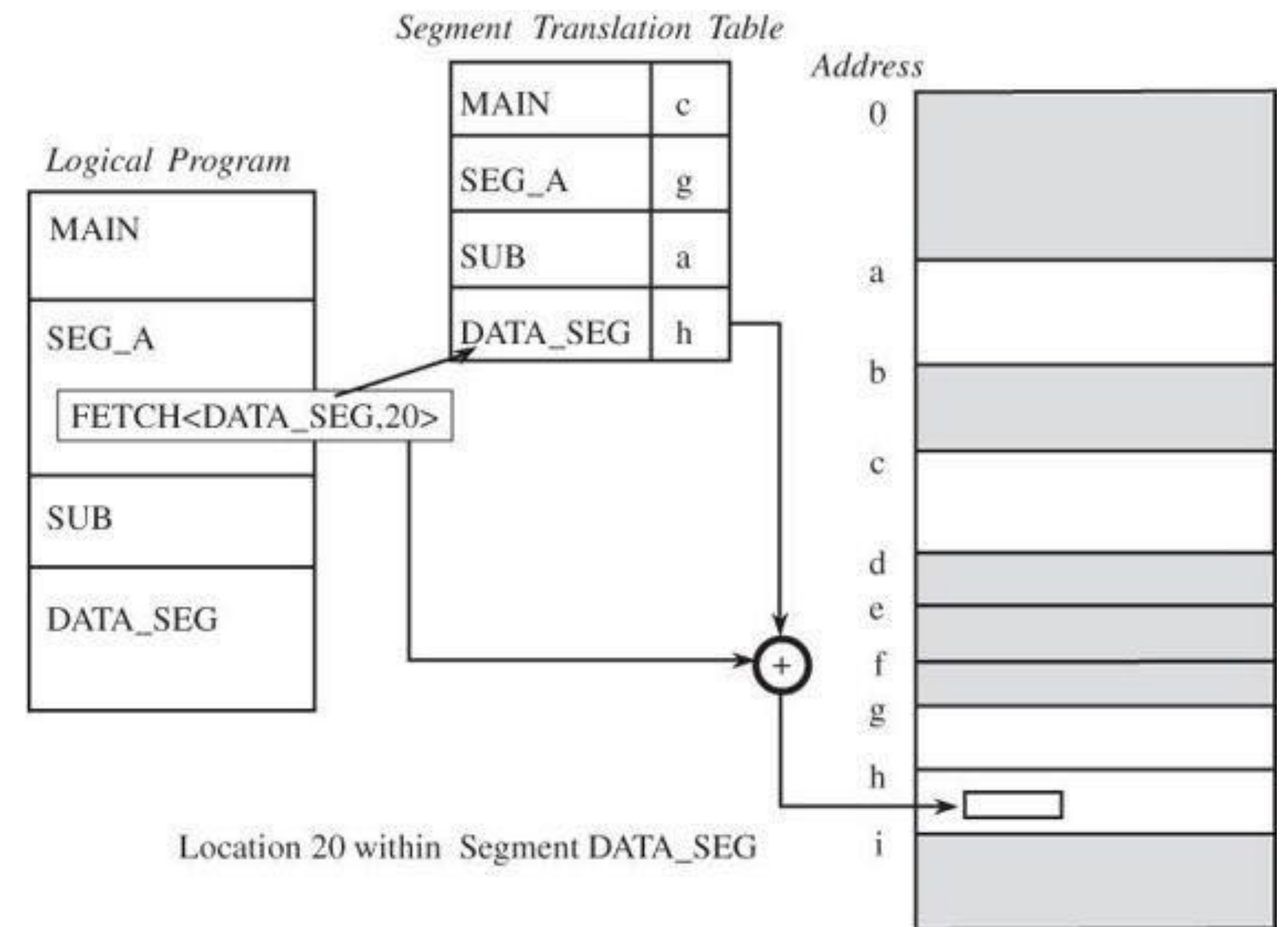
- Segmentation:
 - program divided into logical units (different sizes)
 - addressing: name of segment plus offset
 - similar to an “unbound” number of base/bounds pairs
 - process: OS maintains a table of accessible segment names
 - map: segment names and real addresses (and size)



Operating Systems

Memory Protection: Segmentation

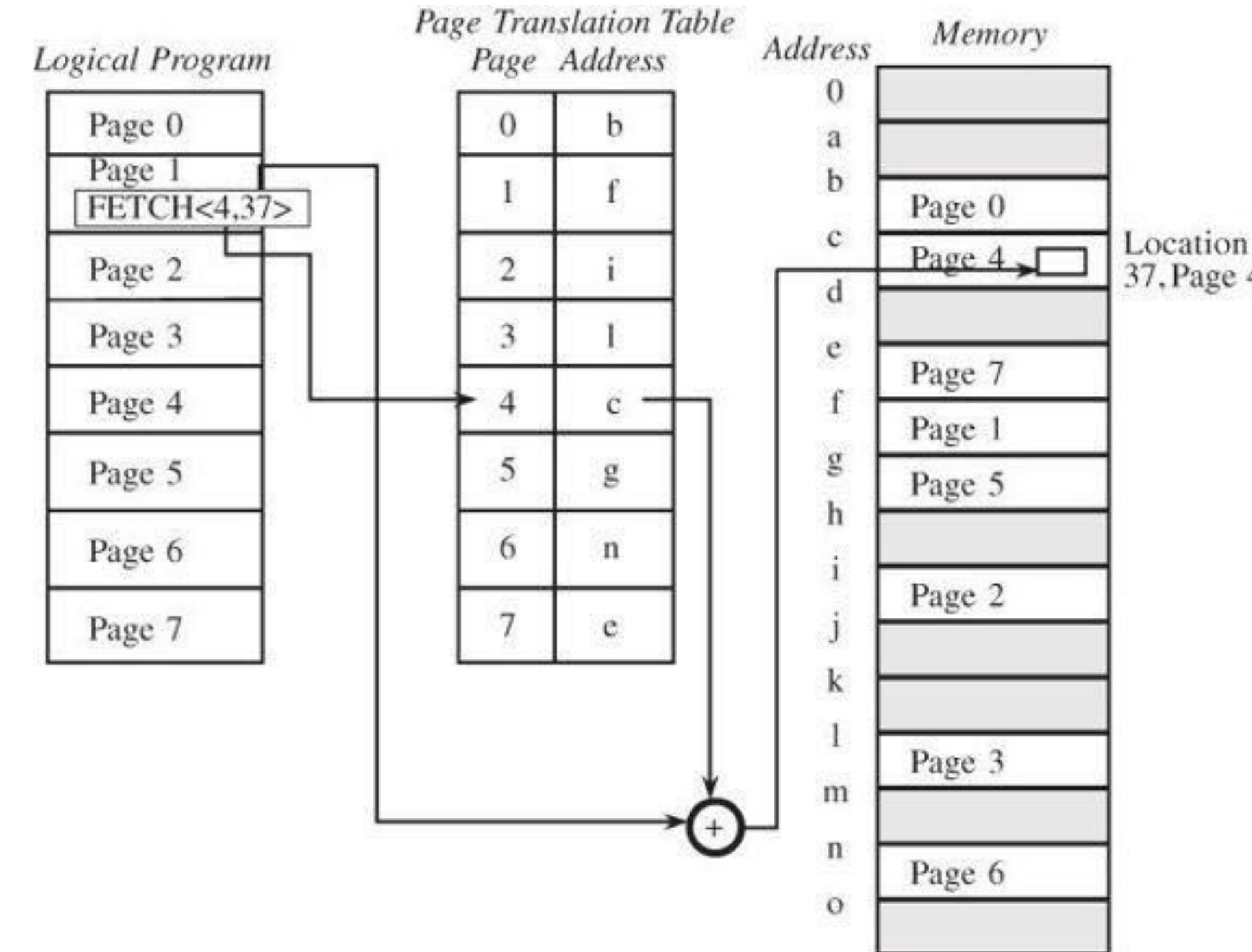
- Segmentation:
 - advantages: relocation, sharing, swap, protection (on each access, MMU), fast address translation, no internal fragmentation
 - disadvantages: inconvenience (e.g. address must identify a segment), external fragmentation



Operating Systems

Memory Protection: Paging

- Paging:
 - program divided into equal-sized units (frame, e.g. 4KB)
 - addressing: page frame (number) plus offset
 - process: OS maintains a table of accessible page frames
 - map: page frames and real addresses
 - very low internal and no external fragmentation
 - advantage: no internal or external fragmentation
 - disadvantage: more complex, slower than segmentation, no logical unity



Operating Systems

Memory Protection: Paged Segmentation

- Paged Segmentation:
- process: set of logical segments
- segment: set of fixed size pages
- advantages: flexible page sizes, simplified memory allocation, additional level of protection

