

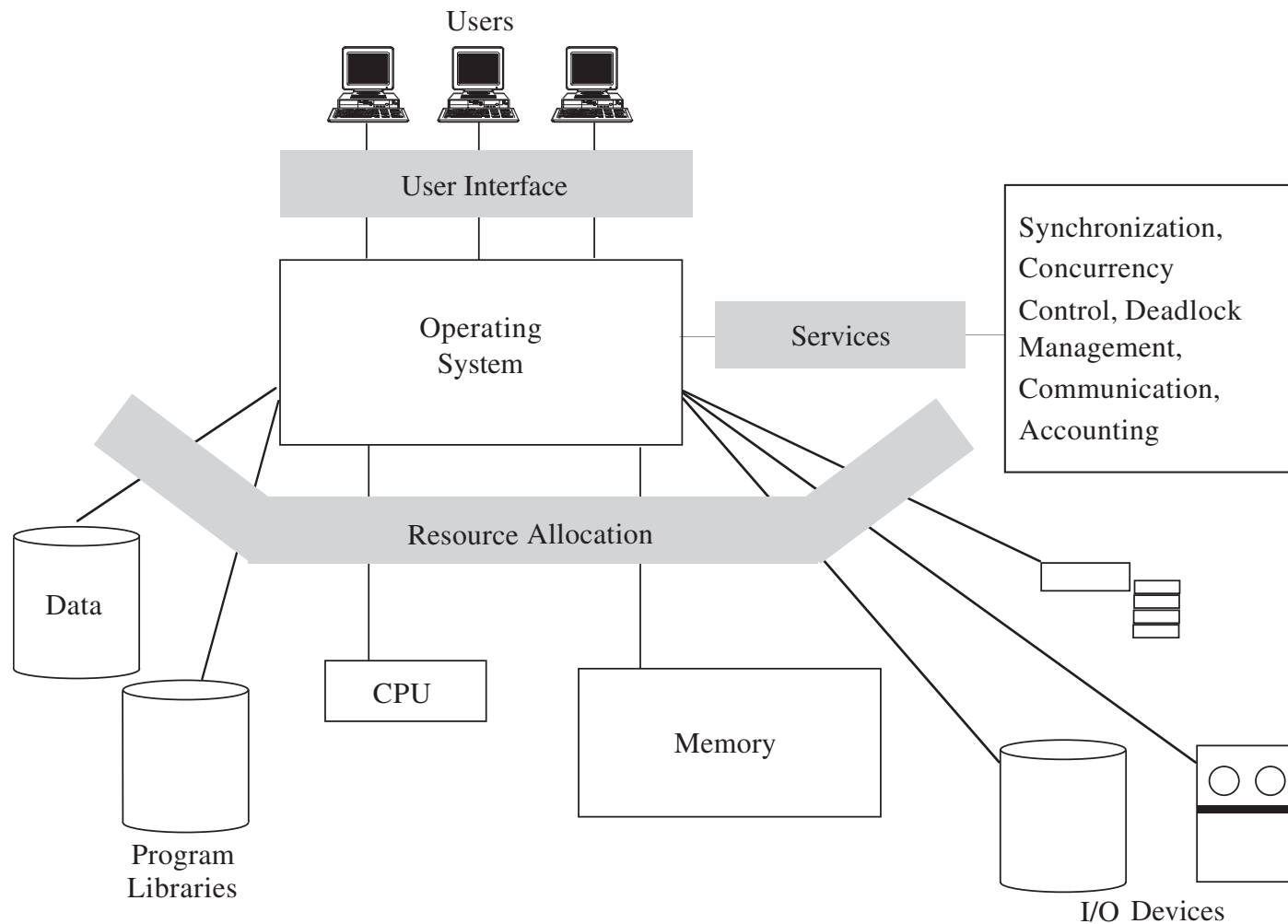
CISC 7320X - COMPUTER SECURITY

Chapter 5 - Operating Systems

Chapter 5 Objectives

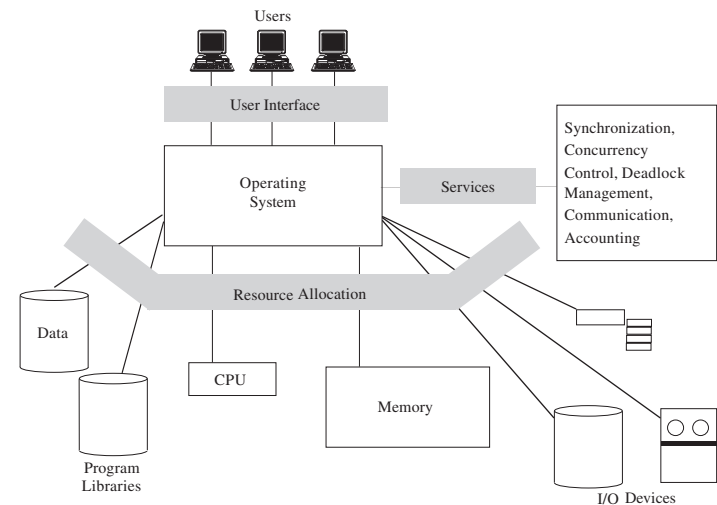
- Basic security functions provided by operating systems
- System resources that require operating system protection
- Operating system design principles
- How operating systems control access to resources
- The history of trusted computing
- Characteristics of operating system rootkits

Operating System Functions



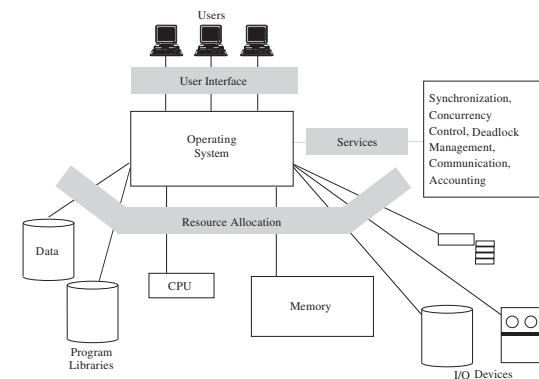
Operating System Functions

- Security-relevant features:
 - Enforced sharing
 - Inter-process communication and synchronization
 - Protection of critical data
 - Guaranteed fair service
 - Interface to hardware
 - User authentication
 - Memory protection



Operating System Functions

- Enforced sharing:
 - Resources should be made available to users
 - appropriate sharing brings about the need to guarantee integrity and consistency.
- Controlled sharing is supported
 - though table lookup, combined with integrity controls such as monitors or transaction processors



Operating System Functions

- Inter-process communication and synchronization:
 - Executing processes sometimes need to communicate with other processes
 - or to synchronize their accesses to shared resources.
 - Operating systems act as a bridge between processes
 - responding to process requests for asynchronous communication with other processes or synchronization.
 - Interprocess communication is mediated by access control tables.

Operating System Functions

- Protection of critical data:
 - The operating system must maintain data by which it can enforce security
 - if these data are not protected against unauthorized access the operating system cannot provide enforcement.
 - Protect against unauthorized read, modify, and delete
 - Various techniques support protection of operating system security data
 - encryption, hardware control, and isolation

Operating System Functions

- Guaranteed fair service:
 - CPU usage and other service should ensure no user is indefinitely starved from receiving service
 - Hardware clocks combine with scheduling disciplines to provide fairness.
 - Hardware facilities and data tables combine to provide control

History of Operating Systems

- Single-user systems, no OS
- Multiprogrammed OS, aka monitors
 - Multiple users
 - Multiple programs
 - Scheduling, sharing, concurrent use
- Personal computers

History of Operating Systems

- First, an entire computer was dedicated to one program at a time
 - but this approach proved wasteful
- The first operating systems saved startup, loading, and shutdown time
 - made much better use of limited resources

History of Operating Systems

- The first personal computers took a major step back, as they were dedicated to single users
 - effectively one program at a time
- Multitasking returned to the mainstream in the 1990s
 - With all the lessons of the early shared computers

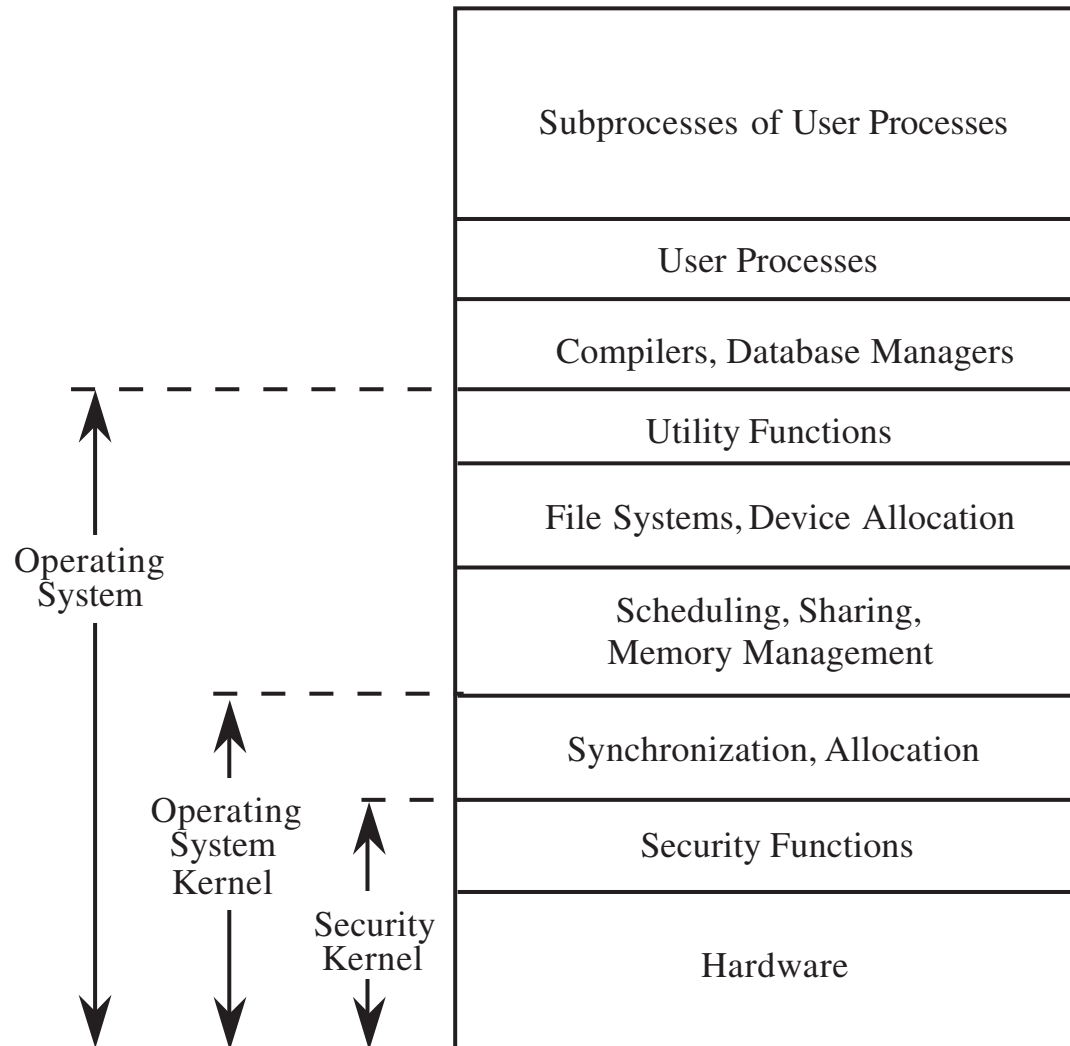
Protected Objects

- Memory
- Sharable I/O devices, such as disks
- Serially reusable I/O devices, such as printers
- Sharable programs and subprocedures
- Networks
- Sharable data

OS Layered Design

- OS can be visualized in layers,
- From most critical (bottom) to least critical

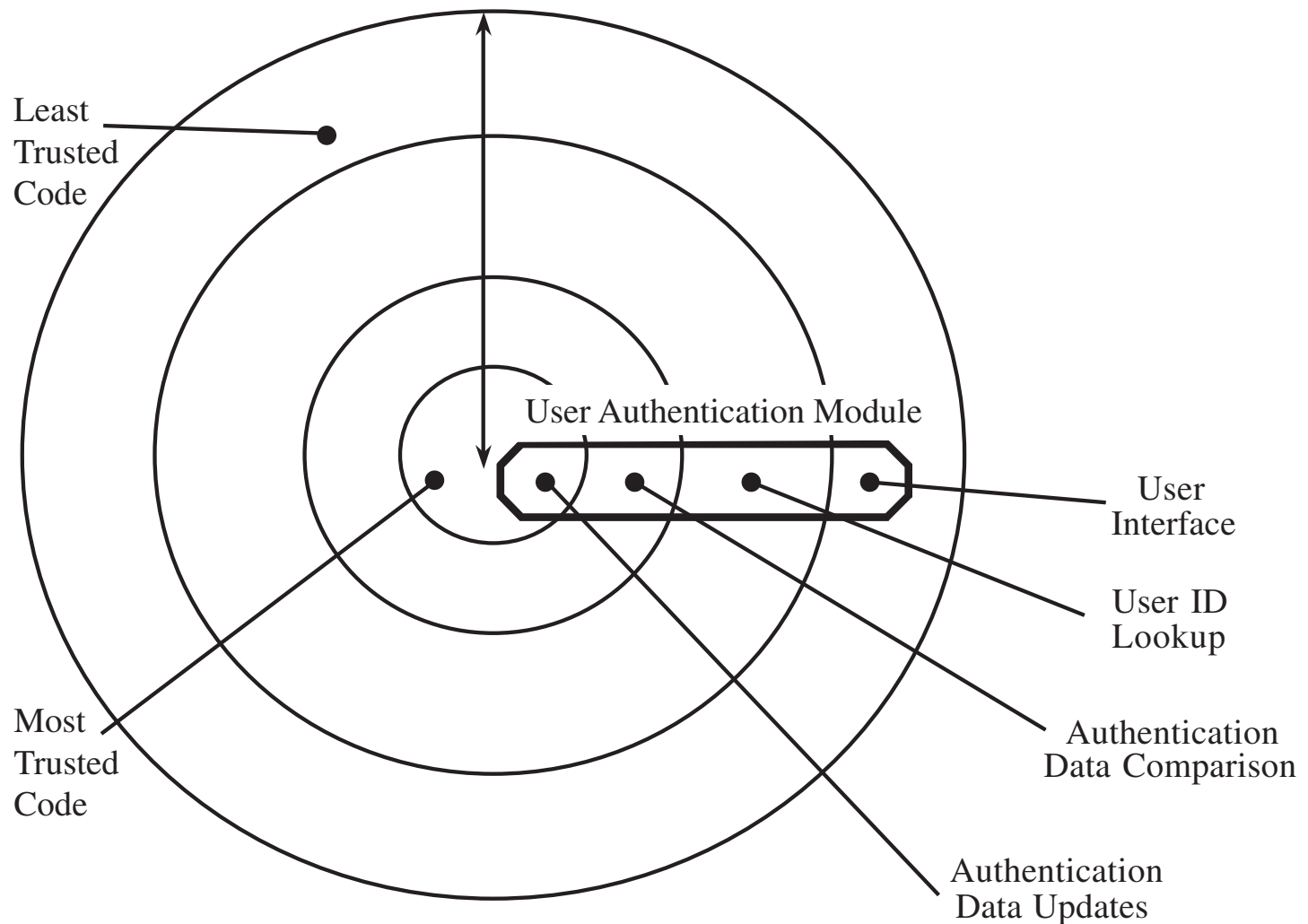
OS Layered Design



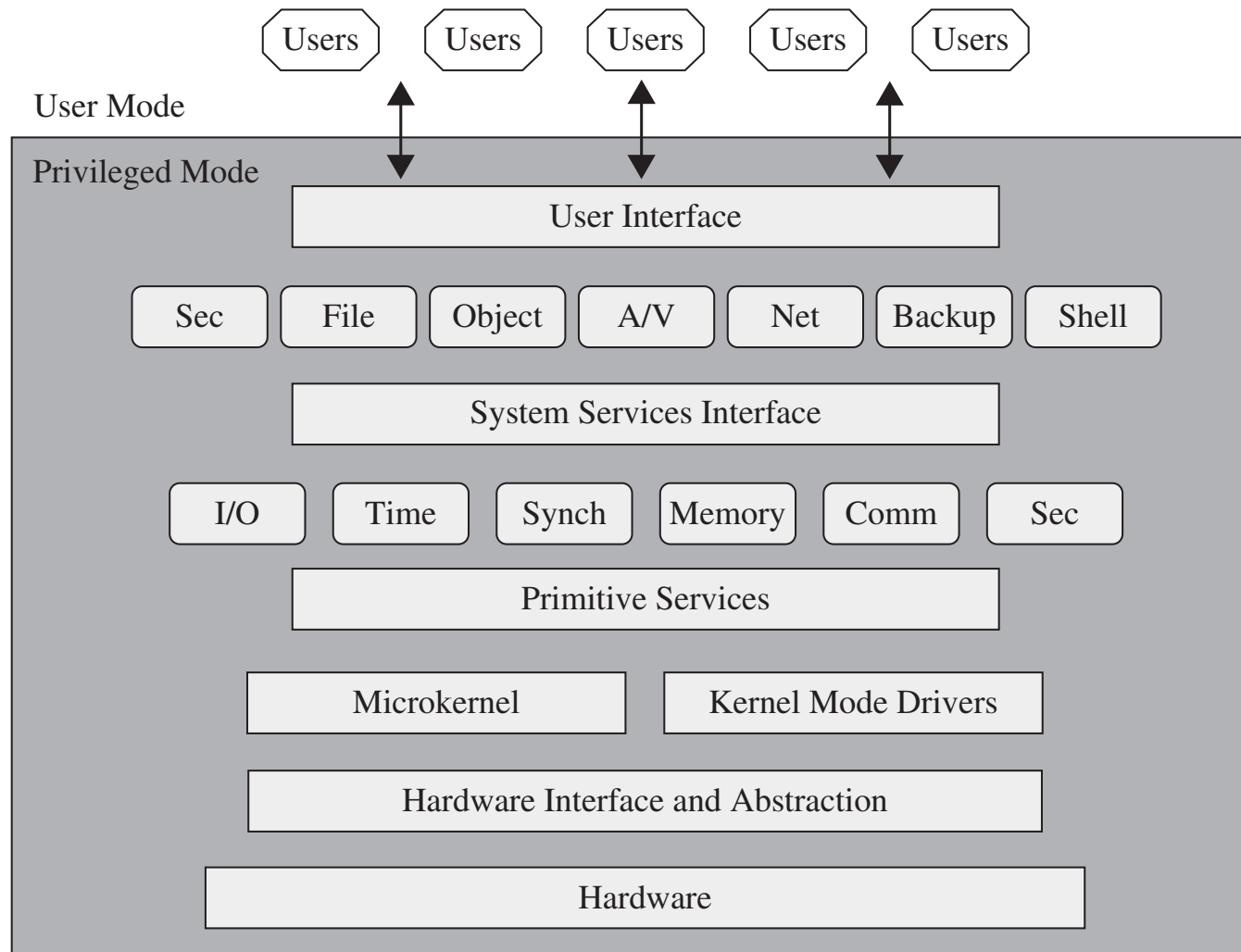
OS Layered Design

- OS can be visualized in layers,
- From most critical (bottom) to least critical
- Authentication is a good example of a function that needs to span the layers
 - in the layered model

Functions Spanning Layers



Modular OS Design



Modular OS Design

- Modern OSs are built from discrete modules
- These modules generally come from a variety of sources
 - are subject to updating/overwriting
 - => so they cannot trust one another.

Virtualization

- With virtualization, the OS presents each user with just the resources that user should see
- The user has access to a virtual machine (VM), which contains those resources
- The user cannot access resources that are available to the OS but exist outside the VM

Virtualization

- A hypervisor, or VM monitor, is the software that implements a VM
 - Translates access requests between the VM and the OS
 - Can support multiple OSs in VMs simultaneously
- Honeypot: A VM meant to lure an attacker into an environment that can be both controlled and monitored

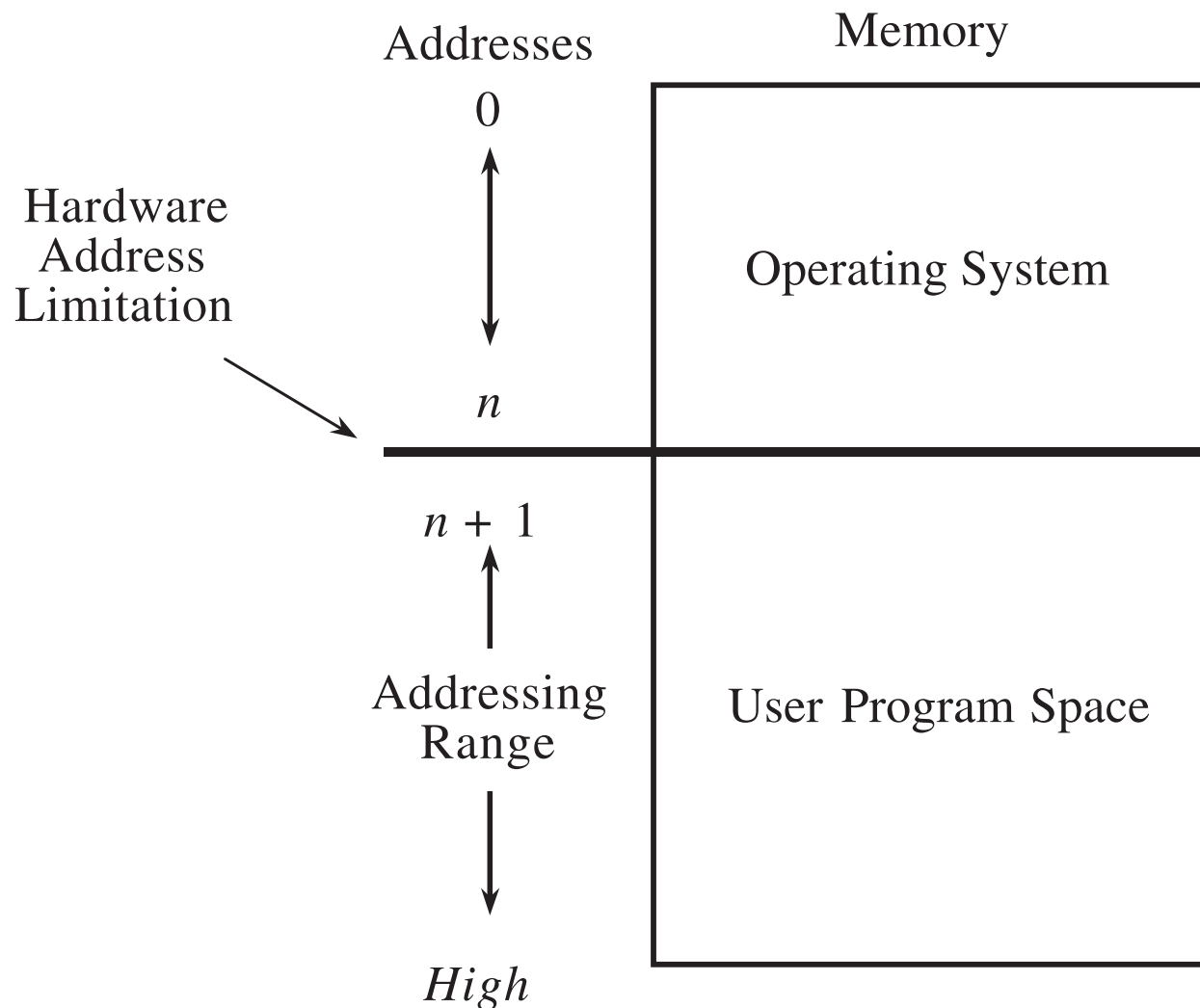
Separation and Sharing

- Methods of separation:
 - Physical
 - Temporal
 - Logical
 - Cryptographic

Separation and Sharing

- Methods of supporting separation/sharing:
 - Do not protect
 - Isolate
 - Share all or share nothing
 - Share but limit access
 - Limit use of an object

Hardware Protection of Memory



Hardware Protection of Memory

- A fence defined by a fixed memory address
- Users have access only to memory above a certain address.

Fence Registers

- Hardware registers used in low-level OS memory management techniques
 - to confine users to one side of a boundary
- Contain the address of the end of the operating system
 - the location of the fence could be changed in time

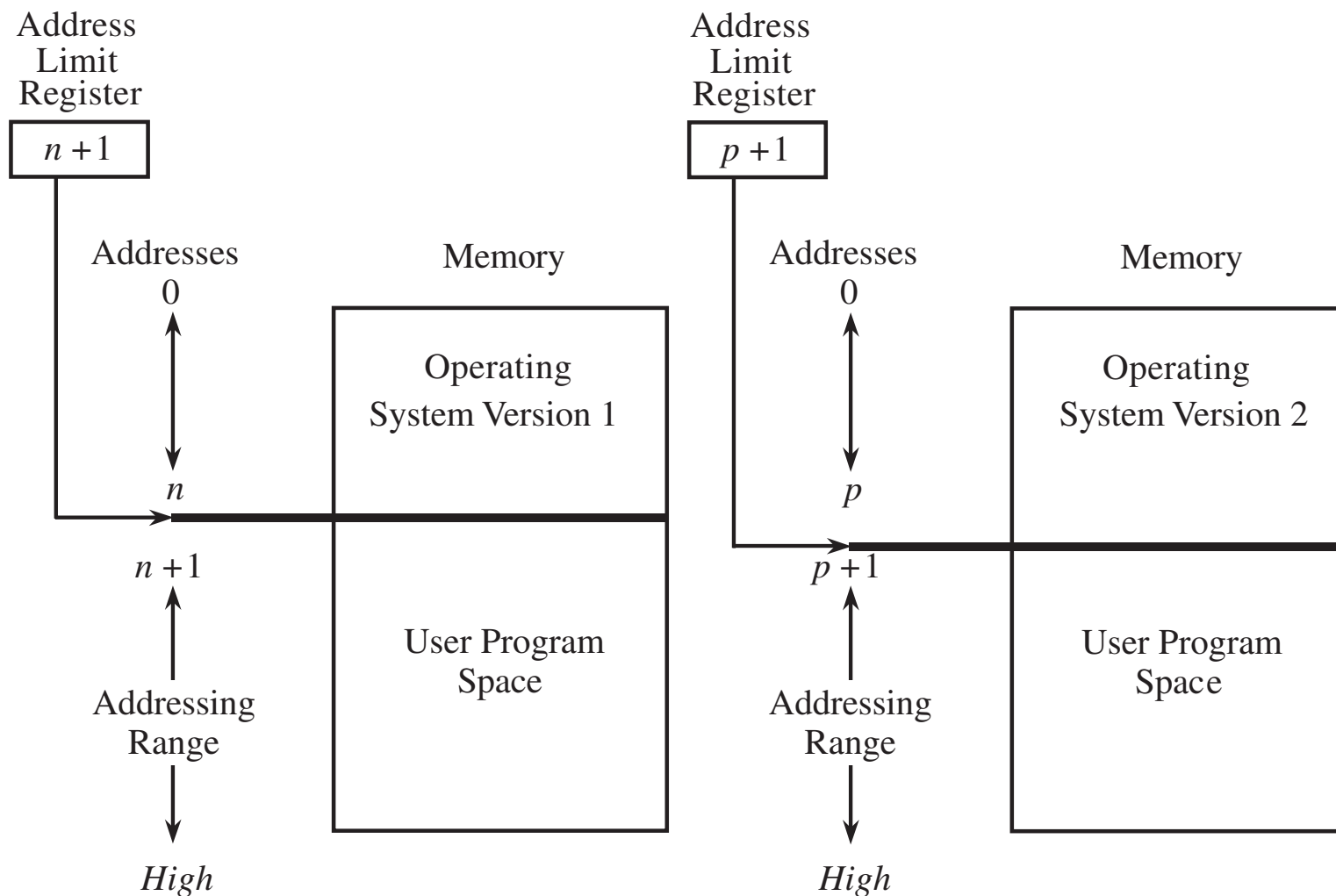
Fence Registers

- A user program generates addresses for data modification
- Each address is automatically compared with the fence address.
- If the address is greater than the fence address (that is, in the user area), the instruction is executed;
 - Otherwise, an error condition is raised

Fence Registers

- Fence Registers protect only in one direction:
 - An operating system can be protected from a single user
 - but the fence cannot protect one user from another user
 - Similarly, a user cannot identify certain areas of the program as inviolable
 - such as the code of the program itself or a read-only data area

Fence Registers



Fence Registers

- Fence Registers protect only in one direction:
 - An operating system can be protected from a single user
 - but the fence cannot protect one user from another user
 - Similarly, a user cannot identify certain areas of the program as inviolable
 - such as the code of the program itself or a read-only data area
- Like fences, but fence registers allow for the boundary to change

Base/Bounds Registers

- A simple form of virtual memory
- Access to computer memory is controlled by one or a small number of sets of processor registers
 - called *base and bounds registers*

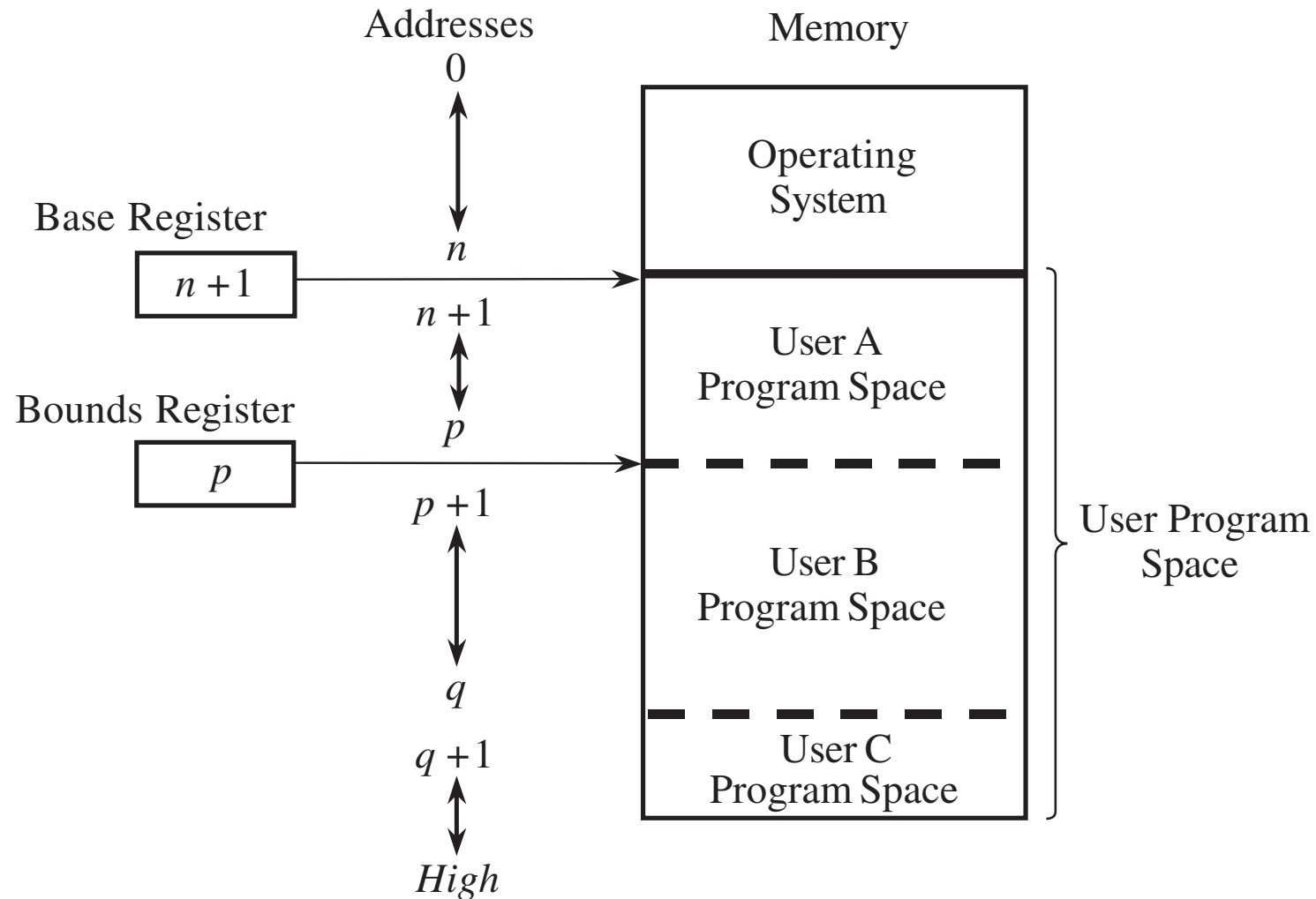
Base/Bounds Registers

- Each user process is assigned a single contiguous segment of main memory.
- The operating system:
 - Loads the physical address of this segment into a base register
 - Loads its size into a bound register.
- Virtual addresses seen by the program are added to the contents of the base register
 - to generate the physical address
- .

Base/Bounds Registers

- The address is checked against the contents of the bounds register
 - to prevent a process from accessing memory beyond its assigned segment.

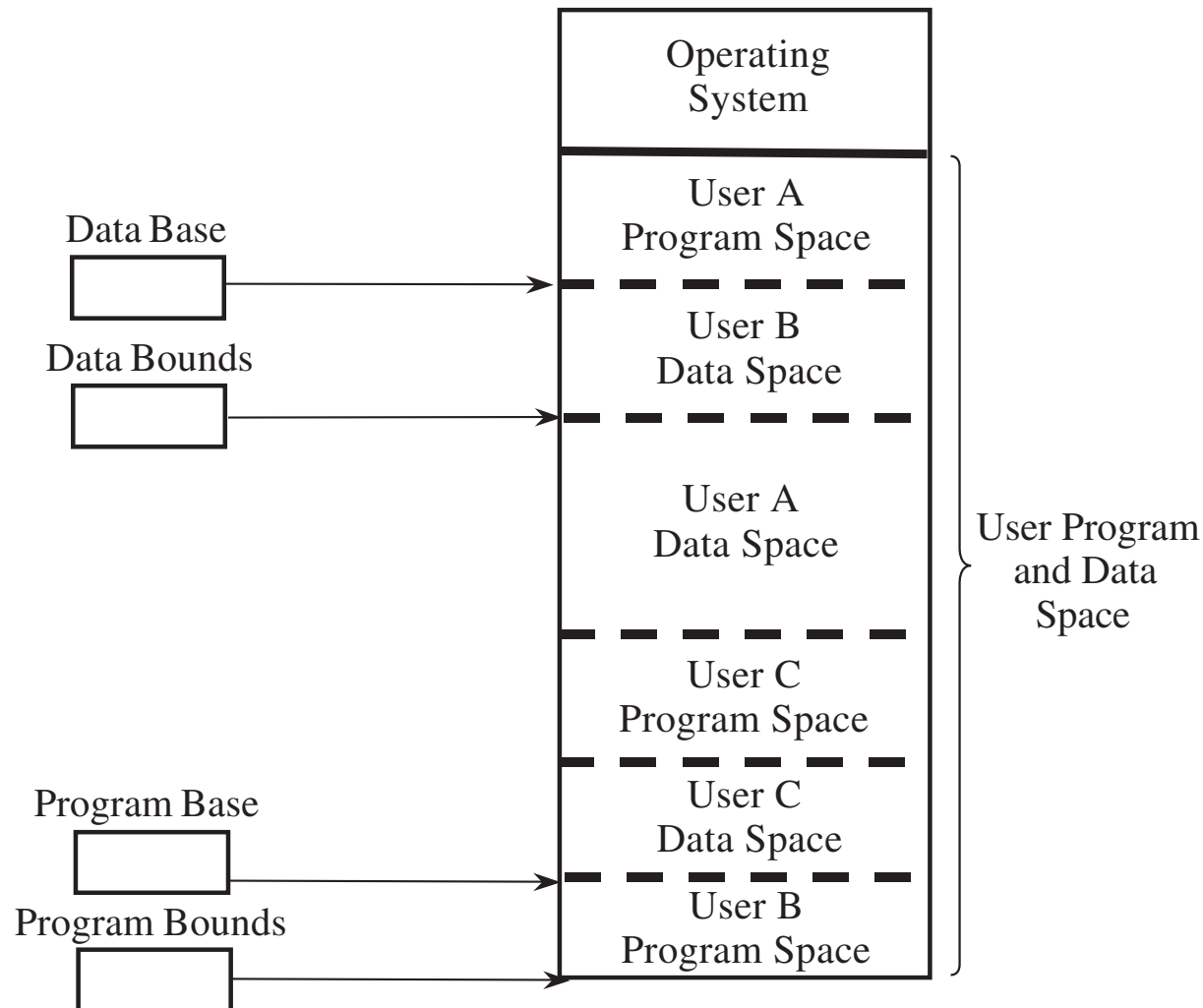
Base/Bounds Registers



Base/Bounds Registers

- With base and bounds registers, memory space can be broken into more than two sections
 - allowing for multiple users

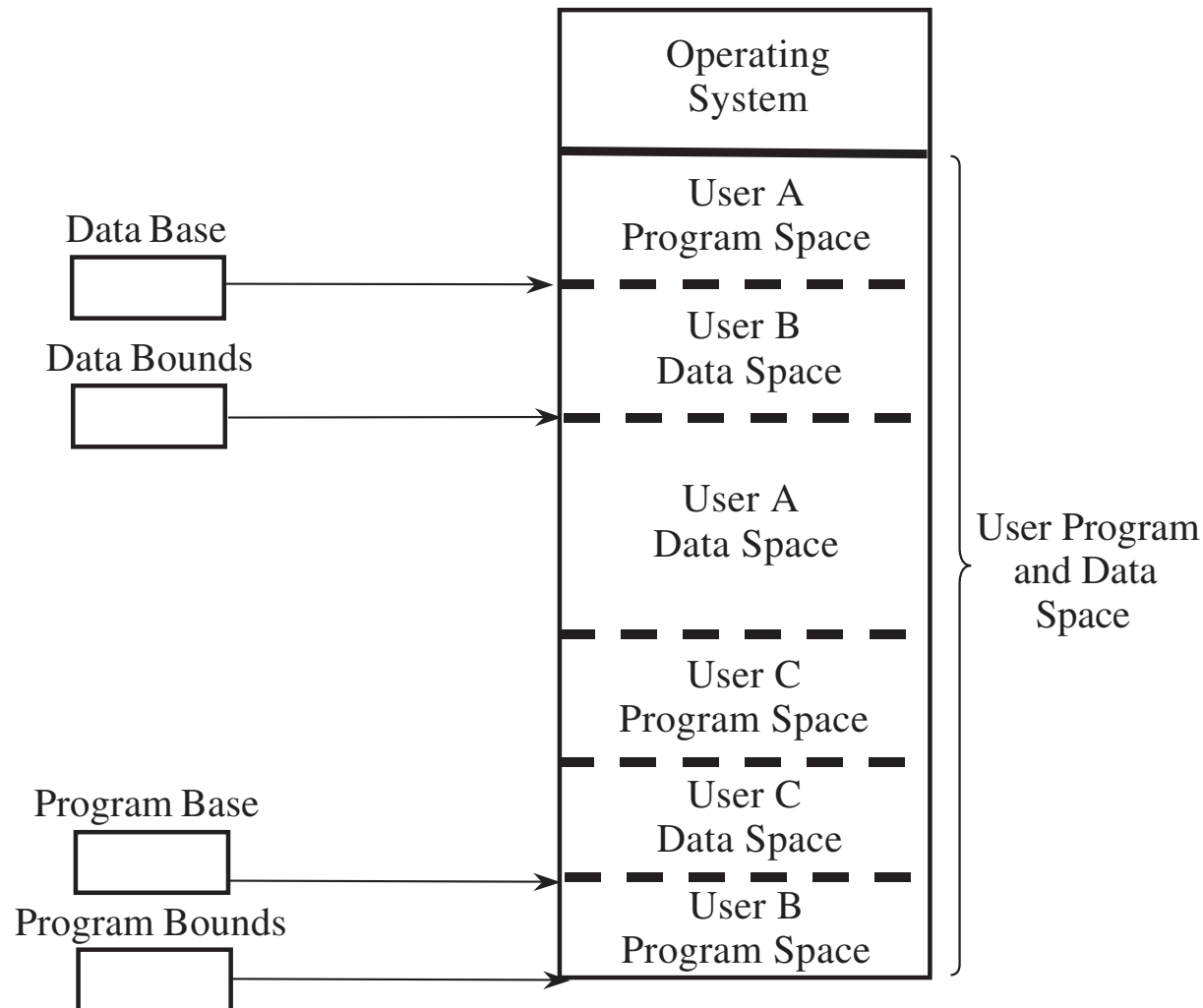
Two Pairs of Base/Bounds Registers



Two Pairs of Base/Bounds Registers

- This separates executable memory from data memory for each user
- making it harder for bugs/attacks to overwrite code







Two Pairs of Base/Bounds Registers



Tagged Architecture

- A particular type of computer architecture
- Extra data bits are attached to each word to denote the data type, the function of the word, or both
- Each tagged union is divided into two sections:
 - Data (a number of bits)
 - A tag section that describes the type of the data:
 - how it is to be interpreted, and, if it is a reference, the type of the object that it points to

Tagged Architecture

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

Tagged Architecture

- In a tagged architecture, each word of machine memory has one or more extra bits to identify its access rights
- The big benefit is that access rights aren't based on contiguous memory locations
- Tagged architecture has not been widely adopted

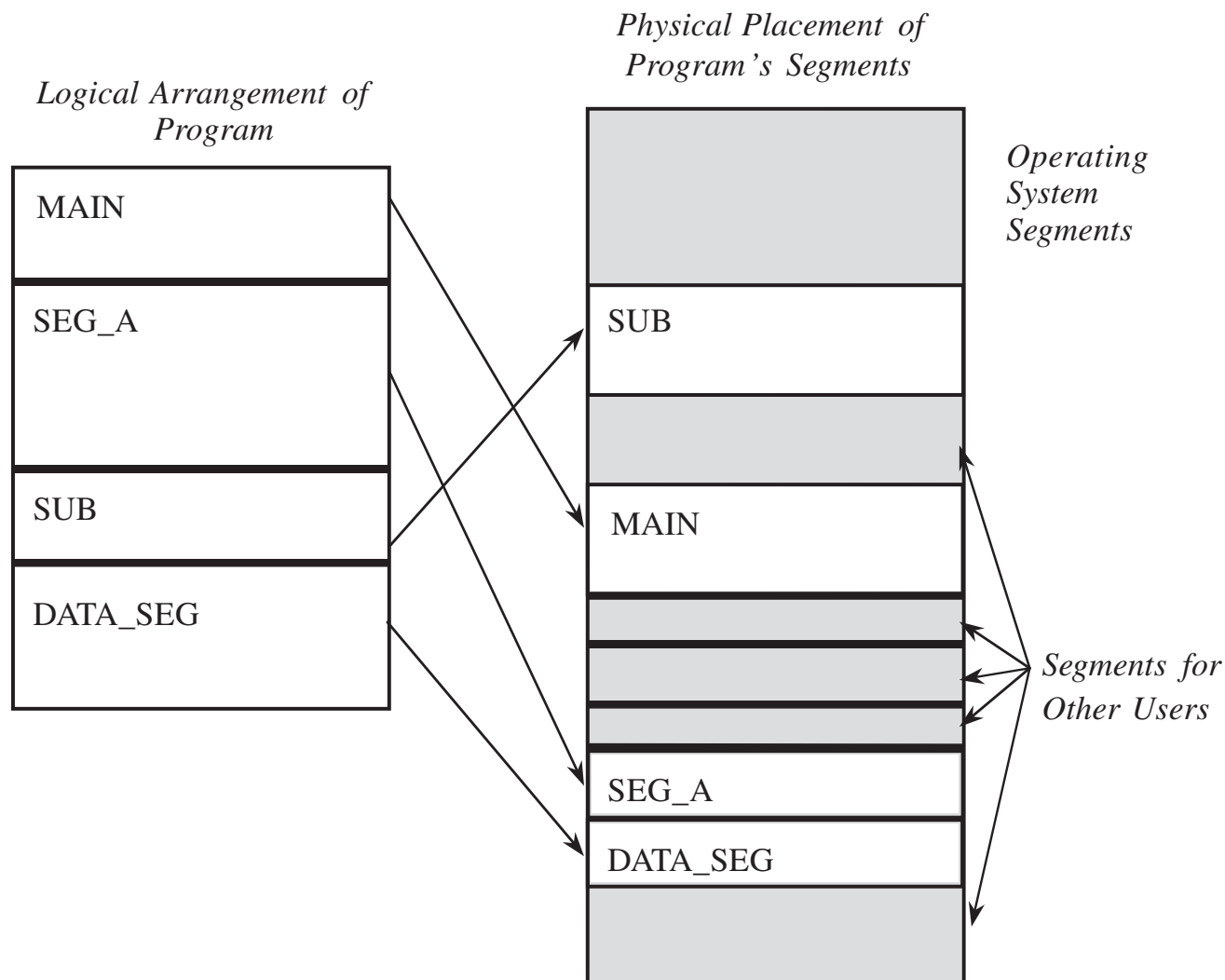
Segmentation

- Dividing the computer's primary memory into segments or sections.
- If segmentation is used
 - => a reference to a memory location includes a value that identifies a segment and an offset (memory location) within that segment.
- Segments or sections are also used in object files of compiled programs
 - Segments are linked together into a program image

Segmentation

- Segments usually correspond to natural divisions of a program such as individual routines or data tables
 - segmentation is generally more visible to the programmer than paging alone
- Different segments may be created for different program modules
 - or for different classes of memory usage such as code and data segments.
- Certain segments may be shared between programs.

Segmentation



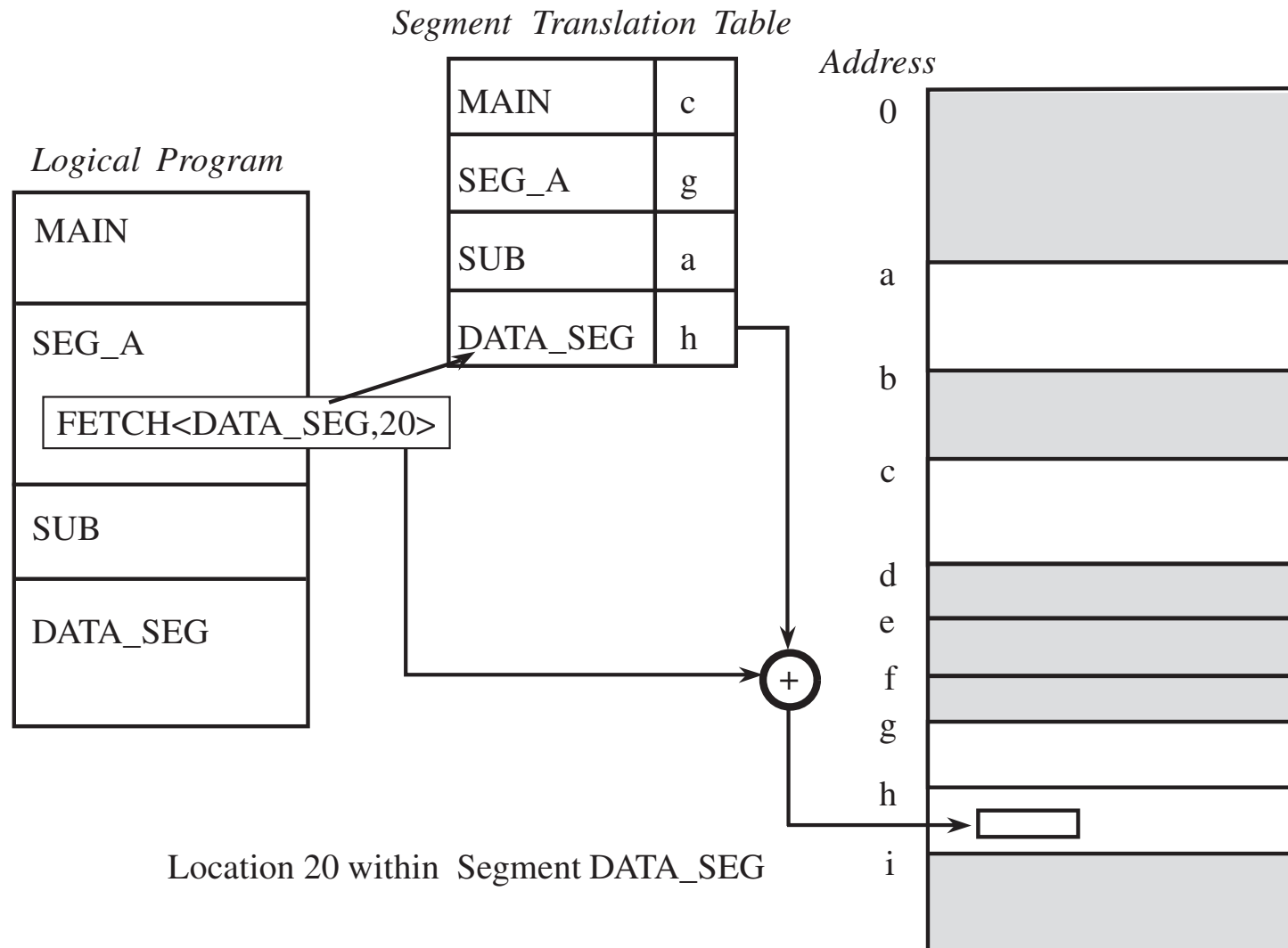
Segmentation

- A program is divided into separate, logical pieces (e.g., an array, a procedure).
- Each segment has its own set of access rights
- The operating system maintains a table of each segment and its true memory address, and it translates calls to each segment using that table

Segmentation

- Advantages:
 - The OS can move segments around as necessary
 - very helpful as segments grow and shrink.
 - Segments can be removed from memory if they aren't being used currently
 - Every legitimate address reference must pass through the OS
 - providing an opportunity for access control

Segment Address Translation



Paging

- Paging is a memory management scheme by which a computer stores and retrieves data from secondary storage
 - for use in main memory.
- In this scheme, the operating system retrieves data from secondary storage in same-size blocks called pages.

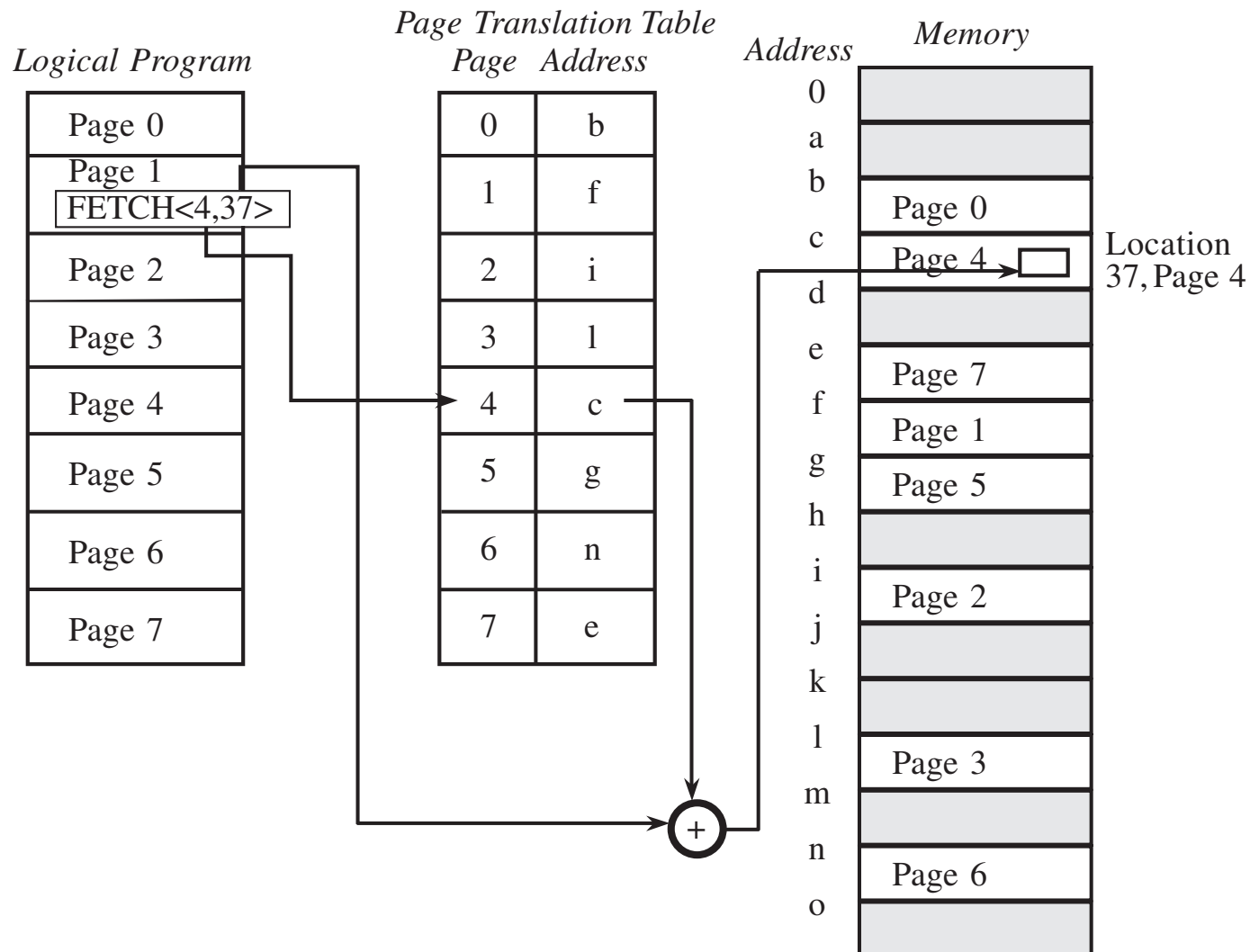
Paging

- Paging is an important part of virtual memory implementations in modern operating systems,
 - using secondary storage to let programs exceed the size of available physical memory.

Paging

- Typically, the main memory is called "RAM" (an acronym of "random-access memory")
- The secondary storage is called "disk" (a shorthand for "hard disk drive")
- The concepts do not depend on whether these terms apply literally to a specific computer system

Paging



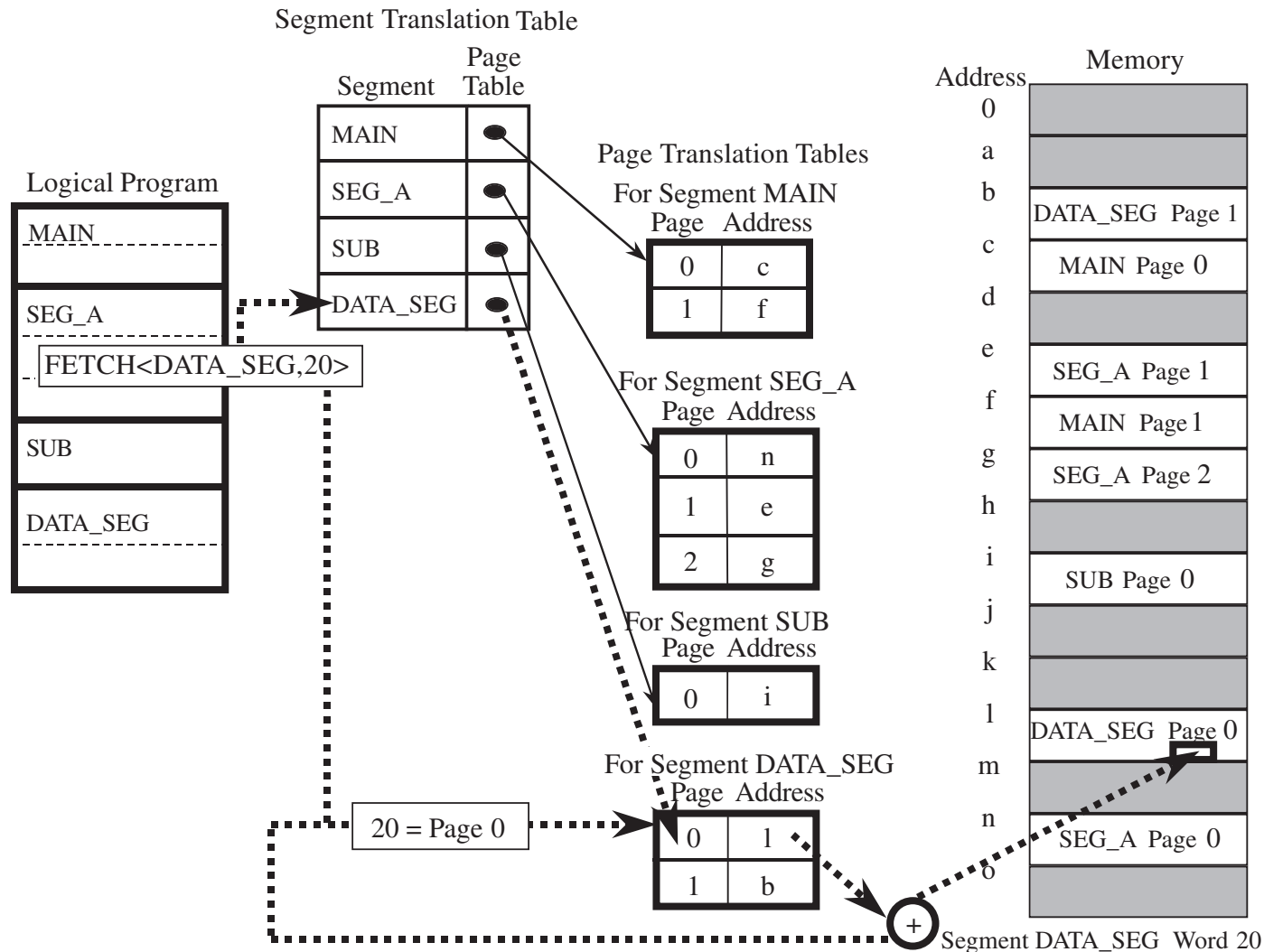
Paging

- Similar to segmentation
 - but programs are broken into fixed-size fragments (pages) rather than being broken down by logical unit
- Programs aren't broken into logical units
 - => paging doesn't allow different parts of a program to have different access rights

Paged Segmentation

- Memory management mechanism:
 - Partition segments into fixed-size pages
 - Allocate and deallocate pages
- Individual segments can be implemented as a paged, virtual address space

Paged Segmentation



Paged Segmentation

- Programs can be broken into segments, and the segments are then combined to fill pages
- This approach creates an extra layer of translation
 - allows for the benefits of both paging and segmentation

SECURE OS DESIGN

Access Control

Principles of Secure OS Design

- Simplicity of design
 - OSs are inherently complex, and any unnecessary complexity only makes them harder to understand and secure
- Layered design
 - Enables layered trust

Principles of Secure OS Design

- Layered trust
 - Layering is both a way to keep a design logical and understandable and a way to limit risk
- Examples:
 - very tight access controls on critical OS functions
 - fewer access controls on important noncritical functions
 - few if any access controls on functions that aren't important to the OS

OS Security

- It is the responsibility of the Operating System to create a protection system
- Ensure that a user who is running a particular program is authentic
 - Identify/authenticate the user as part of access control

-

https://twitter.com/com_mediation

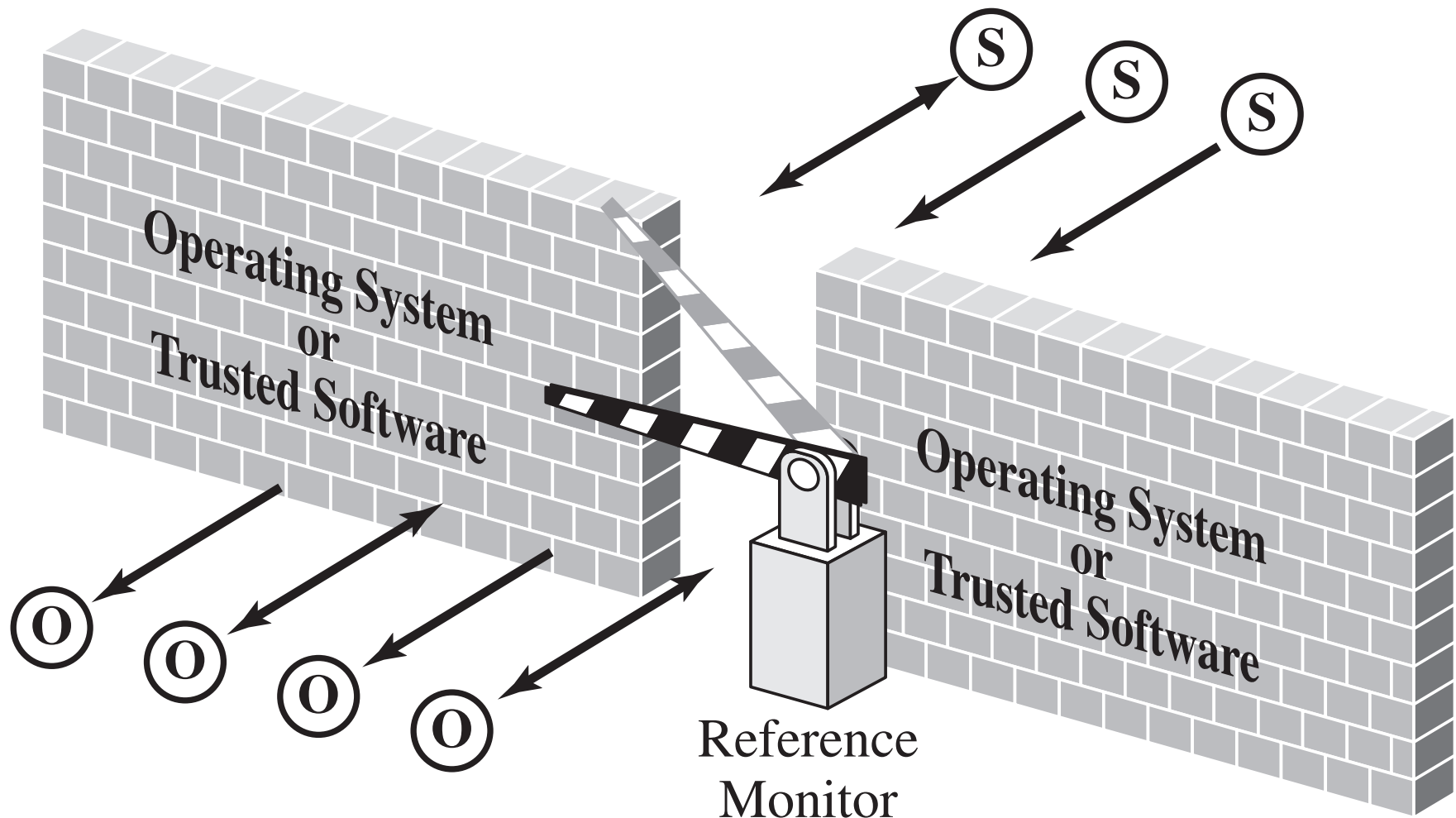
Kernelized Design

- A kernel is the part of the OS that performs the lowest-level functions
 - Synchronization
 - Inter-process communication
 - Message passing
 - Interrupt handling
- A security kernel is responsible for enforcing the security mechanisms of the entire OS
 - Typically contained within the kernel

Reference Monitor

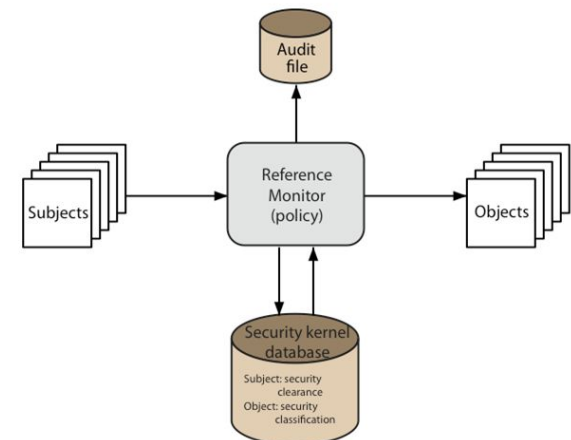
- The most important part of a security kernel is the **reference monitor**
 - the portion that controls accesses to objects

Reference Monitor



Reference Monitor

- Defines a set of design requirements on a reference validation mechanism
- Enforces an access control policy over subjects ability to perform operations on objects
 - Subjects, e.g., processes and users
 - Operations, e.g. read and write
 - Users, e.g. files, etc.

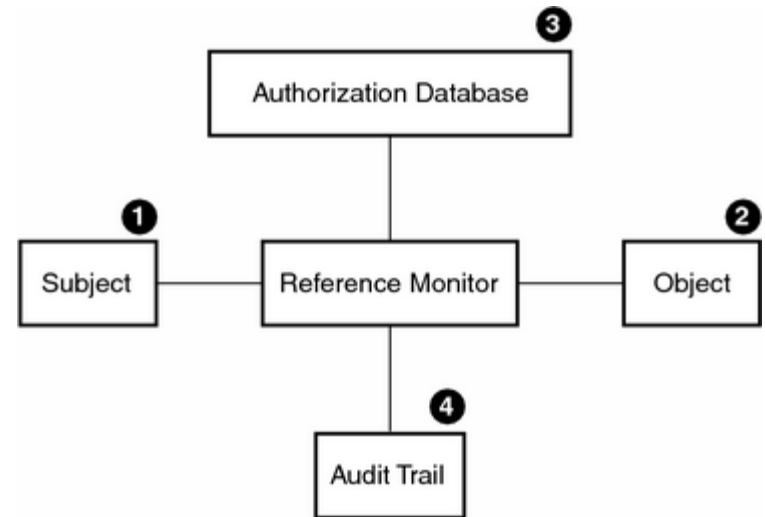


<http://slideplayer.com/slide/697485/>

Reference Monitor

- A reference monitor is responsible for mediating all access to data
- Subject cannot access data directly; operations must go through the reference monitor, which checks whether they're OK

Reference Monitor



VM-0994A-AI

- Authorization Database: Repository for the security attributes of subjects and objects
- Audit trail: Record of all security-relevant events

Criteria for a Reference Monitor

- Ideally, a reference monitor should be:
 - Non-bypassable: mediate every attempt by a subject to gain access to an object
 - Tamper-resistant: Provide a tamperproof database and audit trail
 - that are thoroughly protected from attackers
 - Verifiable: should be simple and well-structured software
 - so that it is effective in enforcing security requirements
 - Unlikely to have bugs

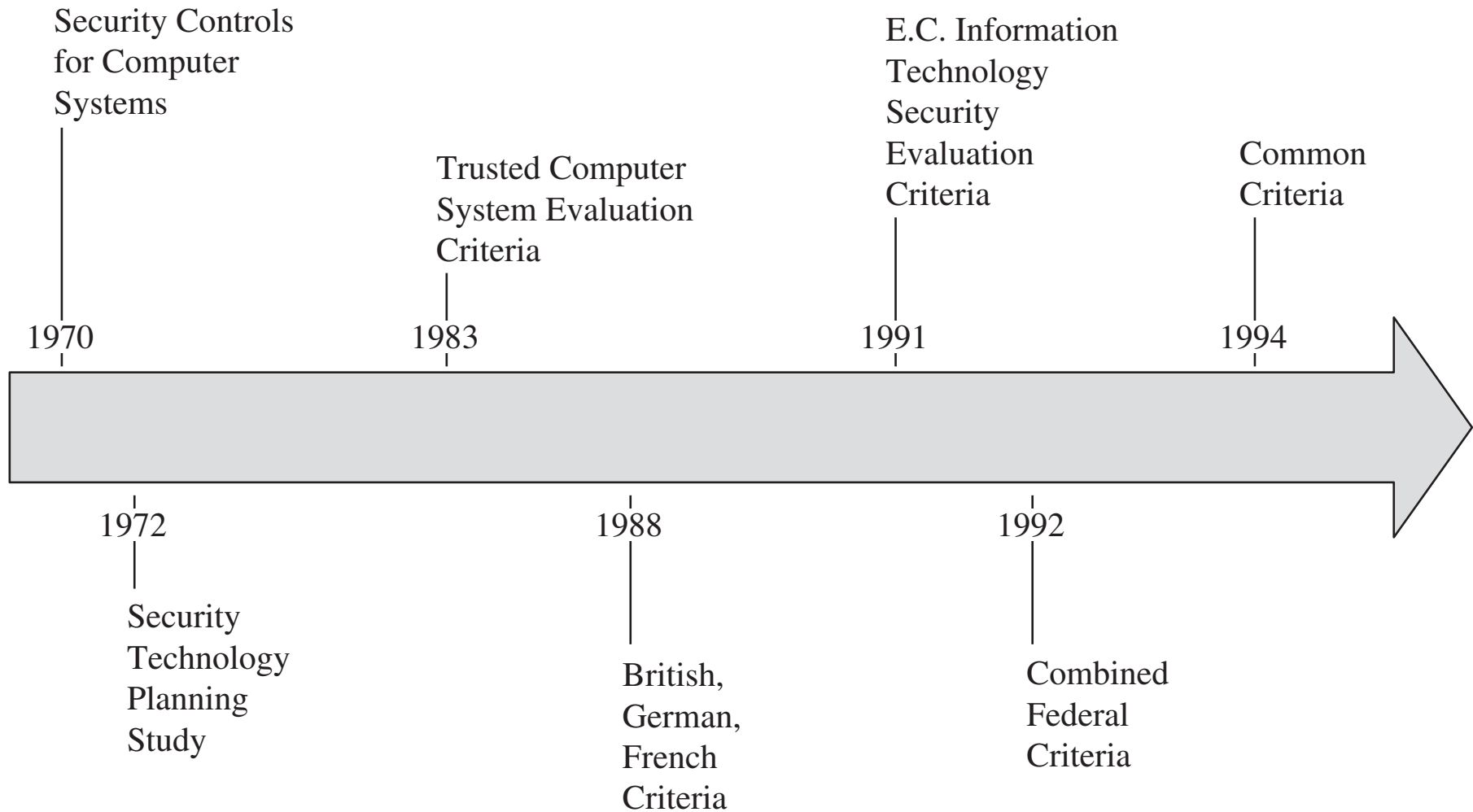
Example: Windows Kernel-Mode Protection

- Windows uses an access control list (ACL) to determine which objects have what security
- The reference monitor provides routines for your driver to work with access control
- Ensures that drivers can only access devices available to them.
 - enforces limits on what resources each process can access

Trusted Systems

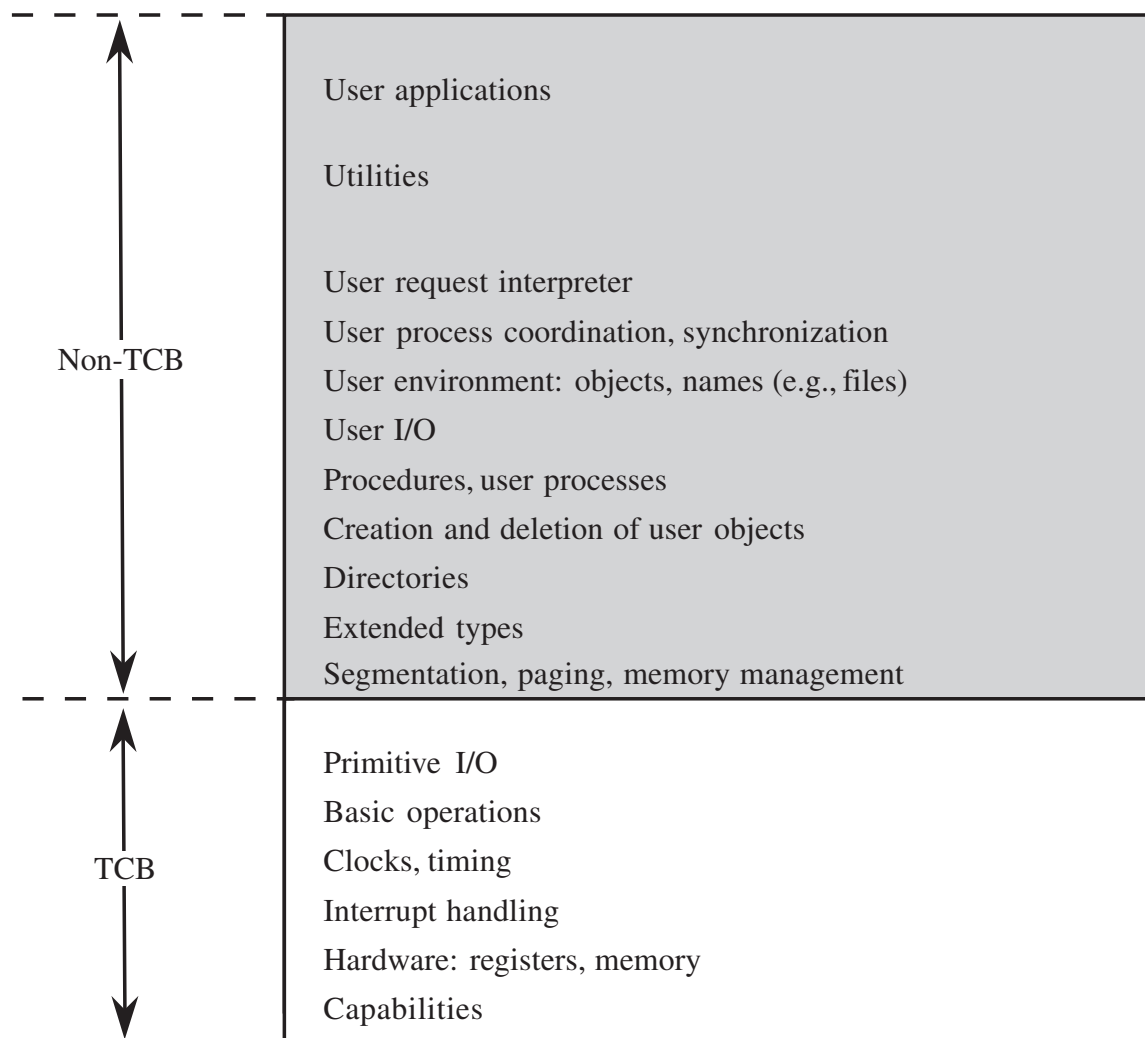
- A trusted system is one that has been shown to warrant some degree of trust that it will perform certain activities faithfully
- Characteristics of a trusted system:
 - A defined policy that details what security qualities it enforces
 - Appropriate measures and mechanisms by which it can enforce security adequately
 - Independent scrutiny or evaluation to ensure that the mechanisms have been selected and implemented properly

History of Trusted Systems



- Attempts to declare computers trustworthy go back almost 50 years
- Over the years, changes in technology have resulted in new requirements
- the explosion of new devices and software have made it challenging to impossible to keep up

Trusted Computing Base (TCB)



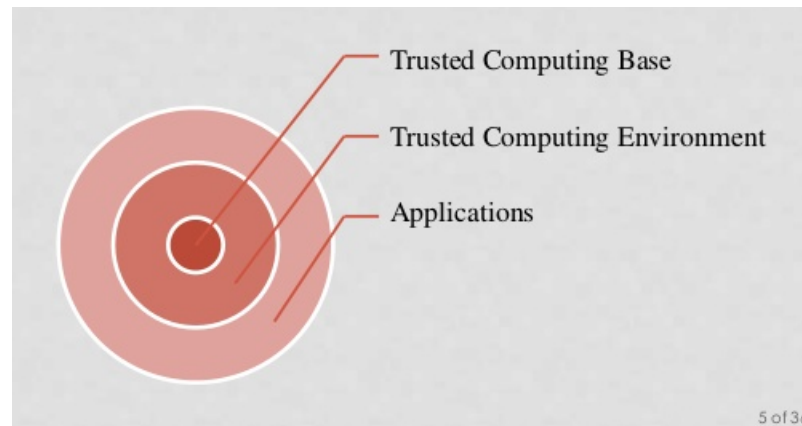
Trusted Computing Base (TCB)

- The TCB portion of the OS is the part we depend on for enforcement of security policy
- The TCB monitors and protects the secrecy and integrity of four basic interactions:
 - process activation
 - execution domain switching
 - memory protection
 - I/O operation

Trusted Computing Base (TCB)

- TCB is the set of all hardware, firmware, and/or software components that are critical to its security
- Example: the TCB for enforcing file access permissions includes the OS kernel and filesystem drivers
- Ideally, TCBs should be non-bypassable, tamper-resistant, and verifiable

Trusted Computing Base (TCB)



<https://www.slideshare.net/k33a/trusted-platform-module-tpm>

Trusted Computing Base (TCB)

- Every system has a TCB:
 - Your reference monitor
 - Compiler
 - OS
 - CPU
 - Memory
 - Keyboard.....

Trusted Computing Base (TCB)

- Security requires the TCB be
 - Correct
 - Complete (can't be bypassed)
 - Secure (can't be tampered with)
- How can we improve the security of software, so security bugs are less likely to be catastrophic?

Trusted Computing Base (TCB)

- Two key principles behind a good TCB:
 - Keep it small and simple
 - To reduce overall susceptibility to compromise
 - Use Privilege Separation: A technique in which a program is divided into parts which are limited to the specific privileges
 - Don't give a part of the system more privileges than it needs to do its job (“need to know”)
 - Principle of “least privilege”

Trusted Computing Base (TCB)

- How can we improve the security of software, so security bugs are less likely to be catastrophic?
 - Design the software so it has a separate, small TCB.
 - Isolate privileged operations to as small a module as possible
 - any bugs outside the TCB will not be catastrophic

Other Trusted System Characteristics

- Secure startup
 - System startup is a tricky time for security, as most systems load basic I/O functionality before being able to load security functions
- Trusted path
 - An unforgeable connection by which the user can be confident of communicating directly with the OS

Other Trusted System Characteristics

- Object reuse control
 - OS clears memory before reassigning it to ensure that leftover data doesn't become compromised
- Audit
 - Trusted systems track security-relevant changes, such as installation of new programs or OS modification
 - Audit logs must be protected against tampering and deletion

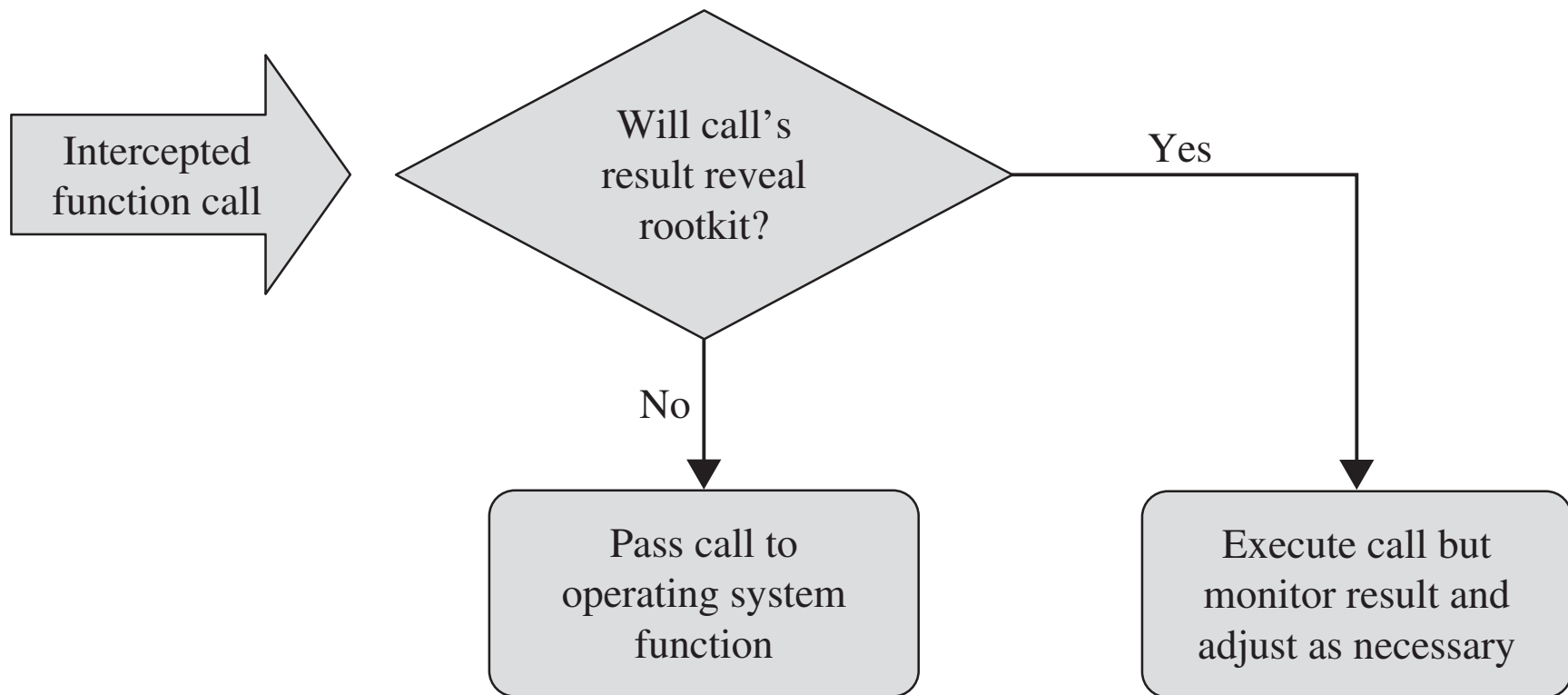
Rootkits

- A rootkit is a malicious software package that attains and takes advantage of root status
 - or effectively becomes part of the OS
- Rootkits often go to great length to:
 - avoid being discovered
 - if discovered and partially removed, reestablish themselves
 - This can include intercepting or modifying basic OS functions

Rootkit Evading Detection

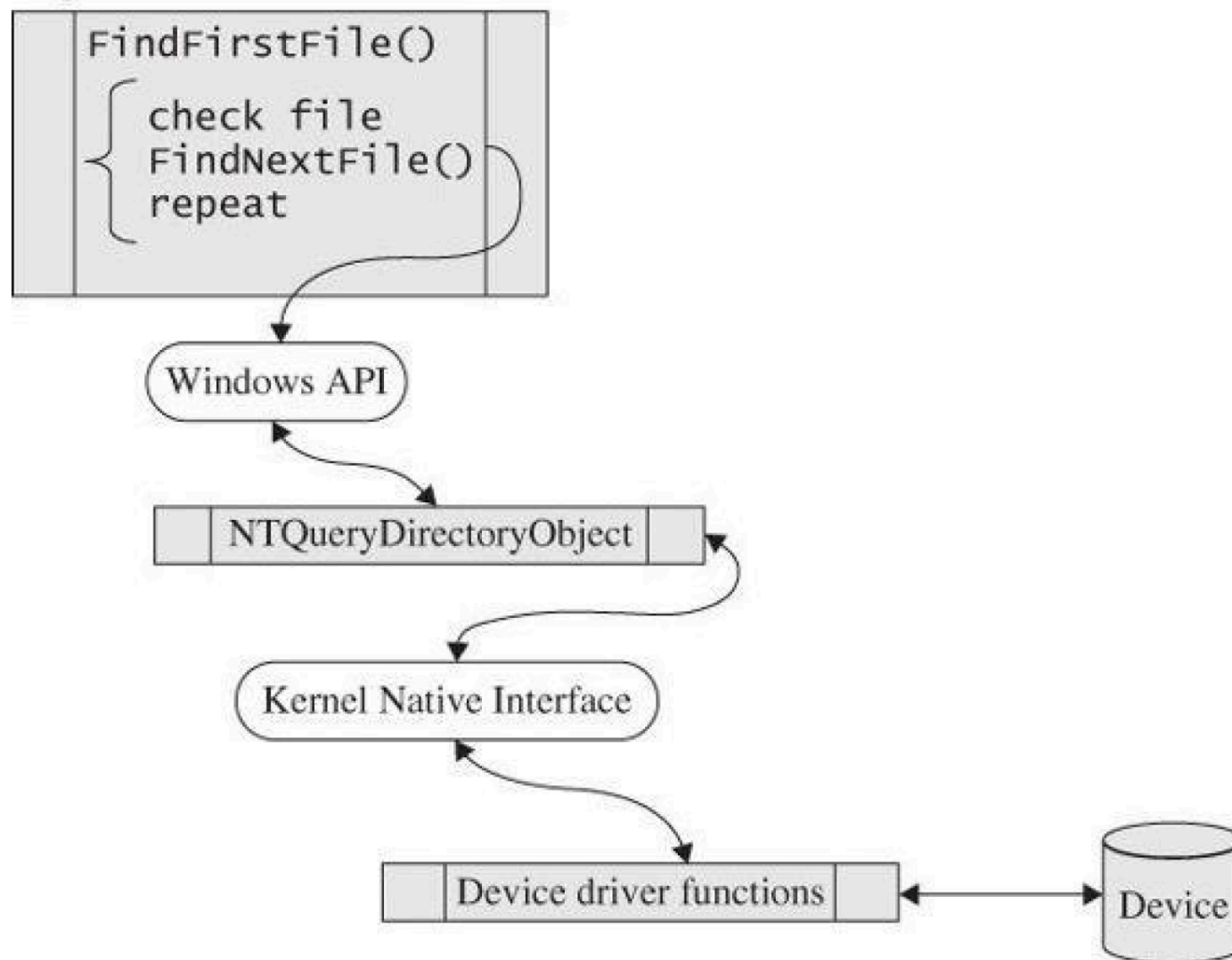
- Example: a rootkit may monitor a system call in order to intercept potentially threatening results

Rootkit Evading Detection



Example - Windows

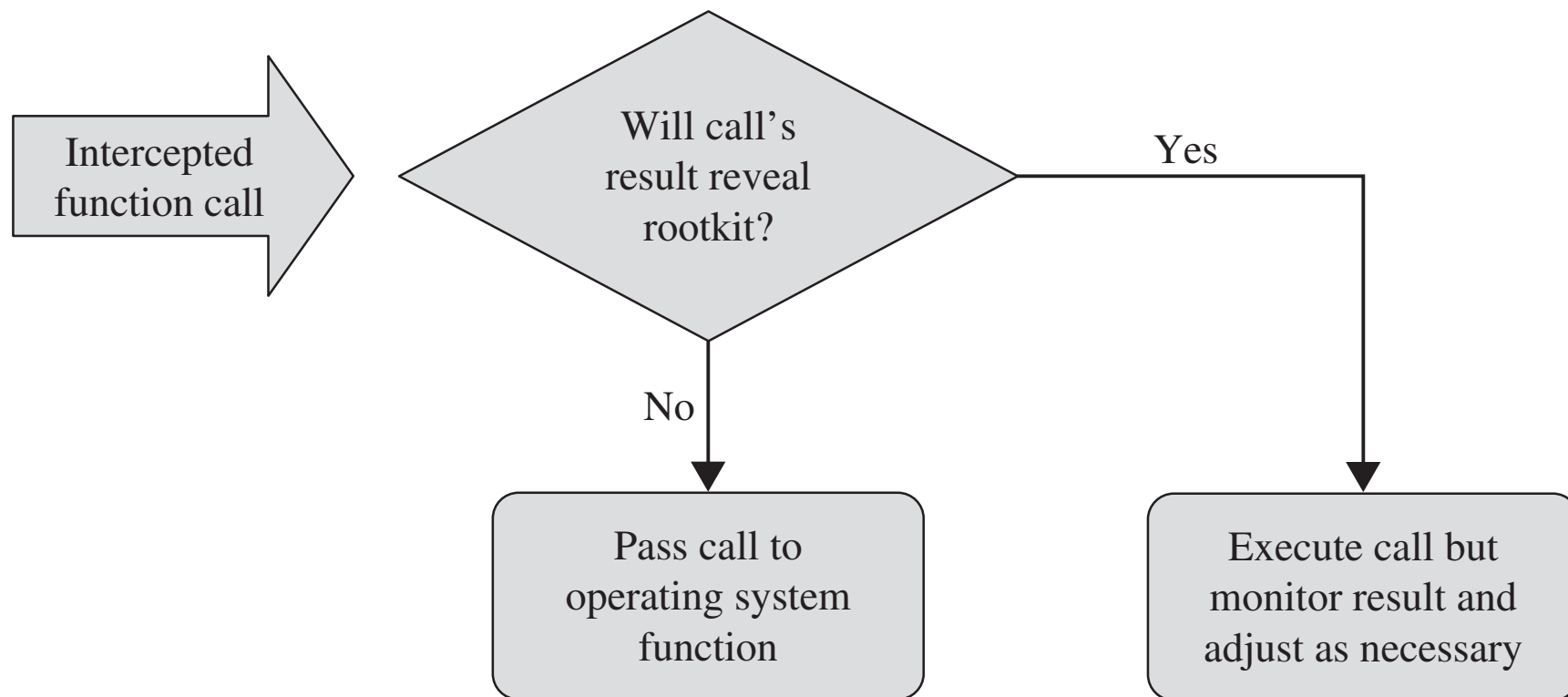
Inspect all files



Example - Windows

- A malicious file, named “mal_code.exe” may exist
- To remain invisible, the rootkit intercepts OS calls
- If the result from FindNextFile() points to mal_code.exe:
 - the rootkit skips that file and executes FindNextFile() again
 - finds the next file after mal_code.exe
 - The higher-level utility keeps the running total of file sizes for the files of which it receives information
 - so the total in the listing correctly reports all files except mal_code.exe

Rootkit Evading Detection



Summary

- OSs have evolved
 - from supporting single users and single programs to many users and programs at once
- Resources that require OS protection:
 - memory, I/O devices, programs, and networks
- OSs use layered and modular designs
 - for simplification
 - to separate critical functions from noncritical ones

Summary

- Resource access control can be enforced in a number of ways
 - including virtualization, segmentation, hardware memory protection, and reference monitors
- Rootkits are malicious software packages that attain root status
 - or effectively become part of the OS

- Questions?

