

Security in and for Operating Systems

Jun Li

lijun@cs.uoregon.edu

Learning Objectives

- Security methods of ordinary OS
 - Memory and address protection
 - Control of access to general objects
 - File protection
 - User authentication
- Trusted OS
 - Concepts
 - Security policies and models
 - Trusted OS design:
 - kernelized design, separation/isolation, virtualization

Security Methods of Ordinary OS

Protected Objects

- Memory
- I/O devices (sharable, serially usable)
- Networks
- Sharable programs and subprocedures
- Sharable data

Security Methods of OS

- Separation: One user's objects separate from other users
- Physical separation
 - E.g., separate printers
- Temporal separation
 - Processes executing at different times
- Logical separation
 - Users operate under the illusion that no other processes exist
- Cryptographic separation
 - Data and computations unintelligible to outside processes

Levels of Separation

more difficult
↓

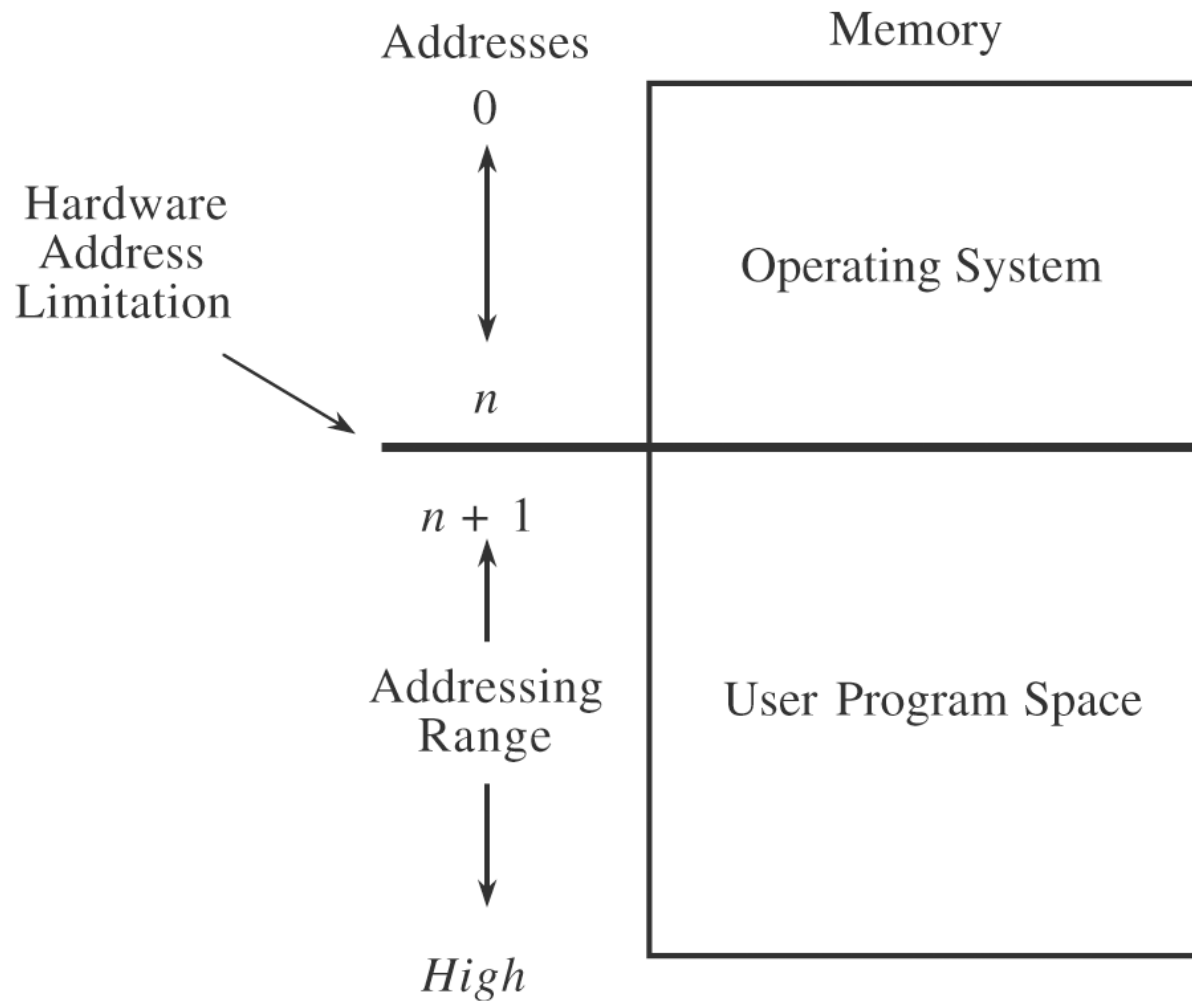
- Do not protect
- Isolate: Processes are unaware of each other
 - Each has its own address space, files, and other objects
- Share all or nothing
 - The owner of an object make it either public or private
- Discretionary access control
 - The owner controls the access to its objects
- Mandatory access control
 - The O.S. controls the access to objects
- Limit use of an object
 - Not just the access, but also the usage made after access

Memory and Address Protection

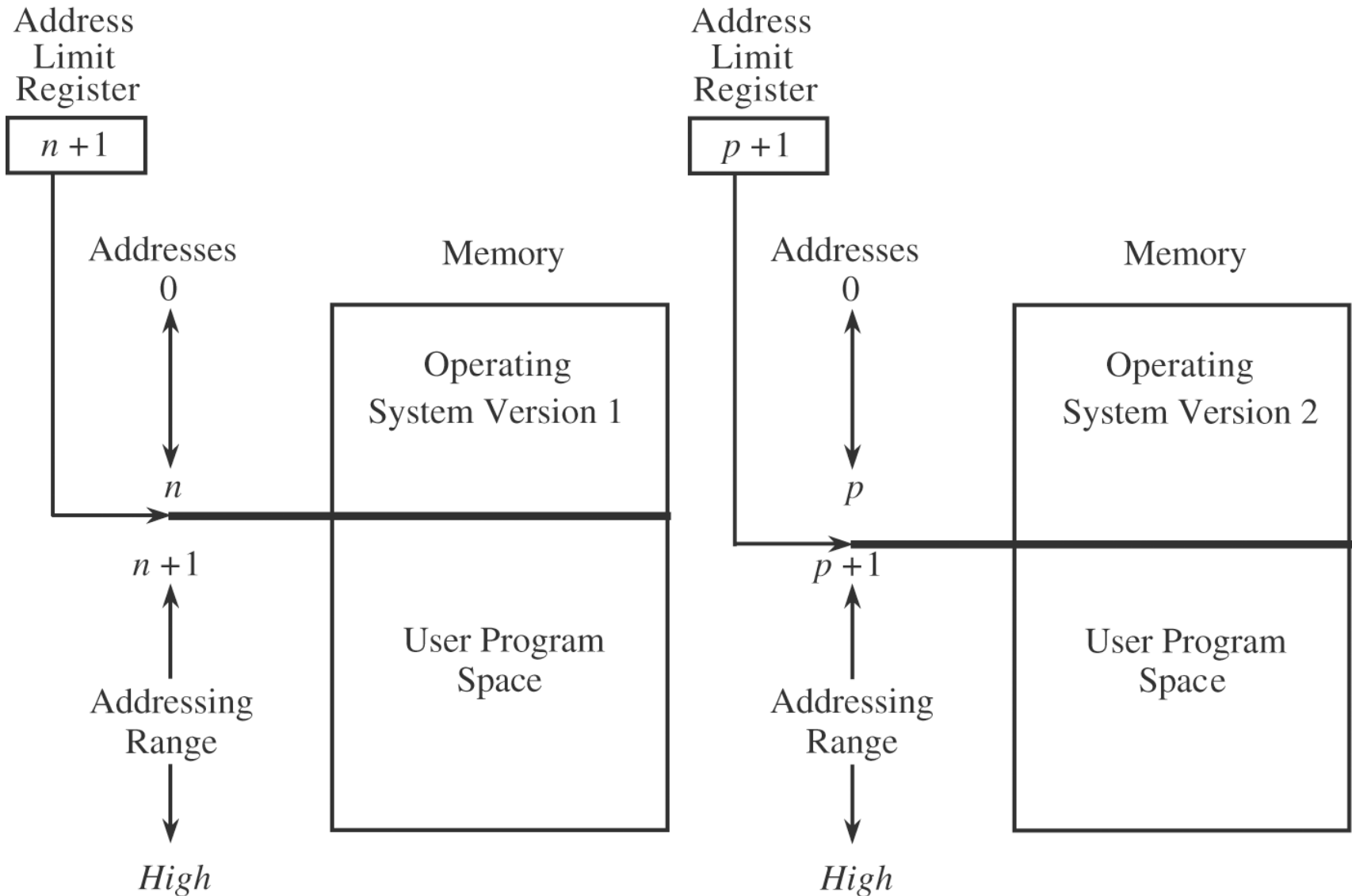
Every access goes through certain HW points

- Fence
- Relocation
- Base/Bound Registers
- Tagged Architecture
- Segmentation
- Paging
- Combined Paging with Segmentation

Fence - Predefined



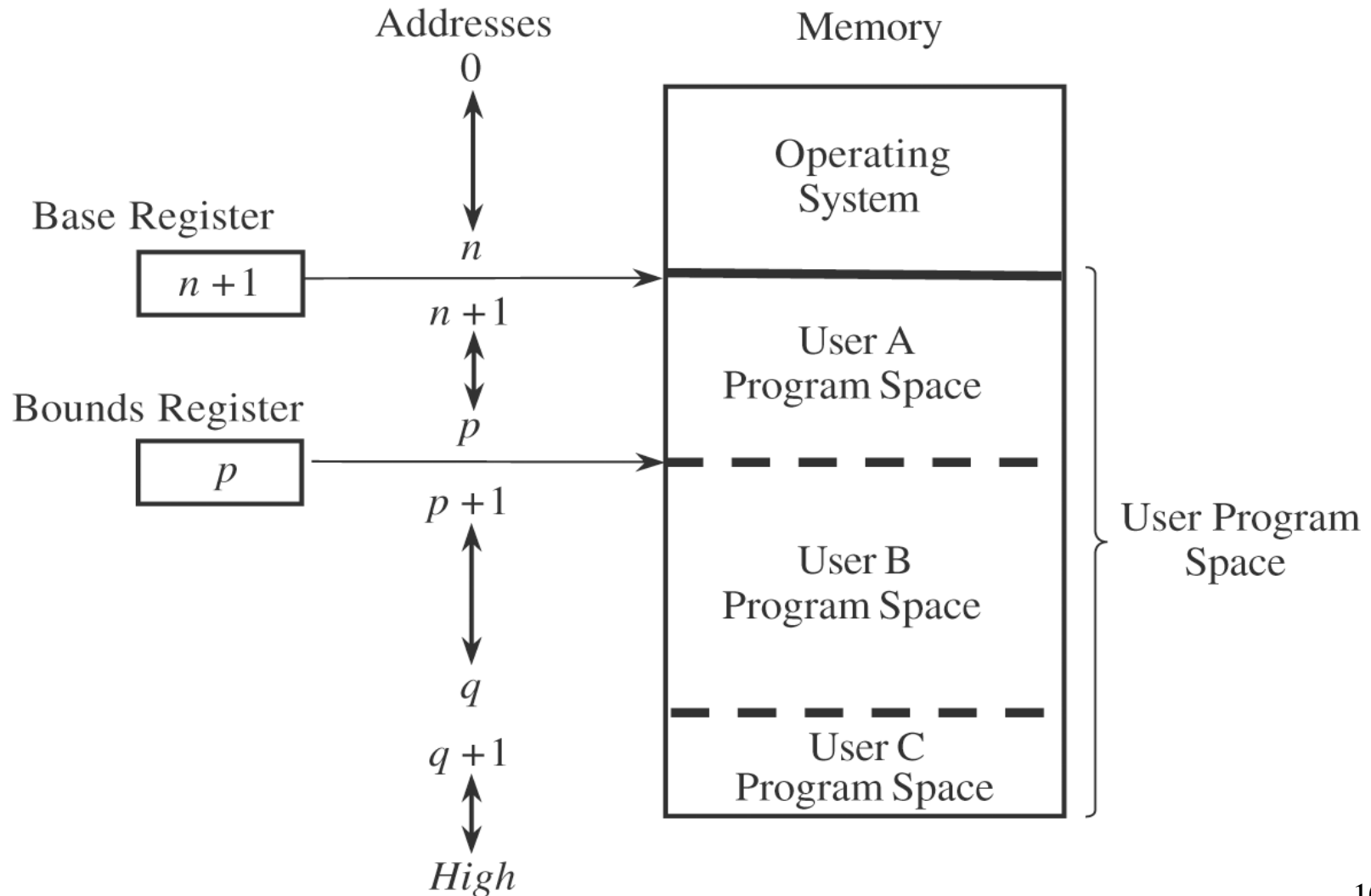
Fence – Fence Register

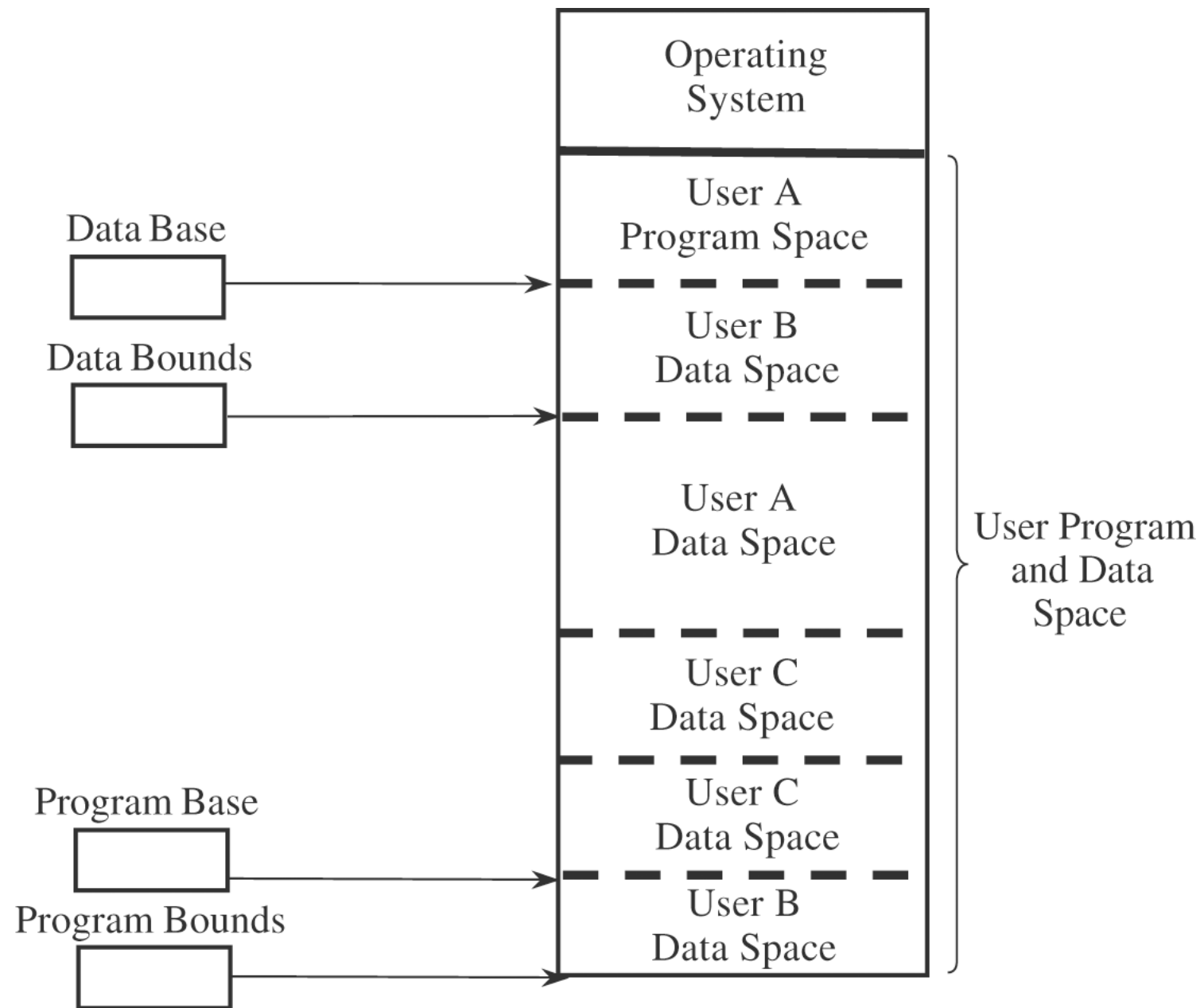


Relocation







- Program written as if it began at address 0
- Adding a constant relocation factor when loading the program into memory
- The fence register can be a hardware relocation device

Base/Bounds Registers



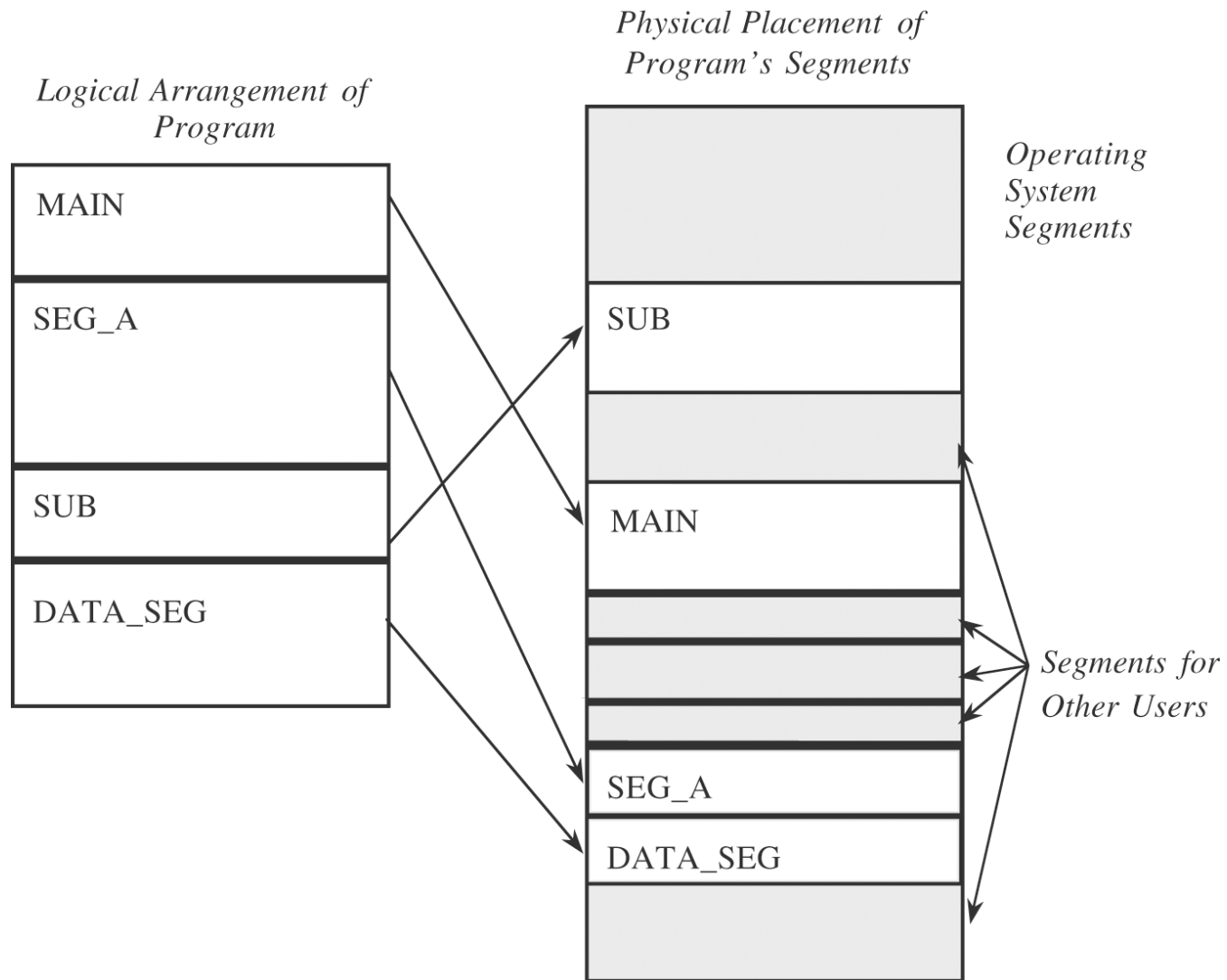


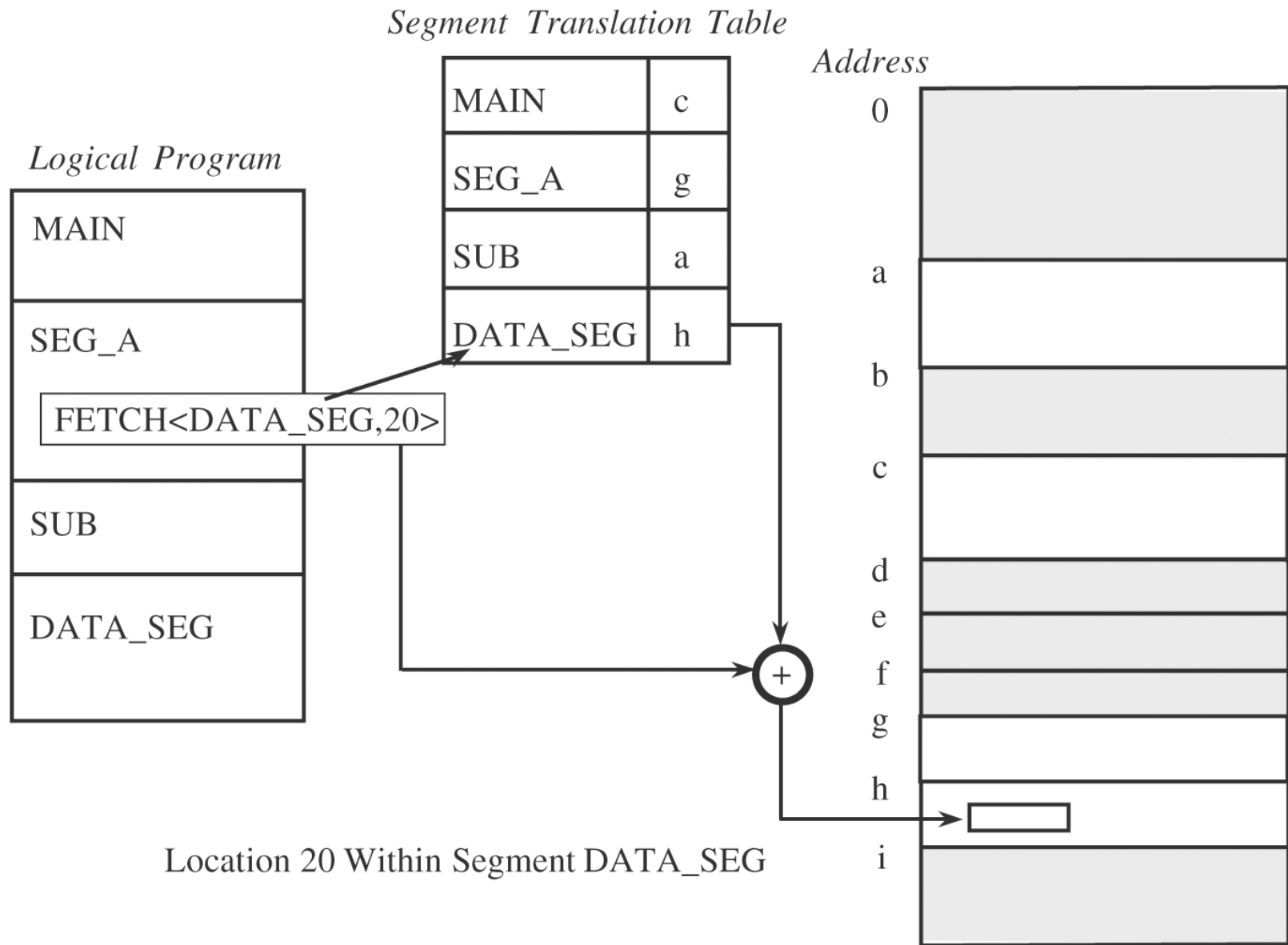
Tagged Architecture

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

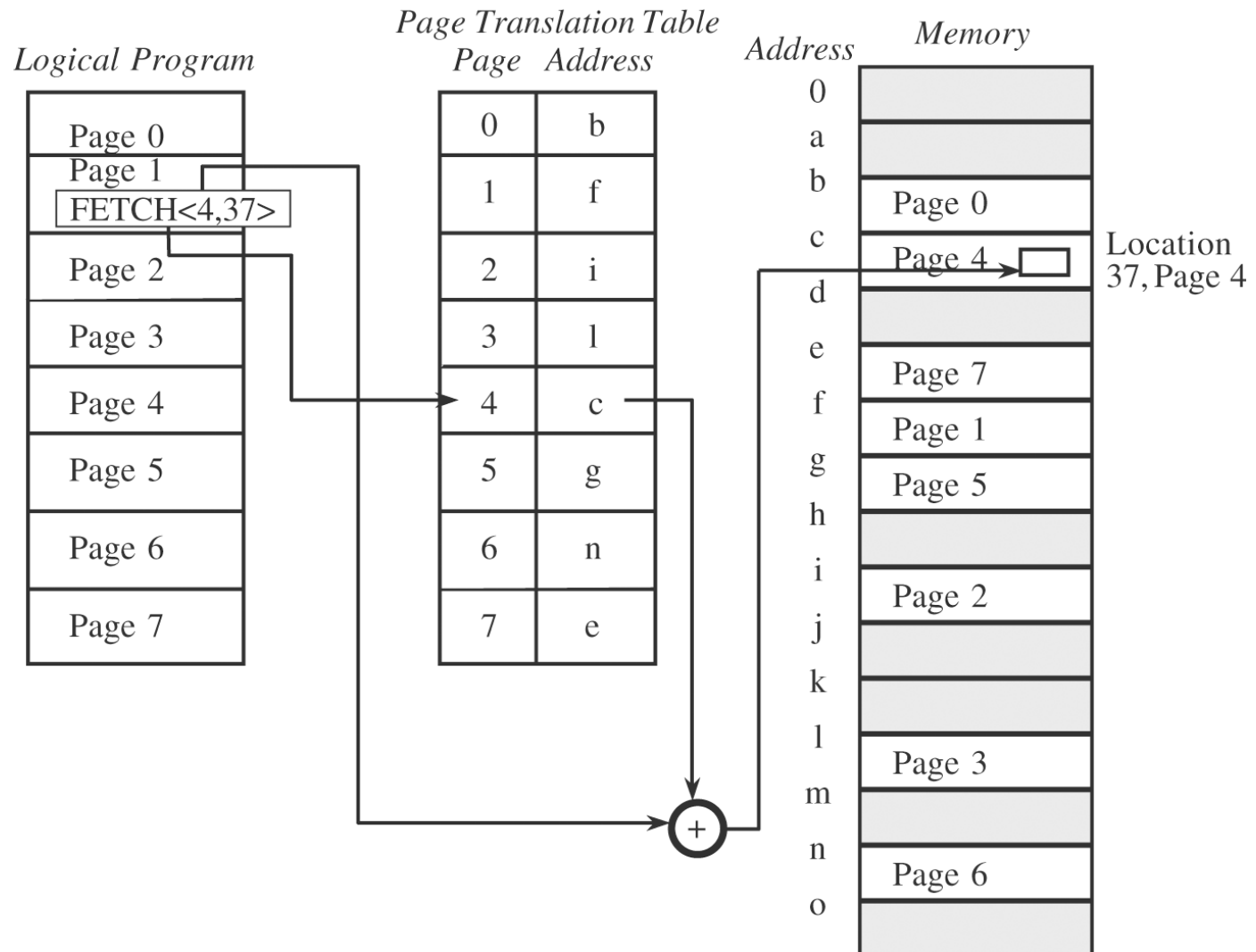
Code: R = Read-only RW = Read/Write
X = Execute-only

Segmentation

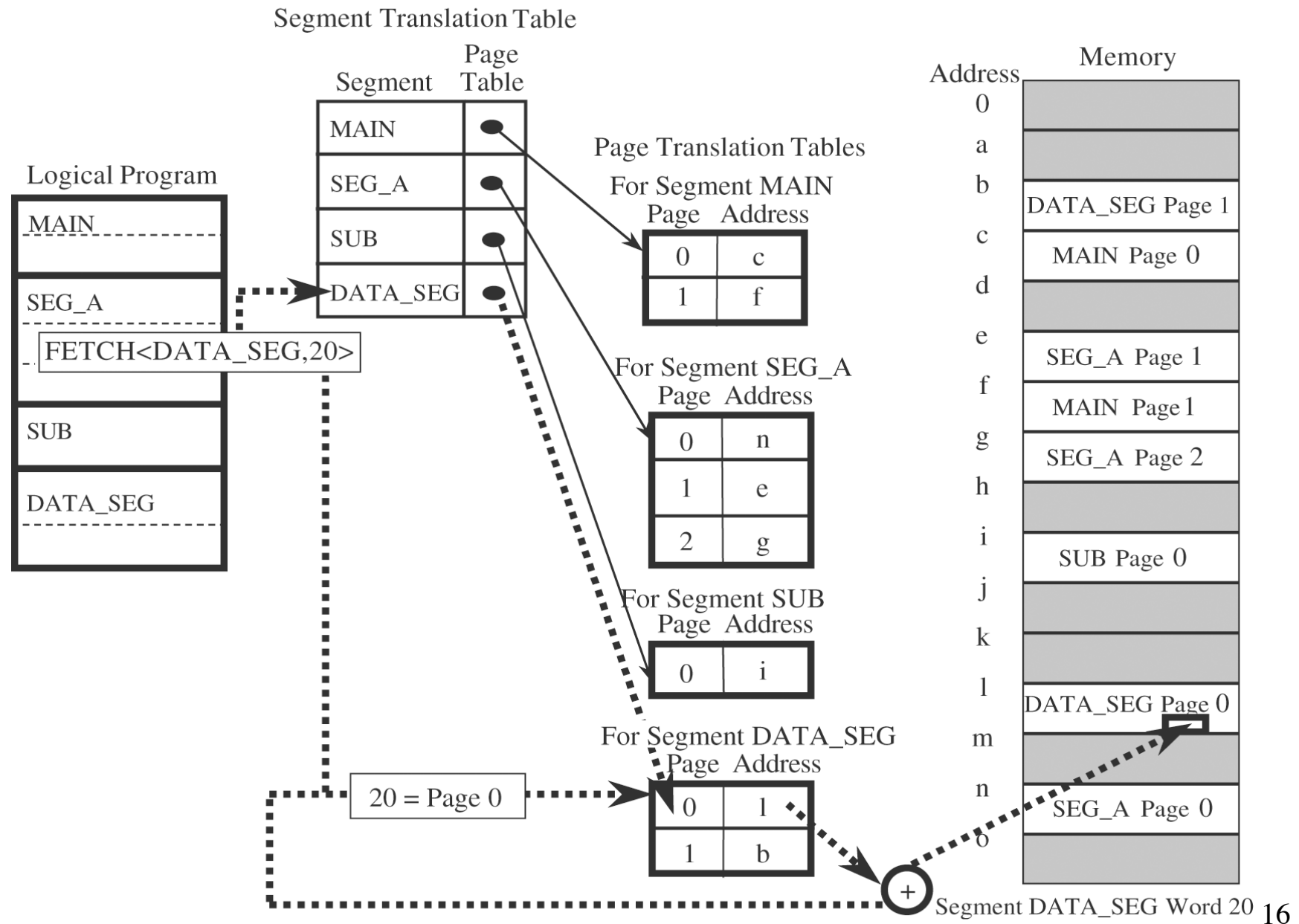




Paging



Combined Paging with Segmentation



Control of Access to General Objects

- Memory is a special object
- Objects can be:
 - A memory space
 - A file or data set on an auxiliary storage device
 - An executing program in memory
 - A directory of files
 - A hardware device
 - A data structure, such as a stack
 - A table of the operating system
 - Privileged instructions
 - Passwords and user authentication mechanism
 - The protection mechanism itself
- Refer to Chapter 2 on Authentication and Access Control

Designing Trusted Operating Systems

4 Underpinnings of a *Trusted* OS

- **Security policy:** requirement statements of what an OS should do and how it should do it, usually a set of rules on what to be secured and why.
- **Model:** a representation of the policy the OS will enforce (and compared with the security policy to ensure the security needs are met)
- **Design:** design/implement the OS w.r.t the model
- **Trust:** *features* (the OS has all the functionalities to enforce the security policy) and *assurance* (it will do so correctly and effectively)

Secure vs. Trusted

- *Either-or*: something either is or is not secure
- Property of *presenter*
- *Asserted* based on product characteristics
- *Absolute*: not qualified as to how used, where, when, or by whom
- *A goal*
- *Graded*: degrees of trustworthiness
- Property of *receiver*
- *Judged* based on evidence and analysis
- *Relative*: viewed in context of use
- *A characteristic*

Security Policies

- Military Security Policy
- Commercial Security Policies
 - Clark-Wilson
 - Separation of Duty
 - Chinese Wall

Military Security Policy: object

- Each piece of information is ranked at a sensitivity level: *unclassified, restricted, confidential, secret, top secret*
- Each piece of information is also associated with *compartments*
 - Need-to-know principle
- **class or classification** of a piece of information: *<rank; compartments>*

Military Security Policy: subject

- **clearance** of a subject: $\langle rank; compartments \rangle$
 - The subject is trusted to access information up to a certain level of sensitivity (*rank*)
 - The subject needs to know certain categories of information (compartments)
- A subject s dominates an object o ($s \geq o$) *iff*
 $rank(s) \geq rank(o)$ *and*
 $Compartments(s) \geq Compartments(o)$

Military Security Policy

- A subject s can read an object o only if s dominates o ($s \geq o$)
- Appropriate for a setting in which access is rigidly controlled by a central authority

Models of Security

- Test a particular policy
 - Completeness
 - Consistency
- Document a policy
- Help conceptualize and design an implementation
- Check whether an implementation meets its requirements

Models

- Models for multilevel security
 - Bell-La Padula Confidentiality Model
 - Biba Integrity Model
- Models proving theoretical limitations of security systems
 - Graham-Denning
 - Harrison-Ruzzo-Ullman
 - Take-Grant
- We focus on models for multilevel security

Bell-La Padula Confidentiality Model

- The system has a set of subjects $S=\{s\}$ and a set of objects $O=\{o\}$
- Each subject s or object o has a **security class** $C(s)$, $C(o)$
- Enforces two properties on the secure flow of information

Properties

- **Simple Security Property.** A subject s can read an object o only if $C(s) \geq C(o)$.
 - Prevents *read-up*
- ***-property** (called “star property”). A subject s who can read an object o can write to p only if $C(o) \leq C(p)$.
 - Prevents *write-down*

Biba Integrity Model

- To prevent inappropriate modification of data
- Subjects and objects are ordered by an integrity classification scheme. $I(s)$, $I(o)$.
- **Simple integrity property.** Subject s can modify (write) object o only if $I(s) \geq I(o)$.
- **Integrity *-property.** If subject s has *read* access to object o with integrity level $I(o)$, s can write object p only if $I(o) \geq I(p)$.

Graham-Denning Model

- Model the access control mechanisms of a protection system
- A set of subjects S , a set of objects O , a set of rights R , and an access control matrix A .
- 8 primitive protection rights
 - Create object, create subject, delete object, delete subject
 - Read access right, grant access right, delete access right, transfer access right

Harrison-Ruzzo-Ullman

- A variation of the Graham-Denning model
- Based on **commands** with conditions and primitive operations

```
command name( $o_1, o_2, \dots, o_k$ )  
    if       $r_1$  in  $A[s_1, o_1]$  and  
            $\dots$   
            $r_m$  in  $A[s_m, o_m]$   
    then  
            $op_1$   
            $\dots$   
            $op_n$   
end
```

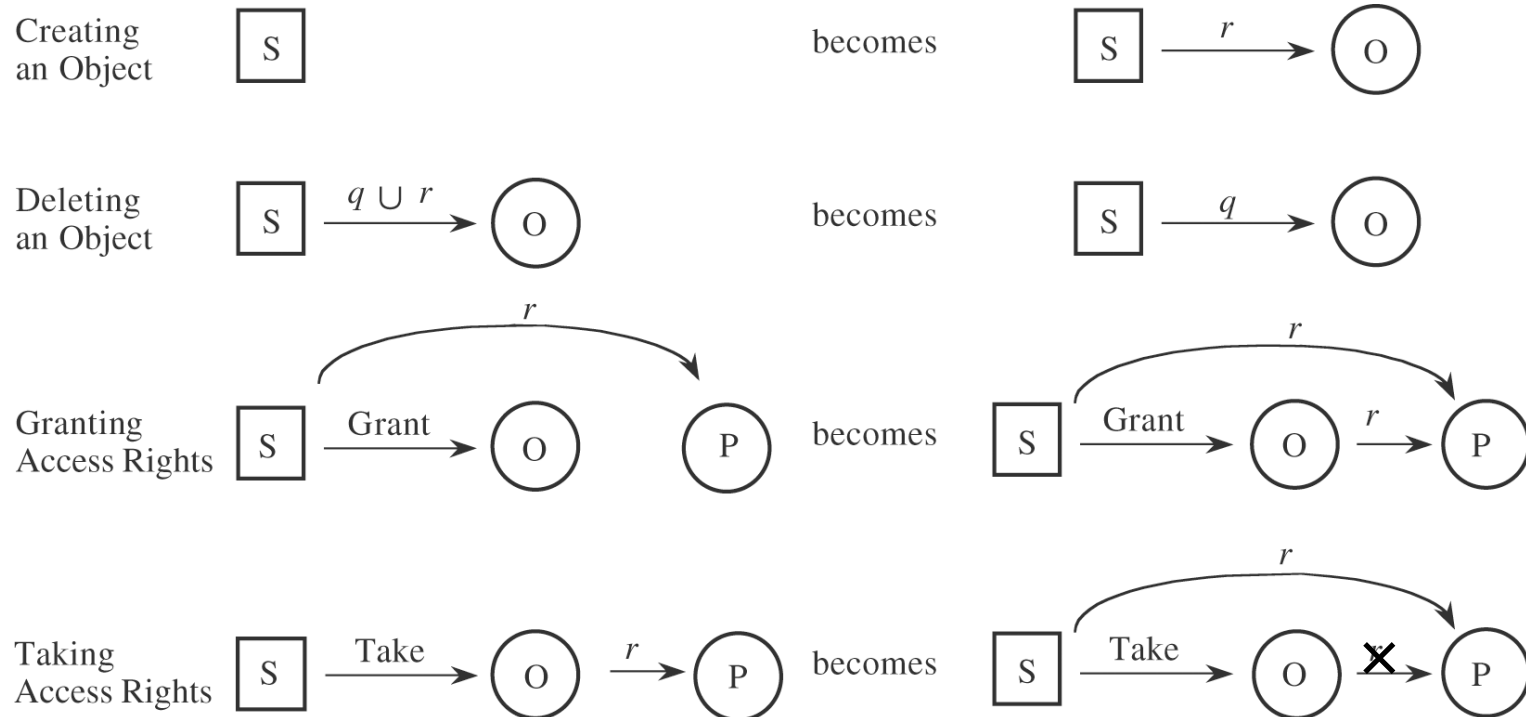
- Primitive operations:
 - create/destroy subject/object
 - enter/delete right r into/from $A[s,o]$

HRU Results

- In the modeled system, in which commands are restricted to a single operation each, it *is* possible to decide whether a given subject can even obtain a particular right to an object.
- If commands are not restricted to one operation each, it is *not* always decidable whether a given protection system can confer a given right.

Take-Grant Model

- 4 primitive operations by a subject s : $\text{create}(o, r)$, $\text{revoke}(o, r)$, $\text{grant}(o, p, r)$, $\text{take}(o, p, r)$.



Take-Grant Results

Decidable on certain protection questions in reasonable time:

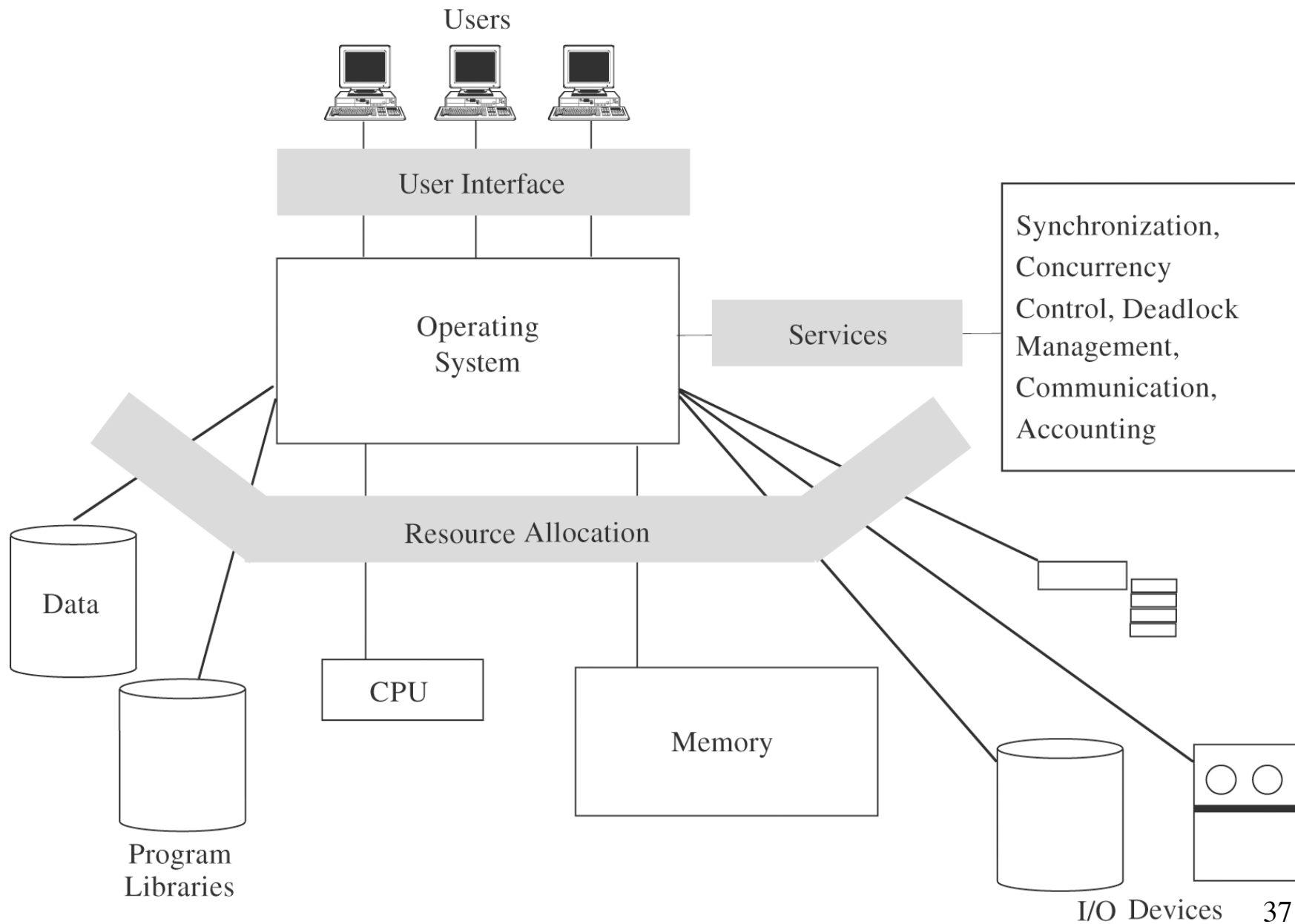
1. Can we decide whether a given subject can share an object with another subject?
2. Can we decide whether a given subject can steal access to an object from another subject?

Design of a Trusted OS

- Principles
- Security features of an ordinary OS
- Security features of a trusted OS
- Kernelized design
- Separation/Isolation
- Virtualization
- (Layered design)

Design Principles by Saltzer & Schroeder

- Least privilege
- Economy of mechanism (small, simple, straightforward)
- Open design
- Complete mediation
- Permission based (default is denial of access)
- Separation of privilege (object access via >1 condition)
- Least common mechanism (minimal sharing)
- Ease of use



Security Features of a Trusted OS

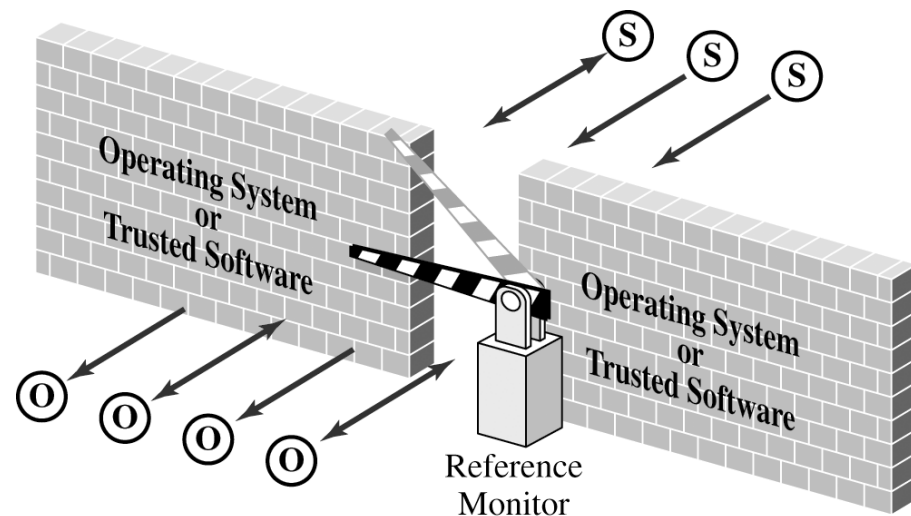
- User identification and authentication
- Mandatory access control (MAC)
- Discretionary access control (DAC)
- Object reuse protection
- Complete mediation
- Trusted path
- Audit
- Intrusion detection

Kernelized Design

- A **security kernel** is responsible for enforcing the security mechanisms of the entire OS
 - Coverage (complete mediation)
 - Separation (from the rest of OS and applications)
 - Unity (a single set of code thus easier to debug)
 - Modifiability (easy to modify)
 - Compactness (small)
 - Verifiability (friendly to rigorous analysis)

Reference Monitor

- The most important part of a security kernel
- It must be:
 - Tamperproof
 - Unbypassable
 - analyzable

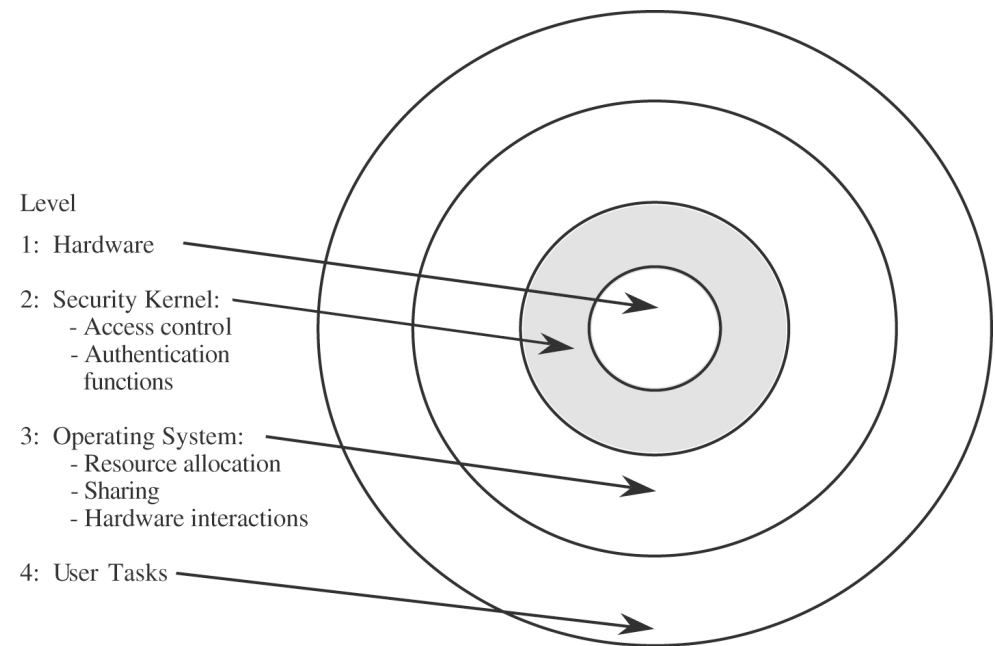
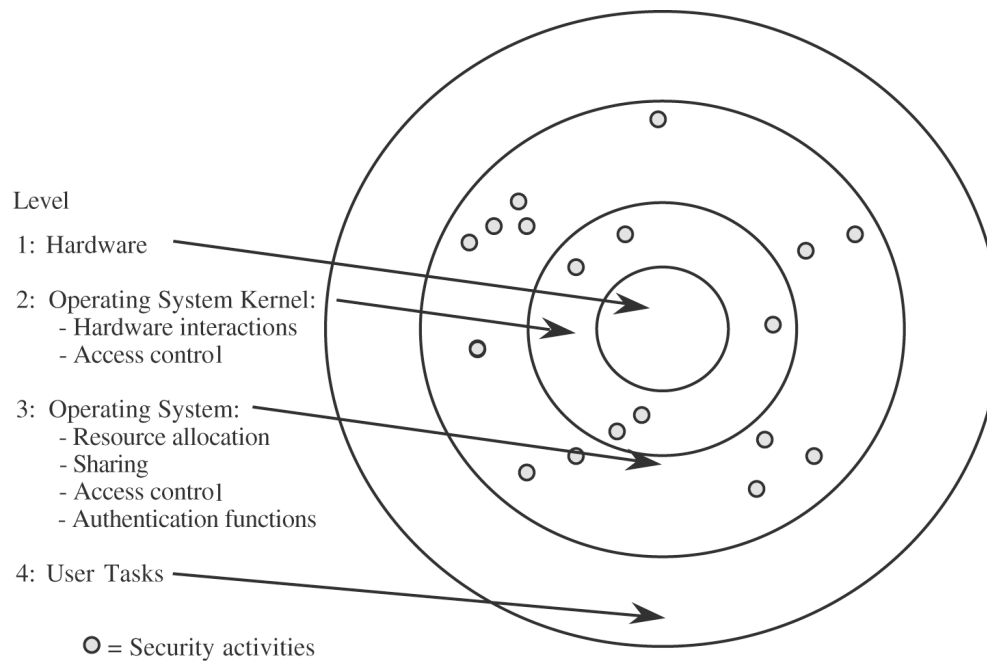


Pfleeger/Pfleeger Fig. 05-12 rev 9/30/02

Trusted Computing Base (TCB)

- Everything in a trusted OS necessary to enforce the security policy
 - Trust in the security of the whole system depends on the TCB
 - Non-TCB part can be malicious w/o affecting security enforcement
- Usually include:
 - Hardware
 - Protected memory
 - Some notion of processes and IPC
 - Primitive files (security-related)

TCB Implementation



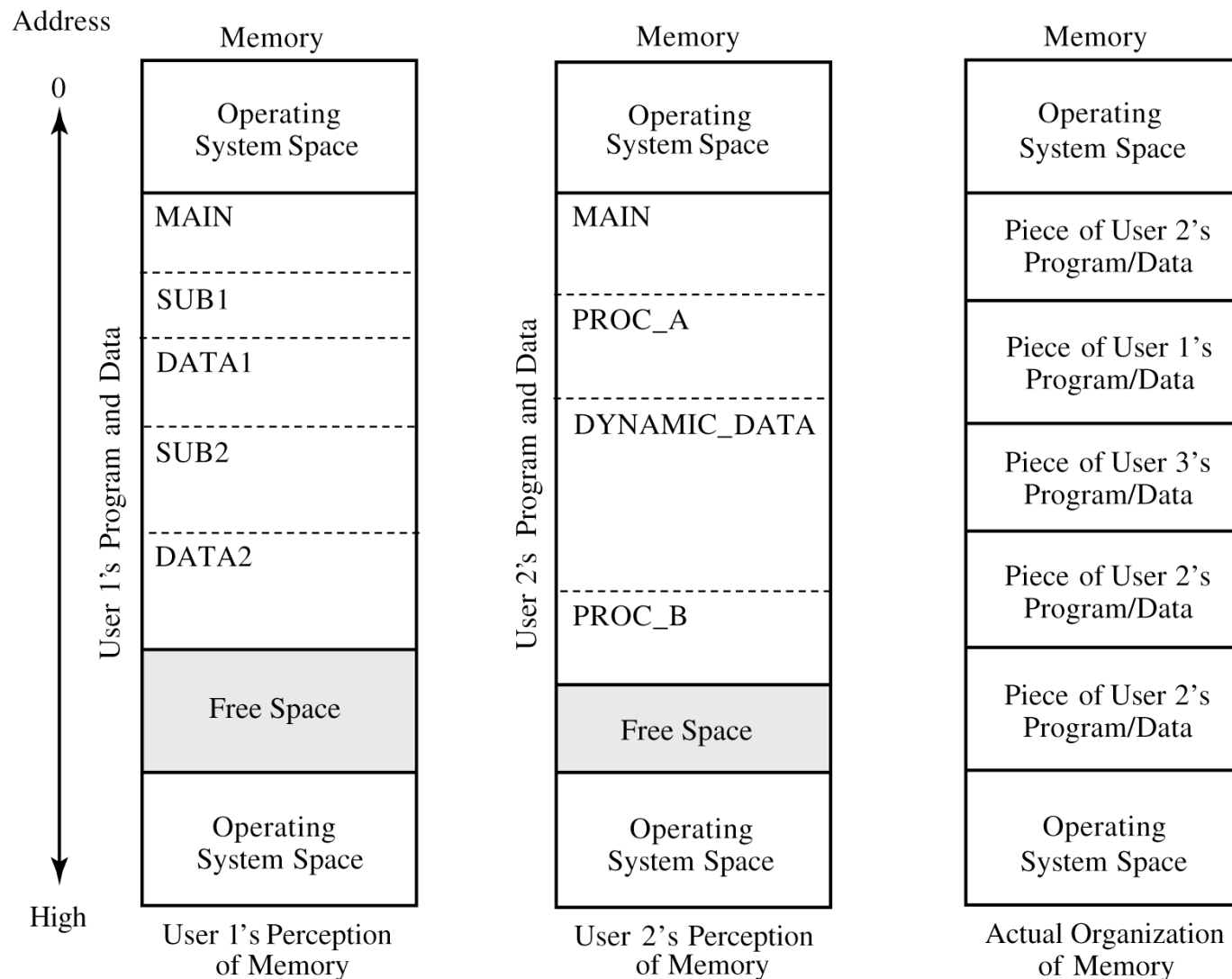
Separation

- Physical separation
- Temporal separation
- Cryptographic separation
- Logical separation

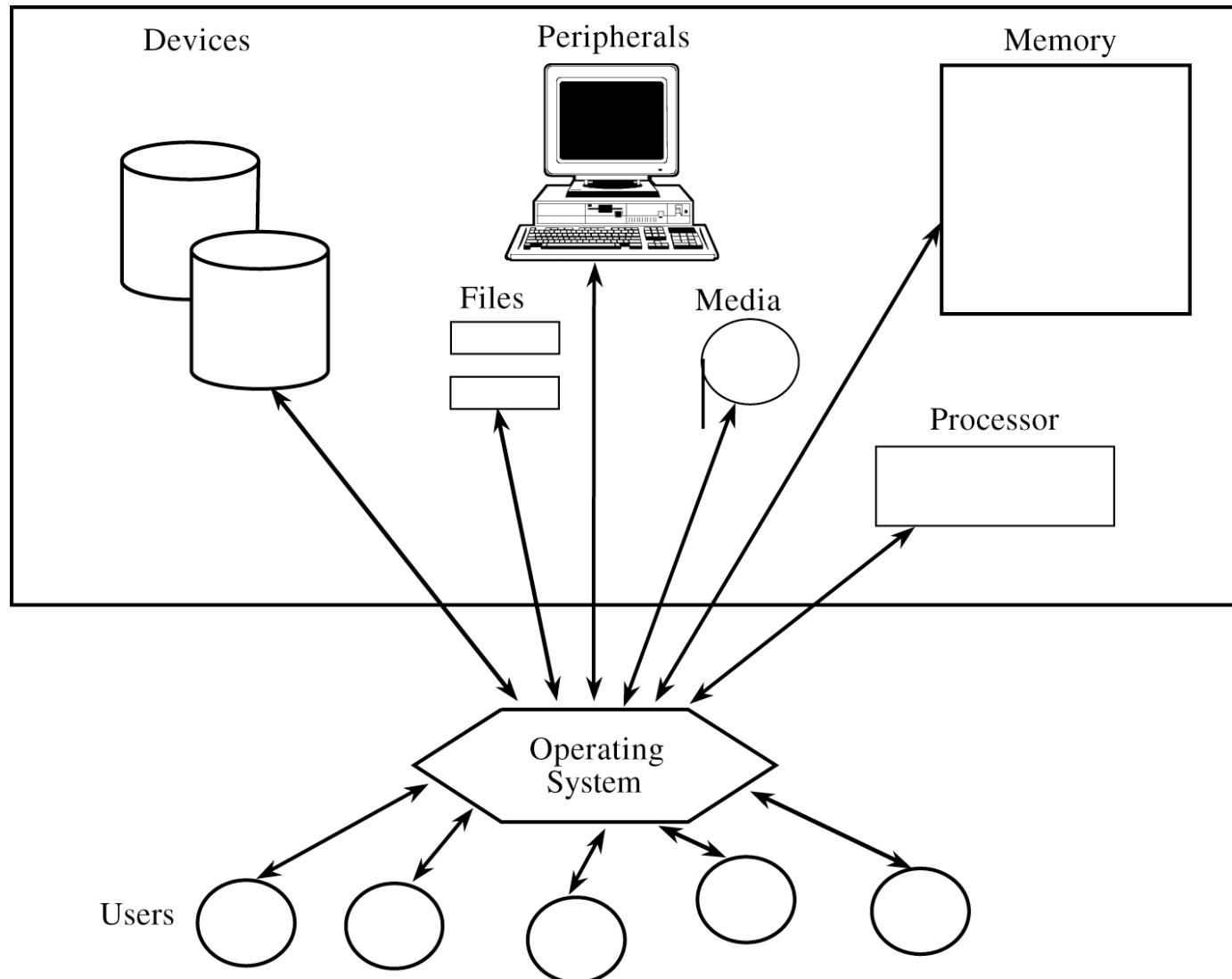
Virtualization

- Multiple Virtual Memory Spaces
- Virtual Machines

Multiple Virtual Memory Spaces



Virtual Machines



Virtual Machines

