System Protection

- Goals of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection

Protection

- Protection must ensure that only those processes that have gained proper authorization from the OS can operate on memory segments, the CPU, and other resources.
- Operating system consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations.
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.
- Enforcement of the policies governing resource usage.
- A protection system must have the flexibility to enforce a variety of policies that can be declared to it.

Goals of Protection

- Provide mechanisms for enforcement of policies.
  - Mechanisms determine how some thing will be done. Policies decide what will be done.
- Policies may change over time and can be decided by application programmer or system programmer.
- In this chapter we discuss the protection mechanism the OS should provide so that the application designers can design their own protection software.

Guiding principle

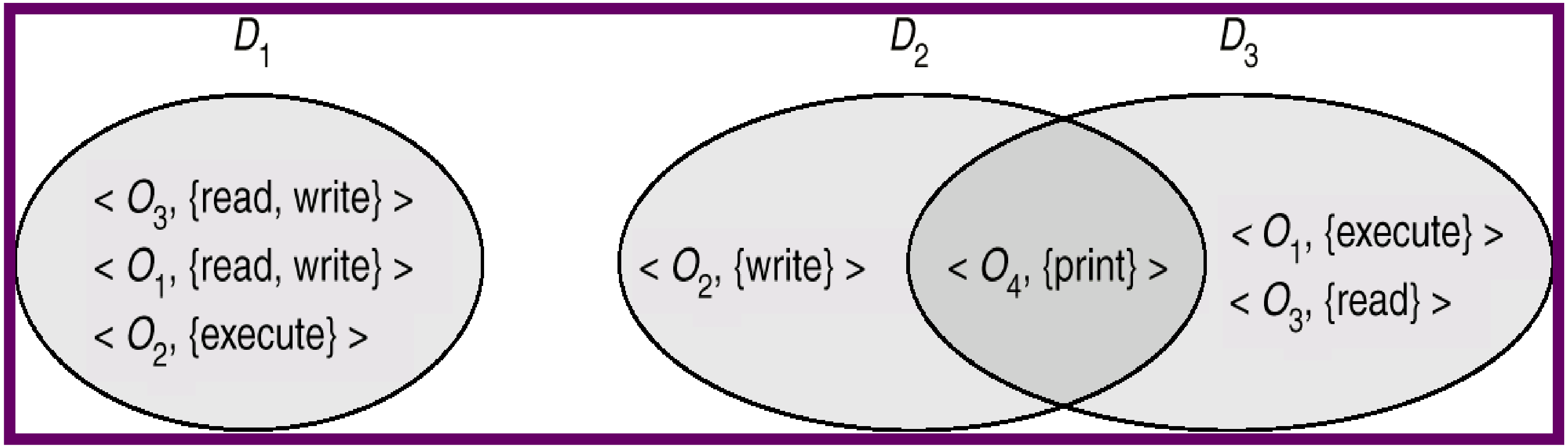
- Principle of least privilege
  - **Dictates that programs, users and even systems be given just enough privileges to perform their task.**
- Uses fine grained access controls
- Beneficial to create a audit trial
- Audit trial allows tracing of violations.

Domain Structure

- Computer system is a collection of processes and objects
- Objects are abstract data types
  - Hardware objects (cpu, memory segments, printers..) and software objects (files, programs, and semaphores).
- Each object is accessed by a well-defined and meaningful operations.
- Operations depend on the object
  - A CPU may only be executed on. Memolry systems may read or written. Files can be open, close, read, write, executed, deleted.
- The process should access only those resources which it requires or allowed to access.
  - Need to know principle

Domain Structure

- A process operates within a **protection domain**.
- Domain specifies the resources a process may access
- Access right= The ability to execute an operation on an object
- Domain is a collection of access rights.
- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object.



Ways of realizing domains

- Each user may be a domain
  - Set of objects that can be accessed depends up on the identity of the user.
- Each process may be a domain
  - Set of objects that can be accessed depends on the identity of the process.
- Each procedure may be a domain
  - Set of objects that can be accessed corresponds to the local variables defined within the procedure.

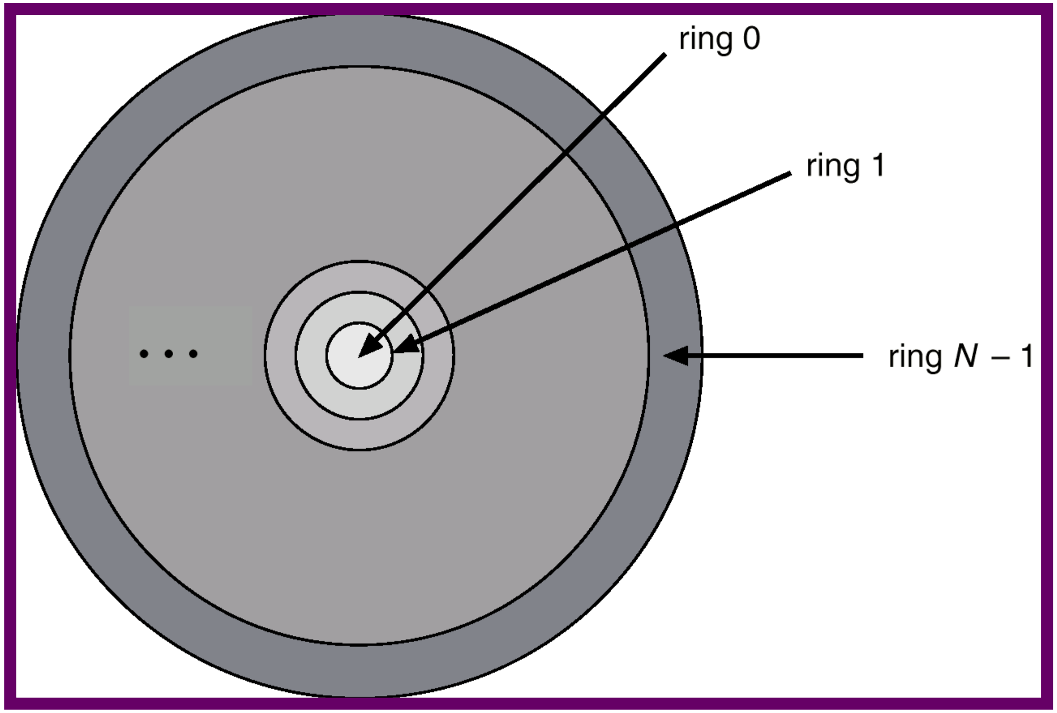
Domain Implementation (UNIX)

- System consists of 2 domains:
  - User
  - Supervisor

- UNIX
  - Domain = user-id
  - Domain switch accomplished via file system.
    - Each file has associated with it a domain bit (setuid bit).
    - When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset.

Domain Implementation (Multics)

- Protection domains are organized hierarchically into a ring structure.
- Let D_i and D_j be any two domain rings.
- If $j < i \Rightarrow D_i \subseteq D_j$
- Protection system is more complex and less efficient.



Multics Rings

Access Matrix Method

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- Each entry in the matrix consists of set of access rights.
- *Access(i, j)* is the set of operations that a process executing in Domain$_i$ can invoke on Object$_j$
- Access Matrix gives flexibility to implement various policies.

Access Matrix

object domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figure A

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix.
- Can be expanded to dynamic protection.
  - Operations to add, delete access rights.
  - Special access rights:
    - ✓ *owner of* O_i
    - ✓ *copy op from* O_i *to* O_j
    - ✓ *control –* D_i *can modify* D_j *access rights*
    - ✓ *transfer – switch from domain* D_i *to* D_j

Use of Access Matrix (Cont.)

- Access matrix design separates mechanism from policy.
  - Mechanism (how something will be done)
    - ✓ Operating system provides access-matrix + rules.
    - ✓ If ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.
  - Policy (what will be done)
    - ✓ User dictates policy.
    - ✓ Who can access what object and in what mode.

Implementation of Access Matrix

- Each column = Access-control list for one object
  Defines who can perform what operation.

Domain 1 = Read, Write
Domain 2 = Read
Domain 3 = Read

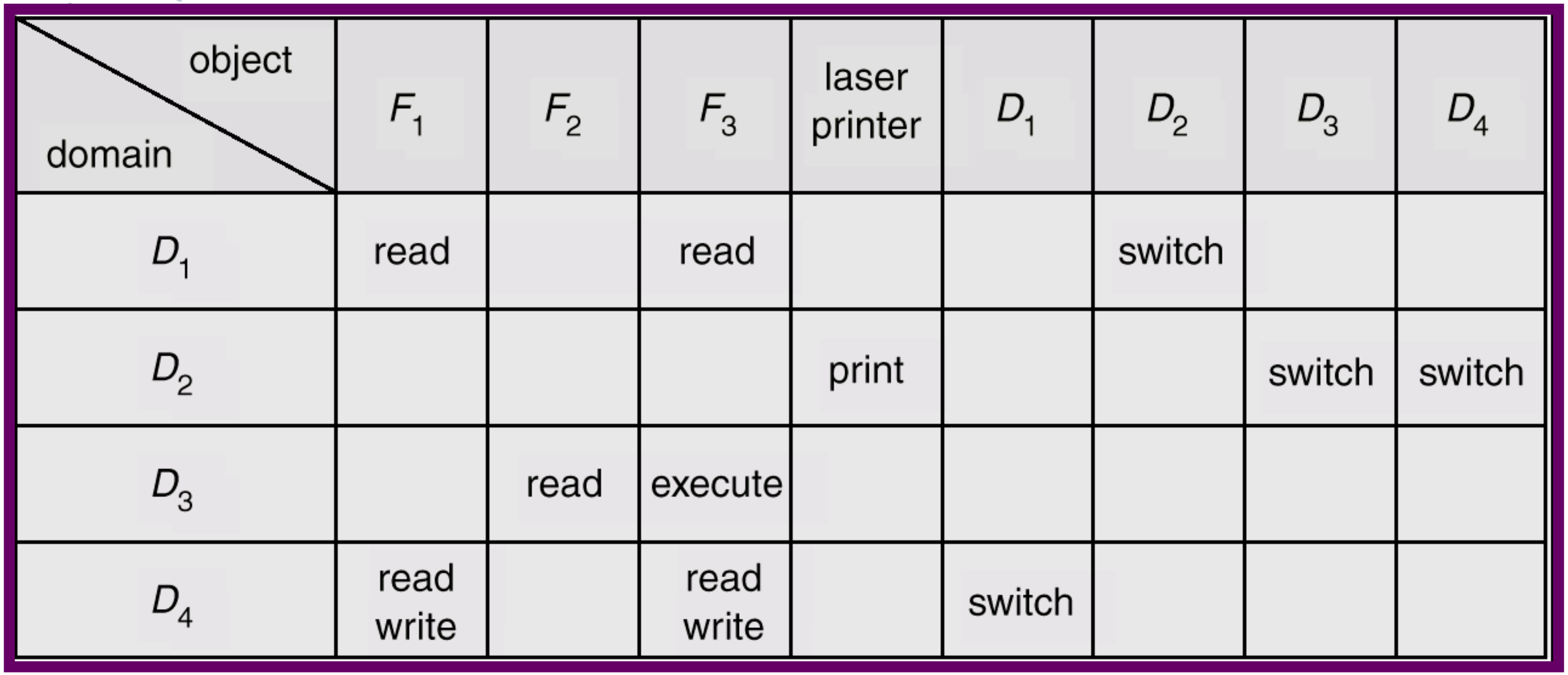
⋮

- Each Row = Capability List (like a key)
  Fore each domain, what operations allowed on what objects.

Object 1 – Read
Object 4 – Read, Write, Execute
Object 5 – Read, Write, Delete, Copy

Access Matrix of Figure A With Domains as Objects

A process in domain D4 can switch to D1, and one in domain D1 can switch to D2

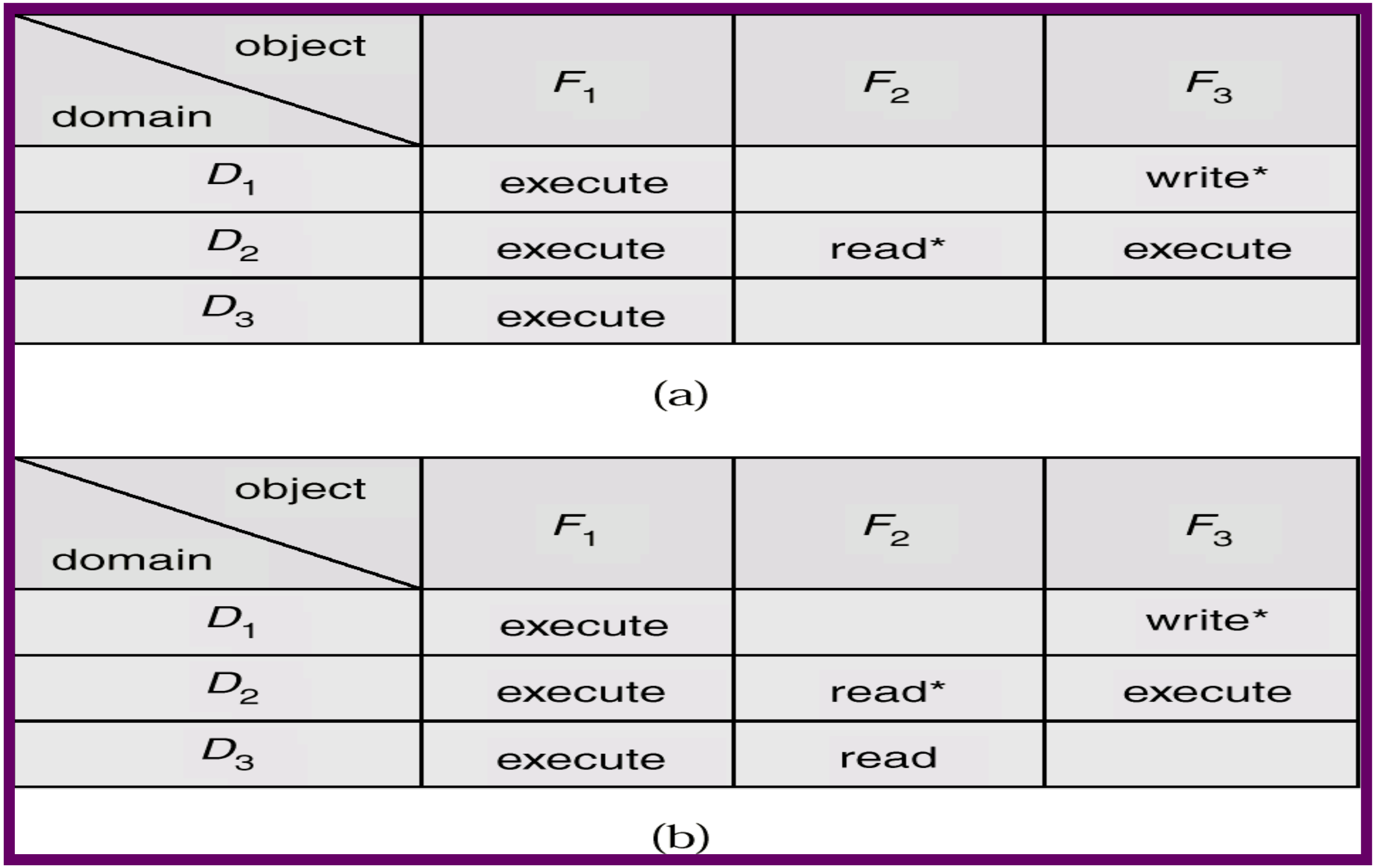


object domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Figure B

Access Matrix with *Copy* Rights

- A process executing in domain D2 can copy the read operation in to any entry associated with F2. Propagation may be limited.



object domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

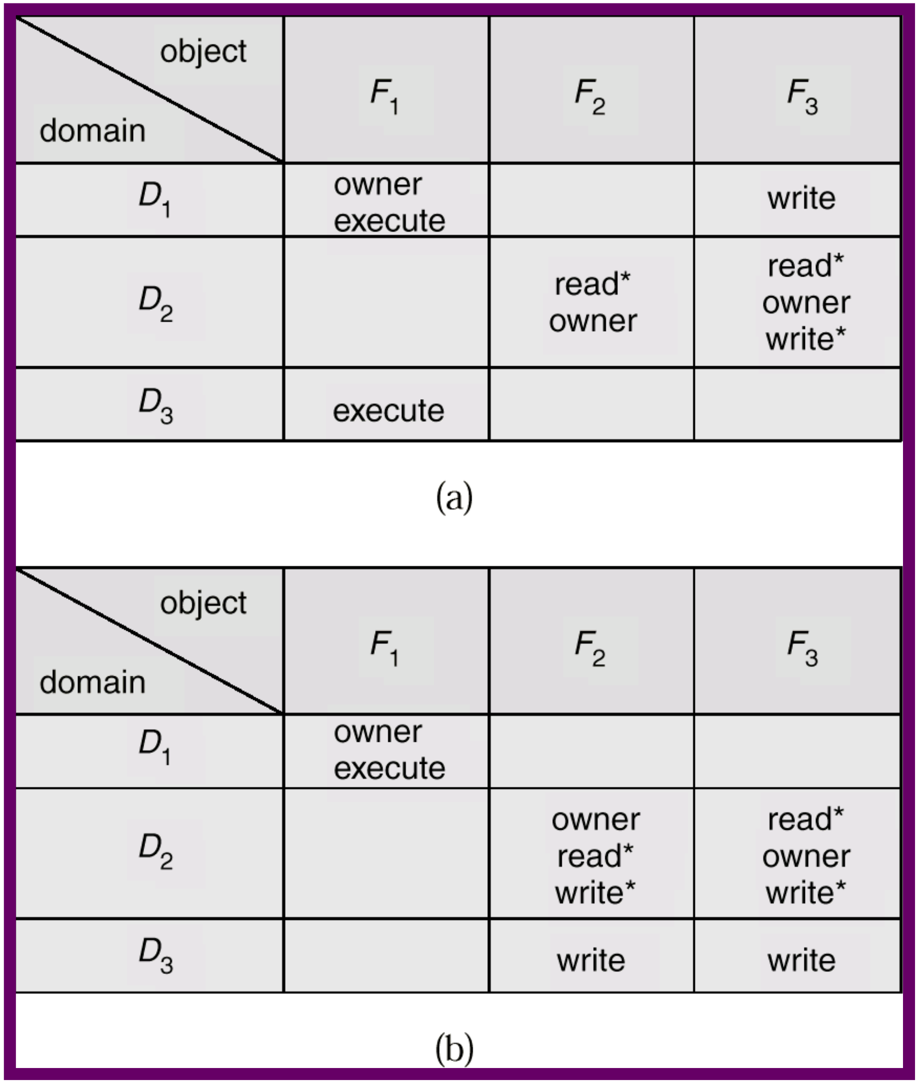
(a)

object domain	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix With *Owner* Rights

- Domain D1 is owner of F1 and Can add or delete any valid right in F1 column. Similarly D2 is owner of F2 and F3.



object domain	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write*
D_3	execute		

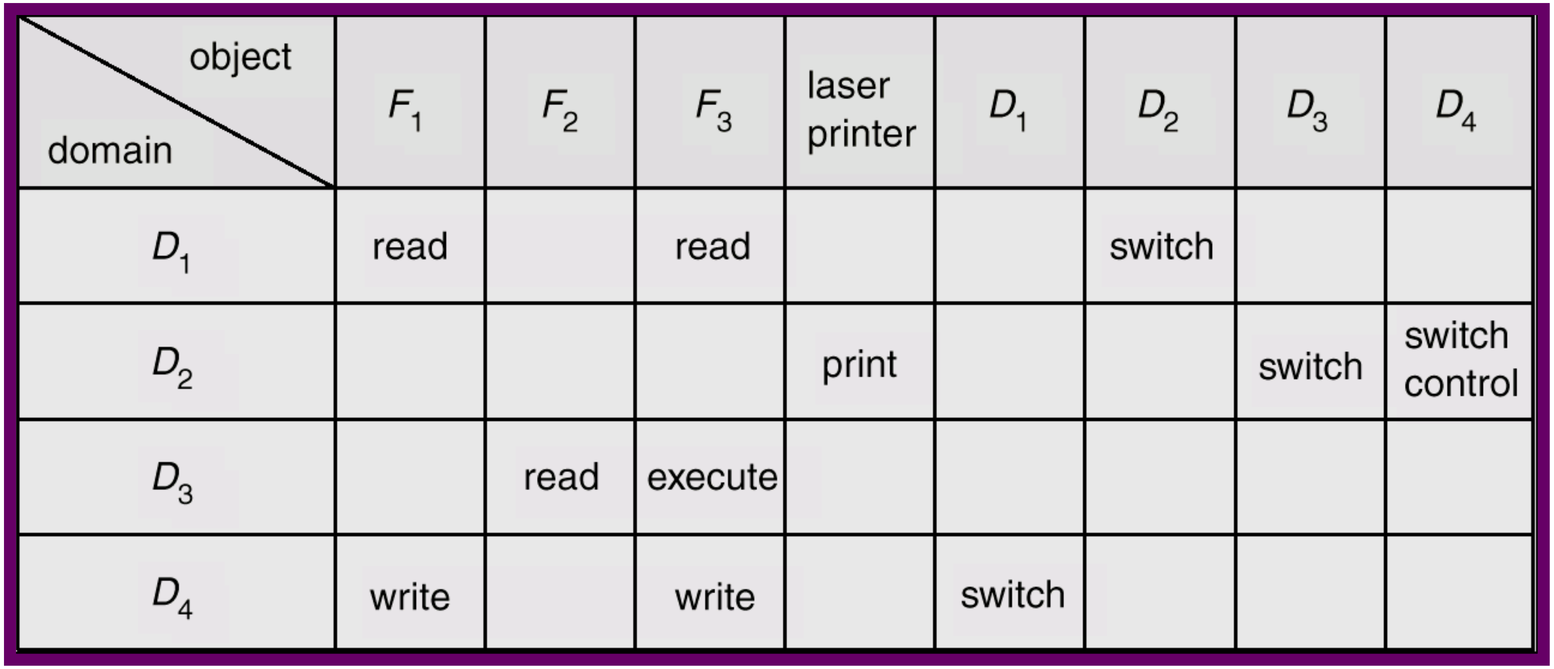
(a)

object domain	F_1	F_2	F_3
D_1	owner execute		
D_2		owner read* write*	read* owner write*
D_3		write	write

(b)

Access Matrix: Switch control

A process executing in D2 could modify domain D4.



object domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Confinement problem

- *The copy and owner rights provide us the mechanism to limit the propagation of access rights.*
- *However, they do not give appropriate tools for preventing the propagation of information.*
- The problem of guaranteeing that no information initially held in an object can migrate outside of its execution environment is called **the confinement problem**.
- **Confinement problem is unsolvable.**

Implementation of the Access Matrix

- *Global table of <domain, object, rights-set>*
  - *Table becomes large and additional I/O is needed*

- *Access list for every object*
  - *Each column can be implemented as a access list for the object. The llist for each object consists of <Domain, rights-set>*

- *Capability List* for domains
  - A capability list a domain is a list objects together with the operations allowed on those objects

Capability-Based Systems

- Hydra
  - Fixed set of access rights known to and interpreted by the system.
  - Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights.
- Cambridge CAP System
  - Data capability - provides standard read, write, execute of individual storage segments associated with object.
  - Software capability -interpretation left to the subsystem, through its protected procedures.

Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.

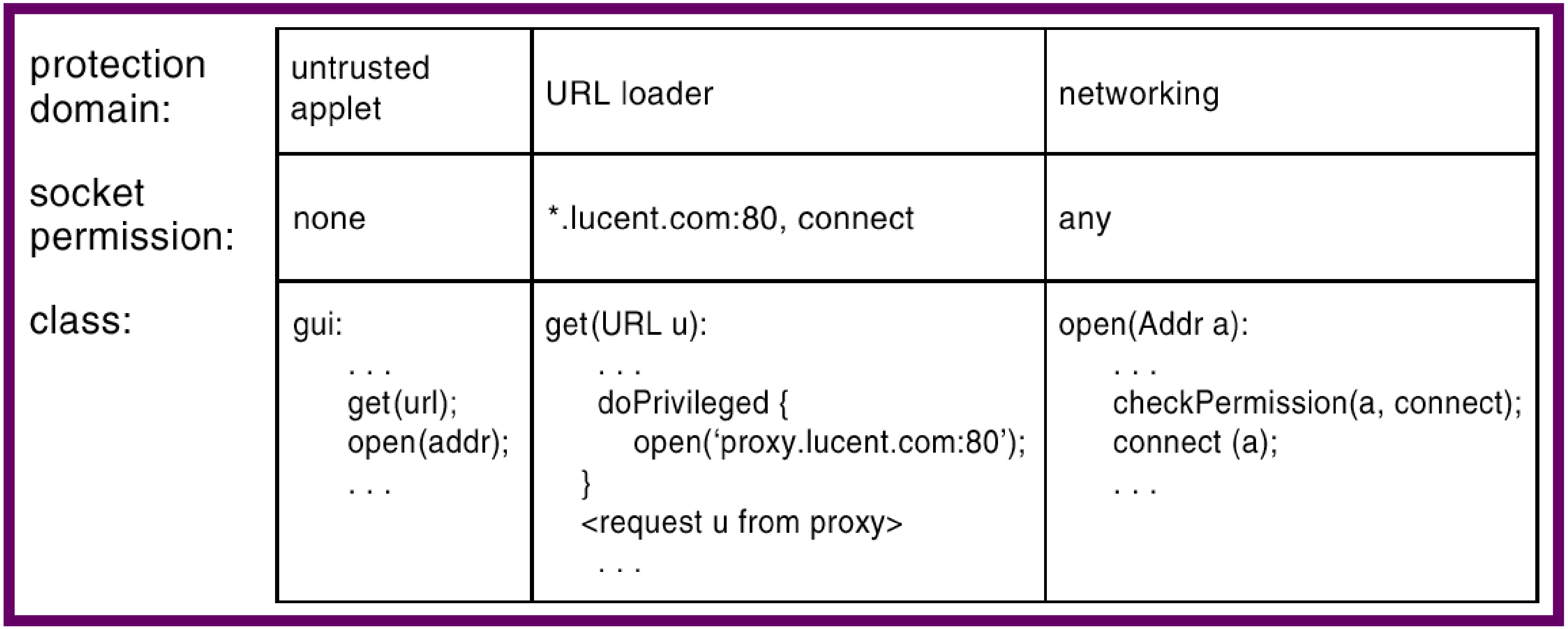
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.

- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A class is assigned a protection domain when it is loaded by the JVM.
- The protection domain indicates what operations the class can (and cannot) perform.
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library.

Stack Inspection



protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	. . .
	get(url);
	open(addr);
	. . .
		get(URL u):
		. . .
		doPrivileged {
		open(‘proxy.lucent.com:80’);
		}
		<request u from proxy>
		. . .
			open(Addr a):
			. . .
			checkPermission(a, connect);
			connect (a);
			. . .