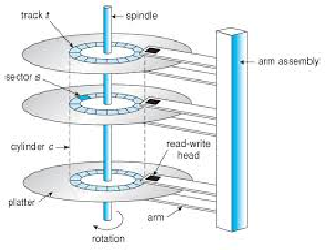
**DISK MANAGEMENT**

* Disk drives are addressed as large 1-dimensional arrays of *logical blocks*, where the logical block is the smallest unit of transfer.
* The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
* Sector 0 is the first sector of the first track on the outermost cylinder.
* Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.



**Disk Access Time**

• seek time: time to position heads on cylinder (a fixed head disk does not require seek time but is more

expensive than a moving-head disk)

• rotational latency: delay in accessing material once seek accomplished (time required to wait for data

to rotate around under head)

• Transmission time: time to transfer information once it is under the head. • access time = seek time +

rotational latency +read/write transmission time seek time >> read/write time

**Disk Scheduling**

* The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
* Access time has two major components
* *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
* *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
* Minimize seek time
* Seek time ≈ seek distance

Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

* Many requests may be pending at once. Which should be handled first?

•Head moving strategy developed

•Attempting to manage the overall disk seek time. Latency is not controllable and transfer time depends on the size of the transfer request

•Different strategies:

–FCFS

–SSTF

–SCAN

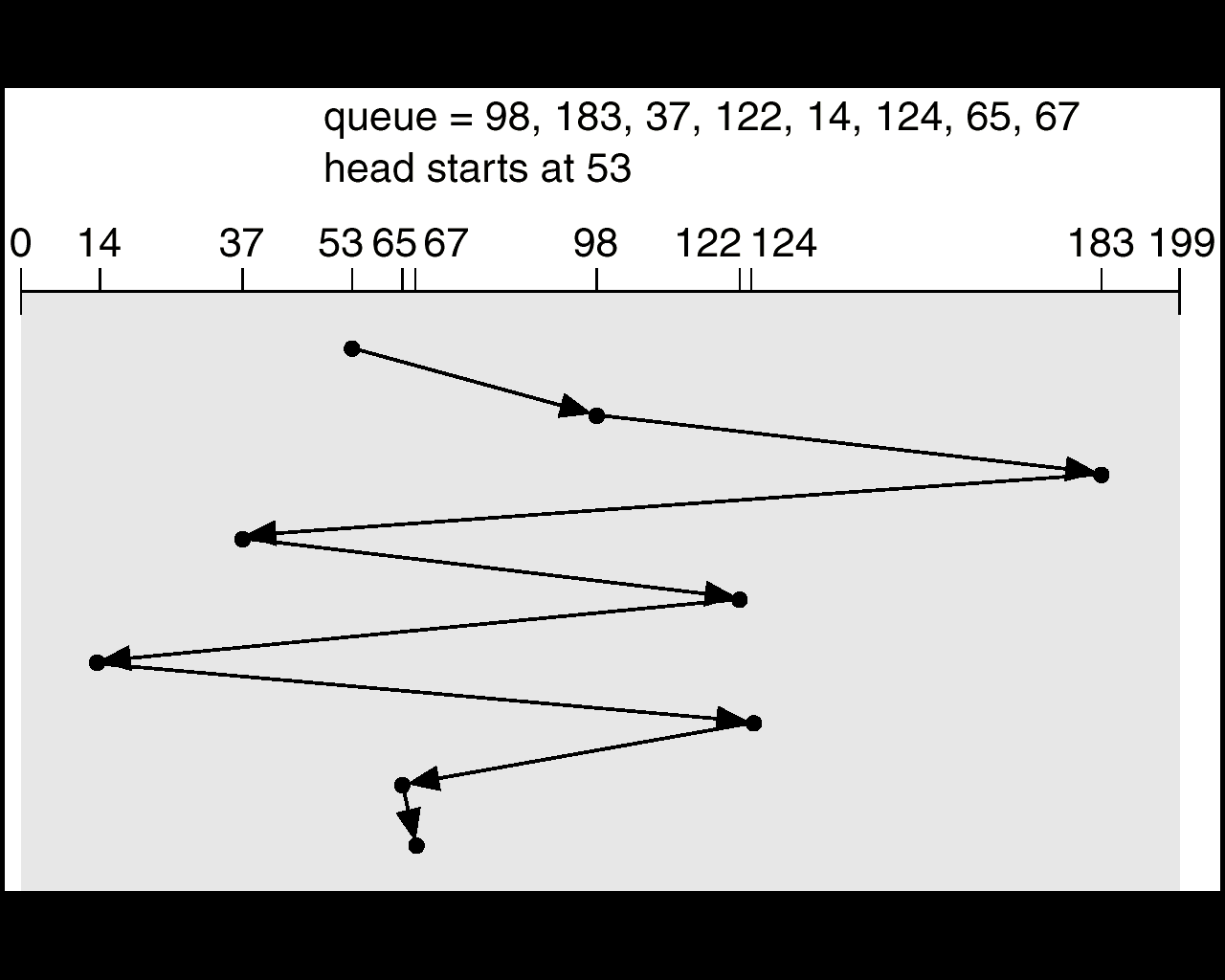
--C-SCAN

–LOOK

1. **FCFS Scheduling**

The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. Consider, for example, a disk queue with requests for I/O to blocks on cylinders in that order. If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124/65, and finally to 67, for a total head movement of 640 cylinders.

98, 183, 37,122, 14, 124, 65, 67

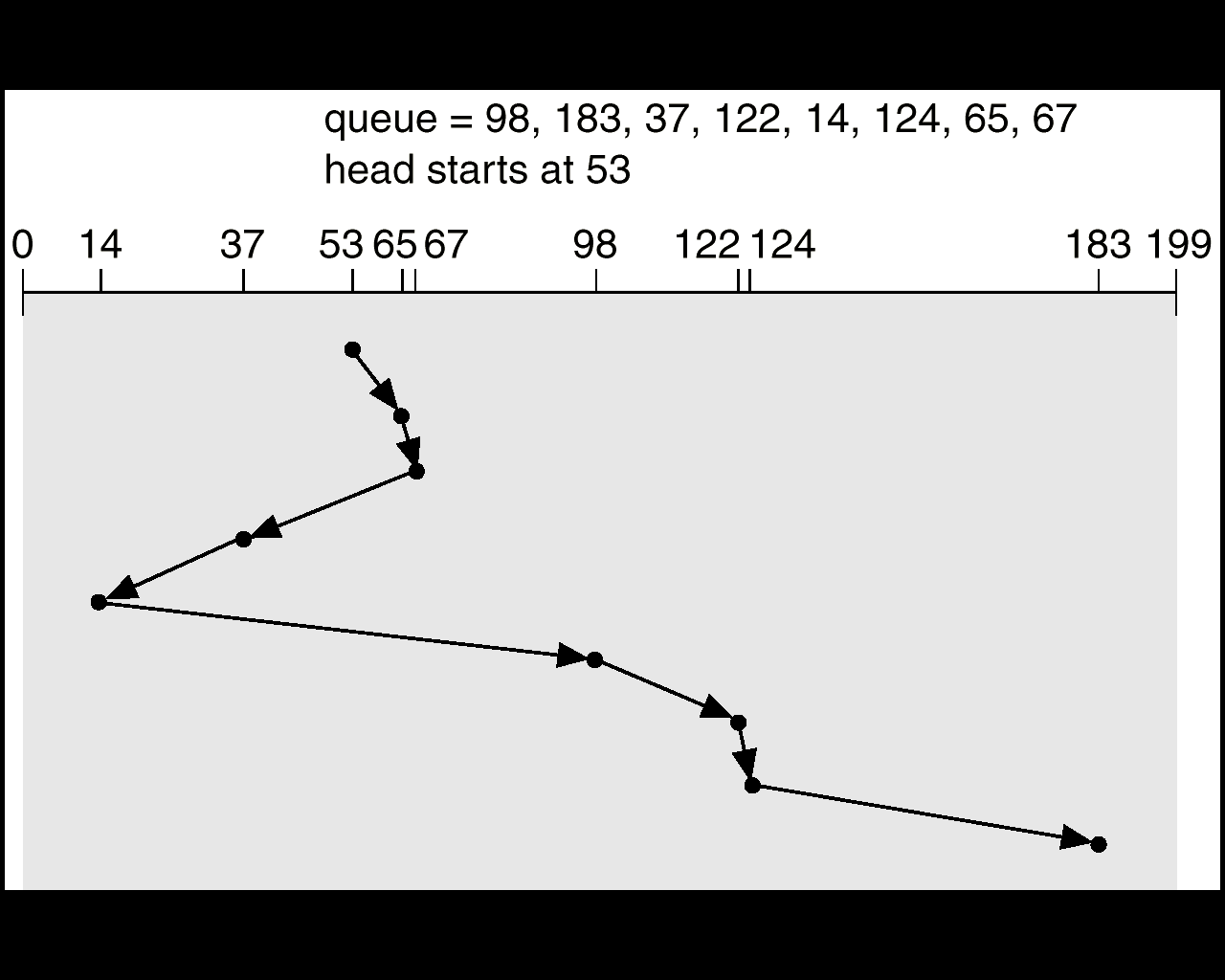


1. **SSTF Scheduling**

It seems reasonable to service all the requests close to the current head position before moving the head far away to service other requests. This assumption is the basis for the **shortest-seek-time-first (SSTF) algorithm.** The SSTF algorithm selects the request with the minimum seek time from the current head position.

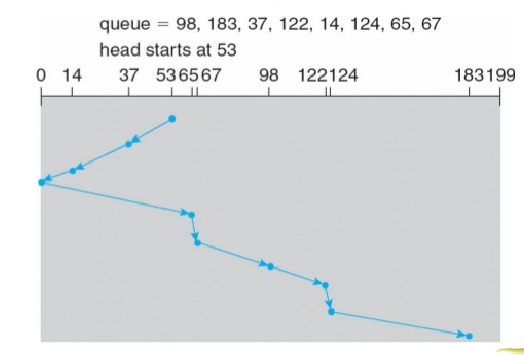
Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position. For our example request queue, the closest request to the initial head position (53) is at cylinder 65. Once we are at cylinder 65, the next closest request is at cylinder 67. From there, the request at cylinder 37 is closer than the one at 98, so 37 is served next. Continuing, we service the request at cylinder 14, then 98,122, 124, and finally 183. This scheduling method results in a total head movement of only 236 cylinders—little more than one-third of the distance needed for FCFS scheduling of this request queue. This algorithm gives a substantial improvement in performance.

SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling, it may cause **starvation** of some requests. Remember that requests may arrive at any time. Suppose that we have two requests in the queue, for cylinders 14 and 186, and while servicing the request from 14, a new request near 14 arrives. This new request will be serviced next, making the request at 186 wait. While this request is being serviced, another request close to 14 could arrive. In theory, a continual stream of requests near one another could arrive, causing the request for cylinder 186 to wait indefinitely.



1. **SCAN Scheduling**

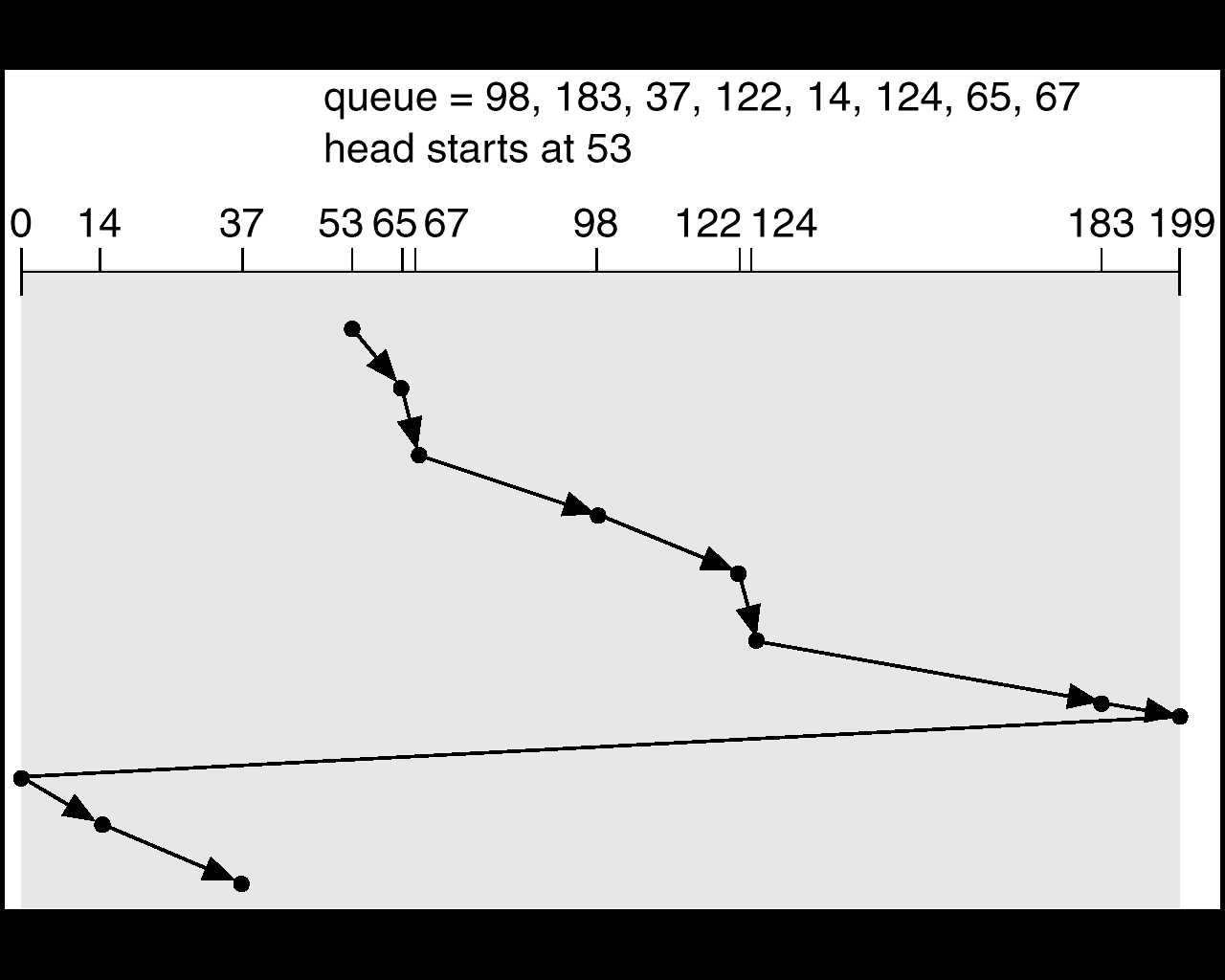
In the **SCAN algorithm,** the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the **elevator algorithm,** since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.



Let's return to our example to illustrate. Before applying SCAN to schedule the requests on cylinders 98,183, 37,122,14, 124, 65, and 67, we need to know the direction of head movement in addition to the head's current position (53). If the disk arm is moving toward 0, the head will service 37 and then 14. At cylinder 0, the arm will reverse and will move toward the other end of the disk, servicing the requests at 65, 67, 98, 122, 124, and 183 .If a request arrives in the queue just in front of the head, it will be serviced almost immediately; a request arriving just behind the head will have to wait until the arm moves to the end of the disk, reverses direction, and comes back.

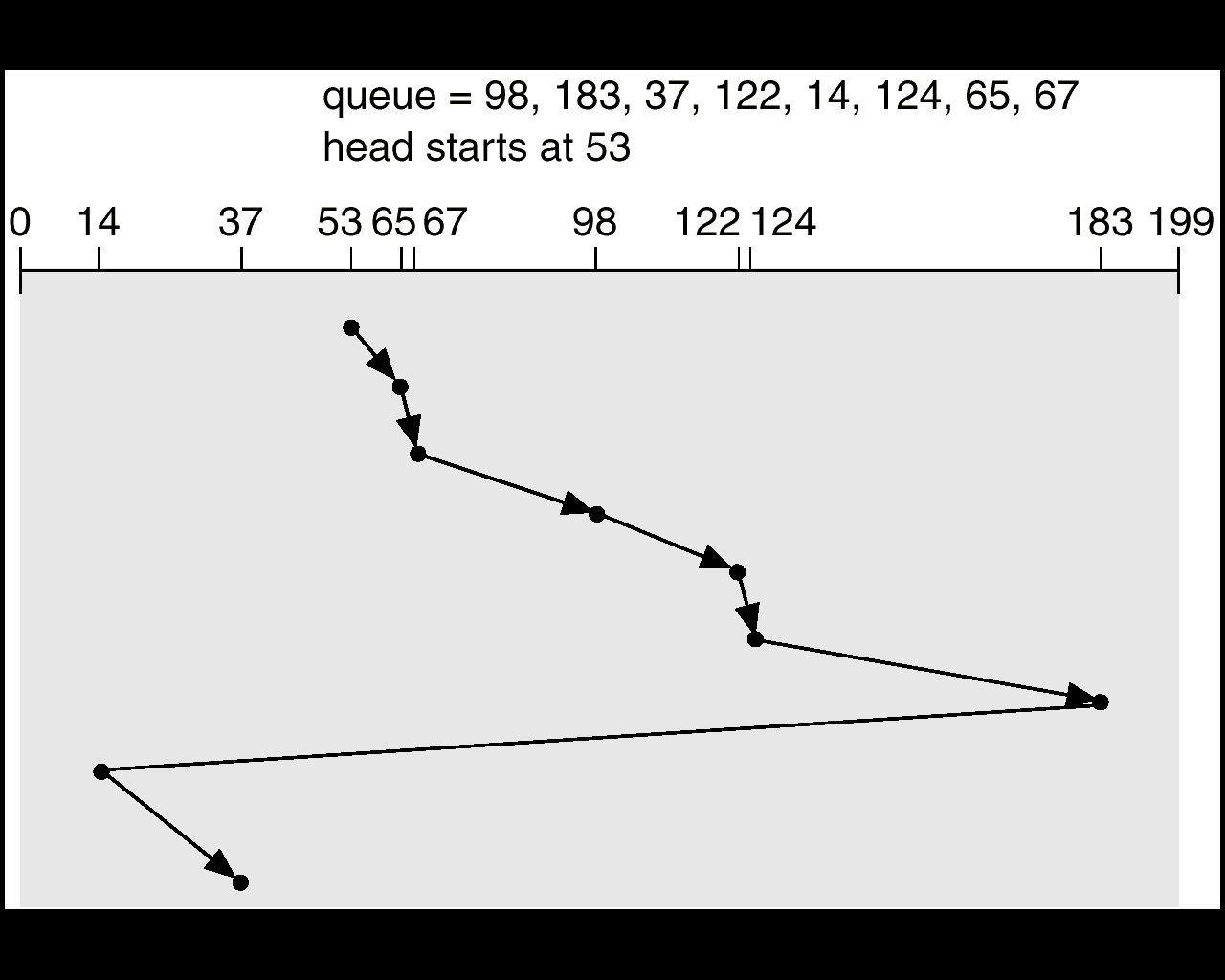
1. **C-SCAN Scheduling**

Circular SCAN (C-SCAN) **scheduling** is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip .The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.



1. **LOOK Scheduling**

As we described them, both SCAN and C-SCAK move the disk arm across the full width of the disk. In practice, neither algorithm is often implemented this way. More commonly, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk. Versions of SCAN and C-SCAN that follow this pattern are called **LOOK** and **C-LOOK scheduling,** because they *look* for a request before continuing to move in a given direction.



**Goals of Protection** 

* Operating system consists of a collection of objects, hardware or software .
*  Each object has a unique name and can be accessed through a welldefined set of operations. The operations that are possible may depend on the object (read , write, rewind, open,…etc) 
* Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so. 
* Protection: control access to a system by limiting the types of file access permitted to users. Ensure that only processes that have gained proper authorization from the operating system can operate on memory segments, the CPU, and other resources. 
* The O.S. provides protection mechanisms, which are described, so that an application designer can use them in designing her or his own protection software.

**Principles of Protection** 

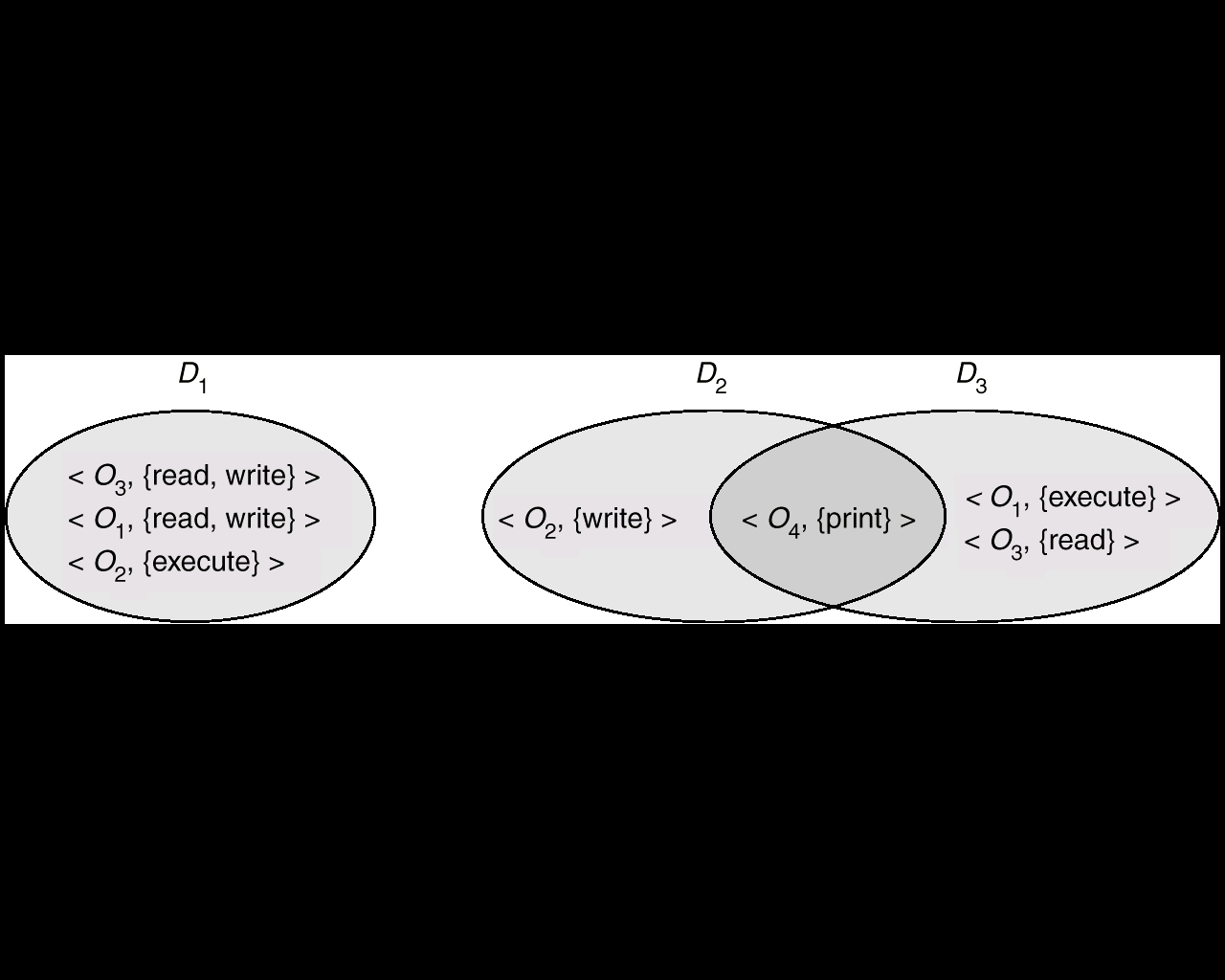
* The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use. 
* Mechanism vs Policy Mechanisms determine how something will be done; policies decide what will be done 
* Guiding principle – principle of least privilege
* Programs, users and systems should be given just enough privileges to perform their tasks.
* failure or compromise of an OS component does the minimum damage and allows the minimal damage to be done 
* need-to-know principle: a process should be able to access only those resources that it currently requires to complete its task.
* useful in limiting the amount of damage a faulty process can cause in the system.

Domain of Protection 

* A process operates within a protection domain, which specifies the resources that the process may access. 
* Each domain defines a set of objects and the types of operations that may be invoked on each object. 
* The ability to execute an operation on an object is an access right. 
* A domain is a collection of access rights, each of which is an ordered pair: 
* Example: If domain D has the access right: , then a process executing in domain D can only read and write file F.

**Domain Structure** 

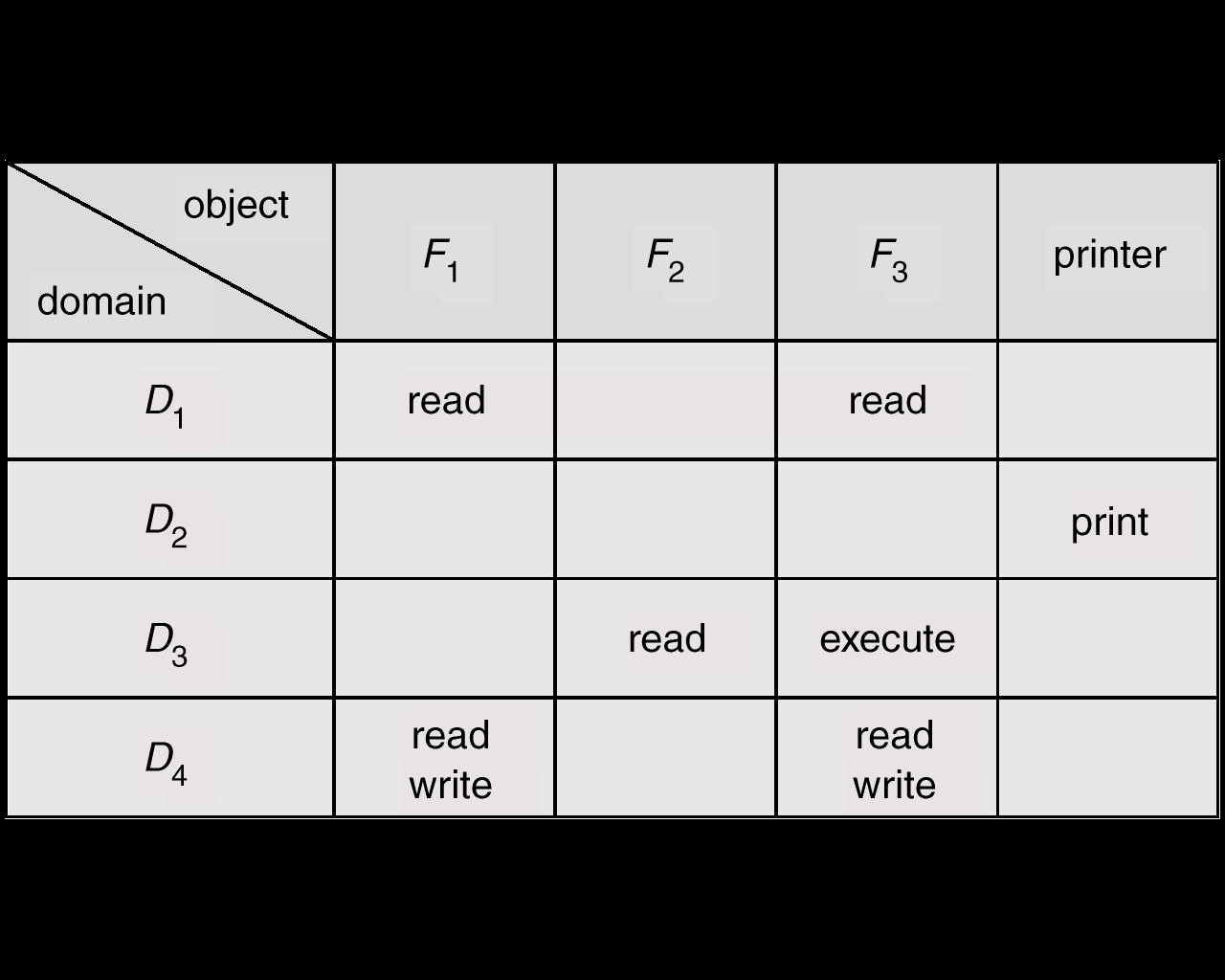
* Access-right = where rights-set is a subset of all valid operations that can be performed on the object. 
* Domain = a collection of access-rights 
* A protection domain specifies the resources that the process may access 



* Domains may share access rights.
* A process executing in either D2 or D3 can print O4
* A process must be executing in D1 to read and write O1. Also, only process in D3 may execute O1

**Access Matrix** 

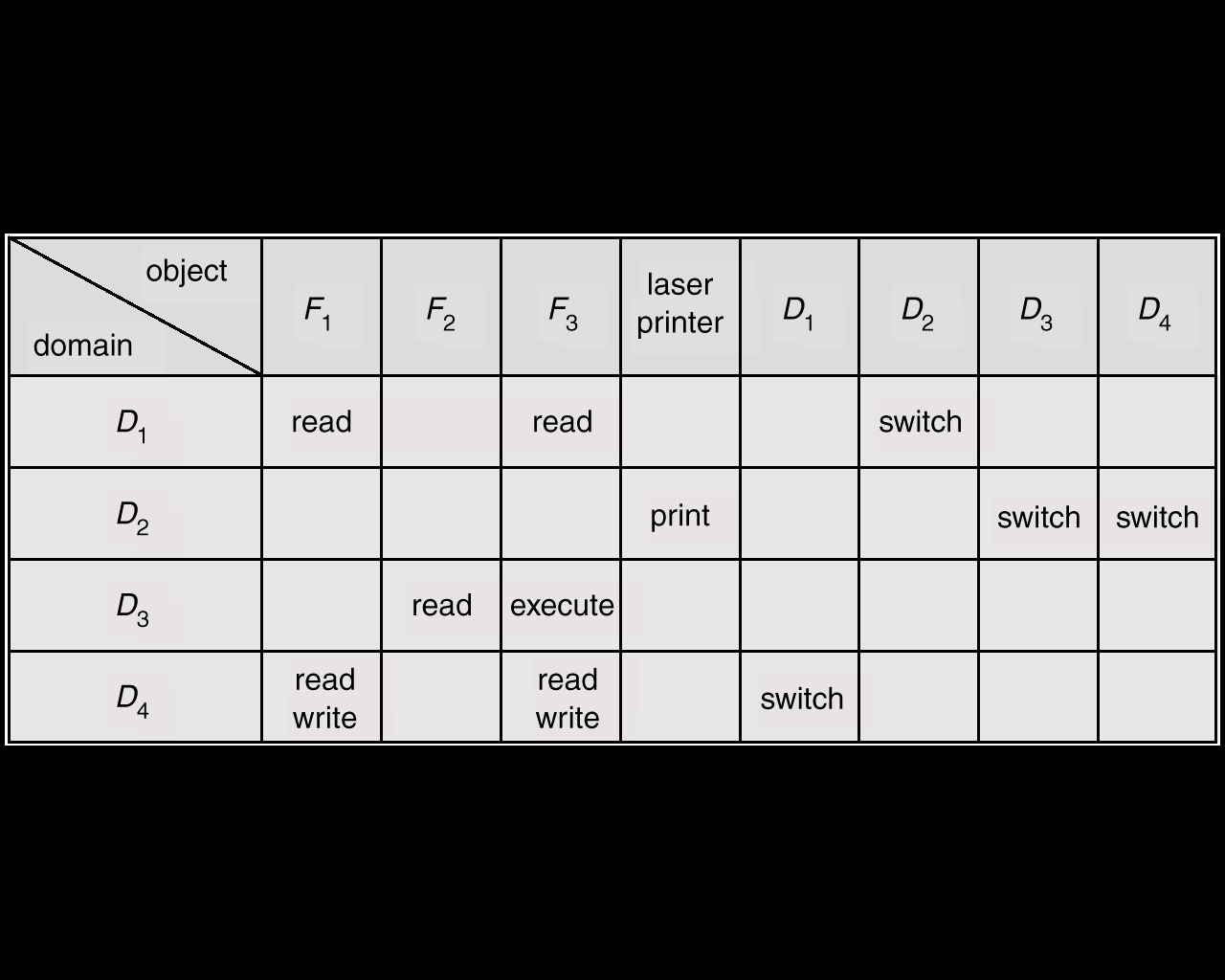
* View protection as a matrix (access matrix)
* Rows represent domains z Columns represent objects
* Each entry in the matrix consists of a set of access rights.
* Access(i, j) is the set of operations that a process executing in Domaini can invoke on Objectj  access list keeps track of which object belongs to which domain



Access Matrix illustration: 

* 4 domains and 4 objects (3 files and one printer). 
* When a process executes in D1, it can read files F1 and F3. 
* A process executing in D4 has the same privileges as it does in D1, it can also write onto files F1 and F3. 
* Printer can be accessed by a process executing in D2
* Users decide the contents of the access-matrix entries.
* When a user creates a new object Oj, the column Oj is added to the access matrix with the appropriate initialization entries.
* The user may decide to enter some rights in some entries in column j and other rights in other entries. 
* Access matrix provides mechanism for defining and implementing strict control for both static and dynamic association between processes and domains.
* Controls changing content of access-matrix entries.
* Controls switching between domains.
* When we switch a process from one domain to another, we are executing an operation (switch) on an object (the domain).
  + We can control **domain switching** by including domains among objects of the access matrix. 
* When we change the content of the access matrix, we are performing an operation on an object which is the access matrix.
* We can control these changes by including the access matrix itself as an object.
* Processes should be able to switch from one domain to another.
* Domain Switching from Di to Dj is allowed to occur iff access right switch є access(i,j).

**Domain Switching: Access Matrix with domains as objects**



• A process executing in D4 (row) can switch to D1. A process executing in D2 (row) can switch to D3 or D4. A process executing in D1 (row) can switch to D2.

**Access Matrix with Copy Rights**

* A process executing in D2 can copy the read operation into any entry associated with F2. 
* The copy scheme has 3 variants:

1. A right is copied from access(i,j) to access(k,j) is not limited: This action is called copy.

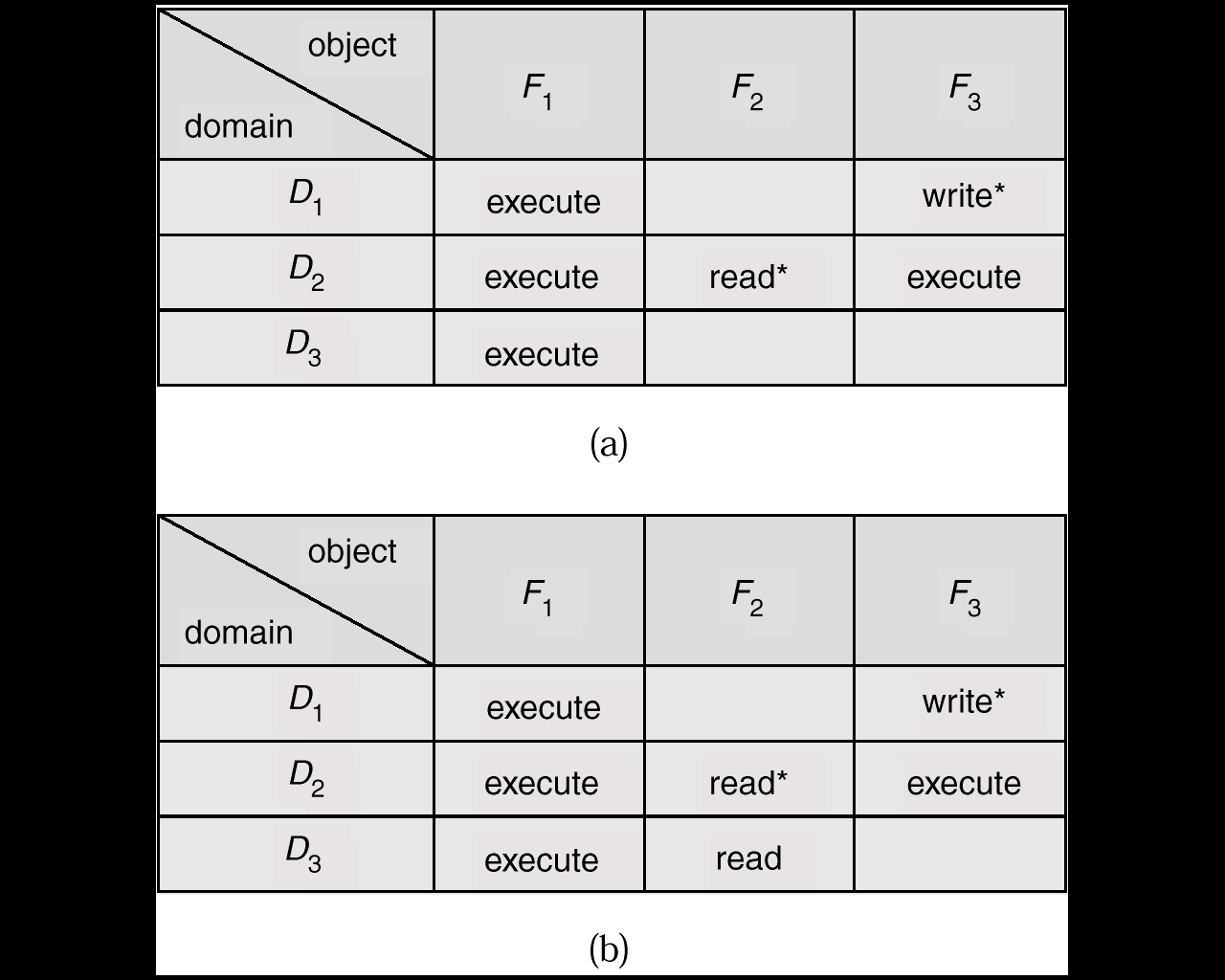
* When the right Read\* is copied from access(i,j) to access(k,j), the Read\* is created.
* So, a process executing in Dk can further copy the right Read\*

1. Propagation of the copy right may be limited: This action is called limited copy.

* When the right Read\* is copied from access(i,j) to access(k,j), only the Read (not Read\*) is created.
* So, a process executing in Dk cannot further copy the right Read.

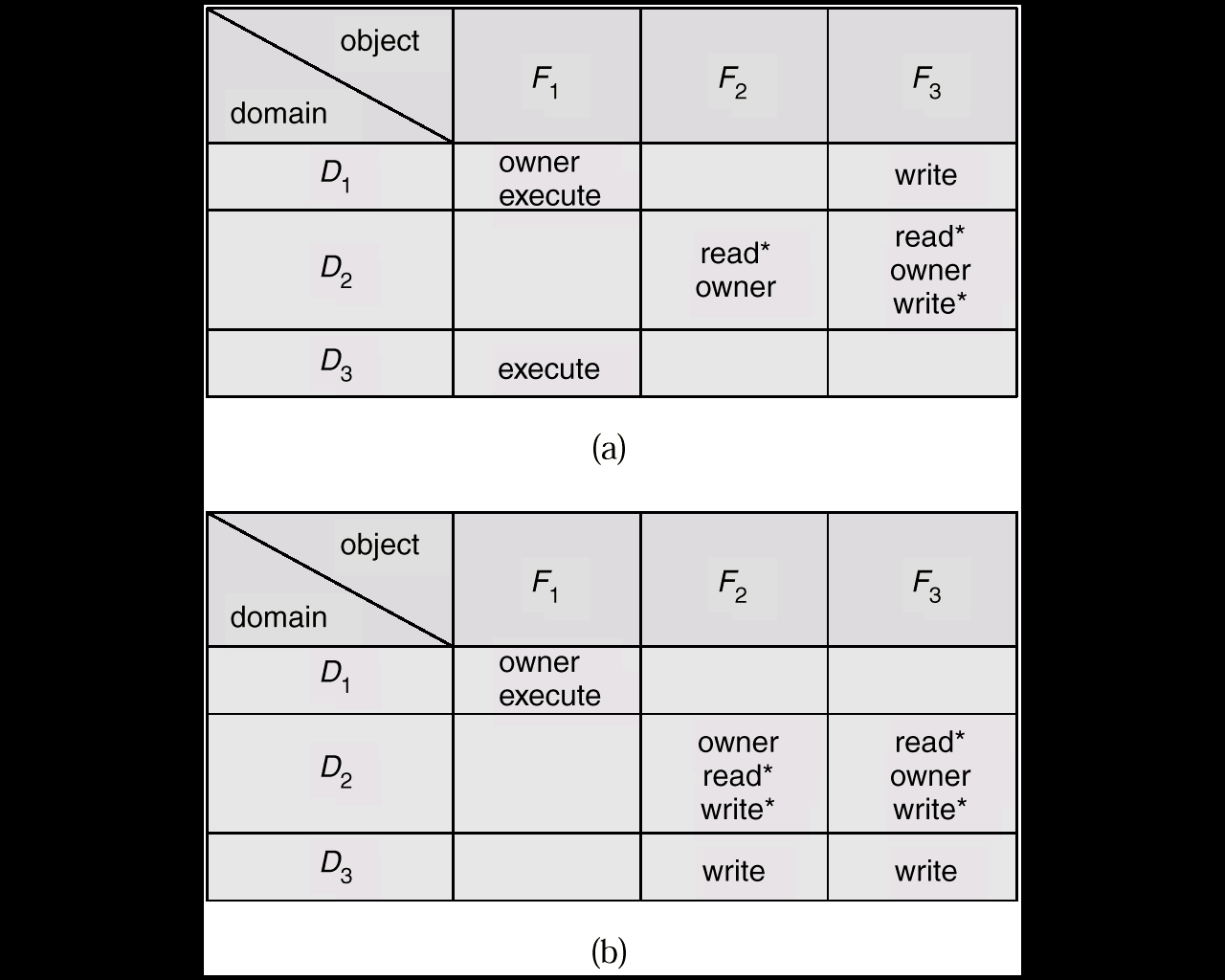
1. A right is copied from access(i,j) to access(k,j); it is then removed from access(i,j).

* This action is called a transfer of a right, rather than a copy

****

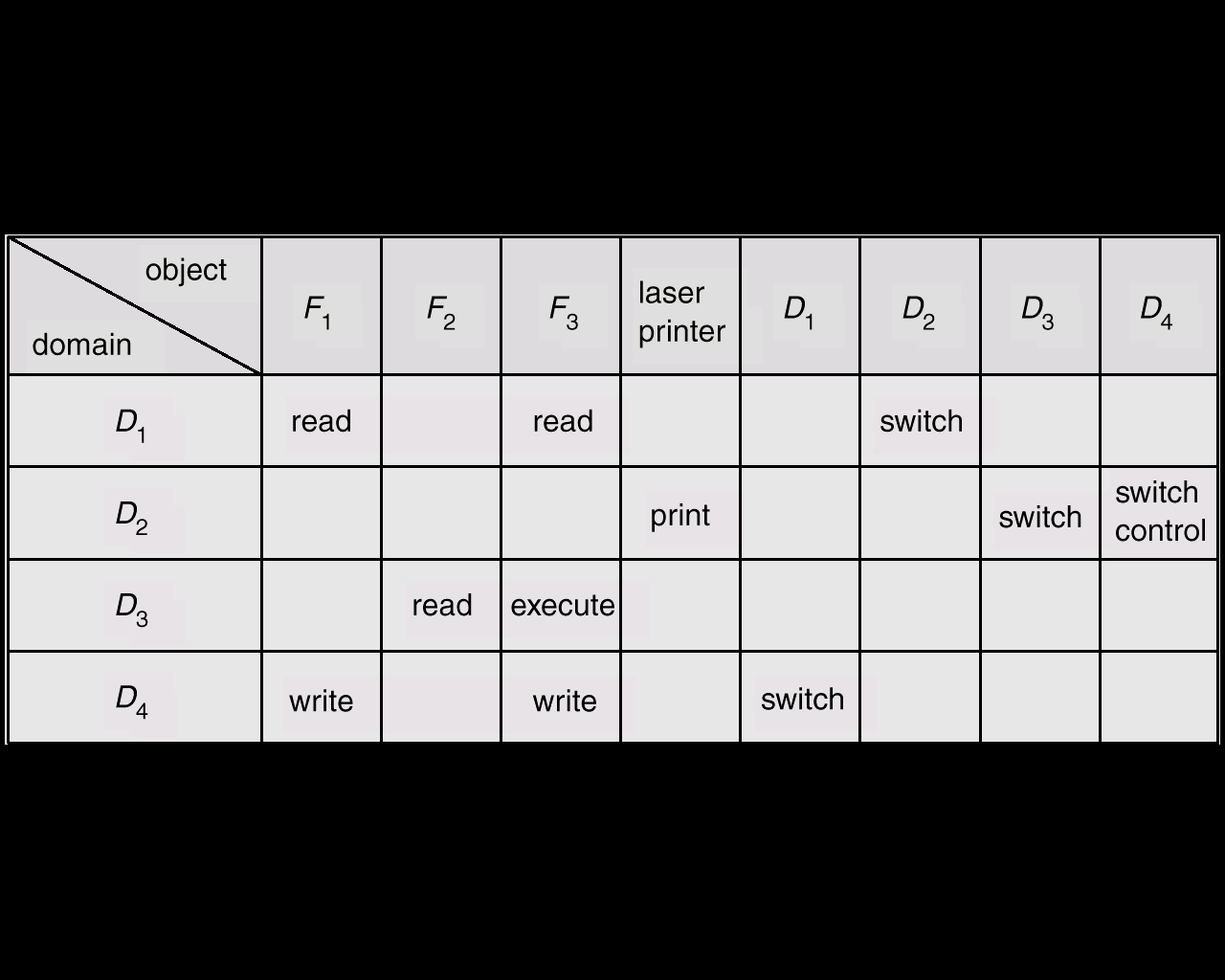
**Access Matrix With Owner Rights**

* We need a mechanism to allow addition of new rights and removal of some rights.
* The owner right controls these operations.
* If access(i,j) includes the owner right, then a process execution in Di can add and remove any right in any entry in column j. 
* D1 is the owner of F1, and can add and delete any valid right in column F1. 
* D2 is the owner of F2 and F3, and can add and delete any valid right within these 2 columns.

****

**Access Matrix With Control Rights **

* The copy and owner rights allow a process to change the entries in a column.
* limit the propagation of access rights
* However, they do not give us the appropriate tools for preventing the propagation (or disclosure) of information. 
* So, a mechanism is needed to change the entries in a row.
  + The control right is applicable only to domain objects (rows).
  + If access(i,j) includes the control right, then a process executing in Di can remove any access right from row j.

****