



# Operating Systems ICS 431

## Week 14

### Ch.: Protection

**Dr. Tarek Helmy El-Basuny**

## Protection and Security Definitions

- The **Protection** and **Security** concepts are often used together, and the distinction between them is a little bit ambiguous.
- Let me hear from you about the meaning of Protection and Security in computer system.
- **Protection** refers to a mechanism for controlling the access of running processes, or users to the resources of the computer system. i.e.
  - Processes or users in an operating system must be protected from one another's activities.
- **Security** is the process of:
  - Detecting and preventing unauthorized usage of your system's resources by outsider users or processes. i.e.
    - Protect the system from viruses, worms, malware or remote hacker intrusions.

# Protection and Security Definitions

## ➤ Types of misuse are :

- Intentional/planned misuse (**performed with purpose and intent**). i.e.
  - Trying to violate University IT Policies.
  - Trying to download of copyrighted materials.
- Accidental/unplanned/**unintended** misuse (**by chance**). i.e.
  - You forgot to logout your account and someone gets access to it.
- **Protection** is to prevent either accidental or intentional misuse.
  - Guarding users' data and processes **against internal threats** by other users or processes of the same system (Internal to OS).
- **Security** is to prevent intentional misuse.
  - Guarding users' data and processes **against external threats**, by users or processes outside the system (external to OS).
  - Ch. 15 in the text book discusses more about **Security**.

## A Policy & A Mechanism Definition

- A **Policy**: Decides **which/what/who** can access an object and in which mode?
  - For instance, FCFS, SJF are kind of policies to know which process can access the CPU?
  - **Policies** are ways to choose which activities to perform.
- A **Mechanism**: Determines, **how** to do something?
  - Is to know how? For instance, how processes can be granted resources?
  - **Mechanisms** are the implementations that enforce **policies**.
- **Protection: Mechanisms and Policies for:**
  - Preventing processes and users from accessing objects they are not allowed to access.
  - Issues internal to OS.
- **Security: Mechanisms and Policies for:**
  - Authentication of users,
  - Validation of messages,
  - Malicious intrusion detection, etc.
  - Issues external to OS.

## Security & Protection in OS

- Protection and Security allow the OS to do:
  1. **Authentication**: who is the user?
  2. **Authorization**: who is allowed to do what?
  3. **Enforcement**: make sure that users/processes do only what they are authorized to do.

Term	Explanation
Authentication	Verifying the identity of a user. Operating systems most often perform authentication <i>by knowledge</i> . That is, a person claiming to be some user X is called upon to exhibit some knowledge shared only between the OS and user X, such as a password.
Authorization	Authorization has two aspects: (1) Granting a set of access privileges to a user, for example, some users may be granted read and write privileges to a file, while others are granted read-only privileges, (2) Verifying a user's right to access a resource in a specific manner.

## Goals/objectives of Protection and Security in OS

Goal	Description
Secrecy	<u>Only authorized users should be able to access information.</u> This goal is also called confidentiality.
Privacy	<u>Information should be used only for the purposes for which it is intended and shared.</u>
Authenticity	<u>It should be possible to verify the source or sender of information,</u> and also verify that the information is preserved in the form in which it was created or sent.
Integrity	<u>It should not be possible to destroy or corrupt information.</u>



## Outline of Protection Chapter

- Goals and Guiding Principles of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Language-Based Protection

## Protection Goal and Principle

- A computer system is a collection of processes and objects,
- Objects means hardware (CPU, memory segments, printers, disks, ...) or software (files, data structure, messages, etc.).
- Each object has a unique name and can be accessed through a well-defined set of operations.
- **Goal of Protection:**
  - To ensure that each object is accessed correctly and only by those processes/users that are allowed to do so.
- **Guiding Principles of Protection:**
  - **Principle of least privilege:** Processes, users and systems should be given just enough privileges/rights to perform their tasks.
  - **Separate policy from mechanism:**
    - Policy: the code built in the OS that says who can do what.
    - Mechanism: the code built in the OS to know how to do something.

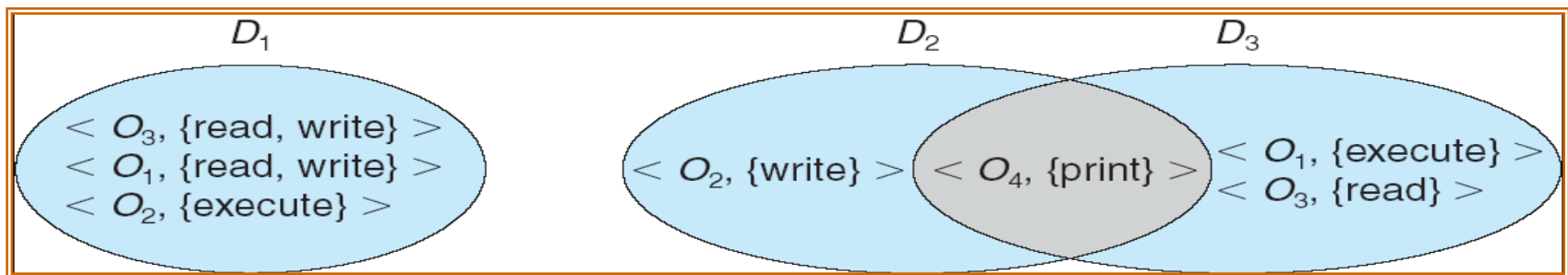


## Domain of Protection

- A Domain = a set of objects and a set of access-rights that a process within that domain can do on these objects.
- Access-rights: is a set of all valid operations that can be performed on the object.
- The operations that are possible depend on the object,
  - i.e., data files can be created, opened, read, written, closed, deleted.
- At any time, a process should be able to access only those objects that it currently required to complete its task.
  - Example: When process P invokes procedure A, A should be allowed to access only its own local variables, along with the parameters explicitly passed to it.
  - A should NOT be able to access other variables of P.
- Example: A compiler should be able to access only well defined subset of files (source, linked library files) related to the file to be compiled.

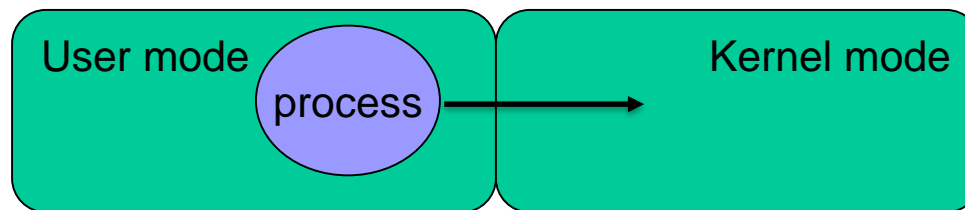
# Domain Structure

- A domain can be realized in variety of ways:
- **Each user can be a domain:** in this case, the set of objects that can be accessed depends on the identity of the user.
  - Domain switching corresponds to logout and login of users.
- **Each process may be a domain:** in this case, the set of objects that can be accessed depends on the identity of the process.
  - Domain switching corresponds to sending messages and waiting for responses.
- **Each procedure/method may be a domain:** in this case, the set of objects that can be accessed corresponds to the local variables defined within the procedure.
  - Domain switching occurs when a procedure call is made.
- Domains may share access rights:
  - For example in the following domains, **D1**, **D2**, and **D3**.
  - The print access right of **<O4, {print}>** is shared by **D2** and **D3**.
- This means a process executing in either of these two domains can print **O4**.



## Association between a Process & a Domain

- The association **between a process and a domain** may be either static or dynamic.
- If the association is static: it means the set of objects available to a process is fixed through out the life time of a process.
- If the association is dynamic: it means the OS allows the process to switch from one domain to another.
- **For example**, consider the dual-mode of the operating system execution.
  - When a process executes in kernel mode, it can execute **privileged** instruction and thus gain complete control of the computer system.
  - On the other hand, if the process executes in a user's mode, it can invoke only **non-privileged** instruction.

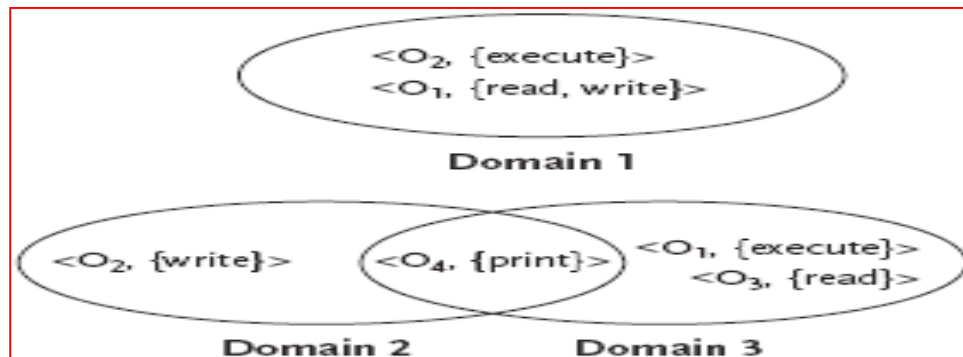


## Access Matrix: To implement Protection Domains

- Basic elements of the system are:
  - **A process**: An entity capable of accessing objects.
  - **An object**: Any resource to which access is controlled.
    - i.e. Memory, Files, CPU, Disks, Segments of Memory, ....
  - **An access right**: The way in which an object is accessed by the process.
    - **Examples**: read, write, and execute a file.
- **Protection Domain** = a set of objects and a set of access-rights.
- **Access Matrix**: represents the protection domains and access rights:
  - **Rows** represent the domains (**users, processes, procedures, etc.**).
  - **Columns** correspond to the objects/**resources**.
  - **Matrix entries** specify the access rights to an object in the corresponding domain.

## Protection Domains: Example 1

- Creating a protection domain for a university to control the access-rights for students, faculty and administration to the objects.
- **Assume** we have three domains ( $D_1$ ,  $D_2$ , and  $D_3$ ), and four objects ( $O_1$ ,  $O_2$ ,  $O_3$ , and  $O_4$ ) such that:
  - All students are going to log into to  $D_1$
  - All faculty members are going to log into to  $D_2$
  - All system administrators are going to log into  $D_3$
- **With the following access-rights to the objects  $O_1$ ,  $O_2$ ,  $O_3$ , and  $O_4$ .**
  - Students' processes can execute  $O_2$  and read & write  $O_1$
  - Faculty' processes can write  $O_2$  and print  $O_4$
  - System administrators' processes can execute  $O_1$ , read  $O_3$ , and print  $O_4$
- That can be represented as following where  $O_4$  is shared between  $D_2$  and  $D_3$ .



## Access Matrix: Example 1

- The access matrix of the previous example that specifies the **access control policy** for the system can be represented as:
  - 3 rows for the domains and 4 columns for the objects as following.

	Object 1	Object 2	Object 3	Object 4
Domain 1	{read, write}	{execute}		
Domain 2		{write}		{print}
Domain 3	{execute}		{read}	{print}

- This access matrix allows:**
  - A student in ( $D_1$ ) to read & write Object<sub>1</sub> and execute Object<sub>2</sub>.
  - A faculty member in ( $D_2$ ) to write Object<sub>2</sub> and to print Object<sub>4</sub>.
  - An admin in ( $D_3$ ) to execute Object<sub>1</sub>, read Object<sub>3</sub> and print Object<sub>4</sub>.

## Access Matrix: Example 2

- Column represents the objects ( $F_1$ ,  $F_2$ ,  $F_3$  and printer).
- Row represents a domain ( $D_1$ ,  $D_2$ ,  $D_3$ , and  $D_4$ ).
- Access** ( $i, j$ ) is the set of operations that a process executing in Domain  $i$  can do on Object  $j$ .
- Process/users working in:
  - Domain<sub>1</sub> can Read  $F_1$  and  $F_3$
  - Domain<sub>2</sub> can use printer
  - Domain<sub>3</sub> can Read  $F_2$  and execute  $F_3$

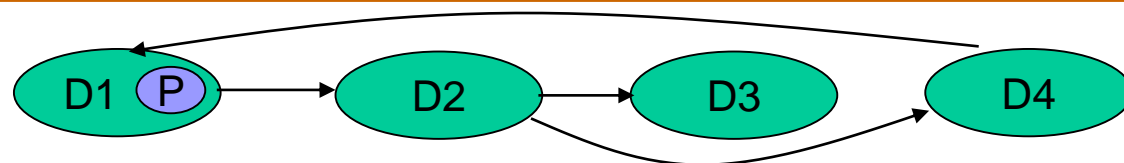
		Objects			
		$F_1$	$F_2$	$F_3$	printer
Domains	object \ domain				
	$D_1$	read		read	
	$D_2$				print
	$D_3$		read	execute	
	$D_4$	read write		read write	

- Can a process executes in Domain <sub>$i$</sub>  switch into Domain <sub>$j$</sub> ?

## Access Matrix **with** Domains as Objects

- Adding domains as objects to the access matrix so that a process can switch from one domain to another:
  - Use operation “**Switch**” on an object “Domain”
  - Process in domain  $D_4$  can switch to domain  $D_1$
  - Process in domain  $D_1$  can switch to domain  $D_2$
  - A process in  $D_1$  has read access to  $F_1$  and  $F_3$ , and
    - Can switch to  $D_2$  in order to print on the laser printer,
    - Then switch to  $D_3$  if it wants to obtain read access to  $F_2$  and execute access to  $F_3$ ,
    - Then switch to  $D_4$  to obtain write access to  $F_1$  and  $F_3$ .

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			





## Access Matrix with **Copy**, **Owner** and **Control** Rights

- Access matrix can be expanded to support dynamic protection.
  - By adding or removing any access rights.
- **Special access rights:**
  - Copy operation from  $O_i$  to  $O_j$  (access allows an object to copy, transfer rights from any domain)
  - Owner of  $O_i$  (access allows an object to copy, transfer or delete rights from any domain).
  - Control allows a process in this domain to remove any right from that domain,  $D_i$  can modify  $D_j$  access rights.

## Access Matrix with **Copy** Rights

- With **copy** rights (denoted with an \*), a domain can copy an access right to another domain.
- Domain  $D_2$  can copy read access of  $F_2$  into  $D_3$ .
- Domain  $D_1$  can copy write access of  $F_3$  to  $D_3$ .

- Process running in  $D_2$ 
  - Has a copy right of  $F_2$
  - Can copy a **read** operation to  $D_3$

- Process running in  $D_1$ 
  - Has a copy right of  $F_3$
  - Can copy a **write** operation to  $D_3$

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	write

(b)

# Access Matrix with **Owner** Rights

- **Owner** → a process executing in a domain with **Owner** right access **can add or remove any rights in the column  $j$** .
- $D_2$  can give itself write\* access to  $F_2$  and then give  $D_3$  write access.
- Can give  $D_3$  write access to  $F_3$  (**even though  $D_3$  does not have write privileges**).

- Process running in  $D_2$ 
  - Owns  $F_2$
  - Can add and remove any access right on  $F_2$  in  $D_1$  and  $D_3$ .

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		write
$D_2$		owner read* write*	read* owner write
$D_3$		write	write

(b)

## Access Matrix with **Control** Rights

- **Control** → applicable only to domain objects
  - Can remove any rights from the row  $i$

- A process executing in domain  $D_2$  can switch to  $D_4$  and also has a right to modify all access rights in  $D_4$ .

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					
$D_4$	<del>read</del> write		<del>read</del> write		switch			

System designers and users are responsible for defining what to be included in the Access matrix. **But how to implement the Access Matrix?**

## Implementing the Access Control Matrix

- **Access Control Matrix** is the security model of protection in the computer system. It is used to define the rights of each process executing in the domain with respect to each object.
- The Access Control Matrix can be implemented in different ways:
  1. The simplest implementation of the access matrix is to use a **Global Table**.
    - A global table is used to store ordered triples  $\langle \text{Domain}, \text{Object}, \text{Rights-set} \rangle$
    - Whenever an operation  $M$  is executed on an object  $O_j$  within domain  $D_i$ ,
    - The global table is searched for a triple  $\langle D_i, O_j, R_k \rangle$ , with  $M \in R_k$  ( $R_k$  are the authorized access rights).
    - If this triple is found, the operation is allowed to continue; otherwise an error is raised.
    - **Disadvantages:**
      - Since objects or domains are too many and that means the table could be too lengthy and can not be kept in the main memory.
      - More I/O access is required to get the table if the OS uses virtual memory to hold the table.

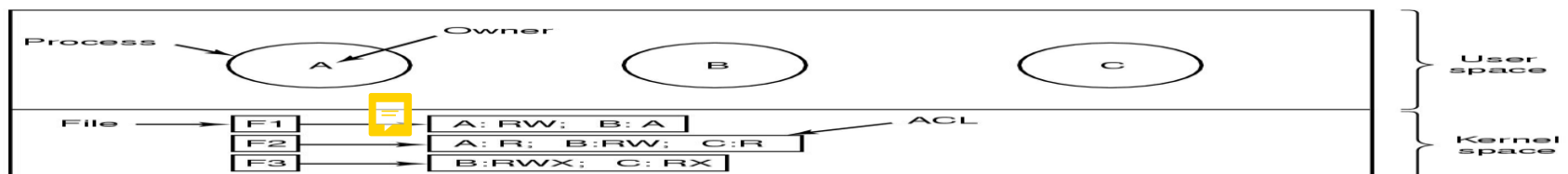
# Implementing the Access Matrix

## 2. Access Control Lists:

- Placing on each object a list of rights associated to that object.
- For example, if we have  $F_1$ ,  $F_2$  and  $F_3$ , and users  $A$ ,  $B$  and  $C$ , an access control list might look like:

Users\Objects	F1	F2	F3
A	RW-	R--	RW-
B	RW-	RW-	RWX
C	---	R--	R-X

- The rights are R (Read), W (Write) and X (Execute).
- A dash indicates the user does not have that particular right.
- i.e. user  $A$  does not have permission to execute  $F_3$ , and  $C$  has no rights at all on  $F_1$ .
- A control list of each object contains an ordered pairs [Domain, rights-set]
- Whenever an operation  $M$  is executed on an object  $O_j$  within domain  $D_i$ , the list is searched for  $O_j$  looking for an entry  $[D_i, R_k]$ , with  $M \in R_k$ .
- If the entry is found, the operation is allowed; otherwise the access is denied.

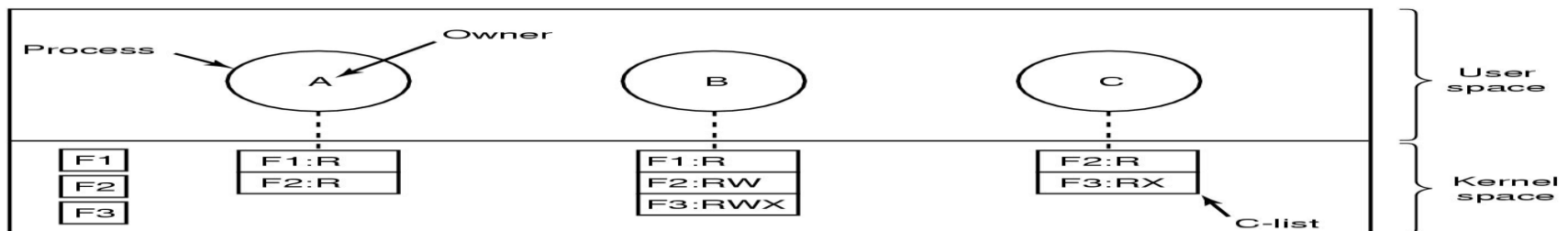


# Implementing the Access Matrix

## 3. Capability List

- Storing on **each Domain** a list of rights the user has for every object.
- To execute an operation **M** on an object **O<sub>j</sub>**, the **capability list** is searched.
- **Having of the capability means access is allowed.**
- The **capability list** must be protected to avoid user modification.
- Only the OS should be able to maintain the **capability list**.

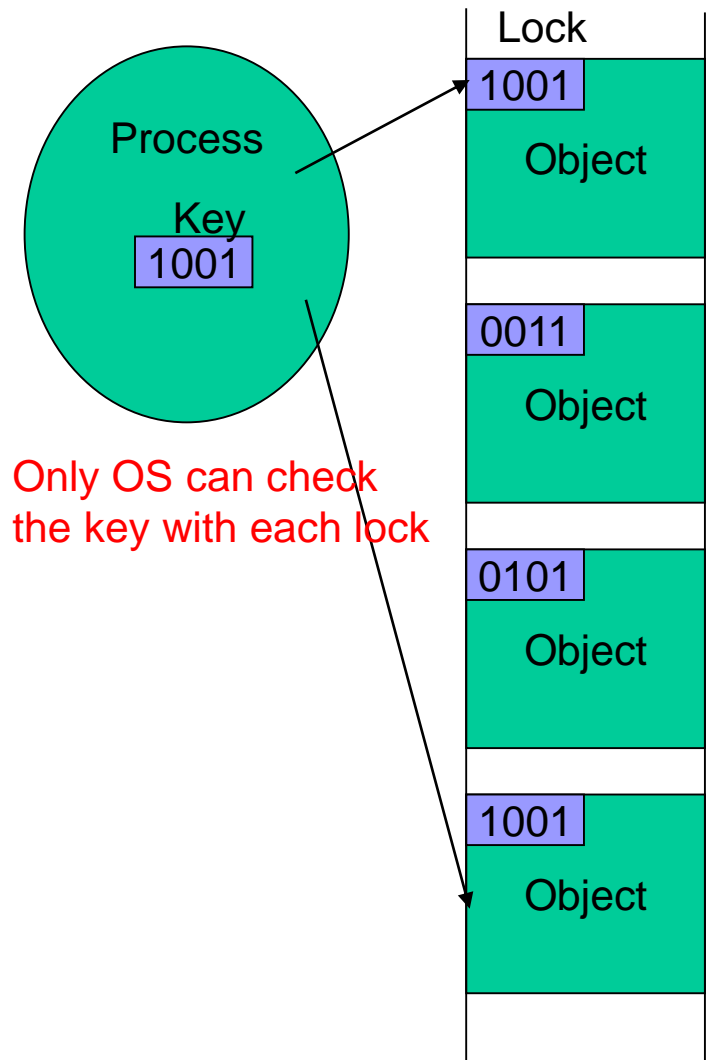
Users\Objects			
A	F1:RWX	F2:R-X	F3:RW-
B	F1:---	F2:RWX	F3:R--



# Implementing the Access Matrix

## 4. A Lock-Key Mechanism:

- It compromises between Access List and Capability List.
- Each object has a list of unique bit patterns called locks
- Each Domain has a list of unique bit patterns called Keys
  - A lock-key is managed by the OS on behalf of the domain.
  - There is no direct access by the user
- A process executing in a domain can access an object only if that domain has a key that matches one of the locks of the object.





## Revocable Access Rights

- The OS must be able to remove/revoke/cancel access rights when necessary.
- Various options to remove the access right of a domain to an object:
  - Immediate or delayed?
  - Selective (domain or user) or general?
  - Partial (rights) or full (all rights)?
  - Temporary or permanent?
- The implementation of the **Access Control Matrix** dictates how easy the above issues are implemented.
- i.e. with control lists, the list is searched for any access-right and remove it.

- Implementing Revocation using capability list requires:
- **Reacquisition**: Periodically delete capabilities from each domain. If the domain still needs the access right, it will ask for it again
- **Back-pointers**: Each object has a set of pointers to all domains which have access rights.
- **Indirection**: Capabilities point to a global table (whose location is known to the OS). To revoke a right, just edit the global table.
- **Keys**: If the Lock and Key method is used, simply change the lock values and force the processes/users to request new keys.

# HW or Language-Based Protection

- Motivation

- Comprehensive access validation requires considerable overheads and **satisfying all protection goals by the OS may be difficult.**
- There should be a support from **HW** or from the programming languages.
- Policies for resource use may vary

- Merits

- A HW or language-based system can provide security in addition to OS mechanisms.
- Software support when hardware supports are not available.

## Example: Protection in Java

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



**The End!!**

**Thank you**

**Any Questions?**