# Module 5

## Deadlocks Protection

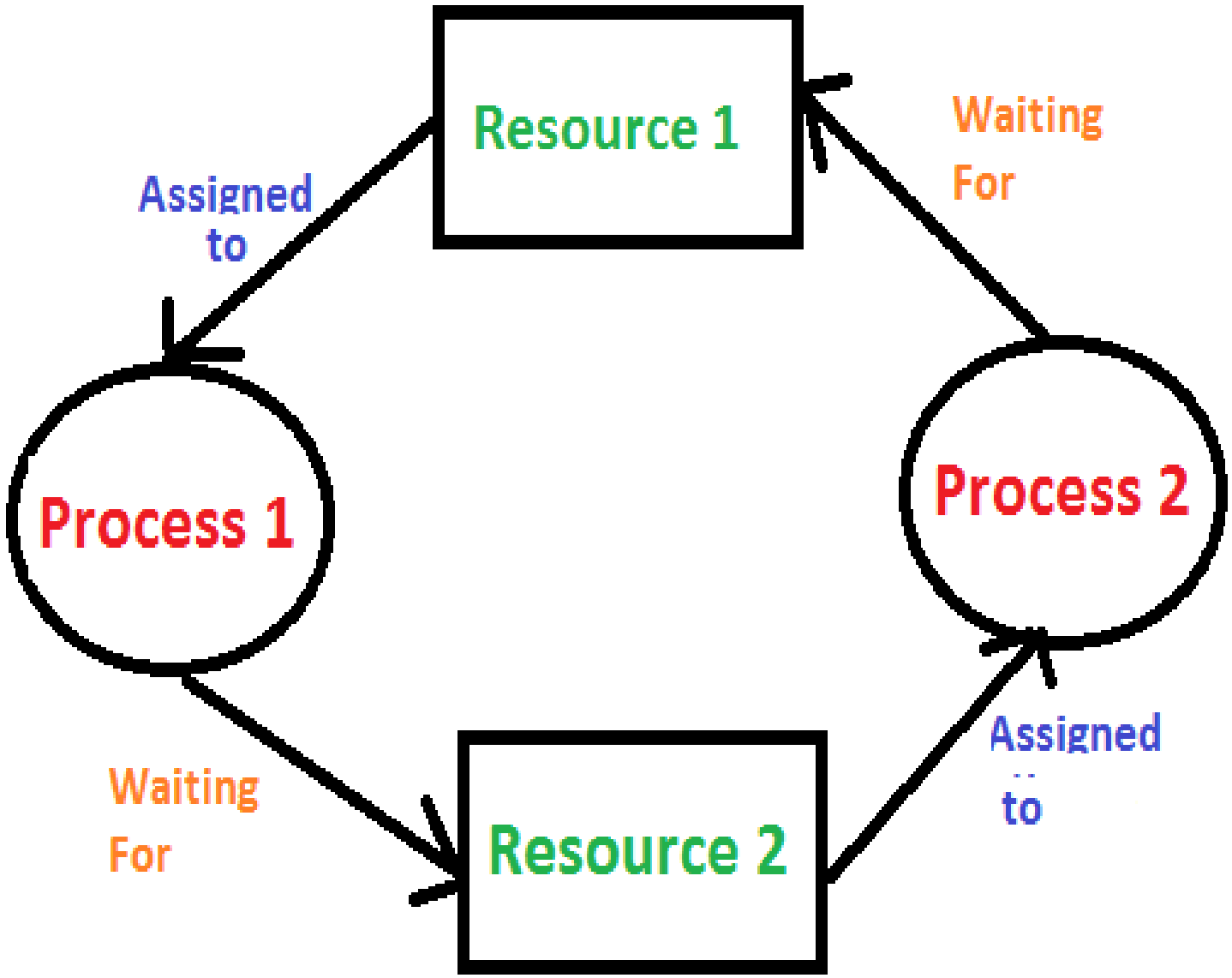
# Deadlock characterization

- A process in operating system uses resources in the following way.

1. Requests a resource
2. Use the resource
3. Releases the resource

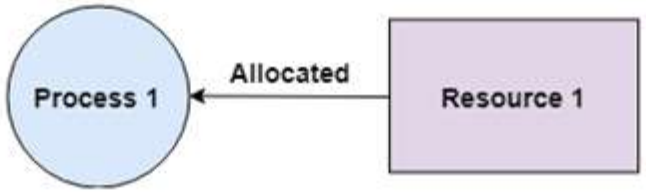
# Deadlock

- ***Deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
- Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other.
- A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).
- For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.

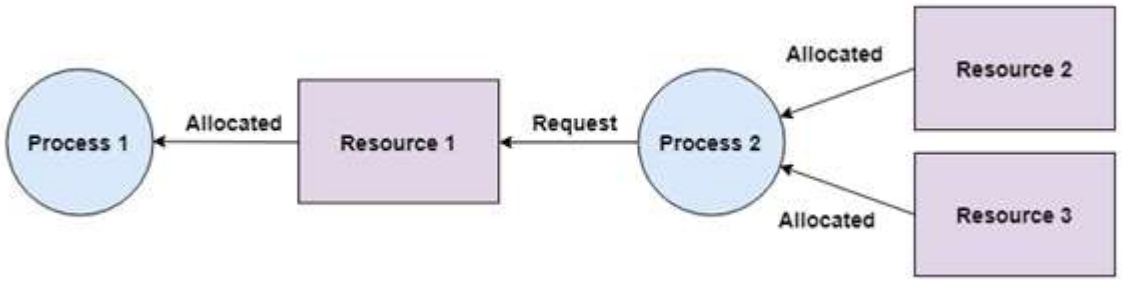

Resource 1
Assigned
to
Waiting
For
Process 1
Process 2
Waiting
For
Resource 2
Assigned
to

## **Deadlock can arise if** the **following four conditions hold simultaneously (Necessary Conditions)**

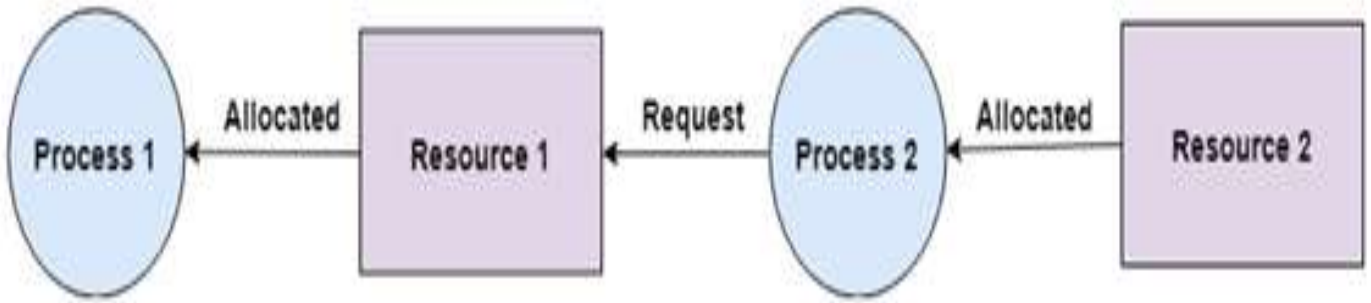
- ***Mutual Exclusion:*** Two or more resources are non-shareable (Only one process can use at a time)



- ***Hold and Wait:*** A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.

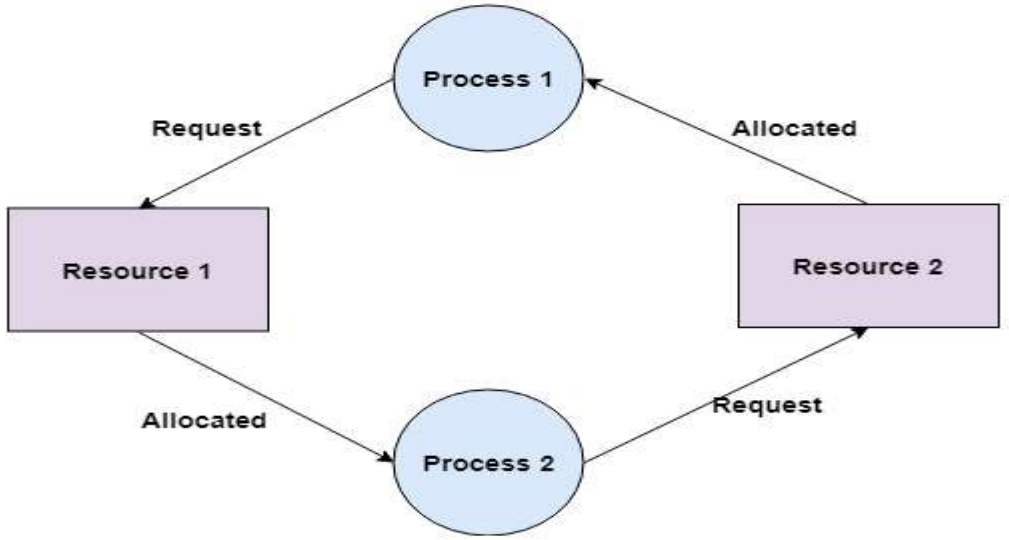


***No Preemption:*** A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



## *Circular Wait:*

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



# Methods for handling deadlock

## 1) Deadlock prevention or avoidance:

**Prevention:**

The idea is to not let the system into a deadlock state. This system will make sure that above mentioned four conditions will not arise. These techniques are very costly so we use this in cases where our priority is making a system deadlock-free. One can zoom into each category individually, Prevention is done by negating one of above mentioned necessary conditions for deadlock. Prevention can be done in four different ways:

1. Eliminate mutual exclusion
2. Solve hold and Wait
3. Allow preemption
4. Circular wait solution

**Avoidance:**

Avoidance is kind of futuristic. By using the strategy of “Avoidance”, we have to make an assumption. We need to ensure that all information about resources that the process will need is known to us before the execution of the process. We use Banker’s algorithm (Which is in turn a gift from Dijkstra) to avoid deadlock.

In prevention and avoidance, we get correctness of data but performance decreases.

**2) Deadlock detection and recovery:** If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:

1. In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
2. If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

- In Deadlock detection and recovery, we get the correctness of data but performance decreases.

### 3) Deadlock ignorance:

- If a deadlock is very rare, then let it happen and reboot the system.
- This is the approach that both Windows and UNIX take.
- we use the ostrich algorithm for deadlock ignorance.
- In Deadlock, ignorance performance is better than the above two methods but the correctness of data .

# Protection

- A mechanism that controls the access of programs, processes, or users to the resources defined by a computer system is referred to as protection.
- You may utilize protection as a tool for multi-programming operating systems, allowing multiple users to safely share a common logical namespace, including a directory or files.
- It needs the protection of computer resources like the software, memory, processor, etc.
- Users should take protective measures as a helper to multiprogramming OS so that multiple users may safely use a common logical namespace like a directory or data.
- Protection may be achieved by maintaining confidentiality, honesty and availability in the OS.
- It is critical to secure the device from unauthorized access, viruses, worms, and other malware

# Security violation categories

- Breach of confidentiality- unauthorised reading of data
- Breach of integrity- unauthorised modification of data
- Breach of availability- Unauthorised destruction of data
- Theft of service- Unauthorised use of resources
- Denial of service(DOS)- Prevention of legitimate use

## Need of Protection in Operating System

Various needs of protection in the operating system are as follows:

1. There may be security risks like unauthorized reading, writing, modification, or preventing the system from working effectively for authorized users.
2. It helps to ensure data security, process security, and program security against unauthorized user access or program access.
3. It is important to ensure no access rights' breaches, no viruses, no unauthorized access to the existing data.
4. Its purpose is to ensure that only the systems' policies access programs, resources, and data.

## Goals of Protection in Operating System

Various goals of protection in the operating system are as follows:

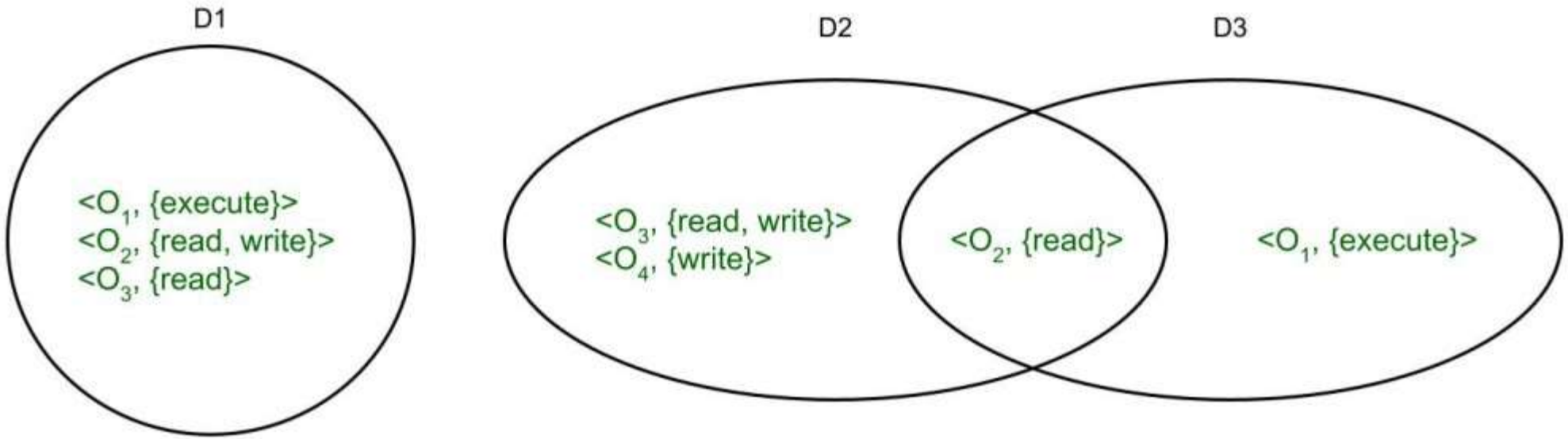
1. The policies define how processes access the computer system's resources, such as the CPU, memory, software, and even the operating system.
2. It is the responsibility of both the operating system designer and the app programmer. Although, these policies are modified at any time.
3. Protection is a technique for protecting data and processes from harmful or intentional infiltration.
4. It contains protection policies either established by itself, set by management or imposed individually by programmers to ensure that their programs are protected to the greatest extent possible.
5. It also provides a multiprogramming OS with the security that its users expect when sharing common space such as files or directories.

# Domain of protection

- A domain is a collection of access rights, each of which is an ordered pair <object name, rights-set>
- The protection policies limit the access of each process with respect to their resource handling.
- A process is bound to use only those resources which it requires to complete its task, in the time limit that it requires and also the mode in which it is required. That is the protected domain of a process.
- A computer system has processes and objects, which are treated as abstract data types, and these objects have operations specific to them.
- A domain element is described as <object, {set of operations on object}>.

- Each domain consists of a set of objects and the operations that can be performed on them.
- A domain can consist of either only a process or a procedure or a user.
- Then, if a domain corresponds to a procedure, then changing domain would mean changing procedure ID.
- Objects may share a common operation or two. Then the domains overlap.

- Hardware objects (such as the CPU, memory segments, printers, disks, and tape drives) and
- Software objects (such as files, programs, and semaphores).
- Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations.
- Objects are essentially abstract data types. The operations that are possible depend on the object.
- For example, on a CPU, we can only execute.
- Memory words can be read and written, whereas a DVD-ROM can only be read.
- Tape drives can be read, written, and rewound.
- Data files can be created, opened, read, written, closed, and deleted;
- program files can be read, written, executed, and deleted.


D1
D2
D3
<O1, {execute}>
<O2, {read, write}>
<O3, {read}>
<O3, {read, write}>
<O4, {write}>
<O2, {read}>
<O1, {execute}>

**Association between process and domain :**

Processes switch from one domain to other when they have the access right to do so. It can be of two types as follows.

**1.Fixed or static –**

In fixed association, all the access rights can be given to the processes at the very beginning but that give rise to a lot of access rights for domain switching. So, a way of changing the contents of the domain are found dynamically.

**2.Changing or dynamic –**

In dynamic association where a process can switch dynamically, creating a new domain in the process, if need be.

- A domain can be realized in a variety of ways:
- Each user may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed—generally when one user logs out and another user logs in.
- 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 occurs when one process sends a message to another process and then waits for a response.
- Each procedure 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.

# Access matrix

- **Access Matrix** is a security model of the protective state of a computer system.
- For each object, the permissions for every process executing in the domain are specified using an **access matrix**.
- Access matrix in OS is shown as a **two-dimensional matrix**, where the columns of the matrix represent **objects**, while the rows represent **domains**.
- Each matrix cell represents a specific set of access rights that are granted to processes of the domain this indicates that each entry access (i, j) specifies the set of actions that a process executing in domain Di may invoke on object Oj.
- The access matrix implements policy decisions and these policy decisions involve which rights should be included in the (i, j)th entry like reading, writing, and executing. (This policy is usually decided by the operating system).

# Implementation of access matrix

- The access matrix in the operating system likely occupies a significant amount of memory and is very sparse.
- Therefore, it is storage inefficient to directly implement an access matrix for access control.
- The access matrix can be subdivided into rows or columns to reduce the inefficiency.
- To increase efficiency, the columns, and rows can be collapsed by deleting null values. **Four widely used access matrix implementations can be formed using these decomposition methods**:
- Global Table
- Capability Lists for Domains
- Access Lists for Objects
- Lock and key Method

## Various Methods Used to Implement the Access Matrix in OS

- Some widely used methods for implementing the access matrix in os are as follows:

### 1. Global Table

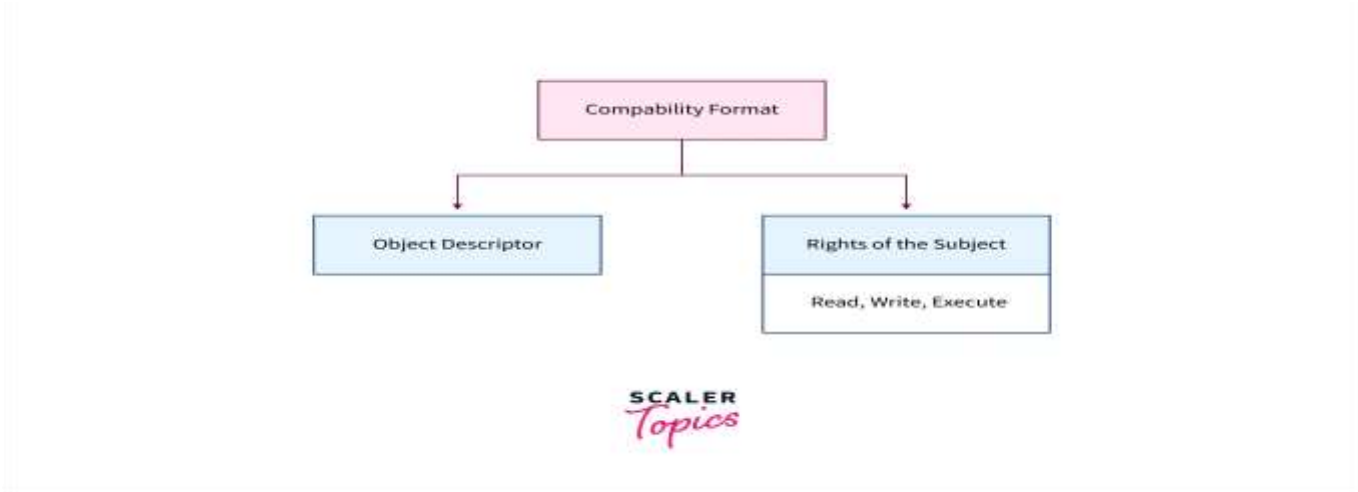
- The global table is the most basic and simple implementation of the access matrix in the operating system which consists of a set of an ordered triple **<domain , object, right-set>**.
- When an operation M is being executed on an object Oj within domain Di, the global table searches for a triple **<Domain(Di), Object(Oj), right-set(Rk)>** where M € Rk.
- If the triple is present, the operation can proceed to continue, or else a condition of an **exception** is thrown.
- There are various drawbacks to this implementation, the main **drawback** of global table implementation is that because the table is sometimes too large, it cannot be stored in the main memory, that's why input and output are required additionally.

## 2. Access Lists

- In the **Access Lists** method, the access matrix in OS is divided into columns (Column wise decomposition).
- When an operation M is being executed on an object Oj within domain Di, We search for an entry **<Domain(Di), right-set(Rk)>** with M € Rk in the access list for object Oj.
- If the triple is present, the operation can proceed to continue, or else we check the initial set.
- If M is included in the default set, access is allowed; otherwise, access is denied, and an exception is raised.

## 3. Capability Lists

- In the access matrix in the operating system, Capability Lists is a **collection of objects** and the operations that can be performed on them.
- The object here is specified by a physical name called **capability**.
- In this method, we can associate each row with its domain instead of connecting the columns of the access matrix to the objects as an access list.



## 4. Lock-Key Mechanism

- It is a **comparison** between capability lists and access lists.
- Every domain has a distinct bit pattern called **keys**, and every object has a distinct bit pattern called **locks**.
- Only if a domain's key matches one of the locks of the object, A process can access it.
- In simple words, When a process running in a specific domain (Di) try to access an object (Oj) then the **key** of that Di must match with the lock of that Oj, then only an object can be accessed.
- The operating system should **handle the keys and locks** in such a way that any unauthorized access should not be allowed on them.

## Examples

- Consider the below example for a better understanding of implementing an **access matrix in the operating system**.
- Now, let's take an example when there are four files having the following access rights (files f1, f2, f3, and f4) and three domains (D1, D2, D3)

| ACCESS MATRIX | | | | |
|---|---|---|---|---|
| Object / Domain | File 1 | File 2 | File 3 | File 4 |
| D1 | *read*<br>*write* | *read* | *read* | *read* |
| D2 | *read* | *write* | - | *exceute* |
| D3 | *write* | - | *write* | *exceute* |

SCALER Topics

- The access matrix, as seen in the figure above, here represents the set of access rights as:
- Any file can be **read** by a process running in the D1 domain and can only be **written** into f1.
- A process in D2 domain has access to **read** f1, **write** to f2, and **execute** f4.
- D3 processes in the domain can **write** to f1 and 3 and can **execute** f4.
- The **contents** of the access-matrix entries are normally decided by the users. The access matrix's Oj column is added when a user creates a new object, Oj, with the proper initialization entries as provided by the creator. Processes should be able to switch between different domains.
- An access matrix can be used to specify and implement both **static and dynamic access rights**. When a process switches between different domains, it's similar to when we do a **(domain switch)** operation on an object (the domain). The table below demonstrates the possibility of domain switches from the D1 domain to D2, D2 to D3, and D2 to D1.

| ACCESS MATRIX WITH DOMAIN AS OBJECTS | | | | | | |
|---|---|---|---|---|---|---|
| Object / Domain | File 1 | File 2 | File 3 | D1 | D2 | D3 |
| D1 | *read*<br>*write* | *read* | *write* | - | *switch* | - |
| D2 | *write* | *exceute* | *write* | *switch* | - | *switch* |
| D3 | *read* | - | *exceute* | - | - | - |

SCALER Topics

# Access Control

- Access control is a fundamental component of data security that dictates who's allowed to access and use company information and resources.
- Through authentication and authorization, access control policies make sure users are who they say they are and that they have appropriate access to company data.
- Access control can also be applied to limit physical access to campuses, buildings, rooms, and datacenters.

# How does access control work?

- Access control identifies users by verifying various login credentials, which can include usernames and passwords, PINs, biometric scans, and security tokens.
- Many access control systems also include **multifactor authentication (MFA),** a method that requires multiple authentication methods to verify a user's identity.
- Once a user is authenticated, access control then authorizes the appropriate level of access and allowed actions associated with that user's credentials and IP address.
- There are four main types of access control. Organizations typically choose the method that makes the most sense based on their unique security and compliance requirements. The four access control models are:

1. **Discretionary access control (DAC):** In this method, the owner or administrator of the protected system, data, or resource sets the policies for who is allowed access.
2. **Mandatory access control (MAC):** In this nondiscretionary model, people are granted access based on an information clearance. A central authority regulates access rights based on different security levels. This model is common in government and military environments.
3. **Role-based access control (RBAC):** RBAC grants access based on defined **business functions** rather than the **individual user's identity**. The goal is to provide users with access only to data that's been deemed necessary for their roles within the organization. This widely used method is based on a complex combination of role assignments, authorizations, and permissions.
4. **Attribute-based access control (ABAC):** In this dynamic method, access is based on a set of attributes and environmental conditions, such as time of day and location, assigned to both users and resources.

# Why is access control important?

- Access control keeps confidential information such as customer data, personally identifiable information, and intellectual property from falling into the wrong hands.
- It’s a key component of the modern zero trust security framework, which uses various mechanisms to continuously verify access to the company network.
- Without robust access control policies, organizations risk data leakage from both internal and external sources.
- Access control is particularly important for organizations with hybrid cloud and multi-cloud cloud environments, where resources, apps, and data reside both on premises and in the cloud.
- Access control can provide these environments with more robust access security beyond single sign-on (SSO), and prevent unauthorized access from unmanaged and BYO devices.

# Revocation of access rights

- In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users.

Various questions about revocation may arise:

- **Immediate versus delayed:** Does revocation occur immediately, or is it delayed? If revocation is delayed, can we find out when it will take place?
- **Selective versus general:** When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?
- **Partial versus total:** Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
- **Temporary versus permanent:** Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?

- With an access-list scheme, revocation is easy. The access list is searched for any access rights to be revoked, and they are deleted from the list.
- Revocation is immediate and can be general or selective, total or partial, and permanent or temporary.
- Capabilities, however, present a much more difficult revocation problem, as mentioned earlier.
- Since the capabilities are distributed throughout the system, we must find them before we can revoke them.
- Schemes that implement revocation for capabilities include the following:

- **Reacquisition**:
- Periodically, capabilities are deleted from each domain.
- If a process wants to use a capability, it may find that that capability has been deleted.
- The process may then try to reacquire the capability.
- If access has been revoked, the process will not be able to reacquire the capability.
- **Back-pointers**:
- A list of pointers is maintained with each object, pointing to all capabilities associated with that object.
- When revocation is required, we can follow these pointers, changing the capabilities as necessary.
- This scheme was adopted in the MULTICS system.
- It is quite general, but its implementation is costly.

## Indirection:

- The capabilities point indirectly, not directly, to the objects.
- Each capability points to a unique entry in a global table, which in turn points to the object.
- We implement revocation by searching the global table for the desired entry and deleting it.
- Then, when an access is attempted, the capability is found to point to an illegal table entry.
- Table entries can be reused for other capabilities without difficulty, since both the capability and the table entry contain the unique name of the object.
- The object for a capability and its table entry must match.
- This scheme was adopted in the CAL system. It does not allow selective revocation.

## Keys

- A key is a unique bit pattern that can be associated with a capability.
- This key is defined when the capability is created, and it can be neither modified nor inspected by the process that owns the capability.
- A master key is associated with each object; it can be defined or replaced with the set-key operation.
- When a capability is created, the current value of the master key is associated with the capability.
- When the capability is exercised, its key is compared with the master key.
- If the keys match, the operation is allowed to continue; otherwise, an exception condition is raised.
- Revocation replaces the master key with a new value via the set-key operation, invalidating all previous capabilities for this object.

# Capability Based System

- The concept of capability-based protection was introduced in the early 1970s.
- Two early research systems were Hydra and CAP.
- Neither system was widely used, but both provided interesting proving grounds for protection theories.
- Here, we consider two more contemporary approaches to capabilities.

# Linux Capabilities

- Linux uses capabilities to address the limitations of the UNIX model, which we described earlier.
- The POSIX standards group introduced capabilities in POSIX 1003.1e.
- Although POSIX.1e was eventually withdrawn, Linux was quick to adopt capabilities in Version 2.2 and has continued to add new developments.
- In essence, Linux's capabilities "slice up" the powers of root into distinct areas, each represented by a bit in a bitmask, as shown in Figure 17.11. Fine grained control over privileged operations can be achieved by toggling bits in the bitmask.
- In practice, three bitmasks are used—denoting the capabilities permitted, effective, and inheritable. Bitmasks can apply on a per-process or a per-thread basis.
- Furthermore, once revoked, capabilities cannot be reacquired.

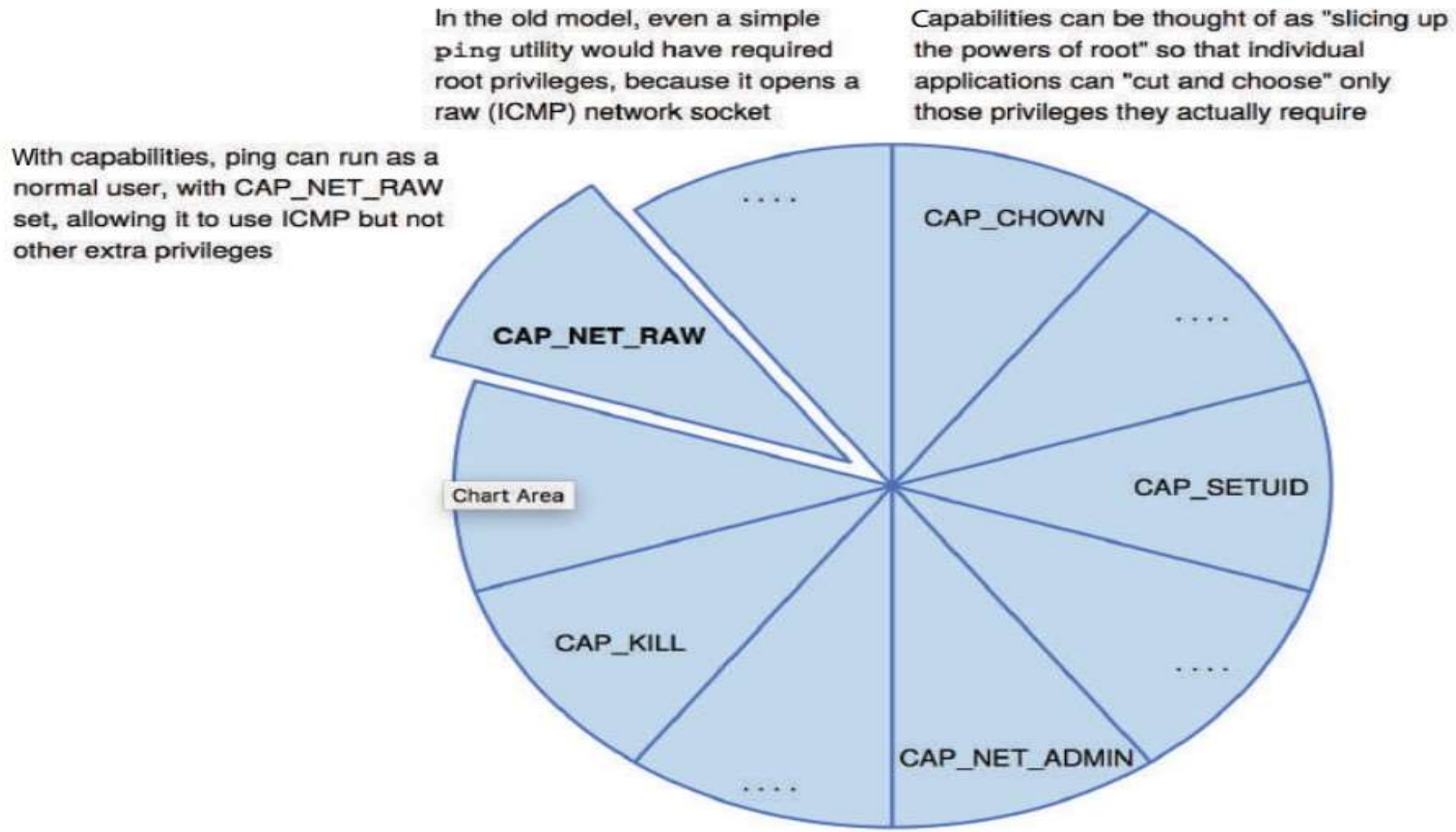



**Figure 17.11** Capabilities in POSIX.1e.

- The usual sequence of events is that a process or thread starts with the full set of permitted capabilities and voluntarily decreases that set during execution.
- For example, after opening a network port, a thread might remove that capability so that no further ports can be opened.
- You can probably see that capabilities are a direct implementation of the principle of least privilege.
- As explained earlier, this tenet of security dictates that an application or user must be given only those rights than are required for its normal operation.
- Android (which is based on Linux) also utilizes capabilities, which enable system processes (notably, “system server”), to avoid root ownership, instead selectively enabling only those operations required. The

- Linux capabilities model is a great improvement over the traditional UNIX model, but it still is inflexible.
- For one thing, using a bitmap with a bit representing each capability makes it impossible to add capabilities dynamically and requires recompiling the kernel to add more.
- In addition, the feature applies only to kernel-enforced capabilities.

# Language Based Protection

- To the degree that protection is provided in computer systems, it is usually achieved through an operating-system kernel, which acts as a security agent to inspect and validate each attempt to access a protected resource.
- Since comprehensive access validation may be a source of considerable overhead, either we must give it hardware support to reduce the cost of each validation, or we must allow the system designer to compromise the goals of protection.
- Satisfying all these goals is difficult if the flexibility to implement protection policies is restricted by the support mechanisms provided or if protection environments are made larger than necessary to secure greater operational efficiency

- Policies for resource use may also vary, depending on the application, and they may be subject to change over time.
- For these reasons, protection can no longer be considered a matter of concern only to the designer of an operating system.
- It should also be available as a tool for use by the application designer, so that resources of an application subsystem can be guarded against tampering or the influence of an error.

# Compiler-Based Enforcement

- At this point, programming languages enter the picture.
- Specifying the desired control of access to a shared resource in a system is making a declarative statement about the resource.
- This kind of statement can be integrated into a language by an extension of its typing facility.
- When protection is declared along with data typing, the designer of each subsystem can specify its requirements for protection, as well as its need for use of other resources in a system.
- Such a specification should be given directly as a program is composed, and in the language in which the program itself is stated.
- This approach has several significant advantages:

1. Protection needs are simply declared, rather than programmed as a sequence of calls on procedures of an operating system.
2. Protection requirements can be stated independently of the facilities provided by a particular operating system.
3. The means for enforcement need not be provided by the designer of a subsystem.
4. A declarative notation is natural because access privileges are closely related to the linguistic concept of data type.

# Run-Time-Based Enforcement—Protection in Java

- Because Java was designed to run in a distributed environment, the Java virtual machine—or JVM—has many built-in protection mechanisms.
- Java programs are composed of classes, each of which is a collection of data fields and functions (called methods) that operate on those fields.
- The JVM loads a class in response to a request to create instances (or objects) of that class.
- One of the most novel and useful features of Java is its support for dynamically loading untrusted classes over a network and for executing mutually distrusting classes within the same JVM.

- Because of these capabilities, protection is a paramount concern.
- Classes running in the same JVM may be from different sources and may not be equally trusted.
- As a result, enforcing protection at the granularity of the JVM process is insufficient. Intuitively, whether a request to open a file should be allowed will generally depend on which class has requested the open. The operating system lacks this knowledge.
- Thus, such protection decisions are handled within the JVM.
- When the JVM loads a class, it assigns the class to a protection domain that gives the permissions of that class.
- The protection domain to which the class is assigned depends on the URL from which the class was loaded and any digital signatures on the class file.

- For example, classes loaded from a trusted server might be placed in a protection domain that allows them to access files in the user’s home directory, whereas classes loaded from an untrusted server might have no file access permissions at all.
- It can be complicated for the JVM to determine what class is responsible for a request to access a protected resource.
- Accesses are often performed indirectly, through system libraries or other classes. For example, consider a class that is not allowed to open network connections.
- It could call a system library to request the load of the contents of a URL.
- The JVM must decide whether or not to open a network connection for this request. But which class should be used to determine if the connection should be allowed, the application or the system library?

- The philosophy adopted in Java is to require the library class to explicitly permit a network connection.
- More generally, in order to access a protected resource, some method in the calling sequence that resulted in the request must explicitly assert the privilege to access the resource.
- By doing so, this method takes responsibility for the request. Presumably, it will also perform whatever checks are necessary to ensure the safety of the request.
- Of course, not every method is allowed to assert a privilege; a method can assert a privilege only if its class is in a protection domain that is itself allowed to exercise the privilege

- This implementation approach is called stack inspection. Every thread in the JVM has an associated stack of its ongoing method invocations.
- When a caller may not be trusted, a method executes an access request within a doPrivileged block to perform the access to a protected resource directly or indirectly.
- doPrivileged() is a static method in the AccessController class that is passed a class with a run() method to invoke.
- When the doPrivileged block is entered, the stack frame for this method is annotated to indicate this fact. Then, the contents of the block are executed. When an access to a protected resource is subsequently requested, either by this method or a method it calls, a call to checkPermissions() is used to invoke stack inspection to determine if the request should be allowed.

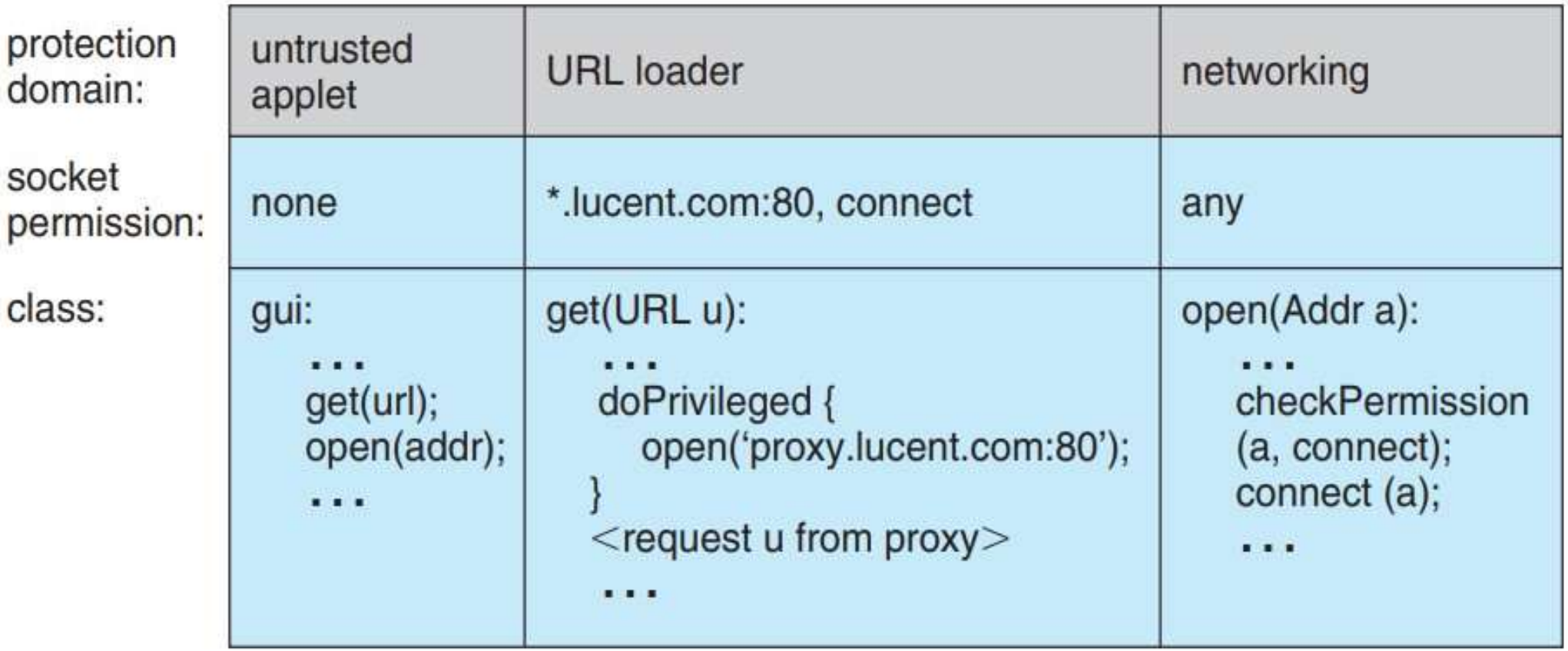



**Figure 17.14** Stack inspection.

- The inspection examines stack frames on the calling thread’s stack, starting from the most recently added frame and working toward the oldest.
- If a stack frame is first found that has the doPrivileged() annotation, then checkPermissions() returns immediately and silently, allowing the access.
- If a stack frame is first found for which access is disallowed based on the protection domain of the method’s class, then checkPermissions() throws an AccessControlException.
- If the stack inspection exhausts the stack without finding either type of frame, then whether access is allowed depends on the implementation (some implementations of the JVM may allow access, while other implementations may not).

- Stack inspection is illustrated in Figure 17.14. Here, the gui() method of a class in the untrusted applet protection domain performs two operations, first a get() and then an open().
- The former is an invocation of the get() method of a class in the URL loader protection domain, which is permitted to open() sessions to sites in the lucent.com domain, in particular a proxy server proxy.lucent.com for retrieving URLs.
- For this reason, the untrusted applet's get() invocation will succeed: the checkPermissions() call in the networking library encounters the stack frame of the get() method, which performed its open() in a doPrivileged block.
- However, the untrusted applet's open() invocation will result in an exception, because the checkPermissions() call finds no doPrivileged annotation before encountering the stack frame of the gui() method.

- More generally, Java’s load-time and run-time checks enforce type safety of Java classes.
- Type safety ensures that classes cannot treat integers as pointers, write past the end of an array, or otherwise access memory in arbitrary ways.
- Rather, a program can access an object only via the methods defined on that object by its class.
- This is the foundation of Java protection, since it enables a class to effectively encapsulate and protect its data and methods from other classes loaded in the same JVM.
- For example, a variable can be defined as private so that only the class that contains it can access it or protected so that it can be accessed only by the class that contains it, subclasses of that class, or classes in the same package.
- Type safety ensures that these restrictions can be enforced.