https://brown-csci1660.github.io

# CS1660: Intro to Computer Systems Security
# Spring 2026

## Lecture 15: OS Security II

Instructor: **Nikos Triandopoulos**

March 19, 2026

BROWN

# CS1660: Announcements

- Course updates
  - Project 1, project 2, HW 1, HW2, midterm
  - Project 3 is out, due March 31 (likely to be extended)
  - One lecture today, then Spring break!

# Last class

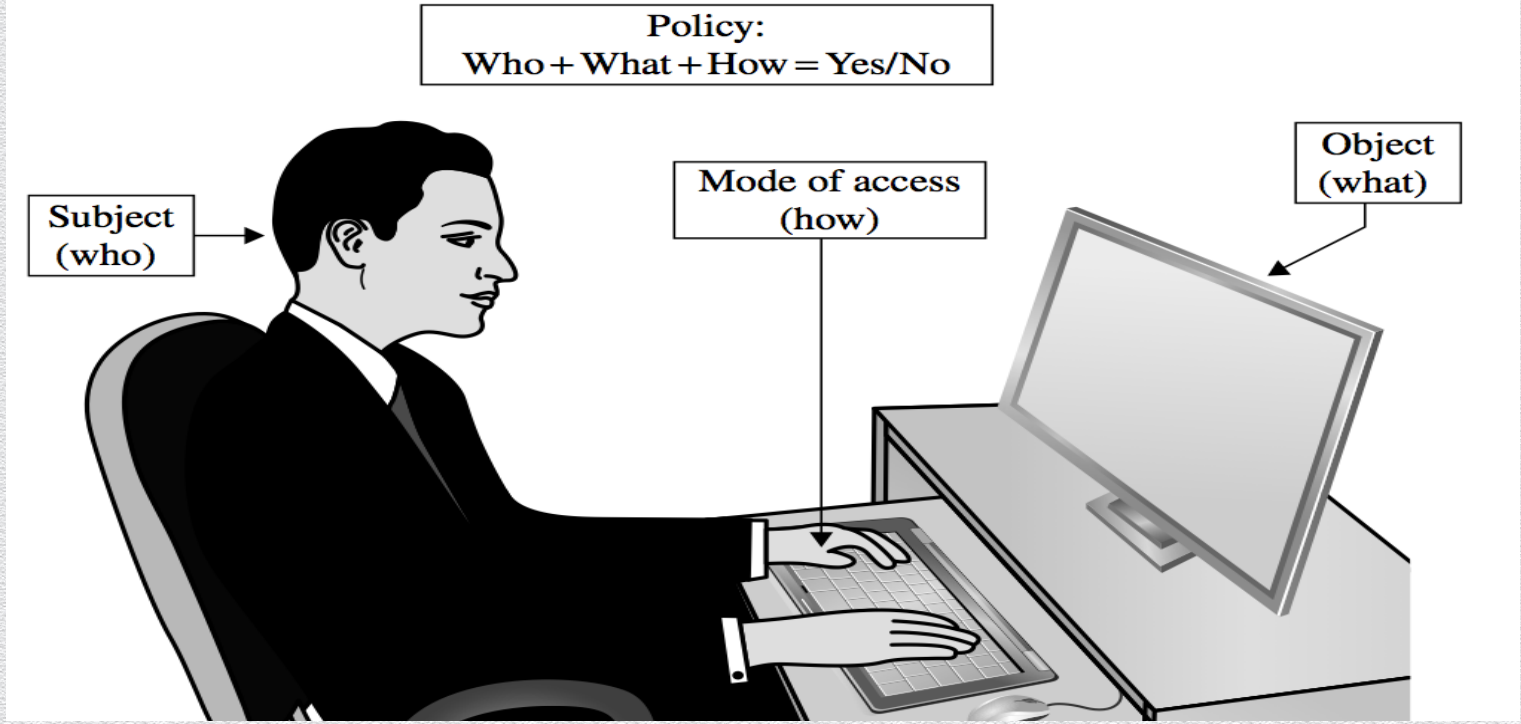
- Cryptography
- Authentication
- Web security
  - Cross-site references and risks
    - SQL injection + Cross-site scripting (XSS)
    - Database security + Buffer overflow attacks
  - Operating system (OS) security
    - Authorization and Access Control (AC)

# Today

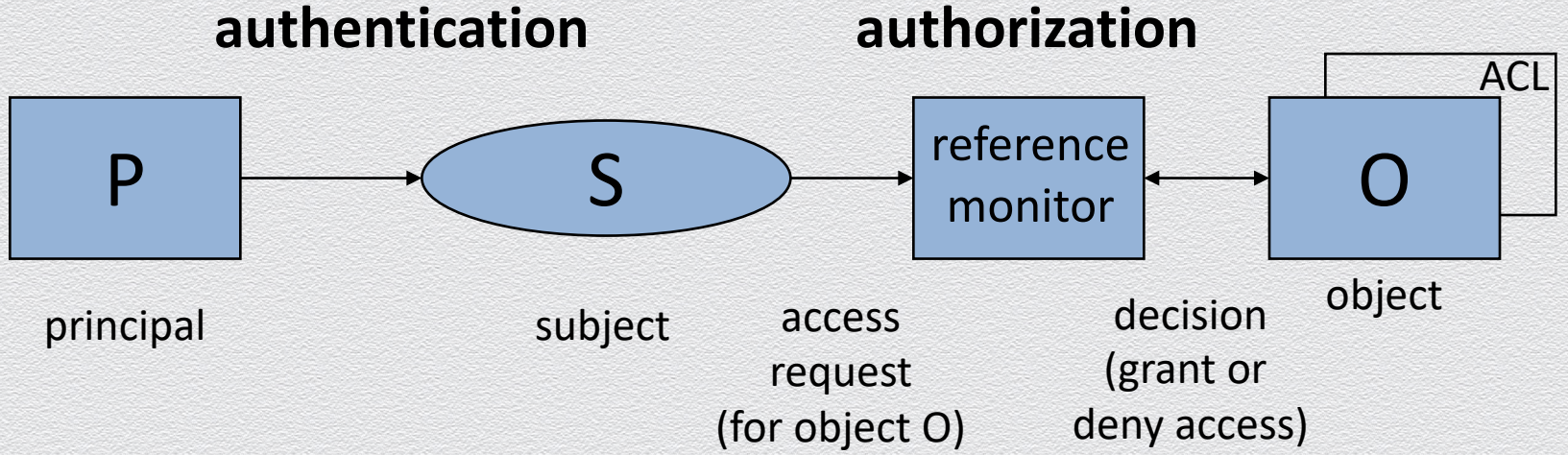
- Cryptography
- Authentication
- Web security
  - Cross-site references and risks
    - SQL injection + Cross-site scripting (XSS)
    - Database security + Buffer overflow attacks
- Operating system (OS) security
  - Access control, OS access control, file-system access control

# 15.1 Access control

# Access control (AC)



# General structure of access control mechanism



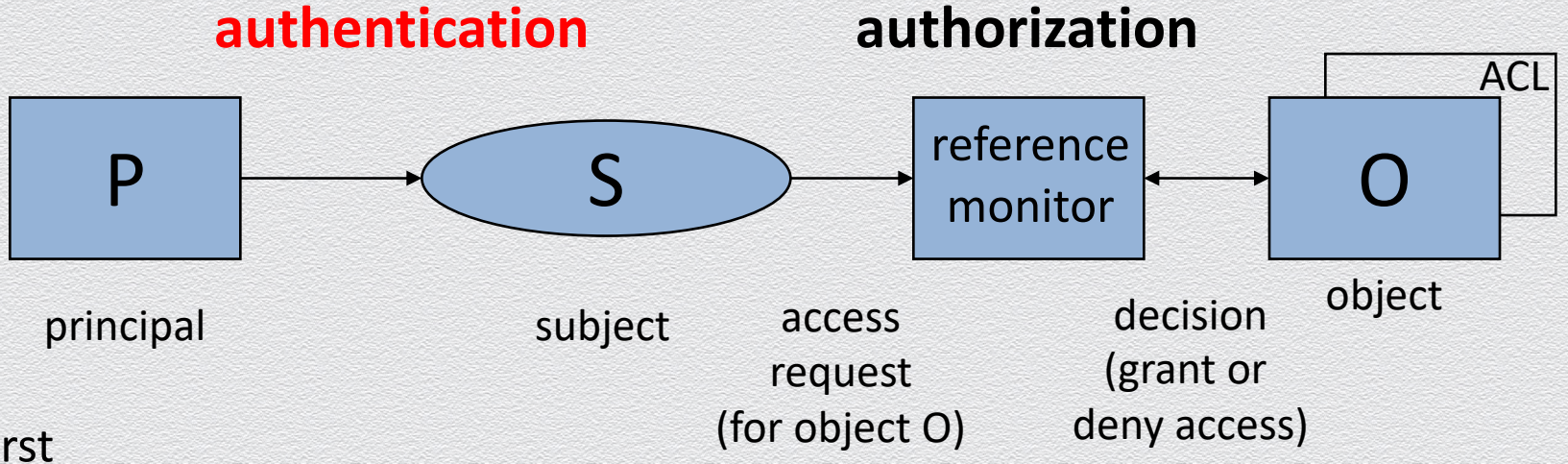
# Basic terminology

- Subject/Principal
  - active entity – user or process
- Object
  - passive entity – file or resource
- Access operations
  - vary from basic memory access (read, write) to method calls in object-oriented systems
  - comparable systems may use different access operations or attach different meanings to operations which appear to be the same

# Access operation

- Access right
  - right to perform an (access) operation
- Permission
  - typically, a synonym for access right
- Privilege
  - typically, a set of access rights given directly to roles like administrator, operator, ...

# Authentication




First

- reference monitor verifies the identity of the principal making the request
  - a user identity is one example for a principal
  - cf. authentication Vs. identification

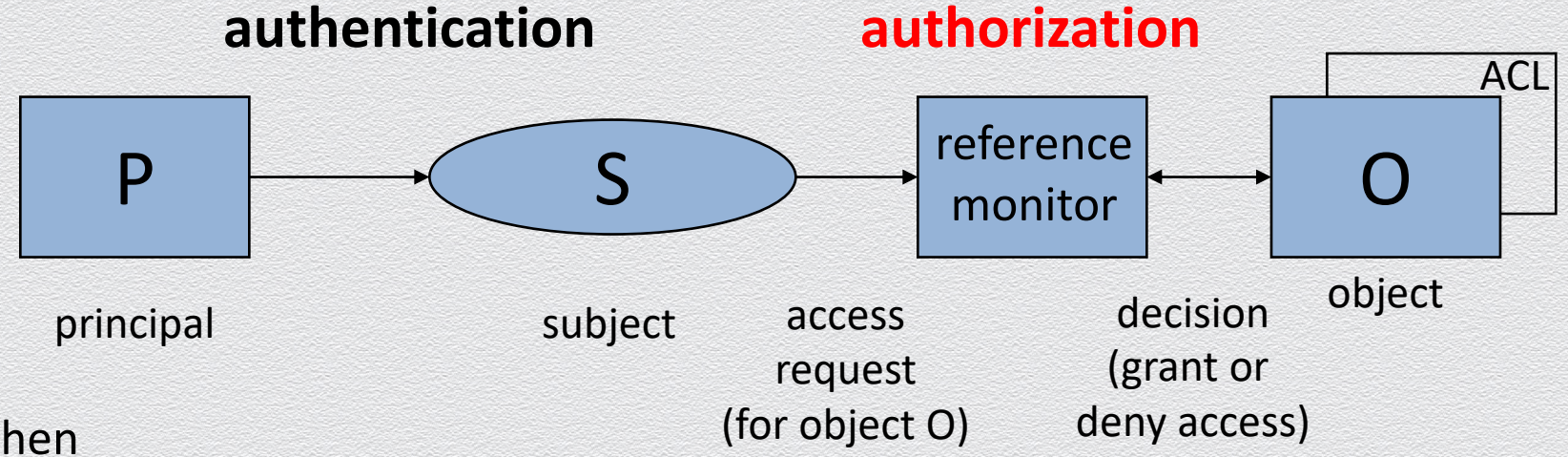
# Authentication

- user enters username and password
- if the values entered are correct, the user is “authenticated”
- we could say: “The machine now runs on behalf of the user”
  - this might be intuitive, but it is imprecise
- log on creates a process that “runs with access rights” assigned to the user
  - the process runs under the user identity of the user who has logged on

# Users & user identities

- requests to reference monitor do not come directly from a user or a user identity, but from a process
- in the language of access control, the process “speaks for” the user (identity)
- the active entity making a request within the system is called the subject
- must distinguish between three concepts
  - user: person
  - principal: identity (e.g., user name) used in the system, possibly associated with a user
  - subject: process running under a given user identity

# Authorization



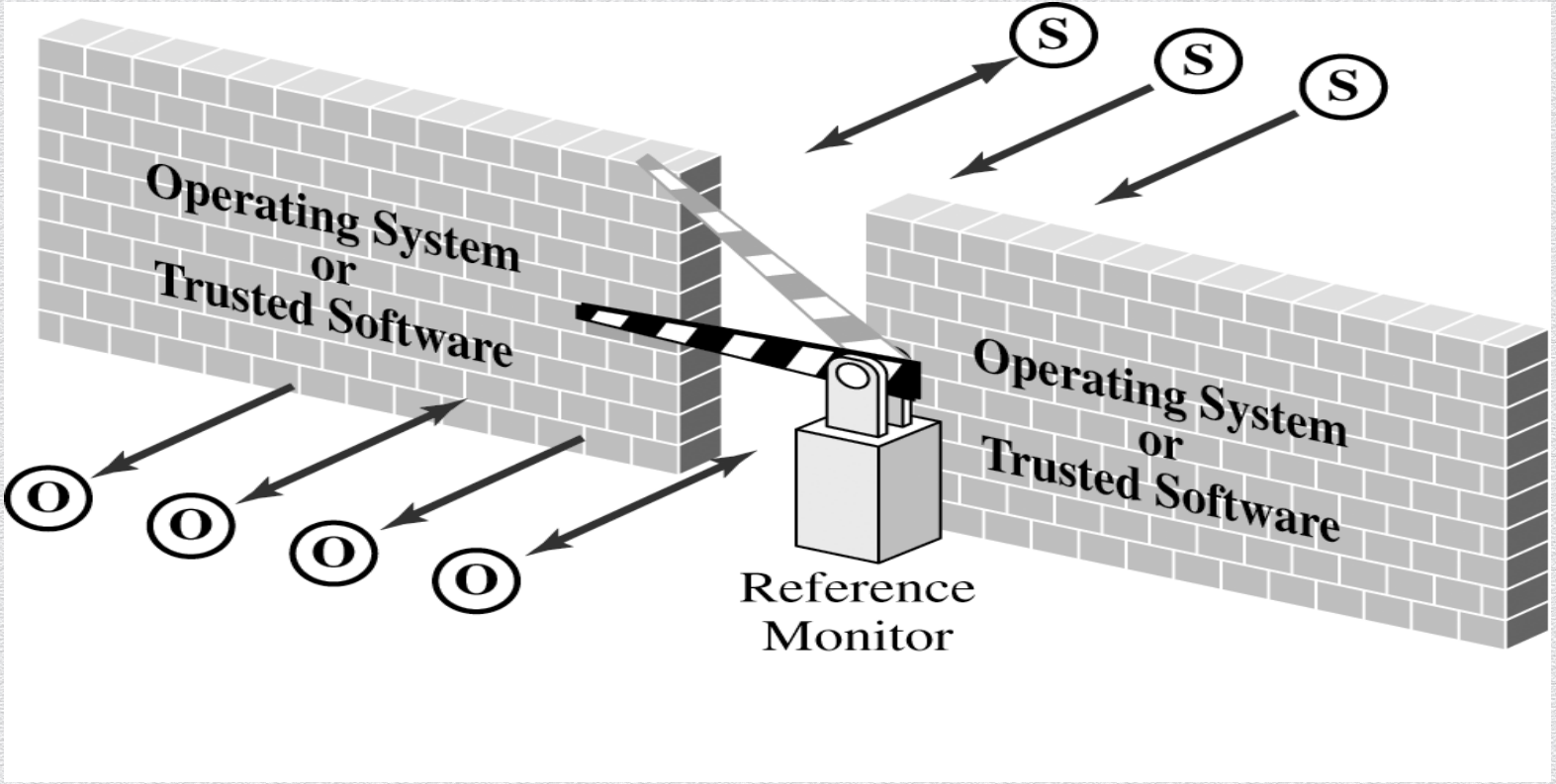

Then

- reference monitor decides whether access is granted or denied
- has to find and evaluate the security policy relevant for the given request
- “easy” in centralized systems; in distributed systems,
  - how to find all relevant policies? how to make decisions if policies may be missing?

# Principals & subjects

- a principal is an entity that can be granted access to objects or can make statements affecting access control decisions
  - example: user ID
- subjects operate on behalf of (human users we call) principals
- access is based on the principal's name bound to the subject in some unforgeable manner at authentication time
  - example: process (running under a user ID)

# Reference monitor



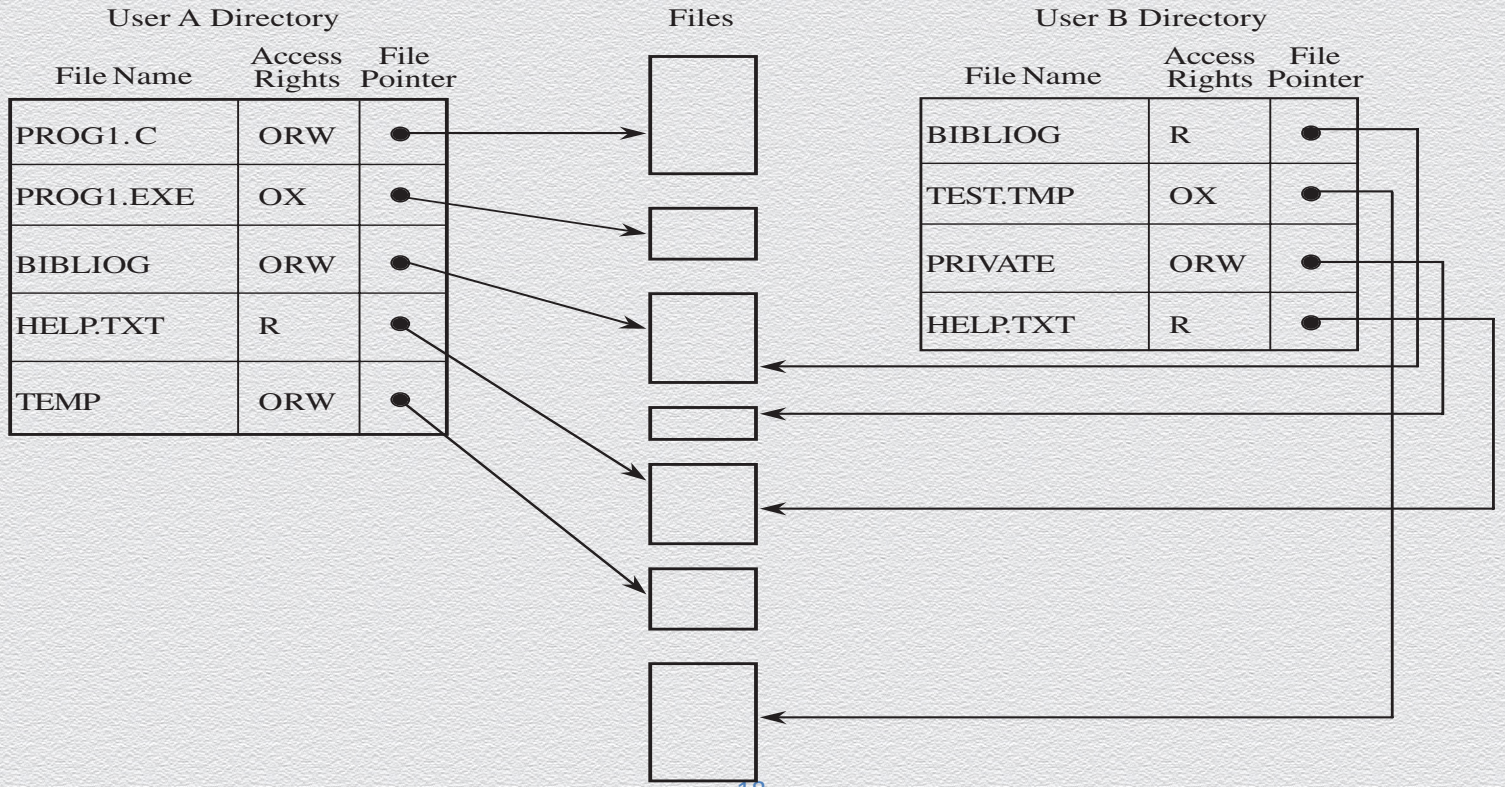
# AC policies

- Goals
  - Check every access
  - Enforce least privilege
  - Verify acceptable usage
- Track users' access
- Enforce at appropriate granularity
- Use audit logging to track accesses

# Implementing AC policies

- Reference monitor
- Access control directory
- Access control matrix
- Access control list
- Privilege list
- Capability
- Procedure-oriented access control
- Role-based access control

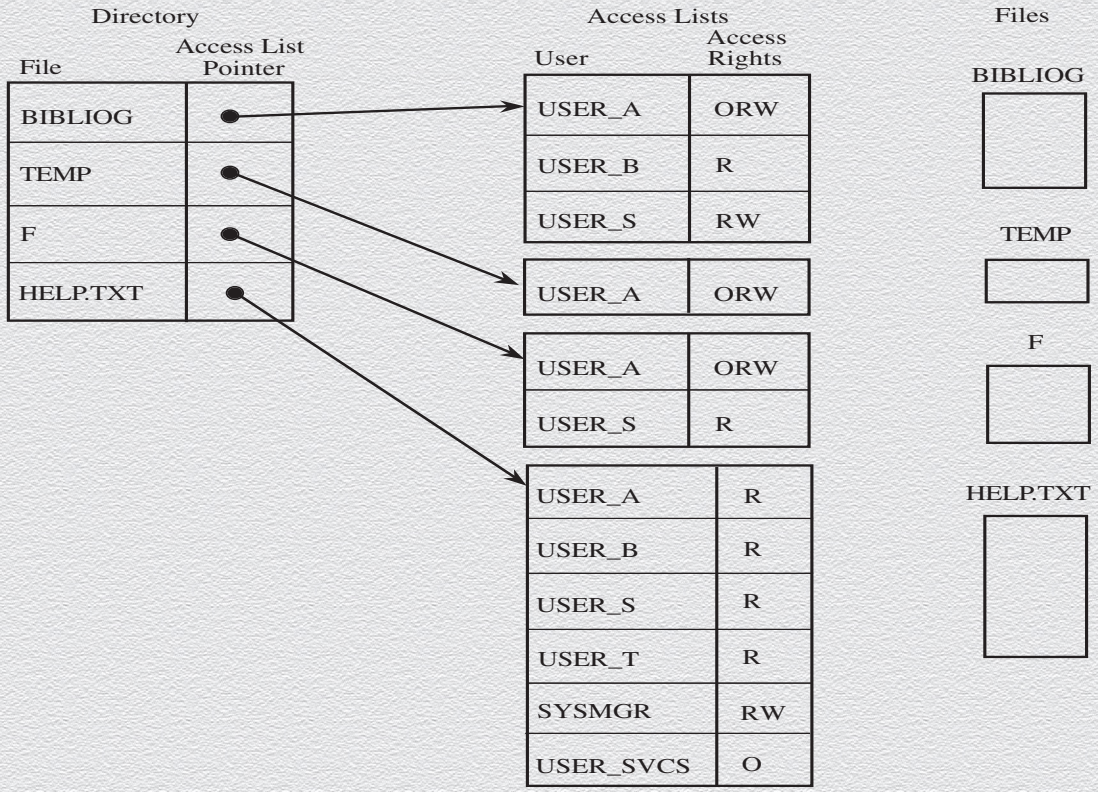
# Access control directory



# Access control matrix

| | BIBLIOG | TEMP | F | HELP.TXT | C_COMP | LINKER | SYS_CLOCK | PRINTER |
|---|---|---|---|---|---|---|---|---|
| **USER A** | ORW | ORW | ORW | R | X | X | R | W |
| **USER B** | R | - | - | R | X | X | R | W |
| **USER S** | RW | - | R | R | X | X | R | W |
| **USER T** | - | - | - | R | X | X | R | W |
| **SYS_MGR** | - | - | - | RW | OX | OX | ORW | O |
| **USER_SVCS** | - | - | - | O | X | X | R | W |

# Access control list



# Basic access control and information flow models

- Discretionary access control (DAC)
  - owner determines access rights
  - typically, identity-based access control: access rights are assigned to users based on their identity
  - e.g., ACM
- Mandatory access control (MAC)
  - system enforce system-wide rules for access control
  - e.g., law allows a court to access driving records without the owners' permission

# DAC

- In DAC the user (e.g., owner of resources/files) is responsible for deciding how information is accessed
- Local access decisions of users might conflict with each other
- Basic terms
  - Access control matrix
  - Security policy (specifying who has the access rights to what)
  - Security mechanism (enforce security policies)

# DAC and MAC

- When is DAC insufficient?
  - when owner cannot be trusted for the discretion of the data and external protection of the data is necessary
  - e.g., DAC has the danger of right propagation
    - A can read X and write Y
    - B can read Y, but no access to X
    - A reads X, write the content of X to Y, B got access to X
- MAC
  - non-discretionary
  - labels are assigned to subjects and objects
  - owner has no special privileges
  - e.g., Bell-LaPadula, lattices models, SELinux by NSA

# Traditional models for MAC

- Bell-LaPadula (BLP)
  - About confidentiality
- Biba
  - About integrity with static/dynamic levels

# Bell-LaPadula security model

- The Bell-LaPadula (BLP) model is about information confidentiality
- It was developed to formalize the US Department of Defense multilevel security policy

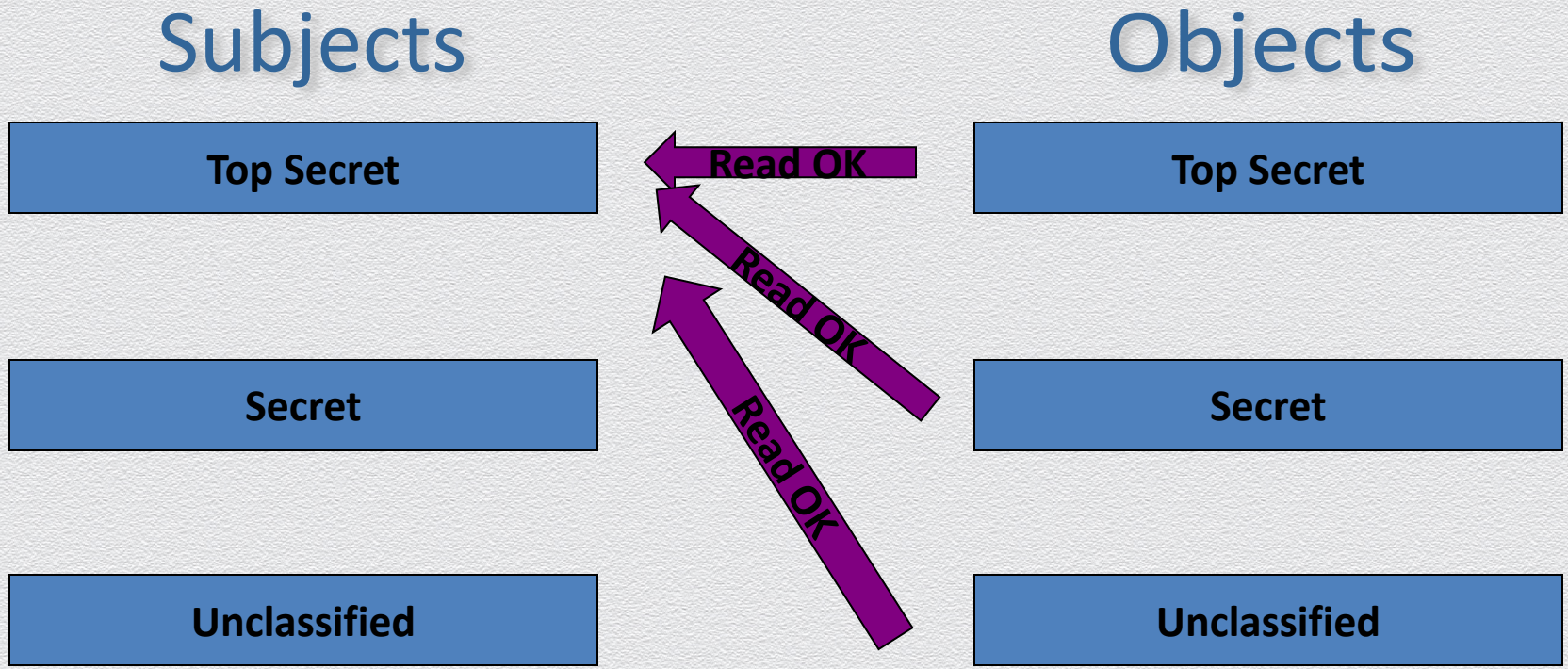
# Bell – LaPadula - details

- Each user subject and information object has a fixed security class – labels
- Use the notation **≤** to indicate **dominance**
- Simple Security (ss) property:

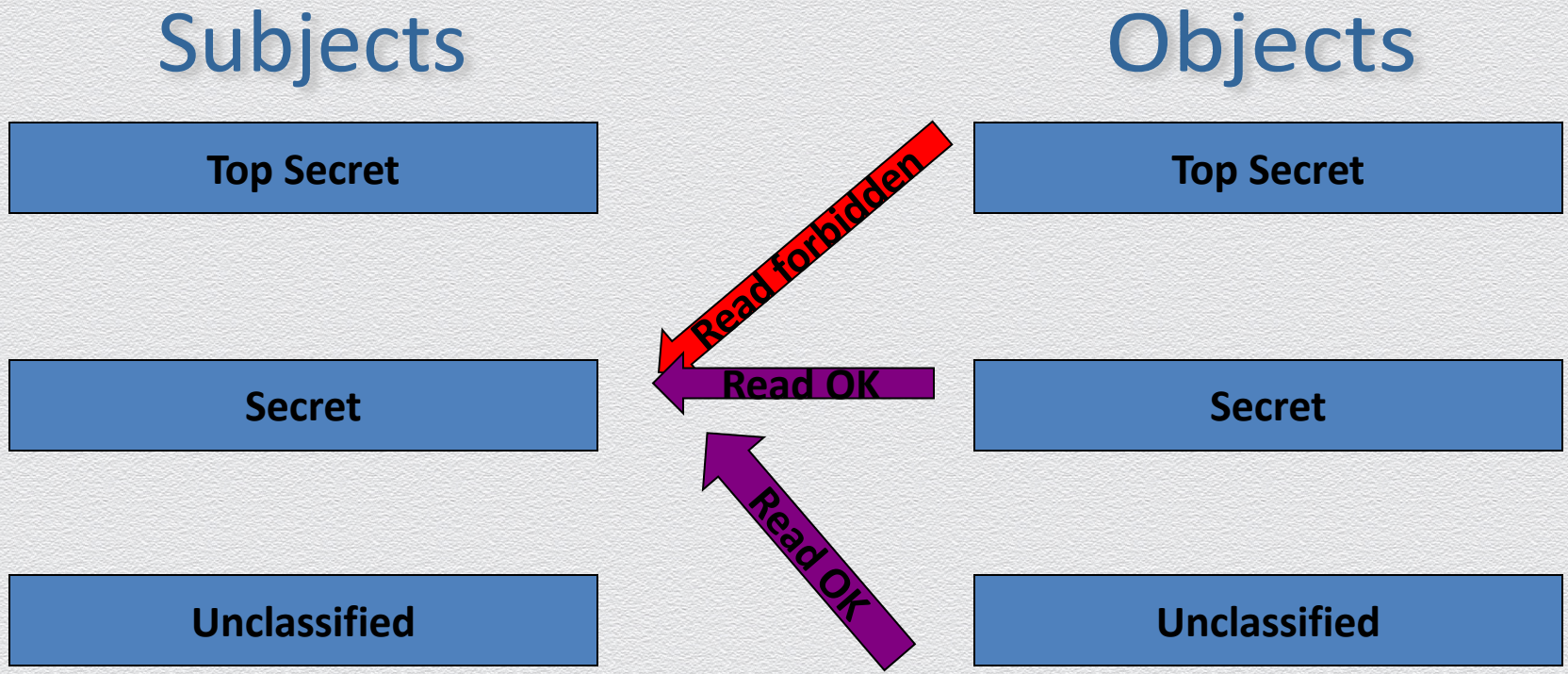
  **no read-up property**

  - a subject s has read access to an object o iff the class of the subject C(s) is greater than or equal to the class of the object C(o)
  - i.e. subjects s can read objects o iff C(o) ≤ C(s)

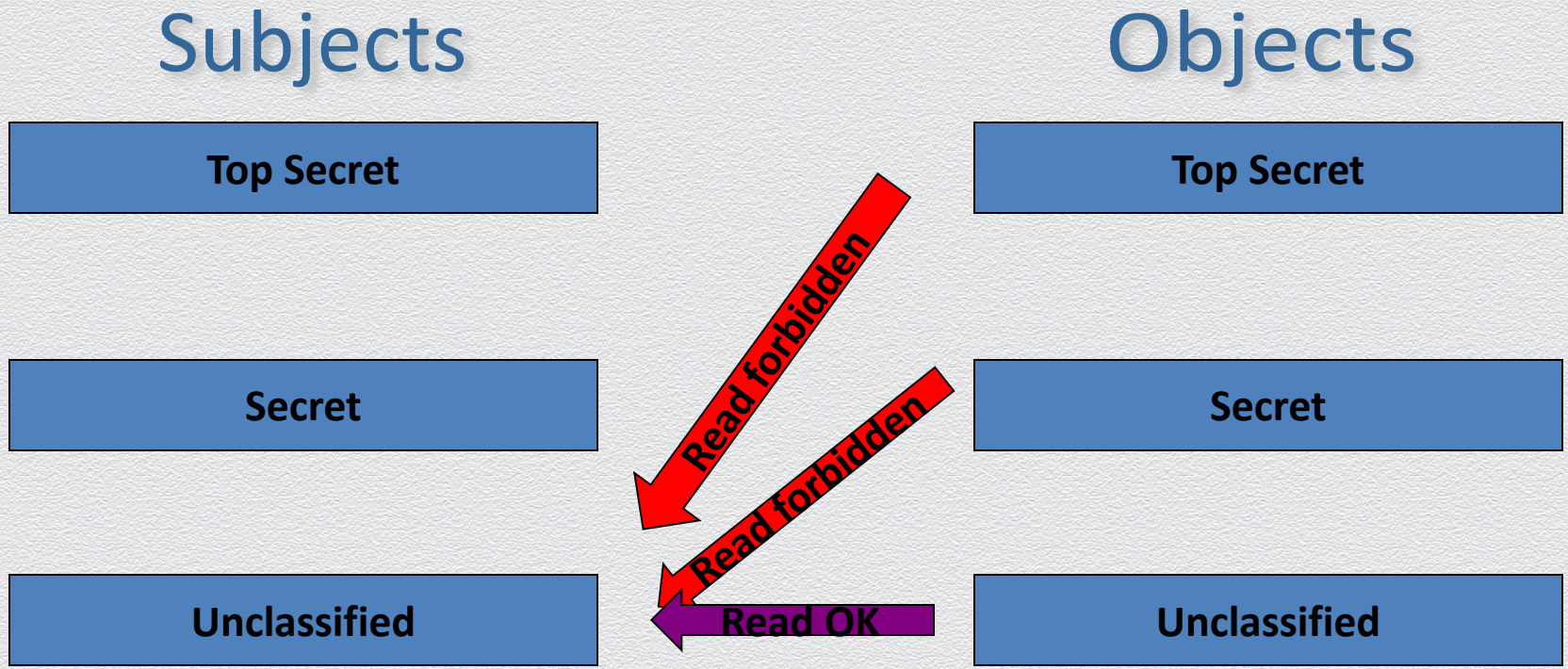
# Access control: Bell-LaPadula



# Access control: Bell-LaPadula



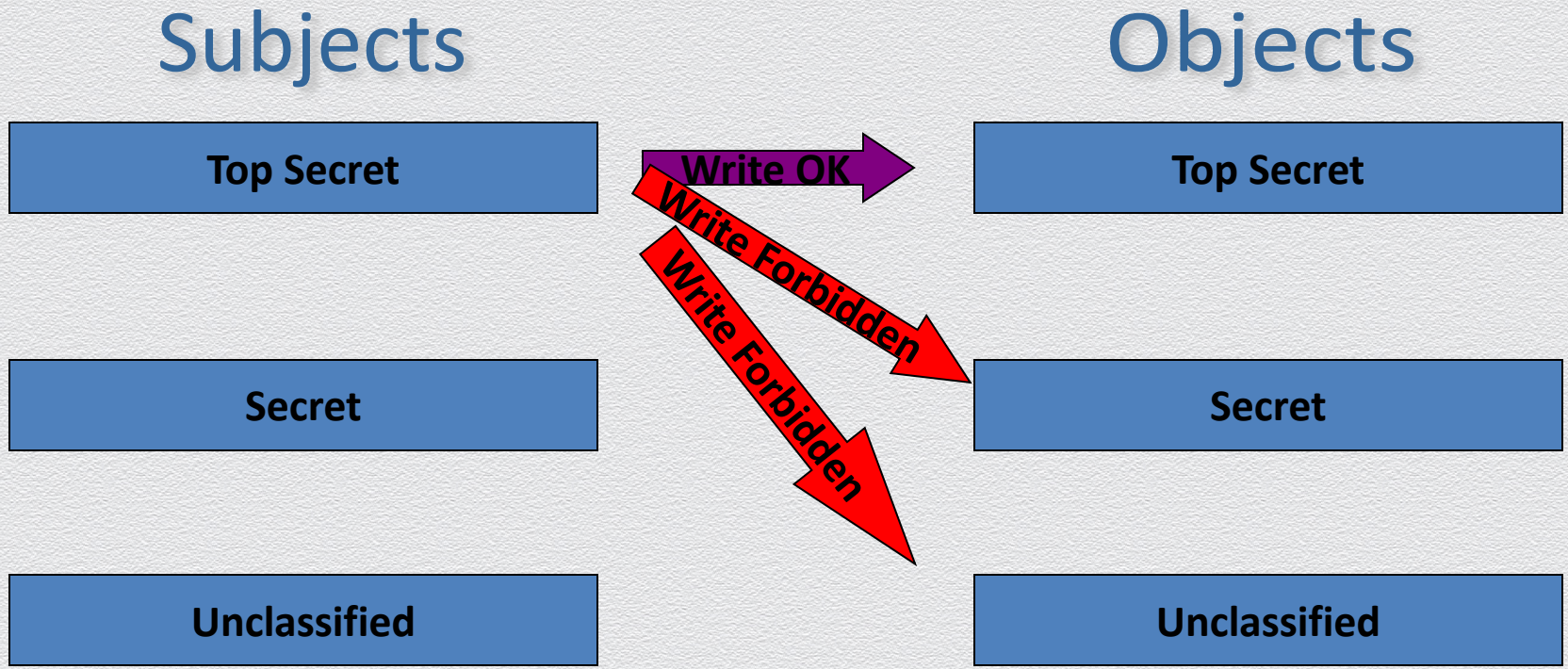
# Access control: Bell-LaPadula



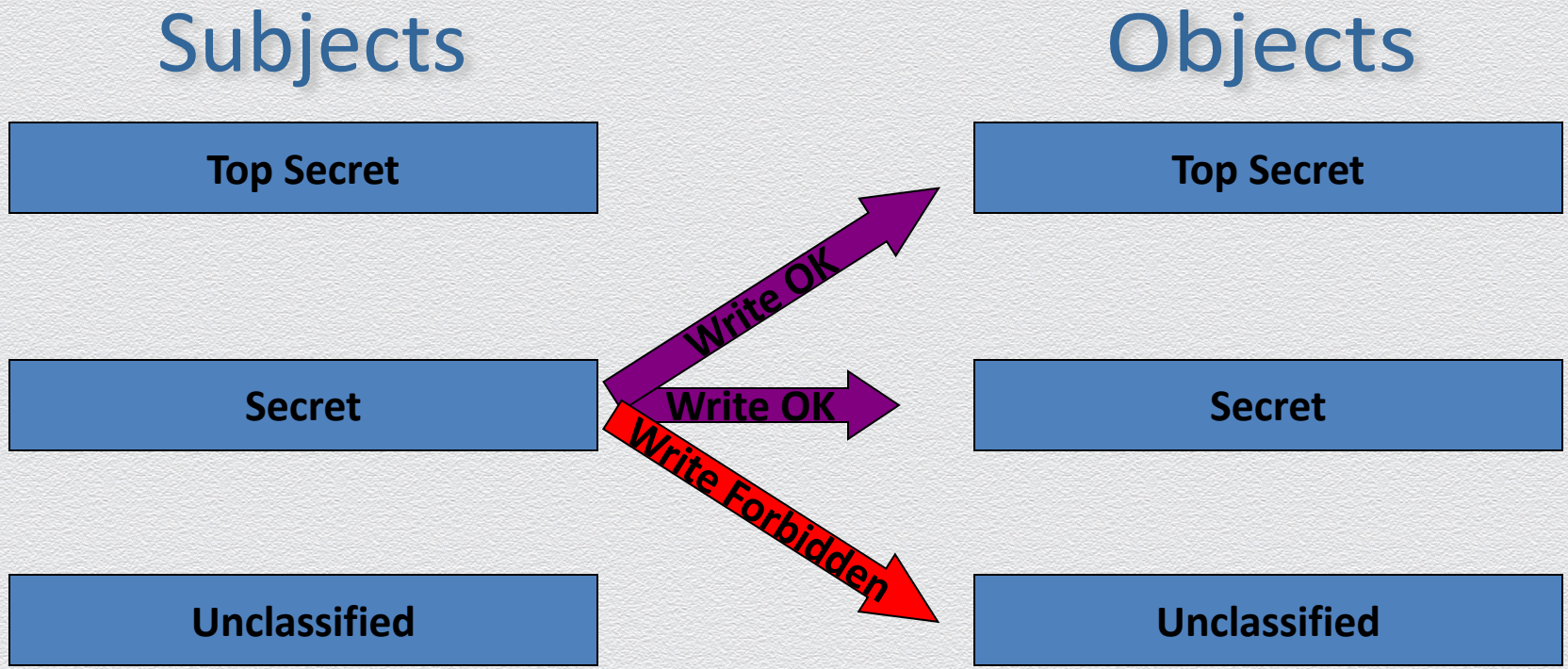
# Bell - LaPadula (2)

- * property (star):
  the no write-down property
  - A subject s can write to object *p* if C(s) ≤ C (p)

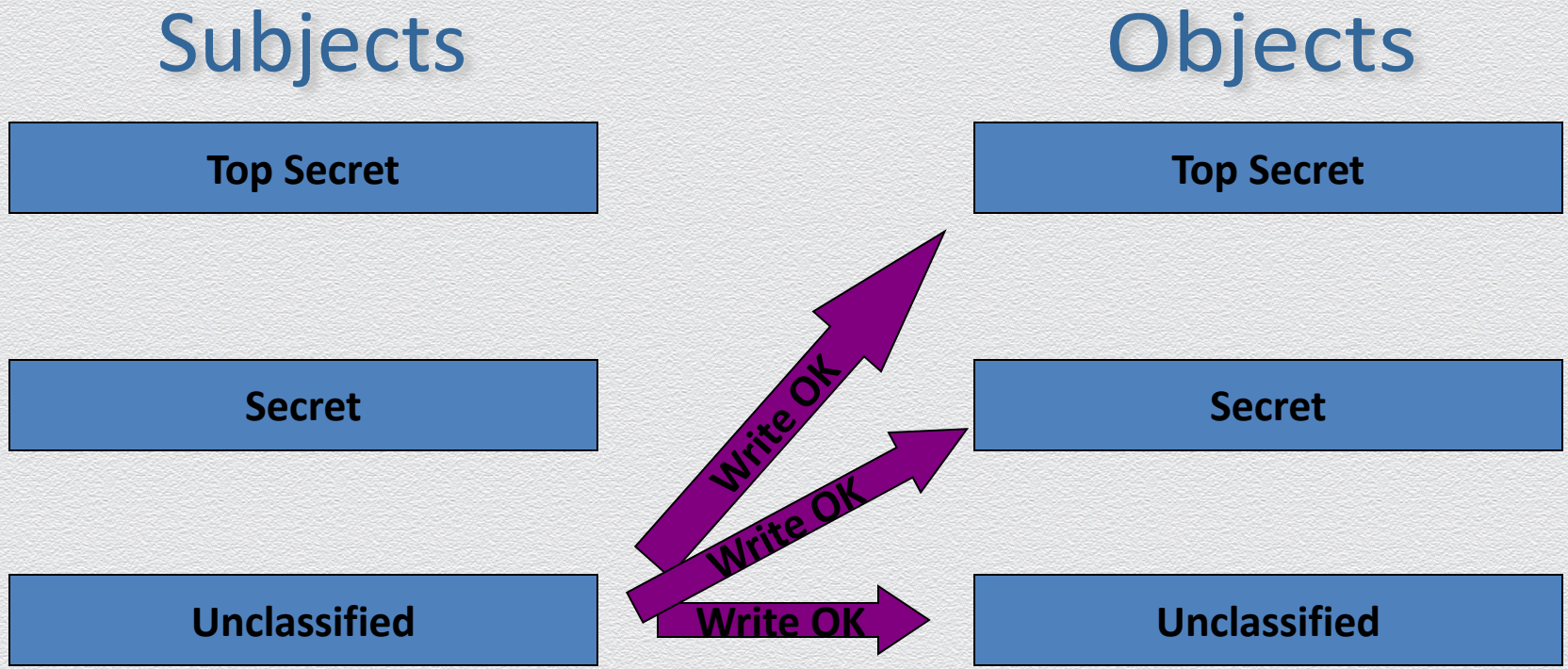
# Access control: Bell-LaPadula



# Access control: Bell-LaPadula



# Access control: Bell-LaPadula



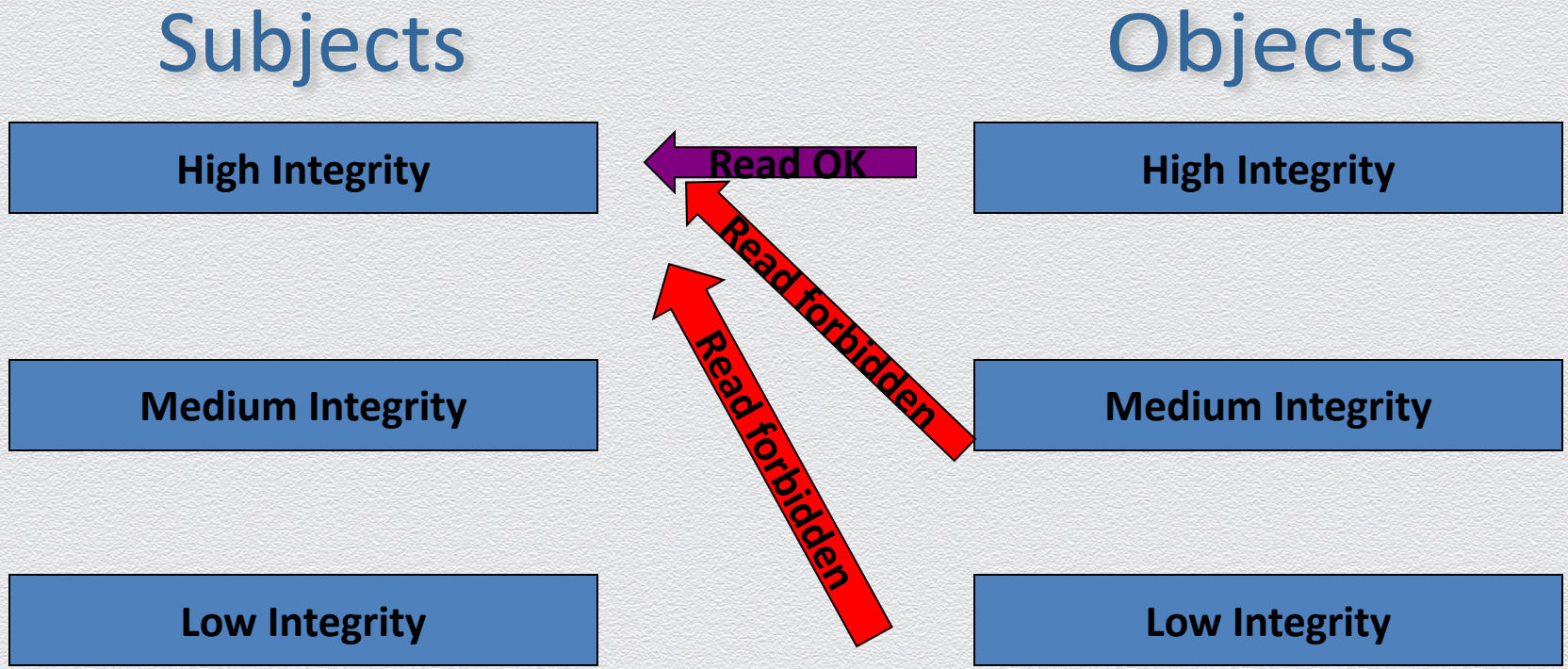
# Security models - Biba

- Based on the Cold War experiences, information *integrity* is also important, and the Biba model, complementary to Bell-LaPadula, is based on the flow of information where preserving integrity is critical.
- The “dual” of Bell-LaPadula

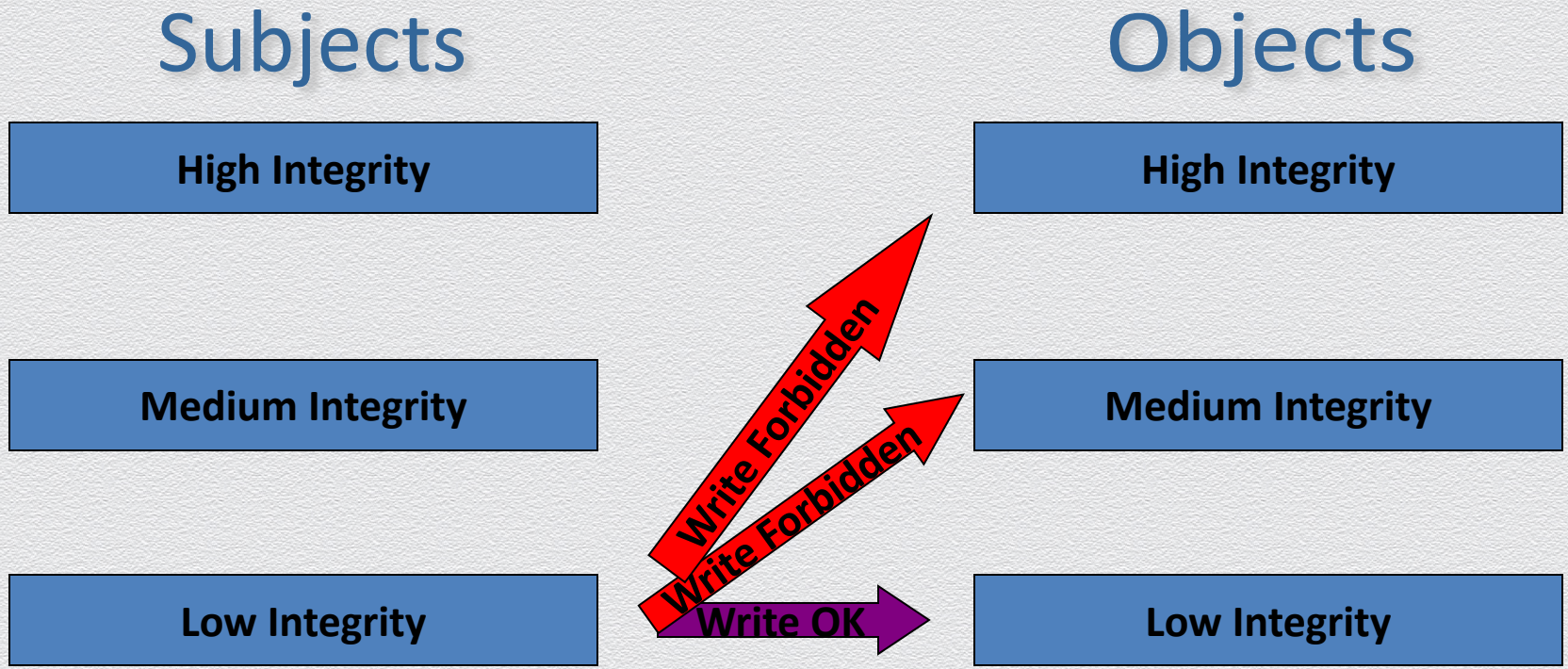
# Integrity control: Biba

- Designed to preserve integrity, not limit access
- Three fundamental concepts:
  - Simple Integrity Property – no read down
  - Star Integrity Property (*) – no write up
  - No execute up

# Integrity control: Biba



# Integrity control: Biba

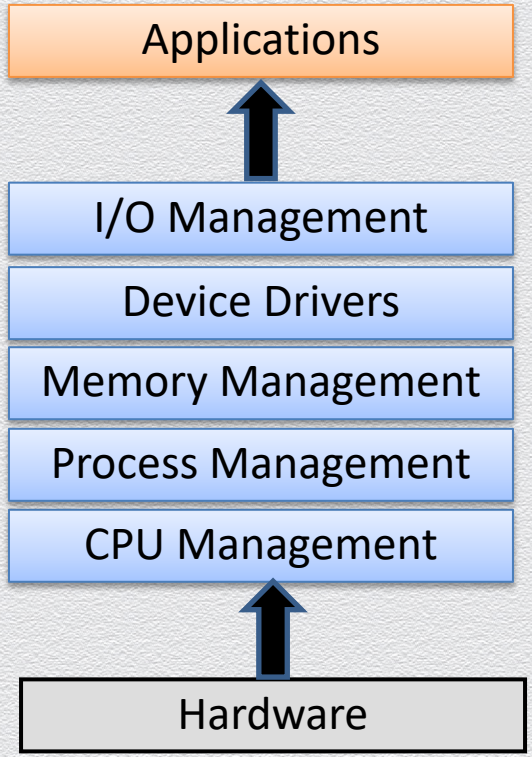


# 14.2 Operating system access control

# Operating System (OS) layers

Many layers of abstraction

- Kernel  core of the OS, controls hardware, resource access
  - Various subsystems
    - memory management, networking, storage, …
- Execution modes
  - user mode: access to resources mediated by the kernel
  - kernel mode: full and direct access to resources



# Processes

The kernel manages applications as processes (or threads)

Every process has

- Process ID (PID), virtual memory, effective user

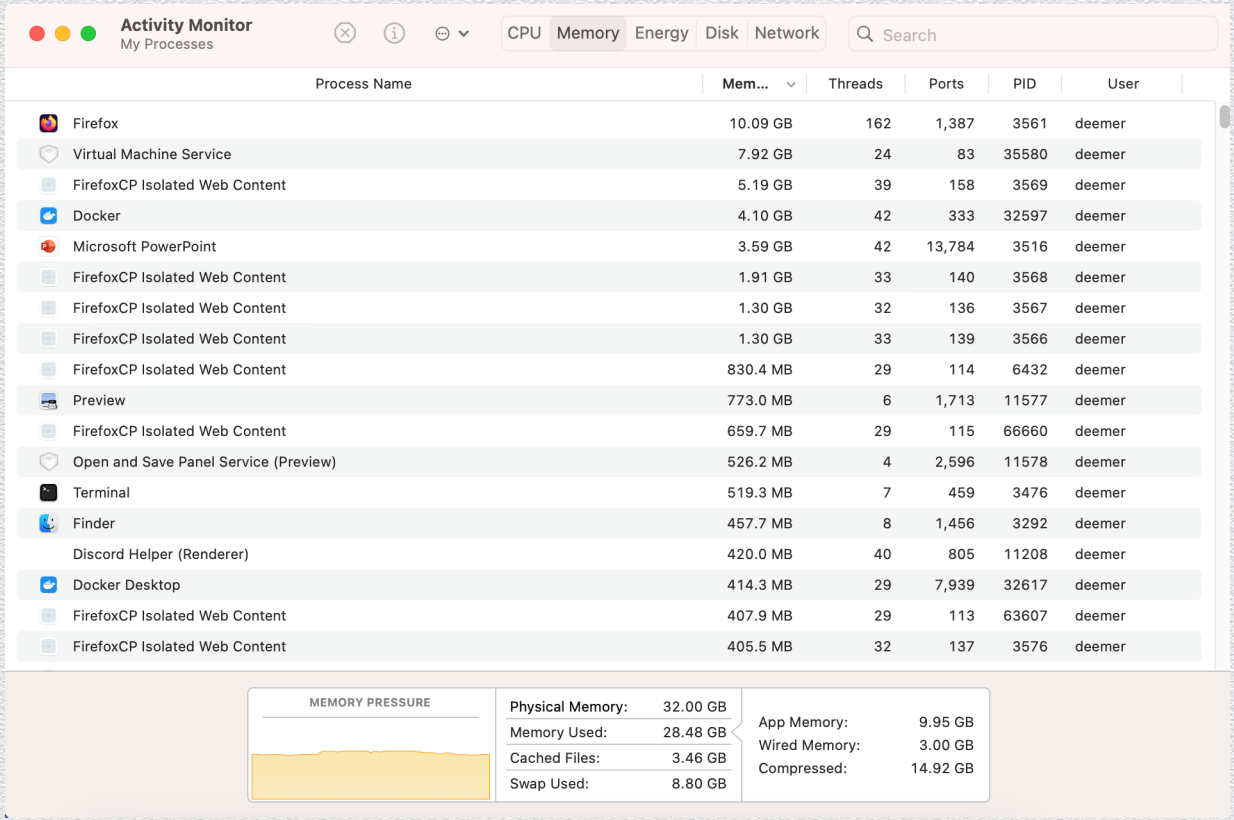
Kernel provides

- Separate address space from other process
- Time/resource sharing
- Access control

# Processes (cont.)

```
emplisi@ubuntu:~$ pstree
systemd─┬─ModemManager───2*[{ModemManager}]
        ├─NetworkManager───2*[{NetworkManager}]
        ├─VGAuthService
        ├─accounts-daemon───2*[{accounts-daemon}]
        ├─acpid
        ├─anacron───sh───run-parts───mlocate───flock───updated
        ├─avahi-daemon───avahi-daemon
        ├─bluetoothd
        ├─boltd───2*[{boltd}]
        ├─colord───2*[{colord}]
        ├─cron
        ├─cups-browsed───2*[{cups-browsed}]
        ├─cupsd
        ├─dbus-daemon
        ├─firefox─┬─3*[Web Content───18*[{Web Content}]]
        │         ├─Web Content───19*[{Web Content}]
        │         ├─WebExtensions───18*[{WebExtensions}]
        │         ├─file:// Content───18*[{file:// Content}]
        │         └─59*[{firefox}]
        ├─fwupd───4*[{fwupd}]
        ├─gdm3─┬─gdm-session-wor─┬─gdm-wayland-ses─┬─gnome-ses
```

# Processes (cont.)



# Processes (cont.)

- ps: displays snapshot of running processes
  - ps -ef : show all processes
  - ps -u <username>: show processes for a user
- top, htop: fancier list of processes
  - top -u <username>: filter by username
- kill  <pid>: terminates a process

```
metcalfe /u/lmeyerov % ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2005 ?        00:00:01 init [2]
root         2     1  0  2005 ?        00:00:00 [ksoftirqd/0]
root         3     1  0  2005 ?        00:00:00 [events/0]
root         4     3  0  2005 ?        00:00:00 [khelper]
root        31     3  0  2005 ?        00:00:00 [kblockd/0]
root       104     3  0  2005 ?        00:00:01 [pdflush]
root       105     3  0  2005 ?        00:00:00 [pdflush]
root       107     3  0  2005 ?        00:00:00 [aio/0]
root       106     1  0  2005 ?        00:00:03 [kswapd0]
root       694     1  0  2005 ?        00:00:00 [kseriod]
root       745     3  0  2005 ?        00:00:00 [ata/0]
root       750     1  0  2005 ?        00:00:00 [scsi_eh_2]
root       751     1  0  2005 ?        00:00:00 [scsi_eh_3]
root       769     1  0  2005 ?        00:00:02 [kjournald]
root      1157     1  0  2005 ?        00:00:00 [khubd]
root      1263     1  0  2005 ?        00:00:00 [kjournald]
root      1264     1  0  2005 ?        00:00:00 [kjournald]
root      1265     1  0  2005 ?        00:00:00 [kjournald]
root      1266     1  0  2005 ?        00:00:06 [kjournald]
root      1521     1  0  2005 ?        00:00:00 [khpsbpkt]
root      1584     1  0  2005 ?        00:00:00 [knodemgrd_0]
root      2813     1  0  2005 ?        00:00:00 dhclient -e -pf /var/run/dhclien
```

```
top - 14:23:25 up 41 days, 20:45, 10 users,  load average: 0,02, 0,12, 0,19
Tasks: 142 total,   1 running, 140 sleeping,   0 stopped,   1 zombie
Cpu(s):  3,7% us,  0,7% sy,  0,0% ni, 95,7% id,  0,0% wa,  0,0% hi,  0,0% si
Mem:   1034436k total,   910324k used,   124112k free,   183132k buffers
Swap:  1048816k total,     4892k used,  1043924k free,   222260k cached
Which user (blank for all):
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4104 root       5 -10  180m  50m 6724 S  1,0  5,0   2:42,53 XFree86
13091 lmeyerov  15   0  132m  72m  26m S  1,0  7,2  17:22,59 mozilla-bin
21114 lmeyerov  15   0 36876  17m  13m S  1,0  1,7   0:01,38 ksnapshot
24555 lmeyerov  15   0 21128 7508 5460 S  0,7  0,7   0:00,79 artsd
21270 lmeyerov  17   0  2036 1116  860 R  0,7  0,1   0:00,02 top
    1 root      16   0  1504  528  468 S  0,0  0,1   0:01,32 init
    2 root      35  19     0    0    0 S  0,0  0,0   0:00,28 ksoftirqd/0
    3 root       5 -10     0    0    0 S  0,0  0,0   0:00,85 events/0
    4 root       6 -10     0    0    0 S  0,0  0,0   0:00,00 khelper
   31 root       5 -10     0    0    0 S  0,0  0,0   0:00,81 kblockd/0
  104 root      15   0     0    0    0 S  0,0  0,0   0:01,81 pdflush
  105 root      15   0     0    0    0 S  0,0  0,0   0:00,32 pdflush
  107 root      15 -10     0    0    0 S  0,0  0,0   0:00,00 aio/0
  106 root      15   0     0    0    0 S  0,0  0,0   0:03,65 kswapd0
  694 root      25   0     0    0    0 S  0,0  0,0   0:00,00 kseriod
  745 root       6 -10     0    0    0 S  0,0  0,0   0:00,00 ata/0
  750 root      18   0     0    0    0 S  0,0  0,0   0:00,00 scsi_eh_2
```

# Processes management

Each process has a context, which includes

- the user
- parent process
- address space

Kernel enforces policies to decide which resources each processes can use

# System calls (syscalls)

- Primary way processes interact with kernel
- OS provides a “library” of syscalls for nearly all OS functions
  - Files:  read, write, open, close, chmod, …
  - Process management:  fork, clone, kill, …
  - Networking:  socket, bind, connect

On syscall, process “yields” to kernel, executes in privileged kernel mode

# System services (daemons)

Background process that performs common tasks

- Started at boot time
- Could run with higher permissions than users

Typical services

- Remote SSH connections
- Web servers
- Logging

# Identification and Authentication (recap)

- A subject should provide a unique identifier
- Authentication is the act of confirming the truth of an attribute of a datum or entity
- There are three authentication factors
  - Knowledge: Something you know
  - Ownership: Something you have
  - Inherence: Something you are

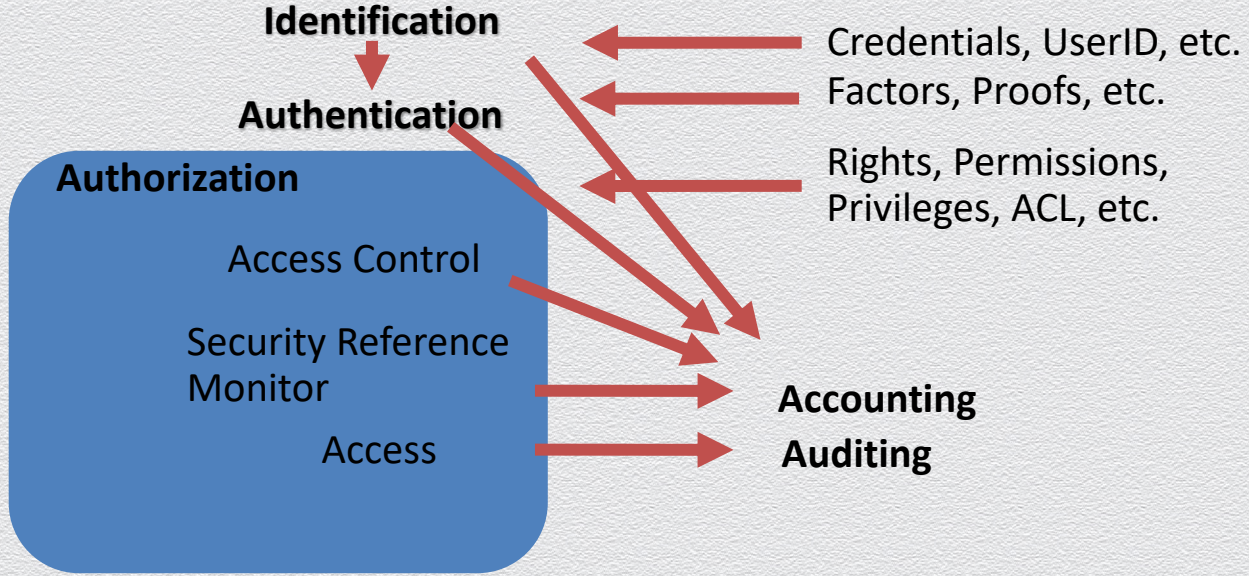
# Authorization

- Once a  subject is Authenticated, access should be authorized
- Authorization is the function of specifying access rights to resources (access control)
- More formally, "to authorize" is to define access policy: permissions, rights, etc.

# AAA

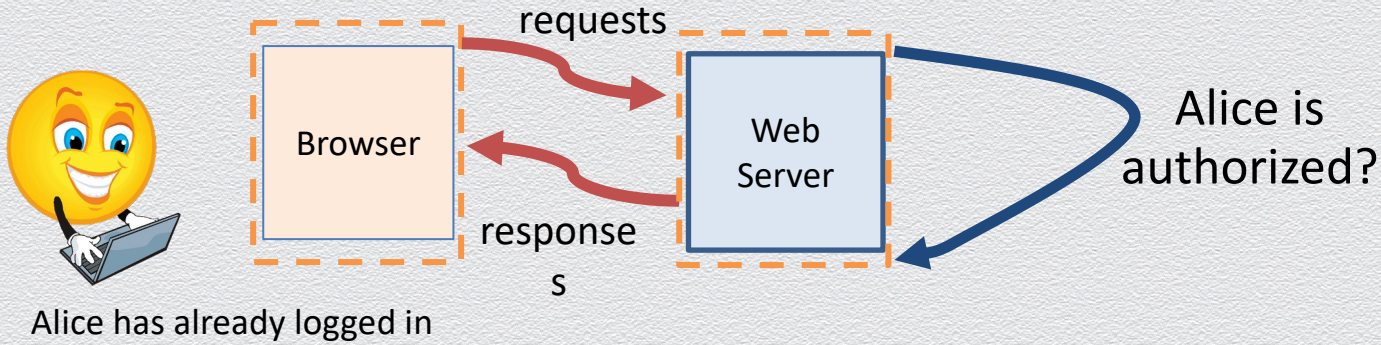
Identification, **Authentication**, **Authorization**, **Accounting**, Auditing

- AAA Working Group, IETF



# Authorization on the Web

- Alice logs in (i.e. authenticates); the web server is now aware of who is logged in
- Alice attempts to access a course
- The application checks to see if Alice has the authorization for the course…
  - If so, Alices receives the requested information; If not, Alice has a denied access response
- Authorization could be just for reading or writing or execute  (more in the future lectures)



# AAA: Authorization

Authorization:  how to specify access rights to resources

- To authorize => to define access policy

# Users

Each process is associated with a user

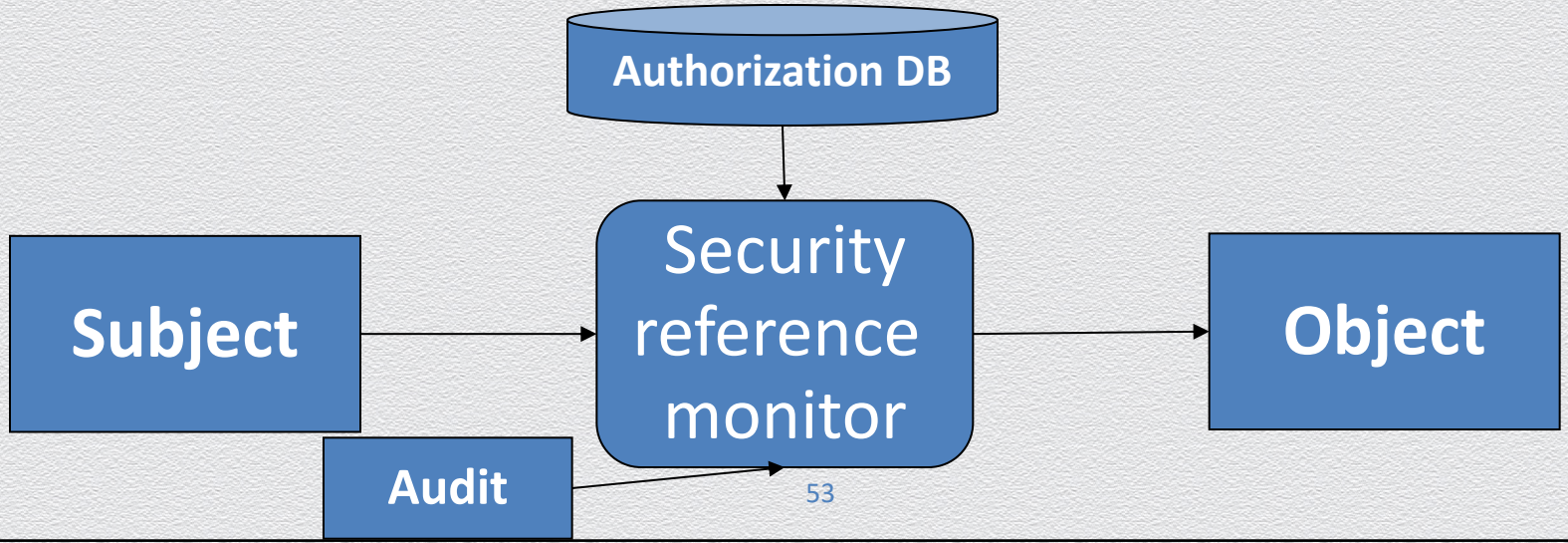
Specific users can have more privileges than regular users

- Install or remove programs
- Change rights of other users
- Modify the configuration of the system

Unix:  root is a “super-user” with no restrictions

# Security Reference Monitor (SRM)

- Checks for proper authorization before granting access to objects
- Object manager asks SRM if a Subject has the proper rights to execute a certain type of action on an Object
- Implements auditing functions to keep track of attempts to access an object



# Discretionary Access Control (DAC)

- Users can protect what they own
  - The owner may grant access to others
  - The owner may define the type of access (read/write/execute) given to others
- DAC is the standard model used in operating systems
- Mandatory Access Control (MAC)
  - Multiple levels of security for users and documents (i.e. confidential, restricted, secret, top secret)
  - A user can create documents with just his level of security

# General principles

- Files and folders are managed by the operating system
- Applications, including shells, access files through an API
- Access control entry (ACE)
  - Allow/deny a certain type of access to a file/folder by user/group
- Access control list (ACL)
  - Collection of ACEs for a file/folder
- A file handle provides an opaque identifier for a file/folder
- File operations
  - Open file: returns file handle
  - Read/write/execute file
  - Close file: invalidates file handle
- Hierarchical file organization
  - Tree (Windows)
  - DAG (Linux)

# AC entries and lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a sorted list of zero or more Access Control Entries (ACEs)
- An ACE refers specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group
- Examples of ACEs for folder “Bob’s CS1660 Grades”
  - Bob; Read; Allow
  - TAs; Read; Allow
  - Nikos; Read, Write; Allow
  - Bob; Write; Deny
  - TAs; Write; Allow

# Closed vs. Open Policy

## Closed policy

Also called “default secure”

- Give Nikos read access to “foo”
- Give Bob r/w access to “bar”
- Nikos: I would like to read “foo”
  - Access allowed
- Nikos: I would like to read “bar”
  - Access denied

## Open Policy

- Deny Nikos read access to “foo”
- Deny Bob r/w access to “bar”
- Nikos: I would like to read “foo”
  - Access denied
- Nikos: I would like to read “bar”
  - Access allowed

# Closed Policy with negative authorization & deny priority

Give Nikos r/w access to “bar”

Deny Nikos write access to “bar”

Nikos: I would like to read “bar”

Access allowed

Nikos: I would like to write “bar”

Access denied

Policy is used by Windows to manage access control to the file system

# Role-Based access control

- Within an organization roles are created for various job functions
- The permissions to perform certain operations are assigned to specific roles
- Users are assigned particular role, with which they acquire the computer authorizations
- Users are not assigned permissions directly, but only acquire them through their role

# Question

An ACL with no entries on a file?

1. Access Allowed to all with Open Policy
   Access Allowed to all with Closed Policy
2. Access Denied to all with Open Policy
   Access Allowed to all with Closed Policy
3. Access Allowed to all with Open Policy
   Access Denied to all with Closed Policy
4. Access Denied to all Open Policy
   Access Denied to all Closed Policy
5. It is not possible to realize

# Answer

An ACL with no entries on a file?

1. Access Allowed to all with Open Policy
   Access Allowed to all with Closed Policy
2. Access Denied to all with Open Policy
   Access Allowed to all with Closed Policy
3. Access Allowed to all with Open Policy
   Access Denied to all with Closed Policy
4. Access Denied to all Open Policy
   Access Denied to all Closed Policy
5. It is not possible to realize

# 14.3 File-system access control

# Linux Vs. Windows

- Linux
  - Allow-only ACEs
  - Access to file depends on ACL of file and of all its ancestor folders
  - Start at root of file system
  - Traverse path of folders
  - Each folder must have execute (cd) permission
  - Different paths to same file not equivalent
  - File's ACL must allow requested access
- Windows
  - Allow and deny ACEs
  - By default, deny ACEs precede allow ones
  - Access to file depends only on file's ACL
  - ACLs of ancestors ignored when access is requested
  - Permissions set on a folder usually propagated to descendants (inheritance)
  - System keeps track of inherited ACE's

# Linux file AC

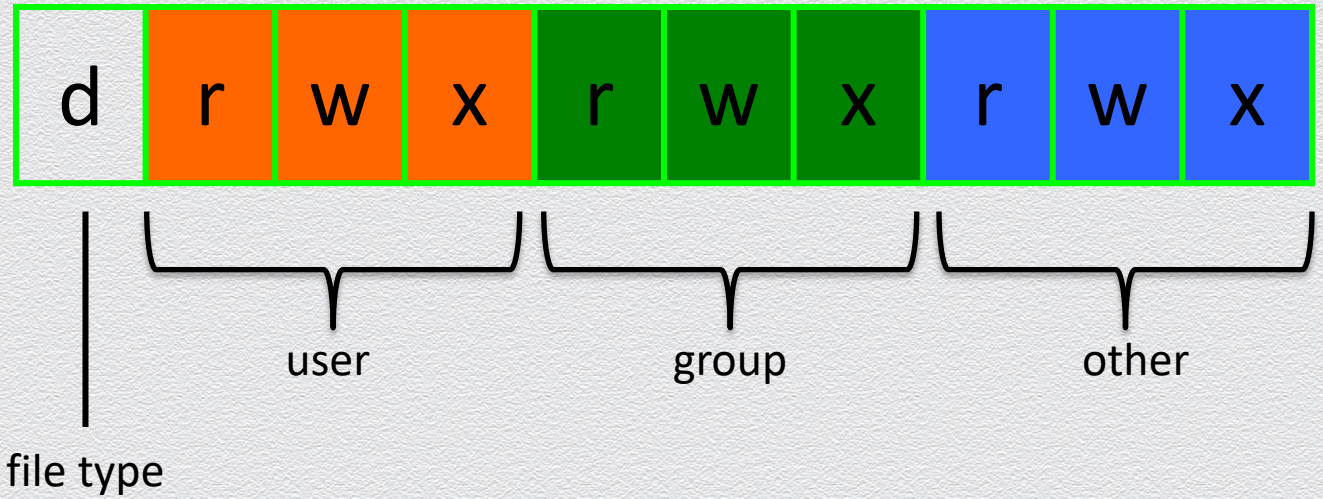
- File Access Control for:
  - Files
  - Directories
  - Therefore…
    - \dev\ : *devices*
    - \mnt\ : *mounted file systems*
    - What else? *Sockets, pipes, symbolic links*…

# Unix permissions

- Standard for all UNIXes
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation
- To see permissions, use ls –l

```
jk@sphere:~/test$ ls –l
total 0
-rw-r-----  1 jk ugrad 0 2005-10-13 07:18 file1
-rwxrwxrwx  1 jk ugrad 0 2005-10-13 07:18 file2
```

# Unix file types and basic permissions



# Permission examples (regular files)

| | |
|---|---|
| -rw-r—r-- | read/write for owner, read-only for everyone else |
| -rw-r----- | read/write for owner, read-only for group, forbidden to others |
| -rwx------ | read/write/execute for owner, forbidden to everyone else |
| -r--r--r-- | read-only to everyone, including owner |
| -rwxrwxrwx | read/write/execute to everyone |

# Permissions for directories

- Permissions bits interpreted differently for directories
- *Read* bit allows listing names of files in directory, but not their properties like size and permissions
- *Write* bit allows creating and deleting files within the directory
- *Execute* bit allows entering the directory and getting properties of files in the directory
- Lines for directories in ls –l output begin with d, as below:

```
jk@sphere:~/test$ ls –l

Total 4

drwxr-xr-x  2 jk ugrad 4096 2005-10-13 07:37 dir1

-rw-r--r--  1 jk ugrad    0 2005-10-13 07:18 file1
```

# Permission examples (directories)

| | |
|---|---|
| drwxr-xr-x | all can enter and list the directory, only owner can add/delete files |
| drwxrwx--- | full access to owner and group, forbidden to others |
| drwx--x--- | full access to owner, group can access known filenames in directory, forbidden to others |
| -rwxrwxrwx | full access to everyone |

# Question

Select the correct symbolic notation for a directory whose user class has full permissions, group class has read and execute permissions, and others class has only read permissions.

A. -rwxr-xr--

B. lr-xr-xr--

C. drwxr--r--

D. drwxr-xr--

# Answer

Select the correct symbolic notation for a directory whose user class has full permissions, group class has read and execute permissions, and others class has only read permissions.

A. -rwxr-xr--

C. drwxr--r--

B. lr-xr-xr--

D. drwxr-xr--

# Changing permissions

- Permissions are changed with chmod or through a GUI like KDE Konqueror
- Only the file owner or root can change permissions
- If a user owns a file, the user can use chgrp to set its group to any group of which the user is a member
- root can change file ownership with chown (and can optionally change group in the same command)
- chown, chmod, and chgrp can take the -R option to recur through subdirectories

# Changing permissions examples

| | |
|---|---|
| chown -R root dir1 | Changes ownership of dir1 and everything within it to root |
| chmod g+w,o-rwx file1 file2 | Adds group write permission to file1 and file2, denying all access to others |
| chmod -R g=rwX dir1 | Adds group read/write permission to dir1 and everything within it, and group execute permission on files or directories where someone has execute permission |
| chgrp testgrp file1 | Sets file1's group to testgrp, if the user is a member of that group |
| chmod u+s file1 | Sets the setuid bit on file1. (Doesn't change execute bit.) |

# Special permission bits

- Three other permission bits exist
  - Set-user-ID (“suid” or “setuid”) bit
  - Set-group-ID (“sgid” or “setgid”) bit
  - Sticky bit

# Set-user-ID

- Set-user-ID (“suid” or “setuid”) bit
  - On executable files, causes the program to run as file owner regardless of who runs it
  - Ignored for everything else
  - In 10-character display, replaces the 4th character (x or -) with s (or S if not also executable)

    -rwsr-xr-x: setuid, executable by all

    -rwxr-xr-x: executable by all, but not setuid

    -rwSr--r--: setuid, but not executable - not useful

# Setuid programs

- Unix processes have two user IDs:
  - real user ID: user launching the process
  - effective user ID: user whose privileges are granted to the process
- An executable file can have the set-user-ID property (setuid) enabled
- If a user A executes setuid file owned by B, then the effective user ID of the process is B and not A

# Setuid programs (cont.)

- System call setuid(uid) allows a process to change its effective user ID to uid
- Some programs that access system resources are owned by root and have the setuid bit set (setuid programs)
  - e.g., passwd and su
- Writing secure setuid programs is tricky because vulnerabilities may be exploited by malicious user actions

# Set-group-ID

- Set-group-ID (“sgid” or “setgid”) bit
- On executable files, causes the program to run with the file’s group, regardless of whether the user who runs it is in that group
- On directories, causes files created within the directory to have the same group as the directory, useful for directories shared by multiple users with different default groups
- Ignored for everything else
- In 10-character display, replaces 7$^{th}$ character (x or -) with s (or S if not also executable)

  -rwxr-sr-x: setgid file, executable by all

  drwxrwsr-x: setgid directory; files within will have group of directory

  -rw-r-Sr--: setgid file, but not executable - not useful

# Sticky bit

- On directories, prevents users from deleting or renaming files they do not own
- Ignored for everything else
- In 10-character display, replaces 10th character (x or -) with t (or T if not also executable)

drwxrwxrwt: sticky bit set, full access for everyone

drwxrwx--T: sticky bit set, full access by user/group

drwxr--r-T: sticky, full owner access, others can read *(useless)*

# Symbolic link

- In Unix, a symbolic link (aka symlink) is a file that points to (stores the path of) another file
- A process accessing a symbolic link is transparently redirected to accessing the destination of the symbolic link
- Symbolic links can be chained, but not to form a cycle

- ln -s really_long_directory/even_longer_file_name myfile

# Octal notation

- Standard syntax is nice for simple cases, but bad for complex changes
  - Alternative is octal notation, i.e., three or four digits from 0 to 7
- Digits from left (most significant) to right(least significant):
  *[special bits][user bits][group bits][other bits]*
- Special bit digit =
  (4 if setuid) + (2 if setgid) + (1 if sticky)
- All other digits =
  (4 if readable) + (2 if writable) + (1 if executable)

# Octal notation examples

| | |
|---|---|
| 644 or 0644 | read/write for owner, read-only for everyone else |
| 775 or 0775 | read/write/execute for owner and group, read/execute for others |
| 640 or 0640 | read/write for owner, read-only for group, forbidden to others |
| 2775 | same as 775, plus setgid (useful for directories) |
| 777 or 0777 | read/write/execute to everyone *(dangerous!)* |
| 1777 | same as 777, plus sticky bit |

# Root

- “root” account is a super-user account, like Administrator on Windows
- Multiple roots possible
- File permissions do not restrict root
- This is *dangerous*, but necessary, and OK with good practices

# Becoming root

- su
  - Changes home directory, PATH, and shell to that of root, but doesn't touch most of environment and doesn't run login scripts
- sudo <command>
  - Run just one command as root
- su [-] <user>
  - Become another non-root user
  - Root does not require to enter password

# The /tmp directory

- In Unix systems, directory /tmp is
  - Readable by any user
  - Writable by any user
  - Usually wiped on reboot
- Convenience
  - Place for temporary files used by applications
  - Files in /tmp are not subject to the user's space quota
- What could go wrong?
  - Sharing of resources may lead to vulnerabilities

# Limitations of Unix permissions

- Unix permissions are not perfect
  - Groups are restrictive
  - Limitations on file creation
- Linux optionally uses POSIX ACLs
  - Builds on top of traditional Unix permissions
  - Several users and groups can be named in ACLs, each with different permissions
  - Allows for finer-grained access control
- Each ACL is of the form *type*:[*name*]:*rwx*
  - Setuid, setgid, and sticky bits are outside the ACL system

# Gone for 10 seconds

- You leave your desk for 10 seconds without locking your machine
- The attacker sits at your desk and types:

  % cp /bin/sh /tmp

  % chmod 4777 /tmp/sh
- The first command makes a copy of shell sh
- The second command makes sh a
- What happens next?
- The attacker can run the copy of the shell with your privileges
- For example:
  - Can read your files
  - Can change your files

# Historical setuid Unix vulnerabilities: lpr

- Command lpr
  - running as root setuid
  - copied file to print, or symbolic link to it, to spool file named with 3-digit job number (e.g., print954.spool) in /tmp
  - Did not check if file already existed
  - Random sequence was predictable and repeated after 1,000 times
- How can we exploit this?

- Attack
  - A dangerous combination: setuid, /tmp, symlinks, …
  - Create new password file newpasswd
  - Print a very large file
  - lpr –s /etc/passwd
  - Print a small file 999 times
  - lpr newpasswd
  - The password file is overwritten with newpasswd

# Beyond setuid and files

- Writing setuid programs is tricky
  - Easy to inadvertently create security vulnerabilities
  - Unix variants have subtle different behaviors in setuid-related calls
- Access control to files is tricky
  - A user file can be accessed by any user process
  - Shared folders and predictable file names create security vulnerabilities
- Consider alternatives
  - Manage system resources via services
  - Use databases instead of files and shared folders
  - Use RPCs (including database queries) to request access to system resources