

CS-457

Information Security

Module 6 Access Control

a) Access Control at the Operating System

Based on Supplied Reading Material

The 3 Types of Access Control Policies

Discretionary Access Control (DAC)

- Based on Requestor's Identity + Access Rules
- Discretionary: Entity may grant others access to resources under its control

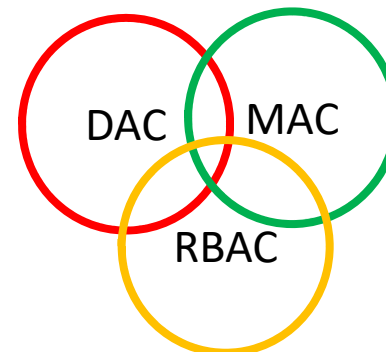
Mandatory Access Control (MAC)

- Based on Resource Security Labels + Requestor's Security Clearance
- Mandatory: Entity may not simply grant access to others

Role-Based Access Control (RBAC)

- Based on requestor's role + access rules
- which role has what access right

May be implemented as stand alone, or as a mix



Access Control: Subjects vs Objects

Subjects (Requesters)

- Process (on behalf of some user)
- Device

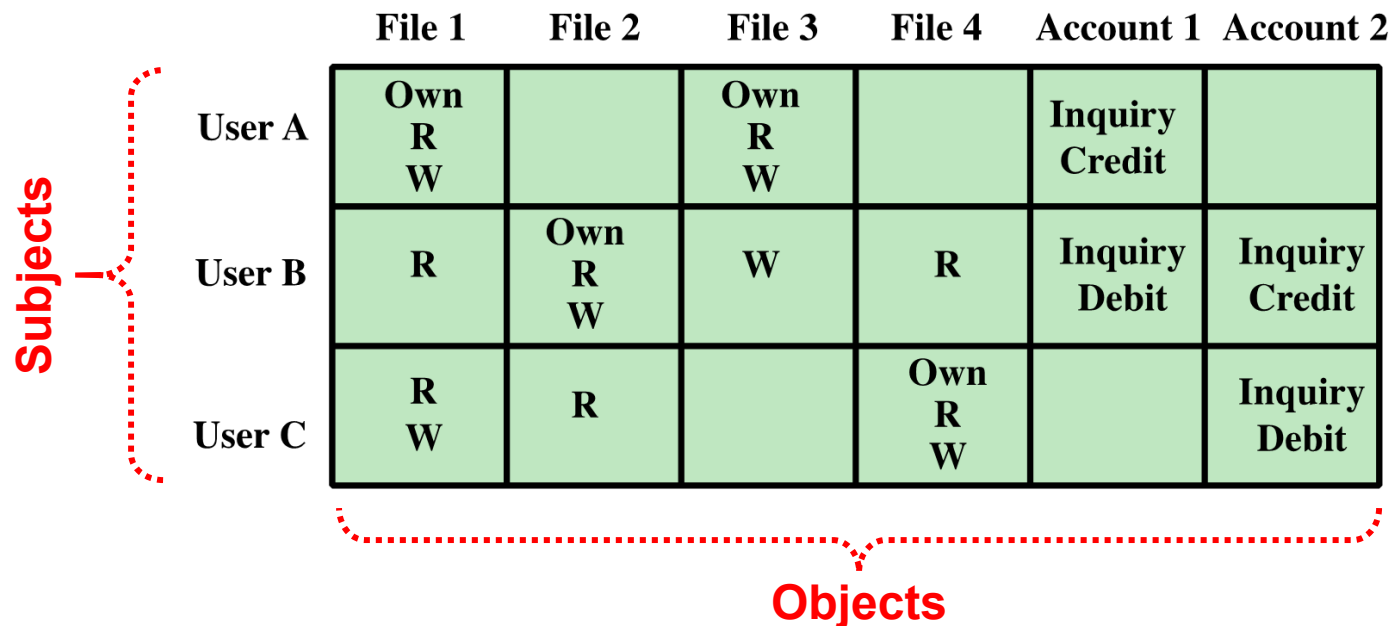
Objects (Resources being accessed)

- Data
- Memory
- CPU time
- Other Subjects treated as objects

The Concept of an Access Control Matrix

- Each entry specifies the set of access “*rights*” a subject has on some object, e.g. a file

$$A[i, j] = \{ right_1, right_2, \dots, right_n \}$$



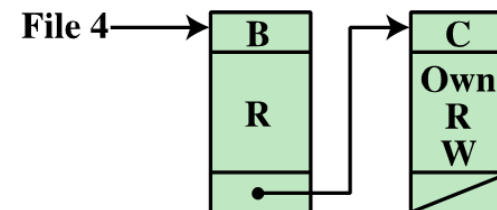
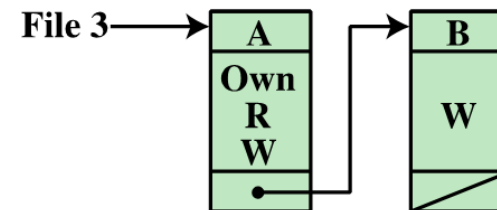
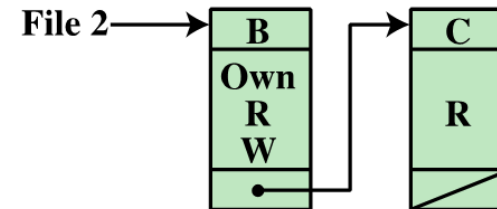
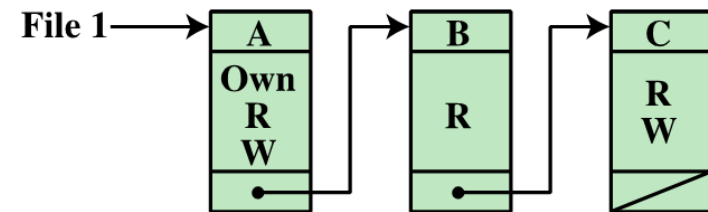
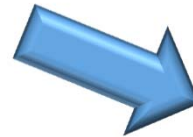
The diagram illustrates an Access Control Matrix. A red bracket on the left labeled "Subjects" groups the rows, which are labeled "User A", "User B", and "User C". A red bracket at the bottom labeled "Objects" groups the columns, which are labeled "File 1", "File 2", "File 3", "File 4", "Account 1", and "Account 2". The matrix cells contain specific rights: "Own", "R" (Read), "W" (Write), "Inquiry", "Credit", and "Debit".

	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit

- For a large number of subjects and objects, the matrix will be sparse, i.e. most entries will be empty.
- Instead of storing it as a matrix, use:
 - Access Control Lists
 - Capability Lists

The Access Control List: *Object-Focused*

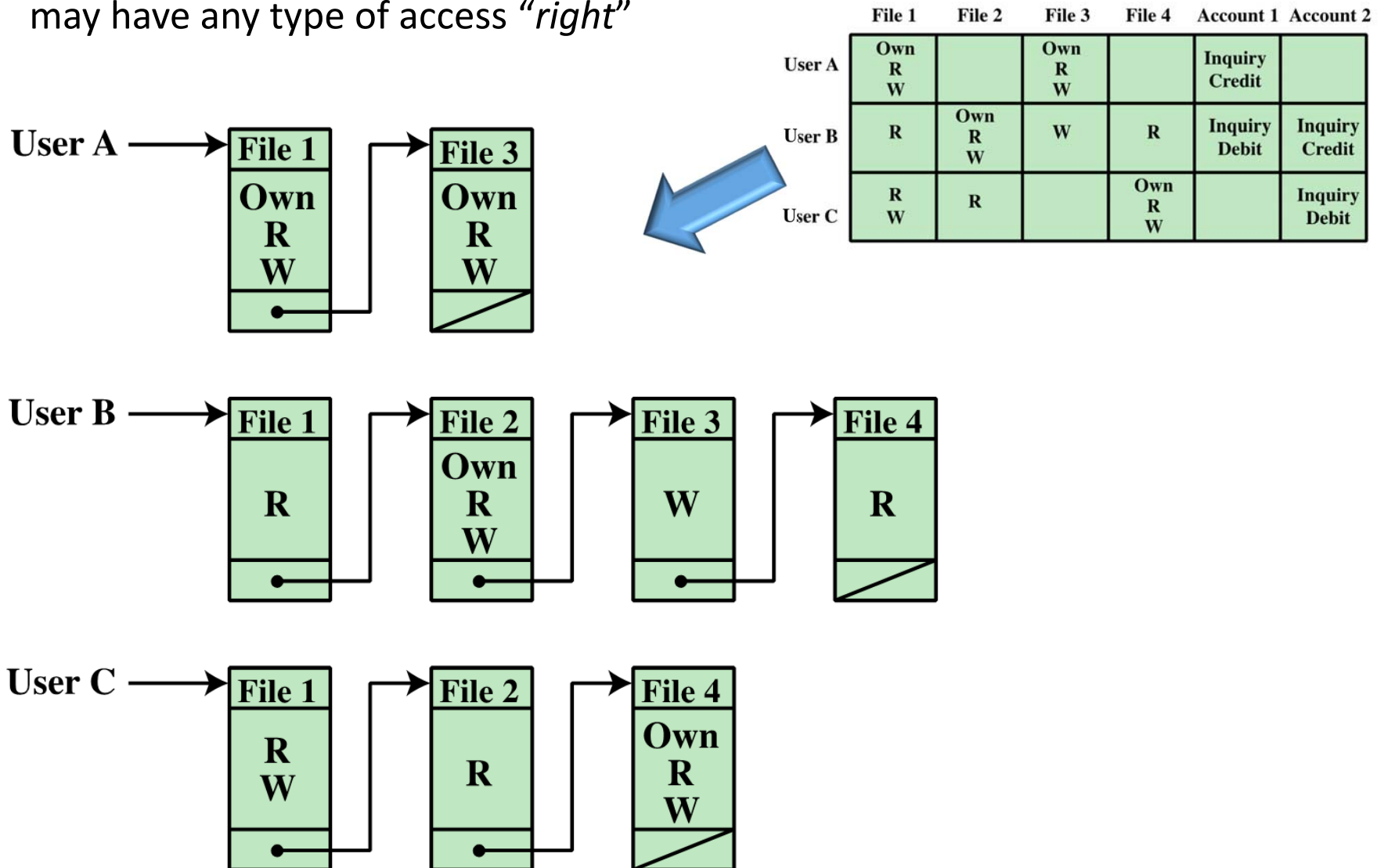
	File 1	File 2	File 3	File 4	Account 1	Account 2
User A	Own R W		Own R W		Inquiry Credit	
User B	R	Own R W	W	R	Inquiry Debit	Inquiry Credit
User C	R W	R		Own R W		Inquiry Debit



- For each object (e.g. a file), create a list of all subjects who have any type of access *“right”* on this object (file)

The Capability List: *Subject-Focused*

- For each subject (e.g. a user), create a list of all objects (e.g. files) on which it may have any type of access “right”



DAC: The *Extended* Access Control Matrix

- Subjects are also treated as objects:
 - There is a column for each subject just like any other object

		OBJECTS								
		subjects			files		processes		disk drives	
		S ₁	S ₂	S ₃	F ₁	F ₂	P ₁	P ₂	D ₁	D ₂
SUBJECTS	S ₁	control	owner	owner control	read*	read owner	wakeup	wakeup	seek	owner
	S ₂		control		write*	execute			owner	seek *
	S ₃			control		write	stop			

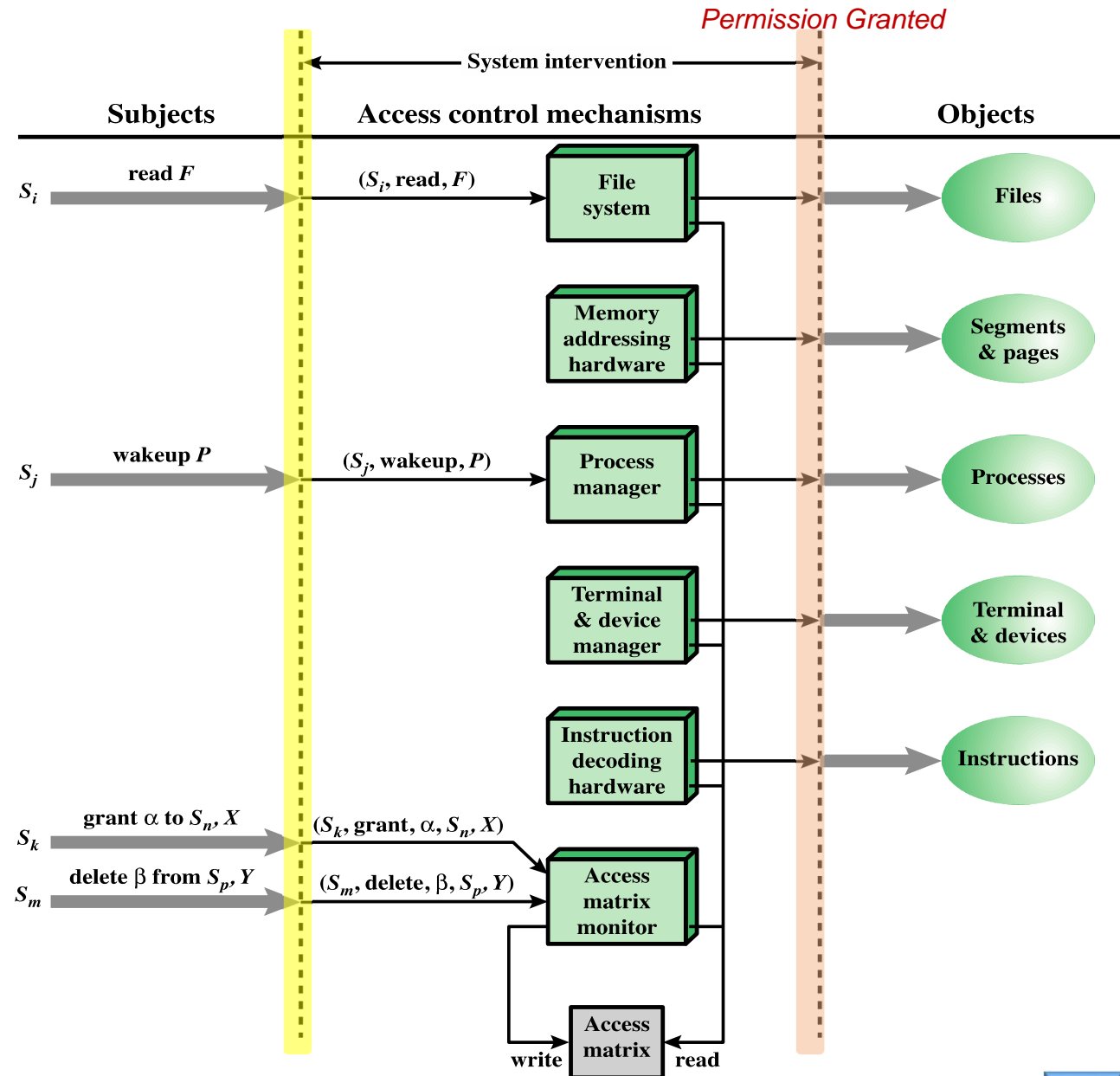
To authorize subject S_i attempting an access operation α (e.g. *read*) on X , it must first be true that:

$$\alpha \in A[S_i, X]$$

*right** means copy flag is set, so the subject can transfer that access right to other subjects

DAC: Organization of the Access Control Function in the OS

For each type of objects,
an interface module
(highlighted in yellow)
receives and reformats
the request, then
forwards it to the
appropriate controller
of that type of objects



DAC: commands (as issued by S_o) and rules

(You may print this as Cheat Sheet)

Rule	Command (by S_o)	Authorization	Operation
R1	transfer $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ to S, X	' α^* ' in $A[S_o, X]$	insert $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ in $A[S, X]$
R2	grant $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ to S, X	'owner' in $A[S_o, X]$	insert $\begin{Bmatrix} \alpha^* \\ \alpha \end{Bmatrix}$ in $A[S, X]$
R3	delete α from S, X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	delete α from $A[S, X]$
R4	$w \leftarrow$ read S, X Tell me what S can do with X	'control' in $A[S_o, S]$ or 'owner' in $A[S_o, X]$	copy $A[S, X]$ into w
R5	create object X	None	add column for X to A ; store 'owner' in $A[S_o, X]$
R6	destroy object X	'owner' in $A[S_o, X]$	delete column for X from A
R7	create subject S	none	add row for S to A ; execute create object S ; store 'control' in $A[S, S]$
R8	destroy subject S	'owner' in $A[S_o, S]$	delete row for S from A ; execute destroy object S

DAC Homework Exercise

Starting from this access control matrix of one row and one column

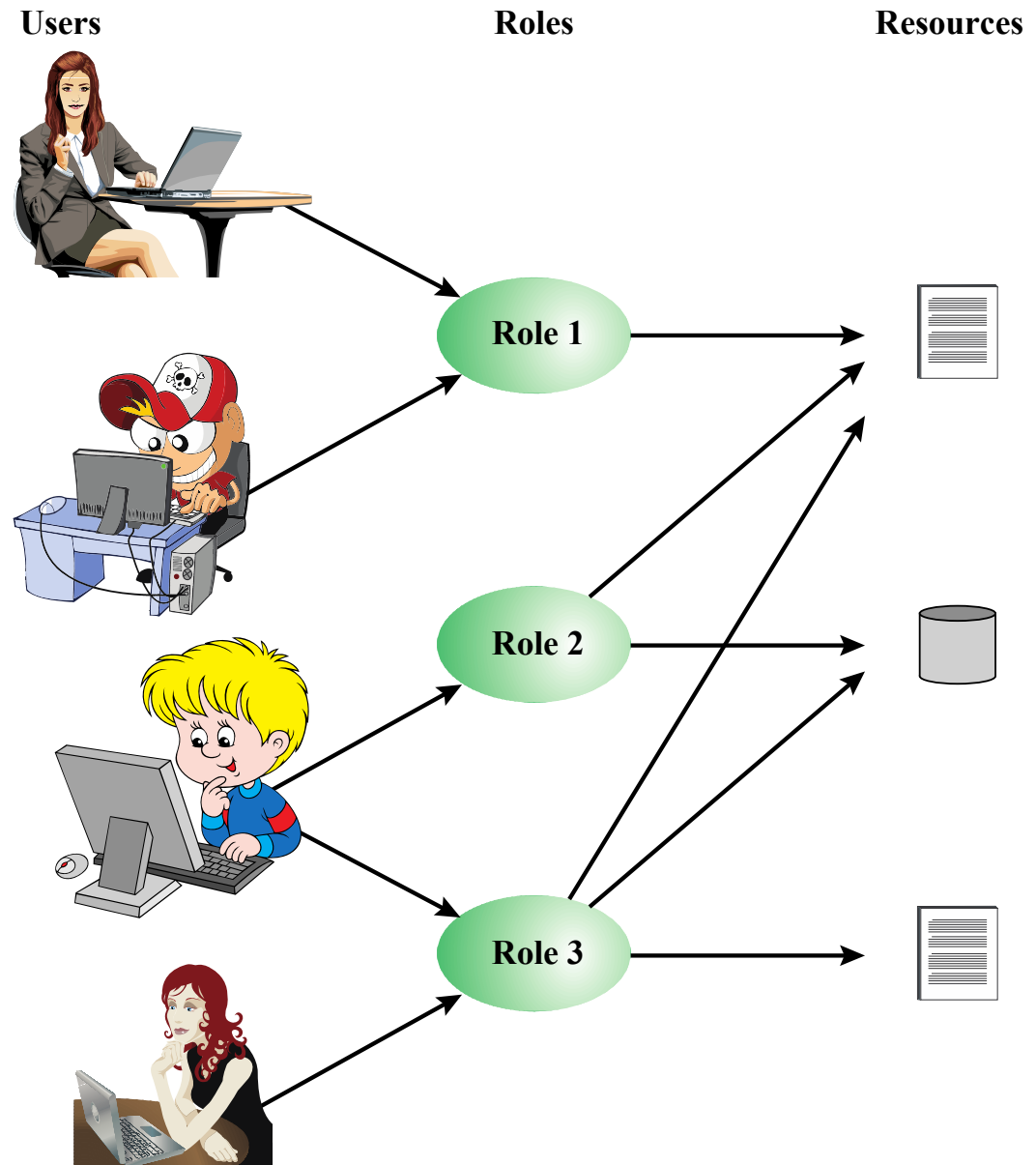
	<i>root</i>
<i>root</i>	Own, Control

Show whether the following sequence of commands will be authorized (explain why/why not). If authorized, show the effect on the access control matrix.

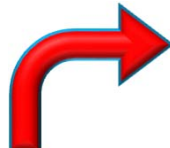
1. (root , create subject , Nancy)	9. (root , grant write* , to Basma , F1)
2. (root , create object , F1)	10. (Basma , transfer write , to Nancy , F1)
3. (root , read , F1)	11. (root , write , F1)
4. (root , grant read , to root , F1)	12. (root , delete read , from Basma , F1)
5. (root , read , F1)	13. (root , grant control , to Nancy , Basma)
6. (root , grant read , to Nancy , F1)	14. (Basma , read , F1)
7. (root , create subject , Basma)	15. (Nancy , delete write , from Basma , F1)
8. (Nancy , transfer read , to Basma , F1)	16. (Nancy , destroy subject , Basma)

Role-Based Access Control: RBAC

- ‘**Role**’ is a job function within some context (e.g. ‘manager’ in an organization)
- Access rights are assigned to roles (*many-to-many*)
- Users are given one or more roles (*many-to-many*)
 - Statically, i.e. permanently
 - Dynamically, i.e. when needed, then role is revoked



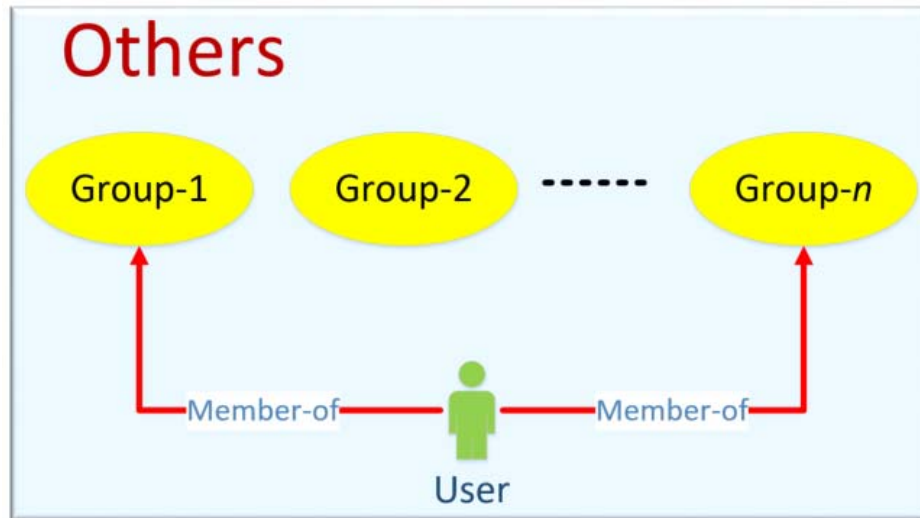
RBAC Example



	R ₁	R ₂	...	R _n
U ₁	×			
U ₂	×			
U ₃		×		×
U ₄				×
U ₅				×
U ₆				×
...				
U _m	×			

		OBJECTS								
		R ₁	R ₂	R _n	F ₁	F ₁	P ₁	P ₂	D ₁	D ₂
ROLES	R ₁	control	owner	owner control	read*	read owner	wakeup	wakeup	seek	owner
	R ₂		control		write*	execute			owner	seek *
	•									
	•									
	R _n			control		write	stop			

Traditional UNIX File Access Control



chown: changes owner
chgrp: changes group

Highest relevant is applied

Owner class
Group class
Other class

rw-	r--	---
-----	-----	-----

*9 (of the 12)
protection bits each
file has*

user: :rw-
group: :r--
other: :---

0640 in Octal

110

100

000

File vs. Directory Access Modes

File Access Modes	Directory Access Modes
<ul style="list-style-type: none">• Read Grants the capability to read, i.e., view the contents of the file.• Write Grants the capability to modify, or remove the content of the file.• Execute User with execute permissions can run a file as a program.	<ul style="list-style-type: none">• Read Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.• Write Access means that the user can add or delete files from the directory.• Execute A user must have execute access to the a directory in order to execute a program inside that directory.

`chmod`: changes protection bits

Using the `chmod` command

```
$chmod o+wx,u-x,g = rx testfile  
$ls -l testfile  
  
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

```
$ chmod 755 testfile  
$ls -l testfile  
  
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile  
$chmod 743 testfile  
$ls -l testfile  
  
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile  
$chmod 043 testfile  
$ls -l testfile  
  
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Traditional UNIX File Access Control: other protection bits

SetUID bit

- *executable files*:
 - temporarily apply rights of file's creators to current executing user, if needed.
- *directory*:
 - ignored

SetGIU bit

- *executable files*:
 - temporarily apply rights of file's group to current executing user, if needed.
- *directory*:
 - newly-created files will inherit group of this director

Sticky bit

- *files*
 - outdated use
- *directory* (shared)
 - for each inside file, only the owner can rename, delete, or move this file

Traditional UNIX File Access Control: *Super User*

- Unrestricted access to ALL files, system-wide.
- Carefully develop programs owned by “[superuser](#)”, especially if they have the [SetUID](#) bit on

Traditional Protection Bits (a.k.a. *minimal access control lists*) are adequate for small number of users and groups.

For larger systems, UNIX uses *extended Access Control Lists*
If interested, search for the many videos on YouTube on Unix ACLs