Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - **All deadlines moved to Monday May 1st at 11:59 pm**
    - But please don't wait to last minute!
  - Homework 11, 12, Bonus Homework
  - Lab 4 and Lab 5
  - Quiz 3 and Quiz 4
  - Project 4 **(no late deadline)**

Final Exam

- **Wednesday 4/26 8:00-9:50**
  - same classroom
  - coffee will be served!
- Same format as midterm
- **Non-cumulative**
- Study guide and practice test on Canvas
- **Review Session** during Finals' Week
  - Date and time TBD
  - recorded

Bonus Opportunities

- **Bonus Homework**
  - worth up to 1%
  - lowest two homework assignments still dropped
- **Post-Course Quiz on Canvas**
  - worth 1%
- bonus point for class when

  OMETs response rate >= 80%
  - Currently at 28%
  - Deadline is Sunday 4/23

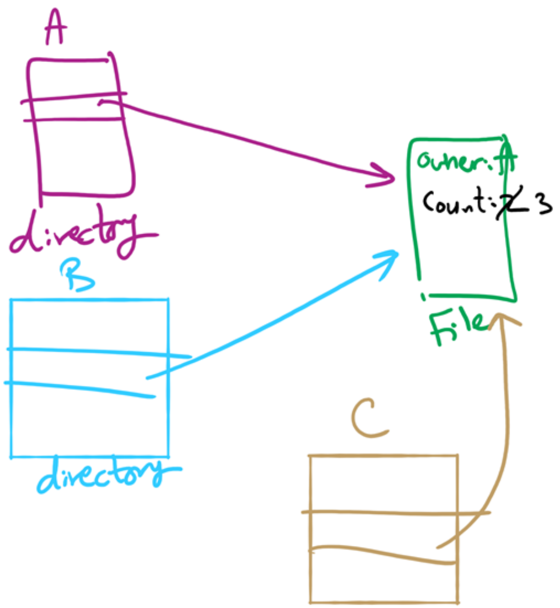
Previous Lecture …

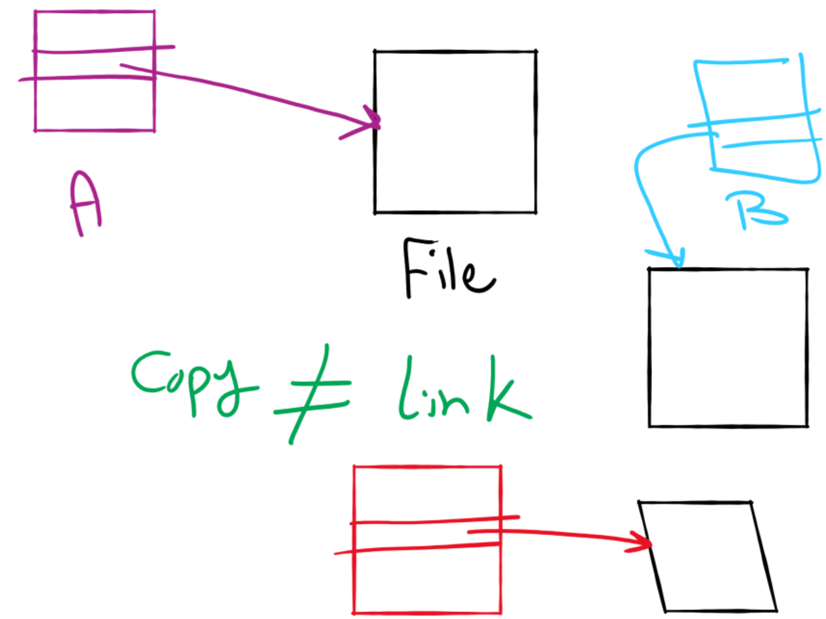
- How does a file system hide disk access delays?
- How do device drivers program I/O devices?
- Disk arm scheduling
  - FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK

Today …

- Miscellaneous issues in File Systems
- Protection in operating systems

Hard Linking vs. Copying





hard linking

- doesn't create a new i-node
- increments the **link count** inside original i-node
- **e.g., ln** command in Linux **without -s**

copying

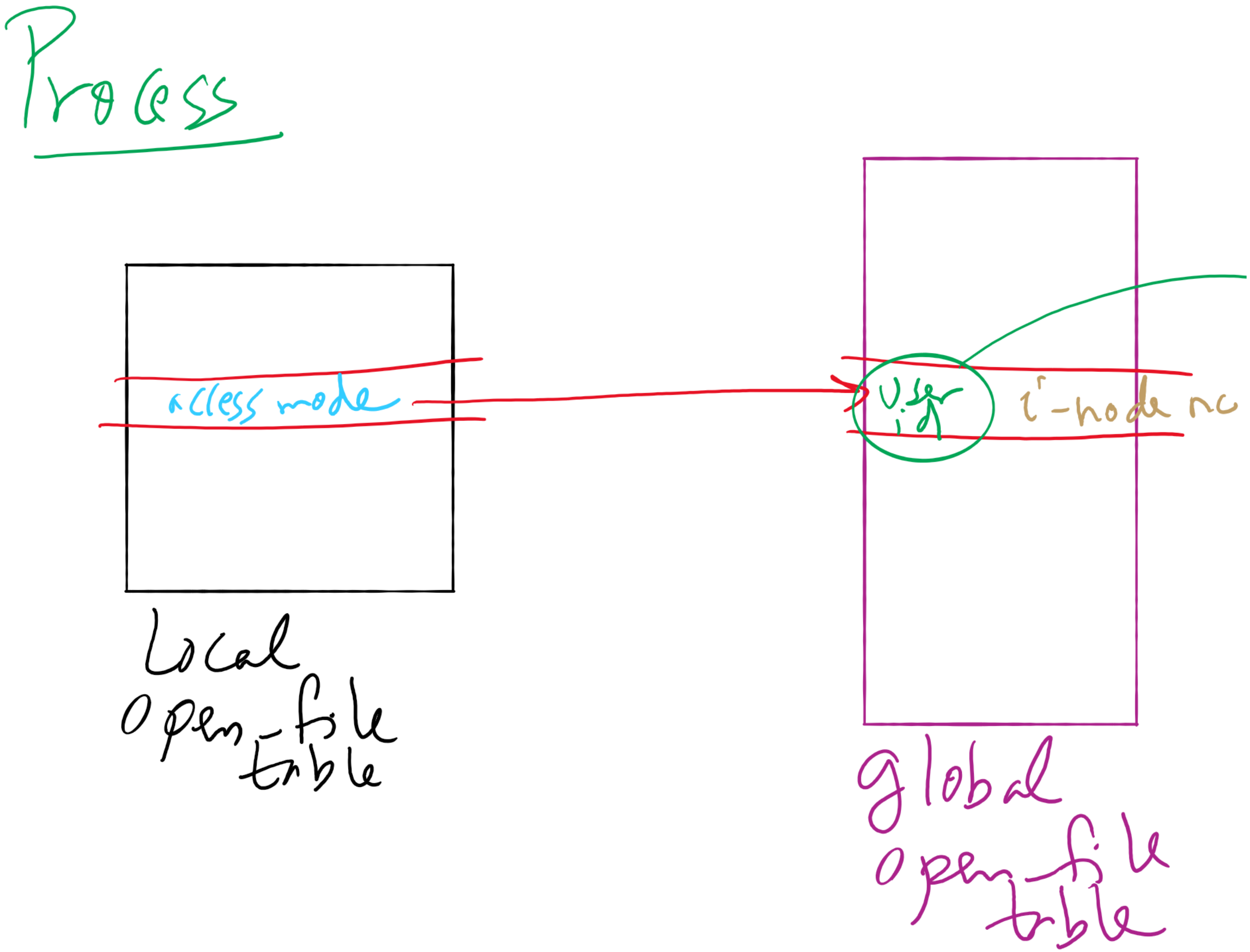
- creates a new i-node
- **e.g., cp** command in Linux

soft (symbolic linking)

creates a new i-node that contains **path** of original file

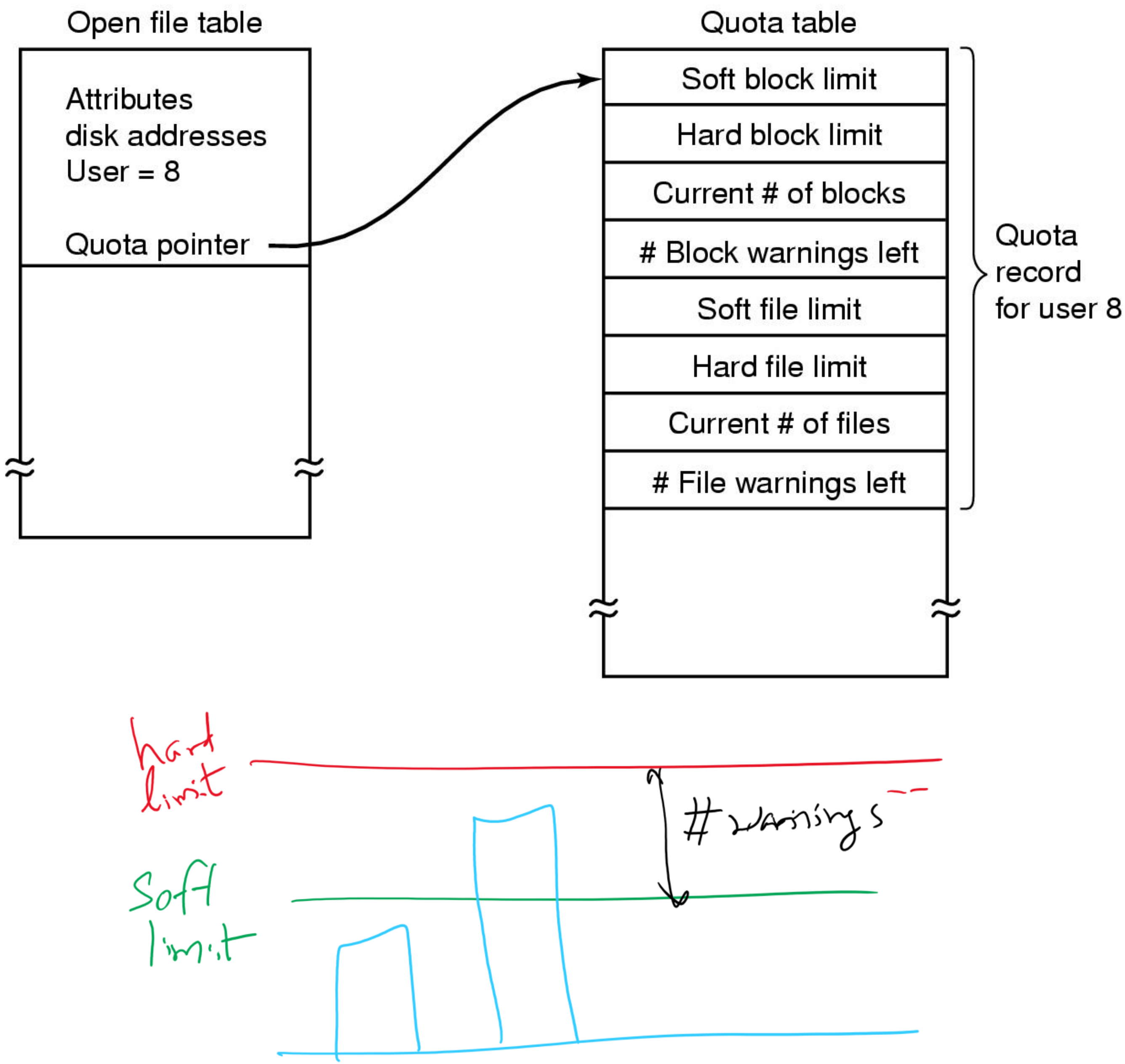
e.g., ln -s command in Linux

File-related kernel structures: open-file tables



per-process and global open-file tables

File-related kernel structures: quota table



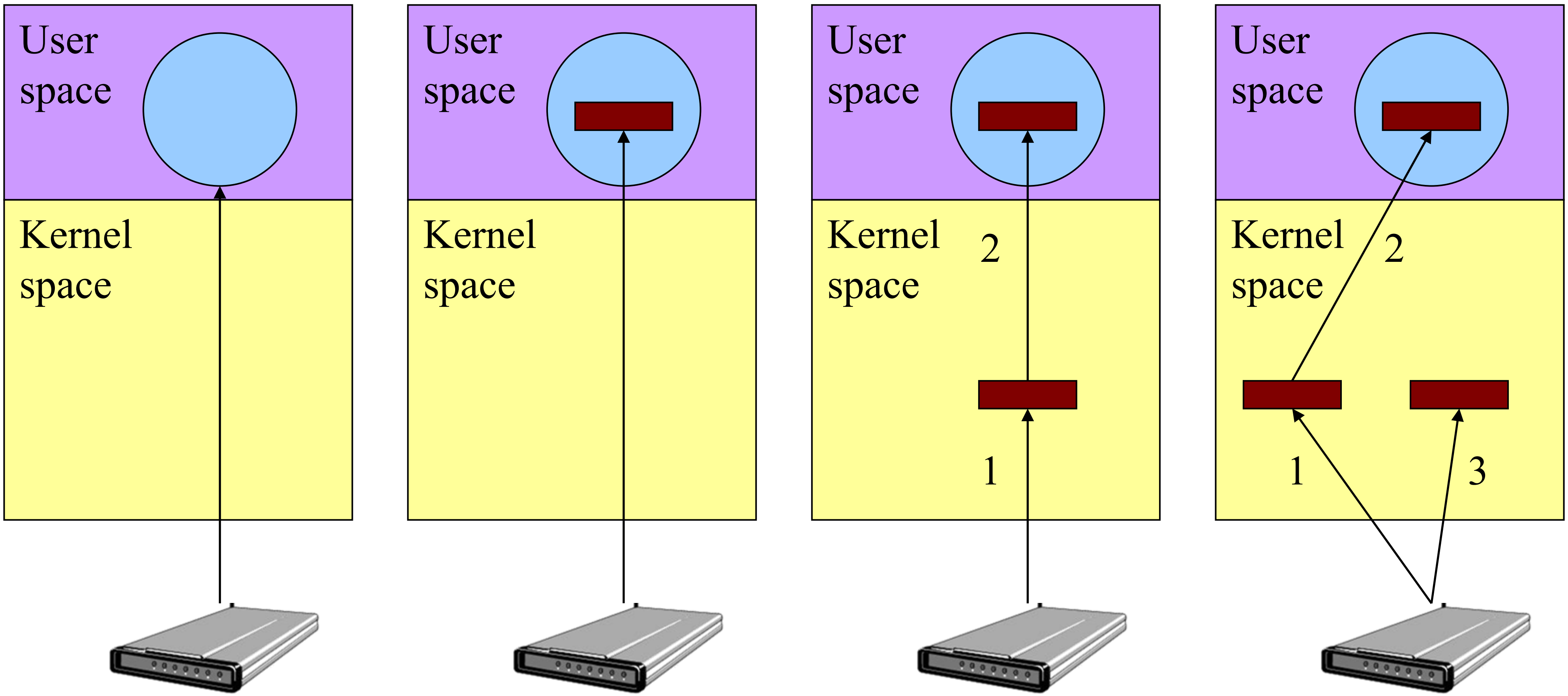
Journaling File System

- **Problem**: regular file system: changes to files and directories result in **multiple separate writes** to disk
    - e.g., deleting a file → three writes
    - **power failures** result in file system **inconsistency**
- **Solution**: Write the changes **twice**
    - first to an on-disk ***journal***
        - for efficiency, journal can be put on **SSD or NVRAM**
    - then **commit** changes to main part
    - atomic operations
- modified **data** may or may not be written to the journal
    - implications?
- Examples
    - Windows NTFS
    - Linux ext3, ext4

Journaling File System

- Interaction with disk arm scheduling?
  - may cause a different write order than wanted
  - **solution**: flush write cache at certain points (barriers)
    - ext3 and ext4 file systems
- Journaling vs. Log-structured file system
  - main file system itself is a journal
  - journaling is not needed in LFS

Buffering device input



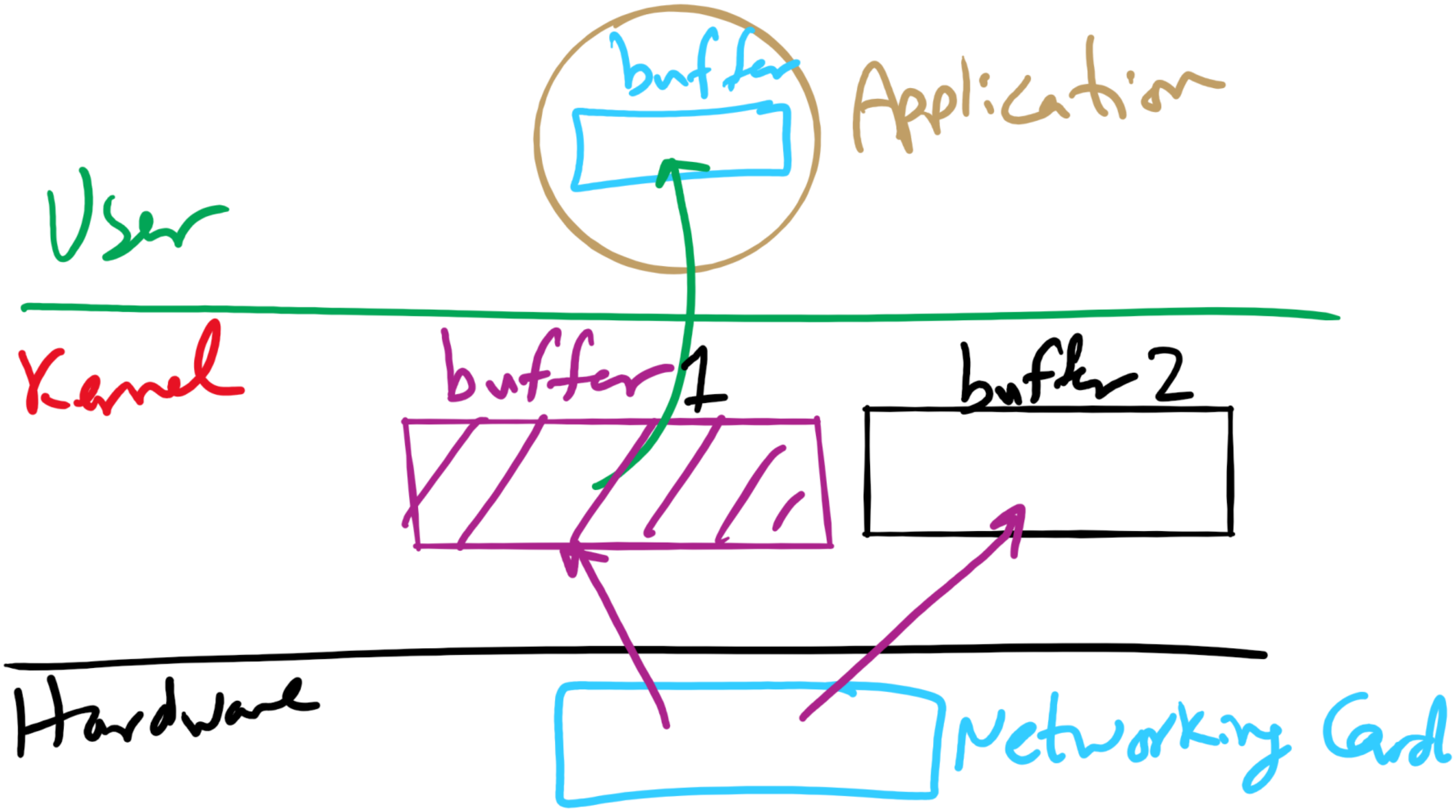
Unbuffered input

Buffering in user space

Buffer in kernel
Copy to user space

Double buffer in kernel

Double Buffering



Max Partition Size

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Max Partition Size

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

32-bit block no.

Max. Partition Size $= 2^{\text{(block no. size)}} * \text{block size}$

FAT-12 : $2^{12} * \frac{1}{2}\text{KB}$
$= 2^{12} * 2^{9} = 2^{21} = 2\text{MB}$

FAT-16 : $2^{16} * 2\text{KB}$
$= 2^{16} * 2^{11} = 2^{27} = 128\text{MB}$

FAT-32 : $2^{28} * 4\text{KB}$
$= 2^{28} * 2^{12} = 2^{40} = 1\text{TB}$

$2^{\text{sector no. size}} * 512 = 2^{32} * 512$
$= 2^{41} = 2\text{TB}$

Max Partition Size

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

32-bit block no.

$$\text{Max. Partition size} = 2^{(\text{block no. size})} \times \text{block size}$$

Max Partition Size

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

32-bit block no.

Max. Partition size $= 2^{(\text{block no. size})} * \text{block size}$

FAT-12 : $2^{12} * \frac{1}{2}$ KB

$= 2^{12} * 2^{9} = 2^{21} = 2$MB

Max Partition Size

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

FAT_16 : $2^{16} * 2\text{KB}$

$= 2^{16} * 2^{11} = 2^{27} = 128\text{MB}$

Max Partition Size

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

FAT-32 : $2^{28} * 4KB$

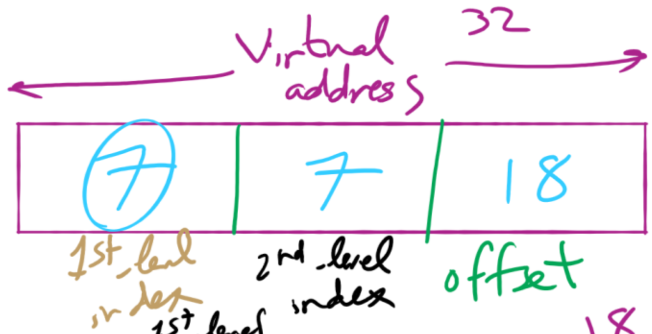
$= 2^{28} * 2^{12} = 2^{40} = 1TB$

sector no. size

$2 * 512 = 2^{32} * 512$

$= 2^{41} = 2TB$

HW 10: Q 2-4



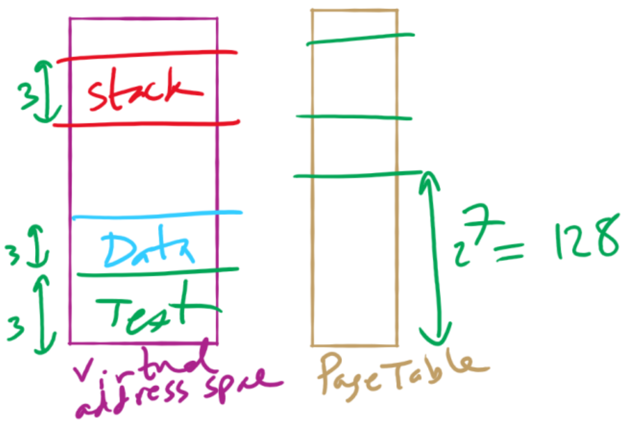
$\# PDEs = 2^{\text{1st\_level index size}}$

$= 2^7$

$2^8 KB = 2^{18}$ bytes

$\# PTEs$ in 2^{nd}_level Page Table $= 2^{\text{2nd\_level index size}}$

$= 2^7$



HW 10: Q 10-13

single-level Page Table

$EAT = h(a+m) + (1-h)(a+m+m)$

2-level Page Table

$EAT = h(a+m) + (1-h)(a+m+m+m)$

(PID, VPN)

(30, 1)

0x1200

Page No

offset

$4KB = 2^{12}$

⇒ offset : 12 bits

frame no. : 3

~~3~~ ~~1~~200

$\boxed{3200}$

0x ~~4~~ 500 (7)

frame no.

offset

PID

(10, ~~7~~) (7)

frame no. : 4

$\boxed{0x7500}$

Effective Disk Access Time

EDAT = h * hittime + (1-h) * misstime

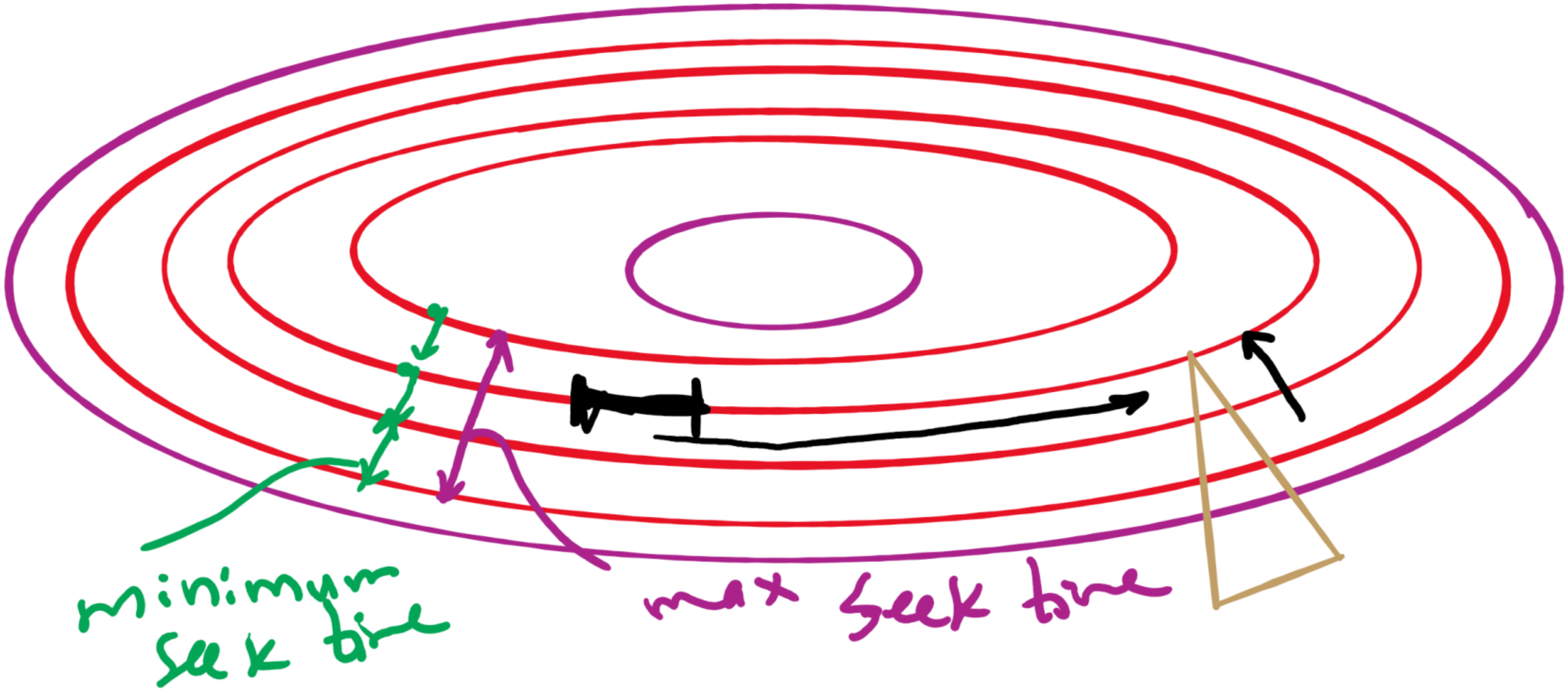
0.93

0.01

0.07

0.01+3*10

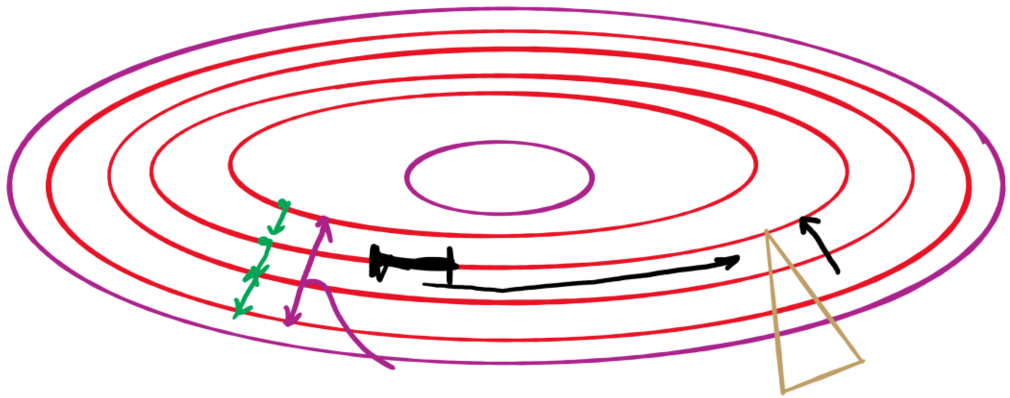
Minimum and Maximum Seek Time



Max seek time = min seek time * (# tracks-1)

#cylinders

Average Rotational Delay



$$\text{Average rotational delay} = \frac{\text{one full rotation time}}{2}$$

$$= \frac{1/\text{rotation speed}}{2}$$

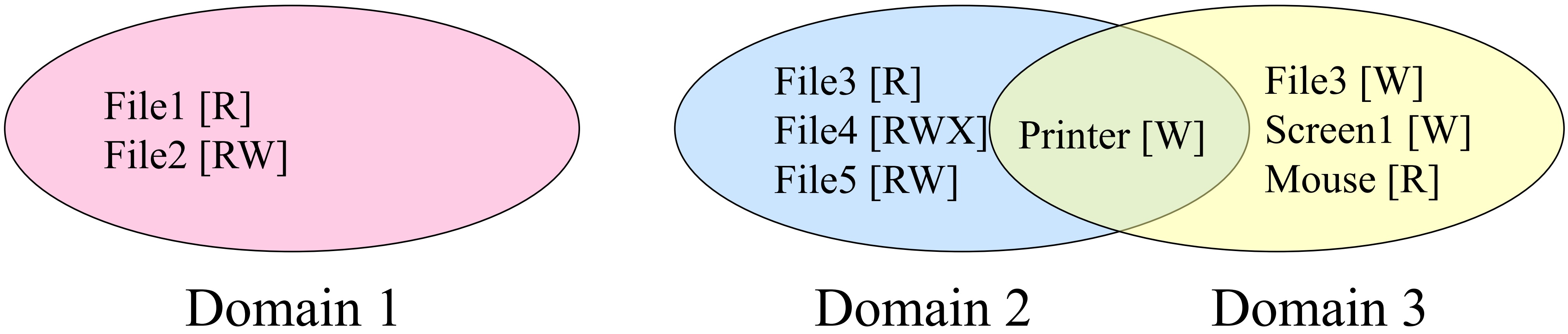
$$= \frac{(1/4800)*60*1000}{2} \text{ ms}$$

Problem of the Day: Protection

- Protection is about controlling access of programs, processes, or users to the system resources (e.g., memory pages, files, devices, CPUs)
  - How to decide who can access what?
  - Specifications must be
    - Correct
    - Efficient
    - Easy to use (or nobody will use them!)

Protection domains

- A process operates within a protection domain
  - resources accessible by the process
  - each domain lists objects with permitted operations
- Domains can share objects & permissions
  - Objects can have different permissions in different domains
  - There need be no overlap between object permissions in different domains
- How can this arrangement be specified more formally?



Protection matrix

	File1	File2	File3	File4	File5	Printer1	Camera
Domain1	Read	Read Write					
Domain2			Read	Read Write Execute	Read Write	Write	
Domain3			Write			Write	Read

- Each domain has a row in the matrix
- Each object (resource) has a column in the matrix
- Entry for <object, column> has the permissions
- Who’s allowed to modify the protection matrix?
    - What changes can they make?
- How is this implemented efficiently?

Domains as objects in the protection matrix

Domain	File1	File2	File3	File4	File5	Printer1	Camera	Dom1	Dom2	Dom3
1	Read	Read Write						Modify		
2			Read	Read Write Execute	Read Write	Write		Modify		
3			Write			Write	Read		Enter	

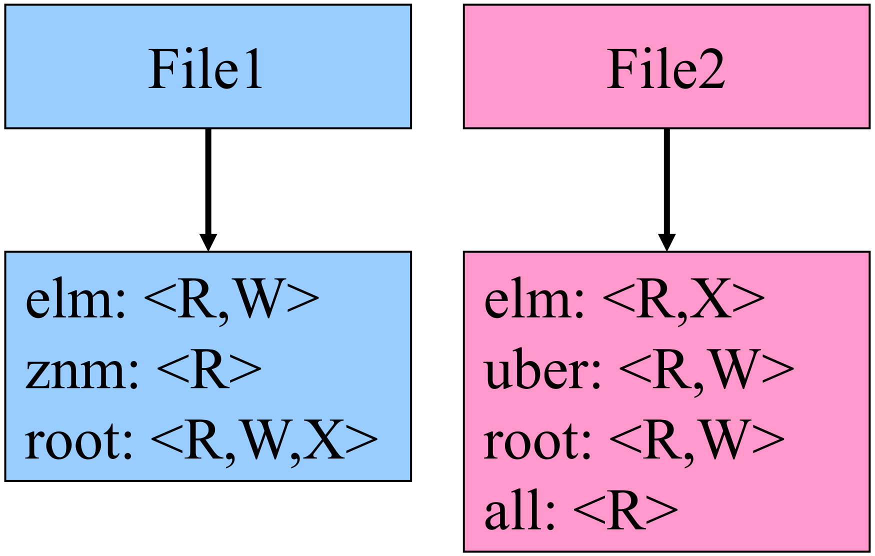
- Specify permitted operations on domains in the matrix
  - Domains may (or may not) be able to modify themselves
  - Domains can modify other domains
  - Some domain transfers (switching) permitted, others not
- Doing this allows flexibility in specifying domain permissions
  - Retains ability to restrict modification of domain policies

Representing the protection matrix

- Need to find an efficient representation of the protection matrix (also called the *access matrix*)
- Most entries in the matrix are empty!
- Compress the matrix by:
  - Associating permissions with each object: *access control list*
  - Associating permissions with each domain: *capabilities*
- How is this done, and what are the tradeoffs?

Access control lists (ACLs)

- Each object has a list attached to it
- List has
  - Protection domain (User name, Group of users, Other)
  - Access rights (Read, Write, Execute, Others)
- No entry for domain => no rights for that domain
- Operating system checks permissions when access is needed



Access control lists in the real world

- Unix file system
  - Access list for each file has exactly three domains on it
    - User (owner)
    - Group
    - Others
  - Rights include read, write, execute: interpreted differently for directories and files
- AFS
  - Access lists only apply to directories: files inherit rights from the directory they're in
  - Access list may have many entries on it with possible rights:
    - read, write, lock (for files in the directory)
    - lookup, insert, delete (for the directories themselves),
    - administer (ability to add or remove rights from the ACL)

ACL in UNIX

special flags			owner (user)			group of owner			others		
set user id	set group id	sticky bit	r	w	x	r	w	x	r	w	x
			1	1	0	0	0	1	0	0	1
				6			1			1	
1	1	1									
			1	1	1	1	0	1	1	0	1
				7			5			5	
			1	1	0	1	0	0	1	0	0
				6			4			4	

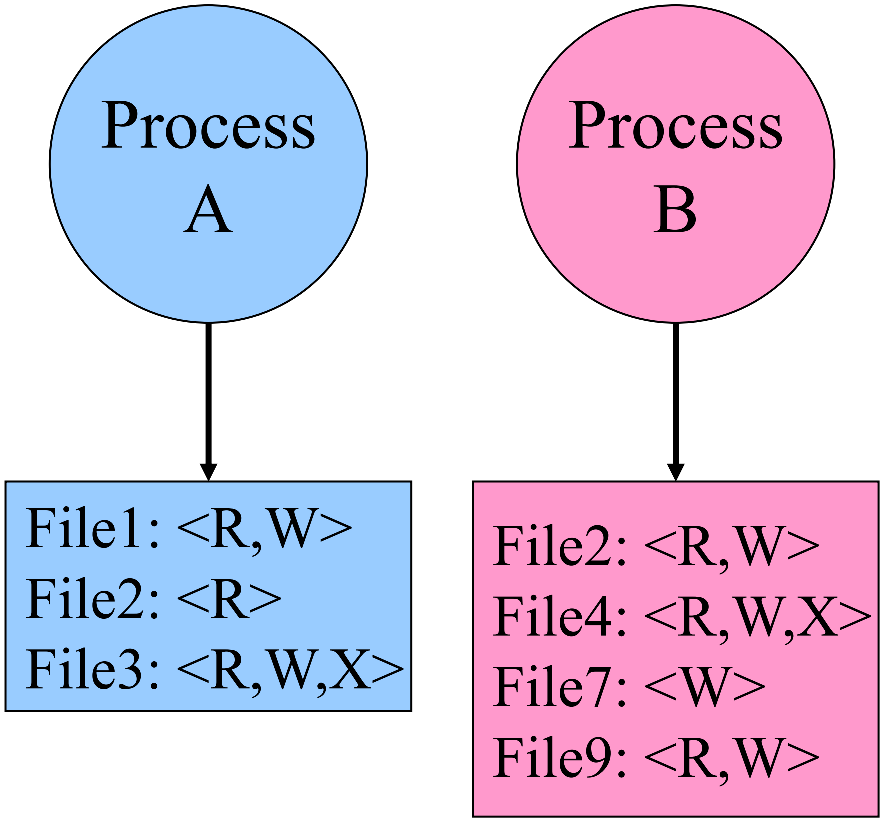
Domains as objects in the protection matrix

Domain	File1	File2	File3	File4	File5	Printer1	Camera	Dom1	Dom2	Dom3
1	Read	Read Write						Modify		
2			Read	Read Write Execute	Read Write	Write		Modify		
3			Write			Write	Read		Enter	

- Specify permitted operations on domains in the matrix
  - Domains may (or may not) be able to modify themselves
  - Domains can modify other domains
  - Some domain transfers (switching) permitted, others not
- Doing this allows flexibility in specifying domain permissions
  - Retains ability to restrict modification of domain policies

Capabilities

- Each process has a capability list
- List has one entry per object the process can access
  - Object name
  - Object permissions
- Objects not listed are not accessible
- How are these secured?
  - Kept in kernel
  - Cryptographically secured



Cryptographically protected capability

Server	Object	Rights	*H(Object,Rights,**Check**)*

- Rights include generic rights (read, write, execute) and
  - Copy capability
  - Copy object
  - Remove capability
  - Destroy object
- Server has a secret (*Check*) and uses it to verify capabilities presented to it
  - Alternatively, use public-key signature techniques

Protecting the access matrix: summary

- OS must ensure that the access matrix isn't modified (or even accessed) in an unauthorized way
- Access control lists
  - Reading or modifying the ACL is a system call
  - OS makes sure the desired operation is allowed
- Capability lists
  - Can be handled the same way as ACLs: reading and modification done by OS
  - Can be handed to processes and verified cryptographically later on
    - May be better for widely distributed systems where capabilities can't be centrally checked