Managing Files

Today's Plan

- **Controlling Files (Chapter 3, Elementary Information Security)**
  - File system and access rights
  - Executable files
  - Viruses and malware
- **Sharing Files (Chapters 3 and 4, Elementary Information Security)**
  - Policies for file protection
  - File Permission Flags
  - Access Control Lists
- DEMO/Intro to Python
- Lab 4

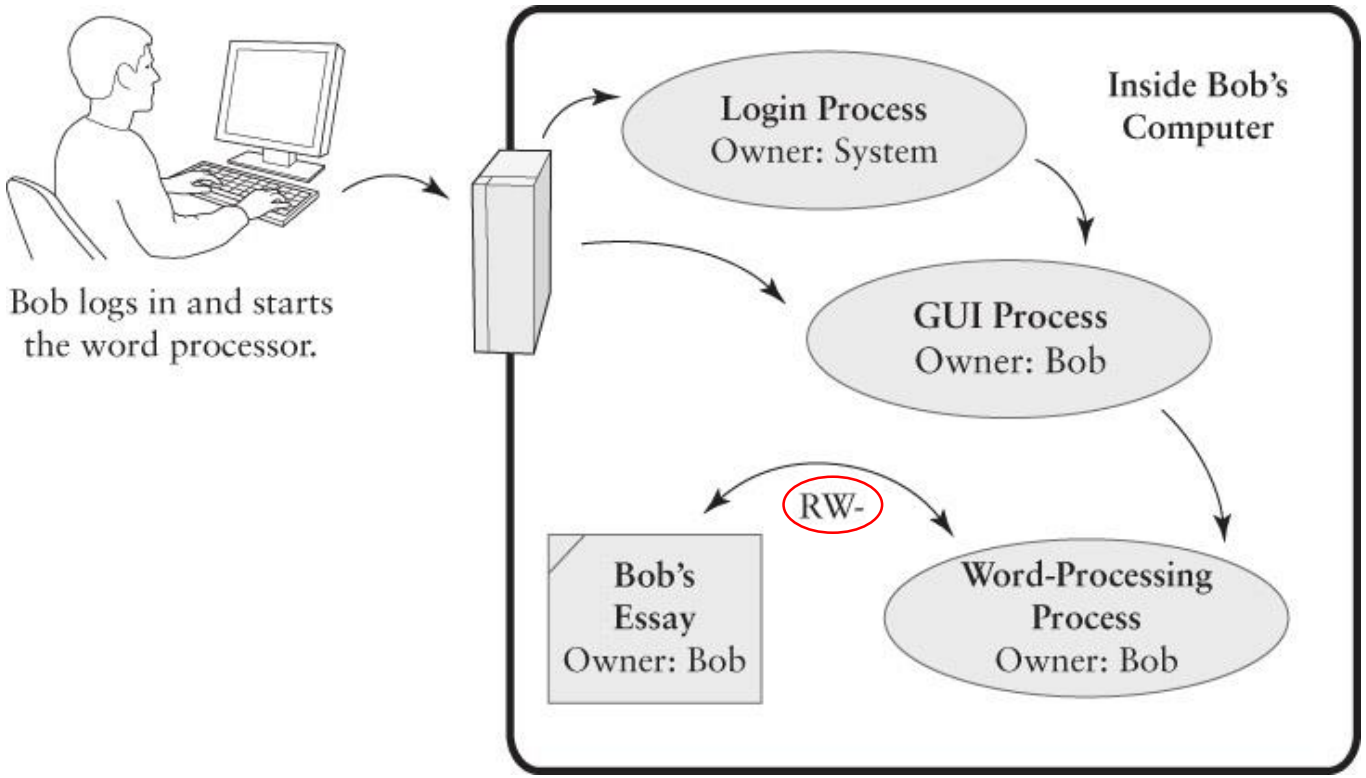
Controlling (accessing) Files

The File System

- How we guarantee users a proper restricted access?
- Similar problem to RAM access (weeks 2&3).
- Solution: hierarchical directories
  - Organise files in groups
  - Tree structure with the **root directory** as parent
    - In Windows, each driver has its own root (C:\, D:\, …).
    - In Unix-based, the system provides the root.
- If a file is not in the root, then it we need a **directory path** to find it (C:\Documents\...)

File Ownership and Access Rights

- If Bob has an essay in his computer, how can he access it without other users to interfere?
- The word processor takes on Bob's user identity to create and modify files in his behalf.

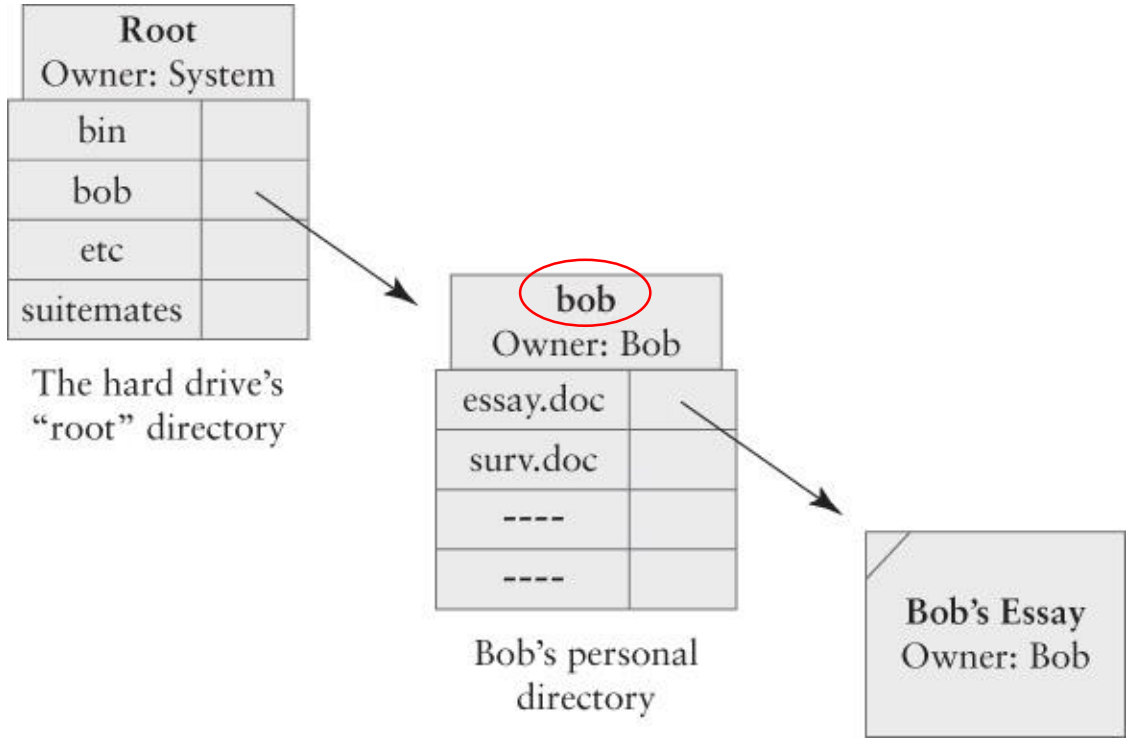


File Access Rights

- CRUD → Four types of rights:
  - Create
  - Read
  - Update (or write)
  - Delete
- Plus append and execute.
- To protect a newly created file, an OS tends to implement one of the following strategies:
  1. Default rights — Apply the same access rights to all new files).
  2. Inherited rights — Apply based on the enclosing directory.

Directory Access Rights

- We don’t have full control over a file unless we have the right to change its directory entry.
- Bob’s essay location.
- Bob’s personal directory (in root).
- To find file, Bob starts at root. Then, process looks for *bob*, and finds the file inside.



Directory Access Rights

- Three basic access rights:
  - *Read*, *Write* and *Seek*.
- Bob can _____ in his directory.
- Bob can _____ in the root, but may not _____.
- *Six specific rights:*
  - *Create (Dir/File)*
  - *Delete (Dir/File)*
  - *Read*
  - *Seek*.
- If the six are implemented, the system can achieve a ***Least Privilege*** high degree.

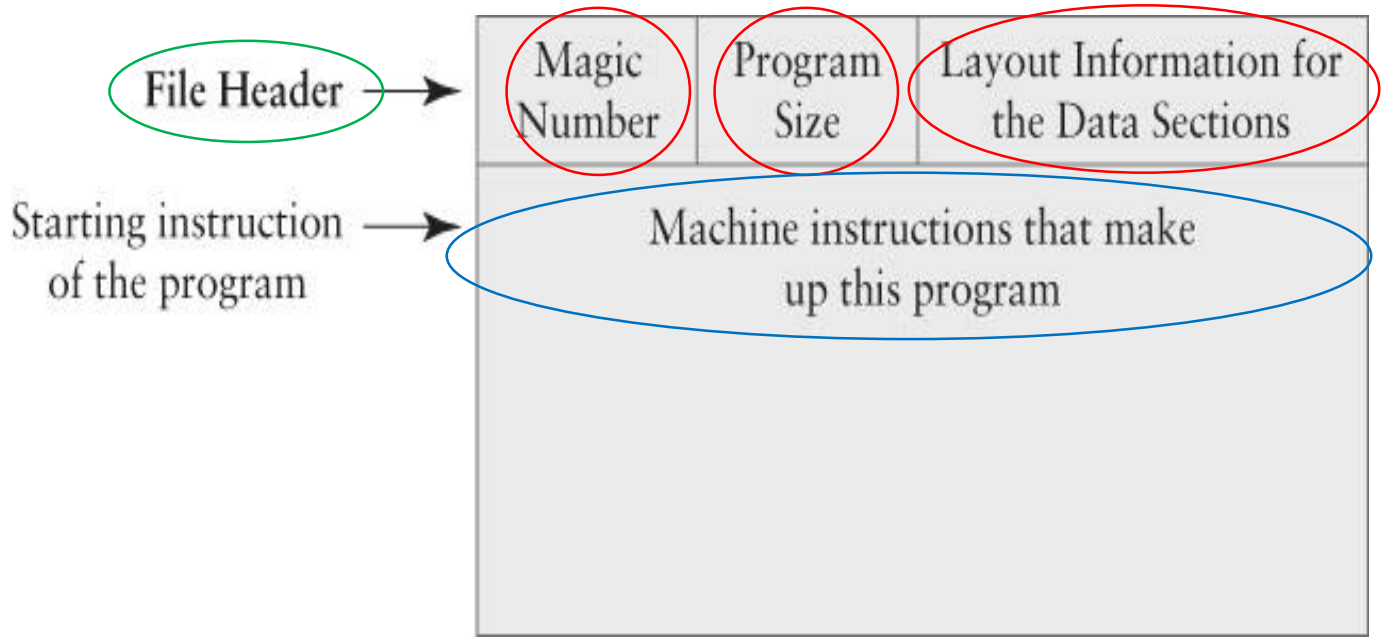
Executable Files

- Contain application programs or other programs.
- Instructions to be executed by the CPU (RAM control section).
- Typical structure of an exe:

Structure and format.

Standard data value (check).

Size of machine instructions.

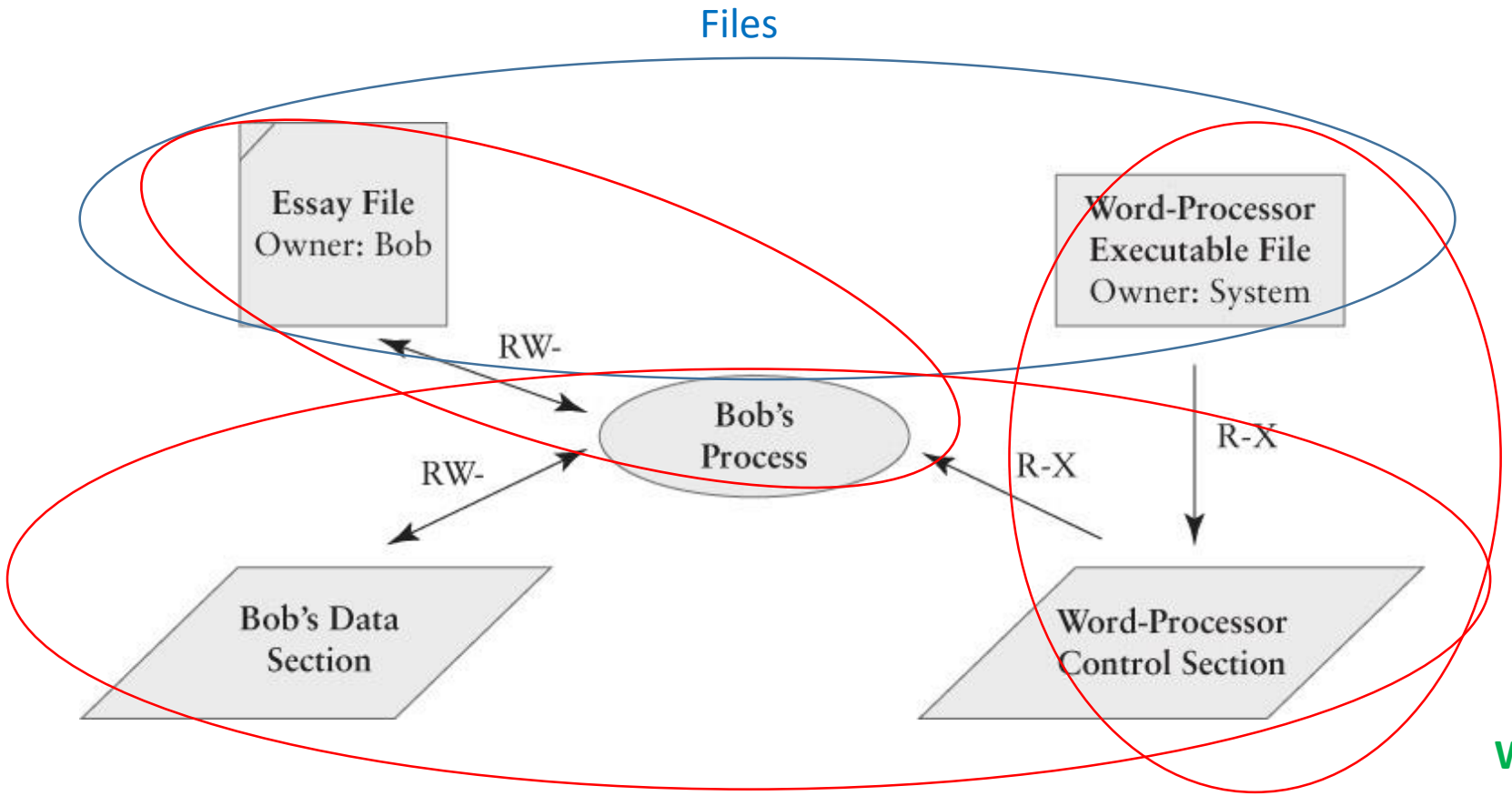


Addresses and offsets to lay out variables and stack locations in the program's data section.

When executing the program in a file, OS loads this instructions in the control section.

Execution Access Rights

- What happens when you execute a program (i.e. Word) to edit a file (i.e. .docx)?



STEP 3: Bob starts program and the file is copied by the process for its edition.

STEP 1: System reads .exe into control section.

Why no one has RWX?

STEP 2: The process uses Bob's data section to store parts of the file.

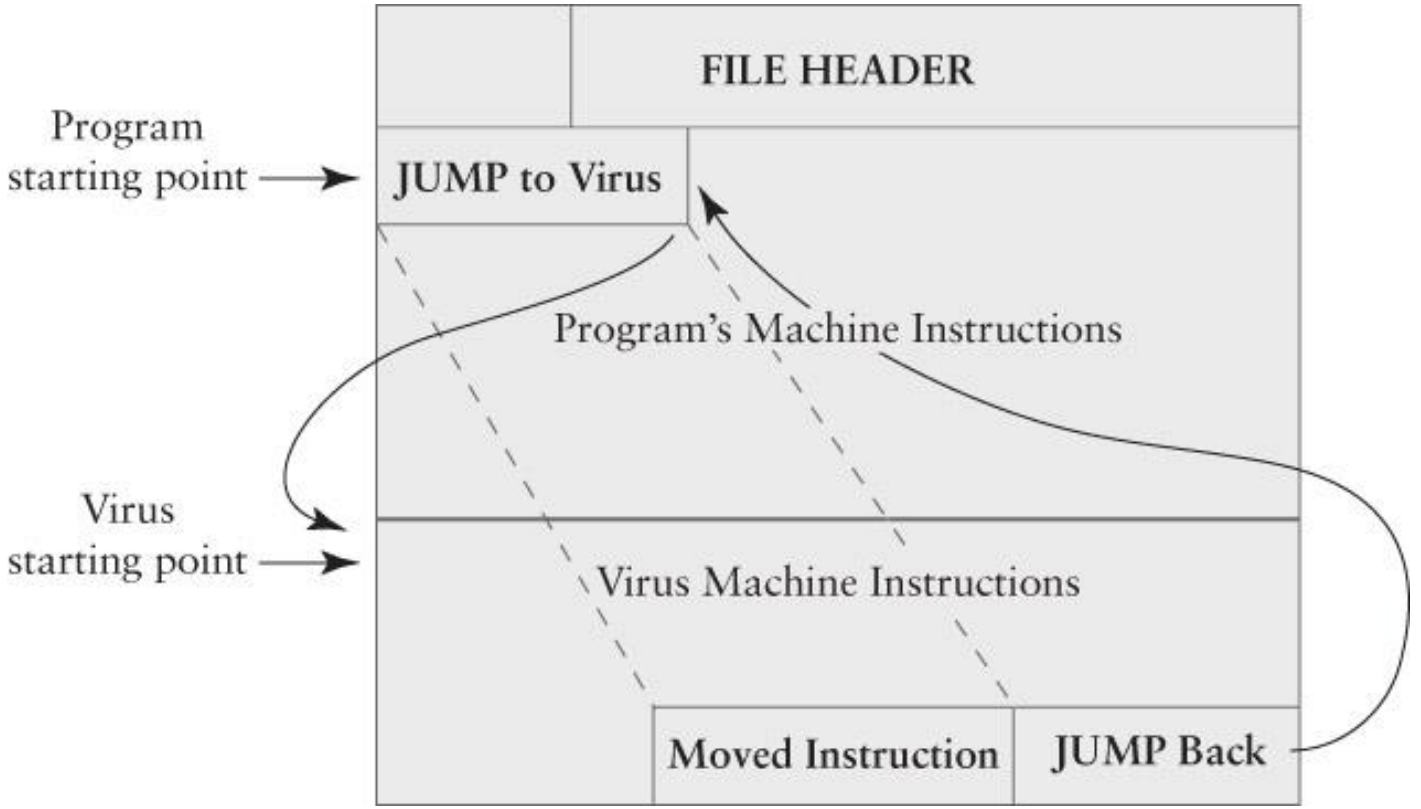
X = Execution right

Viruses and Malware

- What happened when a program gets “infected” by a virus and becomes ***mal****icious soft****ware***?

VIRUS MACHINE INSTRUCTIONS

1. Search OS file directory for every file containing an application program.
2. Skip any infected files.
3. Not infected, copy exe instructions to the end of the file.
4. Modify first instruction to “Jump to Virus” + save first instruction.
5. Jump back to application program.



Is this behaviour bad per se?

Sharing (and protecting) Files

Sharing and Protecting Files

Objectives:

1. Provide controlled access for a specific user.

2. Preserve the *Chain of Control.*

3. Permit or prevent general sharing among users.

Aims to prevent accidental and (some) malicious acts fro damaging files/processes.

Global Policies

- File permissions to apply by default.
- Determines the permissions to apply to new files.
- Two types:
  - Isolation: No access granted to other user's files.
  - File-sharing: Read access granted to other user's files.

- To maintain Chain of Control, file-sharing must avoid system files.

Tailored Policies

- According to specific cases (i.e. working on a common file).
- Three types:
  - Privacy: Block all access to certain files in a sharing environment.
  - Shared reading: Grant read-only access to certain files in an isolation environment.
  - Shared updating: Grant full access to certain files in either environment.

Security Controls

- We can rely on the OS to protect files as long as:
  1. The OS protections are always applied when we access our files.
  2. There is no way to bypass the OS protections.

- Basic Principle: ***Deny by Default***
  - We always start by granting no access.
  - We add access rights.
  - This makes it easier to assign the right permissions and achieve Least Privilege.

Managing Access Rights

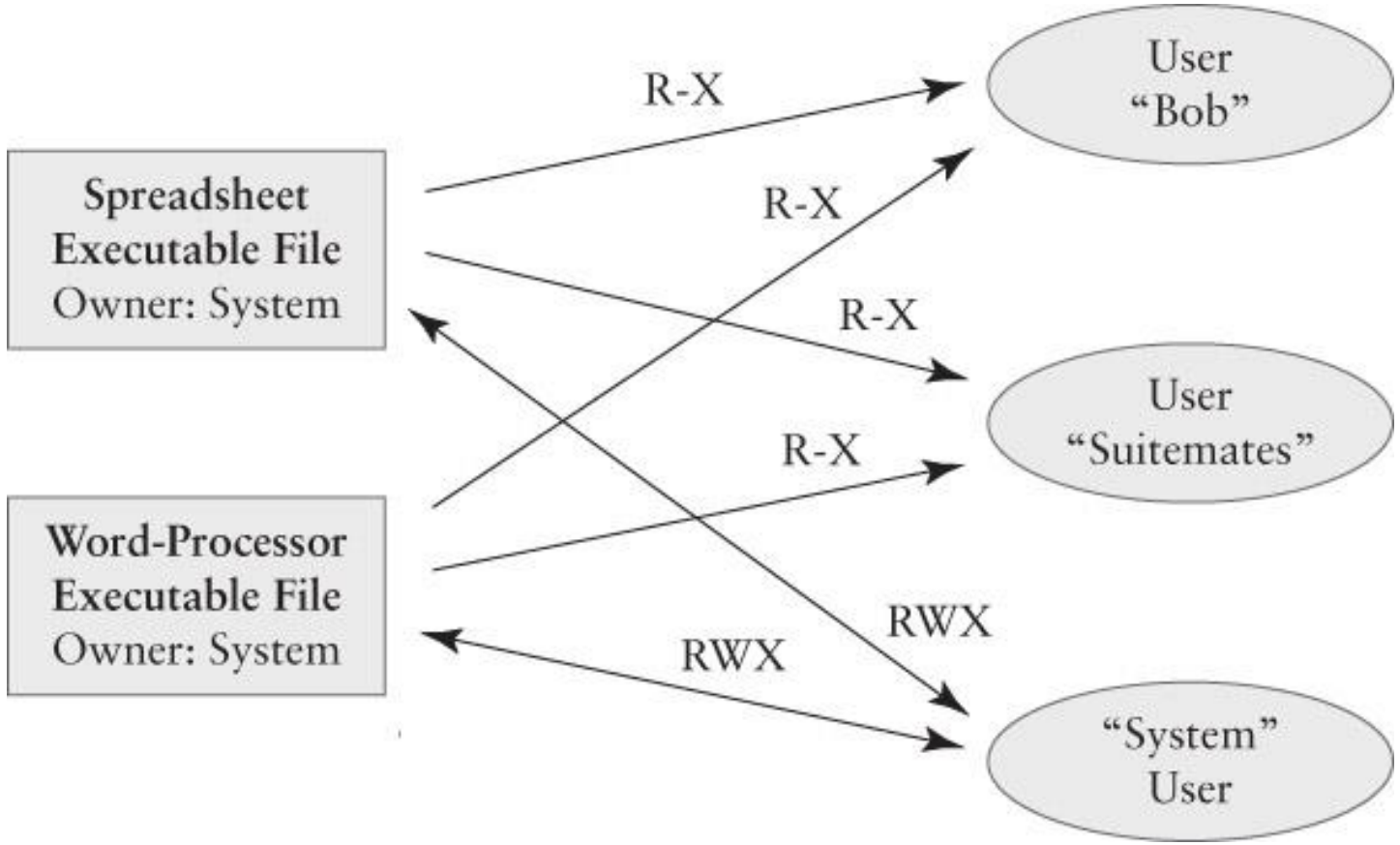
- A full matrix of all files (rows) vs users-processes (columns) can be very long.
  - Windows has 13’000 files just for OS components!
- Solution 1: Cluster by row (associate rights to files). → ***File Permissions***
  - POSIX Standard.
- Solution 2: Cluster by column (associate rights to users or processes). → ***Capability-based Security***
  - Examples: FreeBSD, the POSIX standard.

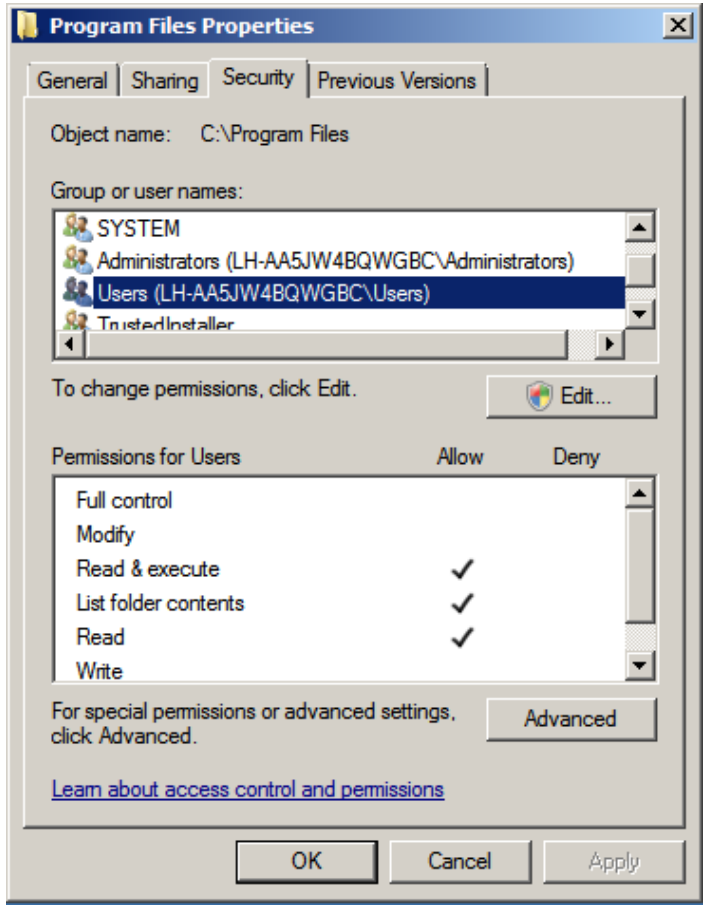
Which one is “easier”?

Which is most common in OS?

Access Rights for Executable Programs

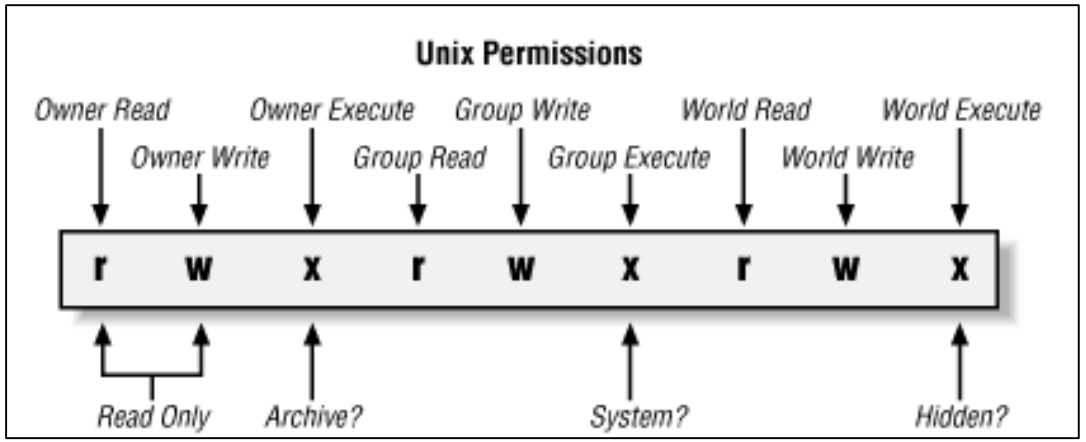
- File Permissions: **Permission flags** (Unix) or **Access Control Lists** (Win).





Permission Flags (Unix)

- Each file has ***nine*** *flags:*
  - 1-3: Read, write and execute for the file's owner ("u").
  - 4-6: for users belonging to the group associated to the file owner ("g").
  - 7-9: for other users ("o").



Other cases:
OpenVMS: 4X4

- A flag is indicated as "–" when is false, otherwise the corresponding letter is used.
- The command CHMOD (change mode) allows to raise/lower flags.

Permission Flags (Unix): Examples

```
$ ls -l

-rw-r--r--@ 1 rick ops  4321  Nov 23 08:58 data1.txt
-rwxr-xr-x  1 rick ops  12588 Nov 23 10:19 hello
-rw-r--r--@ 1 rick rick 59    Nov 23 10:18 hello.c
```

- Ignore first hyphen.
- First file is owned by *rick* and is identified by the *ops* group. It has the identifier 4321, was created on Nov 23 @ 8:58 and is called “data1.txt”. It has default flags (the user that owns it can read and write, and both its group and other users can only read it.).
- Second file (an executable) has same owners, but rick can do anything to it while the group and the world can read it and execute it. This permission is set by using the command `CHMOD 755`.
- Third file is owned by rick, and can only be written by him.

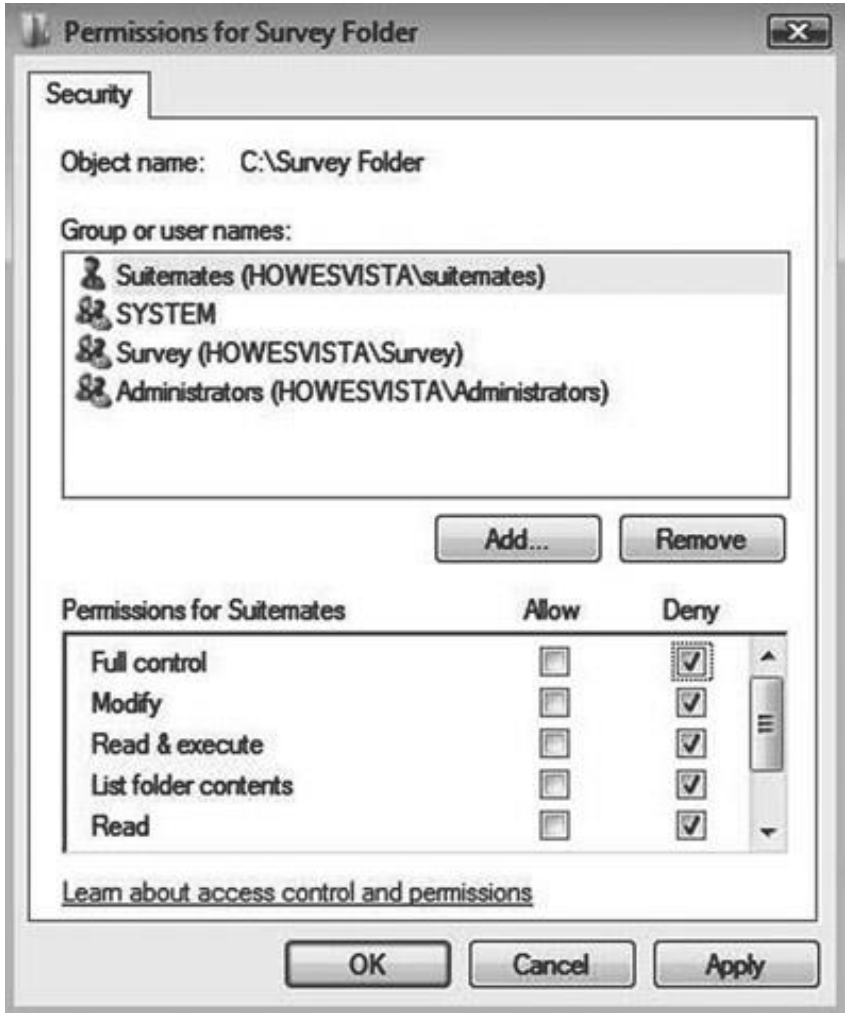
Access Control Lists

- The general purpose technique is to cluster access rights by row.
- Permission flags require a small, fixed amount of storage for each file.
- ACLs are an alternative to user groups:
  - We simply keep a list of individuals with the right to access a particular file or folder.
  - Efficient if each file needs its own tailored list.

- Problem: ACLs may be arbitrarily long (challenge for the OS).

Windows ACLs

- Present in “Professional,” “Business”, or “Enterprise” versions of Windows:
  - “Home” and “Basic” versions use simpler access lists.
- Each ACL entry gives permission for a specific user or group:
  - Users and groups are defined on the computer or by a network-wide “Domain”.
  - Each entry specifies a list of permissions.
  - Each permission can be “Allowed” or “Denied”.



Building Effective ACLs

1. Deny by Default:
   - Start with no rights, or a small set of default ones.
     - Permissions to owner and administrators.
   - Add “Allow” rights as needed.
2. Keep the rules as simple as possible.
3. Example that needs a “Deny” right.
   - A group of all students called “Students”.
   - Need a group “Students Minus Freshmen”?
   - Easiest approach: Deny “Freshmen” group!

Permission flags (UNIX) vs ACLs (Windows)

- *user_1* from the Local group sets the following rights:
  - User *user_1* has full access to the file.
  - Group Local has unspecified access.
  - Others have read-only access.
- What will *user_2* from the Local group be capable to do?

DEMO/Intro to Python

Lab 4: Creating Nested Virtual Machines

ADDITIONAL RECOMMENDATIONS FOR THIS WEEK

- Shell scripting.
- Read Chapters 3 – 4 from the text book.
- Play with permission flags for files in Linux (i.e. use lab 3 files).
- Download Python/Anaconda + find tutorials.