Aalto University

Access control in operating systems

Tuomas Aura
CS-C3130 Information security

Aalto University 2023

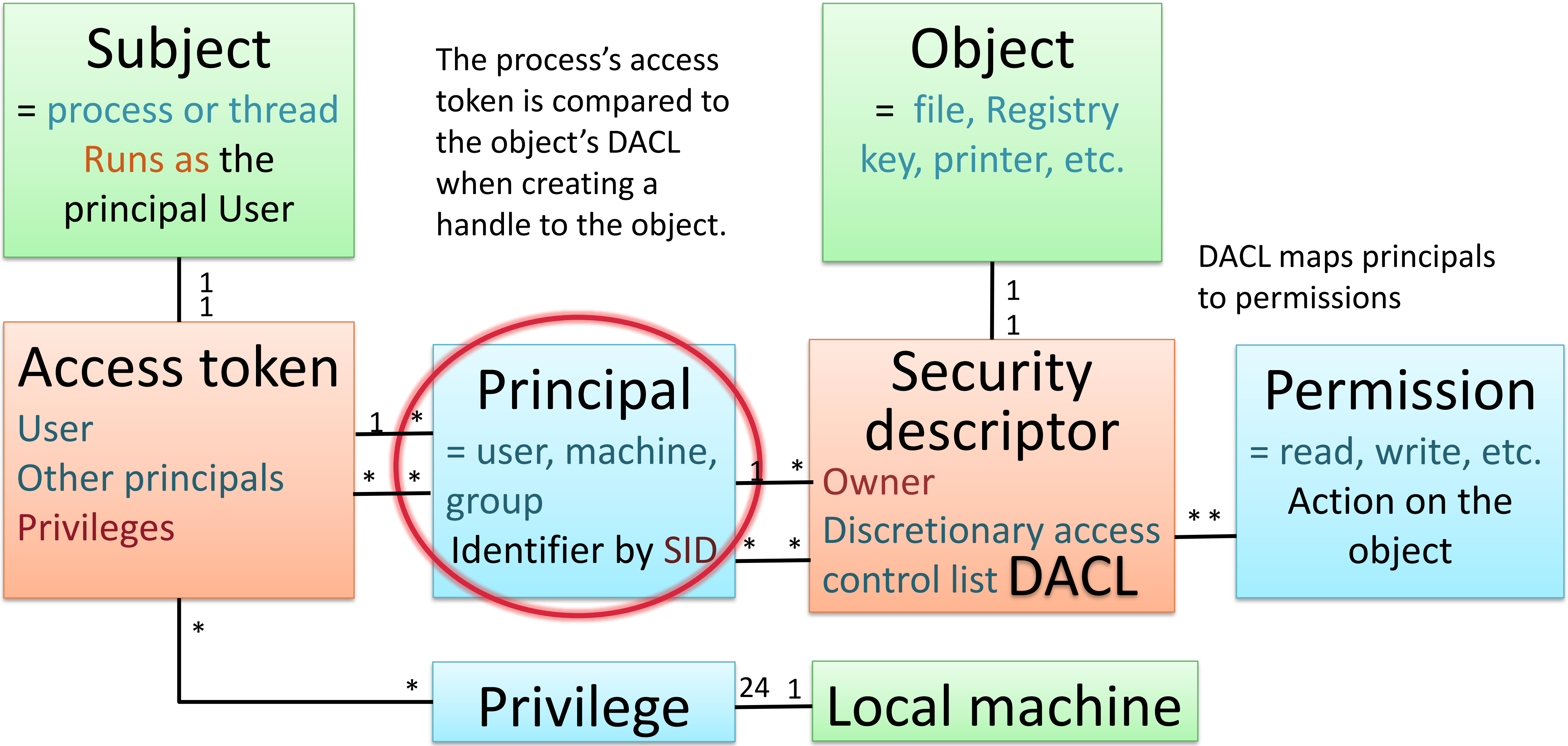
Outline

- Access control models in operating systems:
  1. Windows access control
  2. Unix permissions
  3. Other systems

Acknowledgements: This lecture has evolved from a joint course with Dieter Gollmann

WINDOWS ACCESS CONTROL

Windows security model



Subject

= process or thread

Runs as the principal User

The process's access token is compared to the object's DACL when creating a handle to the object.

Object

= file, Registry key, printer, etc.

DACL maps principals to permissions

Access token

User

Other principals

Privileges

Principal

= user, machine, group

Identifier by SID

Security descriptor

Owner

Discretionary access control list DACL

Permission

= read, write, etc.

Action on the object

Privilege

Local machine

Windows Security Model (recap)

- Principals = users, machines, groups,…
  - Something that can be authenticated
- Objects = files, Registry keys, printers, …
  - Each object has a discretionary access control list (DACL)
- Subjects = processes or threads
  - Process (or thread) runs as a principal
  - Each process (or thread) has an access token
- When is a process allowed to access an object?
  - The object's DACL is compared with the process's access token when creating a handle to the object

Security indentifier

- Principal names: machine\principal or domain\principal

  T30500-LR064\Tuomas, AALTO\aura

- Principal has a unique security identifier (SID)

  S-1-5-21-961468069-954667678-1722769278-1002

- Name can be changed; SID is permanent
  - Causes issues when moving an NTFS disk between computers; machine-scoped SIDs are not recognized

SID examples

- User SIDs (from one computer):

  S-1-5-21-961468069-954667678-1722769278-1002 = Alice

  S-1-5-21-961468069-954667678-1722769278-500 = Administrator

- How Windows creates unique user SIDs:

  S-1-5 + *machine or domain id* + *relative id*

- There are some well-known SIDs:

  S-1-1-0 = Everyone, S-1-5-18 = Local System,
  S-1-5-*domain*-513 = Domain Users, S-1-3-0 Creator/Owner

Where principals live

- Windows computer has a Local Security Authority (LSA), which stores local users and local groups (=aliases) in a hierarchical database called Registry

  T30500-LR064\Tuomas

- Windows domain has a Domain Controller (DC), which stores domain users, domain computers (=machines) and groups centrally in a hierarchical database called Active Directory (AD)
  - Names: domain\principal or principal@domain

  AALTO\aura, aura@org.aalto.fi

Windows domain

Extra material

- A Windows Server can become a Domain Controller (DC)
  - Other machines can join the domain
- DC provides authentication services to other machines
  - Domain user can log into any domain-joined machine
  - Kerberos protocol used for distributed authentication
- DC can deploy policies (group policy) to the domain-joined computers
- In large organizations, domains can form a hierarchy

How to see them

- Show local users and local groups i.e. aliases on your computer:

```
> net user
> net user tuomas
> net localgroup
> net localgroup Administrators
```

- SIDs with SysInternals *psgetsid.exe*:

```
> psgetsid                                       (machine SID)
> psgetsid tuomas
> psgetsid tuomas.aura@iki.fi                    (linked Microsoft ID)
> psgetsid Administrators                        (local administrators group SID)
> psgetsid MicrosoftAccount\tuomas.aura@iki.fi   (Microsoft account global SID)
```

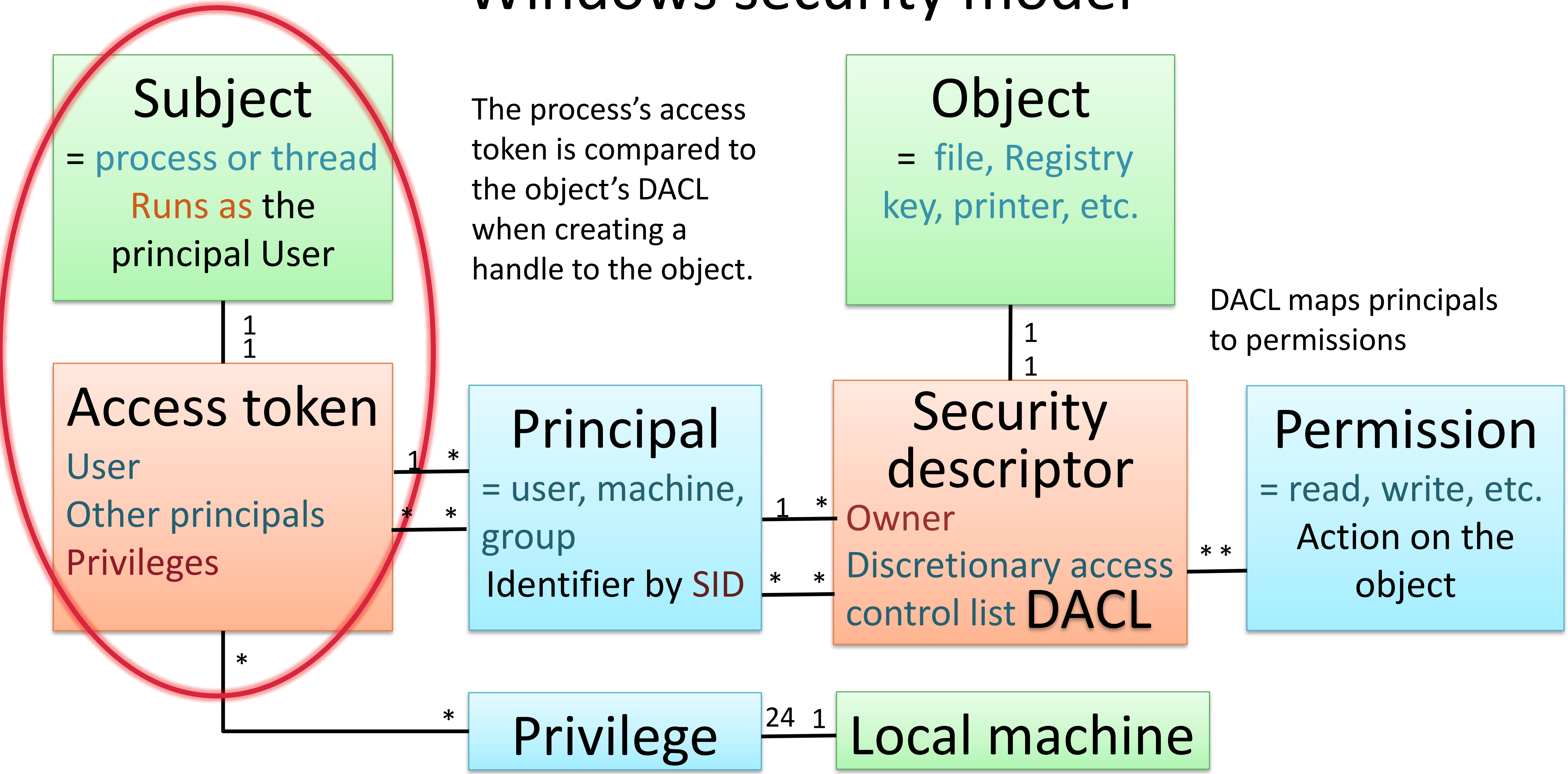
- For a UI view of the groups and users, run *compmgmt.msc*, see System Tools / Local user and Groups
- For the actual storage, run *regedit.exe*, see under HKEY_USERS
- Privileges: Run *secpol.msc*, see Local Policies / User Rights Assignment

- Domain users, groups (must run on a computer that belongs to a domain, e.g. *vdi.aalto.fi*):

```
> net user /domain                               (very slow if it is a large domain!)
> net user aura /domain
> net group /domain
> net group aut-prof-sci /domain
> psgetsid aalto
> psgetsid aalto\aura
```

SysInternals: https://technet.microsoft.com/en-us/sysinternals/bb795533

Windows security model



Subject

= process or thread

Runs as the principal User

The process's access token is compared to the object's DACL when creating a handle to the object.

Object

= file, Registry key, printer, etc.

DACL maps principals to permissions

Access token

User

Other principals

Privileges

Principal

= user, machine, group

Identifier by SID

Security descriptor

Owner

Discretionary access control list DACL

Permission

= read, write, etc.

Action on the object

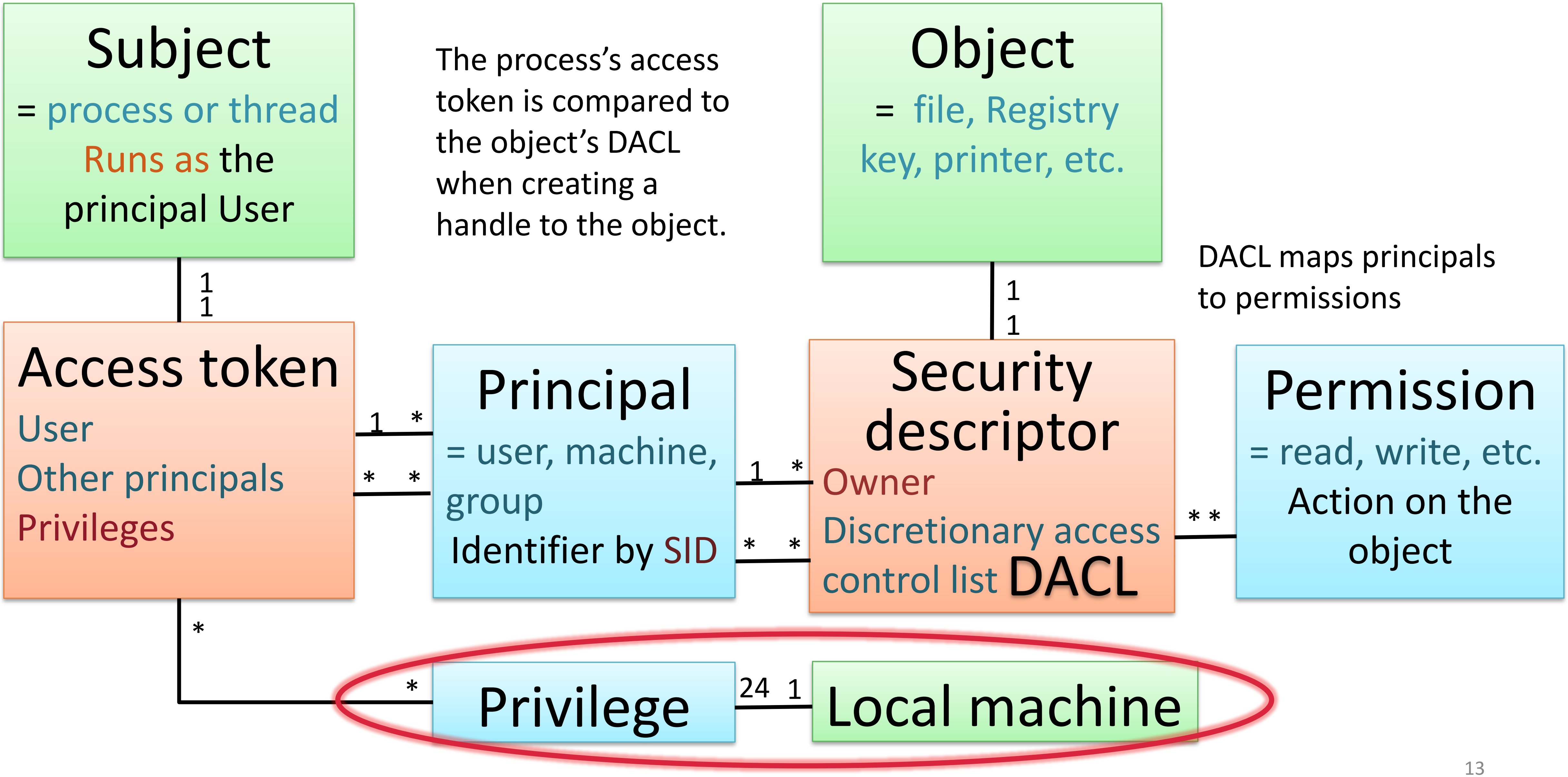
Privilege

Local machine

Access token

- Process has an access token (=security token), which contains:
  - Login user account SID (the process runs as this user)
  - SIDs of all groups where the login user is a member (recursively!)
  - All privileges assigned to these users and groups on the local computer
- Token never changes after it has been created
  - Better reliability and efficiency but slower revocation of access rights
- Child process gets a copy of the parent’s token
  - Child’s token can be restricted, disabling some SIDs

Windows security model



Subject
= process or thread
Runs as the principal User

The process's access token is compared to the object's DACL when creating a handle to the object.

Object
= file, Registry key, printer, etc.

DACL maps principals to permissions

Access token
User
Other principals
Privileges

Principal
= user, machine, group
Identifier by SID

Security descriptor
Owner
Discretionary access control list DACL

Permission
= read, write, etc.
Action on the object

Privilege

Local machine

Privileges

- Privileges are special rights on the local machine
  - Backup, audit security log, take ownership, trusted for delegation, debugging, performance profiling, shutdown, …

Restricted token

Extra material

- Inherited access token may be too powerful
- Process can assign restricted tokens to its child processes or threads
  - remove privileges
  - disable groups: change SIDs to “deny-only” mode; they are not deleted but marked as USE_FOR_DENY_ONLY
  - add restricted SIDs: a second list of SIDs that is also compared against DACLs
- Typically used in services, rarely in desktop apps

Creating subjects at logon

Extra material

- The machine is always running a logon process (winlogon.exe) as the principal SYSTEM
- When a user logs on to a machine:
  - the logon process asks for user credentials (e.g. password) and presents them to the LSA
  - the LSA (lsass.exe) verifies the credentials
  - the logon process starts a shell (explorer.exe) running as the user (=principal) and in a new logon session
- Shell spawns processes to the same logon session
- Logging off destroys the logon session and all processes in it
  - Note: Windows has no equivalent of disown/nohup. To leave a program running in the background, create a service.

Creating more subjects

Extra material

- A process can spawn a new local process (subject) by calling e.g. CreateProcess
  - Each process has its own access token
  - New process gets a copy of its parent's token
  - Threads can be given their own tokens, so that they become independent subjects
- User's network credentials (e.g. password or Kerberos ticket) are cached in the logon session

→ Process can create network logon sessions for the local login user user at other machines

How to see them

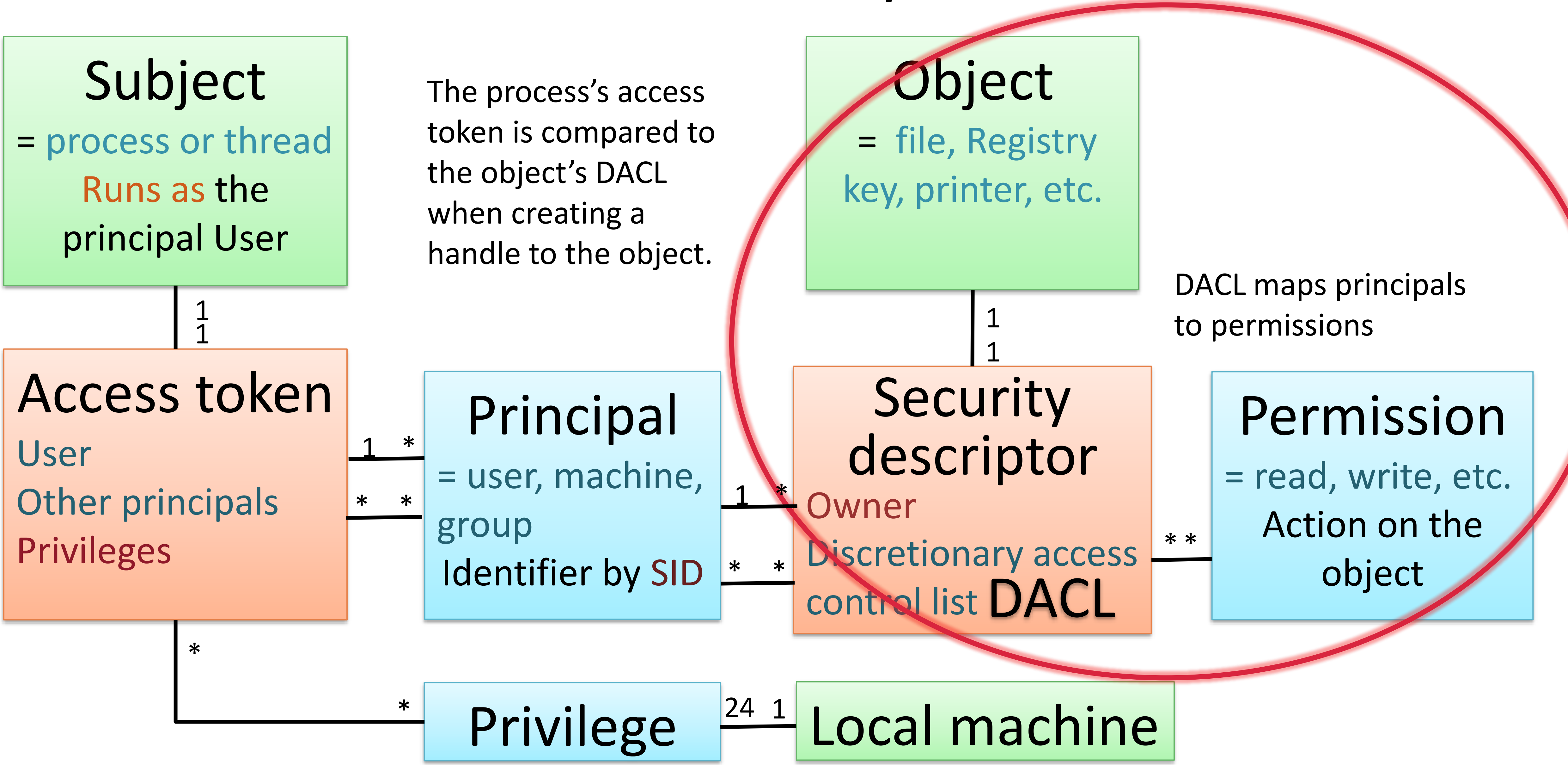
Access tokens:

1. Run **Process Explorer** *procexp.exe* from SysInternals
2. Click on a process, see Properties/Security tab for the access token
3. Note the SIDs and privileges on the process

4. The token itself is also an object (see next slides). Click on Permissions to see its DACL

SysInternals: https://technet.microsoft.com/en-us/sysinternals/bb795533

Windows security model



Subject
= process or thread
Runs as the principal User

Access token
User
Other principals
Privileges

Principal
= user, machine, group
Identifier by SID

Privilege

Object
= file, Registry key, printer, etc.

Security descriptor
Owner
Discretionary access control list DACL

Permission
= read, write, etc.
Action on the object

Local machine

The process's access token is compared to the object's DACL when creating a handle to the object.

DACL maps principals to permissions

Objects

- Objects: files, folders, Registry and AD objects, processes...
  - Some objects are containers for other objects → hierarchy
- Object has a security descriptor, which includes the discretionary access control list (DACL)
- Object also has an owner (identified by SID), who has the implicit right to read and write the DACL

Permissions

- Permissions are actions that apply to an object

- Generic and standard permissions are defined for all objects: Read, Write, Execute, All, Delete, WriteDACL, …
  - although different names are used for different object classes (e.g. Read for file and List for directory)

- Each object class has object-specific permissions: Append (for file), AddSubDir (for folder), CreateThread (for process), …

- Permissions are encoded as a 32-bit mask for fast checking

Access control list (DACL)

		SID	Permissions
ACE1	+	Diego	Full Control
ACE2	+	Lecturers	Read, Write
ACE3	+	EVERYONE	Read

- DACL is a list of access control entries (ACE)

Negative ACEs

		SID	Permissions
ACE1	–	Tuomaura	Write
ACE2	+	Diego	Full Control
ACE3	+	Lecturers	Read, Write
ACE4	+	EVERYONE	Read

- Negative ACEs are placed before positive ones
  - – DACL above grants read but no write access to the user Tuomaura

Negative ACEs are rarely used in practice.
Avoid in the course exercise!

Access control entry (ACE)

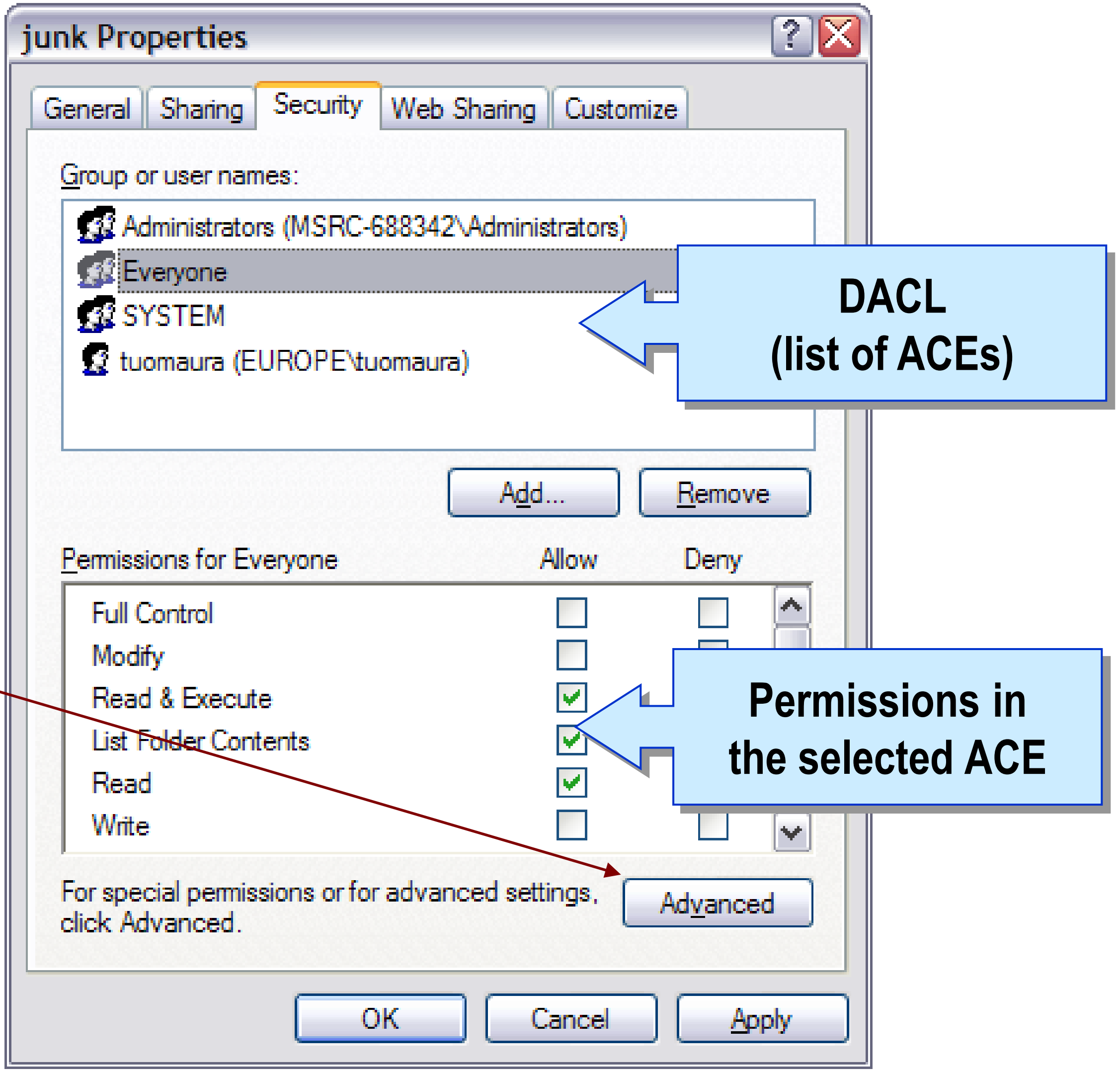
- ACE format:

Type: positive or negative (grant or deny)
Permissions: actions to grant to deny
Principal (SID): who the ACE applies to
Flags
Object Type
Inherited Object Type

How to see them

- Right-click on a file; select Properties/ Security
  - Note: Windows DACLs only exist in NTFS, not in FAT or other file systems

- Click on Advanced for a more complete view of the entire security descriptor



junk Properties

Group or user names:

- Administrators (MSRC-688342\Administrators)
- Everyone
- SYSTEM
- tuomaura (EUROPE\tuomaura)

Permissions for Everyone	Allow	Deny
Full Control	□	□
Modify	□	□
Read & Execute	☑	
List Folder Contents	☑	
Read	☑	
Write	□	□

For special permissions or for advanced settings, click Advanced.

DACL (list of ACEs)

Permissions in the selected ACE

Access check algorithm

- Process requests permissions when creating a handle to the object

```
HFILE hf = OpenFile("D:\\file.txt", &buffer, OF_READWRITE);
```

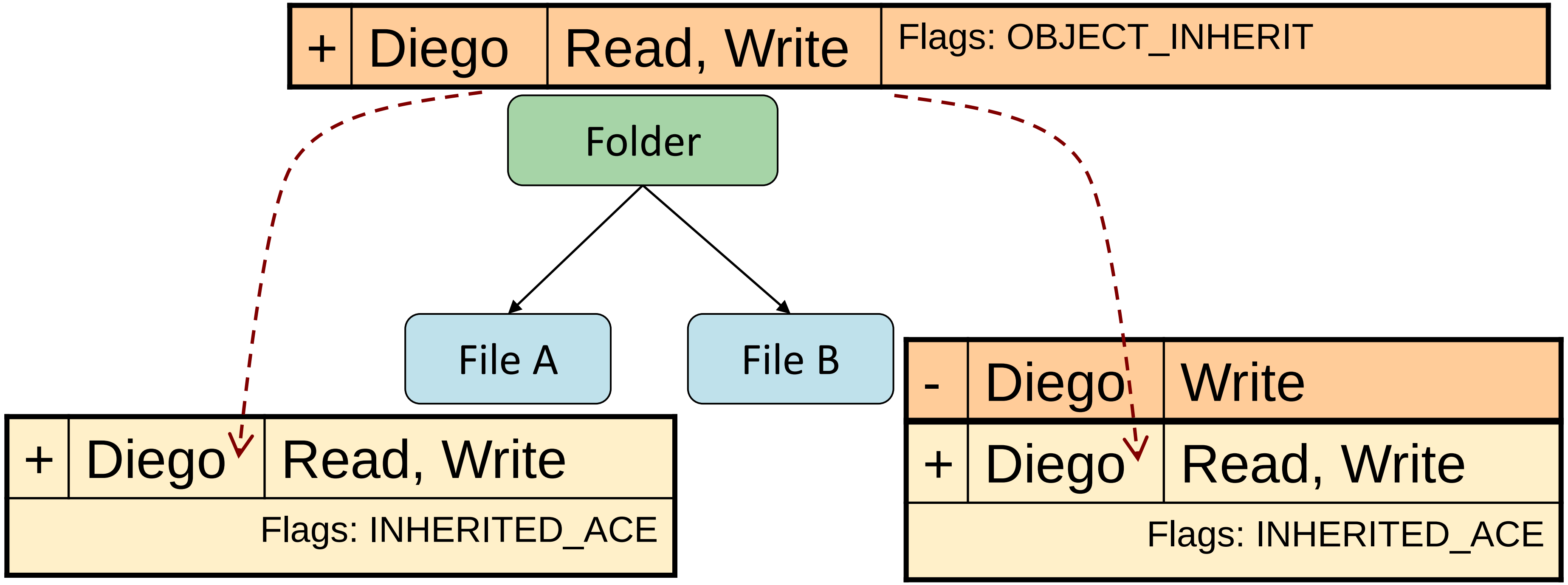
- Privileges or implicit owner permissions may alone be sufficient

- Otherwise, Windows checks the DACL as follows:
  - Look for ACEs that match both (1) any SID in the subject’s access token and (2) any desired access right
  - If a negative ACE matches, deny access
  - If positive ACEs are found for all requested permissions, grant access
  - If the end of DACL is reached, deny access

Performance and reliability

- Group memberships and privileges are determined at logon
  - User’s group SIDs are cached in the token of the login process and copied to sub-processes
  - Token unchanged even when a group or privilege is revoked from the user
- Desired access is compared against the token and DACL when creating a handle to the object – not at access time
  - Changing file DACL does not affect open file handles
- Consequences:
  - Better performance because of fewer checks
  - Better reliability because a process knows in advance whether it has sufficient access rights for a task
  - As a downside, no immediate revocation of access rights

Containers and ACE inheritance



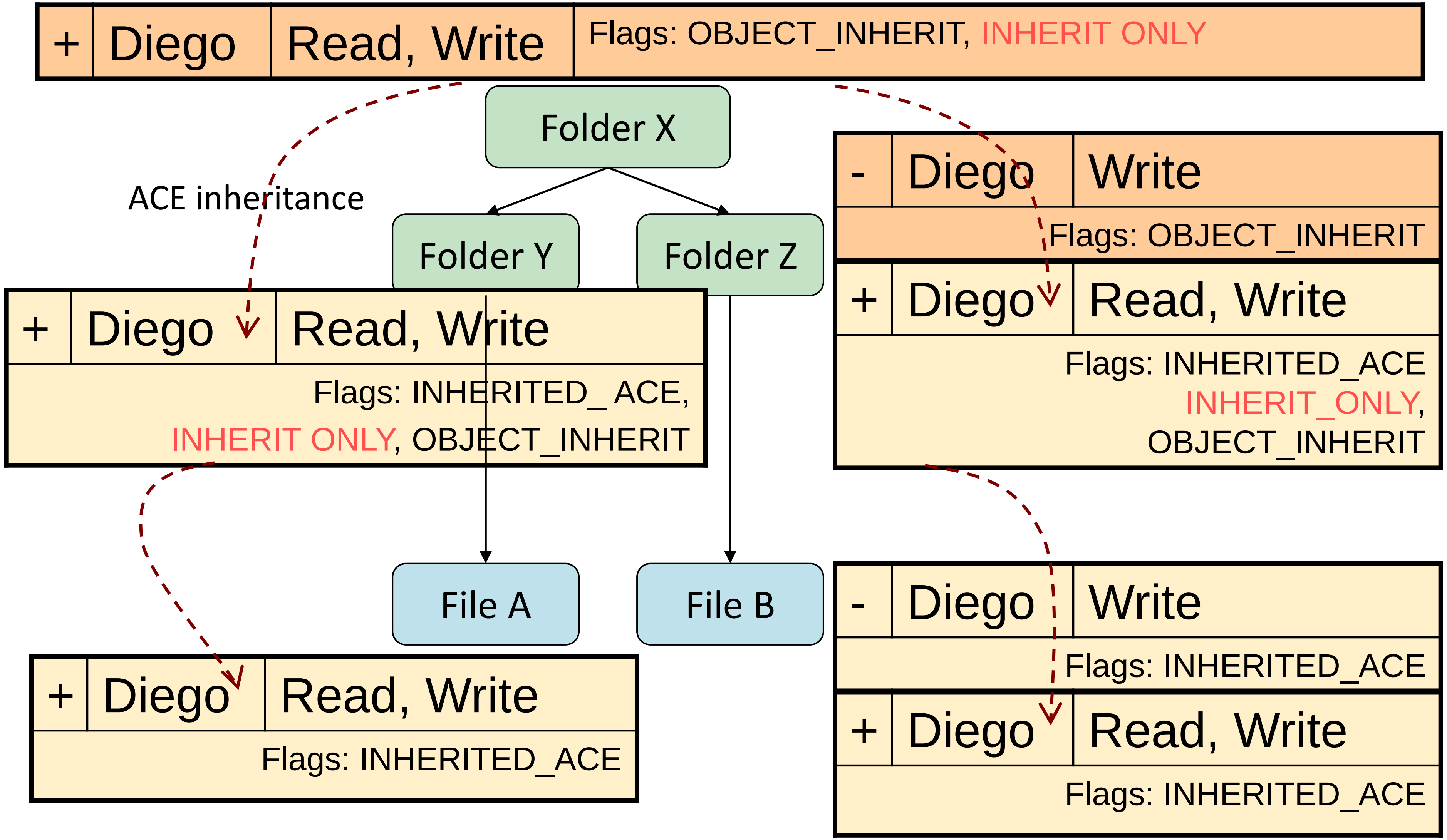
+	Diego	Read, Write	Flags: OBJECT_INHERIT

+	Diego	Read, Write
		Flags: INHERITED_ACE

-	Diego	Write
+	Diego	Read, Write
		Flags: INHERITED_ACE

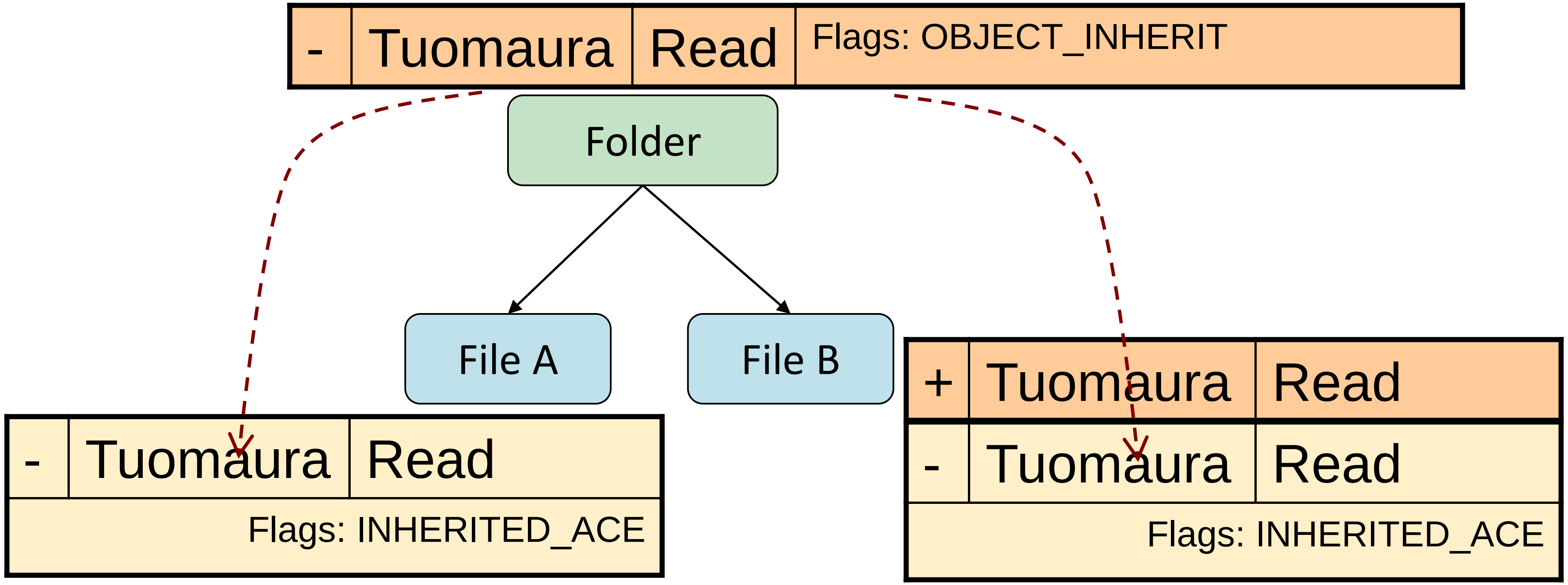
- Container objects can have inheritable ACEs
- Inheritable ACEs are copied down the hierarchy
- Directly assigned ACEs override the inherited ones

Container hierarchy



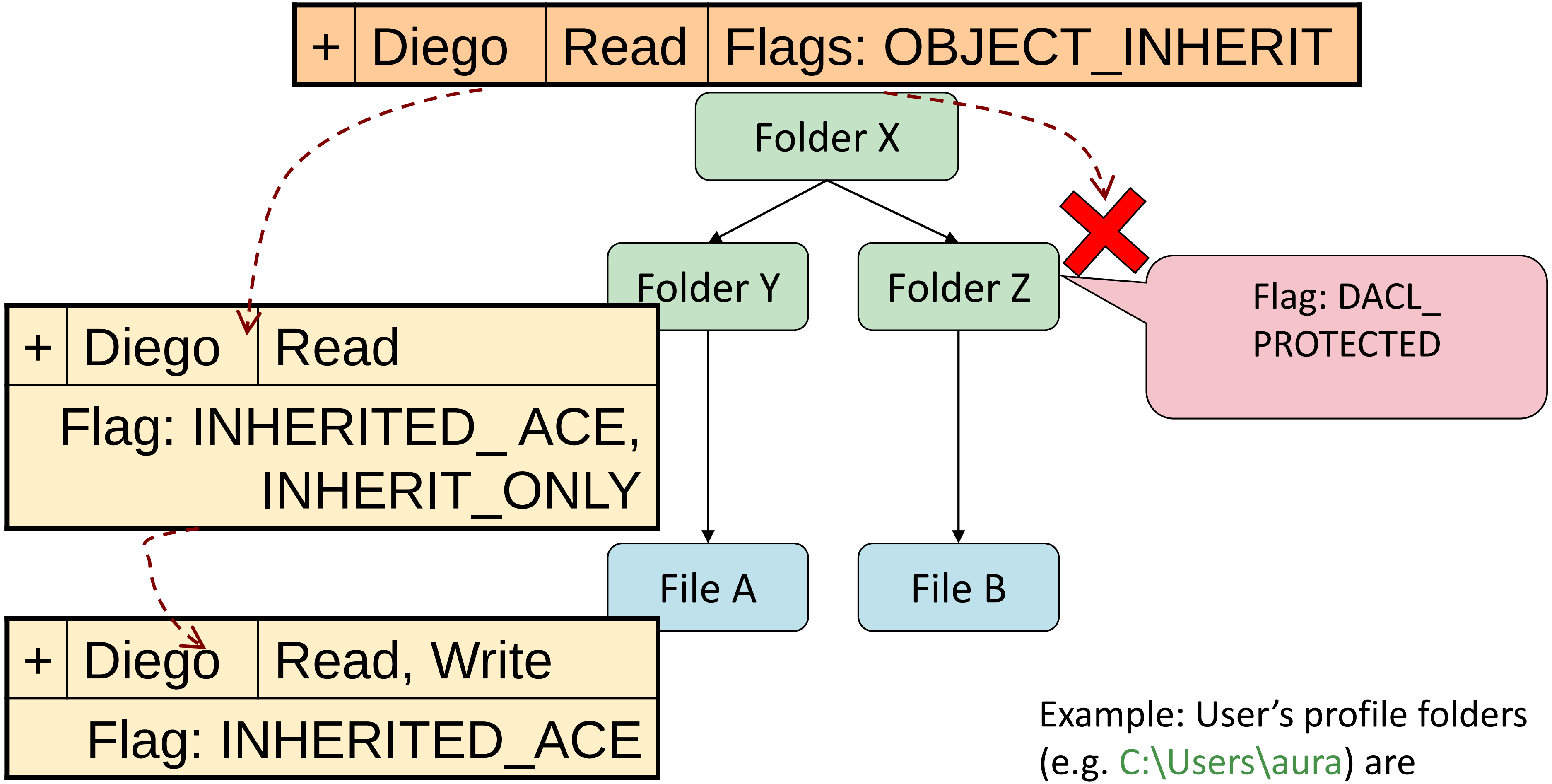
Inheriting negative ACEs

Extra material



- It is possible to override inherited negative ACEs because inherited ACES are placed at the end of the list

Blocking inheritance



+	Diego	Read	Flags: OBJECT_INHERIT

+	Diego	Read
	Flag: INHERITED_ ACE, INHERIT_ONLY	

+	Diego	Read, Write
	Flag: INHERITED_ACE	

Flag: DACL_ PROTECTED

Example: User’s profile folders (e.g. C:\Users\aura) are protected, so that other users can read the file system above but not my files.

Inheritance flags

Extra material

- Flags on ACEs:

OBJECT_INHERIT_ACE	ACE applies to leaf objects
CONTAINER_INHERIT_ACE	ACE applies to container objects
NO_PROPAGATE_INHERIT_ACE	Applies to immediate children only
INHERIT_ONLY_ACE	Does not apply to the container itself
INHERITED_ACE	The ACE has been inherited

(Inheritable ACEs can apply to leaf objects, to containers, or to both)

- Flags on DACLs:

SE_DACL_PROTECTED	This object does not inherit from containers above itself

Try to see these flags in the Windows user interface!

Advanced inheritance

Extra material

- Inheritance simplifies system administration, but few users and developers understand or use it
- Container hierarchies with inheritance:
  NTFS file system, Registry, Active Directory
- Inheritable ACEs can apply only to leaf objects or only to containers
- Inheritable ACEs can apply to all objects or only to a specific object type
- Special CREATOR_OWNER SID in an ACE matches the object owner

Performance and reliability

- Inherited ACEs are cached in sub-object DACLs to make access control decisions faster
- Consequence:
  - Changes in permissions on a folder are copied all the way down the file-system hierarchy. This is why it takes very long, e.g. to give one user access to another users’s home folder, or to Windows

How to see them

- DACL in UI: In Explorer, right-click on a file or folder and select Properties, Security tab. This is the DACL on the object. Click on Advanced for a clearer view.
- DACL from command line (incomplete and difficult to read output):

```
> icacls file.txt
> icacls mydir /T /C                              (recursive)
> icacls mydir /save output.txt /T /C
                                      (complete information as SDDL)
```

- *accesschk.exe* from SysInternals for more readable output:

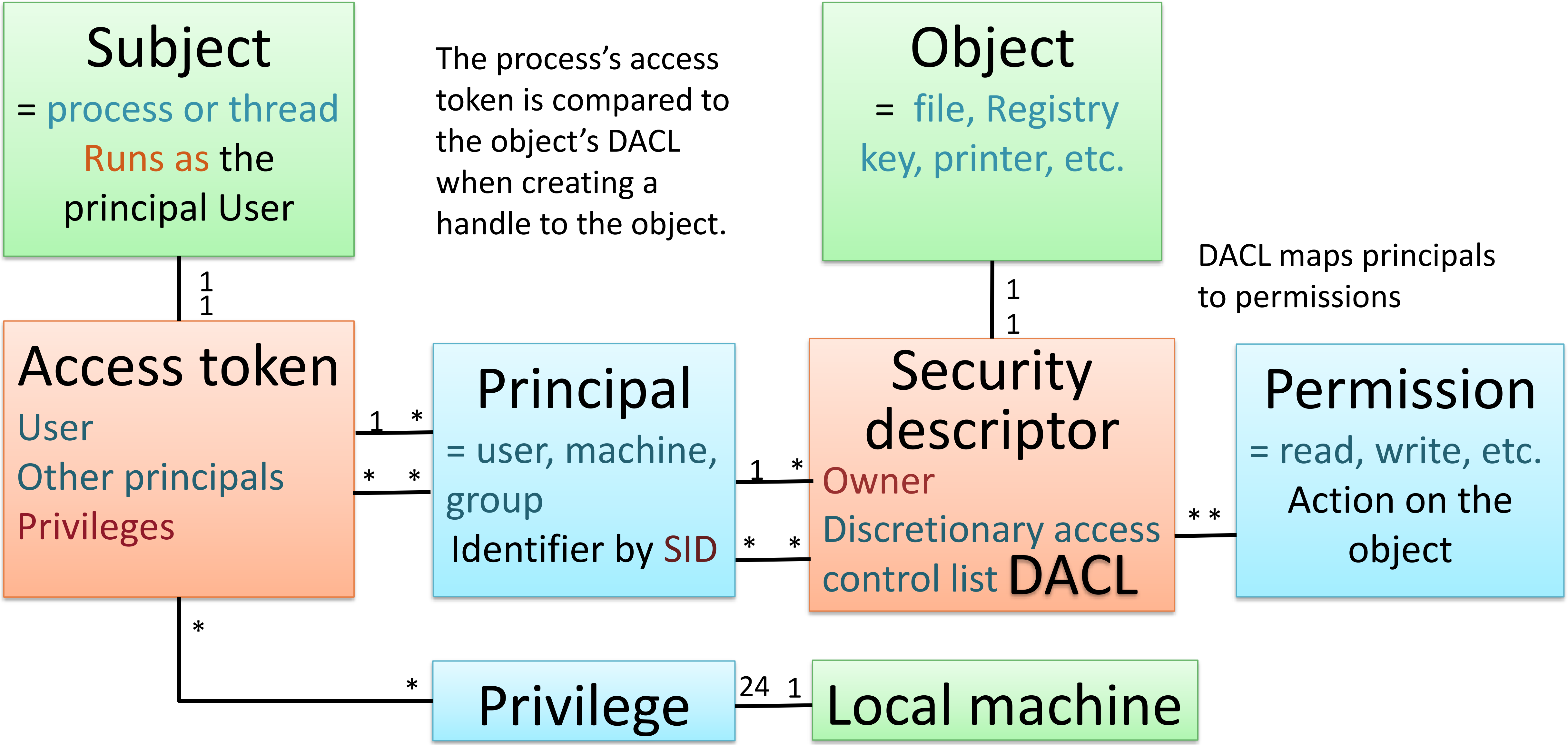
```
> accesschk Desktop\kitten.mp4 –l
> accesschk Desktop –dl
> accesschk Desktop/* -l
```

- *accessenum.exe* from SysInternals finds who has access to files in a given folder

SysInternals: https://technet.microsoft.com/en-us/sysinternals/bb795533

SDDL: https://msdn.microsoft.com/en-us/library/aa379567.aspx

Windows security model



Subject

= process or thread

Runs as the principal User

The process's access token is compared to the object's DACL when creating a handle to the object.

Object

= file, Registry key, printer, etc.

DACL maps principals to permissions

Access token

User

Other principals

Privileges

Principal

= user, machine, group

Identifier by SID

Security descriptor

Owner

Discretionary access control list DACL

Permission

= read, write, etc.

Action on the object

Privilege

Local machine

WINDOWS ACCESS CONTROL – MORE DETAILS

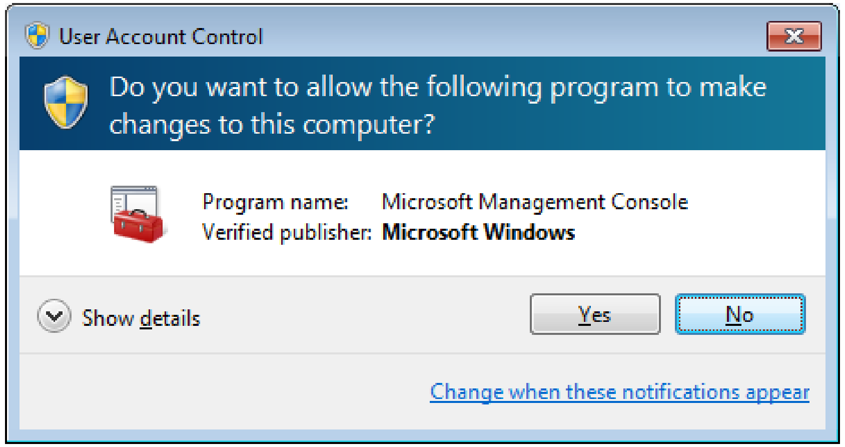
Minimum privilege

- Principle of least privilege: only run with admin rights when needed

- Windows user account control (UAC)

  Interactive user has two security tokens:

  1. Restricted token with Administrator group denied (and medium integrity level)
  2. Unrestricted token with all the Administrators group rights (and high integrity level)



User Account Control

Do you want to allow the following program to make changes to this computer?

Program name: Microsoft Management Console

Verified publisher: **Microsoft Windows**

- Process elevation:
  - Right click and select Run as Administrator
  - In PowerShell (similar to Linux *sudo):*

```
> Start-Process -Verb runas notepad.exe
```

Controlled invocation?

- Controlled invocation: allow user to invoke operations that require system privileges, but in a controlled way
- Windows does not have the equivalent of Linux SUID programs (see the Linux part of this lecture) that run with root privileges but are invoked by ordinary users. Why?
  - Home users belong to Administrators local group on their machine and can run a process as administrator if needed (UAC)
  - Software for non-admin users is written to not require elevated privileges
  - Professional administration of Windows computers is done by domain admins; thus, controlled invocation of local admin rights makes only limited sense
  - Remote users (e.g. on another machine in the same domain) should be treated the same as locally logged-on users, and that leads to a different architectural solution than SUID
- Windows service *can* run as admin and handle requests from applications
  - The user application needs to communicate with the service: inter-process communication with Windows Communication Foundation (WCF) and named pipes (*NetNamedPipeBinding*)
  - Service can learn the client identity from WCF (see *ServiceSecurityContext*)
  - More commonly, the service switches between the service user's and the client user's access rights by impersonating the client

  BUT, remember that if Microsoft wanted you to imitate SUID in Windows, they would have made it easier!

Windows integrity mechanism

- Security token has integrity level
- Security descriptor has mandatory label
  - Levels: Untrusted, Low, Medium, High, System
  - Policies associated with the label: NO_WRITE_UP, NO_READ_UP
  - Implemented as SIDs: S-1-16-xxx in System ACL
  - Process level is assigned based on its groups; object level is equal to or lower than the creator process

https://msdn.microsoft.com/en-us/library/bb625963.aspx

Windows integrity applications

Extra material

Mandatory integrity label is used to:

- Prevent untrusted software from sending window messages to trusted software and to security dialogs
  - User interface privilege isolation (UIPI)
- Sandbox particularly vulnerable processes:
  - IE, Edge, web cache used to be low to prevents compromise of the file system via browser vulnerabilities. (Chromium-based Edge seems to use different app sandboxing techniques.)
  - Q: How does IE write to Downloads? (*runtimebroker.exe*)

https://msdn.microsoft.com/en-us/library/bb625963.aspx

System ACL

- In addition to DACL, object has a SACL
- System ACL (SACL) specifies
  - Integrity level
  - Rules for generation of audit messages for granted and denied access requests (see Event Viewer / Windows Logs / Security)

Yet another label: Zone Identifier

- Windows also labels files with Zone Identifier

0	My Computer
1	Local Intranet Zone
2	Trusted sites Zone
3	Internet Zone
4	Restricted Sites Zone

- Stored in an alternate data stream in NTFS
- Why not use the integrity level instead?
  – User needs more information to decide about upgrading the file

How to see them

- Restricted token:
  1. Run Process Explorer *procexp.exe* from SysInternals
  2. See a normal process like *powerpnt.exe*. The token is restricted: Administrators group is denied.
  3. Run *powershell.exe* as Administrator and compare the process token.

- Integrity level on processes:
  1. Run Process Explorer *procexp.exe* from SysInternals
  2. Find the Mandatory Label on IE/Edge (click on a process, see Properties/Security tab for the access token, and browser the ACEs)
  3. Find also *runtimebroker.exe,* which mediates access to the file system from processes that are on low integrity level
- Integrity level on objects with *accesschk.exe* from SysInternals:

```
> accesschk C:\Users\tuoma\appdata -v
```

- Zone indetifiers: try this in Windows PowerShell:

```
> cd c:\Users\myself\Downloads
> get-item -path .\photo.jpg -stream *
> get-content -path .\photo.jpg -stream Zone.Identifier
```

SysInternals: https://technet.microsoft.com/en-us/sysinternals/bb795533

UNIX PERMISSIONS

Unix / Linux permissions are a a very old topic and there are good online tutorials. Please read them, not just the slides

Principals

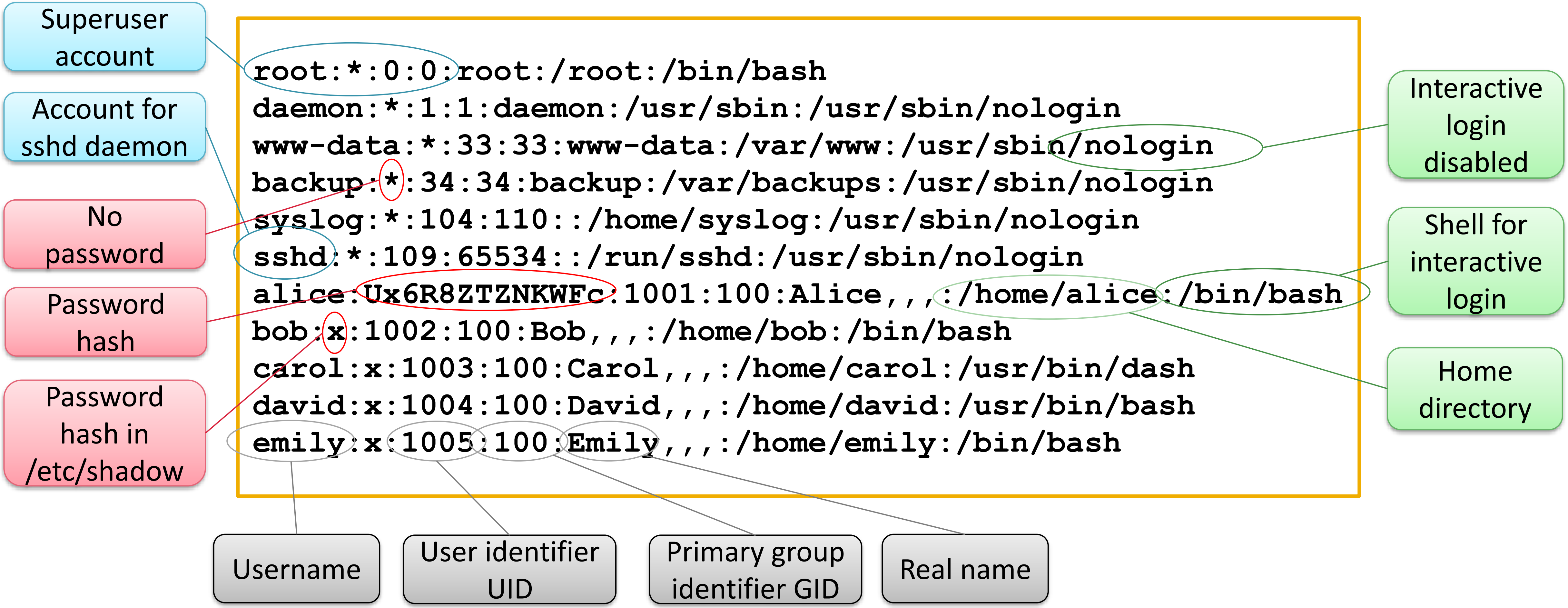
- The principals in Unix are users and groups
- Users have username and user identifier (UID)
- Groups have group name and group identifier (GID)
- UID and GID are usually 16-bit numbers, now 32 bits in Linux

    0 = root

    19057 = aura

- Both names and identifiers are permanent; difficult to change
    – UID values often differ from system to system

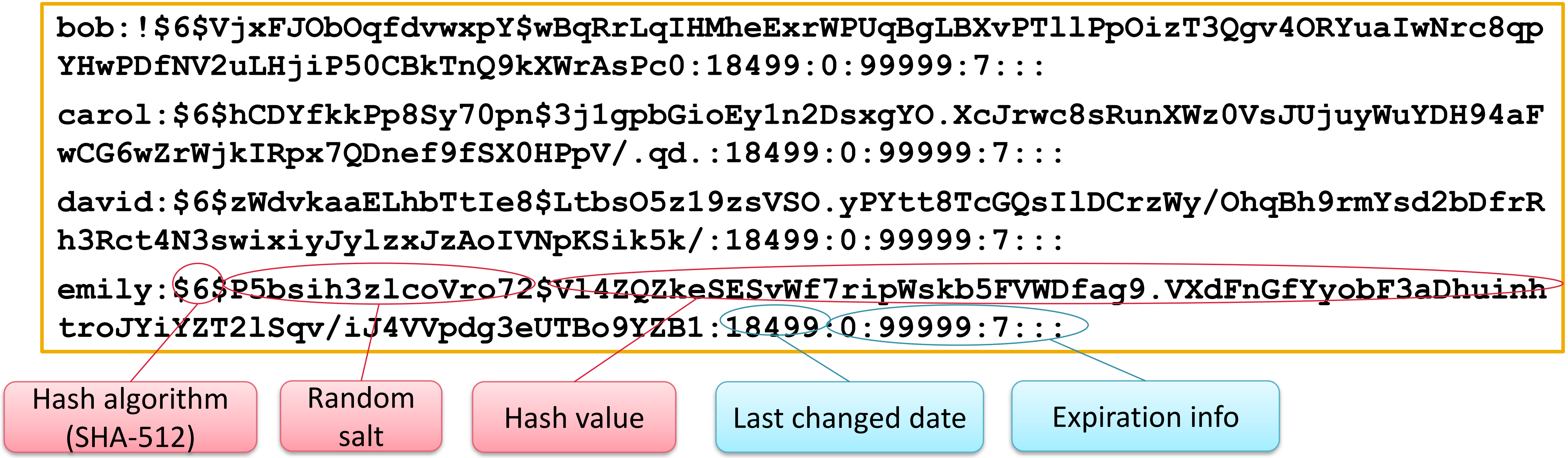
User accounts in /etc/passwd



```
root:*:0:0:root:/root:/bin/bash
daemon:*:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:*:33:33:www-data:/var/www:/usr/sbin/nologin
backup:*:34:34:backup:/var/backups:/usr/sbin/nologin
syslog:*:104:110::/home/syslog:/usr/sbin/nologin
sshd:*:109:65534::/run/sshd:/usr/sbin/nologin
alice:Ux6R8ZTZNKWFc:1001:100:Alice,,,:/home/alice:/bin/bash
bob:x:1002:100:Bob,,,:/home/bob:/bin/bash
carol:x:1003:100:Carol,,,:/home/carol:/usr/bin/dash
david:x:1004:100:David,,,:/home/david:/usr/bin/bash
emily:x:1005:100:Emily,,,:/home/emily:/bin/bash
```

Shadow passwords

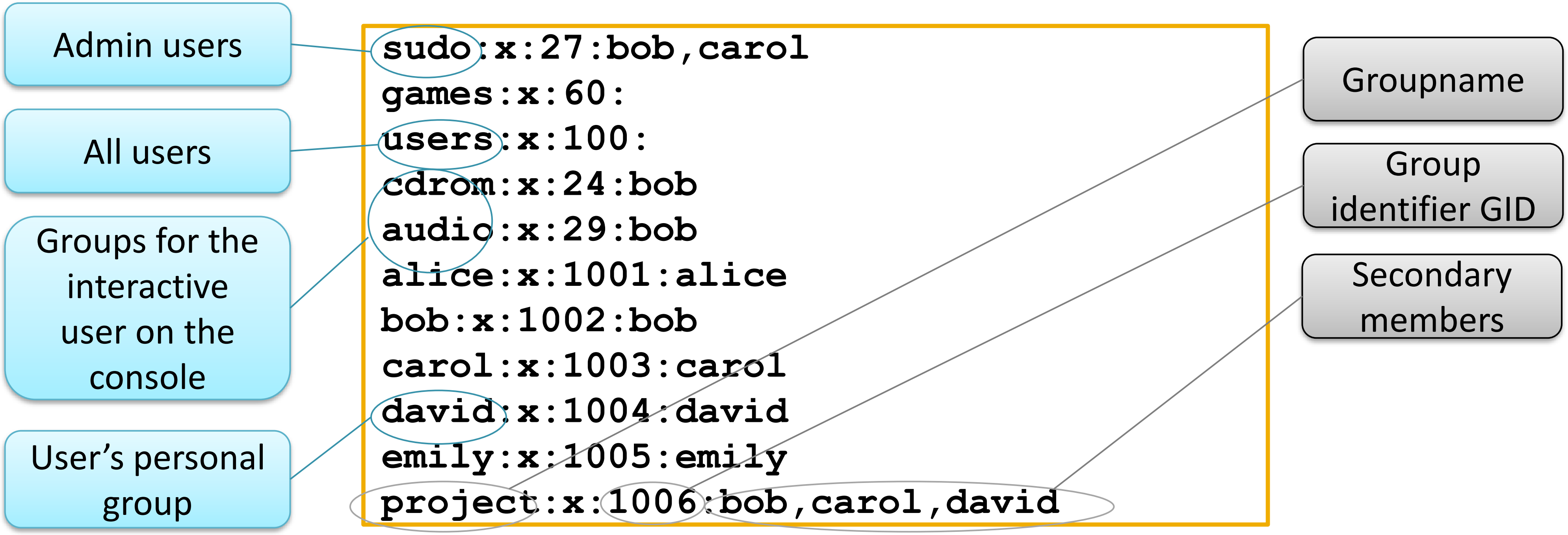
- Password hashes traditionally stored in the world-readable file /etc/passwd, hashed with crypt(3), which is based on DES
- Modern Linux systems store passwords in /etc/shadow, which is read-protected, hashed with a modern hash function such as SHA512



```
bob:!$6$VjxFJObOqfdvwxpY$wBqRrLqIHMheExrWPUqBgLBXvPTllPpOizT3Qgv4ORYuaIwNrc8qp
YHwPDfNV2uLHjiP50CBkTnQ9kXWrAsPc0:18499:0:99999:7:::

carol:$6$hCDYfkkPp8Sy70pn$3j1gpbGioEy1n2DsxgYO.XcJrwc8sRunXWz0VsJUjuyWuYDH94aF
wCG6wZrWjkIRpx7QDnef9fSX0HPpV/.qd.:18499:0:99999:7:::

david:$6$zWdvkaaELhbTtIe8$LtbsO5z19zsVSO.yPYtt8TcGQsIlDCrzWy/OhqBh9rmYsd2bDfrR
h3Rct4N3swixiyJylzxJzAoIVNpKSik5k/:18499:0:99999:7:::

emily:$6$P5bsih3zlcoVro72$Vl4ZQZkeSESvWf7ripWskb5FVWDfag9.VXdFnGfYyobF3aDhuinh
troJYiYZT2lSqv/iJ4VVpdg3eUTBo9YZB1:18499:0:99999:7:::
```

Groups in /etc/group

- User’s primary group in /etc/passwd (often *users*, *staff*, or personal group)
- Secondary group memberships in /etc/groups



Admin users

All users

Groups for the interactive user on the console

User’s personal group

```
sudo:x:27:bob,carol
games:x:60:
users:x:100:
cdrom:x:24:bob
audio:x:29:bob
alice:x:1001:alice
bob:x:1002:bob
carol:x:1003:carol
david:x:1004:david
emily:x:1005:emily
project:x:1006:bob,carol,david
```

Groupname

Group identifier GID

Secondary members

> In Linux, user can belong to many groups at the same time, and only superuser can add groups and members. Other Unix OSs may be different.

Superuser

- The superuser is a special privileged principal with UID zero and the user name root
- All security checks are turned off for the superuser
  - Only minor limitations:
    - The superuser cannot write to a read-only file system but can remount it as writeable
    - The superuser cannot decrypt passwords (because they are hash values) but can reset them
- The superuser can become any other user

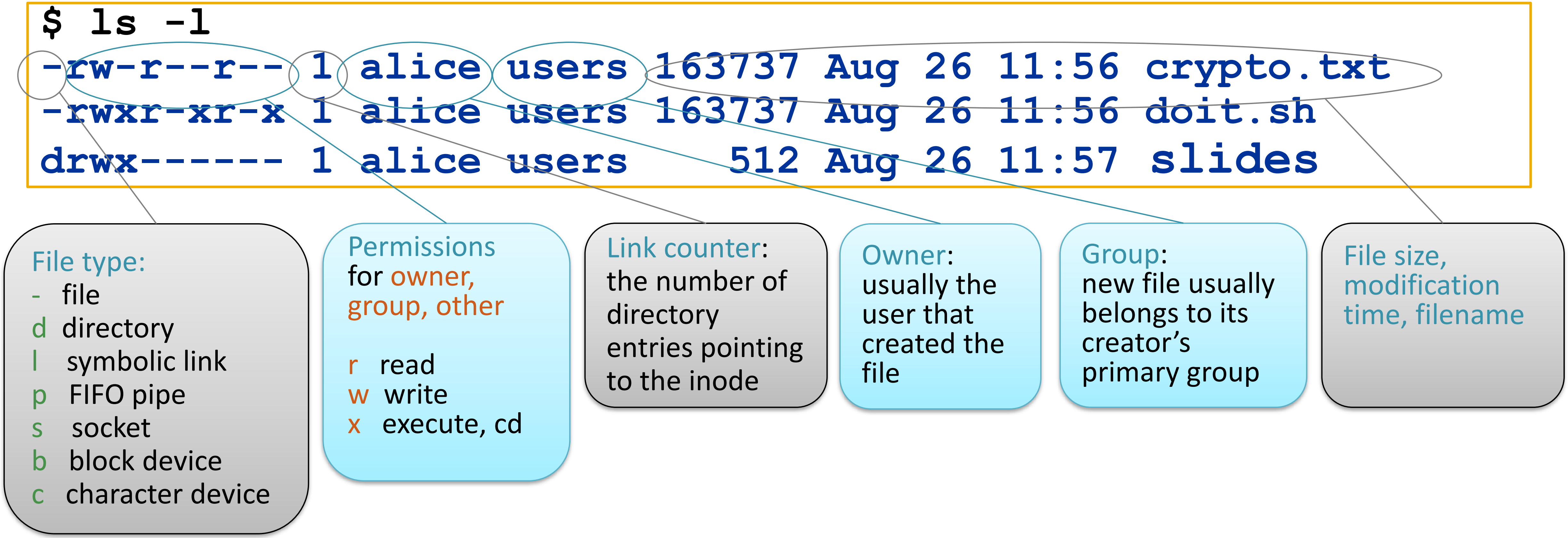
Subjects

- Subjects in Unix are processes with a process ID (PID)
  - Processes can create new processes with fork(2)
- Processes have a real UID/GID
  - Inherited from the parent process
  - Typically the UID/GID of the user logged in
- Processes also have an effective UID/GID

Objects

- Objects of access control are files, directories and devices
  - Organized in a tree-structured file system
- Inode data structure stores information about the object's owner user and group, and permissions
- Directory is a file containing file names and pointers to inodes
  - Filename is stored in the directory, not in inode, so that many names can be linked to the same inode

File metadata



```
$ ls -l
-rw-r--r-- 1 alice users 163737 Aug 26 11:56 crypto.txt
-rwxr-xr-x 1 alice users 163737 Aug 26 11:56 doit.sh
drwx------ 1 alice users    512 Aug 26 11:57 slides
```

File type:
- file
d directory
l symbolic link
p FIFO pipe
s socket
b block device
c character device

Permissions for owner, group, other

r read
w write
x execute, cd

Link counter: the number of directory entries pointing to the inode

Owner: usually the user that created the file

Group: new file usually belongs to its creator’s primary group

File size, modification time, filename

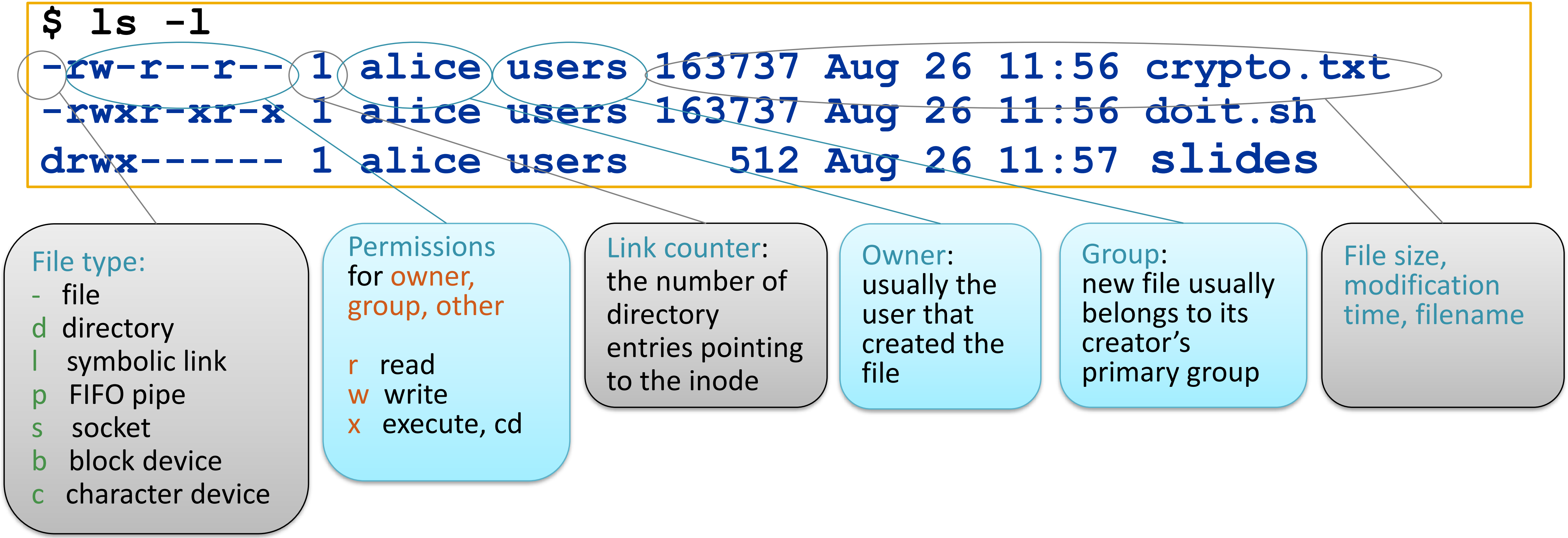
- Owner and root can change permissions (`chmod`);
  root can change the file owner and group (`chown`)
- Owner can change the file group to of one of their own groups (`chgrp`)

File permissions

- Permission bits are grouped in three triples that define read, write, and execute access for owner, group, and other (i.e. world)

- **`rw-r--r--`** read and write access for owner, read access for group and other
- **`rwxr-xr-x`** read, write and execute access for owner, read and execute for other
- **`rwx------`** read, write and execute (cd) access for owner only
  - For directory, **`x`** means ability to cd to the directory

File metadata



```
$ ls -l
-rw-r--r-- 1 alice users 163737 Aug 26 11:56 crypto.txt
-rwxr-xr-x 1 alice users 163737 Aug 26 11:56 doit.sh
drwx------ 1 alice users    512 Aug 26 11:57 slides
```

File type:
- - file
- d directory
- l symbolic link
- p FIFO pipe
- s socket
- b block device
- c character device

Permissions for owner, group, other

- r read
- w write
- x execute, cd

Link counter: the number of directory entries pointing to the inode

Owner: usually the user that created the file

Group: new file usually belongs to its creator’s primary group

File size, modification time, filename

- Owner and root can change permissions (`chmod`); root can change the file owner and group (`chown`)
- Owner can change the file group to of one of their own groups (`chgrp`)

Access control decisions

- Access control uses the effective UID/GID:
  - If the process’s effective UID = file owner, check owner permissions
  - Otherwise, if process’s effective GID = file owner, check group permissions
  - Otherwise, check other (=world) permissions
- The owner can always change the permissions (discretionary access control)

SUID and SGID programs

- Controlled invocation: normal user can invoke operations that require superuser privileges, in a controlled way
- SUID programs run with effective UID of the owner of the executable file
  - Execute permission of the *owner* is shown as **`s`** instead of **`x`**:

```
-rwsr-xr-x 1 root root 68208 May 28 09:37 /usr/bin/passwd
```

- SGID programs run with effective GID of the owner of the executable file
  - Execute permission of the *group* is shown as **`s`** instead of **`x`**

Example

Process	Real UID	Effective UID	Real GID	Effective GID
/bin/login	root	root	root	root

User alice logs in; the login process verifies the password and, with its superuser rights, changes its UID and GID by calling setuid(2), setguid(2):

/bin/login	alice	alice	users	users

The login process executes the user’s login shell:

/usr/bin/bash	alice	alice	users	users

From the shell, the user executes a command, e.g. **ls**

/bin/ls	alice	alice	users	users

The user executes command **passwd** to change his password:

/usr/bin/passwd	alice	root	staff	staff

SUID to root

- When root owns an executable file and the SUID bit is set, the process will get superuser status during execution
- Important SUID programs:

`/usr/bin/passwd`	change password
`/usr/bin/at`	batch job submission
`/usr/bin/sudo`	execute with another UID

- Only processes running as root can listen at the privileged TCP or UDP ports 0..1023
- SUID programs need to be written very carefully so that they only do what is intended (e.g. no injection attacks or buffer overruns)

- Principle of least privilege:
  - Define a new user and group for each service, e.g. print or database daemon
  - Process can use functions **`seteuid(2)`** and **`setegid(2)`** to toggle between privileged and non-privileged identifiers

Permissions for directories

- Read permission: to find which files are in the directory (ls)
- Write permission: to add files and delete (unlink) files
- Execute permission: to make the directory the current directory (cd) and for opening files inside the directory

- Every user has a home directory; what are the correct permissions for the home directory?

Special modes for a directory

- Sticky bit on an executable file historically indicated that the process should not be swapped to disk
- Sticky bit on a directory restricts the deletion of files in the directory only to the file owners (and root)
  - Job queues for printing etc. are implemented as a world-writable directories; anyone can add a file but not delete the files of others

```
ls -ld /tmp
```

- SGID bit on a directory means that new files inherit their group from the directory, not from the user who creates the file
  - Avoid running the print daemon as root: create a special group for the print daemon process and the print queue directory
  - Implement project directory where members can share files

Octal representation

- File permissions can also be specified as octal numbers
- Examples: **`rw-r--r--`** is equivalent to 644; **`rwxrwxrwx`** is equivalent to 777
- Conversion table:

0040 read by group
0020 write by group
0010 execute by group
0004 read by other
0002 write by other
0001 execute by other

4000 set UID on execution
2000 set GID on execution
1000 set sticky bit
0400 read by owner
0200 write by owner
0100 execute by owner

Default permissions: umask

- Unix utilities typically use default permissions 666 for a new data file and 777 for a new executable file
- Permissions can be restricted with umask: a three-digit octal number specifying the rights that should be withheld

  File permissions = default permissions AND (NOT umask)
- Sensible umask values:
  - 022: all permissions for owner, read and execute for group and other
  - 037: all permissions for owner, read permission for group, none for other
  - 077: all permissions for owner, none for group and other
- Example: default permissions 666, umask 077 → permissions for new file 0600

POSIX standard

- Unix permissions were standardized by IEEE as part of the POSIX standards (DOI 10.1109/IEEESTD.1992.106983)
  - Fairly universal across Unix systems
- Unix versions have subtle differences

Unix permissions - discussion

- Limitations and advantages?
  - Files have only one owner and one group
  - Complex policies, e.g. access to several groups, are impractical to implement
  - Superuser needed for maintaining groups
  - All access rights (e.g. shutdown, create user) must be mapped to file access and to read, write and execute permissions
  - Relatively simple and widely understood
  - Relatively easy to check the protection state
- Are Unix file permissions a kind of ACL?

OTHER SYSTEMS

Access control lists in Unix

Most Unix systems support one of:

- Posix ACL (see POSIX 1003.1e Draft 17)
  - – Standardization abandoned but widely implemented
- Network File System NFSv4 ACL (RFC 3550, section 5.11)
  - – Similar to Windows ACL with minor differences

- Linux:
  - – File systems (EXT3, EXT4, XFS) support POSIX ACL
  - – NFS client and server map between POSIX and NFSv4 ACLs

ACLs in Mac OS X

- Mac OS X implements both POSIX permissions and NFSv4 ACL
  - Similar to Windows ACLs
- Most files and folders only have permissions and no ACL; when you give permissions to more than one user or group, the ACL is created
- Negative entries are used to prevent accidental deletion of important folders:

```
$ ls -led Movies
drwx------+ 2 aura  staff  68 14 Syy 20:45 Movies
 0: group:everyone deny delete

$ rm -r Movies
rm: Movies: Permission denied

$ chmod +a# 0 "aura allow delete" Movies
$ rm -r Movies
```

Security-Enhanced Linux (SELinux)

Extra material

- Mandatory access control for Linux servers
  - Originally developed by NSA; goal is security certification
- Policy that cannot be changed by users
  - Modular policy language; complex policy definitions
  - Policy is compiled and installed by admin, not changed often

SELinux access-control model

Extra material

Hybrid access-control model that combines:

1. Type enforcement:
   subjects and objects have a type; rules based on the type
2. Multi-level security (e.g. Bell-LaPadula):
   level = sensitivity + category
3. Role-based access control (RBAC) – rarely used

- Each subject and object is labelled with security context:
  `user:role:`**`type:level`**

SELinux discussion

Extra material

- SELinux is mainly used to lock down a server in case it has software vulnerabilities or Trojans
  - Types or categories are used for sandboxing processes
- Developing and maintaining a new SELinux policy is a lot of work!
  - Original NSA reference policy is not usable
  - Tresys reference policy has commercial support
  - Red Hat Enterprise Linux tries to provide policies for common services
  - There are tools for policy validation and for learning policies by observing the system in operation
- Android uses SELinux for sandboxing system components

Mobile Operating Systems: Android, iOS

Extra material

- Subjects are apps, objects are phone features and services
- Apps are given permissions (which are actually capabilities)
  - E.g. read contacts, send SMS, read camera roll, read GPS
  - User gives permissions at install time or on demand
  - By default, apps can only access Internet and their own data
- Apps are isolated from each other
  - Very limited data sharing between apps
  - Cloud storage like Google Drive is used instead of local file system
- Universal Windows Platform (UWP) copies these ideas

SUMMARY

List of key concepts

- Principal, object, process, run as, SID, access token, privilege, group, alias i.e. local group
- NTFS, Registry, Windows domain, DC, AD
- Security descriptor, DACL, ACE, positive and negative permissions, container, object, ACE inheritance
- UAC, restricted token, controlled invocation, Windows service, integrity level, zone identifier
- User, group, superuser i.e. root, process, real and effective UID and GID, inode, permissions, umask, SUID and SGID program, sticky bit
- Posix and NFSv4 ACL, SELinux

Reading material

- Dieter Gollmann: Computer Security, 2nd ed., chapter 6–7; 3rd ed. chapters 7–8
- Matt Bishop: Introduction to computer security, chapter 25
- Ross Anderson: Security Engineering, 2nd ed., chapter 4
- Online:
  - Windows Developer Reference, MSDN: https://msdn.microsoft.com/en-us/library/windows/desktop/aa374876.aspx
  - https://msdn.microsoft.com/en-us/library/windows/desktop/aa374702.aspx
  - Microsoft Windows Security, Russinovich et al., sample chapter on Protecting Objects, : https://www.microsoftpressstore.com/articles/article.aspx?p=2228450&seqNum=3
  - Wayne Pollock, Unix File and Directory Permissions and Modes https://wpollock.com/AUnix1/FilePermissions.htm

Problems to think about: Windows

- How could Unix file permissions be expressed with Windows ACLs?
- Assume Fred is a member of the group Lecturers. Who gets access to an object with the DACLs:
  1. [-,Fred,READ], [+, Lecturers,READ] ?
  2. [-, Lecturers,READ], [+,Fred,READ] ?
  3. [+,Fred,READ], [-, Lecturers,READ] ?

  (Note: Negative ACEs are rarely used in reality. It is also a trick question.)
- When a new object is created, what goes into its security descriptor?
- Access tokens are objects themselves. How does access control for the tokens work?
- What is the time-of-check-to-time-of-use (TOCTTOU) issue? Where does this create potential problems in the Windows file system?
- Changing permissions on a top-level folder in the NTFS file system is a slow operation. You can try by creating new user and giving it read access to **`C:\`** and all subfolders. This is actually a performance optimization. Explain why.

Problems to think about: Linux/Unix

- Create a subdirectory in your home directory and put a file abc.txt in this subdirectory. Set permission bits on the subdirectory so that the owner has only execute access. Try to
  - list the subdirectory, display the contents of abc.txt, create a copy of abc.txt in the subdirectory, make the subdirectory the current directory with cd
- Repeat the same experiment first with read permission and then with write permission on the subdirectory. Try to understand what you observe.
- Find out how permissions are used to protect files on a web server, a shared temp directory, print queue directory, or shared directory for a project group.
- Write and configure a SUID program in C that allows other users to write log messages to a file which they otherwise cannot access. What if there is a buffer overflow vulnerability or other bugs in your code?
- Consider a normal user mounting a disk (e.g. USB stick) that contains a SUID program. What threat does it pose, and how has it been solved in Linux?
- Devices in Unix are mapped to special files under /dev. How does Linux protect a terminal device (tty) from other users?
- Linux and other new Unix systems have a **`sudo`** group, and instead of logging in as root, the members of this group can use the **`sudo`** command to exercise superuser powers. In what sense is this similar to role-based access control?