Contents based on CSE 541, 03 Fall by Mike Swift and CS 526 Fall 13 by Ninghui Li.

CS 419 COMPUTER SECURITY

OS BASICS

MIDTERM

- Next Friday, 3/13
- In class exam, 3:20 to 4:40 PM
- No notes, computers, phones or digital devices allowed
- Multiple choices + Problem solving

WHAT SECURITY GOALS DOES OPERATING SYSTEM PROVIDE?

- Traditionally: enabling multiple users to securely share a computer
  - Separation and sharing of processes, memory, files, devices, etc.
- What is the threat model?
  - Users may be malicious, users have terminal access to computers, software may be malicious/buggy, and so on
- Security mechanisms
  - Memory protection
  - Processor modes
  - User authentication
  - File access control

WHAT SECURITY GOALS DOES OPERATING SYSTEM PROVIDE?

- Nowadays: ensure secure operation in networked environment
- What is the threat model?
- Security mechanisms
  - Authentication
  - Access Control
  - Secure Communication (using cryptography)
  - Logging & Auditing
  - Intrusion Prevention and Detection
  - Recovery

SAFE SHARING

- Protecting a single computer with one user is easy
  - Prevent everybody else from having access
  - Encrypt all data with a key only one person knows
- Sharing resources safely is hard
  - Preventing some people from reading private data (e.g. grades)
  - Prevent some people from using too many resources (e.g. disk space)
  - Prevent some people from interfering with other programs (e.g. inserting key strokes / modifying displays)

SECURITY IS ABOUT RECONCILING SEPARATION AND SHARING

- Ensure separation
  - Physical
  - Temporal
  - Logical
  - Cryptographical
- OS also need to ensure sharing

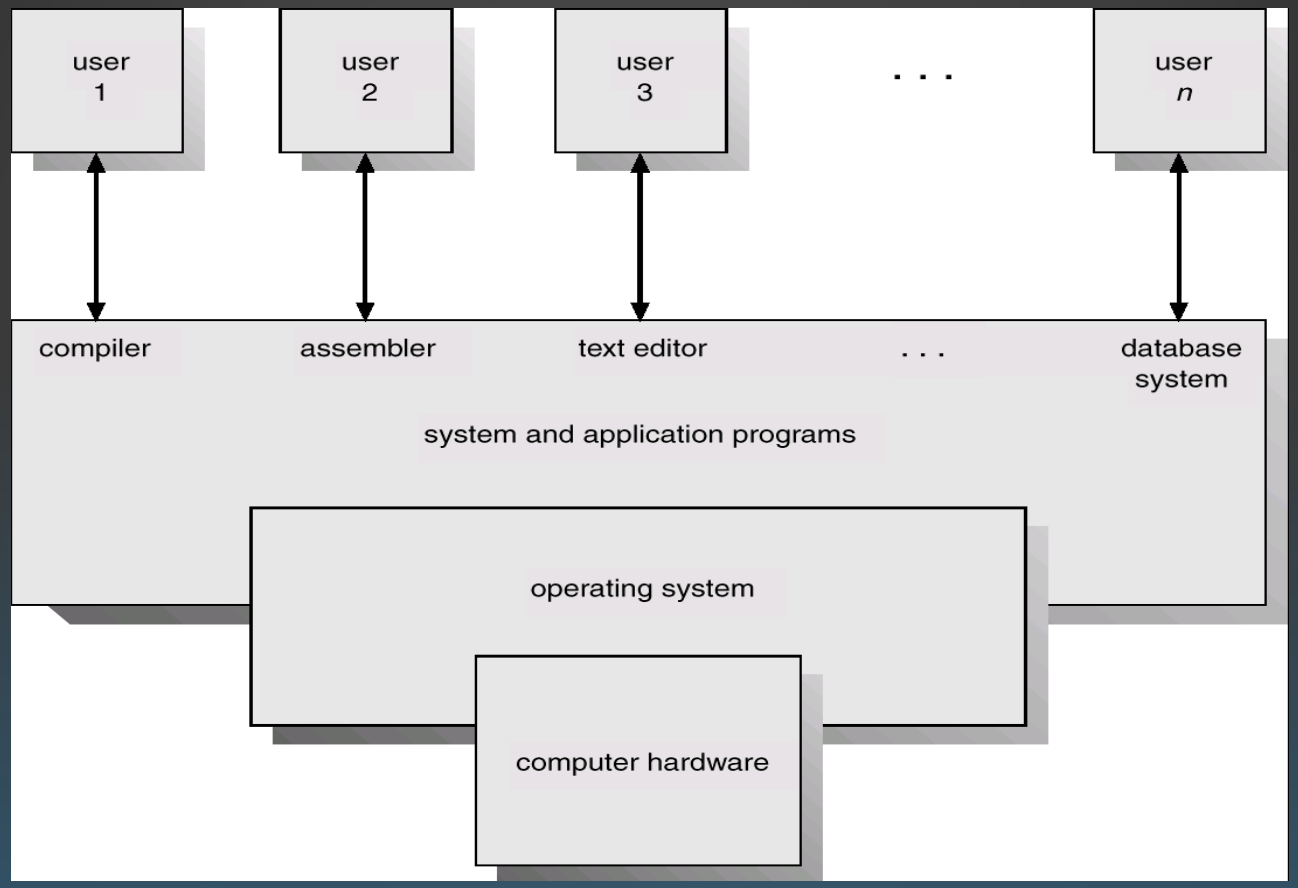
WHY IS SECURITY HARD?

- Security slows things down
- Security gets in the way
- Security adds no value if there are no attacks
- Only the government used to pay for security
  - The Internet made us all potential victims

COMPUTER SYSTEM COMPONENTS

- Hardware
  - Provides basic computing resources (CPU, memory, I/O devices).
- Operating system
  - Controls and coordinates the use of the hardware among the various application programs.
- Applications programs
  - Define the ways in which the system resources are used to solve the computing problems of the users.
- Users
  - E.g., people, machines, other computers.

ABSTRACT VIEW OF SYSTEM COMPONENTS



MEMORY PROTECTION: ACCESS CONTROL TO MEMORY

- Ensures that one user's process cannot access other's memory
  - fence
  - relocation
  - base/bounds register
  - segmentation
  - paging
  - …
- Operating system and user processes need to have different privileges

CPU MODES (A.K.A. PROCESSOR MODES OR PRIVILEGE

- System mode (privileged mode, master mode, kernel mode)
  - Can execute any instruction
  - Can access any memory locations, e.g., accessing hardware devices,
  - Can enable and disable interrupts,
  - Can change privileged processor state,
  - Can access memory management units,
  - Can modify registers for various descriptor tables .

Reading: http://en.wikipedia.org/wiki/CPU_modes

USER MODE

- User mode
  - Access to memory is limited,
  - Cannot execute some instructions
  - Cannot disable interrupts,
  - Cannot change arbitrary processor state,
  - Cannot access memory management units
- Transition from user mode to system mode can only happen via well defined entry points, i.e., through system calls

Reading: http://en.wikipedia.org/wiki/CPU_modes

KERNEL SPACE VS USER SPACE

- Part of the OS runs in the kernel model
  - known as the OS kernel
- Other parts of the OS run in the user mode, including service programs (daemon programs), user applications, etc.
  - they run as processes
  - they form the user space (or the user land)
- What is the difference between kernel mode and processes running as root (or superuser, administrator)?

SYSTEM CALLS

- Guarded gates from user mode (space, land) into kernel mode (space, land)
  - use a special CPU instruction (often an interruption), transfers control to predefined entry point in more privileged code; allows the more privileged code to specify where it will be entered as well as important processor state at the time of entry.
  - the higher privileged code, by examining processor state set by the less privileged code and/or its stack, determines what is being requested and whether to allow it.

http://en.wikipedia.org/wiki/System_call

TRUSTED COMPUTING BASE (TCB)

- Think carefully about what you are trusting with your information
  - if you type your password on a keyboard, you're trusting:
    - the keyboard manufacturer
    - your computer manufacturer
    - your operating system
    - the password library
    - the application that's checking the password
  - TCB = set of components (hardware, software, wetware) that you trust your secrets with
- Public web kiosks should *not* be in your TCB
  - should your OS?
    - but what if it is promiscuous? (e.g., IE and active-X extensions)
  - how about your compiler?
    - A great read: "Reflections on Trusting Trust".

XcodeGhost

From Wikipedia, the free encyclopedia

XcodeGhost (and variant XcodeGhost S) are modified versions of Apple's Xcode development environment that are considered malware.[1] The software first gained widespread attention in September 2015, when a number of apps originating from China harbored the malicious code.[2] It was thought to be the "first large-scale attack on Apple's App Store",[3] according to the BBC. The problems were first identified by researchers at Alibaba, a leading e-commerce firm in China.[4] Over 4000 apps are infected, according to FireEye, far more than the 25 initially acknowledged by Apple,[5] including apps from authors outside China.

Security firm Palo Alto Networks surmised that because network speeds were slower in China, developers in the country looked for local copies of the Apple Xcode development environment, and encountered altered versions that had been posted on domestic web sites. This opened the door for the malware to be inserted into high profile apps used on iOS devices.[6][7]

Even two months after the initial reports, security firm FireEye reported that hundreds of enterprises were still using infected apps and that XcodeGhost remained "a persistent security risk".[8][9] The firm also identified a new variant of the malware and dubbed it XcodeGhost S; among the apps that were infected were the popular messaging app WeChat and a Netease app Music 163.[10]

AUTHORIZATION

	Alice	Bob	Carl
/etc	Read	Read	Read Write
/homes	Read Write	Read Write	Read Write
/usr	None	None	Read

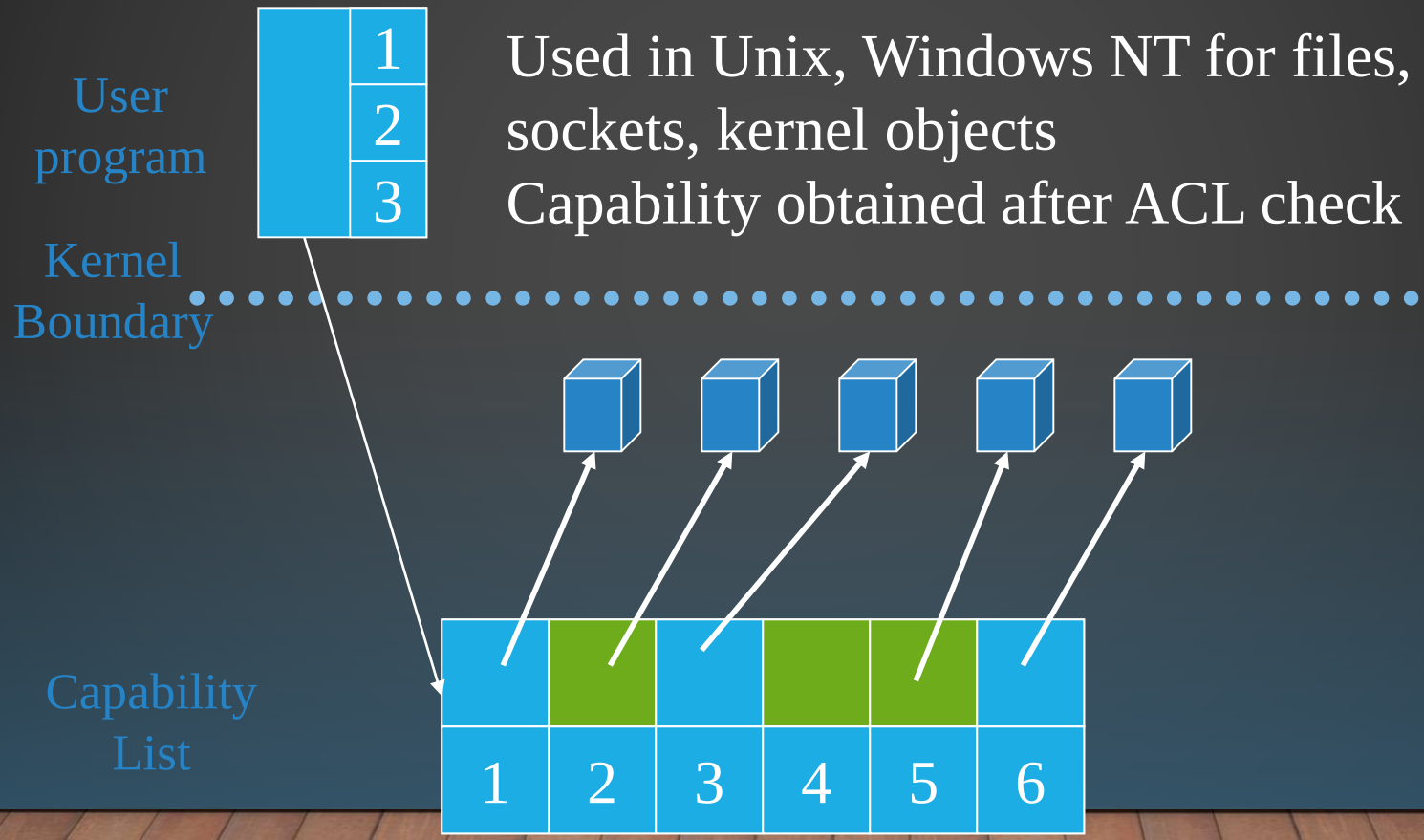
- How does the system know what I'm allowed to do?
  - Authorization matrix:
    - Objects = things that can be accessed
    - Subjects = things that can do the accessing (users or programs)
  - What are the limits?
    - Time of day
    - Ranges of values

ACCESS CONTROL LISTS

Bob	R/W/D
Students	Read
Everyone	Read

- Representation used in Windows NT, Unix for files
- Stored on each file / directory
- Unix:
  - Fixed set of permissions (read,write,exe)
  - Three sets of subjects (owner, group, others)
- Windows NT
  - Arbitrary number of entries
  - 16 permissions per object

CAPABILITIES



Used in Unix, Windows NT for files, sockets, kernel objects
Capability obtained after ACL check

ACLs:

- Can have large numbers of objects
- Easy to grant access to many objects at once
- Require expensive operation on every access

Capabilities

- Hard to manage huge number of capabilities
- They have to come from somewhere
- They are fast to use (just pointer dereferences)

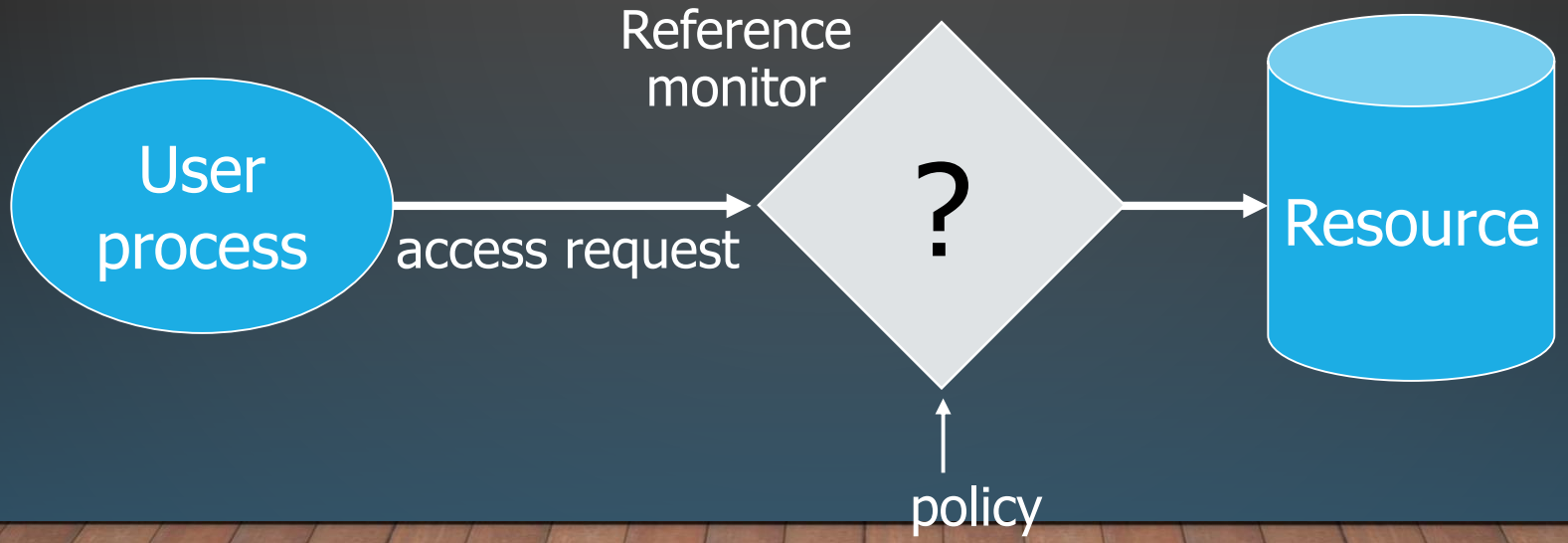
Most systems use both

- ACLs for opening an object (e.g. fopen())
- Capabilities for performing operations (e.g. read())

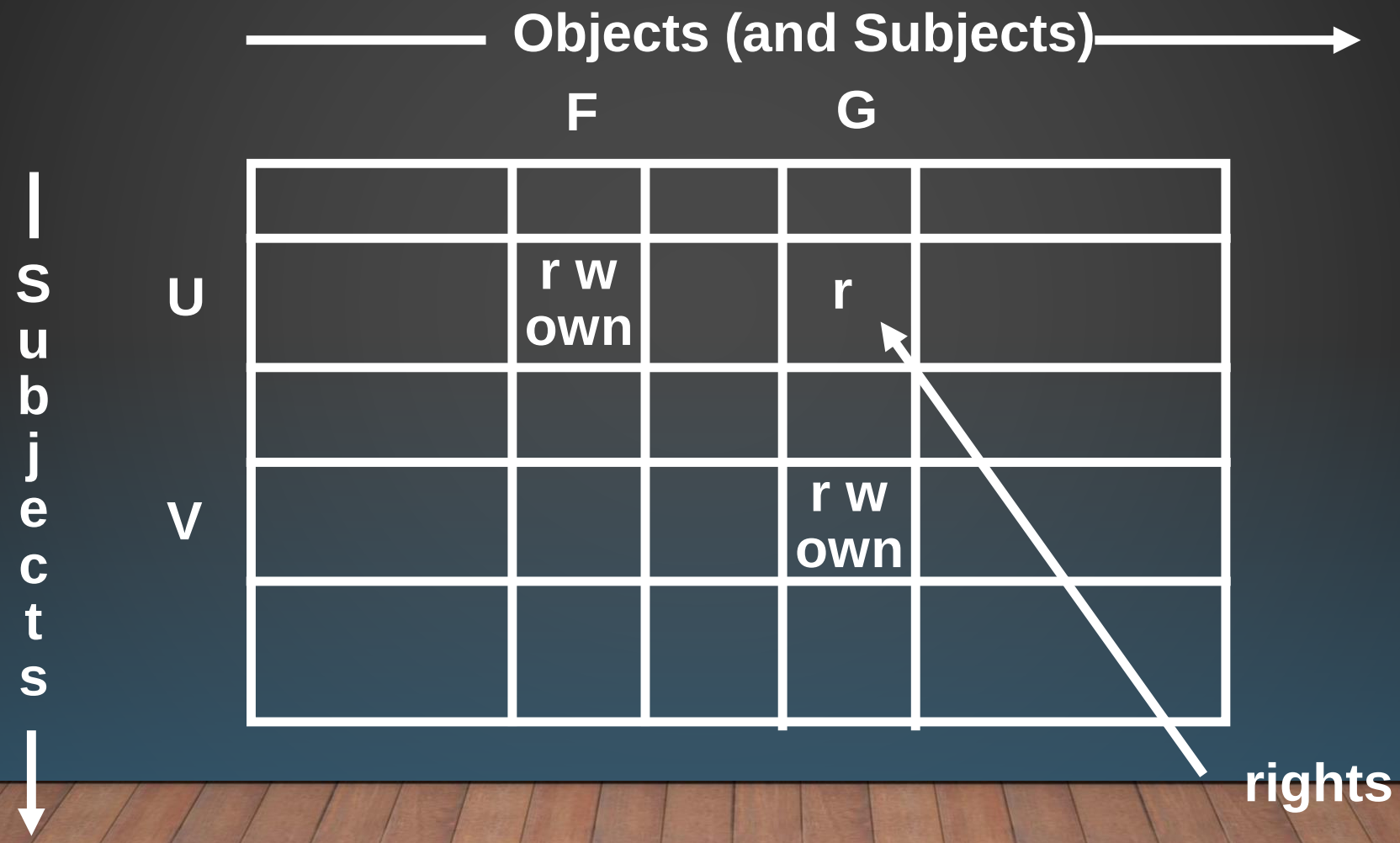
WHICH ONE IS BETTER

ACCESS CONTROL

- A reference monitor mediates all access to resources
  - Principle: Complete mediation: control **all** accesses to resources



ACCESS MATRIX MODEL



ACCESS MATRIX MODEL

- Basic Abstractions
  - Subjects
  - Objects
  - Rights
- The rights in a cell specify the access of the subject (row) to the object (column)

BASIC CONCEPTS OF UNIX ACCESS CONTROL: USERS, GROUPS, FILES, PROCESSES

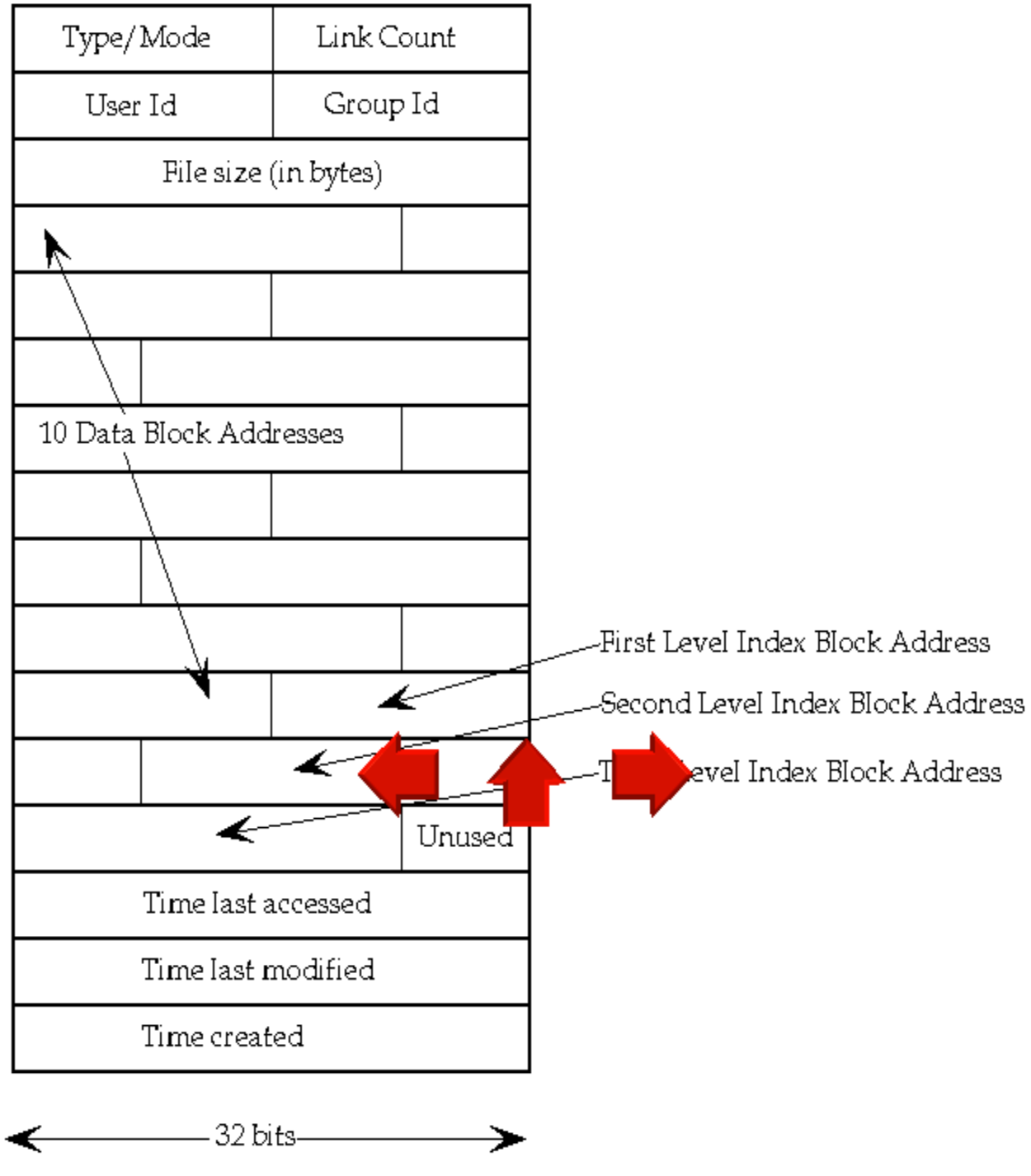
- Each user account has a unique UID
  - The UID 0 means the super user (system admin)
- A user account belongs to multiple groups
- Subjects are processes
  - associated with uid/gid pairs, e.g., (euid, egid), (ruid, rgid), (suid, sgid)
- Objects are files

ORGANIZATION OF OBJECTS

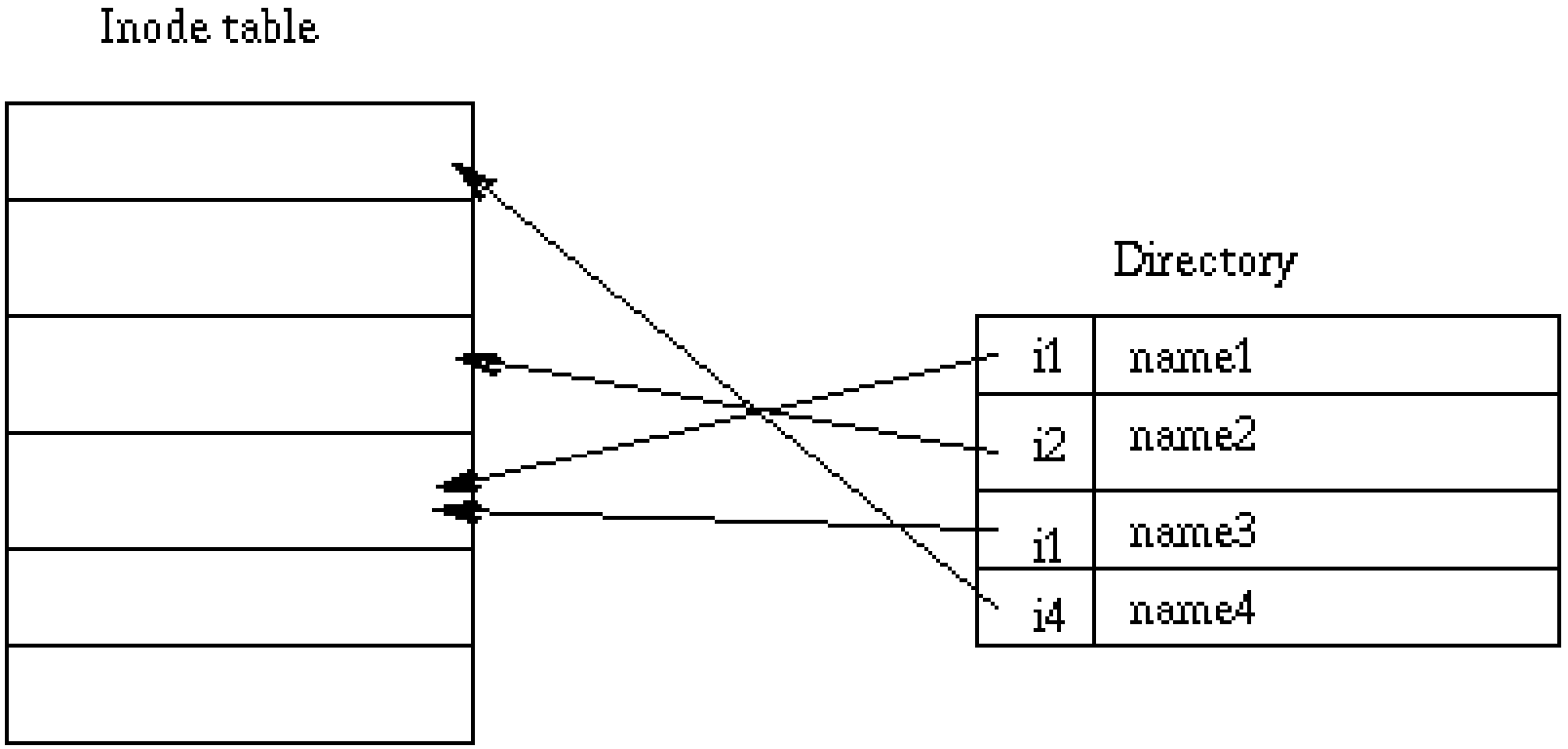
- Almost all objects are modeled as files
  - Files are arranged in a hierarchy
  - Files exist in directories
  - Directories are also one kind of files
- Each object has
  - owner
  - group
  - 12 permission bits
    - rwx for owner, rwx for group, and rwx for others
    - suid, sgid, sticky

UNIX inodes:

Each file corresponds to an inode



UNIX Directories



BASIC PERMISSIONS BITS ON FILES (NON-DIRECTORIES)

01

Read controls reading the content of a file

- i.e., the read system call

02

Write controls changing the content of a file

- i.e., the write system call

03

Execute controls loading the file in memory and execute

- i.e., the execve system call

EXECUTION OF A FILE

- Binary file vs. script file
- Having execute but not read, can one run a binary file?
- Having execute but not read, can one run a script file?
- Having read but not execute, can one run a script file?

PERMISSION BITS ON DIRECTORIES

- Read bit allows one to show file names in a directory
- The execution bit controls traversing a directory
  - does a lookup, allows one to find inode # from file name
  - chdir to a directory requires execution
- Write + execution control creating/deleting files in the directory
  - Deleting a file under a directory requires no permission on the file
- Accessing a file identified by a path name requires execution to all directories along the path

THE SUID, SGID, STICKY BITS

	suid	sgid	sticky bit
non-executable files	no effect	affect locking (unimportant for us)	not used anymore
executable files	change euid when executing the file	change egid when executing the file	not used anymore
directories	no effect	new files inherit group of the directory	only the owner of a file can delete

SOME EXAMPLES

- What permissions are needed to access a file/directory?
  - read a file: /d1/d2/f3
  - write a file: /d1/d2/f3
  - delete a file: /d1/d2/f3
  - rename a file: from /d1/d2/f3 to /d1/d2/f4
  - …
- File/Directory Access Control is by System Calls
  - e.g., open(2), stat(2), read(2), write(2), chmod(2), opendir(2), readdir(2), readlink(2), chdir(2), …

THE THREE SETS OF PERMISSION BITS

- Intuition:
  - if the user is the owner of a file, then the r/w/x bits for owner apply
  - otherwise, if the user belongs to the group the file belongs to, then the r/w/x bits for group apply
  - otherwise, the r/w/x bits for others apply

- Can one implement negative authorization, i.e., only members of a particular group are not allowed to access a file?

OTHER ISSUES ON OBJECTS IN UNIX

- Accesses other than read/write/execute
  - Who can change the permission bits?
    - The owner can
  - Who can change the owner?
    - Only the superuser
- Rights not related to a file
  - Affecting another process
  - Operations such as shutting down the system, mounting a new file system, listening on a low port
    - traditionally reserved for the root user

PROCESS USER ID MODEL IN MODERN UNIX SYSTEMS

- Each process has three user IDs
  - real user ID (ruid) — owner of the process
  - effective user ID (euid) — used in most access control decisions
  - saved user ID (suid)
- and three group IDs
  - real group ID
  - effective group ID
  - saved group ID

PROCESS USER ID MODEL IN MODERN UNIX SYSTEMS

- When a process is created by *fork*
  - it inherits all three users IDs from its parent process
- When a process executes a file by *exec*
  - it keeps its three user IDs unless the set-user-ID bit of the file is set, in which case the effective uid and saved uid are assigned the user ID of the owner of the file
- A process may change the user ids via system calls

THE NEED FOR SUID/SGID BITS

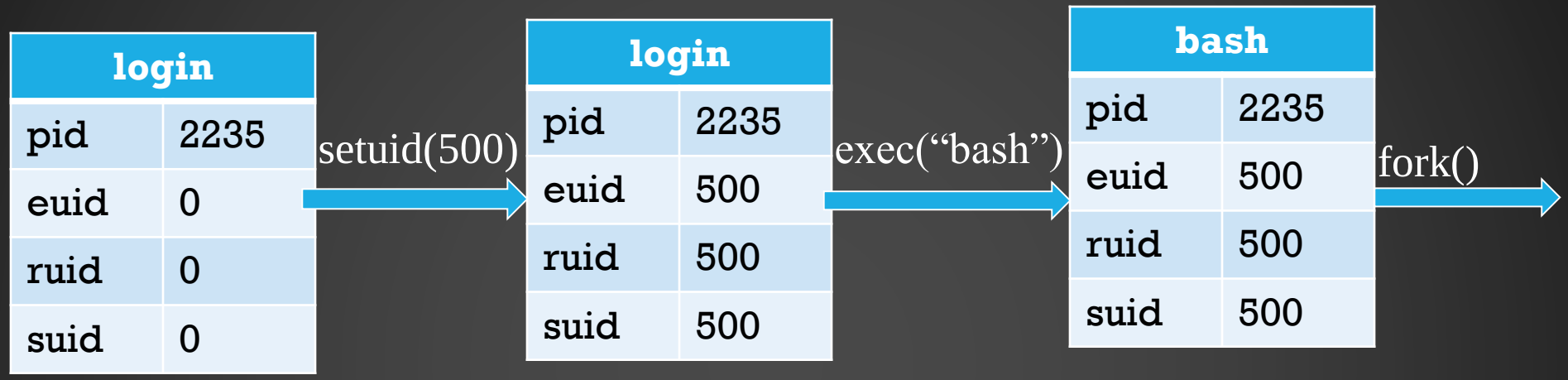
- Some operations are not modeled as files and require user id = 0
  - halting the system
  - bind/listen on "privileged ports" (TCP/UDP ports below 1024)
  - non-root users need these privileges
- File level access control is not fine-grained enough
- System integrity requires more than controlling who can write, but also how it is written

SECURITY PROBLEMS OF PROGRAMS WITH SUID/SGID

- These programs are typically setuid root
- Violates the least privilege principle
  - every program and every user should operate using the least privilege necessary to complete the job
- Why violating least privilege is bad?
- How would an attacker exploit this problem?
- How to solve this problem?

CHANGING EFFECTIVE USER IDS

- A process that executes a set-uid program can drop its privilege; it can
  - drop privilege permanently
    - removes the privileged user id from all three user IDs
  - drop privilege temporarily
    - removes the privileged user ID from its effective uid but stores it in its saved uid, later the process may restore privilege by restoring privileged user ID in its effective uid



After the login process verifies that the entered password is correct, it issues a setuid system call.

The login process then loads the shell, giving the user a login shell.

The user types in the passwd command to change his password.

bash	
pid	2235
euid	500
ruid	500
suid	500

bash	
pid	2297
euid	500
ruid	500
suid	500

exec("passwd")

passwd	
pid	2297
euid	0
ruid	500
suid	0

Drop privilege permanently

passwd	
pid	2297
euid	500
ruid	500
suid	500

Drop privilege temporarily

passwd	
pid	2297
euid	500
ruid	500
suid	0

The fork call creates a new process, which loads "passwd", which is owned by root user, and has setuid bit set.

ACCESS CONTROL IN EARLY UNIX

- A process has two user IDs: real uid and effective uid and one system call setuid
- The system call setuid(id)
  - when euid is 0, setuid set both the ruid and the euid to the parameter
  - otherwise, the setuid could only set effective uid to real uid
    - Permanently drops privileges
- A process cannot temporarily drop privilege

Setuid Demystified, In USENIX Security ‘ 02

SYSTEM V

- Added saved uid & a new system call
- The system call seteuid
  - if euid is 0, seteuid could set euid to any user ID
  - otherwise, could set euid to ruid or suid
    - Setting euid to ruid temp. drops privilege
- The system call setuid is also changed
  - if euid is 0, setuid functions as seteuid
  - otherwise, setuid sets all three user IDs to real uid

BSD

- Uses ruid & euid, change the system call from setuid to setreuid
  - if euid is 0, then the ruid and euid could be set to any user ID
  - otherwise, either the ruid or the euid could be set to value of the other one
    - enables a process to swap ruid & euid

MODERN UNIX

- System V & BSD affect each other, both implemented setuid, seteuid, setreuid, with different semantics
  - some modern UNIX introduced setresuid
- Things get messy, complicated, inconsistent, and buggy
  - POSIX standard, Solaris, FreeBSD, Linux

NEXT CLASS