Operating System Security

Monday, February 2 & Thursday, March 3

Reading: Saltzer & Schroeder (S&S) paper, S&M Ch. 4; Anderson Ch. 4

CS342 Computer Security

Department of Computer Science
Wellesley College

The Big Picture

We've studied some details of authentication and access control in Linux.

But let's step back and consider more fundamental issues:

- What is an operating system (OS)?
- What are OS security issues?
- What are other approaches to access control other than Linux approach?
  - access control lists
  - capabilities

What's an Operating System (OS) For?

- Manage resources and abstract over their low-level details:
  - processes and memory
  - reading/writing files
  - interacting with input devices (keyboards, touchpads, touch screens, camera, microphones) and output devices (screens, speakers, vibration motors)
- Running multiple processes of multiple users on the same machine:
  - illusion that each user/process has access to whole machine
  - process isolation: protect one process from attacks (accidental or malicious) by another.
  - efficiency: while one process is blocked, can run another.
  - convenience: user/system can run multiple programs at once.
  - communication/synchronization between concurrent processes
- *Note:* don't confuse OS user interface (shell or GUI) with the underlying kernel services it provides.

What's a Computer? What's a Process?

From S&M

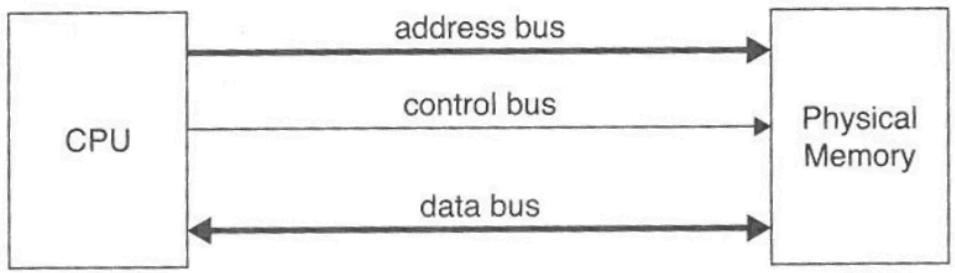


Figure 4.1 In a basic model of physical computer architecture, the CPU reads and writes memory locations via its buses.

process = state of a running program:

- program counter (PC)
- register contents
- logical (virtual) address space, including:
  - stack of activation frames
  - heap of longer-lived data
  - other data (read-only strings, code)

Saltzer and Schroeder 1975 paper (S&S)

The Protection of Information in Computer Systems

JEROME H. SALTZER, SENIOR MEMBER, IEEE, AND
MICHAEL D. SCHROEDER, MEMBER, IEEE

About this paper

Manuscript received October 11, 1974; revised April 17. 1975.

Fourth ACM Symposium on Operating System Principles (October 1973).
Revised version in *Communications of the ACM* 17, 7 (July 1974).

Original web version created by Norman Hardy.

Invited Paper

S&S Guarded Door Model for Information Protection

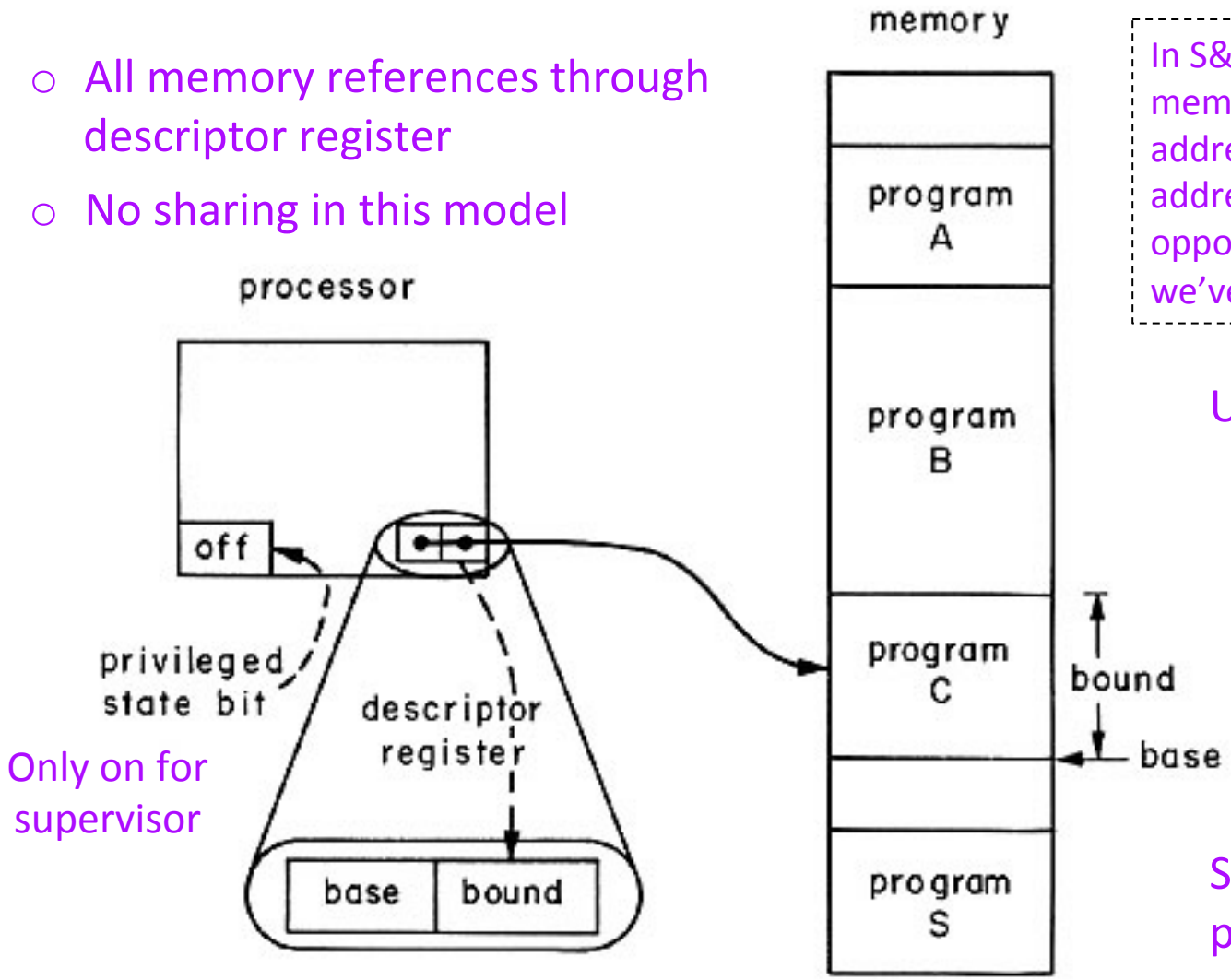
1. Build an impenetrable wall around each object requiring separate protection

2. Construct a door through wall through which access can be obtained

3. Post guard at door who checks authorization by comparing something guard knows with something user possesses

Process Isolation (Saltzer & Schroeder 1975, Fig 1)

- All memory references through descriptor register
- No sharing in this model



In S&S, S&M, and these slides, memory grows from low addresses at bottom to high addresses at top, which is the opposite of the convention we've used so far

User programs

Only on for supervisor

Supervisor (kernel) program

Only changeable by supervisor

Three Guarded Doors in S&S Fig 1

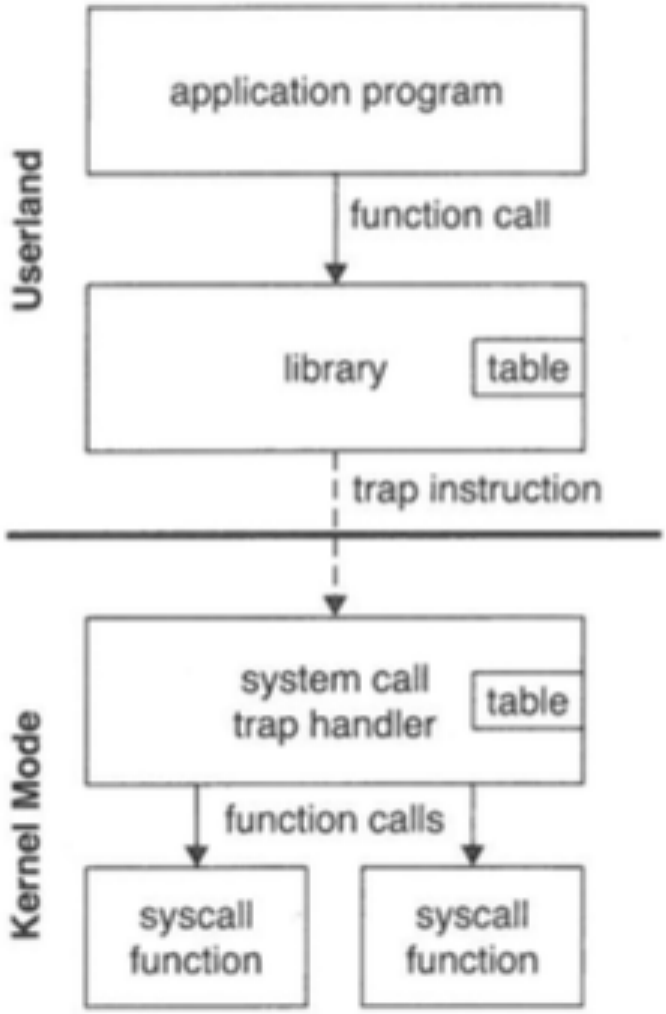
1. Base/bound descriptor register used for protected memory access

2. Descriptor register changeable only if privileged state bit is on

3. State bit is on only when control transferred to supervisor program, which turns it off when transfers back.

System Call/Context Switch

S&M Fig. 4.4



Userland code requests OS service through system call (syscall).

- Put values indicating desired service and any parameters in agreed-upon location (e.g., particular registers)
- Issue system call trap instruction
- Context switch in
  - save process PC, registers, memory (stack, heap, etc.), and other bookkeeping data
  - switch from user mode to kernel mode
  - trap handler processes request
- Context switch out
  - puts result(s) (if any) in agreed-upon location
  - restore userland process (except result locations)
  - switch from kernel mode back to user mode

Intel x86 System Calls

- Put system call number in register %eax
- Put other arguments in other registers
- Execute instruction `int $0x80`
- Results (if any) in %eax or specified by system call

System call documentation:

http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html

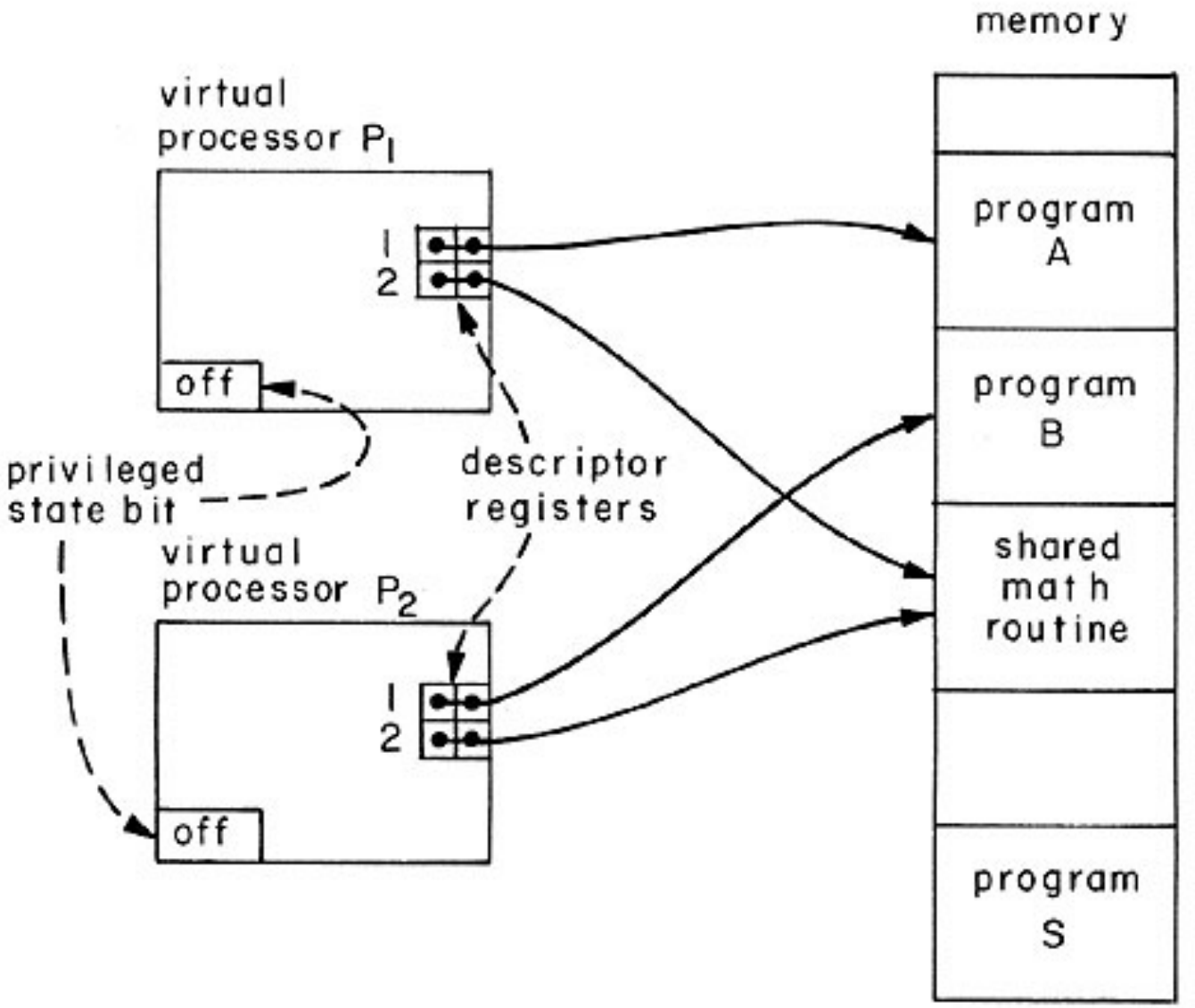
http://man7.org/linux/man-pages/man2/syscalls.2.html

You can track system calls of a Linux executabel with the `strace` command.
This will be the topic of Lab 6 this week.

Protection Levels

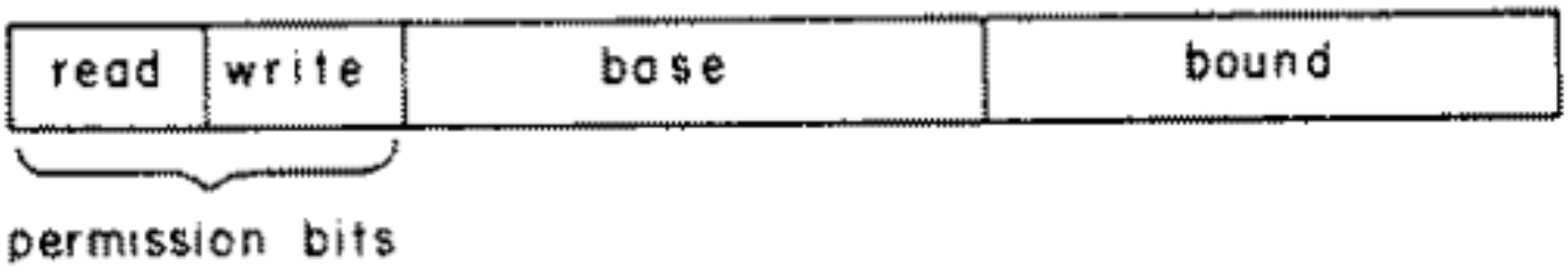
- Many operating systems assign security levels to processes.
- Simplest version has two levels:
  - **supervisor/kernel/privileged mode** (S&S privileged state bit on) can do anything. Implements protection/isolation mechanisms.
    - ✓ file/directory reading/writing (including permissions)
    - ✓ control memory management unit (MMU), which implements logical (virtual) memory address spaces in physical memory.
    - ✓ manage bookkeeping data for multiple processes
  - **user/unprivileged mode** (S&M userland, S&S privileged state bit off): everything else
- Can have many more levels/rings. E.g. x386 and above have 4 rings, where 0 = kenel and 3 = user.

Sharing between processes (S&S, Fig 3)

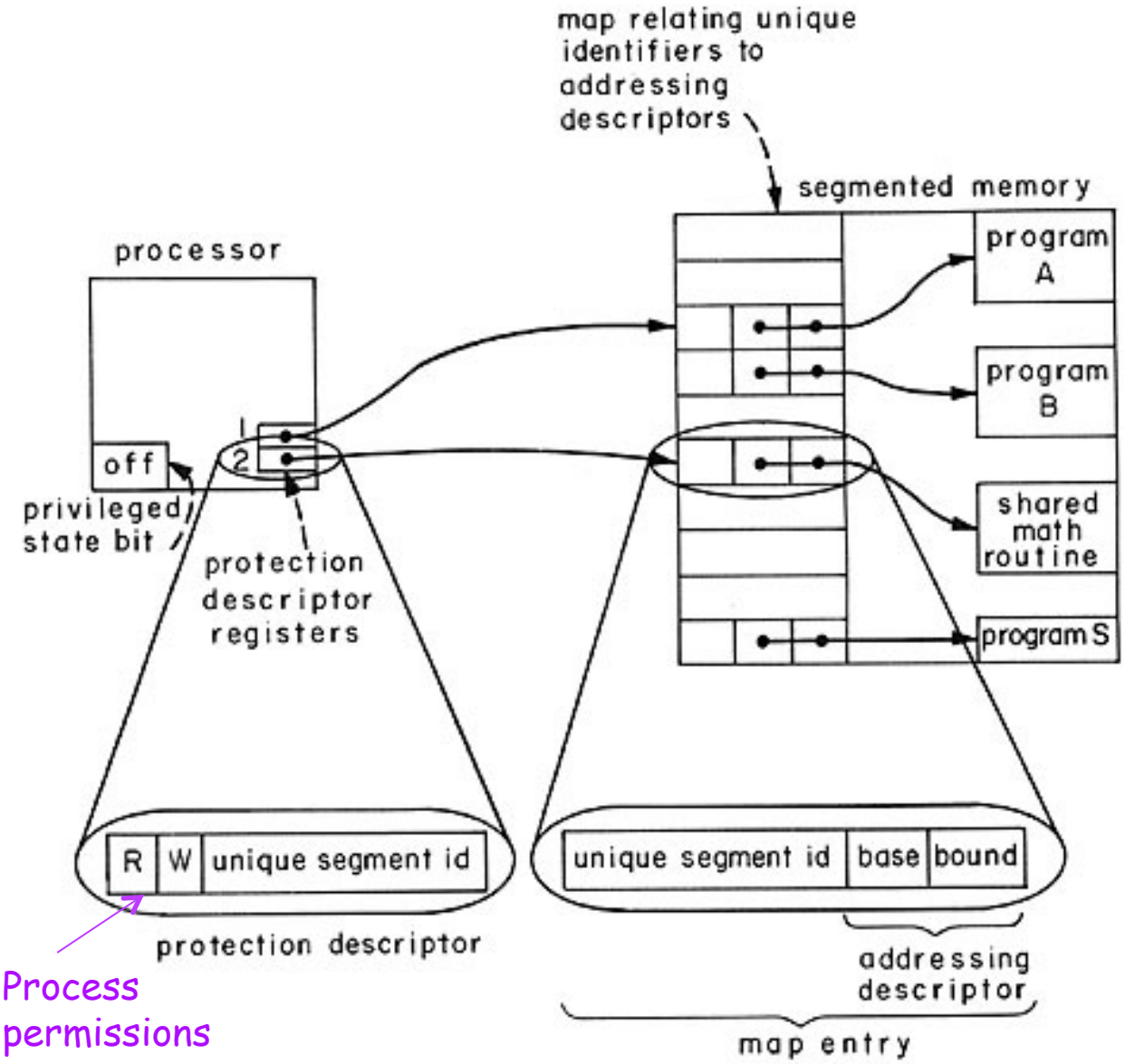


E.g., two Ubuntu users running emacs at same time can share its code

Process permissions (S&S, Fig 4)



Protection vs. addressing descriptors (S&S Fig 5)



Exploits on Processes

- Ability to rewrite kernel bookkeeping memory can give one process access to another.
- Means of interprocess communication (IPC), e.g. pipes, message passing, pokes holes in isolation walls that can be basis for exploits.
- Object reuse (page frames, malloc blocks, file blocks) can leak information.
- OSs are large/complex, so hard to get everything "right" $\Rightarrow$ there are bugs to exploit.

Saltzer & Schroeder's 1975 Design Principles for Security (as interpreted by S&M)

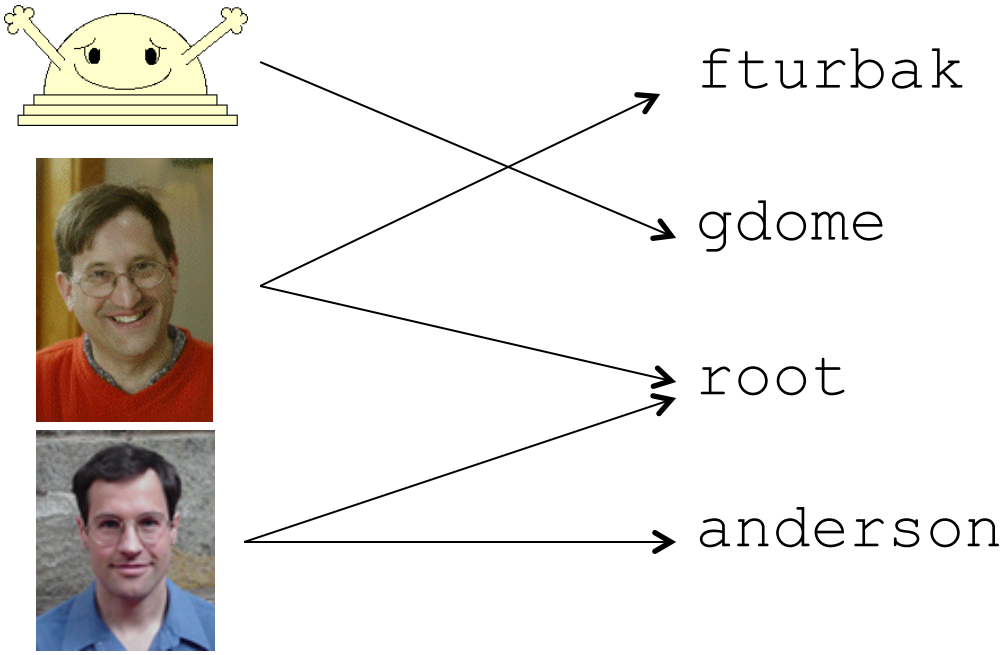
Economy of mechanism	Keep the design simple. Complexity breeds vulnerabilities
Fail-safe defaults	Fail the right way. Denial of service is often preferable to more substantial system compromise
Complete mediation	Check each access. If you don't check, how can you be sure?
Open design	No security via obscurity. Kerckhoff's Principle applies to system design, too.
Separation of privilege	Require the adversary to break multiple components
Least privilege	The more privileges you have, the more damage you can do.
Least common mechanism	You wouldn't share a toothbrush, so why share your data files?
Psychological acceptability	If users can't understand the system, they won't use it correctly

These principles may be old, but still valuable!

Authentication and Access Control

Often view system in terms of subjects/principals, objects, and operations:

- **Authentication** binds people/processes to subjects.



- **Access control** specifies what subjects can do with certain objects

Who Are Subjects (S&S Principals) ?

- subject = single human (e.g., account for individual user)
- subject = multiple humans (e.g., guest account, root, groups)
- subject = system or program (e.g. apache)
- a single human may be multiple subjects (personal account, course staff, webmistress, root, ...)

Access Control Matrix

Objects

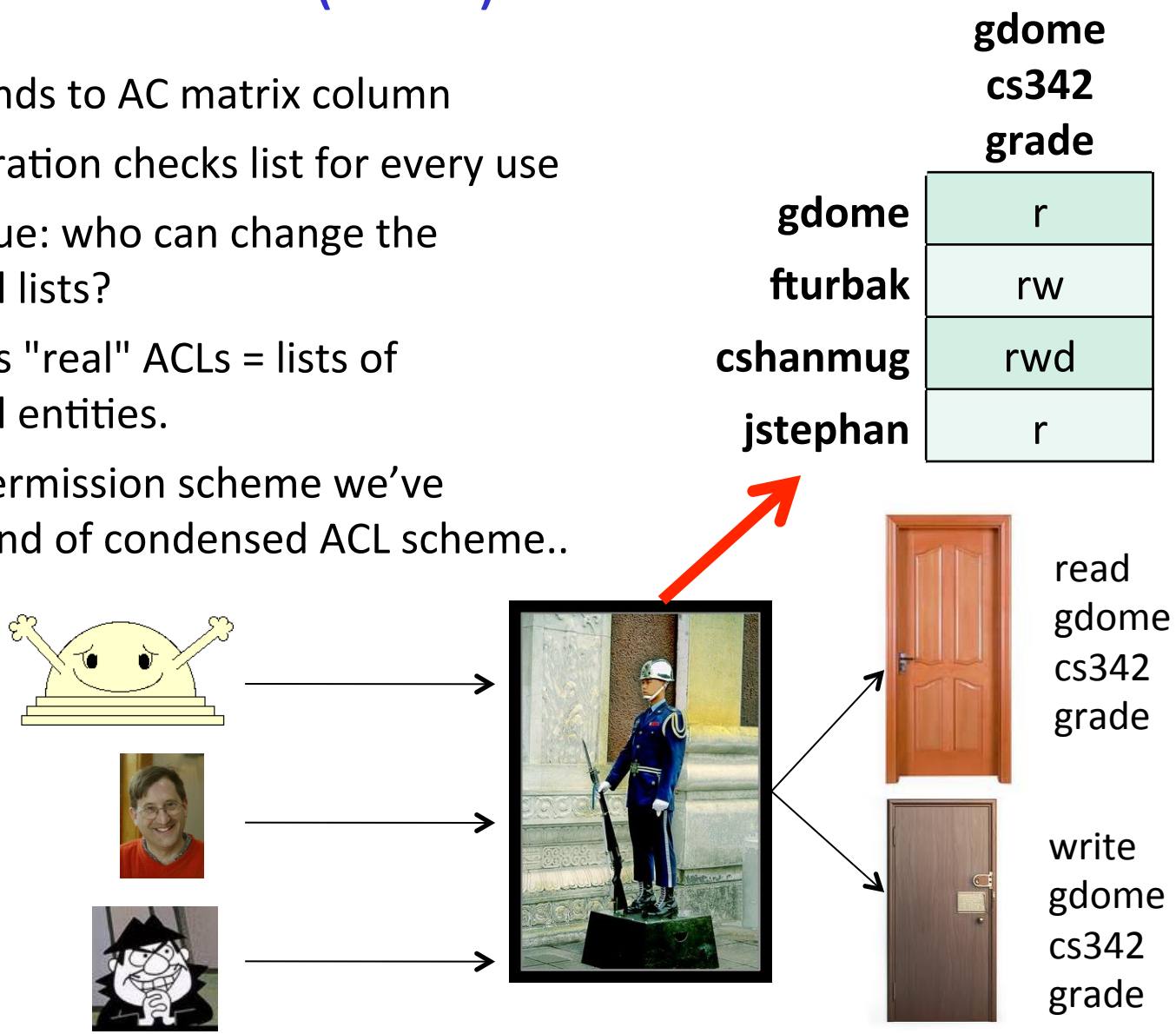
Subjects/Principals	**cs342 ps1**	**crypto program**	**gdome cs342 grade**	**fbar cs342 grade**	**CSDept blog**
gdome	r	rx	r		ra
fbar	r			r	ra
fturbak	rwd	rwx	rw	rw	ra
cshanmug			rwd	rwd	r
jstephan			r	r	r
rpurcell					rwd

Permissions:
r = read
w = write
x = execute
a = append
d = delete

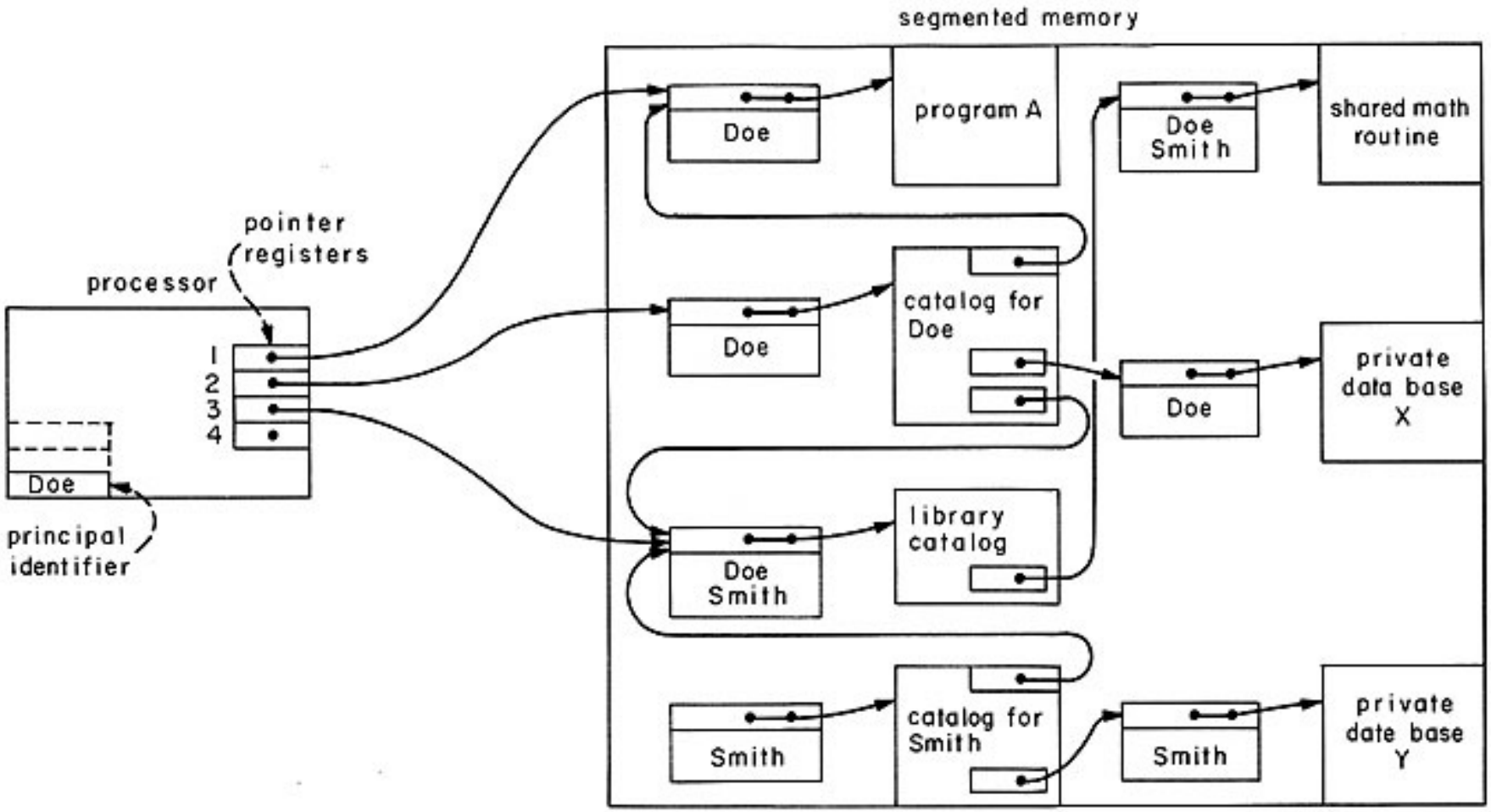
In general, want more permissions than the rwx provided by Linux

Access Control Lists (ACLs)

- ACL corresponds to AC matrix column
- Guard of operation checks list for every use
- Important issue: who can change the access control lists?
- Windows uses "real" ACLs = lists of access control entities.
- The Linux's permission scheme we've studied is a kind of condensed ACL scheme..

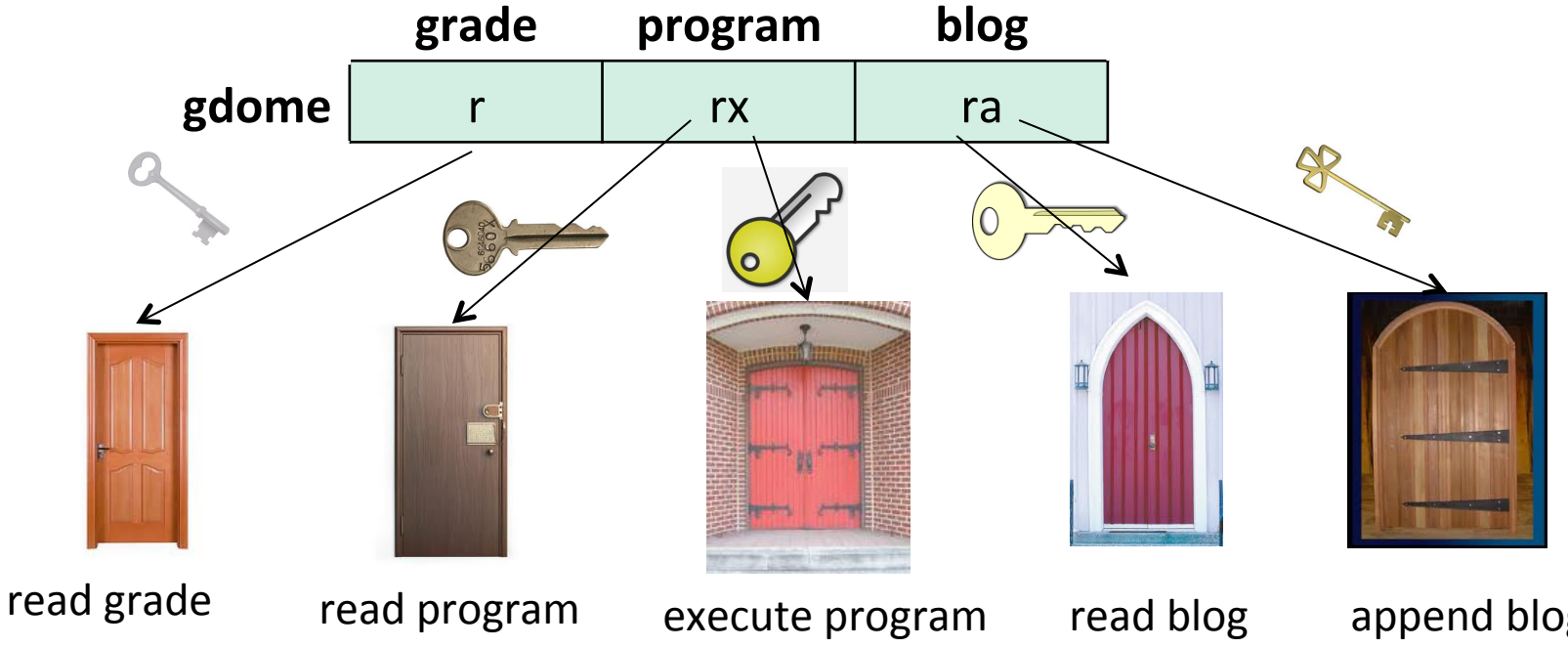


ACLs in Action (S&S, Fig 10)

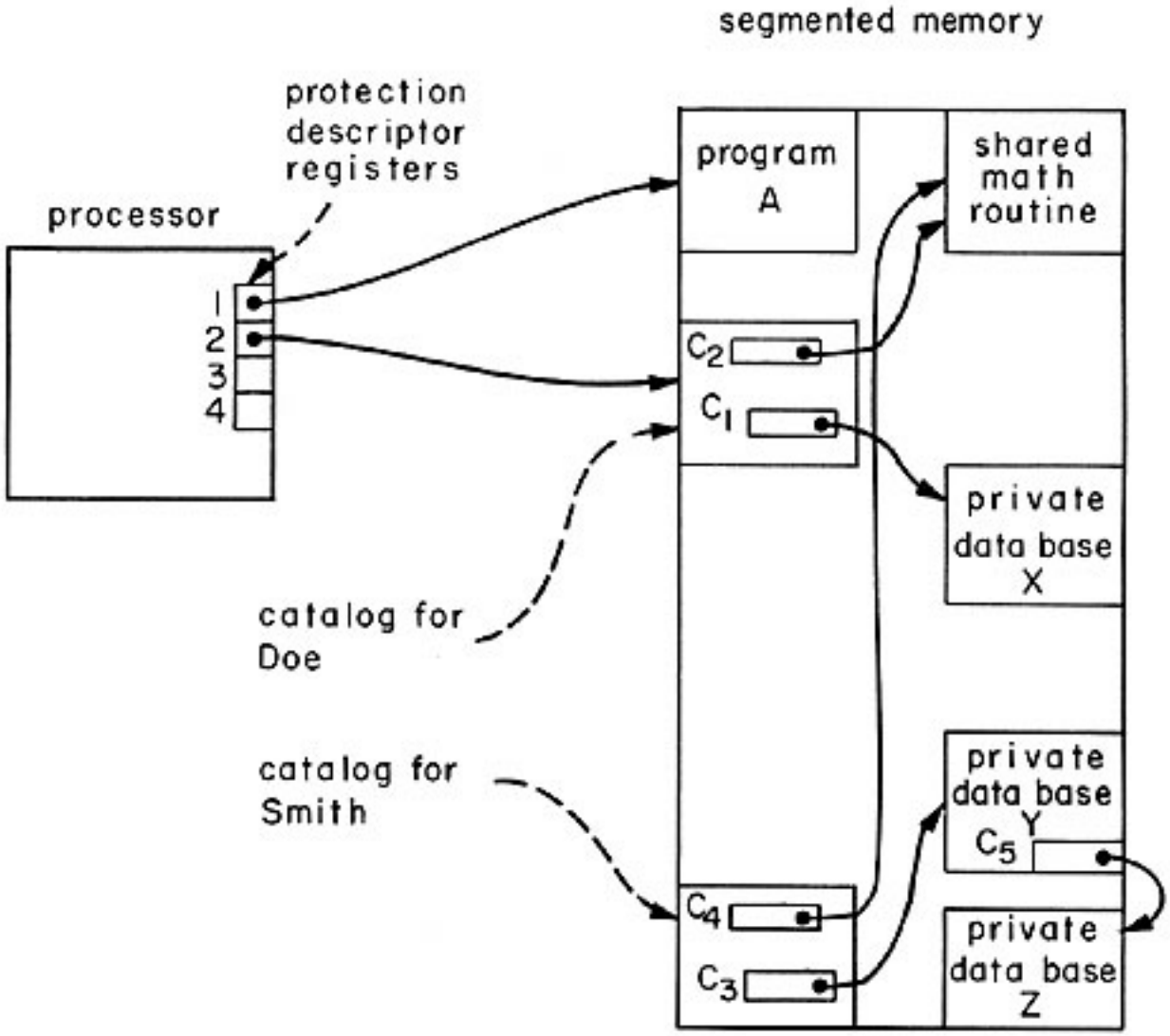


Capabilities

- Capability = *unforgeable* ticket/key
- Guard of operation checks ticket at every access
- Capabilities of user correspond AC matrix row (key ring)
- Important issue: what controls key/ticket propagation?
- Examples: Linux's file handles, Java object pointers



Capabilities in Action (S&S, Fig 6)



ACLs vs. Capabilities

- With ACLs, can audit and revoke access and prevent undesired copying of resources. But every access requires an authorization check.

- With capabilities, authorization is done just once, when initially accessing capability. After that, auditing, revocation, and copying hard to control!

More Realistic Grading Scenario

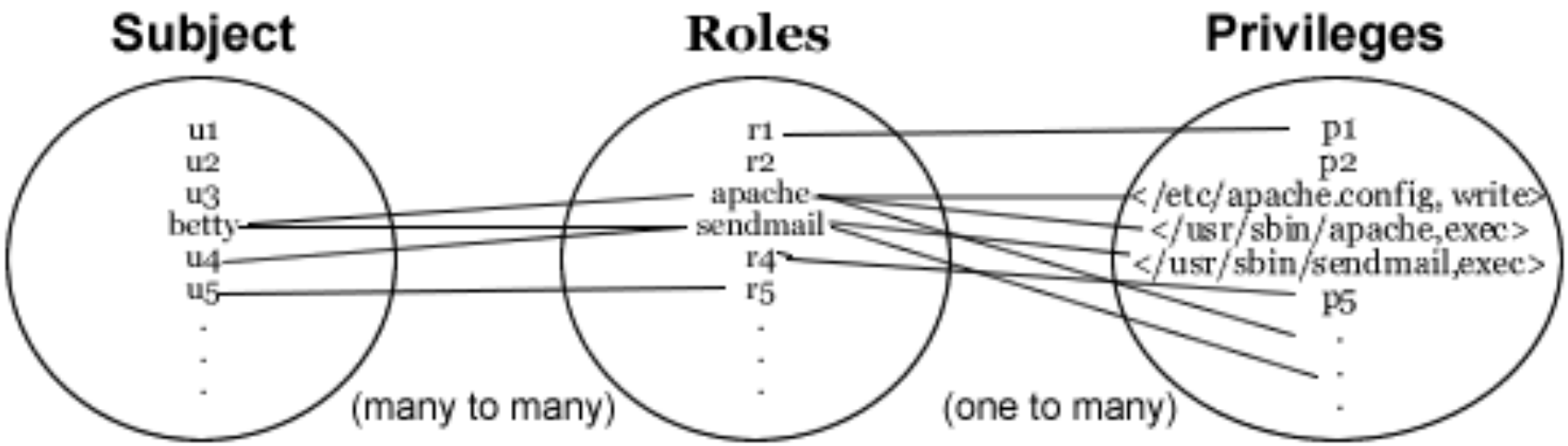
Subjects/Principals	*Objects*		
	grading program	**grade file (all courses, all students)**	**grading log**
gdome	rx		
fbar	rx		
fturbak	rx		
cshanmug	rwxd		r
jstephan	rx		
rpurcell			
grading program		rw	a

- Grading program mediates between users and grading data and effectively implements own access controls
- Similar scenarios for passwords, bank accounts, etc.
- Logs/audit trail problem: even root shouldn't change these!

Almighty Subjects

- Many systems have all-powerful subjects with names like root, admin, superuser
- Convenient for getting work done, but
  - violates least privilege principle
  - easy to make mistakes -- e.g. `rm -rf foo. *`
  - bad if attacker becomes root
- Addressing the problem:
  - force even privileged users to work in unprivileged mode and explicitly invoke privileged operations (e.g. Linux sudo).
  - **role-based access control**: distribute root powers over several subjects (user account manager, webmistress, software updater) ⇒ defense in depth. E.g. Rebecca Shapiro '07's Honor's Thesis *Cooperative and Democratic System Management.*

Role-Based Access Control (RBAC)



Other Security Mechanisms

- **Middleware security**: database looks like one big file to OS, so need access controls at a different level.
- **Sandboxing**: run code in constrained environment that limits damage.
  - Java applets can't access local files; only communicate with source host.
  - JavaScript Same Origin Policy (SOP) tries to prevent communication with servers other than that from which JS code was download.

  But in practice, there can be ways to break out of these sandboxes …
- **Proof-carrying code**: code carries proof that it doesn't do bad things. E.g. JVM stack ops. Easier to check proof than generate it.
- **Virtualization**: run many OS guests on a one host. Guests can't interact directly, but can still copy data between them (insecurely).
- **Trusted computing:** Trusted Platform Module (TPM) chip "approves" PC for running certain software.
- **Web/cloud services**: software running on remote servers less dangerous to your local machine, and can be centrally managed for security. But: do you trust them with your data? Web browsers are the new OSs!