

CS 419 – Computer Security

Week 6

Access Control

Paul Krzyzanowski



Lecture
Notes

© 2026 Paul Krzyzanowski. No part of this content may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.

What We Will Cover

The gatekeeper

The operating system and the hardware that enforce every access decision

The access matrix

Subjects, objects, and rights, stored as access control lists or capabilities

Unix permissions

Owner, group, and other, the identity of a process, and setuid

Limiting privilege

Least privilege, privilege separation, and the limits of owner control

Mandatory control

Bell-LaPadula, compartments, Biba, Chinese Wall, type enforcement, and roles

Capital One (2019)

1 The request

A firewall on a cloud virtual machine could be made to fetch a URL the attacker supplied.



2 The credential

She had it fetch the machine's metadata service, which hands out the credentials of the role the machine runs as.



3 The data

The role could list and read the buckets that held customer records. She copied about 30 GB.

About 106 million applicants, 140,000 Social Security numbers were accessed

**The firewall did not need to read customer records
... but the policy said it could.**

Access control

- Ensure that authorized users can do what they are permitted to do
... *and no more*
- Real world
 - Keys, badges, guards, policies
- Computer world
 - Hardware
 - Operating systems
 - Web servers, databases & other multi-access software
 - Policies



The Operating System as Gatekeeper

The Operating System Controls Access to Resources

The processor

A hardware timer that applications cannot disable returns control to the kernel, and its scheduler decides who runs next.

Memory

The memory management unit translates every address through a per-process page table with permissions on each page.

Files and devices

A program cannot access a disk or a device directly. Every operation is a system call the kernel checks first. Storage is accessed through file systems.

The network

Opening a socket, binding a port, and sending packets.

The Operating System's Protection Responsibilities

Protect itself

An application cannot overwrite kernel code or data.

Protect processes from each other

A process cannot read another's memory (confidentiality) or change it (integrity).

Stay in control

A process cannot hold the processor forever or stop the kernel from running (availability).

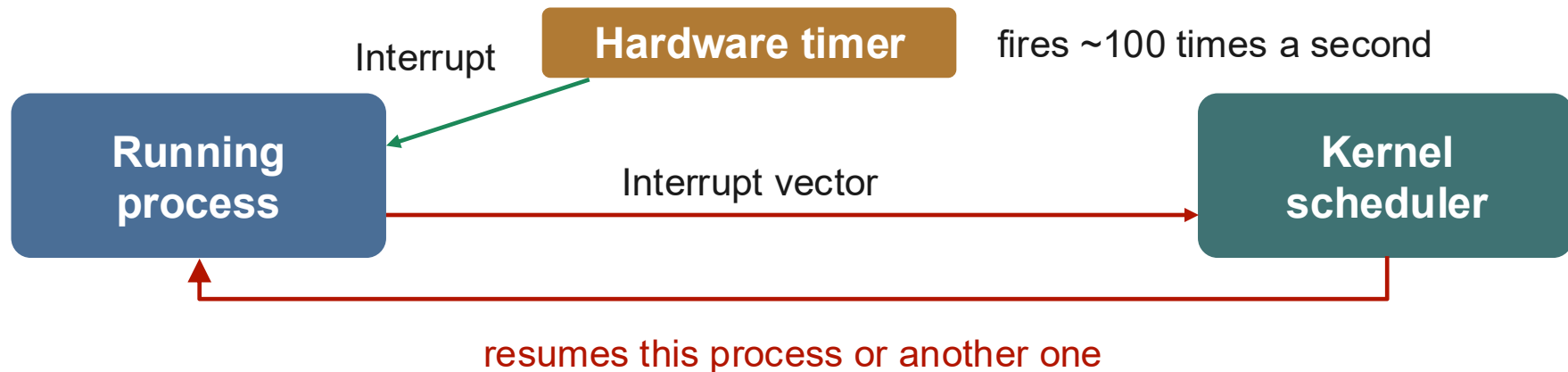
Restrict access to resources

Enforce policies for reading, writing, and modifying files and devices.

MS-DOS, early Windows, and the classic Mac OS did none of these:
One misbehaving application could crash the machine.

The kernel and the hardware that enforce protection are the core of the **trusted computing base (TCB)**

Regaining the Processor: Hardware Timer & Scheduler



Preemptive multitasking

The kernel decides when a process gets suspended.

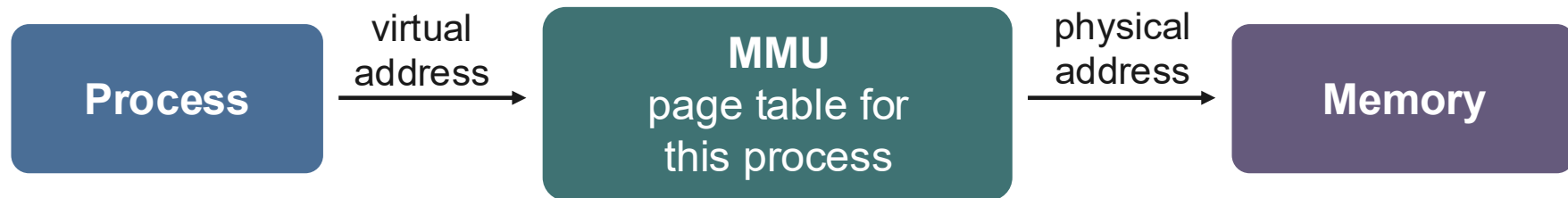
Applications cannot disable the timer; the instructions are privileged.

What it protects

Availability.

A process cannot starve the others or keep the kernel from running.

Isolating Memory: Memory Management Unit



Each page table entry identifies where a page is in physical memory and how the process may access it: read, write, execute.

Isolation

A process can refer only to memory its page table maps.

Each process gets its own address space

Kernel memory

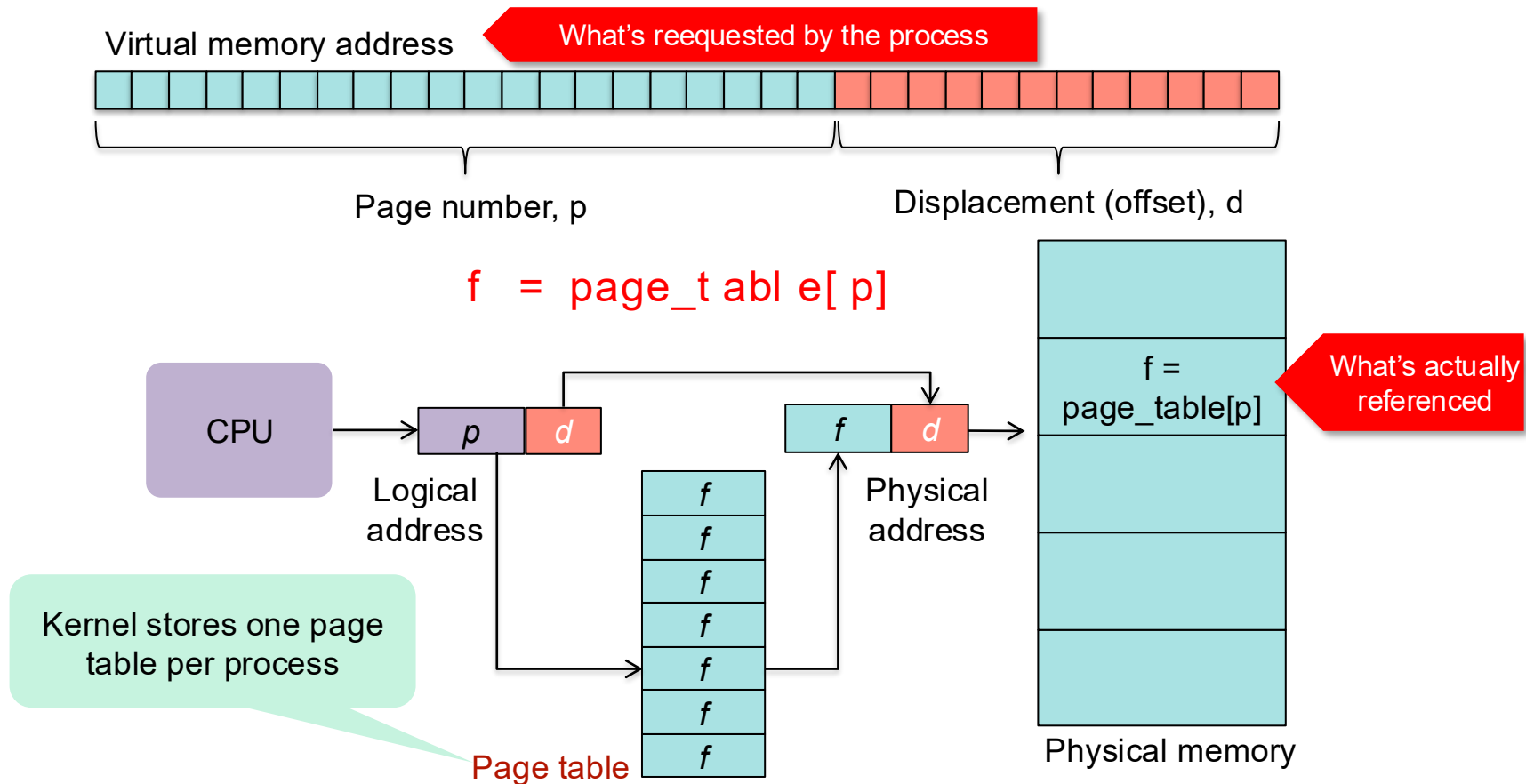
In every page table, but tagged so that only the kernel can access it.

A forbidden access or unmapped location

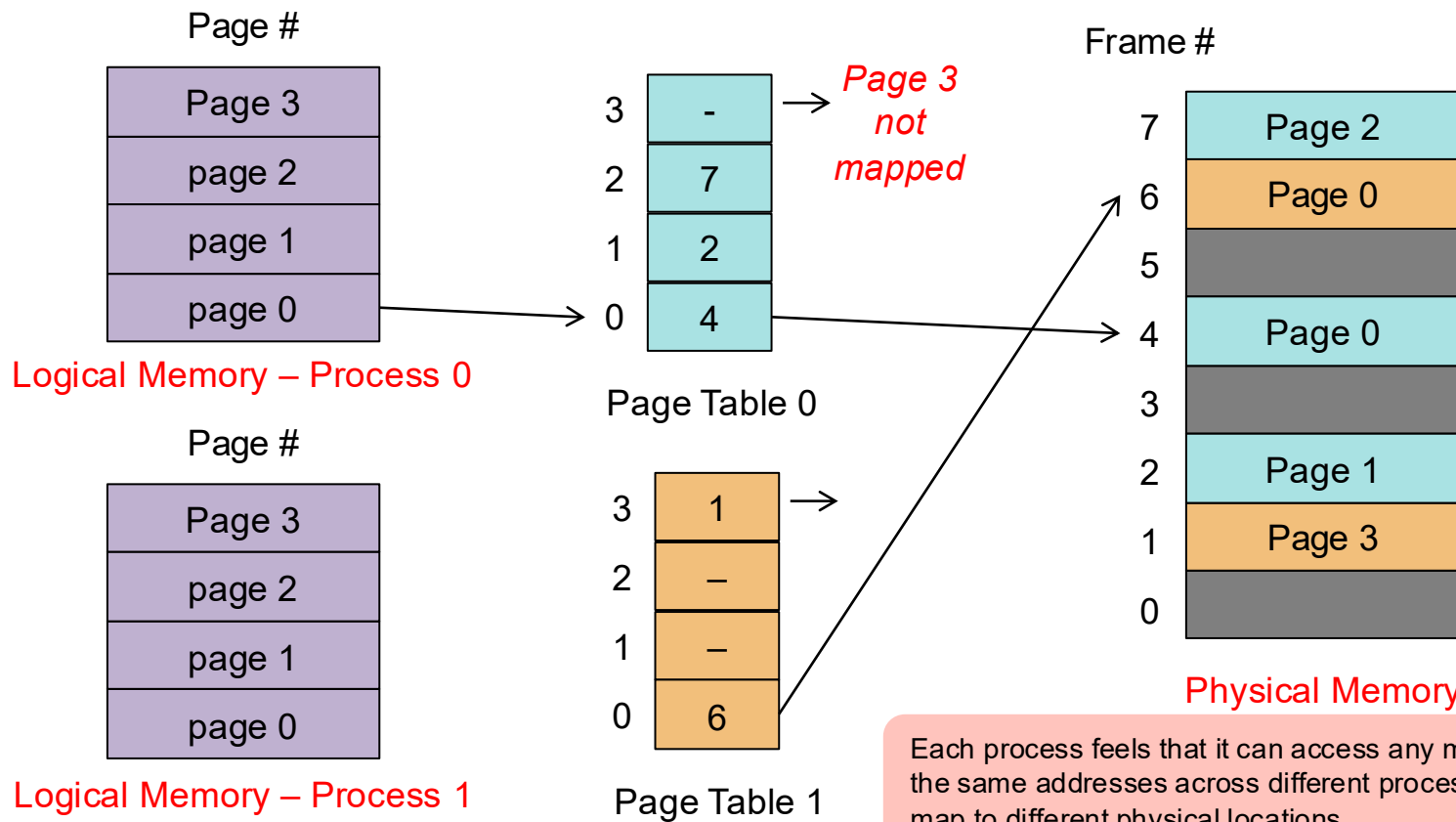
Causes a page fault.

The hardware reads the missing page from the file system or kills the process

Page translation



Logical vs. physical views of memory



Each process feels that it can access any memory but the same addresses across different processes will map to different physical locations.

User Mode and Kernel Mode

User mode

Applications. No privileged instructions, no access to kernel memory.
An attempt gets a trap to the kernel.

System call

A trap instruction,
executed to request a
kernel service

Fault

A forbidden instruction
or memory access

Interrupt

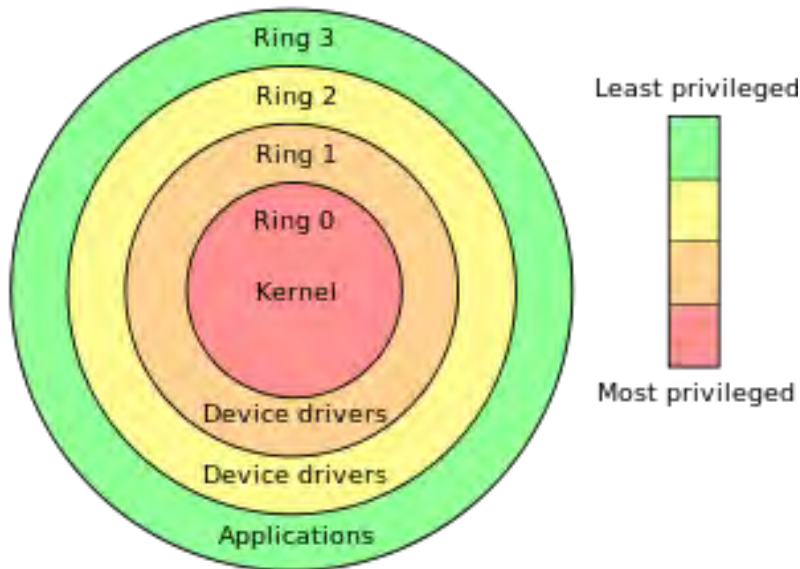
A device or the timer
needs attention

Kernel mode

The operating system.
Every instruction available, including the ones that program the timer and the MMU.

Protection Rings and the Level Below

Multics divided its software into eight rings in the 1960s. Intel provided four.



Special call instruction (**call gates**) to cross rings: jump to predefined locations

Every mainstream operating system uses only two:

- Ring 0: kernel
- Ring 1: user

Additional protection levels

- Ring -1: Hypervisor (virtual machine monitor)
- Ring -2: System Management Mode (SMM)
 - Low-level, high-priority tasks like power management, thermal monitoring

The Reference Monitor

What would a computer need to be trusted with classified information?

Reference Monitor

Checks every object reference attempt against a policy

Always invoked

No path to an object goes around it.

Tamperproof

No program it controls can change it.

Small enough to verify

Someone can establish that it is correct.

The kernel's access checks are the reference monitor of an operating system. A database or a web server is its own reference monitor.

Subjects, Principals, Objects, and Rights

Subject

An active entity that requests access. In an operating system, a **process**.

Principal

The identity a subject acts for, usually a **user** account. Every process has one.

Object

A passive resource: a file, directory, device, socket, region of memory, or another process.

Right

An operation a subject may perform on an object: read, write, execute, delete.

Identification claims an identity, **authentication** verifies it, and **authorization** decides what the principal may do. Access control is authorization.

User authentication

Before we can do access control ...

Establish user identity – determine the *principal*

- Access is granted based on user identity

Common steps:

1. **Identify**: Get the username
2. **Authenticate**: Get the password, validate it
 - Look up user ID(s), group ID(s)
3. **Authorization = Access control**
 - Approve operations subjects can perform on objects

The Access Matrix

The Access Matrix

subjects (domains)	objects			
		File A	File B	Printer
	Alice	read, write	read	print
	Bob	read		
	Process X		write	

One **row** per subject, or **protection domain**.

One **column** per object.

Each **cell** contains the **rights**.

Butler Lampson:

Every access decision is a lookup: **(subject, object) → {rights}**

The matrix is a model. It says what should be allowed without saying how it is stored.

Rights That Change the Matrix

Owner

May change any of the object's columns: **grant or revoke rights for other subjects.**

This right is what makes control discretionary.

Copy

May give a right it holds to another subject.

This is delegation.

Domain transfer

May switch to another row and acquire its rights.

Examples: logging in, changing a password.

These rights are about the matrix itself, not about the object's contents.

The Matrix Is Sparse

$$\begin{array}{ccccc} 1,000 & \times & 1,000,000 & = & 10^9 \\ \text{users} & & \text{files} & & \text{cells,} \\ & & & & \text{most empty or default} \end{array}$$

Most users have rights over very files.
Most cells are empty or hold default rights



The table takes enormous
space store not much info

Store each column with its object
Access control list

Store each row with its subject
Capability list

Access Control Lists: a Column per Object

File A

Alice: read, write

Bob: read

staff: read

*the column,
stored with the object*

Easy

“Who can read this file?”
is one lookup.

Revoking one user’s
access is one edit to one
list.

Hard

“What can Alice access?”
means searching every list
across all objects.

Removing a user means
visiting every list.

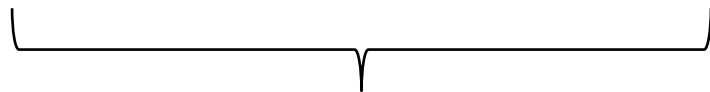
Checking at every access means finding the subject in the
list.

Access Control Lists

Access control list =

For each object:

{ subject₁:permitted_operations, subject₂:permitted_operations, ... }



= ACE: access control entry

Capability Lists: a Row per Subject

Alice

File A: read, write

File B: read

Printer: print

*the row,
stored with the subject*

Capability

An **unforgeable** token naming an object and a set of rights.
Presenting it is the access check.

Easy: checking

“What can Alice access?” is
one lookup.

Hard: revocation

*“Remove all access to File
A”* means searching every
list across users.

- An ACL optimizes *“who can access this object?”*
- A capability list optimizes *“what can this subject access?”*

Capability Lists

Capability list

For each subject:

{ object₁:permitted_operations, object₂:permitted_operations, ... }



capability

= ticket = access token

Possessing the capability means that access is allowed

A capability is a protected object

Capabilities Already in Use

An open file descriptor

The check happens once, at `open()`.

The file descriptor is presented on every `read()` and `write()`, and it can be passed to another process.

A Kerberos ticket, (almost)

One ticket per service, presented with proof of the key inside it.

It names a principal, not an object, and the service decides the rights.

Cloud credentials, (almost)

Web session cookies provide ongoing access credentials to a site ... which attackers may take advantage of,

Operating systems are built on them:
KeyKOS, seL4 microkernel, and Google's Fuchsia.

The Confused Deputy



The Confused Deputy



The deputy's rights are checked, not the original caller's

Hardy's compiler

Allowed to write a billing file.

Asked to write its output to that file's name & it did.

Capital One's firewall

Could reach the credential service.

Attacker could not, so she had it fetch the credentials for her.

A privileged Unix program

More authority than the user who ran it.

A deputy is confused the moment it skips a check.

Discretionary and Mandatory Control

Discretionary access control

The **owner** of an object decides who may access it.

The system enforces the owner's decision and imposes none of its own.

Unix and Windows file permissions.

Mandatory access control

The **organization's policy** sets the rules.

The rules are fixed and owners cannot override them

The matrix says what is allowed — It does not say who can make changes

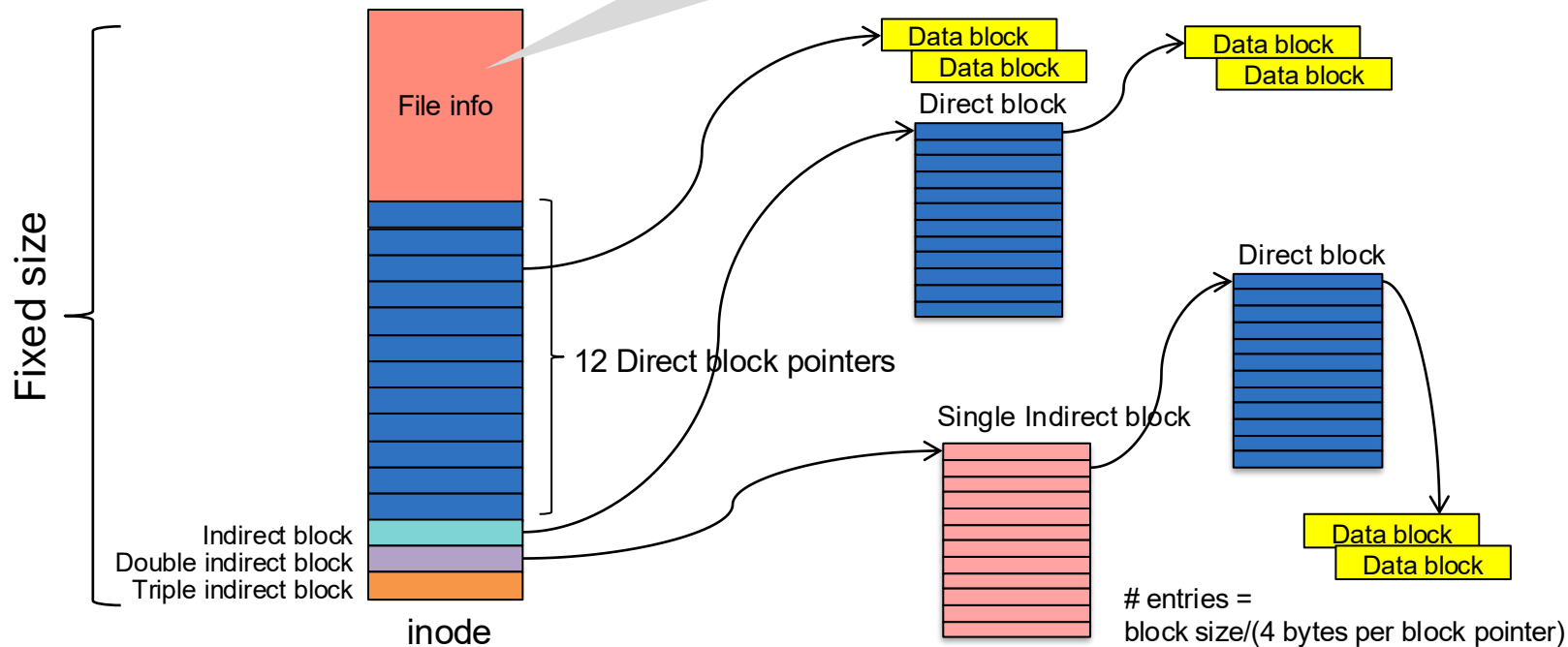
That divides access control into two families: discretionary & mandatory
Nearly every modern system has some of each.

The Unix Permission Model

File permissions are stored in the file's inode

Problem: an ACL takes up a varying amount of space – Won't fit in a fixed-size inode

Stores:
owner id, group id, permissions,
access/creation/modification times



Limited ACLs in POSIX systems

UNIX Compromise:

A file defines access rights for only three domains:

(1) **owner (user)**, (2) **group**, (3) **everyone else**

```
-rwxr-xr-x 1 root wheel 252512 Sep 24 03:10 /bin/ls
```

-

type

rwX

owner

r-X

group

r-X

other

owner **root**
group **wheel**

r read

read contents of the file

w write

modify the contents

x execute

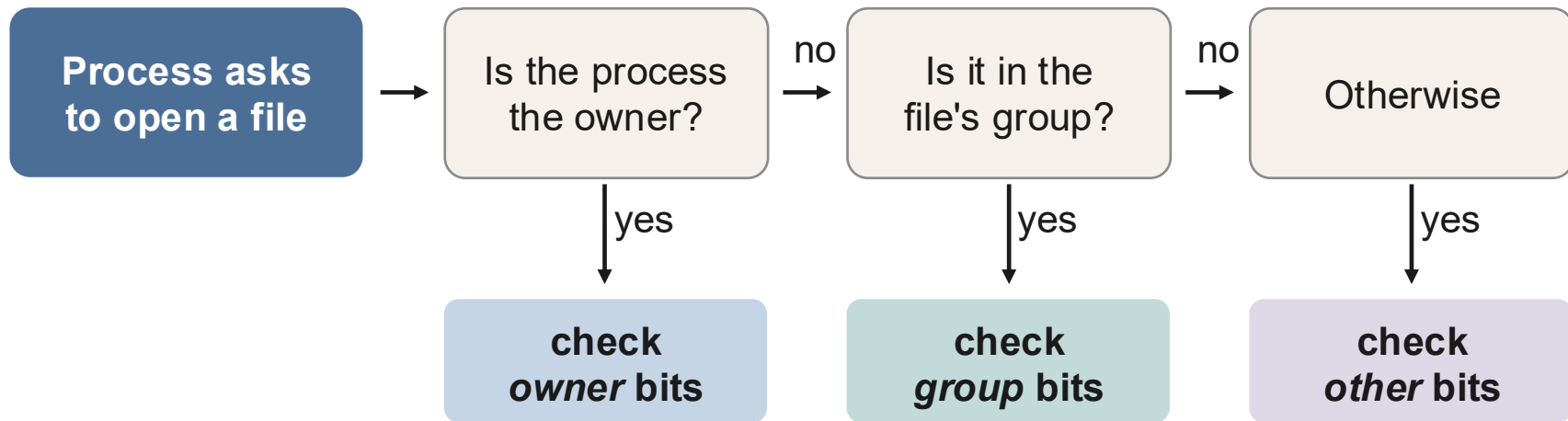
run it as a program

Nine bits in the file's **inode**:

three each for the owner, owner's group, and everyone else.

How a Permission Check Works

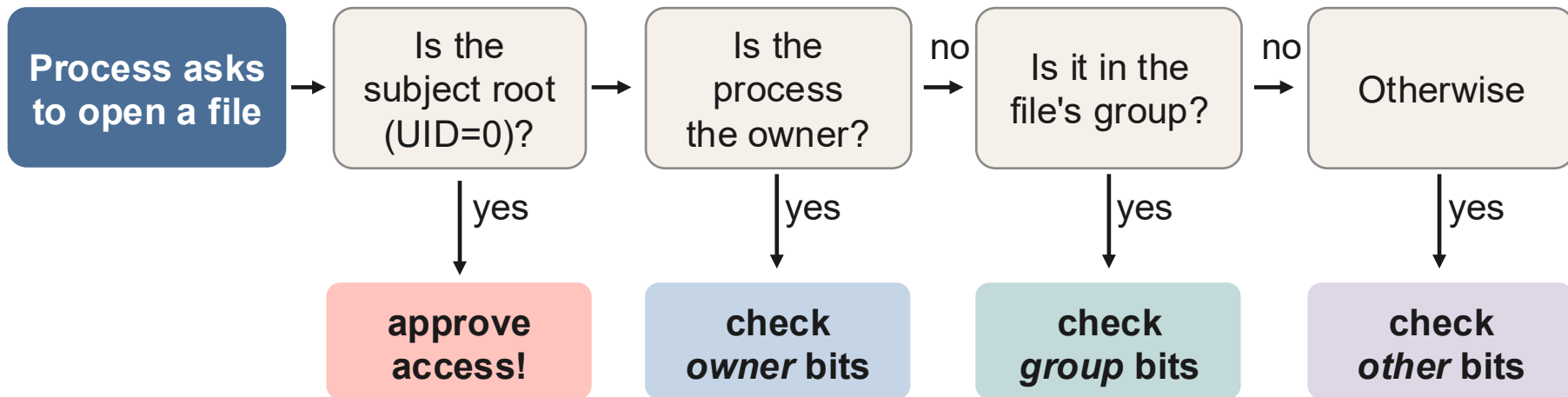
Only one set of bits is consulted. Others are ignored even if they would grant access. An owner whose own bits deny reading is denied, even if the file is readable by the group and the world.



How a Permission Check Works

The root user (superuser, user ID = 0) is special.

Access checks are bypassed for UID=0.



Execute Without Read

```
$ ls -l secretprog
-r-x--x--x 1 root  staff  8492 secretprog
$ cat secretprog
cat: secretprog: Permission denied
$ ./secretprog
(runs)
```

A binary

The kernel loads it.
The process never needs to read it, so the contents stay private.

Why this exists

Lets a program be run without being copied or inspected.

Can't do this with scripts

Run by an interpreter such as /bin/sh or python3, which runs as the user and has to read the file. Not readable means not runnable.

What the Bits Mean for a Directory

Bit	Meaning for a directory
r	List the names in the directory
w	Create, delete, or rename entries in it (with x)
x	Traverse it (search): use it in a path to reach something inside

No x

Nothing inside can be reached, even by a process that knows the exact name.

x without r

A file can be opened by name, but the names cannot be listed.

Deleting a file

Needs w and x on the directory, not w on the file. The sticky bit on /tmp limits it to the owner.

Changing Permissions

```
$ chmod u=rw,g=r,o= notes.txt
$ ls -l notes.txt
-rw-r----- 1 paul  staff  notes.txt
```

symbolic form

```
$ chmod 640 notes.txt

  r  w  x
  4  2  1

6 = rw-, 4 = r--, 0 = ---
```

octal form, one digit per set

chmod

The owner of a file (and root) may change permissions

chown

Only root may change a file's owner.

chgrp

The owner may change the group to another one the owner belongs to.

Default Permissions and the umask

Library passes

0666

&~

umask

022

=

File is created as

0644

022 = --- -w- -w-

rw-rw-rw- remove *write* for group and other = rw-r--r--

What it is

A per-process list of bits to remove from every file the process creates, in the same octal form as chmod.

Inherited by child processes, so a shell's umask applies to everything run from it.

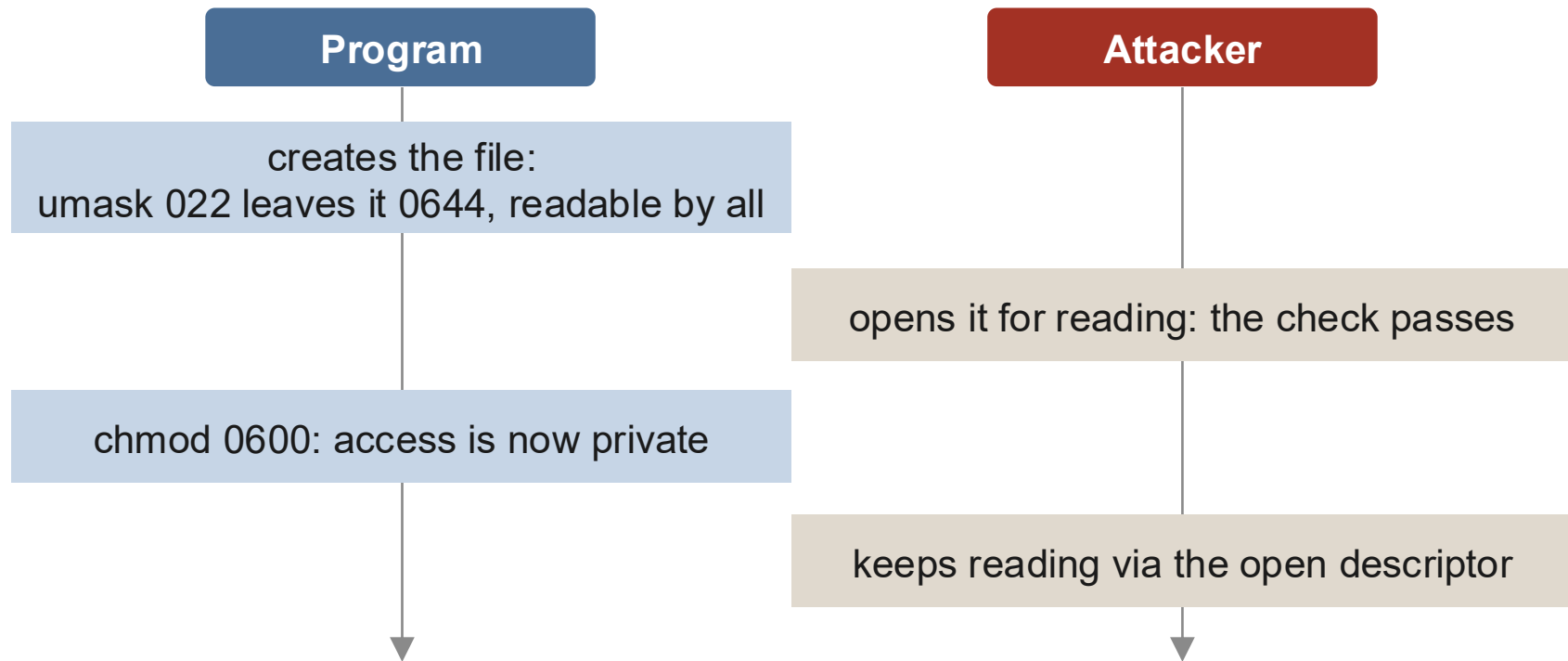
Why it exists

Most programs don't choose a mode. `fopen()`, Python `open()`, and shell redirection all pass 0666.

umask removes permissions.

Defaults are 022 or 027.

The Access Check Happens at Open



The check protects the act of opening, not the name.
Create the file with mode 600, or run under a umask of 077

Users, Groups, and Root

```
root:x:0:0:System Administrator:/root:/bin/sh
paul:x:1000:1000:Paul K:/home/paul:/bin/bash
```

/etc/passwd: login name, (password, now in /etc/shadow), user ID, group ID, name, home directory, shell

User ID

The principal.

A number. The name is a convenience for humans.

Groups

Defined in /etc/group.

A process has a primary group and a list of supplementary groups; any of them counts.

root = uid 0

The kernel skips nearly every permission check.

How Login Sets Up a Process



Identity is inherited

The login program, running as root, sets it for the first process, and everything the user runs descends from that.

setuid() is one way

Once a root process has called it with an ordinary user ID, there is no way back.

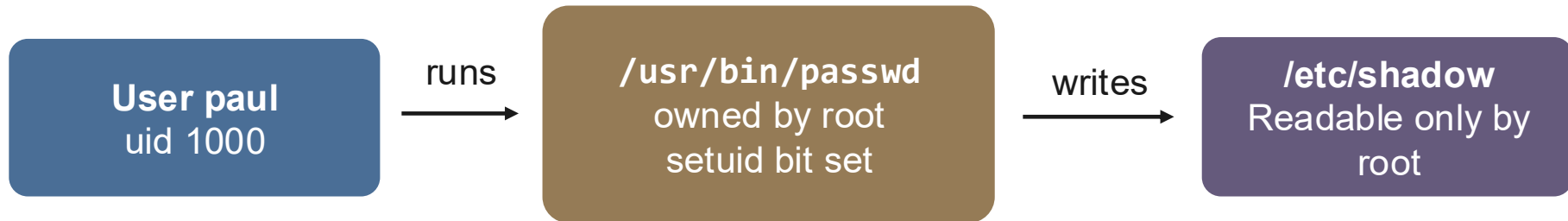
In the access matrix, login performed a **domain transfer** from root's row to the user's.

A process has:

- **real user ID** (who started it)
- **effective user ID** (what the kernel checks). They are usually the same.

Running with Someone Else's Authority

setuid Programs



-rwsr-xr-x 1 root root 45752 /usr/bin/passwd

setuid bit

The process runs with the file owner's ID **effective user ID**.

The **real user ID** stays the caller's.

A domain transfer

From the user's row of the matrix to root's. The only protection is the program's own code.

A deputy

Holds root's authority, acts for an unprivileged caller.

Confused if it fails to check what the caller can do.

Example: PwnKit (2022)



pkexec, a setuid-root program from the polkit package

- Mishandled an empty argument list.
- The result was a memory corruption bug that a local user could turn into a root shell.
- Disclosed by Qualys on January 25, 2022.
- Present since the first version in May 2009.

Confirmed on default installs:

Ubuntu, Debian, Fedora, and CentOS

A Check and a Use That Disagree

```
if (access(path, R_OK) == 0) {  
    fd = open(path, O_RDONLY);  
    ...  
}
```

`access()` checks against the **real** user ID.
`open()` checks against the **effective** one.

Both name a path, but there is no guarantee the name refers to the same file both times.

1 `access(path)` passes

path is the user's own harmless file

2 the user swaps it

deletes path and replaces it with a symbolic link to **/etc/shadow**

3 `open(path)` succeeds

as root, on the file the user could not read

```
rm path; ln -s /etc/shadow path
```

The program checked a name and then used the name.

Time-of-check to Time of use (TOCTTOU) attack

Reducing the Risk of setuid

Fewer setuid programs

Install as few as possible and audit the ones that remain.

Drop the privilege

Do the privileged work, then drop.

`setuid()` while still privileged makes the drop permanent and clears the saved ID.

Run setuid program: sudo

Itself setuid root, but one heavily reviewed program under a policy file replaces a shelf of them.

Split root into pieces

Linux divides the superuser's authority into about forty **capabilities**, such as binding a low port.

Not related to capability lists.

Least Privilege and Privilege Separation

The Principle of Least Privilege

Every program and every user should operate with the smallest set of privileges needed to complete the job.

—Saltzer and Schroeder, 1975

What it does

Limits damage rather than preventing mistakes.

A web server that can only read its content directory cannot leak the password file.

Dedicated user per service

The **nobody** user for programs that do not need file access.

Bind the port as root, then **setuid()** before reading from the network.

For people

Log in as yourself and use **sudo** for individual commands, so a mistyped command runs with the smaller set of rights.

Capital One Revisited

What the firewall needed to do

Inspect web traffic.

No need to access customer data.

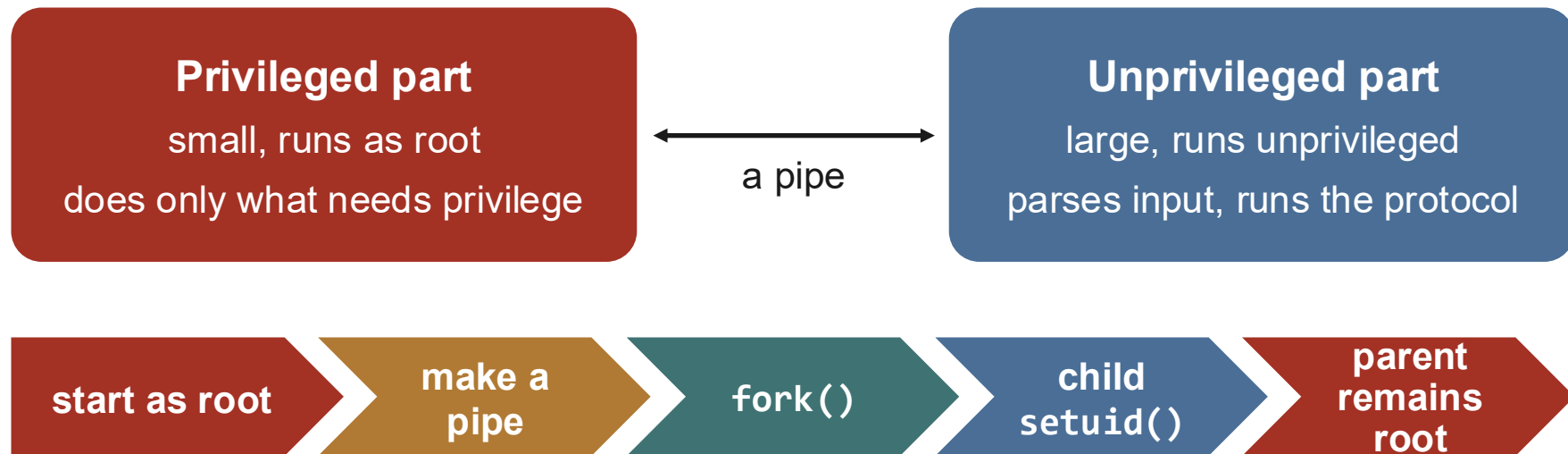
What its role was able to do

List the storage buckets that held customer records and read their contents.

Least privilege would have given the role **only** what a firewall needs.

Stolen credentials would have reached none of the customer data.

Privilege Separation



The parent is small enough to review, and its **attack surface** is the pipe's protocol, not the network.

Bugs in the child have a limited ability to do damage.

OpenSSH (2002)

Monitor

- runs as root
- holds the host private key
- authenticates the user
- creates the user's process



Network child

- runs as sshd, unprivileged
- key exchange
- parses every network packet

Shipped May 2002

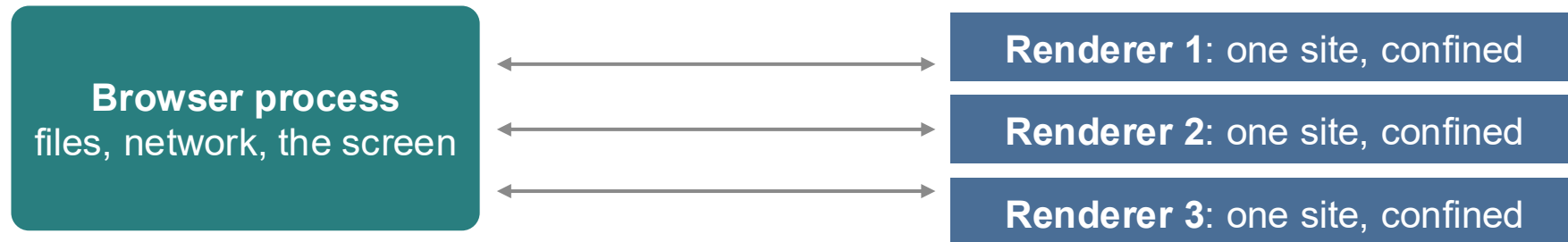
OpenSSH 3.2.2, by Niels Provos, Markus Friedl, and Peter Honeyman. Every Unix login server runs this code.

Six weeks later

A remotely exploitable bug in the authentication code.

The advisory: *systems with privilege separation enabled were protected against the escalation.*

Browsers Separate Privilege the Same Way



A renderer that needs a file sends a request to the browser process,

- The browser process decides whether to grant it.

The code that parses a web page is the most exposed code on the machine

- It runs in a process that cannot open a file or a socket on its own.
- An attacker who exploits a bug in it has minimal privileges

Why Discretionary Control Is Not Enough

Programs act as their user

Malicious software can read every file its user can read and send it anywhere.

The system cannot tell the user's intent from the program's.

Rights propagate

A user who can read a file can copy it.

The copy is a new file whose permissions the user sets.

There is no system policy

Every object's permissions are set by its owner.

“Nothing marked confidential leaves this group” cannot be stated, because any owner can undo it with `chmod`.

Mandatory access control (MAC) exists to remove these three limits

Mandatory Access Control

Labels, Clearances, and Classifications

The problem

The Department of Defense wanted one computer to hold documents for users with different clearances.

What changes

Every subject and object has a **label**.

The OS compares labels on every access.

The owner cannot relax the result.

What varies

What a label contains, and what the comparison is.

Bell-LaPadula

labels, two rules

Multilateral

the rest of the label

Biba

the other direction

Chinese Wall

history

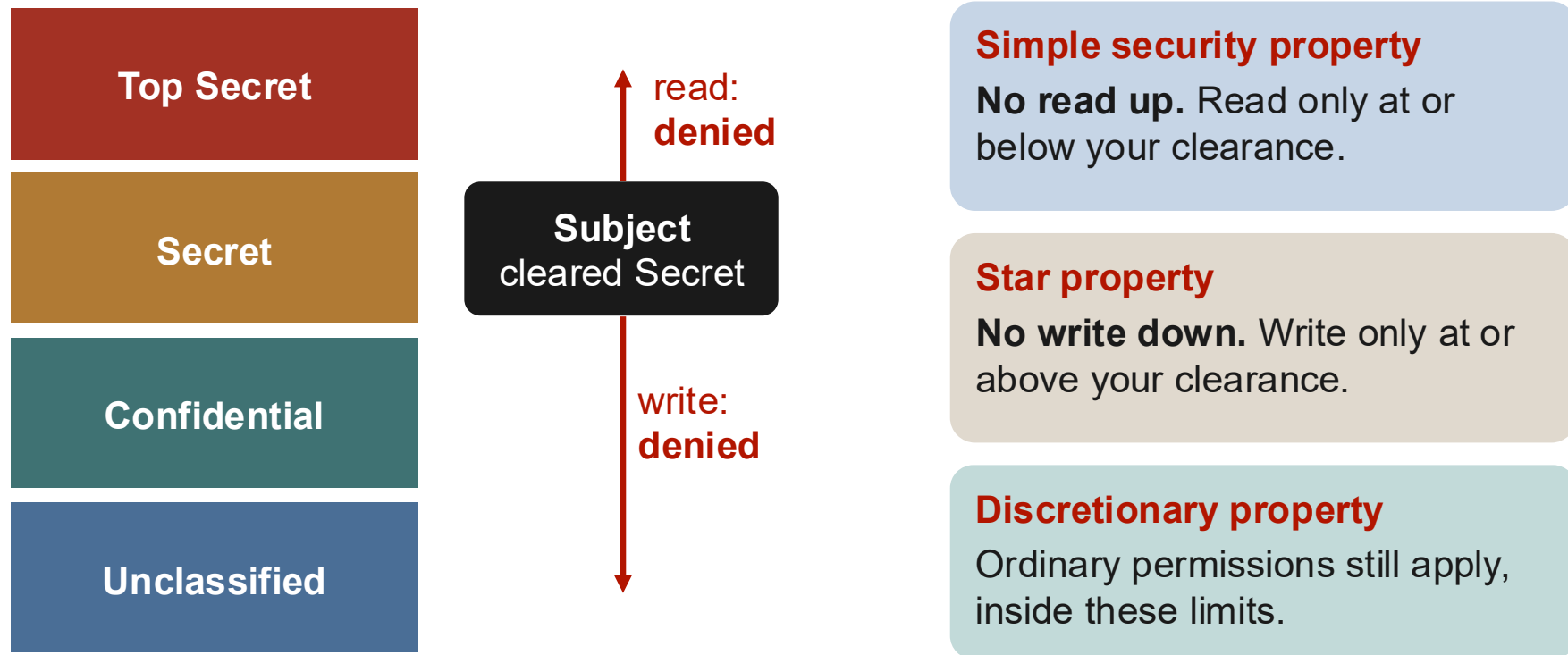
Type Enforcement

domains and types

Role-Based Access

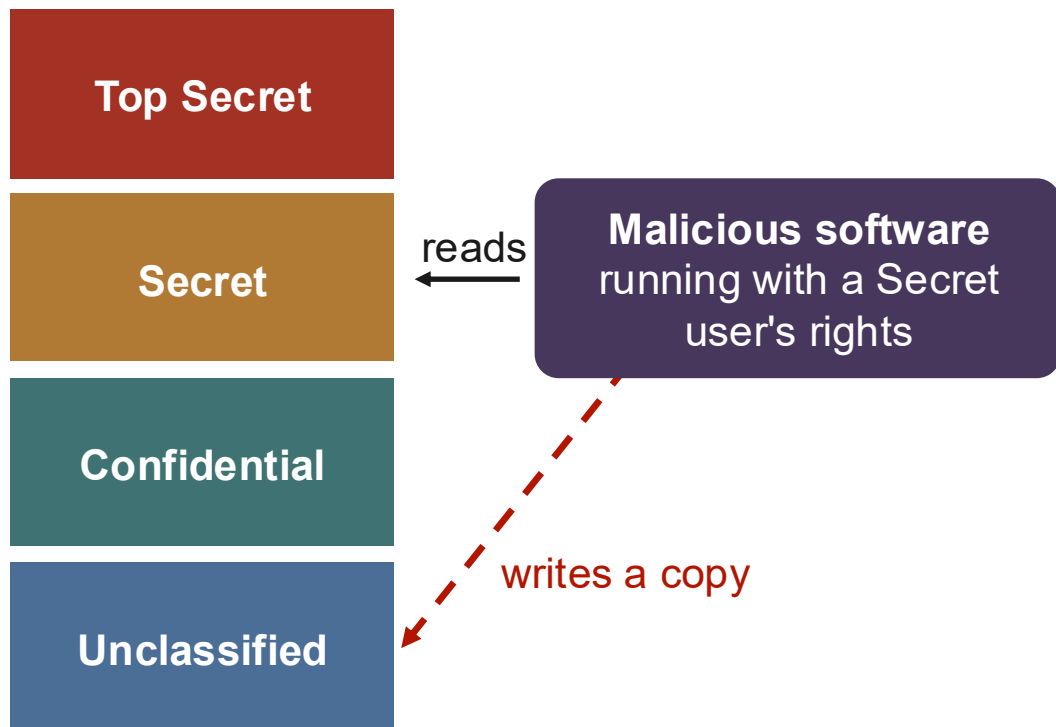
instead of subjects

Bell-LaPadula (BLP) Model – Multilevel Security



Objects have a **classification**, subjects have a **clearance**
Information can move up and never down.

Why No Write Down



Without the rule

The program reads a Secret file and writes it to an Unclassified file, where anyone can read it. Discretionary control allows this.

With the rule

A process at the Secret level may not write anything below Secret.

A Secret-Cleared Analyst and Three Files

File	Classification	Read	Write
A	Confidential	allowed	denied: write down
B	Secret	allowed	allowed
C	Top Secret	denied: read up	allowed

An analyst is cleared for **Secret**.

- The analyst may write to a Top Secret file they cannot read.

Limits of Bell-LaPadula

Nothing can move down

Declassifying and sanitizing need **trusted subjects** that are exempt from the rules.

System security then rests on those programs.

Nothing about integrity

A low-clearance user may write a Top Secret file.

Garbage can flow upward freely.

Only the channels it labels

Two processes at different levels can still signal through timing or resource use.

Covert channels.

Multilateral Security: Compartments and Need to Know

Top Secret

{TS, {Nuclear}}

{TS, {Signals}}

{TS, {Cyber}}

The rest of the label

The full Bell-LaPadula label is a level **and** a **set of compartments**.

Clearance says a person may be trusted at a level

Compartments say which projects at that level they can access.

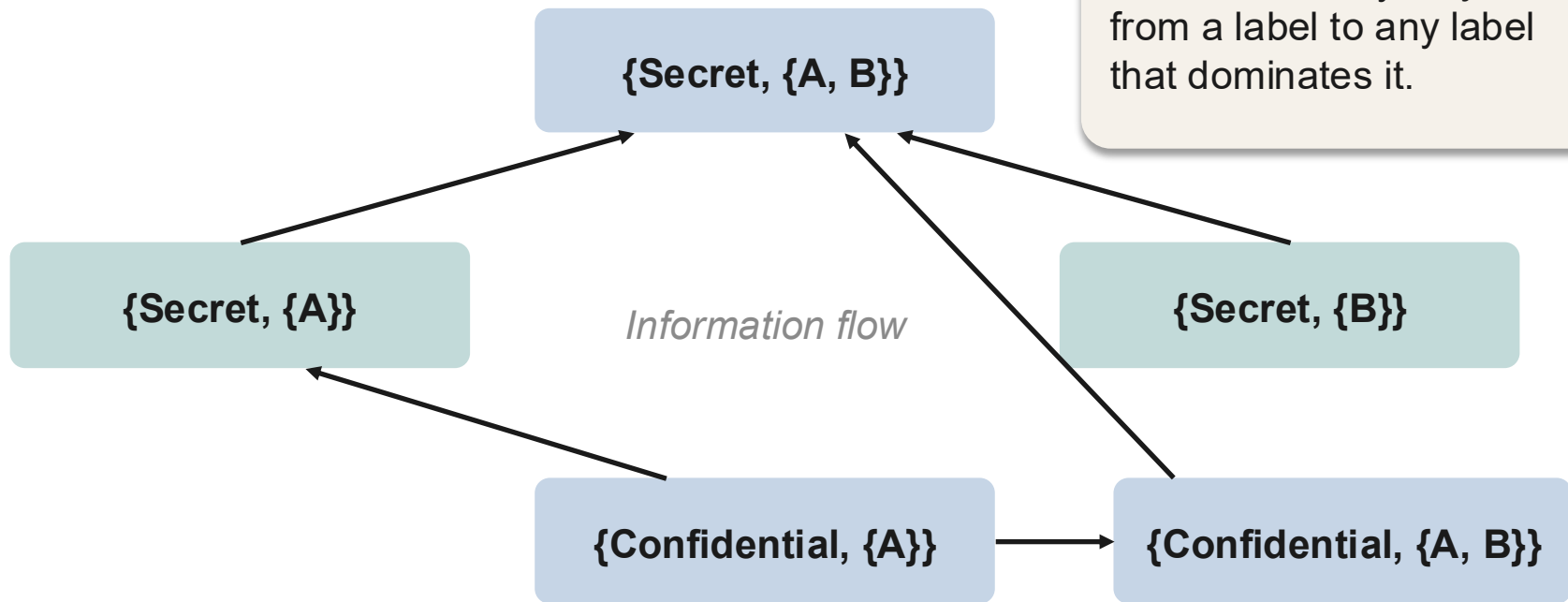
Two required tests

Level at least as high, **and** compartments include every one of the object's.

Passing both checks is called **dominating** the label.

**Read what you dominate;
write what dominates you.**

Labels Form a Lattice

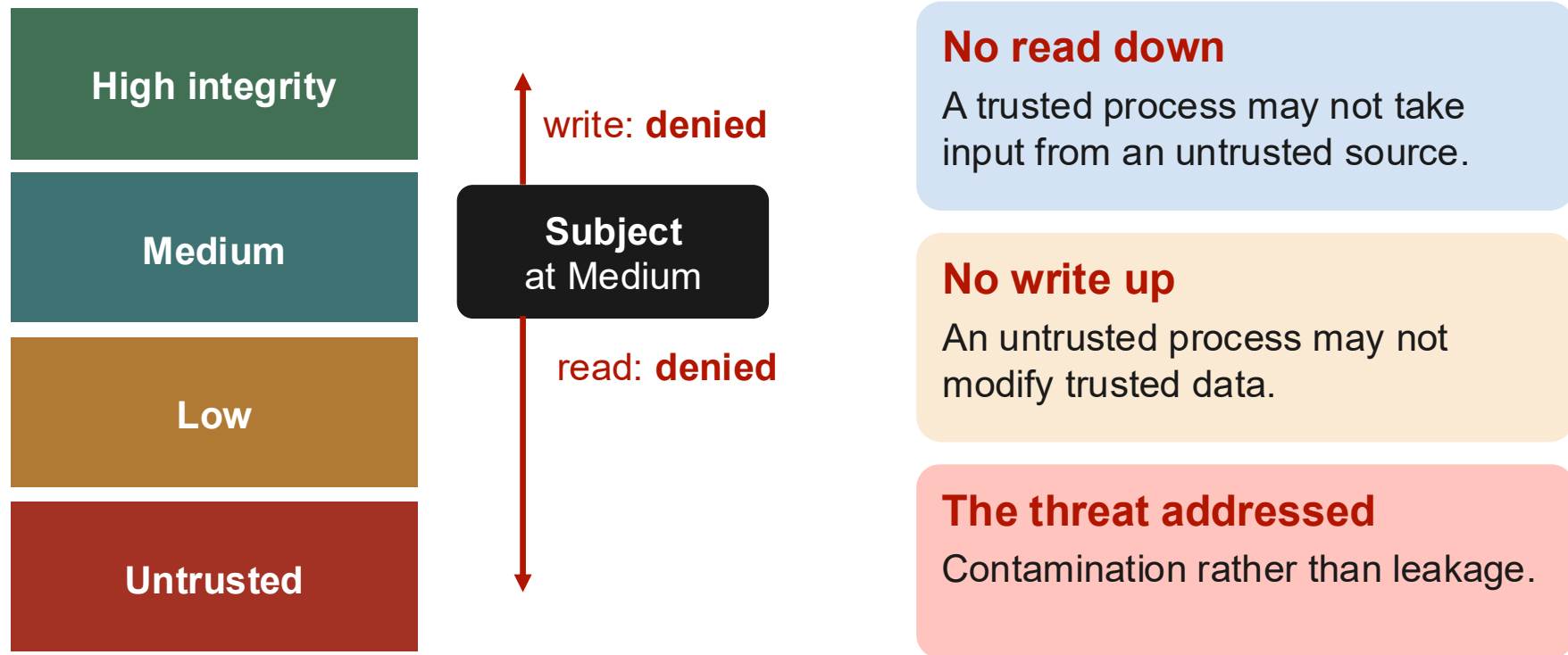


A higher level is not enough:

$\{\text{Secret}, \{A\}\}$ may not read $\{\text{Confidential}, \{A, B\}\}$, since it lacks B.

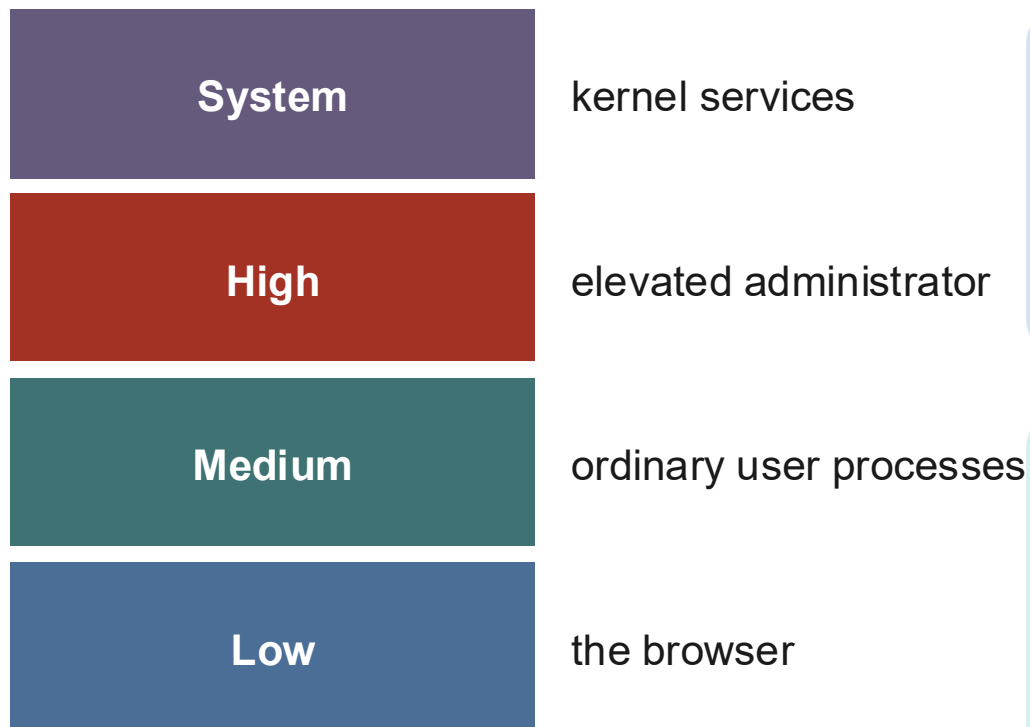
$\{\text{Secret}, \{A\}\}$ and $\{\text{Secret}, \{B\}\}$ are **incomparable**.

Biba (1977): No Read Down, No Write Up



Kenneth Biba, MITRE, 1977. Same structure as Bell-LaPadula, opposite goal: **integrity**.

Windows Integrity Levels (Vista, 2007)



Mandatory Integrity Control

No write up is enforced: a process may not modify an object above its level, whatever the DACL says.

First used for a browser

Internet Explorer 7 ran at Low.
A page that exploited it could not write documents, the registry, or system files.
It could still read them.

AppContainer was later built on top of it.

Bell-LaPadula and Biba Compared

	Bell-LaPadula (1973)	Biba (1977)
Protects	confidentiality	integrity
Read	no read up	no read down
Write	no write down	no write up
Stops	leaks downward	contamination upward

Confidentiality vs. Integrity labels

A system can run both.

Biba rules don't work well in real systems:

programs that process untrusted input are the normal case, not the exception

The Chinese Wall Model

Airlines: a conflict of interest class

Alaska

American

Delta

United



read

Beverages

Coca-Cola

PepsiCo

Keurig

An employee who has read Delta's confidential data may not read Alaska's, American's, or United's.
The beverage companies are still available.

The wall is built by the subject's history.

Company dataset

Everything the firm has about one client.
Each object belongs to one dataset

Conflict of interest class

The datasets of competing companies.
Each dataset belongs to one.

Access That Depends on History

1 Nothing read yet

Every dataset is available.



2 Reads Delta

Alaska, American, and United are denied. Beverages are still available.



3 Reads PepsiCo

Coca-Cola and Keurig are denied. Only Delta and PepsiCo remain.

The write rule

Write to a dataset only if you are not permitted to read any other dataset, except sanitized data. Read widely or write, not both, or Delta's plans end up in a file a United employee reads.

Where it runs

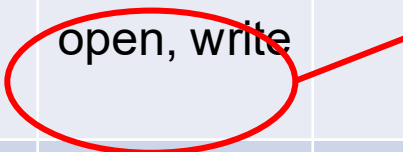
Not in kernels, which do not know what a client is. In the applications of banks, law firms, and consultancies, where regulators expect such barriers. A dynamic separation of duties.

Type Enforcement: Domains and Types

Process runs in a **domain**

Objects are assigned **types**

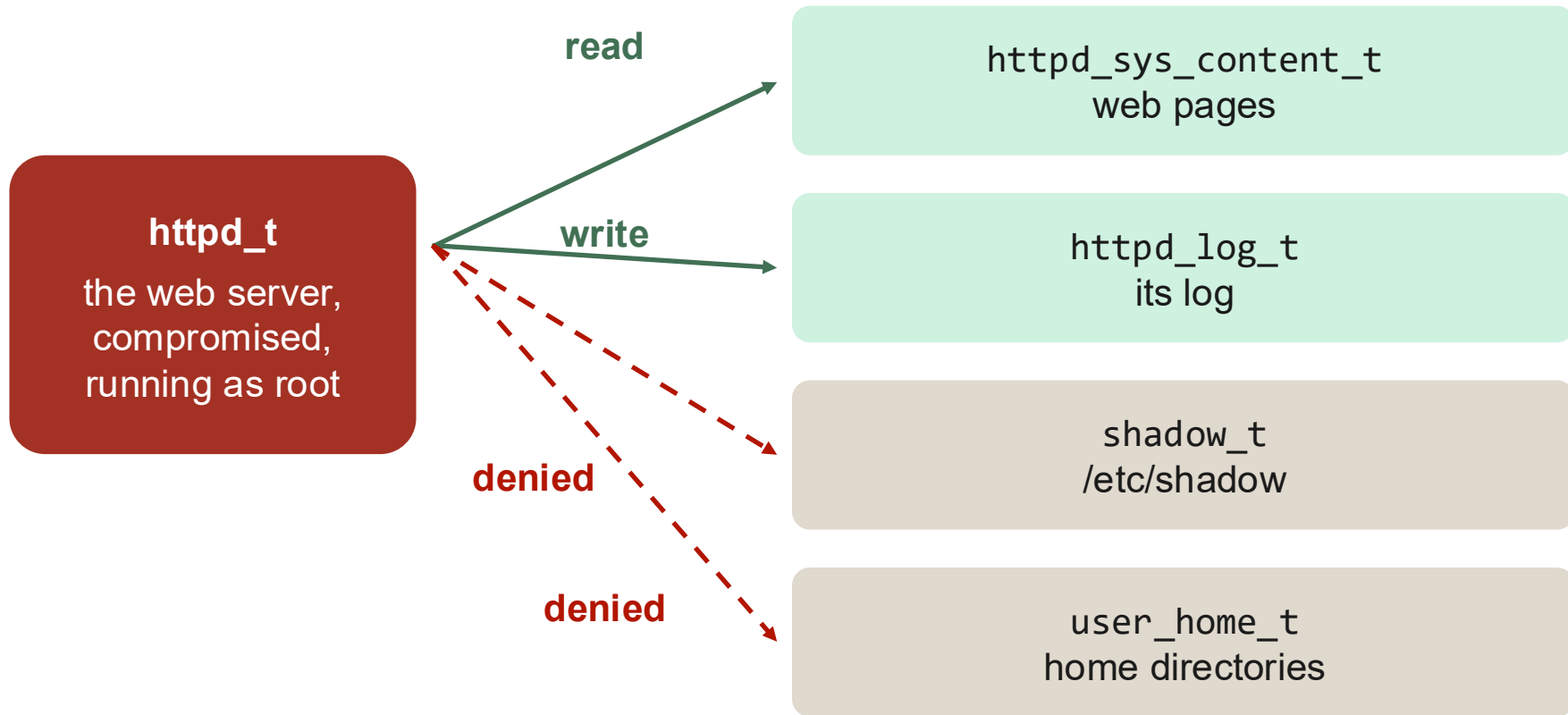
	Type 1	Type 2	Type 3	httpd_sys_content_t
Domain 1	open, read			
Domain 2		open, write		
Domain 3				
httpd_t				read, getattr, open

 **policy**

```
allow httpd_t httpd_sys_content_t:file { read getattr open };
```

the web server domain may read, examine, and open files of the web content type

SELinux Confining a Web Server



Type Enforcement in Use

SELinux

- Released by the NSA in December 2000
- In the mainline kernel since 2003.
- Default on Red Hat Enterprise Linux and Fedora.

Android

- Enforced for every process since 5.0 in 2014.
- Every app runs in a domain restricted to its own files.

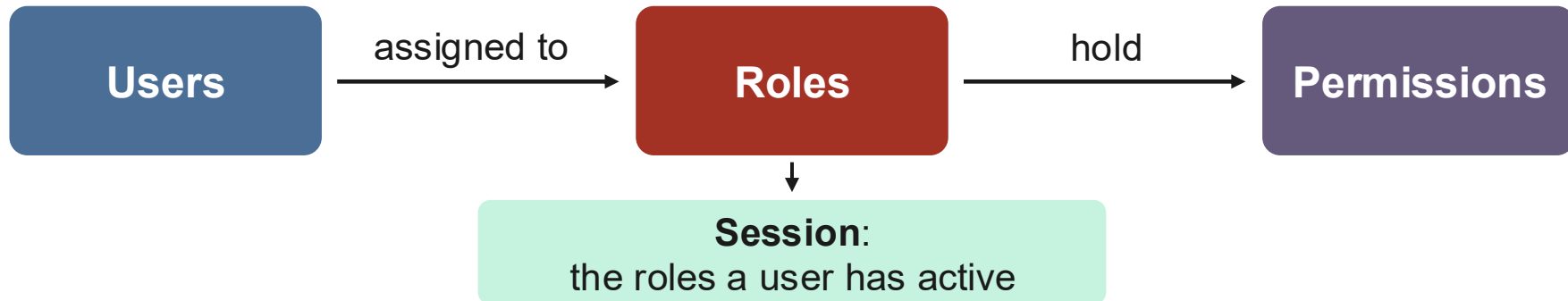
AppArmor

- The alternative on Ubuntu and Debian
- Confines a program by the paths it may access rather than by types.
(we'll cover this later)

A missing rule makes a program fail with an ordinary permission error that never mentions SELinux.

Administrators who cannot find the cause often disable SELinux.

Role-Based Access Control



A user never holds a permission

directly. Rights follow jobs: nurse, billing clerk, administrator.

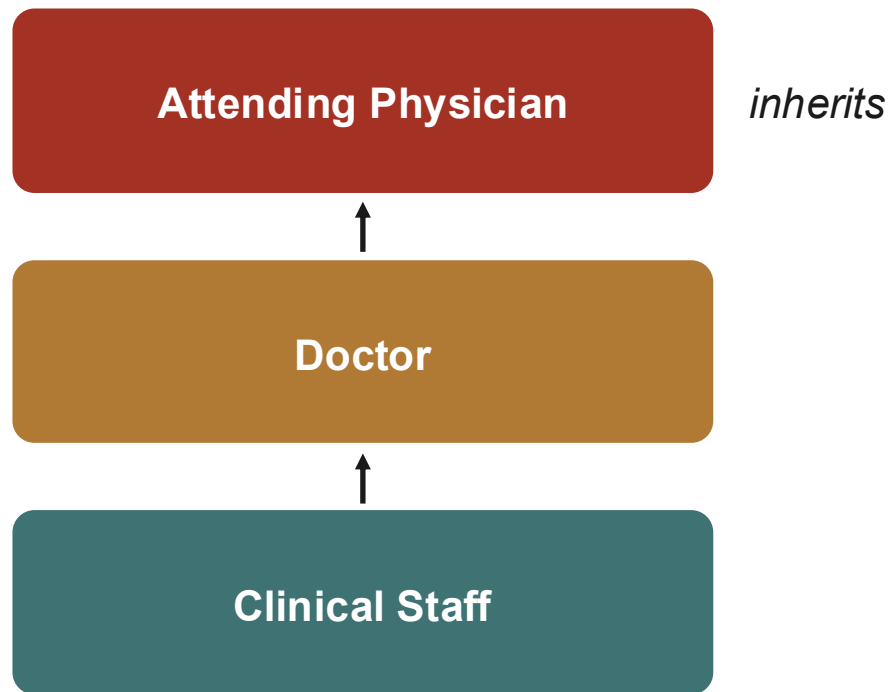
Administrative scale

50 roles turn 2.5 million possible pairs into about 11,000 assignments.

Removing an employee

is one operation, and it removes every right at once.

Role Hierarchies and Separation of Duties



Role hierarchy

A senior role inherits a junior one's permissions, so common permissions are defined once.

It mirrors the organization chart.

Separation of duties

No person performs every step of a sensitive transaction.

Static: the two roles cannot both be assigned.

Dynamic: both can be held, but not activated in one session.

Roles Are Not Groups

A group is a set of **users**.

A file's owner decides what the group may do with the file.

A role is a set of **permissions**.

An administrator decides which users hold it.

“What can a nurse do?”

RBAC system: the list of permissions in the *Nurse* role, available in one lookup.

Group-based system: has to be assembled by searching every object for entries that mention the nurses group.

Reviewing a person's access means producing that list.

Two Meanings of Mandatory

The broad sense

Any access control that users cannot override.

By this definition the Chinese Wall model is mandatory, and so is RBAC when only an administrator assigns roles.

The narrow sense

Label-based control of information flow, as in the Orange Book and the Common Criteria: *Bell-LaPadula* and *Biba*.

SELinux counts, since domains and types are labels the kernel compares.

Security engineers who say *MAC* without qualification usually mean the narrow sense.

Access Control Models Compared

Model	The label	The rule	Protects
Discretionary	owner, entries	the owner decides	owner's choice
Bell-LaPadula	clearance, classification	no read up, no write down	confidentiality
BLP Multilateral	level plus compartments	read what you dominate	need to know
Biba	integrity level	no read down, no write up	integrity
Chinese Wall	dataset, conflict class	no access across a class	conflicts
Type enforcement	domain, type	only what a rule allows	confinement
Role-based	role	permissions follow roles	scale

One System, Several Models

Permission model

Enforced by the Android framework. May this app use the camera or read contacts? Prompts the user.

SELinux type enforcement

In the Linux kernel. Confines every app's process to the files and services the policy grants apps, whatever its user ID.

Unix permissions

Every app gets its own user ID, so the discretionary bits keep apps out of each other's files.

Each layer catches what the one below it cannot express. **An Android phone runs all three; a Linux server or a Windows machine runs two.** How a program is confined beyond the kernel's permissions, and how an app's permissions are enforced, is the subject of a later lecture.

The End