PennState

CSE 443: Introduction to Computer Security

Module: System Security

Access Control

Prof. Syed Rafiul Hussain
Department of Computer Science and Engineering
The Pennsylvania State University

Acknowledgements: Some of the slides have been adopted from Trent Jaeger (Penn State)

Why authenticate?

- Why do we want to verify the identity of a user?

Access Control

- Method for restricting the operations that processes may perform on a computer system
  - aka Authorization

A Brief History

- Early computing systems had no isolation
  - Shared memory space
  - Shared file space
- Some physical limitations made this OK
  - Batch processing
  - Load the tape/disk for the application
  - Network? What network?
- In the mid-60s people started to work on ‘multiuser’ or ‘time-sharing’ systems
  - What about a bug?
  - What about my data?

Multiprogrammed Systems

- Multics project
  - AT&T, MIT, Honeywell, etc.
  - General purpose, multi-user system
  - Comprehensive security
    - Hardware protection
    - Subject labeling
    - Permission management
- UNIX project
  - Spin-off of Multics project
    - When AT&T left
  - A stripped-down multiuser system

Access Control

- Why do you need access control?

Access Control

- Why do you need access control?
  - Protection
    - Prevent errors - oops, I overwrote your files
  - Security
    - Prevent unauthorized access under all conditions

Access Control

- What is needed for "security"?
  - Protect the process - limit others' access to your resources
  - Confine the process - limit your access to others' resources

Control Access

PennState

- An identity permits access to resources
- In computer security this is called
  - *Access control*
  - *Authorization*
- In authorization, we talk about:
  - Subjects (for whom an action is performed)
  - Objects (upon what an action is performed)
  - Operations (the type of action performed)
- Authorization limits a *subject*'s access perform an *operation* on an *object*
  - The combination of object and operations allowed are called a *permission*

Access Control Policy

- What is access control policy?
  - Check whether a process is authorized to perform perform operations on an object
- Authorize
  - Subject: Process
  - Object: Resource that is security-sensitive
  - Operations: Actions taken using that resource
- An object+operations is called a permission
  - Sets of permissions for subjects and objects in a system is called an access control policy

Access Control Policy

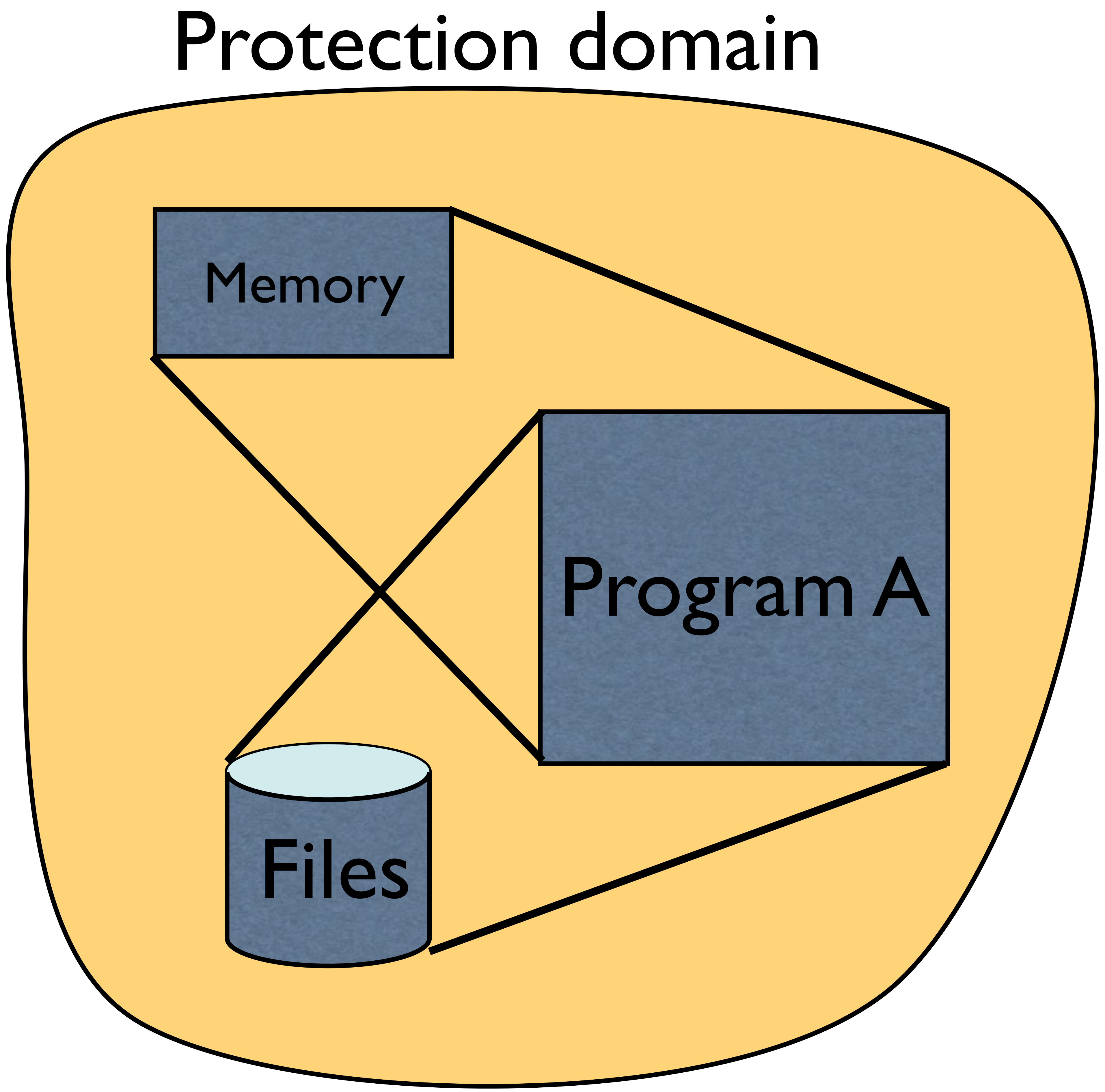
- Access control policy determines what *operations* a particular *subject* can perform for a set of *objects*
- It answers the questions
  - E.g., do you have the permission to read /etc/passwd
  - Does Alice have the permission to view the CSE website?
  - Do students have the permission to share project data?
  - Does Dr. Hussain have the permission to change your grades?

- An Access Control Policy answers these questions

Access Control Concepts

- Subjects are the active entities that do things
  - E.g., you, Alice, students, Prof. Jaeger
- Objects are passive things that things are done to
  - E.g., /etc/passwd, CSE website, project data, grades
- Operations are actions that are taken
  - E.g., read, view, share, change

Protection Domains



- A protection domain specifies the set of resources (objects) that a process can access and the operations that the process may use to access such resources.
- How is this done today?
  - Memory protection
  - E.g., UNIX protected memory, file-system permissions (rwx…)Process memory

What should the protection domain of each process be?

Policy is defined with respect to the protection domain it governs.

Access Policy Model

- A *protection system* answers authorization queries using a protection state (S), which can be modified by protection state methods (M)
  - Authorization query: Can subject perform requested operation on object? Y/N
- A *protection state* (S) relates subjects, objects, and operations to authorization query results
  - E.g., in mode bits, ACLs, … — the policy
- A *protection state methods* (M) can change the protection state (i.e., policy)
  - Add/remove rights for subjects to perform operations on objects — change the policy

Specifying Policy

- Problem - identify subjects, objects, and operations
  - And authorized permissions for subjects
  - And rules for switching between subjects
- Finer policy is better for security and functionality, but is harder to write and manage

Protection Domains

- Balance function and security
- Functionality
  - Operations to get the job done
- Security
  - Prevent operations that may lead to compromise
- Challenge: Figuring out and specifying authorized operations for each process

Access Matrix

- Describe all possible accesses
  - Operations of (S_2,O_2)
  - E.g., read, write, execute
- Specify which users' processes can access which
- Necessary to specify policy to protect users

	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

The Access Matrix

- Suppose the private key file for J is object O_1
  - Only J can read
- Suppose the public key file for J is object O_2
  - All can read, only J can modify
- Suppose all can read and write from object O_3
- What's the access matrix?

	O_1	O_2	O_3
J	?	?	?
S_2	?	?	?
S_3	?	?	?

Access Control Lists

- System stores
  - Which operations can subjects perform
  - For each object
- *Advantage*: Makes you think about how to protect each object
  - Also, easier to confine subjects as we'll discuss later
- *Disadvantage*: Cannot tell what permissions a particular subject has without looking at each object
  - Process always uses all of its permissions, as we'll discuss later

	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

Capabilities

- System stores
  - Which operations can be performed on each object
  - For each subject
- Advantages and disadvantages are reverse of ACL case, naturally

	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

Authentication and Access

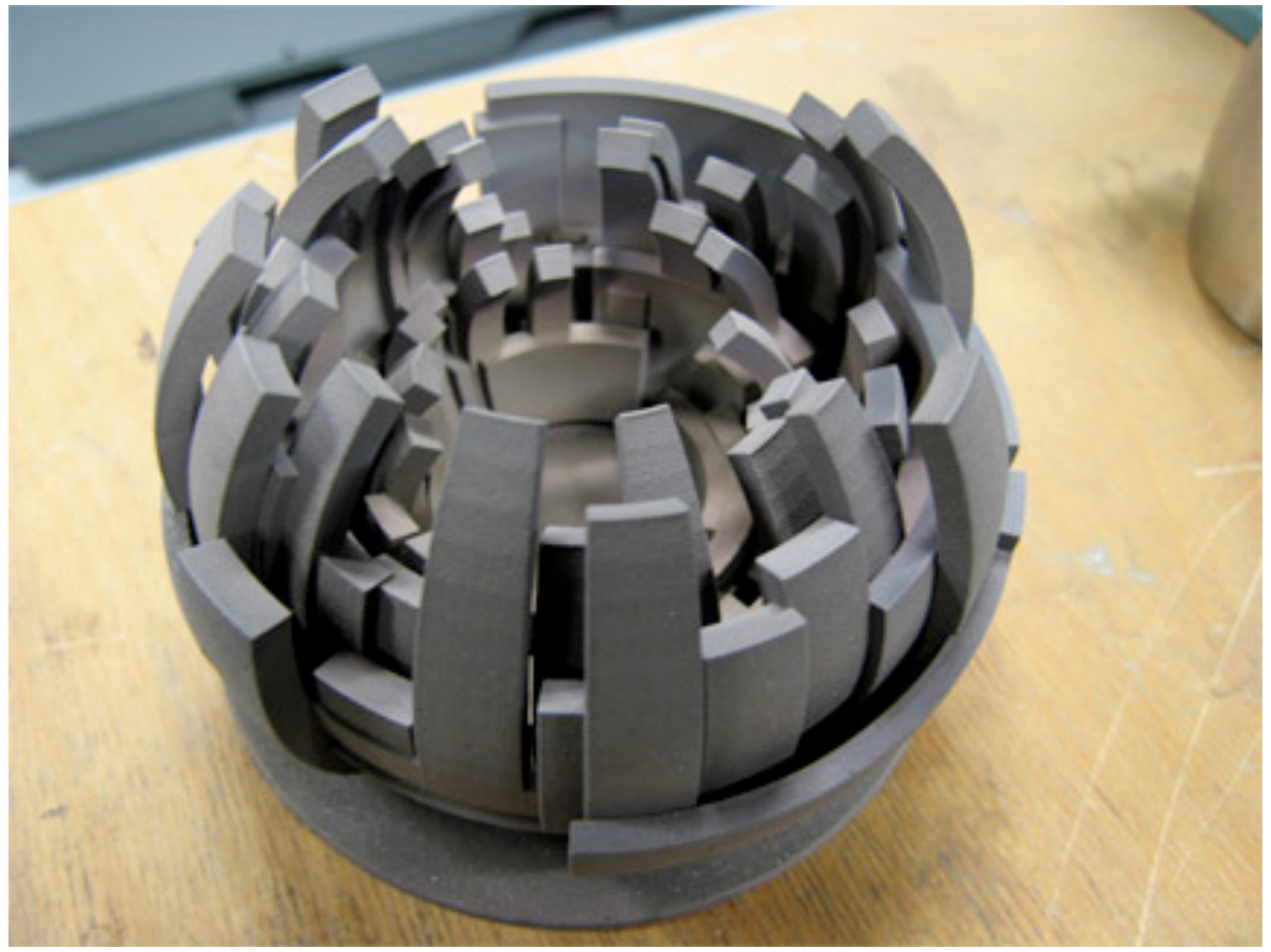
- Authenticate user
  - E.g., login and ssh
  - Verify password or …
- Create processes with appropriate identity (subject)
  - E.g., UNIX user id
- Limit access of these processes using subject
  - E.g., Access control of files based on subject
- Protect one user from another

Authorization Challenges

- Sounds pretty easy, but there are several challenges
  - What's an object?
  - What's an operation?
  - What's a subject?
  - Who's going to manage permissions?

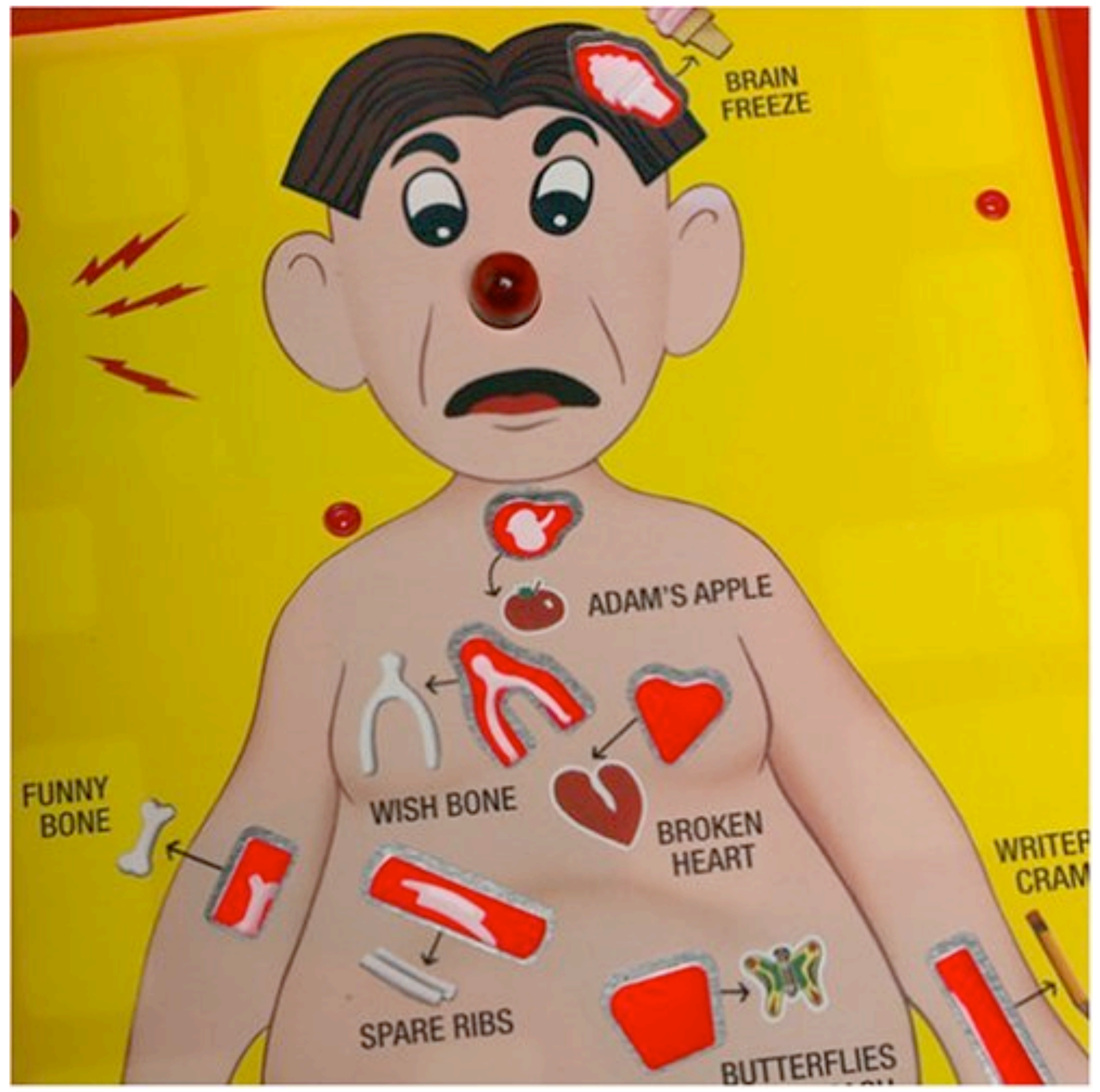
Objects

- What's an object?
  - OS: Many things are files
  - Although not all
- Different software components have their own objects
  - Virtualization
  - Microkernels
  - X Windows
  - Database
  - Apache
  - Logrotate
  - Clouds
  - Social Networks



Operations

- What's an operation?
  - OS: System call
  - Well, not really because many things can happen in a single system call
    - What happens on a file open?
- Security-sensitive operations
  - Any operation that may impact the security of your system
    - Confidentiality, Integrity, Availability
  - A little bit imprecise, but enables some interaction between subjects
- Lots of security-sensitive operations
  - Communication between VMs
  - Cut-and-paste between windows
  - Update a database record
  - Post a message to a social network



Subjects

- What's a subject?
  - OS: System (root/administrator) and Regular Users (you and me)
  - However, even for operating systems this distinction is unsatisfactory
    - System is too coarse
    - User is too coarse/fine
- Why is system too coarse?
  - Might that be the same problem for users?
- Do users even matter to operating systems anymore?
  - How many users on your devices?

Who Are You?

- Identity vs. Permission

Root/Administrative User

- Subjects with full system access
  - Initialize the system
  - Modify the kernel
  - Install software
- Need extra permissions to perform admin
  - Ends up being a lot of processes
- All are part of the trusted computing base

Regular Users

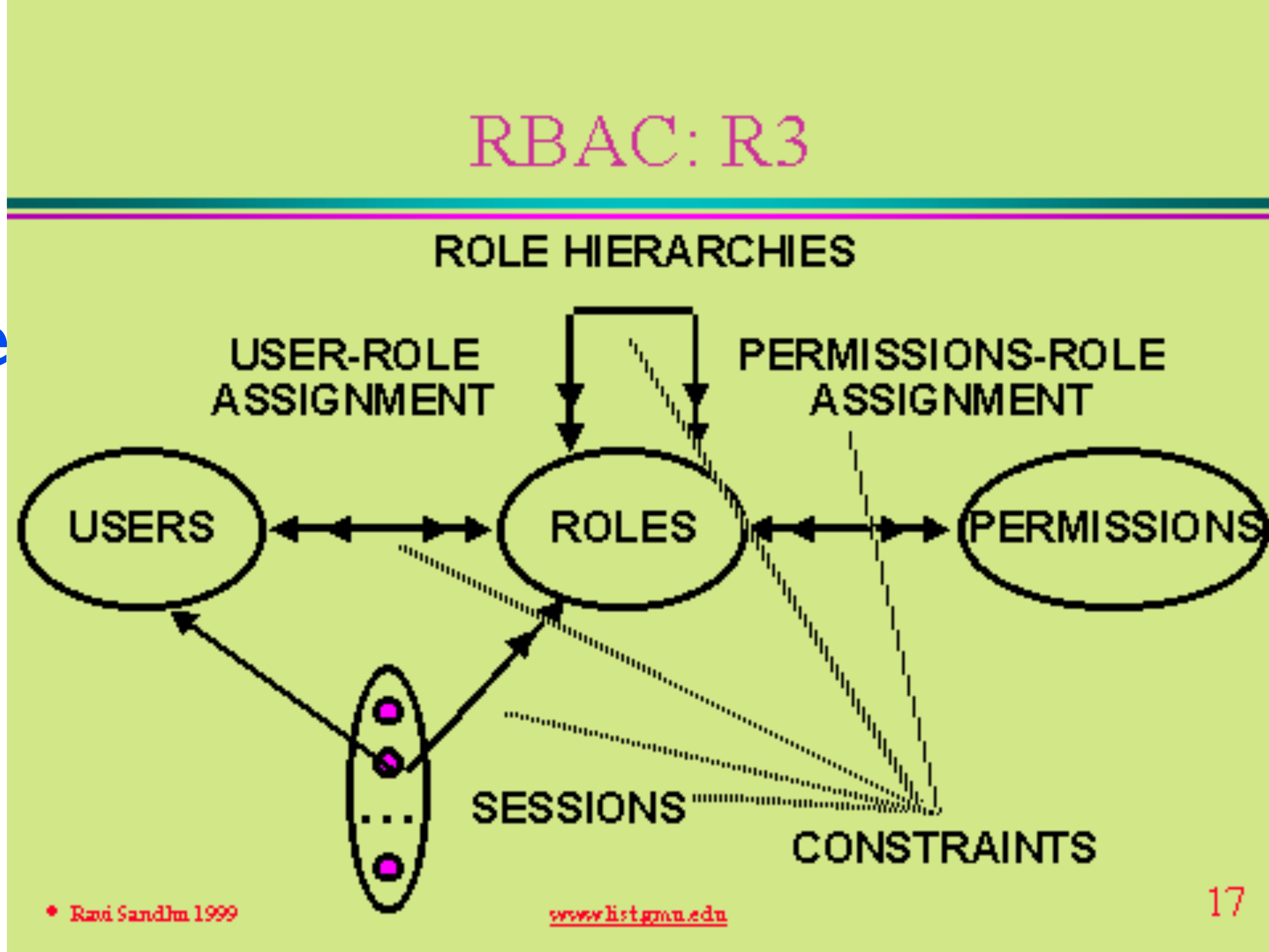
- An unprivileged user
  - However, all your processes run with the same permissions
- What are all the programs that you run?
  - Should they all have full access to any file you can access?
- Sandboxing
  - Run a program with a subset of your permissions

Role-Based Access Control

- Associate permissions with job functions
  - Each job defines a set of tasks
  - The tasks need permissions
  - The permissions define a role
- Bank Teller
  - Read/Write to client accounts
  - Cannot create new accounts
  - Cannot create a loan
  - Role defines only the permissions allowed for the job
- What kind of jobs can we define permission sets for?

Role-based Access Control

- Model consists of two relationships
  - Role-permission assignments
  - User-role assignments
- Assign permissions to roles
  - These are largely fixed
- Assign a user to the roles they can assume
  - These change with each user
  - Administrators must manage this relationship

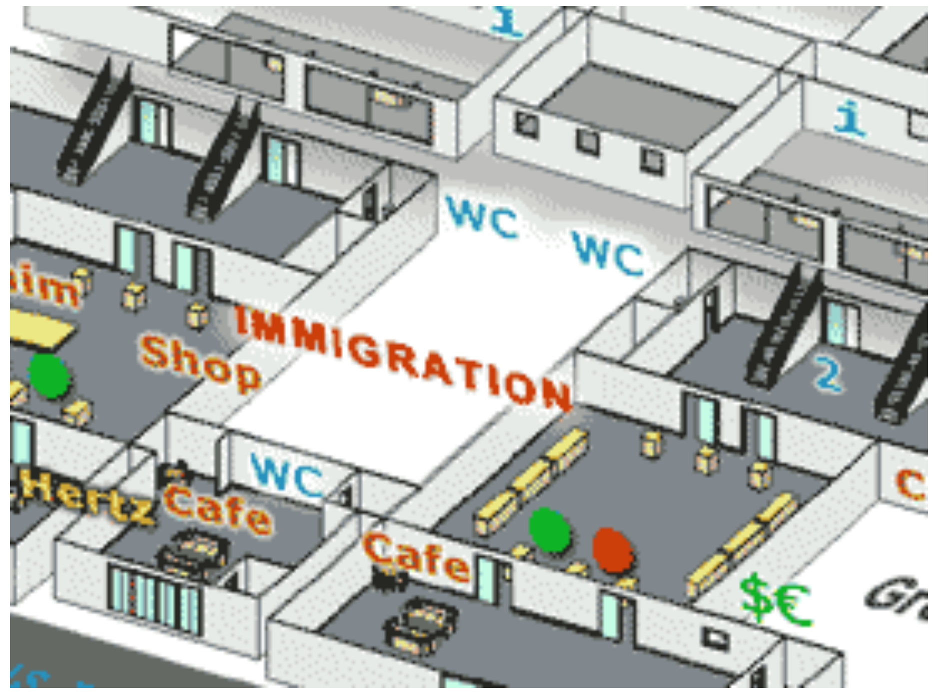


Role Based Access Control

- Most formulations are of the type
  - U: users -- these are the subjects in the system
  - R: roles -- these are the different roles users may assume
  - P: permissions --- these are the rights which can be assumed
- There is a many-to-many relation between:
  - Users and roles
  - Roles and permissions
- Relations define the role-based access control policy

Security Policies

- A security policy specifies the rules of security
  - Some statement of secure procedure or configuration that parameterizes the operation of a system
  - Example: Airport Policy
    - Take off your shoes
    - No bottles that could contain > 3 ozs
    - Empty bottles are OK?
    - You need to put your things through X-ray machine
    - Laptops by themselves, coat off
    - Go through the metal detector



- Goal: prevent on-airplane (metal) weapon, flammable liquid, dangerous objects … (successful?)

Access Policy Enforcement

- A protection state defines what each subject can do
  - E.g., in an access bits --- the policy
- A reference monitor enforces the protection state
  - A service that responds to the query...
- A correct reference monitor implementation meets the following guarantees
  - Tamperproof
  - Complete Mediation
  - Simple enough to verify
- A protection system consists of a protection state, operations to modify that state, and a reference monitor to enforce that state

Access Control Problem

- You run three programs
  - One from the system - passwd
  - One application - editor
  - One from the Internet - email attachment
- What access control policies should be assigned to each program? For protection? For security?
- How to make specifying access control policies easy?

Homework!

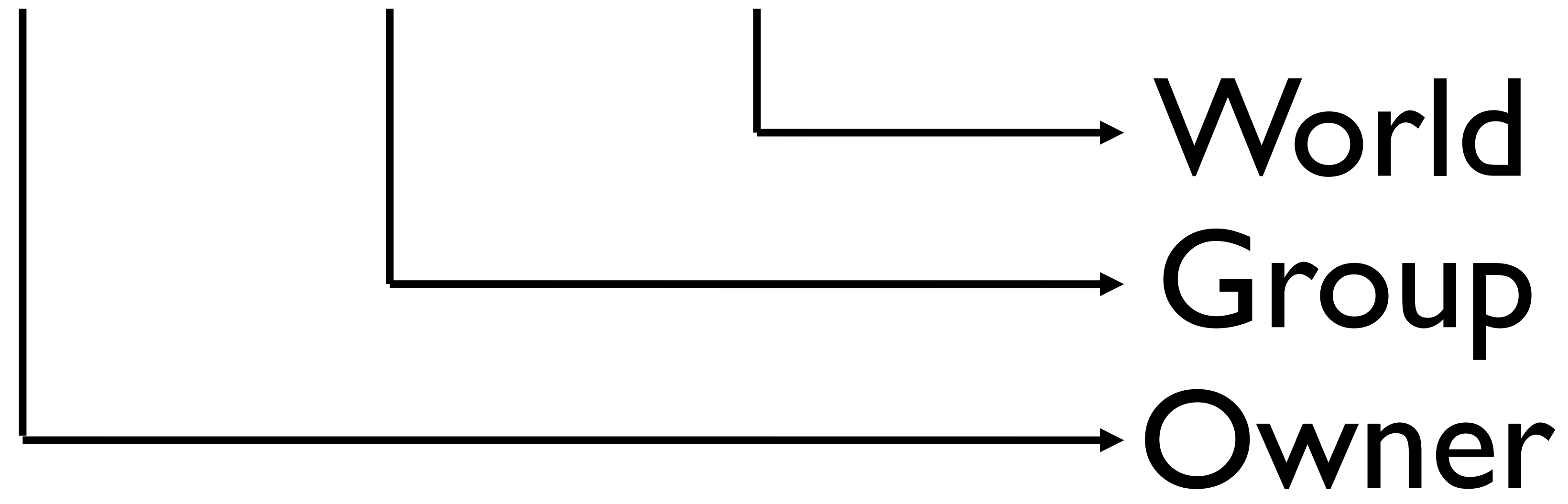
Commodity OS Security

- UNIX and Windows Protection Systems
  - How do they identify subjects/objects to express access control policies?

The UNIX FS access policy

- Really, this is a bit string ACL encoding an access matrix
- E.g.,

rwx rwx rwx



- And a policy is encoded as “r”, “w”, “x” if enabled, and “-” if not, e.g,

rwxrw---x

- Says owner can read, write and execute, group can read and write, and world can execute only.

UNIX UIDs

- Processes and files are associated with user IDs (UIDs)
- File UID indicates its owner (who gets owner perms)
  - Group UID also (who gets group perms)
- Process UID indicates the owner of the process
  - Normal user
  - System (root)
  - Now, some special UIDs for some programs
  - Also, a process may run under multiple Group UIDs

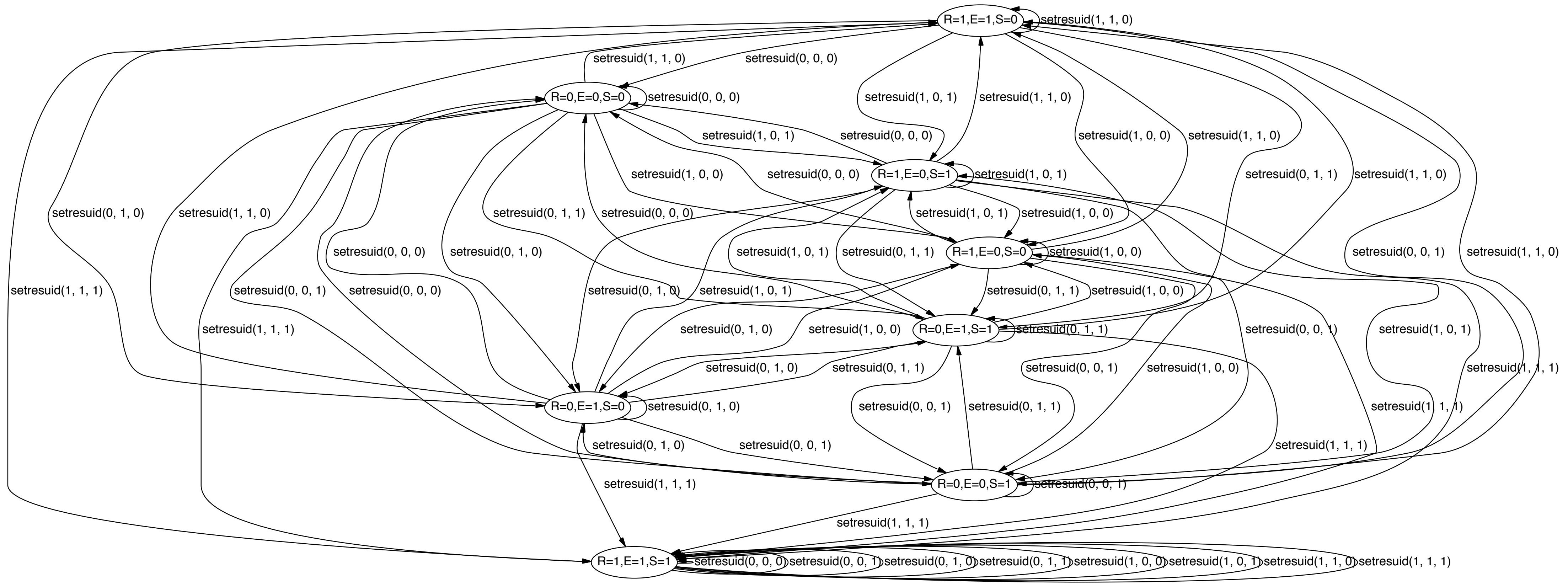
- How do we switch UIDs (e.g., run a privileged program)?

Subjects

- Process
  - User ID (UID)
  - Group ID (GID)
  - Supplementary Groups
- Command: id
  - Provide info for that shell

UNIX UID Transitions

- UNIX represents subjects with a combination of UIDs
  - Effective UID/GID -- used for access control
  - Real UID/GID -- identify real owner of a process - control signals
  - Saved UID -- privileged process - lower privilege temporarily
  - File system UID -- reduce permission to file system
- UID transitions
  - For *login* process: UIDs are root
  - After authentication, the shell's UIDs are: hussain1
  - Exec su: real is hussain1; effective is root



(c) An FSA describing *setresuid* in Linux

UID Transition: Setuid

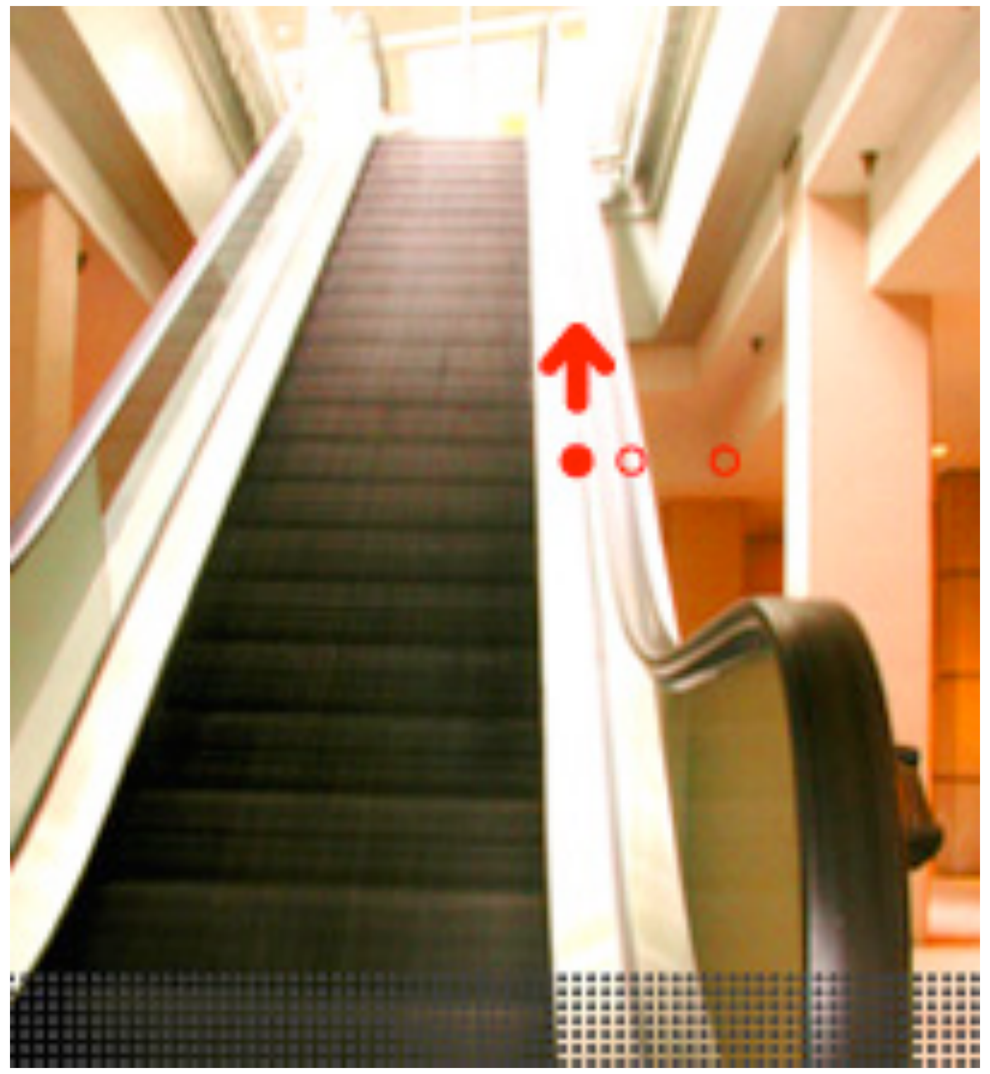
- A special bit in the mode bits
- Execute file
  - Resulting process has the effective (and fs) UID/GID of file owner
- Enables a user to *escalate privilege*
  - For executing a trusted service
- Downside: User defines execution environment
  - e.g., Environment variables, input arguments, open descriptors, etc.
- Service must protect itself or user can gain unauthorized access
  - UNIX services often run as root UID -- many via setuid!

UID Transition: Setuid

- A special bit in the mode bits
- Execute file
  - Resulting process has the effective (and fs) UID/GID of file owner
- Enables a user to escalate privilege
  - For executing a trusted service
- User defines execution environment
  - e.g., Environment variables
- Service must protect itself or user can gain root access

Setuid Execution

- Process A running as
  - UID=X
- Fork process A to create process B
  - Both running with UID=X
- The exec file passwd in process B with setuid bit set and owner of root
  - process A has UID=X
  - process B has UID=root



UNIX Limitations

- How do I create a subject with no permissions?
  - You don't
- How do I give one person access to a file?
  - Make them owner
  - Make a group of one
- How do I give all but one user access to a file?
  - You don't
- Setuid - root or user
- UNIX model is easy to use
  - But, you can't express every case

Changing Effective User ID

- 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

Avoiding Vulnerabilities

- How do we write programs to avoid mistakes that lead to vulnerabilities?
  - Prevent memory errors
  - Detect data handling errors (e.g., truncation)

Access Control == Security?

- Do the Windows and UNIX access control mechanisms provide security for our systems?
  - What is security?

What Is Security?

- *In practice, security methods focus on security or functionality - but not both at the same time!*
- Security Is Foremost
  - Information Flow: No communication with untrusted
  - Advantage: Focus is security
  - Disadvantage: May prevent required functionality
- Restrict based on Functionality
  - Least Privilege: Only rights needed to execute
  - Advantage: Enables required functionality
  - Disadvantage: May not block all attack paths
- Let's look at the two common approaches
  - Least Privilege and Information Flow

Principle of Least Privilege

- Implication 1: you want to limit the process to the smallest possible set of objects
- Implication 2: you want to assign the minimal set of operations to each object

A system should only provide those privileges needed to perform the processes' functions and no more.

- Caveat: of course, you need to provide enough permissions to get the job done.

Least Privilege

- Limit permissions to those required and no more
- Suppose J_1-J_3 must use the permissions below
  - What is the impact of the secrecy of O_1?

	O_1	O_2	O_3
J_1	R	RW	-
J_2	-	R	-
J_3	-	R	RW

Least Privilege

- Can least privilege prevent attacks?
  - Trojan horse
  - Untrusted input

Least Privilege

- Can least privilege prevent attacks?
  - Trojan horse
  - Untrusted input
- Some. No guarantee such attacks are not possible

Secure Protection State

- Set of all protection states P
- Set of secure protection states Q
  - Subjects access to objects to perform operations
  - Meets secrecy, integrity, availability goal
- Example: Protect access to your public key pair
  - Only the protection states in which only you can read the private key file are secure
  - Protection states in which only you may write the public key file are secure
- Problem: Not all processes are necessarily secure
  - Recall programs running on your behalf
    - *Hey, even some programs running on your behalf are not to be trusted with your private key!*

Trusted Processes

- Does it matter if we do not trust some of J's processes?

	O_1	O_2	O_3
J	R	RW	RW
S_2	N	R	RW
S_3	N	R	RW

Secrecy

- Does the following protection state ensure the secrecy of J's private key in O_1?

	O_1	O_2	O_3
J	R	RW	RW
S_2	N	R	RW
S_3	N	R	RW

Integrity

- Does the following access matrix protect the integrity of J's public key file O_2?

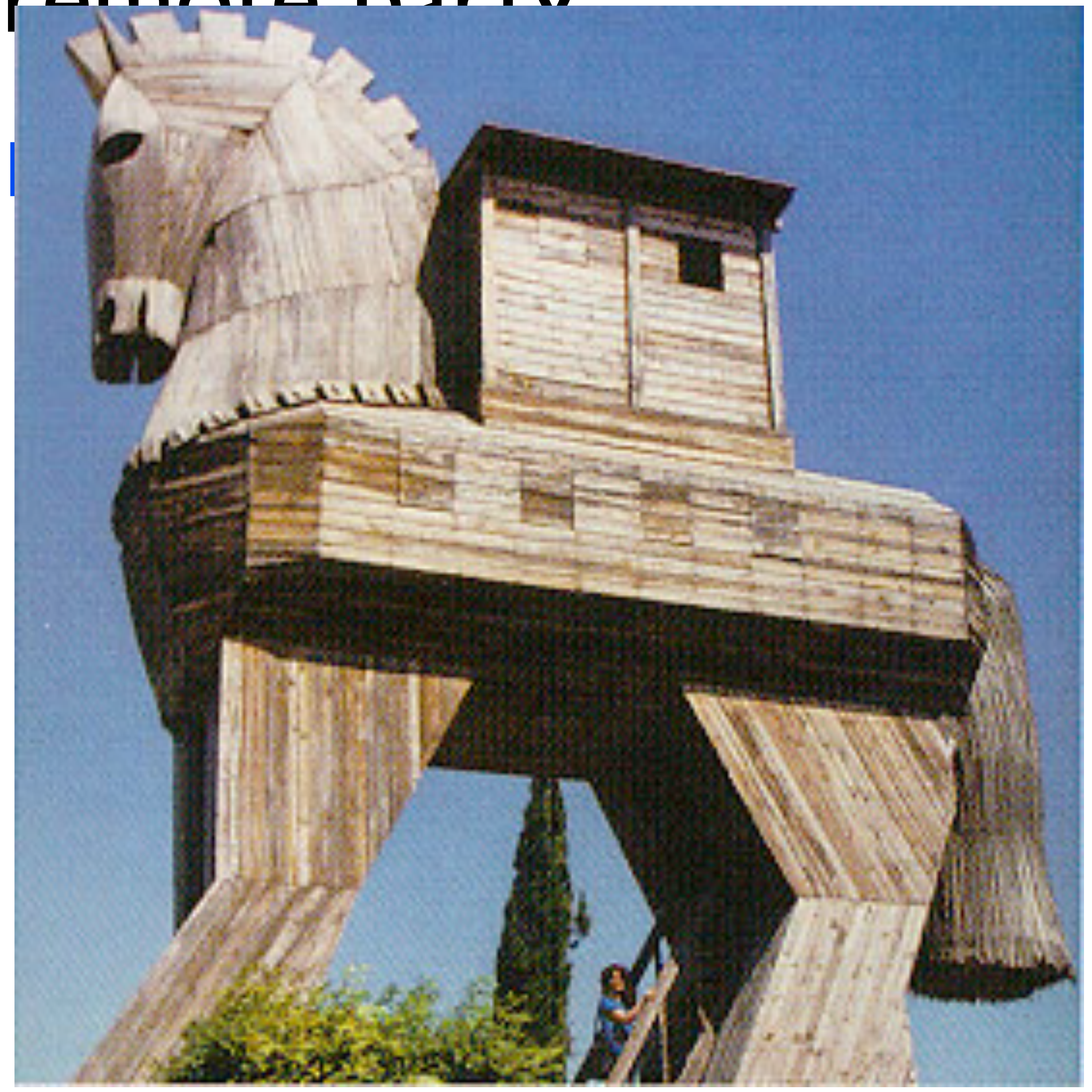
	O_1	O_2	O_3
J	R	RW	RW
S_2	N	R	RW
S_3	N	R	RW

Protection vs Security

- Protection
  - Security goals met under *trusted* processes
  - Protects against an error by a non-malicious entity
- Security
  - Security goals met under *potentially malicious* processes
  - Protects against any malicious entity
- For J:
  - Non-malicious process shouldn't leak the private key by writing it to O_3
  - A malicious process may write the private key to O_3
    - What kind of process might do this?

Trojan Horses

- Trojan horse: A program with a malicious function that masquerades as a benign application
- Suppose you download an editor to modify your secret documents
  - This program can do anything your subject is capable of
  - For example, write the document contents to a remote party
- To prevent leakage, we must block Trojan ho
  - We'll discuss this later



Secrecy Properties

- *Simple-Security Property*
  - Subjects cannot read data that is more secret than their subject is allowed
- **-Security Property*
  - Subjects cannot write data to files that are less secret than they are

- Reverse for protecting integrity
  - Why?

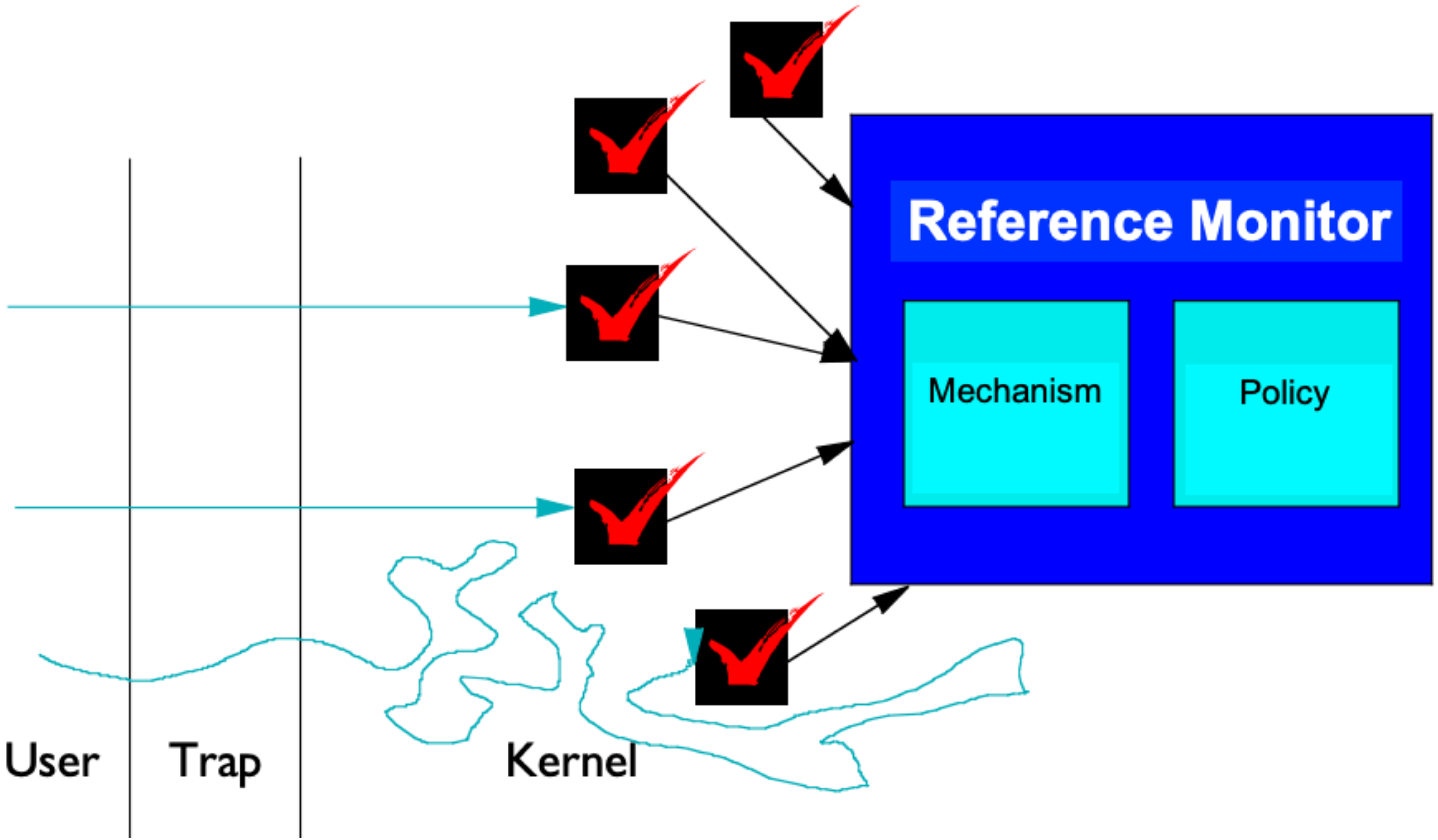
Enforcement Mechanism

- Every system needs to enforce its protection state
- Q: What is required of such an enforcement mechanism?

Reference Monitor Concept

- Properties
  - Complete Mediation of all security-sensitive operations
    - Access control policy is checked before any security-sensitive operation is run
  - Tamperproof
    - No untrusted process can modify the enforcement mechanism or access control policy
  - Simple enough for verification of correctness
    - All code can be verified against correctness criteria
    - Need to enforce a secure protection state

Reference Monitor



Commodity Systems Fail Reference

- Mediation
  - UNIX access control focuses on files, but many other types of system objects enable information flow
    - Windows is better, but UNIX systems have been updated
  - Many setuid processes are not trustworthy
- Tamperproof
  - Protection state transitions may be controlled by untrusted processes
- Correctness
  - UNIX and Windows systems are far too large to verify their correctness

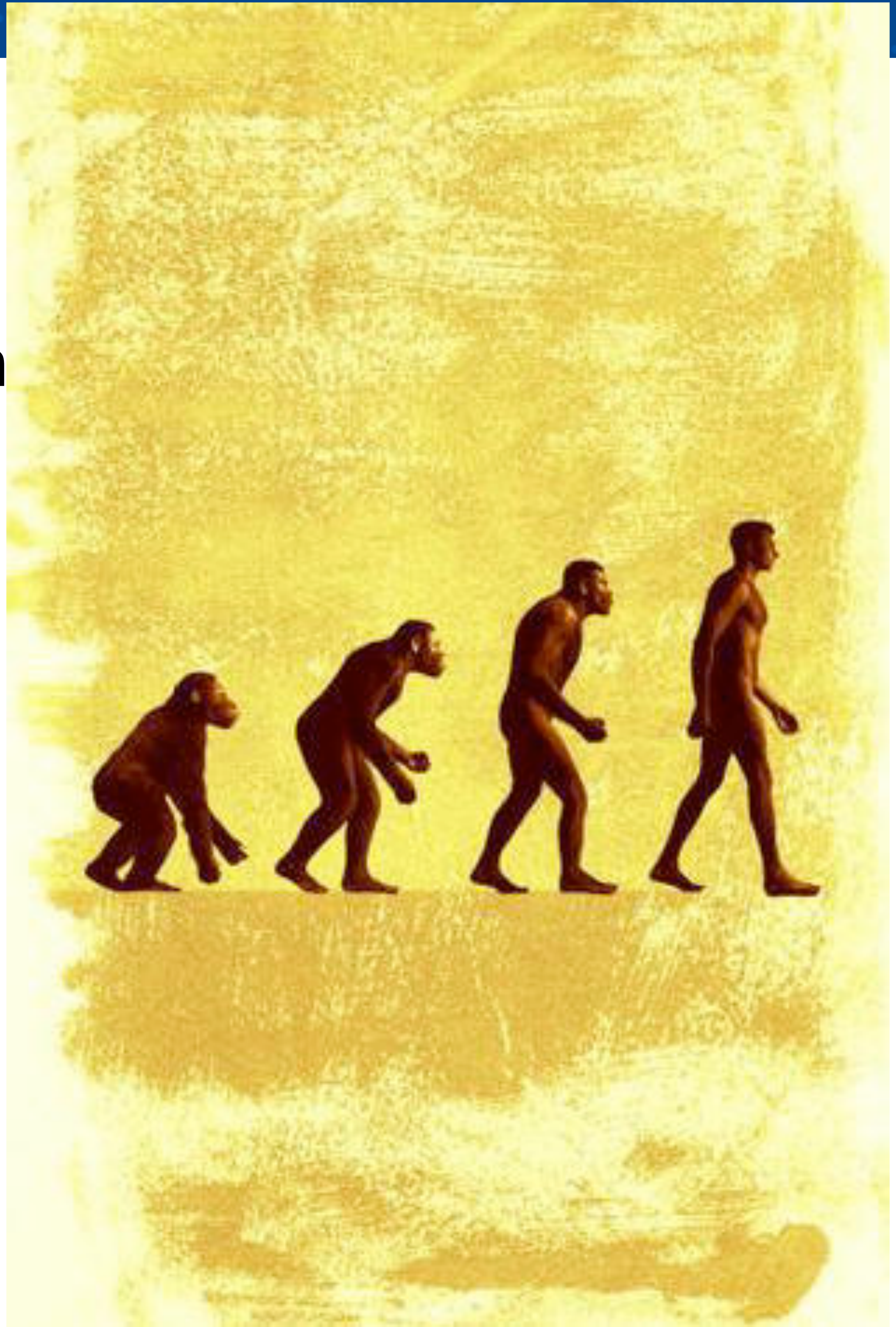
Commodity Systems Fail Reference

- Mediation
  - UNIX access control focuses on files, but many other types of system objects enable information flow
    - Windows is better, but UNIX systems have been updated
  - Many setuid processes are not trustworthy
- Tamperproof
  - Protection state transitions may be controlled by untrusted processes
- Correctness
  - UNIX and Windows systems are far too large to verify their correctness

Protection State Transitions

- Transition
  - From one access matrix state to another
  - Add/delete subject, object, operation assignmen
- Transition principals
  - Owner-driven
  - Delegation
  - Administrator-driven

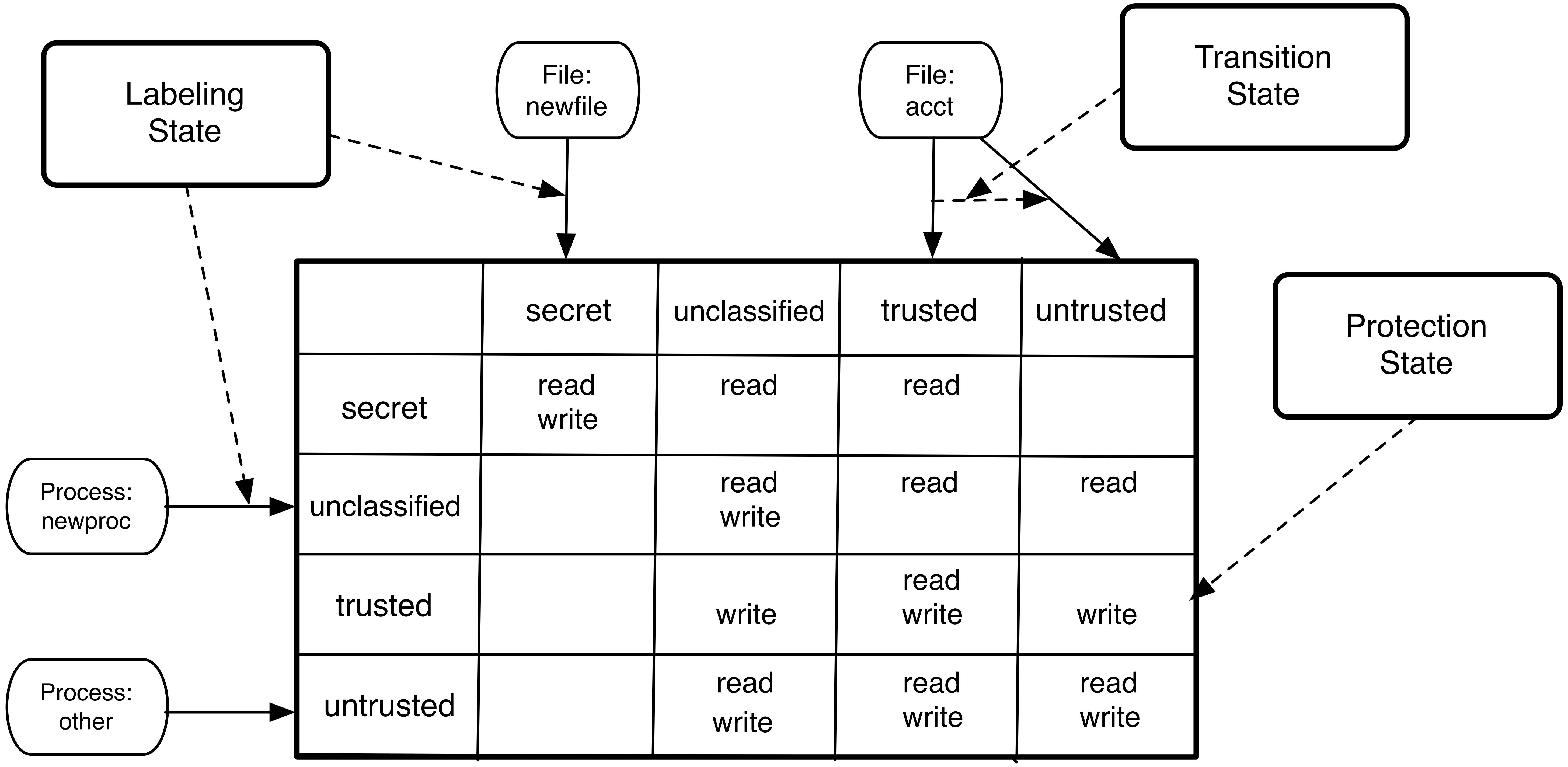
- Attenuation of Rights Principle
  - Can’t grant a right that you do not possess



Safety Problem

- Is there a *general algorithm* that enables us to determine whether a *permission* may be leaked to an *unauthorized user* from any *future protection state*?
- Intuition:
  - From a protection state, users can administer permissions for the objects that they own
  - Enable other subjects to access those objects
- For typical access control models (UNIX)
  - Problem is Undecidable
  - Can also extend representation (new users, objects)
- Practice:
  - Check current protection state for "safety"

Mandatory Protection System



	secret	unclassified	trusted	untrusted
secret	read write	read	read	
unclassified		read write	read	read
trusted		write	read write	write
untrusted		read write	read write	read write

Figure 2.2: A Mandatory Protection System: The *protection state* is defined in terms of labels and is immutable. The immutable *labeling state* and *transition state* enable the definition and management of labels for system subjects and objects.

Mandatory Protection System

A *mandatory protection system* is a protection system that can only be modified by trusted administrators via trusted software, consisting of the following state representations:

- A *mandatory protection state* is a protection state where subjects and objects are represented by labels where the state describes the operations that subject labels may take upon object labels
- A *labeling state* for mapping processes and system resource objects to (subject and object) labels;
- A *transition state* that describes the legal ways that processes and system resource objects may be relabeled.

- An MPS enforces a *mandatory access control* policy
  - User-managed access control is called *discretionary access control*

Mandatory Protection System

- Why is a labeling state necessary?
  - To attach a label to every subject and object dynamically
    - Imagine a system boot process

- Why is a transition state necessary?
  - Necessary for cases where permissions of a process (subject) or access to a file (object) must be changed dynamically
    - Imagine a setuid process

- How does an MPS enable reference monitor?
  - Tamperproofing
  - Utilizes Mediation and Correctness

Integrity Threat

- Untrusted Input
  - *Process reads untrusted input when expects input protected from adversaries*
    - Read a user-defined config file
    - Execute a log file
    - Admin executes untrusted programs

Protection vs Security

- Protection
  - Secrecy and integrity met under *benign* processes
  - Protects against an error by a non-malicious entity
- Security
  - Secrecy and integrity met under *malicious* processes
  - Blocks against any malicious entity from performing unauthorized operations at all times
- Hence, For J:
  - Non-malicious processes shouldn't leak the private key by writing it to O_3
  - A malicious or compromised process may contain a Trojan horse that will write the private key to O_3

Information Flow

- Access control that focuses on information flow restricts the flow of information among subjects and objects
  - Regardless of functional requirements
- Confidentiality
  - Processes cannot read unauthorized secrets
  - Processes cannot leak their own secrets to unauthorized processes
    - Claim: Prevent Trojan horse attacks
- Integrity
  - Processes cannot write objects that are "higher integrity"
  - In addition, processes cannot read objects that are "lower integrity" than they are
    - Claim: Prevent attacks from Untrusted Inputs

Prevent Trojan Horses

- Information Flow Goal
  - Prevent Trojan horse attacks
- Intuition: Prevent flow of secrets to public subjects or objects

Information Flow

- Suppose O_1 must be secret to J_1 only
- No information flow from O_1 to either J_2 or J_3
  - What can you remove to protect the secrecy of O_1?

	O_1	O_2	O_3
J_1	R	RW	-
J_2	-	R	-
J_3	-	R	RW

Denning Security Model

- Information flow model $FM = (N, P, SC, x, y)$
  - N: Objects
  - P: Subjects
  - SC: Security Classes
  - x: Combination
  - y: Can-flow relation
- N and P are assigned security classes ("levels" or "labels")
- $SC_1 + SC_2$ determines the resultant security class when data of security classes SC_1 and SC_2 are combined
- $SC_2 \longrightarrow SC_1$ determines whether an information flow is authorized from security class SC_2 to SC_1
- SC, +, and —> define a lattice among security classes

Denning Security Model

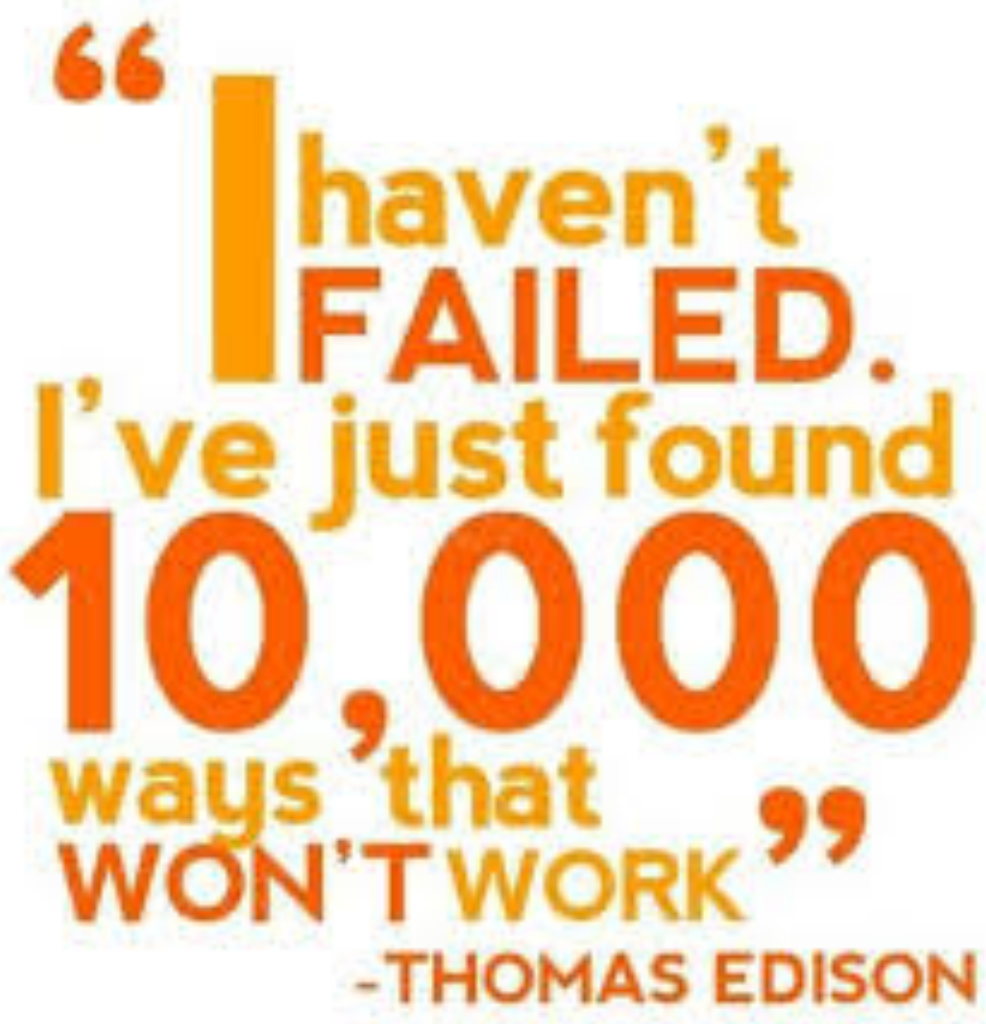
- Preventing Trojan horse attacks
  - Secret files are labeled SC_1 (secret)
  - Secret user logs in and runs processes that are labeled SC_1 (secret)
  - Public objects are labeled SC_2 (public)
  - Only flows within a class or from SC_2 to SC_1 are authorized (public to secret)
  - When data of SC_1 and SC_2 are combined, the resultant security class of the object is SC_1 (public and secret data make secret data)
- How does this prevent a Trojan horse from leaking data?

Information Flow

- Does information flow security impact functionality?

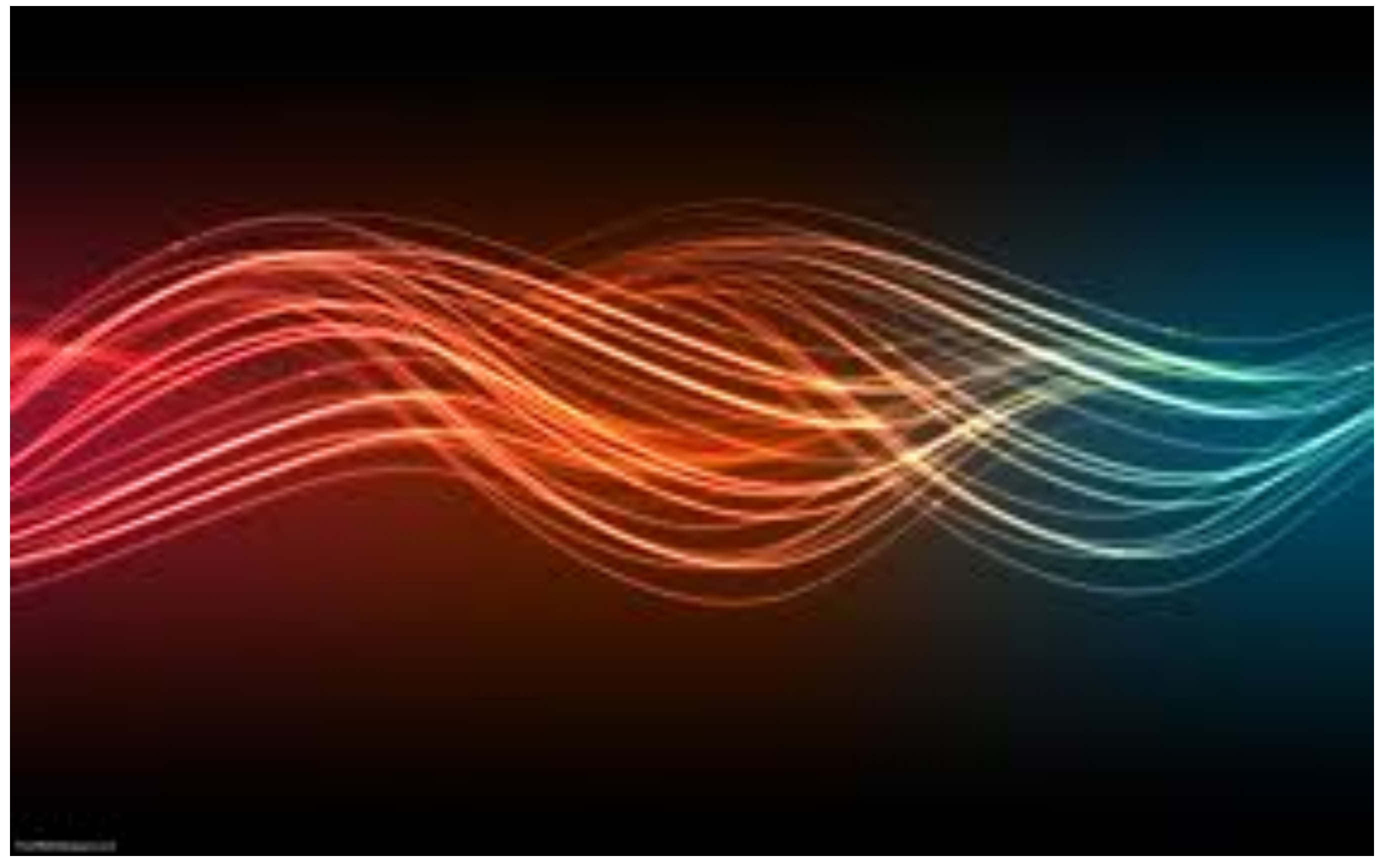
Information Flow

- Does information flow security impact functionality?
  - Yes, so need special processes to reclassify objects
    - Called guards, but are assumed to be part of TCB
      - "Require" formal assurance :-P



Information Flow Models

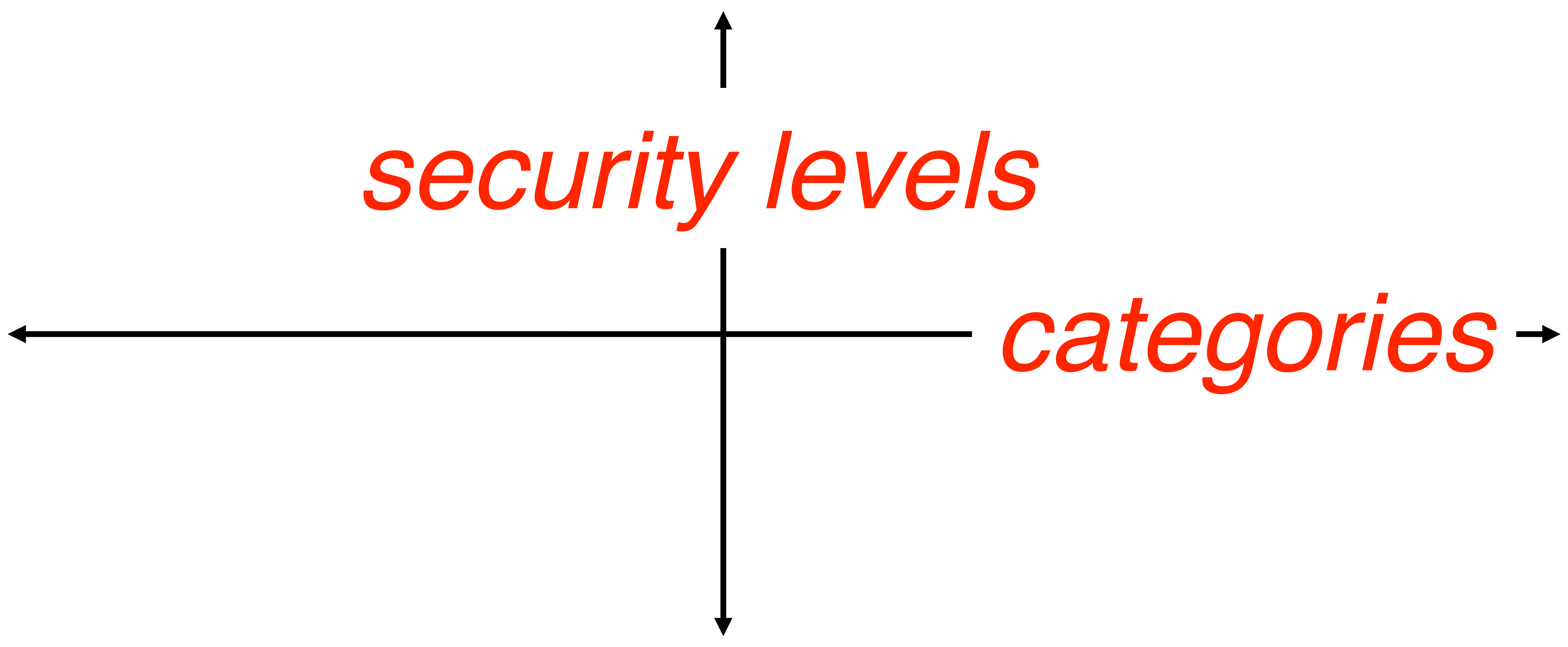
- Secrecy: Multilevel Security, Bell-La Padula
- Integrity: Biba, LOMAC



Multilevel Security

PennState

- A multi-level security system tags all objects and subjects with security tags classifying them in terms of sensitivity/access level.
  - We formulate an access control policy based on these levels
  - We can also add other dimensions, called categories which horizontally partition the rights space (in a way similar to that as was done by roles)



US DoD Policy

- Used by the US military (and many others), uses MLS to define policy
- Levels:

UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

- Categories (actually unbounded set)

NUC(lear), INTEL(igence), CRYPTO(graphy)

- Note that these levels are used for physical documents in the governments as well.

Assigning Security Levels

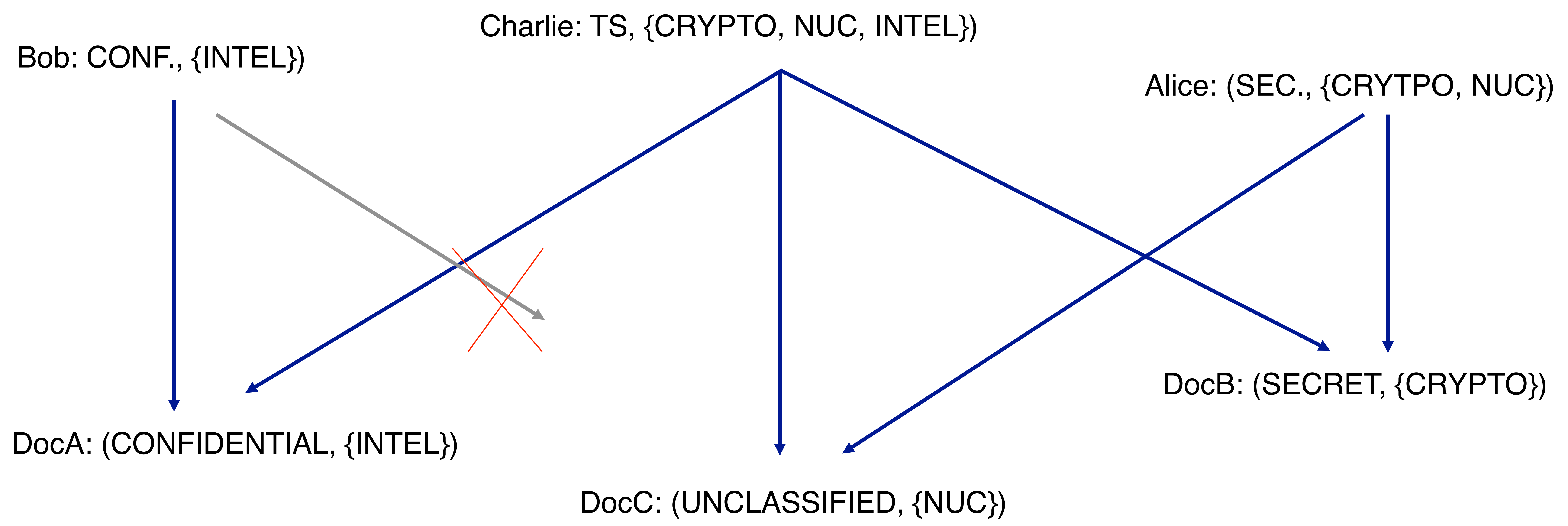
- All subjects are assigned clearance levels and compartments
  - Alice: (SECRET, {CRYTPO, NUC})
  - Bob: (CONFIDENTIAL, {INTEL})
  - Charlie: (TOP SECRET, {CRYPTO, NUC, INTEL})

- All objects are assigned an access class
  - DocA: (CONFIDENTIAL, {INTEL})
  - DocB: (SECRET, {CRYPTO})
  - DocC: (UNCLASSIFIED, {NUC})

Multilevel Security

- Access is allowed if

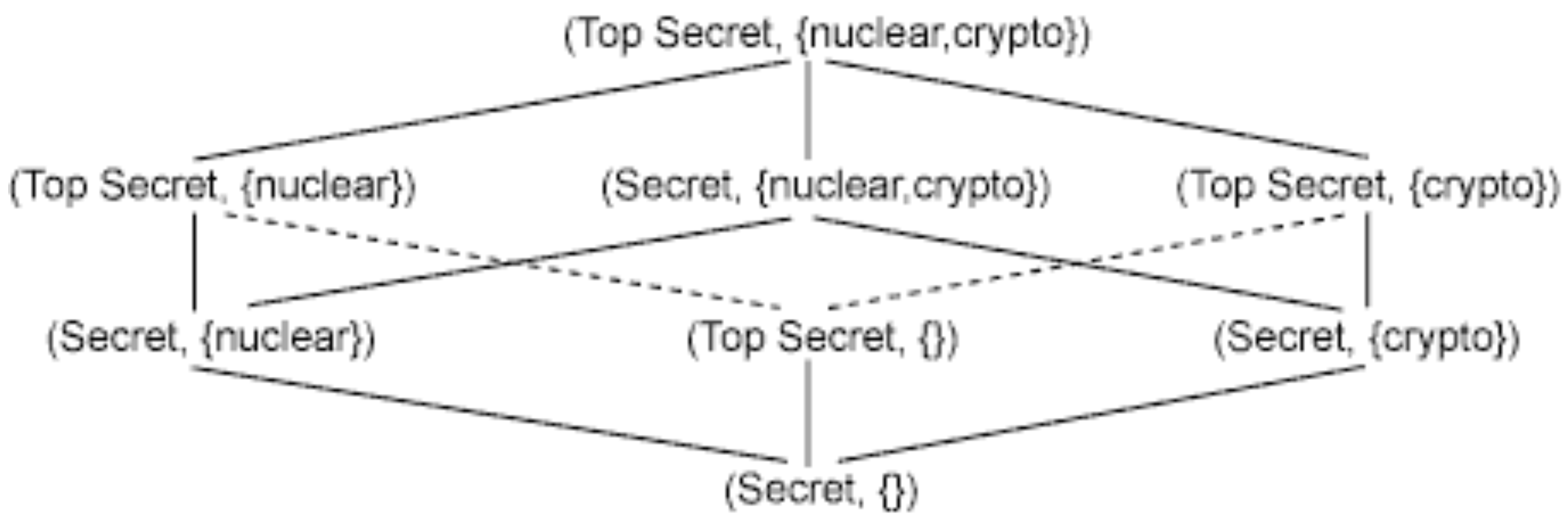
  subject clearance level >= object sensitivity level *and* subject categories ⊇ object categories (*read down*)



- Q: What would *write-up* be?

Bell-La Padula Model

- A Confidentiality MLS policy that enforces:
  - *Simple Security Policy*: a subject at specific classification level cannot read data with a higher classification level. This is short hand for "*no read up*".
  - ** (star) Property*: also known as the confinement property, states that subject at a specific classification cannot write data to a lower classification level. This is shorthand for "*no write down*".



How about integrity?

- MLS as presented before talks about who can "read" a secret document (confidentiality)
- Integrity states who can "write" a sensitive document
  - Thus, who can affect the integrity (content) of a document
  - Example: You may not care who can read DNS records, but you better care who writes to them!
- Biba defined a dual of secrecy for integrity
  - Lattice policy with, "no read down, no write up"
    - Users can only *create* content at or *below* their own integrity level (a monk may write a prayer book that can be read by commoners, but not one to be read by a high priest).
    - Users can only *view* content at or *above* their own integrity level (a monk may read a book written by the high priest, but may not read a pamphlet written by a lowly commoner).

Biba (example)

- Which users can modify what documents?
  - Remember "*no read down*, *no write up*"

Charlie: (TS, {CRYPTO, NUC, INTEL})

Bob: (CONF., {INTEL})

Alice: (SEC., {CRYTPO, NUC})

?????

DocB: (SECRET, {CRYPTO})

DocA: (CONFIDENTIAL, {INTEL})

DocC: (UNCLASSIFIED, {NUC})

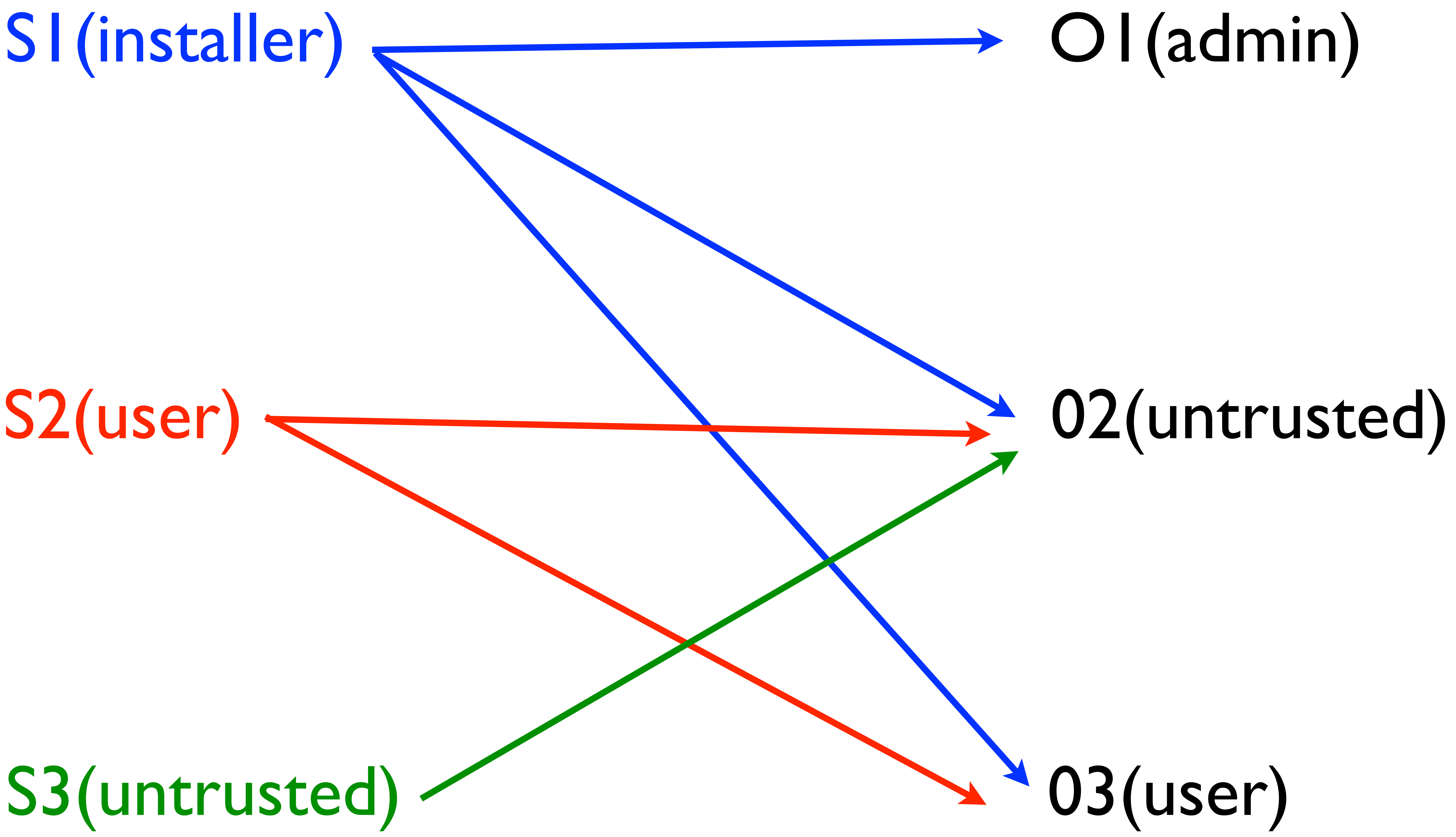
Window Vista Integrity

- Integrity protection for writing
- Defines a series of protection level of increasing protection
  - installer (highest)
  - system
  - high (admin)
  - medium (user)
  - low (Internet)
  - untrusted (lowest)
- Semantics: If subject's (process's) integrity level dominates the object's integrity level, then the write is allowed

Vista Integrity

S1(installer)	O1(admin)
S2(user)	02(untrusted)
S3(untrusted)	03(user)

Vista Integrity



Reduce Integrity Restrictiveness

- Can we allow processes to read lower integrity data without compromising information flow?
  - Still don't trust the process to handle lower integrity inputs without being compromised
- Insight: Could change the integrity level of each process based on the data it accesses

LOMAC

- Low-Water Mark integrity
  - Change integrity level based on actual dependencies

- Subject is initially at the highest integrity
  - But integrity level can change based on objects accessed
- Ultimately, subject has integrity of lowest object read

Integrity, Sewage, and Wine

- Mix a gallon of sewage and one drop of wine gives you?
- Mix a gallon of wine and one drop of sewage gives you?

Integrity is really a contaminant problem: you want to make sure your data is not contaminated with data of lower integrity.

Take Away

- Claim: Traditional access control approaches (UNIX and Windows) do not enforce security against a determined adversary
  - (1) Trojan horses and confused deputies violate security goals
  - (2) DAC models prevent goals from being enforced
- Mandatory Access Control (MAC) is the way these can be achieved
  - MAC policies
    - Information flow models (MLS, Biba)
    - Least privilege MAC is often used (see SELinux)