Goals for Today

- Learning Objective:
  - Contrast strengths/weaknesses of security primitives
  - Understand design concepts of OS-layer Access Control
- Announcements, etc:
  - MP3 Soft Extension: Submit by April 25th for -10pts
  - MP4 due **May 7th**
    - Deadline provides more time than necessary; wanted to give you flexibility
- Schedule:
  - Apr 23 — Security Wrap-Up (lots to cover today)
  - Apr 25 — Final Exam Overview + Review Session
  - Apr 27 — Energy + Power in Operating Systems
  - Apr 30 — Operating System Auditing (feat. Wajih Ul Hassan)
  - May 02 — Final Exam Q&A

CS 423
Operating System Design: Access Control in Operating Systems

Professor Adam Bates
Spring 2018

Security Goals

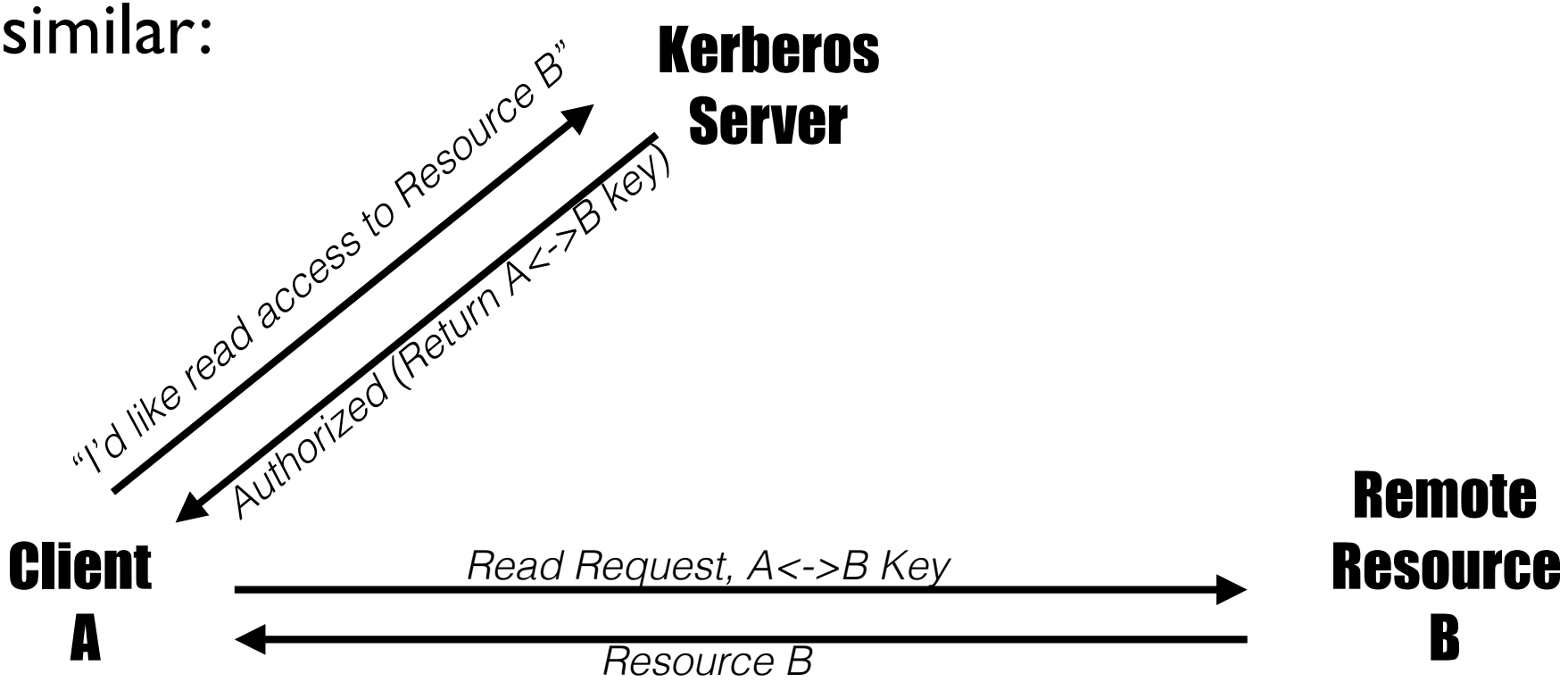
- **Confidentiality**:
  - Prevent data exposure
- **Integrity**:
  - Prevent data tampering
- **Authenticity**:
  - Verify identity of data source
- **Availability**:
  - Preventing denial of service
  - "Unplug computer" != security

Crypto v. Access Control

- Two competing primitives for achieving security
- Both can be used to achieve assurances of data confidentiality, integrity, authenticity

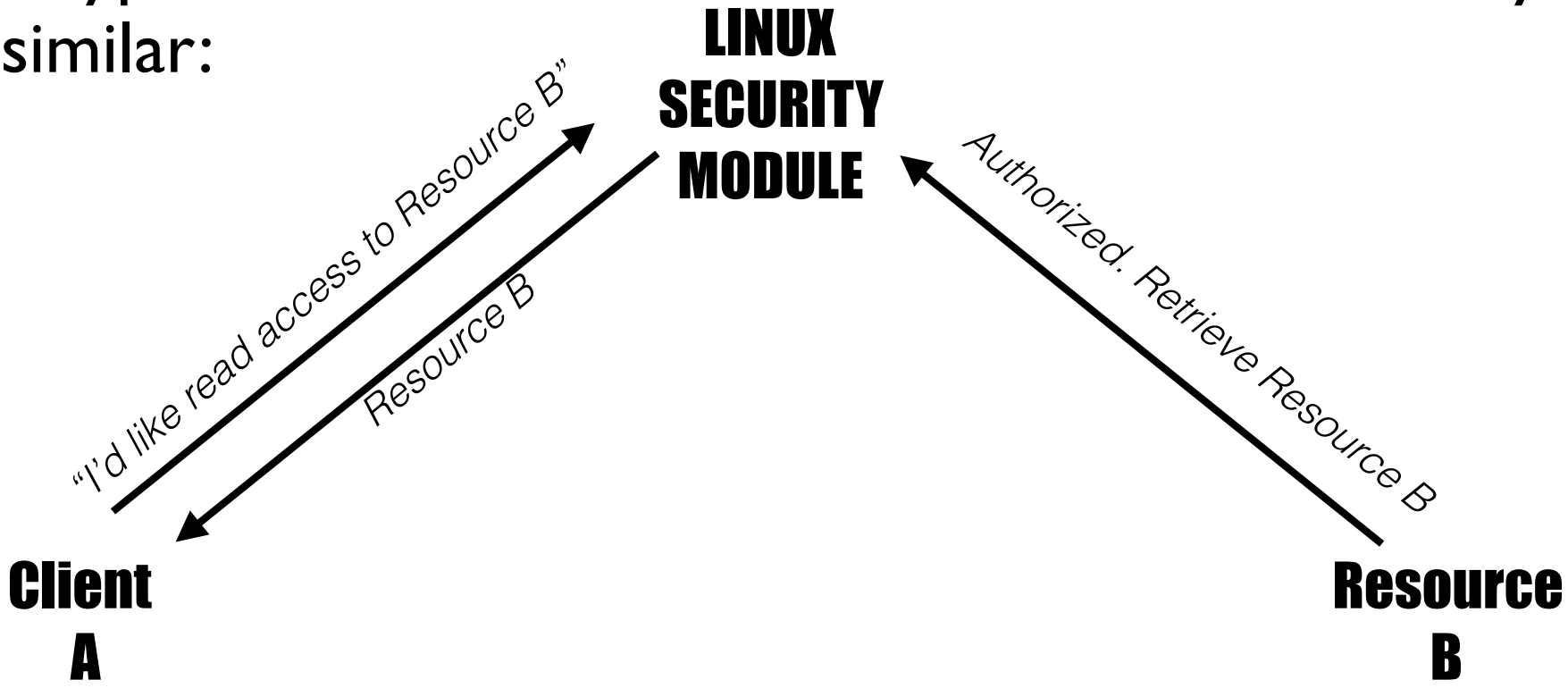
Crypto v. Access Control

- Two competing primitives for achieving security
- Both can be used to achieve assurances of data confidentiality, integrity, authenticity
- Crypto- and AC- based architectures are often very similar:



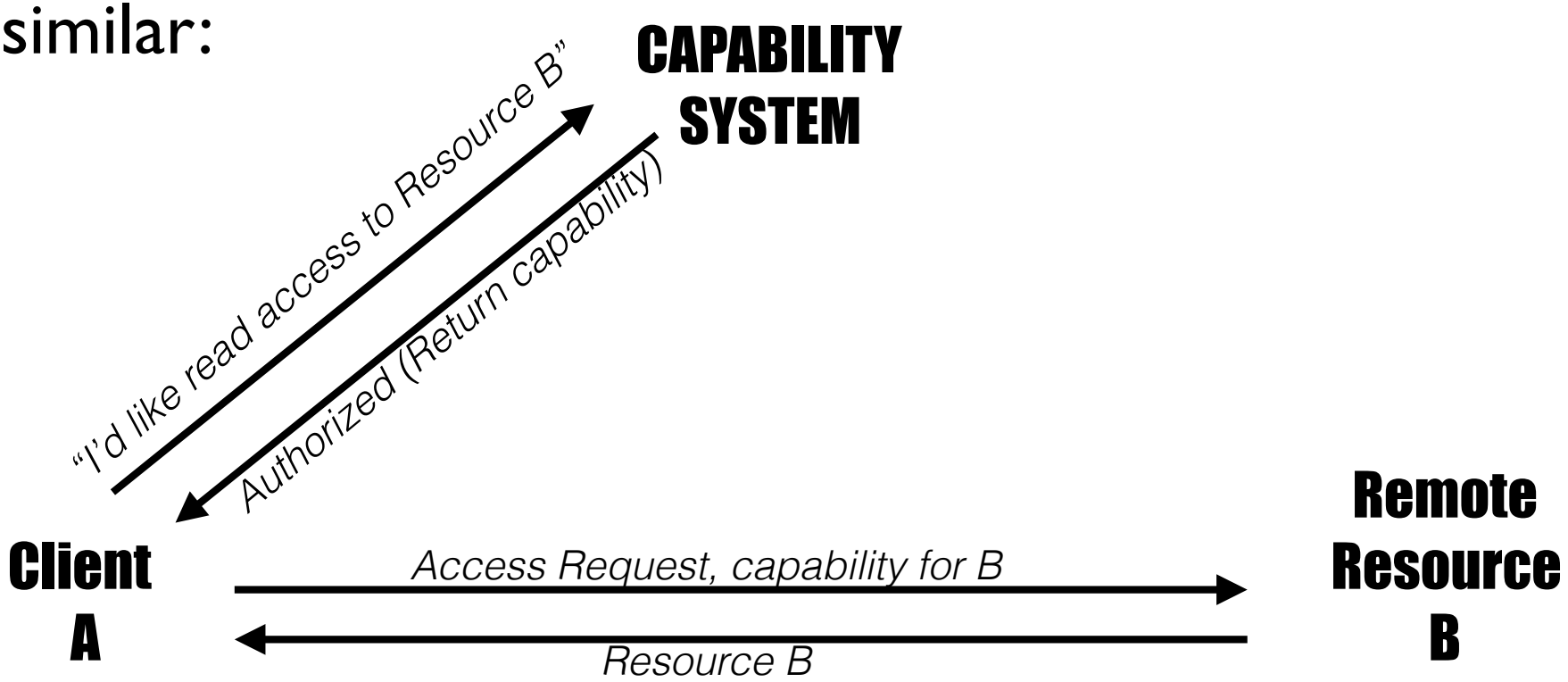
Crypto v. Access Control

- Two competing primitives for achieving security
- Both can be used to achieve assurances of data confidentiality, integrity, authenticity
- Crypto- and AC- based architectures are often very similar:



Crypto v. Access Control

- Two competing primitives for achieving security
- Both can be used to achieve assurances of data confidentiality, integrity, authenticity
- Crypto- and AC- based architectures are often very similar:



Crypto v. Access Control

- **Which primitive should we use in operating systems?**

Cryptography

- Easily supports multiple administrative domains
- Easily supports delegation (i.e., share key)
- Computationally expensive
- Requires key management and distribution mechanisms
- *Requires key security*

Access Control

- Struggles to support multiple administrative domains
- Easily supports centralized global security policy
- Computationally cheap
- Requires policy management mechanisms
- *Requires reference monitor*

Crypto v. Access Control

- Which primitive should we use in operating systems?

Cryptography

- Easily supports multiple administrative domains
- Easily supports delegation (i.e., share key)
- Computationally expensive
- Requires key management and distribution mechanisms
- *Requires key security*

Access Control

- Struggles to support multiple administrative domains
- Easily supports centralized global security policy
- Computationally cheap
- Requires policy management mechanisms
- *Requires reference monitor*

Linux has had broad crypto support since version 2.5…

but in the kernel, access control is king!

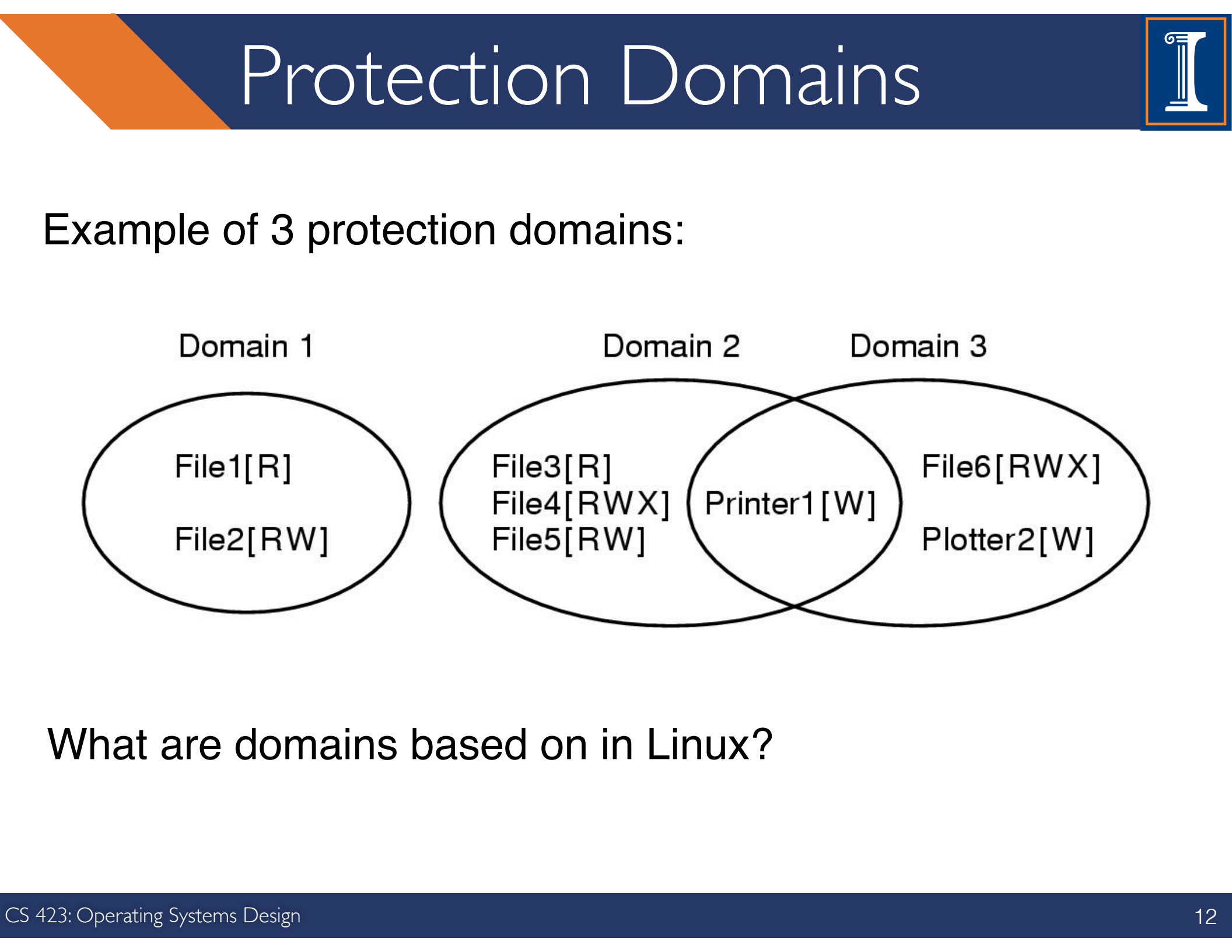
Access Control

- Determine whether a principal can perform a requested operation on a target object
- **Principal/Subject**: user, process, etc.
- **Operation/Action**: read, write, etc.
- **Object**: file, tuple, etc.

Protection Domains

- A computer system is a set of processes and objects
- A process operates within a protection domain
- A protection domain specifies the resources a process may access and the types of operations that may be invoked on the objects.
- The Principle of Least Privilege: The protection domain of a process should be as small as possible consistent with the need of that process to accomplish its assigned task.

Protection Domains

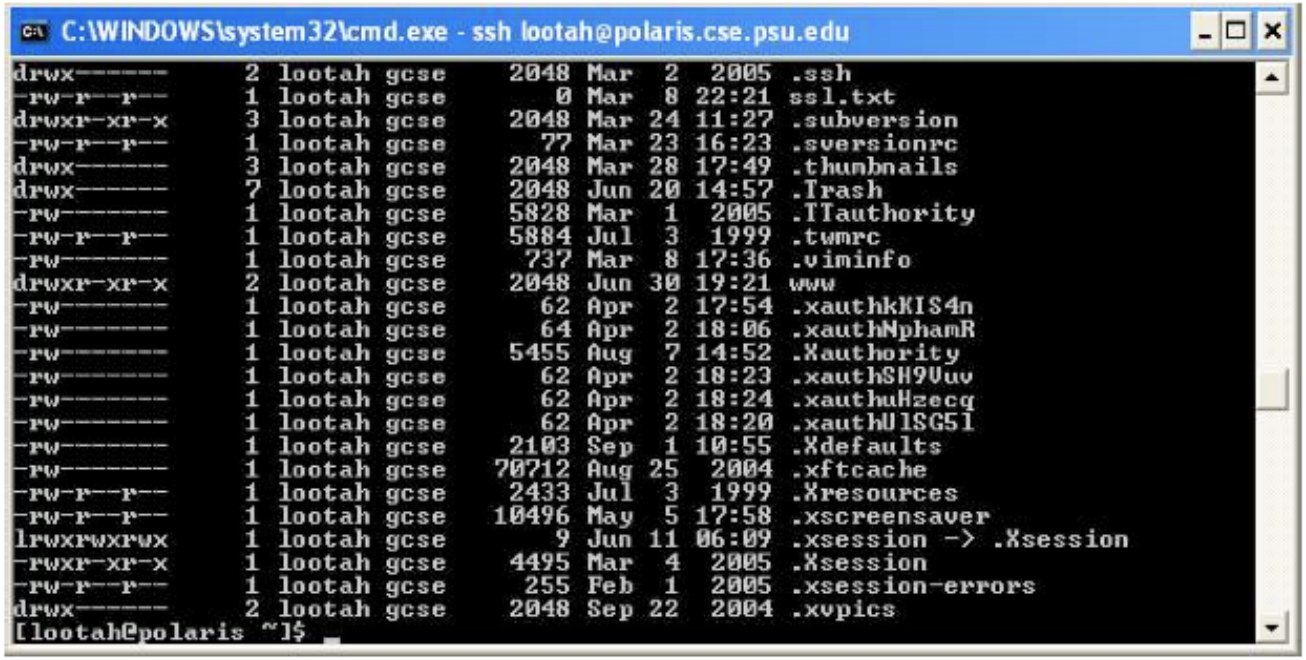


Example of 3 protection domains:

What are domains based on in Linux?

Discretionary Access Control (DAC)

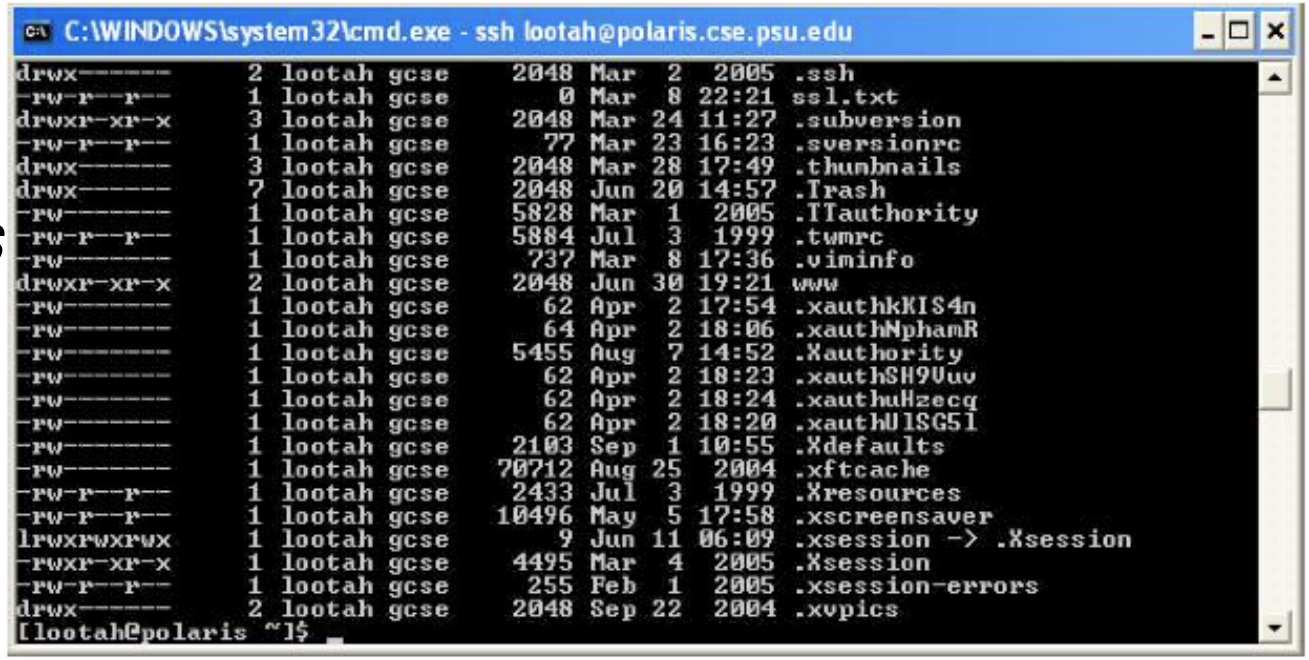
Find the access control!



- Owner of creator of resources specified which subjects have which access to a resources
- Commonly implemented in commercial products
- Access is managed by individual users, not a central security policy

Discretionary Access Control (DAC)

Access Mask defines permissions for User, Group, and Other




```
chmod u=rwx,g=rx,o=r myfile
chmod 754 myfile
```

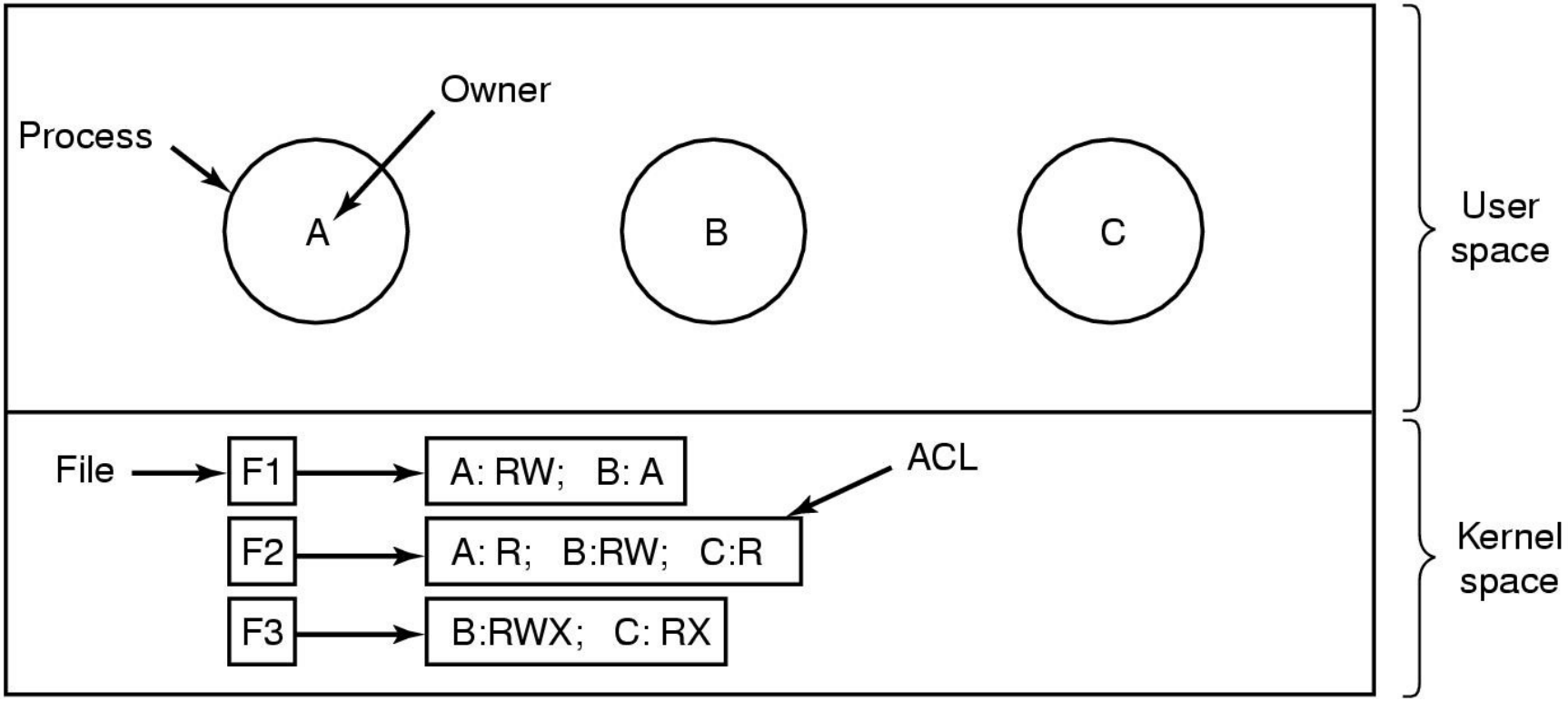
<- Same thing

4 stands for "read",
2 stands for "write",
1 stands for "execute", and
0 stands for "no permission."

Access Control Lists

- Each column in access matrix specifies access for one Object.
- On invocation of a method R on an object O by a process running in a domain D,
- The access control list is searched to check whether D is allowed to perform method R on object O (e.g., allowed to read the file or execute the program)
- A default (e.g., "rest of the world") can be associated with an access list so that any Domain not specified in the list can access the Objects using default methods.
- It is easy for the owner of the Object to grant access to another Domain or revoke access.
- ACL entries can be for individual users or for a group of users.

Access Control Lists

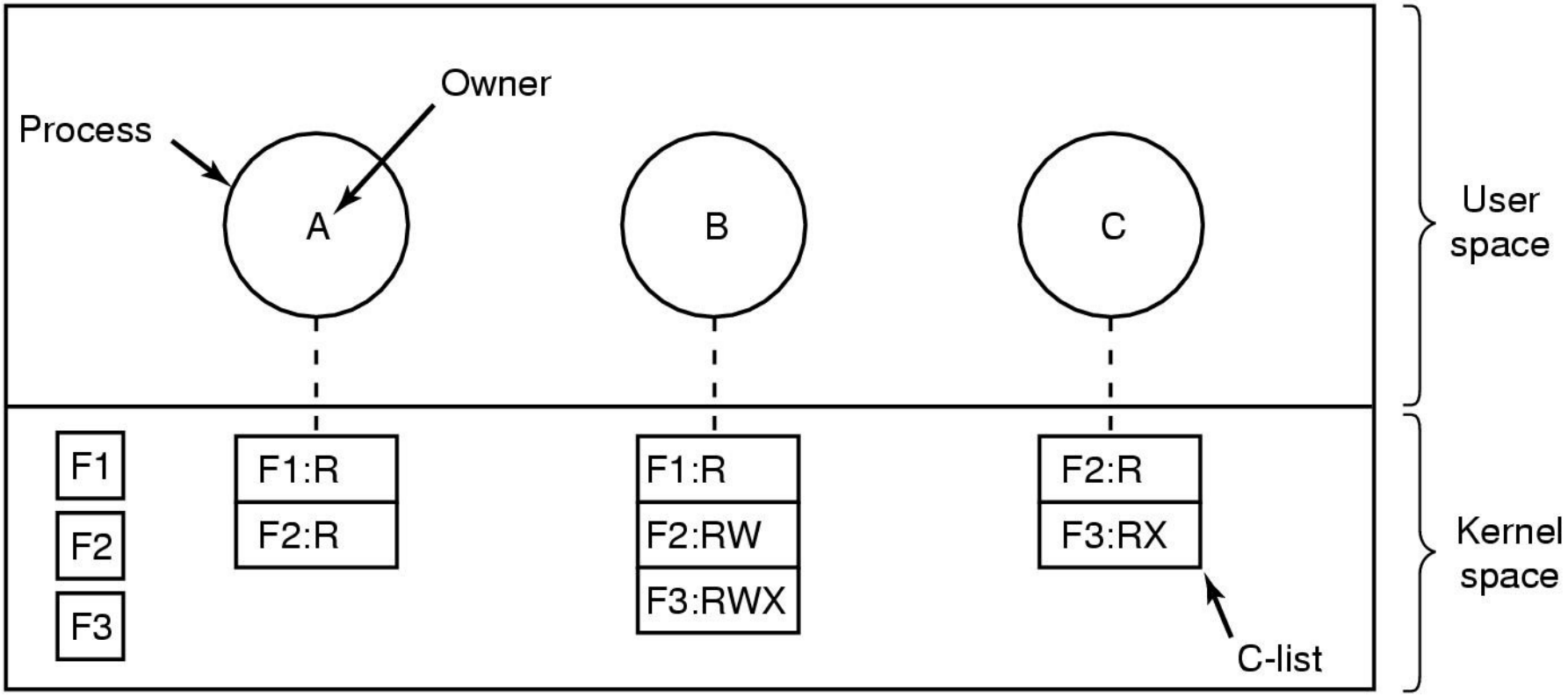


Use of access control lists to manage file access in UNIX

Capabilities

- Capability is an unforgeable ticket
  - Managed by OS
  - Can be passed from one process to another
- Permissive
- OS Mechanism (Reference monitor) checks ticket
  - Does not need to know the identity of the user/proc
- Can be used to partition superuser privilege
- Implementation: POSIX.1e capabilities

Capabilities



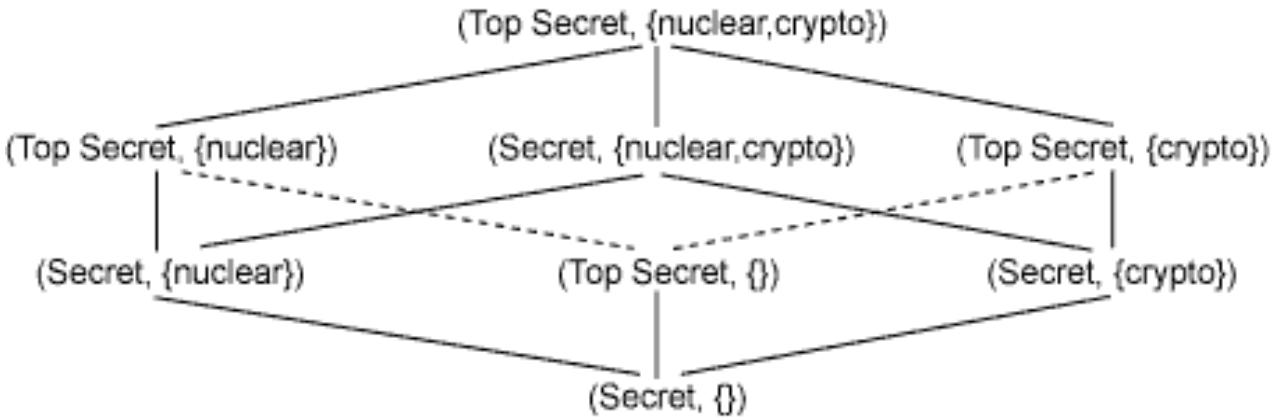
When capabilities are used, each process has a capability list.

Problems?

- What might go wrong with DAC or Capabilities?
  - Security is left to the discretion of subjects
  - Impossible to guarantee security of system
  - Security of system changes over time.
- Solution?
  - Mandatory Access Control: Operating system constrains the ability of subjects (even owners) to perform operations on objects according to a system-wide *security policy*.

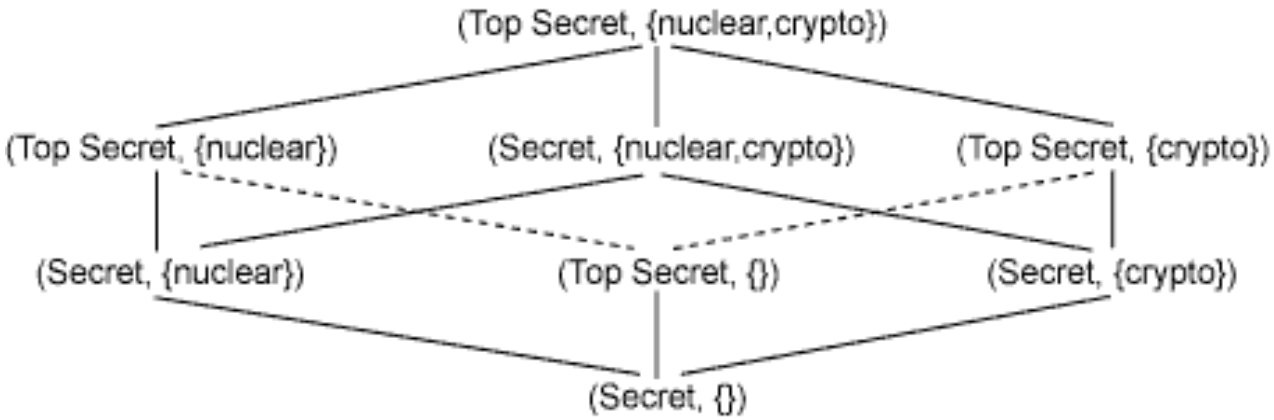
Bell-LaPadula Model

- A multi-level security model that provides strong confidentiality guarantees.
- Formalizes Classified Information
- State machine (Lattice) specifies permissible actions

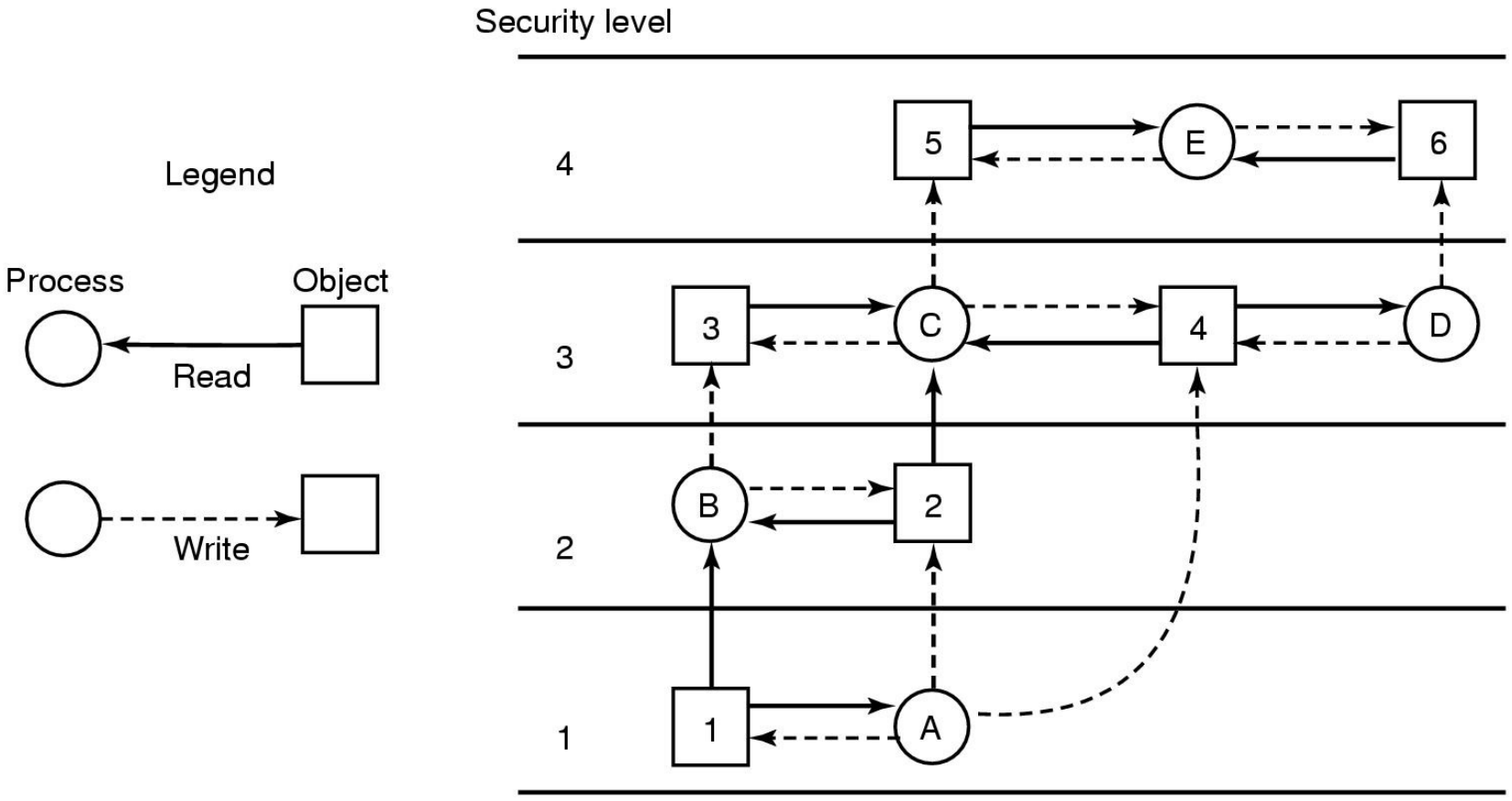


Bell-LaPadula Model

- **The Simple Security Property**: A subject running at security level k can read only objects at its level or lower. (*no read up*)
- **The * Property**: A subject at security level k can write only objects at its level or higher (*no write down)*



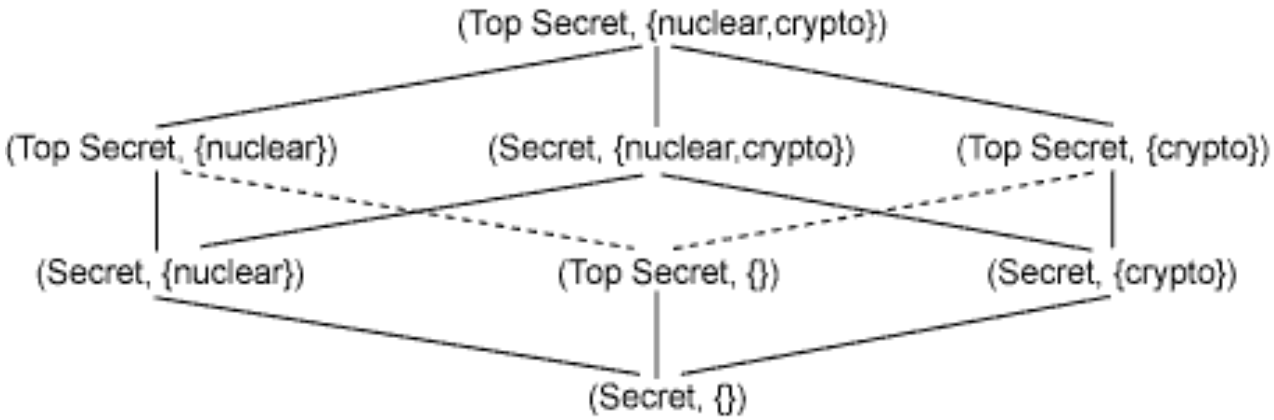
Information Flow



Using Bell-Lapadula, we can reason about permissible information flows in a system.

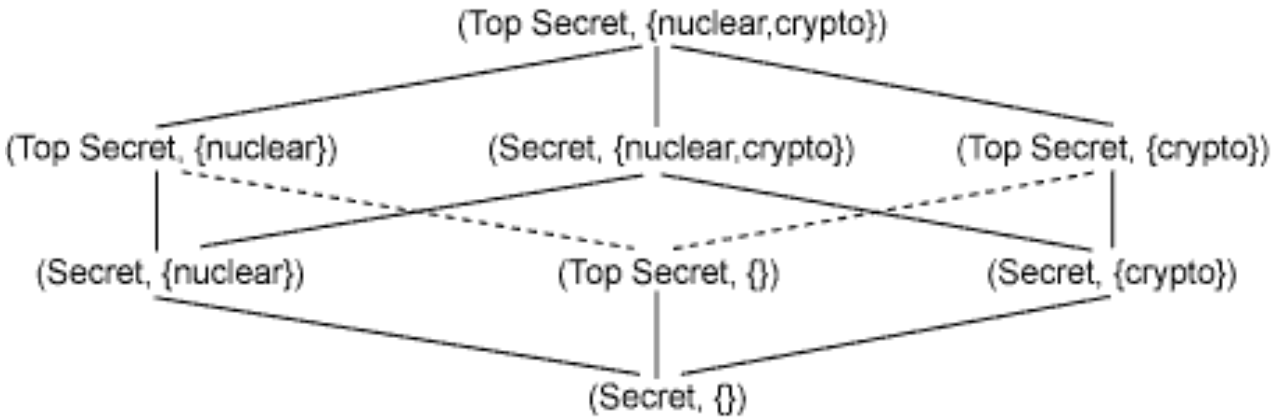
Biba Integrity Model

- Bell-LaPadula provides confidentiality. What about integrity?
- Biba model provides Integrity guarantees in a manner analogous to Bell-Lapadula's secrecy levels.
- Integrity prevents inappropriate modification of data.



Biba Integrity Model

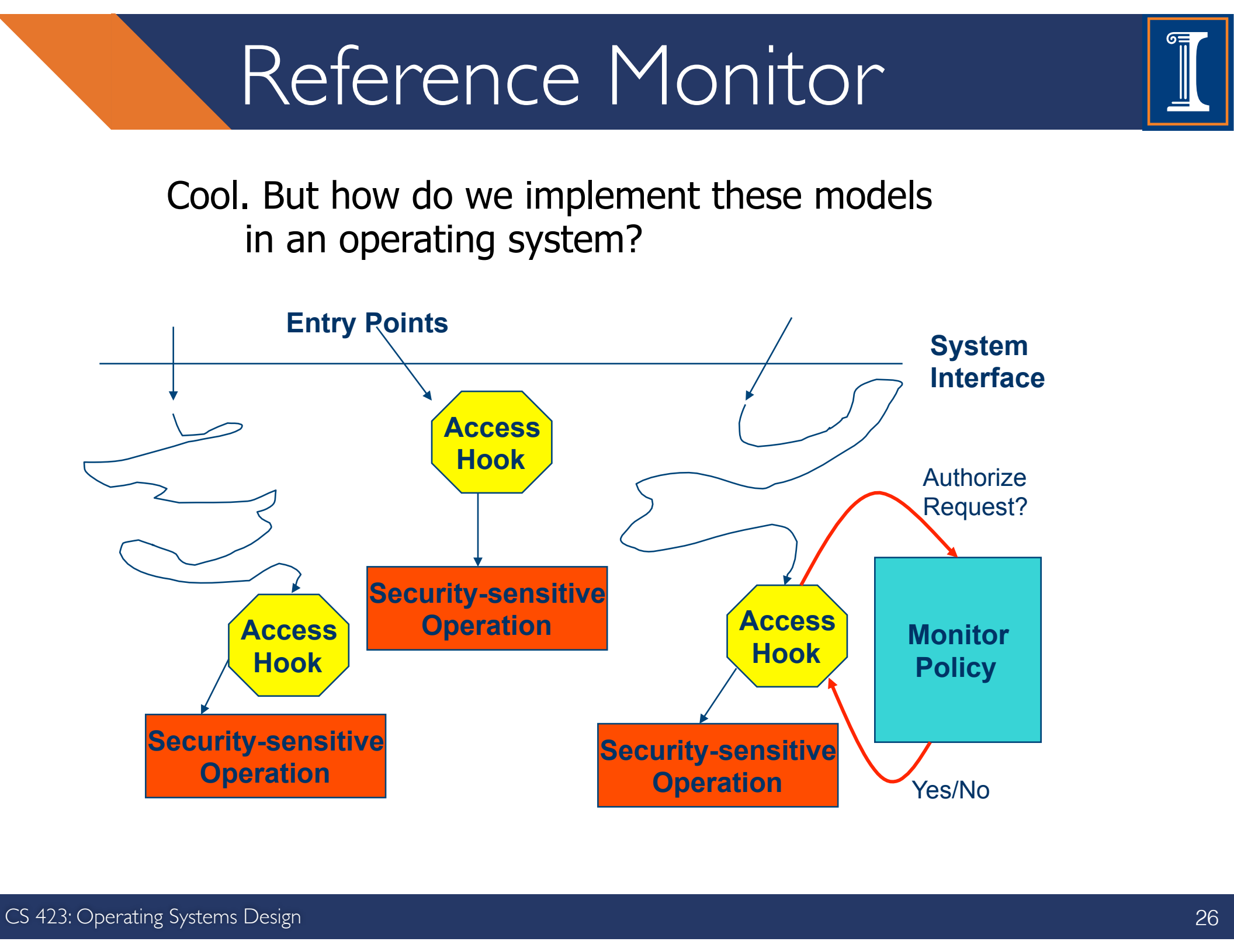
- **The Simple Integrity Property:** A subject running at integrity level k must not read an object at a lower integrity level (*no read down*)
- **The * Integrity Property:** A subject at security level k can only write objects at its level or lower. (*no write up)*



All the Access Controls

- **Basic Access Matrix**
  - **UNIX, ACL, various capability systems**
- **Aggregated Access Matrix**
  - **TE, RBAC, groups and attributes, parameterized**
- **Plus Domain Transitions**
  - **DTE, SELinux, Java**
- **Lattice Access Control Models**
  - **Bell-LaPadula, Biba, Denning**
- **Predicate Models**
  - **ASL, OASIS, domain-specific models, many others**
- **Safety Models**
  - **Take-grant, Schematic Protection Model, Typed Access Matrix**

Reference Monitor



Cool. But how do we implement these models in an operating system?

Reference Monitor

- Where to make access control decisions? (Mediation)
- Which access control decisions to make? (Authorization)
- Decision function: Compute decision based on request and the active security policy
- Reference Monitor Concept (i.e., Goals):
  - Complete Mediation
  - Tamper Proof
  - Verifiable

SELinux

- Designed by the NSA
- A more flexible solution than MLS
- SELinux Policies are comprised of 3 components:
  - Labeling State defines security contexts for every file (object) and user (subject).
  - Protection State defines the permitted <subject,object,operation> tuples.
  - Transition State permits ways for subjects and objects to move between security contexts.
- Enforcement mechanism designed to satisfy reference monitor concept

SELinux Labeling State

- Files and users on the system at boot-time must are associated with their security labels (contexts)
  - Map file paths to labels via regular expressions
  - Map users to labels by names by name
    - User labels pass on to their initial processes
- How are new files labeled? Processes?

SELinux Protection State

- MAC Policy based on *Type Enforcement*
- Access Matrix Policy
  - Processes with subject label…
  - Can access file of object label
  - If operations in matrix cell allow
- Focus: Least Privilege
  - Only provide permissions necessary

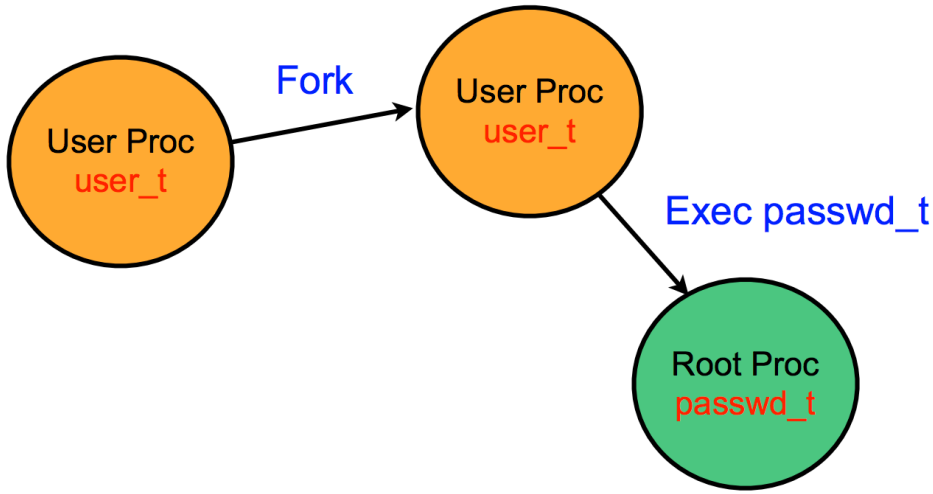
	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

SELinux Protection State

- Permissions in SELinux can be produced with runtime analysis.
- **Step 1**: Run programs in a controlled (no attacker) environment without any enforcement.
- **Step 2**: Audit all of the permissions used during normal operation.
- **Step 3**: Generate policy file description
  - Assign subject and object labels associated with program
  - Encode all permissions used into access rules

SELinux Transition State

- Premise: Processes don't need to run in the same protection state all of the time.
- Borrows concepts from *Role-Based Access Control*
- Example: `passwd` starts in user context, transitions to privileged context to update /etc/passwd, transitions back to user.



@ 2001 Linux Kernel Summit…

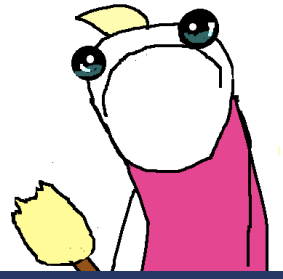
@ 2001 Linux Kernel Summit…

Include SELinux in Linux 2.5!

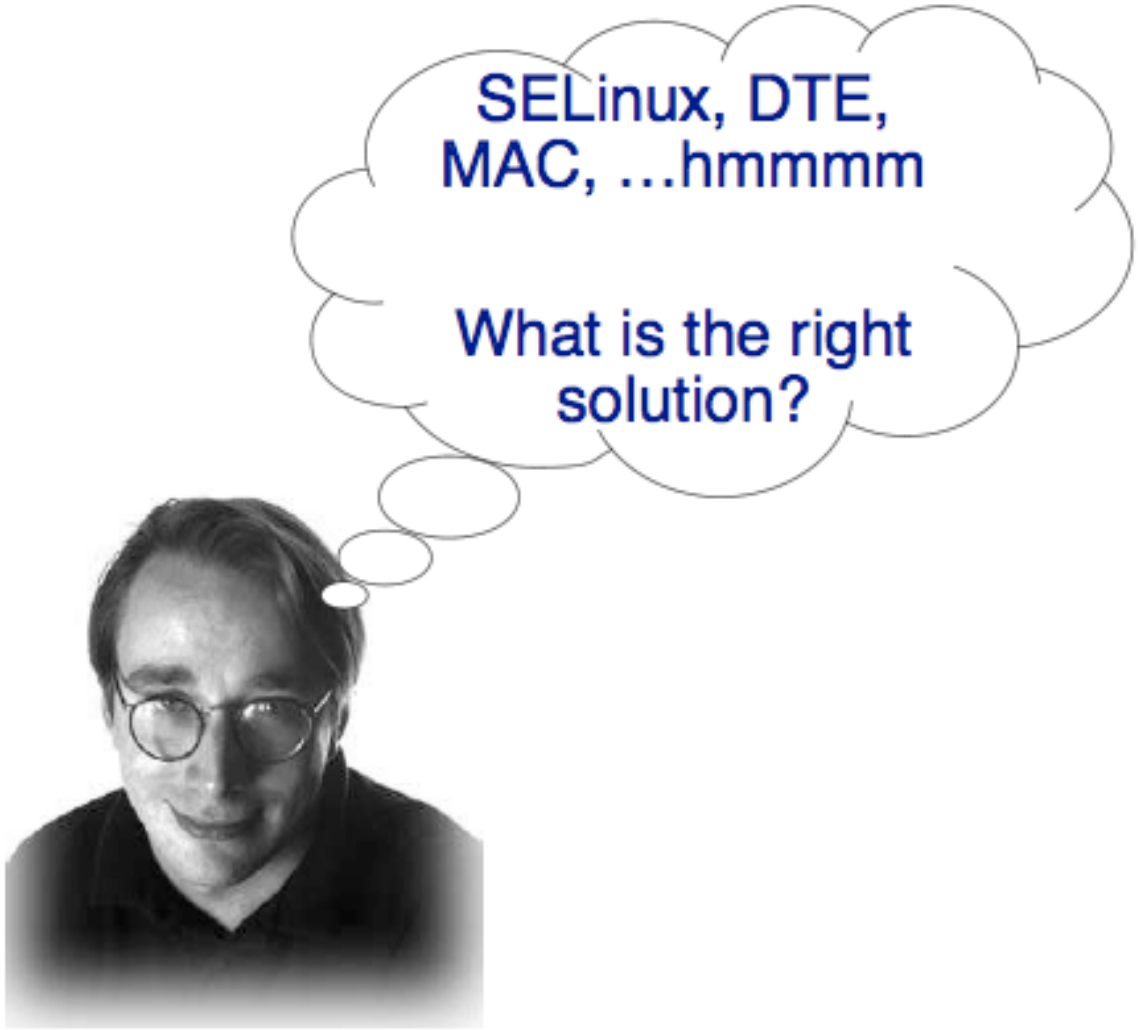
I'm just not that into you…

Linux Security circa 2000

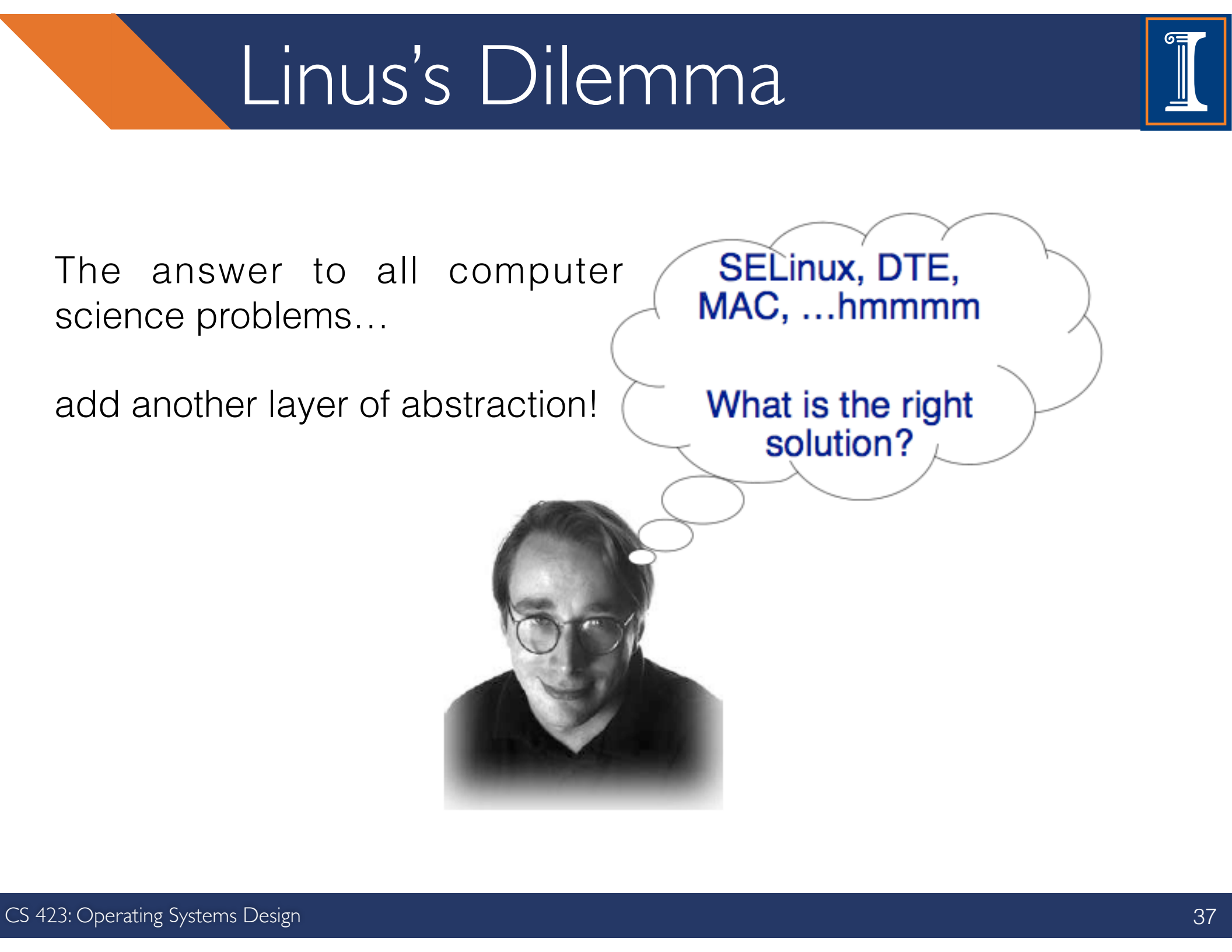
- Patches to the Linux kernel
  - Enforce different access control policy
    - Restrict root processes
  - Some hardening
- Argus PitBull
  - Limited permissions for root services
- RSBAC
  - MAC enforcement and virus scanning
- grsecurity
  - RBAC MAC system
  - Auditing, buffer overflow prevention, /tmp race protection, etc
- LIDS
  - MAC system for root confinement



Linus's Dilemma



Linus's Dilemma



The answer to all computer science problems…

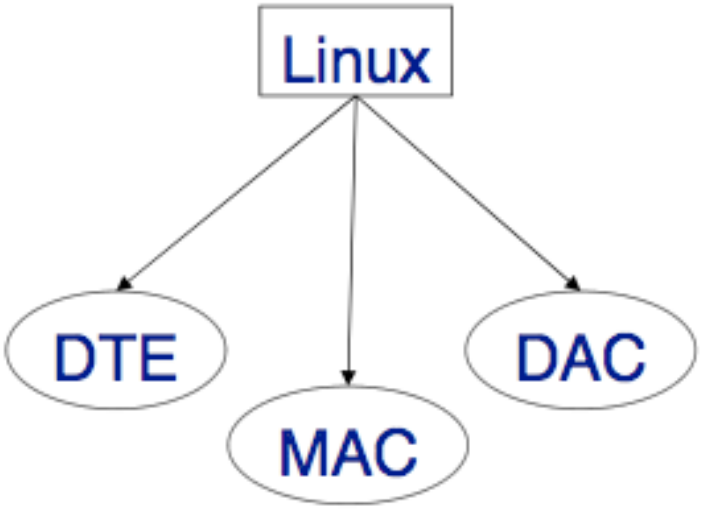
add another layer of abstraction!

Linux Security Models

- "to allow Linux to support a variety of security models, so that security developers don't have to have the 'my dog's bigger than your dog' argument, and users can choose the security model that suits their needs.", Crispin Cowan
  - http://mail.wirex.com/pipermail/linux-security-module/2001-April:/0005.html

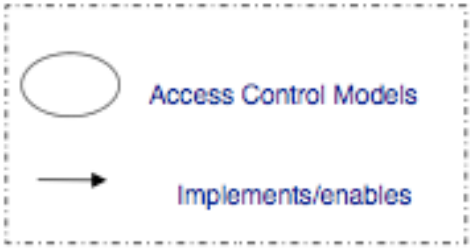
Linux Security Modules

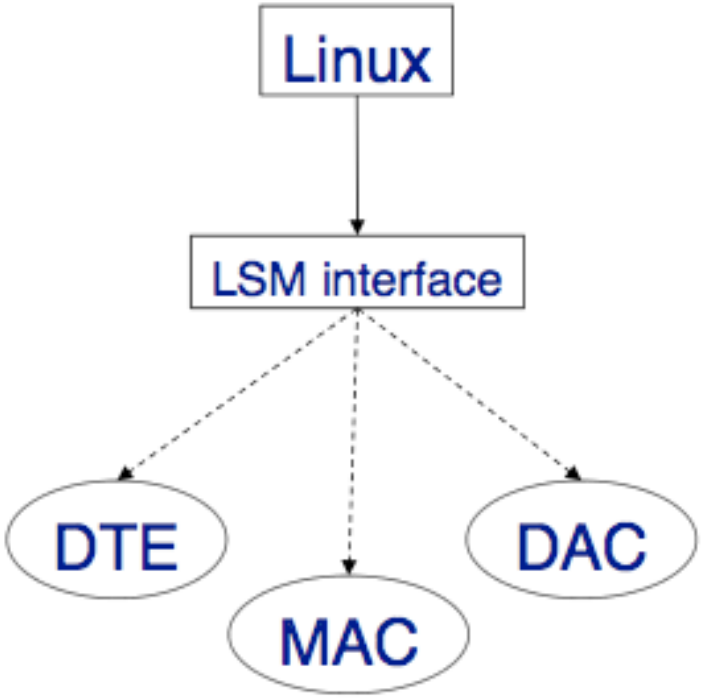
Before LSM



Access control models implemented as Kernel patches

After LSM





Access control models implemented as Loadable Kernel Modules

LSM Requirements

- LSM needs to reach a balance between kernel developer and security developers requirements. LSM needs to unify the functional needs of as many security projects as possible, while minimizing the impact on the Linux kernel.
  - Truly generic
  - conceptually simple
  - minimally invasive
  - Efficient
  - Support for POSIX capabilities
  - Support the implementation of as many access control models as Loadable Kernel Modules

LSM Architecture

- Linux Kernel modified in 5 ways:
  - Opaque security fields added to certain kernel data structures
  - Security hook function calls inserted at various points with the kernel code
  - A generic security system call added
  - Function to allow modules to register and unregistered as security modules
  - Move capabilities logic into an optional security module

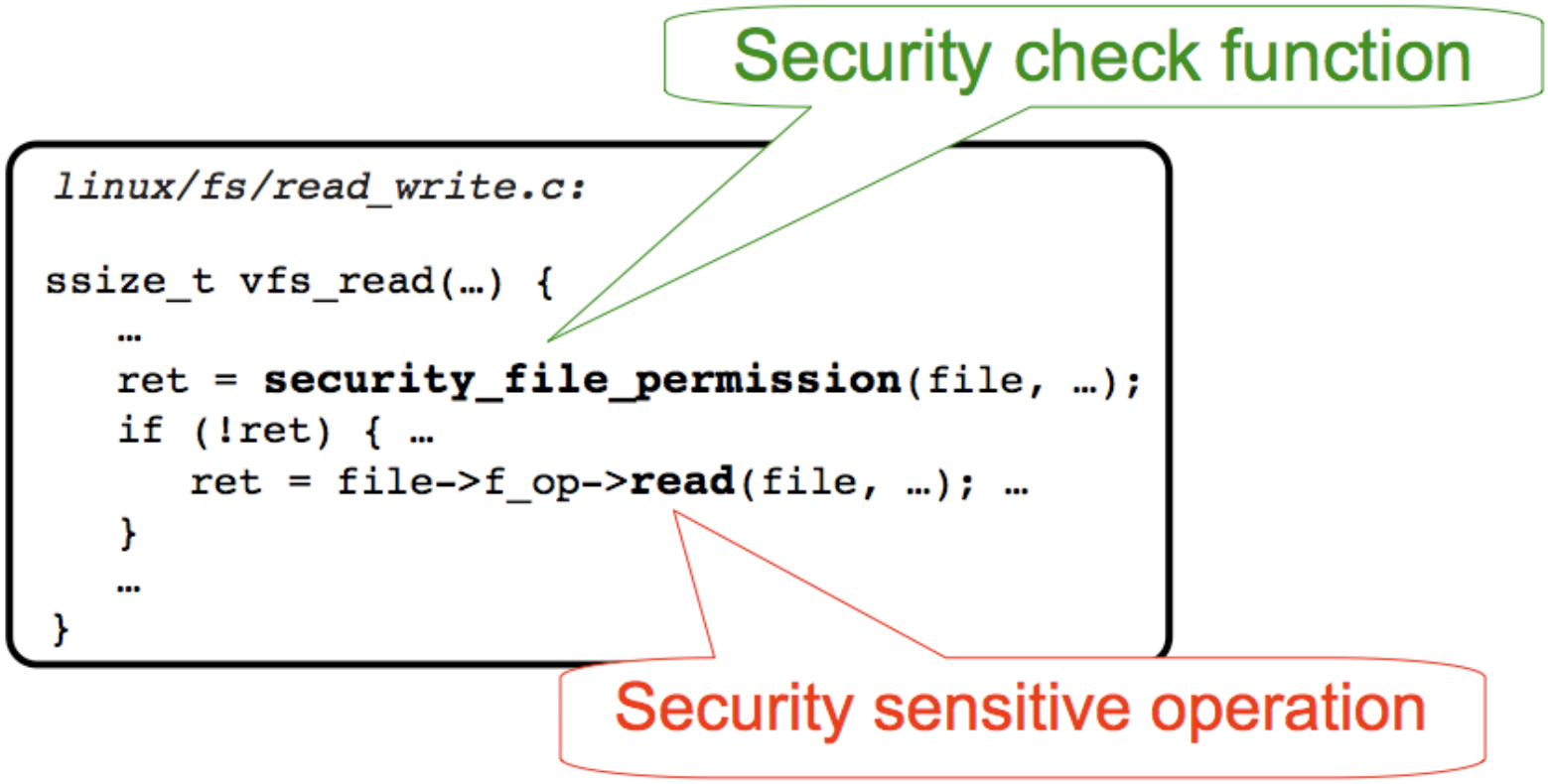
Opaque Security Fiels

- Enable security modules to associate security information to Kernel objects
- Implemented as void* pointers
- Completely managed by security modules
- What to do about object created before the security module is loaded?

Security Hooks

- Function calls that can be overridden by security modules to manage security fields and mediate access to Kernel objects.
- Hooks called via function pointers stored in `security->ops` table
- Hooks are primarily "restrictive"

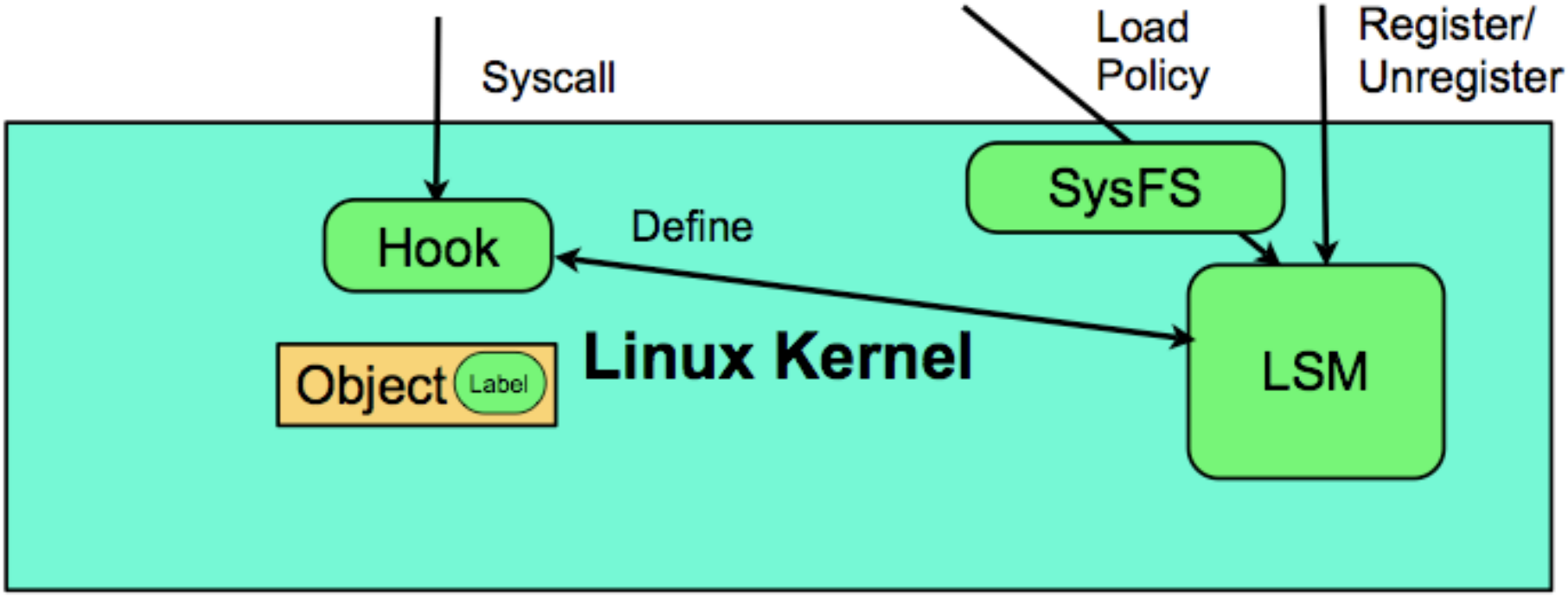
Security Hooks



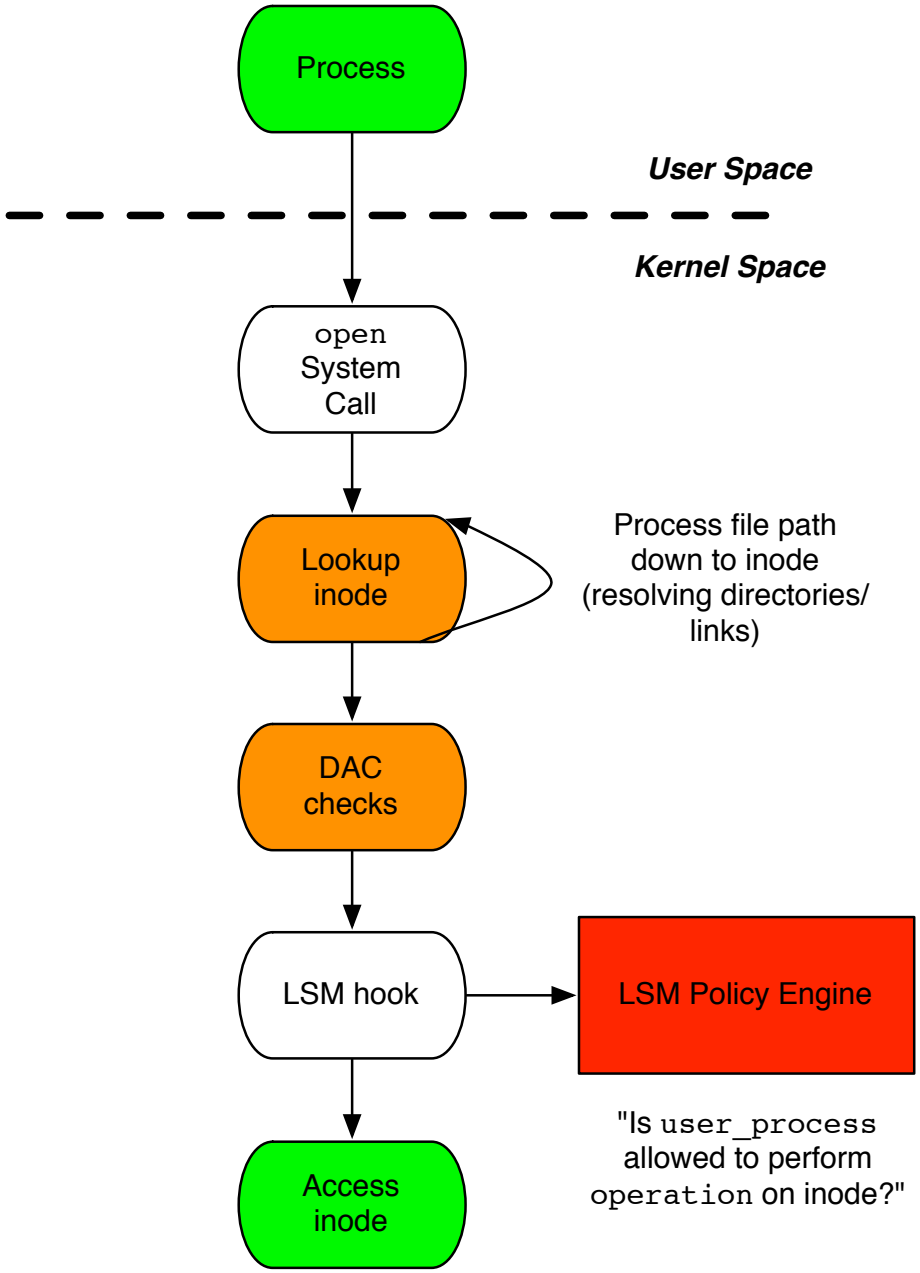
Security Hook Details

- Difference from discretionary controls
  - More object types
    - 29 different object types
    - Per packet, superblock, shared memory, processes
    - Different types of files
  - Finer-grained operations
    - File: ioctl, create, getattr, setattr, lock, append, unlink,
  - System labeling
    - Not dependent on user
  - Authorization and policy defined by module
    - Not defined by the kernel

LSM Hook Architecture



LSM Hook Architecture



Conclusions

- Access Control is supported in operating systems through the "Reference Monitor" concept
- LSM is a framework for designing reference monitors
- Today, many security modules exist
  - Must define authorization hooks to mediate access
  - Must expose a policy framework for specifying which accesses to authorize
- Policy Challenges
  - Is language expressive enough to capture the goals of the user?
  - Is language simple/intuitive enough for ease of use?
    - Policy misconfiguration breaks security!!