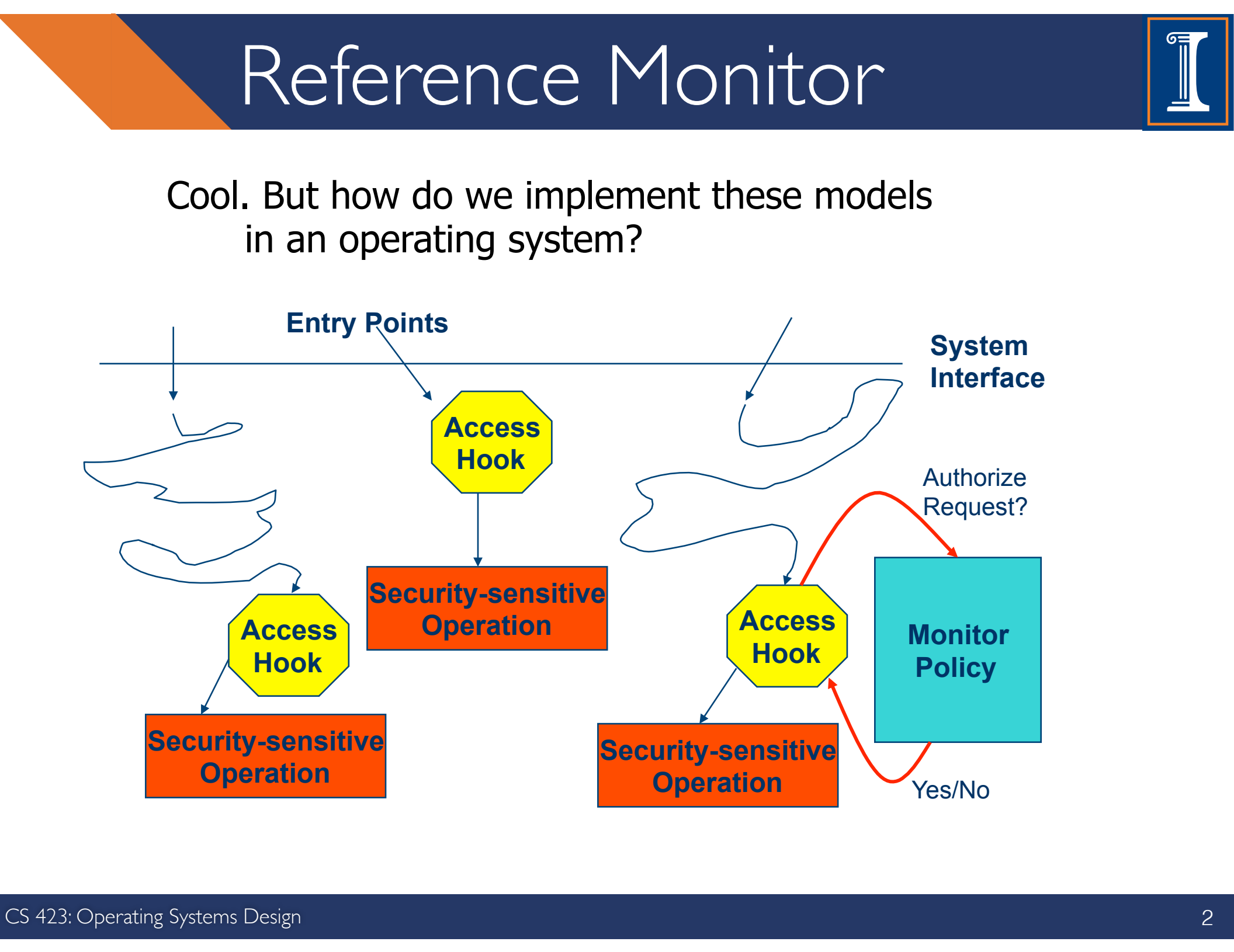
Goals for Today

- 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 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

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

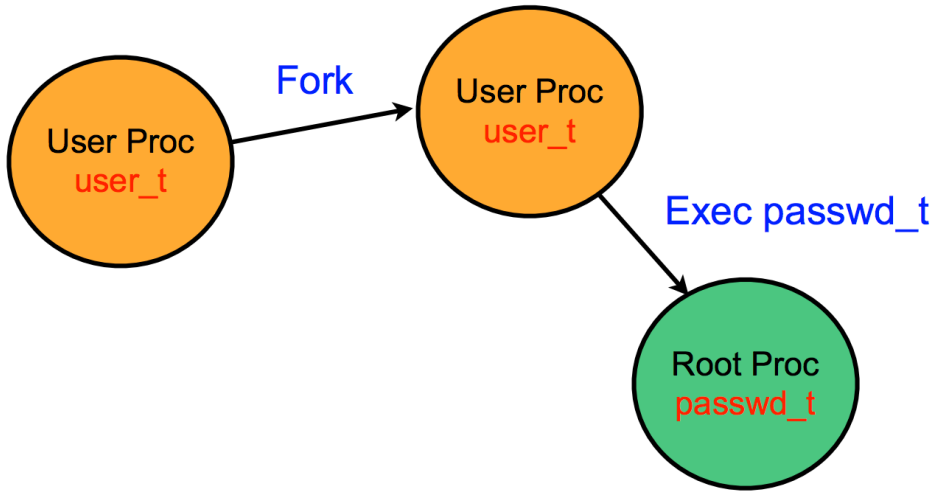
	O_1	O_2	O_3
S_1	Y	Y	N
S_2	N	Y	N
S_3	N	Y	Y

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…

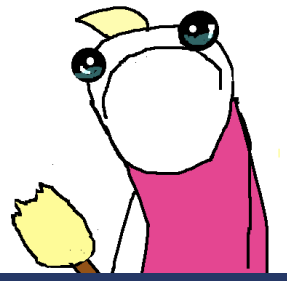
@ 2001 Linux Kernel Summit…

Include SELinux in Linux 2.5!

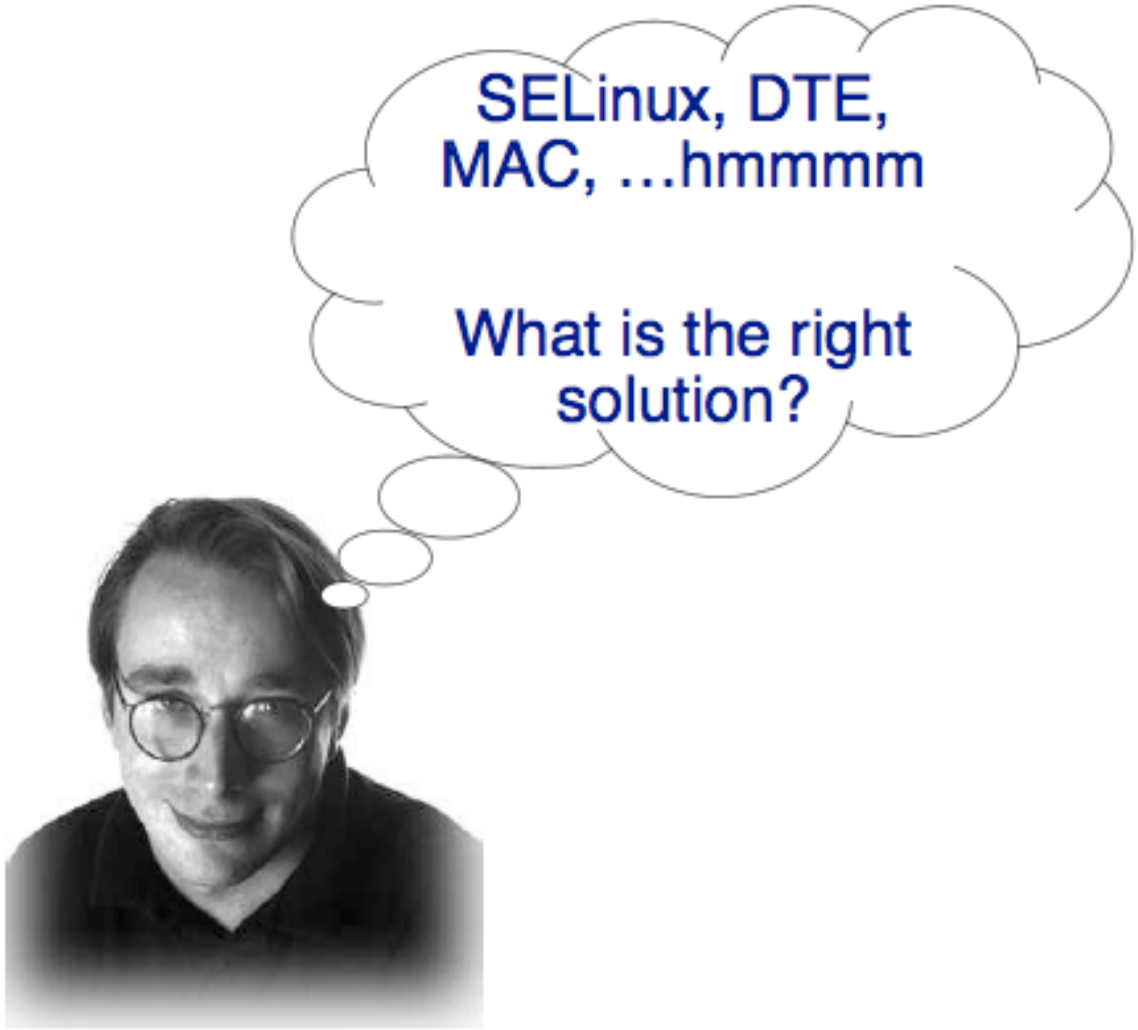
I'm just not that into you…

Linux Security circa 2000

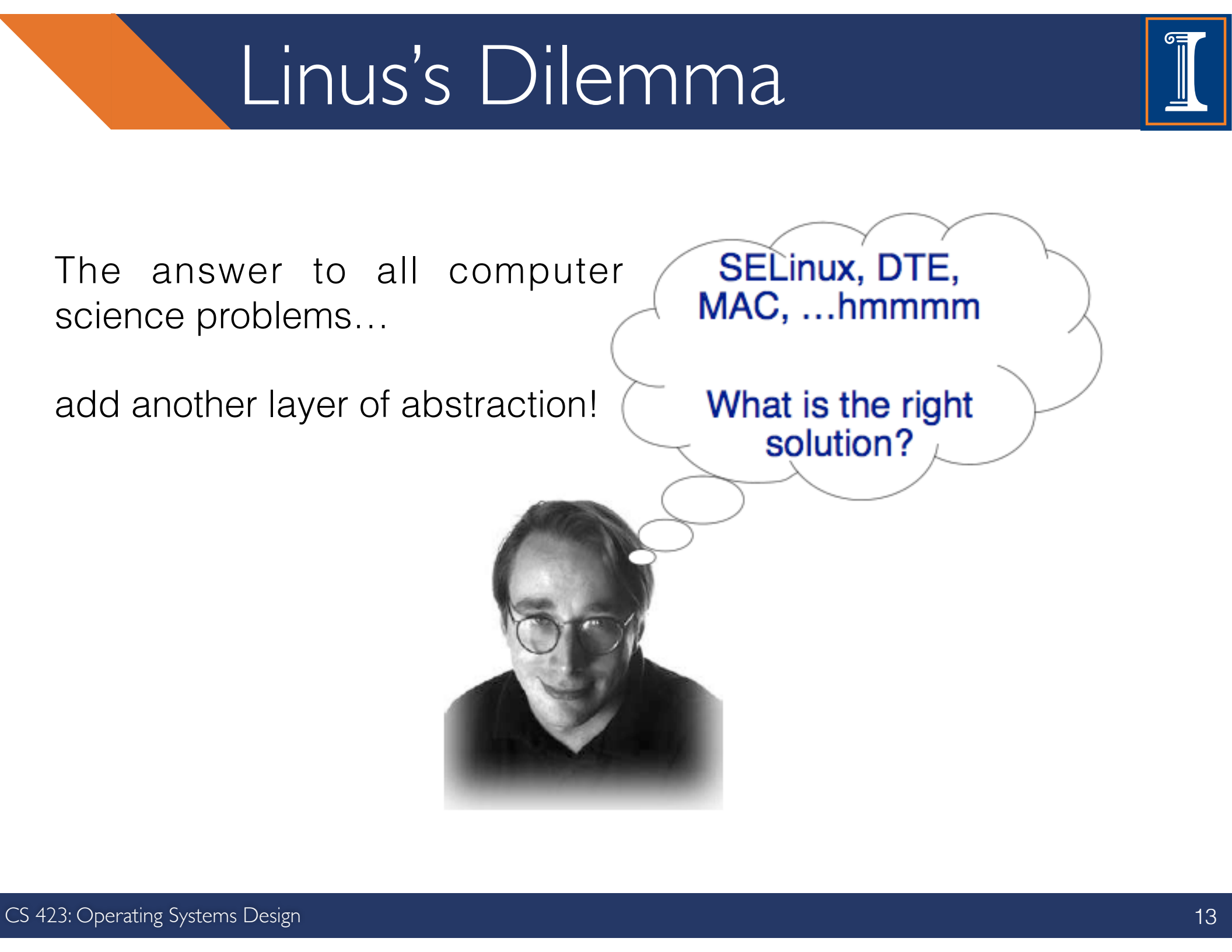
- Patches to the Linux kernel
  - Enforce different access control policy
    - Restrict root processes
  - Some hardening
- Argus PitBull
  - Limited permissions for root services
- RSBAC
  - MAC enforcement and virus scanning
- grsecurity
  - RBAC MAC system
  - Auditing, buffer overflow prevention,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



Linus's Dilemma



Linus's Dilemma



The answer to all computer science problems…

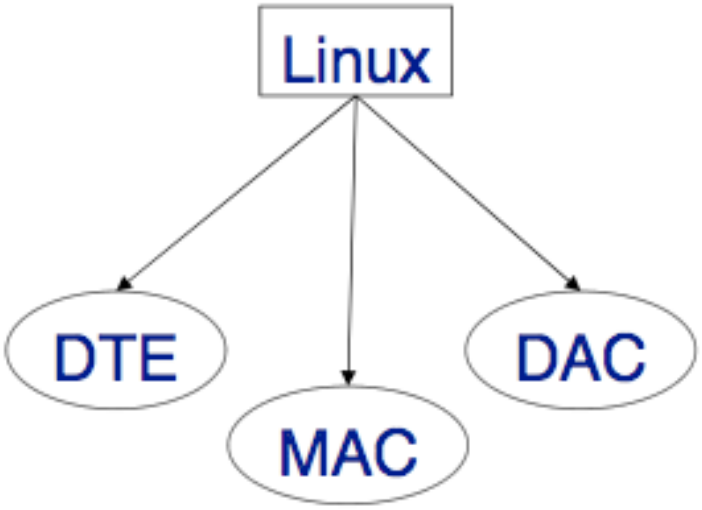
add another layer of abstraction!

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

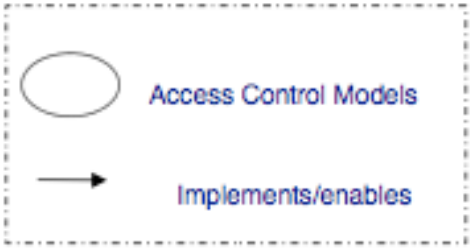
Linux Security Modules

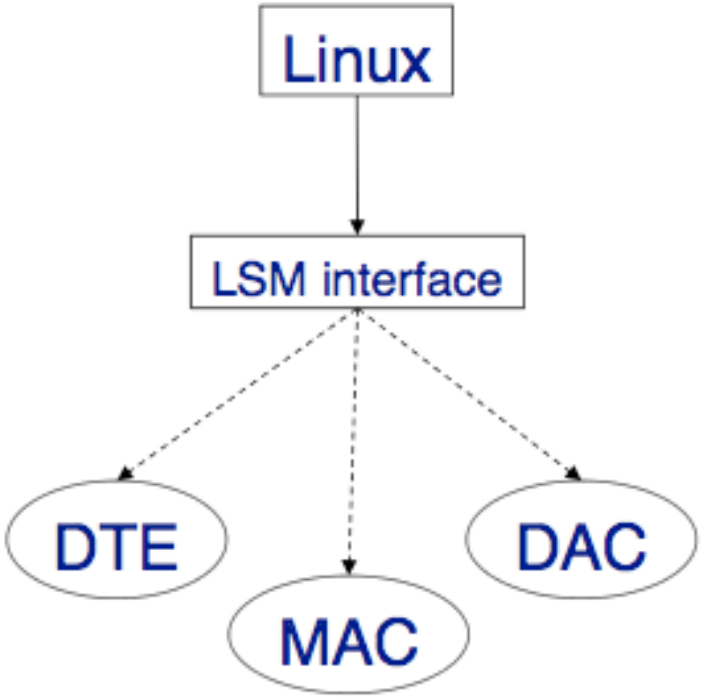
Before LSM



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

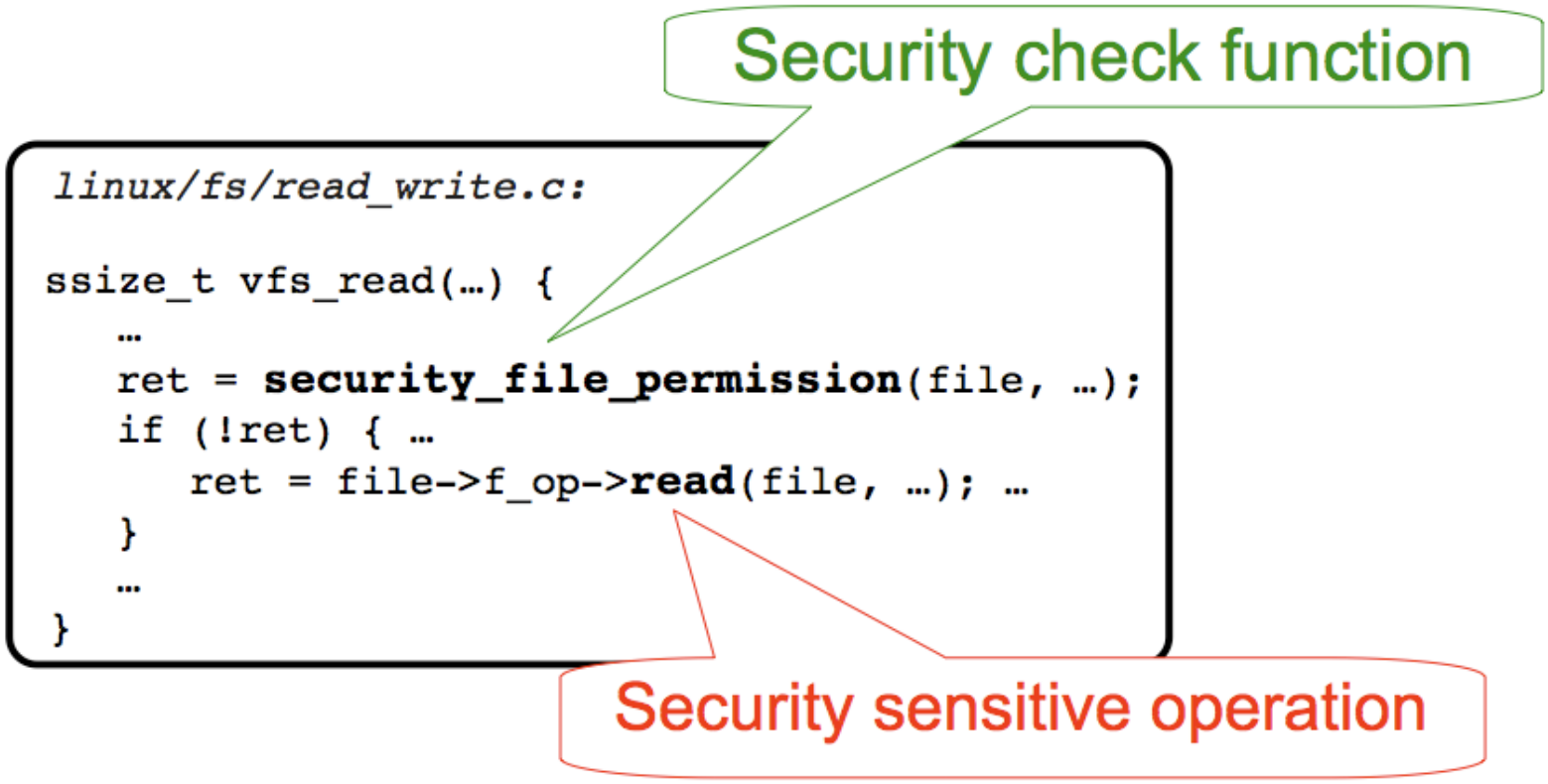
Opaque Security Fiels

- 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"

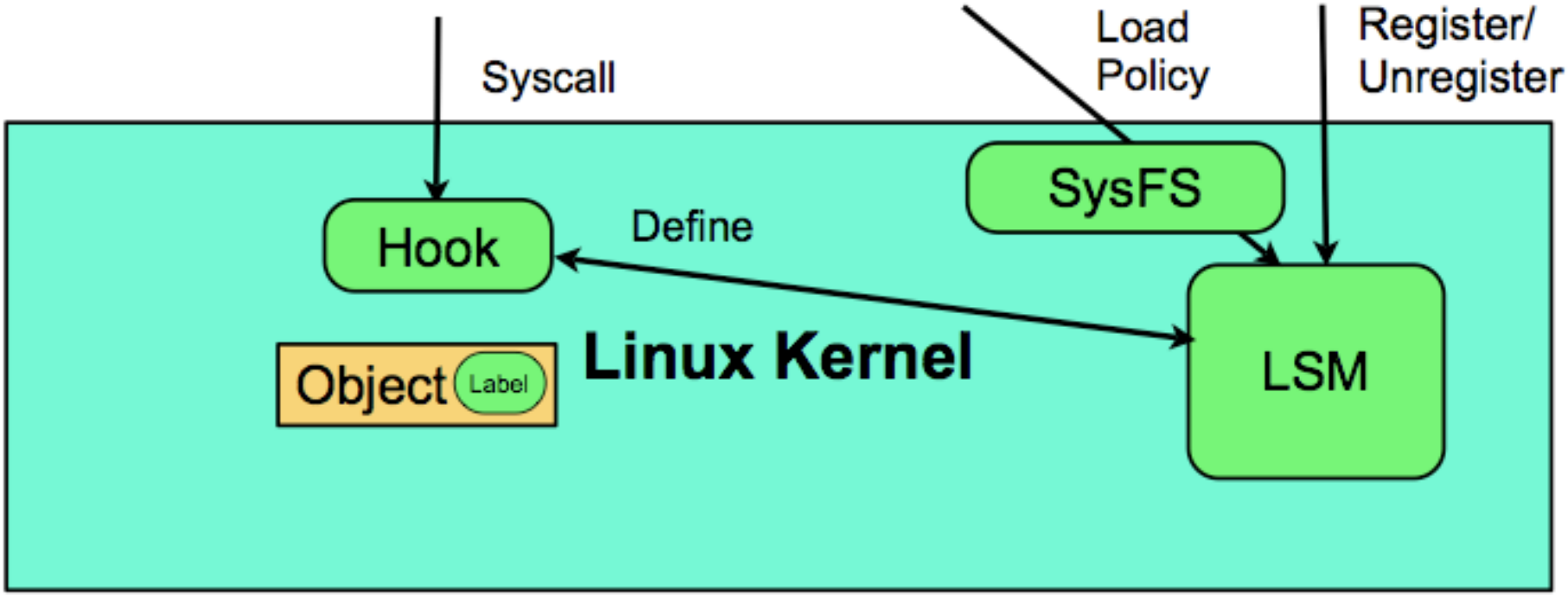
Security Hooks



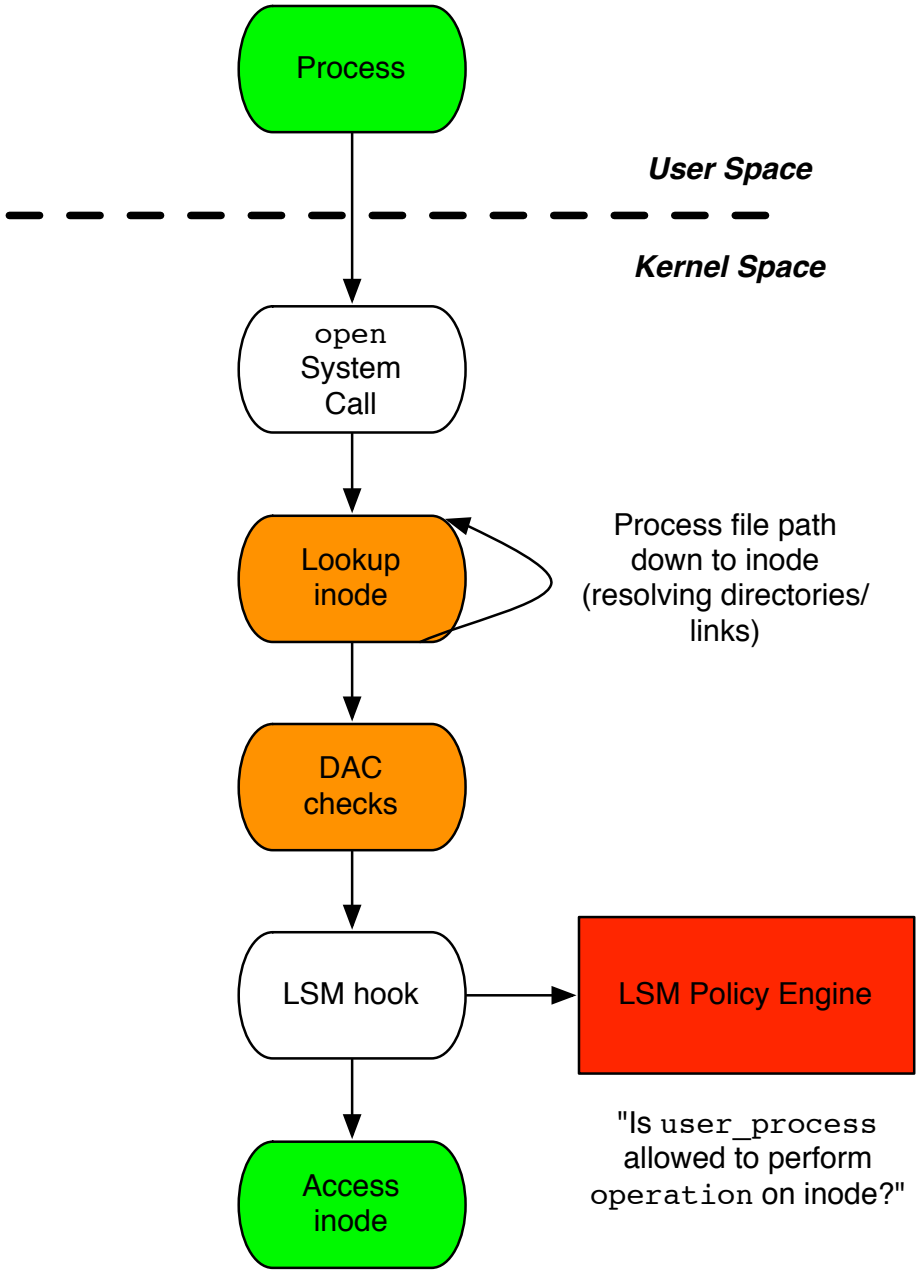
Security Hook Details

- 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!!

LSM Performance

- Microbenchmark: LMBench
  - Compare standard Linux Kernel 2.5.15 with Linux Kernel with LSM patch and a default capabilities module
  - Worst case overhead is 5.1%
- Macrobenchmark: Kernel Compilation
  - Worst case 0.3%
- Macrobenchmark: Webstone
  - With Netfilter hooks 5-7%
  - Uni-Processor 16%
  - SMP 21% overhead

LSM Use

- Available in Linux 2.6
  - Packet-level controls upstreamed in 2.6.16
- Modules
  - POSIX Capabilities module
  - SELinux module
  - Domain and Type Enforcement
  - Openwall, includes grsecurity function
  - LIDS
  - AppArmor
- Not everyone is in favor
  - How does LSM impact system hardening?