Administrivia

- **Encryption utility due next Tuesday (Feb 22)**
- **Midterm following Tuesday (March 1)**
  - Open book, open note
  - Bring copies of all the papers, you will need them
  - You can bring print-outs of the lecture notes
  - Don't count on reading the papers during the exam
  - Covers first six lectures
- **Sections Wednesday 7pm, 719 Bway #709**
  - Attendance highly recommended
  - Won't introduce new areas, but covers existing topics in more detail

A security problem [Hardy]

- **Setting: A multi-user time sharing system**
- **Wanted fortran compiler to keep statistics**
  - Modified compiler `/sysx/fort` to record stats in `/sysx/stat`
  - Gave compiler "home files license"—allows writing to anything in `/sysx` (kind of like Unix setuid)
- **What's wrong here?**

A confused deputy

- **Attacker could overwrite any files in `/sysx`**
  - System billing records kept in `/sysx/bill` got wiped
  - Probably command like `fort -o /sysx/bill file.f`
- **Is this a compiler bug?**
  - Original implementors did not anticipate extra rights
  - Can't blame them for unchecked output file
- **Compiler is a "confused deputy"**
  - Inherits privileges from invoking user (e.g., read `file.f`)
  - Also inherits from home files license
  - Which master is it serving on any given system call?
  - OS doesn't know if it just sees `open ("/sysx/bill", ...)`

Capabilities

- **Recall access matrix from Bell-Lapadula model**
  - Abstraction models arbitrary subject→object access rights
- **Often implemented with access control lists (ACLs)**
  - For each object, store permissible subjects & rights
- **Slicing matrix other way yields capabilities**
  - E.g., For each process, store a list of objects it can access
  - Process explicitly invokes particular capabilities
- **Three general approaches to capabilities:**
  - Hardware enforced (Tagged architectures like M-machine)
  - Kernel-enforced (Hydra, KeyKOS)
  - Self-authenticating capabilities (like Amoeba)

Hydra

- **Machine & programing env. built at CMU in ’70s**
- **OS enforced object modularity with capabilities**
  - Could only call object methods with a capability
- **Agumentation let methods manipulate objects**
  - A method executes with the capability list of the object, not the caller
- **Template methods take capabilities from caller**
  - So method can access objects specified by caller

KeyKOS

- **Capability system developed in the early 1980s**
- **Goal: Extreme security, reliability, and availability**
- **Structured as a "nanokernel"**
  - Kernel proper only 20,000 likes of C, 100KB footprint
  - Avoids many problems with traditional kernels
  - Traditional OS interfaces implemented outside the kernel (including binary compatibility with existing OSes)
- **Basic idea: No privileges other than capabilities**
  - Partition system into many processes akin to objects
  - Capabilities like pointers to objects in OO languages

Unique features of KeyKOS

- **Single-level store**
  - Everything is persistent: memory, processes, ...
  - System periodically checkpoints its entire state
  - After power outage, everything comes back up as it was (may just lose the last few characters you typed)
- **"Stateless" kernel design only caches information**
  - All kernel state reconstructible from persistent data
- **Simplifies kernel and makes it more robust**
  - Kernel never runs out of space in memory allocation
  - No message queues, etc. in kernel
  - Run out of memory? Just checkpoint system

KeyKOS capabilities

- **Refered to as "keys" for short**
- **Types of keys:**
  - *devices* – Low-level hardware access
  - *pages* – Persistent page of memory (can be mapped)
  - *nodes* – Container for 16 capabilities
  - *segments* – Pages & segments glued together with nodes
  - *meters* – right to consume CPU time
  - *domains* – a thread context
- **Anyone possessing a key can grant it to others**
  - But creating a key is a privileged operation
  - E.g., requires "prime meter" to divide it into submeters

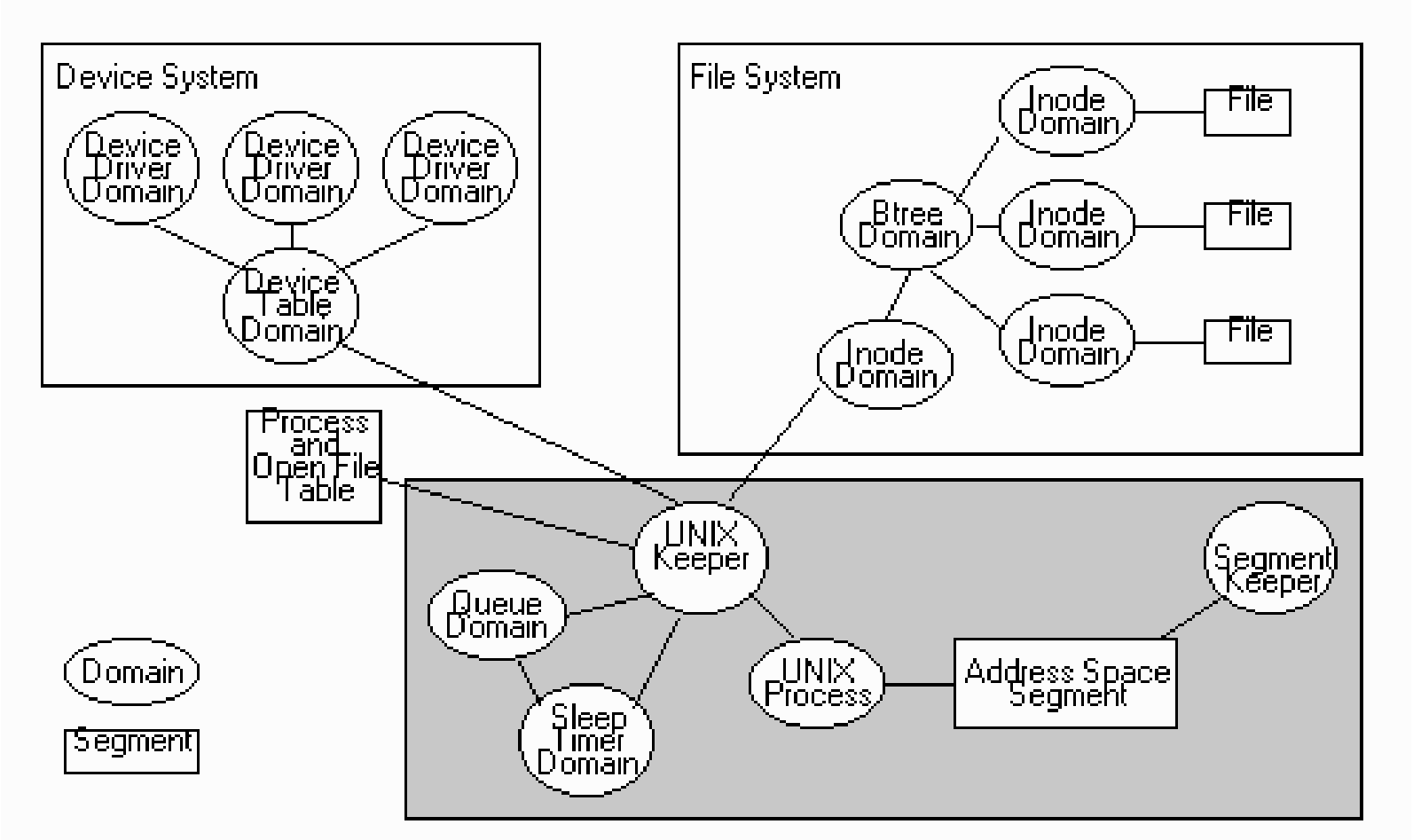
Capability details

- **Each domain has a number of key "slots":**
  - 16 general-purpose key slots
  - *address slot* – contains segment with process VM
  - *meter slot* – contains key for CPU time
  - *keeper slot* – contains key exceptions
- **Segments also have an associated keeper**
  - Process that gets invoked on invalid reference
- **Meter keeper (allows creative scheduling policies)**
- **Calls generate return key for calling domain**
  - (Not required–other forms of message don't do this)

KeyNIX: Unix implementation for KeyKOS

- **"One kernel per process" architecture**
  - Hard to crash kernel
  - Even harder to crash system
- **Proc's kernel is it's keeper**
  - Unmodified Unix binary makes Unix syscall
  - Invalid KeyKOS syscall, transfers control to Unix keeper
- **Of course, kernels need to share state**
  - Use shared segment for process and file tables

KeyNIX overview



Keynix I/O

- **Every file is a different process**
  - Elegant, and fault isolated
  - Small files can live in a node, not a segment
  - Makes the `namei()` function very expensive
- **Pipes require queues**
  - This turned out to be complicated and inefficient
  - Interaction with signals complicated
- **Other OS features perform very well, though**
  - E.g., fork is six times faster than Mach 2.5

Self-authenticating capabilities

- **Every access must be accompanied by a capability**
  - For each object, OS stores random *check* value
  - Capability is: $\{\text{Object}, \text{Rights}, \text{MAC}(\mathit{check}, \text{Rights})\}$
- **OS gives processes capabilities**
  - Process creating resource gets full access rights
  - Can ask OS to generate capability with restricted rights
- **Makes sharing very easy in distributed systems**
- **To revoke rights, must change *check* value**
  - Need some way for everyone else to reacquire capabilities
- **Hard to control propagation**

Limitations of capabilities

- **IPC performance a losing battle with CPU makers**
  - CPUs optimized for "common" code, not context switches
  - Capability systems usually involve many IPCs
- **Capability programming model never took off**
  - Requires changes throughout application software
  - Call capabilities "file descriptors" or "Java pointers" and people will use them
  - But discipline of pure capability system challenging so far
- **Next topic: OS security schemes more compatible with existing applications**
- **After break: Security in distributed systems**

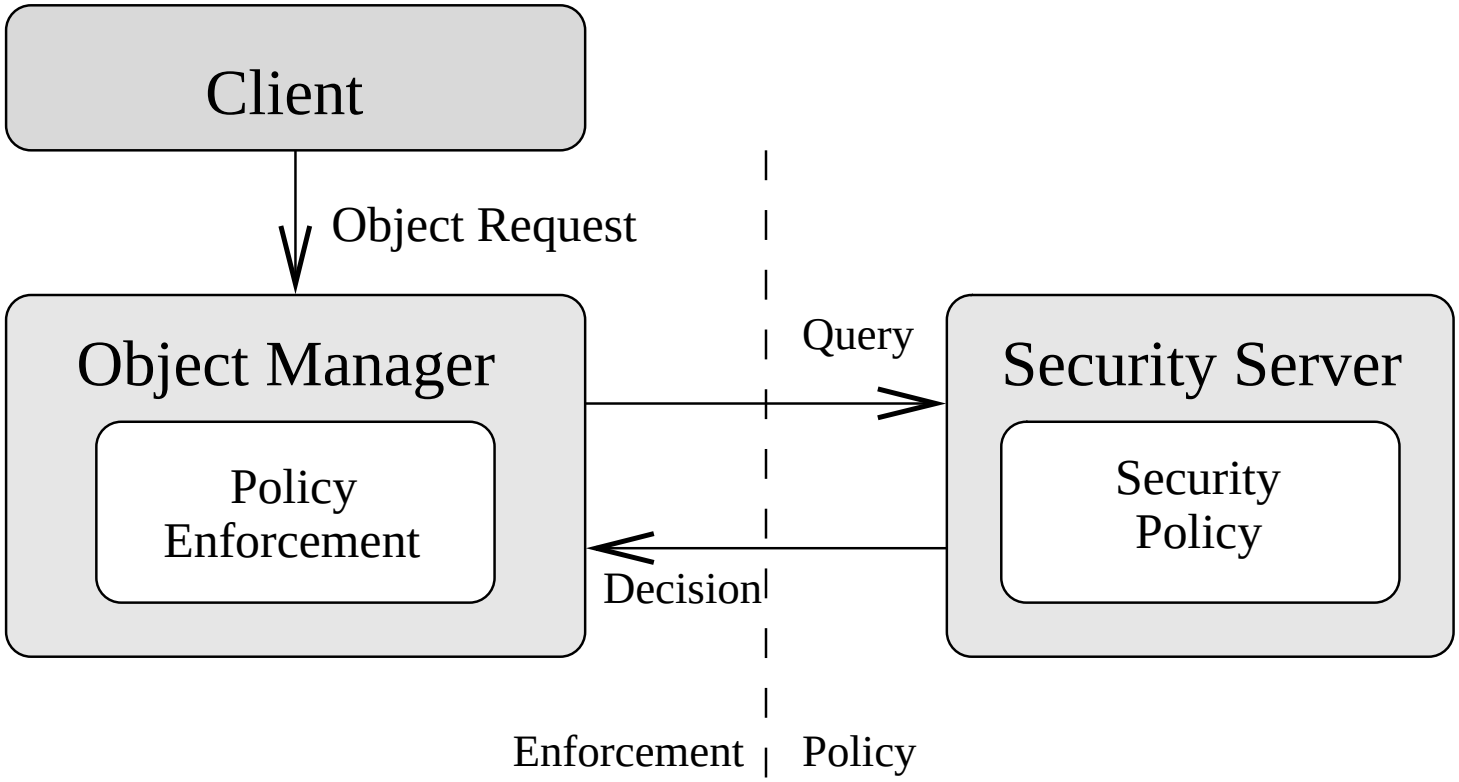
Janus—security through interposition

- **Principle: "An application can do little harm if its access to the underlying operating system is appropriately restricted."**
- **Approach: Use OS debugging facilities to intercept all system calls**
  - Policy config file restricts syscalls allowed to application: `path allow read,write /tmp/*`
- **Limitations**
  - Abstraction level inappropriate (see path, want i-node)
  - Application interface limited (can't IPC as a role)
  - Hard to reflect policy changes (revoke mmapped file)

The flask security architecture

- **Problem: Military needs adequate secure systems**
  - Not enough civilian demand for military security policies
- **Solution: Separate policy from enforcement mechanism**
  - Most people will plug in simple DAC policies
  - Military can take system off-the-shelf, plug in new policy
- **Requires putting adequate hooks in the system**
  - Each object has manager that guards access to the object
  - Conceptually, manager consults security server on each access
- **Flask security architecture prototyped in fluke**
  - Now part of SElinux, which NSA hopes to see accepted

Architecture



- **Separating enforcement from policy**

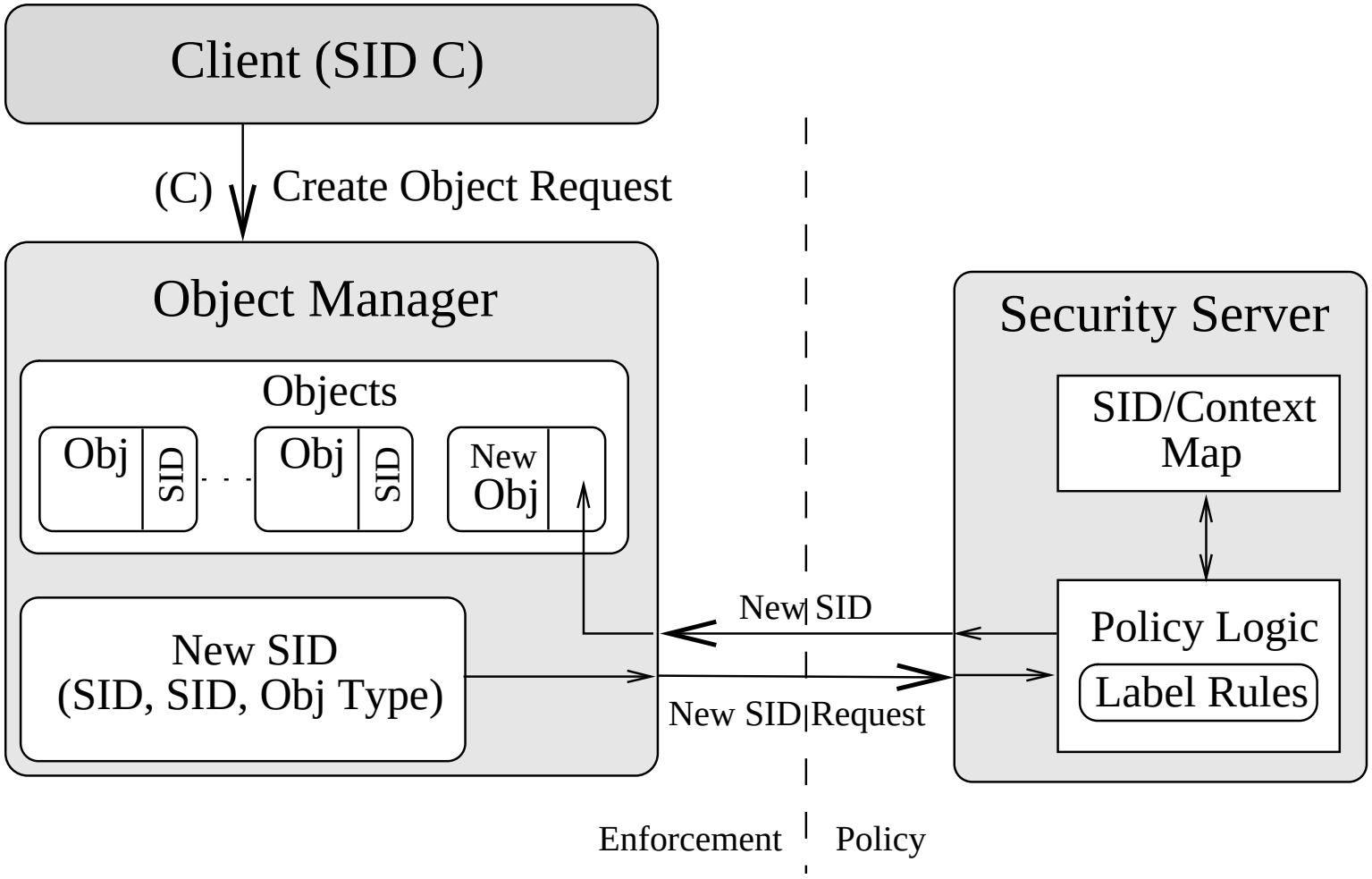
Challenges

- **Performance**
  - Adding hooks on every operation
  - People who don't need security don't want slowdown
- **Using generic enough data structures**
  - Object managers independent of policy still need to associate data structures (e.g., labels) with objects
- **Revocation**
  - May interact in a complicated way with any access caching
  - Once revocation completes, new policy must be in effect
  - Bad guy cannot be allowed to delay revocation completion indefinitely

Basic flask concepts

- **All objects are labeled with a *security context***
  - Security context is an arbitrary string—opaque to obj mgr
  - Example: $\{\text{invoice}\ [(\text{Andy}, \text{Authorize})]\}$
- **Labels abbreviated with security IDs (SIDs)**
  - 32-bit integer, interpretable only by security server
  - Not valid across reboots (can't store in file system)
  - Fixed size makes it easier for obj mgr to handle
- **Queries to server done in terms of SIDs**
  - Create (client SID, old obj SID, obj type)? $\rightarrow$ SID
  - Allow (client SID, obj SID, perms)? $\rightarrow \{\text{yes, no}\}$

Creating new object



Security server interface

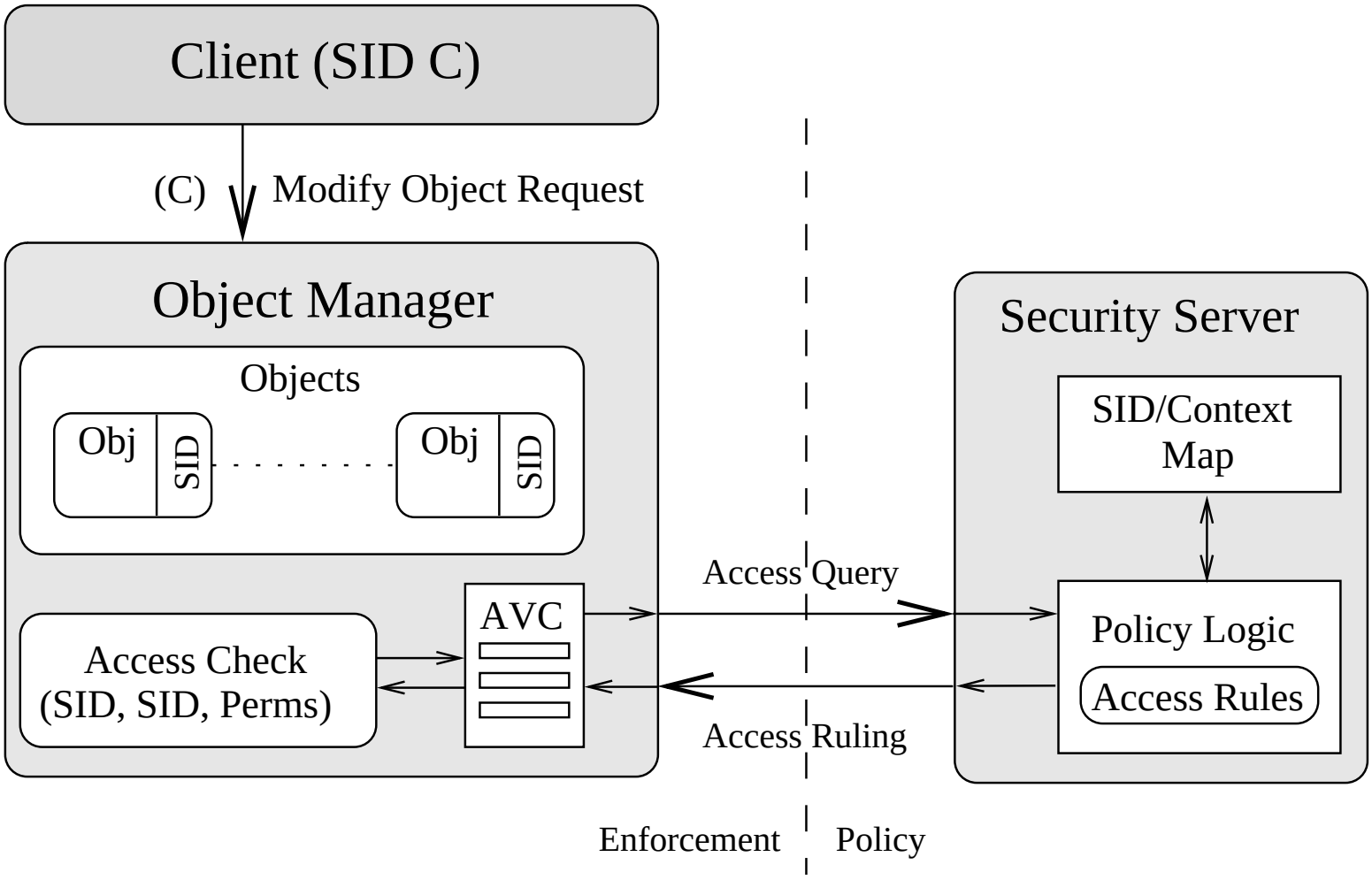
```
int security_compute_av(
    security_id_t ssid, security_id_t tsid,
    security_class_t tclass, access_vector_t requested,
    access_vector_t *allowed, access_vector_t *decided,
    __u32 *seqno);
```

- **Server can decide more than it is asked for**
  - `decided` will contain more than `requested`
  - Effectively implements decision prefetching

Access vector cache (AVC)

- **Want to minimize calls into security server**
- **AVC caches results of previous decisions**
  - Note: Relies on simple enumerated permissions
- **Decisions therefore cannot depend on paremeters:**
  - Andy can authorize expenses up to $999.99
  - Bob can run processes at priority 10 or higher
- **Decisions also limited to two SIDs**
  - Complicates file relabeling—see table 8

AVC in a query



AVC interface

```
int avc_has_perm_ref(
    security_id_t ssid, security_id_t tsid,
    security_class_t tclass, access_vector_t requested,
    avc_entry_ref_t *aeref);
```

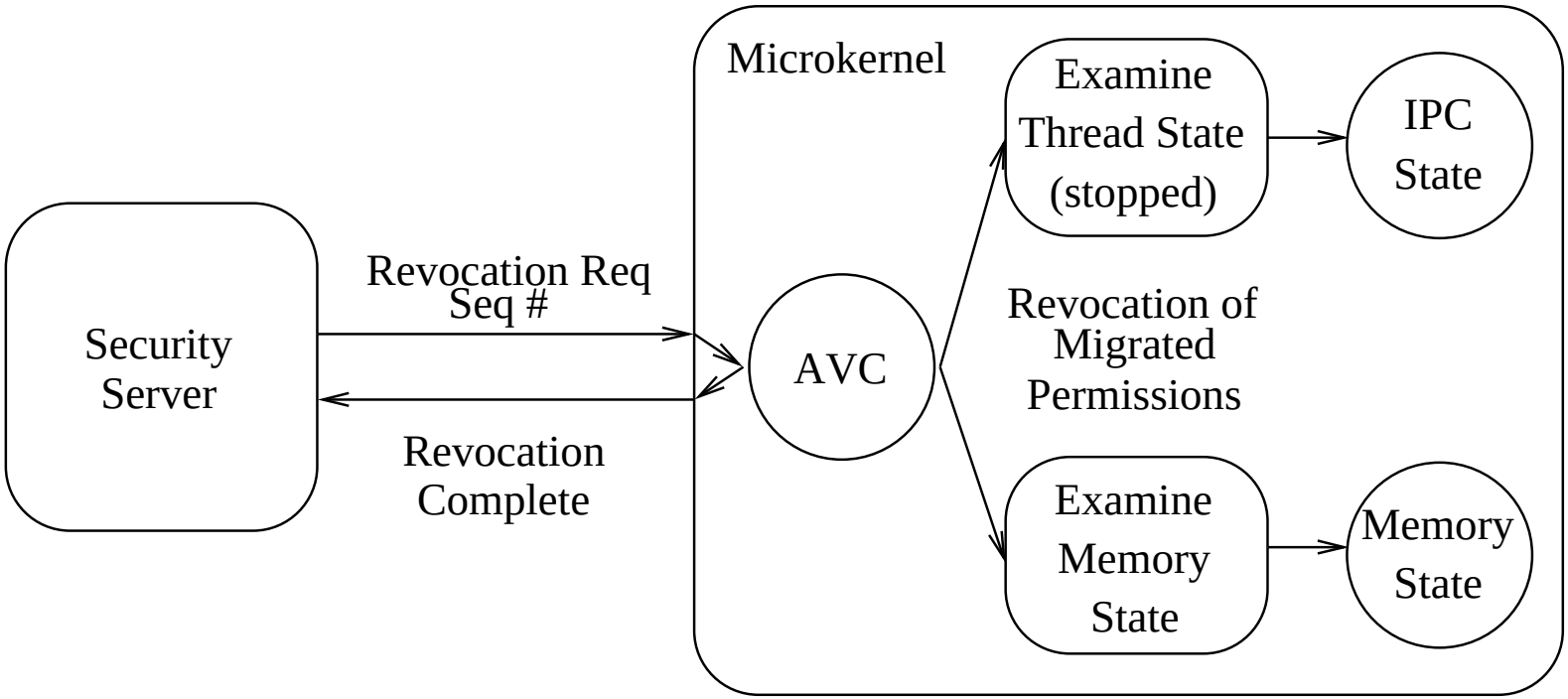
- `aeref` **argument is hint**
  - On first call, will be set to relevent AVC entry
  - On subsequent calls speeds up lookup
- **Example: Binding a socket:**

```
ret = avc_has_perm_ref(
    current->sid, sk->sid, sk->sclass,
    SOCKET__BIND, &sk->avcr);
```

Revocation support

- **Decisions may be cached in in AVCs**
- **Decisions may implicitly be cached in migrated permissions**
  - Unix file descriptors obtained after a file open
  - Memory mapped pages
  - Open sockets/pipes
- **AVC contains hooks for callbacks**
  - After revoking in AVC, AVC makes callbacks to revoke migrated permissions

Revocation protocol



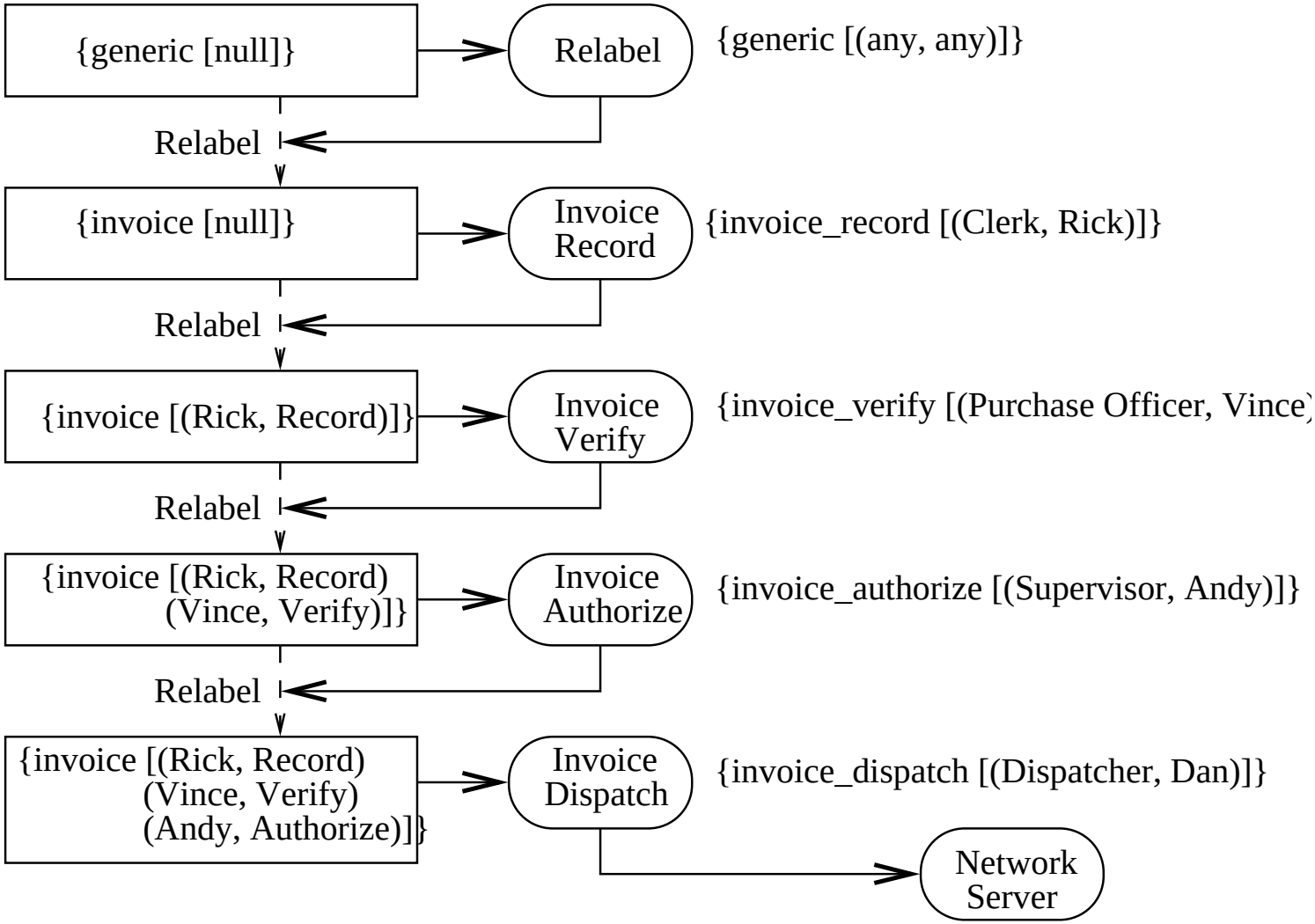
Transitioning SIDs

- **May need to relabel objects (e.g., files)**
  - See Fig 8
- **Processes may also want to transition their SIDs**
  - Depends on existing permission, but also on program
  - SElinux allows programs to be defined as *entrypoints*
  - Thus, one can restrict with which programs users enter a new SID

Example: Paying invoices

- **Invoices are special immutable files**
- **Each invoice must undergo the following processing:**
  - Receipt of the invoice recorded by a clerk
  - Receipt of of the merchandise verified by purchase officer
  - Payment of invoice approved by supervisor
- **Special programs allowed to record each of the above events**
  - E.g., force clerk to read invoice—cannot just write a batch script to relabel all files

Illustration



Example: Loading kernel modules

```
(1) allow sysadm_t insmod_exec_t:file x_file_perms;
(2) allow sysadm_t insmod_t:process transition;
(3) allow insmod_t insmod_exec_t:process { entrypoint execute };
(4) allow insmod_t sysadm_t:fd inherit_fd_perms;
(5) allow insmod_t self:capability sys_module;
(6) allow insmod_t sysadm_t:process sigchld;
```

1: Allow sysadm domain to run insmod

2: Allow sysadm domain to transition to insmod

3: Allow insmod program to be entrypoint for insmod domain

4: Let insmod inherit file descriptors from sysadm

5: Let insmod use CAP_SYS_MODULE (load a kernel module)

6: Let insmod signal sysadm with SIGCHLD when done

Stretch break