Midterm Exam

- **Next monday March 25**
  - Only one short paper to read for class
- **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**
  - There will be at least one question on protocols
- **Office hours after class**
  - Extra office hours? Thursday or Friday?

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
- **Three general approaches to capabilities:**
  - Hardware enforced (Tagged architectures like M-machine)
  - Kernel-enforced (like Hydra)
  - 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

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**

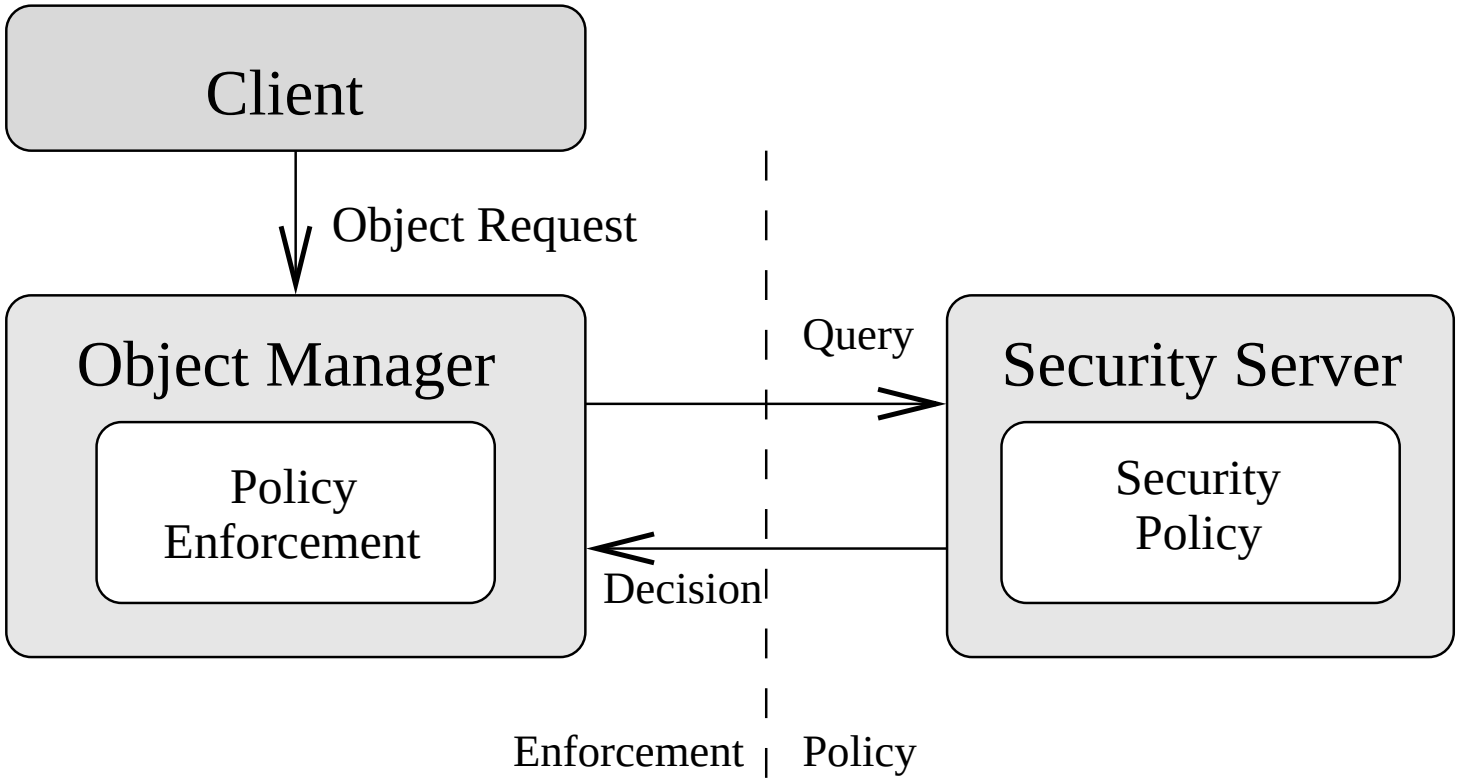
Janus—security through interposition

- **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**

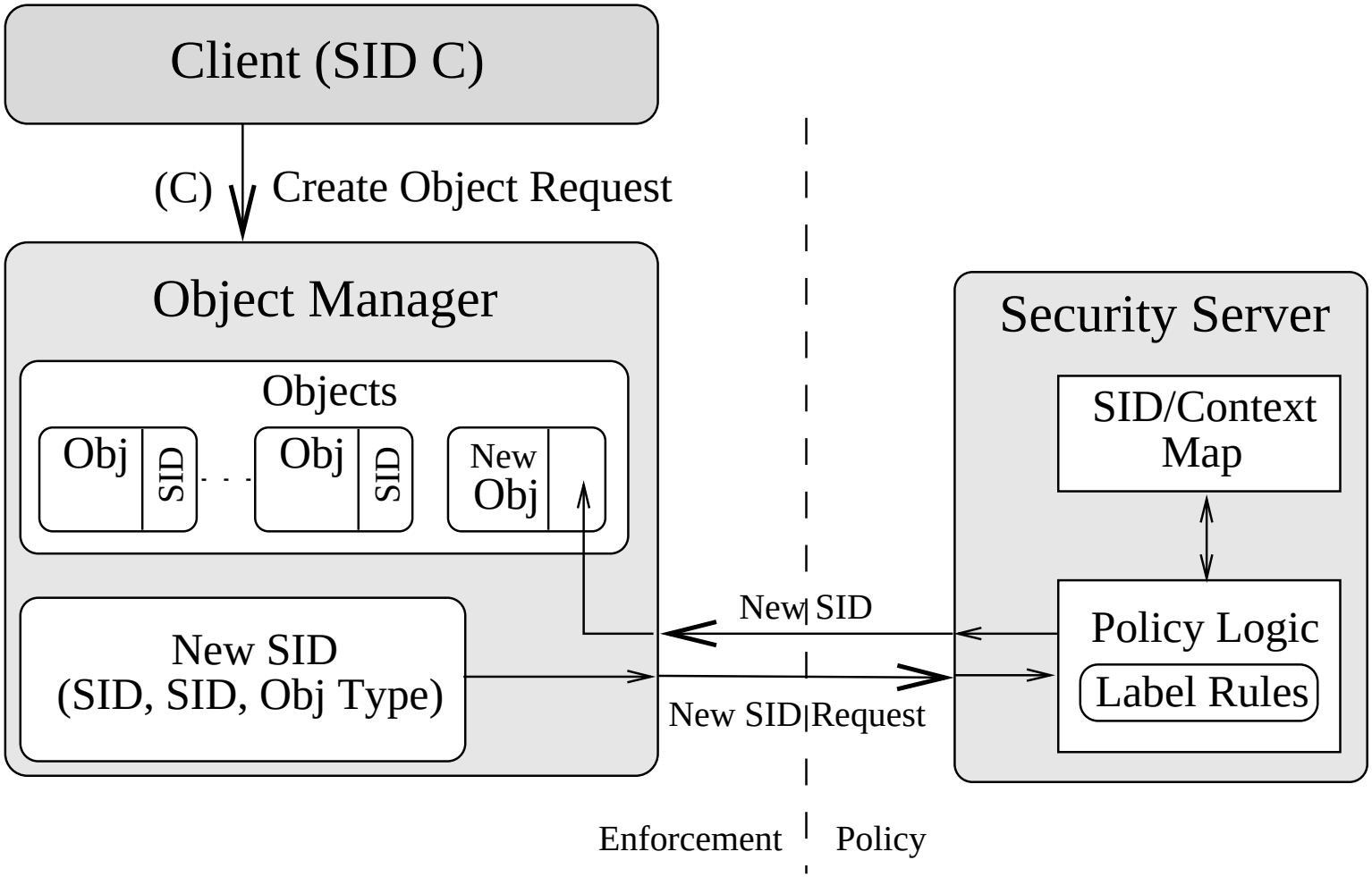
Challenges

- **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

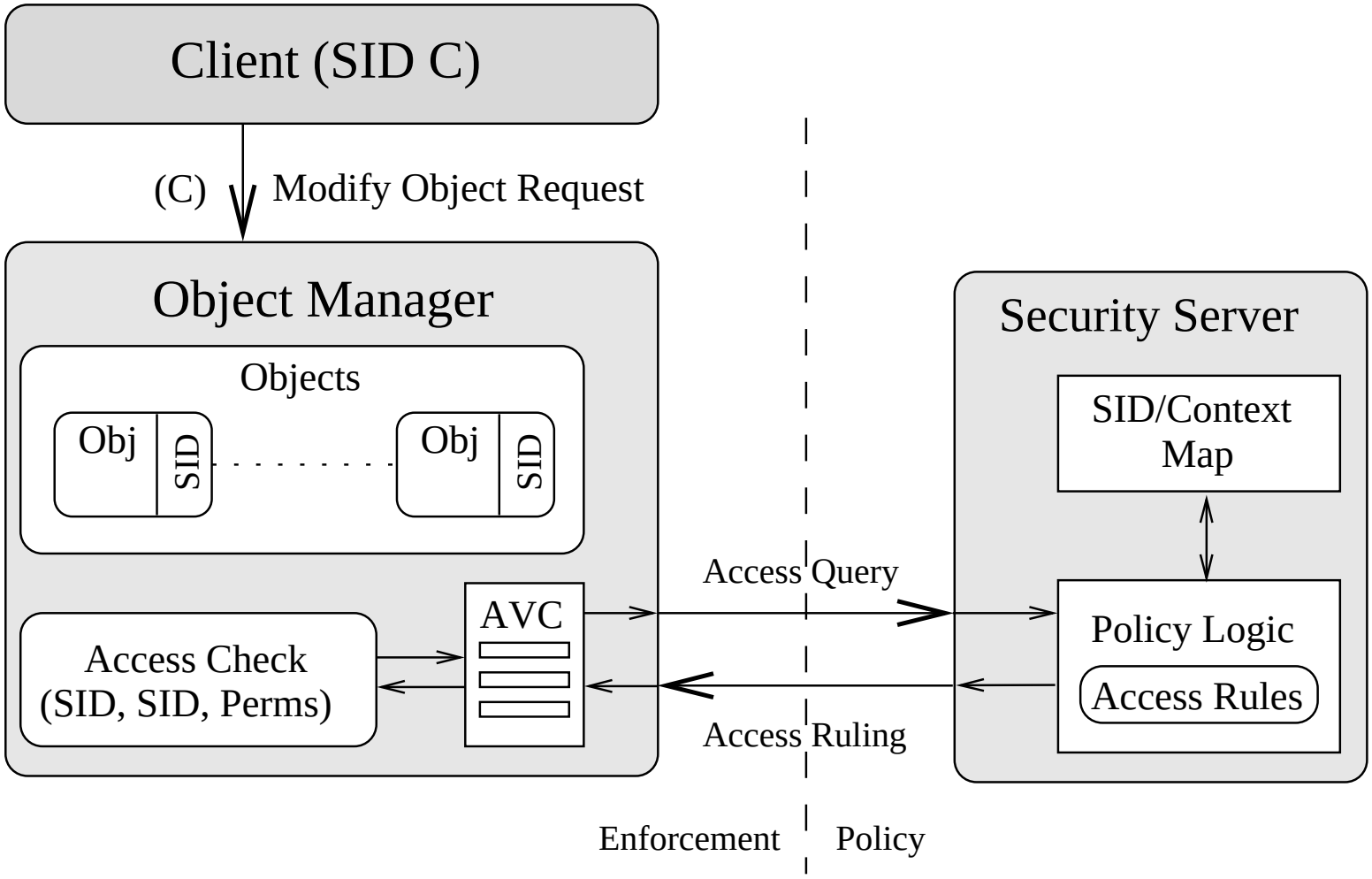
```
int security_compute_av(
    security_id_t ssid, security_id_t tsid,
    security_class_t tclass, access_vector_t requested,
    access_vector_t *allowed, access_vector_t *decided,
    __u32 *seqno);
```

- **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 fig 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);
```

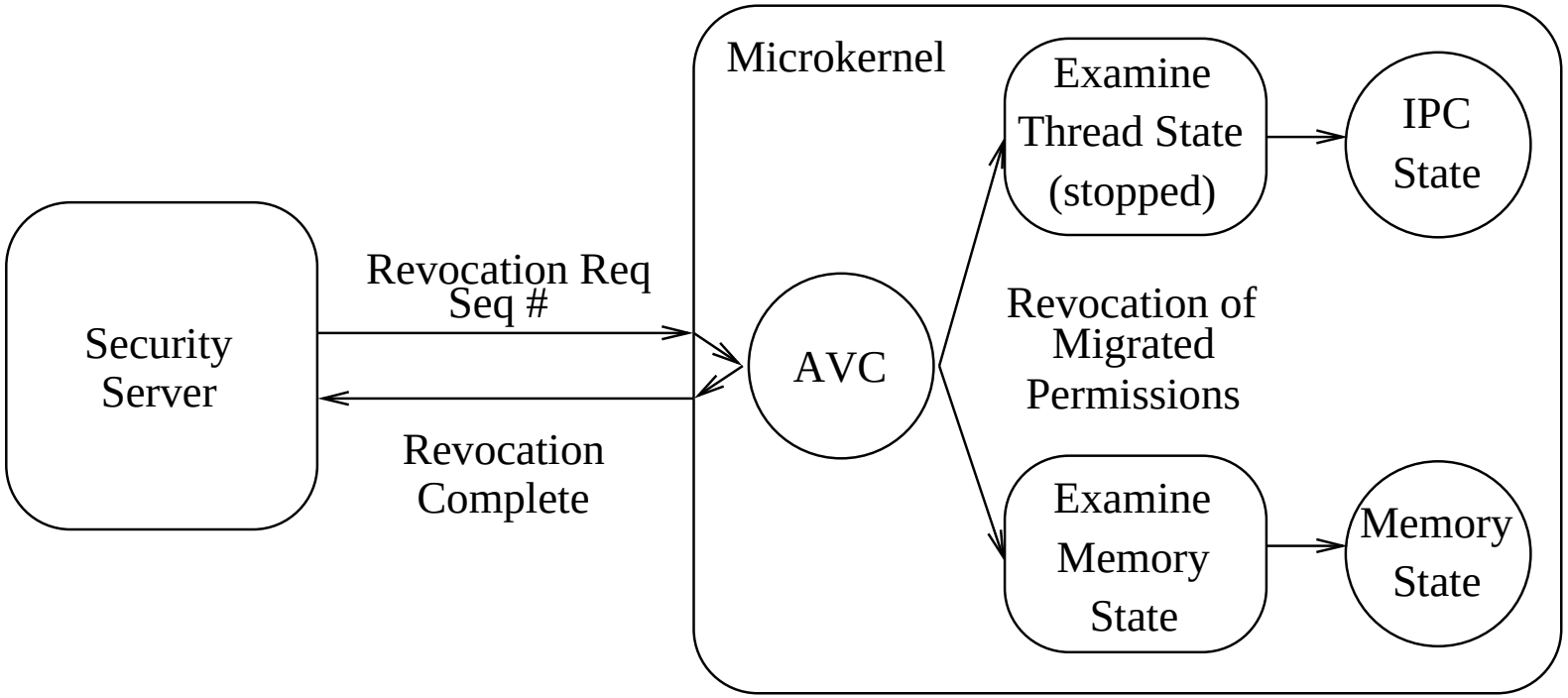
- `aeref` **argument is hint**
  - On first call, will be set to relevent AVC entry
  - On subsequent calls speeds up lookup
- **Example: Binding a socket:**

```
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



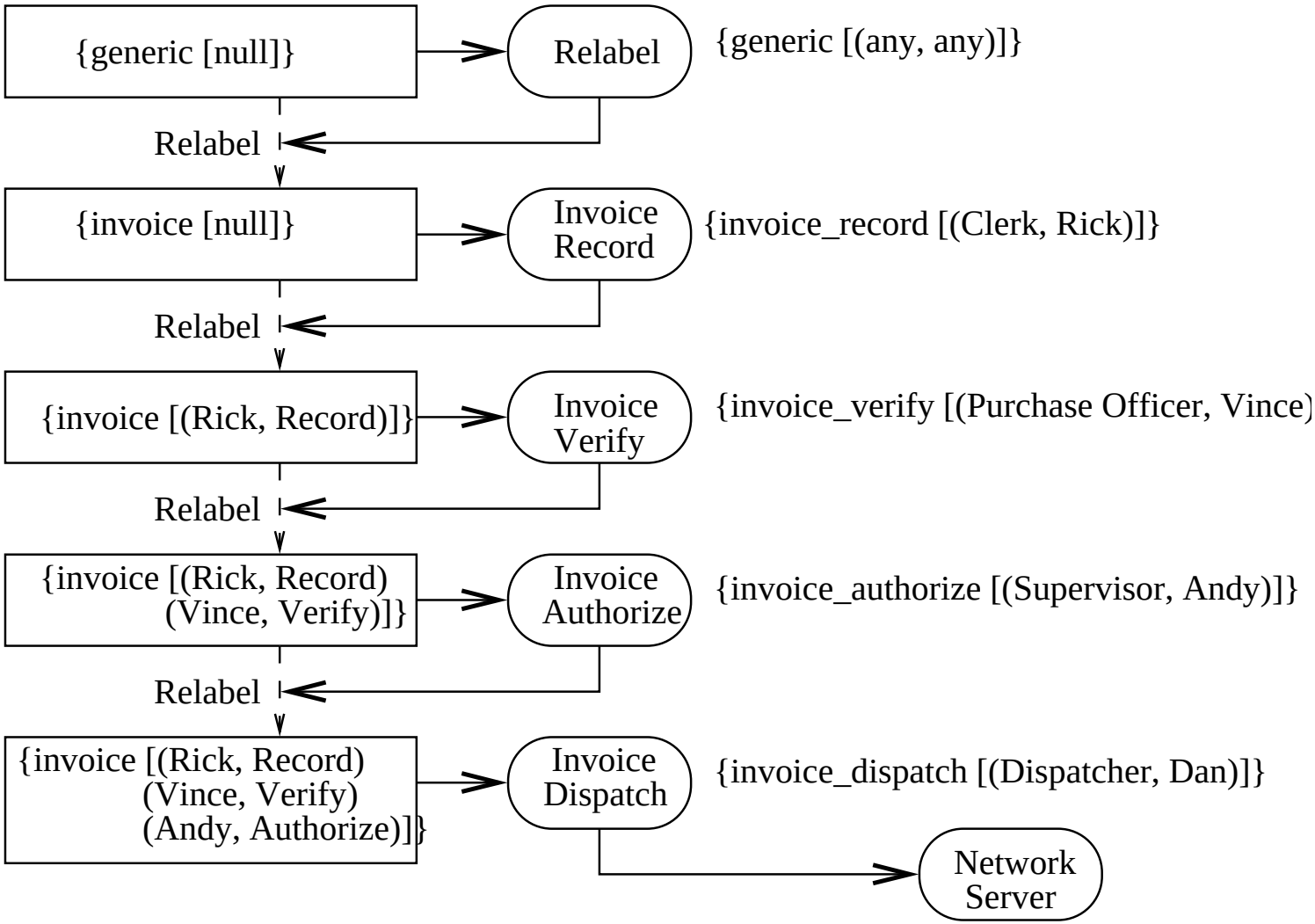
Transitioning SIDs

- **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

Questions to ask about protocols

- **Is everything explicitly stated in messages?**
- **Does one party act as a decryption/signing oracle?**
  - E.g., kerberos kinit decrypts data and sends it onto network
- **Can one party to impersonate the other?**
- **Does the protocol provide freshness?**
  - Might some message be recyclable in a replay attack?
- **Does the protocol provide forward secrecy?**
  - Long-lived keys should be for authentication, not secrecy

Questions to ask about passwords

- **Who can mount off-line guessing attacks?**
  - Should the system be using something like SRP?
- **Is the password "salted"?**
  - Is there a cost parameter? (Usually not, but it's a good idea.)
- **What are consequences of server compromise?**
  - For users who use same password on several systems?
  - For users with good (hard to guess) passwords?
- **Is any timing information leaked?**
  - e.g., `strcmp (passwd, system_passwd)`

Web security

- **SSL authenticates server to client**
  - Based on globally-trusted CA
  - SSL usually not used for user-authentication
- **User-authentication usually w. passwords/cookies**
  - User supplies password, gets cookie from server
  - Future requests from client contain the cookie
- **Beware home-brew cookie authentication schemes**
  - "Interrogative adversary" surprisingly powerful attacker
  - Must not be possible for attacker to make up valid cookies
  - Don't rely on clients to invalidate cookies

Kerberos

- **Basic idea: Emulate CA without public key crypto**
  - Users and servers each share a secret with trusted KDC
  - KDC leverages shared secrets to establish session keys
- **Basic protocol:**
  - Ticket: $T = \{s, c, \text{addr}, \text{expire}, K_{s,c}\}_{K_s}$
    (K_S is key s shares with KDC)
  - Authenticator: $A = T, \{c, \text{addr}, \text{time}\}_{K_{s,c}}$
- **Login protocol:**
  - $c \rightarrow t$: c, t (t is name of TG service)
  - $t \rightarrow c$: $\{K_{c,t}, T_{c,t} = \{s, t, \text{addr}, \text{expire}, K_{c,t}\}_{K_t}\}_{K_c}$
  - $c \rightarrow t$: $s, T_{c,t}, \{c, \text{addr}, \text{time}\}_{K_{c,t}}$
  - $t \rightarrow c$: $\{T_{s,c}, K_{s,c}\}_{K_{c,t}}$

SDSI

- **The SDSI/SPKI public key infrastructure**
  - No global names, as in other infrastructures
  - All terms must begin with a Key: K_{dm} verisign
- **Name certificates bind terms to names in local namespace**
  - Can reference other people's namespaces, e.g.: K_{dm} verisign shop.com
- **Authorization certificates specify actions allowed to which principals**
- **Clarke result: Terms efficiently evaluatable**

SFS

- **Goal: Allow many different key management schemes to coexist and share the same file system**
- **Philosophy very close to SDSI**
  - Egalitarian file namespace puts public keys in filenames
  - Lets file namespace double as key certification namespace
  - Use symbolic links to assign human readable names to keys
- **Adds ability to lookup public keys on the fly**
  - Dynamic servar authentication spawns external programs
  - Can compose multiple certification techniques

Mandatory access control (MAC)

- **Goal: Prevent information flow**
  - E.g., Just because A can read a classified file, does not mean he should be able to send the contents to B
- **Classify data by sensitivity (secret, top secret, . . .)**
  - Security policy prevents "read up" and "write down"
- **Classify data by compartments**
  - E.g., cryptography, nuclear, special intelligence, . . .
- **Integrity uses same mechanism upside down**

Jif

- **Controls information flow at programming language level**
  - Assumes trusted execution environment
  - Lets users run untrusted code on sensitive data
- **Makes declassification part of the model**
  - Labels remember which users impose which restrictions
  - E.g., $\{o_1 : p_1;\ o_2 : p_2\}$
- **Nice language features**
  - Automatic label inference, label polymorphism, etc.