Lecture 32:
Isolation & Authorization

Professor Adam Bates
CS 461 / ECE 422
Fall 2019

Goals for Today

- <u>Learning Objectives</u>:
  - Define Isolation, its properties, and why is isolation important
  - Understand at what levels isolation can be implemented
  - Provide examples at each level
- <u>Announcements, etc</u>:
  - Networking CP2: Due Nov 11
  - **<u>Wednesday is cancelled!</u>**
    - … not really, but no lecture or discussion
    - Next lecture is on Friday

Reminder: Please put away devices at the start of class

Security Properties

- Confidentiality?
  - Only trusted parties can read data
- Integrity?
  - Only trusted parties have modified data
- Authenticity?
  - Data originates from the correct party
- Availability?
  - Data is available to trusted parties when needed

Security Functions

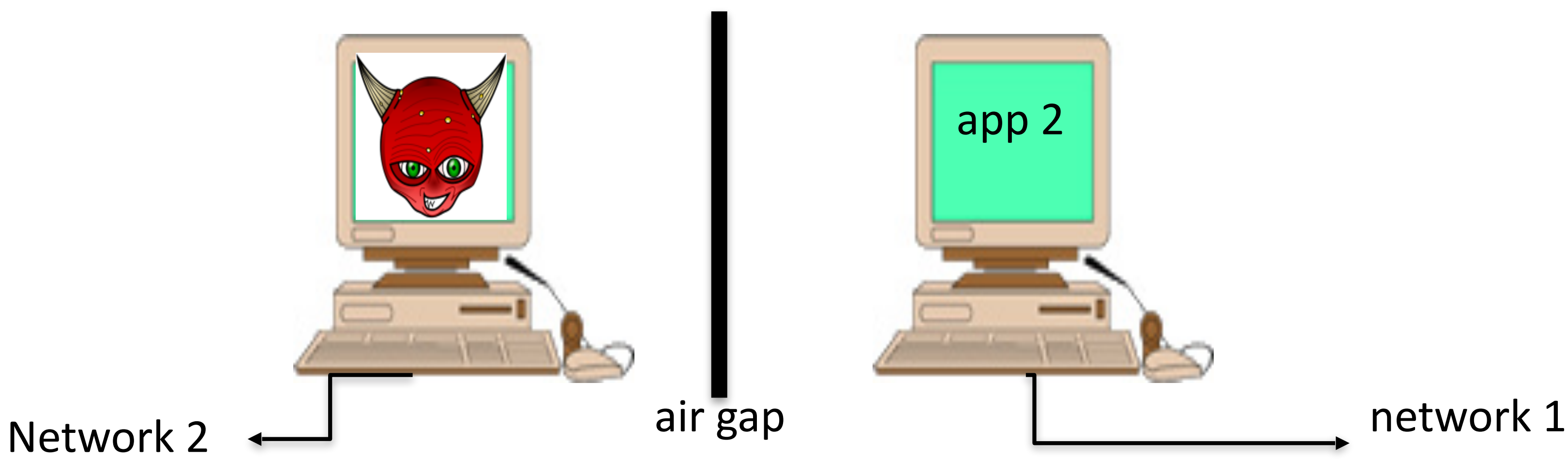
- Define security functions over *principals* (e.g., users, programs, sysadmins)
- … and also *entities* (e.g., files, network sockets, ipc)
- Authentication
  - How do we determine the identity of the principal?
- Authorization
  - Which principals are permitted to take what actions on which objects?
- Auditing
  - Record of (un) authorized actions that took place on the system for post-hoc diagnostics

Running untrusted code

- We often need to run buggy/unstrusted code:
  - programs from untrusted Internet sites:
    - apps, extensions, plug-ins, codecs for media player
  - exposed applications: pdf viewers, outlook
  - legacy daemons: sendmail, bind
  - honeypots
- Goal: if application "misbehaves" ⇒ kill it

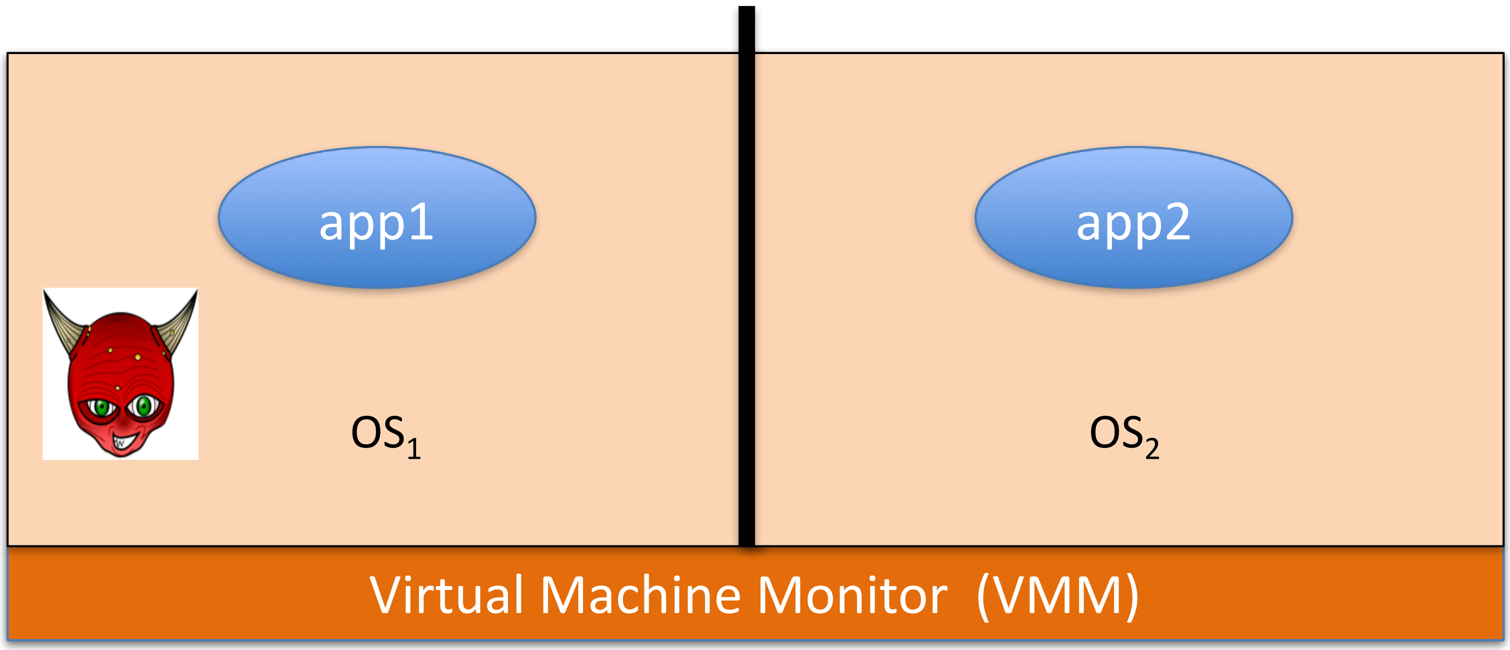
Approach: Confinement

- Confinement: ensure app cannot affect rest of system, or vice versa
- Can be implemented at many levels:
  - Hardware: run application on isolated hw (air gap)



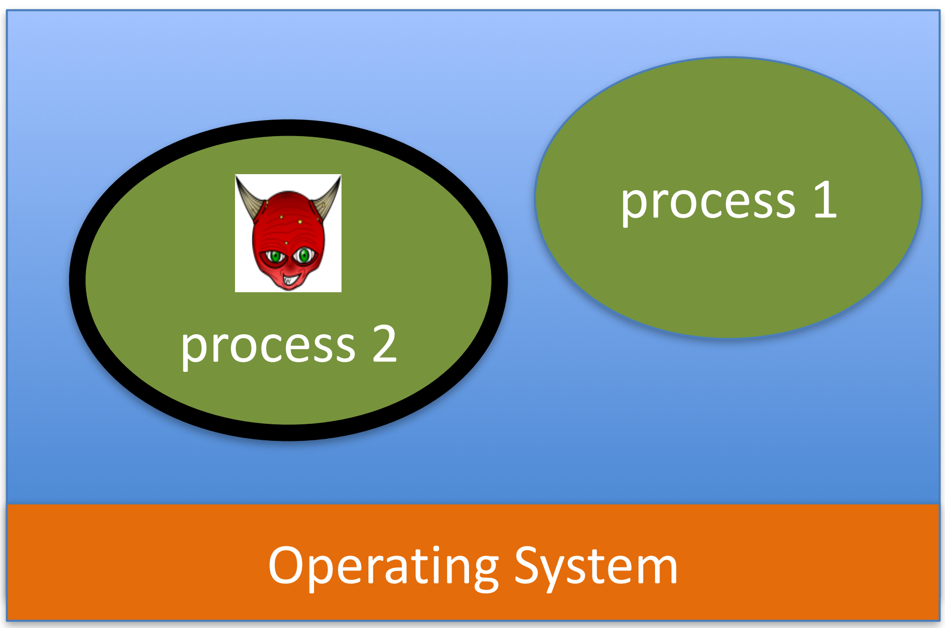
Approach: Confinement

- Confinement: ensure app cannot affect rest of system, or vice versa
- Can be implemented at many levels:
  – Virtual machines: isolate OS's on a single machine



Approach: Confinement

- Confinement: ensure app cannot affect rest of system, or vice versa
- Can be implemented at many levels:
  - Process: System Call Interposition, Isolate a process in a single operating system



A old example: chroot

- Often used for "guest" accounts on ftp sites

- To use do: (must be root)

chroot /tmp/guest	root dir "/" is now "/tmp/guest"
su guest	EUID set to "guest"

- Now "/tmp/guest" is added to file system accesses for applications in jail
- open("/etc/passwd", "r") $\Rightarrow$ open("/tmp/guest/etc/passwd" ,"r")
- application cannot access files outside of jail

Jailkit

- Problem: all utility progs (ls, ps, vi) must live inside jail
- jailkit project: auto builds files, libs, and dirs needed in jail env
  - jk_init: creates jail environment
  - jk_check: checks jail env for security problems
    - checks for any modified programs,
    - checks for world writable directories, etc.
  - jk_lsh: restricted shell to be used inside jail
- note: simple chroot jail does not limit network access

Escaping from jails

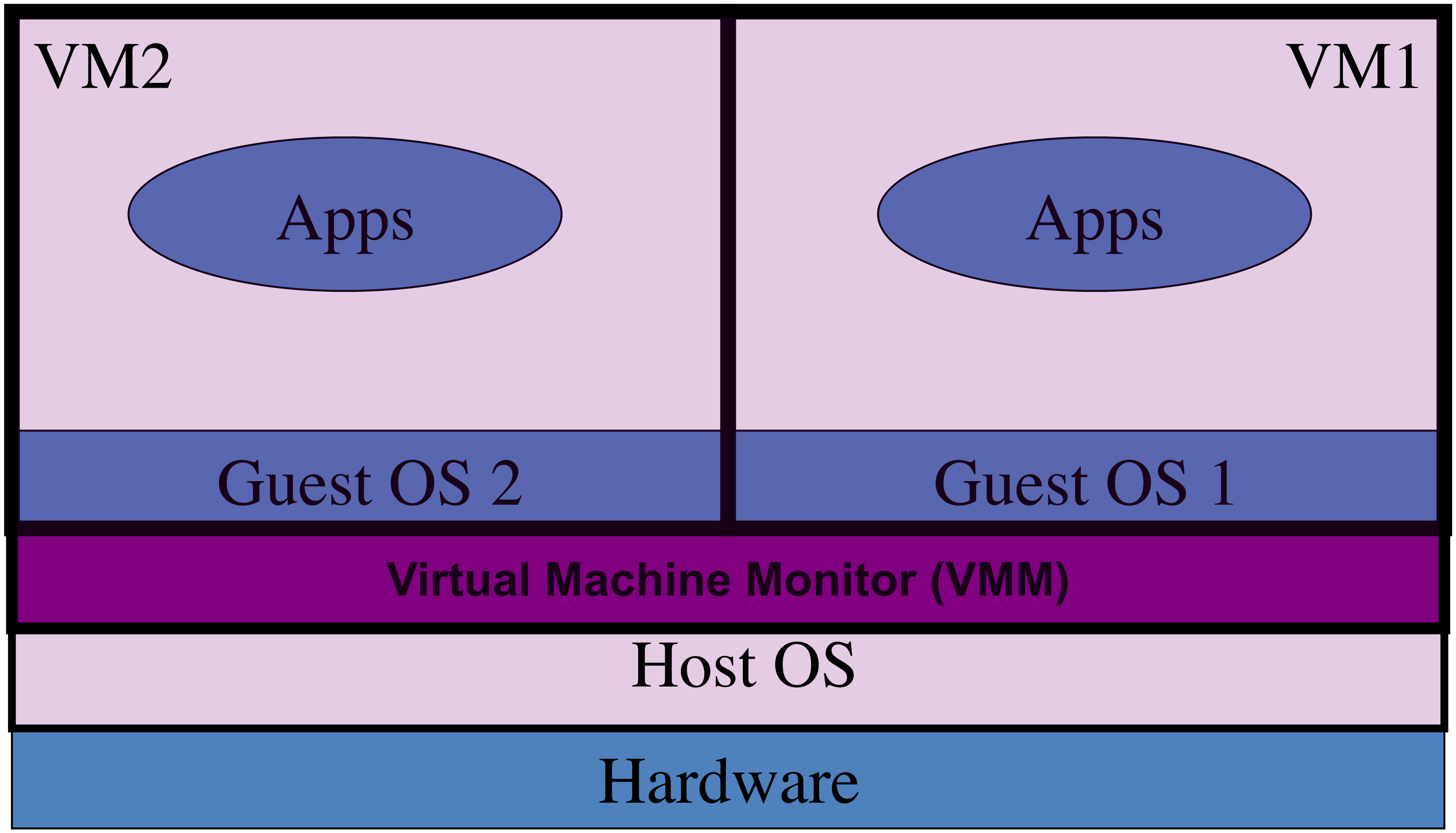
- Early escapes: relative paths

  open("../../etc/passwd", "r") ⇒
  open("/tmp/guest/../../etc/passwd", "r")

- chroot should only be executable by root.
  – otherwise jailed app can do:
    - create dummy file "/aaa/etc/passwd"
    - run chroot "/aaa"
    - run su root to become root

(bug in Ultrix 4.0)

Virtual Machines



Example: **NSA NetTop**

single HW platform used for both classified and unclassified data

Why so popular now?

- VMs in the 1960's:
  - Few computers, lots of users
  - VMs allow many users to shares a single computer
- VMs 1970's – 2000: non-existent
- VMs since 2000:
  - Too many computers, too few users
    - Print server, Mail server, Web server, File server, Database , …
  - Wasteful to run each service on different hardware
  - More generally: VMs heavily used in cloud computing

VMM security assumption

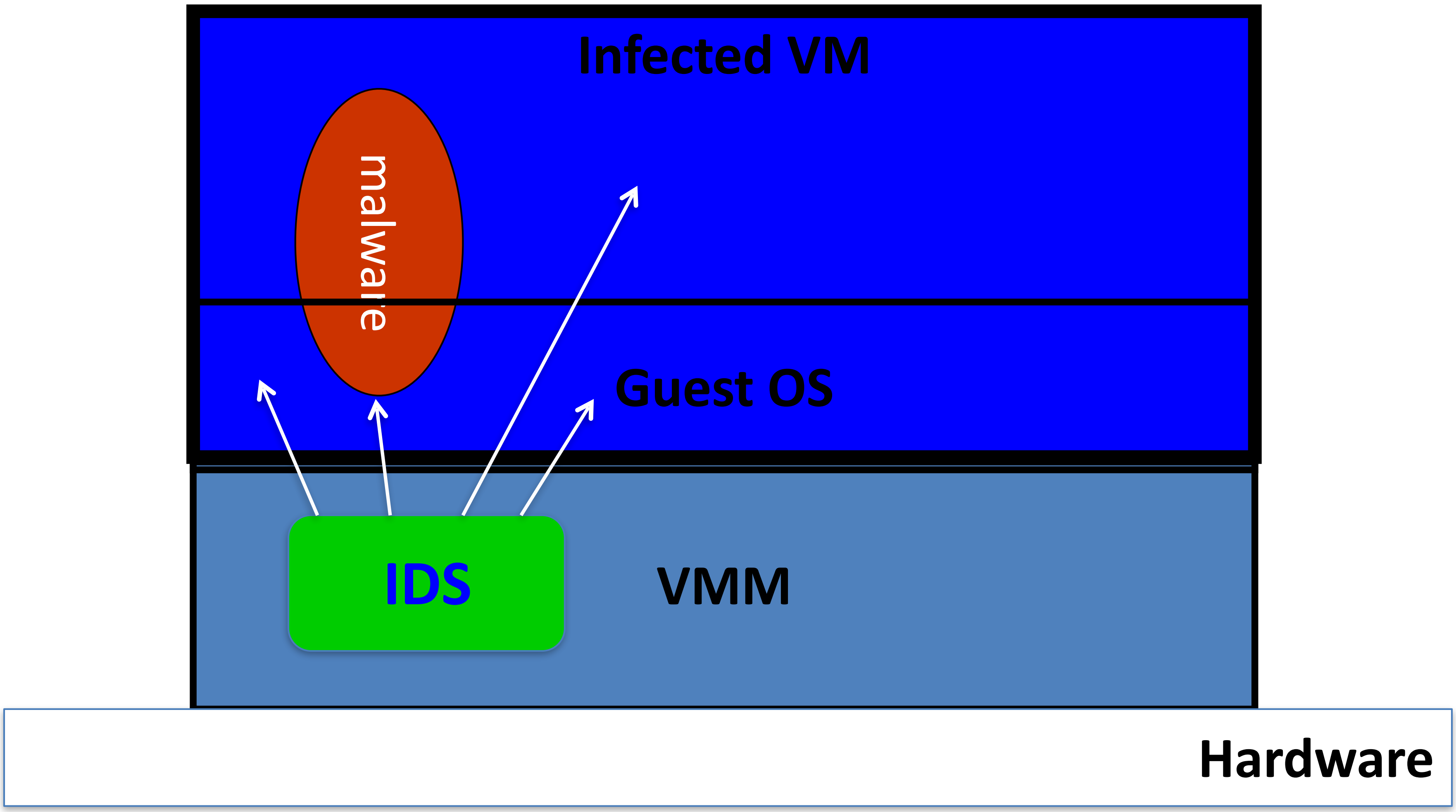
- VMM Security assumption:
  - Malware can infect guest OS and guest apps
  - But malware cannot escape from the infected VM
    - Cannot infect host OS
    - Cannot infect other VMs on the same hardware

- Requires that VMM protect itself and is not buggy
  - VMM is much simpler than full OS
  - … but device drivers run in Host OS

Intrusion Detection / Anti-virus

- Runs as part of OS kernel and user space process
  - Kernel root kit can shutdown protection system
  - Common practice for modern malware
- Standard solution: run IDS system in the network
  - Problem: insufficient visibility into user's machine
- Better: run IDS as part of VMM (protected from malware)
  - VMM can monitor virtual hardware for anomalies
  - VMI: Virtual Machine Introspection
    - Allows VMM to check Guest OS internals

Intrusion Detection / Anti-virus



Sample checks

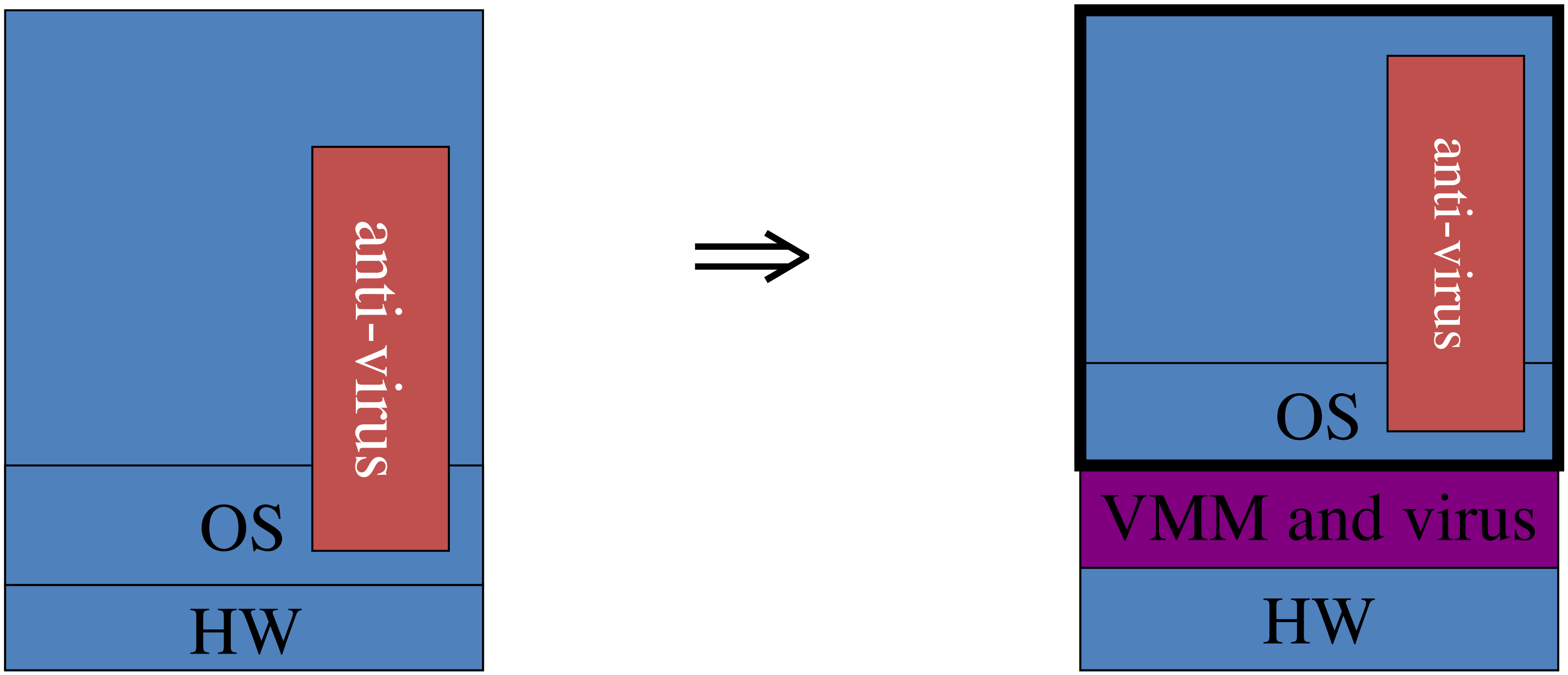
- Stealth root-kit malware:
  - Creates processes that are invisible to "ps"
  - Opens sockets that are invisible to "netstat"
- 1. Lie detector check
  - Goal: detect stealth malware that hides processes and network activity
  - Method:
    - VMM lists processes running in GuestOS
    - VMM requests GuestOS to list processes (e.g. ps)
    - If mismatch: kill VM

Sample checks

- 2. Application code integrity detector
  - VMM computes hash of user app code running in VM
  - Compare to whitelist of hashes
    - Kills VM if unknown program appears
- 3. Ensure GuestOS kernel integrity
  - example: detect changes to sys_call_table
- 4. Virus signature detector
  - Run virus signature detector on GuestOS memory

Subvirt [King et al. 2006]

- Problem: Attackers can leverage VM isolation too
  - Once on victim machine, install a malicious VMM
  - Virus hides in VMM
  - Invisible to virus detector running inside VM



Problems with chroot, VM isolation

- ***Isolation is extremely <u>coarse-grained</u>***:
  - – All or nothing access
  - – Inappropriate for apps like a web browser
    - Needs read access to files outside jail
      (e.g. for sending attachments in Gmail)
- chroot does not prevent malicious apps from:
  - – Accessing network and messing with other machines
  - – Trying to crash host OS
- In practice, processes need to be able to cooperate

Authorization

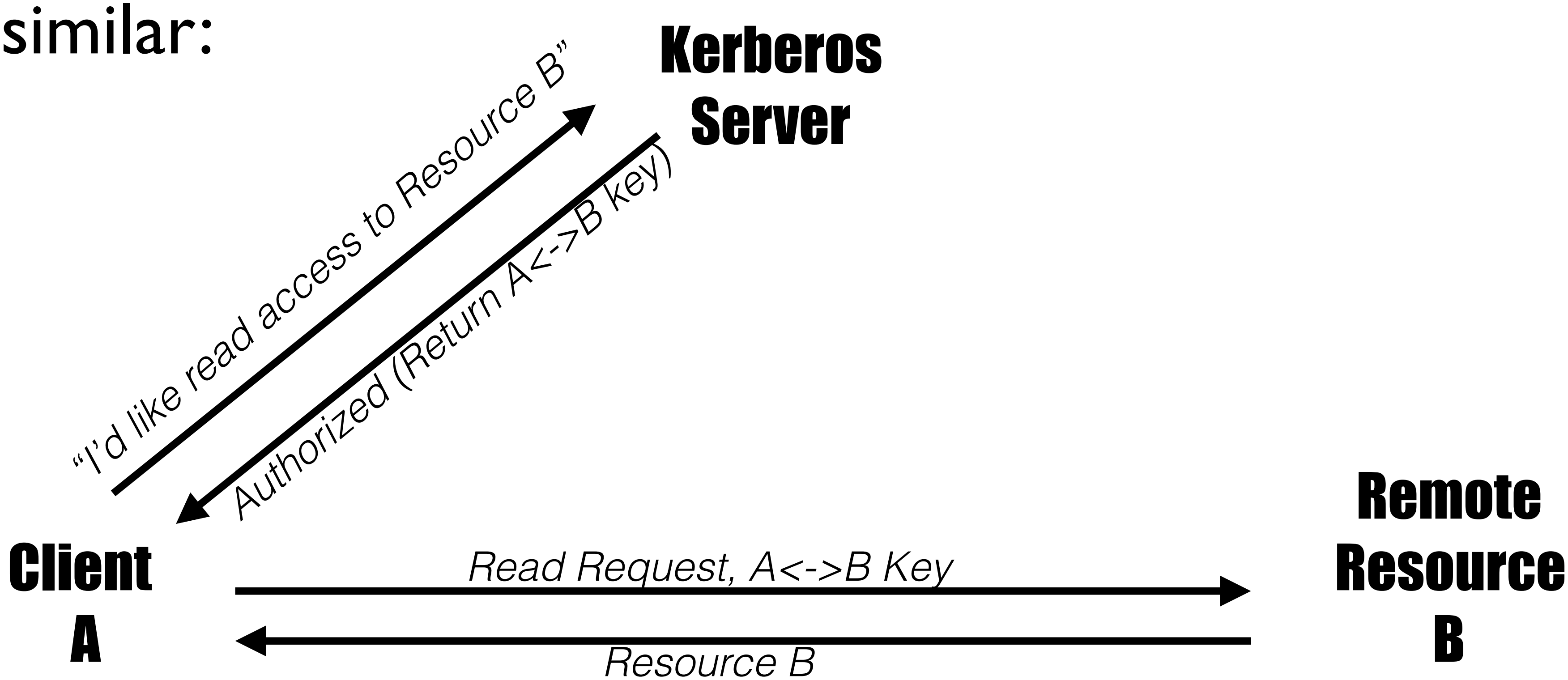
- Access control matrix
  - For every protected resource, list of who is permitted to do what
  - Example: for each file/directory, a list of permissions
    - Owner, group, world: read, write, execute
    - Setuid: program run with permission of principal who installed it
  - Smartphone: list of permissions granted each app

Crypto v. Access Control

- Two competing primitives for achieving security
- Both can be used to achieve assurances of data confidentiality, integrity, authenticity
- Crypto: "Encrypt the data for security!"
  - Key is authorization mechanism to assure confidentiality, integrity, authenticity properties.
- Access Control: "Label the data for security!"
  - Software includes authorization mechanism to assure confidentiality, integrity, authenticity properties according to policy.

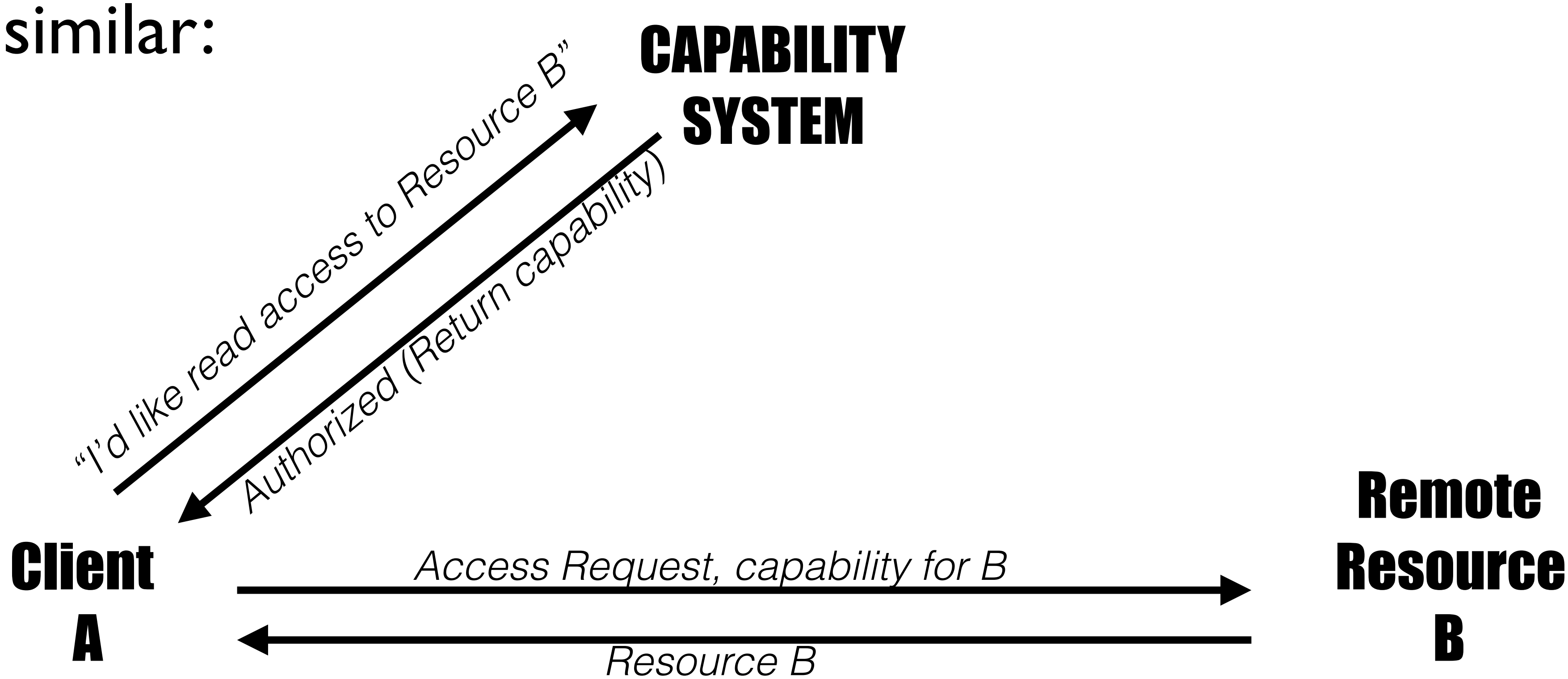
Crypto v. Access Control

- 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:



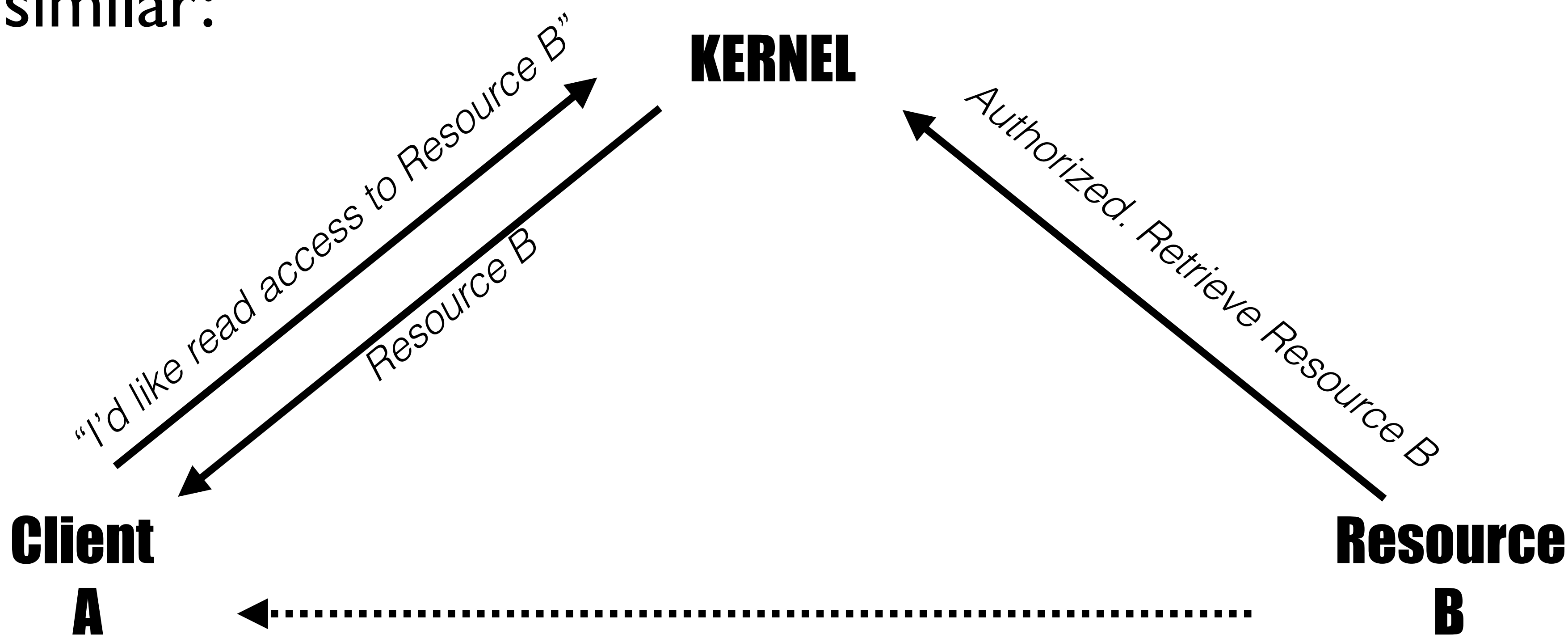
Crypto v. Access Control

- 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, primarily for verifying signatures…

…but in the kernel, access control is king!

Access Control

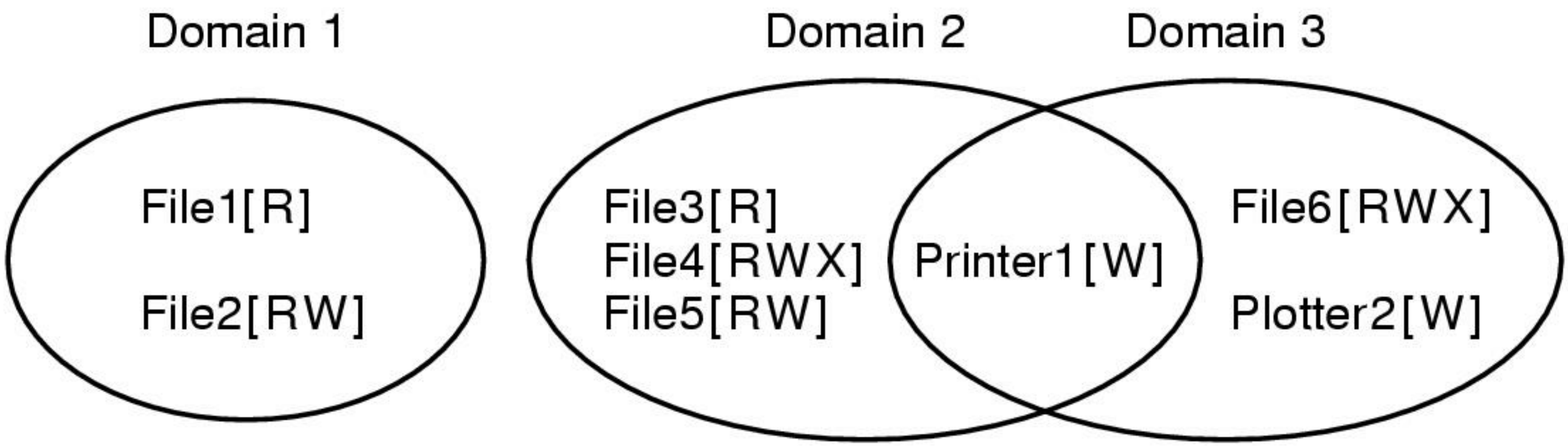
- Determine whether a principal can perform a requested operation on a target object
- **Principal/Subject**: user, process, etc.
- **Operation/Action**: read, write, etc.
- **Object**: file, tuple, etc.

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 given the need of that process to accomplish its assigned task.

Protection Domains

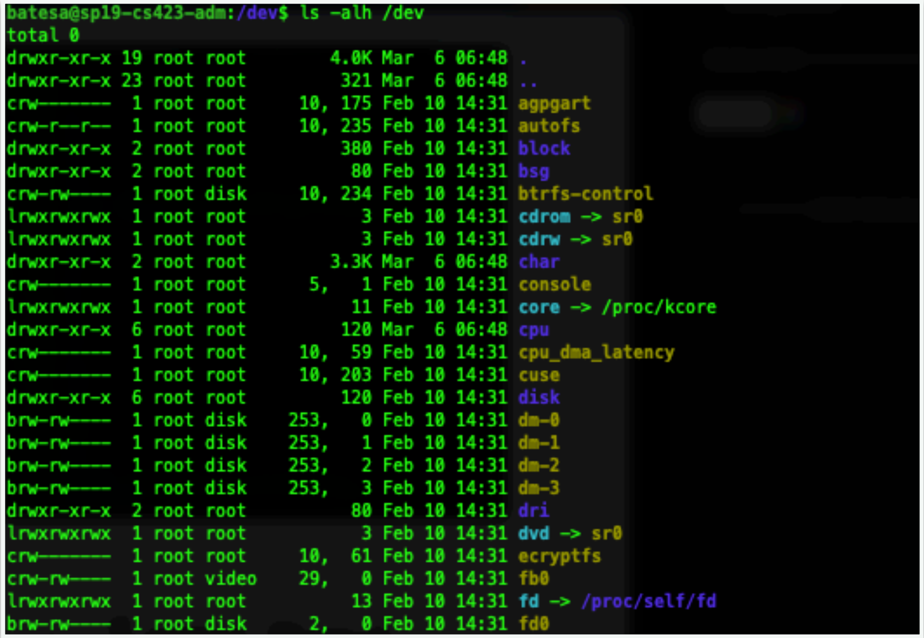
Example of 3 protection domains:



What are domains based on in Linux?

Discretionary Access Control (DAC)

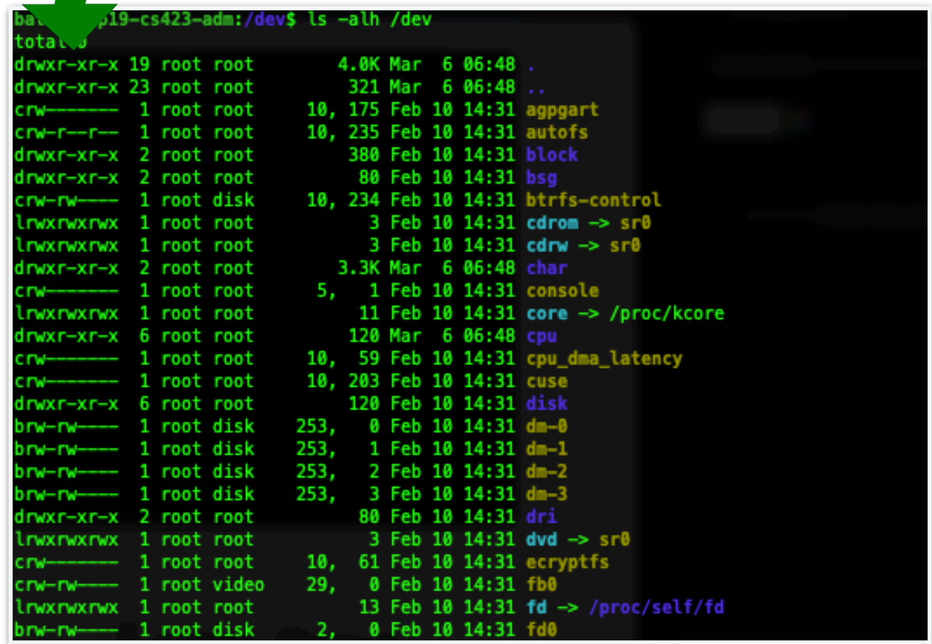
Find the access control!



- Owner of creator of resources specify which subjects have which access to those 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
```

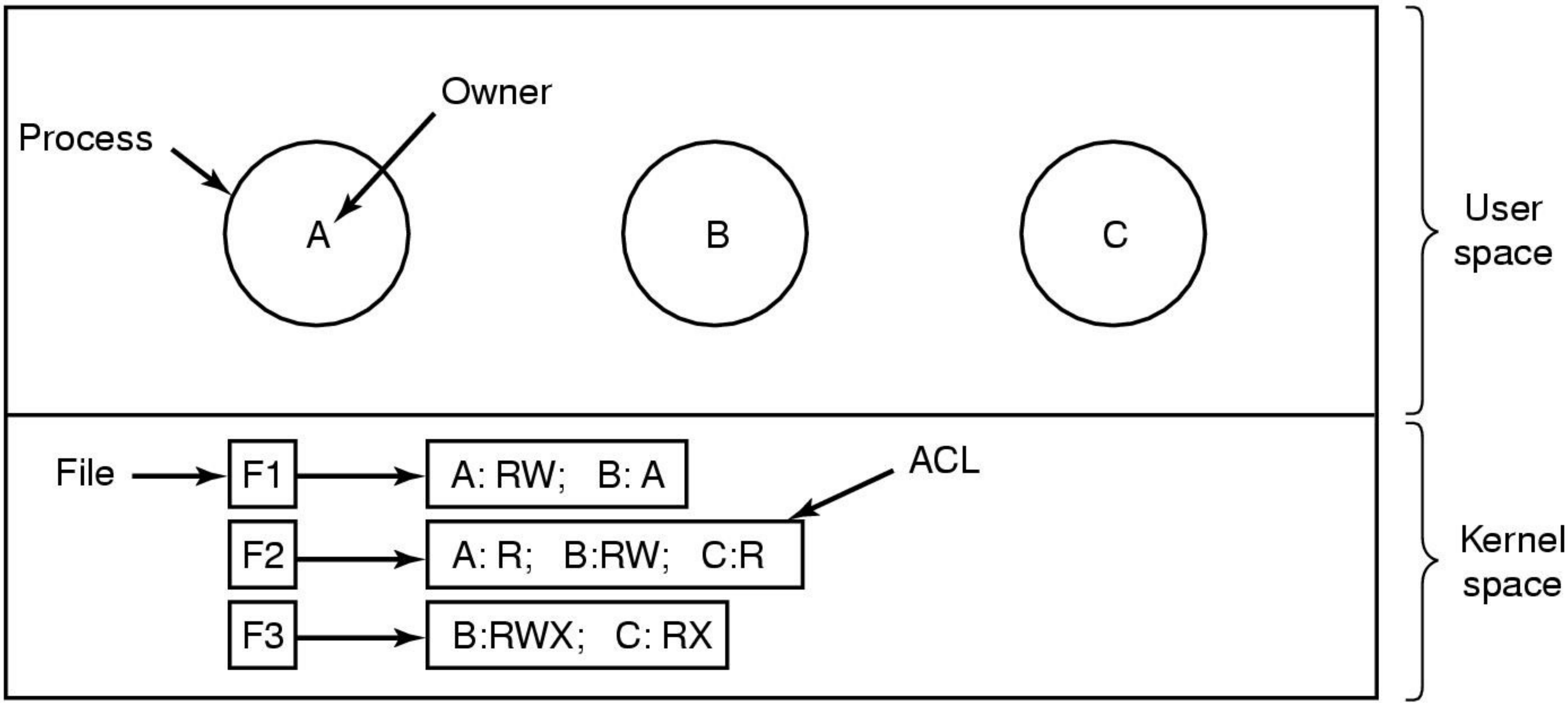
<- Same thing

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

Access Control Lists (ACLs)

- 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.

UNIX ACLs

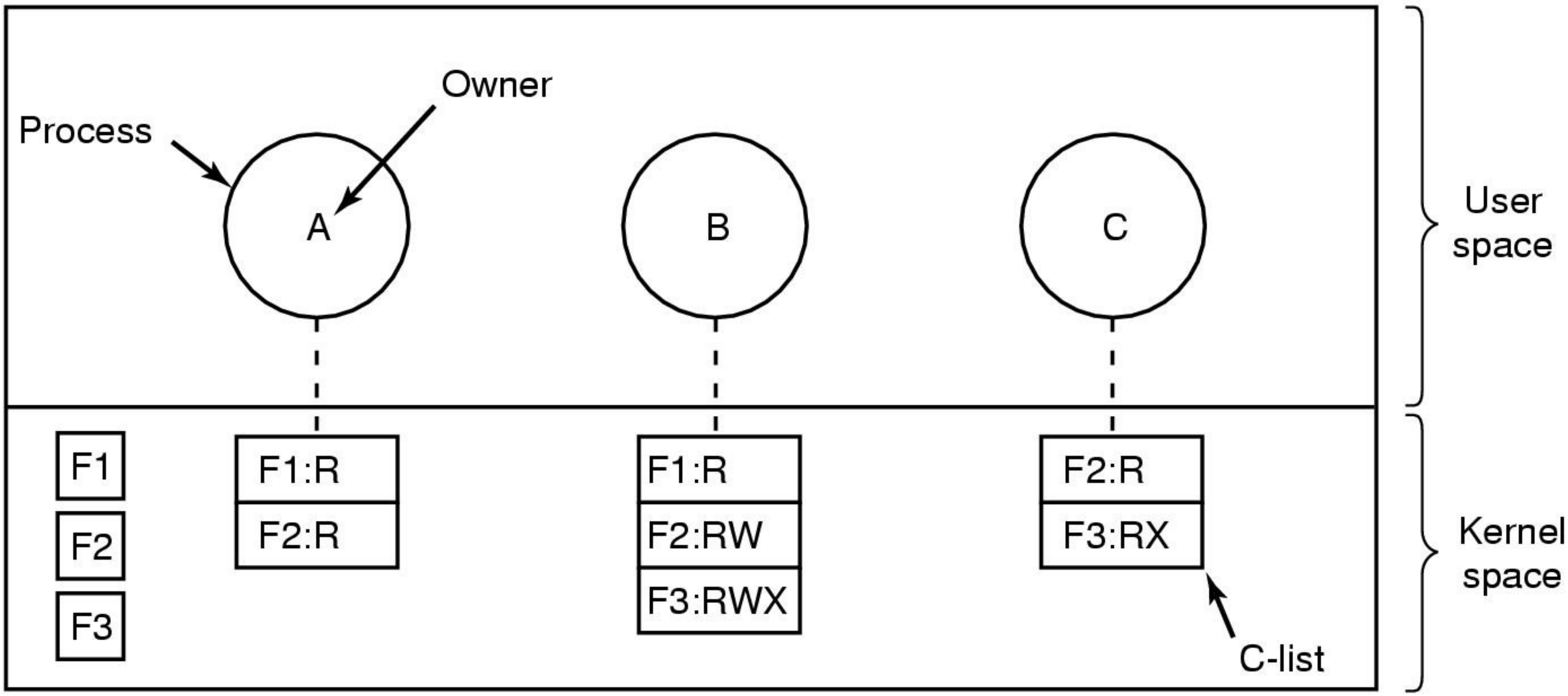


Use of access control lists to manage file access in UNIX

Capabilities

- An alternative access control mechanism
- 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?
  - <u>Mandatory Access Control</u>: Operating system constrains the ability of subjects (even owners) to perform operations on objects according to a system-wide *security policy*.

To Learn More ...

- Books
  - Stallings and Brown, Chapter 12
  - Pfleeger and Pfleeger, Chapter 5
  - Goodrich and Tamassia, Chapter 3
  - Bishop, Chapter 16, 26, 29
- Papers
  - App Isolation: Get the Security of Multiple Browsers with Just One - Chen
  - Native Client: A Sandbox for Portable, untrusted x86 Native Code - Yee*
  - Innovative Instructions and Software Model for Isolated Execution - McKeen
  - The Security Architecture of the Chromium Browser – Barth*