CS165 – Computer Security

Access Control, BLP and Biba Policies
Nov 2, 2021

Agenda

NEXT

- Access Control Matrix
  - Overview
  - Access Control Matrix Model
  - Protection State Transitions
    - Commands
    - Conditional Commands
- Foundational Results

Protection States

- An abstraction that focuses on security properties
  - ✓ Primarily interested in characterizing Safe states
  - ✓ Goal is to prove that all operations in the system preserve "security" of the protection state
  - ✓ **Access Control Matrix** is our first Protection State model

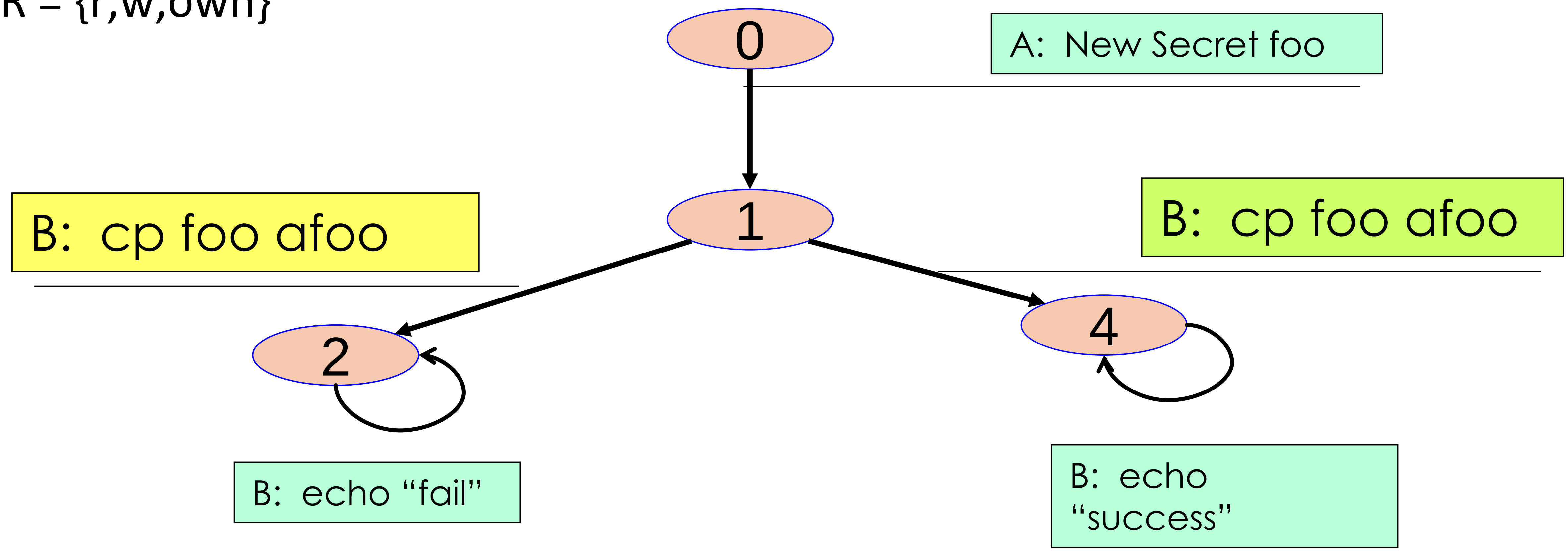
- Protection state of system
  - – Describes current settings, values of system relevant to protection

Confidentiality Scenario

Initial State

- Subjects S_0 = {A,B}
- Objects O_0 = {}
- AC Matrix A_0 = {}

Rights R = {r,w,own}



Confidentiality Scenario

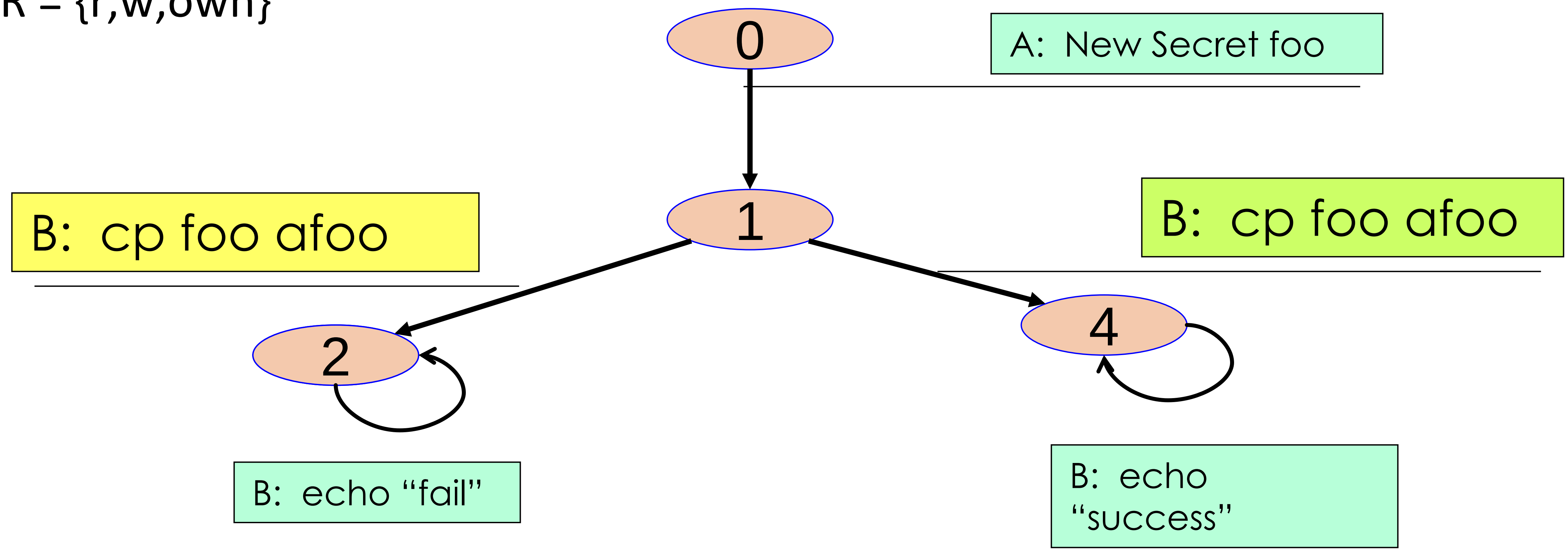
Initial State

Subjects S_0 = {A,B}

Objects O_0 = {}

AC Matrix A_0 = {}

Rights R = {r,w,own}



Intended State 1

Subjects S_1 = {A,B}

Objects O_1 = {foo}

AC Matrix A_1 = { (A,foo,[r,w,own]), (B,foo,[])}

Confidentiality Scenario

Initial State

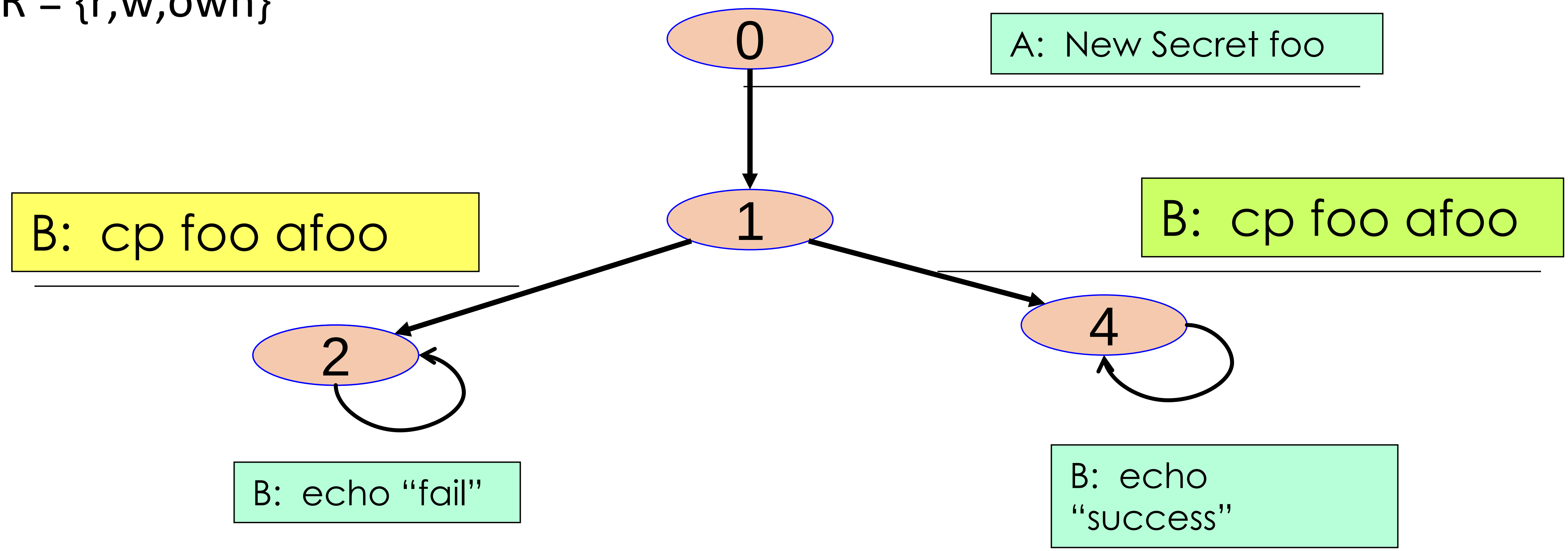
Subjects $S_0 = \{A,B\}$

Objects $O_0 = \{\}$

AC Matrix $A_0 = \{\}$

Rights R = {r,w,own}

(S_0, O_0, A_0) A: New Secret foo (S_1, O_1, A_1)



Intended State 1

Subjects $S_1 = \{A,B\}$

Objects $O_1 = \{foo\}$

AC Matrix $A_1 = \{ (A,foo,[r,w,own]),$
$(B,foo,[])\}$

Confidentiality Scenario

Initial State

Subjects $S_0 = \{A,B\}$

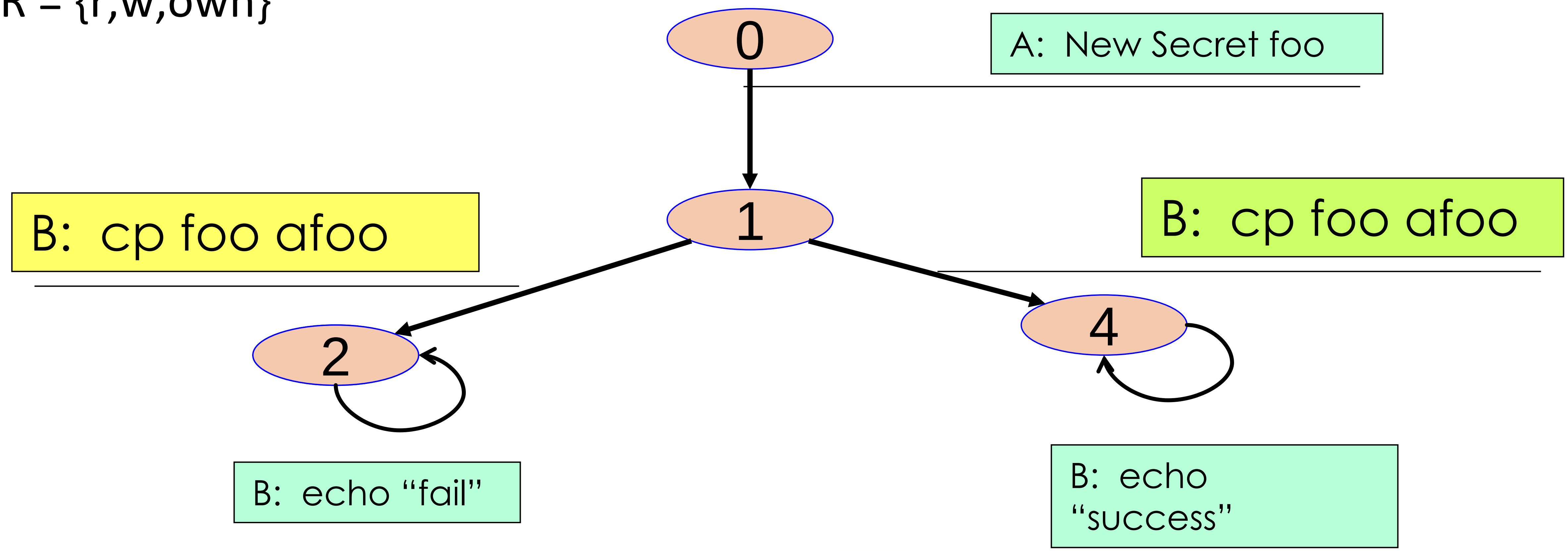
Objects $O_0 = \{\}$

AC Matrix $A_0 = \{\}$

Rights $R = \{r,w,own\}$

(S_0, O_0, A_0) A: New Secret foo (S_1, O_1, A_1)

(S_1, O_1, A_1) **B: cp foo afoo**



Intended State 1

Subjects $S_1 = \{A,B\}$

Objects $O_1 = \{foo\}$

AC Matrix $A_1 = \{ (A,foo,[r,w,own]), (B,foo,[])\}$

Confidentiality Scenario

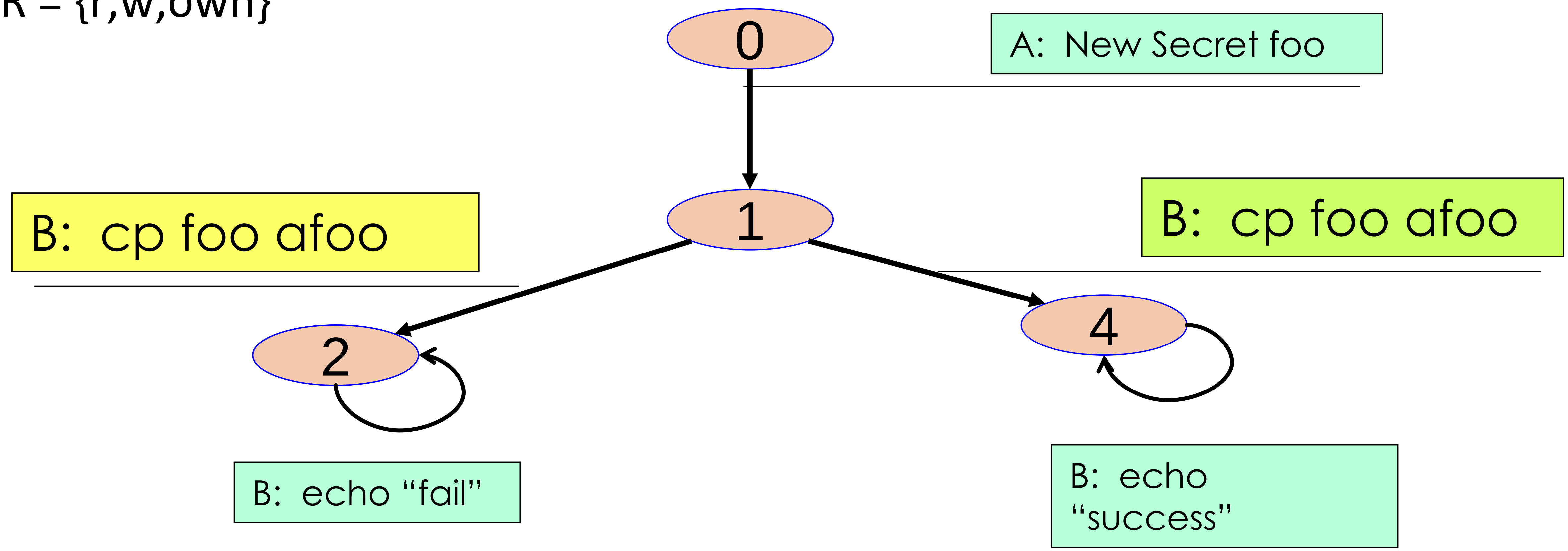
Initial State

- Subjects $S_0 = \{A,B\}$
- Objects $O_0 = \{\}$
- AC Matrix $A_0 = \{\}$

Rights $R = \{r,w,own\}$

(S_0, O_0, A_0) A: New Secret foo (S_1, O_1, A_1)

(S_1, O_1, A_1) **B: cp foo afoo** $(S_?, O_?, A_?)$



Intended State 1

- Subjects $S_1 = \{A,B\}$
- Objects $O_1 = \{foo\}$
- AC Matrix $A_1 = \{ (A,foo,[r,w,own]), (B,foo,[])\}$

Confidentiality Scenario

Initial State

Subjects $S_0 = \{A,B\}$

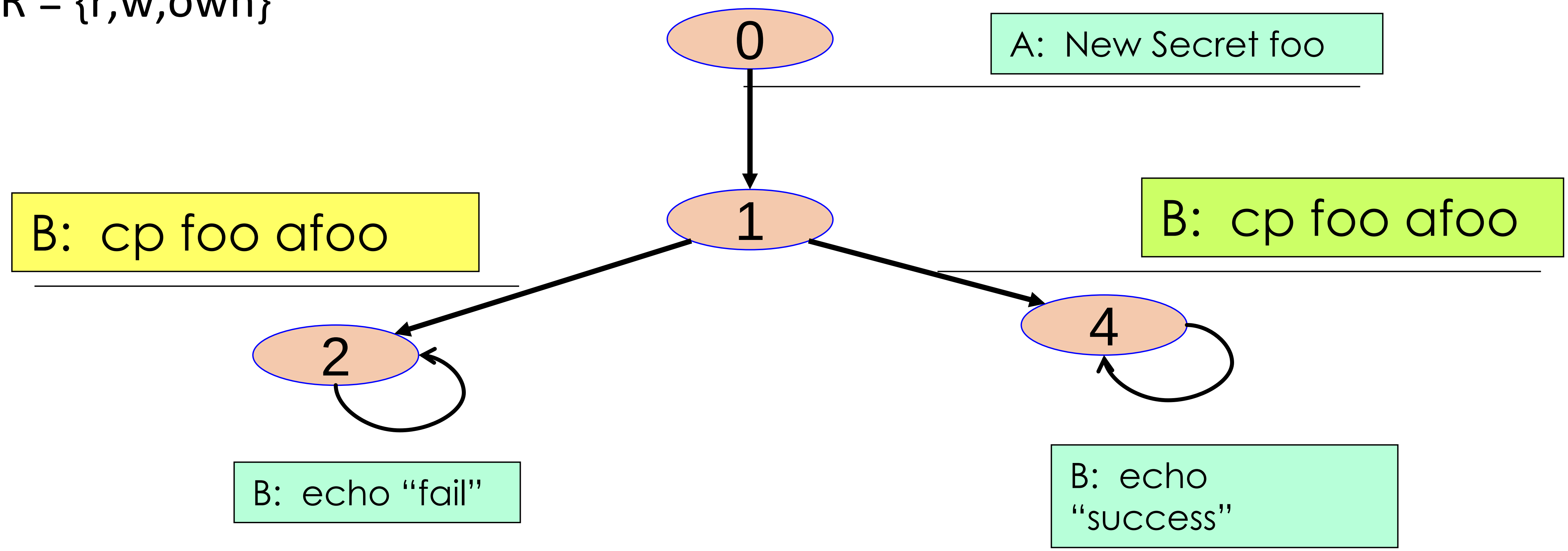
Objects $O_0 = \{\}$

AC Matrix $A_0 = \{\}$

Rights R = {r,w,own}

(S_0, O_0, A_0) A: New Secret foo (S_1, O_1, A_1)

(S_1, O_1, A_1) B: cp foo afoo $(S_?, O_?, A_?)$



Intended State 1

Subjects $S_1 = \{A,B\}$

Objects $O_1 = \{foo\}$

AC Matrix $A_1 = \{ (A,foo,[r,w,own]), (B,foo,[])\}$

Critical issue is definition of cp … …

Agenda

- Access Control Matrix
  - Overview
  - Access Control Matrix Model
  - Protection State Transitions
    - Commands
    - Conditional Commands
- Foundational Results

NEXT

Access Control Matrix Model

- Lampson '71 (http://www.computerhistory.org/fellowawards/hall/bios/Butler,Lampson/), refined by Graham and Denning ('71, '72)

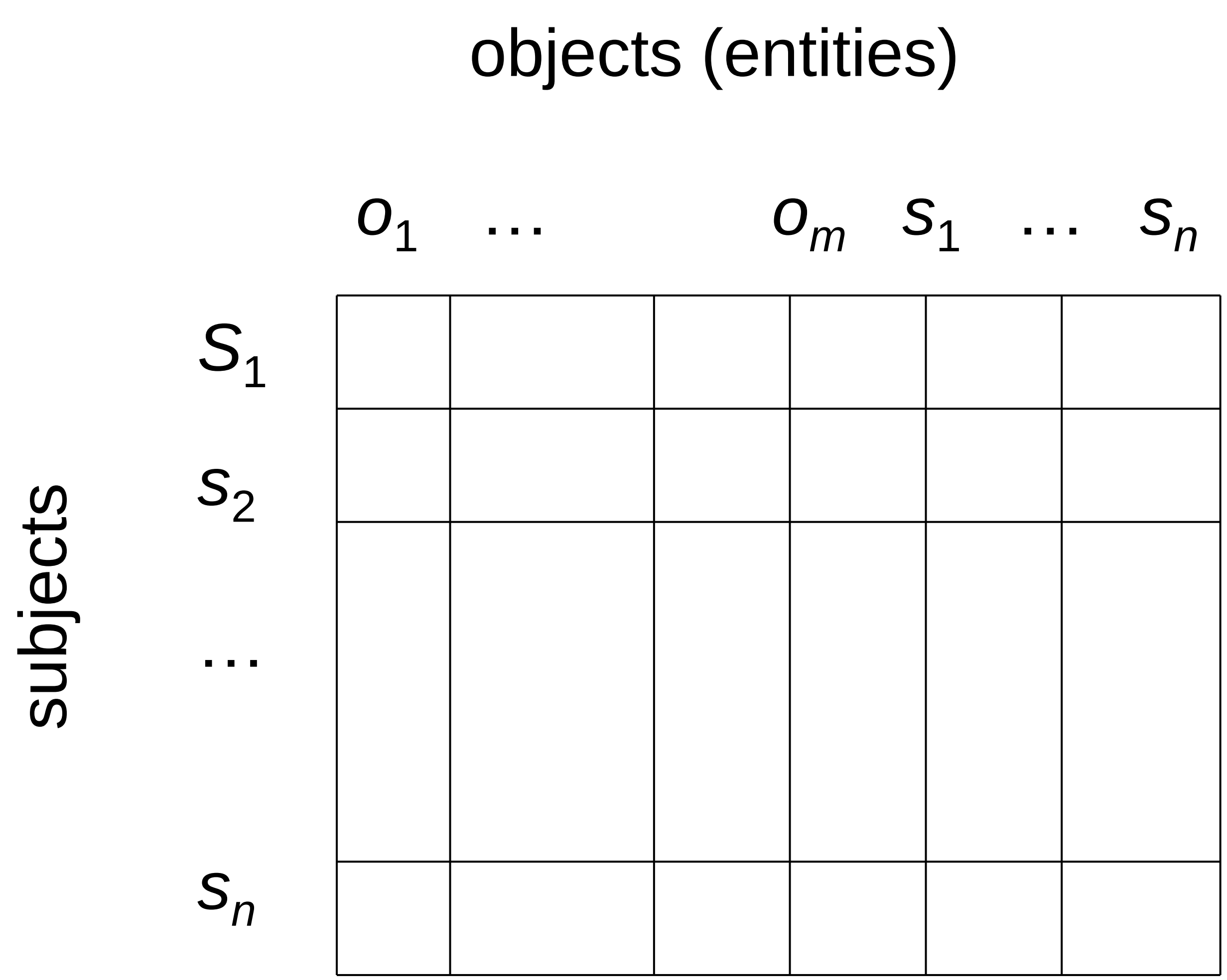
- Key Concepts
  - **Objects**, the protected entities, O
  - **Subjects**, the active entities acting on the objects, S
  - **Rights**, the controlled operations subjects can perform on objects, R

  - **Access Control Matrix**, A, maps Objects and Subjects to sets of Rights
  - State: (S, O, A)

Access Control Matrix Model

- Access control matrix
  - Describes protection state *precisely*
  - Matrix describing rights of subjects
  - State transitions change elements of matrix

Description



- Subjects $S = \{ s_1,\ldots,s_n \}$
- Objects $O = \{ o_1,\ldots,o_m \}$
- Rights $R = \{ r_1,\ldots,r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject s_i has rights $r_x, \ldots, r_y$ over object o_j

Example

- Processes *p*, *q*
- Files *f*, *g*
- Rights *r*, *w*, *x*, *a(ppend)*, *o(wn)*

	f	*g*	*p*	*q*
p	*rwo*	*r*	*rwxo*	*w*
q	*a*	*ro*	*r*	*rwxo*

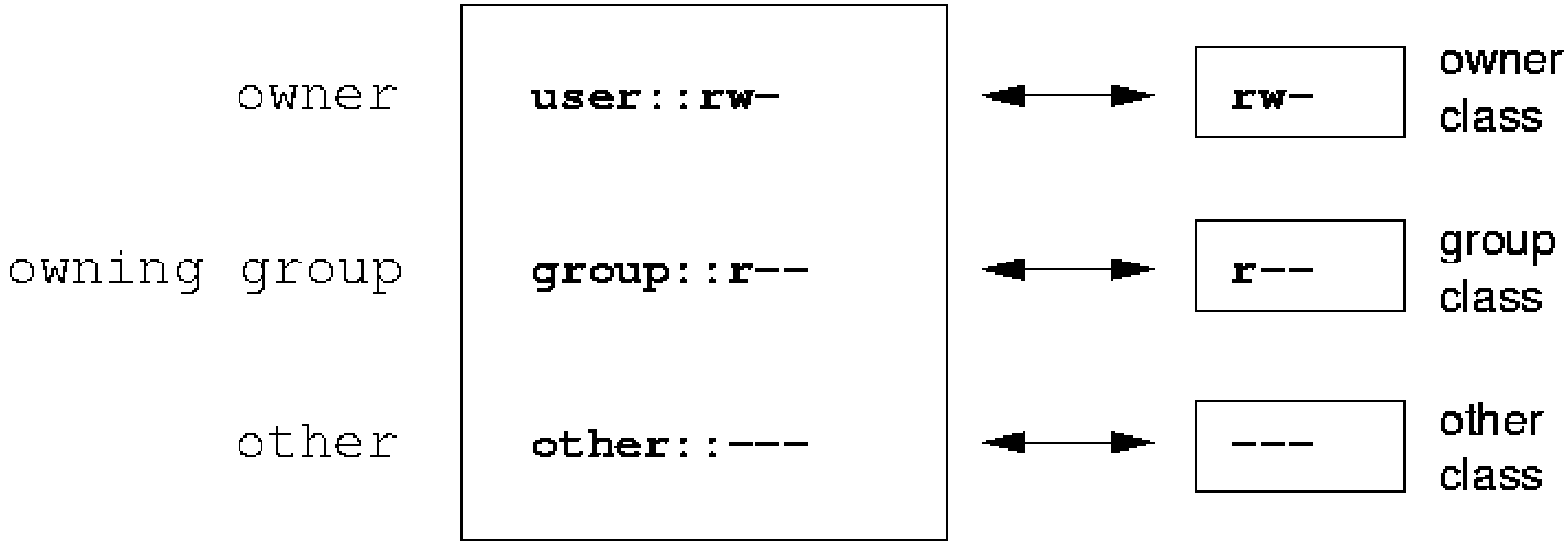
Example 2

- Procedures *inc_ctr*, *dec_ctr*, *manage*
- Variable *counter*
- Rights *+*, *–*, *call*

	counter	*inc_ctr*	*dec_ctr*	*manage*
inc_ctr	+			
dec_ctr	–			
manage		*call*	*call*	*call*

Example in Linux

```
user@user-desktop:~/cs165$ ls -l
total 32
-rw-r----- 1 user group 26498 Feb 17 15:19 code.tar.gz
```



Example in Linux

- [root@target /]# ls -l /home/

  drwxrwxr-x 2 team-temp admin 36864 … admin

Agenda

- Access Control Matrix
  - Overview
  - Access Control Matrix Model
  - Protection State Transitions

    NEXT
    - Commands
    - Conditional Commands
- Foundational Results

Protection State Transitions

- Sequences of state transitions are represented by commands that update the access control matrix (ACM)
- Primitive commands
  - **create subject** *s*; **create object** *o*
    - Creates new row, column in ACM; creates new column in ACM
  - **destroy subject** *s*; **destroy object** *o*
    - Deletes row, column from ACM; deletes column from ACM
  - **enter** *r* **into** *A***[***s*, *o***]**
    - Adds *r* rights for subject *s* over object *o*
  - **delete** *r* **from** *A***[***s*, *o***]**
    - Removes *r* rights from subject *s* over object *o*

Creating File

- Process *p* creates file *f* with *r* and *w* permission

```
command create•file(p, f)
    create object f;
    enter own into A[p, f];
    enter r into A[p, f];
    enter w into A[p, f];
end
```

Mono-Operational Commands

- Make process *p* the owner of file *g*

```
command make•owner(p, g)
 enter own into A[p, g];
end
```

- Mono-operational command
  – Single primitive operation in this command

Conditional Commands

- Let *p* give *q r* rights over *f*, if *p* owns *f*

```
command grant•read•file•1(p, f, q)
 if own in A[p, f]
 then
    enter r into A[q, f];
end
```

- Mono-conditional command
  – Single condition in this command

Multiple Conditions

- Let p give q (r and w) rights over f, if p owns f and p has c rights over q

```
command grant•read•file•2(p, f, q)
 if own in A[p, f] and c in A[p, q]
 then
    enter r into A[q, f];
    enter w into A[q, f];
end
```

Special Rights

- The copy right (or grant right) allows the possessor to grant rights to another
- The own right enables the possessors to add or delete privileges for themselves and others

- Principle of attenuation of Privilege
  - A subject may not give rights it does not possess to another

Short Summary: Key Points

- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations alter matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions

Outline

- Access Control Matrix
  - Overview
  - Access Control Matrix Model
  - Protection State Transitions
    - Commands
    - Conditional Commands
- Foundational Results

NEXT

Safety

- Let R be the set of generic (primitive) rights of the system

Safety

- Let *R* be the set of generic (primitive) rights of the system
- Definition: when a generic right *r* is added to an element of the access control matrix not already containing *r*, that right *r* is said to be *leaked*

Safety

- Let R be the set of generic (primitive) rights of the system
- Definition: when a generic right r is added to an element of the access control matrix not already containing r, that right r is said to be *leaked*

- Definition: If a system can never leak right r, the system is called safe with respect to the right r. If the system can leak right r, the system is called unsafe with respect with the right r

Example: project 3

- On cs165-internal,
- Unprivileged users: team0, team1, …
- Privileged user: admin
- File: /home/admin/

	/home/admin/
team0	read
team1	read
…	read
admin	read/write

Outline

- Access Control Matrix
- Other Security Policy Models

NEXT

Other Types of Security Policies

- Military (governmental) security policy
  - Policy primarily protecting **confidentiality**
- Commercial security policy
  - Policy primarily protecting **integrity**

- **Confidentiality** policy
- **Integrity** policy

- More abstract and conceptual

Confidentiality Policy

- Goal: prevent the unauthorized disclosure of information
  - Deals with information flow
  - Extensive redundancy in military makes integrity/availability less of a problem

Confidentiality Policy

- Goal: prevent the unauthorized disclosure of information
  - Deals with information flow
  - Extensive redundancy in military makes integrity/availability less of a problem
- Multi-level security (MLS) models are best-known examples
  - **Bell-LaPadula Model (BLP model), 1970s**

Bell-LaPadula Model

- Security levels arranged in linear ordering

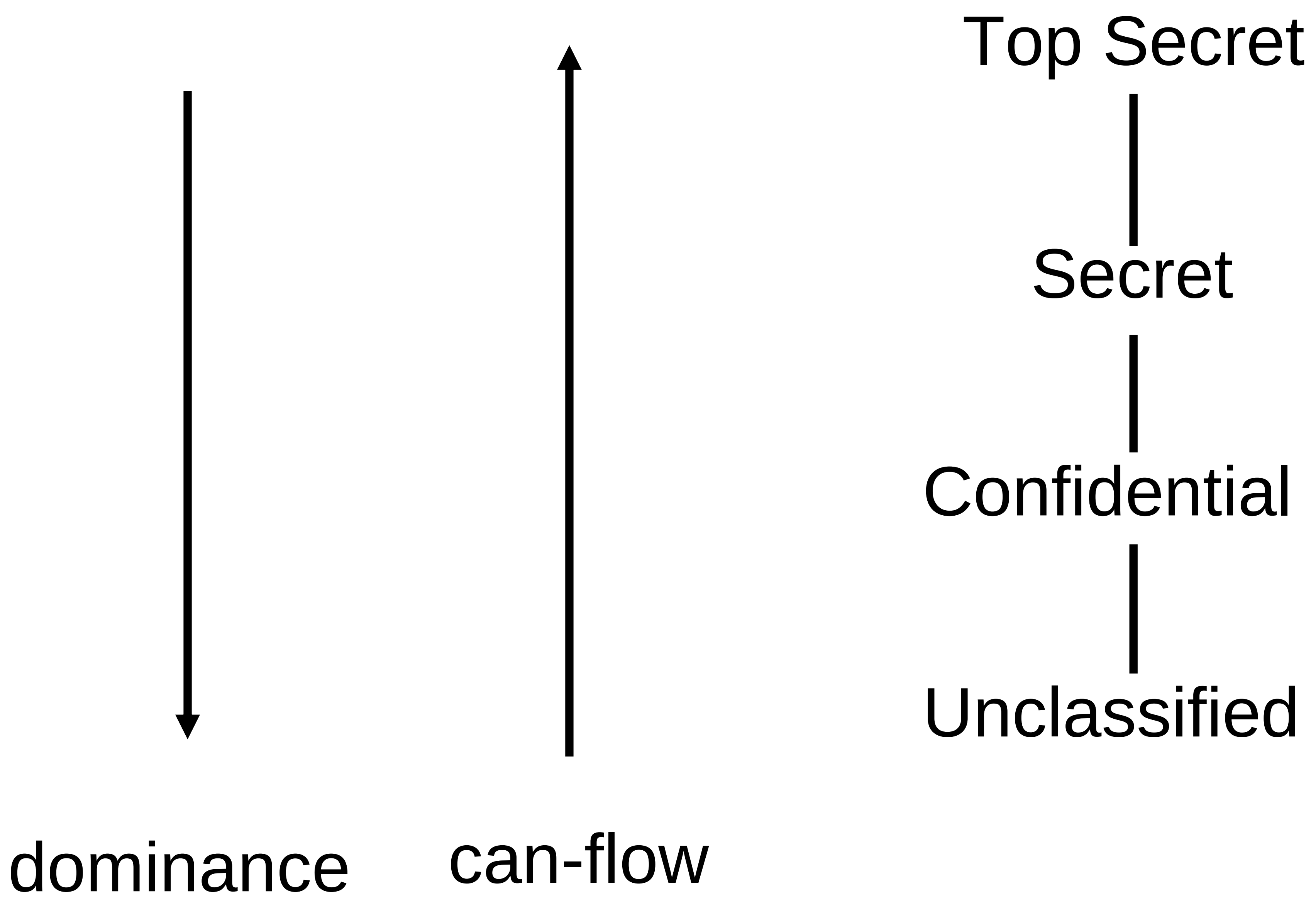
- Subjects have *security clearance* $L(s)$
- Objects have *security classification* $L(o)$

Summary of BLP Model

Policy (Write/Create): Information flows up, not down

"Write up" allowed, "write down" disallowed

"Read down" allowed, "read up" disallowed



Example

security level	***subject***	***object***
Top Secret	Terry	Personnel Files
Secret	Sam	E-Mail Files
Confidential	Charlie	Activity Logs
Unclassified	Umed	Telephone Lists

- Terry can read all files
- Charlie cannot read Personnel or E-Mail Files
- Umed can only read Telephone Lists

Lattice Model

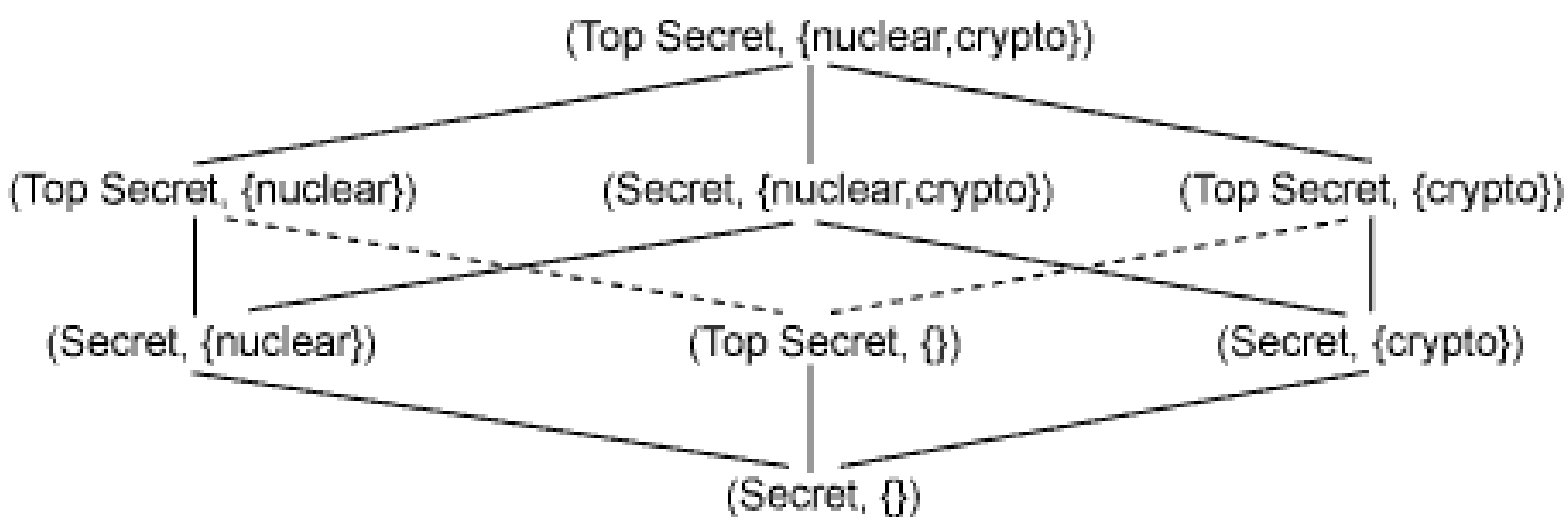
- Used by the US military (and many others), the Lattice model uses MLS to define policy
- Expand notion of security level to include categories
  - Categories (actually unbounded set)
    - NUC(lear), INTEL(igence), CRYPTO(graphy)
    - Note that these levels are used for physical documents in the governments as well.
- Security level is (*clearance*, *category set*), or formally *(L,C)* where *L* is the clearance level, and *C* is the set of categories

Lattice Model – Label-based Access Control

- Examples
  - Alice: (TOP SECRET, {CRYTPO})
  - Bob: (SECRET, {NUC})
  - Charlie: (TOP SECRET, {CRYPTO, NUC})
  - DocA: (SECRET, {CRYPTO, NUC})
  - DocB: (SECRET, {CRYPTO})

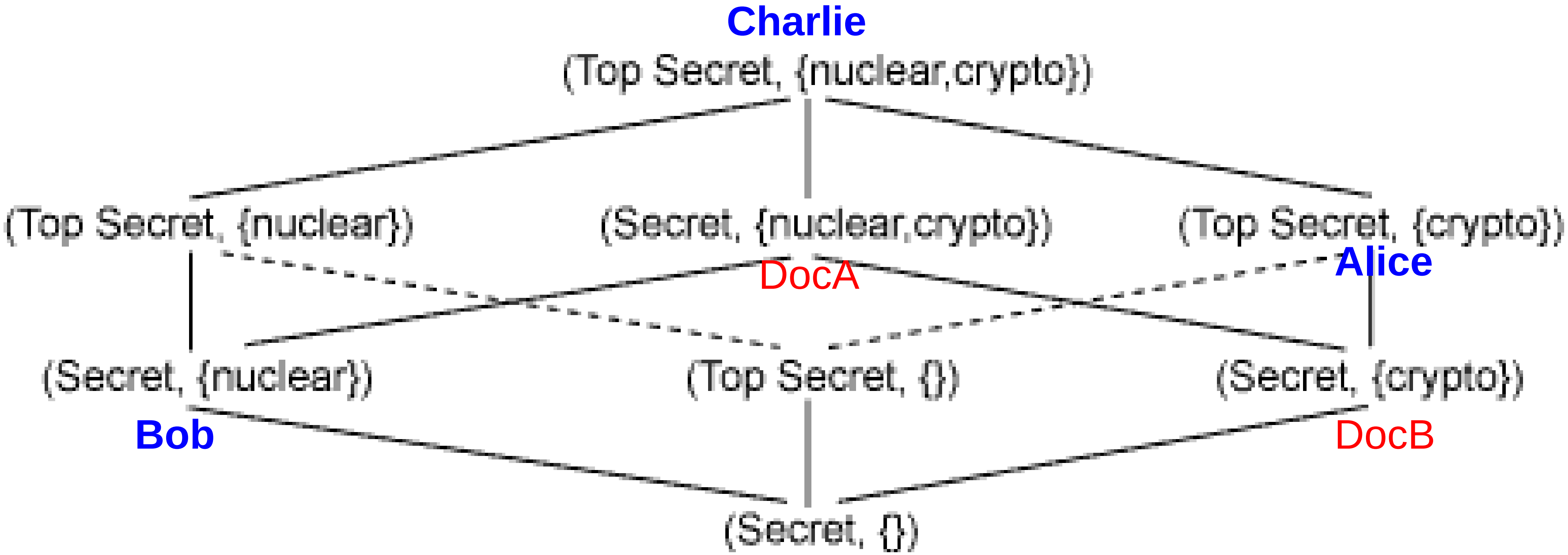
Lattice Model – Label-based Access Control

- Examples
  - Alice: (TOP SECRET, {CRYTPO})
  - Bob: (SECRET, {NUC})
  - Charlie: (TOP SECRET, {CRYPTO, NUC})
  - DocA: (SECRET, {CRYPTO, NUC})
  - DocB: (SECRET, {CRYPTO})



Lattice Model – Label-based Access Control

- Examples
  - Alice: (TOP SECRET, {CRYTPO})
  - Bob: (SECRET, {NUC})
  - Charlie: (TOP SECRET, {CRYPTO, NUC})
  - DocA: (SECRET, {CRYPTO, NUC})
  - DocB: (SECRET, {CRYPTO})

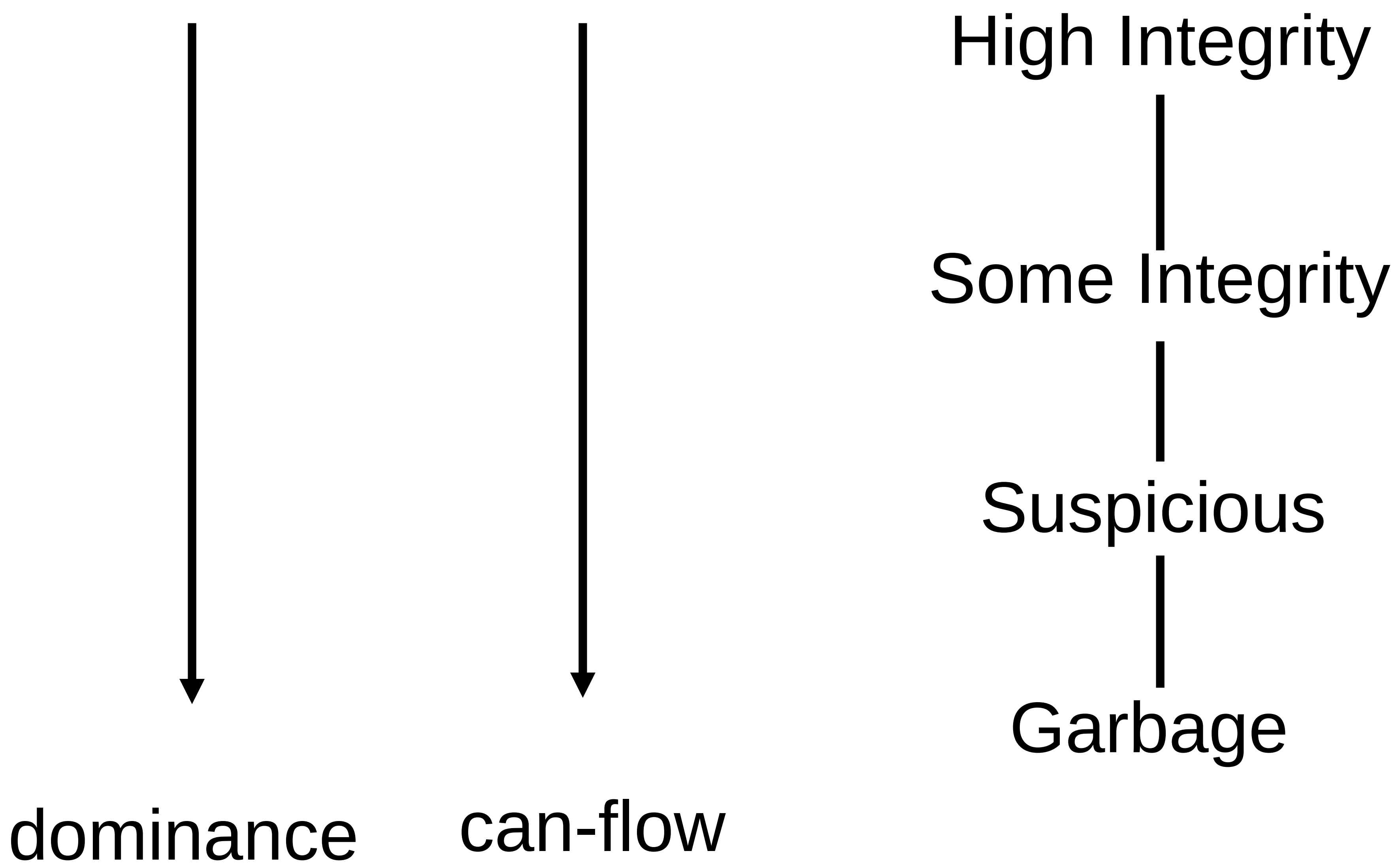


Integrity Policy

- MLS talks about who can “read” a document (*confidentiality*)
- *Integrity* is considered who can “write or make change to” a document
  - Thus, who can effect the integrity (content) of a document
  - Example: You may not care who can read DNS records, but you better care who changes them!

BIBA Model

- Biba defined a dual of secrecy for integrity
  - “no read down, no write up”



In Reality

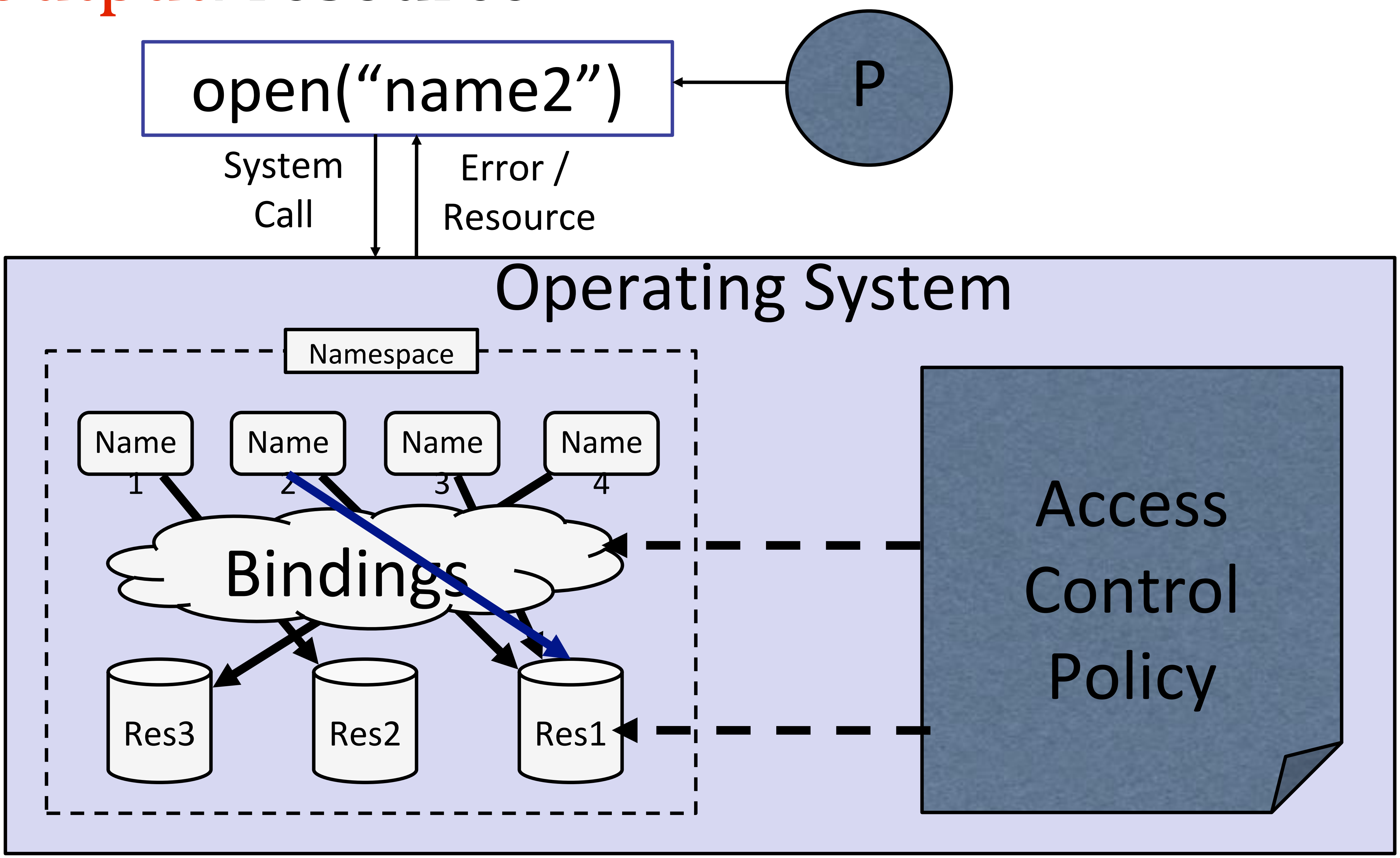
- High-integrity programs may need access to low-integrity data
  - Command line input
  - Web server accepting data from the Internet
  - Attacker-controlled files
    - SSH daemon runs as root and reads /home/[user] directory
- Or …
  - Sometimes programs may expect high-integrity objects mistakenly

Example Integrity Problem: Resource Access

- Programs require a variety of resources to function
  - *Files*
  - *Inter-process communication channels*
  - *Signals*
- How hard can accessing resources securely be?
  - Just a simple **`open(filename)`**, right?
  - Wrong!

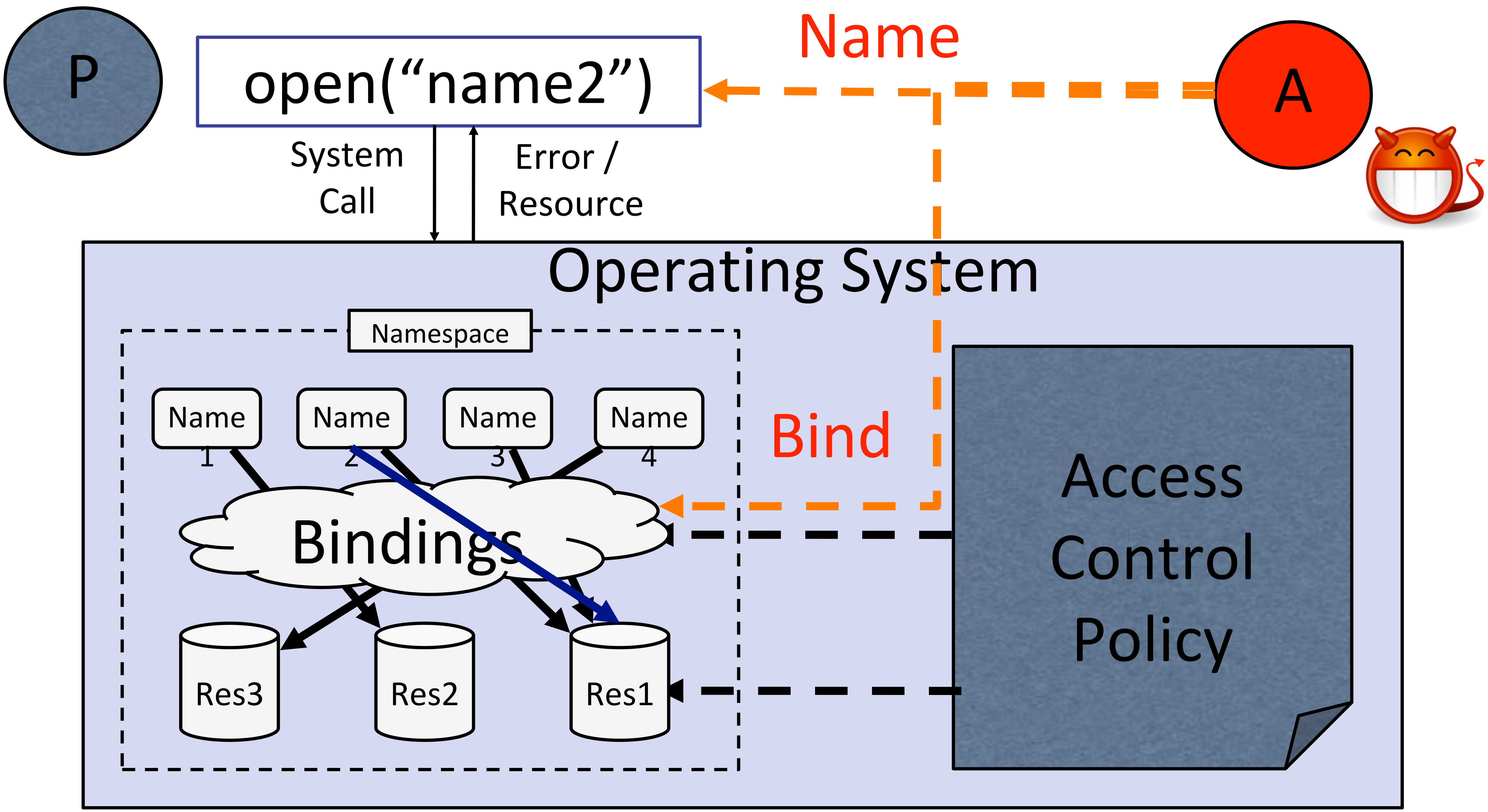
Resource Access

- Inputs: name, bindings (dirs, symbolic links)
- Output: resource



Resource Access Attacks

- Adversaries control inputs -- name, bindings to direct victim programs to malicious resources



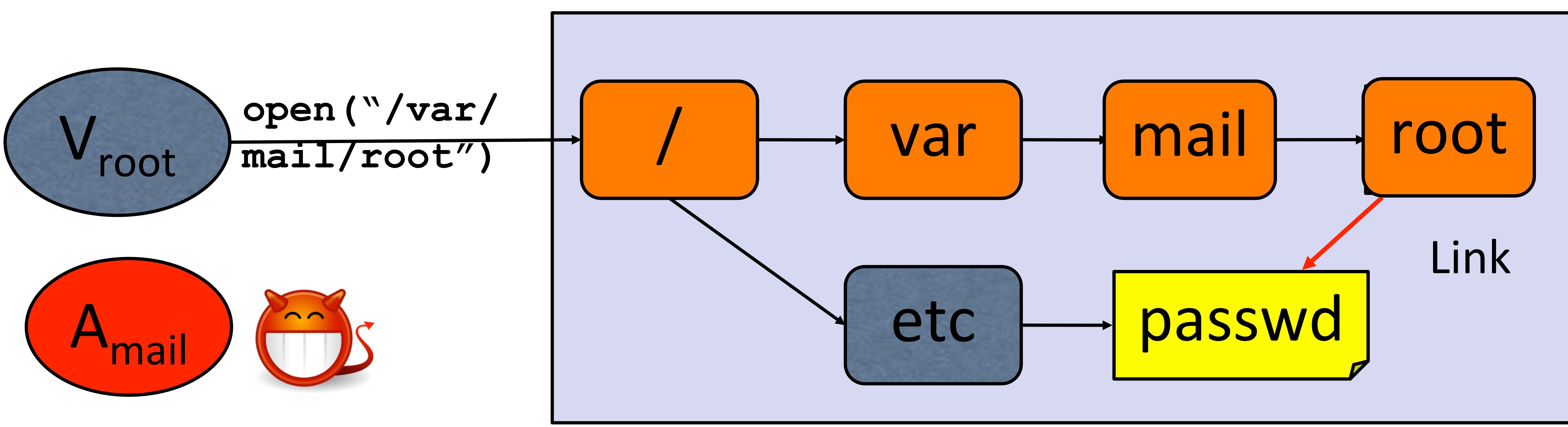
Adversary Control

- Who is an adversary?
  - Determined by threat model and access control policy
    - Based on access control policy
      - UID *x* has as adversary any UID $y \neq x$ (except superuser *root*)
- Adversary control of name or binding

Link Following

- Link following attack
  - Adversary controls bindings to redirect victim to a resource not under adversary's control (confused deputy)

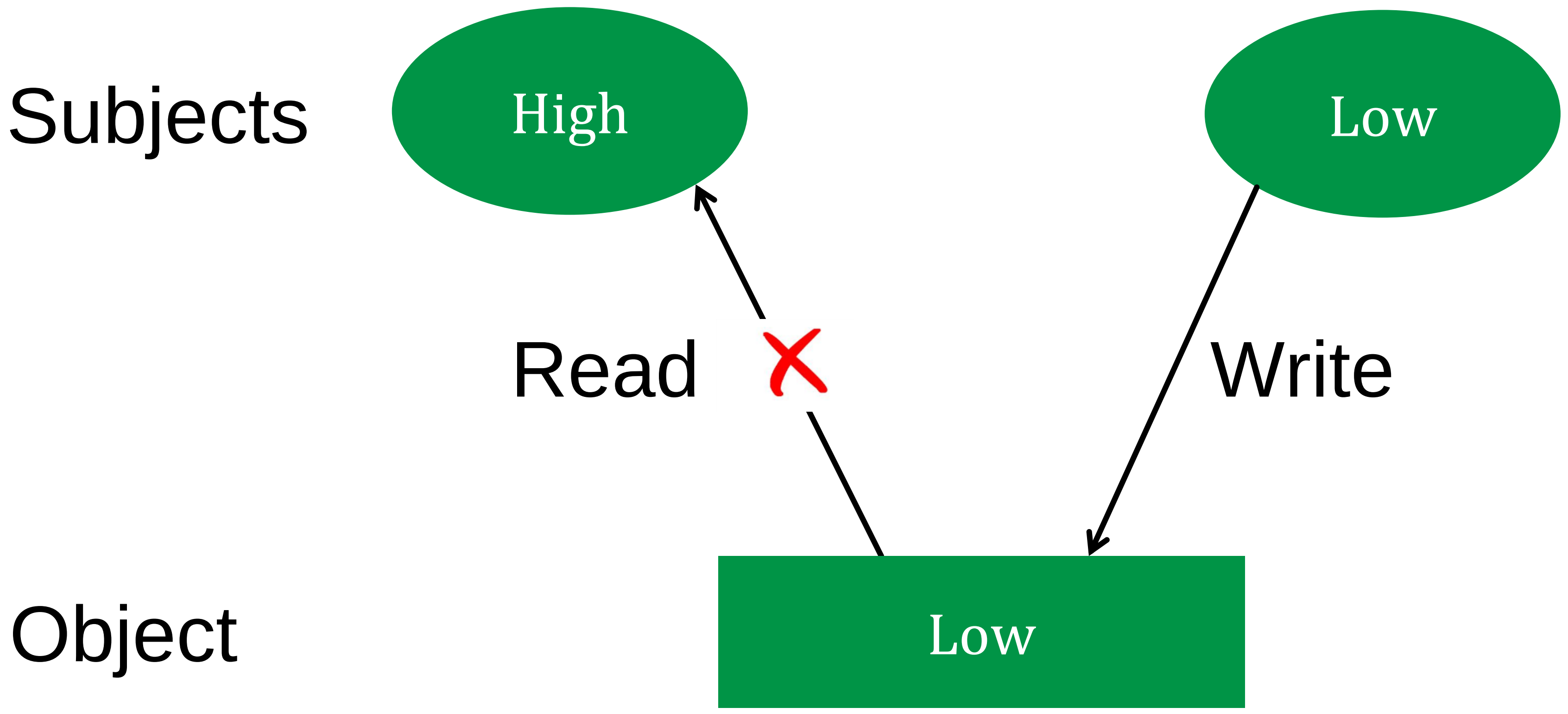
```
drwxrwsr-x  2 root mail  4096 2011-01-13 10:06 mail
```



Link Following

Integrity level	*Subject*	*Object*
High	root process	
Low	mail process	/var/mail/root

Victim expects high integrity, gets low integrity instead



Link Following

Access control matrix point-of-view: read access leaked

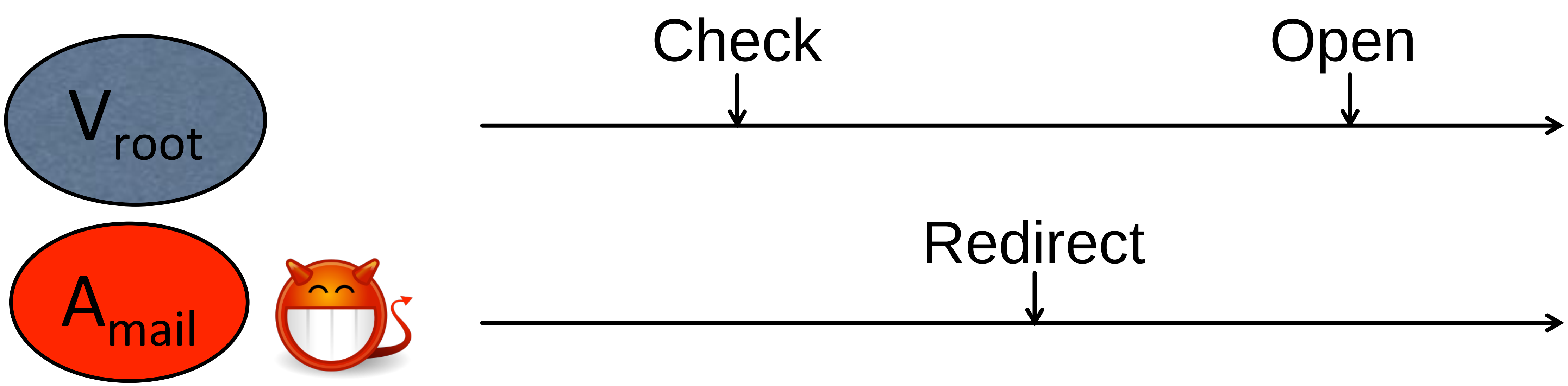
	/var/mail/root	/etc/passwd
root (high-privilege)	read/write	read/write
mail	read/write	no access -> read

Achieved through two commands:

1. rm /var/mail/root
2. ln -s /var/mail/root /etc/passwd

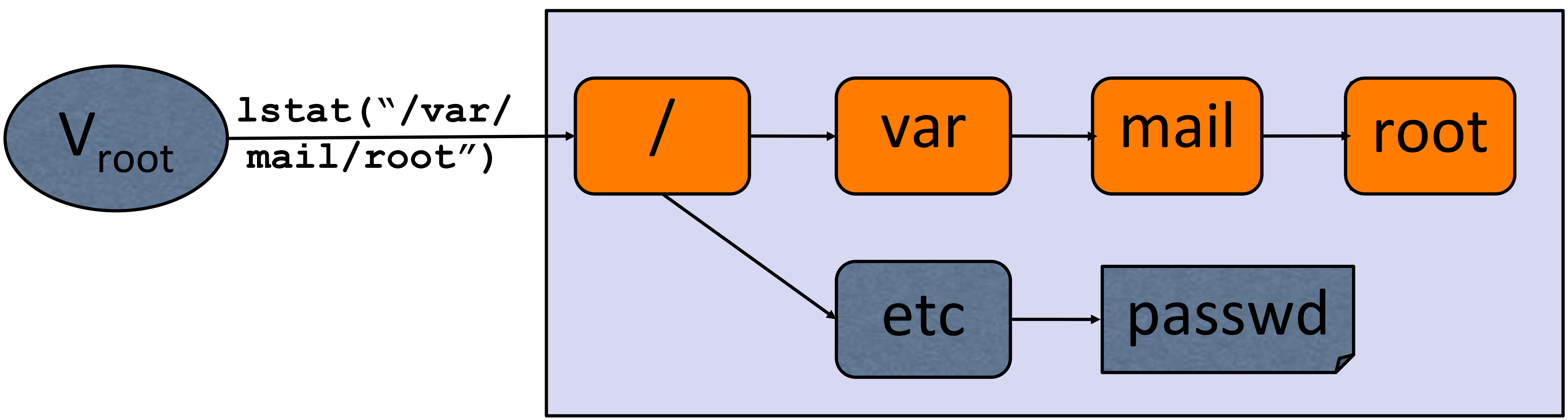
Link Following

- Can the root process protect itself by checking the low-integrity input?
  - Whether /var/mail/root is a *symbolic link*?
  - Unfortunately, it doesn't work



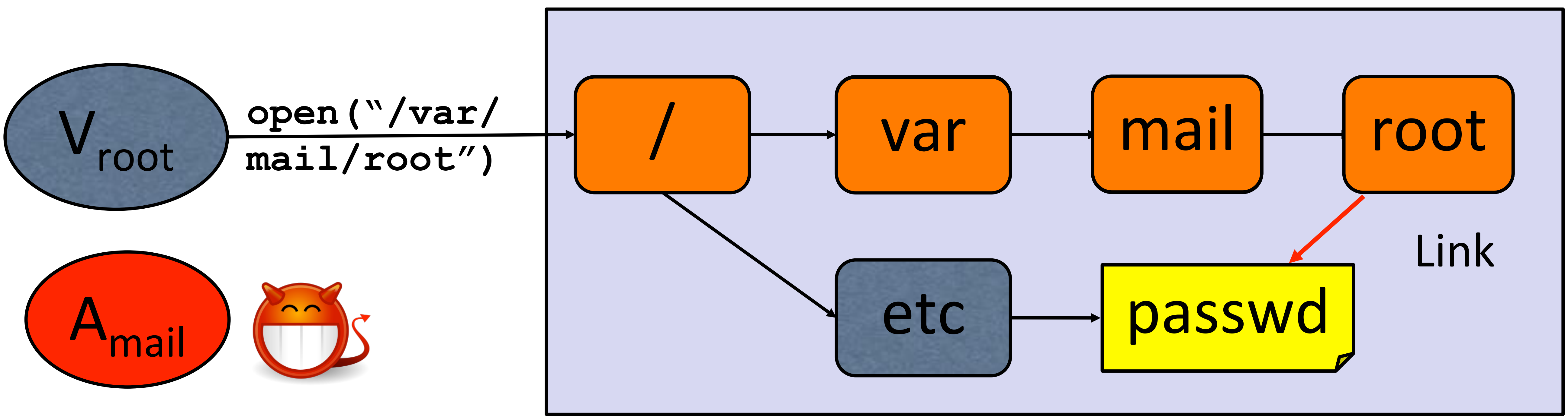
Race Conditions

- Race Conditions
  - Adversary exploits non-atomicity in “check” and “use” of resource
  - Time-of-check, time-of-use (TOCTTOU) attacks



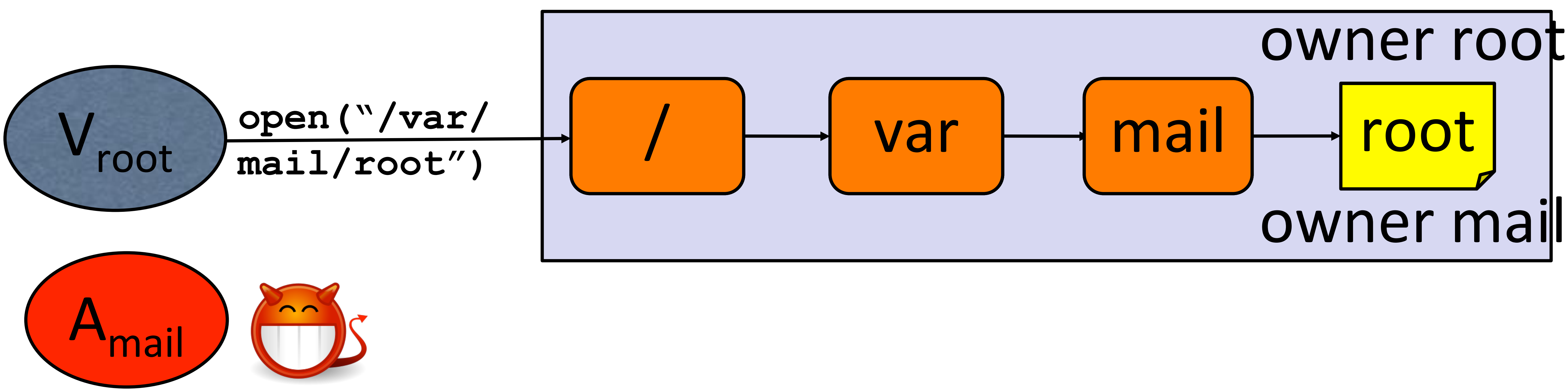
Race Conditions

- Race Conditions
  - Adversary exploits non-atomicity in “check” and “use” of resource
  - Time-of-check, time-of-use (TOCTTOU) attacks



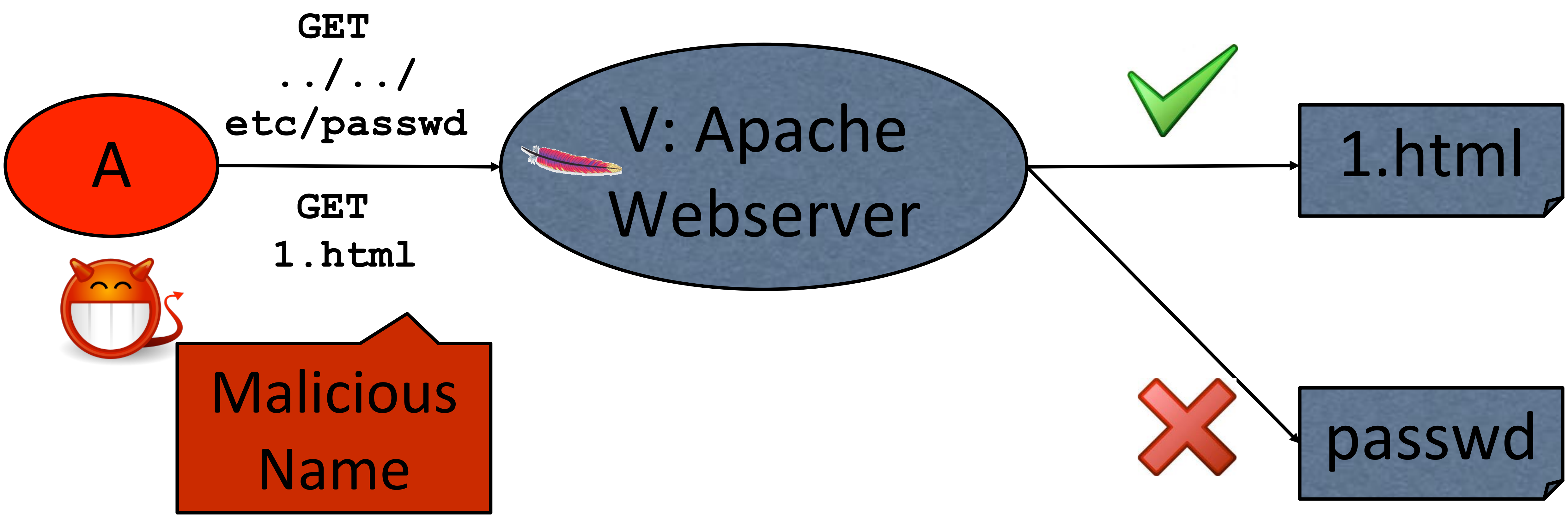
Another attack: File Squatting

- Adversary controls bindings to direct victim to an adversary accessible (low integrity) resource
  - Can trick the victim to read fake emails!
- Victim expects high integrity



Yet another attack: Directory Traversal

- Adversary controls the name (as opposed to binding) to direct victim to a high-secrecy resource
- Root cause?



Fundamental Problem

- When a program expects a high-integrity resource but gets low-integrity one
- Or when a program expects low-integrity resource but did not perform adequate checks

- They are all attack surface!

Questions