ENEE 457: Computer Systems Security

Lecture 15
Access Control and Information Flow

Charalampos (Babis) Papamanthou

Department of Electrical and Computer Engineering
University of Maryland, College Park

What is access control?

- After we authenticate into a system using, for example, a password, the system decides what resource we can access
- The mechanism and rules for the system to do that are encoded in access control information
- Example
  - After I log into my Google drive, I can only access files that other people have shared with me (access control rules)
  - Note that the action of logging in is not governed by access control, but by password authentication

Basic Access Control and Information Flow Models

- **Discretionary access control (DAC)**
  - Owner determines access rights
  - Typically it is an identity-based access control: access rights are assigned to users based on their identity
  - E.g., Google drive, File system in Windows/Linux
- **Mandatory access control (MAC)**
  - System enforce system-wide rules for access control
  - E.g., law allows a court to access driving records without the owners' permission

Access Control Matrix (ACM)

- S: subjects, users or processes
- O: objects, resources such as files, devices, messages, etc.
- A: access matrix A: $S \times O \rightarrow R$ (rights)
- Example:

O S	file 1	file 2	file 3
process 1	R W	R	R W E
process 2	R	R W	R

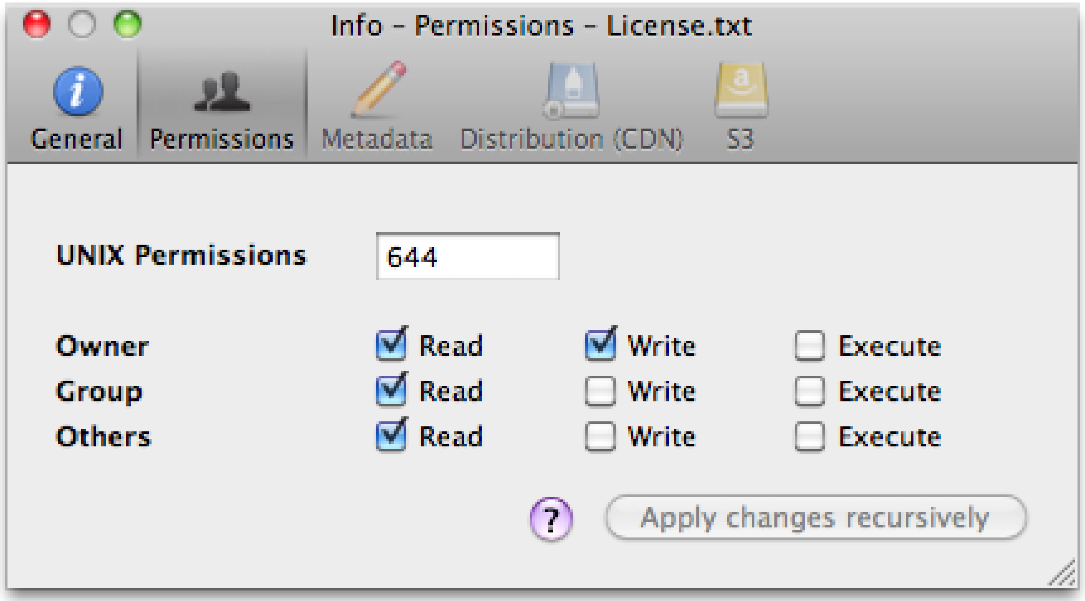
ACM

- DAC implementation
  - **Access Control Matrix (ACM):** Inefficient
  - **Access control list (ACLs):** describe the access policies for each object
  - **Capabilities:** describe the access rights each subject has

- Advantages and Disadvantages of ACLs
  - Easy to find which people can access an object
  - Not that easy to find which objects can be accessed by a specific person
- Advantages and Disadvantages of Capabilities
  - Easy to find which objects can be accessed by a specific person
  - Not that easy to find which persons can access a specific object

ACL in Unix

- In a real system
  - Too many subjects to deal with (think about users in a linux system)
- Unix
  - Classify subjects into: owner, group, world
  - Use ACL for each object, but in terms of owner, group, world
- Not the same system in Google drive



UNIX Permissions 644

Owner	☑ Read	☑ Write	□ Execute
Group	☑ Read	□ Write	□ Execute
Others	☑ Read	□ Write	□ Execute

Setting Special Permissions

suid	sgid	stb	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1
7			7			7			7		
Special			user			group			others		

Use the “chmod” command with octal mode:

```
chmod 7777 filename
```

Special Bits

- suid: If suid is set, then the program can run with id of the user that owns the program (and is not bound to run with the id of the user that executes the program)
- sgid: If sgid is set, then the program can run with group id of the owner’s group (and is not bound to run with the group id of the user that executes the program)
- sticky bit: If it is set, then no one with write and execute access can rename or delete a file or directory

Why do we need setUiD?

- Every user has a user id that is called uid.
- When user A executes program B, program B is using A’s uid
- However:
- Programs can change to use the effective user id euid
  - Effective user id euid is the uid of the program owner
  - e.g. the `passwd` program changes to use its effective uid (`root`) so that it can edit the `/etc/passwd` file
  - This special permission allows a user to access files and directories that are normally only available to the owner
  - SUID bit enables this functionality

Sample SETUID Scenario

- `/dev/lp` is owned by root with protection `rw-------`
  - This is used to access the printer
- `/bin/lp` is owned by root with `rwsr-xr-x` (with SETUID=1)
- User A issues a print process p
- Process p has the same UID as user A
- Process p executes `exec(“/bin/lp”,…)`
- But lp is a setuid program and now p can use root’s UID
- Consequently, `/dev/lp` can be accessed to print
- When `/bin/lp` terminates so does p
- User A never got the access to `/dev/lp`, just the process he executed

A simple program

- Say I (cpap) own the program

```
FILEWRITE(file,uid,data):   rwx--x--x
IF write_access(file,uid) = 0    //write_access checks real user_id
        exit;
ELSE
        open_for_write(file); //open_for_write checks for effective user_id
        write_data(file,data);
```

- This program can only write to Bob’s file if executed by Bob.
- Can it write to cpap’s file **private** if executed by Bob?
  - NO!! It is going to exit after the first access control check
- What if cpap decides to make it setuid?

Problem with setUiD: Race conditions

- Now, let’s see the setuid program

```
FILEWRITE(file,uid,data):   rws--x--x
IF write_access(file,uid) = 0
        exit;
ELSE
        open_for_write(file);
        write_data(file,data);
```

Attacker enters symbolic link
symlink(file,cpap/private)

- This program can be executed by Bob
- And it can write to cpap’s file **private** due to race condition
- CAREFUL with SETUID programs!!

DAC and MAC

- When is DAC insufficient?
    - When owner cannot be trusted for the discretion of the data and external protection of the data is necessary
    - E.g., DAC has the danger of right propagation
        - A can read X and write Y
        - B can read Y, but no access to X
        - A reads X, write the content of X to Y, B got access to X
- MAC
    - Non-discretionary
    - Labels are assigned to subjects and objects
    - Owner has no special privileges
    - E.g., Bell-Lapadula, lattices models, SELinux by NSA
    - In our example: track the flow of labels. E.g., When A writes to Y, Y is tainted with X, and therefore B will not be able to read Y since B should have no access to X.

Traditional Models for MAC

- Bell-LaPadula (BLP)
  - About confidentiality (it was developed to formalize the US Department of Defense multilevel security policy)
- Biba
  - About integrity with static/dynamic levels

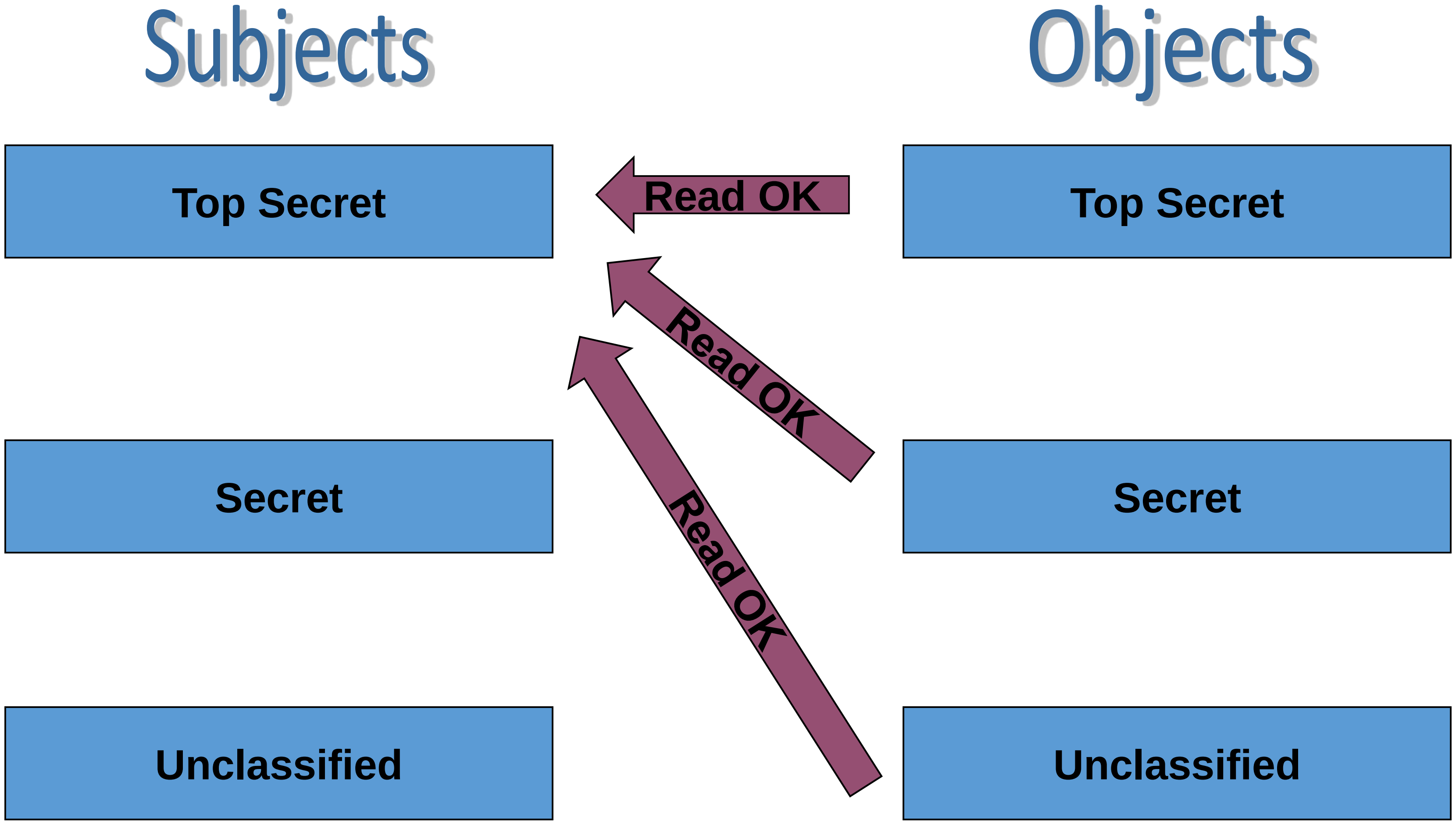
Bell – LaPadula - Details

- Each user subject and information object has a fixed security class – labels
- Use the notation ≤ to indicate dominance
- Simple Security (ss) property:

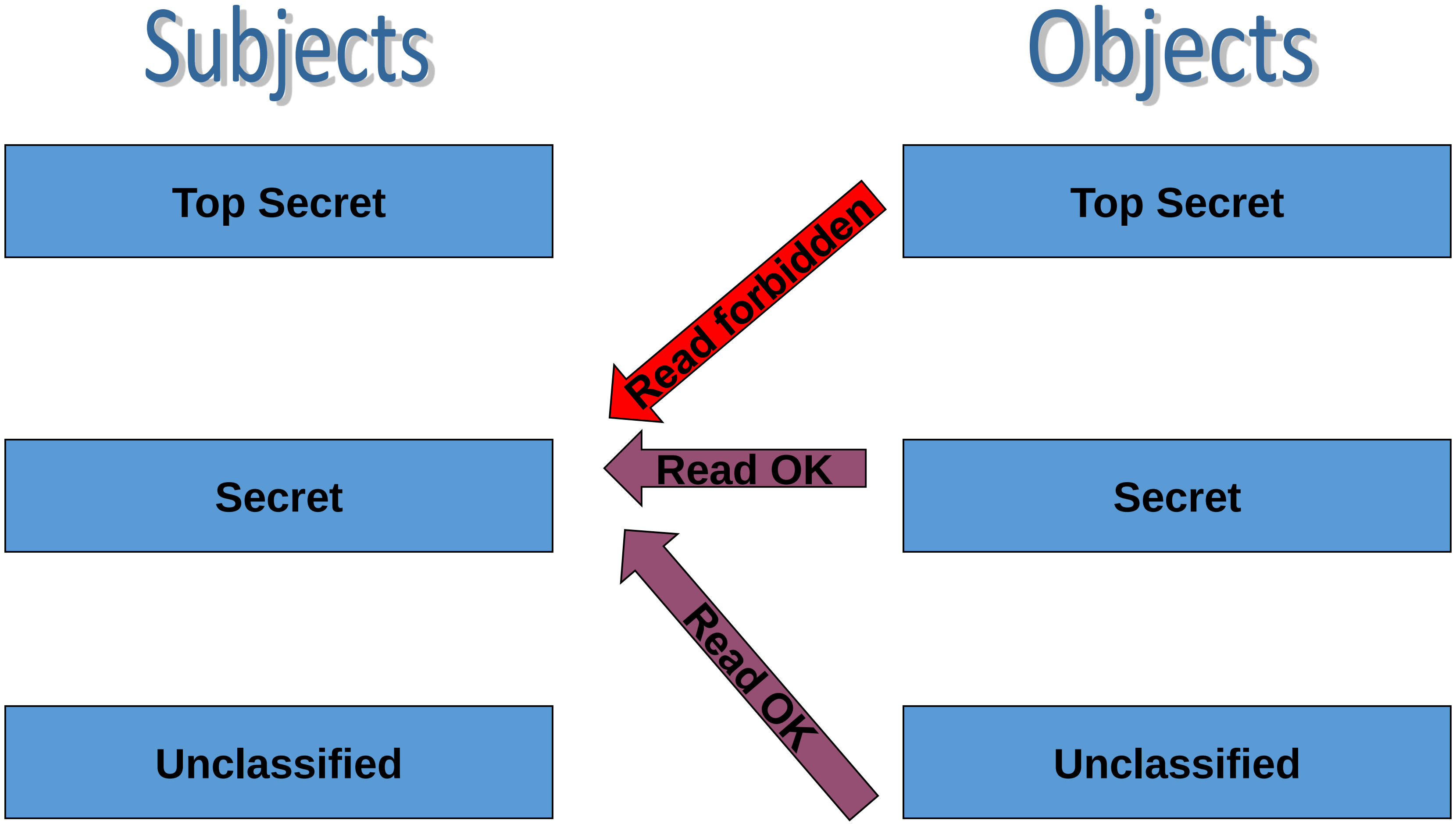
  the no read-up property

  - A subject s has read access to an object o iff the class of the subject C(s) is greater than or equal to the class of the object C(o)
  - i.e. Subjects s can read Objects o iff $C(o) \leq C(s)$

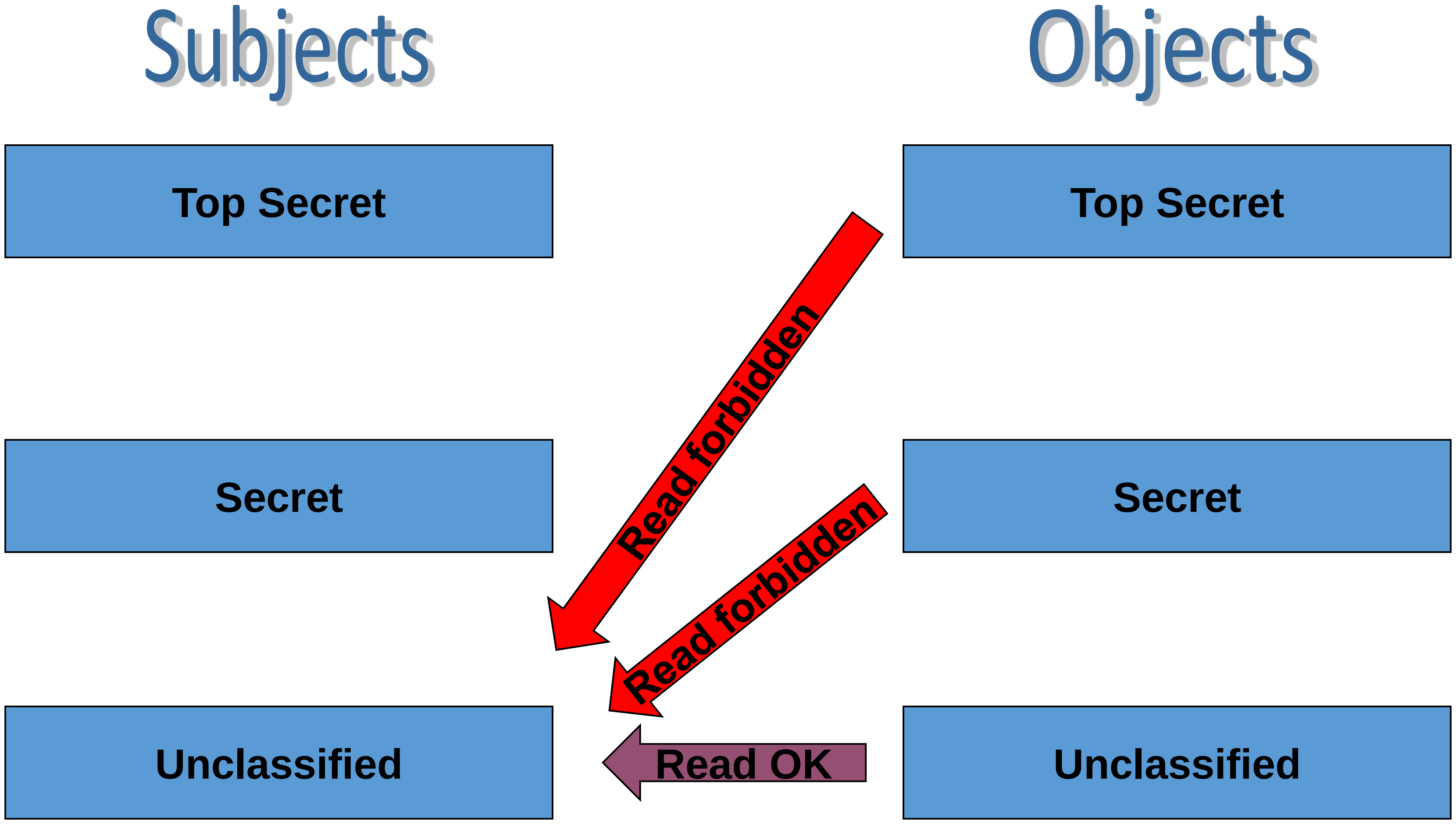
Access Control: Bell-LaPadula



Access Control: Bell-LaPadula



Access Control: Bell-LaPadula



Subjects

Objects

Top Secret

Top Secret

Read forbidden

Secret

Secret

Read forbidden

Unclassified

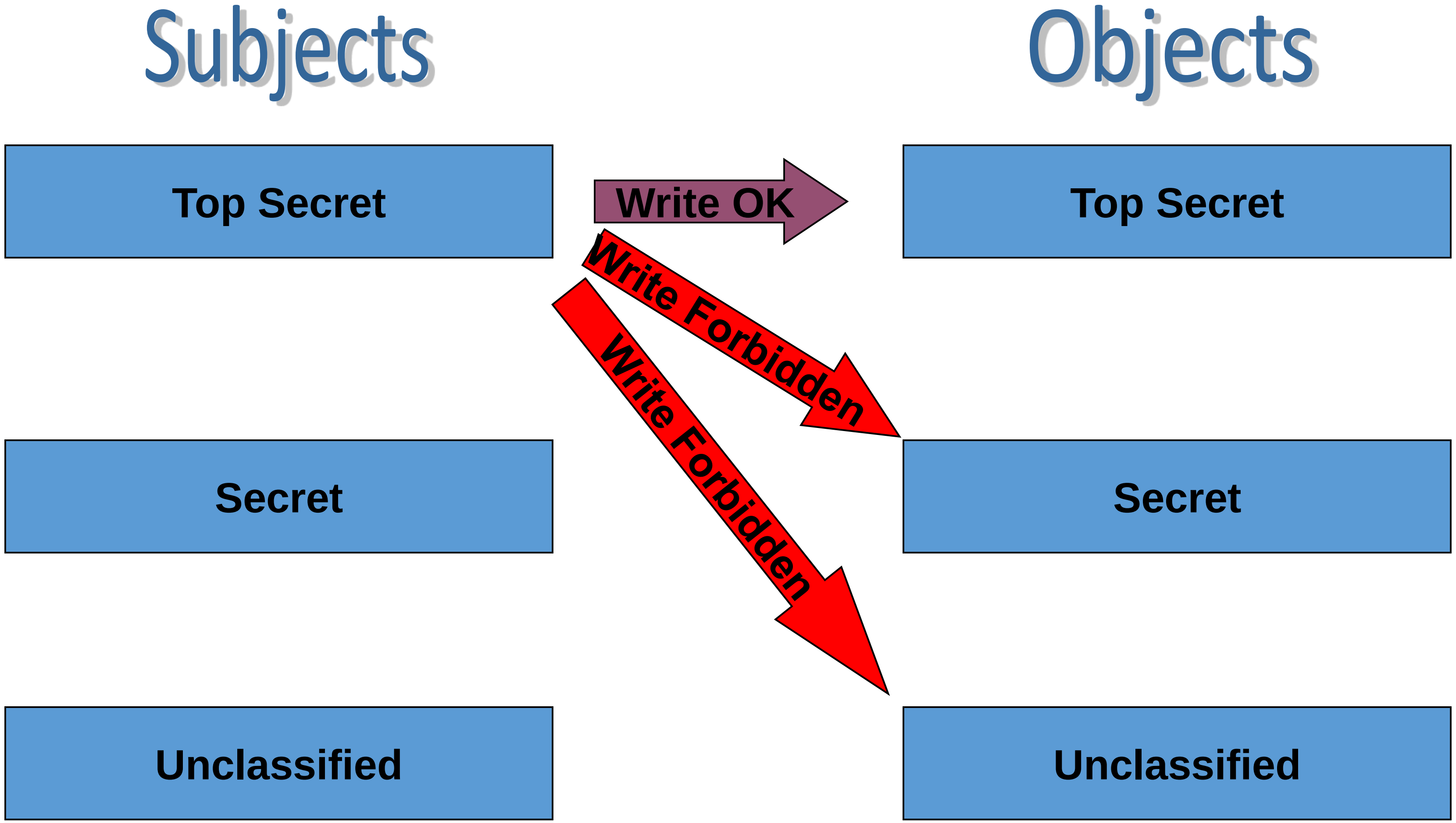
Read OK

Unclassified

Bell - LaPadula (2)

- * property (star):
  the no write-down property
  - A subject s can write to object *p* if C(s) ≤ C (p)

Access Control: Bell-LaPadula



Subjects

Objects

Top Secret

Write OK

Top Secret

Write Forbidden

Write Forbidden

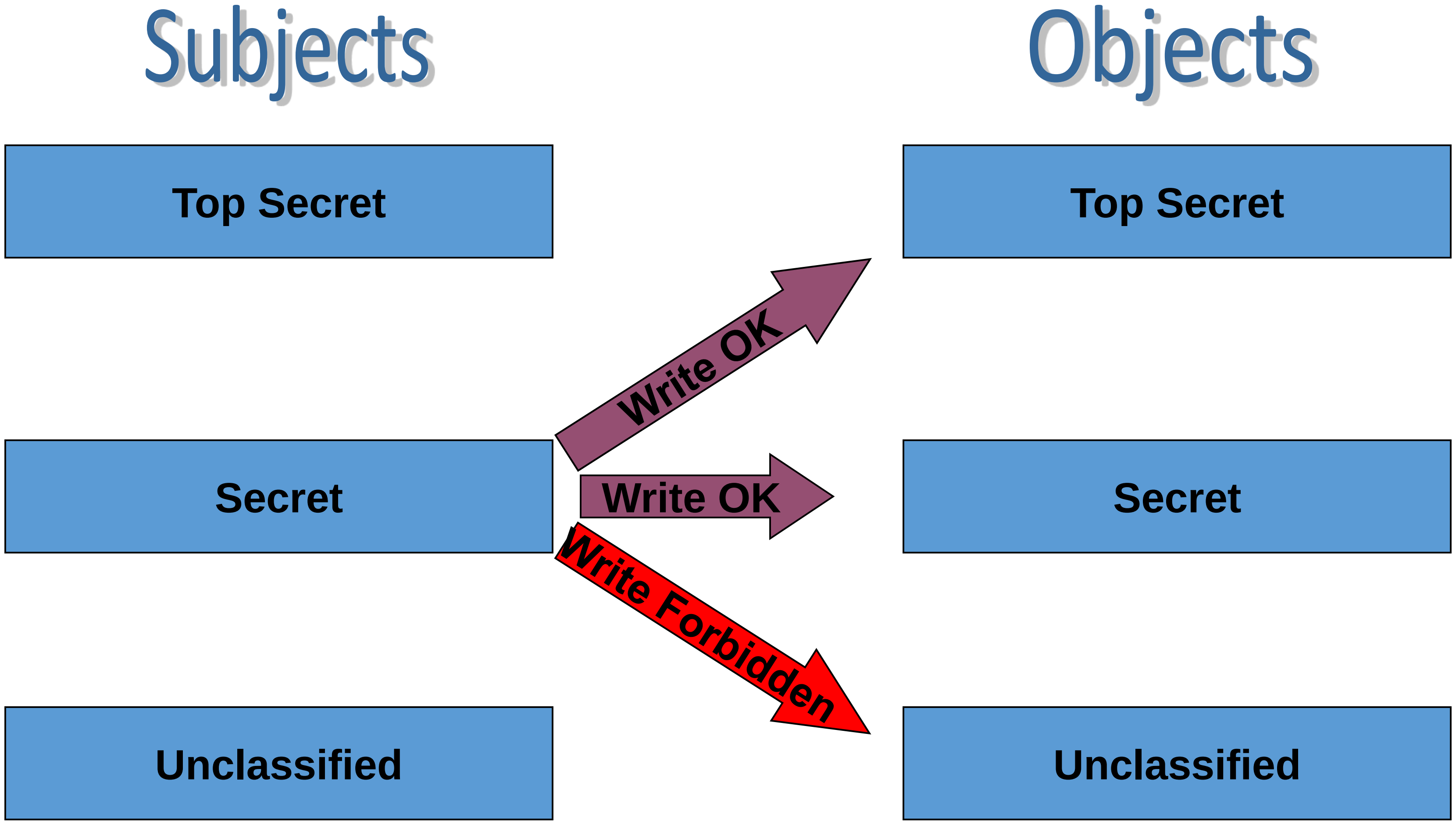
Secret

Secret

Unclassified

Unclassified

Access Control: Bell-LaPadula



Subjects

Objects

Top Secret

Top Secret

Secret

Secret

Unclassified

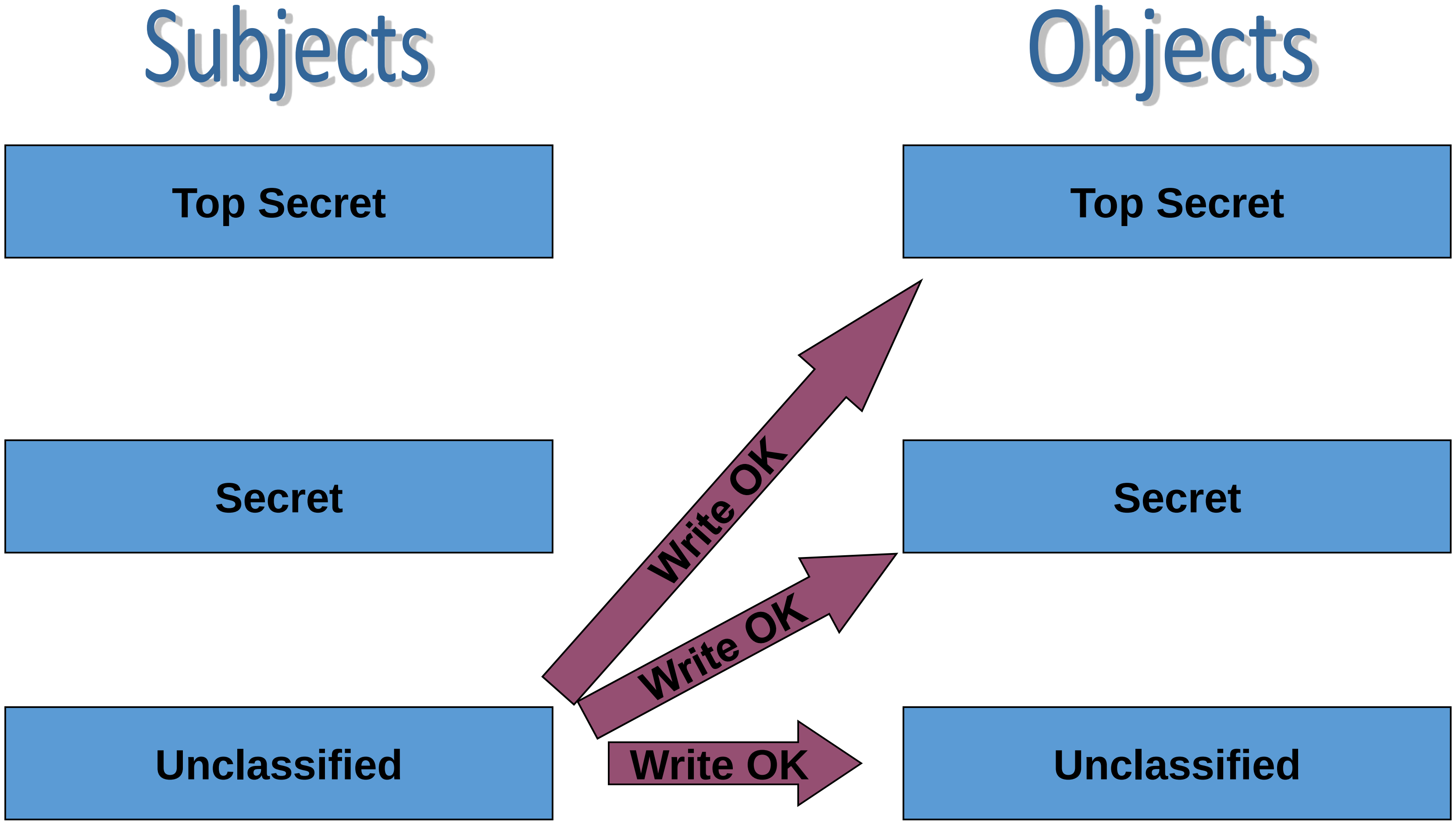
Unclassified

Write OK

Write OK

Write Forbidden

Access Control: Bell-LaPadula



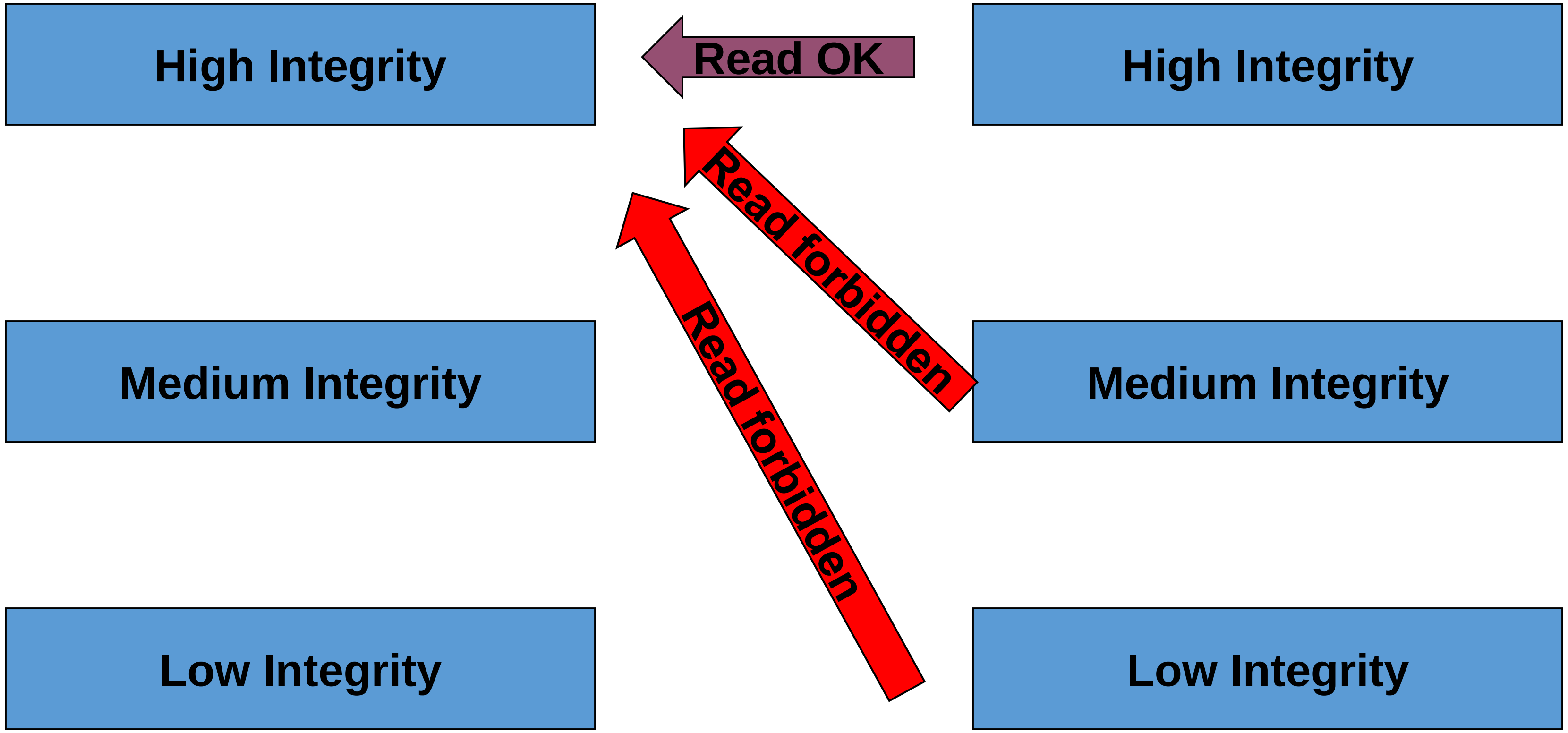
Security Models - Biba

- Based on the Cold War experiences, information *integrity* is also important, and the Biba model, complementary to Bell-LaPadula, is based on the flow of information where preserving integrity is critical.
- The “dual” of Bell-LaPadula

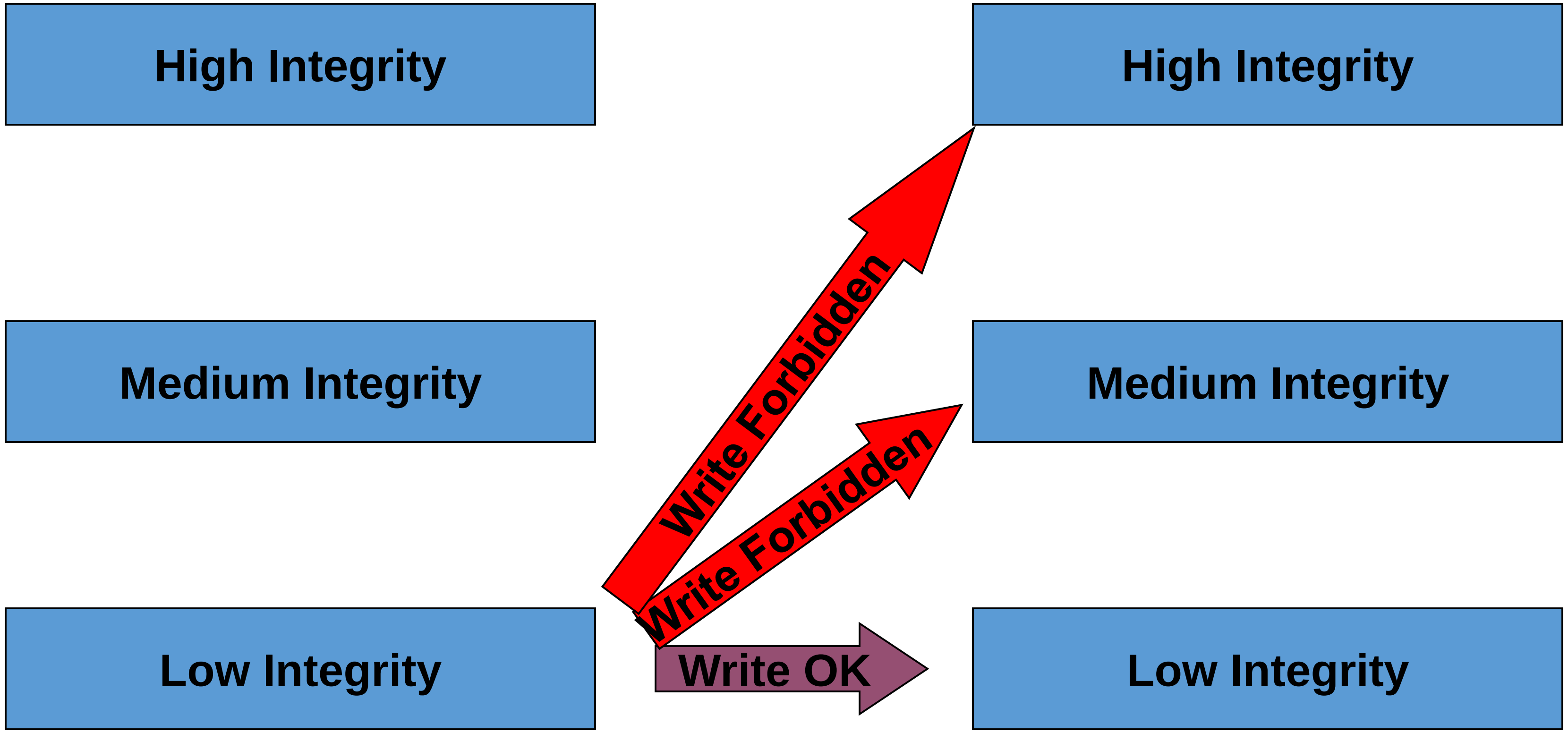
Integrity Control: Biba

- Designed to preserve integrity, not limit access
- Three fundamental concepts:
  - Simple Integrity Property – no read down
  - Star Integrity Property (*) – no write up
  - No execute up

Integrity Control: Biba



Integrity Control: Biba



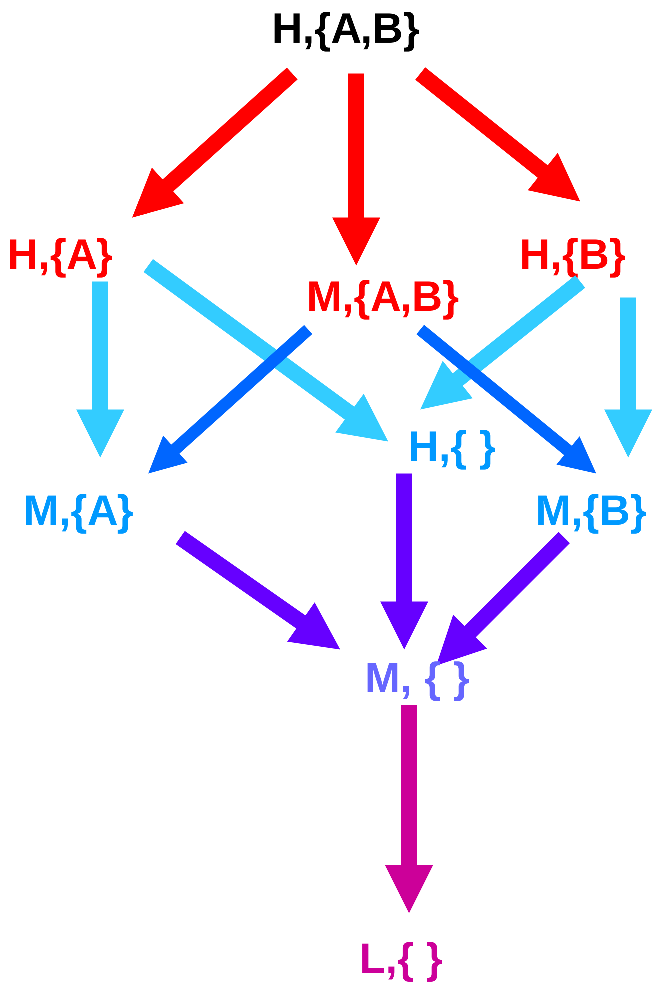
Combining integrity and privacy into a lattice

- Integrity
  - High Integrity (H)
  - Medium Integrity (M)
  - Low integrity (L)
  - No integrity (N)
- Confidentiality
  - {A,B} can be read by both A and B
  - {A} can be read only by A
  - {B} can be read only by B

Security Lattice

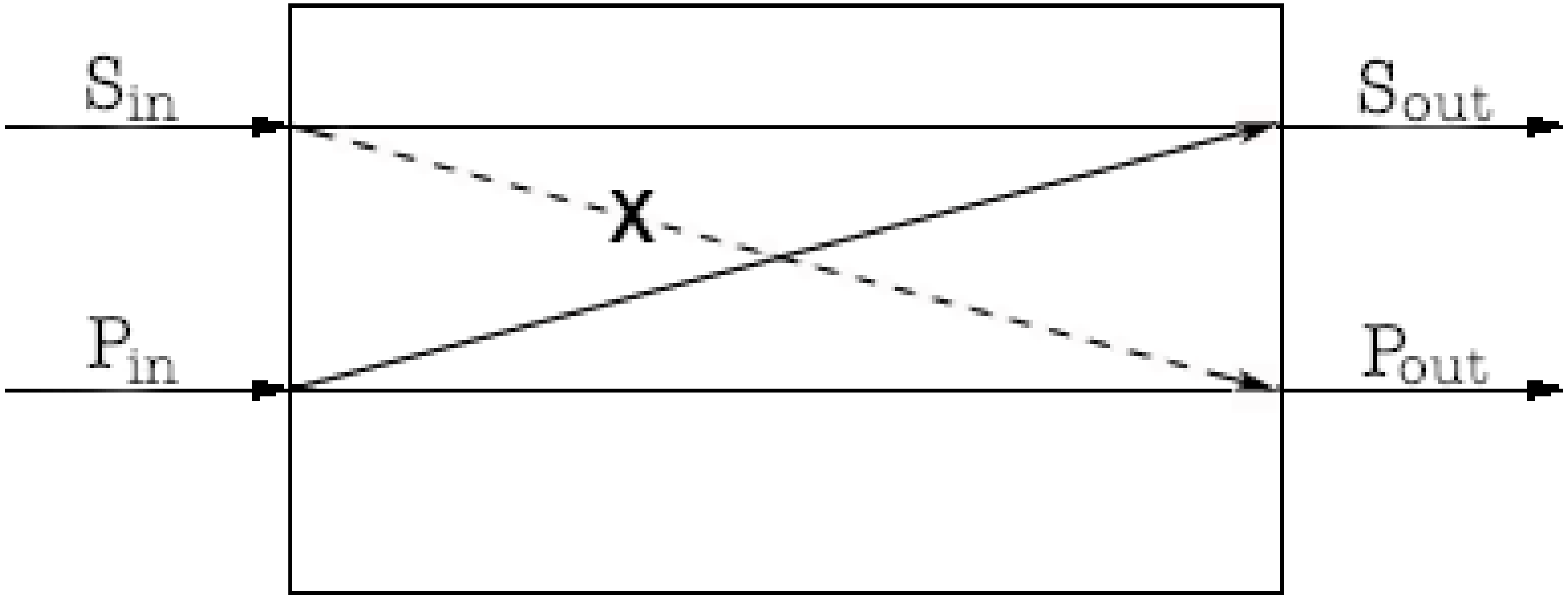
- *S* is the set of all security levels
  - Suppose the integrity categories are H (high integrity), M (medium integrity), L (low integrity)
  - Suppose the confidentiality categories are {A},{B},{A,B} and {}.
  - Then States = [(H, {}), (H,{A}), (H,{B}), (H,{A,B}), (M, {}), (M,{A}), (M,{B}), (M,{A,B}), (L, {}), (L,{A}), (L,{B}), (L,{A,B}),].

Information Flow in a security lattice



Information Flow in Programming

- Make sure secret inputs do not flow to public outputs
- This is called the property of non-interference (NI)



Do these programs satisfy NI?

- p_out = p_in + s_in (no)
- s_out = p_in (yes)
- p_out = s_in;
  p_out = 1; (yes)
- if (s_in mod 2) = 0 then p_out = 0; else p_out = 1; (no)
- while (s_in != 0) do { //nothing } (yes)