Lec 20: Mandatory Access Control

CSED415: Computer Security

Spring 2026

Seulbae Kim

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Recap

- Discretionary Access Control: Owner decides everything!
  - Owner of a resource decides who can access what and how
  - Strengths
    - Flexibility: Users can instantly share or revoke access without system's help
    - Psychological acceptability: "My file, my rules"

DAC failure examples

- Example 1:
  - Alice owns a file (f_A). She sets its permissions to: `rw- r-- ---`
    - Owner: Alice, Group: Friends
  - Bob is in the Friends group. Therefore, Bob can read Alice's file
  - Claire is not in that group. She cannot read the file
  - Bob duplicates Alice's file to f_b
  - Bob sets f_b's permissions to `rw- rw- r--`
  - Now Claire can see the data in Alice's file

DAC failure examples

- Example 2:
  - Alice works in HR
  - Alice can access `salary.xlsx`
  - Malware infects Alice's computer
  - Malware runs with Alice's privileges
  - Malware reads and copies `salary.xlsx`

DAC only asks, "Can Alice access this file?"

DAC does not ask, "Should this program be allowed to read HR data?"

Mandatory Access Control (MAC)

Two problems with DAC

- Information flow control problem:
  - You cannot stop users from copying user data to a wider audience
- Administrative issue:
  - In many organizations (e.g., a company), administrators should decide how the organization's sensitive data can be shared
  - Users should not be able to control anything

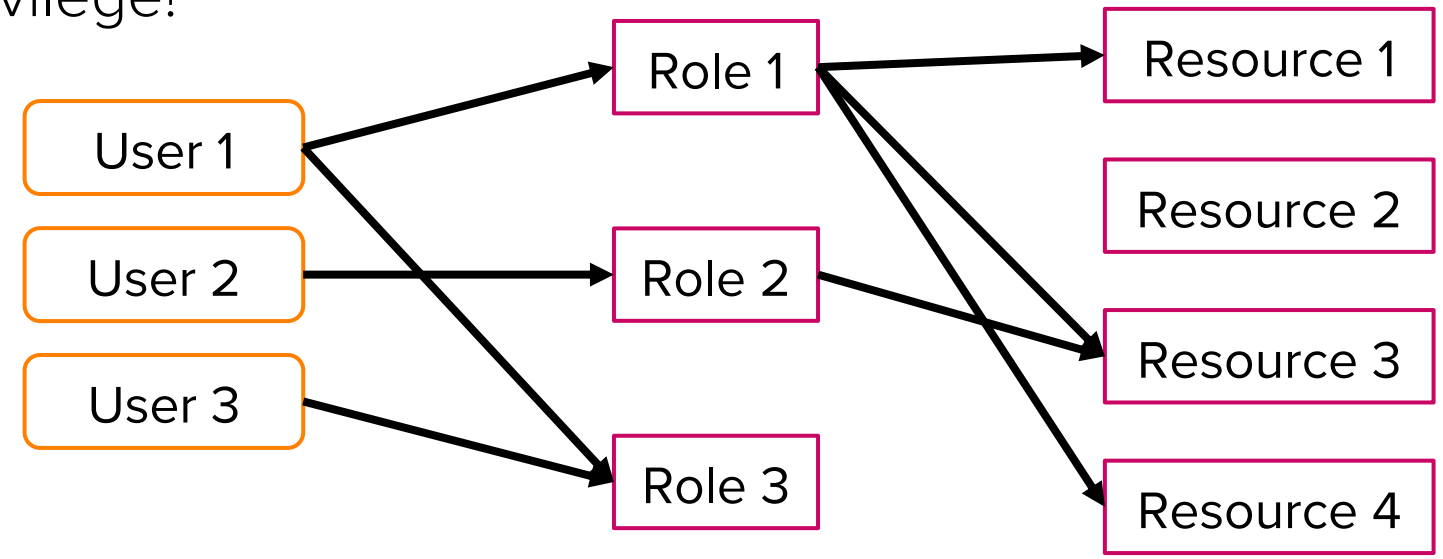
Mandatory Access Control (MAC) helps address these problems by assigning system-wide rules that users cannot override

Mandatory Access Control (MAC)

- Core idea:
  - Assign additional **attributes** to subjects and objects
    - Access is controlled based on the attributes
  - A system-wide policy checks these attributes on every access
    - Users cannot override it
    - Hence, “mandatory”

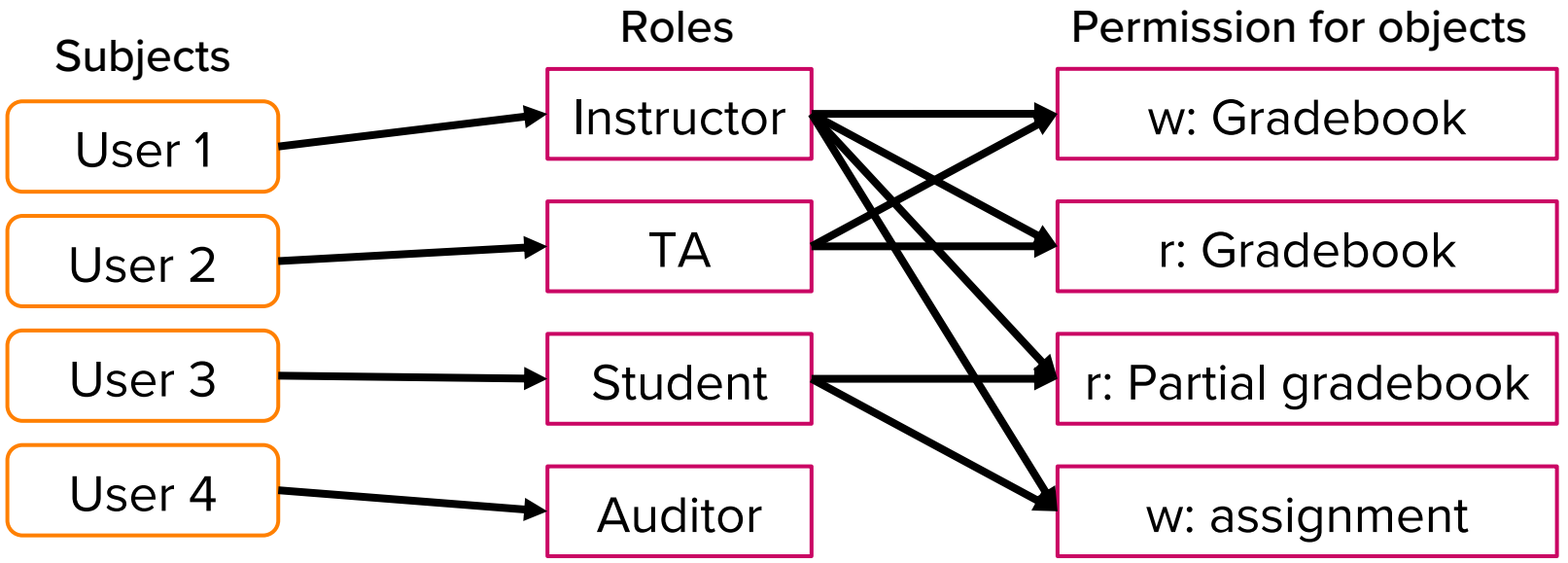
Role-based Access Control (RBAC)

- Attributes can be “roles”
  - Assign multiple roles to users
  - Each role is associated with a different permission
  - RBAC controls access based on the roles of the users
    - Least privilege!



Role-based Access Control (RBAC)

- Attributes can be “roles”
  - Example: CSED415



Q) Can User 2 create a copy of the gradebook and let User 3 read it?

A) No. Copied gradebook inherits the original role-based permissions.
User 3 (role: Student) cannot access it.

Attribute-based Access Control (ABAC)

- ABAC is a generalization of MAC
  - Three key elements
    - Attributes: Defined (or naturally determined) for entities
      - Age, department, login time, etc.
    - Policy: Defines access policies for attributes
      - e.g., “If age is over 21, allow access to alcohol”
    - Architecture model: Defines the relationship between policies
  - Flexible and expressive!
    - Attributes are dynamic → We will see an example

Attribute-based Access Control (ABAC)

- RBAC vs ABAC Example: Movie Rating
  - If using RBAC
    - Roles:
      - Adult, Juvenile, or Child
    - Permissions:
      - Can view R-rated movies, can view PG-13-rated movies, and can view G-rated movies
    - Policy:
      - Adult role gets assigned all three permissions
      - Juvenile role gets permissions for PG-13- and G-rated movies
      - Child role gets permission for G-rated movies

Movie Rating	Allowed Viewrs
R	Age 17+
PG-13	Age 13+
G	Everyone

User-to-role assignment and role-to-permission assignments are manual

Management of roles is also manual (e.g., need to change role when a child turns 17)

Attribute-based Access Control (ABAC)

- RBAC vs ABAC Example: Movie Rating
  - If using ABAC
    - Do not need explicitly defined roles
    - Permissions:
      - Can view R-rated movies, can view PG-13-rated movies, and can view G-rated movies
    - Policy:
      - `if (get_age(user) >= 17) return {R, PG-13, G}`
      - `else if (get_age(user) >= 13) return {PG-13, G}`
      - `else return {G}`

Movie Rating	Allowed Viewrs
R	Age 17+
PG-13	Age 13+
G	Everyone

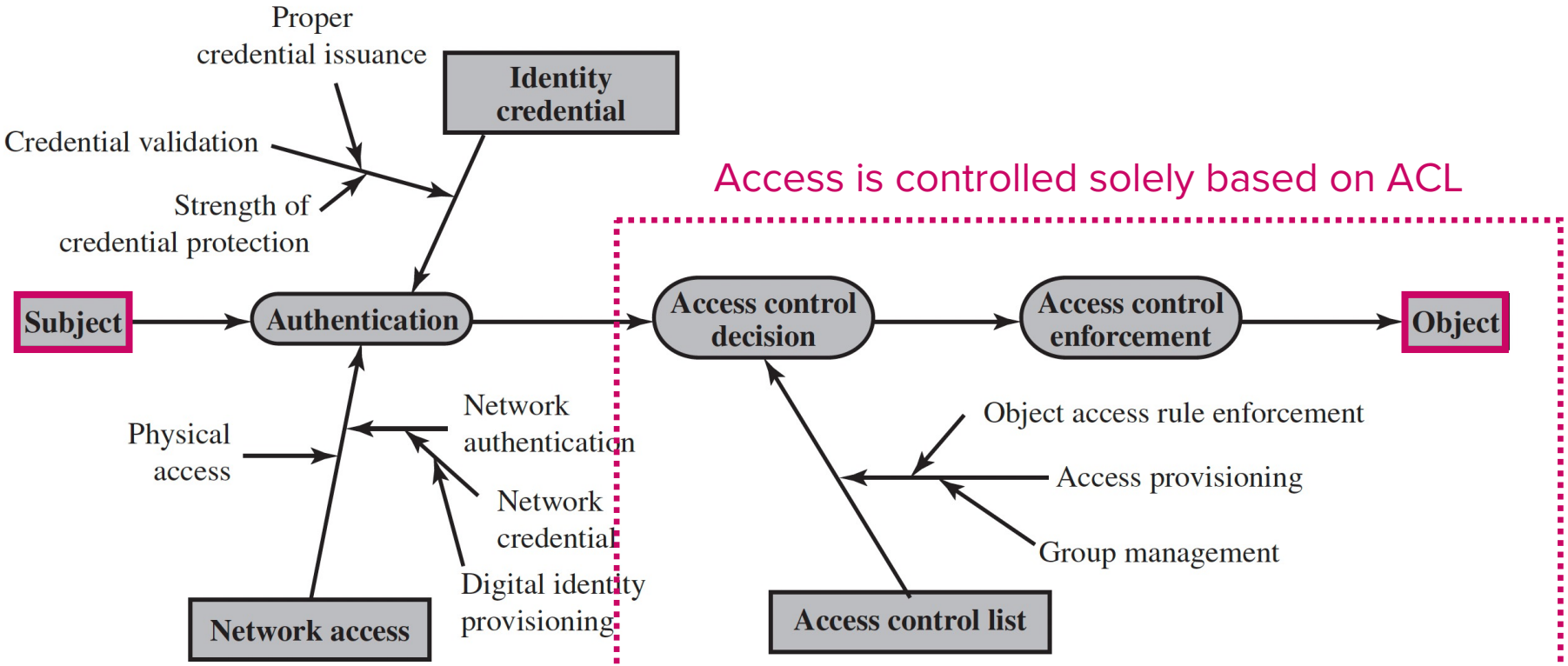
Only need to define policy with subject's attributes (age)

Do not need to redefine/manage static roles (people age automatically)

Additional ratings can be readily handled (e.g., VIP-only-rating → add one policy)

Attribute-based Access Control (ABAC)

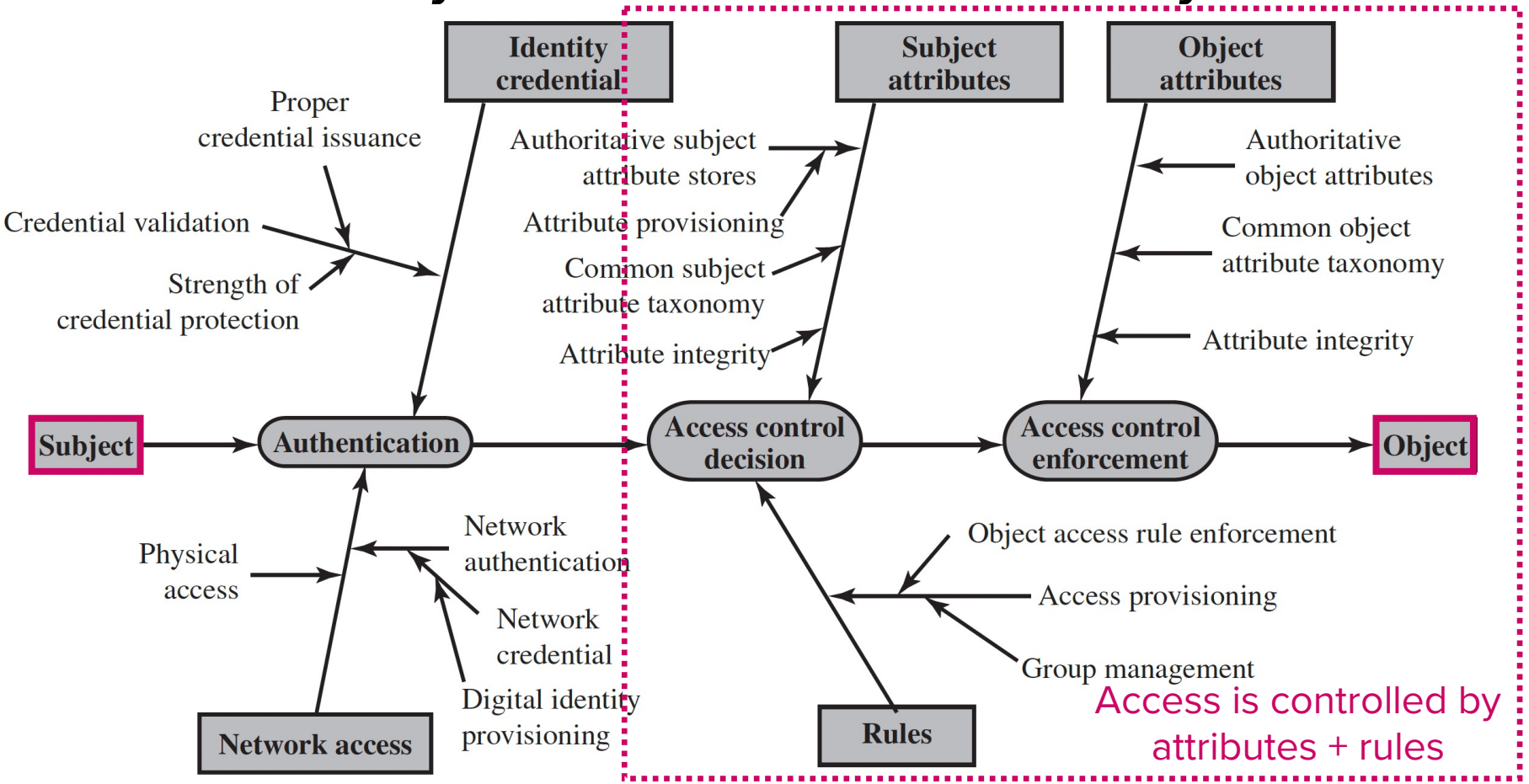
- DAC vs ABAC – Subject wants to access Object



Access control using DAC

Attribute-based Access Control (ABAC)

- DAC vs ABAC – Subject wants to access Object



Access control using ABAC (MAC)

MAC for Sensitive Data

Security labels & clearances

- Label:
  - Metadata that describes the nature of resource (e.g., attribute)
  - Indicate sensitivity, category, and clearance requirements of users
    - How sensitive is the data?
    - What kind of data is contained in the object?
  - OS associates labels with each user and object
- Clearance:
  - What a user/process is cleared to read/write
  - e.g., Alice has a top-secret clearance

Security labels & clearances

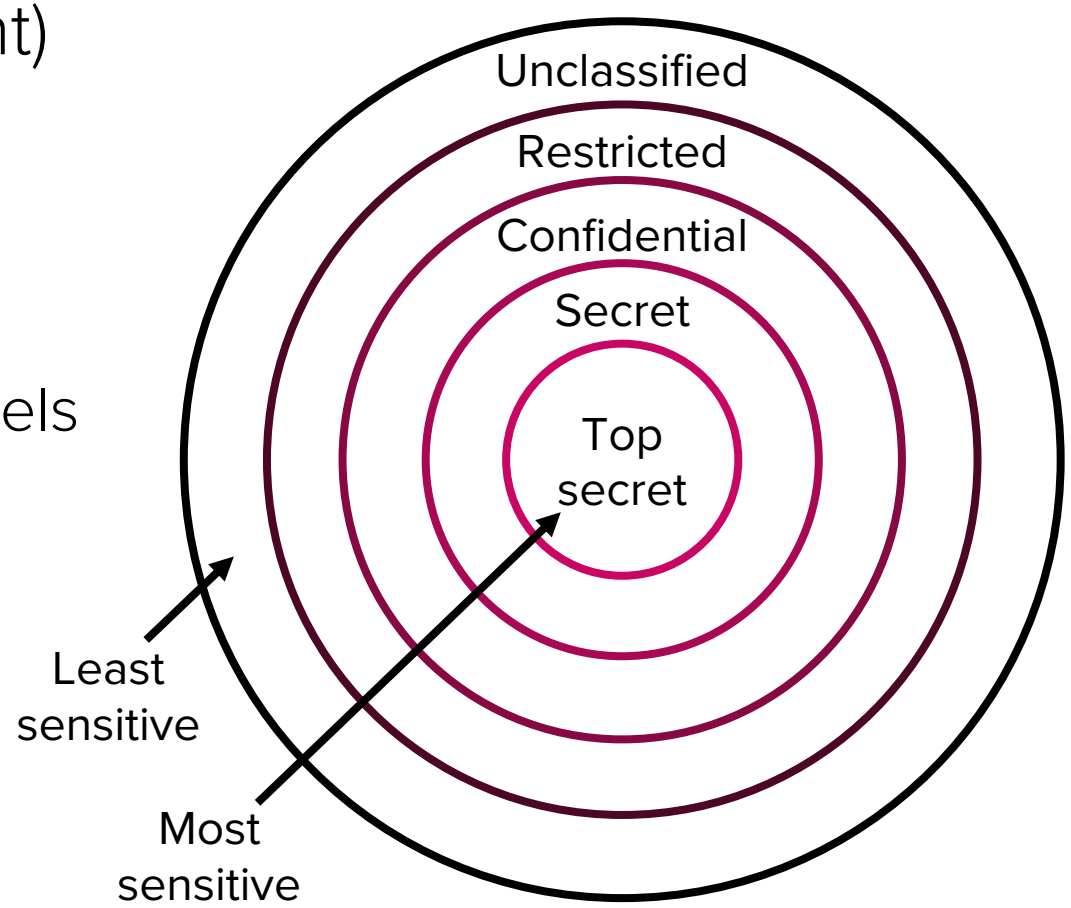
POSTECH

- Label-based access control:
  - Compare the security label of an object with the security clearance of a subject for access control
    - Label indicates how sensitive a resource is
    - Clearance indicates how eligible a subject is

Example: Military security policy

- US Department of Defense (DoD) model:
  - Label = (sensitivity level, compartment)
  - e.g., weapon documents
    - Label 1 = $(TS, \{nuclear, chemical\})$
    - Label 2 = $(S, \{nuclear, missile\})$

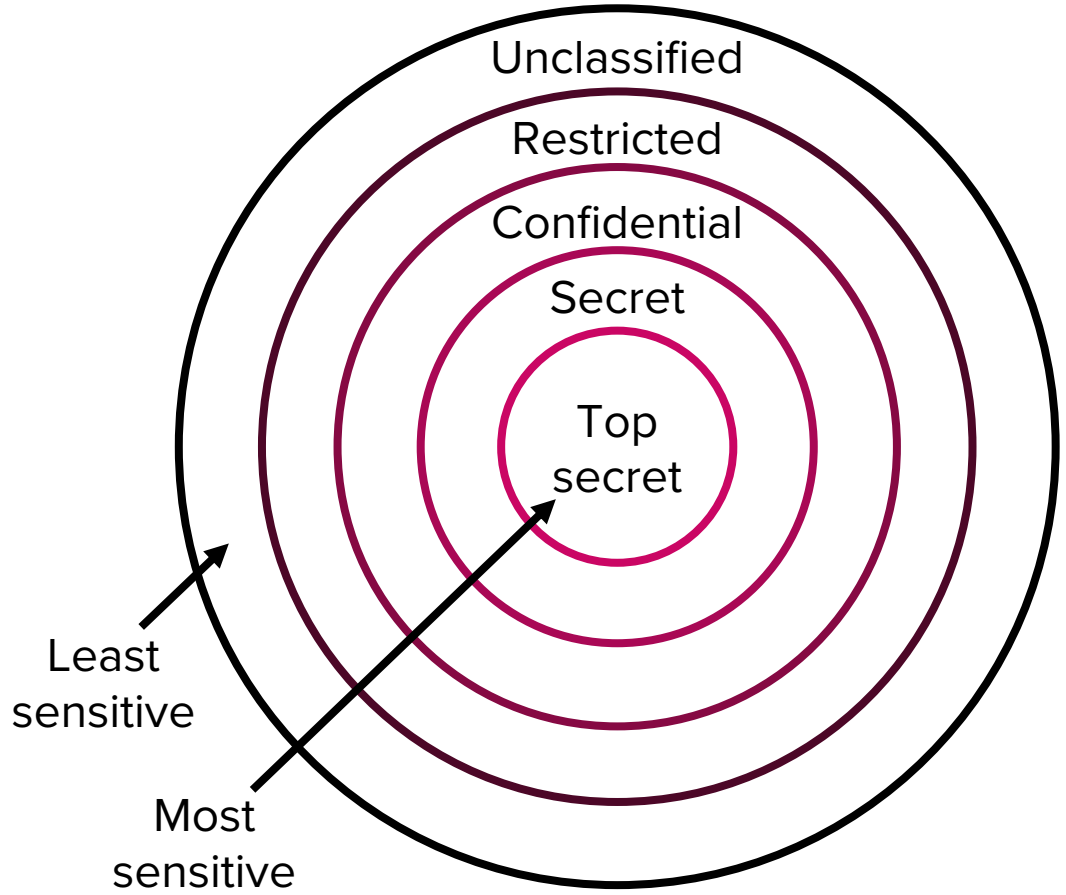
    → Users are authorized based on the labels



Comparing labels

- Sensitivity levels are totally ordered
  - i.e., $TS > S > C > R > U$

- Compartments are sets that are partially ordered
  - $nuclear \in weapon$
  - $chemical \in weapon$
  - $nuclear \in ? chemical$



Comparing labels

- levels (l_i) are compared by their orders
- Compartments (c_j) are compared using containment
- Example: $L_1 = (l_1, c_1), L_2 = (l_2, c_2)$

L_1 dominates L_2	$l_1 > l_2$ and $c_1 \supseteq c_2$
L_1 is dominated by L_2	$l_1 < l_2$ and $c_1 \subseteq c_2$
L_1 is equivalent to L_2	$l_1 = l_2$ and $c_1 = c_2$
L_1 and L_2 are not comparable	All other cases

Ordering labels

$TS > S > C > R > U$

POSTECH

- By comparing the labels, we can order them
  - Example:
    - $L_1 = (TS, \{CSE, EE, ME\})$
    - $L_2 = (S, \{CSE, EE\})$
    - $L_3 = (S, \{EE, PHY\})$
    - $L_4 = (C, \{CSE, PHY\})$

L_1 dominates L_2

All other pairs of labels are not comparable

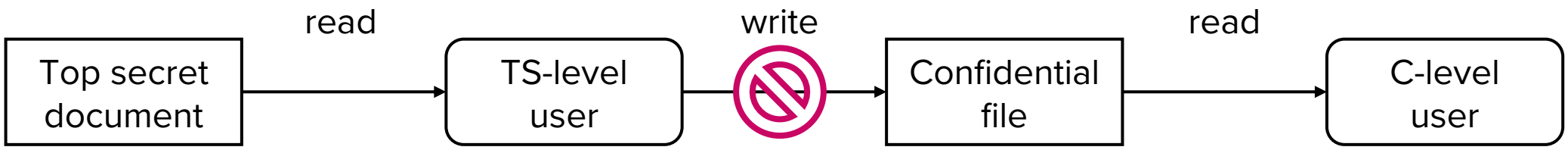
Now, what are actual MAC policies that utilize these labels?

Bell-LaPadula (BLP) model

$TS > S > C > R > U$

POSTECH

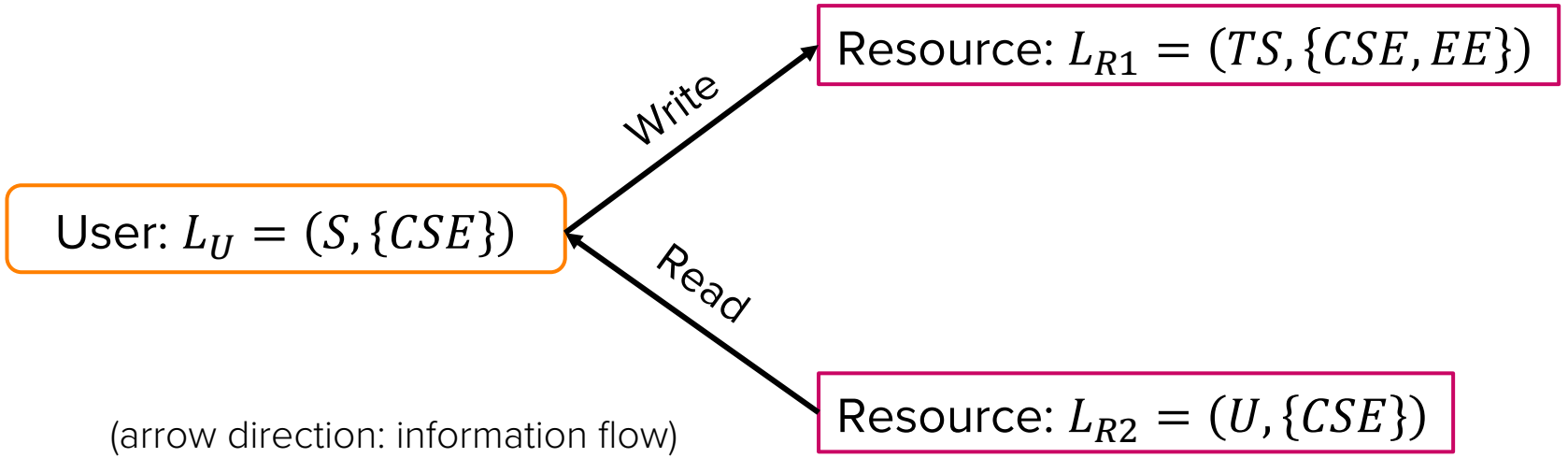
- Access control model focusing on confidentiality
  - BLP goal: Preventing leakage from high to low sensitivity



Bell-LaPadula (BLP) model

$TS > S > C > R > U$

- Access control model focusing on confidentiality
  - BLP enforces "Read Down, Write Up" (RDWU) rules
    - **Read-down rule**: User with L_1 clearance can read document with label L_2 only when L_1 is equivalent to or dominates L_2
    - **Write-up rule**: User with L_1 clearance can write document with label L_2 when L_1 is equivalent to or is dominated by L_2

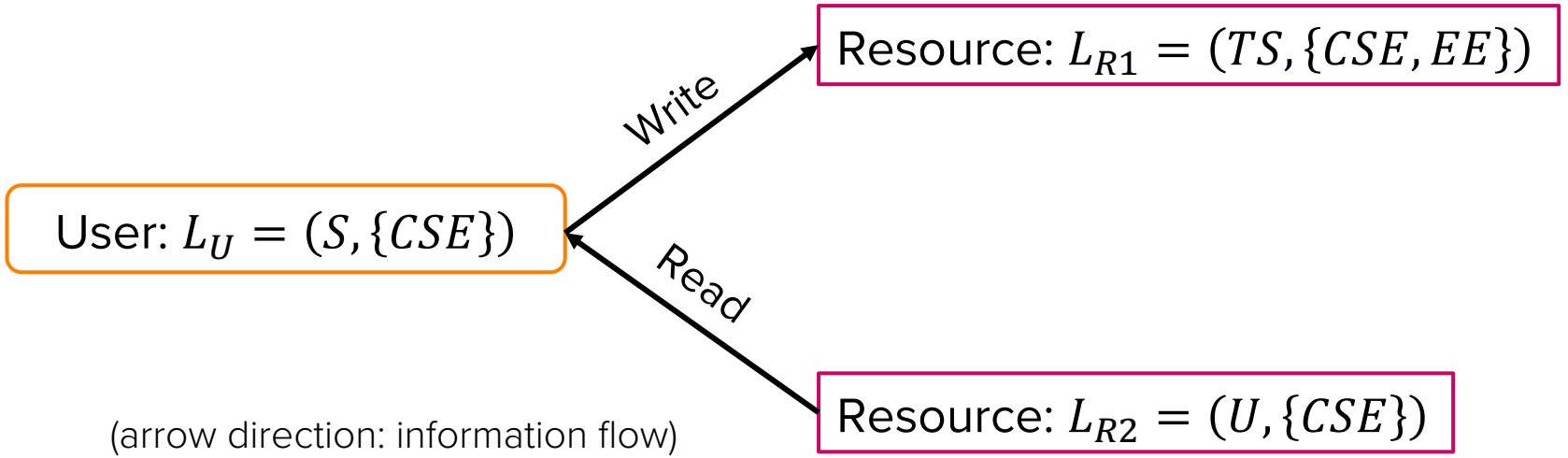


(arrow direction: information flow)

Bell-LaPadula (BLP) model

$TS > S > C > R > U$

- Access control model focusing on confidentiality
  - BLP enforces “Read Down, Write Up” (RDWU) rules
    - Rationale: More sensitive information should not flow to users who do not have clearance for that level
      - If write-down is allowed, user with high security clearance can leak information to lower security level users



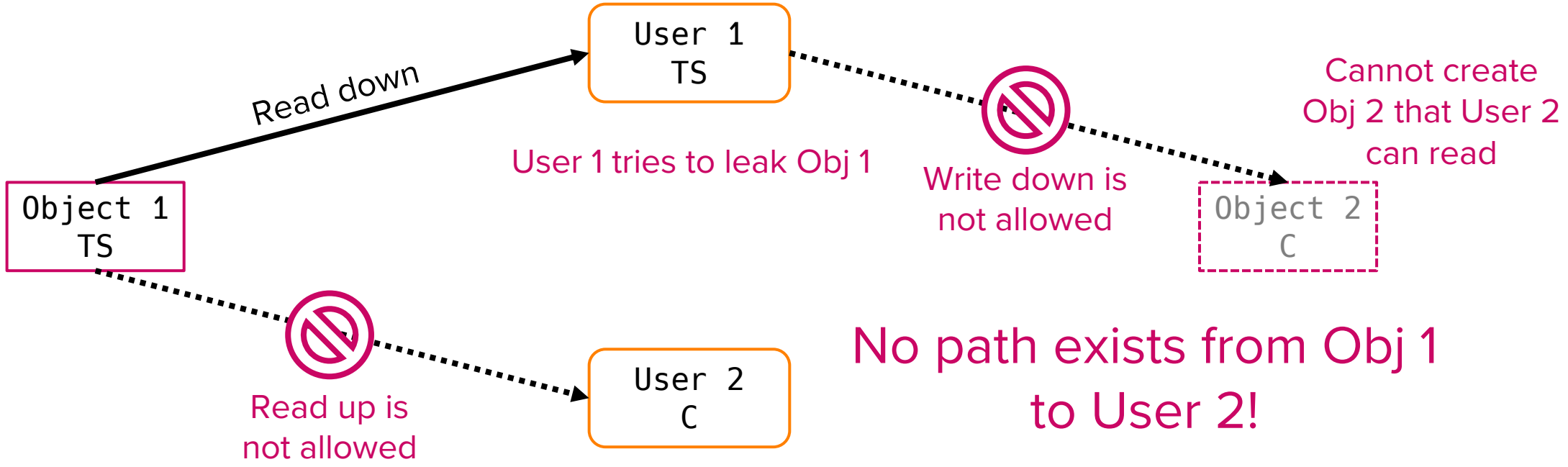
(arrow direction: information flow)

Bell-LaPadula (BLP) model

$TS > S > C > R > U$

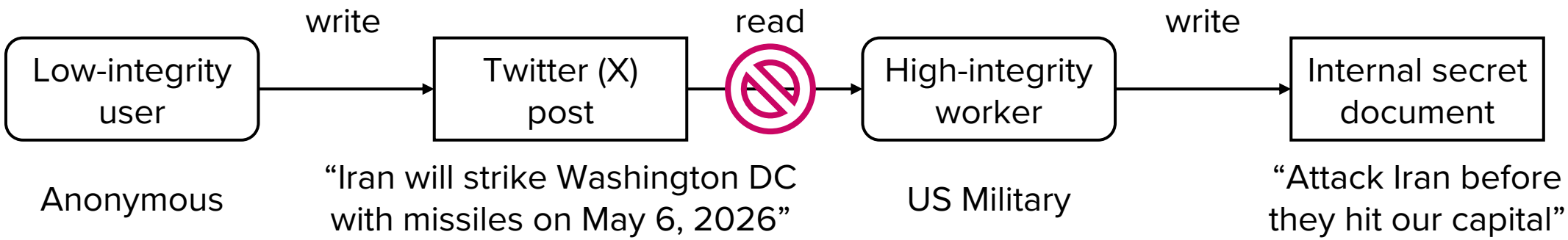
POSTECH

- BLP solves the information flow control problem
  - Confidential-level user (User 2) cannot read top secret Object 1
  - Top secret-level user (User 1) cannot create confidential Object 2



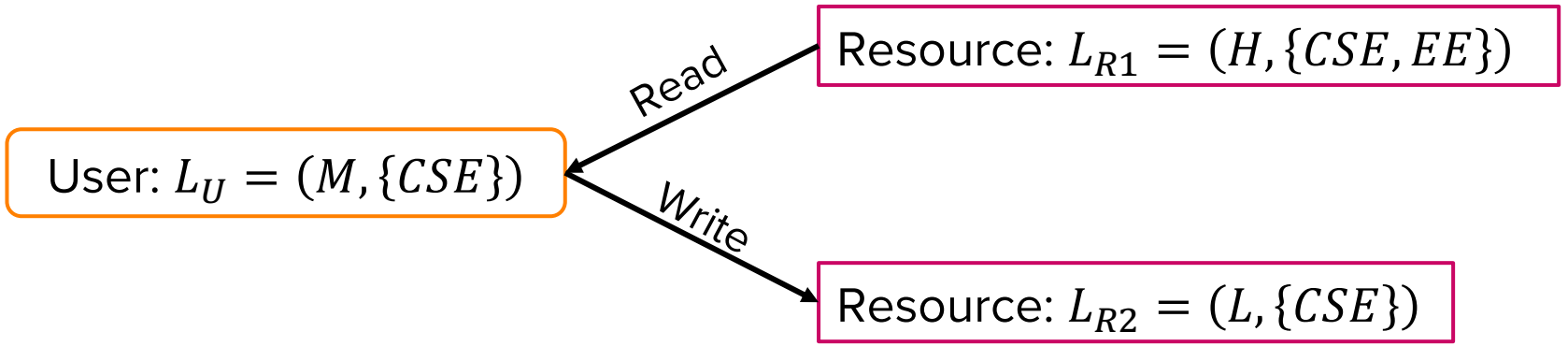
Biba Integrity model

- Access control model focusing on integrity
  - Biba goals: Preventing low-integrity information from contaminating trusted systems



Biba Integrity model

- Access control model focusing on integrity
  - Integrity level defines the quality of information
    - Levels: High (trustworthy), medium (mediocre), or low (untrustworthy)
  - Biba enforces “Read Up, Write Down” (RUWD) rules (opposite of BLP)
    - **Read-up rule**: Low integrity users can access high integrity information
    - **Write-down rule**: High integrity users can produce low integrity information



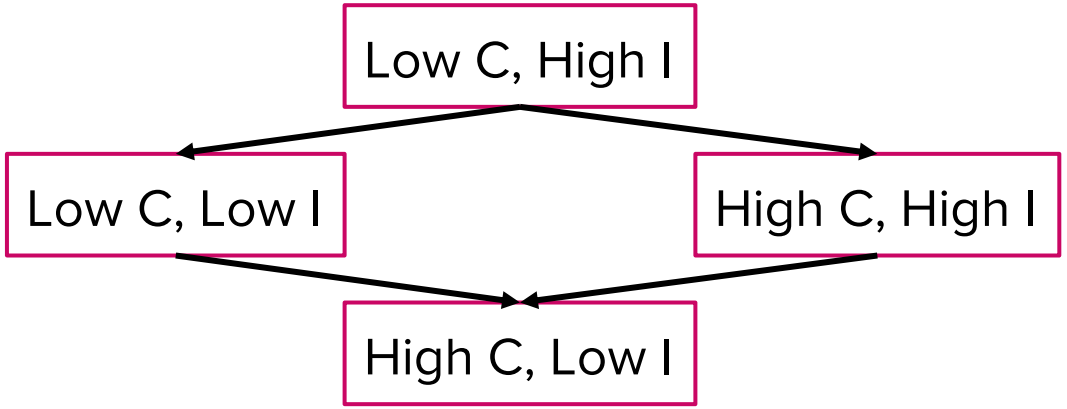
(arrow direction: information flow)

Biba Integrity model

- Access control model focusing on integrity
  - Example: You are a junior employee at a stock trading firm
    - RU: You may read market reports (high integrity) from your firm
    - No WU: You may not write or edit the market reports your firm publishes
    - No RD: You may not consult Reddit posts (low integrity) or tweets for market research
    - WD: You may tweet about the markets on your twitter account

BLP and Biba are contradictory

- Bell-LaPadula: Read Down, Write Up for confidentiality
- Biba Integrity: Read Up, Write-Down for integrity
- Can we combine the BLP and Biba models?
  - If a single label is used for both confidentiality and integrity, then the two models conflict
  - We need to use independent labels for confidentiality and integrity



MAC in Commercial Context

Models for commercial environments

- BLP / Biba Integrity models are intended for use in military settings where users (soldiers and officers) have clearances (labeled), and documents are classified (also labeled)
- MAC is also needed in commercial settings
  - Companies should limit how information can be shared
  - Challenges
    - Normal users usually do not have clearances
    - Labeling information is challenging

Clark-Wilson model

- Clark-Wilson model focuses on integrity in commercial setting
  - “No user of the system, even if authorized, may be permitted to modify data items in such a way that assets or accounting records of the company are lost or corrupted”

Clark-Wilson model

- Two principles for data integrity
  1. Well-formed transaction
     - A user cannot manipulate data arbitrarily
     - Users are only allowed to make “transactions”
     - Transactions constrain the ways in which users can modify the data
       - Transactions correspond to high-level operations that could be performed on data
       - e.g., `add_employee()`, `set_salary()`, `pay_salary()`, …
     - All transactions are recorded in a write-only log

Data can only be manipulated through trusted code!

Clark-Wilson model

- Two principles for data integrity
  2. Separation of duty
     - Responsibilities are divided among different users
     - All operations are divided into subparts
       - Each subpart must be executed by a different person
       - e.g., Two-person rule for critical operations
         (such as launching a missile)
         - One person inserts and turns a launch key
         - Another person types in the launch password

Clark-Wilson model

- Example: Placing an order
  1. A purchasing agent creates an order for a supply
     - The agent sends copies of the order to both the supplier and the logistics agent
  2. The supplier ships the goods to the logistics agent
     - The logistics agent conducts an integrity check to verify the correctness of the shipment (amount, quality, …)
  3. A delivery confirmation is sent to finance department agent
     - The finance agent pays the supplier after reviewing both the order and delivery confirmation

Clark-Wilson model

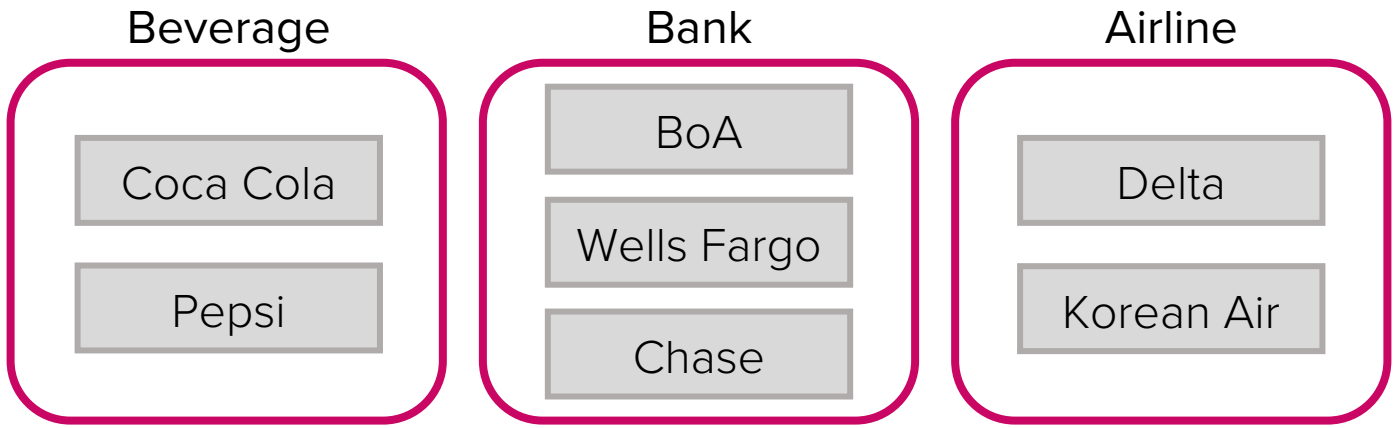
- Example: Placing an order
  - `<User, transaction, {data}>` triples:
    - `<Purchasing agent, place_order, {order book}>`
    - `<Logistics agent, receive_delivery, {inventory}>`
    - `<Finance agent, make_payment, {account balance}>`
  - Separation of duty:
    - Different agents for subparts of “order supply” operation
    - What if the same agent takes charge of the entire process?
  - Constraints:
    - The logistics agent must have the order before accepting delivery
    - The finance agent must have the delivery confirmation prior to payment

Chinese Wall policy

- Chinese Wall policy focuses on confidentiality
  - Motivated by Conflict of Interest (CoI) requirements
  - Example:
    - A law firm has many clients
    - Some clients have competitive relationships (e.g., Coca Cola and Pepsi)
    - Chinese Wall policy aims to avoid CoI between competitors

Chinese Wall policy

- Focus on confidentiality
  - Conflicting groups



  - Policy: User U can access object O that belongs to company C as long as U has not accessed any object from other companies in C's conflicting group

MAC is difficult

- Traditional MAC systems are secure, yet often painful to deploy
  - Operational challenges
    - Policies are hard to configure
    - Labels are hard to maintain
    - Misconfiguration breaks applications
    - Users dislike reduced flexibility

→ Modern systems instead use practical policy languages and sandboxing

MAC in practice: AppArmor

AppArmor

- Part of Linux Security Modules (LSM) that provides practical mandatory access control
  - Goal: Make MAC policies easy to write and deploy
- Widely used for sandboxing privileged programs
- Enabled by default on Ubuntu Linux
  - e.g., CSED415 lab server (Ubuntu 22.04):

```
→ sudo aa-status
apparmor module is loaded.
44 profiles are loaded.
44 profiles are in enforce mode.
```

AppArmor

- AppArmor allows you to write per-process security
  - Each program is associated with a profile
  - Profile defines:
    - What files it can access
    - What capabilities it has
    - What operations are allowed
  - Profiles are easy to write
    - e.g., Disallowing read and write on files in `/tmp`: `deny /tmp/** rw`

Key design: Path-based policies

- Example: read-file
  - read-file.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  if (argc < 2) {
    fprintf(stderr, "Usage: %s <file>\n", argv[0]);
    return 1;
  }

  FILE *fp = fopen(argv[1], "r");
  if (fp == NULL) {
    perror("fopen");
    return 1;
  }
  int c;
  while ((c = fgetc(fp)) != EOF) {putchar(c);}

  fclose(fp);
  return 0;
}
```

Key design: Path-based policies

- Example: read-file
  - Compile and run

```
$ gcc read-file.c -o read-file

$ echo "hello world" > /tmp/testfile

$ ./read-file /tmp/xx
fopen: No such file or directory

$ ./read-file /tmp/testfile
hello world
```

Key design: Path-based policies

- Example: read-file
  - Policies can be written using file paths
    - e.g., `/etc/apparmor.d/read-file`

```
profile read-file /home/seulbae/apparmor-demo/read-file {
  # deny all access to /tmp
  deny /tmp/** rwlkmx,
  deny /tmp/ r,

  # allow everything else
  file,
}
```

Key design: Path-based policies

- Example: read-file
  - Applying and testing the policy

```
$ ./read-file /tmp/testfile
hello world

$ sudo apparmor_parser -r /etc/apparmor.d/read-file

$ ./read-file /tmp/testfile
fopen: Permission denied
```

→ Powerful and practical MAC for controlling program permissions!

Confused deputy problem in AppArmor

POSTECH

- CrackArmor: A vulnerability in AppArmor (2026)
  - Recommended reading: https://cdn2.qualys.com/advisory/2026/03/10/crack-armor.txt
  - Summary:
    - When a user executes **su**, it runs with root privilege due to the setuid bit

```
→ ls -al $(which su)
-rwsr-xr-x 1 root root 55680 Mar  7 01:10 /usr/bin/su
```

    - CrackArmor allowed unprivileged users to abuse **su** (i.e., confused deputy) to manipulate AppArmor profiles, e.g., to remove existing profiles or load malicious profiles

Summary

- Today: MAC (BLP, Biba, Clark-Wilson, Chinese Wall) enforces system-wide policy for information flow control
- So far: Authentication and access control for system security
  - All users are authenticated first
  - Based on the authenticated identity, users can request access to a resource
  - Access control policies compares the access rights of the identity to access control policy (e.g., ACL)

Coming up next

- Unfortunately, strong authentication and access control are not enough
  - Even perfectly authenticated users and correctly configured access control policies can become meaningless if malware compromises the system itself

  → Next topic: Malware and anti-malware

Questions?