Data Security & Privacy

CIS 545

Access Control Mechanisms

Birhanu Eshete
birhanu@umich.edu

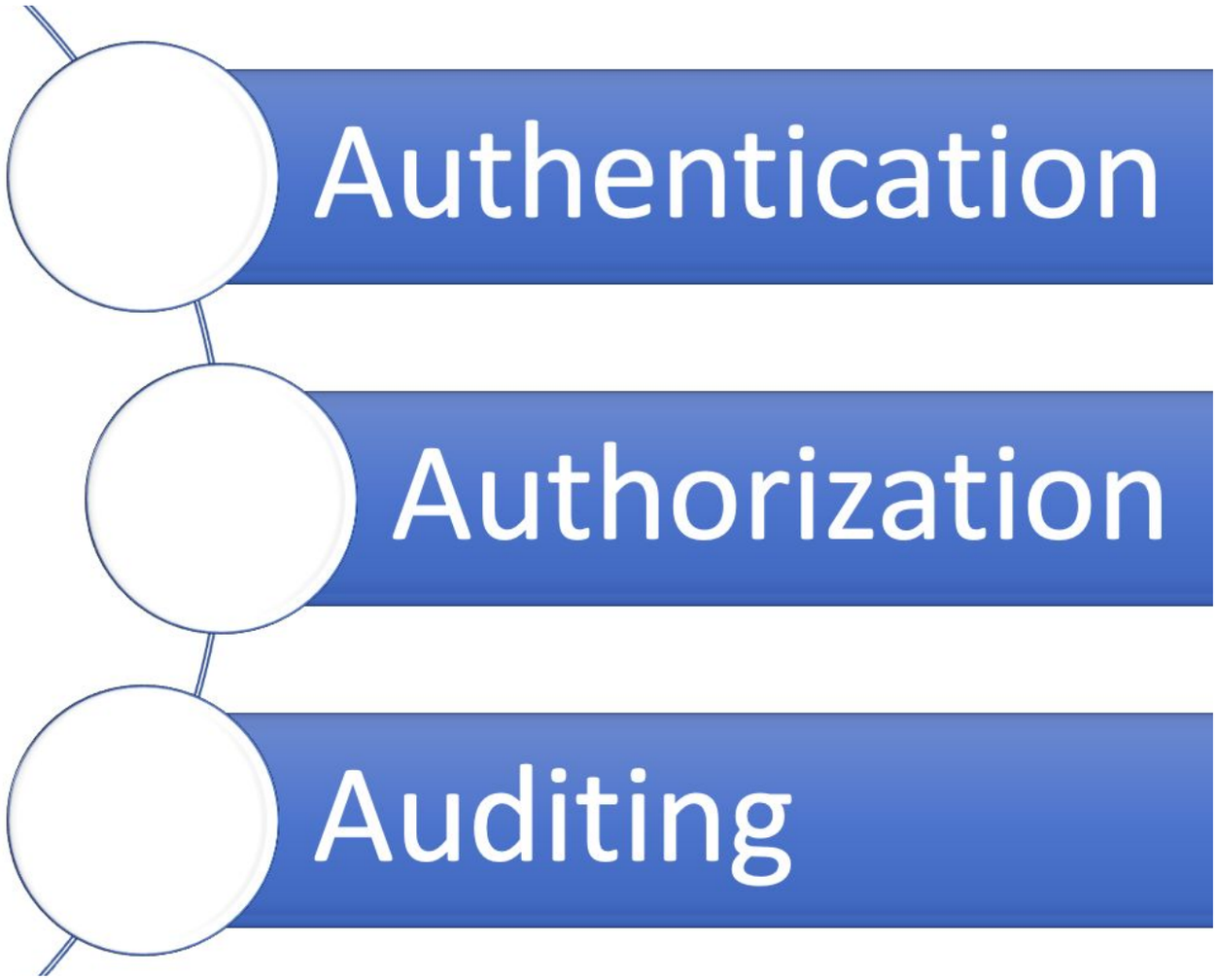
Lecture Goals

- Access Control Basics (what, guiding principles, typical setup, types)
- Access control modeling (ACM, ACL, CAP, …)
- More in depth on access control types:
  - Mandatory Access Control (MAC)
  - Discretionary Access Control (DAC)
  - Role-Based Access Control (RBAC)
  - Attribute-Based Access Control (ABAC)

Motivating Example

- Policy: UM-Dearborn’s academic integrity policy disallows cheating (includes copying others' homework, with or without permission)
- Context: students do homework on a shared server (e.g., shared.umd.umich.edu)
- What happened: Student A forgets to read-protect homework file hw1.py. Student B copies hw1.py and submits it as their own
- Question: Who breached security? A, B, Both?
  - Student A: did nothing wrong, except failure to read-protect hw1.py
  - Student B: violated the policy!
- Should A be liable for not read-protecting h1.py?
- What if A allowed B to copy hw1.py?

Access Control: More than Permissions



- Verifying identity (something: you know, you have, you are)
- Verifying authority (are you entitled to perform an operation on an asset?)
- Book-keeping for future evidence

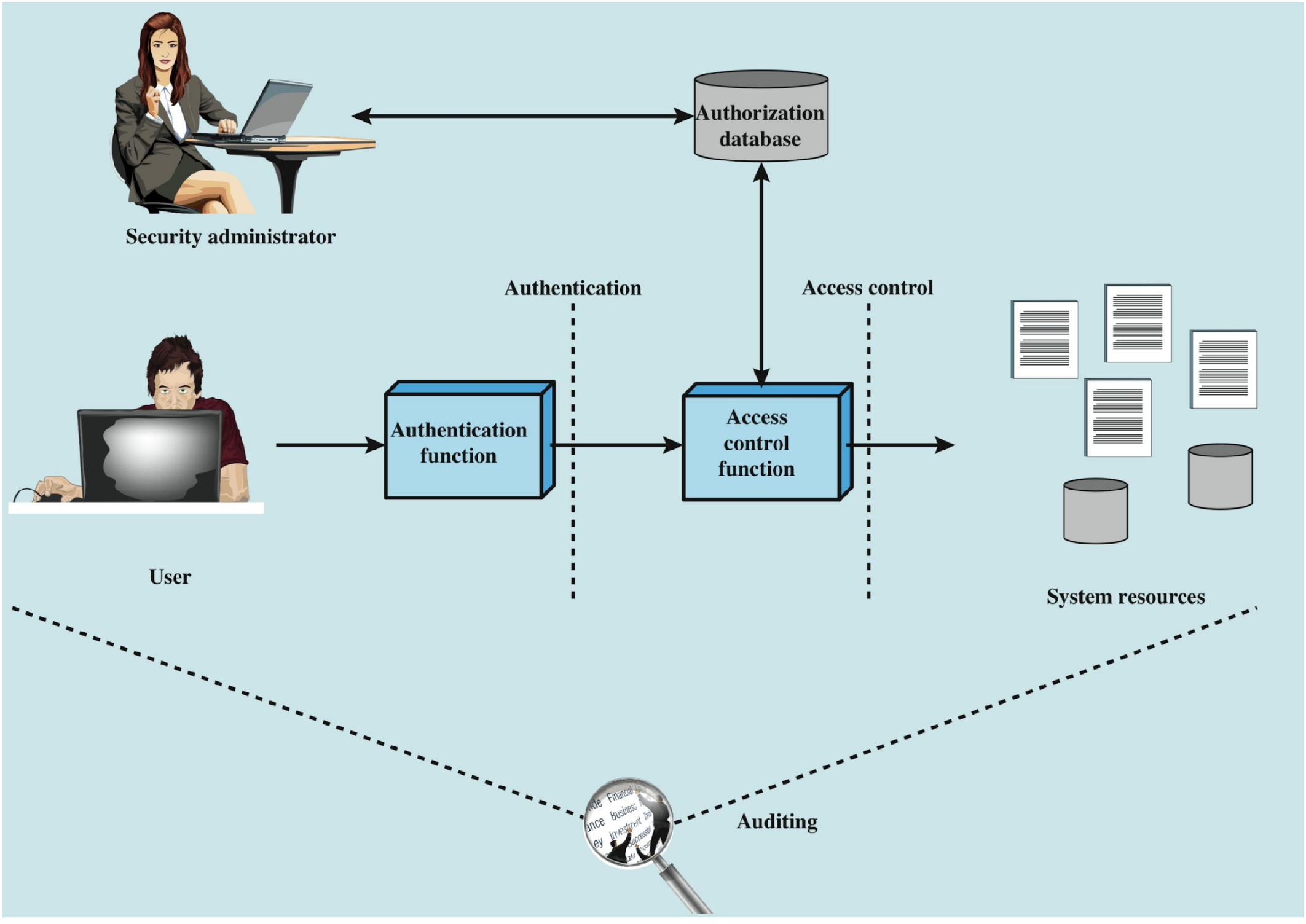
Authorization vs. Access Control

- Authorization:
  - the policy (who should be allowed to do what?)
- Access control:
  - the mechanism to enforce the policy
- Analogy:
  - the law on paper vs. (courts, the police)

Access Control Design: Guiding Principles

- Least privilege:
  - Never grant more than minimum access required to execute duties
- Need to know:
  - Specific data and specific times
- Separation of duties:
  - Segregate responsibilities to limit powers

Access Control Landscape



Example: Access Modifiers in OO-Languages

- Purpose: realization of encapsulation

Access Modifier	C++	Java	Python	PHP	C#
private	class	class	class	class	class
protected	subclass	subclass and/or same package	subclass	subclass	subclass
public	everybody	everybody	everybody	everybody	everybody
none (default)	class	same package	everybody	everybody	class

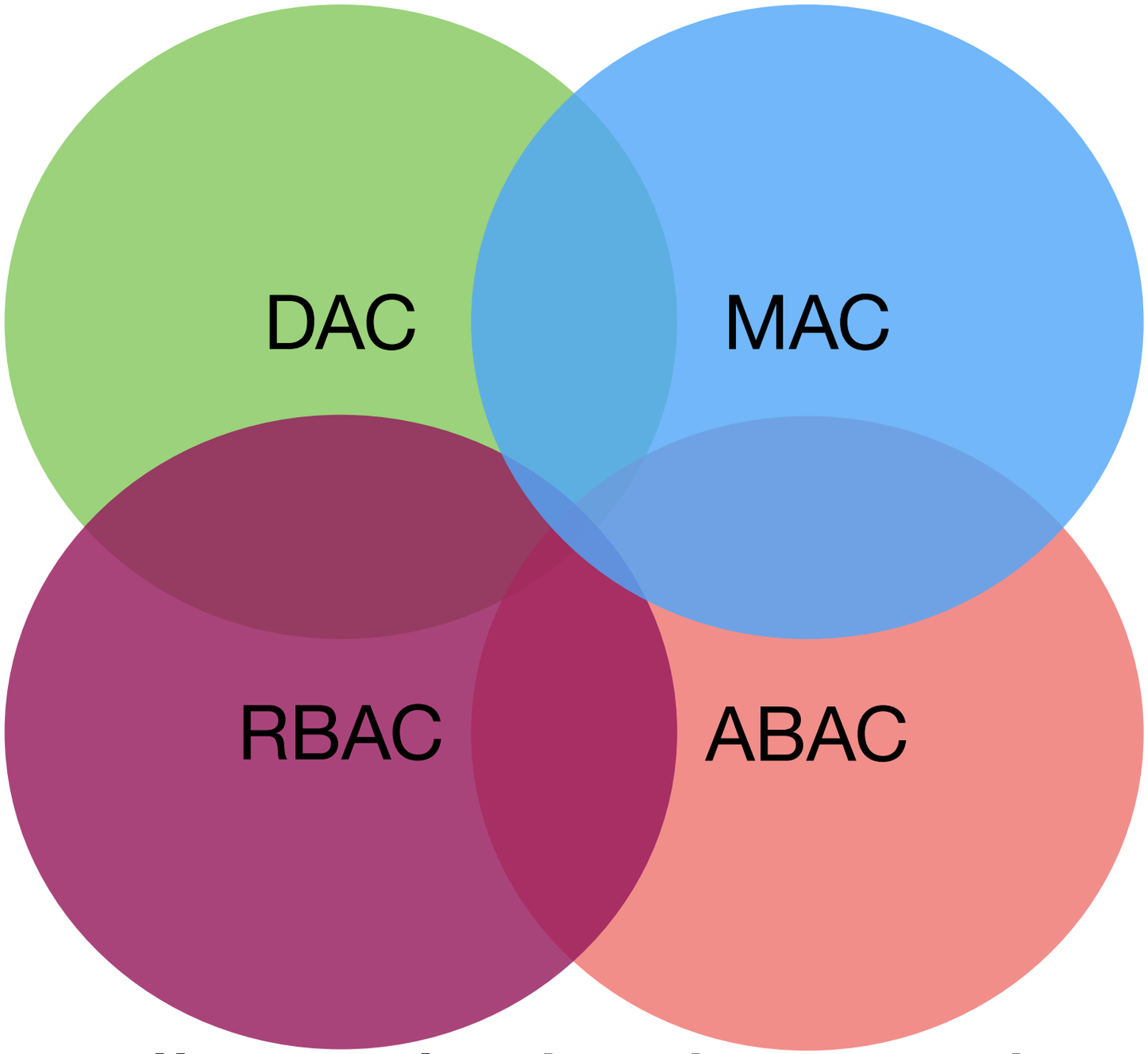
Access Control Types

- **Discretionary Access Control (DAC)**:
  - owner controls who can access object (e.g., Windows, Linux, Mac)
  - system's decision is limited by access privileges set by the owner
- **Mandatory Access Control (MAC)**:
  - system controls access to an object (e.g., SE Linux, Trusted Solaris)
  - based on security labels of subjects (clearance) and objects (classification)
  - system matches clearance (subject) with the classification (object)
  - usually used in confidentiality-critical environments (e.g., military)

Access Control Types …

- **Originator-Based Access Control (OBAC)**:
  - originator (creator) of object controls who does what on object
  - e.g., NDAs on code changes, licensing agreements
- **Role-Based Access Control (RBAC)**:
  - roles in the organization dictate rights
  - can implement DAC, MAC, …
- **Attribute-Based Access Control (ABAC)**:
  - attributes of subjects and objects dictate rights
  - authorization is based logical rules that evaluate attribute values
  - e.g., age-appropriate access to movies on a streaming service

Practical Deployment



- Not mutually exclusive in practice

Modeling Access Control

- **Subjects S**:
  - active entities in the system, that can act (e.g., users, programs, processes)
- **Objects O**:
  - passive entities in the system, acted upon by subjects (e.g., files/directories, sockets, devices, programs)
- **Rights R**:
  - what can the subject do on the object (e.g., execute, read, write, create, destroy, modify)

Simplified UNIX Model

- Processes are subjects:
  - p1, p2, p3, …
- Files are objects:
  - f1, f2, f3, …
- Operations are rights:
  - r,w,x,a,o, …

Access Control Matrix (ACM)

	F1	F2	F3
P1	wro		ao
P2	rx	w	rwxa
P3	rw		o

- Imagine the size of this matrix for a typical modern system (e.g., your own laptop)
- The most comprehensive model, but has issues

Pros and Cons of ACM

- **Pros:**

  - comprehensive (given s, O, R, you can always assign a subset of R to s on a subset of O)

- **Cons:**

  - does not model rules by which permissions can change (creation, deletion, update)
  - excessive memory requirements as the ACM grows (blank or same entries)
  - lookup performance degrades as the ACM grows

Modeling Alternatives

- **Alternative 1**: For each object (e.g., file), store the rights of each user (e.g., process) on that object
  - Con: doesn't scale with many users
  - E.g.: new user w/ read permission on every file
- **Alternative 2**: for each subject (e.g., process), store its rights to all objects (e.g., files)
  - Con: every time an object (e.g., file) is created, visit every subject and update the metadata
- Note 1: file vs. user creation? Which one is more frequent?
- Note 2: orphan files (user removed, but file is sitting there)

Access Control List (ACL)

- Each column of the ACM is stored with the object

F1
P1: rwo
P2: rx
P3: rw

F2
P1: rw
P2: rwx
P4: oa

F3
P3: r
P4: rwxa
P5: x

- What to do with a subject that has no rights to any object?
- What if many subjects have the same right over an object?

ACL Formalism

- Let S be the set of subjects, and R the set of rights:
- An ACL, l is a set of pairs $l = \{ (s, r): s \in S, r \subseteq R \}$
- Let *acl* be a function that determines the access control list l associated with a particular object o:
  - $acl(o) = \{ (s_i, r_i): 1 \le i \le n \}$ means subject s_i may access o using any right in r_i

Capability List (CAP)

- Each row of the ACM is stored with the subject

F1	F2	F3
P1: rwo	P1: rw	P3: r
P2: rx	P2: rwx	P4: rwxa
P3: rw	P4: oa	P5: x

ACL

P1	F1: rwo	F2: rw
P2	F1: rx	F2: rwx
P3	F1: rw	F3: r
P4	F2: oa	F3: rwxa
P5	F3: x	

CAP

Capability List Formalism

- Let O be the set of objects, and R the set of rights, of a system
- A capability list, $\mathcal{c}$ is a set of pairs:
  - $\mathcal{c} = \{ (o, r): o \text{ in } O, r \subseteq R \}$
- Let *cap* be a function that determines the capability list $\mathcal{c}$ associated with a particular subject s
  - $cap(s) = \{ (o_i, r_i): 1 \leq i \leq n \}$ means subject s may access o_i using any right in r_i

ACL vs. CAP (1/2)

- Q1: Given a subject, what objects can it access & how?
- Q2: Given an object, what subjects can access it & how?
- In theory, either can answer Q1 and Q2, but how simple or efficient?

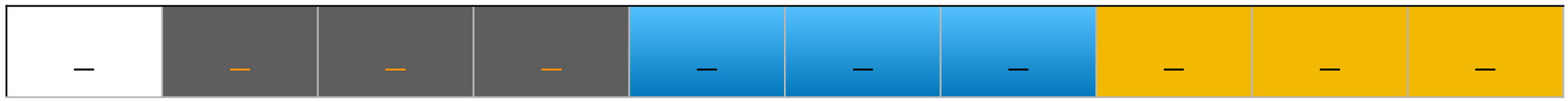
- For Q1: Capabilities (list elements of subject's CAP-list)
- For Q2: ACLs (list elements of object's ACL)
- ACL: to answer Q1, has to scan all objects
- CAP: to answer Q2, has to scan all subjects

ACL vs. CAP (2/2)

- Access review:
  - ACL better for access review of objects
  - CAP better for access review of subjects
- Revocation:
  - ACL better for revocation on object basis
  - CAP better for revocation on subject basis

ACL Examples in Linux

- 10 permission blocks for each object (file, directory), logically grouped into sets of three blocks for owner, group, and the world



d/l/- Owner Group Other

```
birhanu@brex:~/projects/marple$ ls -la
total 96
drwxr-xr-x   20 birhanu  staff    640 Sep 26  2018 .
drwxr-xr-x   35 birhanu  staff   1120 Jul 11 20:57 ..
-rw-r--r--    1 birhanu  staff  18436 Sep 19 13:39 .DS_Store
drwxr-xr-x   11 birhanu  staff    352 Sep 25  2018 TC2015
-rw-r--r--    1 birhanu  staff  16624 Sep 25  2018 THEIAParser.py
drwxr-xr-x    8 birhanu  staff    256 Sep 25  2018 TransparentComputing
drwxr-xr-x    7 birhanu  staff    224 Sep 25  2018 containers
drwxr-xr-x   12 birhanu  staff    384 Sep 25  2018 dsm-theia
drwxr-xr-x   17 birhanu  staff    544 Sep 25  2018 dsm-theia1
drwxr-xr-x  106 birhanu  staff   3392 May 10 12:07 eng1
drwxr-xr-x   11 birhanu  staff    352 Sep 25  2018 eng1-achrive
drwxr-xr-x   24 birhanu  staff    768 Sep 25  2018 eng2
drwxr-xr-x    8 birhanu  staff    256 Sep 25  2018 engagement-scenarios
-rw-r--r--    1 birhanu  staff      0 Sep 25  2018 file-name.pdf
drwxr-xr-x   15 birhanu  staff    480 Sep 25  2018 platform
-rw-r--r--    1 birhanu  staff    322 Sep 25  2018 plot-dot.py
-rw-r--r--    1 birhanu  staff    816 Sep 25  2018 plot1.py
drwxr-xr-x    6 birhanu  staff    192 Sep 25  2018 propatrol
drwxr-xr-x    9 birhanu  staff    288 Sep 25  2018 sbu-src
```

UNIX Examples of DAC

- chmod a+r file1.txt
- chmod a-r file1.txt
- chmod a-rwx file1.txt
- chmod g+rw file1.txt
- chmod u+rwx file1.txt
- chmod og+rw file1.txt

- readable by all
- cancels read right for all
- cancels all access for all
- give the group read and write permission
- give the owner all permissions
- give the world and the group read and write permission

UNIX Examples of DAC …

- 0: no permission
- 1: x
- 2: w
- 3: wx
- 4: r
- 5: rx
- 6: rw
- 7: rwx

- chmod 444 file1.txt
- chmod 555 file1.txt
- chmod 760 file1.txt
- chmod 700 file1.txt
- chmod 766 file1.txt
- chmod 777 file1.txt

Mandatory Access Control (MAC)

- User can't change any object access control policies
- System owner configures policies of all objects in the system
- More restrictive (hence more secure) than DAC
- But, obviously more rigid

Security Levels

- Organizations have hierarchical relationship between security sensitivity of digital assets
- One file might have the highest security sensitivity
  - Office environments: memos, reports, customer lists, backup data
  - Defined sensitivity and importance

Security Levels

Levels
Top Secret (TS)
Secret (S)
Confidential (C)
Unclassified (U)

Subjects
Thomas, Tony
Sam, Sally
Claire, Carla
Ursula, Ugo

Objects
Personnel Files
Email Files
Server Log Files
Telephone Directory

- Clair's security clearance is C, and Sam's is S
- Email Files' classification is S, and Personnel Files' is TS
- Note: for subjects:clearance, for objects: classification

Notation

- $L(S) = I_s$: security clearance of subject S
- $L(O) = I_O$: security classification of object O

- For all classifications I_i, i in $[0,K-1]$: $I_i < I_{i+1}$
  - Example: for k = 4, i in [0,3]:

$li : (l0 = \text{U}) < (l1 = \text{C}) < (l2 = \text{S}) < (l3 = \text{TS})$

Security Conditions

- Simple-Property (focus: read):
  - S can read O iff $I_o \leq I_s$ and S has discretionary read access to O
- *-Property (focus: write):
  - S can write O iff $I_s \leq I_o$ and S has discretionary write access to O

Examples on Conditions

Levels
Top Secret (TS)
Secret (S)
Confidential (C)
Unclassified (U)

Subjects
Thomas, Tony
Sam, Sally
Claire, Carla
Ursula, Ugo

Objects
Personnel Files
Email Files
Server Log Files
Telephone # Files

- S.read(*O*):
  - -iff $I_o \leq I_s$
- S.write(*O*):
  - -iff $I_s \leq I_o$

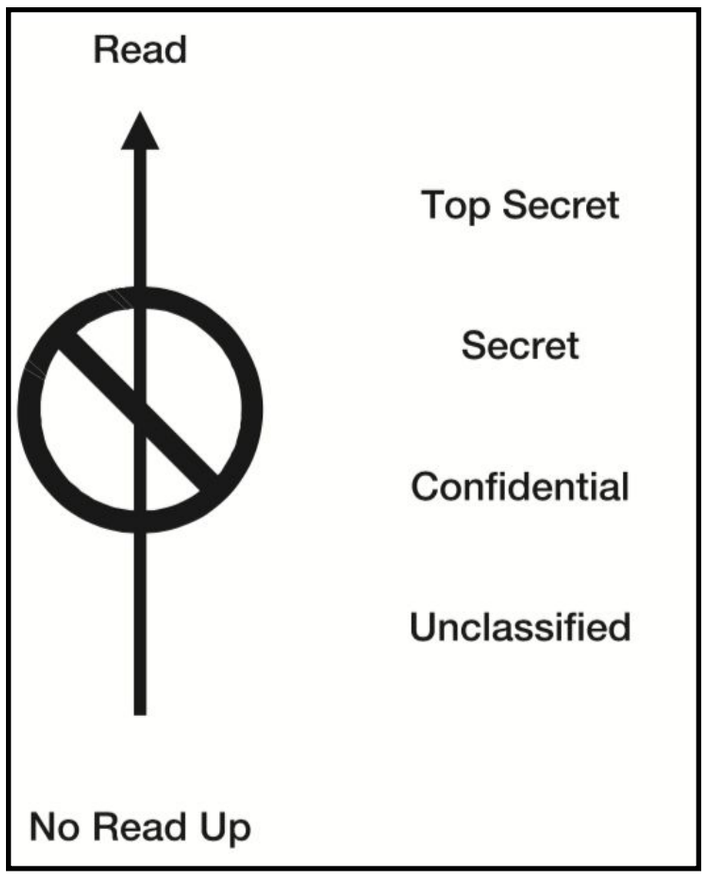
- Clair and Carla: read(Personnel Files) : NO!
- Thomas and Sally: read (Server Log Files): YES
- Tony: read (Personnel Files): YES
- Tony: write (Server Log Files): NO!

The Bell-LaPadula Model

- Widely used by governments & military
- Subjects are given clearance levels
- Objects are associated with a classification
- Three properties govern access control:
  - The Simple Property
  - The Star Property
  - The Tranquility Property

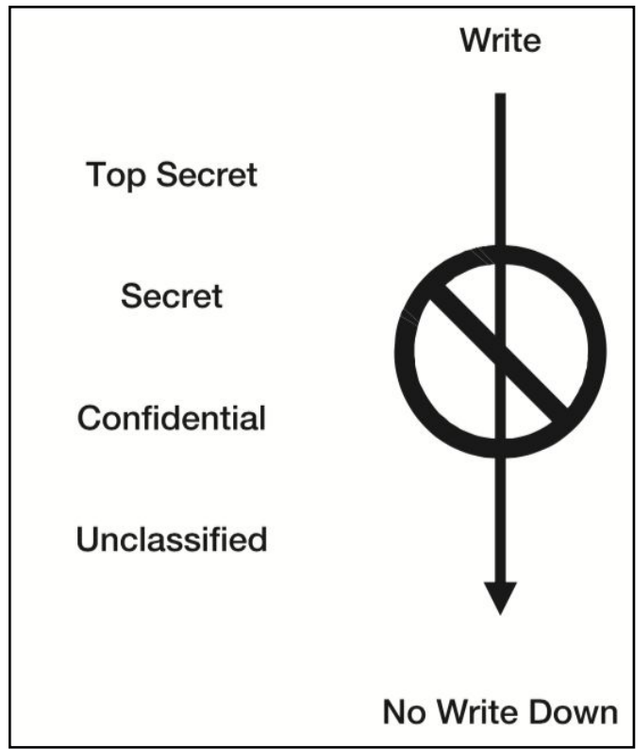
The Simple Property

- Also called the no read up rule
- Idea: a subject with a certain level of access (e.g., unclassified) is not allowed to read objects with higher classification (e.g., confidential)
- Goal: subjects can only read objects at their access level or below



The Star Property

- Also called the no write down rule
- Idea: a subject with a certain access level (e.g., secret) is not allowed to write any object that has a lower level of access (e.g., confidential)
- Goal: we would not want any information to leak from a higher level to a lower level



The Tranquility Property

- Idea: the classification of an object cannot be changed while the object is in use by any subject of the system

- Goal: to avoid accidental declassification of confidential objects while someone with confidential access is still writing to an object

- A synchronization constraint placed upon the objects in a system

Example: Read

- Users: S1(C) , S2 (U), S3 (C), S4 (S), S5 (TS)
- Files: F1, F2, F3, F4, F5
- Note: F1-F5 are created by S1-S5 respectively
- Question: allow/deny for a read attempt

	F1	F2	F3	F4	F5
S1	A	A	A	D	D
S2	D	A	D	D	D
S3	A	A	A	D	D
S4	A	A	A	A	D
S5	A	A	A	A	A

Example: Write

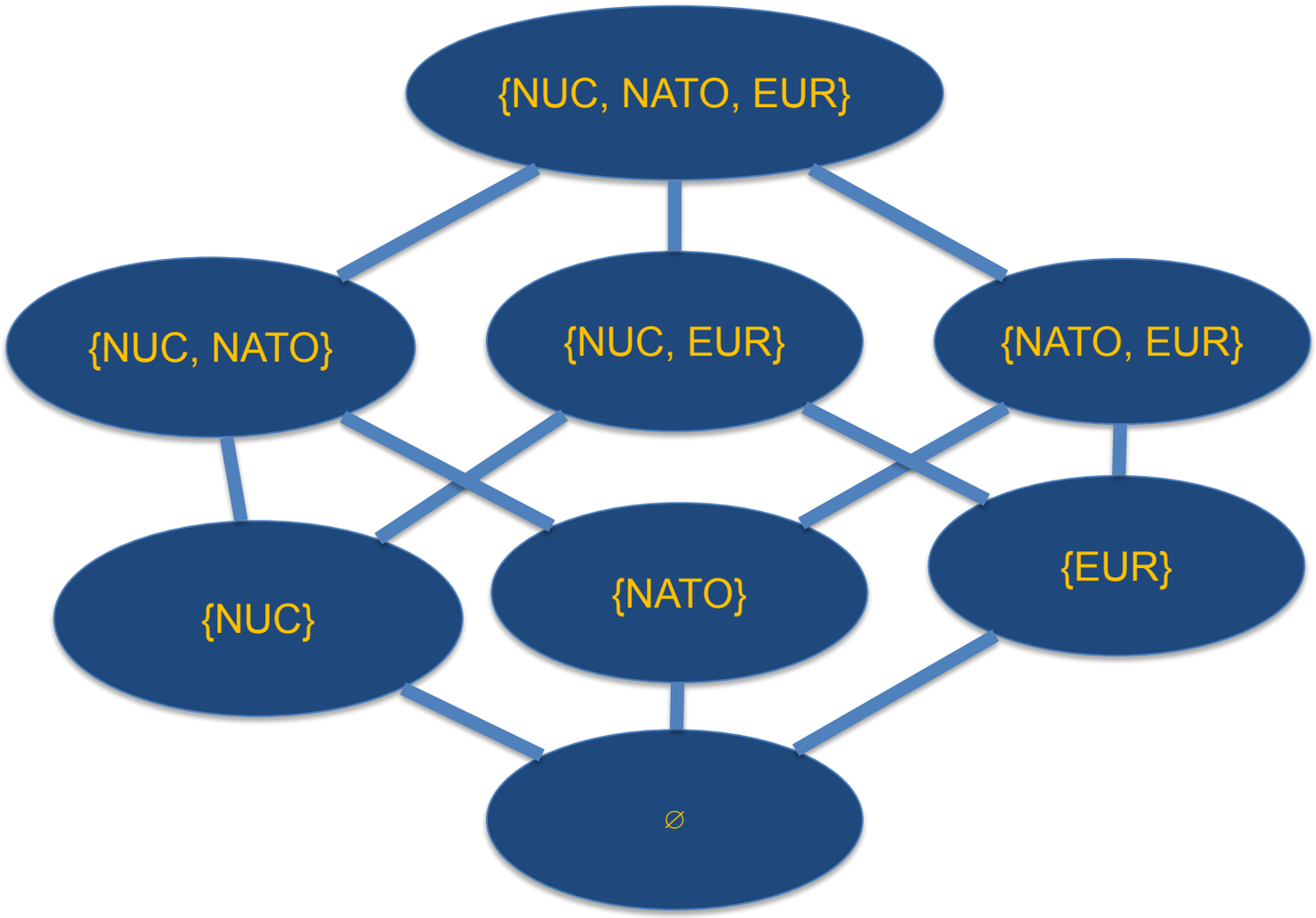
- Users: S1(C) , S2 (U), S3 (C), S4 (S), S5 (TS)
- Files: F1, F2, F3, F4, F5
- Note: F1-F5 are created by S1-S5 respectively
- Question: allow/deny for a write attempt

	F1	F2	F3	F4	F5
S1	A	D	A	A	A
S2	A	A	A	A	A
S3	A	D	A	A	A
S4	D	D	D	A	A
S5	D	D	D	D	A

Expanding BLP with Categories

- Categories arise from the "need to know" principle
- Need to know: no subject should be able to read objects unless reading them is necessary for that subject to perform its functions
- Sets of categories to which a subject may have access = the power set of the set of categories
- Example: given categories NUC, EUR, and NATO, the set of categories: ∅ (none), {NUC}, {EUR}, {NATO}, {NUC, EUR}, {NUC, NATO}, {EUR, NATO}, and {NUC, EUR, NATO}

Security Lattice

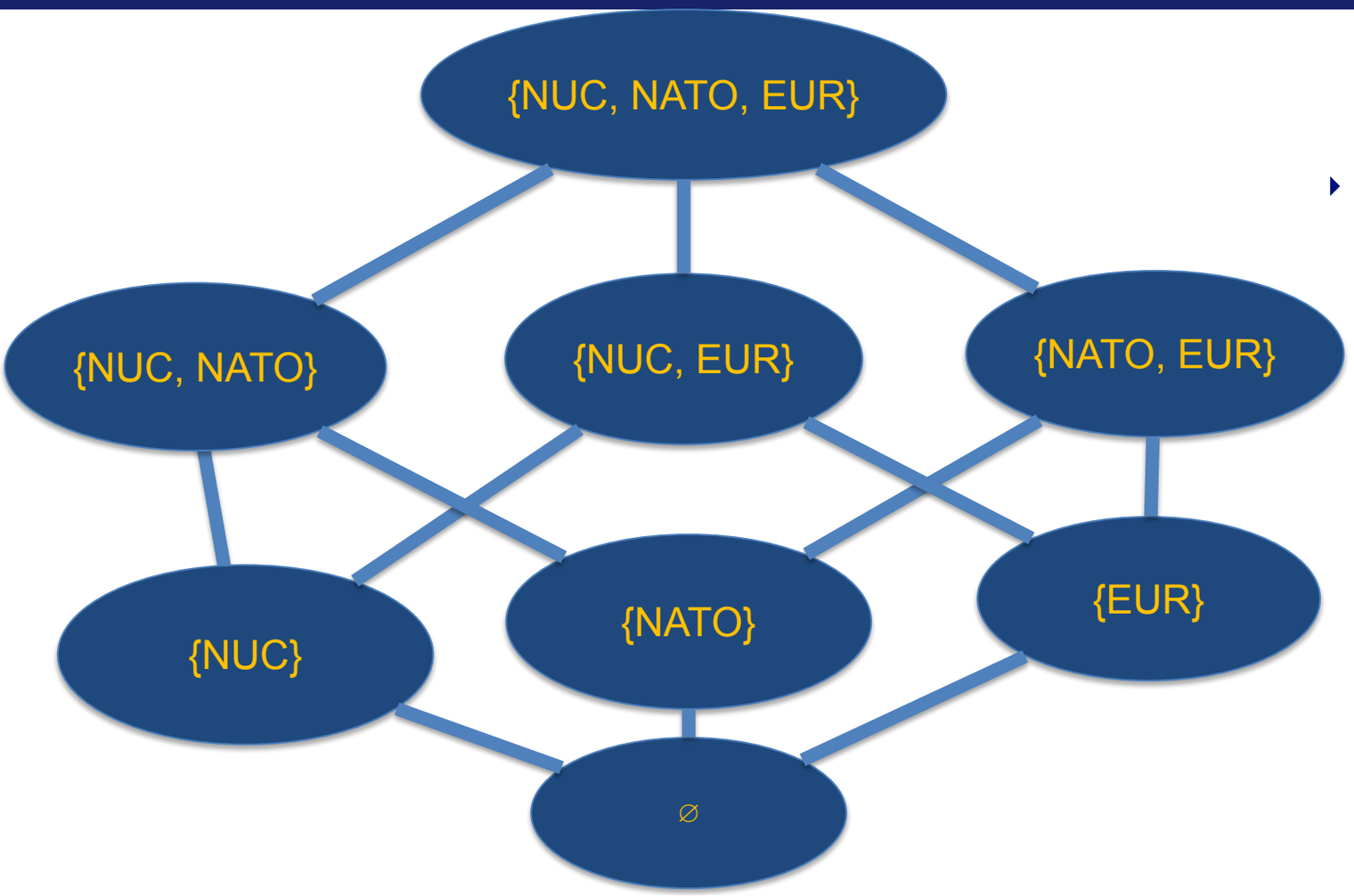


- The sets of categories form a lattice under the operation ⊆ (subset of)

Combining Levels and Categories

- **Dominates Property:**
  - (*L*,*C*) dominates (*L'*,*C'*) iff *L'* ≤ *L* and *C'* ⊆ *C*
- **Simple Property Revisited:**
  - *S* can read *O* iff *S* dom *O*
- **Star Property Revisited:**
  - *S* can write to *O* iff *O* dom *S*

Read/Write Attempts



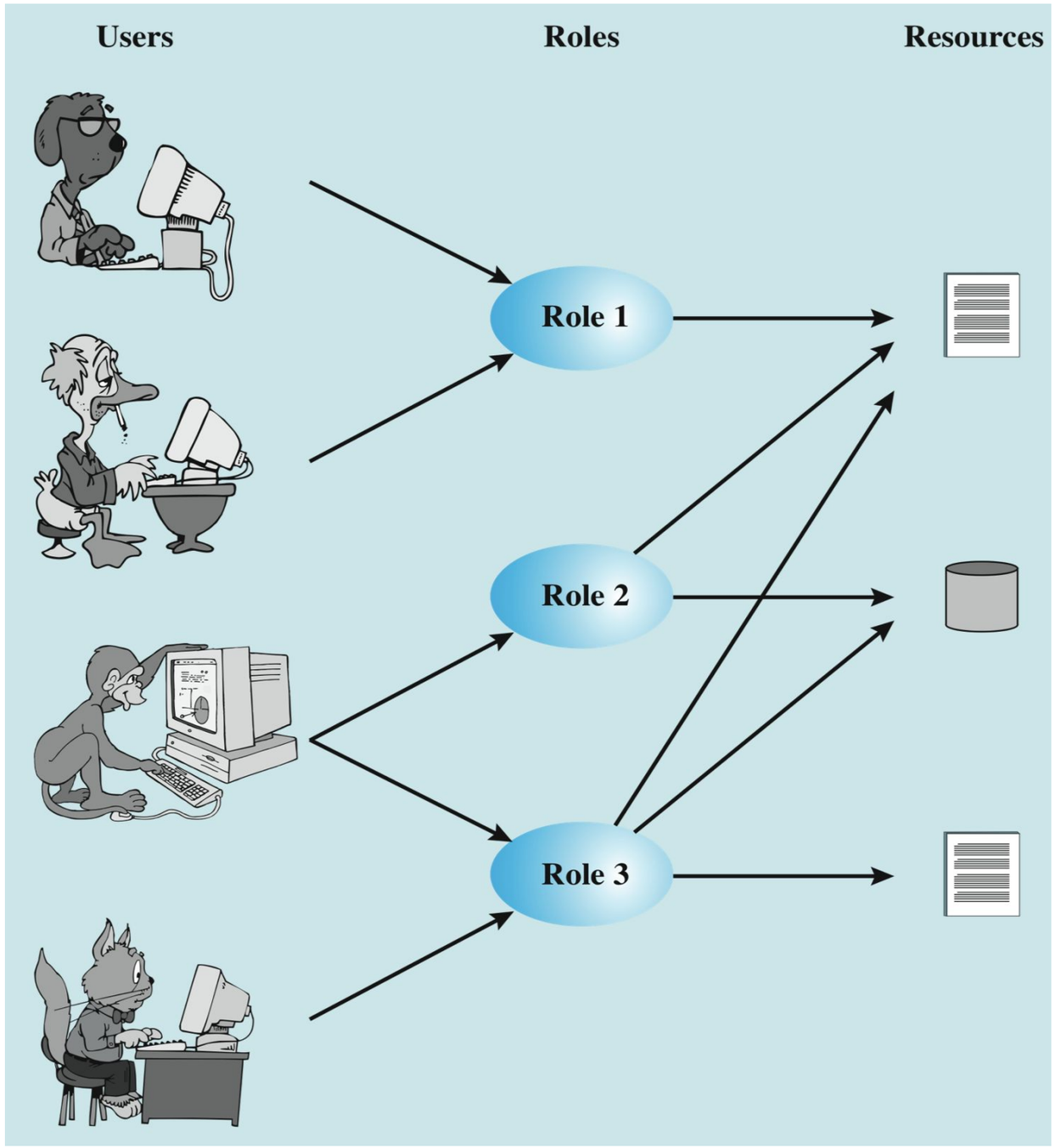
- **Dominates Property:**
  - (L,C) dominates (L',C') iff L' ≤ L and C' ⊆ C
- **Simple Property Revisited:**
  - S can read O iff S dom O
- **Star Property Revisited:**
  - S can write to O iff O dom S

- Given: William: (TS, {EUR}), George: (S, {NATO, EUR})
  - William.read (S, ∅)
  - William.write (S, {EUR})
  - William.read (TS, {NATO, EUR})
  - William.write (TS, {EUR, NATO})
  - George.write (S, {NATO})
  - George.read (TS, {NATO, EUR})
  - George.read (S, {EUR, NUC})
  - George.write (U, ∅)
  - George.write(U, {EUR})

Role-Based Access Control (RBAC)

- Rather than identity (DAC) or clearance (MAC), subject's permissions are determined by the user's role
- More natural expression of business logic
- Role: logical grouping of one or more users that have some common affiliations (e.g., same department, grade, age, physical location, or user type)

Typical RBAC Setup



	R_1	R_2	...	R_n
U_1	X			
U_2	X			
U_3		X		X
U_4				X
U_5				X
U_6				X
...				
U_m	X			

	OBJECTS								
ROLES	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
...									
R_n			control		write	stop			

RBAC Example

Activity	Default Access
Read	Granted
Write	Deny
Sign	Deny

- Subject (s): Bob
  - -role (clerk)
  - -group (courthouse)
- Verb (activity): sign
  - -Default: Deny
- Object: tax-doc

Access rule for tax-doc:

sign: ‘clerk’ in s.role and ‘courthouse’ in s.group and 0800 <= hour <= 1700 and “Monday” <= day <= “Friday”

Possible policy: ∀s ∈ Subjects, t ∈ Times, d ∈ Days, sign(s)<=>(role(s) = clerk and group(s) = courthouse) and (0800 <= t <=1700) and d ∈ {M, T, W, Th, F}

Allow/Deny Based on Example

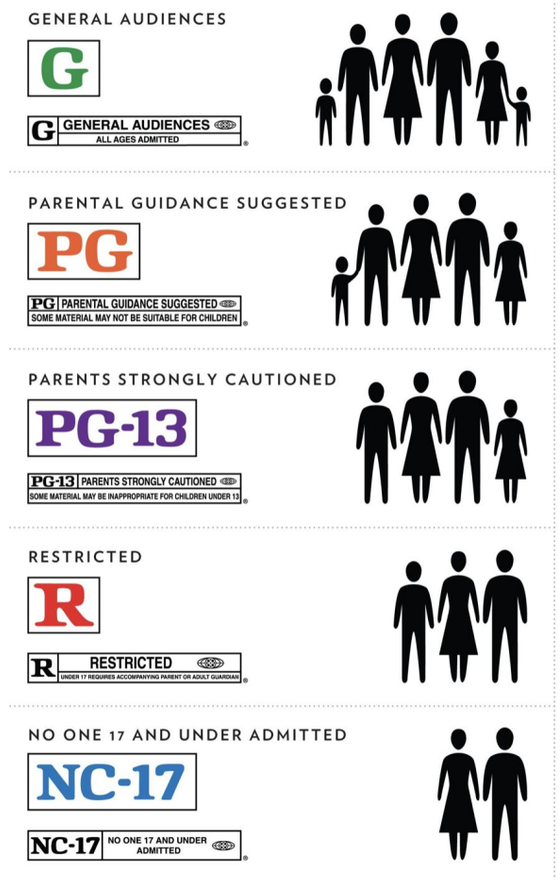
- Bob attempts to:
  - Read tax-doc at 1am on Monday
  - Read and sign tax-doc at 12pm on Wednesday
  - Write and sign tax-doc at 3pm on Thursday
  - Read tax-doc at 5pm on Saturday

Attribute-Based Access Control (ABAC)

- Users have attributes (age, ID number, group membership, etc.)
- Objects have attributes (e.g., movie title, rating, release date, etc.)
- Preferred for fine-grained access control
- Policy is a complex Boolean expression on the attributes of subjects and/or objects

Typical ABAC Implementation

- Example: online movie viewing service
- Basic policy: access to a movie will be granted based on age of user and rating of the movie
- Precise policy: children will be allowed to watch movie with G rating



Access Rule (R1):

can_access(u,m):
(u.age>=21 && m.rating in ['R', 'PG13', 'G']) or
(13<=u.age<21 && m.rating in ['PG13','G']) or
(u.age<13 && m.rating in ['G'])

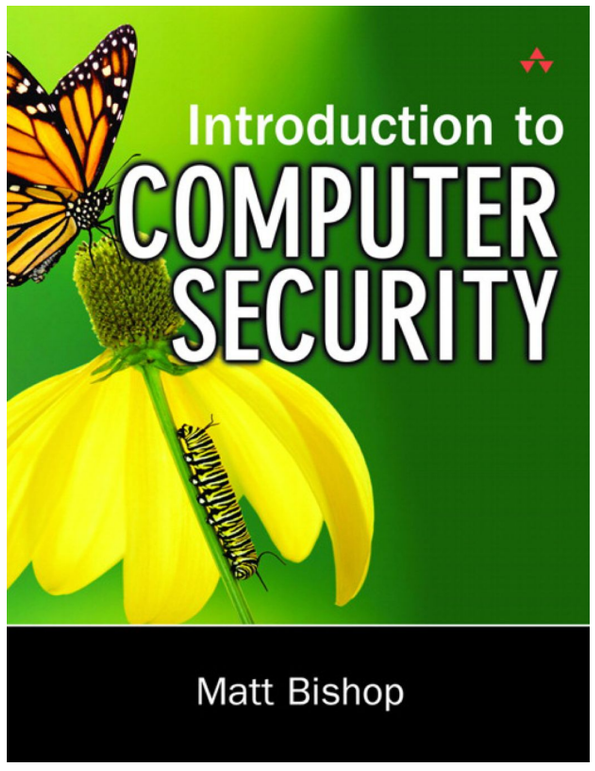
Access Control Models: Pros and Cons

- DAC: easy to implement, highly flexible; doesn't scale well, potential ACL explosion, prone to mistakes
- MAC: most secure, easy to scale; not flexible, limited user functionality, high admin overhead
- RBAC: scalable, flexible, less admin overhead; role needs provisioning and maintenance, potential role explosion, unable to accommodate real-time context
- ABAC: dynamic, contextual, fine-grained; complex to analyze, potential attribute explosion

Lecture Summary

- Access Control: mechanism to enforce authorization policies
- ACM and its issues (scalability, efficiency)
- ACL and CAP: per-object vs. per-subject
- DAC and MAC: owner's discretion vs. systems' constraints
- Security Levels: clearance (subj), classification (obj)
- Security properties: simple, star
- The Bell-LaPadulla model: no read up, no write down, tranquility
- Bell-LaPadulla extended with categories: security lattice
- RBAC, ABAC and pros and cons

Further Reading



Chapters 2, 5, 14