



Access Control Theory

...boring part with some maths which will however
“outlast current technology”...

Nicolas T. Courtois

– University College London

Roadmap

- Policies and Mechanisms
- Set models,
- Maths: Relations, Bounds, Lattices
- DAC, [Matrix model](#), [HRU Model](#)
- DAC in practical OS
- The safety problem, undecidability
- MAC, Multi-level security
- Product lattices, [Bell LaPadula = BLP](#) model
- [Biba](#) model,
- [Clark-Wilson](#) model
- [Chinese Wall](#) model
- RBAC, conflict resolution

Can We Help?



insecure rubbish!

vs.



Science

Is There a Need For Access Control ?

The problem of access control remains largely unsolved. And seems almost unsolvable,

- OS+add-on security all-in-one security packages will
 - either decide everything for you,
 - or leave the customer with choices that nobody understands

Policies



Policy:

Meaning we want to use:

Policy, is what we want.

How things should be.


A Security Policy:

Short, succinct statement. **High-level**.


Describes what is and what is not allowed.

Security and protection requirements, rules, and goals.

It defines what it means to be “secure”
for a system or organisation/entity.



Here, it usually
means a set of
requirements.



Here, it means
usually a set of
behaviour rules to
obey.

** “Information Security Policy”

A bit different usage:

Usually refers to a plan / document

on how an organisation is going to comply with laws regarding personal/financial/customer data protection...

Example:

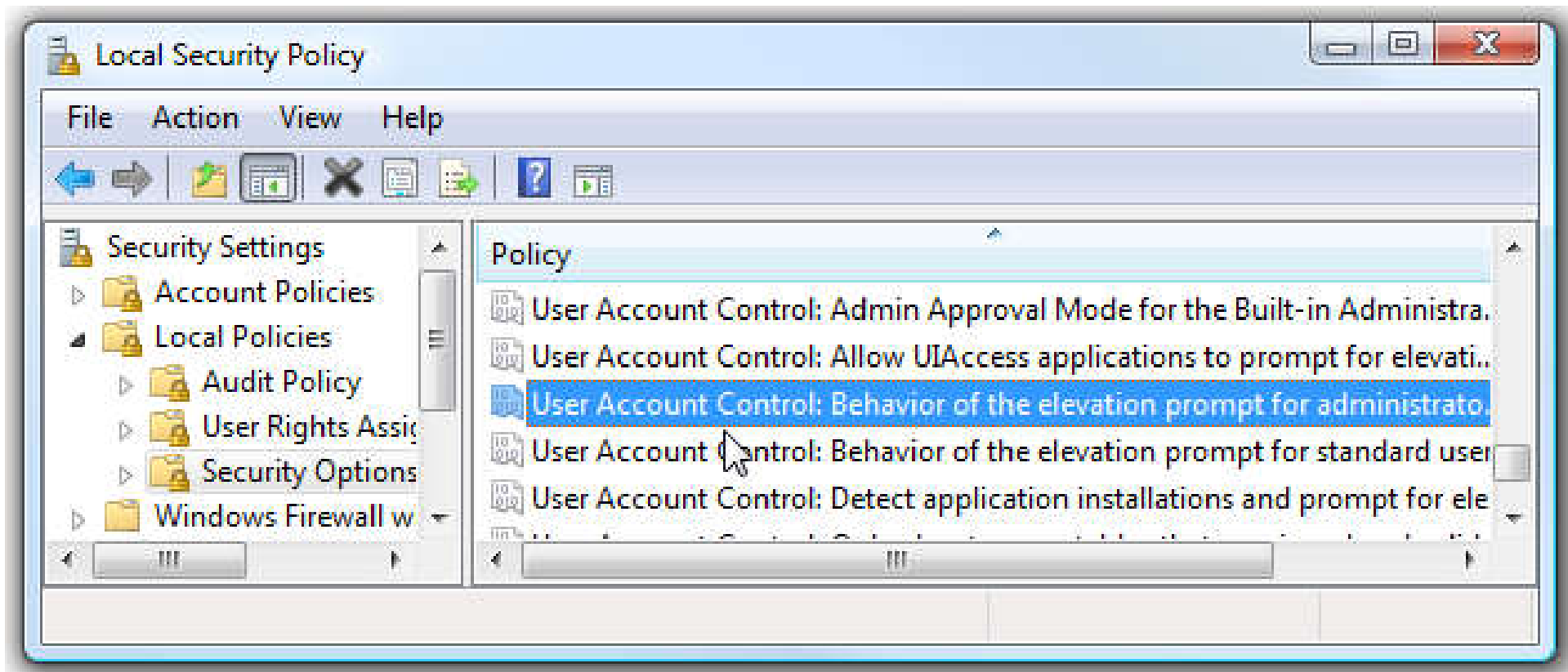
EU Directive 95/46/EC about data protection [1995].

UK Data Protection Act [1998].

- Data may only be used for the specific purposes for which it was collected.
- Data must not be disclosed to other parties without the consent of the individual
- Individuals have a right of access to the information held about them -exceptions.
- Personal information may be kept for no longer than is necessary
- Data must be kept up to date. Right to rectify it.

W10 has a “Local Security Policy”

Can be edited



Mechanisms



Policies and Mechanisms

Mechanisms are there to enforce policies.

- various sorts of mechanisms, HW, SW, crypto, and combinations...
- A policy can be implemented in several different ways, relying on different mechanisms.
- Mechanisms can be built into “control systems” (e.g. access control systems) capable of enforcing several policies, or flexible policies.

Some Maths



Relations

set of all ordered
pairs a_1, a_2

Let A be a set. We call relation any subset $R \subseteq A \times A$.

We write things such as:

$a R b$ which reads

a is “in relation R ” to b

Example of a Relation

Let $a, b \in \mathbb{N}$

Definition: $a \mid b$

iff

$\exists x \in \mathbb{N}$ such that $ax=b$.

Relations

Sub-categories:

- equivalence relations,
- order relations (orderings),
- etc.

***Equivalence

Equivalence Relation:

1. Reflexive: $a \equiv a$
2. Symmetric $a \equiv b$ if and only if $b \equiv a$
3. Transitive $a \equiv b$ and $b \equiv c$ implies $a \equiv c$.

Order Relations

Order:

1. Reflexive: $a \leq a$
2. Antisymmetric: if $a \leq b$ and $b \leq a$ then $a = b$.
3. Transitive $a \leq b$ and $b \leq c$ implies $a \leq c$.

Partial ordering:

For any couple a, b we have either

$a \leq b$ or $b \leq a$ or neither –

when we say that “ a and b are unrelated”.

Total ordering (= linear order = simple order = chain):

4. For any couple a, b we have either
 $a \leq b$ or $b \leq a$.

all pairs are related =
mutually comparable

POSET = Partially Ordered Set

A set A and an order relation \leq .

Poset is the couple (A, \leq) .

Maths view:

we write formulas on the board and we use axioms 123 on the last slide to prove theorems.

Pragmatic computational functional view of a relation:

we have

objects $a \in A$

data type A

2-ary function called $\leq: A \times A \rightarrow \{\text{True}, \text{False}\}$.

Example of a POSET

Let $a, b \in \mathbb{N}$

Definition: $a \mid b$ means $\exists x$ such that $ax=b$.

(\mathbb{N}, \mid) is a poset

- Reflexive: $a \mid a$
- Antisymmetric: if $a \mid b$ and $b \mid a$ then $a = b$.
- Transitive $a \mid b$ and $b \mid c$ implies $a \mid c$.

Proof:

But not a total order:

Prove it:

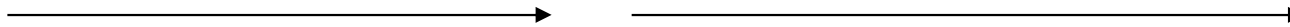
Applications

Order relations are useful in formalising and analysing security...

Bounds

Exist for both total and partial orders.

Total orders are simple in sense they are “one-dimensional”. Like a straight line...



Partial orders describe much more complex situations....

Bounds

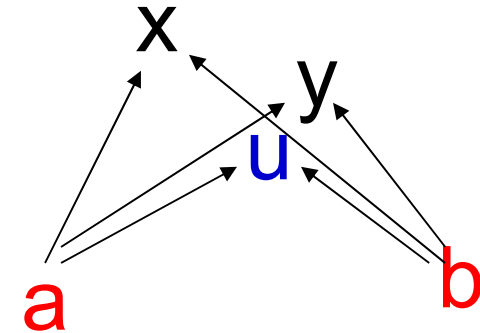
Definition:

u is an upper bound for a and b iff
 $a \leq u$ and $b \leq u$.

Definition:

u is an lower bound for a and b iff
 $u \leq a$ and $u \leq b$.

LUB = Least Upper Bound = Supremum = Sup = Join



Definition:

Let U be the set of all upper bounds for a and b .

Let u be the smallest element in U ,

which means $\forall x \in U$ we have $u \leq x$.

Then u is called the **Least Upper Bound** of a and b .

We write:

$$u = a \vee b$$

and say “least upper bound for a and b ”

or “ a Vee b ”

In LaTeX $\text{\texttt{\textbackslash vee}}$

LUB = Least Upper Bound = Supremum = Sup = Join

$$a \vee b$$

and we have the dual concept:

GLB = Greatest Lower Bound = Infimum = Inf = Meet

$$a \wedge b$$

defined in the same way...

BTW. we say “greatest lower bound for a and b”
or “a Wedge b”

In LaTeX `/wedge`

Example

Claim 1: NI, \leq is a total ordering.

Proof: check 123+total

Important

Bounds do NOT have to exist.

Least upper bounds don't have to exist either.

Lattices



Lattices

Definition:

An ordered set S, \leq is called a lattice if:

- $\forall a, b$ the LUB $a \vee b$ exists.
- $\forall a, b$ the GLB $a \wedge b$ exists.

Corollary 1: every finite subset has a SUP and an INF.

Corollary 2: In every finite lattice we have two special elements called **top** \top and **bottom** \perp .

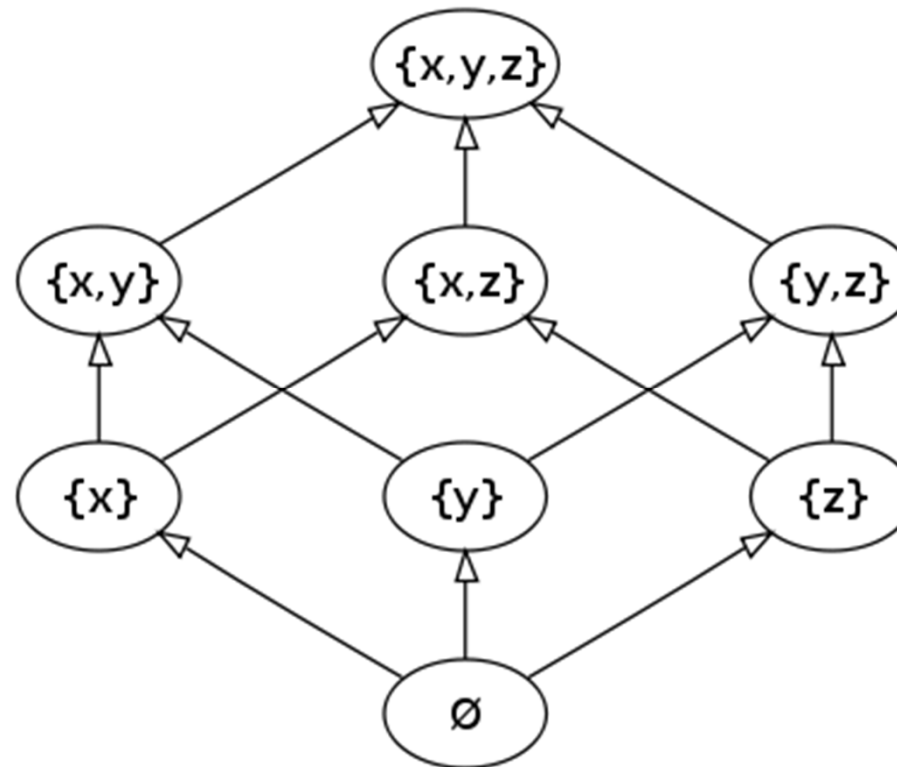
Lattices

$\forall a, b:$

- $a \vee b$ exists.
- $a \wedge b$ exists.

Example 1: For any set P , we call the “Power of P ” and denote 2^P the set of all subsets of P . Then $2^P, \subseteq$ forms a lattice.

“Hasse Diagrams” – For All Lattices



Lattices

Example 2: \mathbb{N} , $|$ is a lattice.

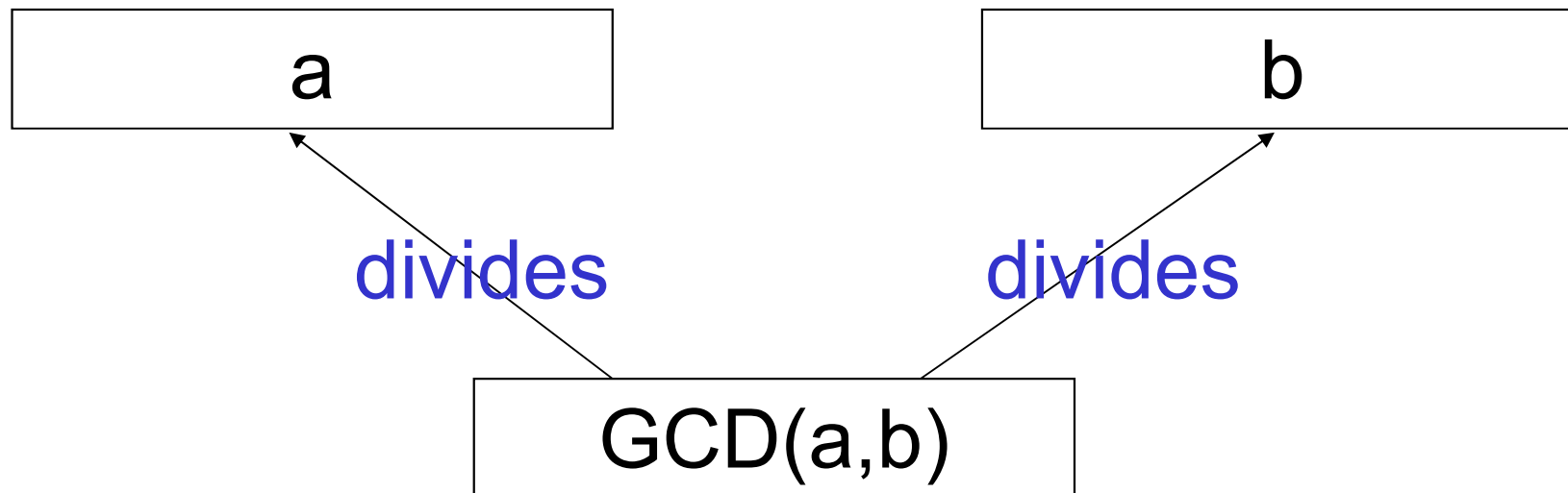
2 and 3 cannot be compared.

$$2 \vee 3 = ?$$

$$2 \wedge 3 = ?$$

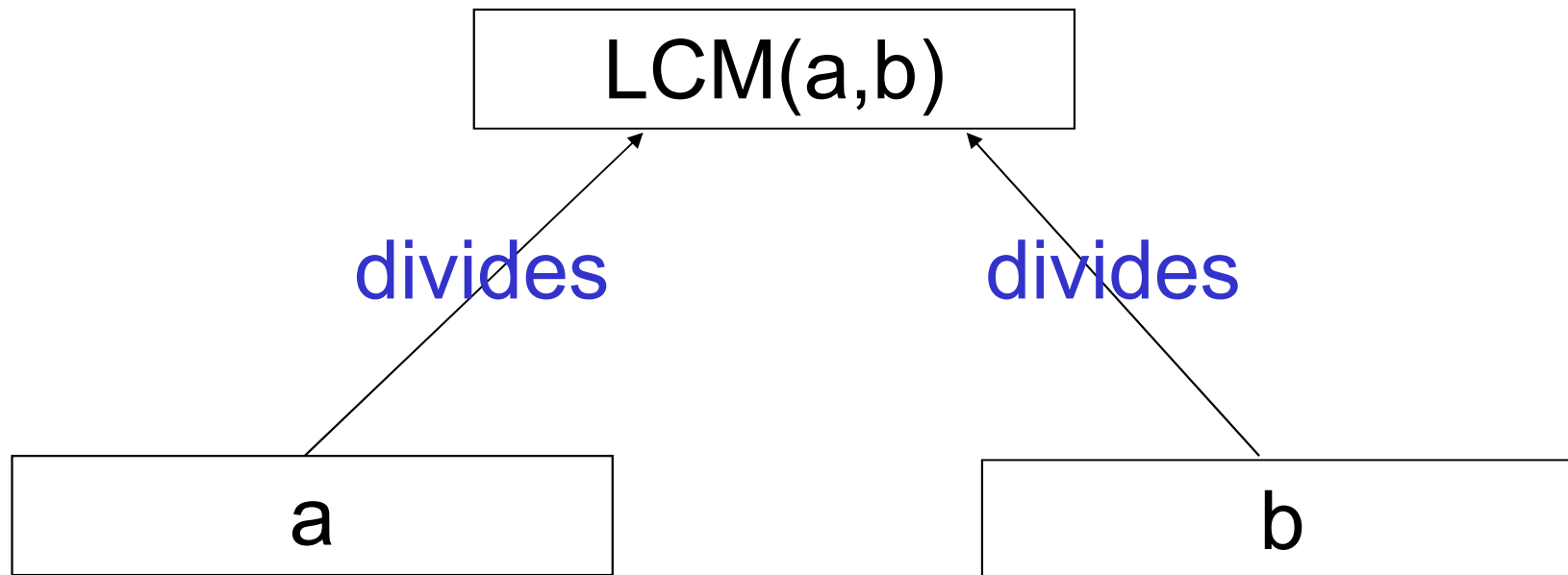
Greatest Common Divisor

the biggest such that



Least Common Multiple

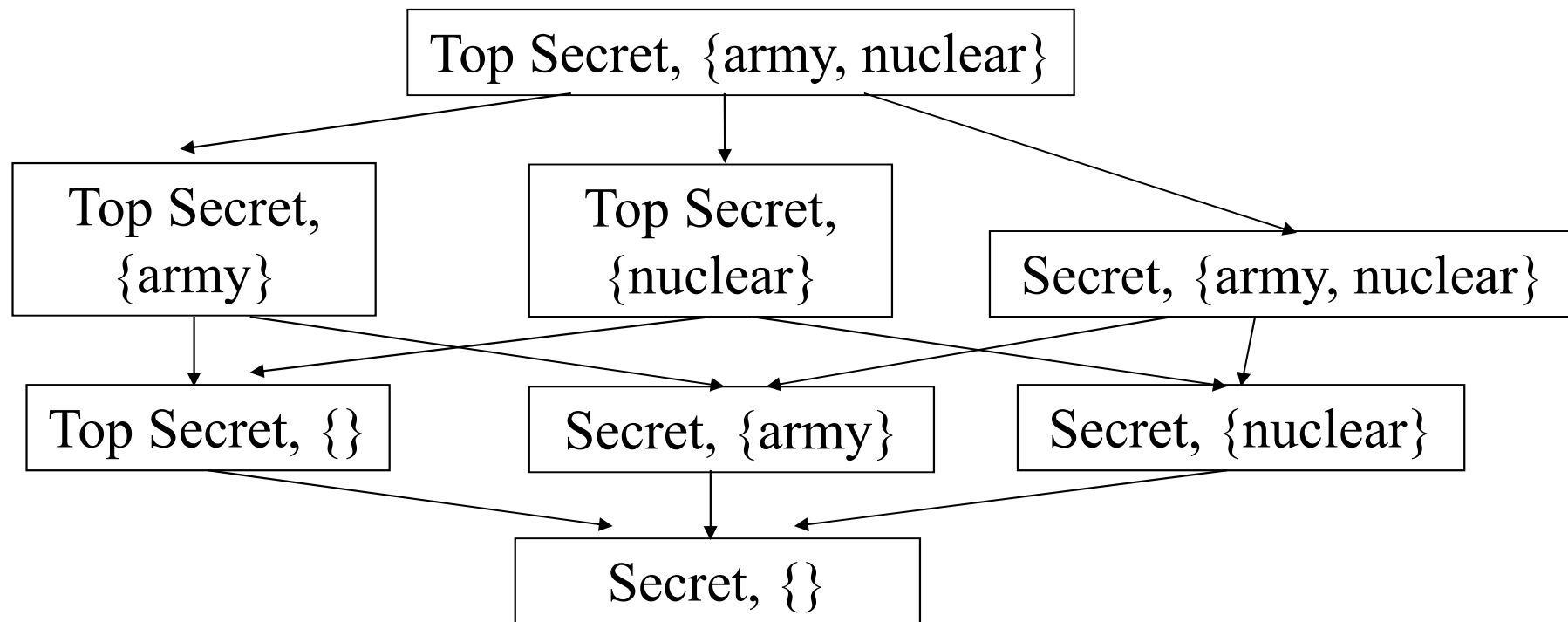
the smallest such that



Lattices

Example 3: Any total ordering.

More “Hasse Diagrams” Later...



File Access Control



Example of a Security Policy

No user should be able to access
other user's files.

Benefits:

- Accountability
- Trace-ability
- Confidentiality, Privacy



Two methods to implement this, can be combined:

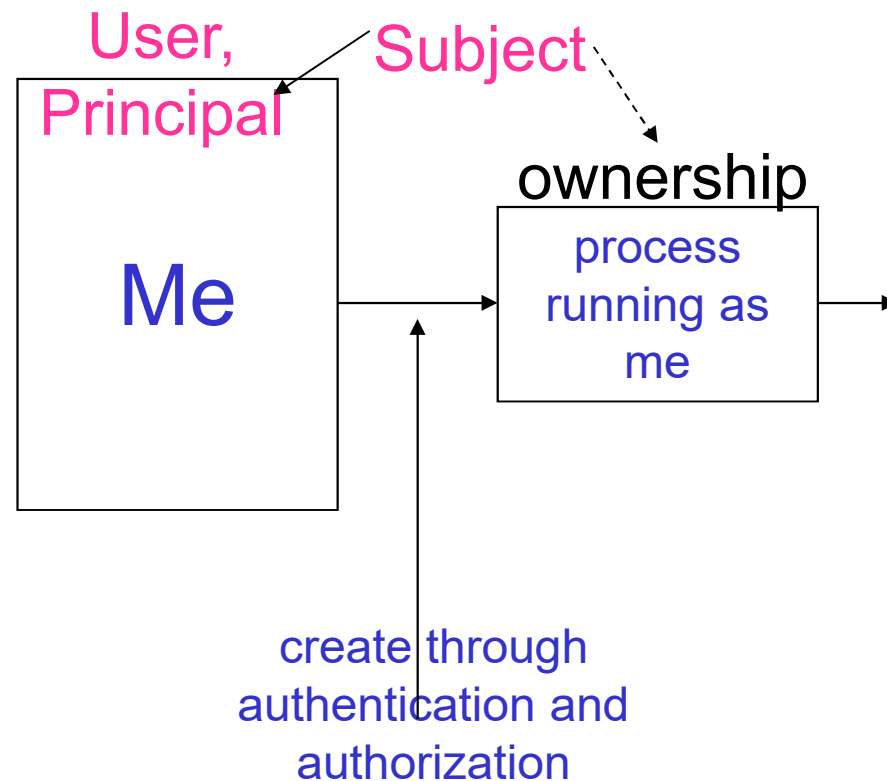
↓ 1. Follow the people:

authentication, authorization.

2. Follow the data:

information flow control.

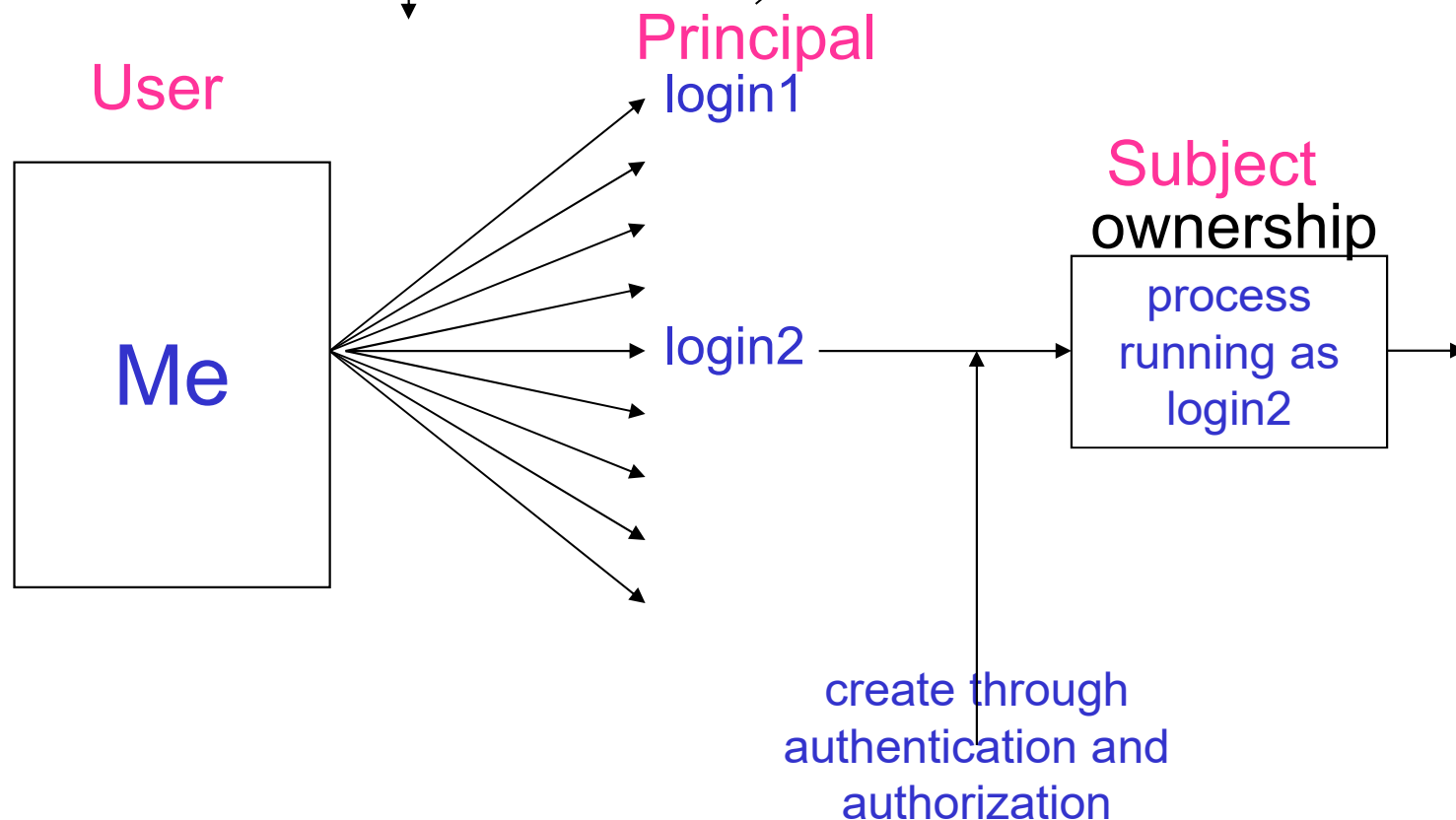
Users, Subjects, Principals



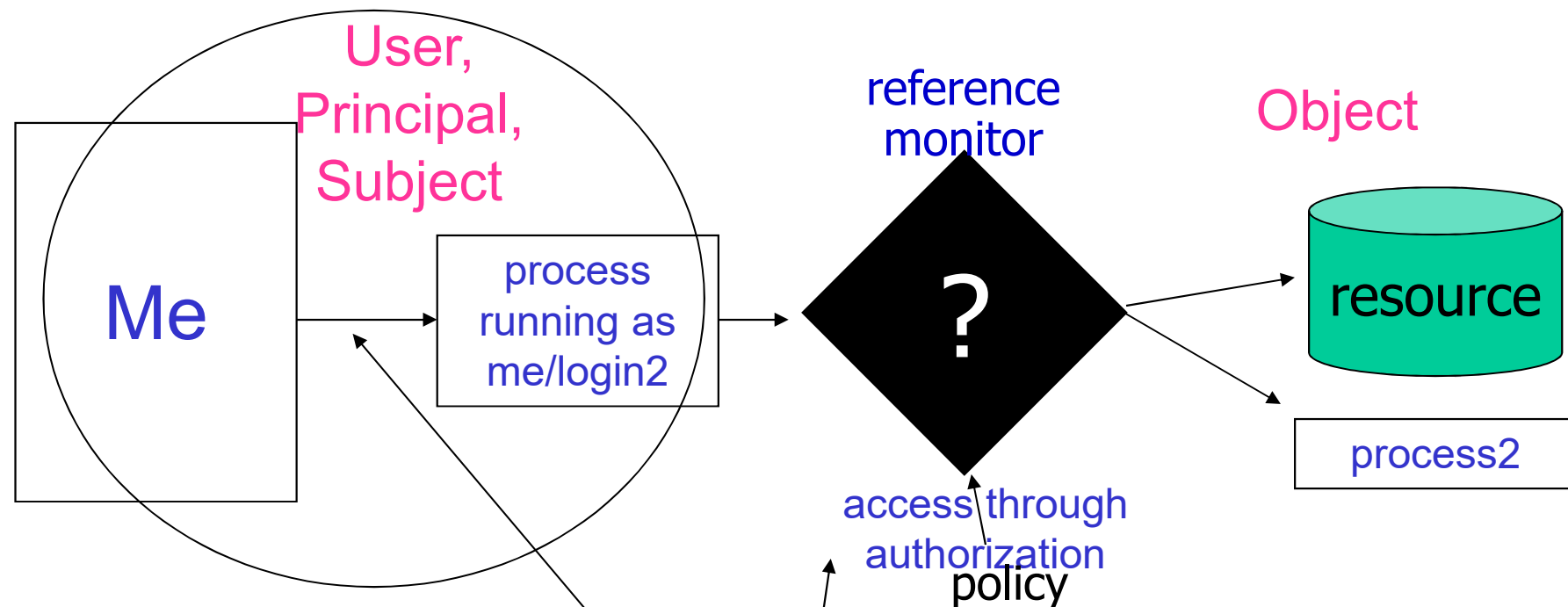
Distinction Users vs. Principals

One to Many.

= def: Unit of Access Control
and Authorization



Subjects and Objects



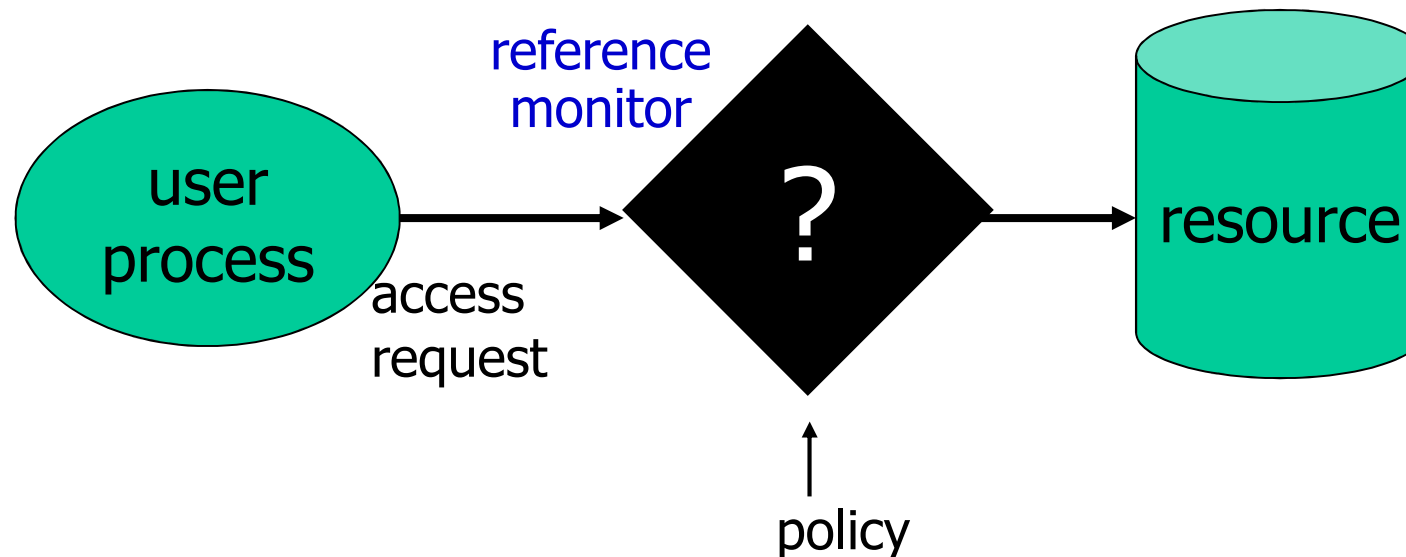
access control
occurs at 2 moments!

In Unix processes are both subjects and objects, we can execute operations on processes: kill, suspend, resume..

Reference Monitor

Def:

module that **controls** all software access to data objects or devices.



exists since Windows NT (XP,Vista).

Reference Monitor - @Exam!

Must be:

1. tamperproof,
2. always-invoked = non-bypassable =
a.k.a. complete mediation
3. economical, simple
 - small enough to be build in a rigorous way,
 - and fully tested and analysed

Windows: exists since Windows NT.

Technical Difficulties

- Residue Channels
 - Inadvertent or built-in duplication/storage of information.
 - need to actively clean disk sectors, memory, CPU cache etc.
- Covert Channels
 - information is leaking
 - intentional or not (side channels).

Access Control Models

Benefits:

We can formally prove security properties of a system.
Derived from basic premises.

Nice split between conceptual and practical security:

- Prove that model is “secure”.
- And that the implementation is correct.

Allows to claim that security is achieved.

- And if it isn't, we should be able to blame EITHER the model
OR the implementation,
without any ambiguity.

3 Main Paradigms of Access Control

Discretionary Access Control (DAC)

- Owners decide about rules, at their discretion, can pass rights on others

Mandatory Access Control (MAC)

- System-wide policy, possibly denying users full control over the access to resources they created

frequently
combined

Role-Based Access Control (RBAC)

Two levels

In most policies,

- except in pure Mandatory AC policies.

Two main levels:

- Access Control Policy.
 - who can access the resources?
- Administrative Policy.
 - who can specify rules and authorizations?

And big problem: things change. Ownership can be changed. Permissions can be changed.

Discretionary Access Control



What is DAC?

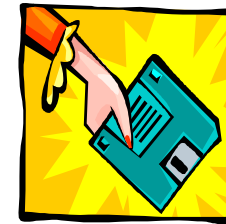
DAC policies are a family of access control policies such that:

1. They enforce the access to files on the basis of
 - identity of the requestors
 - or their verifiable properties
 - a set of explicit access rules:
 - who can access what
2. In addition, files have owners
 - “Discretionary” meaning:
 - an ability to grant/revoke rights for others (for the owner)

Matrix Paradigm [Lampson, Graham-Denning, Harrison-Ruzzo-Ullmann]

A way to describe mathematically access conditions in many real systems.

- A set **S** of Subjects (Principals).
- A set **O** of Objects (e.g. files).
- A set **A** of Operations.



Example: $A = \{\text{exec}, \text{read}, \text{append}, \text{write}\}$.

An access control matrix.

$$M = (M_{so})_{s \in S, o \in O}$$

Where each entry $M_{so} \subseteq A$.

Matrix - Example

Example:

$S = \{\text{System, Admin, Bob}\}.$

$O = \{\text{exe, doc}\}.$


$A = \{\text{read, write, exec, delete}\}.$

$M =$

		Objects →		rights
Subjects ↓		exe	doc	
	System	{e, r, w, d}	{r, w, d}	
	Admin	{e, w, d}	{w, r, d}	
	Bob	{e}	{r, w}	

50

Nicolas T. Courtois, 2009-2016



Examples: Standard File Systems



Simple Example - Unix

$S = \{\text{Process1}, \text{Process2}; \text{User1}\}.$

$O = \{\text{file1}, \text{file2}; \text{directory3}; \text{process5}; \text{device6}\}.$

$A = \{\text{read}, \text{write}, \text{exec}\}.$ Octal: $r=4$ $w=2$ $e=1$, e.g. 775.

`ls -l` \Rightarrow `-rwx-r-x---`

these are the famous “9 bits”, **user,group,other**

1. For directories, already quite special:
 - read – means list files
 - write – means add/remove files and subdirs
 - exe – means one can CD to that dir, and maybe access any file the name of which is known (and others with read right).
2. For a process:
 - read – receive signals
 - write – send signals
 - exe – be able to execute as a sub-process
3. For devices and named pipes:
 - treated as files for read/write
 - Exe has no meaning.

Windows NT

In comparison - excessively complex:

Was developed much more recently!

All depends if your OS supports NTFS,
the Microsoft file system designed to work in multi-user
environments.

NTFS is only present in NT and in XP and later.

DAC in Practice



Matrices - Implementation

Matrix storage: waste of space, not very practical.

- Authorization table – sparse matrix kind
- Capabilities - rows
- Access Control Lists (ACL) - columns

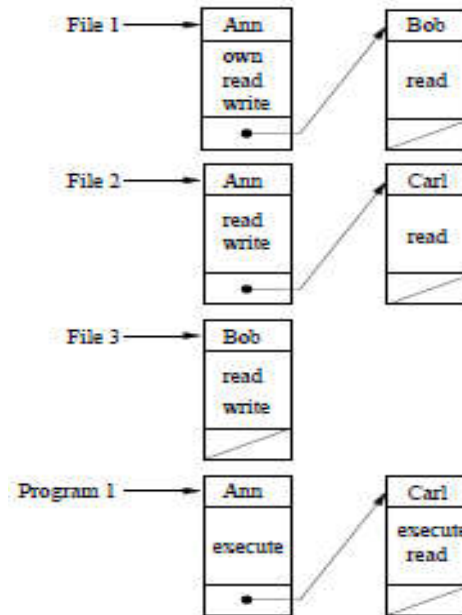
Sparse Matrices - Authorization Tables

- Authorization tables,
 - Commonly used in relational DBMS
 - Store table of non-null triples (s,o,a).

User	Access mode	Object
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2

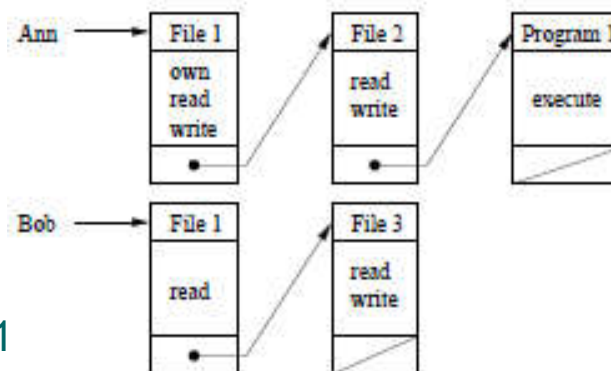
Matrices – ACLs and Capabilities

- Access Control Lists (ACL)
 - store M by columns,
 - together with each object,



most popular,
Unix, Win

- Capabilities
 - store M by rows.
 - for each user store his capabilities,



ACLs vs. Capabilities



In theory...

ACLs are widely used (Linux, Windows, etc.)

In theory, Access Control Lists (ACLs) and capabilities represent the same thing.

So if we implement ACLs, no need for capabilities.

In practice however, they lead to very different systems.

Managing Permissions

- With ACLs, the power to edit the authorities (permissions) is aggregated by resource.
 - naturally compatible with Discretionary Access Control, where there is often the notion of an owner
- With capabilities it will be rather aggregated by Subjects.

Authentication

ACL's:

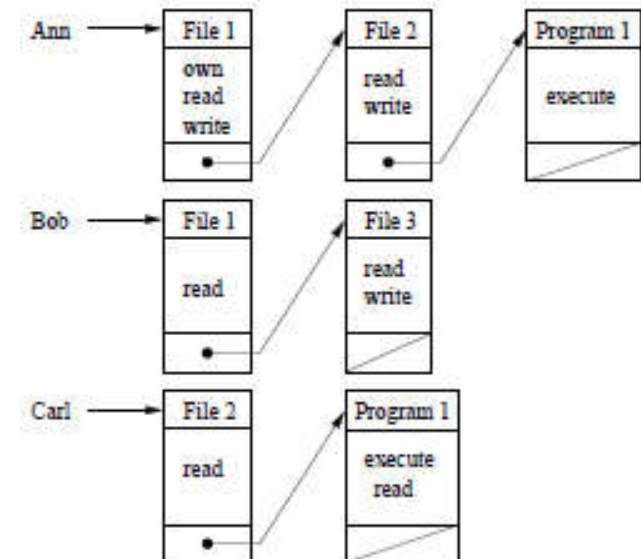
- store rights together with each object,
- requires a form of authentication of subjects at the moment of access: a process should not be able to pretext it belongs to sb.else

Capabilities:

- for each user store his capabilities,
- does not require authentication of subjects:
 - capabilities are explicit rights in a form of a token, that represents the user's capabilities.
- but require some form of **unforgeability**
 - + maybe some form control of propagation of capabilities...
 - token:
 - now the hacker may try to copy this token from one user to another. So it should be cryptographically signed, and depend on the user's ID! (some people encrypt capabilities too).

**Beware - Capabilities

- the term “capability” has several different meanings in the literature (from weaker to stronger):
 - frequent informal usage of this word:
 - means that a process has or not certain rights/privileges
 - Example: the *SETUID* capability of a process in Linux
 - In access control it should be an “un-forgable reference” token, with greater security, whatever we mean by “un-forgable”...



Fast Access, Review, revocation

- ACL's provide **faster** access, review and revocation on a per-object basis
 - but if we want to revoke permissions for a particular user, we have to search a whole hard drive...
- Capabilities provide **faster** access and review and revocation on a per-subject basis

Least Privilege

- Capabilities are better in this respect,
 - especially for dynamic short-lived subjects created for specific tasks

Ambient Authority

= def = Making a request that only specifies

- the names of the object(s) involved and
 - the operation to be performed on them,
- is enough for a permitted action to succeed.

⇒ dominant method today

(POSIX ACLs, Windows as well).

With **capability-based security** programs receive also permissions as they might receive data.

- this allows programs to determine where the permissions came from.

The Confused Deputy Problem

Definition: The confused deputy problem occurs when one process tricks another process to do an action he doesn't have permissions to do.

Example 1: A compiler is given a permission to write in a directory. The user compiles a program and specifies some very special filename for the output log. So he can overwrite some files he should not have access to.

Solution:

- Bundle together the designation of an object and the permission to access that object = **a capability**.

Implement... Both?!

ACLs are widely used (Linux, Windows, etc.)

But implementing **Capabilities** too can be seen as a useful complement, and better security can be implemented.

Most real-world OS use ACLs, not capabilities.

In fact programming with capabilities would be way more difficult...

Remark: Better Than Capabilities?

It is called **Procedure-Oriented Access Control**.

Here the only access to an object is
a procedure that encapsulates the object:

- controls the access
- provides trusted interface
- implements various protections

here the object could be “in the clouds” too... the only thing we know about is the procedure.

Composition of Policies

Composition of Policies

Can we have all the benefits of DAC and MAC?

The simplest method:

(works like a logical AND)

- allow an operation only if all policies implemented allow it.

Important Digression: Rice Theorem



Interesting Questions

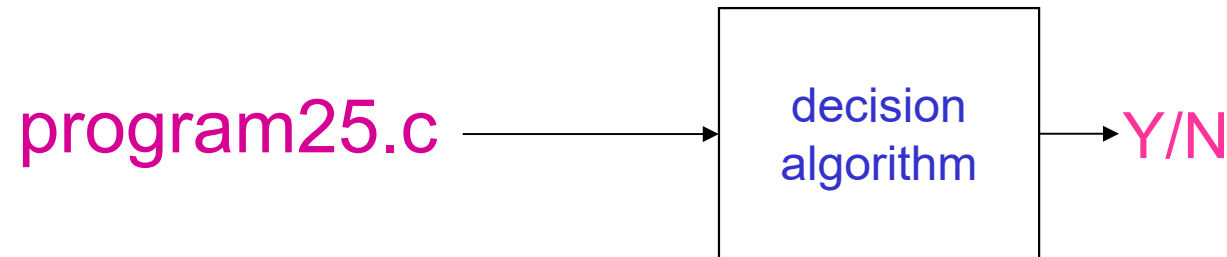
Example Questions:

Q1: Does `program25.c` always return 0?

Q2: Does `program25.c` compute the sum of two 32-bit integers correctly?

Q3: Do 2 programs do the same thing?

Etc..



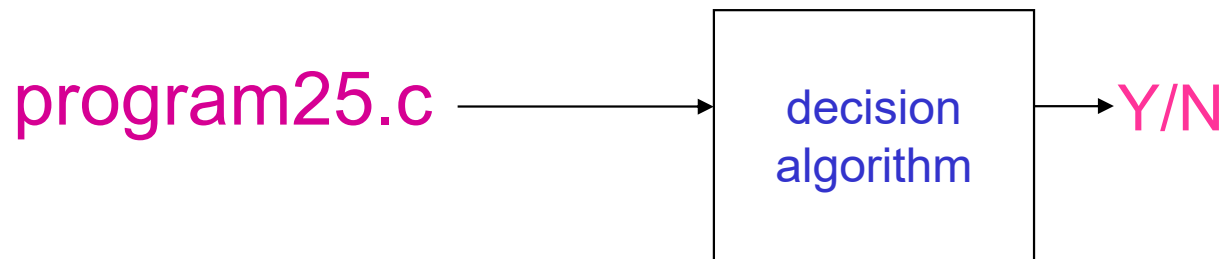
Answer [Rice 1953]: there is no algorithm that can solve this problem.

Rice Theorem [1953]

very important theory result in computer science.

About computer programs seen as a **black box**.

Theorem [Rice]: there is no such algorithm. Except for trivial properties where the answer is the same for all programs.



The Turing Machines vs. Science

Science can

- prove that many programs have some property,
 - it can disprove the property for many programs,
 - but there will always be complex computer programs which escape our understanding and – they will be in the **grey zone**.
-
- Examples:
 - Is AES a secure encryption algorithm?
 - Is ECDSA+secp256k1+SHA256 used in bitcoin secure?
 - Is there a virus on my PC?

Too General a Question...

Almost all questions we want to ask, though they do have an answer Yes or No, cannot be answered, neither by a machine, neither by any of UCL professors.

Conclusion: if our model is very powerful, there is no method to know certain things. The model must be restricted.

The essence of security is restriction.

In particular!

We don't want to have a Turing machine.

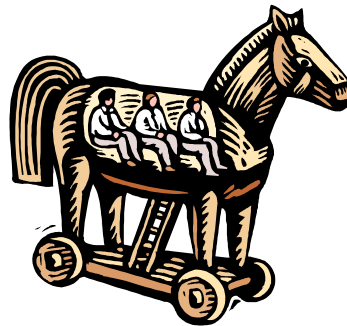
We want to have a simpler machine so that we can know its properties sufficiently well and predict the outcomes better.

Important insights into the problem of securing PCs

if we **restrict** what can be done, it may be possible to have a computer to decide whether our program is correct or not. And whether is secure or not.

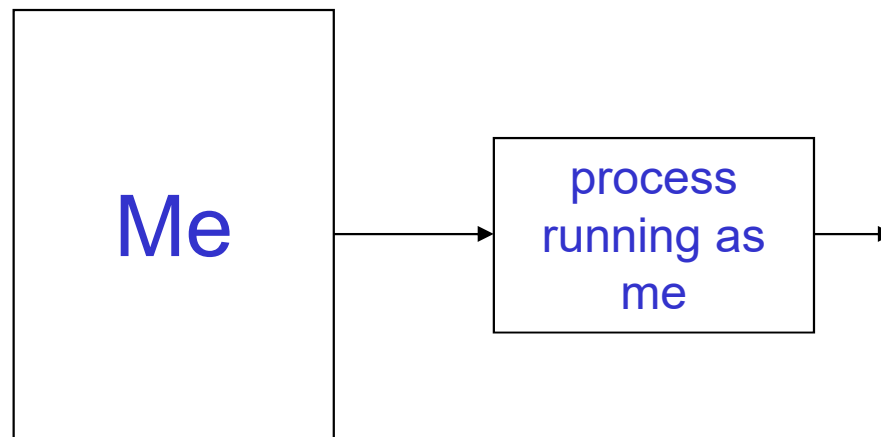
Example: JavaCard defines a limited subset of Java, where it is MUCH easier to know if a given program is harmful or not...

What's Wrong with DAC?



Trojan Horses

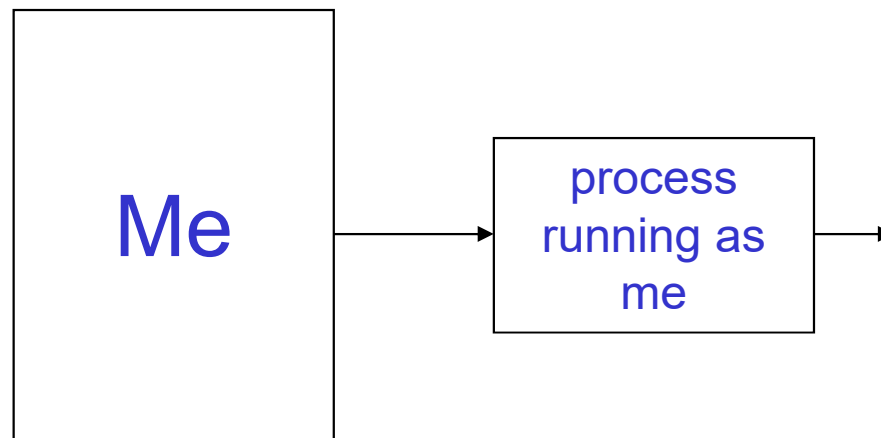
DAC: the tendency is that users grant their privileges to programs they run.



This can be exploited by Trojans.

Trojan Horses

Trojans : Malicious software that is given legitimate access rights, and that will exploit the privileges it is given to do an action that the user would not authorize, breaching the security from the inside.



But **I cannot know** what this process is up to, and this not (not only) because I'm not smart enough: cf. Rice theorem.

Secure State (1)

ACL

File F

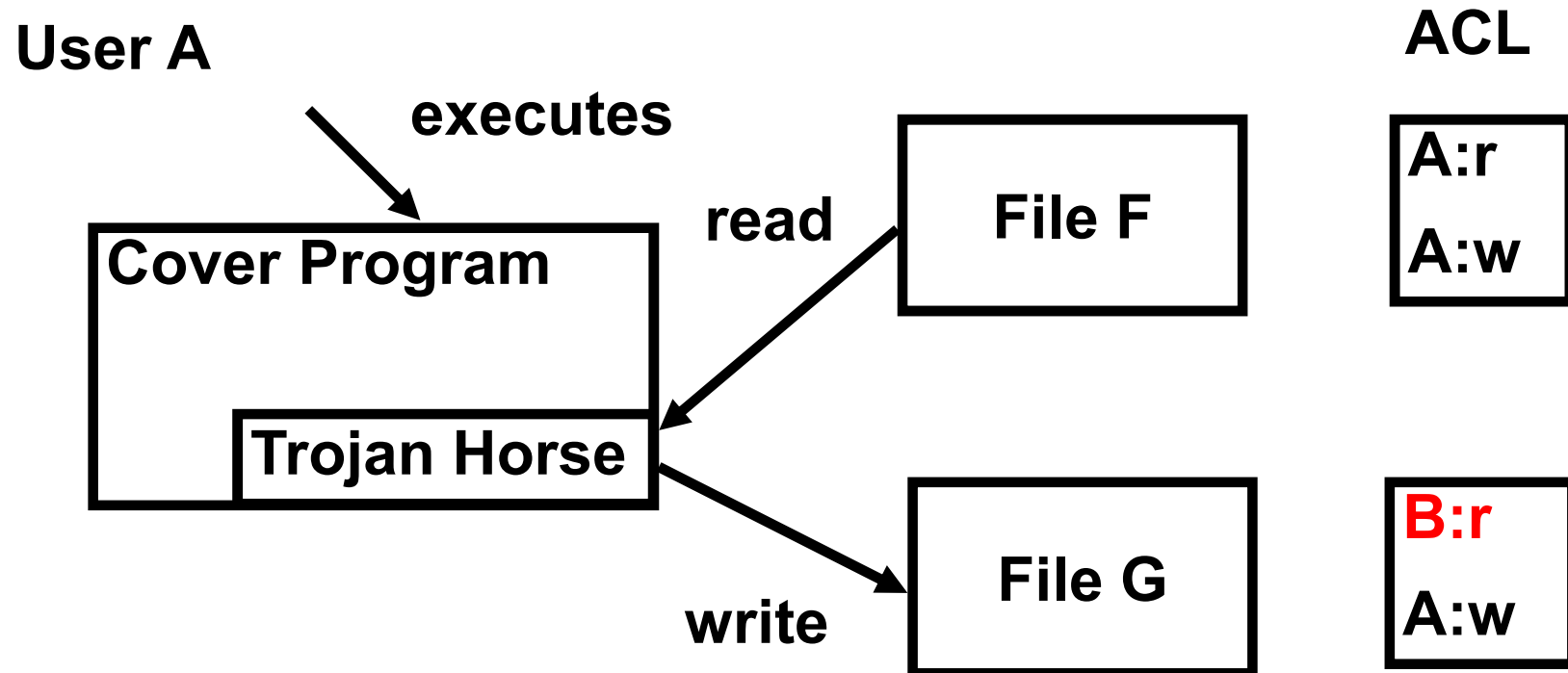
A:r
A:w

File G

B:r
A:w

User B cannot read file F

Now We Introduce a Trojan (2)



Now user B can read file F copied to file G .

Mandatory Access Control



Definition

Mandatory Access Control restricts the access based on a system-wide policy

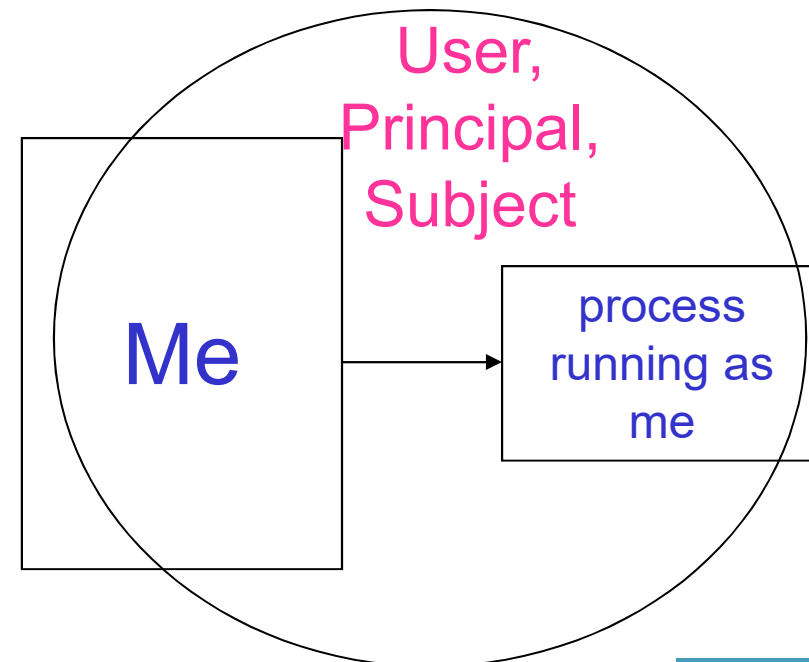
- Possibly denying users full control over the resources that they created themselves.
 - the system policy (set by the administrator), not individual choices of participants, determines the access rights

Motivation for Mandatory Access Control

One of the motivations:

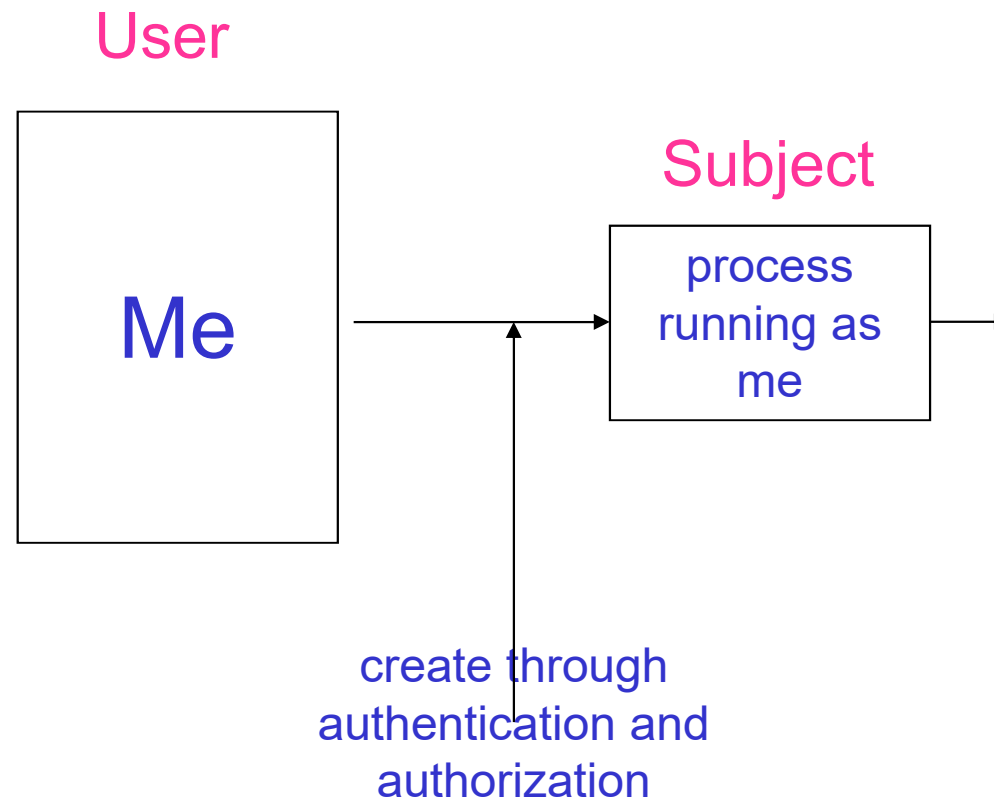
can we impose stricter restrictions and limitations that, for example could NOT be bypassed by Trojan Horses?

Avoid confusion
between me,
and a process
run by me



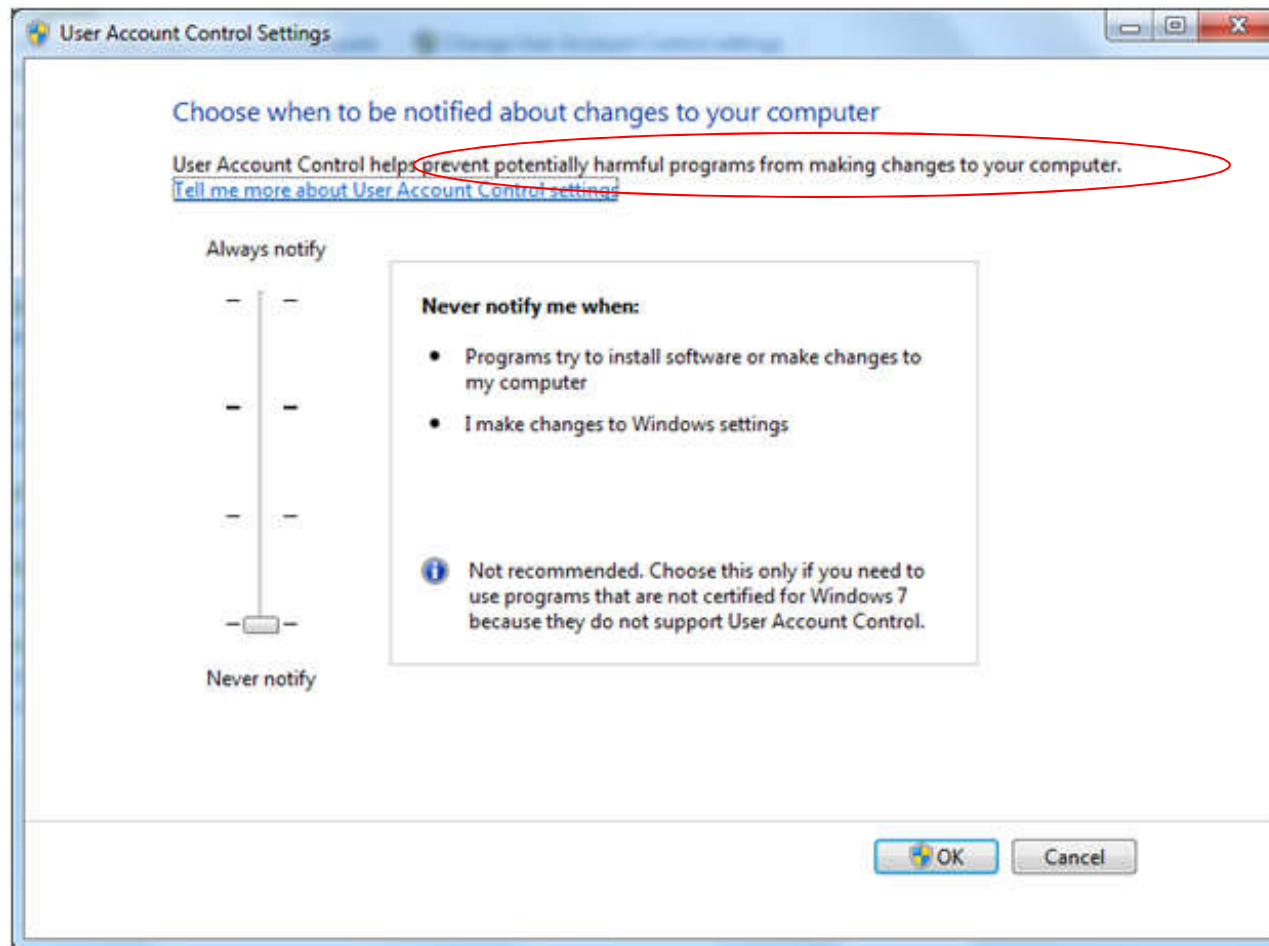
Clear Distinction + Mistrust

Here even if we trust the user,
we DO NOT trust the programs(!).



Windows 7 and Vista

UAC = User Access Control



Popular W7 UAC Policy

in one popular setting: mouse clicks have the admin power, programs don't

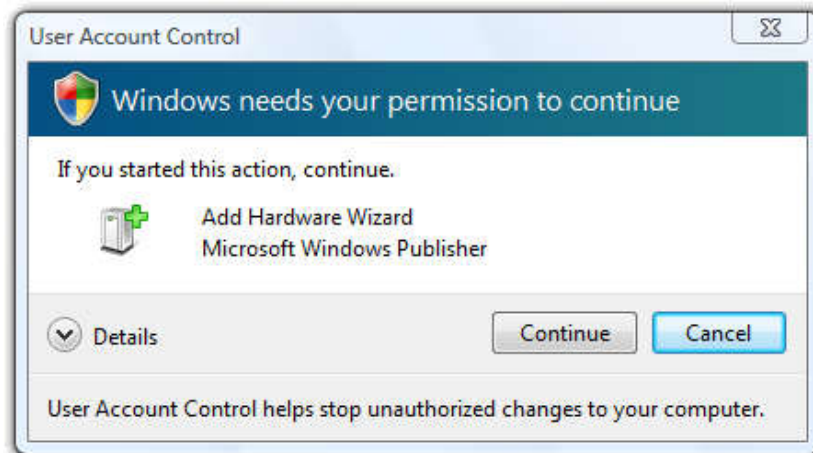
User

Subject

Me

process
running as
Me

has Administrator rights

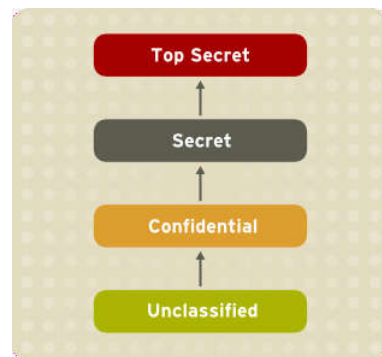


1. CANNOT do it normally
will ask for permission!!!

2. =>unless
run as admin
from the start



Multilevel Security + (later) BLP Model



Multilevel Security

The most common form of MAC.

- Based on a classification of subjects AND objects.
- Two main classes of policies:
 - Secrecy-oriented: Bell LaPadula (BLP) model.
 - Integrity-oriented: Biba model.

Bell LaPadula [1973]

Research was motivated by US army.

Main Objective:

- Be able to formally show that a given computer system can securely process classified information

Main Idea in Multilevel Security

The idea originated in the military.

Security levels, or levels of security clearance are ordered.



Example:

unclassified \leq confidential \leq secret \leq topsecret

Both subjects and objects are placed at one level...

for now we have a total ordering...

Detailed Semantics

This type classification is officially used by many countries and organisations. Originated in WW2 and Cold War.

- **Top Secret**
 - ✓ | ▪ in-depth background checks, highly trusted individual
- **Secret**
 - ✓ | ▪ routine background checks, trusted individual
- **Confidential / For Official Use Only / Sensitive**
 - no background check, but very limited distribution
- ✓ | ▪ minimally trusted individuals, but in limited numbers and on the need-to-know basis
 - no obligation to disclose, keep confidential
- **Unclassified**
 - Unlimited distribution
 - Can be given to untrusted individuals

A “Commercial”-style Example

Example:

public \leq sensitive/official use \leq proprietary \leq restricted

Similar semantics except that **proprietary** is mostly about intellectual property, there are strong laws in this domain and private business will rather only create security rules that do have vital and legal importance, otherwise they maybe wouldn't bother about secrecy that much.

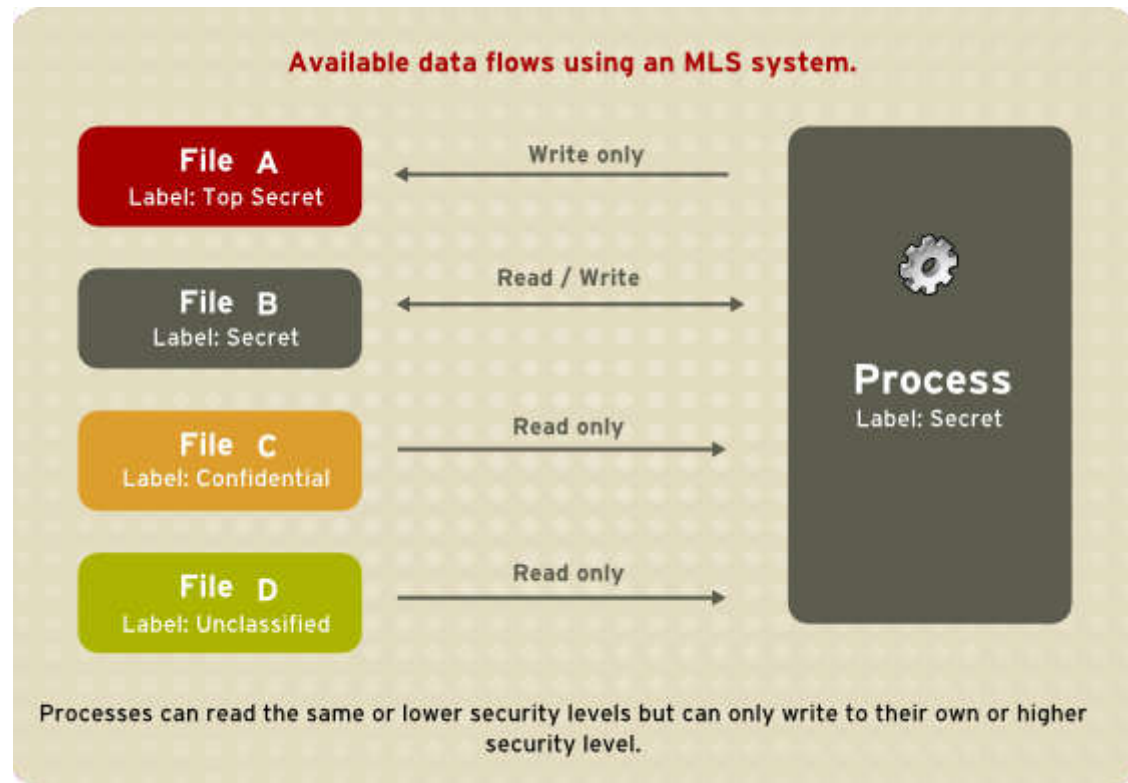
What's the Point?

Any ordering, immediately allows to have policies such as: (see BLP Model studied later)

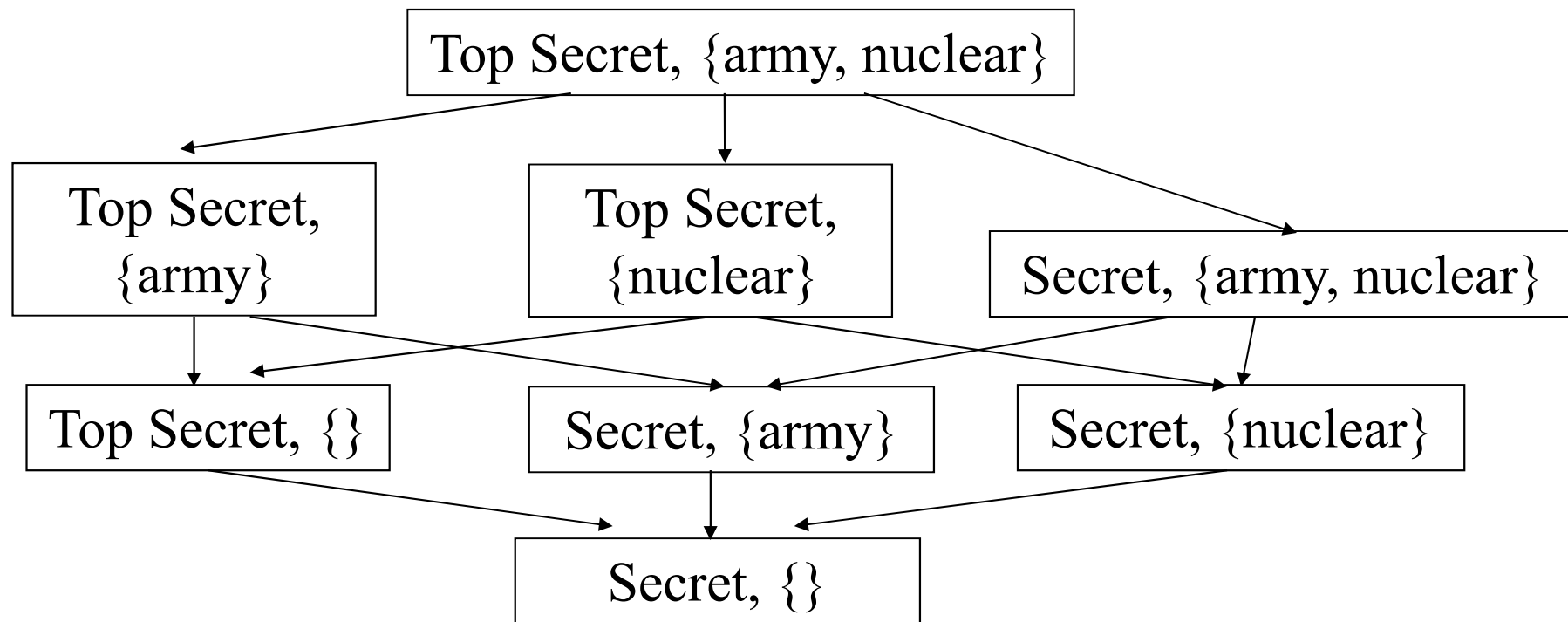
- no read up:
- no write down:

Misleading or “Easy” Example – Total Order

With **no read up** and **no write down** rules:



and with **Partial** Order??



What's the Point?

We need to revise/clarify or simply distort(!) the meaning of these words:

- no read up:
- no write down:

The Essence of BLP Model = Bell-LaPadula

Any ordering, immediately allows to have policies such as (part of BLP, studied later):

- **no read up:**
 - cannot read a file at a higher level.
- **no write down:**
 - a process with a confidential clearance cannot write a non-classified file
 - because it might contain confidential information...
 - this rule says that the “confidential clearance” is not only a right, but also an obligation.

A Small Precision Needed

to avoid misunderstandings,
it should be clear that we allow
neither write nor read
for levels that are unrelated
w.r.t. our **partial** order \leq .

BAD TERMINOLOGY!!!

Any ordering, immediately allows to have policies such as (part of BLP, studied later):

- **no read up?:**
 - cannot read a file at a higher level.
- **no write down?:**
 - a process with a confidential clearance cannot write a non-classified file
 - because it might contain confidential information...
 - this rule says that the “confidential clearance” is not only a right, but also an obligation.

BAD TERMINOLOGY!!!

Any ordering, immediately allows to have policies such as (part of BLP, studied later):

- **no read up?: read down only!** + must be related by \leq
 - cannot read a file at a higher level.
- **no write down?: write up! only** + must be related by \leq
 - a process with a confidential clearance cannot write a non-classified file
 - because it might contain confidential information...
 - this rule says that the “confidential clearance” is not only a right, but also an obligation.

posets

Forget a total ordering...

We usually need more flexibility and security:
achieved through **compartmentalization**.

Some levels will not be comparable and
we will have posets.

Compartmentalization

We want to add additional categorizations, for example describing different departments in an organisation, or different armed forces, or the authority granted in different domains...

- this will allow us to implement severe access control limitations of the type NTK = “Need-To-Know”
- this is able of managing complex situations, usually imposed by law or contractual obligations, therefore one needs to be strict. Example: The U.S. have two allies that maybe do not wish to share foreign intelligence data (say Israel and Saudi Arabia).

The policy will then have three distinct levels:

1. Top Secret, Israel
2. Top Secret, Saudi Arabia
3. Top Secret, Middle East (including Israel and Saudi Arabia)

And as will se later, in our model the only allowed information flows are $3 \rightarrow 1$ and $3 \rightarrow 2$.

Compartmentalization - Examples

- army, navy, air force

In civilian sector:

- Sales, R&D, HR

*Further Mistrust

Multi-level security policies allow in fact to get a further degree mistrust in people, and processes they own/create (that could be Trojans).

And this can go as far as we want:

The policies of no read-up and no write-down become more and more restrictive when the ordering is partial, so that there is less objects accessible to read or write from.

*Lattices

Definition:

An poset S, \leq is called a lattice if:

- $\forall a, b$ the LUB $a \vee b$ exists.
- $\forall a, b$ the GLB $a \wedge b$ exists.

Corollary 1: every finite subset has a SUP and an INF.

Corollary 2: In every finite lattice we have two special elements called **top** \top and **bottom** \perp .

*Multilevel Security and Lattices

in security jargon, when

$$a \leq b$$

we say:

“b dominates a”.

We call:

- **system low** is the **bottom** level \perp
 - dominated by all others.
- **system high** is the top level T
 - which dominates all others.



The Meaning of \vee and \wedge in MLS Policies

- For every **two objects** at levels **a** and **b**,
 - there is a minimal security level **a \vee b** for a **subject** to read both.
 - there is a maximal security level **a \wedge b** for a **subject** to be able to write both.

- For every **two subjects** at levels **a** and **b**
 - there is a minimal security level **a \vee b** for an **object** to be accessible in writing by both.
 - there is a maximal security level **a \wedge b** for an **object** to be accessible in reading by both.

The Product Lattice

- Let **H** be a set of **classifications** and a total ordering \leq_H . our first lattice.
- Let **Cat** be a set of **categories**.
 - We call **compartments** = NTKa's arbitrary subsets of **Cat**. our 2nd lattice.

= Need To Know areas

security levels = secrecy classes = levels of clearance
 are pairs of (classification, compartment) where by definition

$$(L_1, C_1) \leq (L_2, C_2)$$

iff

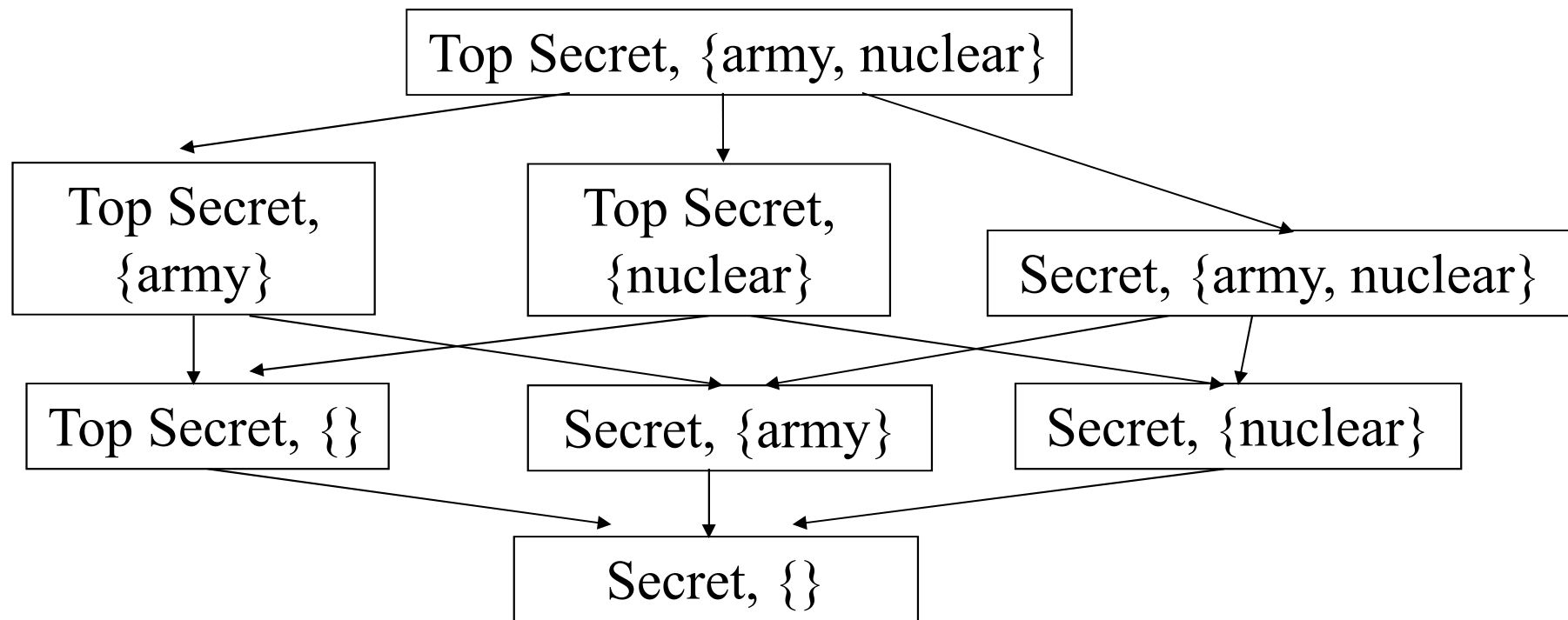
$$L_1 \leq_H L_2 \text{ AND } C_1 \subseteq C_2.$$

Remark: mathematicians will explain that it is in fact indeed what is known as “a product lattice” of the two lattices (H, \leq_H) and the lattice of subsets of **C**.

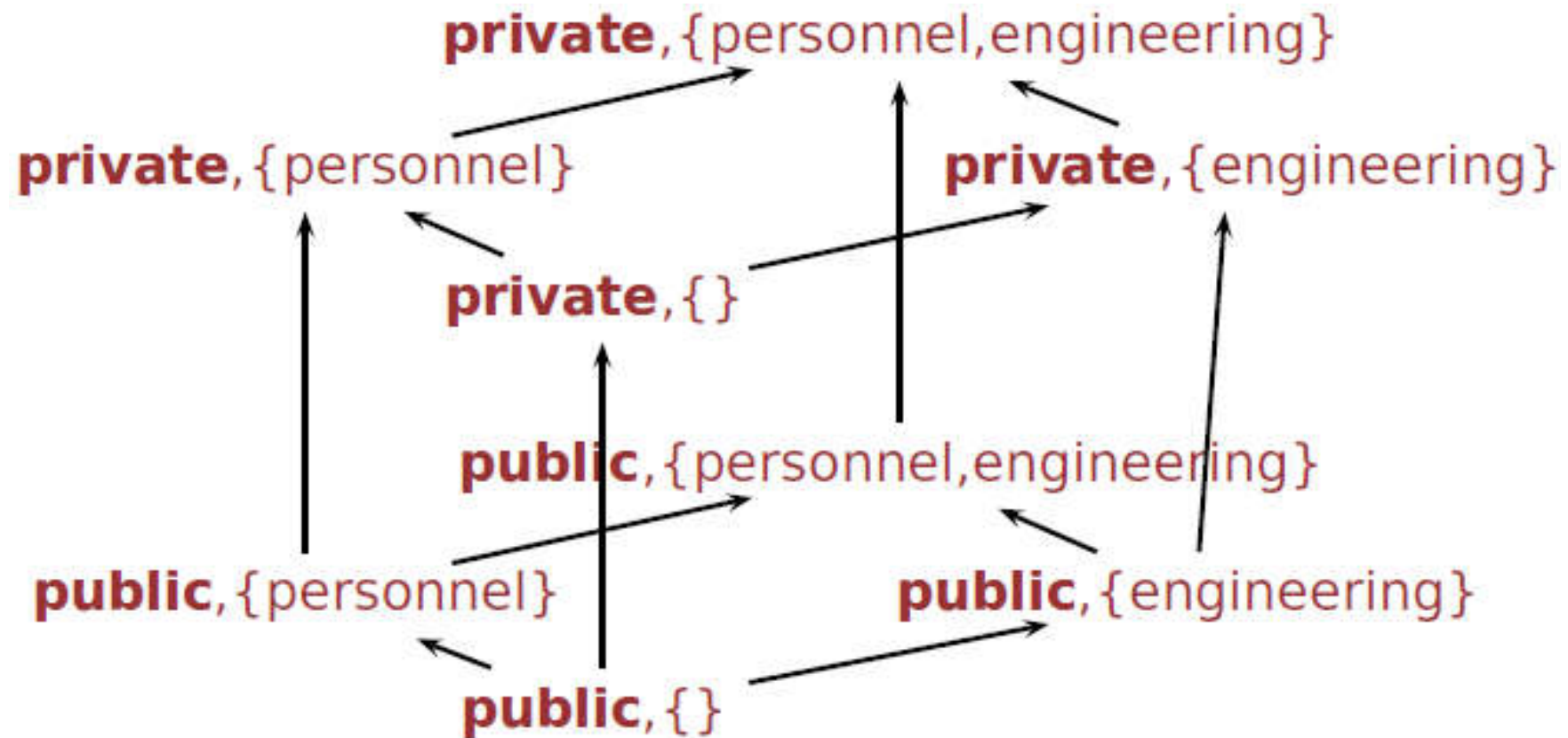
Military Example:

Levels={**Top Secret**, **Secret**}

Categories={**army**, **nuclear**}



“Civilian” Example (could be for a university):



Intended Application

- Each user is assigned a security class = clearance.
- A user can “connect to the system” at **a lower, user specified level of security**, allowed for any level that is dominated by his clearance.
 - A process activated by a subject takes the level of clearance with which the user has connected (not the user’s level of clearance).
- Examples (cf. our 2 pictures).
 - (private,{personnel})
 - can access private data about UCL personnel, but not inside the faculty of Engineering.
 - (Secret,{Army,Nuclear})
 - can access Secret information in both domains, but none of Top Secret information.

Further Usage and Semantics

- The class assigned to **a user**, reflects user's trustworthiness not to disclose sensitive information to individuals who do not hold appropriate clearance.
 - should disclose any information only to people with security clearance that dominates this user
- For **an object**, it reflects the sensitivity of information that this object will or may contain,
 - should be disclosed only to people/processes with security clearance that dominates this object.

Semantics of Categories/Compartments

- Each category can be viewed as the right to know (right to read) certain things (not all) in a given dimension/domain.
- The larger is the number of categories, the less the data is shared following the “need-to-know” principle.
 - But daily business may be very difficult if there are too many categories...
 - so we need a “sensible split” that is both secure and practical...
 - Ross Anderson: “MLS systems... impair operational effectiveness”.

Bell LaPadula (BLP) Multi-Level Security Model



BLP Model [1973]

This model is simply

1. Our **product lattice** +
2. two mandatory rules:
 - **no write down** and
 - **no read up**.
3. + extra **DAC** mechanism.

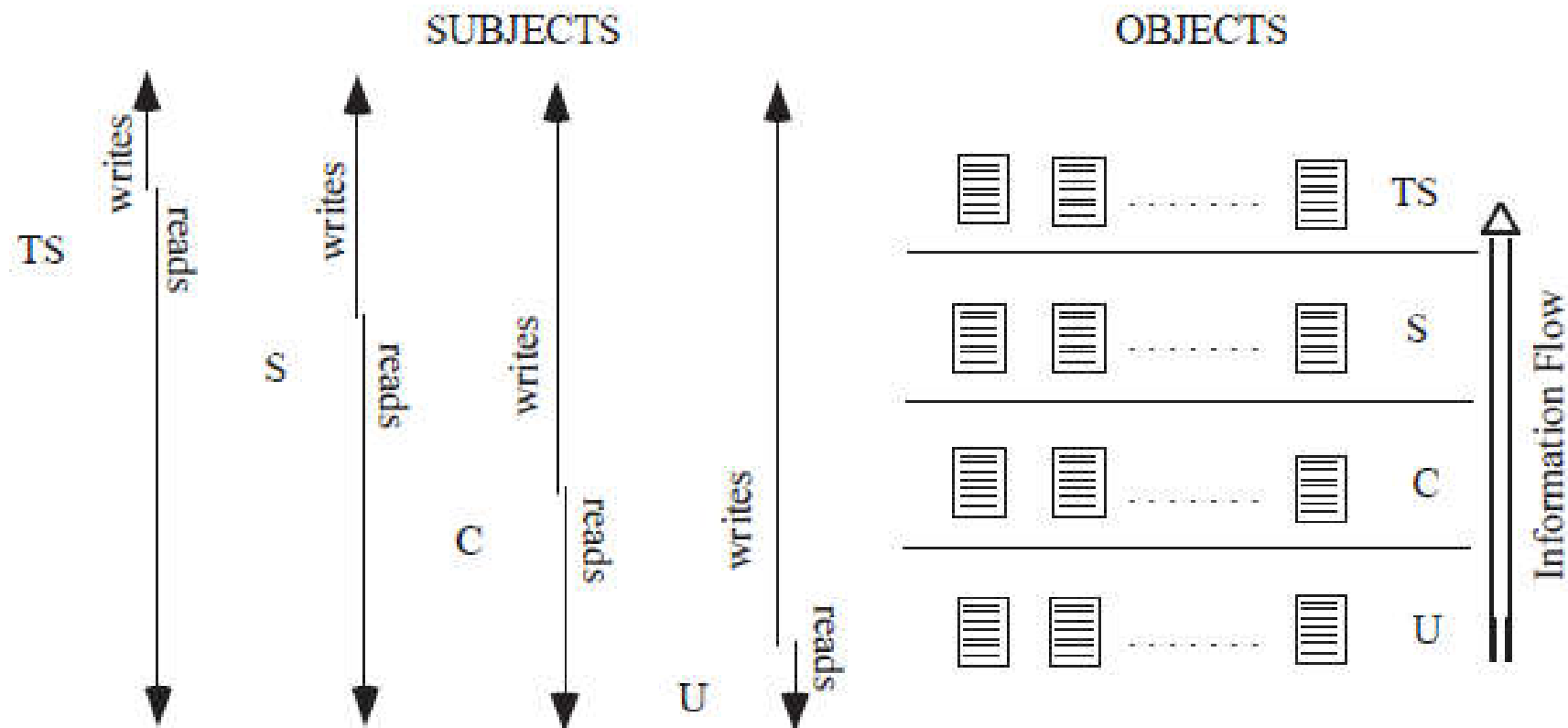
defence in depth:



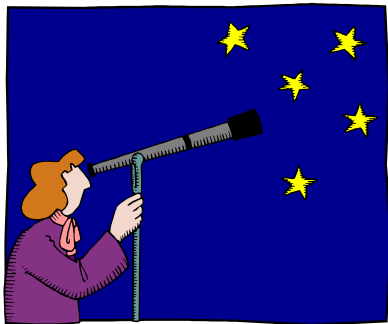
Goal: prevent information flowing from high level to low level.

Information Flow and Confidentiality

With **no read up** and **no write down** rules active we get a very good enforcement of **confidentiality** as intended.



Formalization and Theory for the BLP Model



*Access Control Models

Formally and mathematically define the access control method. It should be:

- Complete
 - Encompass all our security desiderata.
- Consistent.
 - Free of contradictions.

*Access Control Models

Benefits:

We can formally prove security properties of a system.
Derived from basic premises.

Nice split between conceptual and practical security:

- Prove that model is “secure”.
- And that the implementation is correct.

Allows to be claim that security is achieved.

- And if it isn't, we should be able to blame EITHER the model
OR the implementation,
without ambiguity.

BLP Model: A Long Definition

- A set **S** of Subjects (e.g. processes) which can execute privileges in the system.
- A set **O** of Objects (e.g. files) on which the privileges can be executed.
- A set **A** of Operations. Example: $A = \{\text{exec}, \text{read}, \text{append}, \text{write}\}$.
- A **poset** (L, \leq) of levels of security clearance, e.g. $\langle \text{un}, \{\} \rangle \leq \langle \text{cl}, \{\text{HR}, \text{ENG}\} \rangle$.
- Let $b \subseteq S \times O \times A$ be a **current access** defined as a set of triples (s, o, a) meaning **s** is performing operation **a** on an object **o**. Defines what is happening at a time.
- Let $B = 2^{S \times O \times A}$ be the set of all possible current accesses **b** (all possible outcomes).
- We will define an **access control matrix** as before:

$$M = (M_{so})_{s \in S, o \in O} \in M_{S,O}(A)$$

- where each entry $M_{so} \subseteq A$. being the set of **permissions** **s** is given for accessing an object **o**: **s** can perform only operations **a** such that $a \in M_{so} \subseteq A$.
- We call a **security level assignment** **f** a triple of functions $f = (f_s, f_c, f_o)$ where:
 - $f_s: S \rightarrow L$ gives the **maximum security level** each subject can have.
 - $f_c: S \rightarrow L$ gives the **current security level** each subject has at this moment, with $f_c \leq f_s$.
 - $f_o: O \rightarrow L$ is the **security classification level** for all objects.
- We call $F = L^S \times L^S \times L^O$ the set of all **security level assignments**.
- We call a **BLP state** a triple (b, M, f) .

BLP Mandatory Access Policy = 1+2

Let the current BLP state be some triple (b, M, f) .

- here b will be the current access = a set of triples (s, o, a) .

1. The simple security property = ss property = no read up property:

For each access $(s, o, a) \in b$,

- if $a \in \{\text{read}\}$, then $f_o(o) \leq f_c(s) \leq f_s(s)$.

maximum security level

levels of
protection/classification
for objects

2. The *-property = no write down property:

For each access $(s, o, a) \in b$,

current security level

- if $a \in \{\text{append}, \text{write}\}$, then $f_c(s) \leq f_o(o)$ and moreover
- $\forall o'$ with $(s, o', a') \in b$ and $a' \in \{\text{read}\}$ we have $f_o(o') \leq f_o(o)$
 - in other words, the level o must dominate every other object o' which s can read at the same time (to prevent data flow).

3. BLP Can Also Obey DAC

3. The **discretionary security property** = **ds property** = **DAC property**:

For each access $(s,o,a) \in b$, $a \in M_{so}$.

We say that a state (b,M,f) is **secure** if the three properties 1,2,3 are satisfied.

If these properties hold, the security in the BLP model is entirely captured by this notion of current state of permissions:

BLP state = a triple $(b, M, (f_s, f_c, f_o)) \subseteq (S \times O \times A) \times M_{S,O}(A) \times (L^S \times L^O \times L^A)$.

State Transitions in BLP

We say that a state (b, M, f) is **secure** if the three properties 1,2,3 are satisfied.

Let $r \in R$ be an arbitrary set of operations
and consider any a given **current access** $b \in B$.

set of triples (s, o, a) meaning s is
performing operation a on an object o

We define a **state transition function** $T: B \times R \rightarrow B$,

- For any **current access** $b \subseteq S \times O \times A$, and any request $r \in R$.
- It gives $T(b, r) = b'$, the next state.

Now we say that a transition function T preserves the three properties, if...
(long definition - on the next page). Then we have:

Key Theorem [BLP]: The security in BLP is preserved inductively: If the initial state is secure and all transitions are secure, the system remains secure at any moment.

So all we need to check is if the transitions preserve 123.

**Missing Definition

T is a **Security Preserving State Transition** if:

For every $b \in \mathbf{b}$ reachable from $b_0 \in \mathbf{b}$

after executing one or more requests from \mathbf{R} , we have:

if $T(b, c) = b'$, where

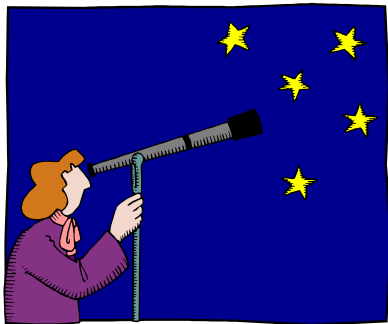
$b = (b, M, (f_s, f_c, f_o))$, and

$b' = (b', M', (f'_s, f'_c, f'_o))$,

then $\forall s \in S, o \in O$:

- | | | | |
|---------------------------------------|---------------------------------|---------------|----------------------------------|
| $-(s, o, \text{read}) \in b' \wedge$ | $(s, o, \text{read}) \notin b$ | \Rightarrow | $f'_c(s) \geq f'_o(o)$ |
| $-(s, o, \text{read}) \in b \wedge$ | $f'_c(s) < f'_o(o)$ | \Rightarrow | $(s, o, \text{read}) \notin b'$ |
| $-(s, o, \text{write}) \in b' \wedge$ | $(s, o, \text{write}) \notin b$ | \Rightarrow | $f'_o(o) \geq f'_c(s)$ |
| $-(s, o, \text{write}) \in b \wedge$ | $f'_o(o) < f'_c(s)$ | \Rightarrow | $(s, o, \text{write}) \notin b'$ |

Beyond BLP Model



Applications of BLP Model

- Multics [a very influential mainframe time-sharing OS, 1964-2000] used BLP.
- Apparently the current DAC in Unix was inherited from Multics

Limitations of BLP Model

- It only says what to do with read and write.
- What about exe?
- It is really VERY VERY strict.
 - And even with this, a certain McLean criticized BLP for being not strict enough, to prevent **covert channels**
 - which are very hard to prevent anyway.

On Need to Write-Down!!

As stated, in the BLP model, a user at a higher level, connected at higher level cannot even send commands to a lower level.

Not very practical for sending orders for example(!).

There are two ways out of it:

1. temporal downgrade of Subject (already built-in in our current formulation of BLP):

This why we have the current security clearance level with $f_c \leq f_s$.

This approach works because users cannot at the same time access highly sensitive files and send messages to lower levels. It works as far as people themselves can be trusted, but it does prevent any Trojan horse attack (!) because the legitimate channels for the Trojan to use are blocked when connected at a lower level.

2. identify a set of trusted subjects – a variation of BLP – that will be allowed to write-down.

Example: Certain people will be allowed to publish or diffuse parts of a secret document...

**One Can Make It Stricter

It is possible to make BLP stricter with the notion of data aggregation:

- A file or Subject which has access to a certain set of values can be automatically classified **higher** than the value singularly taken.
 - Example: name + salary.

DAC + MAC

BLP implements both.

DAC provides “discretionality”, but only within the strict boundaries of Mandatory AC rules.

Can We Change the Levels?

- Raising the classification security level of an object in the lattice:
 - not a problem.
- Changing to an unrelated one
 - a problem.
- Changing to a lower one
 - a possible security breach...

Need for special procedures: [declassification](#), [sanitization](#) etc.

Def: [Sanitization](#): editing the document so that it does not leak any extra information that the one that is intended to make public such as exact names, exact places, file meta data, etc etc.

Tranquillity

The key problem in BLP is that,
if we change levels in a live system
then actually the BLP rules can be violated
(the Trojan can accumulate some data in a file and leak
information directly because levels has changed).

So many implementations require either

- some of form of the so called “Tranquility”
- or that the subjects “forget everything” when changing levels, for example stop all tasks, log off, clear all temporary files etc.

Tranquillity

There are 4 versions:

- **Strong** Tranquillity: Security level of objects never change.
- **Weak** Tranquillity: Security level of objects never change in a manner that would violate the rules of a given security policy.
- Either
 - **life-time** tranquility: security levels never change during the lifetime of the system.
 - **temporary**: during operation = during the time the object is accessed.

Attacks on BLP Systems in Operation



Types of Failures – Adapted Slide

- Failure in the model
- Failure in the implementation
- Failure in operation

Covert Channels

The BLP model protects very well against **overt** (legitimate) channels. **Covert** channels are hard to prevent, but it is good to know where the problems may arise.

Examples: low level subject requests a resource used by high-level subjects. Access is poor or refused (can be a file lock). He deduces at which moment of the day his boss is not in office...

Remark: this kind of channel can be used to leak whole files 1 bit / second or so...

Two Cultures



**History of Computer Security

Cold War: very important questions were

- detecting Russian missiles coming from the north
- predicting the weather very reliably for a plane to be able to carry nuclear weapons

=> these questions were main sources for an unlimited appetite for computing power

=> necessity to:

- use many distributed computers
- securely aggregate data from many remote stations

James P. Anderson report:

“Computer Security Technology Planning Study” (1972)

Ever since then, very significant research funds were allocated by US DoD to study computer security.

The Heritage of the Military

(military funding dominated this area in the 1970s)

Sort of **historical bias** and focus on **Secrecy...**
Neglecting **Integrity** and **Availability...**

Today even the military regret that...



Military vs. Commercial Data

Military data security: focus on secrecy, what counts is what you can read (can be classified) and write. Prevent **leaks**.



Commercial data security: focus on integrity and authenticity of data: prevent **fraud**.

Focus on which program one can execute (prevent Trojans, and illegal software, for which usually the business, not the employee is legally liable – criminal charges).



Access Control in Commercial Environment

The requirements are almost contrary to the BLP model that is a model for the military (and relevant to organizations that cultivate “the spirit of a fortress”, e.g. a bank).



In most commercial environments, integrity, authenticity and authentication are key.

- Privacy and confidentiality are by far less important.



Why Integrity is Paramount

In business applications:

- An inventory system may function correctly if data is released, but the data should not be changed [example taken from Bishop].
- Do bank operation records balance each other?
- Did we pay all the employees correctly?
- Which outstanding invoices are unpaid?

Opportunities for fraud everywhere...

Fraud happens on regular basis in organisations. Usually not publicised (embarrassing for the company).

Integrity



What is Integrity?

A very general notion, one can put a lot of things here...

Technology + the human and organisational context + do our data correspond to the real world? Need to prevent flaws, errors. Bugs and threats alike.

The most general definition that tries to encompass it all, would be: **a system has its integrity if it can be trusted.**

A perfectly secure system can still lose its integrity, because of

- hostile penetration, or
- operational failure (bug, accident), or
- corrupt insiders, or
- rogue/careless employees,
- etc.

Basic Principles, Operational Integrity



Operational Integrity

Every system dealing with data has integrity mechanisms.

- File sharing and concurrent access support.
- Recovery in case of errors (file system inconsistencies after a power failure).
- Recovery mechanisms in case of interrupted system updates.
- Value range checks.
- Etc...

*Operational Integrity Properties

- **Atomicity**: either all of the actions of a transaction are performed or none of them are .
- **Consistency**: of the data (prevent error and accidental data corruption).
- **Persistence**: the results of committed transactions are permanent (prevent reset attacks).
- **Durability**: data will not be lost over long periods of time:
 - a hard drive kept for more than 10 years will lose data, many CD/DVDROMs are chemically unstable, flash memory and hard drives frequently fail during operation after 1-2 years, etc.

*Criteria to Safeguard Integrity [Clark-Wilson]

A bit of Big Brother stuff... Very interesting in the financial sector (problem with rogue traders, corrupted employees, and very large losses).

- **Authentication**: the system must separately authenticate each user, so that his actions can be monitored.
- **Audit (Logging)**: logs should contain many details.
- **Well-formed Transactions**: certain data can ONLY be manipulated by a restricted set of programs.
 - Interesting: Clark and Wilson postulated that it is THE DATA CENTRE controls (so external or complementary to these programs) would check if transactions are well-formed (how? See Clark-Wilson model).
- **Separation of Duty**: each user has programs to execute a part of the task
 - And again: Clark and Wilson postulated that it is THE DATA CENTRE that should ensure the principle of separation is applied.

**More Guiding Principles for Integrity



*Lipner 1982

These can be seen as management principles:

How to manage development and production, avoid random failures and security breaches alike.

Principle of Segregation of Duties = Separation of Duty.

*Segregation of Duties

Achieved by (closely related):

- Principle of **Functional separation**:

Several people should cooperate. Examples:

- one developer should not work alone on a critical application,
- the tester should not be the same person as the developer
- If two or more steps are required to perform a critical function, at least two different people should perform them, etc.
- This principle makes it very hard for one person to compromise the security, on purpose or inadvertently.
- Principle of **Dual Control**:
 - Example 1: in the SWIFT banking messaging management systems there are two security officers: **left security officer** and **right security officer**. Both must cooperate to allow certain operations.
 - Example 2: nuclear devices command.
 - Example 3: cryptographic secret sharing

*Auditing / Monitoring

- Record what actions took place and who performed them

*Lipner 1982:

Requirements, focus on integrity of the business processes and of the “production” data whatever it means:

1. Users will not write their own programs.
2. **Separation of Function:** Programmers will develop on test systems and test data, not on “production” systems.
3. A special procedure will allow to execute programs.
4. Compliance w.r.t. 3 will be controlled / audited.
5. Managers and auditors should have access to the system state and to all system logs.

Integrity Policies



Integrity Levels

Remark: these integrity levels will have nothing in common with classification levels we have known before in BLP...

Key insight / semantics: a higher integrity level means more confidence that

- A program will be executed correctly
- Data is accurate, reliable and not contaminated.
 - (again nothing about its secrecy is postulated, just integrity)

Example 1 [Lipner]:

- C = crucial
- VI = very important
- I = important

Integrity Levels - Example 2 [Lipner]

- **ISP** = Integrity level System Program
- ^{VI}**IO** = Integrity level Operational = production program.
- ^{VI}**ISL** = Integrity level System Low
 - at this **integrity level** ordinary users log in.

Example of **categories** [Lipner]:

- **ID** = Development
- **IP** = Production

Again, **compartments**

will be all subsets
of the set **{ID,IP}**.

Again, from here we will have **a product lattice**
defined in the same way as in BLP model.

Integrity Levels - Example 2 [Lipner]

Our **a product lattice** is as follows (same as for BLP):

Integrity levels are
ordered couples (level, compartment).

$$(L_1, C_1) \leq (L_2, C_2)$$

iff

$$L_1 \leq_H L_2 \text{ AND } C_1 \subseteq C_2.$$

Biba Requirement

Actually, a family of models, with more or less strict requirements.

We describe major variants.

- One can choose one of these versions/policies for a specific application, say securing your web browser...

Operations in Biba Model

In Biba's model, usual operations have different somewhat misleading names:

- Read == Observe;
- Write == Modify;

Biba's Model

The exact dual of the BLP model.

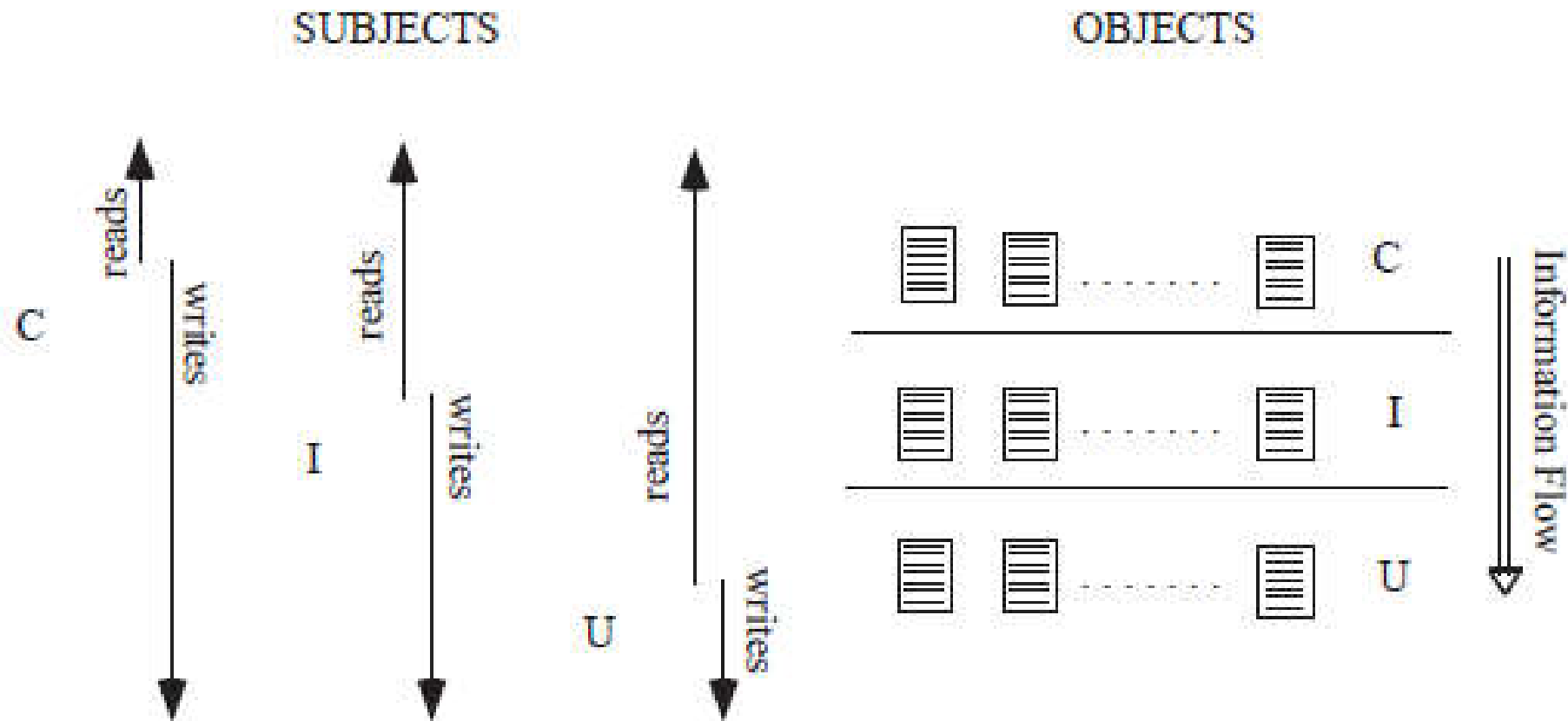
- Replace word confidentiality level with integrity level.
- Replace no read-up with **no read-down**.
- Replace no write-down with **no write-up**.

But with respect to **the integrity order**.

Another ordered set.

Biba's Model

no read-down + no write-up.



Main Part of the Biba Model

It has the **strict integrity policy** defined as follows:

- **no read-down:**
 - in fact it means **read up and only up**
 - prevents the integrity of a trusted subject from being contaminated by a less trusted data object.

- **no write-up:**
 - in fact it means **write down and only down**
 - restricts the contamination of data at higher level, since a subject is only allowed to modify data at their level or at a lower level.
 - This limits the damage that can be done by Trojans.

Categories/Compartments in BLP vs. Biba

BLP: Each category can be viewed as the right to know
(right to read) certain things (not all)
in a given dimension/domain.

The “need-to-know” principle.

Biba: Each category is a the right to modify (right to write)
certain things (not all)
in a given dimension/domain.

The “right-to-act-upon”.

Variants of Biba



1. Low-Water-Mark Policy for Subjects [for Reading][Bishop]

An extension of Biba's model.
A sort of relaxed no read-down.

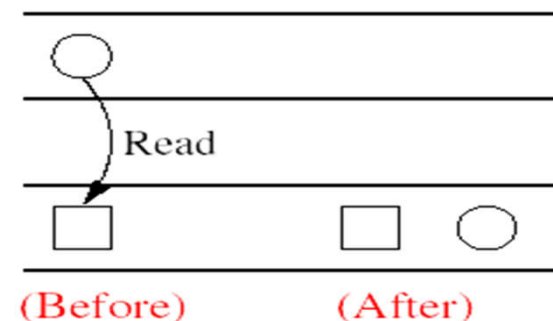
Definition:

Each time a subject accesses an object, **the integrity level** of the subject changes down to the GLB of the two.

Secure Downgrade.

=> Moreover, it is sufficient to use it
to lower the Subject's
level only for this session,

this works against Trojans already



2. Low-Water-Mark Policy for Objects [for Writing]

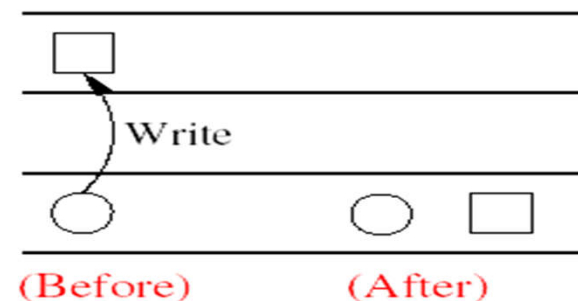
An extension of Biba's model.

A sort of relaxed no write-up.

Definition:

Each time a subject accesses an object, **the integrity level** of the object changes down to the GLB of the two.

Important: this policy is NOT a defence, it simply will indicate that certain objects may have been contaminated...



3. Low-Water-Mark Integrity Audit

Combines the two previous relaxations. Weak.

- **s** can always read **o**;
 - after reading
$$i(s) \leftarrow \min[i(s), i(o)]$$
- **s** can always write to **o**;
 - after writing
$$i(o) \leftarrow \min[i(s), i(o)]$$



4. Ring Policy – 4th Variant of Biba's Model

It is A+B+C+D.

- A. Integrity levels of both subjects and objects are fixed.
- B. **no write-up.**
- C. a subject may read an object at any level (**can read down**)
- D. **ring invocation property:** can nevertheless write up indirectly, by invoking another process with high integrity that is able to write but not arbitrarily, according to its own (presumed secure) rules of behaviour
 - » more about invocation later (2 different conflicting policies exist, later).

Informal statement:

when a program is placed at a certain integrity/trust/security level, it means that its behaviour is secure enough to securely write anything at this level or below, even if it can import data from low levels. We trust software at higher integrity levels to do anything.

Biba + Invocation



Operations in Biba Model

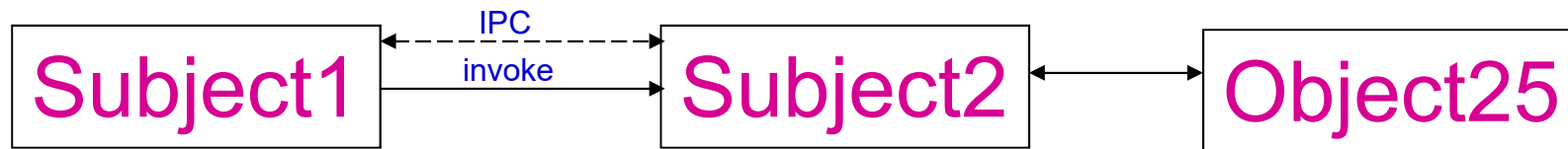
In Biba's paper, usual operations are given different more general names:

- **Read \leq Observe;**
 - (more general concept potentially applies to both Objects and Subjects)
- **Write \leq Modify;**
 - (again more general meaning)

Now Biba is frequently extended by:

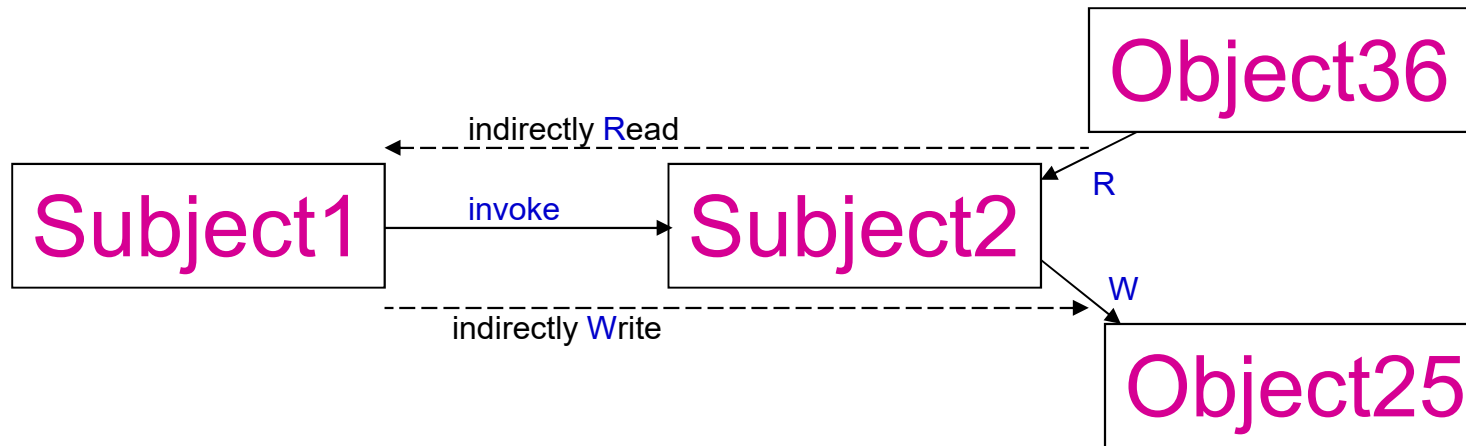
- **Execute \leq Invoke:**

Invocation



- **Execute \leq Invoke:**
 - the right for a Subject to run and communicate with another process = another Subject
 - thus accessing other Objects indirectly through invocation of a software tool.

2 Contradicting Policies for Invocation



This extra operation **Invoke** requires a policy to tell when it is allowed:

choice of:

- **Invocation Property** – invoke below itself.
- **Controlled Invocation = Ring Invocation** – only above itself.

2 Contradicting Policies for Invocation

1. **Invocation Property** – A subject s can only **invoke** another subject s' at or below its own integrity level, i.e. if it dominates this process with $i(s) \geq i(s')$.
 - Example: Admin invokes a tool to change ordinary user's password.
 - Motivation: Otherwise a “dirty” tool could use a clean tool to contaminate a clean object. Prevent abuse of a trusted program.
 - Example of what it prevents: prevent a virus, being unable to connect to the network due to a firewall, will call IE to connect for him and an offensive exploit string out to compromise a server, transmitted to IE as a command line parameter.
2. **Controlled Invocation = Ring Invocation:** – A subject s can **invoke** another process s' only when it is dominated by the level of process s' , if $i(s) \leq i(s')$.
 - Here low-level subjects should have access to high-level objects only through trusted/certified higher-level processes/tools. High-level tool is designed to perform all consistency checks to ensure object remains clean.
 - Example – user wants to change their own password – through password changing tool – controlled to only change user's password and nothing else in `/etc/shadow`.
 - Motivation: The process called has to be at least as trusted as the calling process (prevent calling a Trojan by a process that is not a Trojan).
 - Example of what it prevents: prevent admin using a third party management tool which has no verification of its security/integrity

2 Contradicting Policies?!

1. **Invocation Property** – subject can only invoke another subject below itself.
2. **Controlled Invocation = Ring Invocation:** – only above itself.

These reflect two different philosophies:

Question 1: what does it mean for a process to be at a high integrity level?

Does it mean it is:

1. **Trusted**, means has a lot of access privileges, then invocation up one's level means gaining privileges, should be banned.
2. **Trustworthy**, means whatever you do, even in an insecure environment it has a secure behaviour, for example only to do certain changes in a high-level file, and not to abuse access powers available?

Question 2: Do we want to prevent harm from:

1. **Indirect modification** abusing of a trusted software called by a rogue software to gain privileges such as altering other high integrity files?
2. **Direct modification**, and therefore we use trustworthy programs that can manipulate higher integrity objects such as system settings observing sensible rules?

now if we don't want any of the two, we need another policy:

3. **Prevent both**, invocation only at the same level.

Applications of Biba



Is All This Worth the Pain?

Imagine a PC with a USB port in an Internet café.

- Can Biba policy prevent contamination of the PC system?
 - Integrity only, confidentiality should be considered as a separate problem.

We can have 3 levels in the system:

3. Trusted OS components
2. Ordinary OS components and most PC files
1. Untrusted I/O and USB ports.

USB stick example, continued

Now, Biba's **strict integrity policy** will give the following benefits:

- **no read-down:**
 - prevents the system (levels 2,3) from reading anything from the USB stick. So no contamination possible.
 - **Low-Water-Mark Policy for Subjects:** relaxed. A pdf application, when reading the file from the USB stick, will be **downgraded** to integrity level 1 and then the policy will allow the user to print his file from his USB stick (this program will not need any access to level 2 and 3). then the process is not as trusted, no possibility for any contamination, and later it is closed.

USB stick example, continued

- no write-up.
 - It is OK to output a file downloaded from the web on the USB stick.
 - A program executed from the USB stick can be executed BUT in a simulated environment: with access to a false (simulated) hard drive and access to false (simulated) registry (cf. XP mode in Windows 7).
 - **Low-Water-Mark Policy for Objects** (dangerous): relaxed but allows contamination. When the user runs a program from his USB stick, and it modifies a file in the system, the file will be marked at system low.

Applications of Biba

- Fully implemented in FreeBSD 5.0.
 - The TrustedBSD MAC framework is a kernel extension of FreeBSD 5.0.
 - The integrity levels are defined for subjects and objects in a configuration file.
 - Support for both hierarchical and non-hierarchical labeling of **all system objects** with integrity data.
 - Supports the strict enforcement of information flow to prevent the corruption of high integrity objects by low integrity processes.

Drawbacks of Biba

- Nothing to support confidentiality.
- No support for revocation of rights (but can downgrade subjects).
- Apparently, there is no network protocol that would support Biba-like integrity labels over remote data volumes...

Biba + BLP

- Can be combined.
- Again, by composition, what is allowed is what is allowed by both policies.

Lipner has developed such a practical/simple combined policy framework for industrial applications, see Bishop's book.

Clark–Wilson Integrity Model

David D. Clark and David R. Wilson.

“A Comparison of Commercial and Military Computer Security Policies.”

In IEEE SSP 1987



Back to Operational Integrity



*Operational Integrity

Every system dealing with data has integrity mechanisms.

- File sharing and concurrent access support.
- Recovery in case of errors (file system inconsistencies after a power failure).
- Recovery mechanisms in case of interrupted system updates.
- Value range checks.
- Etc...

Operational Integrity Properties

- **Atomicity**: either all of the actions of a transaction are performed or none of them are .
- **Consistency**: of the data (prevent error and accidental data corruption).
- **Persistence**: the results of committed transactions are permanent (prevent reset attacks).
- **Durability**: data will not be lost over long periods of time:
 - a hard drive kept for more than 10 years will lose data, many CD/DVDROMs are chemically unstable , flash memory lasts only for a few years, flash memory and hard drives frequently fail during operation after 1-2 years, etc.

Criteria to Safeguard Integrity [Clark-Wilson]

A bit of Big Brother stuff... Very interesting in the financial sector (problem with rogue traders, corrupted employees, and very large losses).

- **Authentication**: the system must separately authenticate each user/employee, so that his actions can be monitored.
- **Audit (Logging)**: logs should contain many details.
- **Well-formed Transactions**: certain data can ONLY be manipulated by a restricted set of programs.
 - Interesting: Clark and Wilson postulated that it is THE DATA CENTRE controls (so external or complementary to these programs) would check if transactions are well-formed (how? See Clark-Wilson model).
- **Separation of Duty**: each user has programs to execute a part of the task
 - And again: Clark and Wilson postulated that it is THE DATA CENTRE that should ensure the principle of separation is applied.

Clark–Wilson Model



Clark-Wilson Integrity Model

Radically different model.

The Biba's model does NOT address the following problem:

whatever you do, principals in the system (e.g. employees of a bank)
have immense powers that can be used for fraud.

Clark-Wilson:

It is a model for real commercial environments.

- designed for systems with mathematical properties that are invariant over the time: in finance, banking, insurance, accounting,
- but also in production (accounting for raw materials and parts used) etc...

It also provides a model for an actual business implementation... Allows to ensure and certify that the bank is using proper procedures.

Preserves assets and accounting records of a business.

Clark-Wilson Philosophy

- Users don't have permissions for files. Never. They just have rights to run certain programs (!).
- Programs are certified,
 - under strict control regarding who can install them
 - must be inspected for proper construction (trustworthy)
 - duly authorized to access specific pieces of data and no other (limited trust)
 - only manipulated by users that are duly authorized
- Data objects don't have classifications either, their integrity is determined by rules saying which programs can access them.
- **The system** will do a lot of extra checks on the data and system state.

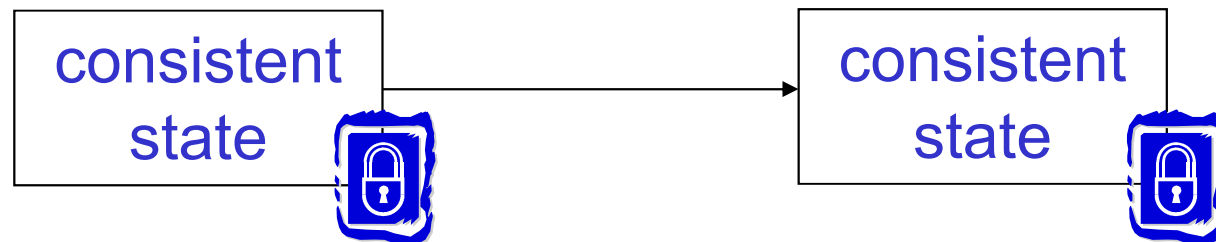
Clark-Wilson Integrity Model

In CW, Integrity is defined by a set of constraints

Data is in a **consistent** or **valid state** when it satisfies these constraints

- Example: Bank
 - Today: **TD** = deposits, **TW** = withdrawals, **TB** = balance
 - Yesterday: **YB** = yesterday's balance,
 - Integrity constraint: **$TB - TD - YB + TW = 0$**

A **well-formed transaction**:



Key Concepts in Clark-Wilson

Constrained Data Items = CDI

are those subject to strict integrity controls at any time,

- Example: bank accounts

Unconstrained Data Items = UDI

Like in Biba, two integrity levels,
with possibility to later transform a UDI into CDI (after doing some checks!). In Biba model, this promotion would be either forbidden (tranquility) or would require a process running at the higher (CDI or higher) integrity level, that would be able to access both CDIs and UDIs.

Here we will certify once for all each method (an operational business procedure) for upgrading data to CDIs. The notion of a highly-trusted subject (that could be a Trojan anyway) is replaced by a notion of a certified operation that is allowed by the system, and executed with extra checks by the system.

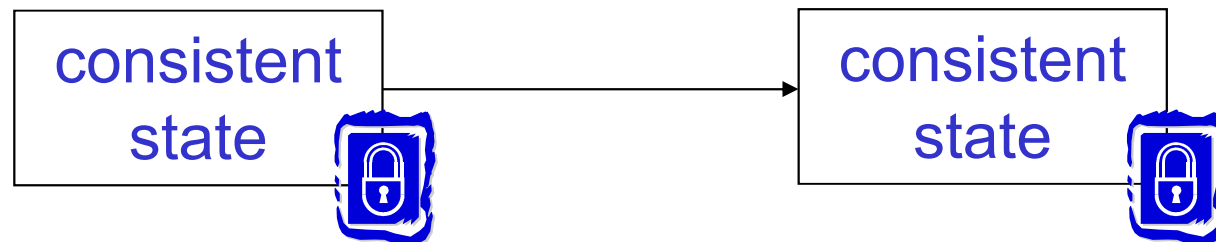
More Key Concepts

Constrained Data Items = CDI

- Integrity Constraints = are methods to check the integrity, say $TB - TD - YB + TW = 0$.
- Integrity Verification Procedures = IVP are procedures that test if CDIs conform to the integrity constraints

Transformation Procedures = TP are procedures that take the system from one valid state to another,

- Example: bank transactions.



Two Sorts of Rules in Clark-Wilson

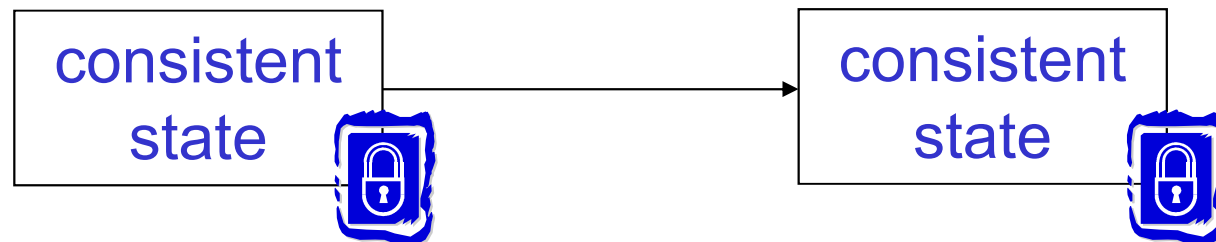
Certification Rules: CR

Example:

- I certify we received the product, so we can pay.
- Another person certifies that the expense was authorized.

One employee should not be able to make the company pay for phony invoices...

Enforcement Rules: ER



Certification Rules 1 and 2

CR1 When any IVP (int. verification procedure) is running, it must ensure all CDIs are in a valid state during its running time.

CR2 Each TP must,
for the associated set of CDIs,
transform these CDIs from a valid state into
a (possibly different) valid state

For this we define a relation **Certified C** that precisely says which CDIs are associated with which TP.

We say that $(f, o_{20}) \in C$ or f operates on CDI o_{20} .

For example $f = \text{ATM withdrawal}$
and $o_{20} = \text{number of } \pounds 20 \text{ bills left inside the ATM.}$

Enforcement Rule 1

Each f operates on several CDIs. And should not be applied to any other accounts (when f,o' not in C). So:

ER1 The system must maintain the **Certified** relations at all times, and must ensure that only TPs f certified to run on a CDI o can manipulate that CDI, i.e. only when $(f,o) \in C$.

Now, not everybody can execute a (valid) TP f .

Enforcement Rule 2

We define a relation* **allowed A** that actually will be defined as triples $(\text{user}, \text{TP}, \{\text{set of CDIs}\}) \in A$, and we say that $(u, f, O) \in A$, iff the user u is allowed to execute operation f that will potentially modify the state of all CDIs o with $o \in O$.

But this relation has to be certified at a higher level, for example by a CFO or chief accountant:

- ER2** The system must associate a user with each TP and a set of allowed CDIs for this user and this TP.
- The TP may access those CDIs on behalf of the associated user. The TP cannot access that CDI on behalf of a user not associated with that combination of TP and CDI.
 - System restrict access to f based on user ID following the relation **allowed A**.
 - system must enforce the **Certified** relation at the same time: all triples in **A** must be consistent with the relation **C**.

Users and Rules

- CR3** The allowed relations **A** must meet the requirements imposed by the principle of separation of duty.
- Informal rule... can be interpreted more or less strictly?
 - Ross Anderson wrote:
 - “[...] Clark-Wilson ducks the hardest question [...] it’s something that auditors tend to learn on the job, Companies’ internal controls tends to evolve over time in response to real or feared incidents”[...]
 - **ER4** is way more precise (later).
- ER3** The system must authenticate each user attempting to execute a TP. It is **not** required before any use of the system, but **is** required before manipulation of CDIs (when one performs one of the TP tasks). NOT A SINGLE SIGN-ON!

Logging and Append-Only CDI Objects

- CR4** All TPs must append enough information to reconstruct the operation to an append-only CDI.
- This CDI is simply a WORM log (or similar mechanism)
 - auditor needs to be able to determine what happened and when

Example of Actual Fraud [from Ross Anderson]:

A bank clerk in Hastings noticed that their system did not audit address changes. So he changed an address of a lady to his own, issued a credit card and a PIN, then changed it back again to the real address of the lady. He stole £8600 from her.

When the lady complained, she was not believed, the bank maintained that their systems are secure, and that the withdrawals must have been her fault (she let somebody take her card and her PIN, negligence). She had to pay.

Handling Untrusted Input

Unconstrained Data Items = UDI are data items of lower integrity, that are not really trusted, or those that we cannot check at the present moment...

Examples:

- the client has written on his form the total amount of cheques deposited is, £1234,56. But this cannot be verified before a human checks them.
- The client has declared that has a mortgage with this bank. This will be checked.
- The client has entered a number. The number will be accepted and validated or not according to some rules.

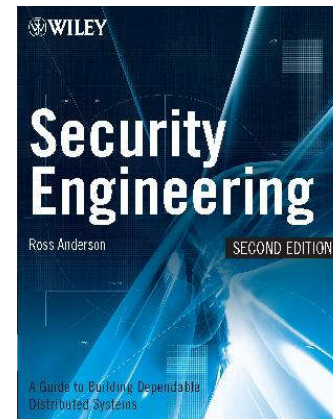
CR5 Any transformation either **rejects** the UDI (no change) or **transforms** it into a CDI (validating the data with some extra checks if possible).

Separation of Duty In Model

- ER4** Only the certifier of a TP (a certain procedure) may change the list of entities associated with that TP. No certifier of a TP, or of an entity associated with that TP, may ever have execute permission with respect to that entity.
- Enforces separation of duty with respect to **certified** and **allowed** relations.

To Learn More About Financial Systems:

Ross Anderson,
Chapter 10,
Banking and Bookkeeping.



<http://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c10.pdf>

****Applications of Clark-Wilson

SWIFT, worldwide inter-bank money transfer systems
(>50 % of the world's GDP is transiting here) is considered as a system
enforcing all these principles,
though strictly speaking Clark-Wilson was published later...

Examples of Fraud



Example 1 [from Ross Anderson]

A clerk at the Australian national education authority has created a fictitious school, with staff. He collected all salaries for all the school employees.

How he was caught?

Clark-Wilson: somebody has compared if the number of schools in two different systems was the same, and started checking schools one by one.



Interesting Statistics

[after Ross Anderson] In English-speaking banking industry,

- 1% of all staff are sacked each year because they have embezzled some money (usually small amounts).
- no method known to predict which staff will go bad.
 - it seems that people turn bad due to some specific reasons, such as alcohol, drugs, gambling or divorce, but these really cannot be predicted in advance.
- if it is a senior people that go wrong, banks go into great lengths to hide this fact from the public opinion
 - Example: Barings Bank [1762-1995] was the oldest bank in London. Collapsed totally and disappeared after its star super trader Nick Leeson lost £827 million in speculation he was actively hiding. Book: **Rogue Trader**
 - he hold two positions of floor manager and head of settlement.
 - Should be held by two different people (Clark Wilson CR3).

When Clark-Wilson Will Fail



Good Bye \$\$\$...



Clark-Wilson type of controls can stop one corrupted employee.
Larger conspiracy will always work(!!!).

Example [from Ross Anderson]

- Paul Stubbs, a password reset clerk at HSBC conspired with “somebody” inside one of the HSBC’s customers, AT&T to change the password that AT&T used to access their bank account remotely.
 - the password was reset
 - “somebody” used it to transfer \$20 M to offshore bank accounts
 - the money was never recovered!
 - a vulnerable young man, the court took mercy on him and he got away with 5 years...
 - now if he still has the money (who knows?), for each hour spent in prison, he will earn 600 dollars

“if a senior people go wrong”? The Bigger, The Better



In 1938 a large US drug and chemicals company has collapsed.

20 % of recorded assets and inventory was nonexistent:

- the President + three brothers with key positions.
- they have set up several fake companies abroad + bogus shipping company + a fake bank.
 - the auditors did not notice anything, all the accounts looked OK.
- the Clark-Wilson model cannot check if assets (money in banks, stocks of raw material) do really exist...

Hybrid Policies



Hybrid Policies

Combine integrity and confidentiality.

Two examples of policies that can be used in practice, adapted or/and applicable to a specific context:

- Chinese Wall policy:
 - financial sector, centralized.
- British Medical Association (BMA) policy,
 - decentralized.

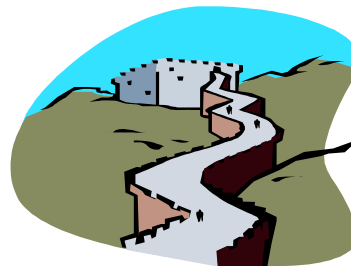
British Medical Association (BMA) policy

- Decentralized,
- Goals:
 - confidentiality,
 - avoid data aggregation
- Rules
 - one doctor can add another doctor (referral)
 - but not anybody that already has access to a large number of records

To study at home, see Ross Anderson Security Engineering, chapter 9.2.3.

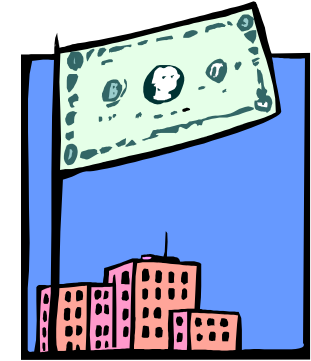
Chinese Wall

David FC. [Brewer](#) and Michael J. [Nash](#).
“The Chinese Wall Security Policy.” in IEEE SSP 1989.



Chinese Wall Model

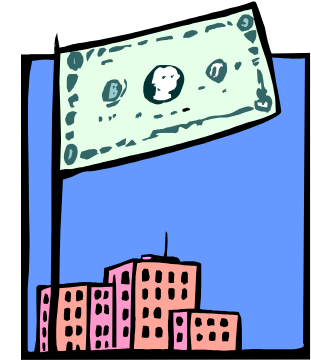
a.k.a. **Firewall**, or **Brewer-Nash** model [1989],
(\neq broader terms (used by corporate lawyers):
Ethical Wall, or **Cone of Silence Wall**,
or **Paper Wall** (this last one is ironic))



Applications:

stock exchange, trading house, ratings agency,
investment bank, hedge fund,
law firm, advertisement agency, etc.

Chinese Wall Model



Brewer-Nash, UK, 1989,

Based on UK Laws and regulations concerning securities and handling of conflicts of interest.

(keys ideas go back to US regulatory responses to the financial crisis of 1929.)

- as important for the financial sector as BLP is for the military.
- criminal charges, and/or astronomical fines if rules are not applied (!).

Main goal: prevent conflicts of interest.

- Example: Can I now use my privileged knowledge about client A to execute my paid assignment with client B? This is not allowed!

Chinese Wall and Related Concepts

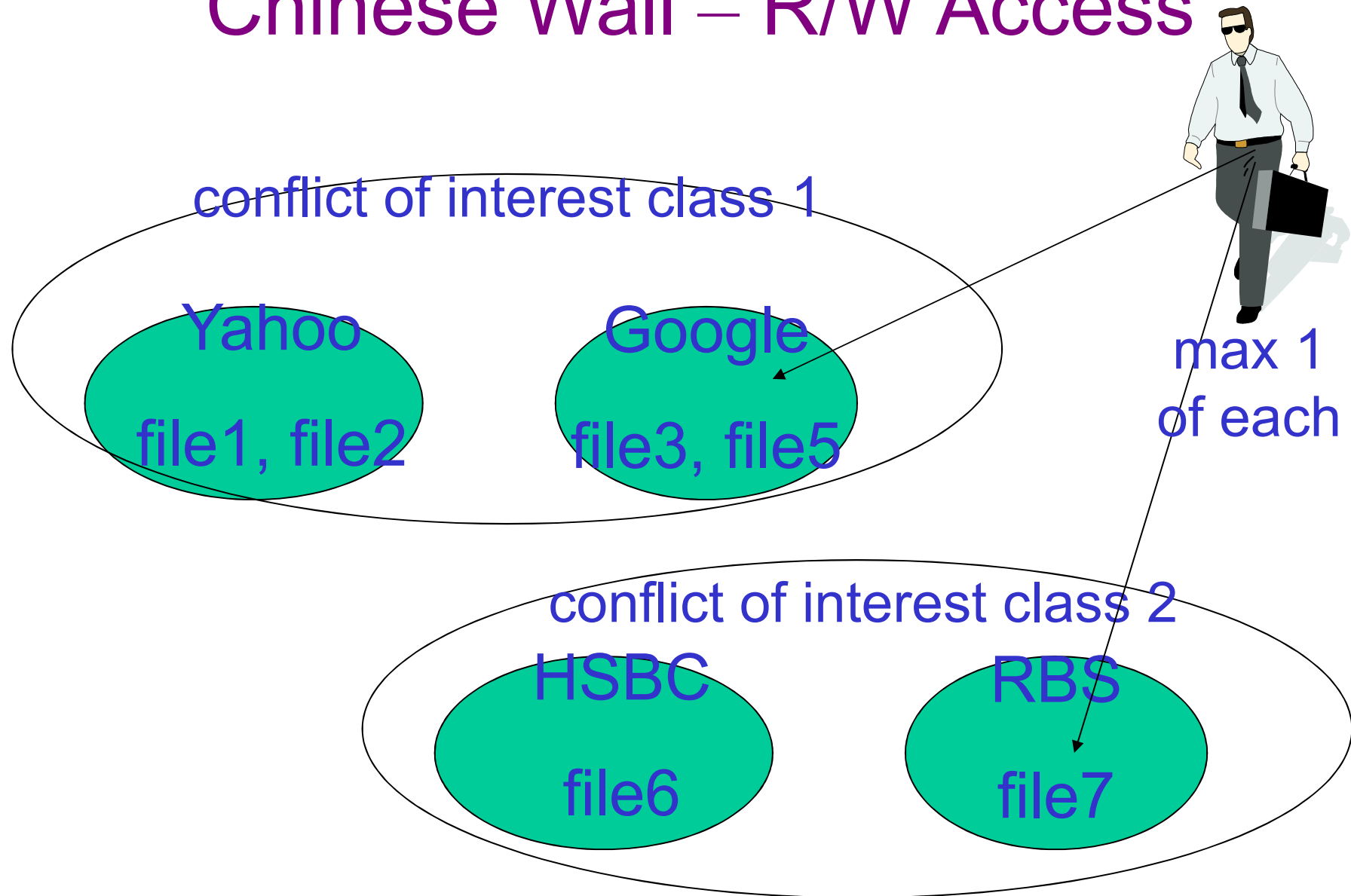
- These are **ORCON** = Originator-Controlled access policies.
 - The originator can determine who can access the data and how.
- These are also **Role-Based** policies = **RBAC**.

Chinese Wall Model:

- The policies are not 100 % formalized.
- Objects don't have security labels like in BLP or Biba.
- Dynamic separation of duty:
 - subjects have free choice of data to access or to work on,
 - but their choices affect them later.
 - subjects under "either or, but not both" rules w.r.t objects.
 - current rights depend on current and past data that one employee already has or had access to:
 - once you worked for Pepsi, you cannot work for Coca-Cola !



Chinese Wall – R/W Access



Transitive Closure

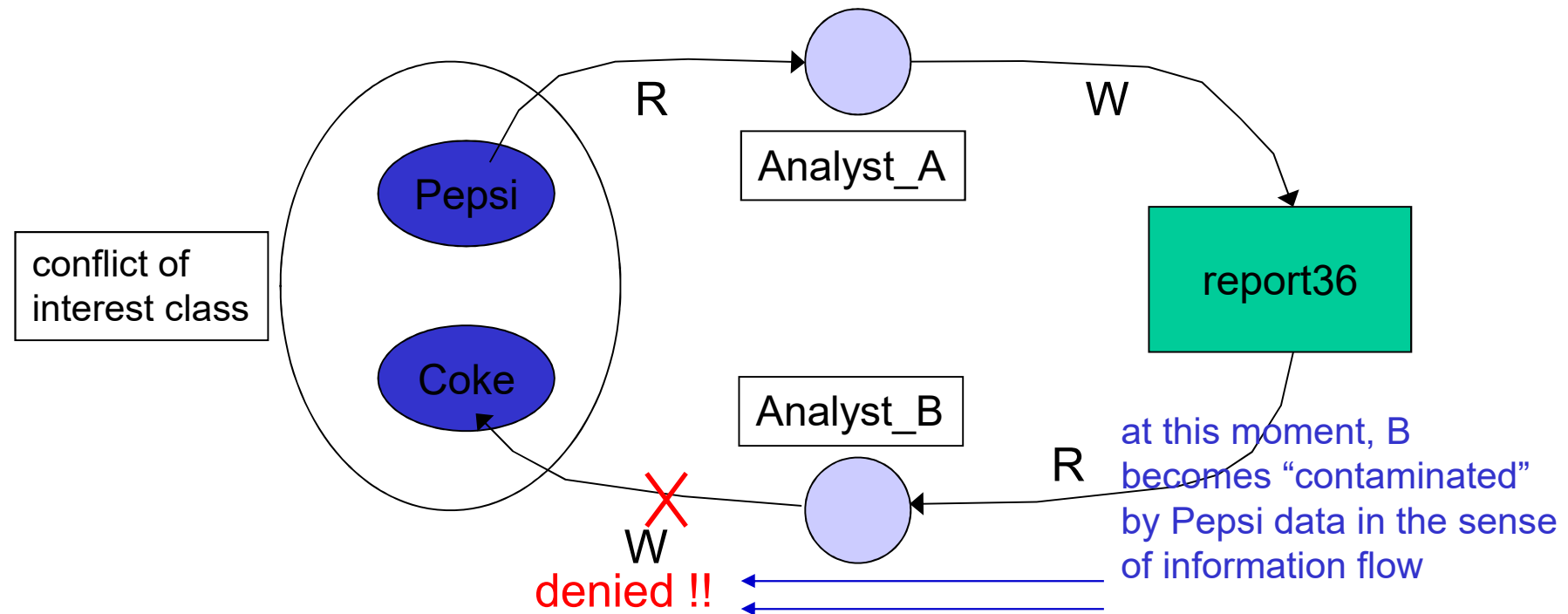
- This model is strict: prevent potential data flows
 - Can be implemented based on graphs and transitivity:
 - We need to compute the "transitive closure" of all data flows
 - » Definition: $a \rightarrow b$ in the transitive closure of G , if there is a path from a to b in the graph G of all possible information flows allowed (this includes paths of length 0).
 - » See next two pages
 - » Then we need to prevent that the transitive closure leads to a flow within one conflict of interest class.



Chinese Walls – Write Rule

Write access granted if no other object can be read that:

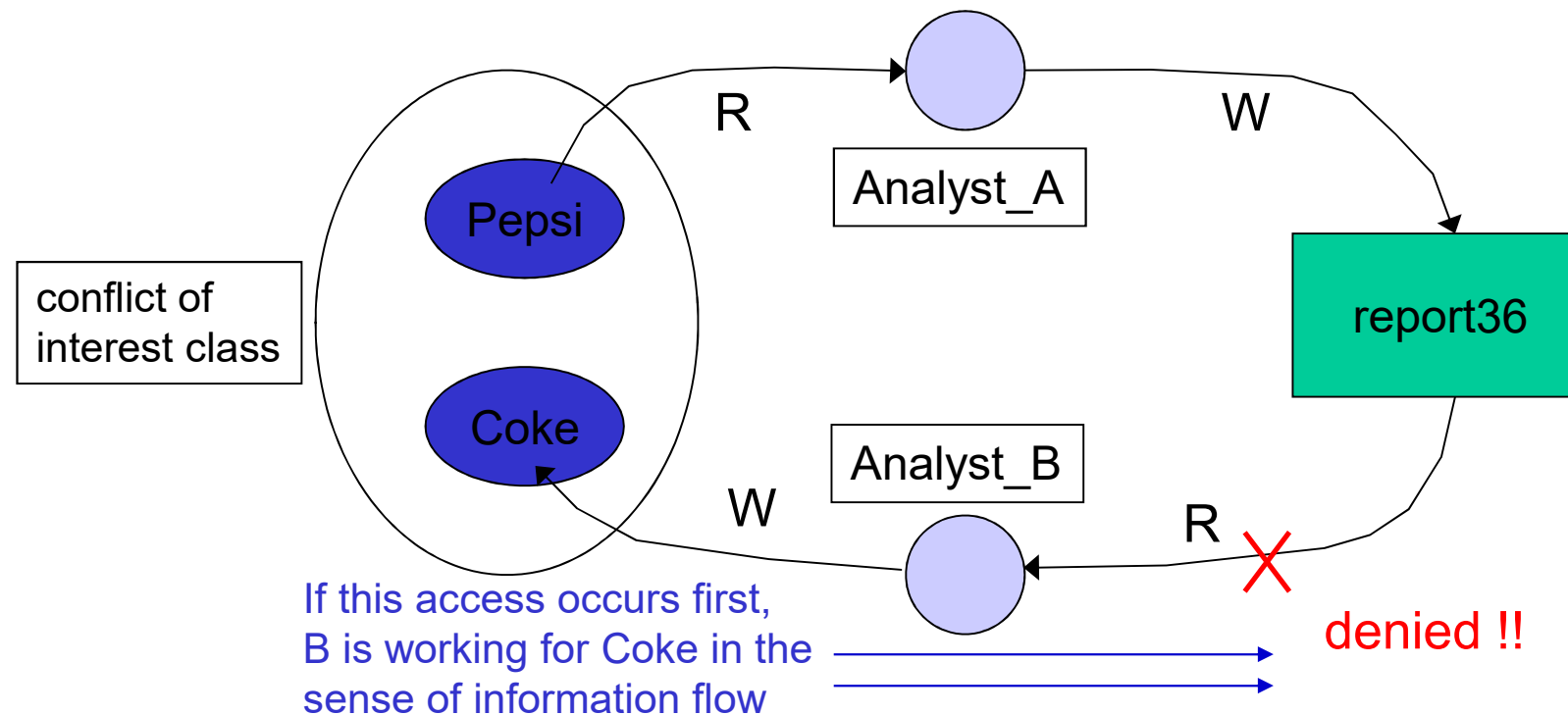
- Belongs to a competing company dataset
- Contains un-sanitized information



Chinese Walls – Read Rule

Read access granted if no other object can be written that:

- Belongs to a competing company dataset
- Contains un-sanitized information



Chinese Walls – What Is Allowed

- **Sanitization** and anonymization of data will be applied to serve legitimate business needs.

In the original Brewer-Nash paper: the analyst is ..

however he is also to draw some “**general market information**”...

RBAC



Role-Based Policies [RBAC]

Access by users/subjects/principals to objects/processes is mediated by roles.

Roles are “aggregated privileges” related to the execution of certain business activity.

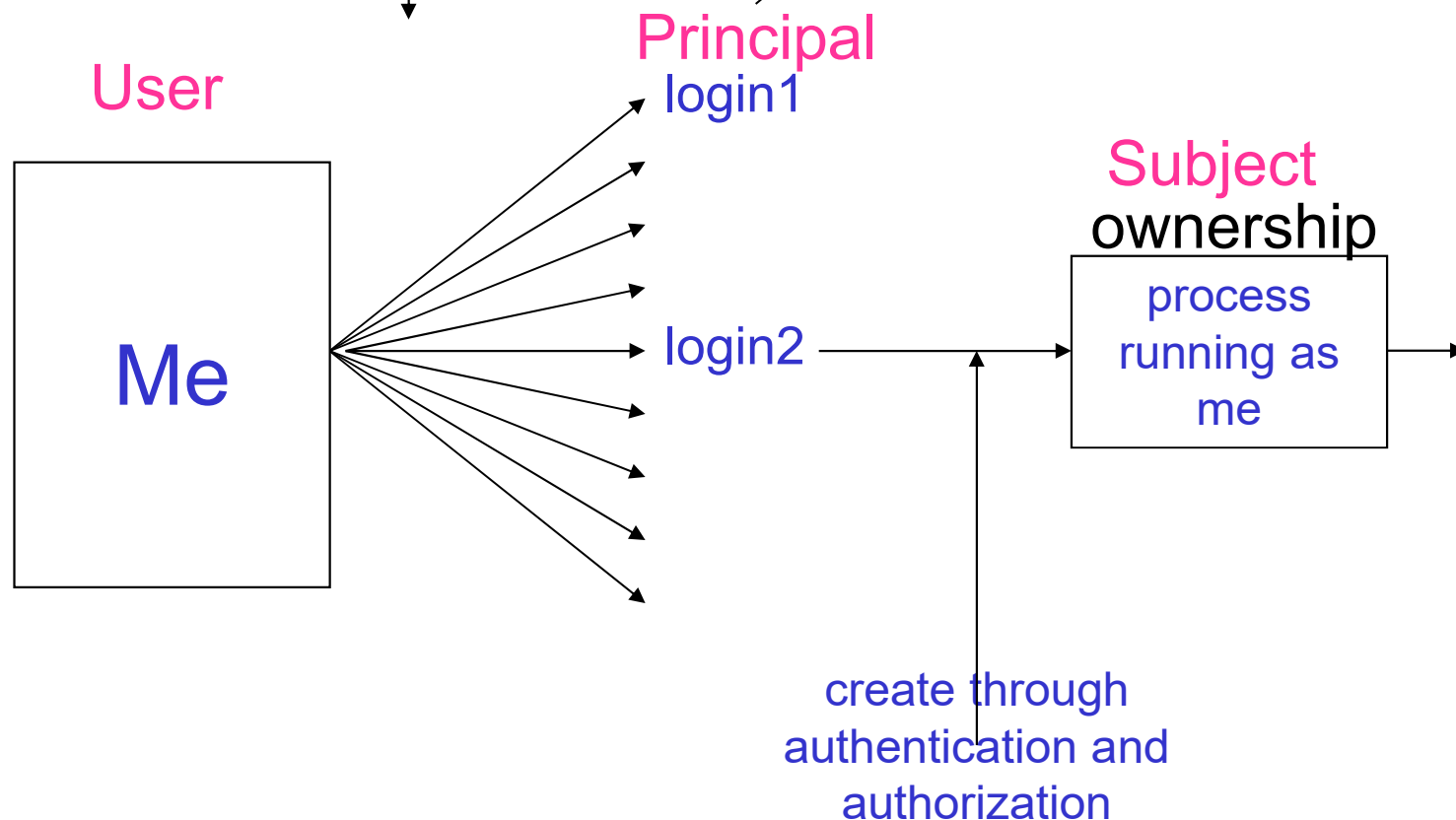
- users must activate their roles
 - privileges not valid if the role is not active

- Remark: Anonymity is possible: one is identified by its role alone. But not automatic, most systems will not have very good anonymity, extra care will be needed...
 - For example, if we make a secure connection from home for a given role, does the method of securing the connection expose our home IP address nevertheless? Probably it does...

Remember our Most General Slide?

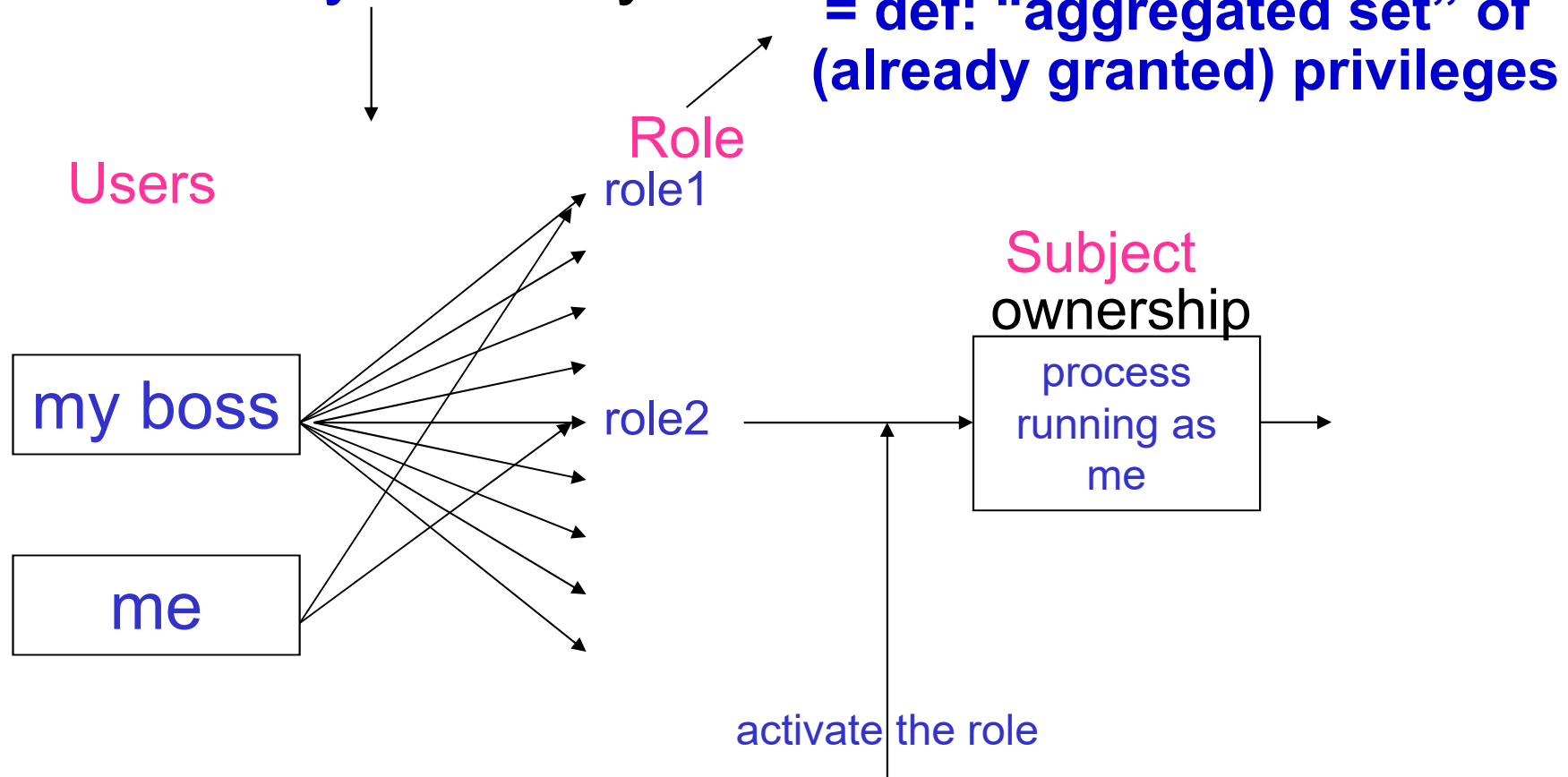
One to Many.

= def: Unit of Access Control
and Authorization



Here it is Different:

Many to Many.



Groups \neq Roles

Groups: sets of users.

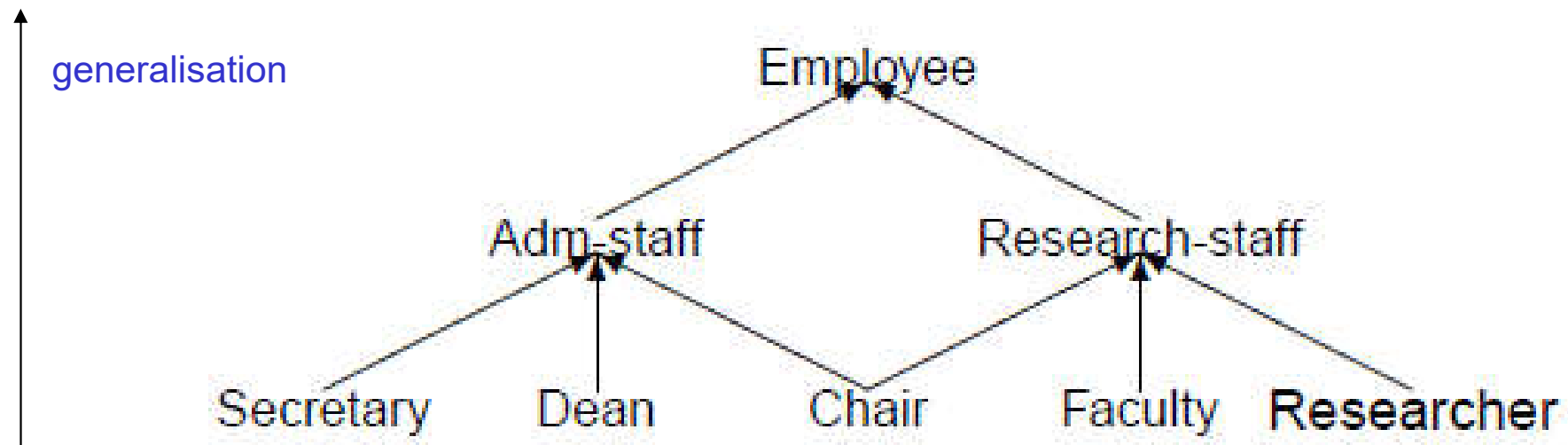
Roles: sets of privileges.

Also implies some group of people that are allowed to take this role, but the members of this group are not fixed, we can add and remove members.

Roles are natural in business / organizations,
easy to understand,
quite stable in attributions
(tranquility, anomaly detection).

Role Hierarchies

Role hierarchy defines **specialization** relationships



Hierarchical relationship \Rightarrow authorization propagation.

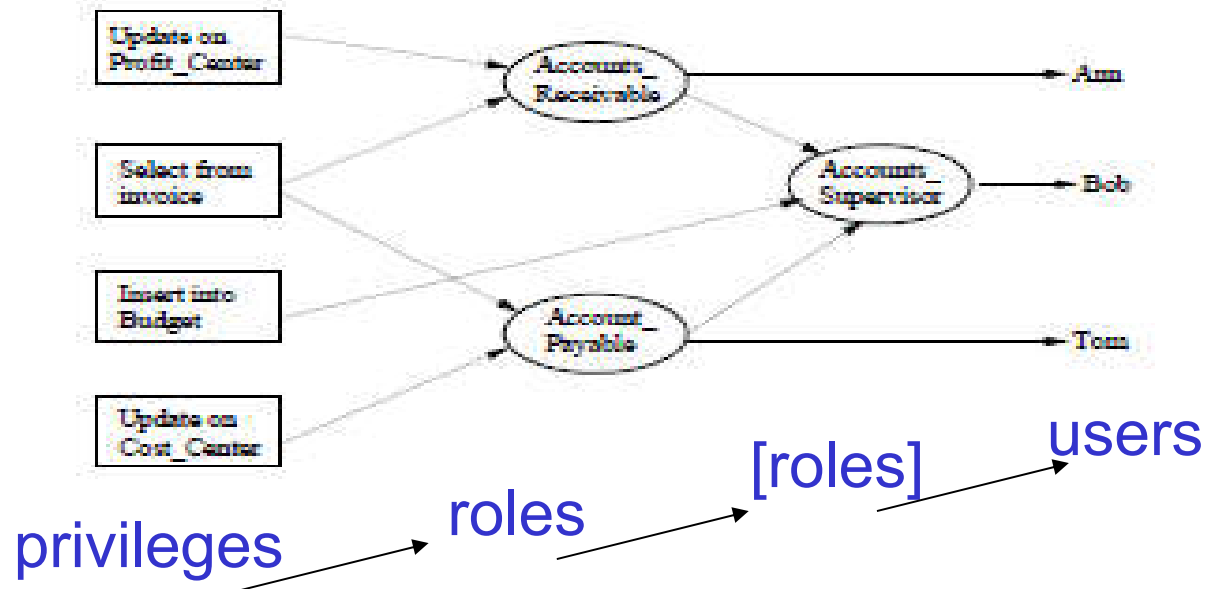
- User inheritance: is also member of a higher role
- Activation inheritance: if user can activate a role, it can activate all its generalisations
 - Question: why would I activate the role of Employee if I can be logged as chair?
- Permission inheritance: higher roles get all access permissions for lower roles.

Applications of RBAC



SQL and RBAC

In SQL privileges can be grouped in roles that can be assigned to users or to other roles (nested roles).



SQL Administrative Policies

user who creates a table is its owner

- and can grant authorizations to others,
 - including the authorizations to grant (can build chains of successive authorizations)

users can also revoke authorizations they have granted (and only those).

what if the user is deleted or revoked?

- all authorizations he granted are revoked
 - optional: **recursive = with cascade**: also revoke authorizations granted further (delete the whole chain).

Fine Grained Policies And Exceptions



Exceptions

Very widely used method.

We will call it **Expanding Authorizations**:
means adding extra possibilities through
exceptions to rules...

Can be based on

- user groups/sub-groups,
- conditions of time, location, data type, past history of requests
 - Example: an employee of a bank can usually access a fixed limited number of client records per day, prevents employees selling these data in bulk (has happened many times!)

Types of Policy

open policy: if not forbidden, can be executed,

=**black list** concept,

- like virus signatures on a PC

closed policy: must be explicitly allowed,

=**white list** concept,

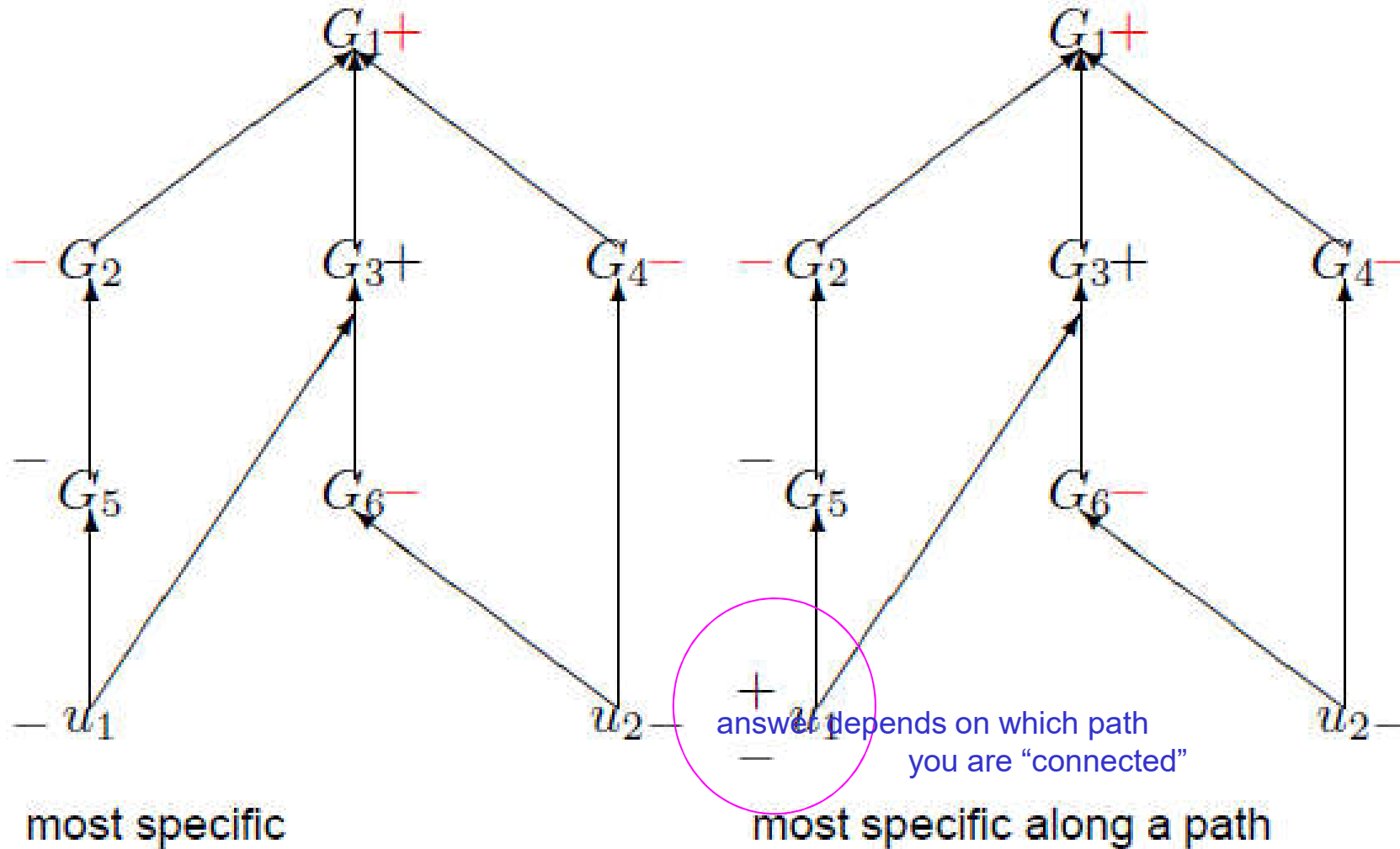
- like Microsoft-approved drivers,
- like Nokia-approved apps for Symbian phones

Conflict Resolution Policies

What if we have a Yes and a No? Several methods:

- **denials take precedence** (fail-safe method)
- ordering/prioritising permissions, for example **strong** and **weak** ones, a strong one can override weak one
- making them **grantor-dependent**
- making them **dependent on parameters**, again time, location, data type, past history of requests
- **most specific takes precedence**:
 - example: all academic staff except Bob.
- **“most specific along a path”**: what indeed is more specific if we have multiple hierarchies???
 - the decision will depend on the current role of Bob with which he is connected or acting...

most specific vs. most specific along a path



Example of Application

Apache HTTP servers,
look inside the `.htaccess` file:

Example:

Order Deny,Allow

Deny from all

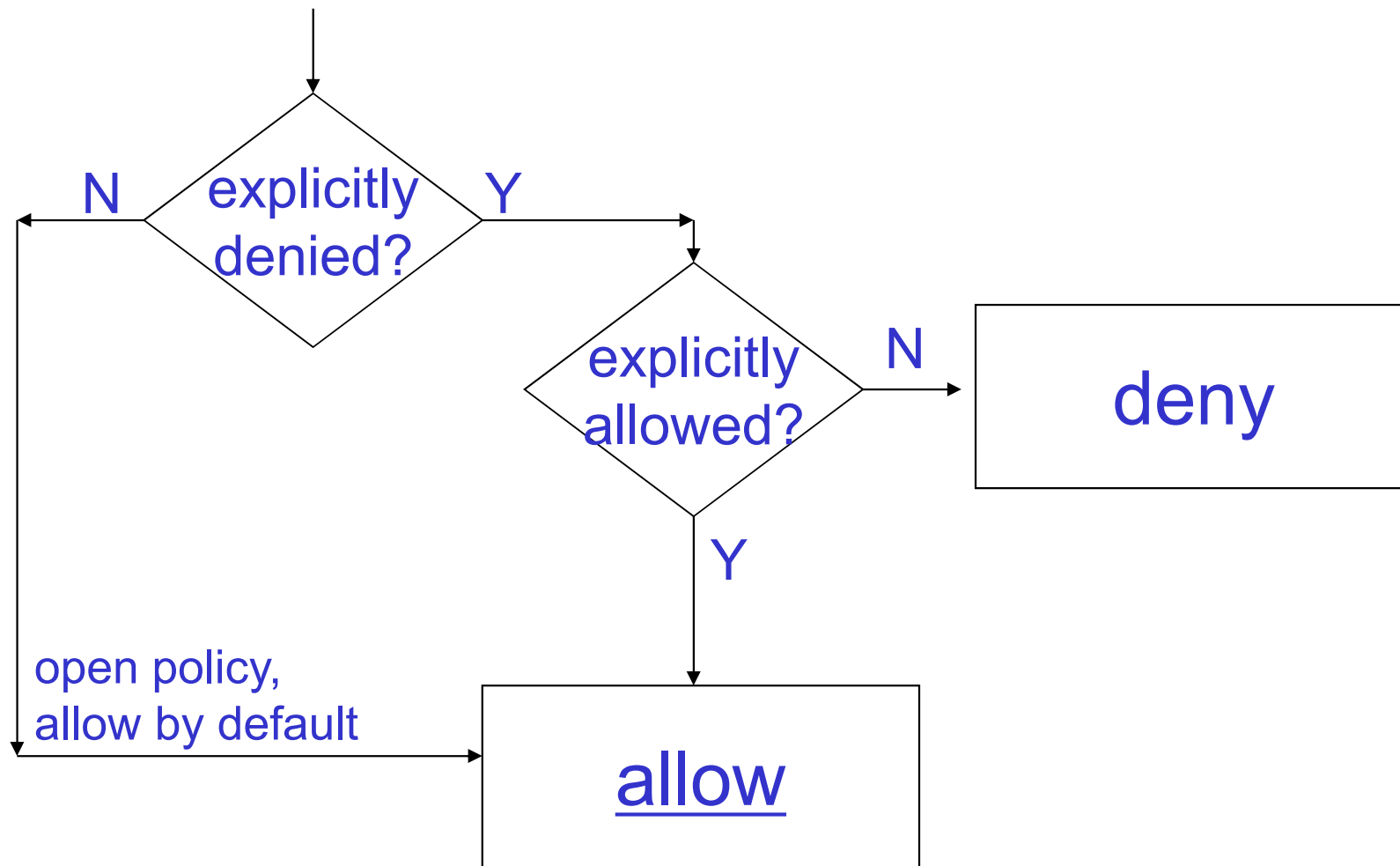
Allow from ucl.ac.uk

Allow from 10.0.0.0/255.0.0.0 ← means 10.____

Allow from 10.0.0.0/8

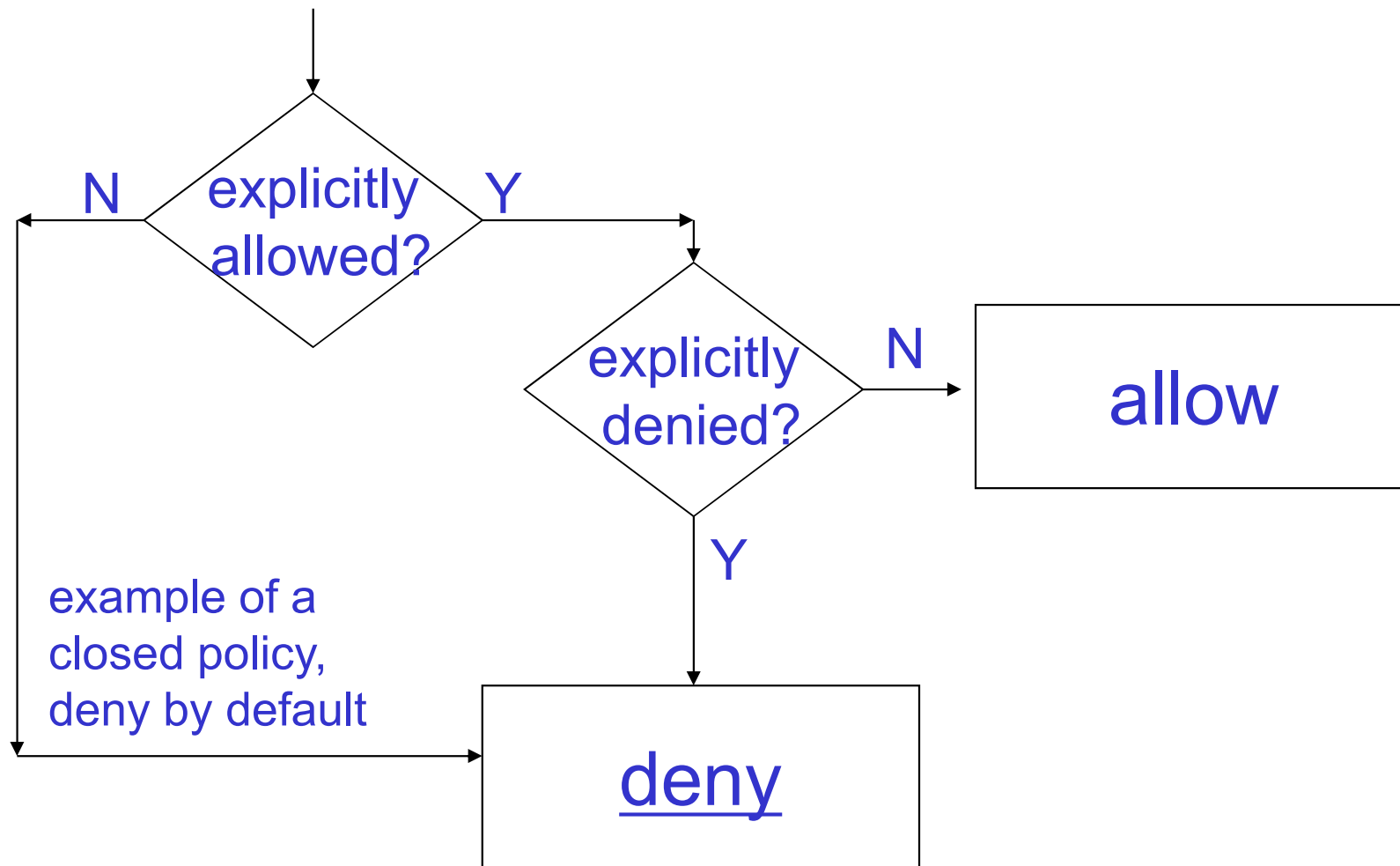
Deny,Allow

allow by default, allow overrides deny

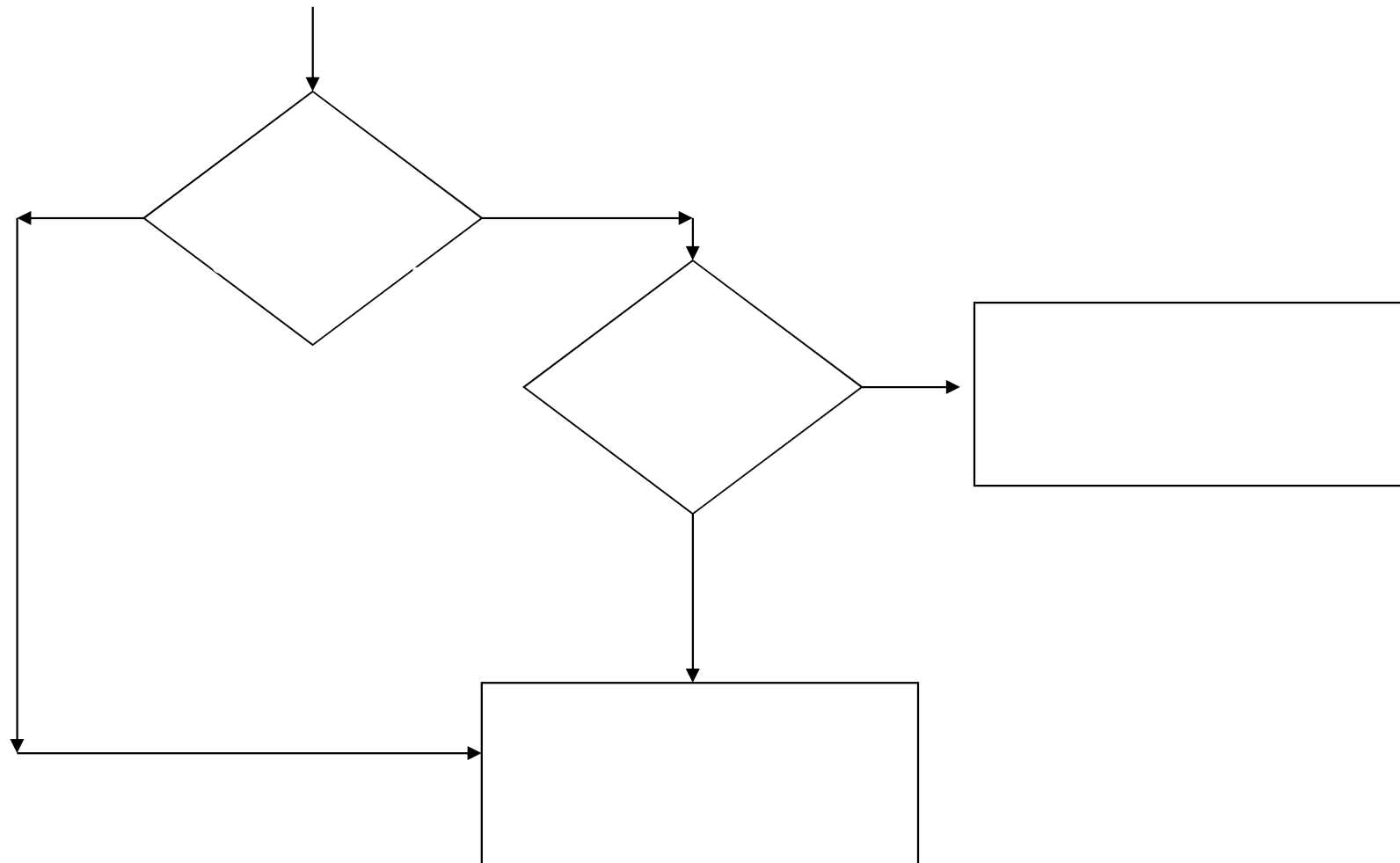


Allow, Deny

deny by default, deny overrides allow



**Mutual Failure – the Same, deprecated



Quiz



Quiz

What is a

- A security policy for an organisation? For a system?
- A “broad” security mechanism? opposite of it?
- An order relation (RAT)
- Give an example of a totally ordered set.
- Give an example of an order that is NOT a total order.
- GLB? The dual notion?
- Lattice?
- reference monitor?
- what is the meaning of the “exe” permission for directories in Unix?
- DAC?
- What is an Ambient Authority system?
- Can one make a program that always says Y when a file contains a virus?
- MAC?
- How exactly do we define is the BLP product lattice?
- BLP model? The dual model of BLP? What is the purpose of the dual model?
- How can one implement an append only policy for a file in hardware? WORM?
- Is the Chinese wall model sensitive to random events and how?

Perspectives



Military vs. Commercial

Military data security:
focus on secrecy,
prevent **leaks**.



Commercial data security:
integrity and authenticity:
prevent **fraud**.





One Key Insight

Is that
confidentiality
and integrity
are really



TWO INDEPENDENT dimensions.

One can be very good,
the other very bad
at the same time (!!!).

Integrity

a system has its integrity if can be trusted.

It is like the number of medicines that people in a rich country buy, each year the industry invents a new condition that people they did not know that they had...

The notion of Integrity is likely to expand forever...

Remark [Mark Ryan]

- Confidentiality is neither verifiable, nor correctable.
 - can hardly restore privacy which was once lost.
- I+A are verifiable, can be corrected.
 - Can fix it later, correct errors, roll-back fraudulent transactions, etc... etc...