IS 3350 -Doctoral Seminar
focus:
Security and Privacy Assured Health Informatics

Overview of Access Control Models

James Joshi
Associate Professor, SIS, Pitt
Sept 3, 2015

Access Control

- Discretionary Access Control (DAC)
  - Owner determines access rights
  - Typically *identity-based access control*: Owner specifies other users who have access
- Mandatory Access Control (MAC)
  - Rules specify granting of access
  - Also called *rule-based access control*
- Originator Controlled Access Control (ORCON)
  - Originator controls access
  - *Originator need not be owner!*
- Role Based Access Control (RBAC)
  - Identity governed by role user assumes

Discretionary Access Control (DAC)

- Subjects have ownership over objects
  - A subject can pass access rights to other subjects at his discretion
- Highly flexible and currently most widely used
- Not appropriate for
  - high assurance systems, e.g., a military system
  - Many complex commercial security requirements
- “Trojan horse” problem

DAC: Access Control Matrix model Background

- Access Control Matrix
  - Captures the current protection state of a system
- Butler Lampson proposed the first Access Control Matrix model
- Refinements
  - By Graham and Denning
  - By Harrison, Russo and Ulman – with some theoretical results

Protection System

- Subject (*S*: set of all subjects)
  - Eg.: users, processes, agents, etc.
- Object (*O*: set of all objects)
  - Eg.:Processes, files, devices
- Right (*R*: set of all rights)
  - An action/operation that a subject is allowed/disallowed on objects
  - Access Matrix *A*: $a[s, o] \subseteq R$
- Set of Protection States: (S, O, A)
  - Initial state $X_0 = (S_0, O_0, A_0)$

Primitive commands (HRU)

Create subject s	Creates new row, column in ACM; s does not exist prior to this
Create object o	Creates new column in ACM o does not exist prior to this
Enter r into $a[s, o]$	Adds r right for subject s over object o Ineffective if r is already there
Delete r from $a[s, o]$	Removes r right from subject s over object o
Destroy subject s	Deletes row, column from ACM;
Destroy object o	Deletes column from ACM

Fundamental questions

- How can we determine that a system is secure?
  - Need to define what we mean by a system being “secure”
- Is there a generic algorithm that allows us to determine whether a computer system is secure?

What is a secure system?

- A simple definition
  - A secure system doesn't allow violations of a security policy
- Alternative view: based on distribution of rights
  - Leakage of rights:
    - Assume that *A* representing a secure state does not contain a right *r* in an element of A.
    - A right *r* is said to be leaked, if a sequence of operations/commands adds *r* to an element of *A,* which did not contain *r*

What is a secure system?

- Safety of a system with initial protection state X_o
  - Safe with respect to r: System is *safe with respect to r* if *r* can never be leaked
  - Else it is called unsafe with respect to right *r*.

Decidability Results *(Harrison, Ruzzo, Ullman)*

- Theorem:
  - Given a system where each command consists of a single *primitive* command (mono-operational), there exists an algorithm that will determine if a protection system with initial state X_0 is safe with respect to right *r*.

- process *p* creates file *f* with owner *read* and *write* (*r*, *w*) will be represented by the following:

  Command *create_file*(*p*, *f*)
    Create object *f*
    Enter *own* into *a*[*p*,*f*]
    Enter *r* into *a*[*p*,*f*]
    Enter *w* into *a*[*p*,*f*]
  End

Command *make_owner*(*p*, *f*)
  Enter *own* into *a*[*p*,*f*]
End

- Mono-operational: the command invokes only one primitive

Decidability Results *(Harrison, Ruzzo, Ullman)*

- It is **undecidable** if a given state of a given protection system is safe for a given generic right
- For proof – need to know Turing machines and halting problem
  - **REDUCE** TM problem to HRU problem

- Other general models:
  - Take-Grant Model; Schematic Protection Model, etc.

Other theorems

- The safety question for biconditional monotonic protection systems is undecidable
- The safety question for monoconditional, monotonic protection systems is decidable
- The safety question for monoconditional protection systems with **create, enter**, **delete** (and no **destroy**) is decidable.
- Observations
  - Safety *is undecidable for the generic case*
  - Safety *becomes decidable when restrictions are applied*

Some Existing Models

- Abstract models
  - HRU's Access Control Matrix
  - Schematic Protection Model and variation
- Mandatory
  - Confidentiality model - Bell-LaPadula
  - Integrity model
    - Biba, Lipner's, Clark-Wilson
- Hybrid
  - Chinese wall

Mandatory Access Control (MAC)

- Subjects/objects have security levels forming a lattice
- Flow of information is restricted.
  - Example: *(no-readup), (no-writedown)*
- Well-know MAC model is the Bell-LaPadula model

“No Read Up”

- Information is allowed to flow *up*, not *down*
- *Simple security property:*
  - *s* can read *o* if and only if
    - $l_o \le l_s$ and
    - *s* has read access to *o*
- **property*
  - *s* can write *o* if and only if
    - $l_s \le l_o$ and
    - *s* has write access to *o*

Integrity Policies

- Biba's Model: Strict Integrity Policy (dual of Bell-LaPadula)
  - s **r** $o \Leftrightarrow i(s) \le i(o)$ (no read-down)
  - s **w** $o \Leftrightarrow i(o) \le i(s)$ (no write-up)
  - s_1 **x** $s_2 \Leftrightarrow i(s_2) \le i(s_1)$

- Low-Water-Mark Policy
  - s **w** $o \Leftrightarrow i(o) \le i(s)$ prevents writing to higher level
  - s **r** $o \Rightarrow i'(s) = min(i(s), i(o))$ drops subject's level
  - s_1 **x** $s_2 \Leftrightarrow i(s_2) \le i(s_1)$ prevents executing higher level objects

- Ring Policy
  - s **r** o allows any subject to read any object
  - s **w** $o \Leftrightarrow i(o) \le i(s)$ (same as above)
  - s_1 **x** $s_2 \Leftrightarrow i(s_2) \le i(s_1)$

Other policies

- Clark-Wilson Model
  - Transactions oriented; includes SoD constraints
- Lipner's Model
  - Integrates BLP and Biba models

Requirements of Commercial Integrity Policies (Lipner's)

1. Users will not write their own programs, but will use existing production programs and databases.
2. Programmers will develop and test programs on a nonproduction system; if they need access to actual data, they will be given production data via a special process, but will use it on their development system.
3. A special process must be followed to install a program from the development system onto the production system.
4. The special process in requirement 3 must be controlled and audited.
5. The managers and auditors must have access to both the system state and the system logs that are generated.

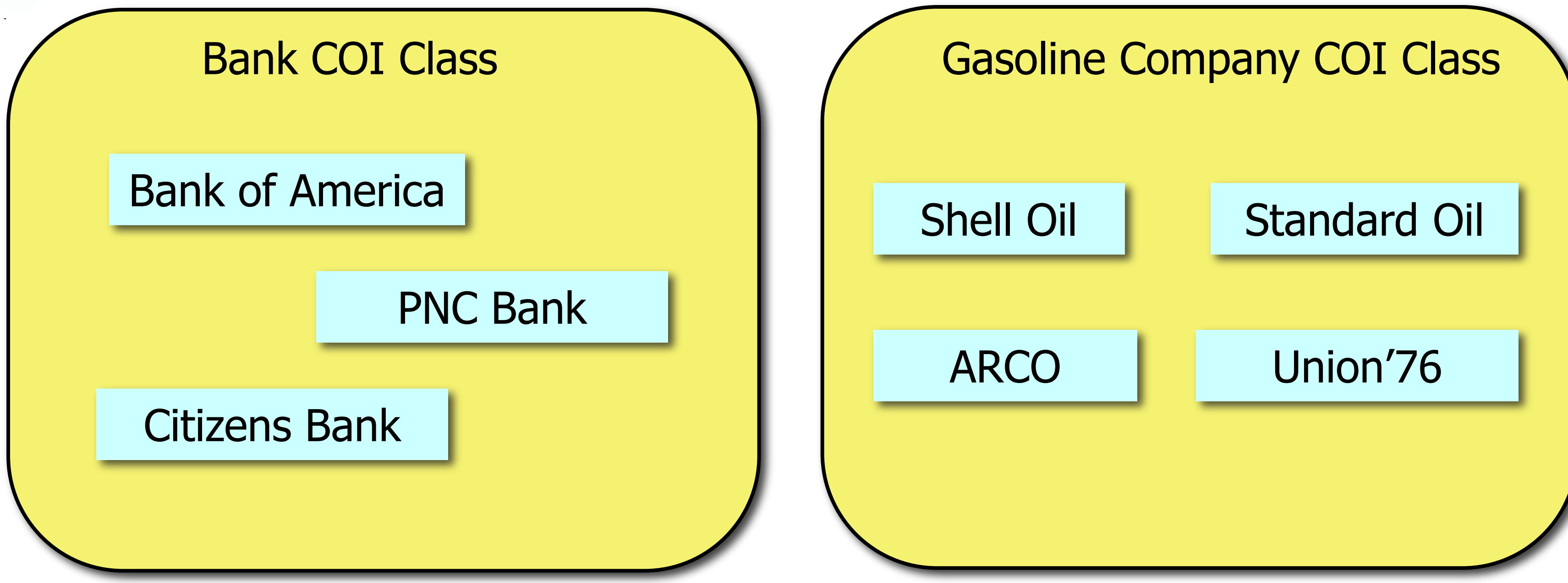
Clark-Wilson

- Transaction based – integrity verification function
- Commercial firms do not classify data using multilevel scheme
- They enforce separation of duty
- Notion of certification and enforcement;
  - enforcement rules can be enforced,
  - certification rules need outside intervention, and
  - process of certification is complex and error prone

Chinese Wall Model

- Supports confidentiality and integrity
  - Information flow between items in a Conflict of Interest set
  - Applicable to environment of stock exchange or investment house
- Models conflict of interest
  - *Objects*: items of information related to a company
  - *Company dataset* (CD): contains objects related to a single company
    - Written *CD*(*O*)
  - *Conflict of interest class* (COI): contains datasets of companies in competition
    - Written *COI*(*O*)
    - Assume: each object belongs to exactly one *COI* class

Example



Bank COI Class
Bank of America
PNC Bank
Citizens Bank
Gasoline Company COI Class
Shell Oil
Standard Oil
ARCO
Union'76

CW-Simple Security Property (Read rule)

- CW-Simple Security Property
  - *s* can read *o* *iff* any of the follo
    - $\exists\, o' \in PR(s)$ such that $CD(o$
    - $\forall\, o', o' \in PR(s) \Rightarrow COI(o') \neq COI(o)$, or
    - *o* has been "sanitized"

    ($o' \in PR(s)$ indicates o' has be

- CW-*- Property
  - *s* can *write* *o* iff the followin
    - The CW-simple security condition permits S to read O.
    - For all unsanitized objects o', s can read o' $\Rightarrow CD(o') = CD(o)$

Allow read on CD items if other items from CD has been read

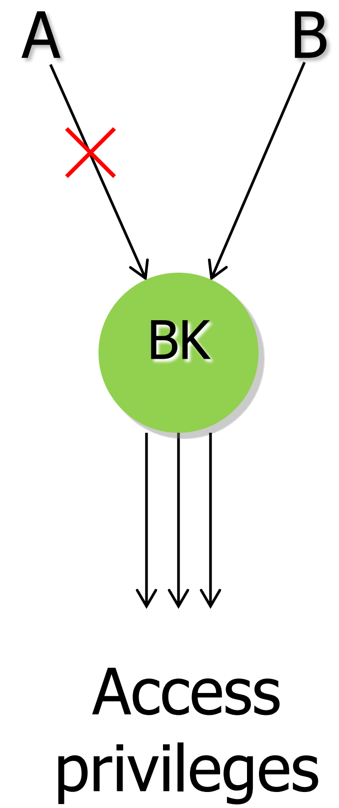
Allow read on CD items if this CD is not in COI with CD of other items read

If simple security property allows read to it &
All other items that he can read also belongs to it

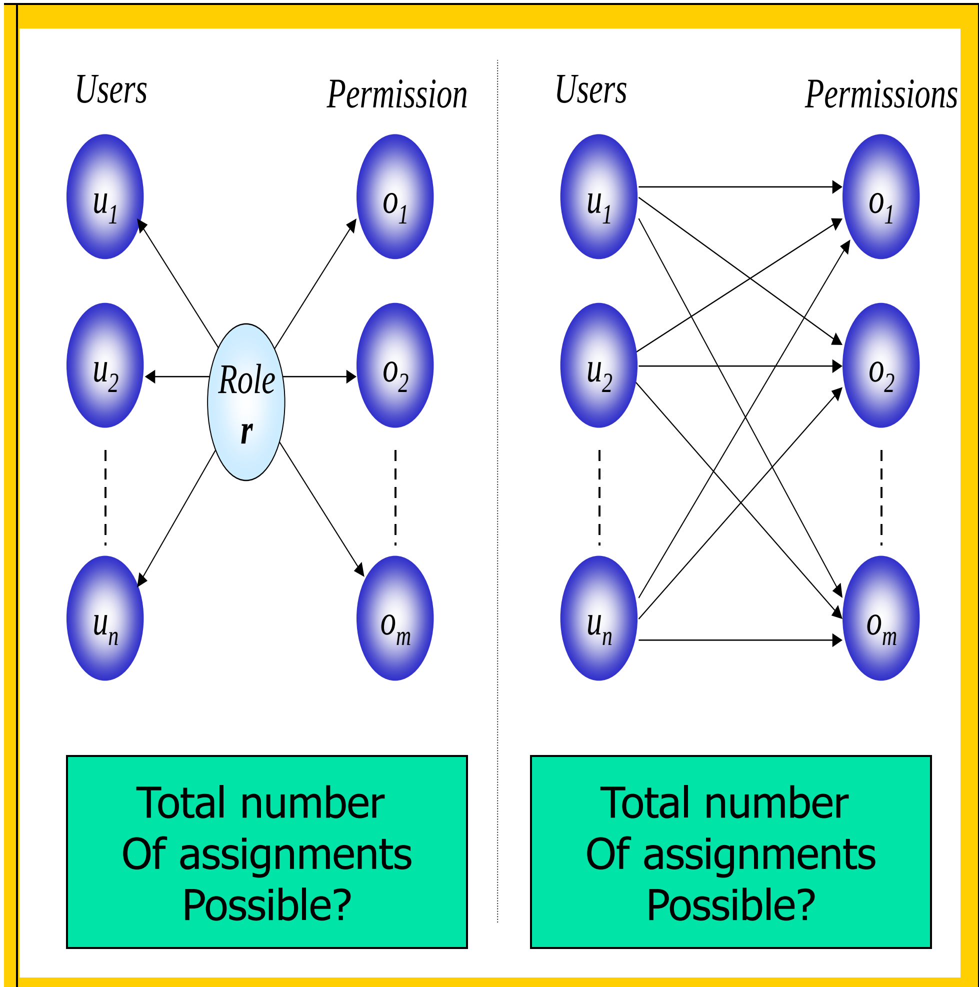
Role-Based Access Control

RBAC: Role Based Access Control

- Access control in organizations is based on "roles that individual users take on as part of the organization"
- A role is "is a collection of permissions"



RBAC


Users
Permission
u_1
u_2
u_n
Role
r
o_1
o_2
o_m
Users
Permissions
u_1
u_2
u_n
o_1
o_2
o_m
Total number Of assignments Possible?
Total number Of assignments Possible?

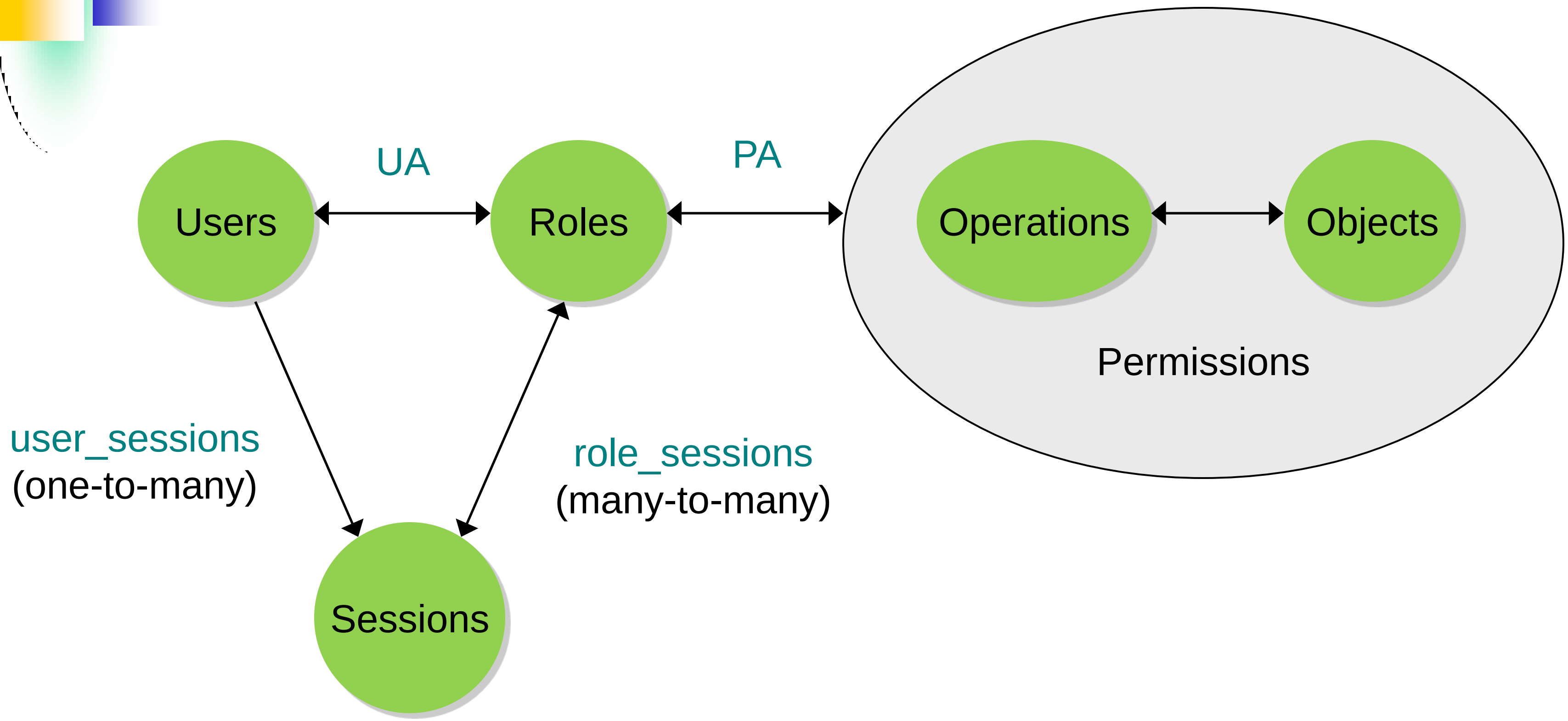
RBAC standard

- Standards efforts
  - ACM RBAC workshops – in 90s
  - NIST Standard proposed in 2001 (TISSEC)
  - XACML Profile for RBAC
  - ANSI INCITS 359-2004 RBAC standard in 2004
- The ANSI standard consists of two parts
  - Reference Model
  - System and Administrative Functional Specification

ANSI RBAC standard – Reference Model

- Reference Model
  - Basic elements of the model
    - Users, Roles, Permissions, Relationships
  - Four model components
    - Core RBAC
    - Hierarchical RBAC
    - Static Separation of Duty RBAC
    - Dynamic Separation of Duty RBAC

Core RBAC

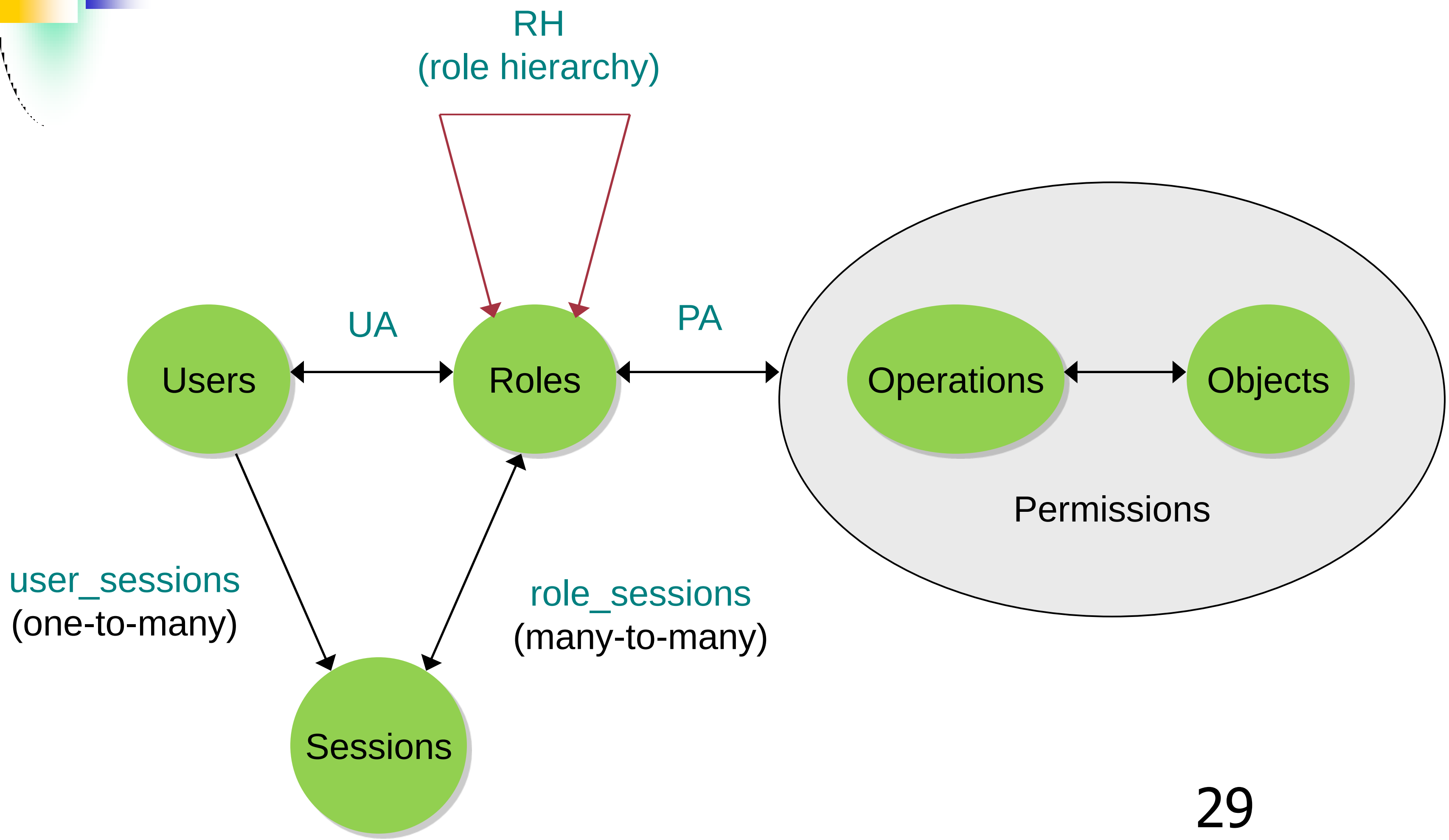


UA
PA
Users
Roles
Operations
Objects
Permissions
user_sessions
(one-to-many)
role_sessions
(many-to-many)
Sessions

Core RBAC (relations)

- Permissions = $2^{\text{Operations x Objects}}$
- UA $\subseteq$ Users x Roles
- PA $\subseteq$ Permissions x Roles
- *assigned_users*: Roles $\rightarrow 2^{\text{Users}}$
- *assigned_permissions*: Roles $\rightarrow 2^{\text{Permissions}}$
- *Op*(p): set of operations associated with permission p
- *Ob*(p): set of objects associated with permission p
- *user_sessions*: Users $\rightarrow 2^{\text{Sessions}}$
- *session_user*: Sessions $\rightarrow$ Users
- *session_roles*: Sessions $\rightarrow 2^{\text{Roles}}$
  session_roles(*s*) = $\{r \mid (\text{session\_user}(s), r) \in \text{UA}\}$
- *avail_session_perms*: Sessions $\rightarrow 2^{\text{Permissions}}$

Hierarchical RBAC



RBAC with General Role Hierarchy

- *authorized_users*: Roles $\rightarrow 2^{\text{Users}}$

  $authorized\_users(r) = \{u \mid r' \geq r \,\&\, (r', u) \in UA\}$

- *authorized_permissions*: Roles $\rightarrow 2^{\text{Permissions}}$

  $authorized\_permissions(\text{r}) = \{p \mid r \geq r' \,\&\, (p, r') \in PA\}$

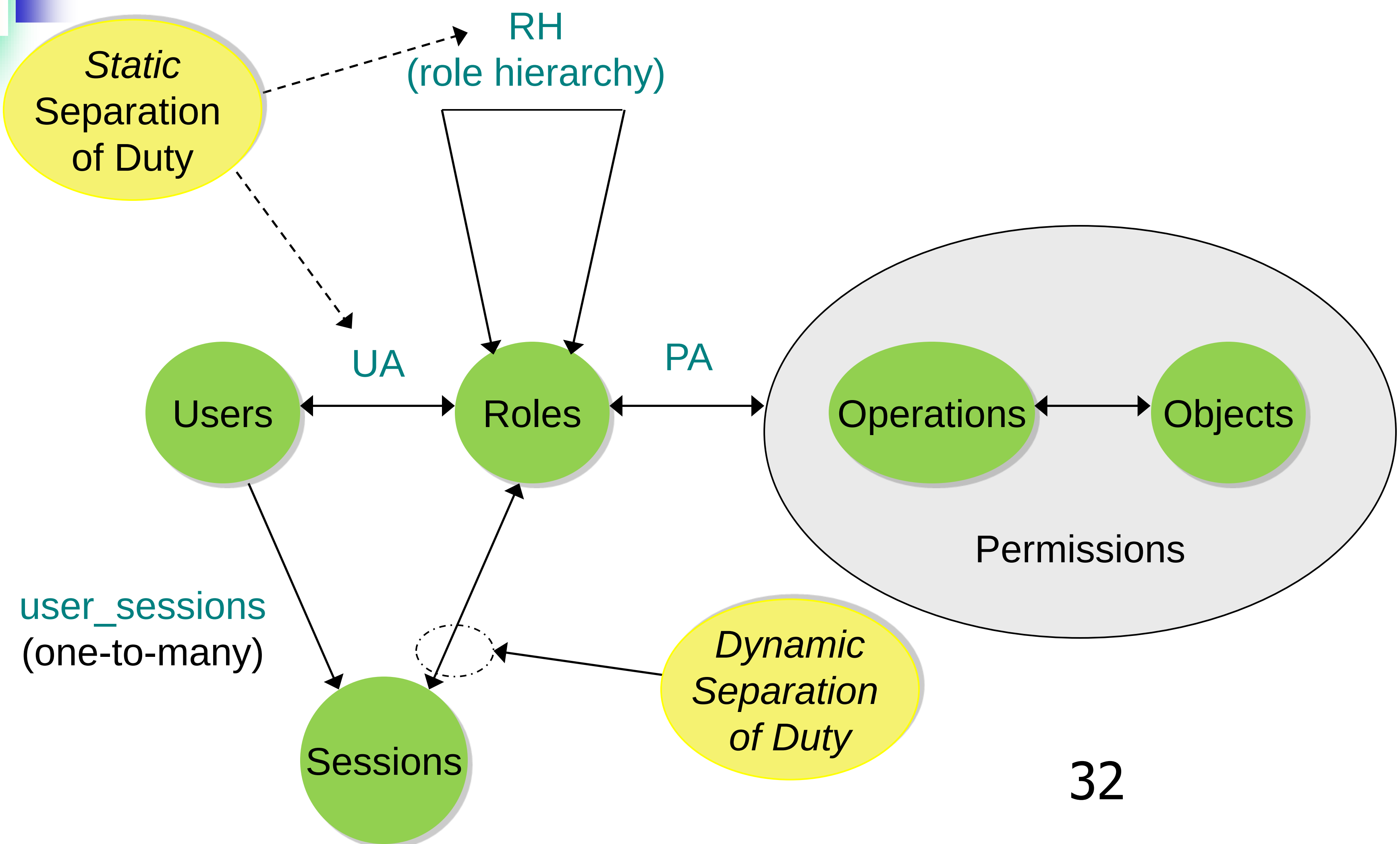
- RH $\subseteq$ Roles x Roles is a partial order
  - called the inheritance relation
  - written as $\geq$.

  $(r_1 \geq r_2) \rightarrow authorized\_users(r_1) \subseteq authorized\_users(r_2)$ &
  $authorized\_permisssions(r_2) \subseteq authorized\_permisssions(r_1)$

Separation of Duty

- SoD Security principle
  - Widely recognized
  - Captures conflict of interest policies to restrict authority of a single authority
    - Prevent Fraud
- Example,
  - A single person should not be allowed to "approve a check" & "cash it"

Constrained RBAC: SSD RBAC & DSD RBAC



Static Separation of Duty

- $SSD \subseteq 2^{Roles} \times N$
- In absence of hierarchy
  - Collection of pairs (*RS*, *n*) where *RS* is a role set, $n \geq 2$

    $\text{for all } (RS, n) \in SSD, \text{ for all } t \subseteq RS:$

    $|t| \geq n \rightarrow \cap_{r \in t} \, assigned\_users(\mathrm{r}) = \emptyset$

- In presence of hierarchy
  - Collection of pairs (RS, n) where RS is a role set, n ≥ 2;

    $\text{for all } (RS, n) \in SSD, \text{ for all } t \subseteq RS:$

    $|t| \geq n \rightarrow \cap_{r \in t} \, authorized\_uers(r) = \emptyset$

Dynamic Separation of Duty

- $DSD \subseteq 2^{\text{Roles}} \times N$
  - Collection of pairs (*RS*, *n*) where *RS* is a role set, $n \geq 2$;
    - A user cannot activate *n* or more roles from RS
  - What is the difference between SSD or DSD containing:

(*RS*, *n*)?

- Consider (*RS*, *n*) = ({r_1, r_2, r_3}, 2)?
- If SSD – can r_1, r_2 *and* r_3 be assigned to *u*?
- If DSD – can r_1, r_2 *and* r_3 be assigned to *u*?

ANSI RBAC standard – Functional specification

- Administrative operations
  - Creation and maintenance of sets and relations
- Administrative review functions
  - To perform administrative queries
- System level functionality
  - Creating and managing RBAC attributes on user sessions and making access decisions

Advantages of RBAC

- Allows Efficient Security Management
  - Administrative roles to manage other roles
  - Role hierarchy allows inheritance of permissions
- Principle of least privilege
- Separation of Duties constraints
- Grouping Objects
- Policy-neutrality
- Encompasses DAC and MAC policies
- Potential for use in multidomain environment
  - Open interconnected systems
    - Similarity of role concepts

RBAC Extensions

- Temporal RBAC model
- Geo RBAC model
- Spatio-temporal RBAC model
- Context aware RBAC models
- Geo Social RBAC model
- …
- …

Time-based Access Control Requirement

- Organizational functions and services with temporal requirements
  - A part-time staff is authorized to work only between 9am-2pm on weekdays
  - A day doctor must be able to perform his/her duties between 8am-8pm
  - An external auditor needs access to organizational financial data for a period of three months
  - In an insurance company, an agent needs access to patient history until a claim has been settled

Generalized Temporal RBAC (GTRBAC) Model

- Triggers and Events
- Temporal constraints
  - Roles, user-role and role-permission assignment constraints
  - Activation constraints (cardinality, active duration,..)
- Temporal role hierarchy
- Time-based Separation of duty constraints

Event and Trigger

- Simple events
  - `enable` *r* — `disable` *r*
  - $\texttt{assign}_U$ *r* `to` *u* — $\texttt{deassign}_U$ *r* `to` *u*
  - $\texttt{assign}_P$ *p* `to` *r* — $\texttt{deassign}_P$ *p* to *r*
  - `activate` *r* for *u* — `deactivate` *r* for *u*

- Prioritized event *pr:E*, where *pr* ∈ `Prios`
- Status expressions (e.g., Role, assignment status)
    - `enabled(`*r, t*`)`; `p_assigned(`*p , r, t*`)`
- Triggers: $E_1, \ldots, E_n, C_1, \ldots, C_k \rightarrow pr{:}E$ `after` Δt,
  - *where* E_i *are events,* C_i *are status expressions*
- User/administrator run-time request: *pr:E* `after` Δt

Temporal Constraints: Roles, User-role and Role-permission Assignments

- Periodic time
  - (*I*, *P*) : ⟨[`begin, end`], `P`⟩ is a set of intervals
  - `P` is an infinite set of recurring intervals
- Calendars:
  - *Hours, Days, Weeks, Months, Years*
- Examples

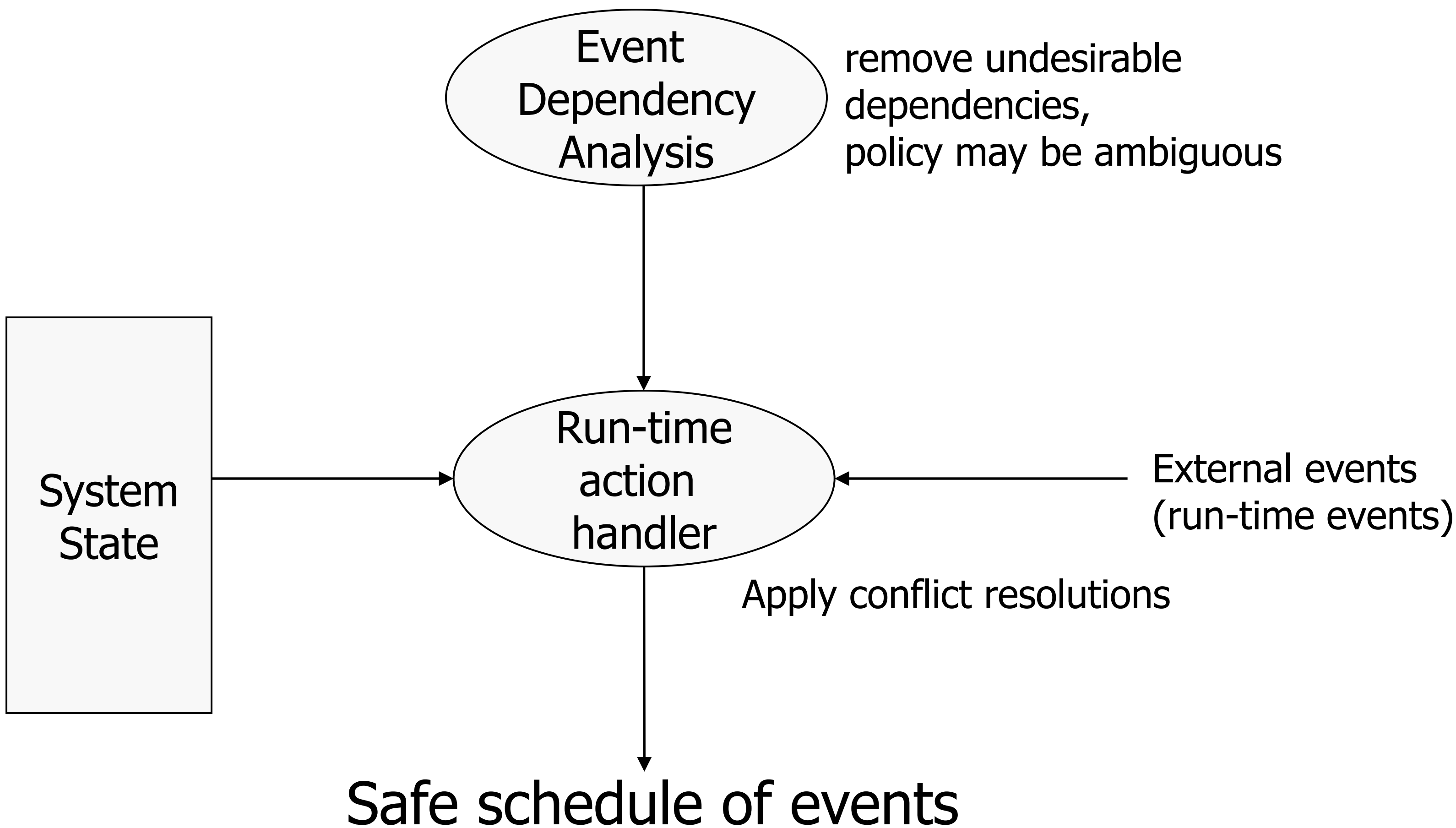
  all.Weeks + {2, …, 6}.*Days* + 10.*Hours* ▷ 12.*hours*

  - Daytime (9am to 9pm) of working days

Temporal Constraints: Roles, Assignments, Activation

- Periodicity: (I, P, *pr*:*E*)
  - ([1/1/2000, ∝], Daytime, `enable` DayDoctor)
- Duration constraint: (D, *pr:E*)
  - (Five hours, `enable` DoctorInTraining)
  - `activate` DayDoctor `for` Smith → `enable` DoctorInTraining after 1 hour
- Activation time constraints
  - E.g., Total duration for role activation
    1. Per role: D_{active}, [$D_{default}$], $\texttt{active}_{\texttt{R\_total}}$ *r*
    2. Per user role: $D_{uactive}$, *u*, $\texttt{active}_{\texttt{UR\_total}}$ *r*

GTRBAC Execution Model



Conflicts in GTRBAC

- GTRBAC specification can generate 3 types of conflicts
  - *Type* 1: between events of same type but opposite nature,
    - e.g., `enable` *r* ***vs.*** `disable` *r*
  - *Type* 2: between events of dissimilar types
    - e.g., `activate` *r* `for` *u* ***vs.*** `de-assign` *r* `to` *u* OR `disable` *r*
  - *Type* 3: between constraints

    (a) (X, pr:E) ***vs.*** (X, q:E)

    (b) Per-role ***vs.*** per-user-role constraints

Handling Conflicts

- *Type* 1 *and Type* 3(a)
  - Higher priority takes precedence
  - Disabling event takes precedence if priorities are the same
    - e.g., `disable` *r* takes precedence over `enable` *r*
- *Type* 2
  - activation event has lower precedence
- *Type* 3(b)
  - per-user-role constraints take precedence

Ambiguous Event Dependency

- A set of triggers may give rise to ambiguous semantics
- Example:
  - *tr*1: enable R1 → disable R2
  - *tr*2: enable R2 → disable R1
  - Let the runtime requests be: {enable R1; enable R2},

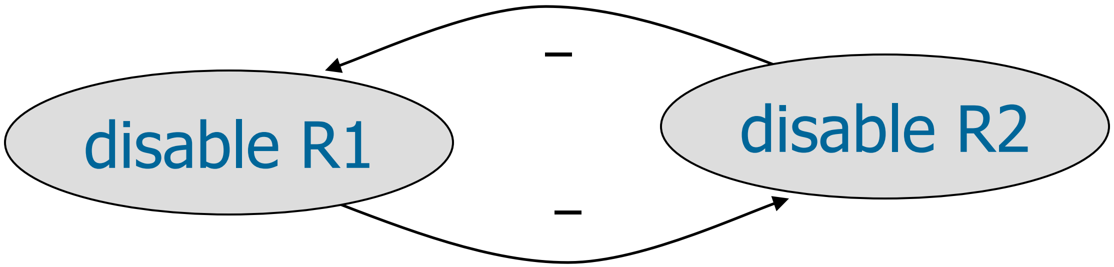
1. *tr*1 fires: {enable R1; disable R2}
   (Intuitively, *tr1* blocks *tr*2)
2. *tr*2 fires: {enable R2; disable R1}
   (Intuitively, *tr2* blocks *tr1*)

two symmetric possibilities

- Solution: Detect ambiguity using Labeled dependency graph

Dependency Graph Analysis

- Labeled Dependency Graph
  - Directed graph (*N*, *E*)
  - *N*: set of prioritized events in the head of some trigger
  - *E*: set of triples of the form (X, l, Y)
    - For all triggers [$B \rightarrow p{:}E$]
    - For all events *E*' in the body *B*, and for all nodes *q*:*E*' in N
    - <*q*:*E*', + , *p*:*E*>
    - <*r*:conf(*E*'), -, *p*:*E*> for all [*r*:conf(*E*')] in *N* such that *q* <= *r*
- Dependency Graph for the Example:



Safe Set of Triggers

- A set of triggers *T* is *safe* if its labeled dependency graph has no cycles with label "-".
- **Theorem:** If a *T* is *safe*, then there exists exactly one execution model.
- **Complexity of** *DAG-based safeness algorithm* : $O(|T|^2)$.

Role Hierarchy in GTRBAC

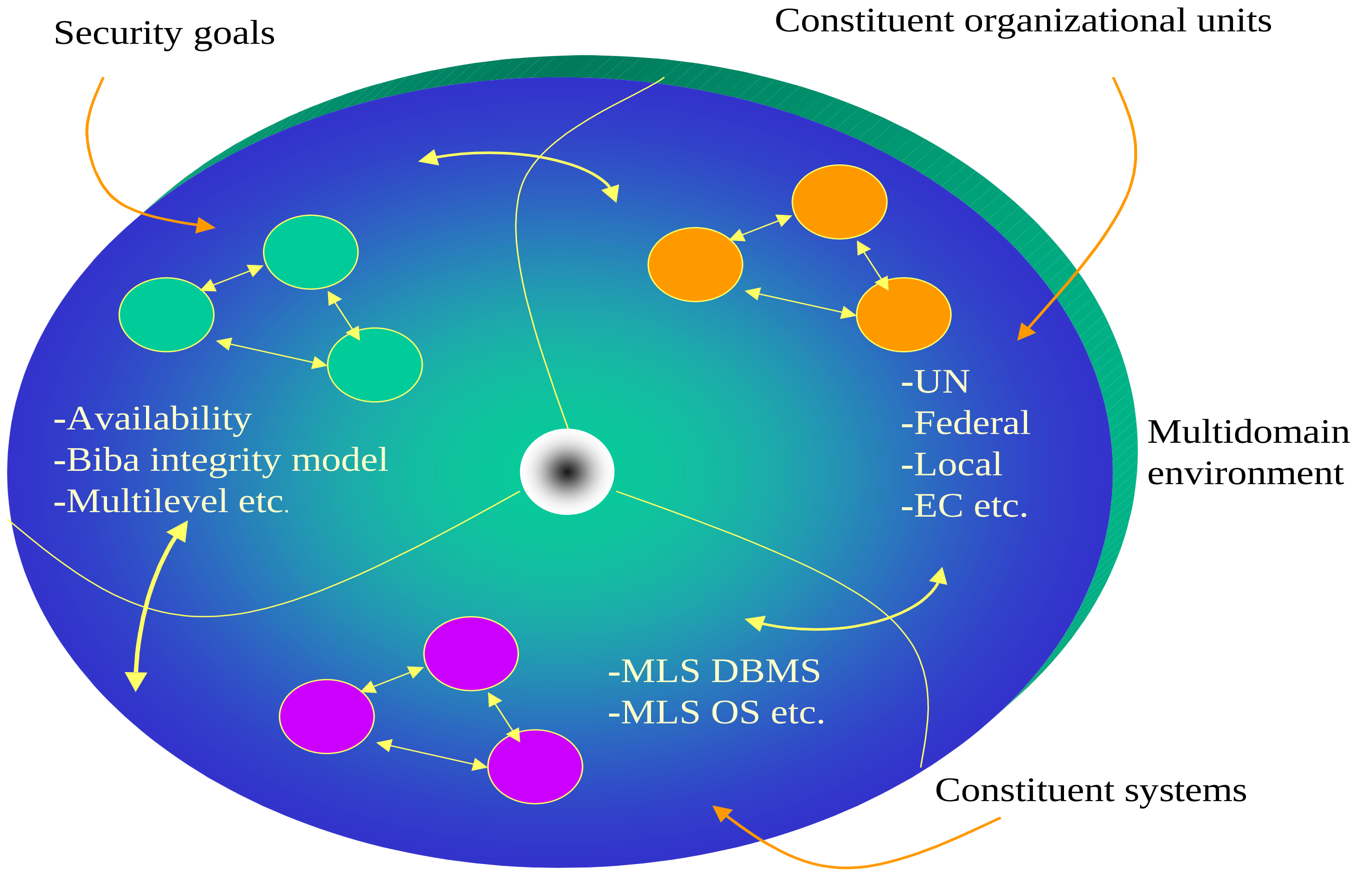
- Useful for efficient security management of an organization
  - No previous work has addressed the effect of temporal constraints on role hierarchies
- GTRBAC temporal role hierarchies allow
  - Separation of permission inheritance and role activation semantics that facilitate management of access control
  - Capturing the effects of the presence of temporal constraints on hierarchically related roles

Types of role Hierarchy – to accommodate temporal constraints

- Permission-inheritance hierarchy (I-hierarchy)
  - Senior inherits juniors' permissions
  - User assigned to senior cannot activate juniors
- Role-Activation hierarchy (A-hierarchy)
  - Senior does not inherit juniors' permissions
  - User assigned to senior can activate junior
  - Advantage: SOD constraint can be defined on hierarchically related roles
- Activation Inheritance hierarchy (IA-hierarchy)
  - Senior inherits juniors' permissions
  - User assigned to senior can activate junior

Multidomain Environments

- Dimensions of heterogeneity



Key Access Control Challenges in a Multi-Domain Environment

- Semantic heterogeneity
- Secure interoperation
- Assurance and risk propagation
- Security Management

Semantic heterogeneity

- Different systems may use different security policies
  - e.g., DAC, MAC, Chinese wall, Integrity policies etc.
- Variations of the same policies
  - e.g., BLP model and its several variations
- Naming conflict on security attributes
  - Similar roles with different names
  - Similar permission sets with different role names
- Structural conflict
  - different multilevel lattices / role hierarchies

Secure Interoperability

- Principles of secure interoperation

  Principle of autonomy

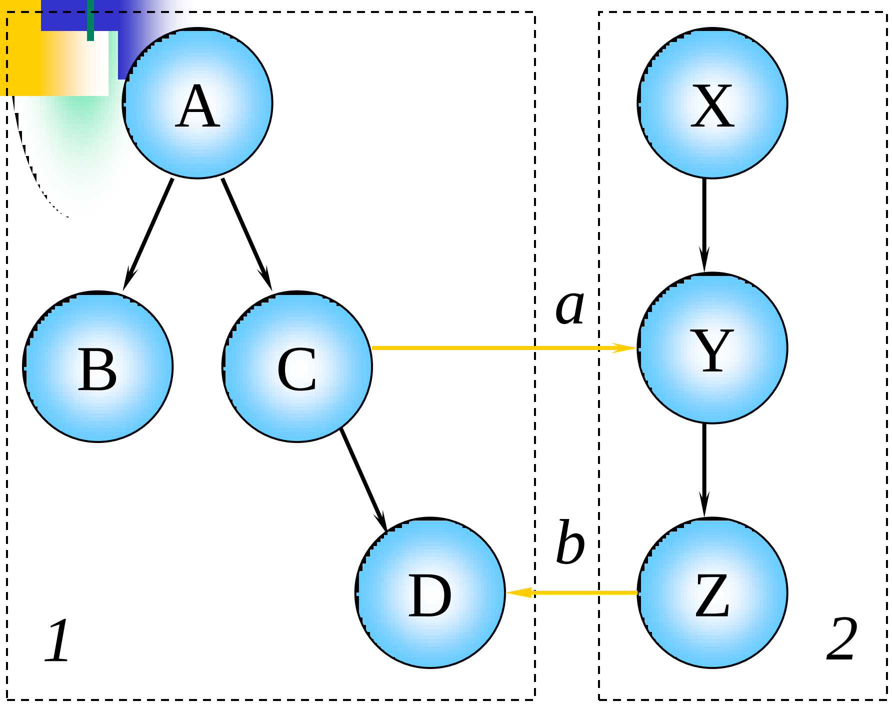
  - If an access is permitted within an individual system, it must also be permitted under secure interoperation in a multi-domain environment.

  Principle of security

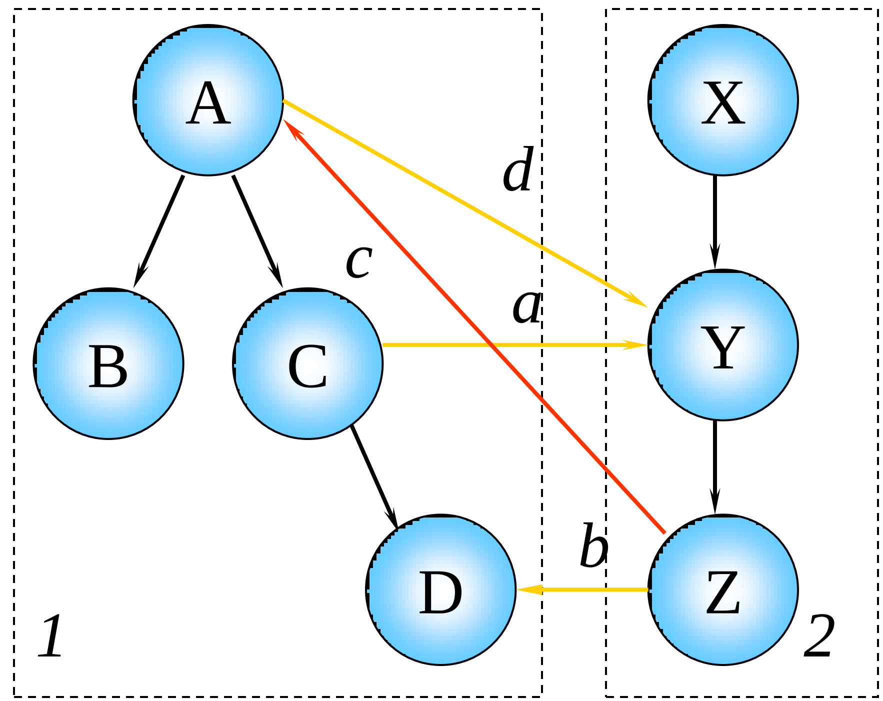
  - If an access is not permitted within an individual system, it must not be permitted under secure interoperation.

- Interoperation of secure systems can create new security breaches

Unsecure Interoperability



F_{12} = {a, b}



F_{12} = {a, b, c, d}

F_{12} - permitted access between systems 1 and 2

(1) F_{12} = {a, b, d}
Direct access

(2) F_{12} = {c}

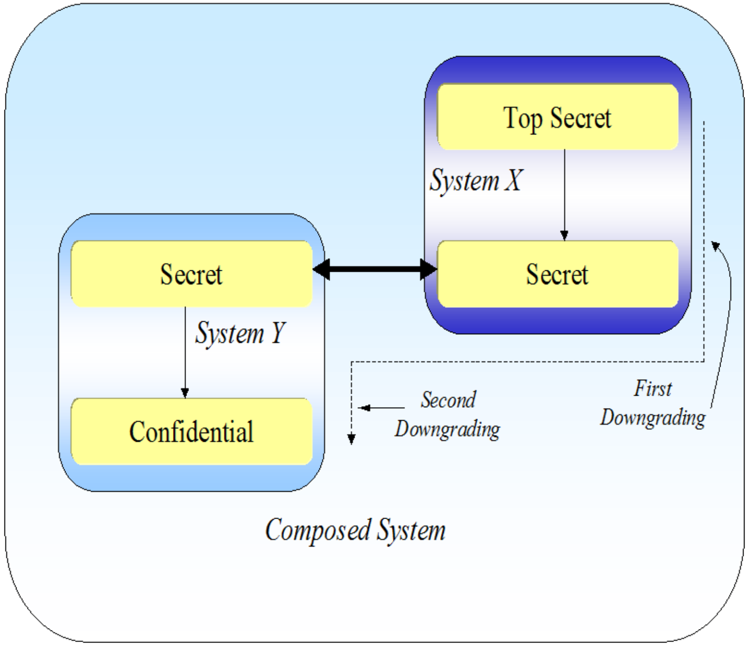
Challenges in Secure Interoperability

How to ensure autonomy and security principles?

- Determining inconsistencies/incompleteness in security rules.
- Identifying security holes
- Selecting optimality criteria for secure interoperability: maximizing number of domains, direct accesses

Assurance and Risk Propagation & Security Management

- Assurance and Risk propagation
  - Breach in one domain can render the whole environment insecure
  - Cascading problem
- Security Management
  - Centralized/Decentralized
  - Managing global metapolicy
  - Managing policy evolution

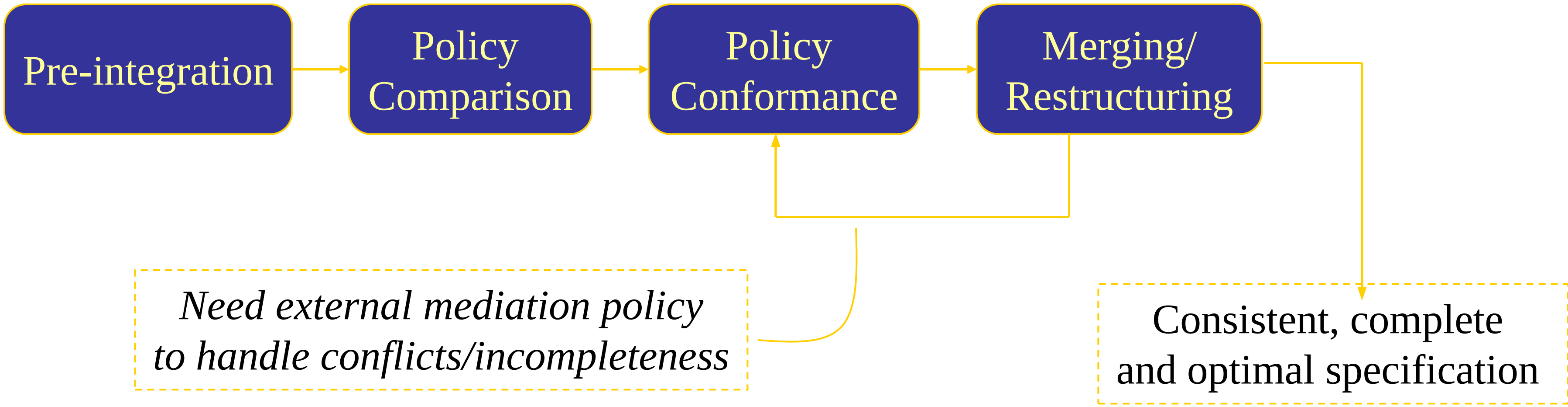


Approaches to Multidomain Problem

- Policy-Metapolicy specification framework
  - Ad-hoc, Formal models: lattice merging, RBAC
- Agent based approach (Policy agents)
- Architectural approaches (CORBA, DCE)

A Multi-Domain Access Control Framework

- A Multi-Phase Framework
- Based on RBAC model



Pre-integration Phase

- Requires RBAC representation of arbitrary policies. A policy mapping technique is needed for non-RBAC systems.
- Uses an information base
  - Semantic information about domains including policies, roles and attributes
  - Integration/merging strategies to generate the overall configuration of the multi-domain environment.

Policy Comparison and Conformance

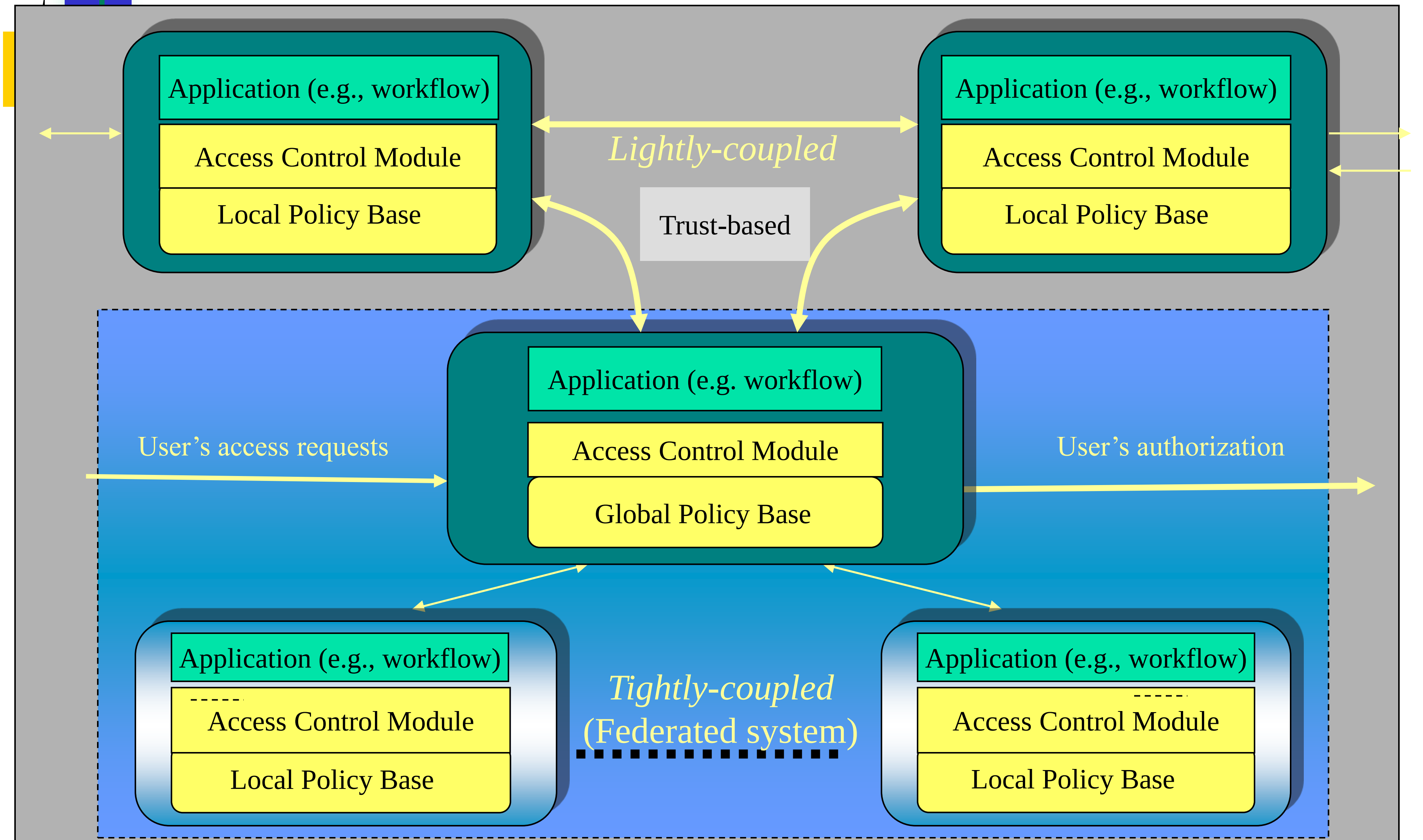
- Tools & techniques for detecting
  - Semantic conflicts
    - Naming conflicts
    - Structural conflicts
  - Rule conflicts
- Mediation policies are needed for resolution
  - Predefined meta-policies
  - Domain cooperation by administrators
- Tradeoffs
  - Determine optimal/heuristic solutions secure interoperability

Merging/Restructuring

- Merging/integrating policies
  - Restructure domain policies according to the selected optimal criteria
  - Generate integrated global policy

- Repeat policy conformance step
  - Re-evaluation and restructuring of meta-policy

Multidomain Security



Summary

- Overview of Access control models
- Multidomain challenges ..