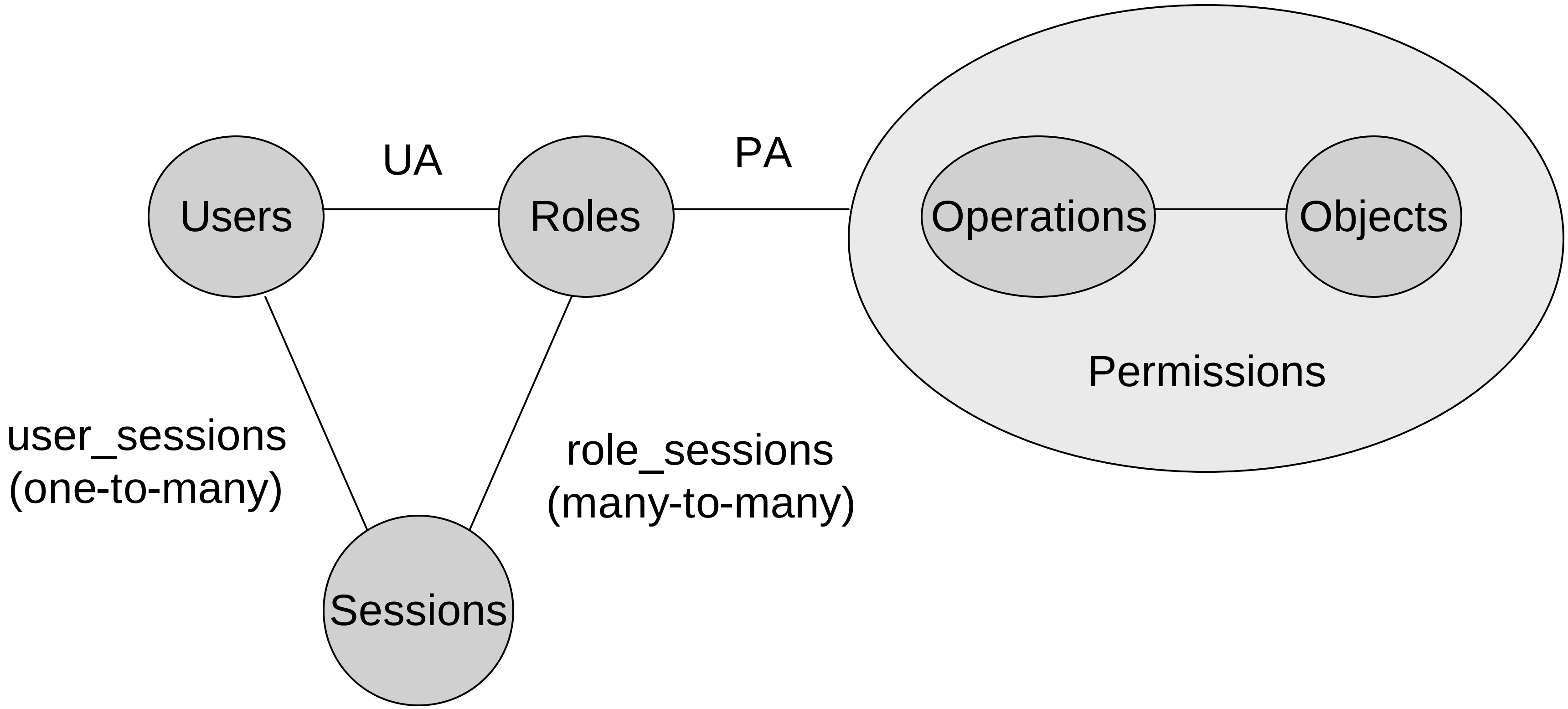
Introduction to Computer Security

Lecture 6
RBAC,
Policy Composition
Design Principles

October 14, 2003

RBAC (NIST Standard)

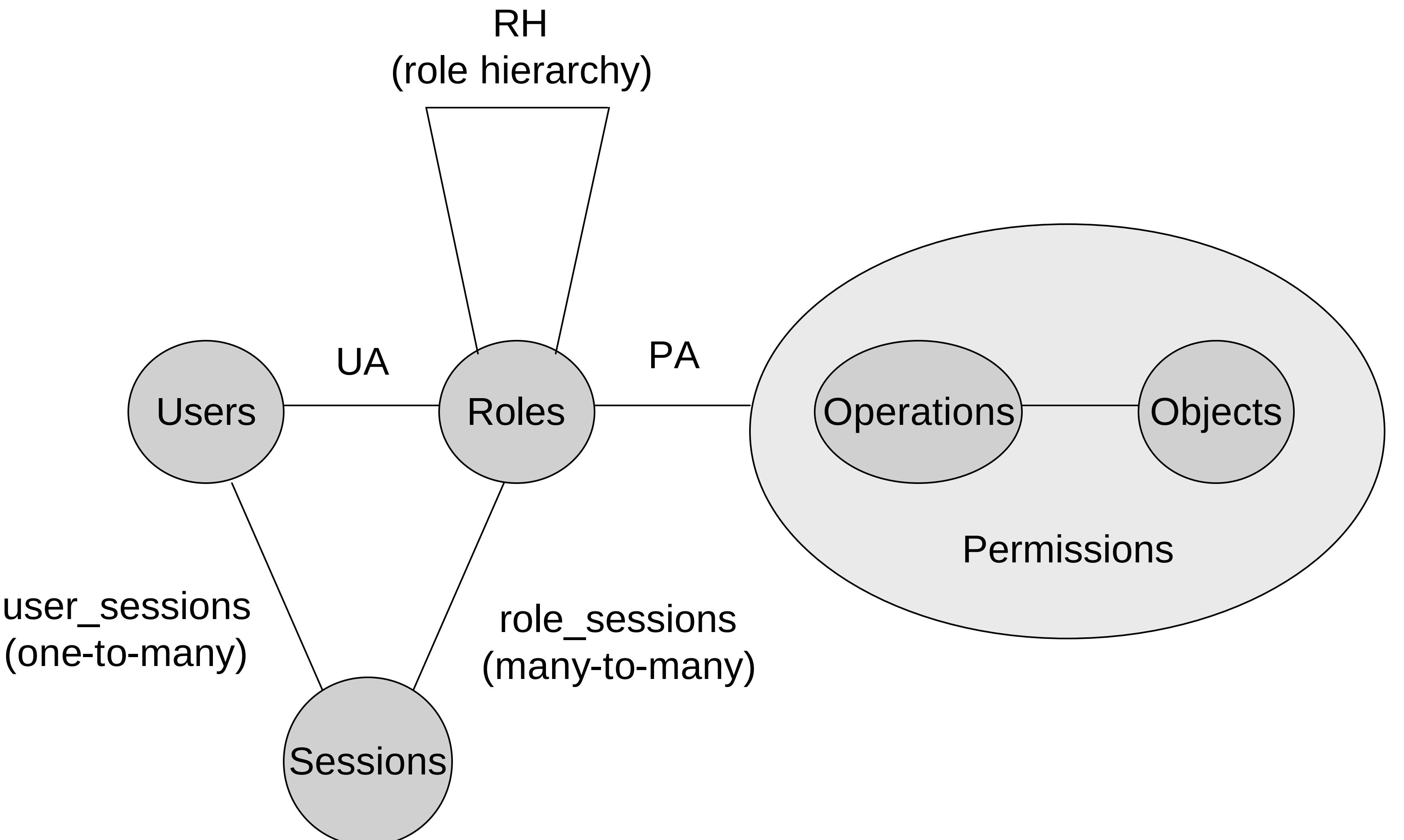


An important difference from classical models is that Subject in other models corresponds to a Session in RBAC

Core RBAC (relations)

- Permissions = $2^{\text{Operations x Objects}}$
- UA ?   Users x Roles
- PA ?   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 $\rightarrow 2^{\text{Permissions}}$

RBAC with General Role Hierarchy



RBAC with General Role Hierarchy

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

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

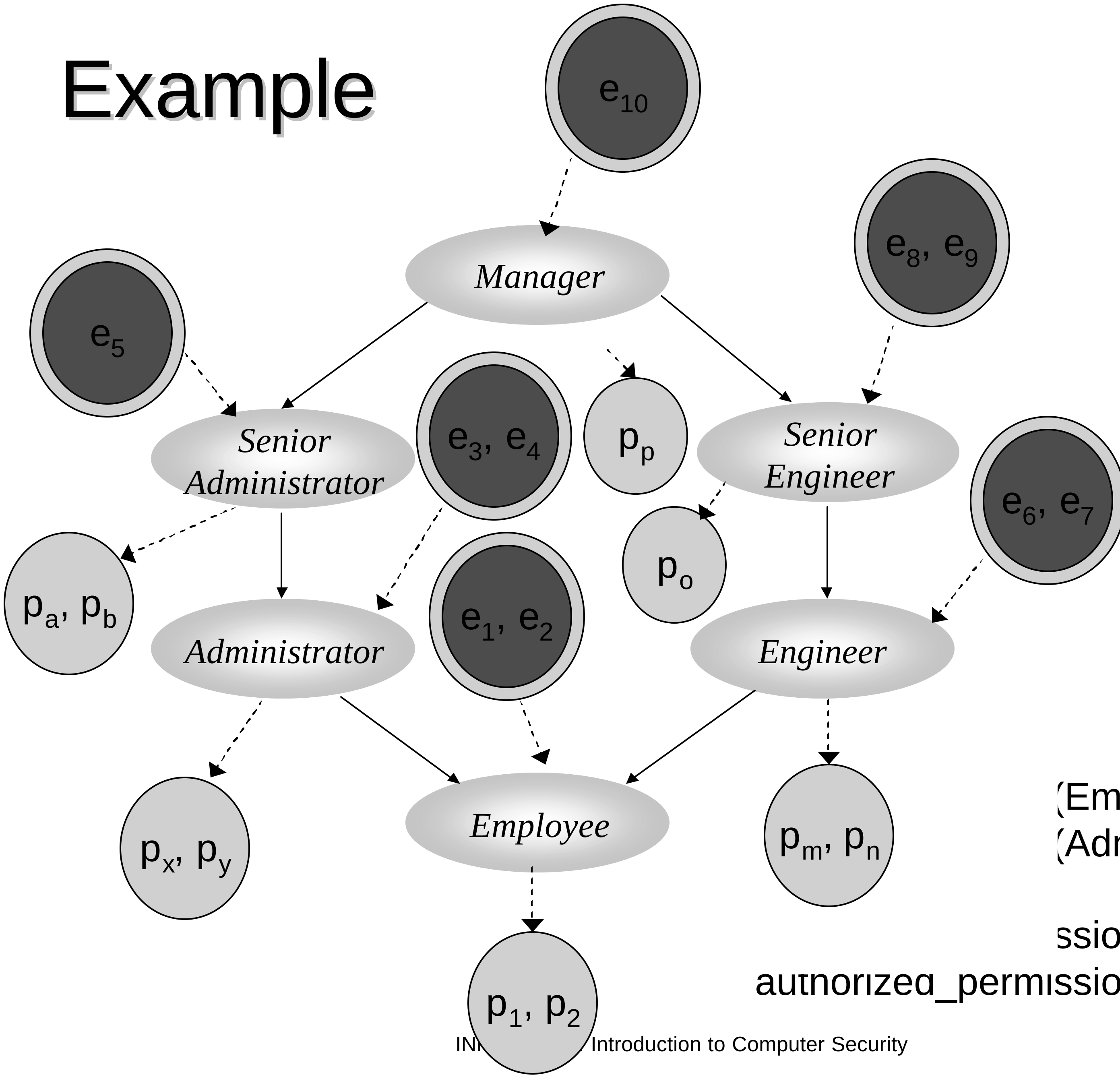
- *authorized_permissions*: Roles$\rightarrow$ $2^{Permissions}$

  $authorized\_users(\mathrm{r}) = \{p \mid r' = r \ \&(p, r') \in PA)$

- RH ? Roles x Roles is a partial order
  - called the inheritance relation
  - written as =.

  $(r_1 = r_2) \rightarrow authorized\_users(r_1)$? $authorized\_users(r_2)$ & $authorized\_permisssions(r_2)$? $authorized\_permisssions(r_1)$

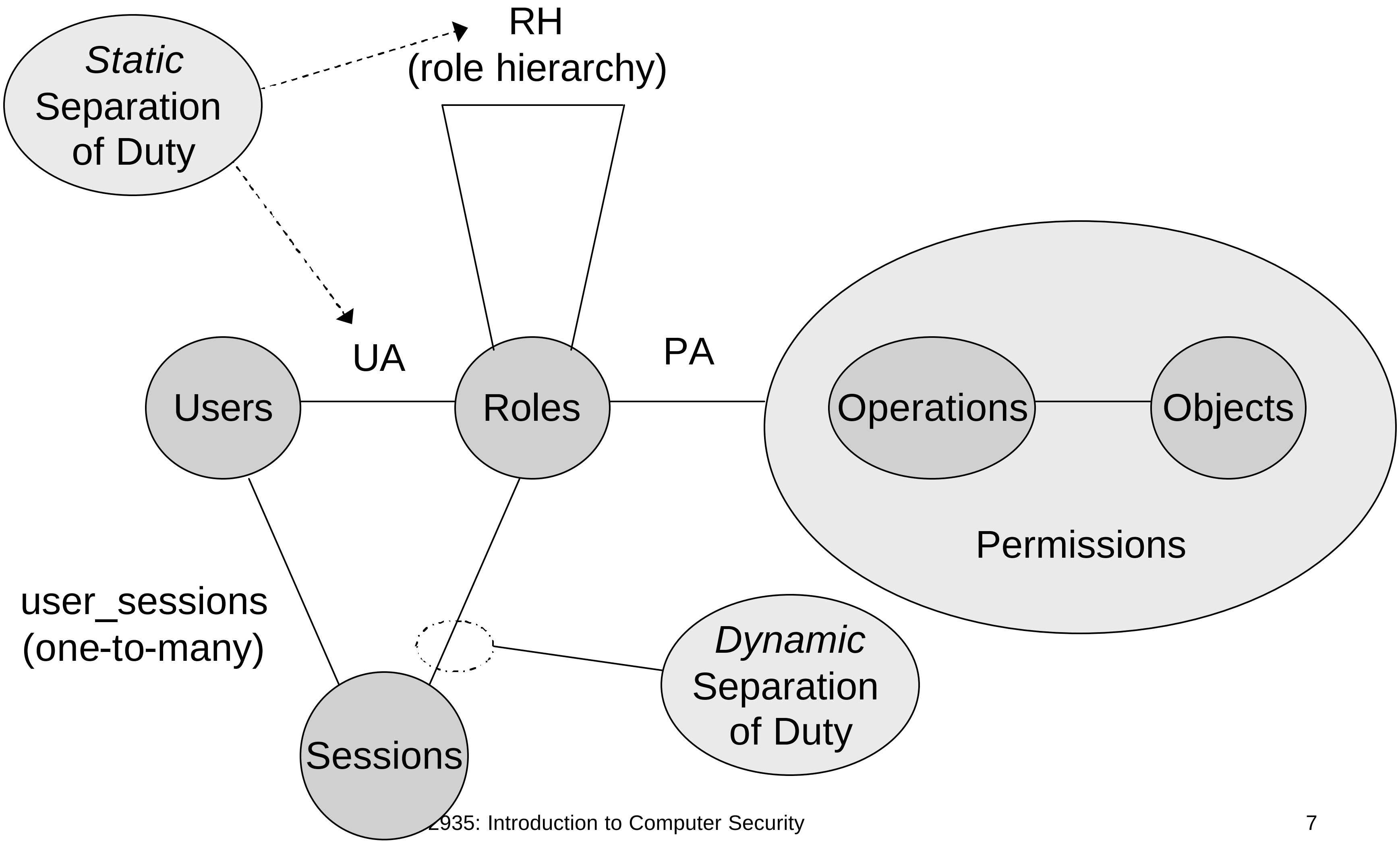
Example



(Employee)?
(Administrator)?

ssions(Employee)?
authorized_permissions(Administrator)?

Constrained RBAC



Static Separation of Duty

- *SSD* ? 2^{Roles} x N
- In absence of hierarchy
  - Collection of pairs (*RS*, *n*) where *RS* is a role set, *n* = 2;
    for all (*RS*, *n*) $\in$ *SSD*, *for all t* ? *RS*:

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

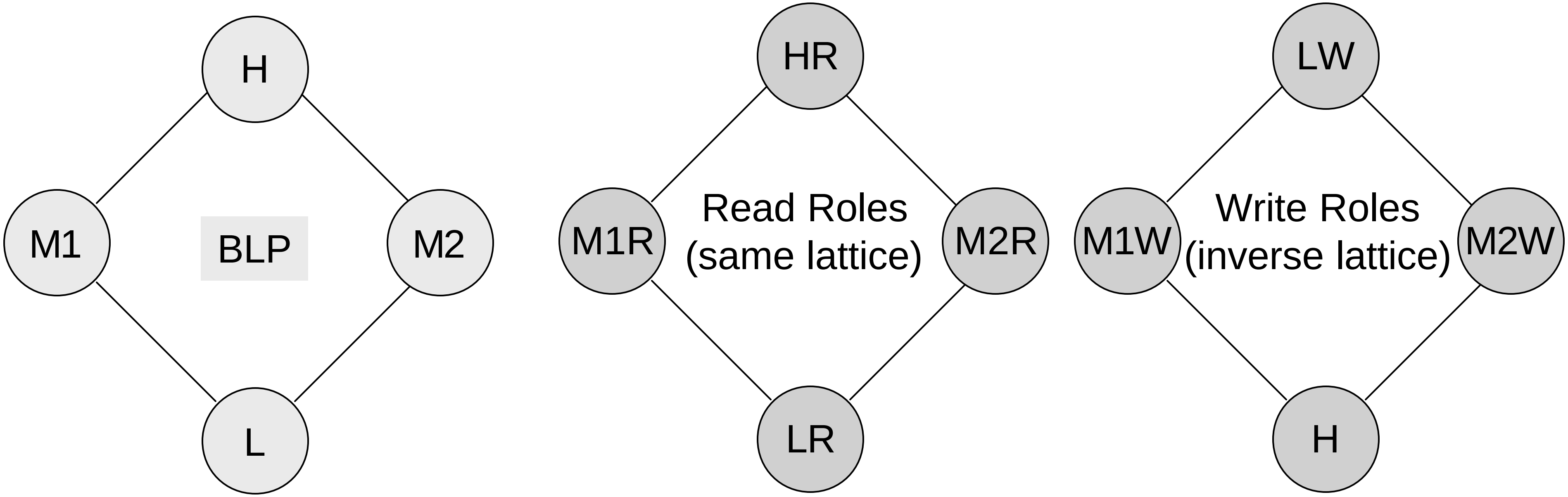
- In presence of hierarchy
  - Collection of pairs (RS, n) where RS is a role set, n = 2;
    for all (*RS*, *n*) $\in$ *SSD*, *for all t* ? *RS*:

$$|t| = n \rightarrow \mathrm{n}_{r \in t}\ authorized\_uers(r) = \emptyset$$

Dynamic Separation of Duty

- *DSD* ? 2^{Roles} x N
  - Collection of pairs (*RS*, *n*) where *RS* is a role set, *n* = 2;
  - A user cannot activate *n* or more roles from RS
  - What if both SSD and DSD contains (*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*?

MAC using RBAC



Transformation rules

- $R = \{L_1R, L_2R, \ldots, L_nR, L_1W, L_2W, \ldots, L_nW\}$
- Two separate hierarchies for $\{L_1R, L_2R, \ldots, L_nR\}$ and $\{L_1W, L_2W, \ldots, L_nW\}$
- Each user is assigned to exactly two roles: xR and LW
- Each session has exactly two roles yR and yW
- Permission (o, r) is assigned to xR iff (o, w) is assigned to xW)

RBAC's Benefits

TABLE 1: ESTIMATED TIME (IN MINUTES) REQUIRED FOR ACCESS ADMINISTRATIVE TASKS			
TASK	RBAC	NON-RBAC	DIFFERENCE
Assign existing privileges to new users	6.14	11.39	5.25
Change existing users' privileges	9.29	10.24	0.95
Establish new privileges for existing users	8.86	9.26	0.40
Termination of privileges	0.81	1.32	0.51

Cost Benefits

- Saves about 7.01 minutes per employee, per year in administrative functions
  - Average IT amin salary - $59.27 per hour
  - The annual cost saving is:
    - $6,924/1000; $692,471/100,000
- Reduced Employee downtime
  - if new transitioning employees receive their system privileges faster, their productivity is increased
  - 26.4 hours for non-RBAC; 14.7 hours for RBAC
  - For average employee wage of $39.29/hour, the annual productivity cost savings yielded by an RBAC system:
    - $75000/1000; $7.4M/100,000

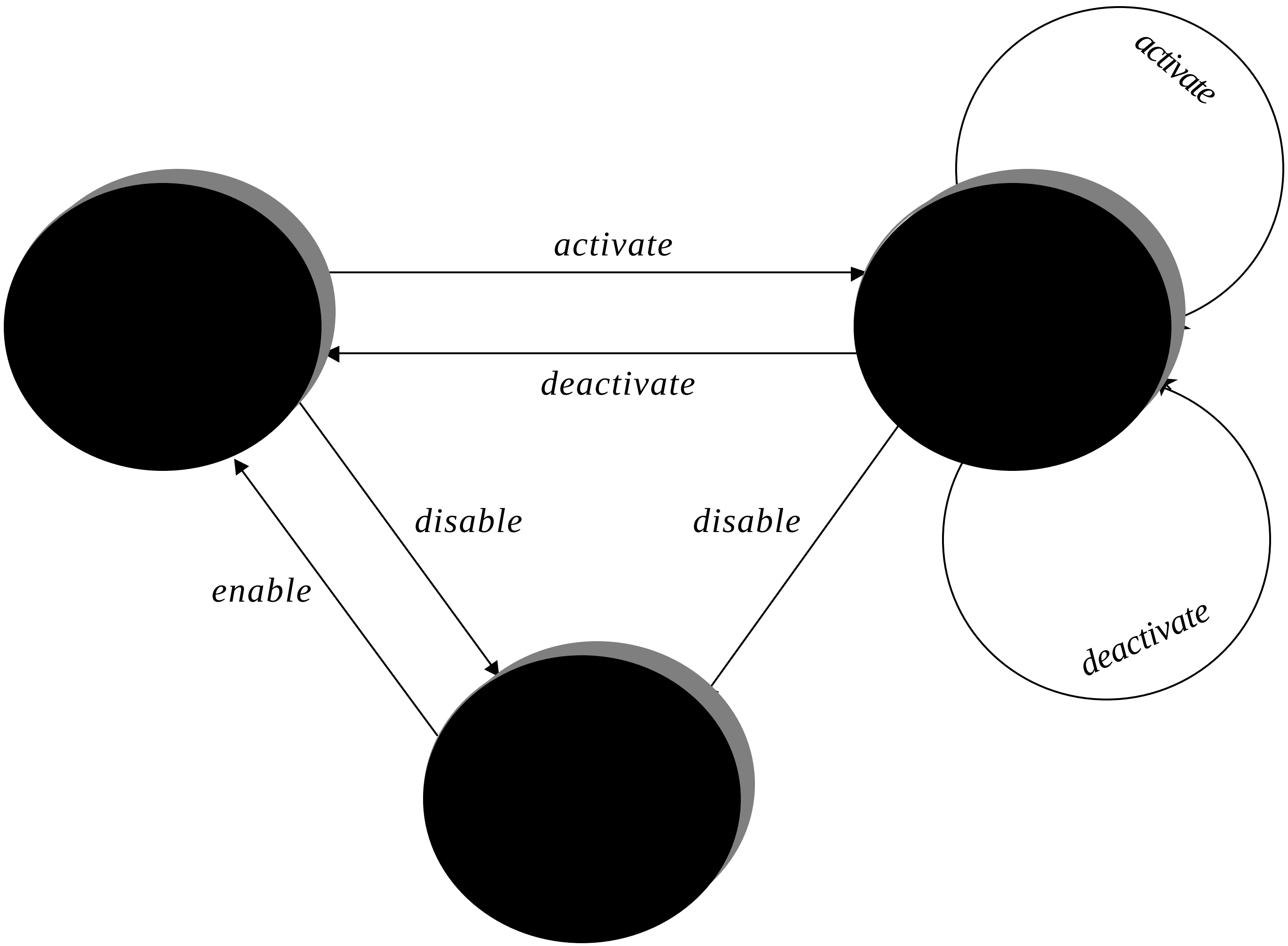
Time-based Access Control Requirement

- 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
  - A video library allows access to a subscriber to view at most three movies every week
  - In an insurance company, an agent needs access to patient history until a claim has been settled

Generalized Temporal RBAC

- 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

States of a Role in GTRBAC



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
  - Role, assignment status – e.g.. `enabled(`*r*`)`; `p_assigned(`*p* `,`*r*`)`
- Triggers: $E_1, \ldots, E_n, C_1, \ldots, C_k$ ® *pr:E* **`after`** *? t* , *where E_i are events, C_i are status expressions*

  Example:

  `enable` DayDoctor → `enable` DoctorInTraining after 1 hour
- User/administrator run-time request: *pr:E* **`after`** *? t*

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

- Periodic time
  - (I, P) : $\langle$`[begin, end]`, `P`$\rangle$ 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

  all.Weeks + {2, …, 6}.*Days*

  - Working days

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

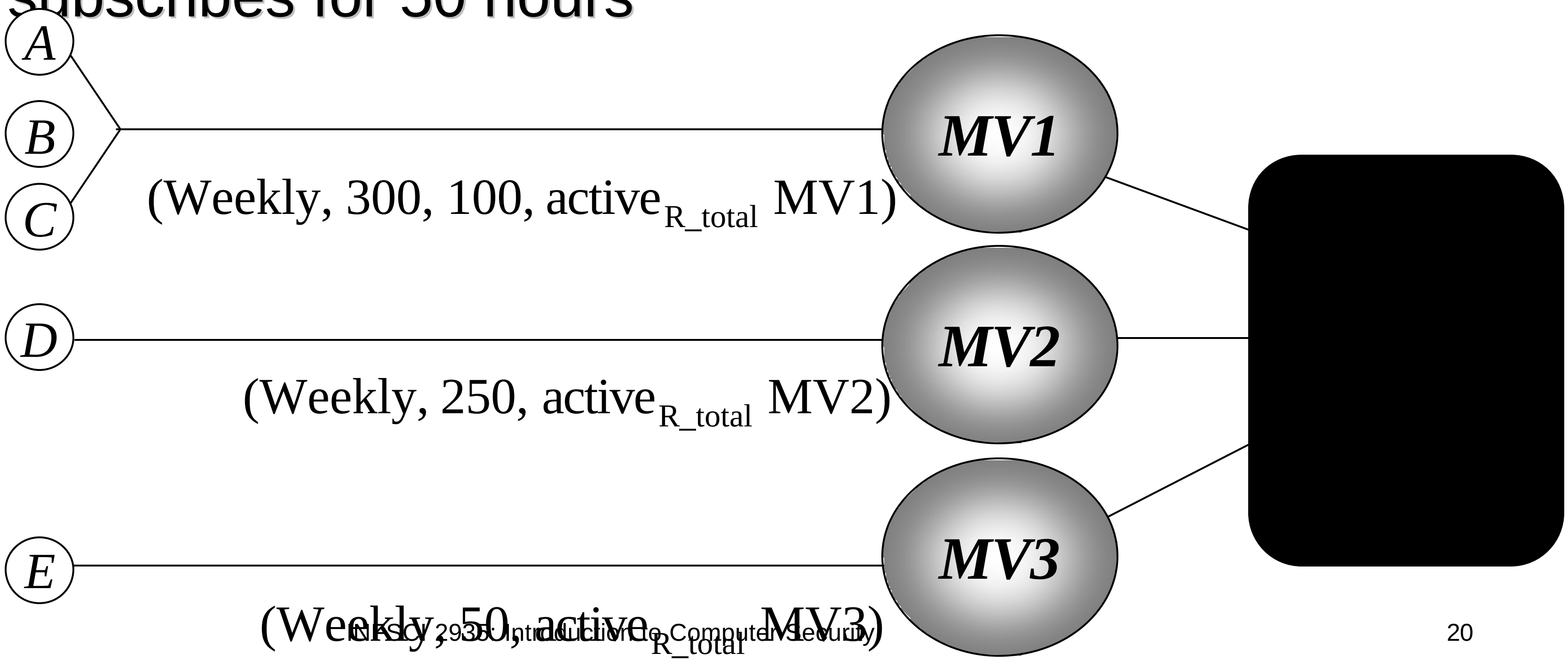
- Periodicity: (I, P, *pr*:*E*)
  - ([1/1/2001, ∝], Daytime, `enable` DayDoctor)
  - ([1/1/2000, ∝], {Mon,Wed}, $\texttt{assign}_U$ DayDoctor `to` Smith)
- Duration constraint: (D, *pr:E*)
  - (Five hours, `enable` DoctorInTraining)
  - `activate` DayDoctor `for` Smith → `enable` DoctorInTraining after 1 hour
- Cardinality constraint: ([I, P], N, $\texttt{assign}_U$ *r*)
  - ([1/1/2000, ∝], {Mon, Wed}, 5, $\texttt{assign}_U$ DayDoctor)

Activation Time Constraints

- Active role duration
  - 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, $\texttt{active}_{\texttt{UR\_total}}\ r$
  - Max active role duration per activation C
    1. Per role: D_{max}, $\texttt{active}_{\texttt{R\_max}}\ r$
    2. Per user role: D_{umax}, u, $\texttt{active}_{\texttt{UR\_max}}\ r$
- Cardinality
  - Total number of role activations
    1. Per role: N_{active}, $[N_{default}]$, $\texttt{active}_{\texttt{R\_n}}\ r$
    2. Per user role: $N_{uactive}$, u, $\texttt{active}_{\texttt{UR\_n}}\ r$
  - Max number of concurrent activations C
    1. Per role: N_{max}, $[N_{default}]$, $\texttt{active}_{\texttt{R\_con}}\ r$
    2. Per user role: N_{umax}, u , $\texttt{active}_{\texttt{UR\_con}}\ r$

Example of Activation Time Constraint

- Video library offers 600 hours of total time per week
- *A, B* and *C* subscribe for 100 hours each
- *D* subscribes for 250 hours
- *E* subscribes for 50 hours



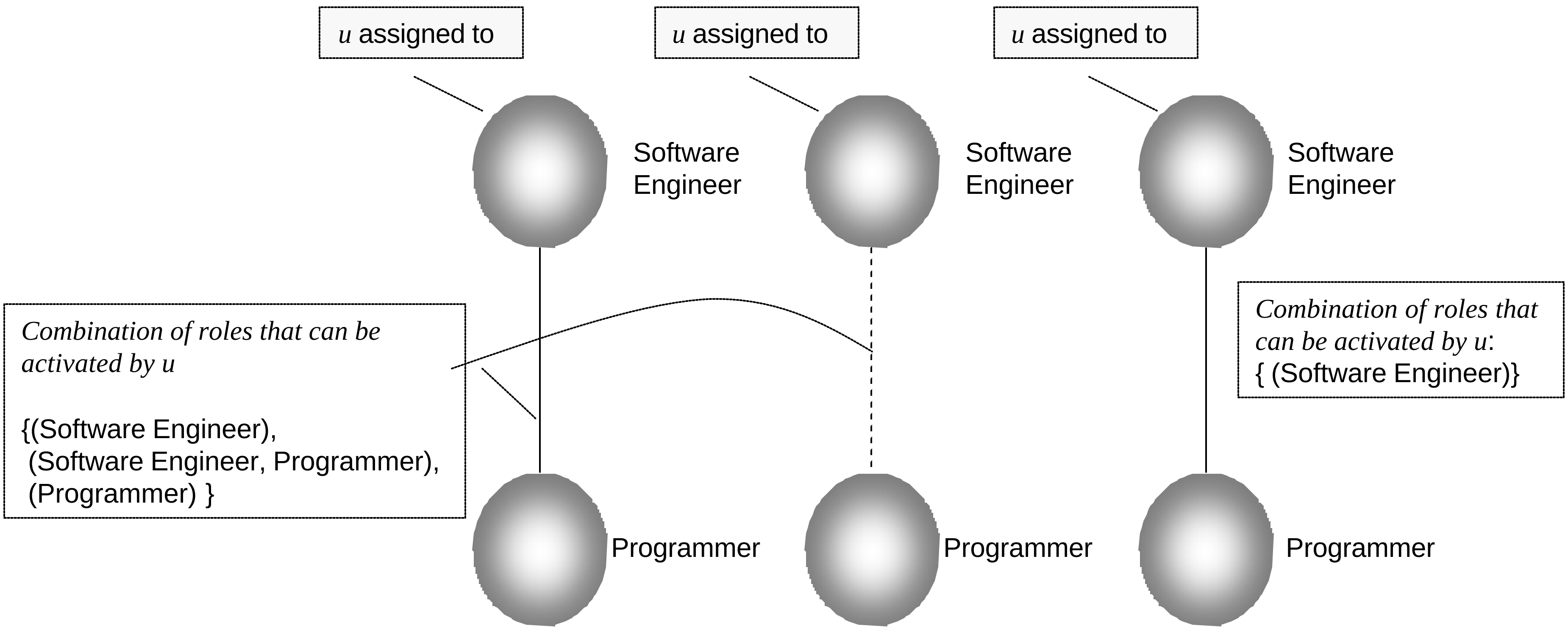
Role Hierarchy in GTRBAC

- GTRABC-based temporal role hierarchies allow
  - Separation of permission inheritance and role activation semantics that facilitate management of access control
  - Capturing the effect of the presence of temporal constraints on hierarchically related roles and therefore allowing fine-grained access control

Types of Role Hierarchy

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

Types of Role Hierarchy

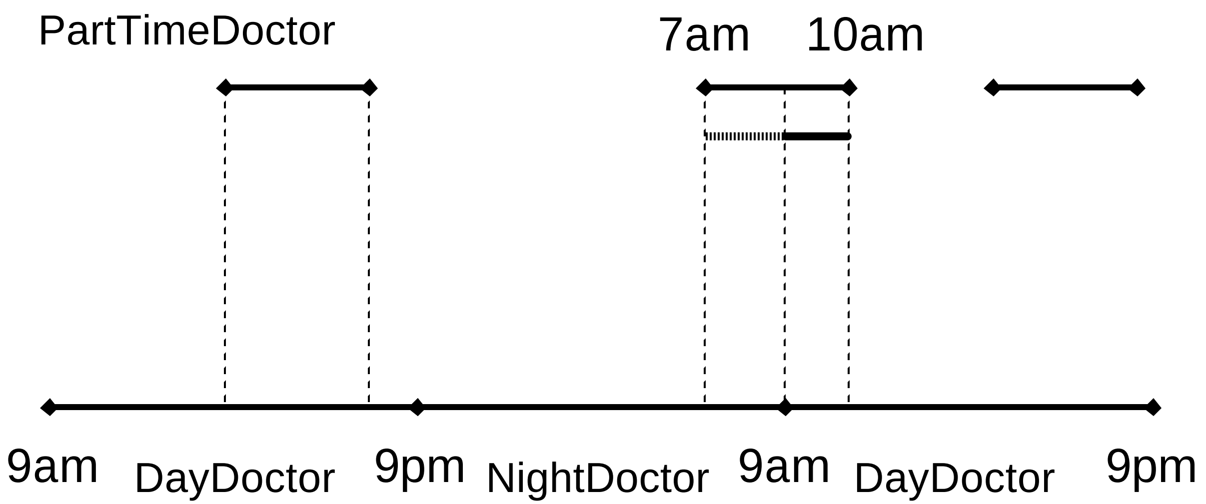


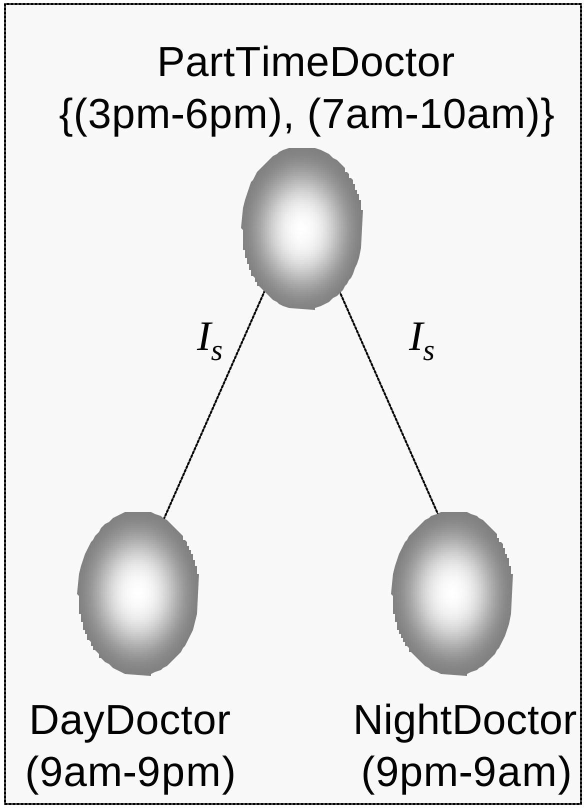
(a) *IA* Hierarchy (b) *A* Hierarchy (c) *I* Hierarchy

Weakly Restricted and Strongly Restricted Temporal Role Hierarchies

- *I*-hierarchy: (assume x is senior of y)
  - Weakly restricted hierarchy
    - x inherits y's permissions
    - y need not be enabled
  - Strongly restricted hierarchy
    - x inherits y's permissions only when both x and y enabled
- *A*-hierarchy: (assume x is senior of y and u is assigned to x)
  - Weakly restricted hierarchy
    - u can activate y
    - x need not be enabled
  - Strongly restricted hierarchy
    - u can activate y only when both x and y are enabled
- *IA*-hierarchy: x and y are related by both *I*-hierarchy and *A*-hierarchy

Temporal Role Hierarchy Example





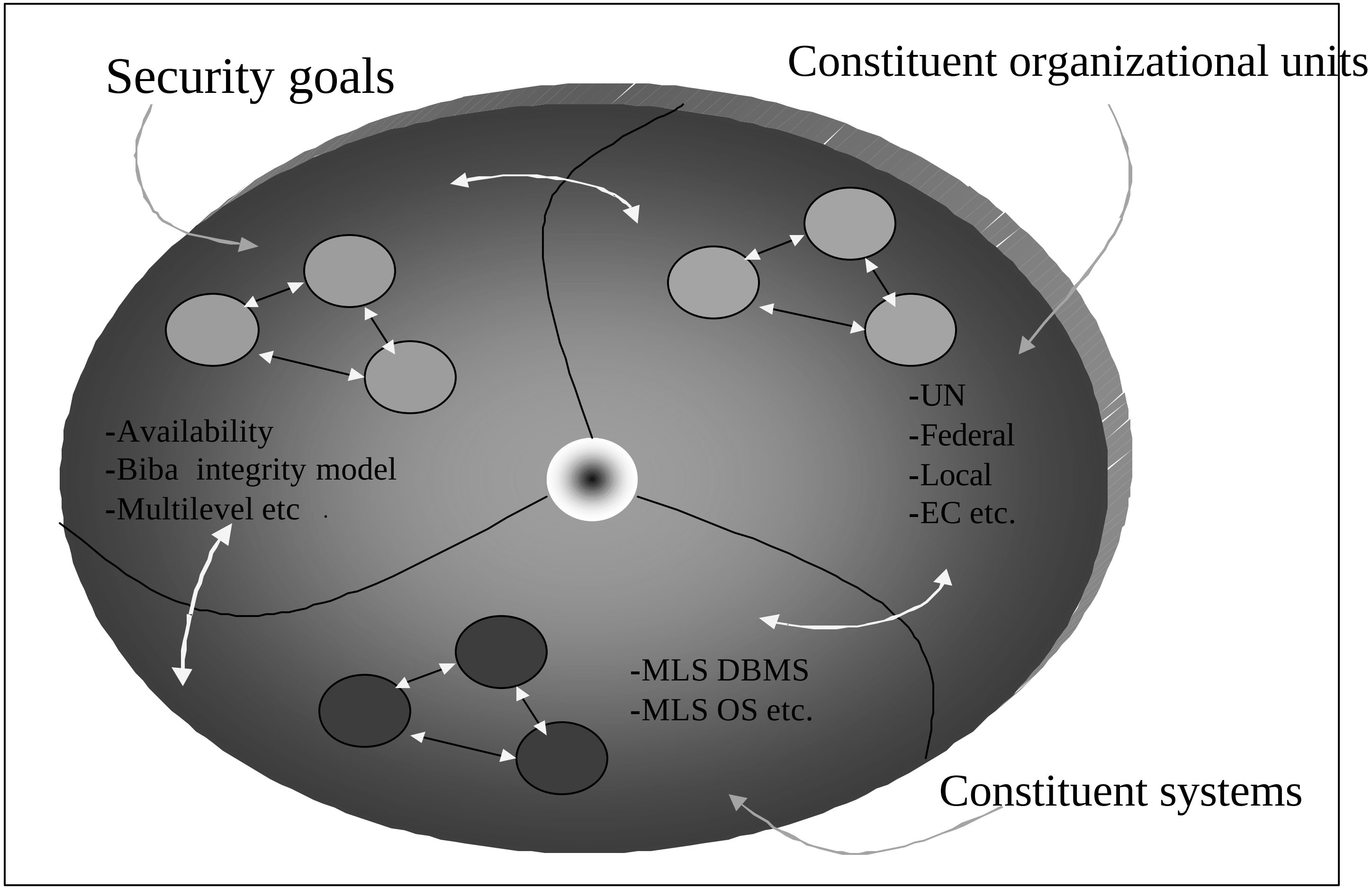
Policy Composition

Problem: *Consistent* Policies

- Policies defined by different organizations
  - Different needs
  - But sometimes subjects/objects overlap
- Can all policies be met?
  - Different categories
    - Build lattice combining them
  - Different security levels
    - Need to be *levels* – thus must be able to order
  - What if different DAC and MAC policies need to be integrated?

Multidomain Environments

- Heterogeneity exists at several levels



Multidomain Challenges

Key challenges

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

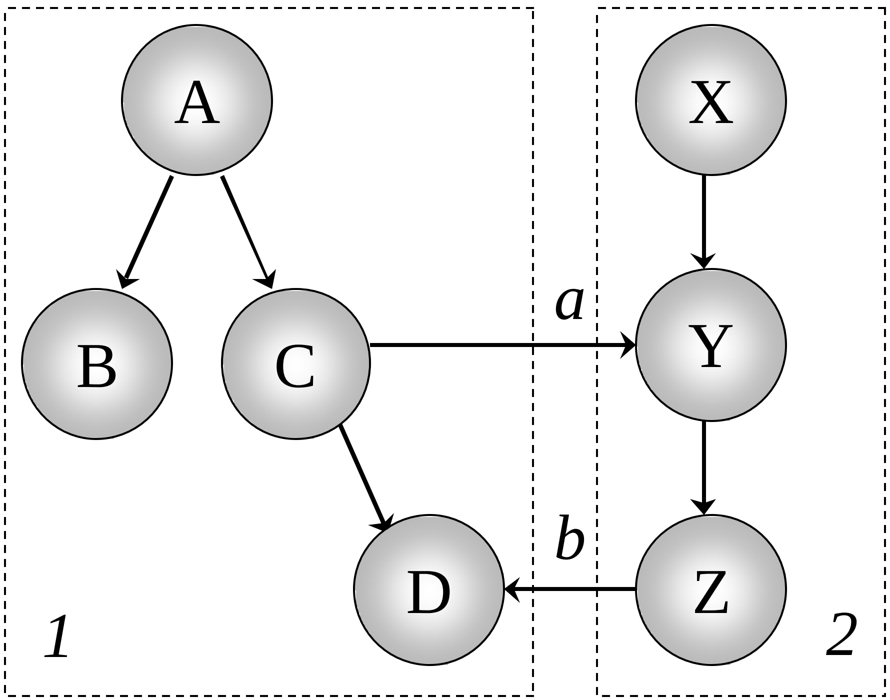
Semantic heterogeneity

- Different systems use different security policies
  - e.g., Chinese wall, BLP policies etc.
- Variations of the same policies
  - e.g., BLP model and its 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
- Different Commercial-Off-The-Self (COTS) products

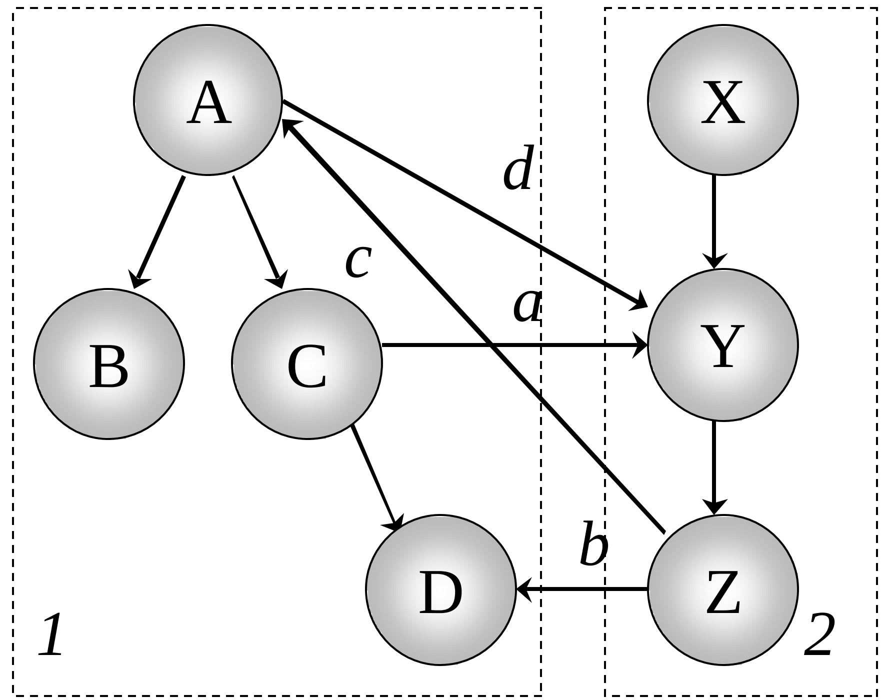
Secure Interoperability

- Principles of secure interoperation [Gong, 96]
  - *Principle of autonomy*
    - If an access is permitted within an individual system, it must also be permitted under secure interoperation
  - *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

Secure Interoperability (Example)



F_{12} = {a, b}



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

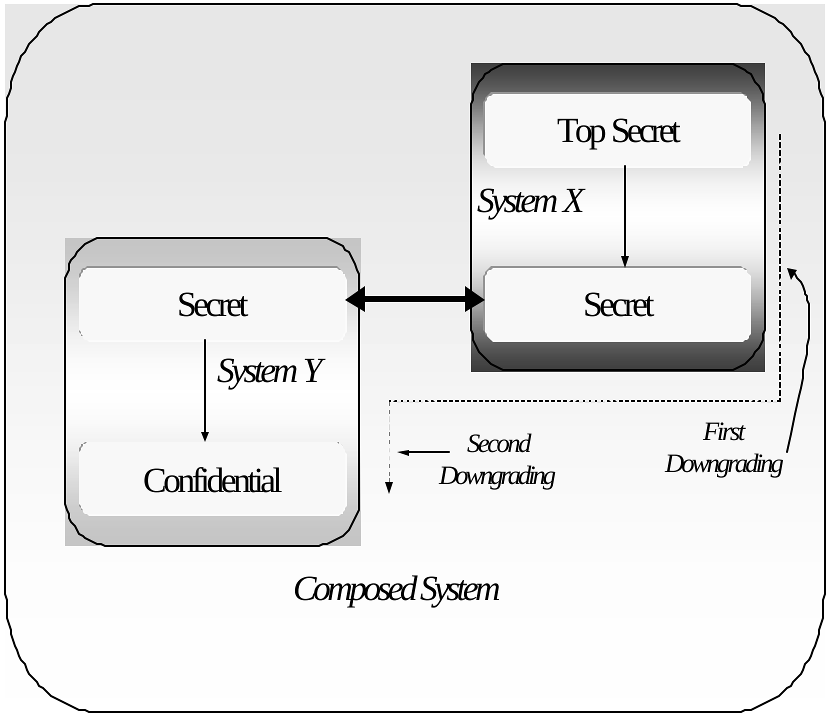
F_{12} - permitted access between systems 1 and 2

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

(2) F_{12} = {c}
Indirect access

Assurance and Risk Propagation & Security Management

- Assurance and Risk propagation
  - A breach in one component affects the whole environment
  - Cascading problem
- Management
  - Centralized/Decentralized
  - Managing metapolicy
  - Managing policy evolution



Design Principles

Design Principles for Security Mechanisms

- Principles
  - Least Privilege
  - Fail-Safe Defaults
  - Economy of Mechanism
  - Complete Mediation
  - Open Design
  - Separation of Privilege
  - Least Common Mechanism
  - Psychological Acceptability
- Based on the idea of *simplicity* and *restriction*

Overview

- Simplicity
  - Less to go wrong
  - Fewer possible inconsistencies
  - Easy to understand
- Restriction
  - Minimize access power (need to know)
  - Inhibit communication

Least Privilege

- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
    - RBAC!
  - Rights added as needed, discarded after use
    - Active sessions and dynamic separation of duty
  - Minimal protection domain
    - A subject should not have a right if the task does not need it

Fail-Safe Defaults

- Default action is to deny access
- If action fails, system as secure as when action began
  - Undo changes if actions do not complete
  - Transactions (commit)

Economy of Mechanism

- Keep the design and implementation as simple as possible
  - KISS Principle (Keep It Simple, Silly!)
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Interfaces and interactions

Complete Mediation

- Check every access to an object to ensure that access is allowed
- Usually done once, on first action
  - UNIX: Access checked on open, not checked thereafter
- If permissions change after, may get unauthorized access

Open Design

- Security should not depend on secrecy of design or implementation
  - Popularly misunderstood to mean that source code should be public
  - “Security through obscurity”
  - Does not apply to information such as passwords or cryptographic keys

Separation of Privilege

- Require multiple conditions to grant privilege
  - Example: Checks of $70000 must be signed by two people
  - Separation of duty
  - Defense in depth
    - Multiple levels of protection

Least Common Mechanism

- Mechanisms should not be shared
  - Information can flow along shared channels
  - Covert channels
- Isolation
  - Virtual machines
  - Sandboxes

Psychological Acceptability

- Security mechanisms should not add to difficulty of accessing resource
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here