Access Control

Access Control

- **Access control**: ensures that all *direct accesses* to object are authorized
- Protects against accidental and **malicious** threats by regulating the *reading, writing and execution* of data and programs
- Need:
  - Proper *user identification* and *authentication*
  - Information specifying the *access rights is protected* form modification

Access Control Requirement

- Cannot be bypassed
- Enforce least-privilege and need-to-know restrictions
- Enforce organizational policy

Access Control

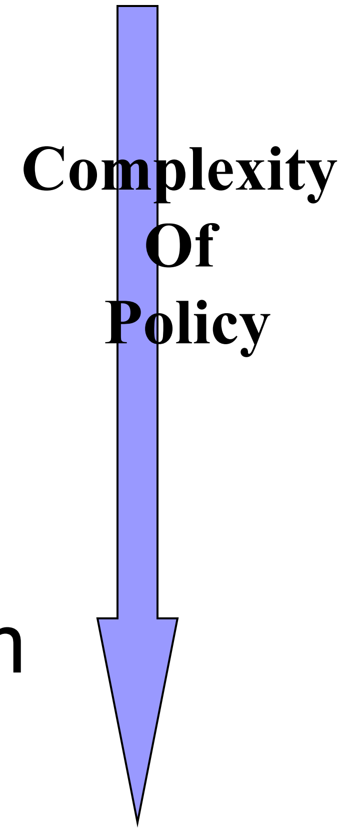
- **Protection objects**: system resources for which protection is desirable
  - Memory, file, directory, hardware resource, software resources, etc.
- **Subjects**: active entities requesting accesses to resources
  - User, owner, program, etc.
- **Access mode**: type of access
  - Read, write, execute

Access Control

- Access control components:
  - *Access control policy*: specifies the authorized accesses of a system
  - *Access control mechanism*: implements and enforces the policy
- **<u>Separation of components allows to:</u>**
  - Define access requirements independently from implementation
  - Compare different policies
  - Implement mechanisms that can enforce a wide range of policies

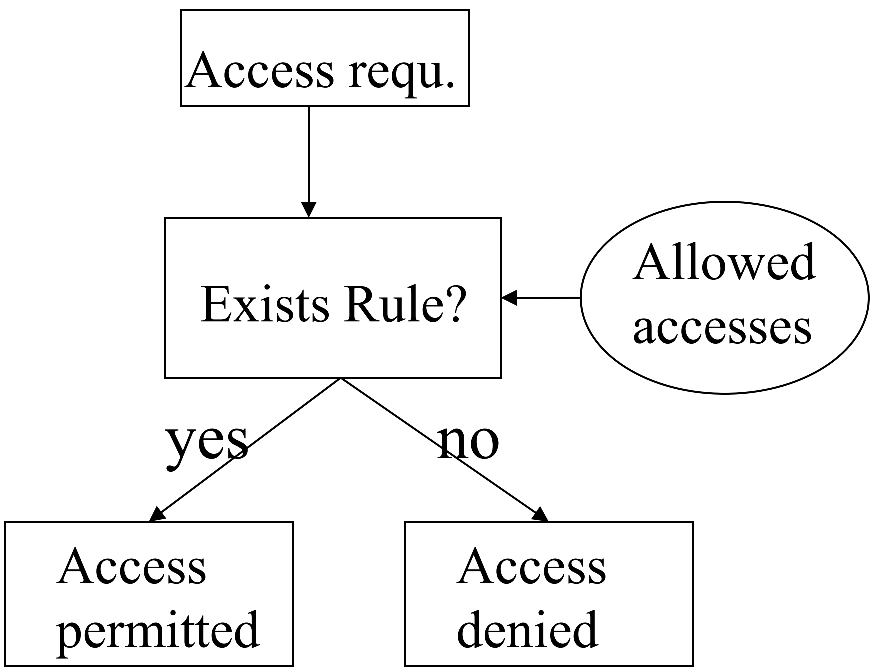
System Architecture and Policy

- Simple monolithic system
- Distributed homogeneous system under centralized control
- Distributed autonomous systems homogeneous domain
- Distributed heterogeneous system

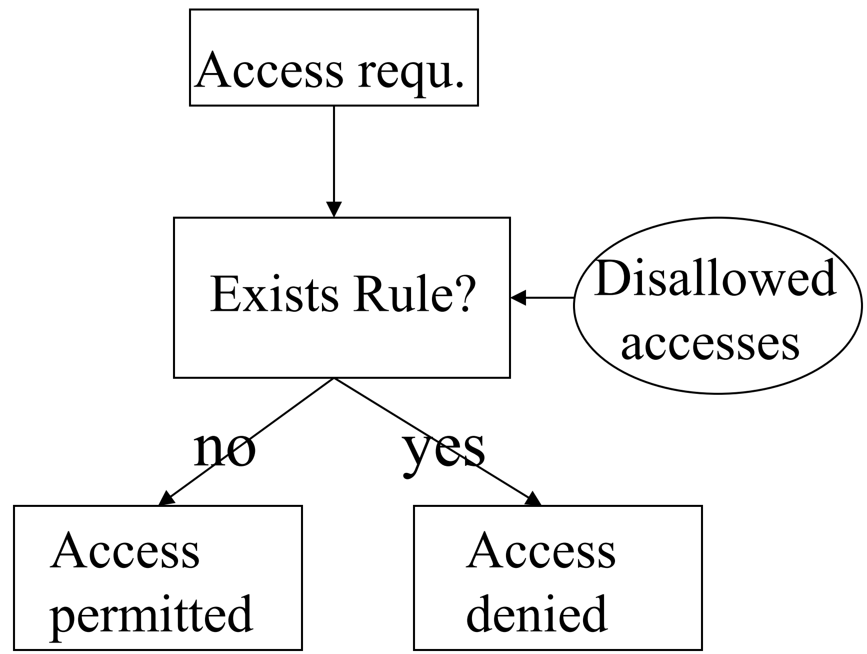


Closed v.s. Open Systems

Closed system
(minimum privilege)



Open System
(maximum privilege)



Negative Authorization

- Traditional systems: Mutual exclusion
- New systems: combined use of positive and negative authorizations
  - Support exceptions
  - Problems: How to deal with
    - Incompleteness – Default policy
    - Inconsistencies – Conflict resolution

Conflict Resolution

- Denial takes precedence
- Most specific takes precedence
- Most specific along a path takes precedence
- Priority-based
- Positional
- Grantor and time-dependent
- Single strategy vs. combination of strategies

Policy Specification Language

- Express policy concepts
- Required properties of policy languages:
  - Support access control, delegation, and obligation
  - Provide structuring constructs to handle large systems
  - Support composite policies
  - Must be able to analyze policies for conflicts and inconsistencies
  - Extensible
  - Comprehensible and easy to use

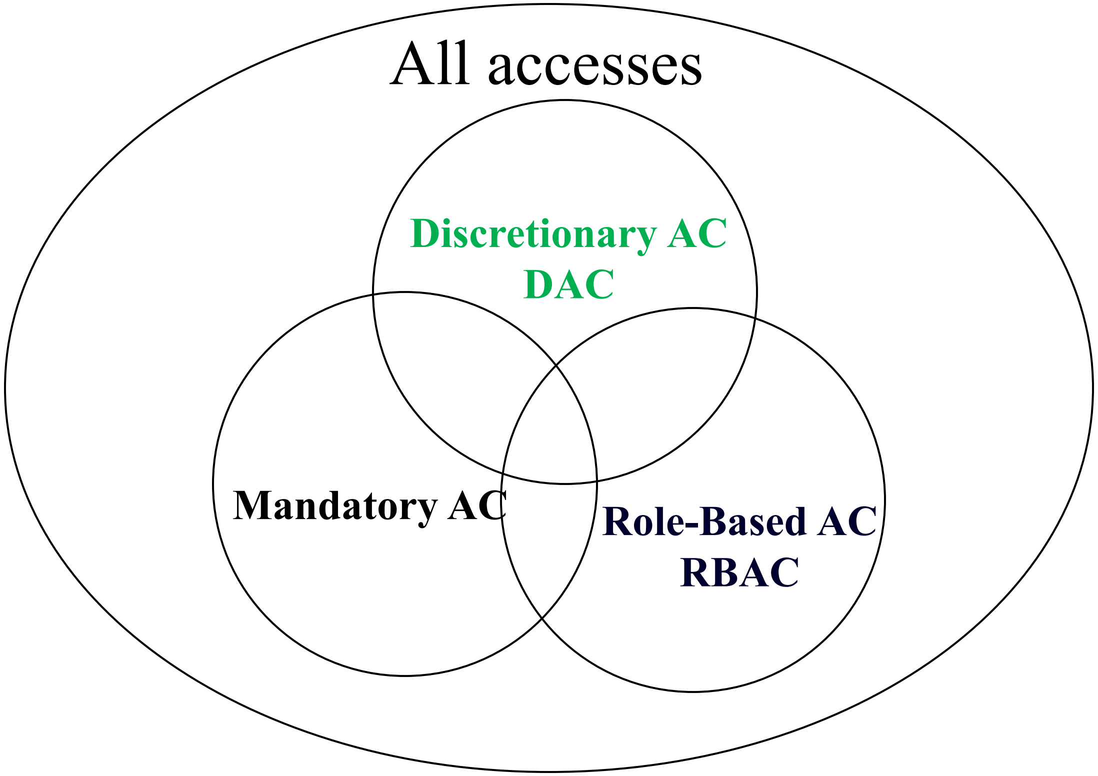
Policy Development

- Policy maker:
  - Start with high-level policies
  - Refine high-level policies to low-level policy specification
    - Determine resources required to satisfy the policy
    - Translate high-level policies into enforceable versions
    - Support analysis that verifies that lower level policies actually meet the needs of higher level ones.

Authorization Management

- *Who* can grant and revoke access rights?
- *Centralized* administration: security officer
- *Decentralized* administration: locally autonomous systems
- *Hierarchical decentralization*: security officer > departmental system administrator > Windows NT administrator
- *Ownership based*: owner of data may grant access to other to his/her data (possibly with grant option)
- *Cooperative authorization*: predefined groups of users or predefined number of users may access data

Access Control Models



Access Control
DAC

Discretionary Access Control

- Access control is based on
  - User's identity and
  - Access control rules
- Most common administration: owner based
  - Users can protect what they own
  - Owner may grant access to others
  - Owner may define the type of access given to others

Access Matrix Model

OBJECTS AND SUBJECTS

S U B J E C T S

	File 1	File 2
Joe	Read Write Own	Read
Sam		Read Write Own

Implementation

Access Control List (column) (ACL)

File 1	**File 2**
Joe:Read	Joe:Read
Joe:Write	Sam:Read
Joe:Own	Sam:Write
	Sam:Own

Capability List (row)

Joe: File 1/Read, File 1/Write, File 1/Own, File 2/Read
Sam: File 2/Read, File 2/Write, File 2/Own

Access Control Triples

Subject	Access	Object
Joe	Read	File 1
Joe	Write	File 1
Joe	Own	File 1
Joe	Read	File 2
Sam	Read	File 2
Sam	Write	File 2
Sam	Own	File 2

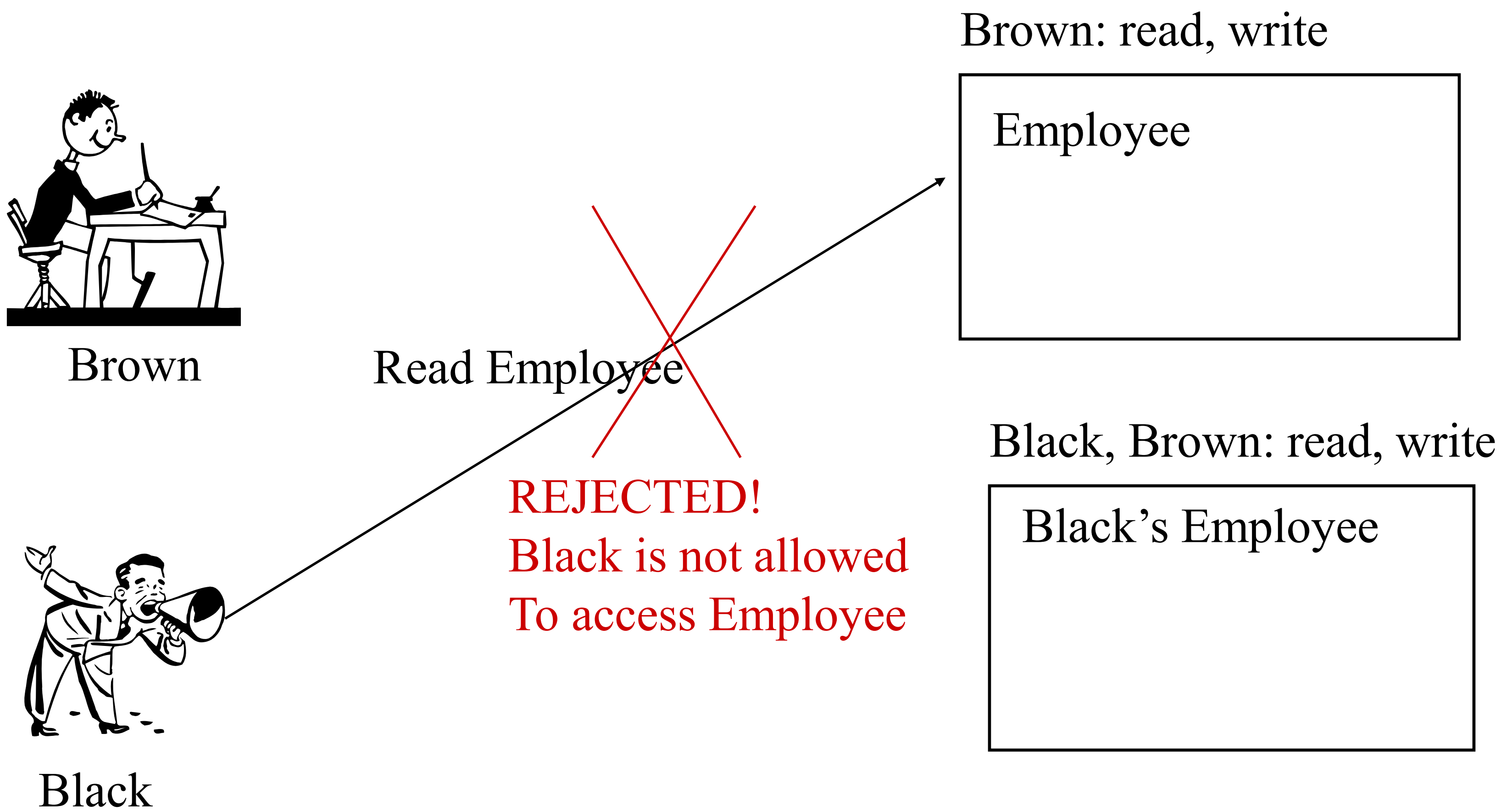
ACL vs. Capabilities

- ACL:
  - Per object based
  - Good for file systems
- Capabilities:
  - Per subject based
  - Good for environment with dynamic, short-lived subjects

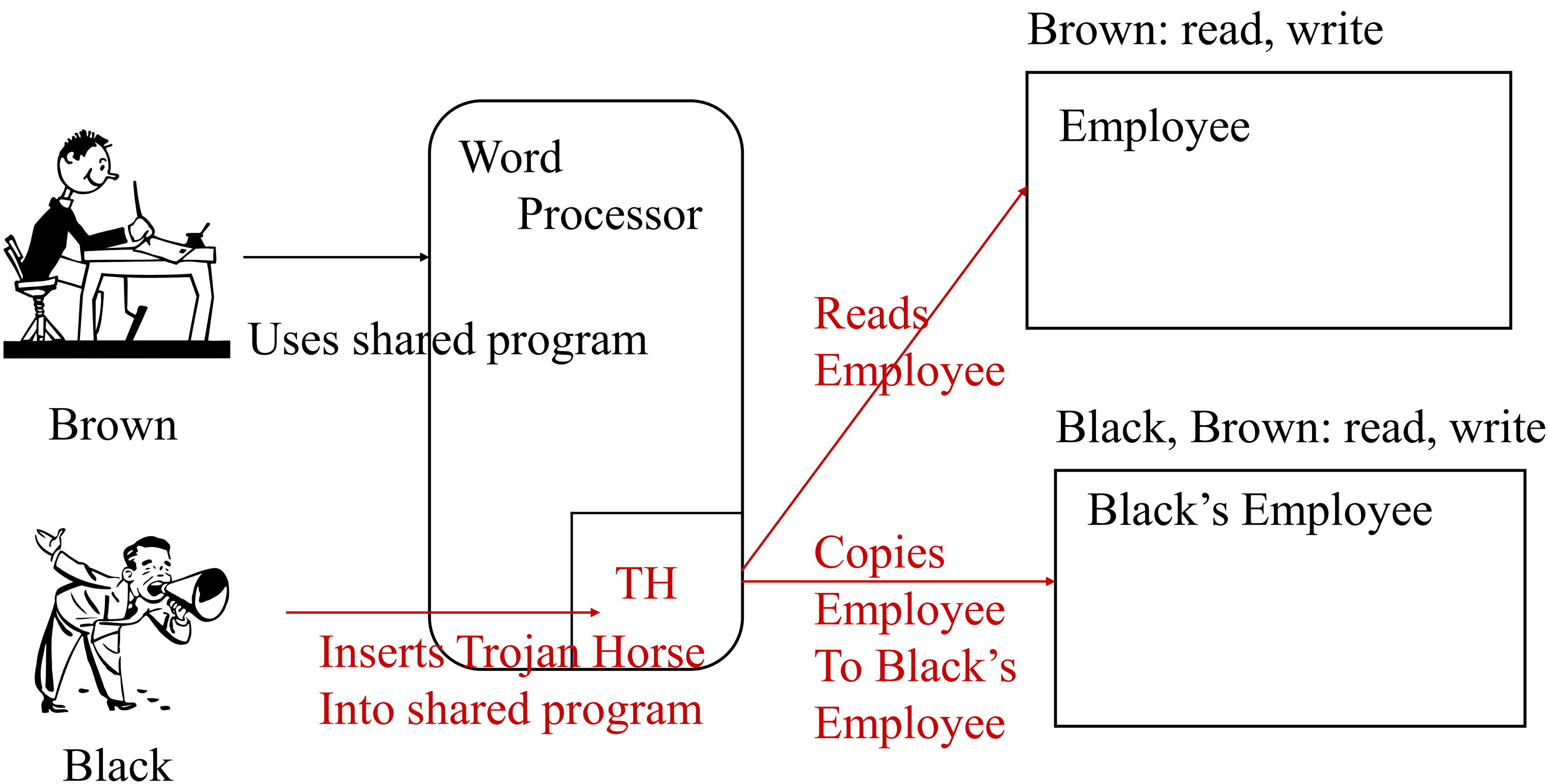
Access Control Conditions

- **Data-dependent conditions**: access constraints based on the value of the accessed data
- **Time-dependent**: access constraints based on the time of the data access
- **Context-dependent**: access constraints based on combinations on data which can be accessed
- **History-dependent**: access constraints based on previously accessed data

DAC and Trojan Horse



DAC and Trojan Horse



DAC Overview

- Advantages:
  - Intuitive
  - Easy to implement
- Disadvantages:
  - Inherent vulnerability (look TH example)
  - Maintenance of ACL or Capability lists
  - Maintenance of Grant/Revoke
  - Limited power of negative authorization

Access Control
RBAC

RBAC Motivation

- Multi-user systems
- Multi-application systems
- Permissions are associated with roles
- Role-permission assignments are persistent v.s. user-permission assignments
- Intuitive: competency, authority and responsibility

Motivation

- <u>Express organizational policies</u>
  - Separation of duties
  - Delegation of authority
- <u>Flexible</u>: easy to modify to meet new security requirements
- <u>Supports</u>
  - Least-privilege
  - Separation of duties
  - Data abstraction

RBAC

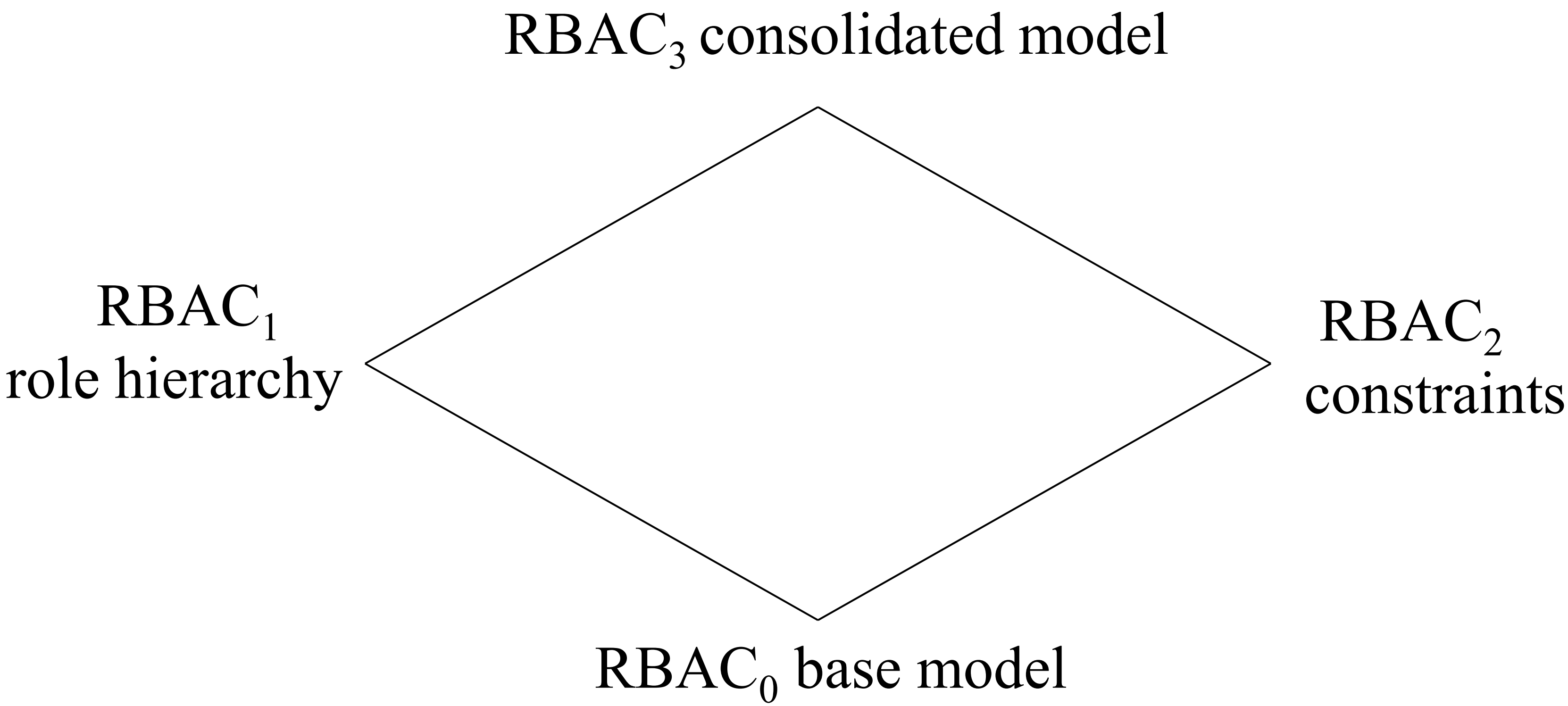
- Allows to express security requirements but
  CANNOT ENFORCE THESE PRINCIPLES

e.g., RBAC can be configured to enforce BLP rules but its correctness depend on the configuration done by the system security officer.

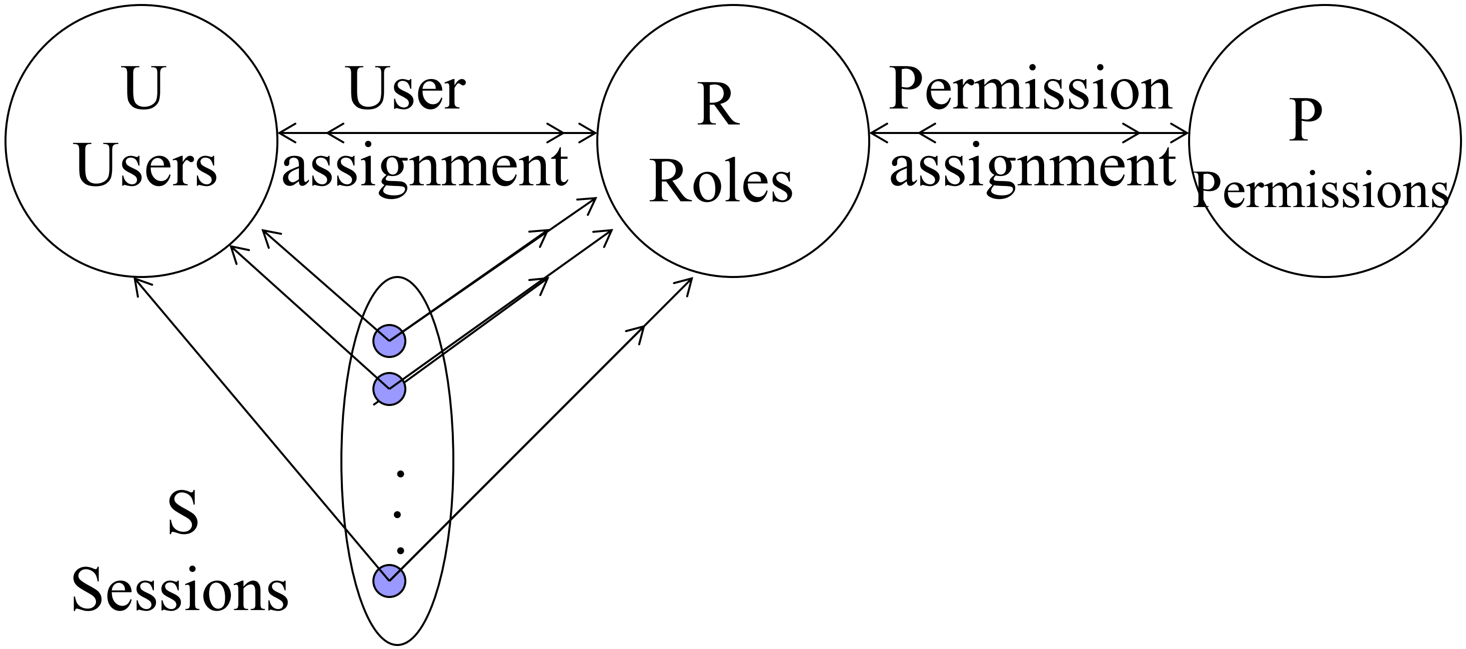
Roles

- <u>User group</u>: collection of user with possibly different permissions
- <u>Role</u>: mediator between collection of users and collection of permissions
- RBAC independent from DAC and MAC (they may coexist)
- <u>RBAC is policy neutral</u>: configuration of RBAC determines the policy to be enforced

RBAC


RBAC$_3$ consolidated model
RBAC$_1$ role hierarchy
RBAC$_2$ constraints
RBAC$_0$ base model

RBAC$_0$



U
Users
User
assignment
R
Roles
Permission
assignment
P
Permissions
S
Sessions

RBAC0

- User: human beings
- Role: job function (title)
- Permission: approval of a mode of access
  - (object, access mode)
  - Always positive
  - Abstract representation
  - Can apply to single object or to many

$RBAC_0$

- UA: user-role assignments
  - Many-to-many
- PA: role-permission assignment
  - Many-to-many
- Session: mapping of a single user to possibly may roles
  - Multiple roles can be activated simultaneously
  - Permissions: union of permissions from all roles
  - Each session is associated with a single user
  - User may have multiple sessions at the same time

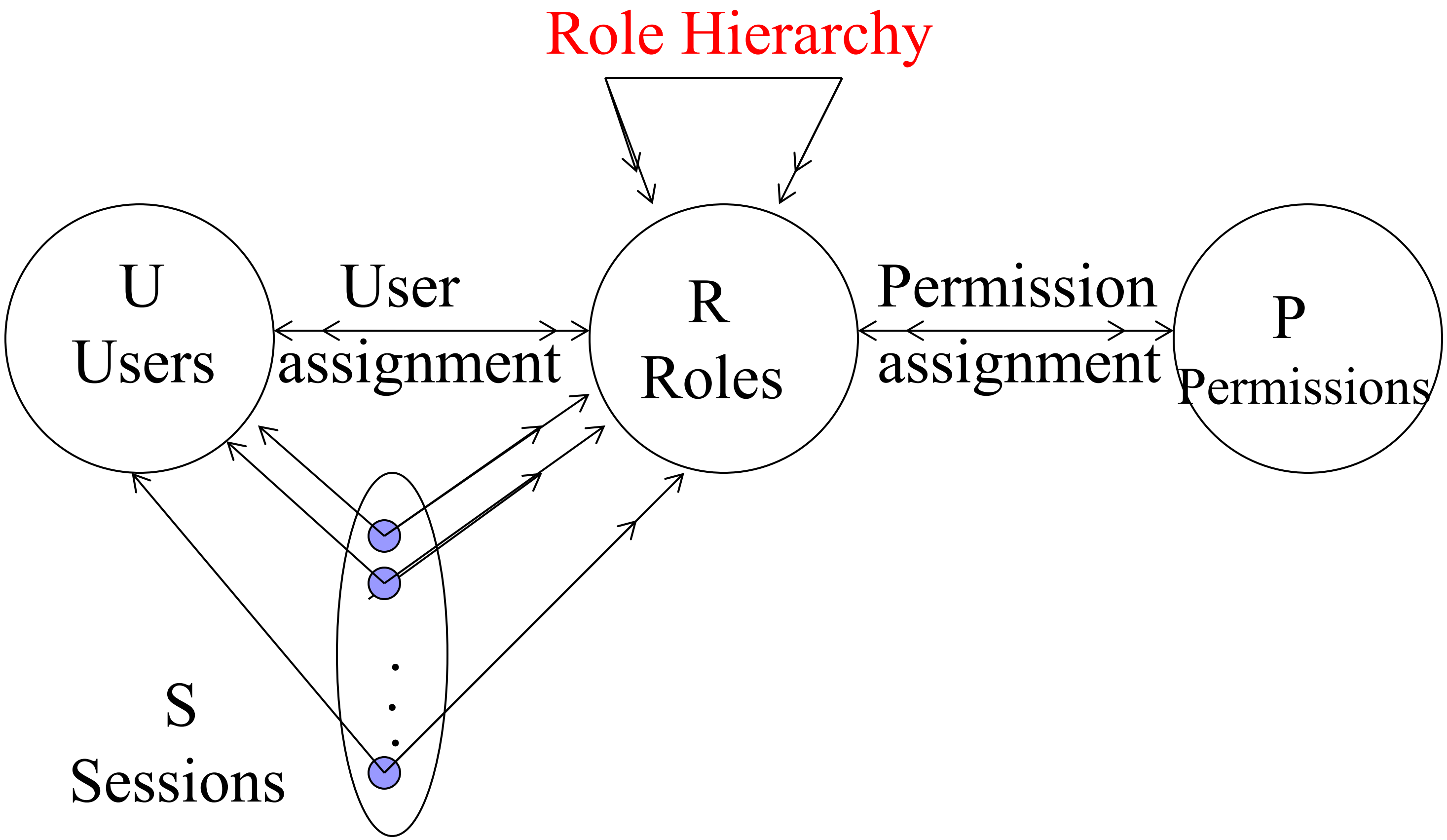
RBAC$_0$ Components

- **U**sers, **R**oles, **P**ermissions, **S**essions
- $PA \subseteq P \times R$ (many-to-many)
- $UA \subseteq U \times R$ (many-to-many)
- user: $S \rightarrow U$, mapping each session s_i to a single user user(s_i)
- roles: $S \rightarrow 2^R$, mapping each session s_i to a set of roles roles(s_i) $\subseteq \{r \mid (user(s_i),r) \in UA\}$ and s_i has permissions $\cup_{r \in roles(si)} \{p \mid (p,r) \in PA\}$

$RBAC_0$

- Permissions apply to data and resource objects only
- <u>Permissions do NOT apply to RBAC components</u>
- Administrative permissions: modify U,R,S,P
- Session: under the control of user to
  - Activate any subset of permitted roles
  - Change roles within a session

RBAC$_1$



Role Hierarchy
U
Users
User
assignment
R
Roles
Permission
assignment
P
Permissions
S
Sessions

$RBAC_1$

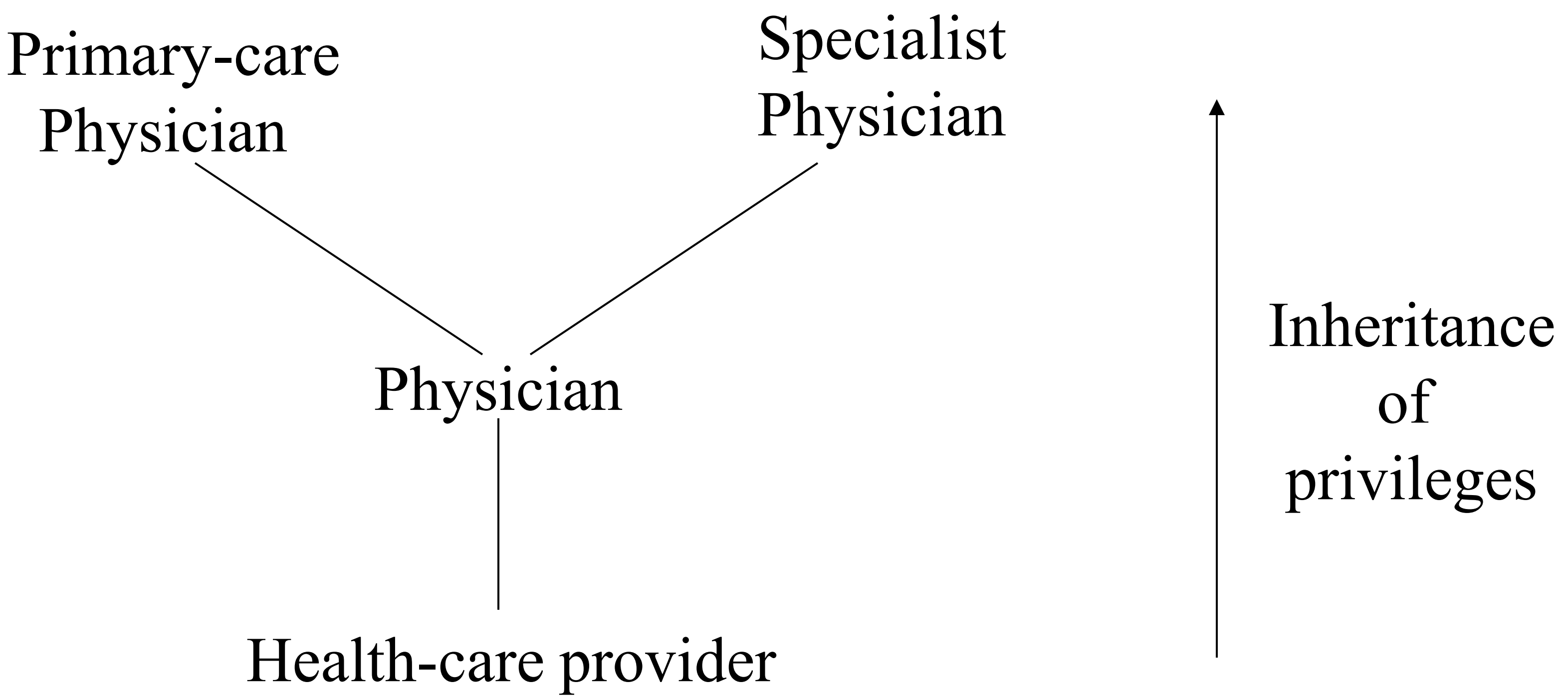
- Structuring roles
- Inheritance of permission from junior role (bottom) to senior role (top)
- Partial order
  - Reflexive
  - Transitive
  - Anti-symmetric

RBAC$_1$ Components

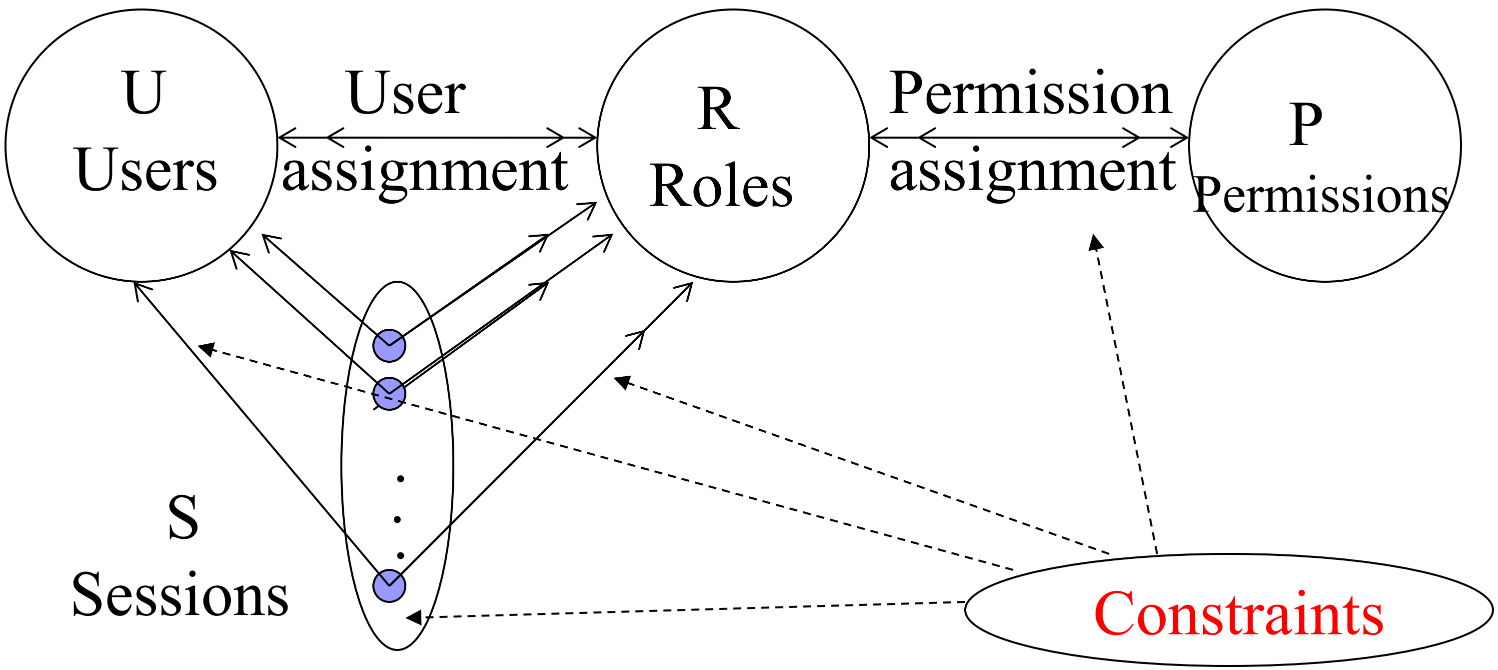
- Same as RBAC$_0$: **U**sers, **R**oles, **P**ermissions, **S**essions, PA $\subseteq$ P x R, UA $\subseteq$ U x R, user: S $\rightarrow$ U, mapping each session s_i to a single user user(s_i)
- RH $\subseteq$ R x R, partial order ($\geq$ dominance)
- roles: S $\rightarrow$ 2^R, mapping each session s_i to a set of roles roles(s_i) $\subseteq$ $\{r \mid (\exists r' \geq r)\ [(user(s_i), r') \in UA]\}$ and s_i has permissions $\cup_{r \in roles(si)} \{p \mid (\exists r'' \leq r)\ [(p, r'') \in PA]\}$

$RBAC_1$

Role Hierarchy



RBAC$_2$



U
Users
User
assignment
R
Roles
Permission
assignment
P
Permissions
S
Sessions
Constraints

$RBAC_2$ – Constraints

- Enforces high-level organizational policies
- Management of decentralized security
- Constraints define “acceptable” and “not acceptable” accesses

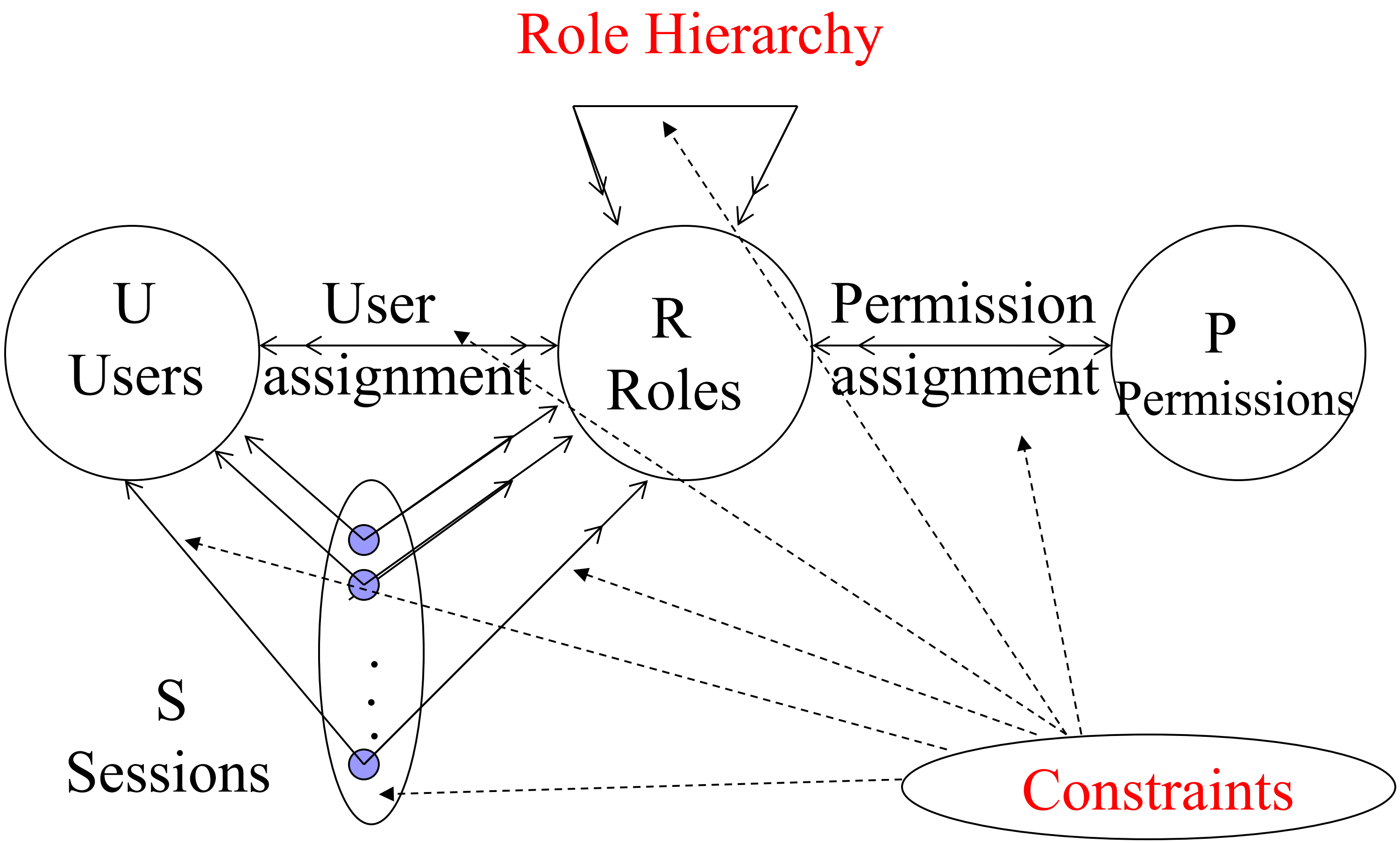
$RBAC_2$

- Mutually exclusive roles
- Dual constraint of permission assignments (permission assigned to at most one mutually exclusive role)
- Cardinality constraints (e.g., # of roles an individual can belong)
- Prerequisite roles

$RBAC_2$

- Constraints can apply to sessions, user and roles functions

RBAC$_3$


Role Hierarchy
U
Users
User
assignment
R
Roles
Permission
assignment
P
Permissions
S
Sessions
Constraints

Questions?