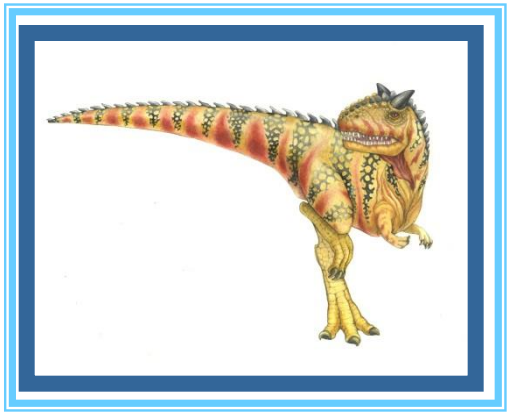
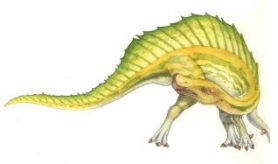
Chapter 14: Protection



Goals of Protection

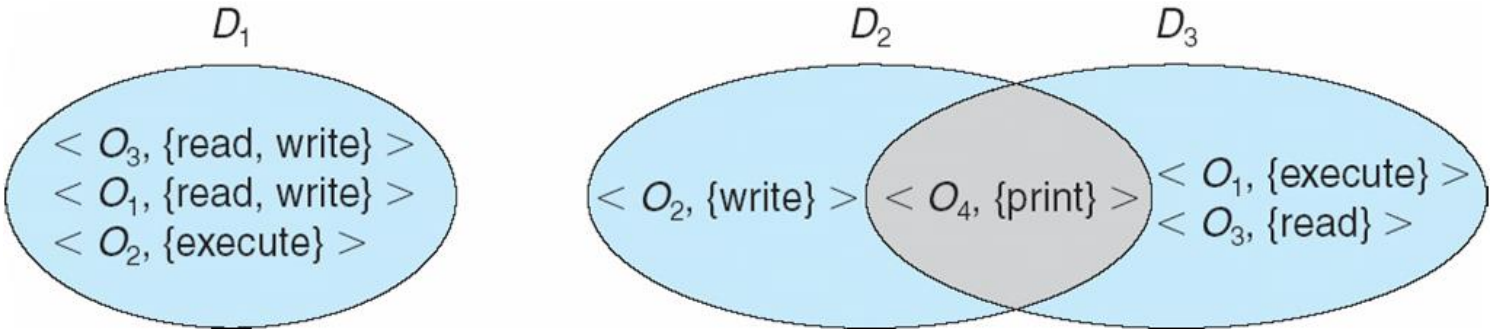
- Operating system consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations.
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.
- Guiding principle – principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks

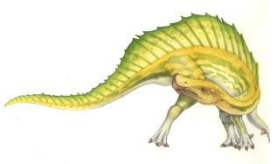


Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object.

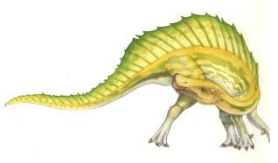
- Domain = set of access-rights





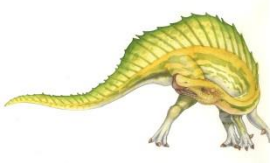
Access Matrix

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- *Access(i, j)* is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$



Access Matrix

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	



Implementation of Access Matrix

- Each column = Access-control list for one object
  Defines who can perform what operation.

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

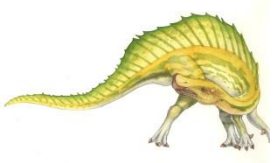
  ⋮

- Each Row = Capability List (like a key)
  Fore each domain, what operations allowed on what objects.

  Object 1 – Read

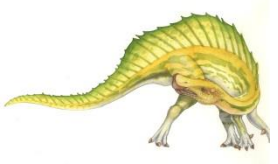
  Object 4 – Read, Write, Execute

  Object 5 – Read, Write, Delete, Copy

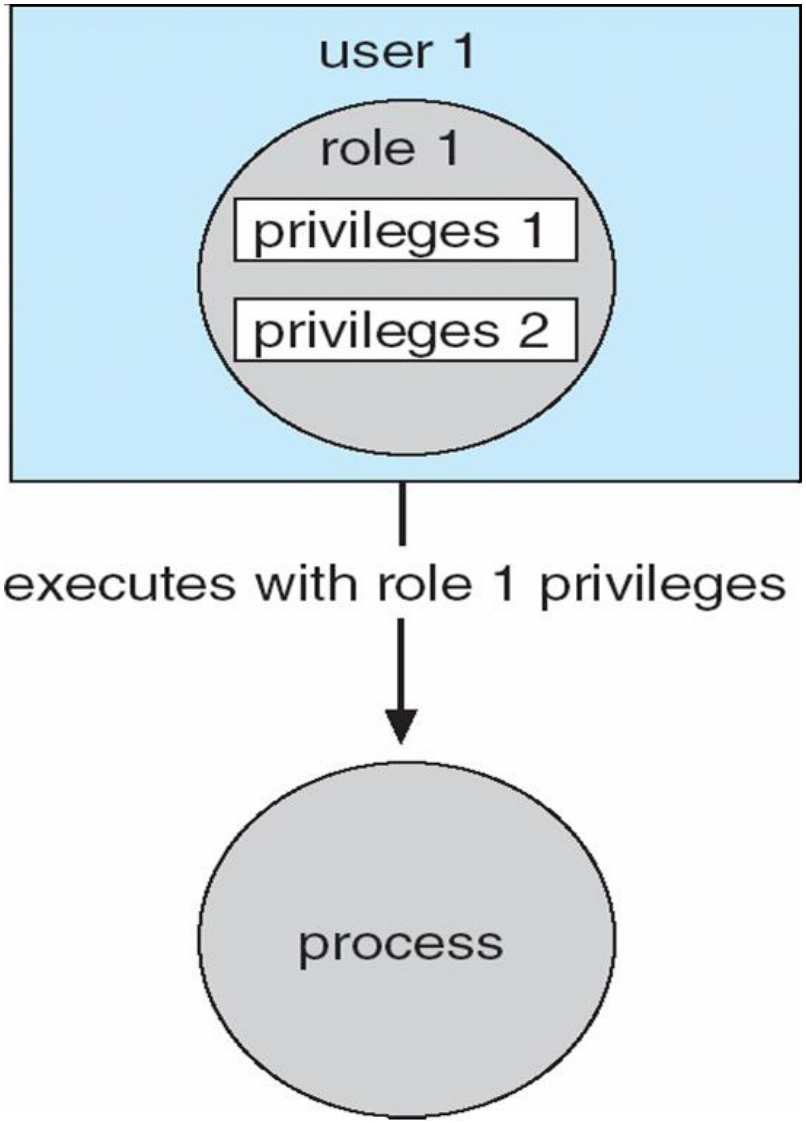


Revocation of Access Rights

- *Access List* – Delete access rights from access list.
  - Simple
  - Immediate
- *Capability List* – Scheme required to locate capability in the system before capability can be revoked.
  - Reacquisition
  - Back-pointers
  - Indirection
  - Keys

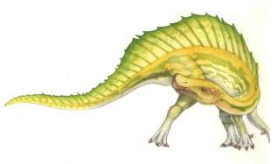


Role-based Access Control in Solaris 10



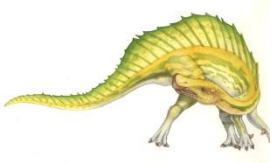
Role-Based Access Control (RBAC)

- Users are assigned to roles.
  - Example: UA = {(Bob, Doctor)}
- Permissions are associated with roles.
  - Example: PA = {(Doctor, Modify, Prescriptions)}
- A user has a permission if he is a member of some role with that permission.



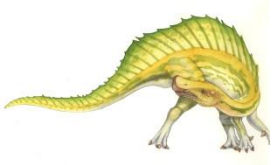
ARBAC Syntax

- Example:
  - can_assign(President, Professor $\wedge$ ¬DeptChair, Dean): Administrator in role President can assign a user in role Professor and not in role DeptChair to role Dean.
    - Professor is a positive precondition, DeptChair is a negative precondition and Dean is a target.
- Example:
  - can_revoke(DeptChair, TA): an administrator in role DeptChair can remove any user from role TA.
- Roles President and DeptChair are administrative: have administrative permissions, i.e. are the first components of can_assign or can_revoke rules.
- All other roles are regular.
- Separate Administration Restriction: administrative roles and regular roles are disjoint.



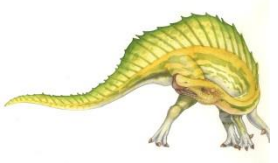
Analysis of ARBAC Policies

- Large organizations have large and complex policies which are managed by multiple administrators.
- The effects of an ARBAC policy are often hard to understand by manual inspection alone.
  - Changes by one administrator may interact in unintended ways with changes by other administrators.
- ARBAC Policy Analysis can help by answering questions such as:
  - User-Role Reachability Problem: given an initial RBAC policy, an ARBAC policy, a set of administrators, a target user, and a set of roles (called the "goal"), is it possible for those administrators to modify the RBAC policy so that the target user is a member of those roles?



Algorithm

- User-Role Reachability: can user u be assigned to all roles in goal = $\{r_1 ... r_n\}$ by a group of administrators A?
- Approach: express the analysis problem as a finite state machine where:
  - Initial State: initial role assignments of users.
  - Accept State: a state where user u is assigned to all roles in goal.
  - Transitions: changes allowed by ARBAC policy.



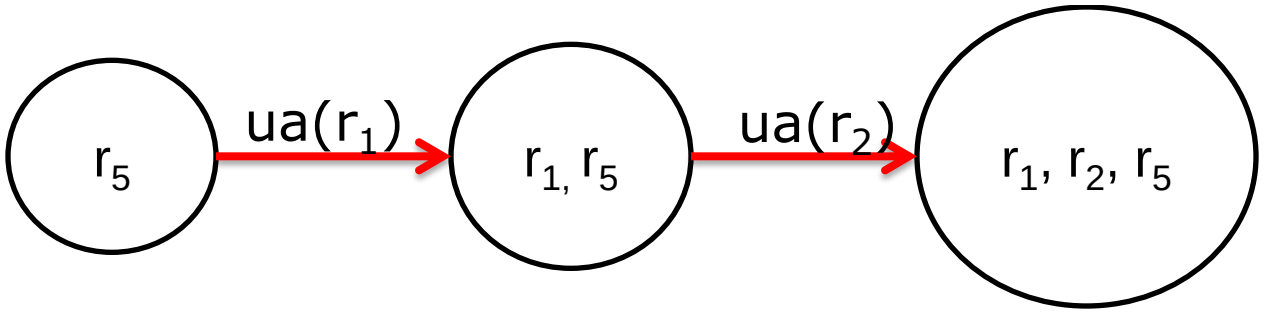
Example

ARBAC Policy:

- 1. can_assign(r_a, $\neg r_6$, r_1)
- 2. can_assign(r_a, $r_4 \wedge \neg r_7$, r_5)
- 3. can_assign(r_a, r_1, r_2)
- 4. can_assign(r_a, r_1, r_6)

User-role Reachability Query: Can administrator in $\{r_a\}$ assign user initially in $\{r_5\}$ to $\{r_1, r_2\}$?

Answer: Yes



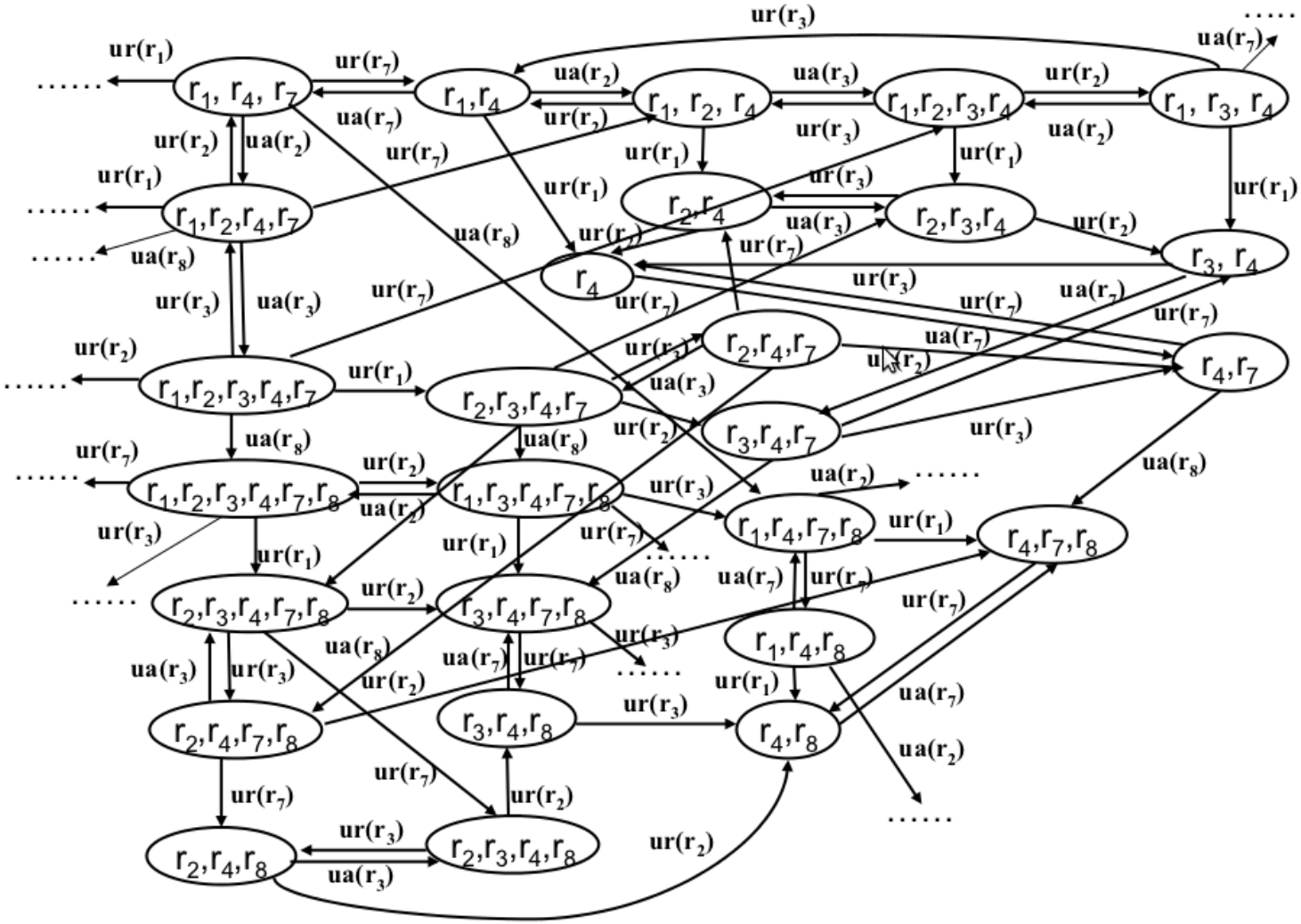
ARBAC Policy:

1. can_assign(r_a, $\neg r_1$, r_2)
2. can_assign(r_a, r_2, r_3)
3. can_assign(r_a, $r_3 \wedge \neg r_4$, r_5)
4. can_assign(r_a, r_5, r_6)
5. can_assign(r_a, $\neg r_2$, r_7)
6. can_assign(r_a, r_7, r_8)
7. can_revoke(r_a, r_1)
8. can_revoke(r_a, r_2)
9. can_revoke(r_a, r_3)
10. can_revoke(r_a, r_5)
11. can_revoke(r_a, r_6)
12. can_revoke(r_a, r_7)

User-role Reachability Query: Can administrator in $\{r_a\}$ assign user initially in $\{r_1, r_4, r_7\}$ to $\{r_6\}$?

Example Contd.

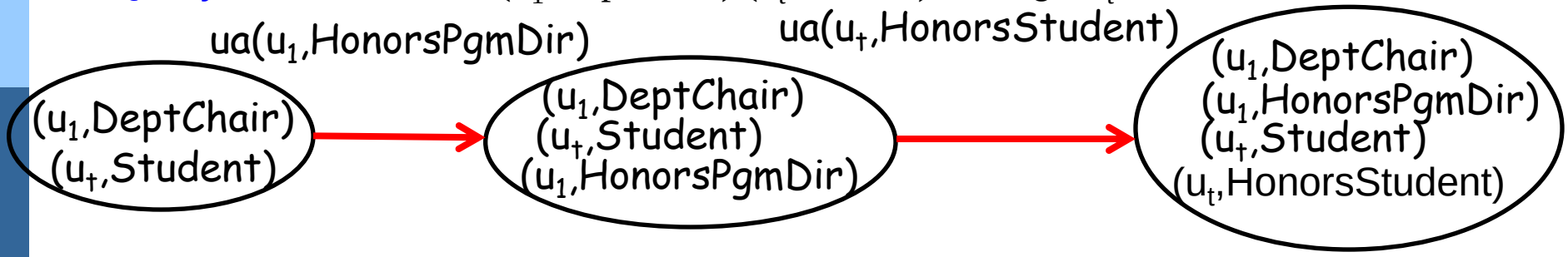
- 32 states, 96 transitions:



Beyond Separate Administration

- In realistic ARBAC policies the sets of administrative and regular roles may not always be disjoint.
  - They violate the separate administration restriction
- Administrators may assign themselves to new roles:
  - Example:
    1. can_assign(DeptChair, DeptChair, HonorsPgmDir)
    2. can_assign(HonorsPgmDir, Student, HonorsStudent)

Query: Can users in $\{(u_1,\text{DeptChair}),(u_t,\text{Student})\}$ assign u_t to role $\{\text{HonorsStudent}\}$?



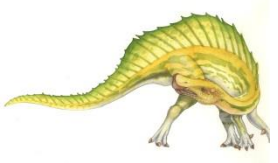
Beyond Separate Administration

- ARBAC Policy:

- Role Hierarchy: $r_3 \geq r_2$; $r_3 \geq r_8$
- 1. can assign(r_8, $r_9 \wedge \neg r_1$, r_4)
- 2. can assign(r_1, $r_1 \wedge r_{10} \wedge r_5$, r_9)
- 3. can assign(r_1, $r_3 \wedge \neg r_5$, r_{10})
- 4. can assign(r_3, $r_1 \wedge \neg r_6 \wedge \neg r_4$, r_2)
- 5. can revoke(r_3, r_5)
- 6. can assign(r_3, $r_9 \wedge \neg r_1$, r_4)
- 7. can assign(r_2, r_3, r_4)
- 8. can assign(r_5, true, r_6)
- 9. can assign(r_3, r_3, r_4)
- 10. can assign(r_6, true, r_7)
- 11. can revoke(r_3, r_1)

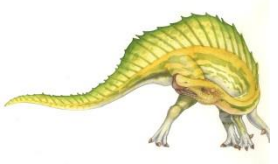
Query: Can users in $\{(u_1,r_1), (u_1,r_3), (u_1,r_5), (u_2,r_1), (u_2,r_3), (u_2,r_5), (u_t,r_1), (u_t,r_3), (u_t,r_5)\}$ together assign user u_t to roles $\{r_4,r_7\}$?

- The graph contains 5,312 states and 27,216 transitions!



Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.



End of Chapter 14