On the Formal Definition of Separation-of-Duty (SoD) Policies and their Composition

Virgil D. Gligor
Department of Electrical Engineering
University of Maryland
College Park, Maryland 20742
gligor@eng.umd.edu

Serban I. Gavrila
VDG Inc.
6009 Brookside Drive
Chevy Chase, Maryland 20815
gavrila@csmes.ncsl.nist.gov

David Ferraiolo
NIST
US Department of Commerce
Gaithersburg, Maryland 20899
ferraiolo@csmes.ncsl.nist.gov

May 5, 1998

SoD premise:

• Violations that require collusion are less likely to happen

SoD goals:

• Separate sensitive tasks of an application such that
integrity violations => collusion
• Minimize risk of collusion
by *careful* assignment of users to separate tasks

SoD implementation:

• Define integrity property of an application
• Partition application into separate operations and objects
• *carefully* assign of users to separate application partitions

SoD Policies

Advantage:

- wide-spread acceptance by business, industry, government

Drawbacks:

- *application-oriented policy*

=> limited scope

=> separate administration

- *family of policies*

=> required system flexibility

- *uncertain policy interpretation*

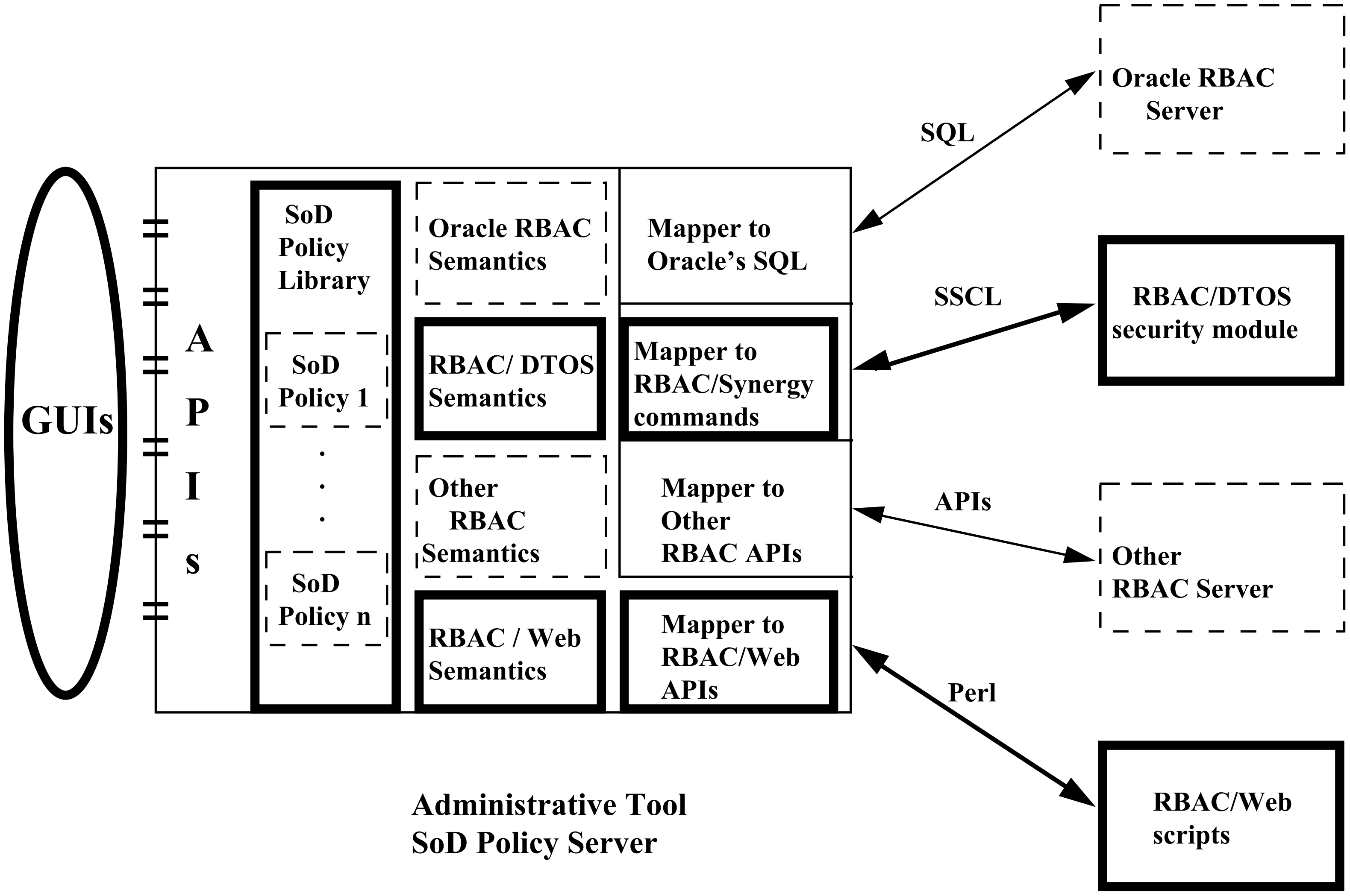
=> uncertain relative strength

Drawbacks:	**Mitigation:**
- *application-oriented policy* **=> limited scope** **=> separate administration**	- ***make it a feature of a global policy***
- *family of policies* **=> required system flexibility**	- ***provide administrative tools***
- *uncertain policy interpretation* **=> uncertain relative strength**	- ***define formally***

Solution: Define, implement, and administer SoD policies in systems suporting Role-Based Access Control (RBAC)

$$\begin{tikzcd} \{ \text{users} \} \arrow[r] & \{ \text{roles} \} : \{\text{operations}\} \arrow[r] & \{ \text{objects} \} \end{tikzcd}$$

Vision: SoD Administrative Tool



Systems

- *state machine*
  STATES, SUBJECTS, USERS, OPERATIONS, OBJECTS

- *state transitions*
  - commands: $op(s_1, S, obj, s_2)$
  - command sequence: $op_1(s_0, S_1, obj, s_1)op_2(s_1, S_2, obj_2, s_2)\ldots,$
  - tranquil commands: do not alter security attributes

- *system:* a set of command sequences with start states s_0 in STATES_0.

- *secure state, commands*: those that satisfy properties

- *reachable state*: a state appearing in a command sequence of a system

- *secure system*: all state transitions and reachable states are secure

- Ω : set of all command sequences of a secure system

Applications and Executability

- application: App = [ObjSet, OpSet, Plan]

  - **plan: a finite set of pairs $\{(\mathbf{obj_i}, \mathbf{op_i})\}$**
  - ordered plan: an ordered set of pairs $\{(\text{obj}_i, \text{op}_i)\}$
  - plans with "operation bracketing" (e.g., least-privilege princ.)

    . . .

- $\text{App}_1 \cup \text{App}_2 =$

  $[\text{ObjSet}_1 \cup \text{ObjSet}_2,\ \text{OpSet}_1 \cup \text{OpSet}_2,\ \text{Plan}_1 \cup \text{Plan}_2]$

- command sequence σ *executes* App if for any pair (obj_i, op_i) in Plan there is a command $op_i(s_k, S, obj_i, s_{k+1})$ in σ

Property Types

P = Attribute (AT) properties $\wedge$

Access Management (AM) properties $\wedge$

Access Authorization (AA) properties

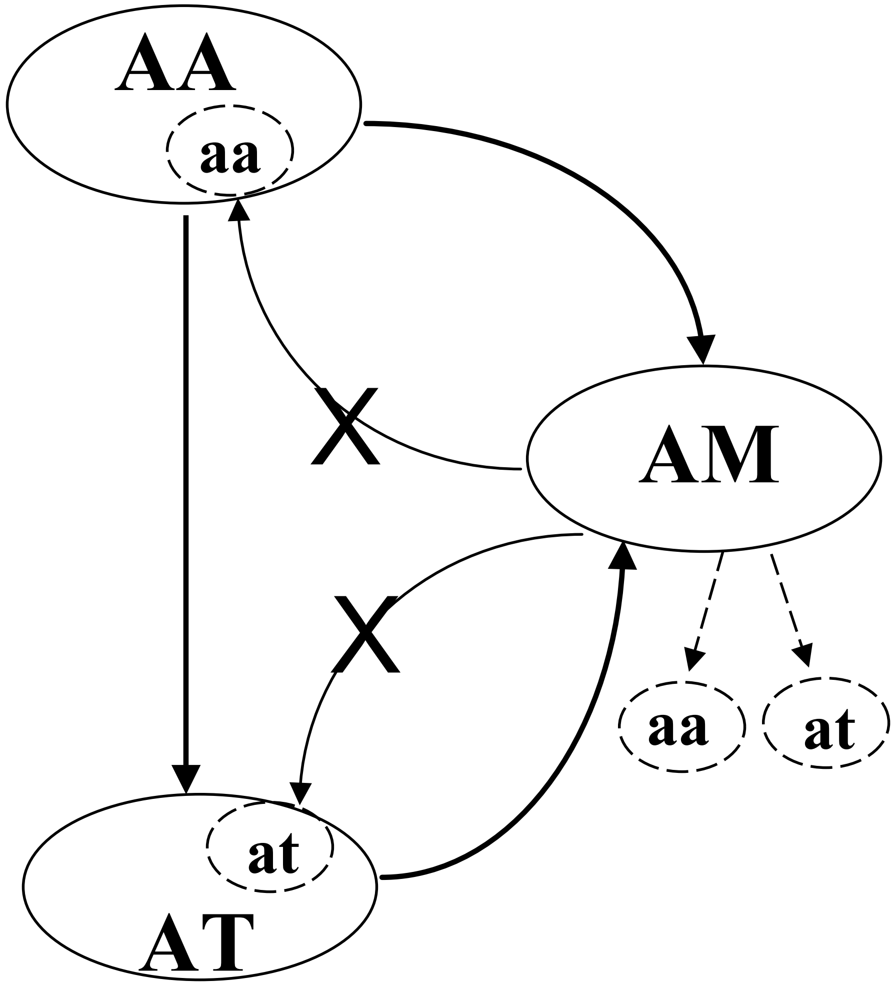
Examples of Property Types

- Attribute (**AT**) Properties
  - security (integrity) levels, partial order, lattice property
  - roles, hierarchy, permissions, membership, inheritance
- Access Management (**AM**) Properties
  - distribution, review, revocation of permissions
    - selectivity, transitivity, independence ...
  - object / subject creation and destruction
  - object encapsulation
- Access Authorization (**AA**) Properties
  - required subject and object attributes for access
    - BLP, Biba, RBAC, UNIX ...

Property Dependencies

“uses”
$\longrightarrow$

other types of dependencies exist



Individual policy properties cannot be composed independently

Policy Structure

$$P = \underset{\uparrow}{P} \wedge \underbrace{\mathrm{Admin}(P) \wedge \mathrm{Compat}(P, App)}_{\uparrow}$$

- access management
- access authorization
- attribute properties

Safety Properties

Safety or Liveness Properties ?

SoD Policy Structure

$$\text{SoD-P} = \text{SoD-P} \wedge \text{Admin(SoD-P)} \wedge \text{Compat(SoD-P, App)} \wedge \text{RBAC-P}$$

Admin(P)

P: a set of tranquil command sequences with the start state in $STATES_0$

for all

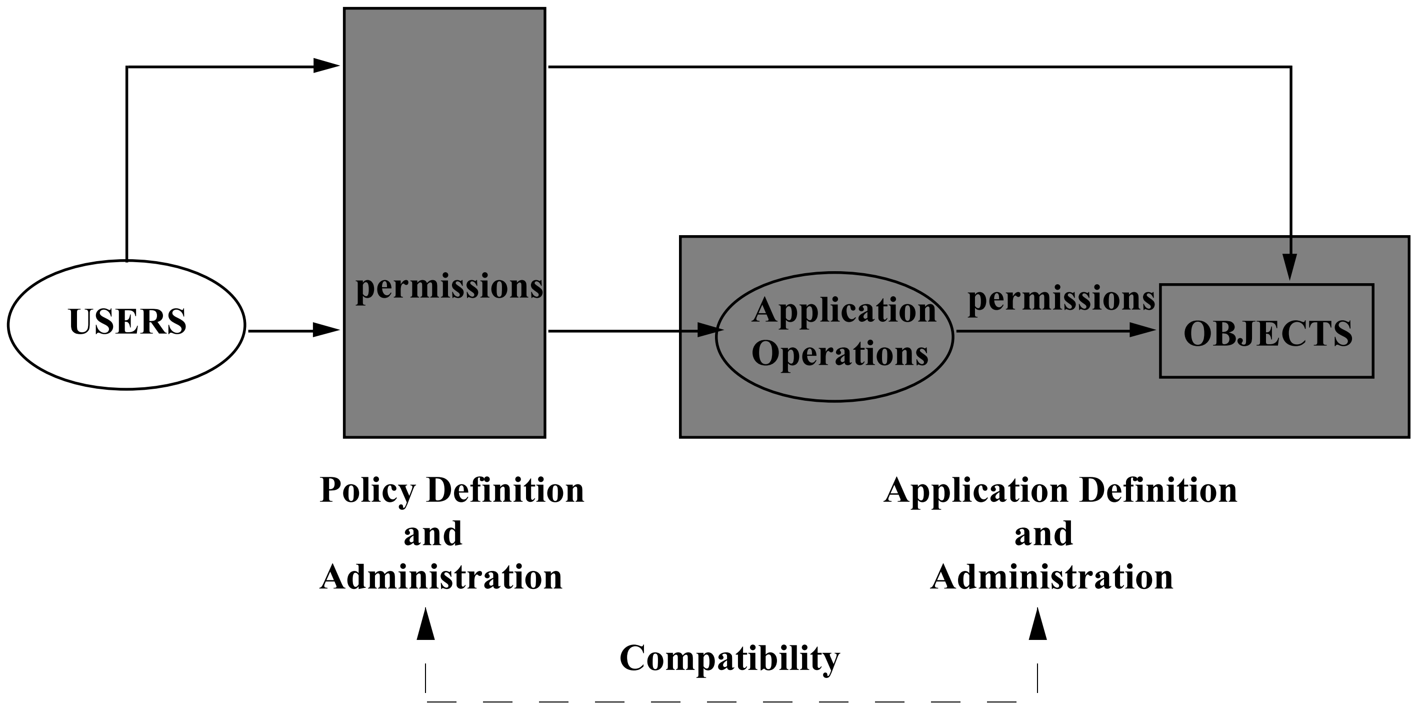
Admin(P) = "for each s in STATES, *there exists* $s_0 \in STATES_0$,
there exists $\omega \in \Omega$ such that: ω starts in s, and
ω reaches s_0 and
s_0* is in P"

Compat(P, App)

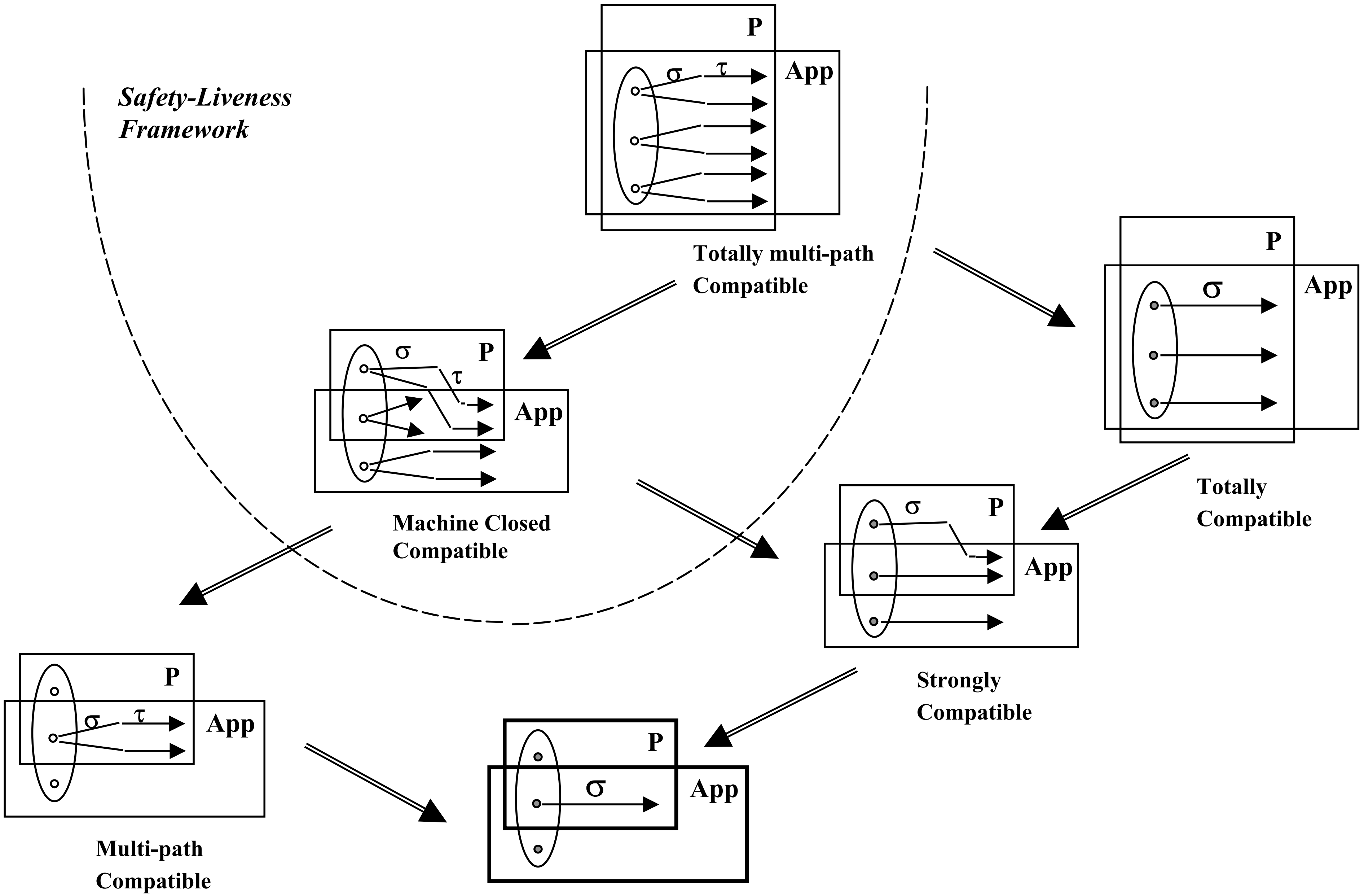
Compat(P) = "there exists $s_0 \in STATES_0$ and $\sigma \in P$ starting in s_0
such that σ executes App"

.... neither Safety nor Liveness ....

Mandated Compatibility


USERS
permissions
Application Operations
permissions
OBJECTS
Policy Definition and Administration
Application Definition and Administration
Compatibility

Types of Compatibility



Safety-Liveness Framework
P
App
σ
τ
Totally multi-path Compatible
Totally Compatible
Machine Closed Compatible
Strongly Compatible
Multi-path Compatible

Compat(P, App)

Totally Multi-path Compatible

For each start state s_0 there is a comand sequence σ in P starting in s_0 , and for each finite command sequence σ in P there is a command sequence τ such that $\sigma\tau$ is in P and executes *App*.

Machine-Closed Compatible

For each finite command sequence σ in P there is a command sequence τ such that $\sigma\tau$ is in P and executes *App*.

Multi-path Compatible

There is a start state s_0 such that for each finite command sequence σ in P starting in s_0 there is τ such that $\sigma\tau$ is in P and executes *App*.

Totally Compatible

For each start state s_0 there is a command sequence σ in P starting in s_0 such that σ executes *App*.

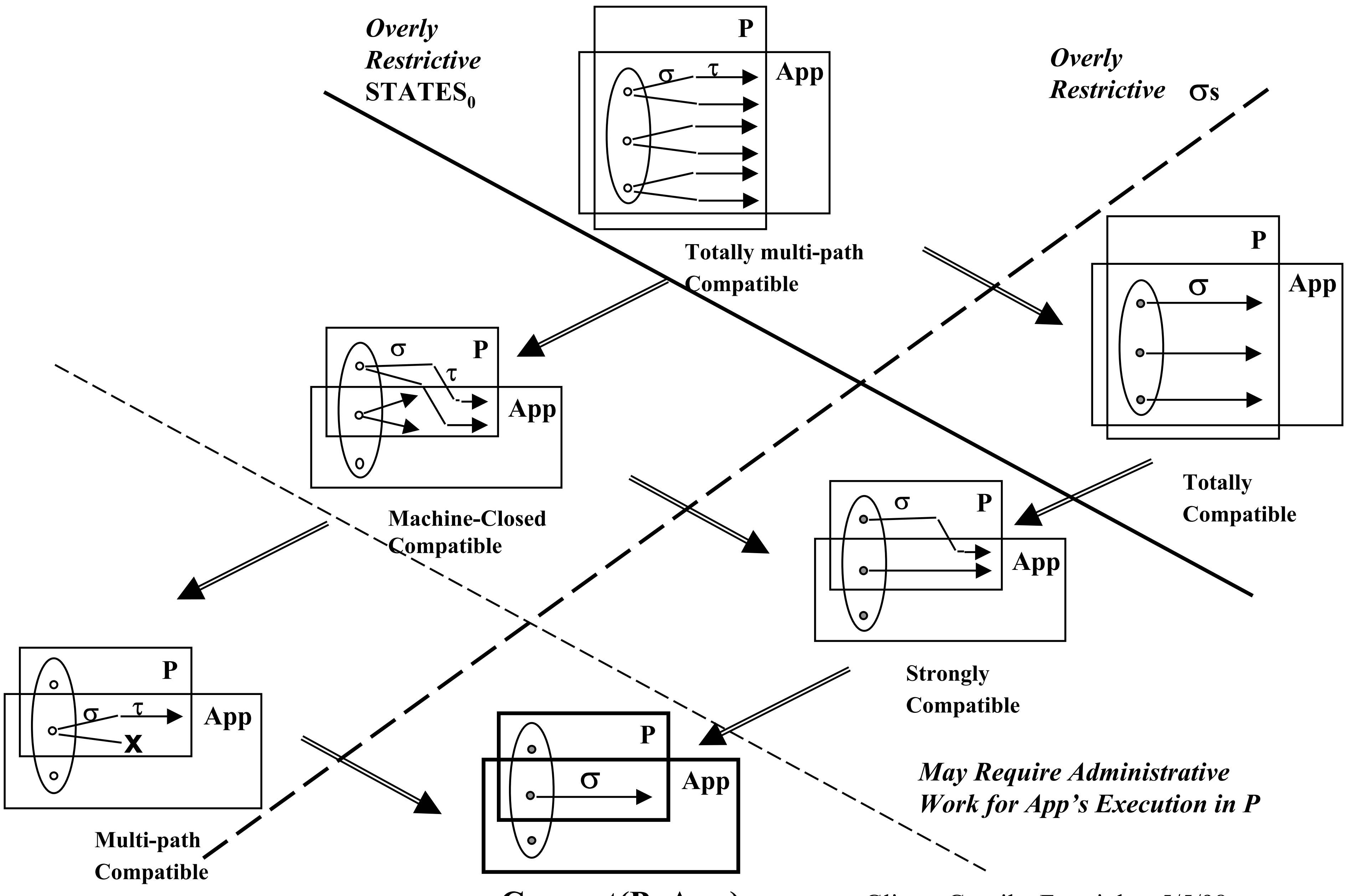
Strongly Compatible

For each start state s_0 such that s_0* is in P, there is a command sequence σ in P starting in s_0 that executes *App*.

Compatible

There is a start state s_0 and a command sequence σ in P starting in s_0 that executes *App*.

Types of Compatibility



Overly Restrictive STATES_0
P
App
σ
τ
Overly Restrictive σs
Totally multi-path Compatible
σ
τ
Machine-Closed Compatible
Totally Compatible
Strongly Compatible
x
Multi-path Compatible
May Require Administrative Work for App's Execution in P

Compat(P, App)

Overly Restrictive σ_s

Example:

App = [{*obj*}, {op_1, op_2}, *plan*]; *plan* = {(*obj*, op_1), (*obj*, op_2)}

P : "u_1 and u_2 are the only users who may execute App *and*
a user may not execute two distinct (or all) operations on the same object"

Compat(P, App) is true

$$\sigma = \begin{tikzcd} s_0 \arrow[r, "S_1:(op_1,\, obj)"] & s_1 \arrow[r, "S_2:(op_2,\, obj)"] & s_2 \end{tikzcd}$$

u_1: (op_1: *obj*), S_1 = subject

u_2: (op_1, op_2 : *obj*), S_2, S_2`= subjects

Compat_M(P, App) is false

$$\sigma` = \begin{tikzcd} s_0 \arrow[r, "S_2`:(op_1,\, obj)"] & s_1` \arrow[r, dashed, "\tau = S_2`:(op_2,\, obj)"] \arrow[dr, dashed, "S_1:(op_2,\, obj)"'] & \times \\ & & \times \end{tikzcd}$$

u_1: (op_1: *obj*), S_1

u_2: (op_1, op_2 : *obj*), S_2, S_2'

Simple Policy Composition

$$\mathbf{P}_1 = P_1 \wedge \text{Admin}(P_1) \wedge \text{Compat}(P_1, \text{App}_1)$$

$$\mathbf{P}_2 = P_2 \wedge \text{Admin}(P_2) \wedge \text{Compat}(P_2, \text{App}_2)$$

Let $CS(\mathbf{P}_i) = P_i$, if $\text{Admin}(P_i) \wedge \text{Compat}(P_i, \text{App}_i)$ is True;

ϕ, otherwise.

(Emerging policy) $\mathbf{P}_1 \circ \mathbf{P}_2 =$

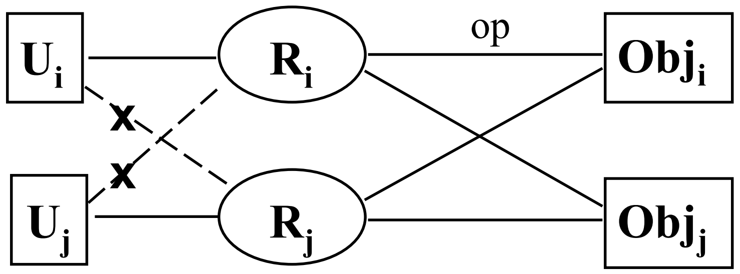
$= P_1 \wedge P_2 \wedge \text{Admin}(P_1 \wedge P_2) \wedge \text{Compat}(P_1 \wedge P_2, \text{App}_1 \cup \text{App}_2)$

$\mathbf{P}_1, \mathbf{P}_2$ are composable if and only if

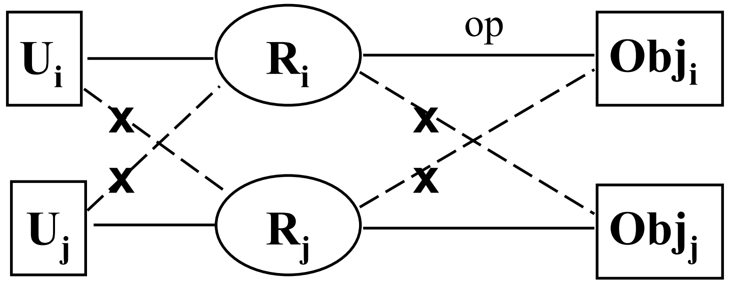
$CS(\mathbf{P}_1 \circ \mathbf{P}_2) \neq \phi$ whenever $CS(\mathbf{P}_1), CS(\mathbf{P}_2) \neq \phi$

SoD Properties (1)

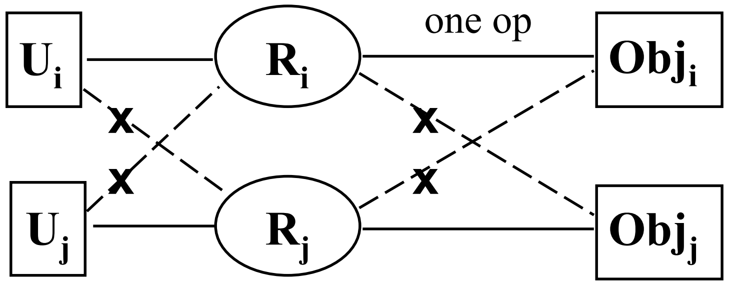
Static SoD



Strict Static SoD

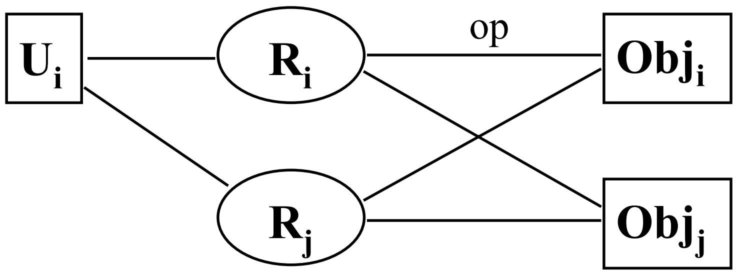


1-step Strict Static SoD

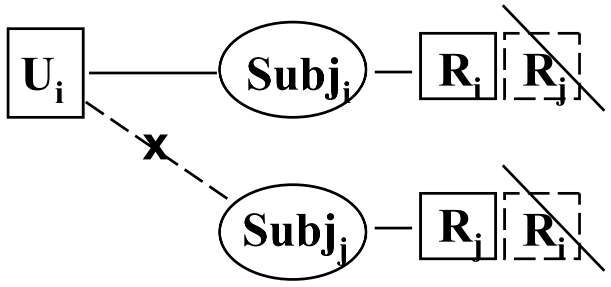


Dynamic SoD

role membership

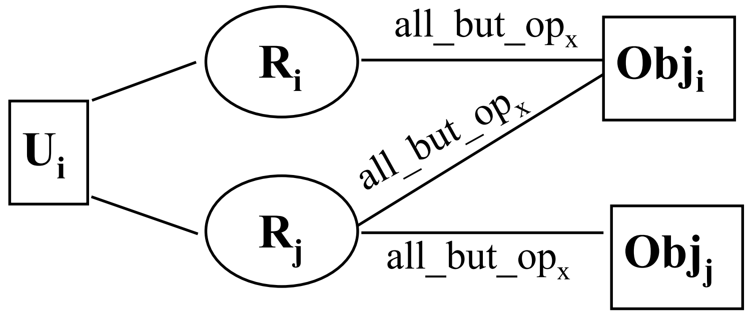


role activation

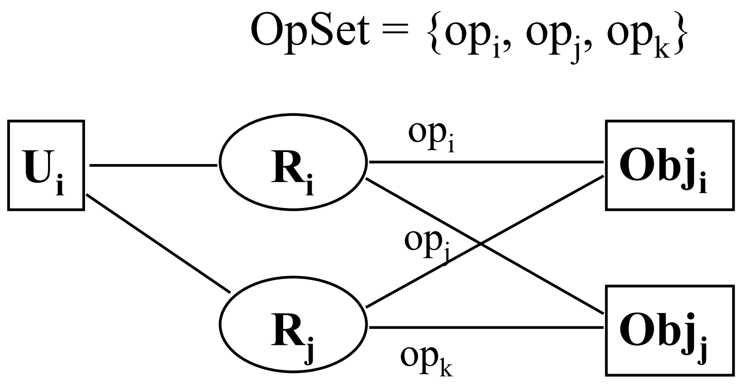


SoD Properties (2)

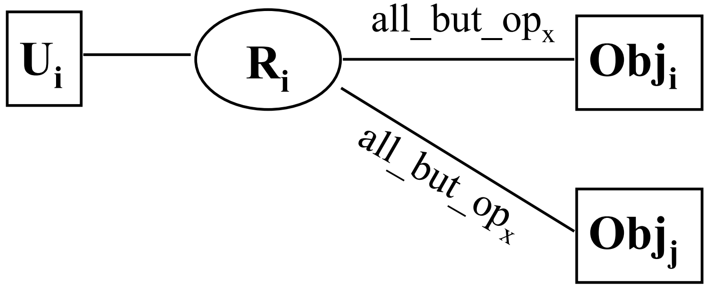
Operational Static SoD



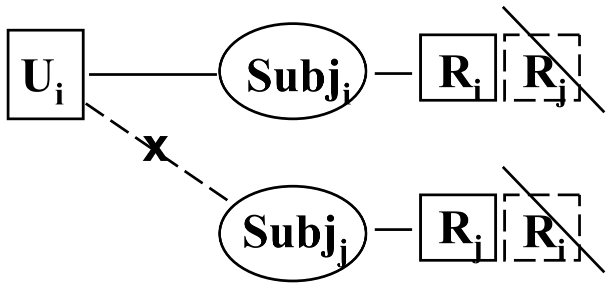
Operational Dynamic SoD



per-Role Operational Static SoD

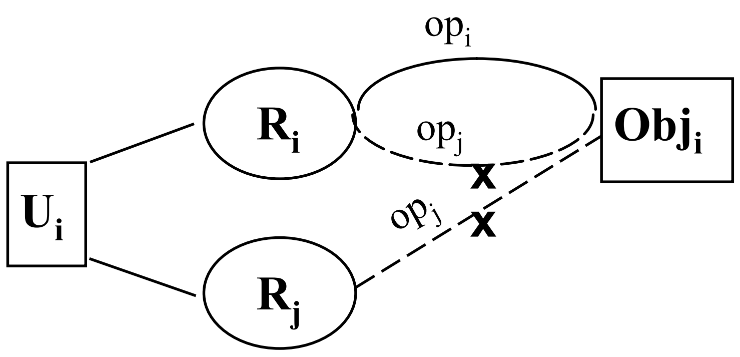


role activation

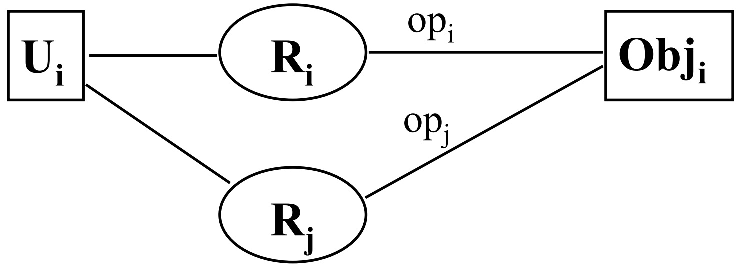


SoD Properties (3)

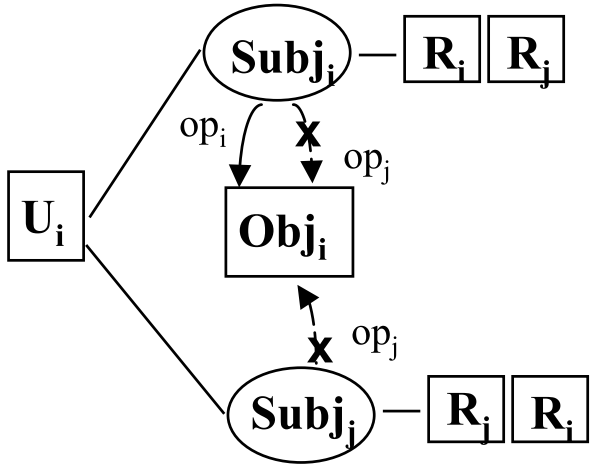
Object-based Static SoD



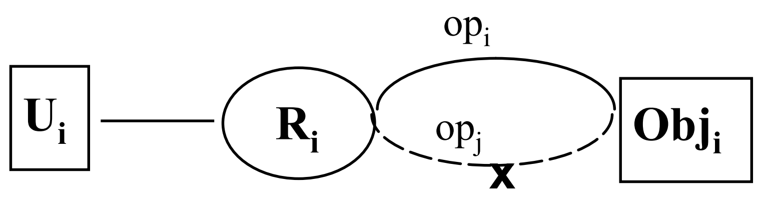
Object-based Dynamic SoD



object access



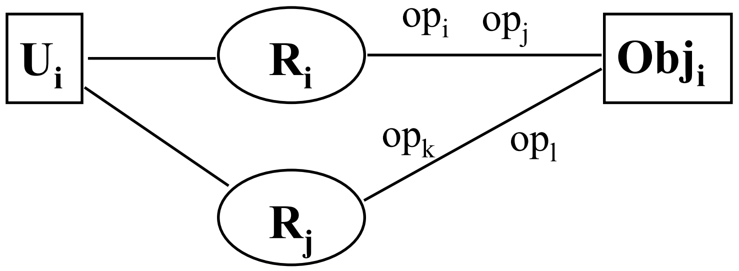
per-Role, Object-based Static SoD



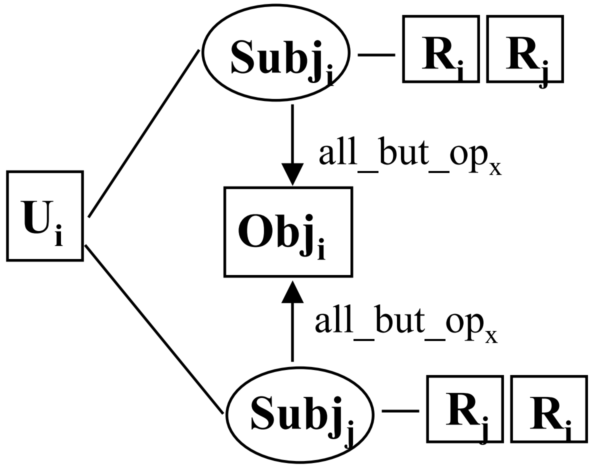
SoD Properties (4)

History-based Dynamic SoD

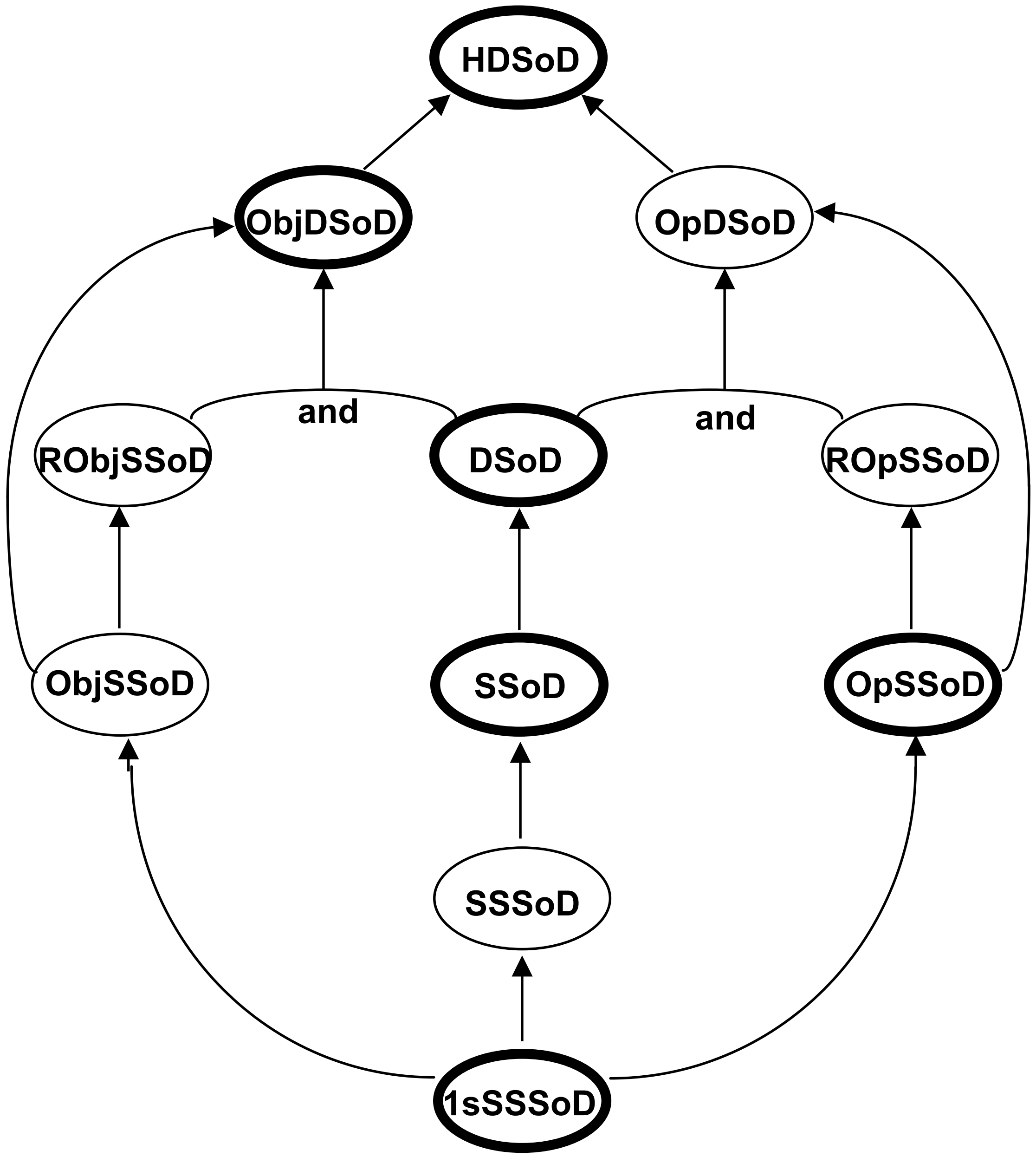
OpSet = $\{op_i, op_j, op_k, op_l\}$



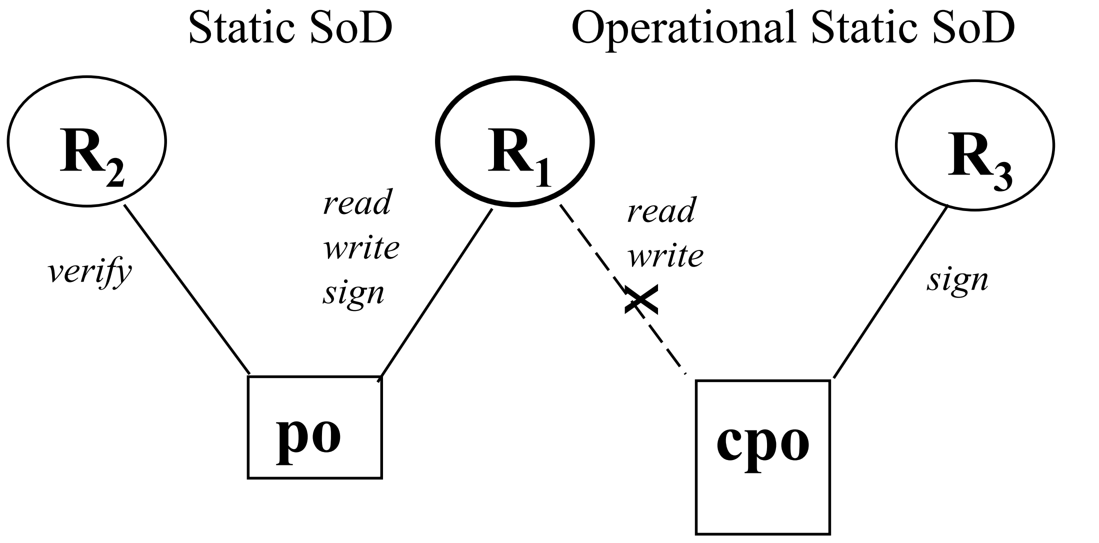
object access



Relationships among SoD Properties

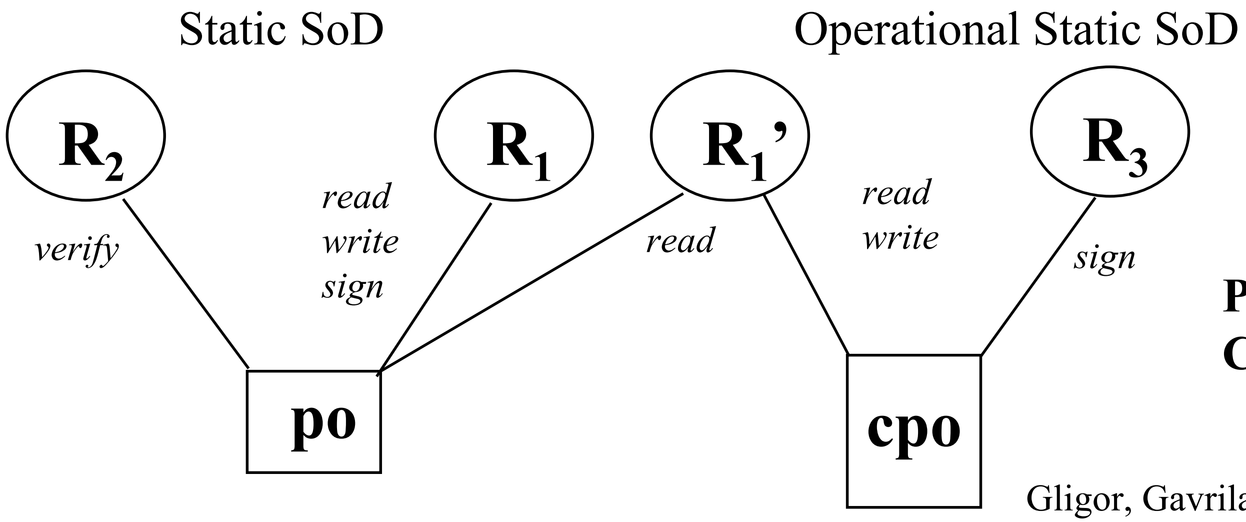


Example: Non-Composable Separation-of-Duty Policies



Purchasing Staff Department

Purchasing Staff Central Administration



Policy-Management Change