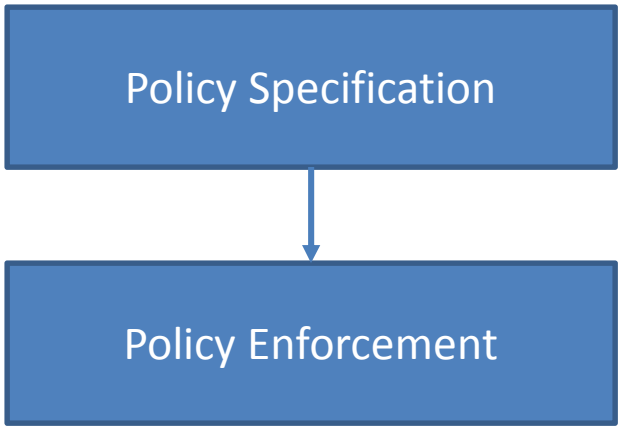
Authorization Policy Specification and Enforcement for Group-Centric Secure Information Sharing

Ram Krishnan and Ravi Sandhu

University of Texas at San Antonio

Problem

- 2 phases in access control system design



Concerned about precise conditions for authorization to hold

Concerned about how to enforce the policy

- How do we know 1 and 2 are consistent?
  - That is, that they have the same behavior wrt authorization state

Goal

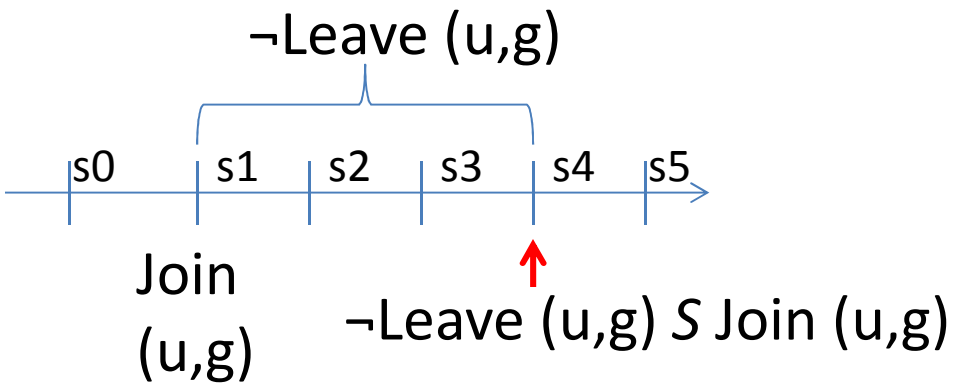
- Investigate a methodology to show consistency
- Approach
  - Pick a specific application domain
  - Design a “stateless” policy specification
  - Develop a “stateful” enforcement specification
  - Show authorization equivalence

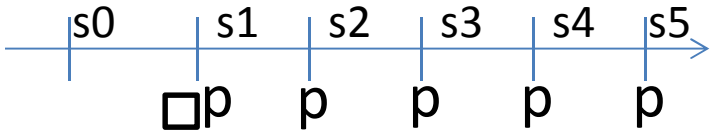
Stateless Policy Specification

- Focus on conditions under which authorization should hold
- History of actions that can lead to access
  - E.g. Alice can access an object if in the past she had paid membership dues up to current time
- Linear Temporal Logic (LTL)
  - Excellent fit to specify stateless policies

First-Order Linear Temporal Logic

- Extends familiar first-order logic with temporal operators
- Consider “Henceforth” and “Since” operators

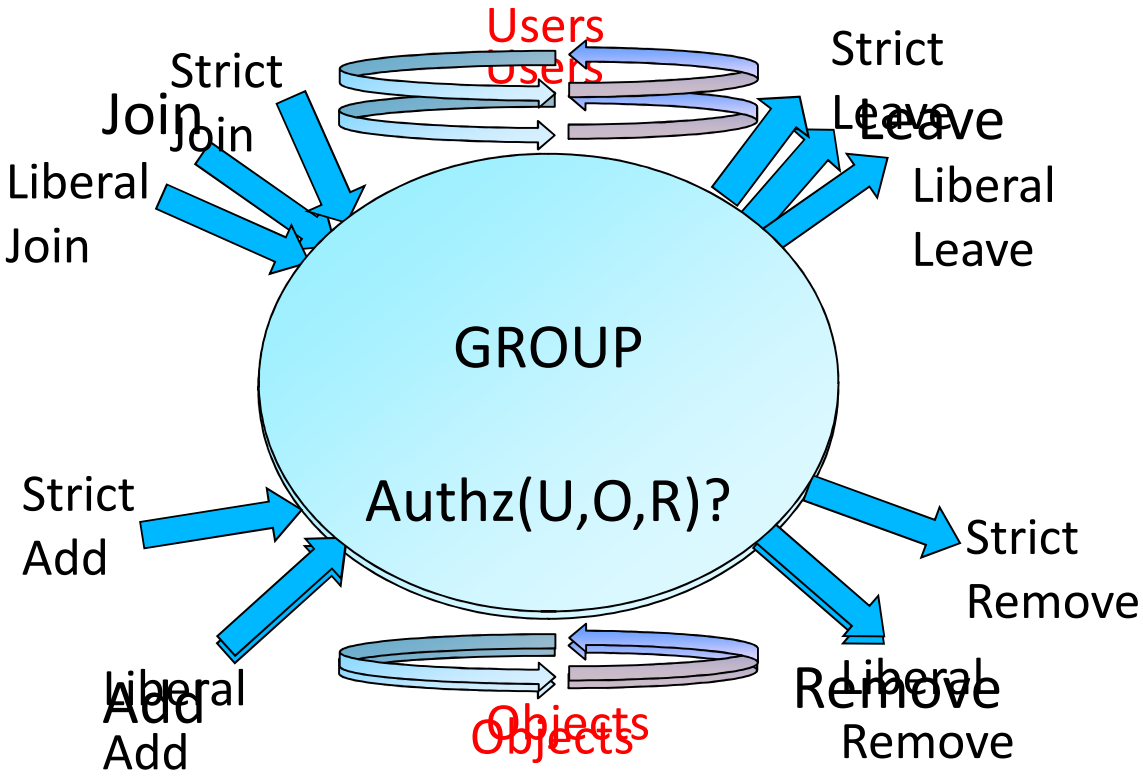




Application Domain

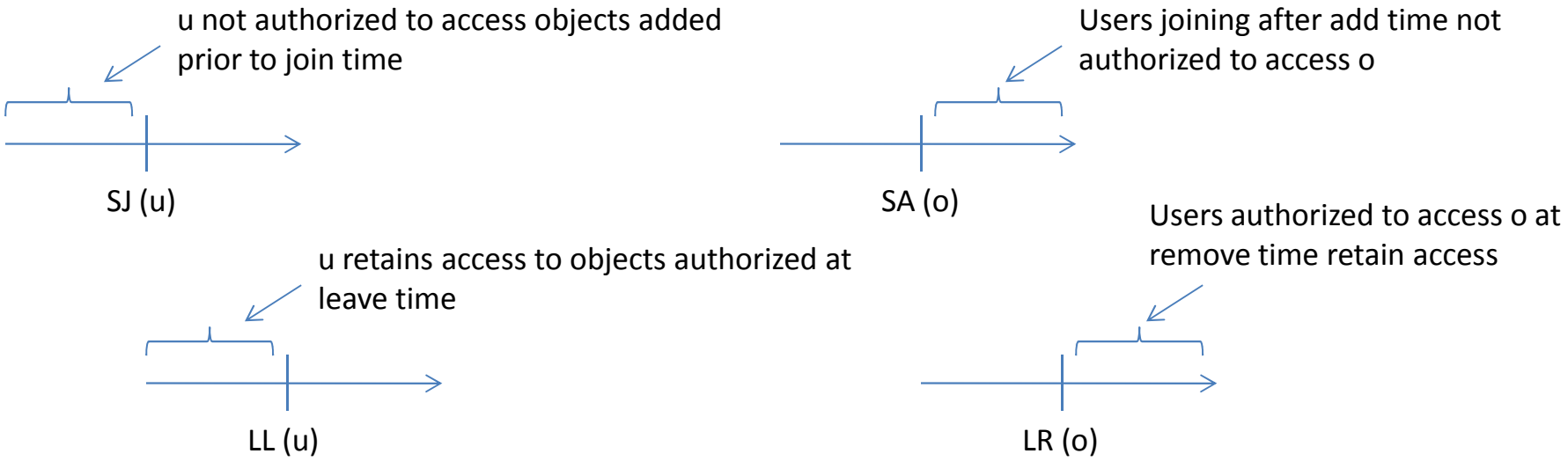
Group-centric
secure information sharing (g-SIS)

Group-centric Secure Information Sharing (g-SIS)



Group Operation Semantics

- Strict Vs Liberal operations
  - User operations (SJ, LJ, SL, LL)
  - Object operations (SA, LA, SR, LR)



The π-System g-SIS Specification (contd)

Well-formed traces

π-system g-SIS Specification:

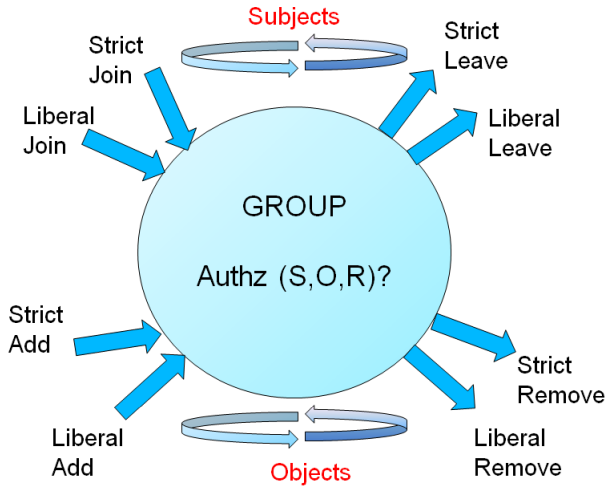
$$\pi = \Box(\text{Authz} \leftrightarrow \lambda_1 \vee \lambda_2) \wedge \bigwedge_{0 \leq j \leq 3} \tau_j$$

$$\lambda_1 = ((\neg \text{SL} \wedge \neg \text{SR})\ \mathcal{S}\ ((\text{SA} \vee \text{LA}) \wedge ((\neg \text{LL} \wedge \neg \text{SL})\ \mathcal{S}\ (\text{SJ} \vee \text{LJ}))))$$

Add after Join

$$\lambda_2 = ((\neg \text{SL} \wedge \neg \text{SR})\ \mathcal{S}\ (\text{LJ} \wedge ((\neg \text{SR} \wedge \neg \text{LR})\ \mathcal{S}\ \text{LA})))$$

Add before Join



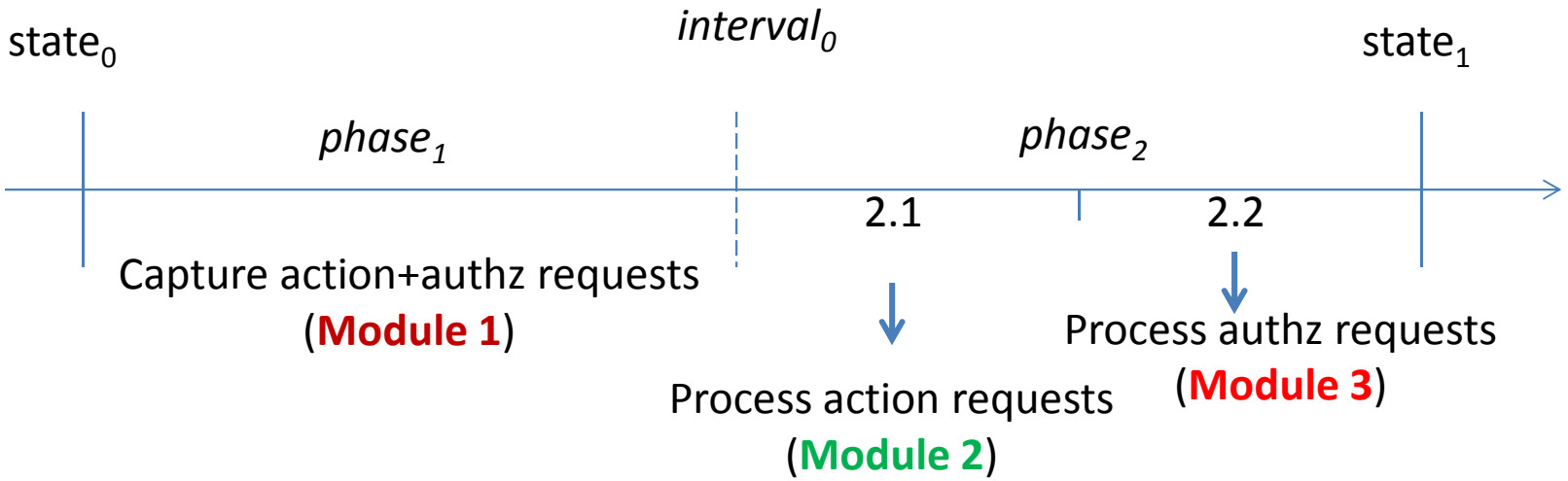
Well-formedness constraints rule out invalid traces

Examples: 1. user joining and leaving in the same state

2. leaving before joining

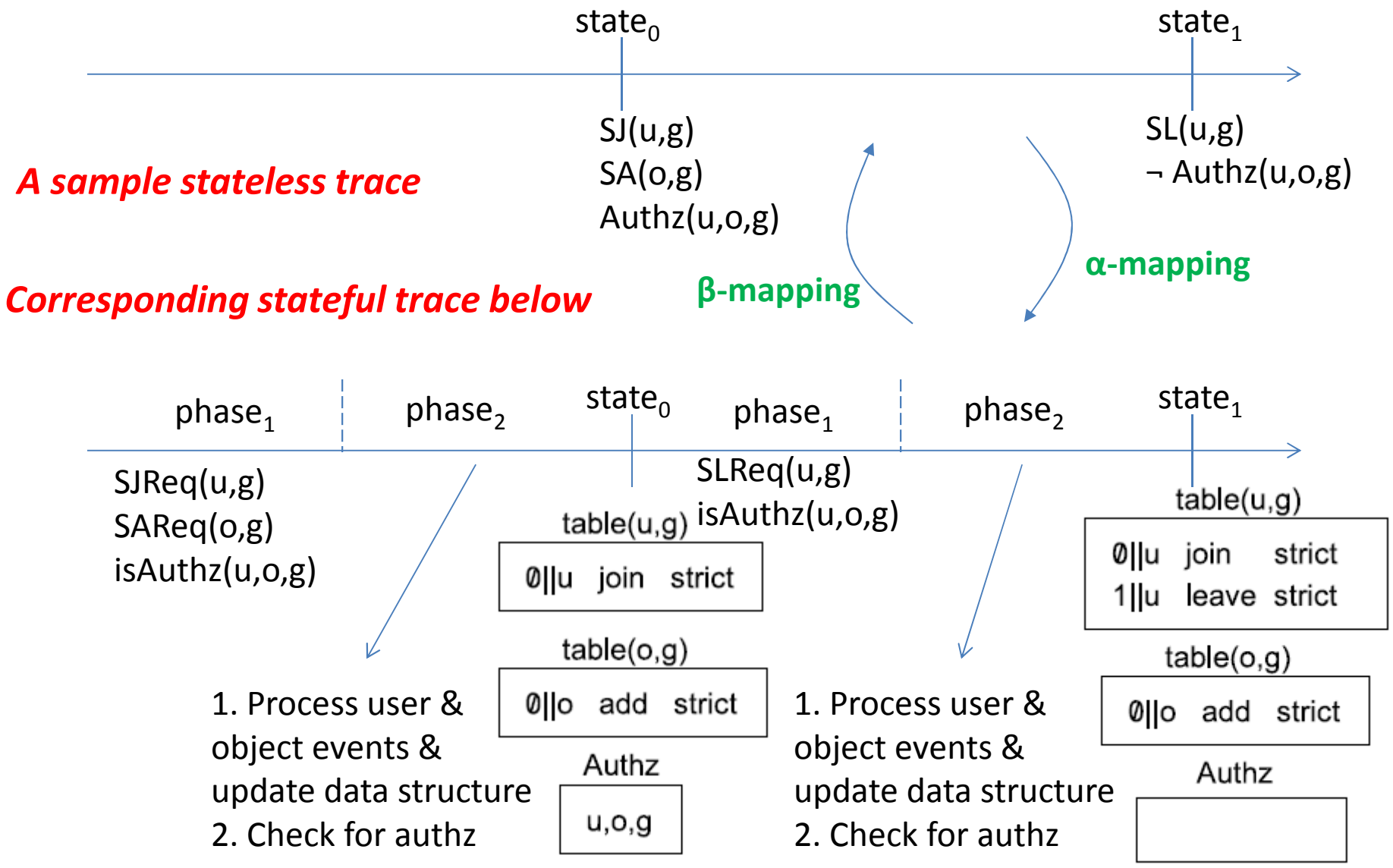
Stateful Specification

- Consists of three modules



- Module 2 maintains and manages data structures
  - ➢ Keeps track of historical joins and leaves and adds and removes for users and objects
- Module 3 consults with that data structure

Mapping Stateless and Stateful



$\text{Authz}_{\text{stateful}} \leftrightarrow \text{Authz}_{\text{stateless}}$

- Lemma 3
  - *for every trace in stateless, an alpha-mapped action trace exists in stateful*
- Lemma 4
  - *for every trace in stateful, a beta-mapped action trace exists in stateless*
- Lemmas 3 and 4 together prove authorization equivalence

Future Work

- We assumed a centralized, ideal situation
- Distributed enforcement model (PIPs, PDPs, PEPs, etc.)
  - Next step towards real policy enforcement
  - Proving equivalence is very difficult
    - E.g. staleness of authorization info due to network delay
- Stale-safe equivalence with respect to centralized specification feasible
  - E.g. in the presence of stale information, a user should be authorized for an action only if it was authorized using accurate information in the recent past
- Currently investigating for g-SIS application