Access Control Policy Mining:
RBAC to ABAC

L10-2

CS6393

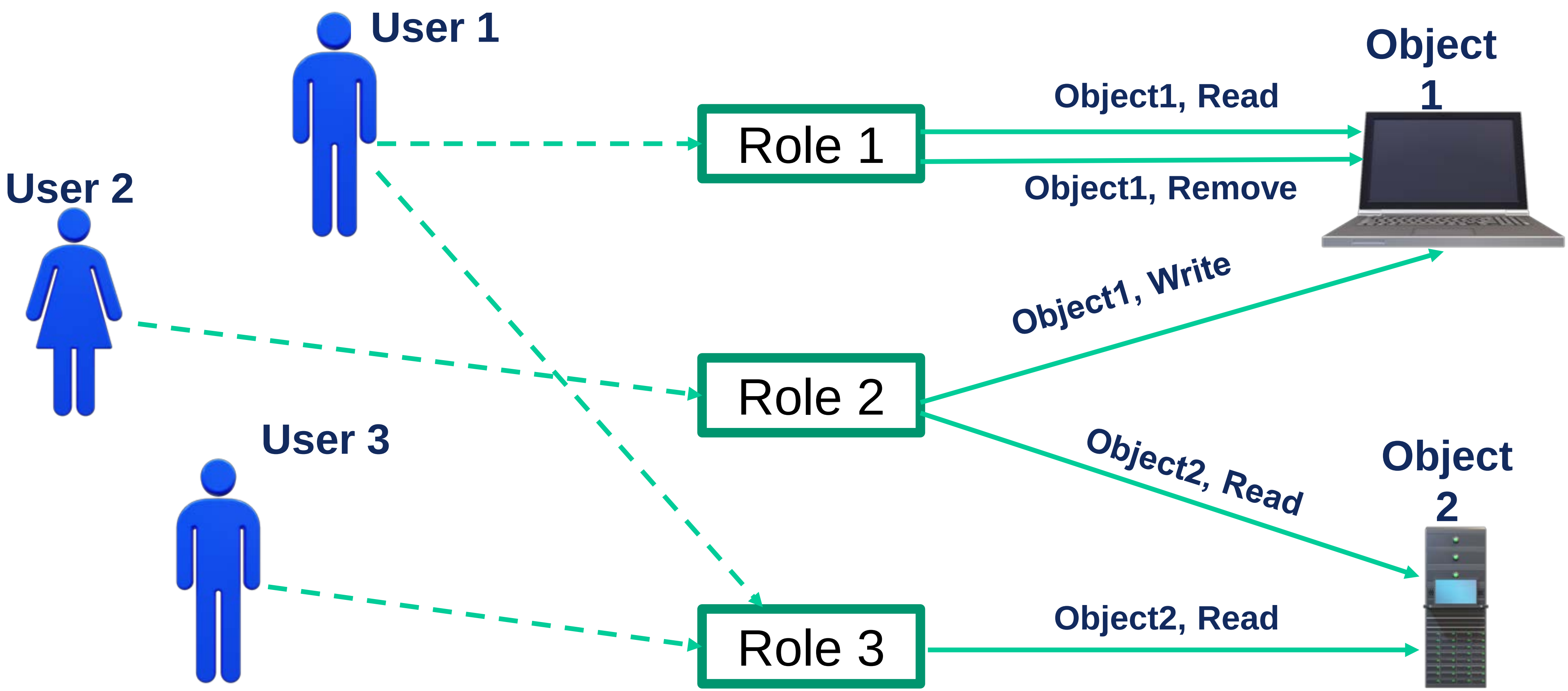
Spring 2020

Paper 2

2. Shuvra Chakraborty, Ravi Sandhu and Ram Krishnan, *On the Feasibility of RBAC to ABAC Policy Mining: A Formal Analysis.* In Proceedings of the 7th International Conference on Secure Knowledge Management in Artificial Intelligence Era (SKM), Goa, India, December 21-22, 2019, 17 pages.

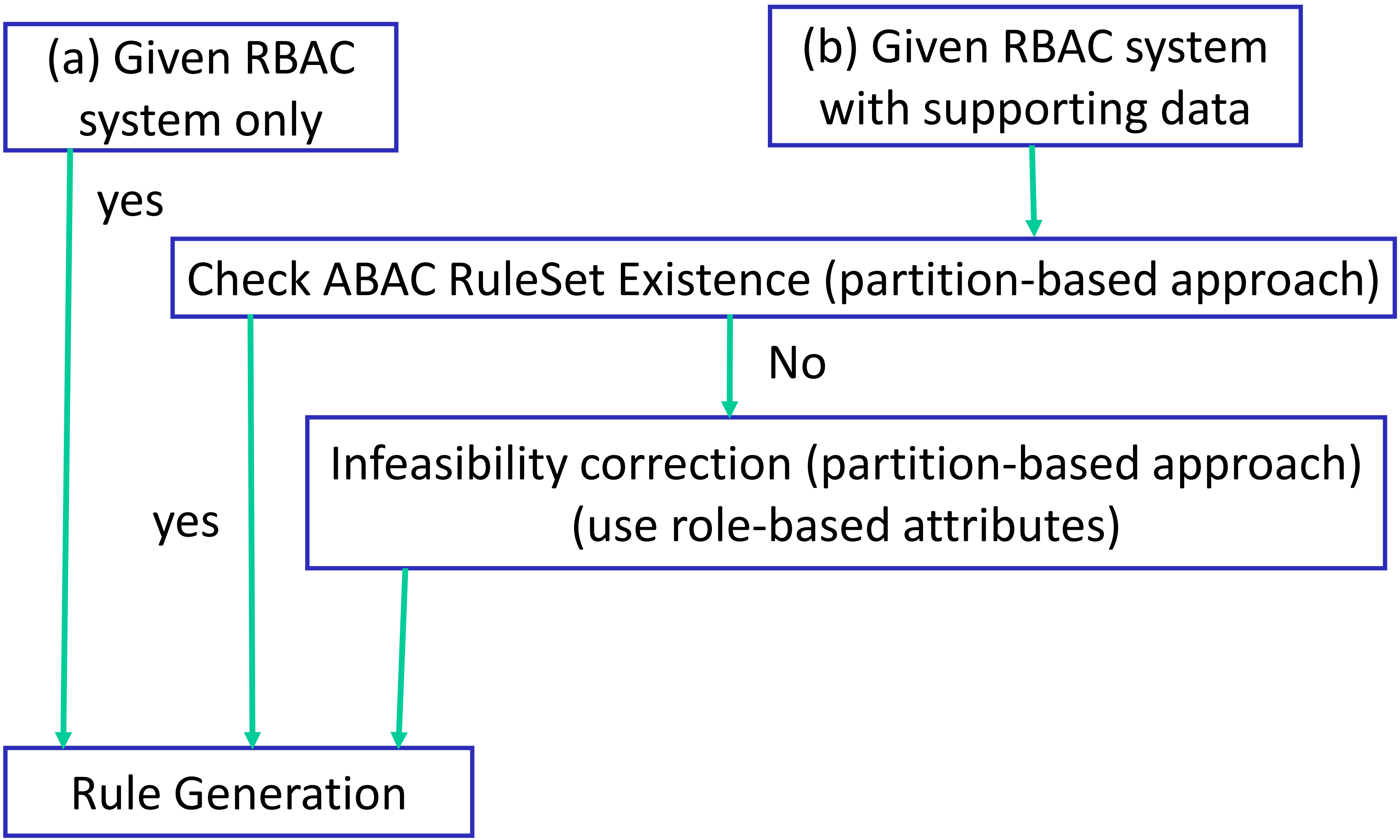
**** Role Mining and ABAC policy mining references can be found in "Related works" section of paper 2

Review: RBAC



Role-Based Access Control (RBAC) assigns user to "role"

Paper 2 Workflow



Problem Motivation

- RBAC (Role-Based Access Control) is widely used but has notable limitations (e.g., role explosion)
- Using ABAC (Attribute-Based Access Control), access control policies can be written in more flexible and higher level way
- Automated migration of an existing RBAC system to ABAC system (defined as ABAC policy mining problem) cuts the cost and human efforts needed
- Stoller et. al. use explicit unique IDs in attribute set to resolve ABAC policy mining problem which is somehow conflicting with basic principle of ABAC

❖ *We introduce ABAC RuleSet Existence problem: questions the feasibility of ABAC policy mining problem in RBAC context*
  - ❖ *If not feasible without ID, infeasibility correction technique is applied*
  - ❖ *Eliminates use of explicit ID in ABAC policy mining*

Review terminologies…

1. Access control

2. An Access control system must have a checkAccess function which evaluates an access request (user, object, operation) to true/false

3. Two access control systems are equivalent iff i) set of users (U), objects (O), and operations (OP) are identical ii) for any access request, $\text{checkAccess}_{\text{system1}}$ and $\text{checkAccess}_{\text{system2}}$ evaluates the same

4. The following example includes 3 types of Access Control System
   a. Enumerated Authorization System (EAS)
   b. RBAC System
   c. ABAC System

Review: EAS

EAS is a tuple <U, O, OP, AUTH, $checkAccess_{EAS}$>

- ❖ U, O, and OP are finite sets of users, objects and operations, respectively
- ❖ AUTH $\subseteq$ U x O x OP

Example 1:

- ❖ U = {John, Lina, Ray, Tom}, OP = {read, write}, O = {Obj1, Obj2}

AUTH	Explanation
(John, Obj1, write) (John, Obj2, write) (John, Obj1, read) (Lina, Obj2, write) (Tom, Obj1, read) (Ray, Obj1, read)	e.g., John is allowed to do read operation on Obj1 but not allowed to do read operation on Obj2

Review: RBAC

RBAC system

- is a tuple <U, O, OP, Roles, RPA, RUA, RH, $checkAccess_{RBAC}$>

- Roles is finite set of roles
- RH is the role hierarchy relation [RH': reflexive transitive closure of RH]
- RPA : Role Permission Assignment
- RUA: Role User Assignment
- Permission is an object-operation pair
- $authPerm(r) = \{p \in RPA(r') | (r, r') \in RH'\}$, where $r, r' \in Roles$
- $authUser(r) = \{u \in RUA(r') | (r', r) \in RH'\}$ where $r, r' \in Roles$

- $checkAccess_{RBAC}(u{:}U, o{:}O, op{:}OP) \equiv \exists r \in Roles.(u \in authUser(r) \wedge p \in authPerm(r) \wedge (o, op) = (obj(p), ops(p)))$

Review: RBAC

Example 2:

- U = {John, Lina, Ray, Tom}, OP = {read, write}, O = {Obj1, Obj2} [same as Example 1]
- Roles = {R1, R2, R3}
- RPA(R1) = {(Obj1, write)}, RPA(R2) = {(Obj2, write)}, RPA(R3) = {(Obj1, read)}
- RUA(R1) = {John}, RUA(R2) = {Lina}, RPA(R3) = {Ray, Tom}
- RH={(R1,R2), (R1, R3)}   [R1 is a senior role than R2, R3]

EAS and RBAC system defined in example 1 and 2 are equivalent

Review: ABAC

- **ABAC system** is a tuple <U, O, OP, UA, OA, UAValue, OAValue, RangeSet, RuleSet, checkAccess$_{ABAC}$ >

Example 3

- U, O, OP are same as Example 1
- UA ={Position, Dept.}, OA = {Type}

UAValue		
User (U)	**Position**	**Dept.**
John	Officer	CS
Lina	Student	CS
Ray	Officer	CS
Tom	Officer	CS

RangeSet	
Position	{Officer, Student, Faculty}
Dept.	{CS, EE}
Type	{File, Printer, Scanner}

OAValue	
Object (O)	**Type**
Obj1	File
Obj2	Printer

- RuleSet contains one separate rule for each operation, {Rule$_{read}$, Rule$_{write}$}
- ***ABAC system is incomplete in Example 3 (No rules given!)***

Review: ABAC

ABAC rule structure

For any operation op ∈ OP, $Rule_{op}$ grammar

- $Rule_{op}$ ::= $Rule_{op} \vee Rule_{op}$ | (Atomicexp)
- Atomicexp ::= Atomicuexp ∧ Atomicoexp | Atomicuexp | Atomicoexp
- Atomicuexp ::= Atomicuexp ∧ Atomicuexp | uexp
- Atomicoexp ::= Atomicoexp ∧ Atomicoexp | oexp
- uexp ∈ {ua(u) = value | ua ∈ UA ∧ value ∈ Range(ua)}
- oexp ∈ {oa(o) = value | oa ∈ OA ∧ value ∈ Range(oa)}

- **$checkAccess_{ABAC}$ (a:U, b:O, op:OP) ≡ $Rule_{op}$(a:U, b:O)**

*** Illustrated ABAC rule examples can be found in later slides

*** Example 1,2 and 3 will be used to demonstrate the workflow of paper 2

ABAC RuleSet Existence

Role Based Access Control System (Example 2)

Supporting Data (Incomplete ABAC system without RuleSet in Example 3)

Equivalent ABAC system

Does an equivalent ABAC system exist for the given RBAC system and supporting data?
Find the RuleSet -> *With ID, always possible, *No IDs → Not possible
e.g., cannot separate John from Ray and Tom in Example 3

(a) RBAC only

Step 1. Generate role-based attribute set

- For a user u, role-based user attribute denotes the set of roles possessed by u
- For an object-operation pair (obj, op), role-based object attribute denotes the set of roles where each role contains permission (obj, op)

Role-based user attribute (Example 2)	
User(U)	**uroleAtt**
John	{R1, R2, R3}
Lina	{R2}
Ray	{R3}
Tom	{R3}

Role-based object attribute (Example 2)		
Object(O)	**$oroleAtt_{write}$**	**$oroleAtt_{read}$**
Obj1	{R1}	{R1, R3}
Obj2	{R1, R2}	{}

Next step: partition set is generated on set UxO based on similarity in attribute value assignment

Step 2

Partition set w.r.t. write

Ray, Obj1
Tom, Obj1

Ray, Obj2
Tom, Obj2

John, Obj2

Lina, Obj1

John, Obj1

Lina, Obj2

Partition set w.r.t. read

Ray, Obj1
Tom, Obj1

Ray, Obj2
Tom, Obj2

John, Obj2

Lina, Obj1

John, Obj1

Lina, Obj2

Partition set is conflict-free w.r.t. read and write → YES

Step 3

•*Given an operation op, if partition set is conflict-free and each partition is uniquely identified by the set of (attribute name, value) pair then RuleSet can be generated [Proved]*

•A conjunction of (attribute name, value) pair is made for each conflict-free bold black partition and OR'ed to $Rule_{op}$

e.g., $Rule_{read} \equiv$ <**(uroleAtt(u) = {R3} Λ oroleAtt$_{write}$ (o)={R1} Λ oroleAtt$_{read}$ (o) = {R1, R3}) V (uroleAtt(u) = {R1, R2, R3} Λ oroleAtt$_{write}$(o)= {R1} Λ oroleAtt$_{read}$ (o)= {R1, R3})**>

*****$Rule_{write}$ can be constructed same way**
***RuleSet = {$Rule_{write}$, $Rule_{read}$}**
Example 2 and completed ABAC system in example 3 are equivalent
*****Equivalent ABAC system generation is always possible!**

(b) With supporting data

Role Based Access Control System (Ex. 2)

Equivalent ABAC system

Supporting Data (Ex. 3)

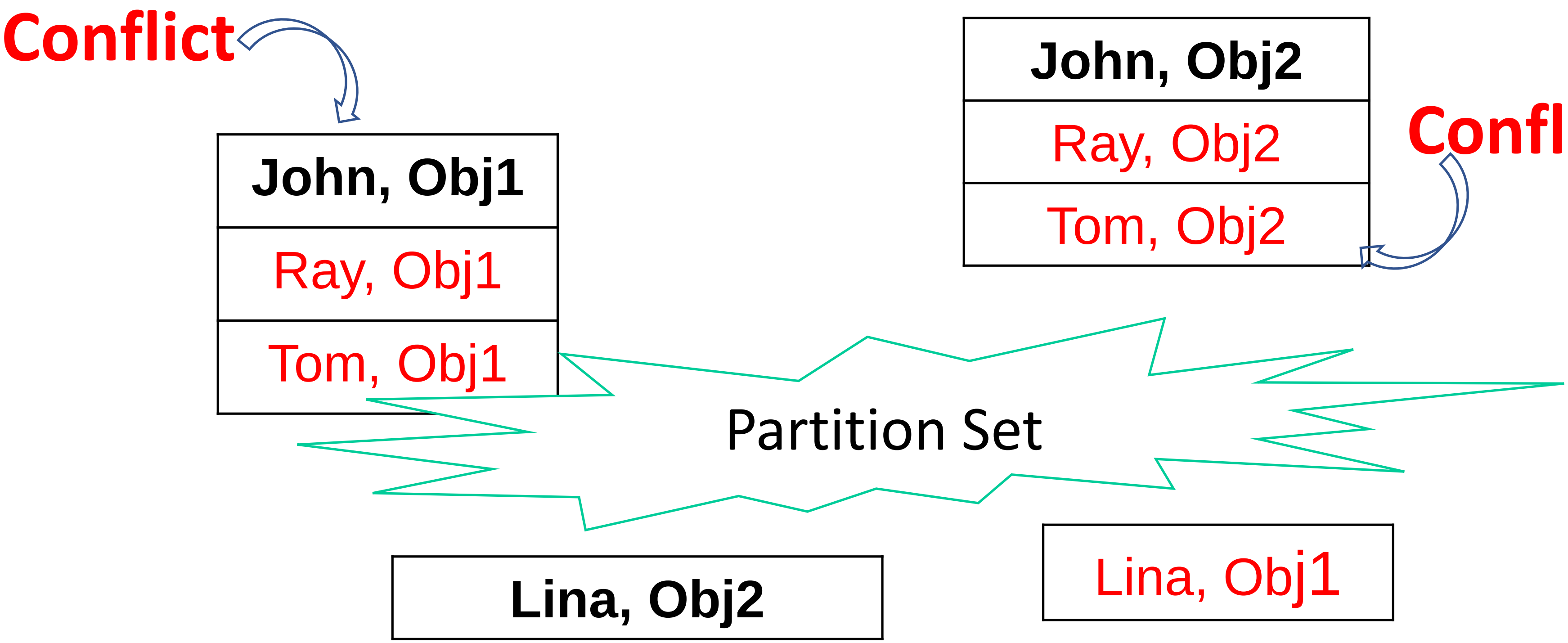
UAValue		
User (U)	**Position**	**Dept.**
John	Officer	CS
Lina	Student	CS
Ray	Officer	CS
Tom	Officer	CS

OAValue	
Object (O)	**Type**
Obj1	File
Obj2	Printe r

RangeSet	
Position	{Officer, Student, Faculty}
Dept.	{CS, EE}
Type	{File, Printer, Scanner}

Step 1: Generate partition set based on similarity in attribute value assignment. Partition set might have conflicts!

Step 1



*Partition set has conflict w.r.t. write → YES (Ex. 2 and 3)

Next step: Apply infeasibility correction

Step 2 and 3

Partition

conflict →

Partition the sets of users and objects present

e.g., {John, Ray, Tom} is partitioned as {John} and {Ray, Tom}

conflict-free, (UA, OA)

Generate a conjunctive clause

<u>if needed</u>

a. Add role-based user attribute to UA

b. Add role-based obj attributes to OA

conflict-free

(UA U uroleAtt),

(OA U oroleAtt$_{op}$, op∈OP)

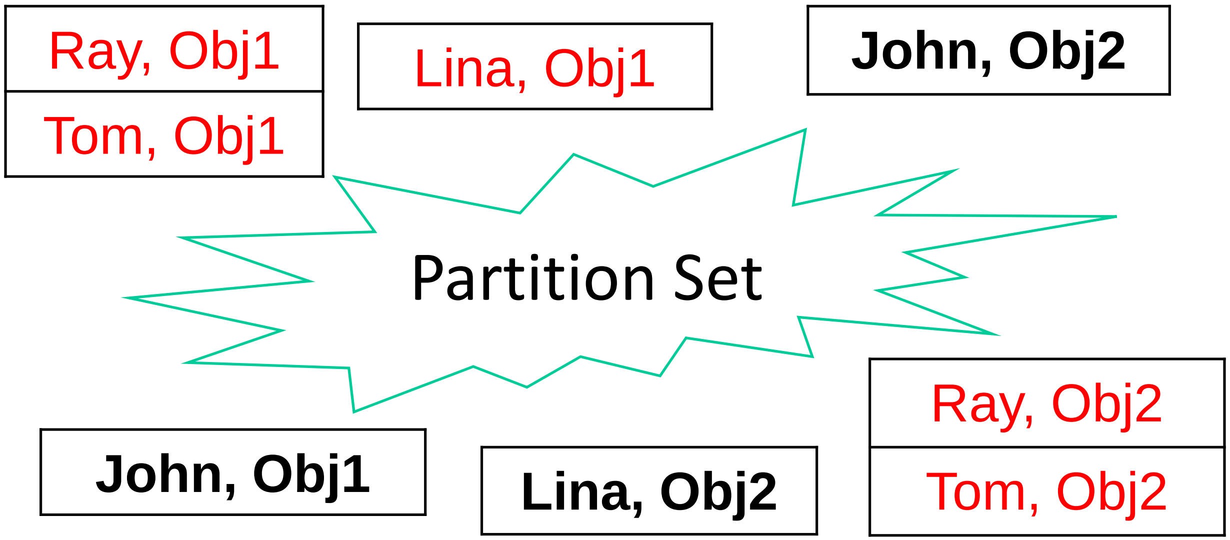
Use role-based attribute values to identify

e.g., {John} and {Ray, Tom} are assigned different role-based attribute values

OR to Rule$_{op}$

Infeasibility correction: exact solution can be achieved many ways

Partition set: corrected



UAValue	
User(U)	uroleAtt
John	{R1, R2, R3}
Lina	{R2}
Ray	{R3}
Tom	{R3}

OAValue		
Object (O)	$oroleAtt_{write}$	$oroleAtt_{read}$
Obj1	{R1}	{R1, R3}
Obj2	{R1, R2}	{}

$Rule_{write}$ ≡ <(Position(u) = officer ∧ Dept(u) = CS ∧ uroleAtt(u)={R1, R2, R3} ∧ Type(o) = File) ∨ (Position(u) = officer ∧ Dept(u) = CS ∧ uroleAtt(u)={R1, R2, R3} ∧ Type(o) = Printer) ∨ (Position(u) = student ∧ Dept(u) = CS ∧ Type(o) = Printer)>

***RuleSet = {$Rule_{write}$, $Rule_{read}$}**

Paper 2: Summary

- Formalized notion of feasibility on RBAC to ABAC policy mining: first time
- The overall asymptotic complexity of ABAC RuleSet Existence problem is $O(|OP| \times (|U| \times |O|))$
- The overall asymptotic complexity of ABAC RuleSet Infeasibility Correction in RBAC context is $O(|OP| \times (|U| \times |O|)^3)$

Challenges

- Can you ensure partition split always equals 2?
- More compact set of rule generation
- Negative rules?
- Exact solution:
  - ❖Reduce number of split partitions
  - ❖Change number of attributes required
  - ❖Changing existing attribute set, possible?