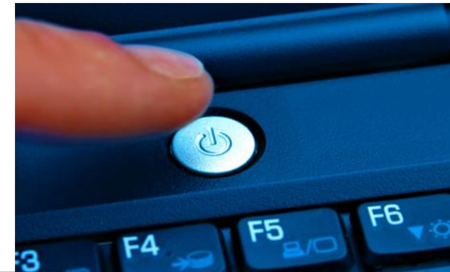
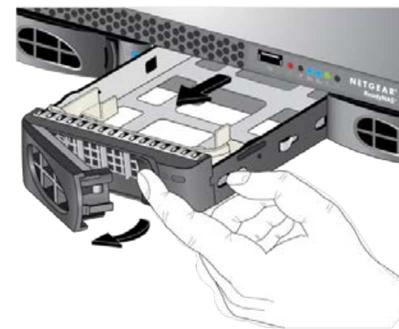
Role and Attribute Based Collaborative Administration of Intra-Tenant Cloud IaaS *(Invited Paper)*

Xin Jin, Ram Krishnan and Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio

October 22–25, 2014
10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing

IT Infrastructure Operations





Access Control

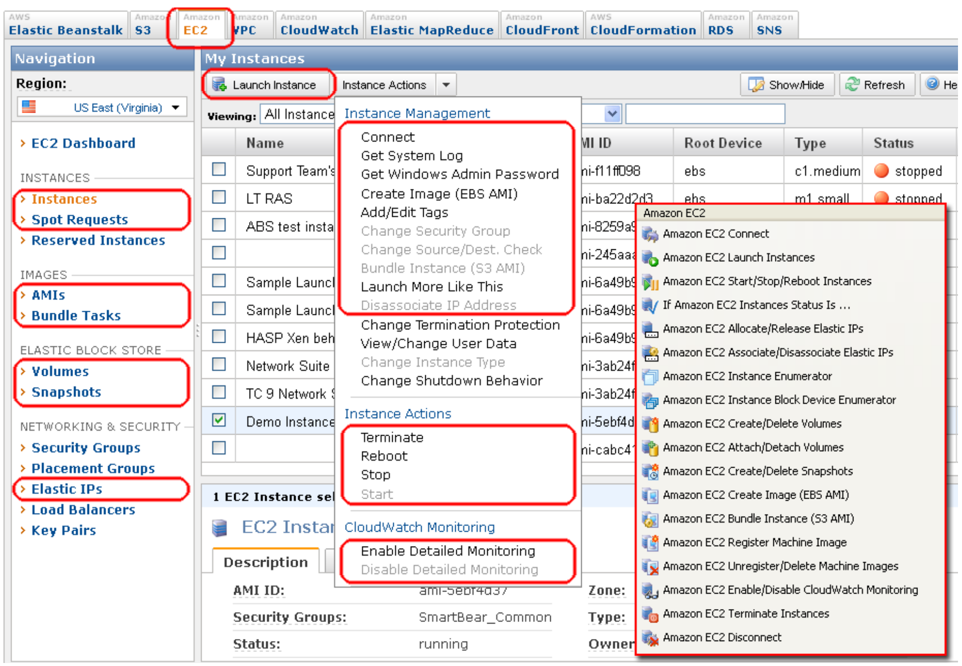
Cloud Service Models

Software as a Service (SaaS)	*Network accessible software*
Platform as a Service (PaaS)	*App dev environment with cloud characteristics*
Infrastructure as a Service (IaaS)	*Virtualized hardware infrastructure*

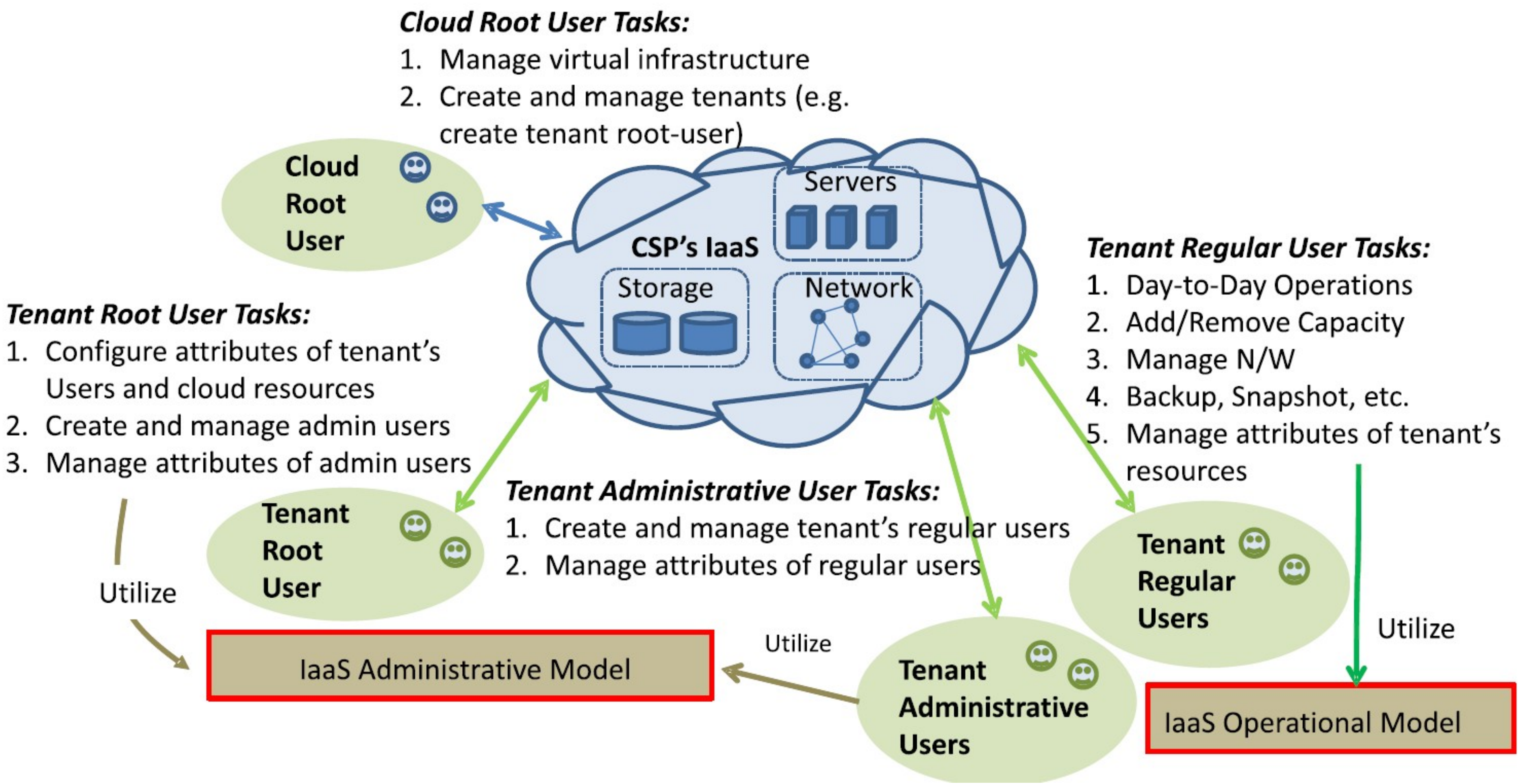
"Moving" to Cloud

Equivalent policies should be configurable using cloud access control service

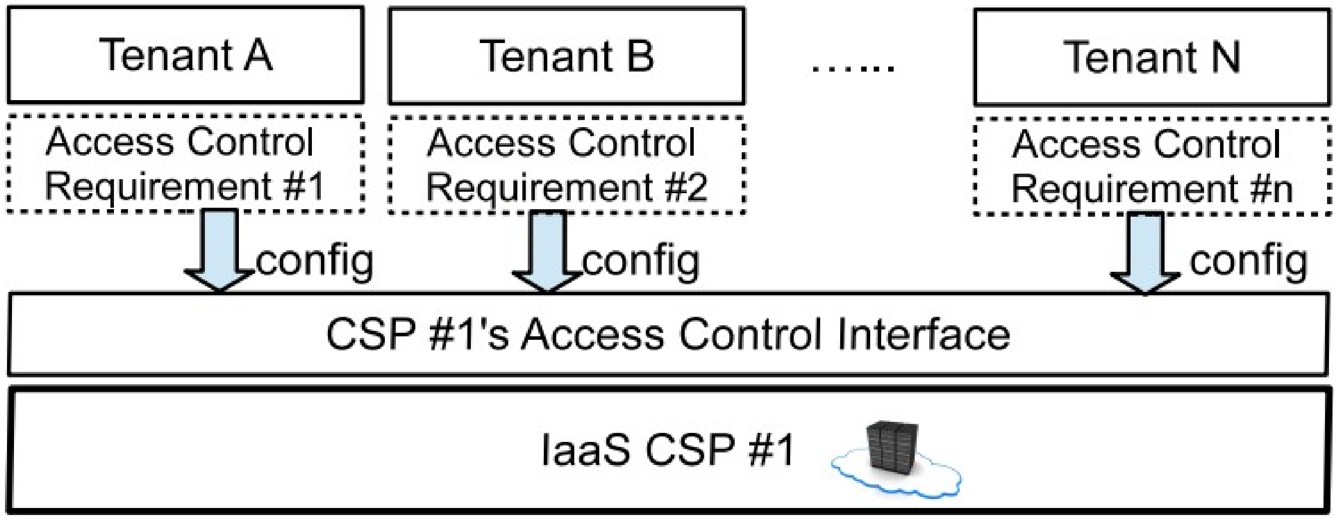
With virtualization, cloud may provide more fine-grained access control



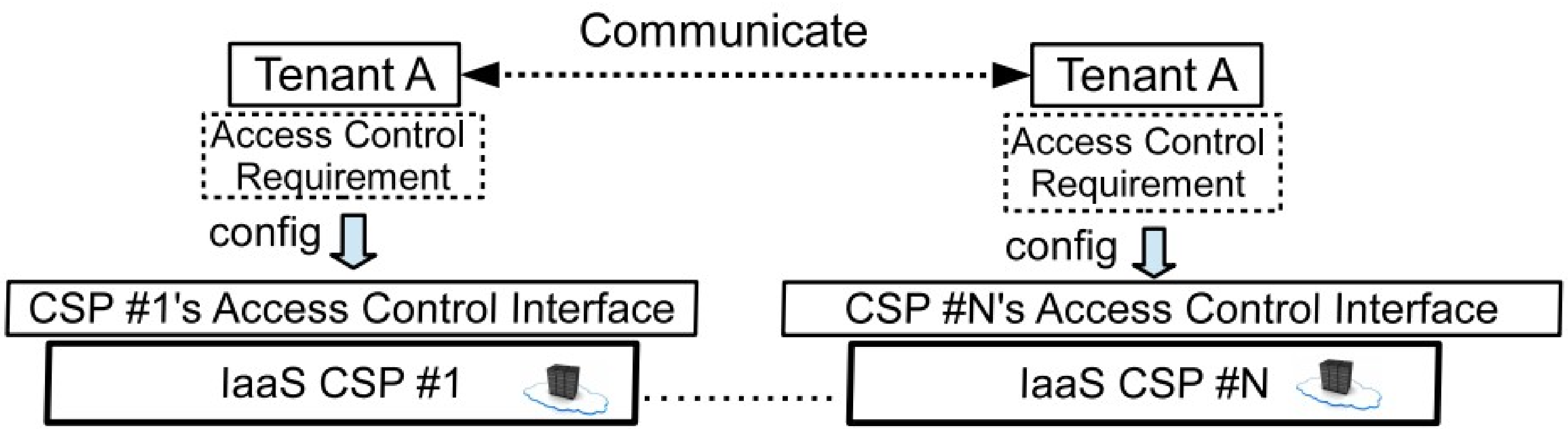
Access Control in IaaS



Requirements: Intra-CSP



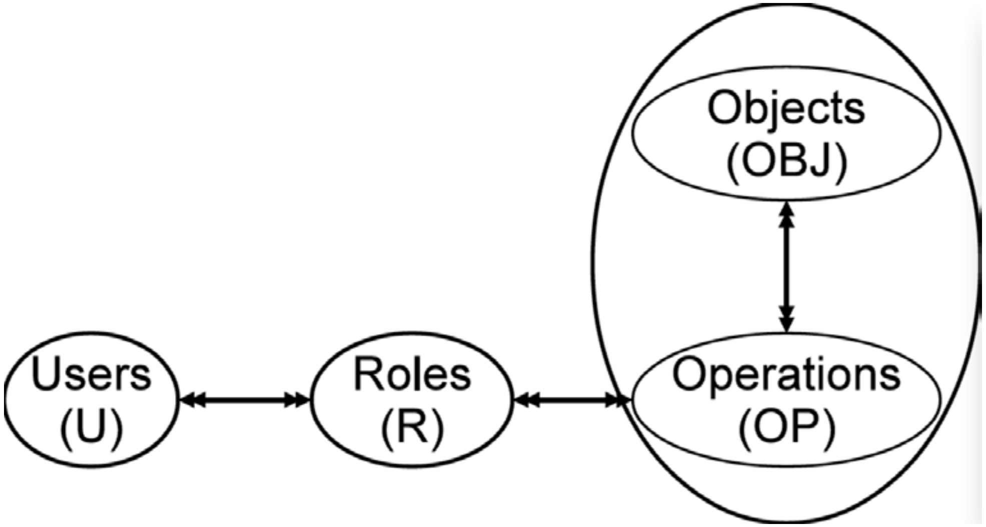
Requirements: Inter-CSP



Key Requirements

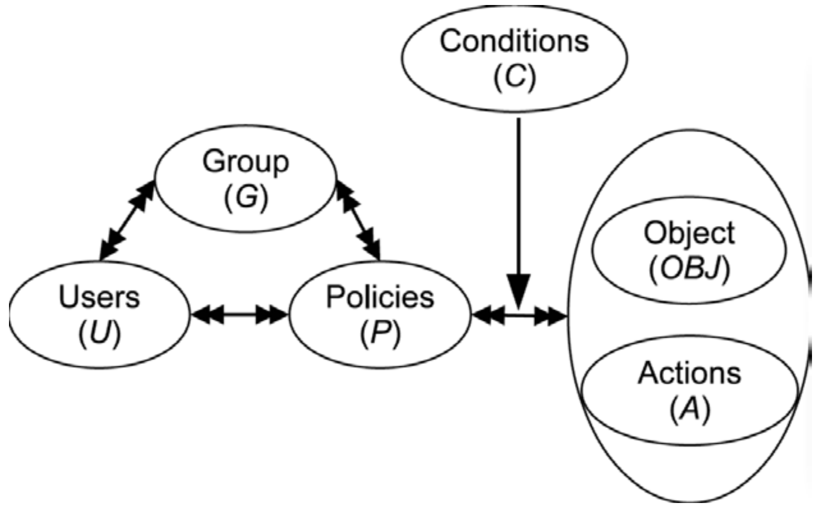
- Requirements
  - Tenants’ full control over their access control design
  - Simple yet flexible administrative policy
  - Flexible operational model
  - Strong formal foundations
- Existing Models
  - Industry Models
    - OpenStack and Amazon Web Service
  - RBAC-based Models
    - Using the legend RBAC model
  - ABAC-based Models
    - More details to follow

OpenStack (Grizzly Release)



- Limitations
  - Tenant can not configure their own policy, uses cloud role instead
  - Not able to configure tenant administrator
  - Access control on operation level, no control on object level
    - Give *identity:createUser* permission to role r1, then r1 can create users in any tenant
    - Give *nova:stop* permission to role r1, r1 can stop any machine in the tenant
  - Access control only based on role

AWS Access Control




```
{
    "Version": "2012-10-17",
    "Statement":[{
        "Effect": "Allow",
        "Action": [
            "iam:AddUserToGroup",
            "iam:RemoveUserFromGroup",
            "iam:GetGroup"
         ],
        "Resource": "arn:aws:iam::123456789012:group/MarketingGroup"
        }
    ]
}
```

AWS Access Control

- Advantages over OpenStack
  - Tenant has full control over their own policy, by account root user
  - Flexible policy : groups, user id, time, address.
  - Control over resources and operations

- Limitations
  - No automation
  - Restricted set of attributes
  - Not flexible enough, group explosion
  - No extension available (e.g., can not include customized attributes)
  - No subject and user distinction

Related ABAC models

- Formal Model
  - UCONABC (Park and Sandhu, 01): authorization, mutable attributes, continuous enforcement
  - Logical framework (Wang et al, 04): set-theory to model attributes
  - NIST ABAC draft (Hu et al, 13): enterprise enforcement

No difference between user and subject (classical models can not be configured)
No relationship of user, subject and object attributes.

- Policy Specification Language
  - SecPAL (Becker et al 03, 04), DYNPAL (Becker et al 09), Rule-based policy (Antoniou et al, 07), Binder (DeTreville 02) , EPAL1.2 (IBM, 03) , FAF (Jajodia et al 01)
- Enforcement Models
  - ABAC for web service (Yuan et al 06), PolicyMaker (Blaze et al 96)
- Implementations
  - XACML: authorization
  - SAML: pass attributes
  - OAuth: authorization

Focus on authorization and attribute release among organizations

- Attribute Based Encryption
  - KP-ABE (Goyal et al 06), CP-ABE (Bethencourt et al 07)

Limited Policy Language

Proposed Model

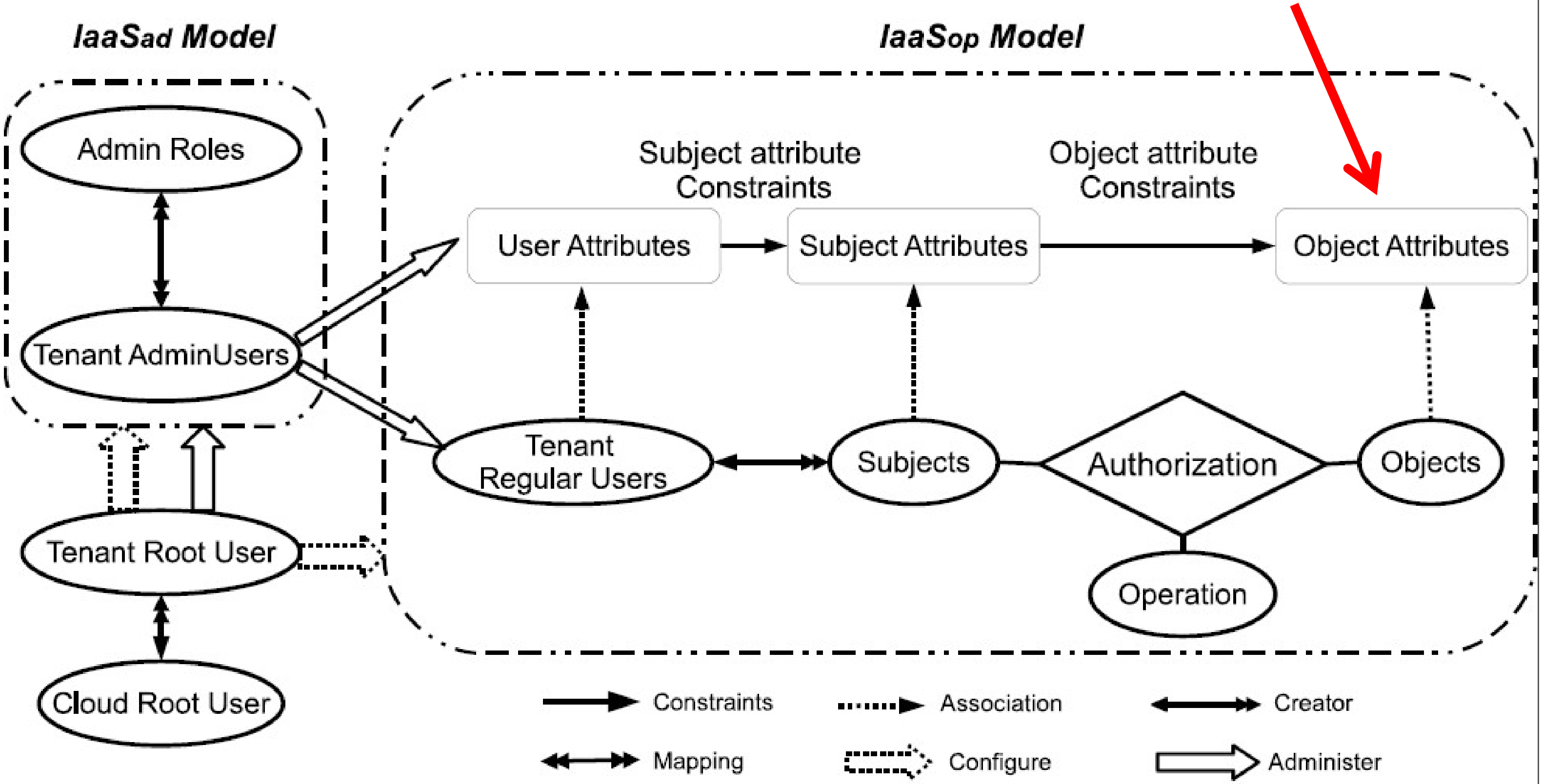
- ABAC-alpha model [1] and GURA model [2]
- Flexibility
  - Covers DAC, MAC and RBAC
  - Potentials to covers various RBAC extensions
  - Resource-level fine-grained access control
- Automation
  - User attributes inherited by subject and further object, access control automatically added for newly created objects
- Ease in policy specification and administration
  - Attributes defined to reflect semantic meaning and policy specified with certain level of relationship to natural language

[1] Xin Jin, Ram Krishnan and Ravi Sandhu, A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC In Proceedings 26th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy DBSec 2012.

[2] Xin Jin, Ram Krishnan and Ravi Sandhu, A Role-Based Administration Model for Attributes. In Proceedings of the First ACM International Workshop on Secure and Resilient Architectures and Systems (SRAS '12), Minneapolis, Minnesota, September 19, 2012

IaaS$_{ad}$ and IaaS$_{op}$ Model

Different types of object may have different sets of attributes.



World-Leading Research with Real-World Impact!

IaaS$_{op}$ Model

TReU, S and O represent finite sets of *existing* regular users, subjects and objects respectively.

UA, SA and OA represent finite sets of user, subject and object attribute functions respectively.

objType: O $\rightarrow$ OT. For each object, objType gives its type.

$\forall\, t \in$ OT, $\mathrm{O}_t = \{\mathrm{obj} \mid \mathrm{obj} \in \mathrm{O} \wedge t = \mathrm{objType(obj)}\}$, represents objects of type t.

oaType: OA$\rightarrow 2^{\mathrm{OT}}$. For each object attribute, oaType gives its types.

$\forall\, t \in$ OT, $\mathrm{OA}_t = \{\mathrm{oa} \mid \mathrm{oa} \in \mathrm{OA} \wedge t \in \mathrm{oaType(oa)}\}$, represents object attributes of type t.

SubCreator: $S \rightarrow U$. For each subject SubCreator gives its creator.

For each att in UA $\cup$ SA $\cup$ OA, SCOPE_{att} represents the attribute's scope, a finite set of *atomic* values.

attType: UA $\cup$ SA $\cup$ OA $\rightarrow$ {set, atomic}. It specifies attributes as set or atomic valued.

PER represents finite set of operations.

Each attribute function maps elements in TReU, S and O to atomic or set values.

$$\forall ua \in \mathrm{UA}.\, ua : \mathrm{TReU} \rightarrow \begin{cases} \mathrm{SCOPE}_{ua} & \text{if attType}(ua) = \text{atomic} \\ 2^{\mathrm{SCOPE}_{ua}} & \text{if attType}(ua) = \text{set} \end{cases}$$

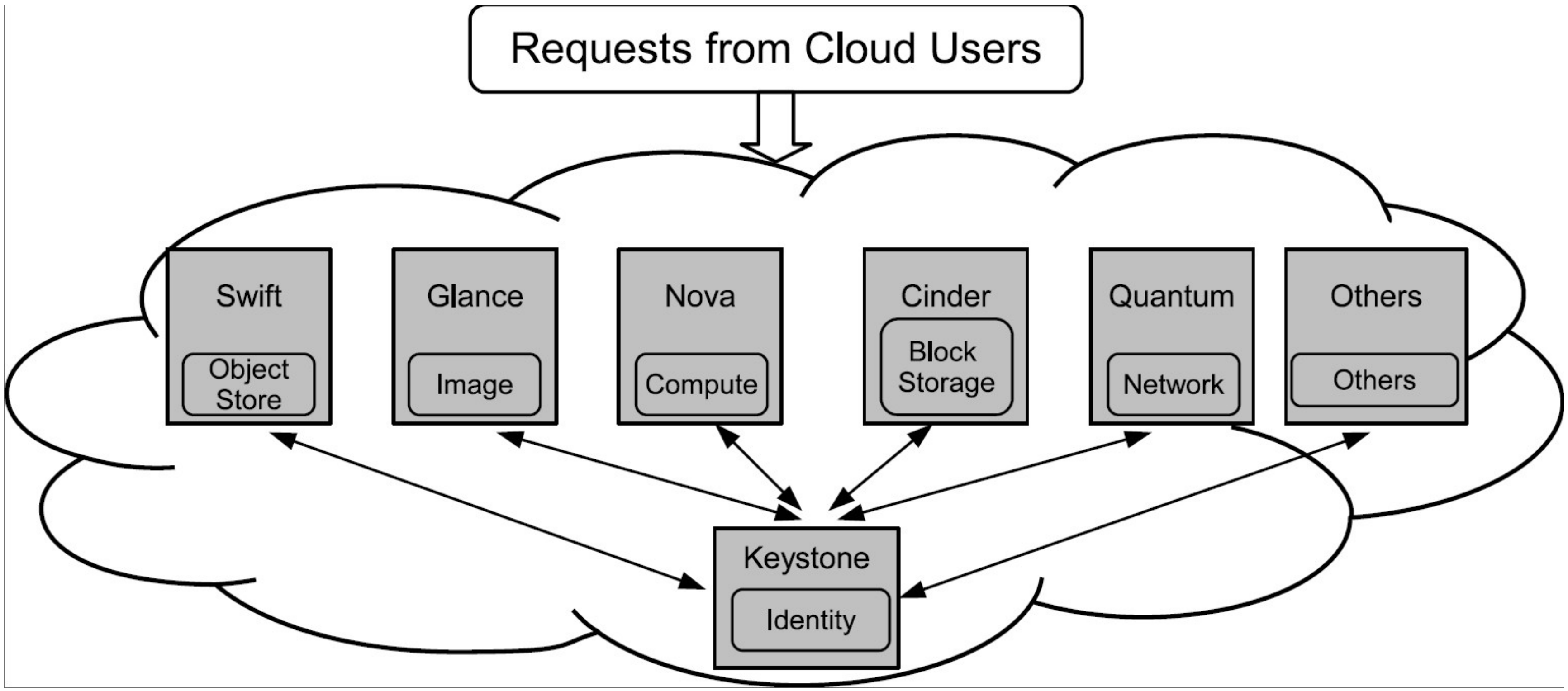
$$\forall sa \in \mathrm{SA}.\, sa : \mathrm{S} \rightarrow \begin{cases} \mathrm{SCOPE}_{sa} & \text{if attType}(sa) = \text{atomic} \\ 2^{\mathrm{SCOPE}_{sa}} & \text{if attType}(sa) = \text{set} \end{cases}$$

$$\forall t \in \mathrm{OT}.\forall oa \in \mathrm{OA}_t.oa : \mathrm{O}_t \rightarrow \begin{cases} \mathrm{SCOPE}_{oa} & \text{if attType}(oa) = \text{atomic} \\ 2^{\mathrm{SCOPE}_{oa}} & \text{if attType}(oa) = \text{set} \end{cases}$$

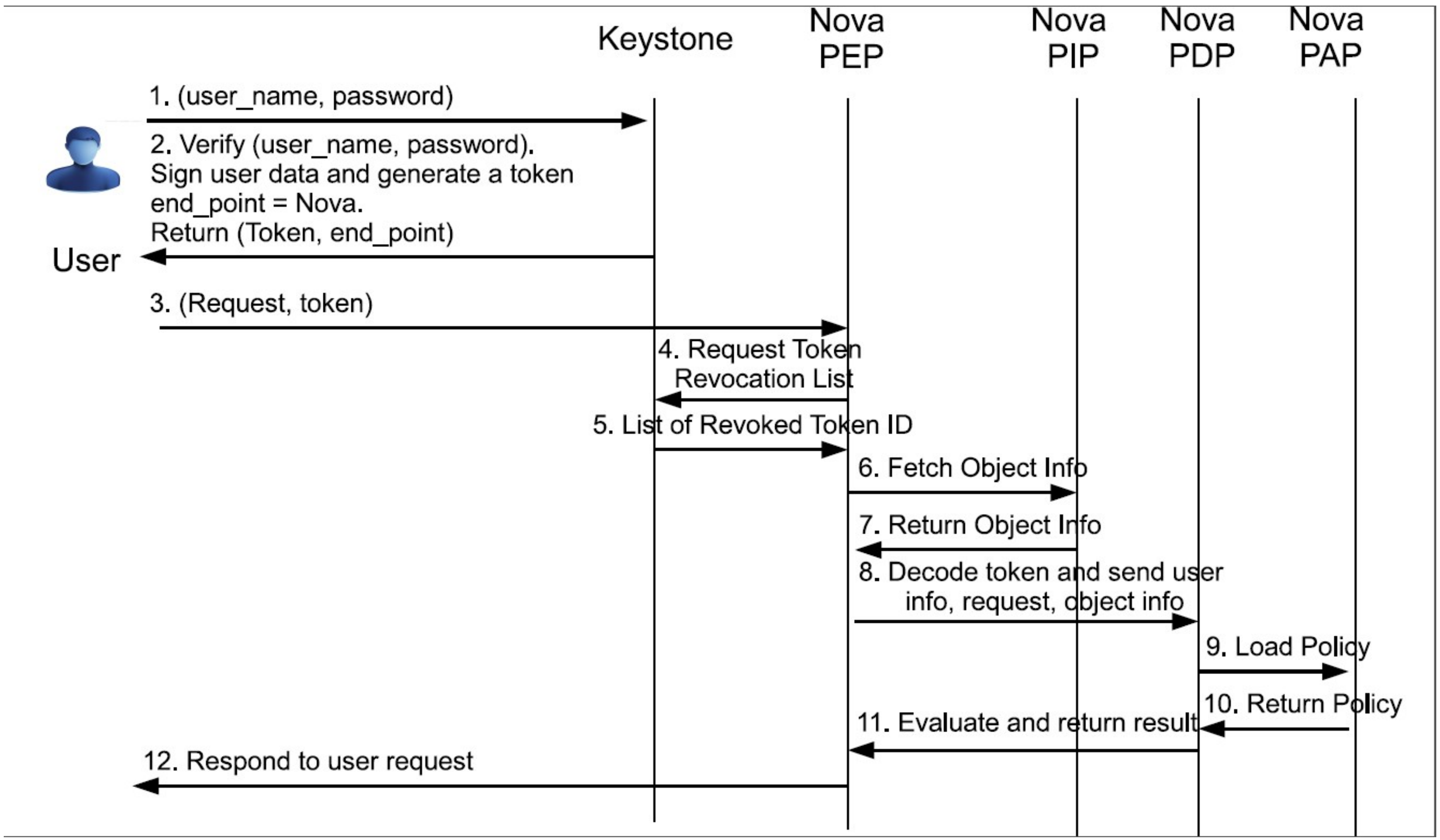
IaaS_{ad} Model

Part I. Basic Sets and Functions	
CRU, TRU represent the cloud root user and tenant root user respectively. TAU represents finite set of tenant administrative users. AR represents a set of administrative roles and UAR represent user-role assignment, i.e., UAR $\subseteq$ TAU$\times$ AR.	
Part II. Operations	
Operations	Updates
1. Operations for Cloud Root User	
1.1 createTenant(req:CRU, tenant:NAME)	$T' = T \cup \{tenant\}$
1.2 createRootUser(req:CRU, u:NAME, tenant:T)	$TRU = \emptyset$, $TRU = \{u\}$
2. Operations for Tenant Root User	
2.1 createUserAttr(req:TRU, ua:NAME, type: {set, atomic})	$UA' = UA \cup \{ua\}$, attType(ua) = type
2.2 createSubAttr(req:TRU, sa:NAME, type: {set, atomic})	$SA' = SA \cup \{sa\}$, attType(sa) = type
2.3 addSubConstr (req:TRU, policy:POLICY)	$SubConstr' = SubConstr \cup \{policy\}$
2.4 createObjAttr (req:TRU, oa:NAME, type: {set, atomic}, oat:OT)	$OA' = OA \cup \{oa\}$, attType(oa) = type, oaType(oa) = oat
2.5 addObjConstr (req:TRU, policy:POLICY)	$ObjConstr' = ObjConstr \cup \{policy\}$
2.6 addAuthz (req:TRU, policy:POLICY)	$Authz' = Authz \cup \{policy\}$
2.7 createAdminRole(req:TRU, adminrole:NAME)	$AR' = AR \cup \{adminrole\}$
2.8 createAdminPolicy(req:TRU, policy:POLICY)	$AdminPolicy' = AdminPolicy \cup \{policy\}$
2.9 addAminUserRole(req:TRU, u:TReU, r:AR)	$UAR' = UAR \cup \{(u, r)\}$
3. Operations for Tenant Administrative Users [17]	
3.1 addUser(req:TAU, u:NAME)	$TReU' = TReU \cup \{u\}$
3.2 add(req:TAU, u:TReU, att:UA, value:SCOPE_{att})	$att(u)' = att(u) \cup \{value\}$
3.3 delete(req:TAU, u:TReU, att:UA, value:SCOPE_{att})	$att(u)' = att(u) \setminus \{value\}$
3.4 assign(req:TAU, u:TReU, att:UA, value:SCOPE_{att})	$att(u)' = value$

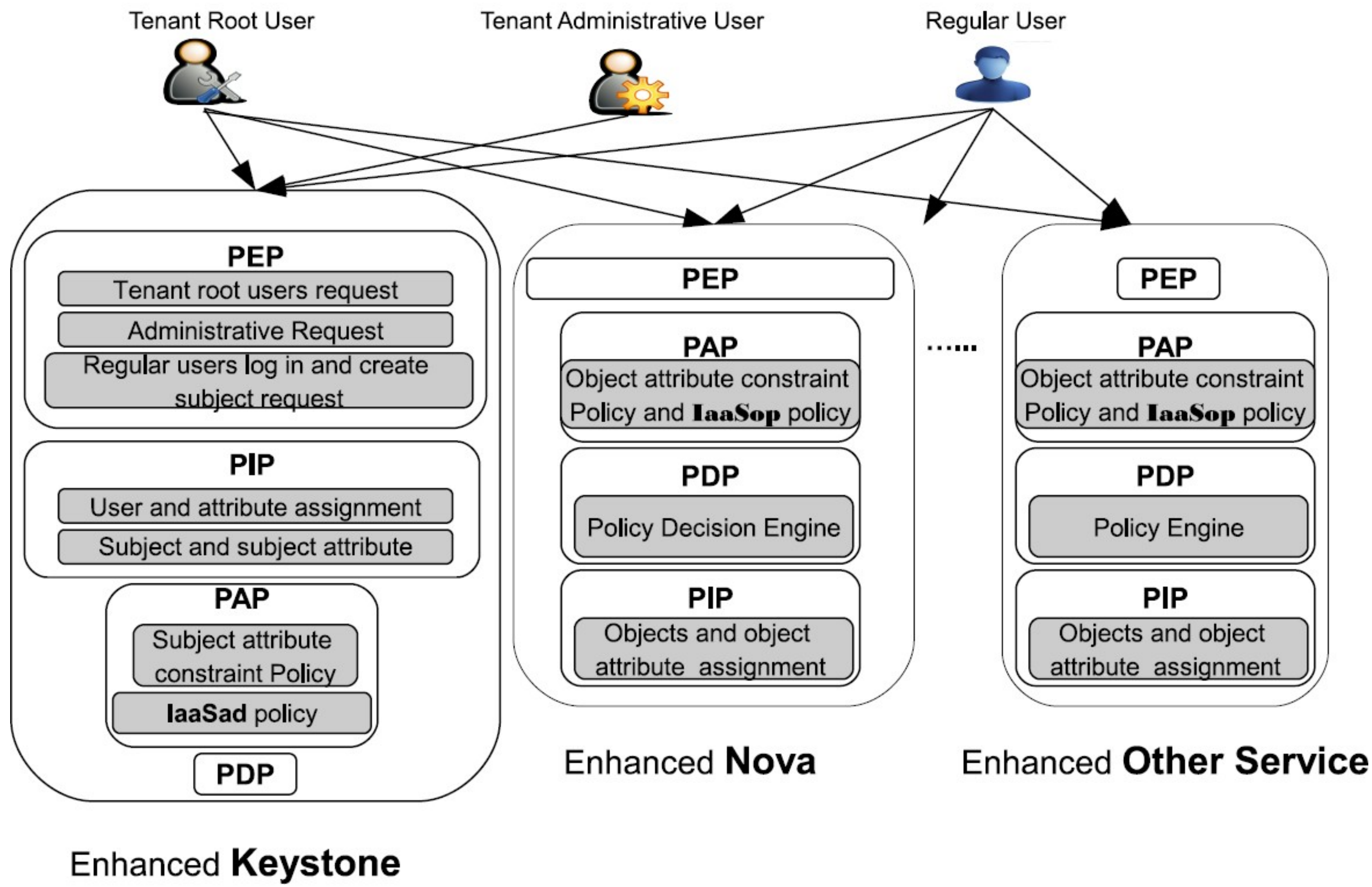
Proof of concept in OpenStack



OpenStack Authorization for Nova

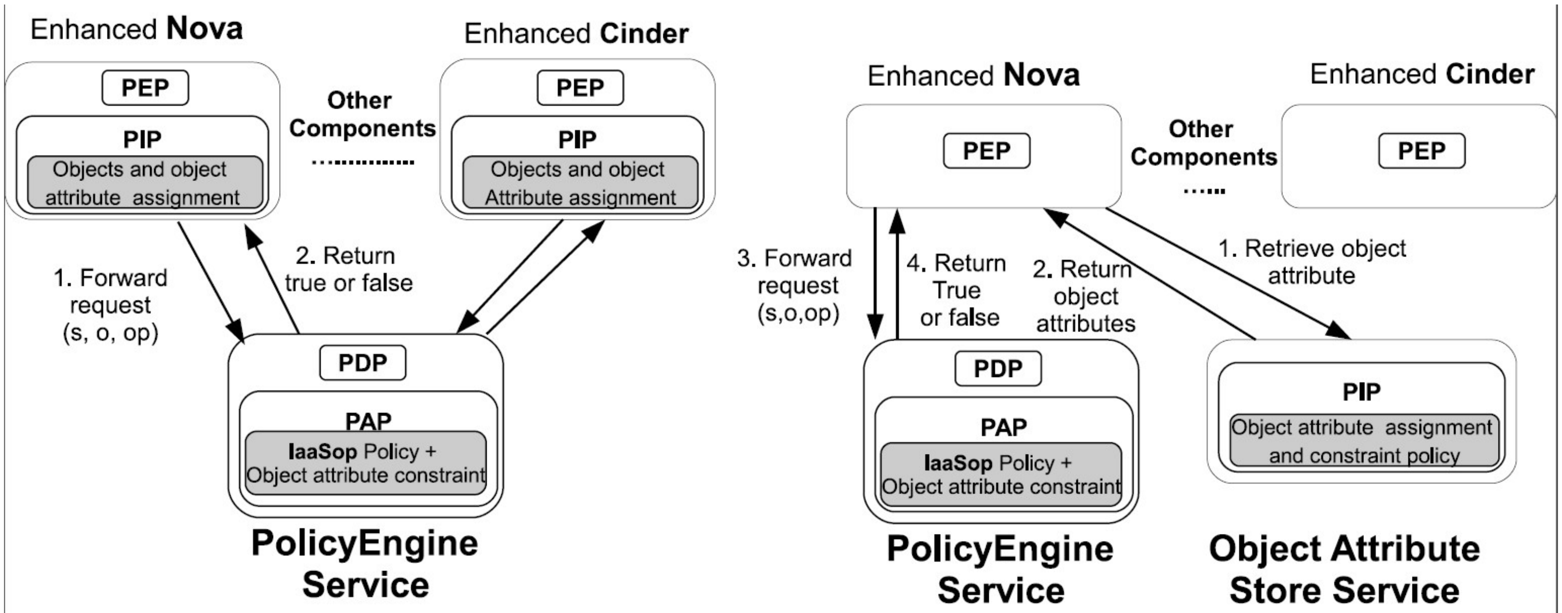


ABAC Enforcement in OpenStack



Enforcement Model I

Alternative Enforcement Models



(a) Enforcement Model II

(b) Enforcement Model III

Conclusion

- Summary
  - We illustrate the case of access control in cloud IaaS
  - We summarize four core requirements of access control models
  - Existing models fail to satisfy those requirements
  - By connecting existing models with additional features, we proposed $IaaS_{op}$ and $IaaS_{ad}$ models based on ABAC
- Future work
  - Different types of attributes: system wide, service-specific attributes.
  - Various types of subject attributes constraints, object attribute constraints.
  - Reachability analysis on $IaaS_{op}$ and $IaaS_{ad}$ instance.

Conclusion

- Thanks. Questions?