SPECIFYING SECURITY POLICY: ORCA

CS6410

Ken Birman

Context

- As we roll out increasingly ambitious distributed platforms, and share them among demanding users who have “it’s all mine” mentalities, we get really challenging resource sharing scenarios
- Today’s PlanetLab illustrates the risks
  - Internet-scale experimental resource (created by Larry Peterson and team members)
  - More than 400 small clusters that comprise a global platform for experiments
  - Used by researchers worldwide

How does Planetlab fit the “cloud”?

- Cloud platforms are supported by data centers: large clusters (very large), but the similarity is real
- And at multiple locations worldwide
- They try to keep their systems running near full load

- But PlanetLab has no way to limit users from overloading their machines
- So in the cloud, we often see “full” but not “massive overload”; PlanetLab can easily be totally swamped

GENI

- Goal is to replace PlanetLab the GENI testbed
  - Eventually, a world-wide infrastructure
  - Whereas PlanetLab offers a standard Unix VM environment, GENI permits "raw" access to layer-2 of the network. So one can define new kinds of routing services and "tunnel" traffic through GENI applications
- GENI vision: developers create services or even entire overlay networks that could even control dedicated network links, if those are available
  - Notice that this is actually like a VLAN in a datacenter...

What might these services do?

- One could imagine a wide range
  - A system for monitoring the Internet and continuously tracking loads: a form of continuous tomography
  - A new kind of service for building secure end-to-end network paths
  - A next generation of cloud-hosted technologies for "monitoring" the real world (the "Internet of Things")
  - A completely new "overlay" Internet with great realtime (or security, or "streaming"...) properties
  - A platform that caches objects on behalf of mobile users so that their connectivity experience is improved
  - New kinds of discovery services, fancier than DNS...

So… GENI's job is to…

- … host the experimental prototypes of these kinds of next-generation Internet applications and services
- Our hope would be that many result in fantastic research papers in the networks community (SIGCOMM, NSDI, SOSP, SIGCOMM-IMC) and that some transition and become exciting new products
- GENI could be an incredible enabler… if successful

What's the big risk?

- Even in the cloud, sharing machines is a really hard thing to pull off successfully
  - Not every style of computing works well in the cloud
  - Some applications experience too many delays and break: various scheduling requirements are violated
- Hakim: In Planetlab, as much as 60% of the resources in your slice tend to be unavailable or flakey!
- In GENI, where many services will have realtime needs, special resource ownership requirements or other kinds of "rules", overload could be a disaster

How to deal with contention?

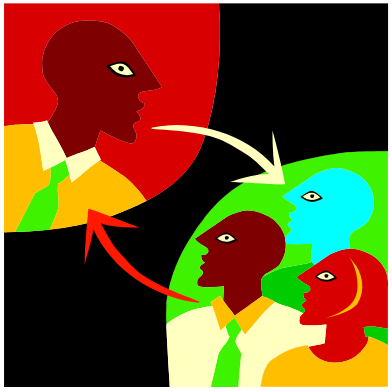
- On today's cloud platforms, the approach is fairly ad-hoc
  - Scheduler tries to "pack" each physical machine with enough work to keep it fairly busy
  - Because of variable loads, the usual approach is to "learn" a load model for each application and use the learned behavior in the scheduler packing algorithm
- This works, but can leave nodes overcommitted
  - If that happens, many cloud systems just kill some of the excess load and restart those tasks elsewhere

GENI goals

- Each user has an associated "slice" of GENI
  - Slices has a predicted resource use profile
  - GENI wants to guarantee that these profiles will be respected
  - Better to refuse a requested allocation than to grant it but then be overloaded and unable to meet demand
- Leads to a requirement: a way for GENI users to specify their needs

Principals

Researcher: A user that wishes to run an experiment or service in a slice, or a developer that provides a service used by other researchers.



A **slice authority (SA)** is responsible for the behavior of a set of slices, vouching for the users running experiments in each slice and taking appropriate action should the slice misbehave.

A **management authority (MA)** is responsible for some subset of substrate components: providing operational stability for those components, ensuring the components behave according to acceptable use policies, and executing the resource allocation wishes of the component owner.

Next few slides By Aaron Falk

Components & Resources

Component

Resource

Some resources describe non-configurable characteristics of the component.

Some measurements are available as resources

Transmission Channel	
Route	ρ_r
Cable	ρ_c
Fiber	ρ_f
Spectrum	ρ_s
Endpoint ID	ρ_e
S/N measurements	μ_e

Computer	
CPU	
Memory	
Disk	
BW	

Other resources are pools which may be allocated under some constraints.

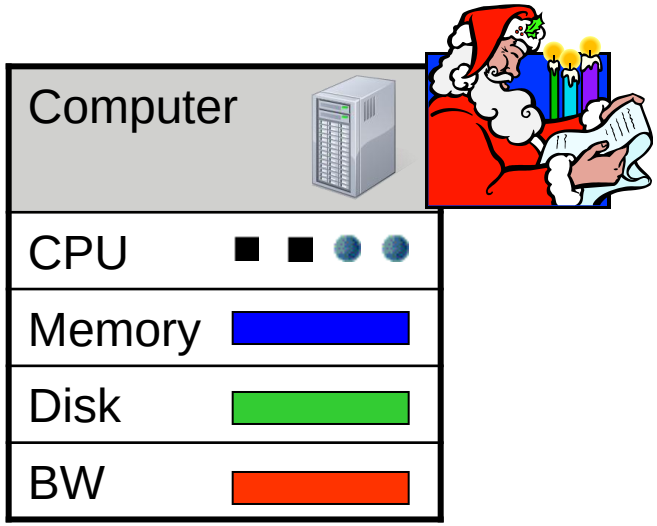
Optical Switch	
Fiber ID	ρ
Switch Port	
Channel	
Band	

Spectrum Analyzer	
Location	
Sample period	
Sample BW	

Measurement equipment may also appear as components

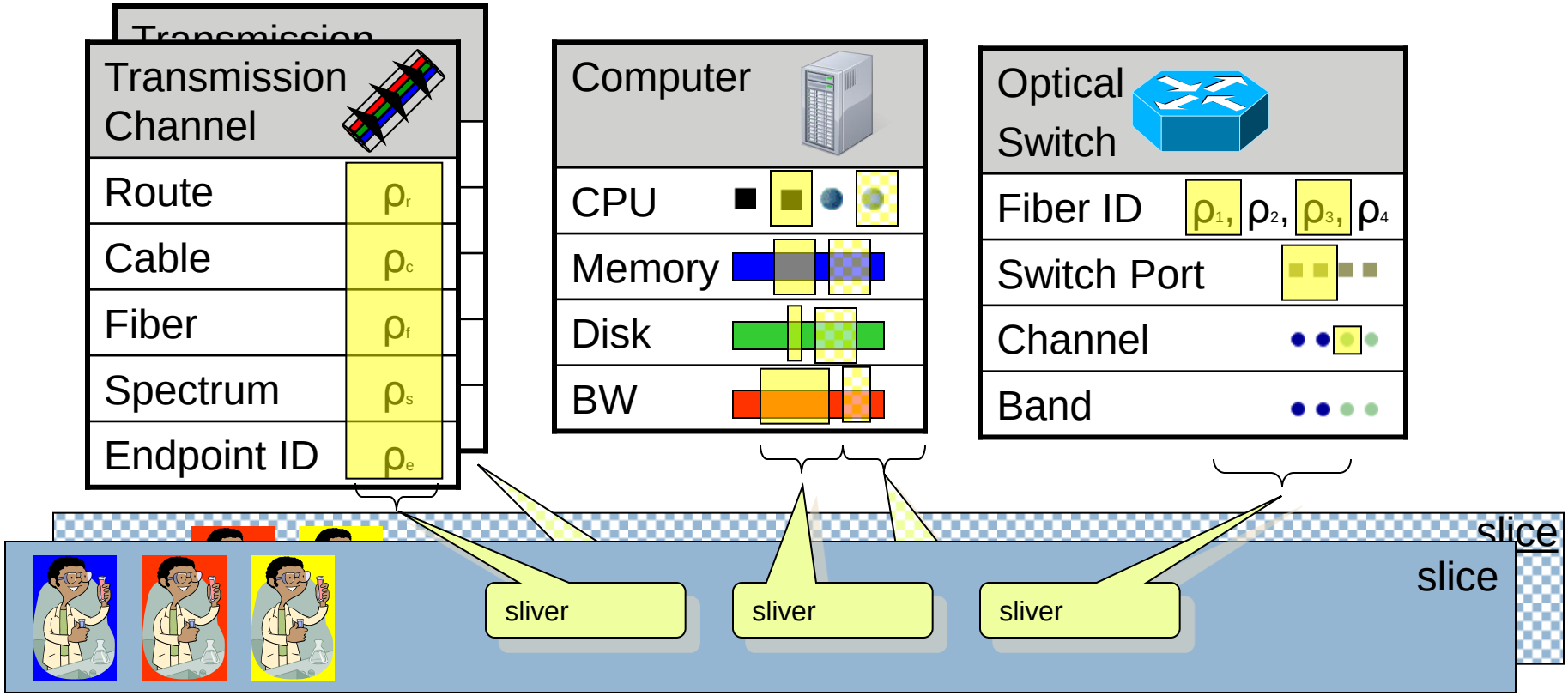
Component: An object representing a physical device in the GENI substrate. A component consists of collection of **resources**. Such physical resources belong to precisely one component. Each component runs a **component manager** that implements a well-defined interface for the component. In addition to describing physical devices, components may be defined that represent logical devices as well.

Component Managers



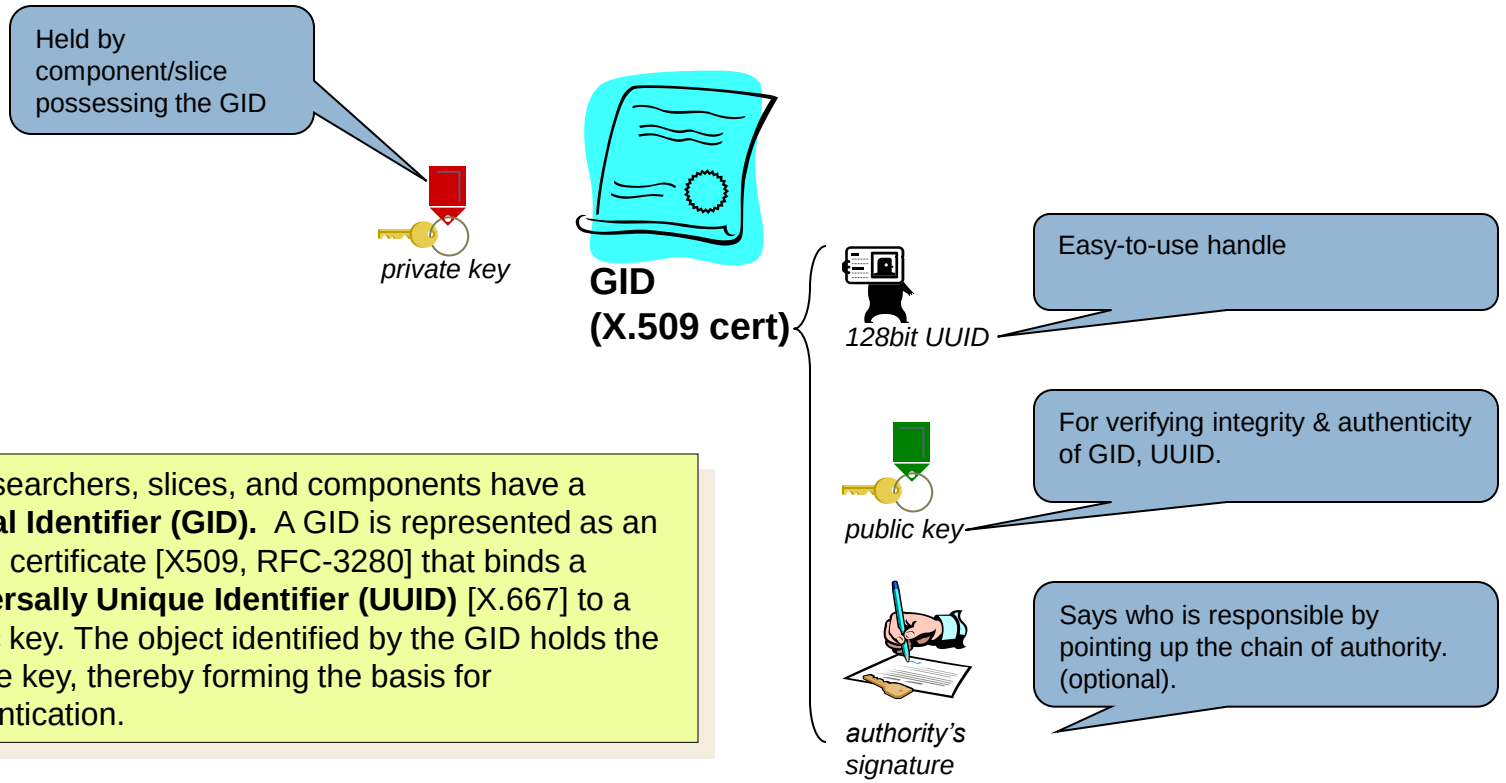
Each component is controlled via a **component manager (CM),** which exports a well-defined, remotely accessible interface. The component manager defines the operations available to user-level services to manage the allocation of component resources to different users and their experiments. A management authority (representing the wishes of the owner) establishes policies about how the component's resources are assigned to users.

Slivers & Slices



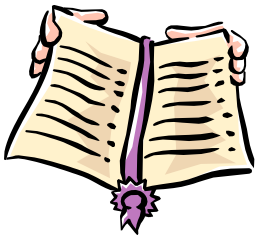
From a researcher's perspective, a **slice** is a substrate-wide network of computing and communication resources capable of running an experiment or a wide-area network service. From an operator's perspective, slices are the primary abstraction for accounting and accountability—resources are acquired and consumed by slices, and external program behavior is traceable to a slice, respectively. A slice is defined by a set of slivers spanning a set of network components, plus an associated set of users that are allowed to access those slivers for the purpose of running an experiment on the substrate. That is, a slice has a name, which is bound to a set of users associated with the slice and a (possibly empty) set of slivers.

Identifiers



All researchers, slices, and components have a **Global Identifier (GID).** A GID is represented as an X.509 certificate [X509, RFC-3280] that binds a **Universally Unique Identifier (UUID)** [X.667] to a public key. The object identified by the GID holds the private key, thereby forming the basis for authentication.

Registries & Names



Top-level authority name: `geni`

Names are human-readable and hierarchical

Top-level authority GID: 

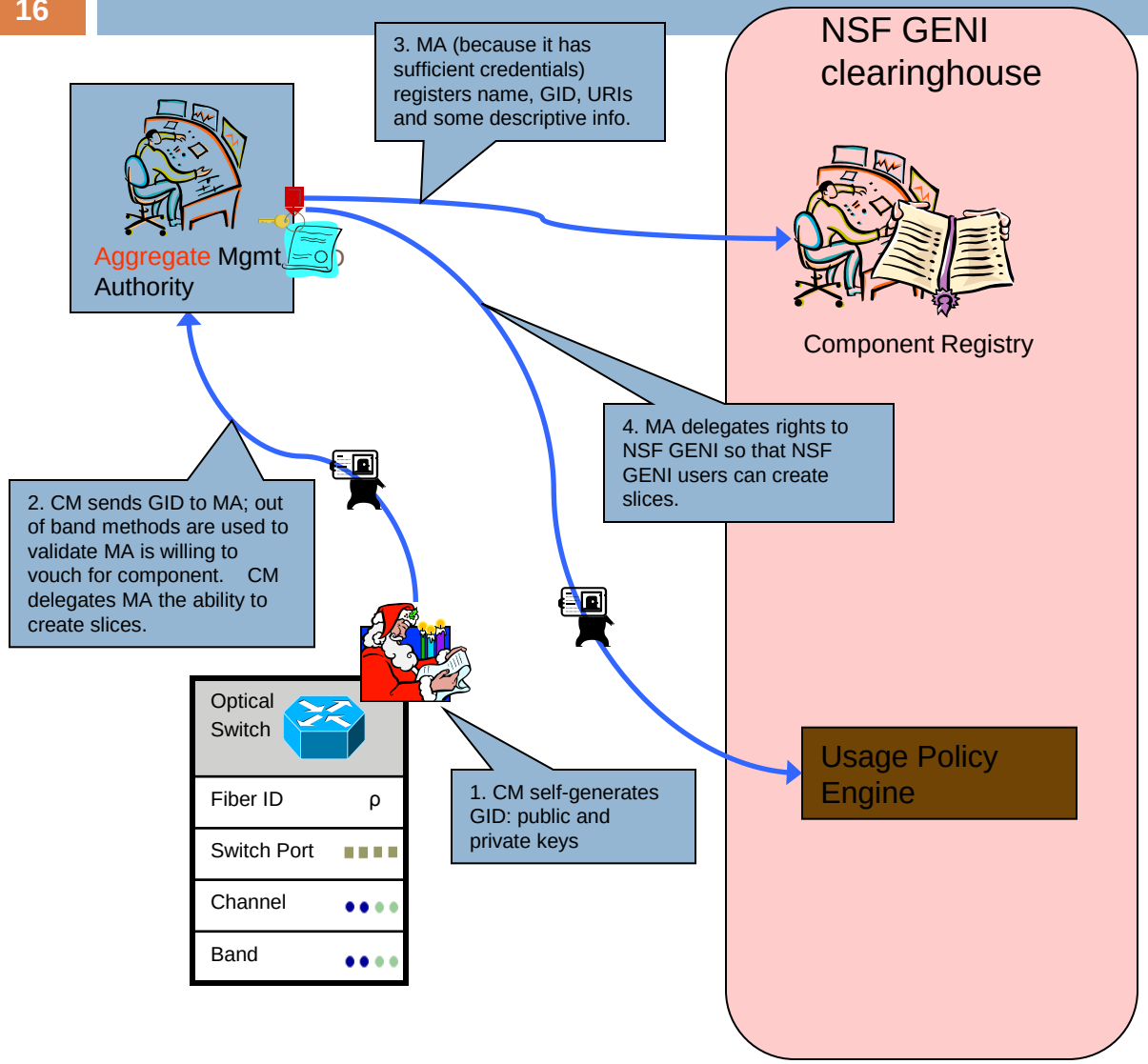
Sub-authority name	Sub-authority GID	Sub-authority contact info (e.g., URI, etc)	other
`geni.sl` 		`http://geni.net/ops/sl`	
`geni.cm` 		`http://geni.net/ops/cmp`	

A **name registry** binds strings to GIDs, as well as to other domain-specific information about the corresponding object (e.g., the URI at which the object's manager can be reached, an IP or hardware address for the machine on which the object is implemented, the name and postal address of the organization that hosts the object, and so on).

The **component registry** maintains information about a hierarchy of management authorities, along with the set of components for which the MAs are responsible. This registry binds a human-readable name for components and MAs to a GID, along with a record of information that includes the URI at which the component's manager can be accessed; other attributes and identifiers that might commonly be associated with a component (e.g., hardware addresses, IP addresses, DNS names); and in the case of an MA, contact information for the organization and operators responsible for the set of components.

The **slice registry** maintains information about a hierarchy of slice authorities, along with the set of slices for which the SAs have taken responsibility. This registry binds a human-readable name for slices and SAs to a GID, along with a record of information that includes email addresses, contact information, and public keys for the set of users associated with the slice; and in the case of an SA, contact information for the organization and people responsible for the set of slices.

Component Registration



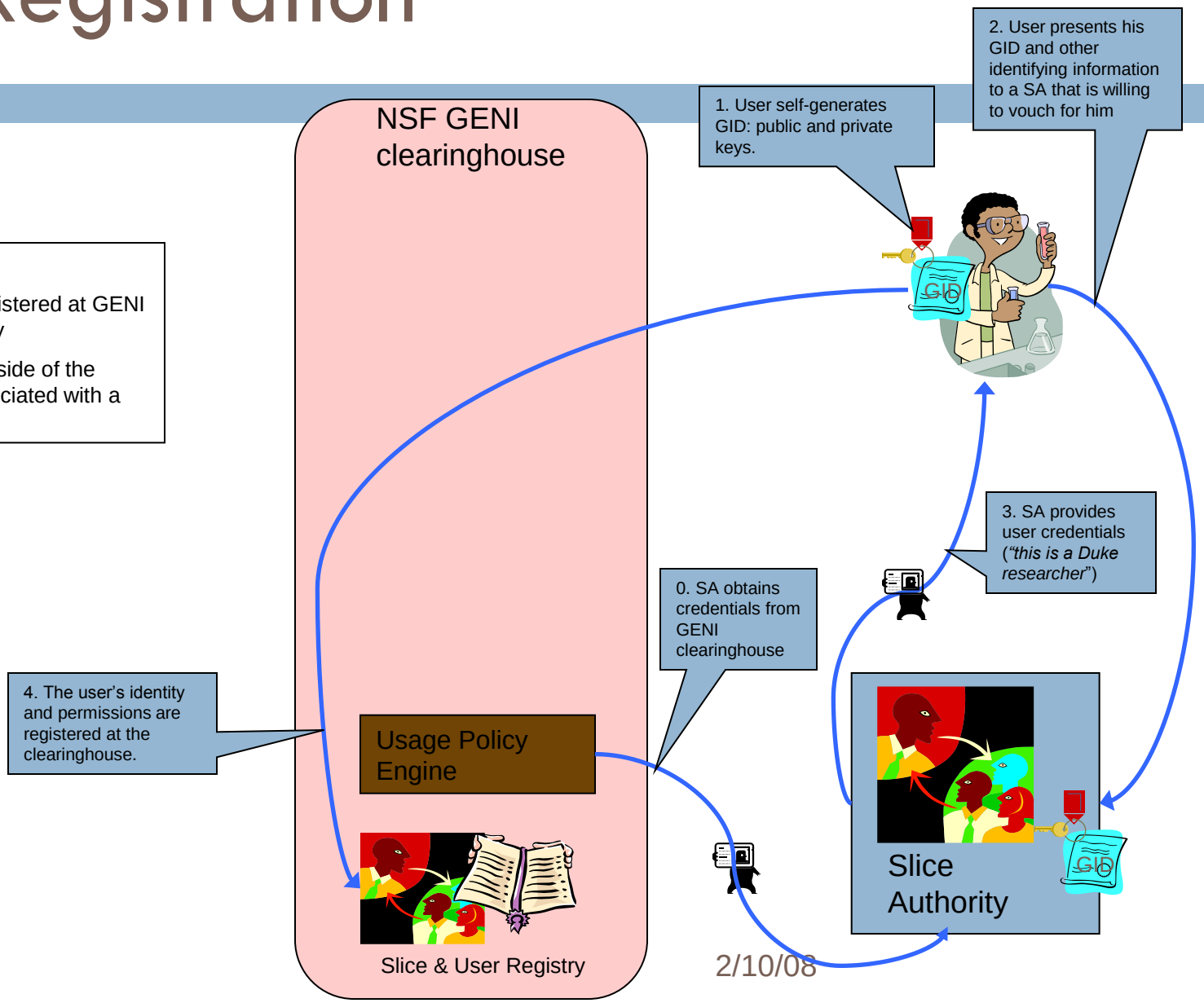
Notes:

- Identity and authorization are decoupled in this architecture. GIDs are used for identification only. Credentials are used for authorization. I.e., the GID says only *who* the component is and nothing about *what it can do* or *who can access it.*
- Assuming aggregate MA already has credentials permitting access to component registry

User Registration

Notes:

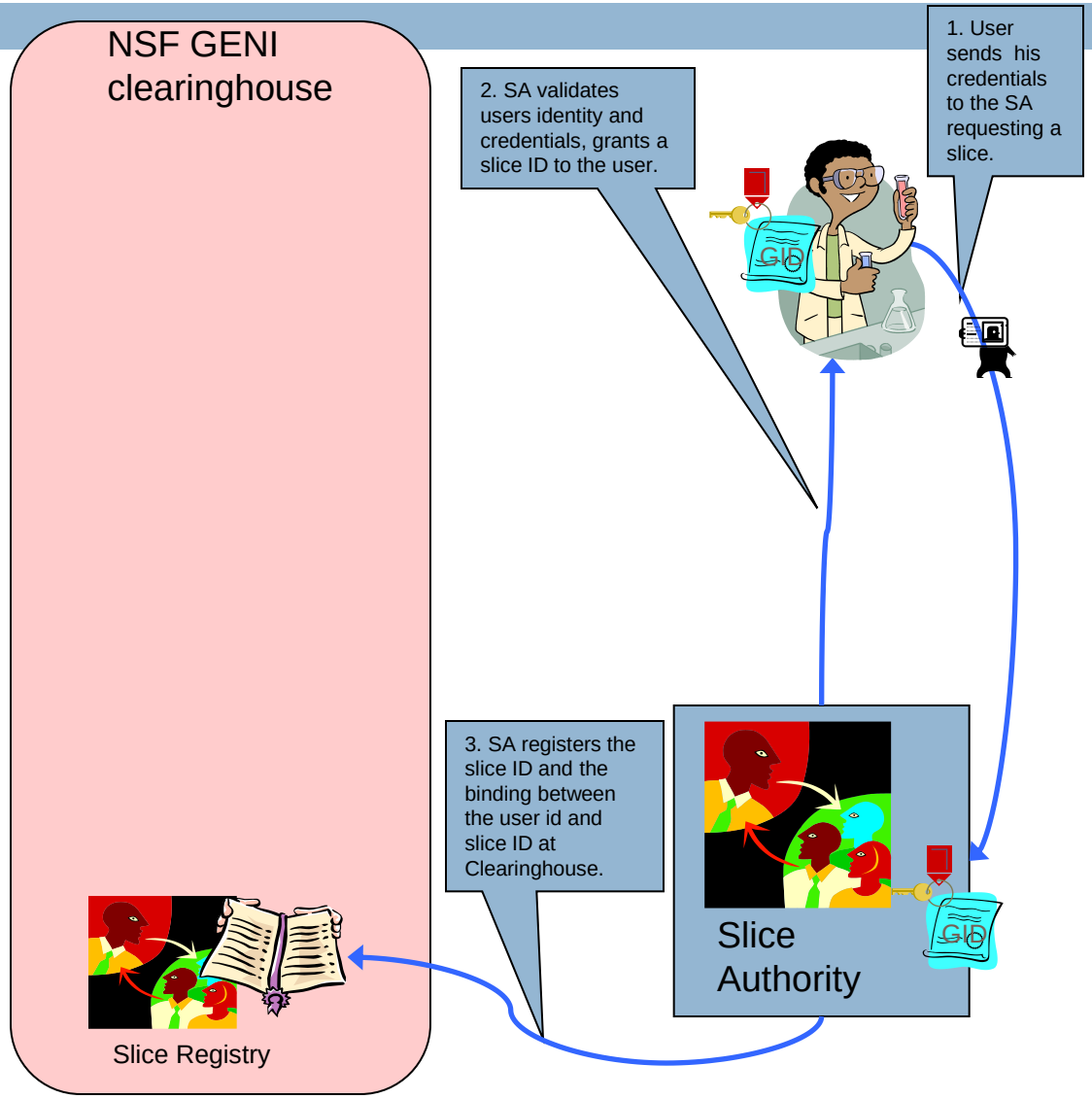
- Assuming SA is registered at GENI Slice & User registry
- Assuming SA is outside of the clearinghouse, associated with a research institution.



Slice Creation

Notes:

- A slice can exist with no components in it. So, the minimal slice consists of a slice ID bound to a user.



Today's papers

- First paper focuses on the resource scheduling and management challenges they face in GENI
- Second paper is about the proposed security infrastructure, which is based on identity and trust
- Third (optional) paper shows how the trust mechanisms map to logic formalism

Paper 1: Controlling dynamic guests

- Presents the GENI slice model
  - Slice hosts user "appliances"
  - Each appliance visible at one or more GENI hosting sites
- Each appliance communicates its needs via what they call a "guest controller"
  - Allows appliance to dynamically vary its needs, e.g. add or remove VMs from the deployment, or other resources
  - Schedules resource use within the appliance deployment
- A set of applicances, managed by ORCA, can then be deployed onto a collection of GENI nodes

Secure instantiation of an appliance on an ORCA-managed node

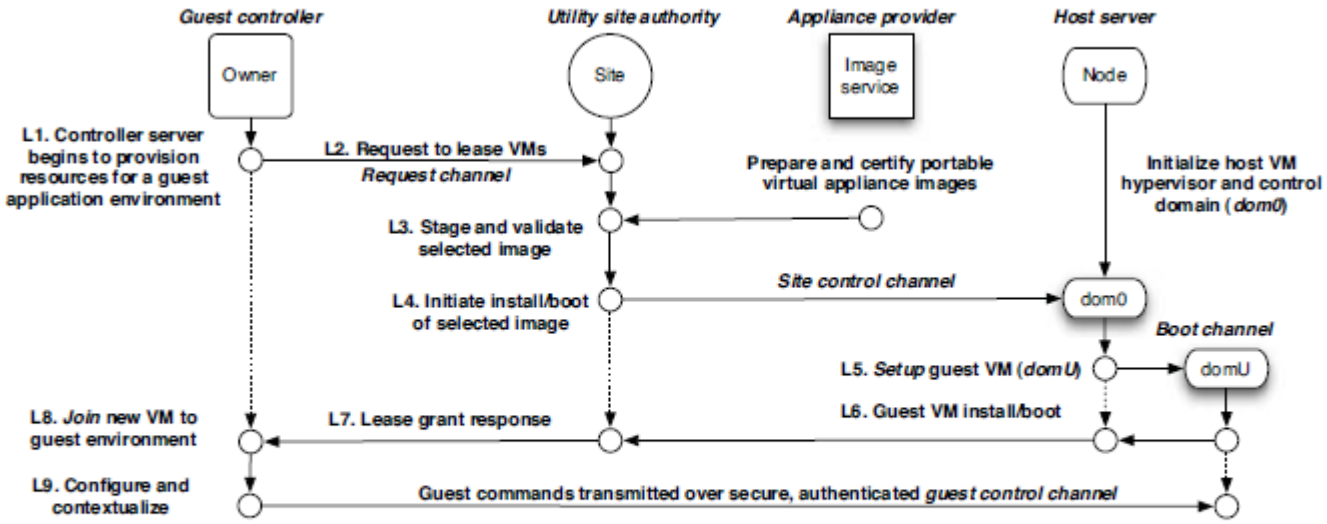


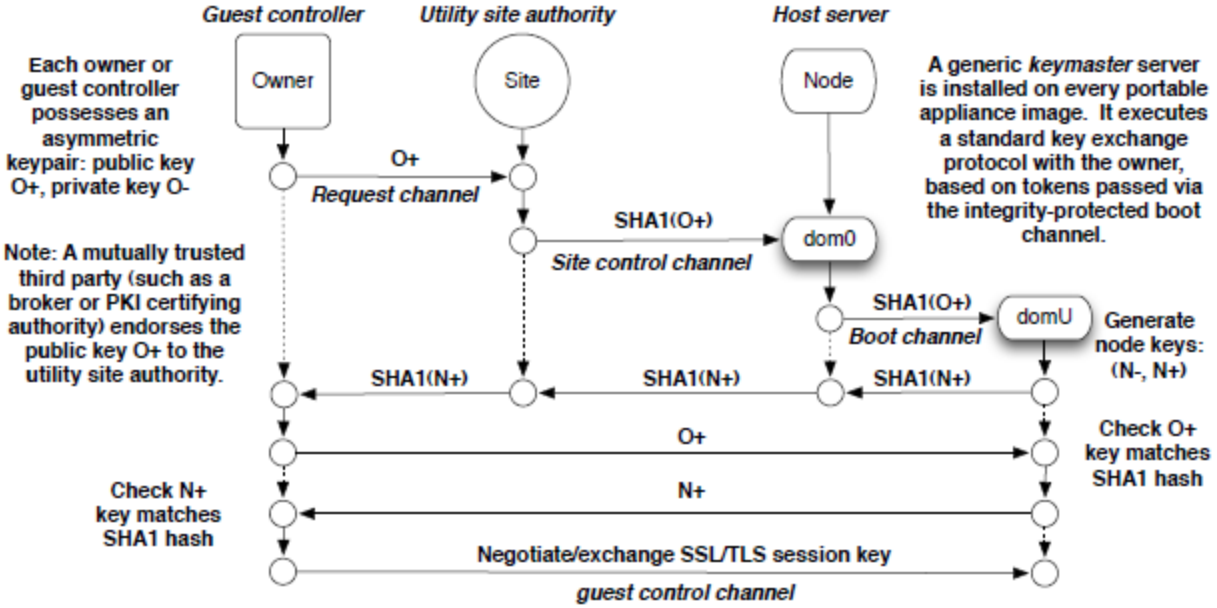
Fig. 1. Secure contextualization of a node instantiated from a portable image in ORCA.

Each guest lives in a form of VLAN

- A virtual LAN is a kind of subnet architecture
  - Mimics a dedicated ethernet
  - Widely used in enterprise data centers
- GENI deployment on a set of nodes is "like" a VLAN in that the appliances are isolated from one-another and see a virtual, private, infrastructure
  - But resources are "leased"
  - When lease expires, GENI will reclaim them

Security keys play a central role

- GENI has an efficient security key exchange scheme
- Uses public keys to efficiently create SSL or TLS session keys



Fair resource sharing

- Based on a modified "weighted fair queuing" algorithm (originally invented by Alan Demers!)

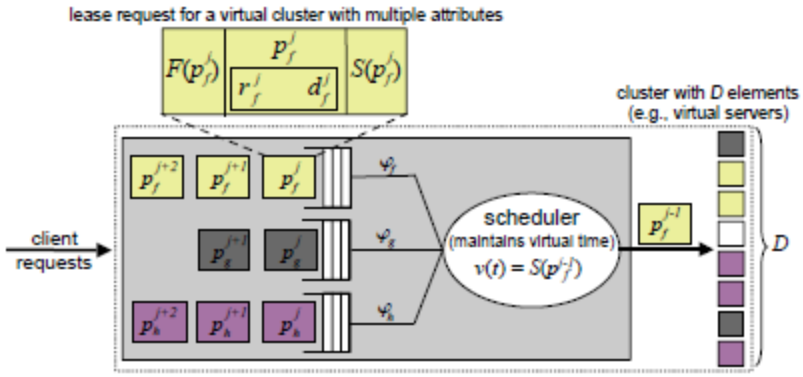


Fig. 3. Overview of the system model for WINKS.

Guest controller API

- Guest controller has options to:
  - Issue requests that span multiple hosts ("wide" requests)
  - Maintain flow ordering even during resource allocation
  - Perform "calendar" scheduling (advance reservations)
  - Backfill scheduling: First schedules wide requests, then "backfills" with smaller tasks
  - Dynamic resizing (to the extent feasible)

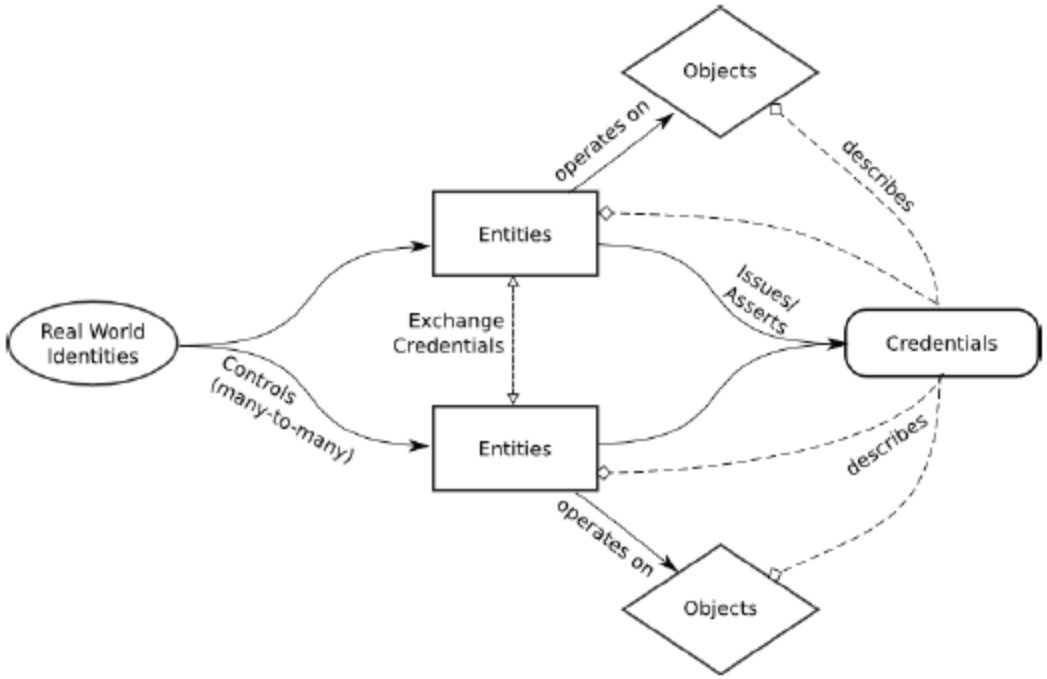
Identity and authorization

- For these purposes GENI uses a Stanford-based logic (RT/ABAC) for trust-based role management
  - The logic is a *mathematical framework* for expressing:
    - Identity (individual and group entities),
    - Relationships (as in "John is a Cornell student and is taking CS6410"), or "Ken is a consultant to Z's Red Sky Corp."
    - Policies written using these base elements, such as "Cornell CS6410 students are entitled to a 20% discount on Ken's book if they buy it directly from Springer Verlag"
  - Previous logics were much less expressive

Features of RT/ABAC

- Authorization based on user identity, group affiliations, nature of the specific activity (slice)
- Flexible support for delegation of rights, including "capability"-based access control
  - In this model, if you present a valid ticket for a resource that ticket (capability) will ensure access
  - Flexible authorization policies and delegated policy evaluation that allows policies from multiple agents to be combined
  - Captures trust policies in a declarative way with a rigorous formalism that reduces to an inductive logic

Applying RT/ABAC to GENI



Implementation approach

- Public keys used for communication internal to and between elements of the system
  - An entity is a component that can send and receive messages using a secured channel (e.g. SSL/TLS)
  - ORCA security architecture employs only signed messages, allows each entity to know what entities it is interacting with in an unbreakable way
- All notions of identity are implemented using “roles”

Role

- In ORCA, the decision of an entity to trust another entity is determined by roles bound to it
  - A role is like an *attribute*: a form of named property
  - A single entity can have arbitrarily many roles and new roles can be defined dynamically
  - Roles can be extended and also can be retracted
- An "entity domain" is a set of users for some institution, like "Cornell University" or "Microsoft"
- ORCA also allows creation of groups, like "CS6410 students"

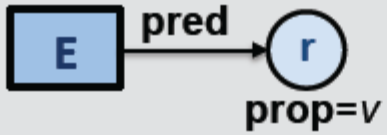
Federating identities

- ORCA is designed to federate collections of identies managed via ORCA but by different organizations
  - E.g. Cornell manages Cornell entities
  - Z's Red Sky company manages its own entities
  - ORCA is still able to carry out a policy in which Red Sky permits Cornell CS6410 students to experiment with Z's new network sensor devices

Key elements of the trust logic

- The logic is normally written using a simple high-level notation standard for deductive frameworks
- "Compiles" into datalog: a kind of database for storing logic tuples and performing computation on them
- Deciding if an action should be authorized becomes a datalog computation of this kind, e.g. by asking "does Ken belong to { entities allowed to edit X }"?

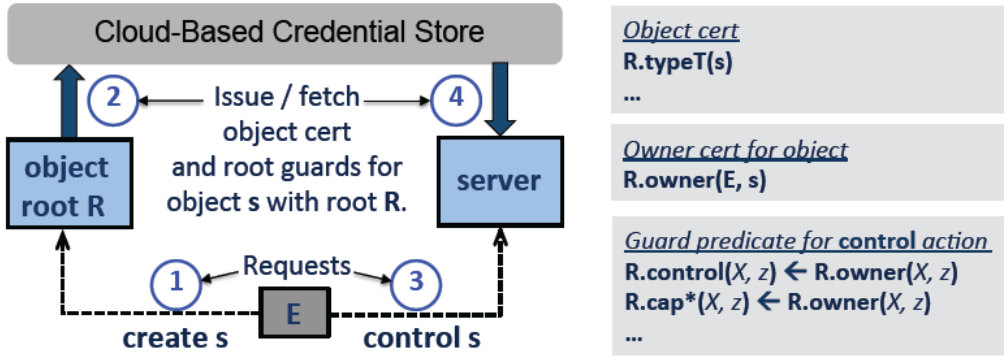
Trust logic syntax

Names Entities: A,B,C,D,E… Objects: r,s,… (or fully qualified name A.r) Quantified variables: entity *X*, object *y*	Entities represent principals and are named by public keys (e.g., A). Each object has a string name (e.g., r) qualified by its issuing **root** entity, e.g., A.r.
Facts **pred**(E, r) **prop**(r, v) 	Facts are first-order predicates over entities, objects, and values. We use only unary and binary predicates.
Beliefs A.**pred**(E, r)	Each fact or predicate is tagged with the entity (e.g., A) that *says* or believes it.
Rules A.**pred**(*X*,*y*) ← B.**pred**(*X*,*y*), A.**type**(*y*,'str')	"If B says **pred**(*X*,*y*) then A believes it if A also believes *y* has property **type**= 'str'."

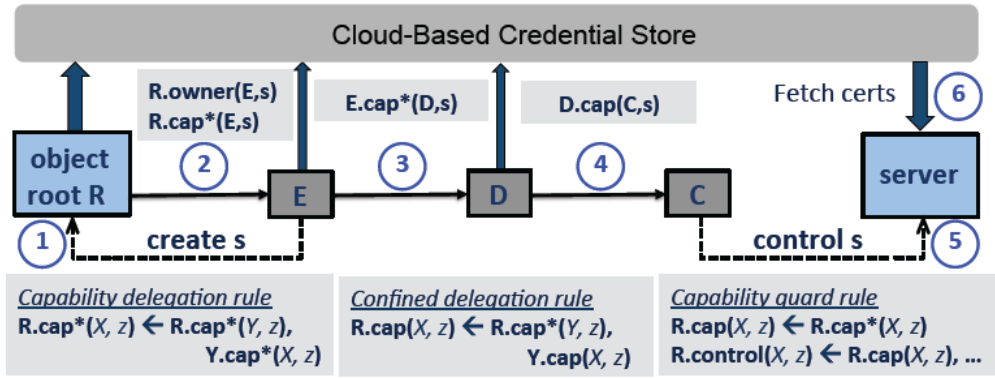
Notation: A.pred(X,y) means "A says that pred holds for entity X, object y.
A.pred*(X,y) means A can delegate whatever "pred" represents

Creation and use of a global object

- Creating an object and using it



- Delegating a capability to perform some action on an object

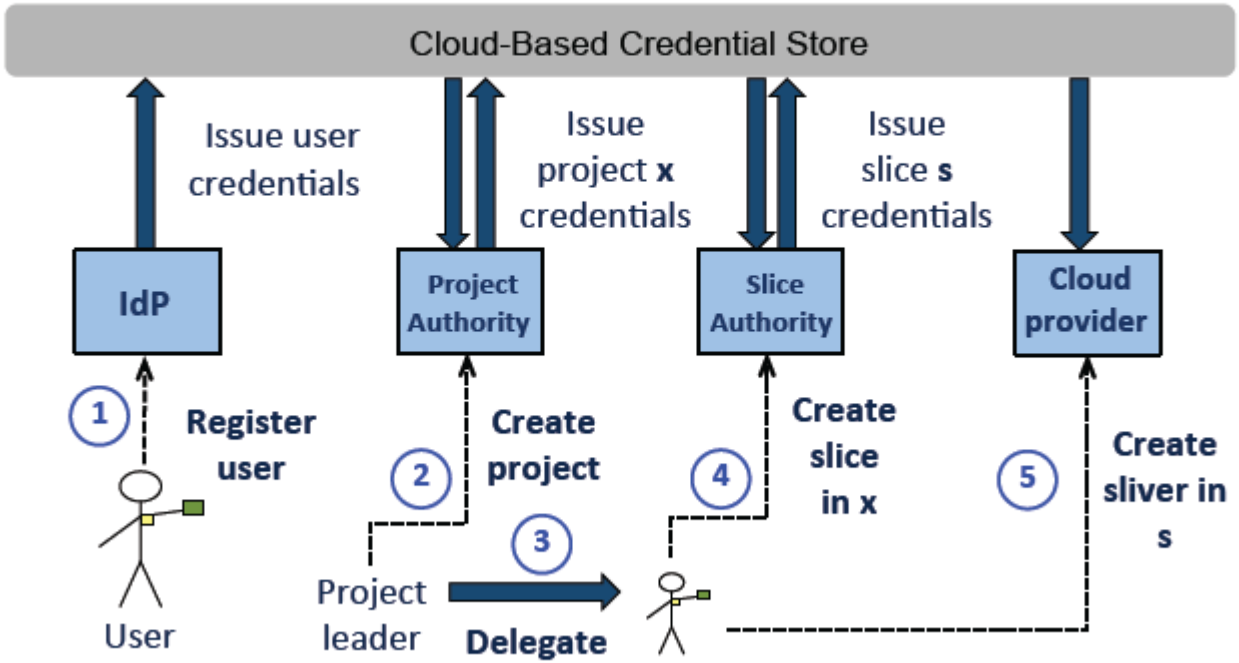


Example: Coordinator entities and credential flow for GENI resource use

- **Identity Provider (IdP)** is any entity that is trusted to assert attributes of a real-world identity without proof. Every GENI user must possess a keypair that is endorsed by a GENI-approved IdP.
- **Project Authority (PA)** approves the creation of projects. A decision to approve a project is based at least in part on validated attributes of the requester. A PA acts as the root entity for projects that it approves: it issues a credential declaring the project and binding it to a name and other attributes. The PA also issues a credential designating the requester as the leader of the new project.
- **Slice Authority (SA)** approves the creation of slices. A decision to approve a slice is based at least in part on validated attributes of the requesting user and the user's association with a project. An SA acts as the root entity for slices that it approves: it issues a credential declaring the slice and binding it to a project, a name, and other attributes. The SA also issues a credential designating the requester as the owner of the slice, conferring a privilege to control the slice.

Example: Coordinator entities and credential flow for GENI resource use

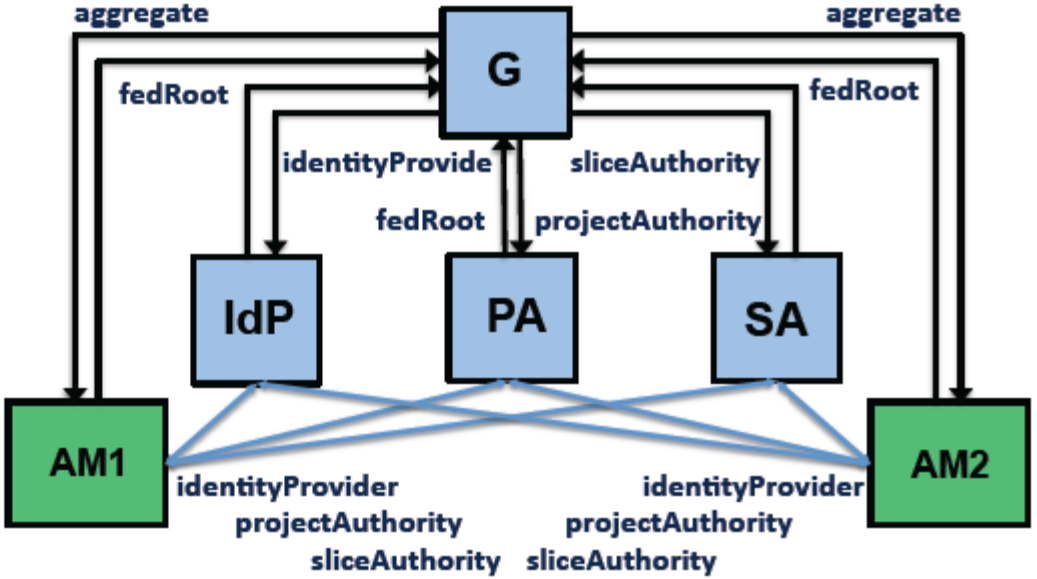
- Each virtual resource instance (“sliver") is obtained from a single resource provider (“aggregate"), and is bound to a project and slice that have been approved by authorities trusted by that aggregate according to its policy.
- Access is fast because projects and slices are coarse-grained objects, and credentials for them are cached at the aggregates.



Trust structure that arises

Fed root endorsements
G.**identityProvider**(IdP)
G.**projectAuthority**(PA)
G.**sliceAuthority**(SA)
G.**aggregate**(AM{1,2})

Fed root acceptance
IdP.**fedRoot**(G)
PA.**fedRoot**(G)
SA.**fedRoot**(G)
AM{1,2}.**fedRoot**(G)



Example policy rule for inferring trust structure
PA.**identityProvider**(*X*) ← PA.**fedRoot**(*Y*), *Y*.**identityProvider**(*X*)

"PA accepts any identity provider endorsed by a federation root."

Summary of policies

Guard policy to create a project PA.**createProject**(*X*) ← PA.**identityProvider**(*Y*), *Y*.**geniUser**(*X*), *Y*.**faculty**(*X*)	"The Project Authority PA approves creation of a project if an identity provider *Y* accepted by PA says that the requester *X* is a **faculty** member who has registered as a **GENI user**."
Guard policy to create a slice SA.**createSlice**(*X*) ← SA.**projectAuthority**(*Y*), *Y*.**project**(*p*), SA.**root**(*Y*, *p*), *Y*.**createSlice**(*X*,*p*)	"The Slice Authority SA approves creation of a slice if a Project Authority *Y* accepted by SA says that the requester *X* has **createSlice** rights in a valid project *p*, where *Y* is the **root** of *p*."
Guard policy to create a slice in a project PA.**createSlice**(*X*, *p*) ← PA.**owner**(*X*,*p*)	"The Project Authority PA approves creation of a slice in a project *p* if the requester *X* is the leader/**owner** of *p*."

Li/Mitchell/Winsborough

- This paper provides the mathematic framework on which ORCA was implemented

- Defines the full identity, role and trust framework
  - Offers a careful logic semantics for each action
    - Expressed as a base logic, RT_0, RT_1, RT_2, RT^D and RT^T
      - Each builds on the prior one (without breaking it), typically by showing how a construct can "map" to the more basic construct
      - For example, RT_1 adds parametized roles to RT_0 using a scheme a bit like macro expansion in a language like C
  - Shows that the resulting scheme is highly expressive and decidable: any expressed policy can be decided using a fast procedure that encodes nicely into Datalog

Summary and conclusions

- GENI/ORCA exemplify a trend
  - A general move to offer more powerful security languages in systems
  - These permit extremely flexible, expressive policies but they also bring complexity
  - Implementation centers on cryptographic "identity" combined with proofs
    - But deduction is slow, hence use capabilities as a kind of pre-certified proof
    - Even so, speed of ORCA is not discussed in Chase's paper