GENI Distributed Services Preliminary Requirements and Design

Tom Anderson and Amin Vahdat (co-chairs)
David Andersen, Mic Bowman, Frans Kaashoek, Arvind Krishnamurthy, Yoshi Kohno, Rick McGeer, Vivek Pai, Mark Segal, Mike Reiter, Mothy Roscoe, Ion Stoica

Distributed Services Work Status

Work split into subgroups:

- Security Architecture (Mike Reiter, Tom Anderson, Yoshi Kohno, Arvind Krishnamurthy)
- Edge cluster definition (Mic Bowman, Tom Anderson)
- Storage (Frans Kaashoek, Dave Andersen, Mic Bowman)
- Resource Allocation (Amin Vahdat, Rick McGeer)
- Experiment Support (Amin Vahdat, Arvind Krishnamurthy)
- Operations Support (Mark Segal, Vivek Pai)
- Communications Substrate (Arvind Krishnamurthy, Amin Vahdat, Tom Anderson)
- Legacy Systems (Tom Anderson)

Distributed Services Work Status

Each section progressed against a defined sequence:

- Overview
- Requirements description
- Preliminary design
- Related work discussion
- Modules and dependencies identified
- WBS estimate

Every part of the design subject to change as science goals are refined, additional information gathered

- Including during construction

Distributed Services Work Status

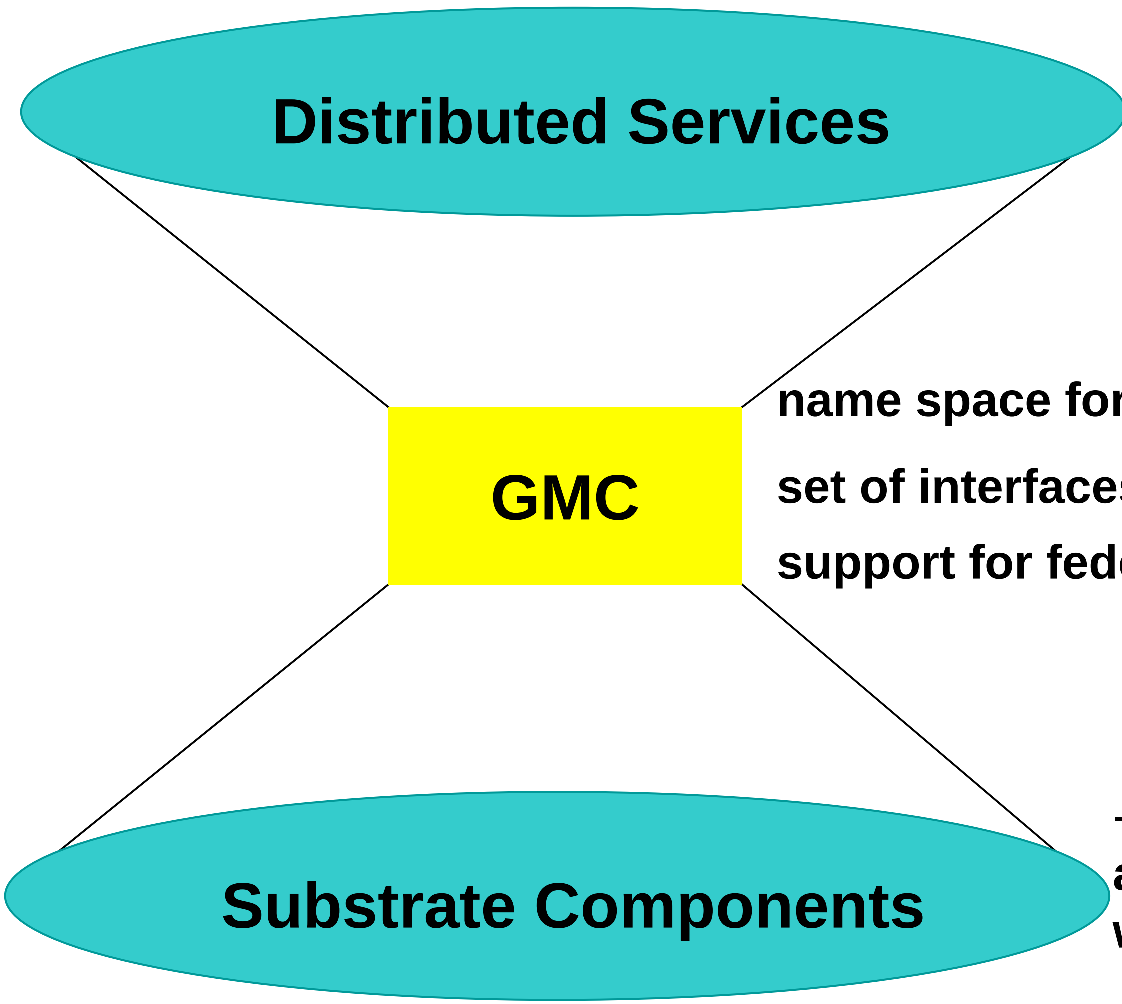
Overall state:

- Rationale/design needs better documentation and an independent review
- Modules identified/initial WBS estimates completed
- Need clarity from the GSC as to prioritization

Specifics

- Security: Design solid; user scenarios needed
- Edge Node/Cluster: Requirements in flux depending on budget issues; moved to GMC
- Storage: Requirements solid; modules identified
- Resource allocation: Design solid; user scenarios needed
- Experimenter support: User experience needed to drive requirements
- Operations support: Requirements outlined
- Communication Services: Requirements outlined
- Legacy Support: Requirements outlined

Facility Software Architecture



achieve system-wide properties such as security, reproducibility, ..

name space for users, slices, & components

set of interfaces (“plug in” new components)

support for federation (“plug in” new partners)

- provide ability to virtualize and isolate components in a way meaningful to expts

Facility Software Architecture

At hardware device level, component manager, virtualization and isolation layer

Minimal layer (GENI management core) to provide basic building blocks

- Robustness of this layer is critical to the entire project, so keep it small, simple, well-defined
- Avoid “big bang” integration effort

Services layered on top of GMC to provide system-level requirements

- Modular to allow independent development and evolution
- As technology progresses, post-GENI efforts can replace these services piece by piece

User Centric View

- Researchers
  - Ease of describing, launching, and managing experiments
  - Network-level, not node-level
- Operations staff
  - Administrative cost of managing the facility
- Resource owners (hardware contributors)
  - Policy knobs to express priorities, security policy for the facility
- System developers (software contributors)
  - GENI developers and the broader research community building tools that enhance GENI
- End users
  - Researchers and the public

Goal of distributed services group is to make the system more *useful*, not more *powerful*

Principal Concerns

- Security and isolation
- Operational cost and manageability
- Usability and experiment flexibility
- Scalability, robustness, performance
- Experiment development cost
- Construction cost and schedule
- Policy neutrality: avoid binding policy decisions into GENI architecture

Topics

- Security architecture
- Edge cluster hardware/software definition
- Storage services
- Resource allocation
- Experiment support
- Operations support
- Communications substrate
- Legacy Internet applications support

Security Architecture

- What is the threat model?
- What are the goals/requirements?
- Access control
- Authentication and key management
- Auditing
- Operator/administrative interfaces

Threat model

- Exploitation of a slice
  - Runaway experiments
    - Unwanted Internet traffic
    - Exhausting disk space
  - Misuse of experimental service by end users
    - E.g., to traffic in illegal content
  - Corruption of a slice
    - Via theft of experimenter's credentials or compromise of slice software
- Exploitation of GENI itself
  - Compromise of host O/S
  - DoS or compromise of GENI management infr

Requirements: Do no harm

- Explicit delegations of authority
  - Node owner → GMC → Researcher → students → …
- Least privilege
  - Goes a long way toward confining rogue activities
- Revocation
  - Keys and systems will be compromised
- Auditability
- Scalability/Performance
- Autonomy/Federation/Policy Neutrality
  - Control ultimately rests with node owners, can delegate selected rights to GMC

Modeling Access Control in Logic

Expressing Beliefs:

`Bob says F`

– It can be inferred that Bob believes that *F* is true

`Bob signed F`

– Bob states (cryptographically) that he believes that *F* is true

Types of Beliefs:

`Bob says open(resource, nonce)`

– Bob wishes to access a resource

`Bob says (Alice speaksfor Bob)`

– Bob wishes to delegate all authority to Alice

`Bob says delegate(Bob, Alice, resource)`

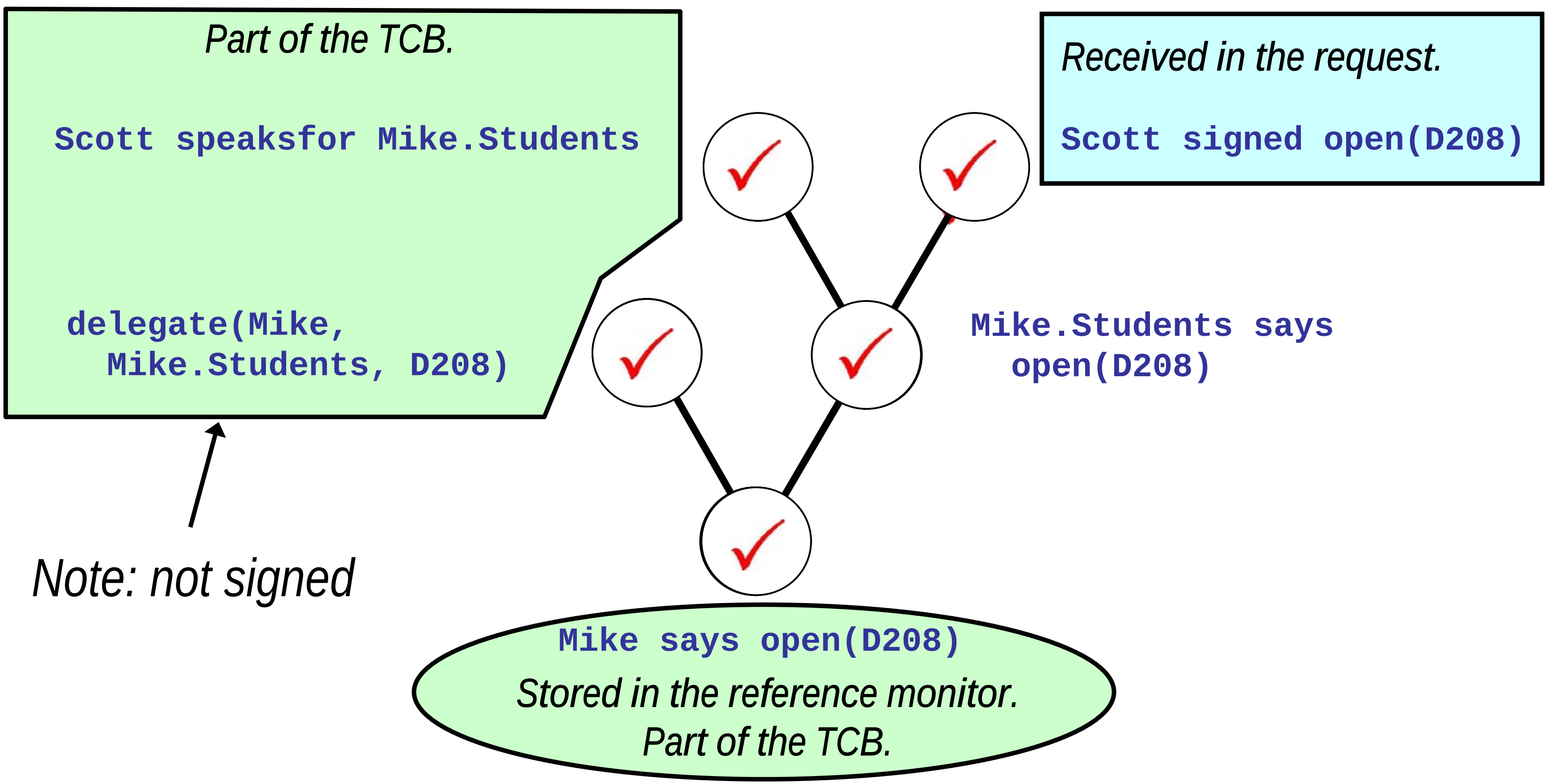
– Bob wishes to delegate authority over a specific resource to Alice

Inference Rules (examples):

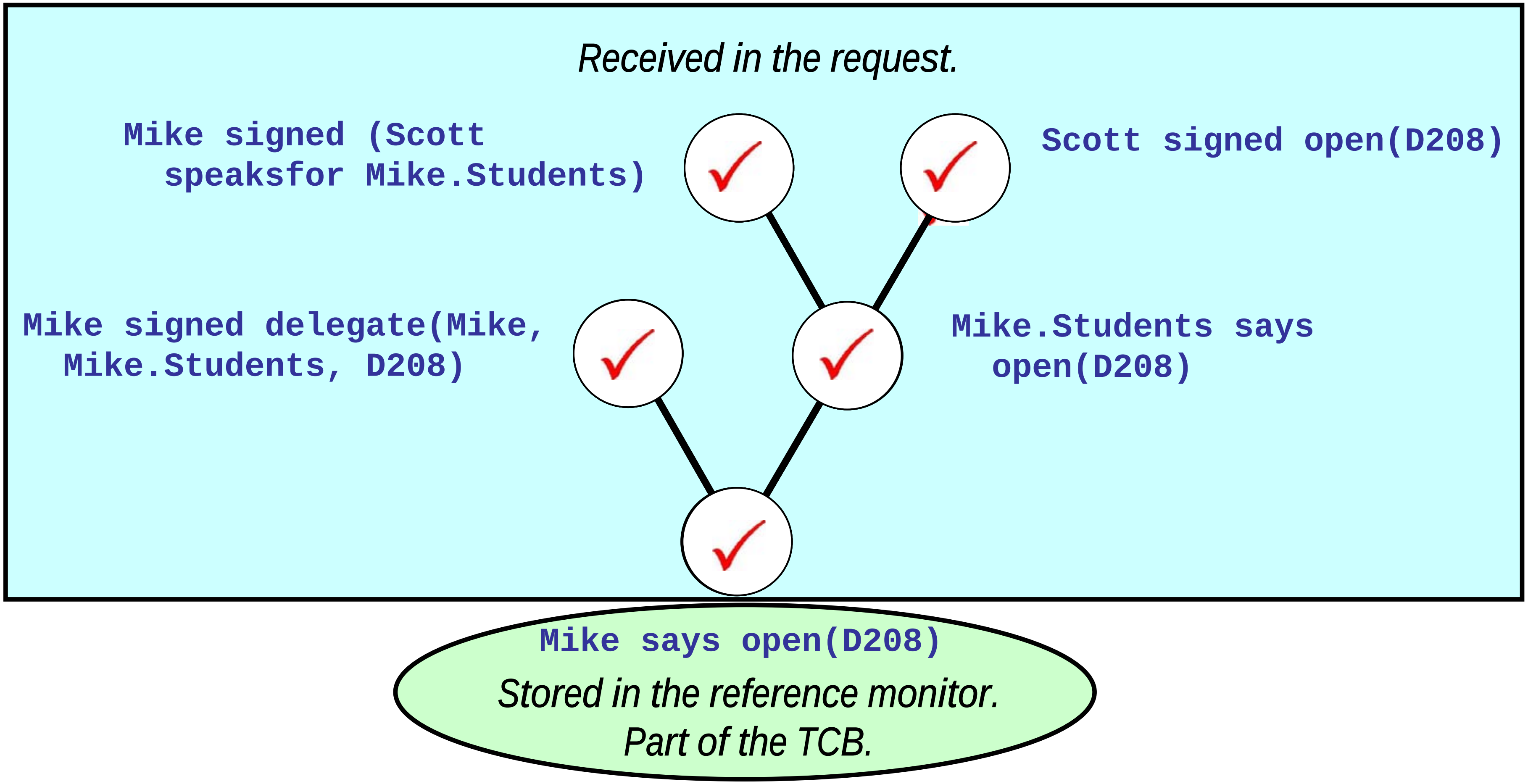
$$\frac{\texttt{A says (B speaksfor A)} \qquad \texttt{B says F}}{\texttt{A says F}}\ \texttt{speaksfor-e} \qquad\qquad \frac{\texttt{A signed F}}{\texttt{A says F}}\ \texttt{says-i}$$

Proofs: Sequence of inference rules applied to beliefs

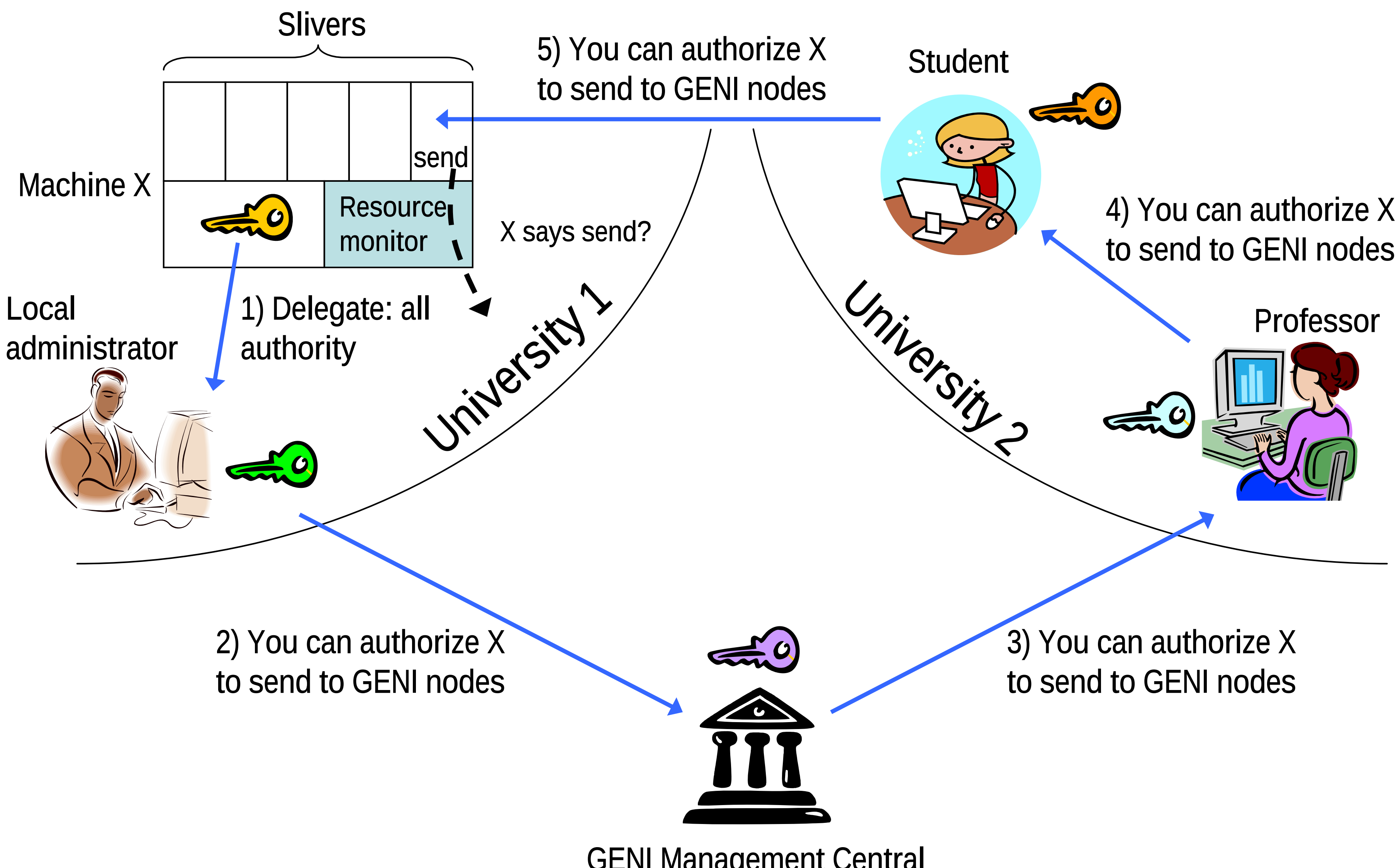
Traditional Access Control Lists



A “Proof Carrying” Approach



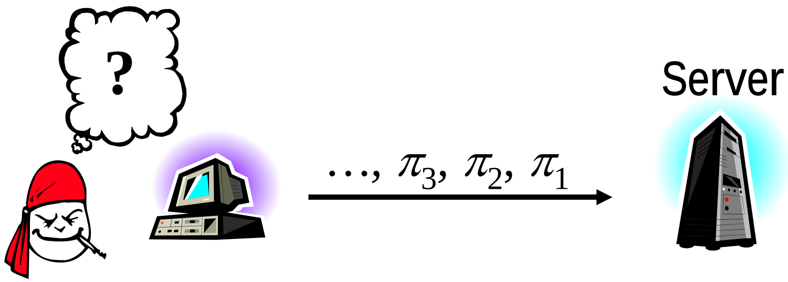
Authorization Example (simplified)



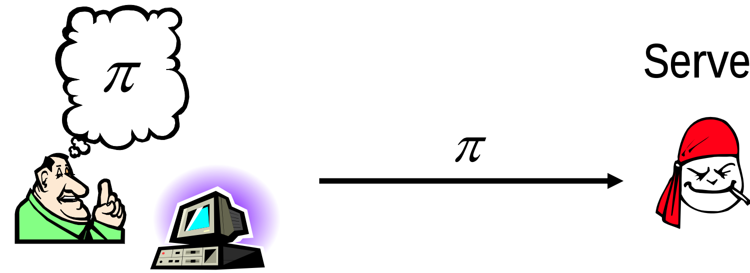
Authentication and key management

- GENI will have a PKI
  - Every principal has a public/private key
    - E.g., users, administrators, nodes
  - Certified by local administrator
  - Keys sign certificates to make statements in formal logic (identity, groups, authorization, delegation, …)
- Private key compromise an issue
  - Encrypted with user’s password? Off-line attacks
  - Smart card/dongle? Most secure, but less usable
  - Capture-resilient protocols: A middle ground

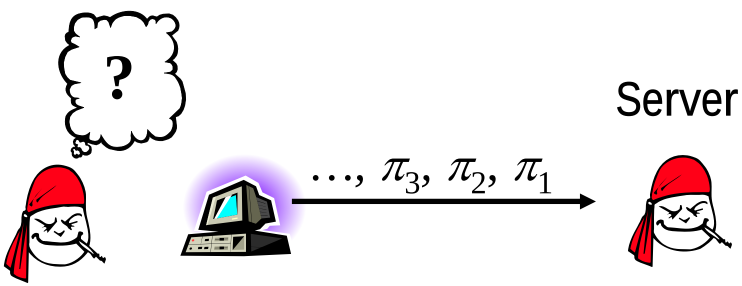
Capture-Resilience Properties



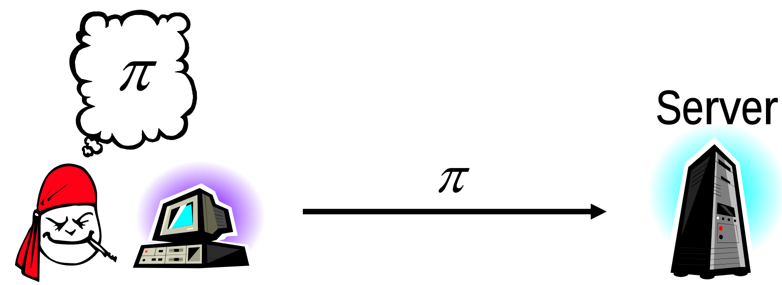
Attacker must succeed in online dictionary attack



Attacker gains no advantage

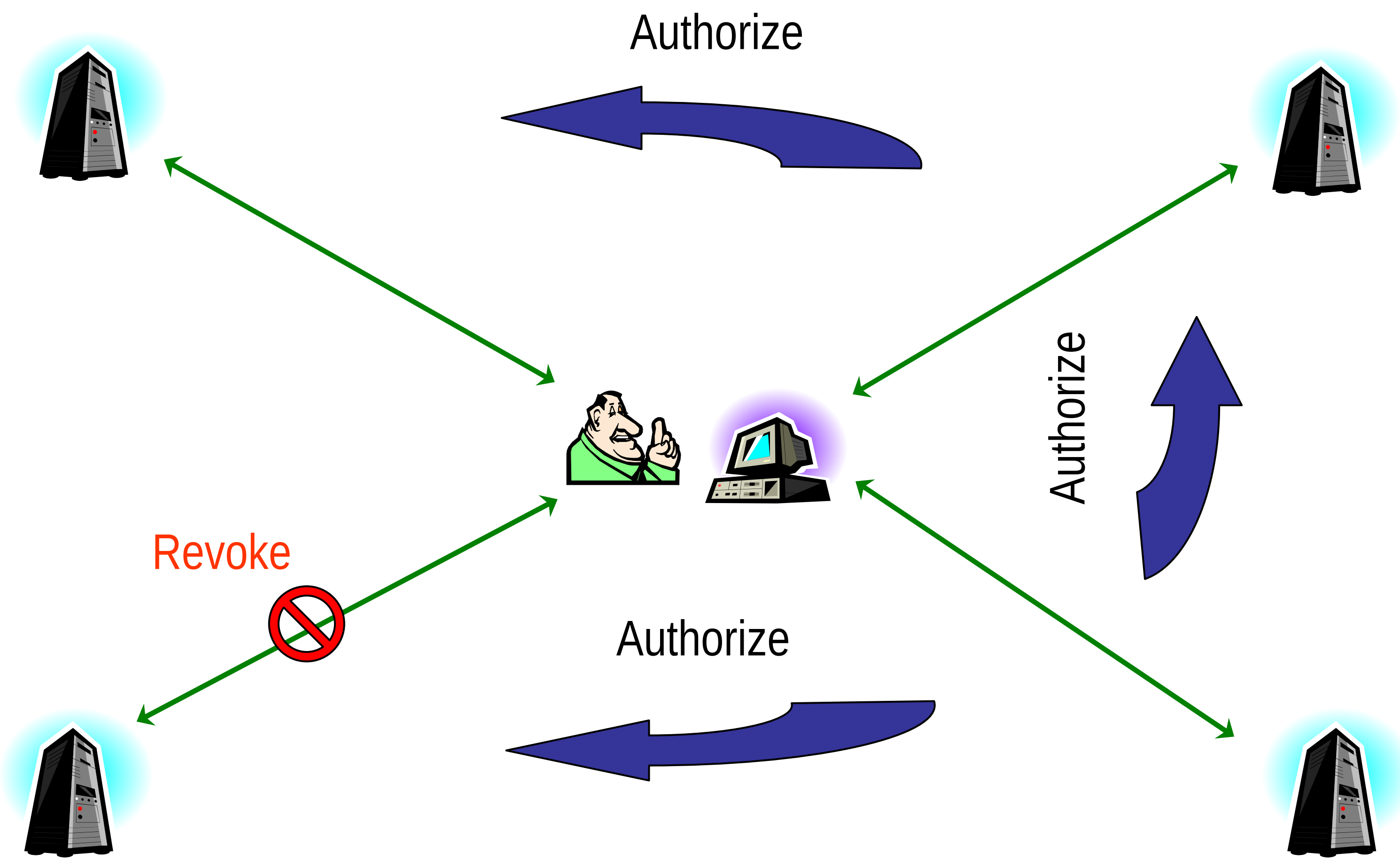


Attacker must succeed in offline dictionary attack



Attacker can forge only until server is *disabled* for device

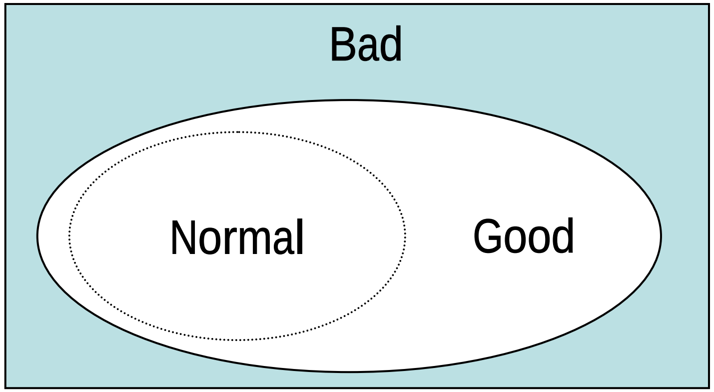
Delegation in Capture-Protection



Intrusion Detection

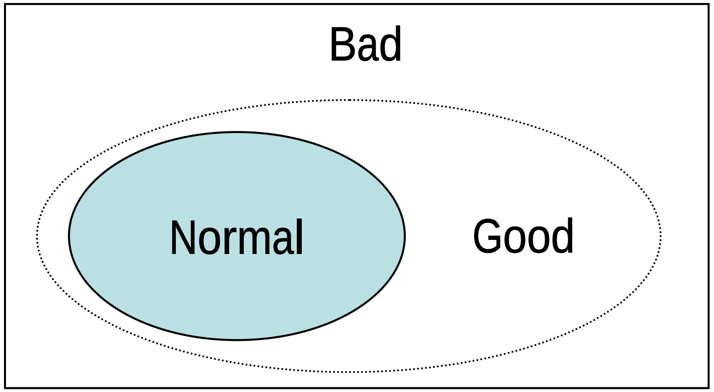
- Traditional intrusion detection methods may not suffice *for monitoring experiments*

<u>Misuse detection</u>
Specify bad behavior and watch for it



Problem: Experiments do lots of things that look “bad”

<u>(Learning-based) Anomaly detection</u>
Learn “normal” behavior and watch for exceptions



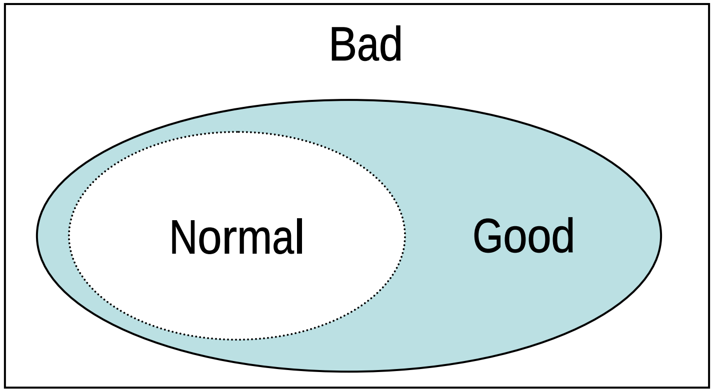
Problem: Experiments may be too short-lived or ill-behaved to establish “normal” baseline

Intrusion Detection

- Specification-based intrusion detection is more appropriate for monitoring experiments
  - Fits in naturally with authorization framework, as well

<u>Specification-based intrusion detection</u>

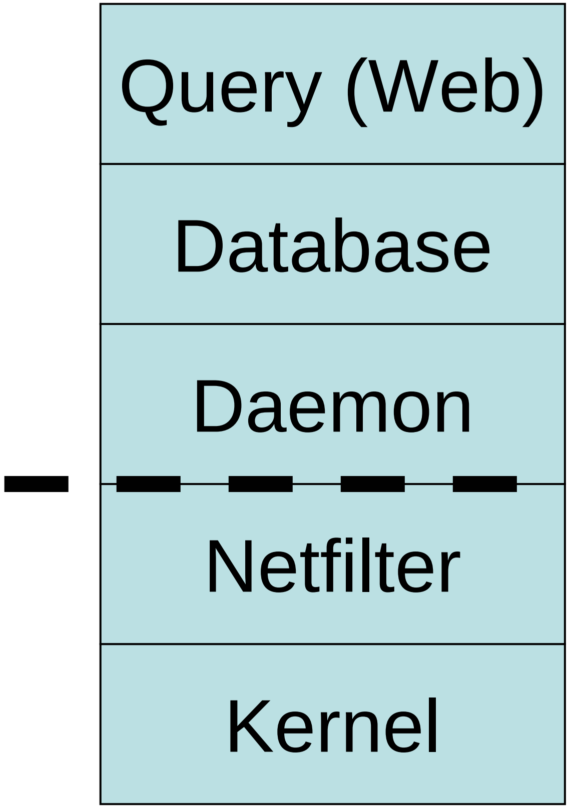
Specify good behavior and watch for violations



Audit Log Example: PlanetFlow

- PlanetFlow: logs packet headers sent and received from each node to Internet
  - Enables operations staff to trace complaints back to originating slice
  - Notify experimenter; in an emergency, suspend slice
- All access control decisions can be logged and analyzed post-hoc
  - To understand why a request was granted (e.g., to give attacker permission to create a sliver)
  - To detect brute force attacks

Packet Logging Architecture



CGI → SQL query

MySQL, etc

Packet headers → sessions

Packet headers → batch

Divert packets

Performance

- Straightforward approach
  - 2.5% of CPU; < 1% of bandwidth
- Modifications
  - Group sessions in kernel
  - Lazily add to database
  - Eliminate intra-GENI traffic
  - Limit senders if auditing too expensive
- 10 Gbps?
  - Large flows easy, small flows even realistic?

Security Deliverables (21E)

1. Definition of certificate format and semantics (2E)
2. Certificate mgmt svc (construction, storage, lookup and caching) (5E)
3. Access control guard (resource monitor) (2E)
4. Security policy language and certificate revocation, and UI (3E)
5. Secure and reliable time service (purchase)
6. Proof generator (2E)
7. Specification-based intrusion detection service (5E)
8. Capture protection server and client software (2E)

#E represents estimate in developer-years, assuming a five year construction span, excluding management, system test, overhead, and risk factor

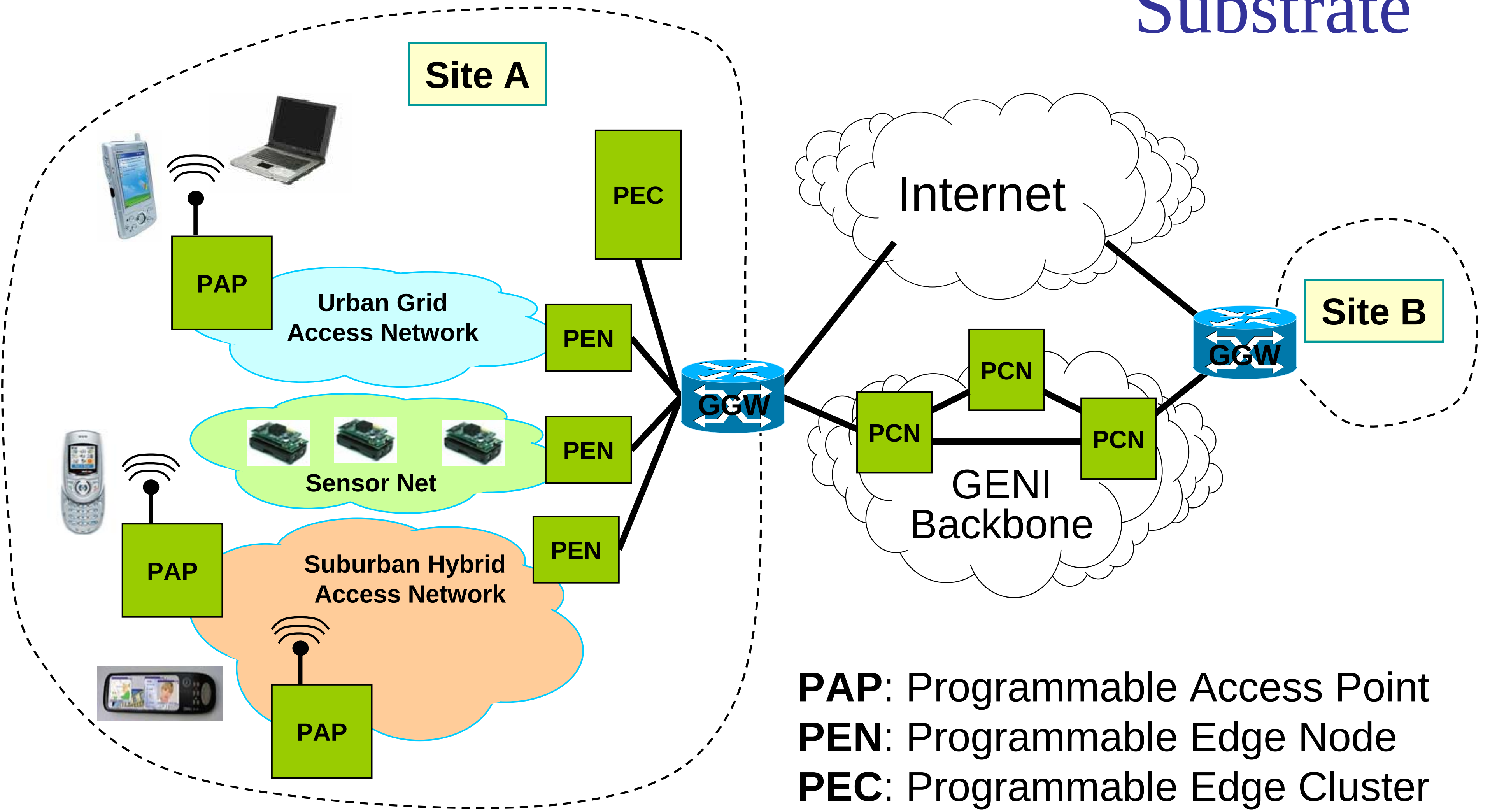
Security: Open Issues

- DoS-resistant GENI control plane?
  - Initial control plane will employ IP and inherit the DoS vulnerabilities thereof
  - GENI experimentation may demonstrate a control plane that is more resistant
  - Design proof-carrying certificates to operate independently of communication channel
- Privacy of operational data in GENI?
- Operational procedures and practices
  - Central to security of the facility

Topics

- Security architecture
- Edge cluster hardware/software definition
- Storage services
- Resource allocation
- Experiment support
- Operations support
- Communications substrate
- Legacy Internet applications support

Example Substrate



PAP: Programmable Access Point
PEN: Programmable Edge Node
PEC: Programmable Edge Cluster
PCN: Programmable Core Node
GGW: GENI Gateway

Programmable Edge Cluster: HW

Capabilities should be driven by science plan

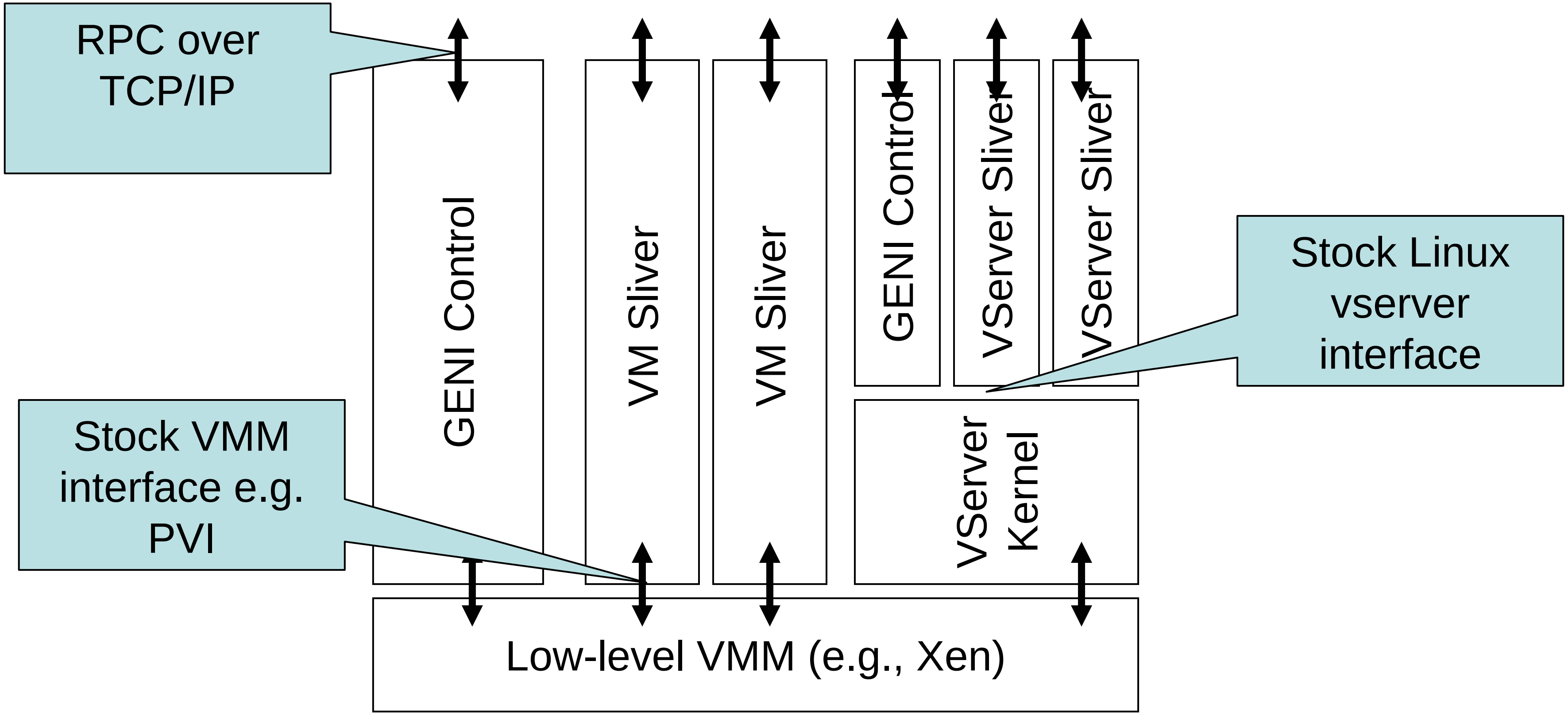
Draft:

- 200 sites, cluster of 10-20 PCs at each
  - Workhorse nodes: running experiments, emulating higher speed routers, distributed services
  - Multicore CPU, 8GB of DRAM, 1TB disk, gigE
  - High speed switch and router connecting to rest of GENI

Cut in latest iteration of draft plan:

- 20 sites, cluster of 200 PCs each
  - Compute/storage intensive applications

Programmable Edge Cluster: SW



Experiments run as a vserver sliver or as a VM sliver
Communicate with GENI management code (running as sliver) through RPC

Execution environments

- PlanetLab-like best-effort VServers
  - Fixed kernel, convenient API
  - Weak isolation between slivers
  - Weaker security for critical components
  - Small number of standard configurations
    - minimal, maximal, expected
- Virtual machine monitors (e.g., Xen, VMWare)
  - Choice of prepackaged or custom kernels (as in Emulab)
    - Linux + click
    - Others possible: singularity, windows, raw click
  - Stronger resource guarantees/isolation
  - poor I/O performance
  - limited number of VMs (scalability)

Service Location

Services can be implemented at any of a set of levels:

- Inside VMM
  - if kernel changes are required
  - e.g., to implement fast segment read/write to disk
- In its own VM on the VMM
  - To configure VMM, or if security is needed
  - E.g., the GENI component manager; GENI security monitor
- In the linux vserver
  - If linux kernel changes are needed, e.g., traffic monitoring
- In its own vserver sliver
  - Running as a best effort service, e.g., vserver component manager
- In a library linked with experiment
  - E.g., database, cluster file I/O

Booting

- To boot a node
  - Trusted computing hardware on each node
  - Secure boot fetches initial system software
  - Initial machine state eventually comprises:
    - Virtual Machine Monitor (e.g. Xen)
    - Initial domain: GENI Domain (GD).
    - Possibly VServer kernel by default
- To boot a sliver
  - Send authorized request to GENI Domain
  - GD verifies request; creates new xen/vserver domain
  - Loads software that contains sliver secure boot (GENI auth code, ssh server, etc.)
  - See reference component design document for details

Containment & Auditing

- Limits placed on slice “reach”
  - restricted to slice and GENI components
  - restricted to GENI sites
  - allowed to compose with other slices
  - allowed to interoperate with legacy Internet
- Limits on resources consumed by slices
  - cycles, bandwidth, disk, memory
  - rate of particular packet types, unique addrs per second
- Mistakes (and abuse) will still happen
  - auditing will be essential
  - network activity $\longrightarrow$ slice $\longrightarrow$ responsible user(s)

Edge Cluster WBS Deliverables

- See GMC specification

Open Questions

- Resource allocation primitives on each node
  - Reservation model:
    - % CPU in each a given time period?
    - Strict priorities?
  - What about experiments/services whose load is externally driven (e.g., a virtual ISP)?
  - Other resources with contention: memory, disk
    - Fine-grained time-slicing of disk head with real time guarantees is unlikely to work as intended
    - Either best effort, or disk head per application (means we need at least k+1 disk heads for k disk intensive applications)
  - How are service-specific resources represented (e.g., segment store)?
  - How are resources assigned to services? Through experiments giving them resources explicitly, or via configuration?

More Open Questions

- Kernel changes needed in xen, vservers to implement resource model
  - Custom GENI OS to run on xen?
- Allocation of IP address/port space to slivers
  - well known ports
- Efficient vserver sliver creation
  - Configure new sliver (e.g., to run a minimal script) with minimal I/O overhead, minimal CPU time
  - Can/should vservers run diskless?
  - Make it easy to share file systems read only
  - Vserver image provided by symlinks or NFS loopback mounts?

More Open Questions

What is the agreement with hosting sites?

- Rack space
- IP address space (\24 per site?)
- Direct connectivity to Internet
- BGP peering?
- Bandwidth to Internet?
- Local administrative presence?
- Ability to add resources under local control?
- Absence of filtering/NATs

Topics

- Security architecture
- Edge cluster hardware/software definition
- Storage services
- Resource allocation
- Experiment support
- Operations support
- Communications substrate
- Legacy Internet applications support

Storage for GENI

- Enables future network applications, which integrate storage, computation, and communication
  - Large-scale sensor networks
  - Digital libraries that store all human knowledge
  - Near-on-demand TV
- Experiments also need storage:
  - Experiment results
  - Logs (e.g., complete packet traces)
  - Huge data sets (e.g., all data in Web)
  - Running the experiment (binaries, the slice data, etc.)
- Managing GENI requires storage:
  - Configuration
  - Security and audit logging
- Storage will be distributed and shared

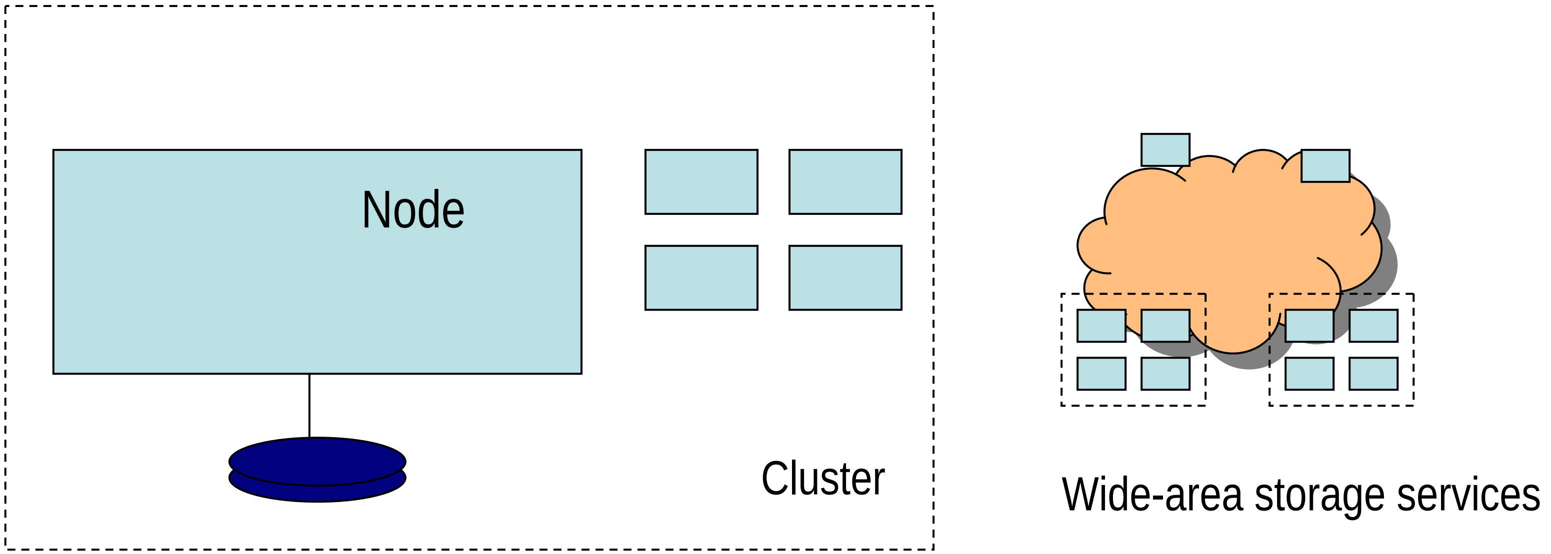
Storage Goals

1. Enable experiments that integrate computation and storage
    - Provide sufficient storage (e.g., 200 Petabytes)
    - Provide convenient access to the storage resources
    - Provide high performance I/O for experiments
2. Allow the storage services to evolve
    - Allow experimenters to build new storage services
    - Balance expectation of durability
3. Permit effective sharing of storage resources
    - User authentication and access control
    - Resource control

Overall Storage Design

- Node-level storage building blocks
- Higher level distributed storage abstractions

Dependencies on authorization, resource management



Node-level Storage Support

- Convenient access for experimenters & admins
  - File system interface
  - SQL database on node (likely)
    - Needed by many apps + GENI managers
    - e.g., auditing system
- Extensible access for service creators
  - Raw disk / block / extent store
    - Direct access for building services
  - "Loopback" filesystem support
    - Facilitate creating distributed storage services
  - Efficient use of disk bandwidth

Distributed Storage Support

- Consolidate frequently used data management services

- Convenient administration and experimentation
  - Transparent wide-area file system
  - Data push services: install data "X" on 200 nodes
  - Log storage and collection for research and management
- High performance distributed I/O
  - e.g., an improved Google File System (cluster)
  - ok to compromise on application-level transparency
  - Possible high-peformance wide-area filesystem
  - Write-once, global high-peformance storage
- Storage for constrained nodes (e.g., sensors)

Storage Deliverables (28E)

1. Local filesystem interface (1E)
2. SQL database (1E)
3. Services for creating new storage services and intercepting storage system calls (1E)
4. Raw disk interface (3E)
5. Block-based storage interface (3E)
6. A wide-area filesystem for administration and experiment management (4E)
7. A high-performance cluster filesystem (4E)
8. Fast write-once/read only storage services. (3E)
9. A reduced complexity storage interface for constrained nodes. (3E)
10. Maintenance and bug fixing throughout life cycle. (5E)

Topics

- Security architecture
- Edge cluster hardware/software definition
- Storage services
- Resource allocation
- Experiment support
- Operations support
- Communications substrate
- Legacy Internet applications support

Resource Allocation Goals

- Define framework for expressing policies for sharing resources among global participants
- Design mechanisms to implement likely policies
- Resource allocation mechanisms should
  - provide resource isolation among principles
  - be decentralized
    - support federation and local site autonomy
  - be secure
  - provide proper incentives
    - incentive for participants to contribute resources to the system and to keep them up and running
    - incentive for participants to use only as much resources as they really need

Existing Resource Allocation Model

- Existing model for PlanetLab resource allocation
  - all resources placed in a central pool
  - all users compete for all resources
- Pros:
  - simple, no complex policy
  - well understood, tried and true time sharing
- Downsides:
  - no incentive for anyone to add additional resources
  - no incentive to keep local machines up and running
  - no incentive for anyone to use less than "as much as possible"
  - all best-effort—can't reserve fixed share of a node

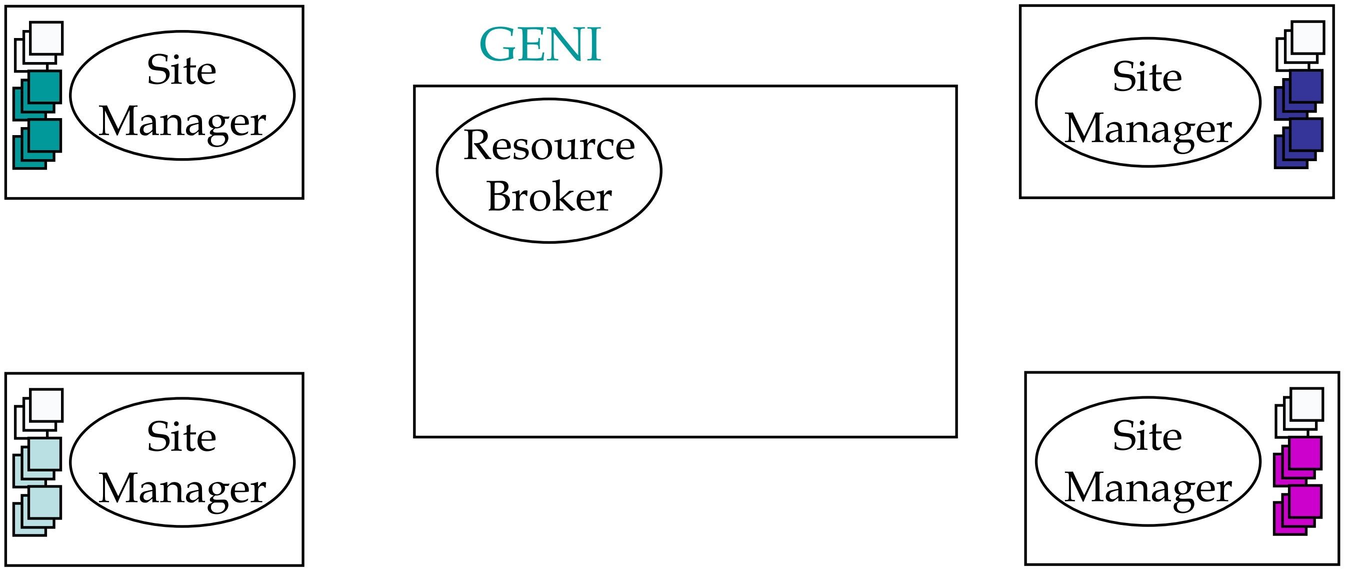
Example Allocation Policies

- All resources placed in central pool [Supercomputer Center]
- Portion of resources reserved for dedicated use [SIRIUS]
- Portion of resources available for bidding [Bellagio]
- Pair-wise resource peering [SHARP]

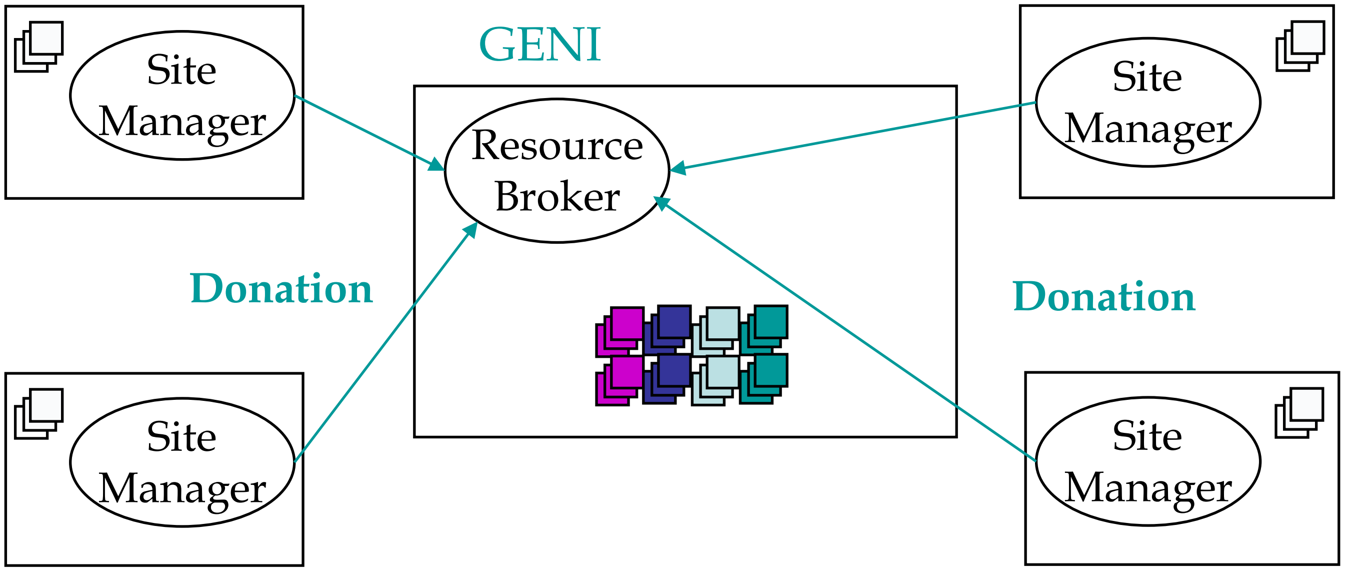
Resource Allocation Proposal

- Three pieces
  - GMC runs a centralized **Resource Broker** (RB)
  - Each site runs a **Site Manager** (SM)
  - Each component (e.g. node) runs a **Component Manager** (CM)
- Site donates some portion of its resources to GENI
- Site's SM receives a Token of value proportional to value of resources contributed
  - SM subdivides Token among site users
- To access a resource
  - User presents token + resource request to RB
  - RB returns Ticket (a lease for access to requested resource)
  - User presents Ticket to resource's CM to obtain sliver
- "Back door": GENI Science Board can directly issue Tokens and Tickets to users and sites

GENI Resource Allocation

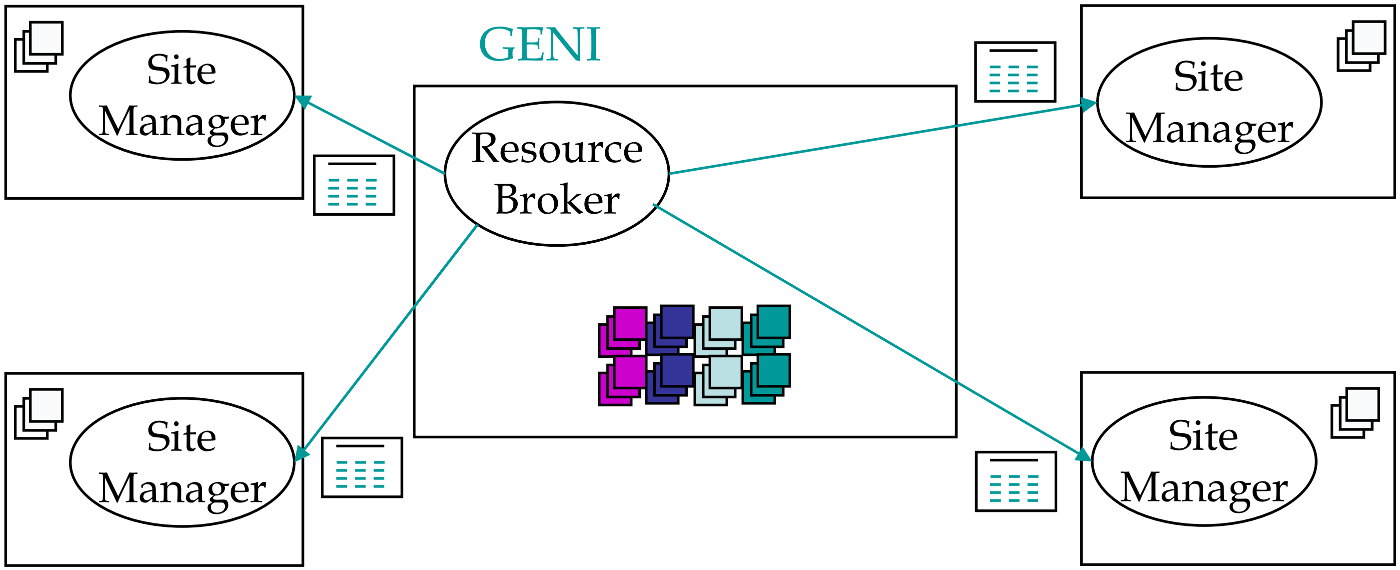


GENI Resource Allocation



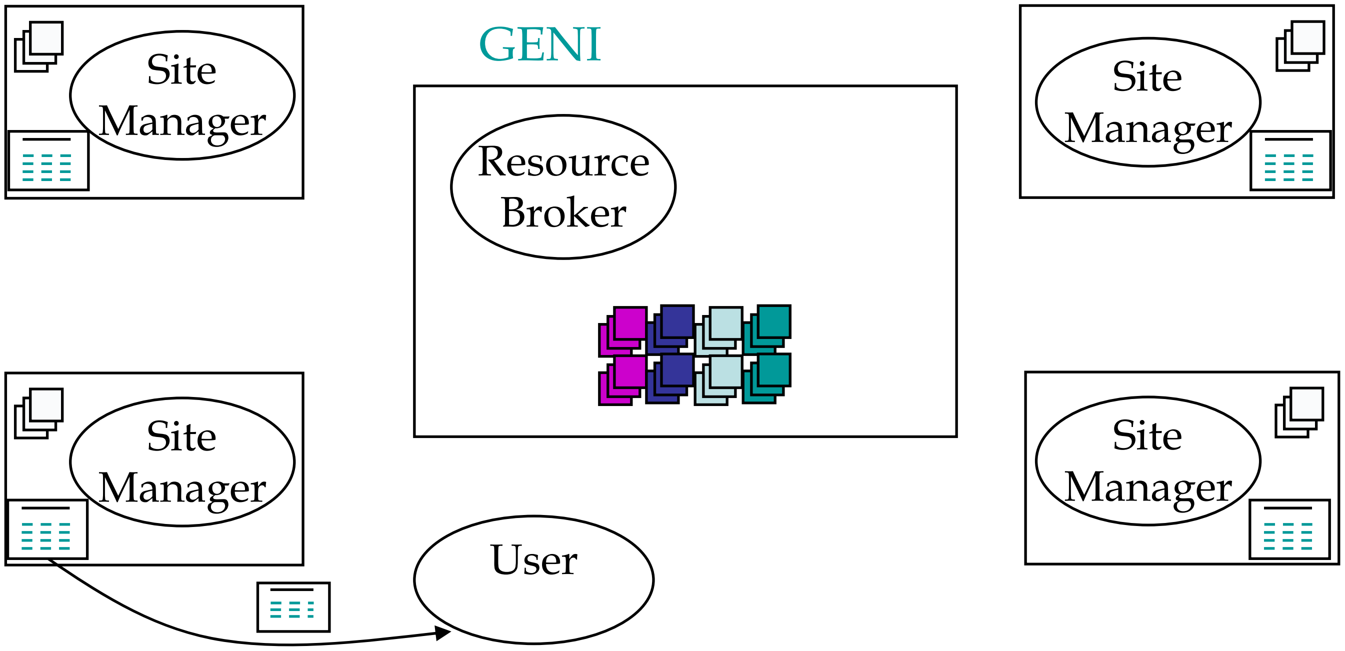
- Sites donate some portion of resources to GENI

GENI Resource Allocation



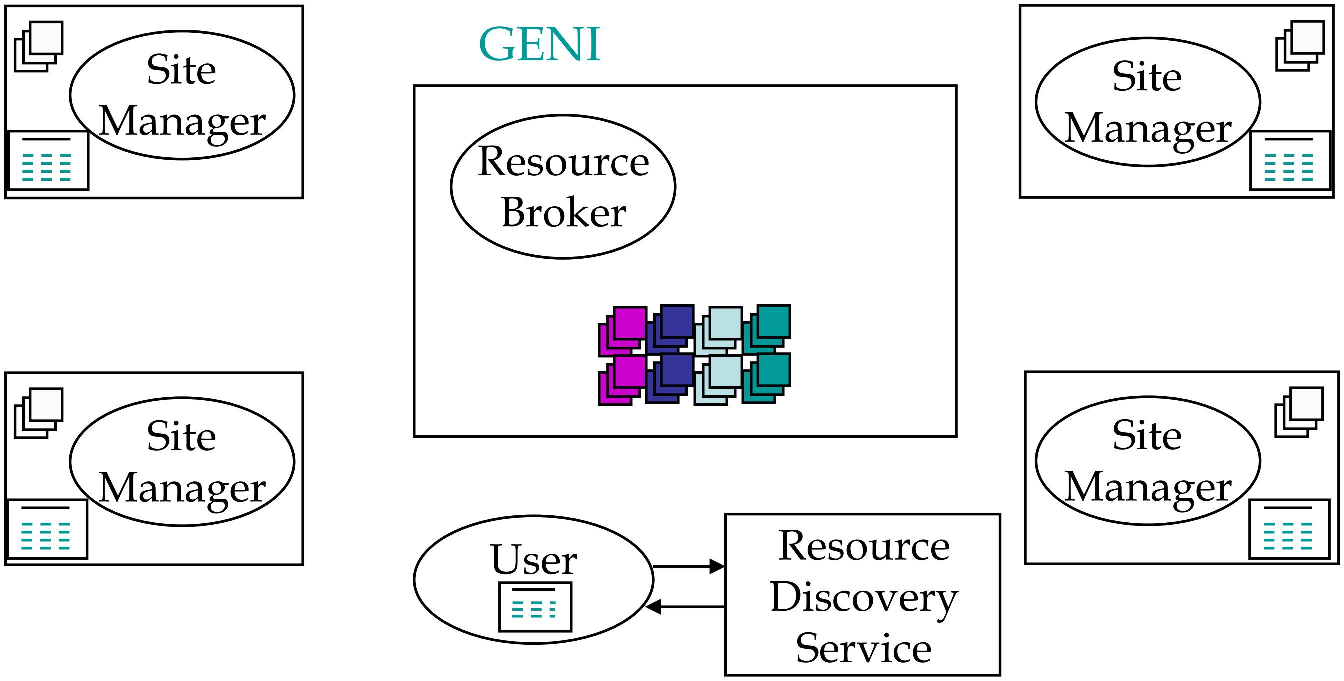
- In exchange, GENI issues Tokens to each site with value proportional to that of donated resources
  - each token carries a value, so 10 tokens of value 1 are equivalent to 1 token of value 10
  - any principal can subdivide tokens

GENI Resource Allocation



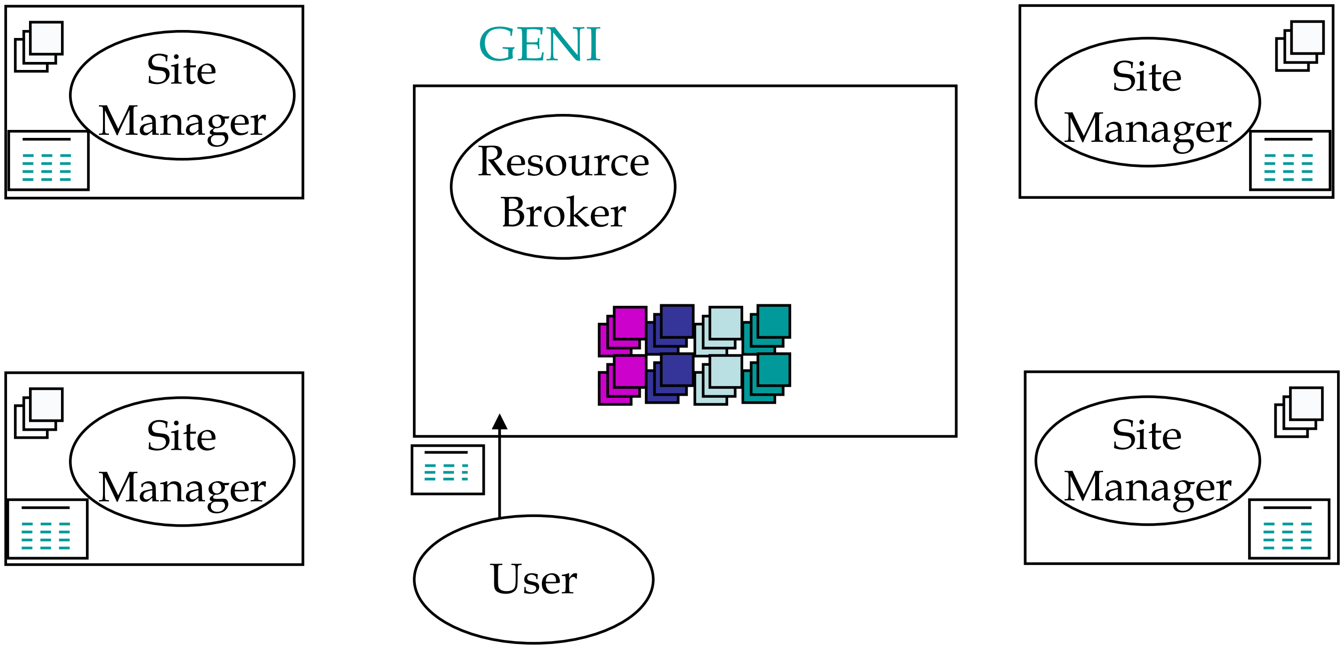
- Site Manager delegates some resource privileges to user by issuing a Token of smaller denomination

GENI Resource Allocation



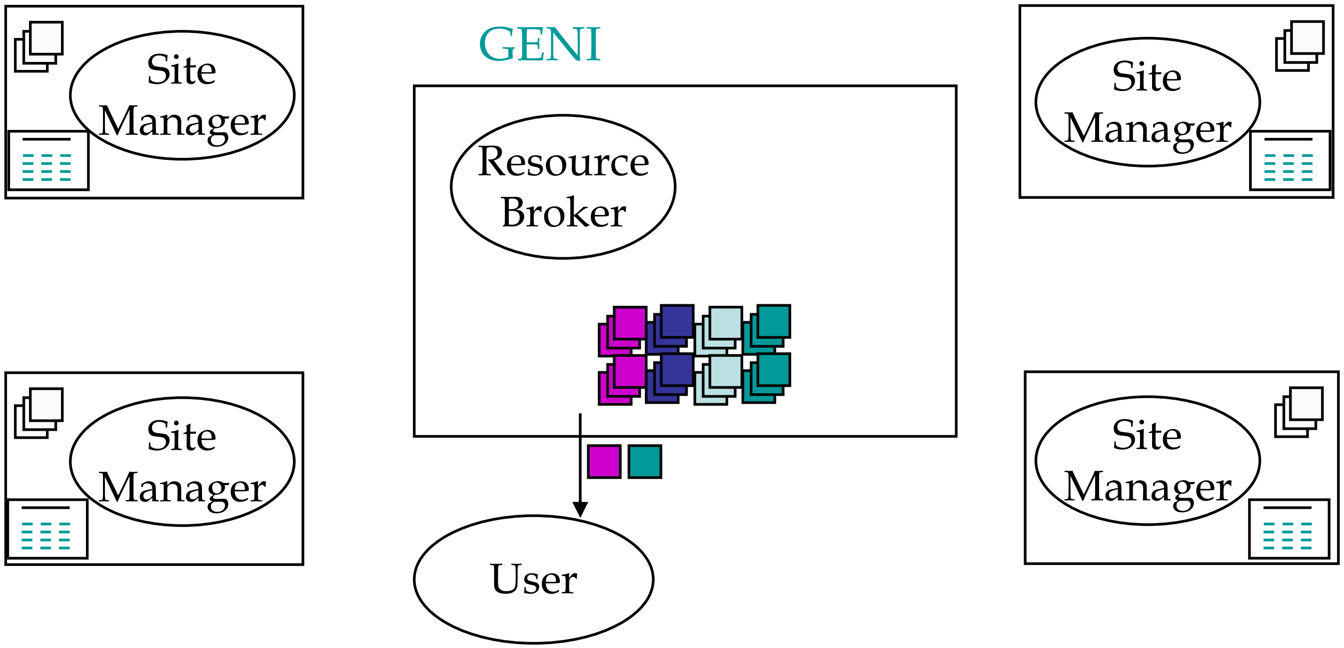
- User consults any resource discovery service to locate desired resources

GENI Resource Allocation



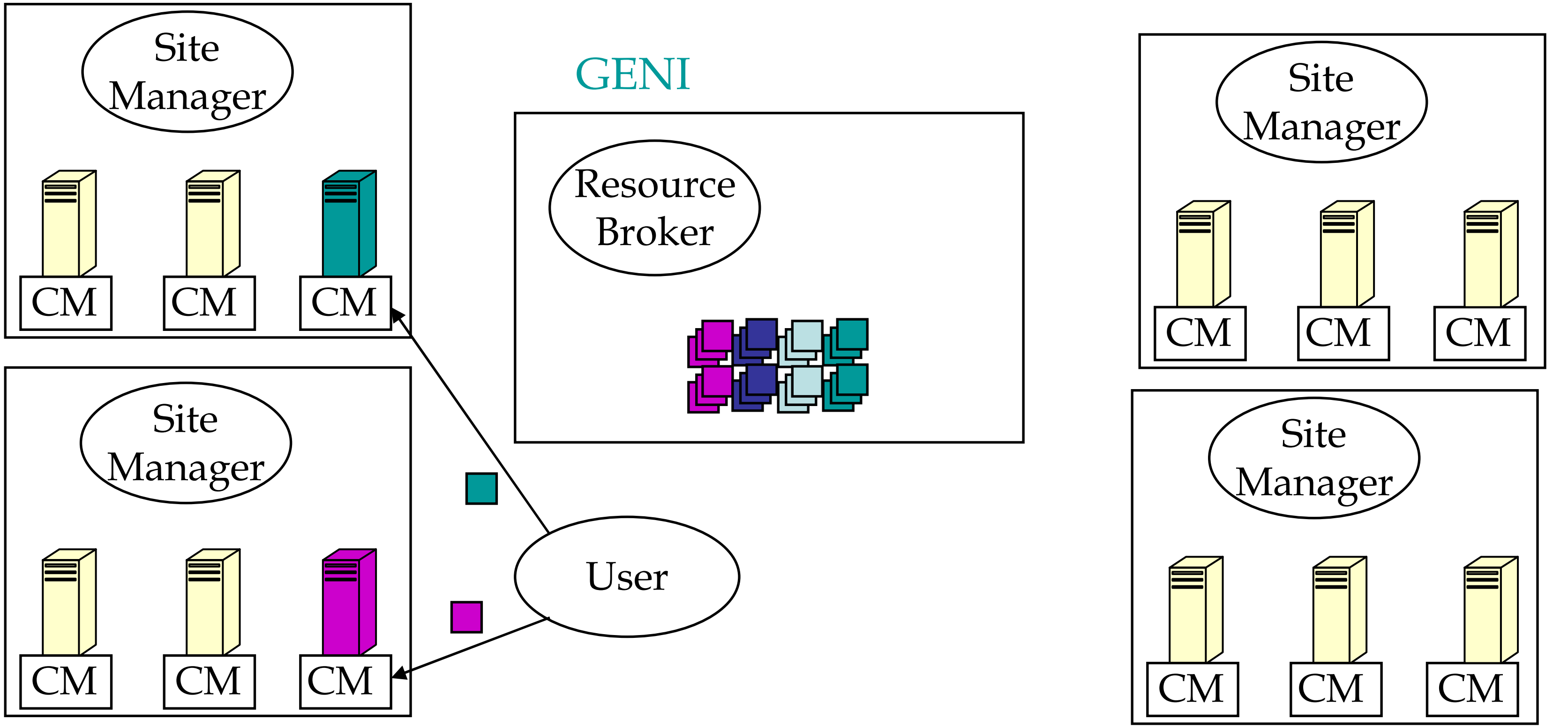
- User presents Token and resource request to Resource Broker

GENI Resource Allocation



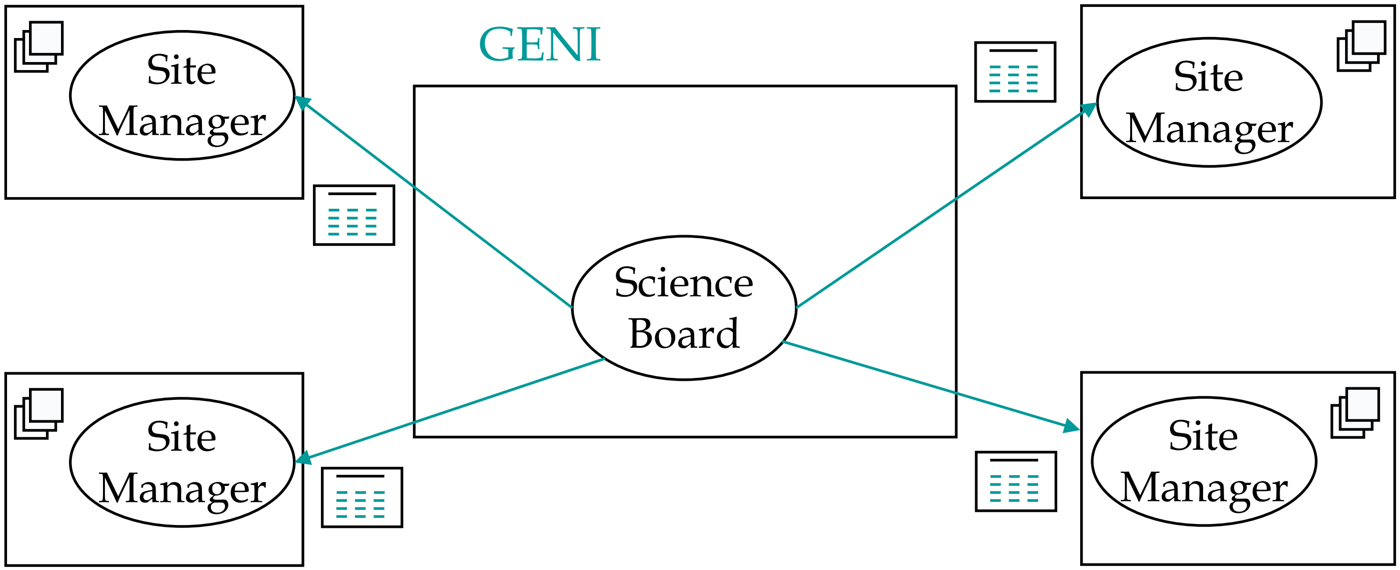
- Resource Broker (possibly after consulting with Component Managers on requested resources) returns one Ticket for each requested resource
  - Ticket is a lease: guarantees access to resource for period of time

GENI Resource Allocation



- User presents each Ticket to a Component Manager, receives Sliver (handle to allocated resources)

GENI Resource Allocation



- GENI Science Board can directly issue Tokens and Tickets to users and sites to reward particularly useful services or hardware

Additional Details: Donations

```
<xsd:complexType name="Donation">
  <xsd:sequence>
     <xsd:element name="GUID" type="xsd:string"/>
     <xsd:element name="Recipient" type="xsd:string"/>
     <xsd:element name="RSpec" type="tns:RSpec">
     <xsd:element name="Signature" type="xsd:base64Binary"/>
     </xsd:sequence>
</xsd:complexType>

 <xsd:complexType name="RSpec">
  <xsd:sequence>
     <xsd:element name="Issuer" type="xsd:string"/>
     <xsd:element name="Resources"
  type="tns:ResourceGroup"/>
     <xsd:element name="IsolationPolicy"
  type="tns:IsolPolicy"/>
     <xsd:element name="AUP" type="tns:AUP“>
     <xsd:element name="ValidStart" type="xsd:dateTime"/>
     <xsd:element name="ValidEnd" type="xsd:dateTime"/>
  </xsd:sequence>
</xsd:complexType>
```

Additional Details: Tokens

```
<xsd:complexType name="Token">
    <xsd:sequence>
      <xsd:element name="Issuer" type="xsd:string"/>
      <xsd:element name="GUID" type="xsd:string"/>
      <xsd:element name="Recipient" type="tns:SliceName"/>
      <xsd:element name="Value" type="xsd:decimal"/>
      <xsd:element name="ValidStart" type="xsd:dateTime"/>
      <xsd:element name="ValidEnd" type="xsd:dateTime"/>
      <xsd:element name="ParentGUID" type="xsd:string"/>
      <xsd:element name="Signature" type="xsd:base64Binary"/>
    </xsd:sequence>
</xsd:complexType>
```

Additional Details: Tickets

```
<xsd:complexType name="Ticket">
    <xsd:sequence>
      <xsd:element name="GUID" type="xsd:string"/>
      <xsd:element name="Recipient" type="tns:SliceName"/>
      <xsd:element name="RSpec" type="tns:RSpec"/>
      <xsd:element name="ValidFor" type="xsd:duration"/>
      <xsd:element name="Signature" type="xsd:base64Binary"/>
    </xsd:sequence>
</xsd:complexType>
```

Implementing RA policies

- Current PlanetLab RA policy (per-node proportional share)
  - Site Manager donates nodes
  - SM receives >= N*M Tickets
    - N=# of PL nodes, M=# users at site
  - SM gives each user N Tokens of value 1
  - User presents one Token of value 1 and a resource request to RB
  - RB returns a Ticket authorizing prop.-share use of requested node
  - User presents Ticket to CM, which returns Sliver on its node
  - User's share = 1/P where P=number of users (slivers) on the node
- Weighted proportional share
  - As above, but user presents Token of value T to RB (T may be > 1)
  - User's share = T/Q where Q=number of Tokens redeemed by other slivers that are using the node

Implementing RA policies (cont.)

- User wants guaranteed share of a node's resources
  - User presents token of value T + resource request to RB
  - RB returns a Ticket for guaranteed T% share of requested node
  - User presents Ticket to CM, which returns Sliver on its node
    - sliver is guaranteed a T% share of the node's resources
  - But if RB has already committed more than 100-T% of the node, either
    - 1) RB refuses to grant Ticket, then
    - (a) user tries again later, or
    - (b) user tries again immediately, specifying a later starting time, or
    - (c) out-of-band mechanism used to queue the request and issue callback
      to user when T% of the resource is available
    - 2) Or, RB grants the Ticket, setting *ValidFor* to requested duration; user presents Ticket at any time between *ValidFrom* and *ValidTo*

Implementing RA policies (cont.)

- Resource auctions
  - RB coordinates the bidding
  - “Cost” of using a resource is dynamically controlled by changing “exchange rate” of Token value to Ticket share

- Loan/transfer/share resources among users, brokers, or sites
  - Tokens are transferrable, *ParentGuid* traces delegation chain
  - Sites and users can give tokens to other sites or users

Resource Alloc Deliverables (17E)

1. Public API to set resource privileges on per-user/per-site basis.
2. Public API to set use privileges on per-component basis (for site admins).
3. Initial web-based interface to allow GENI Science Council to set per-user/per-site privileges using API in step 1.
4. Initial web-based interface to allow administrators to set policy for locally available components.
5. Refined versions of 1, 2, 3, 4 above based on user and community feedback.
6. Design of capabilities to represent GENI resources.
7. Design of tickets representing leases for access to individual GENI components.
8. Initial implementation of Resource Brokers and client software to present requests for resources and to obtain the appropriate set of tickets.
9. Site administrator API and web interface to assign privileges on a per-user basis.
10. Integration with resource discovery service.
11. Integration with experiment management software.

Open Issues

- Specifying resource aggregates (e.g. a cluster)
- Multiple, decentralized RBs rather than a single centralized RB run by GENI
- Describing more complex sharing policies
- Build and deploy real implementation
  - Site Manager, Resource Broker, Component Manager as SOAP web services
  - build on top of existing GMC XSD specifications
    - http://www.geni.net/wsdl.php

Resource Allocation Conclusion

- Goal: flexible resource allocation framework for specifying a broad range of policies
- Proposal: centralized Resource Broker, per-site Site Managers, per-node Component Managers
- Properties
  - rewards sites for contributing resources
    - with special back-door to give users and sites bonus resources
  - encourages users to consume only the resources they need
  - allows to express a variety of sharing policies
  - all capabilities (donations, tokens, tickets, slivers) time out
    - allows resources to be garbage collected
    - allows dynamic valuation of users and resources
  - currently centralized, but architecture allows decentralization
  - secure (all capabilities are signed)

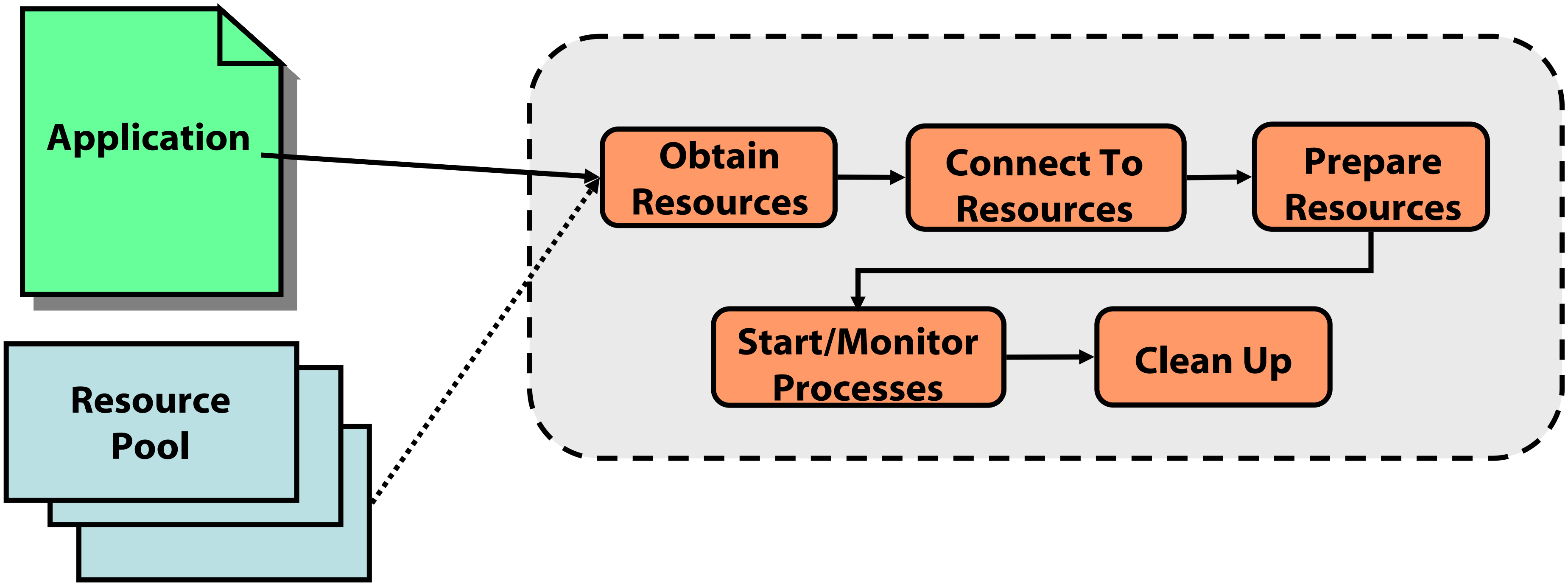
Topics

- Security architecture
- Edge cluster hardware/software definition
- Storage services
- Resource allocation
- Experiment support
- Operations support
- Communications substrate
- Legacy Internet applications support

Experimenter's Support Toolkit

- Make it easy to set up and run experiments on GENI
- Goal: make GENI accessible to the broadest set of researchers, including those at places with little prior institutional experience
- Support different types of experiments/users:
  - Beginners vs. expert programmers
  - Short-term experiments vs. long-running services
  - Homogeneous deployments vs. heterogeneous deployments

Typical Experiment Cycle

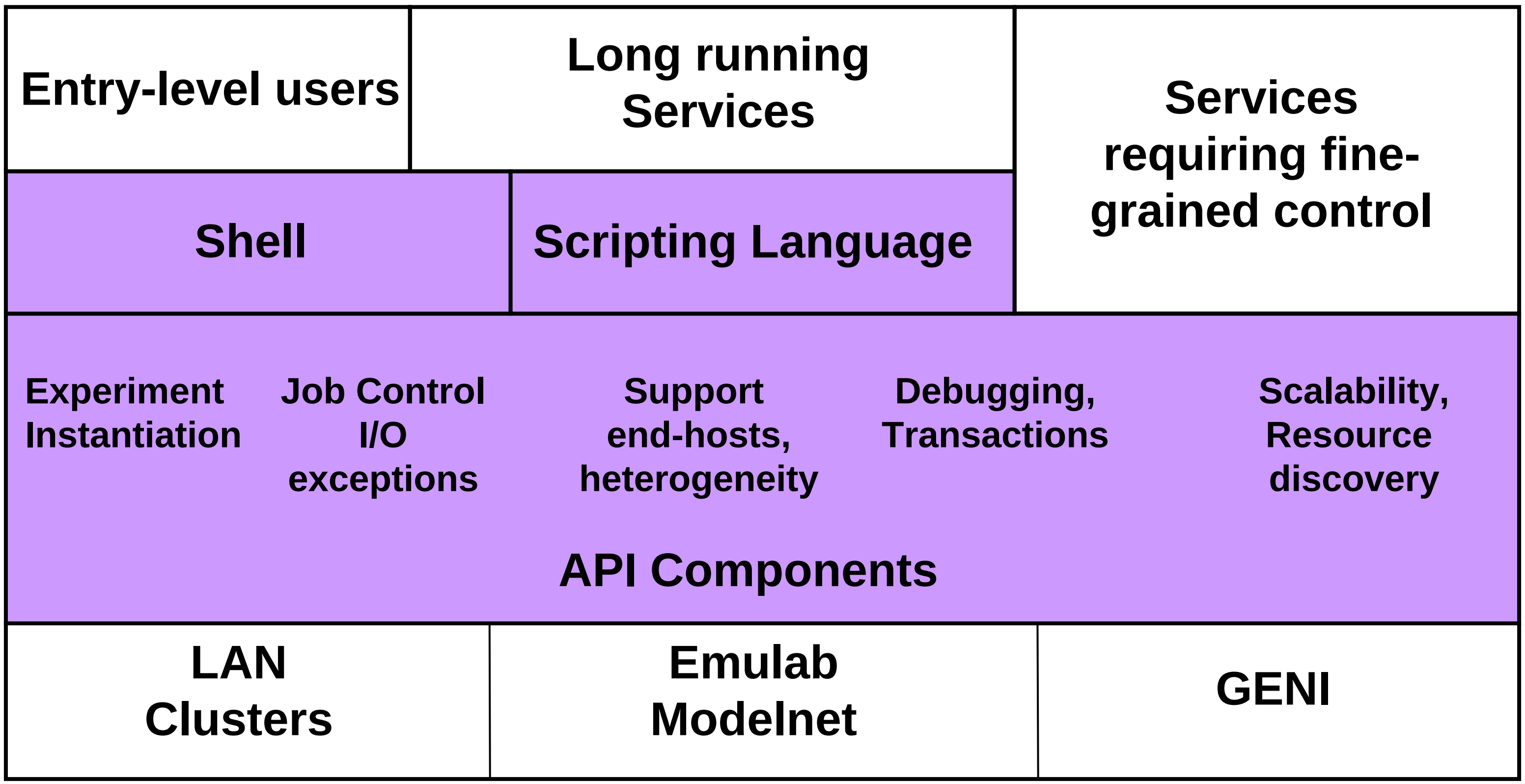


Picture will look different for long-running services as process monitoring, resource preparation, etc. will proceed in a cycle

Desired Toolkit Attributes

- Support gradual refinement:
  - Smooth implementation path from simulation to deployment
  - Same set of tools for both emulation and real-world testing
- Make toolkit available in different modes
  - Stand-alone shell
  - Library interface
  - Accessible from scripting languages
- Enable incremental integration with other services
  - For instance, should be able to change from one content distribution tool to another by just changing a variable
- Sophisticated fault handling
  - Allow experimenters to start with controlled settings and later introduce faults and performance variability
  - Library support for common design patterns for fault-handling

Toolkit as an Abstraction Layer



Basic Toolkit Components

- System-wide parallel execution
  - Start processes on a collection of resources
  - Integrate support for suspend/resume/kill
  - Issue commands asynchronously
  - Support various forms of global synchronization (barriers, etc.)
- Node configuration tools:
  - Customizing node, installing packages, copying executables, etc.
- Integrate with monitoring sensors
  - Distributed systems sensors such as slicestat, CoMon
  - Information planes for network performance (such as iPlane)
- Integrate with other key services
  - Content distribution systems, resource discovery systems, etc.

Advanced Components

- Key stumbling block for long-running services is ensuring robustness in the presence of failures
- Need to provide support for incremental resource allocation
- Library support for common design patterns to handle faults
  - Support for transactional operations and two-phase commits, support “execute exactly once” semantics, etc.
- Support for detecting abnormal program behavior, application-level callbacks, debugging, etc.
- Reliable delivery of control signals, reliable delivery of messages

Experiment Support (30E)

1. Tools for performing system-wide job control: such as executing the same command on all nodes with desired levels of concurrency, etc.
2. Tools for performing operations in asynchronous manner and synchronizing with previously executed commands.
3. Tools for setting up necessary software packages and customizing the execution environment.
4. Tools for coordinated input-output (copying files and logs).
5. Exposing the toolkit functionality in a library API.
6. Exposing the toolkit functionality using a graphical user interface (6-8E)
7. Integration of tools into scripting languages.
8. Provide simple ways for users to specify desired resources.
9. Resilient delivery of control signals.
10. Provide transactional support for executing system-wide commands.
11. Provide support for detecting faults in experiments.
12. Scalable control plane infrastructure -- dissemination of system-wide signals, coordinated I/O, and monitoring program execution should all be done in a scalable manner (3-5E)
13. Interface with content distribution, resource discovery, slice embedding systems.
14. Interface with the information plane for communication subsystem and various sensors monitoring the testbed.
15. Tools for checking for global invariants regarding the state of a distributed experiment (4E)
16. Logging to enable distributed debugging.
17. Debugging support for single-stepping and breakpoints.

Slice Embedding Deliverables (25E)

1) Resource specification language for describing user's needs.
2) Generic matching engine.
3) Algorithms for efficient matching.
4) Matching engine for each subnet.
5) Stitching module to compose results from different subnets.
6) Integration with the resource discovery system to identify available resources.
7) Integration with the resource allocation system to ensure allocation.

Topics

- Security architecture
- Edge cluster hardware/software definition
- Storage services
- Resource allocation
- Experiment support
- Operations support
- Communications substrate
- Legacy Internet applications support

Monitoring

- Goals
  - Reduce cost of running system through automation
  - Provide mechanism for collecting data on operation of system
  - Allow users to oversee experiments
  - Infrastructure (i.e., node selection, slice embedding, history, etc.)
- History
  - Clusters, Grid – Ganglia
  - PlanetLab – CoMon, Trumpet, SWORD
- Metrics
  - Node-centric: CPU, disk, memory, top consumers
  - Project-centric: summary statistics (preserves privacy)
- Triggers
  - Node, project activity "out of bounds"
  - Warning messages, actuators
  - Combinations with experiment profiles

Operations Support Issues

- Two categories of support systems
  - Online: monitor the function and performance of GENI components in real-time
    - Use the ITU FCAPS model to classify necessary support systems
  - Offline: problem tracking, maintenance requests, and inventory
- Build or buy decisions
  - First preference is to use open-source if available, appropriate, and competitive
    - Develop re-distributable extensions as appropriate
  - Second preference is to purchase COTS software
    - Evaluate cost per seat, educational discounts, and impact of restricted access to system data
  - Last choice is to build systems from scratch if no suitable alternatives exist

FCAPS (Fault, Configuration, Accounting, Performance, Security)

- Fault management
  - Detect and track component faults in running system
  - Initiate and track the repair process
  - Example systems: Nagios, HP OpenView, Micromuse Netcool
- Configuration management
  - Automate and verify introduction of new GENI nodes
  - Provision and configure new network links
  - Track GENI hardware inventory across sites
  - Examples: PlanetLab boot CD, Telcordia Granite Inventory, Amdocs Cramer Inventory, MetaSolv
- Accounting
  - Manage user and administrator access to GENI resources
  - Map accounts to real people and institutions
  - Examples: PlanetLab Central, Grid Account Management Architecture (GAMA)

FCAPS (Fault, Configuration, Accounting, Performance, Security)

- Performance management
  - Fine-grained tracking of resource usage
  - Queryable by administrators and adaptive experiments
  - Detecting and mitigating transient system overloads and/or slices operating outside their resource profiles
  - Examples: CoMon, HP OpenView, Micromuse Netcool
- Security management
  - Log all security-related decisions in an auditable trail
    - Viewable by cognizant researcher and operations staff
  - Monitor compliance with Acceptable Use Policy
    - Try to detect certain classes of attacks before they can cause significant damage
  - Examples: Intrusion detectors, compliance systems, etc.

Problem Tracking

- All researcher/external trouble reports, plus any traffic incident reporting
  - Examples: this filesystem seems corrupt, this API does not seem to match the behavior I expect, or "why did I receive this traffic?"
- Receive alerts/alarms from platform monitoring system (e.g., Nagios, OpenView, etc.)
  - Track all reported alarms, delegate to responsible parties, escalate as needed
  - Classify severity, prioritize development/repair effort
- Examples: Request Tracker (RT), Bugzilla, IBM/Rational ClearQuest

Operations Support (31E)

1) GENI Fault Management System software (4E)
2) GENI Configuration Management System software (4E)
3) GENI Accounting Management System software (2E)
4) GENI Performance Management System software (3E)
5) GENI Security Management System software (2E)
6) GENI Problem Tracking System software (2E)
7) GENI Community Forum software (2E)
8) Lifecycle management of all software components (12E)

Communication Substrate

- Bulk data transfer.
- Small message dissemination (e.g., application level multicast) for control messages
- Log/sensor data collection
- Information plane to provide topology information about both GENI and the legacy Internet
- Secure control plane service running on GENI
  - so that device control messages traverse over the facility itself, and therefore cannot be disrupted by legacy Internet traffic.
  - essential if the facility is to be highly available.

Communication Deliverables (11E)

1. Bulk data transfer (e.g., CoBlitz or Bullet), to load experiment code onto a distributed set of machines (3E)
2. Small message dissemination (e.g., application level multicast) for control messages to a distributed set of machines (1E)
3. Log/sensor data collection, from a distributed set of machines to a central repository/analysis engine (3E)
4. Information plane to provide topology information about GENI and the legacy Internet (1E)
5. Software maintenance and upgrades (3E)

Legacy Services

- Virtualized HTTP (2E)
  - Allow experiments to share port 80
- Virtualized DNS (2E)
  - Allow experiments to share port 53
- Client opt-in (12E)
  - Assumes Symbian, Vista, XP, MacOS, Linux, WinCE
- Distributed dynamic NAT (2E)
  - Connections return to source
- Virtualized BGP (backbone group)

Prioritization

High priority ("can't live without it"): 60E
- Data transfer to set up experiments
- Local storage
- Resource allocation/mgt
- Operations support

Medium priority ("should have"): 50E
- Legacy services
- Quick and dirty experiment support
- Efficient disk storage
- Log data collection

Nice to have: 40E
- Information plane
- Simplified file system interfaces for execution
- More functional workbench
- Slice embedding

Notes:
- Security and edge cluster prioritized elsewhere (part of GMC)
- Prioritization also needed within each functional area (e.g., ops support)

Conclusion

- We understand most of what is needed to build security, user support into GENI
  - Lots of work still to do to refine design

- Comments welcome (tom@cs.washington.edu)
  - Design not intended as a fixed point