Middleware Security

> in Selected Grid Infrastructures

David Groep, Nikhef

Grid Security Middleware mechanisms for protecting the e-Infrastructure

What to expect?

What might be covered

- How to deal with AuthN
- AuthZ frameworks
- Access control in services
- Unix credential mapping
- Pilot jobs and late binding
- Security interoperability
- Storage access control
- Data Security and Privacy

What will not be covered

- How to write secure code
  - Look at http://pages.cs.wisc.edu/~kupsch/
- Current vulnerabilities
  - They're secret for a reason…
- What might be there in 2-3 years' time
- Most of the federation work
- The latest WS-* *ML specs

- *… with a slight EGEE & C/Unix bias (sorry)*

Terminology: virtual organisation models

Authentication requirements

Delegation and proxies

Virtual Organisation membership

SECURITY MECHANISM FOUNDATIONS

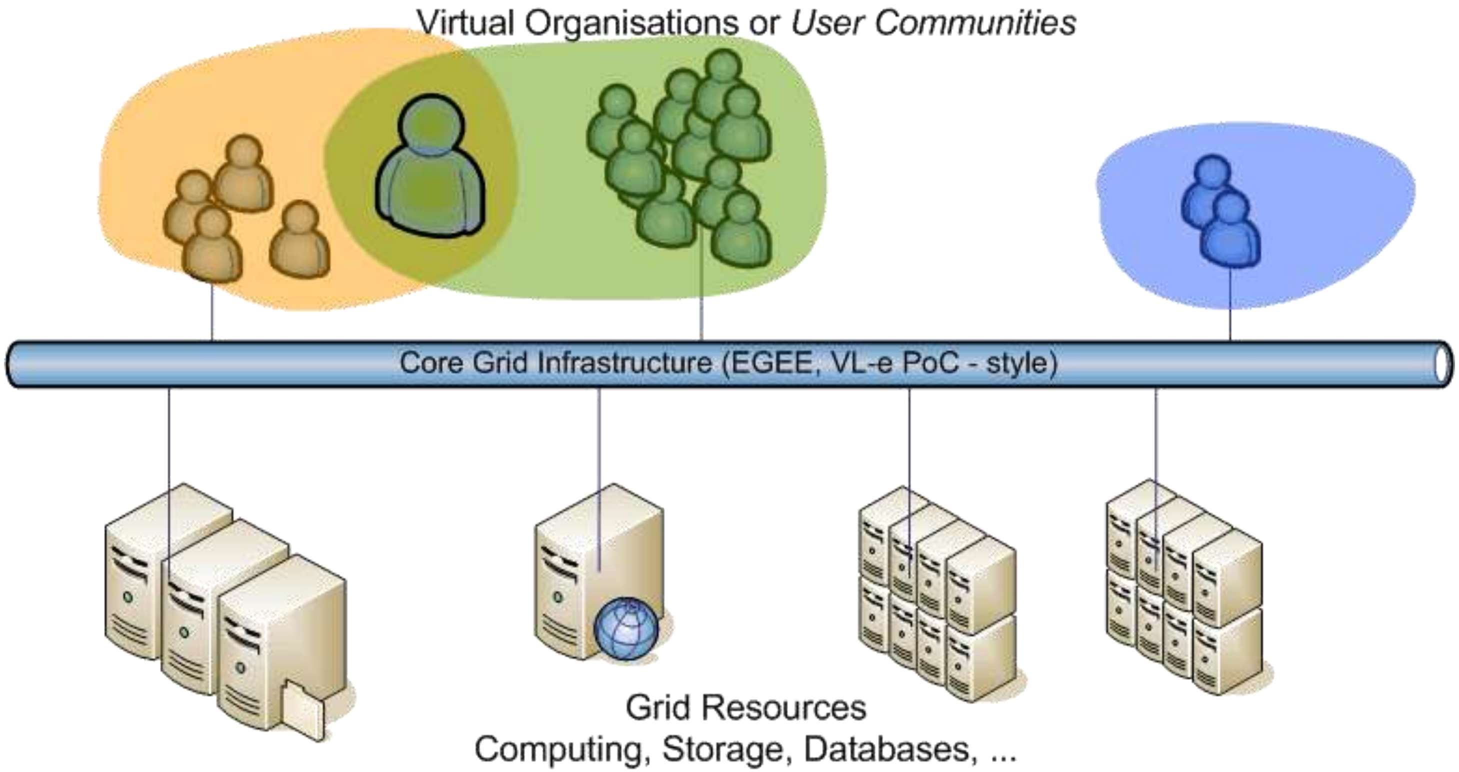
The Virtual Organisation, or ‘VO’

Grids organised around ‘virtual organisations’

- A set of individuals or organisations, **not under single hierarchical control**, (temporarily) **joining forces** to solve a particular problem at hand, bringing to the collaboration a subset of their resources, sharing those **at their discretion** and each **under their own conditions**.

However, the term is used in different ways …

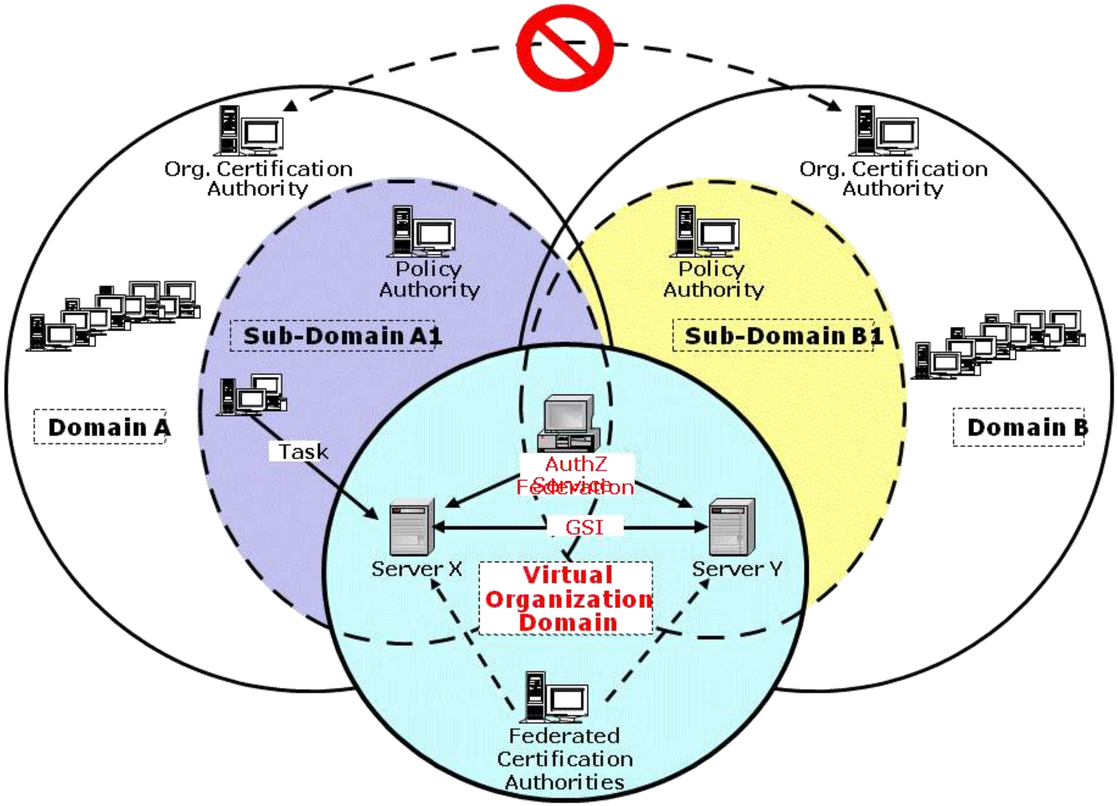
VOs on an e-Infrastructure


Virtual Organisations or *User Communities*
Core Grid Infrastructure (EGEE, VL-e PoC - style)
Grid Resources
Computing, Storage, Databases, ...

- User communities 'live' on a persistent infrastructure
- Communities can exist without their 'own' resources
- ... and resource centres can do without local users

Trust

- For the model to work parties have to *trust* each other
- Organisational trust is hard; Grid's user-resource scales better



Elements of Trust

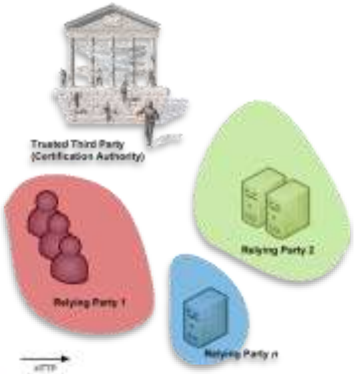
- Authentication
  - Who are you?
  - Who says so?
  - How long ago was that statement made?
  - Have you changed since then?
- Authorization
  - Why should I let you in?
  - What are you allowed to do?
  - By whom? *Who* said you could do that?
  - And how long ago was *that* statement made?

Authentication models

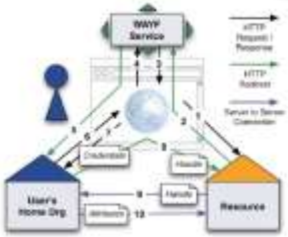
- Direct user-to-site
  - passwords, enterprise PKI, Kerberos

Identity Store
Authentication Server / KDC
Ticket Request
Client
Service Request
Service

- PKI with trusted third parties



- Federated access
  - Controlled & policy based
  - Free-for-all, e.g., OpenID

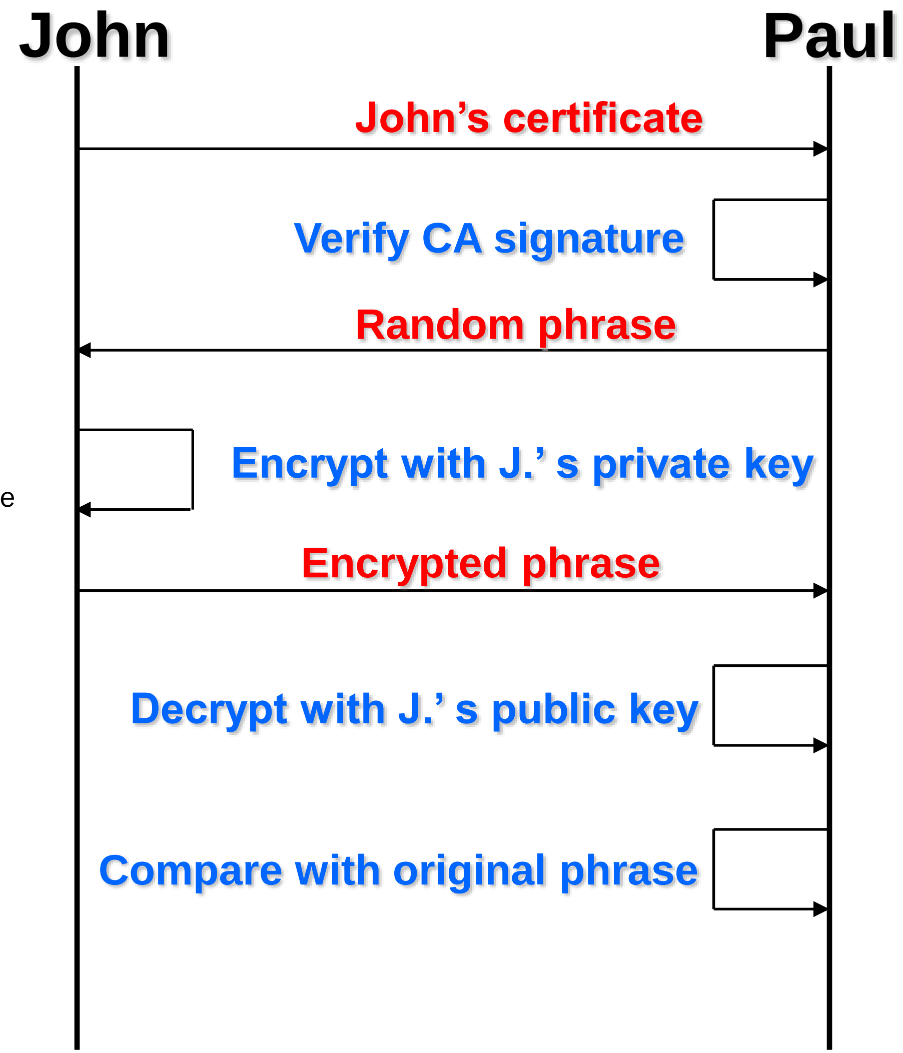


- Identity meta-system
  - Infocard type systems

PKI (1): Asymmetric cryptography

Based on X.509 PKI:

- every user/host/service has an X.509 certificate;
- certificates are signed by trusted (by the local sites) CA's;
- every Grid transaction is **mutually authenticated**:
  1. John sends his certificate;
  2. Paul verifies signature in John's certificate;
  3. Paul sends to John a challenge string;
  4. John encrypts the challenge string with his private key;
  5. John sends encrypted challenge to Paul
  6. Paul uses John's public key to decrypt the challenge.
  7. Paul compares the decrypted string with the original challenge
  8. If they match, Paul verified John's identity and John can not repudiate it.


John
Paul
John's certificate
Verify CA signature
Random phrase
Encrypt with J.' s private key
Encrypted phrase
Decrypt with J.' s public key
Compare with original phrase

PKI (2): Communications

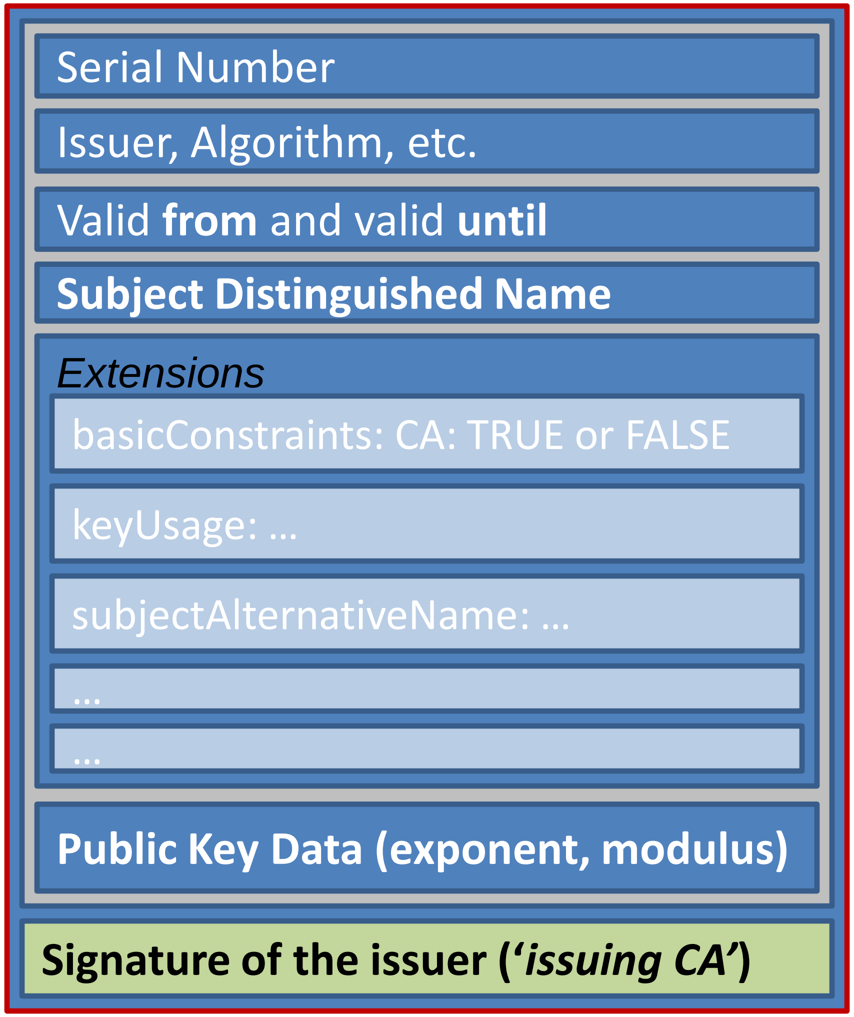
1. Securing the channel
   > ‘Transport Layer Security’ (TLS, formerly SSL)
   > Exchange a symmetric cipher through PK challenge
   > Communications on channel authentic and encrypted

2. Securing the message
   > Can be sent and forwarded over any medium
   > Each and every message needs a signature
   > Examples: XML-DSIG, XML-ENC, but also PGP…

X.509: add identifiers to a public key

- Authentic binding between
  - Subject name
  - A public key
  - A *validity period*
  - Zero or more extensions
  - … that can contain identifiers
  - … or policies

- Signed by an issuer
  - Yourself: self-signed cert
  - Trusted third party, ‘CA’



Serial Number

Issuer, Algorithm, etc.

Valid **from** and valid **until**

Subject Distinguished Name

Extensions

basicConstraints: CA: TRUE or FALSE

keyUsage: …

subjectAlternativeName: …

…

…

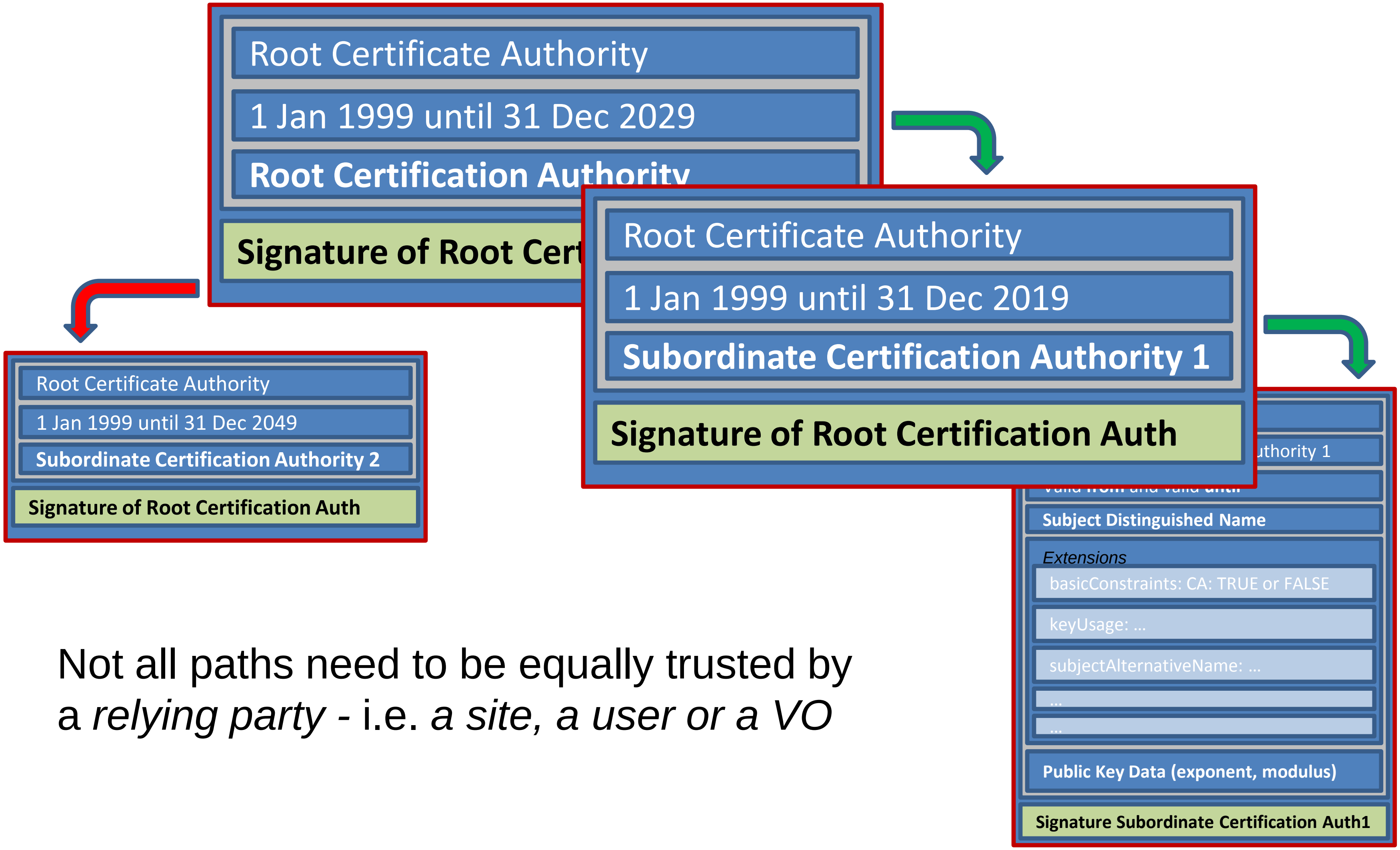
Public Key Data (exponent, modulus)

Signature of the issuer (‘*issuing CA*’)

Example certificate

```
    Version: 3 (0x2)
    Serial Number: 2113 (0x841)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=NL, O=NIKHEF, CN=NIKHEF medium-security certification auth
    Validity    Not Before: Oct 23 00:00:00 2008 GMT
                Not After : Oct 23 07:35:37 2009 GMT
    Subject: O=dutchgrid, O=users, O=nikhef, CN=David Groep
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        RSA Public Key: (1024 bit)
            Modulus (1024 bit):
                00:f1:14:78:97:9b:38:84:69:e0:b7:df:d9:f2:31:
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:FALSE
        X509v3 Key Usage: critical
            Digital Signature, Key Encipherment, Data Encipherment
        X509v3 Extended Key Usage:
            TLS Web Client Authentication, E-mail Protection
        X509v3 CRL Distribution Points:
            URI:http://ca.dutchgrid.nl/medium/cacrl.der
        X509v3 Certificate Policies:
            Policy: 1.3.6.1.4.1.10434.4.2.2.1.3.1
            Policy: 1.2.840.113612.5.2.2.1
        X509v3 Subject Alternative Name:
            email:davidg@nikhef.nl
Signature Algorithm: sha1WithRSAEncryption
    19:d3:82:19:af:96:7d:34:97:61:58:76:4f:a8:56:45:34:90:
```

Building up: CA hierarchies



Root Certificate Authority

1 Jan 1999 until 31 Dec 2029

Root Certification Authority

Signature of Root Cert

Root Certificate Authority

1 Jan 1999 until 31 Dec 2019

Subordinate Certification Authority 1

Signature of Root Certification Auth

Root Certificate Authority

1 Jan 1999 until 31 Dec 2049

Subordinate Certification Authority 2

Signature of Root Certification Auth

uthority 1

Valid from and valid until

Subject Distinguished Name

Extensions

basicConstraints: CA: TRUE or FALSE

keyUsage: …

subjectAlternativeName: …

…

…

Public Key Data (exponent, modulus)

Signature Subordinate Certification Auth1

Not all paths need to be equally trusted by a *relying party* - i.e. *a site, a user or a VO*

Signing policy files

- Constrain name space to specified subject names

```
access_id_CA   X509    '/C=NL/O=NIKHEF/CN=NIKHEF medium-security certification auth'
pos_rights     globus  CA:sign
cond_subjects globus   '"/C=NL/O=NIKHEF/CN=NIKHEF medium-security certification auth"
                        "/O=dutchgrid/O=users/*" "/O=dutchgrid/O=hosts/*"
                        "/O=dutchgrid/O=robots/*"'
```

- For now, specific to Grids – with talk in IETF
  - Recognised in
    - Globus Toolkit C core, and Java in 4.2+
    - gLite Trust Manager
    - GridSite (recent versions only)
  - But still lacking in some places – need patches!
- See OGF CAOPS-WG “RPDNC Policies” document

Verification steps

- Check signature chain up to a trusted root
  - In OpenSSL (and thus most grid middleware) the root of trust *must* be self-signed
- Check basicConstraints and keyUsage
- Check *revocation* of any certificates in the chain
  - Using Certificate Revocation Lists (CRLs) when installed
  - Download on-demand or OCSP not yet supported
- Check RP namespace constraints

Needed for verification

- Trust anchors
  - ‘.0’ files in ‘PEM’ format, e.g. from IGTF
  - download and update in RPM format with yum/apt
  - Or install and refresh with tar balls or JKSs
- Revocation lists
  - ‘.r0’ files in PEM format, retrieved by tools: fetch-crl
- RP namespace constraints
  - ‘.signing_policy’ files, and ‘.namespaces’ files

- Java JKS’s (used in Unicore) are of course different

Proxy certificate verification

VO technologies: LDAP, VOMS-push, VOMS-pull, SAML

Towards a multi-authority world: interlinking federations

DELEGATION AND VIRTUAL ORGANISATIONS

Delegation

- Mechanism to have someone, or some-thing – a program – act on your behalf
  - as yourself
  - with a (sub)set of your rights
- Essential for the grid model to work

- GSI/PKI and recent SAML drafts define this
  - GSI (PKI) through ‘proxy’ certificates (see RFC3820)
  - SAML through *Subject Confirmation*, (linking to at least one key or name)

Delegation, but to whom?

- ‘normal’ proxies form a chain
  - Subject name of the proxy derived from issuer

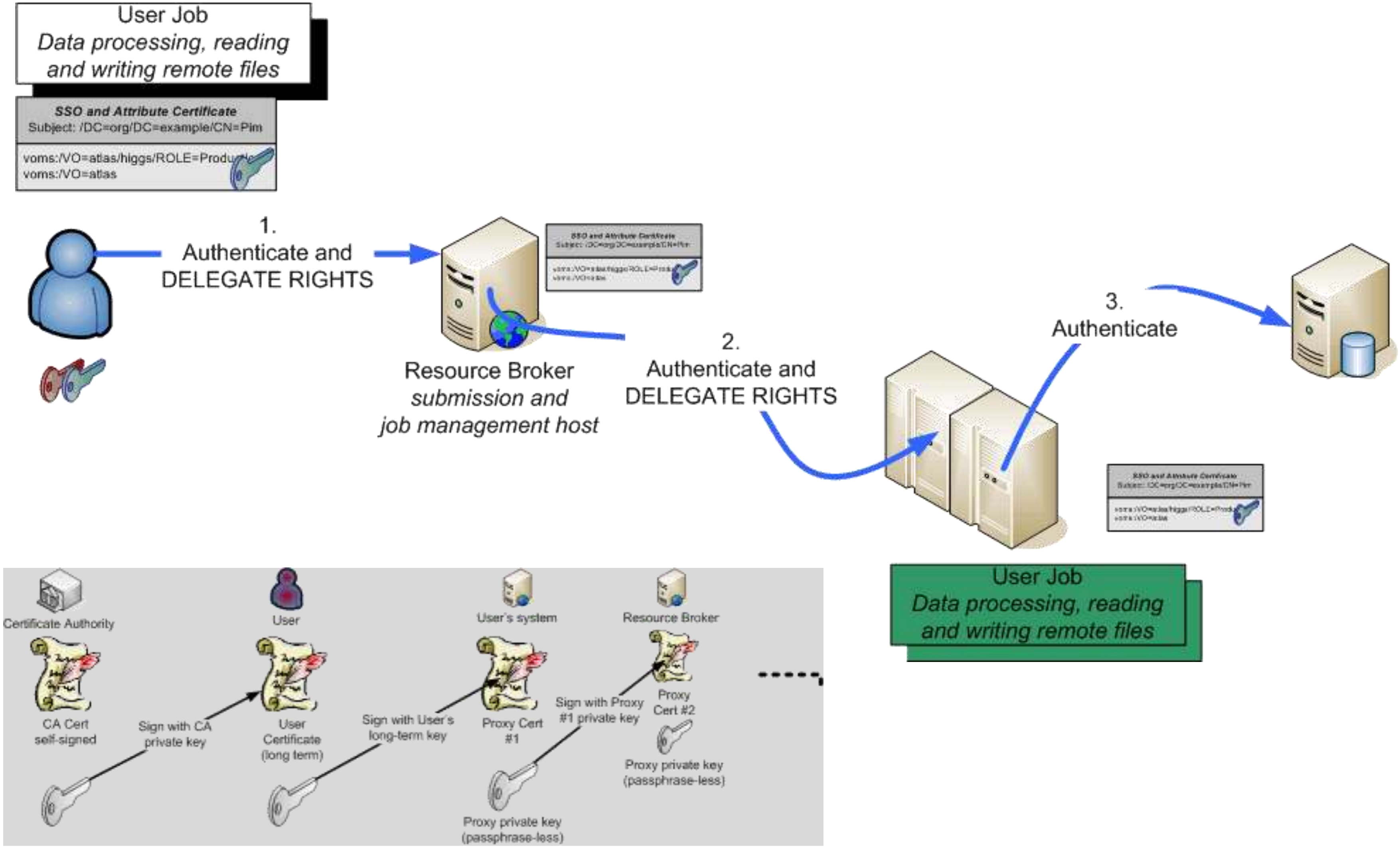
    “/DC=org/DC=example/CN=John Doe/CN=24623/CN=535431”
    is likely a proxy for user
    “/DC=org/DC=example/CN=John Doe”

  - May contain path-length constraint
  - May contain policy constraints
  - And: legacy (pre-3820) proxies abound
- *But: use the name of the real end-entity for authZ!*

Note that

- in SAML, delegation can be to any NameID
- in RFC3820 these are called ‘independent proxies’

Daisy-chaining proxy delegation



User Job

Data processing, reading and writing remote files

SSO and Attribute Certificate

Subject: /DC=org/DC=example/CN=Pim

voms:/VO=atlas/higgs/ROLE=Produ...

voms:/VO=atlas

1. Authenticate and DELEGATE RIGHTS

Resource Broker *submission and job management host*

2. Authenticate and DELEGATE RIGHTS

3. Authenticate

User Job

Data processing, reading and writing remote files

Verifying authentication and X.509

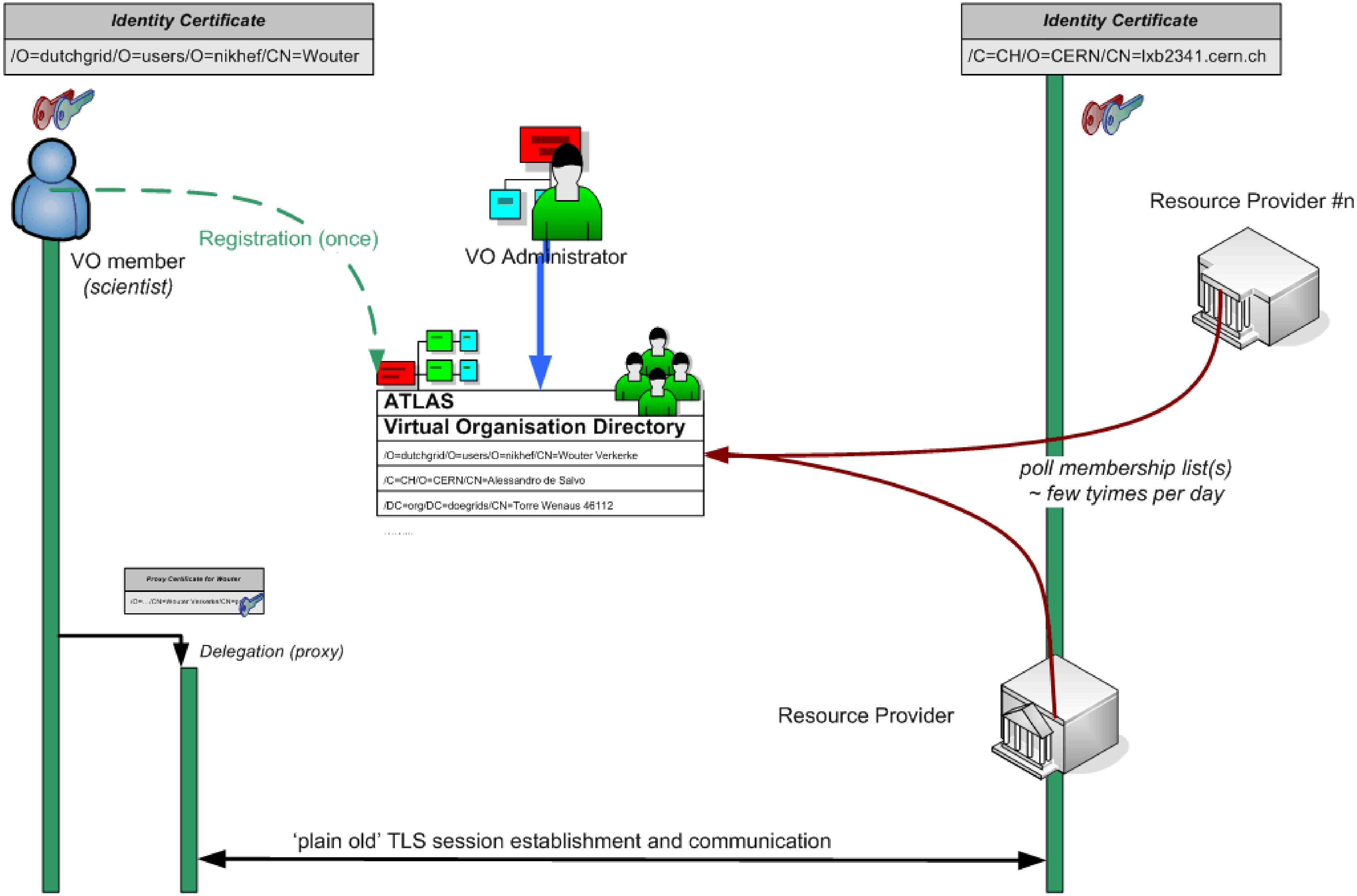
- ‘Conventional’ PKI
  - OpenSSL, Apache mod_ssl
  - Java JCE providers, such as BouncyCastle
  - Perl, Python usually wrappers around OpenSSL
- With proxy support
  - OpenSSL (but beware of outstanding issues!)
  - Globus Toolkit (C, Java)
  - GridSite
  - ProxyVerify library
  - TrustManager
- Always ensure the *proxy policies* are implemented

Authorization: VO representations

- VO is a directory (database) with members, groups, roles
- based on identifiers issues at the AuthN stage
- Membership information is to be conveyed to the resource providers

> configured statically, out of band	early days, *i.e.* < 2001
> in advance, by periodically pulling lists VO LDAP directories	DEISA
> in VO-signed assertions pushed with the request: VOMS, Community AuthZ Service	EGEE (+OSG pulls VOMS)
> Push&pull of assertions via SAML	2010 +

VO LDAP model



Identity Certificate

/O=dutchgrid/O=users/O=nikhef/CN=Wouter

Identity Certificate

/C=CH/O=CERN/CN=lxb2341.cern.ch

VO member *(scientist)*

Registration (once)

VO Administrator

Resource Provider #n

ATLAS

Virtual Organisation Directory

/O=dutchgrid/O=users/O=nikhef/CN=Wouter Verkerke

/C=CH/O=CERN/CN=Alessandro de Salvo

/DC=org/DC=doegrids/CN=Torre Wenaus 46112

……..

poll membership list(s)
~ few tyimes per day

Proxy Certificate for Wouter

/O=…/CN=Wouter Verkerke/CN=p

Delegation (proxy)

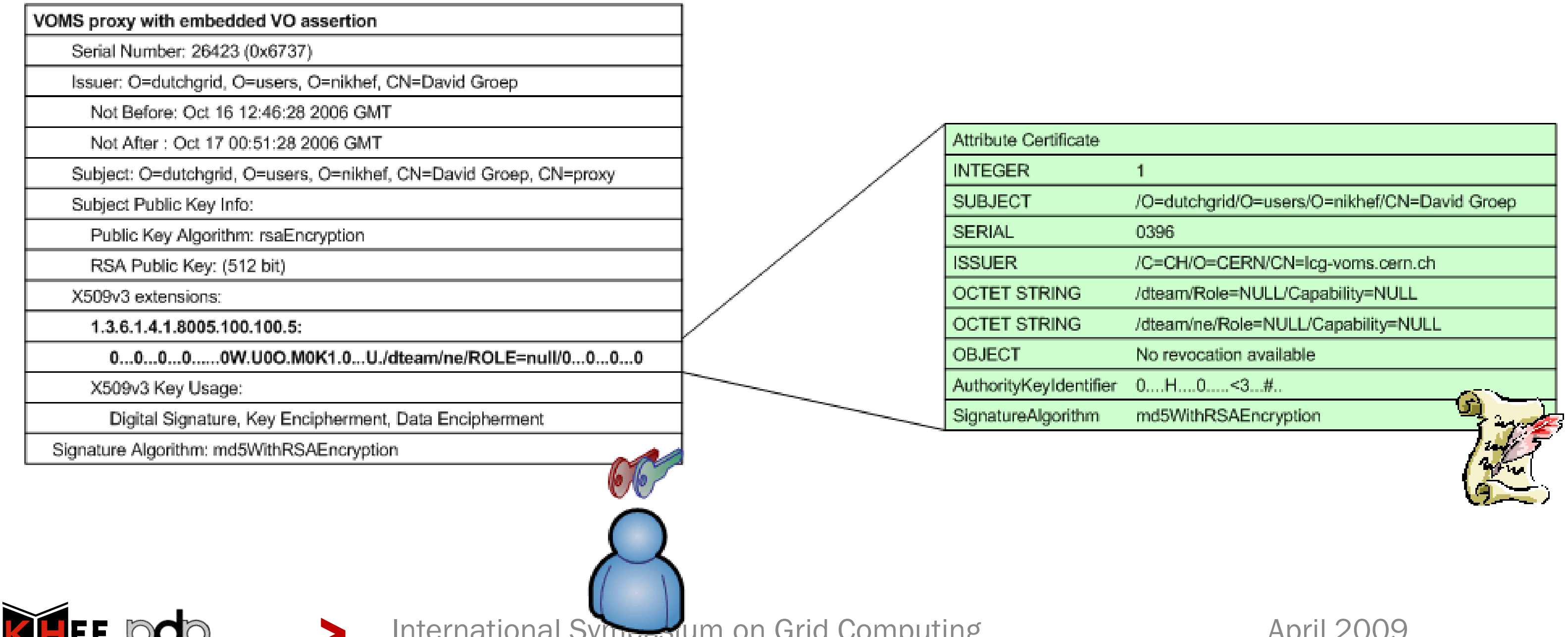
Resource Provider

'plain old' TLS session establishment and communication

VOMS v1: X.509 as a container

Virtual Organisation Management System (VOMS)

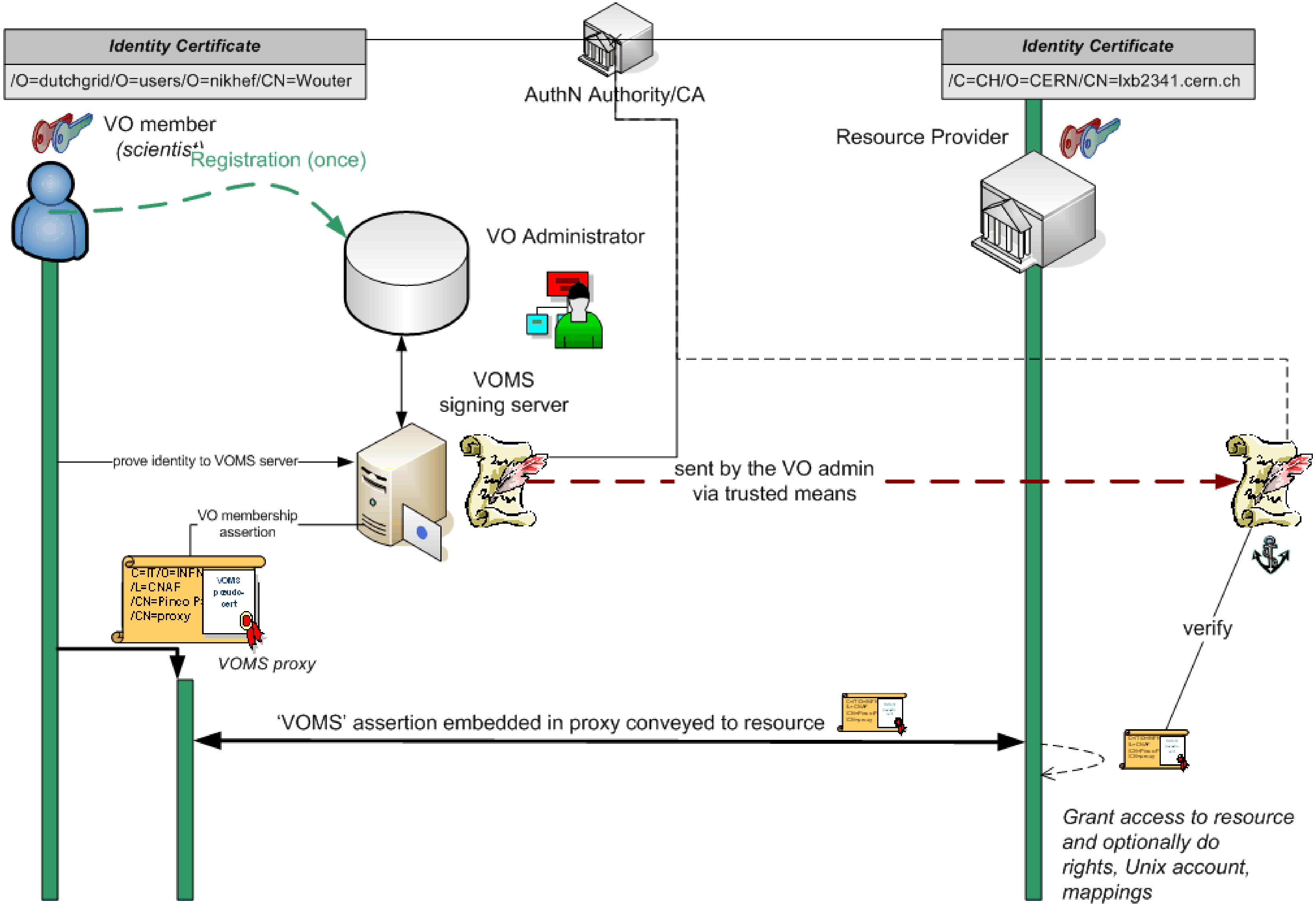
- developed by INFN for EU DataTAG and EGEE
- used by VOs in EGEE, Open Science Grid, NAREGI, …
- push-model signed VO membership tokens
  - using the traditional X.509 ‘proxy’ certificate for trans-shipment
  - fully backward-compatible with only-identity-based mechanisms



VOMS proxy with embedded VO assertion
Serial Number: 26423 (0x6737)
Issuer: O=dutchgrid, O=users, O=nikhef, CN=David Groep
Not Before: Oct 16 12:46:28 2006 GMT
Not After : Oct 17 00:51:28 2006 GMT
Subject: O=dutchgrid, O=users, O=nikhef, CN=David Groep, CN=proxy
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (512 bit)
X509v3 extensions:
1.3.6.1.4.1.8005.100.100.5:
0...0...0...0......0W.U0O.M0K1.0...U./dteam/ne/ROLE=null/0...0...0...0
X509v3 Key Usage:
Digital Signature, Key Encipherment, Data Encipherment
Signature Algorithm: md5WithRSAEncryption

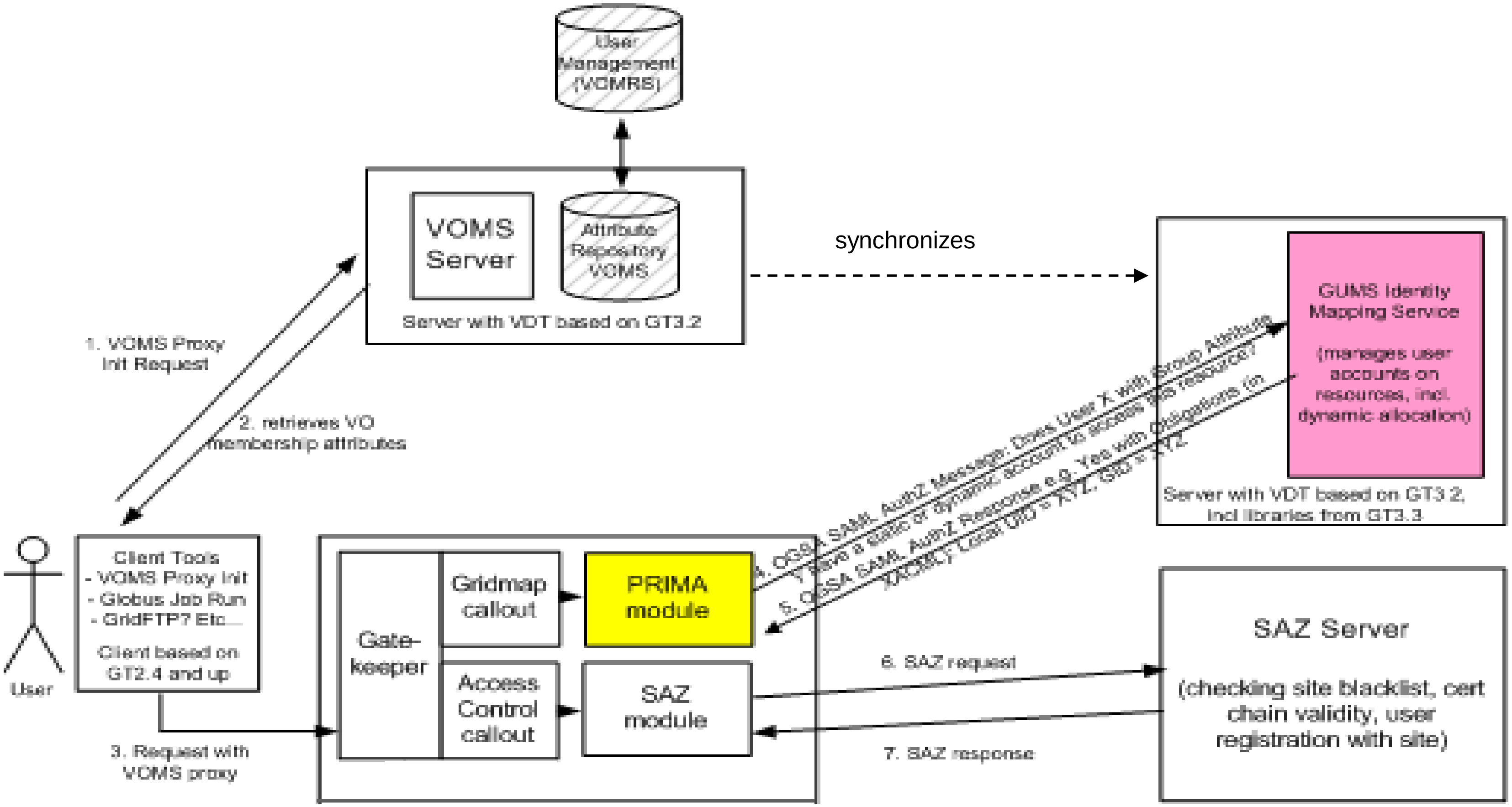
Attribute Certificate	
INTEGER	1
SUBJECT	/O=dutchgrid/O=users/O=nikhef/CN=David Groep
SERIAL	0396
ISSUER	/C=CH/O=CERN/CN=lcg-voms.cern.ch
OCTET STRING	/dteam/Role=NULL/Capability=NULL
OCTET STRING	/dteam/ne/Role=NULL/Capability=NULL
OBJECT	No revocation available
AuthorityKeyIdentifier	0....H....0.....<3...#..
SignatureAlgorithm	md5WithRSAEncryption

VOMS model


Identity Certificate
/O=dutchgrid/O=users/O=nikhef/CN=Wouter
AuthN Authority/CA
Identity Certificate
/C=CH/O=CERN/CN=lxb2341.cern.ch
VO member
(scientist)
Registration (once)
Resource Provider
VO Administrator
VOMS
signing server
prove identity to VOMS server
sent by the VO admin
via trusted means
VO membership
assertion
C=IT/O=INFN
/L=CNAF
/CN=Pinco P
/CN=proxy
VOMS
pseudo-
cert
VOMS proxy
verify
'VOMS' assertion embedded in proxy conveyed to resource
Grant access to resource
and optionally do
rights, Unix account,
mappings

GUMS model

- VO configuration replicated locally at the site
- Here, pushed VOMS attributes are advisory only


User Management (VOMRS)
VOMS Server
Attribute Repository VOMS
Server with VDT based on GT3.2
synchronizes
GUMS Identity Mapping Service
(manages user accounts on resources, incl. dynamic allocation)
Server with VDT based on GT3.2, incl libraries from GT3.3
1. VOMS Proxy Init Request
2. retrieves VO membership attributes
User
Client Tools
- VOMS Proxy Init
- Globus Job Run
- GridFTP? Etc...
Client based on GT2.4 and up
3. Request with VOMS proxy
Gate-keeper
Gridmap callout
PRIMA module
Access Control callout
SAZ module
4. OGSA SAML AuthZ Message: Does User X with Group Attribute Y have a static or dynamic account to access this resource?
5. OGSA SAML AuthZ Response e.g. Yes with Obligations (in XACML): Local UID = XYZ, GID = XYZ
6. SAZ request
7. SAZ response
SAZ Server
(checking site blacklist, cert chain validity, user registration with site)

Graphic: Gabriele Garzoglio, FNAL

Towards a multi-authority world (AAI)

Interlinking of technologies can be cone at various points

1. Authentication: linking (federations of) identity providers to the existing grid AuthN systems
   - ‘Short-Lived Credential Services’ translation bridges
2. Populate VO databases with UHO Attributes
3. Equip resource providers to also inspect UHO attributes
4. Expressing VO attributes as function of UHO attributes

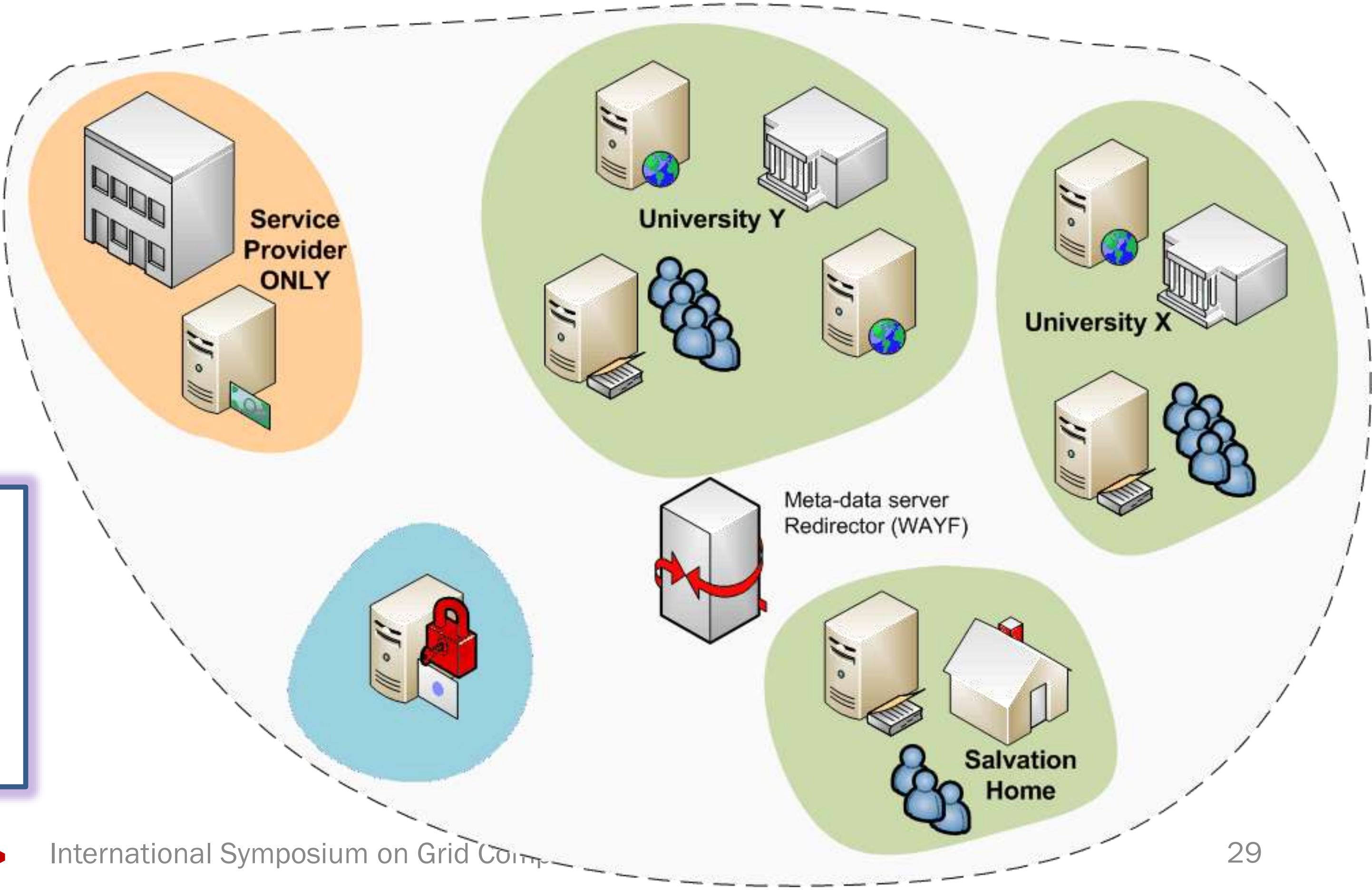
- *and most probably many other options as well …*

Leads to assertions with multiple LoAs in the same decision

- thus all assertions should carry their LoA
- expressed in a way that’s recognisable
- and the LoA attested to by ‘third parties’ (i.e. the federation)

Federations

- A common Authentication and Authorization Infrastructure
- Allow access to common resources with a single credential

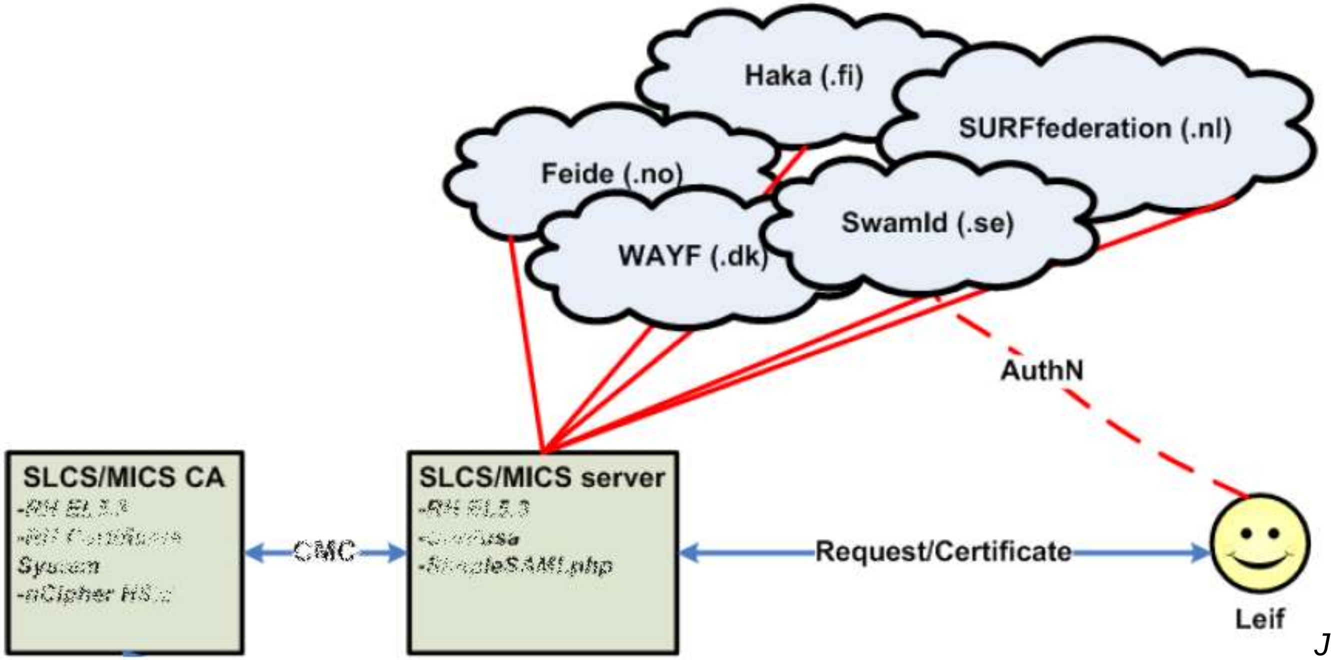


grid structure was not too much different!

A Federated Grid CA

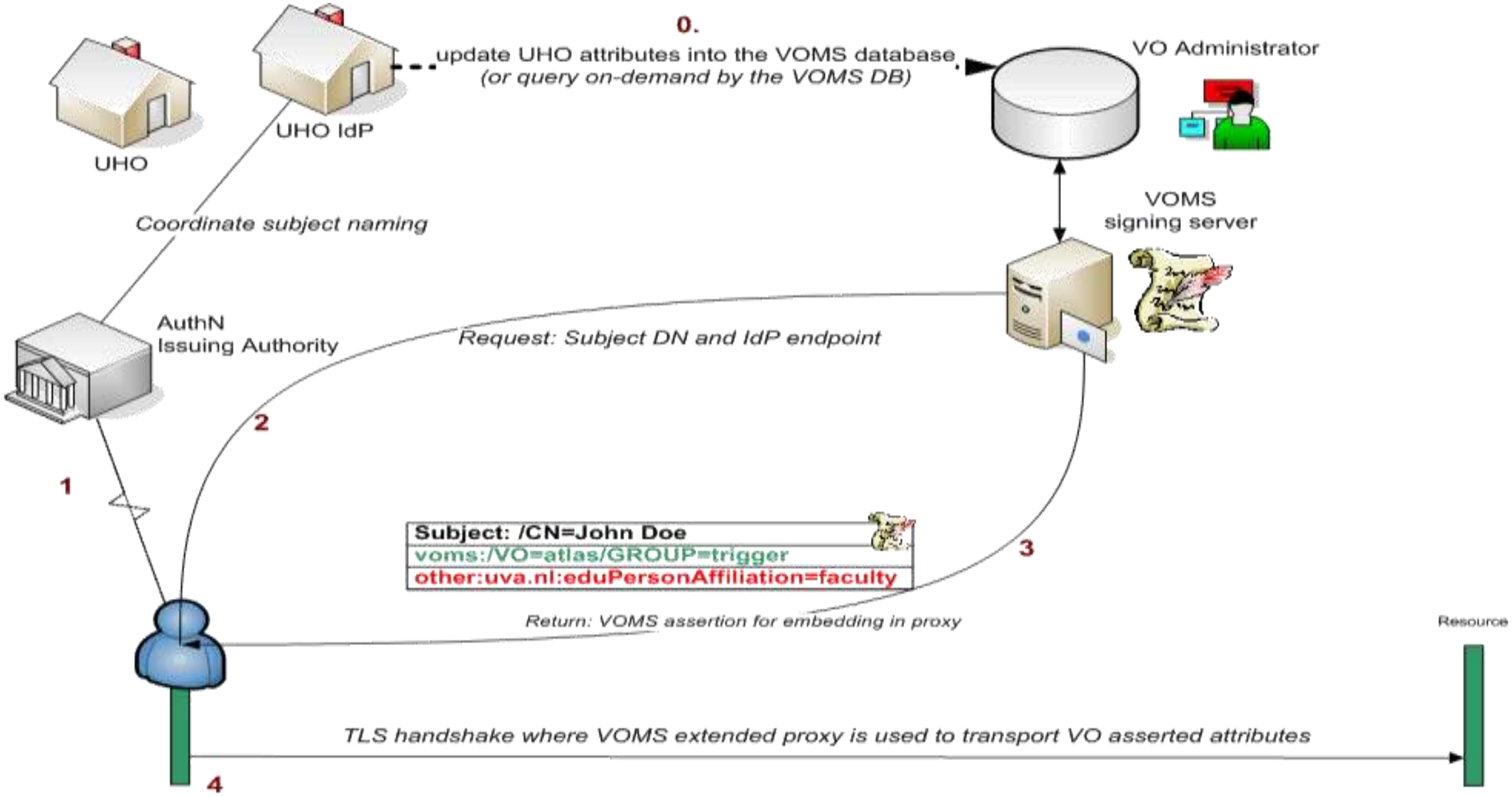
- Use your federation ID
- … to authenticate to a service
- … that issues a certificate
- … recognised by the Grid today

Implementations:
- *SWITCHaai SLCS*
- *TERENA Grid CA Service*



Graphic from:
Jan Meijer, UNINETT

Putting home attributes in the VO

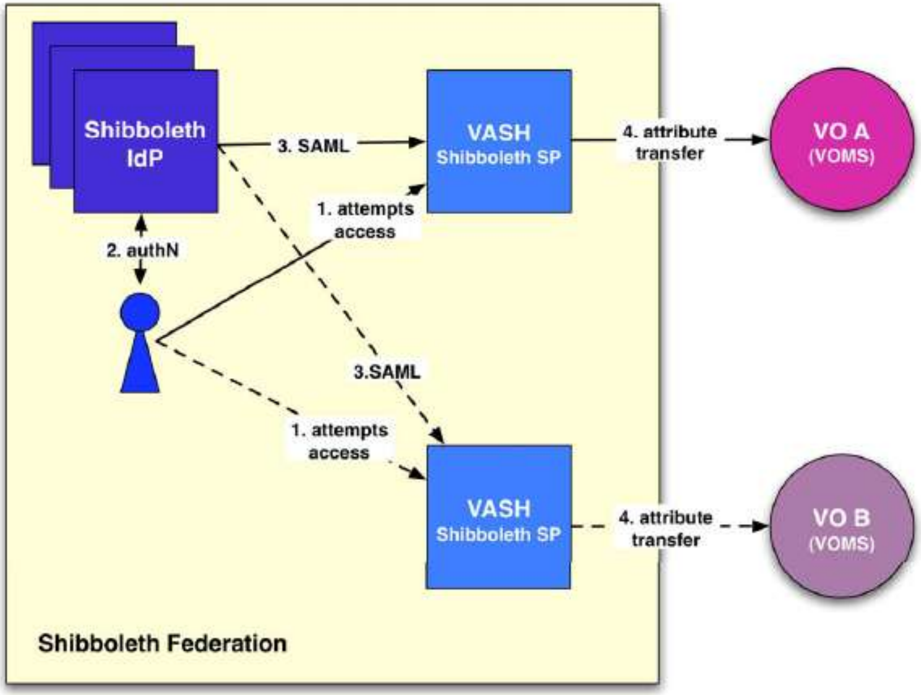

0.
update UHO attributes into the VOMS database
(or query on-demand by the VOMS DB)
VO Administrator
UHO IdP
UHO
Coordinate subject naming
VOMS
signing server
AuthN
Issuing Authority
Request: Subject DN and IdP endpoint
2
1
Subject: /CN=John Doe
voms:/VO=atlas/GROUP=trigger
other:uva.nl:eduPersonAffiliation=faculty
3
Return: VOMS assertion for embedding in proxy
Resource
TLS handshake where VOMS extended proxy is used to transport VO asserted attributes
4

- Characteristics
  - The VO will know the source of the attributes
  - Resource can make a decision on combined VO and UHO attributes
  - but for the outside world, the VO now has asserted to the validity of the UHO attributes – over which the VO has hardly any control

Attributes from multi-authority world

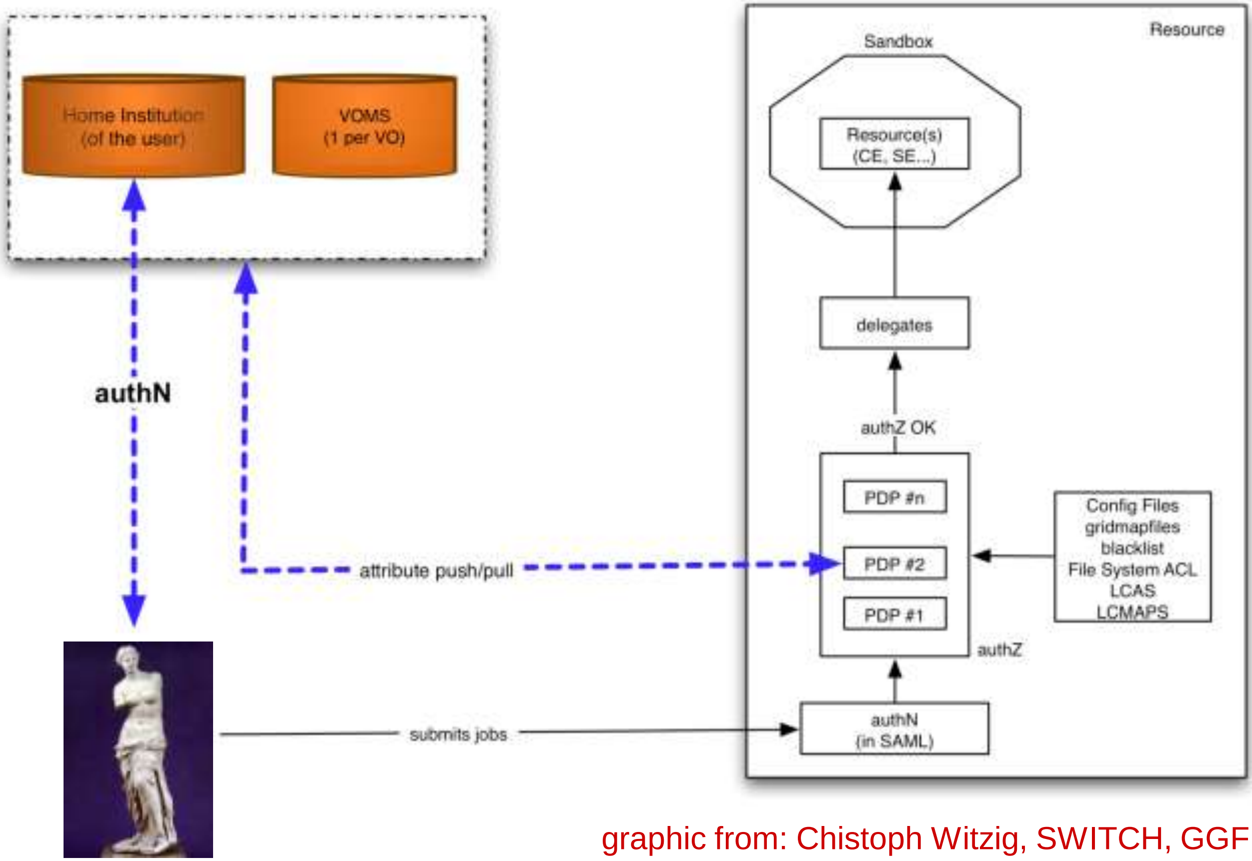
- In 'conventional' grids, all attributes assigned by VO
- But there are many more attributes

- VASH: 'VOMS Attributes from Shibboleth'
  - Populate VOMS with generic attributes
  - Part of gLite (SWITCH)



http://www.switch.ch/grid/vash/

Attribute collection at the resource



graphic from: Chistoph Witzig, SWITCH, GGF16, February 2006

- Characteristics
  - The RP (at the decision point) knows the source of all attributes
  - but has to combine these and make the ‘informed decision’
  - is suddenly faced with a decision on quality from different assertions
  - needs to push a kind of ‘session identifier’ to select a role at the target resource

Container versus service level

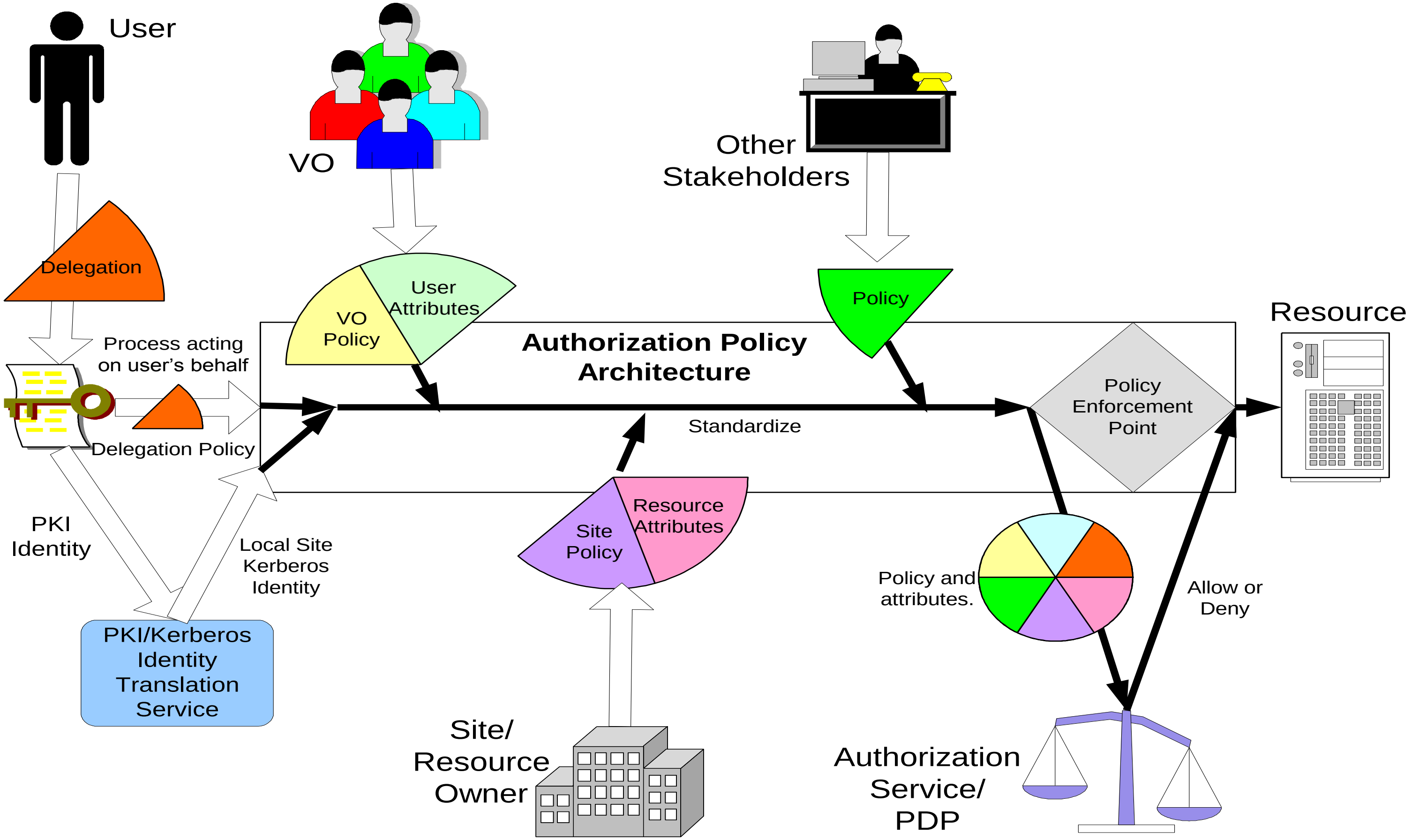
Logical authZ structure: PEP,PDP,PAP,PEP

Frameworks

AUTHORIZATION FRAMEWORKS

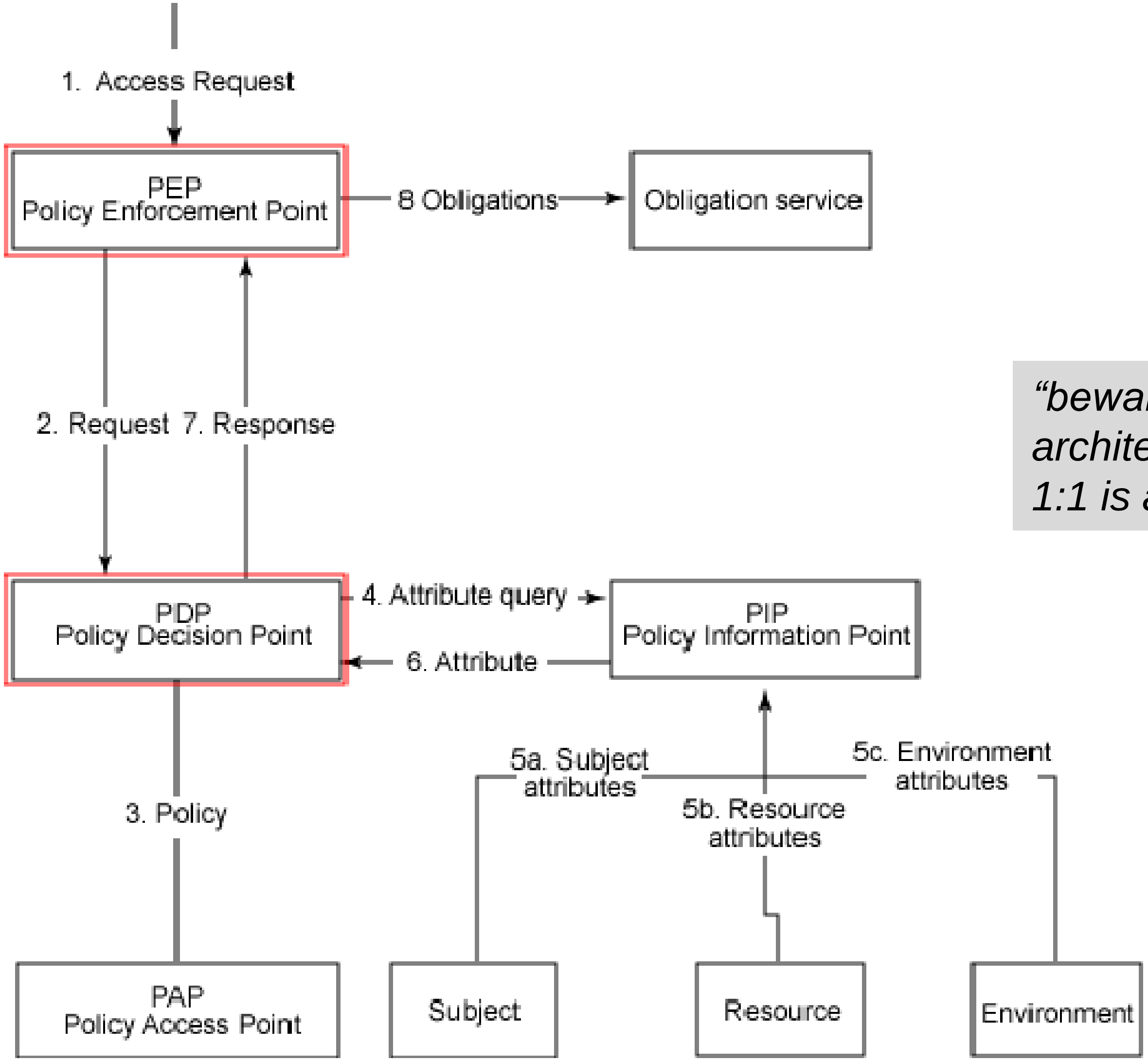
A multi-authority world

- Authorization elements (from OGSA 1.0)



Graphic: OGSA Working Group

Logical Elements in authorization



"beware that translating architecture to implementation 1:1 is a recipe for disaster "

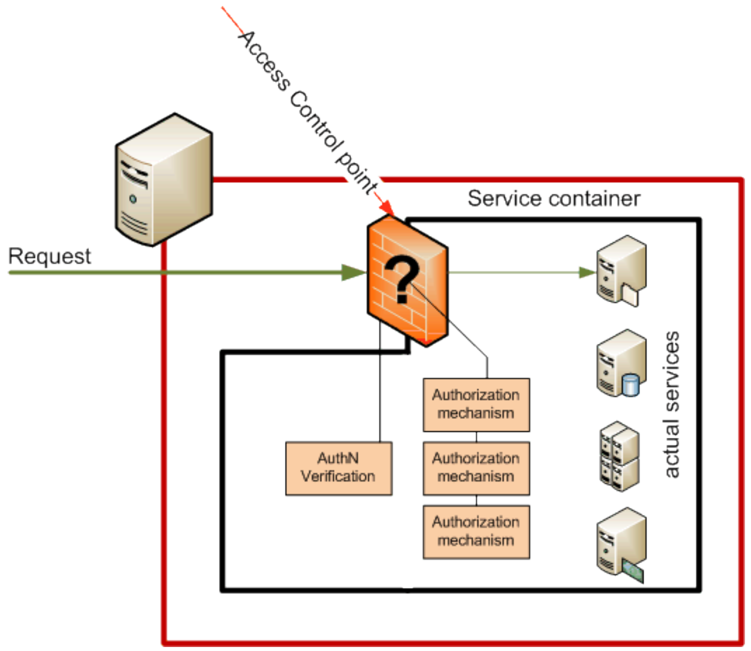
Control points

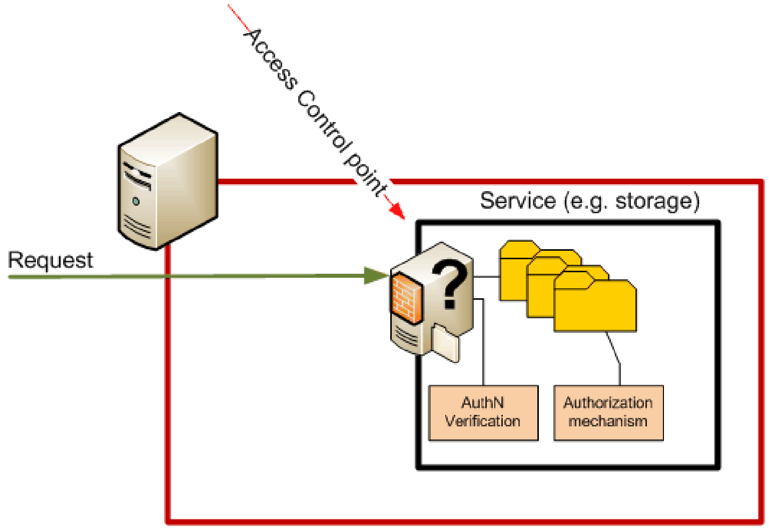
Container based

- Single control point
- Agnostic to service semantics

Service based

- Many control points
- Authorization can depend on requested action and resource

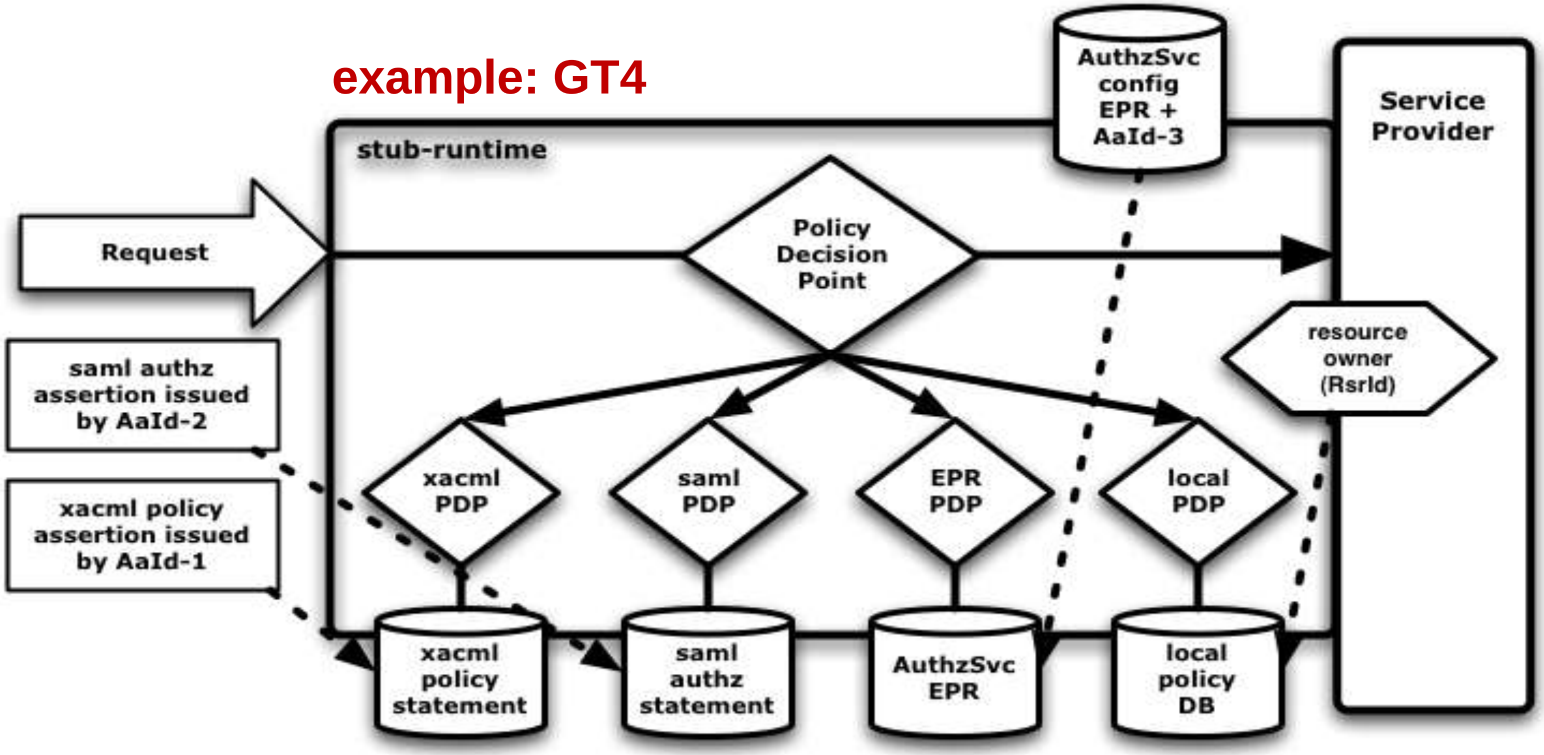




Frameworks

- (chain of) decision making modules controlling access
  - Loosely or tightly coupled to a service or container
  - Generic ‘library’, or tied into the service business logic

example: GT4



Graphic: Frank Siebenlist, Globus and ANL

Some framework implementations

- Globus Toolkit v4 Authorization Framework
- Site Access Control 'LCAS-LCMAPS' suite
- PRIMA-SAZ-GUMS-gPlazma suite

interop.
per 1/2009

- GridSite & GACL

- *gLite Authorization Framework (v2), under construction*
- …

… but don't forget 'native' service implementations

Implementation example: LCAS

- Enforcement point: service calls framework

gatekeeper.c

```
retval=lcas_get_fabric_authorization(user_cred_handle, lcas_lcmaps_request);
if (retval) {
  failure(FAILED_AUTHORIZATION, "LCAS failed authorization.");
}
```

- Framework executes (PDP(s))

/opt/glite/etc/lcas/lcas.db

```
# LCAS database/plugin list
pluginname=lcas_userban.mod,pluginargs=ban_users.db
pluginname=lcas_voms.mod,pluginargs="-vomsdir/etc/grid-security/vomsdir/ ..."
```

- And renders a decision

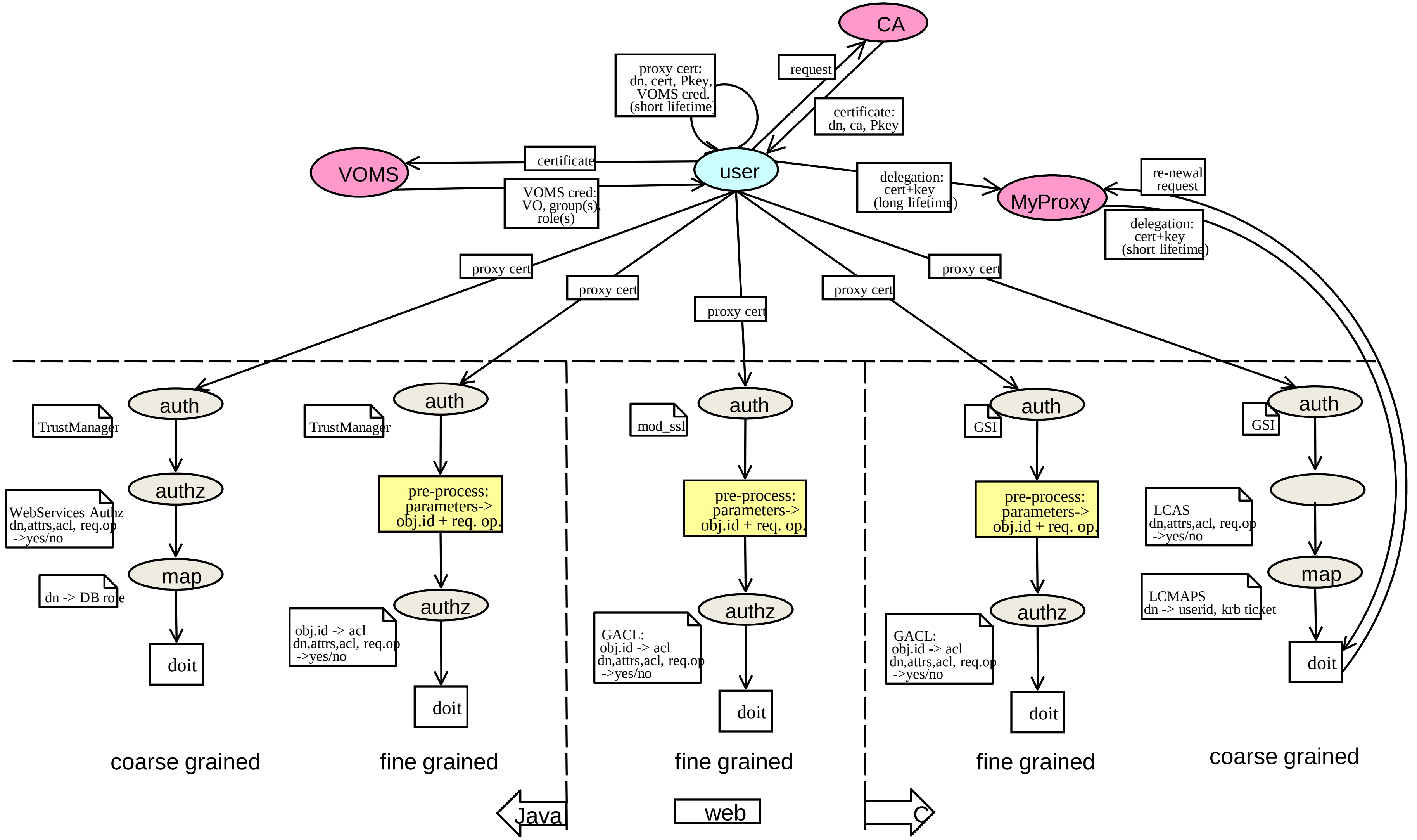
/var/log/globus-gatekeeper.log

```
LCAS 2: LCAS authorization request
LCAS 0:  lcas_userban.mod-plugin_confirm_authorization():
         checking banned users in /opt/glite/etc/lcas/ban_users.db
LCAS 0: 2009-04-14.18:13:40 :
         lcas_plugin_voms-plugin_confirm_authorization_from_x509():
         voms plugin succeeded
LCAS 0: lcas.mod-lcas_run_va(): succeeded
```

Different frameworks

- Each framework has
  - own calling semantics (but may interoperate at the back)
  - its own form of logging and auditing
- Most provide
  - Validity checking of credentials (all except ‘new’ gLite FW)
  - Access control based on Subject DN and VOMS FQANs
  - Subject DN banning capability
- And some have specific features, e.g.,
  - Capability to process arbitrary ‘XACML’ policies
  - Calling out to obtain new user attributes
  - Limiting the user executables, or proxy life time, …

Different targets, different implementations



Which framework to use?

Unfortunately, this question has no clear answer ☹

- Each service uses a particular framework
  - If you want a service, have to use its chosen framework
- Semantics are mostly different

However, there is progress!

- There is interop if you use a central service
  - Using an agreed ‘SAML2’ profile of ‘XACML2’ Req/Resp
  - See the interop section later on

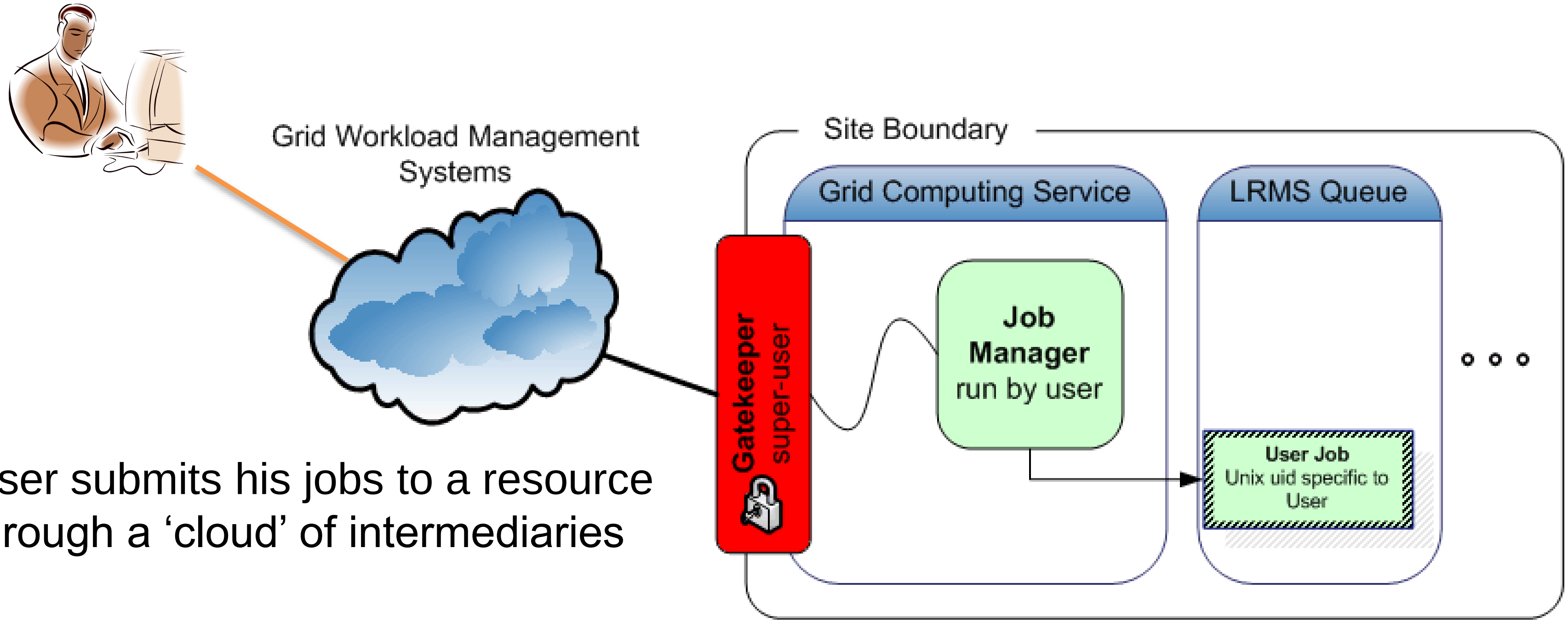
Let’s look at a few examples …

Example: running compute jobs

Access control: gatekeepers, gLExec, ban lists, and GACL

ACCESS CONTROL FOR COMPUTE

Job Submission Today



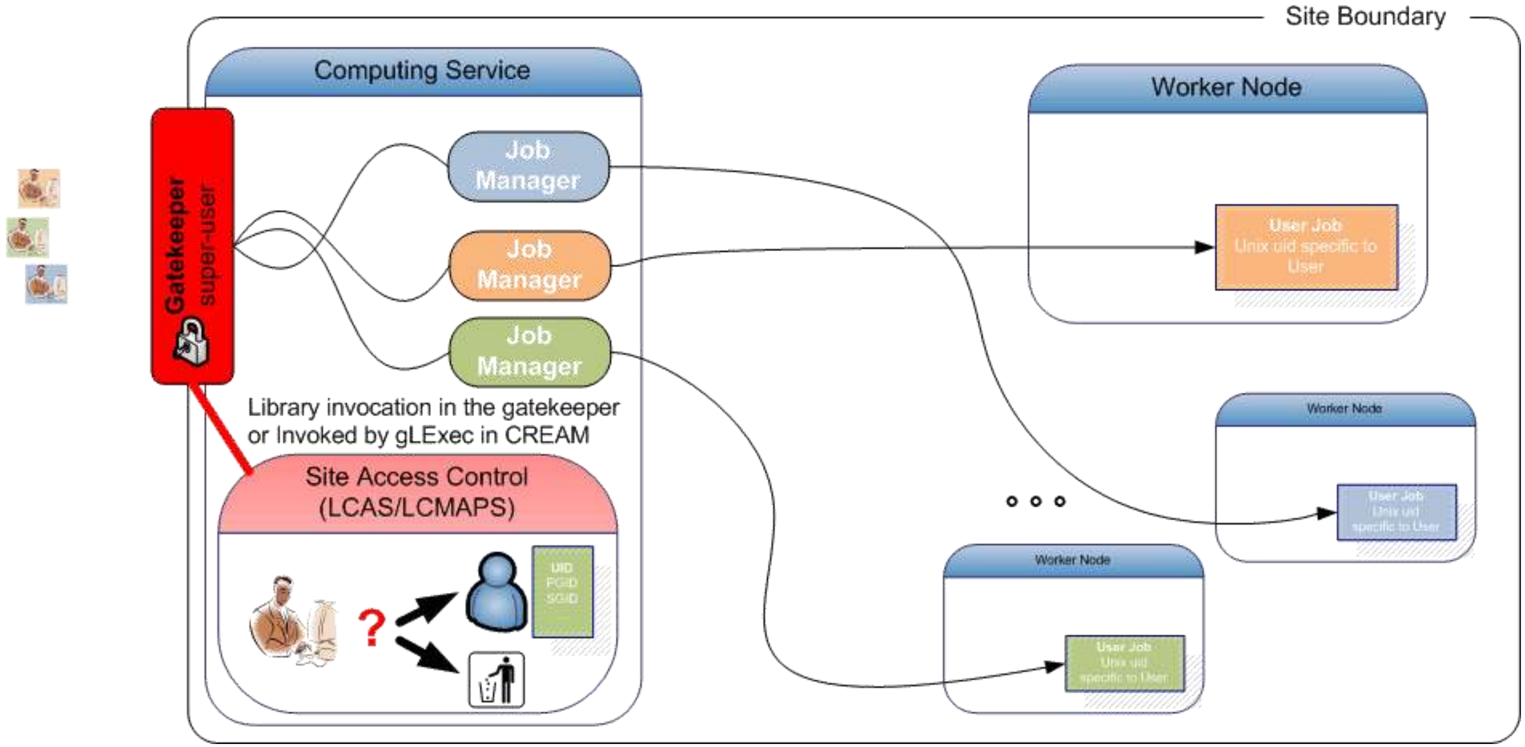
User submits his jobs to a resource through a 'cloud' of intermediaries

Direct binding of payload and submitted grid job

- job contains all the user's business
- access control is done at the site's edge
- inside the site, the user job has a specific, site-local, system identity

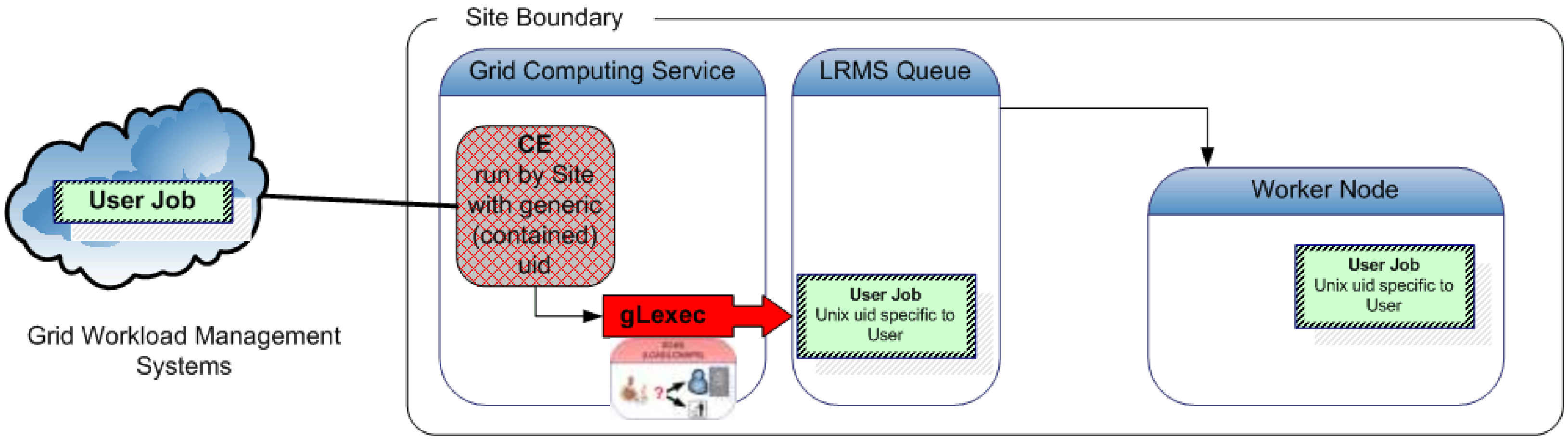
Access Control on the CE

- System access, written in C, means LCAS-LCMAPS
- Native or through ‘call-out hooks’ like in GT4



Similar services in C or using Unix

- gLite ‘CREAM’ submission



  - globus toolkit pre-WS GRAM in EGEE
  - gLExec (via CREAM and in late-binding pilot scenarios)
  - globus gridftp, gsi-openssh
  - DPM (LCAS-only in new version!)
  - SCAS (‘recursive’ invocation)

Decision attributes and obligations

Example: LCAS, the oldest - and simplest - one

- Input attributes
  - Submitting user certificate
  - VOMS FQANs are known
  - Target service is known
  - Action, partially known (‘RSL’ or executable name)
- Requested decisions
  - Is access granted?

- *LCMAPS needed for obligations, i.e., the unix account*

LCAS: basic authorization

- Pluggable authorization framework in C
  - Independent modules (‘shared objects’) called based on simple ‘boolean-AND’ policy description
- Decisions based on
  - Allowed user or VOMS FQAN list
  - Deny based on a separate ‘ban’ list with wildcards
  - GACL policy
  - Allowed-executable (‘RSL’ matching)
  - Time slots
  - L&B2-policy module

http://www.nikhef.nl/grid/lcaslcmaps/

LCAS example

/opt/glite/etc/lcas/lcas.db

```
# @(#)lcas.db
pluginname=lcas_userban.mod,pluginargs=ban_users.db
pluginname=lcas_voms.mod,pluginargs="-vomsdir/etc/grid-security/vomsdir/ ..."
```

/opt/glite/etc/lcas/ban_users.db

```
# @(#)ban_users.db
/DC=org/DC=example/CN=Sherlock Holmes
/DC=gov/DC=somelab/OU=CDF/CN=*
```

only DN c.q. FQAN used from ... /etc/grid-security/grid-mapfile

```
"/O=dutchgrid/O=users/O=nikhef/CN=David Groep" .pvier
"/O=dutchgrid/O=users/O=nikhef/CN=Oscar Koeroo" okoeroo
"/C=AT/O=AustrianGrid/OU=UIBK/OU=OrgUnit/CN=Name Suppressed" .esr
"/vlemed/Role=NULL/Capability=NULL" .vlemed
"/vlemed" .vlemed
"/vo.gear.cern.ch/Role=NULL/Capability=NULL" .poola
"/vo.gear.cern.ch" .poola
"/vo.gear.cern.ch/Role=lcgadmin/Capability=NULL" .troi
"/vo.gear.cern.ch/Role=lcgadmin" .troi
```

But notably different

- gLite WMS
  - Uses GACL libraries directly and exclusively
- Storage access control, e.g. DPM
  - Has built-in native handing of groups via POSIX ACLs expressed as VOMS FQANs
- Native GT4 pre-WS-GRAM and GridFTP
  - Has only a static DN map file
  - Unless configured to use LCAS-LCMAPS or PRIMA-GUMS
- …

gLite WMS access control: GACL

/opt/glite/etc/ glite_wms_wmproxy.gacl

```
<gacl version="0.0.1">
  <entry>
    <voms>
      <fqan>lofar/ROLE=admin</fqan>
    </voms>
    <allow><exec/></allow>
  </entry>
  ...
  <entry>
    <voms>
      <fqan>lsgrid</fqan>
    </voms>
    <allow><exec/></allow>
  </entry>
  <entry>
    <person>
      <dn>/DC=org/DC=example/O=HEP/O=PKU/OU=PHYS/CN=Some Person</dn>
    </person>
    <deny><exec/></deny>
  </entry>
</gacl>
```

GridSite and LCAS can do GACL as well, though ...

GUMS access control

GUMS is a central-service only mapping service

- Database with a ‘site’ dump of the VO membership
- Tools to manipulate that database
- e.g. banning a user or a VO

https://twiki.grid.iu.edu/bin/view/Security/GUMS--DevelopmentandAdditions

```
# an individual that is not a VO member
/DC=org/DC=doegrids/OU=People/CN=Jay Packard 335585,

# an invidual from any VO
/DC=org/DC=doegrids/OU=People/CN=Jay Packard 335585, .*

# or an individual from the Atlas production role
/DC=org/DC=doegrids/OU=People/CN=Jay Packard 335585, //atlas/usatlas/Role=production.*
```

- *please hold for a central service based on LCAS-LCMAPS...*

Credential mapping

Running jobs

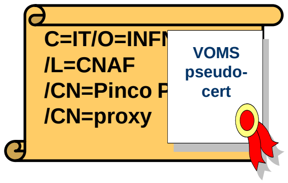
Long-running jobs and MyProxy

Addressing late-binding with gLExec

TO THE UNIX WORLD

To the Unix world: Problem

grid identity



(X509, VOMS)
/dc=org/dc=example/CN=John Doe

translate

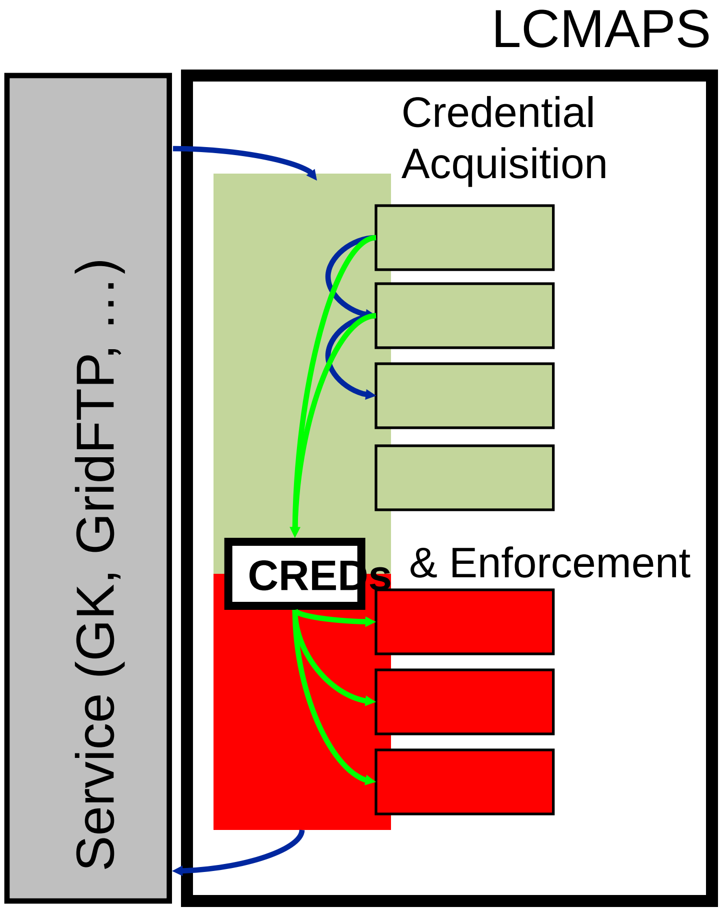
```
pvier001:x:43401:2029:PoolAccount VL-e P4 no.1:/home/pvier001:/bin/sh
```

> Unix does not talk Grid, so
translation is needed between grid and local identity

1. this translation has to happen somewhere
2. something needs to do that

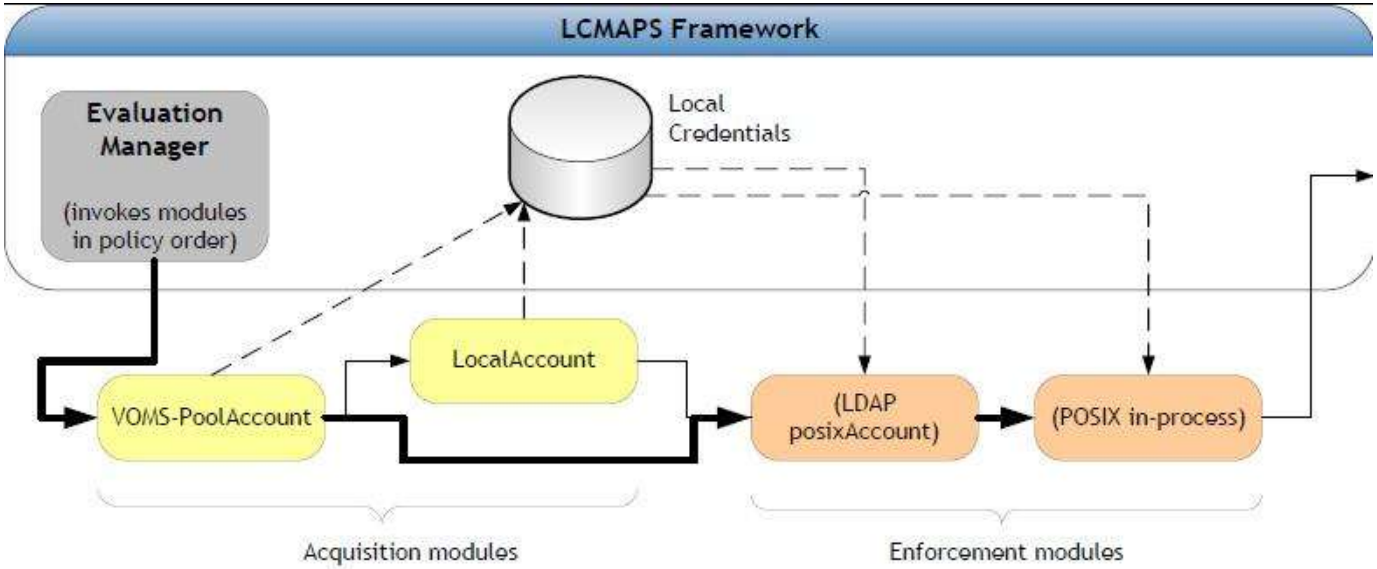
To the Unix world: LCMAPS

- Again a pluggable framework
  - Separated in two phases, since *#1 may require 'root' privileges, that a plug-in in #2 might drop*
  - **Acquisition** collect attributes and obligations
  - **Enforcement** make all obligations honoured interact with local unix system
- Modules are shared objects



http://www.nikhef.nl/grid/lcaslcmaps/

LCMAPS modules



- Acquisition
  (voms)local{account,group}, (voms)pool{account,group}, GUMS, verify-proxy, scas-client
- Enforcement
  posix_enf, ldap_enf, afs, jobRepository

LCMAPS configuration example

/opt/glite/etc/lcmaps/lcmaps-scas.db

```
# LCMAPS config file for glexec generated by YAIM

vomslocalgroup = "lcmaps_voms_localgroup.mod ...”
vomslocalaccount = "lcmaps_voms_localaccount.mod ...”
vomspoolaccount = "lcmaps_voms_poolaccount.mod ...”
localaccount = "lcmaps_localaccount.mod"
               " -gridmapfile /etc/grid-security/grid-mapfile”
poolaccount = "lcmaps_poolaccount.mod"
              " -override_inconsistency"
              " -gridmapfile /etc/grid-security/grid-mapfile"
              " -gridmapdir /share/gridmapdir"
good = "lcmaps_dummy_good.mod”

# Policies: DN-local -> VO-static -> VO-pool -> DN-pool
static_account_mapping:
localaccount -> good

voms_mapping:
vomslocalgroup -> vomslocalaccount
vomslocalaccount -> good | vomspoolaccount

classic_poolaccount:
poolaccount -> good
```

Policy sequence depends on the service!

MyProxy

Renewal daemons

What About VOMS

LONG RUNNING JOBS

MyProxy in EEGE

- EGEE security based on proxy certificates
  - often carrying VOMS attribute certificates

- MyProxy used for several purposes:
  - Solution for portals (P-GRADE, Genius)
    - a common way of using MyProxy
  - Long-running jobs and data transfers
    - credential renewal

http://myproxy.ncsa.uiuc.edu/

Slides based on: Ludek Matyska and Daniel Kouril, CESNET

Long-running Jobs

- Jobs require valid credentials
  - e.g. to access GridFTP data repositories on the user‘s behalf
  - these operations must be secured, using the users‘ credentials
- Job's lifetime can easily exceed the lifetime of a proxy
  - consider waiting in the queues, possible resubmissions, computation time, data transfers, etc.
  - also VOMS certificates have limited lifetime
- Impossible to submit a job with sufficiently long credentials
  - the overall job lifetime not known in advance
  - violation of the meaning of short-time proxies
  - increased risk when the credential is stolen
  - might be unacceptable for the end resources
- *How to provide jobs with a valid short-lived credential throughout their run?*

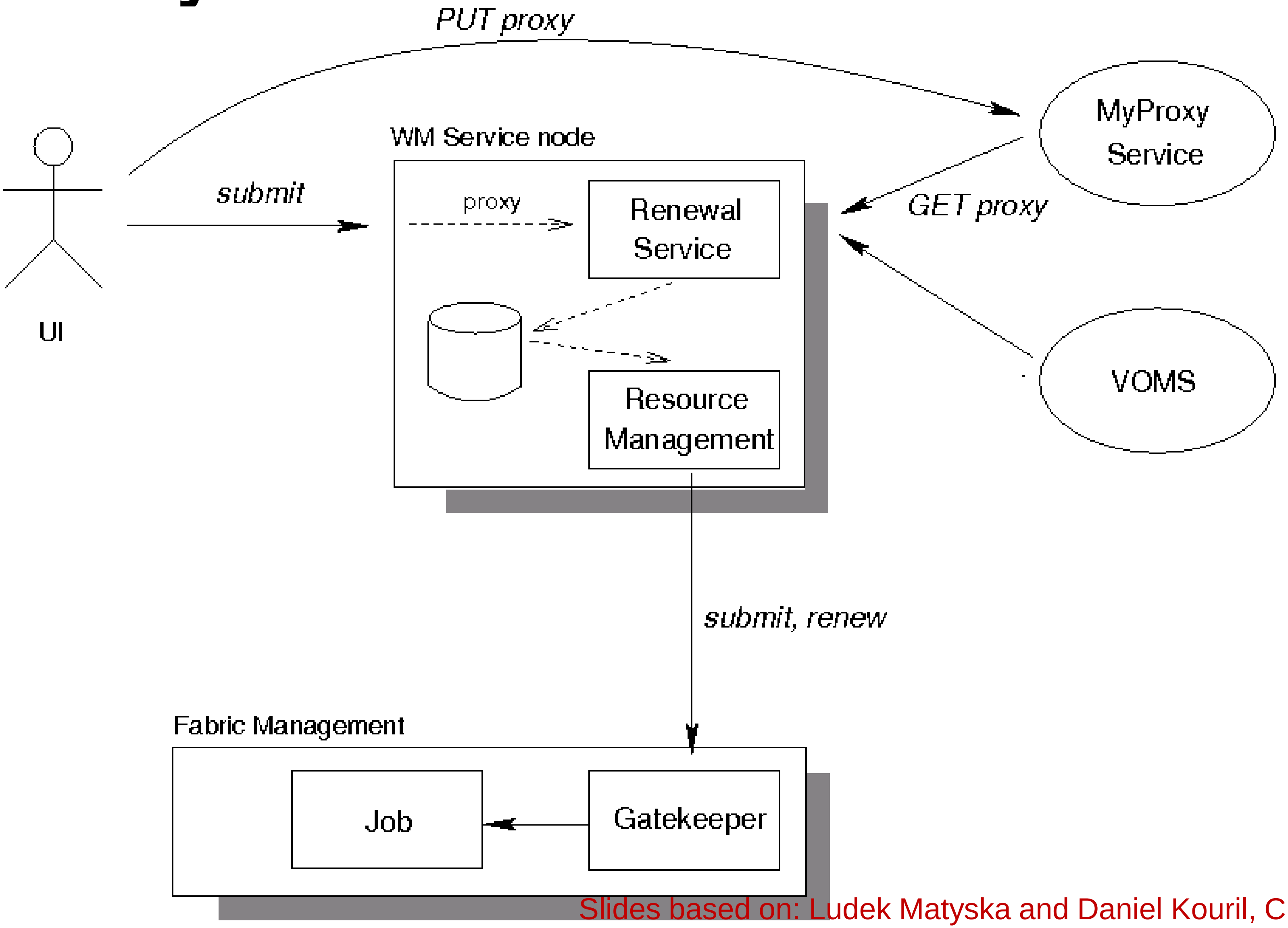
Slides based on: Ludek Matyska and Daniel Kouril, CESNET

Proxy Renewal Service

- Periodical renewal of credentials
  - maintains a list of jobs' proxy certificates to be kept valid
  - using MyProxy repository
    - server specified by user in the job description
    - uses the renewal mode
    - authenticates using the WMS credential AND authorizes using the proxy being renewed
  - Support for renewal of VOMS attributes
- Part of the broker node (WMS)
  - A proxy of a job is registered upon submission
  - It is renewed whenever it is going to expire
    - several attempts done until renewal succeeds
  - After renewal a new proxy is pushed to the computing resource, where the job is running
  - After the job completion the proxy is unregistered

Slides based on: Ludek Matyska and Daniel Kouril, CESNET

Proxy Renewal Service



Slides based on: Ludek Matyska and Daniel Kouril, CESNET

Proxy Renewal Service

- Ensures that jobs always have a valid short-time proxy
- Users have full control over their proxies and renewal
  - Using the MyProxy repository
- Support for VOMS
- All operations are logged
  - allows an audit
- Stolen credentials can't be renewed easily
  - the WMS credential are necessary for renewal
- An older (still valid) proxy must be available for renewal
  - reduces the risk when services are compromised
- Developed in EU Datagrid, in production use in EGEE

Slides based on: Ludek Matyska and Daniel Kouril, CESNET

MyProxy and Trust Establishment

- Relationship between MyProxy and its client is crucial
  - clients must be authorized to access the repository
- So far trust based on a static configuration
  - each service and client must be listed
  - regular expressions aren‘t sufficient
  - a subject name of a service must be added on each change or addition
- VOMS support introduced recently
  - generated by needs of EGEE
  - allows to specify VOMS attributes (roles, groups) instead of specifying identity
  - requires adding service certificates to VOMS machinery

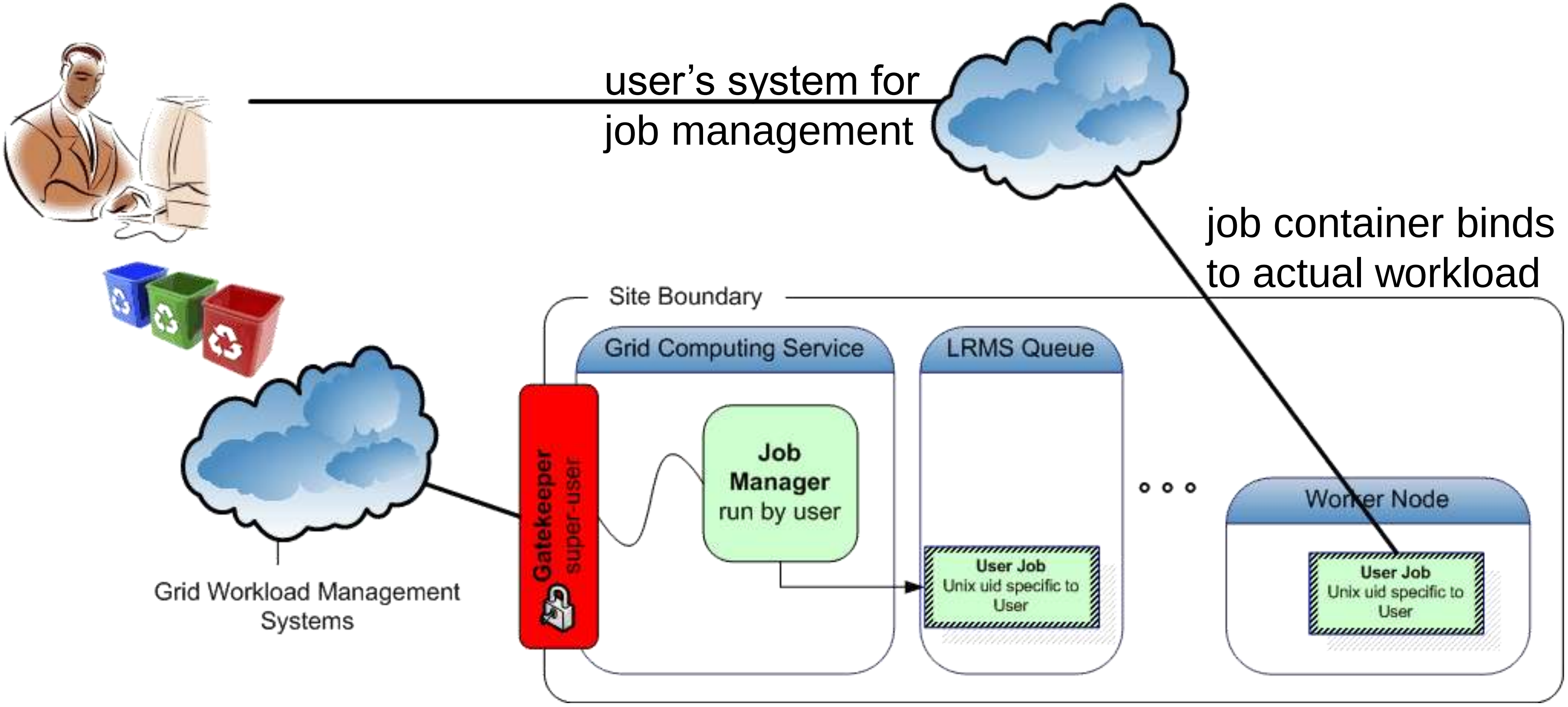
Slides based on: Ludek Matyska and Daniel Kouril, CESNET

Pilot jobs

Impact on sites

LATE BINDING

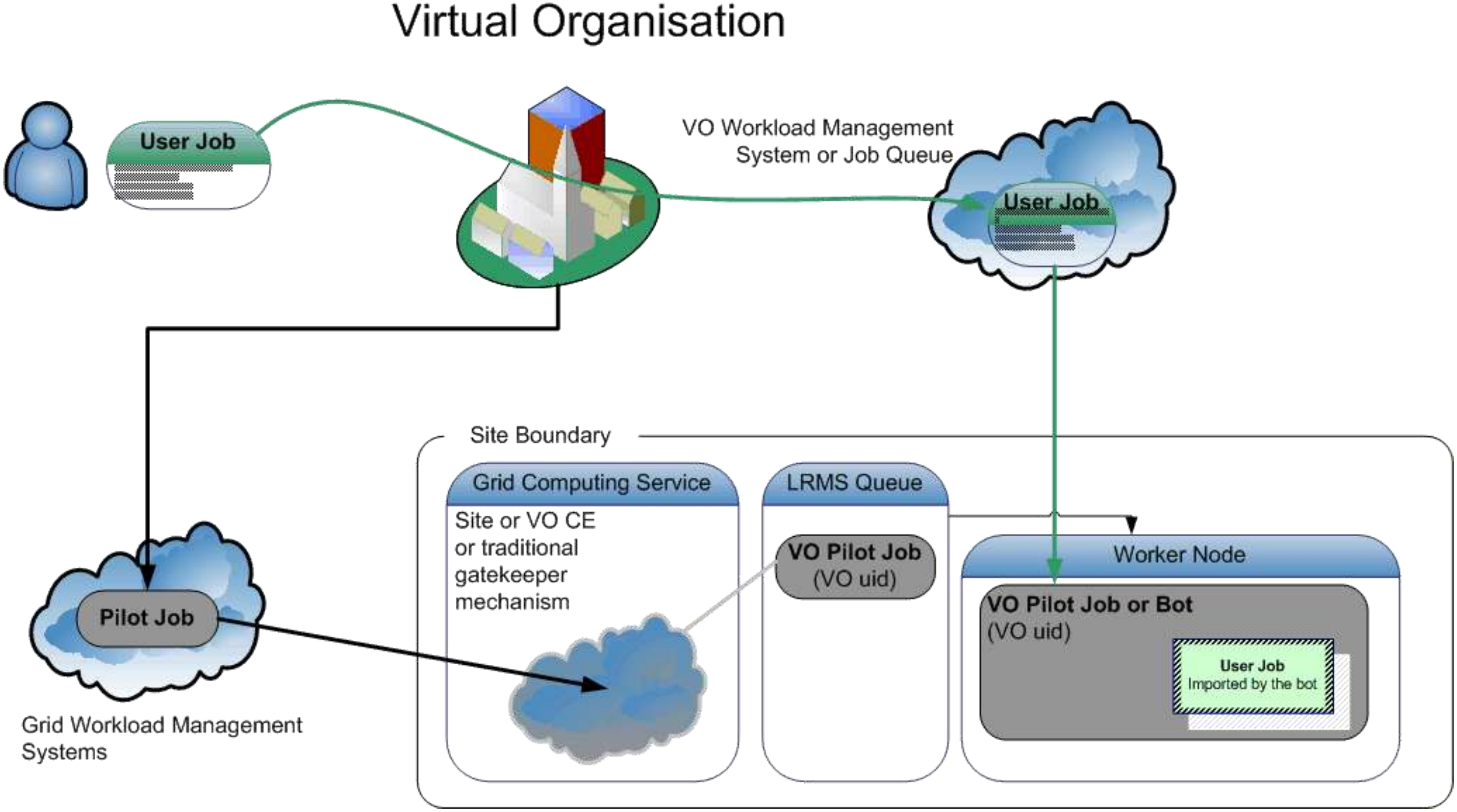
Binding Late


user's system for job management
job container binds to actual workload
Site Boundary
Grid Computing Service
LRMS Queue
Job Manager run by user
Gatekeeper
super-user
User Job
Unix uid specific to User
Worker Node
User Job
Unix uid specific to User
Grid Workload Management Systems

Late binding of work load using 'pilot jobs'

- generic job containers are sent, which can verify the 'surroundings'
- retrieve payload from a repository 'elsewhere'
- *if the repository is run by the user, on a per-user bases, then it is likely that it's the users' payload – if communication is secure*

Multi-User Pilot Jobs

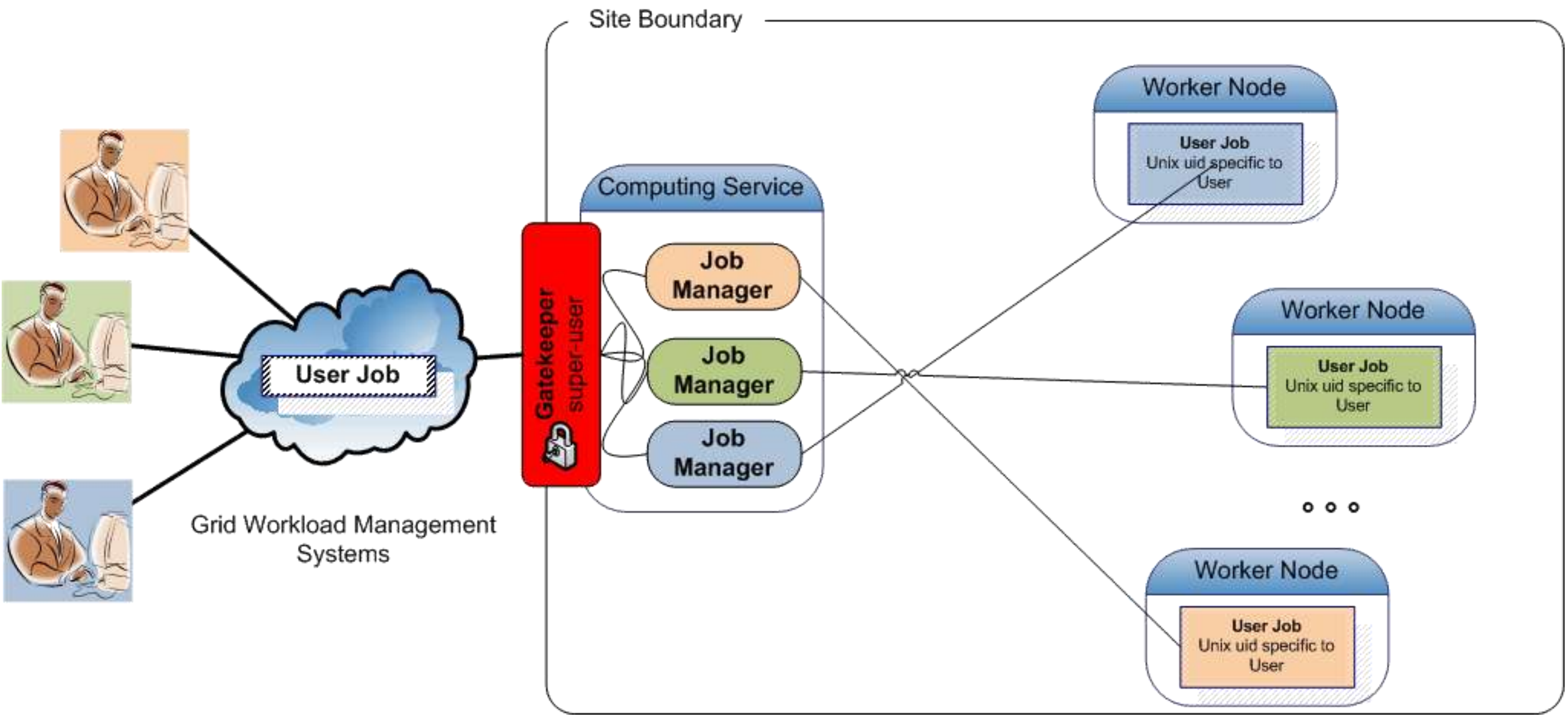

Virtual Organisation
User Job
VO Workload Management System or Job Queue
User Job
Site Boundary
Grid Computing Service
Site or VO CE or traditional gatekeeper mechanism
LRMS Queue
VO Pilot Job (VO uid)
Worker Node
VO Pilot Job or Bot (VO uid)
User Job Imported by the bot
Pilot Job
Grid Workload Management Systems

What if the user 'outsources' the running of the pilot jobs?

- then whoever runs the pilot jobs, will run workload for multiple users
- but the site only grants access to the 'service provider' (VO) …

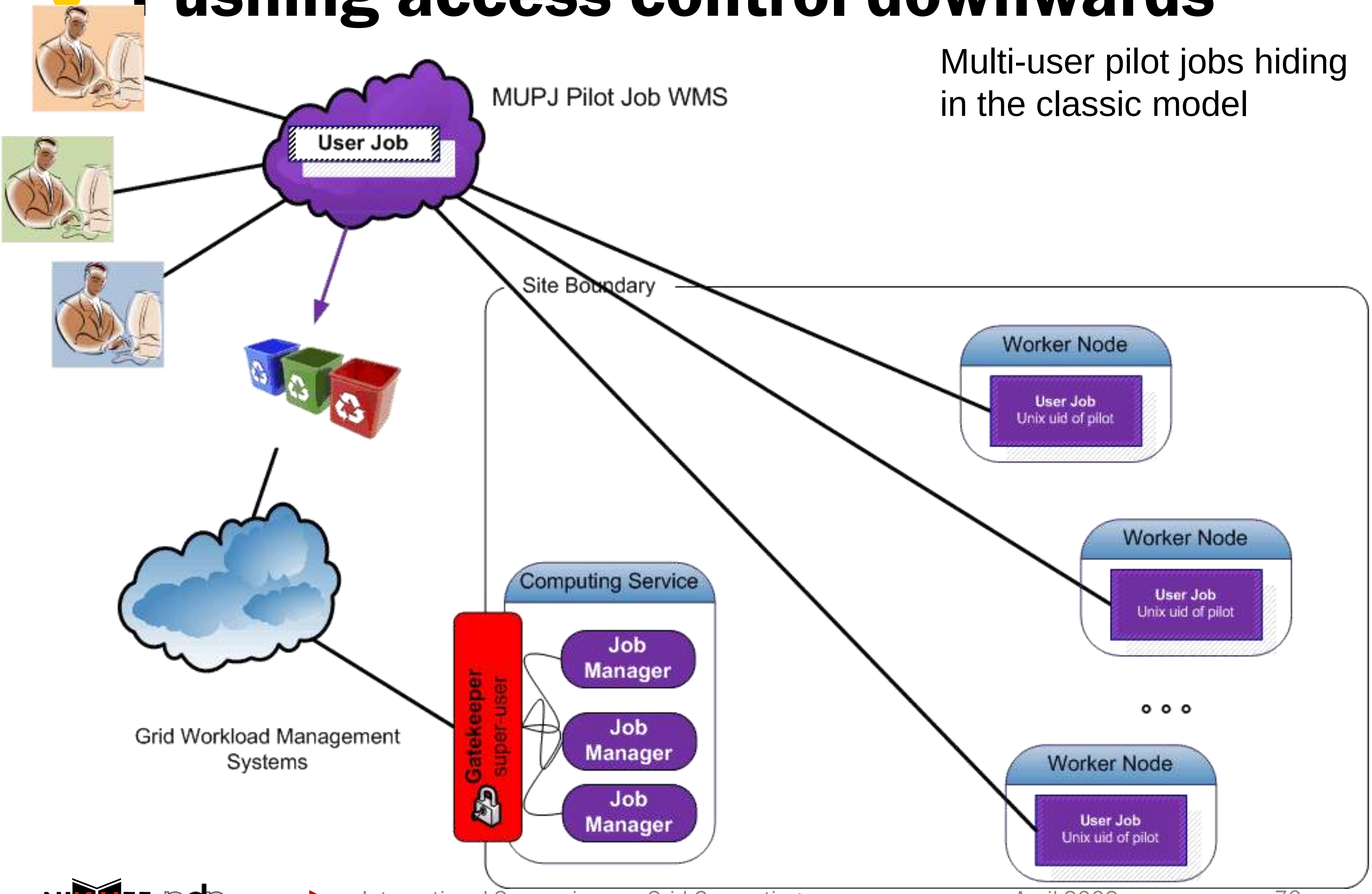
Pushing access control downwards

Classic model



Pushing access control downwards

Multi-user pilot jobs hiding in the classic model



MUPJ security issues

With multi users use a common pilot job deployment Users, by design, will use the same account at the site

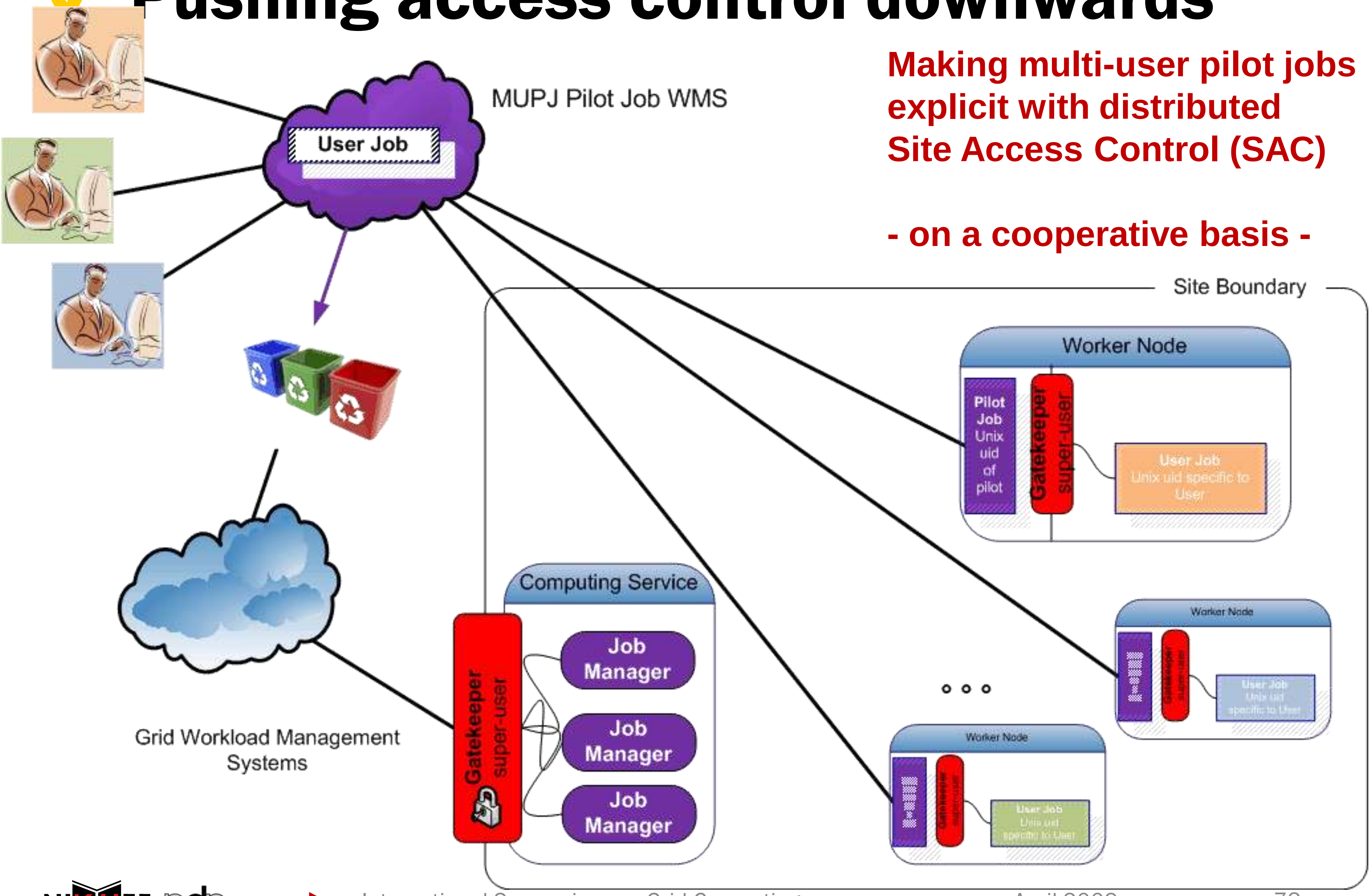
- Accountability
  no longer clear at the site who is responsible for activity
- Integrity
  a compromise of any user using the MUPJ framework ‘compromises’ the entire framework

  the framework can’t protect itself against such compromise
  unless you allow change of system uid/gid
- Site access control policies are ignored
- … and several more …

Pushing access control downwards

Making multi-user pilot jobs explicit with distributed Site Access Control (SAC)

- on a cooperative basis -



Implementing distributed SAC

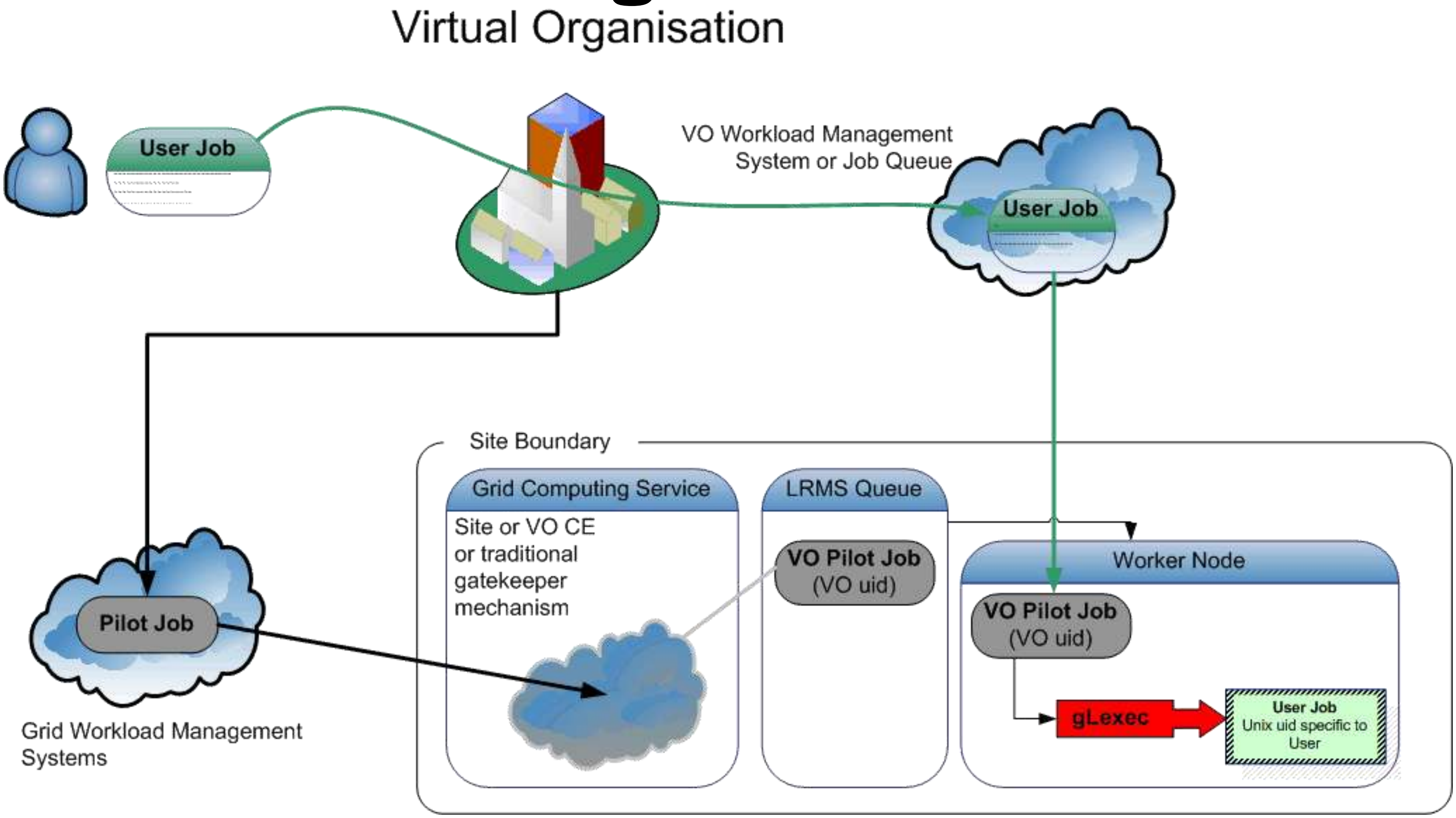
Component 1: gLExec

a thin layer
to change Unix domain credentials
based on grid identity and attribute information

you can think of it as:

- ‘a replacement for the gatekeeper’
- ‘a *griddy* version of Apache’s `suexec`’
- ‘a program wrapper around LCAS, LCMAPS or GUMS’

Pilot Jobs and gLExec



On success: gLExec will set the uid/gid to the new user's job and execute it

On failure: gLExec returns with an error, and pilot job can terminate or obtain other user's job

gLExec deployment modes

- Identity Mapping Mode – ‘just like on the CE’
  - have the VO query (and by policy honour) all site policies
  - actually change uid based on the true user’s grid identity
  - enforce per-user isolation and auditing using uids and gids
  - requires gLExec to have *setuid* capability

- Non-Privileged (‘Logging Only’) Mode – declare only
  - have the VO query (and by policy honour) all site policies
  - do not actually change uid: no isolation or auditing per user
  - the gLExec invocation will be logged, with the user identity
  - does not require setuid powers – job keeps running in pilot space

- ‘Empty Shell’ – do nothing but execute the command…

Identity change

Let’s assume you make it *setuid*. Fine. Where to map to:

- To a shared set of common pool accounts
  - Uid and gid mapping on CE corresponds to the WN
  - Requires SCAS or shared state (gridmapdir) directory
  - Clear view on who-does-what

- To a per-WN set of pool accounts
  - No site-wide configuration needed
  - Only limited (and generic) set of pool uids on the WN
  - Need only as many pool accounts as you have job slots
  - Makes cleanup easier, ‘local’ to the node
- *Or something in between … e.g. 1 pool for CE other for WN*

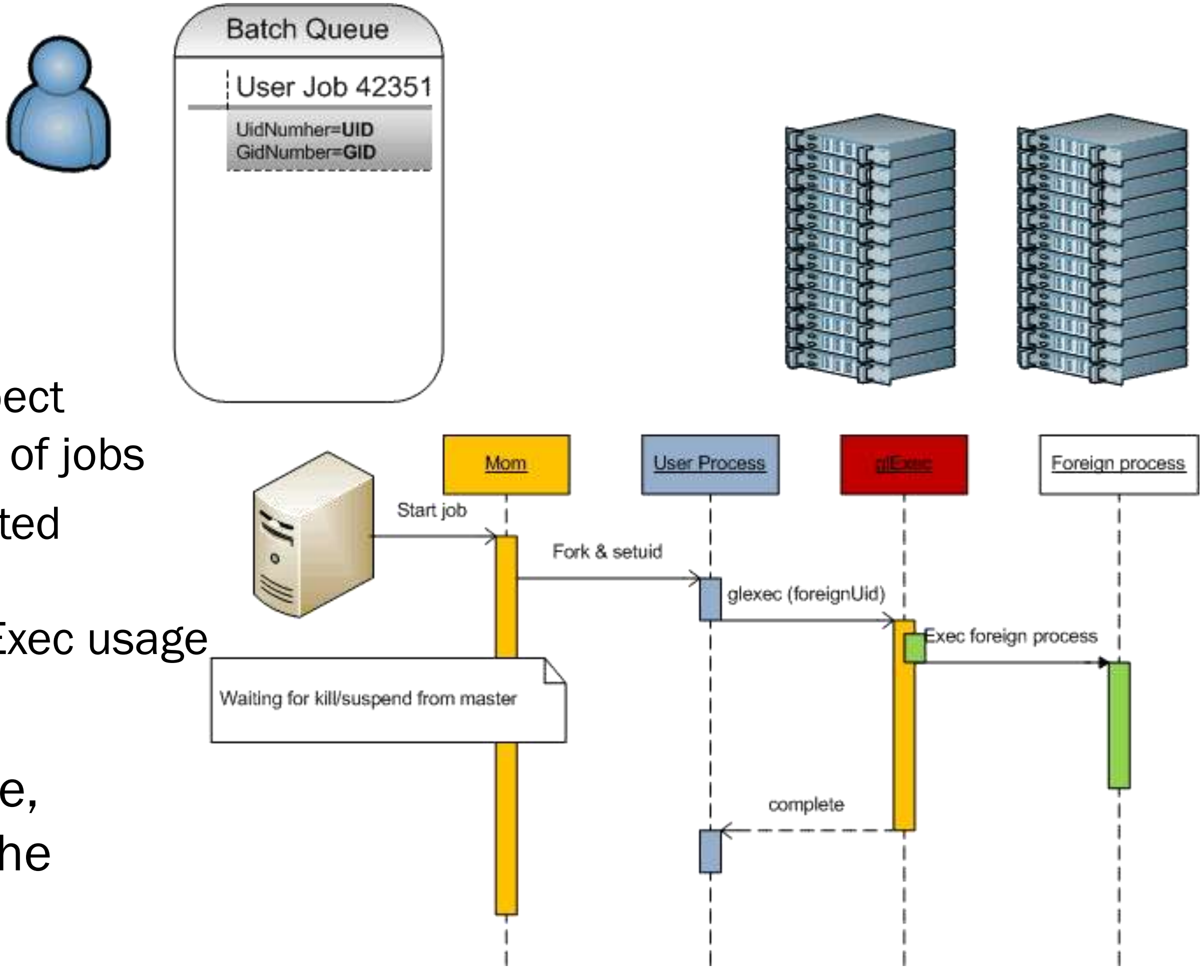
But if it is not setuid, it cannot isolate & protect the pilot.

Batch system and OS compatibility

How does gLExec affect the basic functions of a batch system?

1. Job Submission
2. Job Suspend/Resume
3. Job Kill

4. CPU time accounting
   - > No change with respect to current behaviour of jobs
   - > Times are accumulated on wait and collated with the gLExec usage



by keeping the process tree, gLExec is transparent for the tested batch systems

tests based on work by Ulrich Schwickerath

But all pieces should go together

1. *glexec on the worker-node deployment*

2. way to keep the pilot jobs submitters to their word
   > mainly: monitor for compromised pilot submitters credentials

   > system-level auditing of the pilot jobs,
   > but auditing data on the WN is useful for incident investigations only

3. ‘internal accounting should be done by the VO’
   > the regular site accounting mechanisms are via the batch system, and these will see the pilot job identity

   > the site can easily show from those logs the usage by the pilot job

   > making a site do accounting based glexec jobs is non-standard, and requires non-trivial effort

Centralizing Authorization in the site in gLite today

GUMS and SAZ: long time experience

Interoperability through common protocols

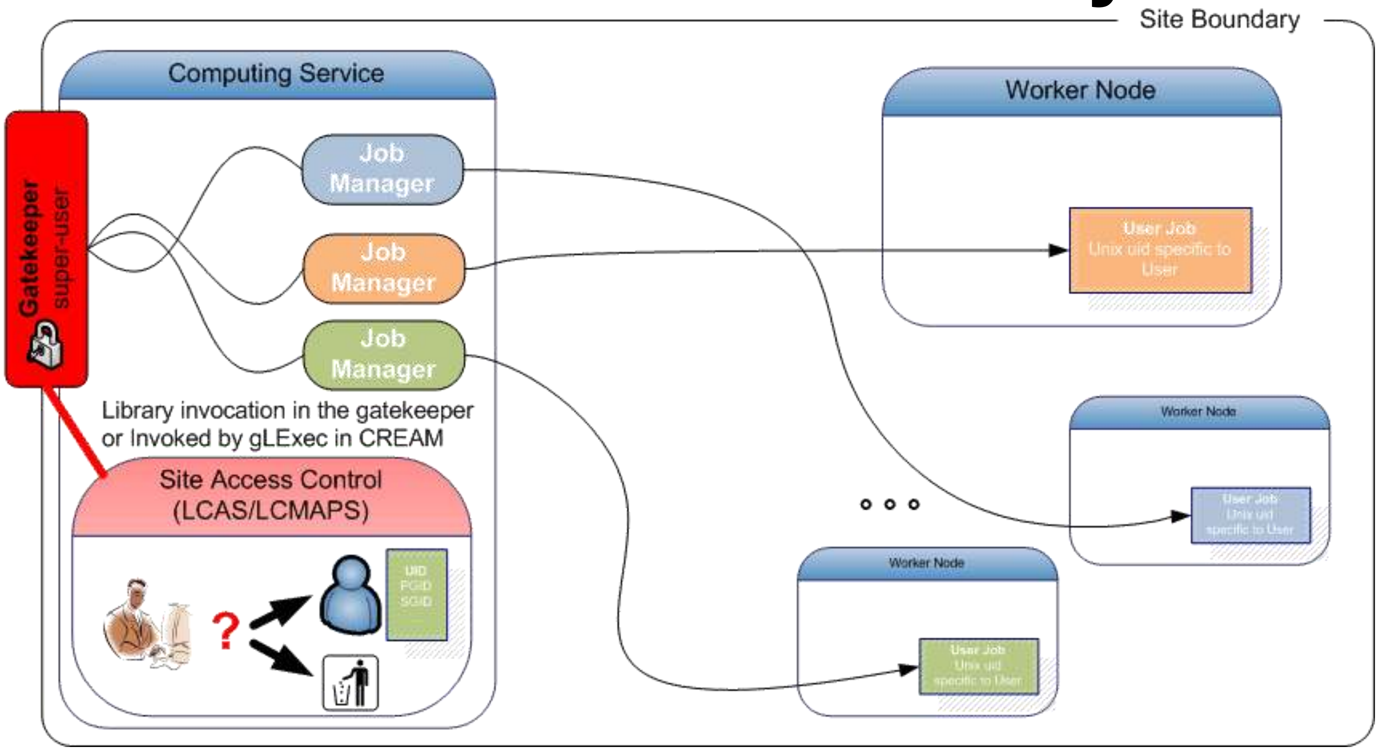
TOWARDS CENTRAL CONTROL

What Happens to Access Control?

So, as the workload binding get pushed deeper into the site, access control by the site has to become layered as well …

… how does that affect site access control software and its deployment ?

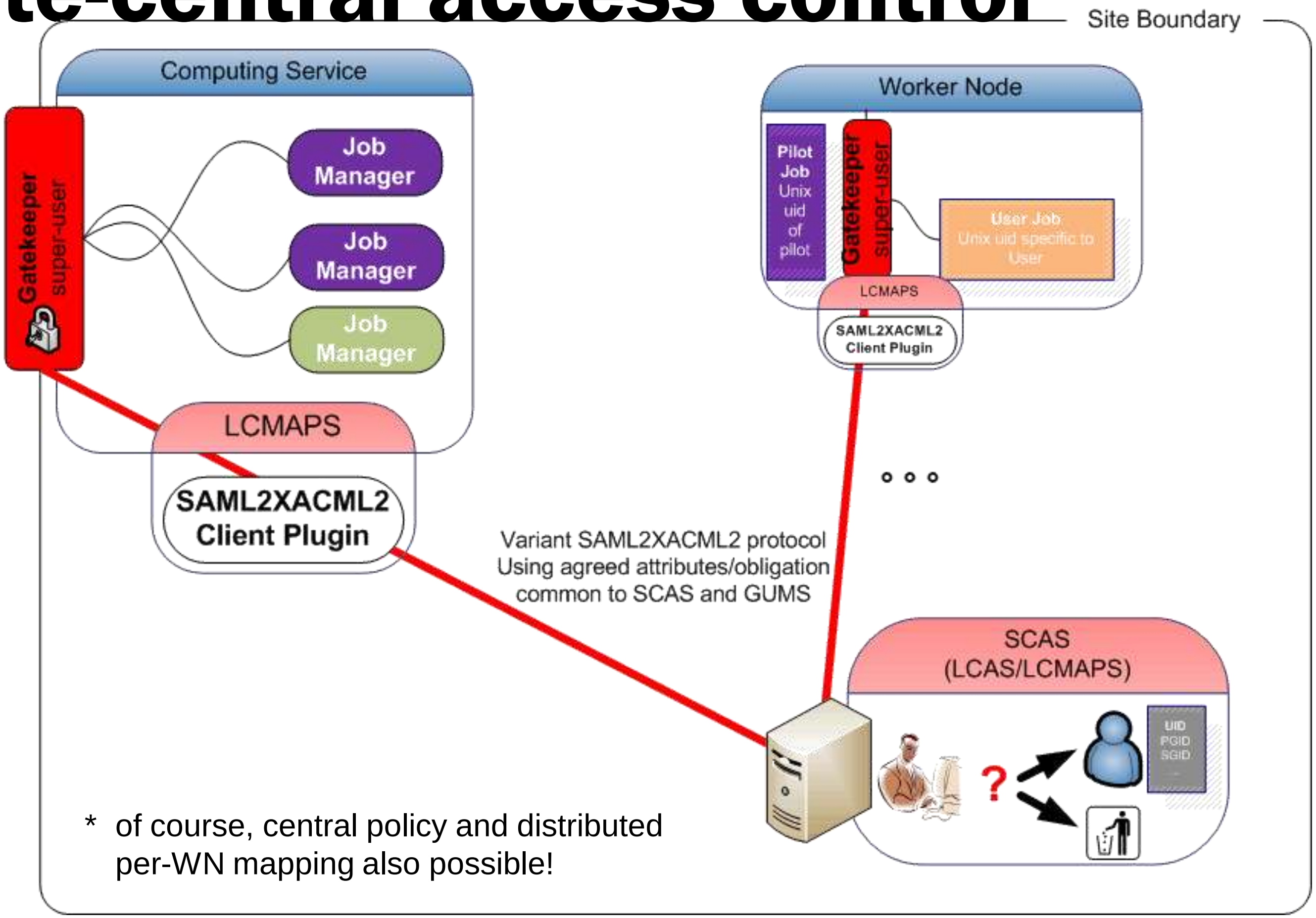
Site Access Control today


Site Boundary
Computing Service
Gatekeeper
super-user
Job Manager
Job Manager
Job Manager
Library invocation in the gatekeeper or Invoked by gLExec in CREAM
Site Access Control (LCAS/LCMAPS)
?
UID
PGID
SGID
Worker Node
User Job
Unix uid specific to User
Worker Node
User Job
Unix uid specific to User
Worker Node
User Job
Unix uid specific to User
o o o

PRO already deployed
no need for external components, amenable to MPI

CON when used for MU pilot jobs, all jobs run with a single identity
end-user payload can back-compromise pilots, and cross-infect other jobs
incidents impact large community (everyone utilizing the MUPJ framework)

Site-central access control



* of course, central policy and distributed per-WN mapping also possible!

PRO single unique account mapping per user across whole farm, CE, and SE*
can do instant banning and access control in a single place
protocol profile allows interop between SCAS and GUMS (but no others!)

CON replicated setup for redundancy needed for H/A needs more boxes
cannot yet do credential validation (formalistic issues with the protocol)

Centralizing decentralized SAC

Supporting consistent

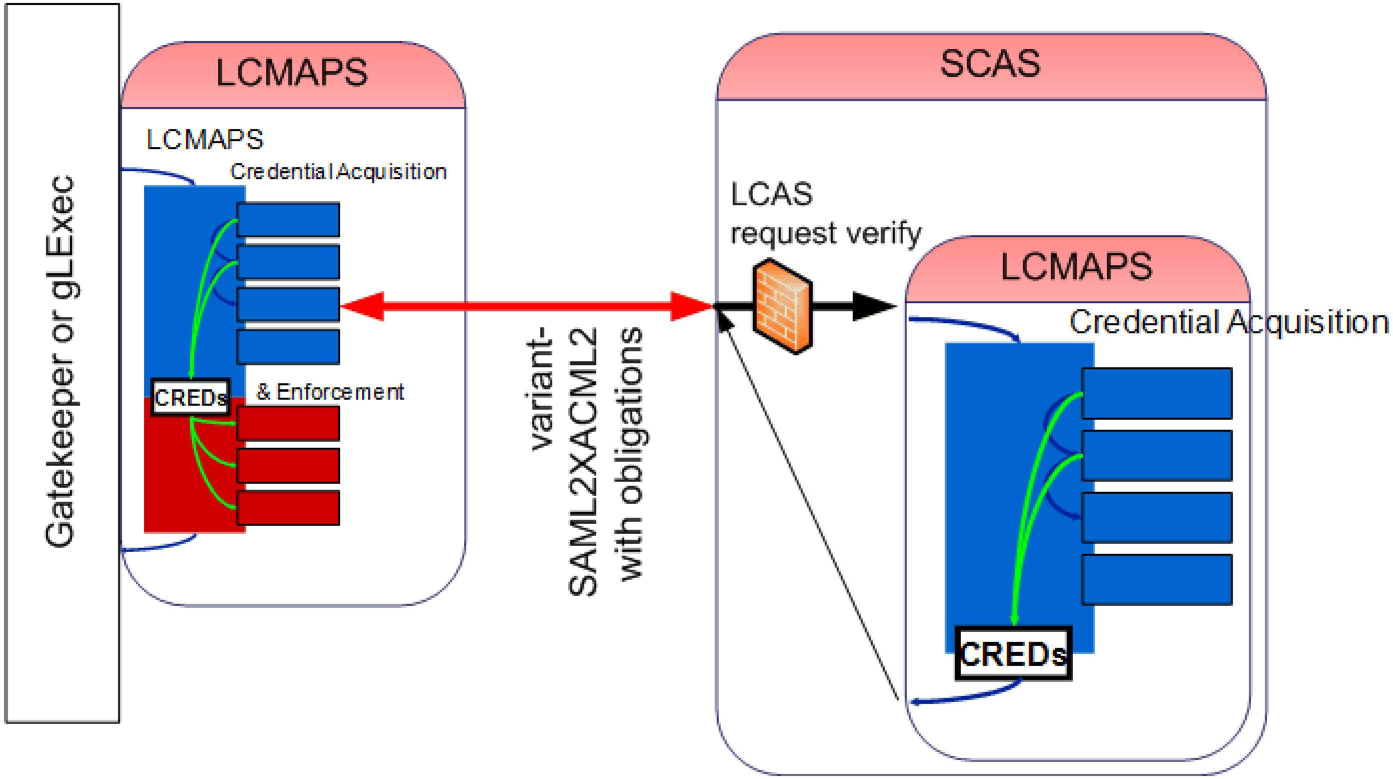
- policy management
- mappings (if the are not WN-local)
- banning

via the

Site Central Authorization Service SCAS

- network wrapper around LCAS and LCMAPS
- it's a variant-SAML2XAML2 client-server
- it is itself access controlled

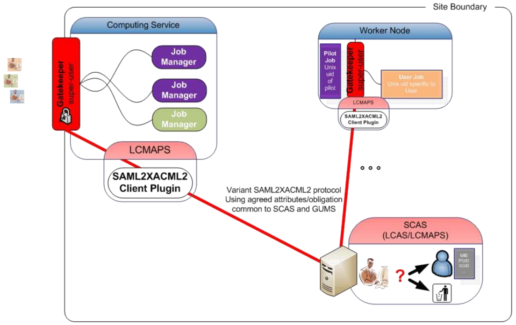
SCAS: LCMAPS in the distance



- Application links LCMAPS dynamically or statically, or includes Prima client
- Local side talks to SCAS using a variant-SAML2XACML2 protocol
  - with agreed attribute names and obligation between EGEE/OSG
  - remote service does acquisition and mappings
  - both local, VOMS FAQN to uid and gids, etc.
- Local LCMAPS (or application like gLExec) does the enforcement

Talking to SCAS

- From the CE
  - Connect to the SCAS using the CE host credential
  - Provide the attributes & credentials of the service requester, the action (“submit job”) and target resource (CE) to SCAS
  - Using common (EGEE+OSG+GT) attributes
  - Get back: yes/no decision and uid/gid/sgid obligations
- From the WN with gLExec
  - Connect to SCAS using the credentials **of the pilot job submitter**
    An extra control to verify the invoker of gLExec is indeed an authorized pilot runner


Site Boundary
Computing Service
Gatekeeper super-user
Job Manager
Job Manager
Job Manager
LCMAPS
SAML2XACML2 Client Plugin
Worker Node
Pilot Job Unix uid of pilot
Gatekeeper super-user
User Job Unix uid specific to User
LCMAPS
SAML2XACML2 Client Plugin
Variant SAML2XACML2 protocol Using agreed attributes/obligation common to SCAS and GUMS
SCAS (LCAS/LCMAPS)
?
UID PGID SGID

  - Provide the attributes & credentials of the service requester, the action (“run job now”) and target resource (CE) to SCAS
  - Get back: yes/no decision and uid/gid/sgid obligations
- The obligations are now coordinated between CE and WNs

authz-interop.org

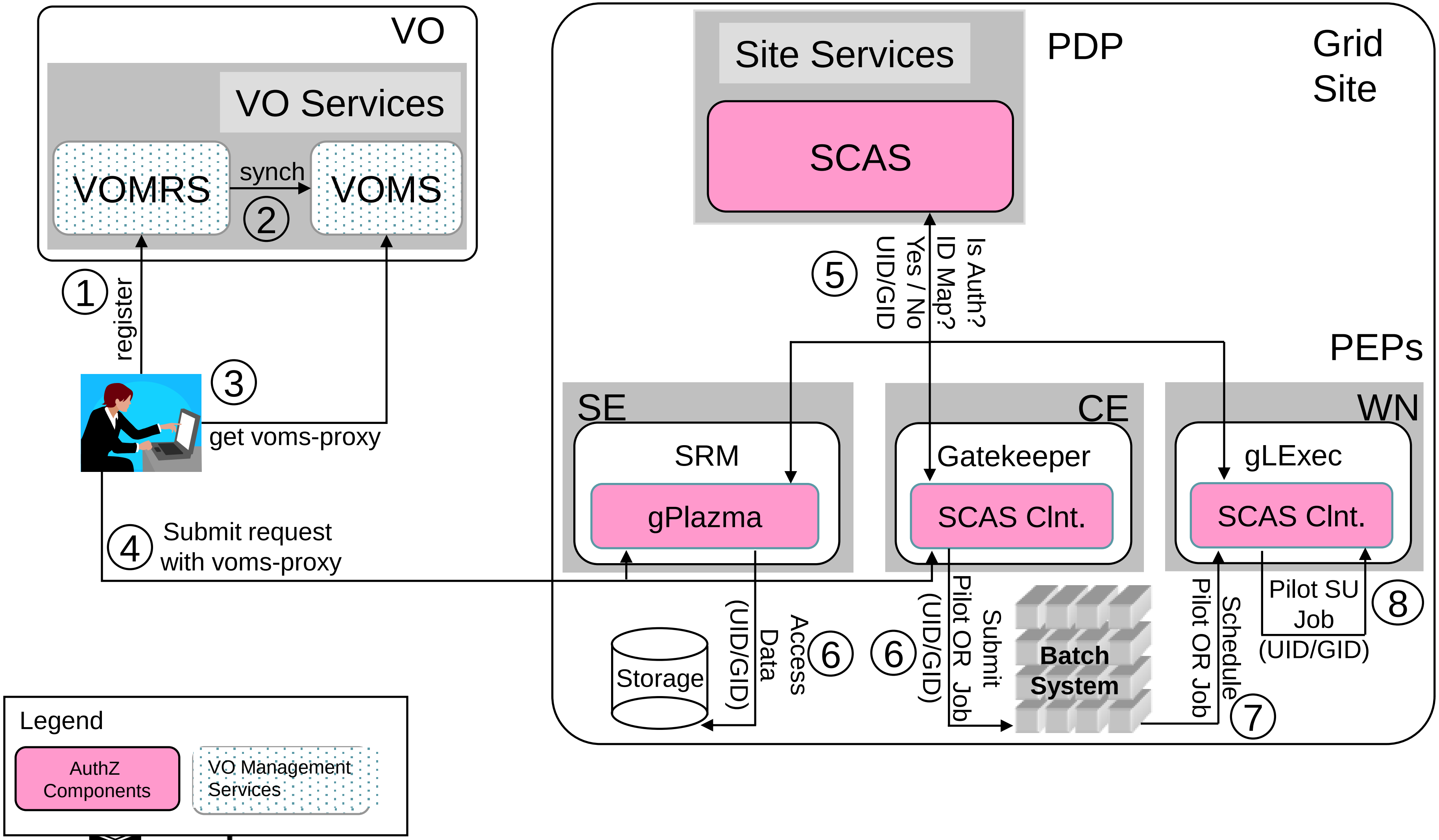
Harmonizing the internal semantics: XACML2-SAML2

Common protocol and common attributes

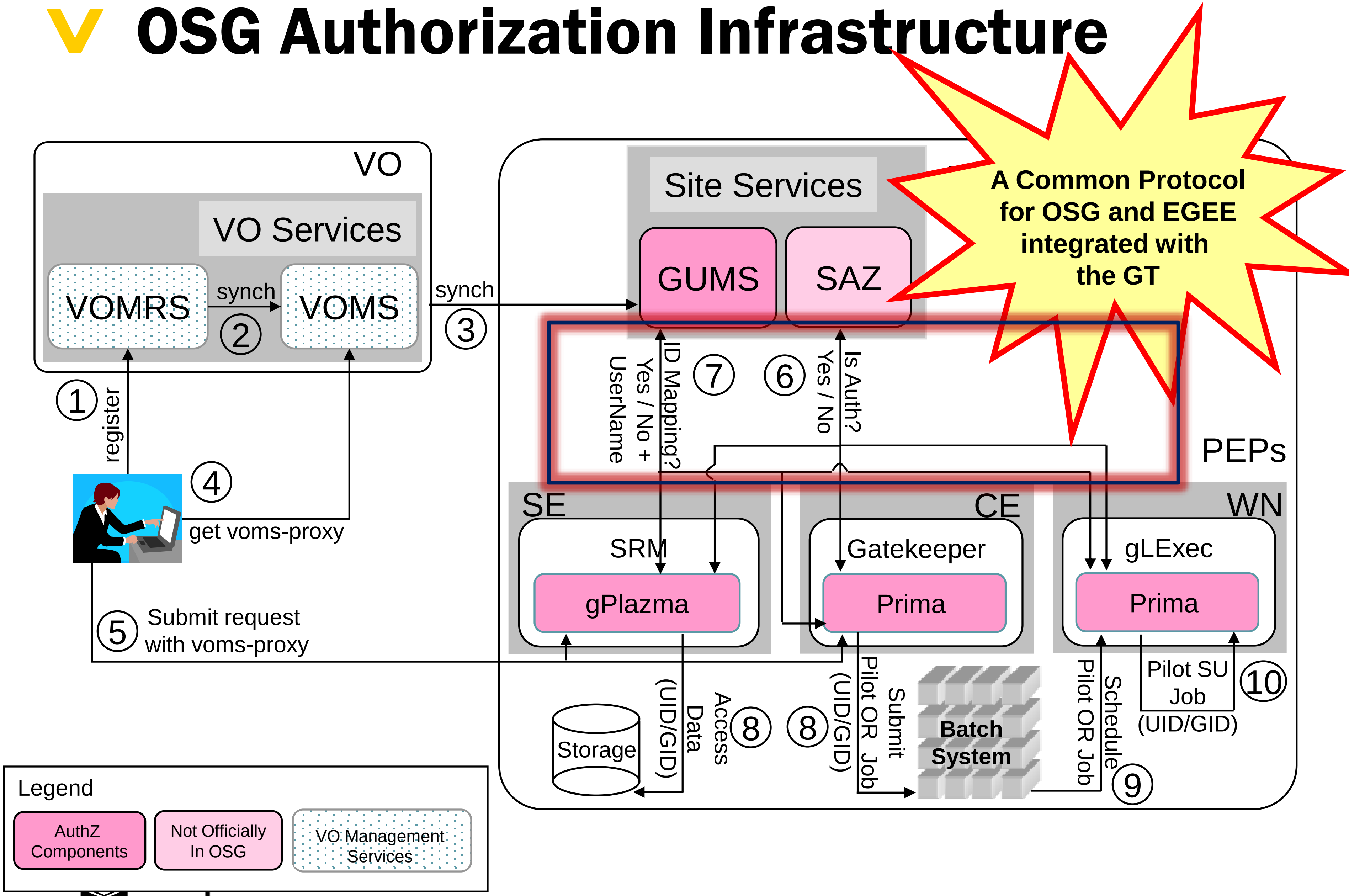
It works today!

INTEROPERABILITY NOW: INSIDE THE SITE

EGEE scenario ‘today’



OSG Authorization Infrastructure



A Common Protocol for OSG and EGEE integrated with the GT

What did we start with?

- GUMS and PRIMA using proprietary SAML1 based protocol
- SAZ (authorization service)
- GT3, GT4 ‘authz callout’ for PRIMA by Markus Lorch
- LCAS/LCMAPS
- gLExec
- gPlazma

Not only used GUMS and the ‘EGEE’ a different logical model of dealing with attributes and authorization (local VOMS dumps vs. VOMS attribute push)

Also, the services build in one ecosystem (e.g. ‘EGEE’) could not be used in the other (‘OSG’) → hacks and double work

Aims of the authz-interop project

- Provide interoperability within the authorization infrastructures of OSG, EGEE, Globus and Condor
- See www.authz-interop.org

Through

- Common communication protocol
- Common attribute and obligation definition
- Common semantics **and**
  actual interoperation of production system

So that services can use either framework
and be used in both infrastructures

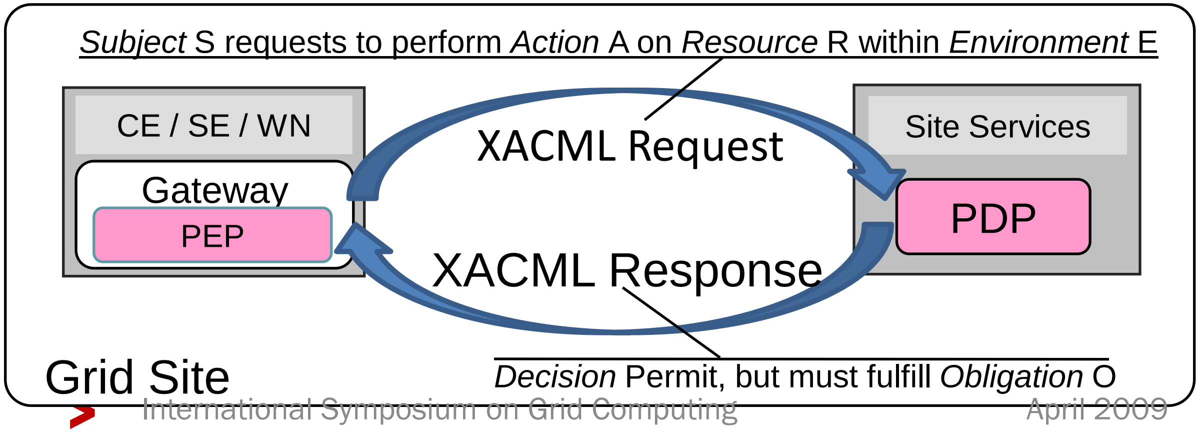
Two Elements for interop

- Common *communications* profile
  - Agreed on use of SAML2-XACML2
  - http://www.switch.ch/grid/support/documents/xacmlsaml.pdf

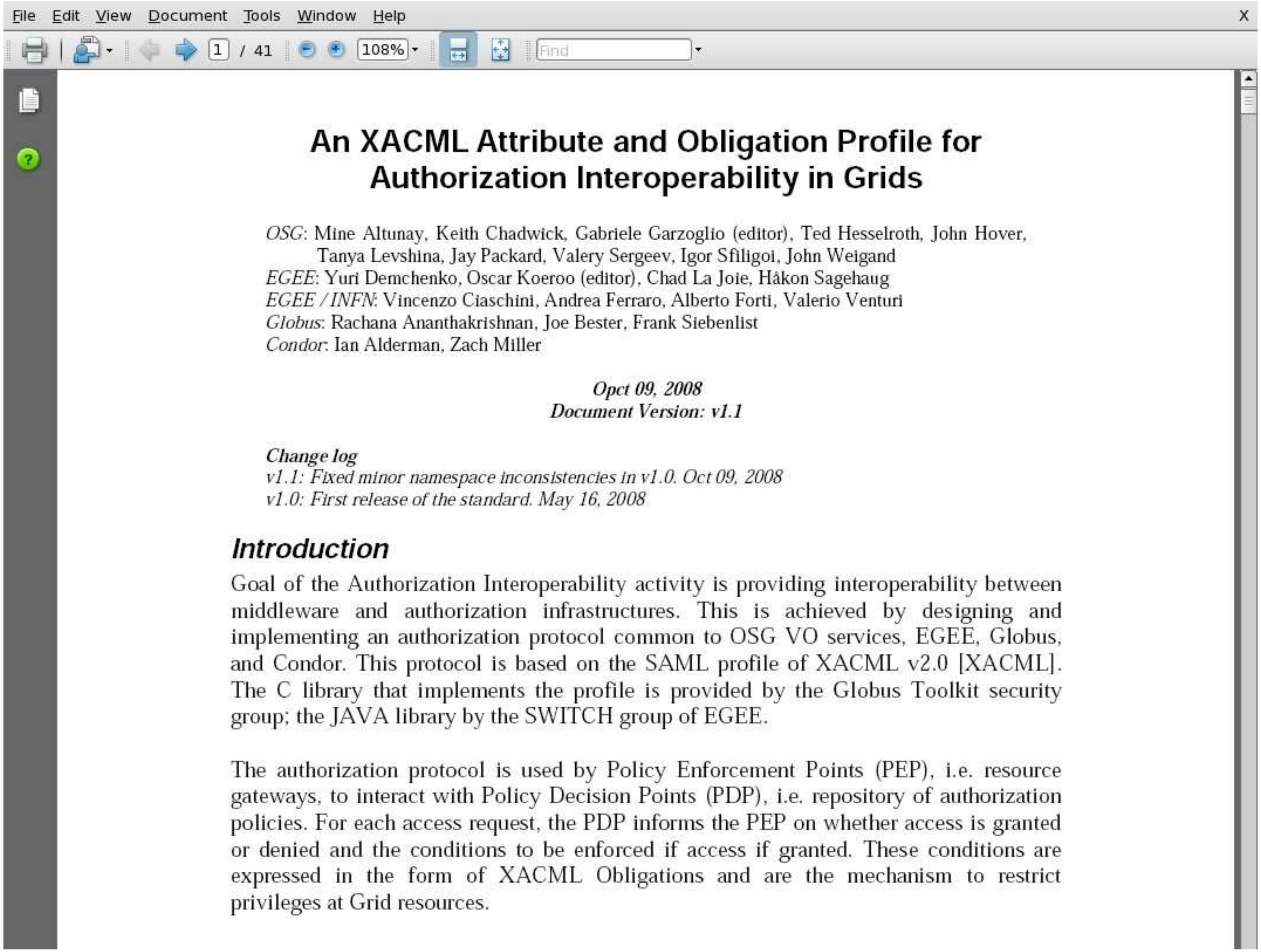
- Common *attributes and obligations* profile
  - List and semantics of attributes sent and obligations received between a ‘PEP’ and ‘PDP’
  - Now at version 1.1
  - http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=2952
  - http://edms.cern.ch/document/929867

Profile in a nutshell

- Existing standards:
  - XACML defines the XML-structures that are exchanged with the PDP to communicate the security context and the rendered authorization decision.
  - SAML defines the on-the-wire messages that envelope XACML's PDP conversation.
- The Authorization Interoperability profile augments those standards:
  - standardize names, values and semantics for common-obligations and core-attributes such that our applications, PDP-implementations and policy do interoperate.


Subject S requests to perform *Action* A on *Resource* R within *Environment* E
CE / SE / WN
Gateway
PEP
XACML Request
XACML Response
Site Services
PDP
Grid Site
Decision Permit, but must fulfill *Obligation* O

An XACML AuthZ Interop Profile



An XACML Attribute and Obligation Profile for Authorization Interoperability in Grids

OSG: Mine Altunay, Keith Chadwick, Gabriele Garzoglio (editor), Ted Hesselroth, John Hover, Tanya Levshina, Jay Packard, Valery Sergeev, Igor Sfiligoi, John Weigand
EGEE: Yuri Demchenko, Oscar Koeroo (editor), Chad La Joie, Håkon Sagehaug
EGEE / INFN: Vincenzo Ciaschini, Andrea Ferraro, Alberto Forti, Valerio Venturi
Globus: Rachana Ananthakrishnan, Joe Bester, Frank Siebenlist
Condor: Ian Alderman, Zach Miller

Opct 09, 2008
Document Version: v1.1

Change log
v1.1: Fixed minor namespace inconsistencies in v1.0. Oct 09, 2008
v1.0: First release of the standard. May 16, 2008

Introduction

Goal of the Authorization Interoperability activity is providing interoperability between middleware and authorization infrastructures. This is achieved by designing and implementing an authorization protocol common to OSG VO services, EGEE, Globus, and Condor. This protocol is based on the SAML profile of XACML v2.0 [XACML]. The C library that implements the profile is provided by the Globus Toolkit security group; the JAVA library by the SWITCH group of EGEE.

The authorization protocol is used by Policy Enforcement Points (PEP), i.e. resource gateways, to interact with Policy Decision Points (PDP), i.e. repository of authorization policies. For each access request, the PDP informs the PEP on whether access is granted or denied and the conditions to be enforced if access if granted. These conditions are expressed in the form of XACML Obligations and are the mechanism to restrict privileges at Grid resources.

- Authorization Interoperability Profile based on the SAML v2 profile of XACML v2
- Result of a 1yr collaboration between OSG, EGEE, Globus, and Condor
- Releases:
  v1.1 → 10/09/08
  v1.0 → 05/16/08

Structure of the AuthZ Interop Profile

- Namespace prefix: *http://authz-interop.org/xacml*

Request Attribute Identifiers

- Subject: *<ns-prefix>/subject/<subject-attr-name>*
- Action: *<ns-prefix>/action/<action-attr-name>*
- Resource: *<ns-prefix>/resource/<resource-attr-name>*
- Environment: *<ns-prefix>/environment/<env-type>*

Obligation Attribute Identifiers

- ObligationId: *<ns-prefix>/obligation/<obligation-name>*
- AttributeId: *<ns-prefix>/attributes/<obligation-attr-name>*

Most Common Request attributes

- Subject (see profile doc for full list)
  - Subject-X509-id
    - String: OpenSSL DN notation
  - Subject-VO
    - String: “CMS”
  - VOMS-FQAN
    - String: “/CMS/VO-Admin”
- Resource (see doc for full list)
  - Resource-id (enum type)
    - CE / SE / WN
  - Resource X509 Service Certificate Subject
    - resource-x509-id
  - Host DNS Name
    - Dns-host-name
- Action
  - Action-id (enum type)
    - Queue / Execute-Now / Access (file)
  - Res. Spec. Lang.
    - RSL string
- Environment
  - PEP-PDP capability negot.
    - PEP sends to PDP supported Obligations
    - Enables upgrading of the PEPs and PDPs independently
  - Pilot Job context (pull-WMS)
    - Pilot job invoker identity
    - Policy statement example: “User access to the WN execution environment can be granted only if the pilot job belongs to the same VO as the user VO”

see document for all attributes and obligations

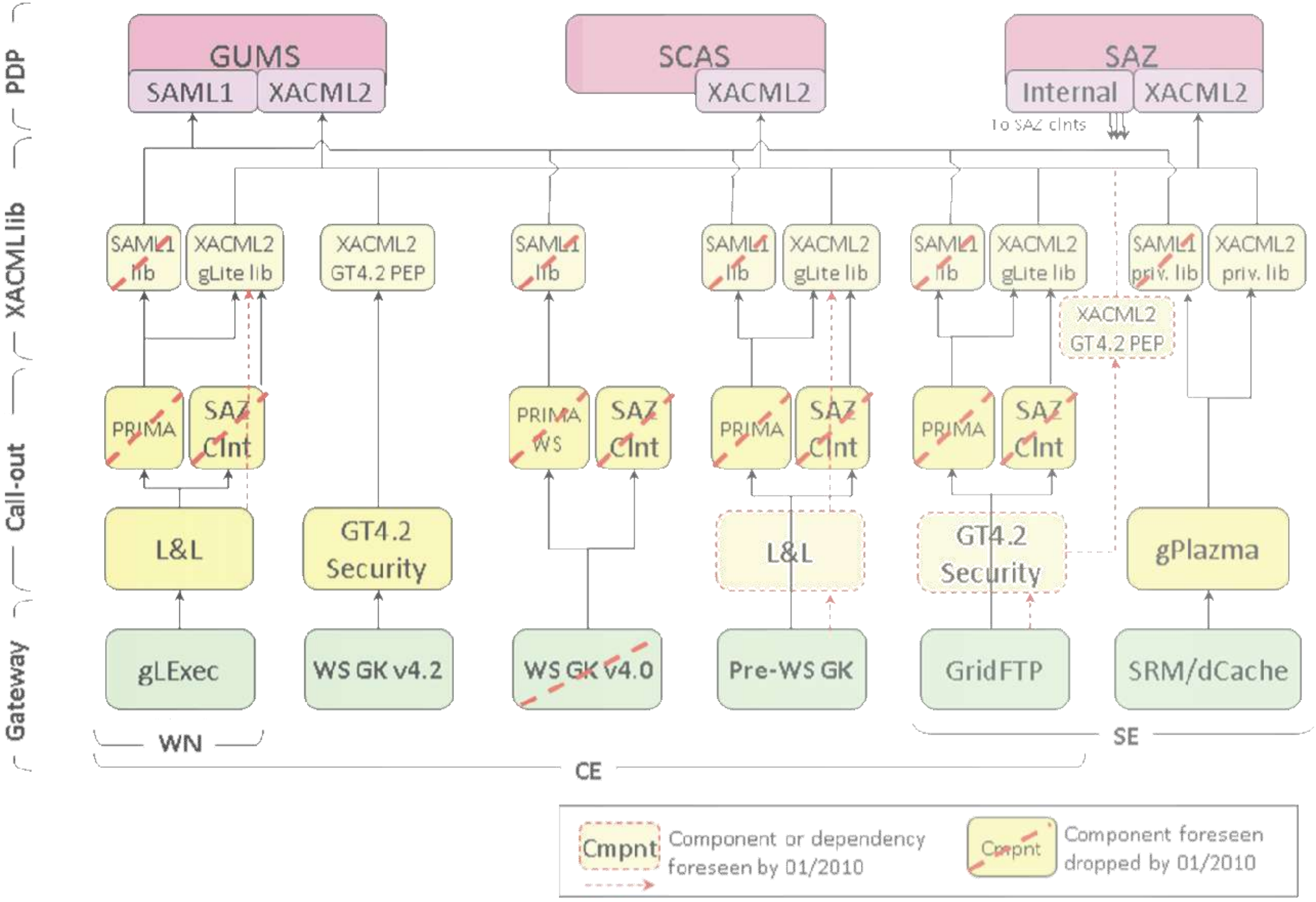
Most Common Obligation Attributes

- UIDGID
  - UID (integer): Unix User ID local to the PEP
  - GID (integer): Unix Group ID local to the PEP
- Secondary GIDs
  - GID (integer): Unix Group ID local to the PEP (Multi recurrence)
- Username
  - Username (string): Unix username or account name local to the PEP.

- Path restriction
  - RootPath (string): a sub-tree of the FS at the PEP
  - HomePath (string): path to user home area (relative to RootPath)
- Storage Priority
  - Priority (integer): priority to access storage resources.
- Access permissions
  - Access-Permissions (string): “read-only”, “read-write”

see document for all attributes and obligations

Full interoperability



What has been achieved now

- All profiles written and implemented
- Common libraries available in Java and C implementing the communications protocol
- Common handlers for
  Joint Interoperable Attribute and Obligations
- Integrated in all relevant middleware in EGEE and OSG:
  - Clients: lcg-CE (via LCMAPS scasclient), CREAM and gLExec (ditto), GT pre-WS gram (both prima and LCMAPS), GT GridFTP, GT4.2 WS-GRAM, dCache/SRM
  - Servers: GUMS, SCAS
- Other (lower-prio) components in progress
  - SAZ, RFT, GT4.x native-AuthZ, Condor (& -G), BeStMan

OSG Integration Test Results

Component	Test	PDP Component		
		Old GUMS	New GUMS	SCAS
WS-Gatekeeper **(Out of Scope)**	Test call-out component	NO	YES	YES
	Run job w/o Delegation or File Transfer	NO	YES	out of scope
	Run job with Delegation and File Transfer	NO	YES	out of scope
SCAS / PRIMA cmd line tool **(OOS)**	AuthZ call via Legacy protocol call-out	YES	YES	NO
	AuthZ call via XACML protocol call-out	NO	YES	YES
Pre-WS Gatekeeper **(VTB-TESTED)**	Run job. AuthZ via Legacy protocol	YES	YES	NO
	Run job. AuthZ via XACML protocol	NO	YES	YES
GridFTP **(VTB-TESTED)**	Transfer file. AuthZ via Legacy protocol	YES	YES	NO
	Transfer file. AuthZ via XACML protocol	NO	YES	YES
gLExec **(REL. Jan 20)**	Run pilot job. AuthZ via Legacy protocol	YES	YES	NO
	Run pilot job. AuthZ via XACML protocol	NO	YES	YES
SRM/dCache gPlazma **(REL. Jan 20)**	Transfer file. AuthZ via Legacy protocol	YES	YES	NO
	Transfer file. AuthZ via XACML protocol	NO	YES	YES

Latest news

- dCache v1.9.2-4 has been released
  Pre-release tests have been conducted successfully against GUMS and SCAS.
  Will be the recommended release in a few months

- Development of native authz XACML call-out module for GridFTP: tests with the Globus Toolkit call-out module for authorization speaking the interop protocol started

Participants

- EGEE
- VO Services (‘Privilege’) Project
- Globus
- Condor
- VDT (OSG)

Joint development and definition effort 2007 until early 2009

In production phase as of mid 2008

Institutes:
ANL, BCCS, BNL, FNAL, INFN, Nikhef, Switch, UvA, UWMadison

V

Access Control semantics

Breakdown of the container model

Legacy forever: mapping grid storage onto Unix semantics

The DPM model

DATA STORAGE

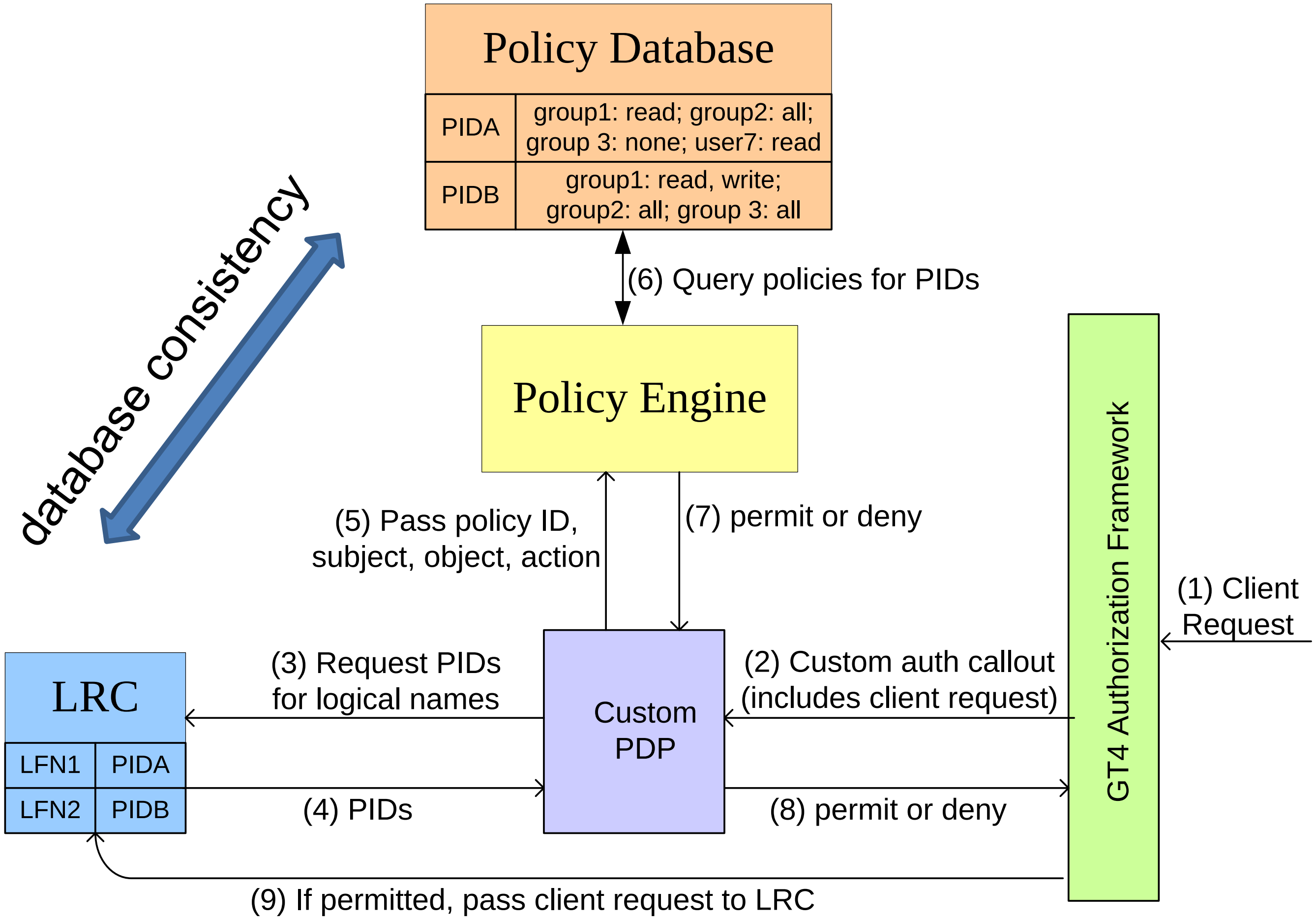
Storage: Access Control Lists

- Catalogue level
  - protects access to meta-data
  - is only advisory for actual file access
    unless the storage system only accepts connections from a trusted agent that does itself do a catalogue lookup
- SE level
  - either natively (i.e. supported by both the SRM and transfer services)
- SRM/transfer level
  - SRM and GridFTp server need to lookup in local ACL store for each transfer
  - need “all files owned by SRM” unless underlying FS supports ACLs
- *OS level?*
  - *native POSIX-ACL support in OS would be needed*
  - *Mapping would still be requires (as for job execution)*

Grid ACL considerations

- Semantics
  - Posix semantics require that you traverse *up* the tree to find all constraints
  - behaviour both costly and possibly undefined in a distributed context
  - VMS and NTFS container semantics are self-contained
  - taken as a basis for the ACL semantics in many grid services
- ACL syntax & local semantics typically Posix-style

‘Container abstraction’ breakdown



Policy Database

PIDA	group1: read; group2: all; group 3: none; user7: read
PIDB	group1: read, write; group2: all; group 3: all

(6) Query policies for PIDs

Policy Engine

database consistency

(5) Pass policy ID, subject, object, action

(7) permit or deny

GT4 Authorization Framework

(1) Client Request

LRC

LFN1	PIDA
LFN2	PIDB

(3) Request PIDs for logical names

Custom PDP

(2) Custom auth callout (includes client request)

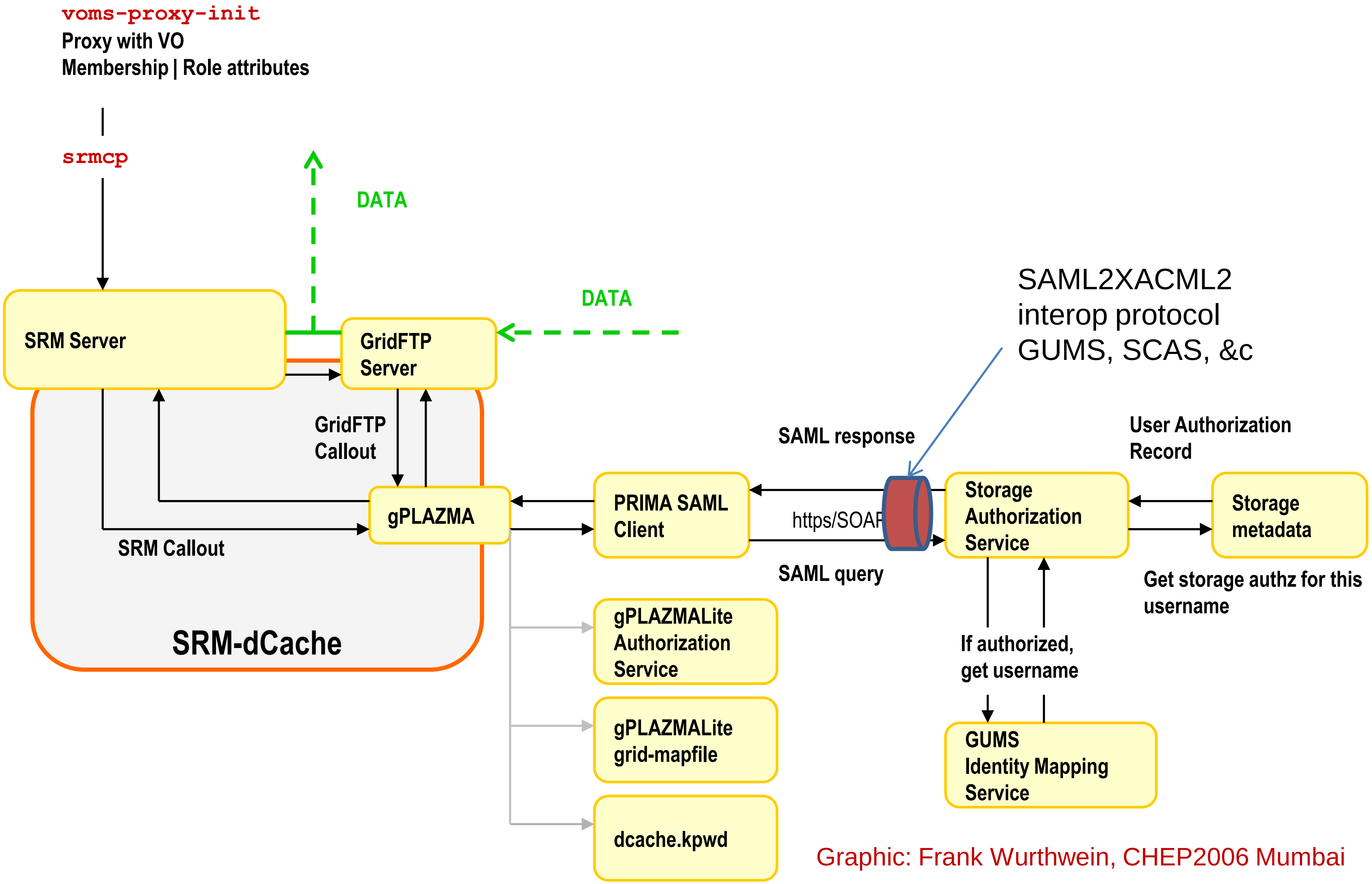
(4) PIDs

(8) permit or deny

(9) If permitted, pass client request to LRC

graphic: Ann Chervenak, ISI/USC, from presentation to the Design Team, Argonne, 2005

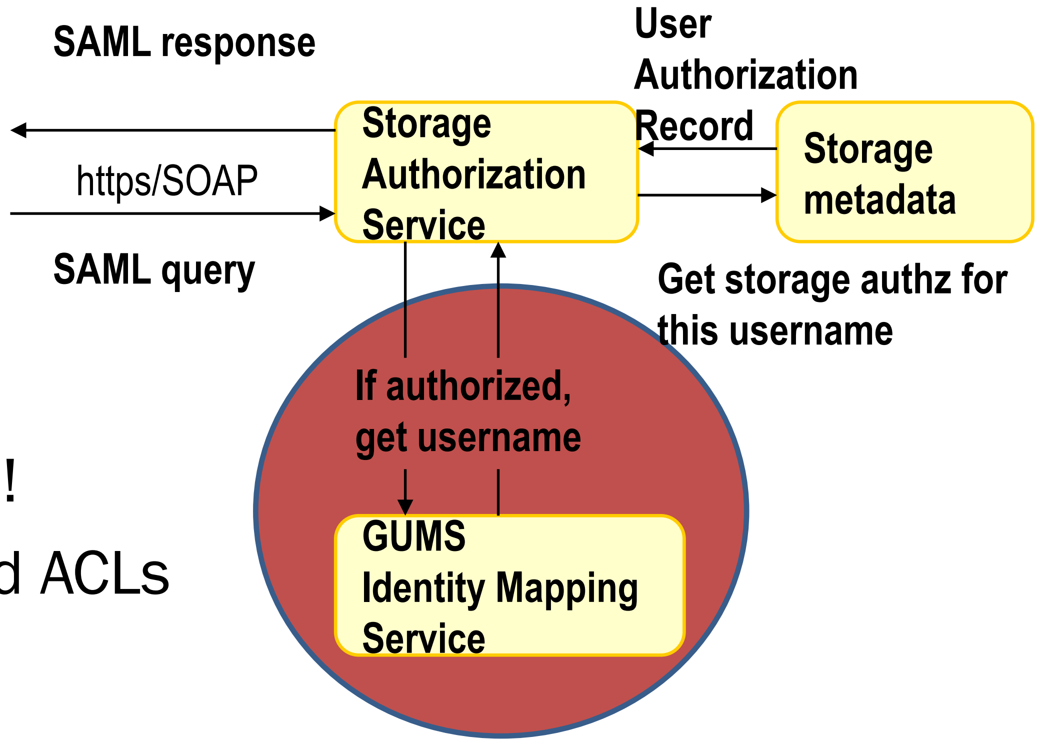
Embedded access control: dCache



Graphic: Frank Wurthwein, CHEP2006 Mumbai

Legacy persists, though

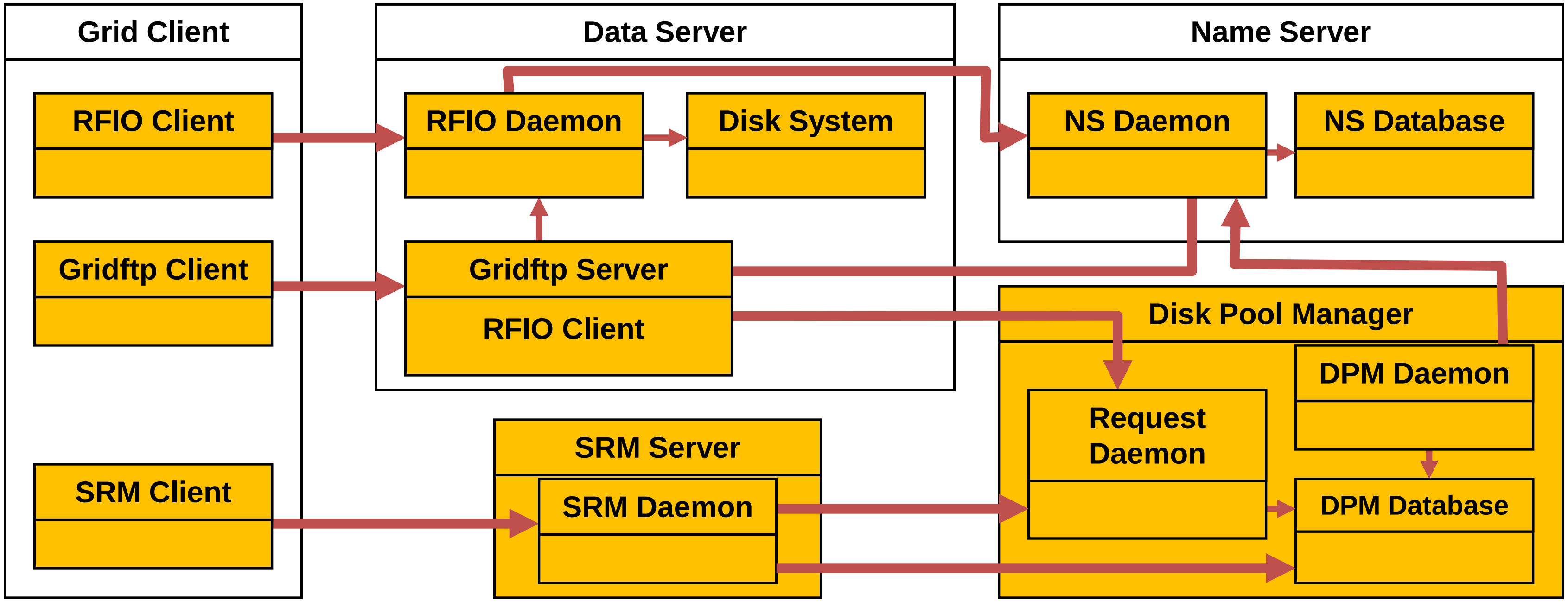
- dCache/gPlazma maps back to
  - Unix username
  - ‘root path’
- Files stored with *Unix* uid and gid
  - Can have local access!
  - But doing VOMS-based ACLs over simple Unix ACLs results in a combinatorial group explosion


SAML response
https/SOAP
SAML query
Storage Authorization Service
User Authorization Record
Storage metadata
Get storage authz for this username
If authorized, get username
GUMS Identity Mapping Service

Grid storage access control

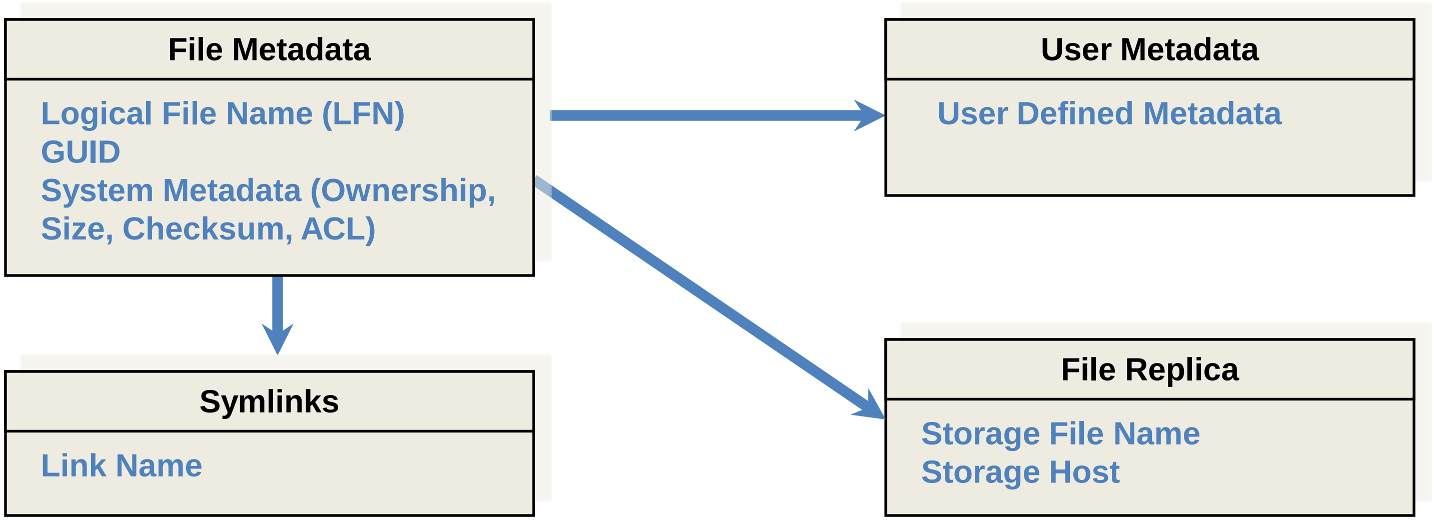
- Use ‘grid’ identity and attributes to define ACLs
- With ‘POSIX’ semantics
  - So traversal based, not object based
  - Needs ‘good’ database schema to store ACLs&metadata
- Example: DPM “Disk Pool Manager”
  - See https://twiki.cern.ch/twiki/bin/view/EGEE/GliteDPM

DPM Architecture



Slides and graphics: ‘ACLs in Light Weight Disk Pool Manager’ MWSG 2006, Jean Philippe Baud, CERN

DPM File Catalog Schema



File Metadata

Logical File Name (LFN)
GUID
System Metadata (Ownership, Size, Checksum, ACL)

User Metadata

User Defined Metadata

Symlinks

Link Name

File Replica

Storage File Name
Storage Host

LFN acts as main key in Database. Has:

- Unique Identifier (GUID)
- Information on Physical Replicas
- Symbolic Links to it
- A small amount (one field) of user attached metadata

Slides and graphics: ‘ACLs in Light Weight Disk Pool Manager’ MWSG 2006, Jean Philippe Baud, CERN

Virtual Ids and VOMS integration

- DNs are mapped to virtual UIDs: the virtual uid is created on the fly the first time the system receives a request for this DN (no pool account)
- VOMS roles are mapped to virtual GIDs
- A given user may have one DN and several roles, so a given user may be mapped to one UID and several GIDs
- Currently only the primary role is used in LFC/DPM
- Support for normal proxies and VOMS proxies
- Administrative tools available to update the DB mapping table:
  - To create VO groups in advance
  - To keep same uid when DN changes
  - To get same uid for a DN and a Kerberos principal

Slides and graphics: ‘ACLs in Light Weight Disk Pool Manager’ MWSG 2006, Jean Philippe Baud, CERN

DPNS mapping tables

```
CREATE TABLE Cns_groupinfo (
    gid NUMBER(10),
    groupname VARCHAR2(255));

CREATE TABLE Cns_userinfo (
    userid NUMBER(10),
    username VARCHAR2(255));
```

- *included in GridFTP through ‘dli’ plugin mechanism, and in SRM through call-outs to dpns*

Slides and graphics: ‘ACLs in Light Weight Disk Pool Manager’ MWSG 2006, Jean Philippe Baud, CERN

Access Control Lists

- LFC and DPM support Posix ACLs based on Virtual Ids
  - Access Control Lists on files and directories
  - Default Access Control Lists on directories: they are inherited by the sub-directories and files under the directory
- Example
  - dpns-mkdir /dpm/cern.ch/home/dteam/jpb
  - dpns-setacl -m d:u::7,d:g::7,d:o:5 /dpm/cern.ch/home/dteam/jpb
  - dpns-getacl /dpm/cern.ch/home/dteam/jpb

```
# file: /dpm/cern.ch/home/dteam/jpb
# owner: /C=CH/O=CERN/OU=GRID/CN=Jean-Philippe Baud 7183
# group: dteam
user::rwx
group::r-x              #effective:r-x
other::r-x
default:user::rwx
default:group::rwx
default:other::r-x
```

Slides and graphics: 'ACLs in Light Weight Disk Pool Manager' MWSG 2006, Jean Philippe Baud, CERN

Hydra distributed key store

SSSS

SPECIALISED MIDDLEWARE

Encrypted Data Storage

Medical community as the principal user

- large amount of images
- privacy concerns vs. processing needs
- ease of use (image production and application)

Strong security requirements

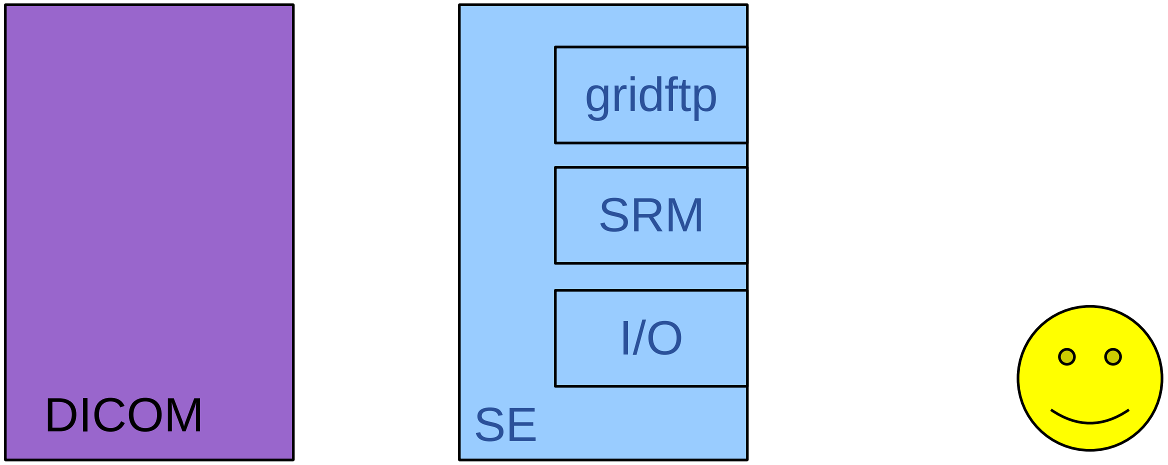
- anonymity (patient data is separate)
- fine grained access control (only selected individuals)
- privacy (even storage administrator cannot read)

Described components are under development

Slides based on Akos Frohner, EGEE and CERN

Building Blocks

- Hospitals:
  - DICOM = Digital Image and COmmunication in Medicine
- Grid: SE = SRM + gridftp + I/O
  - and a client (application processing an image)



Goal: data access at any location

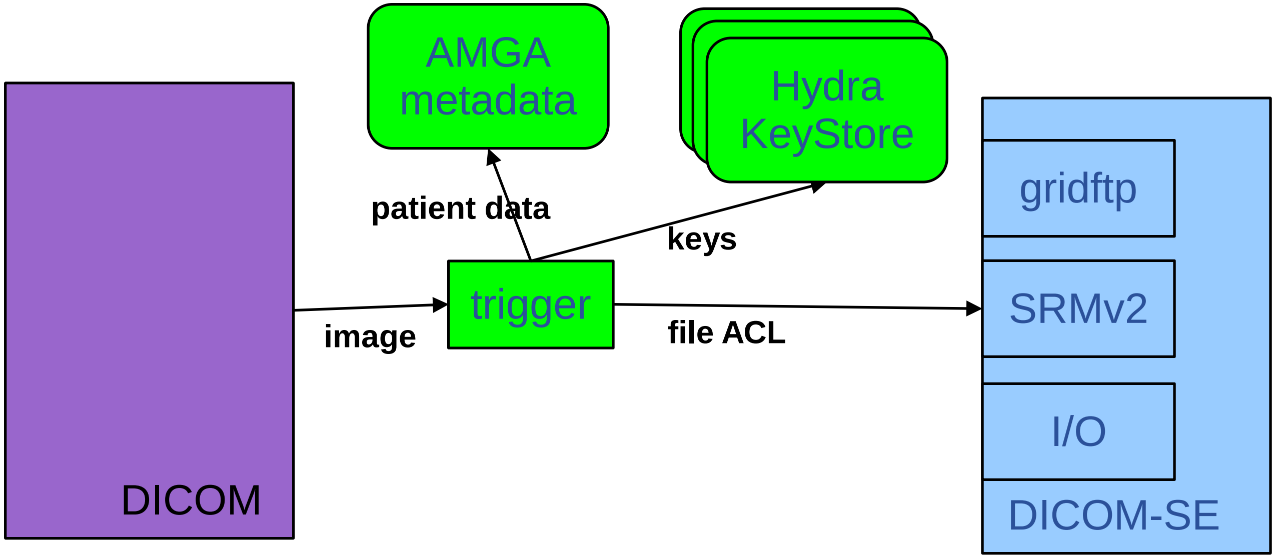
Slides based on Akos Frohner, EGEE and CERN

Exporting Images

"wrapping" DICOM :

- anonymity: patient data is separated and stored in AMGA
- access control: ACL information on individual files in SE (DPM)
- privacy: per-file keys
  - distributed among several Hydra key servers
  - fine grained access control

Image is retrieved from DICOM and processed to be "exported" to the grid.



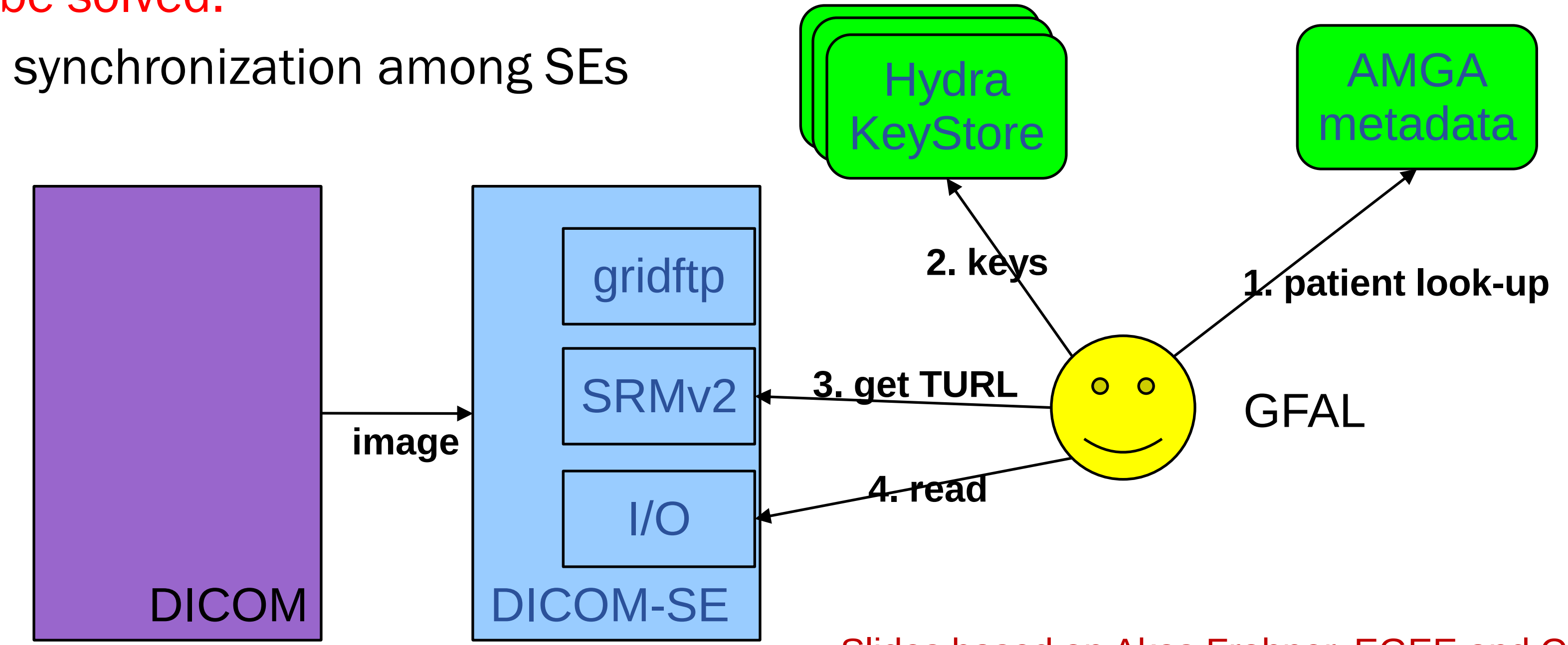
Slides based on Akos Frohner, EGEE and CERN

Accessing Images

- image ID is located by AMGA
- key is retrieved from the Hydra key servers (implicitly)
- file is accessed by SRM (access control in DPM)
- data is read and decrypted block-by-block
  in memory only (GFAL and hydra-cli)---> useful for all

Still to be solved:

- ACL synchronization among SEs



Slides based on Akos Frohner, EGEE and CERN

Hydra key store theory, and SSSS

- Keys are split for security and reliability reasons using Shamir's Secrect Sharing Scheme (org.glite.security.ssss)
  - standalone library and CLI
  - modified Hydra service and Hydra client library/CLI
  - the client contacts all services for key registration, retrieval and to change permissions
    - there is no synchronization or transaction coordinator service

```
$ glite-ssss-split-passwd -q 5 3 secret
137c9547aba101ef 6ee7adbbaacac1ef 1256bcc160eda592
  fdabc259cdfbacc9 3113be83f203d794
$ glite-ssss-join-passwd -q 137c9547aba101ef NULL \
  1256bcc160eda592 NULL 3113be83f203d794
secret
```

Slides based on Akos Frohner, EGEE and CERN

Integration into DPM

- lcg-cp -bD srmv2 srm://dpm.example.org:8446/srm/managerv2?
  - SFN=/dpm/example.org/home/biomed/mdm/<ID> file:picture.enc
- glite-eds-decrypt <ID> picture.enc picture
- glite-eds-get -i <ID> rfio:////dpm/example.org/home/biomed/mdm/<ID> picture
  - file is opened via gfal_open()
  - decryption key is fetched for <ID>
  - loop on gfal_read(), glite_eds_decrypt_block(), write()

- 'glite-eds-get' is a simple utility over the EDS library.

Slides based on Akos Frohner, EGEE and CERN

V

Some pending issues

gLite Authorization Framework v2

FROM HERE?

Current issues

- Different Services use different authorization mechanisms
  - Some services even use internally more than one authorization framework
- Site administrators do not have simple debugging tools to check and understand their authorization configuration
- Site administrators must configure the authorization for each service at their site separately
  - Consequence 1: At a site, there is no single point to ban users/groups of users for the entire site
  - Consequence 2: many site administrators don’t know how to ban users
  - Consequence 3: no central banning at the infrastructure level
- Limited means of advertising specific policies
  - Only GlueAccessControlBaseRules with VOMS FQANs or DNs
- Inhomogeneous and limited monitoring

New gLite: AuthZ Framework (v2)

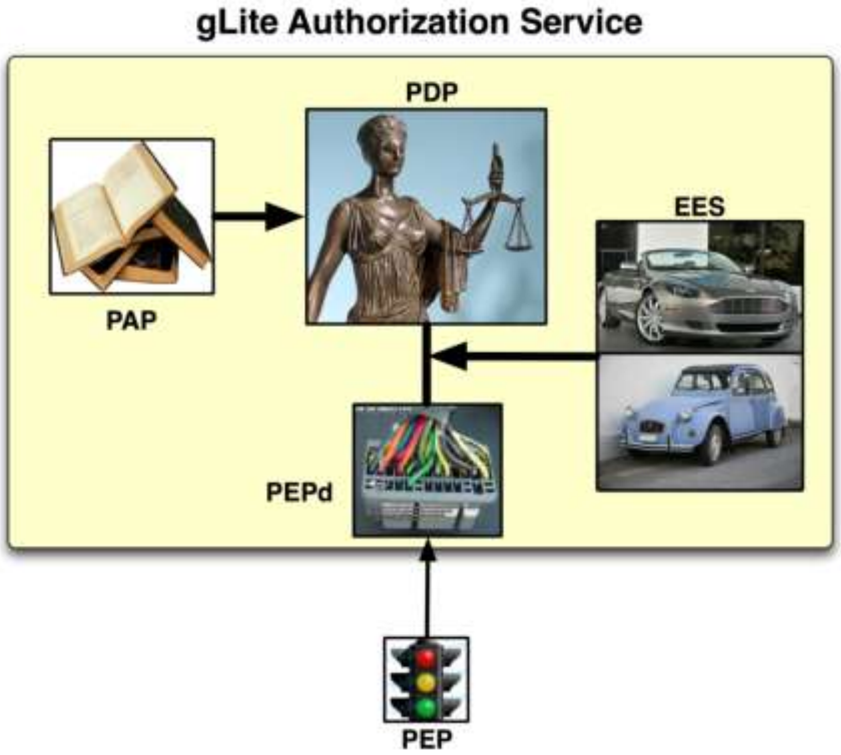
‘the only new gLite work item in EGEE-III’

- Addressing the above list of short-comings
- Service based (no embedded framework)
- Limited to *only* authorization,
  so you will still need credential validation outside of it!
- Resistance to failure and simple means for scaling service
  - Flexible deployment model, high availability options
- Client component is very lightweight
  - Small amount of code, few dependencies (especially on WN)
  - Portability: support on other OS and languages easy

https://edms.cern.ch/document/944192/1

gLite Authorization Framework (v2)

- Administration Point
  Formulating rules through CLI and/or file-based input
- Decision Point
  Evaluating a request from a client based on the rules
- Enforcement Point
  Thin client part and server part: all complexity in server part
- Runtime Execution Environment
  Under which env. must I run? (Unix UID, GID, …)



Graphic: Christoph Witzig, SWITCH and EGEE

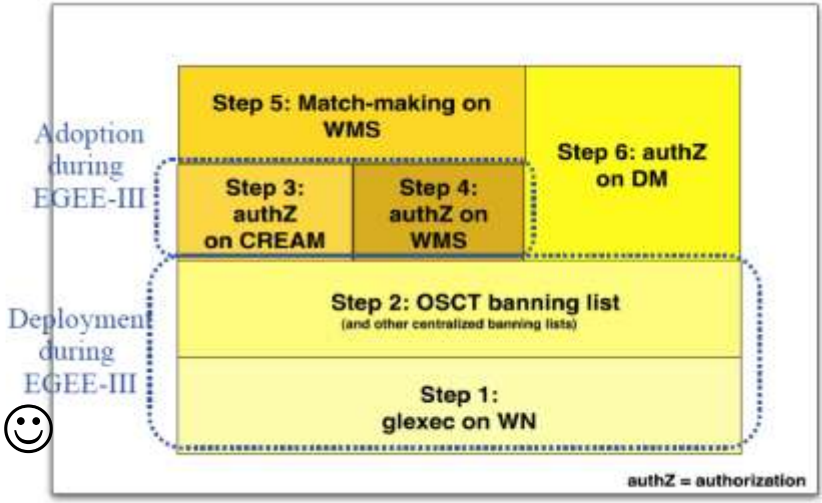
Capabilities

- Enables/eases various authorization tasks:
  - Banning of users (VO, WMS, site, or grid wide)
- Composition of policies – CERN policy + experiment policy + CE policy + OCST policy + NGI policy=> Effective policy
- Support for authorization based on more detailed information about the job, action, and execution environment
  - Support for authorization based on attributes other than FQAN
  - Support for multiple credential formats (not just X.509)
  - Support for multiple types of execution environments
  - Virtual machines, workspaces, …
- Nagios plug-ins provided for monitoring of service

https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework

Introduction of the service in gLite

- Focus is on computing services (again ☹)
  - Initial introduction through gLExec on the WN
  - As a new LCMAPS plug-in
    (used in conjunction with the others, esp. verify-proxy)
  - With OSCT ban list
- Has all the right buzzwords
  - PIP, PEP, PAP, PDP, and SAML
  - XACML policies and attributes
  - But *with* a simplified language ☺
- Testing well on its way!
- Interop and general adoption unclear, though… ☹



Deployment plans

- Expect service to enter certification in April
  - *Gradual* deployment in the six self-contained steps
- Initial focus on gLExec on WN and OSCT ban list
  - Configuration option for gLExec
  - Integration into CREAM and WMS for authorization
  - Integration into data management
  - Offers perspective to manage access to a site from one site-specific service
  - Longer term option for inclusion into match-making
- Feedback and volunteer sites
  for trying service out are welcome

Summary

- Security middleware is everywhere
  - An integral part of almost any grid service
  - Implemented in a myriad of ways
- Most of the core capabilities are there
  - VOMS based access, banning on VO or DN
  - But methodology varies,
    and the documentation is not well read or disseminated
- We're getting there with interop
  - authz-interop.org collaboration composed
    working mesh of interoperable security services
- But we're far from done …

QUESTIONS? DINNER?