On Multi-User Pilot Jobs

Recovering control in an obfuscated world

BiG Grid

the dutch e-science grid

Overview

- Grid participants
- Workload models
  - user workload management and multi-user pilot jobs
  - impact on traceability and incident handling
- Recovering control
  - policy actions and liability; containing mechanisms
- gLExec
  - sources and early deployments
  - controls and security audits
  - system interactions
- Deployment on clusters
  - site-wide access control,
- Towards integrated authorization

Traditional submission and brokering models

User-based job management

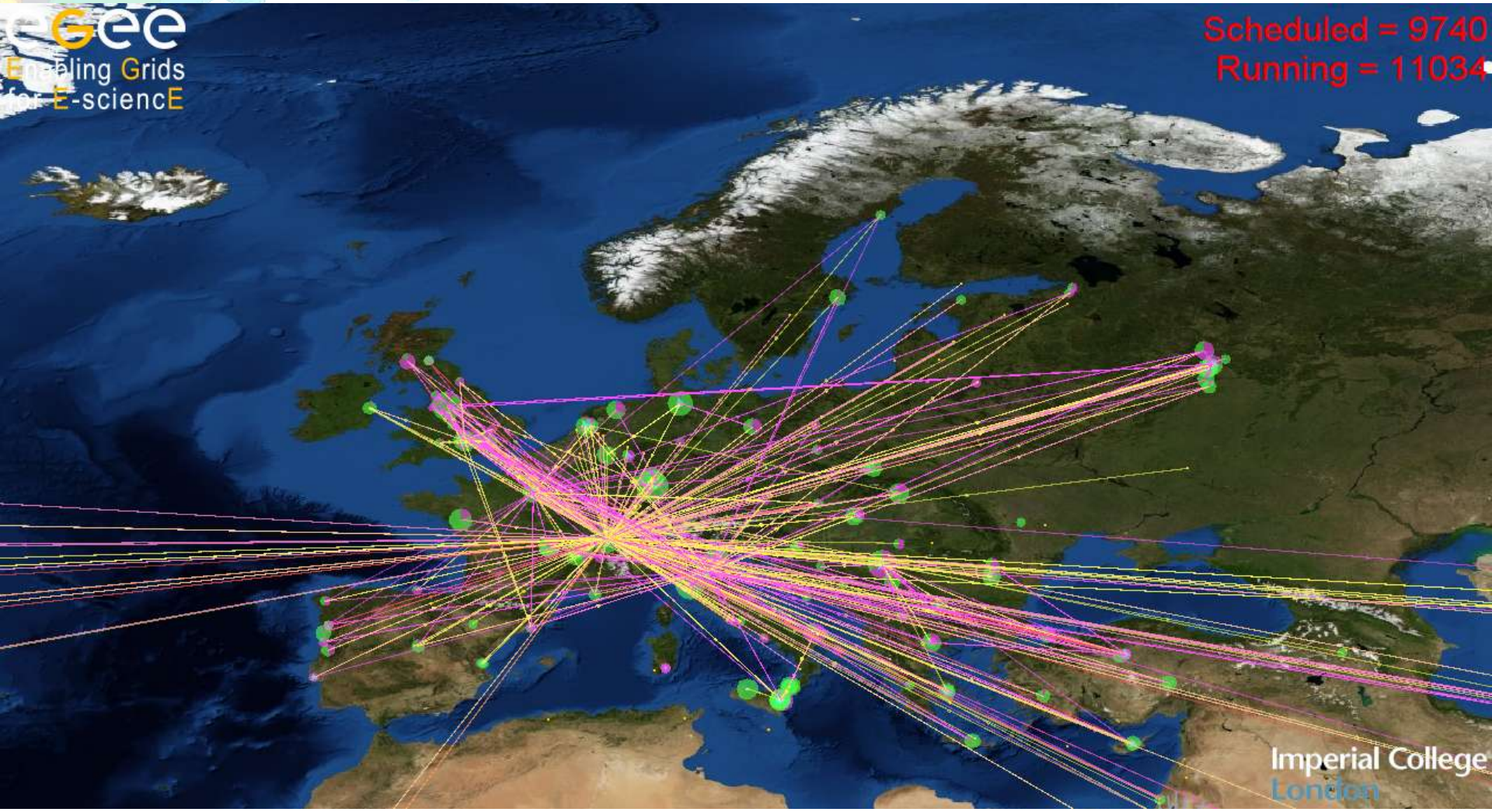
Multi-user pilot jobs

JOB SUBMISSION MODELS

BiG Grid

the dutch e-science grid

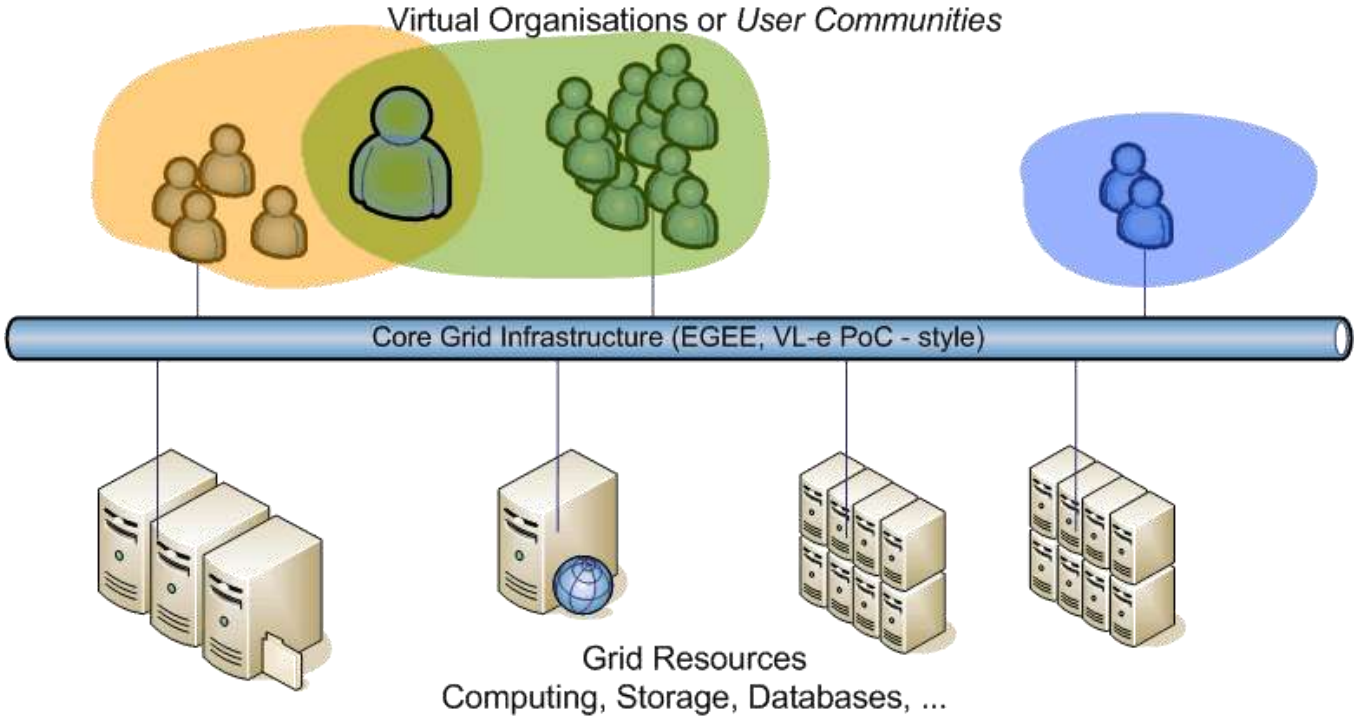
Multi-domain Grids



BiG Grid
the dutch e-science grid

e-Infrastructure model for Grid

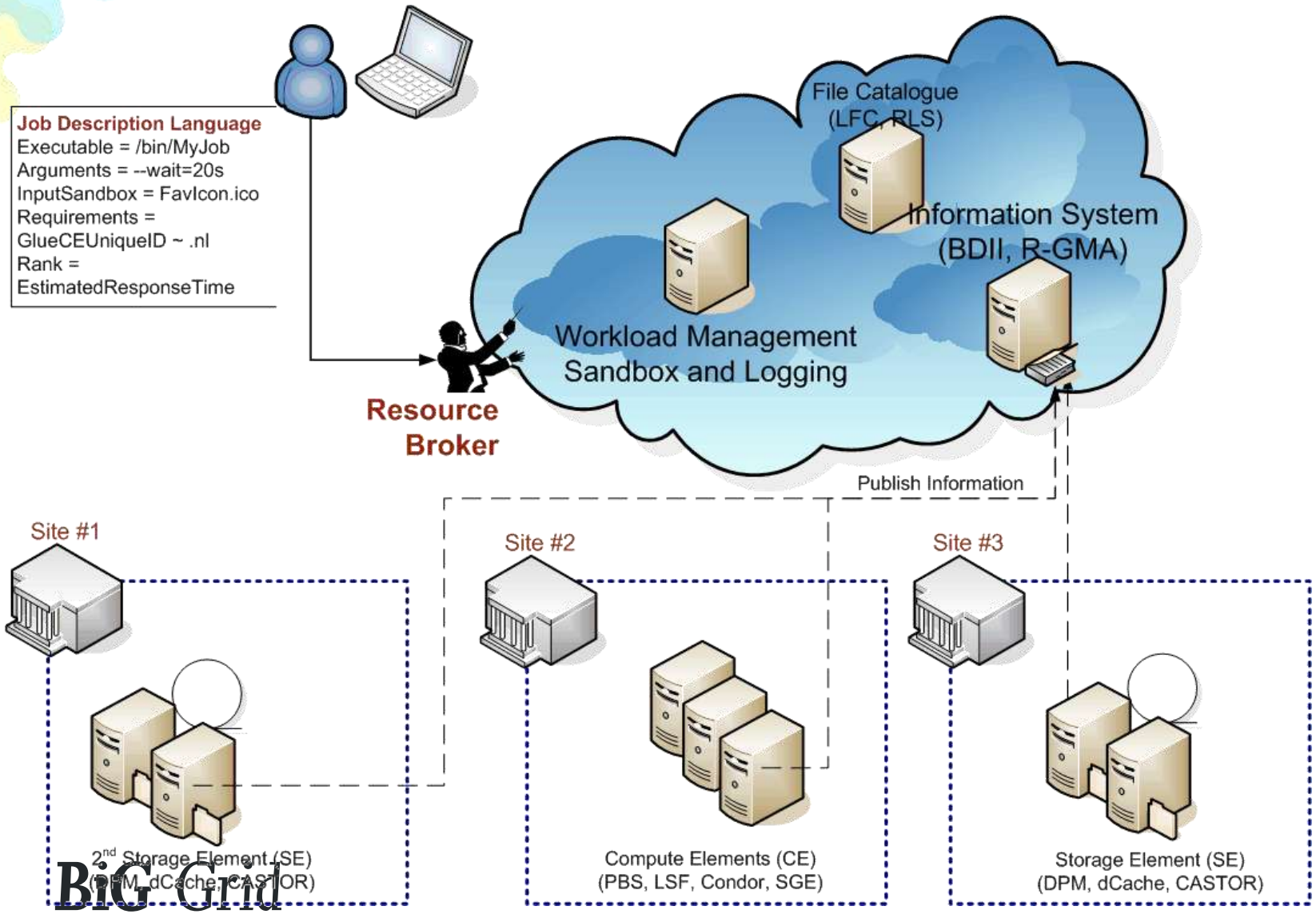
Users and user communities de-coupled from the sites



- Policy Coordination: User and VO AUPs, operations, trust
- Facilitating negotiation: VO meta-data, SLAs, op. environment

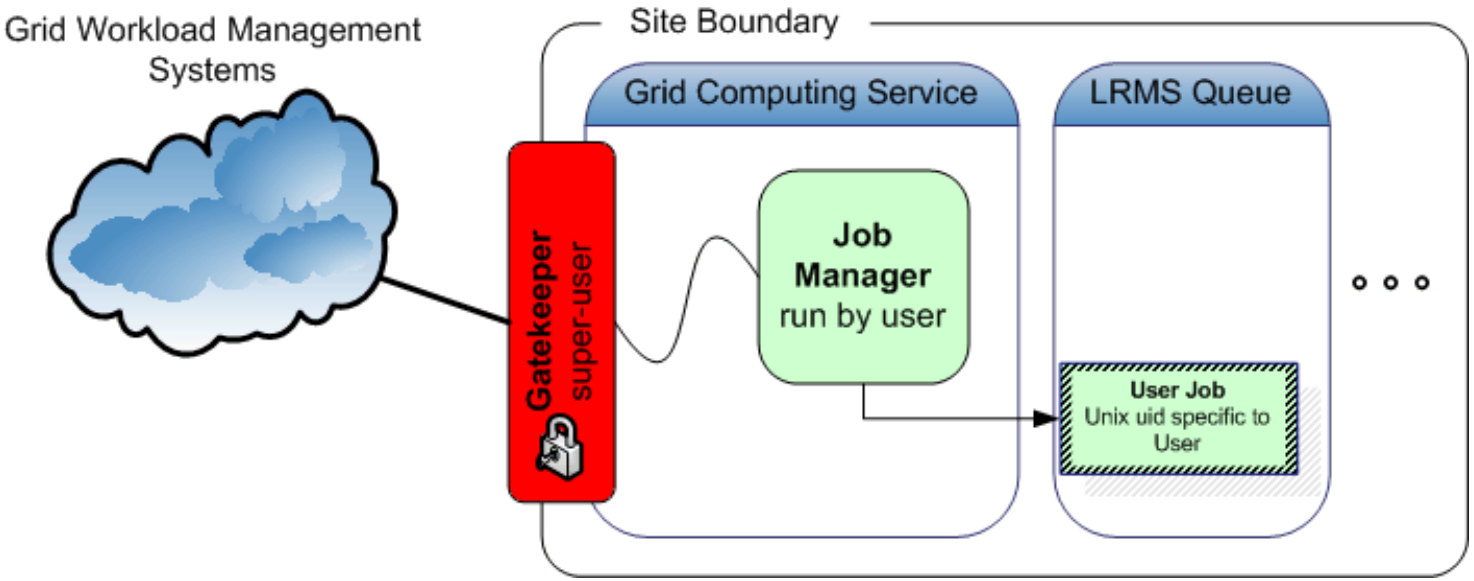
BiG Grid
the dutch e-science grid

Job Submission Scenario

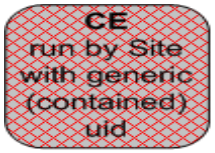


BiG Grid
the dutch e-science grid

Inside the site: traditional model



Colour coding:



Site controlled service, running with 'generic' uid

Real User Job
(with *uid* matching actual workload being run)

Site controlled, running as super-user

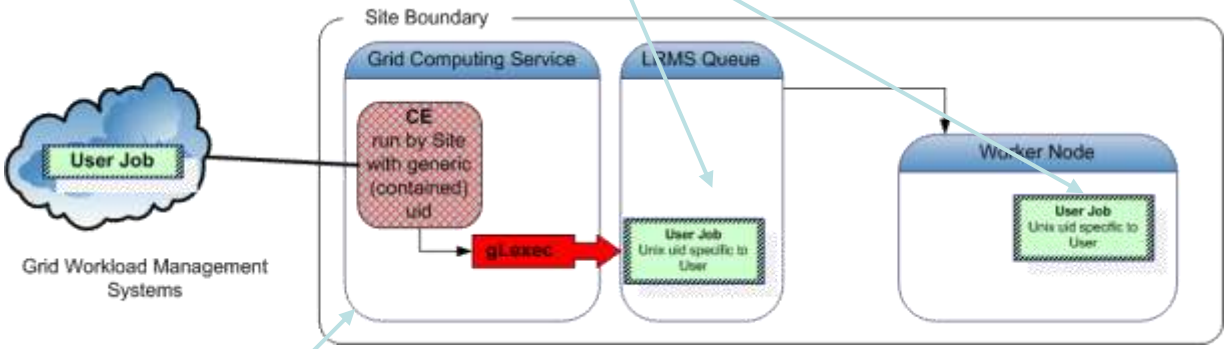
Site controlled super-user daemon

VO-run process
(generic VO *uid*, or generic VO pool)

BiG Grid
the dutch e-science grid

Classic job submission models

- In the submission models shown, submission of the user job to the batch system is done with the *original job owner's* mapped (uid, gid) identity



- grid-to-local identity mapping is done *only* on the front-end system (CE)
  - batch system accounting provides per-user records
  - inspection shows Unix process on worker nodes and in batch queue per-user

BiG Grid
the dutch e-science grid

Late binding: pilot jobs

Job submission gets more and more intricate …

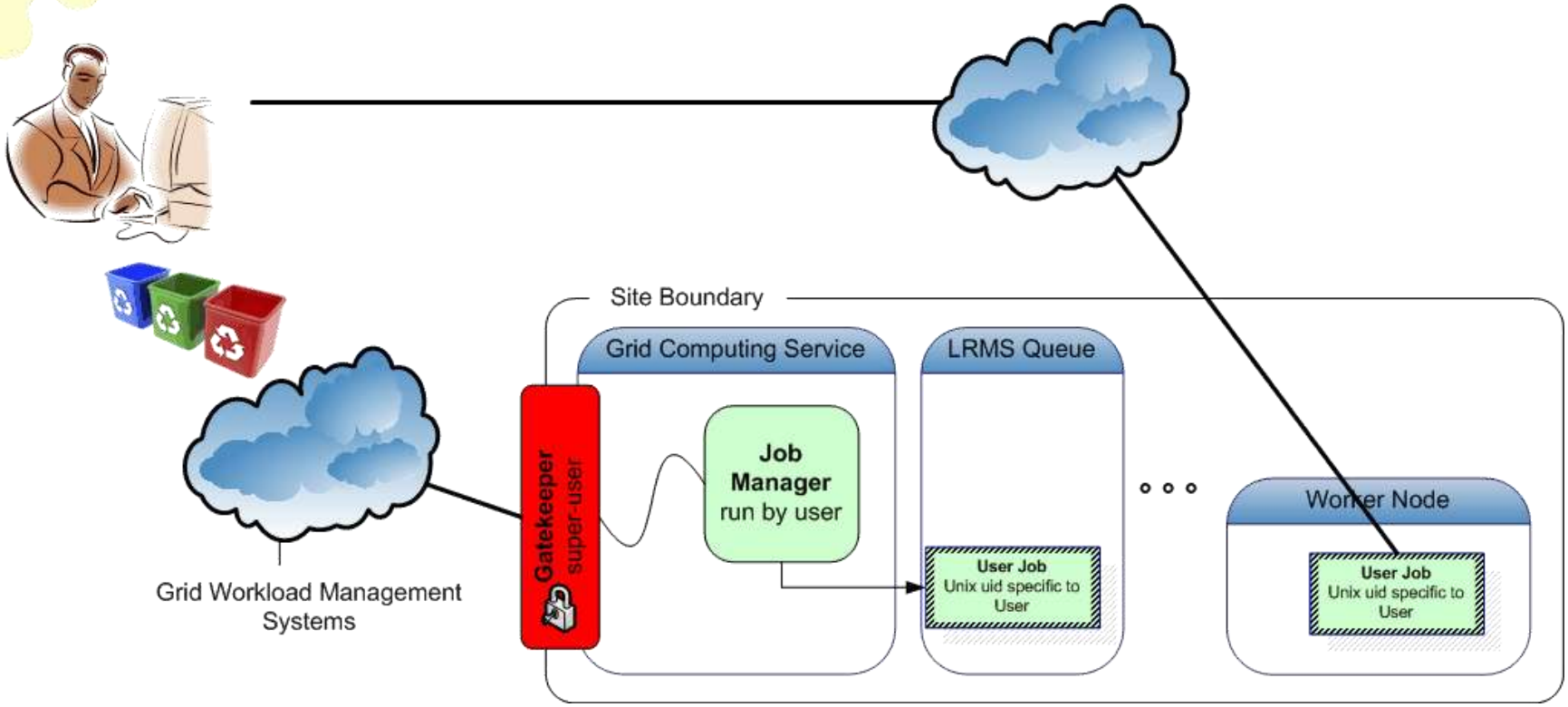
- Late binding of jobs to job slots via *pilot jobs*
  some users and communities develop and prefer to use proprietary, VO-specific, scheduling & job management
  - 'visible' job is a pilot: a small placeholder that downloads a real job
  - first establishing an overlay network,
  - subsequent scheduling and starting of jobs is faster

  - it is not committed to any particular task on launch
  - *perhaps not even bound to a particular user!*

- this scheduling is orthogonal to the site-provided systems

BiG Grid
the dutch e-science grid

Every user a pilot



BiG Grid
the dutch e-science grid

Pilot job pros and cons

Some Pros:

- Worker node validation and matching to task properties
- Intra-VO priorities can be reshuffled on the fly without involving site administrators
- Avoid jobs sitting in queues when they could run elsewhere

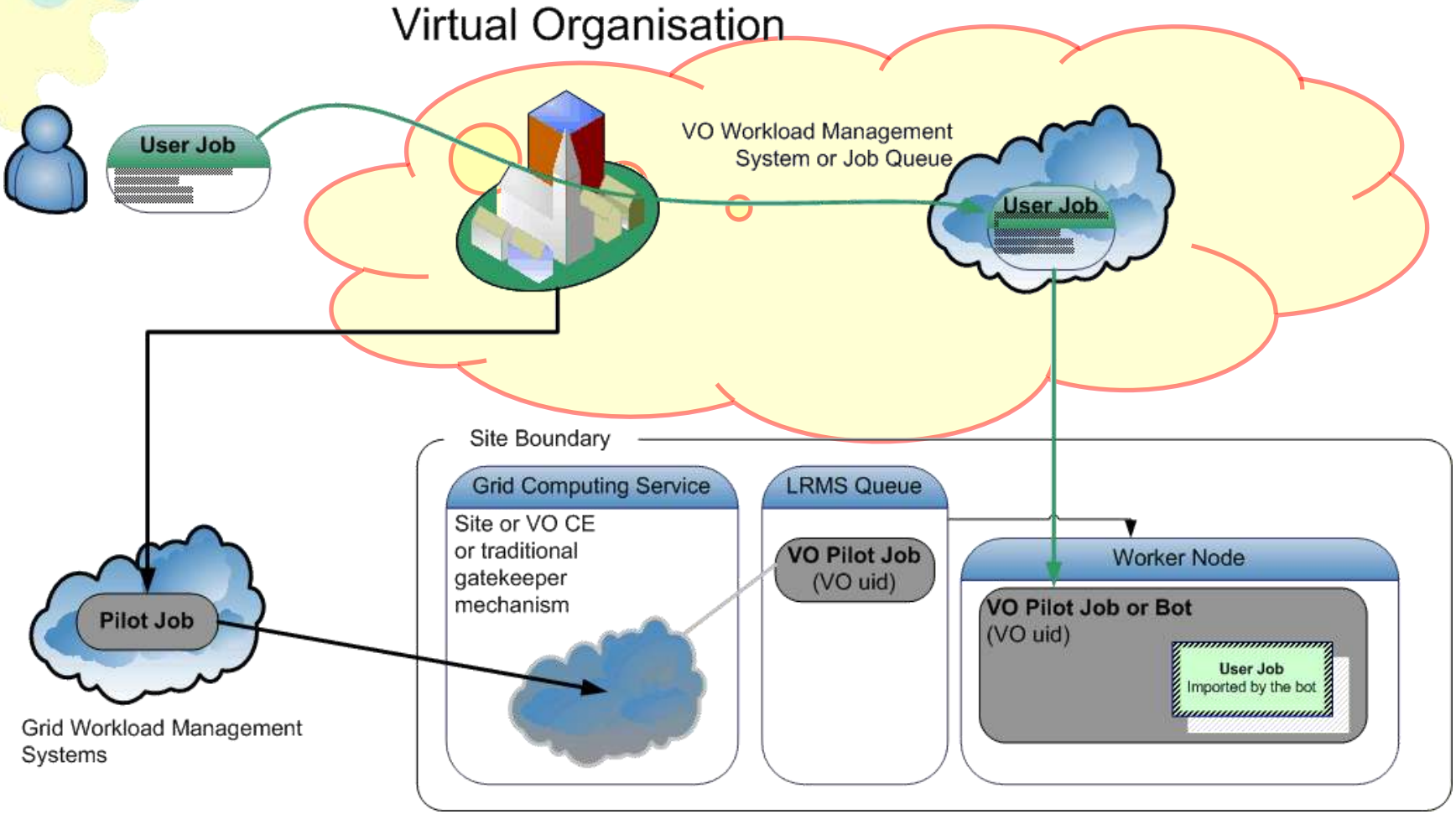
From: https://wlcg-tf.hep.ac.uk/wiki/Multi_User_Pilot_Jobs

- For any kind of pilot job:
  - Frameworks such as Condor glide-in, DIRAC, PANDA, … or Topos, are popular, because they are 'easy' (that's why there are so many of them!)
  - Single-user pilot jobs are no different than other jobs when you allow network connections to and from the WNs
  - Of course: any framework used to distribute payload gives additional attack surface

Multi-user pilot jobs

1. All pilot jobs are submitted by a single (or a few) individuals from a user community (VO)
   - Creating an overlay network of waiting pilot jobs
2. VO maintains a task queue to which people (presumably from the VO) can submit their work
3. Users put their programs up on the task queue
4. Pilot jobs on the worker node looks for work from that task queue to get its payload
5. Pilot jobs can execute work for one or more users in sequence, until wall time is consumed

VO overlay networks: MUPJ



BiG Grid
the dutch e-science grid

Pros and Cons of MUpilot jobs

In current 'you only see the VO pilot submitter' model:

- Loss of control over scheduling/workload assignment, e.g.
  - site admin cannot adjust share of specific user overloading e.g. the Storage Element (only the pilots are seen by the batch system) and might need to:
  - ban entire VO instead of user from the SE and/or CE, or
  - reduce the entire VO share
  - *Is that acceptable in case of a non-confirmed incident?*
- Traceability and incident handling issues

Advantages

- you only see & need to configure a single user
- It's not complicated, and no software/config is needed

Extensive summary of technical issues (pros and cons):
https://wlcg-tf.hep.ac.uk/wiki/Multi_User_Pilot_Jobs

Traceability and compromises

- In case of security incidents:
  - Complete & confirmed compromise is simple: ban VO
  - In case of suspicion: to ban or not to ban, that's the question
    - There is no 'commensurate' way to contain compromises
    - Do you know which users are inside the VO?
      No: the list is largely private
      No: it takes a while for a VO to respond to 'is this user known'?
      No: the VO will ban user only in case *they* think (s)he is malicious – that may be different from *your* view, or from the AIVD's view, or ...
    - So: the VO may or may not block
    - The site is left in the cold: there is no 'easy' way out except blocking the entire VO, which then likely is not 'acceptable'

Traceability and compromises

- Protecting user payload, other users, and the pilot framework itself from malicious payloads
  - To some extent a problem for the VO framework, not for the site
  - Not clear which payload caused the problem: all of them are suspect
  - User proxies (when used) can be stolen by rogue payloads
  - … or the proxy of the pilot job submitter itself can be stolen
  - Risk for other user to be held legally accountable
- Helps admins understand which user is causing a problem
- Sites may need proof of the identity of who was (or is about to!) use the resources at any time, in particular the identities involved in any ongoing incidents
  - Information supplied by the VO may be (legally) insufficient or too late
  - Privacy laws might hamper the flow of such information

RECOVERING CONTROL

BiG Grid
the dutch e-science grid

Recovering control: policy

- Policy on MUPJ
  - https://edms.cern.ch/document/855383
  - Implemented to varying degrees
  - Actual policy requires use of fine-grained control tools

- Document-based reviews of LHC VO frameworks
  - A wLCG working group reviewed the LHC VO frameworks based on documents (or slideware where no document was available)
  - No actual code review was done
  - Some frameworks are likely good (security was taken as part of design considerations), others unknown (but trivial exploits closed down after public demonstrations)

wLCG MUPJ Policy 1.0

1. Before submitting pilot jobs to a Site **the VO must have approval from the Grid and from that Site**.
2. Each pilot job must be the responsibility of one of a limited number of authorised and registered members of the VO. The VO is responsible for implementing a process for authorising pilot job owners and ensuring that they accept the conditions laid down here. The pilot job owner and the VO on behalf of whom the job is submitted are held responsible by the Grid and by the Site for the safe and secure operation of the pilot job and its associated user job(s).
3. The pilot job must only execute user jobs belonging to registered and authorised members of the VO.
4. **The pilot job framework must meet the fine-grained monitoring and control requirements defined in the Grid Security Traceability and Logging policy.** The use of gLexec in identity switchingmode is one solution that meets these needs.
5. The pilot job must use the approved system utility to map the application and data files to the actual owner of the workload and interface to local Site authorization, audit and accounting services. The owner of the user job is liable for all actions of that user job.
6. **The pilot job must respect the result of any local authorisation and/or policy decisions**, e.g. blocking the running of the user job.
7. The pilot job must not attempt to circumvent job accounting or limits placed on system resources by the batch system, for example the execution of more parallel jobs than allowed.
8. The pilot job framework must isolate user jobs from one another, including any local data files created during execution and any inter-process communication.
9. When fetching a user job and credentials into the worker node, the pilot job must use means at least as secure as the original pilot job submission process.
10. The Grid and/or the Sites reserve the right to terminate any pilot jobs and associated user jobs that appear to be operating beyond their authorisation and/or are not in compliance with this policy. Other possible consequences include blacklisting of users or the VO as a whole.
11. **The VO and/or pilot job owner must produce and keep audit logs**, as defined in the Grid Security Traceability and Logging policy**, and must assist Grid Security Operations in security incident response**.
12. The VO must make a description of the architecture, the security model and the source code of their pilot job system available to Grid Security Operations and/or Sites on request.

BiG Grid
the dutch e-science grid

Liabilities

- General statements on liability are hard to make
  - Dutch law is purposefully vague
  - Artikel 54a WvS today may provide some protection:
    "Een tussenpersoon die een telecommunicatiedienst verleent bestaande in de doorgifte of opslag van gegevens die van een ander afkomstig zijn, wordt als zodanig niet vervolgd indien hij voldoet aan een bevel van de officier van justitie, na schriftelijke machtiging op vordering van de officier van justitie te verlenen door de rechter-commissaris, om alle maatregelen te nemen die redelijkerwijs van hem kunnen worden gevergd om de gegevens ontoegankelijk te maken."
  - but we still don't know whom to block to prevent it from re-occuring
  - and it covers data, but not the offering of compute or network services

  - And then that is not the only relevant piece of text ...
    - are we doing enough for 'due diligence' ?
    - are we able to control and restrict illegal use of resources?
    - even when its illegal just in,e.g., NL, FR, DE, but fine elsewhere?

BiG Grid
the dutch e-science grid

Recovering control: mechanisms

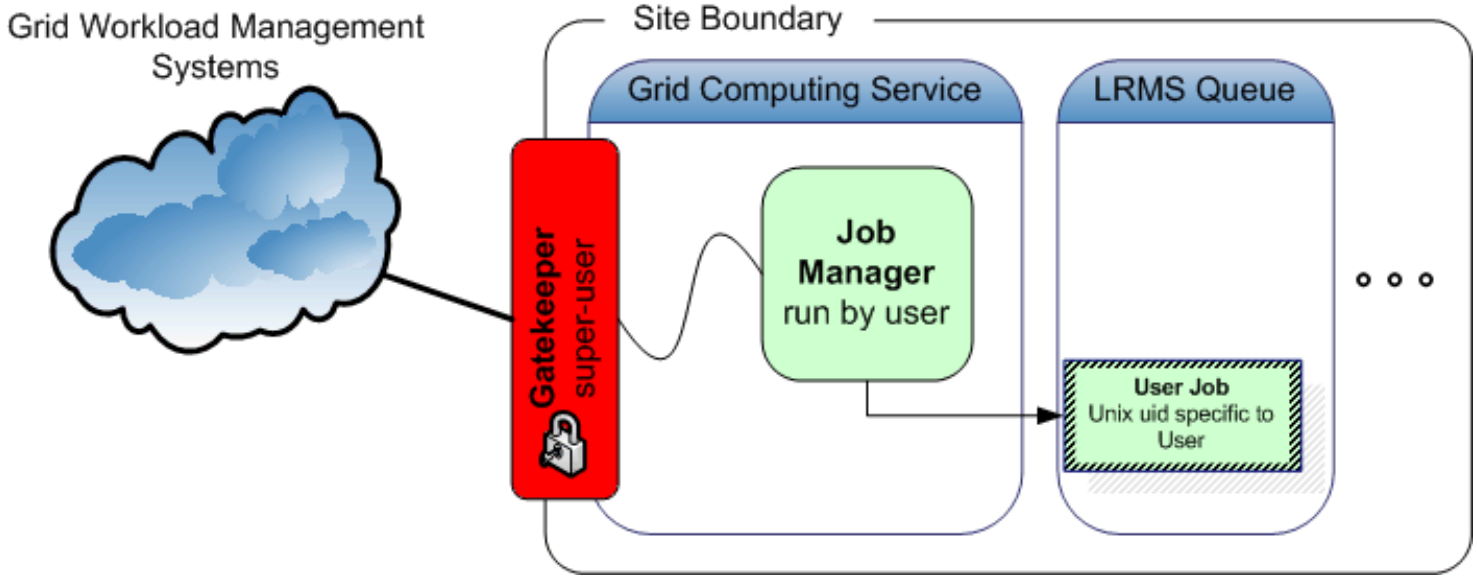
1. Unix-level sandboxing
   - POSIX user-id and group-id mechanisms for protection
   - Enforced by the 'job accepting elements':
     - Gatekeeper in EGEE (Globus and lcg-CE), TeraGrid and selected HPC sites
     - Unicore TSI
     - gLite CREAM-CE via sudo
2. VM sandboxing
   - Not widely available yet

... a slight technical digression on (1) follows ...

BiG Grid
the dutch e-science grid

Submission via a Gatekeeper

Traditional job submission scenario, model ‘gatekeeper’



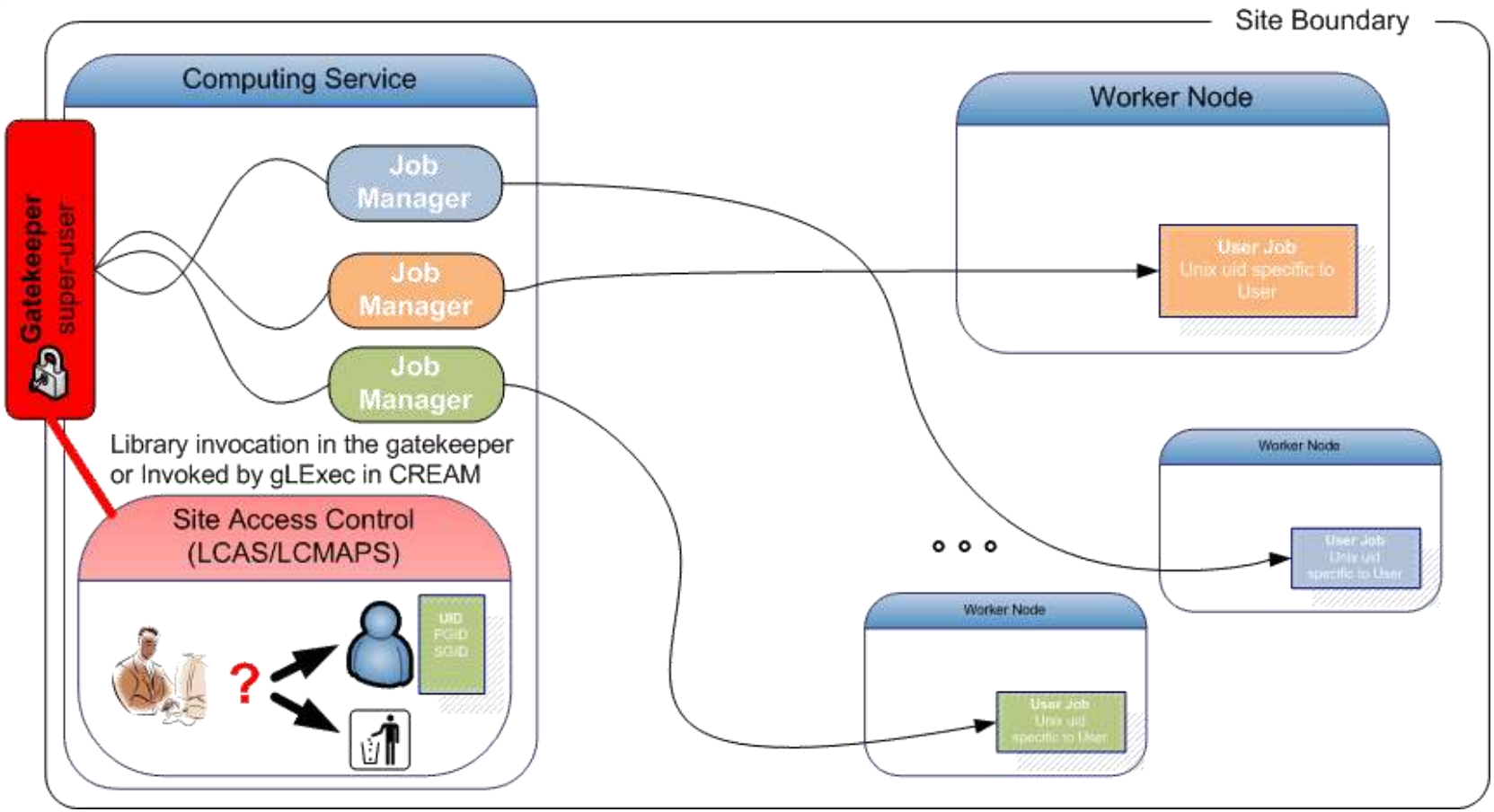
change of credentials at the site edge

networked service (‘gatekeeper’) with super-user privileges

job management in a per-user account (be it for single or multiple jobs)

BiG Grid
the dutch e-science grid

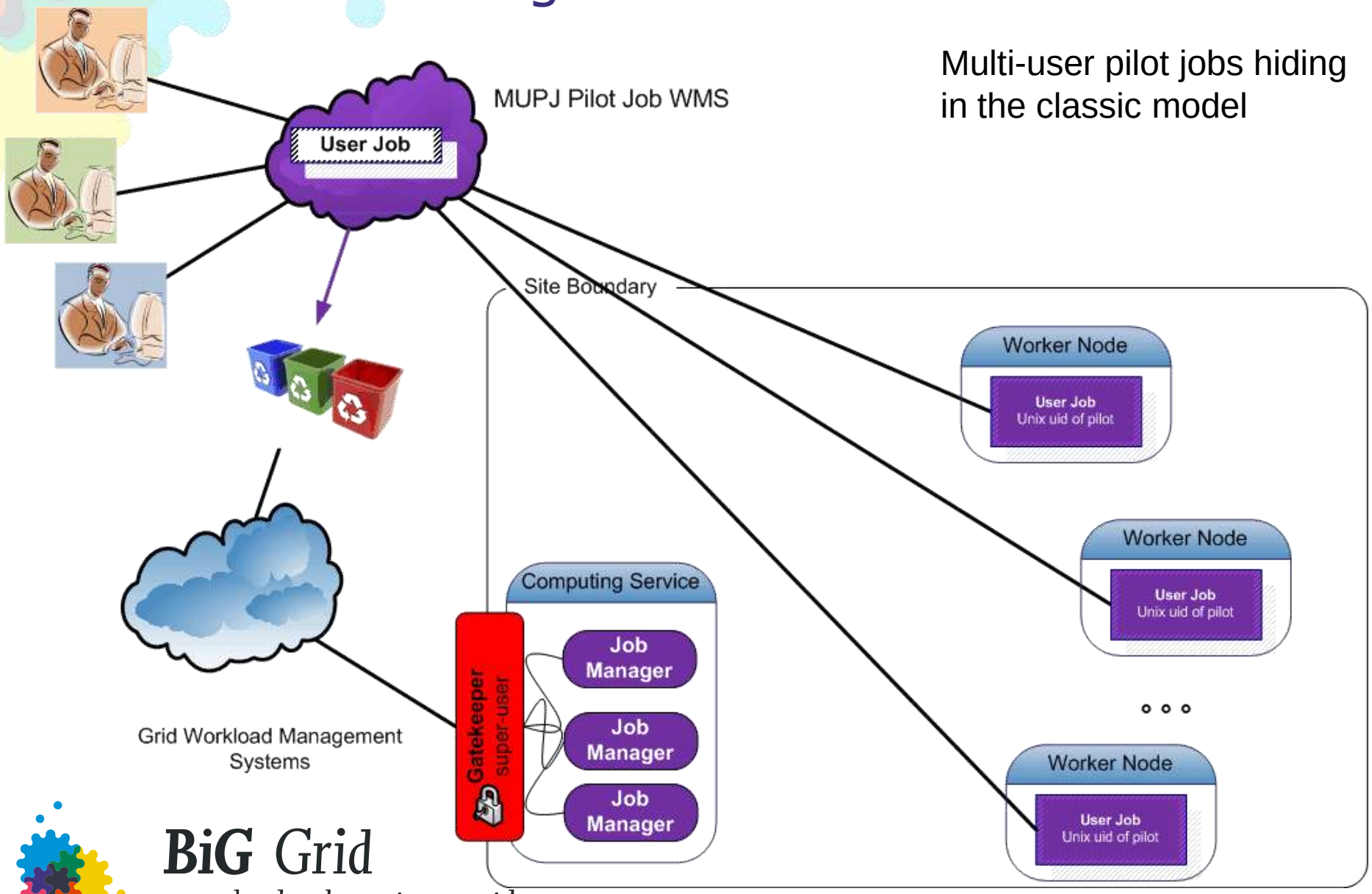
Implementing mapping: LCMAPS



BiG Grid
the dutch e-science grid

Pushing access control downwards

Multi-user pilot jobs hiding in the classic model

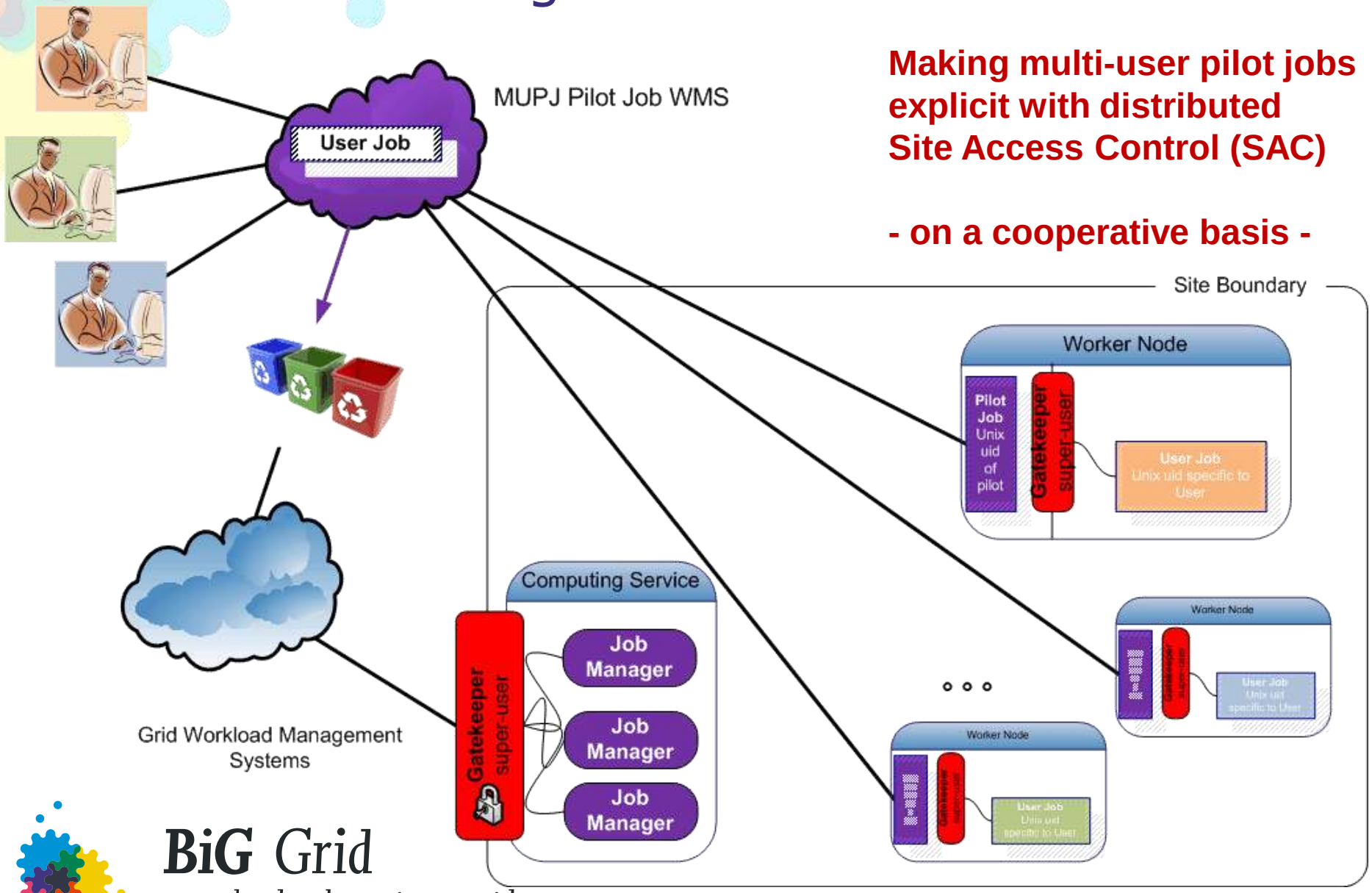


BiG Grid
the dutch e-science grid

Pushing access control downwards

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

- on a cooperative basis -



BiG Grid
the dutch e-science grid

Recovering Control

1. Make pilot job subject to normal site policies for jobs

- VO submits a pilot job to the batch system
  - the VO 'pilot job' submitter is responsible for the pilot behaviour

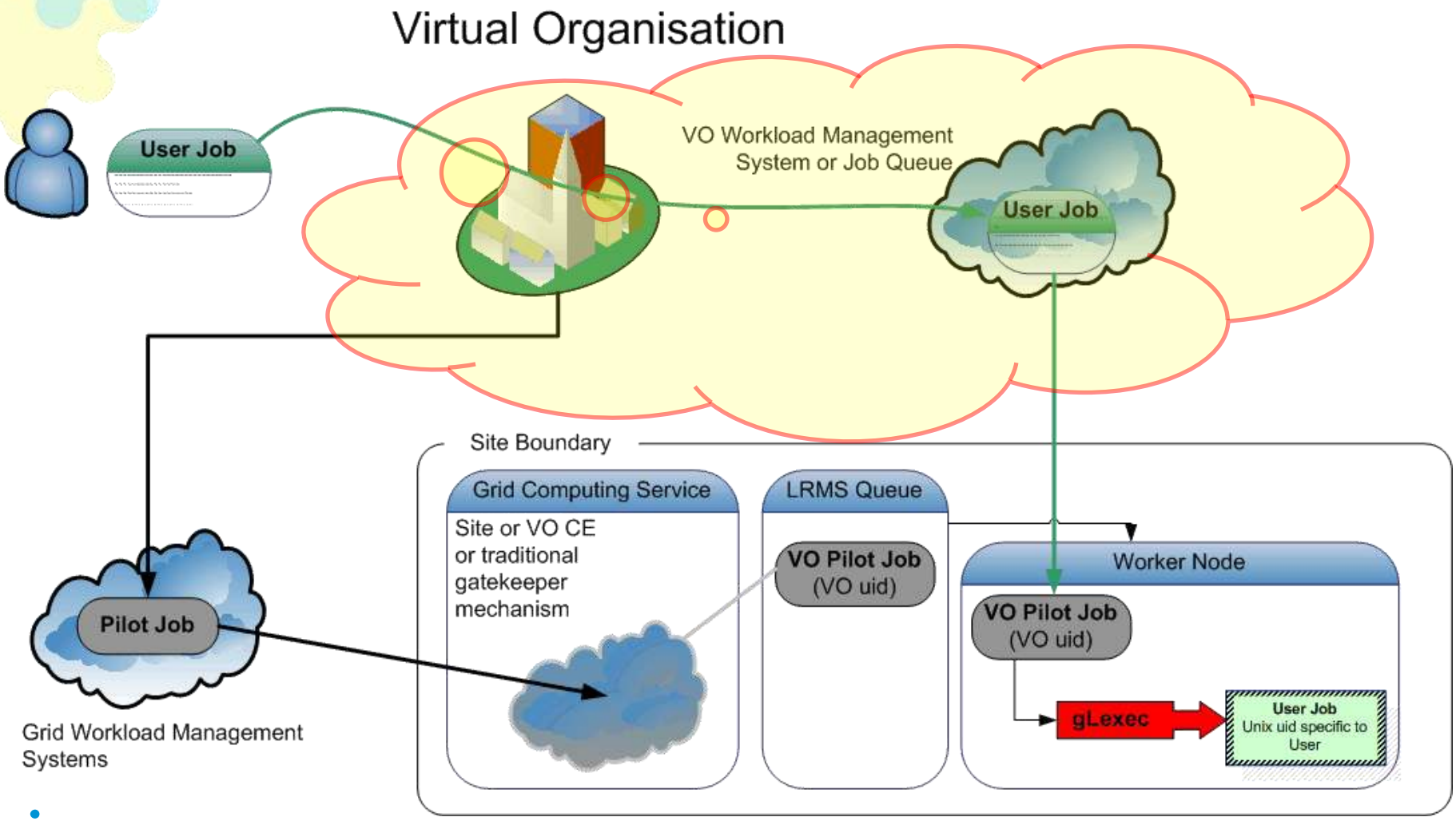
    this might be a specific role in the VO, or a locally registered 'special' user at each site
  - Pilot job obtains the true user job, and presents the user credentials and the job (executable name) to the site (glexec) to request a decision on a cooperative basis

2. Preventing 'back-manipulation' of the pilot job
   - make sure user workload cannot manipulate the pilot
   - project sensitive data in the pilot environment (proxy!)

- by changing uid for target workload away from the pilot

BiG Grid
the dutch e-science grid

Recovering control: gLExec



BiG Grid
the dutch e-science grid

GLEXEC

BiG Grid
the dutch e-science grid

What is gLExec?

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'

What gLExec does …

- User grid credential (subject name, VOMS, …)
- command to execute
- *current uid allowed to execute gLExec*

cryptographically protected by CA or VO AA certificate

gLExec

Authorization ('LCAS')
check white/blacklist
VOMS-based ACLs
is executable allowed?
…

Credential Acquisition
voms-poolaccount
localaccount
GUMS, …

LCMAPS

'do it'
LDAP account
posixAccount
AFS, …

Execute command with arguments as user (*uid, pgid, sgids …)*

BiG Grid
the dutch e-science grid

Sources of gLExec

- A History of gLExec
  - Taken from apache suexec(8)
  - Modified for certificate parsing and LCMAPS
  - Added more restrictions to execution environment
    - Start with minimal permissions
    - On request of end-users, review each requested feature
    - Like allowing execution of symlinks (or /bin/sh does not work on a linux system ...)
  - Early alpha code-reviewed by UW-Madison MIST team
    - Asset-based review, looking for exploitable code
    - By the time they finished, the root-exploitable vulnerabilities found has already disappeared ...
  - Additional source review by PSNC (Poznan) people

gLExec, early deployments

- Deployed in Autumn 2008 at FNAL
  - Driven by urgent need to comply with DoE regulatory constraints
  - Resulted in lots of interoperability and code testing
  - Deployed on productions systems since end 2008

- EGEE PPS and production deployment
  - Nikhef, IN2P3 Lyon, FZK, various T2s in UK
  - Focus in install-ability – but picked on purpose widely varying sites (including one where the system administrators did not have root access ...)

Built-in controls (1)

- Configuration and start-up
  - Configuration file must be protected
  - $PATH is reset regardless, to compiled-in default
  - All environment is cleaned, except for designated variables TZ,X509_USER_PROXY, GLEXEC_* and those specified in config

- On invocation
  - Invoking ruid and rgid must be registered users
  - Invoking ruid must be in white list (from config file) or must be member of compiled-in access group
  - Invoking user must have a valid proxy cert
  - Uses privilege separation (also for reading source files)

Built-in controls (2)

- Identity switching
  - New uid and gids must be larger than compiled-in minimum
  - Regardless of mapping source, target uid must not be 0
  - Target username and group must be reasonable and must exist

- Installing the proxy cert of the target user
  - Write with target credentials
  - Does not write to symlinks
  - Only re-writes proxy files as the target user itself (switches euid/egids before writing)
  - Permissions on target proxy are always 0600

Built-in controls (3)

- Invocation of payload
  - Target proxy must either exist and be owned by target user, or directory must be owned by target uid
  - Neither CWD nor target program must not be writable by others, and target program must not be setuid
  - CWD and program must be owned by either invoker or target user, or by root
  - Glexec can linger around to manage process, or just execve it (but then you do not get usage statistics at the end)

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 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…

Pieces of the solution

VO supplied pilot jobs must observe and honour

the same policies the site uses for normal job execution

(e.g. banned individual users)

Three pieces that go together:

- glexec on the worker-node deployment
  - the mechanism for pilot job to submit themselves and their payload to site policy control
  - give 'incontrovertible' evidence of who is running on which node at any one time (in mapping mode)
    - gives ability to identify individual for actions
    - by asking the VO to present the associated delegation for each user
  - VO should want this
    - to keep user jobs from interfering with each other, or the pilot
    - honouring site ban lists for individuals may help in not banning the entire VO in case of an incident

Pieces of the solution

- glexec on the worker-node deployment

- keep the pilot jobs to their word
  - mainly: monitor for compromised pilot submitters credentials
  - process or system call level auditing of the pilot jobs
  - logging and log analysis

- gLExec cannot to better than what the OS/batch system does
  - 'internal accounting should now 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
    - accounting based glexec jobs requires a large and unknown effort
  - time accrual and process tree remain intact across the invocation
    - but, just like today, users can escape from both anyway!

Installation

- Actually only identity mapping mode really helps

  Otherwise

  – back-compromise (and worm infections) remain possible
  – attributing actions to users on WN is impossible (that needs a uid change)

Starting and Killing Jobs

The batch system performs the following basic functions

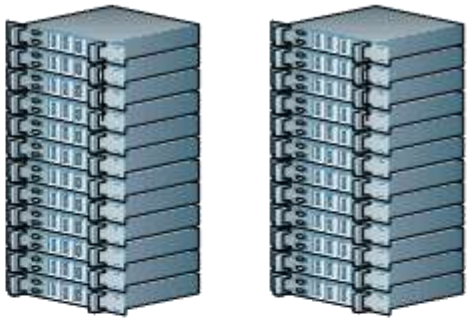
1. Job Submission
2. Job Suspend/Resume
3. Job Kill
4. CPU time accounting

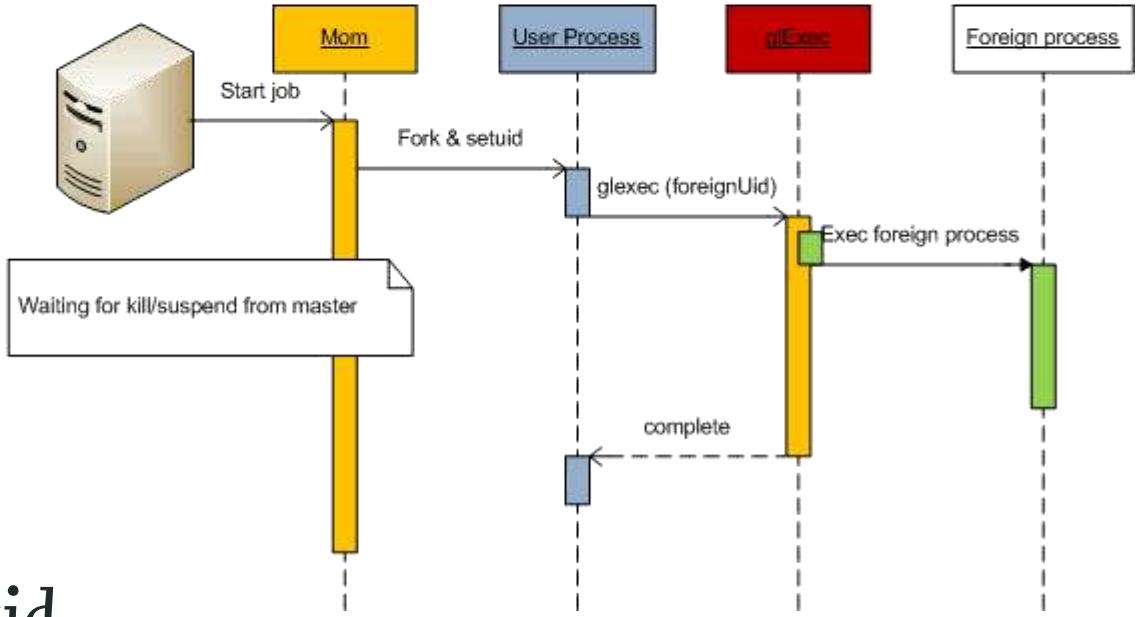
does not yet address enforcing sanity and user compliance

BiG Grid
the dutch e-science grid

from the test description of Ulrich Schwickerath

Starting and Killing Jobs





BiG Grid
the dutch e-science grid

Testing it

Can batch system suspend/kill the gLExec'ed processes?

- Test using gLExec itself
  - Most 'true' tests
  - Requires installation of gLExec and all dependencies
- Test using the *sutest* mini programme
  - Same logic, but a stand-alone small (50-line) C programme
  - No dependencies
  - A few hard-coded constants to be set before compilation
  - Trivial to test

Session Preservation

```
$ date && ps --forest -eo pid,ppid,sess,user,uid,euid,cmd

Fri Feb  8 14:02:41 CET 2008

  PID  PPID  SESS USER   UID UID CMD
3294     1  3294 root      0   0 /usr/sbin/pbs_mom
18668  3294 18668 davidg 502 502  \_ -bash
18713 18668 18668 davidg 502 502      \_ /bin/sh …/jobs/33.tbn05.ni.SC
18715 18713 18668 nobody  99  99          \_ /project/sutest /bin/sleep 120
18716 18715 18668 nobody  99  99              \_ /bin/sleep 120
...
```

... and Torque will kill all processes in the tree:

```
tbn05:~:1018$ cat tmp/tt.pbs
#! /bin/sh
date
/project/sutest /bin/sleep 120
date
```

```
$ date && qsub -q test tmp/tt.pbs
Fri Feb  8 14:02:21 CET 2008
33.tbn05.nikhef.nl
```

All vanishes after a 'qdel 33' ...

CPU accounting

- No change with respect to current behaviour of jobs
- Times are accumulated on wait and collated with the gLExec usage

Forcing havoc on yourself

```
$ ( date && ./sutest /bin/sleep 60 && date )
Fri Feb  8 16:41:24 CET 2008
Notice: identity changed to uid 99
```

```
...
 7508     1  7508 root     0 /usr/sbin/sshd
32122  7508 32122 root     0  \_ sshd: davidg [priv]
32124 32122 32122 davidg 502  |   \_ sshd: davidg@pts/0
32126 32124 32126 davidg 502  |       \_ -bash
17001 32126 32126 davidg 502  |           \_ -bash
17003 17001 32126 nobody  99  |               \_ ./sutest /bin/sleep 60
17004 17003 32126 nobody  99  |                   \_ /bin/sleep 60

# kill -9 17001
```

```
Killed
```

```
17003     1 32126 nobody      99    99 ./sutest /bin/sleep 60
17004 17003 32126 nobody      99    99  \_ /bin/sleep 60
```

BiG Grid
the dutch e-science grid

Cleaning up files

File cleanup: what do sites use today?

- Check for files owner by users not currently running a job?
  - Who 'is running' becomes ill defined
  - Need a 'back-mapping' tool that can trawl log files or a state dir
- tmpwatch(8) for old files?
  - Change of uid does not influence this solution
- Transient TMPDIR facilities (PBSPro, Torque 2+)?
  - Runs with root privileges anyway
  - TMPDIR is inherited by the gLExec'ed child
  - And is thus unaffected by gLExec
  - will be cleaned as intended

Pruning stray processes

- Killing stray user processes
  ‘not owned by a user with a currently running process’
  - E.g. used at CERN/LSF
  - Need a ‘back-mapping’ tool that can trawl log files or a state dir
  - But: *is not trustworthy to begin with on multi-job-slot machines*!

- Kill processes that are ‘too old’
  - Will run as root anyway
  - Unaffected, but is not trustworthy either

- Kill processes not ‘parented’ in a batch job
  - gLExec will preserve the process tree, and thus this will work
  - Will also slaughter daemonizing jobs today … a Good Thing™

Pruning User Processes

https://www.nikhef.nl/grid/sysutils/prune_users/

- For Torque in perl (simple migration to other systems)
- Kill processes that are not a child of a registered pbs_mom
- Uses the *momctl* command on the node

- Caveats
  - Will usually not kill processes with a uid < 99
  - May optionally preserve top-level sshd sessions (beware of MPI)
  - Does not protect against fork bombs

DEPLOYMENT ON A CLUSTER

BiG Grid
the dutch e-science grid

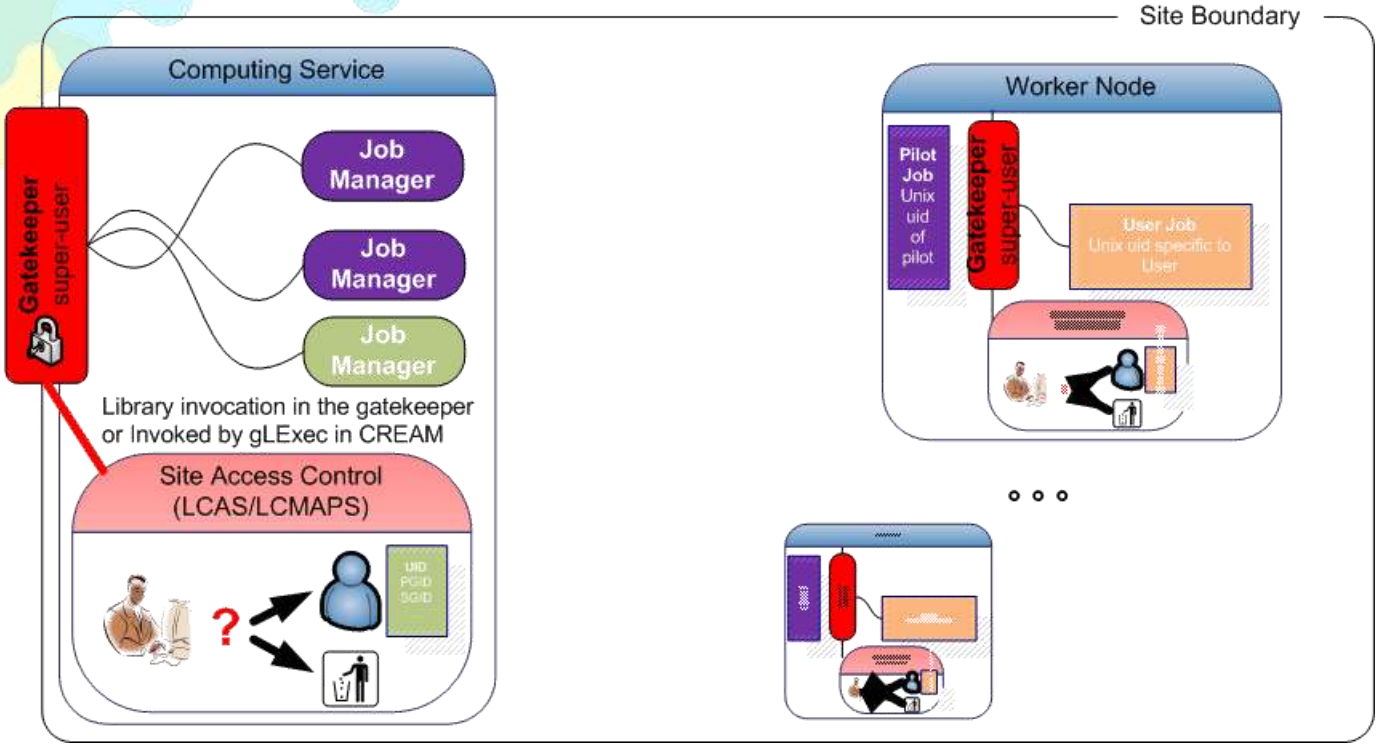
Deploying on a Cluster

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*

Node-local access control

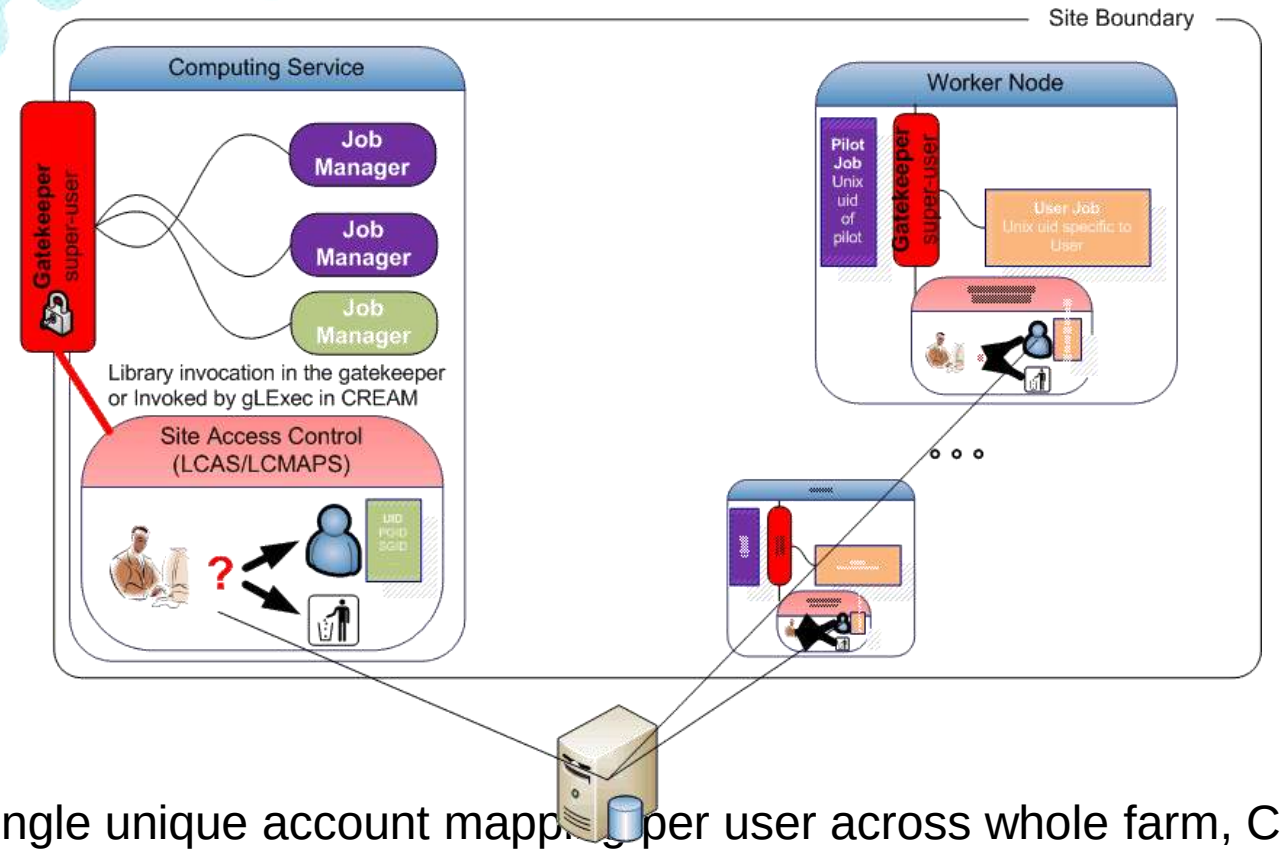


PRO no single points of failure
well defined number of pool accounts (as many as there are job slots/node)
containment of jobs (no cross-WN infection)

CON need to distribute the policy through fabric management/config tools
no cross-workernode mapping (e.g. no support for pilot-launched MPI)

BiG Grid
the dutch e-science grid

WN-coordinated access control

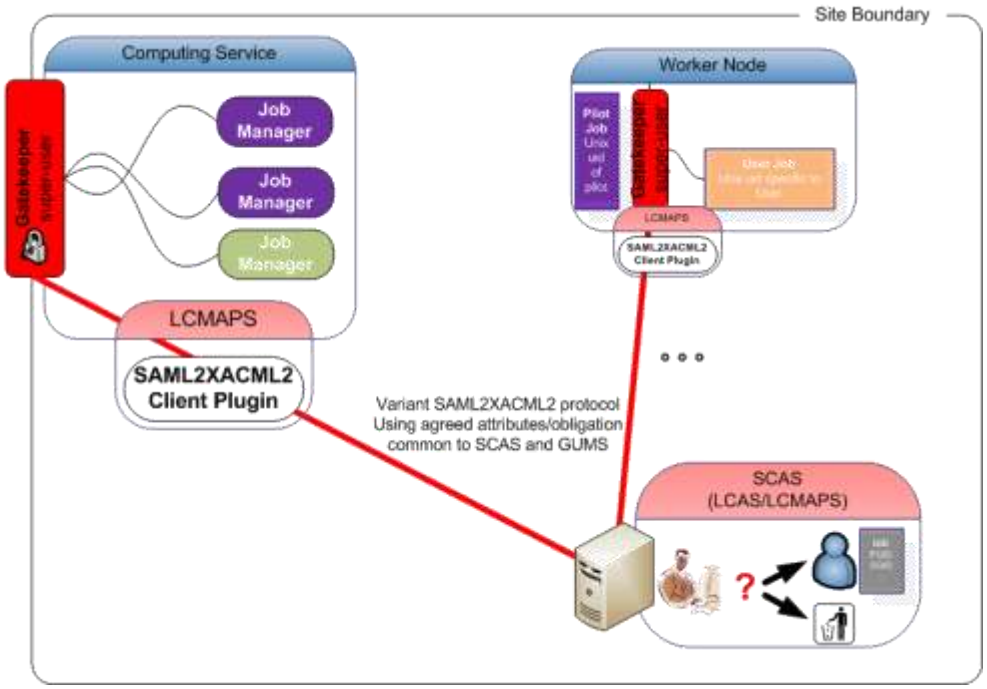


PRO single unique account mapping per user across whole farm, CE, and SE
transactions database is simple (implemented as an NFS file system)
communications protocol is well tested and well known

CON need to distribute the policy through fabric management config tools
coordination only applies to the account mapping, not to authorization

BiG Grid
the dutch e-science grid

Site-central access control



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 sites
still cannot do credential validation (formal issues with the protocol)

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

BiG Grid
the dutch e-science grid

Centralizing decentralized SAC

Supporting consistent

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

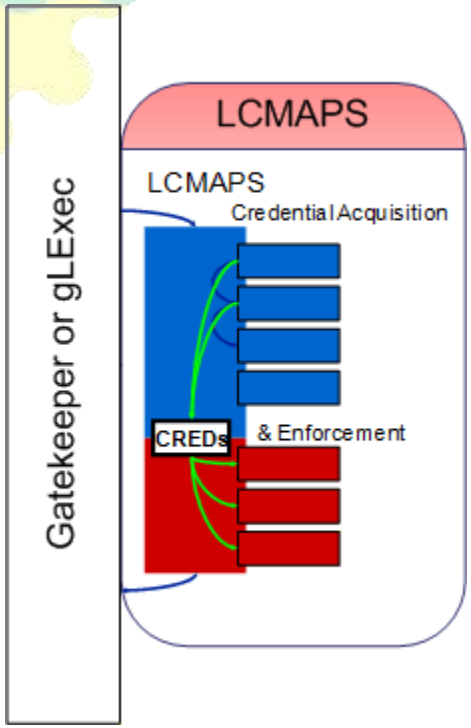
via the

Site Central Authorization Service SCAS

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

BiG Grid
the dutch e-science grid

Local LCMAPS

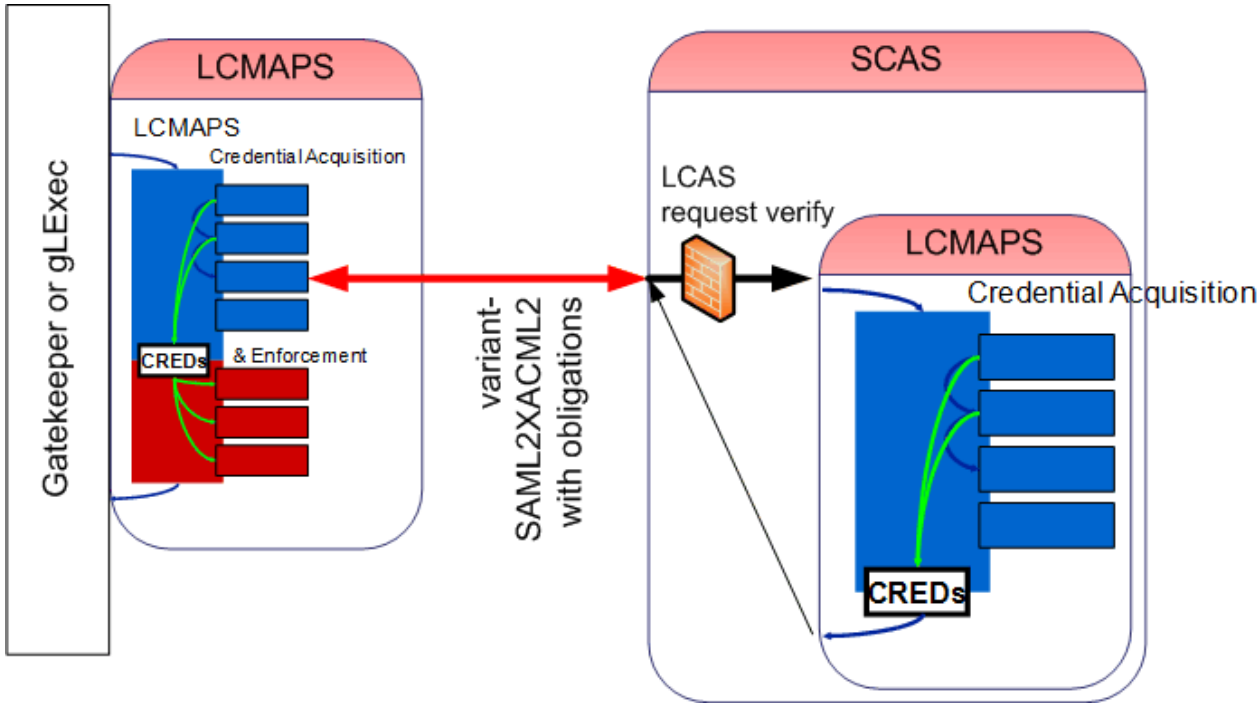


- Linked dynamically or statically to application
- does both credential acquisition
  - local grid map file
  - VOMS FAQN to uid and gids
- and enforcement
  - setuid
  - krb5 token requests
  - AFS tokens
  - LDAP directory update

LCAS is similar is use and design, but makes the basic Yes/No decision

BiG Grid
the dutch e-science grid

SCAS: LCMAPS in the distance

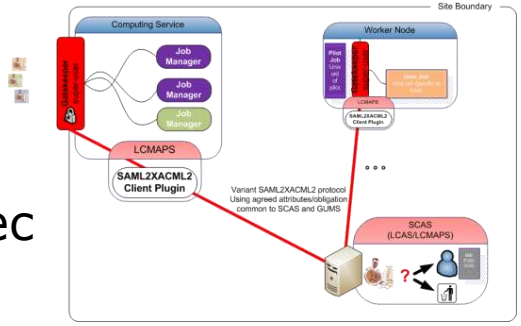


- Application links LCMAPS dynamically or statically, or includes Prima client
- Local side talks to SCAS using a variant-SAML2XACML2 protocol
  - with agreed attribute names and obligation between EGEE/OSG
  - remote service does acquisition and mappings
  - both local, VOMS FAQN to uid and gids, etc.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

BiG Grid
the dutch e-science grid

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
  - 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



BiG Grid
the dutch e-science grid

Supported services

- SCAS communicates based on a few standards and the joint “Authorization Interoperability” profile
  - Supported by Globus, gLite 3.x, VDT, OSG
  - Defines common naming of obligations such as uid/gid
- Compatible software
  - Globus gatekeepers, lcg-CE
  - gLExec (on WNs and on CREAM-CEs)
  - dCache > 1.9.2-4
  - GT GridFTP
  - GT4.2 WS-GRAM, GRAM5

TOWARDS MANAGED AUTHORIZATION

BiG Grid
the dutch e-science grid

Central banning

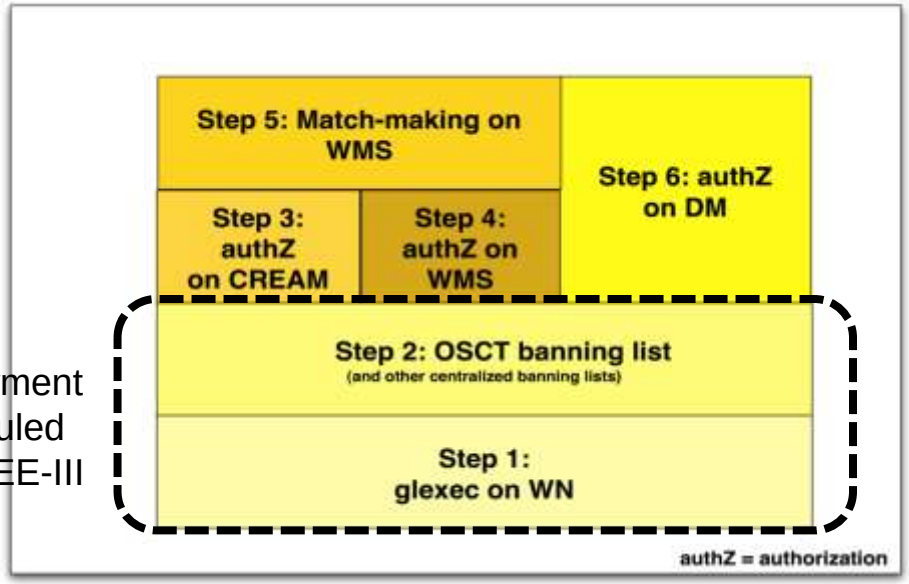
- EGEE (and EGI) Operational Security teams will be maintaining a list of all 'banned' users
  - Rapid reaction (preferably within 6 hrs)
  - When consistently deployed, will help contain incidents

- Implementations
  - Usually done (easily) via standard fabric management using service-native tools (LCAS, GACL for WMSes, DB entries for DPM, SAZ via gPlazma where configured)
  - For SCAS-enabled services: can be done in one place on the site
  - Grid-wide banning without site intervention is not trivially possible today (but could be scripted)

Next? Argus

- Creating a homogeneous authZ interface: Argus
  - For the moment specific to gLite
  - Not in production yet –
    and you really ought to wait for release >= 1.1
    which is due to start certification on Jan 15th, 2010
  - Supports distributed policy administration and automatic propagation of the ban list
  - Phased deployment foreseen

Deployment scheduled for EGEE-III



BiG Grid
the dutch e-science grid

Deployment options

- Argus and SCAS can be deployed ...
  - As alternatives (when Argus is complete)
  - In parallel, at the service side
  - In sequence, with SCAS fronting it

- For the moment, though, Argus is only certified for the gLExec-on-WN use case

CONCLUSIONS

BiG Grid
the dutch e-science grid

Conclusions

- Grid access model has changed fundamentally with the introduction of MUPJs
  - Both benefits and drawbacks to the new situation
  - But this reality cannot be ignored, nor banned
- There are good reasons to retain control
  - Ability to react commensurately
  - Implementation of site-local or national controls
  - Systems-operational reasons
- Regaining control through both of:
  - Policy: JSPG MUPJ policy document, reviews
  - Systems tooling: gLExec-on-WN in mapping mode
  - Monitoring: compliance verification of VO
- Deployment: locally, site-centralised, or ...

QUESTIONS?

BiG Grid
the dutch e-science grid