glexec deployment models

local credentials and grid identity mapping
in the presence of complex schedulers

Jeff Templon

(slides mostly shamelessly stolen from David Groep)

NIKHEF

What is glexec?

glexec

a thin layer
to change unix 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(8)`'**
- **'a program wrapper around LCAS, LCMAPS or GUMS'**

What glexec does

Input

1. a certificate chain, possibly with VOMS extensions
2. a user program name & arguments to run

Action

1. check authorization (LCAS, GUMS)
   - user credentials, proper VOMS attributes, executable name
2. acquire local credentials
   - local (uid, gid) pair, possibly across a cluster
3. Execute user program with these credentials

Result

1. user program is run with the mapped credentials

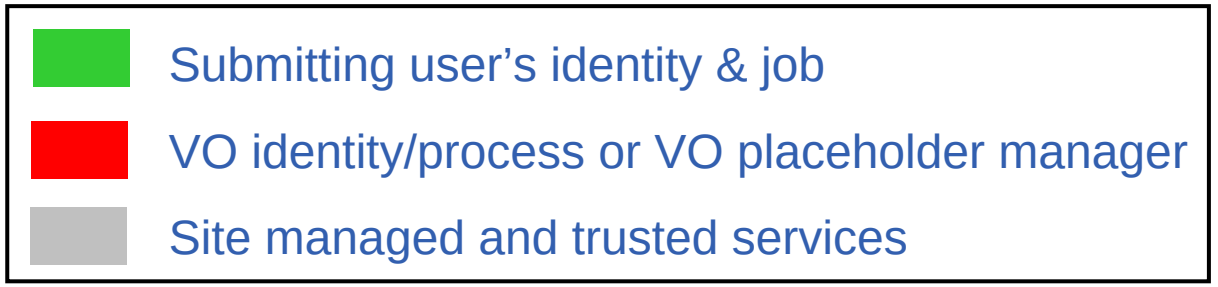
Why was glexec devised?

- **gatekeeper and other schedulers are complex, and are now being run with root privileges …**
  - Compare with Apache httpd, where user *cgi* scripts can be run under their own identity, but without the web server itself having to run as root
  - to accomplish this, a small, program is needed with setuid(2) power: ‘suexec(8)’
- **Achieve the same for variety of grid job submission systems**
  - need a common way of obtaining and enforcing site policy and credential mapping
  - without the need to modify each and every system
  - as such, glexec in this deployment mode is an alternative to having authorization and mapping *call-out*s in each system

glexec traditional deployments

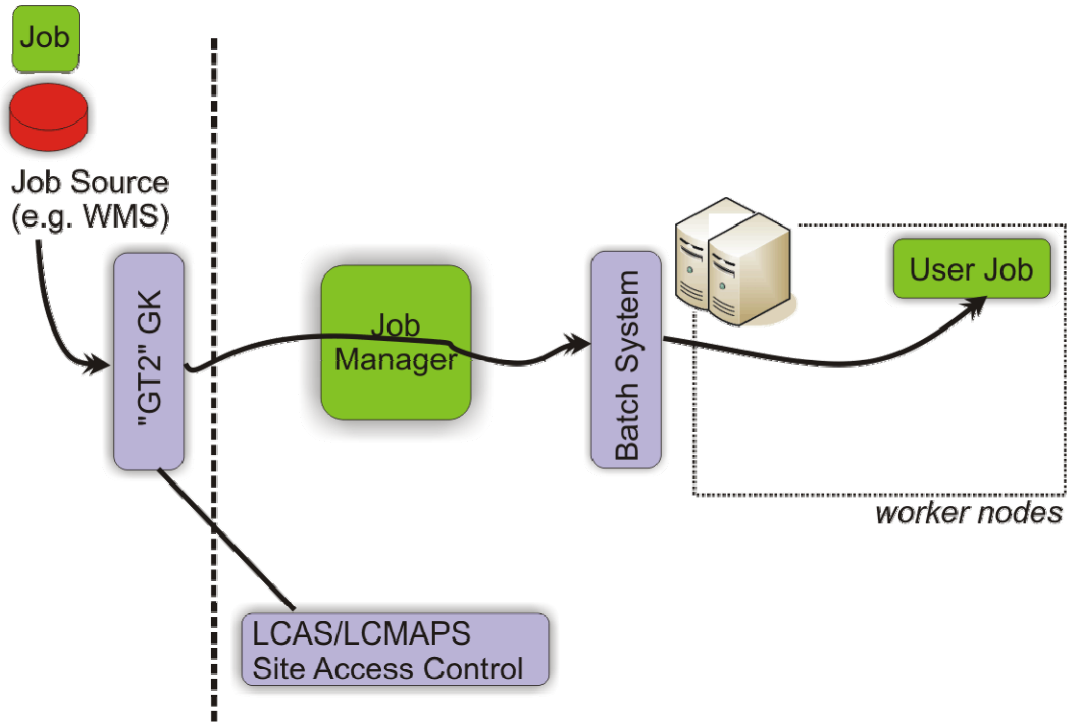
There are three ‘traditional’ deployment models, where glexec has a role in two of these

1. direct per-user job submission to a ‘gatekeeper’ running with root privileges (GT2GK, today’s model)
2. a non-privileged dedicated CE or scheduler, accepting authenticated user jobs and submitting to the batch system
3. on-demand CE, submitted by VO or user to a front-end system, that then receives user jobs and submits these to the batch system



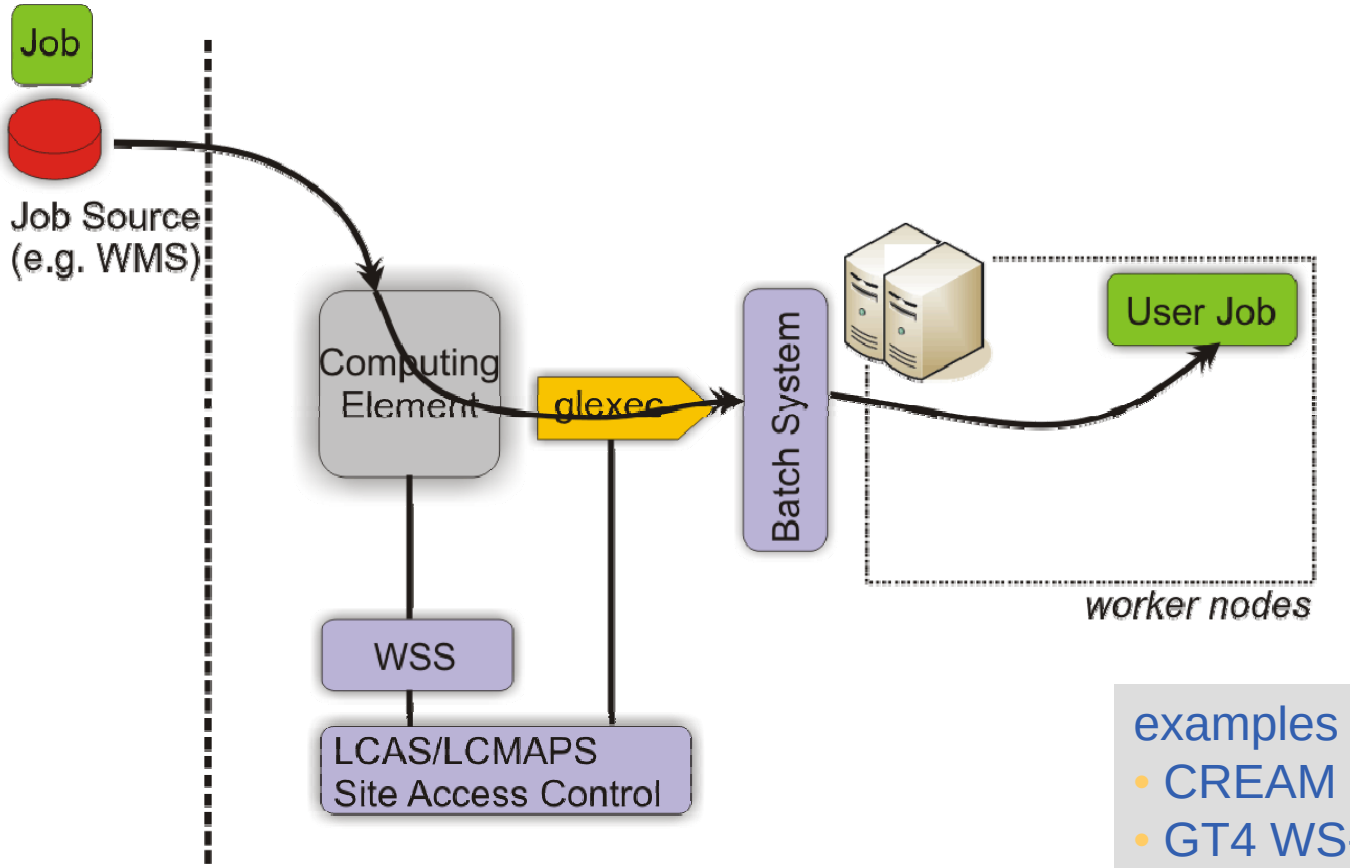
Jobs submission today (GT2 GK)

- **Deployment model without glexec (‘mode GT2GK’)**
  - jobs are submitted with *an identity* (hopefully the original user’s one) to the site Gatekeeper running as root
  - one job manager is run for each user on the head node
  - with the user’s (uid,gid) as set by the gatekeeper



Glexec in a one-per-site mode

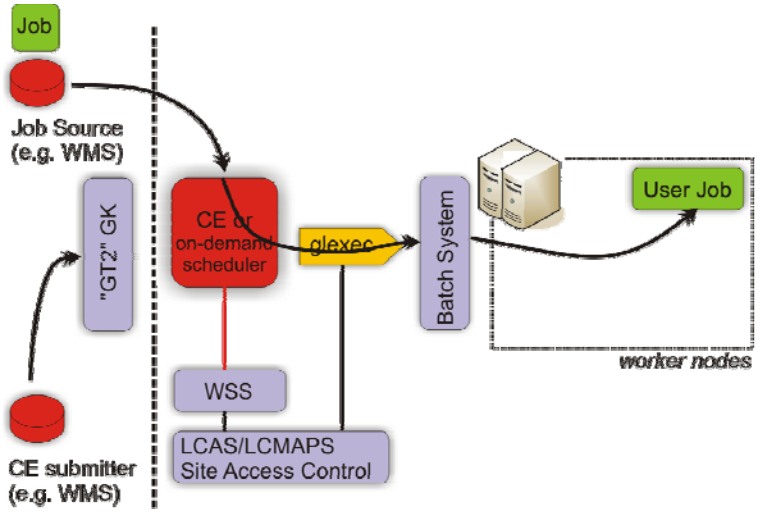
- **Deployment model with a CE 'service'**
  - running in a non-privileged account or
  - with a CE run (maybe one per VO) on a single front-end per site



examples
- CREAM
- GT4 WS-GRAM

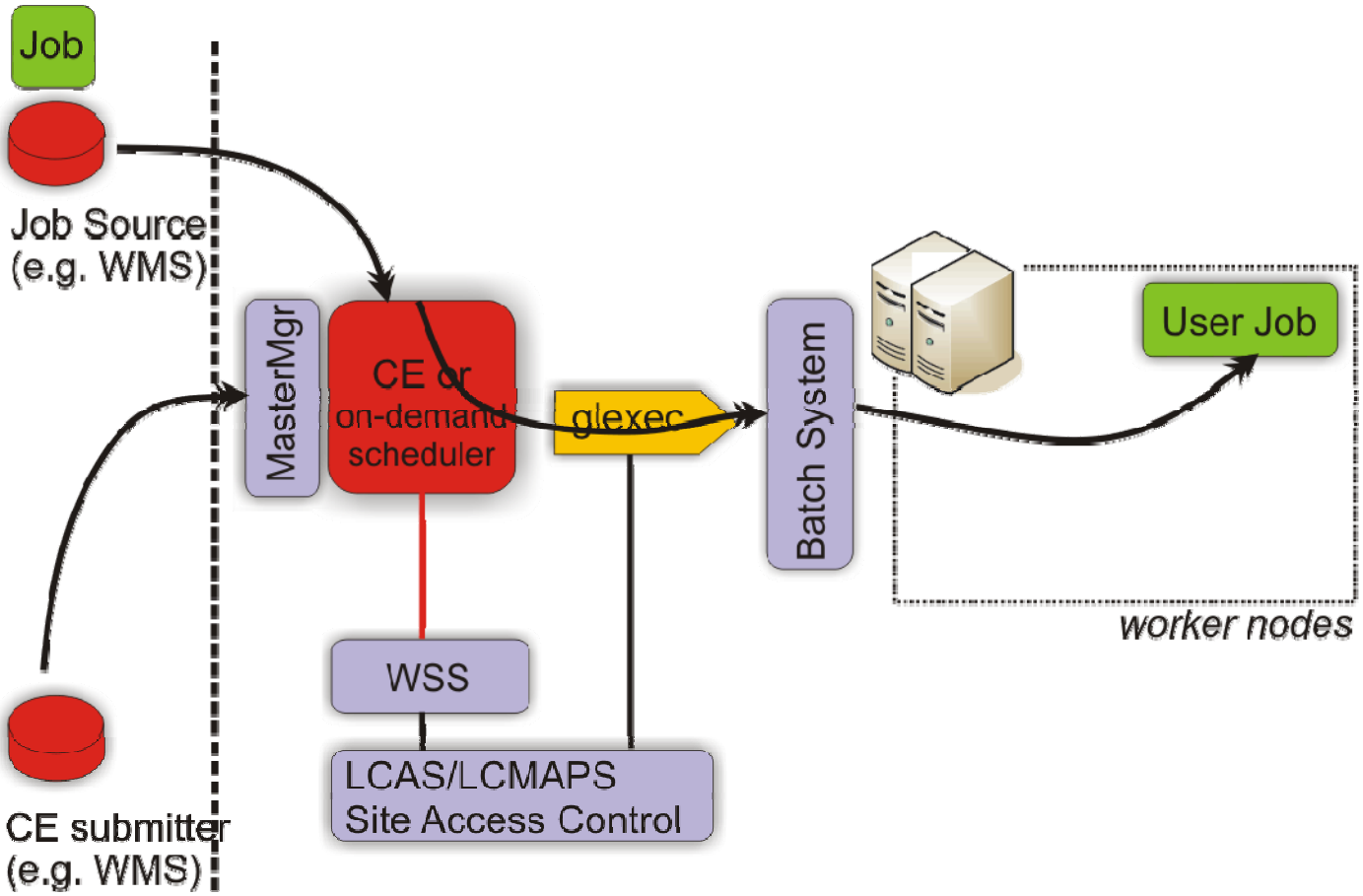
glexec with an on-demand CE

- **Deployment model with on-demand CEs ('mode on-demand CEs')**
  - The user or the VO start their own scheduler on a front-end system
  - All these on-demand schedulers are resource-limited by a site-managed master scheduler (via a GT2GK or Condor)
  - the on-demand schedulers eat jobs for their VO or user
  - and set the proper identity *before the job gets submitted to the site batch system*



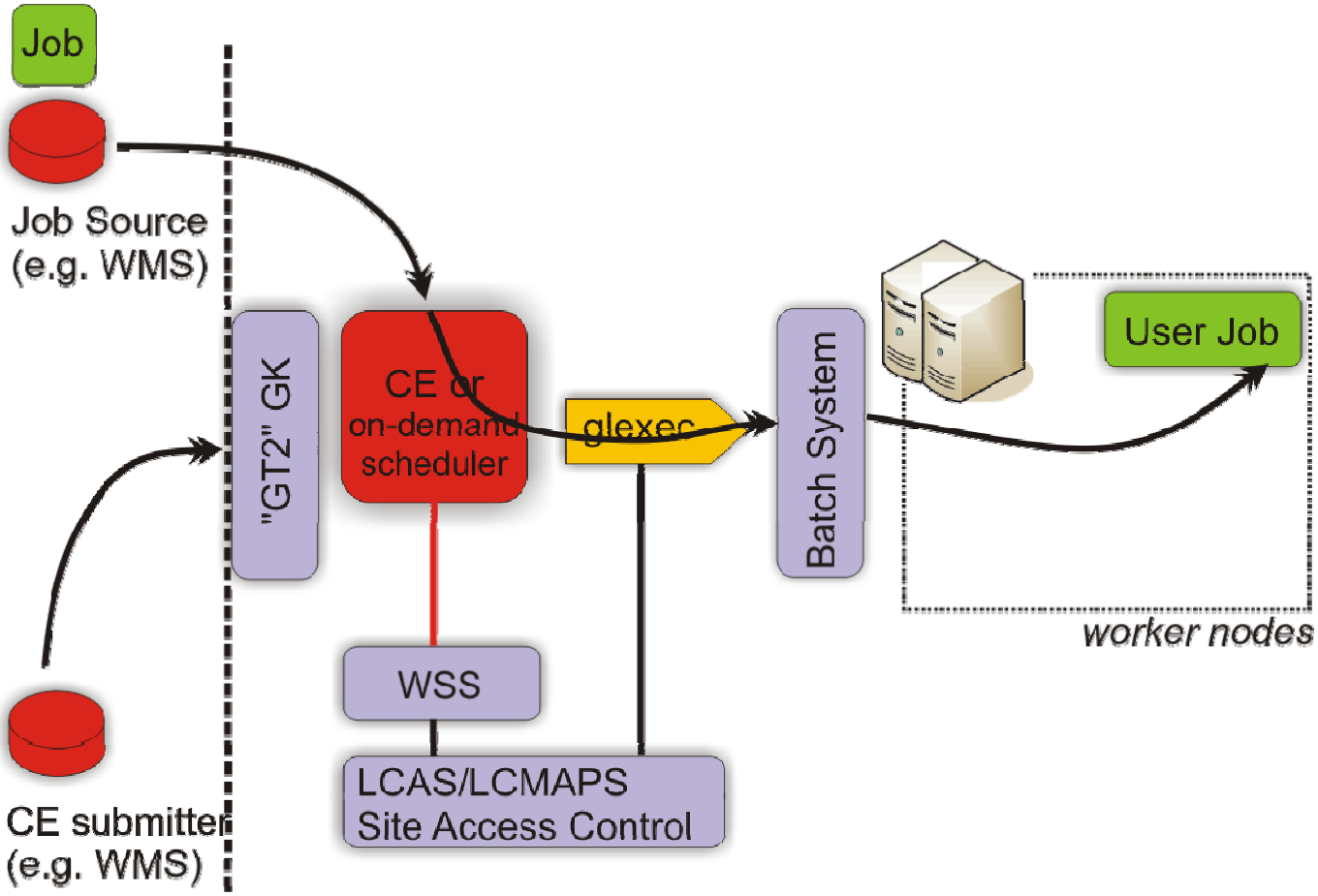
glexec with on-demand CE

- **Deployment model with on-demand CEs ('mode on-demand for VOs' with native interface)**



glexec with an on-demand CE

- **Deployment model with on-demand CEs ('mode on-demand for VOs' with legacy interface)**



Traditional model summary

- In all three models, the 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 of Unix process on worker nodes are per-user

Pilot jobs

A pilot job is basically just

- a small script which downloads a real job
- from a repository once it starts executing, hence
- it is not committed to any particular task, or perhaps even a particular user, until that point.
- If there are no tasks waiting the pilot job exits immediately.
- In principle, if the time limits on the queue are long enough a single pilot job could run more than one real job, although I'm not sure if anyone is actually doing that at the moment.

(thanks to Stephen Burke, on LCG-ROLLOUT)

Reasons

- **Don't run *my* job until everything is OK**
  - Get WN via pilot jobs : don't fail user jobs, fail pilots
  - Drastically reduce resubmission of failed / aborted user jobs
- **Run the jobs *I* think are important**
  - Impossible to exactly predict execution order
  - So dictate what jobs must execute as they start
- **User payloads can be more 'flavor independent'**
- **Reduce overhead by putting multiple user payloads into single WN LRMS job instance**

Situation today

- **'VO-type' pilot jobs submitted as regular user jobs**
  - run with the identity of one or a few individuals from a VO
  - obtain jobs from any user (within the VO) and run that payload on the WN allocated
  - site 'sees' only a single identity, not the true owner of the workload
  - no effective mechanisms today can deny this use model
- **note that this does not apply to the regular 'per-user' pilot jobs (also in current circulation)**

glexec-on-WN rationale

Issues that drove the original scenario:

- VO supplied pilot jobs must observe and honour
  - **the same policies the site uses for normal job execution**

  preferably
  - without requiring alternate mechanisms to describe the policies
  - be continuously in synch with the site policies

again, 'per-user' pilot jobs satisfy these rules by design

glexec-on-WN rationale

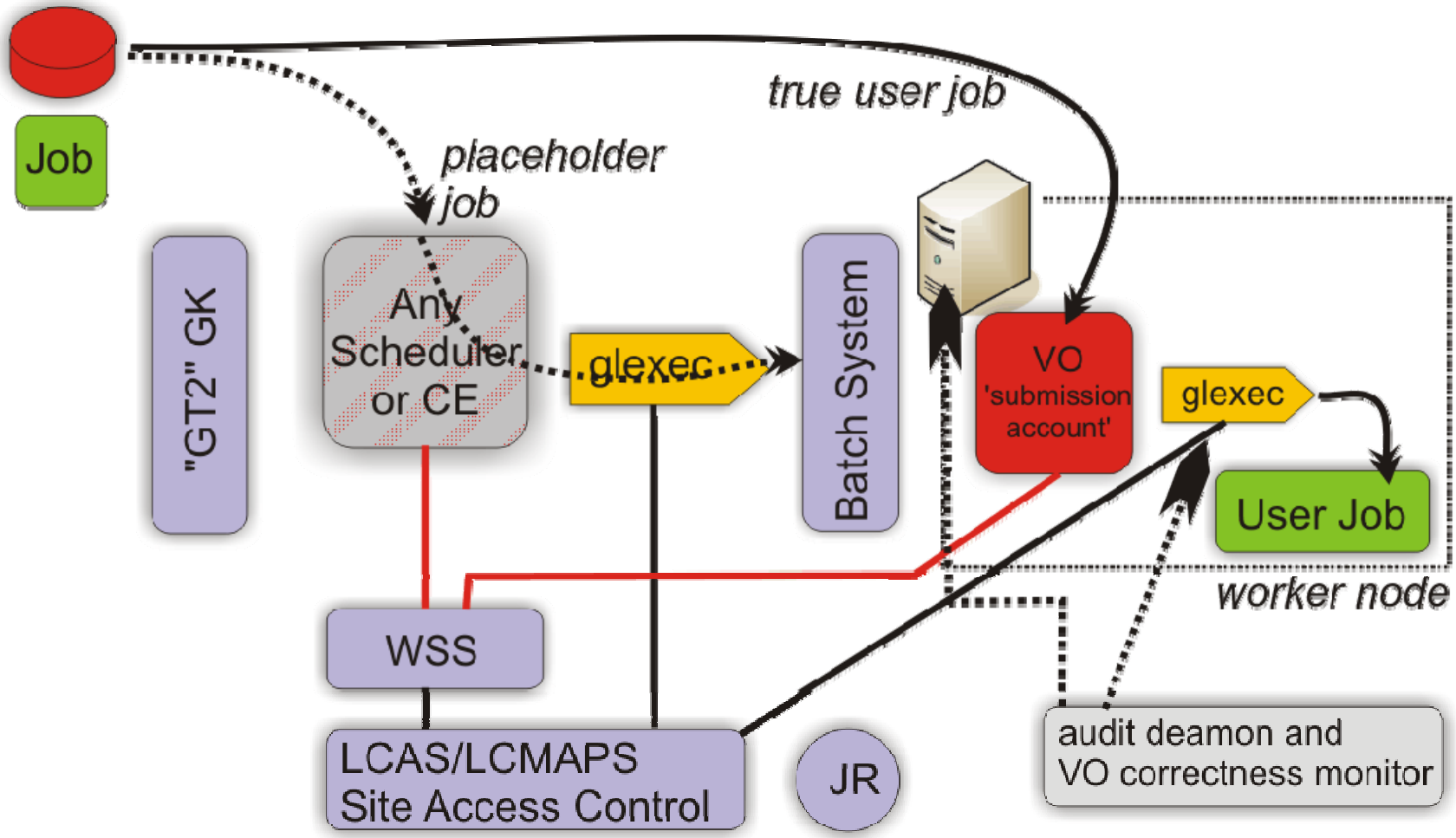
- Allow pilot jobs (VO happiness)
- Recover traceability (site happiness)
- Pilot jobs follow same policies and enforcement points as “normal” jobs (site happiness)
- Per-user and per-group accounting is VO problem (site and operations happiness)

pieces

- **Glexec on WN**
  - VO submits “CE” as grid job
  - CE starts to run as “VO” on WN
  - CE gets user jobs and credentials from VO task queue
  - CE runs job under user credentials via glexec
- **Central sitewide policy server (LCAS / LCMAPS)**
  - Allowed lists, ban lists, etc etc etc
- **Auditing on WN**
  - Checks that VO “CE” process is not using wall time
  - Checking that VO “CE” is not doing multiple glexecs
  - Allow discovery of id of suspect activity (footprints)

VO pilot job on the node

- **On success: the site will set the uid/gid of the new user's job**
- **On failure: glexec will return with an error, and pilot job can terminate or obtain other job**



Note: proper uid change by Gatekeeper or Condor-C/BLAHP on head node **should remain default**

What is needed in this model?

1. **Agreement on the three ingredients**
   - deployment of glexec on the WN to do setuid
   - detailed auditing on the head node and the WNs
   - site accounting done at the VO (i.e. pilot job) level
2. **glexec**
   - needs feature enhancements compared to single-CE version
   - *see status of glexec on the next slide*
3. **Inspection of the audit logs**
   - detect abuse patterns in the system-call auditing logs
4. **Grid job logging capabilities**
   - glexec will log (uid, user/system/real time usage) via syslog
   - credential mapping framework (LCMAPS) will log mapping (also via syslog)
   - centralisation of glexec mappings, e.g. via JobRepository

Status today

- **Status of 'glexec' today**
  - implementation ready & tested,
    based off the Apache HTTP suexec code base
  - uses the LCAS and LCMAPS for enforcement and mapping in their library-based implementation
  - new modules have been added
    - LCAS: RSL (executable path) constraints
    - validation of cert chain and proxy lifetime
  - restrictions
    - policy should be located on local POSIX-accessible file systems
    - policy transport should be 'trustworthy'
    - glexec executable by specific users only (today via Unix permissions)
- **Needed specifically for the –on-WN model**
  - make the credential acquisition process (LCAS/LCMAPS) work with a *site-central policy engine*
    - enforcement will have to stay local
  - changeover to standard callouts for both are needed
  - needs more site-sysadmin configuration capabilities

Needed components, procedures

- **Auditing the VO placeholder job/scheduler on the WN**
  - check number of ‘fork-execs’ done by the placeholder with the number of glexec invocations
    a discrepancy means the VO is cheating on you
  - check the VO placeholder job is not using too much CPU
    the CPU-time / Walltime should be close to zero

- **credential mapping auditing/logging**
  - ‘JobRepository’ fits the bill
    - *schema allows for recording and retrieving all aspects of credential mapping*
    - *records both user identity and any VO attributes*
    - *retains the credential mapping for each ‘job’ or glexec invocation*
  - JR is part of the stack, but not widely deployed yet

Notes and alternatives

- **glexec, like any site-managed ingress point, trusts the submitter not to have mixed up the user credentials and the jobs**
  - we trust the RB today do this correctly, and RBs are unknown quantities to the receiving site
- **a longer term solution is to have the job request singed by the submitting user**
  - since the description is modified by intermediaries (brokers), the signature can only be to the original content, and the site would have to evaluate whether the job received matches the signed JDL
  - or use an inheritance model for the job description, and treat the job like you would, e.g., a CIM entity

Summary

- Realize that today some VOs are doing ‘pilot’ jobs today
  - there is no effective enforcement against this
  - some sites may even just don’t care yet, whilst others have hard requirements on auditability and regulatory compliance

- The glexec-on-WN model gives the VOs tools to comply with site requirements
  - at least makes it ‘better’ than it is today
  - but you, as a site, will miss that warm and fuzzy feeling of trust

- a glexec-on-WN is always replaceable by the ‘null operation’ for sites that don’t care or want it
  - *but realize this is for just one of the glexec deployment models*