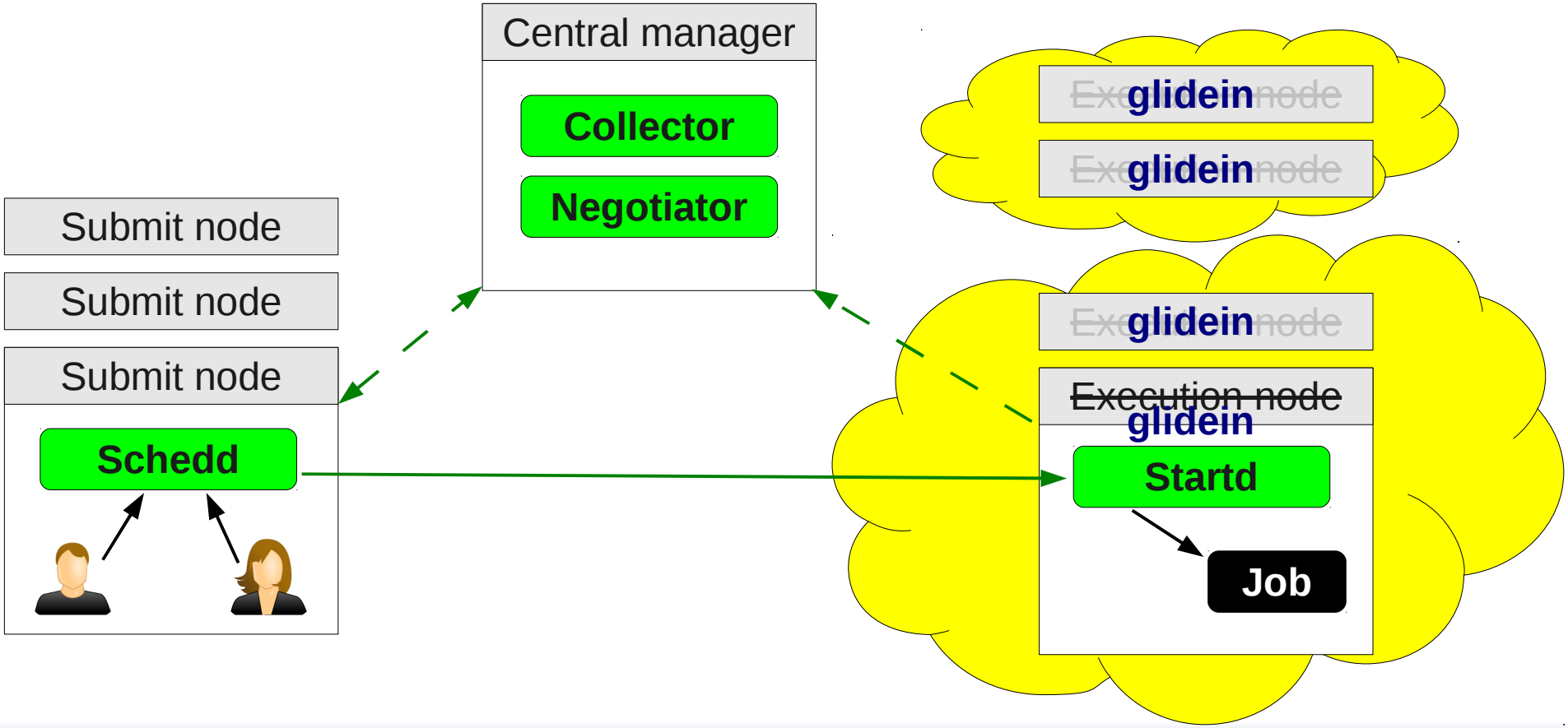
# Glidein startup Internals

by Igor Sfiligoi (UCSD)

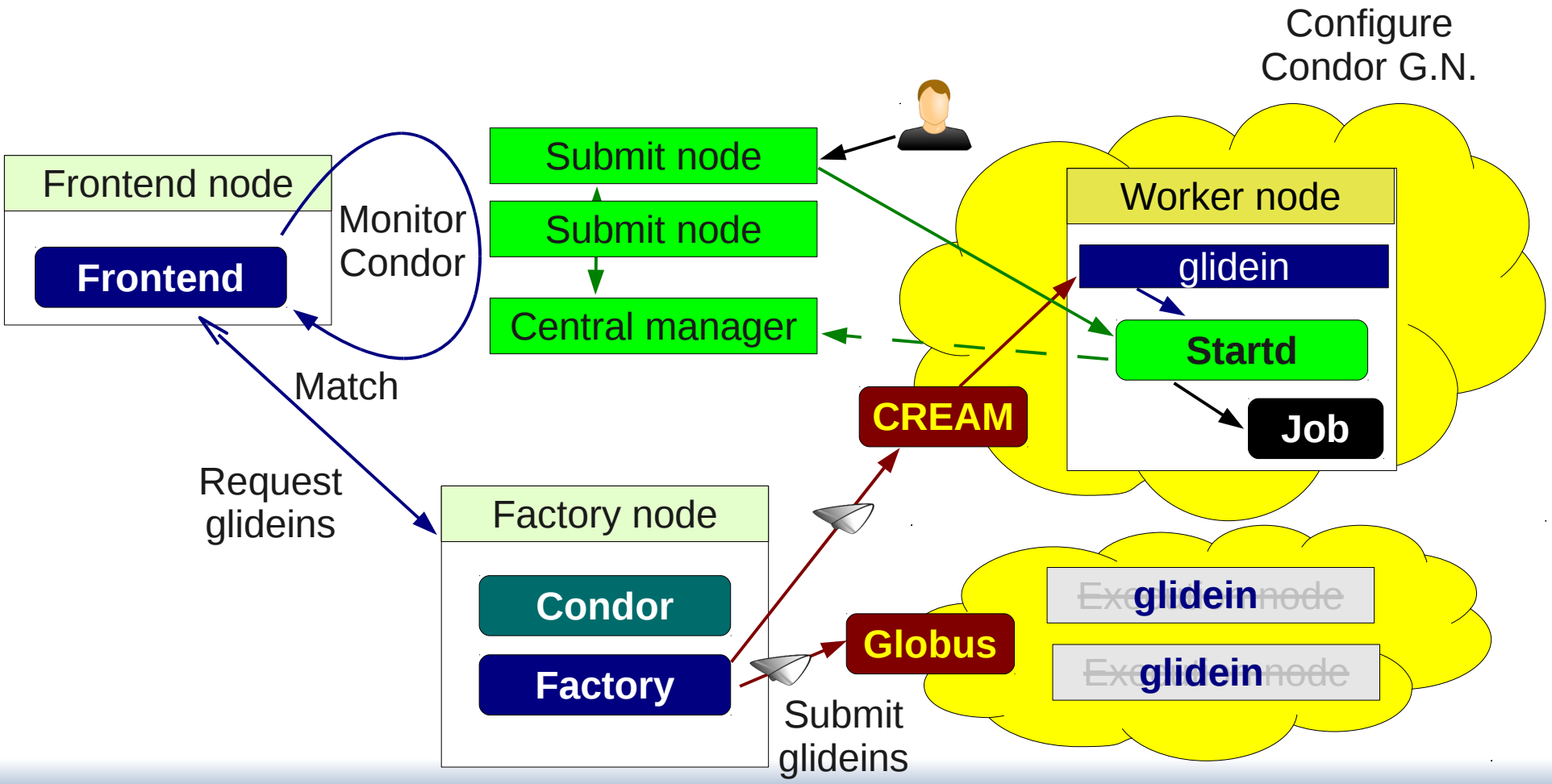
# Refresher - What is a glidein?

- A glidein is just a properly configured execution node submitted as a Grid job



# Refresher – Glidein startup

- glidein_startup configures and starts Condor

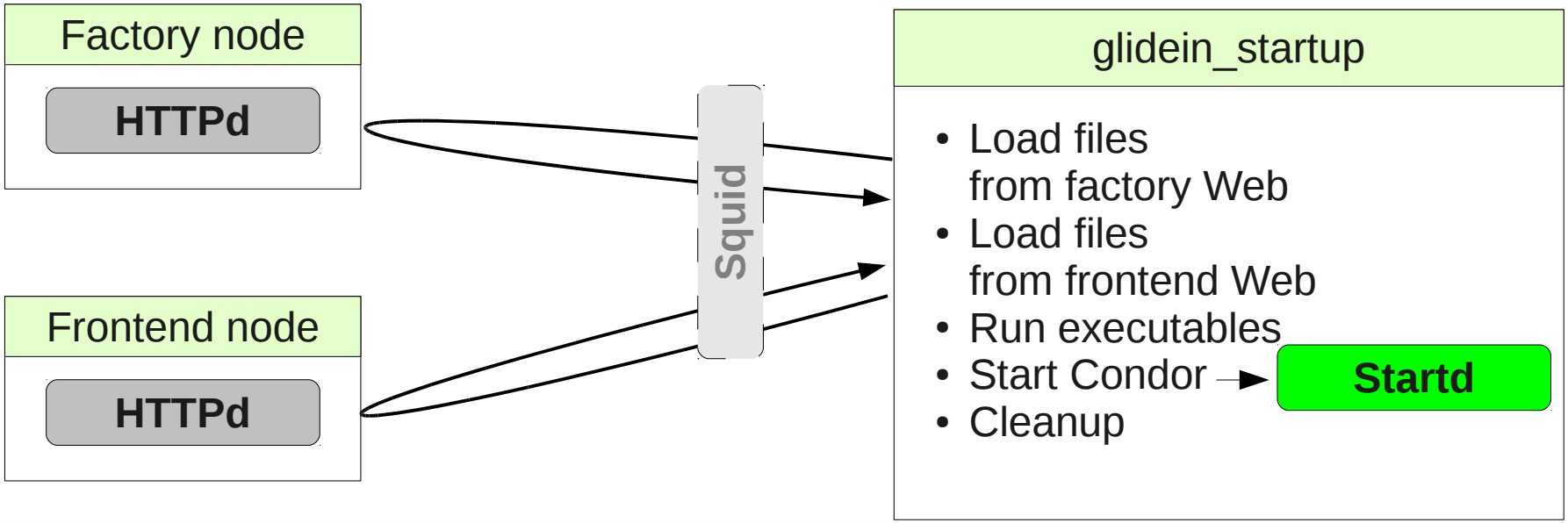


# glidein_startup tasks

- Download scripts, parameters and Condor bins
- Validate node (environment)
- Configure Condor
- Start Condor daemon(s)
- Collect post-mortem monitoring info
- Cleanup

# Downloading files

- Files downloaded via HTTP
  - From both the factory and the frontend Web servers
  - Can use local Web proxy (e.g. Squid)
  - Mechanism tamper proof and cache coherent



# URLs

- All URLs passed to glidein_startup as arguments
  - Factory Web server
  - Frontend Web server
  - Squid, if any

# Security measures

- Using SHA1 hashes
  - Each file has a SHA1 has associated with it
- The hashes delivered in a single file
  - List of (fname,hash) pairs
- The hash of the hash list is delivered as a parameter to glidein_startup
  - This is guaranteed to be secure with GRAM

# Cache coherence

- Files never change, once uploaded to the Web area
- If the source file changes, a file with a different name is created on the Web area
  - Essentially fname.date
- There is a logical to physical name map file
  - file_list.id.lst
  - The id of this list is passed as an argument to glidein_startup

# Node validation

- Run scripts / plugins provided by both factory and frontend
  - The list of files that are scripts vs “regular” files is separate, so obvious for glidein_startup
- If a script returns with exit_code !=0, glidein_startup stops execution
  - Condor never started → no user jobs ever pulled
  - Will sleep for 20 mins → blackhole protection

# Condor configuration

- Condor config values coming from files downloaded by glidein_startup
  - Static from both factory and frontend config files
- Scripts can add, alter or delete any attribute
  - Based on dynamic information
  - Either discovered on the WN
    or by combining info from various sources
- More details at
  http://tinyurl.com/glideinWMS/doc.prd/factory/custom_scripts.html

# Frontend provided config

- Collector host
- Trusted DNs (usually just the Collector)
- Start expression
- Potentially Rank, Preempt, …
- Matchmaking attributes

# Example Frontend attr

- START expression
  - As GLIDECLIENT_Group_Start
  - Automatically **&&** with Factory provided one

```
GLIDECLIENT_Group_Start =
  ((DESIRED_SEs=!=UNDEFINED) &&
   stringListMember(GLIDEIN_SEs,DESIRED_SEs)) ||
  ((DESIRED_Gatekeepers=!=UNDEFINED) &&
    stringListMember(GLIDEIN_Gatekeeper,
                     DESIRED_Gatekeepers)) ||
  ((DESIRED_Sites=!=UNDEFINED) &&
    stringListMember(GLIDEIN_Site,DESIRED_Sites))
```

# Example Frontend script

- CMSSW discovery (and validation)

```
#!/bin/sh
glidein_config="$1"
if [ -f "$VO_CMS_SW_DIR/cmsset_default.sh" ]; then
   source "$VO_CMS_SW_DIR/cmsset_default.sh"
else
   echo "cmsset_default.sh not found!\n" 1>&2
   exit 1
fi
tmpname=$PWD/installed_cms_software_tmp_$$.tmp
export SCRAM_ARCH=slc5_amd64_gcc434
scramv1 list -c CMSSW|grep CMSSW|awk '{print $2}'|sort|uniq|grep ^CMSSW > $tmpname
sw_list=`cat $tmpname | awk '{if (length(a)!=0) {a=a "," $0} else {a=$0}}END{print
a}'`
if [ -z "$sw_list" ]; then
  echo "No CMS SW found!" 1>&2
  exit 1
fi
echo "GLIDEIN_CMSSW_LIST $sw_list" >> "$glidein_config"
condor_vars_file=`grep -i "^CONDOR_VARS_FILE " $glidein_config | awk '{print $2}'`
echo "GLIDEIN_CMSSW_LIST S - + Y Y +" >> "$condor_vars_file"
```

http://tinyurl.com/glideinWMS/doc.prd/factory/custom_scripts.html

# Job execution

- After validation and configuration, glidein just start condor_master
  - Which in turn start condor_startd
- It is just regular Condor from here on
  - Any policy the VO needs must be part of Condor configuration

# Glidein lifetime

- Glideins are temporary resources
  - Must go away after some time
- We want them to go away by their own will
  - So we can monitor progress and clean up
- Condor daemons configured to die by themselves
  - Just need to tell them when
  - Mechanism changed between 2.5.2 and 2.5.3, but policies still the same

# Glidein lifetime policies

- Two termination triggers (Factory set)
  - Reach End-of-life – GLIDEIN_Max_Walltime
  - Unused for too long - GLIDEIN_Max_Idle
- Running jobs may be killed if EOL reached
  - Resulting in badput
- Reaching max_idle another waste
  - Especially when 0 jobs ran
  - Due to either no (more) jobs in VO queue or Frontend misconfiguration

# Minimizing badput

- Glideins must be told how long will jobs run
  - No standard way to extract this info in Condor
- Frontend should define typical expected jobs lifetime – GLIDEIN_Job_Max_Time
  - Glideins will not start any jobs not expected to finish
- Finer grained per-job matching policies can be defined as well
  - e.g. for jobs that are longer than “typical ones”

# Example START expression

- Have two different thresholds
  - One for first startup, one for all following
  - Useful when wide dynamic range, unpredictable

```
GLIDECLIENT_Start =
  ifthenelse(
    LastVacateTime=?=UNDEFINED,
    (NormMaxWallTimeMins*60) <
      (GLIDEIN_ToRetire+GLIDEIN_Job_Max_Time-MyCurrentTime),
    (MaxWallTimeMins*60) <
      (GLIDEIN_ToRetire+GLIDEIN_Job_Max_Time-MyCurrentTime)
  )
```

# Other sources of badput

- Glideins must stay within the limits of the resource lease
  - Typical limit is on memory usage
- If user jobs exceed that limit, the glideins may be killed by the resource provider (e.g. Grid batch system)
  - Resulting in the user job being killed
  - Thus badput

# Example START expression

- Prevent startup of jobs that are known to use too much memory
  - Known only at 2nd re-run
- Also kill jobs as soon as they pass that

```
GLIDECLIENT_Start =
  ImageSize<=(GLIDEIN_MaxMemMBs*1024)

PREEMPT =
  ImageSize>(GLIDEIN_MaxMemMBs*1024)
```

# Multi User Pilot Jobs

- A glidein typically starts with a proxy that is not representing a specific user
  - Definitely true for AnaOps
- Three potential security issues:
  - Site does not know who the final user is (and thus cannot ban specific users)
  - If 2 glideins land on the same node, 2 users may be running as the **same UID** (no system protection)
  - **User and pilot code** run as the same UID (no system protection)

# glexec

- We have one tool that can help us with all 3
  - Namely **glexec**
- glexec provides UID switching
  - But must be installed by the Site on WNs
  - Only a subset of sites have done it so far
- glexec requires a valid proxy from the final users
  - Or jobs will not match
  - Not a problem for CMS, but it may be for other VOs

# glexec, cont

- It is in VO best interest to use glexec
  - Only way to have a secure MUPJ system
  - Should push sites to deploy it
- Some sites require glexec for their own interest
  - To cryptographically know who the final users are
  - To easily ban users
- Note on site banning
  - Condor currently does not handle well glexec failures – VO admin must look for this

# Post mortem monitoring

- After a glidein terminates, the logs are sent back to the Factory
- Frontend does not have access to them
  - Work in progress for future glideinWMS release
  - For now, ask Factory admins if debugging

# Cleanup

- glidein_startup will remove all files before terminating
  - Unless killed by the OS, of course
  - Condor does something similar after every job
- Users should not expect anything to survive their jobs

# Pointers

- The official project Web page is http://tinyurl.com/glideinWMS
- glideinWMS development team is reachable at glideinwms-support@fnal.gov
- CMS AnaOps Frontend at UCSD
  http://glidein-collector.t2.ucsd.edu:8319/vofrontend/monitor/frontend_UCSD-v5_2/frontendStatus.html

# Acknowledgments


- The glideinWMS is a CMS-led project developed mostly at FNAL, with contributions from UCSD and ISI
- The glideinWMS factory operations at UCSD is sponsored by OSG
- The funding comes from NSF, DOE and the UC system