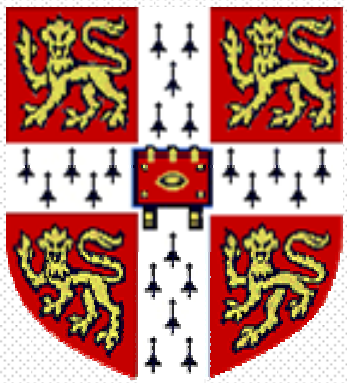
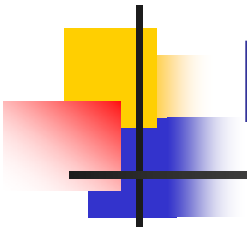
CamGrid: Experiences in constructing a university-wide, Condor-based, grid at the University of Cambridge

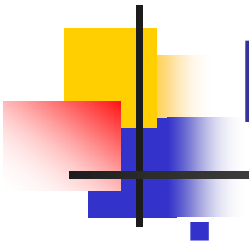


Mark Calleja
Bruce Beckles
University of Cambridge



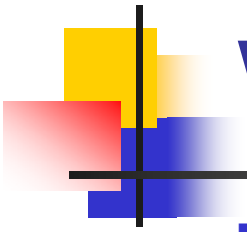
Project Aims

- To develop a University-wide “grid”:
  - To utilise as many CPUs as possible across the University for high throughput computing (HTC) jobs.
  - To utilise the University’s centrally-provided computing resources à la UCL Condor.
  - To allow flexible participation of resource owners in a safe, controlled manner, on their own terms (where this is feasible).



Project Overview

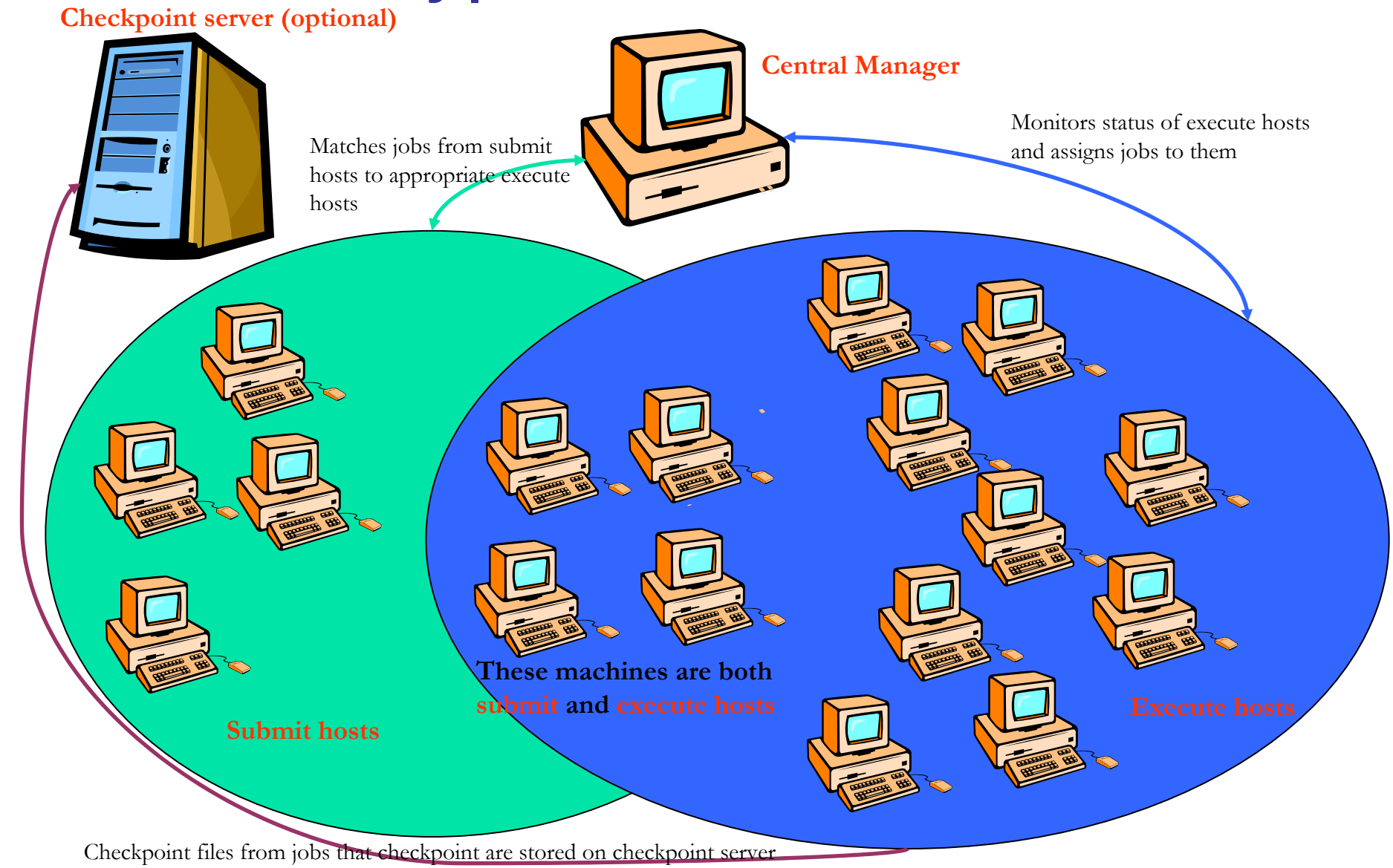
- Condor will be deployed across the University on as many CPUs as possible.
- Initially concentrating on the Linux platform, but eventually expanding to take in the Windows and MacOS X platforms. Also some IRIX and Solaris machines.
- Condor will provide the basic job management and 'micro-scheduling' required for CamGrid.
- More sophisticated meta-scheduling may be required and will be developed in-house if necessary.
- A choice of different deployment architectures will be offered to participating stakeholders.
- As far as possible, will seek not to impose any constraints on resource owners' management and configuration of their own resources.
- As far as possible, underlying divisions of the resources in CamGrid should be transparent to the end-user – users should not need to know which machines they are allowed to run jobs on, etc.



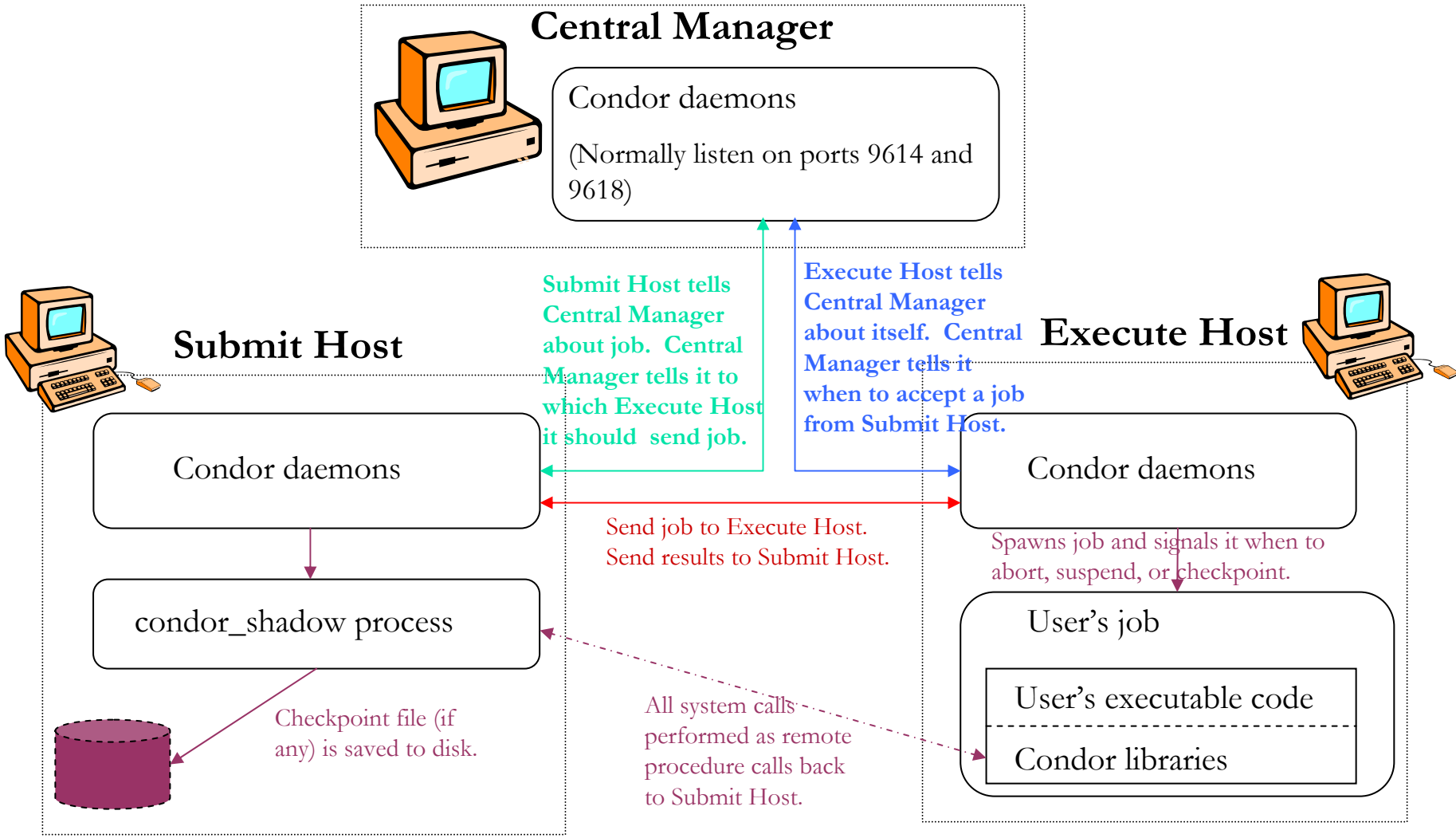
Why Condor?

- Proven itself capable of managing such large-scale deployments (UCL Condor).
- Flexible system which allows different deployment architectures.
- Gives the resource owner **complete control** over who can use their resources and the policies for resource use.
- Local experience with deploying and using Condor.
- Interfaces with the Globus Toolkit.
- Development team actively developing interfaces to other “grid toolkits”.
- Already used within the e-Science and grid communities, and increasingly within the wider scientific academic community.

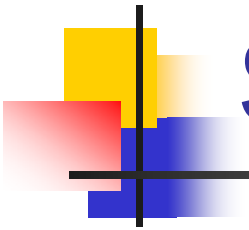
Condor Overview (1): A typical Condor Pool



Condor Overview (2): A typical Condor job

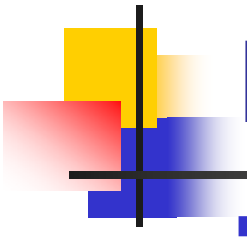


The University Computing Service (UCS)

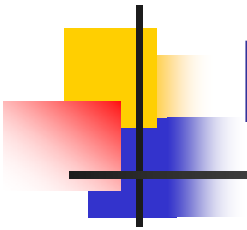


- Currently, the largest resource owner in CamGrid is the University Computing Service (UCS), which provides central computing facilities for the University *as a whole*.
- Particular requirements for participation in CamGrid:
  - Must interface smoothly with existing UCS systems and infrastructure.
  - Must comply with UCS security policies.
  - Must **not** make existing UCS systems any *less secure* or *less stable*.
- This gives rise to a particular requirement:
  - CamGrid 'middleware' on UCS systems must follow the principle of ***least privilege***: always run with the least security privilege necessary to achieve the current task.
- Condor, when run in its default configuration (the *condor_master* daemon running as `root`), violates this principle.

Two Deployment Environments



- The other current participants in CamGrid are mainly small research groups within Departments (though some of these can bring Departmental-wide resources into CamGrid).
- In general they have more relaxed security requirements regarding Condor daemons running on their individual resources.
- However, they have varying and complex requirements concerning who can use their resources, whose jobs get priority, etc.
- Thus it became clear that two different styles of Condor deployment would be necessary for CamGrid:
  - Environment 1: UCS owned and/or managed systems
  - Environment 2: Condor pools managed by anyone other than the UCS (research groups, etc.)



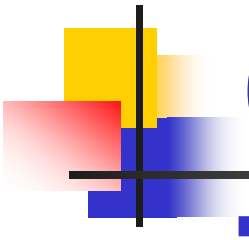
Environment 1

- Consists of two separate classes of machine:
  - Machines owned by the UCS and provided for the use of members of the University in ‘public’ computer rooms, teaching rooms, etc. – the Personal Workstation Facility (PWF).
  - Machines managed by the UCS on behalf and Departments and Colleges – the Managed Cluster Service (MCS).

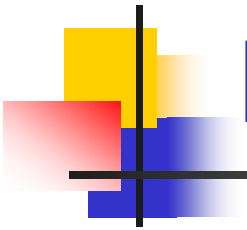
The Personal Workstation Facility (PWF)

- Centrally managed workstations set up to provide a consistent environment for end-users:
  - PCs running Windows XP
  - Dual-boot PCs running Windows XP/SuSE Linux 9.0
  - Apple Macintosh computer running MacOS X v10.3
- Not behind a firewall, public (‘globally visible’) IP addresses
- Of the order of 600 such workstations

The Managed Cluster Service (MCS)

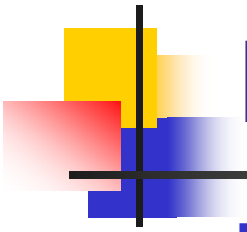


- A service offered by the UCS to Departments and Colleges of the University:
  - Institution buys equipment of a given specification
  - UCS installs workstations as a PWF in the institution, and remotely manages it on behalf of the institution in conjunction with the institution's local support staff
  - Institutions can have their MCS customised (installations of particular software applications, restrictions on who can use the machines, etc.) if they wish
- This service has proved extremely popular and has increased the number of 'PWF-style' workstations in the University significantly.
- Currently planning to deploy Condor as part of CamGrid (appropriately customised) to agreeable institutions who already participate in the MCS (at no additional cost).



Environment 1 architecture (1)

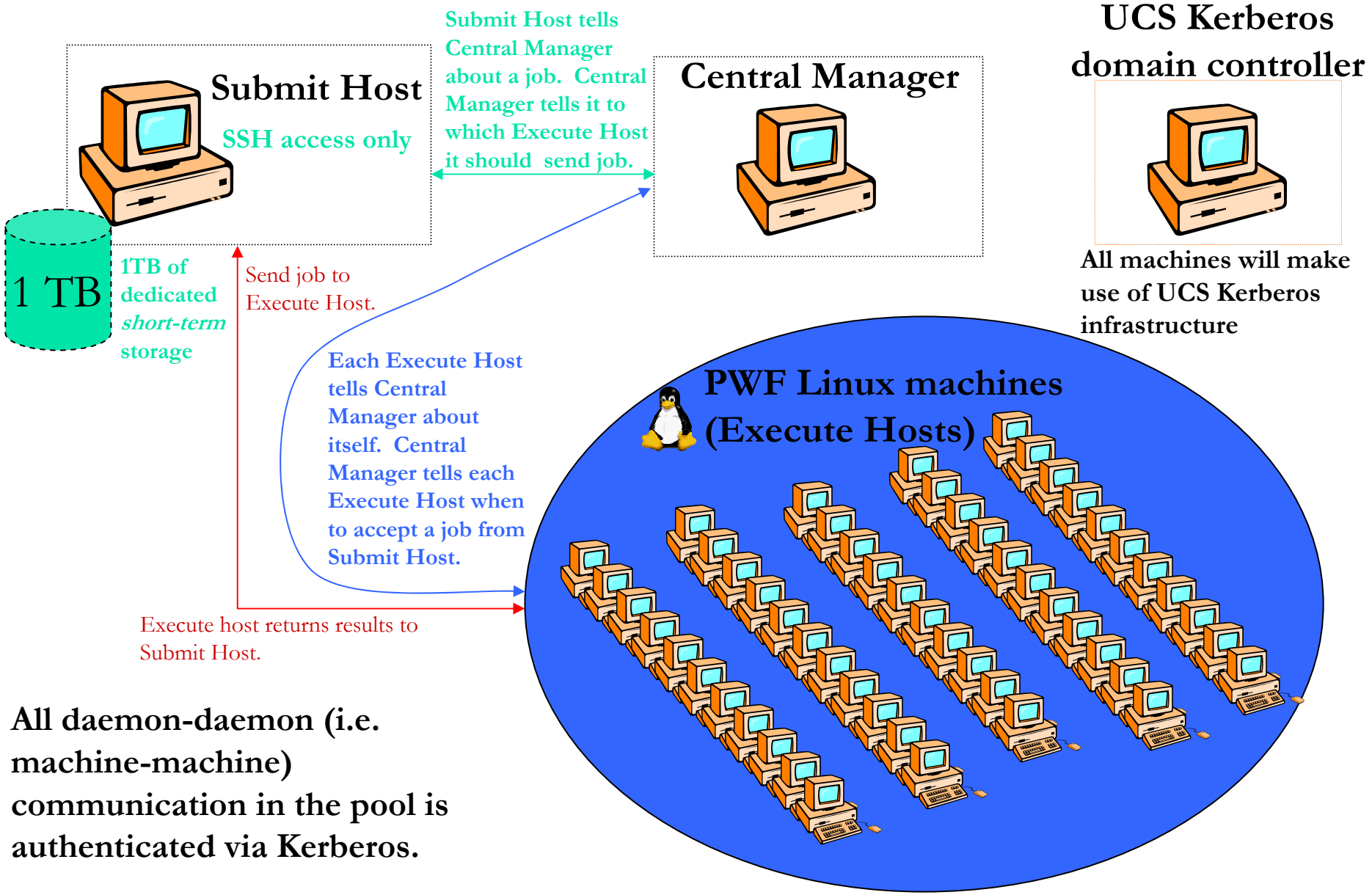
- Will consist of a mixture of UCS-owned PWF workstations and MCS workstations belonging to willing institutions.
- Authentication will be based on existing user authentication for the PWF. (PWF accounts are freely available to all current members of the University.)
- UCS-owned workstations will accept jobs from any current member of the University with a PWF account.
- MCS workstations belonging to other institutions will accept jobs according to the institution's policies (e.g. only jobs from members of the Department, priority to jobs from members of the Department, etc.)



Environment 1 architecture (2)

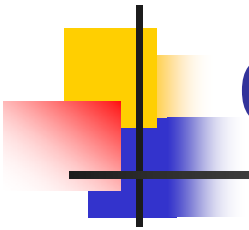
- Will use a dedicated Linux machine as the central manager.
- Initially only a single (Linux) submit host (more added as necessary to cope with more execute hosts) and 1TB of *short-term* storage will be provided. Access to this submit host will initially be via SSH only.
- All daemon-daemon communication will be authenticated via Kerberos.
- Initially only run jobs on PWF Linux workstations.
- No checkpoint server will be provided (maybe later if there is sufficient demand).

Environment 1 architecture (3)



All daemon-daemon (i.e. machine-machine) communication in the pool is authenticated via Kerberos.

Environment 1: Security considerations



- As provided, Condor cannot run securely under UNIX/Linux on submit and execute hosts unless it is started as `root`. (Condor on a dedicated central manager, as is being used here, does not require `root` privilege.)
- As previously noted, this is not acceptable in this environment.
- Solution is to not run Condor as `root` *and* provide secure mechanisms to enable it to make use of elevated privilege *where necessary*. How do we do this?

Least privilege: Execute host (1)

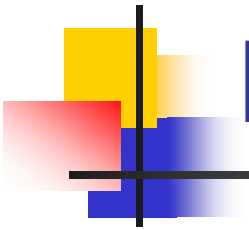
- On execute hosts, Condor makes use of two privileges of `root`:
  - Ability to switch to any user context (`CAP_SETUID` capability)
  - Ability to send signals to any process (`CAP_KILL` capability)

  Note that if you have the ability to switch to any user context you effectively have the ability to send signals to any process.
- Condor provides a facility (USER_JOB_WRAPPER) to pass the Condor job to a “wrapper” script, which is then responsible for executing the job.
- GNU userv allows one process to invoke another (potentially in a different user context) in a secure fashion when only limited trust exists between them (see http://www.chiark.greenend.org.uk/~ian/userv/).

Least privilege: Execute host (2)

- Putting these together we arrive at the following solution:
  - No Condor daemon or process runs as `root`
  - Condor passes job to our wrapper script
  - Our script installs a signal handler to intercept Condor's signals to the job
  - Our script `fork()`'s a userv process which runs the job in a different (also non-privileged) user context
  - On receipt of a signal from Condor, our script calls userv to send the signal to the job
  - userv uses pipes to pass the job's standard output and standard error back to our script (and so to Condor)
  - A cron job (using Condor's STARTD_CRON mechanism) runs once a minute to make sure that if there is no job executing then there are no processes running in our dedicated "Condor job" user context
  - **Voilà!**

Least privilege: Execute host (3)



- What does this gain us?
  - If there is a vulnerability in Condor then the entire machine is not compromised as a result.
  - We do not have any Condor processes, whose real user ID is `root`, listening to the network.
  - Much greater control over the job's execution environment – could run the job `chroot`'d if desired.
  - If a Condor job leaves behind any processes running after job completion they will be killed – normally, Condor only does this properly if specially configured (EXECUTE_LOGIN_IS_DEDICATED).

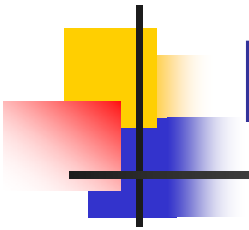
Least privilege: Execute host (4)

- What do we lose?
  - Can no longer suspend jobs (cannot catch SIGSTOP) – not a problem as we only want jobs to either run or be evicted.
  - We now need to handle passing the job environment variables, setting resource limits and scheduling priority, etc., which Condor would normally handle.
  - Condor can no longer correctly distinguish between the load Condor processes and Condor jobs are putting on the machine, and the load generated by other processes.
  - May be problems with jobs run in Condor's Standard universe (testing required).

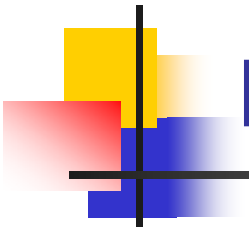
Least privilege: Submit host

- This has yet to be implemented, but, in principle it would work as follows:
  - No Condor daemon runs as `root`
  - Condor is told to call a “wrapper” script or program instead of *condor_shadow* when a job starts on an execute host (just change the filenames specified in the SHADOW and SHADOW_PVM macros).
  - “Wrapper” installs a signal handler and then `fork()`’s userv to call *condor_shadow* in the context of the user who submitted the job.
  - userv allows us to pass arbitrary file descriptors to the process we’ve invoked and we may need to use this to handle the network sockets normally inherited by *condor_shadow*.

Least privilege: A better way



- Under Linux, we use the dynamically linked version of Condor.
- So, using LD_PRELOAD we can intercept function calls Condor makes to signal processes (and also to change user context).
- With a specially written library we could then catch such calls and handle them via userv, thus transparently allowing Condor to function as intended.
- *Maybe...!?* (Intend to investigate such a solution as a long term strategy of the UCS deployment of Condor.)



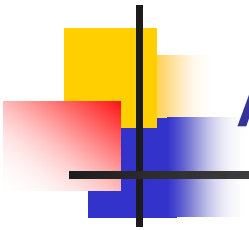
Environment 2

Aims to federate any willing Condor pools across the university into one flock.

Must allow for many complications, e.g:

1) Department firewalls
2) “Private” IP addresses, local to departments/colleges.
3) Diverse stakeholder needs/requirements/worries.
4) Licensing issues: “Can I pre-stage an executable on someone else’s machine?”

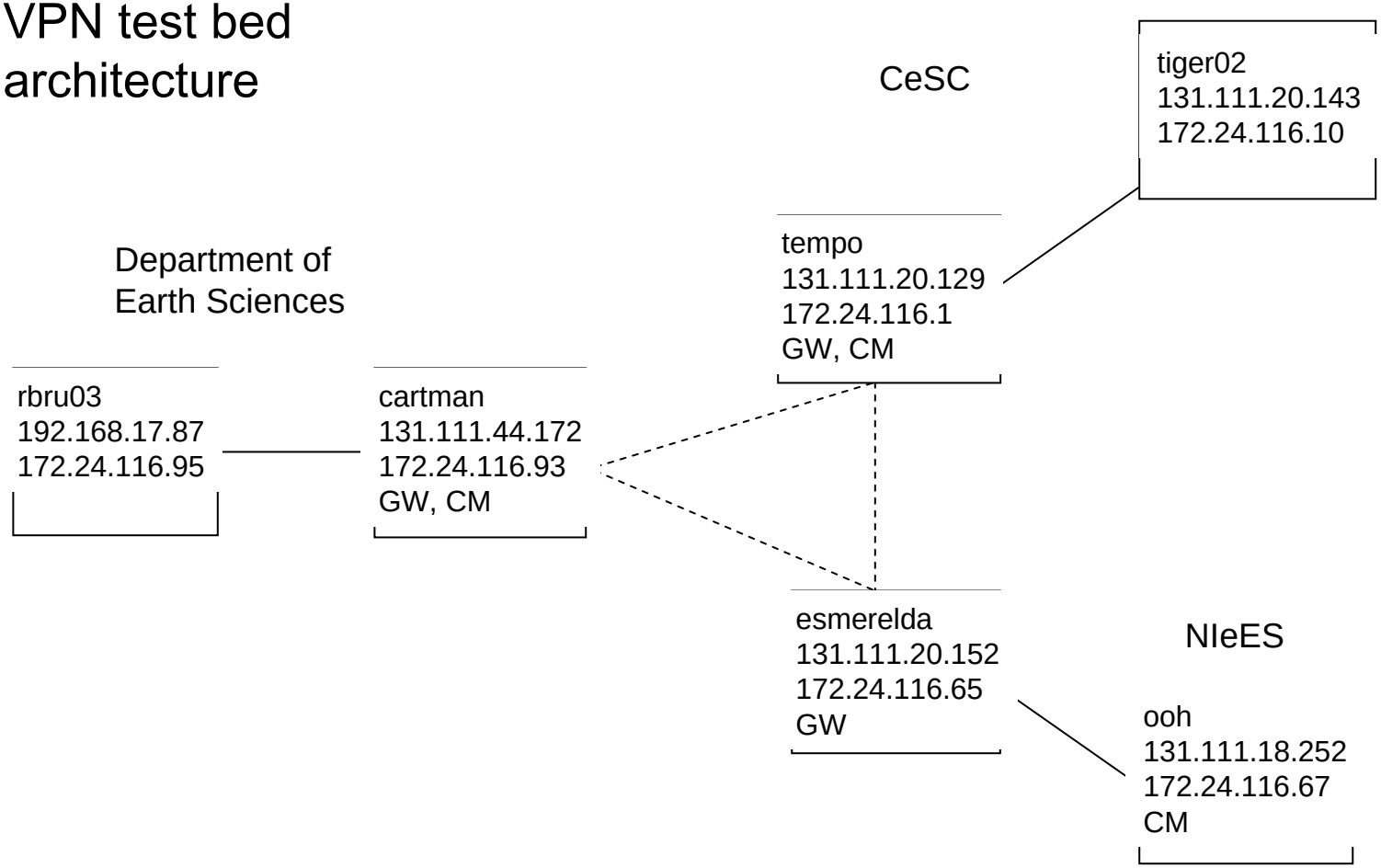
Initially we’ve started with just a small number of departments, and have considered two different approaches to solving 1) and 2): one that uses a VPN and another that utilises university-wide “private” IP addresses.

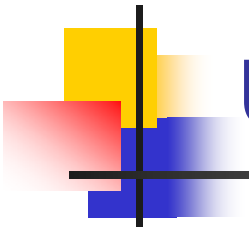


A VPN approach

- Uses secnet: http://www.chiark.greenend.org.uk/~secnet
- Each departmental pool configures a VPN gateway, and all other machines in the pool are given a second IP address.
- All Condor-specific traffic between pools goes via these gateways, so requires minimal firewall modifications.
- Also means that only the gateway needs a “globally” visible IP address.
- Inter-gateway traffic is encrypted.
- Initial tests with three small pools worked well, though hasn’t been stress tested.
- However, there are unanswered security issues since we’re effectively bypassing any resident firewall.

Environment 2
VPN test bed architecture





University-wide IP addresses

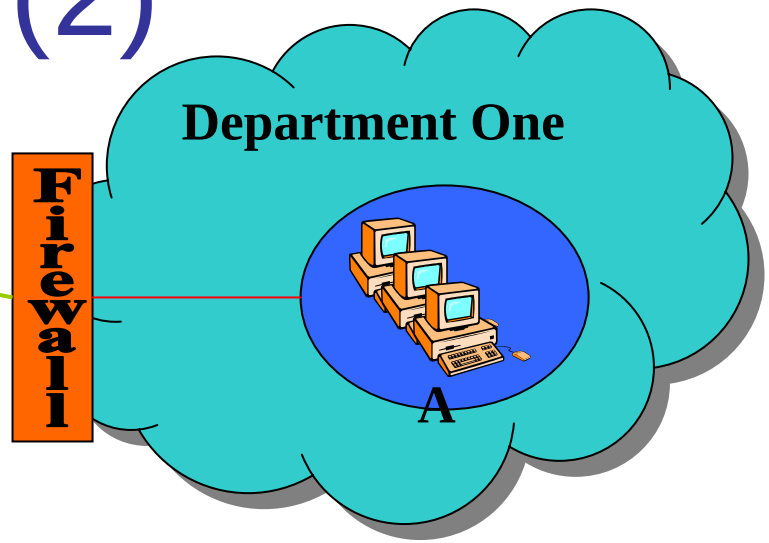
- Give each department a dedicated subnet on a range of university-routable addresses
- Each machine in a department's Condor pool is given one of these addresses in addition to its original one (á la VPN).
- Condor traffic is now routed via conventional gateway.
- Department firewalls must be configured to allow through any external traffic arising from flocking activities.
- We've only just started testing this model: involves more work for our Computer Officer but he can sleep easier.
- Performance appears comparable to the VPN approach, but has involved more work in setting up.

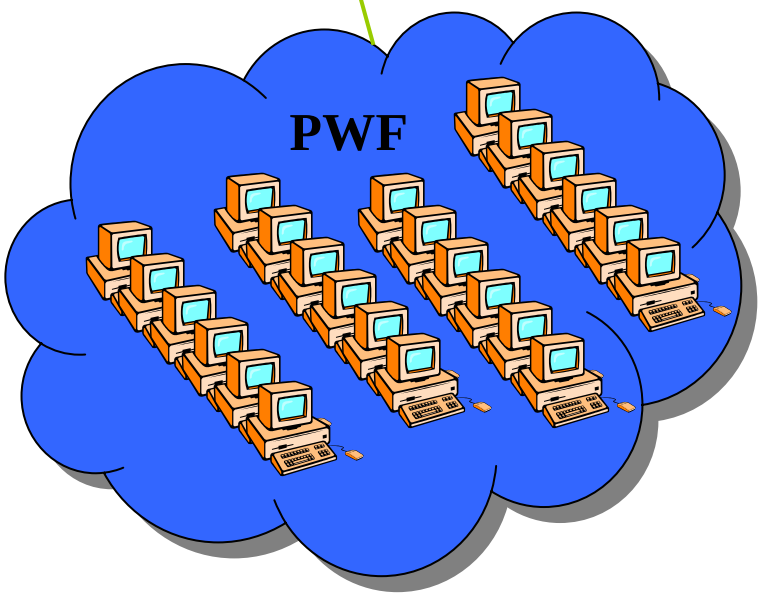
A Proposal for a Federated CamGrid (1)

- It is still not yet clear how Environment 1 and Environment 2 will interact, and a number of possibilities are under active investigation.
- One proposal is to make use of the UCS provided submit node(s) as a central submission point for CamGrid:
  - Makes firewall administration significantly easier (à la VPN).
  - Makes the underlying divisions between groups of resources (e.g. those in Environment 1 and those in Environment 2) transparent to end-users.
  - Makes it easy for resource owners who do not wish to set up their own Condor pools to join CamGrid, whilst still supporting those with their own Condor pools.
  - Preserves autonomy of resource owners, who can still have separate arrangements between their Condor pools if desired (e.g. for cross-Departmental research projects).
  - As under Condor the policies to determine what jobs can run on a resource are configured and enforced on that resource itself, resource owners still maintain complete control of their resources.

A Proposal for a Federated CamGrid (2)

Departmental Firewall only needs to allow traffic from one host through, regardless of the number of machines participating in CamGrid, and it can apply some of its rules to filter traffic from that host if desired.





Machines in **A** and **C** must have public IP addresses or University-wide private IP addresses. PWF machines have public IP addresses.