Running **real jobs** in *virtual machines*

Andrew McNab
University of Manchester

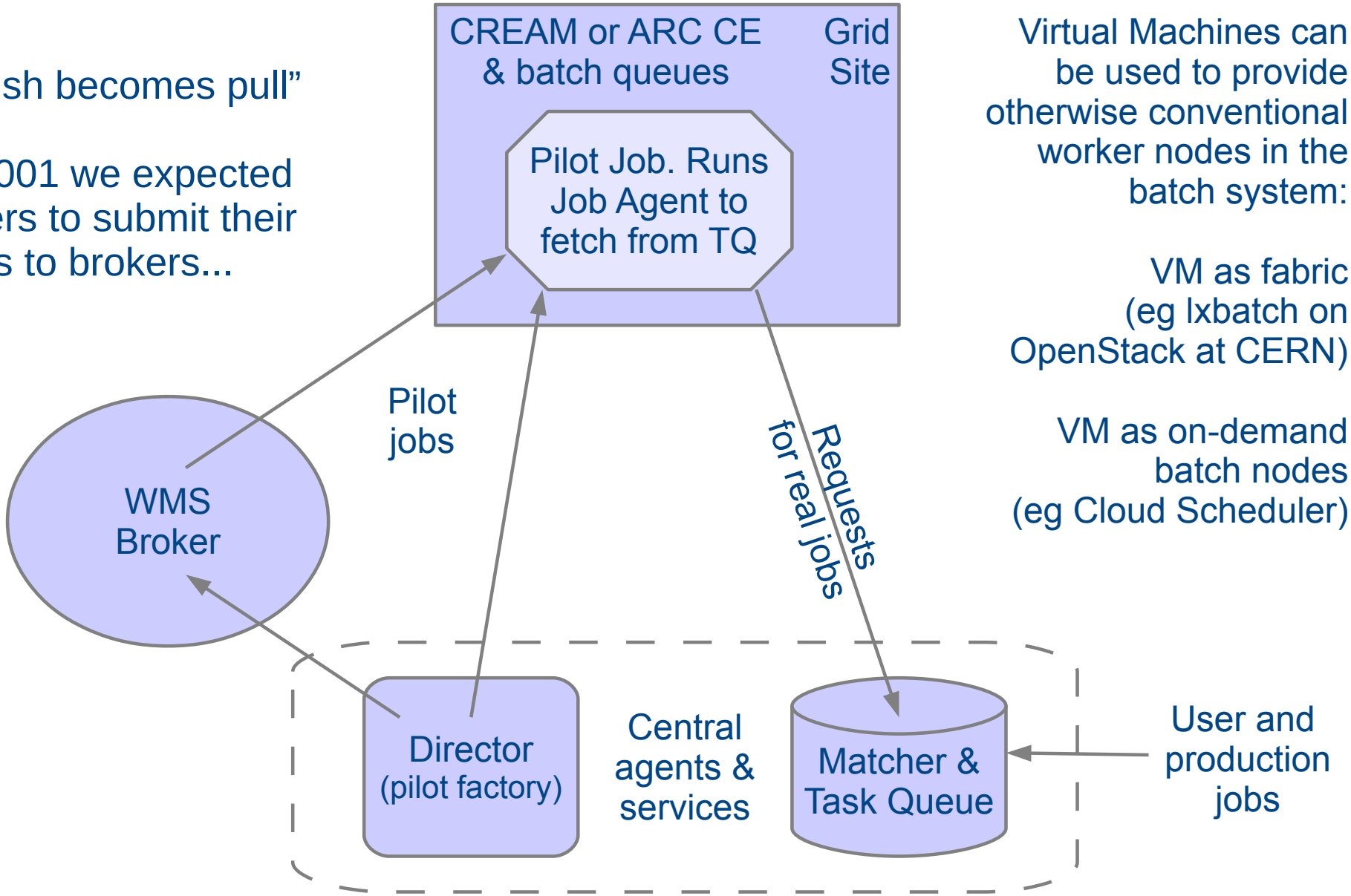
Overview

- The Grid we have now
- Strategies for starting VMs
- Fabric / Cloud / Vac
- Hypervisors
- Disk images
- Contextualization
- Pilot frameworks
- LHCb / GridPP / ATLAS VMs
- Accounting
- Monitoring

The Grid

"Push becomes pull"

c.2001 we expected users to submit their jobs to brokers...



Virtual Machines can be used to provide otherwise conventional worker nodes in the batch system:

VM as fabric (eg lxbatch on OpenStack at CERN)

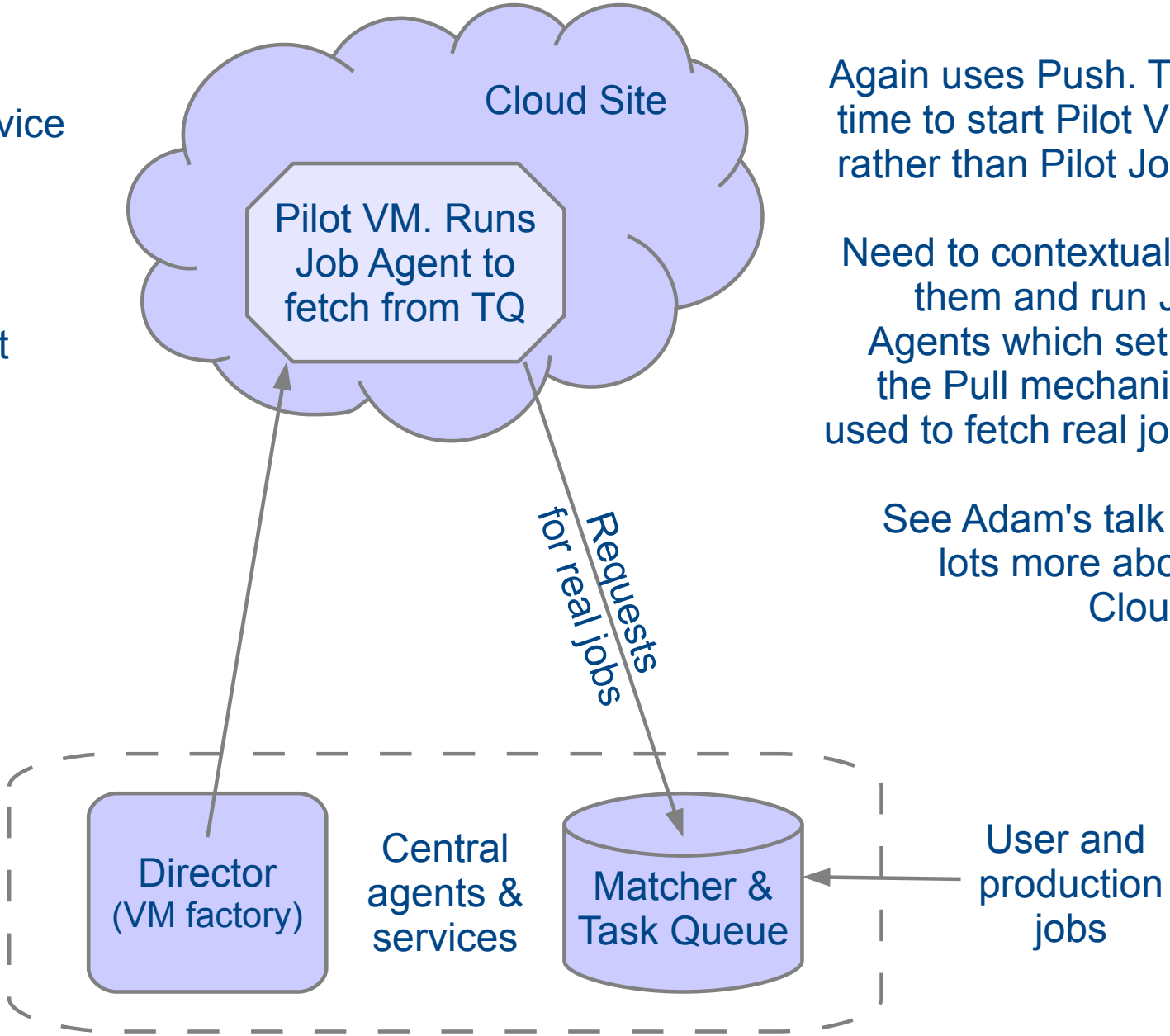
VM as on-demand batch nodes (eg Cloud Scheduler)

The Cloud

Infrastructure-as-a-Service (IaaS)

Expose the Cloud infrastructure to the experiments, and let them create VMs directly.

Can typically use the same Task Queues as for Grid sites.



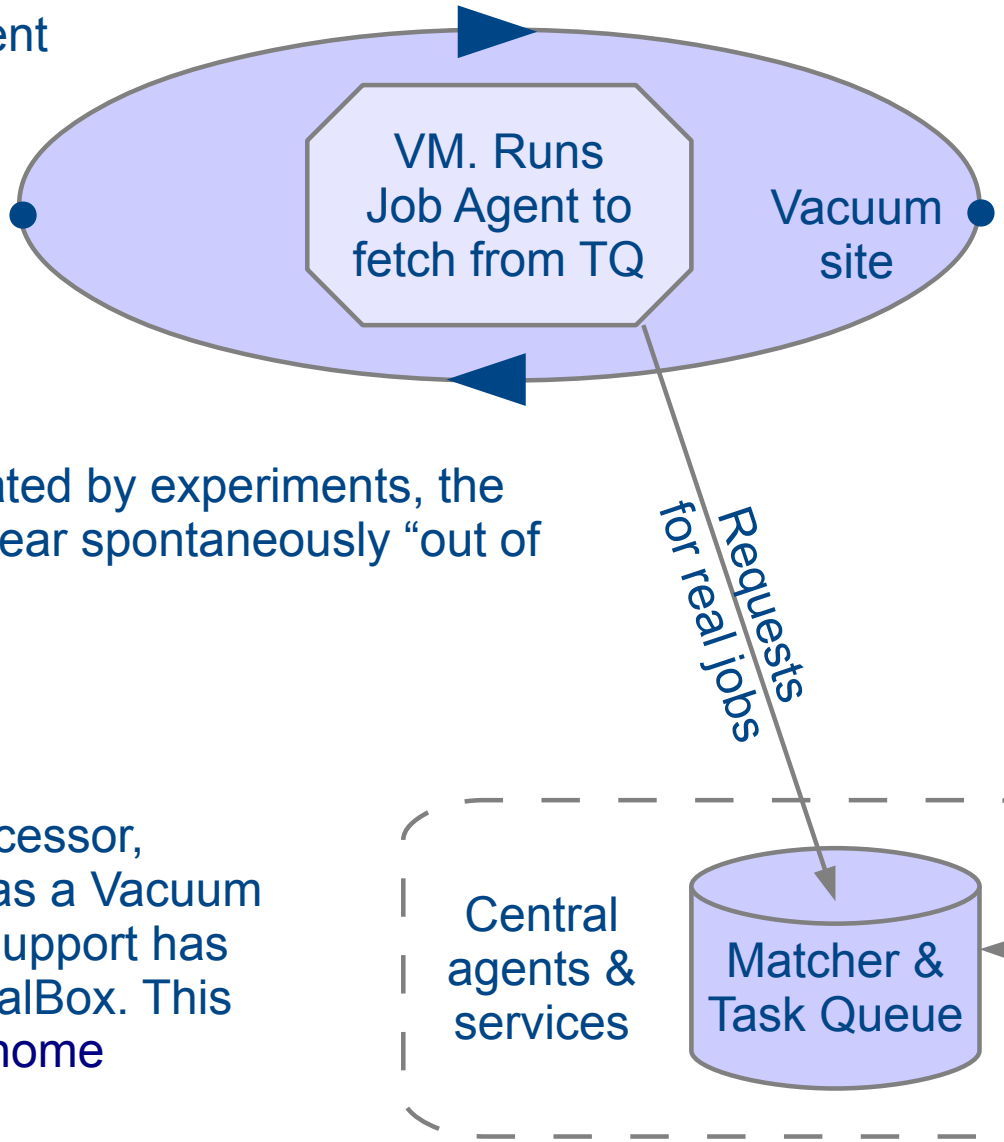
Again uses Push. This time to start Pilot VMs rather than Pilot Jobs.

Need to contextualize them and run Job Agents which set up the Pull mechanism used to fetch real jobs.

See Adam's talk for lots more about Clouds.

“The Vacuum”

Infrastructure-as-a-Client
(IaaC)



With the Vac implementation, host machines can run VMs for particular experiments depending on work available and target shares for each experiment.

Instead of being created by experiments, the virtual machines appear spontaneously “out of the vacuum” at sites.

The SETI@home successor, BOINC, can be seen as a Vacuum system now that VM support has been added with VirtualBox. This is the basis of LHC@home projects.

Vacuum Model

- For the experiments, VMs appear by “spontaneous production in the vacuum”
  - Like virtual particles in the physical vacuum: they appear, potentially interact, and then disappear
- From the CHEP 2013 paper:
  - *“The Vacuum model can be defined as a scenario in which virtual machines are created and contextualized for experiments by the site itself. The contextualization procedures are supplied in advance by the experiments and launch clients within the virtual machines to obtain work from the experiments' central queue of tasks.”*
- At many sites, 90% of the work is done by 2 or 3 experiments
  - This justifies effort to set them up at the site (comparable to site-info.def etc)
  - In return, the site is no longer dependent on all the CREAM/batch or Cloud machinery for these jobs

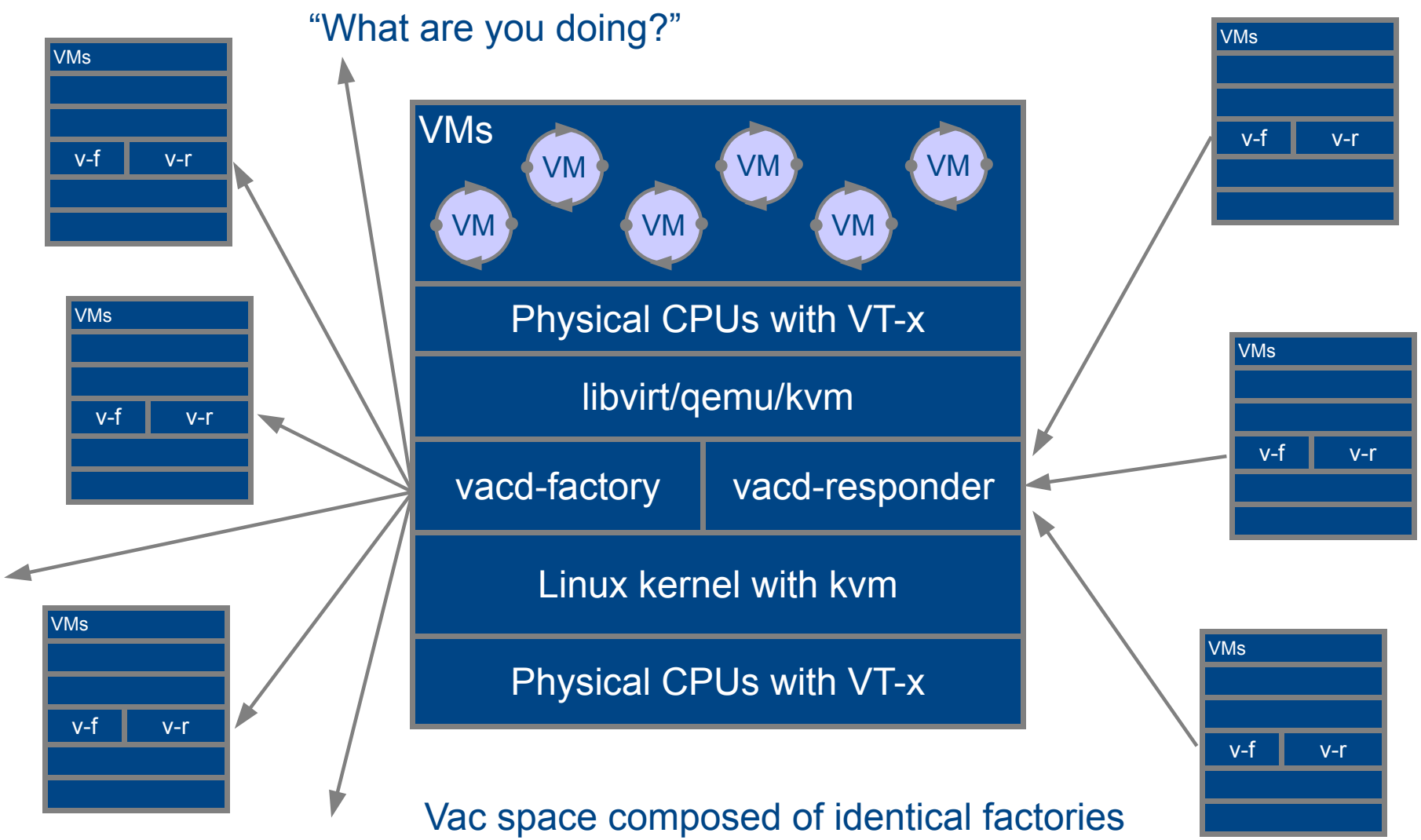
Vac implementation

- On each physical node, Vac VM factory daemon runs to create and apply contextualization to transient VMs
- Multiple VM flavours (“VM types”) are supported, ~1 per experiment
- Each site or Vac “space” is composed of autonomous factory nodes
  - All using the same /etc/vac.conf, /etc/vac-targetshares.conf etc
- Factories communicate with each other via UDP
  - Type of VM to start in a free slot based on what else is running and target shares
  - So no headnode central point of failure; robust against losing individual nodes
- Supports CernVM 2 (~SL5) and now CernVM 3 (~SL6 in Vac 0.15.0)
- In production with LHCb since last year.
- This month: successfully running ATLAS production jobs; test jobs working with GridPP DIRAC.

Hypervisors

- Hypervisors manage virtual machines
  - Either an operating system component that does this
  - Or the physical machine that hosts VMs (a “factory” in Vac terminology)
- General view is that KVM is the best way to do this on Linux
  - Kernel support for hardware virtualization extensions
  - Benefits from Kernel Samepage Merging (KSM) to save on physical memory
- Other hypervisors are relevant to interactive environments but not really applicable here
  - VirtualBox and VMware
  - May need to get the right disk image for a particular hypervisor
- Typically have libraries (qemu, libvirt) and/or daemons (libvirtd) which keep track of VMs and their devices (names, disks, IO, network etc)
  - So virtual machines can look like virtual computers

Hypervisors etc in Vac factory machines



Disk images

- Simplest method is to have a file or partition on the hypervisor machine which is a disk image
  - ATLAS Cloud Scheduler work, for instance, using an SL6 snapshot
  - CMS used SL5 and now SL6 snapshots
  - CernVM 2 does this too, with a large (9GB) but sparse (still 1.4GB) disk image containing a snapshot of an OS very similar to SLC5
- New CernVM 3 uses cvmfs to require a much smaller image
  - Boots off an ISO image (12MB), which uses cvmfs to provide a root partition populated via the network and reboots off that
  - Clever bit is use of aufs to provide copy-on-write facility
  - Have a completely sparse 20GB backing file, that receives anything the VM writes to the root partition. Everything else still from cvmfs.
  - OS is basically SLC6; supports RPM/YUM for extra packages at runtime
- Probably want a faster logical partition too, mounted as /scratch?

Contextualization

- Could just put everything needed into the image
  - But more flexible to supply some information at runtime (eg site name)
  - cvmfs makes it easy to create generic images with experiment specific files in /cvmfs/atlas.cern.ch etc.
  - So with LHCbVMDIRAC contexualization, we use a ~40 line site-specific script written by the user_data_generator command and everything else comes from cvmfs + generic CernVM image.
- All of the VM platforms provide some way of supplying contextualization like this
- Getting a particular experiment to run in a VM usually amounts to coming up with a suitable contextualization
  - Needs to capture any specific needs of the experiment
  - Plus generic grid software (eg from /cvmfs/grid.cern.ch)
- **But the pilot frameworks have done almost all this work already**

Contextualization: pilot framework

- The last step of the contextualization process is to run the pilot framework client
  - DIRAC for LHCb, ILC, Biomed, GridPP etc
  - PANDA for ATLAS
  - Condor glideinWMS for CMS
- For VMs, run the same scripts that conventional pilots do
  - JobAgent for DIRAC
  - Peter Love's runpilot3-wrapper-oct02.sh for PANDA
- Payload job downloaded and then run as normal
  - Work gets done
- Ideally experiment's monitoring will work without modification
  - The more it relies on the payload job rather than the pilot the easier this is

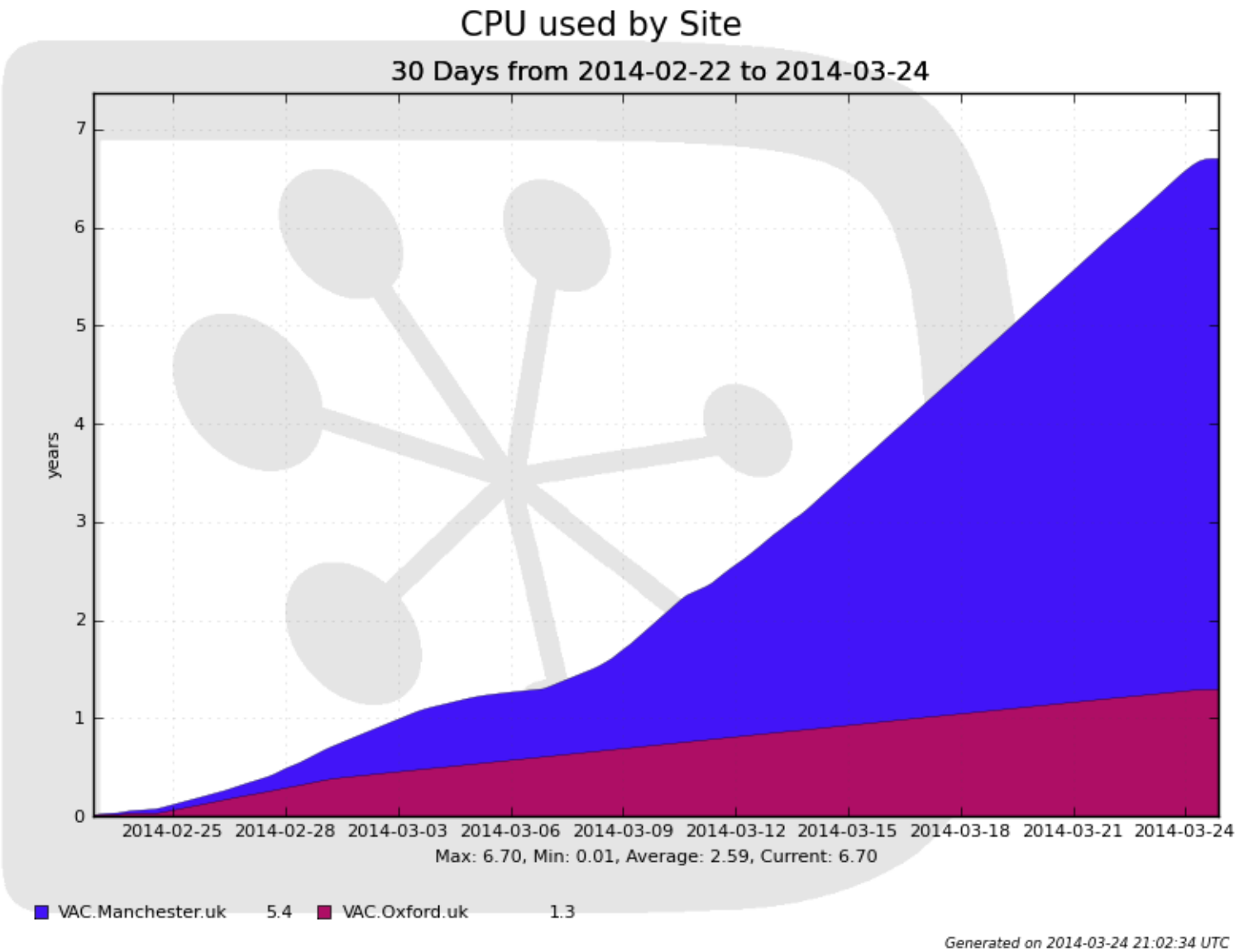
Contextualization: LHCb

- Developed during 2013 by Mario Ubeda Garcia, with contributions from me
- Works for Clouds, Vac, and BOINC
- Runs DIRAC JobAgent managed by runit for long-lived Cloud VMs
  - With Vac, just run once to get one payload job
- All the scripts are fetched from /cvmfs/lhcb.cern.ch
- The site (Vac) or VM manager (Cloud, BOINC) just creates user_data using user_data_generator command from SVN:

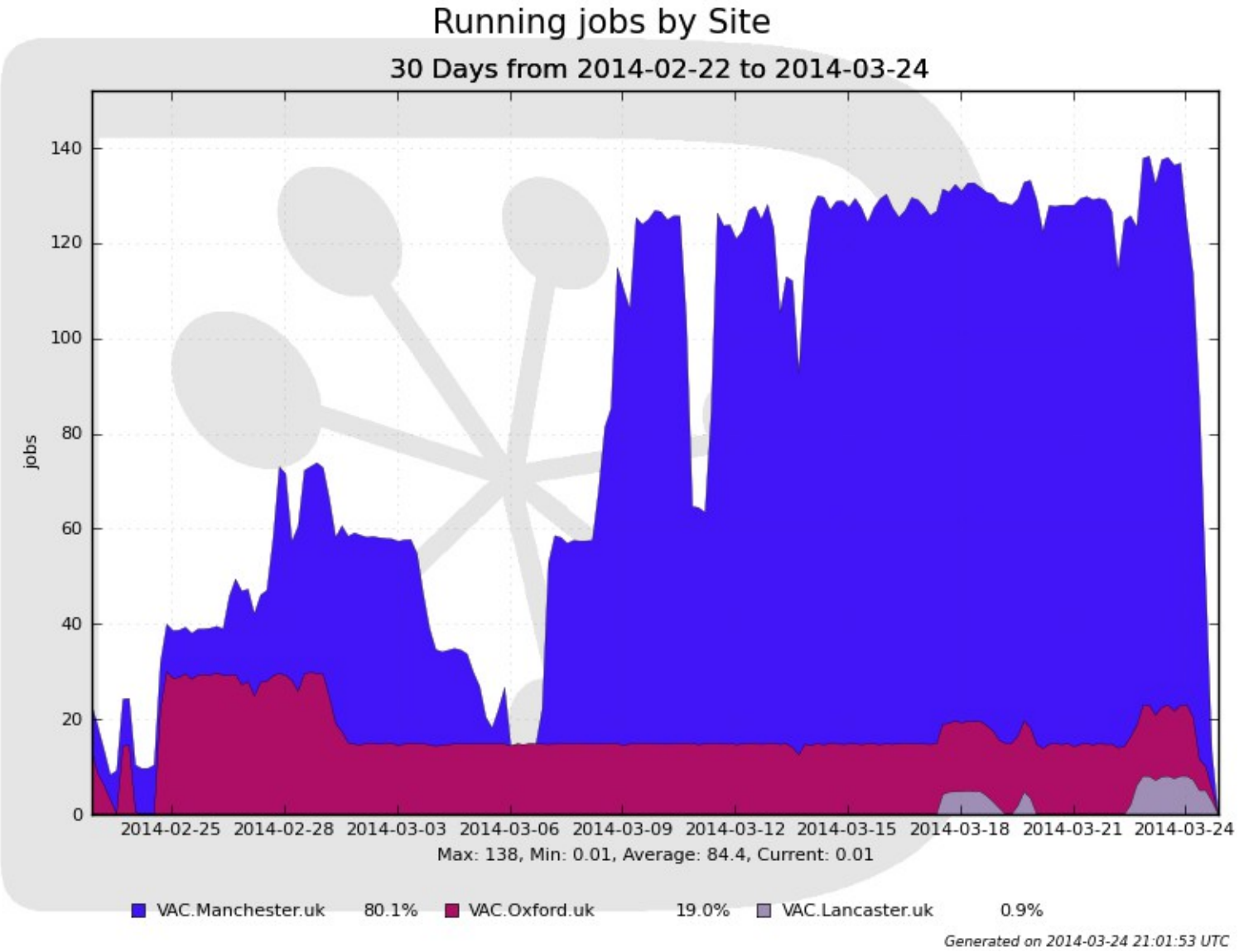
```
/user_data_generator --mode vac --hostcert ~/lhcb-vm.example.cc.cert.pem \
  --hostkey ~/lhcb-vm.example.cc.key.pem --site VAC.Example.cc \
  --gridce vac01.example.cc --cvmfsproxy http://squid-cache.example.cc:3128
```

- Introduced role separation using sudo (see later)

LHCb jobs in VMs on Vac



LHCb jobs in VMs on Vac



Contextualization: GridPP DIRAC

- GridPP DIRAC service intended to support smaller experiments
  - Working for conventional CREAM/batch sites
- VM contextualization developed by me during March 2013
  - Just for Vac at the moment, but could be extended to Clouds
- Sites use a make_user_data command to create a custom user_data contextualization file
  - VM gets scripts from SVN instead of cvmfs, and runs bootstrap
  - Bootstrap does root-level configuration of /scratch, cvmfs etc and runs vm-pilot
- vm-pilot runs DIRAC JobAgent as dirac unix user
  - DIRAC glexec-mode is used to run payload job as diracuser
- This contextualization could be used with little modification by any DIRAC-based experiment or VO
  - ILC, Biomed, EGI DIRAC, ...

Contextualization: ATLAS

- Contextualizations already exist
  - See CHEP 2013 talks for outlines
- I've developed a simple method based on the two DIRAC ones
  - Was surprisingly easy to do
  - All the real work is done by runpilot3-wrapper-oct02.sh and PANDA
- Site uses make_user_data and again bootstrap and vm-pilot scripts come from SVN
  - Everything else from cvmfs as normal
- Have successfully run a batch of ATLAS jobs on Vac VMs
- Still need to test on the larger 96 VM slot Vac space
- Required release of CernVM 3 to get SLC6 environment
  - So also requires next Vac release (0.15.0)

Role separation

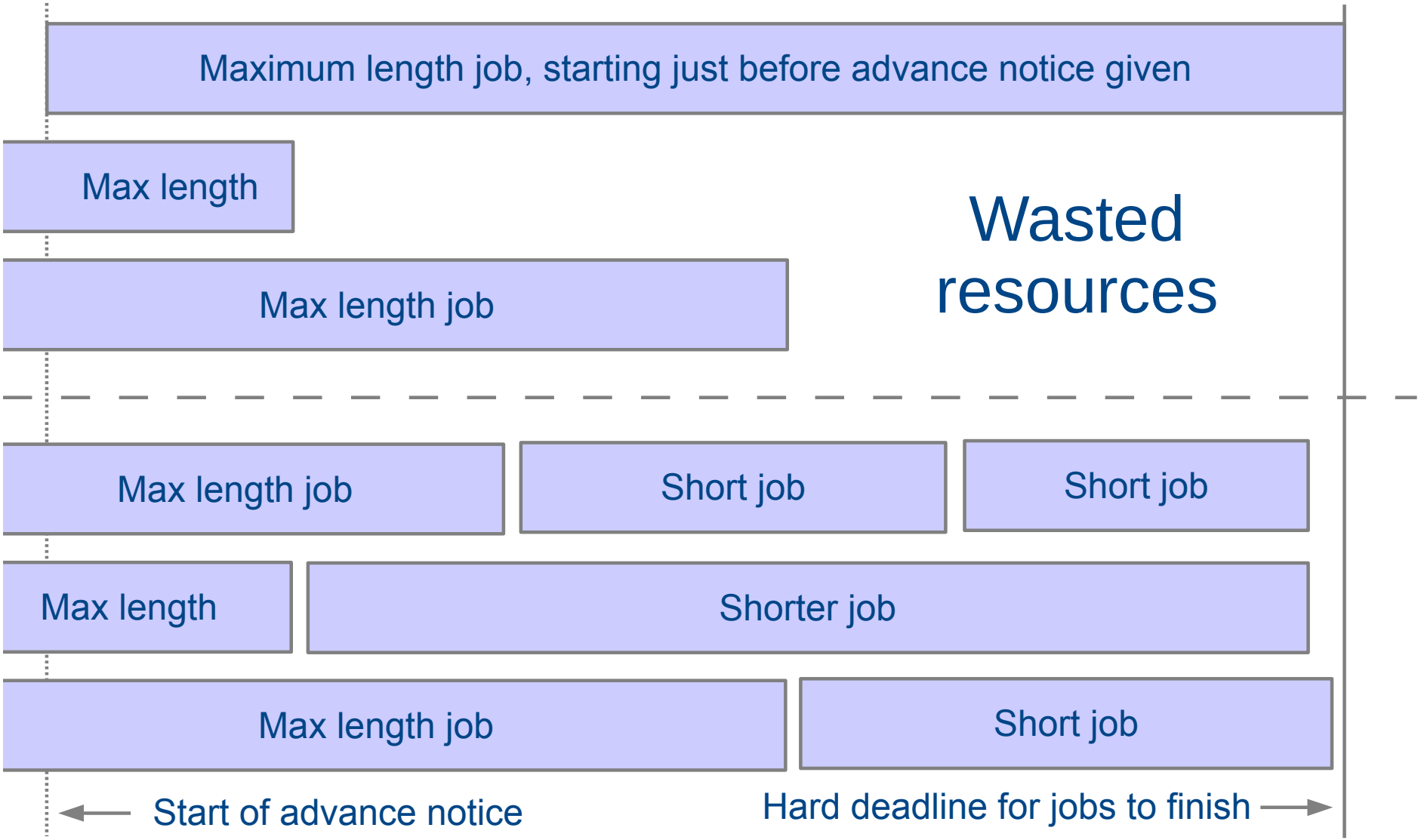
- Typically have 3 roles in a VM
  - root which configures the operating system, cvmfs, mounts partitions, ...
  - the pilot client which has credentials for accessing the pilot framework
  - the payload jobs themselves which potentially run scripts from random users
- Natural to use Unix user accounts to separate these roles
  - glexec aims to do this on conventional CE/Batch worker nodes
  - CMS are using glexec in CERN cloud VMs
- However, since we are in full control of the VM, it's possible to just make payload user accounts on demand and access them with sudo
  - LHCb and GridPP DIRAC contextualizations do this (but just 1 job per VM)
  - Since pilot frameworks already supposedly support glexec, this can be exploited immediately with a sudo wrapper placed at /usr/sbin/glexec ...
- If use glexec itself, could access local ARGUS banning service

machine/job features

- Proposed HEPiX protocol and current WLCG task force
- Allows site/host to communicate details of machine and the job slot to the job or VM
  - HS06, shutdown time for VM/host, CPU and memory limits
- A lot of this is made available by some batch systems already
  - But, different methods for different systems; absent on some platforms (Clouds)
- One file per value, either in a directory in the filesystem (local, NFS) or on a web server at well known addresses (169.254.169.254 …?)
- The basis for several scenarios for telling VMs and payload jobs what resources they have and how long they can run for
  - Want graceful termination of VMs to avoid disrupting payload jobs
  - eg on Vac, /etc/machinefeatures/shutdowntime is always set using the max_wallclock_seconds time for this experiment's VMs

The Masonry Problem...

The Masonry Problem



Masonry in VMs

- This problem applies in batch job scheduling
  - eg current WLCG Multicore TF discussions about reserving 8-processor job slots in the future and then trying to pack single processor jobs in while you wait for all 8 processors to become available
- But it also comes up with VMs running multiple payload jobs in parallel
  - How to get all payload job slots to finish at the same time so VM can be shut down without wasting resources
- And with provisioning multiprocessor VMs when the hypervisor is running a mix of single and multiprocessor VMs
  - How to back fill VM slots while wait for multiple processors to become free
- Solutions: good job length estimates from experiments? elastic jobs that can fill available time slots? elastic VMs whose resources of processors and memory can be increased and decreased?
  - Machine/job features very useful for communicating to VMs

Accounting

- For conventional grid sites we use APEL
  - either local database/publisher that CEs talk to (for CREAM/batch)
  - or direct publishing from CE into central APEL database (for ARC/batch)
- EGI does have a Cloud accounting task
  - APEL being extended in this context to support clouds and therefore VMs
- Also possible to treat VM-based systems as conventional CEs and report using the existing infrastructure and record formats
  - This is what Vac currently does. It writes PBS and BLAH accounting files. Read by APEL PBS parser. Each Vac space reported as one virtual CE.
- One question is whether we should be using wallclock or cputime for pledges etc with VMs
  - Sites tend to be strongly in favour of going to wallclock (nothing new there)
  - Tension between risk of inefficient I/O hardware vs risk of inefficient jobs

Monitoring

- Currently sites put a lot of effort into monitoring worker nodes, and services like batch systems and CE
- When jobs are run in VMs, “system management” responsibility shifts from site to experiment
  - As always, need monitoring systems to keep things working
  - And people looking at them to flag problems, write tickets, etc
- Might hope that as people's time at sites is freed up by move to VMs, then they will have more scope to take on shifts for experiments
- As all the instances of a VM are identical, we should gain from economies of scale
  - When a change to the VM or releases in cvmfs break something, it is immediately obvious because it is so widespread
  - Less time spent ticketing random sites that have forgotten to install some library
- Experiments might look at increasing regional support structures

Security

- A lot of focus on certification of VM images initially
  - Feels like this is less of an issue now we are starting to use them in production
  - We don't certify user jobs either ...
- “Having root access” pushes some buttons
  - But VMs are effectively user processes
  - Especially if you don't let them listen on or send from ports < 1024
- Key issue is probably the timely update of VMs when vulnerabilities become known
  - Reissue updated VM image?
  - Just restart services if using CernVM 3 because of hotfixes
- Same firewalling issues as with user (payload) jobs
  - Might want to stop DDoS attacks on popular websites etc
- Logging harder, but we don't log behaviour of users' binaries either

CHEP 2013 and pre-GDB talks

- “Distributed Processing and Data Handling A: Infrastructure, Sites, and Virtualization” track at CHEP
  - https://indico.cern.ch/event/214784/session/4/?slotId=3#20131015
  - Mostly Tuesday and Thursday sessions
  - Talks from ATLAS, CMS, LHCb, NA61
  - Plus sites using virtualization
- Stefan's track summary
  - https://indico.cern.ch/event/214784/session/10/contribution/519
  - Slides 8 to 17
- Virtualization discussed in July 2013 and Jan 2014 preGDBs
  - http://indico.cern.ch/event/253858/
  - http://indico.cern.ch/event/272783/

Summary

- Virtual machines in use at all types of site
  - Fabric, Cloud, Vacuum
- Well established model of image, contextualization, pilot framework
- Example contextualizations for LHCb, GridPP DIRAC, and ATLAS presented
  - Working with production jobs
- The Masonry Problem familiar from multiprocessor job slot reservation reappears with VMs
- Supporting VMs raises questions about how to do accounting, monitoring, and security
- Uniformity between VMs may save us support effort
  - But require some shift of effort from sites to experiments

Extra slides

Vac “Back Off” procedure

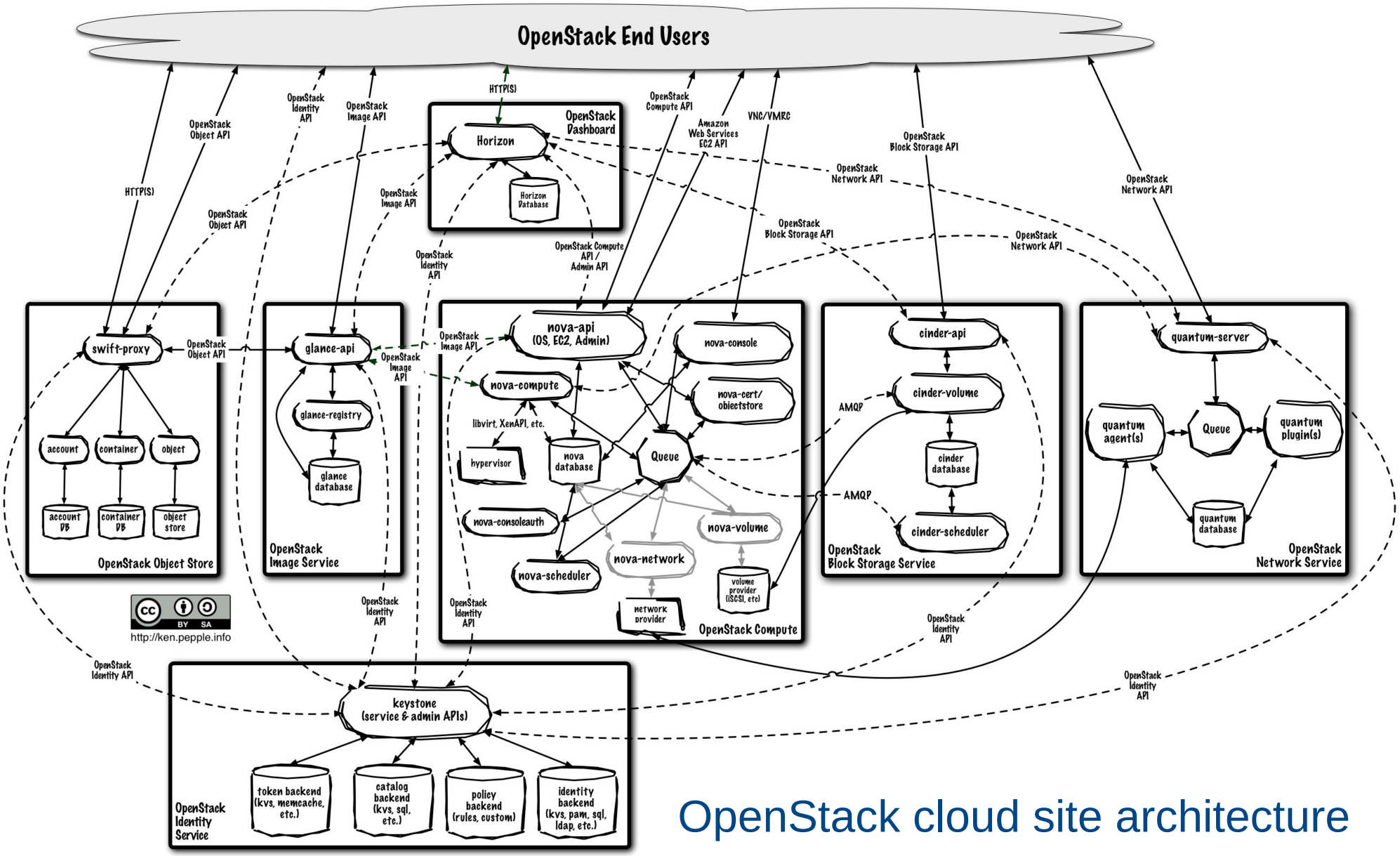
- To avoid overloading Matcher/TaskQueue, Vac implements “back off”
- If a VM finishes with “no work” / “banned” / “site misconfigured” outcomes then it counts as an abort
  - If no outcome given, then if a VM finishes after less than fizzle_seconds (600sec?) then it counts as an abort
- For a VM type (~experiment), if an abort has happened on any factory in the last backoff_seconds (600 sec?), then no more VMs of that type will be started
- After that, if an abort happened in the last backoff_seconds + fizzle_seconds and any new VMs have run for less than fizzle_seconds, then no more VMs of that type will be started
  - ie try to run one or two test VMs to see if ok now
- If backoff_seconds + fizzle_seconds have passed without more aborts, then can start VMs again as fast as slots become available

Vac UDP protocol

- Each factory has a list of all the factories in the same Vac space
  - (May offer multicast as an alternative in the future...)
- Sends UDP packet containing a JSON-encoded Python dictionary:
  - {"cookie": "179e6....cd3a5", "method": "status",
    "space": "vac01.tier2.hep.manchester.ac.uk"}
- One UDP reply for each VM assigned to a factory:
  - State (shutdown / starting / running), VM type etc
  - Outcomes of last VM instance run here for each VM type
- Use cookies to avoid external denial of service, since UDP
- Can also be used to query nodes (vac command ≈ PBS qstat)
- Use UDP port 995 (Roman Numerals: V=5, M=1000. 995=1000-5 !)

Target shares with Vac

- Vac avoids the phrase “fair shares”
- No history recorded: just a targetshares list in the configuration
  - Each time a new VM must be created, the VM type with the least running VMs, weighted by the shares, is tried
  - The back-off mechanism is there to stop trying VM types which have “recently” failed to get any work and failed to stay running
- This approach is very simple, and means the factory nodes can decide themselves what to do
  - Avoids a central management daemon which would be a single point of failure
- But these target shares are instantaneous
  - They are fair, in that if all experiments submit lots of jobs, the site shares out the capacity according to the stated shares
  - But they are unfair in that quiet periods aren't credited and carried forward



OpenStack cloud site architecture