Vac, graceful termination, and target shares

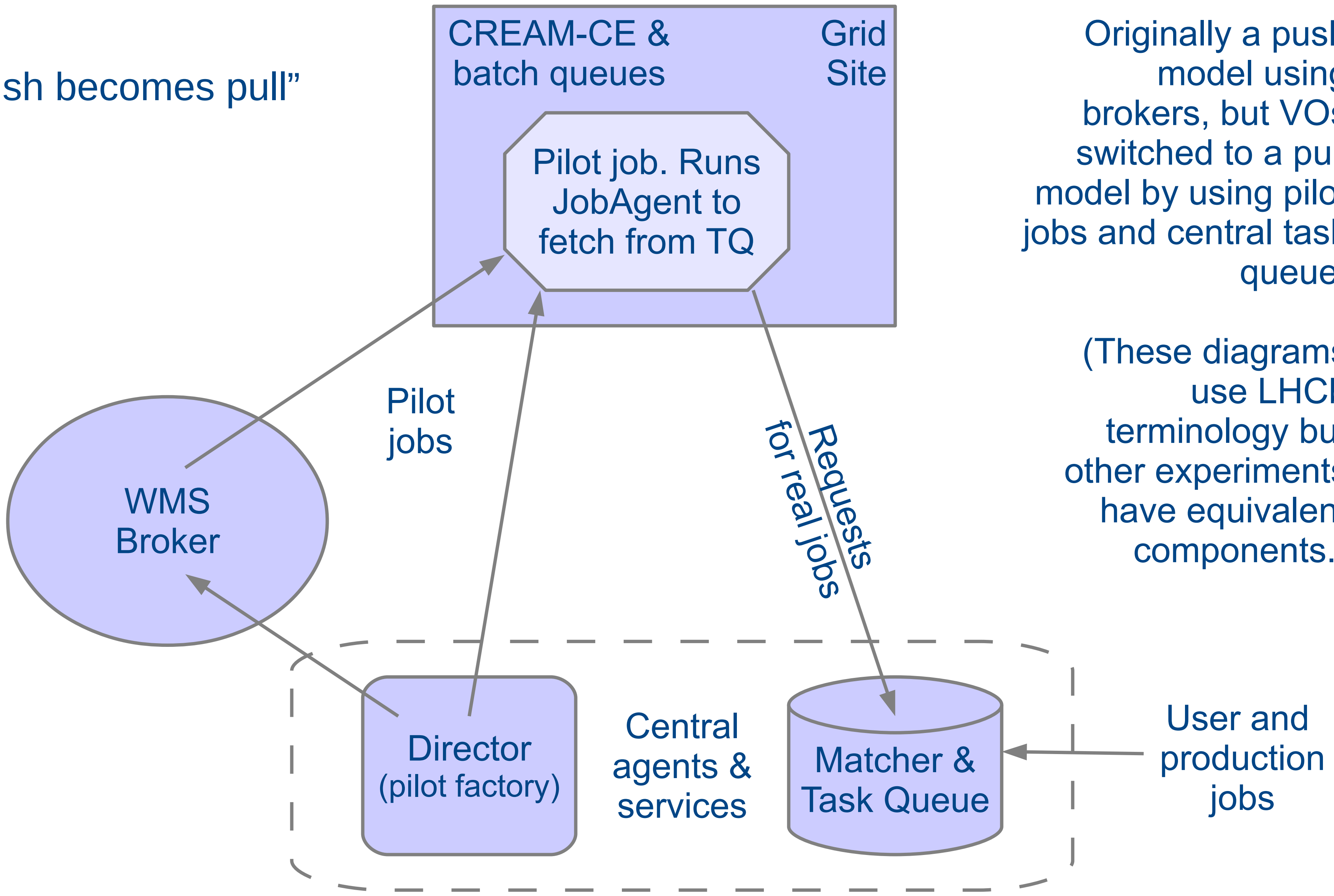
Andrew McNab
University of Manchester

Overview

- Vac and Vacuum
  - The Grid vs The Cloud vs The Vacuum
  - The Vac implementation
  - Production use within LHCb
- Graceful termination
  - Contracts
  - shutdown messages
- Target shares
  - Instantaneous target shares
  - Long-term target shares
- Summary
- (Extra slides with more details)

The Grid

“Push becomes pull”

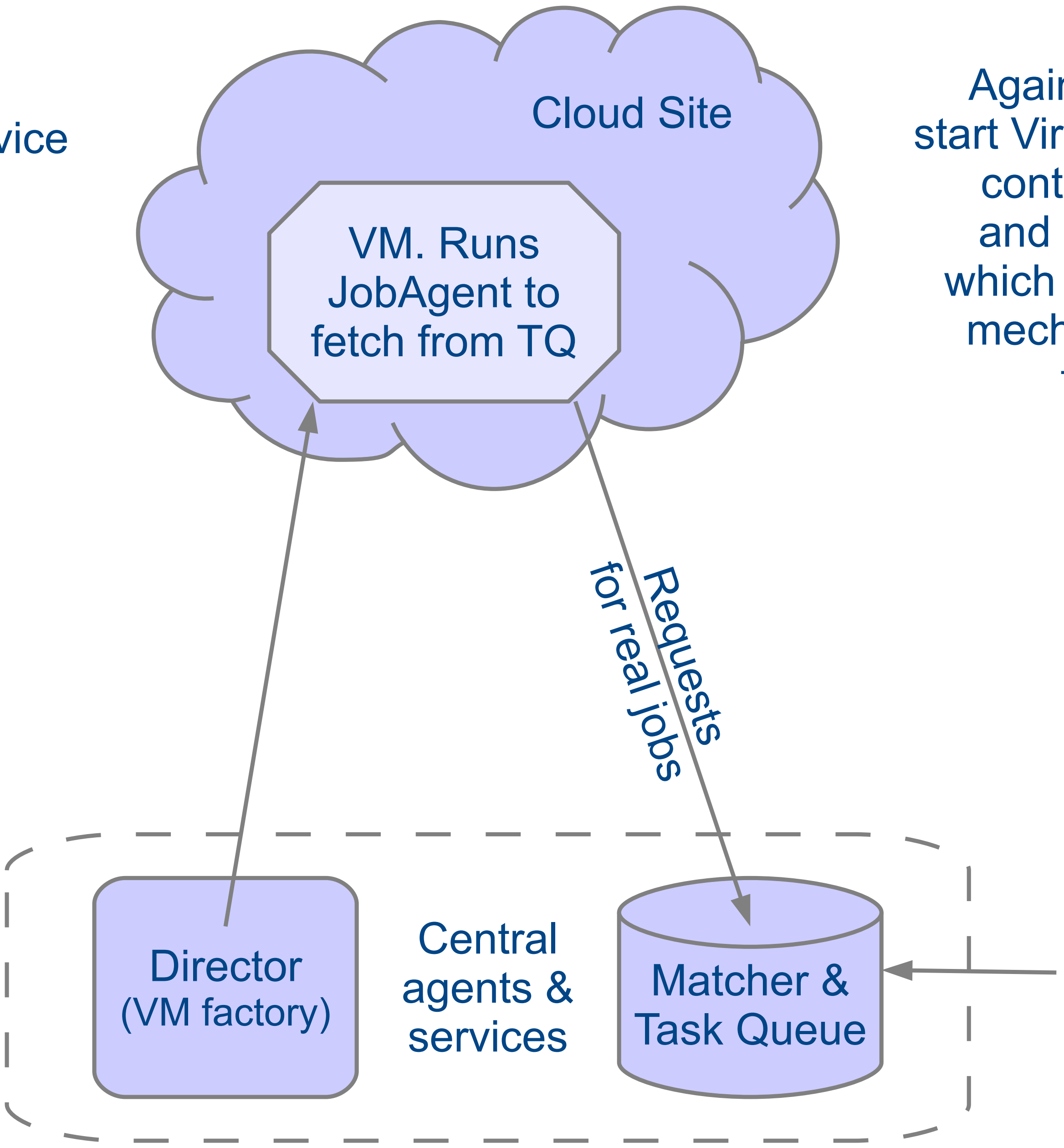


Originally a push model using brokers, but VOs switched to a pull model by using pilot jobs and central task queue.

(These diagrams use LHCb terminology but other experiments have equivalent components.)

The Cloud

Infrastructure-as-a-Service (IaaS)



Again uses Push to start Virtual Machines, contextualize them and run JobAgents which set up the Pull mechanism used to fetch real jobs.

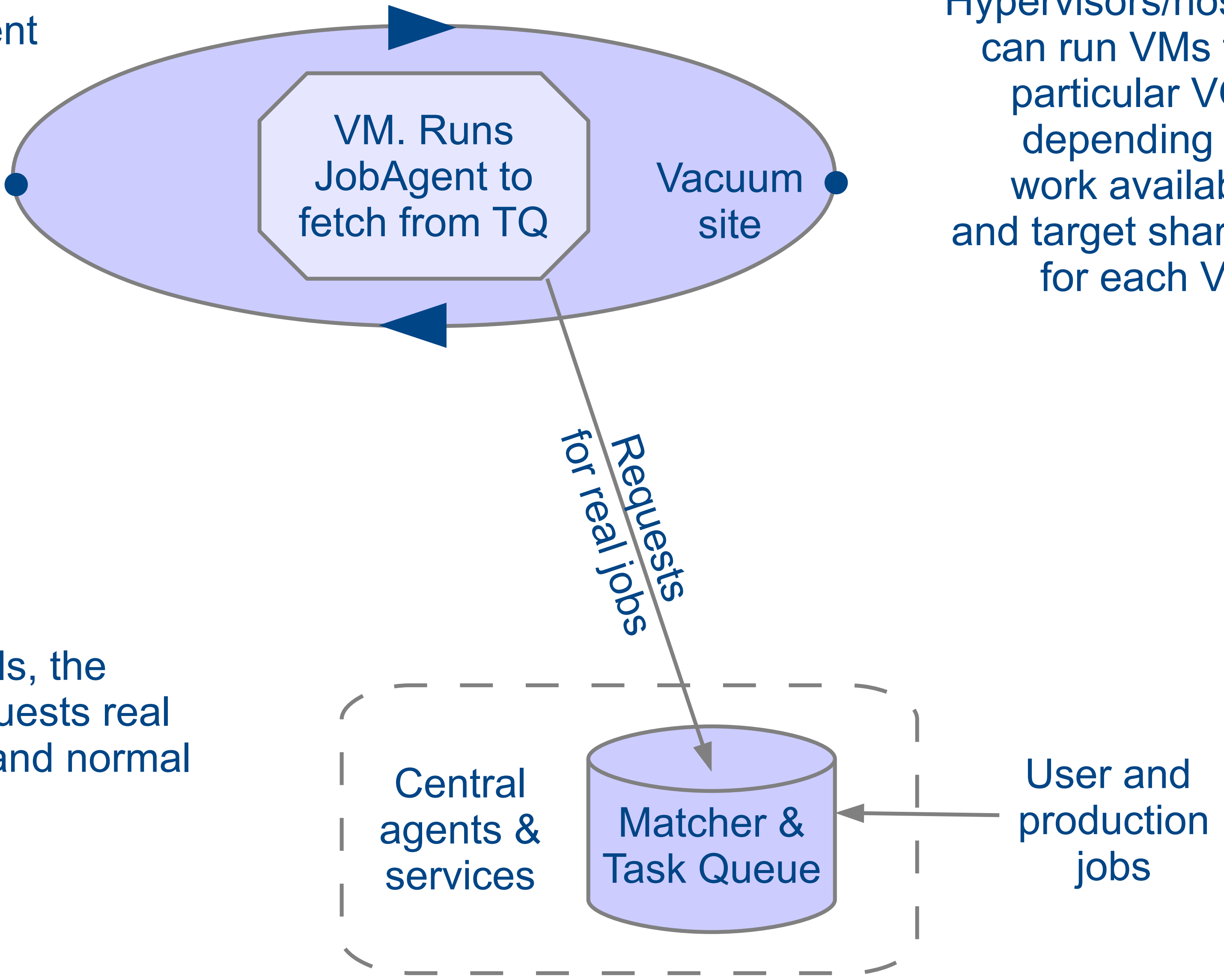
In LHCb, use the same TQ as for Grid and direct DIRAC execution of jobs.

"The Vacuum"

Infrastructure-as-a-Client (IaaC)

Instead of being created by VOs, the Virtual Machines appear spontaneously "out of the vacuum" at sites.

As with the other models, the JobAgent runs and requests real jobs from the Matcher and normal Task Queue.



Hypervisors/hosts can run VMs for particular VOs depending on work available and target shares for each VO.

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
- Independent hypervisors / VM factory nodes
  - Maybe site-wide configuration via Cfengine/Puppet/..., but no head node
- Factory nodes decide which types of virtual machine to create
  - One or more VM types per experiment
  - Factories may communicate with each other to achieve desired shares
- 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

- A vacd daemon creates and applies contextualization to transient VMs on each physical “factory node”
- Each site or Vac “space” is composed of independent factory nodes
  - All using the same /etc/vac.conf, /etc/vac-targetshares.conf etc
- Factories communicate with each other via UDP
  - Factory chooses which type of VM to start in free slots based on what else is running and the target shares – one VM per JobAgent
- So far supports CernVM image and ISO (AMI) contextualization
  - VO supplies user_data, prolog.sh and epilog.sh; Vac makes the ISO image
  - Vac also makes and NFS exports HEPiX machinefeatures etc directories to the VM, including shutdowntime and shutdown_command
- See http://www.gridpp.ac.uk/vac/ for RPMs, documentation etc

Vac sites

- So far Vac is running at 4 sites:
  - Manchester, Lancaster, Oxford, Imperial College
- 3 are running routine LHCb production Monte Carlo:
  - Manchester (3 weeks), Lancaster and Imperial (since last week)

- Aim to make Vac sites look like “normal” WLCG sites where possible
- uk.ac.gridpp.vac service endpoint type registered in GOCDB
- Accounting data successfully published to site APEL database
  - vacd writes PBS and BLAHP format accounting files which work with the existing APEL PBS parser
- Beginning to look at BDII, although this may not be necessary
  - Use UDP protocol to gather data; republish via redundant slapds on factories

Graceful termination

- Vac strategy is to use machinefeatures and jobfeatures
  - NFS exported from factory node into VM, so easy to populate and to update
- So far we set shutdowntime when the VM is created
  - Always kill the VM at that time if still running
  - Will add the other VM/job length values as part of new task force effort
- We set /etc/machinefeatures/shutdown_command
  - Allows the VM to shut itself down. This may not be appropriate for clouds, but is extremely useful for IaaC systems, like Vac and BOINC
- Vac's default shutdown wrapper saves any command line arguments as the shutdown message
  - These help the site see why VMs are terminating
  - This is part of the VM being graceful!
  - Uses Vac's writeable NFS directory /etc/machineoutputs

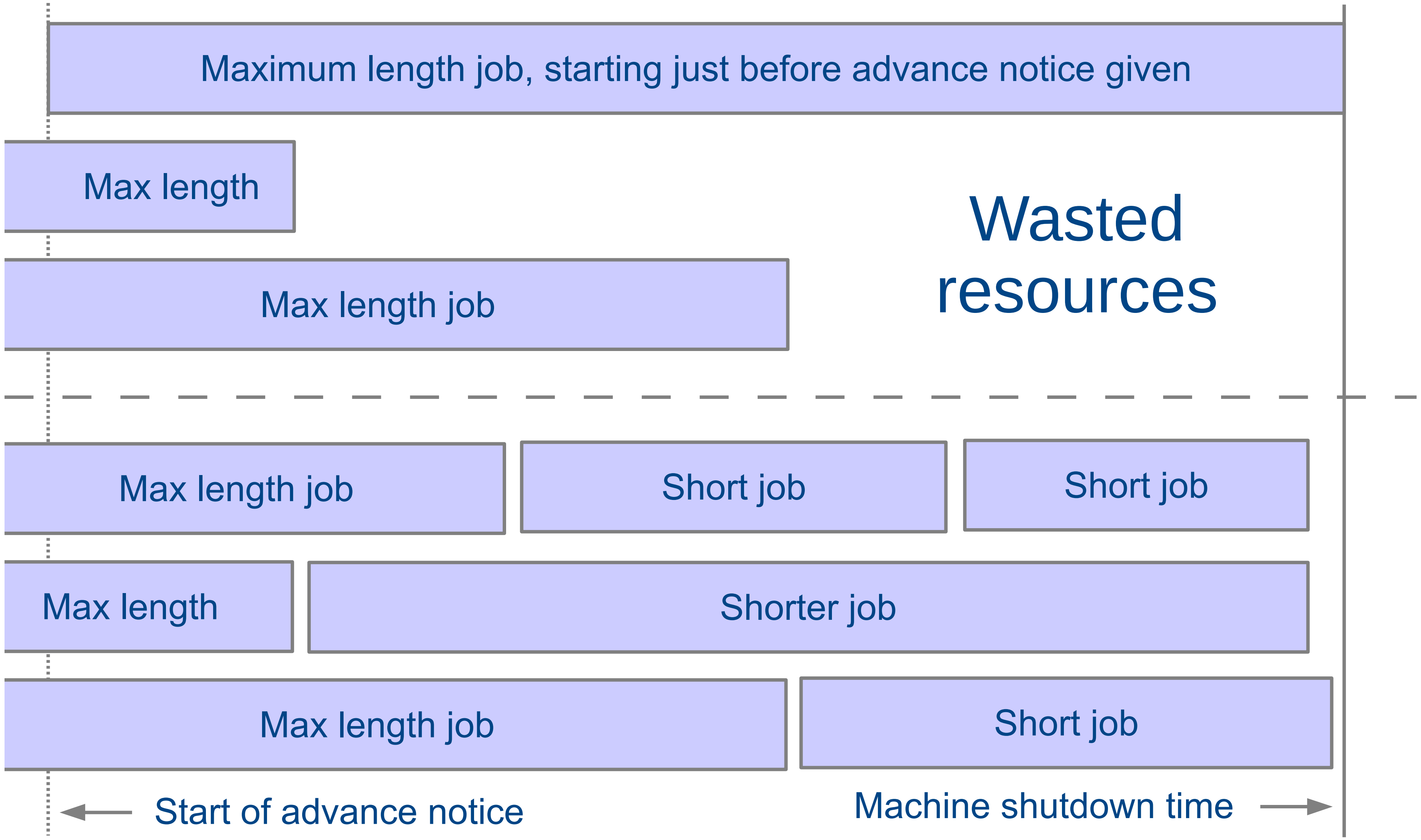
Shutdown message codes

- Passed as arguments to shutdown_command:
  - 100 Shutdown as requested by the VM's host/hypervisor
  - 200 Intended work completed ok
  - 300 No more work available from task queue
  - 400 Site/host/VM is currently banned/disabled from receiving more work
  - 500 Problem detected with environment/VM provided by the site
  - 600 Error related to job agent or application within VM
- As with HTTP codes, room to insert more numbers for finer grained information in the future
- Vac uses this information programmatically, but useful to admins too
- More details and rationale:
  - https://www.gridpp.ac.uk/wiki/HEPiX_shutdown_command

Graceful termination as a contract

- Sites and jobs / VMs always have some kind of understanding about what the rules are
  - Site can kill jobs at any time?
  - Sites will always abide by job time limits?
  - Sites will always give 30 seconds notice?
- If termination is graceful, the job/VM has a chance to save the work already done
  - What is enough needs to be agreed between sites and individual experiments, or at the WLCG level, so people submitting jobs and productions have predictability
- Sites want to be able to terminate jobs/VM for operational reasons (to reboot a rack say) or perhaps to rebalance workloads.
- Sites want to do this gracefully too because they want to demonstrate that they do lots of useful work for the experiments.

Graceful mechanisms allow job “masonry”...



Target shares

- Vac avoids the phrase "fair shares"
- No history recorded: just a targetshares list in configuration
  - Each time a new VM must be created, the VM type with the least running VMs, weighted by the shares, is tried
  - (There is a back-off mechanism to stop trying vmtypes 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

Long-term target shares

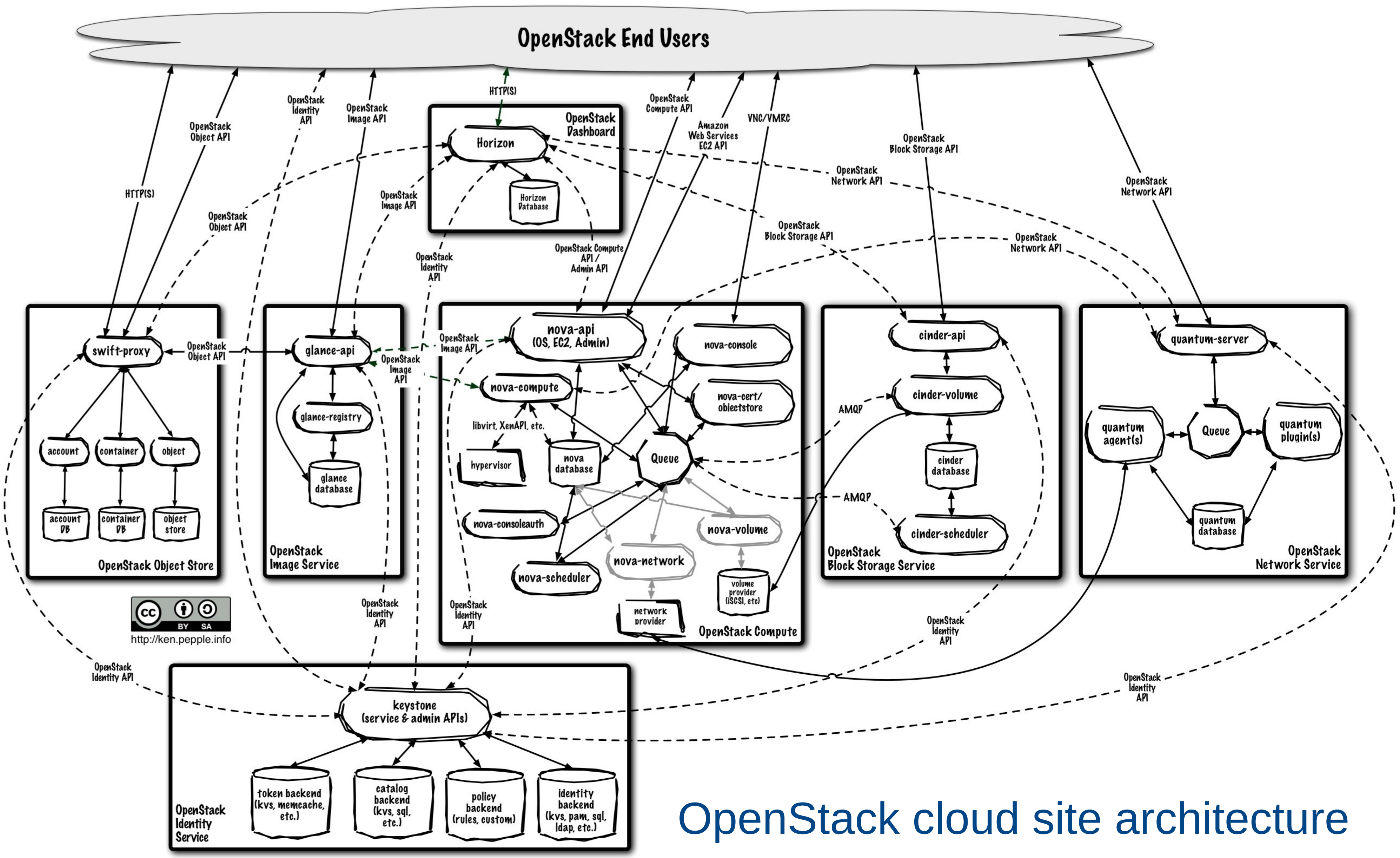
- The intention is that sites address this by updating the targetshares list and pushing it out to the factory nodes
  - Can use puppet or whatever they use for configurations elsewhere
  - Separate /etc/vac-targetshares.conf can be used for convenience
- The plan is to provide a tool to use the local APEL database to calculate targetshares to achieve long term target shares figures
  - For example: “We've not run any jobs from Expt. X this quarter so far. Set X's instantaneous target share so if some jobs do arrive from X, they will all be run.”
- Rather like an offline version of MAUI with a long time constant
- The big advantage is that if this tool fails to run, the system still carries on running jobs with the current share values and doing useful work while you fix it.
- Smaller sites can also just do this by hand every week or so (as some do with short term MAUI shares...)

Summary

- Vacuum is an alternative to Grid and Cloud models
  - Quite complementary to Clouds + VMs with the same VMs on both
- Vac is an implementation of this model
  - VM factory on each physical machine, communicating via UDP
- Running at 4 sites; running production LHCb jobs

- Graceful termination as a contract
  - Communication with site part of this
- Instantaneous vs long-term target shares
  - Use existing site APEL database to achieve long term shares

Extra slides



OpenStack cloud site architecture

Vac UDP protocol

- Each factory has a list of all the factories in the same Vac space
- 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
  - Start time
  - Outcomes of last VM instance run here for each VM type
- Use cookies to avoid external denial of service, since UDP
- Use UDP port 995 (Roman Numerals: V=5, M=1000. 995=1000-5 !)

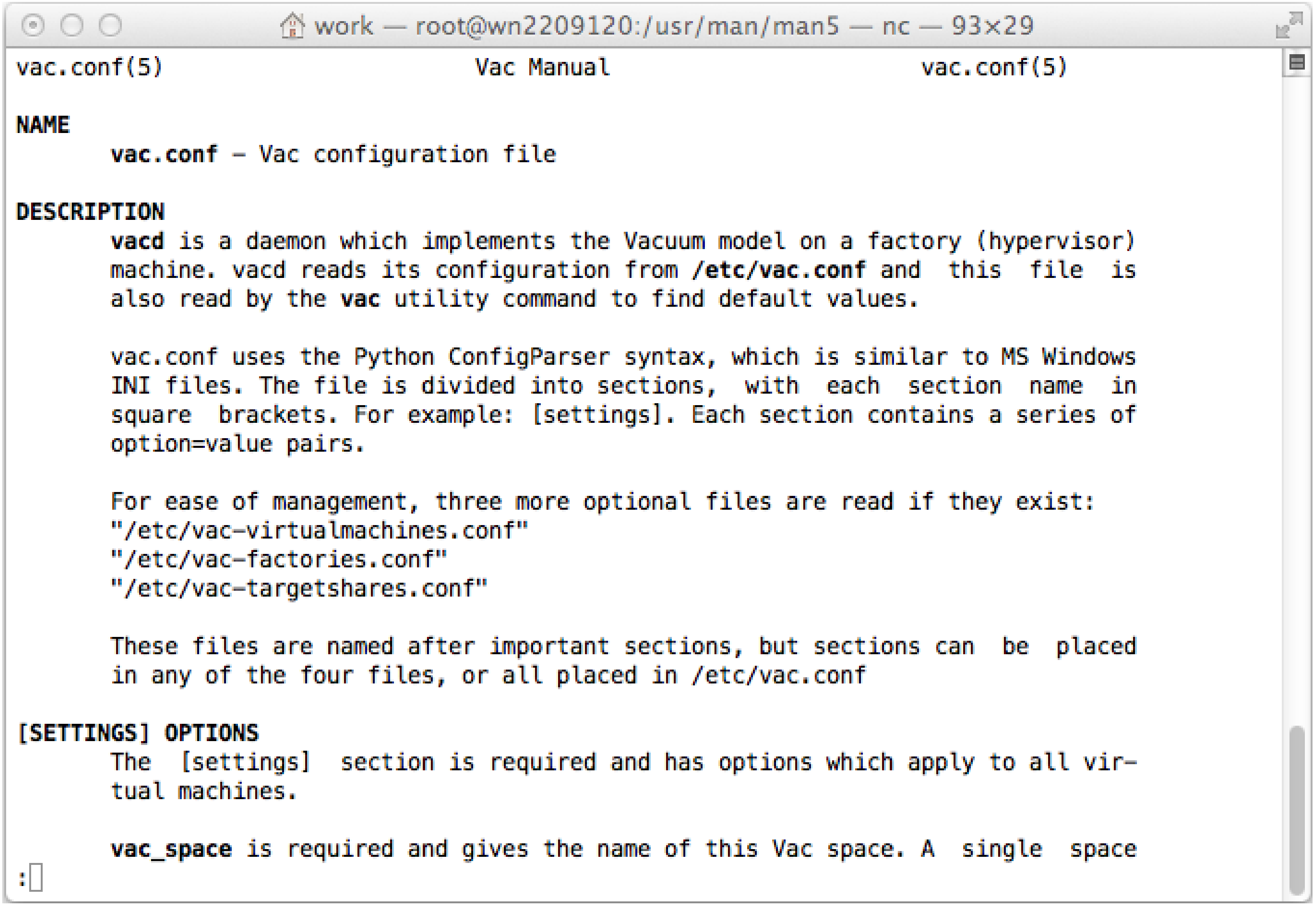
“Back off”

- 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

Interface with LHCb JobAgent

- Contextualization procedure causes JobAgent to be started in VM
- Vac expects VMs to shutdown if they can't find any work to do
- HEPiX VM working group provides shutdown_command protocol for site/factory to tell the VM how to shut itself down
  - vac-shutdown-vm is a wrapper around “sudo shutdown -h now”
- Proposal for VM to return message code + message saying why it has shutdown (eg “300 Nothing to do” if no more work in TQ)
  - vac-shutdown-vm will write this to /etc/machineoutputs/shutdown_message
  - this directory is NFS-exported from factory into VM; Vac examines it afterwards
- TimeLeft.py is being extended to read HEPiX shutdowntime
  - Vac creates shutdowntime using maximum allowed lifetime of a VM of this type
  - Can also be used to ask VM to stop with, say, 24 hours warning

Documentation...



```
vac.conf(5)                       Vac Manual                        vac.conf(5)

NAME
       vac.conf - Vac configuration file

DESCRIPTION
       vacd is a daemon which implements the Vacuum model on a factory (hypervisor)
       machine. vacd reads its configuration from /etc/vac.conf and  this  file  is
       also read by the vac utility command to find default values.

       vac.conf uses the Python ConfigParser syntax, which is similar to MS Windows
       INI files. The file is divided into sections,  with  each  section  name  in
       square  brackets. For example: [settings]. Each section contains a series of
       option=value pairs.

       For ease of management, three more optional files are read if they exist:
       "/etc/vac-virtualmachines.conf"
       "/etc/vac-factories.conf"
       "/etc/vac-targetshares.conf"

       These files are named after important sections, but sections can  be  placed
       in any of the four files, or all placed in /etc/vac.conf

[SETTINGS] OPTIONS
       The  [settings]  section is required and has options which apply to all vir-
       tual machines.

       vac_space is required and gives the name of this Vac space. A  single  space
:
```