Fermilab | U.S. DEPARTMENT OF ENERGY | Office of Science

Best practices in job management

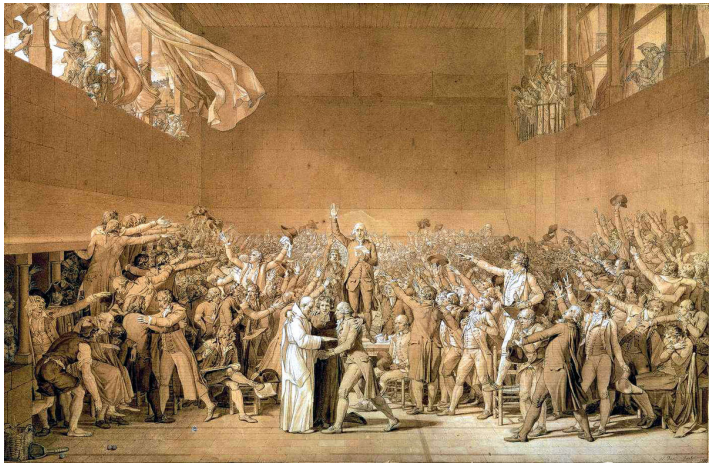
Ken Herner

FIFE Workshop

20 June 2016

FABRIC FOR FRONTIER EXPERIMENTS

20 Jun 1789: The Tennis Court Oath



Jacques-Louis David, *Le_Serment_du_Jeu_de_paume* (public domain)

Outline

- The job environment (what happens to your job after submission)
- Common problems and solutions via best practices
- Quotas vs. Priorities
- Limits and large submissions

Girl Scout Law

I will do my best to

…

use resources wisely,

…

make the world a better place,

….

Your job on the worker node

- You execute jobsub_submit ... whatever
  - You get credentials on the jobsub server, executables and input files get copied around, and condor_submit is issued for you
  - You wait in the queue for a while and you start running on a worker node
- Your job doesn't just run via HTCondor on the worker node. It actually runs in a *glidein* (part of the glideinWMS system) on the worker node, which has its own HTCondor instance
- The glideins are *partitionable*: they start out as a certain number of cores with some memory (8 CPUs and 16000MB memory on GPGrid; other sites differ) and they're then carved up into smaller pieces for individual jobs. Could have a 4-CPU, 8000MB job and four 1-CPU, 2000MB jobs running in same glidein

Example .log file

000 (8859960.000.000) 04/08 09:33:44 Job submitted from host: <131.225.67.139:9615?addrs=131.225.67.139-9615&noUDP&sock=3611905_50e2>

...

...

001 (8859960.000.000) 04/09 16:11:11 Job executing on host: <18.12.8.228:56921?CCBID=131.225.67.218:9630%3faddrs%3d131.225.67.218-9630#3677262%20131.225.67.219:9630%3faddrs%3d131.225.67.219-9630#3675980&noUDP>

...

028 (8859960.000.000) 04/09 16:11:11 Job ad information event triggered.

JOB_GLIDEIN_Name = "gfactory_instance"

JOB_Site = "MIT"

Proc = 0

JOB_GLIDEIN_Entry_Name = "CMSHTPC_T2_US_MIT_ce01"

EventTime = "2016-04-09T16:11:11"

TriggerEventTypeName = "ULOG_EXECUTE"

JOB_GLIDEIN_SiteWMS_Queue = "ce01.cmsaf.mit.edu"

TriggerEventTypeNumber = 1

ExecuteHost = "<18.12.8.228:56921?CCBID=131.225.67.218:9630%3faddrs%3d131.225.67.218-9630#3677262%20131.225.67.219:9630%3faddrs%3d131.225.67.219-9630#3675980&noUDP>"

JOB_GLIDEIN_Site = "MIT"

JOB_GLIDEIN_SiteWMS_JobId = "116651.0"

MyType = "ExecuteEvent"

JOB_GLIDEIN_ProcId = "6"

JOB_GLIDEIN_Schedd = "schedd_glideins7@gfactory-1.t2.ucsd.edu"

JOB_GLIDEIN_ClusterId = "3661662"

Cluster = 8859960

JOB_GLIDEIN_Factory = "SDSC"

JOB_GLIDEIN_SiteWMS_Slot = "slot1_1@t2bat0228.cmsaf.mit.edu"

Subproc = 0

EventTypeNumber = 28

JOB_GLIDEIN_SiteWMS = "HTCondor"

...

Setting Up for Success

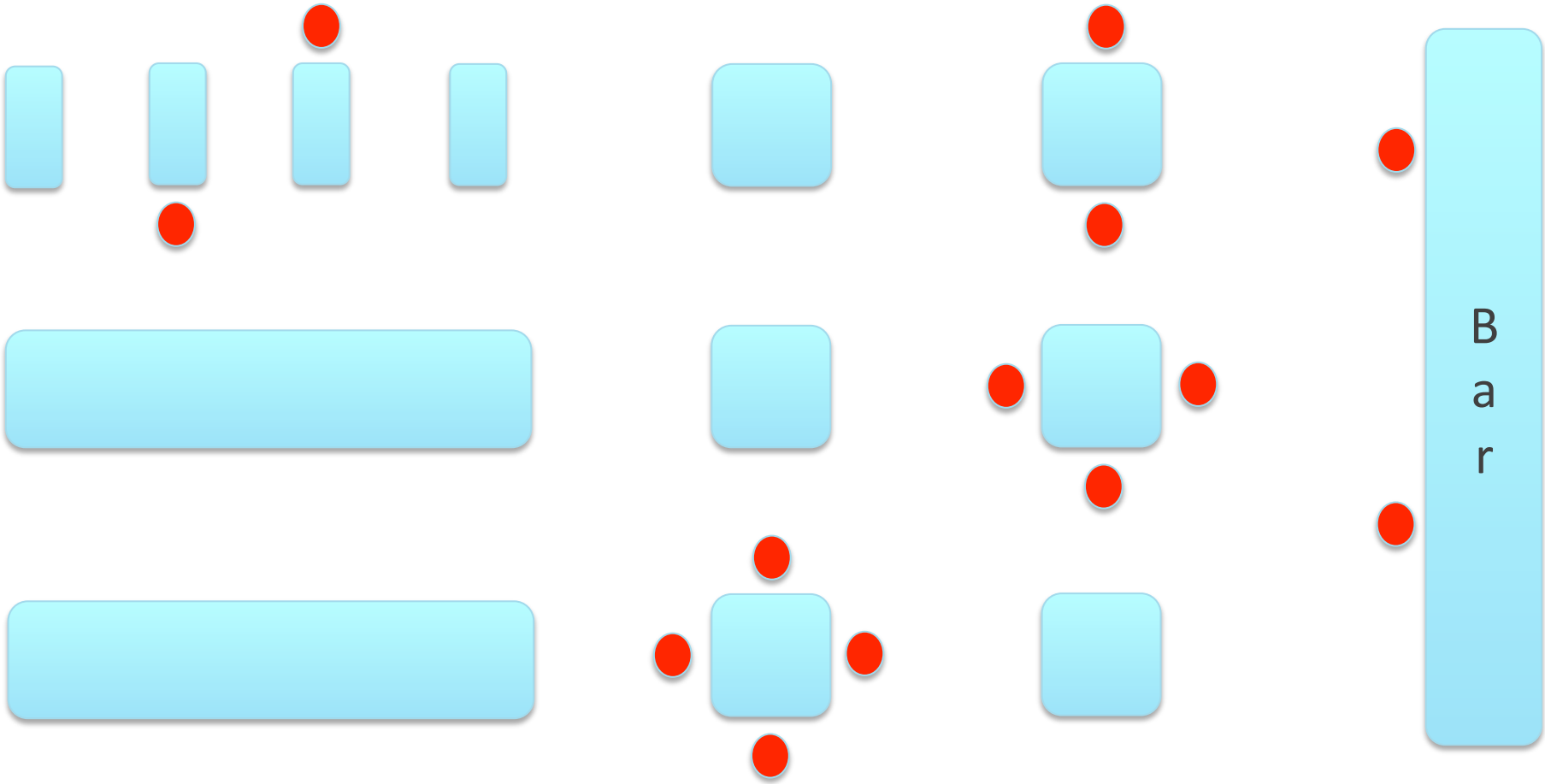
- Successful management entails these things:
  - Getting jobs to start quickly
  - Making jobs run efficiently and minimizing wasted time
  - Maximizing success rate
- Most job failures that we see stem from several things:
  - Inappropriate job resource requests (too much and too little)
  - Resource access issues (databases, etc.)
  - Inadequate testing outside of the interactive machines
- Let's give some helpful hints on each of these

Best Practice #1: Accurate resource requests

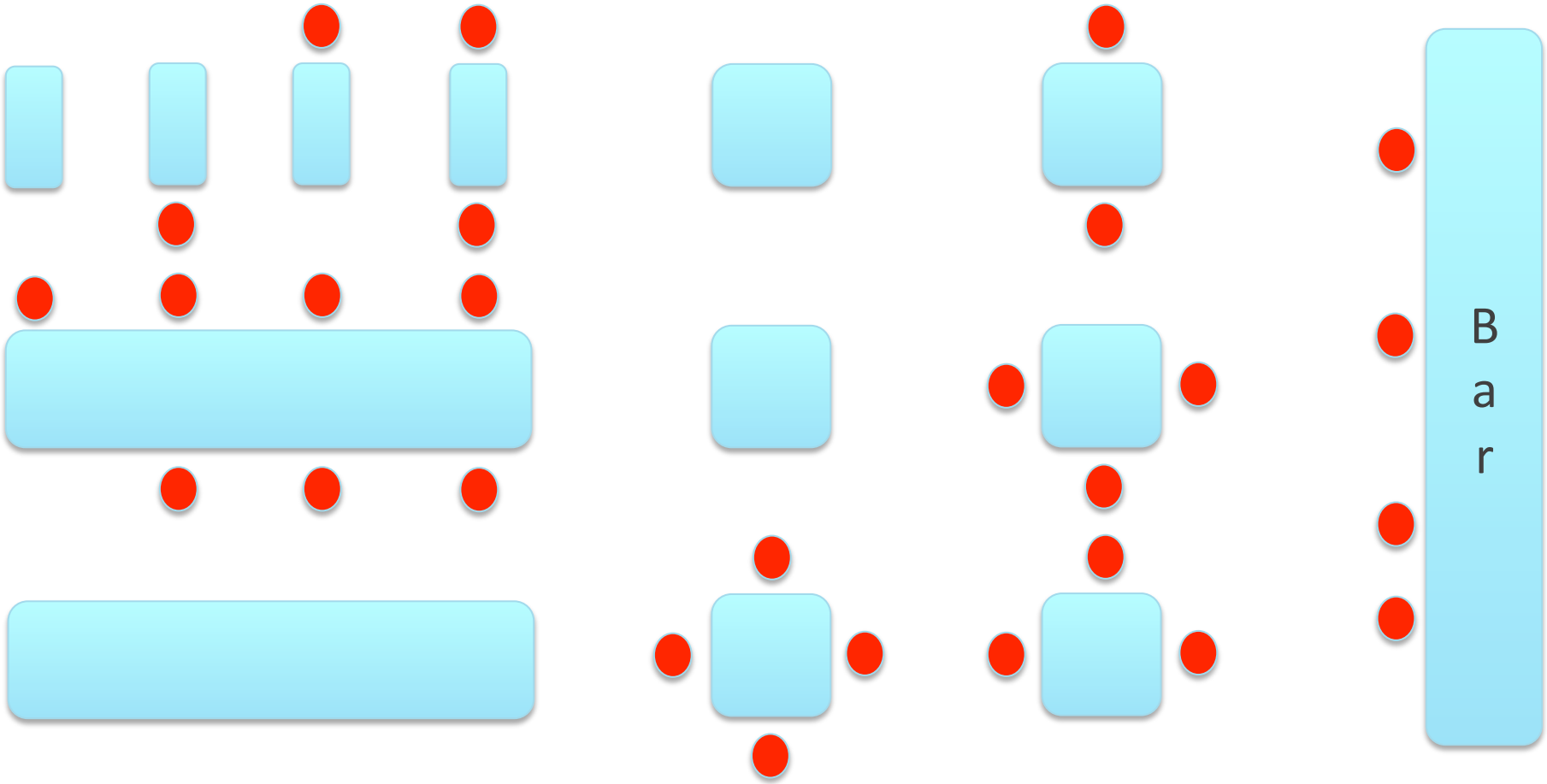
- Single most important thing you can do. Check memory, disk, and TIME
- If you use Art, use the profiling tools (see Marc Paterno's talk)
- Request only what you need. The default might be more than you need. If it is, don't just stick with it. You can gain a lot by requesting *less* than the default.
- Consider the problem of getting a table in a restaurant:

Lunchtime

- A table seating more than one is no problem right now...

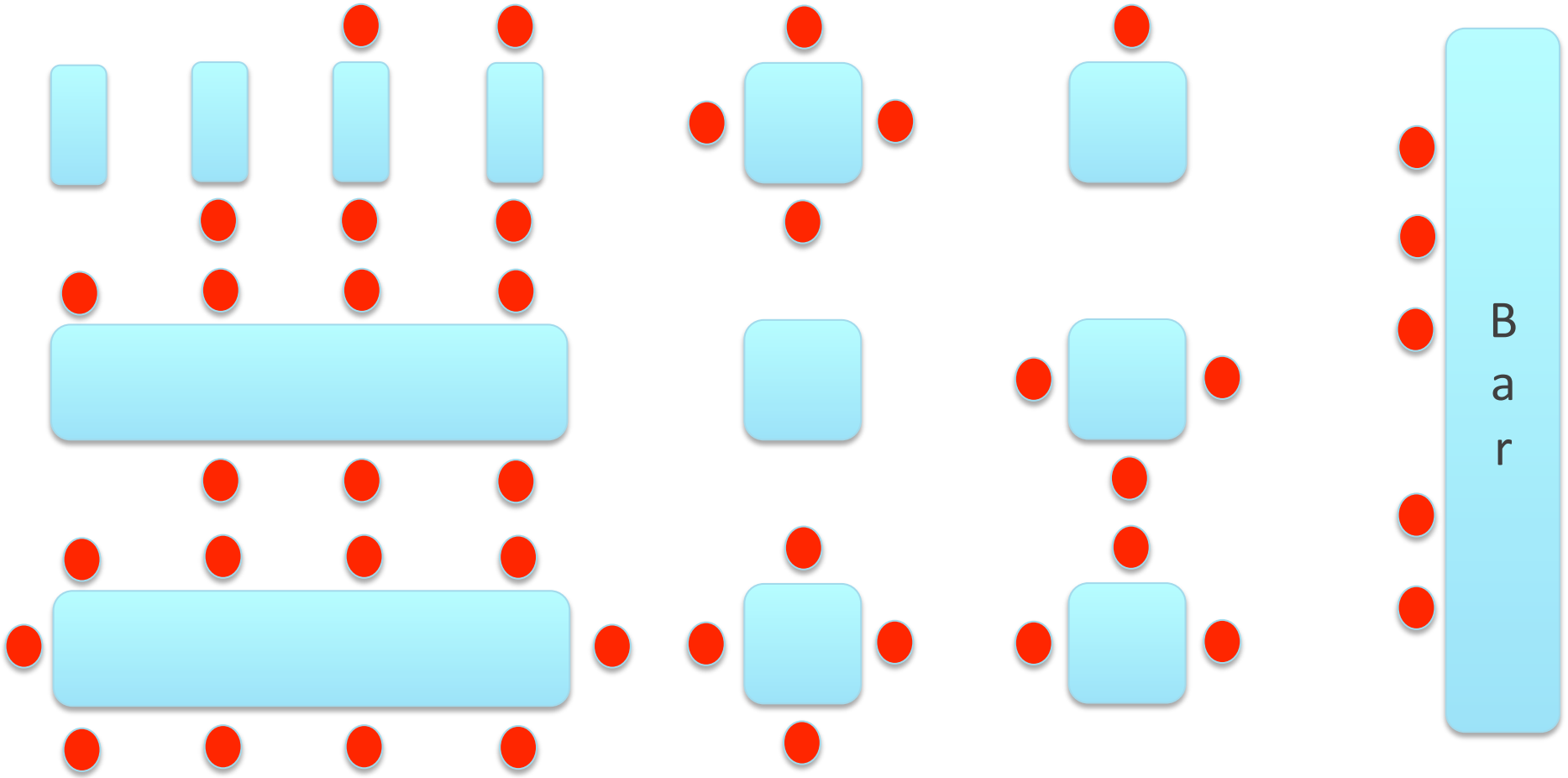


Lunchtime



Fermilab

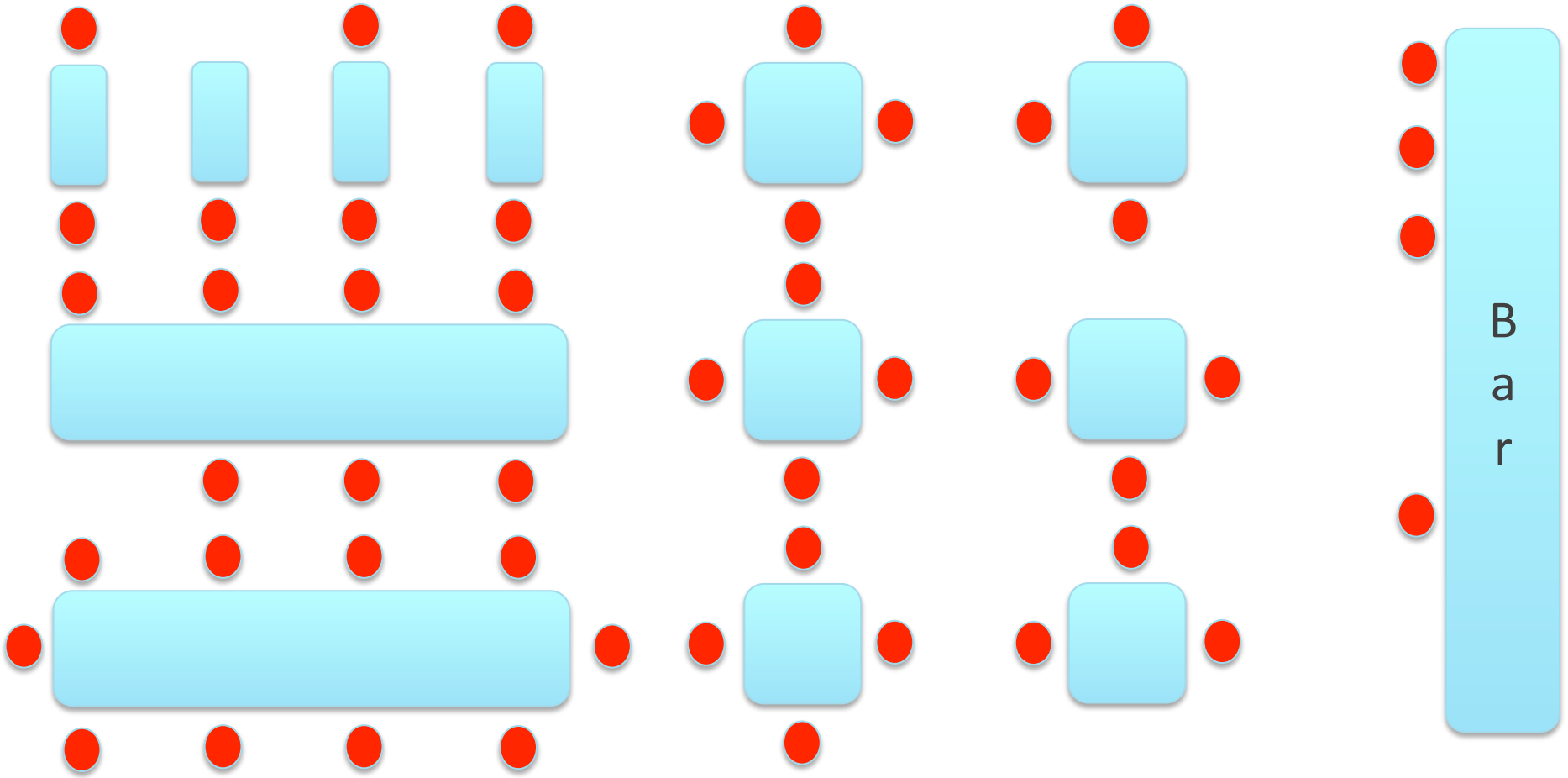
Lunchtime



Fermilab

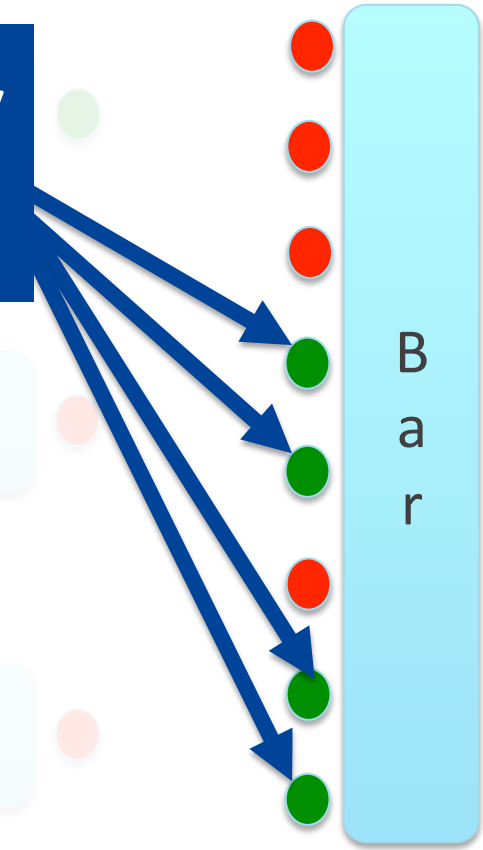
Lunchtime

- But if you want a table now, you are going to wait



Lunchtime

But there are plenty of seats at the bar!



Extending the analogy

- Suppose the grid is very busy and for a moment there is only one 8-core 16 GB glidein with some free space. Currently it's running jobs with the following requests:

Slots	CPUs	Memory (MB)
Job 1	1	2000
Job 2	1	2000
Job 3	1	1900
Job 4	1	1900
Job 5	1	2500
Job 6	2	4000
Tot	7	14300

Extending the analogy

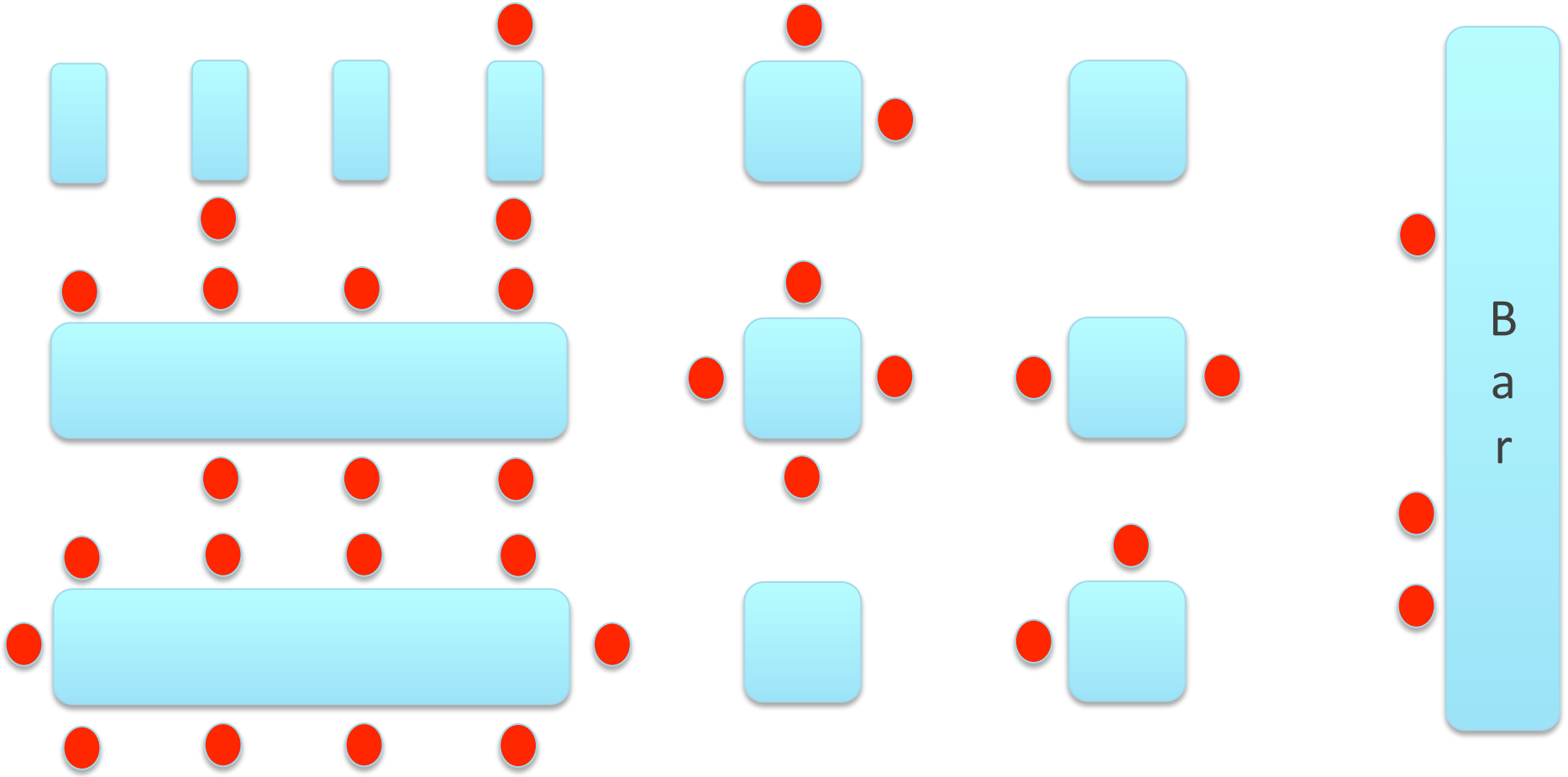
- Suppose the grid is very busy and for a moment there is only one 8-core 16 GB glidein with some free space. Currently it's running jobs with the following requests:

Slots	CPUs	Memory (MB)
Job 1	1	2000
Job 2	1	2000
Job 3	1	1900
Job 4	1	1900
Job 5	1	2500
Job 6	2	4000
Tot	7	14300
Free	1	1700

A job requesting 1 CPU
and <= 1700 MB memory
can run here; all others have to wait...
We see glideins looking like this
all the time on GPGrid

Another scenario

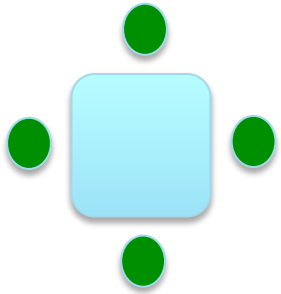
- Let's say we go as a party of 8 (they don't take reservations)
- Here, we will have to wait for the big tables to open up

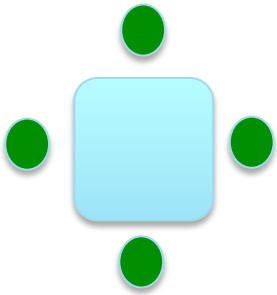


Another scenario

- Let's say we go as a party of 8 (they don't take reservations)
- Here, we will have to wait for the big tables to open up

But if we split into two parties of 4, we can start immediately.





But requesting too little isn't good either

- As you may know, your job will be automatically **held** if it goes above its memory or local disk request
- With partitionable slots this is very important
- At some sites if the overall glidein memory usage goes over the max (could be caused by only one job), the *entire* glidein is killed immediately (all jobs within it are lost)
- Memory/disk usage checks runs every 2 minutes
- It is *possible* that a job could go over its request between the checks for just a moment and not get held (but a remote site might be doing other monitoring and kill it anyway)
  - Considering ways to deal with this case in Condor
- Going over run time is slightly different, but also bad

Job run times

- Each glideins only last for a finite time
- For about 6 months we have had a run time option in jobsub
  - Intended to reflect the *upper* limit of your expected run time
  - Default is now 8 hours unless you change with the --expected-lifetime option
- Previously glidein's remaining time was not considered
  - If a job is still running when glidein expired, there was an exception, a disconnection, and the job went back into the queue, restarting from the beginning
- Job matchmaking now ensures that you will not run inside a glidein that will expire less than (requested run time) in the future
  - Number of unexplained disconnects and restarts has gone down considerably

Working Over Time

- What happens if your job runs over its request?
  - Right now it will continue to run until it finishes or the glidein expires (or you could get pre-empted for some reason)
  - If the glidein expires, job is lost and will start from beginning
- The glidein expiration time could be one second after you go over, or it could be days. It is just luck.
  - *Do not assume you can run for any longer than the requested time.*
- **Soon, we will start holding jobs that go over their requested time**
- Will probably also move to preferentially matching jobs to shortest glidein that still satisfies request
- For certain workflows, the run time can easily be made variable, so trying to figure out the “right” request is difficult

Getting more resources with any-length glideins

- What if your workflow could deal with glideins of any length?
- If yes, and you want more resources by making use of soon-to-expire glideins for as you as you can, you can do the following: set your expected lifetime to 0
  - You will then get the first glidein that matches your other requirements, but there is no guarantee of any minimum
- It is **your responsibility** to structure your workflow to check the amount of time remaining in the glidein, and to exit cleanly if your workflow can't finish the next step or process another file (this is *ideal* for SAM projects!) But if you do that:
- When we move to holding jobs based on going over the run time request, **jobs that request 0 would be exempt**

Checking remaining time and exiting safely

- We now provide an environment variable in the job called FIFE_GLIDEIN_ToDie (expressed in time since Epoch)
- You can compare that to the current time and then take action if the remaining time is below some threshold. Maybe something like this:

```
# Do something like this before processing another input file (each time.)
# Set some number of seconds below which you can't process another file.
MIN_TIME=600
tnow=`date +%s` # time since Epoch
tleft=$(( $FIFE_GLIDEIN_ToDie - $tnow ))
if [ $tleft -le $MIN_TIME ]; then
# do stuff to exit cleanly before time runs out
fi
```

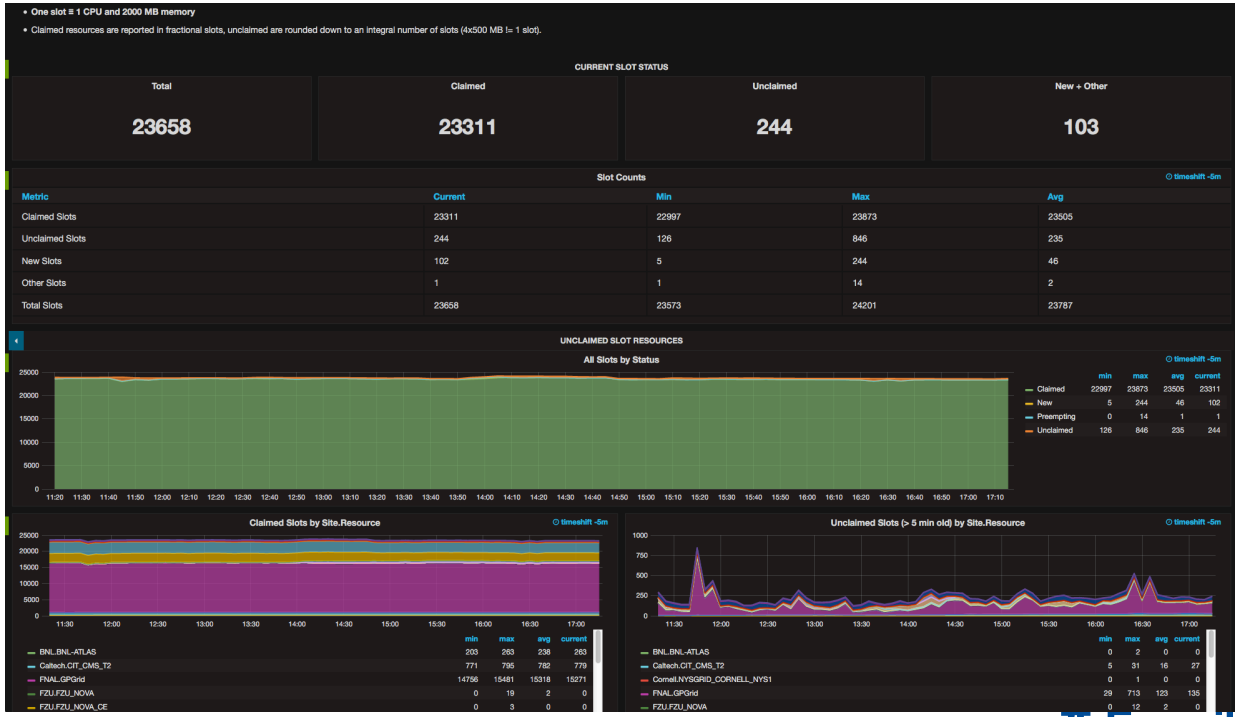
- This variable should be in all glideins now; test it out!

Setting up your resource request

- Jobsub has --memory, --disk, --cpu, --expected-lifetime opts
- --memory and disk will take units of KB, MB, GB, TB (default MB for memory and KB for disk)
  - Important note: **Right now 1 GB = 1024 MB, not 1000 MB**
  - Requesting 2000MB vs. 2048MB does make a big difference!
  - Default request is 2000 MB memory and 35000000KB disk
- --cpu takes an integer; default is 1
- --expected-lifetime can take a number with units of h,m, or s (default s) or a “short”, “medium”, or “long” preset (currently 3, 8, 24 hours)
  - default lifetime request is 8 hours
- A well-formed request might look like:
- --cpu=1 --memory=1800MB --disk=15GB --expected-lifetime=6h

How many “slots” am I using?

- Recall a “slot” is normalized to 1 CPU, 2 GB memory and multiples thereof
  - e.g. 1 CPU, 4GB is 2 “slots”; 2 CPU, 8 GB is 4; 2 CPU, 1 GB is 2
- Based on resource **request**, not actual usage
- Some FIFEMON plots are slot-weighted numbers (can have fractional slots).



Fermilab

Best Practice #2: Accessing non-Grid resources

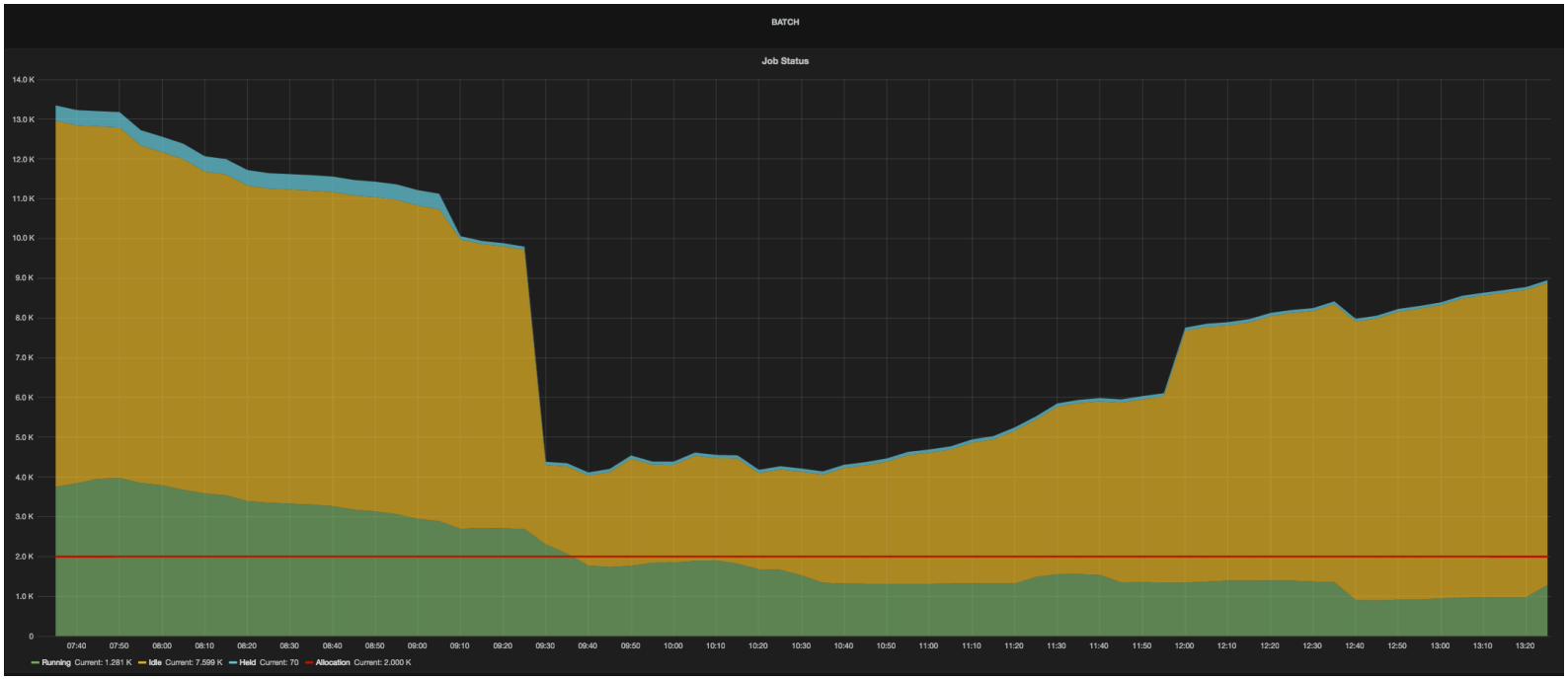
- Some experiments have certain workflows that talk to databases, copy files from non-FNAL resources, etc.
- Sometimes these resources get overwhelmed
- Some things you can do to help these cases:
  - Implement **timeouts** and **retry logic** in your scripts
    - Goes for file repos and databases especially
  - Consider limiting the number of simultaneous jobs with the --max_concurrent jobsub option (will create a DAG)

Best Practice #3: Write scripts that are OSG-ready

- What does OSG-ready mean? It includes, but isn't limited to:
  - No BlueArc dependence. Get code from CVMFS and/or tarballs copied in from dCache areas via ifdh. Do not copy to or from BlueArc, period.
  - Avoid hard-coding file paths, especially home areas (CVMFS is OK though)
  - Do not count on a certain user IS when running the job
  - Check for the existence of necessary files before using them
  - Check exit codes of all major steps; print useful messages on failure
- **Getting on the OSG is the single best way of increasing the amount of (free) resources available to you.**
- See Bo's talk and the OSG submission tutorial tomorrow

Spotting Problems

- Sometimes jobs don't start when we want them to (yesterday)
- FIFEMON is of course a good place to start
- We have to be careful to understand what we're seeing though. Example: is there a problem based on this plot?



Running jobs Idle jobs Allocation

Fermilab

Quotas vs. Priorities

- Most everyone is used to the quota setup: You have an enhanced claim on slots up to your allocation.
  - You did not get your full quota on demand; we did not hold slots for anyone in reserve
  - Going above quota is then on an opportunistic basis
- Exceedingly difficult to come up with an easy to manage quota system in the new GPGrid setup, especially with desired hierarchical functionality. Don't want everything subject to quota, especially offsite jobs
- So far for most of this year we have been operating without quotas, instead using *only the Condor priorities*
  - Easy to tune certain users when needed
  - Production accounts usually have a priority boost
  - We try to tune things so that total number of running jobs is usually around your allocation
    - **This is not guaranteed, however**

Condor Priorities

- Each user has a unique priority. Roughly speaking, *the higher the numerical value, the farther back in line*
- Amount of resources you're using, how long you've had them, and the "priority factor" assigned to your accounting group all influence your priority (and it is constantly changing)
  - For those interested: https://research.cs.wisc.edu/htcondor/manual/v8.4/3_4User_Priorities.html
- When people talk about a "priority boost" they usually mean the priority factor.

Today: how to get a priority boost

- Experiment Offline/Computing Coordinator should open a Service Desk Request and assign it to FIFE Support
  - **Requests directly from users will not be granted w/o approval**
- Should state the user(s) involved along with the expected duration of the boost
- Priorities of other users on the experiment may have to be reduced to keep overall experiment numbers close to allocation
  - If unwilling to do that, or request is for the entire experiment:
- For reqs that will on average raise the number of jobs running for the experiment, request must go through the SPPM Working Group. Meetings are Thursdays at 9 am; contact Margaret Votava to schedule

Back to the Future II (negotiators)

- Priority requests are fine, but it's extremely difficult to juggle more than a handful at once
- **Future plan is to return to quotas**, with other jobs (opportunistic and offsite) being handled separately
- Requires two Condor negotiators: first one matches jobs to quota slots on GPGrid, then allows jobs to come in if there are leftover slots. Second one deals with offsite jobs
- You would NOT need to submit quota and non-quota jobs separately
- Should get us where we collectively want to go
- Plan to deploy later in the summer, but likely not before ICHEP
  - If your experiment has been below your allocation for a while (hours) and you have on-site jobs queuing, open a ticket with FIFE Support

Priorities aren't the whole story

- Priority is part of determining which job will run next
- Don't forget the contribution of resource requests
  - Remember our restaurant examples
  - The next "free" slot might not match your request (though the system is trying to configure things all the time)
- Consider your reesource requests, look at the fifebatch slots page, or use jobsub_q --better-analyze --jobid=xxxxxx on the command line to get some insights into why your job isn't starting

Maxima and Large submissions

- What are the effective resource limits, if any?
  - Memory: 16000MB on GPGrid; varies on OSG sites
  - CPU: 8 on GPGrid; typically that or less on OSG
  - Run time: 4 days on GPGrid; varies on OSG sites
  - Contact FIFE if you have a workflow that has to go over these limits
- How many jobs can I submit?
  - Now: max in a single submission is 10K (but need to be careful with these.) **Considering dropping to 5K. Objections?**
    - Needs to be a significant amount of time between such submissions
    - Can also take a very long time to fetch logs (could easily be 50GB to tar and transfer!)
- How many can I (a user) have queued?
  - No hard limit, but O(100K) starts to get out of the ordinary

Summary

- We've gone over glideins briefly and explained how they're important
- We have looked at accurate job profiling and resource requests
- Given some hints for best practices with resource access
- Gone over priorities vs. quotas
- Discussed known limits and large submissions
- Following these best practices will help everyone get their work done better, **especially you!**
- Please talk to us if anything comes up of you have needs/ requests that weren't discussed here.