

# glideinWMS

## The OSG overlay DHTC system

Tuesday morning session

Igor Sfiligoi <[isfiligoi@ucsd.edu](mailto:isfiligoi@ucsd.edu)>

University of California San Diego



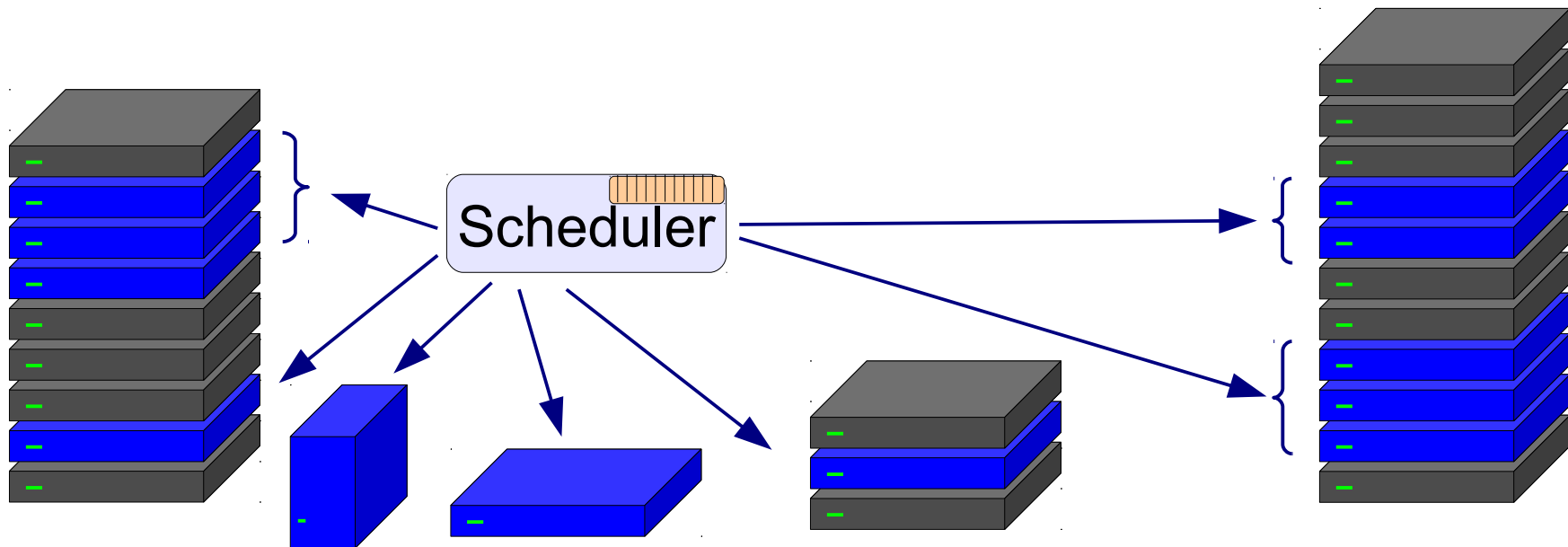
# Logistical reminder

---

- It is OK to ask questions
  - During the lecture
  - During the demos
  - During the exercises
  - During the breaks
- If I don't know the answer,  
I will find someone who likely does

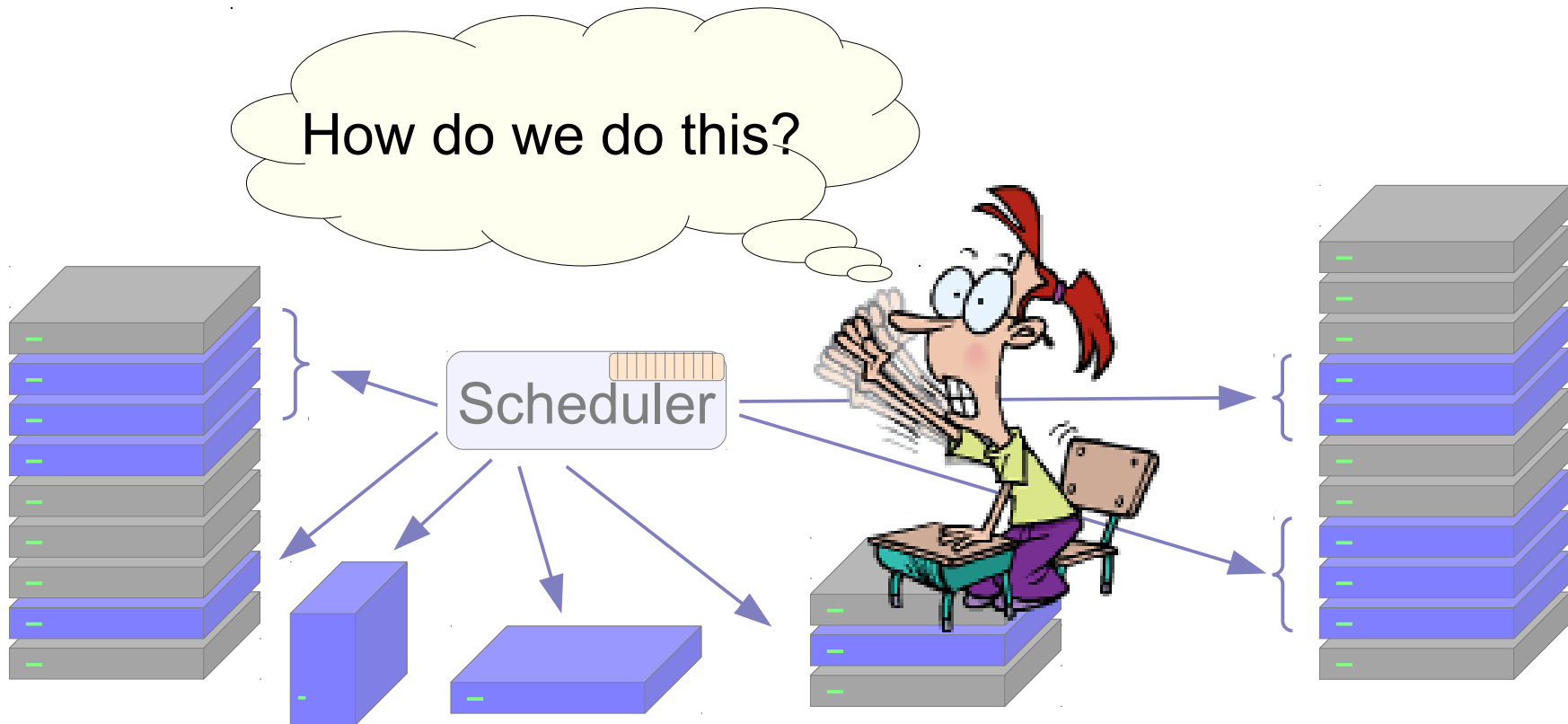
# Creating a dynamic overlay sys

- We can lease a subset of other's nodes
- And instantiate our HTC system on them



# Creating a dynamic overlay sys

- We can lease a subset of other's nodes
- And instantiate our HTC system on them



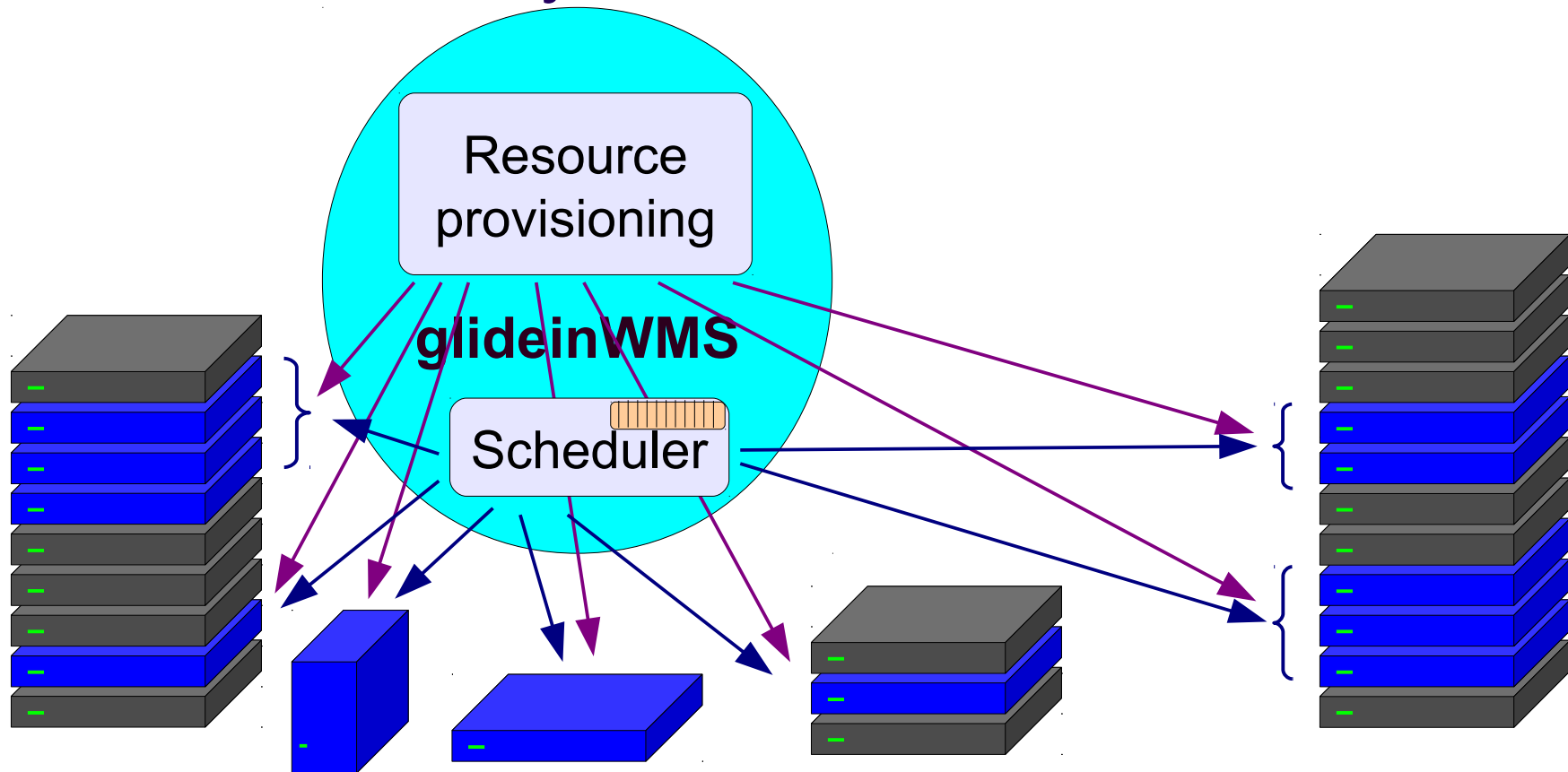
# Lecture content

---

- Brief overview of how the glideinWMS works
  - You hopefully will never need to deal with back-end details
  - But knowing them will help you troubleshoot problems with your jobs
- Some hands-on advise on how to use it

# glideinWMS as an overlay sys

- glideinWMS leases a subset of other's nodes
- Acts as a HTC system on them



# Two components

---

- The resource provisioning

# Two components

---

- The resource provisioning
  - Implements the logic
    - Decides when more resources are needed
    - Decides where to get them from
    - Decides when to get rid of them
  - Implements the technical bits
    - Grid resources (e.g. GRAM, CREAM, ARC)
    - Cloud (EC2, OpenStack, Google Engine)
    - BOSCO (i.e. HTC-over-ssh)



# Two components

- The resource provisioning
  - Implements the logic
    - Decides when resources are needed
    - Decides what resources are needed
    - Decides how resources are needed
  - Implements the resource provisioning
    - Grid resources (e.g. GPC, GPC, GPC)
    - Cloud (EC2, OpenStack)
    - BOSCO (i.e. MITC)

As users, you don't really need to know any more details.

But if you are interested in details we do have more training material.



# Two components

---

- The resource provisioning
- The HTC scheduler proper

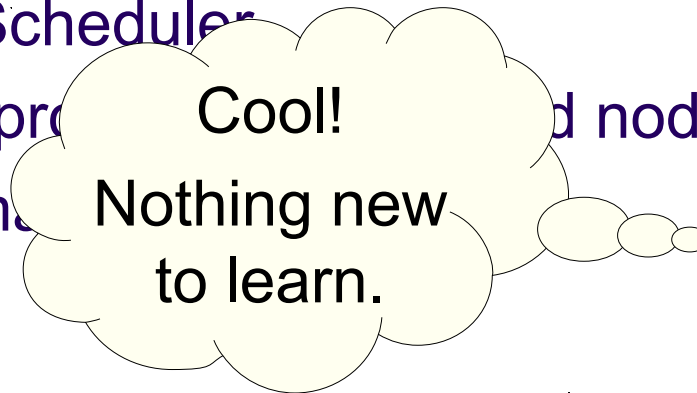
# Two components

---

- The resource provisioning
- The HTC scheduler proper
  - Which happens to be HTCondor!
  - HTCondor on all the nodes
    - User-facing Scheduler
    - The execute processes on leased nodes
    - The central manager

# Two components

- The resource provisioning
- The HTC scheduler proper
  - Which happens to be HTCondor!
  - HTCondor on all the nodes
    - User-facing Scheduler
    - The execute pro and nodes
    - The central ma



# Two components

- The resource provisioning
- The HTC scheduler proper
  - Which happens to be HTCondor!
  - HTCondor on all the nodes
    - User
    - The
    - The c

**Almost...**

Cool!

d nodes

thing new  
to learn.



# Steering the provisioning

---

- Your jobs will likely want to run only on a subset of possible resources, due to e.g.
  - Data locality
  - Platform restrictions
- The usual **requirements** attribute is not good enough
  - Attributes of the provisioned machines not known in advance

# Two level matchmaking

---

- The system has **two matchmaking points**
  - The glideinWMS decides **when and where** to provision resources
  - The HTCondor negotiator decides **which job runs on which node** (after the nodes have been provisioned)
- The two operate independently
  - You will need to provide information to both

# Standard convention

---

- The glideinWMS convention is for users to just publish the list of desired properties, e.g.
  - `+DESIRED_Sites="UCSD,UW"`
- The provisioning policy engine then does the right thing behind the scenes
- Please notice that there is no “standard glideinWMS policy”
  - See your local instance for details



# Runtime limits

---

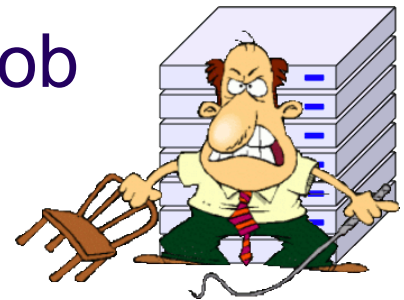
- Resource leases often come with **runtime limits** associated with them
  - In the Grid world, it is typical to be between 24h and 48h
- glideinWMS will **not start a job** on a resource that needs to be returned before your job finishes
  - Since it would have to **kill** your job on deadline, else



# Runtime limits

- Resource leases often come with **runtime limits** associated
  - In the Grid world, it is typical between 24h and 48h
- glideinWMS will **not start a job** on a resource that needs to be returned before your job finishes
  - Since it would have to **kill** your job on deadline, else

And how is it supposed to know how long will your job run?



# Runtime limits

---

- Resource leases often come with runtime limits associated with them
- glideinWMS will not start a job on a resource that needs to be returned before your job finishes
- You need to tell glideinWMS how long will the job run
  - As close to the worst-case as you can
  - But don't over-estimate
    - There may be very few resources willing to run jobs with long estimated runtimes

# Runtime limits

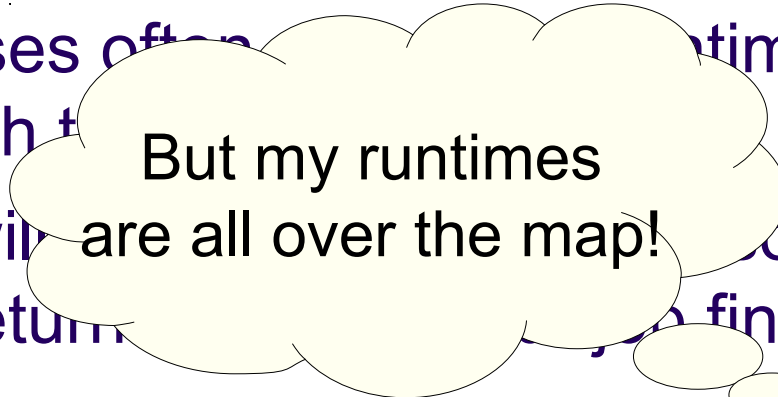
---

- Resource leases often come with runtime limits associated with them
- glideinWMS will not start a job on a resource that needs to be returned before your job finishes
- You need to tell glideinWMS how long will the job run
  - Once again, no standard way
    - See your local installation instructions
- Else, there is a system default
  - Which may not be appropriate for you!



# Runtime limits

- Resource leases often have runtime limits associated with them
- glideinWMS will find a resource that needs to be returned when the job finishes
- You need to tell glideinWMS how long will the job run
  - Once again, no standard way
    - See your local installation instructions
- Else, there is a system default





# Runtime limits

- Resource leases often have runtime limits associated with them
  - glideinWMS can't find a resource that needs it
  - You can't tell glideinWMS how long it will take to finish
- But my runtimes are over the map!
- Indeed, it may not be trivial.
- But it is needed.
- Once a resource is found, it's a way to get it
  - See the local instructions

Else, there is a system default



# Using glideinWMS

---

- Pretty much “just a HTCondor system”
- Use the standard commands
  - `condor_submit`
  - `condor_q`
  - `condor_rm`

# Using glideinWMS

---

- Pretty much “just a HTCondor system”
- Use the standard commands
  - `condor_submit`
  - `condor_q`
  - `condor_rm`
- Monitoring the system similar
  - `condor_status`





# Using glideinWMS

- Pretty much “just a HTCondor system”
- Use the standard condor commands
  - `condor_submit`
  - `condor_q`
  - `condor_rm`
- Monitoring the system similar
  - `condor_status`

But the number of slots grows and shrinks very often.



# Not-yet provisioned resources

---

- There may be resources available that don't have a single node provisioned yet
  - So `std. condor_status` will not show them

# Not-yet provisioned resources

- There may be resources available that don't have a single node provisioned yet
  - So std. **condor\_status** will not show them
- glideinWMS does publish the list
  - But no pretty tool available
  - Requires a bit of condor\_status magic

```
$ condor_status -any -const 'MyType=="glideresource"' -af Name \  
    -af GLIDEIN_Site -af GLIDEIN_Max_Walltime -af GLIDEIN_MaxMemMBs  
CMS_T2_US_UCSD_gw7@osg.edu.main          UCSD          171000 2500  
CMS_T2_US_Wisc_cms02@osg.edu.main        Wisconsin 114840 2500  
...
```

# Anything else?

---

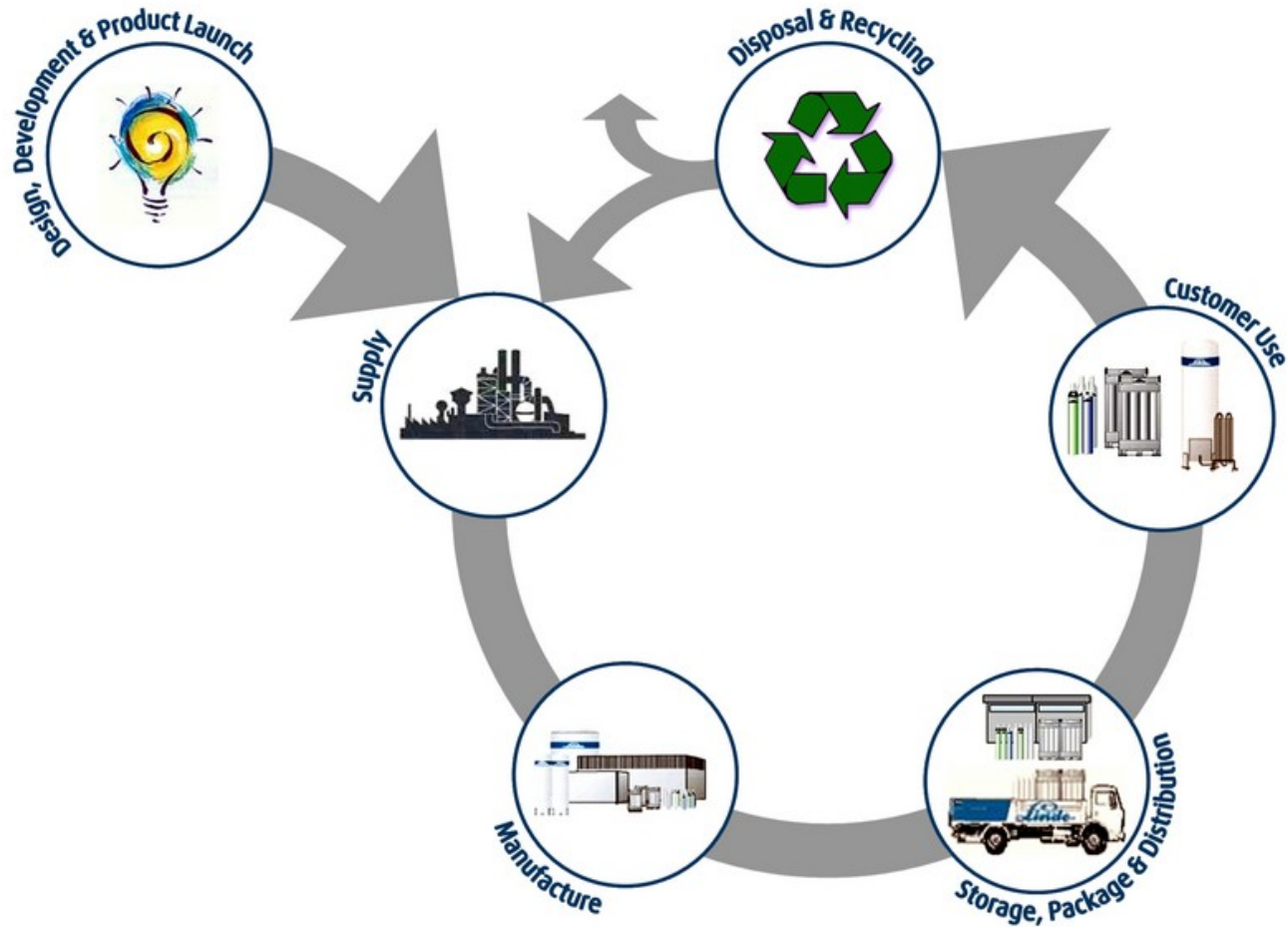
- Being a DHTC system, there is no shared file system
  - You will have to explicitly move files around
- Being a DHTC system, different sites likely have different libraries installed
  - Minimize dependencies
  - Be flexible

# Questions?

---

- Questions? Comments?
  - Feel free to ask me questions later:  
Igor Sfiligoi <isfiligoi@ucsd.edu>
- Upcoming sessions
  - Hands-on exercises
  - How to get the needed resources
  - Tour
  - Security in OSG

# Computing mimics real life



Courtesy of [peelscrapmetalrecycling.com](http://peelscrapmetalrecycling.com)

# Copyright statement

---

- Some images contained in this presentation are the copyrighted property of ToonClipart.
- As such, these images are being used under a license from said entities, and may not be copied or downloaded without explicit permission from ToonClipart.