HPC Symposium Fall 2022

John Huffman, Director, IT Research CyberInfrastructure (IT-RCI)
Jeff Frey, Michael Kyle, Anita Schwartz & Olena Smith

GIS Day 2022

- *Date*: Wed., Nov. 16, 2022
- *Time*: 9AM - 1:30PM (welcome to join for any part of the event)
- *Location*: Roselle Center for the Arts (Gore Recital Hall & Lobby)

See https://sites.udel.edu/gisday/ for more information and registration

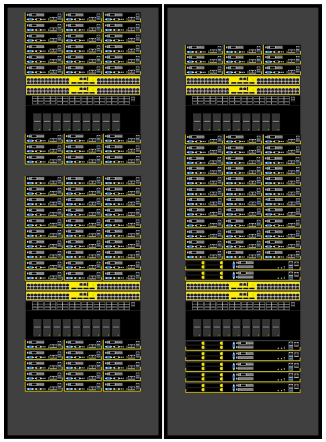
Agenda

- Caviness Expansion: Generation 3 update
- Data Center Issues and CI updates
- Farber EOL Plan and Policies Announcement
- CAREERS Recruiting for Projects
- ACCESS Successor to XSEDE
- DARWIN Allocation Accounting (sproject)
- Open discussion

Caviness Expansion: Generation 3 Update

High Performance Computing Updates

+

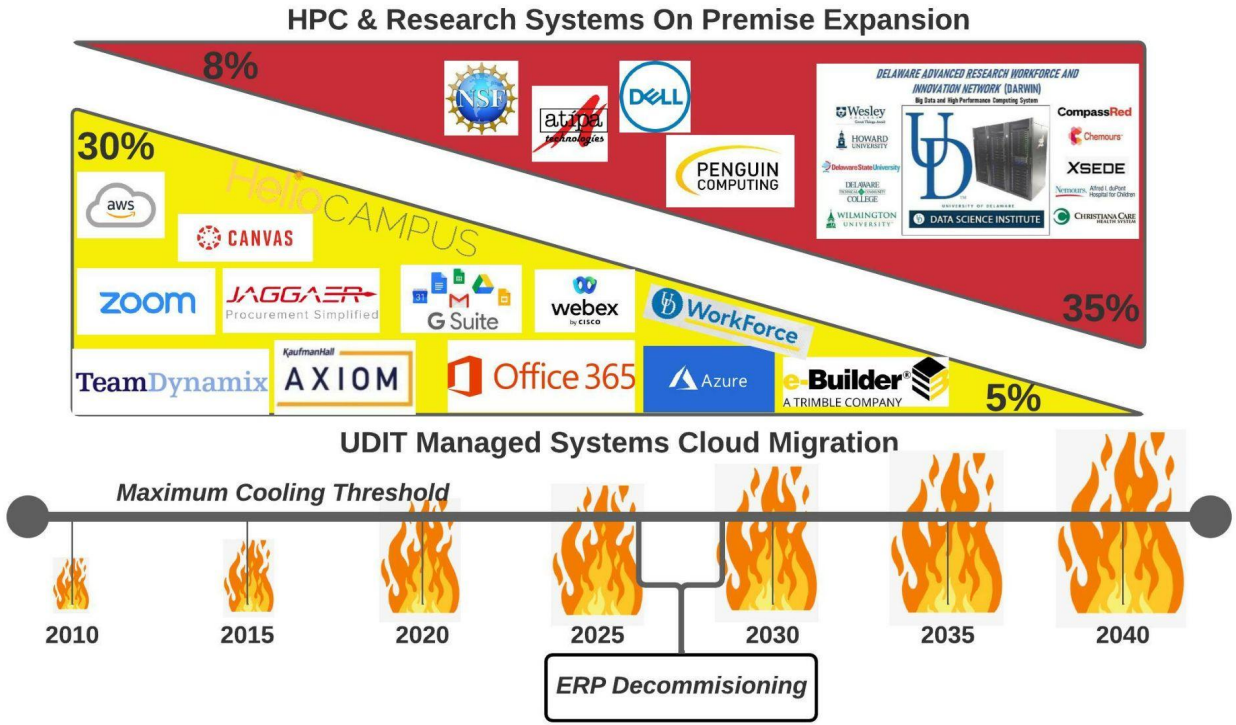


- 272 compute nodes
  - +94 nodes
- 10,376 cores
  - + 4,672 cores
- 102 TiB RAM
  - + 43 TiB of RAM
- 18 GPU nodes
  - +7 GPU nodes
- Petabyte of Storage

Existing Caviness HPC system

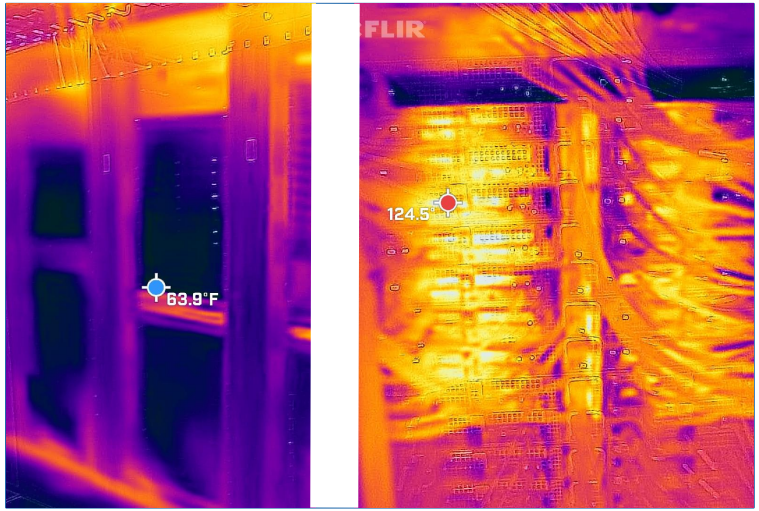
2022 Caviness HPC Expansion

Future of Chapel Computing Center



What's generating the heat?

Darwin cluster
Intake & Exhaust Temperatures



New research servers consume 2.5x power & generate significantly more heat.

Intake temp for servers:

63-68°

Exhaust temp for research servers:

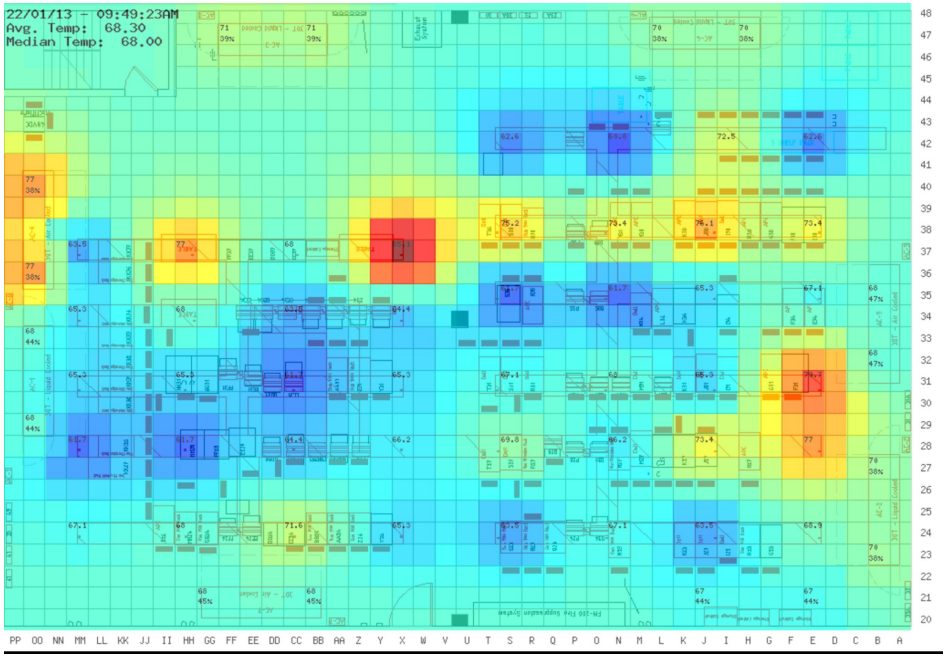
120°+

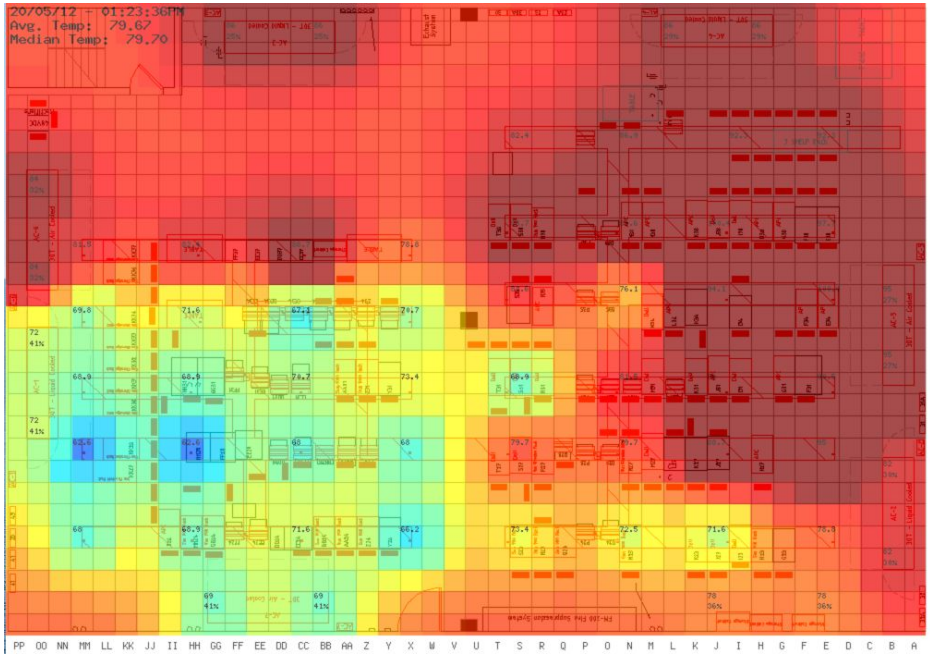
Exhaust temp for standard servers:

85-90°

Heat Map of Data Center During CRAC Failure

If one CRAC fails, Data Center room temperature increases by 10 degrees in 10 minutes.





Data Center Issues

Chapel Computing Center Overheating

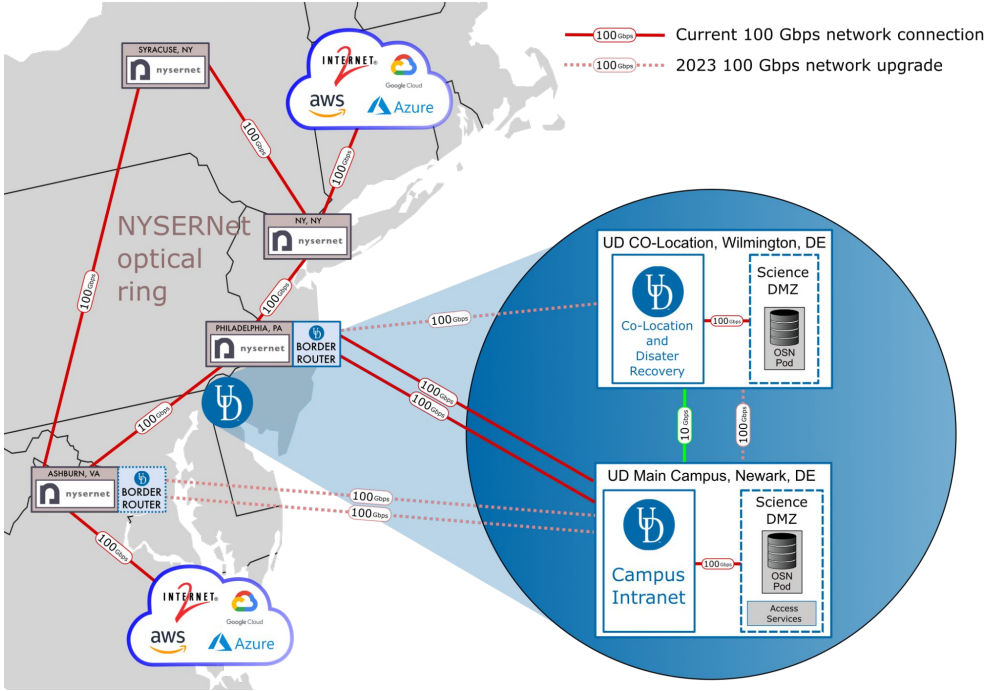
- ***Current cooling and mitigation strategies are no longer effective in managing heat output from high density research compute servers.***

- ***UDIT initiated a facilities project to upgrade Chapel Computing Center's cooling capacity to keep pace with the expansion of research compute systems.***

Cyberinfrastructure

Network and Storage

- ***UDIT is establishing a new 100 Gbps connection to Ashburn, VA, through NYSERNet***
- ***A new Co-Location site is being established in Wilmington, DE***
- ***UDIT is working to bring multi-petabyte storage to support research and scholarly activities***



Farber EOL Plan and Policies Announcement

January 2023: All compute nodes will be shut down and the job scheduler taken offline. No updates to OS or other software maintained by IT, no new accounts created and limited support

July 2023: Hardware will be removed from the Datacenter

IT has no plans to mass-migrate any of the data present in /home/work or in home directories to longer-term storage systems.

https://docs.google.com/document/d/1A1ke9SZnQMiqAKmRfRI0-LtRBwcH9_Gs6rCAcH1IntA/

CAREERS Recruiting for Projects

Cyberteam to **A**dvance **R**esearch and **E**ducation in **E**astern **R**egional **S**chools

1 year left on grant with funding and 1 year no extension

Supporting Research at Smaller Institutions with Student Research Computing Facilitators (RCFs)

UNIVERSITY OF DELAWARE

RUTGERS

Rensselaer

PennState

THE UNIVERSITY OF RHODE ISLAND

Yale University

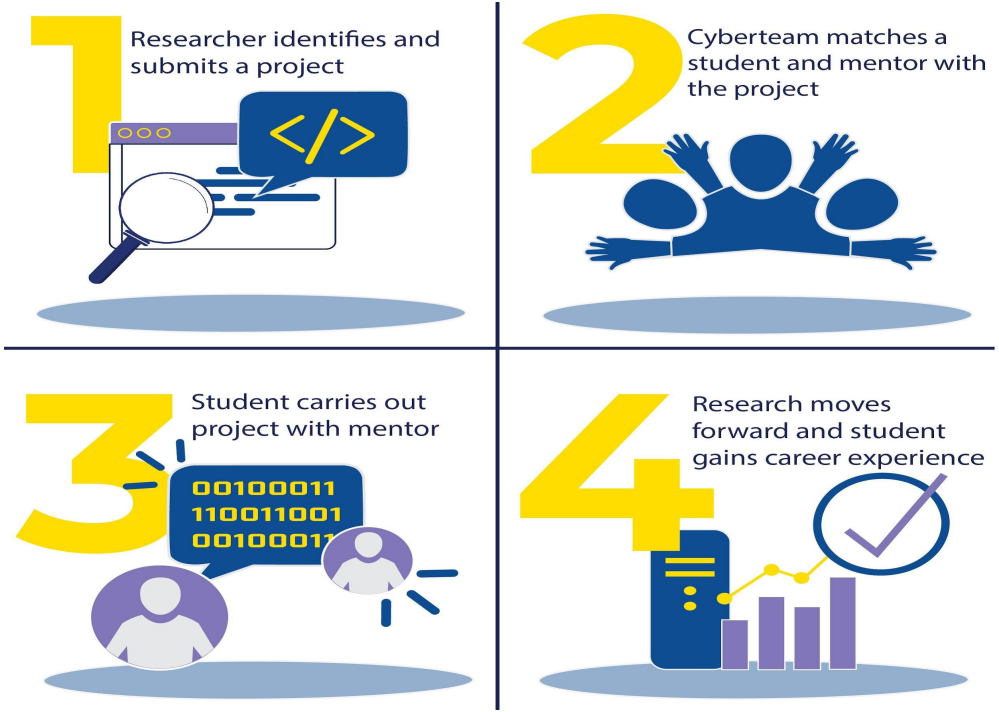
Cyberteam Projects

Students provide project updates at our monthly meetings

- launch presentation introducing the project
- monthly updates
- wrap presentation of completed project

Project completes at a clearly defined end point

- Code is hosted in organizational GitHub
- Opportunity and funding to present research at national meetings or conferences



Cyberteam Projects

- Projects are typically 3 or 6 months in length with well-defined milestones
- Student facilitators receive stipends unless other compensation has been arranged (such as course credit)
  - $3000-$6000 depending on grad vs undergrad and the length of the project

ACCESS Successor to XSEDE

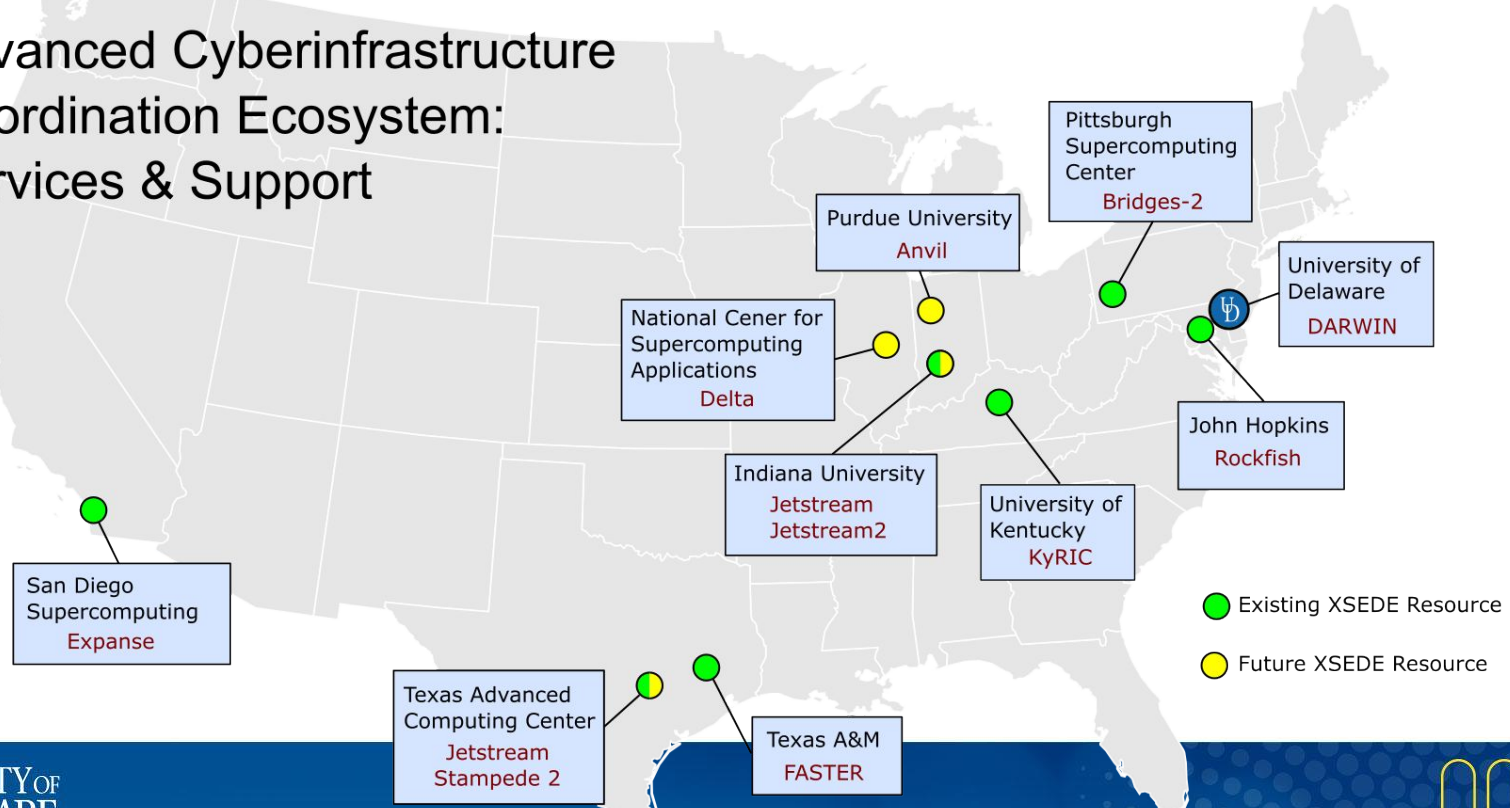
- Account
  - Your ACCESS ID is the same as your XSEDE Portal account. Please do not create a new ACCESS ID.
  - You do not need to change your password or your Duo registration for ACCESS.
  - Select the “ACCESS CI (XSEDE)” identity provider to log on using your XSEDE account.
  - Avoid creating duplicate accounts.
- Allocations
  - Explore ACCESS, Discover ACCESS, Accelerate ACCESS, Maximize ACCESS
  - UD Startup, UD Education, UD Research

ACCESS | Advancing Innovation

Multi Institutional Resources

- Open Science Grid
- Open Storage Network

Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support



ACCESS is an advanced computing and data resource supported by the National Science Foundation and made possible through these lead institutions and their partners – Carnegie Mellon University; University of Colorado Boulder; University of Illinois at Urbana-Champaign; and State University of New York at Buffalo.

DARWIN Allocation Accounting (sproject)

Presentation about the implementation of allocation accounting on DARWIN and tools available to users to check their allocation status.

Hermes Allocation Data Manager (HADM)

Implementing Allocation Accounting on DARWIN

Presented by Jeff Frey, IT RCI Systems Administrator

Goals

- Keep track of resource allocations made by DSI and ACCESS (was XSEDE)
  - CREDITS
- Tabulate resource usage by jobs run on DARWIN
  - DEBITS
- Deny jobs that exceed remaining CREDIT on an allocation
- Provide tools for users and ACCESS agents to query allocation status

What is an Allocation?

- A time period
  - Usually 1 year

What is an Allocation?

- A time period
- A category of allocation
  - Dictates upper bounds on the requestable resources
  - XSEDE and DSI: **education**, **startup**, **research**
  - ACCESS: **explore**, **discover**, **accelerate**, **maximize** (we don't offer this one)

What is an Allocation?

- A time period
- A category of allocation
- Credit applied to one or more consumable
  - CPU•time : a *service unit* consists of havin
  - GPU•time : a *service unit* consists of havin
  - GiB•time : a *service unit* consists of 1 GiB c
    - By default an allocation gets 500 C

::: **CPU vs. MEMORY** :::::::::::::::::::::::::::::::::::::

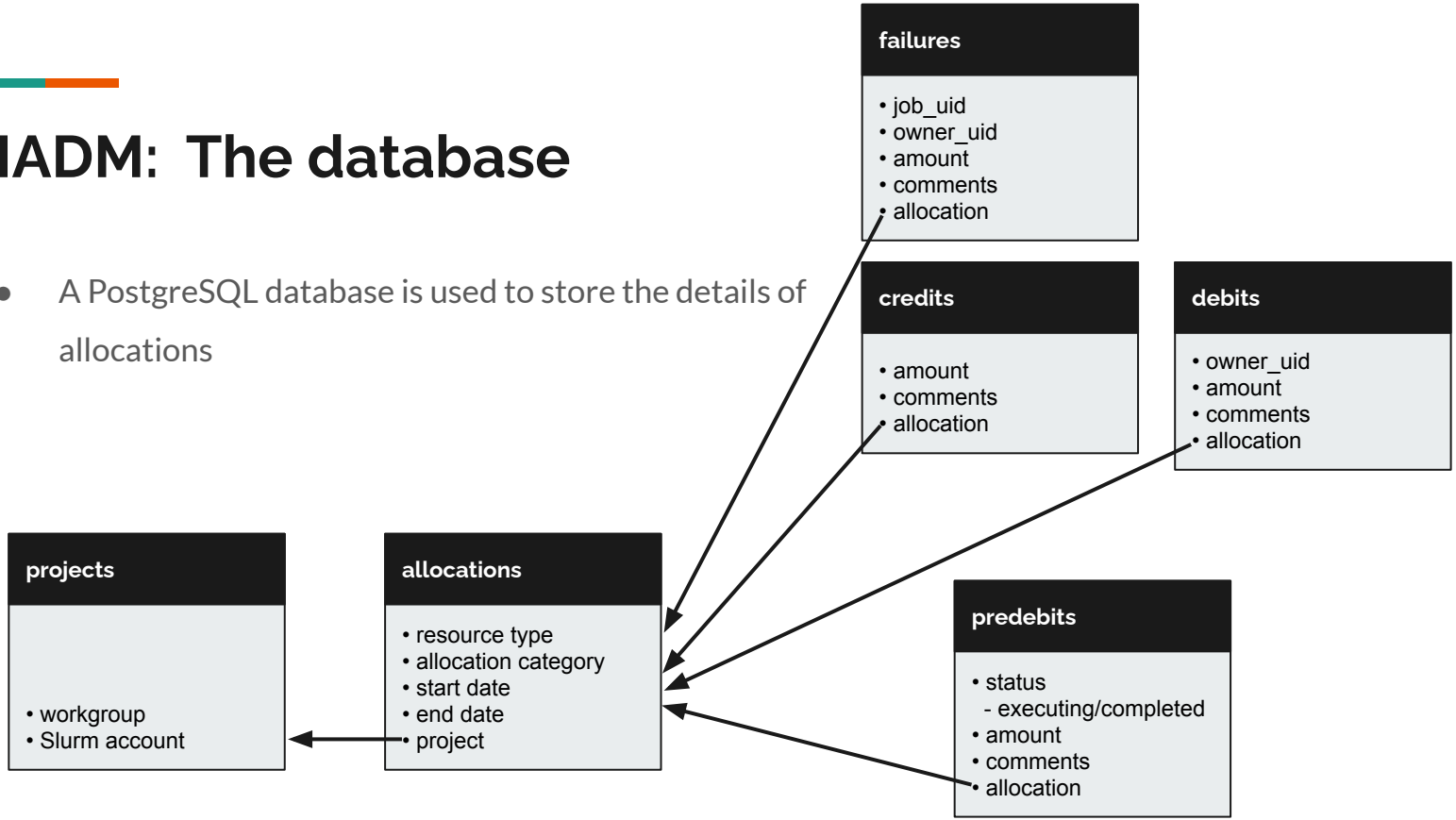
Each type of node in DARWIN contains *N* CPU cores and *M* bytes of memory. The fraction *(M/N)* is the amount of memory per CPU core. Consider a job that requests 1 CPU core and memory ***in excess*** of *(M/N)*: this job is using the memory resources of more than 1 CPU. For this reason, Slurm is configured to use whichever is greater: the requested CPU count, or the number of CPUs associated with the requested memory.

What is an Allocation?

- A time period
- A category of allocation
- Credit applied to one or more consumable computational resources
- In any period of time, a workgroup on DARWIN can have:
  - A *single* category of allocation: cannot have a **research** allocation and an **education** allocation concurrently
  - Credits to: CPU, GPU, CPU+GPU, CPU+storage, GPU+storage, CPU+GPU+storage

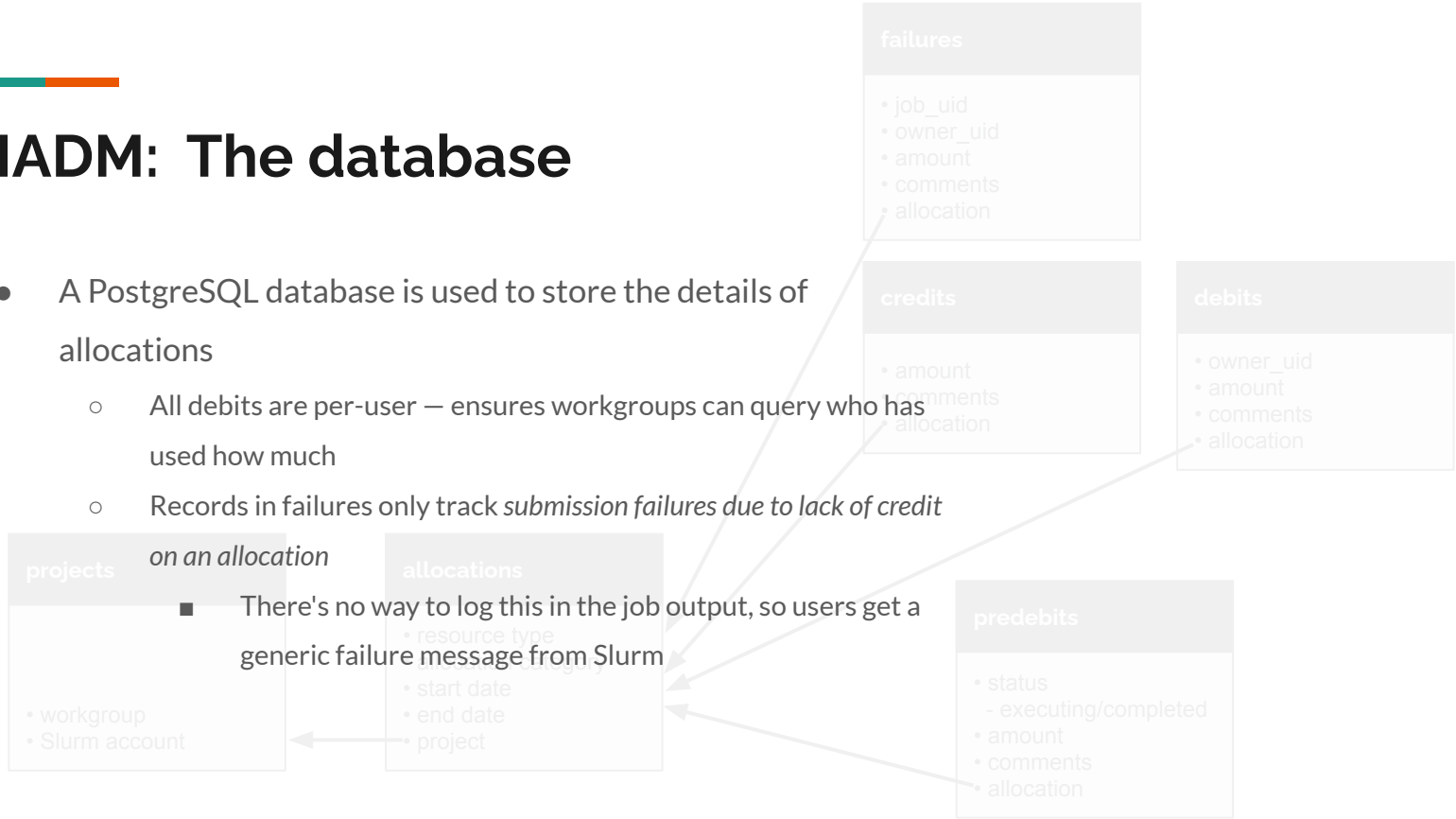
HADM: The database

- A PostgreSQL database is used to store the details of allocations



HADM: The database

- A PostgreSQL database is used to store the details of allocations
  - All debits are per-user — ensures workgroups can query who has used how much
  - Records in failures only track *submission failures due to lack of credit on an allocation*
    - There's no way to log this in the job output, so users get a generic failure message from Slurm



HADM: The administrative interface

- PostgreSQL client
  - Direct access to database

HADM: The administrative interface

- PostgreSQL client
- Web API
  - Python web application (Flask, GUnicorn) that provides REST access to the database
  - Authentication of requests via MUNGE (same message-signing used by Slurm)
  - Three roles:
    - superuser: Create, Read, Update, and Delete all records of all types, run as alternative uid
    - admin: Read all records of all types, run as alternative uid
    - standard: Read records associated with user's workgroup(s)

HADM: The administrative interface

- PostgreSQL client
- Web API
- hadm-curl
    - Streamlines queries against the Web API
    - Automatically adds the MUNGE authentication HTTP header, run-as header (for superuser or admin)

HADM: Slurm integration

- Ideally, we would like:
  - job submissions to be checked for available allocation credit given *projected resource consumption*
    - if the job will require 6 SU and the allocation has just 5 SU remaining, deny the job
  - pending jobs' *projected resource consumption* to influence additional job submissions without technically being a debit (these are *predebit* records in the database)
    - if a pending job is going to leave only 5 SU, a submission requesting 6 SU should not be allowed
  - completed jobs to debit their *actual resource consumption* from the allocation (these are *debit* records in the database)

HADM: Slurm integration

- Slurm has limited support for introducing consumable resource limits into its job accounting
- Use of job submission plugins or prolog scripts to verify sufficient credit vs. *projected resource consumption*
  - At submission the job has no assigned job id — no way to supply it to the HADM database for a *predebit*
  - Prolog scripts in slurmd have time limits and would require all compute nodes have access to HADM
- Batch processing of job completion logs or use of epilog scripts to generate *debits*
  - Can affect the accuracy of credit verification based on frequency of batch processing
  - Epilog scripts in slurmd have time limits, would require all compute nodes have access to HADM, and would not have access to *actual resource consumption* data for the job

HADM: Slurm integration

- We chose to use a Slurm PrEp plugin
  - Plugin interface allowing C functions to be executed during job **Pr**olog or **Ep**ilog stages
  - The functions can execute asynchronously (in another thread) so the rest of the scheduler is not blocked

HADM: Slurm integration

- We chose to use a Slurm PrEp plugin
- Prolog function: before a job is handed-of
  - Determine resource type from partition
    - **REQUIREMENT: jobs MUST be su**
  - Allocation to be predebited inferred from
    - **REQUIREMENT: jobs MUST be su**
    - account is inferred from submittin
  - Job record contains calculated billable TR
    - **REQUIREMENT: jobs MUST be su**
  - Asynchronously submit a request to the HADM WebAPI with the jobid and parameters summarized above

: : : **PLEASE NOTE** :

At this point the job's stdout and stderr file(s) are **not** available. If the Web API returns a failure to submit — due to insufficient credit on the allocation — then the accounting record for the job will show an error of ESLURM_ACCOUNTING_POLICY for the job. The HADM database will have logged a record in the failures table which the user can check using the sproject utility (discussed next).

HADM: Slurm integration

- We chose to use a Slurm PrEp plugin
- Prolog function: before a job is handed-off to a slurmd to begin execution
- Epilog function: before job completes and is finalized in the accounting database…
  - If the job failed due to a node's being down or never having booted, zero the usage — no charge!
  - Otherwise, calculate *actual resource consumption* (a real number, not an integer)
    - To avoid erroneous round-up, limit the value to 10 significant digits
    - Ensure a minimum value of 1.0 so long as the job accumulated non-zero run time
  - Asynchronously submit a request to the HADM WebAPI with the jobid and *actual resource consumption*

HADM: Slurm integration

- We chose to use a Slurm PrEp plugin
- Prolog function: before a job is handed-off to a slurmd to begin execution
- Epilog function: before job completes and is finalized in the accounting database
- Web API query throughput
    - Jobs are constantly being submitted and completing: sometimes concurrently, so throughput is critical
        - The Web API application is parallelized, so > 1 request can be handled at once
        - The Slurm prolog/epilog requests are already asynchronous, so why not parallelize them?
    - Configurable, limited-lifespan worker thread model for Web API requests in Slurm
        - Min/max thread counts, max request/Δt lifespan for thread, load-based addition of more threads

HADM: Standard user interface

- sproject
  - Collects parameters via command-line options provided by the user
  - Generates the Web API requests for those parameters
  - Formats the returned JSON data for human-consumable display
    - table, comma-separated value (CSV), or raw JSON
    - substitutes names for object ids for the sake of clarity (workgroup name instead of project id)

HADM: sproject usage

```
# Show me all allocations on record for my project:

$ sproject allocations --project=it_nss
Project id Alloc id Alloc descr Category RDR Start date                End date
---------- -------- ----------- -------- --- ------------------------ ------------------------
         1        1 it_nss::cpu startup  cpu 2021-07-12 00:00:00-04:00 2021-07-25 23:59:59-04:00
         1        2 it_nss::gpu startup  gpu 2021-07-12 00:00:00-04:00 2021-07-25 23:59:59-04:00
         1       41 it_nss::cpu startup  cpu 2021-07-26 00:00:00-04:00 2023-07-31 00:00:00-04:00
         1       42 it_nss::gpu startup  gpu 2021-07-26 00:00:00-04:00 2023-07-31 00:00:00-04:00


# Limit that to only those allocations that are currently active for job submissions:

$ sproject allocations --project=it_nss --current-only
Project id Alloc id Alloc descr Category RDR Start date                End date
---------- -------- ----------- -------- --- ------------------------ ------------------------
         1       41 it_nss::cpu startup  cpu 2021-07-26 00:00:00-04:00 2023-07-31 00:00:00-04:00
         1       42 it_nss::gpu startup  gpu 2021-07-26 00:00:00-04:00 2023-07-31 00:00:00-04:00
```

HADM: sproject usage

```
$ sproject allocations --project=it_nss --current-only --detail
Project id Alloc id Alloc descr Category RDR Credit Run+Cmplt Debit Balance
---------- -------- ----------- -------- --- ------ --------- ----- -------
         1       41 it_nss::cpu startup  cpu  33333         0  -855   32479
         1       42 it_nss::gpu startup  gpu  33333         0    -4   33329


# Since DSI and ACCESS allocations are in units of hours, the default display is
# in that unit.  HADM internally maintains balances in units of minutes:


$ sproject --exact allocations --project=it_nss --current-only --detail
Project id Alloc id Alloc descr Category RDR  Credit Run+Cmplt  Debit Balance
---------- -------- ----------- -------- --- ------- --------- ------ -------
         1       41 it_nss::cpu startup  cpu 2000000         0 -51281 1948719
         1       42 it_nss::gpu startup  gpu 2000000         0   -240 1999760
```

HADM: sproject usage

```
# By default the aggregate debits are displayed; but since debits are tracked per-user
# they can be displayed instead:

$ sproject allocations --project=it_nss --current-only --by-user
Project id Alloc id Alloc descr Category RDR User Transaction Amount
---------- -------- ----------- -------- --- ---- ----------- ------
         1       41 it_nss::cpu startup  cpu frey debit         -854
                 41 it_nss::cpu                 debit           -1
                 41 it_nss::cpu                 credit       33333
         1       42 it_nss::gpu startup  gpu      credit       33333
                 42 it_nss::gpu              frey debit           -4
```

HADM: sproject usage

```
$ sproject jobs --project=xg-phy150040
Activity id Alloc id Alloc descr        Job id Owner      Status    Amount Modification date         Creation date
----------- -------- ----------------- ------- ---------- --------- ------ ------------------------ ------------------------
    4349218      369 xg-phy150040::gpu 3353939 xsedeu2798 executing    -24                          2022-11-02 12:26:04-04:00
    4349219      369 xg-phy150040::gpu 3353941 xsedeu2798 executing    -24                          2022-11-02 12:26:07-04:00
    4349017      369 xg-phy150040::gpu 3353927 xsedeu2798 completed    -24 2022-11-02 12:26:03-04:00 2022-11-01 12:45:30-04:00

$ sacct --job=3353939 --format=alloctres,time -pXn
billing=1,cpu=2,gres/gpu:tesla_t4=1,gres/gpu=1,mem=5G,node=1|1-00:00:00|


# Job 3353939 has a "billing" value of 1 coming from the 1 GPU requested.
# It has a maximum run time of 1 day = 24 hours.
# The projected resource consumption is thus 24 GPU•hour and it is currently executing.


$ sacct --job=3353927 --format=alloctres,time -pXn
billing=1,cpu=2,gres/gpu:tesla_t4=1,gres/gpu=1,mem=5G,node=1|1-00:00:00|

# Job 3353927 has already finished executing and had the same resources.
# Its actual resource consumption was 24 GPU•hour and is waiting to be finalized (from predebit to debit record).
```

HADM: sproject usage

```
# My job (with id 3165763) failed to execute and Slurm says it was cancelled by root:

$ sacct --job=3165763 --format=jobid,account,state,exitcode -pX
JobID|Account|State|ExitCode|
3165763|biophysics|CANCELLED by 0|0:0|


# Maybe it was due to a lack of credit on my allocation; let me check:

$ sproject failures --job=3165763
 Job id Error message
------- ------------------------------------------------------------
3165763 Requested allocation has insufficient balance: 290 < 360
```

HADM: Post-production issues

- Submission race conditions
  - Since prolog requests are parallelized, what happens when multiple jobs submitted against the same allocation are processed concurrently?
  - Assume 4 requests for 5 SU each and allocation has 10 SU of credit
  - This yields 4 concurrent *SELECT* queries to determine if sufficient credit exists
    - All four queries show 5 SU ≤ 10 SU, thus they are all okayed
  - *SELECT* queries are not serialized by default in PostgreSQL: all 4 clients have a shared lock on the table

HADM: Post-production issues

- Submission race conditions
  - Credit validation and initial predebit in SQL stored procedure must be serialized to avoid overdrafts!
  - Initially tried using an SQL exclusive lock on the tables.
    - The release of the lock is not controllable by software — PostgreSQL server handles it as part of transaction block *COMMIT*/*ROLLBACK*.
    - Under high job submission load, there would be cyclic contention for the exclusive lock that effectively deadlocked the PostgreSQL server.

HADM: Post-production issues

- Submission race conditions
  - Credit validation and initial predebit in SQL stored procedure must be serialized to avoid overdrafts!
  - Initially tried using an SQL exclusive lock on the tables.
  - No more SQL locks, added software-controllable advisory lock around the critical region in the SQL stored procedure.
    - Success…so far!
    - Same advisory lock around epilog queries, as well: stored procedure must serialize the combination of *INSERT* into debits table and *DELETE* from predebits table

Questions?

- HADM software is available online at https://gitlab.com/udel-itrci/hadm

Open Discussion