High Throughput Workflow Tools and Strategies

Part 3: HyperShell

Geoffrey Lentner
Lead Research Data Scientist
Rosen Center for Advanced Computing

PURDUE UNIVERSITY
Rosen Center for
Advanced Computing

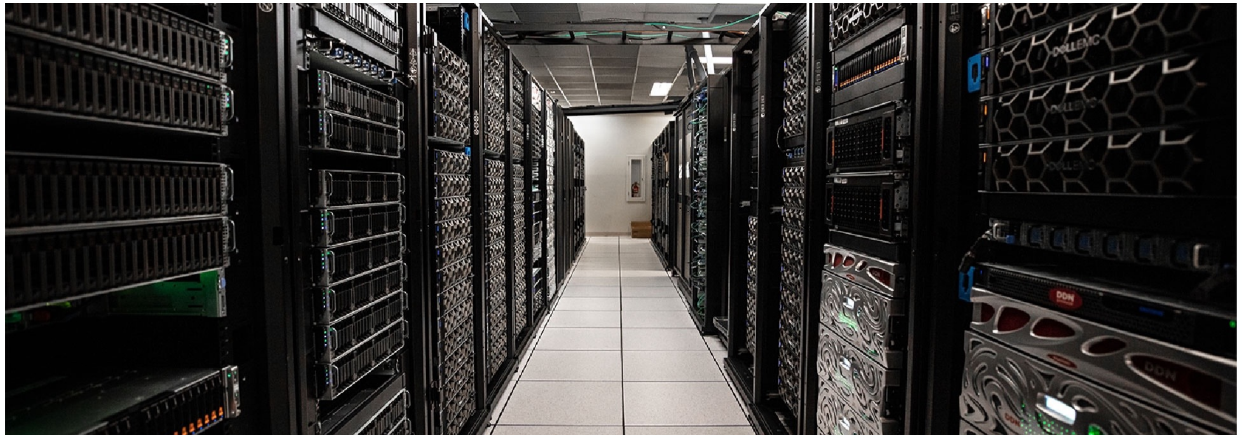
1

Preface

About me, background, chapter goals

Rosen Center for Advanced Computing

Research computing and data services at Purdue University



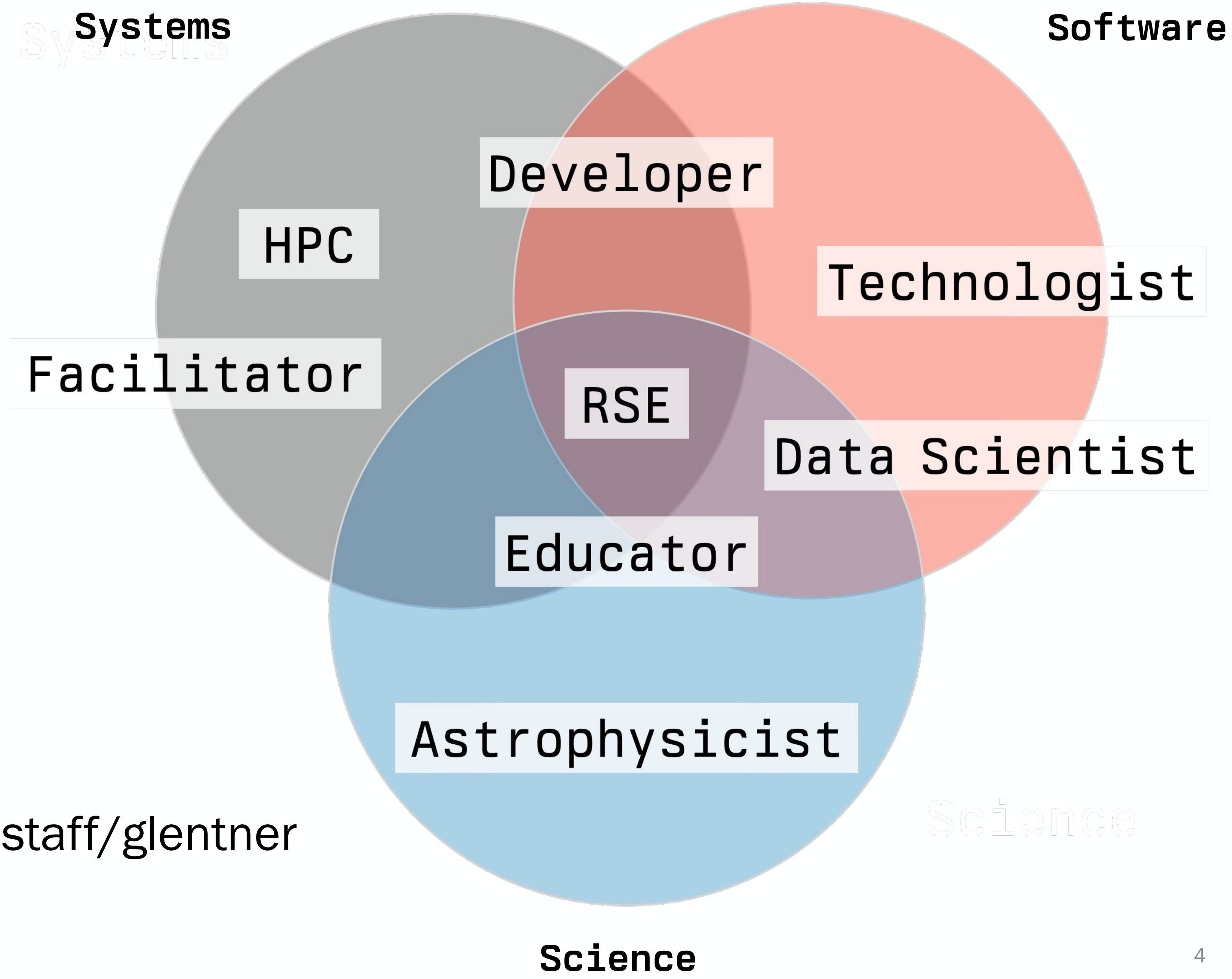
- *RCAC* for short, Part of ***Purdue IT***
- Operate *Top500* supercomputers, including PB-scale storage systems.
- Scientific Applications
- Envision Center
- Research Software Engineering
- Visit rcac.purdue.edu

About Me

Background and Roles

Geoffrey Lentner

Email: glentner@purdue.edu
Page: rcac.purdue.edu/about/staff/glentner



Goals

What I hope to accomplish

- Speak on high-throughput computing issues.
- HyperShell features relative to other tools.
- Demonstrate various scenarios.

2

Overview

Motivation; TLDR; project overview.

Why use a workflow tool at all?

Why not use the main scheduler?

Practical Limits

Site administrators do not want users ...

- *submitting millions of jobs,*
- *filling up the database,*
- *impacting site-wide throughput,*
- *polluting the queue.*

Users don’t want to be rate limited by the main schedulers bandwidth,

Or lose their job history over time.

…

User: *I want to run my job a million times…*

Me: *Um… About that…*

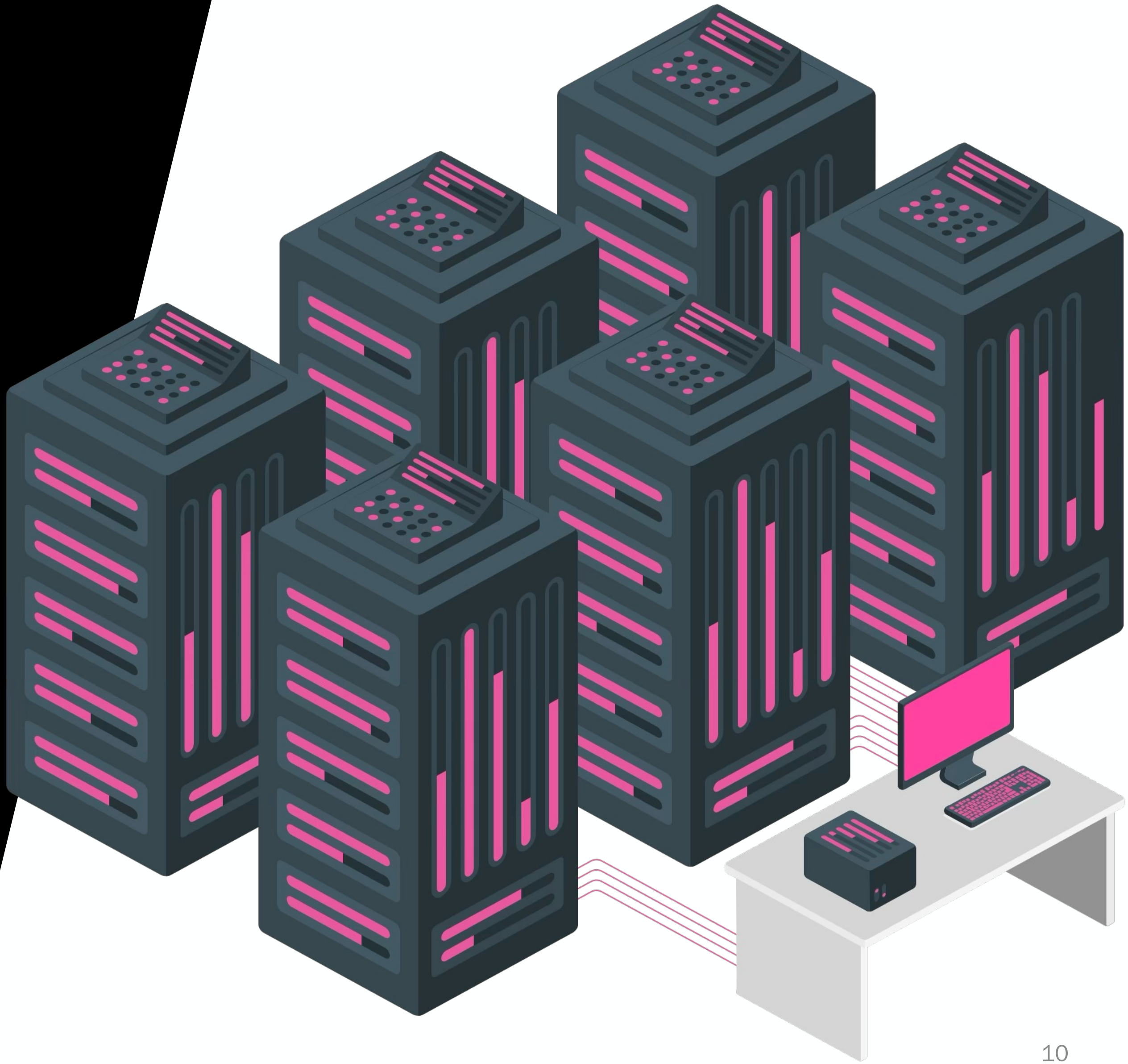
HyperShell 2.7.0

The ultimate workflow automation tool

An elegant, cross-platform *high-throughput computing* utility for processing shell commands over a distributed, asynchronous queue.

A highly scalable workflow automation tool for many-task scenarios.

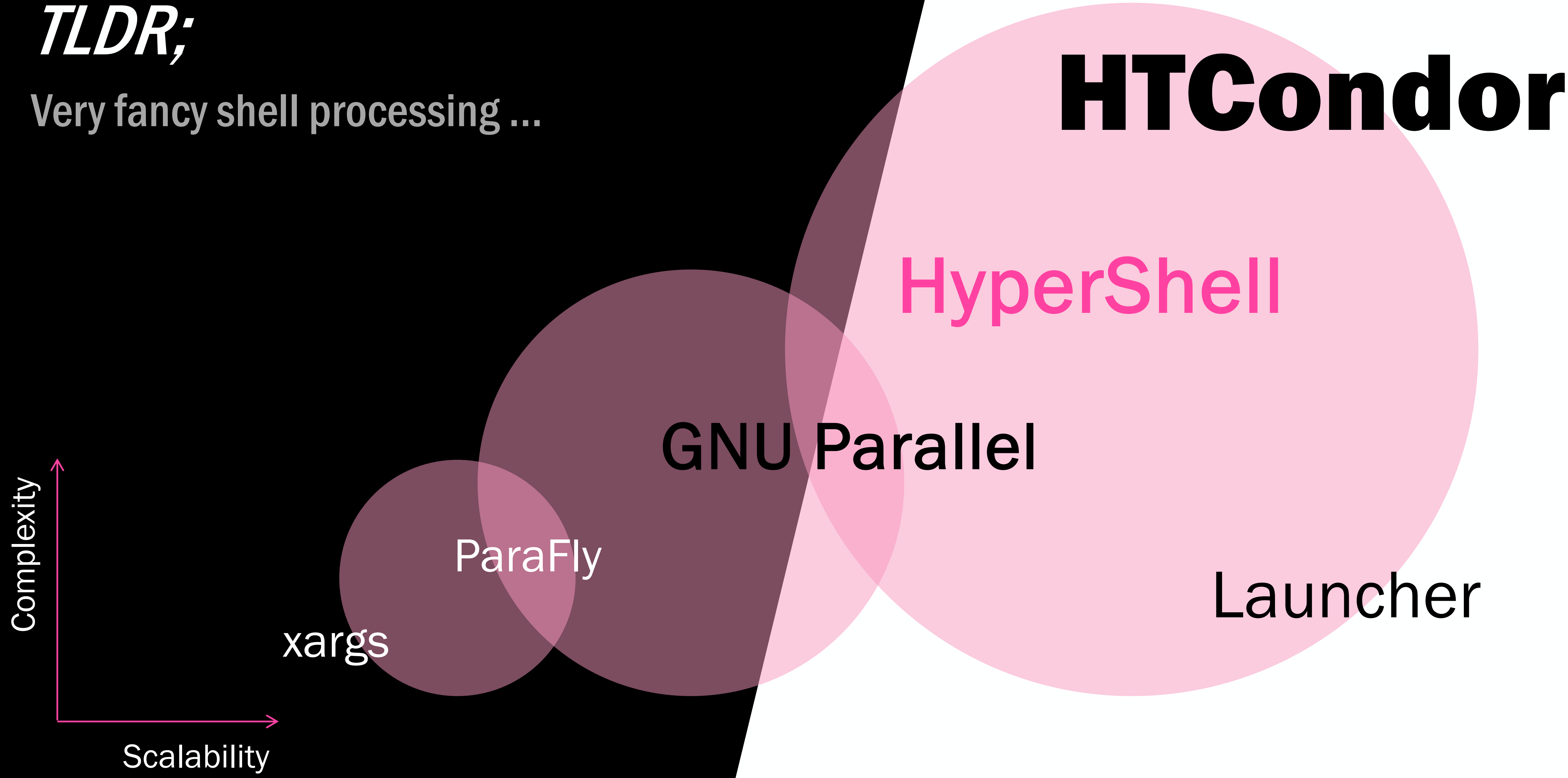
Windows Mac Linux python



But why a new tool?

TLDR;

Very fancy shell processing ...



Rosen Center for Advanced Computing

TLDR;

Combine the best features from tools in this genre

- Scale farther with task *aggregation*,
- Manage tasks with persistent *database*,
- Elastic (client-server) with *scale-to-zero*,
- *Cross-platform* (Linux server, Windows clients),
- Better *ergonomics* for researchers,
- Embed as a *library* in your project.

TLDR;

Online resources

- *Code* at github.com/hypershell/hypershell,
- *Docs* at hypershell.readthedocs.io,
- *Web* at hypershell.org, (under construction)
- *Discord* at discord.gg/wmv5gyUfkN.

Getting Started

Installing as an unprivileged user (any platform)

See installation guide for details.
Optional dependency on *PostgreSQL* – can use *Anaconda* if necessary.

Install HyperShell on any platform

```
$ pip install hypershell
$ pip install 'hypershell[postgres]'

$ uv tool install 'hypershell[postgres]' --python 3.13
```

Install “uv” from https://docs.astral.sh/uv/getting-started/installation/

Getting Started

Installing on MacOS

Local installation on *MacOS* with *Homebrew*

Install HyperShell on macOS

```
$ brew tap hypershell/tap
$ brew install hypershell
```

Getting Started

Alternative installations

docker

AlmaLinux

APPTAINER

Ubuntu

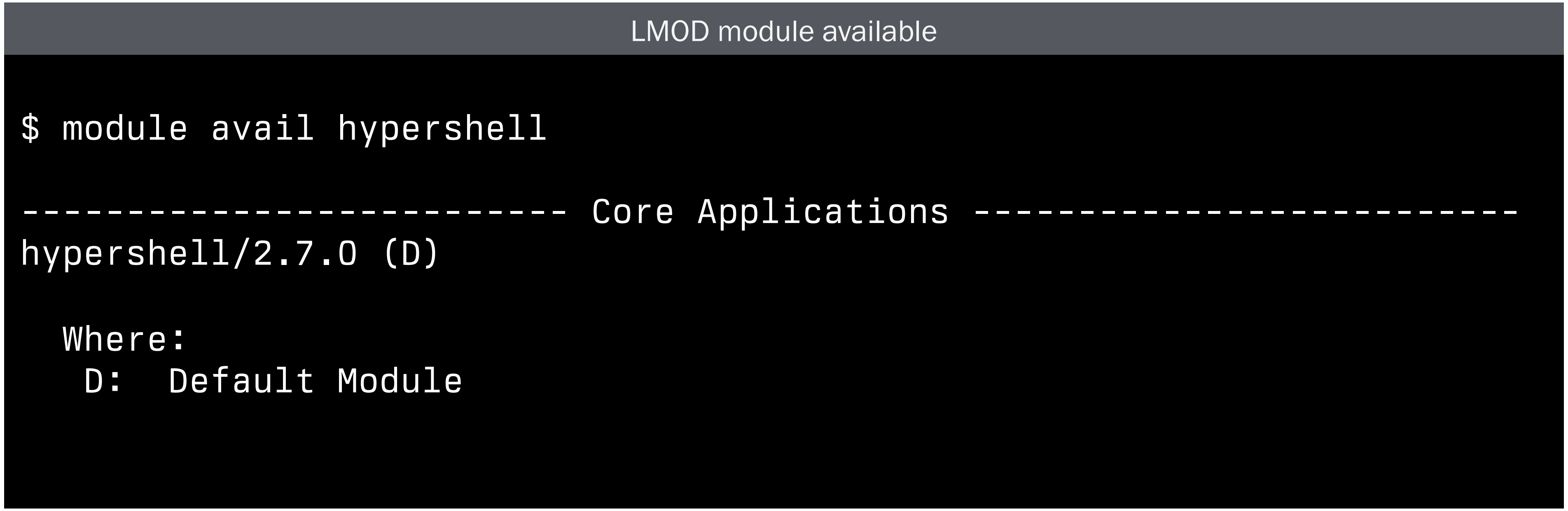
```
 dnf install hypershell  (EPEL 10)
 yum install hypershell

 apt install hypershell  (coming soon)
snap install hypershell  (coming soon)

apptainer build ...
   docker build ...              (ghcr.io/hypershell/hypershell) **
```

Getting Started

Alternative installations

Reach out to me for details!

Ask your HPC admins to add HyperShell as a module!



LMOD module available

```
$ module avail hypershell

-------------------------- Core Applications --------------------------
hypershell/2.7.0 (D)

  Where:
   D:  Default Module
```

3

Basics

The ‘hello world’ of workflows. Basic end-to-end.

Basics

Hello World

https://github.com/NERSC/htc_training/tree/main/hypershell/00_first

- Let’s run the simplest possible example to get started.
- The `hsx` program is short-hand for `hs cluster` and `-t` applies a command template to incoming arguments – not unlike *GNU Parallel*.

Simple ‘hello world’ example

```
$ seq 4 | hsx -t 'echo {}'
 WARNING [hypershell.server] No database configured - automatically disabled
1
2
3
4
```

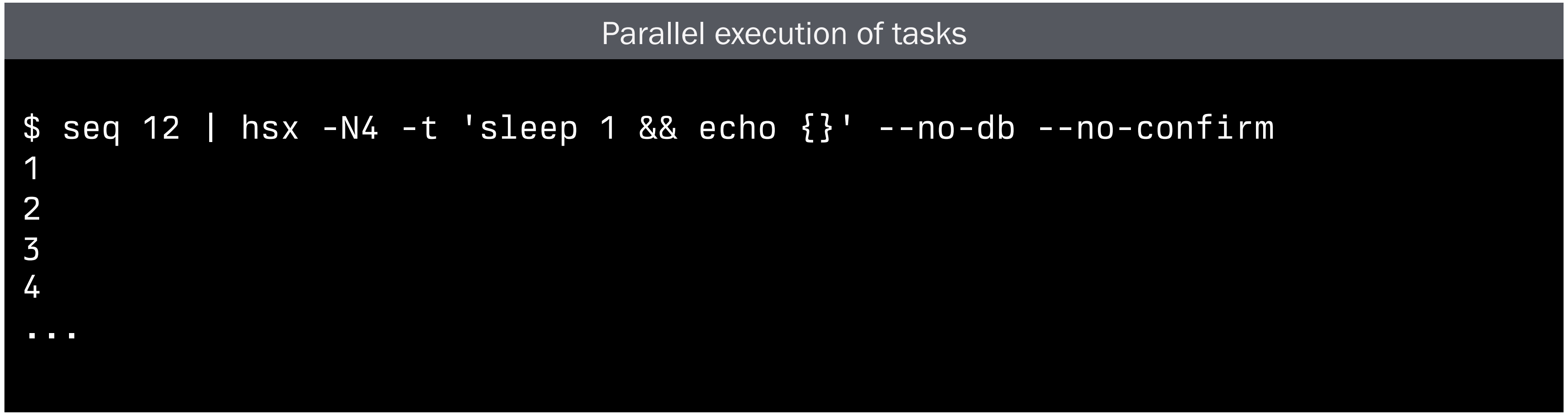
Use `'hs --help'` or `'man hs’` for help and detailed usage information. How can we disable this warning?

Basics

https://github.com/NERSC/htc_training/tree/main/hypershell/01_basic

Parallel execution of tasks

- The **-N** flag defines the number of parallel works (task executors).
- Later, we’ll see this is the number of workers *per-client*.



Parallel execution of tasks

```
$ seq 12 | hsx -N4 -t 'sleep 1 && echo {}' --no-db --no-confirm
1
2
3
4
...
```

Notice while this runs that the output comes back in bursts of 4.

Basics

https://github.com/NERSC/htc_training/tree/main/hypershell/02_basic_templates

Template patterns

- Rich template pattern syntax for quickly mapping inputs to commands.
- Inspired by *GNU Parallel*.
- See documentation for reference.

Template patterns for mapping inputs to outputs

```
$ find in/ -type f | hsx -N2 -t 'grep NEEDLE {} > out/{/-}.out' --no-db --no-confirm
```

Select out arguments from input lines using `{[2]}` or `{[-3:]}` slicing like in Python.

Basics

https://github.com/NERSC/htc_training/tree/main/hypershell/03_basic_failures

Capturing failed tasks

- Use `-f/--failures` to redirect inputs that result in a non-zero exit status.
- We're simulating failures using the Unix `true` and `false` programs.
- Later, we'll see how to use `--max-retries` with the database enabled.

Capturing failed task inputs

```
$ hsx task.in -N4 --no-db --no-confirm -f task.failed
...
WARNING [hypershell.server] Non-zero exit status (1) for task (4806ada6-7b47-
47e9-b78a-f6aa042a830b)
```

We can cycle the failed task file back in as input for a crude retry mechanism – not unlike *ParaFly*.

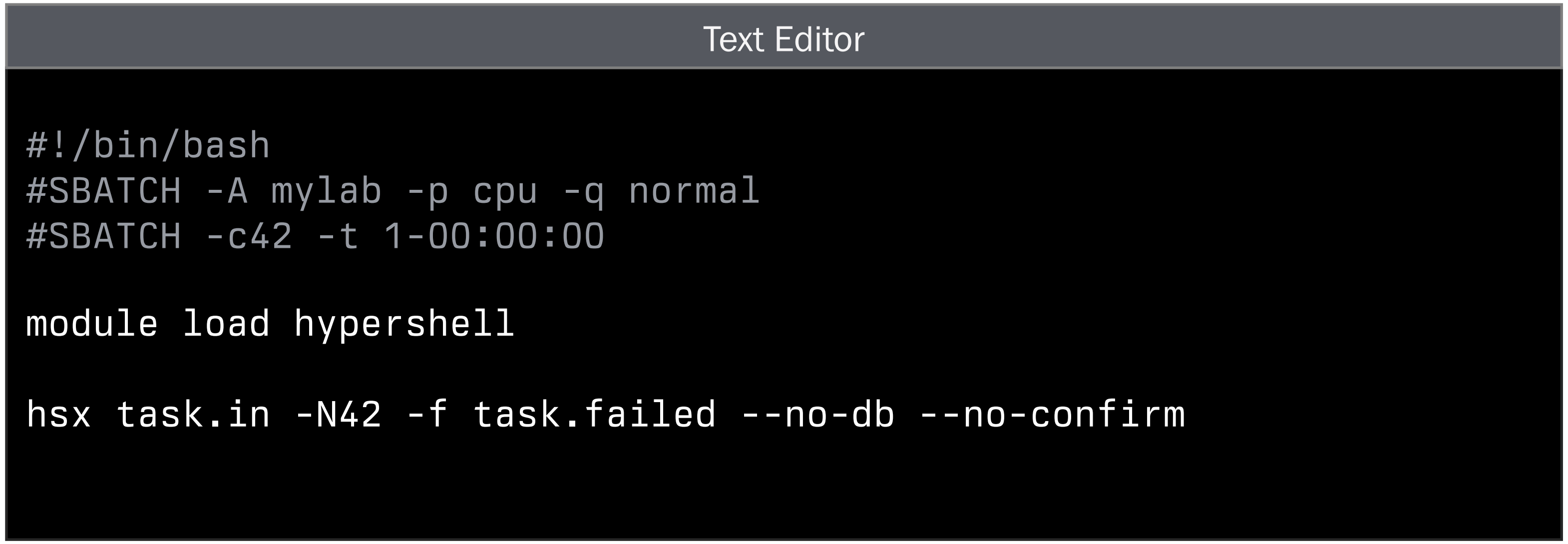
PURDUE UNIVERSITY | Rosen Center for Advanced Computing

Basics

https://github.com/NERSC/htc_training/tree/main/hypershell/04_basic_slurm

Slurm job script

- Slurm is the most common scheduler at HPC centers.
- This is a single-node job running single-core tasks.



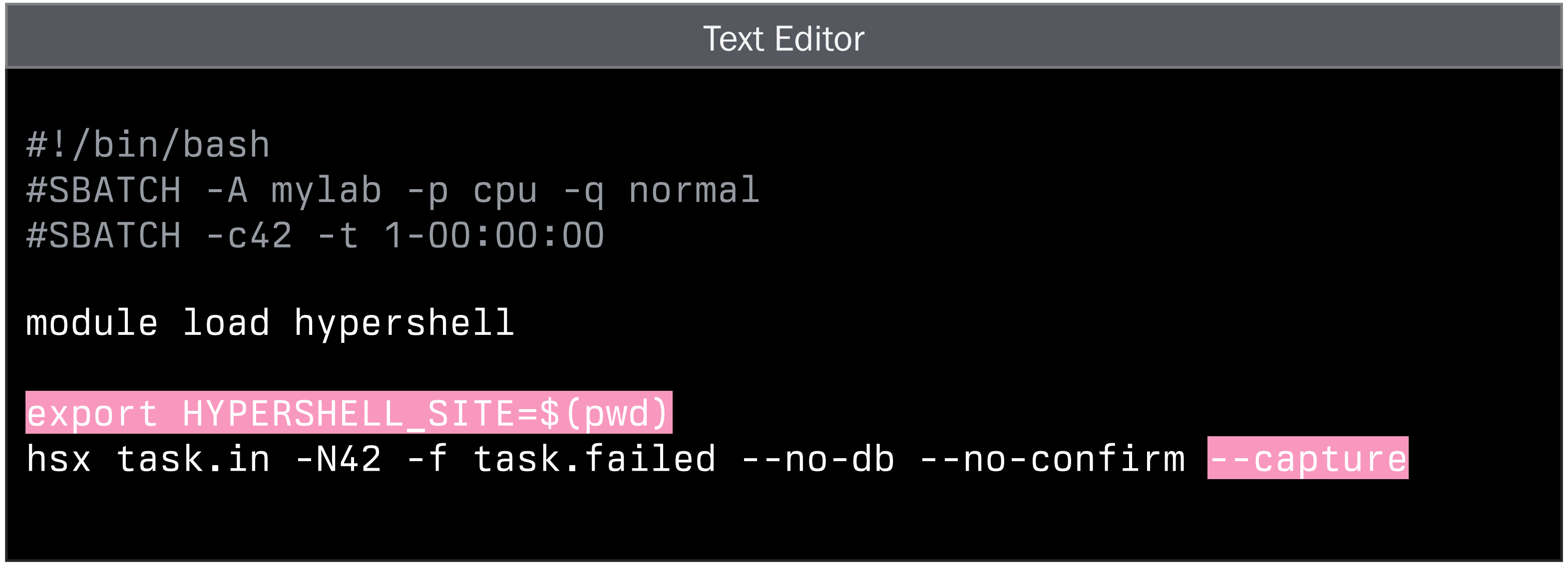
```
#!/bin/bash
#SBATCH -A mylab -p cpu -q normal
#SBATCH -c42 -t 1-00:00:00

module load hypershell

hsx task.in -N42 -f task.failed --no-db --no-confirm
```

Basics

https://github.com/NERSC/htc_training/tree/main/hypershell/05_basic_capture_output

Capturing output

- We’ll learn more about configuration soon.
- Use **`HYPERSHELL_SITE`** to control where outputs are directed with **`--capture`**



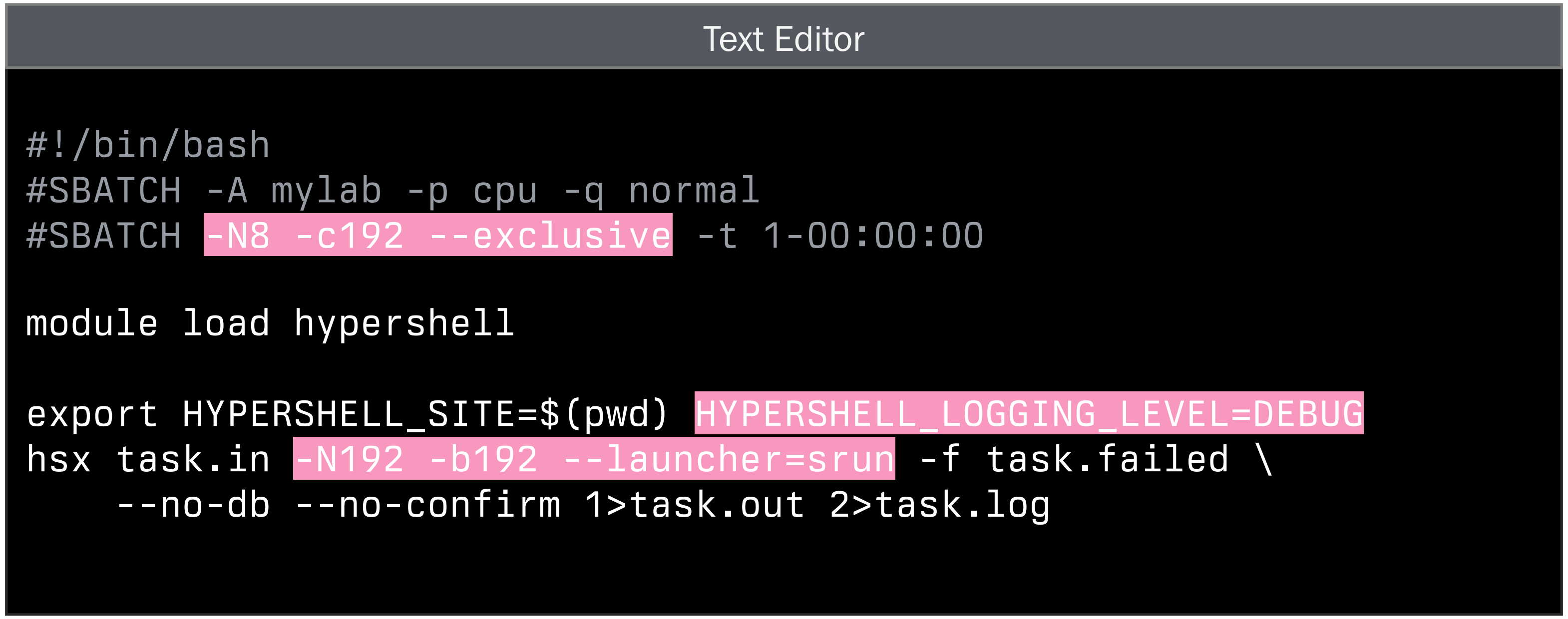
```
#!/bin/bash
#SBATCH -A mylab -p cpu -q normal
#SBATCH -c42 -t 1-00:00:00

module load hypershell

export HYPERSHELL_SITE=$(pwd)
hsx task.in -N42 -f task.failed --no-db --no-confirm --capture
```

Basics

https://github.com/NERSC/htc_training/tree/main/hypershell/06_basic_multinode

Distributed computing (many node)

- Let’s do the same thing again but with many tasks (100k) across many nodes.
- The **`--launcher=srun`** brings up the **`hs client`** on each node.



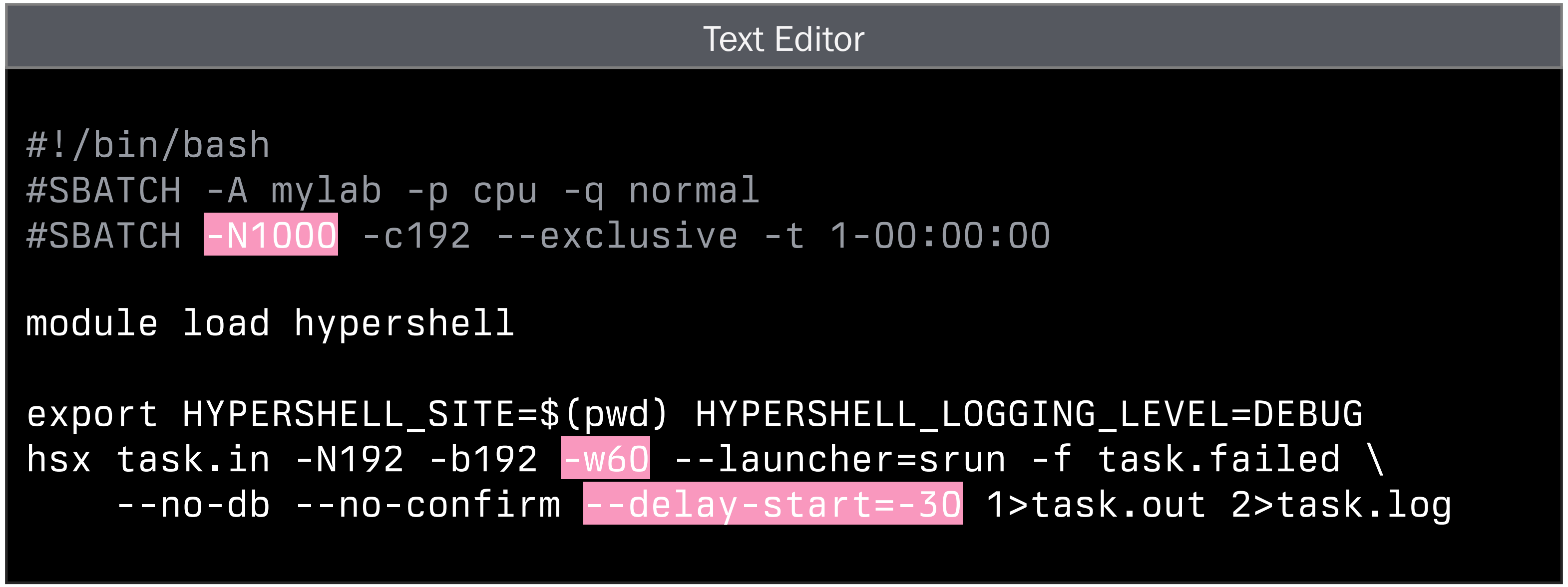
```
#!/bin/bash
#SBATCH -A mylab -p cpu -q normal
#SBATCH -N8 -c192 --exclusive -t 1-00:00:00

module load hypershell

export HYPERSHELL_SITE=$(pwd) HYPERSHELL_LOGGING_LEVEL=DEBUG
hsx task.in -N192 -b192 --launcher=srun -f task.failed \
    --no-db --no-confirm 1>task.out 2>task.log
```

Basics

https://github.com/NERSC/htc_training/tree/main/hypershell/07_extreme_scale

Extreme scale computing

- Use **`-w`** / **`--bundlewait`** to govern task bundle synchronization (avoid under-filled bundles).
- The **`--delay-start`** option staggers the launch across a large cluster.



```
#!/bin/bash
#SBATCH -A mylab -p cpu -q normal
#SBATCH -N1000 -c192 --exclusive -t 1-00:00:00

module load hypershell

export HYPERSHELL_SITE=$(pwd) HYPERSHELL_LOGGING_LEVEL=DEBUG
hsx task.in -N192 -b192 -w60 --launcher=srun -f task.failed \
    --no-db --no-confirm --delay-start=-30 1>task.out 2>task.log
```

4

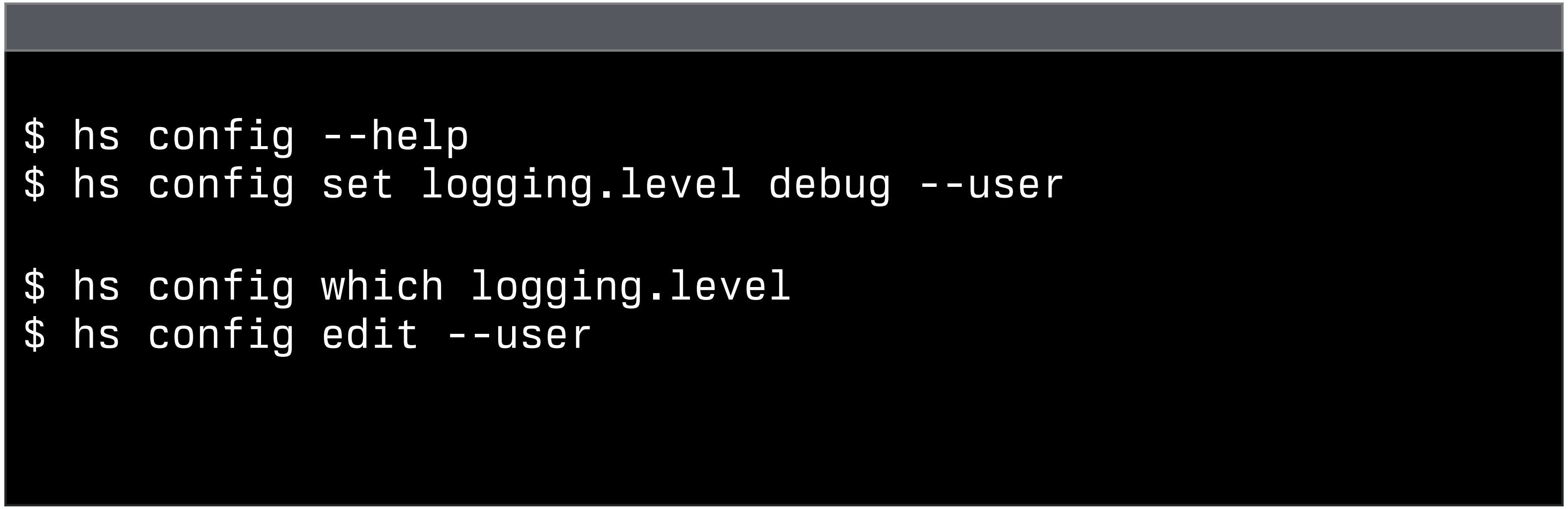
Configuration

Configuration. Submitting to a database.

Configuration

Setting parameters globally

- Define parameters globally that persist between invocations.
- Set from the command-line or edit the file directly.

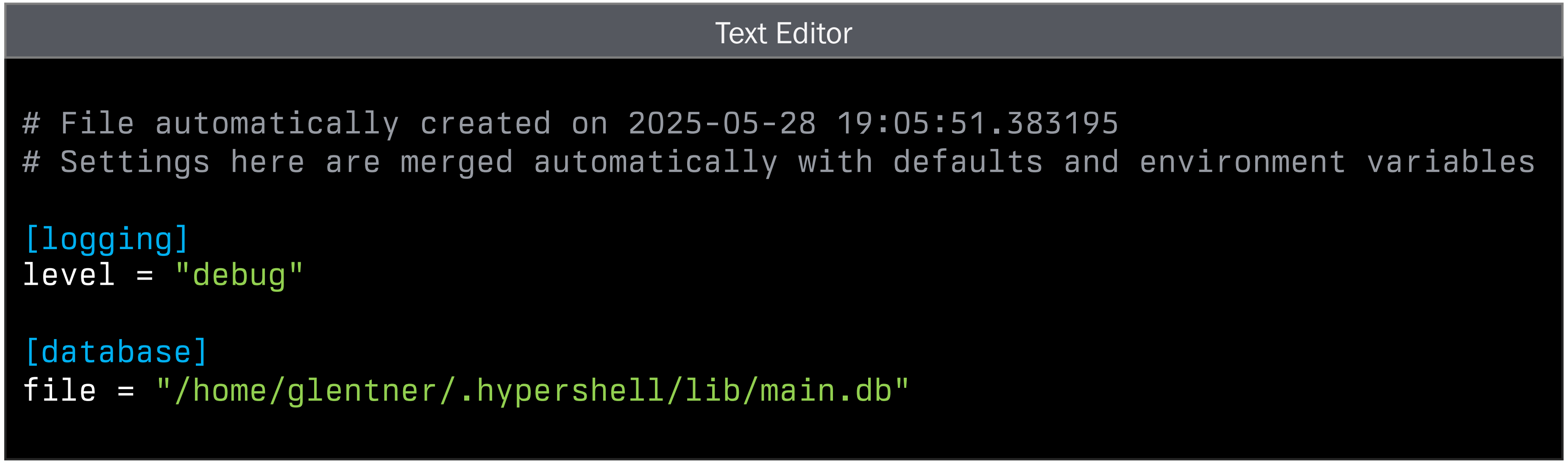


```
$ hs config --help
$ hs config set logging.level debug --user

$ hs config which logging.level
$ hs config edit --user
```

Configuration

Local SQLite database

- Include a database path and HyperShell will persist tasks.
- Be careful which filesystem this points to (e.g., /tmp or /home is best).



```
# File automatically created on 2025-05-28 19:05:51.383195
# Settings here are merged automatically with defaults and environment variables

[logging]
level = "debug"

[database]
file = "/home/glentner/.hypershell/lib/main.db"
```

Configuration

https://github.com/NERSC/htc_training/tree/main/hypershell/08_database_intro

Submitting tasks to the database

- The server/cluster need not be running to add tasks.
- Use **`hs submit`** with either single commands or collections from a file.

Submit tasks

```
$ hs submit echo "hello world"
 DEBUG [hypershell.submit] Submitted single task (explicit)
  INFO [hypershell.submit] Submitted task (ce8893cd-c587-4f44-9499-7c679bd2b437)

$ hs submit <(seq 100) --template "echo {}"
 DEBUG [hypershell.submit] Submitted from /proc/self/fd/11 (implicit - not executable)
...
  INFO [hypershell.submit] Submitted 100 tasks
```

What command-line option can we use to make batch file submission “explicit”?

Restarts

Submit tasks now - restart the cluster later

- It is often better to have your workflow “restart” even from the beginning.
- If there are no tasks remaining the program will simply shut down.

Restart cluster where it left off

```
$ hsx --restart ...
   DEBUG [hypershell.server] Started
...
 WARNING [hypershell.server] Database exists (101 previous tasks)
    INFO [hypershell.server] Found 101 unfinished task(s)
    INFO [hypershell.server] Reverted 0 previously interrupted task(s)
   DEBUG [hypershell.server] Scheduled 1 tasks
   DEBUG [hypershell.server] Scheduled task (ce8893cd-c587-4f44-9499-7c679bd2b437)
...
```

5

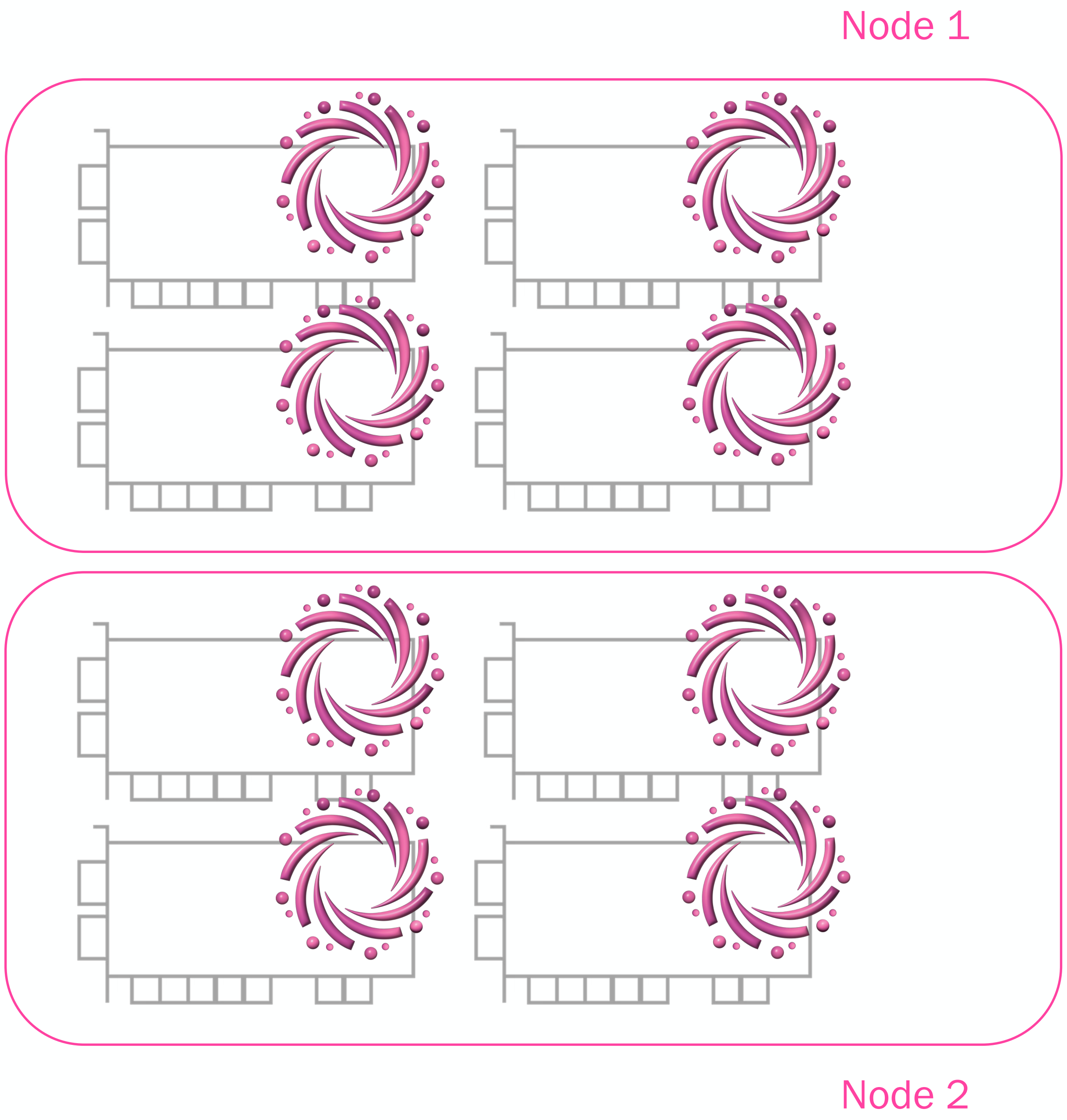
GPU Multiplexing

Specific use-case in demand right now.

GPU Multiplexing

Packing small tasks

- GPUs are expensive!
- Make better use of large GPUs.
- Accelerate smaller tasks in parallel with one client per GPU.
- Or deploy persistent autoscaling cluster for throughput training, more on this next!

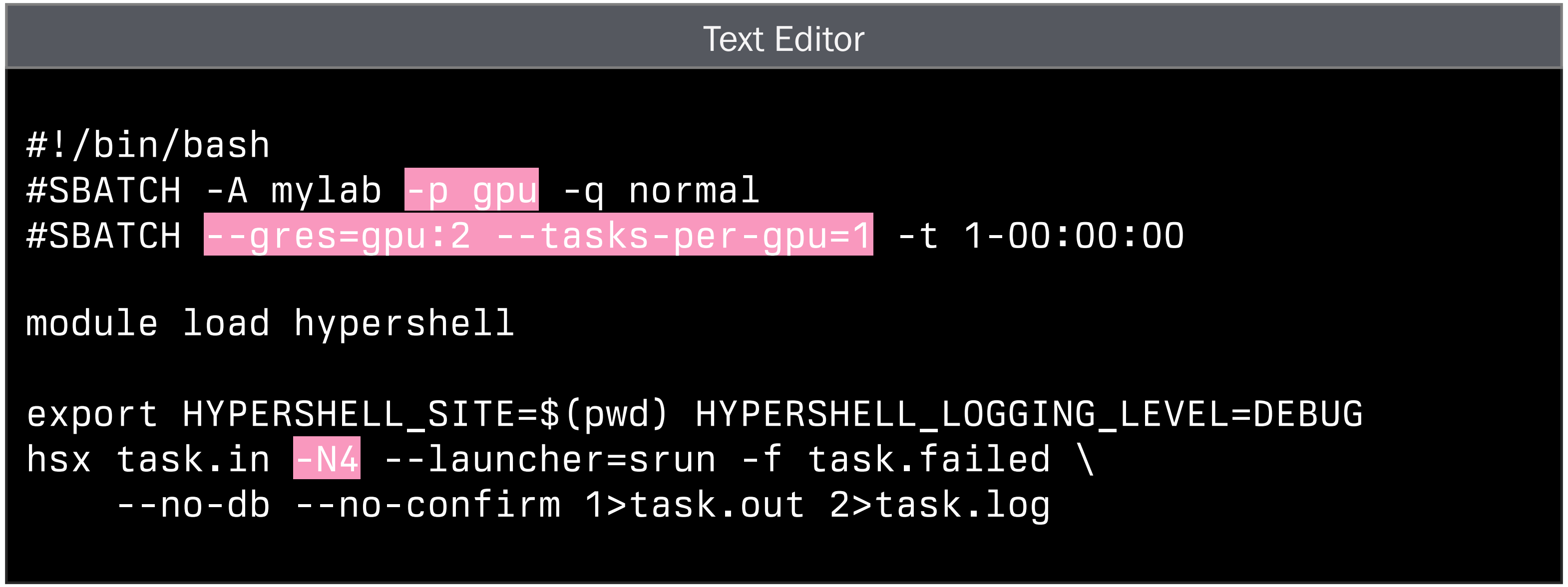


Rosen Center for Advanced Computing

GPU Multiplexing

Packing small tasks correctly

- Large GPUs are expensive -- researchers still operate small models/tasks.
- One Slurm task per GPU launches one **`hs client`** per GPU.



```
#!/bin/bash
#SBATCH -A mylab -p gpu -q normal
#SBATCH --gres=gpu:2 --tasks-per-gpu=1 -t 1-00:00:00

module load hypershell

export HYPERSHELL_SITE=$(pwd) HYPERSHELL_LOGGING_LEVEL=DEBUG
hsx task.in -N4 --launcher=srun -f task.failed \
    --no-db --no-confirm 1>task.out 2>task.log
```

6

Server vs Client

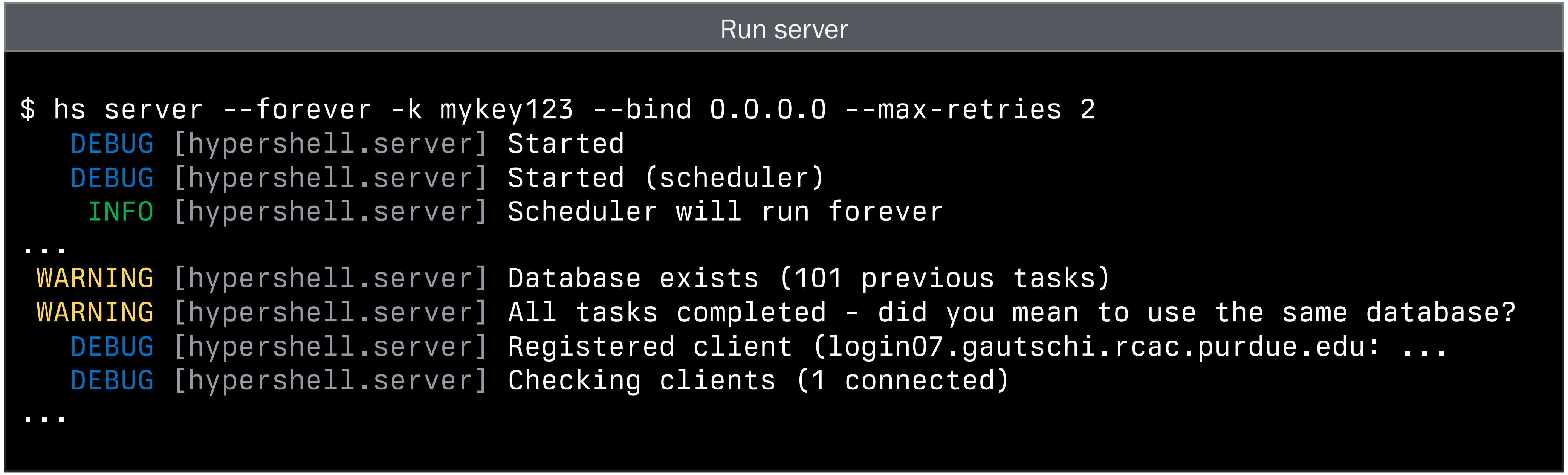
Running the server separate from clients.

Submitting tasks.

Server vs Client

One server many clients

- Keep the server alive elsewhere and scale out clients as needed.
- Protect using secure auth key! (automatic with cluster)
- Bind to 0.0.0.0 to allow remote connections (default: localhost)



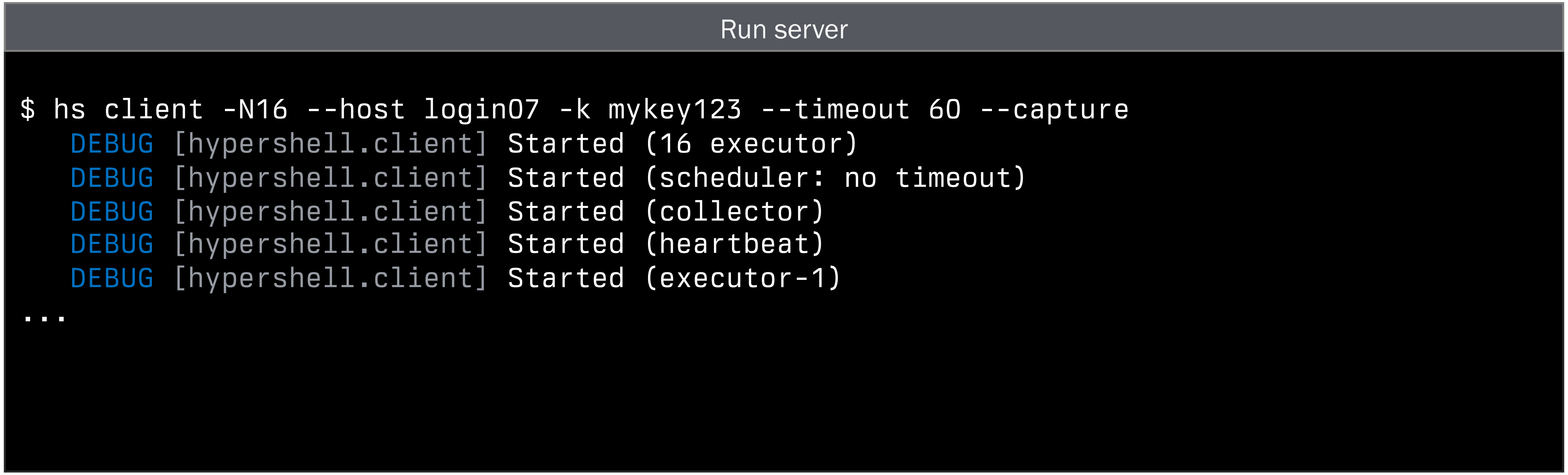
Run server

```
$ hs server --forever -k mykey123 --bind 0.0.0.0 --max-retries 2
   DEBUG [hypershell.server] Started
   DEBUG [hypershell.server] Started (scheduler)
    INFO [hypershell.server] Scheduler will run forever
...
 WARNING [hypershell.server] Database exists (101 previous tasks)
 WARNING [hypershell.server] All tasks completed - did you mean to use the same database?
   DEBUG [hypershell.server] Registered client (login07.gautschi.rcac.purdue.edu: ...
   DEBUG [hypershell.server] Checking clients (1 connected)
...
```

Server vs Client

Clients connect and process tasks

- Clients register themselves with the server and process tasks locally.
- Clients can be evicted if they miss too many heartbeats. (tasks are re-scheduled)
- Set a timeout to have them shutdown if there are no tasks. (default: never disconnect)



Run server

```
$ hs client -N16 --host login07 -k mykey123 --timeout 60 --capture
   DEBUG [hypershell.client] Started (16 executor)
   DEBUG [hypershell.client] Started (scheduler: no timeout)
   DEBUG [hypershell.client] Started (collector)
   DEBUG [hypershell.client] Started (heartbeat)
   DEBUG [hypershell.client] Started (executor-1)
...
```

7

Task Management

Task search and updates.

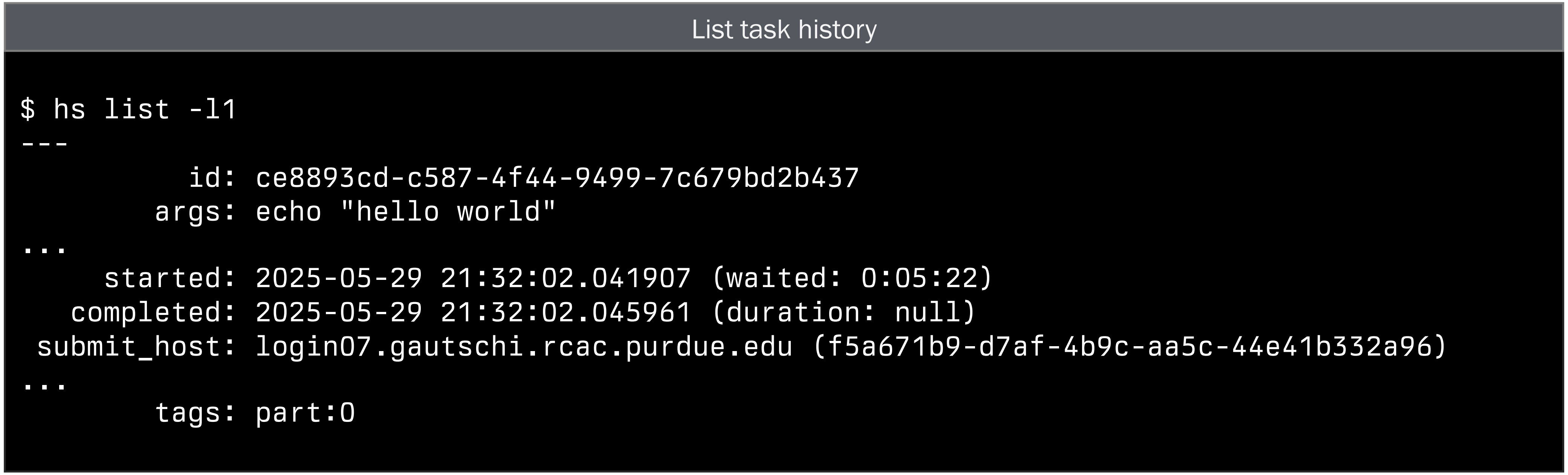
Cancelling, reverting, and deleting tasks.

User-defined tags (command, inline, group).

Task Management

Search database for task history

- Use **`hs list`** to search task history and remaining tasks.
- Many output formats (normal, list, table, csv, json).
- Output file paths stored in database for retrieval.



List task history

```
$ hs list -l1
---
          id: ce8893cd-c587-4f44-9499-7c679bd2b437
        args: echo "hello world"
...
     started: 2025-05-29 21:32:02.041907 (waited: 0:05:22)
   completed: 2025-05-29 21:32:02.045961 (duration: null)
 submit_host: login07.gautschi.rcac.purdue.edu (f5a671b9-d7af-4b9c-aa5c-44e41b332a96)
...
        tags: part:0
```

Task Management

Search database for task history

- Find most recent completed tasks, all remaining tasks, by client host, etc …
- User defined tags allow for custom attributes for single or groups of tasks.

Count of remaining tasks within group belonging to tag

```
$ hs list --count --remaining --with-tag site:b
13
```

Including only the tag “key” will return any task with that key regardless of value.

Task Management

User-defined tags

- Apply tags at submit time from the command-line.
- All tasks inherit these tags.

Submit group of tasks with shared tag

```
$ seq 1000 | hs submit -t group:1 demo
...
```

Task Management

Inline tag assignments

- Include tag assignments with special comment syntax within submission files.
- Combine these two approaches together in one invocation.

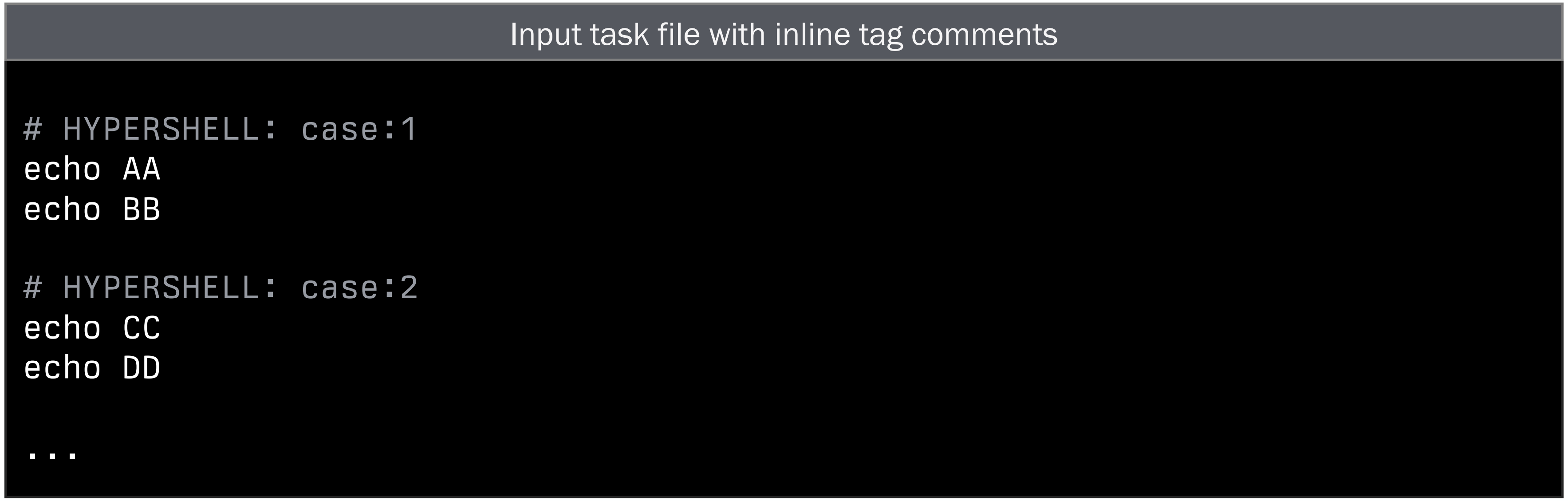
Input task file with inline tag comments

```
echo 1  # HYPERSHELL: data:1
echo 2  # HYPERSHELL: data:2
echo 3  # HYPERSHELL: data:3
echo 4  # HYPERSHELL: data:4
echo 5  # HYPERSHELL: data:5
echo 6  # HYPERSHELL: data:6

...
```

Task Management

Inline tag assignments

- Stand-alone comments apply tags to all tasks. (until overridden)
- Combine these three approaches together in one invocation.



Input task file with inline tag comments

```
# HYPERSHELL: case:1
echo AA
echo BB

# HYPERSHELL: case:2
echo CC
echo DD

...
```

8

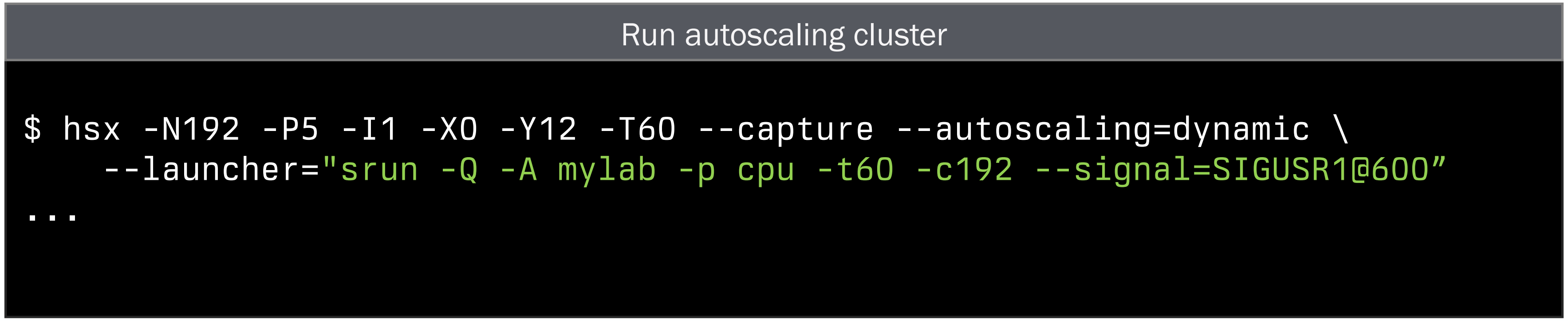
Autoscaling

Elastic scaling the cluster - even to zero.

Autoscaling

Dynamically launch clients

- Based on “task pressure” (dimensionless quantity)
- The launcher is used as a prefix to provision a client.
- Policy based with initial / min / max sized cluster.
- Set a client timeout and send SIGUSR1 ahead of job timeout to drain the worker.



Run autoscaling cluster

```
$ hsx -N192 -P5 -I1 -X0 -Y12 -T60 --capture --autoscaling=dynamic \
    --launcher="srun -Q -A mylab -p cpu -t60 -c192 --signal=SIGUSR1@600”
...
```

9

Automated Clusters

User facility (service) deployments.

Automatic scaling of pilot jobs.

Database partitioning.

Automated Clusters

Service account pilot jobs

- Run dedicated server with *PostgreSQL* off-cluster (e.g., Kubernetes).
- Submit **`xcore-pilot`** job to run clients, as necessary.

Pilot job

```
#!/bin/bash
#SBATCH -A xcore -p cpu -q normal -J xcore-pilot
#SBATCH -N1 -n1 -c48 -t 01:00:00
#SBATCH -o /dev/null
#SBATCH --signal B:USR1@300

exec ~/bin/xcore-hs client -N48 -T60 --capture \
        --host hs-server.xcore.university.edu -k `cat ~/.hypershell/key` \
        2>>$SCRATCH/xcore-pilot/log/hypershell-`date +"%Y%m%d"`.log
```

Automated Clusters

Submit jobs only when necessary

- Cron (or similar) execution of **`xcore-autoscale`** script.
- The **`xcore-hs`** script is just a wrapper around **hs** itself.

Pilot job

```
#!/bin/bash

JOBS=$(/usr/bin/sacct -n -X -u xcore -s R,PD --name xcore-pilot | wc -l)
TASKS=$(~/bin/xcore-hs list --count --remaining)

if [ $JOBS -gt 8 ] || [ $(((TASKS / 48) - JOBS)) -lt 1 ]
then
    echo "no scale"
else
    sbatch ~/bin/xcore-pilot
fi
```

So much more we could dive into, but this covers the gist of it.

End

Please fill out the survey!

Questions?