Introduction to Roar Collab (RC)

Emery Etter
emery@psu.edu
Institute for Computational and Data Sciences

ICDS

The Institute for Computational and Data Sciences (ICDS) is one of seven interdisciplinary research institutes within Penn State's Office of the Senior Vice President for Research.

The mission of ICDS is to build capacity to solve problems of scientific and societal importance through cyber-enabled research.

ICDS enables and supports the diverse computational and data science research taking place throughout Penn State. Users come from all corners of the university and conduct interdisciplinary research using these high-performance computing (HPC) systems.

Research Computing Clusters

ICDS provides and supports research computing platforms for university faculty, staff, and students.

Roar Collab (RC) is the flagship computing cluster for Penn State researchers and utilizes the Red Hat Enterprise Linux (RHEL) version 8 OS.

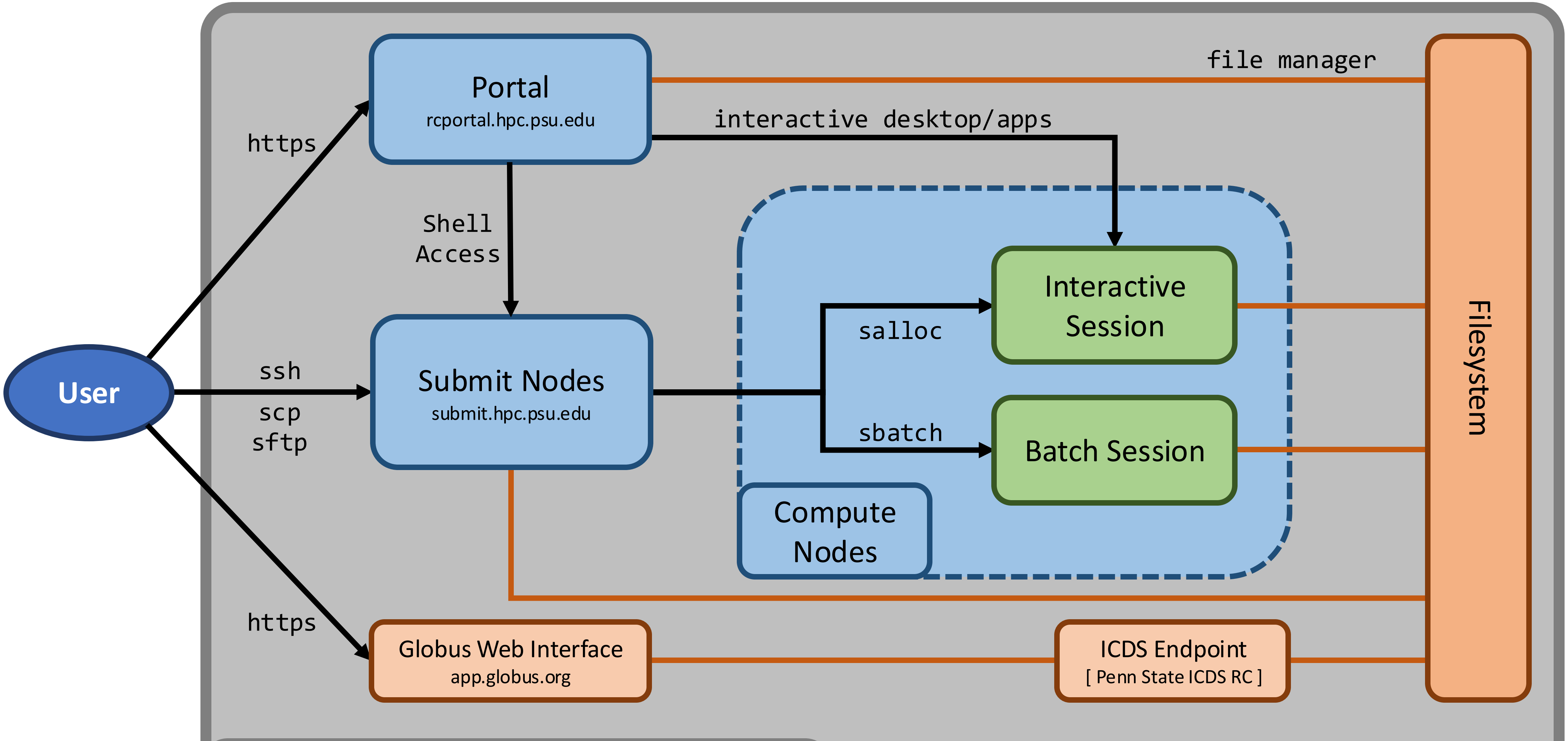
Today's session is specifically geared towards RC usage, but this material is, for the most part, generally applicable to other research computing clusters as well.

Shared Computing Systems

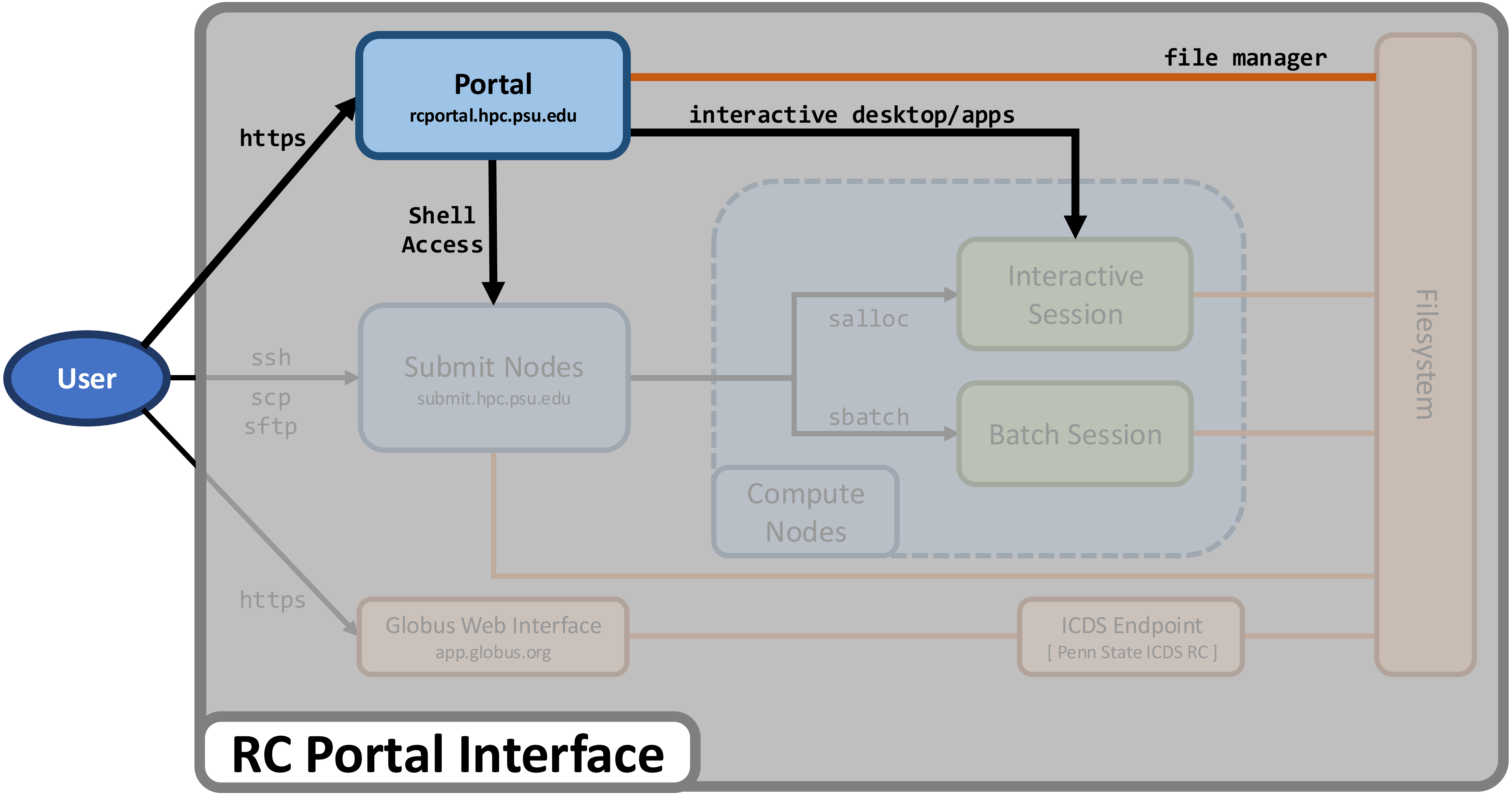
Research computing clusters are shared computational resources. To perform computationally-intensive tasks, users must request compute resources and be provided access to those resources to perform the tasks.

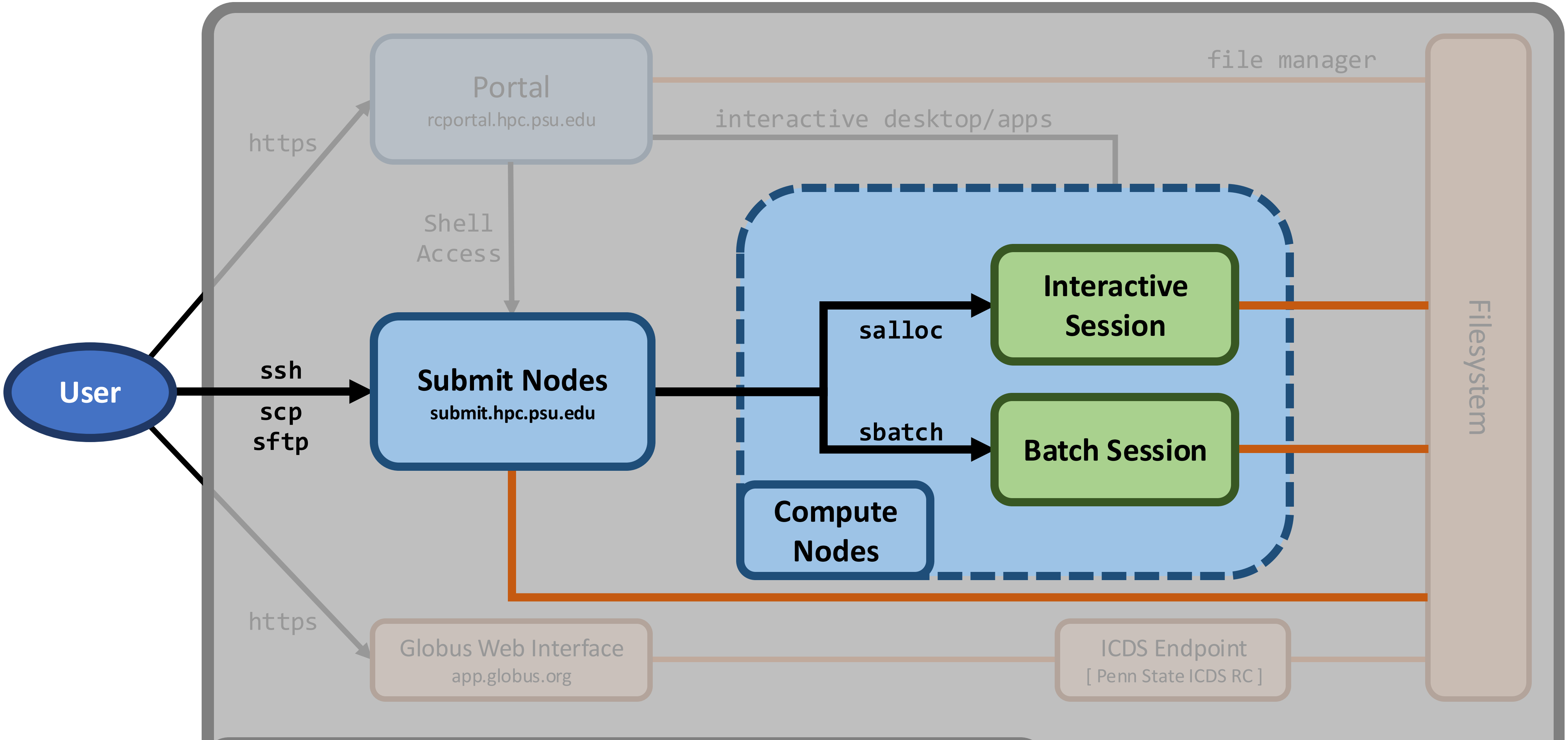
The request/provision process allows the tasks of many users to be scheduled and carried out efficiently to avoid resource contention.

Slurm (**S**imple **L**inux **U**tility for **R**esource **M**anagement) is utilized by both RC as the job scheduler and resource manager.

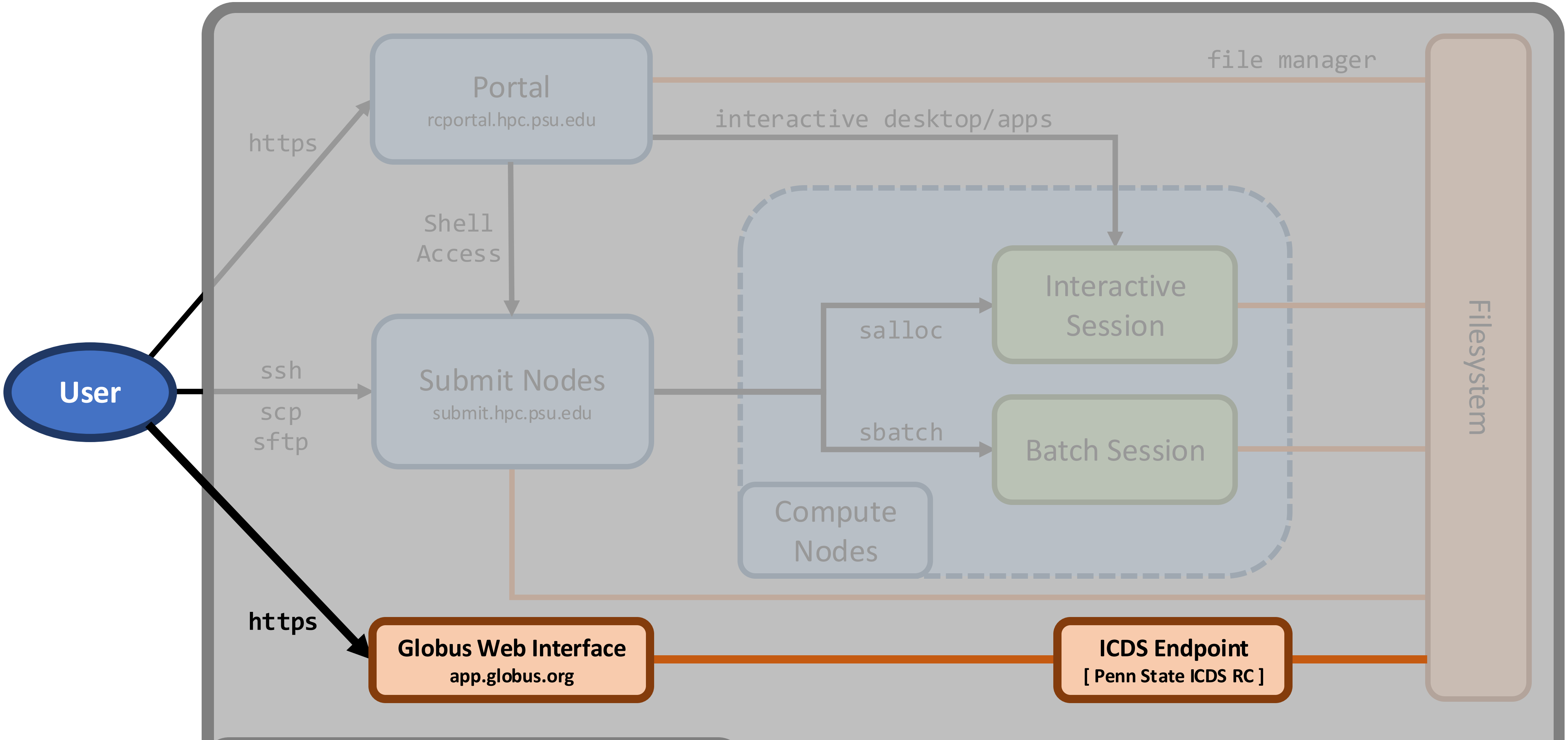


Roar Collab User Flow Diagram


file manager
Portal
rcportal.hpc.psu.edu
interactive desktop/apps
https
Shell
Access
Interactive
Session
salloc
User
ssh
scp
sftp
Submit Nodes
submit.hpc.psu.edu
sbatch
Batch Session
Filesystem
Compute
Nodes
https
Globus Web Interface
app.globus.org
ICDS Endpoint
[Penn State ICDS RC]
RC Portal Interface

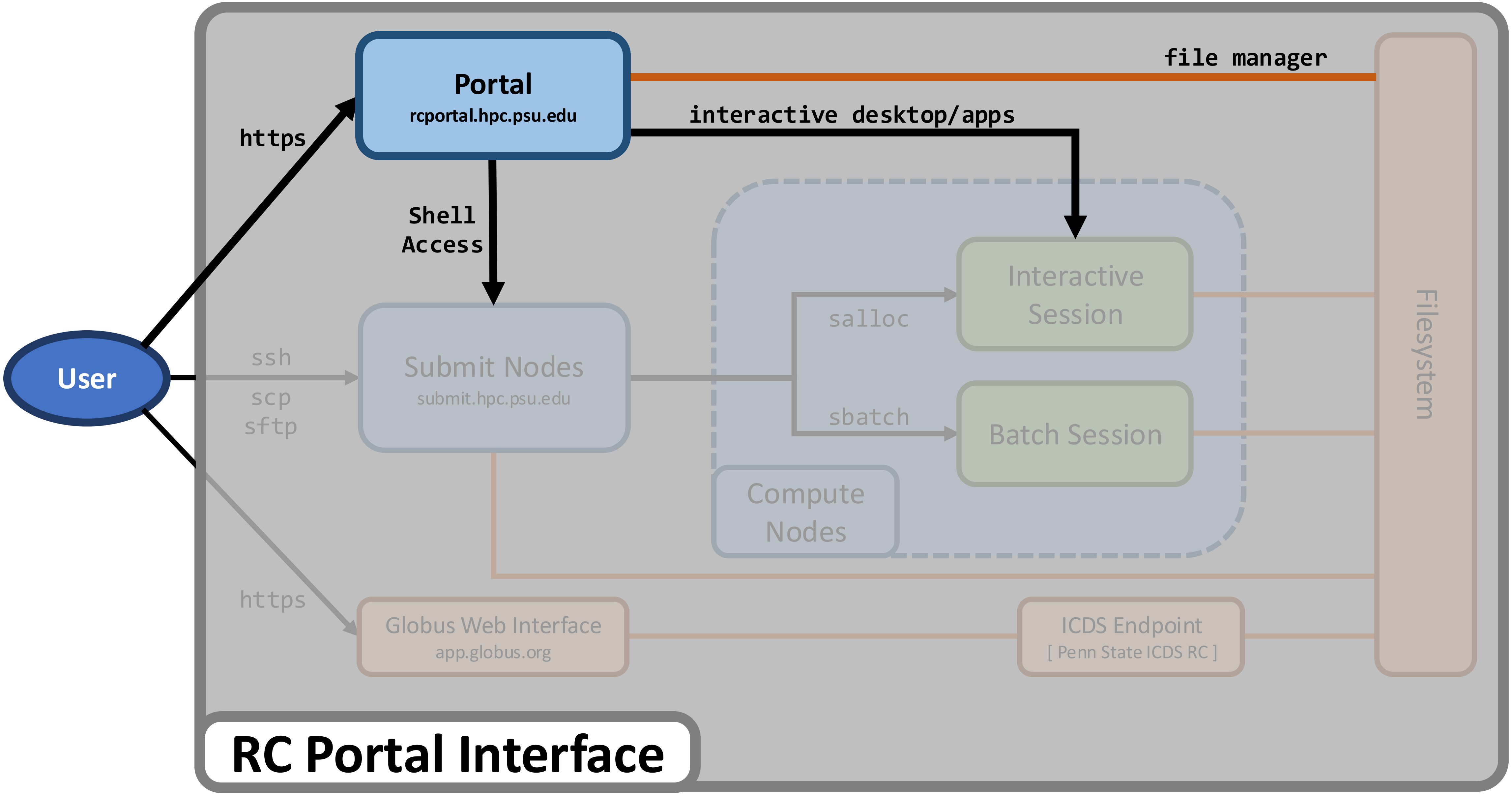


RC Command Line Interface (CLI)



RC Globus Interface

Interfaces Overview



Portal
rcportal.hpc.psu.edu
file manager
interactive desktop/apps
https
Shell
Access
User
ssh
scp
sftp
Submit Nodes
submit.hpc.psu.edu
salloc
sbatch
Interactive Session
Batch Session
Compute Nodes
Filesystem
https
Globus Web Interface
app.globus.org
ICDS Endpoint
[Penn State ICDS RC]
RC Portal Interface

Open OnDemand (OOD)

The RC Portal is offered via OOD software developed by Ohio Supercomputing (OSC).

OOD is an NSF-fund, open-source HPC portal that provides users with a simple graphical web interface to HPC resources.

Users can submit and monitor jobs, manage files, and run applications using just a web browser.

RC Portal

HANDS ON

To access the RC Portal, navigate to the following website in a web browser: portal.hpc.psu.edu

To log into the RC Portal successfully, users must log in using valid Penn State access account credentials and must also have an account on RC.

To request an account on RC, submit a request at icds.psu.edu/account-setup

RC Portal Menu Bar

Apps: Lists all available Portal apps

Files: Provides a convenient graphical file manager and lists accessible file locations

Jobs: Lists active jobs and allows use of the Job Composer

Clusters: Provides shell access to submit nodes on RC

**Clusters > RC Shell Access specifics will be discussed within the CLI section.*

Interactive Apps: Provides access to Portal interactive apps and interactive servers

My Interactive Sessions: Lists any active sessions

Interactive Desktop Session

Under the **Interactive Apps** menu, the **Interactive Desktop** option allows users to request an interactive desktop session on compute nodes.

Selecting this option directs users to a form to easily request resources for the session.

Resource Options

Account: All users have access to the **open** account, but any paid compute accounts that a user has access to will appear in the dropdown.

Partition: The **open** account has the **open** and **interactive** partitions, but paid compute accounts have different partitions available which have different characteristics and are charged at various rates.

Hours: Select the maximum length of the session.

Enable advanced Slurm options: If the checkbox is selected, it enables a text field to enter custom Slurm resource directives for the session. Redefining any form-selected options here will supersede previous selections.

**More specifics on partitions and Slurm resource directives will be discussed later.*

Request A Session

HANDS ON

Let's request "1 laptop-worth of resources" for a session. For example, my laptop has a 6-core processor (6 "performance cores" at least) and 36 GB of RAM.

Select **Interactive Apps > Interactive Desktop** and enter the following form options...

Account: open

Partition: open

Number of Hours: 2

Enable advanced Slurm options: Checked

Sbatch options: -N 1 -n 6 --mem 36GB

Then select **Launch** to submit the request which generates a tile in the My Interactive Sessions list.

Enter A Session

HANDS ON

The session will be listed in the My Interactive Sessions list as it awaits placement. When the job is placed a blue Launch button becomes available:

The session can be launched either using the **noVNC Connection** option or by using the **Native Instructions** displayed within the session tile.

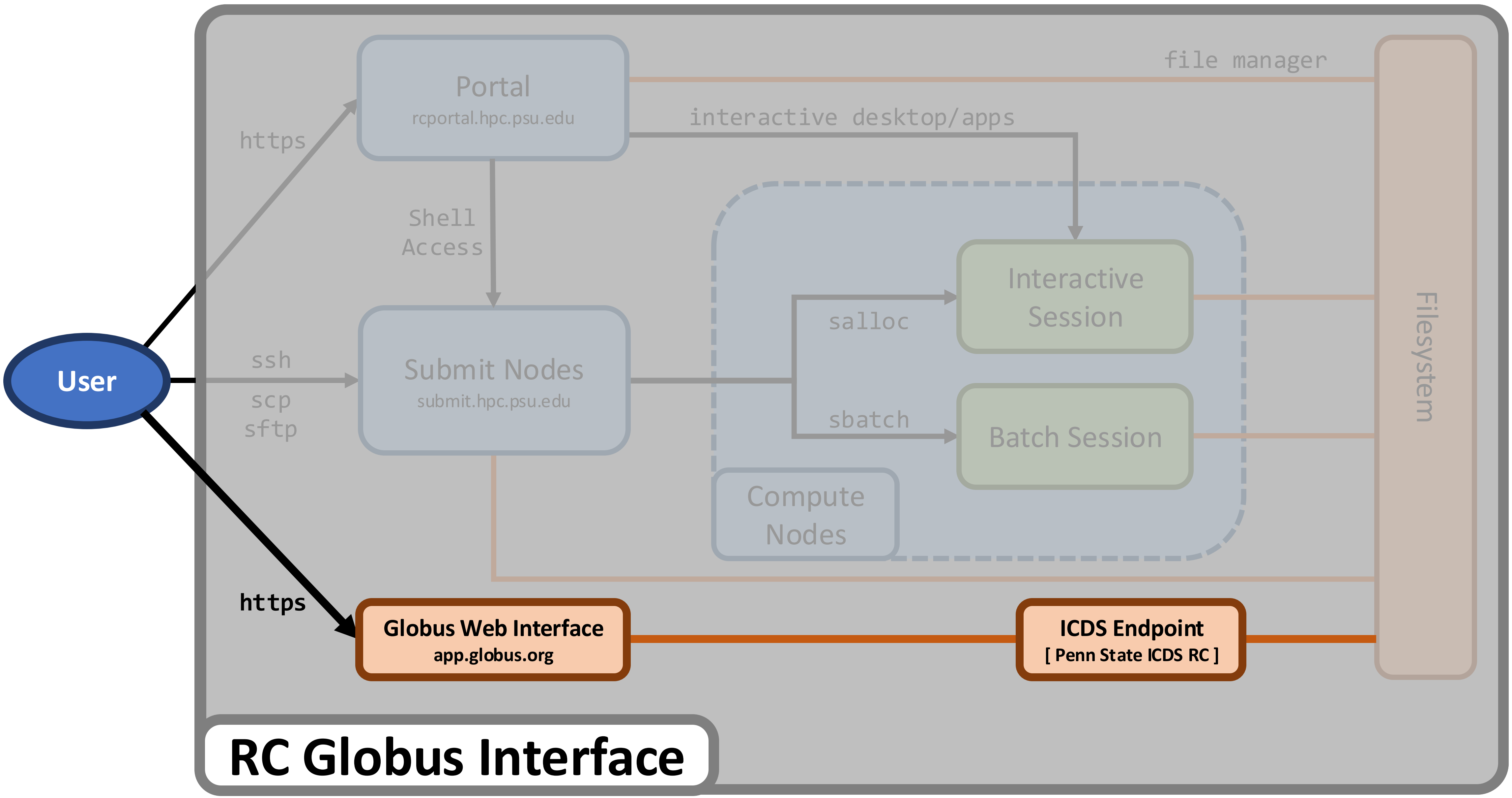
The session launches into a graphical desktop of the RHEL8.

Other Interactive Apps

Under the **Interactive Apps** menu, apps for specific software packages are available, allowing users to request interactive sessions on compute nodes.

The resource selections are essentially the same as the options for the Interactive Desktop form, only with minor software-specific differences such as selecting a software version.

For example, the RStudio Server app requires a version of R to be specified. Under **Environment selection**, selecting **Use default environment** allows the user to select R version 4.5.0 from the **R version** dropdown.


User
https
Portal
rcportal.hpc.psu.edu
file manager
interactive desktop/apps
Shell
Access
ssh
scp
sftp
Submit Nodes
submit.hpc.psu.edu
salloc
sbatch
Interactive
Session
Batch Session
Compute
Nodes
Filesystem
https
Globus Web Interface
app.globus.org
ICDS Endpoint
[Penn State ICDS RC]
RC Globus Interface

Globus

Globus is a web-based file transfer tool. With Globus, users can easily, reliably, and securely transfer data to and from RC.

To access the Globus File Manager utility, either

- Select the Globus icon in the RC Portal's File Manager

OR

- Navigate to the globus.org in a web browser and **Log In** via the Penn State organization with your Penn State Access Account credentials

Transfer Data with Globus

In the Globus File Manager, search for the following RC endpoint in the **Collection** field: **Penn State ICDS RC**

After adding this Collection, accessible RC locations are now available within the File Manager window.

Manage and transfer files with simple web interface operations:

- To download a file, right-click the file and select **Download**.
- To upload a file, select **Upload** from the **Pane 1 Menu** on the right.
- To transfer to another Globus endpoint, **select Transfer or Sync To** from the **Pane 1 Menu** and search for the other endpoint in the other panel.

Transfer Workshop Repo

HANDS ON

In a new browser tab, download the following file from the following link to your device:

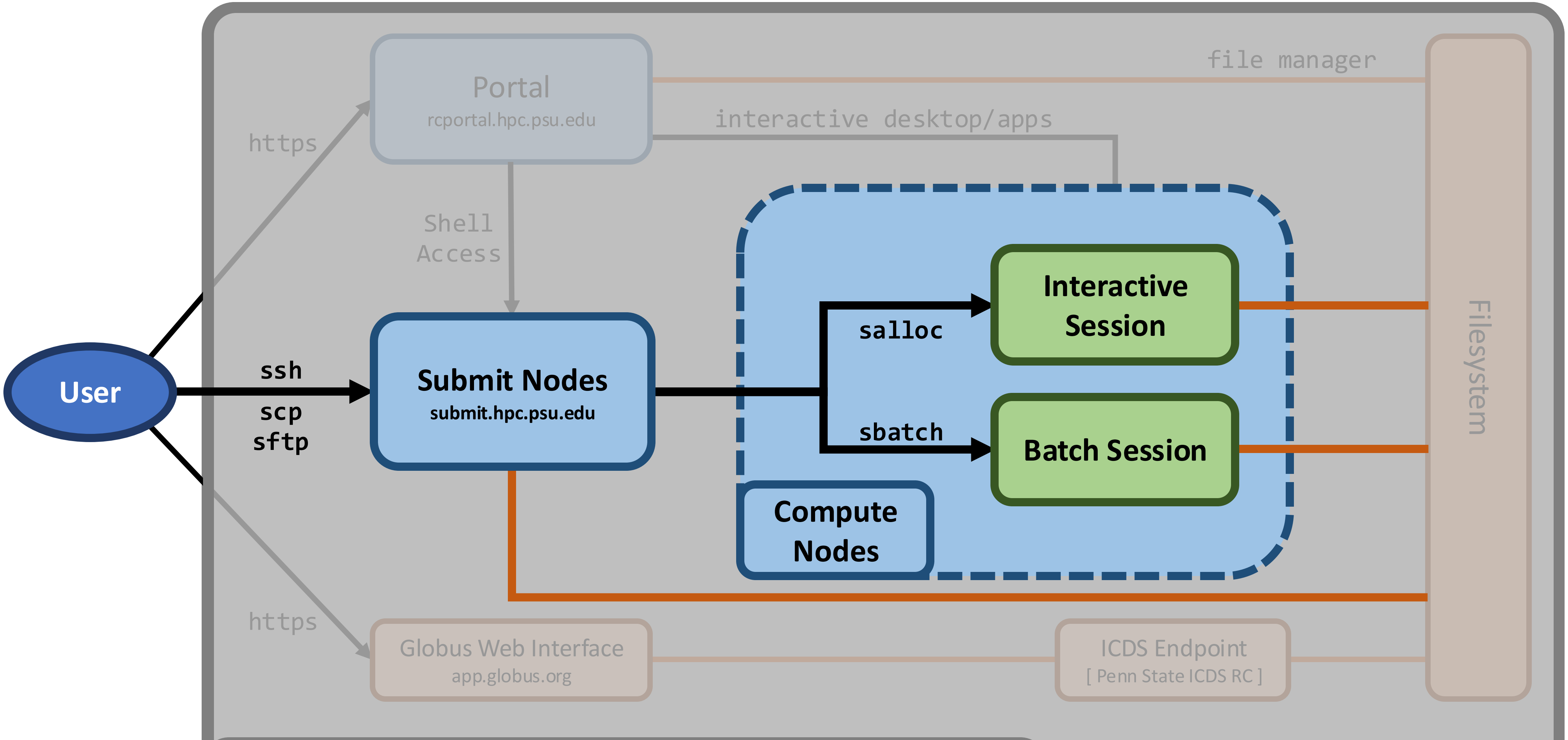
https://github.com/PSU-ICDS/rc_intro/archive/refs/heads/main.zip

In the Globus File Manager, select the following:

Collection: Penn State ICDS RC

Path: /scratch/<userid>

Upload this file by selecting **Upload > Select Files to Upload** from the **Pane 1 Menu** on the right.



RC Command Line Interface (CLI)

Key Notation

Green highlight identifies a `command` that can be run in a terminal session.

Light `$BLUE` highlight with a dollar sign identifies a user environment variable.

Grey highlight identifies `something` in a file or a system prompt.

Angle brackets like **<this>** represent the need to insert a string. Replace **<this>** with the string, and do not keep the angle brackets.

Shell and Environment

Linux uses a **shell** to operate between the core operating system and other programs. In a command line interface, users interact with their **environment** via the **shell**.

A user's **environment** establishes the locations of the **user files** and **software files**.

Users on RC typically utilize the Bourne Again Shell, known as **bash**. The default shell can be modified at accounts.psu.edu in the Computing Settings section by selecting a different Login Shell from the dropdown menu.

Environment Variables

A user environment is established by **environment variables**.

To show your environment, use the `env` command.

→Tip: To search for something specific in your environment, use the `env | grep -i <string>` command.

The `$PATH` environment variable is an important part of the user environment that allows the shell to find the location of the commands being entered.

Some Basic Linux Commands

```
pwd     print working directory
mkdir   make directory
ls      list files and directories
cd      change directory
rm      remove file
cp      copy file
mv      move / rename file or directory
```

These commands can be run by the shell because the environment tells the shell where to find them using the $PATH variable.

The Most Important Linux Command Of All Time for Learning:

```
man     launch manual for a command
```

Note: Use auto-complete with the Tab key to make command entry efficient!

There is a great tutorial for the Unix Shell here:

https://swcarpentry.github.io/shell-novice

Access Submit Node

HANDS ON

Choose one of two options to initiate a Command Line Interface session on a submit node:

- From a terminal session, run the following command:

```
ssh <userid>@submit.hpc.psu.edu
```

- Access portal.hpc.psu.edu and from the banner select

```
Clusters > RC Shell Access
```

After completing the MFA process, the prompt will indicate that the interactive session is on a submit node.

Prepare Workshop Repo

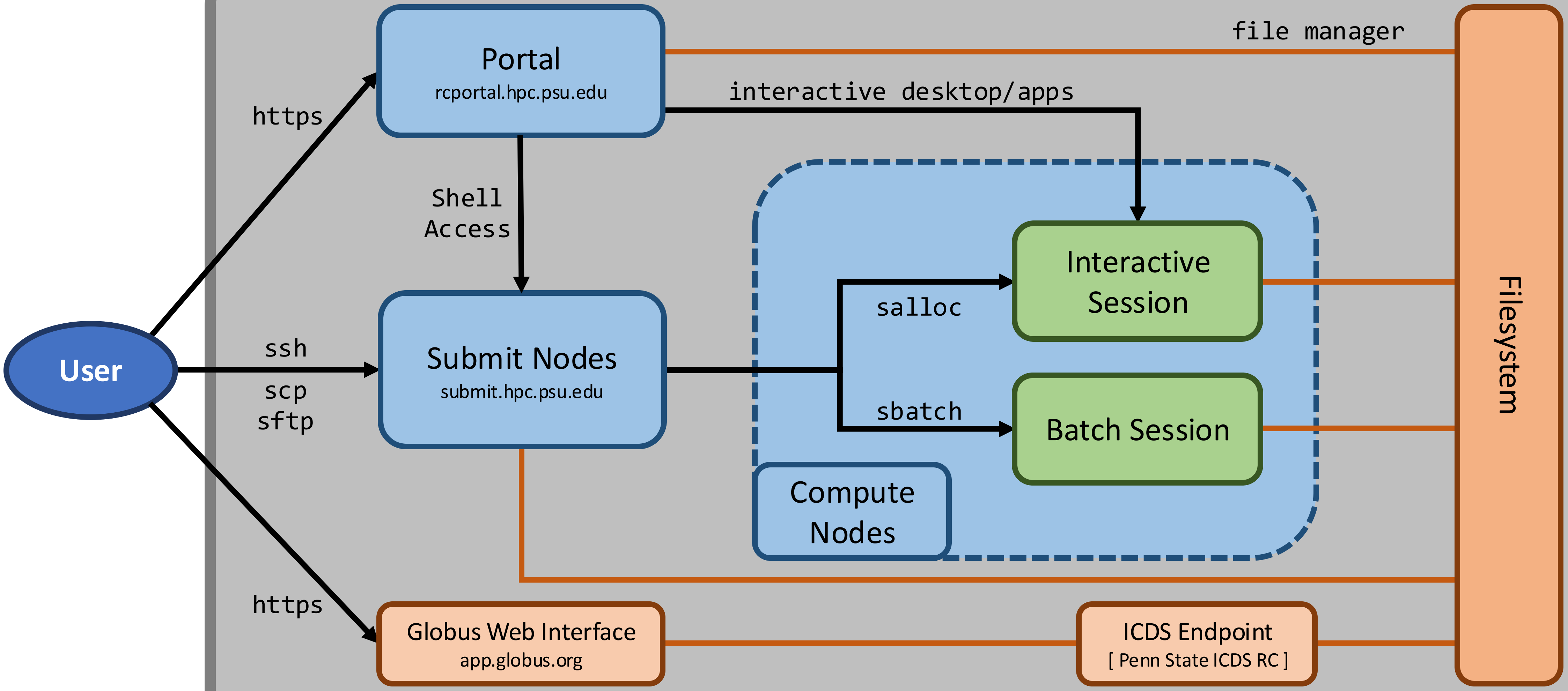
HANDS ON

Within the interactive compute session, navigate to your scratch directory with the following command:

`cd ~/scratch` OR `cd /scratch/<userid>`

Unzip the workshop files and enter the directory with

```
unzip rc_intro.zip
cd rc_intro
```



Roar Collab User Flow Diagram

Computing Workflow Overview

Computing Workflow Elements

- Data Storage and Data Transfer
- Accessing and Utilizing Software
- Performing Computational Tasks

RC Data Storage

Users can access personal storage locations and shared group storage locations from any node.

Storage	Location	Quota	Backup Policy	Use Case
Home	/storage/home	16 GB 500,000 files	Daily Snapshot	Configuration files
Work	/storage/work	128 GB 1 million files	Daily Snapshot	Primary user-level data storage
Scratch	/scratch	No space quota 1 million files	No backup Files purged after 30 days	Temporary, unimportant files
Group	/storage/group	Allocation-dependent	Daily Snapshot	Primary storage for shared data

Data Storage Information

To check access and storage usage information use the following command:

```
quota_check
```

Alternatively, you can check a directory directly using

```
df –h <storage-location>
```

To view a sorted breakdown of the sizes of files and directories within a storage location, use something like this:

```
du -sch .[!.]* <storage-location>/* | sort -h
```

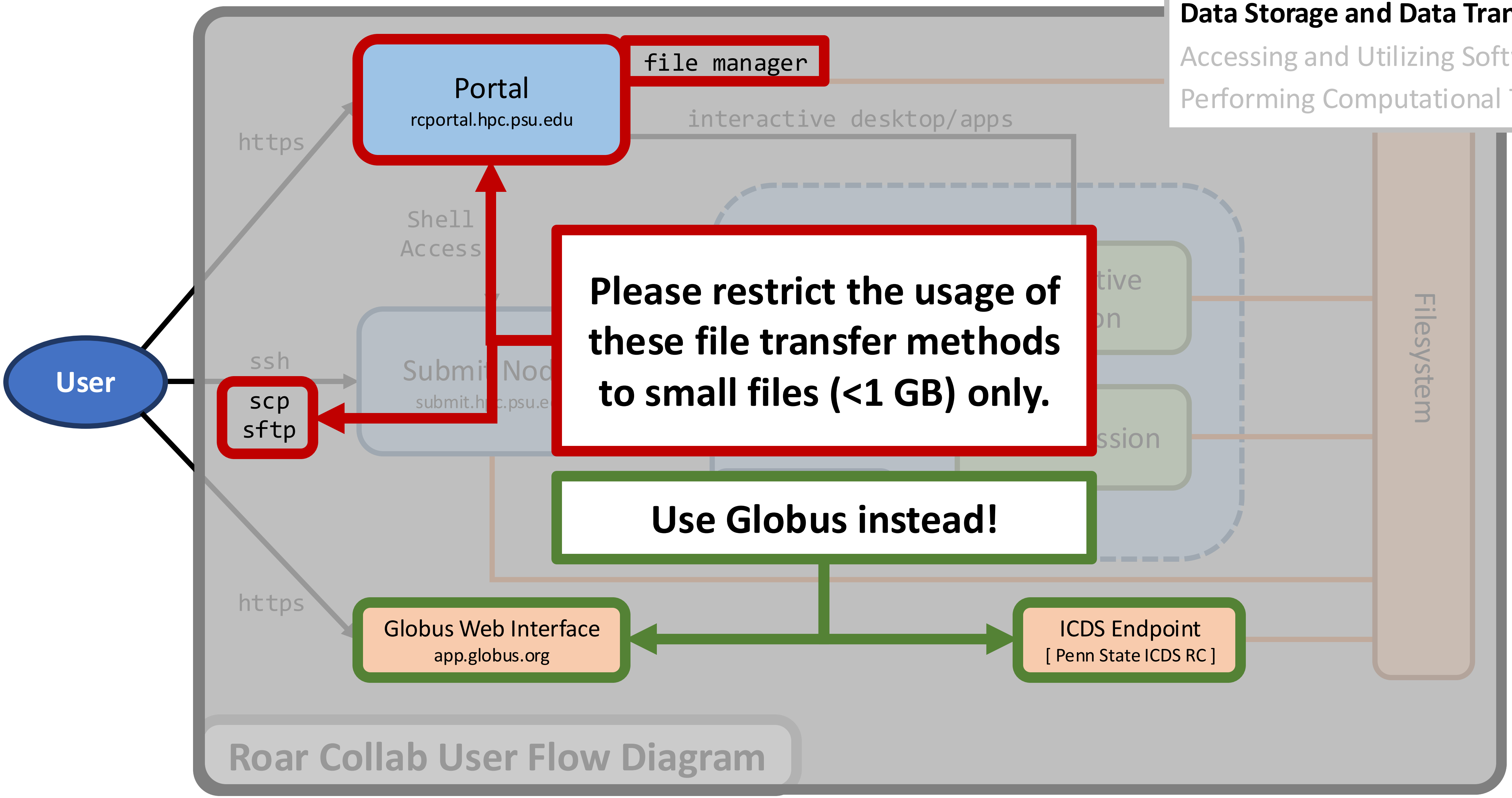
Managing Storage Accounts

Every user account on RC automatically comes with a **<userid>_collab** group. Users may submit a support ticket (via email to icds@psu.edu) to have a user added to their **<userid>_collab** group. This method of sharing / access management requires ICDS intervention for each and every change.

Alternatively, users can create their own user-managed group (UMG), submit a single request to ICDS to associate that UMG with their **<userid>_collab** group, and then manage access on their own via the UMG dashboard.

UMGs also enable more fine-grained access control for group storage locations.

Detailed instructions on this process are included in the Roar User Guide (docs.icds.psu.edu/accounts/managing-accounts/#user-managed-groups).


User
https
Portal
rcportal.hpc.psu.edu
file manager
interactive desktop/apps
Shell
Access
ssh
scp
sftp
Please restrict the usage of these file transfer methods to small files (<1 GB) only.
Use Globus instead!
https
Globus Web Interface
app.globus.org
ICDS Endpoint
[Penn State ICDS RC]
Filesystem
Roar Collab User Flow Diagram

Software Options

Along the arrow, there is an increase in the level of:

- Performance
- User Control
- User Responsibility
- Install Complexity

Software Stacks

Package Managers

Containerization

Install from Source

Software Stacks

The **system software stack** contains software that is installed at the system level and is available to all system users by default.

The supplemental **central software stack** is available for users to load and is more flexible and dynamic than the system software stack.

Central Software Stack

The central software stack uses Lmod to package available software.

Lmod is a useful tool for managing user software environments using **environment modules** that can be dynamically added or removed using module files.

Lmod is **hierarchical**, so sub-modules can be nested under a module that it is dependent upon.

Lmod alters environment variables, most notably the $PATH variable.

Lmod Commands

Command	Action
`module avail`	Lists all modules that are available to be loaded
`module show <module_name>`	Shows the contents of a module
`module spider <module_name>`	Searches the module space for a match
`module load <module_name>`	Loads a module or multiple modules if given a space-delimited list of modules
`module load <module>/<version>`	Loads a module of a specific version
`module unload <module_name>`	Unloads a module or multiple modules if given a space-delimited list of modules
`module list`	Lists all currently loaded modules
`module purge`	Unloads all currently loaded modules
`module use <path>`	Adds an additional path to $MODULEPATH to expand module scope
`module unuse <path>`	Removes a path from $MODULEPATH to restrict module scope

Package Managers

Package managers simplify software package installation and manage dependency relationships while increasing both the **repeatability** and the **portability** of your code.

The user environment is modified by the package manager so the shell can access different software packages.

Package managers alter the user environment at different levels.

- System-level (requires root): dnf, yum, apt
- Generic: Anaconda, Spack, Homebrew
- Software-specific: pip (python), CRAN (R), CPAN/CPANM (perl)

Anaconda (Conda)

Anaconda is a very useful package manager for software on ICDS systems. It was originally created for python, but it can package and distribute software for any language.

It is usually very simple to create and manage new environments, install new packages, and import/export environments. Many packages are available for installation through Anaconda.

Load Anaconda from the central software stack with the following command:

```
module load anaconda/2023
```

Anaconda allows you to keep your environments in a silo and reduce cross-dependencies between different packages that may perturb environments.

Anaconda Commands

Command	Action
`conda create -n <env_name>`	Creates a conda environment by name
`conda create -p <env_path>`	Creates a conda environment by location
`conda env list`	Lists all conda environments
`conda env remove -n <env_name>`	Removes a conda environment by name
`conda activate <env_name>`	Activates a conda environment by name
`conda list`	While in an active environment, lists all packages
`conda deactivate`	Deactivates the active conda environment
`conda install <package>`	While in an active environment, installs a package
`conda search <package>`	Uses conda to search for a package
`conda env export > env_name.yml`	Exports active environment to a file
`conda env create -f env_name.yml`	Loads environment from a file

Install Python Packages with pip

Python packages can be installed using pip with the following:

```
pip install --user <package>
```

The --user option installs the package in the ~/.local directory and must be used since pip wants to install the package to system locations.

Another option is to install to a specified directory with the following:

```
pip install --target <install_dir> <package>
```

RC is a **shared system**, so users have access to user locations but do not have access to system locations.

**Note: If `pip` is not available, try `pip3` (for python3) or `pip2` (for python2) instead.*

Software Versions on RC

Be sure to use consistent software versions, especially for Python and R or other software that relies on sub-packages.

RC has several versions available for many software modules, and when a package is installed for one version, it is not always accessible by another version.

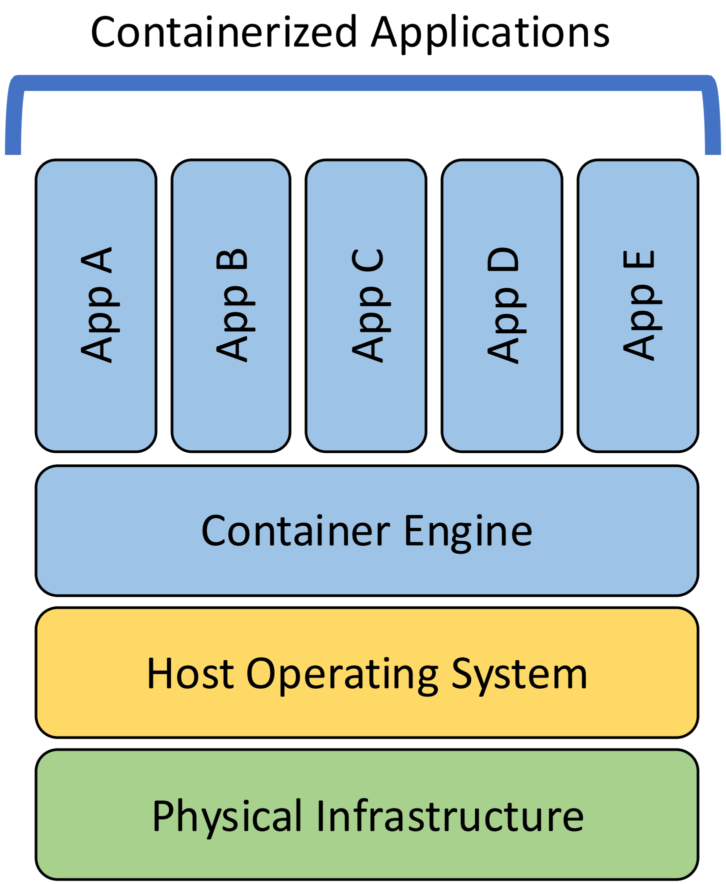
Always check the software version and remain consistent!

Container Basics

A **container** is a standard unit of software with two modes:

- **Idle**: When idle, a container is a file that stores everything an application (or collection of applications) requires to run (code, runtime, system tools, system libraries and settings).
- **Running**: When running, a container is a Linux process running on top of the host machine kernel with a user environment defined by the contents of the container file, not by the host OS.

A container is an abstraction at the application layer. Multiple containers can run on the same machine and share the host kernel with other containers, each running as isolated processes.



Apptainer / Singularity

Apptainer is a container platform designed for HPC use cases and is available on RC.

Containers (or **images**) can either be pulled directly from a container repository or can be built from a definition file.

A **definition file** or **recipe file** contains everything required to build a container. Building containers requires root privileges, so containers are built on your laptop and can be deployed on RC.

Apptainer Commands

Command	Action
apptainer build <container> <definition>	Builds a container from a definition file
apptainer shell <container>	Runs a shell within a container
apptainer exec <container> <command>	Runs a command within a container
apptainer run <container>	Runs a container where a runscript is defined
apptainer pull <resource>://<container>	Pulls a container from a container registry
apptainer build --sandbox <sbox> <container>	Builds a sandbox from a container
apptainer build <container> <sbox>	Builds a container from a sandbox

Custom Software from Source

It is somewhat common in academic computing to have to build software from source.

Building from source offers the greatest level of control and performance, but it requires the **highest level of involvement** from the user.

It can become difficult and time-consuming to wrangle dependencies and options.

Advice for Building Software

Read the install instructions provided by the developer!

Install guides can be in the the following forms:

- Webpage on the software's website
- Quickstart guide
- README file
- Under the `./configure --help` command
- Tutorials from web searches

Creating your own module files will help you manage your software environment more efficiently.

Processor Compatibility

RC is a heterogeneous cluster. To see the different node configurations on RC, use the following command:

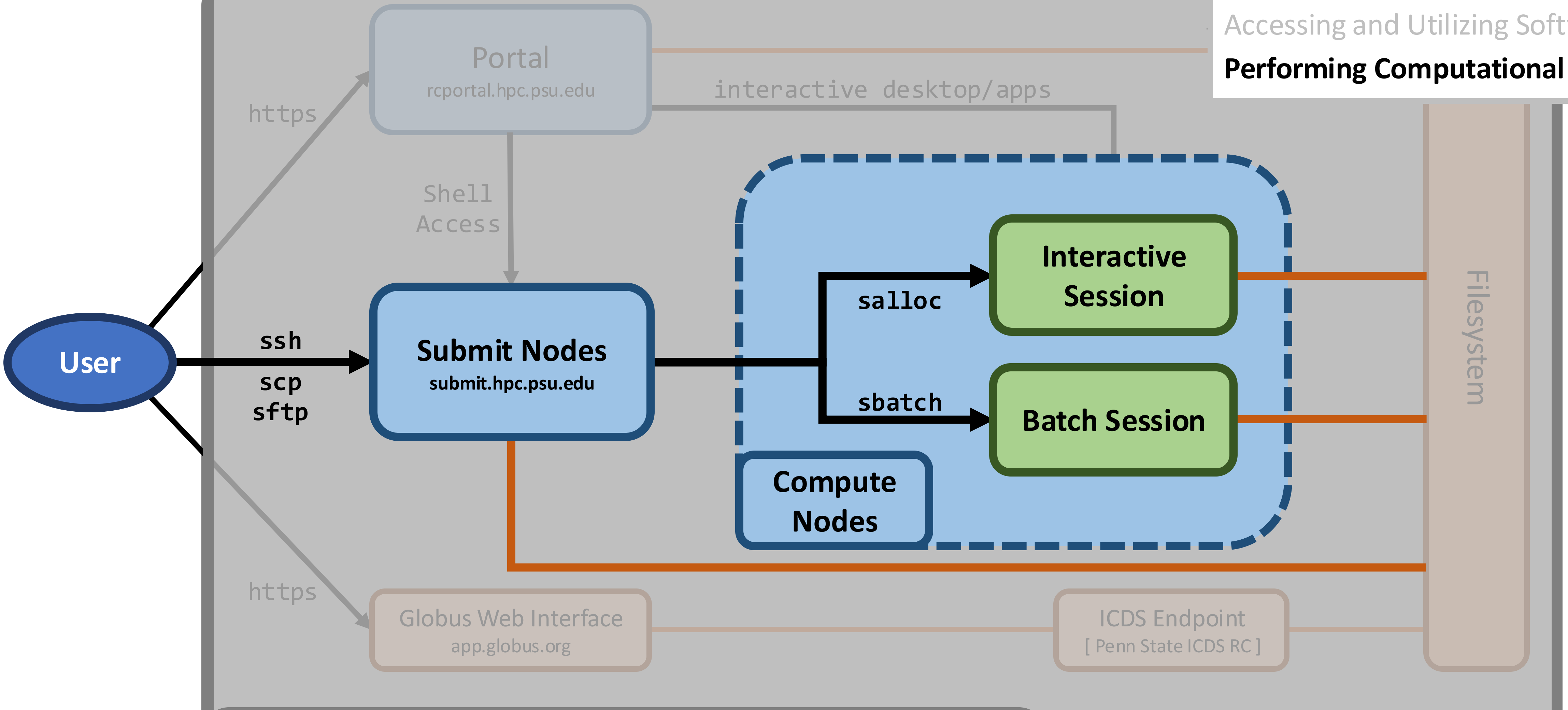
```
sinfo --Format=features:40,nodelist:40,cpus:5,memory:10,gres:30
```

On a compute node, running the following command displays the processor type:

```
cat /sys/devices/cpu/caps/pmu_name
```

Software builds are not typically back-compatible and will not run successfully on processors older than the processor used to build. It is recommended to build on haswell (the oldest processor architecture on RC) if you wish to have full compatibility across all RC compute nodes. To optimize for performance, however, build on the same processor on which the software runs.

Release Date	Processor
2013	haswell
2014	broadwell
2015	skylake
2019	cascadelake
2019	icelake
2023	sapphirerapids



RC Command Line Interface (CLI)

Slurm Job Submission

Requesting an interactive compute session can be accomplished using the `salloc` command.

Submitting a batch job with a submission script can be accomplished using the `sbatch` command.

Detailed information for each of these commands (salloc, sbatch) can be found in Slurm's documentation.

Common Resource Directives

Directive	Description
-J or --job-name	Specify a name for the job
-A or --account	Charge resources used by job to specified compute account
-p or --partition	Request a specific partition
-N or --nodes	Request the number of nodes for the job
-n, --ntasks, or --ntasks-per-node	Request the number of tasks for the job
--gres=gpu[:type]:1	Request a GPU per node, and optionally select the GPU type
-C or --constraint	Specify any required node features for job
--mem or --mem-per-cpu	Specify the memory required per node or CPU, respectively
-t or --time	Set a limit on the total run time of the job
--requeue	Specify that the batch job should be eligible for requeuing
-e or --error	Instruct Slurm to connect the batch script's standard error to a non-default file
-o or --output	Instruct Slurm to connect the batch script's standard output to a non-default file

Compute Accounts & Partitions

There are two categories of paid compute resources available on RC, a **credit account** or a **reserved allocation**.

A credit account allows users to purchase credits and then use those credits across various compute resources. The credit accounts are only charged when a job on that account is occupying compute resources. A reserved allocation offers users unlimited and immediate access to a specifically defined set of resources, essentially occupying the resources around the clock.

The selected partition determines the class of resources used and also determines the credit consumption rate for that job.

For credit rates and pricing information, visit the following page:
icds.psu.edu/roar-restricted-and-roar-collab-price-lists-2025

Compute Account Information

To check access, credit balance, and additional information for compute accounts, use the following command:

```
get_balance
```

For reserved allocations, the reserved resources are displayed, and the credit balance is displayed for compute credits accounts.

The help menu for this command shows more options that enable searching for a specific compute account, other user’s access, and per-user usage for credit accounts.

Managing Compute Accounts

Account coordinators can add/remove other user and coordinators. The account owner is automatically designated as an account coordinator, but they can appoint other users to serve as coordinators.

Add/remove a user:

```
sacctmgr add user account=<compute-account> name=<userid>
sacctmgr remove user account=<compute-account> name=<userid>
```

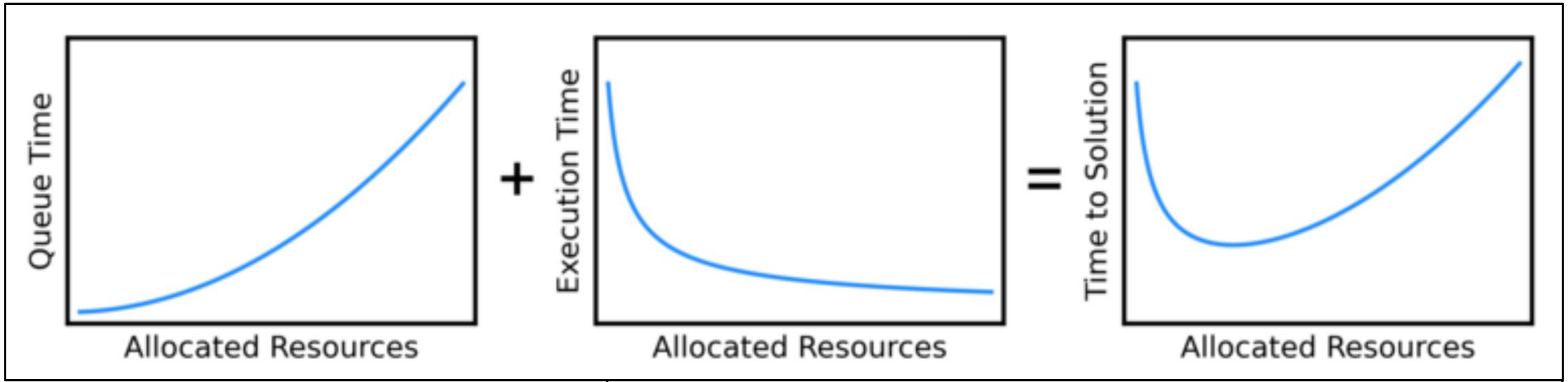
Add/remove a coordinator:

```
sacctmgr add coordinator account=<compute-account> name=<userid>
sacctmgr remove coordinator account=<compute-account> name=<userid>
```

Requesting Resources

Requesting more resources for a job will increase its time in the queue as it must wait longer for resources to be allocated.

Typically, parallelized code will see a reduction in overall runtime as more resources are requested.



https://researchcomputing.princeton.edu/support/knowledge-base/scaling-analysis

The total time to solution is ultimately reduced when the job strikes a balance by requesting the minimal amount of resources needed to provide a reasonable speedup.

Putting It All Together

Access Submit Node

HANDS ON

Choose one of two options to initiate a Command Line Interface session on a submit node:

- From a terminal session, run the following command:

  `ssh <userid>@submit.hpc.psu.edu`

- Access portal.hpc.psu.edu and from the banner select

  `Clusters > RC Shell Access`

Then navigate to the *rc_intro* directory created previously:

`cd /scratch/<userid>/rc_intro`

Interactive Compute Session

HANDS ON

To launch an interactive session on a compute node, run the following command a submit node:

```
salloc -N 2 -n 6 -A open -t 2:00:00
```

- Command to allocate compute resources
- Number of nodes
- Number of tasks
- Account
- Max time of session

Note that the prompt changes from a submit node type to a compute node type after the resources are granted.

Session Distribution

HANDS ON

Our interactive compute session was started with this command:

```
salloc -N 2 -n 6 -A open -t 2:00:00
```

From the *rc_intro* directory, run the *checknodefile.sh* script with the following:

```
bash checknodefile.sh
```

This script shows the hosts on which each of the tasks are running. Note that there is no guarantee as to the distribution of the tasks across the allocated nodes.

If we desire a specific number of tasks per node, we can use the *--ntasks-per-node* option instead of *-n* so the tasks will be evenly assigned to the nodes.

For just a list of hosts, use the following

```
scontrol show hostname
```

Software Environment

HANDS ON

Load necessary software from the software stack:

```
module load anaconda/2023
```

Install the necessary conda packages using the provided environment file:

```
conda create -y -n rc_intro_env numpy mpi4py
```

This stage may take a few minutes. After it completes, we can leave the interactive compute session with the `exit` command.

Submit A Batch Job

```
#!/bin/bash

#SBATCH --job-name=pi_serial         #### give the job a name
#SBATCH --account=open               #### specify the account
#SBATCH --partition=open             #### specify the partition
#SBATCH --nodes=1                    #### request a node
#SBATCH --ntasks=1                   #### request a task / cpu
#SBATCH --mem=1G                     #### request the memory required per node
#SBATCH --time=00:30:00              #### set a limit on the total run time
#SBATCH --output=out/%x-%j.out       #### specify the name of job's output file
#SBATCH --error=out/%x-%j.err        #### specify the name of job's error file


start_time=$(date +%s)

module load anaconda/2023
module list
conda activate rc_intro_env

python pi_serial.py > out/${SLURM_JOB_NAME}-${SLURM_JOB_ID}.pyout

end_time=$(date +%s)
duration=$(( end_time - start_time ))
echo "Job Duration:  $duration  sec"
```

The submission script can then be submitted using the following command: `sbatch submit-pi_serial.slurm`

Submit A Batch Job

```
#!/bin/bash
```

Shebang

```
#SBATCH --job-name=pi_serial        #### give the job a name
#SBATCH --account=open              #### specify the account
#SBATCH --partition=open            #### specify the partition
#SBATCH --nodes=1                   #### request a node
#SBATCH --ntasks=1                  #### request a task / cpu
#SBATCH --mem=1G                    #### request the memory required per node
#SBATCH --time=00:30:00             #### set a limit on the total run time
#SBATCH --output=out/%x-%j.out      #### specify the name of job's output file
#SBATCH --error=out/%x-%j.err       #### specify the name of job's error file
```

Scheduler/ Resource Directives

```
start_time=$(date +%s)

module load anaconda/2023
module list
conda activate rc_intro_env

python pi_serial.py > out/${SLURM_JOB_NAME}-${SLURM_JOB_ID}.pyout

end_time=$(date +%s)
duration=$(( end_time - start_time ))
echo "Job Duration:  $duration  sec"
```

Job Payload

The submission script can then be submitted using the following command: `sbatch submit-pi_serial.slurm`

Filename Pattern Symbols

Both standard output and standard error are directed to the same file by default, and the file name is "slurm-%j.out", where the "%j" is replaced by the job ID.

The output and error filenames are customizable, however.

Symbol	Description
%j	Job ID
%x	Job name
%u	Username
%N	Hostname where the job is running
%A	Job array's master job allocation number
%a	Job array ID (index) number

For example:

```
#SBATCH --output=%x-%j.out
#SBATCH --error=%x-%j.err
```

Common Environment Variables

Variable	Description
SLURM_JOB_ID	ID of the job
SLURM_JOB_NAME	Name of job
SLURM_NNODES	Number of nodes
SLURM_NODELIST	List of nodes
SLURM_NTASKS	Total number of tasks
SLURM_NTASKS_PER_NODE	Number of tasks per node
SLURM_QUEUE	Queue (partition)
SLURM_SUBMIT_DIR	Directory of job submission

Monitor A Job

Find out what node this job is running on with

```
squeue -u <userid>
```

A useful environment variable is the SQUEUE_FORMAT variable and can be set, for example, with the following command:

```
export SQUEUE_FORMAT="%.9i %9P %35j %.8u %.2t %.12M %.12L %.5C %.7m %.4D %R"
```

Further details on the usage of this variable are available in Slurm's squeue documentation.

Another useful job monitoring command is:

```
scontrol show job <jobid>
```

Cancel a job with:

```
scancel <jobid>
```

Monitor Job On Compute Node

Valuable information can be obtained by monitoring a job on the compute node as the job runs.

Connect to the compute node of a running job with the `ssh` command:

```
ssh <comp-node-id>
```

After connecting to the compute node, the `top` and `ps` commands are useful tools.

```
ssh <comp-node-id>
top –Hu <userid>
ps –aux | grep <userid>
```

Usage Notes for `top` and `ps`

Within the top command, use the *f* key to select different column options.

Common Statuses	
R	Running
S	Interruptible Sleep (waiting for event)
D	Uninterruptible Sleep
T	Stopped
Z	Zombie
+	Foreground
s	Session Leader

Monitor A Job

HANDS ON

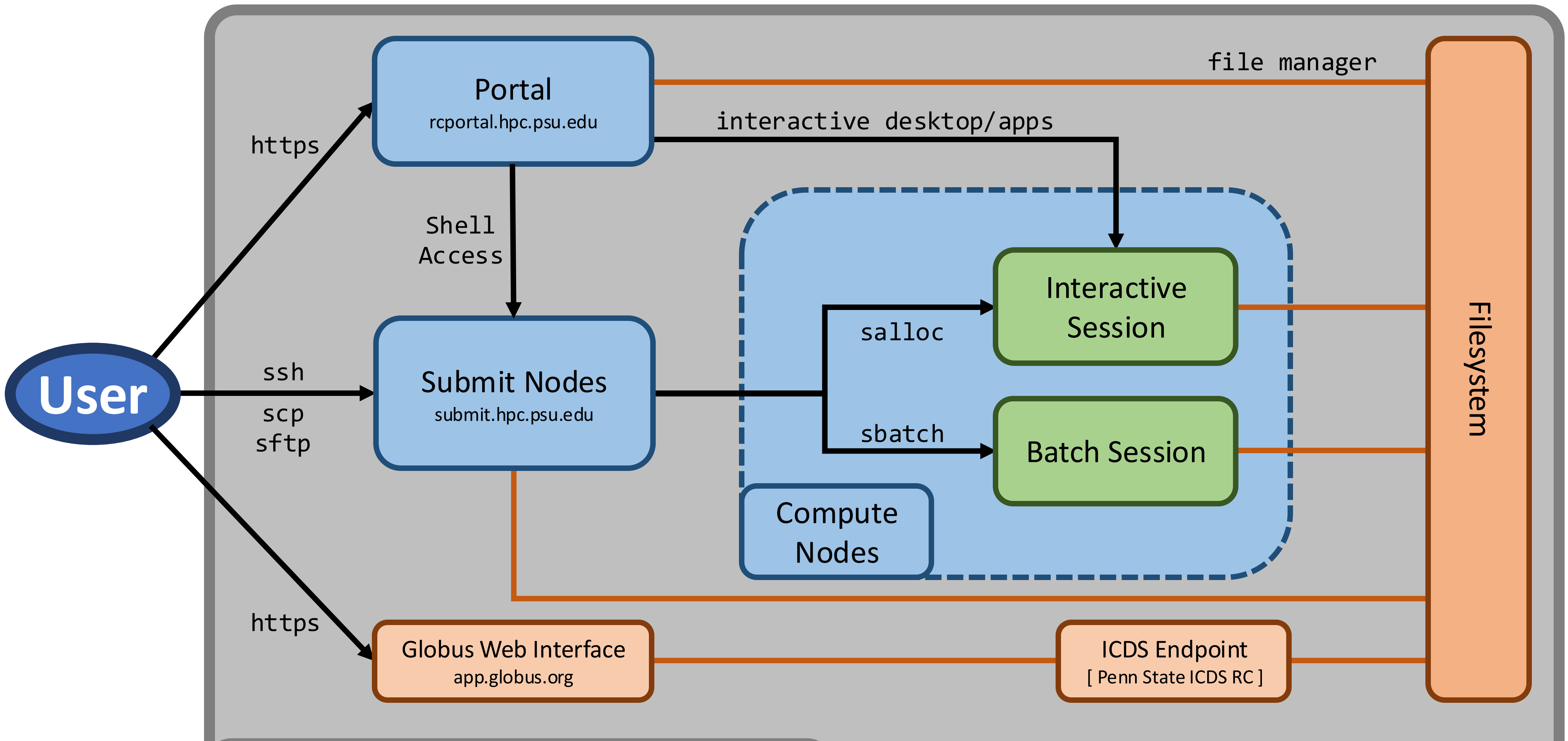
On a submit node enter the *rc_intro/pi_example* directory and launch some jobs:

```
cd pi_example
sbatch submit-pi_serial.slurm
sbatch submit-pi_parallel.slurm
```

Find out what node this job is running on with

```
squeue -u <userid>
```

Navigate to the compute node and use `top` and `ps` to monitor the running job.



Roar Collab User Flow Diagram

Watch for upcoming events on the ICDS Events page at icds.psu.edu/events!

RC User Guide

docs.icds.psu.edu

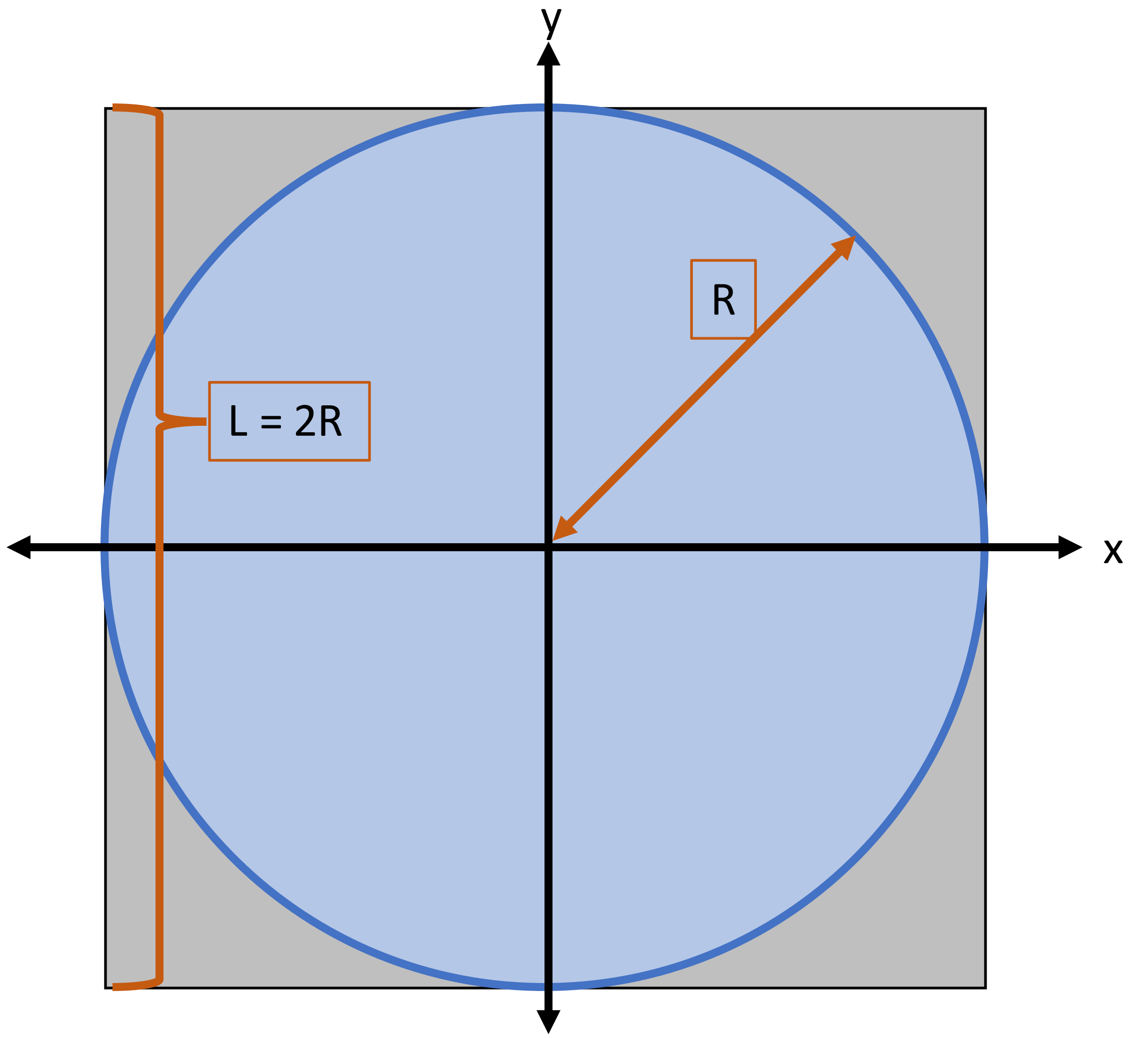
For technical support,

email us at icds@psu.edu

Supplemental Slides

Probabilistic Estimation of Pi Example Explainer

Probabilistically Estimate Pi



A circle is perfectly drawn within a square such that there are four points of intersection between the two shapes.

Area of the Circle:

$$C = \pi R^2$$

Area of the Square:

$$S = L^2$$

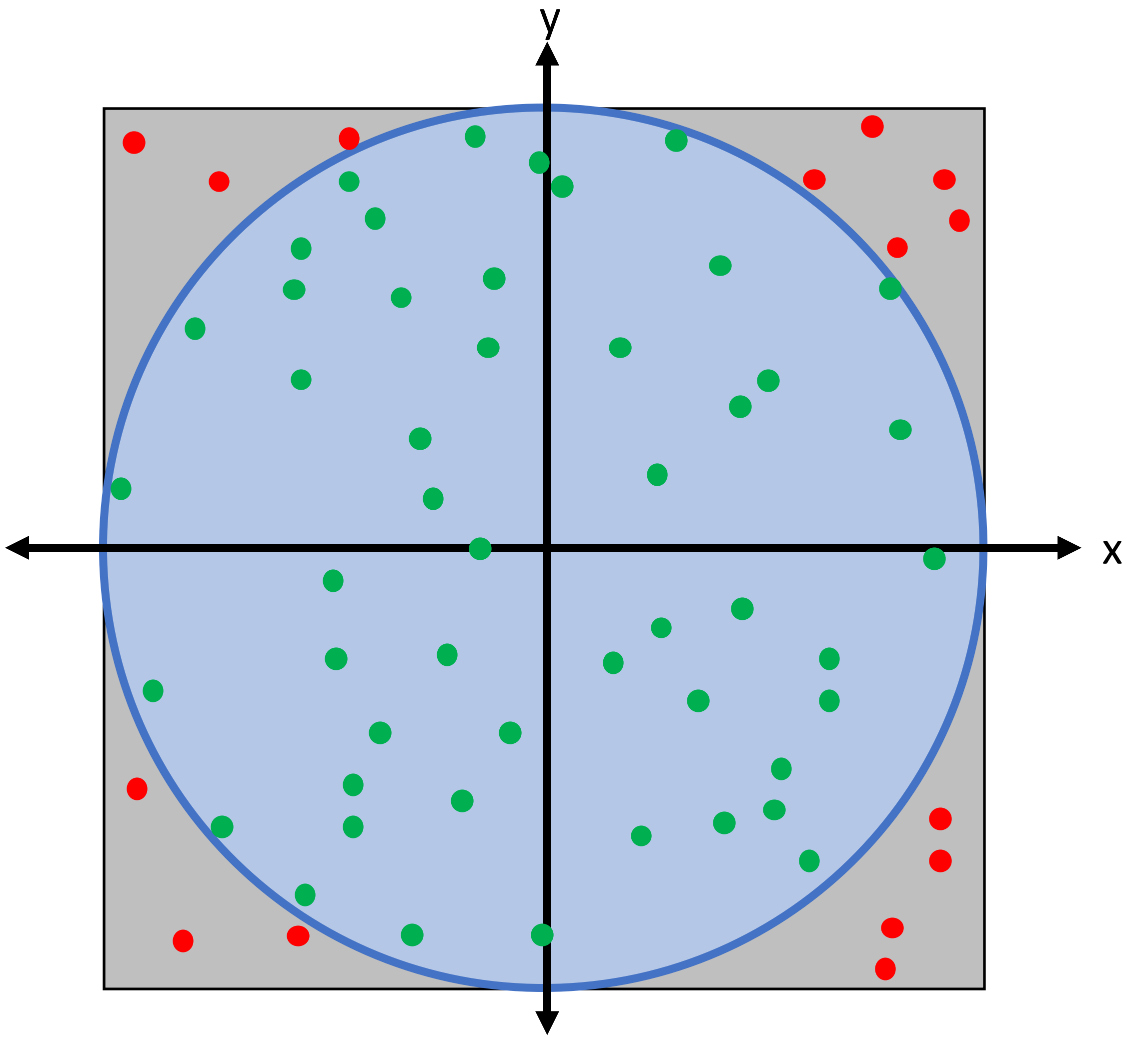
Since $L = 2R$, make a substitution.

$$S = L^2 = (2R)^2 = 4R^2$$

Determine the proportion of the square area within the circle.

$$P = \frac{C}{S} = \frac{\pi R^2}{4R^2} = \frac{\pi}{4}$$

Probabilistically Estimate Pi



Simply rearrange the result on the last slide to produce an expression for pi.

$$\pi = 4P$$

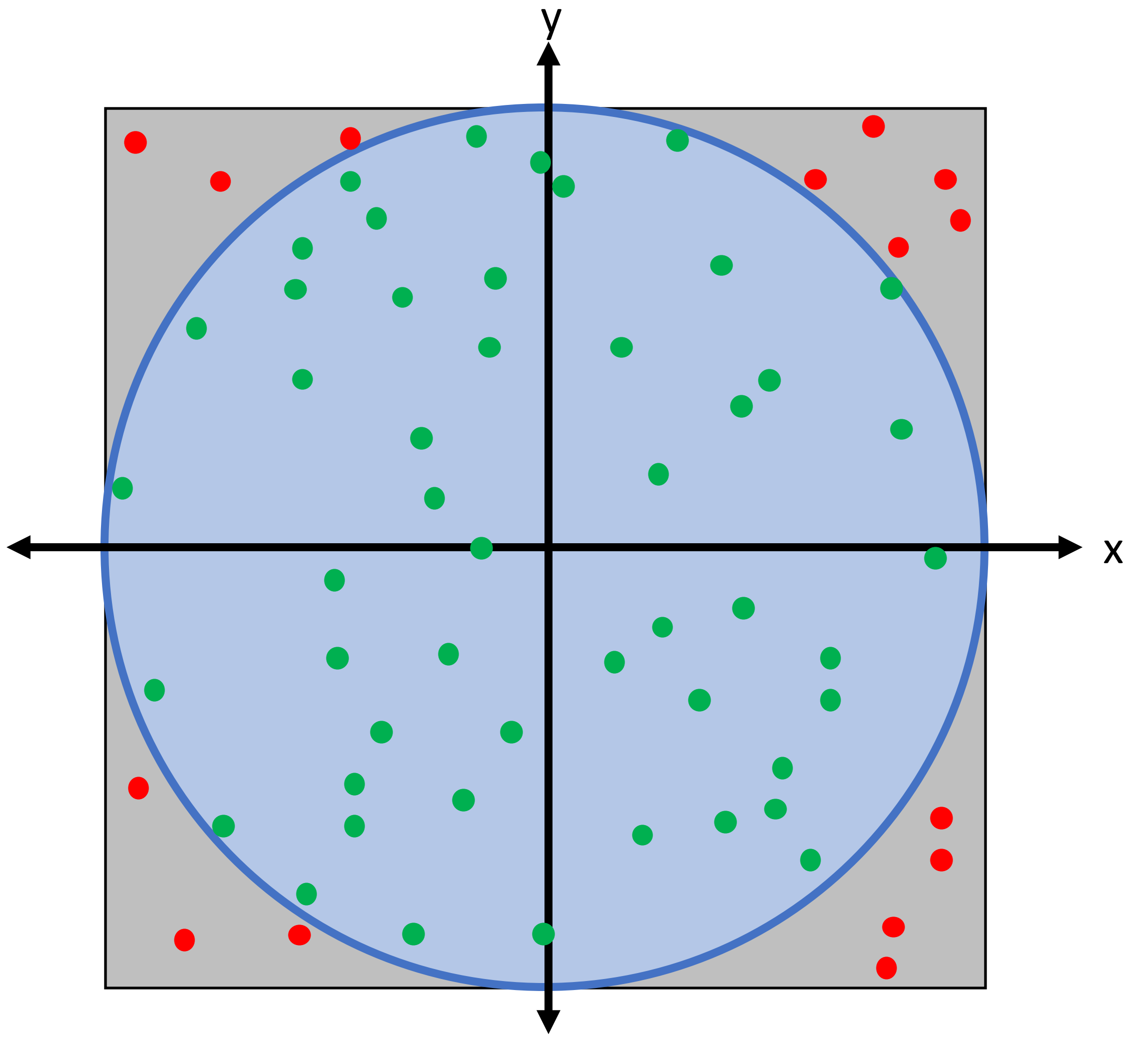
If we randomly select many points within the square and check if they lie within the circle or not, we can produce an estimate of P, called P^*, where

$$P^* = \frac{\#\ points\ within\ circle}{\#\ of\ total\ points\ checked}$$

Using this estimate of the proportion of the square's area that is within the circle, a probabilistic estimate of pi, π^*, is produced.

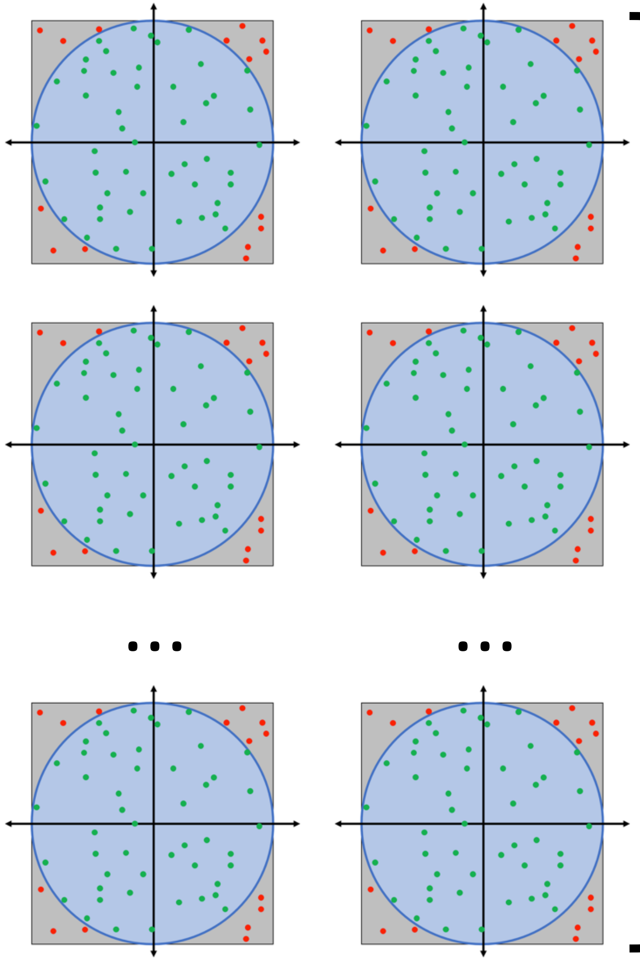
$$\pi^* = 4P^*$$

Serial Execution



If executing this task on a single processor, each individual point must be selected and checked one by one in a serial manner.

Parallel Execution



$\sum points$

If executing this task across multiple processors in parallel, the total number of points can be divided and evenly distributed.

Each individual processor will perform its own subtask serially, but only on a smaller portion of the total points.

Probabilistically Estimate Pi

Using MPI to implement this computational process allows it to be distributed across not only multiple processors, but also across multiple nodes.

The total number of points is divided up and distributed to the worker processes by the root process. The worker processes report the local result back to the root process when finished, and the root process sums up the all the workers' results to produce the total result.

This computational process can be described as an "embarrassingly parallel" process since the individual workers are independent of one another. They only must communicate with the root process when starting up and when complete. This is an example of multi-tasking.