CONDA Environments and Jupyter Notebook on Expanse: Scalable & Reproducible Data Exploration and ML

Peter Rose (pwrose@ucsd.edu)
Director, Structural Bioinformatics Lab

COMPUTING WITHOUT BOUNDARIES

EXPANSE

SAN DIEGO SUPERCOMPUTER CENTER

UNIVERSITY OF CALIFORNIA SAN DIEGO

Outline

- When to run on Expanse
- Setup a reproducible and portable software environment
- Use the Expanse Portal
- Run Jupyter Lab on Expanse
- Scale up calculations on CPU/GPU
- Create a packed Conda environment
- Run Jupyter Lab in batch
- Get ready to use Expanse: accounts, allocations
- Best Practices for Authoring Jupyter Notebooks

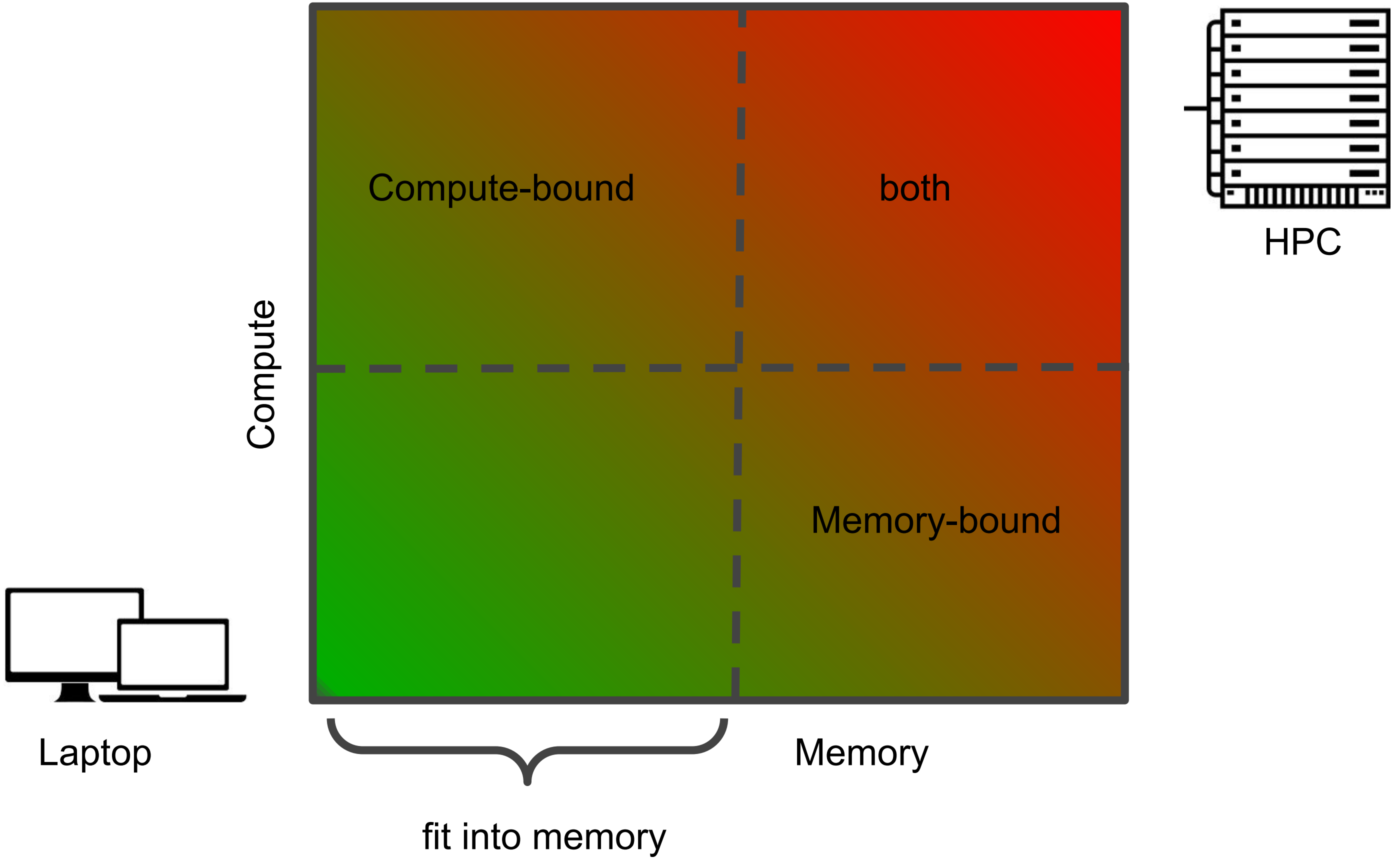
When to run on Expanse

Laptop/Desktop

- Coding
- Exploratory phase
- Small datasets
- Run on single or few cores

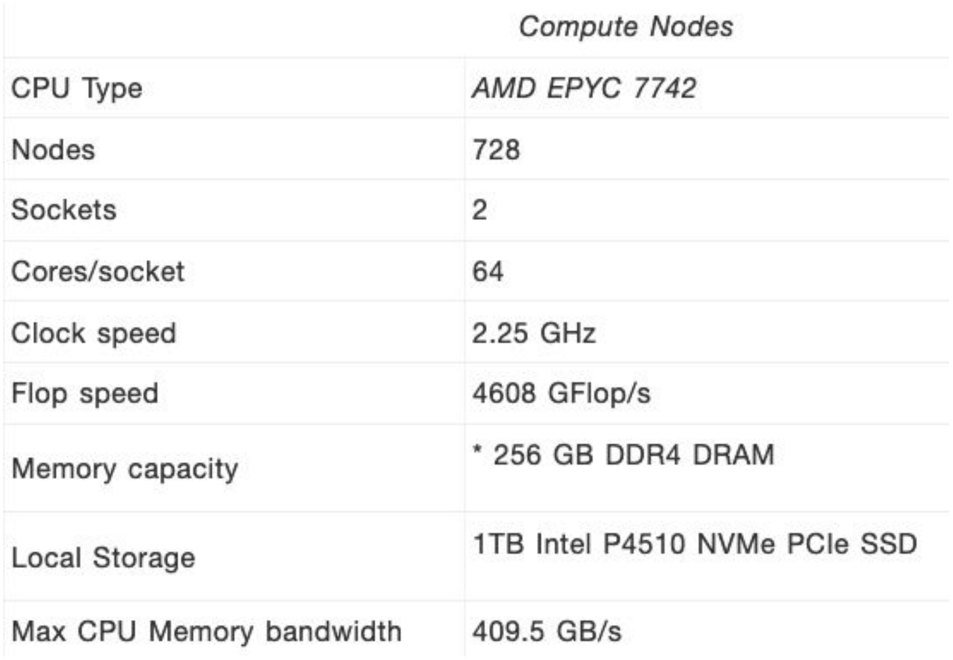
Expanse

- Scaling up to
  - large datasets
  - long runtimes
- Run on many cores
- Run on GPU


Compute-bound
both
HPC
Compute
Memory-bound
Laptop
Memory
fit into memory

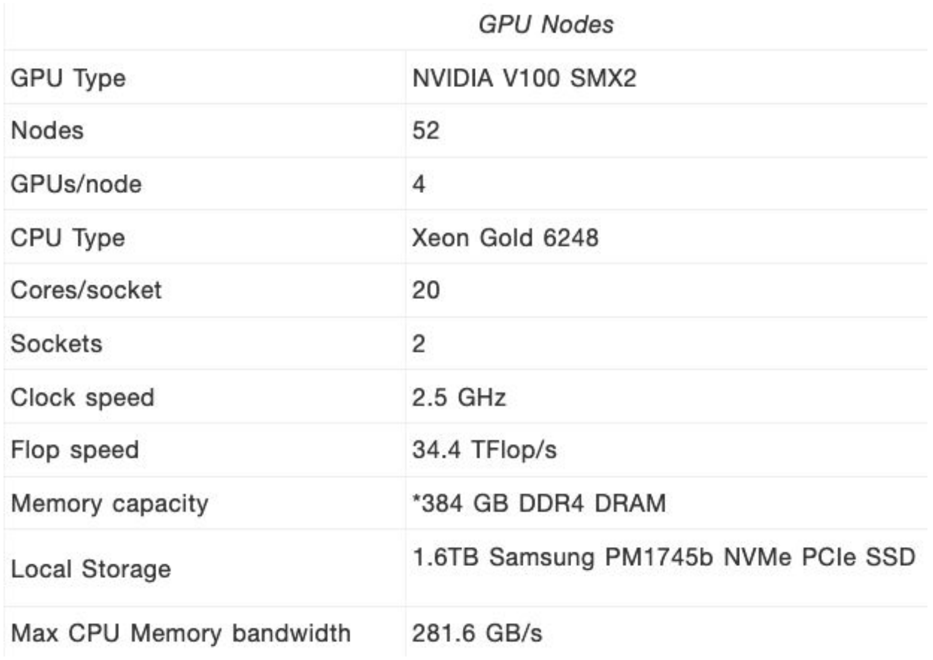
Expanse Nodes

128 CPU cores/node



	Compute Nodes
CPU Type	*AMD EPYC 7742*
Nodes	728
Sockets	2
Cores/socket	64
Clock speed	2.25 GHz
Flop speed	4608 GFlop/s
Memory capacity	* 256 GB DDR4 DRAM
Local Storage	1TB Intel P4510 NVMe PCIe SSD
Max CPU Memory bandwidth	409.5 GB/s

4 GPUs, 40 CPU cores/node



	GPU Nodes
GPU Type	NVIDIA V100 SMX2
Nodes	52
GPUs/node	4
CPU Type	Xeon Gold 6248
Cores/socket	20
Sockets	2
Clock speed	2.5 GHz
Flop speed	34.4 TFlop/s
Memory capacity	*384 GB DDR4 DRAM
Local Storage	1.6TB Samsung PM1745b NVMe PCIe SSD
Max CPU Memory bandwidth	281.6 GB/s

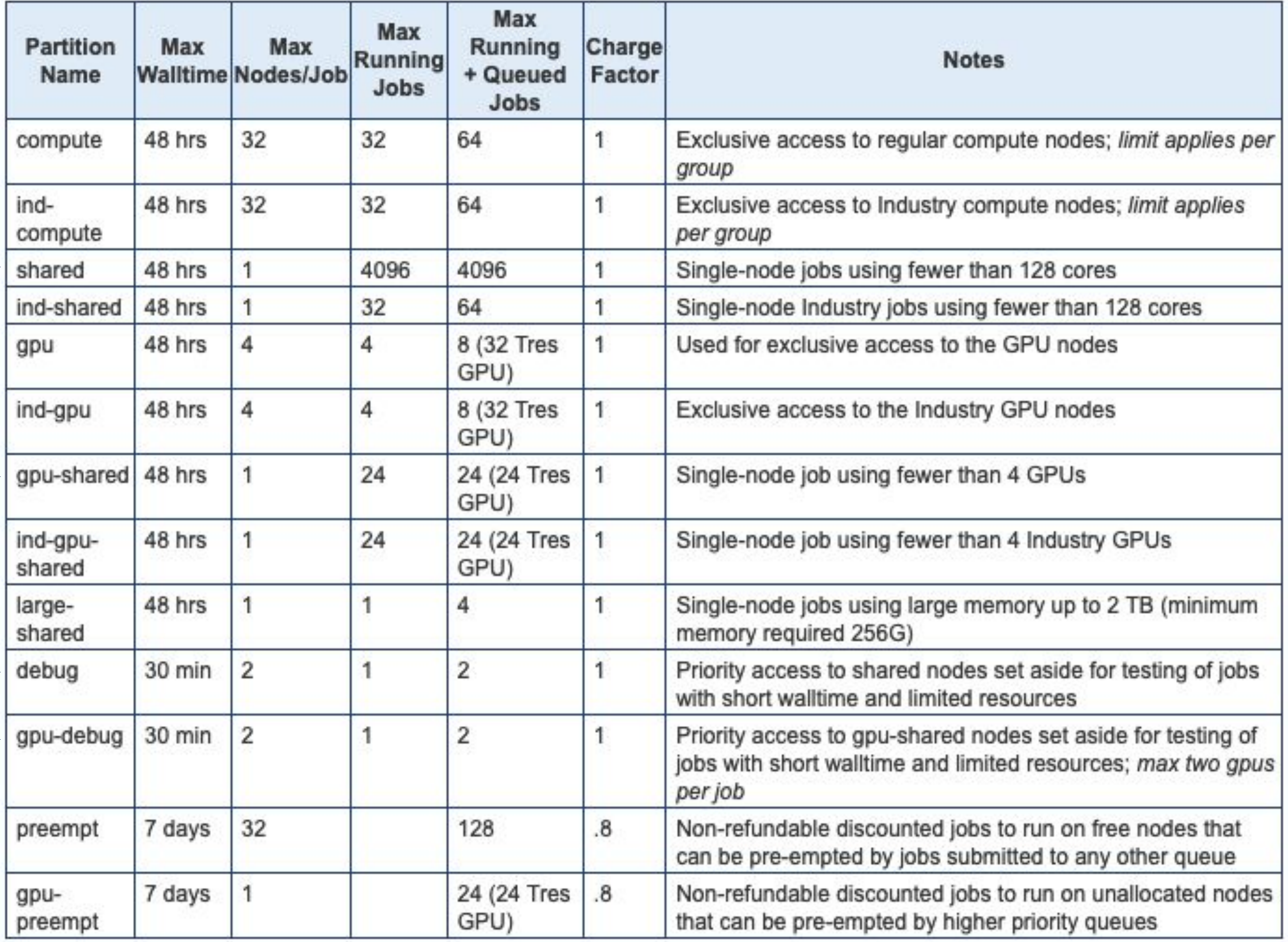
Details: https://portal.xsede.org/sdsc-expanse

Jupyter Notebook (CPU)

Jupyter Notebook (GPU)

Testing (CPU)

Testing (GPU)



Partition Name	Max Walltime	Max Nodes/Job	Max Running Jobs	Max Running + Queued Jobs	Charge Factor	Notes
compute	48 hrs	32	32	64	1	Exclusive access to regular compute nodes; *limit applies per group*
ind-compute	48 hrs	32	32	64	1	Exclusive access to Industry compute nodes; *limit applies per group*
shared	48 hrs	1	4096	4096	1	Single-node jobs using fewer than 128 cores
ind-shared	48 hrs	1	32	64	1	Single-node Industry jobs using fewer than 128 cores
gpu	48 hrs	4	4	8 (32 Tres GPU)	1	Used for exclusive access to the GPU nodes
ind-gpu	48 hrs	4	4	8 (32 Tres GPU)	1	Exclusive access to the Industry GPU nodes
gpu-shared	48 hrs	1	24	24 (24 Tres GPU)	1	Single-node job using fewer than 4 GPUs
ind-gpu-shared	48 hrs	1	24	24 (24 Tres GPU)	1	Single-node job using fewer than 4 Industry GPUs
large-shared	48 hrs	1	1	4	1	Single-node jobs using large memory up to 2 TB (minimum memory required 256G)
debug	30 min	2	1	2	1	Priority access to shared nodes set aside for testing of jobs with short walltime and limited resources
gpu-debug	30 min	2	1	2	1	Priority access to gpu-shared nodes set aside for testing of jobs with short walltime and limited resources; *max two gpus per job*
preempt	7 days	32		128	.8	Non-refundable discounted jobs to run on free nodes that can be pre-empted by jobs submitted to any other queue
gpu-preempt	7 days	1		24 (24 Tres GPU)	.8	Non-refundable discounted jobs to run on unallocated nodes that can be pre-empted by higher priority queues

Setup a reproducible and portable software environment

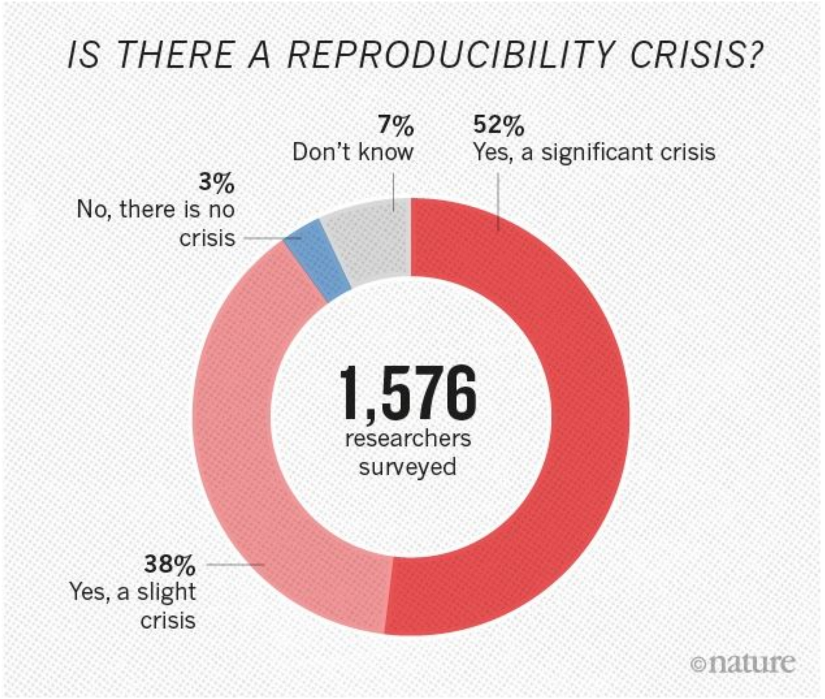
Reproducibility Crisis?

"More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments."

Nature, 2016, M. Baker, 1,500 scientists lift the lid on reproducibility

"Nature journal editors … will, on a case-by-case basis, ask reviewers to check how well the code works."

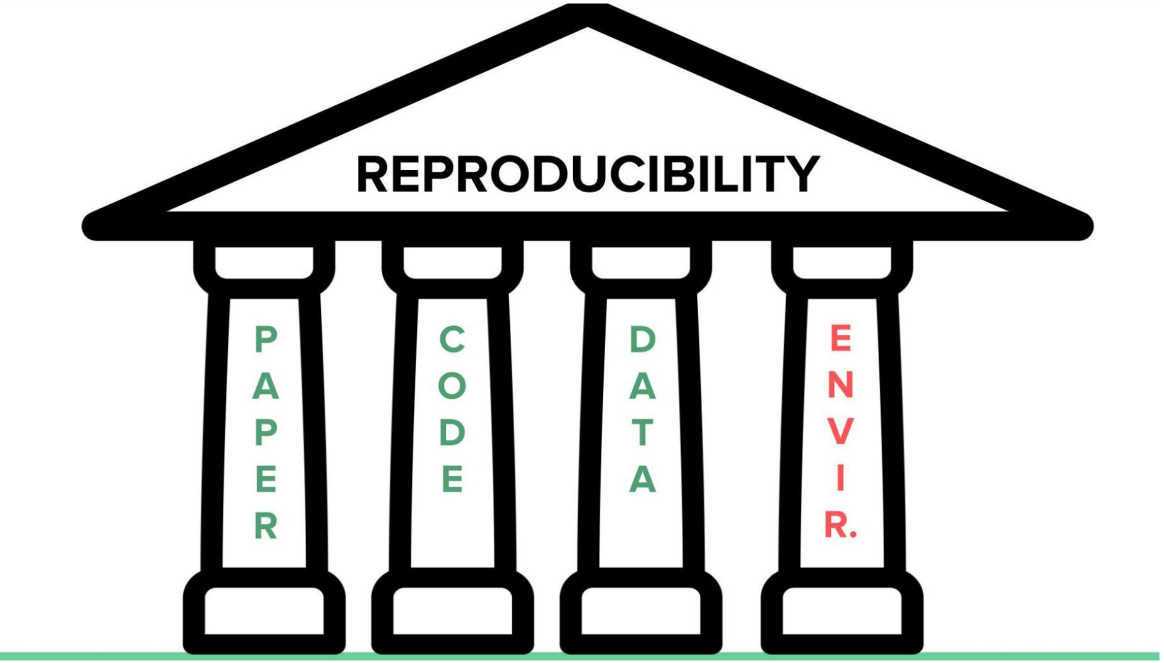
Nature, 2018, Does your code stand up to scrutiny?



Reproducibility*	**Reusability**	**Scalability**
obtaining **consistent** results using	obtaining **new** results using	obtaining **new** results using
same input data or parameters	**different** input data or parameters	**large** input data or parameter sets
same computational steps, methods, and code	**same** computational steps, methods, and code	**same** computational steps, methods, and code
same analysis conditions	**same** analysis conditions	**same** analysis conditions

* L. Barba, https://figshare.com/articles/Next_in_Reproducibility_standards_policies_infrastructure_and_human_factors/8194328/1

Four Pillars of Reproducible Research



http://theoryandpractice.org/2016/05/Reproducibility-Symposium/

Open Science

- Open access publications
- Open source code
- Open data
- Open execution environment

Tools and Infrastructure

jupyter — Computational notebooks: combine documentation, code, and results

git — Version-control system for tracking changes in source code

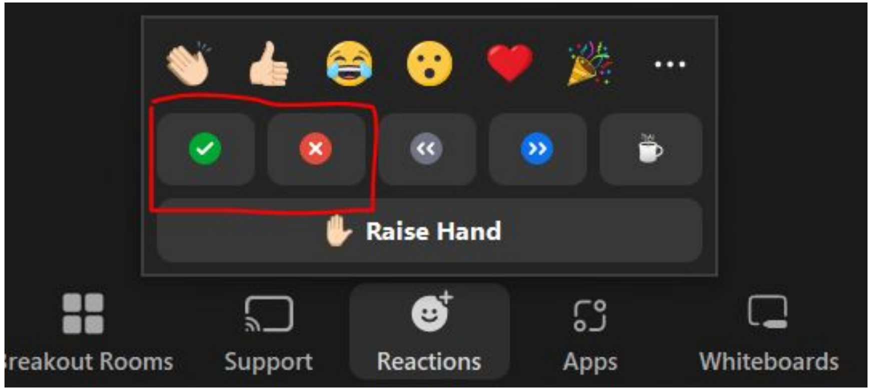
GitHub — Source code repository

CONDA — Open-source package and environment management system

SINGULARITY — Container that packages software and OS in a portable way

EXPANSE — Scalable compute infrastructure

Feedback using Zoom Reactions



We will use Reactions to get feedback during the hands-on exercises

- Yes ✓ — I've successfully completed the task
- No ✗ — I have a problem
(go to Slack and describe your problem or raise your hand)

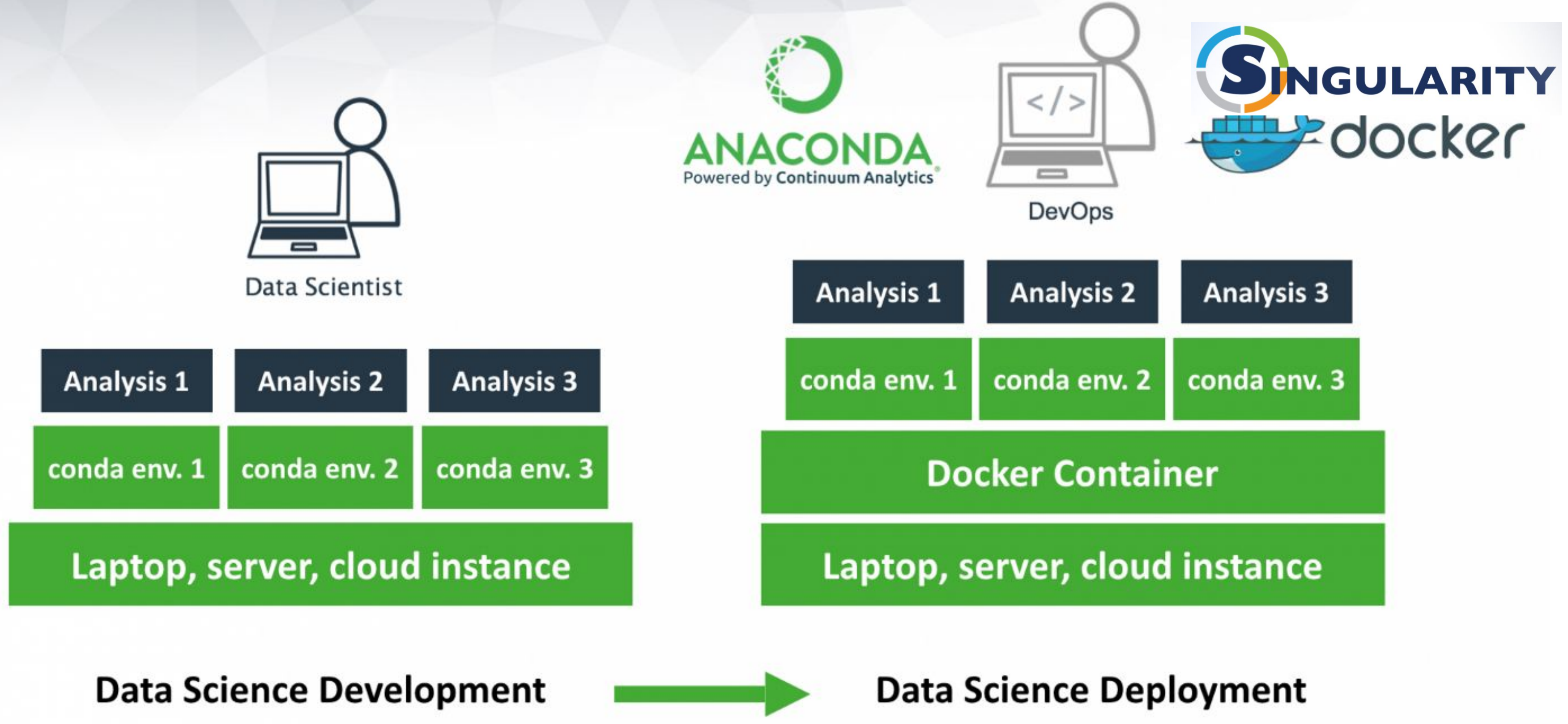
Reproducible Environments

CONDA

- Beginner
- Experience with Conda
- Exploratory development
- Frequently changing dependencies
- Easy to compose an environment with many dependencies
- Run on single node on Expanse
- Supported on Linux, Mac, Windows
- Run on native OS

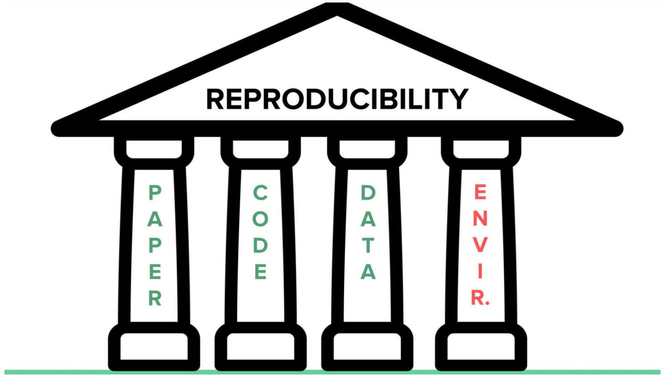
SINGULARITY

- Advanced user
- Experience with containers
- Production environment
- Often setup for a single tool
- Optimized containers for Expanse
  - pytorch, tensorflow, …
  - support for multi-node
- Supported on Linux
- Mac, Windows requires a VM
- Run on packaged OS, e.g. Ubuntu



Source: https://medium.com/@patrickmichelberger/getting-started-with-anaconda-docker-b50a2c482139

CONDA®



- **Package management system**
  - Conda installs, runs, and updates open source packages (e.g., NumPy, Pandas) and their dependencies, while checking compatibility with all preexisting packages.
- **Environment management system**
  - Conda allow you to create, save, load and switch between multiple environments on your local computer, as well as share instructions for how to recreate that environment on a different computer.
- **Multi-platform (Windows, MacOS, and Linux)**
- **Multi-language (Python, R, Ruby, Scala, Java, JavaScript, C/ C++, etc.)**

Why Conda Environments?

```
pip install pandas
pip install scikit-learn

Or

conda install pandas
conda install scikit-learn
```

Directly installing packages into your base environment will lead to version conflicts, errors, and non-reproducible results.

```
environment_1

python=3.7
pandas=0.25.0
scikit-learn=0.20.0
```

```
environment_2

python=3.9
pandas=1.2.4
scikit-learn=0.24.2
```

By creating conda environments, multiple versions of software packages can co-exists without interference.

Conda environment are portable and can be installed on multiple platforms.

Define a Conda Environments

Create an **environment.yml** file in the top level of a Git Repository (https://github.com/pwrose/df-parallel)

```
name: df-parallel

channels:
  - conda-forge
  - anaconda

dependencies:
  - python=3.8
  - jupyterlab=3
  - dask=2022.3.0
  - pyspark=3.2.1
  - openjdk=8.0.152

variables:
  # SPARK conf directory contains logging
  SPARK_CONF_DIR: ../conf
```

Use the same name as your Git repository

Specify the channels where to look for packages. Order matters! The conda-forge channel has newer versions than anaconda.

Specify (**“pin”**) version number to ensure reproducibility and compatibility.

Specify non-Python packages (e.g., Java).

Set environment variables (e.g., configuration options).

Create a Conda Environment

Prerequisite: Miniconda3 (light-weight, preferred) or Anaconda3 installed

https://docs.conda.io/en/latest/miniconda.html

Create a Conda environment

```
conda env create -f environment.yml
    or
mamba env create -f environment.yml (faster)
```

Activate a Conda environment

```
conda activate <environment_name>
```

Run Jupyter Lab

```
jupyter lab
```

Deactivate conda environment

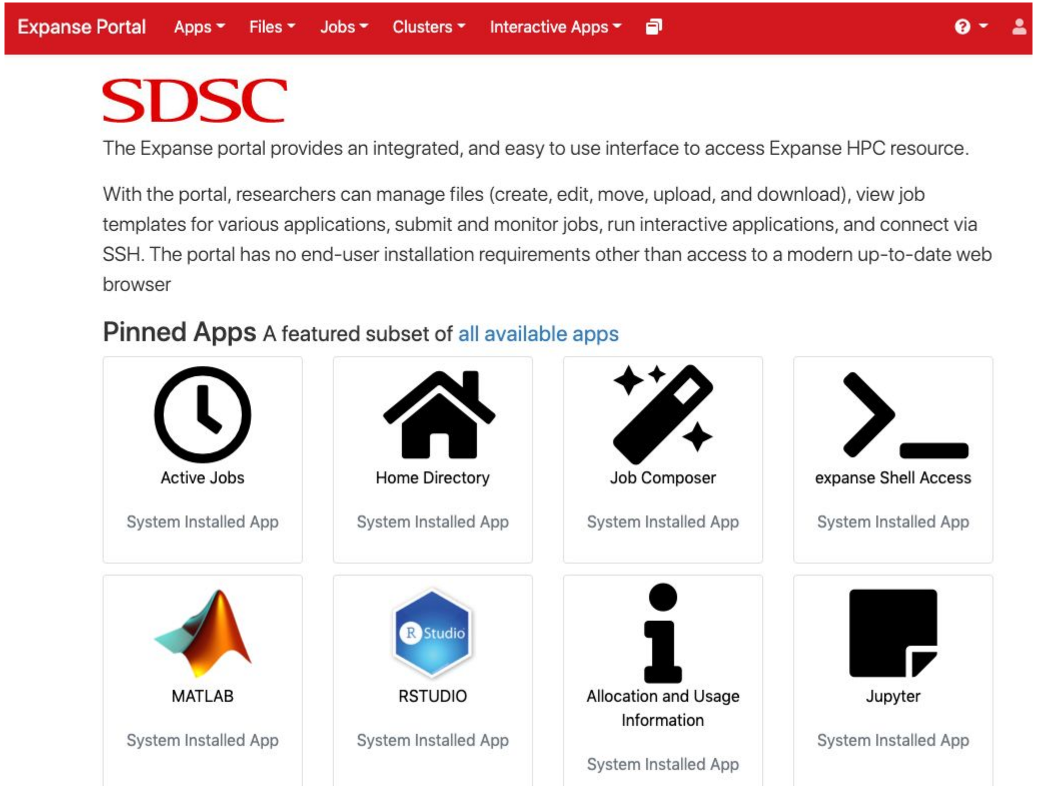
```
conda deactivate
```

✓ Mac, Windows, Linux

✗ Expanse: **Do not** create a Conda environment in your home directory (network file system)
-> Use the **galyleo** script!

Use the Expanse Portal to check allocations, job status, manage files, open a terminal window

Expanse Portal



Expanse Portal Apps Files Jobs Clusters Interactive Apps

SDSC

The Expanse portal provides an integrated, and easy to use interface to access Expanse HPC resource.

With the portal, researchers can manage files (create, edit, move, upload, and download), view job templates for various applications, submit and monitor jobs, run interactive applications, and connect via SSH. The portal has no end-user installation requirements other than access to a modern up-to-date web browser

Pinned Apps A featured subset of all available apps

Active Jobs
System Installed App

Home Directory
System Installed App

Job Composer
System Installed App

expanse Shell Access
System Installed App

MATLAB
System Installed App

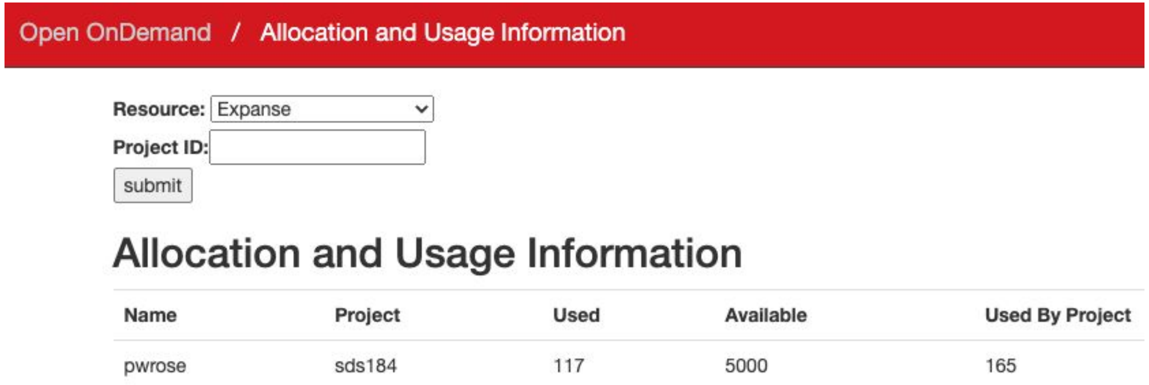
RSTUDIO
System Installed App

Allocation and Usage Information
System Installed App

Jupyter
System Installed App

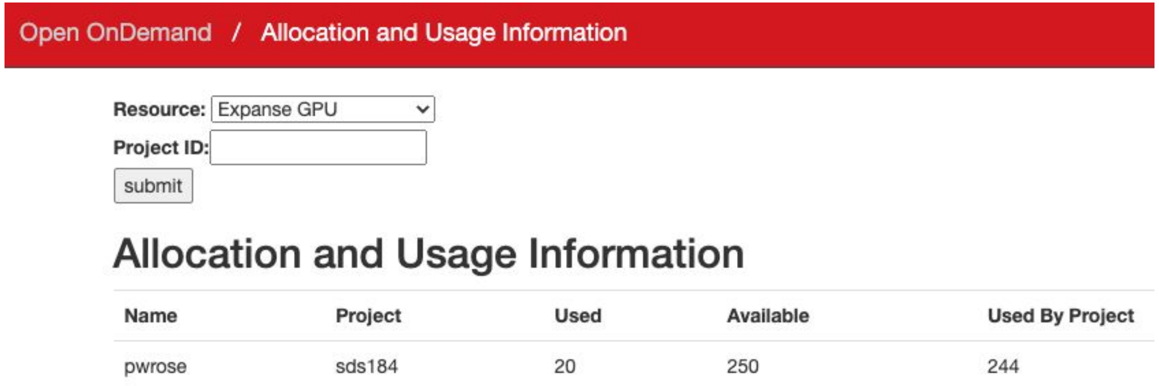
Login with your XSEDE credentials (trainxx):
https://portal.expanse.sdsc.edu/

Expanse Allocations CPU/GPU



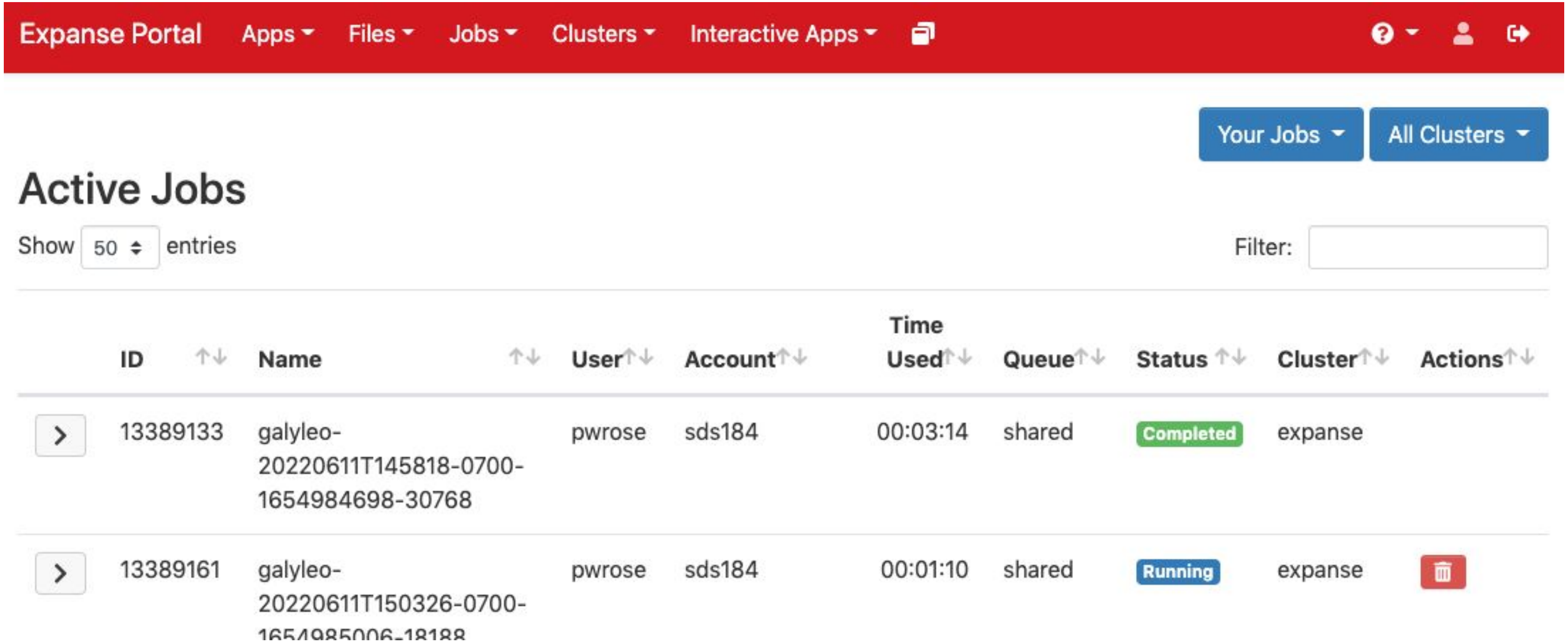
Allocation and Usage Information

Name	Project	Used	Available	Used By Project
pwrose	sds184	117	5000	165



Allocation and Usage Information

Name	Project	Used	Available	Used By Project
pwrose	sds184	20	250	244



Active Jobs

Show 50 entries Filter:

	ID	Name	User	Account	Time Used	Queue	Status	Cluster	Actions
>	13389133	galyleo-20220611T145818-0700-1654984698-30768	pwrose	sds184	00:03:14	shared	Completed	expanse	
>	13389161	galyleo-20220611T150326-0700-1654985006-18188	pwrose	sds184	00:01:10	shared	Running	expanse	🗑

Run Jupyter Lab on Expanse

Galyleo Script

Launches Jupyter Lab/Notebook on high-performance computing (HPC) systems.

Establishes an HTTPS-secured connection between the notebook server and your web browser.

Documentation: https://github.com/mkandes/galyleo

See also: Marty Kandes' webinar:
https://education.sdsc.edu/training/interactive/202112_running_jupyter_notebooks_on_expanse/index.html

Using Galyleo

1. Prepend path to galyleo to your path (e.g., add to `.bash_profile` file)

```
export PATH="/cm/shared/apps/sdsc/galyleo:${PATH}"
```

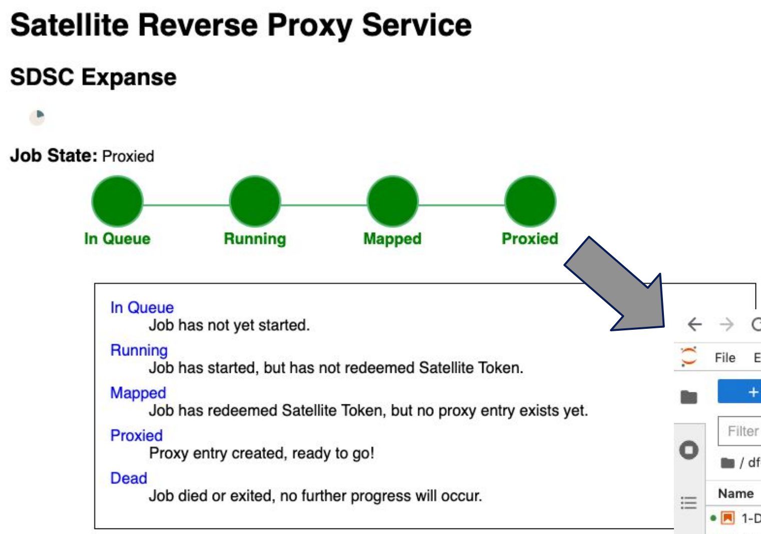
2. Launch your Jupyter Notebook session using a Conda environment.yml file

```
galyleo launch --account <account_number> --partition shared
--cpus 10 --memory 20 --time-limit 00:30:00 --conda-env
df-parallel --conda-yml "${HOME}/df-parallel/environment.yml"
--mamba
```

3. Copy and paste generated URL into your web browser

```
https://anchovy-passion-placidly.expanse-user-content.sdsc.edu?
token=48ee984b9ea07a96c17aaec000bc5fcf
```

Progress Bar and Jupyter Launch



Satellite Reverse Proxy Service

SDSC Expanse

Job State: Proxied

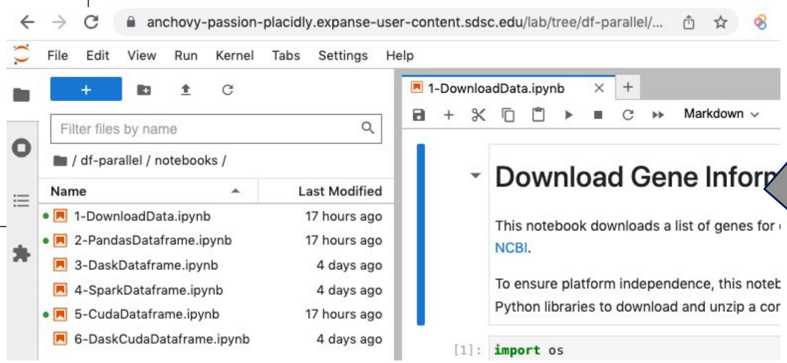
In Queue
Job has not yet started.

Running
Job has started, but has not redeemed Satellite Token.

Mapped
Job has redeemed Satellite Token, but no proxy entry exists yet.

Proxied
Proxy entry created, ready to go!

Dead
Job died or exited, no further progress will occur.



anchovy-passion-placidly.expanse-user-content.sdsc.edu/lab/tree/df-parallel/...
File
Edit
View
Run
Kernel
Tabs
Settings
Help
Filter files by name
/ df-parallel / notebooks /
1-DownloadData.ipynb
Markdown

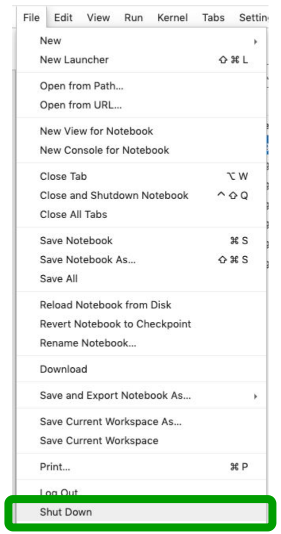
Name	Last Modified
1-DownloadData.ipynb	17 hours ago
2-PandasDataframe.ipynb	17 hours ago
3-DaskDataframe.ipynb	4 days ago
4-SparkDataframe.ipynb	4 days ago
5-CudaDataframe.ipynb	17 hours ago
6-DaskCudaDataframe.ipynb	4 days ago

Download Gene Infor

This notebook downloads a list of genes for NCBI.

To ensure platform independence, this noteb
Python libraries to download and unzip a cor

```
[1]: import os
```



File-> Shut Down to terminate process!

Running the Dataframe Examples

Clone the Git repo

```
git clone https://github.com/sbl-sdsc/df-parallel.git
```

Run on CPU (Pandas, Dask, Spark dataframes)

```
galyleo launch --account <account_number> --partition shared --cpus 10
--memory 20 --time-limit 01:00:00 --conda-env df-parallel
--conda-yml "${HOME}/df-parallel/environment.yml" --mamba
```

Run on GPU (Pandas, Dask, Spark, cuDF, Dask-cuDF dataframes)

```
galyleo launch --account <account_number> --partition gpu-shared --cpus 10
--memory 92 --gpus 1 --time-limit 01:00:00 --conda-env df-parallel-gpu
--conda-yml "${HOME}/df-parallel/environment_gpu.yml" --mamba
```

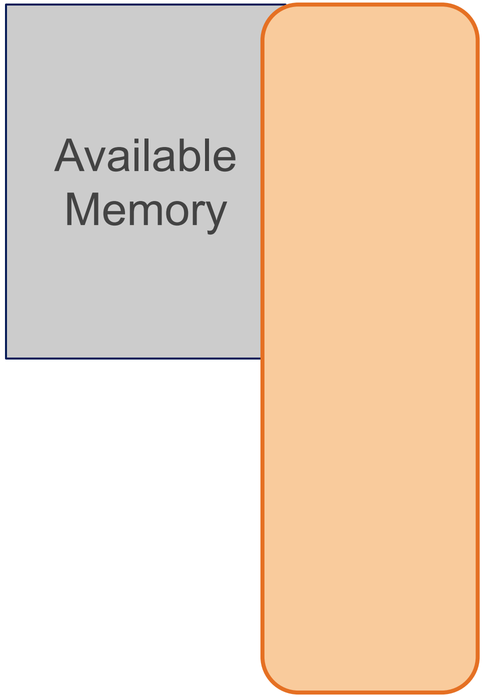
Task 1A

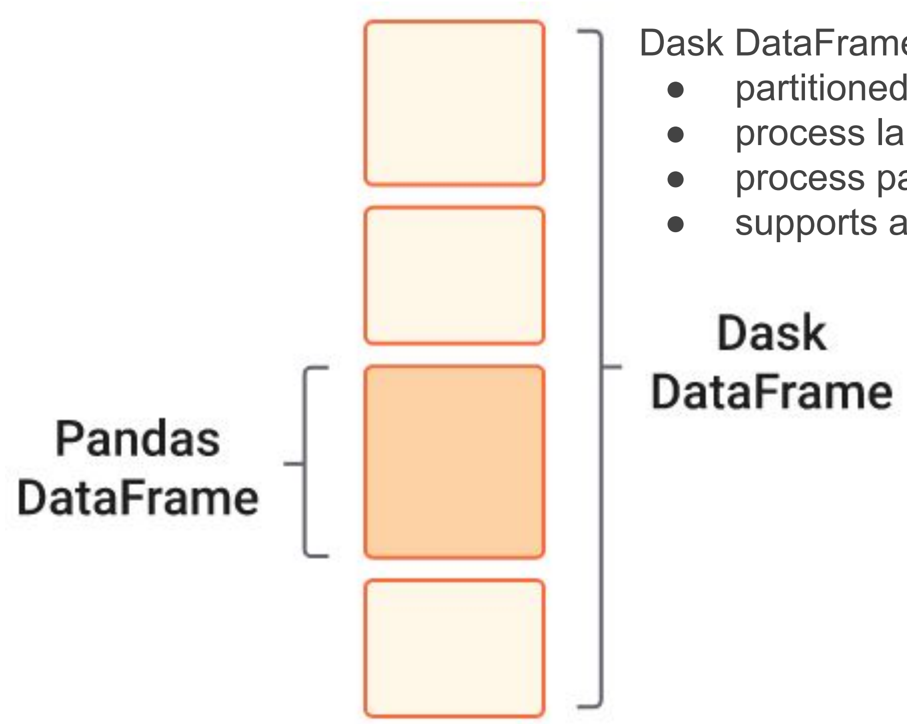
- Clone df-parallel Git repository
- Start galyleo on a GPU node
- Copy the Jupyter Lab URL into your browser

- Follow the instructions in section 3.3:
  https://github.com/ciml-org/ciml-summer-institute-2022

Scale up calculations on CPU/GPU

Processing large Datasets on CPU



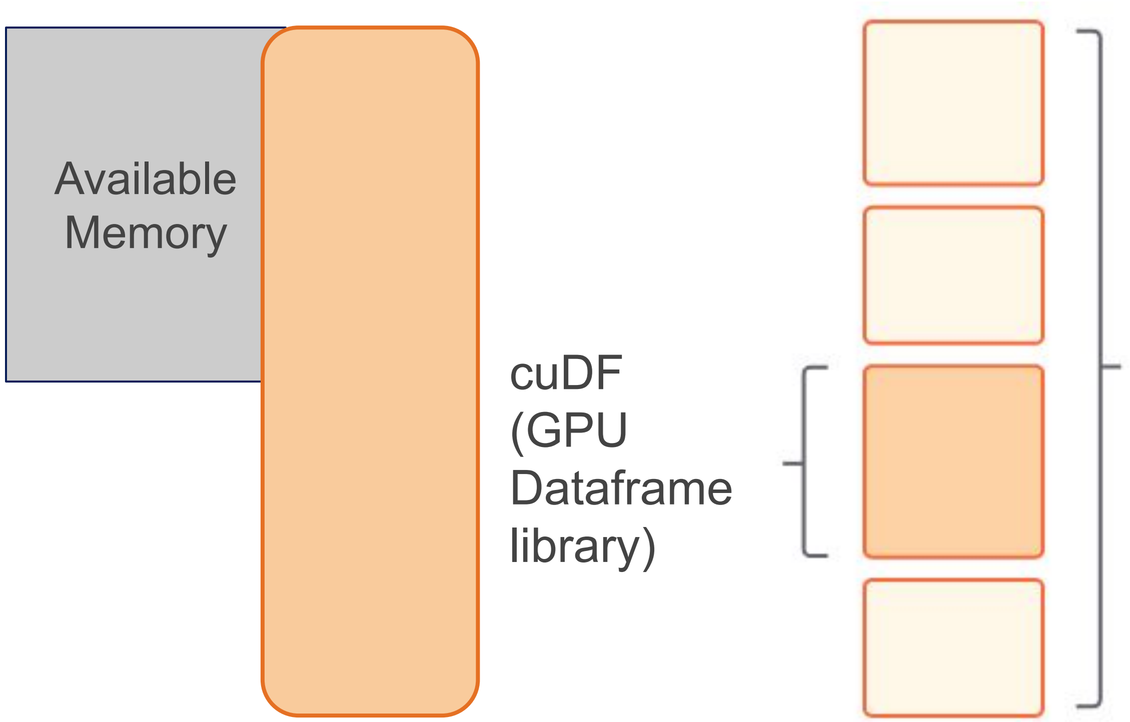


Dask DataFrame
- partitioned *row-wise*
- process large (out of core) datasets
- process partitions in parallel on CPU
- supports a subset of the Pandas API

https://pandas.pydata.org/docs/index.html

https://docs.dask.org/en/stable/dataframe.html

Processing large Datasets on GPU



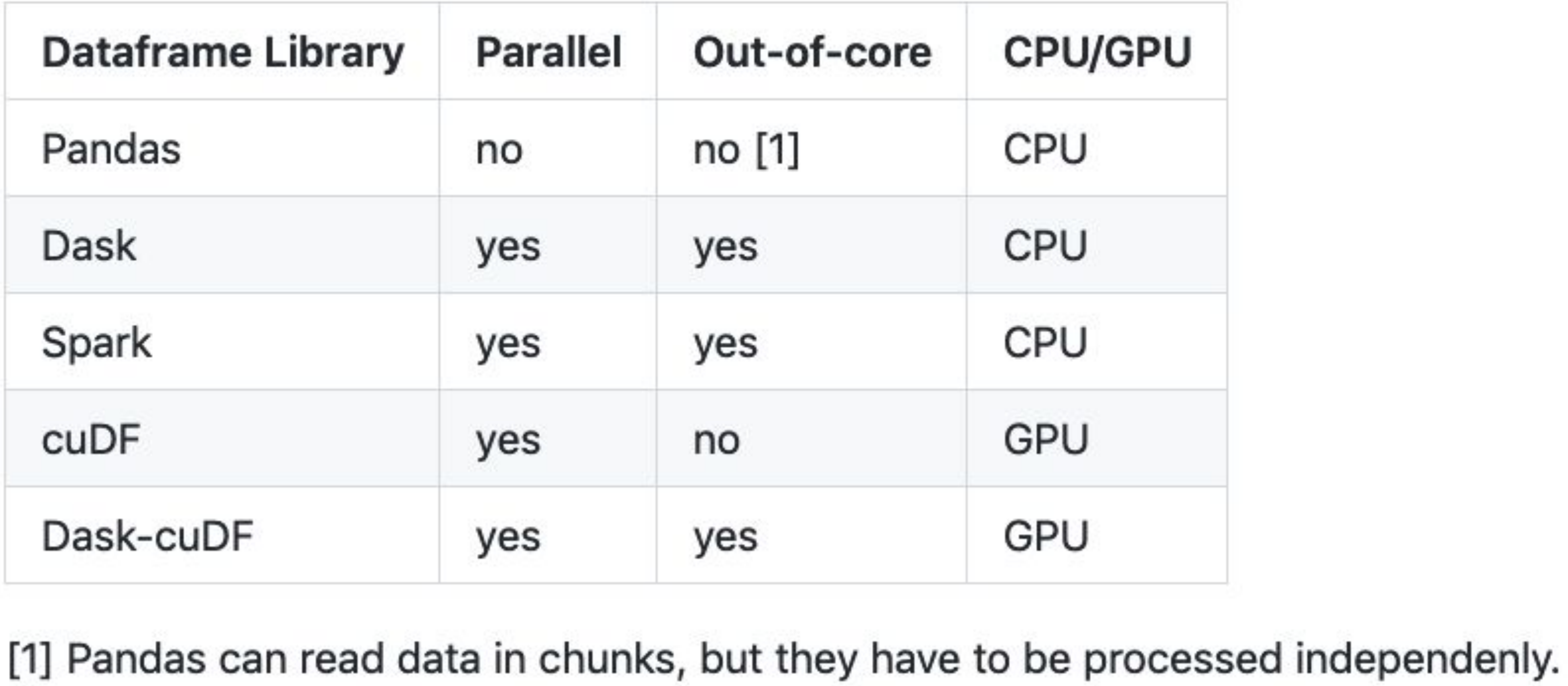
Dask-cuDF DataFrame

- partitioned *row-wise*
- process large (out of core) datasets
- process partitions on GPU
- supports a subset of the Pandas API

https://docs.rapids.ai/api

Example Notebooks

https://github.com/sbl-sdsc/df-parallel



Dataframe Library	Parallel	Out-of-core	CPU/GPU
Pandas	no	no [1]	CPU
Dask	yes	yes	CPU
Spark	yes	yes	CPU
cuDF	yes	no	GPU
Dask-cuDF	yes	yes	GPU

[1] Pandas can read data in chunks, but they have to be processed independenly.

Task 1B

- Run the notebooks
- Compare the execution time for 5 dataframe libraries

- Follow the instructions in section 3.3:
    - https://github.com/ciml-org/ciml-summer-institute-2022

Dataframe Comparison

Results for running on SDSC Expanse GPU node with 10 CPU cores (Intel Xeon Gold 6248 2.5 GHz), 1 GPU (NVIDIA V100 SMX2), and 92 GB of memory (DDR4 DRAM), local storage (1.6 TB Samsung PM1745b NVMe PCIe SSD).

Datafile size: 21.4 GB (`ncopies=4`) In-memory size (Pandas): 62.4 GB

Dataframe Library	time (s)	Parallel	Out-of-core	CPU/GPU
Pandas	222.4	no	no	CPU
Dask	42.1	yes	yes	CPU
Spark	31.2	yes	yes	CPU
cuDF	-- [2]	yes	no	GPU
Dask-cuDF	11.9	yes	yes	GPU

[2] out of memory

Create a Packed Conda Environment

Using a Packed Conda Environment

Galyleo creates a Conda environment on the fly

- Conda is slow!
- Use `--mamba` option (50 - 80% faster)
- https://mamba.readthedocs.io/en/latest/

(a fast moving highly venomous snake)

If you use an environment frequently, create a packed Conda environment

- A packed Conda environment can be moved:
  - compute node -> home directory
- Limitations: mixing Conda and PyPi dependencies may not work

How to Create and Use a Packed Conda Environment

Clone the Git repo

```
git clone https://github.com/sdsc-hpc-training-org/notebooks-sharing.git
```

Create packed Conda Environment:

```
./notebooks-sharing/pack.sh --account <account_number> --conda-env notebooks-sharing
--conda-yml "${HOME}/notebooks-sharing/environment.yml"
```

Use packed Conda Environment

```
galyleo launch --account <account_number> --partition shared --cpus 8 --memory 16
--time-limit 00:30:00 --conda-env notebooks-sharing
--conda-pack "${HOME}/notebooks-sharing.tar.gz"
```

More details about Conda environment: https://github.com/mkandes/galyleo#conda-environments

Task 2A

- Clone notebooks-sharing Git repository
- Create a packed Conda environment

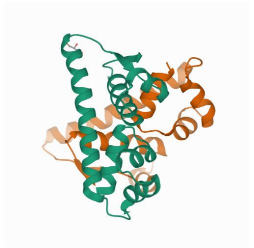
- Follow the instructions in section 3.3:
  https://github.com/ciml-org/ciml-summer-institute-2022

Use Case: Predict Protein Fold Class

Protein Sequence

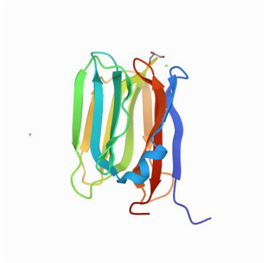
Fold Class

TNKELQAIRKLLMLDVSEAAEHIGRVSARSWQYWESGRSAVPDDVEQEML
DLASVRIEMMSAIDKRLADGERPKLRFYNKLDEYLADNPDHNVIGWRLSQS
VAALYYTEGHADLI



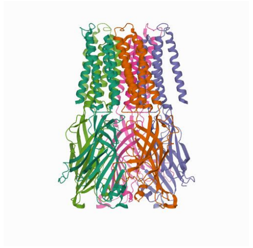
alpha

GARSSSYSGEYGSGGGKRFSHSGNQLDGPITALRVRVNTYYIVGLQVRYG
KVWSDYVGGRNGDLEEIFLHPGESVIQVSGKYKWYLKKLVFVTDKGRYLSF
GKDSGTSFNAVPLHPNTVLRFISGRSGSLIDAIGLHWDVYPSSCSRC



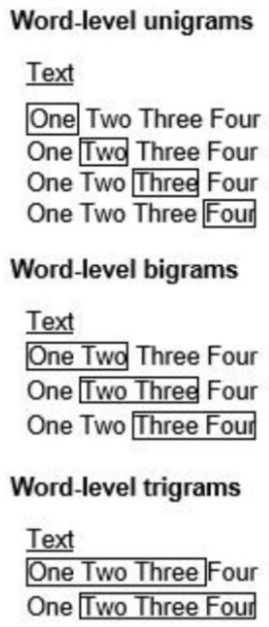
beta

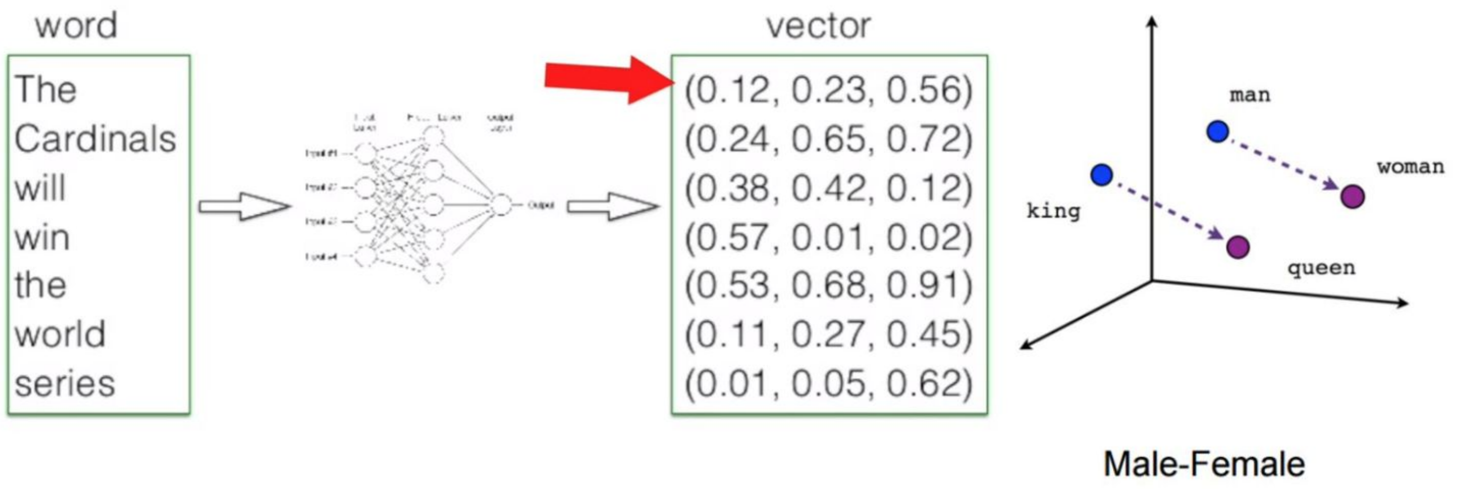
APADNAADARPVDVSVSIFINKIYGVNTLEQTYKVDGYIVAQWTGKPRKTPGD
KPLIVENTQIERWINNGLWVPALEFINVVGSPDTGNKRLMLFPDGRVIYNARFL
GSFSNDMDFRLFPFDRQQFVLELEPFSYNNQQLRFSDIQVYTENIDNEEIDEW
WIRGKASTHISDIRYDHLSSVQPNQNEFSRITVRIDAVRNPSYYLWSFILPLGLII
AASWSVFWLESFSERLQTSFTLMLTVVAYAFYTSNILPRLPYTTVIDQMIIAGYG
SIFAAILLIIFAHHRQANGVEDDLLIQRCRLAFPLGFLAIGCVLVIRGITL



alpha+beta

N-grams and Word2Vec Models





https://spark.apache.org/docs/latest/mllib-feature-extraction.html#word2vec

Embedding a Protein Sequence

Sequence:
TNKELQAIRKLL…

3-grams ("words"):
TNK, NKE, KEL, ELQ, …

Word2Vec (100-dimensional vector) for each 3-gram:
[-2.23197367481583, -0.4659580592717598, …]

Pre-trained Word2Vec model trained on 546,790 protein sequences: ProtVec

Asgari E, Mofrad MR (2015) Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics, PLoS One. 10(11):e0141287. (https://doi.org/10.1371/journal.pone.0141287)

Transfer Learning

Sequence

TNKELQAIRKLL…

3-grams

TNK, NKE, KEL, ELQ, …

ProtVec Model

Feature Vector (embedding)
100-dimensional

[-2.23197367481583,
-0.4659580592717598, …]

Downstream Classification Models

- SVM
- Logistic Regression
- Neural Network

Task 2B

- Run the notebooks using the packed Conda environment
- **Do not shutdown Jupyter Lab** (required for Task 3)

- Follow the instructions in section 3.3:
  https://github.com/ciml-org/ciml-summer-institute-2022

Run Jupyter Lab in Batch

Run Jupyter Lab in Batch

Papermill

- execute notebooks

```
papermill input.ipynb output.ipynb
```

- parameterize notebooks (pass arguments to Jupyter Notebooks)

```
papermill input.ipynb output.ipynb -p variable1 value1 -p variable2 value2
```

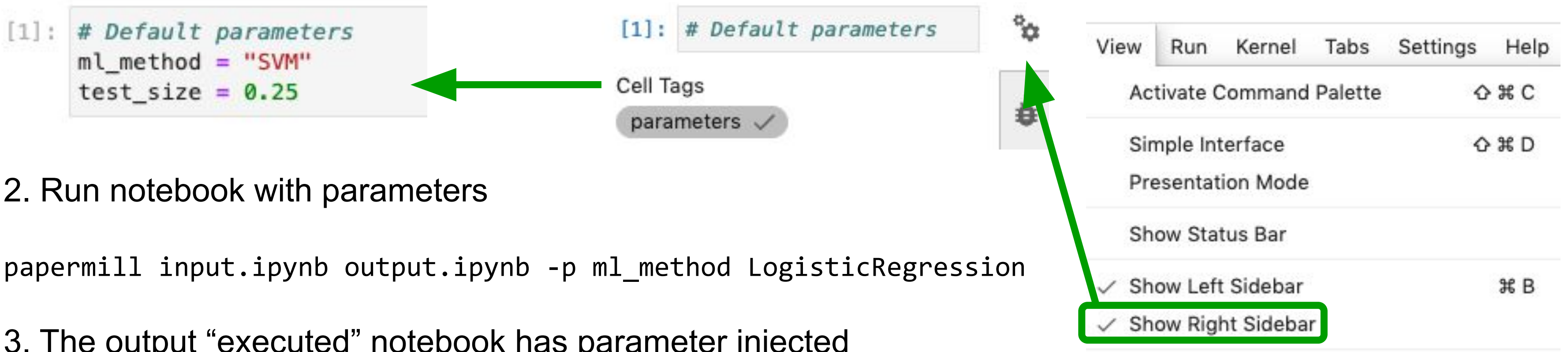
https://papermill.readthedocs.io/en/latest/

Example batch files

https://github.com/sdsc-hpc-training-org/notebooks-sharing/blob/main/batch.sh
https://github.com/pwrose/df-parallel/blob/main/batch_cpu.sh
https://github.com/pwrose/df-parallel/blob/main/batch_gpu.sh

Parameterize a Notebook

1. Add “parameters” tag and save notebook (specify all parameters in a single cell!)



```
[1]: # Default parameters
     ml_method = "SVM"
     test_size = 0.25
```

```
[1]: # Default parameters
```

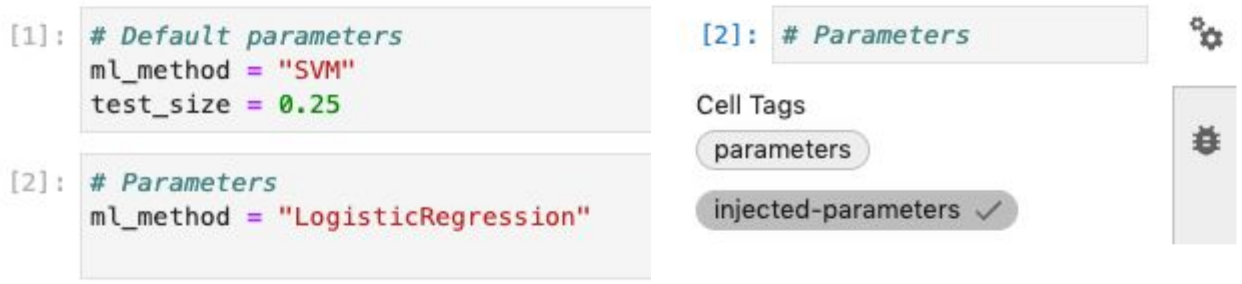
Cell Tags

parameters ✓

2. Run notebook with parameters

```
papermill input.ipynb output.ipynb -p ml_method LogisticRegression
```

3. The output “executed” notebook has parameter injected



```
[1]: # Default parameters
     ml_method = "SVM"
     test_size = 0.25

[2]: # Parameters
     ml_method = "LogisticRegression"
```

```
[2]: # Parameters
```

Cell Tags

parameters

injected-parameters ✓

Task 2C

- Submit the batch job
- In your Jupyter Lab session, navigate to the “results” directory
- Examine the “executed” output notebooks

- Follow the instructions in section 3.3:
  https://github.com/ciml-org/ciml-summer-institute-2022

Get ready to use Expanse: accounts, allocations

Expanse Allocation

- Expanse is an XSEDE computing resource
- Apply for an Expanse trial account
  - https://portal.xsede.org/allocations/startup#rapidaccess-trial
- Submit a submit a proposal through the XSEDE Allocation Request System
  - https://portal.xsede.org/allocations/announcements

How much does it cost to run the jobs?

Run on CPU (Pandas, Dask, Spark dataframes)

```
galyleo launch --account <account_number> --partition shared --cpus 10
--memory 20 --time-limit 00:30:00 --conda-env df-parallel
--conda-yml "${HOME}/df-parallel/environment.yml" --mamba
```

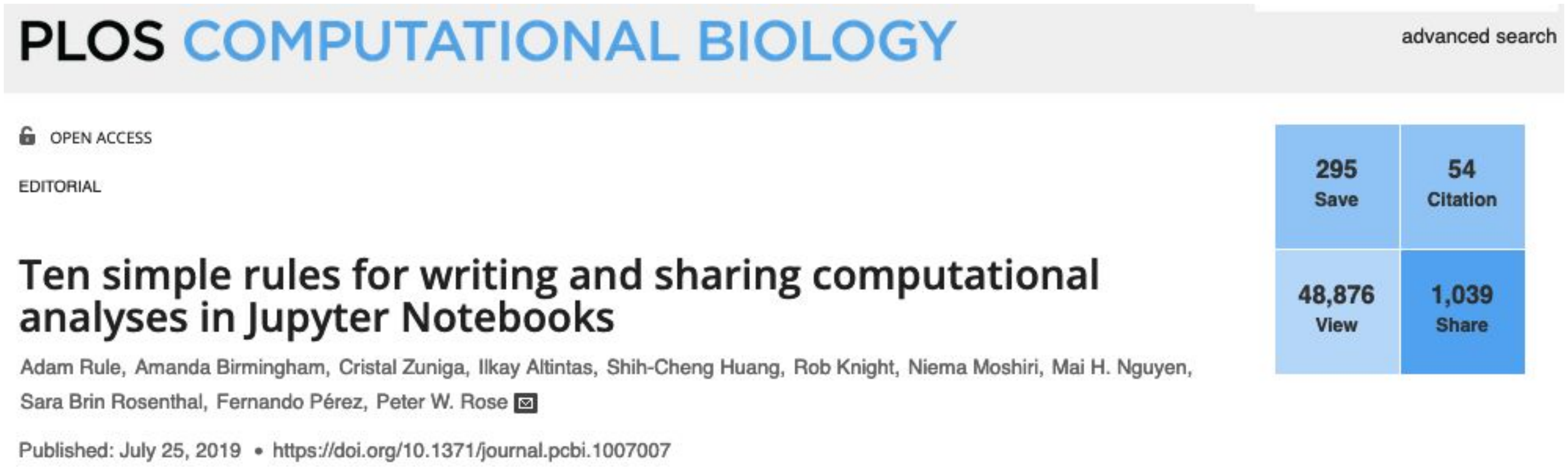
1 CPU or 2GB of memory are charged 1 CPU Service Unit (SU)/hour.
This job will be charged 10 SU/hour or 5 SUs for 30 minutes.

Run on GPU (Pandas, Dask, Spark, cuDF, Dask-cuDF dataframes)

```
galyleo launch --account <account_number> --partition gpu-shared --cpus 10
--memory 92 --gpus 1 --time-limit 00:30:00 --conda-env df-parallel-gpu
--conda-yml "${HOME}/df-parallel/environment_gpu.yml" --mamba
```

1 GPU or 10 CPUs or 92 GB of memory are charged 1 GPU Service Unit (SU)/hour.
This job will be charged 1 GPU SU/hour. The minimum charge for any job is 1 SU. So this job will use 1 SU even though it's just run for 30 minutes.

Best Practices for Authoring Jupyter Notebooks


PLOS COMPUTATIONAL BIOLOGY

advanced search

OPEN ACCESS

EDITORIAL

Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks

Adam Rule, Amanda Birmingham, Cristal Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H. Nguyen, Sara Brin Rosenthal, Fernando Pérez, Peter W. Rose

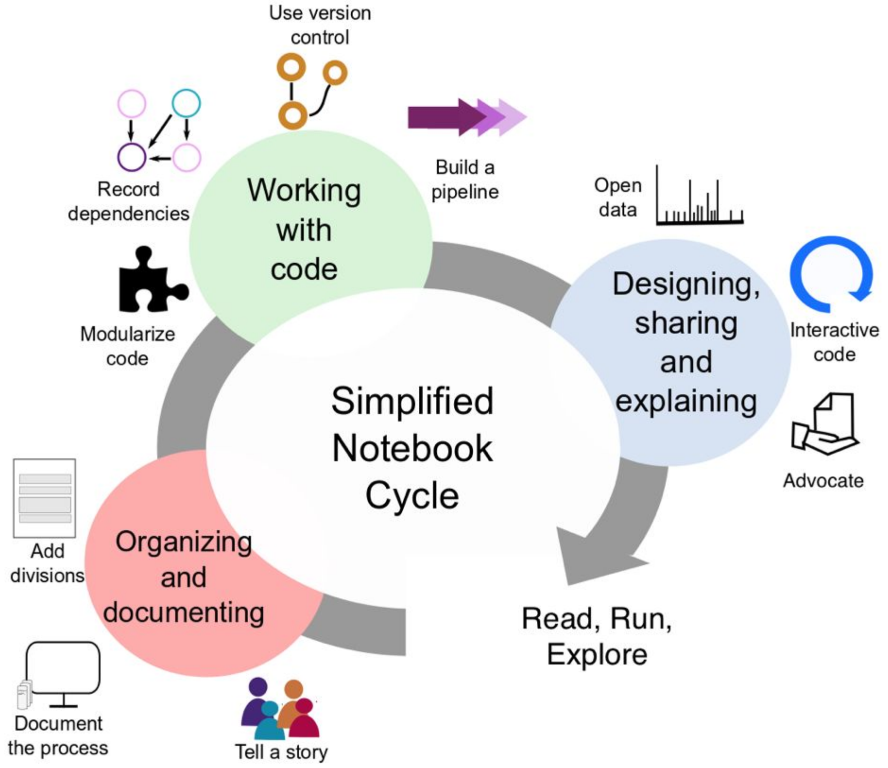
Published: July 25, 2019 • https://doi.org/10.1371/journal.pcbi.1007007

295 Save	54 Citation
48,876 View	1,039 Share

Paper: https://doi.org/10.1371/journal.pcbi.1007007

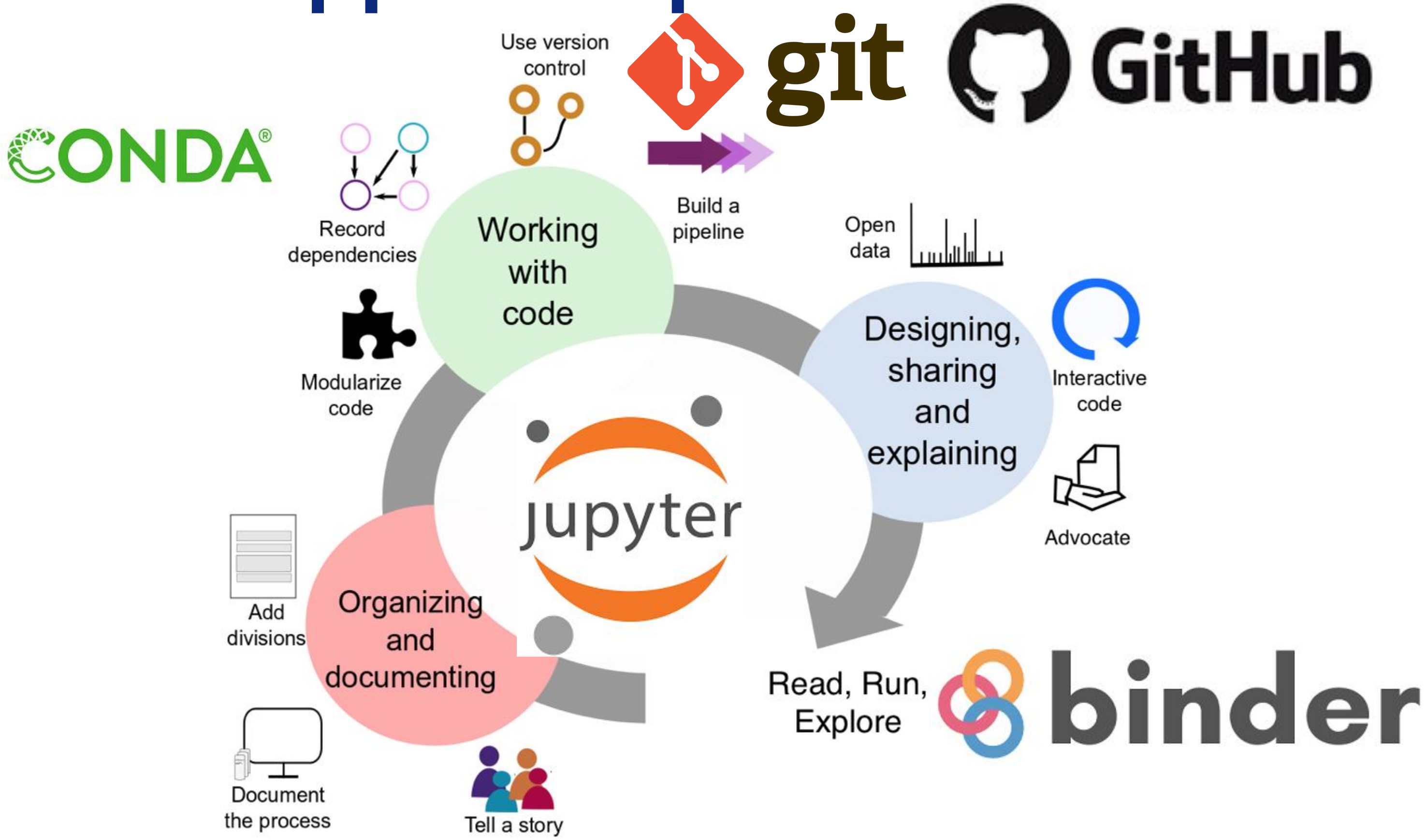
Git repo: https://github.com/jupyter-guide/ten-rules-jupyter

Ten Simple Rules

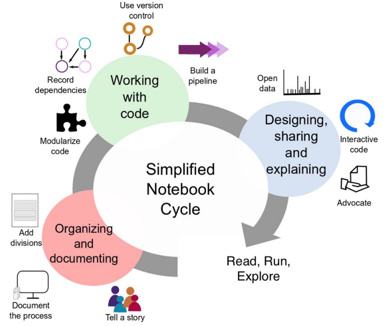


Ten Simple Rules for Writing and Sharing Computational Analyses in Jupyter Notebooks, PLOS Comp. Biol. 2019, https://doi.org/10.1371/journal.pcbi.1007007

Tools to Support Reproducible Workflows

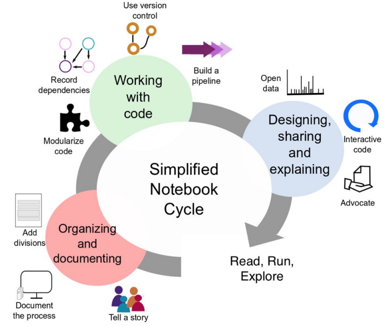


Organizing and Documenting



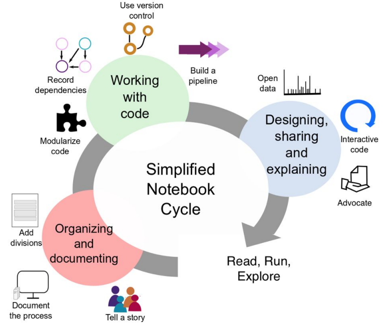
- **Rule 1: Tell a Story for an Audience**
  - Beginning - introduce topic
  - Middle - describe steps
  - End - interprets results
  - Describe not just what you did, by why you did it, how the steps are connected, and what it all means.
  - Adjust your description depending on the intended audience
- **Rule 2: Document the process, not just the results**
  - Add descriptive notes, e.g., why a particular parameter was chosen
- **Rule 3: Use cell divisions to make steps clear**
  - Avoid long cells
  - Limit each cell to one meaningful step
  - Split long notebooks into a series of notebooks
  - Keep a top-level index notebook with links to the individual notebooks

Working with Code



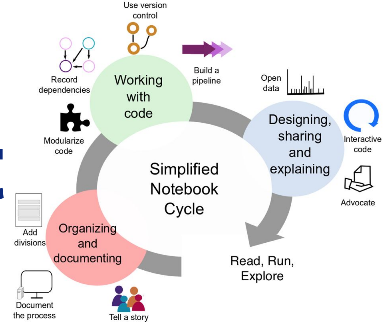
- **Rule 4: Modularize Code**
  - Use functions instead of duplicating code cells
- **Rule 5: Record Dependencies**
  - Manage your dependencies explicitly from the start using a tool such as
    - Conda's environment.yml
    - pip's requirements.txt
- **Rule 6: Use Version Control**
  - Consider using a public repository from the beginning of a project
  - Tie research results to specific software versions
- **Rule 7: Build a Pipeline**
  - Design notebooks with reuse in mind (different input data and parameters)
  - Define key input data and parameters at the top of each notebook
  - Break long notebooks into smaller notebooks that focus on one or a few analysis steps.

Sharing, explaining



- **Rule 8: Share and Explain Your Data**
  - Share your data in a repository with a persistent identifier, e.g., DOI or ARK
    - Bio repositories, e.g., NCBI, Ensemble, PDB
    - General repositories, e.g., Zenodo https://zenodo.org/
  - Small datasets can be stored in GitHub with your source code (< 50MB)
    - E.g., in a /data folder
  - Very large datasets
    - Consider using a sample of the data and a link to the original data
  - Save intermediate data after data processing
    - E.g., in /intermediate_data folder
    - Can be used to verify each step in a workflow

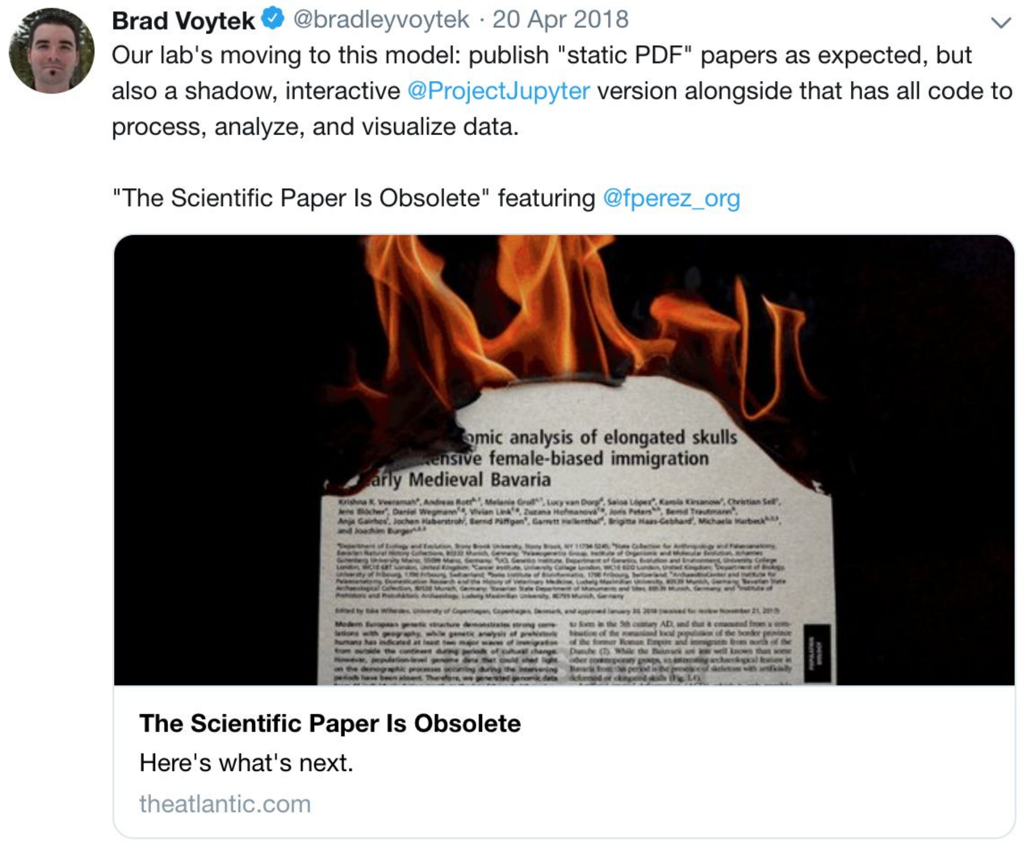
Sharing, explaining cont



- **Rule 9: Design your notebooks to be read, run, and explored**
  - Git repository
    - Add a descriptive README file
    - Add a LICENCE file (liberal licence, e.g., MIT, Apache 2)
    - Add a static HTML/PDF file of your notebooks for long-term preservation
    - Add Binder badge/link to launch notebooks in the cloud (https://mybinder.org/)
  - Consider using ipywidgets to add menus or sliders to enable interactive exploration of parameters

Sharing, explaining cont.

- **Rule 10: Advocate for open research**
  - Apply what you learned in this tutorial in your own research and be an advocate for open and reproducible research in your lab or workplace
  - Publish a fully reproducible paper! Create all figures, data tables, and all other computational results using Jupyter Notebook and deposit in Github.



Brad Voytek @bradleyvoytek · 20 Apr 2018

Our lab's moving to this model: publish "static PDF" papers as expected, but also a shadow, interactive @ProjectJupyter version alongside that has all code to process, analyze, and visualize data.

"The Scientific Paper Is Obsolete" featuring @fperez_org

omic analysis of elongated skulls
ensive female-biased immigration
arly Medieval Bavaria

The Scientific Paper Is Obsolete

Here's what's next.

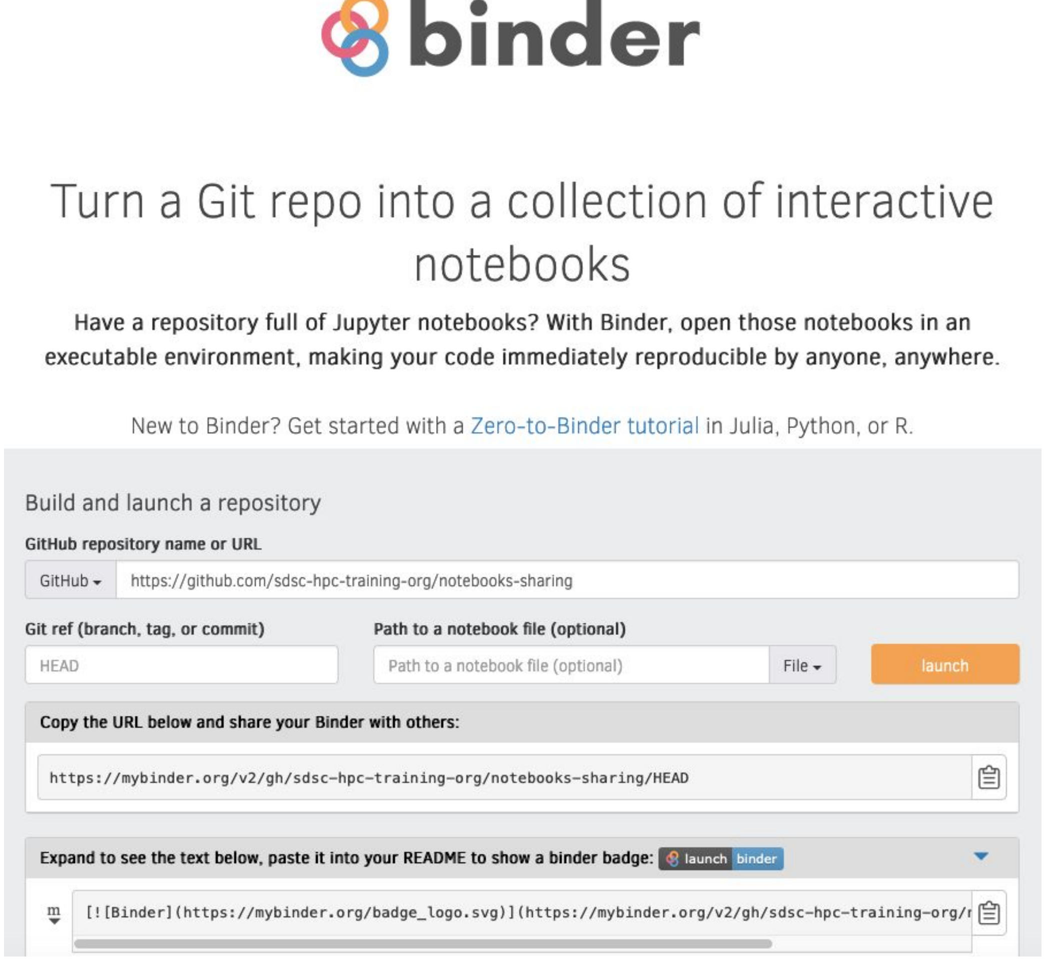
theatlantic.com

The binder Project

https://mybinder.org/

A community that builds **free and open-source** tools for reproducible, sharable scientific environments that are workflow- and platform-agnostic.

Source: Chris Holdgraf, bit.ly/2019-elife-cc-holdgraf


Turn a Git repo into a collection of interactive notebooks

Have a repository full of Jupyter notebooks? With Binder, open those notebooks in an executable environment, making your code immediately reproducible by anyone, anywhere.

New to Binder? Get started with a Zero-to-Binder tutorial in Julia, Python, or R.

Build and launch a repository

GitHub repository name or URL

GitHub https://github.com/sdsc-hpc-training-org/notebooks-sharing

Git ref (branch, tag, or commit): HEAD

Path to a notebook file (optional): Path to a notebook file (optional) File

launch

Copy the URL below and share your Binder with others:

```
https://mybinder.org/v2/gh/sdsc-hpc-training-org/notebooks-sharing/HEAD
```

Expand to see the text below, paste it into your README to show a binder badge: launch binder

```
[![Binder](https://mybinder.org/badge_logo.svg)](https://mybinder.org/v2/gh/sdsc-hpc-training-org/r
```

Demo of binder

https://github.com/sdsc-hpc-training-org/notebooks-sharing

Summary

- When to run on Expanse
- Setup a reproducible and portable software environment
- Use the Expanse Portal
- Run Jupyter Lab on Expanse
- Scale up calculations on CPU/GPU
- Create a packed Conda environment
- Run Jupyter Lab in batch
- Get ready to use Expanse: accounts, allocations
- Best Practices for Authoring Jupyter Notebooks

Acknowledgements

Marty Kandes

Mary Thomas

Scott Sakai

Robert Sinkovitz

Funding: NSF Award Number 2017767

CyberTraining: Implementation: Small: Developing a Best Practices Training Program in Cyberinfrastructure-Enabled Machine Learning Research