

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

CIML Summer Institute 2023

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

COMPUTING WITHOUT BOUNDARIES

EXPANS

SAN DIEGO SUPERCOMPUTER CENTER

UNIVERSITY OF CALIFORNIA SAN DIEGO

Outline

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

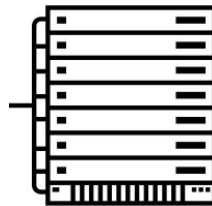
When to run on Expanse

Laptop/Desktop

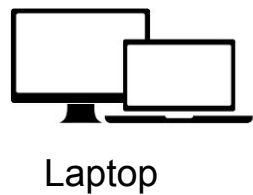


- Coding
- Exploratory phase
- Small datasets
- Run on single or few cores

Expanse

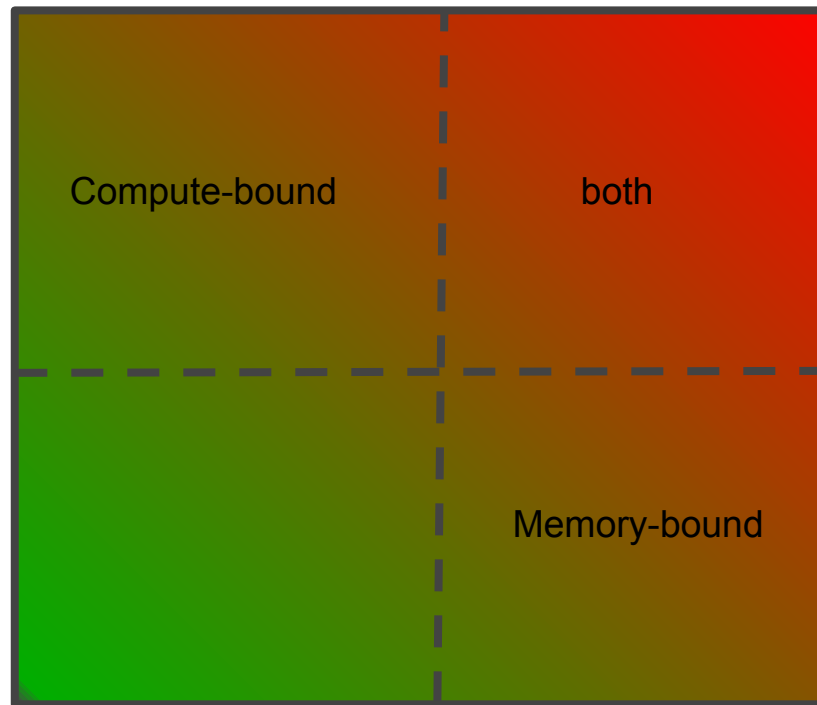


- Scaling up to
 - large datasets
 - long runtimes
- Run on many cores
- Run on GPUs



Laptop

Compute

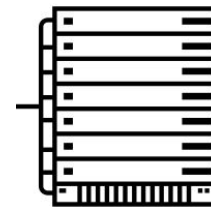


Compute-bound

both

Memory-bound

fit into memory



HPC

Expanse Nodes

128 CPU cores/node

4 GPUs, 40 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

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; <i>limit applies per group</i>
ind-compute	48 hrs	32	32	64	1	Exclusive access to Industry compute nodes; <i>limit applies per group</i>
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; <i>max two gpus per job</i>
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 portable and reproducible software environment

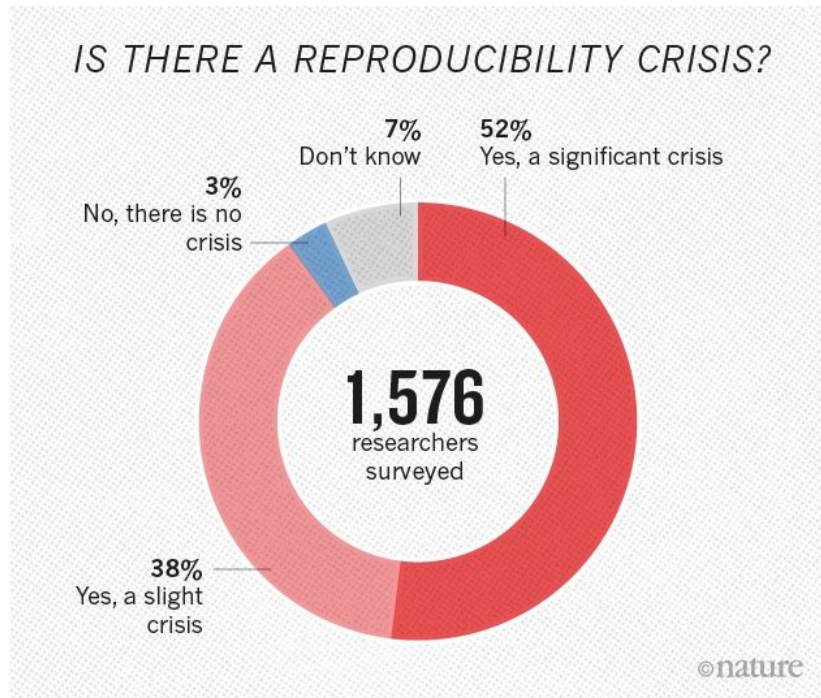
Reproducibility Crisis?

“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*

obtaining **consistent** results using

same input data or parameters

same computational steps, methods, and code

same analysis conditions

Reusability

obtaining **new** results using

different input data or parameters

same computational steps, methods, and code

same analysis conditions

Scalability

obtaining **new** results using

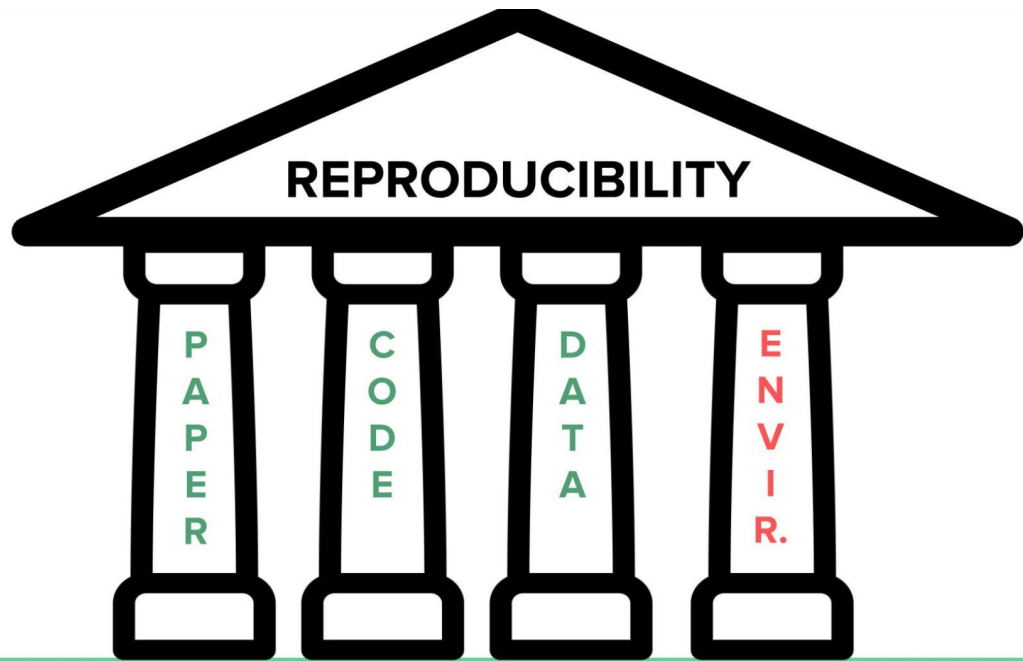
large input data or parameter sets

same computational steps, methods, and code

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



Open Science

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

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

Tools and Infrastructure



Computational notebooks:
combine documentation,
code, and results



git

Version-control system
for tracking changes in
source code



GitHub

Source code
repository



Open-source package
and environment
management system



Container that packages
software and OS in a
portable way



Scalable compute infrastructure

Reproducible Environments



- 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



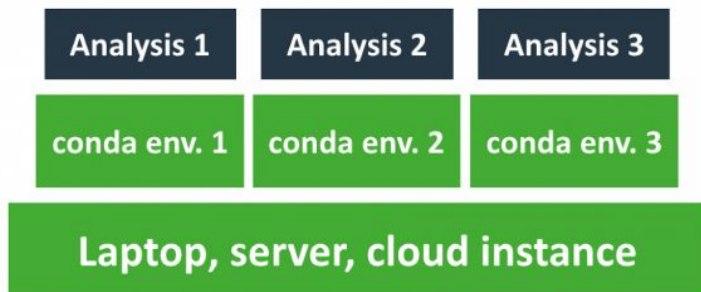
- Advanced user
- Experience with containers
- Production environment
- Often setup for a single tool
- Optimized containers for Expanse
 - pytorch, tensorflow, ...
 - support for multi-node
- Supported on Linux
- Mac, Windows requires a VM
- Run on packaged OS, e.g. Ubuntu



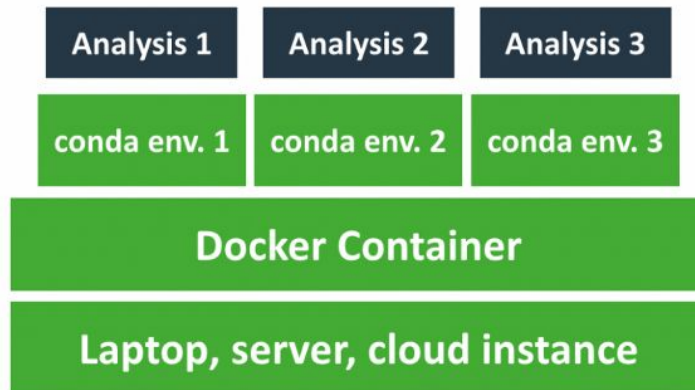
Data Scientist



DevOps

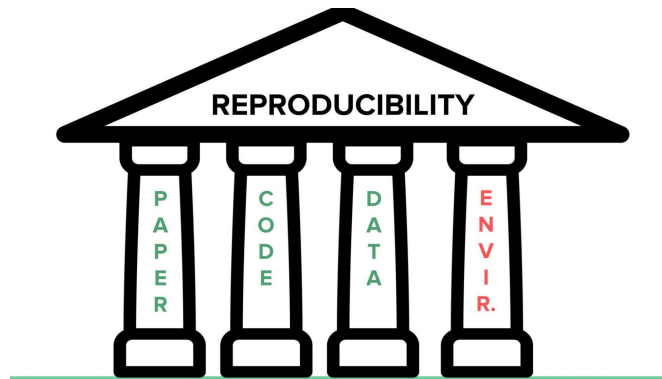


Data Science Development



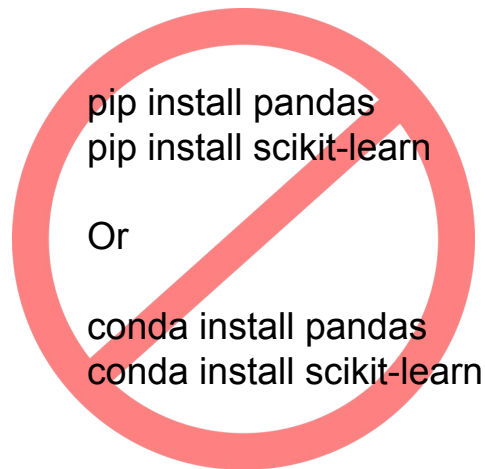
Data Science Deployment

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



- **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 allows 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?



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/sbl-sdsc/df-parallel>)

```
name: df-parallel
```

```
channels:
```

- conda-forge
- anaconda

```
dependencies:
```

- python=3.10
- jupyterlab=4.0.1
- ipywidgets=8.0.6
- matplotlib=3.7.1
- seaborn=0.12.2
- papermill=2.3.4
- dask=2023.3.2
- pyspark=3.4.0
- pyarrow=10.0.1
- openjdk=17.0.3

```
variables:
```

```
# SPARK conf directory with logging configuration
SPARK_CONF_DIR: ../conf
SPARK_DRIVER_MEMORY: 16G
SPARK_DRIVER_MAXRESULTSIZE: 4G
SPARK_WORKER_MEMORY: 4G
```

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)
```



Mac, Windows, Linux

Activate a Conda environment

```
conda activate <environment_name>
```

Run Jupyter Lab

```
jupyter lab
```

Deactivate conda environment

```
conda deactivate
```



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

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

Expanse Portal

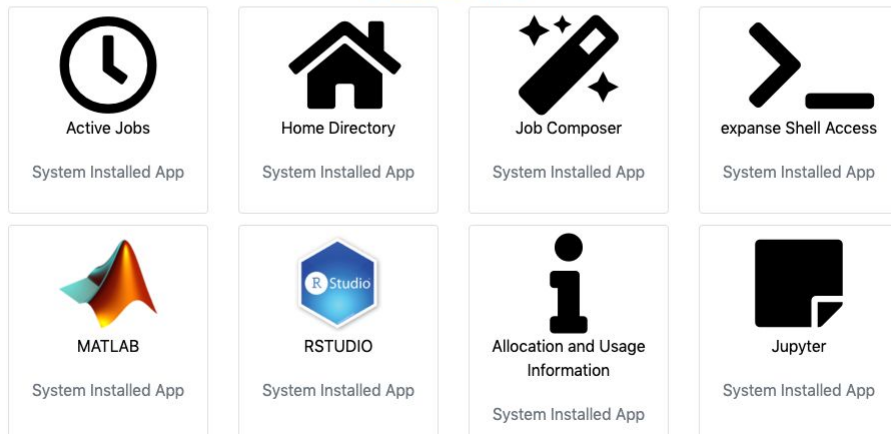


SDSC

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](#)



Login with your ACCESS credentials:

<https://portal.expanse.sdsc.edu/>

Note, for this workshop use your trainxx credentials and sign in on the training page:

<https://portal.expanse.sdsc.edu/training>

Expanse Allocations CPU/GPU

Open OnDemand / Allocation and Usage Information

Resource:

Project ID:

Allocation and Usage Information

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

Open OnDemand / Allocation and Usage Information

Resource:

Project ID:

Allocation and Usage Information

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



Your Jobs ▾

All Clusters ▾

Active Jobs

Show 50 ▾ entries

Filter:

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

Run Jupyter Lab on Expanse

Galileo 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/galileo>

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 01:30:00 --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.expense-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.

The screenshot shows a JupyterLab interface. The top bar includes 'File', 'Edit', 'View', 'Run', 'Kernel', 'Tabs', 'Settings', and 'Help'. Below the top bar is a file browser showing a directory structure: 'df-parallel / notebooks /'. A table lists files with their names and last modified times:

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

The main area displays a notebook titled '1-DownloadData.ipynb'. The notebook content includes a title 'Download Gene Inform' and a description: 'This notebook downloads a list of genes for NCBI. To ensure platform independence, this notet Python libraries to download and unzip a cor'. The first code cell contains the line: `[1]: import os`.

The screenshot shows the 'File' menu in JupyterLab. The menu items are: New, New Launcher, Open from Path..., Open from URL..., New View for Notebook, New Console for Notebook, Close Tab, Close and Shutdown Notebook, Close All Tabs, Save Notebook, Save Notebook As..., Save All, Reload Notebook from Disk, Revert Notebook to Checkpoint, Rename Notebook..., Download, Save and Export Notebook As..., Save Current Workspace As..., Save Current Workspace, Print..., Log Out, and Shut Down. The 'Shut Down' option is highlighted with a green box.

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 01:30:00 --conda-env df-parallel  
--conda-yml "${HOME}/df-parallel/environment.yml" --mamba
```

Run on GPU (cuDF, Dask-cuDF dataframes)

```
galyleo launch --account <account_number> --partition gpu-shared --cpus 10  
--memory 92 --gpus 1 --time-limit 01:30:00 --conda-env df-parallel-gpu  
--conda-yml "${HOME}/df-parallel/environment-gpu.yml" --mamba
```

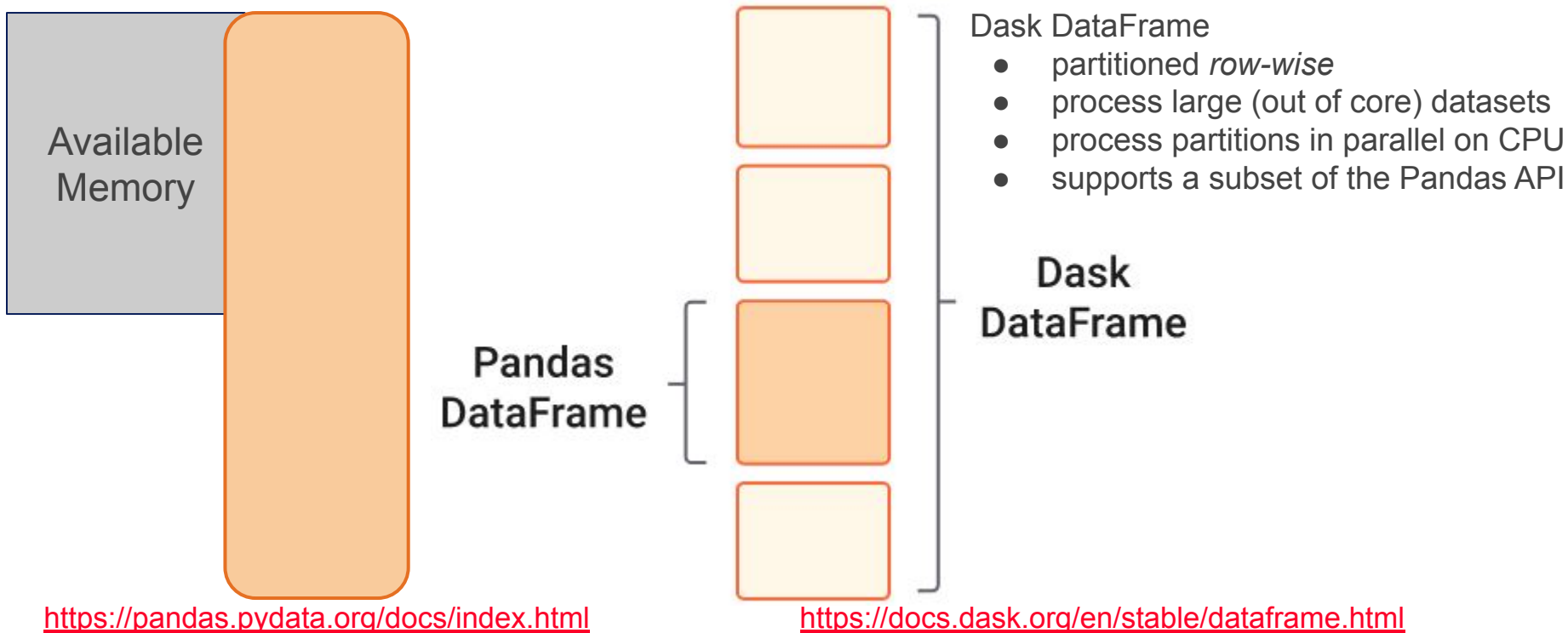
1 GPU + 10 CPUs + 92 GB = ¼ of a GPU node

Task 1 - Launch Jupyter Lab on Expanse using a CONDA environment

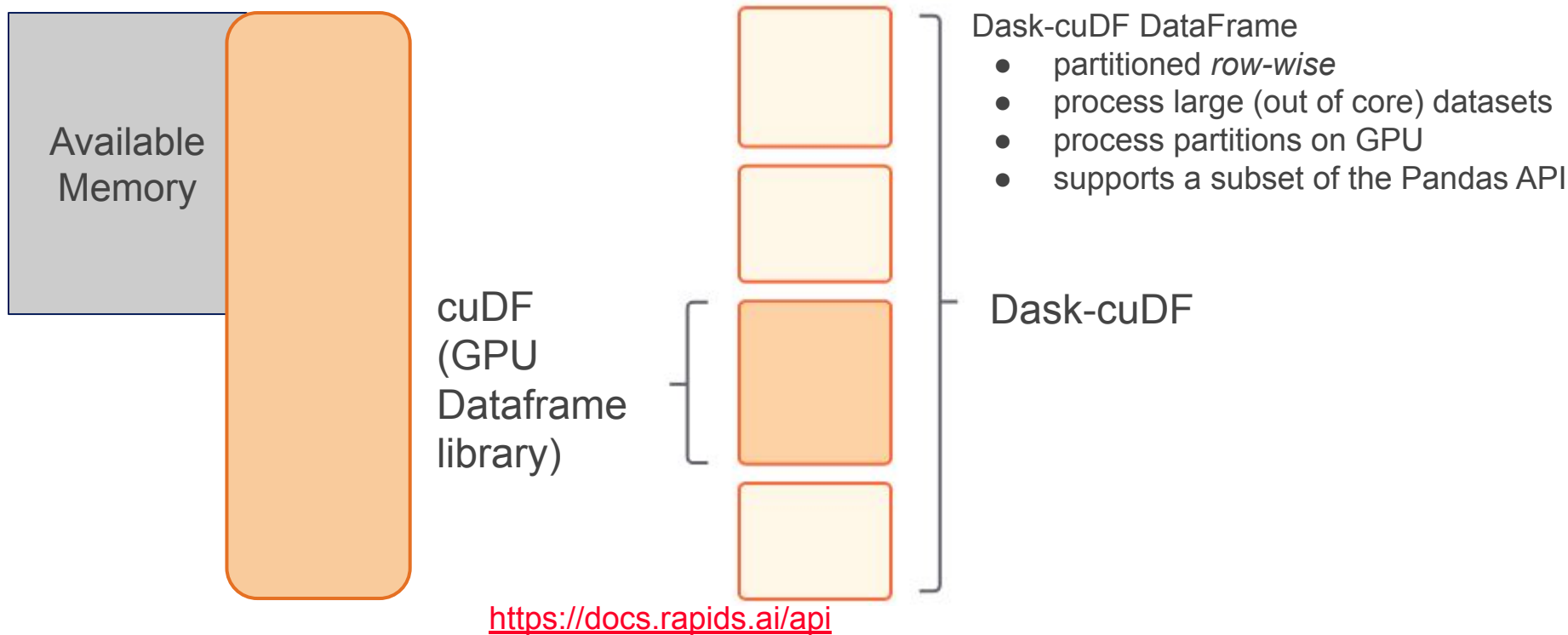
- Clone df-parallel Git repository
- Start galaxy on a GPU node
- Paste the URL for your Jupyter Lab session into your browser
- Follow the instructions for [Task 1](#)

Scale up Calculations on CPU/GPU

Processing large Datasets on CPU



Processing large Datasets on GPU



Example Notebooks

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

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

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

Task 2 - Run Jupyter Lab Interactively

- Compare the runtimes for 5 dataframe libraries
- Follow the instructions for [Task 2](#)

Dataframe Benchmark

Results for running on SDSC [Expansive 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 (gene_info.tsv):

- Dataset 1: 5.4 GB (18 GB in Pandas)
- Dataset 2: 21.4 GB (4 x Dataset 1) (62.4 GB in Pandas)
- Dataset 3: 43.7 GB (8 x Dataset 1)

Dataframe Library	time(5.4 GB) (s)	time(21.4 GB) (s)	time(43.7 GB) (s)	Parallel	Out-of-core	CPU/GPU
Pandas	56.3	222.4	-- [2]	no	no	CPU
Dask	15.7	42.1	121.8	yes	yes	CPU
Spark	14.2	31.2	56.5	yes	yes	CPU
cuDF	3.2	-- [2]	-- [2]	yes	no	GPU
Dask-cuDF	7.3	11.9	19.0	yes	yes	GPU

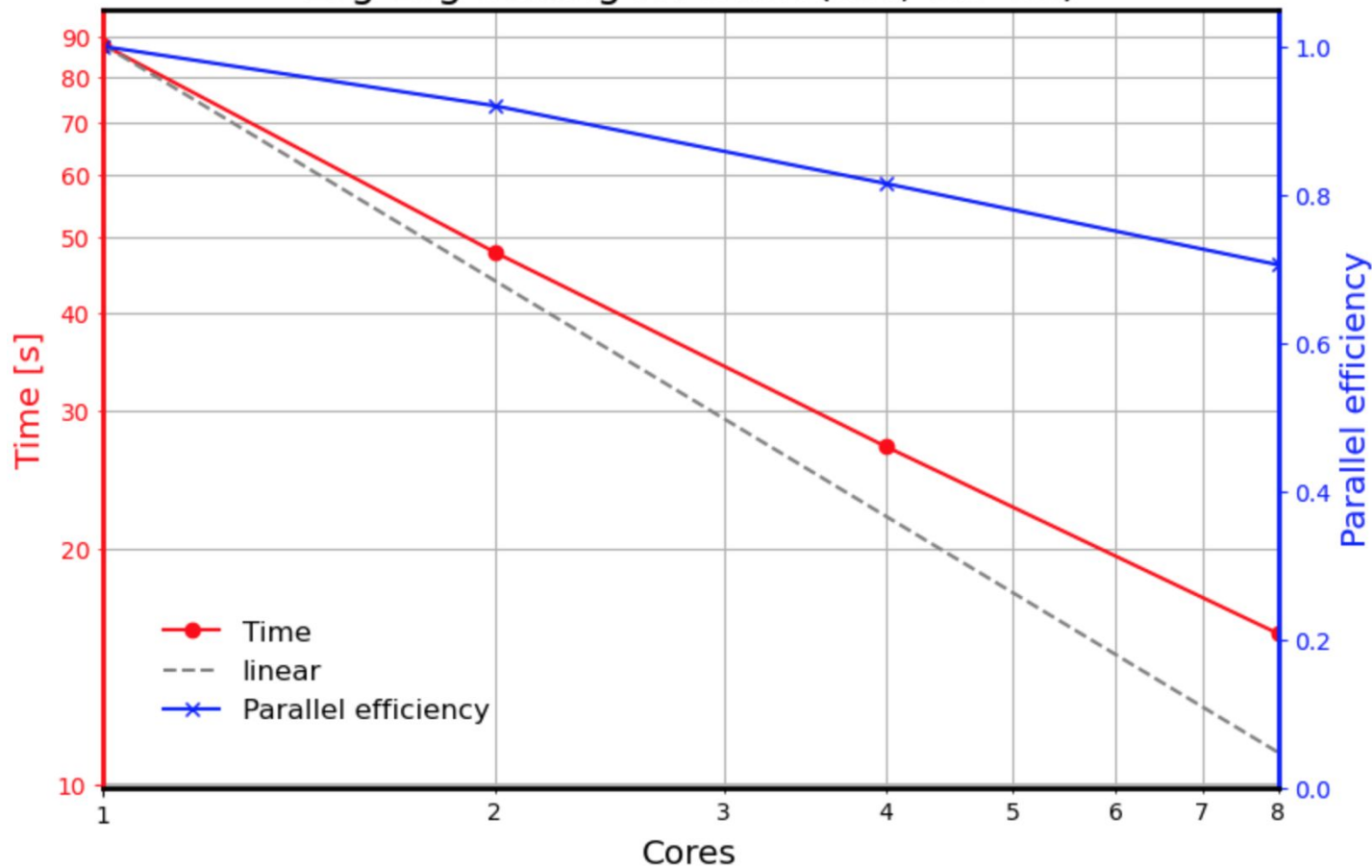
[2] out of memory

Parallel Efficiency

Task 3 - Assess Parallel Efficiency

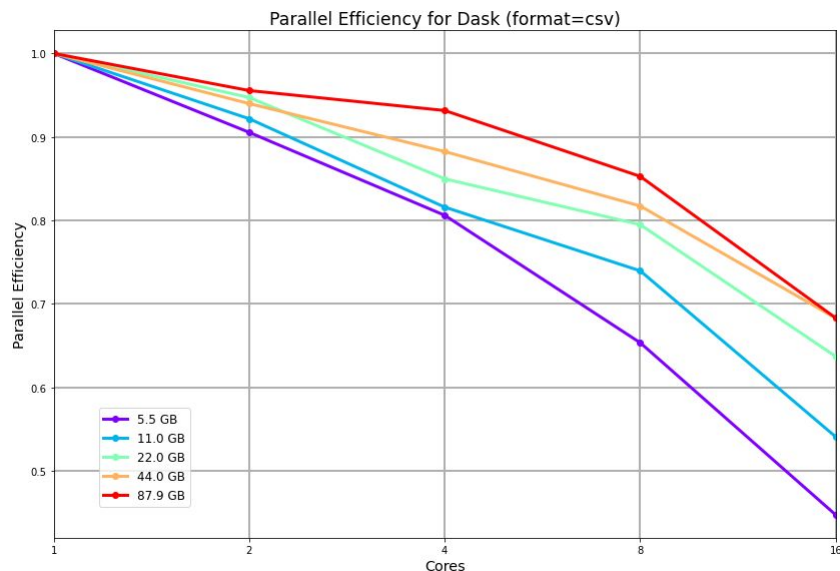
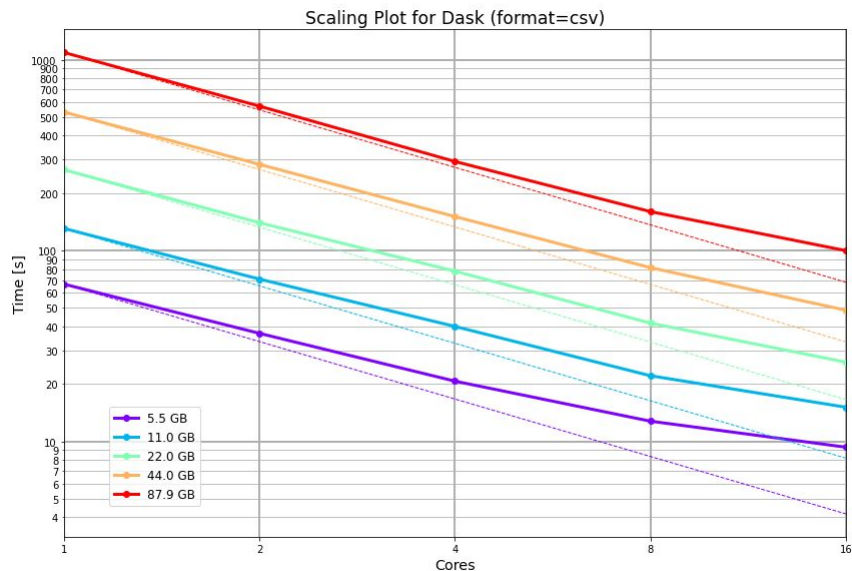
- Evaluate the parallel efficiency for scaling up from 1 to 8 CPUs.
- Follow the instructions for Task 3

Log-Log Scaling for Dask (csv, 6.5 GB)



Scaling for Dask Dataframe

Runtime as function of dataset size and number of cores



Batch benchmark: slurm script: [benchmark.sb](#) notebook: [8-BenchmarkSummary.ipynb](#)

Efficient File Formats

Columnar Storage Format - Parquet

Query and space efficient file format (default: Snappy compression)

Vertical partitioning (projection push down) + Horizontal partitioning (predicate push down) = Read only the data you need!

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

+

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

=

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5



@EmrgencyKittens

Horizontal partitioning uses column min/max statistics from Parquet metadata.

<https://www.slideshare.net/julienledem/if-you-have-your-own-columnar-format-stop-now-and-use-parquet>

Vertical & Horizontal Partitioning

```
# vertical partitioning
column_names = ["GeneID", "Symbol", "Synonyms", "description", "type_of_gene", "#tax_id", "chromosome"]
# horizontal partitioning
filters=[["type_of_gene", "==", "protein-coding"]]

# Pandas
genes = pd.read_parquet(filename, columns=column_names, filters=filters)
# Dask
genes = dd.read_parquet(filename, columns=column_names, filters=filters)
# cuDF
genes = cudf.read_parquet(filename, columns=column_names, filters=filters)
# Dask-cuDF
genes = dask_cudf.read_parquet(filename, columns=column_names, filters=filters)

# Spark
genes = spark.read.parquet(filename)
genes = genes.select(column_names)
genes = genes.filter("type_of_gene == 'protein-coding'")
```


Creating Parquet “Files” with Dask

```
genes = dd.read_csv(input, dtype=str, sep="\t")
```

```
genes.to_parquet(output, write_index=False,  
write_metadata_file=True, engine="pyarrow")
```

```
genes = dd.read_csv(input, dtype=str, sep="\t")
```

```
genes.to_parquet(output, write_index=False,  
write_metadata_file=True, engine="pyarrow",  
partition_on=["type_of_gene"])
```

```
[xdtr104@login02 ~]$ ls -lh gene_info.parquet/  
total 1.2G  
-rw-r--r-- 1 xdtr104 uic157 6.9K Aug  4 10:14 _common_metadata  
-rw-r--r-- 1 xdtr104 uic157 161K Aug  4 10:14 _metadata  
-rw-r--r-- 1 xdtr104 uic157 9.9M Aug  4 10:14 part.0.parquet  
-rw-r--r-- 1 xdtr104 uic157 9.8M Aug  4 10:14 part.1.parquet  
-rw-r--r-- 1 xdtr104 uic157 16M Aug  4 10:14 part.10.parquet  
-rw-r--r-- 1 xdtr104 uic157 18M Aug  4 10:14 part.11.parquet  
-rw-r--r-- 1 xdtr104 uic157 20M Aug  4 10:14 part.12.parquet  
-rw-r--r-- 1 xdtr104 uic157 17M Aug  4 10:14 part.13.parquet  
-rw-r--r-- 1 xdtr104 uic157 19M Aug  4 10:14 part.14.parquet  
-rw-r--r-- 1 xdtr104 uic157 18M Aug  4 10:14 part.15.parquet  
-rw-r--r-- 1 xdtr104 uic157 20M Aug  4 10:14 part.16.parquet  
-rw-r--r-- 1 xdtr104 uic157 21M Aug  4 10:14 part.17.parquet  
-rw-r--r-- 1 xdtr104 uic157 19M Aug  4 10:14 part.18.parquet  
-rw-r--r-- 1 xdtr104 uic157 20M Aug  4 10:14 part.19.parquet
```

- Parquet files are typically directories of files.

```
[xdtr104@login02 ~]$ ls -lh gene_info.parquet/  
total 698K  
-rw-r--r-- 1 xdtr104 uic157 6.9K Aug  4 16:16 _common_metadata  
-rw-r--r-- 1 xdtr104 uic157 1.5M Aug  4 16:16 _metadata  
drwxr-xr-x 2 xdtr104 uic157  4 Aug  4 16:16 'type_of_gene=biological-region'  
drwxr-xr-x 2 xdtr104 uic157  68 Aug  4 16:15 'type_of_gene=miscRNA'  
drwxr-xr-x 2 xdtr104 uic157  87 Aug  4 16:15 'type_of_gene=ncRNA'  
drwxr-xr-x 2 xdtr104 uic157  87 Aug  4 16:15 'type_of_gene=other'  
drwxr-xr-x 2 xdtr104 uic157  87 Aug  4 16:15 'type_of_gene=protein-coding'  
drwxr-xr-x 2 xdtr104 uic157  87 Aug  4 16:15 'type_of_gene=pseudo'  
drwxr-xr-x 2 xdtr104 uic157  87 Aug  4 16:15 'type_of_gene=rRNA'  
drwxr-xr-x 2 xdtr104 uic157  5 Aug  4 16:16 'type_of_gene=scRNA'  
drwxr-xr-x 2 xdtr104 uic157  85 Aug  4 16:15 'type_of_gene=snRNA'  
drwxr-xr-x 2 xdtr104 uic157  85 Aug  4 16:16 'type_of_gene=snoRNA'  
drwxr-xr-x 2 xdtr104 uic157  87 Aug  4 16:15 'type_of_gene=tRNA'  
drwxr-xr-x 2 xdtr104 uic157  62 Aug  4 16:16 'type_of_gene=unknown'
```

- They can be partitioned for query efficiency

Run Jupyter Lab in Batch

Run Jupyter Lab in Batch

Example batch file

<https://github.com/sbl-sdsc/df-parallel/blob/main/problem.sh>

Submit job

```
sbatch problem.sh
```

Papermill (<https://papermill.readthedocs.io/en/latest/>)

- execute notebook on the command line

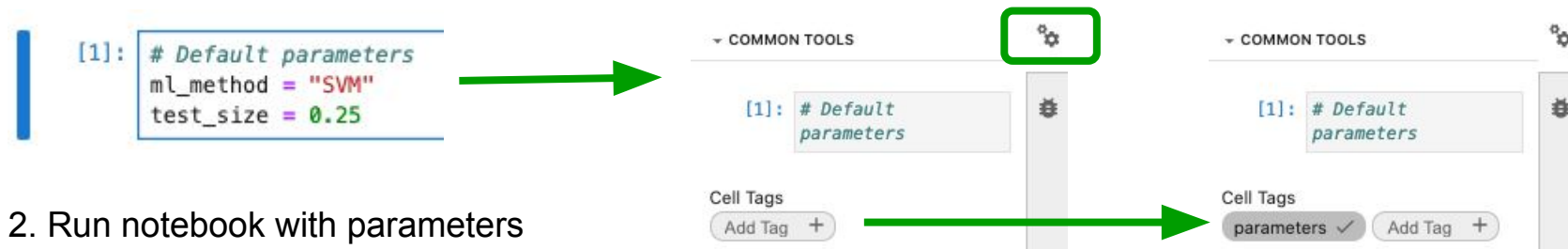
```
papermill input.ipynb output.ipynb
```
- execute notebook in Python

```
import papermill as pm
pm.execute_notebook("input.ipynb", "output.ipynb")
```
- parameterize notebook (pass arguments to Jupyter Notebooks)

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

Parameterize a Notebook

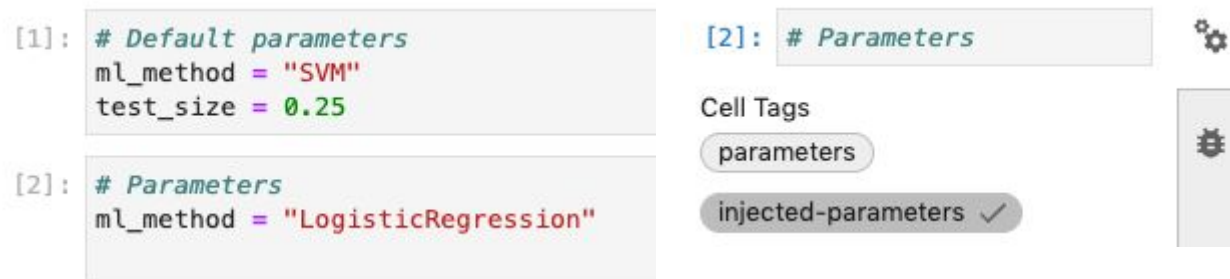
1. Select cell, add “parameters” tag and save notebook



2. Run notebook with parameters

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

3. The output “executed” notebook has parameter injected



Task 4 - Run Jupyter Lab in Batch

- Parameterize a dataframe notebook
- Edit problem.sh batch file
- Submit the batch job
- Examine the “executed” output notebooks
- Follow the instructions for [Task 4](#)

Get ready to use Expanse: accounts, allocations

Expanse Allocation

- Trial allocation: 100 GPU and/or 1000 CPU hours
consult@sdsc.edu
- Apply for allocations through ACCESS
<https://allocations.access-ci.org/prepare-requests-overview>

Allocation	Credit Threshold
<u>Explore ACCESS</u>	400,000
<u>Discover ACCESS</u>	1,500,000
<u>Accelerate ACCESS</u>	3,000,000
<u>Maximize ACCESS</u>	Not awarded in credits.

Allocation Levels

Explore	Discover	Accelerate	Maximize
Resource evaluation, grad student projects, small classes and training events, benchmarking, code development and porting, similar small-scale uses.	Grants with modest resource needs, Campus Champions, large classes and training events, NSF graduate fellowships, benchmarking and code testing at scale, gateway development.	Mid-scale resource needs, consolidating multi-grant programs, collaborative projects, preparation for Maximize ACCESS requests, gateways with growing communities.	Large-scale research projects.

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 GPU SU.
So this job will use 1 GPU SU even though it's just run for 30 minutes.


Best Practices for Authoring Jupyter Notebooks



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>

331

Save

78

Citation

53,861

View

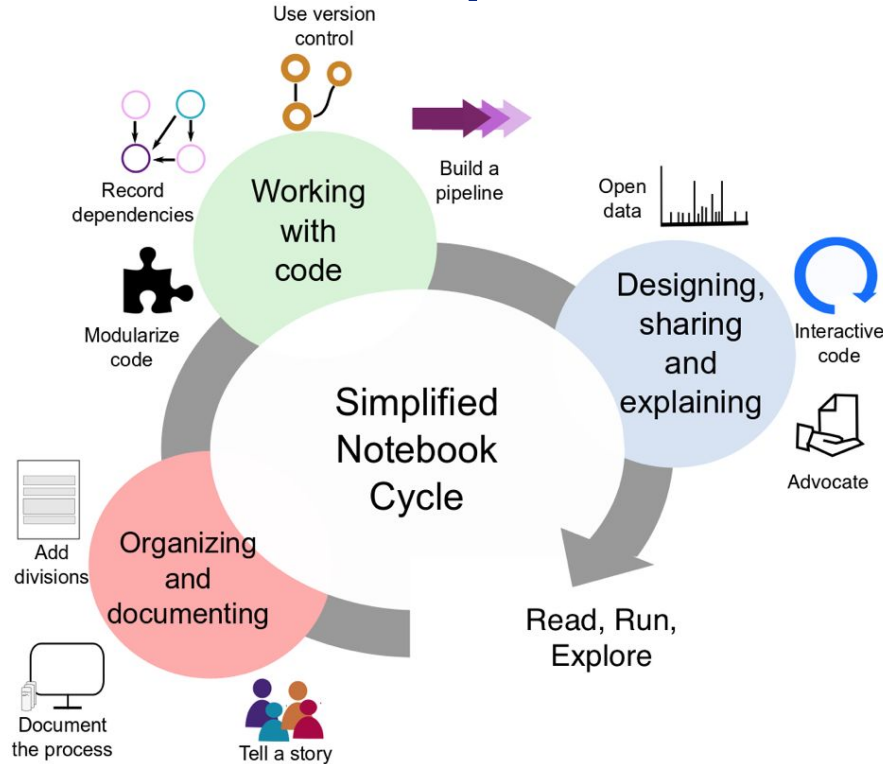
987

Share

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

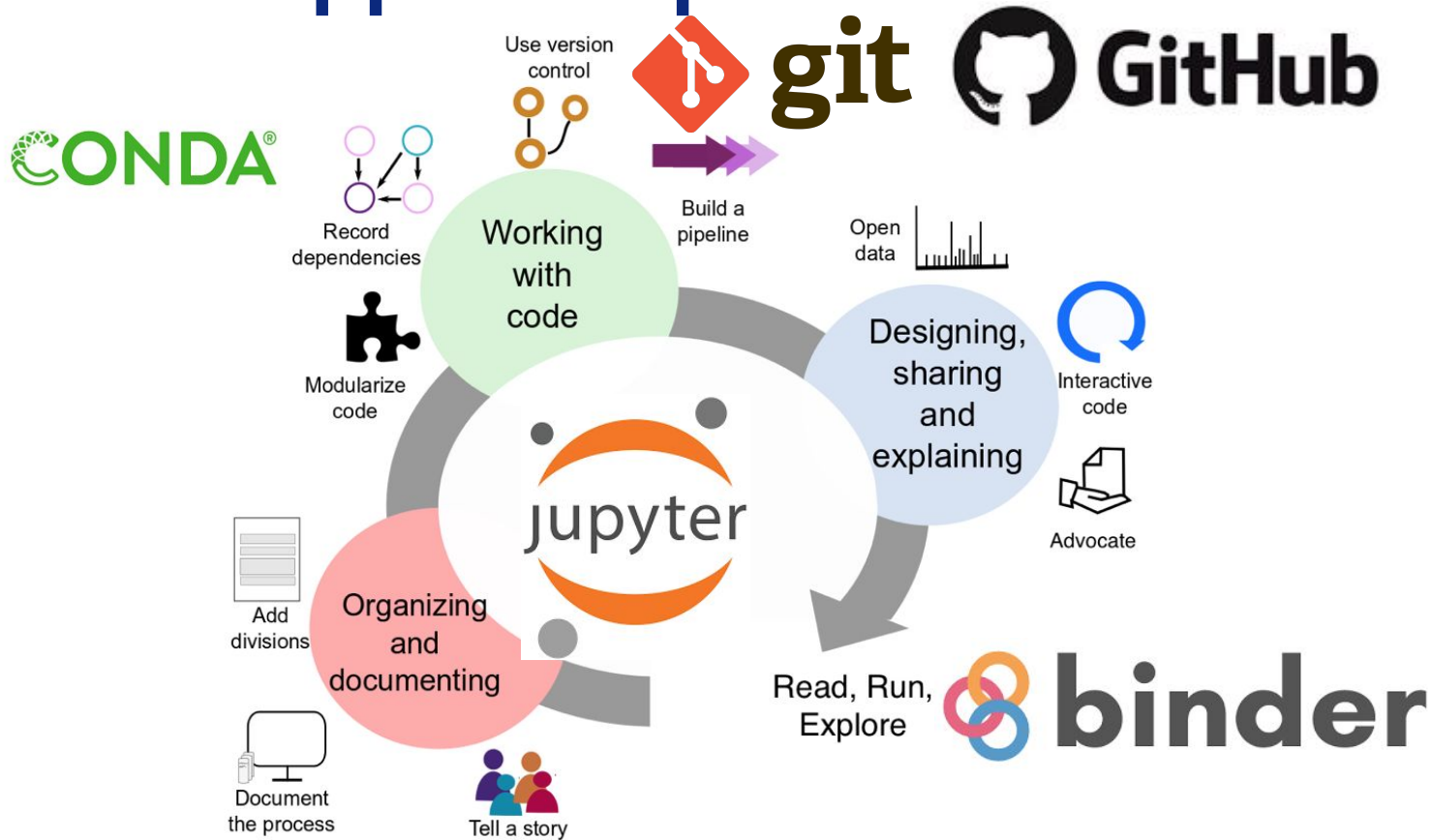
Git repo: <https://github.com/jupyter-guide/ten-rules-jupyter>

Ten Simple Rules



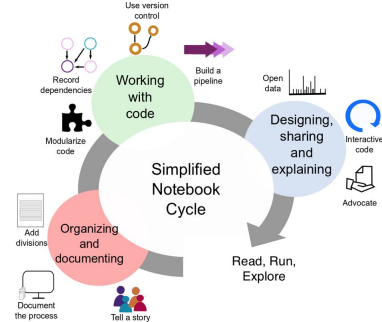
Ten Simple Rules for Writing and Sharing Computational Analyses in Jupyter Notebooks, PLOS Comp. Biol. 2019,
<https://doi.org/10.1371/journal.pcbi.1007007>

Tools to Support Reproducible Workflows

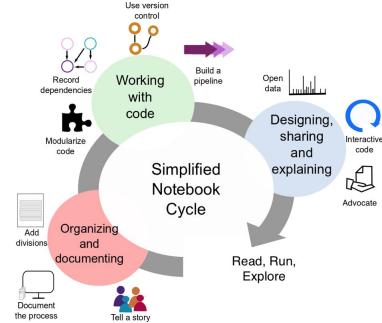


Organizing and Documenting

- **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

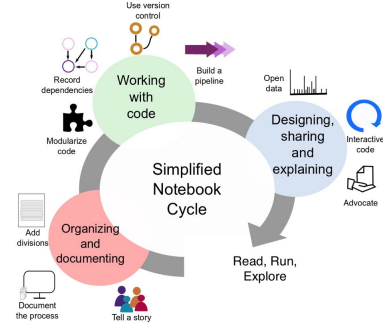


Working with Code



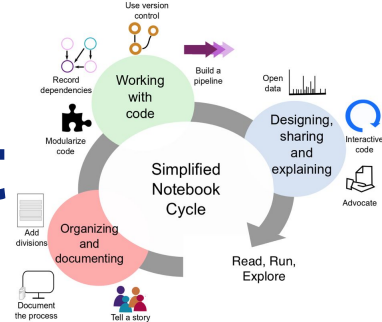
- **Rule 4: Modularize Code**
 - Use functions instead of duplicating code cells
- **Rule 5: Record Dependencies**
 - Manage your dependencies explicitly from the start using a tool such as
 - Conda's `environment.yml`
 - pip's `requirements.txt`
- **Rule 6: Use Version Control**
 - Consider using a 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](#)



The Scientific Paper Is Obsolete

Here's what's next.

theatlantic.com

The binder Project

<https://mybinder.org/>

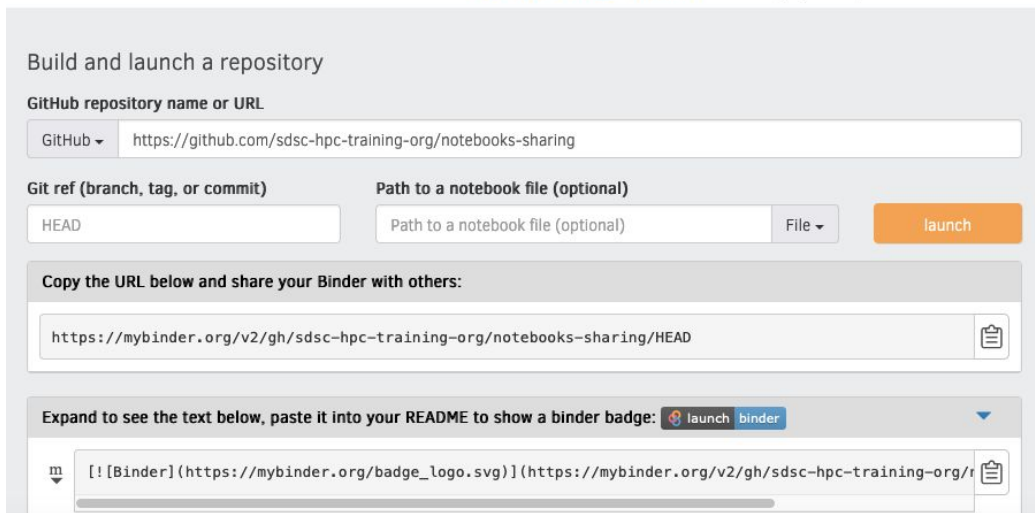
A community that builds free and open-source tools
for reproducible, sharable scientific environments
that are workflow- and platform-agnostic.



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



Git ref (branch, tag, or commit) Path to a notebook file (optional)

HEAD File

Copy the URL below and share your Binder with others:



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` 

Binder Limitations:

Provides a maximum of 2GB of memory

Only suitable for small projects

- few dependencies
- small data files

Launch can be slow or fail

Colab

- Works on a single notebook, not a whole Git repo
- Push changes directly to GitHub
- Cannot (easily) create a Conda environment
- Use pip install commands in notebook
- Supports GPUs
- Install rapidsai (includes cuDF)
 - see section “Cloud Instance GPUs”:
Google CoLab w/ pip” (<https://docs.rapids.ai/install>)

Summary

- When to run on Expanse
- Setup a portable and reproducible software environment
- Use the Expanse Portal
- Run Jupyter Lab on Expanse
- Scale up calculations on CPU/GPU
- Parameterize a Jupyter Notebook
- 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