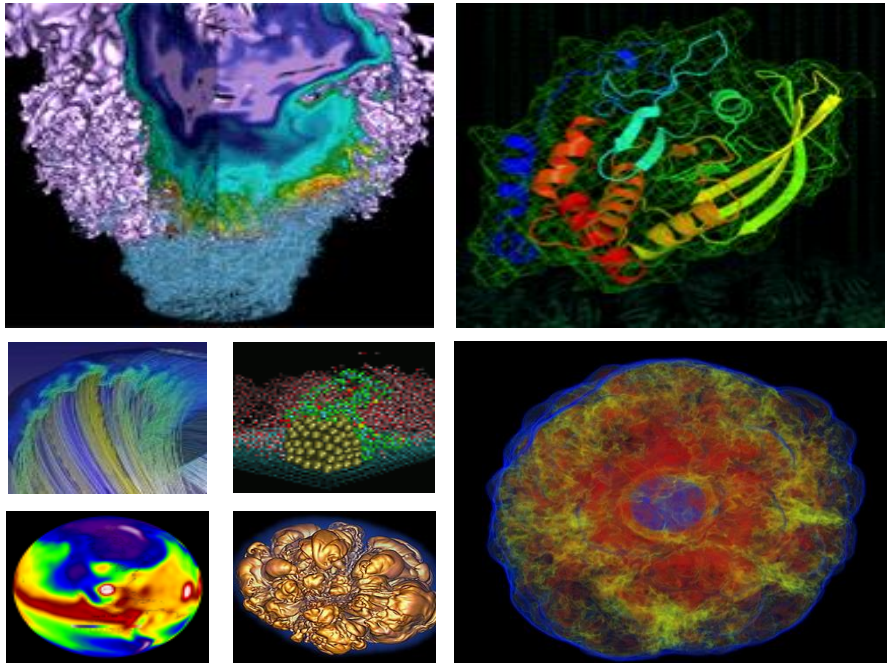
# Containers for Reproducible Workflows



Shane Canon
(Container Evangelist)
Data & Analytics Group, NERSC

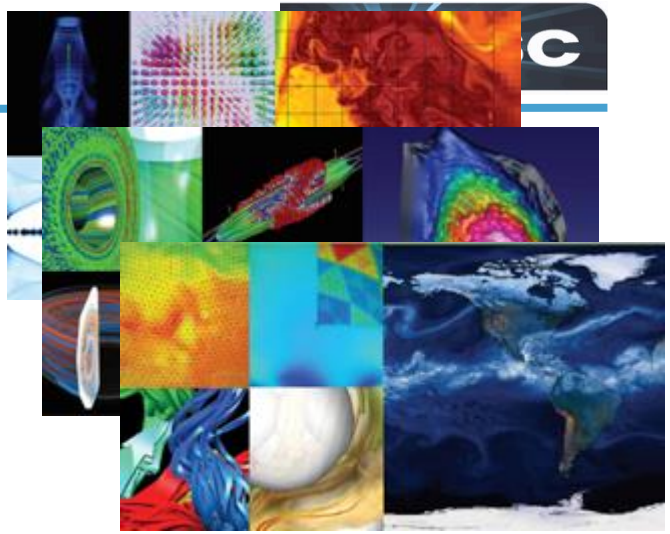
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

# Contents

- **Background and Why Containers**
- **Reproducibility and How Containers Can Help**
- **Gaps and Challenges**
- **Summary**

# Why Containers at NERSC



- **NERSC deploys advanced HPC and data systems for the broad Office of Science community**
- **Approximately 6000 users and 750 projects**
- **Growing number of users around Analyzing Experimental and Observational Data, ”Big Data” Analytics, and Machine Learning**
- **Shift towards converged systems that support traditional modeling and simulation workloads plus new models**

# The Struggles

- **My software doesn't build on this system…**
- **I'm missing dependencies…**
- **I need version 1.3.2 but this system has version 1.0.2..**
- **I need to re-run the exact same thing 12 months from now…**
- **I want to run this exact same thing somewhere else…**
- **I want my collaborators to have the same exact software as me…**
- **I've heard about these Containers, can I just run that?**
- **Can I run docker on this HPC system?**

# Solution - Containers


**What are Containers?**

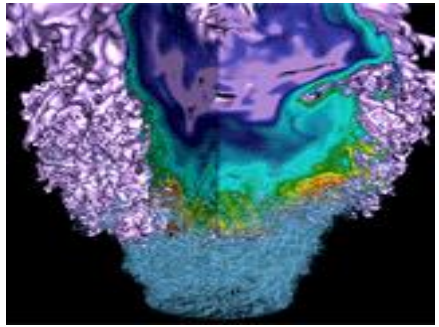
- **Uses a combination of Kernel “cgroups” and “namespaces” to create isolated environments**
- **Long history of containers Solaris Zones (2005), LXC(2008), LMCTFY/Google and then Docker(2013)**
- **Docker provided a complete tool chain to simplify using containers from build to run.**
- **Entire ecosystem has grown around containers especially around orchestration.**
- **Multiple HPC Container runtimes – Shifter, Singularity, Charliecloud, Sarus**

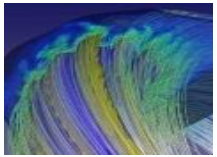

SHIFTER

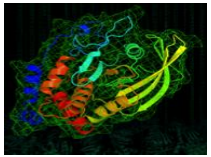
# Containers and Science

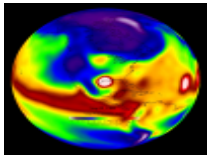
- **Productivity**
  - Pick the OS that works best for your app and use the system package manager to install dependencies.
- **Reusability and Collaboration**
  - Share images across a project to avoid rebuilds and avoid mistakes
- **Reproducibility**
  - Everything you need to redo a scientific analysis can be in the image (apps, libraries, environment setup, scripts)
- **Portability**
  - Can easily run on different resources (of the same architecture)

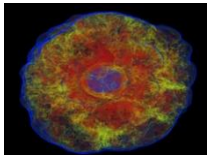
# Reproducibility

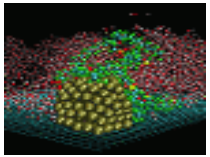










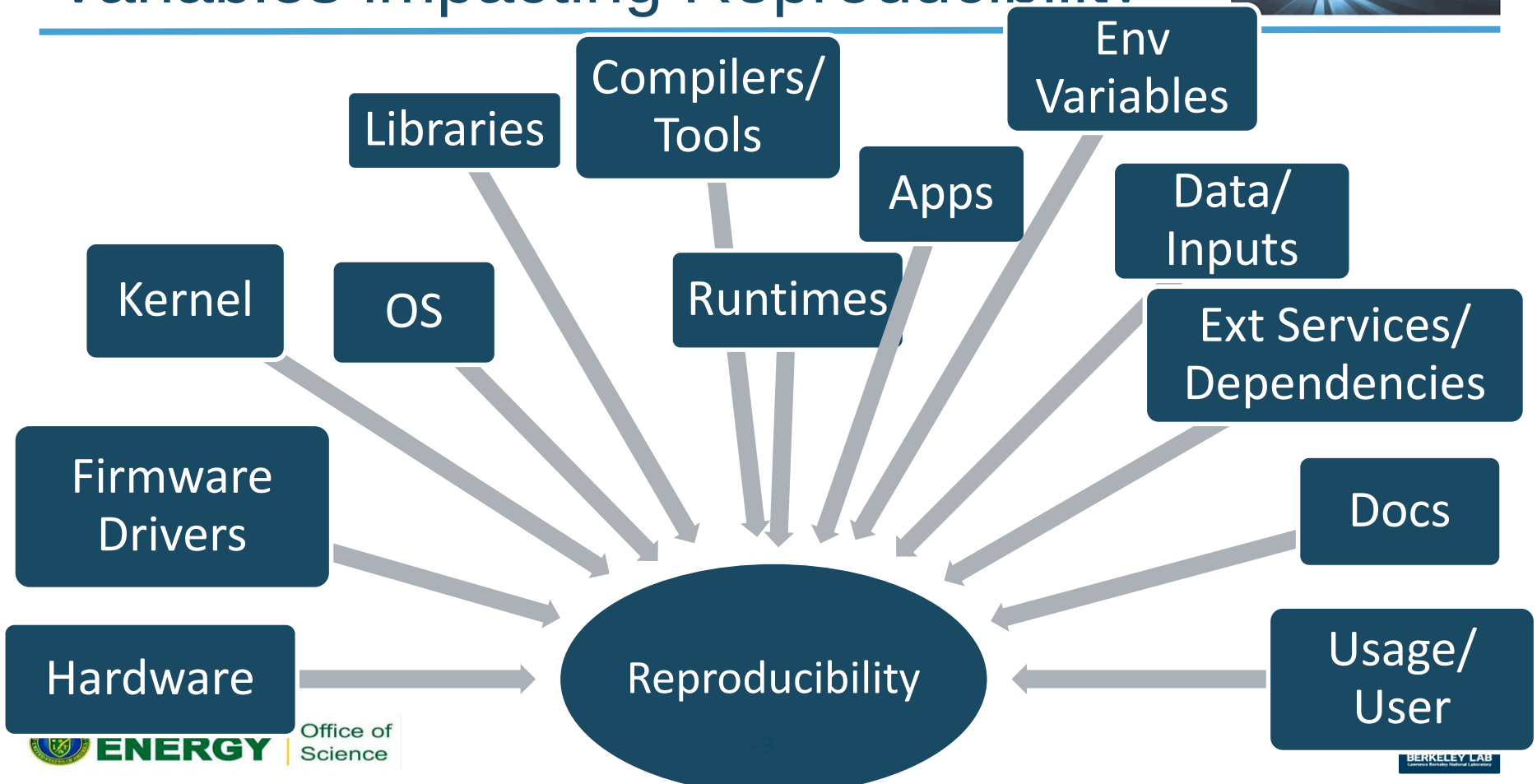


# Forms of Reproducibility Failures

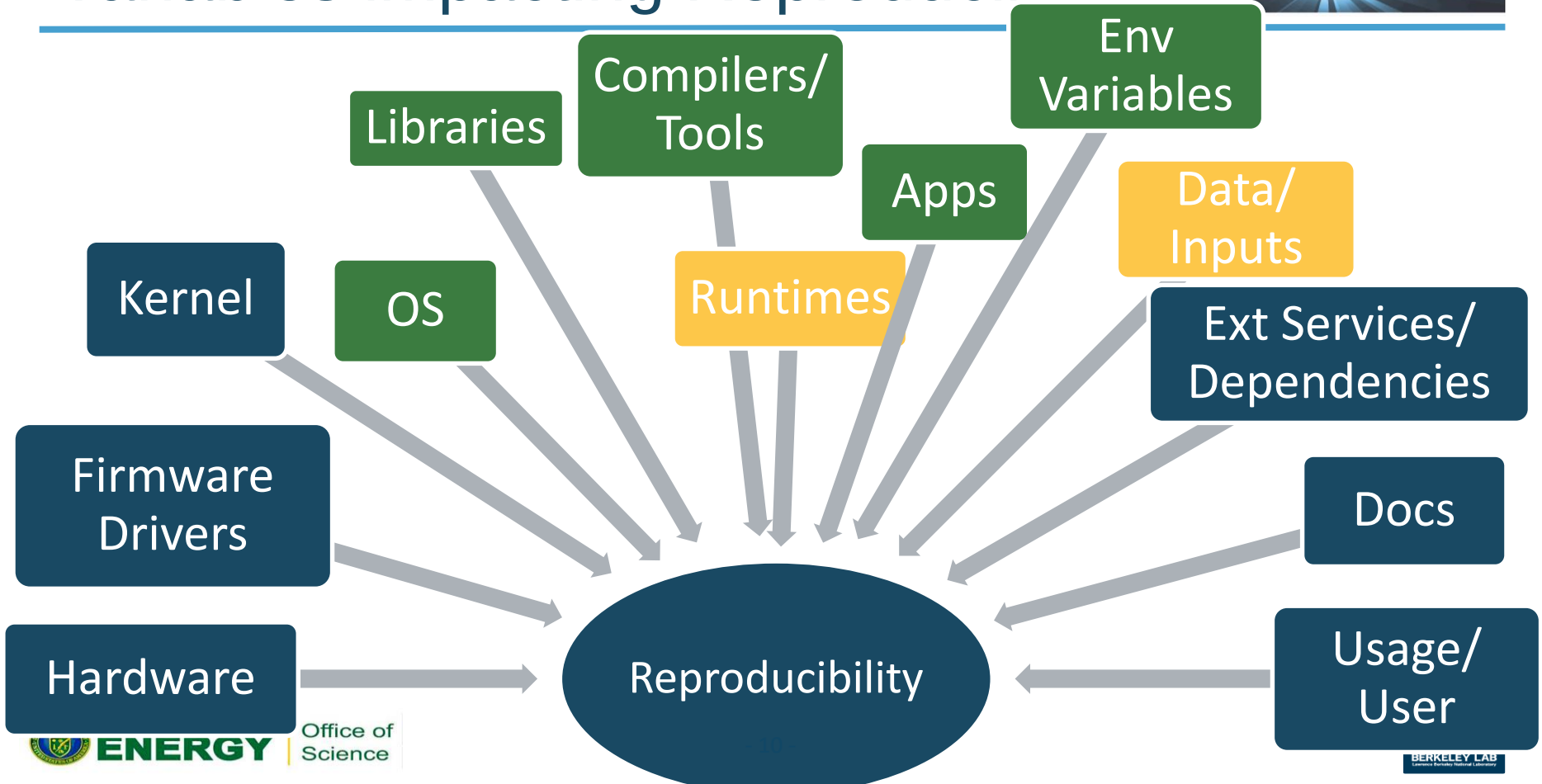
- **I can't reproduce my own results because:**
  - Something on the system changed
  - I can't recompile/build the code any longer
  - I can't find the prerequisites any longer
  - The system no longer exists

- **I can't reproduce someone else's result because:**
  - I can't gather the software any longer
  - Requirements or versions were poorly described or documented
  - I can't access to data
  - I don't have access to the appropriate system or hardware

# Variables impacting Reproducibility

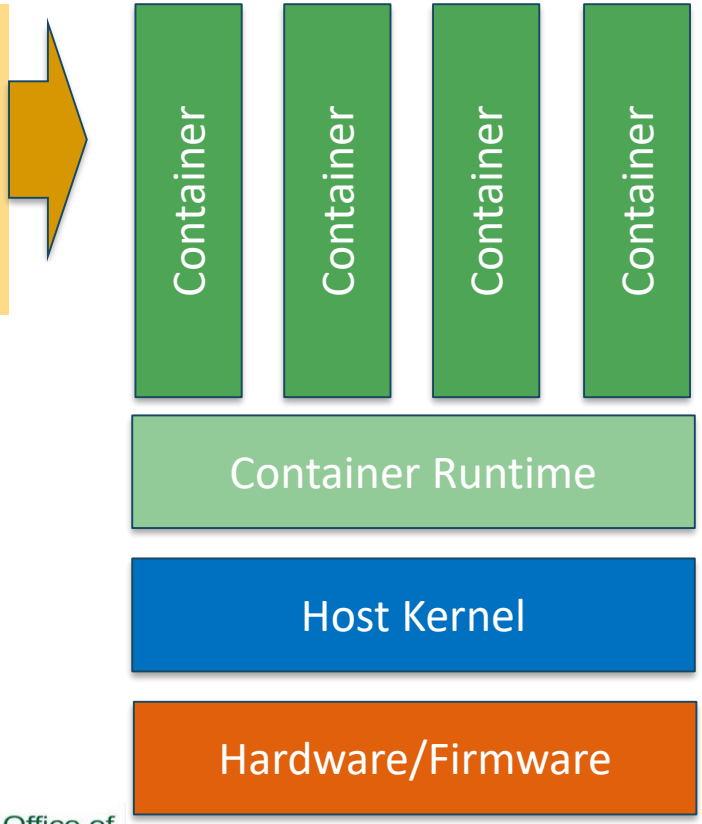


# Variables impacting Reproducibility



# Visual

- Data
- Execution
- External Services




- Linux Distribution
- Libraries
- Tools
- Compilers
- Application
- Data*
- Environment Variables
- Startup

# Example - Dockerfile

```
FROM myproj/mybase:2019.10.15

RUN apt-get -y install
ADD requirements.txt .
RUN pip install -r requirements.txt

ADD . /app
ENV PATH=/app
ENV FOO=BAR
```

Well defined starting point

Requirements file can included versioned packages

Add your app

Customize the environment for your app

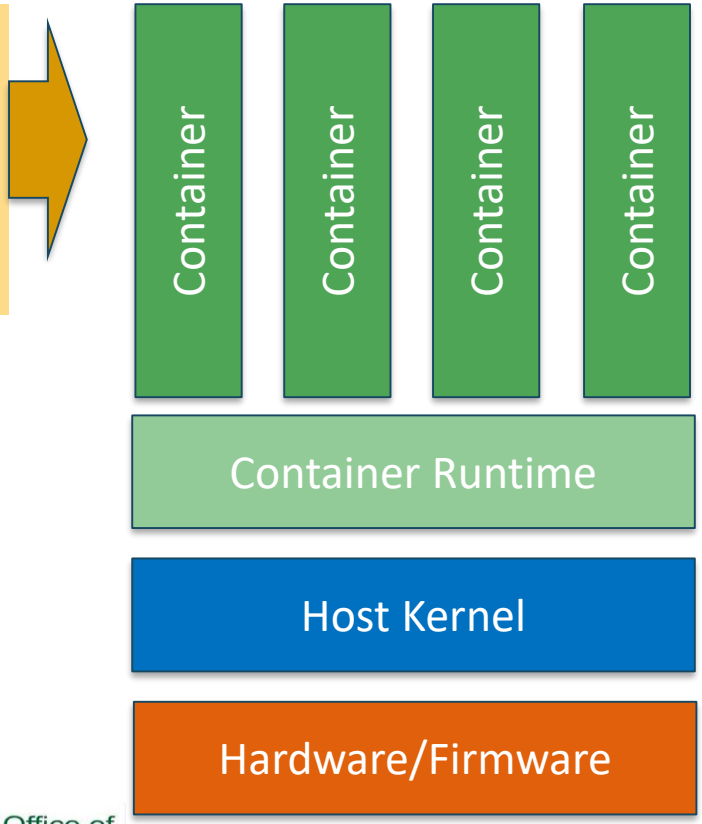
# Image Build Reproducibility

- **Only as good as the weakest link**
- **External repos and package managers introduce variation (e.g. yum, apt, pip, etc)**
- **Can be mitigated with good practices…**
- **…but not entirely**

Tagged and curated images and base images are the more feasible approach.

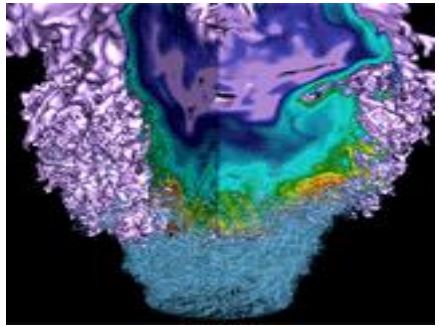
# Runtime Reproducibility

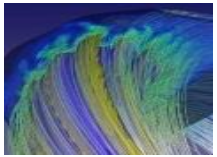
- Data
- Execution
- External Services

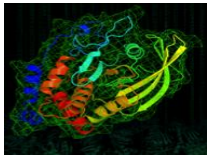


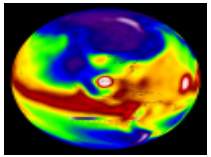

- Linux Distribution
- Libraries
- Tools
- Compilers
- Application
- Data*
- Environment Variables
- Startup

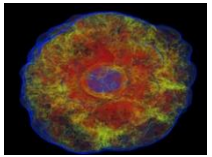
# Gaps and Challenges

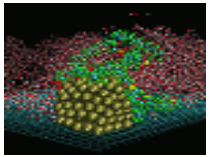






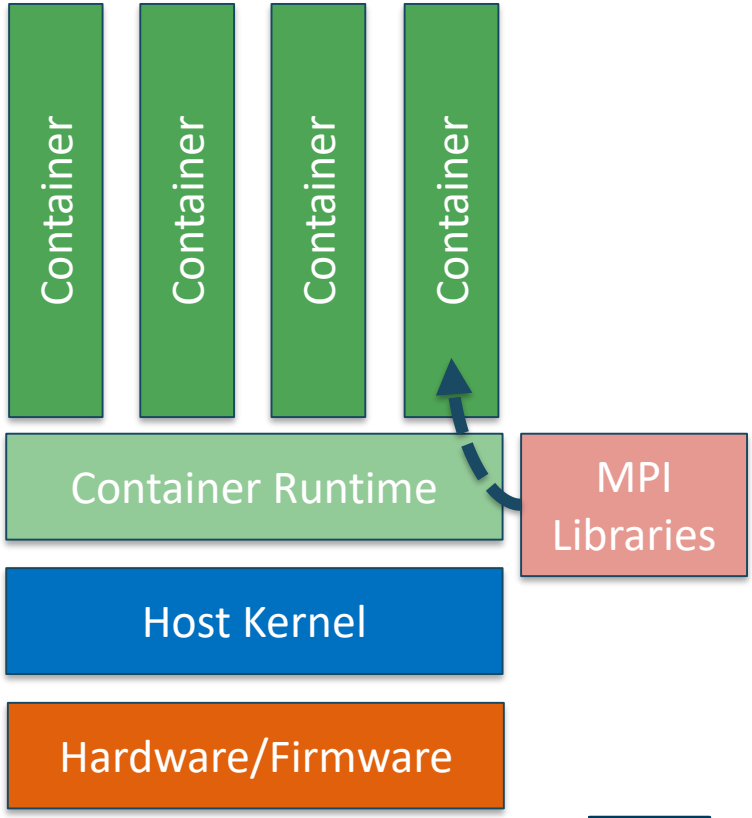




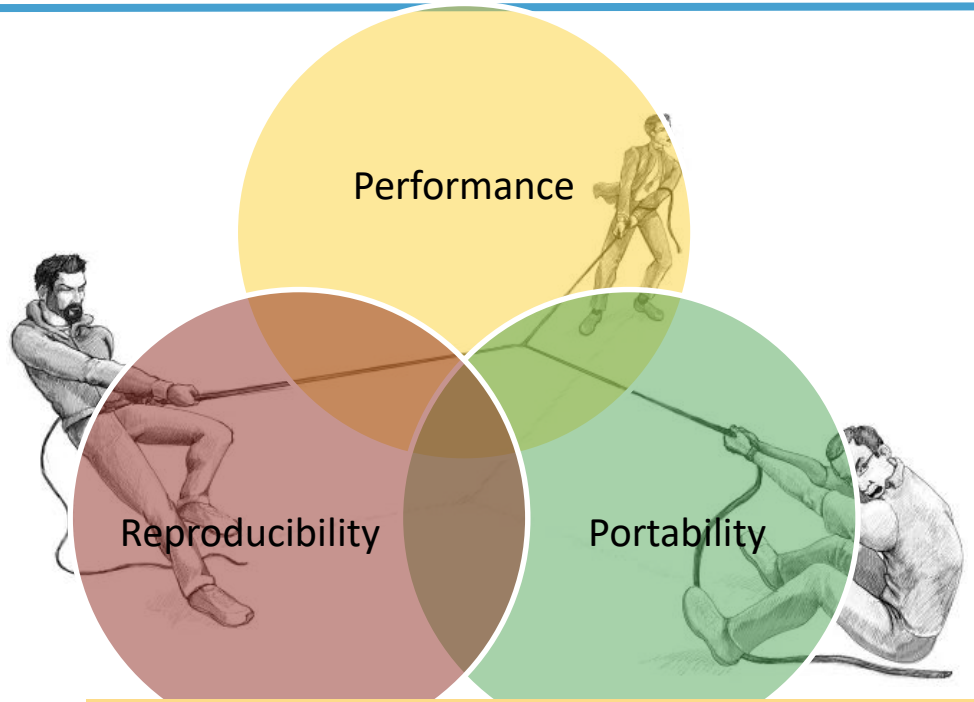


# MPI/GPU caveats

- **HPC Runtimes typically exploit application binary interface (ABI) compatibility to achieve native MPI performance**
- **Similar tricks used for GPUs**
- **These methods do introduce variations that could impact strict reproducibly**
- **… nothing's perfect**



# Competing Goals




Achieving “Ideal” Reproducibility may impact performance and portability

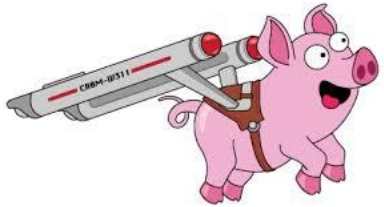
# Gaps and Improvements

- **Better abstractions**
  - Container to Device interfaces
  - Container to resource manager abstractions
- **Tools, interfaces and model for packaging data too**
  - Data Containers

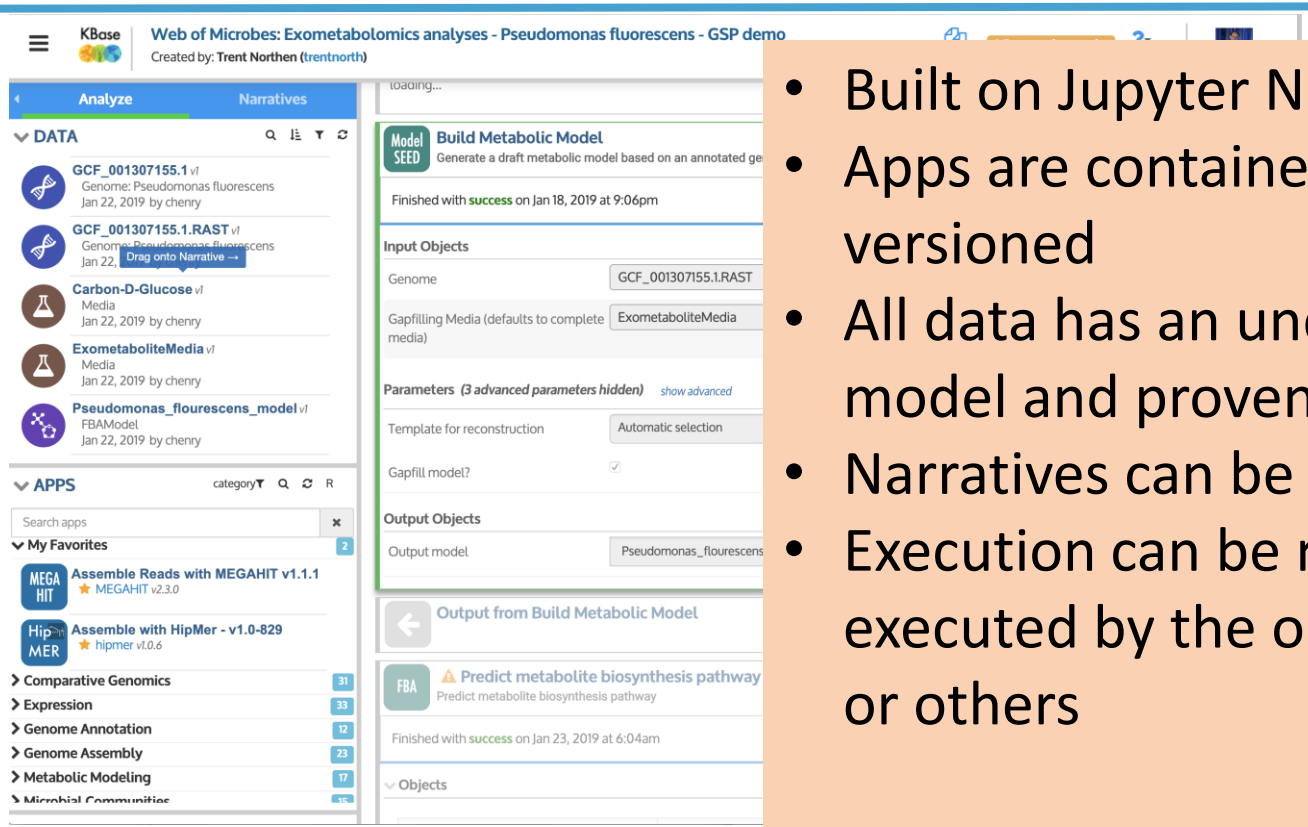
# Workflow Languages - CWL/WDL

- **Workflow Description Standards and Tools**
- **Richer model to express the steps of a workflow and how data flows between steps**
- **”Built-in” model for integrating with Containers**
- **Extensions to capture Provenance in a “standard” format**
- **Still requires best practices**

COMMON WORKFLOW LANGUAGE



# An example from Biology - KBase




- Built on Jupyter Notebooks
- Apps are containerized and versioned
- All data has an underlying data model and provenance
- Narratives can be shared
- Execution can be reproducibly executed by the original author or others

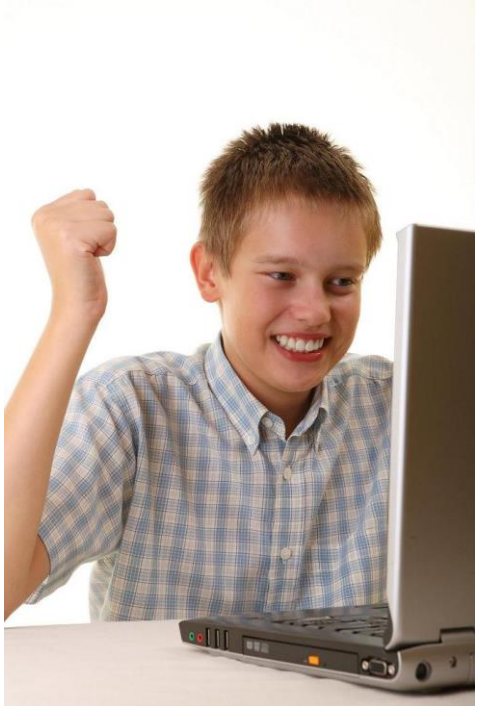
# (Healthy) Best Practices

- **Recipe based builds (e.g. Dockerfile)**
- **Add packages by version (e.g. pip, conda, apt, etc)**
- **Versioned/tagged Base Images and Images**
- **Archive and publish critical images**
- **Cite/reference the version/hash of any images**

# Summary

**Containers can play a role in improving Reproducibility**

- ✓ **Encapsulates key aspects required for an applications (reproducible) execution**
- ✓ **Not a silver bullet but greatly helps**
- ✓ **Portable – Run the same software on different resources (assuming architectural compatibility)**
- ✓ **Sharable – Collaborators can run the same code as you with less chance of problems**
- ✓ **Reusable – Others can reuse your analysis for their own data**



# Gratuitous Plug

**Several great Container related activities at SC19**

- **Tutorial – Sunday**
- **CANOPIE-HPC Workshop – Monday**
- **Container BOF – Wednesday**


**Slack Team - hpc-containers**

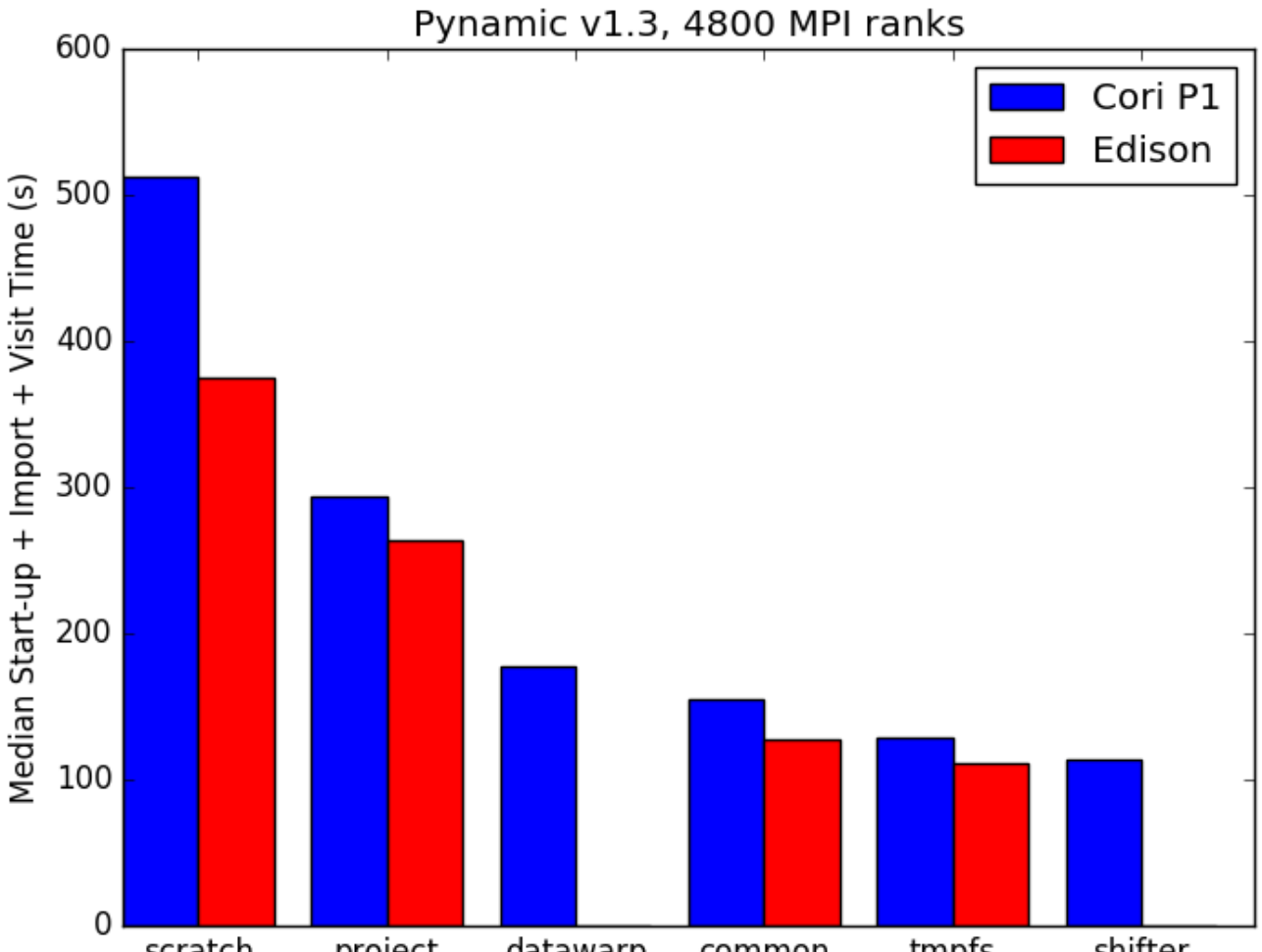
# Questions…

Shane Canon: scanon@lbl.gov

Thanks to Claudia for the invitation and opportunity to share

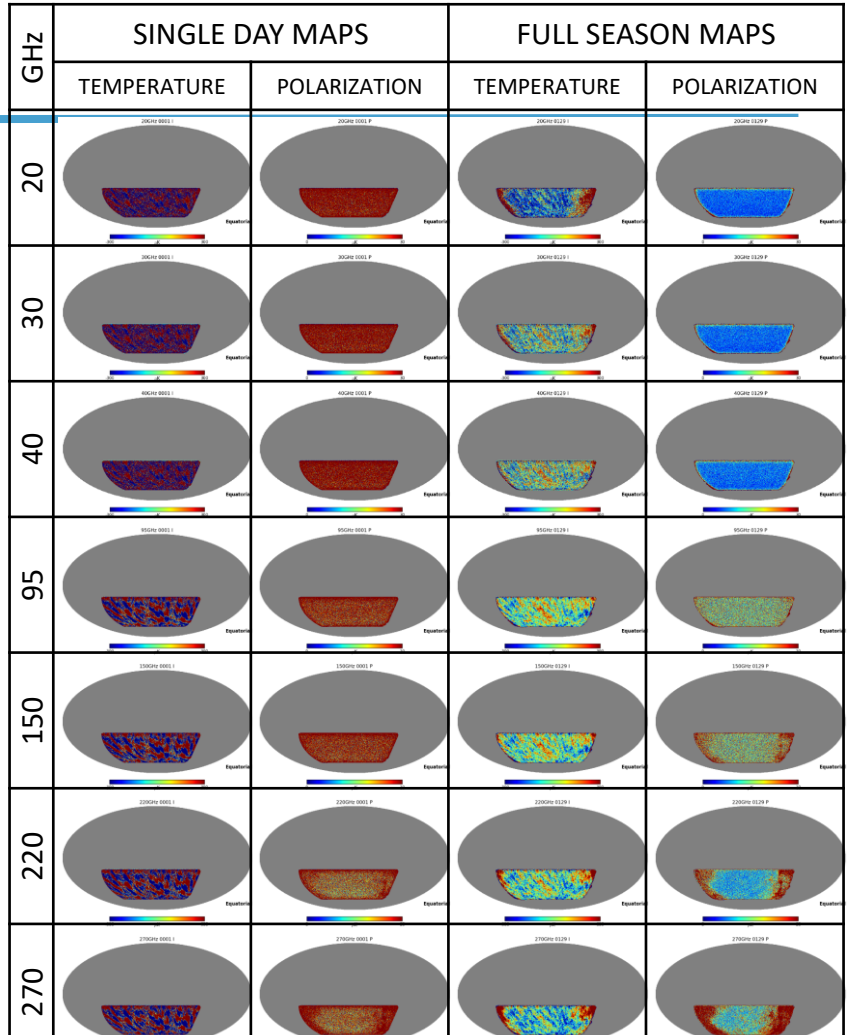
This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of

# Shi


Pynamic v1.3, 4800 MPI ranks
Median Start-up + Import + Visit Time (s)
0
100
200
300
400
500
600
Cori P1
Edison
scratch
project
datawarp
common
tmpfs
shifter

# Measuring the Composition of the Universe

- **CMB – S4**
  - Ambitious collection of telescopes to measure the remnants of the Big Bang with unprecedented precision
- **Simulated 50,000 instances of telescope using 600,000 cores on Cori KNL nodes.**
- **Why Shifter?**
  - Python wrapped code needs to start at scale




U.S. DEPARTMENT OF ENERGY | Office of Science

C3 COMPUTATIONAL COSMOLOGY

# Why not just run Docker

- **Security: Docker currently uses an all or nothing security model. Users would effectively have system privileges**

```
> docker run -it -v /:/mnt --rm busybox
```

- **System Architecture: Docker assumes local disk**
- **Integration: Docker doesn't play nice with batch systems.**

- **System Requirements: Docker typically requires very modern kernel**
- **Complexity: Running real Docker would add new layers of complexity**

# Solution: Shifter


SHIFTER

- **Design Goals:**
  - User independence: Require no administrator assistance to launch an application inside an image
  - Shared resource availability (e.g., file systems and network interfaces)
  - Leverages or integrates with public image repos (i.e. DockerHub)
  - Seamless user experience
  - Robust and secure implementation
- **Hosted at GitHub:**
  - https://github.com/nersc/shifter