USPAS – *Simulation of Beam and Plasma Systems*

Steven M. Lund, Jean-Luc Vay, Remi Lehe, Daniel Winklehner and David L. Bruhwiler

Lecture: Computational Reproducibility

Instructor: David L. Bruhwiler

Contributors: R. Nagler and P. Moeller

radiasoft

U.S. Particle Accelerator School sponsored by **Old Dominion University**

http://uspas.fnal.gov/programs/2018/odu/courses/beam-plasma-systems.shtml

January 15-26, 2018 – Hampton, Virginia

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Offices of High Energy Physics and Basic Energy Sciences, under Award Number(s) DE-SC0011237 and DE-SC0011340.

U.S. DEPARTMENT OF ENERGY

Office of Science

Motivation

- Scientific research should be reproducible
  - every scientist agrees in principle
- There are three primary branches of scientific inquiry
  - theory: logic, physics & mathematics must be clear and correct
    - **essentially no problem, assuming rigorous peer review**
  - experiment: descriptions of apparatus, methods, data collection & analysis
    - **producing and/or collecting data is beyond the scope considered here**
    - **data archival, post-processing, analysis and viz should all be reproducible**
  - computation: similar to experiments, but it's possible to do much better
    - **this is the focus of today's lecture**
    - **software is inherently replicable**
- Funding agencies (i.e. the community) is asking for reproducibility
  - US Department of Energy requires a Data Management Plan
    - **https://science.energy.gov/funding-opportunities/digital-data-management**

> Sharing and preserving data are central to protecting the integrity of science by facilitating validation of results and to advancing science by broadening the value of research data to disciplines other than the originating one and to society at large. To the greatest extent and with the fewest constraints possible, and consistent with the requirements and other principles of this Statement, data sharing should make digital research data available to and useful for the scientific community, industry, and the public.


What is meant by Reproducibility?

- Our intuition comes from the long history of experimental science
  - other scientists try to reproduce an experimental result with their own equipment
  - this attitude has carried over to computational science
- However, software and computation are inherently replicable
  - two scientists should be able to get identical results from ***the same*** code
- Reproducibility implies something more:
  - two scientists should be able to get identical results from ***different*** codes
  - this implies the two codes have been successfully benchmarked
- For this lecture, "computational reproducibility" means "replicability"
  - replicability is the essential first step
- Some links to discussion of the differences in meaning:

M. Liberman (2015), http://languagelog.ldc.upenn.edu/nll/?p=21956

Replicability Research Group, http://www.replicability.tau.ac.il/index.php/replicability-in-science/replicability-vs-reproducibility.html

Wikipedia, https://en.wikipedia.org/wiki/Reproducibility

The Replication Network, https://replicationnetwork.com/2016/04/03/the-national-academy-of-sciences-weighs-in-on-reproducibility

Workshop: "Statistical Challenges in Assessing and Fostering the Reproducibility of Scientific Results" (2016), https://www.nap.edu/catalog/21915/statistical-challenges-in-assessing-and-fostering-the-reproducibility-of-scientific-results

Barriers to Reproducibility

- The first step would be to share simulations with a single code
  - Not just the plots: the full simulation, generating identical results
- Why don't we share our simulations directly?
  - It can be very difficult to reproduce computational results
  - Close collaborators do, of course, but even this can be difficult and error prone
    - **different hardware, different versions of the source code, many input files, etc.**
- Political and social reasons (incomplete list)
  - don't want to share years of effort with others, who would directly benefit
  - software license may forbid sharing
  - software can be used incorrectly
    - **this might generate confusing results that are incorrectly published**
    - **this could hurt the reputation of the original developers**
- Technical reasons
  - the software may be difficult to build/install
  - the software may be difficult to use
  - the simulations may require many processors, generate large data sets, etc.
  - the post-processing tools may be sophisticated

Class discussion:

- Have you successfully shared a simulation with a colleague?
  - How positive was the experience?
  - Did you every try and fail?

- Briefly describe a code benchmarking exercise.
  - Which codes? Were others involved? Did you have access to all codes?
  - How positive was the experience?

- Under what circumstances should a code (or simulation) be made public
  - Do scientists have a right to protect their computational tools from competitors?
  - Does scientific ethics require sharing? If so, then how much?
  - Is it appropriate for funding agencies to require reproducibility?
  - Should journals require access to source code?
  - Should all code be open source?

- Who should bear the cost of reproducibility?
  - Should the journal have an editor dedicated to reproducibility?
  - Should the research project have to demonstrate reproducibility?

Validity – *this is essential to the issue of reproducibility*

- Computational science requires validated software
  - we want our results to be correct
  - need confidence that other published/presented results are correct
- Build environment and computing platform:
  - software is validated on a particular platform, compiler, etc.
    - **supporting multiple platforms is possible, but expensive**
  - what if the software is built, installed, executed on another platform?
    - **can one be sure that it is still valid?**
- Versioning:
  - a particular version of software is validated
    - **regular use of regression tests can help maintain validity across versions**
    - **however: tests are never complete, features are added/removed, etc.**
  - for multiple dependencies, versioning becomes an N^2 problem
  - how can one be sure of the validity of a particular software version?
    - **how can one communicate full versioning information to others?**
- Sharing:
  - difficult to share identical build & version(s), including dependencies

Usability – *without it, reproducibility loses a lot of value*

- Many physics codes are difficult to use
- Particle accelerator codes are a case in point
  - they focus on specific design aspects (not fully general)
  - most rely on complicated command-line interfaces (CLI)
    - **multiple input and configuration files**
    - **multiple output files (text, data, images) in different formats**
  - they typically require computational experts to use correctly
    - **even so, the assistance of an expert and/or much experience is often required**
  - collaboration is difficult and error prone
    - **required exchange of input and output files**
    - **independent installation of codes (different versions on different hardware)**
    - **use of different post-processing and visualization tools**
- If a code is difficult to use, what does reproducibility mean?
  - given a replicated environment to work in, what do you do next?
  - suppose a single script is provided to execute all other scripts and render plots
    - **does this black box experience enable understanding of what is being replicated?**
- Reproducibility should enable understanding, comparison & further work
  - the various steps required should each be clear and replicable
  - it should be possible to modify each step and see the results


Usability via Python – *an example of problems encountered*

- Python wrapping of scientific applications is popular
- This approach can be powerful:
  - extend capabilities with a powerful, expressive language
  - develop convenient GUIs and CLIs
  - develop a common interface to multiple codes
- It also creates serious challenges:
  - Python 2.7.x code is not always compatible with Python 3.x code
  - 32 bit and 64 bit versions of Python are incompatible
    - **also true of many other software libraries**
  - open source library projects issue frequent releases
    - **breaks compatibility with previous versions of other libraries**
  - Python is amazingly cross-platform; however…
    - **C/C++ and Fortran libraries and compilers less so**
  - a computer may have multiple versions of installed Python
    - **this creates additional complexity**

Class discussion:

- When you run a simulation how confident are you in the results?
- Are you skeptical of results presented by others in talks or papers?
  - have you ever wanted to rerun someone else's simulation?
- How do you validate your own results?
  - compare with your physical intuition?
  - compare with your previous work?
  - compare with publicly available papers and presentations?
  - rerun the same case with a different code (or ask a colleague to do it…)?
  - discuss the plots with a colleague…?

- Maybe revisit previous discussion points –
  - Have you successfully shared a simulation with a colleague?
    - **How positive was the experience?**
    - **Did you every try and fail?**
  - Briefly describe a code benchmarking exercise.
    - **Which codes? Were others involved? Did you have access to all codes?**
    - **How positive was the experience?**

The reproducibility literature (pt. 1)

- The topic cannot be covered completely in a single lecture…
  - so we provide some pointers for further study
  - note that this is an active area of research

- Best practices, tips, tools and techniques:

The Software Sustainability Institute, https://www.software.ac.uk/ssisearch?search_api_fulltext_1=reproducible

V. Stodden (2017), https://codeocean.com/workshops/computational-reproducibility

V. Stodden & S. Miguez (2014), https://openresearchsoftware.metajnl.com/articles/10.5334/jors.ay/

S.R. Piccolo & M.B. Frampton (2016), https://gigascience.biomedcentral.com/articles/10.1186/s13742-016-0135-4

G.K. Sandve et al. (2013), http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003285

J. Freire, P. Bonnet & D. Shasha (2012), https://pdfs.semanticscholar.org/57ee/c0917fc84716e5748c2e94139ab156db3ada.pdf

- Reproducibility, reusability, etc. – what is it? …why do it?

O. Guest & N.P. Rougier (2016), https://hal.inria.fr/hal-01358082/file/guest_rougier_2016.pdf

G. Varoquaux (2017), http://gael-varoquaux.info/programming/beyond-computational-reproducibility-let-us-aim-for-reusability.html

rOpenSci Project, http://ropensci.github.io/reproducibility-guide/sections/introduction/

L.A. Barba (2017), https://speakerdeck.com/labarba/introduction-to-computational-reproducibility-and-why-we-care

The reproducibility literature (pt. 2)

- A few more links
  - these are just a sampling of resources; not comprehensive
  - don't read them all; skim to find ones that interest you

- Biology and health sciences:

L. Hatton & G. Warr (2016), https://arxiv.org/abs/1608.06897

B. Grüning *et al*. (2017), https://www.biorxiv.org/content/early/2017/10/10/200683

S. Cohen-Boulakia *et al*. (2017), https://www.sciencedirect.com/science/article/pii/S0167739X17300316

B.K. Beaulieu-Jones & C.S. Greene (2017), https://www.nature.com/articles/nbt.3780

- How can journals (or funding agencies) incentivize scientists…?

P. Montagano & S. Green (2018), https://codeocean.com/webinar/editor

R. Nagler & D.L. Bruhwiler (2018), https://www.researchgate.net/publication/322438060_The_Replication_Tax_Shifting_the_Financial_Burden_to_Incentivize_Reproducibility_in_Computational_Research

Heads up – today's homework

- Read ***at least one*** of the above references, and…
  - Write 3 paragraphs on computational reproducibility:
    - **e.g. How could improved computational reproducibility benefit your science?**
    - **e.g. How could it improve your institution?**
    - **e.g. Describe an experience with computational reproducibility (or the opposite).**
    - **e.g. Discuss what you think would be most effective for improving reproducibility.**
    - **e.g. Discuss pros/cons, trade-offs or how to address cost-benefit concerns.**
    - **…or choose your own theme**
  - Be sure to make a connection with at least one of the above references

- The rest of the lecture ***is*** relevant to this assignment
  - Just giving you a heads up…

Near-term goals – *let's walk before trying to run…*

- Enable reproducible simulations with a single code (i.e. replicability)
  - two scientists ***will*** get identical results from the same code
  - more than two – as many scientists as are interested
- What is required?
  - make community codes portable, easy to install, publicly available
  - simple, easy sharing of code & full environment across users and systems
- And how is that done?
  - with application containers
    - **Docker (commercial, most widely used),** https://www.docker.com/what-docker
    - **Singularity, Shifter (newer, HPC),** https://tin6150.github.io/psg/blogger_container_hpc.html
- Caveat
  - there are other approaches to reproducibility; not just containers
    - **these can require significant infrastructure and constrained workflows**
  - we are focusing on community codes and individual scientists
    - **developers are independent & busy**
    - **there are many codes with diverse applications**
- What about usability?
  - IPython notebooks (on a Jupyter server) can get you part of the way
  - Intuitive, browser-based GUIs show greater promise
- Cloud-based approach minimizes the cost of reproducibility

What do containers do? How are they used?

- Application containers provide OS-level virtualization
  - deploy and run distributed applications
    - **most use cases involve small, serial apps and very short run-times**
    - **no need for an entire virtual machine (VM) for each app**
- Multiple isolated applications or services run on a single host
  - each accesses the same OS kernel
  - multiple containers compete for available resources
- Containers can be deployed on any computer system
  - bare-metal; cloud instances; virtual machines
- Containers are available on every OS
  - multiple flavors of Linux, with essentially no run-time overhead
  - MacOS and Windows (limited, with some computational overhead)

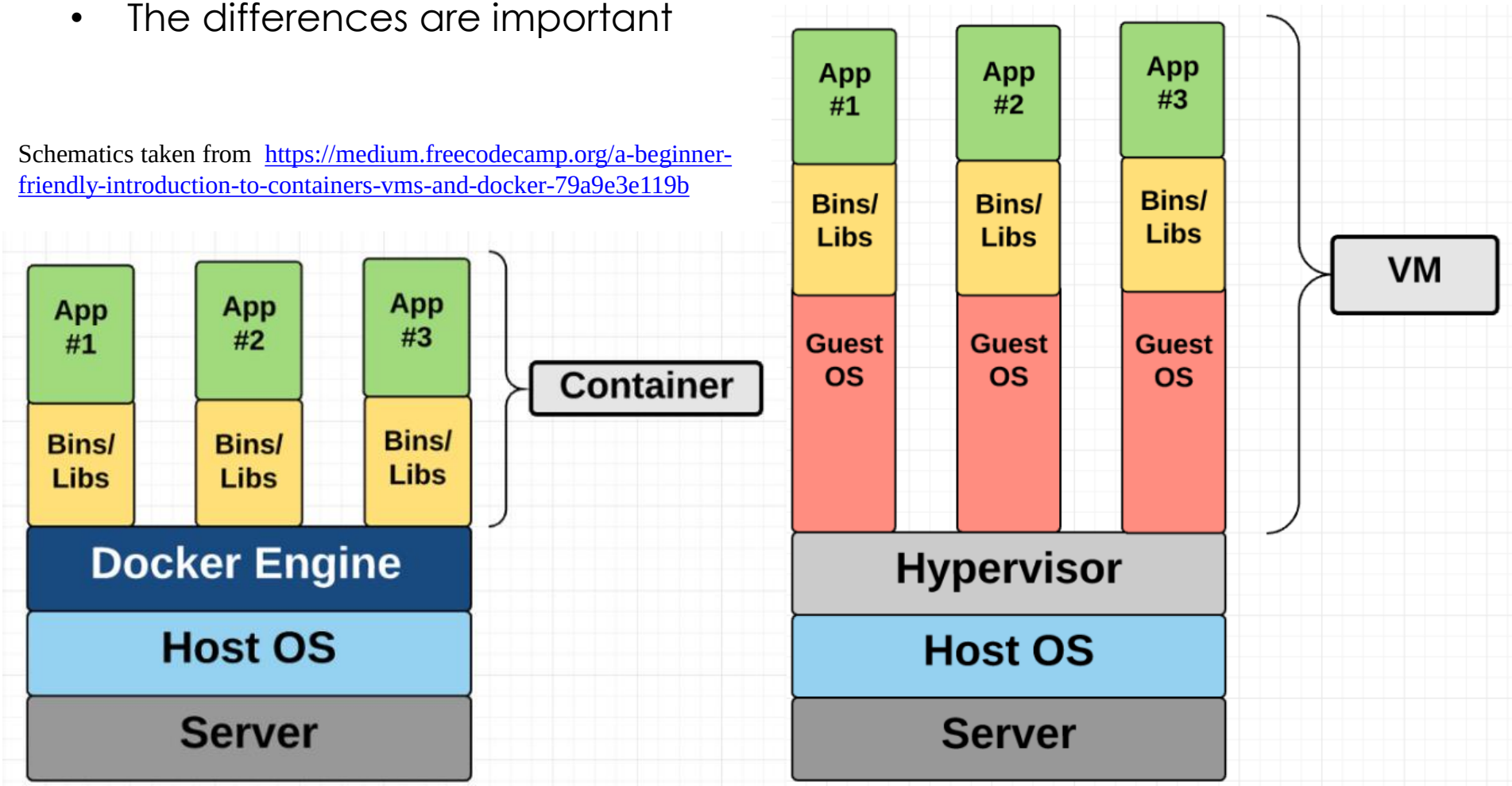
- HPC presents a fundamentally different use case
  - large apps: physics code, dependencies, viz & post-processing tools...
  - highly variable run-times (seconds to days) on multiple processors
  - data generation
    - **subdirs can be mounted to write files outside of the container**

Container vs. Virtual Machine (VM)

- Containers and VMs are similar in their goal:
  - to isolate an application and its dependencies
  - in a self-contained unit that can run anywhere

- The differences are important

Schematics taken from https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b



Application Containers for Reproducibility

- Containerization has become a key technology for reproducibility
  - not the only approach (see 'caveat' on slide #13 above)
  - it is a practical approach for replicating a simulation result in the future
- The colleague you are sharing with could by yourself in 6 months or 6 years
  - an application container can be archived indefinitely and then reused
  - assuming Docker or a compatible technology is still supported
- Some pointers for further study:

Docker Containers for Reproducible Research Workshop (2017), https://www.software.ac.uk/c4rr

G.M. Kurtzer, V. Sochat and M.W. Bauer (2017), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5426675/

D. Nüst & M. Hinz (2017), http://o2r.info/2017/05/30/containerit-package/

J. Cito, V. Ferme and H.C. Gall (2016), https://link.springer.com/chapter/10.1007/978-3-319-38791-8_58

N. Hemsoth (2016), https://www.nextplatform.com/2016/09/13/will-containers-total-package-hpc/

D.L. Bruhwiler, R. Nagler *et al*. (2015), http://accelconf.web.cern.ch/AccelConf/IPAC2015/papers/mopmn009.pdf

R. Nagler, D.L. Bruhwiler *et al*. (2015), https://arxiv.org/abs/1509.08789

C. Boettiger (2014), https://arxiv.org/abs/1410.0846

Class discussion:

- Do you have experience working with VMs?
  - If so, what are the advantages and disadvantages?
- Explain at least one difference between VMs and containers
- A VM is to a house as a container is to an apartment: explain the analogy
- Reproducibility vs Replication: which is more difficult? …more important?
- How does Docker help with replicating a physics simulation?

- Maybe revisit previous discussion points –
  - When you run a simulation how confident are you in the results?
  - Are you skeptical of results presented by others in talks or papers?
    - **have you ever wanted to rerun someone else's simulation?**
  - How do you validate your own results?
    - **compare with your physical intuition?**
    - **compare with your previous work?**
    - **compare with publicly available papers and presentations?**
    - **rerun the same case with a different code (or ask a colleague to do it…)?**
    - **discuss the plots with a colleague…?**

Running Particle Accelerator Codes inside Docker

- RadiaSoft has been running HPC codes via Docker since 2015
  - beam physics: elegant/SDDS, Warp, Synergia
  - X-ray optics & synch. radiation: SRW, Shadow
  - FELs: Genesis

- Linux
  - containers run at native speeds; MPI works well
  - large-scale I/O can be slow (similar to NFS; more testing required)
- MacOS
  - similar to Linux (we have less experience)
- What about supercomputing? *(don't know of any experience w/ accelerator codes)*
  - Singularity, (LBL) — Singularity, http://singularity.lbl.gov
  - Shifter (NERSC) — K. Kincade (2015), http://www.nersc.gov/news-publications/nersc-news/nersc-center-news/2015/shifter-makes-container-based-hpc-a-breeze/
  - Charliecloud (LANL) — R. Priedhorsky & T.C. Randles (2017), http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-16-22370
  - others...?

Delivering codes via the Docker Hub repository

- Docker images can be uploaded to a public repository
  - https://hub.docker.com
- RadiaSoft distributes our containers from Docker Hub
  - https://hub.docker.com/r/radiasoft
  - automated build/test/release is used to ensure working containers
- One command can be used to download/install/run:

```
> curl radia.run | bash -s beamsim
```

  - assumes Docker is already installed and you know how to use it
- JupyterHub servers provide cloud-based access to containerized codes
  - GitHub repository & docs, https://github.com/jupyterhub/jupyterhub
- RadiaSoft provides a public server, https://jupyter.radiasoft.org
  - many accelerator physics codes are pre-installed
  - made available via the `radiasoft/beamsim-jupyter` container
  - supports Jupyter/IPython notebooks
  - also supports browser-based terminal window (bash, without X11)

Wrap up

- Some aspects of computational reproducibility have been discussed
  - you have not been provided with a recipe to follow
  - it's not yet clear how to meet the near-term goals of slide #13 above
  - we haven't yet introduced all the necessary tools
  - reproducibility will be a recurring theme in upcoming lectures
  - by the end of the class, you should have your own ideas on how to achieve it

- Any final questions regarding the material in this lecture?

- Computer Lab this afternoon –
  - you will run the elegant code from Argonne National Lab (M. Borland)
    - **using a public JupyterHub server,** https://jupyter.radiasoft.org
    - **running from the command line**