Portable Workflows

CMSE 890-602

What is portable?

- Being able to run on (*m)any* operating systems
- Able to run on *(m)any* hardware setups

- Why not *every* OS and hardware setup?
  - Performance constraints
  - Dependencies!

Why do we care?

- Portable workflows are
    - Easier to reproduce
    - Easier to use widely
    - Easier to scale up

Portability methods

- Virtual machine (cloud computing)
- OS-agnostic language (Python, Java etc)
- Environments (conda)
- Containers (Singularity, Docker)
- Workflow manager (snakemake, nextflow)
- Can be applied to a part or the whole workflow

Hardware

- Two major CPU types: x64 (common) and ARM64 (newer Macs)
- Software must be compiled for each “architecture” of CPU
- Mostly just an issue when writing software on newer Macs
    - But there are subtle differences between CPUs of the same overall type!
- Other hardware (memory, storage) are typically interoperable
- Software can be *optimized* for specific hardware
    - But this is not portable!

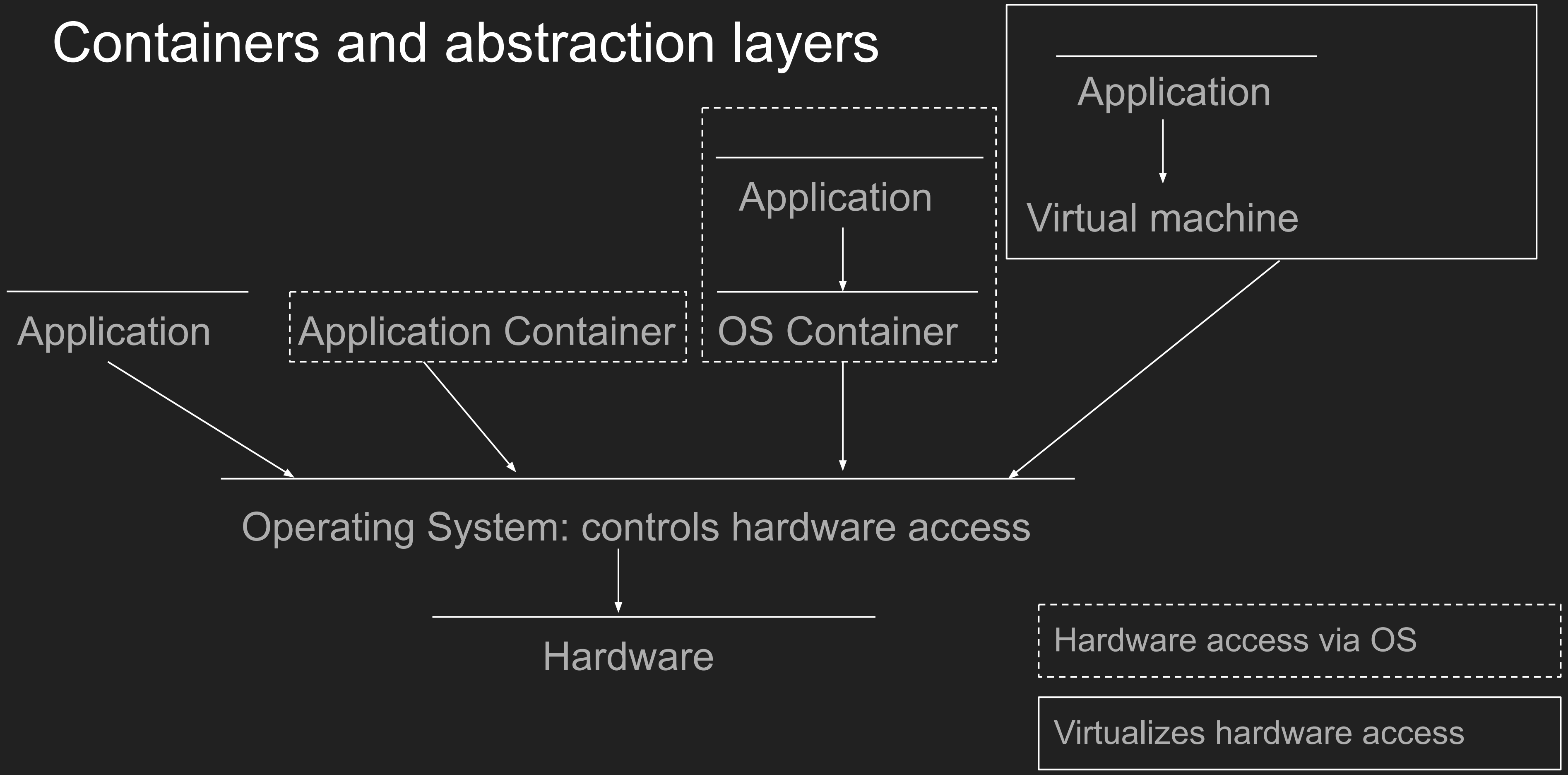
Operating Systems

- Controls hardware access
- Manages process start/stop
- Manages the file system
- 3 major types: Windows, Mac and Linux
- Linux is the typical operating system for data science and HPCC
- Therefore for science workflows using HPCC, Linux support is critical!

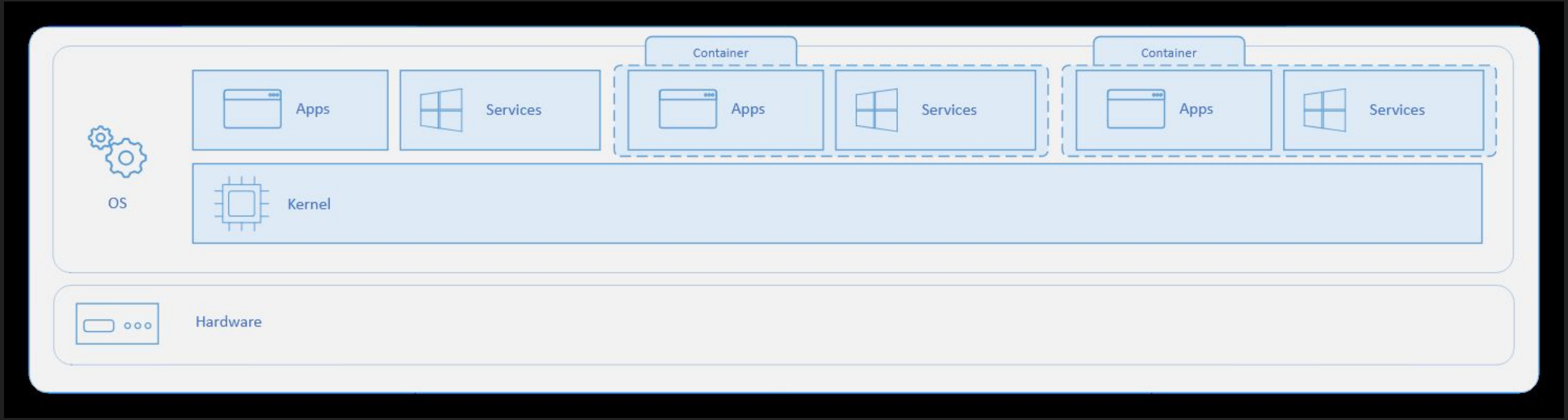
Containers

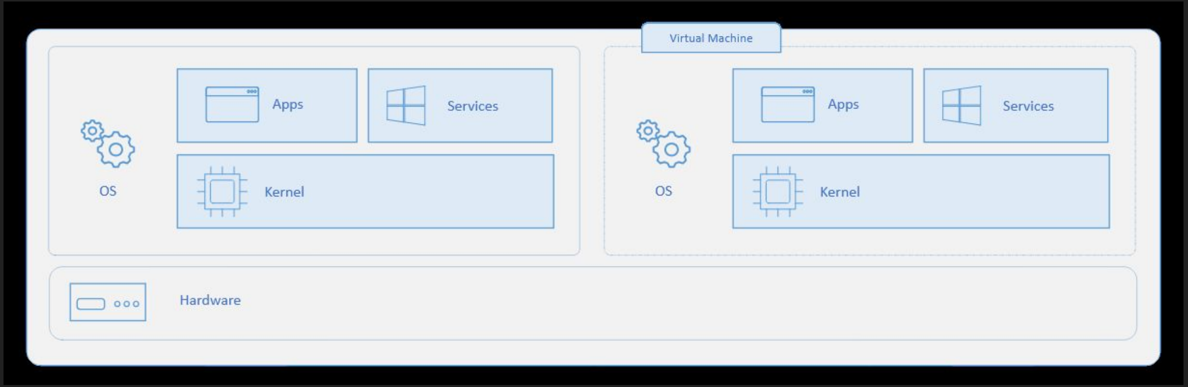
- Containers act as a layer between software and operating system
- The underlying OS determines what hardware is available to the container
- Containers can only access the hardware that is present on the system
- The container can include:
    - Operating system
    - Application software
    - Data

Containers and abstraction layers



Containers vs Virtual Machines





Containers vs Virtual Machines

Containers

- Weak security boundary
- Minimal operating system
- Uses few system resources
- Runs on the host operating system
- Requires a rebuild to update

Virtual machine

- Strong security boundary
- Full operating system
- Uses many system resources
- Run any kind of operating system within
- More complex update process

Adapted from https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm#containers-vs-virtual-machines-1

Why containerize?

- Full control over the environment and specific OS version
  - But does need to match the underlying OS of the system you are running on
- Include the exact software versions required for your workflow
- Guarantee reproducibility (or at least 99% guarantee)
- Increase portability
- High performance
- Share easily via the description file or sites like Docker Hub
- Supported by all major workflow managers

Why Virtual Machine?

- Full control including system resources and complete operating system
- Guarantee reproducibility
- Portable (if system supports VM)
- Maximum security
- Supported by cloud computing

Singularity

- Had an awkward split into SingularityCE and Apptainer
- Both basically the same and currently interoperable
- DOES NOT PROVIDE ROOT/ADMIN ACCESS
- Linux only (but compatible across distributions)
- Provides access to default system paths

Singularity commands

- `shell` - launch a shell within the container
- `exec` - run a command in the container
- `run` - start the container’s default script
- `pull` - download an image
  - Singularity or Docker
- `build` - create your own container (needs ROOT PERMISSIONS)

Docker

- Compatible with all major operating systems
- But the *docker containers* are not portable if they are based on a different OS
- Must manually specify directories external to the container for access
- Many, *many* Docker containers available for use
  - Docker Hub https://hub.docker.com/
- Has a slightly confusing desktop client

Docker commands

- `-t <container> sh` - launch a shell within the container
- `run <container> <command>` - run a command in the container
- `run` - start the container’s default script
- `pull` - download an image
  - Docker only
- `build` - create your own container (needs ROOT PERMISSIONS)

In-class assignment

- Please read through https://docs.icer.msu.edu/Singularity_Introduction/
- Using ICER, follow the steps from "Check Installation" onwards. You may want to consider installing Singularity on your own machine if you want to build your own images.
- Take a screenshot of the terminal output you get for each command and post on D2L.