Break Free from Dependency Hell with Docker!

Containerizing Your Apps for Less Headaches.

James Hurd

jameshurd@ku.edu

Roadmap

- Motivation, or “Why should I care?!”
- Theory
- Containers in practice

ACT I: *Motivation*

Or, “Why should I care?!”

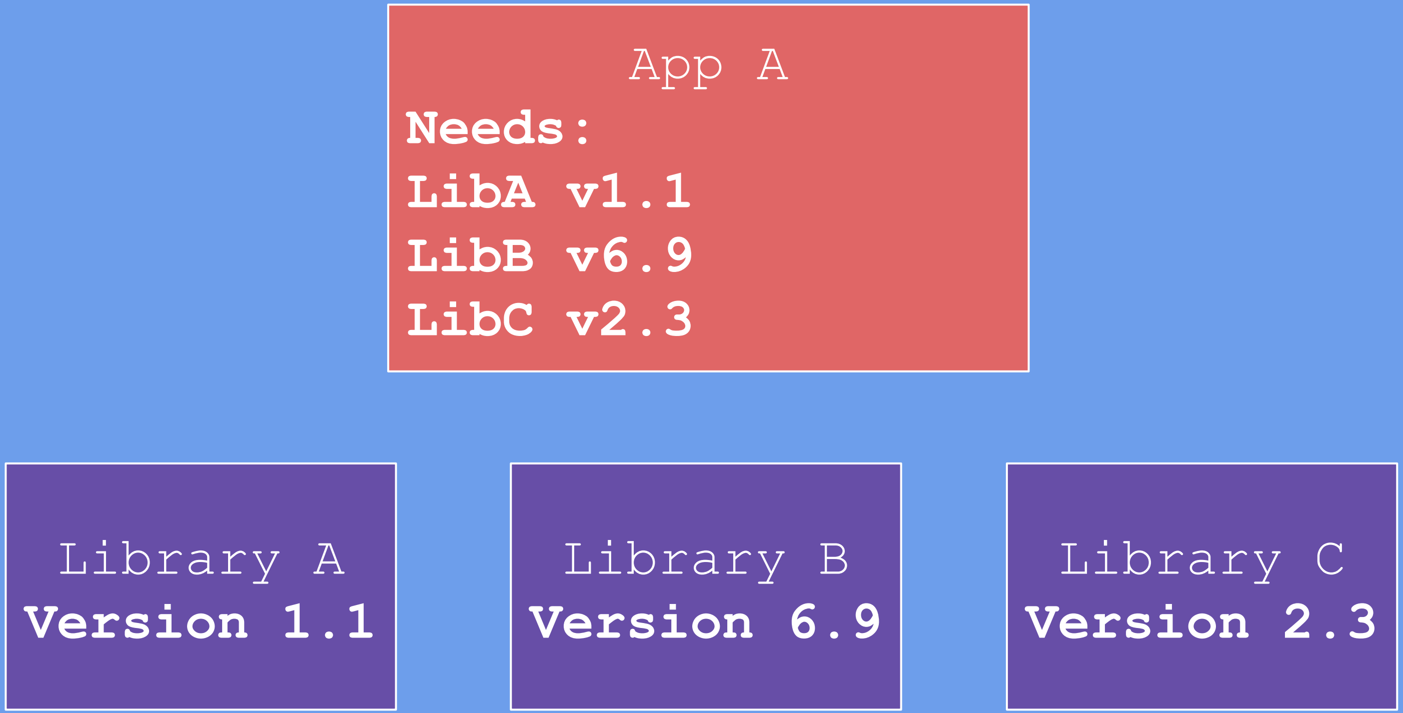
Has this ever been you?

*Code works fine on local machine.

The code in the the dev server

well now I am not doing it
imgflip.com
ProgrammerHumor.io

Or maybe…



App A

Needs:

LibA v1.1

LibB v6.9

LibC v2.3

Library A

Version 1.1

Library B

Version 6.9

Library C

Version 2.3

Perfect, all dependencies met. . .

App A

Needs:

LibA v1.1

LibB v6.9

LibC v2.3

App B

Needs:

LibA v1.2

LibB v6.9

LibC v2.3

Library A

Version 1.1

Library B

Version 6.9

Library C

Version 2.3

Noooo!!!!!!!!!!!!!!!!!!! Can’t install App B!

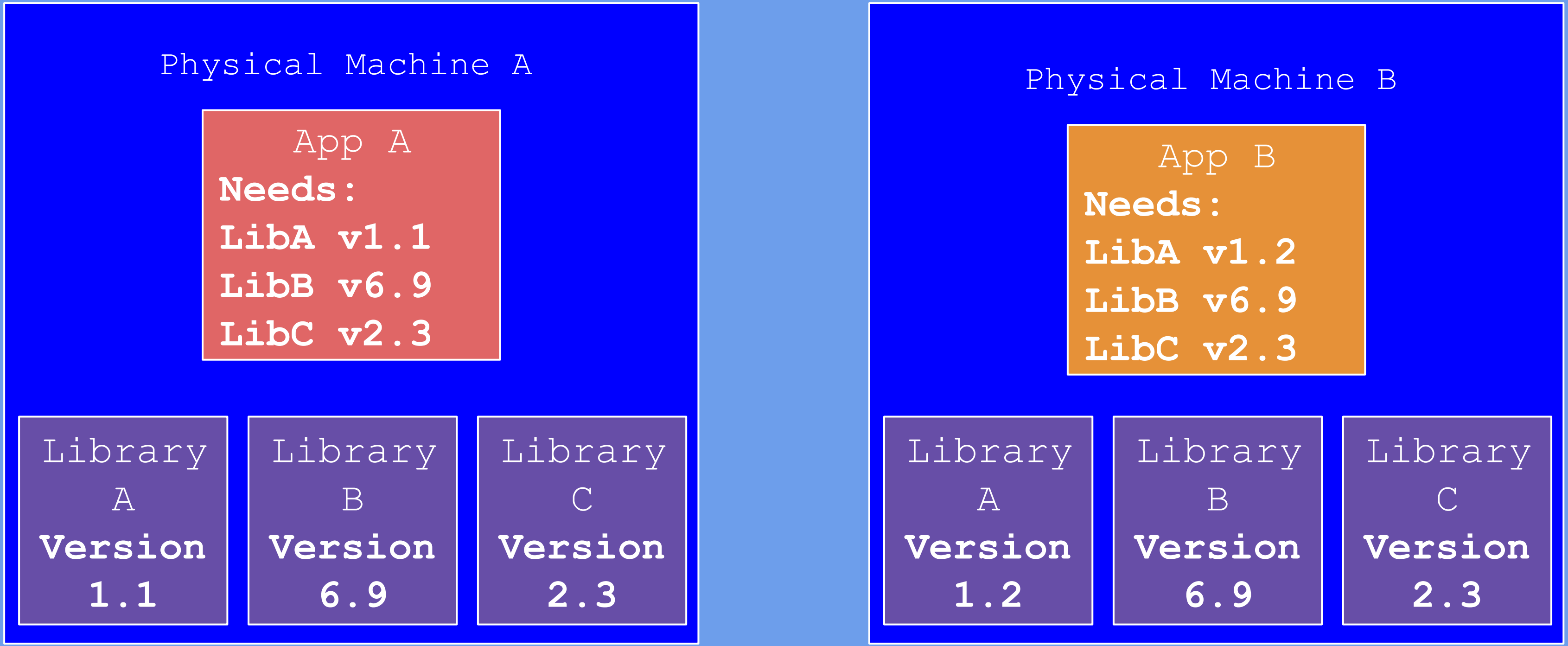
App A and App B are **incompatible**.

Welcome to dependency hell.

How do we get out?

ACT II: *Theory*

Option 1: Use Different Physical Machines



Physical Machine A

App A

Needs:

LibA v1.1

LibB v6.9

LibC v2.3

Library A **Version 1.1**

Library B **Version 6.9**

Library C **Version 2.3**

Physical Machine B

App B

Needs:

LibA v1.2

LibB v6.9

LibC v2.3

Library A **Version 1.2**

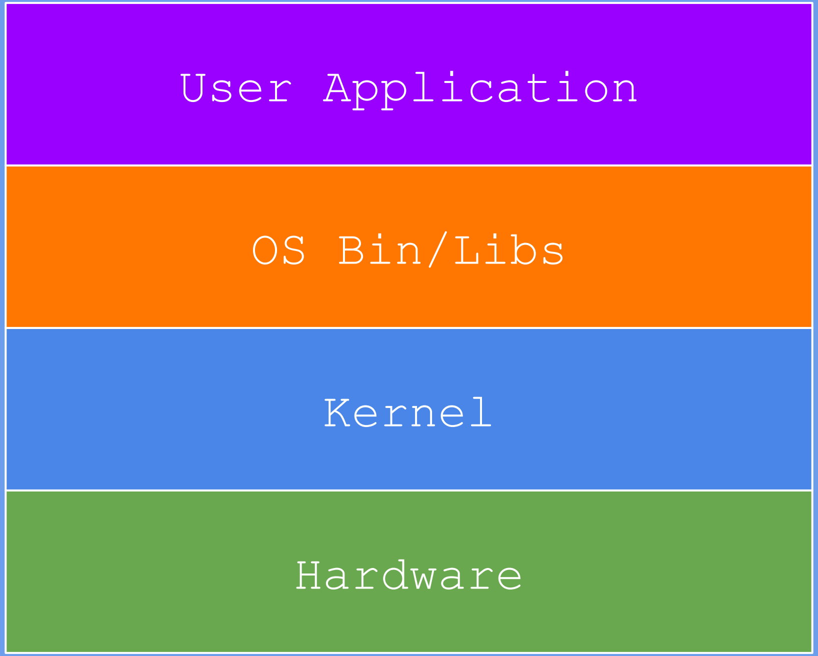
Library B **Version 6.9**

Library C **Version 2.3**

Option 1: Use Different Physical Machines

Sure, this technically works but...

- Not scalable.
- Too expensive.
- Takes up too much space.
- Uses a lot of energy.
- Just generally overkill.

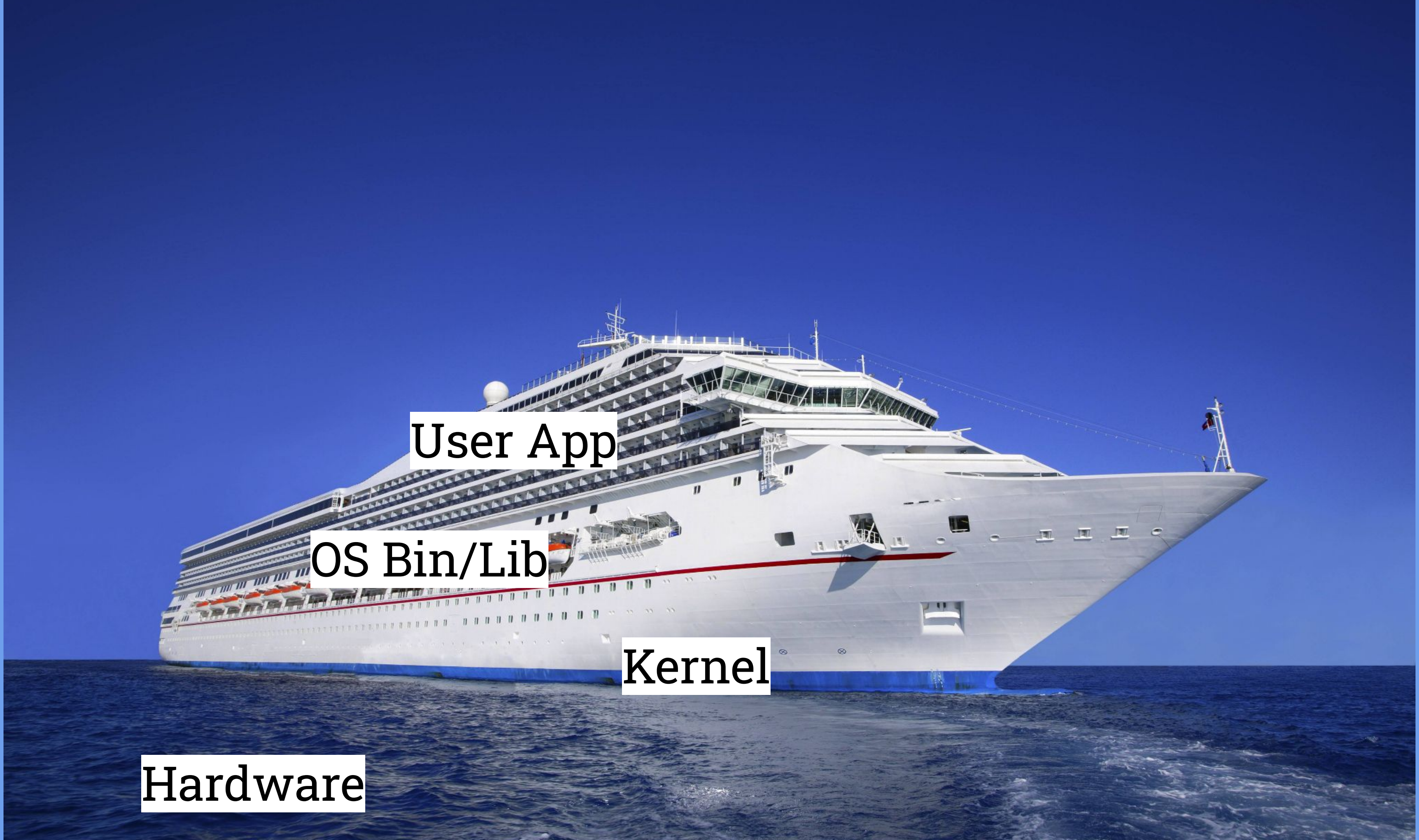


User Application

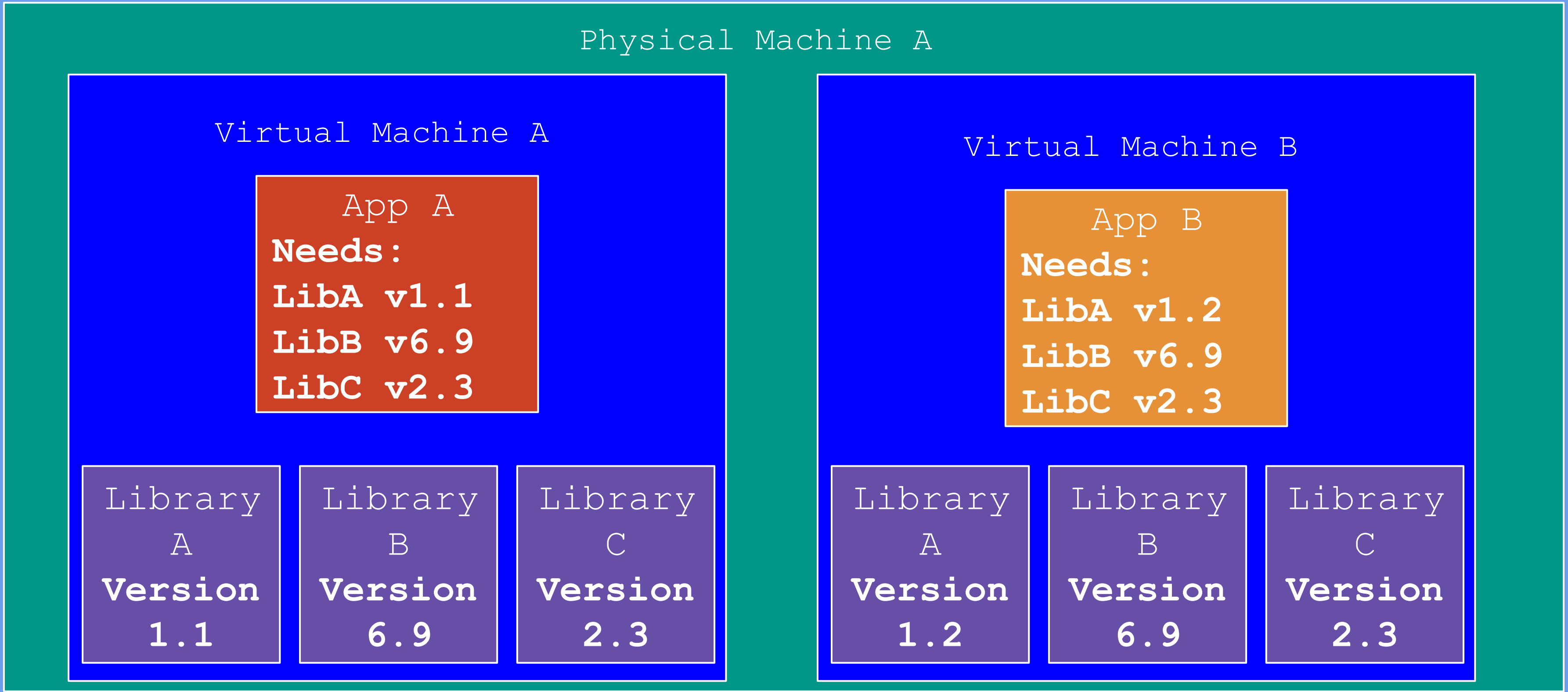
OS Bin/Libs

Kernel

Hardware


User App
OS Bin/Lib
Kernel
Hardware

Option 2: Use Virtual Machines



Physical Machine A

Virtual Machine A

App A

Needs:

LibA v1.1

LibB v6.9

LibC v2.3

Library A **Version 1.1**

Library B **Version 6.9**

Library C **Version 2.3**

Virtual Machine B

App B

Needs:

LibA v1.2

LibB v6.9

LibC v2.3

Library A **Version 1.2**

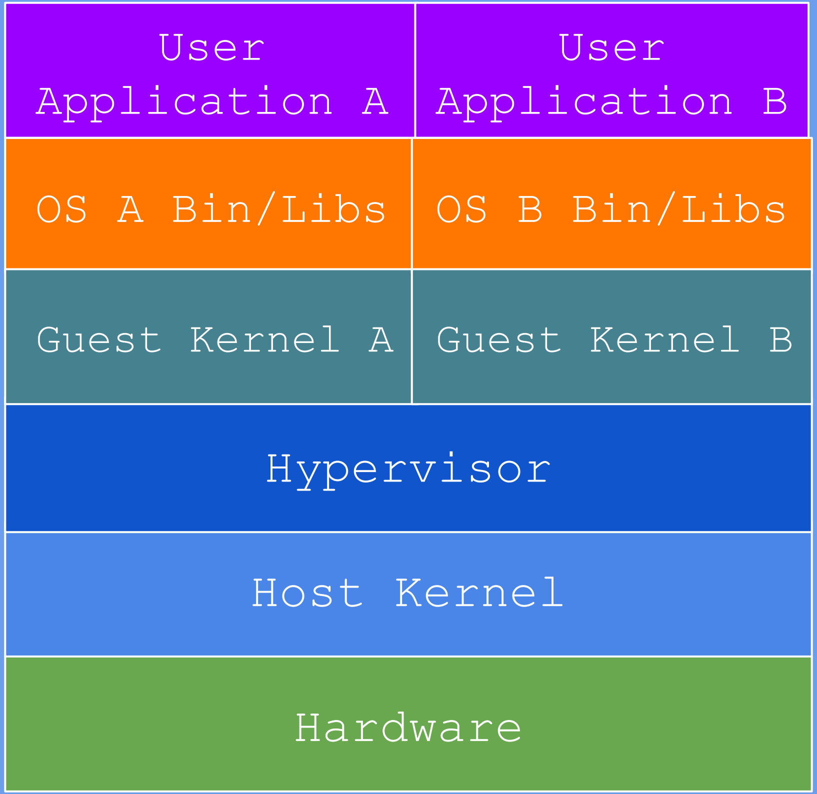
Library B **Version 6.9**

Library C **Version 2.3**

Virtualized System Environment

Better! However...

- Still not super scalable.
- Large overhead in compute-heavy apps.
    - e.g. Things using GPUs.
- Can be annoying to manage.
- Portability across hypervisors can be a problem.
- Resource intensive.



User Application A

User Application B

OS A Bin/Libs

OS B Bin/Libs

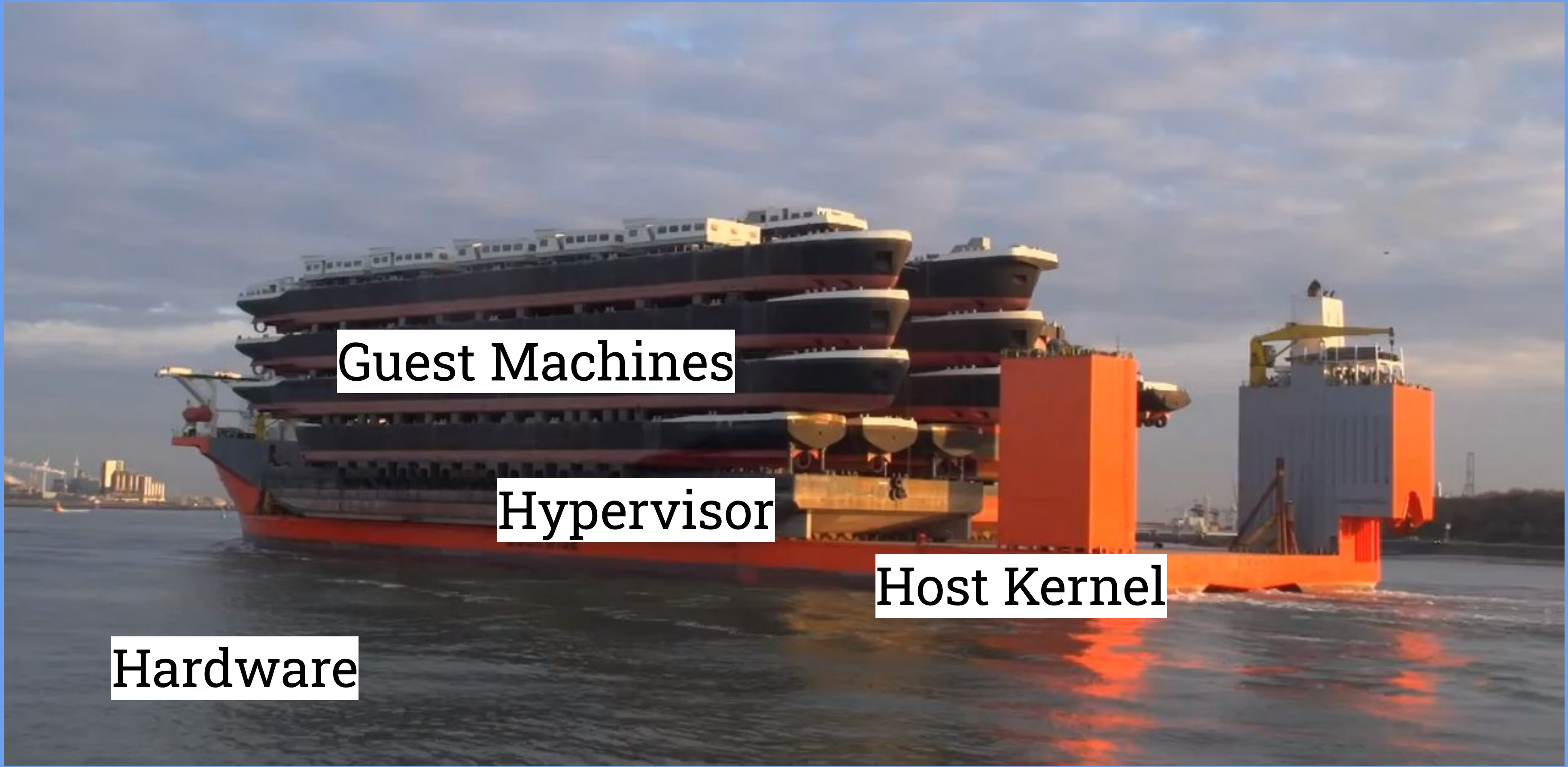
Guest Kernel A

Guest Kernel B

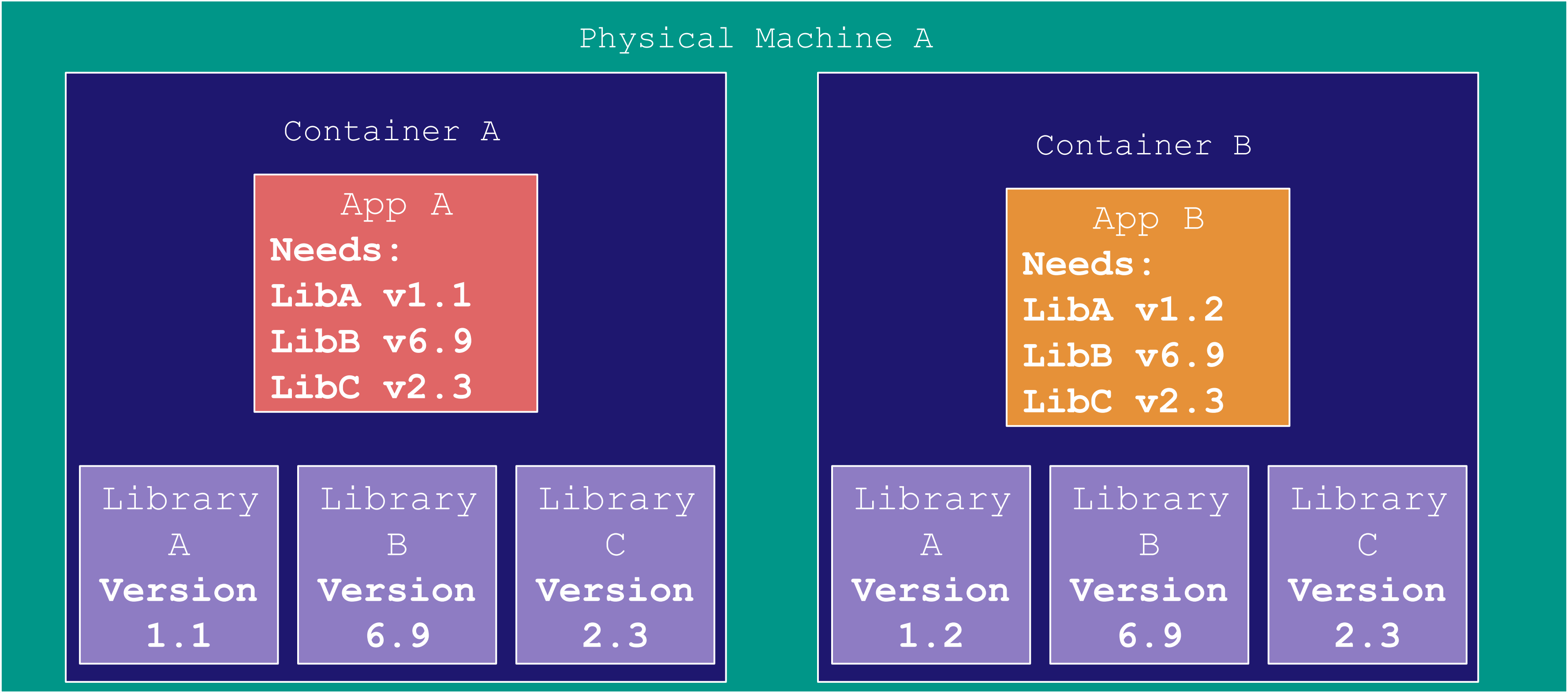
Hypervisor

Host Kernel

Hardware


Guest Machines
Hypervisor
Host Kernel
Hardware

Option 3: Use Containers



Physical Machine A

Container A

App A

Needs:

LibA v1.1

LibB v6.9

LibC v2.3

Library A **Version 1.1**

Library B **Version 6.9**

Library C **Version 2.3**

Container B

App B

Needs:

LibA v1.2

LibB v6.9

LibC v2.3

Library A **Version 1.2**

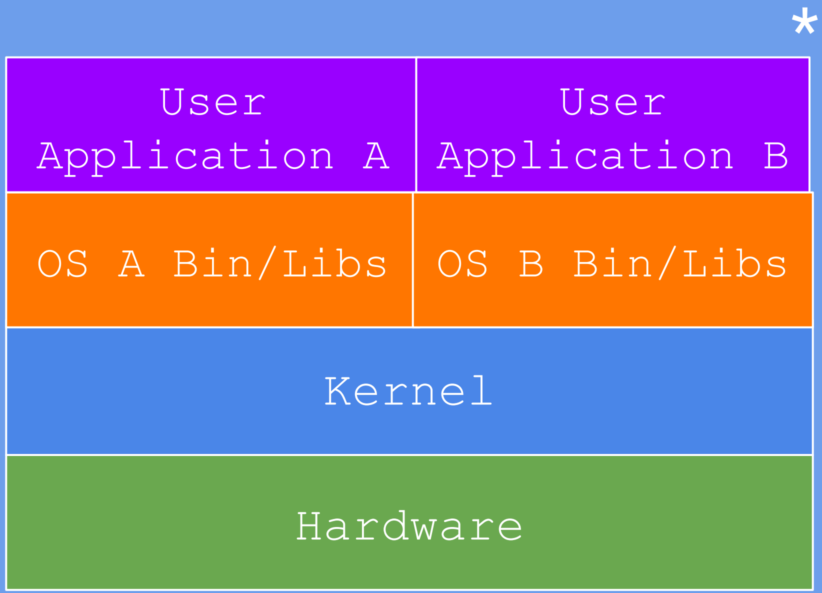
Library B **Version 6.9**

Library C **Version 2.3**

Containerized System Environment

Much better!

- Isolated environments.
- Negligible overhead.
- Lightweight.
- Scalable.
- Very portable!
  - All you need is a container runtime.



*

User Application A

User Application B

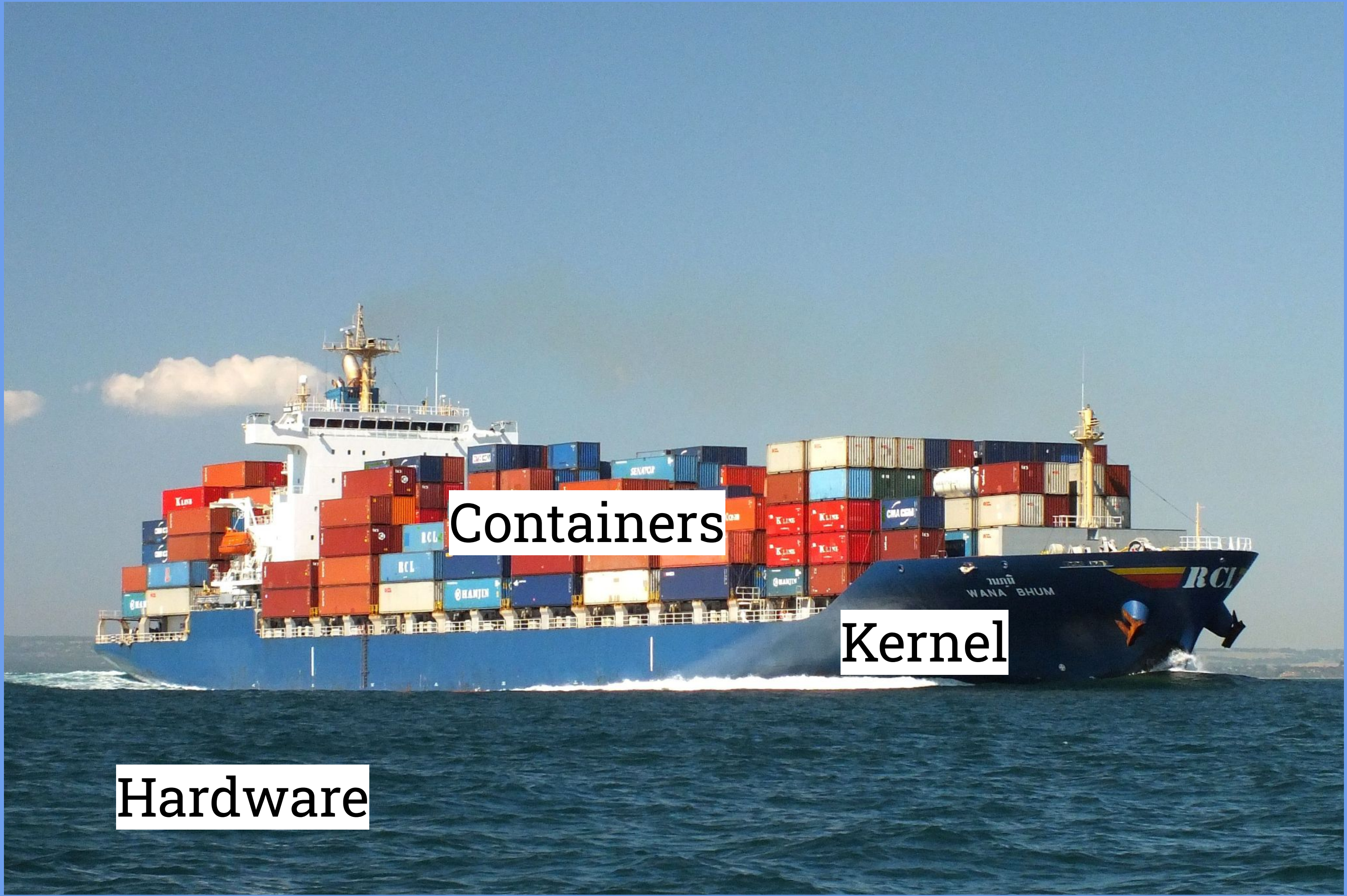
OS A Bin/Libs

OS B Bin/Libs

Kernel

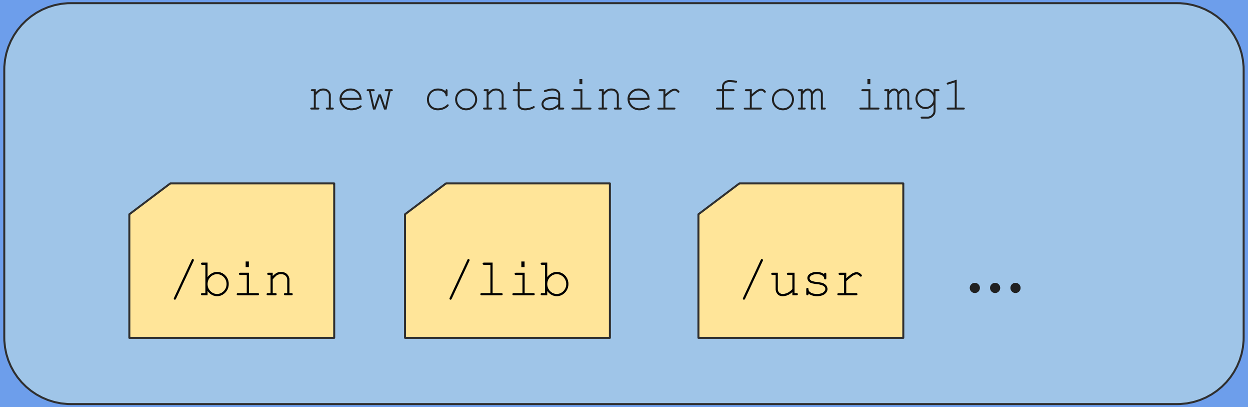
Hardware

*A bit of a lie in some cases since some container runtimes talk through a daemon to do certain things like software-defined networking.


Containers
Kernel
Hardware

A Look Inside the Container

- Assuming your image is something UNIX-y.
- Any processes that run in this container will see this as its root.
  - Essentially fancy `chroot`.
- Processes running in here cannot access/see anything outside.
  - Possible because of kernel magic with `namespaces` and `cgroups`.



And now, a demo

Containers are **NOT** virtual machines.

They are simply isolated processes.

Containers separate **application** from **infrastructure**.

IT WORKS ON MY MACHINE
THEN WE'LL SHIP YOUR MACHINE
AND THAT IS HOW DOCKER WAS BORN

That was a lot... 😵‍💫

Questions? Comments? Concerns?

ACT III: *Containers in Practice*

What is *Exactly* Docker?

- Docker is a container runtime.
- Docker daemon (`dockerd`) manages:
    - Storing Images.
    - Building Images.
    - Running Containers.
    - Networking (in some cases)
- Communicate with daemon through client (`docker`).
- Will need to download, link on `README`.

What is a Docker Image?

- Read-only
- Defines the **environment** and **application** to be run.
- Kinda-sorta like an ISO

How do I get an image?

Option 1: Pull from a Registry

- dockerhub – registry of of pre-created images.
  - Images made by users and official sources.
  - `hub.docker.com`
- `docker pull img`
  - Gets `img` from dockerhub (defaults to latest release)
  - Specify release `tag` with `docker pull img:tag`
- `docker images`
  - See images you have.
- `docker image prune`
  - Remove old/unused images.

Let’s pull some images!!!

Option 2: Bring your own image (BYOI*)

- `Dockerfile`
  - Recipe for creating an image.
  - Goes in the root of the app you want to containerize.
- `docker build -t name /path/to/app/root`
  - Image identified by tag `name:latest`
- `docker tag old new`
  - Retag image with tag `old` to tag `new`.

*Not a technical term... I just made it up 🤪

Essential `Dockerfile` Ingredients

- `# syntax=docker/dockerfile:1`
    - Goes at top of file.
- `FROM img`
    - Specify base image.
- `WORKDIR dir`
    - basically `cd`
- `COPY src dest`
    - Copy files on host at `src` to `dest` in image.
- `RUN command` – Run `command` inside the image.
    - **CANNOT** take user input!
- `CMD [“cmd”, “arg1”, “arg2”, …]`
    - Run `cmd` with `arg1`, `arg2`, … used to launch app.
- `EXPOSE port` – Expose TCP port number `port` to the host.

Building Your Own Docker Image

- `Dockerfile`
  - Contains instructions on how to make an image.
  - Goes in the root of the app you want to containerize.
  - Top line: `# syntax=docker/dockerfile:1`
- `docker build -t name .`
  - To build app from `Dockerfile` in same directory.
  - Image identified by tag `name:latest`
  - `-f path` pass alternate path to Dockerfile
- `docker tag old new`
  - Retag image with tag `old` to tag `new`.

Let’s make one!

✨Sharing your images with the world✨

- Create an account on dockerhub and create a new repository.
- `docker login -u your_username` – Log into dockerhub from client.
- Make sure the tag on your local image == name of the repository.
- `docker push tag`

Let’s share ACM with the world!

Still with me?

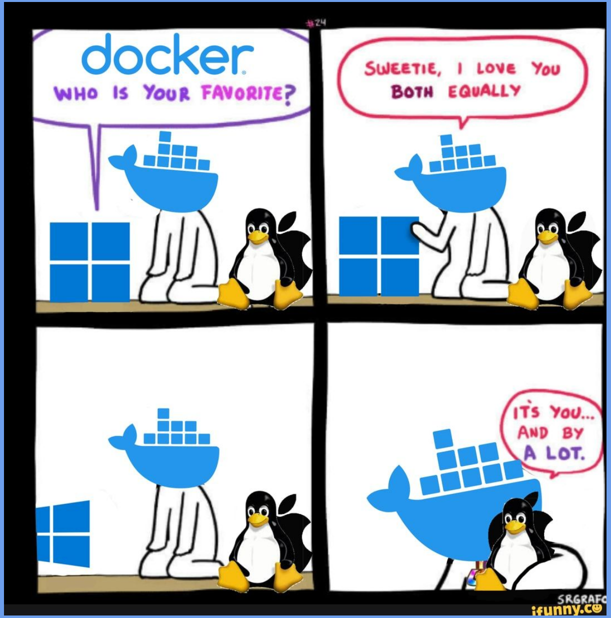
Questions?

These images are cool and all... how do I actually use them?

Notes on Linux and Windows Compatibility

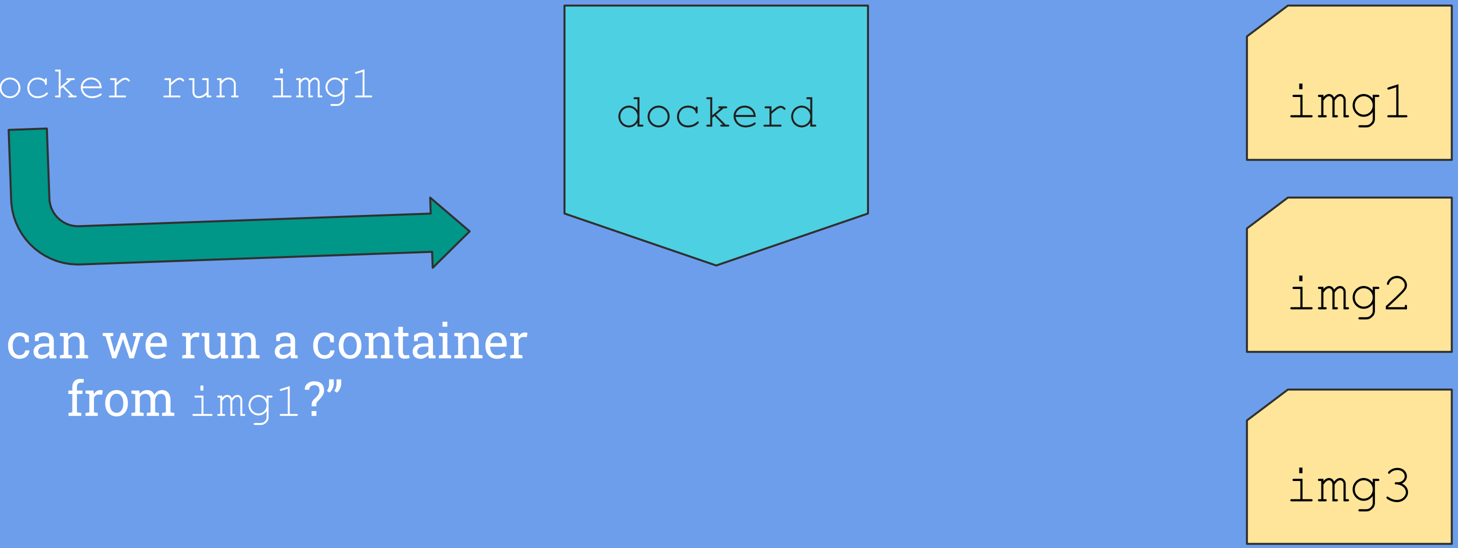
Recall that containers share the host kernel. Consequences for Docker:

- Windows images will not run on Linux
- Docker on Windows spawns a Linux VM in the background to run Linux images.



What Is Docker doing when I run a container?

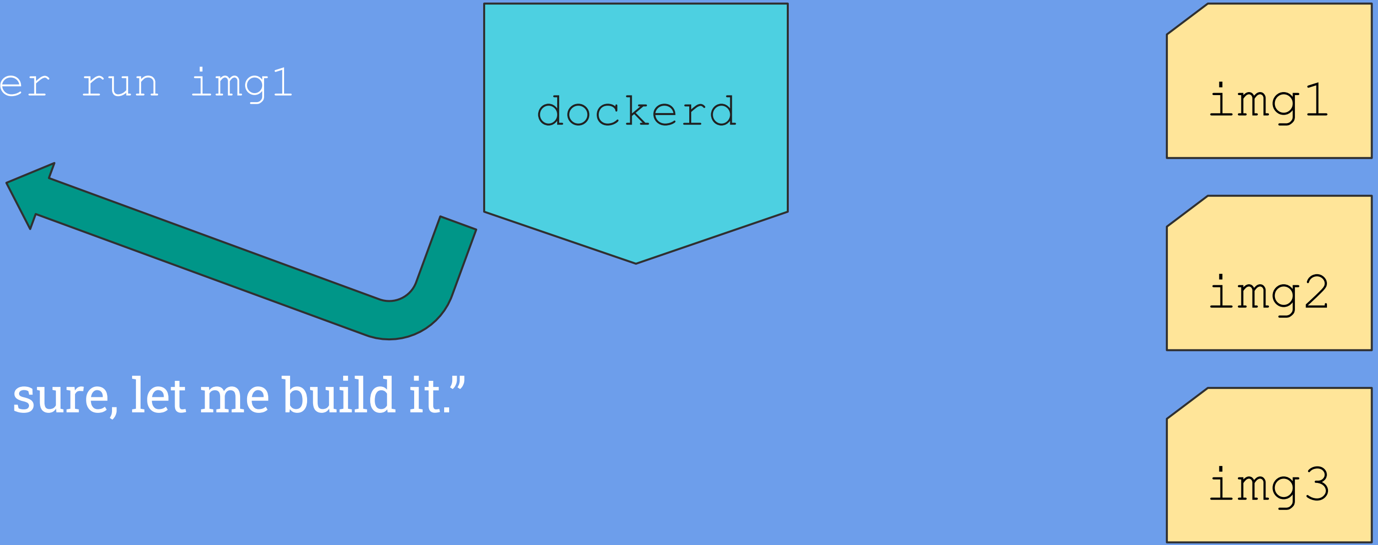
```
$ docker run img1
```



“Hey, can we run a container from `img1`?”

What Is Docker doing when I run a container?

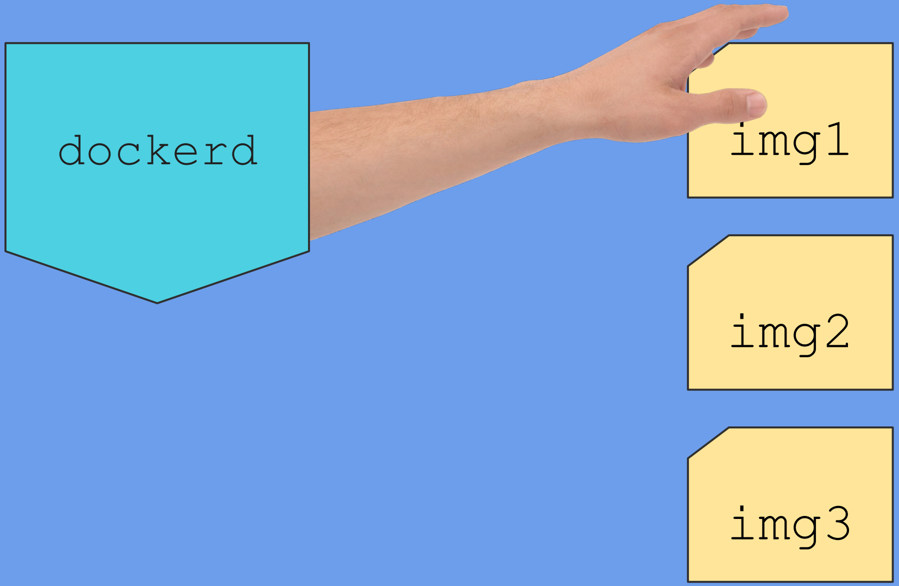
```
$ docker run img1
```



“Yeah sure, let me build it.”

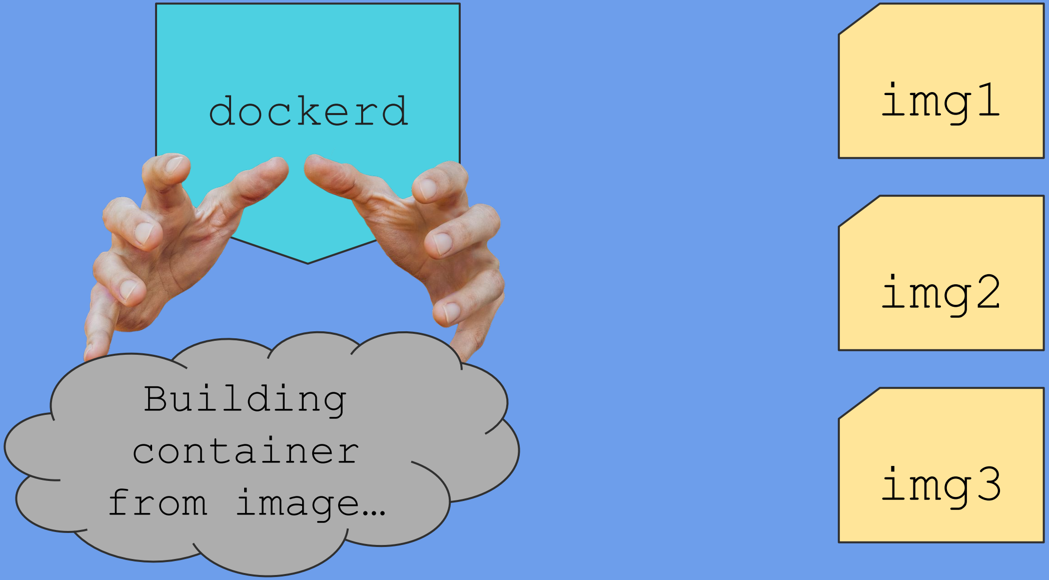
What Is Docker doing when I run a container?

```
$ docker run img1
```



What Is Docker doing when I run a container?

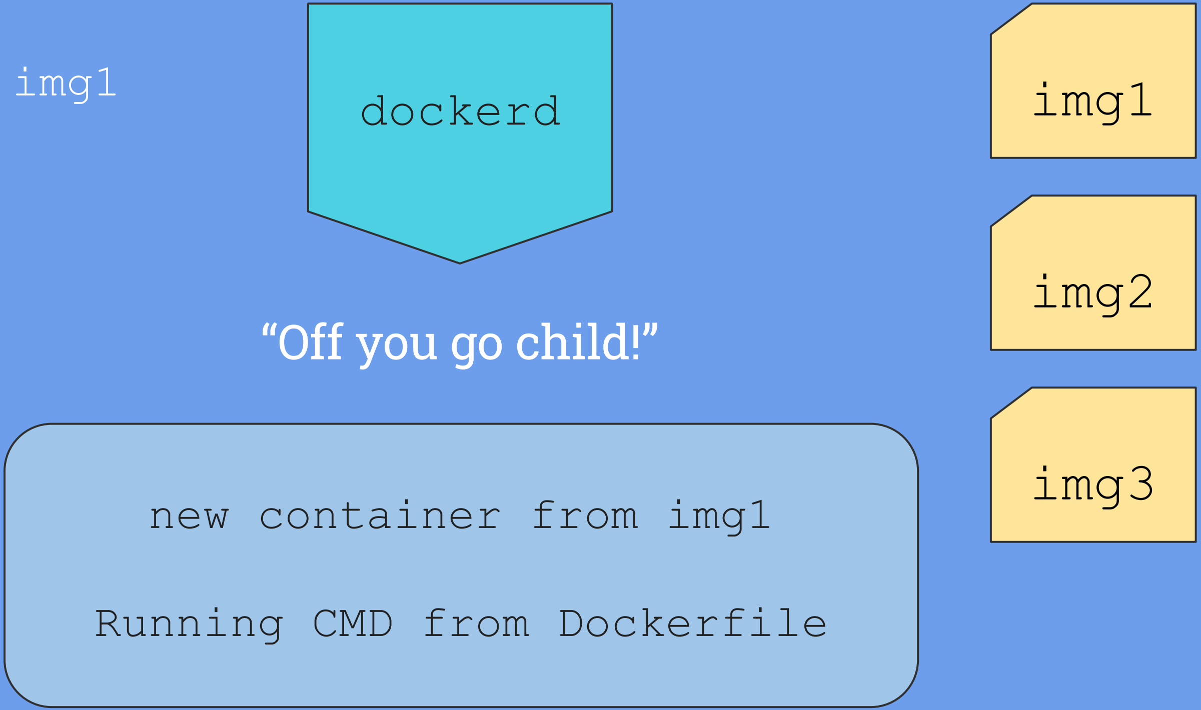
```
$ docker run img1
```



What Is Docker doing when I run a container?

```
$ docker run img1
```

Important takeaway:
Docker builds a **new** container from image **each** run.



"Off you go child!"

new container from img1

Running CMD from Dockerfile

Running Your Container

- `docker run img_name` – Create a container from `img` and start it.
    - `-d` – Detach (i.e. run in background) .
        - Outputs container ID.
    - `-p h:c` – Maps port `c` in container to port `h` on host.
    - `-—name name` – Human readable name to identify container.
- `docker ps` – Show running containers.
    - `-a` – see stopped containers too.
- `docker start id` – Start a stopped container with id `id`.
    - Will start detached.
- `docker exec id command` – Run `command` in container with id `id`.
    - Useful for checking on things/inspecting container.

Stopping and Updating Your Container

- `docker stop id` – Stop container with id `id`.
  - `id` can be extracted from `docker ps` if unknown.
- `docker restart id` – restart container with id `id`.
- `docker rm id` – Remove container with id `id`.
  - Deleting a container will delete all of its data!
- To update your app, simply rebuild and run the new image!
  - Make sure to stop and delete old container.

Time to start our container!

Problems With Sharing/Persistent Data

- All data is deleted when a container is deleted.
  - Containerizing databases :(
- One container cannot read any data from another.
- This is all by design.

Let’s see the problem in action.

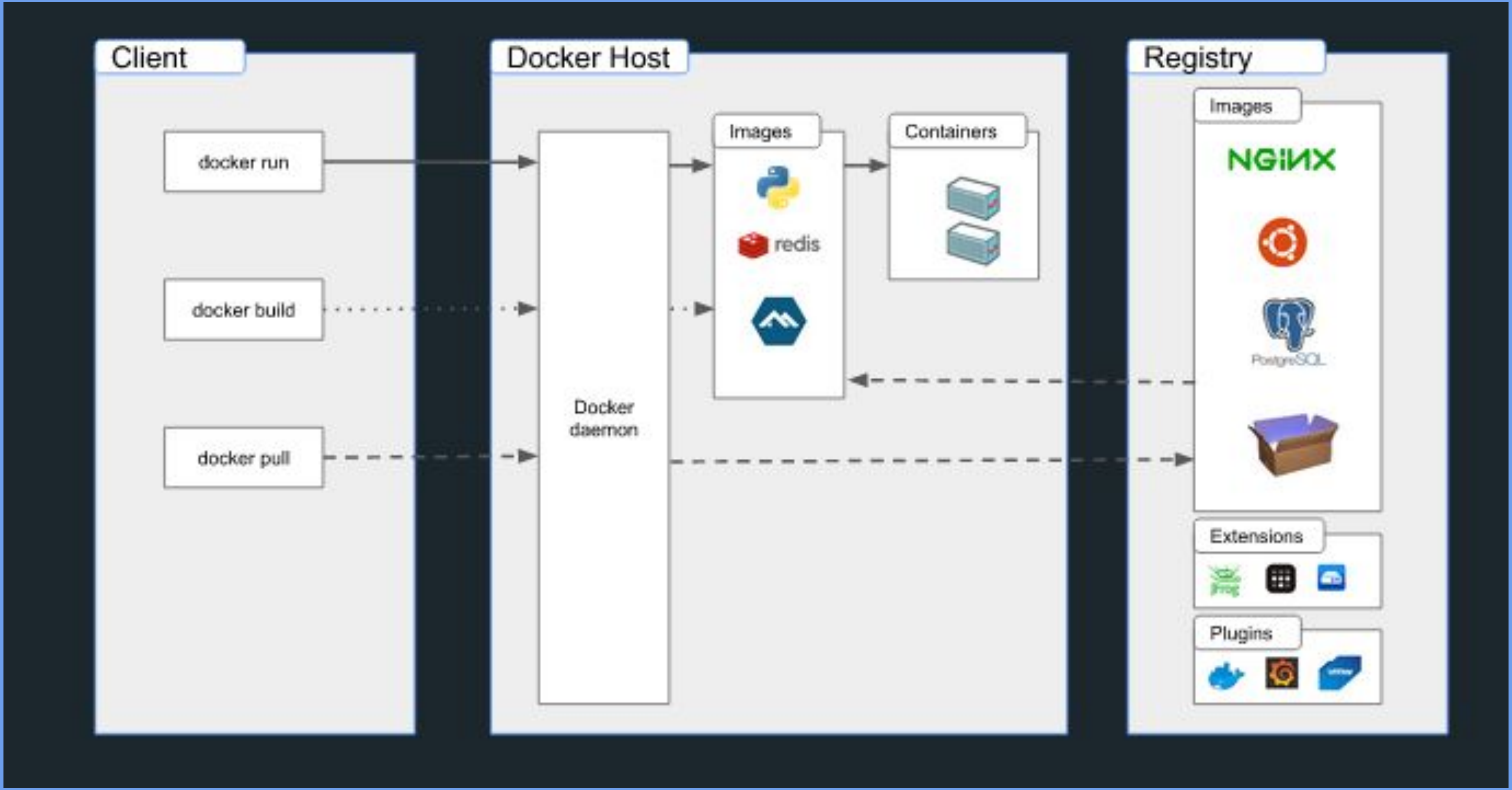
The Solution: Container Volume Mounts!

- Volume mounts connect filesystem paths in the container and host.
- `docker volume create name` – Create a volume called `name`.
- Pass `--mount type=volume,src=name,target=path` to `docker run.`
  - `src` – name of volume to mount.
  - `path` – absolute path to directory from container to include in volume.
- `docker volume inspect` – View details about volume.
- `docker volume rm name` – Delete volume `name`.
  - All containers using `name` must first be deleted!

We can fix him.

checkin.onlynands.org

In conclusion…



Client

docker run

docker build

docker pull

Docker Host

Docker daemon

Images

Containers

Registry

Images

Extensions

Plugins

Now you (hopefully) know the basics of Docker!

`docker.onlynands.org`

Thank you!

jmh@ku.edu

fin.