What you don't know about containers (and why it matters)

Sean Scott

Oracle ACE Director

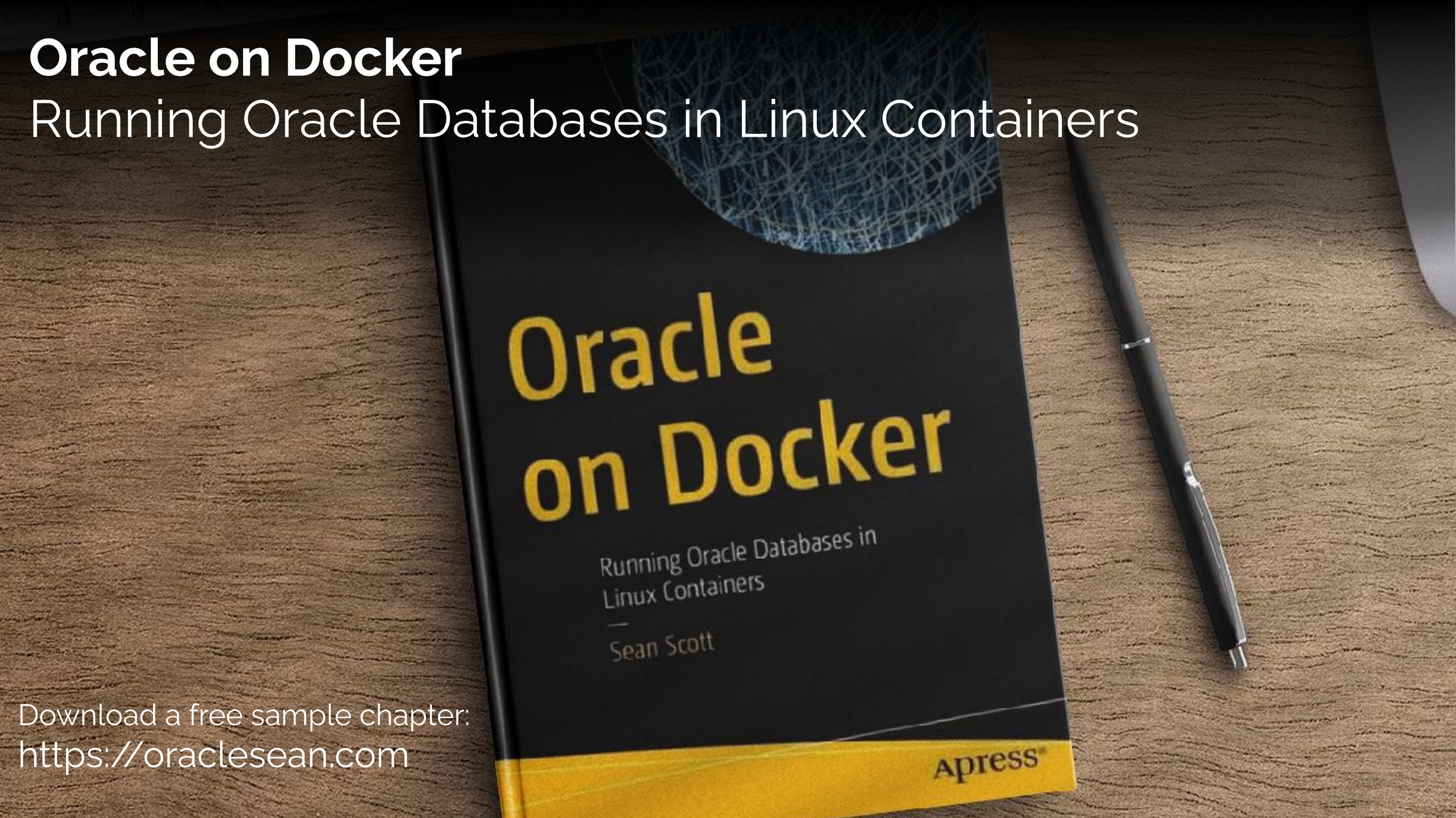
Data on Kubernetes Community Ambassador

Managing Principal Consultant

{ON: The Beach}

Oracle on Docker

Running Oracle Databases in Linux Containers



Download a free sample chapter:
https://oraclesean.com

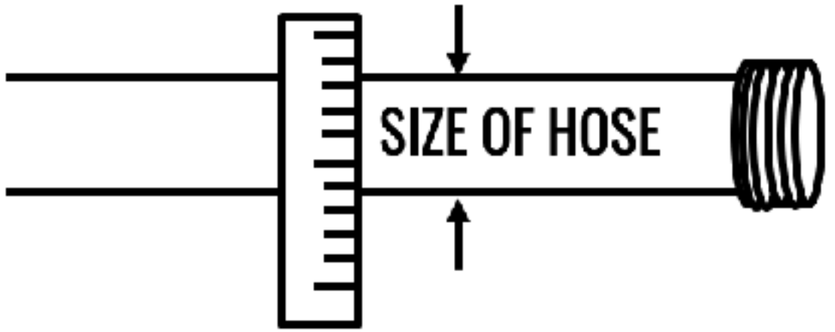
What Are Containers?

Containers are like lightweight Virtual Machines, right?

Electricity is like a water hose

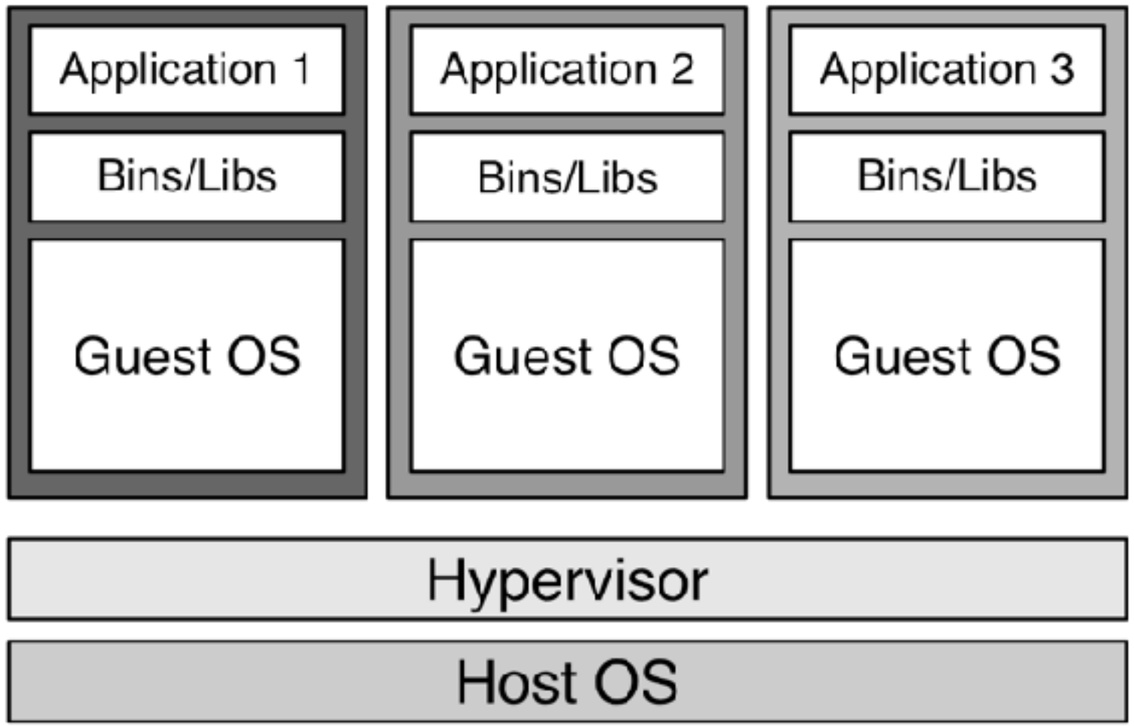
Voltage — Volts (V)

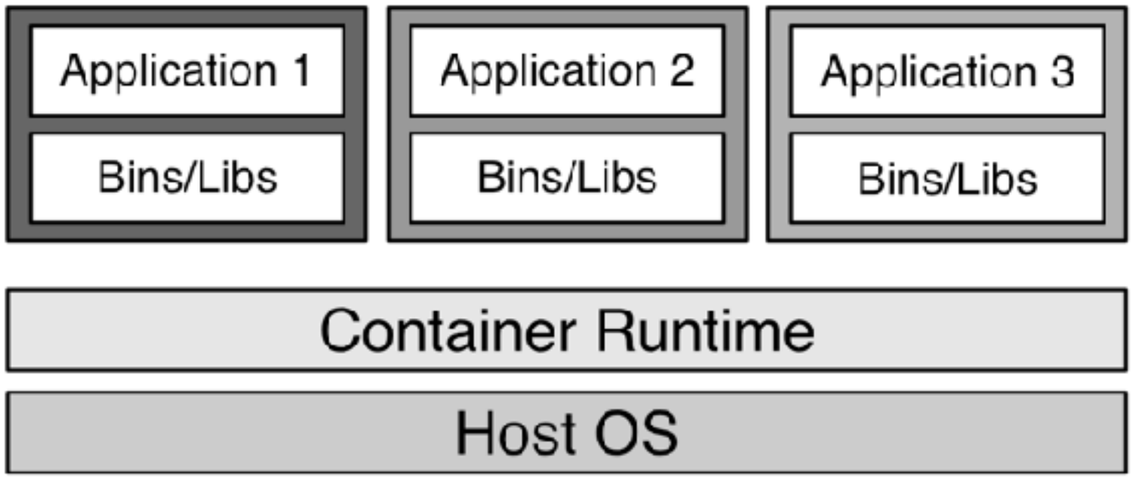
Current — Amps (A or I)



Resistance — Ohms (R or Ω)

FREEING ENERGY





Three full operating systems

Three application/executable directories

Virtual Machines: Bootable OS, often heavy

Containers: Support one application or service

Virtual Machines: Bootable OS, often heavy

A 100-page book of games & puzzles

Containers: Support one application or service

Virtual Machines: Bootable OS, often heavy

A 100-page book of games & puzzles

Containers: Support one application or service

A sheet of paper with a Tic-Tac-Toe grid

Games have structured playing surfaces and rules.

Services deliver games to players.

Images include the
game rules and playing surface.

Images are a *minimal filesystem & metadata*

for running a service.

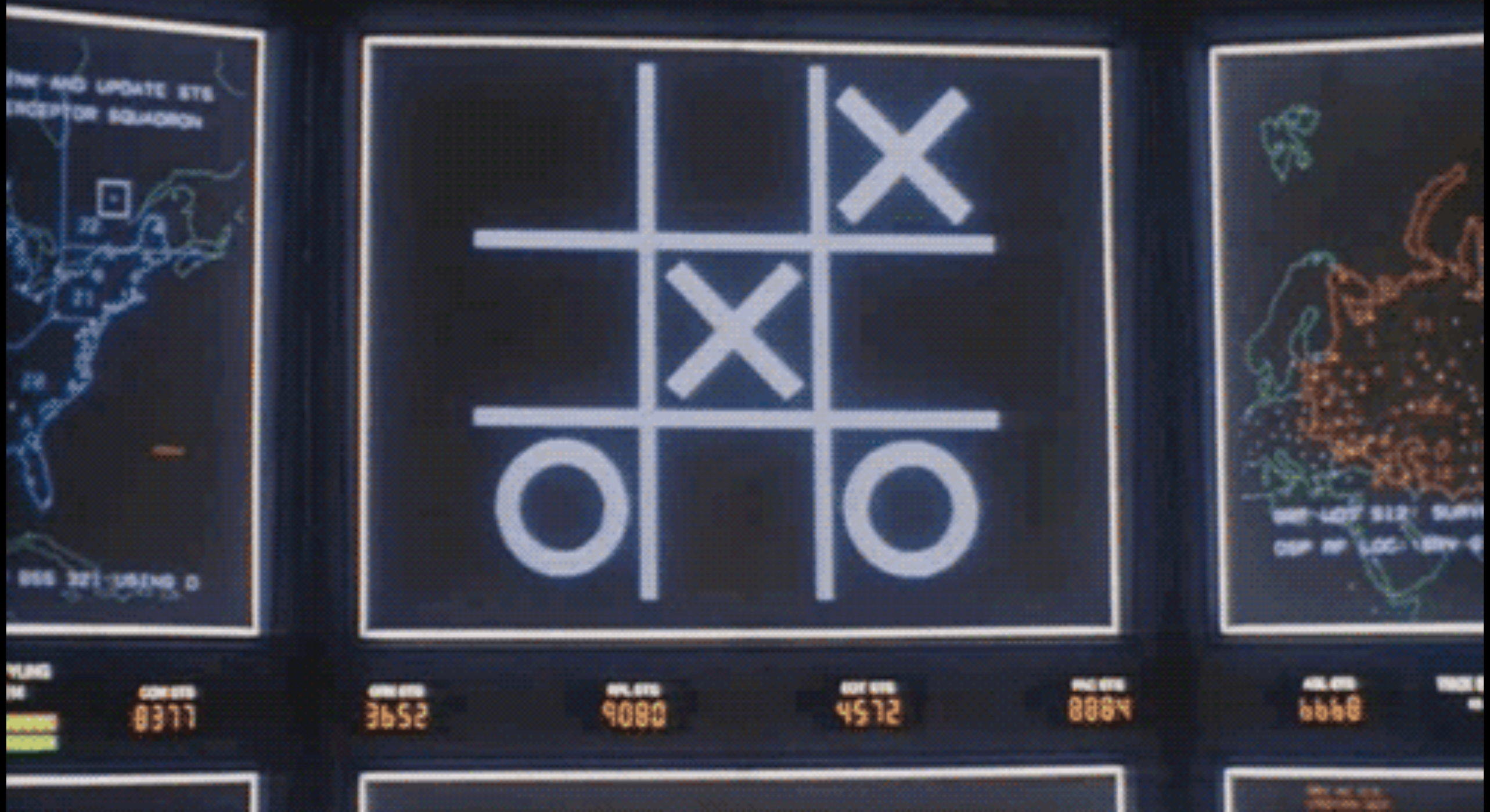
Running an image starts a **Container**.

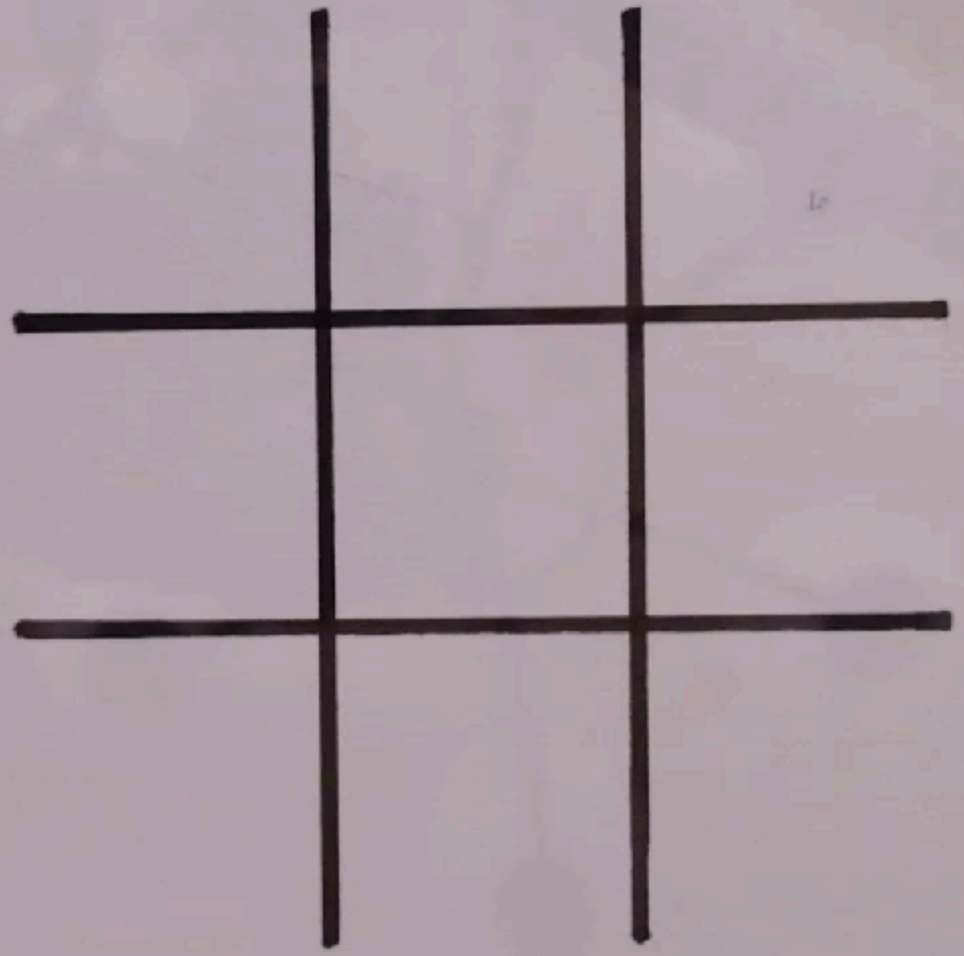
A container is:

an **isolated host process** and

a **Union Filesystem**.


SHALL WE PLAY A GAME?





A **Union Filesystem** has three layers:

A *Merge* or *Union* Layer.

An *Upper* Layer.

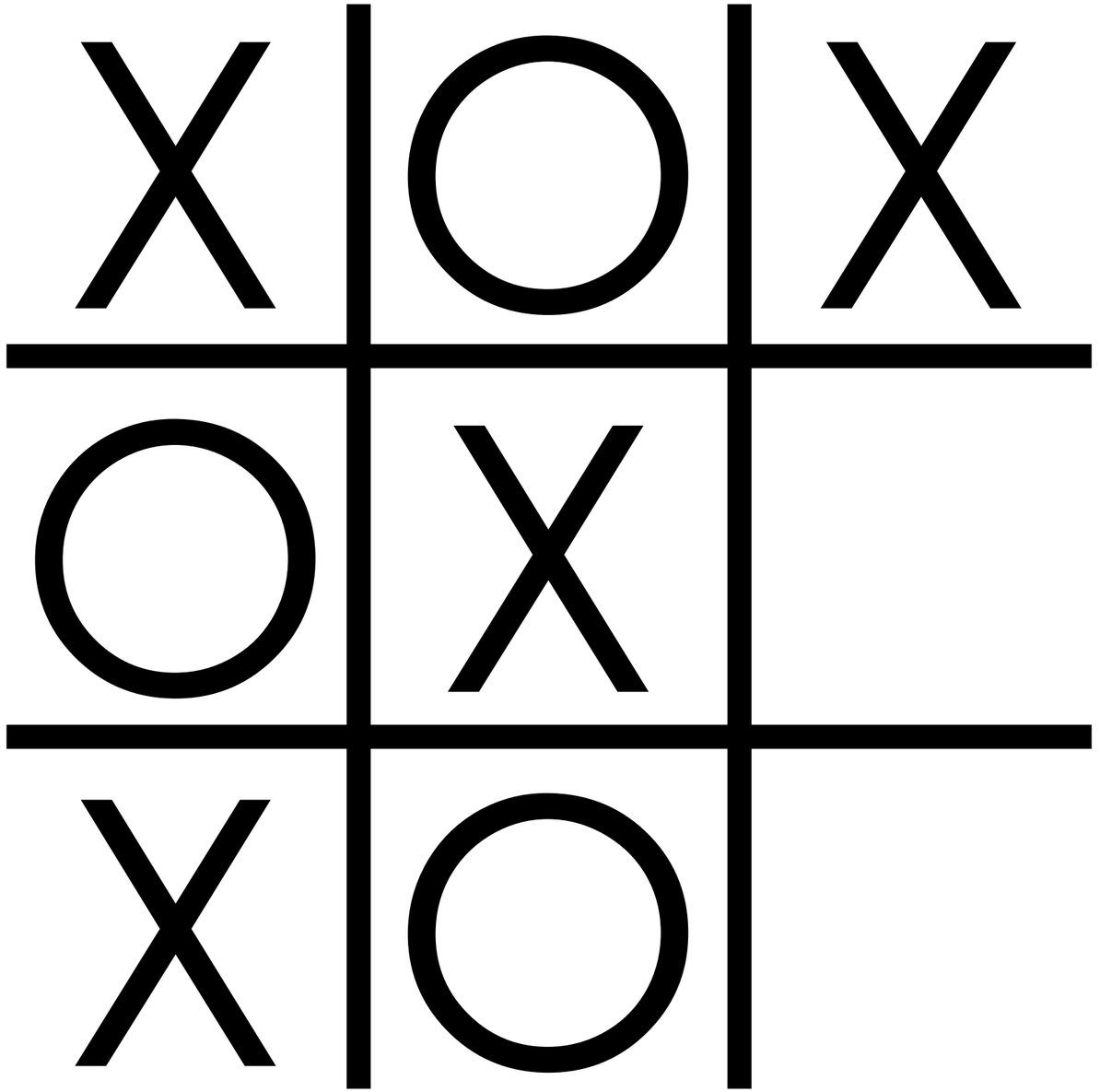
A *Lower* Layer.

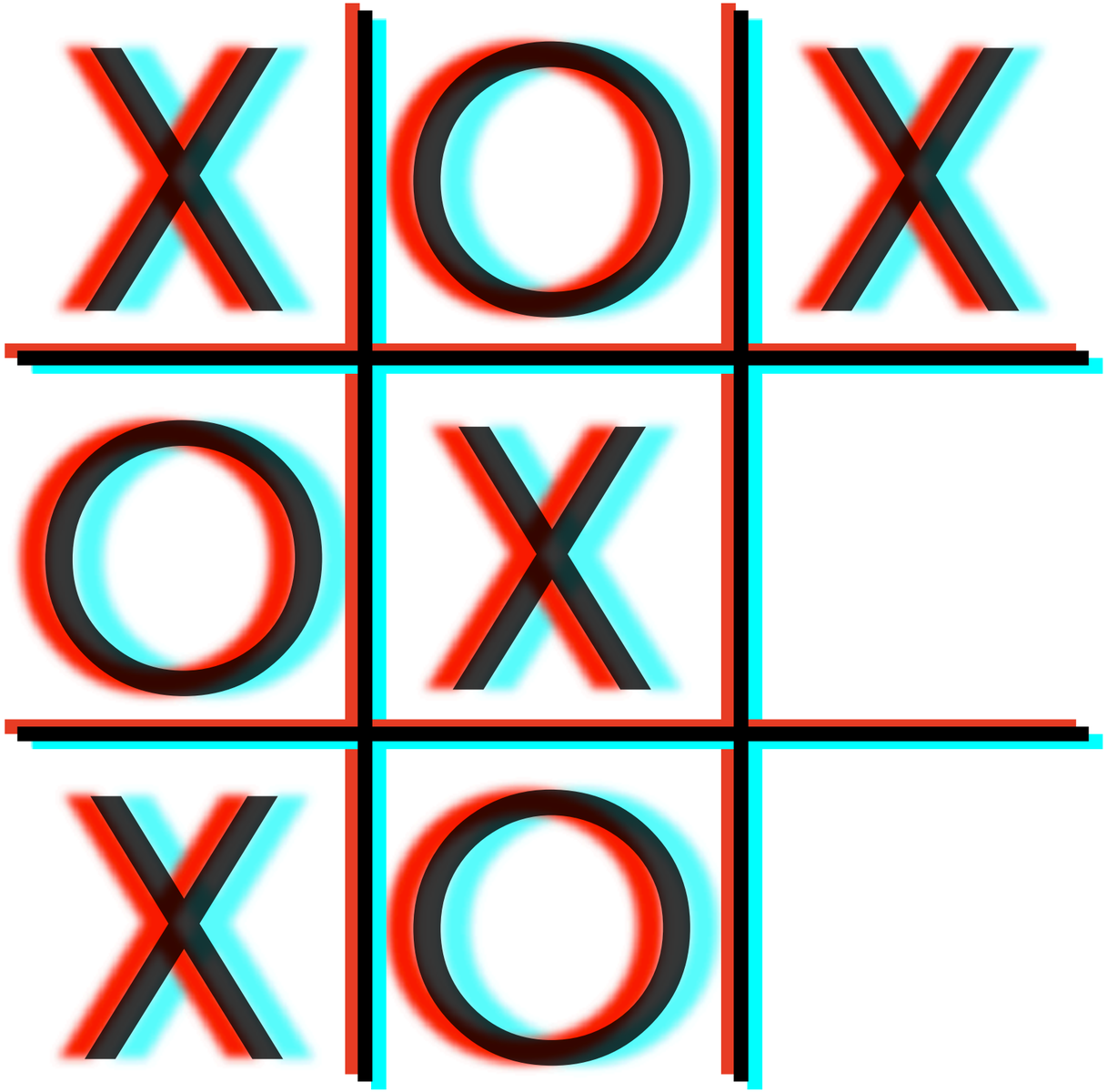
A **Union Filesystem** has three layers:

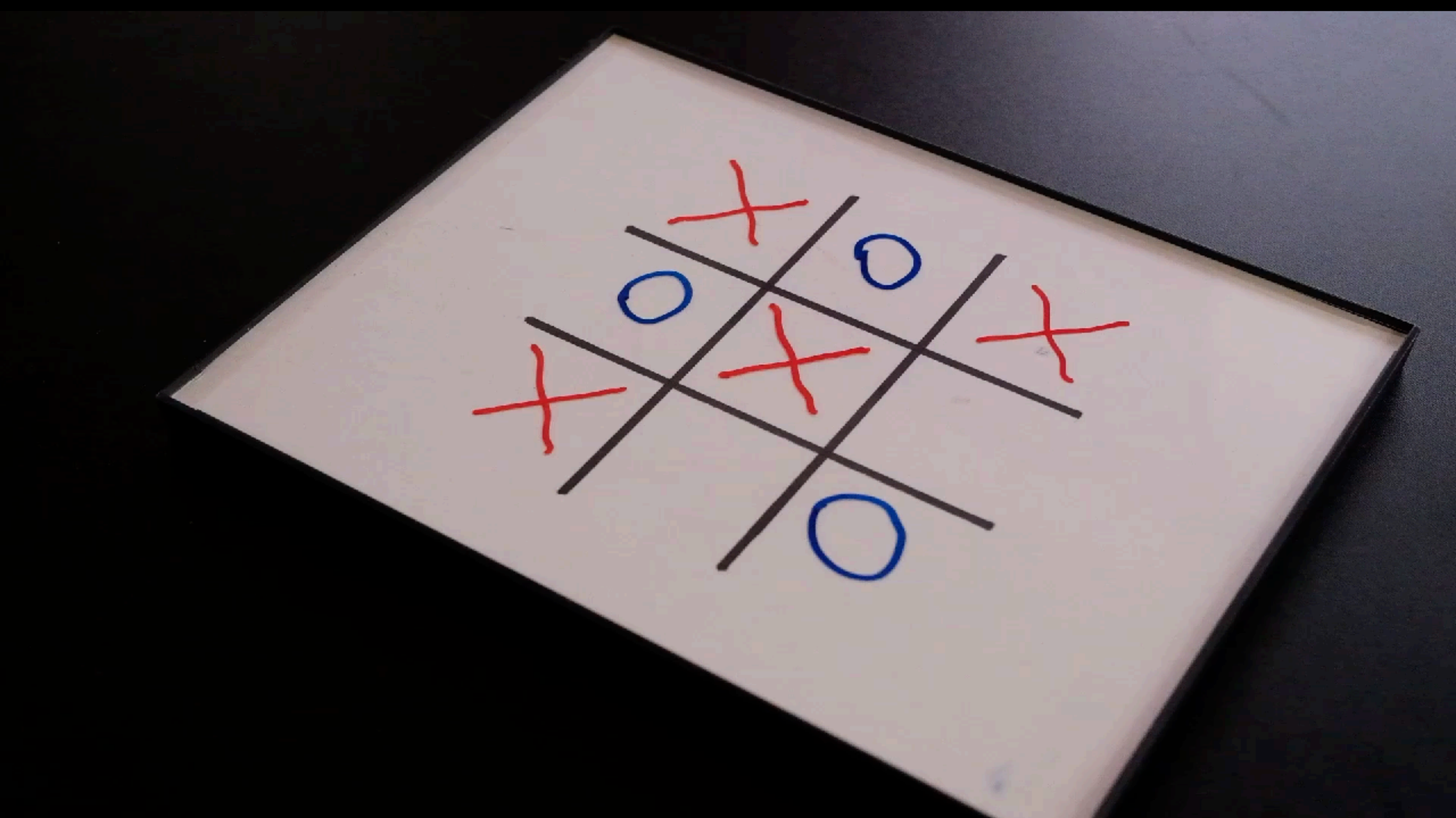
A *Merge* or *Union* Layer: The **game.**

An *Upper* Layer: The **player's moves**.

A *Lower* Layer: The **board and rules**.




X O X
O X
X O



The **Upper** (merge) layer isn't "real."

The **Upper** (merge) layer isn't "real."

It's a perceived (calculated) view of **two overlapping layers**.

The **Upper** (merge) layer isn't "real."

It's a perceived (calculated) view of **moves** and **the board.**

Union Filesystem Lab

```
# Create some directories:

mkdir -p ~/jotb/{image,c1_container,c1_work,container1}

# cd into the main directory:

cd ~/jotb
```

```
# Set the architecture:

arch="$(arch)"
arch="${arch/arm64/aarch64}"
arch="${arch/amd64/x86_64}"

# wget Alpine:

wget -qO- https://dl-cdn.alpinelinux.org/alpine/v3.19/releases/"${arch}"/
alpine-minirootfs-3.19.1-"${arch}".tar.gz \
     | tar xvz -C ~/jotb/image
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c1_container, \
     workdir=c1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c1_container, \
     workdir=c1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c1_container, \
     workdir=c1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c1_container, \
     workdir=c1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c1_container, \
     workdir=c1_work \
     overlay container1
```

```
# List files in different layers:

ls -l ~/jotb/image
ls -l ~/jotb/c1_container
ls -l ~/jotb/container1
```

```
# Create files in different layers:

echo "AAA" > ~/jotb/image/AAA
echo "BBB" > ~/jotb/c1_container/BBB
```

```
# Change a file in the container layer:

echo "aaa" > ~/jotb/c1_container/AAA
```

```
# Check the contents of file AAA in each layer:

cat ~/jotb/image/AAA
cat ~/jotb/c1_container/AAA
cat ~/jotb/container1/AAA
```

```
# Update PATH (just in case!)

PATH=$PATH:/bin
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container1 \
        /bin/sh
```

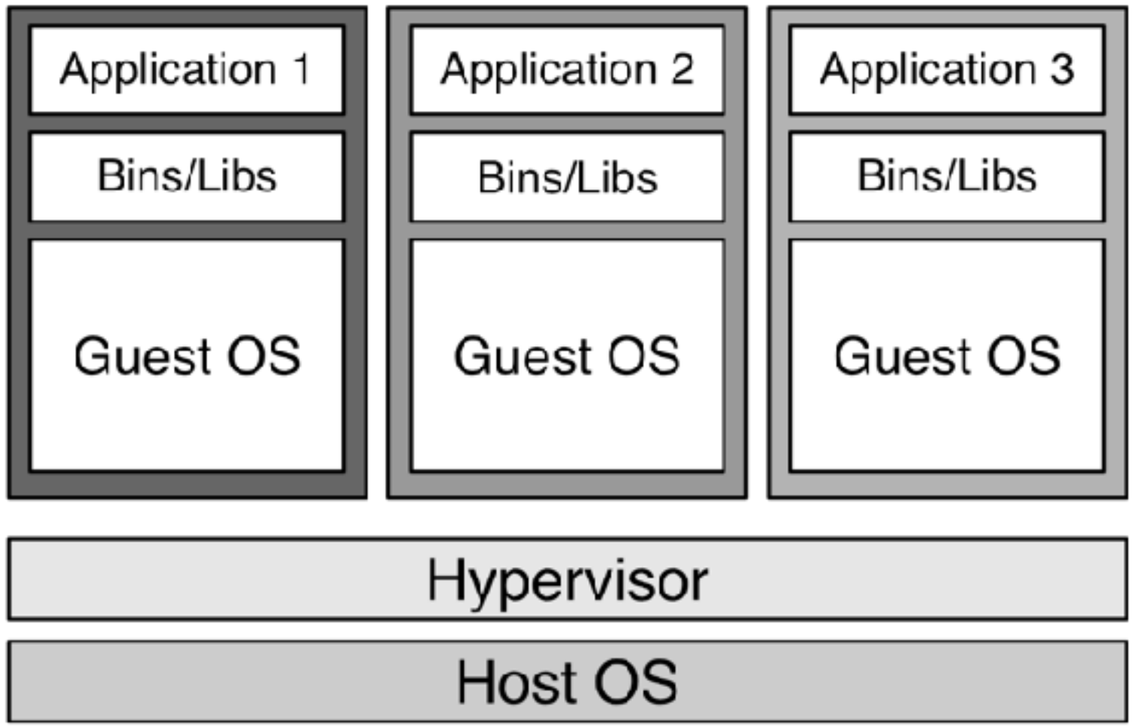
```
# Check the hostname:

hostname
```

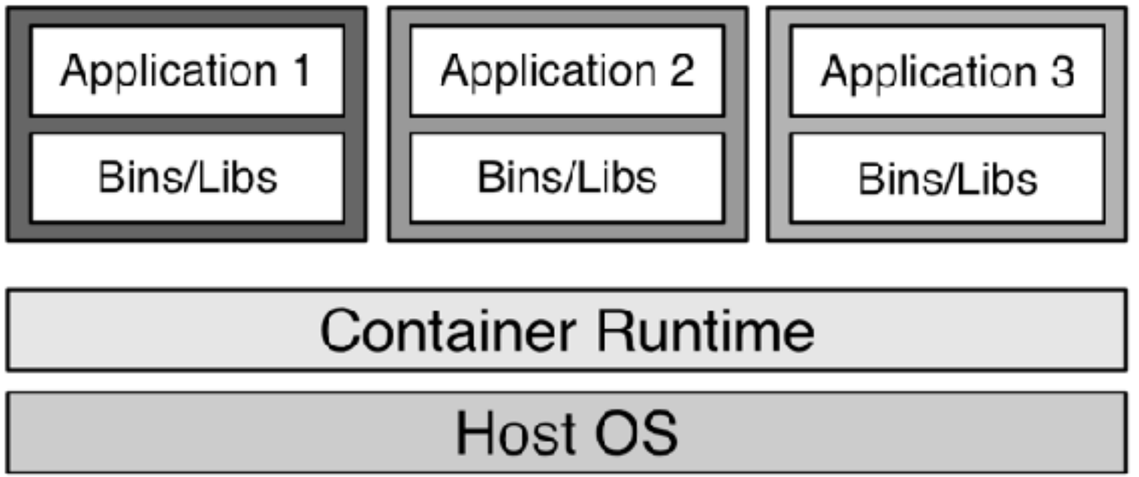
```
# Change the hostname

hostname alpine
hostname
```

```
# Explore the "container"

cat /etc/os-release
uname -a
whoami
echo $$
pwd
```

```
# What files are here?

ls -l /
cat AAA
cat BBB
```

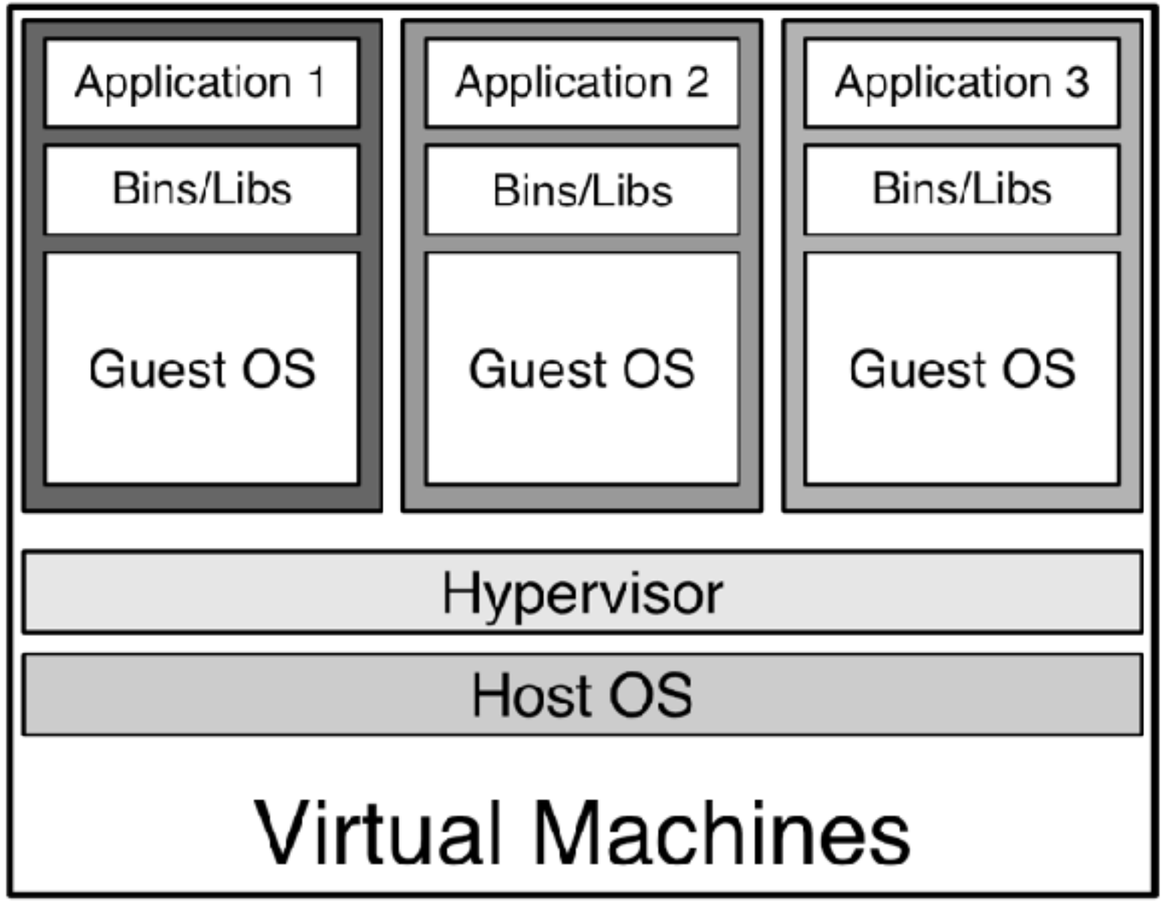
Containers are Magic!



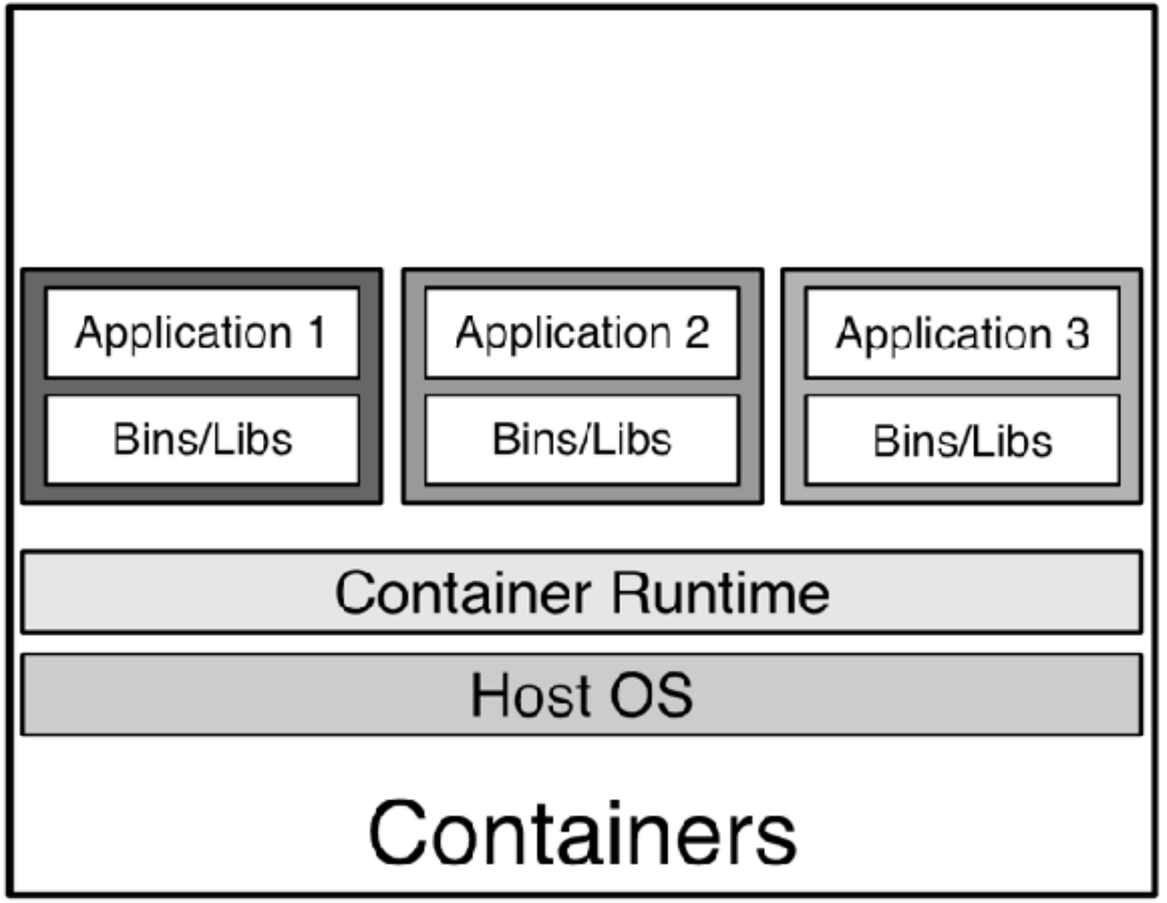


Three full operating systems

Three application/executable directories

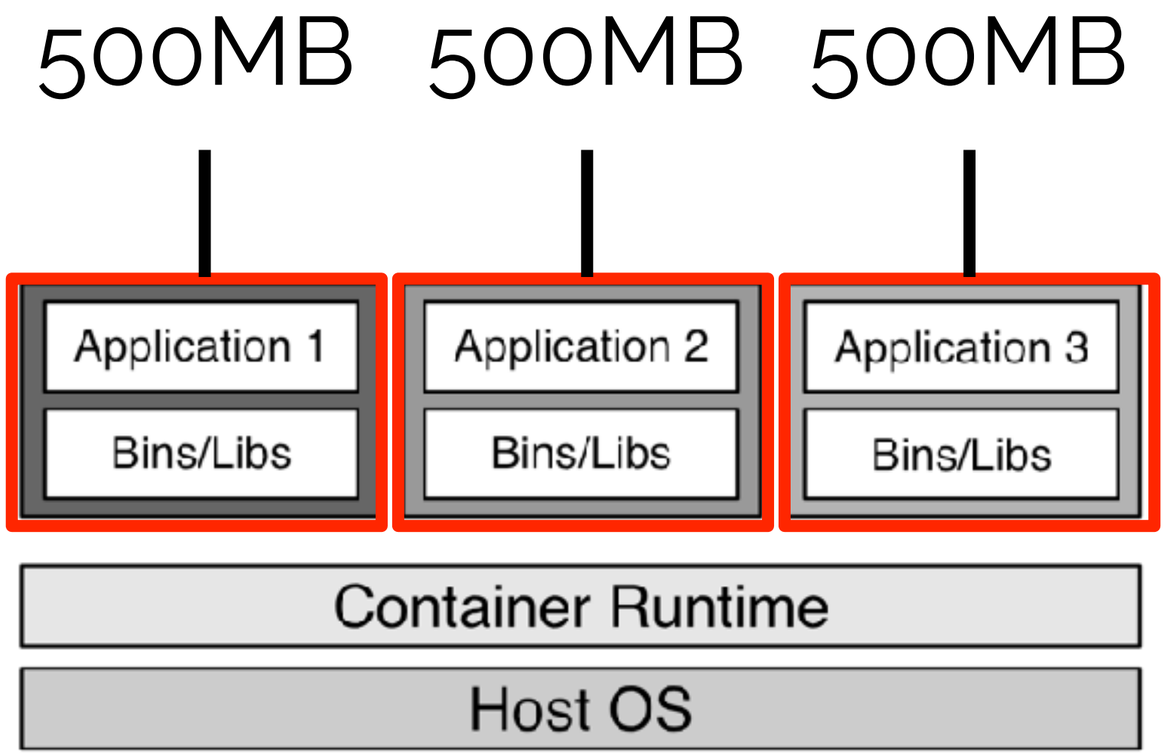


Three full operating systems



Three application/executable directories

How much space do three containers use?



500MB
+ 500MB
+ 500MB

1,500MB

Multiple identical containers will share an **image**.

```
# Prepare directories for a second container

mkdir -p ~/jotb/{c2_container,c2_work,container2}

cd ~/jotb
```

```
# Create a second overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c2_container, \
     workdir=c2_work \
     overlay container2
```

```
# Create a second overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c2_container, \
     workdir=c2_work \
     overlay container2
```

```
# Create a second overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=c2_container, \
     workdir=c2_work \
     overlay container2
```

```
# Set the environment (just in case!)

PATH=$PATH:/bin

# "Run" the "container"

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container2 \
        /bin/sh
```

```
# What files are here?

ls -l /
cat AAA
cat BBB
```

```
# Manipulate files within the container:

echo "111" > /AAA
echo "222" > /BBB
```

```
# Check the contents of files from the host:

cat ~/jotb/image/AAA
cat ~/jotb/c2_container/AAA
cat ~/jotb/c2_container/BBB
```

Containers **persist** state (moves) in the run-time layer of its Union Filesystem. Containers are **stateful**.

```
# Exit and restart the container:

exit

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container2 \
        /bin/sh
```

```
# Confirm the file contents:

ls -l /
cat AAA
cat BBB
```

Container Properties

Dropping a **Container** removes its **Layers.**

```
# "Drop" the container:

sudo umount container2

rm -fr ~/jotb/container2
rm -fr ~/jotb/c2_container
rm -fr ~/jotb/c2_work
```

Containers are **Ephemeral.**

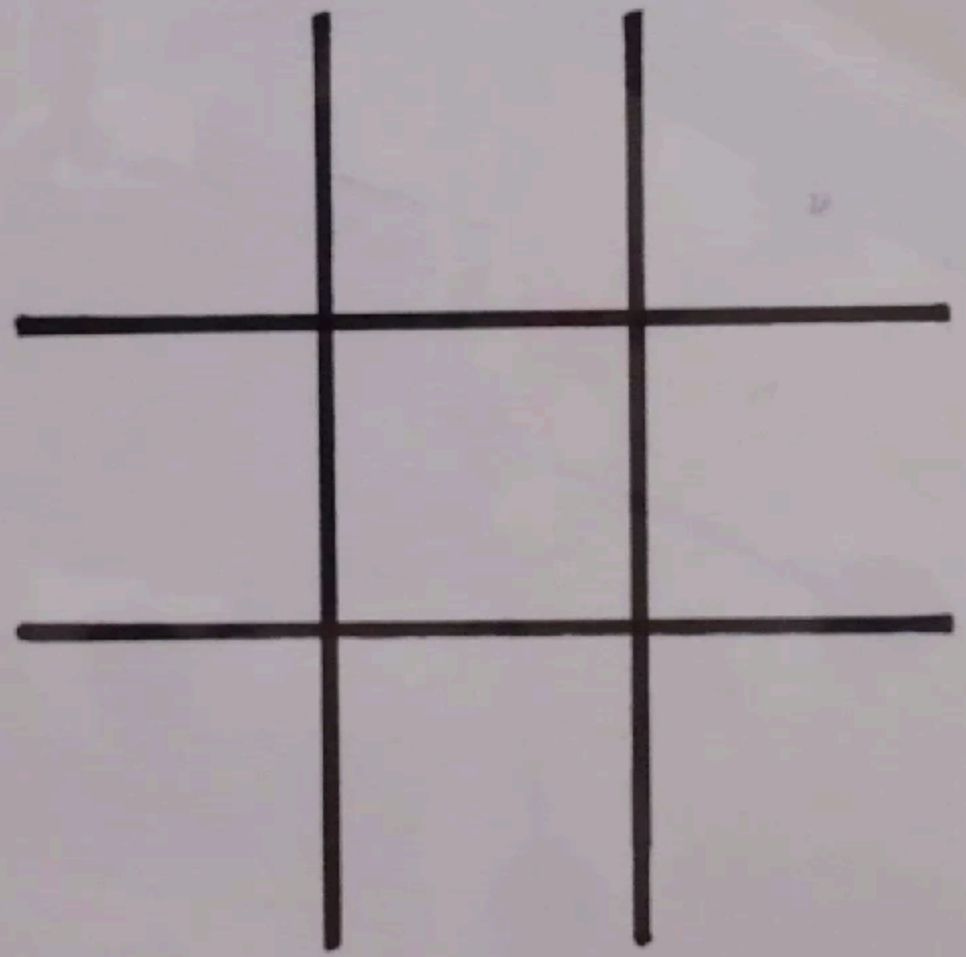
Ephemeral does not have a time limit.

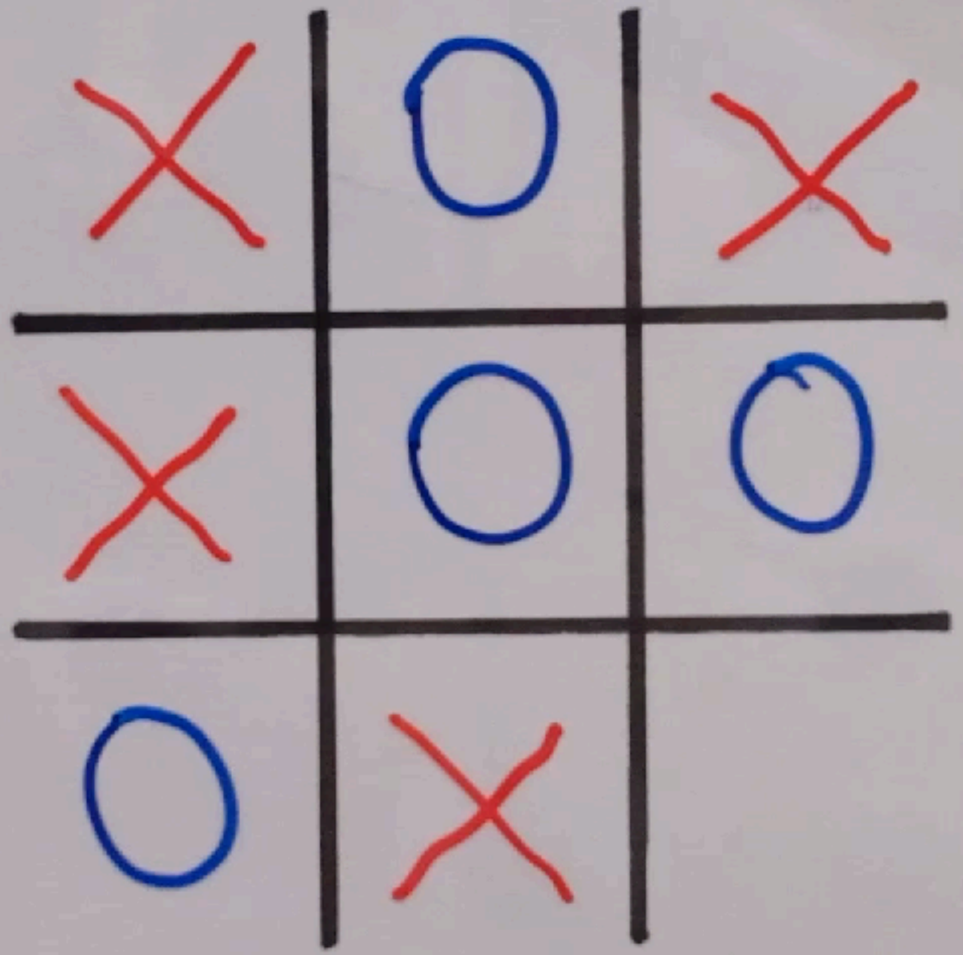
Ephemeral does not have a time limit.

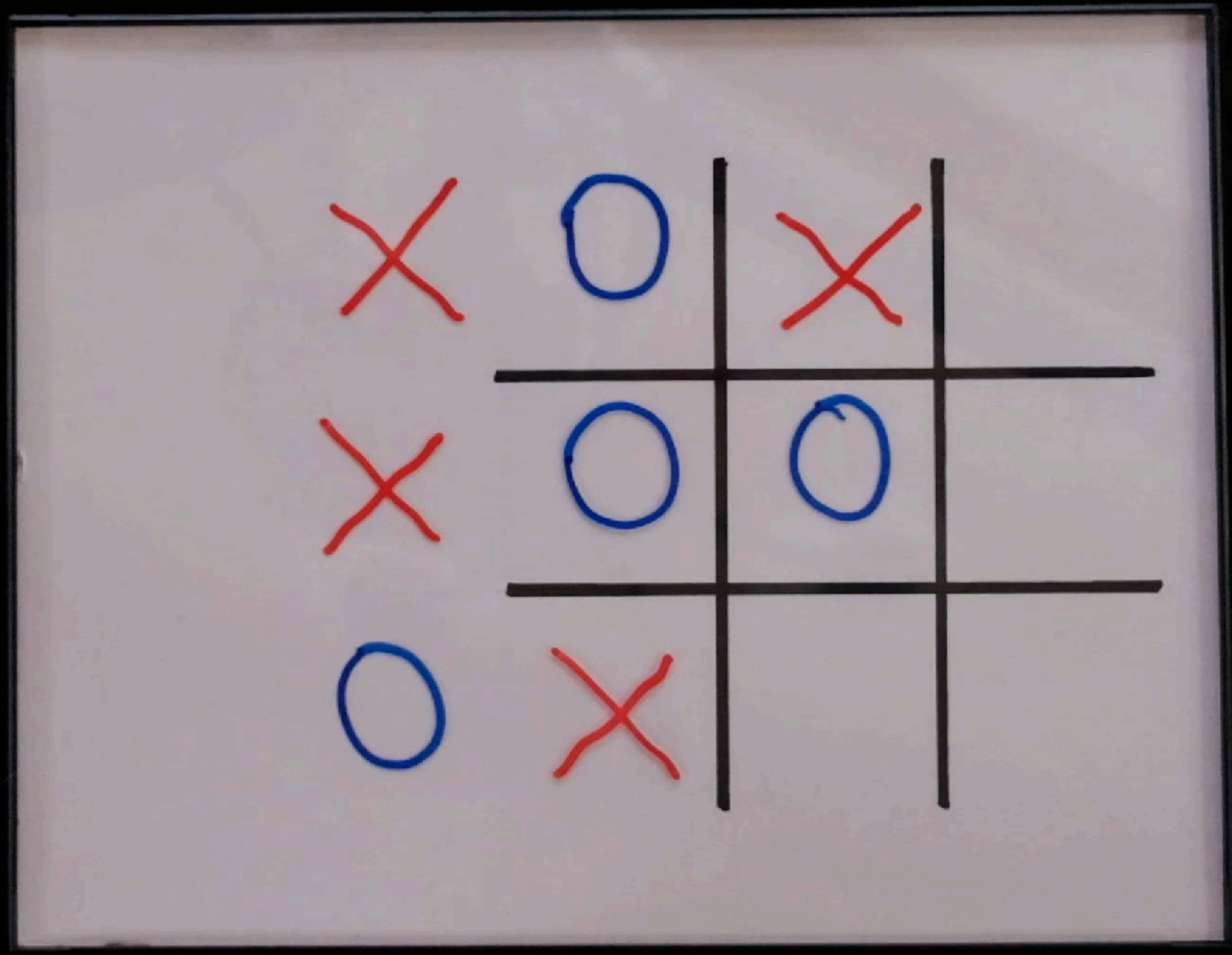
On a long enough scale,
everything is ephemeral.

Images are **immutable** and *never change*.

Stalemate!

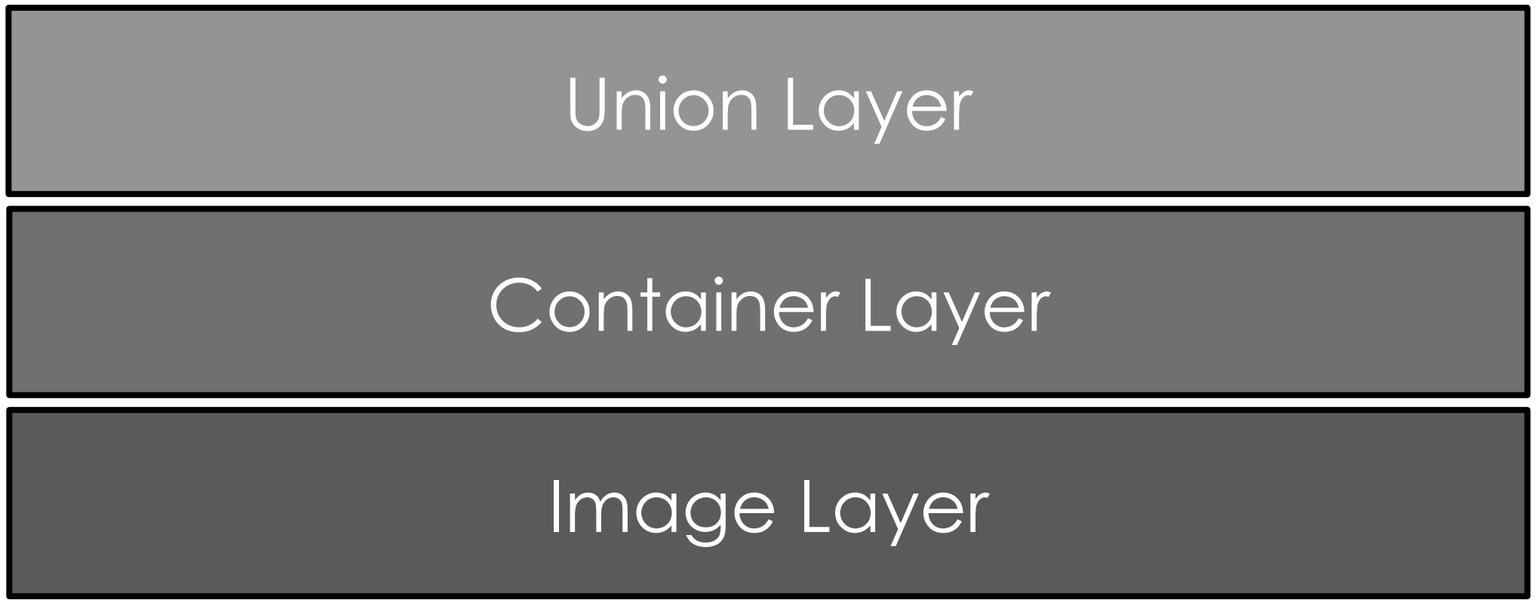


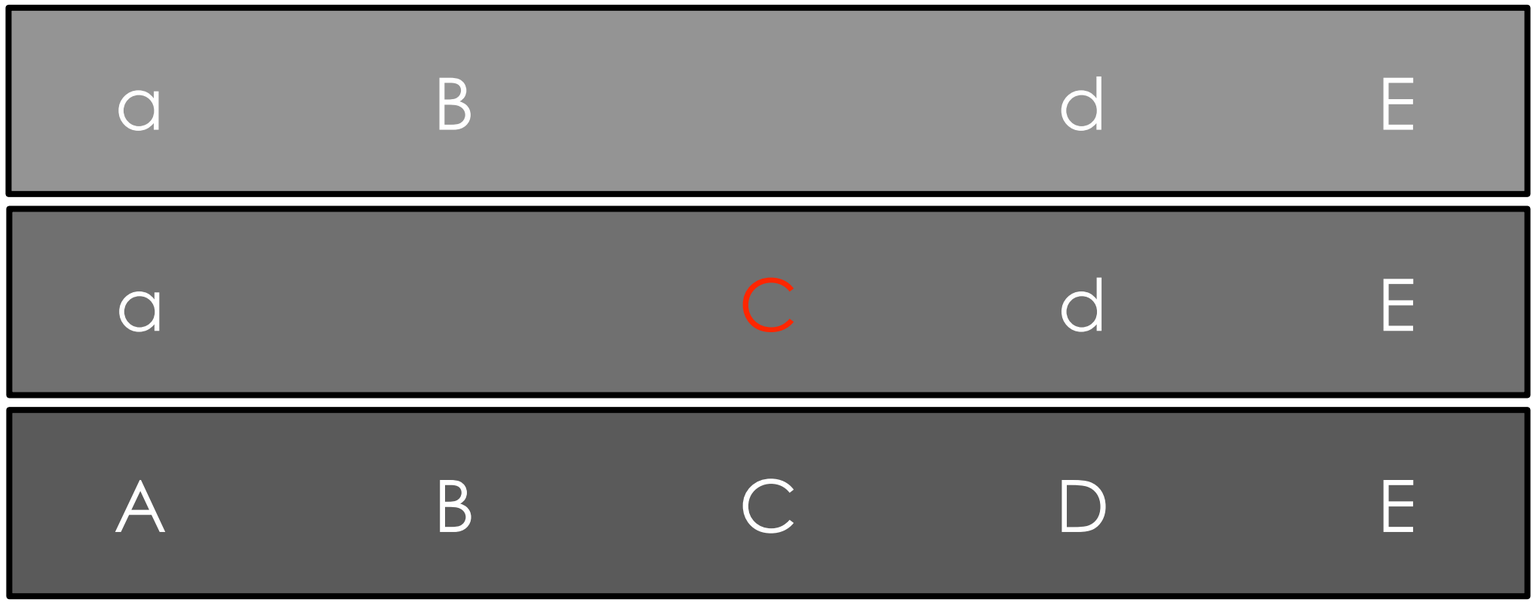




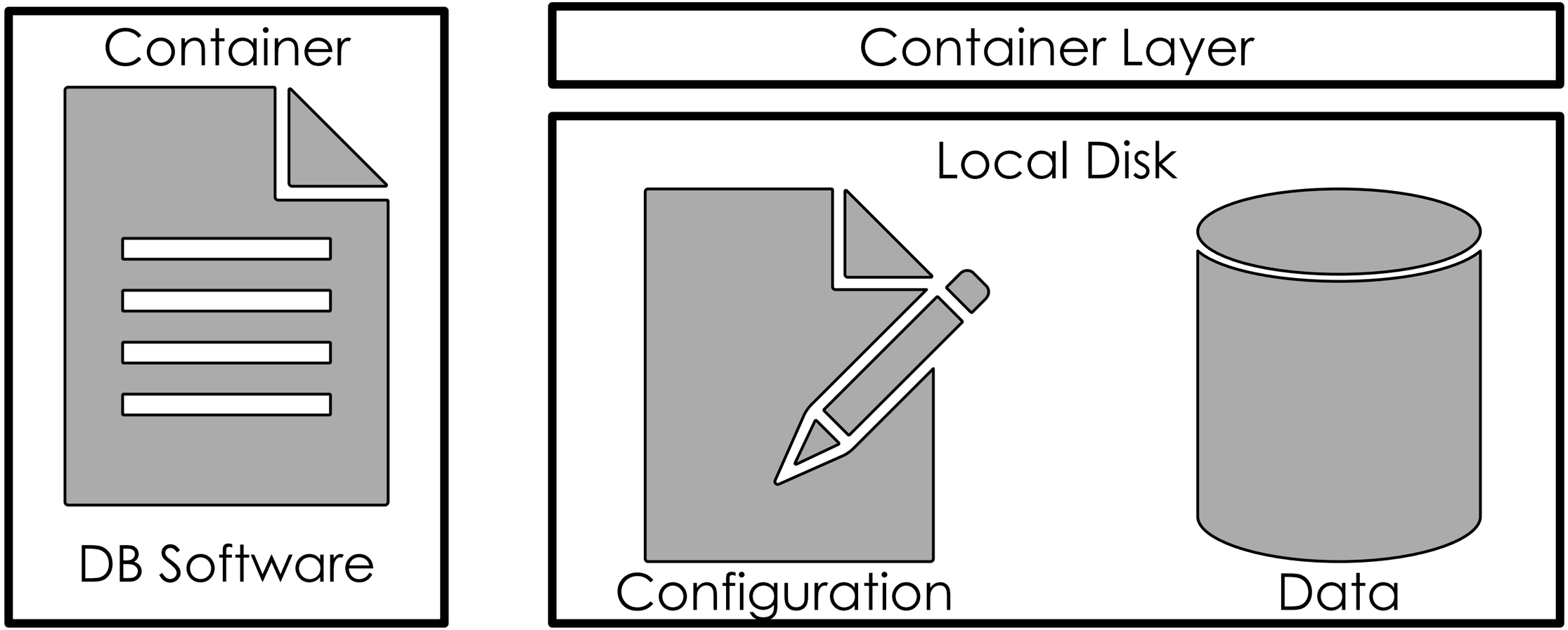
Volumes

Volumes "externalize" container directories.


Union Layer
Container Layer
Image Layer


a B d E
a C d E
A B C D E

Database Containers



Database Image Rule #1:

If there is a database, start it.

If not, create a new database.

Database Container Startup

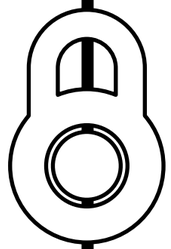
```
docker volume create ${CONTAINER_NAME}_data

docker run -d \
       --name ${CONTAINER_NAME} \
       -v /data/${CONTAINER_NAME}:<data_directory> \
       <image_name>
```

Default Volumes

Linux

```
ls -l /var/lib
drwx--x--- root root docker
```

Docker Desktop



Virtual Machine

`/var/lib/docker`

```
mkdir -p /data/${CONTAINER_NAME}

docker volume create \
        --opt type=none \
        --opt o=bind \
        --opt device=/data/${CONTAINER_NAME} \
        ${CONTAINER_NAME}_data
```

Database Container Startup

```
docker run -d \
       --name ${CONTAINER_NAME} \
       -v /data/${CONTAINER_NAME}:<data_directory> \
       <image_name>
```

Database Image Rule #1:

If there is a database, start it.

If not, create a new database.

Clone the Data Directory

```
cp -rp /data/${CONTAINER_NAME}/ \
       /data/clone
```

Clone the Data Directory

```
cp -rp /data/clone/ \
       /data/${CONTAINER_NAME}
```

Clone the Data Directory

```
cp -rp /data/clone/ \
       /data/${CONTAINER_NAME}

cp -rp /NFS/gold_data/clone/ \
       /data/${CONTAINER_NAME}
```

Clone the Data Directory

```
cp -rp /data/clone/ \
       /data/${CONTAINER_NAME}

cp -rp /NFS/gold_data/clone/ \
       /data/${CONTAINER_NAME}

cp -rp /NFS/gold_data/clone/2024-05-08.0900/ \
       /data/${CONTAINER_NAME}
```

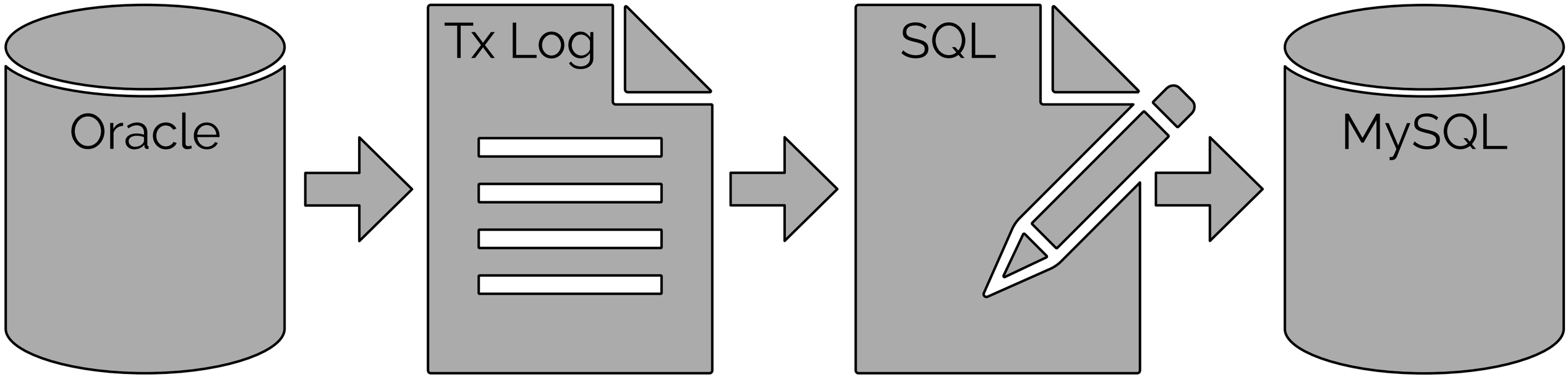
Data is Code

Data *is* Code

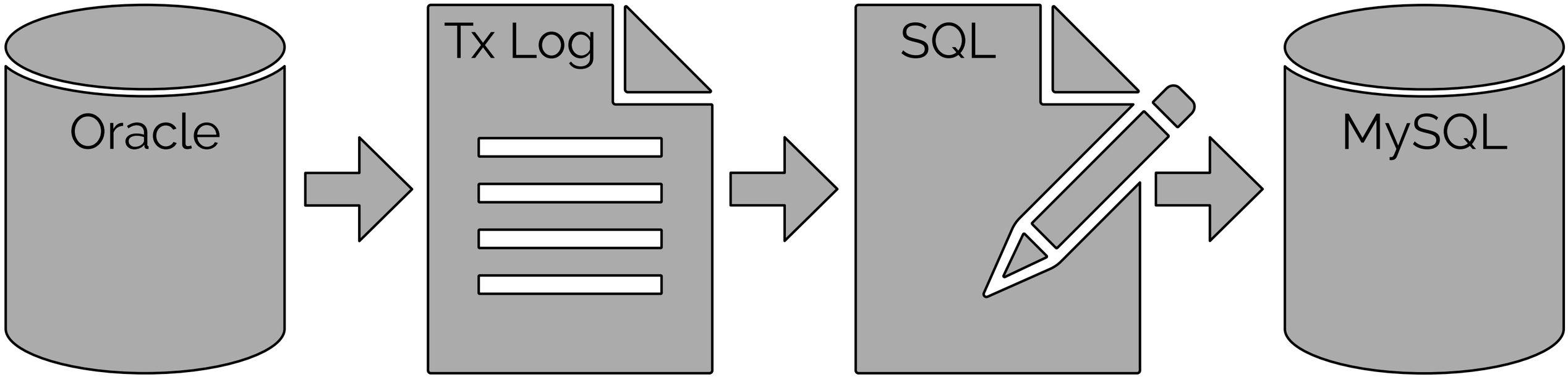
Every record in every database is created in one of three ways:

- `INSERT`
- `UPDATE`
- `DELETE`

Heterogenous replication = synchronous SQL



Heterogenous replication = synchronous SQL



Optional

Transformation

If data can be codified, **data is versionable.**

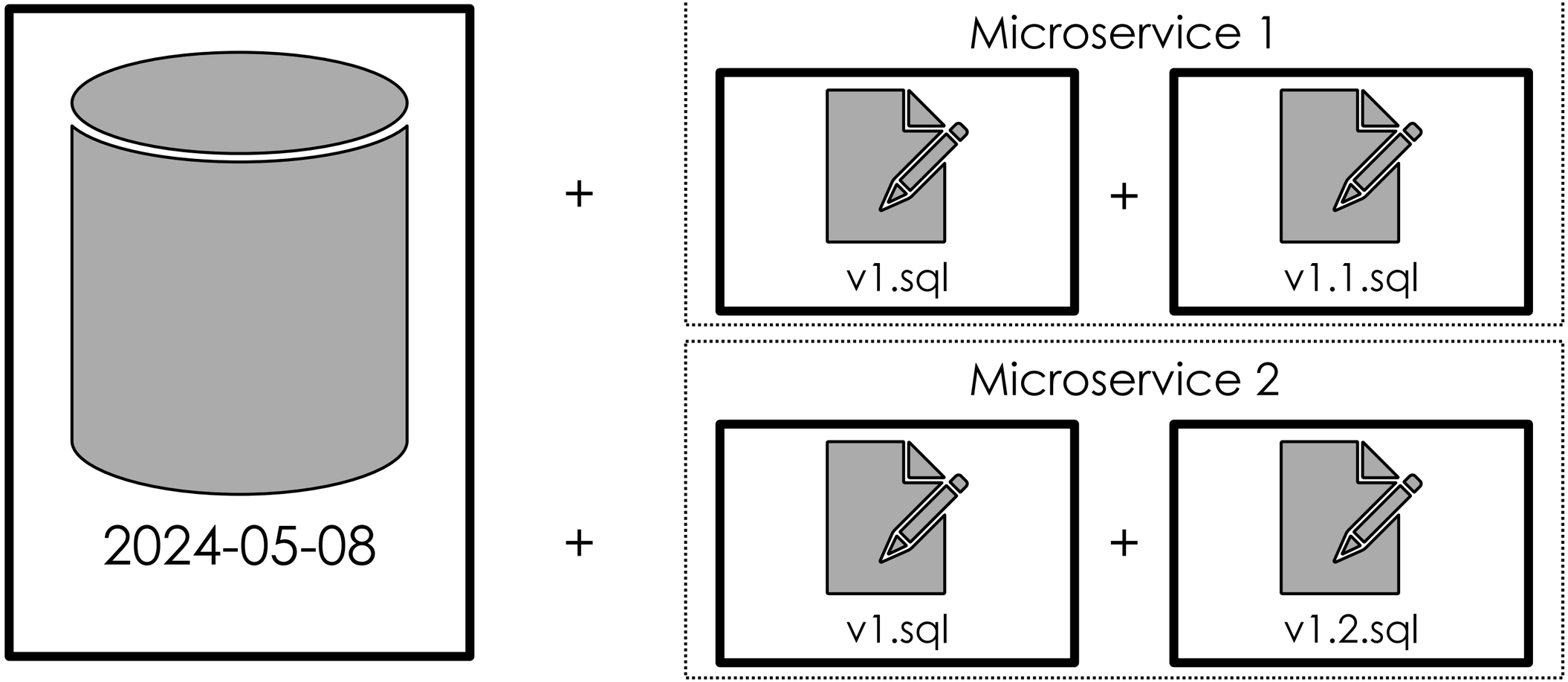
Entrypoints

- Generally adapted but *not* an OCI standard
- Implementation varies by image, vendor/author
    - Setup only
    - Setup & startup

Entrypoints

```
CONTAINER_NAME=V20240508

cp -r /data/2024-05-08/ \
      /data/${CONTAINER_NAME}

docker run -d \
       --name ${CONTAINER_NAME} \
       -v /data/${CONTAINER_NAME}:<data_directory> \
       -v /code/${CONTAINER_NAME}:/docker-entrypoint-initdb.d \
       <image_name>
```

Versioned Data



Building Images

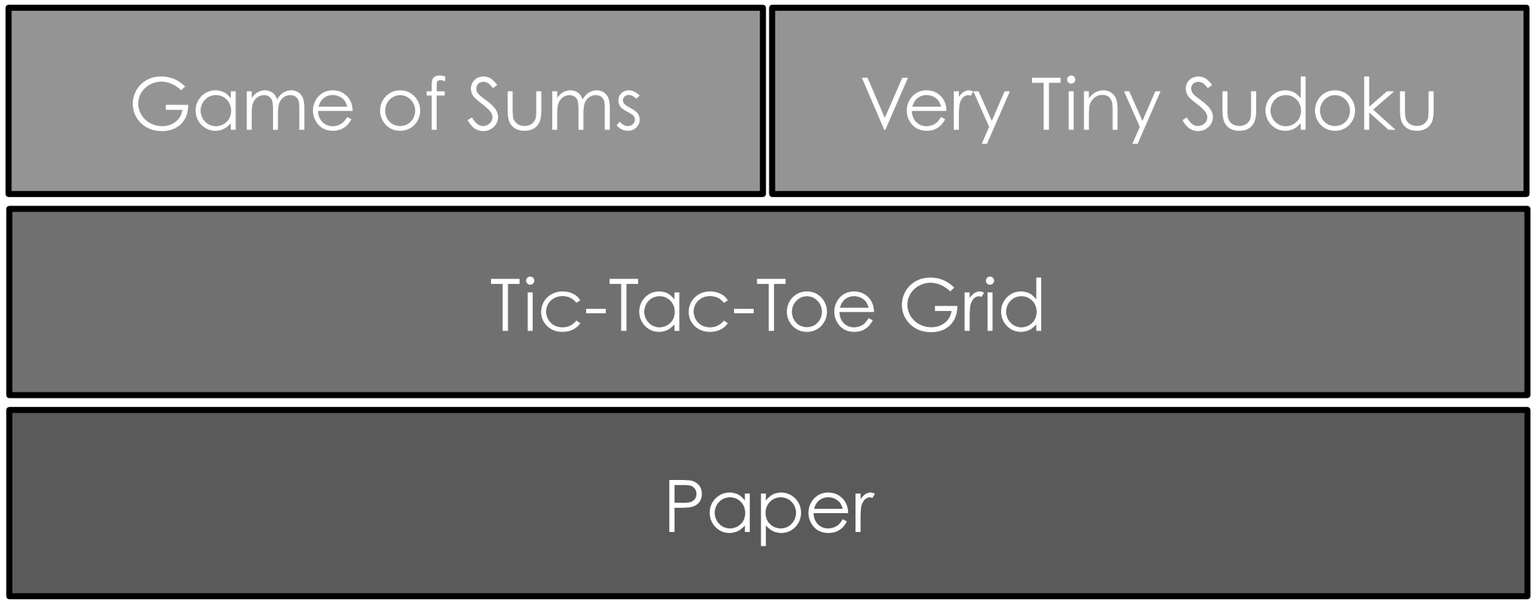
A **Containerfile** is a recipe for building images.

Tic-Tac-Toe Grid

Paper

Image layers are *modular* and *reusable*.

Layers can be **cached** and reused among multiple images.


Game of Sums
Very Tiny Sudoku
Tic-Tac-Toe Grid
Paper

```
# Create some directories:

mkdir -p ~/jotb/{image,c3_container,c3_work,container3}

# cd into the main directory:
cd ~/jotb
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=container1, \
     upperdir=c3_container, \
     workdir=c3_work \
     overlay container3
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=container1, \
     upperdir=c3_container, \
     workdir=c3_work \
     overlay container3
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=container1, \
     upperdir=c3_container, \
     workdir=c3_work \
     overlay container3
```

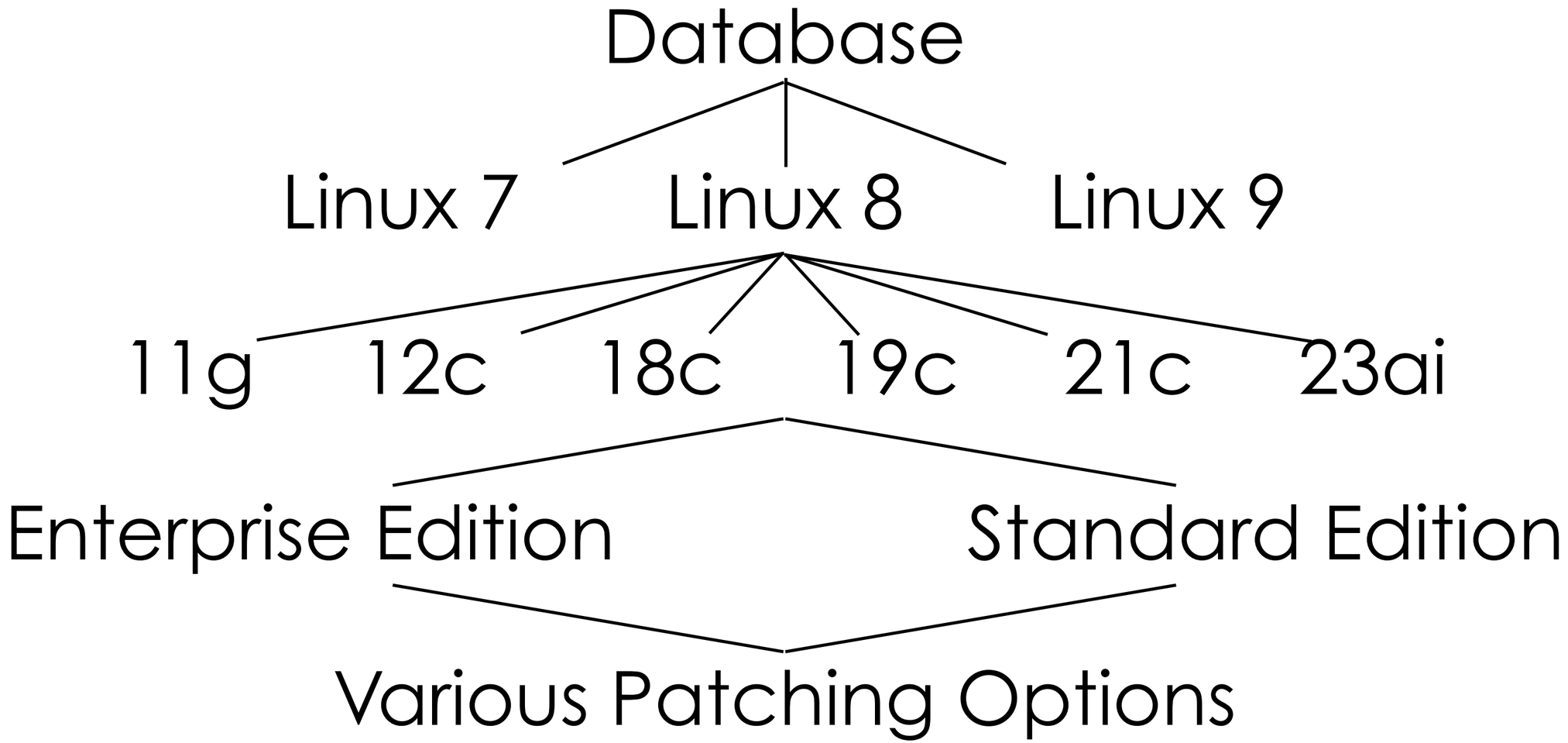
```
# Update PATH (just in case!)

PATH=$PATH:/bin
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/jotb/container3 \
        /bin/sh
```

Context is a pantry of image ingredients.

Ignore files limit and control the build context.

Repository Structure


Database
Linux 7
Linux 8
Linux 9
11g
12c
18c
19c
21c
23ai
Enterprise Edition
Standard Edition
Various Patching Options

Questions

sean.scott@viscosityna.com
https://linktr.ee/oraclesean

Contact Me

sean.scott@viscosityna.com
https://linktr.ee/oraclesean