Container Technology

FH Sustainable Computing Workshop

Christoph Wissing (DESY)
March, 2025

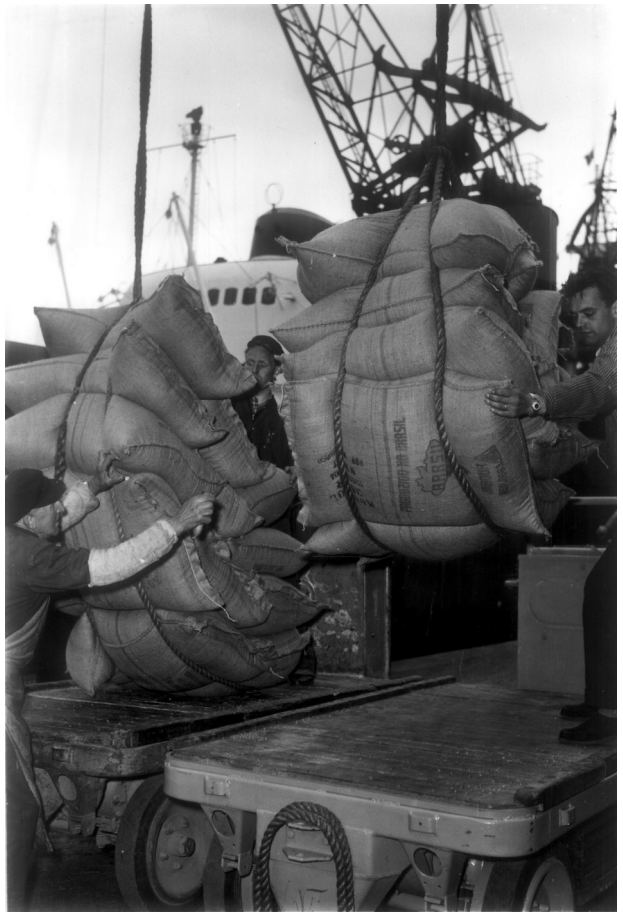
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

DESY.

Containers: Game changer in global economics and modern IT

Treatment of Goods – Hamburg Harbour ~1950

Individual packages



- Individual sacks, bags, vessels
- Many works needed in the harbours

Photo references:
Deutsche Fotothek, Germany - In Copyright - Educational Use Permitted.
https://www.europeana.eu/de/item/437/item_FRTGT7KZTCNQBKLYEYK5UZDSCK36N7E6
Deutsche Fotothek, Germany - In Copyright - Educational Use Permitted.
https://www.europeana.eu/de/item/440/item_SD7XJGL3OJOLQTLDK5GTCUTXOSUCTOGT
Deutsche Fotothek, Germany - In Copyright - Educational Use Permitted.
https://www.europeana.eu/de/item/440/item_RWV43GBGVPDMWLJOJ4NLL4IJACI7UL5F

Treatment of Goods Today

Containers, Containers, Containers

https://pixabay.com/de/users/dendoktoor-14802912/

"https://pixabay.com/de/users/stocksnap-894430/

- Backbone of global shipping: ISO Containers
- Introduced end of the 1950s
- ISO Containers
  - 20” (TEU 20 foot Equivalent Unit): 5,898 m x 2,352 m x 2,393 m
  - 40” (FEU 40 foot Equivalent Unit): 12,032 m x 2,352 m x 2,393 m
- Modern container ships transport over 20,000 TEUs
- Highly automised habour logistics
- Layup times several hours compared to days before

Containers: Game Changer in IT

Again Containers everywhere

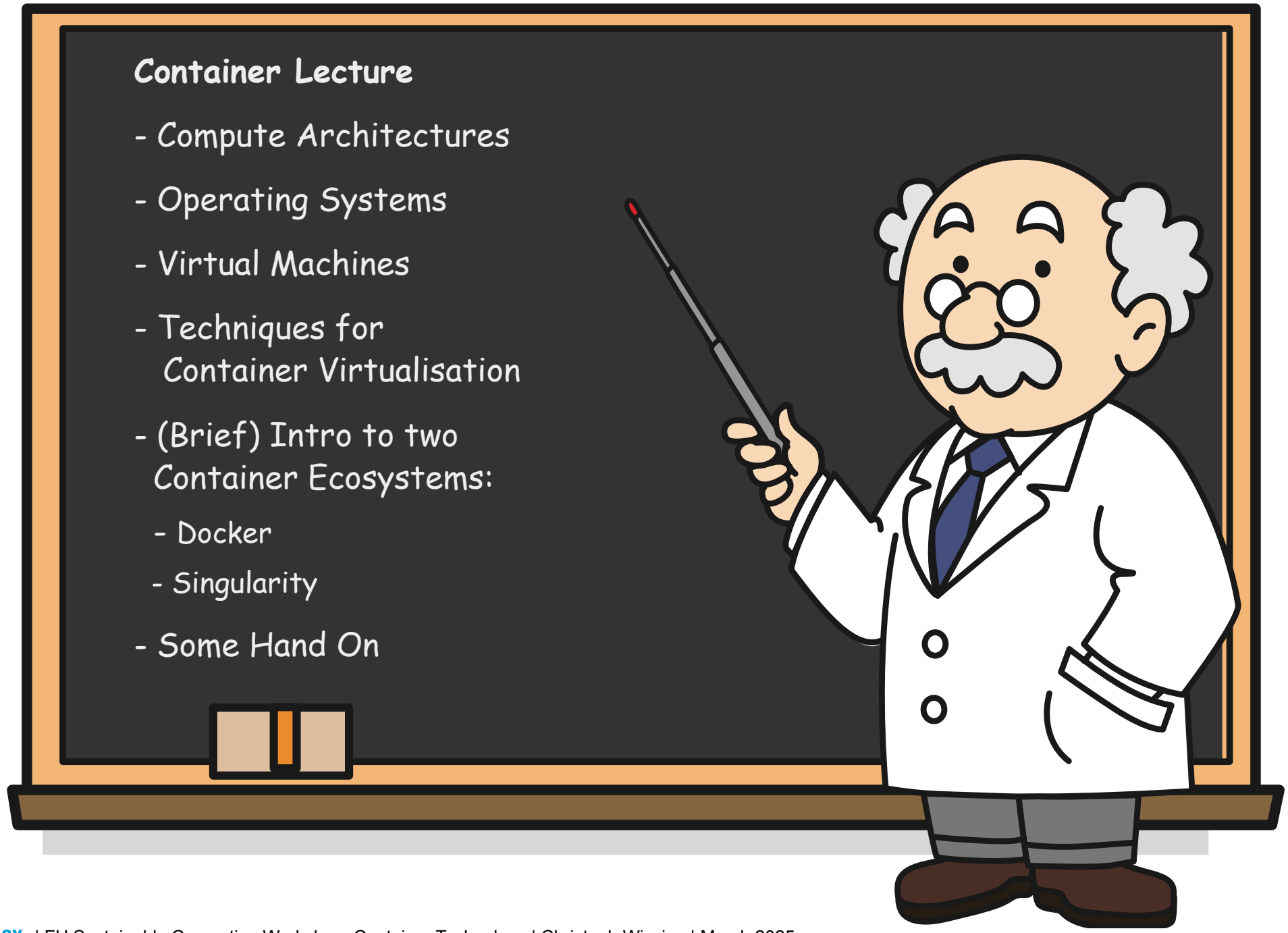
kubernetes Web page https://kubernetes.io

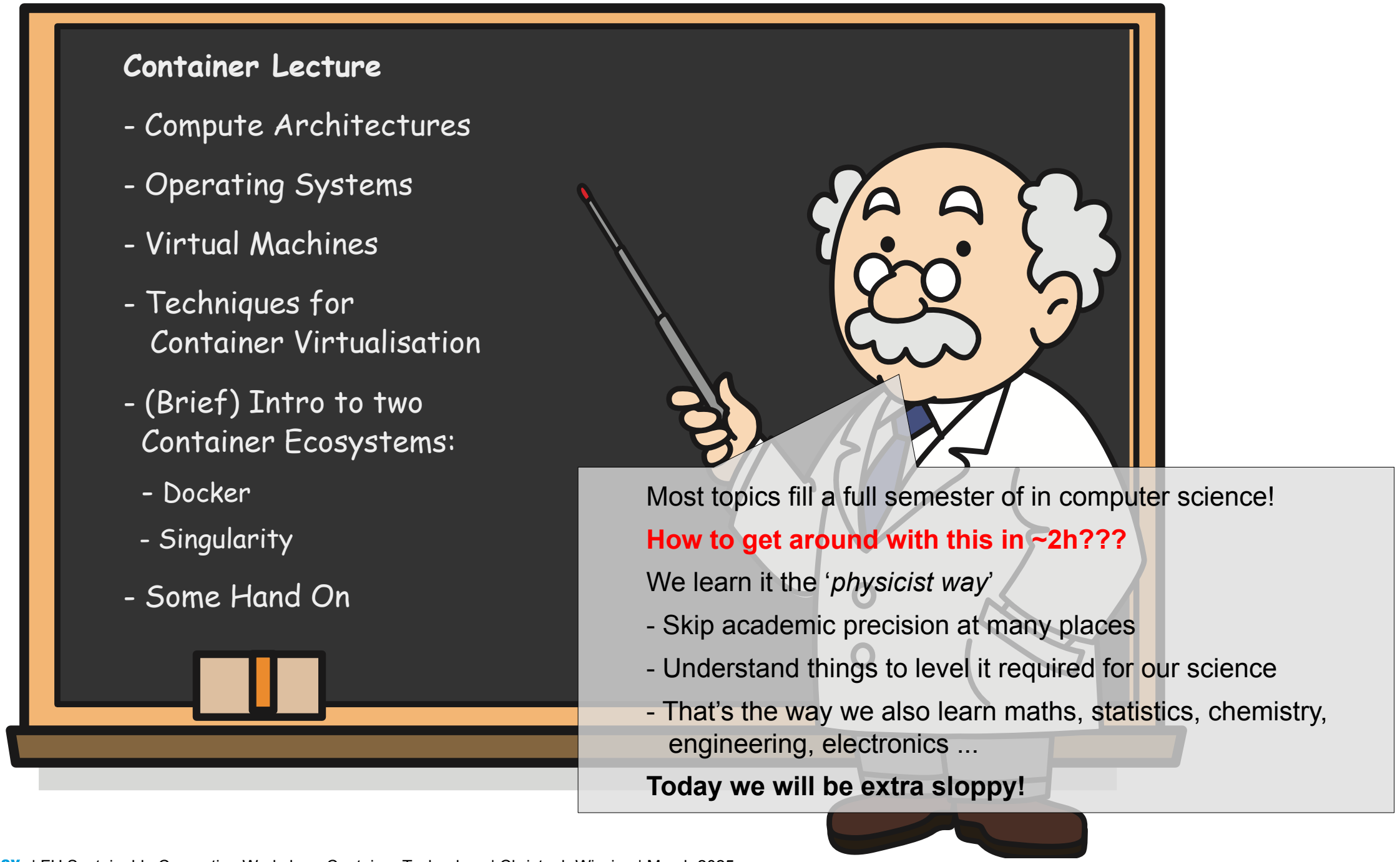
Planet Scale

Designed on the same principles that allows Google to run billions of containers a week, Kubernetes can scale without increasing your ops team.

- Kubernetes is a container orchestration frame work
  - Start (and stop) containers in an elastic mode depending on demand
  - Designed for scalability

- Let’s get started from the beginning...


Container Lecture
- Compute Architectures
- Operating Systems
- Virtual Machines
- Techniques for Container Virtualisation
- (Brief) Intro to two Container Ecosystems:
- Docker
- Singularity
- Some Hand On



Most topics fill a full semester of in computer science!

How to get around with this in ~2h???

We learn it the '*physicist way*'

- Skip academic precision at many places
- Understand things to level it required for our science
- That's the way we also learn maths, statistics, chemistry, engineering, electronics ...

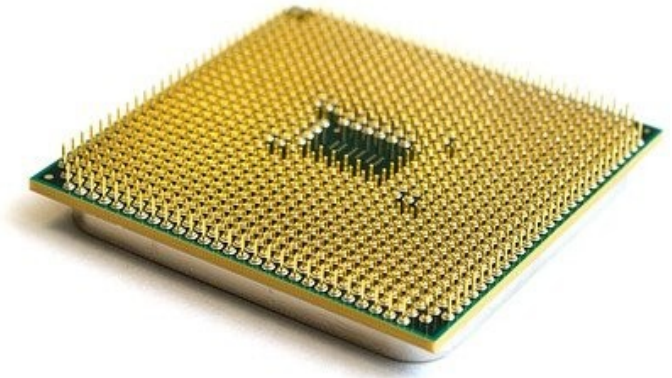
Today we will be extra sloppy!

Computer Architectures

Common CPU Families

x86

- Most common CPU type for desktops and laptops since decades
- Very common also for servers and High Performance Computers
  - In the more recent past only, perhaps 20 years
- Main (competing) manufactures: Intel and AMD



https://pixabay.com/de/users/elasticcomputefarm-1865639/

ARM

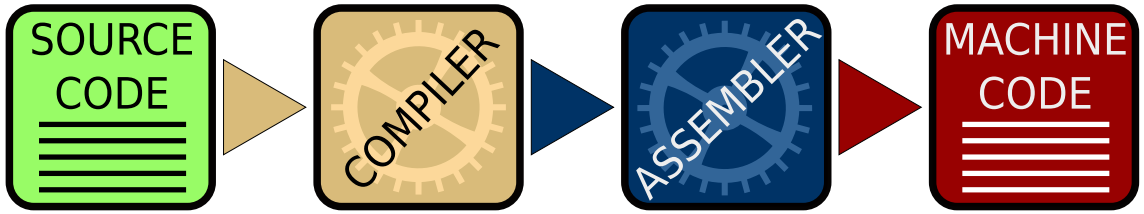
- Used in almost any smartphone or tablet
- Increasing use also in servers, desktops even HPCs
  - Recent Apple Macintosh ‘M1,M2 ...’ is basically ARM
- Many manufactures

Classical Workstation architectures

- In the 1980-90ies each major vendor had its own
  - SUN: UltraSPARC
  - HP: PA-RISC
  - SGI: MIPS
  - Motorola: 68k – Used in early Macs, Amiga, Atari ST
- Of relevance still today: POWER by IBM

From Code to Executable Programs

Compiler



Source code

- Written in programming language (e.g. C, FORTRAN, Python, Go …)
- Simple enough code can be compiled for different architectures

Compiler & Assembler

- In practice one step
- Assembler code architecture dependent (could in principal optimized by a developer)
- (Some) Compiler can generate assembler for a different architecture (*cross-compiler*)

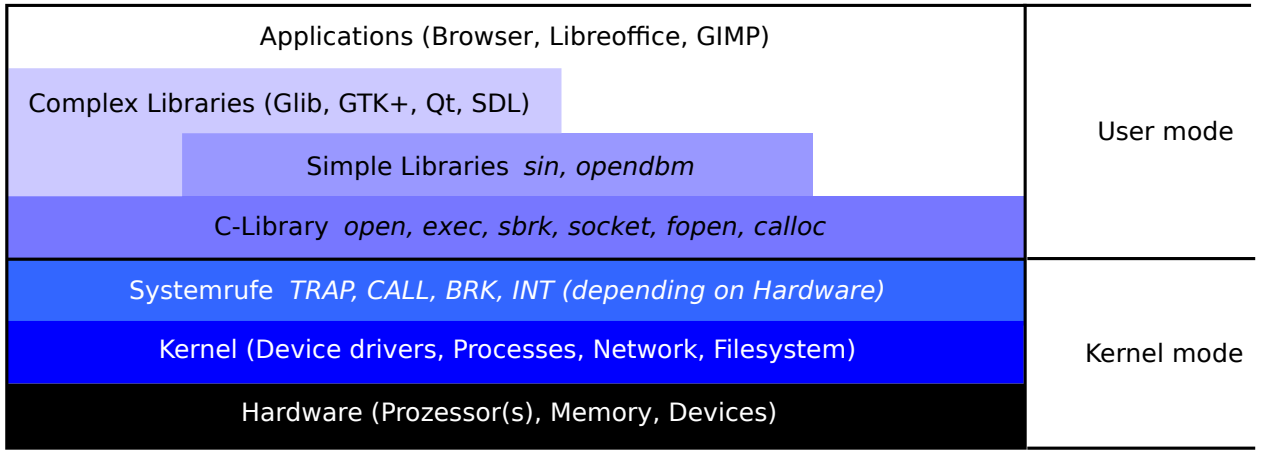
Machine Code

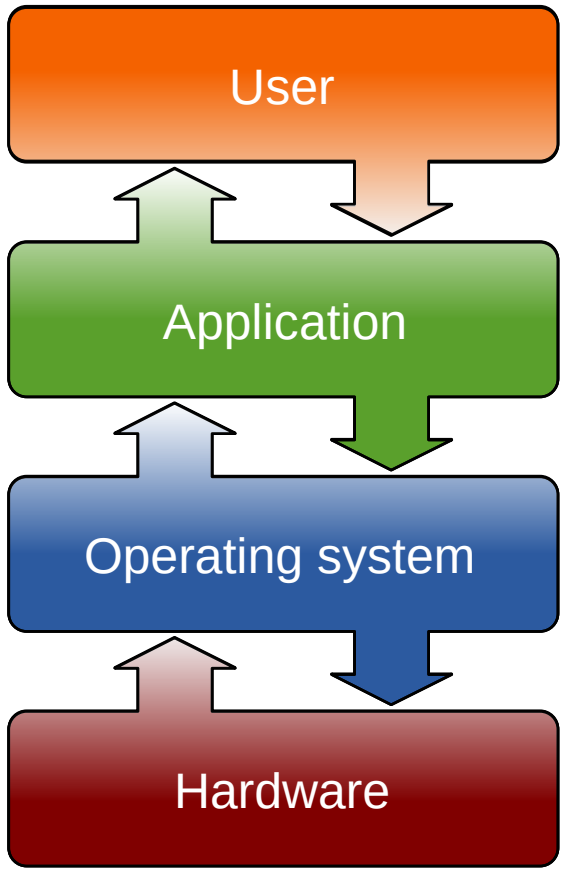
- Can be executed only on the architecture it was compiled for

Operating System

Usually an Operating System (OS) controls a modern computing device

- Not strictly required – a specially prepared application could run directly (very inconveniently)
- OS controls all resources
- Applications run on top of OS
  - Common functionality comes from (core) libraries, e.g. glibc in Linux
- Modern OSes provide support for multiple users





Examples of popular Operating Systems

Microsoft Windows

- Most popular OS for desktop PCs
- Market share over 90%
- Available (mainly) for x86
- Windows 11

MacOS

- OS for Apple Macintosh
  (Usage on other hardware technically possible – but violated license)
- Runs on various architecture
  - Motorola 68k (until early 1990s)
  - PowerPC (until early 2000s)
  - X86 (until ~early 2020s)
  - M1,M2 ... - ARM64 based (recent)

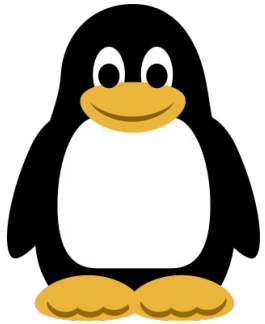
Mac OS

Android

- Most popular OS for mobile devices (phones...)
- Relatively young, appeared in 2008
- Based on Linux
- Mostly used on ARM architecture (but also x86 and others)

android

Linux



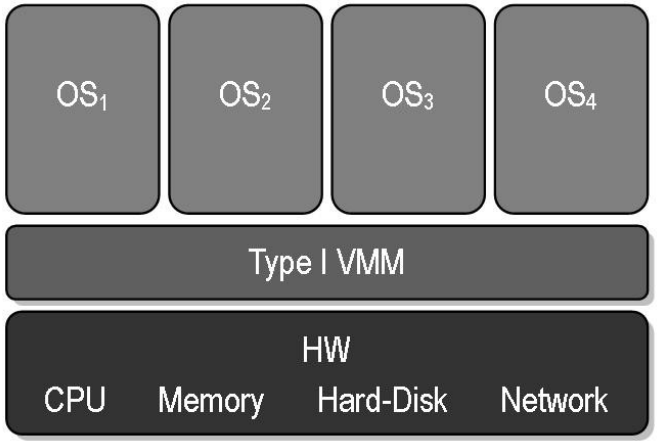
- Strictly speaking: Linux is a Kernel no OS!
- OSes are the distributions
  Popular: Ubuntu, Debian, CentOS, Fedora, Arch
- Leading OS for Server, HPCs but also embedded devices
- Originally for x86, ported to many architectures

Virtual Machines

“A software computer”

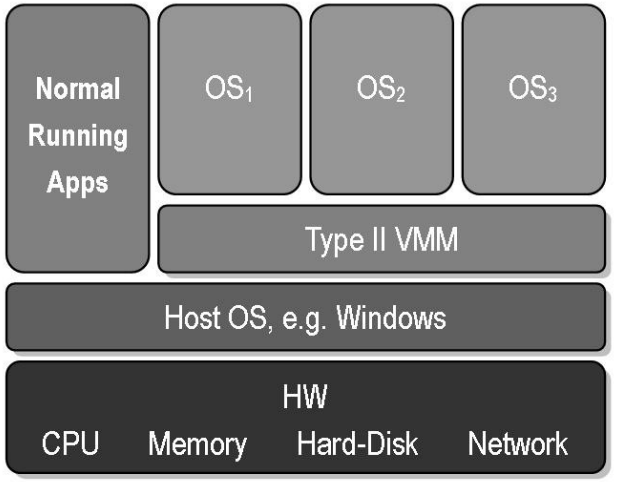
Virtualisation is a key technology in IT

- Flexible abstraction layer
- Resources can be separated
- Virtualisation goes nowadays way beyond Virtual Machine:
  There is virtualisation for e.g. storage or network



Virtual Machine Manager (VMM) or Hypervisor

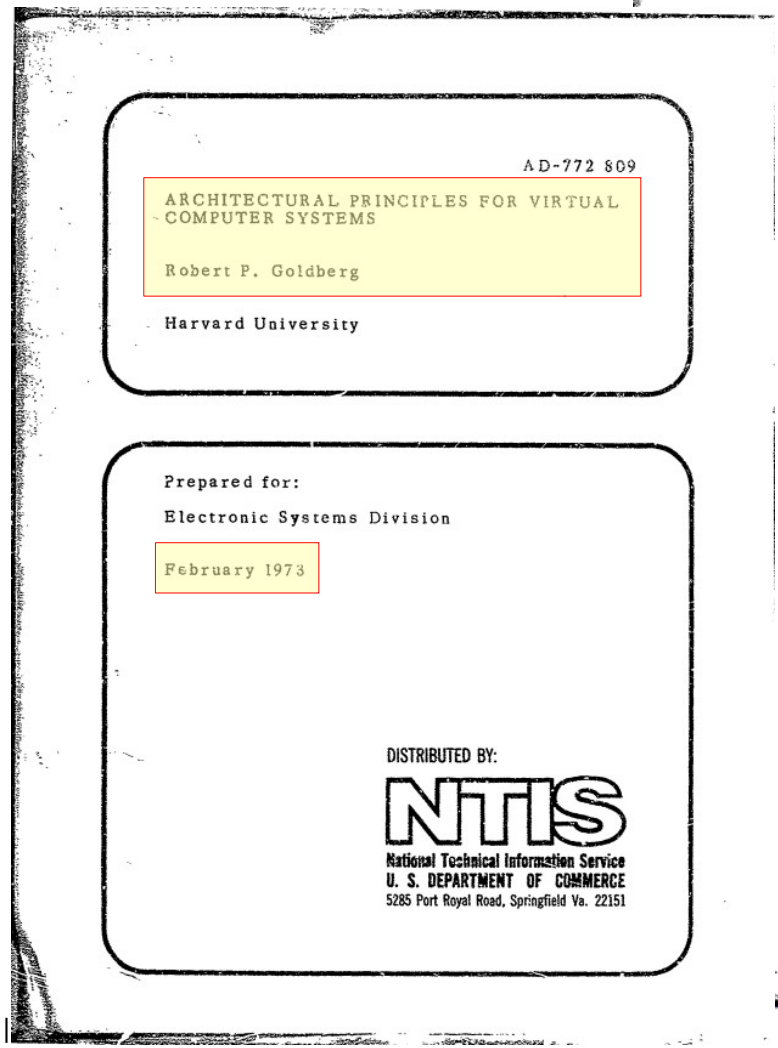
- Controls (and separates) the individual Virtual Machines (VMs)
- Offers the guest OS inside the VM the “virtualised hardware”
  - Can pass-through to real Hardware
  - Can provide e.g. a software version of a processor different from the real HW
  - VMM Type 1: Runs directly on the hardware, no OS
  - VMM Type 2: Runs as an application inside an OS



Virtualisation is (conceptually) not new

Historical Remark

- Concepts invented ~50 years ago
- Early implementations in operation systems for main frames
- Implementations for PCs came much later
  - Limitations in early x86 processors
  - Not need in typical PC environment?



AD-772 809

ARCHITECTURAL PRINCIPLES FOR VIRTUAL COMPUTER SYSTEMS

Robert P. Goldberg

Harvard University

Prepared for:

Electronic Systems Division

February 1973

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

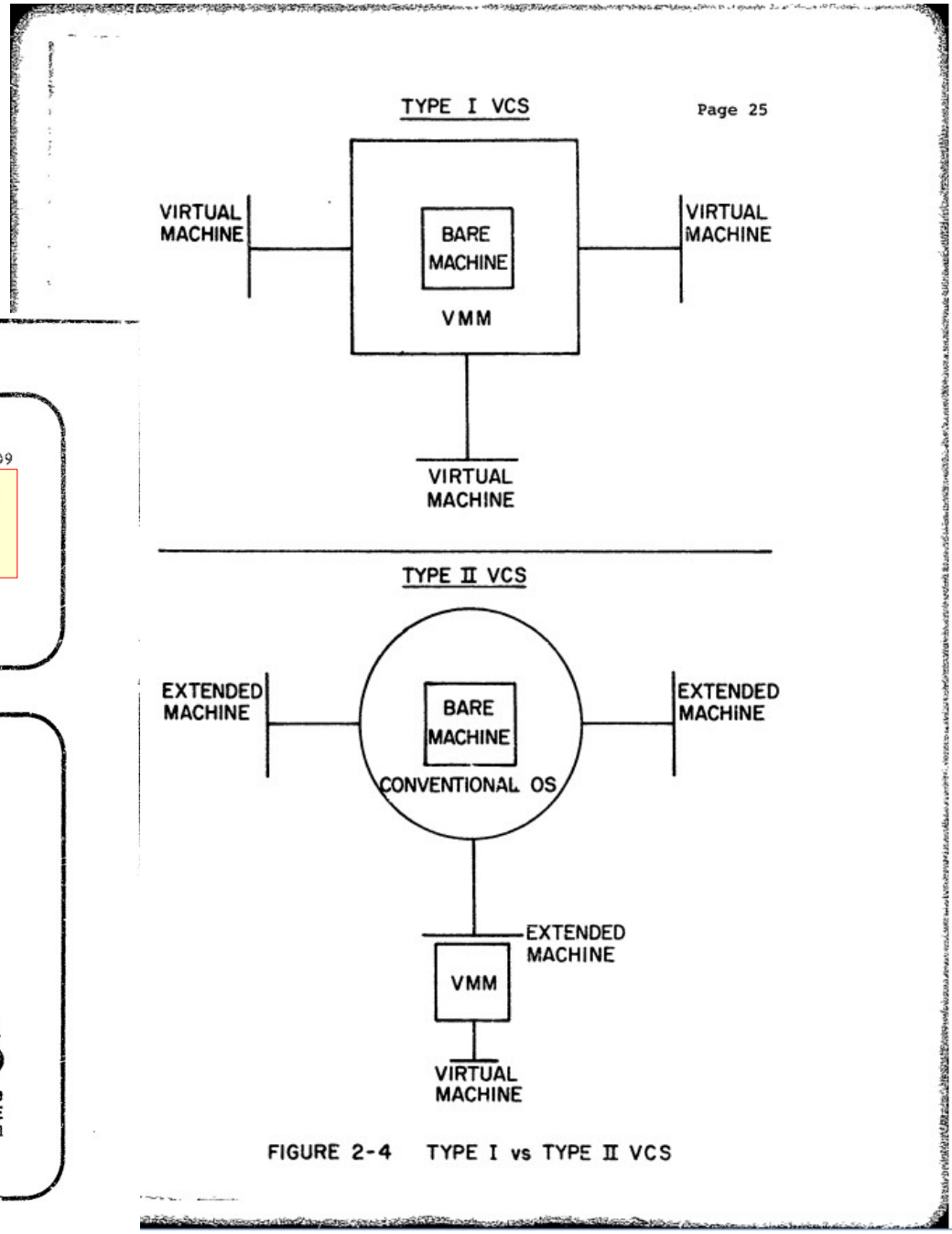


FIGURE 2-4 TYPE I vs TYPE II VCS

Examples for popular Virtualisation Solutions

VMWare vSphere

- Vendor has many other solutions
- VSphere ESXi hypervisor:
  - Type 1VMM running directly on bare metal
  - No OS or any other software running in parallel
  - Certainly a data center solution
  - VMs are controlled from dedicated control application vCenter

KVM/QEMU

- Kernel Virtual Machine (KVM) part of Linux kernel
- QEMU: Quick EMUlation
  - Virtualises full computer
  - Can emulate a processor (e.g. POWER on x86)
  - Typically makes use of KVM for x86 VMs on x86
- Type 2 VMM

Oracle VirtualBox

- Came to Oracle when they bought SUN, who bought VirtualBox before
- Type 2 VMM for x86_64
- Supports many OSs as Hypervisor (VMM)
  - Linux, MacOS, Windows, Solaris
- Popular on Desktops/Laptops

Parallels Desktop

- Popular commercial solution to run VMs on MacOS
  - e.g. run Windows as VM on MacOS
- Type 2 VMM

Containers vs Virtual Machines

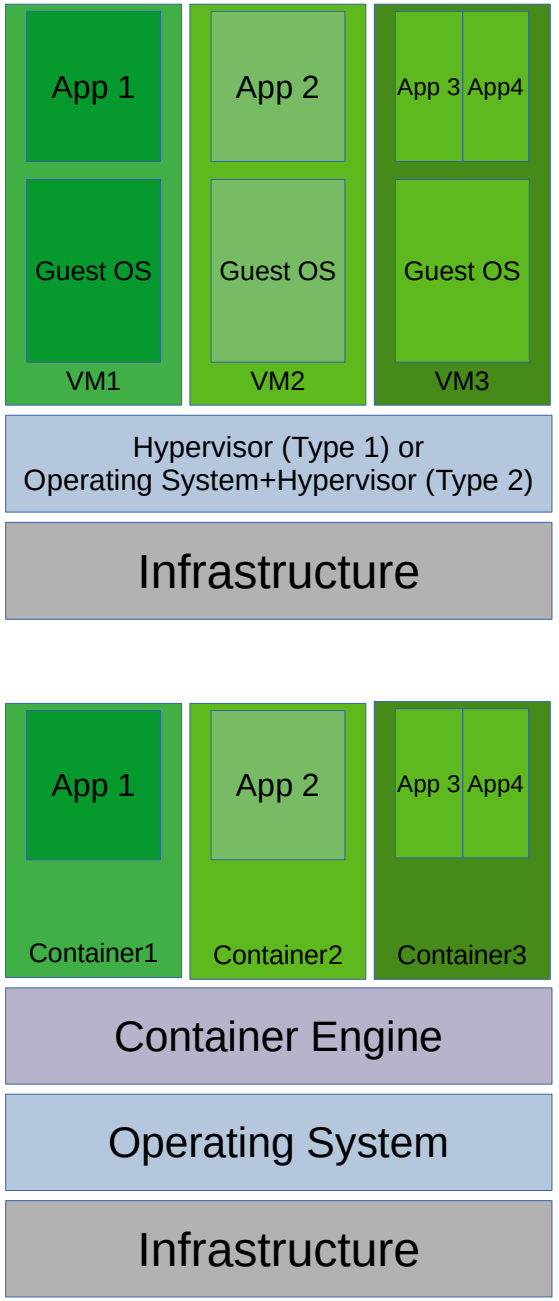
Virtual machine provides a “full computer” in software

Container virtualsation layer is much thinner

- Encapsulation of environment
- Employs kernel of the host operating system
- Container has to *match the host operating system and the host architecture*
  A Linux container only runs on Linux

There are products offering Linux containers on Windows. How about that?

- Behind the scenes works “Windows Subsystem for Linux”
- It’s actually a light weight Virtual Machine

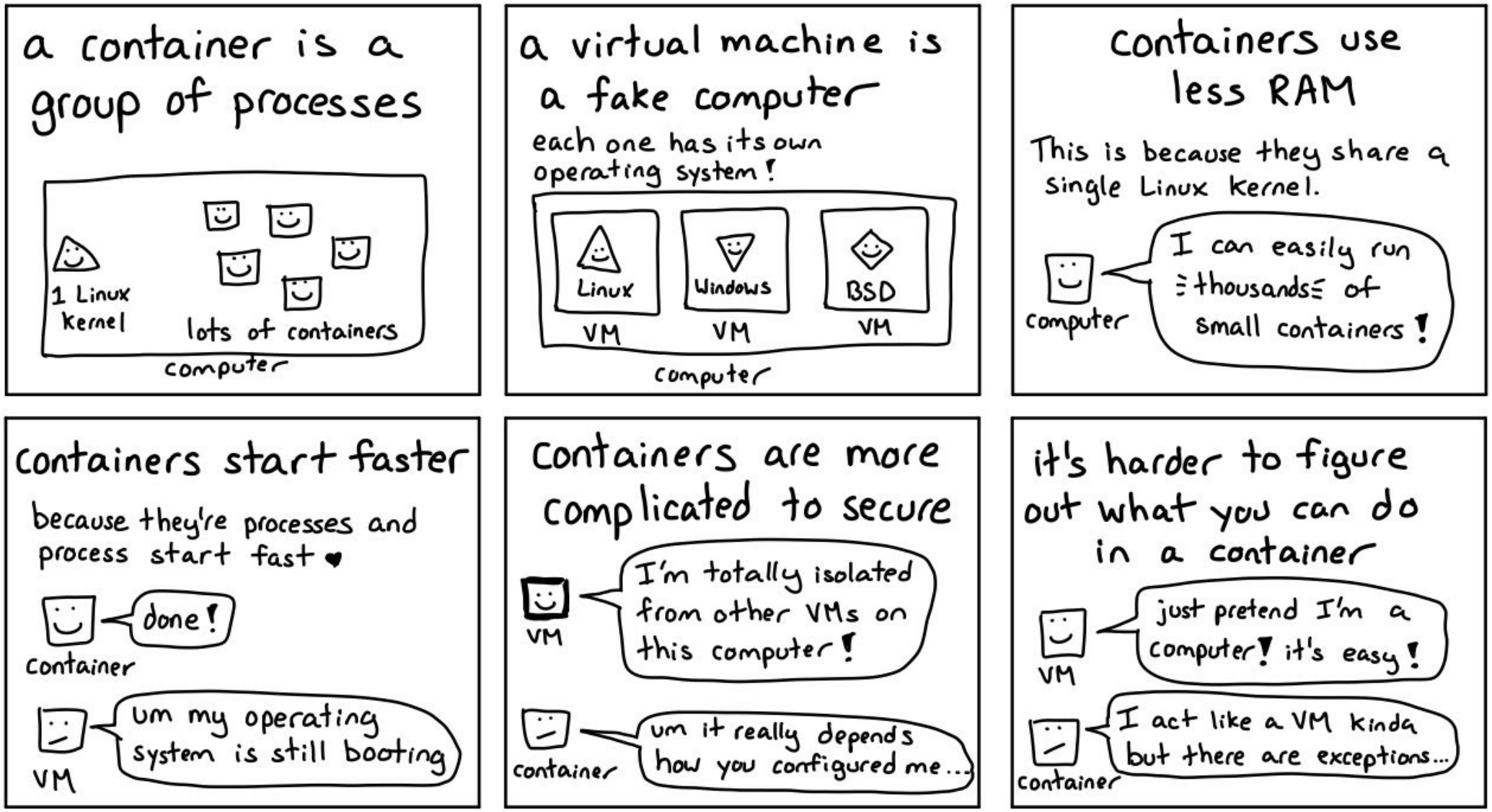


Comparison Virtual Machines vs Container?

Criteria	Virtual Machine	Container
Resources	large	light to medium
Boot-up Time	long (many seconds)	short (less than a second)
Performance	lowish, if more than 1 VM	(almost) native speed
Scaling	difficult	easy
Efficiency	low	high
Portability	medium	high

Julia Evans
@b0rk

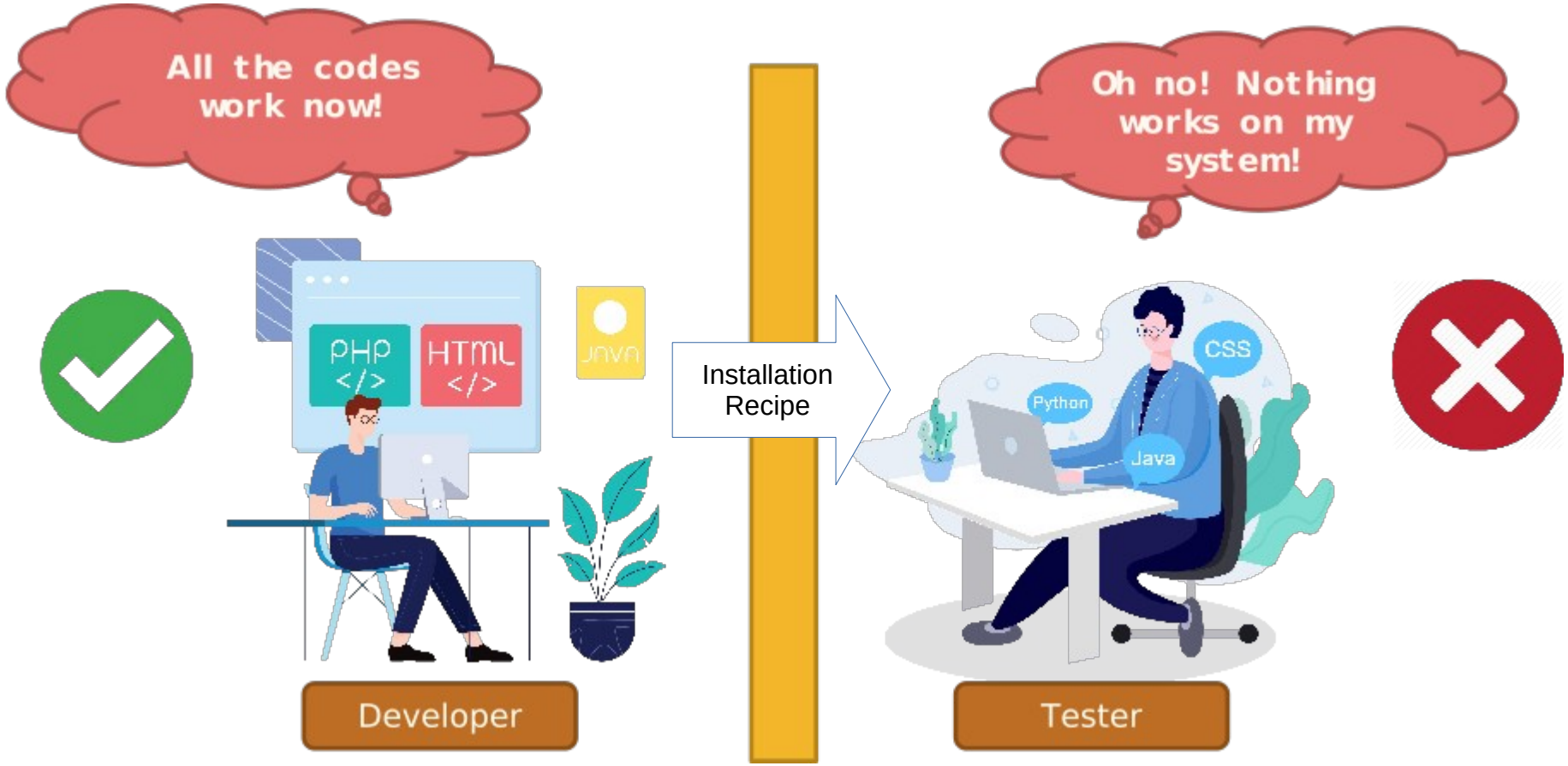
containers vs VMs



https://twitter.com/b0rk/status/1237744128450072578

Why are containers so popular?

“The usual Problem”

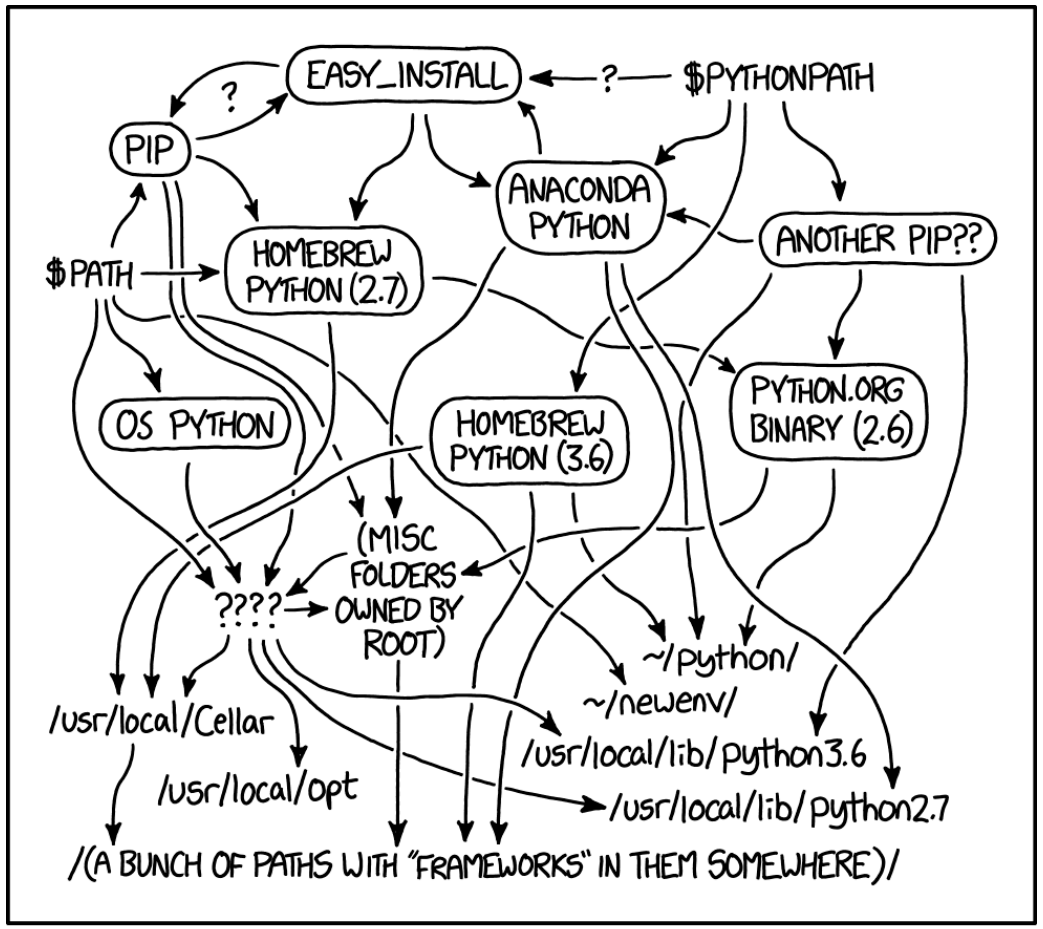


Environments of developer and tester are different: Linux distribution, versions of (other) software

Why are containers so popular?

“Or a typical Python analysis chain”

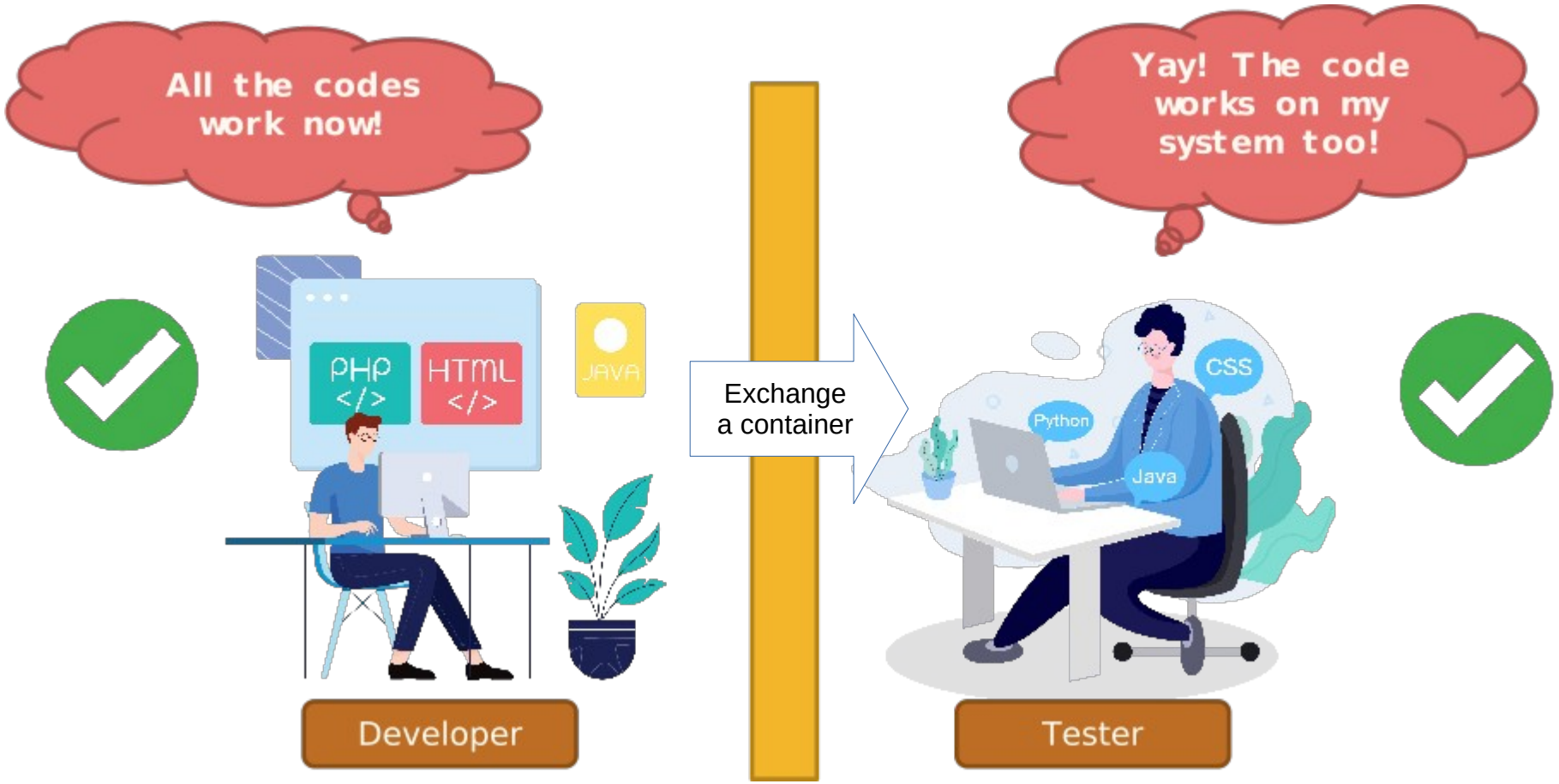
Source: https://xkcd.com/1987/



Hard to port ...

Why are containers so popular?

“A Solution”



Container preserves the proper environment and all dependencies – looks the same “everywhere”

Container Frameworks

Combine low level features into a (easier to use) framework

- Components: chroot, cgroups, kernel namespaces, security frameworks (SELinux or AppAmor), ….
- User tools to build, run and manage containers

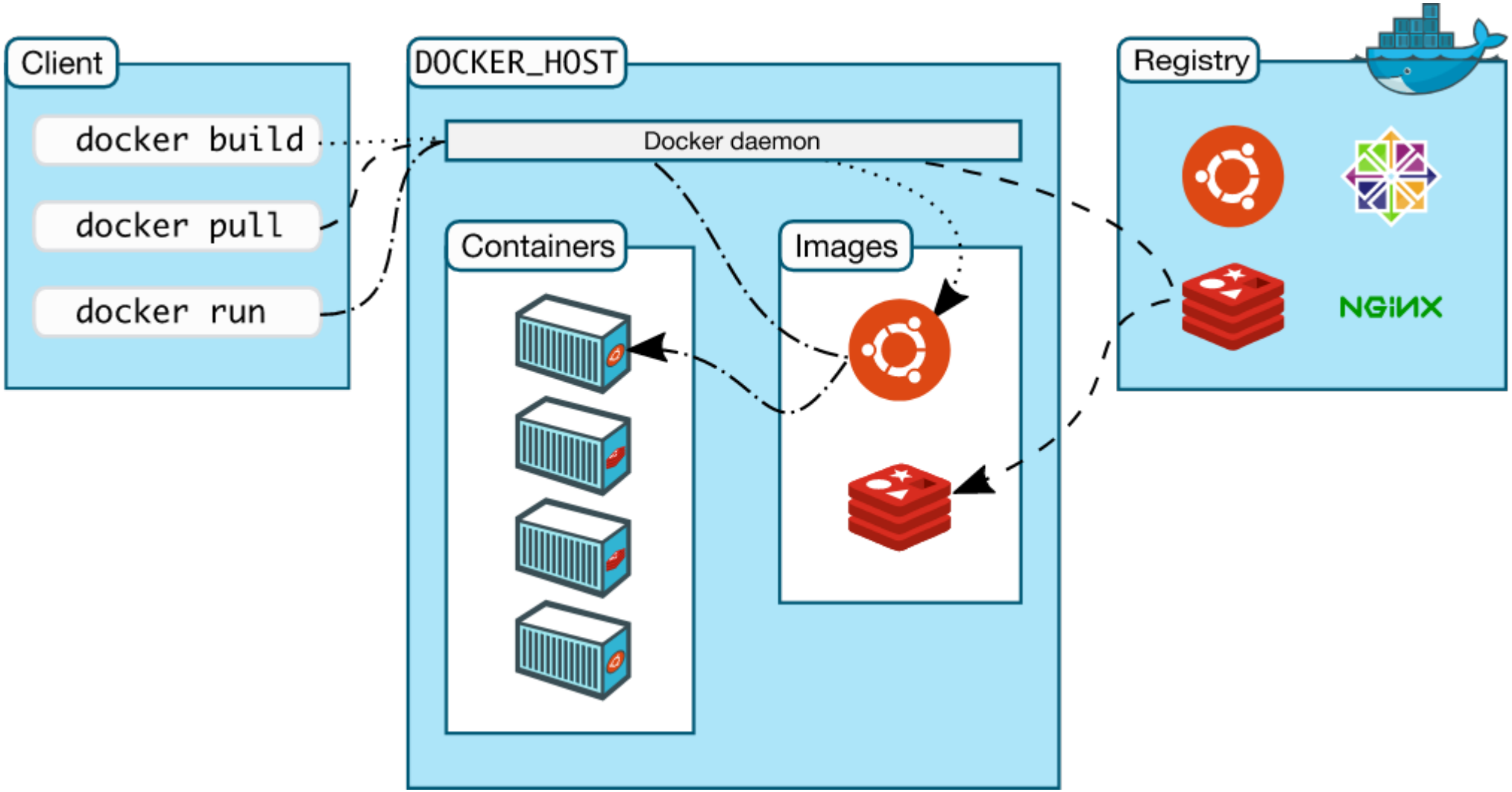
2 prominent examples:

- **Docker**
  - Likely the most common container framework on Linux (even adopted for Windows and MacOS)
  - Was the “break through” for containers (on Linux initially)
  - Frequently used by developers

dotCloud, Inc. https://commons.wikimedia.org/w/index.php?curid=52332268

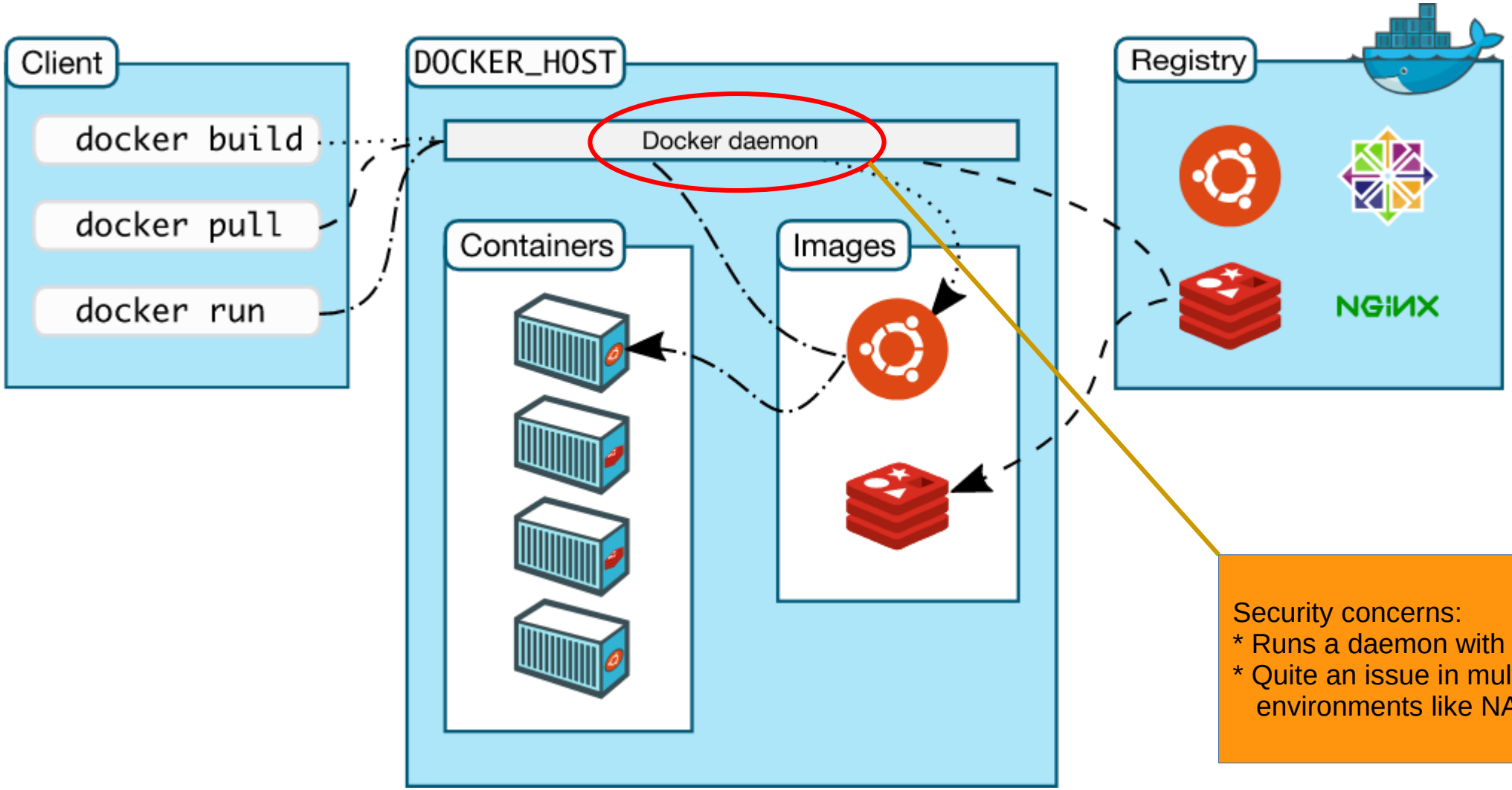
- **Apptainer (formerly known as Singularity)**
  - Very common solution for HPC environments
  - Targeted at security
  - Most adopted container solution at DESY, particularly the NAF

Docker Architecture



From: https://docs.docker.com/get-started/overview/

Docker Architecture



Security concerns:
* Runs a daemon with root privilege
* Quite an issue in multi-user environments like NAF WGS

Modified from: https://docs.docker.com/get-started/overview/

Docker: hello-word

Docker runs as a daemon that needs root privileges

- Commands: `systemctl status docker`, `systemctl start docker`, `systemctl stop docker`

Very instructive hello-world

```
File  Edit  View  Search  Terminal  Help
[wissing@panfilo ~]$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[wissing@panfilo ~]$
```

Which images are on the host:
`docker image ls`
(We see 'hello-world' with a few details listed)

What containers are running:
`docker ps`
(Likely no container listed right now)

If you have a Linux(!) laptop or desktop with root access you might play around with Docker.

Using Docker Desktop on Windows or Macs is forbidden at DESY due to license restrictions!!!

Apptainer

Apptainer is a program – no system daemon (like Docker)

- Images live in directories, typically of the user creating them
- An image can be a single “SIF file” or unpacked (as sandbox) in a sub-directory
- Apptainer can build images from Docker registries or from its own definition files

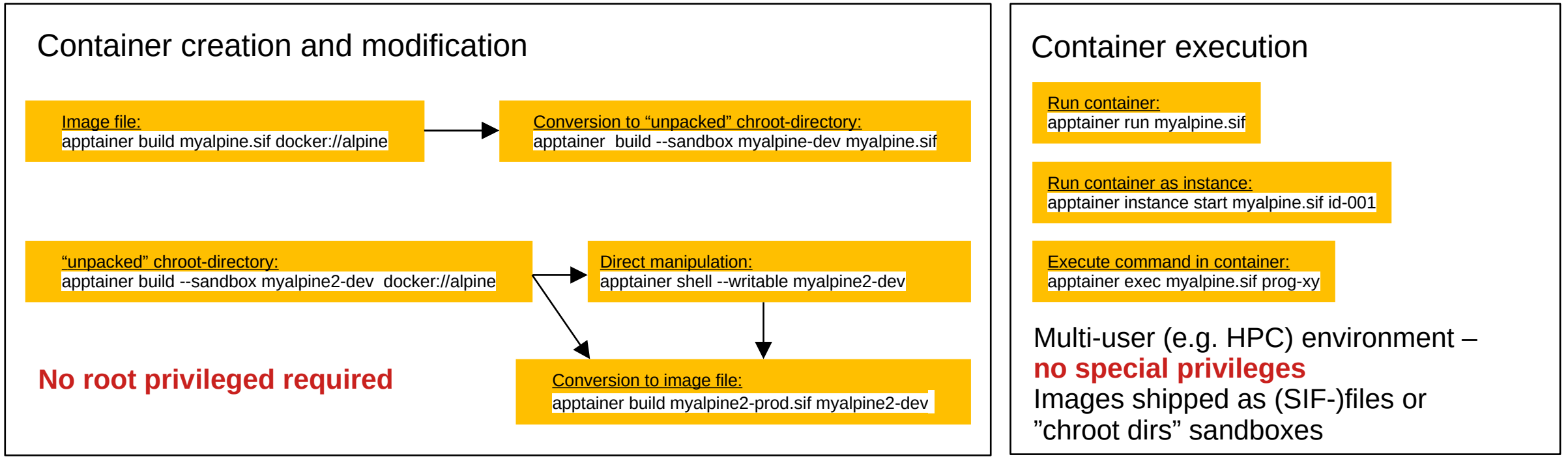


Figure inspired (and updated) from https://doi.org/10.1371/journal.pone.0177459

Apptainer: lolcow saying hello world

Apptainer is a program – no system daemon (like Docker)

- Images live in directories, typically of the user creating them
- Hint regarding disk space, when $HOME is of limited capacity (e.g. AFS home dir at DESY)

  `export APPTAINER_CACHEDIR=<Some-path-with-more-space>/apptainer/cache` (e.g. DUST on the NAF)

```
File  Edit  View  Search  Terminal  Help
[wissing@naf-cms24]~/Grid/NAF/Sustainable-Workshop/Container-WS_2025% apptainer
exec docker://ghcr.io/apptainer/lolcow cowsay "Hello world"
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
Copying blob 5ca731fc36c2 done   |
Copying blob 16ec32c2132b done   |
Copying config fd0daa4d89 done   |
Writing manifest to image destination
2025/03/23 13:08:35  info unpack layer: sha256:16ec32c2132b43494832a05f2b02f7a82
2479f8250c173d0ab27b3de78b2f058
2025/03/23 13:08:36  info unpack layer: sha256:5ca731fc36c28789c5ddc3216563e8bfc
a2ab3ea10347e07554ebba1c953242e
INFO:    Creating SIF file...
 _____________
< Hello world >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
[wissing@naf-cms24]~/Grid/NAF/Sustainable-Workshop/Container-WS_2025%
```

```
File  Edit  View  Search  Terminal  Help
[wissing@naf-cms24]~/Grid/NAF/Sustainable-Workshop/Container-WS_2025% apptainer
exec docker://ghcr.io/apptainer/lolcow cowsay "Hello world again"
INFO:    Using cached SIF image
 ___________________
< Hello world again >
 -------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
[wissing@naf-cms24]~/Grid/NAF/Sustainable-Workshop/Container-WS_2025%
```

Apptainer: Self built container

Usually building starts from a thin base-OS container

- Modify container “interactively”
  Note: There are limitations, Apptainer’s fakeroot feature not always works in all images, so installations via apt-get (Debian or Ubuntu based) or dnf (Alma, Redhat or Fedora based) commands may not work
- Build container from a recipe file

Pull the lolcow container from the Github container registry:
```
apptainer pull docker://ghcr.io/apptainer/lolcow
```

Start a shell in the container:
```
apptainer shell lolcow_latest.sif
```

Start a command in the container:
```
apptainer exec lolcow_latest.sif cowsay “Playing with containers”
```

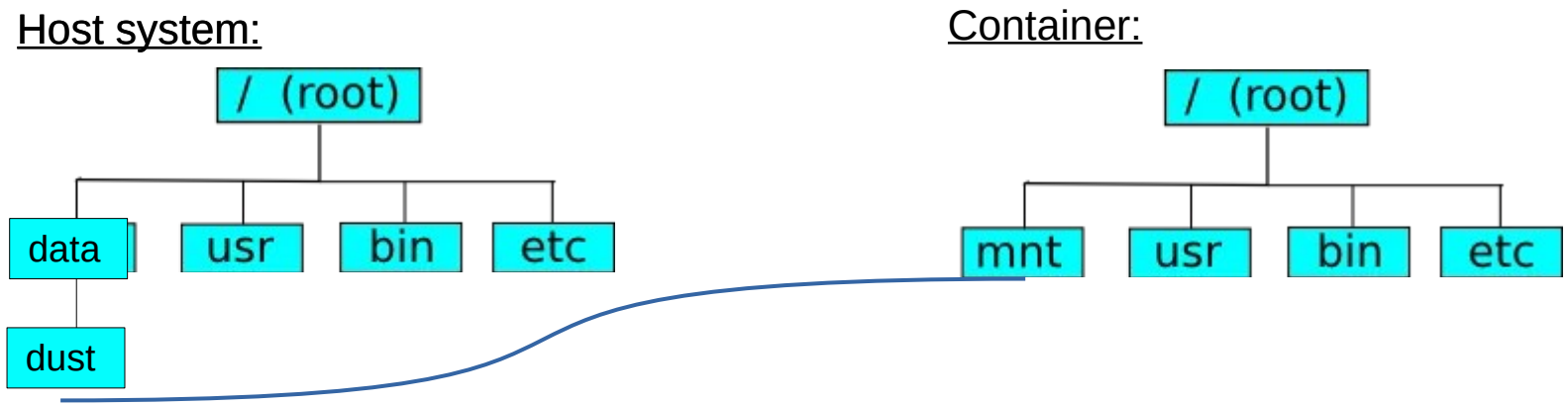
Convert the SIF (single file image) to an unpacked ‘sandbox”:
```
apptainer build --sandbox my_lolcow lolcow_latest.sif
```

Start the unpacked container in writable mode to make modifications (--no-home is need, because /afs is missing in the container):
```
apptainer shell --no-home --writable my_lolcow
```
Inside the container add /afs: `mkdir /afs` (afterwards the –no-home is not needed when starting with –writable)

Convert back unpacked ‘sandbox” to a SIF imalge:
```
apptainer build my_own_lowcow.sif my_lolcow
```

Apptainer: Bindmounts

Essentially mounts a directory of the into the container



Bind mount /data/dust of the host to /mnt in the container:

```
apptainer exec -B /data/dust:/mnt my_own_lolcow.sif ls /mnt
```

shows

```
group  user
```

'-B' *X*:*Y*: Bindmount
X outside container
Y inside container

Automatic bindmounts in the NAF

- /cvmfs
- $HOME
- $PWD
- For other mounts one needs to care

Apptainer: Tetris to go for the linux console

Derive from a thin Linux distro “Alpine”

- Alpine is only a few MB
- A few lines of definition file
  - Bootstrap from Alpine Docker image
  - Add micro-tetris package
  - Add a start script

tetris.def:

```
Bootstrap: docker
From: alpine
Stage: build

%post
   apk add micro-tetris
   touch /etc/localtime

%runscript
   echo "Starting Tetris"
   for s in 5 4 3 2 1; do
     echo -n "$s "
     sleep 1
   done
   /usr/bin/tetris
```

Build image from definition file:

```
apptainer build --ignore-fakeroot-command my_tetris.sif tetris.def
```

Run container image:

```
apptainer run my_tetris.sif
```

It’s only a few MB (copy to any Linux box with apptainer and play...)

Wrap Up

Containers are an efficient and lightweight virtualisation approach

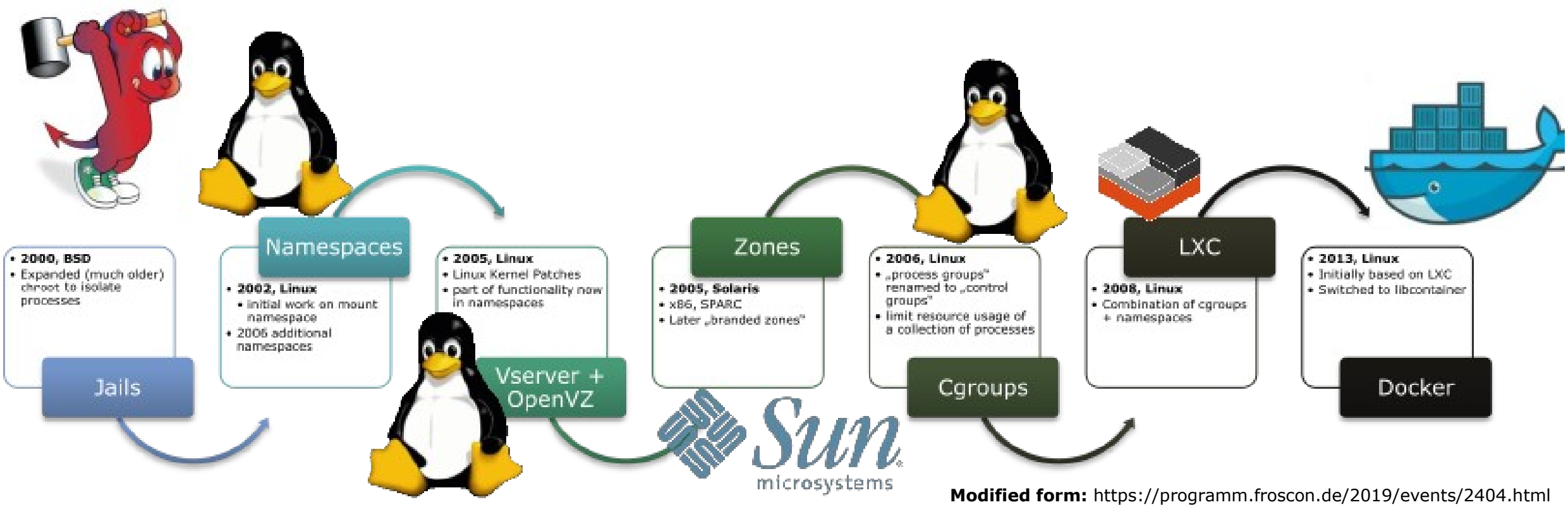
- Realized employing recent isolation features of the Linux kernel
- Widely used in DevOps and scientific computing

Looked a bit into two commonly used Container Engines

- Docker & Apptainer
- Many things can be achieved using either of them

Additional Material

Evolution of Technology for OS-level Virtualisation



Modified form: https://programm.froscon.de/2019/events/2404.html

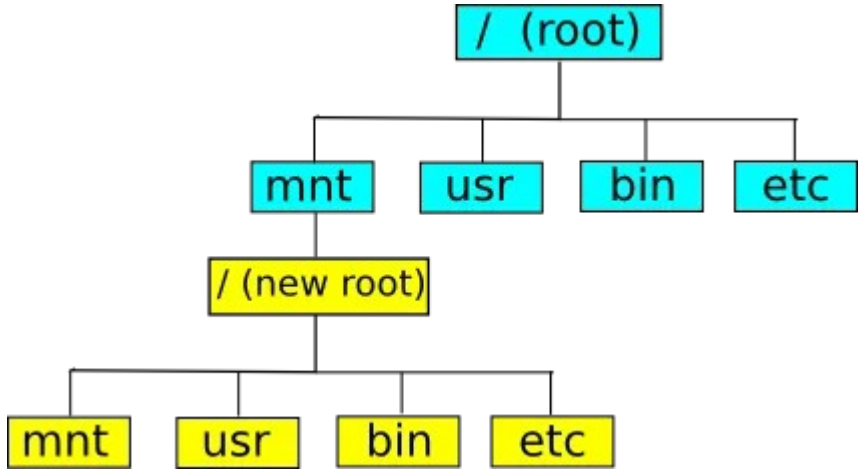
Very Simple “Container”: chroot

CHange ROOT

- Introduce new root entry in existing directory tree
- *mkdir ~/mnt*
  cd ~/mnt
  tar -xzvf /tmp/minimal_Debian.tar.gz

  # Usual directory tree
  cat /etc/redhat-release

  CentOS Linux release 7.9.2009 (Core)
- # Switch into the chroot-environment
  sudo chroot /mnt/
  cat /etc/debian_version
  10.11

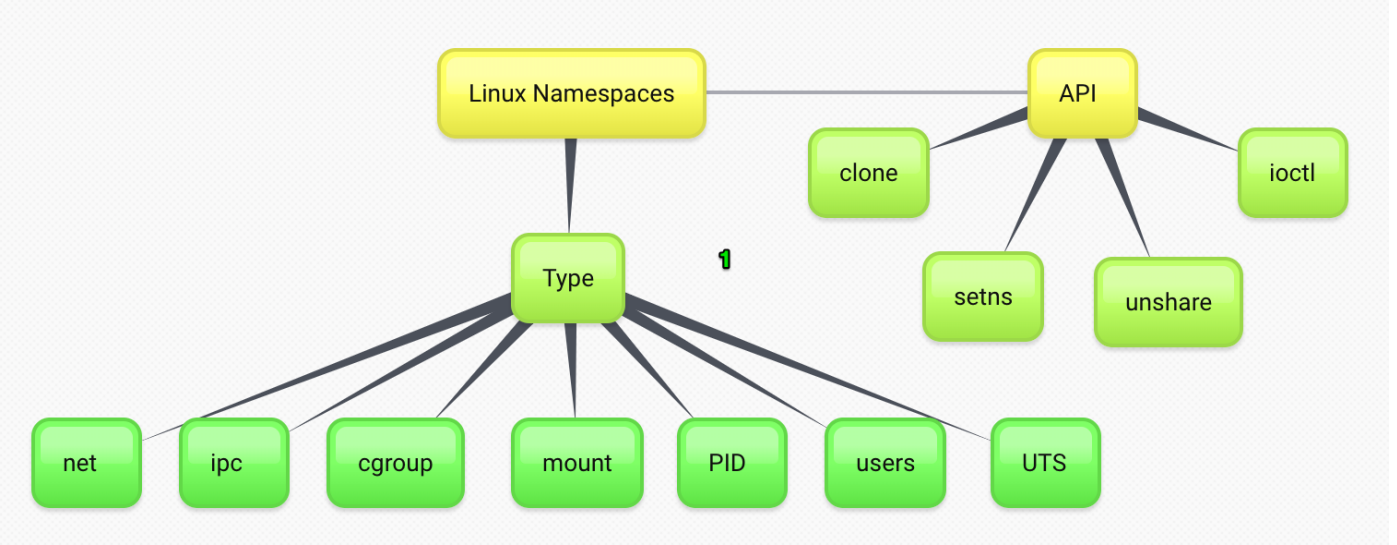


chroot isolates only on filesystem level!
In the chroot environment you still see and
and even kill other processes launched outsite of the chroot-environment

You might want to play with it using the Hands-On VM

Linux – namespaces

- Isolation feature in modern Linux Kernels
  - Mount (mnt)
  - Process ID (pid)
  - Network (net)
  - Interprocess Communication (ipc)
  - UTS (UNiX time sharing)
  - User ID (user)
  - And more



https://8gwifi.org/docs/linux-namespace.jsp

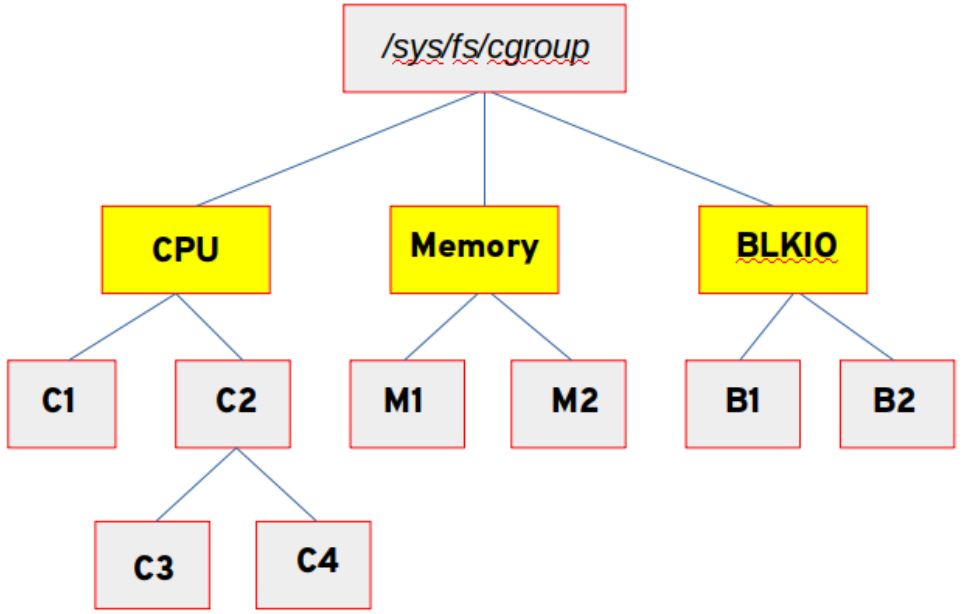
- Example

```
tail -f /tmp/mytestfile.txt
# Get PID of this process e.g. ps aux | grep tail
# Plain chroot via unshare (no isolation)
sudo chroot ~/fake_root /bin/bash
# killing  process from above works
# Now with isolation
unshare --mount --uts --ipc --net --pid --fork --user --map-root-user chroot ~/fake_root/ /bin/bash
# The above PID cannot be killed!
```

Linux – cgroups

Control groups

- Linux kernel feature to limit resource access or consumption:
  - Cpu: Limits or priority for processes to use CPU
  - Cpuset: Which core(s) can be utilized
  - Blkio: Limits block I/O usage of e.g. disks
  - Memory: Memory usage
  - net_prio: Controls network throughput
  - Find the full list: ls /sys/fs/cgroups
- Process (or groups of processes) can get assignment of certain resources



https://www.redhat.com/ja/blog/world-domination-cgroups-rhel-8-welcome-cgroups-v2

Docker: (Simple) self built container - Example

Build own container from a “Dockerfile” (description of an image)

- Simple example: Webserver based on Alpine Linux
  - Caddy: Lightweight but rather featured webserver
  - Alpine Linux: Very small Linux “distribution” – very good base for containers – size is MBs not GBs

‘alpine’ image from hub.docker.com

Execute commands to add packages

Install conf file from local dir into container

Command to run, when container starts

Set **image** name

Dockerfile (no name extension):

```
FROM alpine
RUN apk add --no-cache bash caddy

COPY Caddyfile /etc/caddy/Caddyfile

CMD ["/usr/sbin/caddy", "run", "--config", "/etc/caddy/Caddyfile"]
```

Build the image from Dockerfile (Dockerfile & Caddyfile must exist):

```
docker build -t mycaddy .
```

Run a container from mycaddy image:

```
docker run -d -p 80:80 --name mycaddy01 -v $PWD/html-docs:/usr/share/caddy:z mycaddy
```

Needed for system with SELinux security, e.g. CentOS 7

‘-d’: detach, start container
no shell in container

‘-p X:Y’: X TCP port outside
Y port inside container
(see Caddyfile)

Name of **container**

‘-v’ A:B: Volume
A outside container
B inside container

Docker: Example continued

Work a bit with the container

- Webserver should be reachable (even from outside of the VM): http://192.168.56.104/ (IP might vary)
- Some more commands

Show running container:
```
docker ps
```

Inspect logging:
```
docker logs mycaddy01
```

Connect a shell to your container:
```
docker exec -it mycaddy01
```
In that shell: `ps` shows only caddy
`whoami` shows that you are **root** (by default)

‘-it’: interactive and terminal attached.
Could be used with run instead of ‘-d’

Stop the container:
```
docker kill mycaddy01
```

Fully remove the container:
```
docker container rm mycaddy01
```

Contact

DESY. Deutsches
Elektronen-Synchrotron

Christoph Wissing
FH – CMS group
christoph.wissing@desy.de

www.desy.de