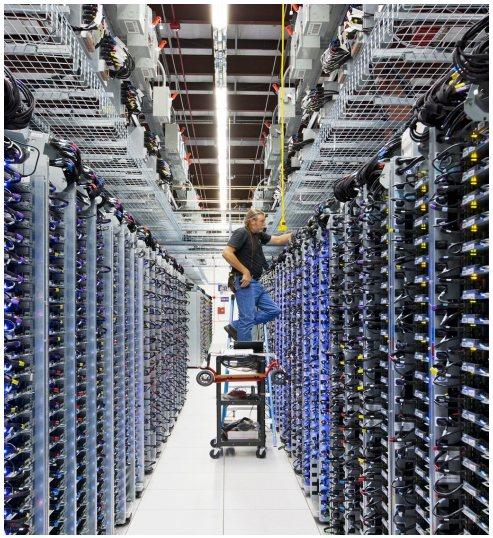
Administration IT — Virtualization





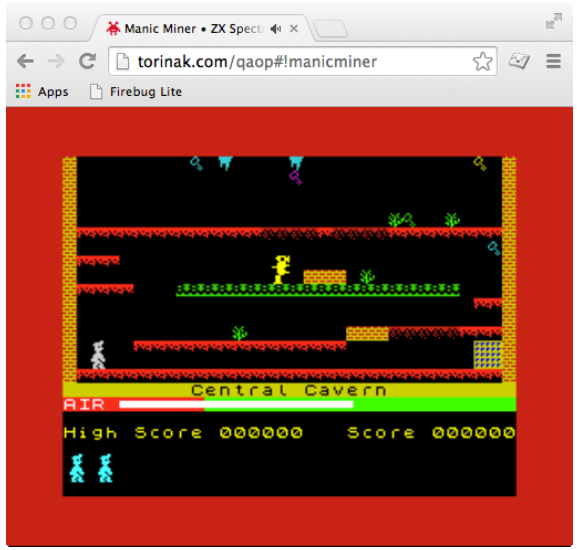

Academic year 2015/16

Virtualization
Introduction

- *Virtualization* in computer science: a virtualized system is a *mapping* of its interface, and all resources visible through that interface, to the interface and resources of a real system.
  - Often several virtualized systems are mapped to the same real system.
- Examples
  - Virtual memory managed by the operating system
  - Virtual LANs offered by network switches
  - Programming language runtimes: Java Virtual Machine, .NET Common Language Runtime
  - Hardware virtualization
  - Storage virtualization
  - Network virtualization

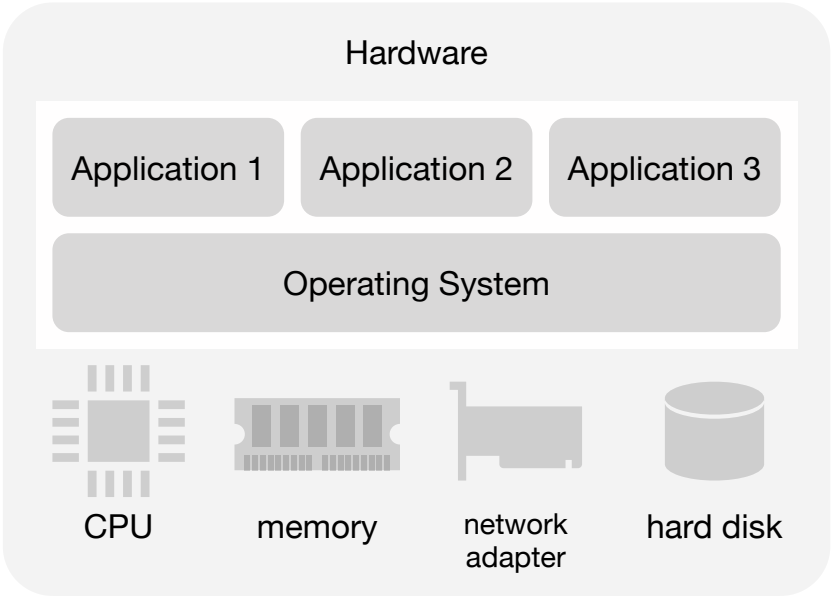
Virtualization
Virtualization vs. Emulation

- Emulation: the process of implementing the interface and functionality of one system on a system having a different interface and functionality.
  - Examples: Game console emulator, terminal emulator, microprocessor emulator
- Virtualization can be seen as a special case of Emulation
  - In virtualization only a small part of the functionality is emulated, most of it is provided by the original component.
  - Many virtualization techniques were derived from emulation techniques.
- There are three basic emulation approaches:
  - Interpretation
    - Emulator interprets one instruction at a time.
  - Static binary translation
    - Emulator translates a block of instructions at a time and optimizes it for repeated executions.
  - Dynamic binary translation
    - Hybrid approach which combines interpretation with static binary translation.

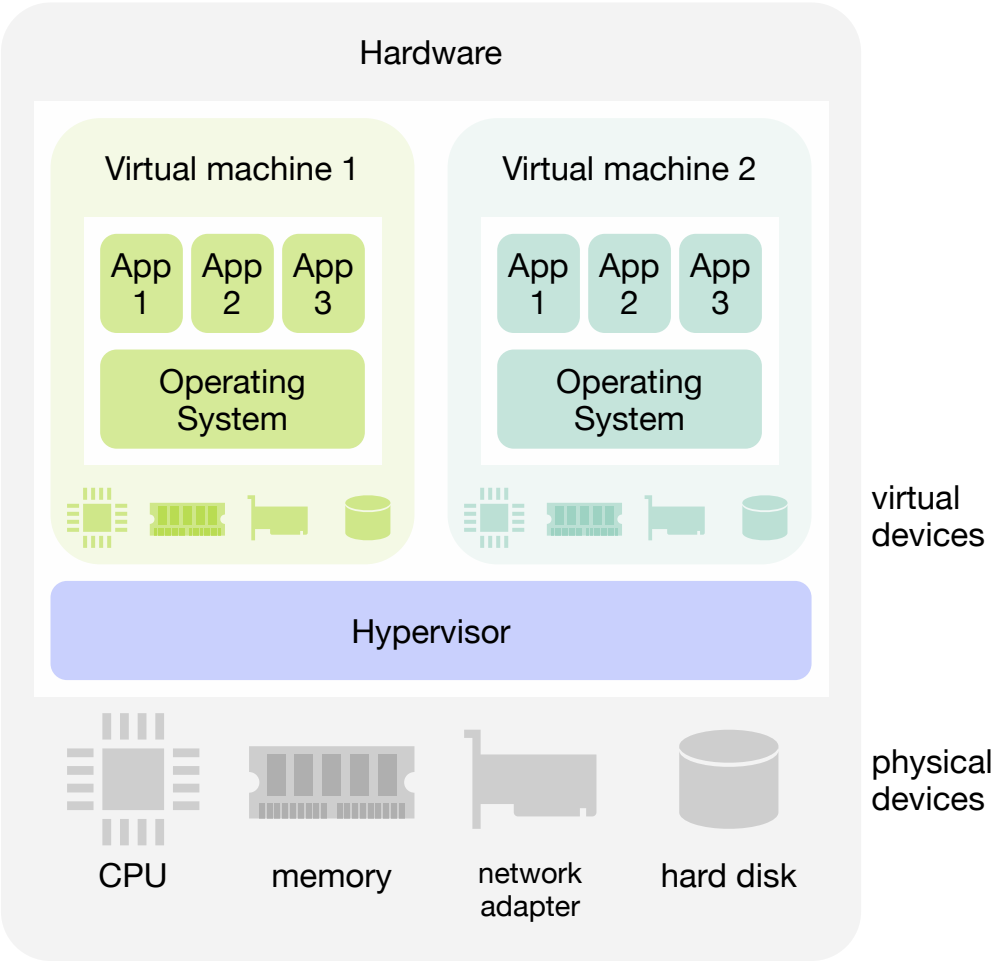


Hardware virtualization

Basic approach



Computer without hardware virtualization



Computer with hardware virtualization
(type 1, bare-metal hypervisor)

Hardware virtualization

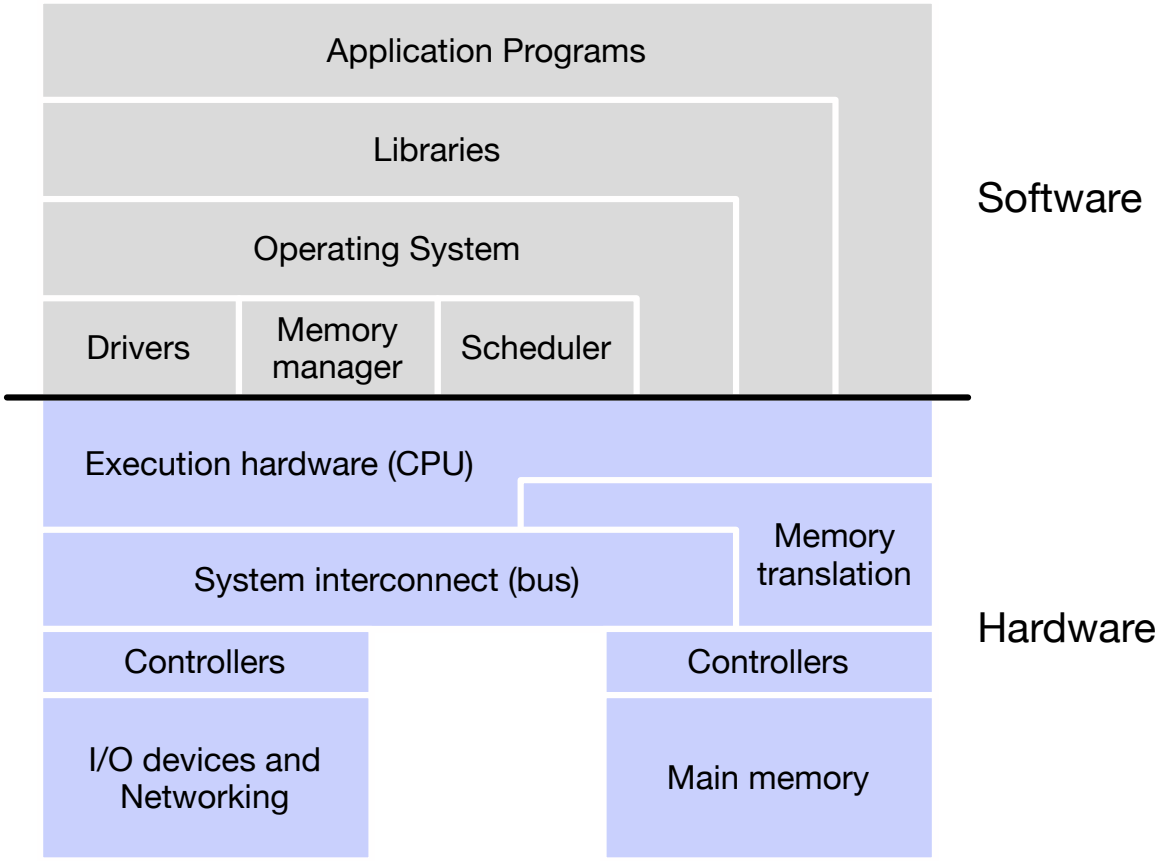
Hypervisors

- When the concept of virtualization is applied to an entire computer one speaks of **hardware virtualization** or **platform virtualization**.
  - The virtualized computer is called **Virtual Machine** (VM).
- A VM is implemented by adding a layer of software to a real machine so as to support the desired VM's architecture.
  - This layer of software is often referred to as **virtual machine monitor** (VMM).
  - Early VMMs were implemented in firmware.
  - Today, VMMs are often implemented as co-designed firmware-software layer, referred to as the **hypervisor**.

Hardware virtualization

Basic approach

- User of the virtualized system: The operating system and the applications running on it
- Interface of virtualized system: CPU instruction set, memory and and controller registers
- Accessible resources of virtualized system: CPU, MMU, buses, I/O devices, ...



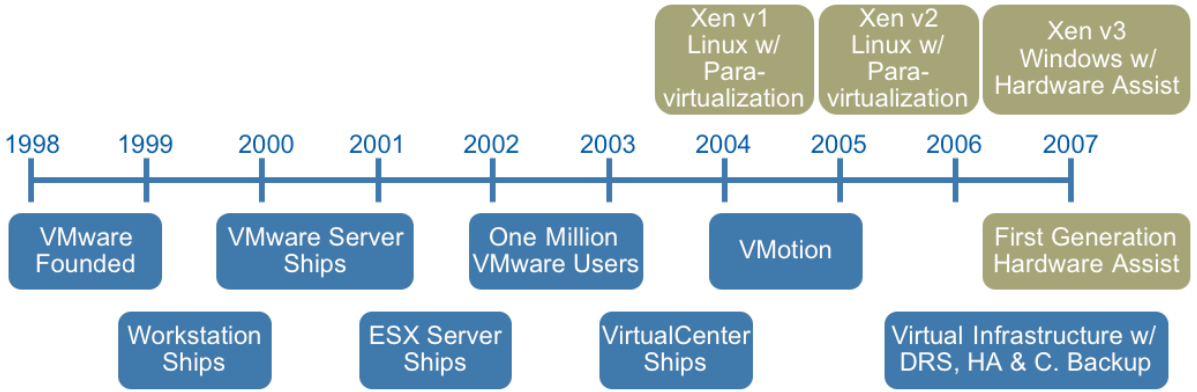
Hardware virtualization
Properties

- **Binary compatibility**
  - The guest behaves like a real machine to the operating system and the applications running within.
- **Interposition**
  - All guest actions go through the virtualizing software which can inspect, modify, and deny operations.
- **Isolation**
  - Programs running in one VM cannot access data in another VM.
    - Software isolation
    - Fault isolation
  - A VM with high load cannot affect the performance of another VM.
    - Performance isolation (accomplished through scheduling and resource allocation)
- **Encapsulation**
  - All VM state can be captured into a file: the VM image
    - The VM can be manipulated like any other file: transferred, duplicated, deleted
  - Complexity is proportional to virtual HW model and independent of guest software configuration

Hardware virtualization

History

- In 1966, the IBM System/360 Model 67 was the first computer offering virtual machines, on which several operating systems could run.
  - For many decades IBM's mainframes were the only computers to support virtualization.
- In 1998, VMware figured out how to virtualize the x86 platform, once thought to be impossible.
  - Pure software solution, using very clever techniques.
- In 2006, both Intel and AMD introduced hardware virtualization support in their processors.
  - Simpler virtualization software, but little speed benefit.
- Later processor models added memory and network virtualization support.



Hardware virtualization
Uses

- Where does hardware virtualization play an important role today?
  - **Server virtualization:** Inside a company, one physical server is able to run several different applications, where each application requires a different environment (operating system, database, middleware, ...).
    - VMware ESXi, Citrix XenServer, Microsoft Windows Server 2008 Hyper-V, Red Hat Enteprise Virtualization, ...
  - **Cloud computing:** Using hardware virtualization a cloud service provider is able to run several virtual machines on a single server and rent them by the hour to different customers.
    - Amazon Web Services, Microsoft Azure, Google Cloud Services, OpenStack, CloudStack, VMware vSphere, ...
  - **Development VMs:** An application developer recreates on her personal computer the server environment in which the application will run, inside a virtual machine.
    - VMware Workstation / Fusion, Oracle VM VirtualBox, Parallels Desktop, Vagrant, ...
  - **Desktop virtualization:** The "desktop", that is the applications an employee uses every day, do not run on the employee's personal computer but on a server in the company's data center and are centrally managed. The personal computer only provides the display.
    - Citrix XenServer, Microsoft Windows Server 2008 Hyper-V, Red Hat Enterprise Virtualization, VMware vSphere, ...

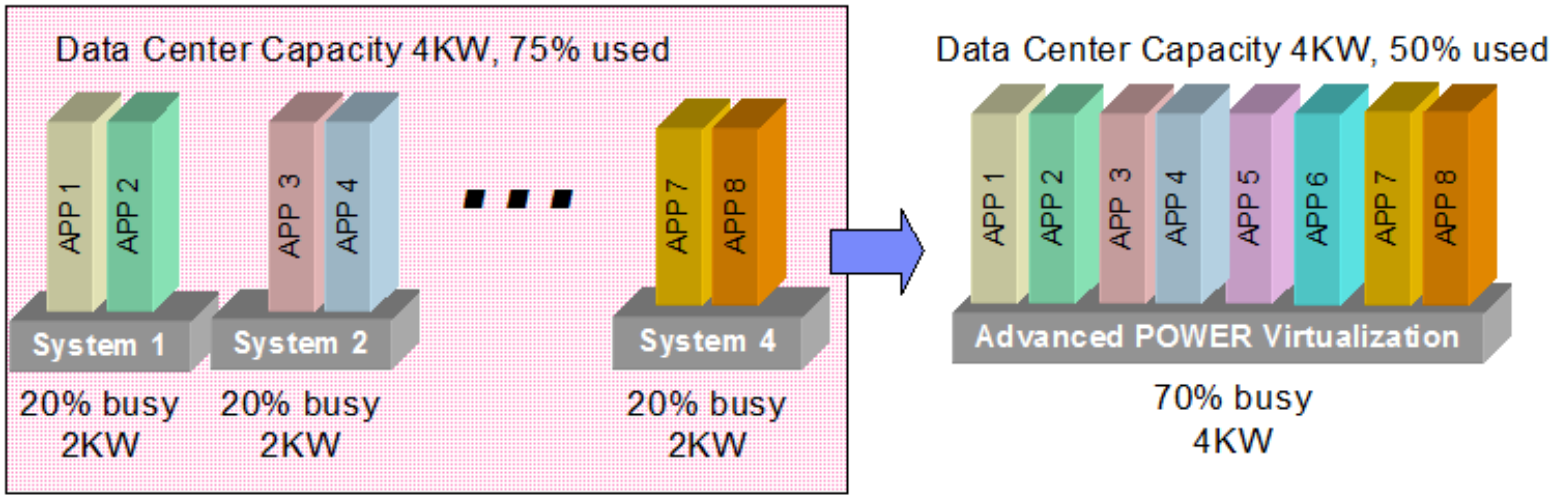
Hardware virtualization

Server virtualization — Benefits

- Why does it make sense for a company to use virtualization on their servers?
  - **Multiple secure environments**
    - A VM provides a sandbox that isolates one system environment from other environments
  - **Failure isolation**
    - Virtualization helps isolate the effects of a failure to the VM where the failure occurred
  - **Mixed-OS environment**
    - A single hardware platform can support multiple operating systems concurrently
  - **Better system utilization**
    - A virtualized system can be (dynamically or statically) re-configured for changing needs
  - **Reduced energy costs**
    - Most non-virtualized servers have low utilization rates. Consolidation on a single server increases utilization.

Hardware virtualization

Server virtualization — Benefits



Total Power 8KW

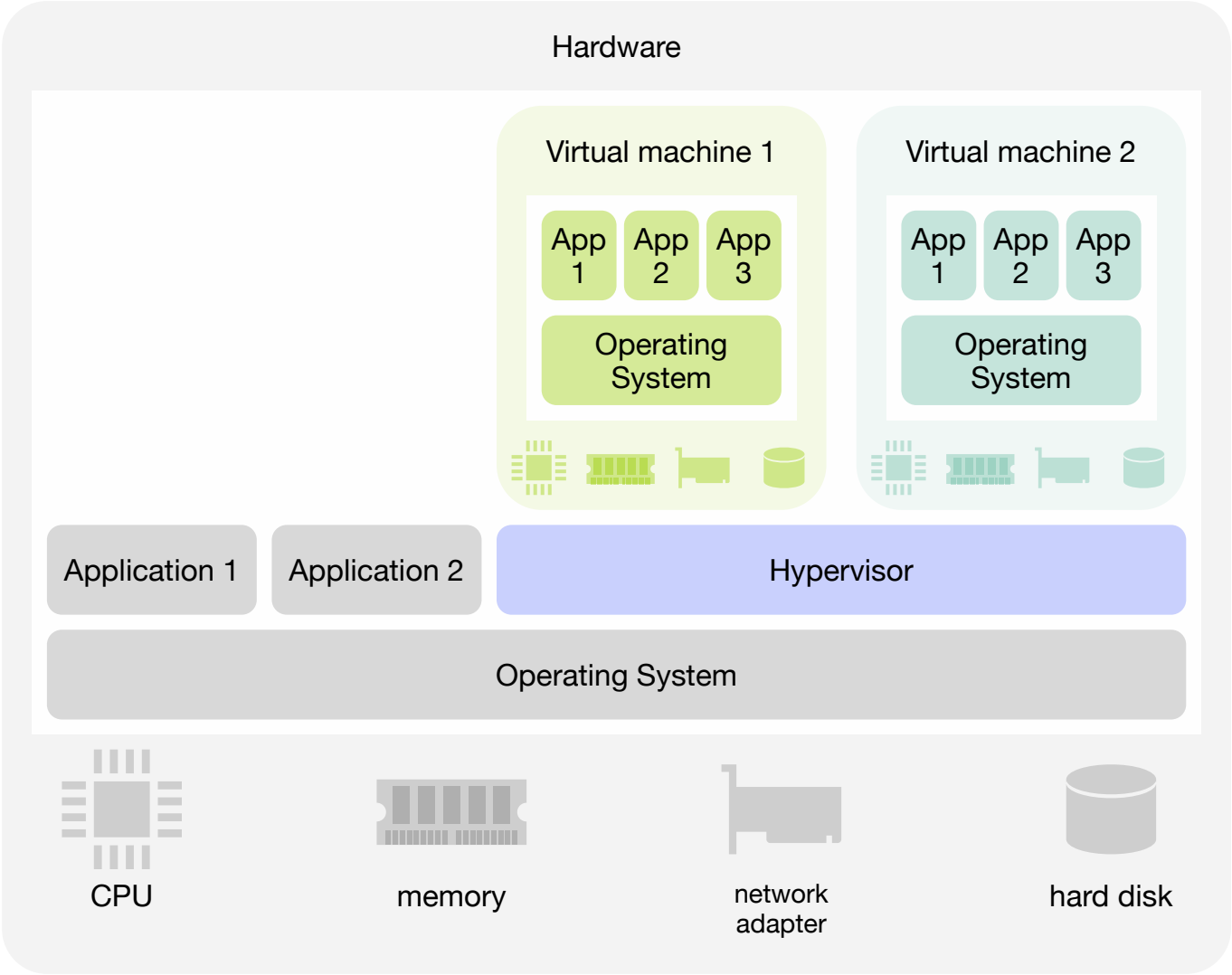
Total Power 4KW

Server consolidation exploiting virtualization is a very effective tool in reducing energy costs

Hardware virtualization

Development VMs

- Developers often need to re-create the environment in which their code will run (for example a PHP web server) on their local workstation.
- A Virtual Machine is a good way to create this environment without interfering with the workstation's software.
- In this case the hypervisor runs on top of the workstation's operating system (type 2 hypervisor).
- Popular products:
  - VirtualBox
  - VMware workstation



Computer with hardware virtualization (type 2, hosted hypervisor)

Hardware virtualization

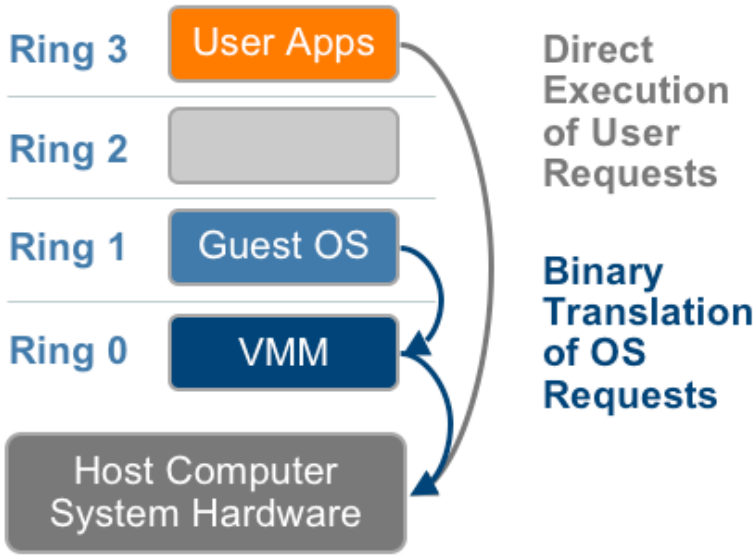
Virtualization techniques

- There are three basic virtualization techniques
  - Full virtualization
  - Para-virtualization
  - Hardware-assisted virtualization

Hardware virtualization

Full virtualization

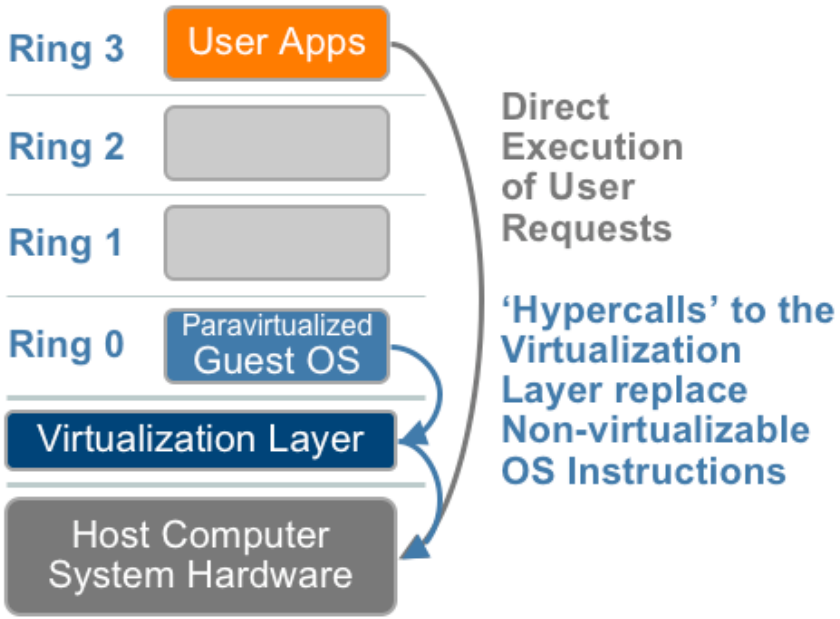
- The guest OS is not aware it is running on a virtualized platform.
- Virtualization uses combination of binary translation (for privileged instructions in the OS) and direct execution (for applications)
  - Performance may suffer from binary translation
- Offers best isolation and security for virtual machines.



Hardware virtualization

Para-virtualization

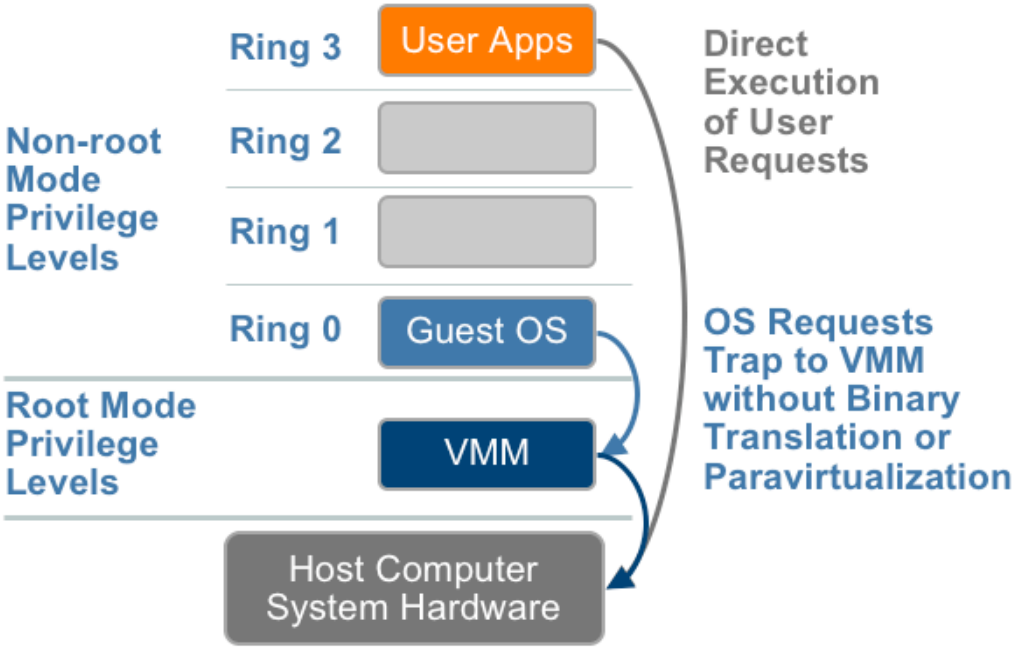
- The guest OS is modified and cooperates with the hypervisor.
- Guest OS is often Open Source software, for example Linux
- Less need for binary translation
- Opportunities for optimization
- Better performance
- Does not work with commercial off-the-shelf OSes



Hardware virtualization

Hardware-assisted virtualization

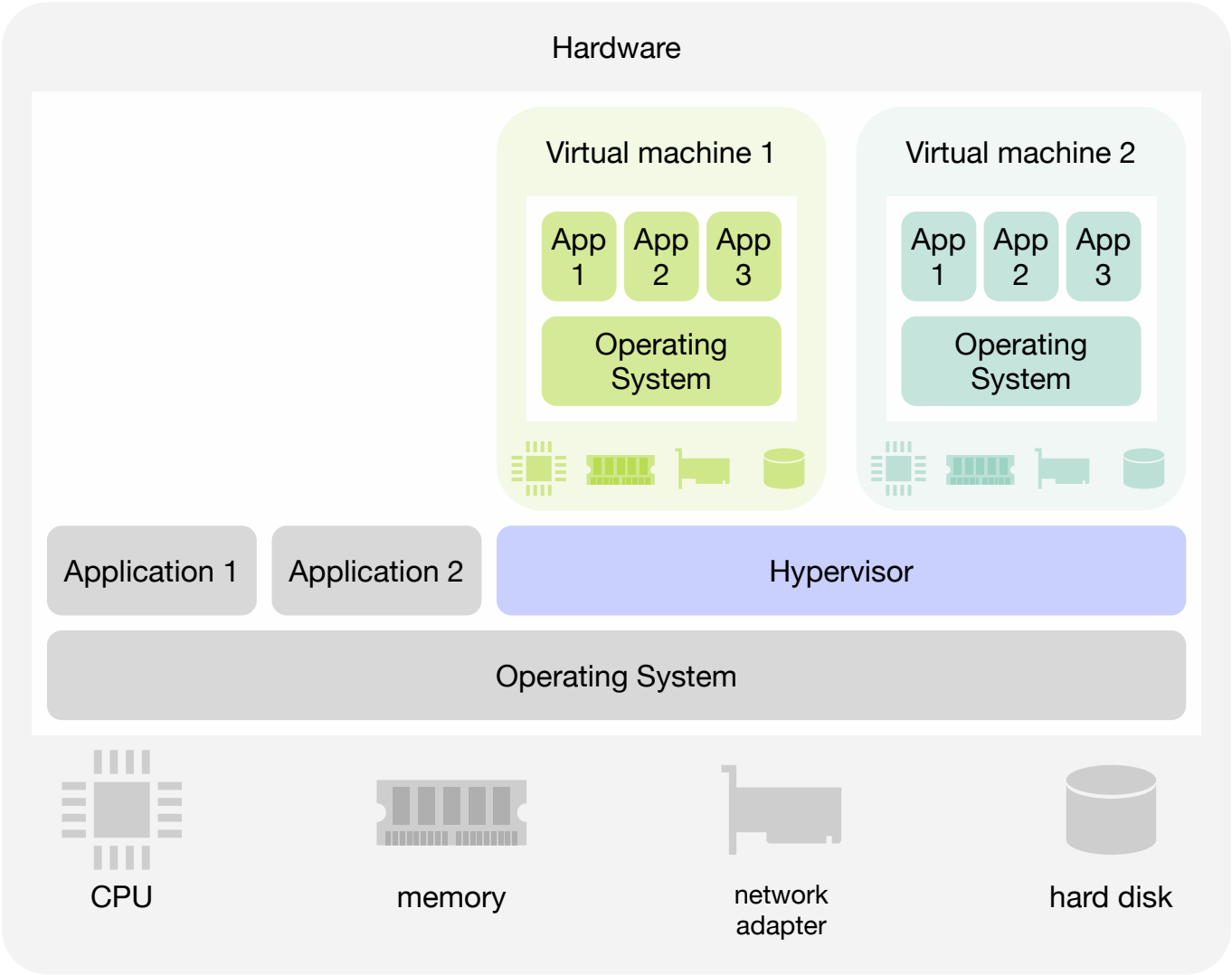
- The CPU supports virtualization with special features
  - New root mode below ring 0 for the hypervisor
  - Virtualized Memory Management Unit (MMU)
  - Virtualized I/O on PCI Express
  - ...
- No need for binary translation or para-virtualized OS
- Best performance



Virtual environments for development

Introduction

- Anne is a web developer. The code she develops will run on a server with the Linux operating system and several pieces of middleware: Apache, MySQL and PHP (LAMP stack). But her development workstation has a different operating system: Mac OS X.
  - To recreate the exact environment in which her application will run, Anne creates a virtual machine that runs Linux and she installs Apache, MySQL and PHP.

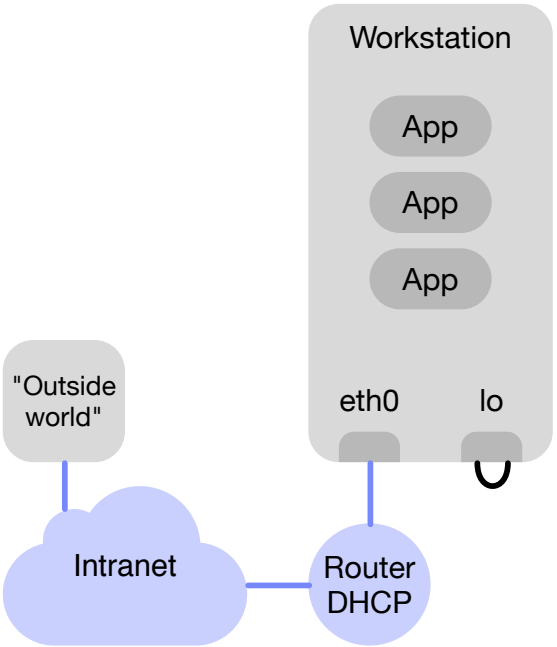


Virtual environments for development
Set-up

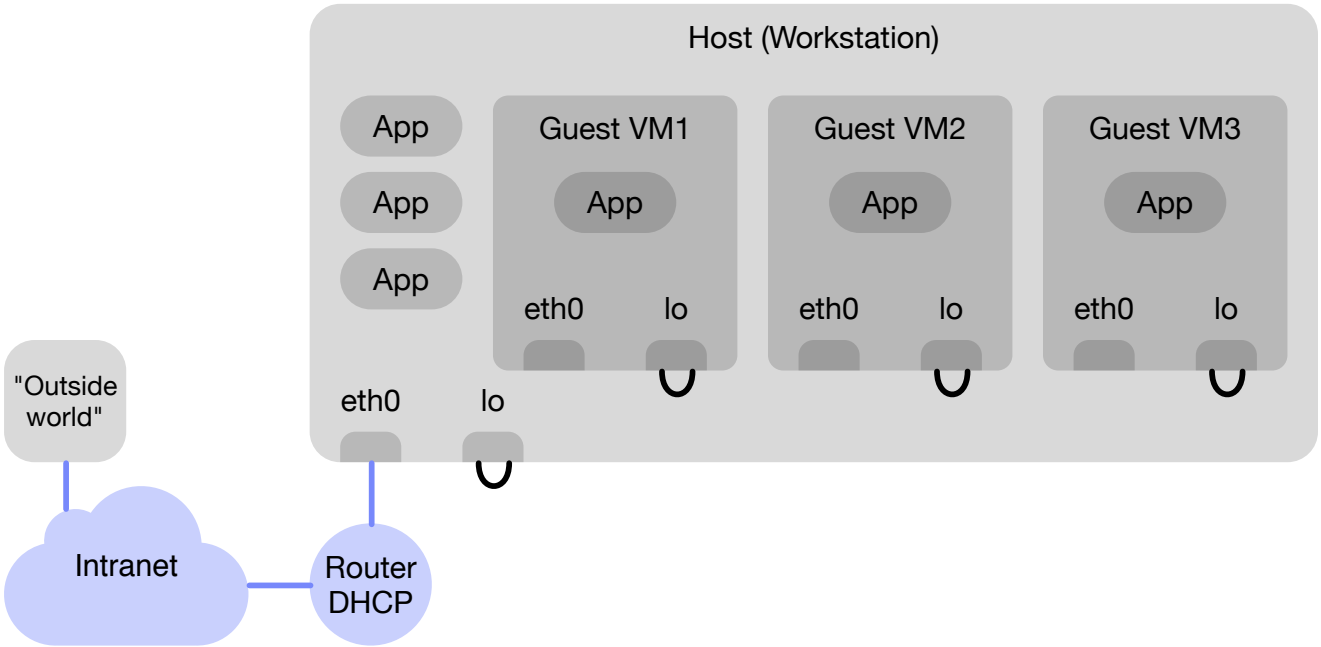
- The set-up of a virtual environment for development usually involves the following steps:
  - Download a VM image that contains the desired operating system
  - Create a new VM, start the VM and log into the VM
  - Install any additional middleware to re-create the server environment
  - Configure networking
    - See following slides
  - Configure file sharing between host and guest
    - Developer wants to use editors and IDEs on host to modify files on guest

Virtual networks
Introduction

Without virtualization



With virtualization



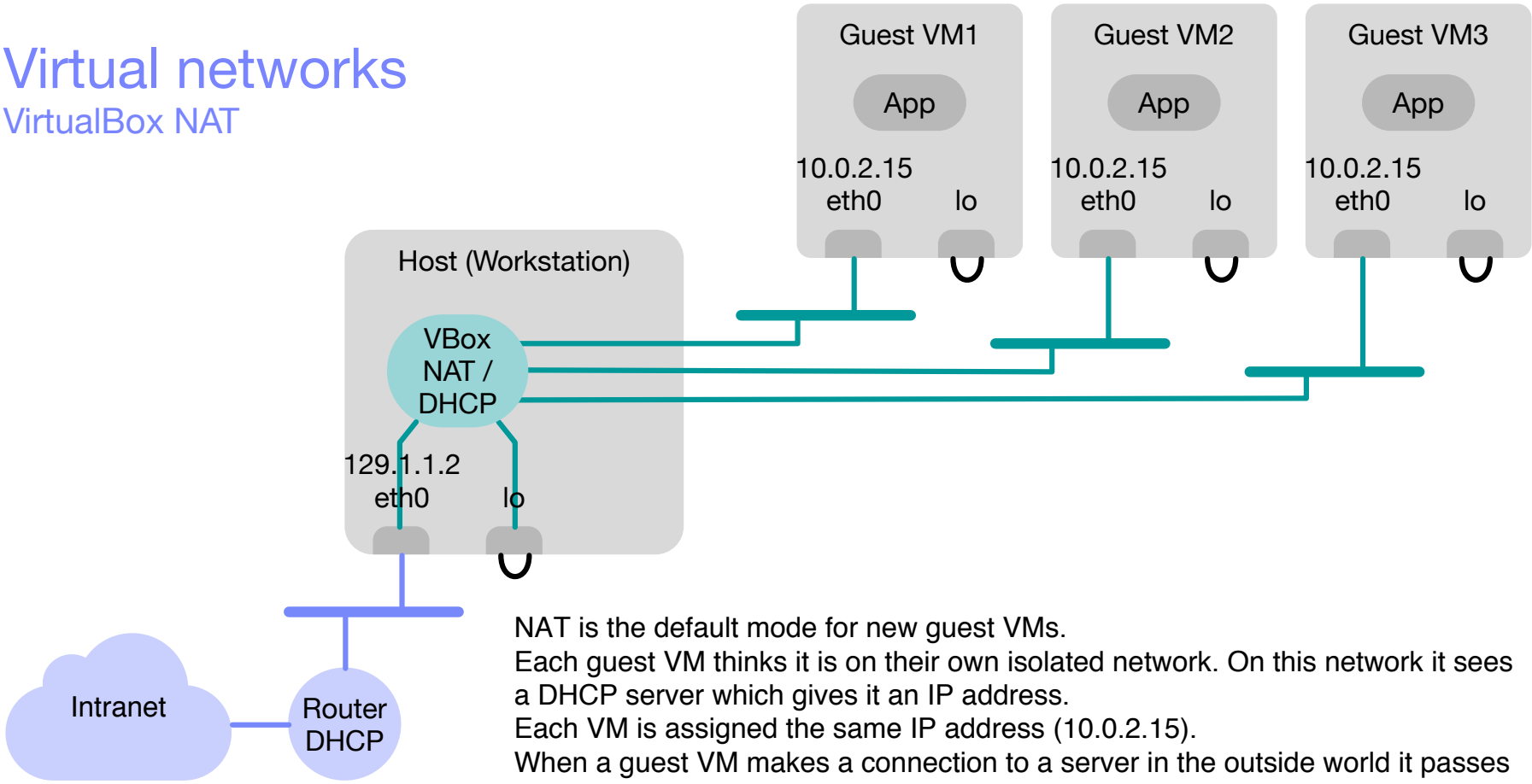
Virtual networks
Requirements

- Depending on the requirements, set-up of networking can become complex. There are essentially four cases to consider:
  - (A) The guest VM needs to access the outside world
    - E.g., to download operating system updates
  - (B) The host needs to access the guest VM
    - Developer wants to log into the guest VM via SSH
    - Developer wants to test the software running in the guest VM
  - (C) The outside world needs to access the guest VM
    - Developer wants to test the software running in the guest VM using tools running on other hosts
  - (D) A guest VM needs to access another guest VM
    - E.g., several components of a distributed system are re-created in guest VMs

- In most cases one wants to avoid interacting with the IT department to obtain additional IP addresses, changes to the routing table, etc.

Virtual networks

VirtualBox NAT



NAT is the default mode for new guest VMs.
Each guest VM thinks it is on their own isolated network. On this network it sees a DHCP server which gives it an IP address.
Each VM is assigned the same IP address (10.0.2.15).
When a guest VM makes a connection to a server in the outside world it passes through a NAT that rewrites the packets to make them appear as though they originated from the host workstation, rather than the guest VM.

⊕ Guest VMs can access servers in the outside world

⊕ When the host workstation connects to another network, nothing needs to be reconfigured

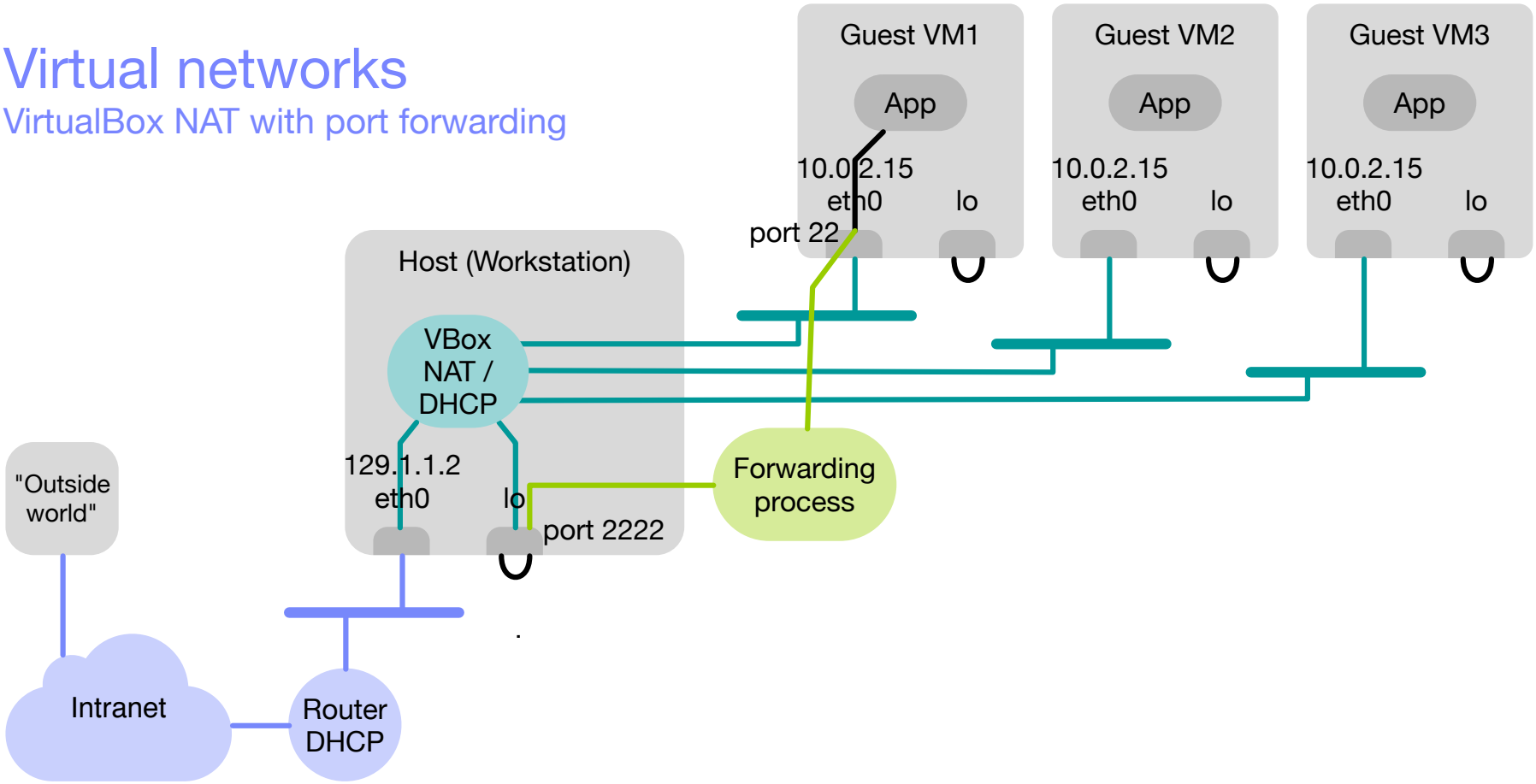
⊖ The outside world cannot (directly) access servers in the guest VMs

⊖ The guest VMs cannot talk to each other

Source: Fat Bloke, "Networking in VirtualBox", https://blogs.oracle.com/fatbloke/entry/networking_in_virtualbox1

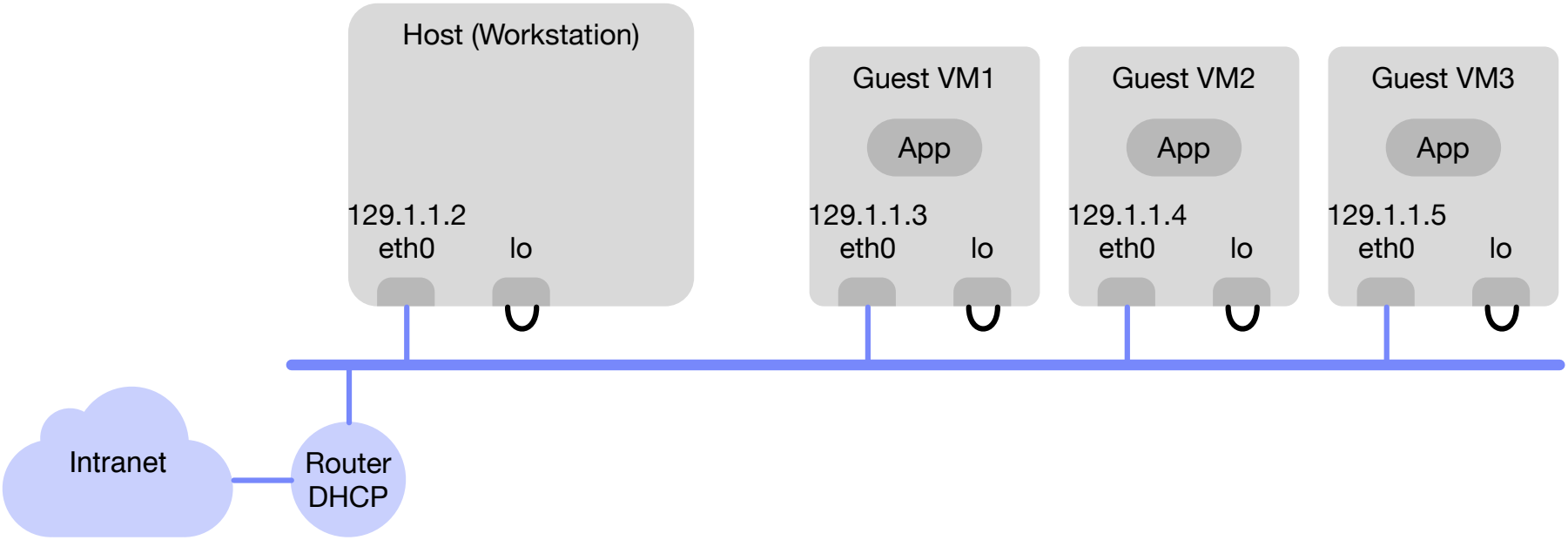
Virtual networks

VirtualBox NAT with port forwarding



Virtual networks

VirtualBox Bridged Adapter



In bridged mode, the virtual NIC of a VM guest is bridged with the physical NIC on the host workstation. The effect of this is that each guest VM has access to the physical network the same way as the host workstation. It can access any service on the network such as DHCP, name lookup and routing information.

⊕ Guest VMs access the outside world exactly like the host workstation

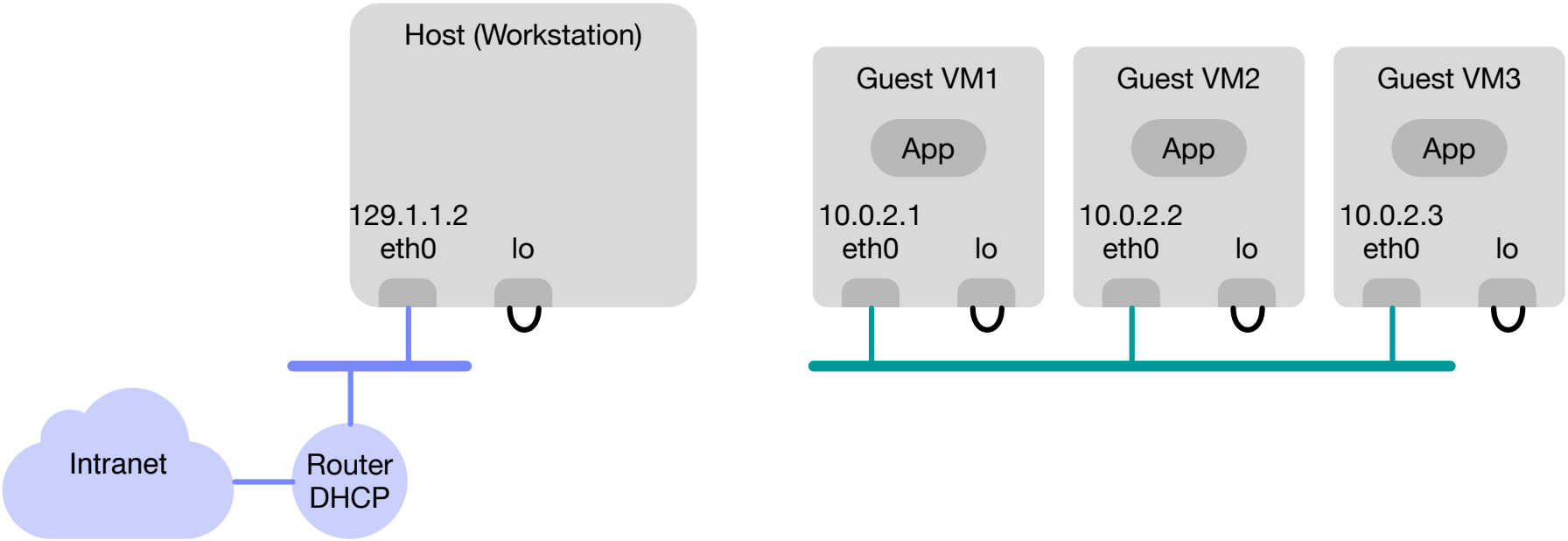
⊕ The outside world sees the guest VMs as directly connected to the network

⊖ The network may quickly run out of IP addresses

⊖ The bridge needs to be reconfigured each time the host workstation jumps networks

Virtual networks

VirtualBox Internal Network



The guest VMs are connected to an internal network that is completely isolated. It is possible to create complex internal networks with VMs that provide their own services to the internal network (e.g., Active Directory, DHCP, etc.). Note that not even the host workstation is member of the internal network.

⊕ Guest VMs can talk to each other

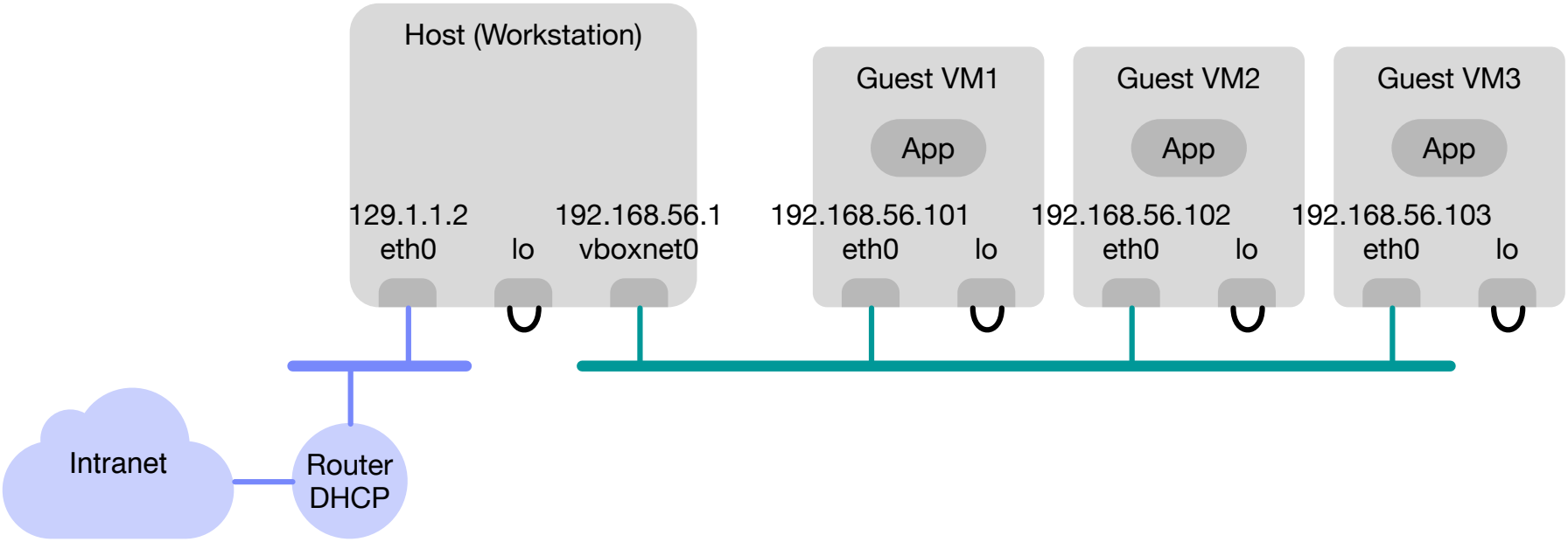
⊕ Guest VMs function even when the host workstation is not connected to any network (e.g. on a plane)

⊖ Host workstation cannot see guest VMs

⊖ Outside world cannot see guest VMs

Virtual networks

VirtualBox Host-only Adapter



Host-only networking is like Internal Networking, but the host workstation is connected to the internal network as well.

⊕ Guest VMs can talk to each other

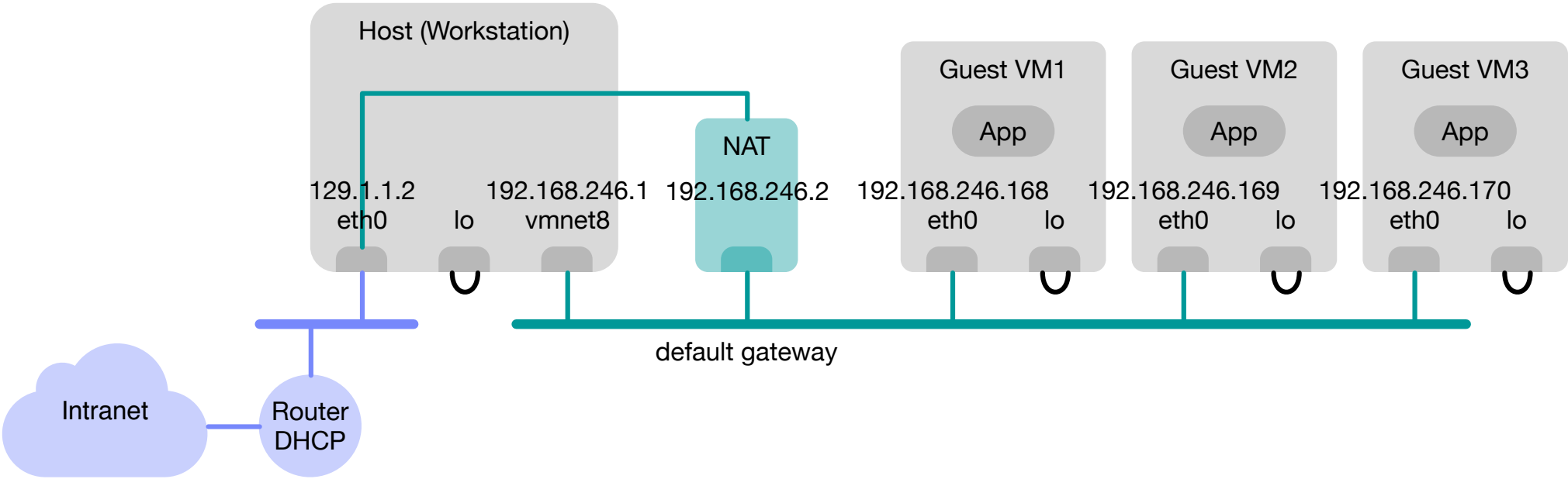
⊕ Guest VMs function even when the host workstation is not connected to any network (e.g. on a plane)

⊕ Host workstation can talk to guest VMs

⊖ Outside world cannot see guest VMs

Virtual networks

VMware Internet Sharing (NAT)



Internet Sharing (NAT) is the default mode for new guest VMs.

Source: VMware, https://www.vmware.com/support/ws45/doc/network_nat_details_ws.html

Vagrant
Introduction

- Vagrant is software that creates and configures virtual development environments
  - It is a layer on top of VirtualBox or VMware
  - It automates a lot of the manual steps a developer has to go through to set up a development VM
- Released as Open Source (MIT) license, with some extensions being proprietary
- Developed by Mitchell Hashimoto (HashiCorp)
- Initially released March 2010
- Runs on Linux, FreeBSD, OS X and Microsoft Windows

Vagrant term	**Common term**
Box	Virtual machine image
Machine	Virtual machine

Vagrant
VM images

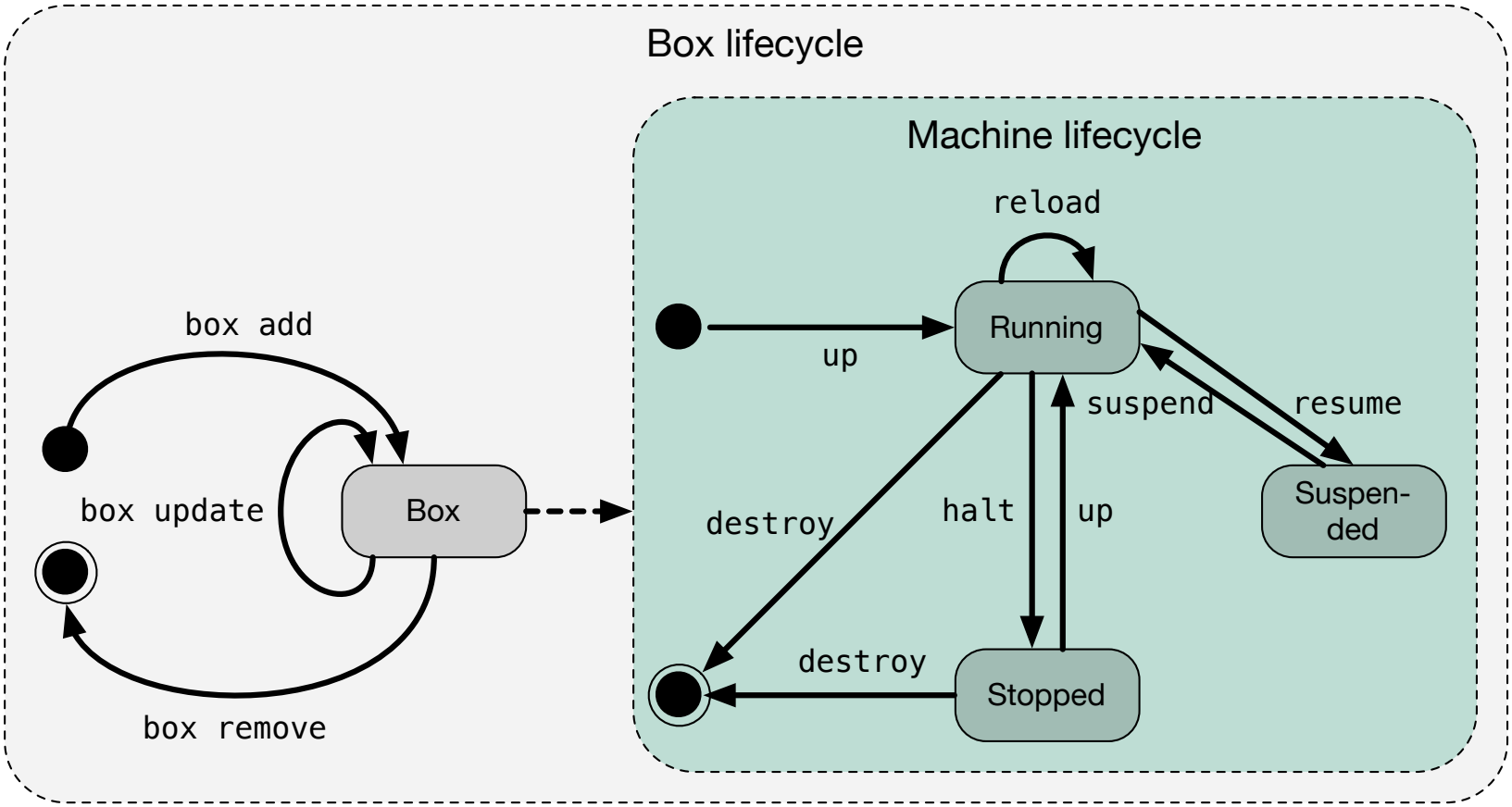
- Vagrant creates VMs from VM images that contain the operating system and possibly additional software
- VM images in Vagrant are called *boxes*
- Vagrant offers a repository of pre-configured VM images
  - Base operating system: Ubuntu, RedHat, CenOS, SUSE, …
  - Base operating system + middleware: Linux + Apache + MySQL + PHP, …
  - User can create his own images

Vagrant
Project directory

- Vagrant makes it easy to deploy code produced in a project in a virtual environment.
- Suppose you have a project directory that contains code that should run a virtual environment.
- You create a VM dedicated to run that code:
  - In the project directory run `vagrant init`. This creates a new file called `Vagrantfile`. Edit the file to
    - Select the image to use for initializing the VM
    - Expose any VM ports on the host machine
    - ...
  - Run `vagrant up` to start the VM.
  - To log into the VM run `vagrant ssh`
  - Vagrant automatically sets up file sharing between host and guest: The project directory is accessible inside the VM at the mount point `/vagrant`.
  - If necessary install and configure additional software in the virtual environment and run the code directly from the `/vagrant` directory or use any other code deployment mechanism.

Vagrant

Lifecycle of boxes and machines



Vagrant
Networking

- Vagrant offers the following networking options:
  - Port forwarding
  - Private network
  - Public network
    - Meaning depends on virtualization provider (VirtualBox, VMware)

Software containers

Introduction

- Virtualization is popular because of its *isolation* abilities: software isolation, fault isolation and performance isolation.
- A different approach to isolation is provided by software containers.
- A software container:
  - Uses virtualization features of the operating system
  - Is much lighter weight than hardware virtualization: no separate operating system needed
- Used by Google internally since many years
- Popularized by Docker, released in 2013

Software containers
Linux containers

- What is a Linux container?
  - A combination of Linux technologies
    - namespaces
    - cgroups
    - normal Linux networking
    - normal Linux security mechanisms
  - which allow you to run userspace programs in the container which will have the illusion that they are alone on the machine.

Software containers

Linux containers

- Linux containers allow to run processes in an isolated environment via *namespaces*.
  - UTS: Allows a different hostname for each container
  - PID: Hides processes outside the namespace from processes in the namespace. Calling shutdown() will perform a shutdown on the processes only in that namespace.
  - MOUNT: Allows a group of processes to mount and unmount filesystems and not have these changes visible outside of the namespace.
  - UID: Allows you to give processes root inside the namespace but have this mapped to a normal user when interacting with processes outside the namespace (eg accessing files).
  - IPC: Allows you to have a separate space for IPC resources such as semaphores and locks.
  - NET: Allows processes to have their own networking stack with different interfaces, firewalls and routing tables.
  - SYSLOG: Only see ksyslog messages that belong to the namespace you are in (eg. dmesg).
  - AUDIT: Allows a namespace to only see messages from the audit subsystem that apply to that namespace.
  - CGROUP: Allows a namespacing of cgroups giving you your own separate hierarchy.

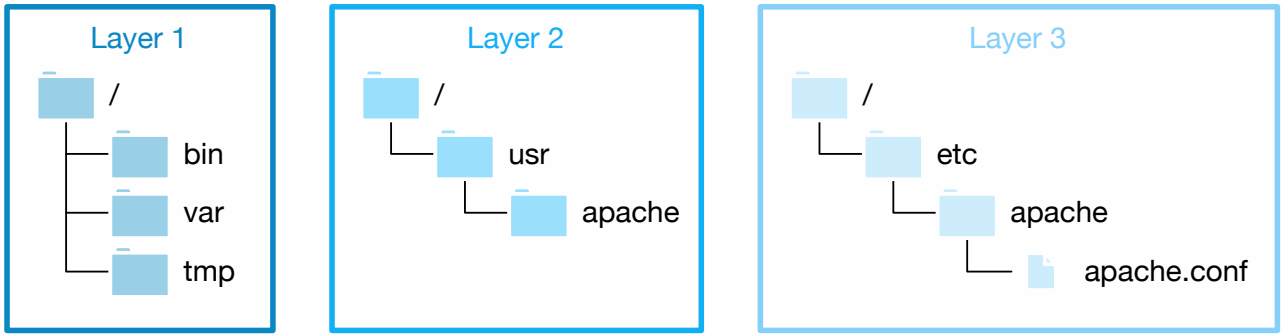
Software containers

Example for isolation: chroot

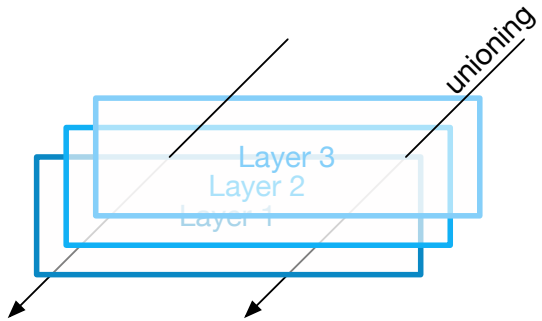
- On Unix operating systems a chroot is an operation that changes the apparent root directory for the current running process and its children to a subtree of the file hierarchy. A program run in a chroot environment cannot access files outside the subtree.

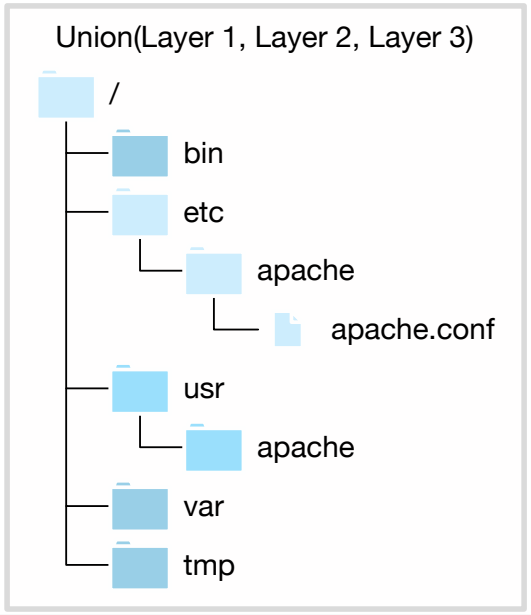
Software containers

Union file system



- UnionFS is a filesystem that implements a *union mount* for other file systems.
- A union mount transparently overlays several filesystems (layers) to form a single coherent virtual file system.
- When mounting layers, the priority of one layer over the other is specified. So when both layers contain a file with the same name, one gets priority over the other.
- The different layers may be read-only or read-write FSs. Writes to the virtual FS are directed to a specific real FS.



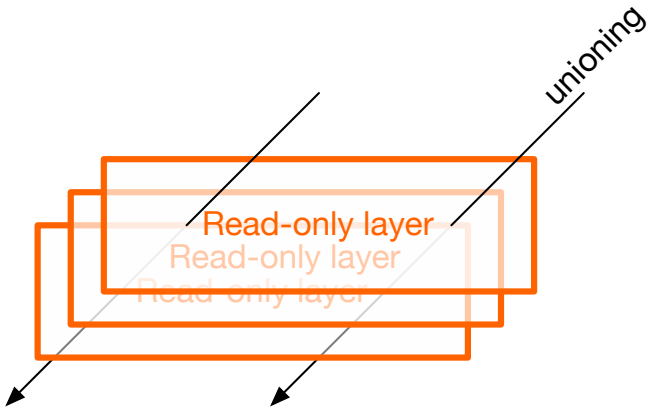


Docker

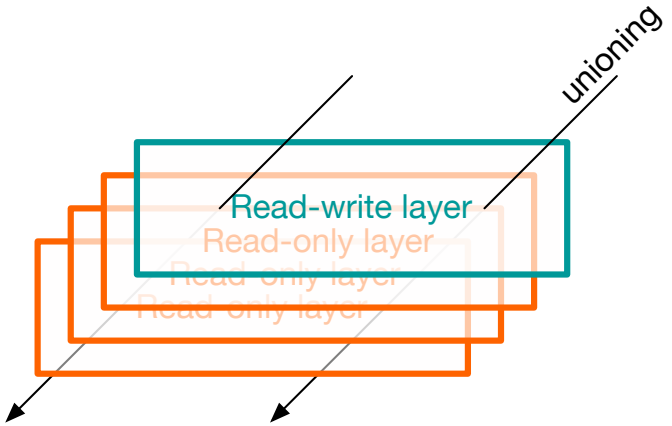
Containers and container images

- Containers are created from container images.

Container image: Union view of several layers of read-only filesystems

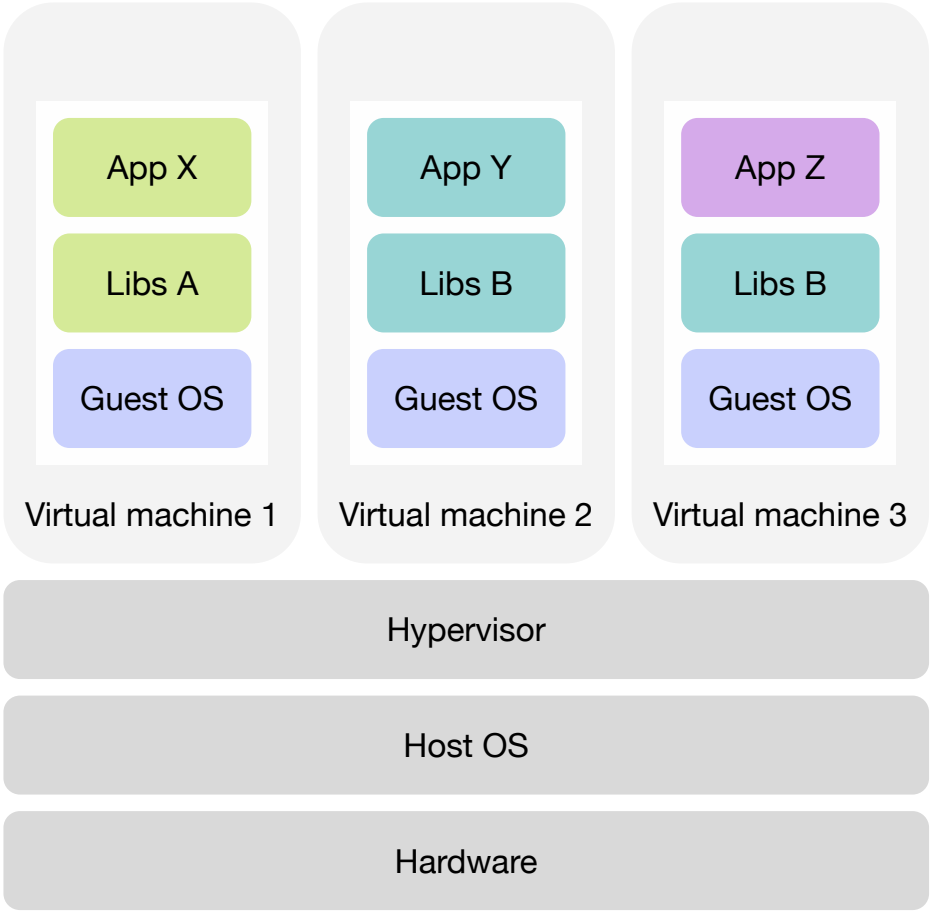


Container: Union view of a read-write filesystem on top of several layers of read-only filesystems

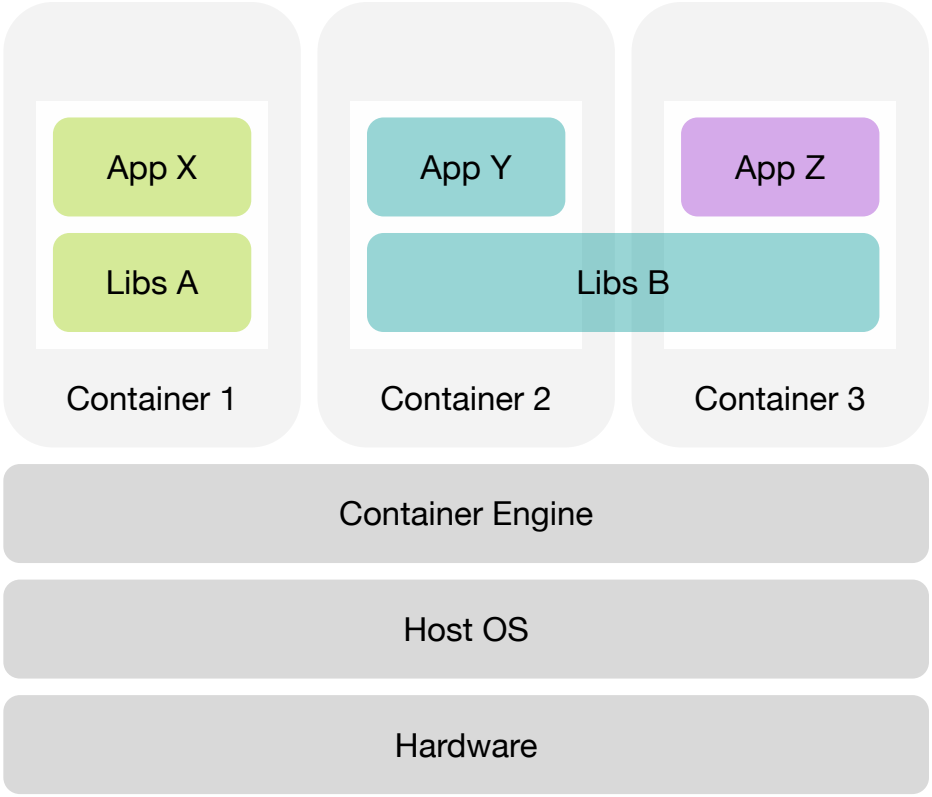


Software containers

Containers vs Virtual Machines



Three VMs running on a single host



Three containers running on a single host

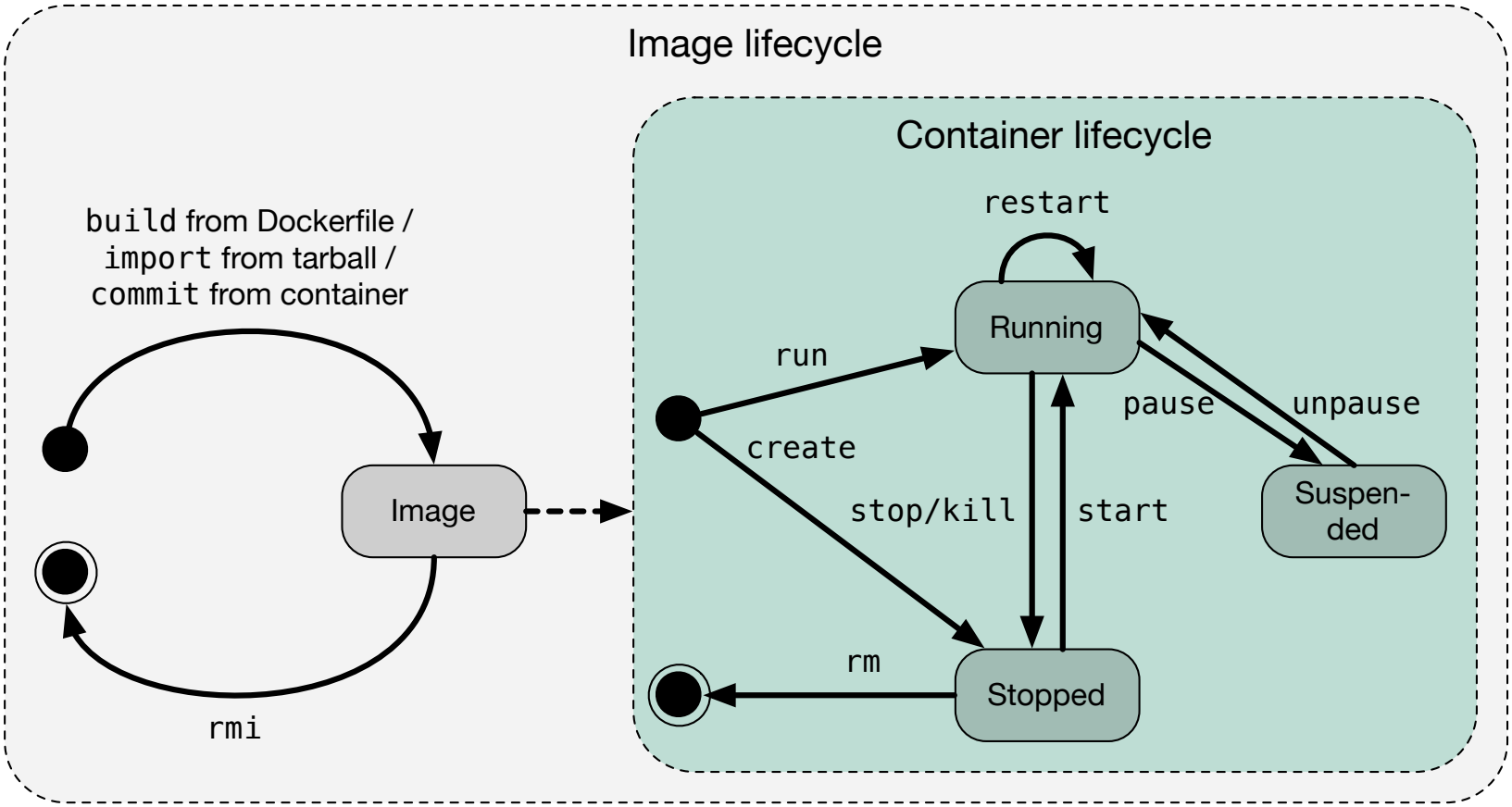
Software containers

Containers vs Virtual Machines

- Virtualization:
  - The Hypervisor creates Virtual Machines, controls access to the host OS and hardware and interprets privileged instructions if necessary.
  - Each VM requires a full copy of the operating system, the application being run, and any supporting libraries.
- Software Containers:
  - The host's kernel is shared with the running containers.
    - This means that containers are always constrained to running the same kernel as the host.
  - Applications that use the same libraries can share them instead of having redundant copies.
  - Processes inside containers are equivalent to processes on the host and do not incur the overheads associated with hardware virtualization.

Docker

Lifecycle of container images and containers



Docker

Naming containers

- By default Docker assigns to each container a unique machine-readable Container ID like `e674bd4a377e`
- Additionally it generates by default a unique human-readable (nonsensical) name like `evil_ptolemy`
- The user can override the human-readable name by using the `--name` option when creating the container. It must be unique.

```
$ docker ps
CONTAINER ID  IMAGE           COMMAND     CREATED      STATUS     PORTS                NAMES
e674bd4a377e  heigvd/ha       "bash"      7 days ago   Up 7 days  0.0.0.0:80->80/tcp   loadbalancer
c845c95bfb48  heigvd/webapp   "./run.sh"  5 weeks ago  Up 7 days  3000/tcp             node1
3fb984306e36  heigvd/webapp   "./run.sh"  5 weeks ago  Up 7 days  3000/tcp             node2
```

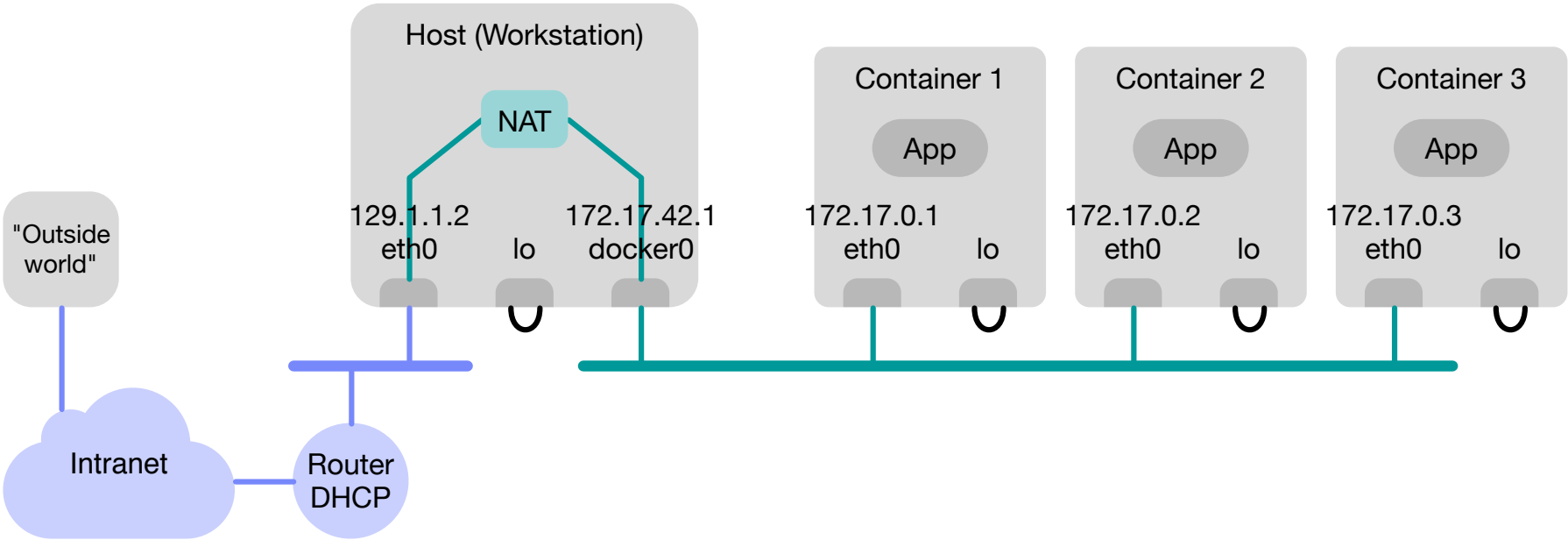
Docker

Default network configuration

- By default Docker configures networking as follows:
  - It creates a private virtual subnet that is not in use (often it is 172.17.0.0/16).
  - On the host it creates a virtual interface named `docker0` connected to this subnet and assigns it an IP address (often it is 172.17.42.1).
  - On each container it creates a virtual interface named `eth0` connected to this subnet and assigns it an IP address (often the first container receives 172.17.0.1, the second 172.17.0.2, and so on).
  - It creates a NAT between the private subnet and the subnet to which the host is connected so that containers can establish connections to the outside world.
- In summary this enables that
  - (A) The containers are able to access the outside world
  - (B) The host is able to access the containers
  - (D) A container is able to access another container
- and disallows that
  - (C) The outside world is able to access a container

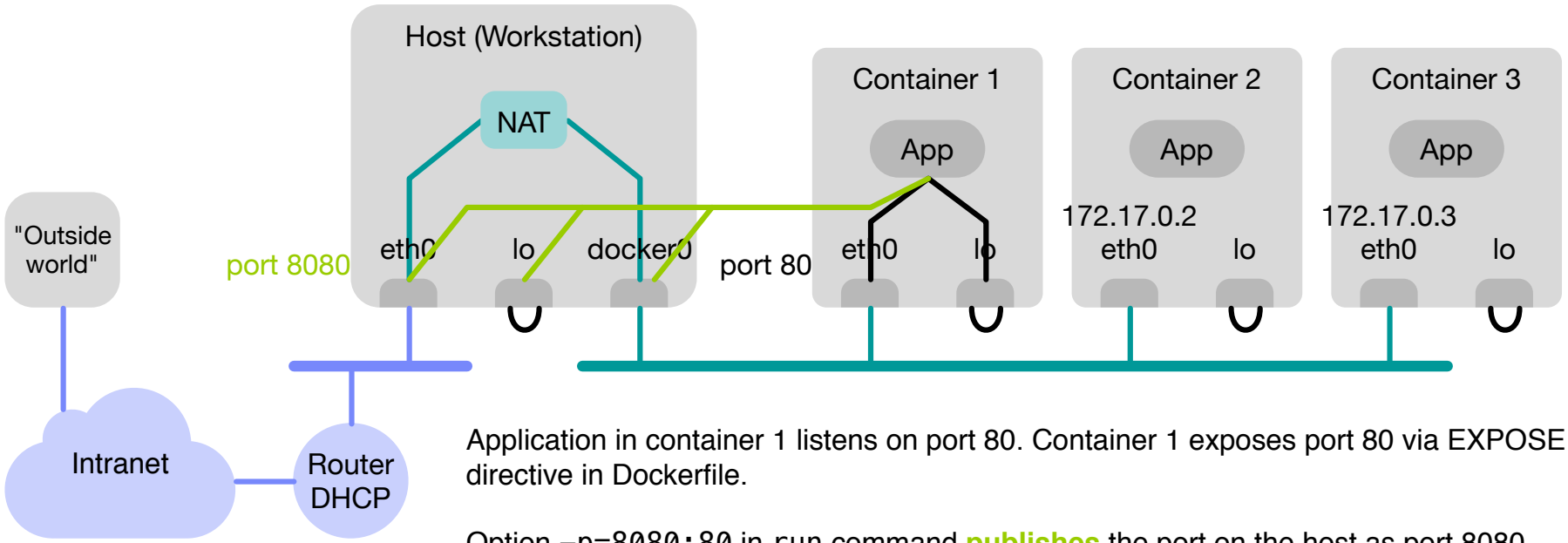
Docker

Default network configuration



Docker

Default network configuration with published exposed ports



Application in container 1 listens on port 80. Container 1 exposes port 80 via EXPOSE directive in Dockerfile.

Option `-p=8080:80` in `run` command **publishes** the port on the host as port 8080.

Docker

Network configuration information

- By default Docker automatically sets up the following configuration files in each container:
  - /etc/hostname
  - /etc/hosts
  - /etc/resolv.conf

File /etc/hostname:

```
e674bd4a377e
```

Container ID

File /etc/hosts:

```
172.17.0.3 e674bd4a377e
127.0.0.1  localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.3 loadbalancer
```

Container name

File /etc/resolv.conf:

```
nameserver 10.0.2.3
```

Nameserver of the host

Docker

Network configuration information — Linking

- When a container B is created with a link to an existing container A, Docker conveniently provides in B's /etc/hosts file the IP address of A.

File /etc/hosts in B:

```
[...]
172.17.0.2 s1 3fb984306e36
```

Container name of A

Container ID of A

Software Containers and Docker

History

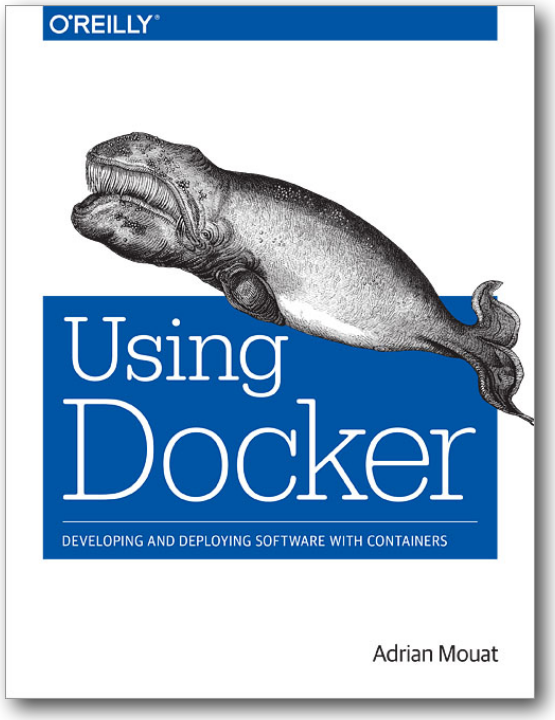
- 1979 Unix gets `chroot` command: file system isolation
- 1998 BSD gets `jail` utility: chroot sandboxing extended to processes
- 2001 Solaris gets *Zones* technology: pretty complete containerization, but limited to Solaris
- 2001 Parallels develops Virtuozzo container technology for Linux, open sources it in 2005 as OpenVZ
- 2005 Google starts development of CGroups for Linux and begins moving its infrastructure to containers
- 2008 Linux Containers (LXC) project started: complete containerization solution
- 2008 dotCloud, a startup, develops a language-agnostic Platform-as-a-Service offering. Core building block is Docker, initially a wrapper around LXC.
- 2013-03 dotCloud open sources Docker. Within six months, it has more than 6'700 stars and on GitHub and 175 new contributors.
- 2013-09 RedHat becomes a major partner and starts using Docker for its OpenShift PaaS offering.
- 2013-10 dotCloud renames itself to Docker.
- 2014-06 Release of Docker 1.0
- 2014-10 Microsoft announces that Windows Server will support Docker.

Software Containers and Docker

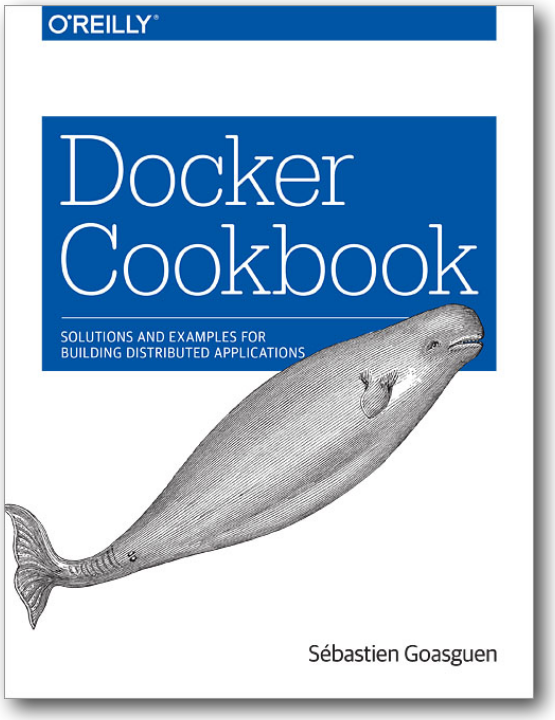
History

- 2014-11 Amazon launches beta of Amazon EC2 Container Service: Docker support on Amazon's cloud. General Availability 2015-04
- 2014-12 CoreOS announces development of rkt, its own container runtime.
- 2015-06 Announcement of the Open Container Initiative: common standard for container formats and runtimes
- 2015-06 FreeBSD project announces Docker is supported on FreeBSD, using ZFS and the Linux compatibility layer
- 2015-08 General Availability of Google Container Engine: Docker support on Google's cloud
- 2015-09 IBM offers IBM Containers: Docker support on IBM's cloud

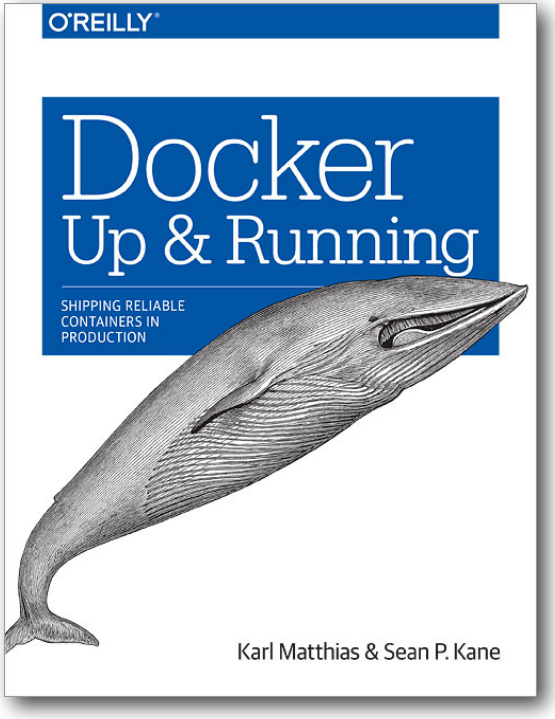
References



Adrian Mouat
Using Docker
2015-12
O'Reilly Media



Sébastien Goasguen
Docker Cookbook
2015-11
O'Reilly Media



Karl Matthias, Sean P. Kane
Docker: Up & Running
2015-06
O'Reilly Media