Software Deployment

Docker

Sergio Peignier sergio.peignier@insa-lyon.fr

Associate Professor
INSA Lyon
Biosciences department

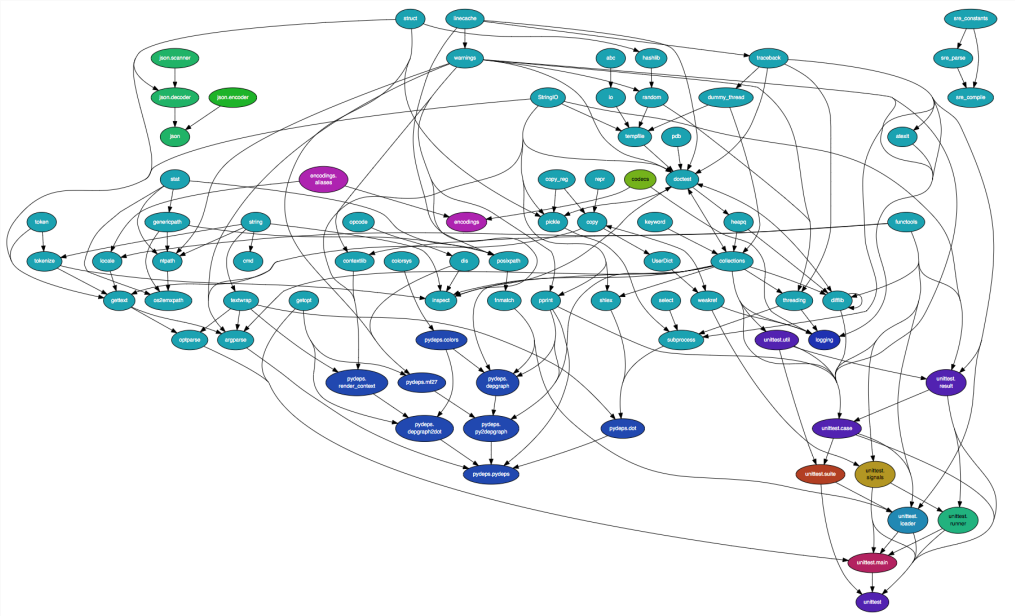
Table of contents

Dependency Hell

Tree Packages

Package Management system:

- Maintains a **tree** of **packages** with **specific versions**.
- **Re-use packages** (dependencies)
- **Host-centric paradigm**
- **No isolation**

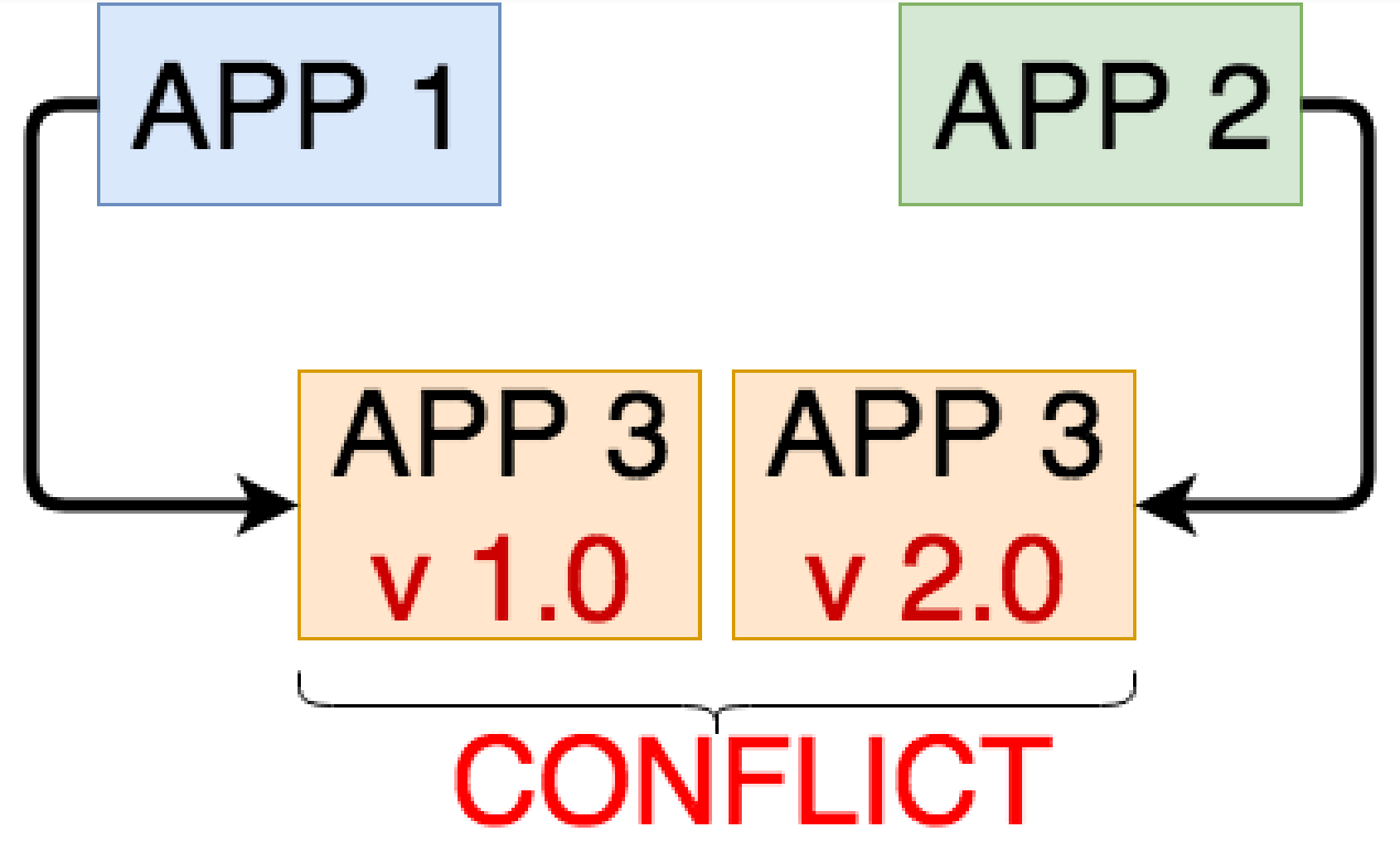


Example of tree dependencies for Python stdlib using pydeps

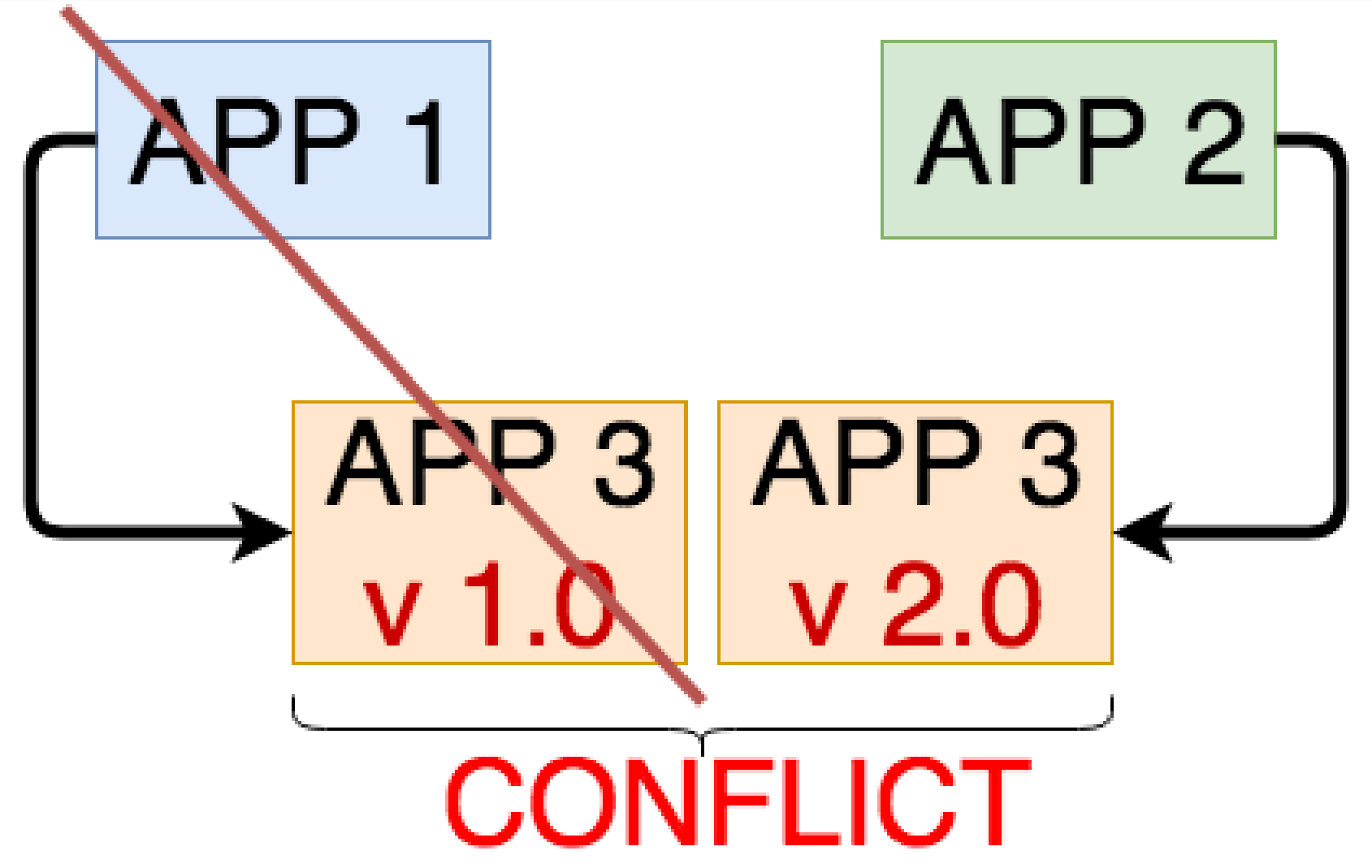
Dependency Hell

- **Packages** depend on **specific versions** of **other packages**
- **Shared packages** (dependencies tree)
- **Conflicts** $\rightarrow$ **Break dependencies**
- **Version numbering**: <Major>.<minor>
  - **Major**: compatibility **not ensured**
  - **minor**: compatibility **ensured**

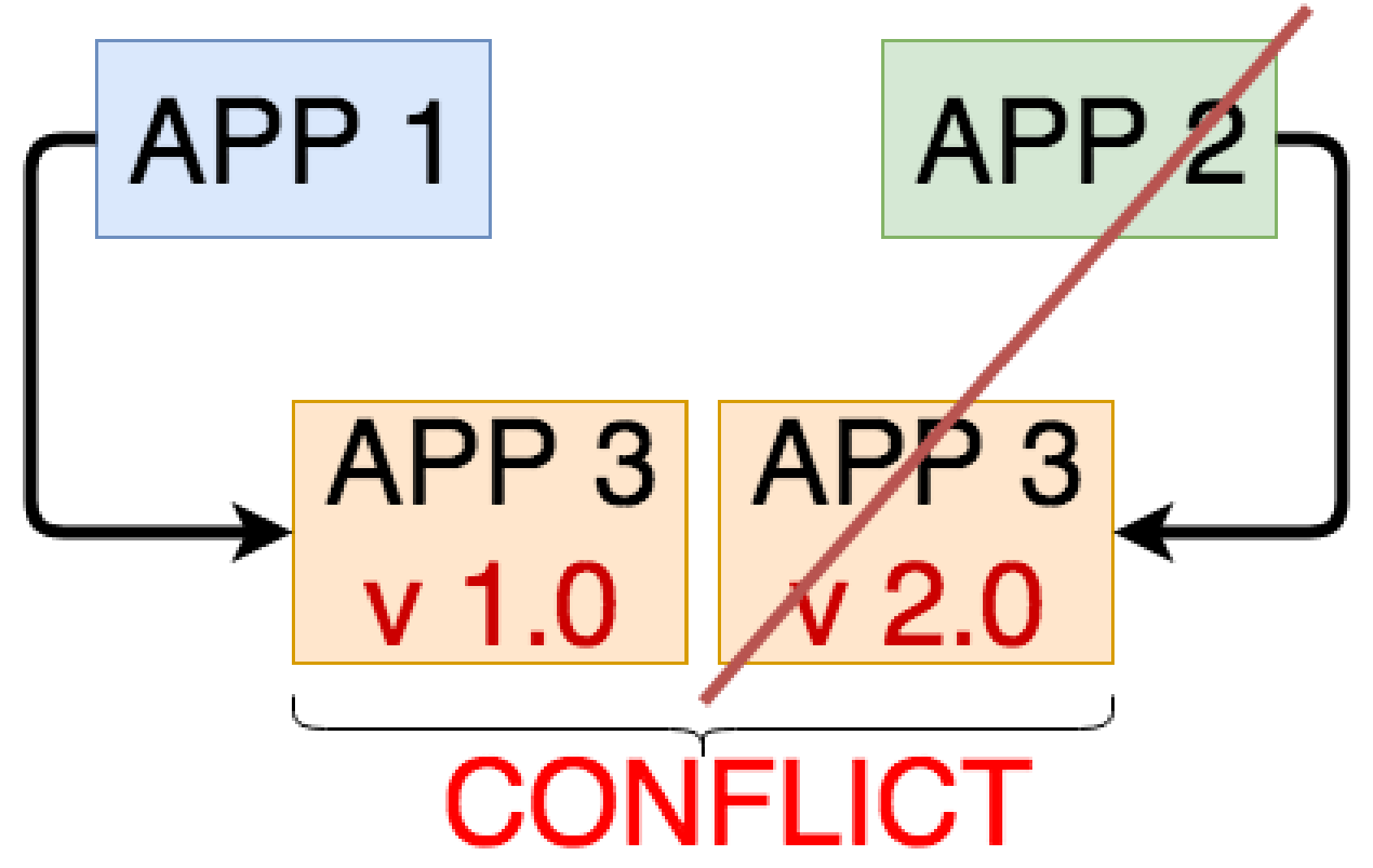
Conflicts



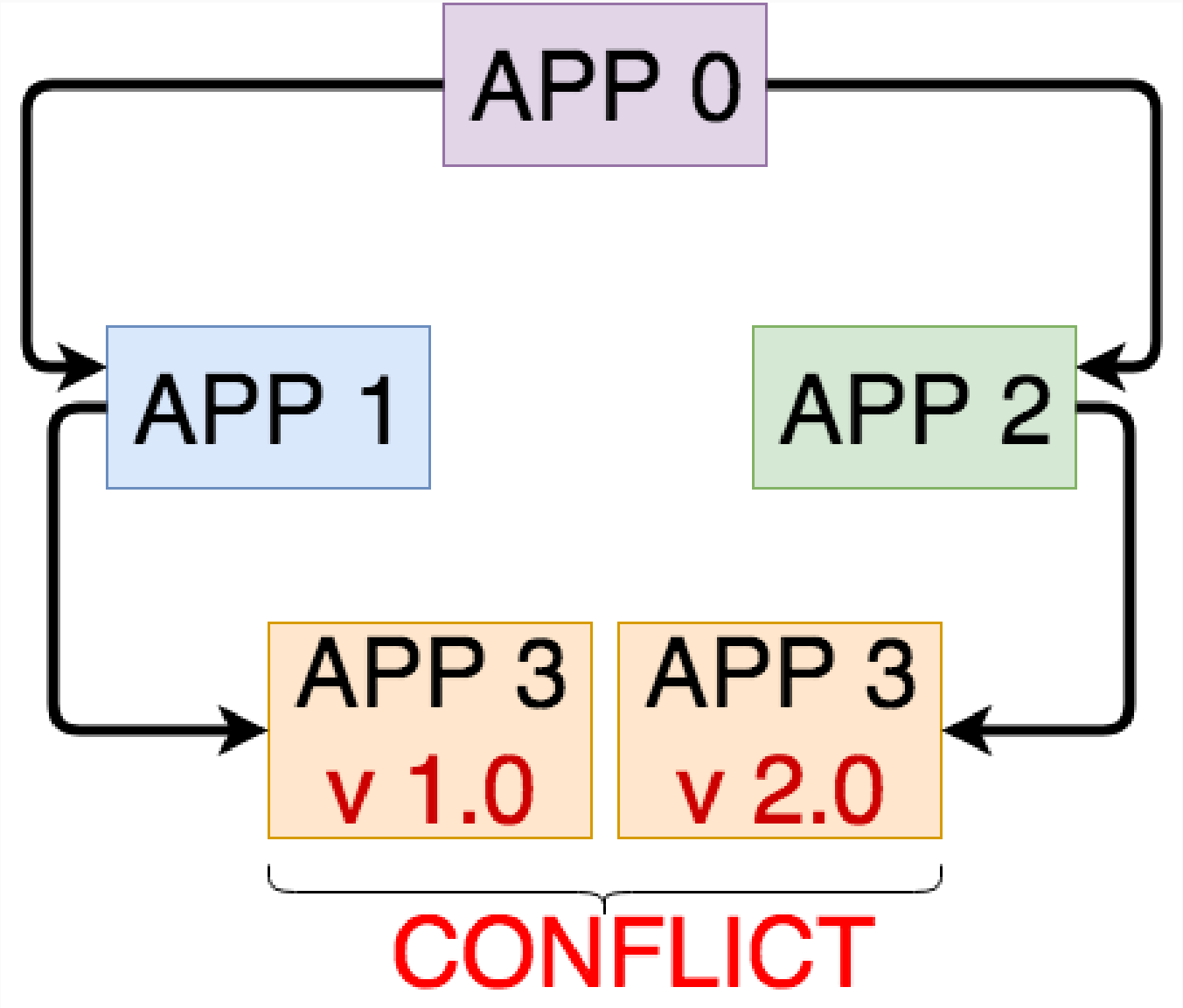
Conflicts



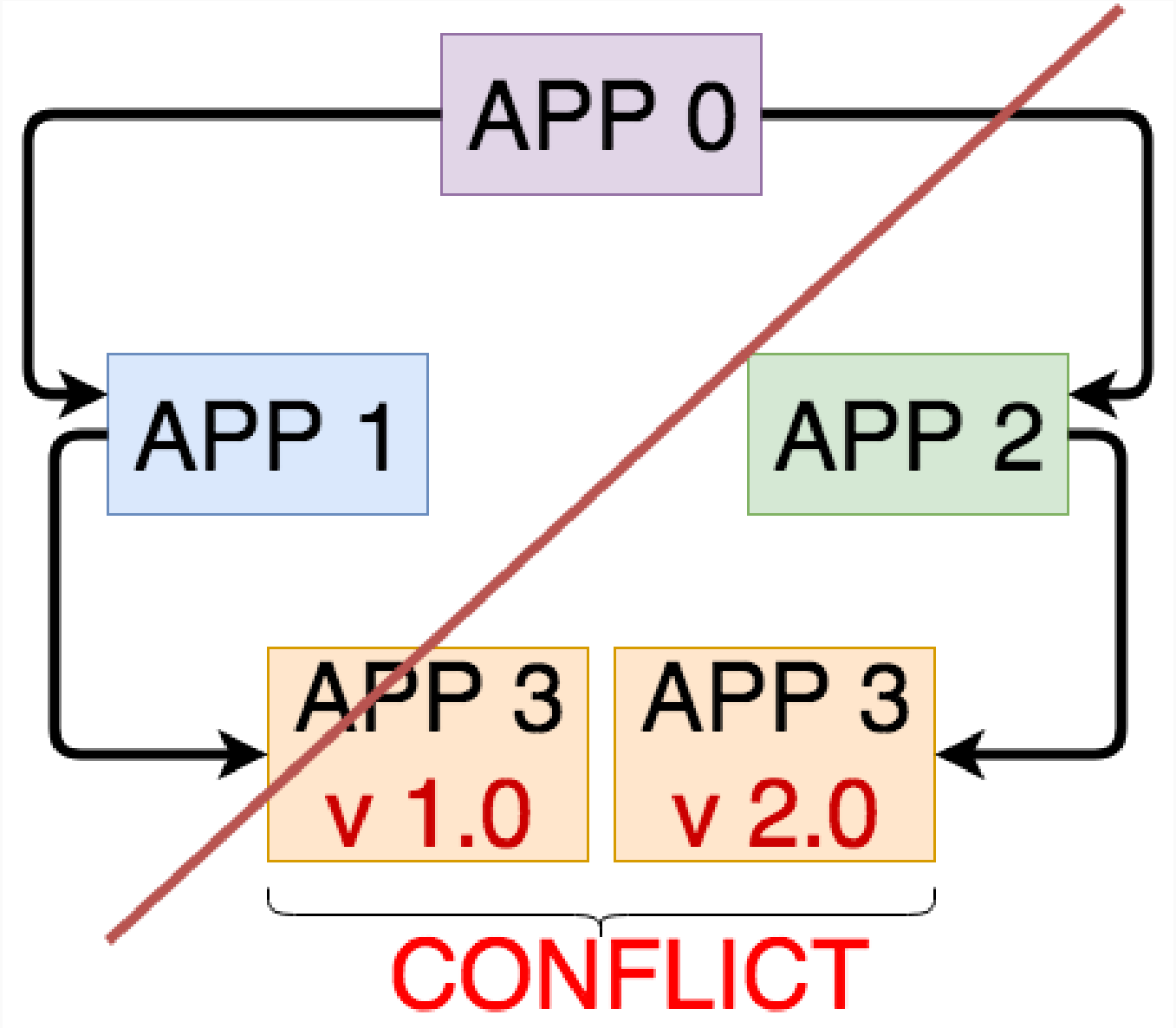
Conflicts



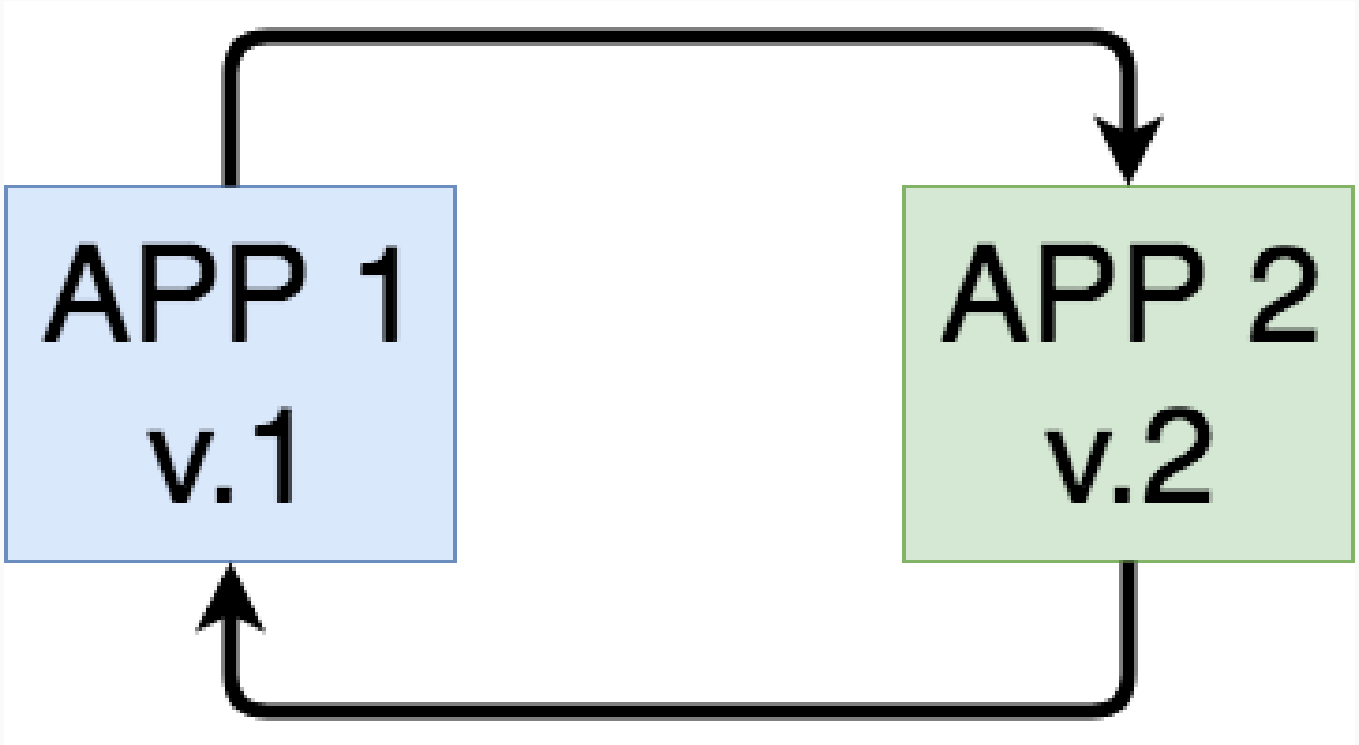
Diamond conflicts



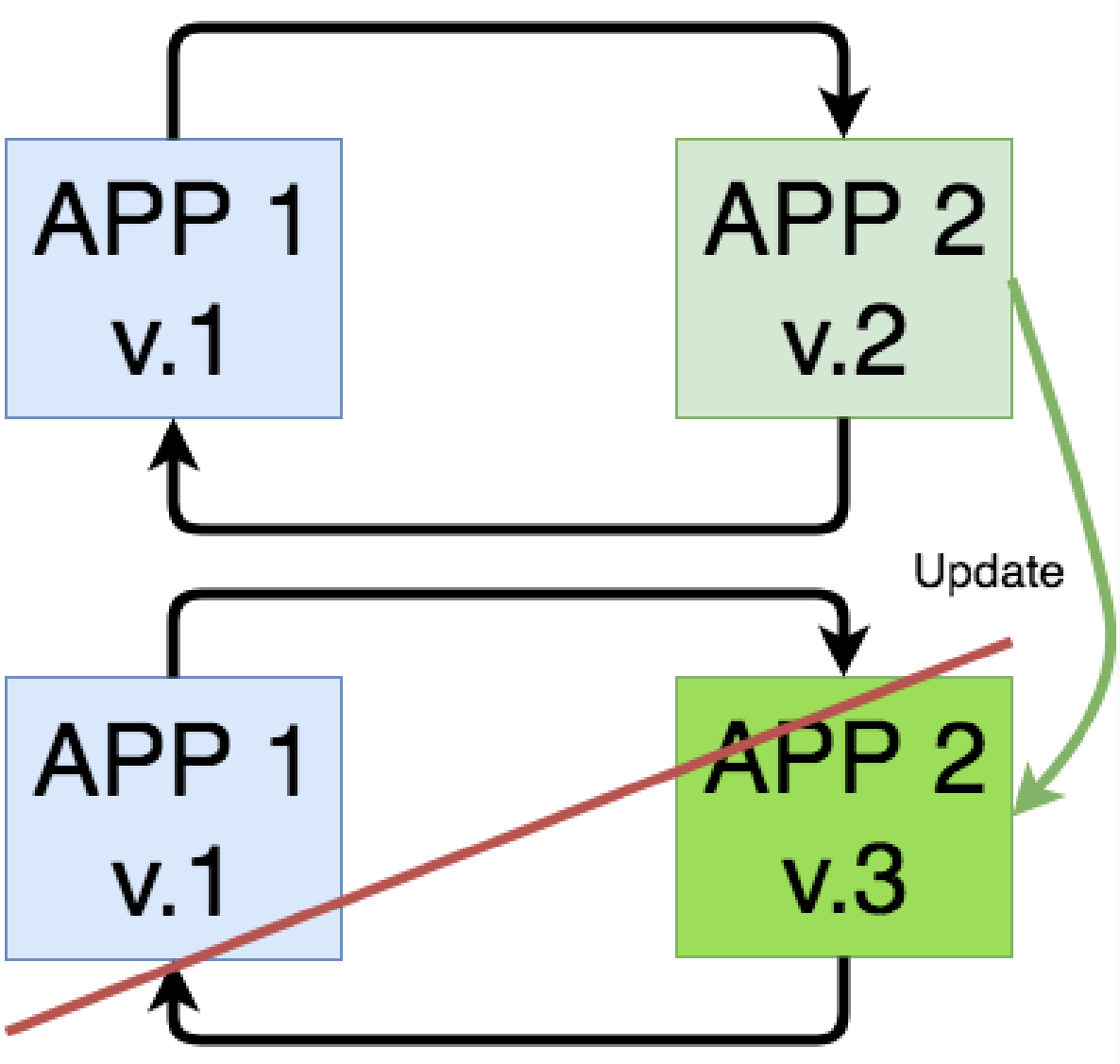
Diamond conflicts



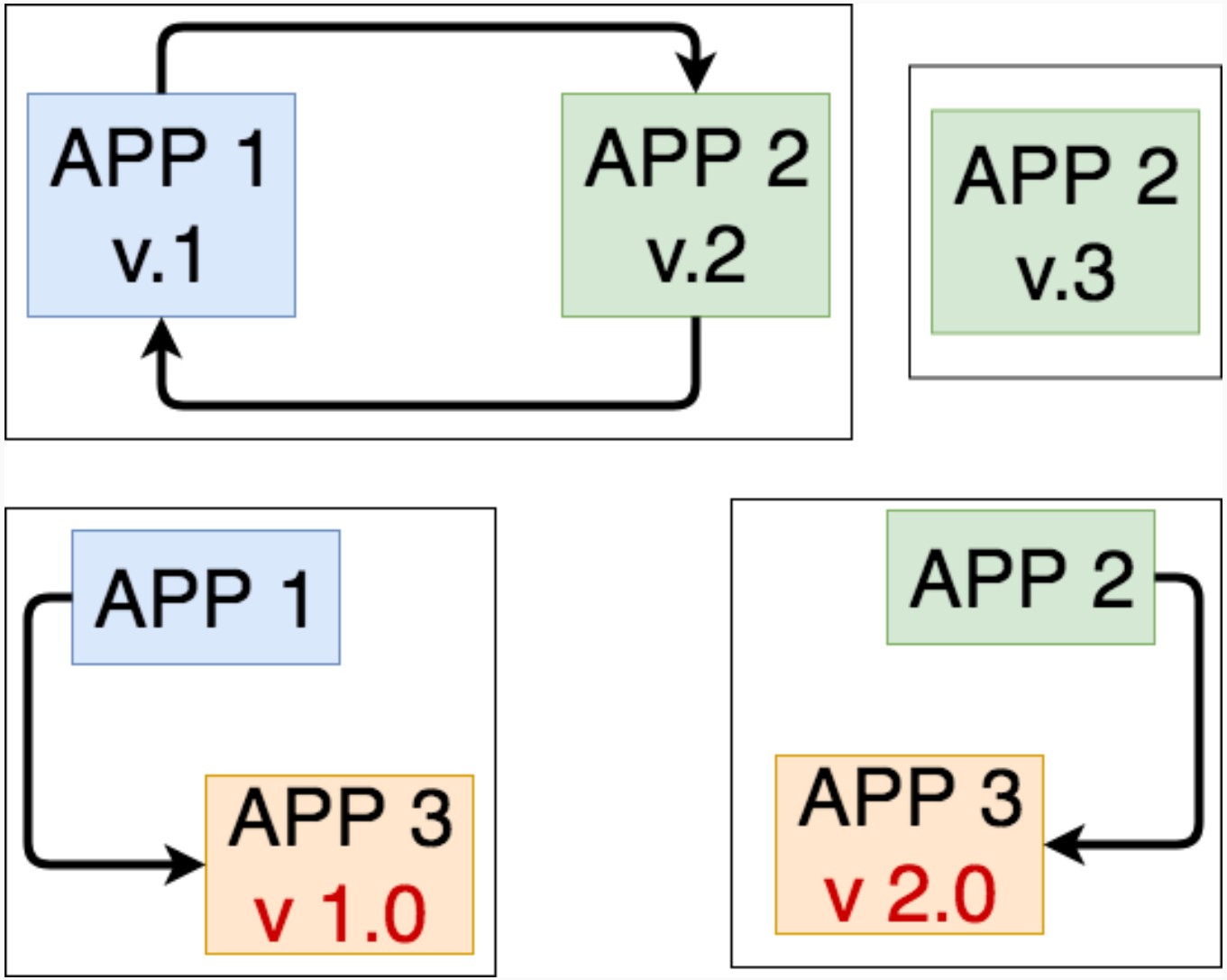
Circular conflicts



Circular conflicts



Avoid hell dependency



Virtualization

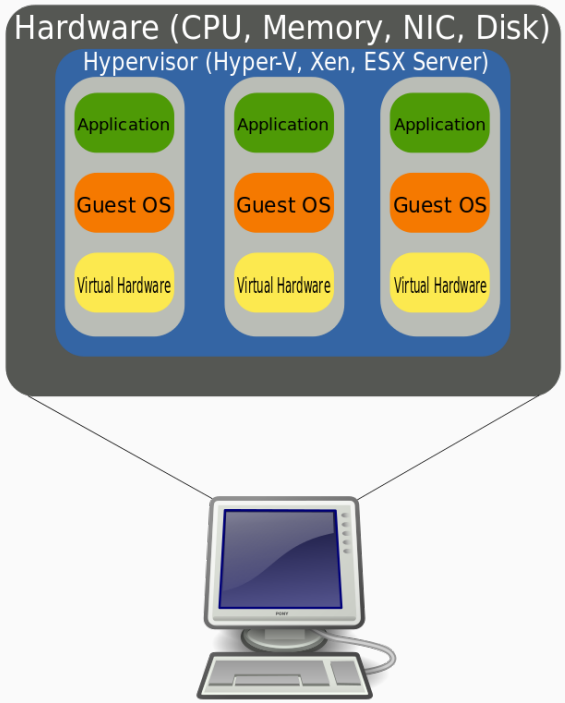
Definitions

- **Virtual Machine**: computer system **emulator**.
- **Emulator**: **hardware/software** enabling a **computer system** to **behave** like **another** one.
- **Emulator System** also called **Host System**
- **Emulated System** also called **Guest System**
- **Guest Systems**: **Isolated** from each other.

Hypervisor

Physical machine $\leftrightarrow$ **hypervisor** $\leftrightarrow$ **virtual machines**

- Creates **virtual environment** to receive the **guest VM**.
- Manages **resource allocation**
- **Intercepts** operations of **guest OS**
- **Emulates** operations on **host OS**

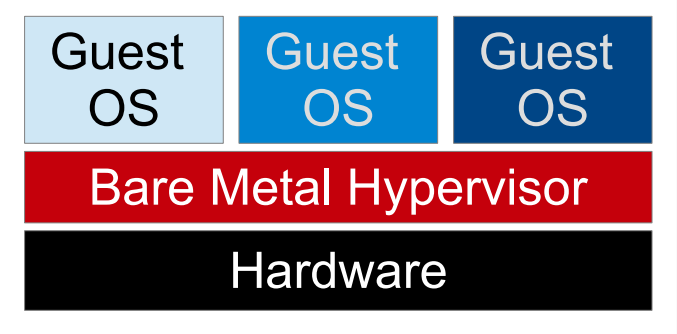


Types of Virtualization

- Bare-metal
- Paravirtualization
- System virtual machines
- Process virtual machines / Container based

Types of Virtualization

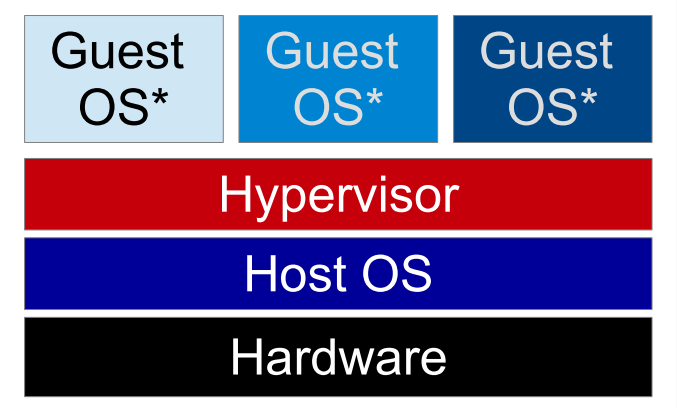
- Bare-metal
  - **Hypervisor** runs on **hardware directly**.
  - **Bare-metal** hypervisor requires **fewer resources**.
  - Better performances



- Paravirtualization
- System virtual machines
- Process virtual machines / Container based

Types of Virtualization

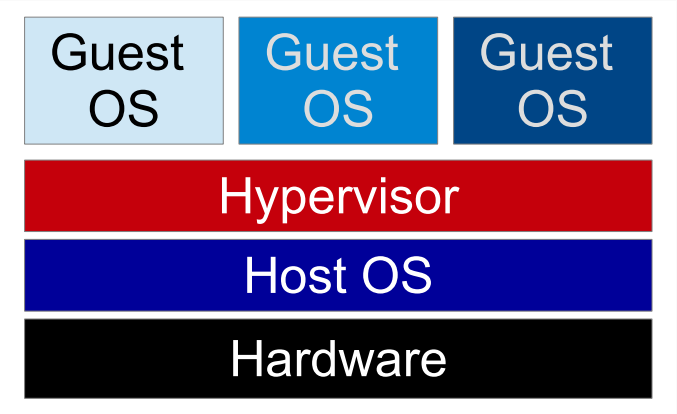
- Bare-metal
- Paravirtualization
  - Guest OS is **recompiled before installation** in a VM.
  - Guest OS within the system **share resources**.
  - Better performances



- System virtual machines
- Process virtual machines / Container based

Types of Virtualization

- Bare-metal
- Paravirtualization
- System virtual machines a.k.a Full virtualization / Emulation::
  - Functionalities to run an entire **OS**.
  - **Multiple isolated OS** run in **1 physical machine**.
  - **Hardware** shared and managed using a **hypervisor**.



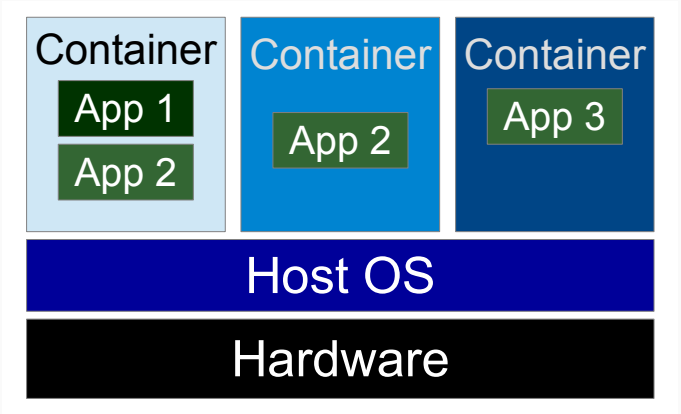
- Process virtual machines / Container based

Types of Virtualization

- Bare-metal
- Paravirtualization
- System virtual machines
- Process virtual machines / Container based a.k.a

  Application virtualization / Managed Runtime Environment:

  - Run **inside** a **host OS** : **OS-level virtualization**
  - Run app. in a **platform-independent** environment.
  - Support a **single process**
  - app. **starts** → VM **created** | app. **exits** → VM **destroyed**



Containers

Containers | Advantages over VMs

Characteristics:

- No need to boot an OS kernel
  $\rightarrow$ Fast creation
- Shares host's scheduler $\rightarrow$ save resources
- Small overhead on host
- Small sizes w.r.t. full VMs

Benefits:

- $\sim$ Native performances
- Easy to distribute

Containers | Advantages over host-centric view

Isolation:

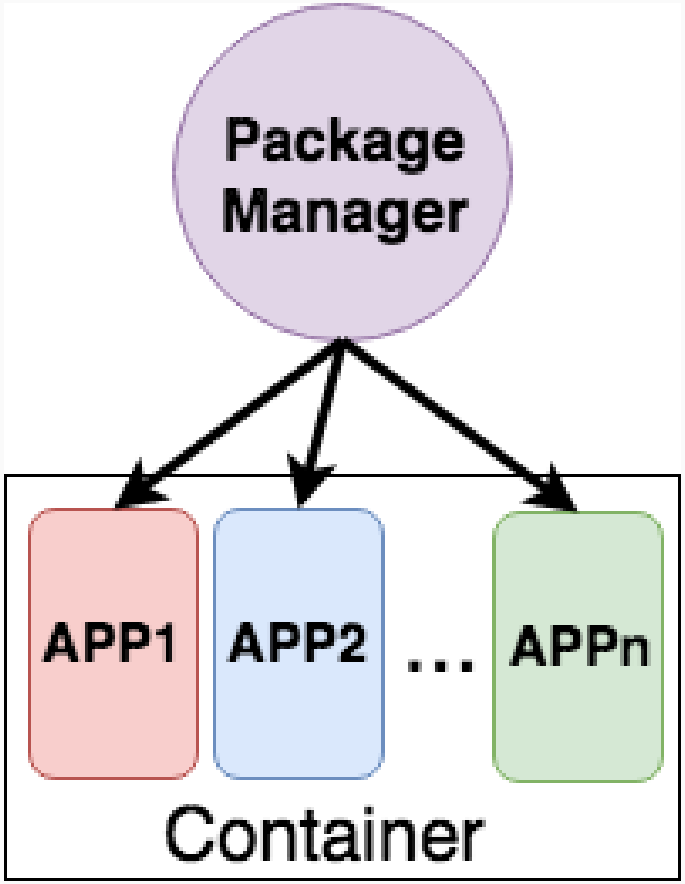
- Install any app. with any version
- Prevent Dependencies Hell
- Resources configuration

Portability:

- Deploy full and consistent software pipelines
- Many predefined containers available
- Easy start
- OS agnostic

Containers and Package managers

- Package Managers → fill container image.
- Keep container image minimal
- Deploy container image



Docker

Docker

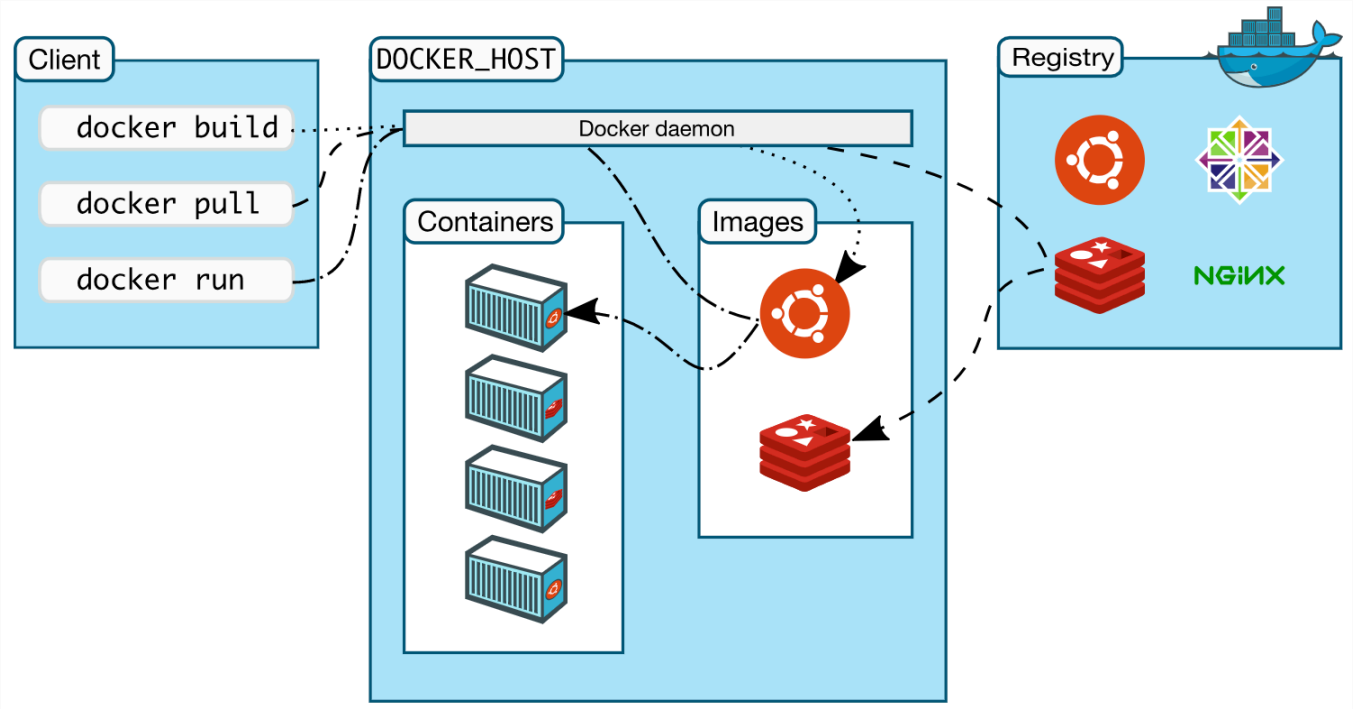
Program performing **OS-level virtualization**.[1]

”**open platform** for developers and sysadmins to **build**, **ship**, and **run distributed applications**”[2]

[1] **License**: open source Apache 2.0
[2] `https://docs.docker.com/engine/docker-overview/`

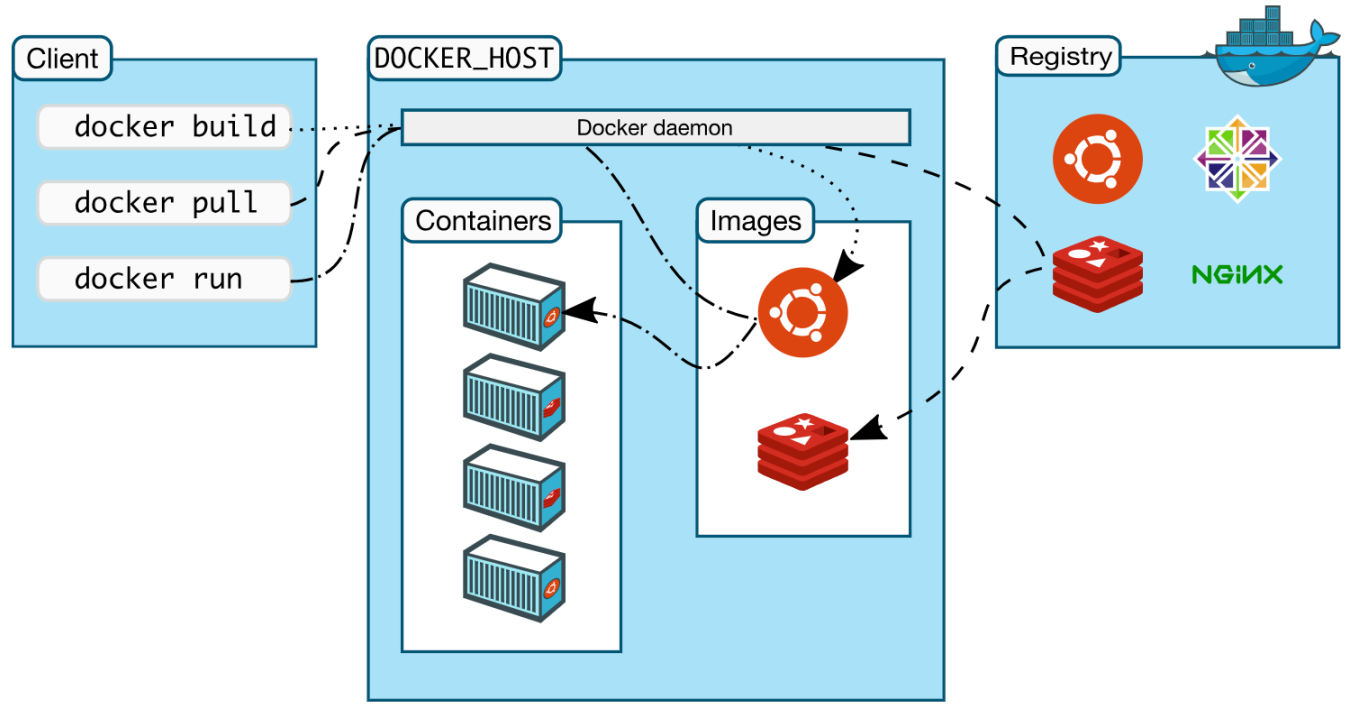
Docker Engine: Client-Server Architecture

- **Server**: long-running daemon process (`dockerd`)
- **Client**: Command line interface (CLI) (`docker`)
- **Registries**: Stores Docker images (e.g. Docker Hub)



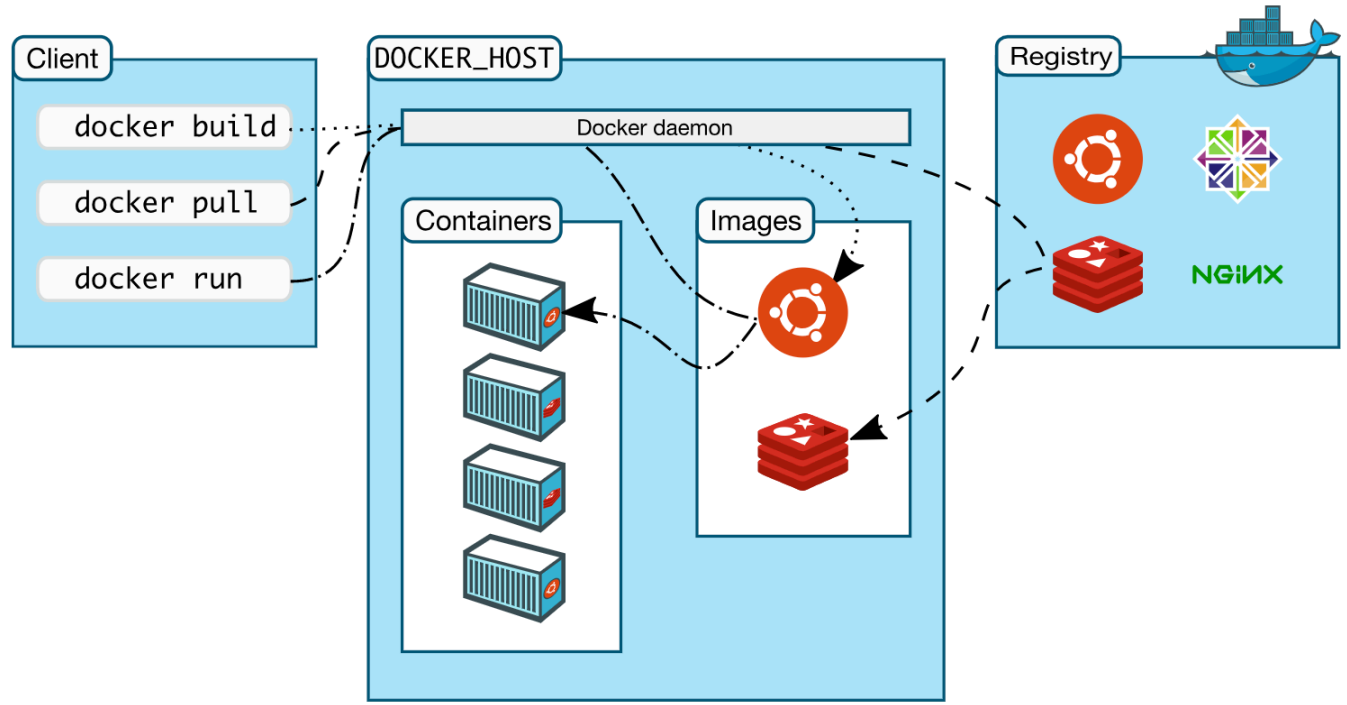
Docker Engine: Client-Server Architecture

- **Server**: long-running daemon process (`dockerd`)
  - create/manage Docker objects
  - e.g. images, containers, ...
- **Client**: Command line interface (CLI) (`docker`)
- **Registries**: Stores Docker images (e.g. Docker Hub)



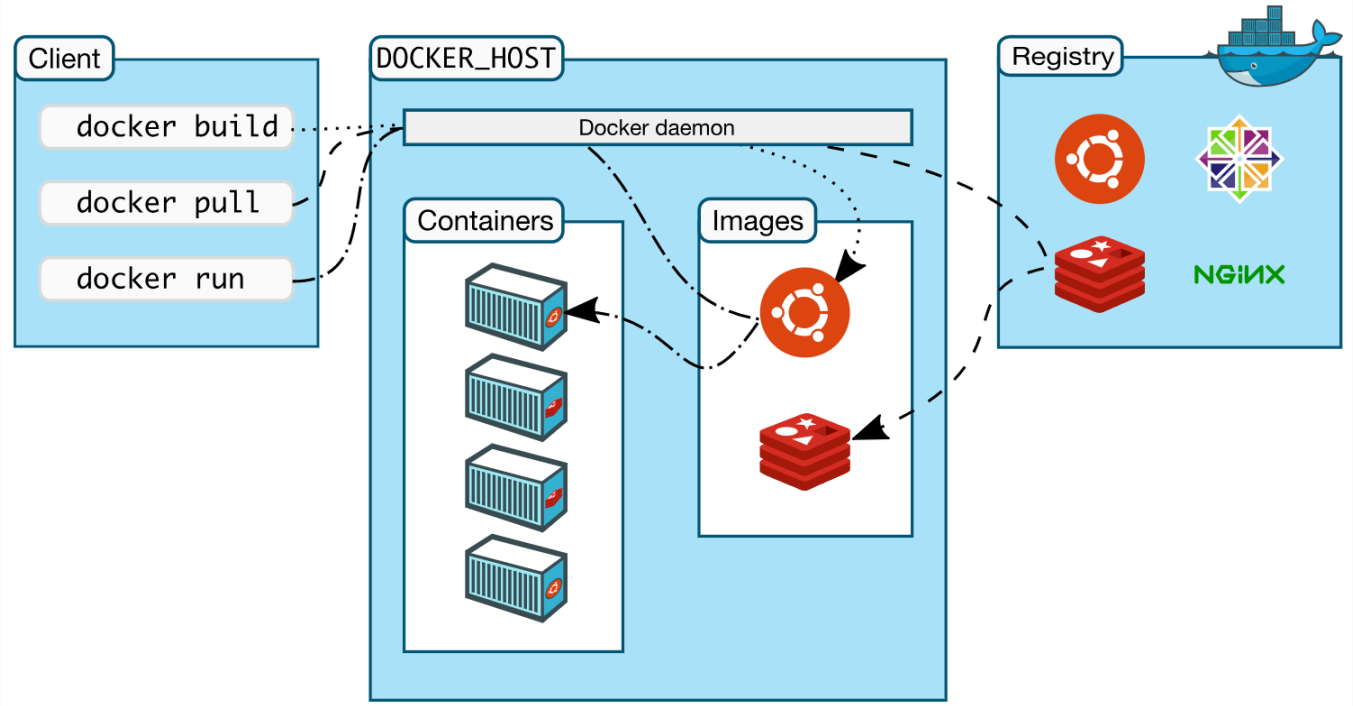
Docker Engine: Client-Server Architecture

- **Server**: long-running daemon process (`dockerd`)
- **Client**: Command line interface (CLI) (`docker`)
  - Tell the daemon **what to do** using REST API
  - **REST API**: Docker Application Programming Interface
- **Registries**: Stores Docker images (e.g. Docker Hub)



Docker Engine: Client-Server Architecture

- **Server**: long-running daemon process (`dockerd`)
- **Client**: Command line interface (CLI) (`docker`)
- **Registries**: Stores Docker images (e.g. Docker Hub)
  - `docker pull`: load image
  - `docker push`: dump image



Docker Objects

- Images
  - Container creation instructions
  - **Read-only** template
- Containers
  - Image's **executable instance**
  - Defined by its **image** + **config. options**
- **Networks**: Connect to Host, other containers, ...[3]
- **Volumes**: Save data in the Host[4]
- ...

[3] `https://docs.docker.com/network/`
[4] `https://docs.docker.com/storage/`

Docker Container | Definition

”A **Docker container image** is a **lightweight**, **standalone**, **executable package** of software that **includes everything needed** to **run** an **application**”[5]

- **Contain** their own **apps.**, **libraries** and **config. files**
- **Container**: **Unit** for **distribution** and **testing**
- Run **Image** on **Docker engine** $\rightarrow$ **container**

[5] `https://www.docker.com/resources/what-container`

Docker Containers | Features

- **Isolated** from each other (customizable isolation)
- **inter-container** exchanges via **dedicated channels**
- **Containers** run in **single OS** kernel: **Lighter** than VMs
- **Containers** run on any **physical**/**VM**: **Portability**

Creation:

- **Created** from **images** which specify the **content**.
- **Create** new image: **customize** existing one
- Images **available** at Dockers' **registries** (repositories)

Under the Hood

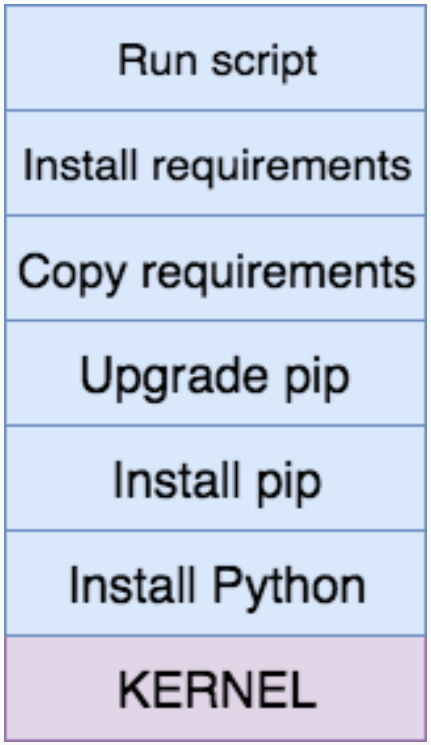
Programmed in **Go** | Relies on **Linux kernel features**:

- **Namespaces**:
  - Provide **isolated workspace**.
  - Run container **components** in separate namespaces (e.g., pid Process isolation, net network interface, ...)
- **Control groups** - cgroups:
  - **Share hardware resources** to containers.
  - Defines **limits** and **constraints**.
- **Union file systems** - UnionFS:
  - Manage containers as **layers**
  - i.e., containers **building blocks**
- **Container format** - libcontainer:
  Wrapper combining namespaces, cgroups, UnionFS.

Docker Images in Practice

Build an Image

- Create a `Dockerfile`:
  Specify **steps** to **create/run** the image.
- Each `Dockerfile` instruction crates a **layer**.
- **Rebuilt image**: only **modified layers** are **rebuilt**.
- Define **level of isolation** from other **containers/host**.



Containers

Docker API or CLI

- **Create** containers
- **Start** containers
- **Stop** containers
- **Move** containers
- **Delete** containers

Links

- Tutorial video:
  `https://www.youtube.com/watch?time_continue=722&v=fqMOX6JJhGo&feature=emb_logo`
- Cheat sheet:
  `https://kapeli.com/cheat_sheets/Dockerfile.docset/Contents/Resources/Documents/index`
- Follow the tutorial:
  `https://docs.docker.com/get-started/part1/`