

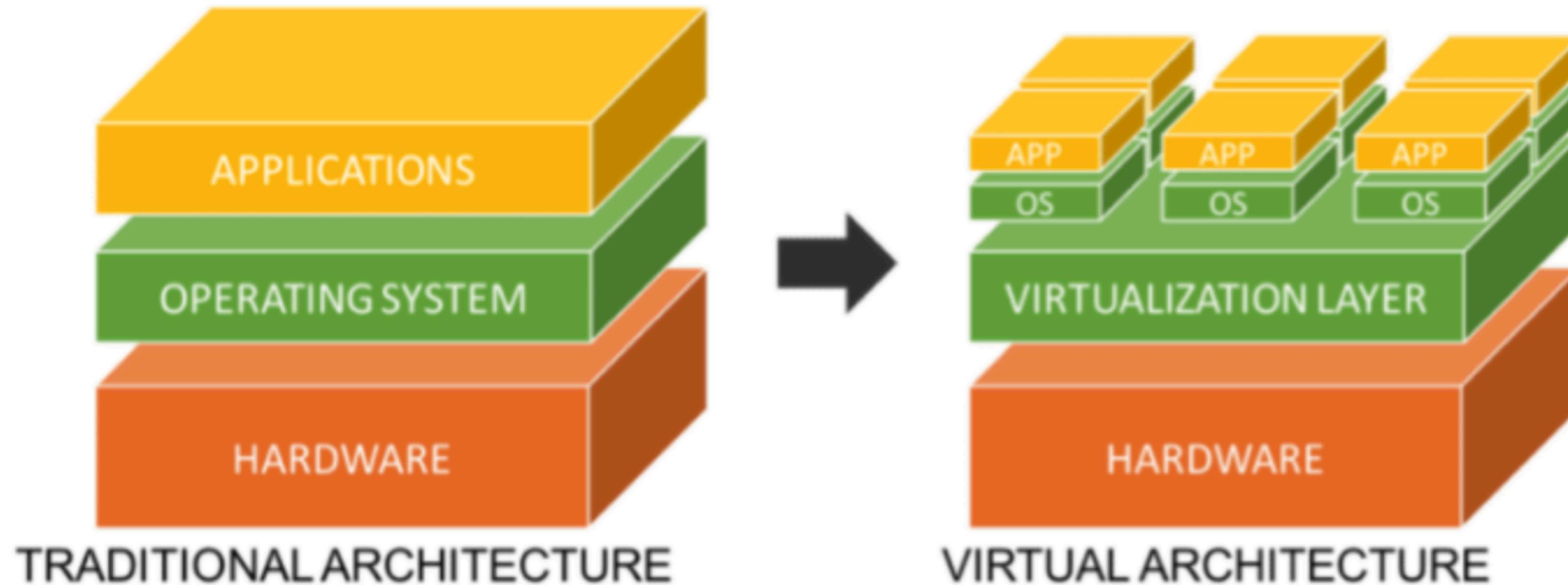
Kubernetes study #1

Virtualization & Docker

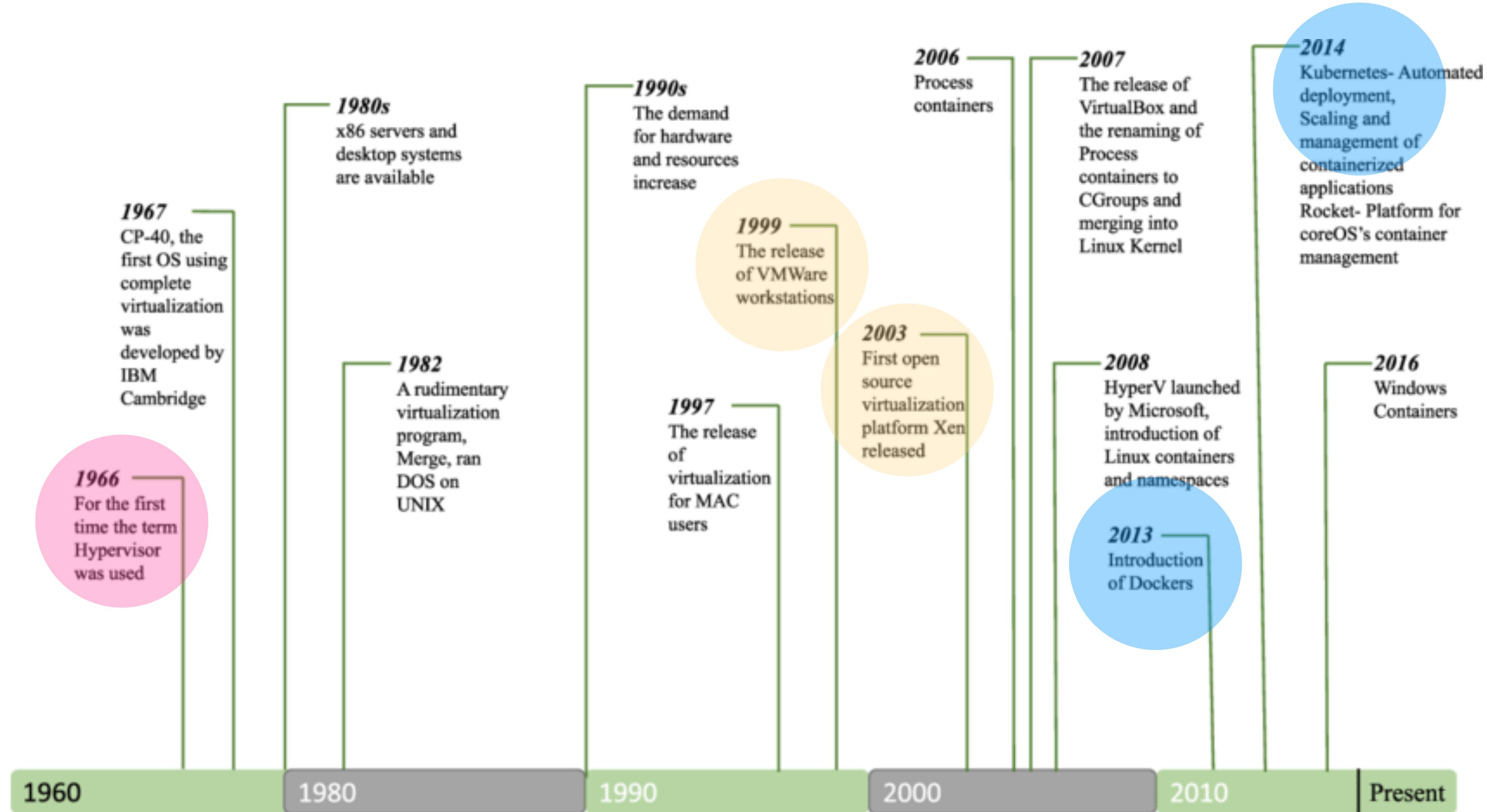
2021.06.01. Sieun Park

Virtualization Background

- Moore's Law: HW 성능의 성장 속도에 비해 SW의 성장 속도가 낮음 -> HW 자원 사용률 낮은 문제
- 복잡한 컴퓨팅 환경으로 인프라 운영 비용 증가
- 하나의 자원을 여러 사용자가 같이 사용할 수 없을까?

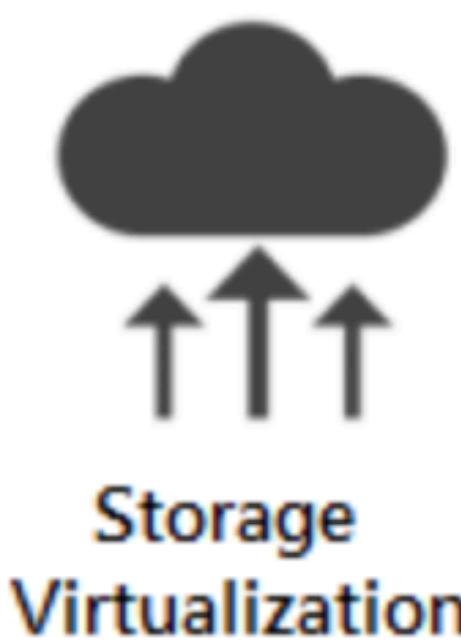
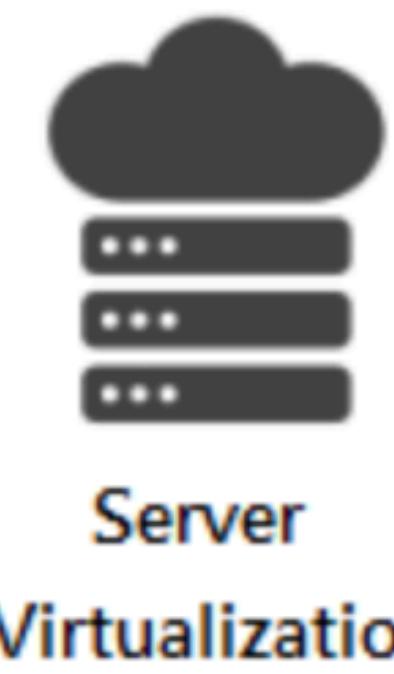


Virtualization History



Virtualization

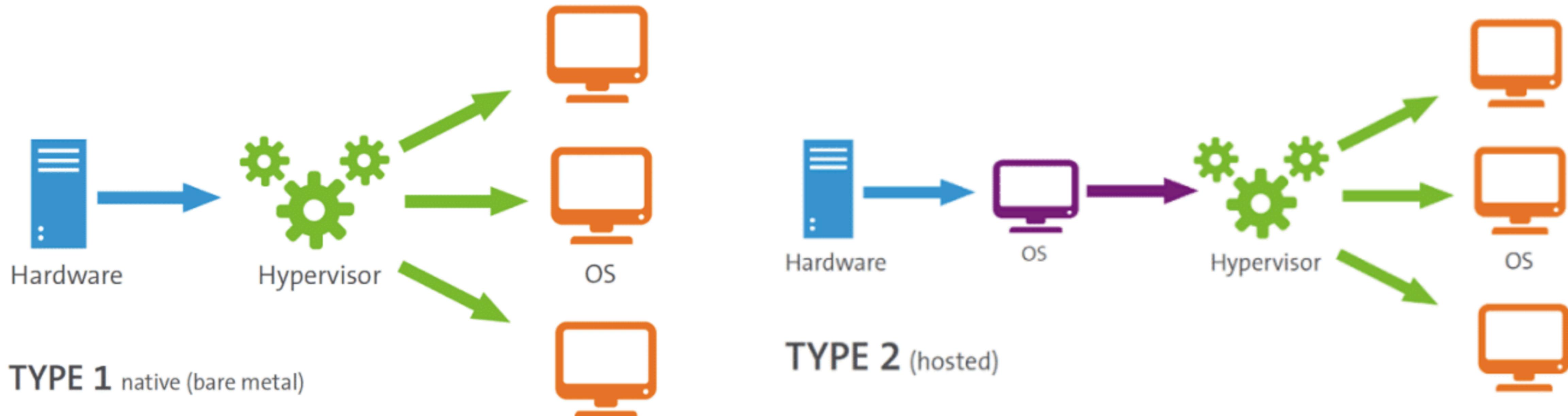
- 물리적인 자원을 논리적인 객체로 추상화 하는 것
- 하나의 장치를 여러 개처럼 동작 시키거나,
반대로 여러 개의 장치를 묶어 하나의 장치인 것처럼 사용할 수 있는 기술
- 가상화 대상: CPU, Memory, Storage, Network, GPU



Types of Virtualization

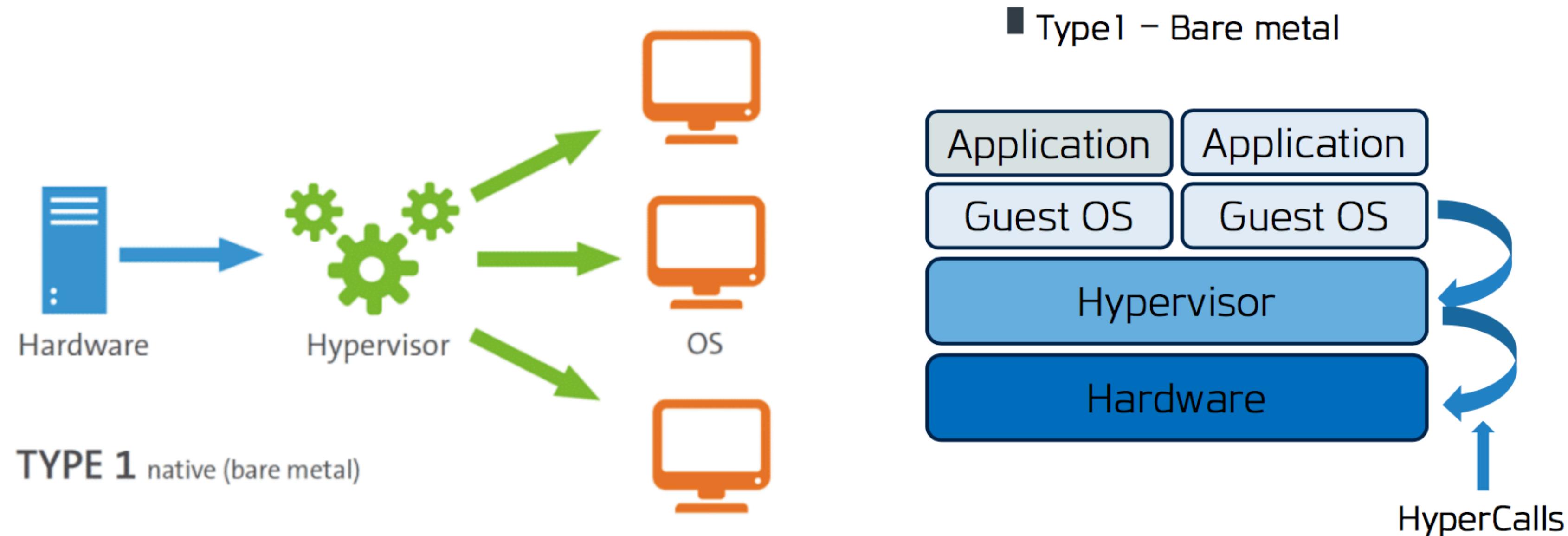
Hypervisor

- **Hypervisor:** 물리적 자원과 VM 사이를 연결시켜주는 중간 관리자
- **Hypervisor의 역할**
 - 가상화 층을 구현하여 물리적 리소스로부터 가상 환경을 분리하고 VM을 생성한다.
 - CPU, 메모리, 스토리지 등과 같은 물리적 컴퓨팅 리소스를 VM에 할당한다.
- **Hypervisor의 종류**



Type-1 (Bare-Metal) Hypervisor

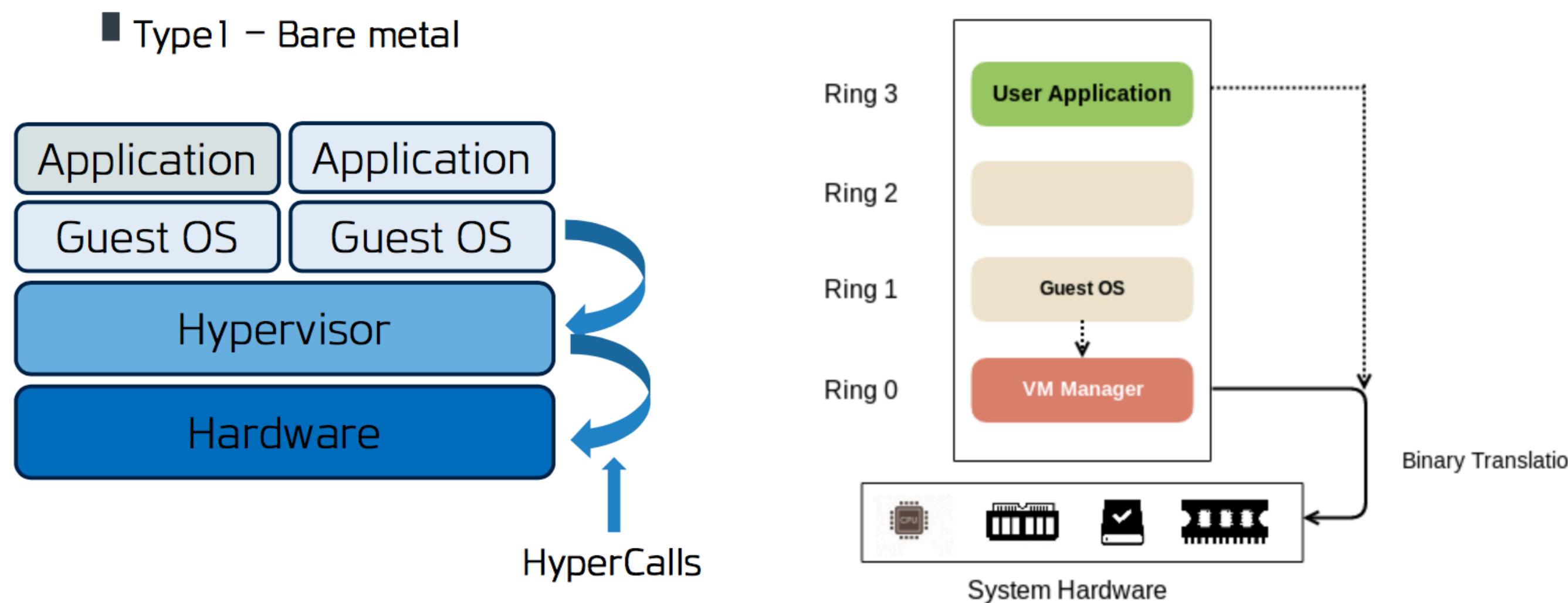
- Bare-Metal HW 위에 Hypervisor가 직접 구동되고, 그 위에 Guest OS 가 올라가는 방식
- VMware의 ESXi, Citrix의 Xen, MS의 Hyper-V 등



- 장점: Bare-Metal 을 관리할 OS 를 설치하지 않기 때문에 오버헤드가 적음
- 단점: 머신에 대한 자체적인 관리 기능이 없어, 관리에 필요한 컴퓨터나 콘솔이 필요하다.
- 전가상화(Full-Virtualization) 와 반가상화(Para-Virtualization)으로 분류

Type-1 Full-Virtualization (1)

- 하드웨어를 **완전히 가상화**하는 방식 = 각 Guest OS는 자신이 가상화되어 있다는 것을 알지 못한다.
- 장점: Guest OS를 수정할 필요 없이 사용할 수 있다.
- 단점: **하이퍼바이저가 모든 명령을 중재 (Trap & Emulate) 하여 성능이 느리다**
- 전가상화의 모드
 - Hypervisor: root 모드
 - Guest OS & App : non-root 모드
 - 만약 Guest OS가 Privileged Instruction을 처리해야하는 상황이 오면, Trap & Emulate 방식으로 처리



Type-1 Full-Virtualization (2)

- Trap & Emulate

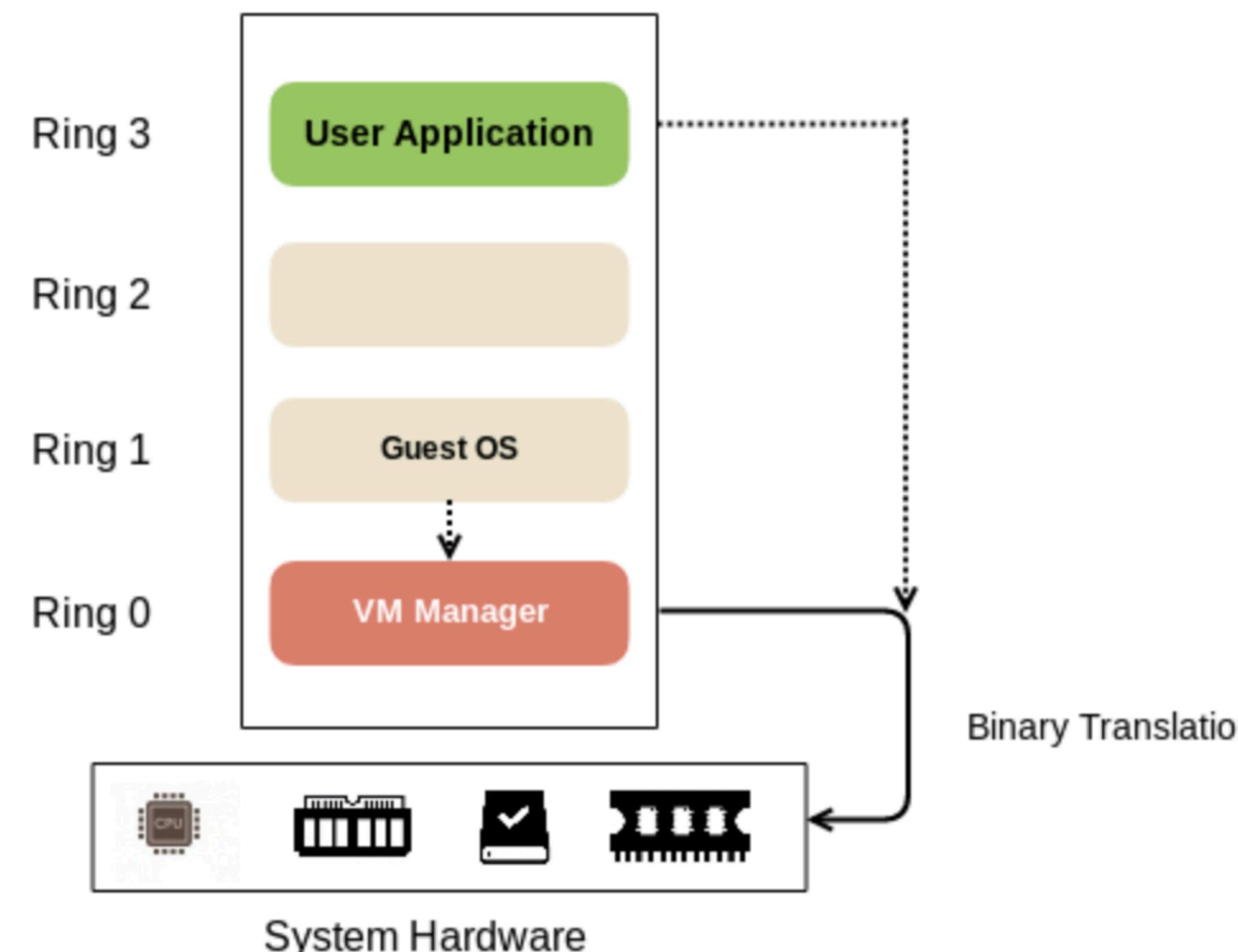
- Trap: 특권 명령을 실행할 권한이 없는 Guest OS에서 특권 명령을 실행할 때 시스템에 발생하는 예외
- Guest OS에 Trap이 발생하면, Trap handler는 VM exit를 통해 제어권을 Hypervisor에 넘긴다.
- Hypervisor가 해당 명령을 처리한다. (Emulate)
- 해당 명령의 처리가 끝나면, VM enter를 통해 실행 결과와 제어권을 Guest OS에 넘겨준다.
- Trap & Emulate 방식은 **시스템 오버헤드가 너무 크다.**



Type-1 Full-Virtualization (3)

- **Binary Translation (이진변환)**

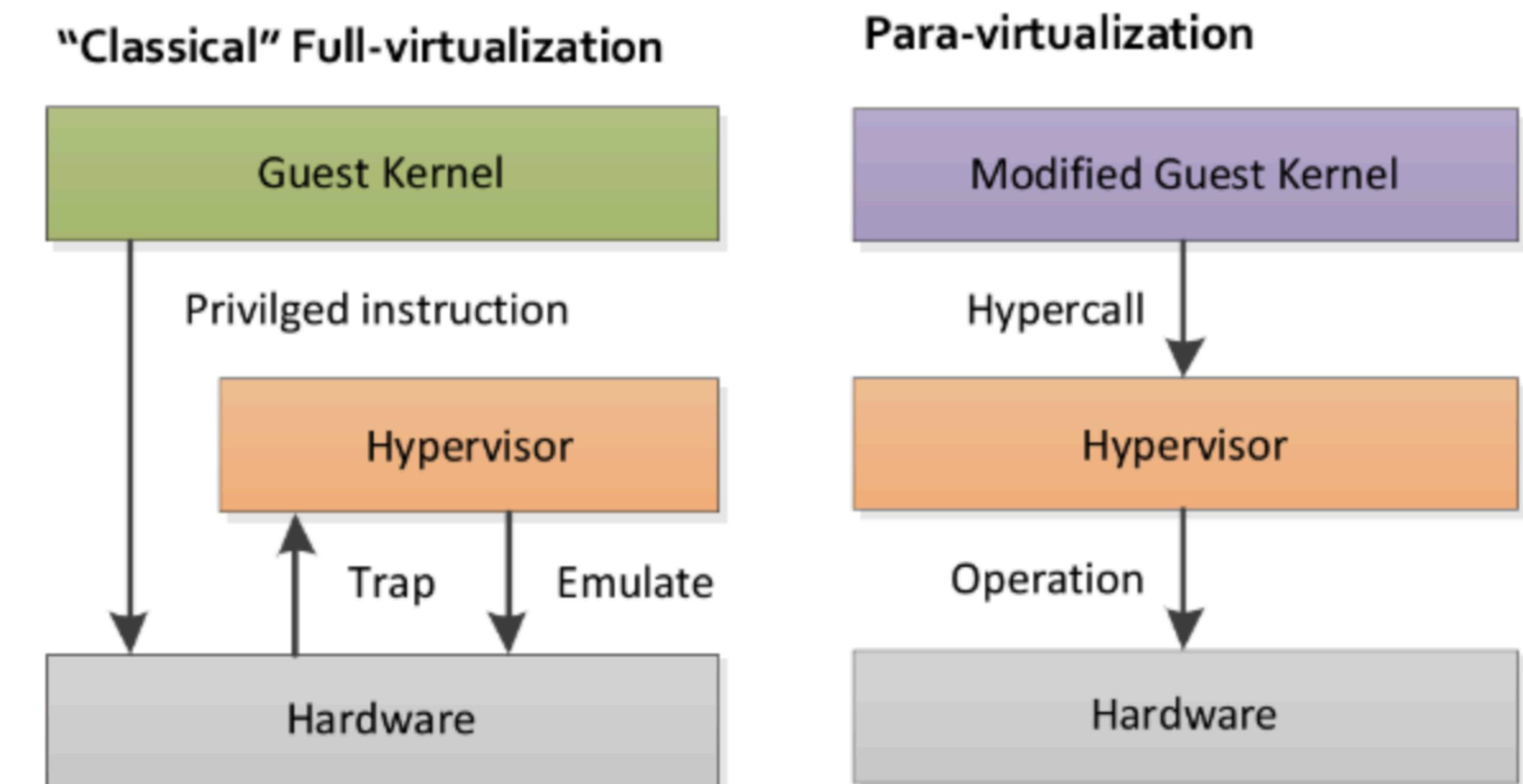
- Trap & Emulate의 단점을 해결하기 위해 VMware에서 선보인 기술
- Guest OS에서 특권 명령을 수행하려고 할 때 Hypervisor가 바이너리 연산을 통해서 하드웨어가 인식할 수 있는 명령어로 변환하여 전달하는 기법 (번역!)
- Binary Translation은 CPU에서 직접 실행하는 방식이지만, 중간에 하이퍼바이저가 번역하는 과정이 추가



Type-1 Para-Virtualization

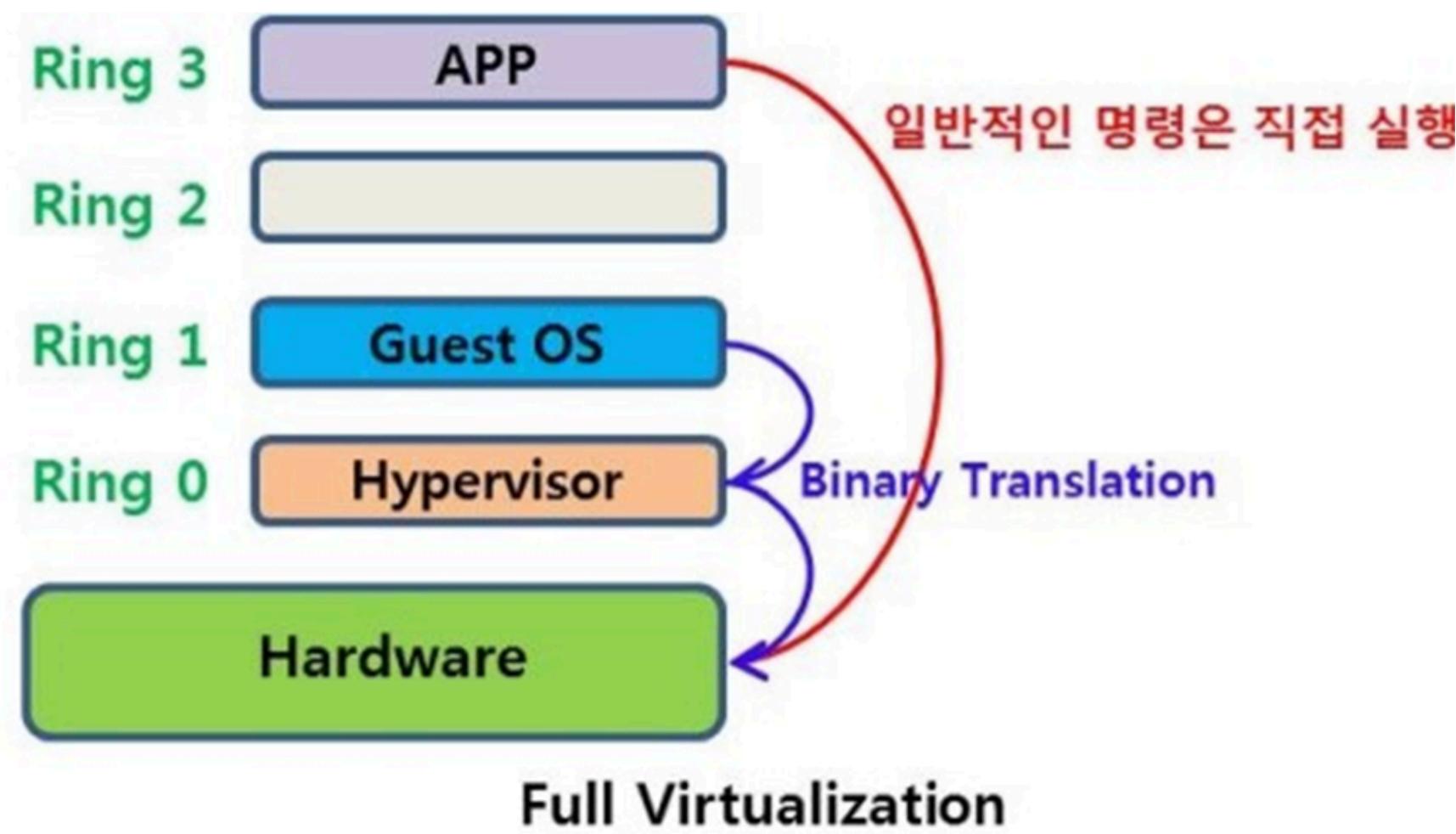
- 전가상화의 성능적인 문제를 해결하기 위해 고안한 기술로, **하드웨어를 완전히 가상화하지 않는다.**
- Guest OS는 스스로 가상화된 환경임을 인지하고 있으며,
특권 명령을 실행할 때 **HyperCall**이라는 인터페이스로 직접 hypervisor에 요청을 날린다.
- Hypervisor에 Hyper Call을 보내는 동작은 기존 OS들은 할 수 없으므로,
HyperCall을 보낼 수 있도록 **커널 수정, 드라이버 설치** 등이 필요하다.

- 장점: 전가상화에 비해 성능이 빠르다.
- 단점: Guest OS 수정이 필요하다.

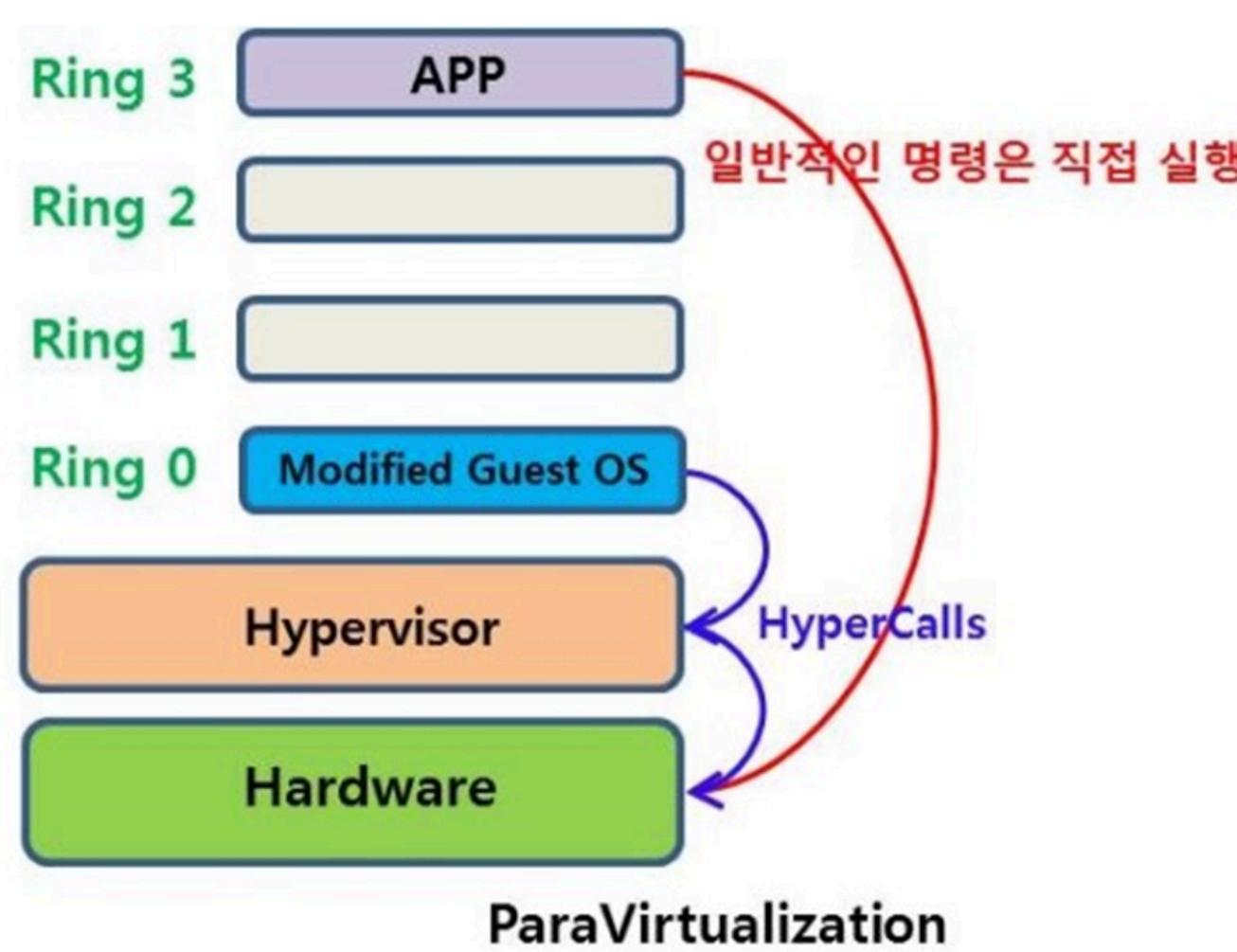


Type-1 Summary

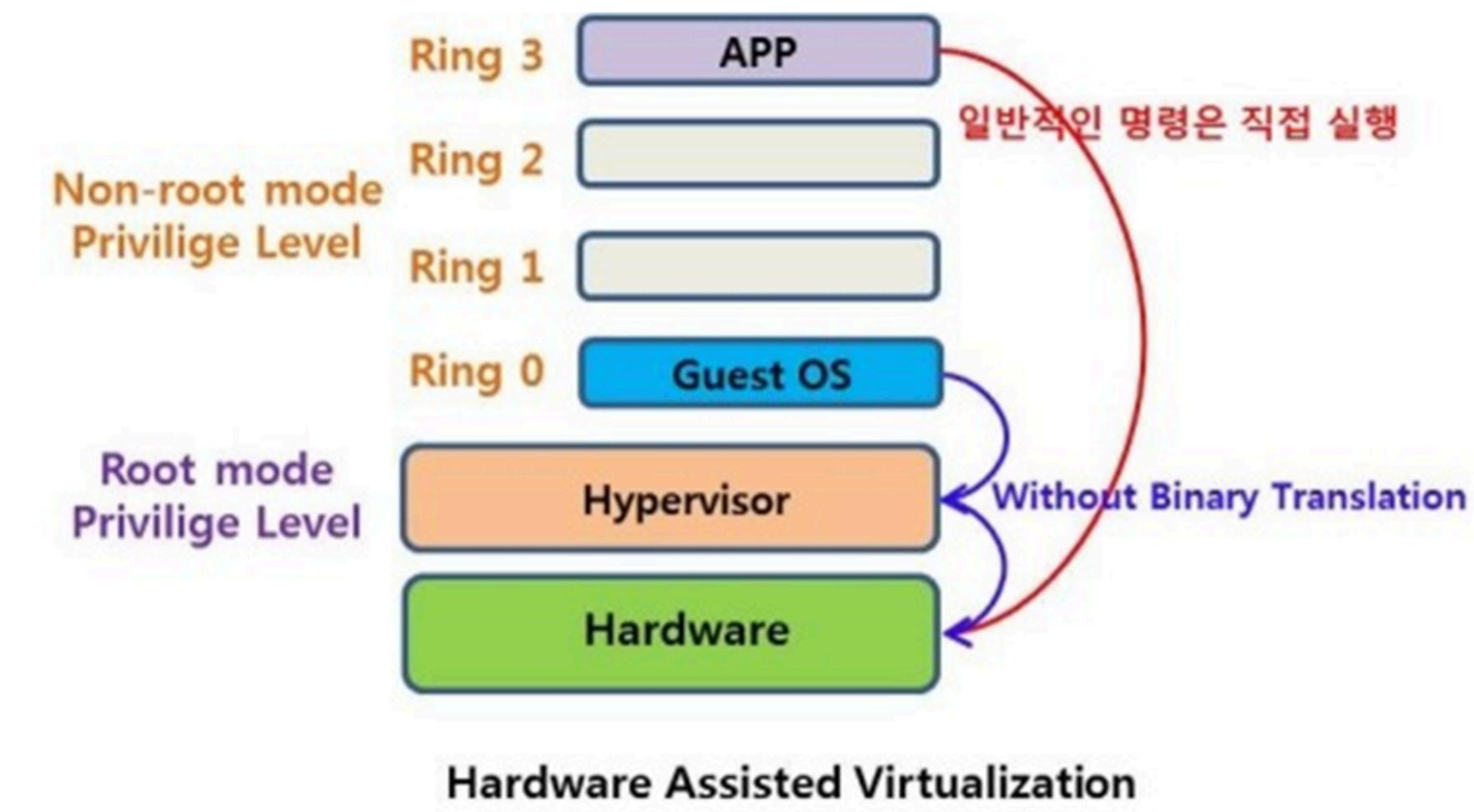
Full-Virtualization



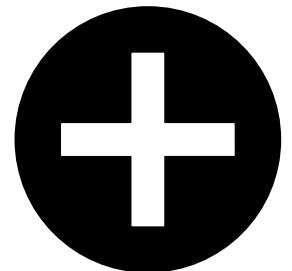
Para-Virtualization



Hardware Assisted Virtualization



- Hypervisor가 일한다.
- 장점: Guest OS에 제약이 없다!
- 단점: Hypervisor 성능 저하



VMware – EXSi

- Guest OS가 일한다.
- 장점: 성능이 좋다.
- 단점: Guest OS 수정 필요

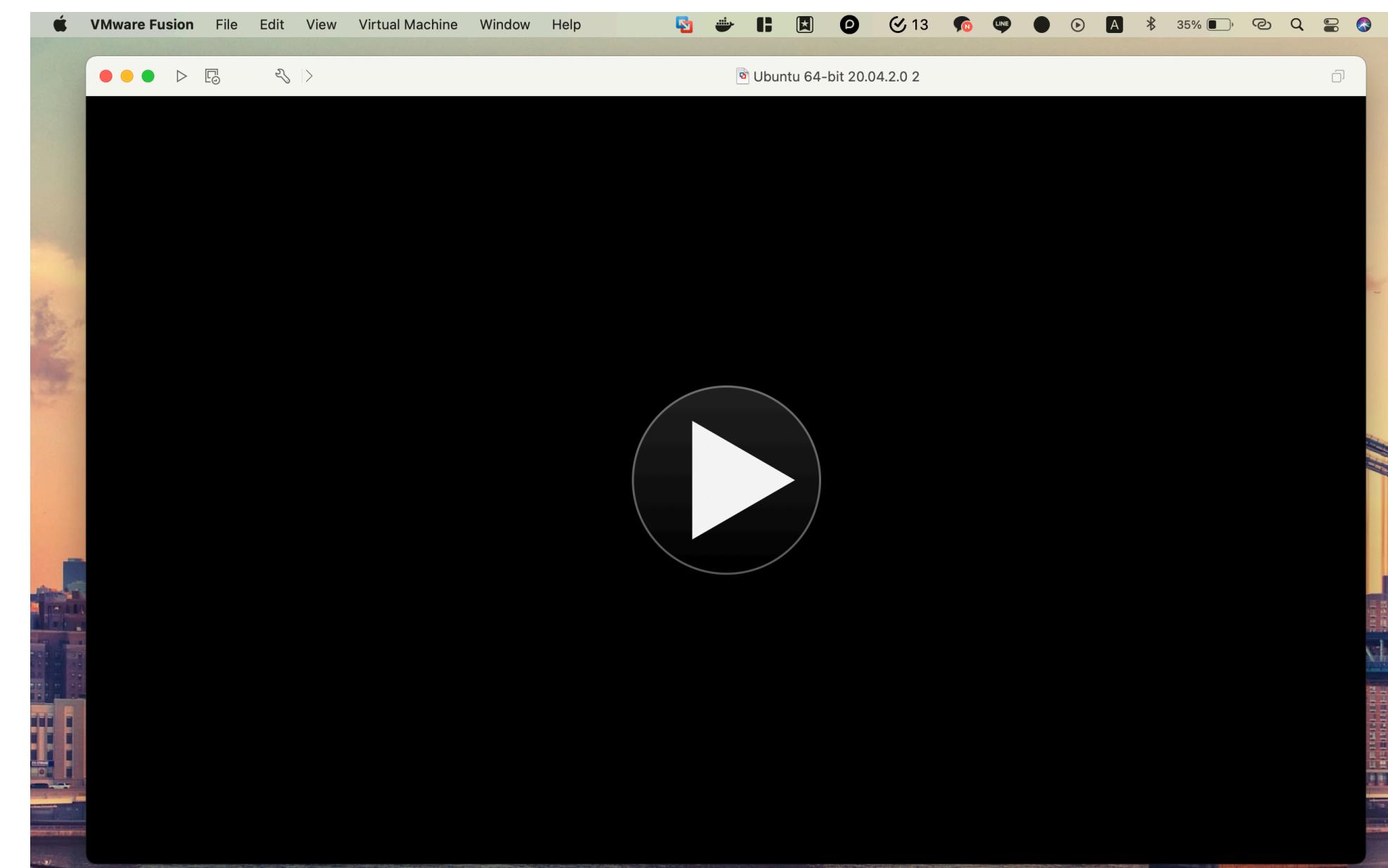
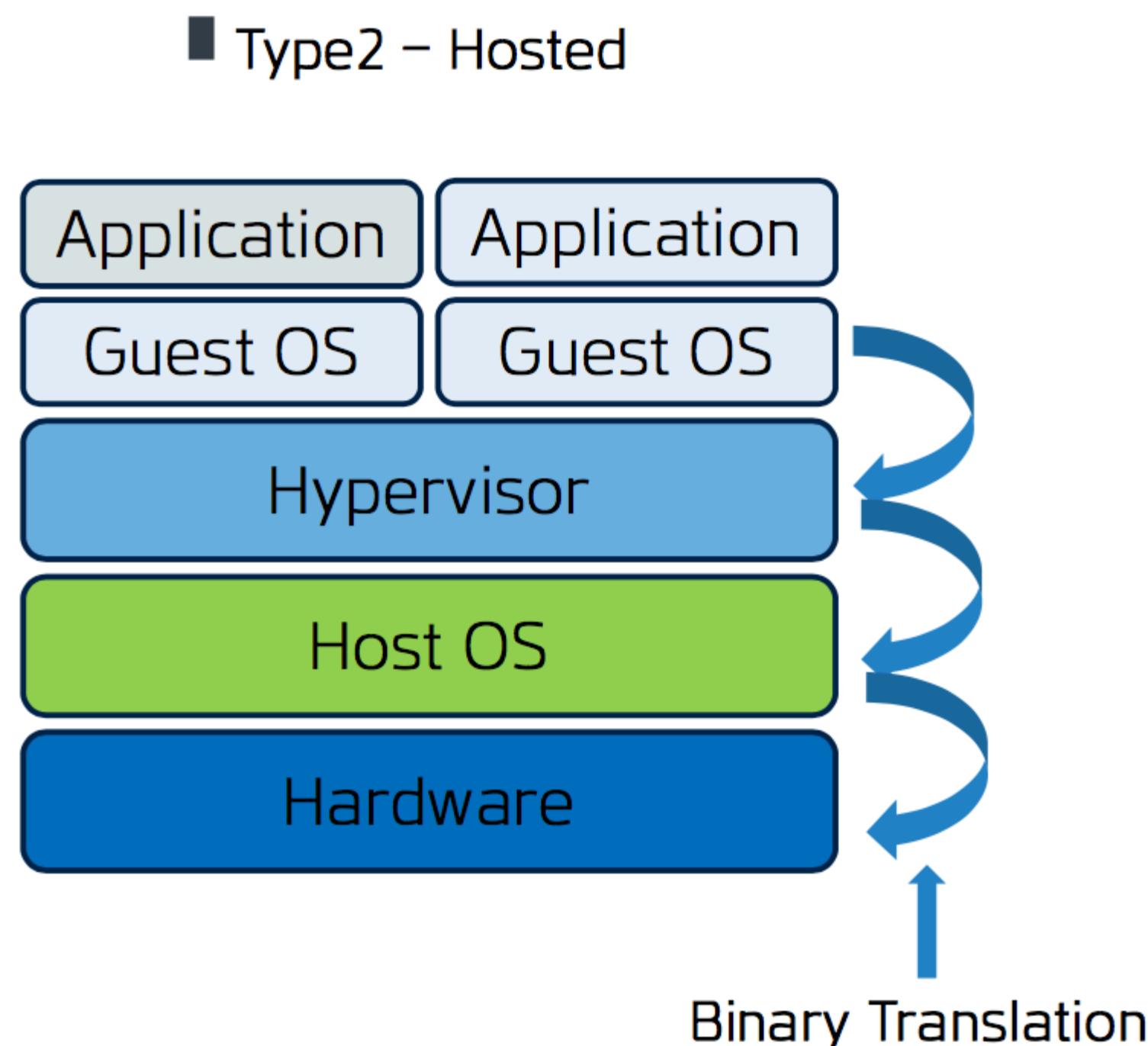
Citrix – Xen

- CPU가 일하자!
- Guest OS 제약 없고, 성능 저하도 없음
- 다만, CPU가 가상화를 지원해야 함

Intel – VT-x / AMD – AMD-V

Type-2 (Hosted) Virtualization

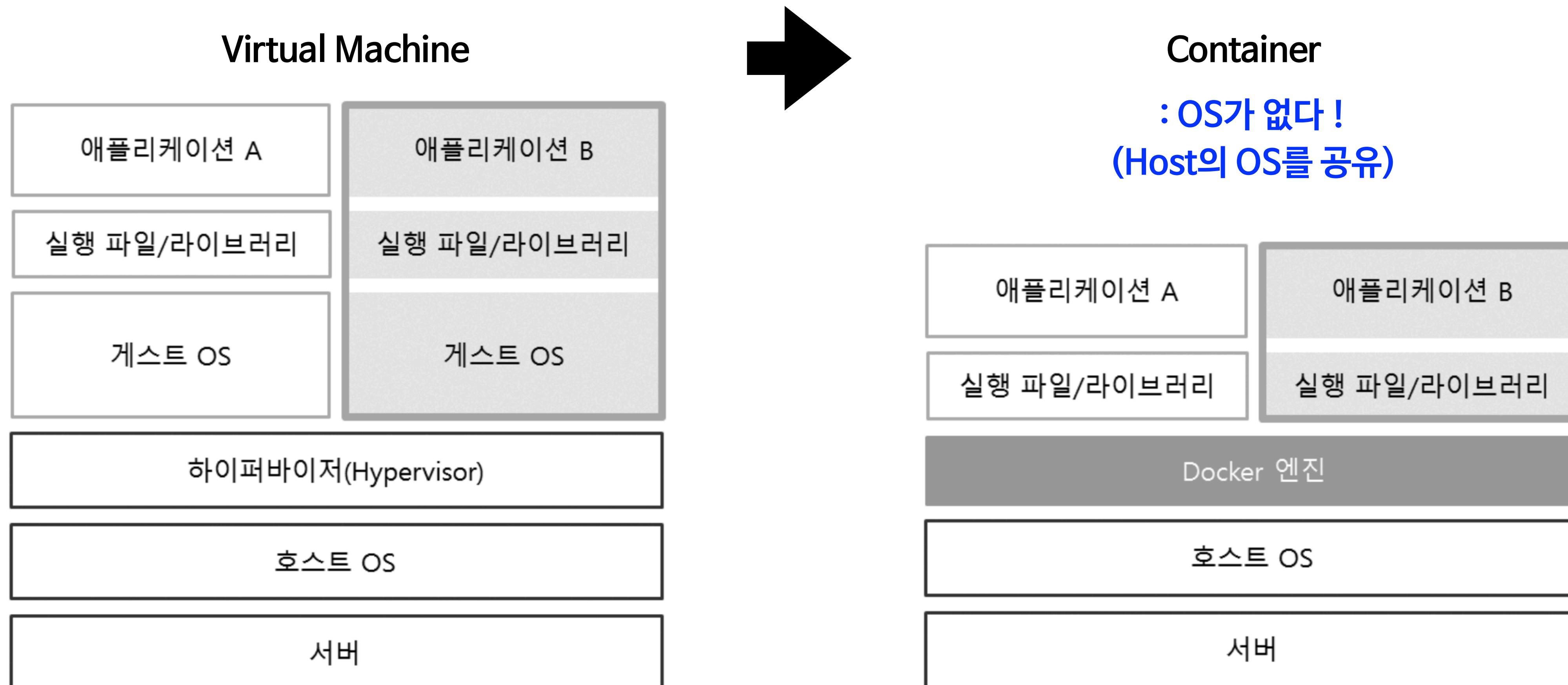
- Bare-Metal을 구동하기 위한 Host OS가 설치 되고, 그 위에 Hypervisor가 실행되는 형태
- Host OS 위에 Hypervisor가 구동되고, 그 위에서 VM을 에뮬레이팅 하기 때문에 오버헤드가 크다.
- Host OS에 크게 제약이 없고, 기존 OS를 사용하는 시스템에 쉽게 사용 가능
- Oracle의 Virtual Box, VMware Workstation, MS의 Virtual PC 등



OS 위의 OS

Disadvantage of VMs

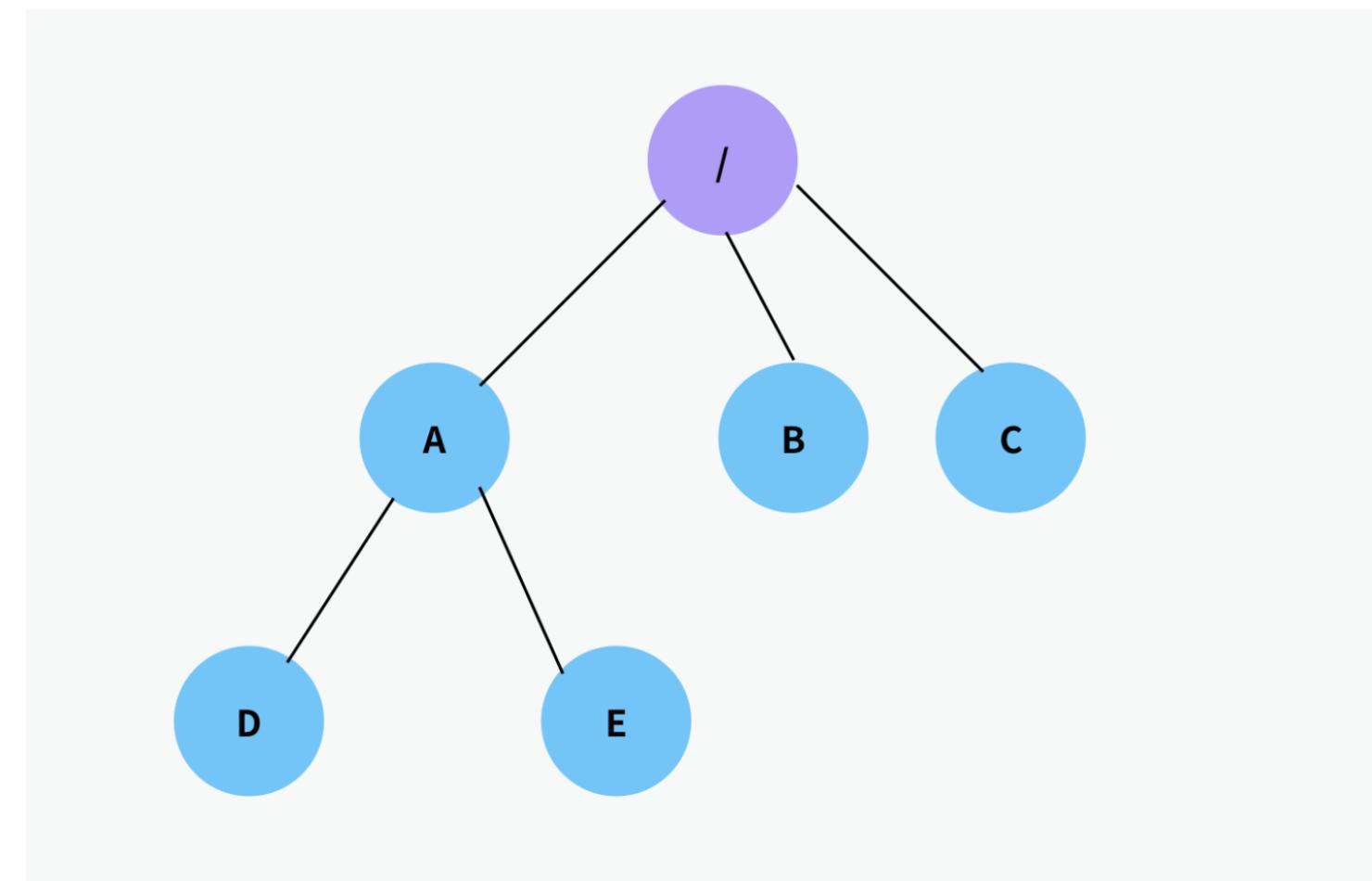
- VM은 OS를 포함하기 때문에 이미지 용량이 크다.
- VM은 OS 가상화에 포커스가 있어 배포와 관리가 어려울 수 있다.
- OS 격리 없이 프로세스만 격리할 수 있을까?



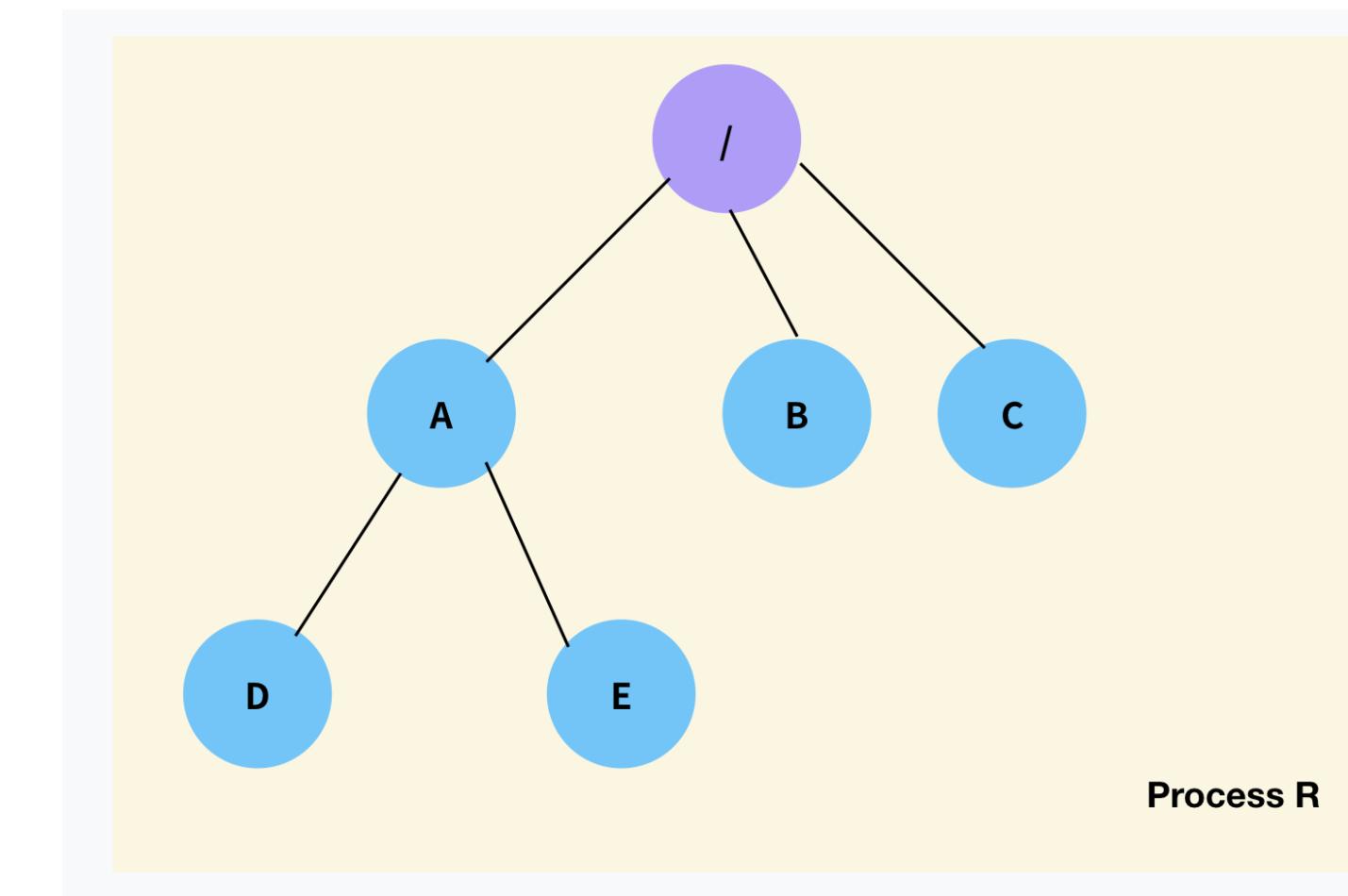
Linux Container (1)

- Chroot (Change root directory)

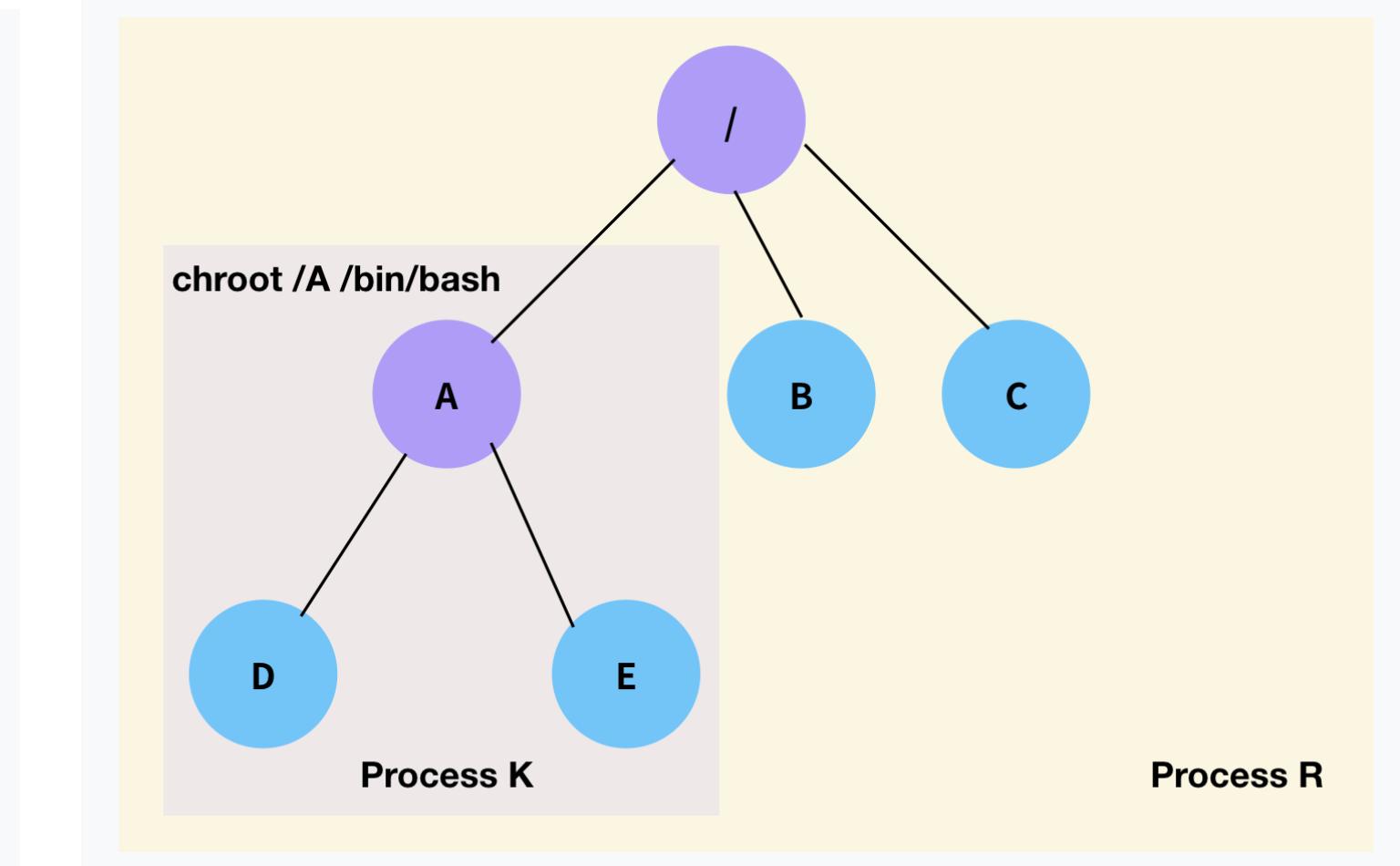
- Linux 파일 시스템에서 root dir(/)를 변경하는 명령
 - root dir: 파일 시스템의 최상위. 모든 디렉터리와 파일은 이 루트 디렉터리 아래에 존재한다.
- chroot로 특정 디렉터리를 루트 디렉터리로 설정하면 chroot jail이라는 환경이 생성
 - chroot jail안에서는 바깥의 파일과 디렉터리에 접근할 수 없다.
- 디렉터리 경로를 격리하여 서버 정보 유출과 피해를 최소화하는데 주로 사용



A File system



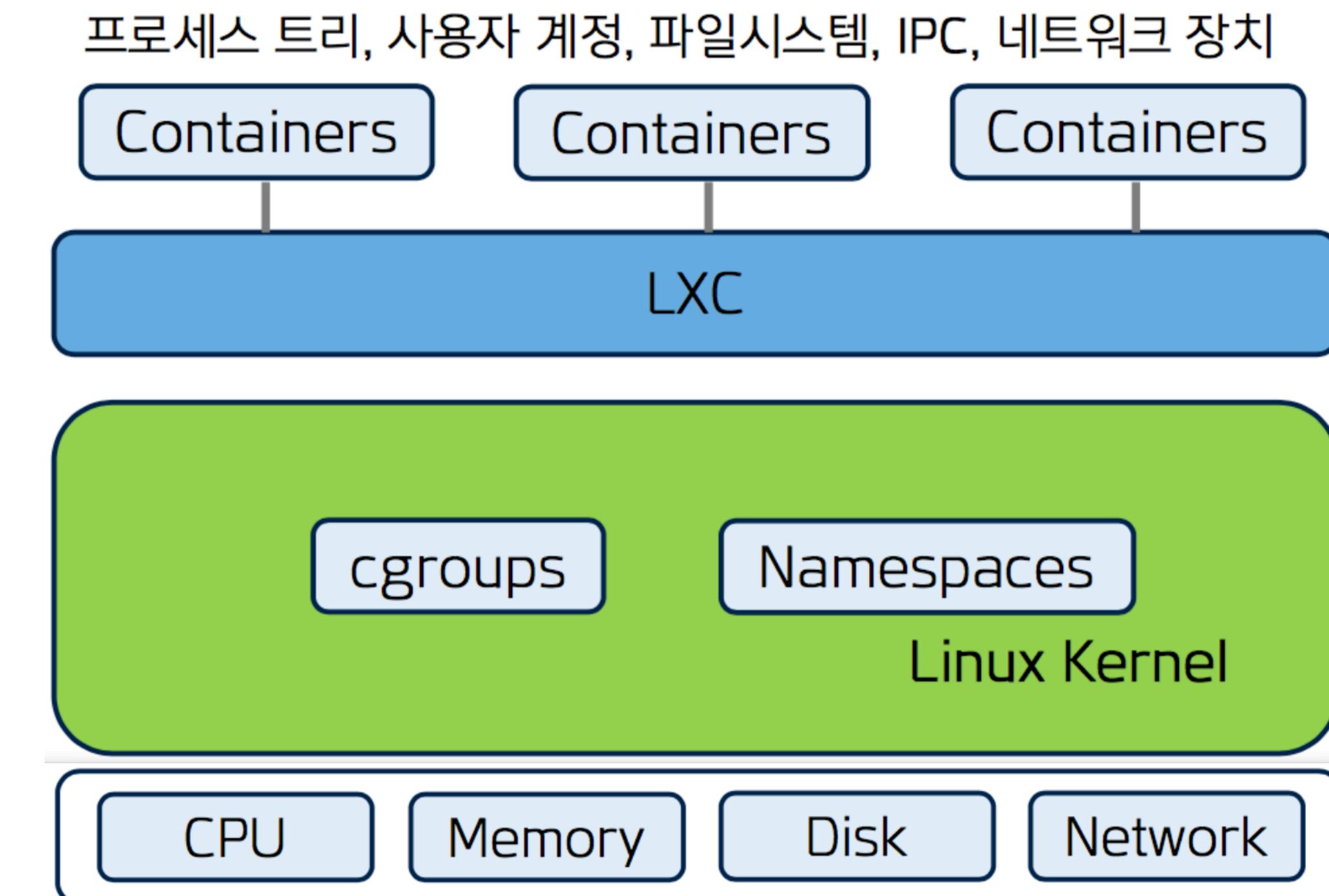
Process R이 실행되면
root(/)부터 파일 탐색 시작



Process K를 root를 A로 시작할 경우
A의 하위 dir에만 접근 가능

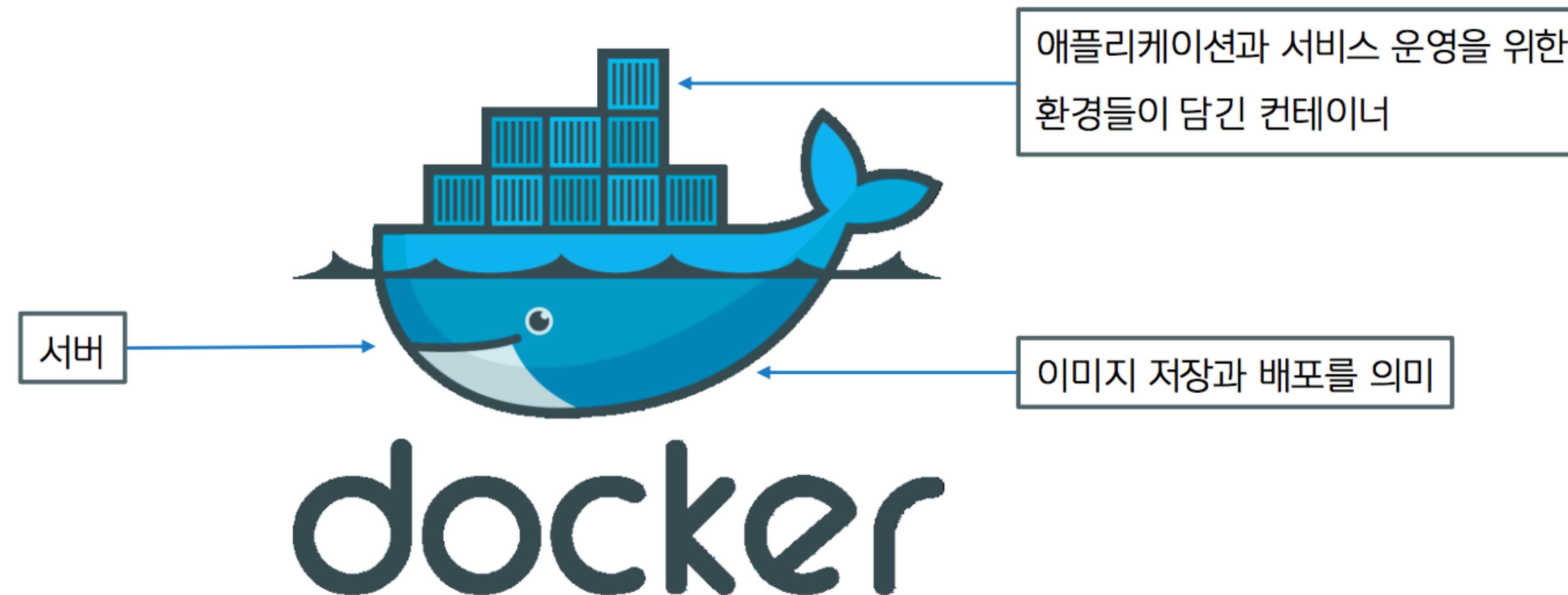
Linux Container (2)

- Chroot를 발전시켜 LXC(LinuX Container)라는 시스템 레벨 가상화 제공
- Namespaces와 cgroups를 통해 격리된 컨테이너 생성
 - Namespaces: 프로세스 트리, 사용자 계정, 파일시스템, IPC, 네트워크 장치 등의 OS을 고립
 - cgroups(control groups): 각각 고립된 환경에 CPU, Memory, disk, network와 같은 자원을 할당
- LXC는 컨테이너의 생성 및 관리, 배포에 대한 부가 기능이 없음



Docker

- 컨테이너의 생성 및 관리를 쉽게 해주고, 컨테이너를 이미지화하여 배포를 손쉽게해주는 Docker 의 등장
- 서비스 운영환경을 뚫어서 손쉽게 배포하고 실행하는 경량 컨테이너 기술
- Docker가 처음 개발될 당시에는 LXC를 기반으로 구현했지만,
버전 0.9부터는 LXC를 대신 자체 기술인 libcontainer를 개발하여 사용하고 있다.



Immutable Infrastructure

- **Mutable Infrastructure (변경 가능한 인프라)**

- 기존에는 서버를 교체하는 비용이 커짐
- 한번 인프라를 구축한 후에는 끊임없이 수정하고 유지 보수하면서 최대한 오랫동안 사용하는 것이 일반적
- 인프라 구축 규모가 클수록 인프라 관리에 대한 부담 증가

- **Immutable Infrastructure (변경 불가능한 인프라)**

- 서버 생성 후 업데이트하지 말자 (고장나면 새것으로 교체하자)
- 변경이 필요시 서버를 완전히 새로 구성하여 배포하고 기존 서버는 폐기한다.
- 장점
 - 서비스 운영을 위해서 이미지만 관리하면 되어 관리가 용이
 - 이미지 하나로 서버를 계속 찍어낼 수 있어 확장성이 좋다.
 - OS를 제외한 서비스 운영환경 이미지만 갖고 있어 가볍다.

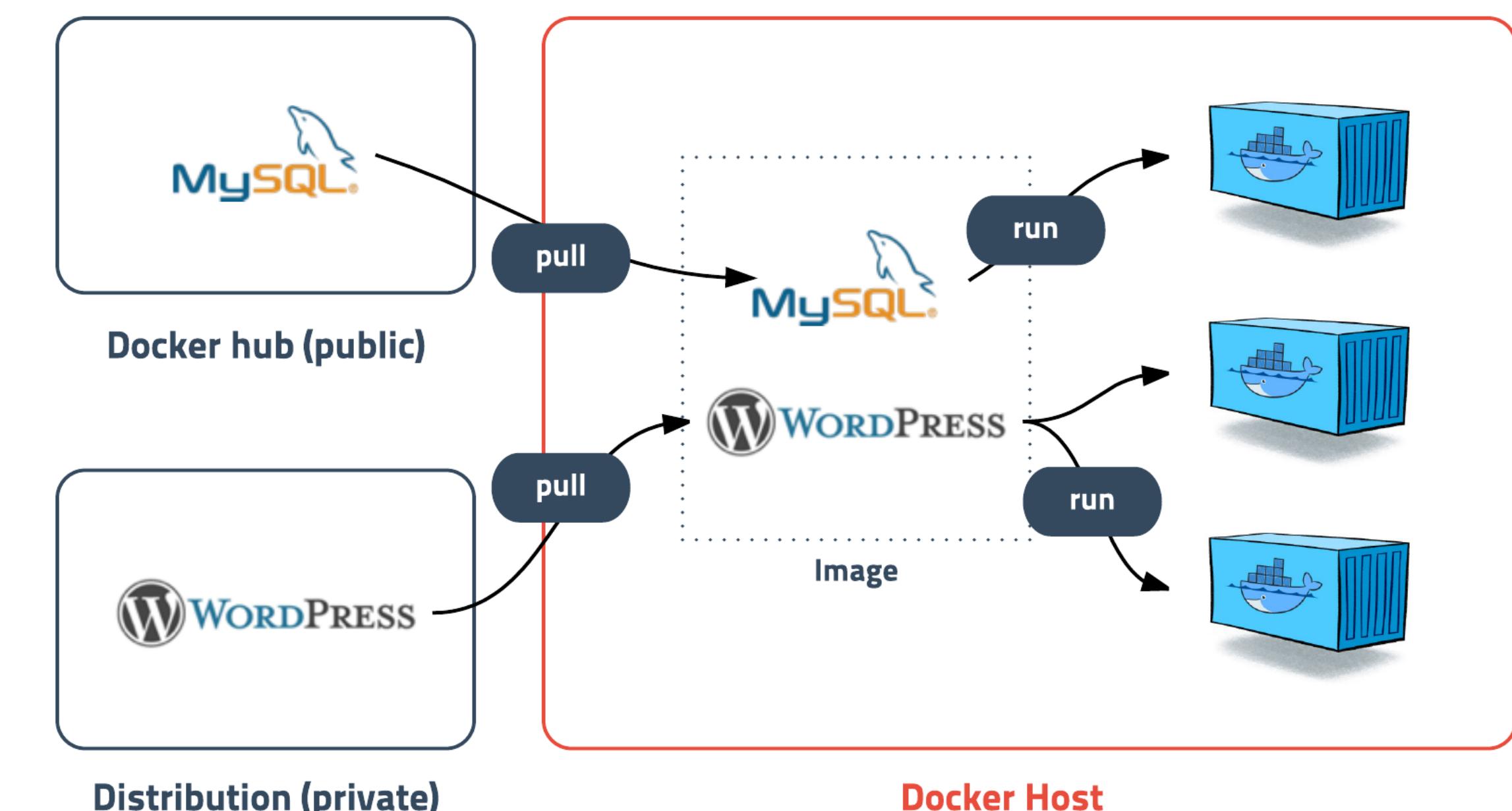
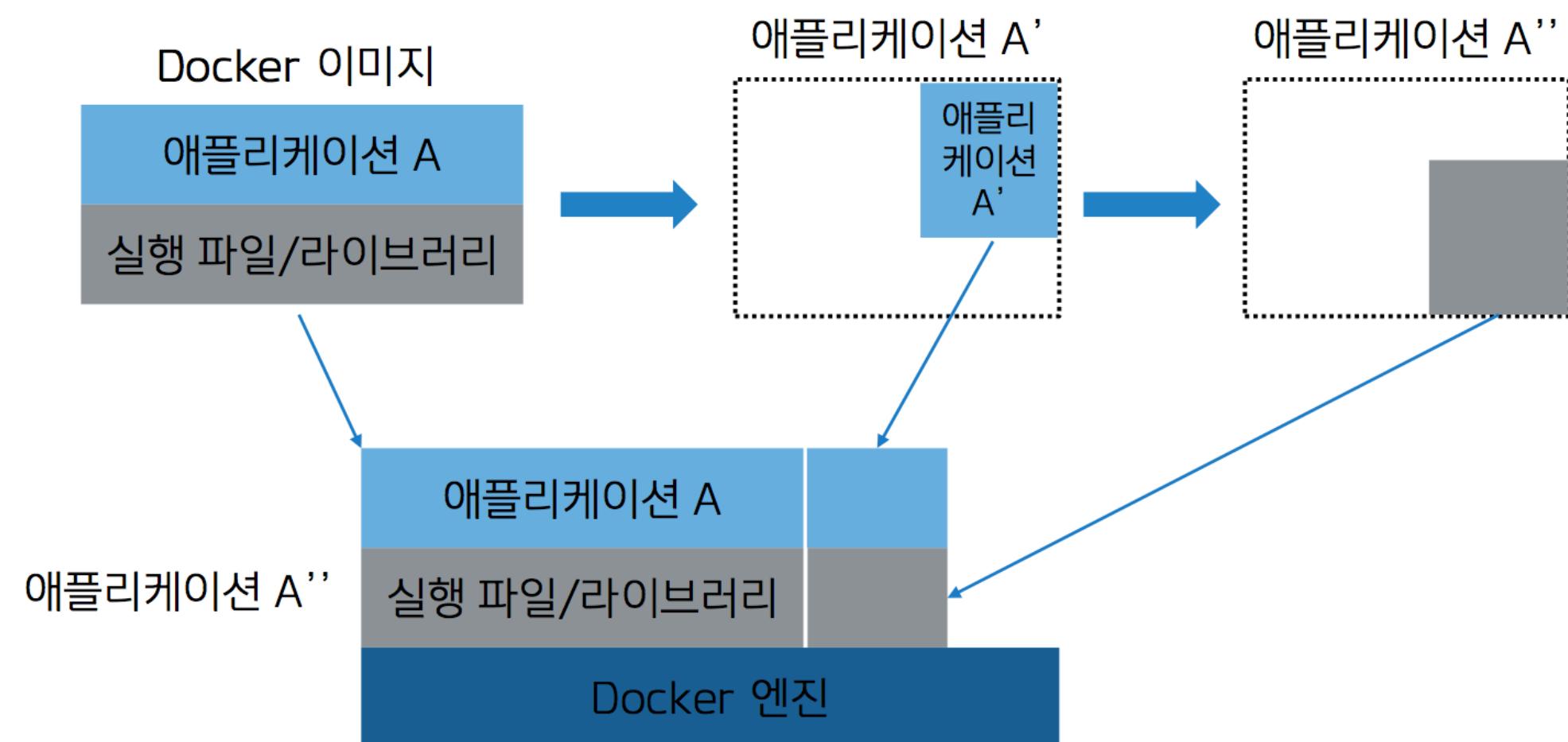
Docker Image (1)

- **Docker Image**

- 컨테이너 실행에 필요한 모든 파일과 설정값 등을 포함하고 있으며, 변하지 않는다 (Immutable)
- 새로운 서버가 추가되면 미리 만들어 놓은 이미지를 다운받고 컨테이너를 생성만 하면 됩니다.

- **Container**

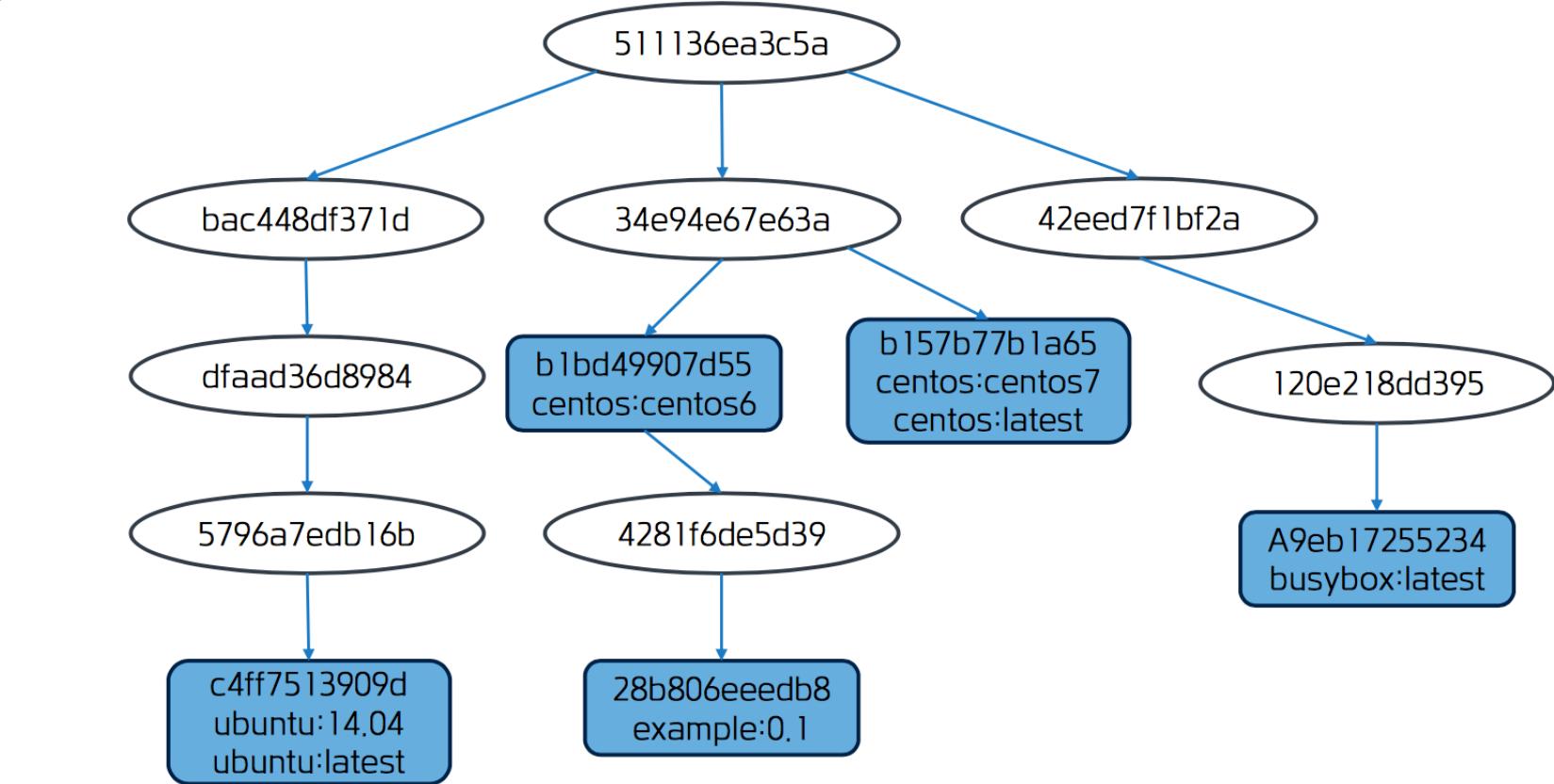
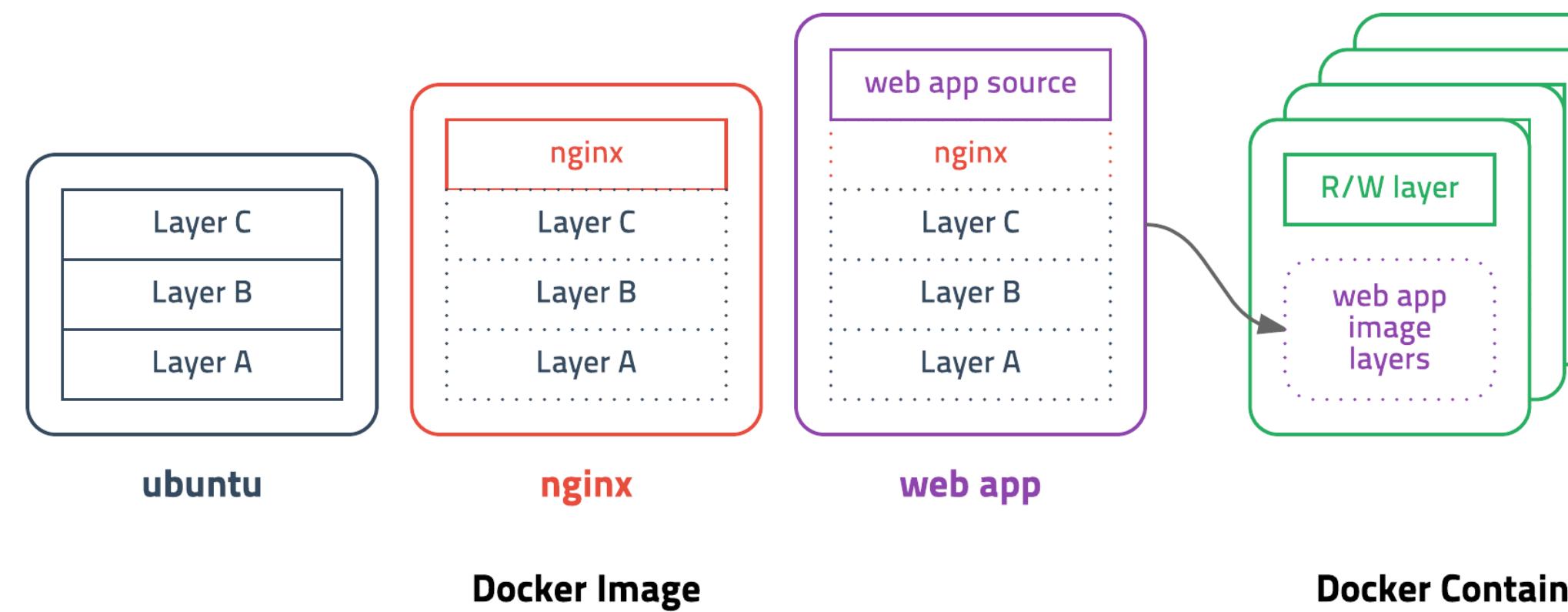
- 이미지를 실행한 상태. 추가되거나 변하는 값은 컨테이너에 저장된다.
- 같은 이미지에서 여러개의 컨테이너를 생성할 수 있다.
- 컨테이너의 상태가 바뀌더라도 이미지는 바뀌지 않는다



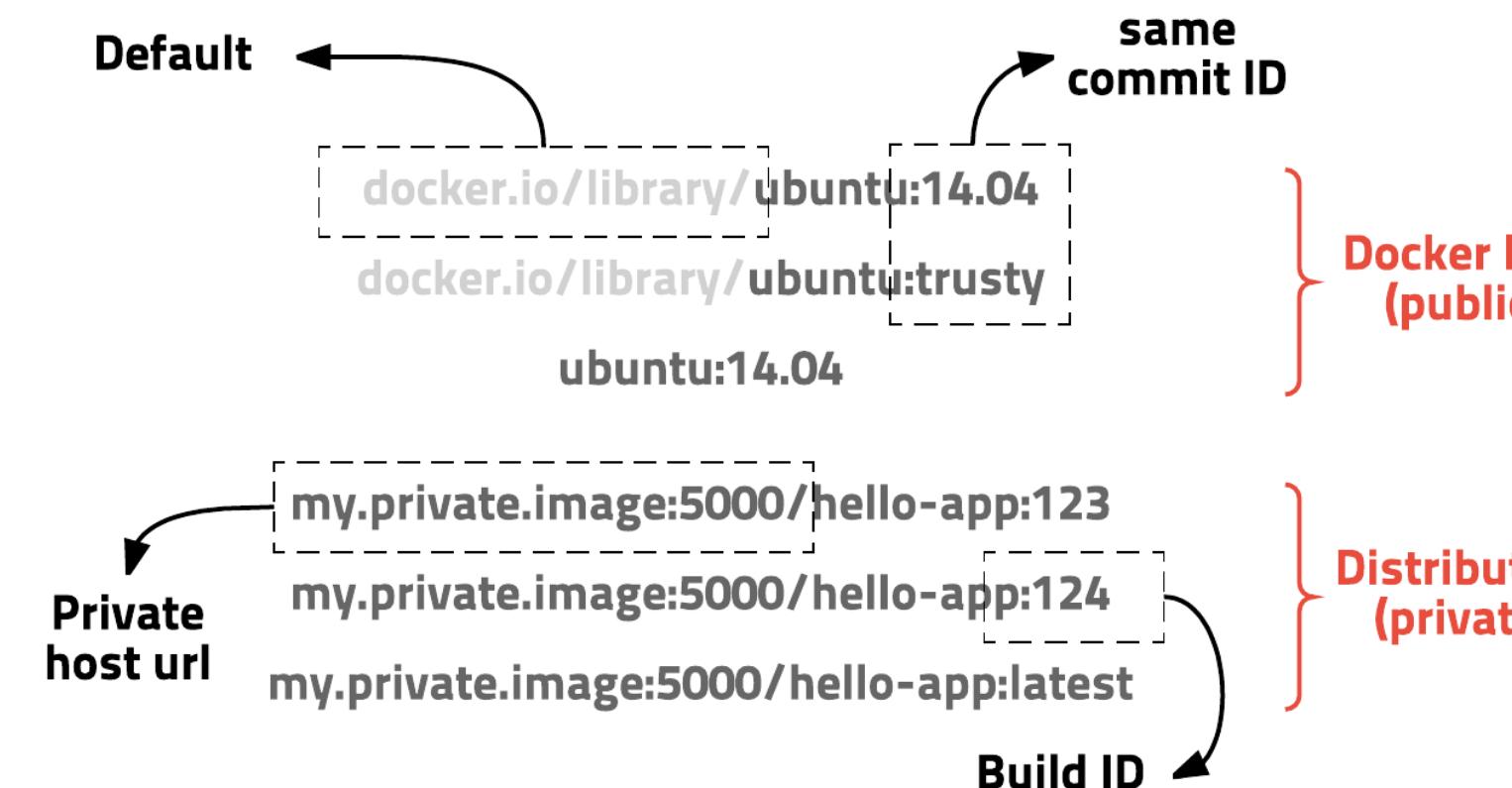
Docker Image (2)

- **Image Layer**

- Docker는 이미지를 통째로 생성하지 않고, 바뀐 부분만 생성한 뒤 부모 이미지를 계속 참조한다. (Layer)
- Docker가 실행될 때는 베이스 이미지와 바뀐 부분을 합쳐서 실행



- **Image Path**



Dockerfile

- **Dockerfile**
 - 이미지를 빌드하기 위해 필요한 파일
 - Base image 지정 / 원하는 SW 및 라이브러리 설치 명령 / 컨테이너 실행 시 수행할 명령 등

```
#Dockerfile
FROM ubuntu:18.04

RUN apt-get update \
    && apt-get install -y \
        curl \
        python-dev

WORKDIR /root
COPY hello.py .
ENV my_ver 1.0

CMD ["python", "hello.py", "guest"]
```

- **FROM:** Base image 지정
- **RUN:** 사용자가 지정한 명령을 실행
- **WORKDIR:** 이미지의 작업 폴더 지정
- **COPY:** 로컬 호스트의 파일을 이미지 안으로 복사
- **ENV:** 이미지의 환경변수 지정
- **CMD:** 이미지 실행 시 default로 실행되는 명령 지정

실습

- **Image** 생성
 - 작성한 Dockerfile을 build 하면 이미지 생성 가능

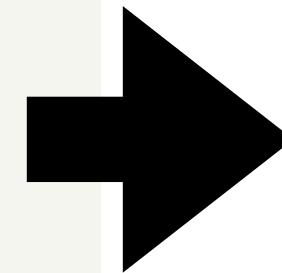
Dockerfile

```
#Dockerfile
FROM ubuntu:18.04

RUN apt-get update \
&& apt-get install -y \
curl \
python-dev

WORKDIR /root
COPY hello.py .
ENV my_ver 1.0

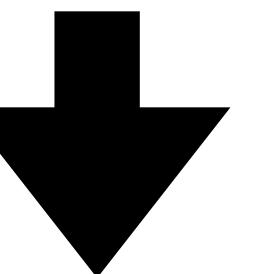
CMD ["python", "hello.py", "guest"]
```



Build Dockerfile

```
docker build [PATH] -t [IMAGE_NAME]:[TAG]
$ docker build . -t hello:1
```

```
#아래와 같이 이미지 생성된 것 확인
$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
hello               1        9bfffed3d7b67   34 hours ago  63.5MB
```



Create a Container

```
$ docker run hello:1
hello guest, my version is 1.0!
```