

Operating System

Subject 1. Virtual Machine

Subject 2. Container Basic

Subject 3. Chroot

Subject 4. Overlay filesystem

Subject 5. namespace

Subject 6. JVM warm-up

Reference

<https://www.youtube.com/watch?v=mSD88FuST80>

이게 돼요? 도커 없이 컨테이너 만들기(김삼영) / if(kakao)2022

<https://www.youtube.com/watch?v=utjn-cDSiog>

JVM warm up / if(kakao)dev2022

Operating System Concepts (10/E, Silberschatz)

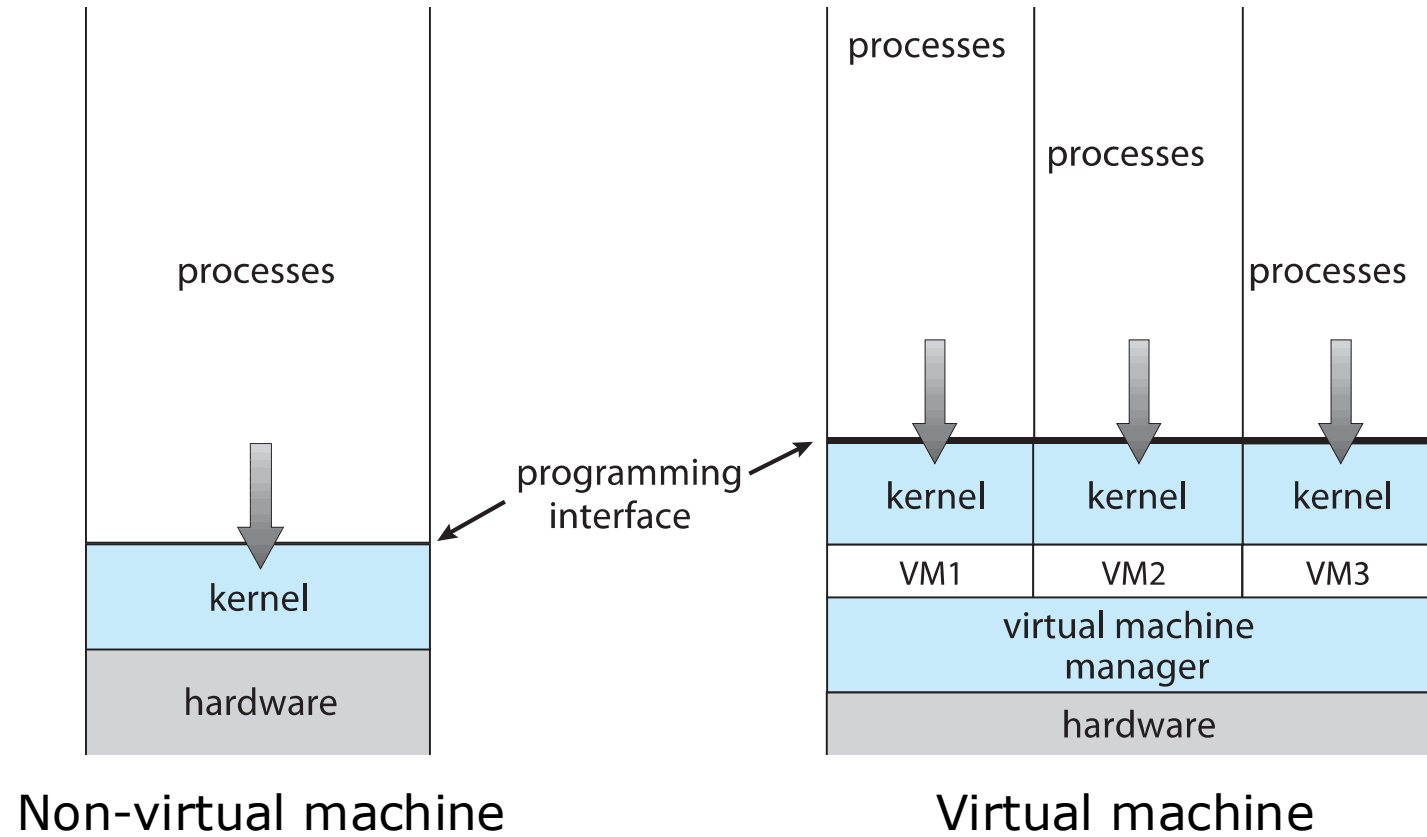
Computer Systems A Programmer's Perspective (3/E, Randal E. Bryan)

Computer Organization And Design (6/E, David A. Patterson)

subject 1.

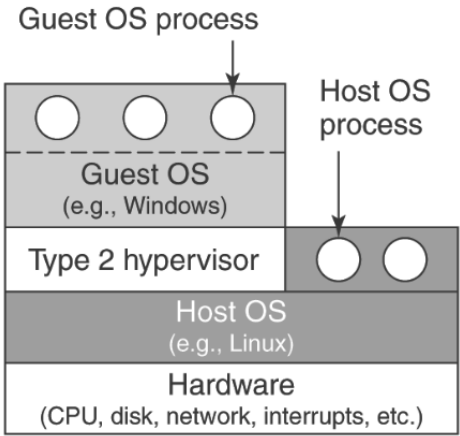
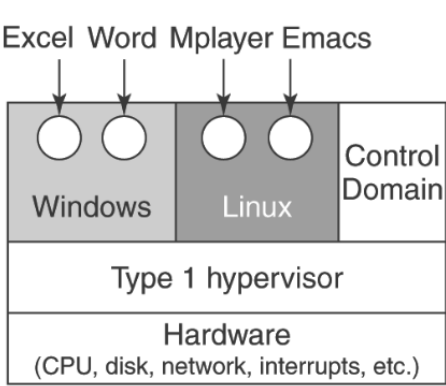
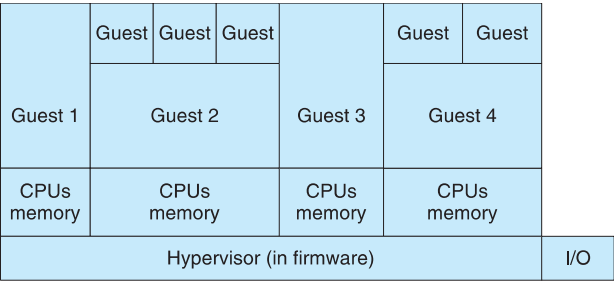
Virtual Machine

System Models



Virtual Machine Manager(VMM) | hypervisor

Type 0	Type 1	Type 2	ETC
펌웨어를 통해 가상 머신 생성 및 관리. 하드웨어 기반 솔루션	가상화를 제공하기 위해 구축된 운영체제와 유사한 소프트웨어	표준 운영체제에서 실행되지만 게스트 운영체제에 VMM 기능을 제공하는 응용 프로그램	programming environment - Java - Microsoft.Net Emulator
IBM LPAR Oracle LDOM	Vmware ESX MS Hyper-V KVM Proxmox	VMware Workstation Fusion Parallels VirtualBox	



Implementation

Generally difficult to provide an **exact** duplicate of underlying machine

Most VMMs implement **virtual CPU (VCPU)** to represent state of CPU per guest as guest believes it to be

- ▶ When guest context switched onto CPU by VMM, information from VCPU loaded and stored

CPU Scheduling

Memory Management

Storage Management

Privileged instructions

Page Tables

I/O

...

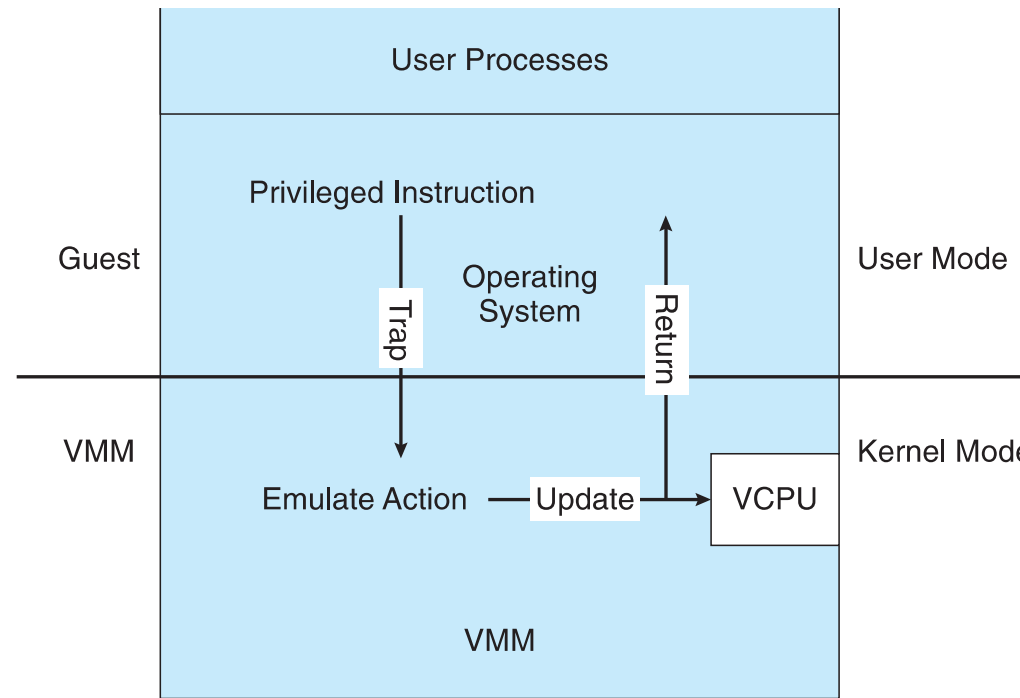
Trap-and-Emulate

Attempting a privileged instruction in user mode causes an error → trap

- virtual user mode → virtual kernel mode → real kernel mode

kernel mode privilege mode code runs slower due to trap-and-emulate

- multiple guests running, each needing trap-and-emulate



made by Seungtaek

Binary Translation

일부 CPU(ex. intel X86)는 특권 명령어와 비권한 명령을 완전히 분리하지 않음

- ex) Intel x86의 popf instruction : 특권상태면 모든 플래그 교체, 유저모드면 일부만 교체
- trap-and-emulate 불가능 (~1998년)

Binary Translation

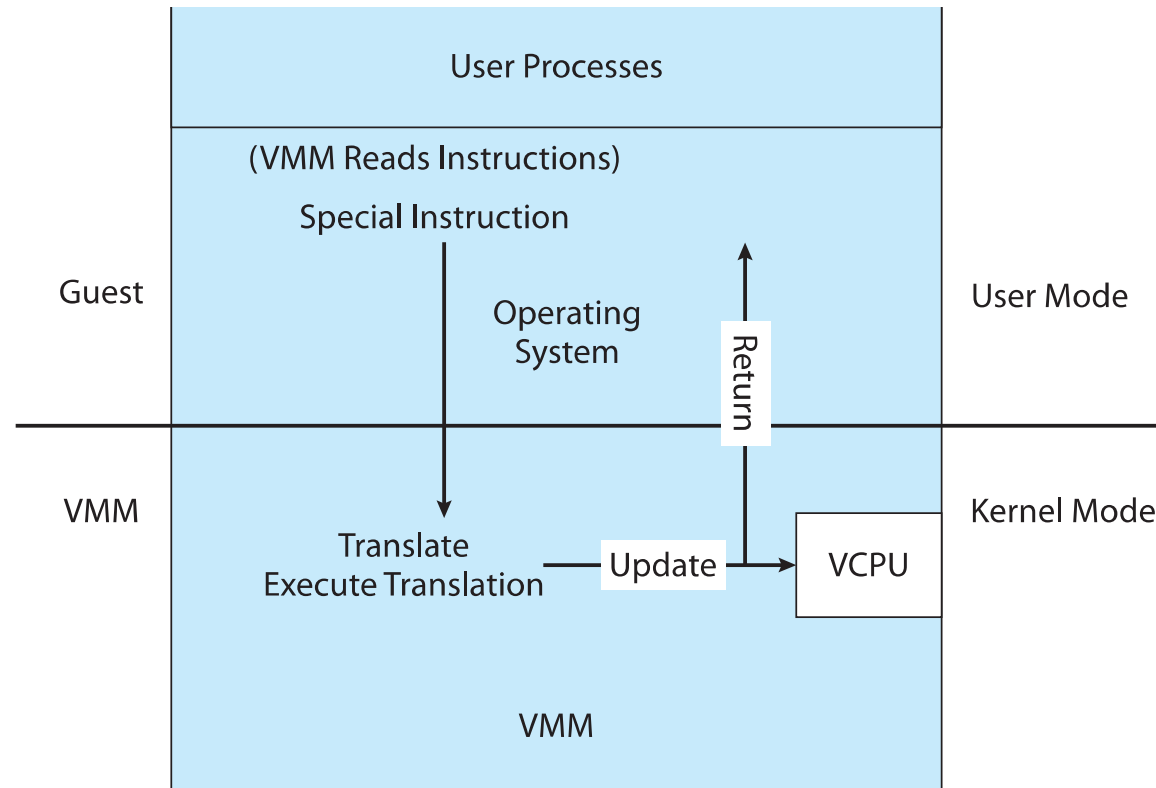
1. 게스트 vCPU가 사용자 모드인 경우 게스트는 물리적 CPU에서 명령어를 그대로 실행할 수 있다.
2. 게스트 vCPU가 커널모드인 경우 (= 게스트 입장에선 kernel mode라고 믿는 경우)
 1. VMM은 게스트의 Program Counter를 기반으로 가상 커널 모드에서 실행하는 모든 명령어를 검사
 2. 특수 명령어는 동등한 작업을 수행하는 새로운 명령어 집합으로 변환, 그 외는 그대로 실행

Binary Translation

성능 최적화 – Cache!

VMware: Windows XP를 부팅하고 바로 종료

- 950,000개의 변환 발생 (각 변환은 마이크로초 소요)
- 네이티브 실행에 비해 3초 증가(약 5%)

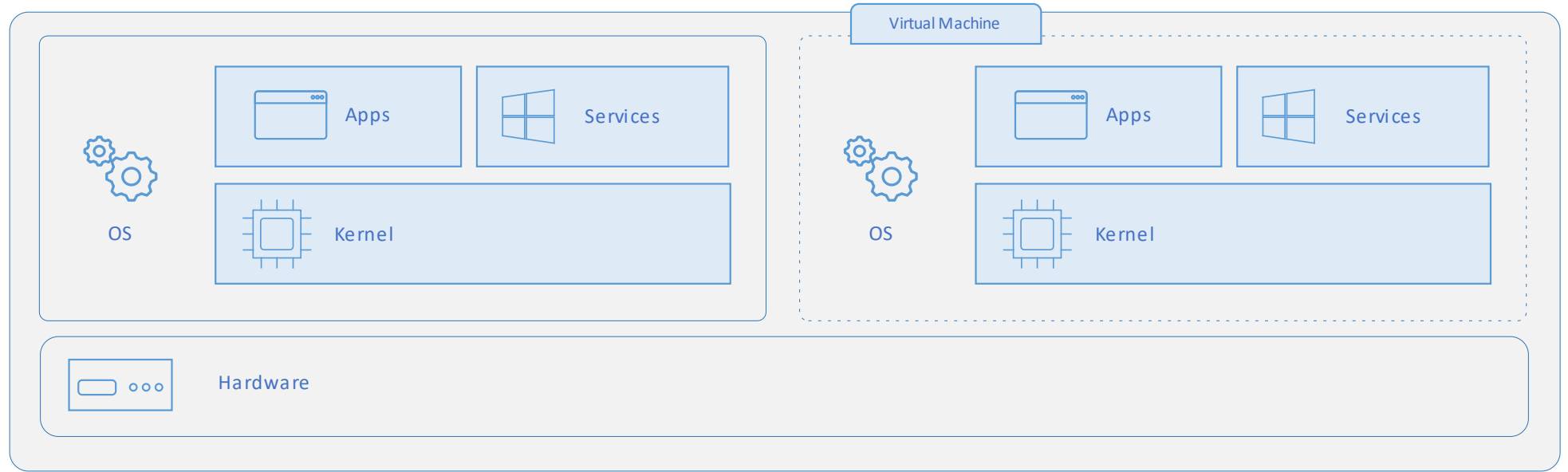


subject 2.

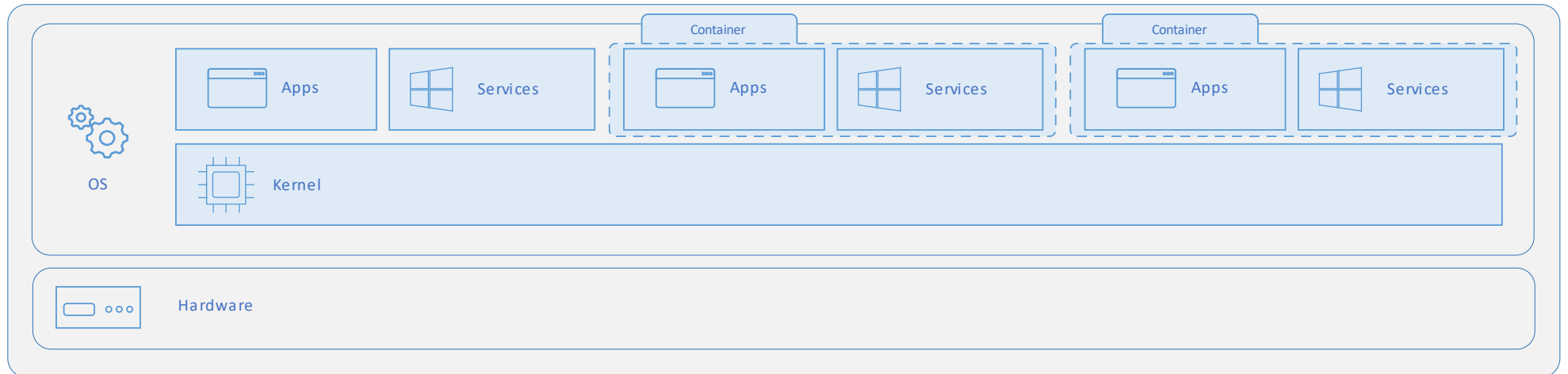
Container Basic

VM vs. Container

VM



Container



Container

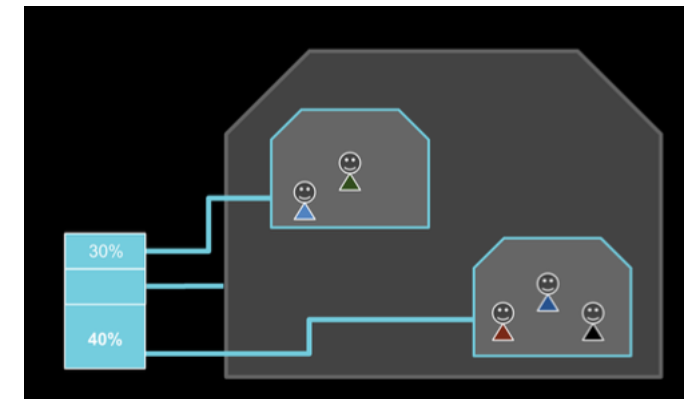
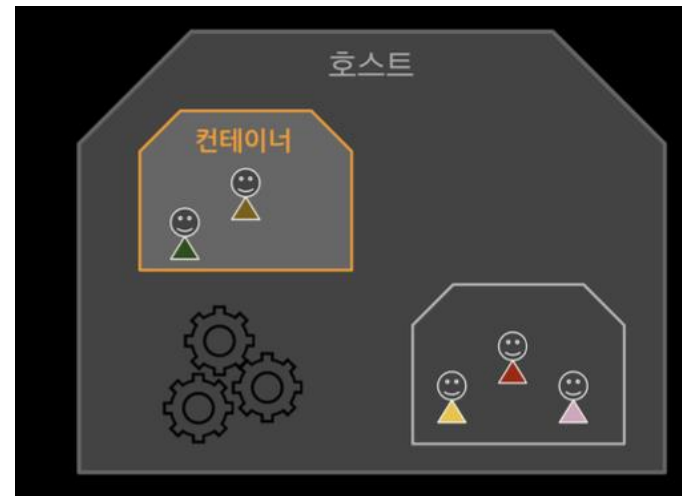
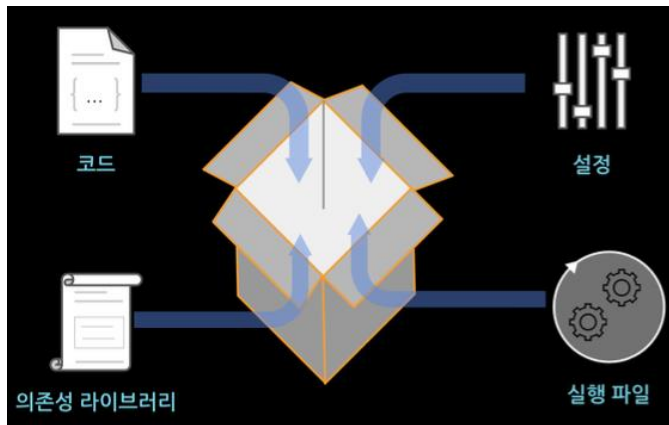
올인원 패키징

+

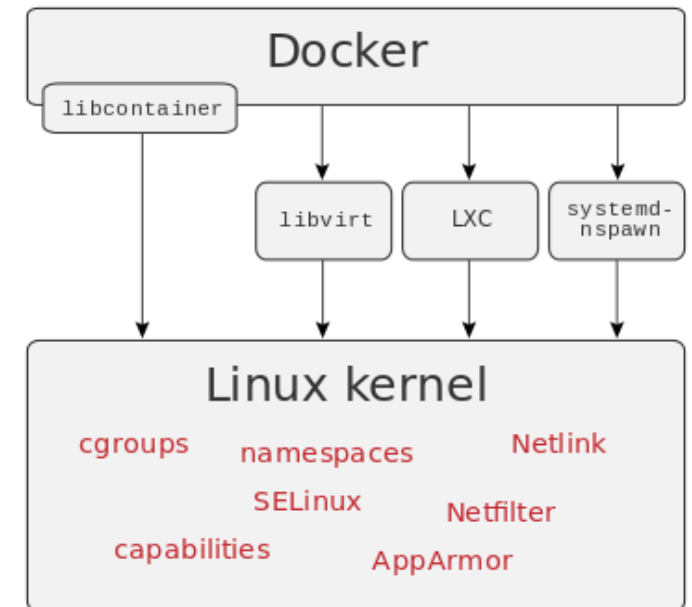
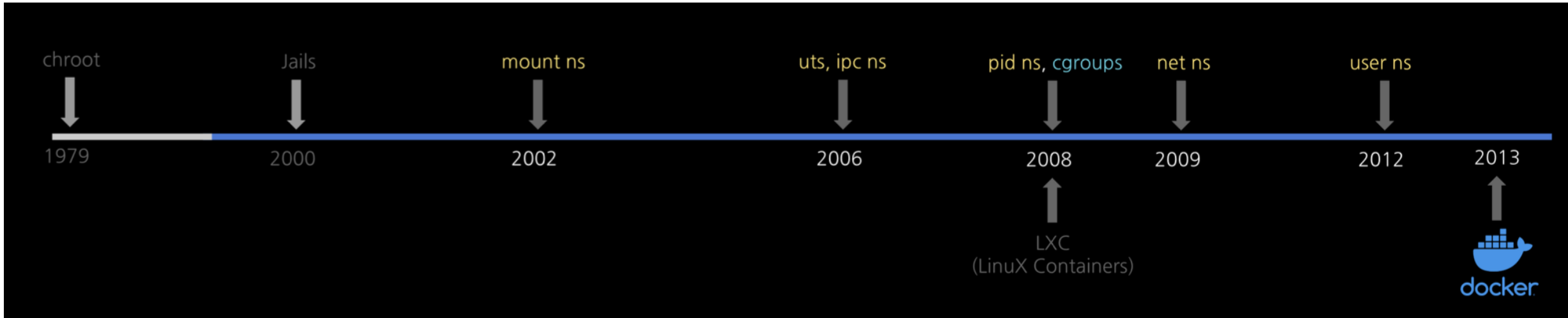
격리하기

+

자원 할당



Container history

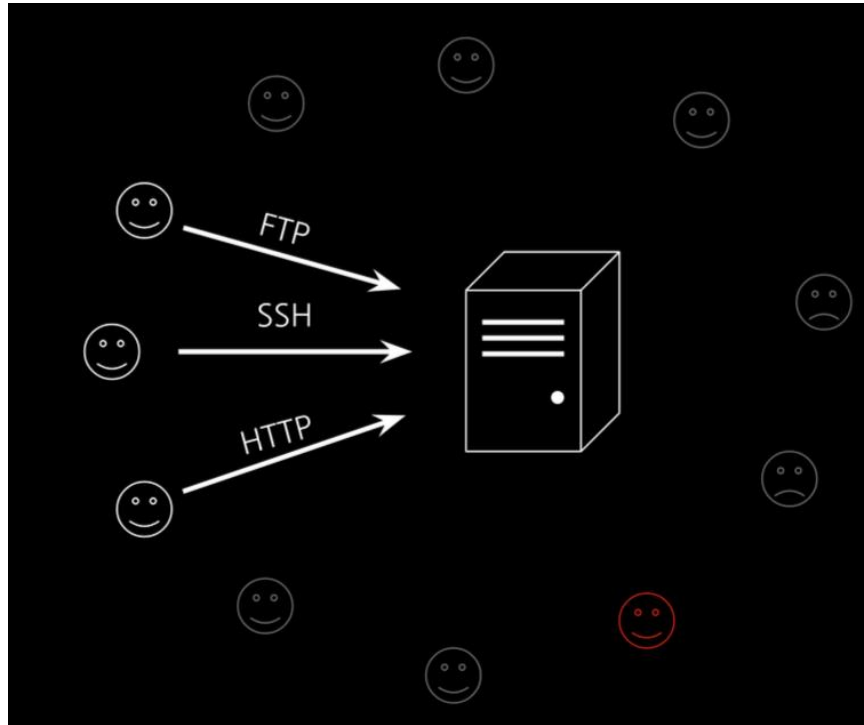


made by Seungtaek

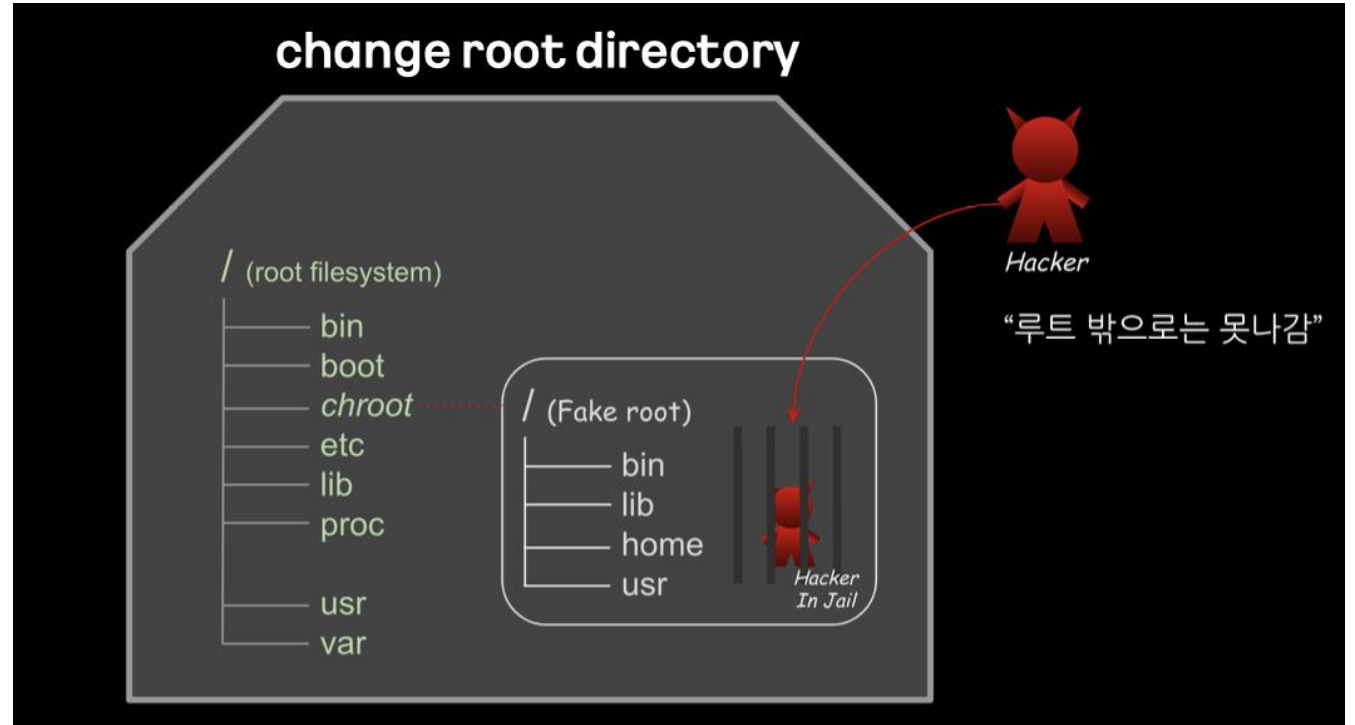
subject 3.

chroot

왜 필요한가



여러 사람들이 이용하는 서버



실습: /bin/sh 복사하기

```
tester1@seungtaek:~$ which sh
/bin/sh
```

```
tester1@seungtaek:~$ ldd /bin/sh
linux-vdso.so.1 (0x00007ffc7d3d0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2f89363000)
/lib64/ld-linux-x86-64.so.2 (0x00007f2f89974000)
```

} **shared library dependencies**

```
tester1@seungtaek:~$ mkdir -p myroot/bin;
tester1@seungtaek:~$ cp /bin/sh myroot/bin/;
tester1@seungtaek:~$ mkdir -p myroot/{lib64,lib/x86_64-linux-gnu};
tester1@seungtaek:~$ cp /lib/x86_64-linux-gnu/libc.so.6 myroot/lib/x86_64-linux-gnu/;
tester1@seungtaek:~$ cp /lib64/ld-linux-x86-64.so.2 myroot/lib64/;
```

```
tester1@seungtaek:~$ tree myroot
myroot
├── bin
│   └── sh
├── lib
│   ├── x86_64-linux-gnu
│   │   └── libc.so.6
└── lib64
    └── ld-linux-x86-64.so.2
```

```
tester1@seungtaek:~$ sudo chroot myroot /bin/sh
# ls
/bin/sh: 1: ls: not found
# exit
tester1@seungtaek:~$
```


실습: /bin/lis 복사하기

```
tester1@seungtaek:~$ which ls
/bin/lis
```

```
tester1@seungtaek:~$ ldd /bin/lis
linux-vdso.so.1 (0x00007ffd025b4000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007fa81cff3000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fa81cc02000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fa81c991000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fa81c78d000)
/lib64/ld-linux-x86-64.so.2 (0x00007fa81d43d000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fa81c56e000)
```

```
tester1@seungtaek:~$ cp /bin/lis myroot/bin/;
```

```
tester1@seungtaek:~$ cp /lib/x86_64-linux-gnu/{libselinux.so.1,libc.so.6,libpcre.so.3,libdl.so.2,libpthread.so.0} \
myroot/lib/x86_64-linux-gnu/;
```

```
tester1@seungtaek:~$ cp /lib64/ld-linux-x86-64.so.2 myroot/lib64/;
```

```
tester1@seungtaek:~$ tree myroot
myroot
├── bin
│   ├── ls
│   └── sh
├── lib
│   └── x86_64-linux-gnu
│       ├── libc.so.6
│       ├── libdl.so.2
│       ├── libpcre.so.3
│       ├── libpthread.so.0
│       └── libselinux.so.1
└── lib64
    └── ld-linux-x86-64.so.2
```

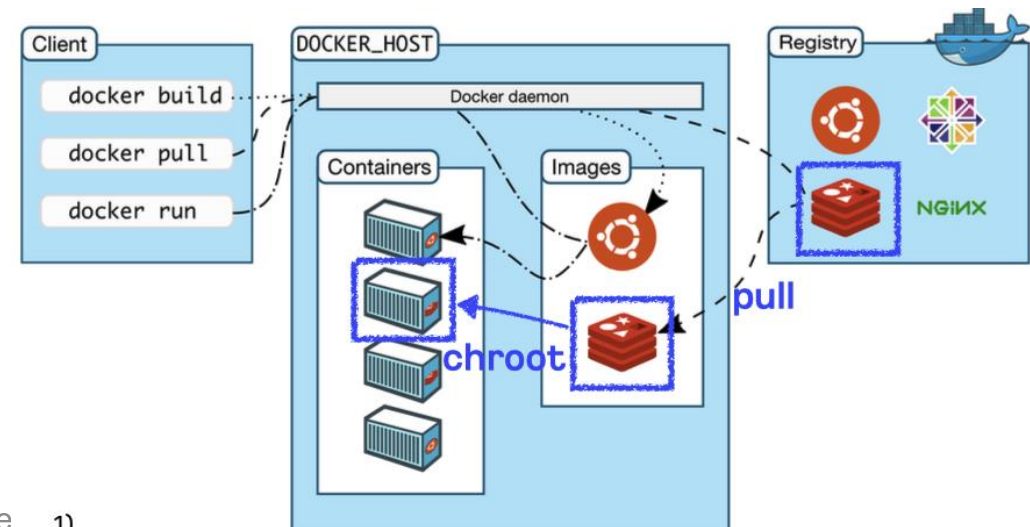
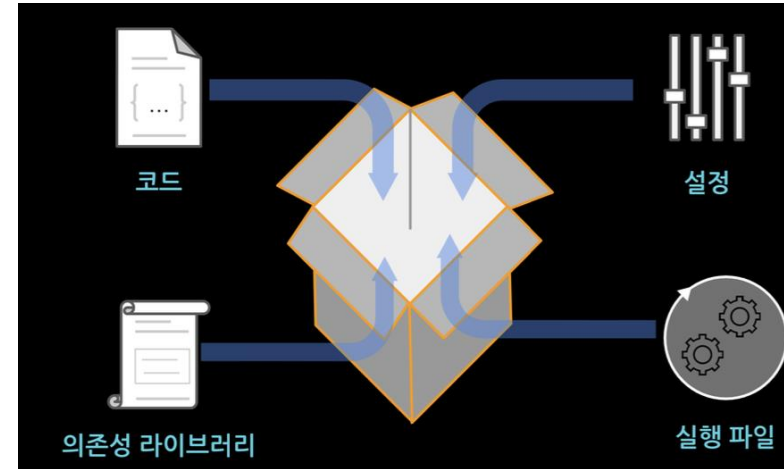
실습: chroot

```
tester1@seungtaek:~$ sudo chroot myroot /bin/sh
# ls
bin lib lib64

# cd ../../../../
# ls
bin lib lib64

# exit
tester1@seungtaek:~$
```

1. **패키징** : myroot 폴더에 우리가 사용할 명령어 파일들을 모두 담기 → **이미지**
2. **격리** : 경로에 가둬서 실행



탈옥하기

tester1@seungtaek:~\$ vi escape_chroot.c

```
#include <sys/stat.h>
#include <unistd.h>
int main(void) {
    mkdir(".out", 0755);
    chroot(".out");
    chdir("../../../../../");
    chroot(".");
    return execl("/bin/sh", "-i", NULL);
}
```

tester1@seungtaek:~\$ gcc -o myroot/escape_chroot escape_chroot.c

tester1@seungtaek:~\$ sudo chroot myroot /bin/sh

ls

bin escape_chroot lib lib64

./escape_chroot

ls

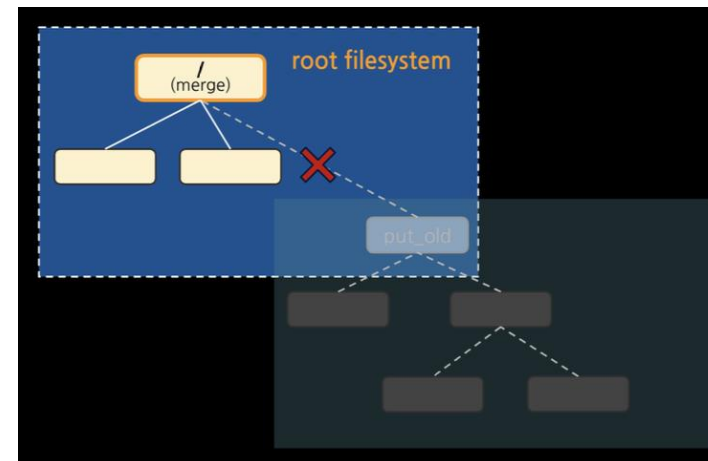
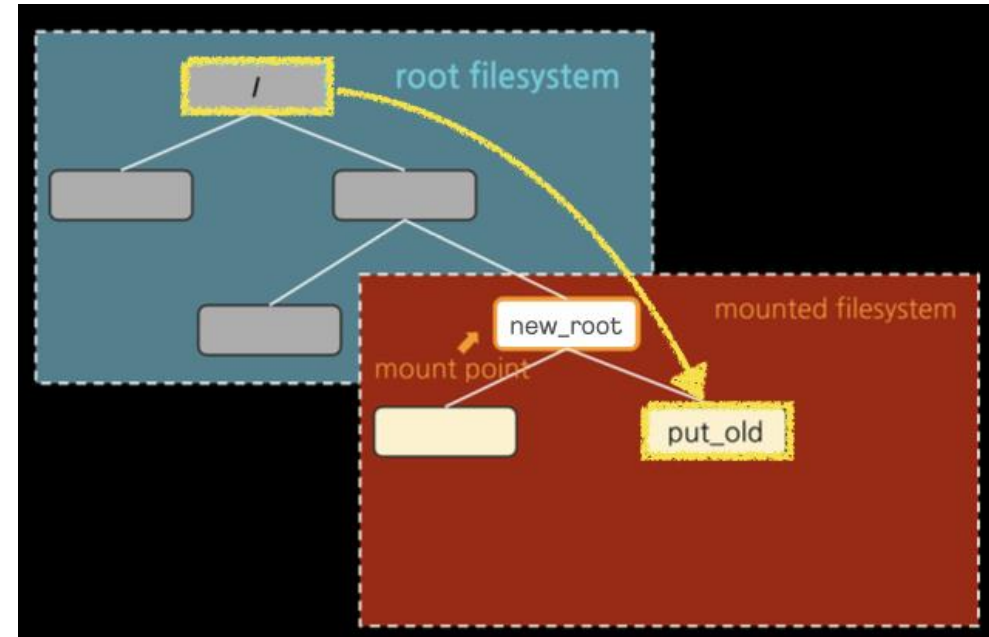
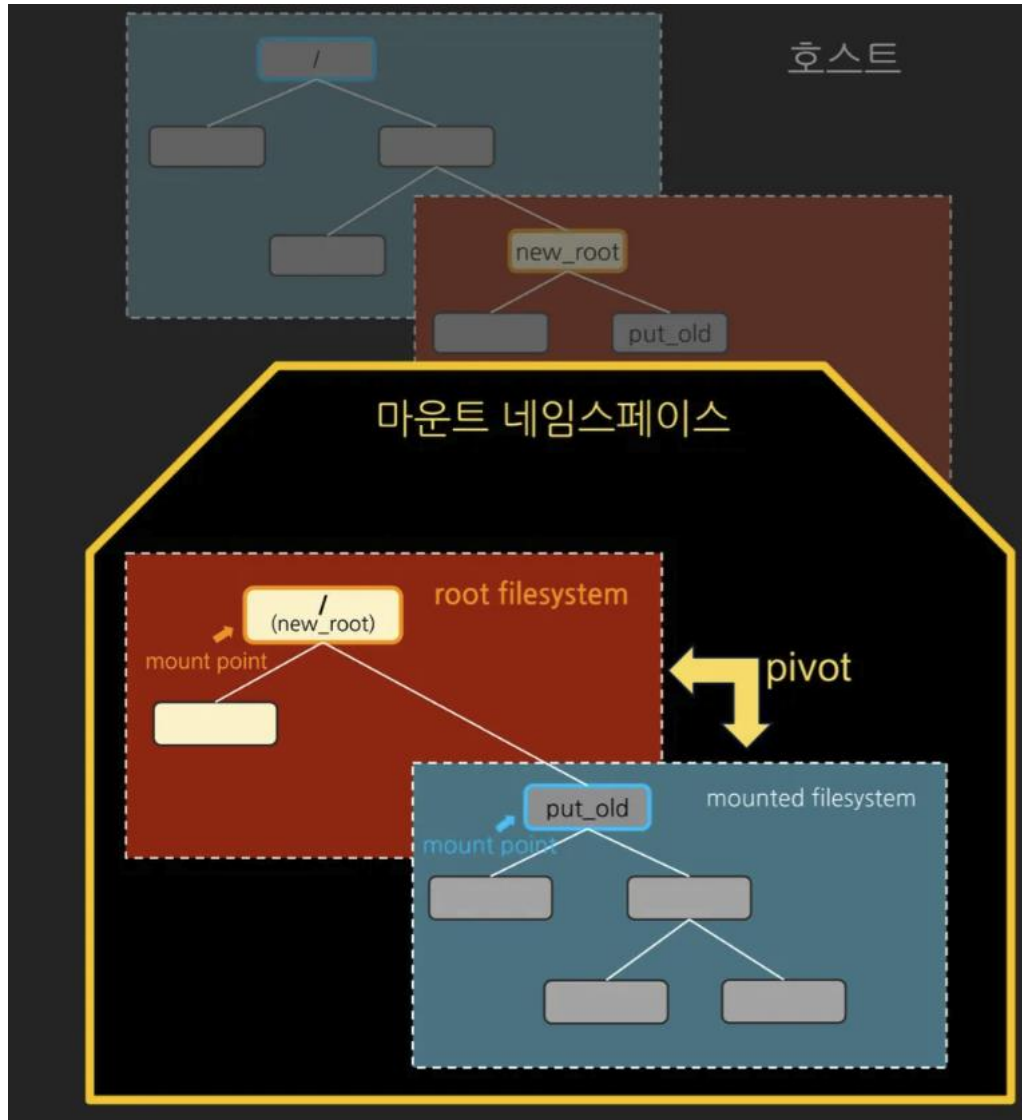
bin dev initrd.img lib64 mnt root snap sys var
boot etc initrd.img.old lost+found opt run srv tmp vmlinuz
cdrom home lib media proc sbin swap.img usr vmlinuz.old

exit

exit

tester1@seungtaek:~\$

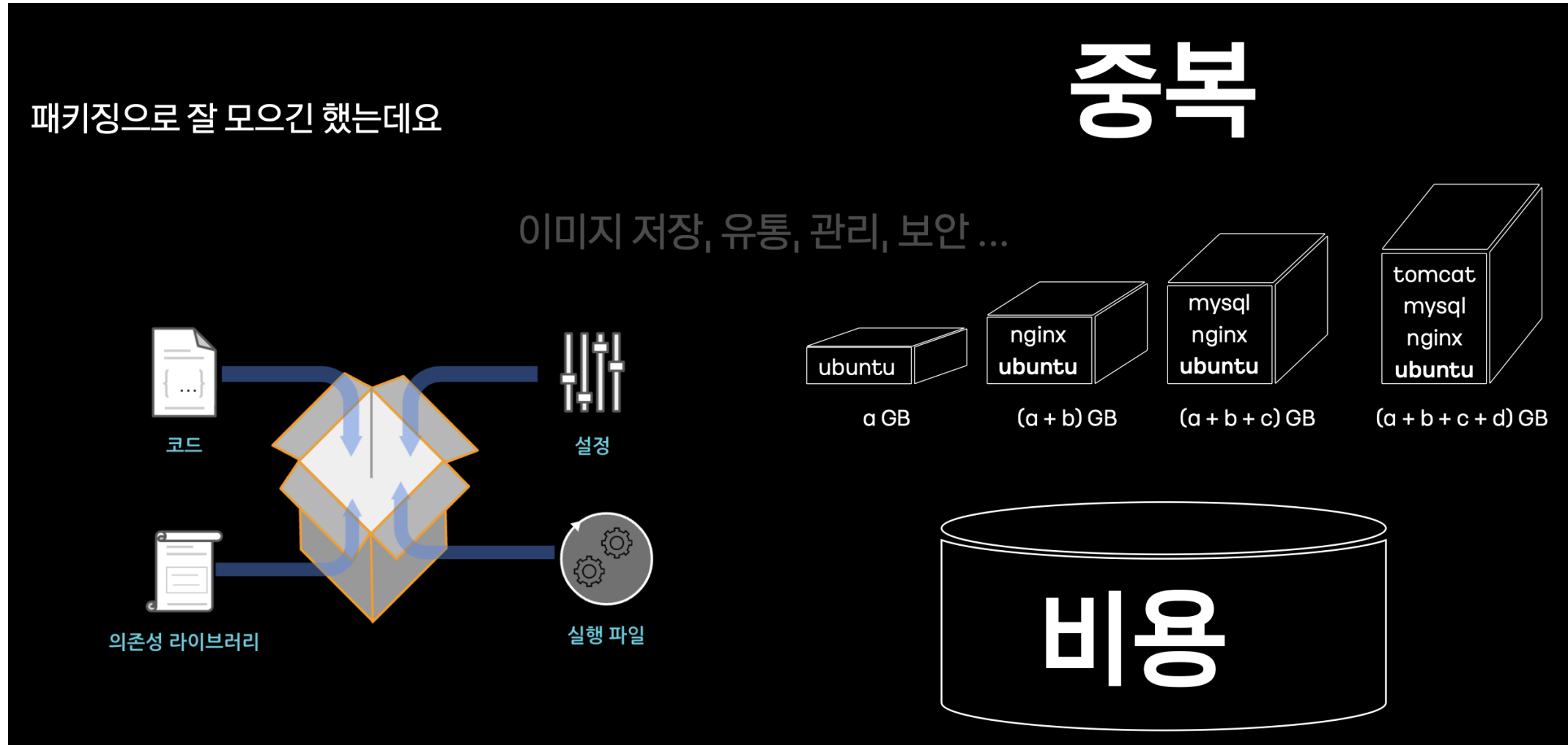
mount namespace + pivot_root



subject 4.

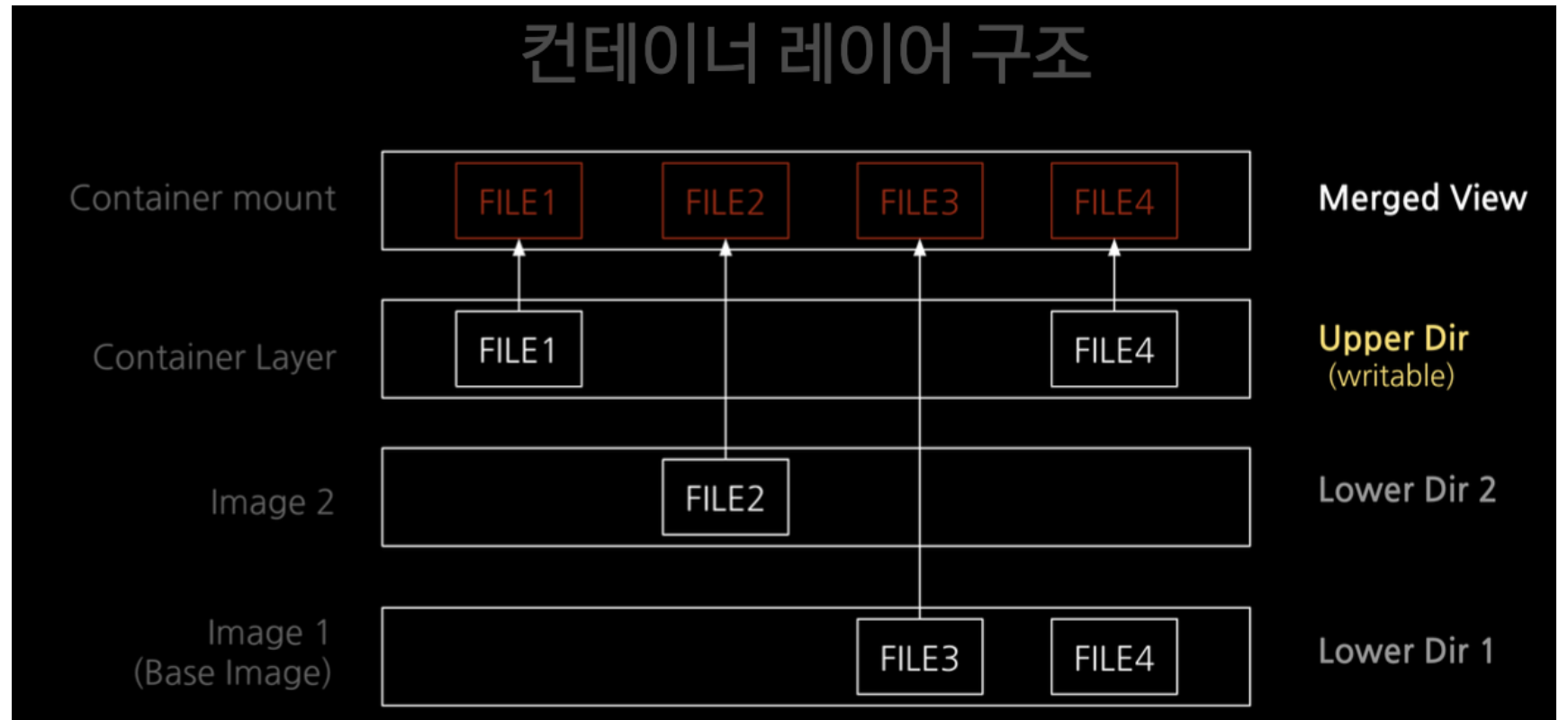
Overlay filesystem

패키징 중복문제 발생

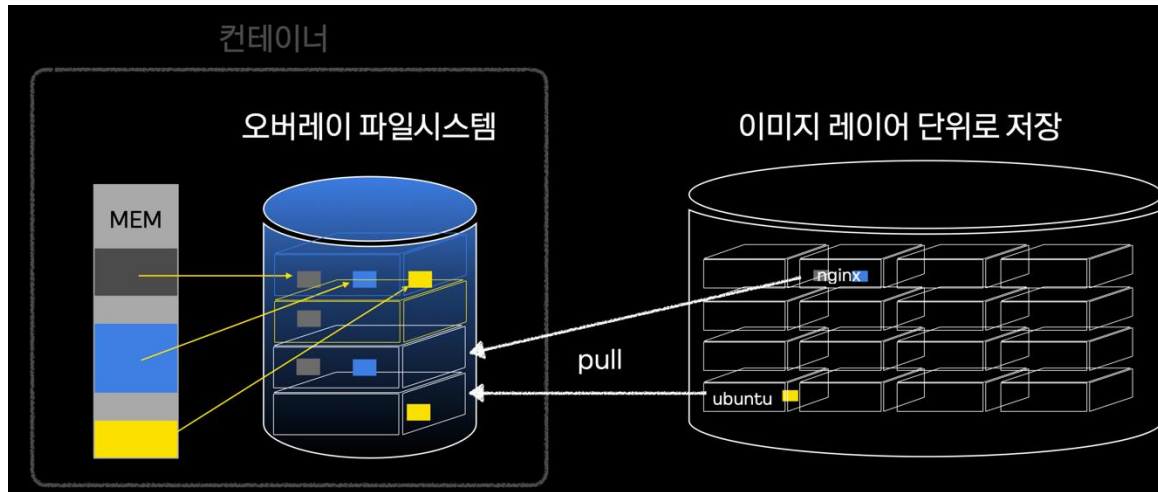


overlay filesystem

- 여러 이미지 레이어를 하나로 마운트
- Lower 레이어는 ReadOnly, Upper 레이어는 Writable
- CoW(copy-on-write) : (원본유지)



overlay filesystem



```
cherry@ubuntu:~$  
cherry@ubuntu:~$ docker pull redis  
Using default tag: latest  
latest: Pulling from library/redis  
faef57eae888: Already exists  
bb595d48e52d: Pull complete  
d479b54c3bb2: Pull complete  
2044989c541a: Pull complete  
01e4ba5495fa: Pull complete  
ed7a9fd4b0ea: Pull complete  
Digest: sha256:08a82d4bf8a8b4dd94e8f5408cbbad9dd184c1cf311d34176cd3e9972c43f872  
Status: Downloaded newer image for redis:latest  
docker.io/library/redis:latest  
cherry@ubuntu:~$  
cherry@ubuntu:~$
```


subject 5.

namespace

네임스페이스 명령어

unshare [옵션] [프로그램 [arguments ...]]

(옵션)

-m, --mount

-u, --uts

-i, --ipc

-p, --pid

-n, --net

-U, --user

여러 종류의 네임스페이스

MOUNT(2002) : 마운트 네임스페이스

- unshare -m
- 마운트 포인트 격리

UTS(2006) : Unix Time Sharing

- unshare -u
- 호스트명, 도메인명 격리

IPC(2006) : Inter-Process Communication (Shared Memory, Pipe, Message Queue...)

- unshare -i
- IPC 네임스페이스를 공유하는 process끼리만 통신할 수 있다.

PID(2008) : PID 넘버스페이스 격리

Cgroups(2008) : 프로세스 자원 격리

NET(2009) : 네트워크 네임스페이스

USER(2012) : uid/gid 넘버스페이스 격리

도커(2013)

PID : PID 넘버스페이스 격리

tester1@seungtaek:~\$ ps -ef

UID	PID	PPID	C	S	TIME	TTY	TIME	CMD
root	1	0	0	13:34	?	00:00:10		/sbin/init maybe-ubiquity
root	2	0	0	13:34	?	00:00:00		[kthreadd]
root	4	2	0	13:34	?	00:00:00		[kworker/0:0H]
root	6	2	0	13:34	?	00:00:00		[mm_percpu_wq]
root	7	2	0	13:34	?	00:00:00		[ksoftirqd/0]

...

tester1@seungtaek:~\$ sudo unshare -fp --mount-proc /bin/sh

ps -ef

UID	PID	PPID	C	S	TIME	TTY	TIME	CMD
root	1	0	0	16:58	pts/2	00:00:00		/bin/sh
root	2	1	0	16:59	pts/2	00:00:00		ps -ef

USER : uid, gid 넘버스페이스 격리

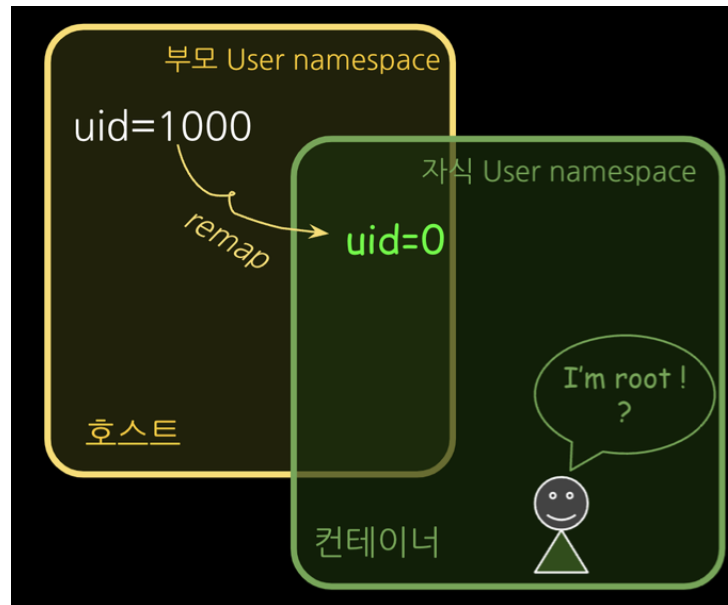
```
tester1@seungtaek:~$ id
```

```
uid=1000(tester1) gid=1000(tester1) groups=1000(tester1),  
4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lxd)...
```

```
tester1@seungtaek:~$ unshare -U --map-root-user /bin/sh
```

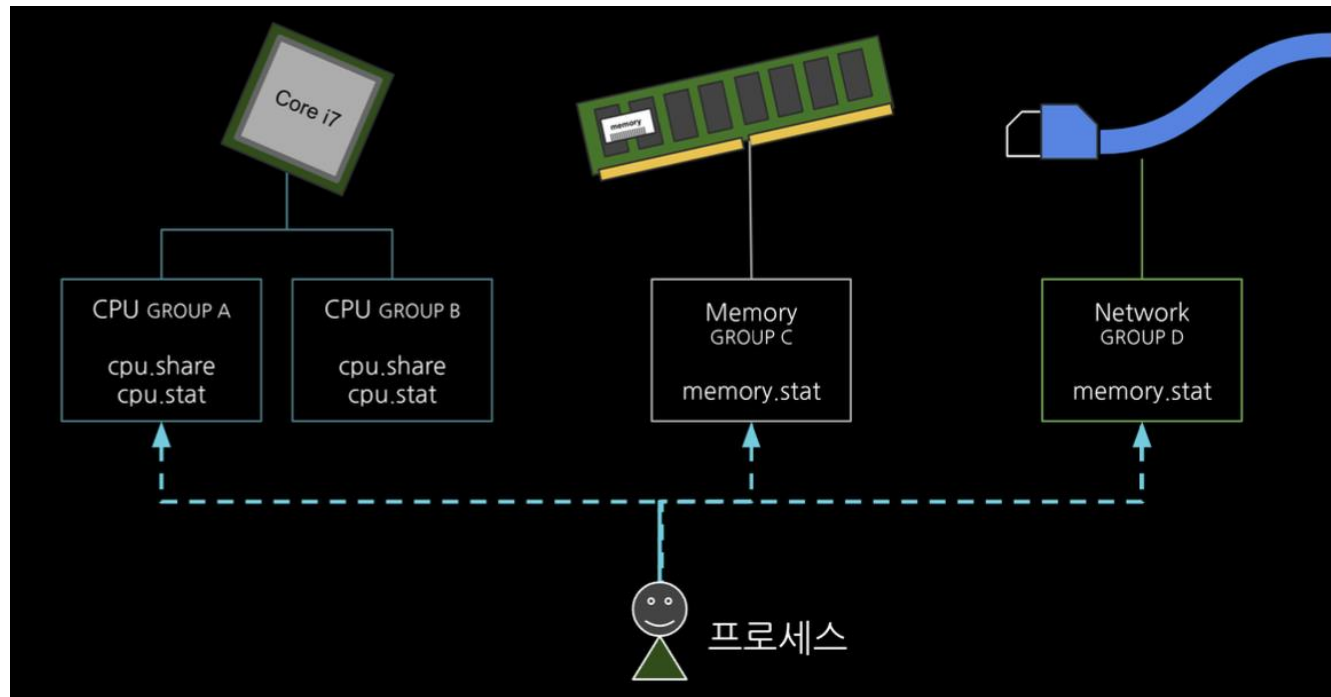
```
# id
```

```
uid=0(root) gid=0(root) groups=0(root),65534(nogroup)
```



Cgroups (Control Groups)

- Cgroups(Control Groups)
 - 2008년 구글에서 만들어서 리눅스에 contribution
 - 컨테이너 별로 자원을 분배하고 limit 내에서 운용
- 하나 또는 복수의 장치를 묶어서 그룹
- 프로세스가 사용하는 리소스 통제



Cgroups (Control Groups)

```
tester1@seungtaek:~$ stress -c 1
```

```
stress: info: [8315] dispatching hogs: 1 cpu, 0 io, 0 vm, 0 hdd  
^C
```

```
root@seungtaek:~# cgcreate -a root -g cpu:mycgroup
```

```
root@seungtaek:~# cgset -r cpu.cfs_quota_us=30000 mycgroup;
```

```
tester1@seungtaek:~$ sudo cgexec -g cpu:mycgroup stress -c 1
```

top 명령어로 점유율 확인

subject 6.

JVM warm-up

Compiled Language

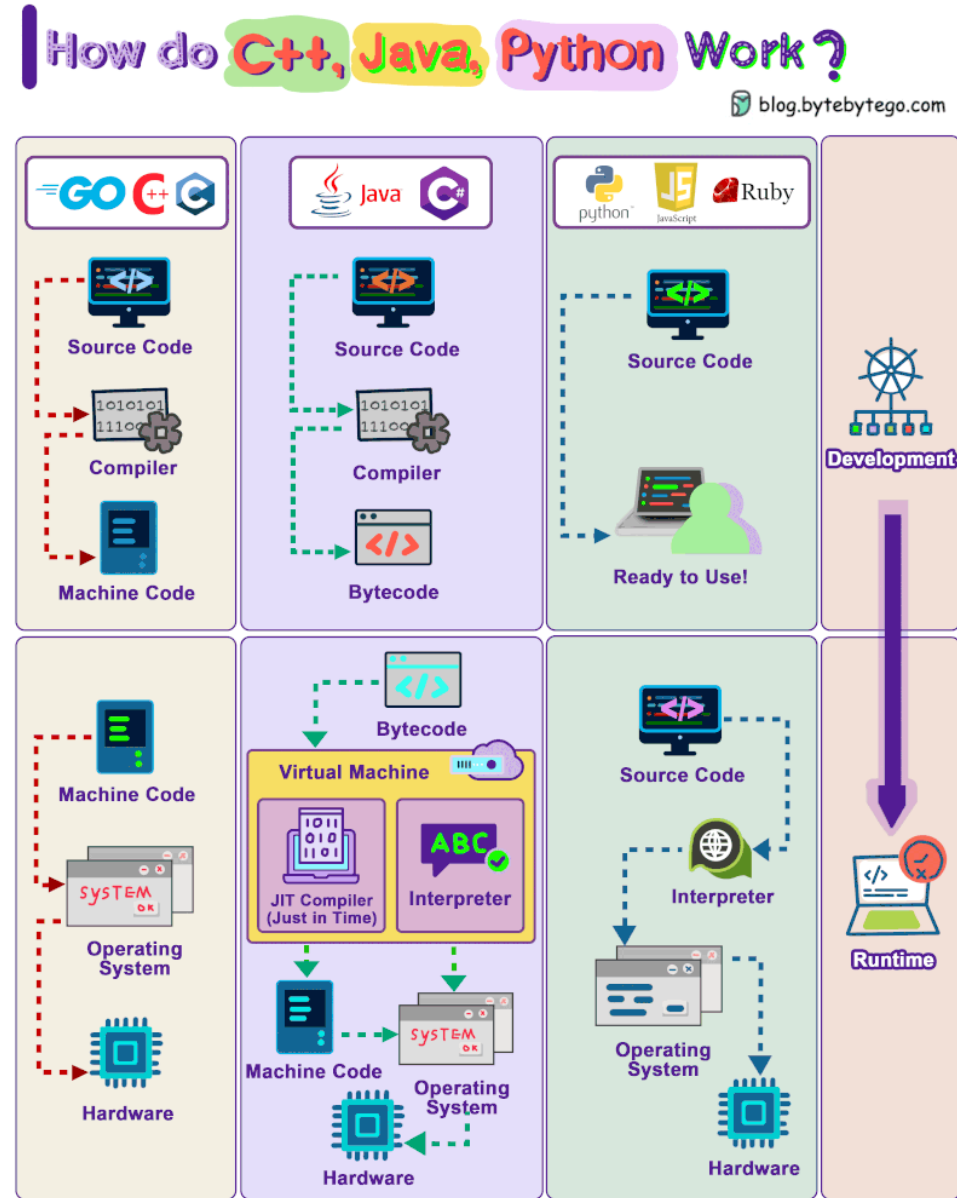
C, C++, Go, Rust, ...



장점: 성능

단점: CPU 아키텍처에 종속적

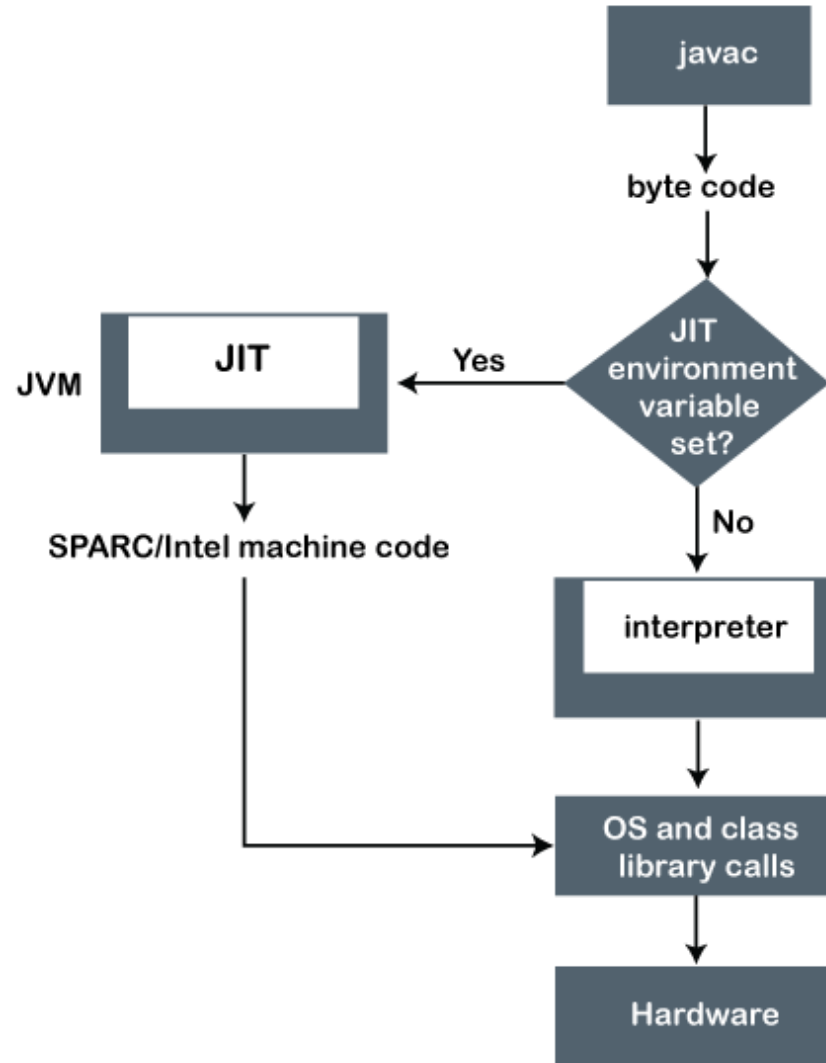
How do language work?



Java JIT(Just In Time) Compiler

오라클에선 Hotspot이라 부름

Cache!
+
**런타임 환경에 맞춰
최적화!**



JIT Compilation Process

JIT Internals

- 메서드 단위로 컴파일
- 네이티브 코드로 전환 후속 최적화 작업을 위해 프로파일리 정보를 수집
- Tiered compilation : 단계별 컴파일
 - C1 : optimization, C2 : fully optimization
 - C1, C2는 별도의 스레드로 동작한다.
- Compilation Level
 - level 0: interpreted code
 - level 1: simple C1 compiled code
 - level 2: limited C1 compiled code
 - level 3: full C1 compiled code
 - level 4: C2 compiled code

Compilation Level Threshold

```
java -XX:+PrintFlagsFinal -version|grep Threshold|grep Tier
```

```
▶ java -XX:+PrintFlagsFinal -version|grep Threshold|grep Tier
openjdk version "21.0.2" 2024-01-16 LTS
OpenJDK Runtime Environment Zulu21.32+17-CA (build 21.0.2+13-LTS)
OpenJDK 64-Bit Server VM Zulu21.32+17-CA (build 21.0.2+13-LTS, mi
  uintx IncreaseFirstTierCompileThresholdAt      = 50
  intx Tier2BackEdgeThreshold                    = 0
  intx Tier2CompileThreshold                    = 0
  intx Tier3BackEdgeThreshold                    = 60000
  intx Tier3CompileThreshold                    = 2000
  intx Tier3InvocationThreshold                  = 200
  intx Tier3MinInvocationThreshold                = 100
  intx Tier4BackEdgeThreshold                    = 40000
  intx Tier4CompileThreshold                    = 15000
  intx Tier4InvocationThreshold                  = 5000
  intx Tier4MinInvocationThreshold                = 600
```

InvocationThreshold : 메서드 호출 수

BackEdgeThreshold : 하나의 메서드의 반복문 횟수

CompileThreshold : 메서드 호출 수+반복문 횟수

JIT Compiler

Practice

https://github.com/CS-Computer-Science-Study/Operating-System/blob/main/practice/Jvm_warmup/RSAWarmup.java

```
public static void main(String[] args) {  
    RSAWarmup rsa = new RSAWarmup();  
  
    printResult(rsa);  
    for (int i = 1; i < 200; i++) {  
        rsa.method();  
    }  
    printResult(rsa);  
    for (int i = 1; i < 200; i++) {  
        rsa.method();  
    }  
    printResult(rsa);  
}
```

```
285115000  
63172209  
49745500
```

Outro

Why?



아빠, 밖에 나가서 놀아도 돼?

왜?

왜?

왜?

왜?

왜?

왜?



안돼

지금 새벽 5시야, 너무 일러

해가 아직 안 떴으니까

왜냐면 해는 조금 늦게 뜨거든

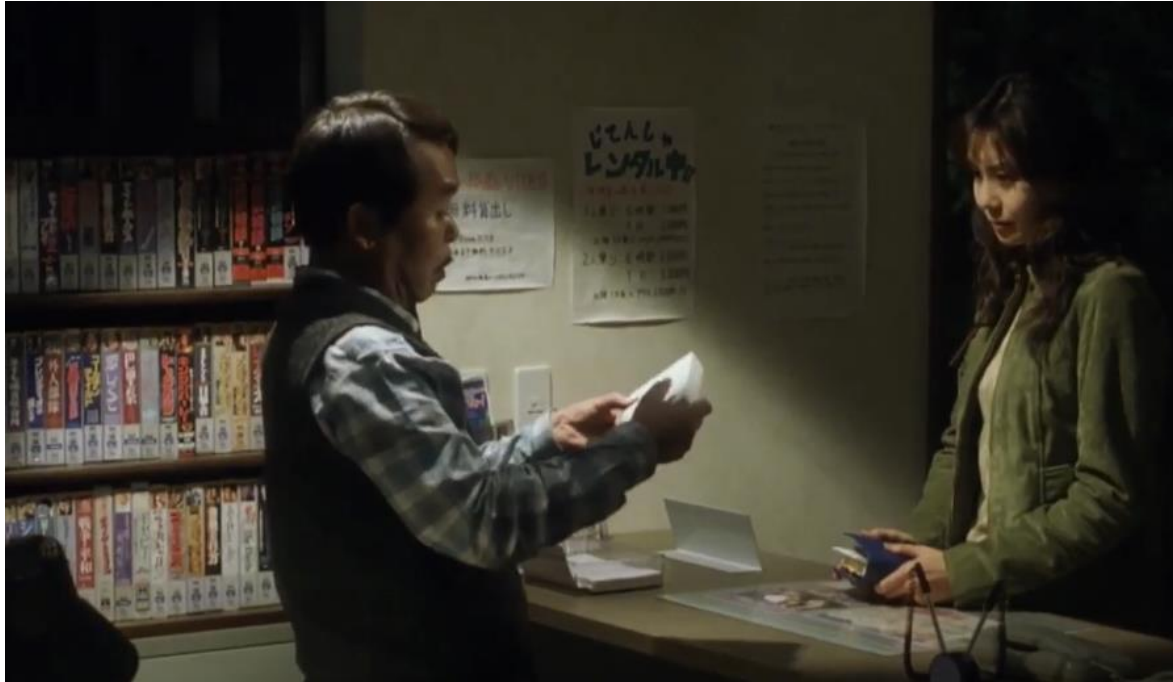
지구가 자전을 해서 얼마간 돌면 지평선에 해가 뜨는거야

아빠도 몰라

아빠가 학교에서 제대로 공부를 안해서 그래

호기심에는 대가가 따른다.

링(1998)



추상화는 그만한 이유가 있다

```
f = open('file', 'w'); f.write('content');
```

추상화는 그만한 이유가 있다

```
f = open('file', 'w'); f.write('content');
```

system call, kernel mode

추상화는 그만한 이유가 있다

```
f = open('file', 'w'); f.write('content');
```

system call, kernel mode

디스크 컨트롤러, 인터럽트, IVT, ISR

추상화는 그만한 이유가 있다

```
f = open('file', 'w'); f.write('content');
```

system call, kernel mode

디스크 컨트롤러, 인터럽트, IVT, ISR

**Memory-Mapped I/O, File
Top-Bottom half, DMA, BUS, PCI**

추상화는 그만한 이유가 있다

```
f = open('file', 'w'); f.write('content');
```

system call, kernel mode

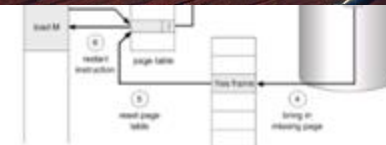
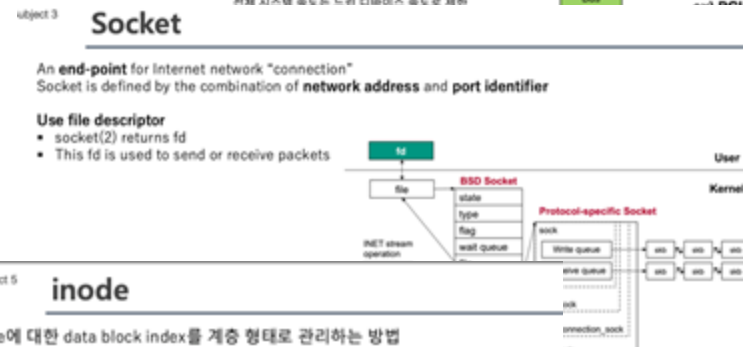
디스크 컨트롤러, 인터럽트, IVT, ISR

Memory-Mapped I/O, File
Top-Bottom half, DMA, BUS, PCI

로컬 디스크가 아닐 수도 있습니다?



part 4 Thread in Linux



Compilation Process

