- 기능 명세 -

Boot Loader

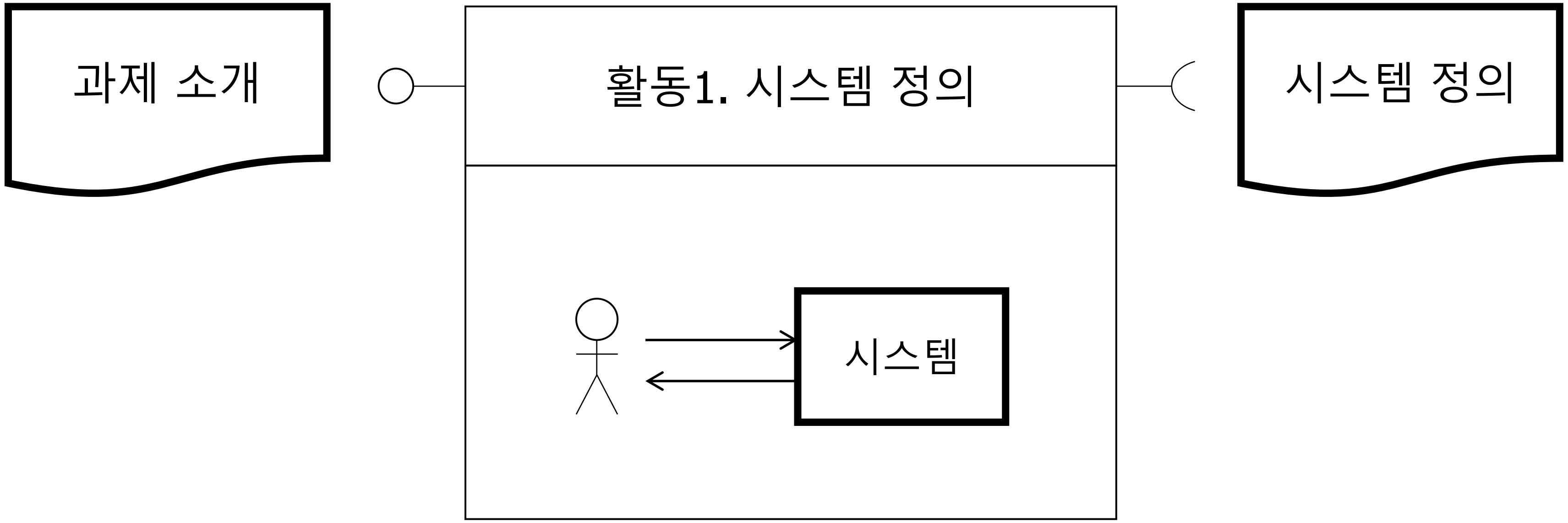
조용진

(drajin.cho@bosornd.com)

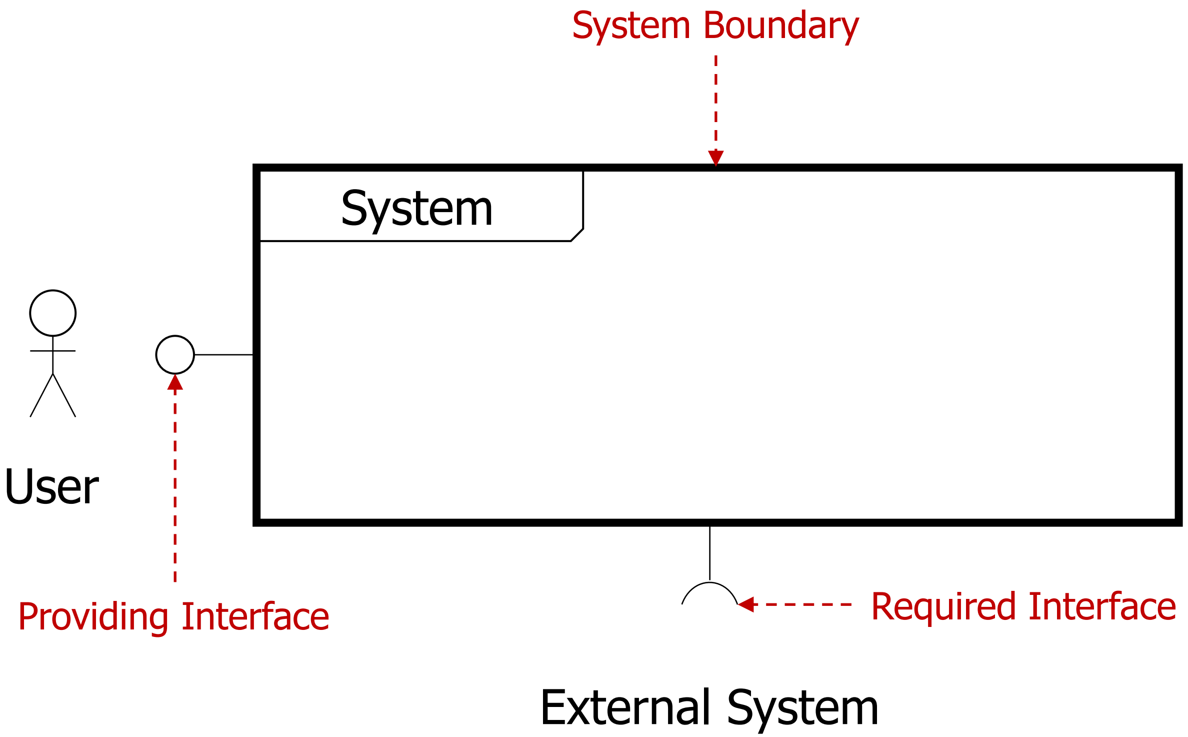
주식회사 보이는 소프트웨어 연구소
http://www.bosornd.com

활동1. 시스템 정의

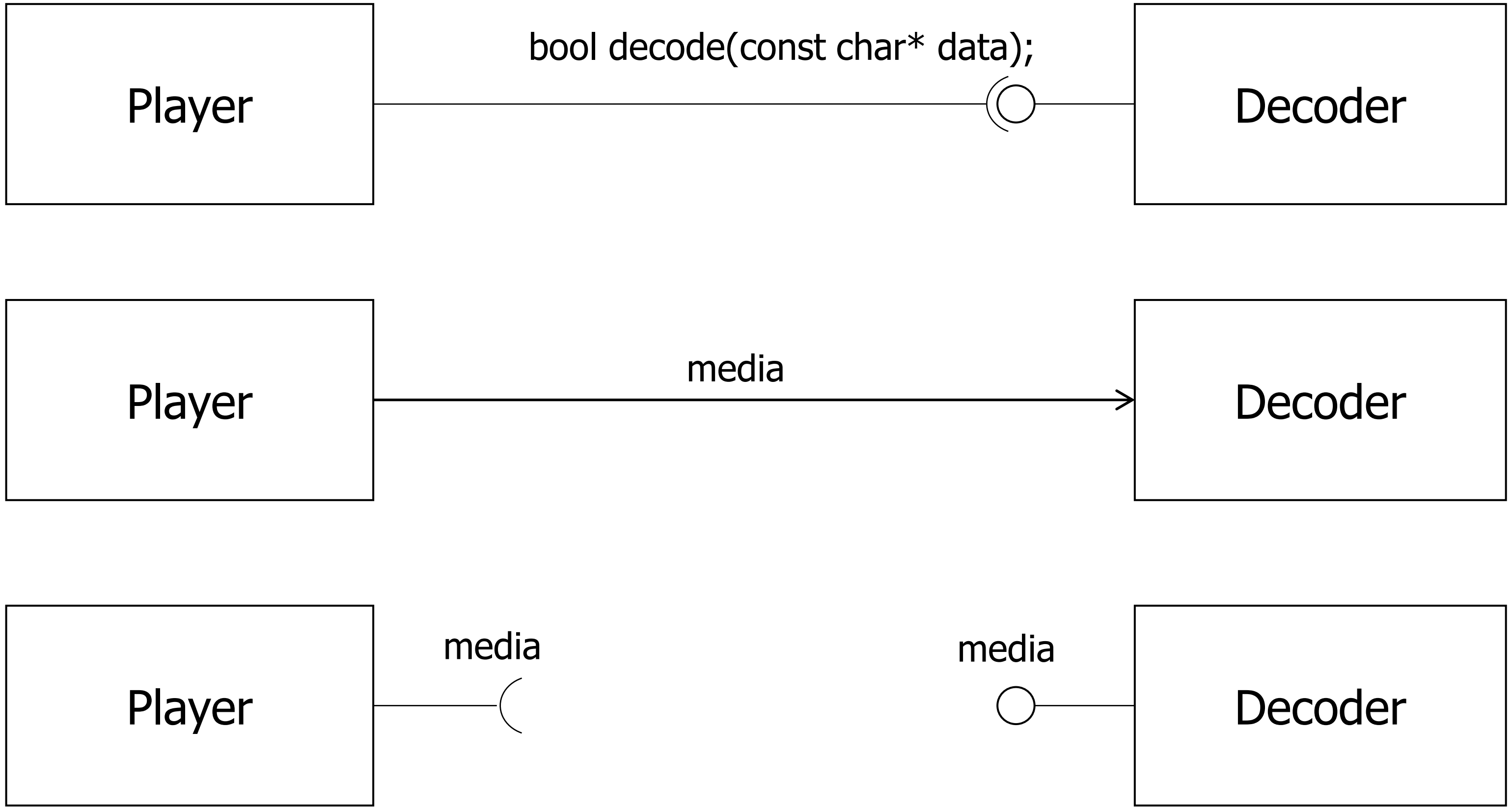
목적	개발하고자 하는 시스템의 경계를 정의한다. 시스템과 상호 작용하는 외부 시스템(이해관계자 포함)을 정의한다. 시스템과 외부 시스템과의 관계를 파악한다. 시스템의 동작 및 사업 환경을 파악한다.



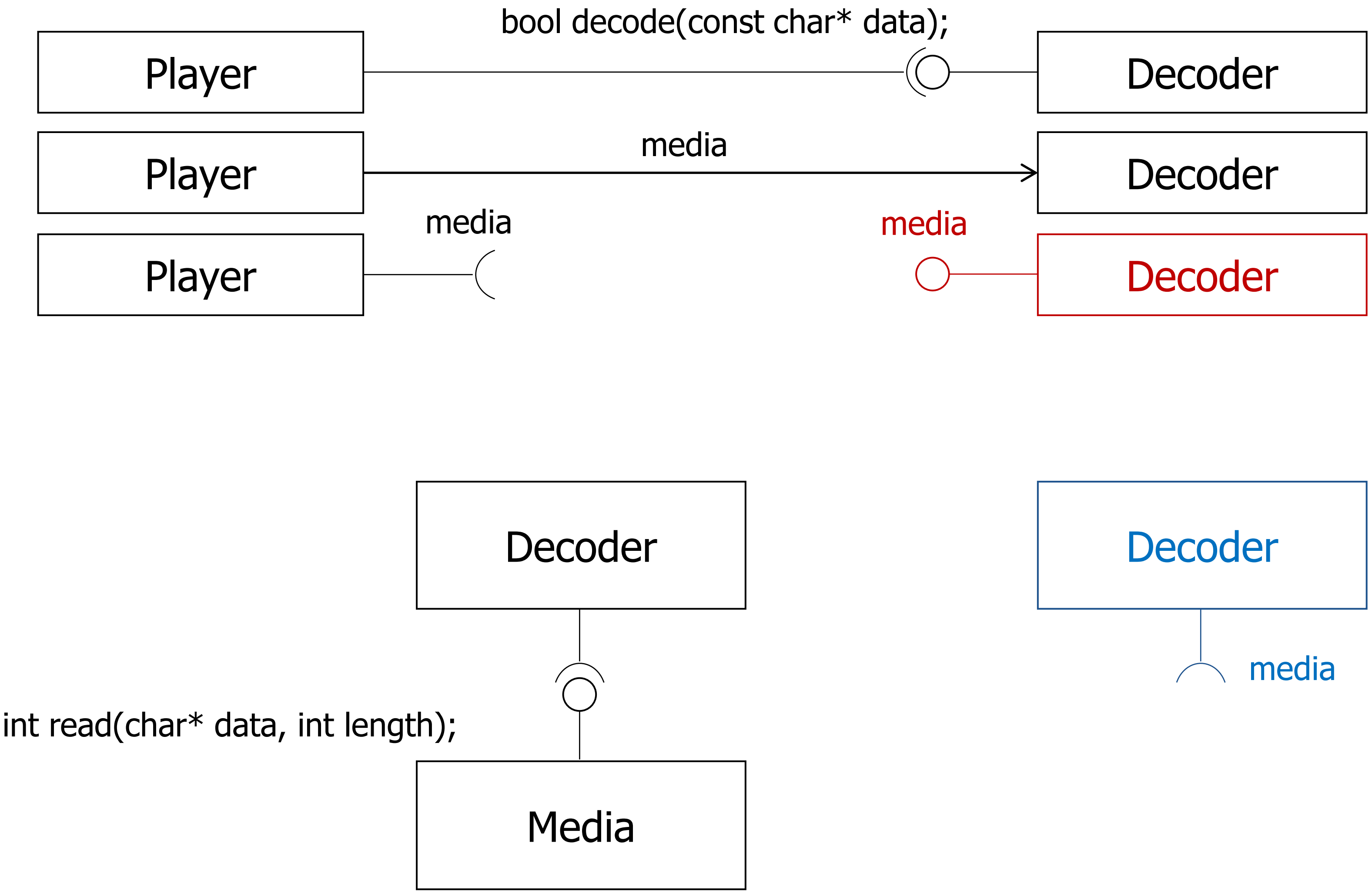
활동1. 시스템 정의


System Boundary
System
User
Providing Interface
Required Interface
External System

활동1. 시스템 정의


Player
bool decode(const char* data);
Decoder
Player
media
Decoder
Player
media
media
Decoder

활동1. 시스템 정의



bool decode(const char* data);

Player

Decoder

media

Player

Decoder

media

media

Player

Decoder

Decoder

int read(char* data, int length);

Media

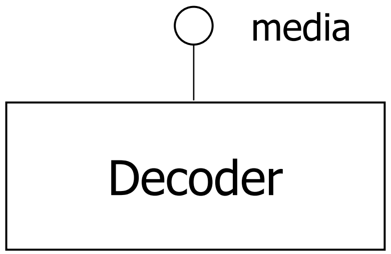
Decoder

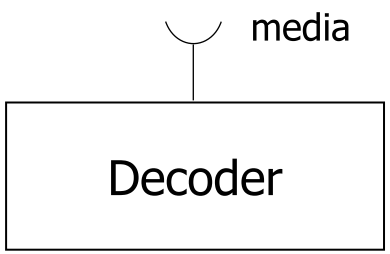
media

활동1. 시스템 정의

Providing interface: system will determine the interface and
the external using it will be developed later.

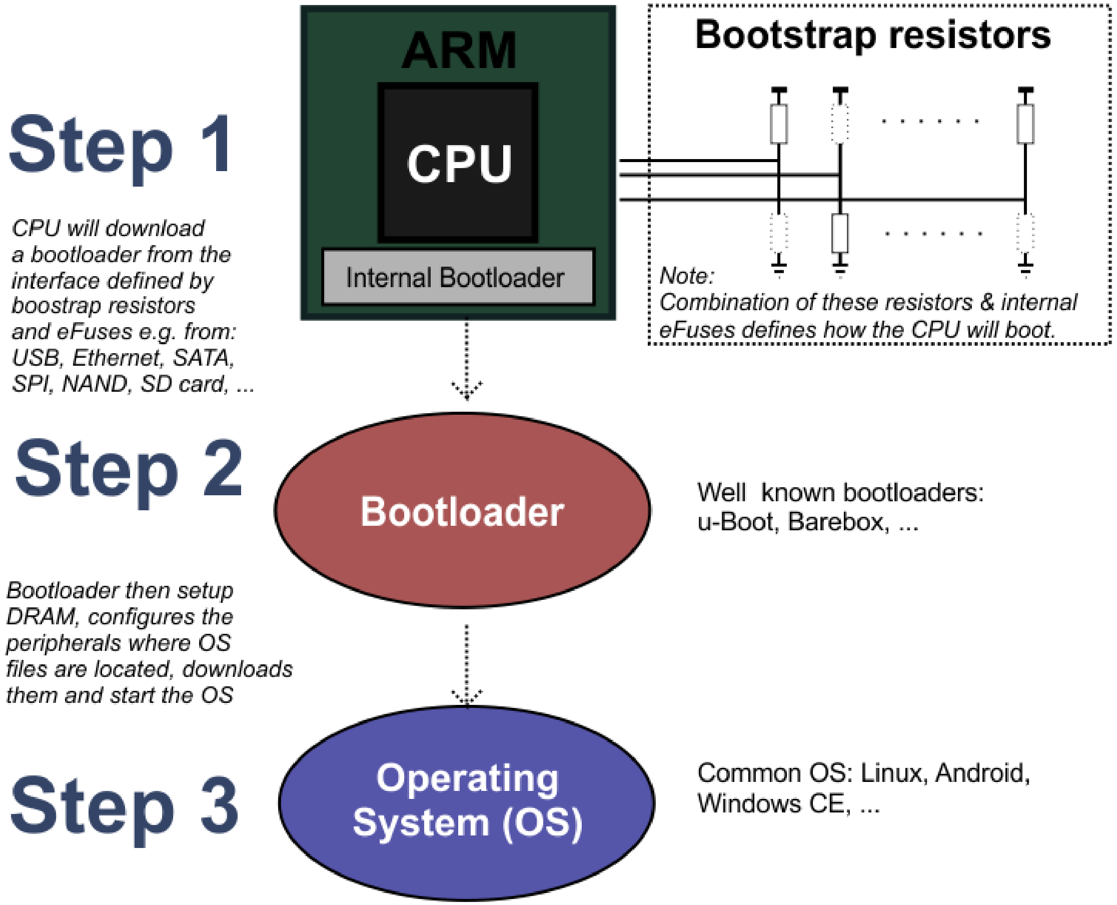
Required interface: the external system already exists and provides interface.
System uses it. The external system will not be changed later.





활동1. 시스템 정의

How does ARM Boot?



Step 1

CPU will download a bootloader from the interface defined by boostrap resistors and eFuses e.g. from: USB, Ethernet, SATA, SPI, NAND, SD card, ...

Note: Combination of these resistors & internal eFuses defines how the CPU will boot.

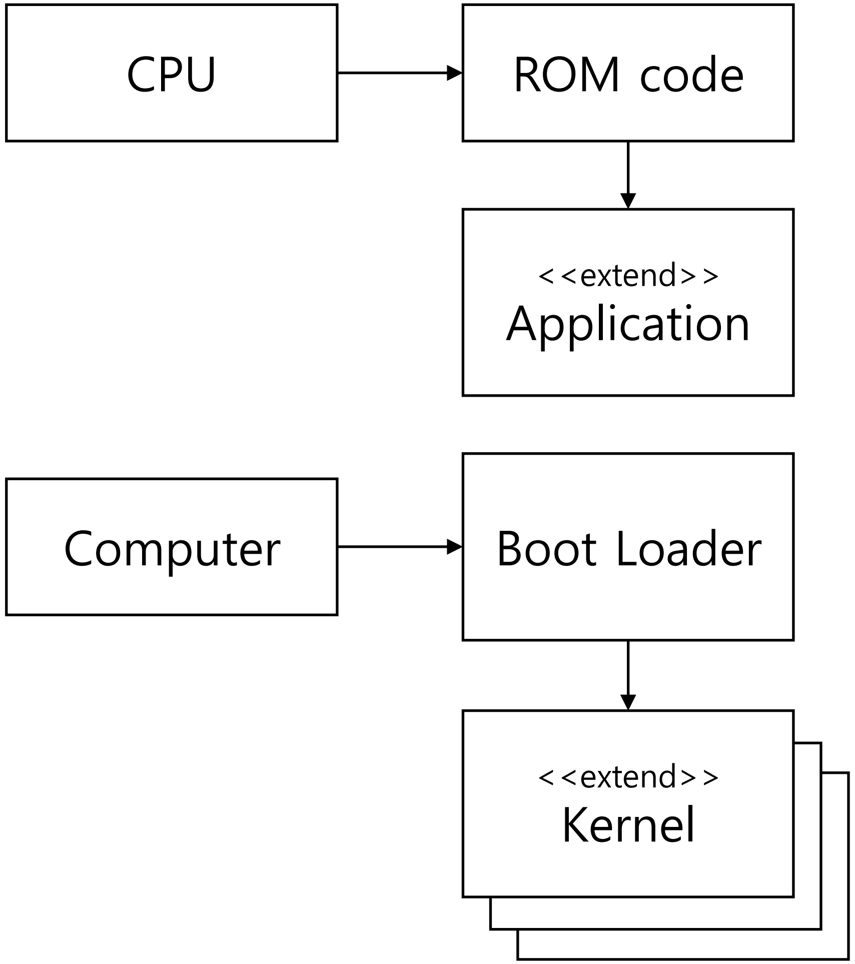
Step 2

Well known bootloaders: u-Boot, Barebox, ...

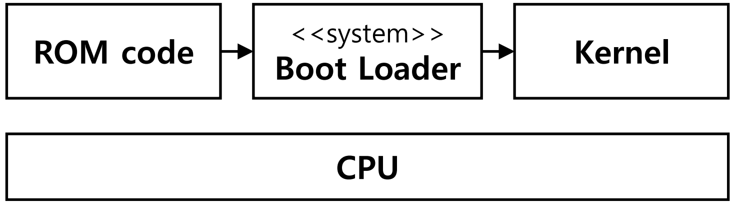
Bootloader then setup DRAM, configures the peripherals where OS files are located, downloads them and start the OS

Step 3

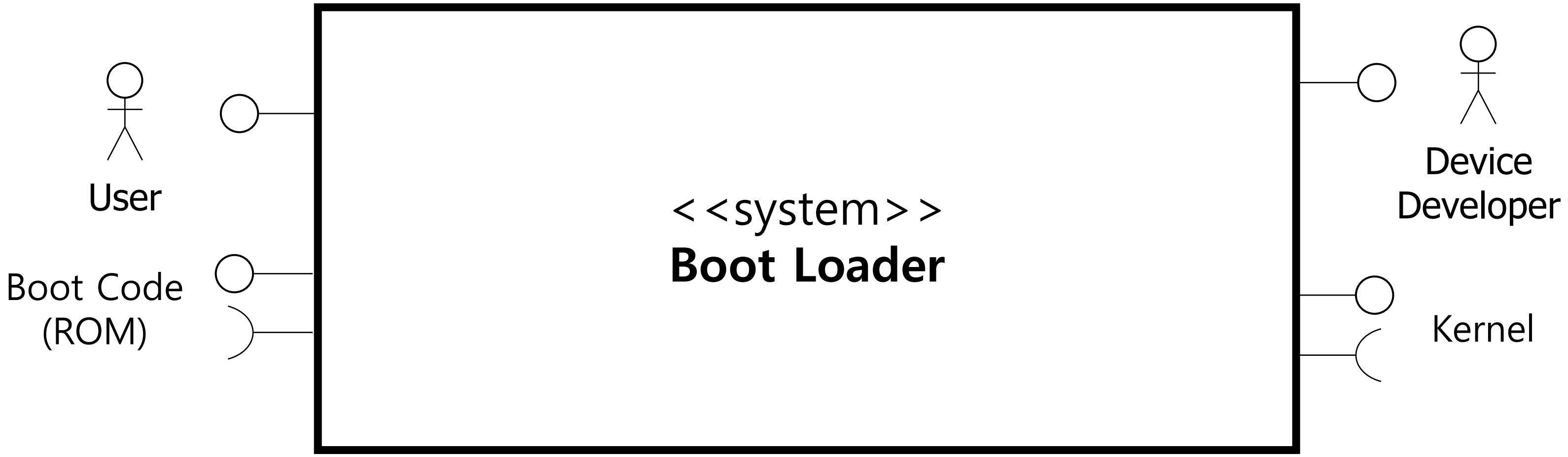
Common OS: Linux, Android, Windows CE, ...



활동1. 시스템 정의

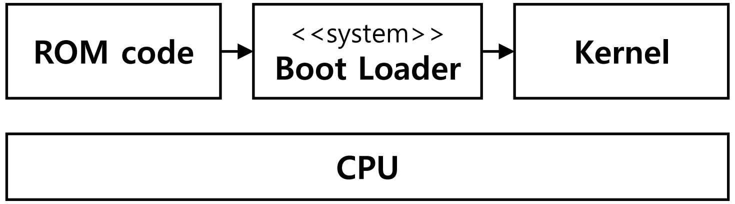


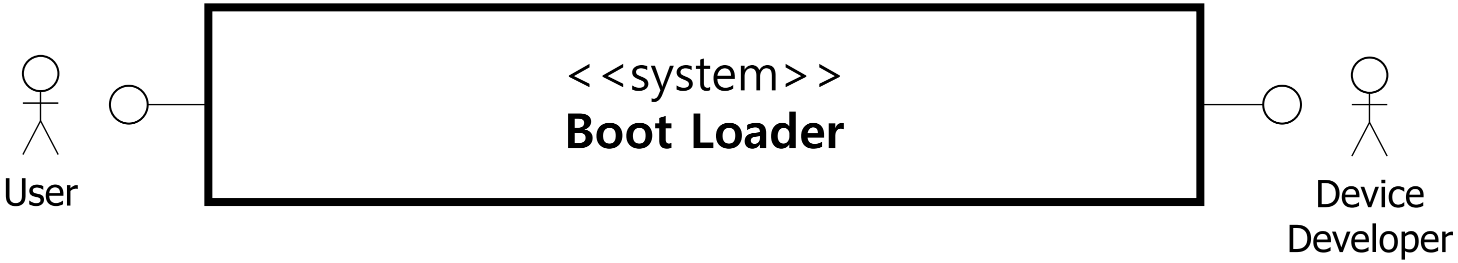
ROM code
<<system>>
Boot Loader
Kernel
CPU



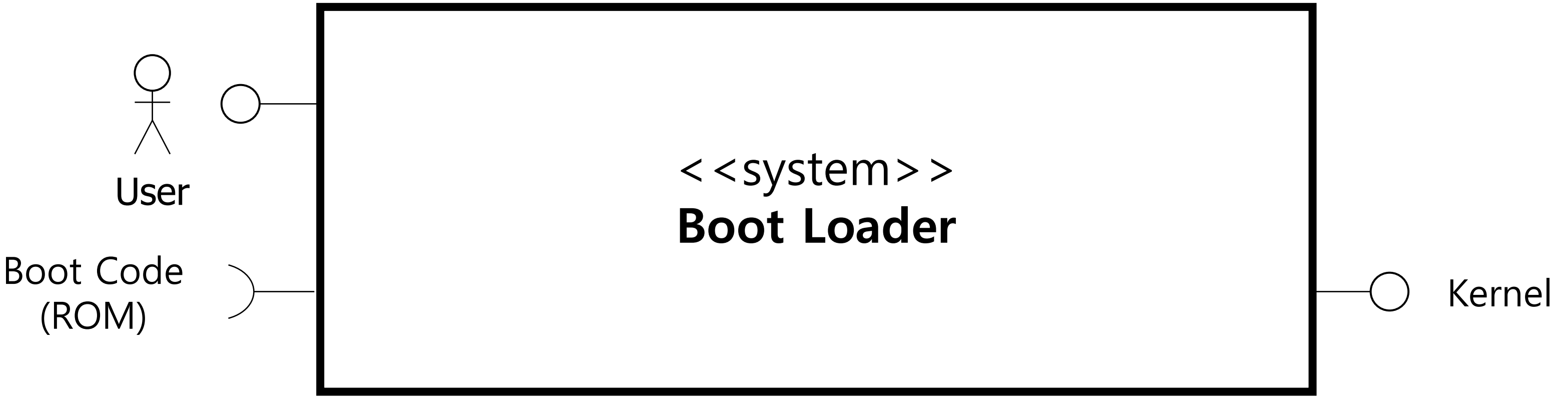
User
Boot Code
(ROM)
<<system>>
Boot Loader
Device
Developer
Kernel

활동1. 시스템 정의

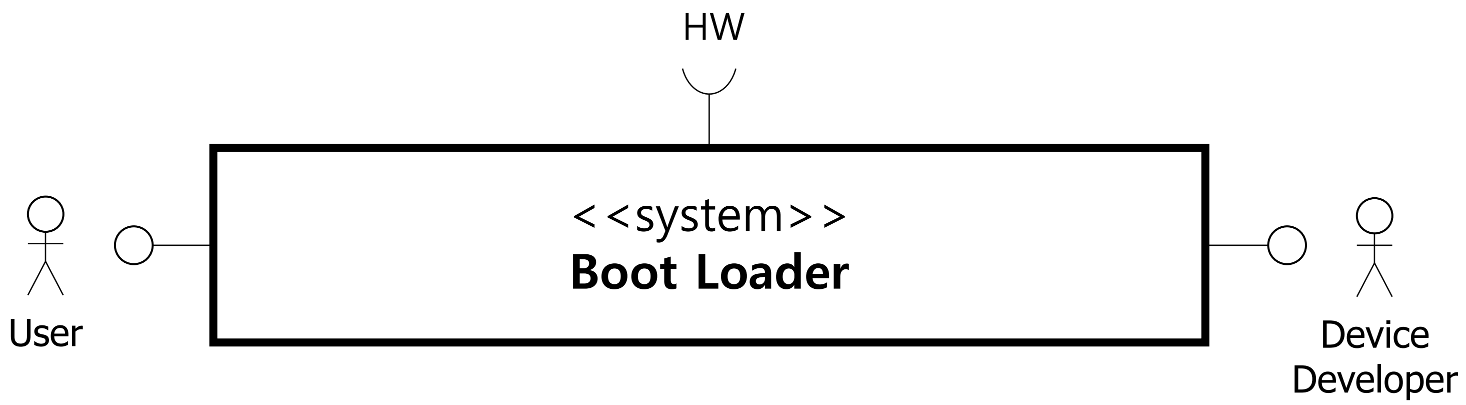

ROM code
<<system>>
Boot Loader
Kernel
CPU


User
<<system>>
Boot Loader
Device
Developer

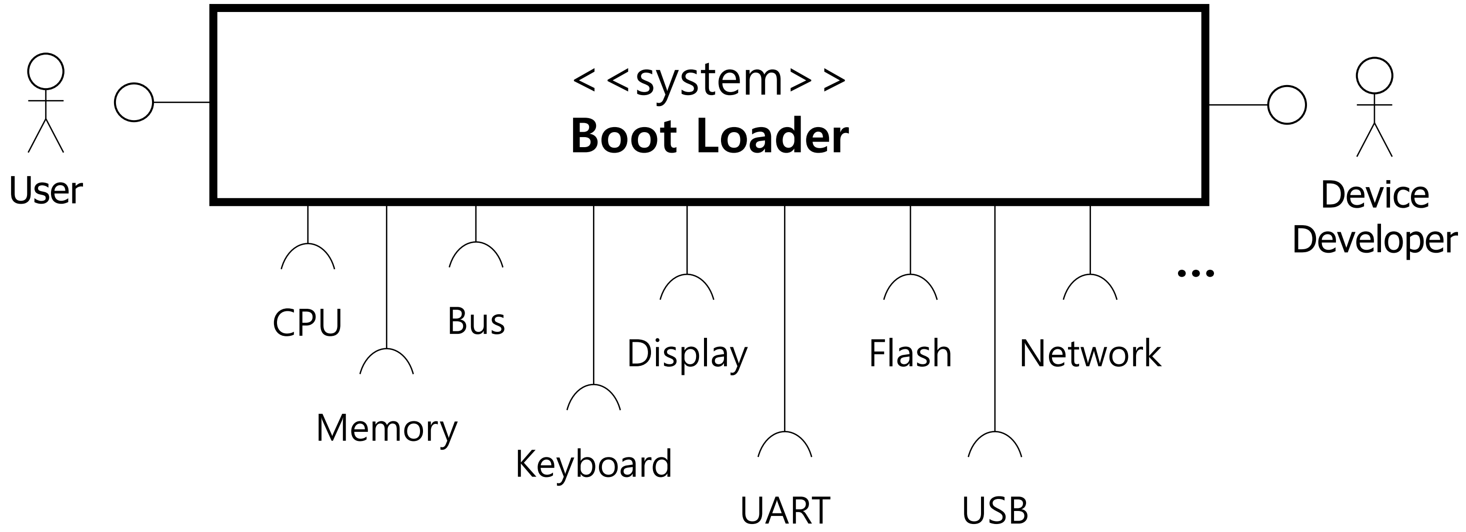
vs


User
Boot Code
(ROM)
<<system>>
Boot Loader
Kernel

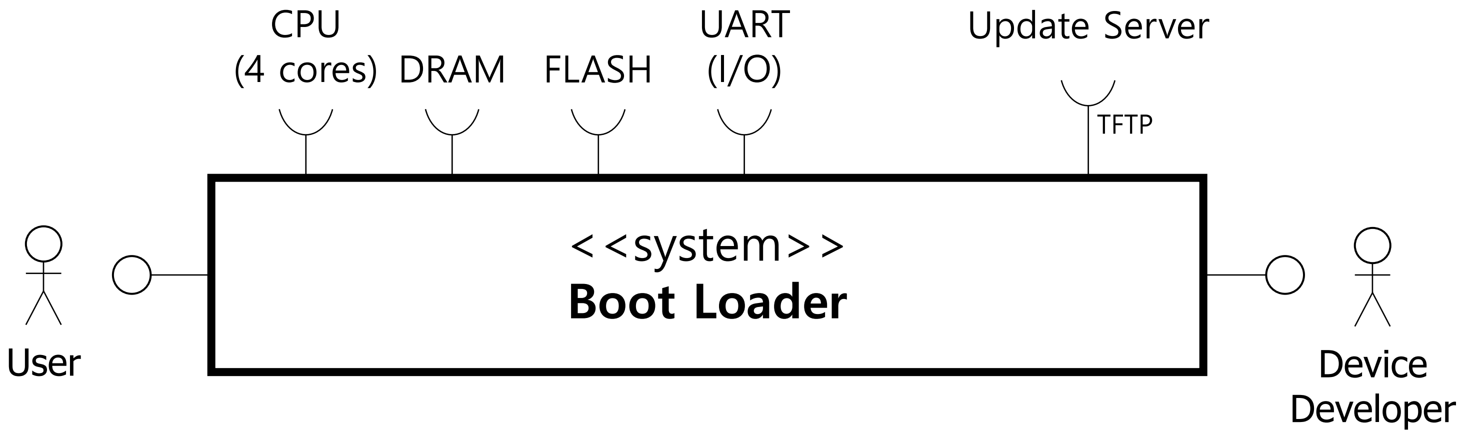
활동1. 시스템 정의


HW
User
<<system>>
Boot Loader
Device
Developer

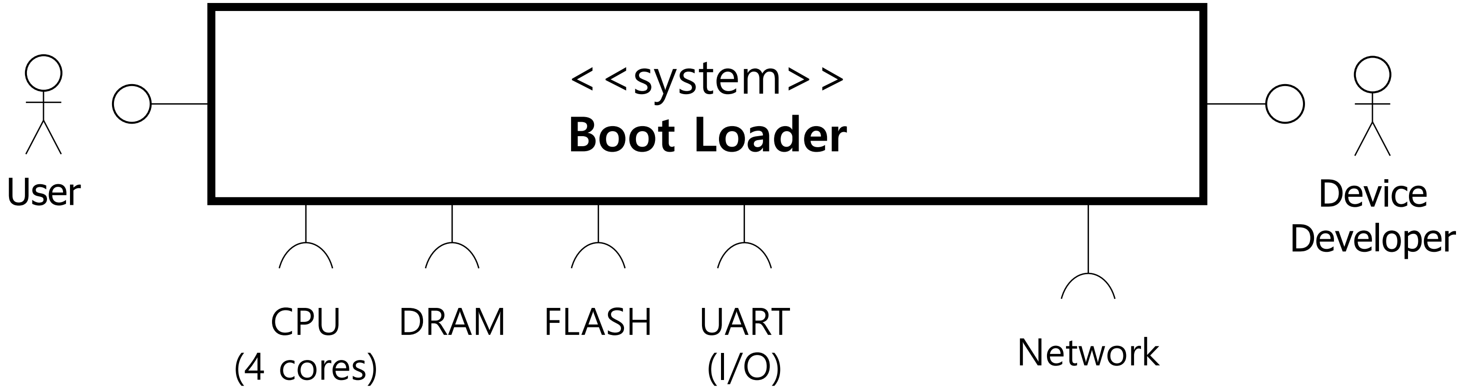
vs


User
<<system>>
Boot Loader
Device
Developer
CPU
Memory
Bus
Keyboard
Display
UART
Flash
USB
Network
...

활동1. 시스템 정의

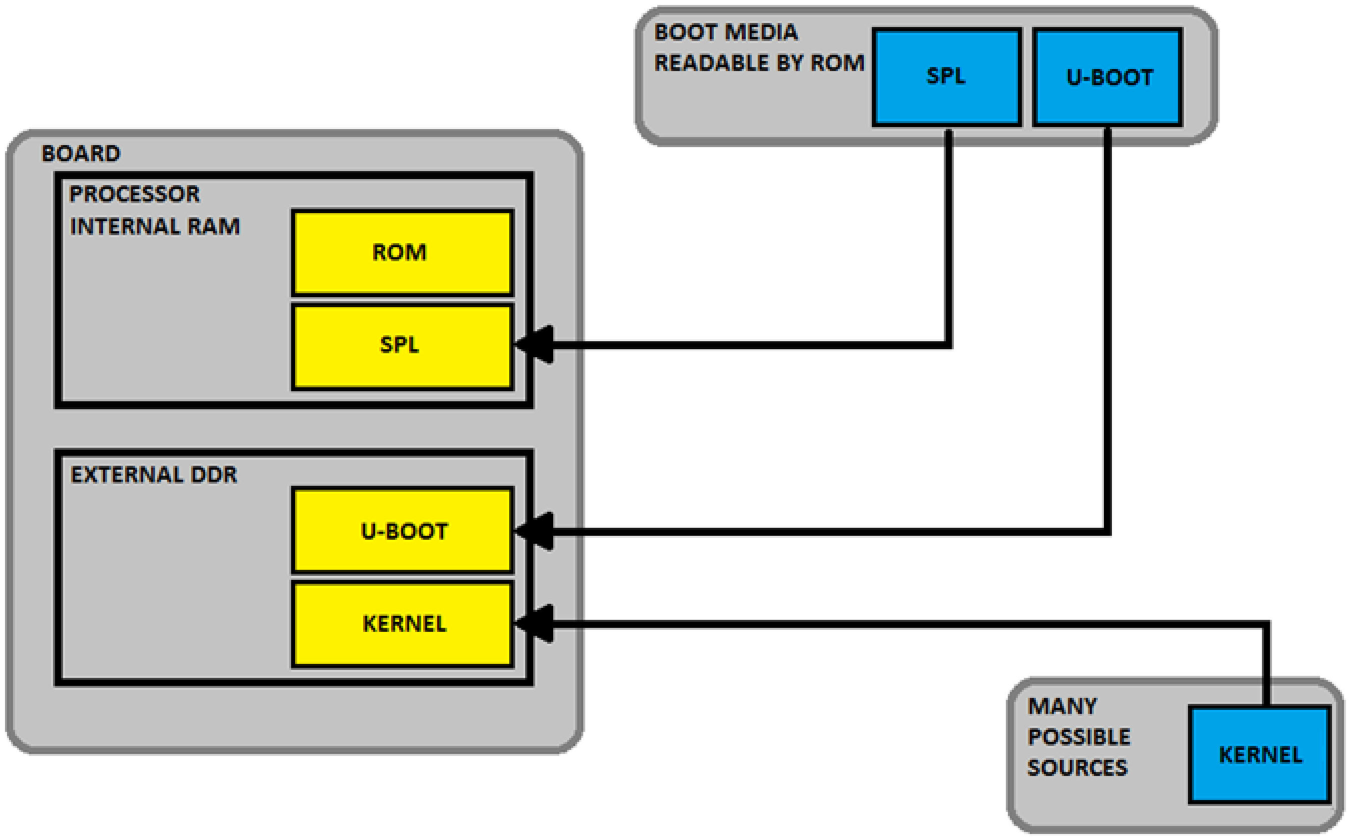

CPU
(4 cores)
DRAM
FLASH
UART
(I/O)
Update Server
TFTP
<<system>>
Boot Loader
User
Device
Developer

vs


<<system>>
Boot Loader
User
Device
Developer
CPU
(4 cores)
DRAM
FLASH
UART
(I/O)
Network

활동1. 시스템 정의

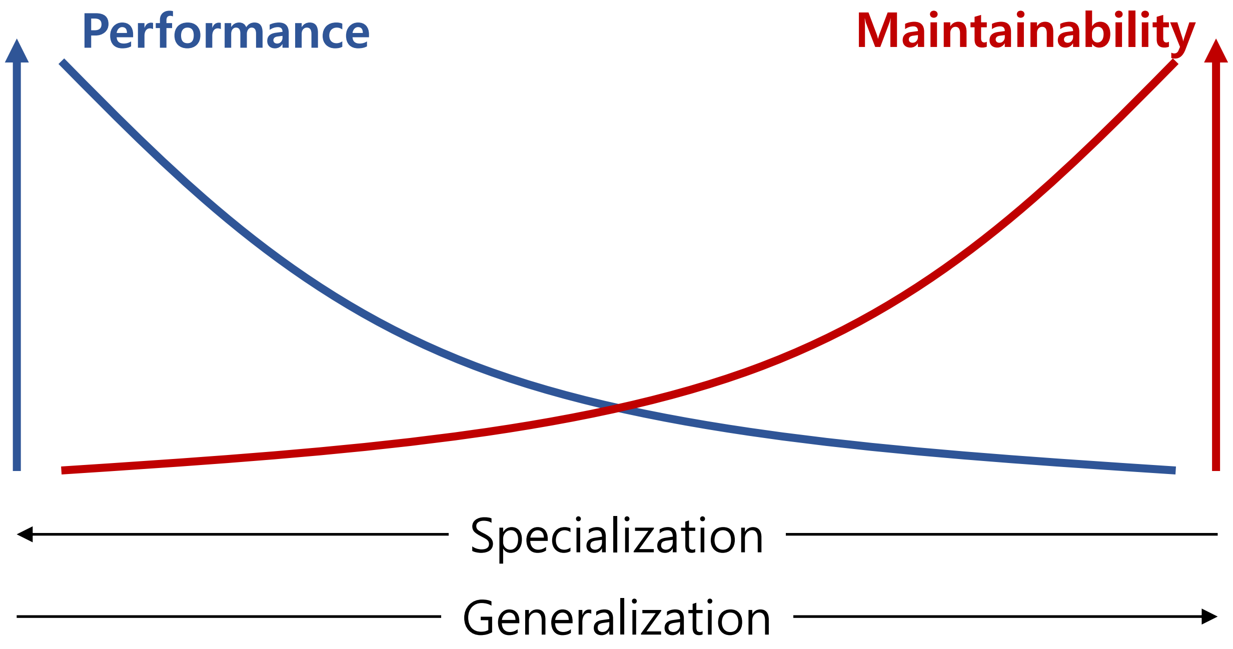
SPL (Secondary Program Loader)



Size of internal RAM or boot media can be a technical constraint.

활동1. 시스템 정의

Business Driver



Performance

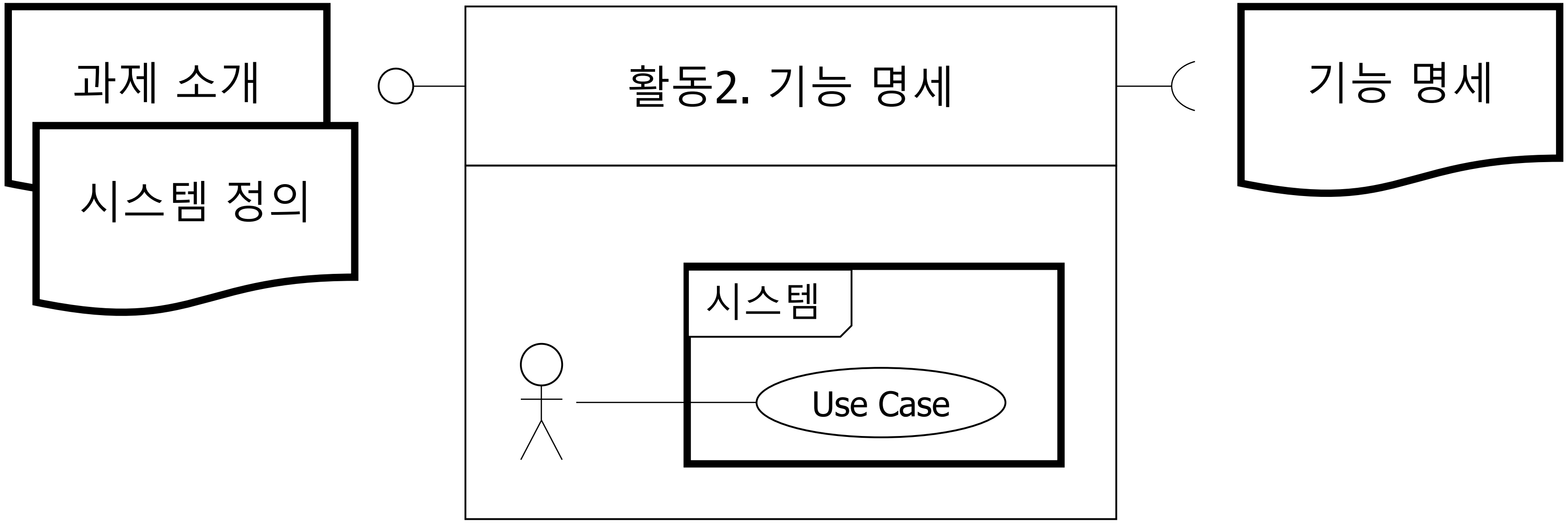
Maintainability

Specialization

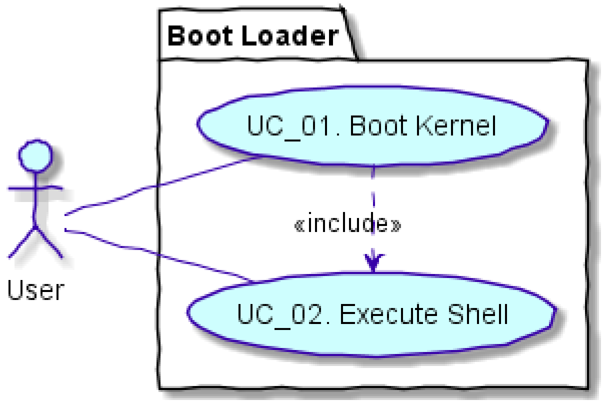
Generalization

활동2. 기능 명세

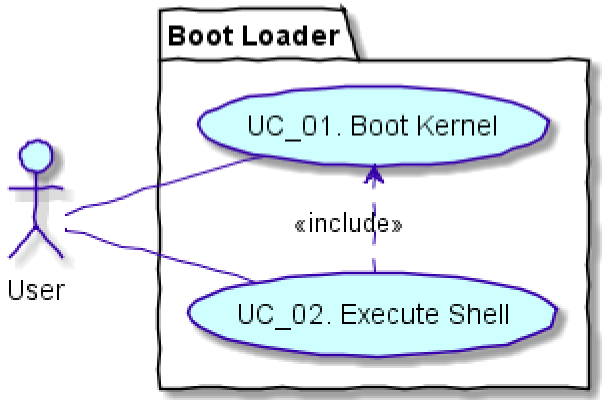
목적	시스템의 기능을 명세한다. 외부 시스템의 요청에 대한 시스템의 반응을 명세한다.



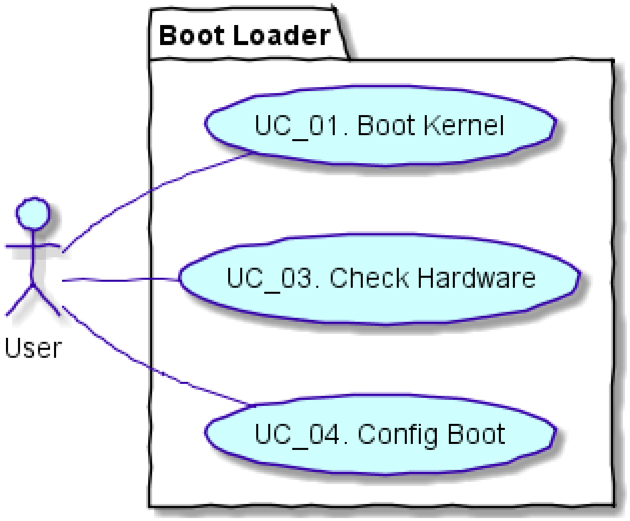
활동2. 기능 명세


Boot Loader
UC_01. Boot Kernel
«include»
UC_02. Execute Shell
User

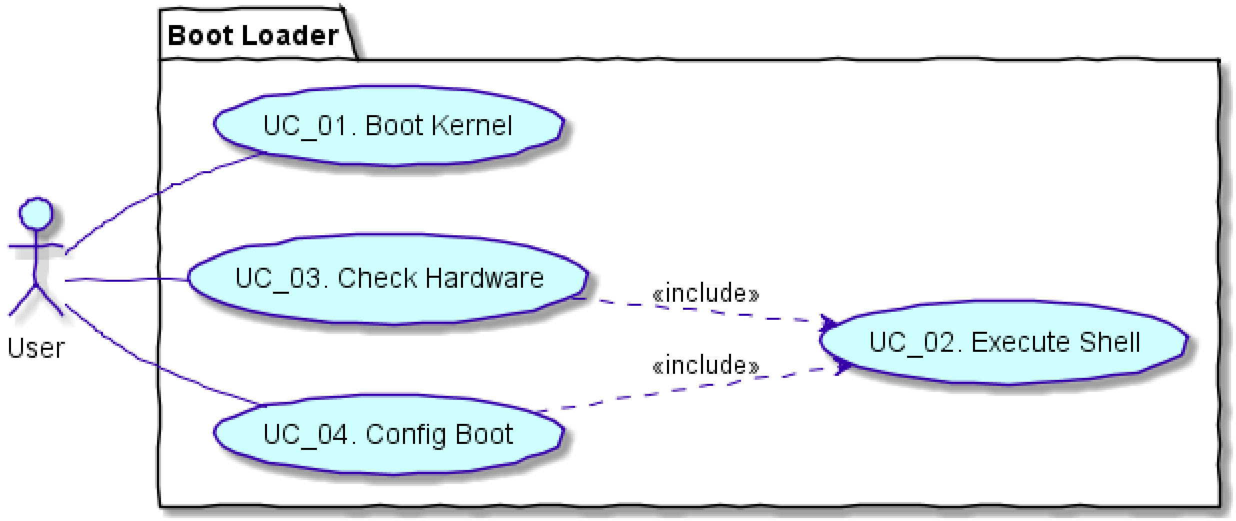
vs

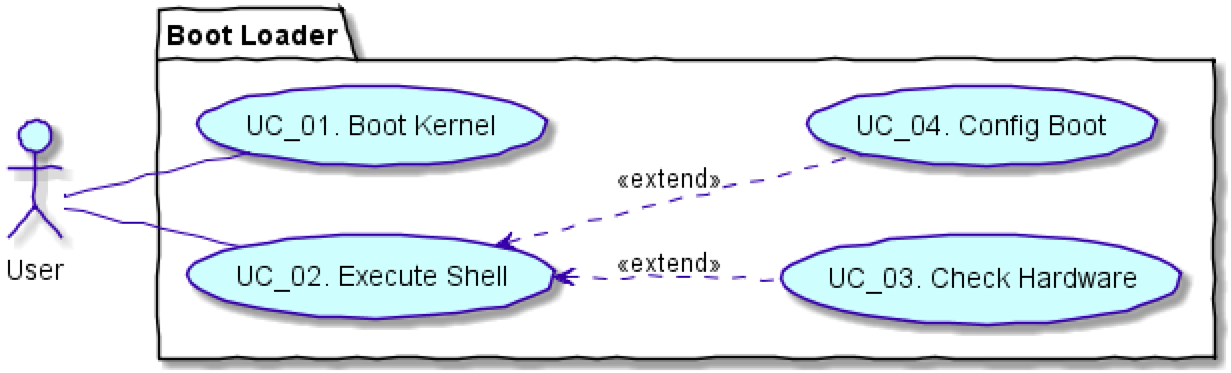

Boot Loader
UC_01. Boot Kernel
«include»
UC_02. Execute Shell
User

활동2. 기능 명세

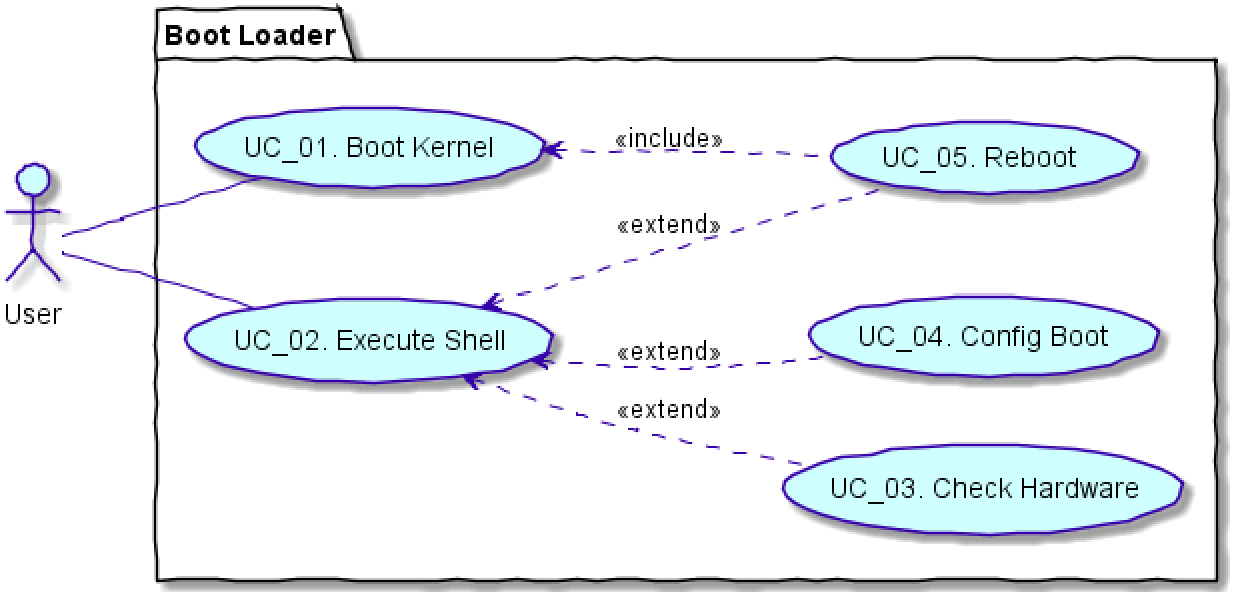

Boot Loader
UC_01. Boot Kernel
UC_03. Check Hardware
UC_04. Config Boot
User

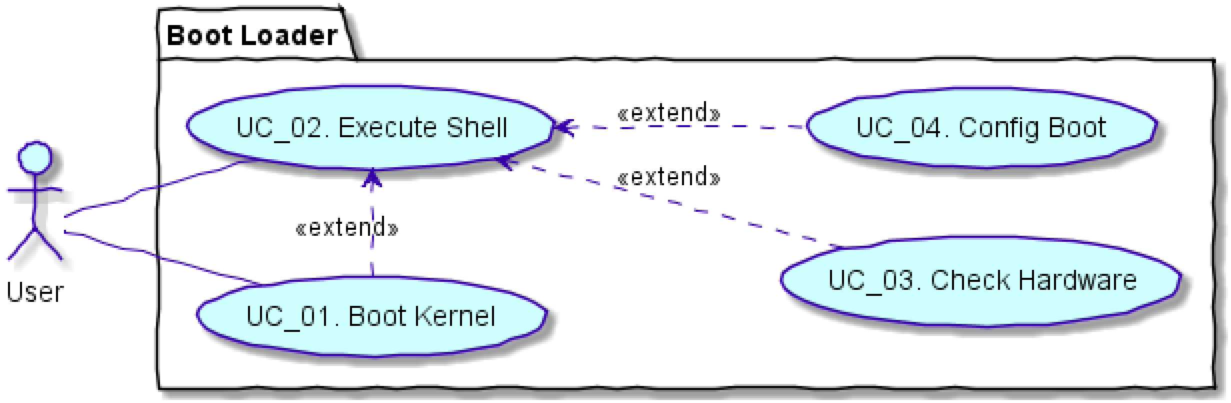
vs


Boot Loader
UC_01. Boot Kernel
UC_03. Check Hardware
«include»
UC_02. Execute Shell
«include»
UC_04. Config Boot
User

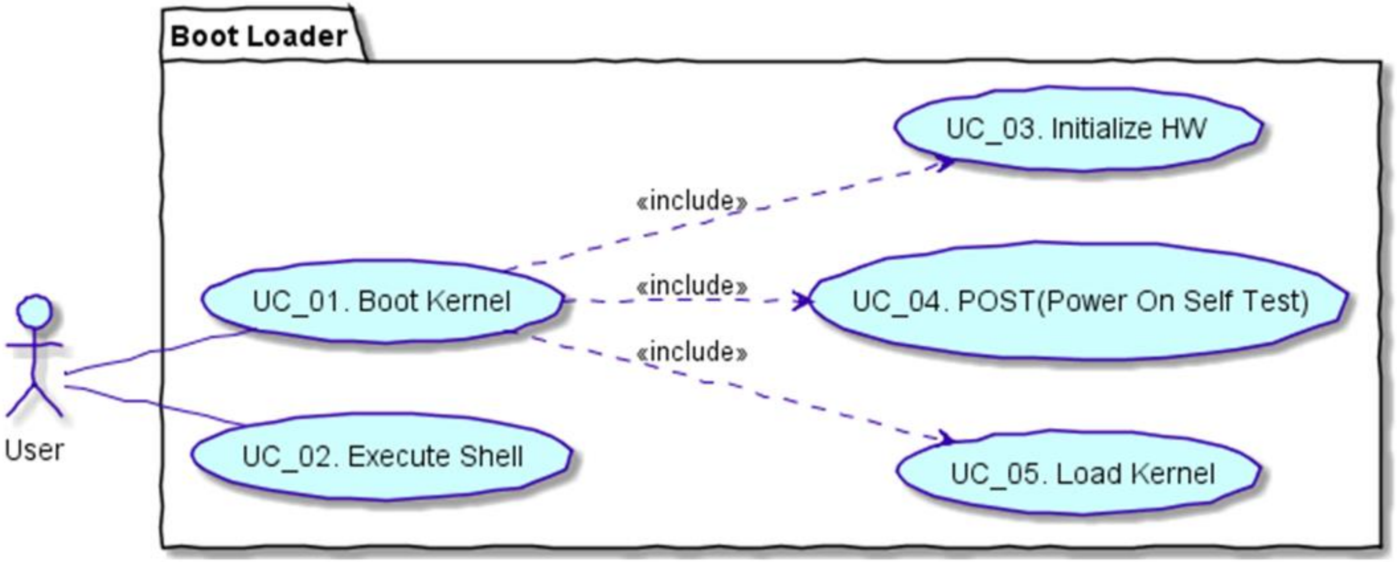

Boot Loader
UC_01. Boot Kernel
UC_04. Config Boot
«extend»
UC_02. Execute Shell
«extend»
UC_03. Check Hardware
User

활동2. 기능 명세


Boot Loader
User
UC_01. Boot Kernel
UC_05. Reboot
«include»
«extend»
UC_02. Execute Shell
UC_04. Config Boot
«extend»
«extend»
UC_03. Check Hardware


Boot Loader
User
UC_02. Execute Shell
UC_04. Config Boot
«extend»
«extend»
«extend»
UC_03. Check Hardware
UC_01. Boot Kernel

활동2. 기능 명세



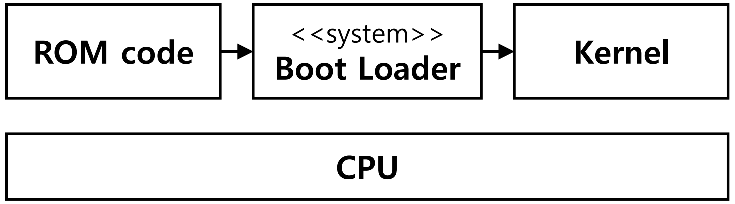
A use case is a metaphor to specify requirements.

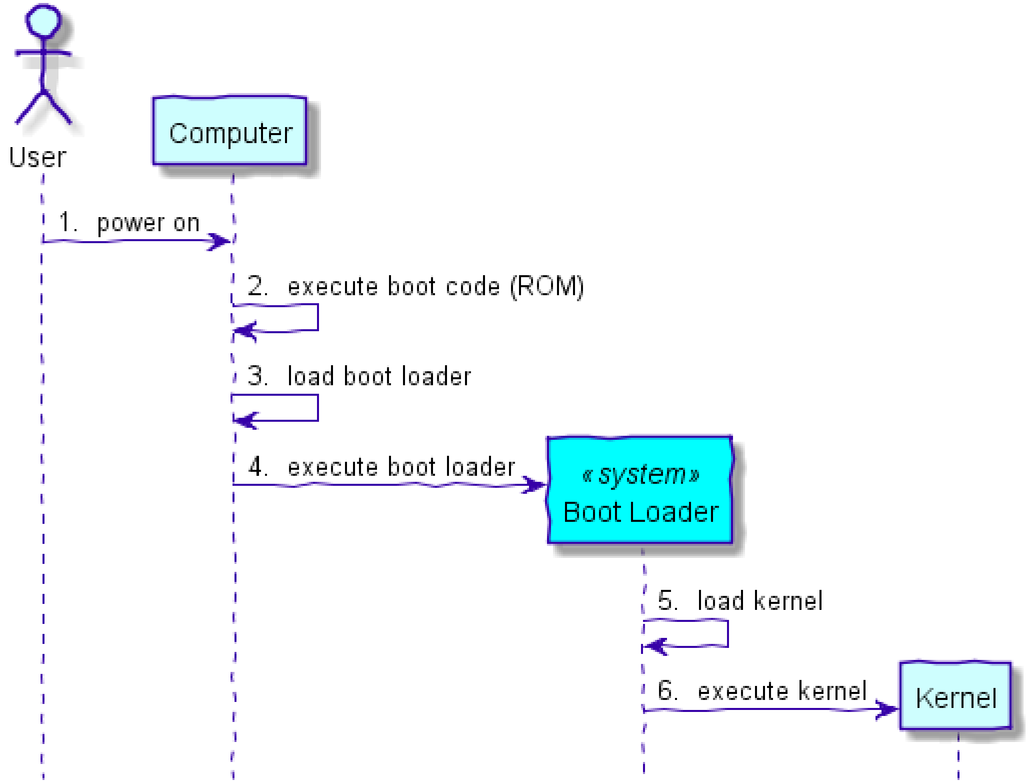
It is better identified by the user's intent,

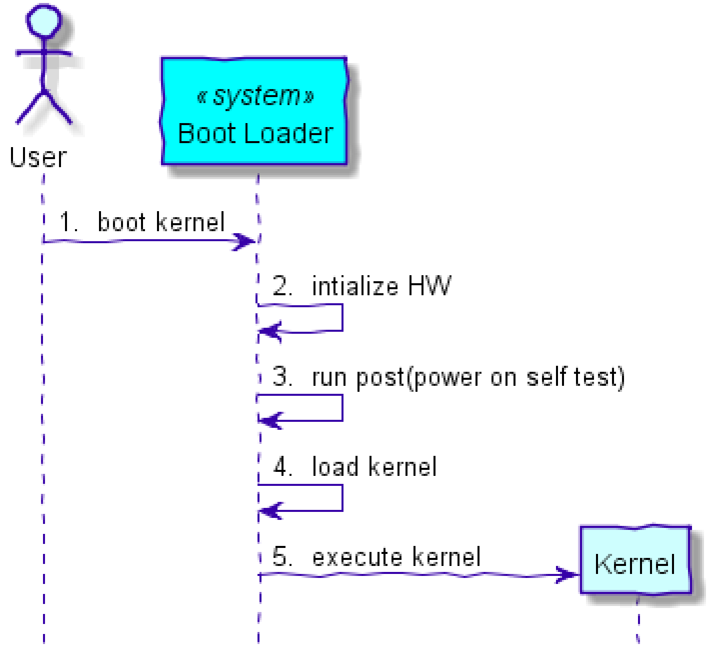
rather than internal functionalities or steps.

Don't design or organize use cases based on internal functionality.

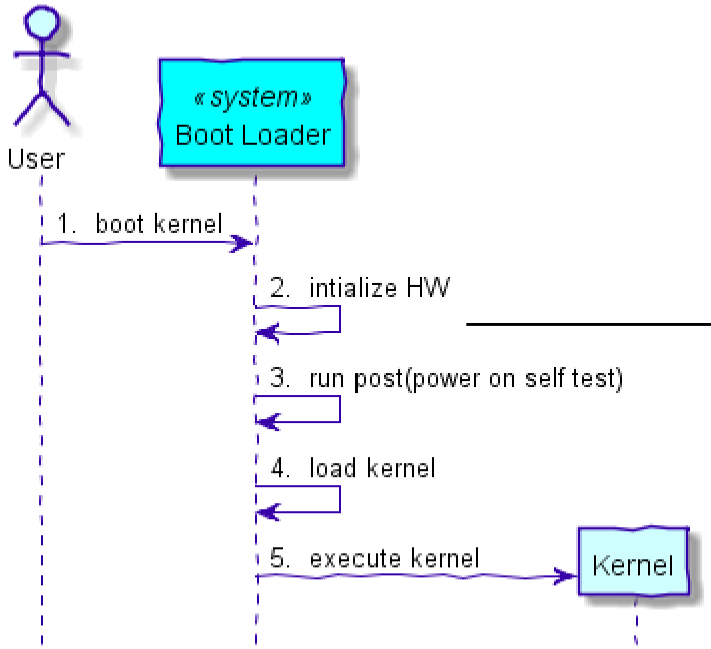
활동2. 기능 명세


ROM code
<<system>>
Boot Loader
Kernel
CPU


User
Computer
1. power on
2. execute boot code (ROM)
3. load boot loader
4. execute boot loader
«system»
Boot Loader
5. load kernel
6. execute kernel
Kernel


User
«system»
Boot Loader
1. boot kernel
2. intialize HW
3. run post(power on self test)
4. load kernel
5. execute kernel
Kernel

활동2. 기능 명세

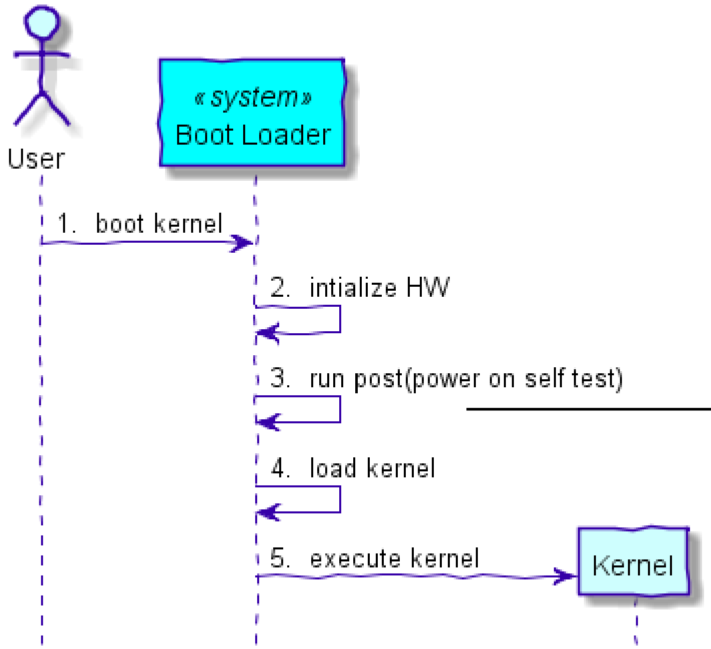


Initialization Flow – common/board_f.c, board_r.c

```
static const init_fnc_t init_sequence_f[] = {
    setup_mon_len,
#ifdef CONFIG_OF_CONTROL
    fdtdec_setup,
#endif
#ifdef CONFIG_TRACE_EARLY
    trace_early_init,
#endif
    initf_malloc,
    log_init,
    initf_bootstage,
#ifdef CONFIG_BLOBLIST
    bloblist_init,
#endif
...
    clear_bss,
    NULL,
};

void board_init_f(ulong boot_flags)
{
    gd->flags = boot_flags;
    gd->have_console = 0;

    if (initcall_run_list(init_sequence_f))  hang();
}
```

```
static const init_fnc_t init_sequence_r[] = {
    initr_trace,
    initr_reloc,
#if defined(CONFIG_ARM) || defined(CONFIG_RISCV)
    initr_caches,
#endif
    initr_reloc_global_data,
#if defined(CONFIG_SYS_INIT_RAM_LOCK) && ₩
     defined(CONFIG_E500)
    initr_unlock_ram_in_cache,
#endif
    initr_barrier,
    initr_malloc,
...
    run_main_loop,
};

void board_init_r(gd_t *new_gd, ulong dest_addr)
{
    gd = new_gd;
    gd->flags &= ~GD_FLG_LOG_READY;

    if (initcall_run_list(init_sequence_r))  hang();

   /* NOTREACHED - run_main_loop() does not return */
    hang();
}
```

활동2. 기능 명세

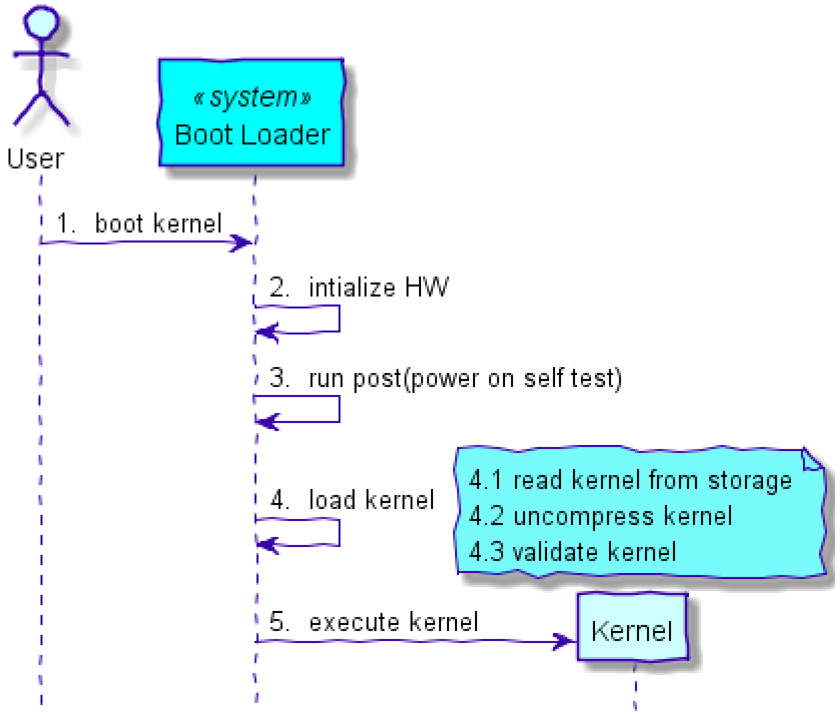


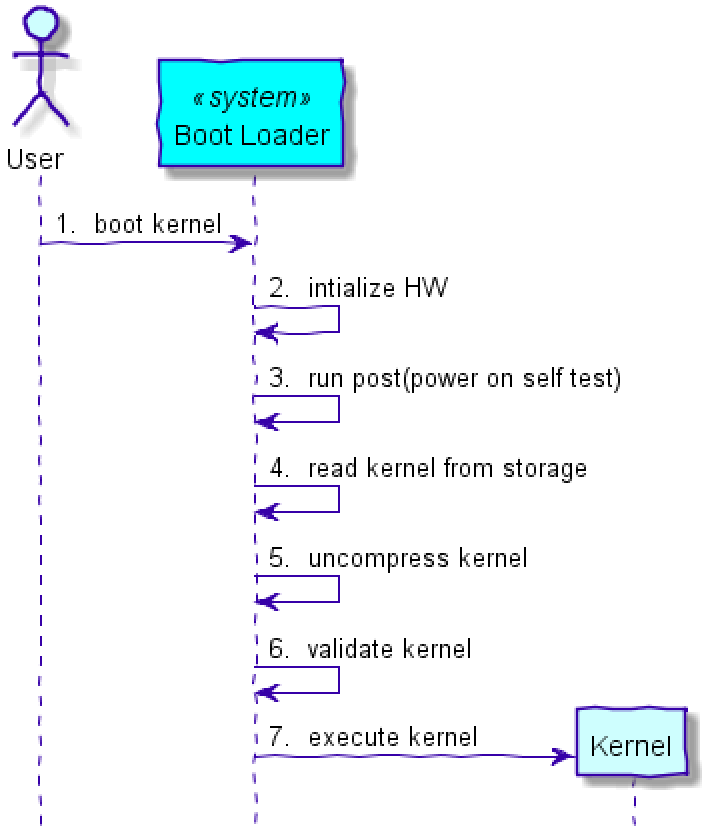
Initialization Flow – post/post.c, tests.c

```
struct post_test post_list[] =
{
#if CONFIG_POST & CONFIG_SYS_POST_OCM
    {   "OCM test",
        "ocm",
        "This test checks on chip memory (OCM).",
        POST_ROM | POST_ALWAYS | POST_PREREL |
        POST_CRITICAL | POST_STOP,
        &ocm_post_test,
        NULL,
        NULL,
        CONFIG_SYS_POST_OCM
    },
#endif
#if CONFIG_POST & CONFIG_SYS_POST_CACHE
    {   "Cache test",
        "cache",
        "This test verifies the CPU cache operation.",
        POST_RAM | POST_ALWAYS,
        &cache_post_test,
        NULL,
        NULL,
        CONFIG_SYS_POST_CACHE
    },
#endif
...
};
```

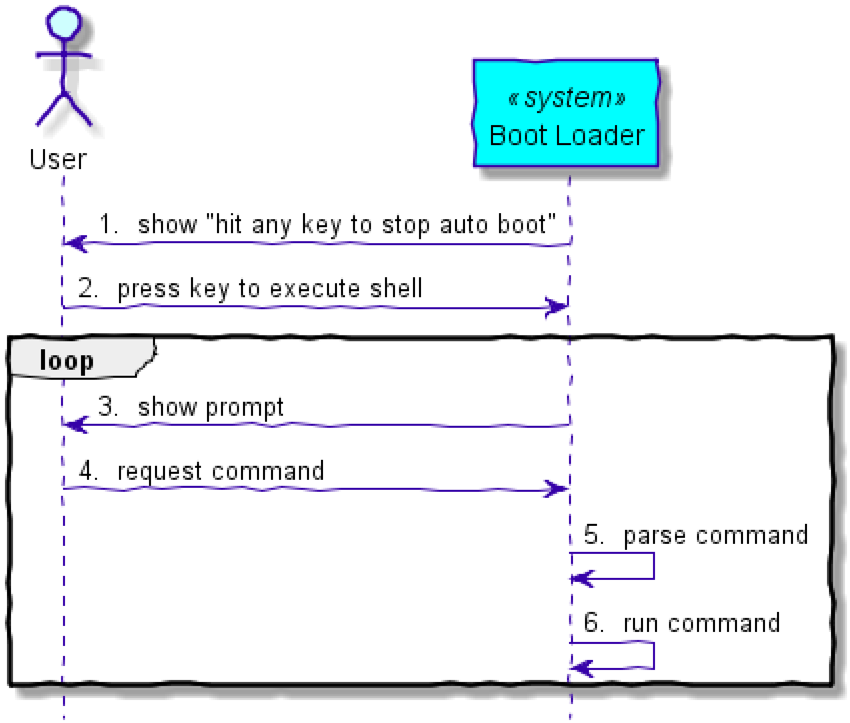
```
int post_run(char *name, int flags)
{
    unsigned int i, last;
    int test_flags[POST_MAX_NUMBER];
    post_get_flags(test_flags);

    if (post_bootmode_get(&last) & POST_POWERTEST) {
        if (last & POST_FAIL_SAVE) {
        last &= ~POST_FAIL_SAVE;
        gd->flags |= GD_FLG_POSTFAIL;
    }

    if (last < post_list_size &&
       (flags & test_flags[last] & POST_ALWAYS) &&
       (flags & test_flags[last] & POST_MEM)) {
        post_run_single(post_list + last,
            test_flags[last], flags | POST_REBOOT,
            last);

        for (i = last + 1; i < post_list_size; i++) {
            if (gd->flags & GD_FLG_POSTSTOP) break;
            post_run_single(post_list + i,
                test_flags[i], flags, i);
        }
    }
}
```

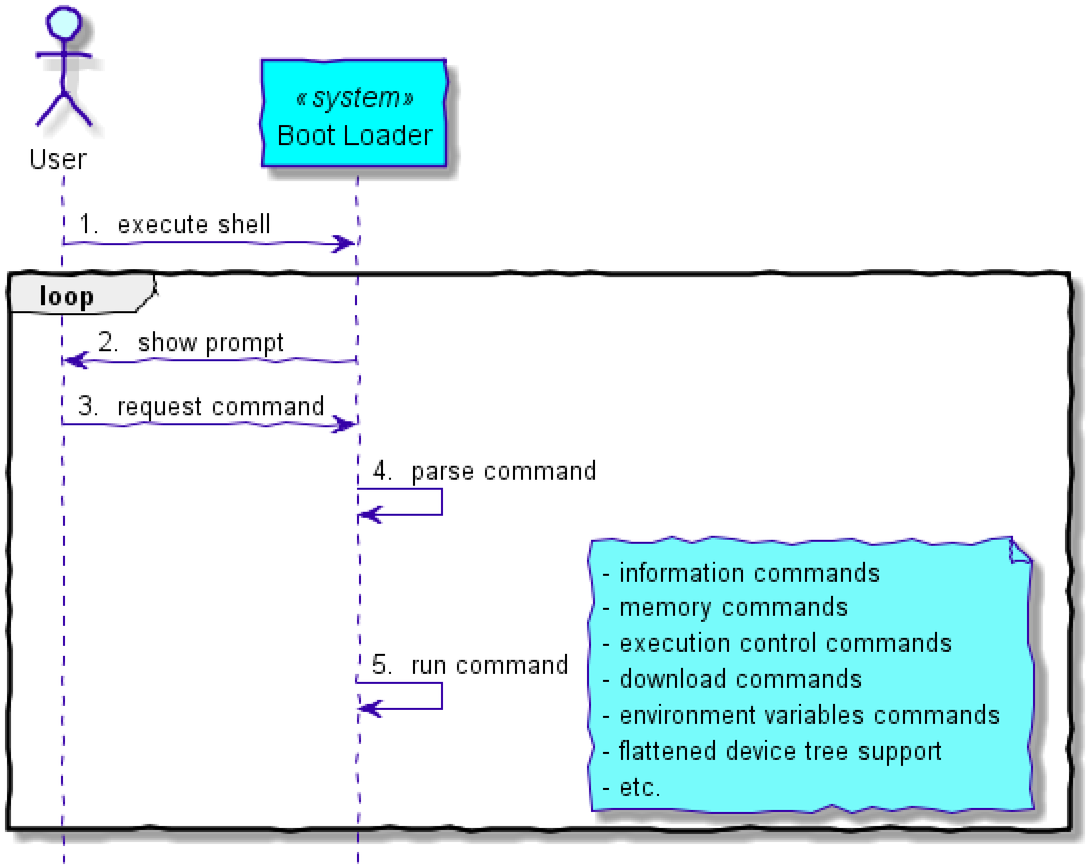
활동2. 기능 명세


User
«system»
Boot Loader
1. boot kernel
2. intialize HW
3. run post(power on self test)
4. load kernel
4.1 read kernel from storage
4.2 uncompress kernel
4.3 validate kernel
5. execute kernel
Kernel


User
«system»
Boot Loader
1. boot kernel
2. intialize HW
3. run post(power on self test)
4. read kernel from storage
5. uncompress kernel
6. validate kernel
7. execute kernel
Kernel

활동2. 기능 명세

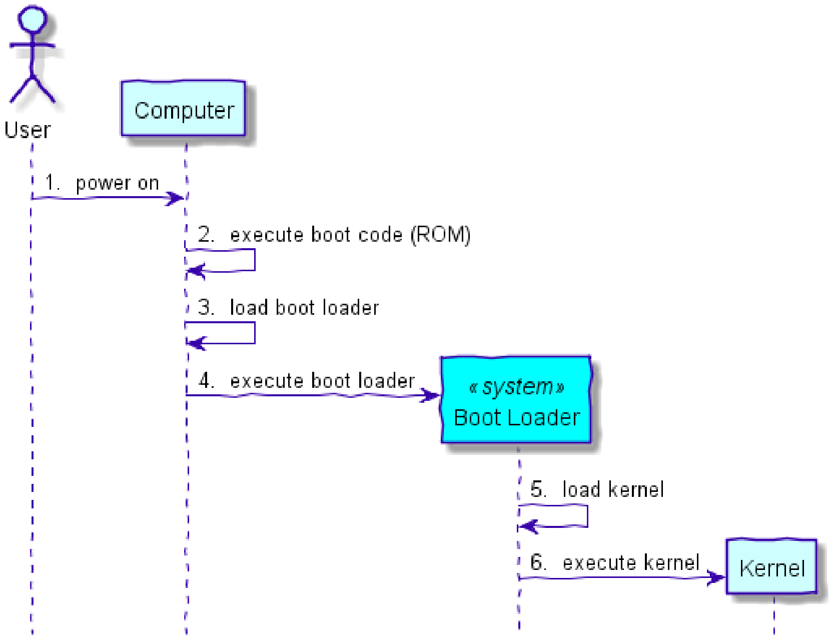


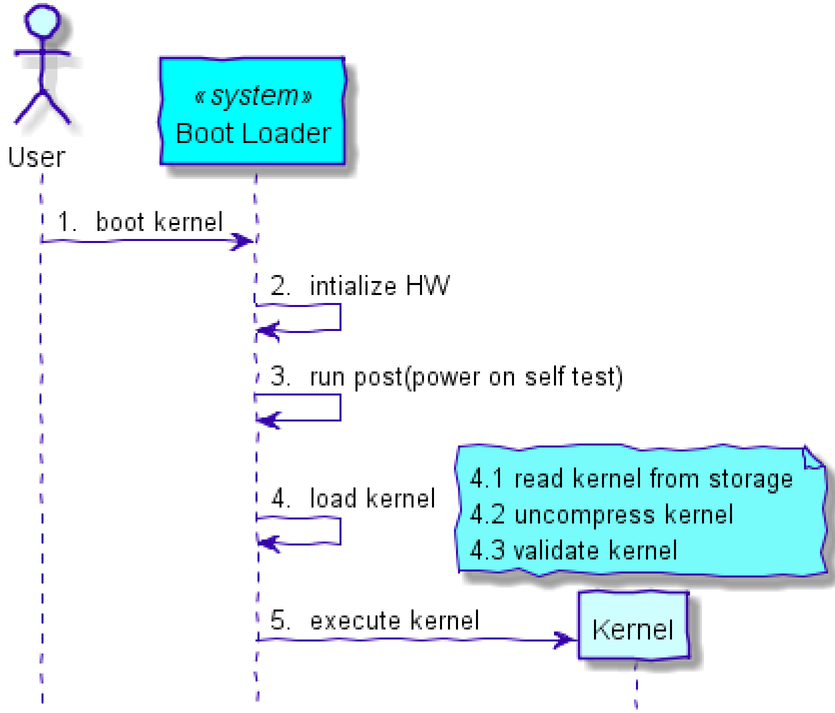


What the system needs to do for actor's request is specified in more detail, rather than user interface.

활동2. 기능 명세

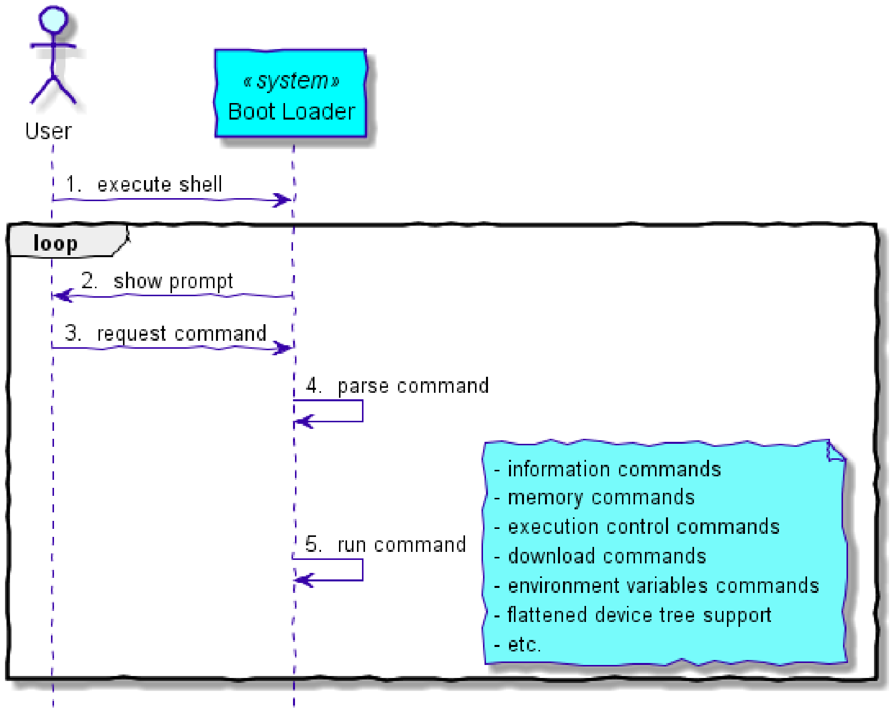
UC_01	Boot kernel
Description	System loads and executes kernel.
Actor	User
Pre-condition	[DEVICE_OFF] Device is powered off.
Post-condition	[KERNEL] Kernel is executed.
Basic Flow	1. User requests system (boot loader) to boot kernel. 1.1 User powers on device. 1.2 CPU executes boot code (ROM). 1.3 Boot code loads boot loader. 1.4 Boot code executes boot loader. 2. System initializes HW. 3. System runs POST (Power On Self Test). 4. System loads kernel. 4.1 System reads kernel from storage. 4.2 System decompresses kernel. 4.3 System validates kernel. 5. System executes kernel.
Additional Flow	2. If HW initialization fails, system halts with a notification.
Additional Flow	4.3 If kernel validation fails, system loads the previously successful kernel.





활동2. 기능 명세

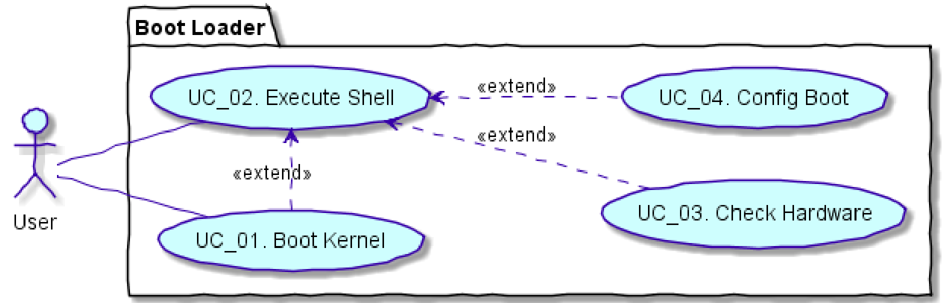
UC_02	Execute Shell
Description	System provides CLI (Command Line Interface) to user.
Actor	User
Pre-condition	[DEVICE_OFF] Device is powered off.
Post-condition	[SHELL] System provides CLI to user.
Basic Flow	1. User requests system (boot loader) to provide CLI. 1.1 System shows “press any key to stop autoboot” with timeout. 1.2 User presses any key to stop autoboot within timeout. 2. System shows prompt. 3. User requests system to execute command. 4. System parses command. 5. System runs command. 6. System goes to 2 (loop).
Additional Flow	1.2 Otherwise, system continues to boot kernel (UC_01).
Additional Flow	3. System provides many commands, extended by UC_03, UC_04 and UC_01, etc.



No need to specify user interface in detail.

UI can be easily changed ➔ UI modifiability.

Good to specify extension point. ➔ CLI modifiability.

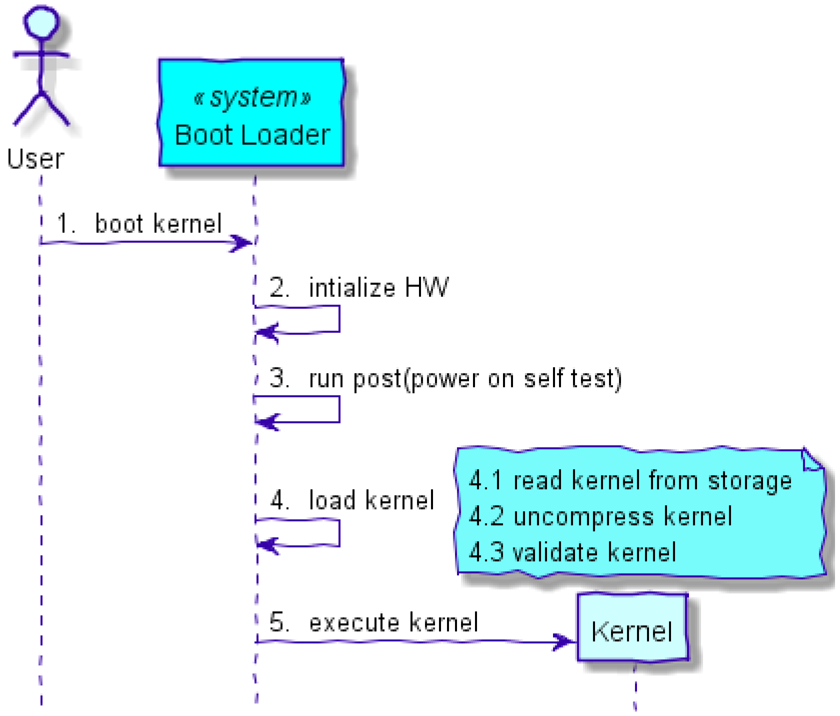


Don’t omit the subject.

Don’t use the passive voice.

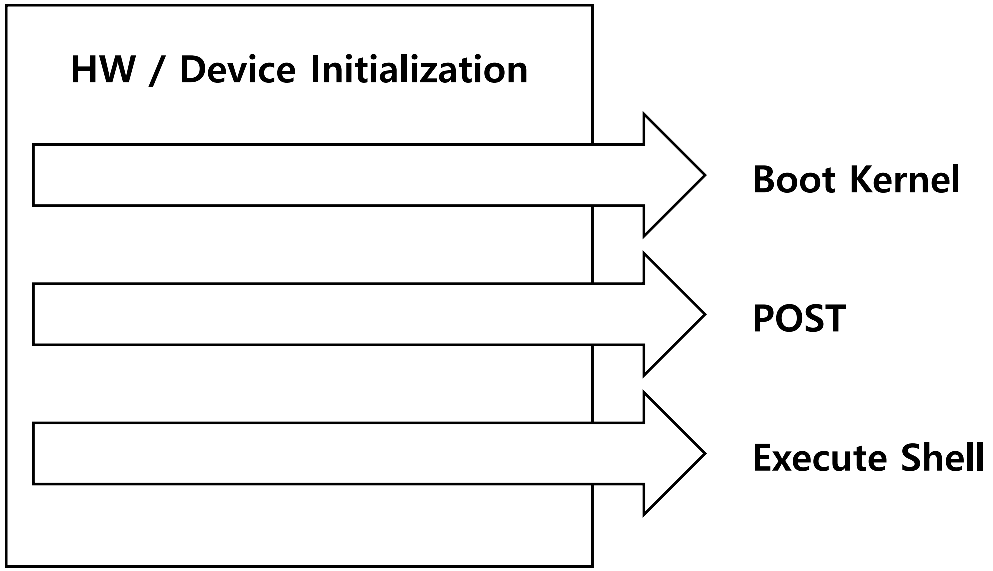
3. System gets command from user.

활동2. 기능 명세

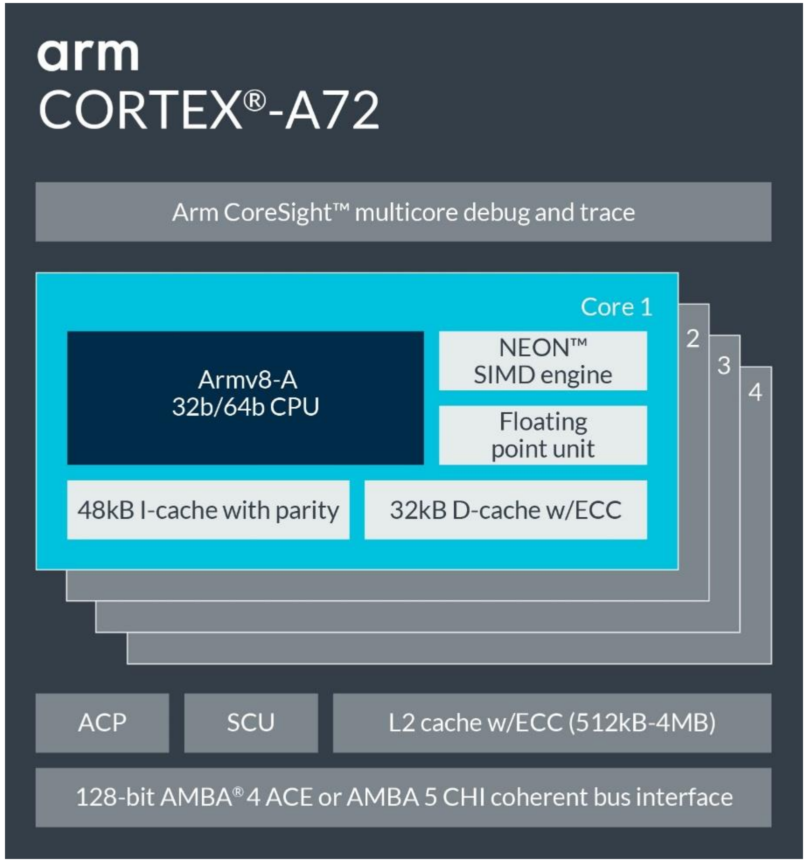


Architectural Concern

- Concurrency -



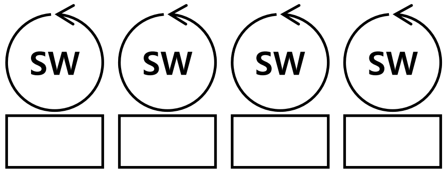
활동2. 기능 명세



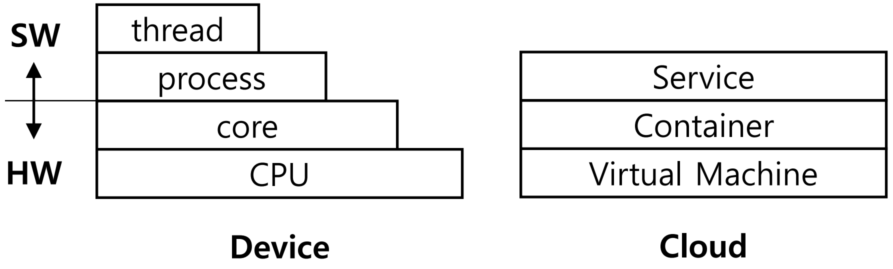
Architectual Concern

- Concurrency -

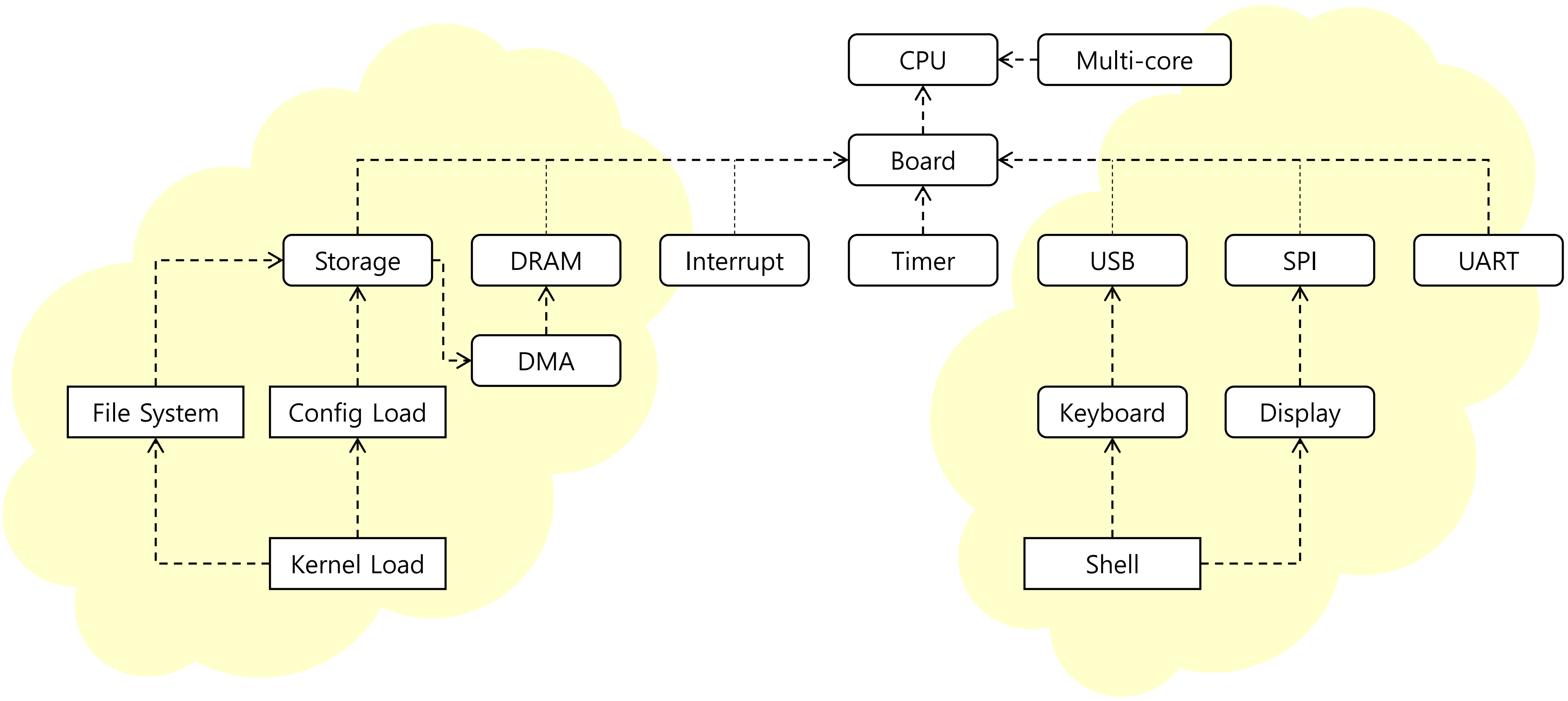
Concurrent Executions



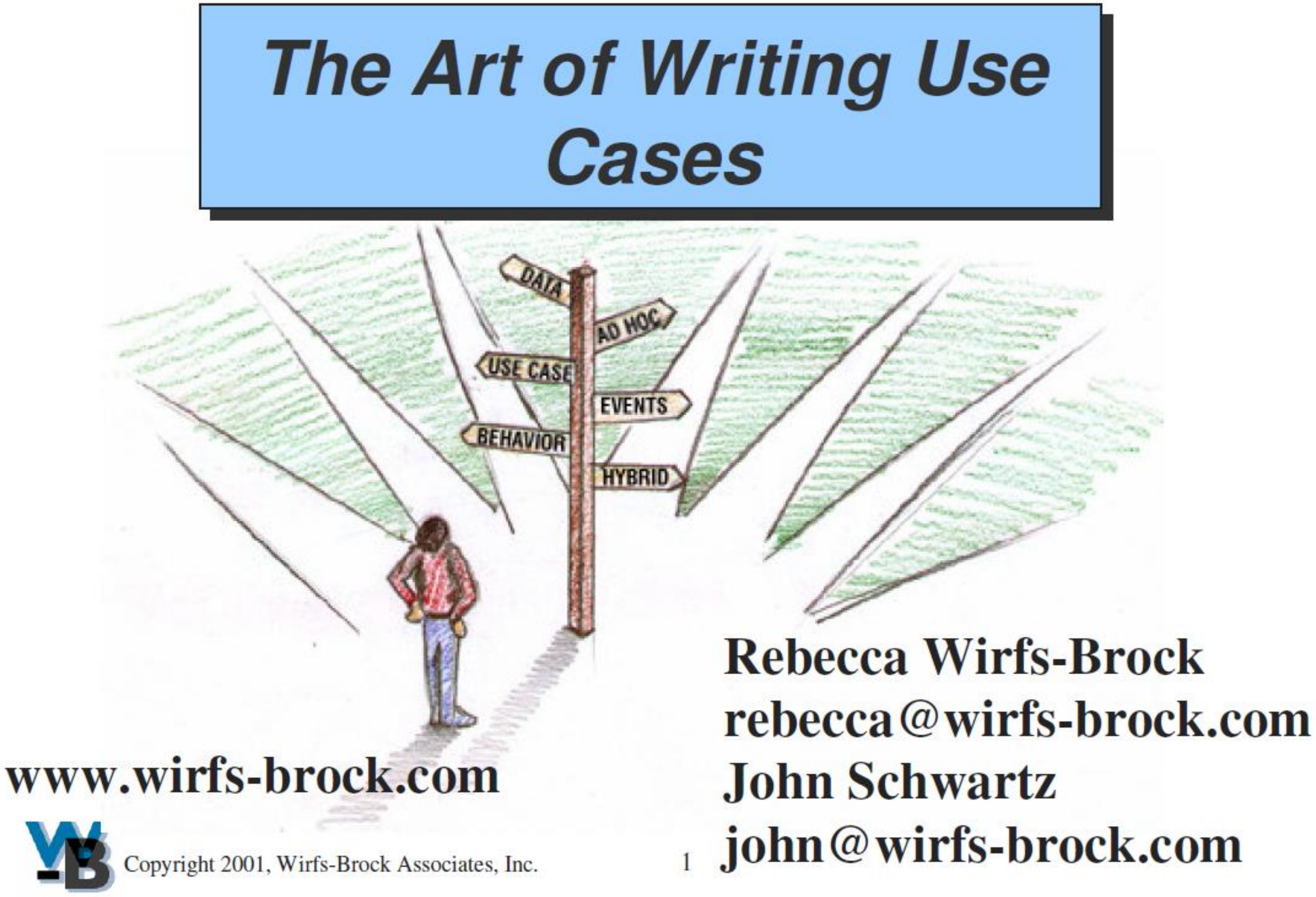
Deployment Unit



활동2. 기능 명세


CPU
Multi-core
Board
Storage
DRAM
Interrupt
Timer
USB
SPI
UART
DMA
File System
Config Load
Keyboard
Display
Kernel Load
Shell

http://www.wirfs-brock.com/PDFs/Art_of_Writing_Use_Cases.pdf



The Art of Writing Use Cases

DATA
AD HOC
USE CASE
EVENTS
BEHAVIOR
HYBRID

Rebecca Wirfs-Brock
rebecca@wirfs-brock.com
John Schwartz
john@wirfs-brock.com

www.wirfs-brock.com