Improvement of Start-up Time on Linux-Based IVI System

2016/7/13

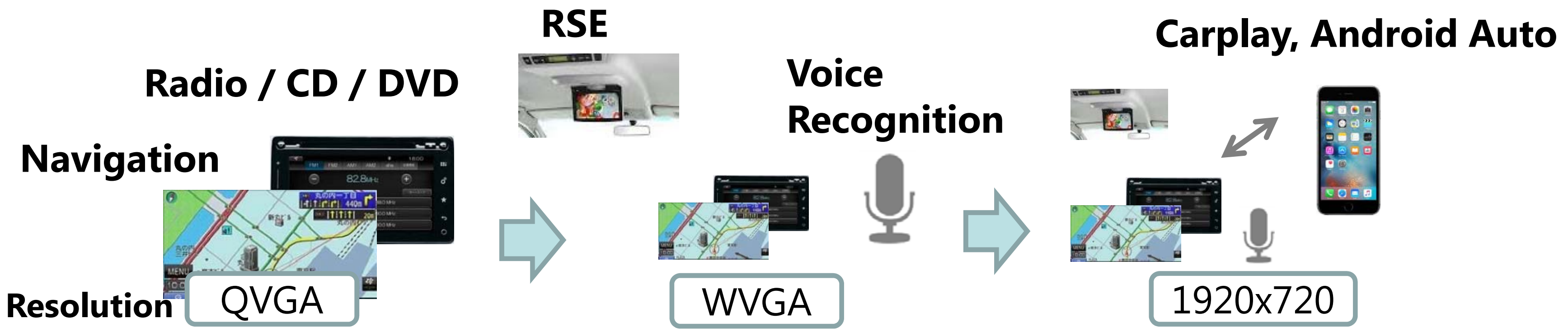
Panasonic Corporation

Kazuomi KATO

AGENDA

Background

Increasing the amount of IVI system software.



	1996	2006	2016
CPU	30MHz	400MHz	1.5GHz
DRAM	4MB	256MB	3GB

Year

- **Due to the functionality such as smart phone link function, more and more functions and large-scale software are being required.**
- **Data size is being increased due to the expansion of the display resolution and the realization of multi-screen being equipped.**
- **At the same time, start-up time and quick response of the system are required by the driver much more.**
- **Mobile devices equipping a lot of functions as much as IVI system can quickly start up, therefore quick start-up time of IVI system is required as well.**

Issue

Start-up time issues of IVI system:

1. **Realization of quick start-up time suitable for users who are used to smart phone.**

2. **In order to put the functions of music player, car navigation into operation as soon as the engine being started, high speed of start-up time of IVI system is necessary.**

3. **To improve the UX and display resolution, the amount size of data are increased. As a results, reading the data from storage costs much longer.**

4. **With the change of platform from RTOS to Linux, the start-up time is increased.**

5. **In the application of vehicle, although large battery is equipped, the issues of dark current has to be considered, when the car not being used for several weeks.**

Methods of optimizing startup time

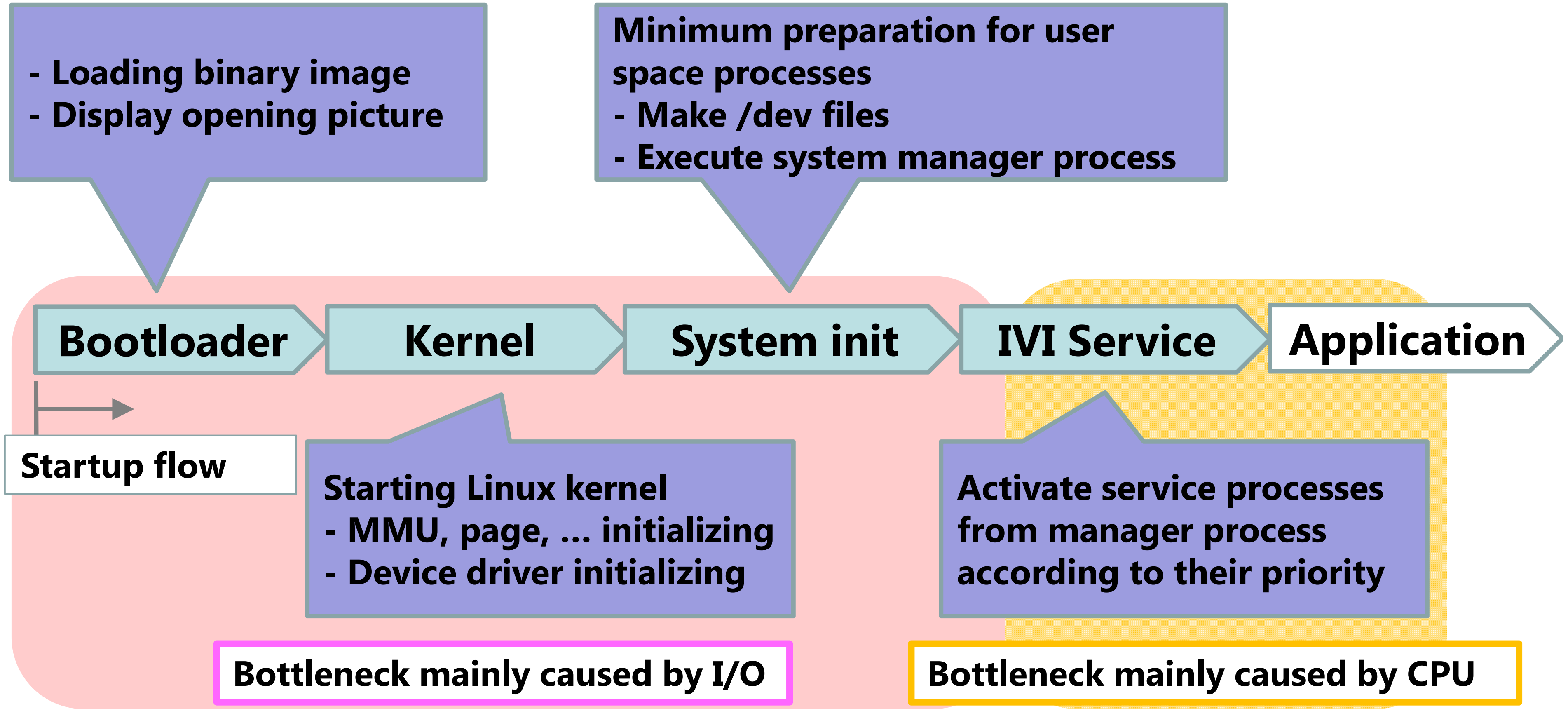
We adopted the approach to analyze and eliminate the bottleneck.

Approach	Typical method	Pros	Cons
1. Eliminate the bottleneck and improve processing	**Profiling and Analysis** Improving method based on the profiling and analysis.	Fundamentally improvable.	Know-how is required to specify the bottleneck and improve.
2. Startup is quickly shown as a user's view	**CAN wakeup** Method of starting the system in advance by CAN signal such as unlocking the door.	It seems to start up fast when getting into a car.	It depends on OEM requirements.
3. Fast return without completely stopping the system	**Suspend / Resume** Method of saving the current state to DRAM, turning peripheral device power off, waiting and restarting from the state saved at that time.	Fast resume is possible at turning power ON.	The electric power for maintaining DRAM data is consumed.
4. Carry out power OFF adding to No.3 approach	**Snapshot boot** Method of saving the kernel image of a certain point into the storage and booting up from that point when resuming the system.	Electric power is not consumed as compared with No.3.	If boot image size becomes large, the load time from storage will become a neck.

How to analyze

- Divide the system into several phase from power ON to start of the application.
- Since the reasons which cause the bottleneck in different phases are different, therefore, considering the approach to those respectively is necessary.

Overview of each phase during the system start-up

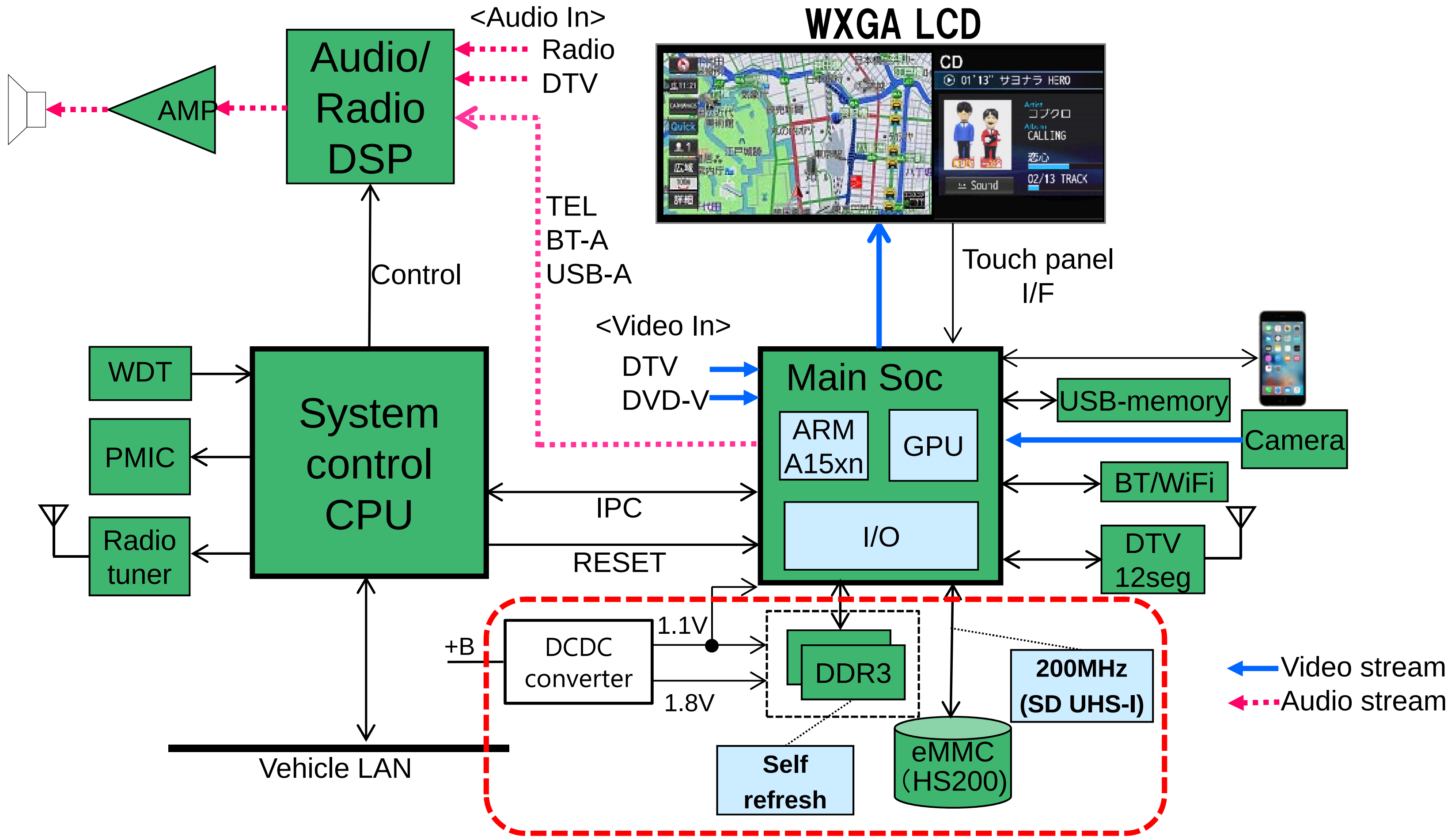


- Loading binary image
- Display opening picture
Minimum preparation for user space processes
- Make /dev files
- Execute system manager process
Bootloader
Kernel
System init
IVI Service
Application
Startup flow
Starting Linux kernel
- MMU, page, … initializing
- Device driver initializing
Activate service processes from manager process according to their priority
Bottleneck mainly caused by I/O
Bottleneck mainly caused by CPU

Hardware Block Diagram of IVI system

In order to realize the fast boot, use the following as hardware

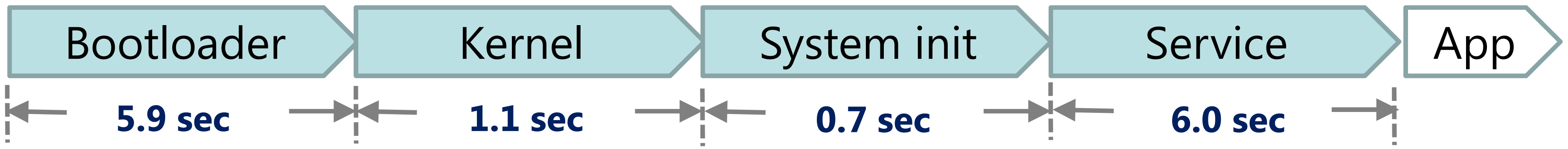
- **DRAM Self-refresh in part of DDR3**
- **Connect the high speed eMMC to the SD UHS-I(200MHz) port.**
  （because of SoC restriction, eMMC I/F speed is less than device speed ）



Analysis of start-up time in each phase

Consider the solution of specific bottleneck based on analysis.

Start-up time in each phase at the beginning (@Cortex A15 1.5GHz x2 SoC)



Issue of start-up time in each phase

1. **Bootloader**
   - **Program loading time from eMMC device occupies.**
2. **Kernel**
   - **Waiting for device driver initialization occurs.**
3. **System init**
   - **The CPU usage rate has only been about average 60%.**
4. **Service**
   1. **Waiting for the process of mounting file system occurs.**
   2. **Low speed of loading from storage device (eMMC) when starting service.**
   3. **CPU load increases due to the interruption of eMMC when system started up (2-6secs).**

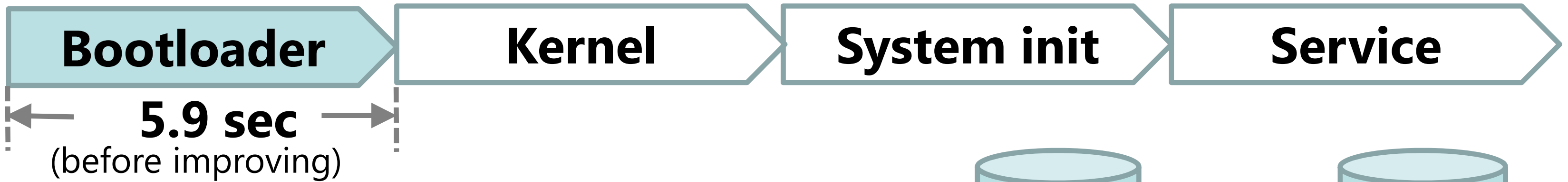
Analyze the method to solve the bottleneck caused by I/O.

Analyze the method to solve the bottleneck caused by CPU.

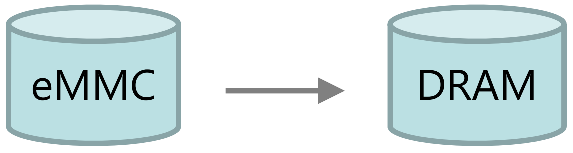
Panasonic

Improvements for bootloader(1)

The bottleneck is loading time from eMMC to DRAM.



Processing	Time
eMMC -> DRAM transferring procedure (100Mbyte)	5,000msec
eMMC initilizing procedure	200msec
Debug log output	200msec
ROM integrity checking	65msec
Primary loader process	40msec
…	

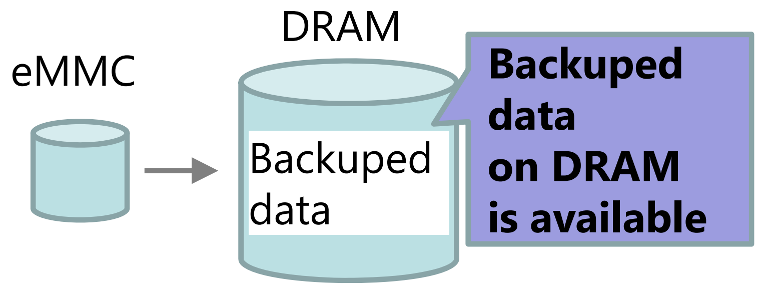


Before	20MB/sec
After	80MB/sec

Connecting eMMC to SD UHS-1 line

However eMMC bottleneck remains.

For more improvement



Improvements for bootloader (2)

Partial DRAM backup system is necessary.

DRAM backup

- **The backup data on DRAM is reused when turning ignition ON.**

Issue:

- **Not allowed to back up whole DRAM area because of the large dark current.**

Approach:

1. Scope for backup is limited to program code.
2. Program for backup is gathered to the specific memory area.
3. Only that area is the target area for DRAM backup.

What program should be backuped?

- **Root file system: Executable binaries required on start-up are chosen.**

Improvements for bootloader (3)

File system available for DRAM backup system should be chosen.

Requirements of DRAM backup for root file system

1. **It is operable on RAM.**
2. **It is not lost by reboot of Linux.**
3. **No matter happens by reusing backup image of initial state.**
4. **Not allowed to consume extra RAM.**
5. **As a security requirement, it is available for dividing the privileges associated with super user into distinct units.**

The tmpfs, romfs and cramfs as a candidate don't meet those requirements.

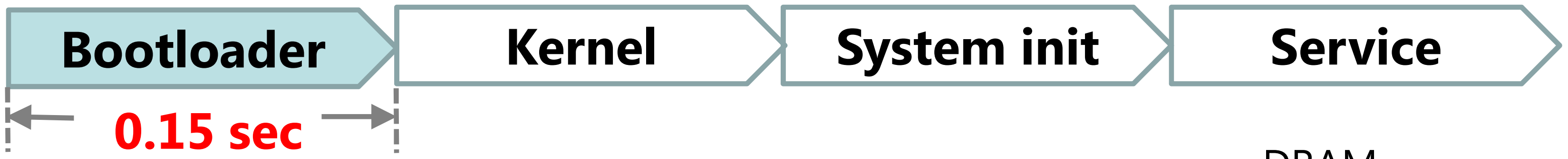
We adopted PRAMFS (Protected and Persistent RAM Filesystem)

DAX (for direct access) system was experimental and not adopted at that time.

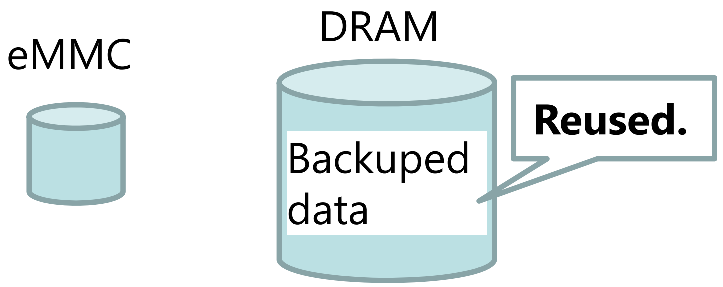
Panasonic

Improvements for bootloader (4)

Result: The startup time of bootloader was improved from 5.9 sec to 0.15 sec by using DRAM backup system.



Processing	**Time**
~~eMMC -> DRAM transferring~~ procedure (100Mbyte)	~~5,000msec~~
~~eMMC initilizing procedure~~	~~200msec~~
~~Debug log output~~	~~200msec~~
ROM integrity checking	65msec
Primary loader process	40msec
…	

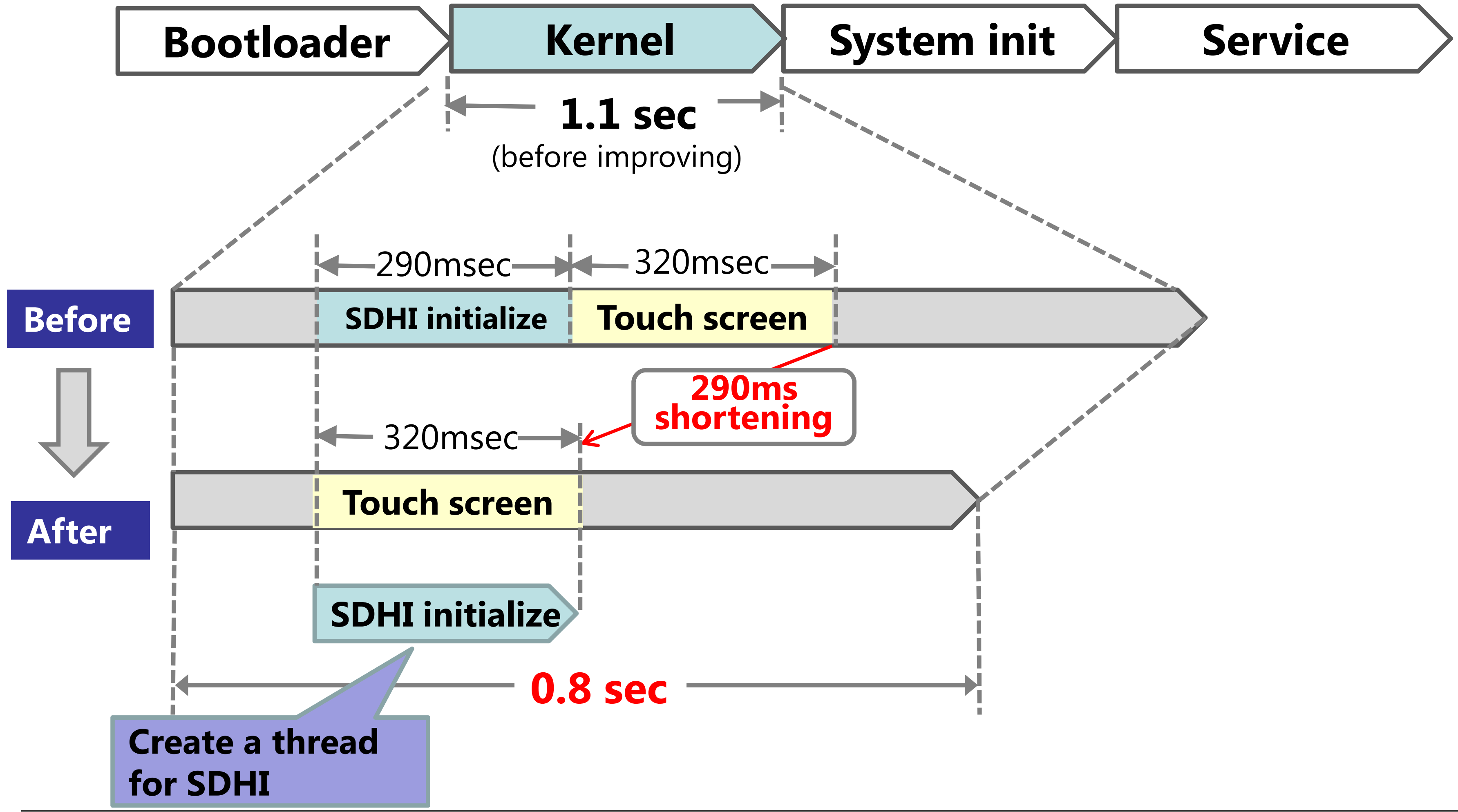


eMMC acesses was eliminated.

The log output process for debug is also deleted.

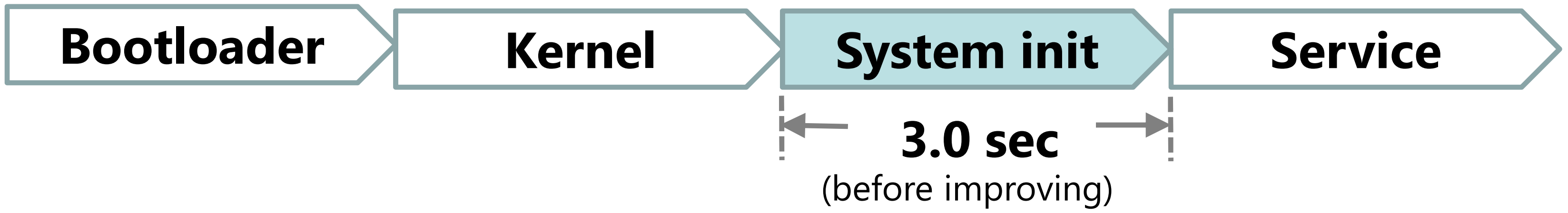
Improvement for kernel

Result:
The startup time of kernel was improved from 1.1 sec to 0.8 sec by parallel execution for device driver initialization.



Improvements for system init (1)

We developed original simple init called “system init”.
(Not System V init)



Originally developed “system init” for running service process and making device environment for user land.

- **Making device files and symbolic links**
- **Mount filesystem**
- **Setting resource parameter**
- **Setting network parameter**
- **Activating a service process**

Why originally developed?

- **Need the minimum init functionality.**
- **Standard init process like systemd is not necessary for our product requirements.**

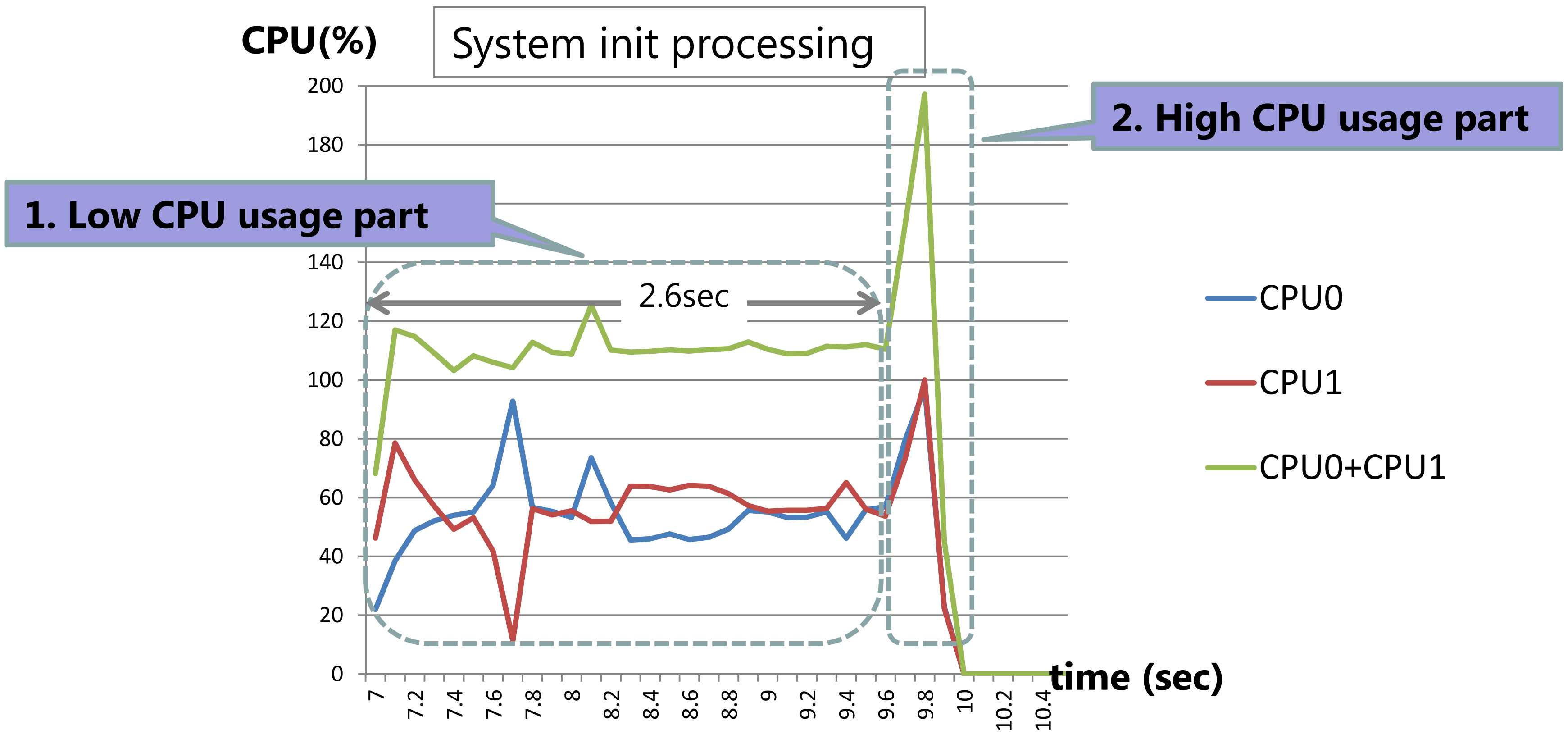
Improvements for system init (2)

1. Low CPU usage part

- **1-1. Modified code from shell script to native C code ⇨ -2.3 sec**
- **1-2. Changed serial execution to parallel execution ⇨ -0.1sec**

2. High CPU usage part ⇨ -0.3 sec

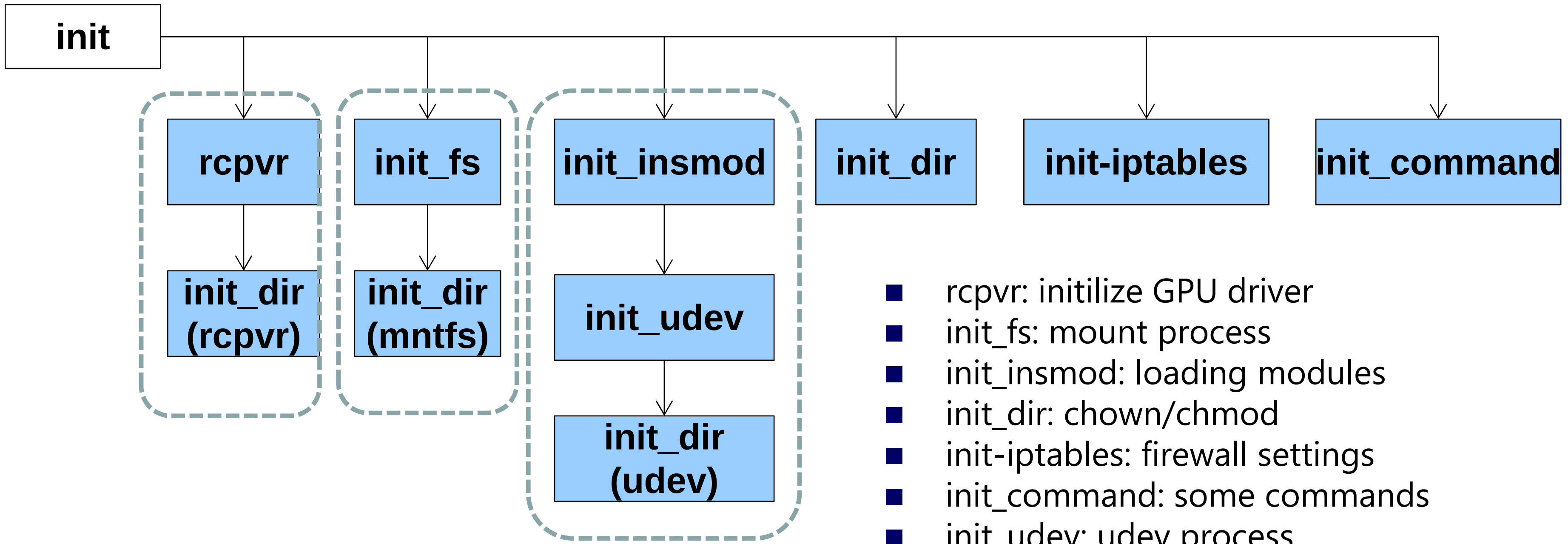
- **Optimized the udev event control**



Improvements for system init (3)

1.2 Parallel execution ⇨ <u>-0.13 sec</u>

Process groups based on each dependencies are parallel executed as the threads.



- rcpvr: initilize GPU driver
- init_fs: mount process
- init_insmod: loading modules
- init_dir: chown/chmod
- init-iptables: firewall settings
- init_command: some commands
- init_udev: udev process

Improvements for system init (4)

2. Optimize the udev event control ⇨ -0.3 msec

Devices to be excepted from “udev” should be clarified and processing of "udev" occupied in start-up should be optimized.

Issue:

1. **Thread processing for handling an udev event spends time from 20 msec to 40 msec. About 400 msec occurs.**

2. **However, the udev operation is necessary for plug and play device in our system.**

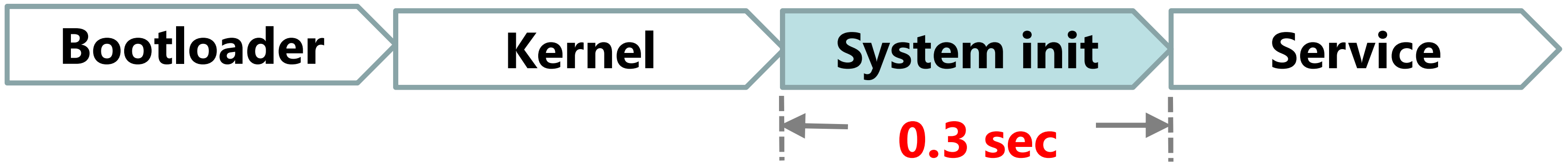
Approach:

1. **The udev which can specify the exceptional events such as the system fixed devices is originally developed.**

2. **The original udev has only the ability to register the event for controlling.**

Improvements for system init (5)

Result :
The startup time of "system init" was improved from 3.0 sec to 0.3 sec



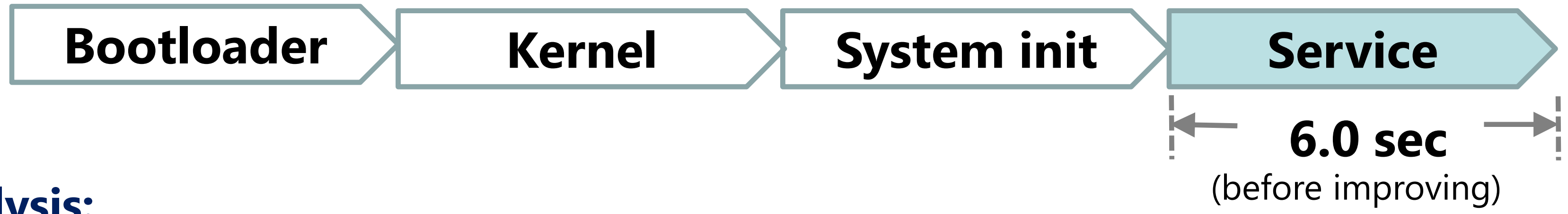
Improved "system init" phase through the approach as bellow

1. **Change to the native C code**
2. **Parallel execution**
3. **Optimization for udev event control**

Improvement for service processing (1)

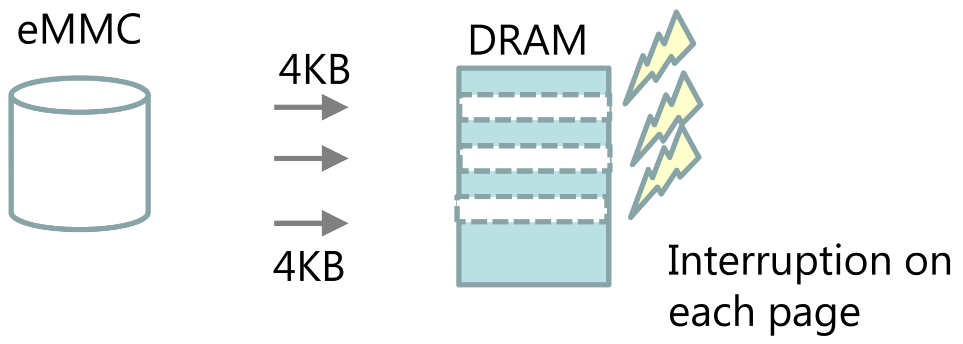
Issue:

CPU load increases when accessing to eMMC during activating service processes.



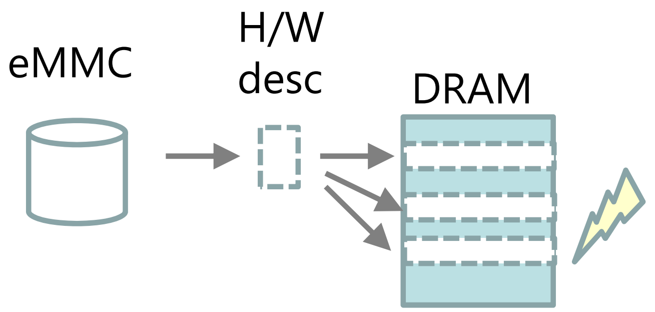
Analysis:

1. **eMMC access after Linux starting is almost loading to not contiguous DRAM area.**
2. **It was found that since DMA transfer completion interruption was occurring frequently, CPU load is increasing.**



Improvement for service processing (2)

Approach:
The DMA for eMMC to non-contiguous areas can be transferred by using the function of the hardware as if they were a contiguous area.



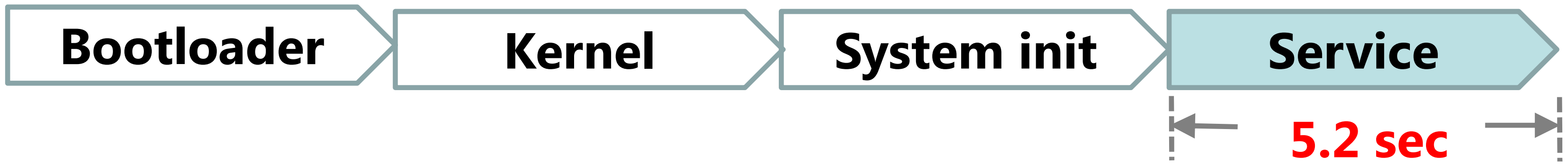
Interruption on each transfer(ex 4KB)

After improvement

Item			before	after
DRAM		Descriptor	off	on
		BounceBuf	Used	Unused
	CPU Usage	mmcqd DMA driver	26.6 %	9.0 %
	Interruptions (times/sec)	SDHI DMAC	11,029	2,221
	Read (MB/sec)		57.1 MB/s	71.5 MB/s

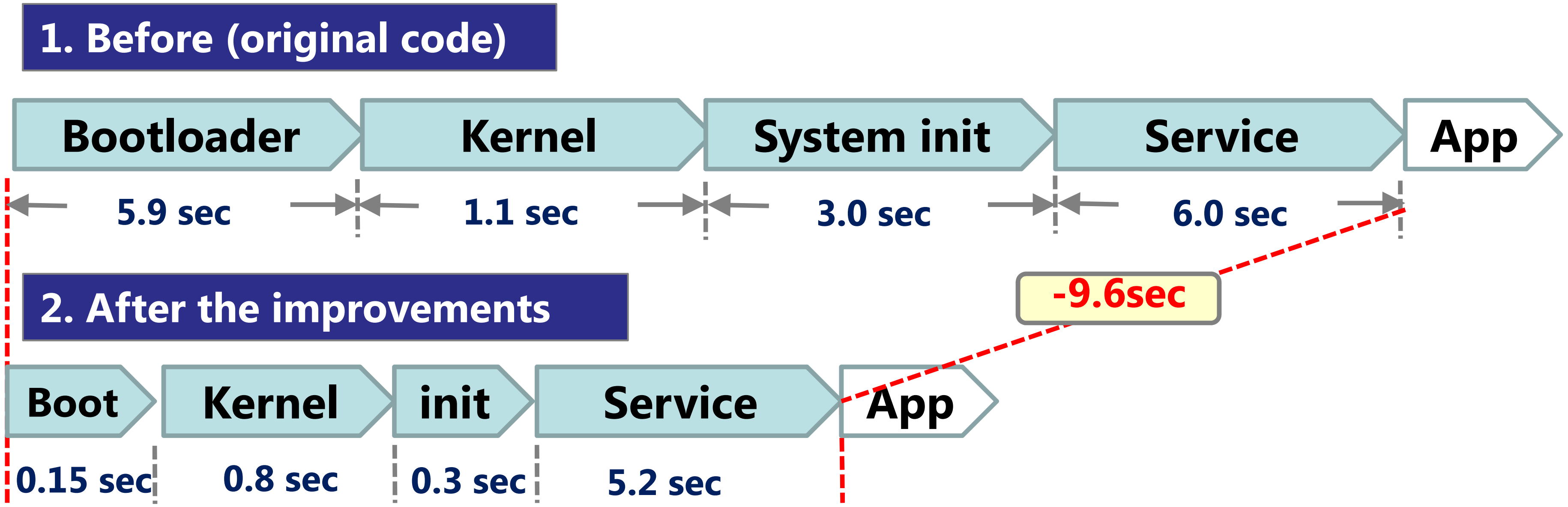
Improvement for service processing (cont.)

As a result of the CPU load decrease of eMMC access, it was shortened from 6.0 sec to 5.2 sec.



Summary

1. We have achieved 9.6 seconds reduce for the startup time.
2. Service phase still now occupies 5.2 seconds during startup time.
3. Improvement for each service is still on the way, and other system boot optimization method will be required to meet user's requests.



Future plan

1. The methods, which are independent of tuning application, as bellow need to be considered to the next IVI system.

- CAN Wakeup : Startup is quickly shown as a user's view
- Suspend and Resume :
Since the performance of DRAM and the dark current will be improved, the availability of backup of the whole DRAM increases.
- Snapshot Boot :
Because of the speedup of eMMC loading, cold boot from snapshot will become quick.

Device trend for IVI system

	Current	**Next Generation**	
DRAM I/F	DDR3 1.6GHz	LP-DDR4 3.2GHz	**x2**
Dark current (12V) 4GB	18.0 mA	2.2mA	**1/8**
eMMC read perf.	80MB/s	300MB/s (expected)	**x3.75**

Thank you!