Instant startup for application using reduced relocation time and rearranged functions

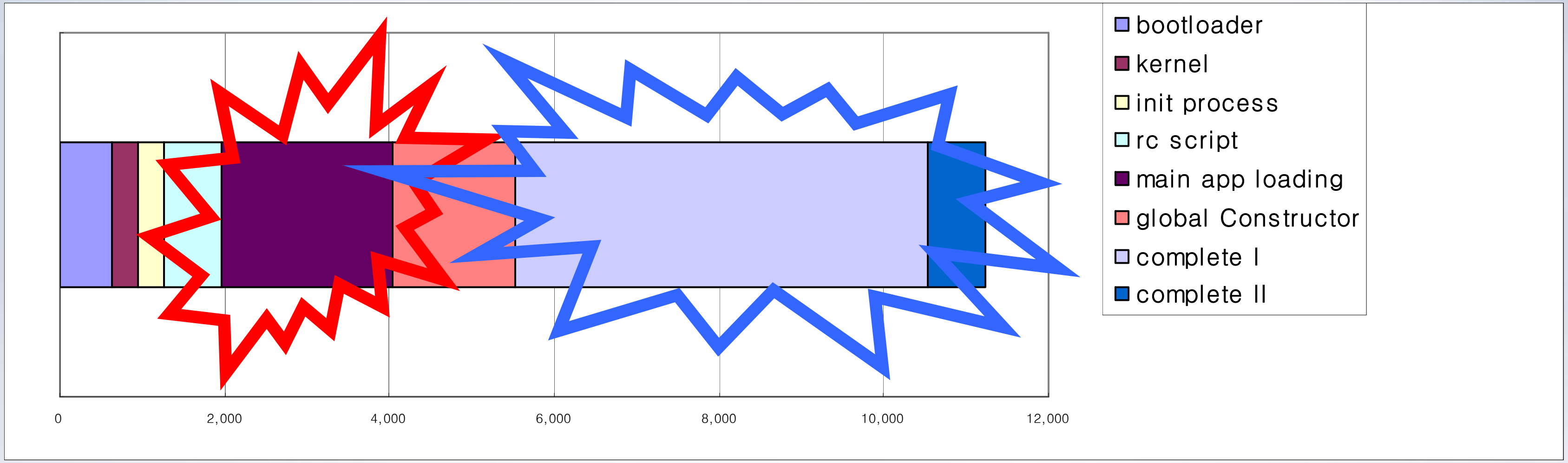
April 15, 2008

Samsung Electronics S/W Lab
Minchan Kim<mc4u.kim@samsung.com>
Oleksiy Kokachev<o.kokachev@samsung.com>

Contents

Bootup latency

- The bootup time is always a hot issue in embedded world
- We had applied many well-known kernel speedup techniques
  - Ex) Disable Console, Preset LPZ, Kernel XIP, …



Relocation time and excessive number of page faults are the key factors affecting application startup time

Relocation Time - DDLink

Page Fault - Functions Reordering

Application execution procedure

- do_execve kernel function overview
  - Determines the type of executable (static or shared) by ELF parsing
  - In case of shared executable - transfers control to dynamic loader
- Dynamic loader overview
  - Loads executable program segments
  - Loads all libraries needed to execute the program
  - Relocates relocate entry of all shared library
  - Calls init function of shared libraries
  - Transfers control to libc

Why do we use shared libraries ?

SAMSUNG

- Advantages
  - The executable is smaller (it does not include the library information explicitly),
  - When the library is changed, the code that references does not usually need to be recompiled.
  - The executable accesses shared library at run time; therefore, multiple applications can access the same shared library at the same time (saves memory)

We don’t have whole library sources

- Disadvantages
  - Need to load shared libraries
  - Need to resolve addresses
  - The possible lack of dynamic library
  - The possibility of library version mismatch

DDLink (Deterministic Dynamic Linker)

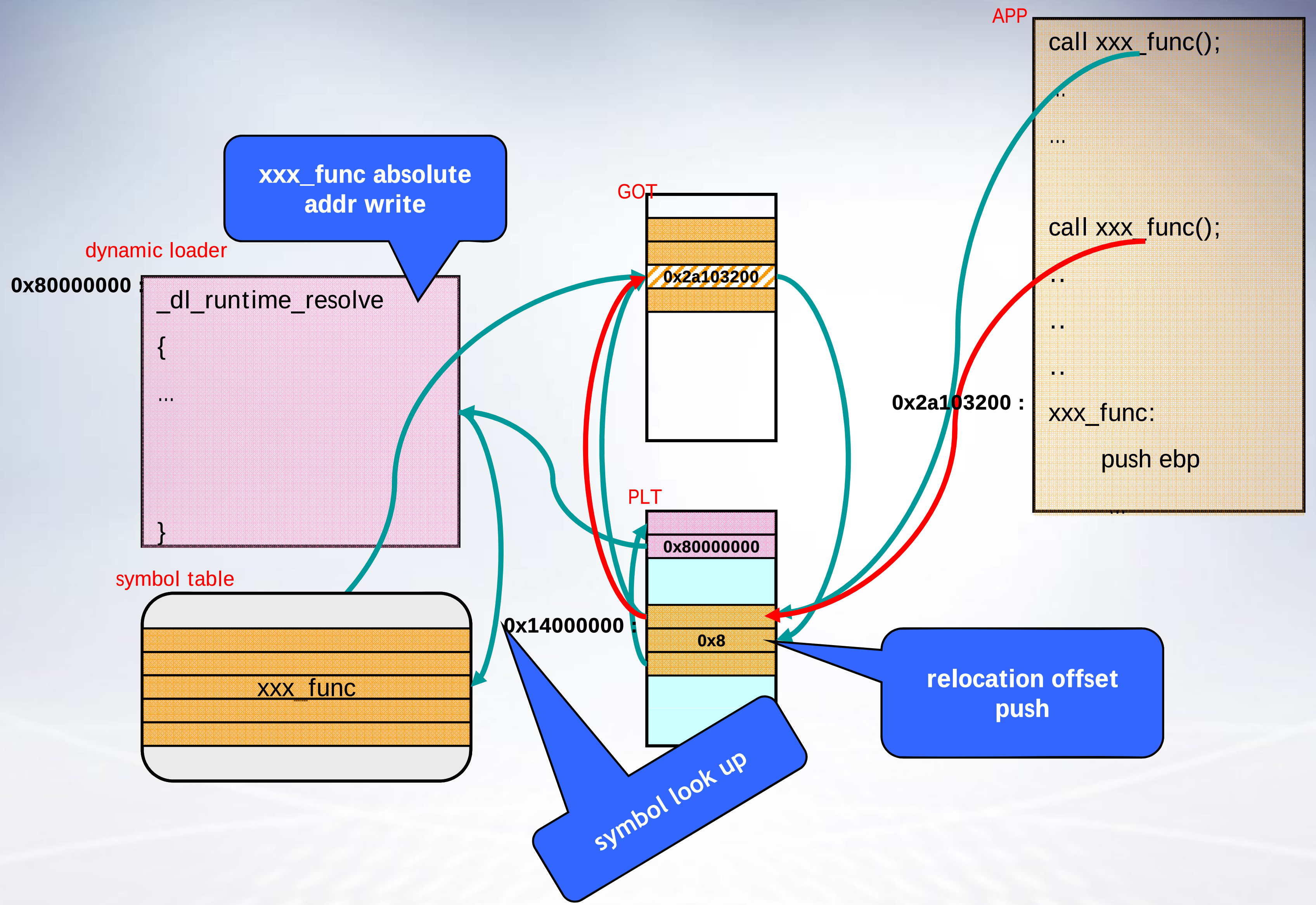
SAMSUNG

- Goal
  - To reduce relocation processing time
- Approach
  - Many embedded systems have the same startup sequence:
    - Hardware reset + bootloader + kernel execution + root file system mount + init + app execution
  - So, we can find the shared library mapping base address, thus we can avoid resolving address

What is GOT and PLT in ELF ?

- GOT and PLT are key for DDLink
- GOT and PLT are parts of ELF
- PLT(Procedure Linkage Table)
  - Redirects position-independent function calls to absolute locations
- GOT(Global Offset Table)
  - Redirects position-independent address calculations to absolute locations

Detail of Lazy Binding



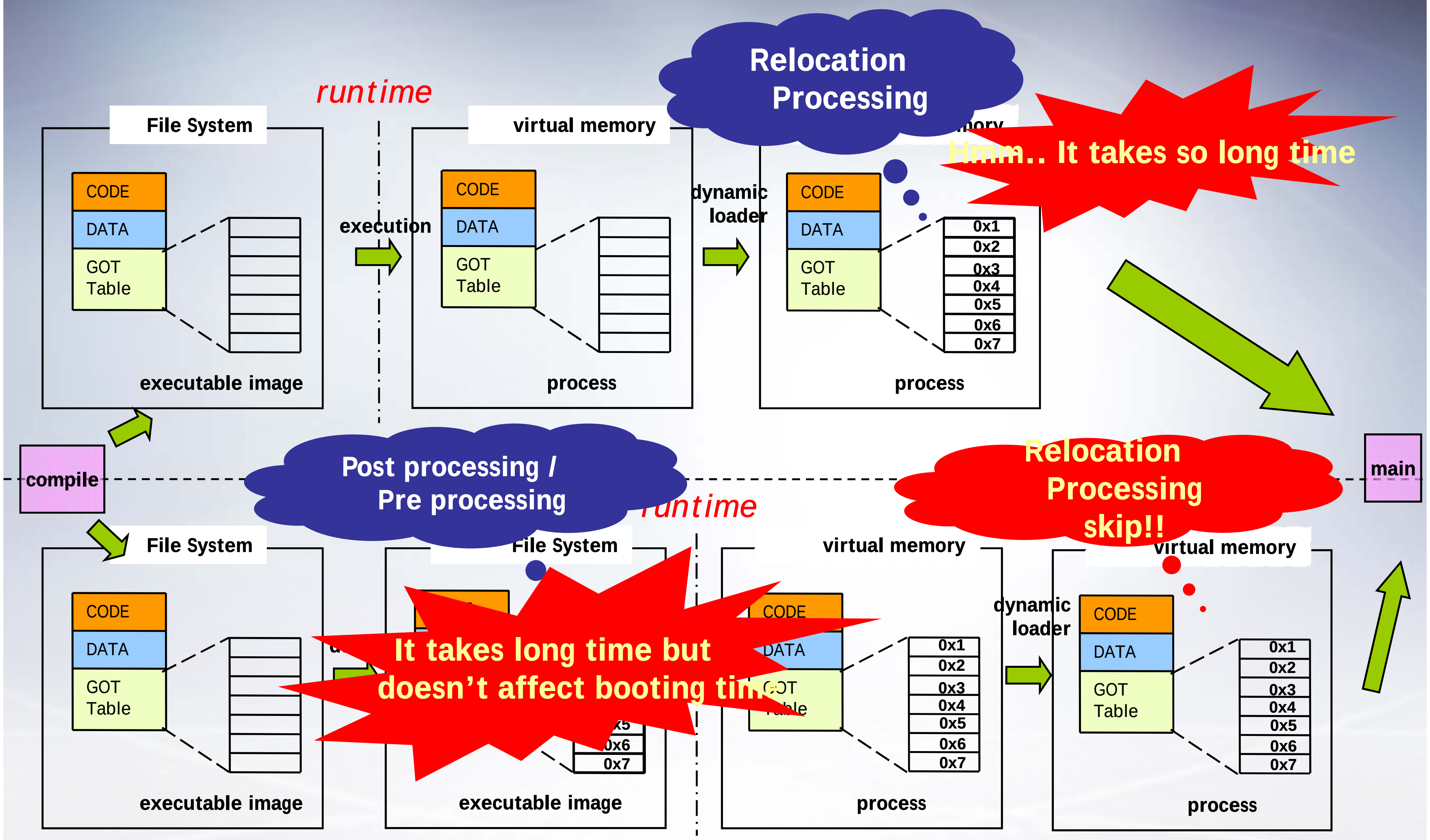
DDLink Procedure

SAMSUNG

- Phase 1 – Preprocessing execution program to get runtime profiling data
  - Mark shared libraries to present that this library will be used by DDLink
  - Execute target program without lazy binding
  - Remains profiling data which include GOT entries resolved by our dynamic loader

- Phase 2 – Postprocessing static binary images with profiling data
  - Write address resolved to execution images's GOT section

DDLink Procedure(cont.)



DDLink supports two modes

- Libraries are used by many program concurrently
  - It can change library's base address in each process address space
- File Read Mode
  - Resolved address is recorded to profile data file
  - Just read profile data when dynamic loader is processing relocation but avoiding relocation which consume much time
  - Generally, this mode is applied to libraries which are shared among many processes
  - Overhead of file I/O
- Relocation Skip Mode
  - Resolved address is recorded to images
  - Dynamic loader never do any operations related to relocation
  - Generally, this mode is applied to proprietary libraries of the application
  - Very fast

DDLink Package

- Glibc dynamic loader
  - Handles relocation according to DDLink mode properly
- DDLink utilities
  - DDLink mode marker
    - File Read Mode
    - Relocation Skip Mode
  - DDLink GOT writer
    - Record GOT table with relocation result which are obtained with profiling

DDLink Issue

SAMSUNG

DDLink performance factor

- Depends on filesystem read overhead
- Compression ratio (In case of Compression File System)
- Flash speed
- Number of symbols

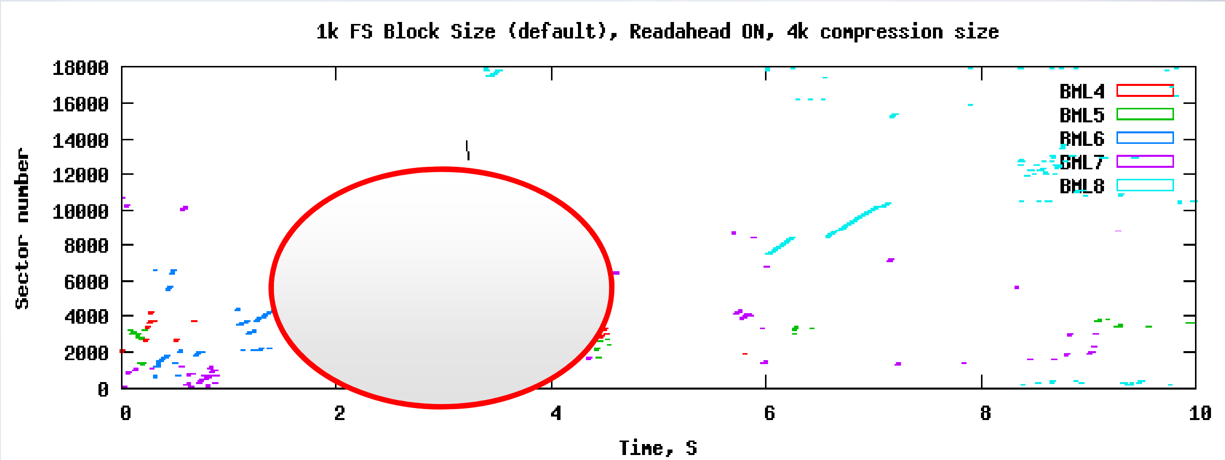
DDLink Issue(cont.)

- Read Ahead
  - Optimization method based on working set locality
  - Read/Write **VS** Disk Seek (Cylinder, Head)
    - Hard disk seek time is very slow
- Most Embedded Devices use the flash memory
  - Flash memory seek time is likely to constant
- Nowadays, ReadAhead have become obstacle against startup of application in Embedded System

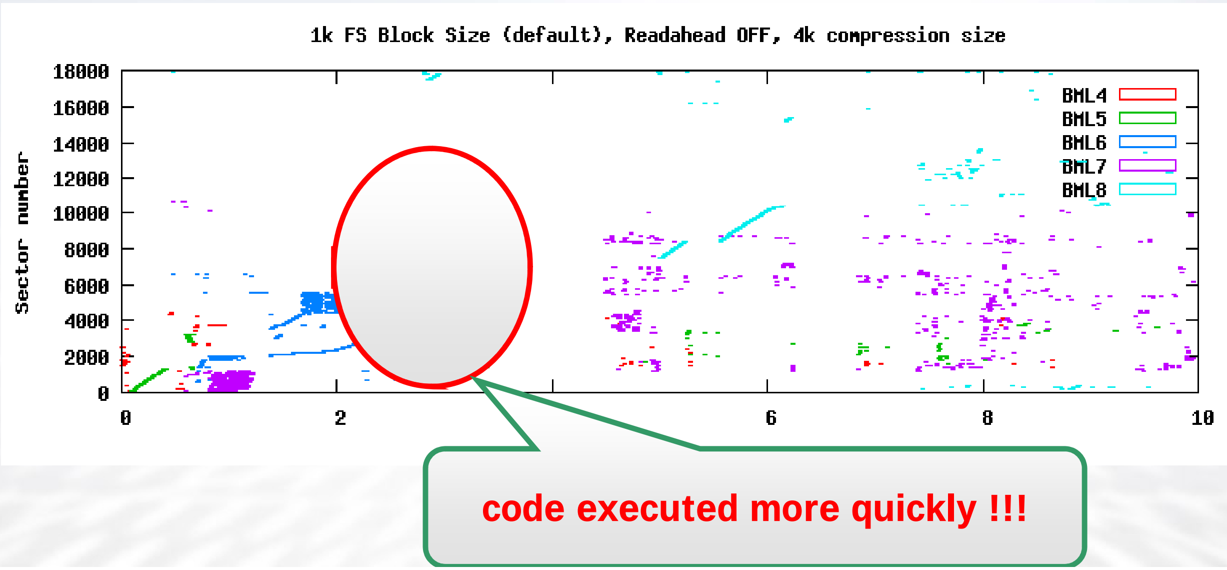
ReadAhead Issue(cont.)

ReadAhead Optimization

* Readahead ON

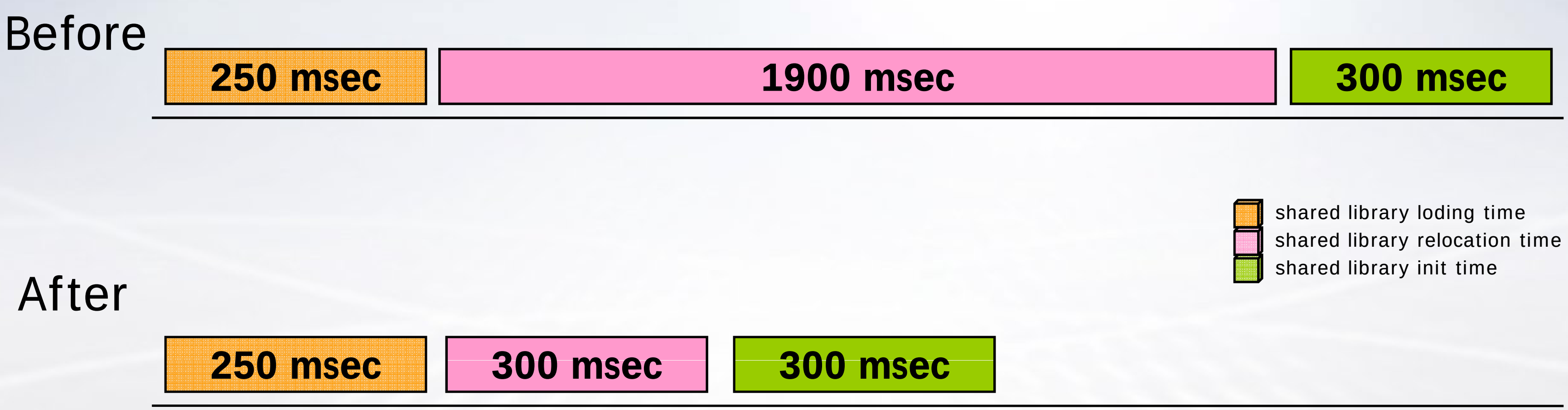


* Readahead OFF



Real-Target measurement

- Environment
  - MIPS(500MHz), DRAM(128M), Flash (64M)
- DDLink configuration
  - Glibc Libraries : File Read Mode
  - Proprietary Libraries : Relocation Skip Mode
- The number of shared libraries : 12
- The number of symbols : around 200,000
- ReadAhead Off



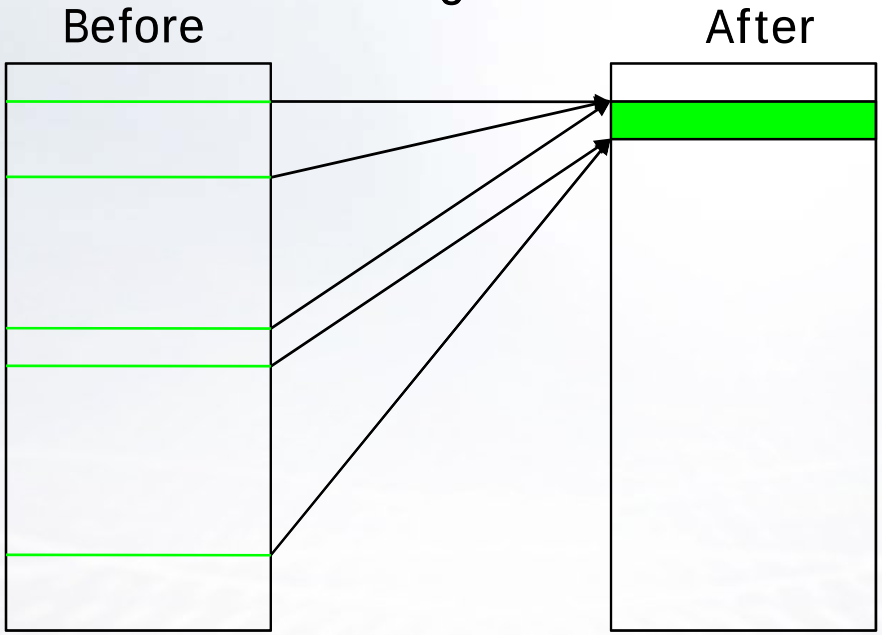
Functions reordering

Problem definition

- In large applications, consisting of big number of object files, functions used during system startup are spread over the application's binary
- During execution application file is mapped to memory and all reads requests from it are performed on a page size boundary
- To load one small function (i.e. 100 bytes in size) which are not currently in page cache, one *page_fault* should occur and whole page (4k in size) containing this function should be loaded into RAM from flash storage device
- Nowadays, flash memory is relatively slow, so reducing the amount of data loaded could reduce booting time

Functions reordering (cont.)

- **Proposed solution**
  - The main idea is to place all the functions used during system startup altogether, one-by-one in their execution order
- **Main benefits**
  - Reduced amount of data read from slow flash storage device
  - Reduced number of page_faults
  - As a result – reduced booting time



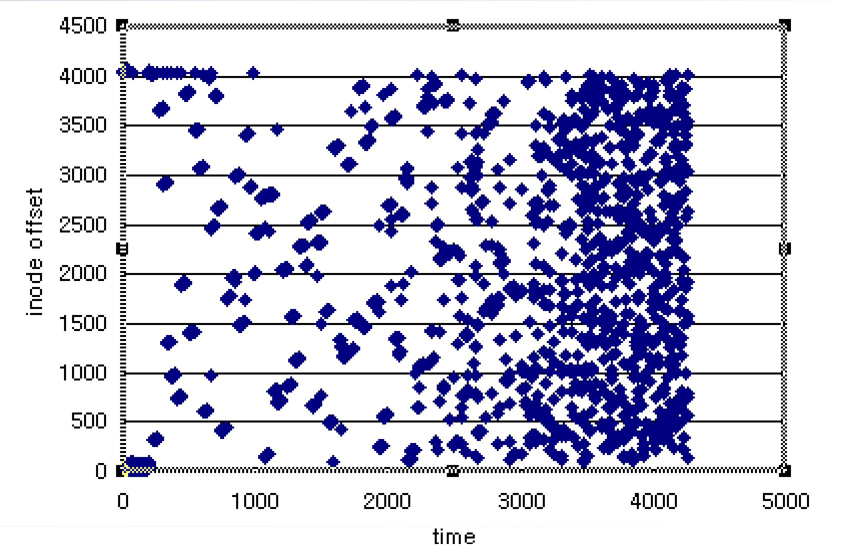
Functions reordering (cont.)

- **Step-by-step description**
  - **Step 1**: Compile every source file with instrumentation support (-finstrument-function GCC option) and link the final binary with libpcprofile.so shared library
  - **Step 2**: Execute application on target and gather the instrumentation data
  - **Step 3**: Process instrumentation data:
    - Remove duplicate function calls
    - Transform function addresses to function names
    - Generate linker script
  - **Step 4**: Finally recompile every source file with -ffunction-sections GCC option and link the final binary in accordance with linker script

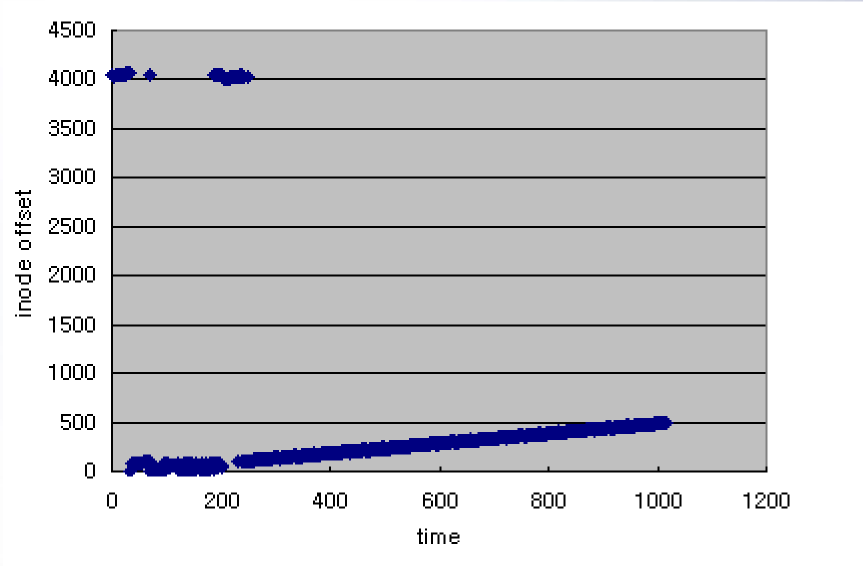
Functions reordering (cont.)

- Test program
  - Calls of different 1000 functions among 10,000 functions
  - Function size is very small
    - It mean there are many functions in one page
- Booting time reduction
  - Around 500msec

Before : 657msec



After : 157msec



Functions reordering (cont.)

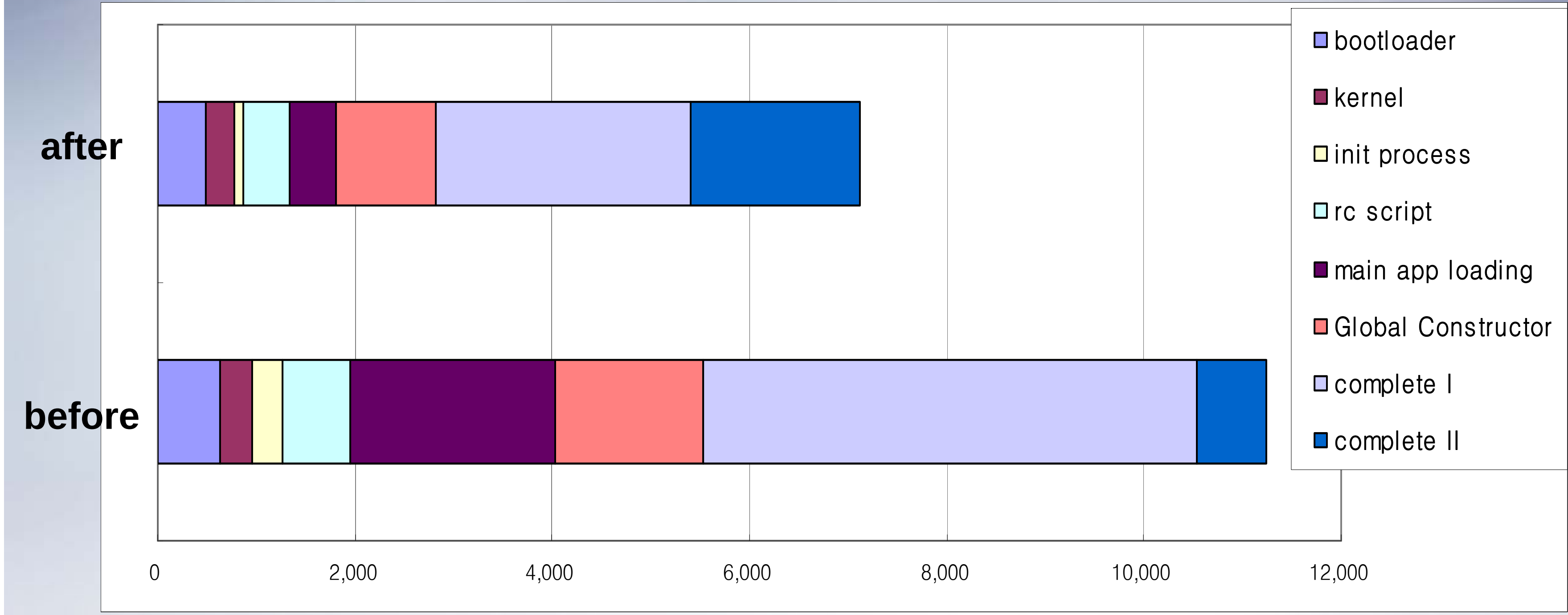
Real-Target results

- The exact time reduction depends on the system, CPU architecture, filesystem, application size and other factors
- Test system description:
  - 500 Mhz MIPS CPU
  - SquashFS filesystem (64k block size)
  - Kernel: 2.6.18

Booting time data:

- Origin : 8.214 sec
- Optimized : 7.496 sec
- Benefit : 0.718 sec

Summary


after
before
0
2,000
4,000
6,000
8,000
10,000
12,000
bootloader
kernel
init process
rc script
main app loading
Global Constructor
complete I
complete II

Summary (cont.)

DDLink

- Advantages
  - DDLink is a simple solution to reduce relocation time
  - Image size is never increased
- Disadvantages
  - Has inconvenient profiling phase
  - Doesn't support general case
  - Doesn't support graceful exit

Functions Reordering

- Advantages
  - Reduced amount of data read from slow flash storage device
  - Reduced number of page faults
- Disadvantages
  - A little complicated profiling and data analysis step

The End

- Questions??

Thank You.
Q&A