Panasonic
ideas for life

Making Mobile Phone with CE Linux

June 13, 2005

Presented by mizuyama.masashige@jp.panasonic.com

Panasonic Mobile Communications Co., Ltd.
Matsushita Electric Industrial Co., Ltd.

Mobile Phone Product Based on CE Linux

Panasonic has shipped mobile phone product based on CE Linux for the mainstream market since Feb. 2005.

- CPU: ARM9
- Base OS : MontaVista Linux CEE 3.1 (2.4.20 kernel, glibc-2.3)
- GUI: X Window, GTK+(1.2.10) / Motif like toolkit

Kernel 2.4.x will be used for several more years.

- product quality assurance, migration cost

Technical Challenges

- Foot Print Size
  - no swap devices, strong cost pressure, feature richness.
  - large footprint size of the OS (RTOS under 200kB, Linux over 2MB)
- Stability
  - no Freeze, no crash…
- Boot Time / UI Response Time
  - quick response expected by customers.
- Realtime
  - video phone, device control…
- Power Consumption
  - long lasting, but small battery required

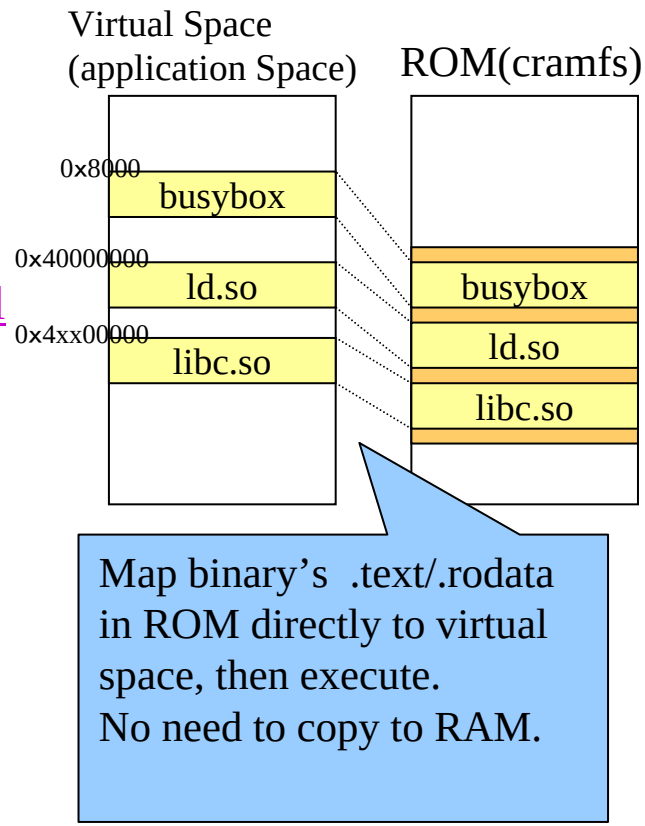
Reduction of the Foot Print Size

	ROM	RAM
“Allocate On Write” for .data Sections*		✔
XIP (eXecute In Place)		✔
Thumb® Instruction Set (Interworking)*	✔	(✔)

* Development efforts done by Panasonic

Application XIP

- Application XIP
  - Map Code Pages on NOR CRAMFS to Virtual Address Space Directly.
    http://tree.celinuxforum.org/CelfPubWiki/ApplicationXIP
  - ROM increase (compression is omitted)/Execution speed
- Result
  - Totally 45% reduction of page cache allocated to processes at stand by after boot up
  - Code ROM increased x 2.



“Allocate On Write” for .data Sections (1/7)

Reduction of unwritten page cache for .data

- Observation:

  Some pages in .data sections remain unwritten throughout process lifetime (only read).

- Original OS Behavior:

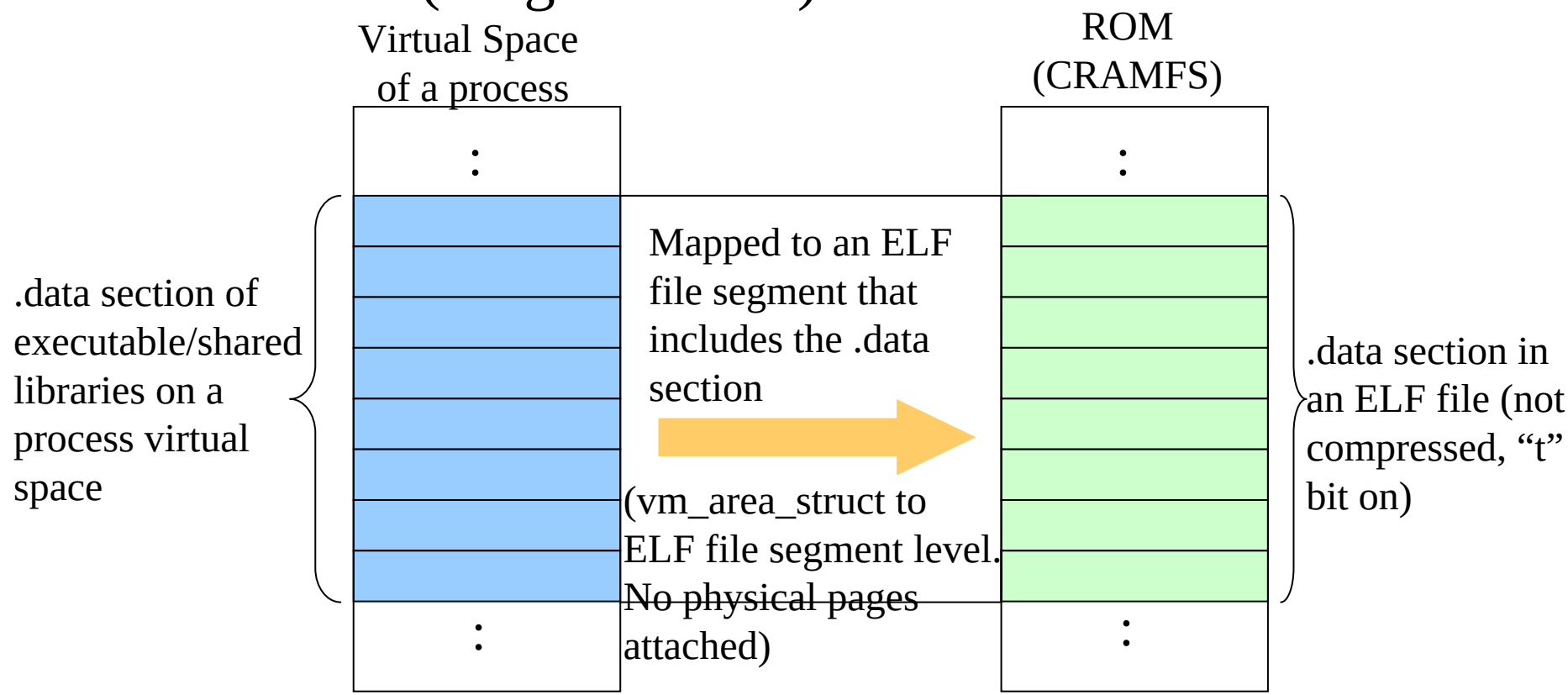
  allocates RAM pages (page cache) to a process whenever it accesses pages in .data sections.

- Our modification:

  defer RAM page allocation for .data until the process writes to the page (exploit XIP and COW mechanism).

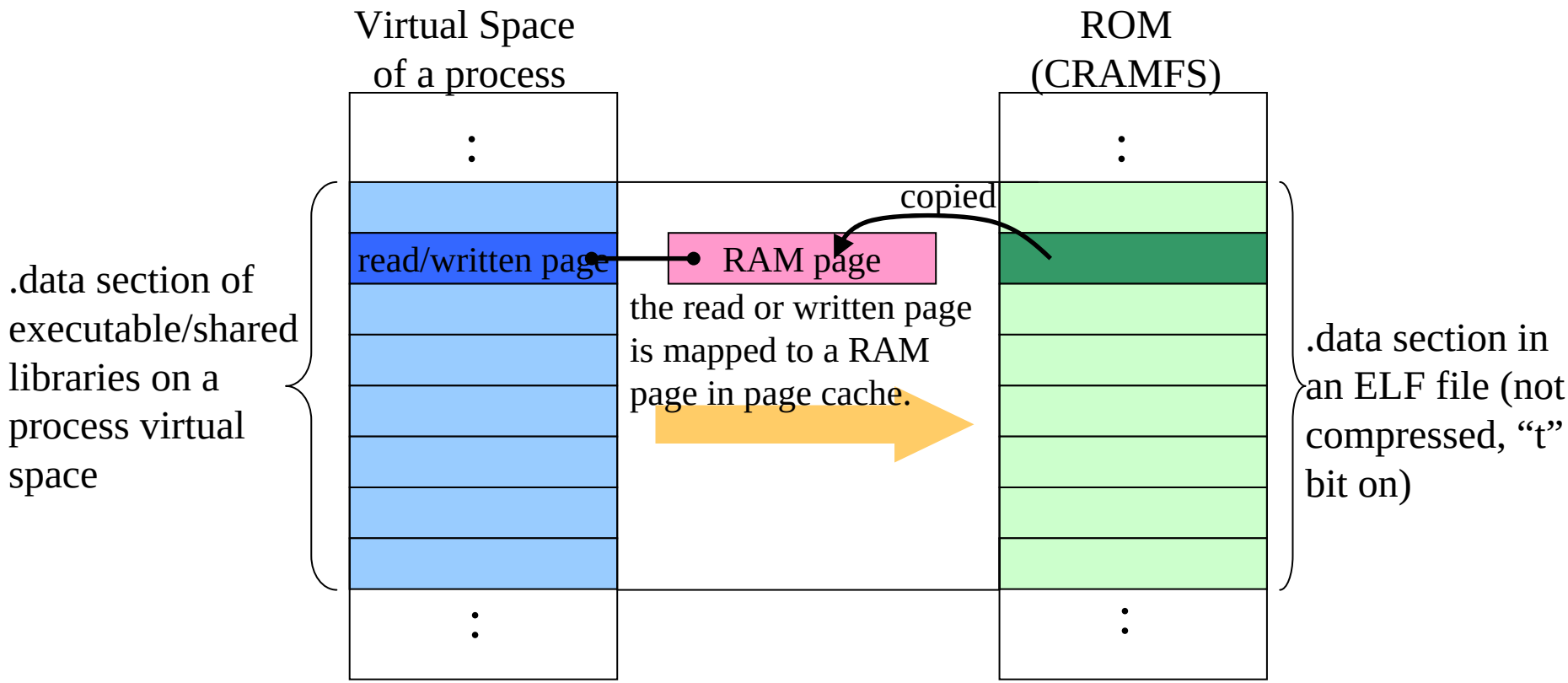
"Allocate On Write" for .data Sections (2/7)

Initial State (Original Code)



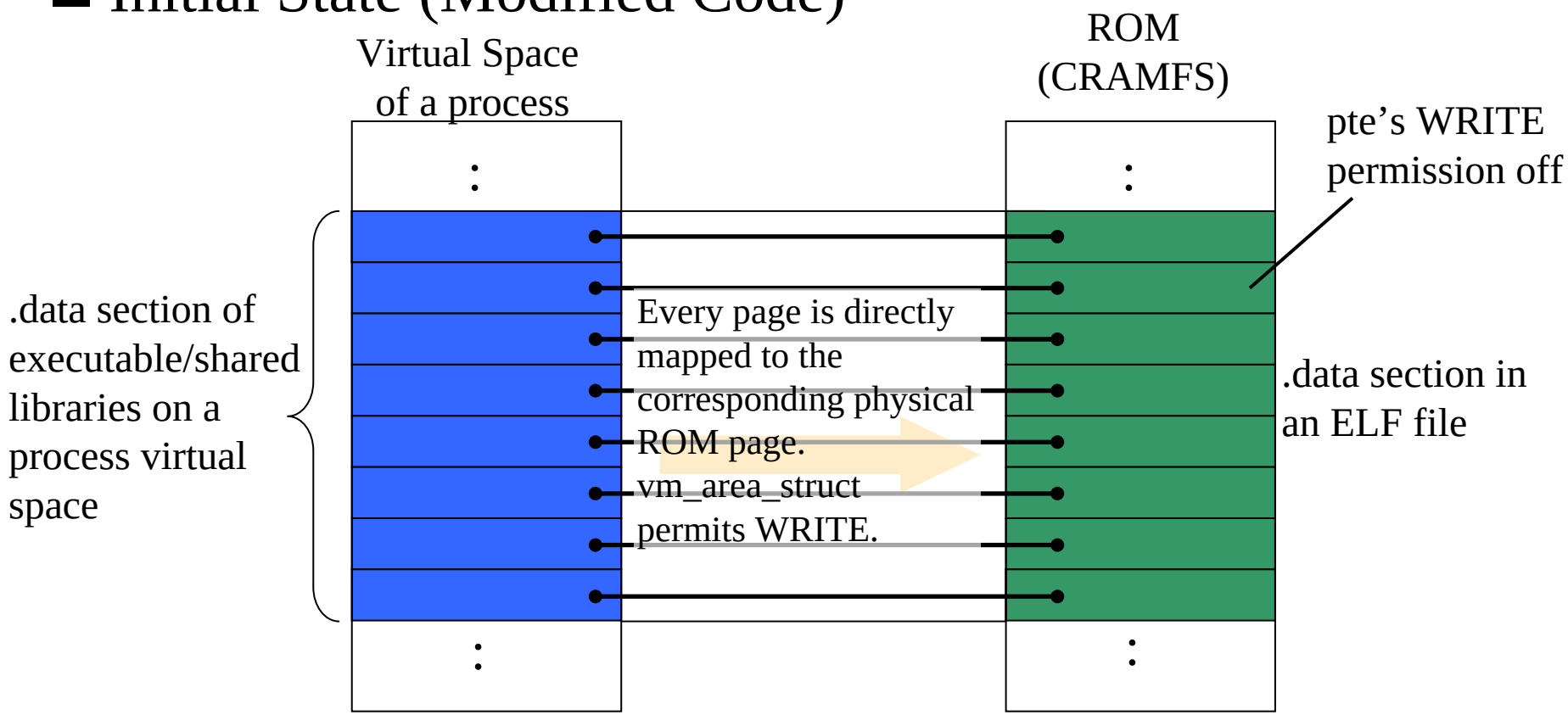
"Allocate On Write" for .data Sections (3/7)

- A page is read or written (Original Code)



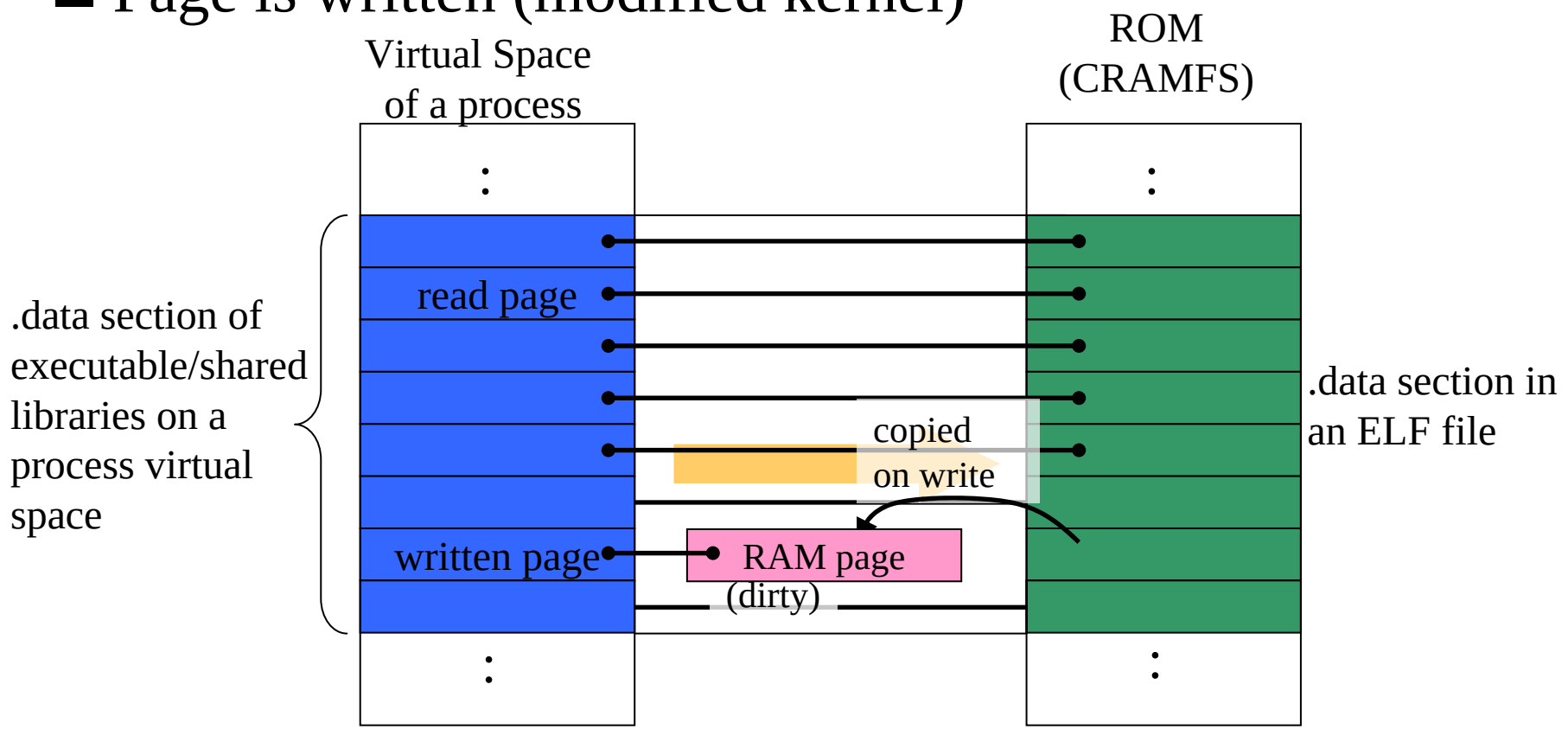
"Allocate On Write" for .data Sections (4/7)

Initial State (Modified Code)



"Allocate On Write" for .data Sections (5/7)

- Page is written (modified kernel)



"Allocate On Write" for .data Sections (6/7)

■ Changes of Code

No change in kernel code.

Subtle change to runtime dynamic linker (ld-linux.so) :

ORIGINAL
  mmap(…, PROT_READ | PROT_WRITE)
CHANGED
  mmap(…, PROT_READ)
  mprotect(…, PROT_READ | PROT_WRITE)

- Drop PROT_WRITE bit when "mmap"ing ELF segment.
  → Kernel (CRAMFS) maps every pages in the .data section to ROM page as if it was XIP .text.
- Then, Set PROT_WRITE by mprotect.
  vm_area_struct permits write.
  Pte's write permission off.
  → Copy-on-Write enabled

“Allocate On Write” for .data Sections (7/7)

- Result
  - Totally 26% reduction of page cache allocated to processes at stand by idle state.
  - ROM access latency slows process execution

    (This effect was trivial for our product case.)
- Issue
  - get_user_pages() does not work for the .data.

    Core dump, /proc/mem, ptrace and mlock may fail for pages in the .data section.

ROM Reduction by Thumb Instruction (1/2)

- Apply ARM Thumb mode to executables and shared libraries
- Linux kernel, glibc and most of standard libraries left to run in ARM mode.
- Average reduction rate of binary file size on ROM: 16%
- Totally 6% of Code ROM was reduced.
- No outstanding performance degradation at product level (4% slow down in average ?).
- Thumb mode interworking with ARM mode did not work with available Linux/gcc at the time.

  We fixed those problems with MontaVista.

ROM Reduction by Thumb Instruction (2/2)

Major problems found and fixed:

- Compiler/Linker:
  - Incomplete intermediate code fails to switch between thumb and ARM in certain conditions.
  - Wrong code can be generated for switch – case when –fPIC specified.
- Kernel
  - Cannot recover alignment fault happened at thumb code.
  - Cannot handle invocations of signal handlers implemented in thumb mode.
- Others
  - Prelink cannot handle symbol type specific to thumb.

…

Stability /Security
Memory Usage API (1/4)

Motivation:

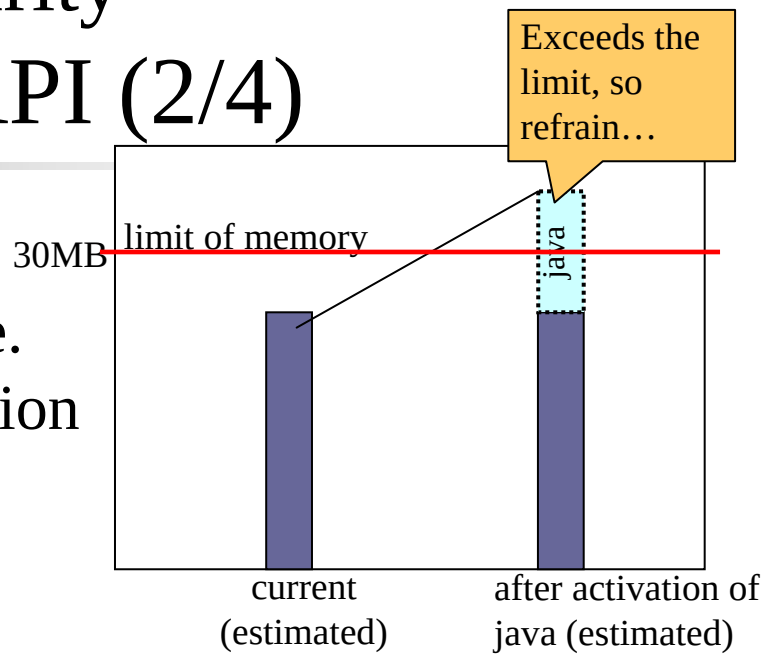
- Customer requirements:
  - Consumer expects mobile phones to be more stable than PC.
- Dynamic characteristics
  - Dynamic characteristics of memory usage introduced by Linux
  - Difficulty to estimate maximum memory usage at design time
- Narrow margin:
  - amount of usual memory usage level is close to the limit of real capacity.

Mobile phone should not crash or freeze when it accidentally hit the limit of memory.

Stability /Security
Memory Usage API (2/4)

Strategy:

- Estimate room of memory at runtime.
- Refrain from activating new application if current room cannot satisfy it. (a “memory alert” window pops up)



Existing means for estimating room of memory:

- /proc/meminfo: underestimates room by excluding pages which can shrink.

→ implemented a *memory usage API* to estimate current room of memory more exactly.

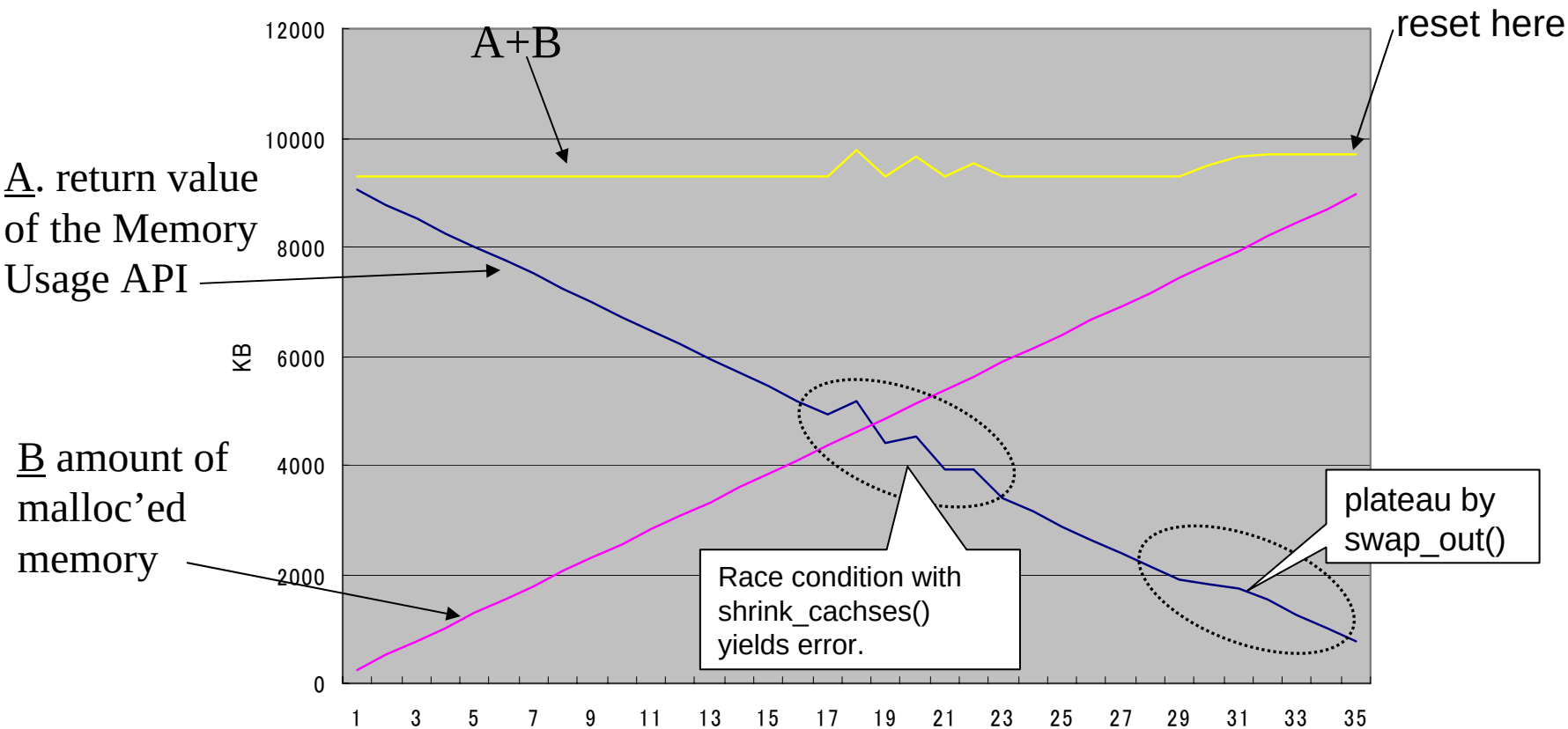
Stability /Security
Memory Usage API (3/4)

Memory usage API :

- Estimates amount of page cache and slabs to be reclaimed by shrink in addition to free pages.
- Execution time < 1msec
- Remained issue
  - Excludes i-node cache and directory entry cache to be reclaimed. (complexity, time consumption)
  - Race condition with shrink_caches() may cause inaccurate result.

Stability /Security
Memory Usage API (4/4)

Memory usage API gives a fairy exact estimation of room of memory.



User Response Time/Boot up time
Application of prelink

- Normal dynamic linked ELF: Over 2 sec. to start up a multimedia application process (fork ~ main)

Stages of Processing in Process Start up	elapsed time
1. Layout and map shared libs to virtual address space	96ms
2. Resolve symbol references	2354ms
3. Init of each ELF file	29ms

- Prelink eliminates symbol reference resolution proceses.

  2,479msec → 125msec

- Boot up time also reduced: Over 1min. → 20sec.

Realtime

- Deadline of mobile phone's AV processing is ~20msec.
- Stable AV processing can be realized by running it as RT threads.
- Remove resource contention condition with TSS threads from realtime path to avoid priority inversion.

  heap access, file access,

  locking for resources shared with TSS, …

Power Consumption Elimination of Tick Interrupt

- At stand by state, the system eliminates tick interrupt for saving power.
- CPU stops executing instruction and waits for interrupt in idle state.
  - Woken up once a minute by a timer interrupt for time adjustment
  - Woken up by other interrupts (e.g. key press, comm. events ..)
- When woken up, jiffies are recovered using external clock source.
- In calm environment, running time of CPU is reduced dramatically.

More Challenges Expected in Mobile Devices (1/2)

CE Linux works very well for mobile phones, while further improvements are also expected.

Very Specific Examples:

- Security
  - Application download OTA, installation of 3rd party software
  - Strict access control, resource usage limit, isolation…
  - Extensive improvement is needed for this area
- IPC Performance
  - ¼ of startup time for one of our *typical* application program is consumed by select system call.
  - Frequent message passing between threads in legacy application programs (inherited from RTOS based system).

More Challenges Expected in Mobile Devices (2/2)

- Priority Inversion
  - RT video threads cannot share resources between TSS threads (e.g. heap).
- Smart Memory Management Algorithm for Limited Virtual Memory
  - No Swap – No Virtually Unlimited Memory, Stability Requirement
  - Accurate means for current memory usage estimation
- Smaller (subset) configuration for mobile devices profile

…

Panasonic
ideas for life

Thank you!