VMM Detection approach Maniacs – No one can be believed ☺

Kunio Miyamoto a.k.a wakatono

wakatono@todo.gr.jp

Twitter ID: wakatono

Today's topic

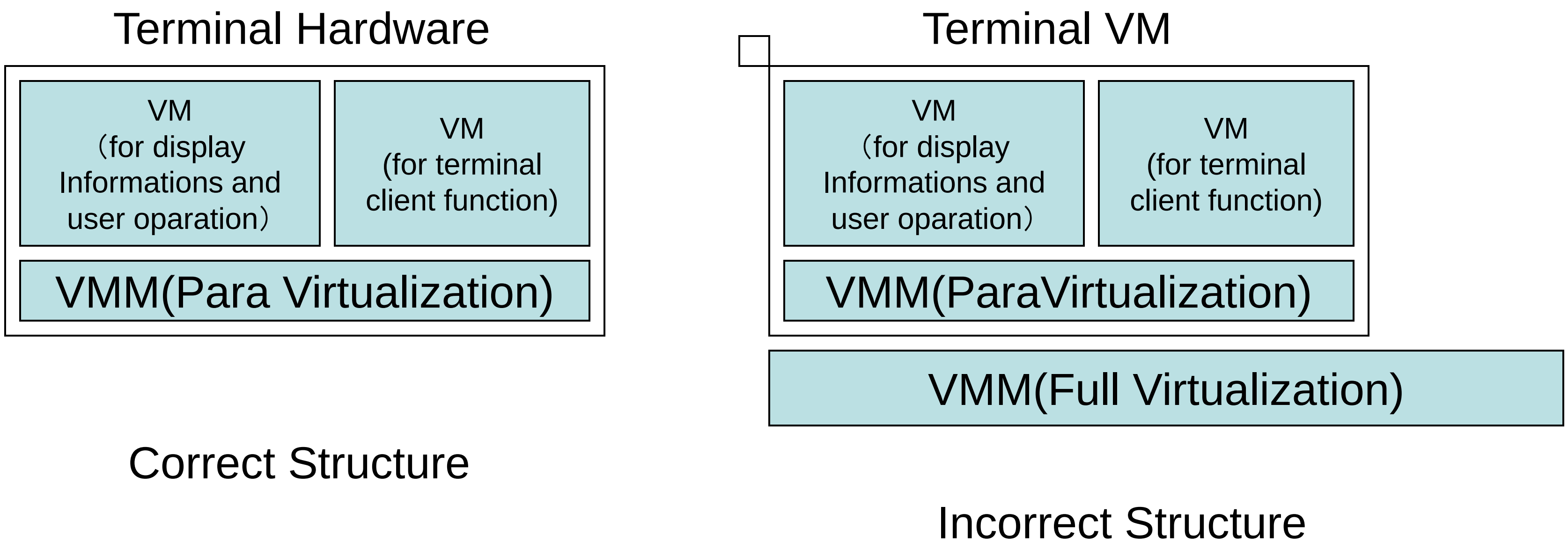
- Background
- VMM detection by using Implementation specific footprint (widely used)
- Problem of Implementation specific footprint use
- New VMM detection method by using a few assembly language instructions
- Sample Implementation using C and Assembler
- VMM detecting apploach in bootstrap process
- VMM detection Implementation in bootloader
- Conclusion

Background

Motivation of This Research

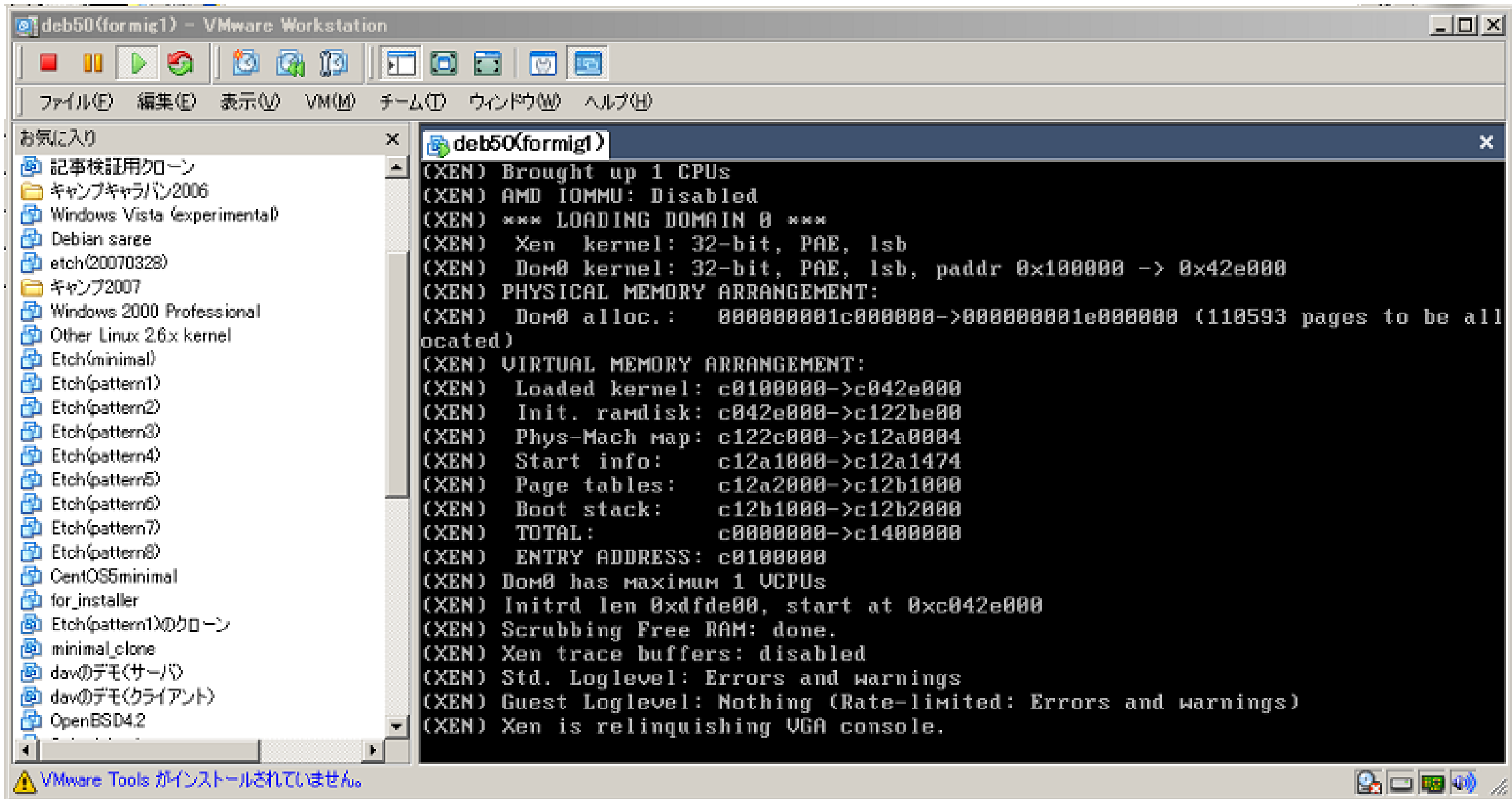
- Some Software must not to be run on the "Virtual Machine Monitor"
  - Assuming Real Hardware as authorized hardware(not Virtual Machine)
  - Software includes Operating System
  - Assuming TPM is not available on the computer
- When avoiding to use some system including OS, VM detecting in bootstrap is needed
  - VMM Detecting code must be made more smaller.

Why for Some Software?



- I want to detect running on “Incorrect Structure”
  - Architecture like “VM on VM” is not suitable for our use

VM on VM example(1)



- Xen is running on VMware Workstation
  - I don’t want to run our software like this situration

VMM detection by using Implementation specific footprint (widely used)

Hints for VMM Detection

- Popularly Used
  - String, Specific Value, Specific Instruction, etc…
- Performance Mismatch
  - Instruction Execution Time
  - Cache Hit Rate difference
- Functionally Mismatch
  - TLB
  - VMM Bugs
- Tools
  - Imvirt, virt-what
  - Checkvm module (in Metasploit)
  - Various malware modules for anti-debugging
  - RedPill

  etc…

Example: QEMU based VMM

```
$ dmesg | grep QEMU
[    0.853843] ata2.00: ATAPI: QEMU DVD-ROM, 0.12.1, max UDMA/100
[    0.855265] scsi 1:0:0:0: CD-ROM            QEMU     QEMU DVD-ROM
0.12 PQ: 0 ANSI: 5
[    1.208713] usb 1-1: Product: QEMU USB Tablet
[    1.208715] usb 1-1: Manufacturer: QEMU 0.12.1
[    1.742372] input: QEMU 0.12.1 QEMU USB Tablet as
/devices/pci0000:00/0000:00:01.2/usb1/1-1/1-1:1.0/input/input4
[    1.742508] generic-usb 0003:0627:0001.0001: input,hidraw0: USB HID
v0.01 Pointer [QEMU 0.12.1 QEMU USB Tablet] on usb-0000:00:01.2-
1/input0
```

- String “QEMU” is appeared here and there ☹

VMM detection tools example(1/3)

- Imvirt
  - VMM detection tool
  - Project Webpage: http://micky.ibh.net/~liske/imvirt.html
  - Known Footprint like resource strings, specific belaviors(e.g. I/O port access result), etc…

VMM detection tools example(2/3)

- virt-what
  - VMM detection tool
  - Project Webpage: http://people.redhat.com/~rjones/virt-what/
  - Known Footprint like resource strings, specific belaviors(e.g. I/O port access result), etc…

VMM detection tools example(3/3)

- Checkvm
  - One of modules in Metasploit.
    - Script in Metapreter
  - Checks the exploited machine is running on some VMM.
  - Win32/Win64 only

Problem of Implementation specific footprint use

Some challenges in VMM detection

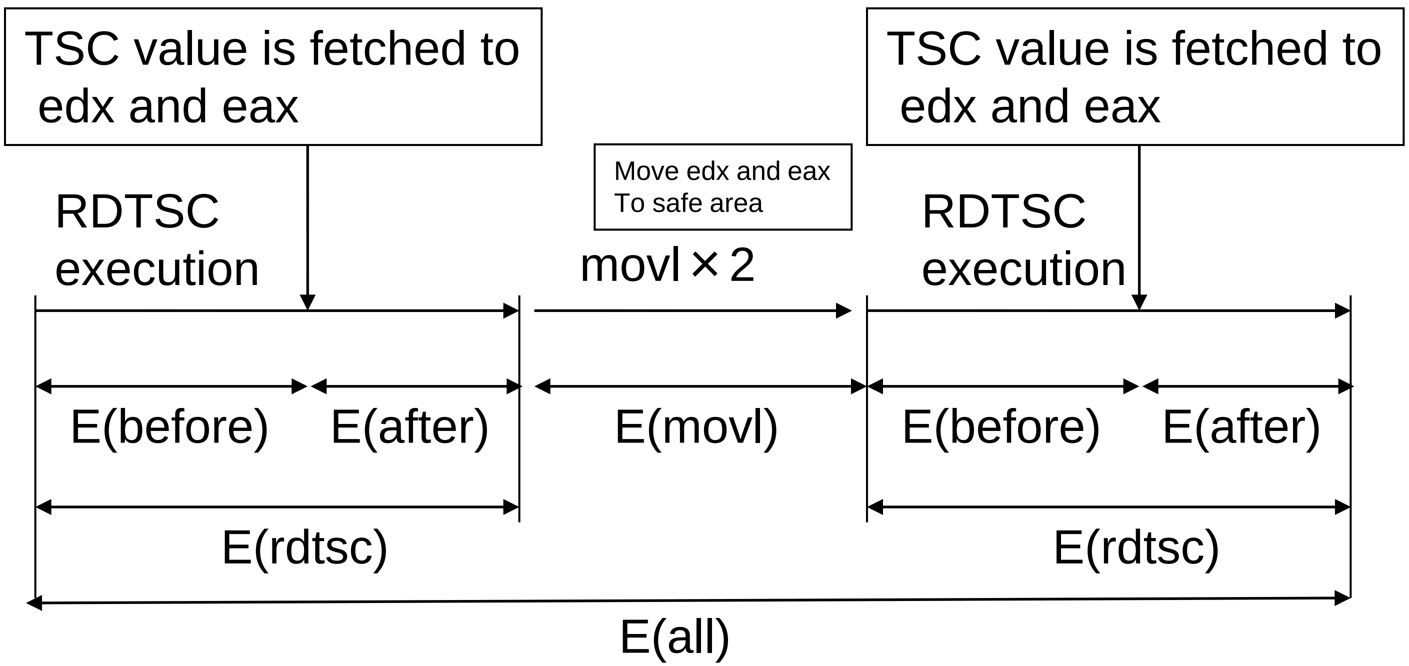
- Footprint detection is easy to bypass detection
  - e.g. Virtual Disk for VMware, vCPU for KVM, etc…
  - Detection by comparing the resource specific string(s) is easy to implement, but easy to fake ☺
- Userland application cannot be use features like raw features of TLB, CPU Cache, etc…
  - These are usable only in the kernel mode.
- Targets are Specific Operating Systems
- Known VMM can be detected
- No one can be believed! (voice from user mode)

New VMM detection method by using a few assembly language instructions

Assumption of This approach

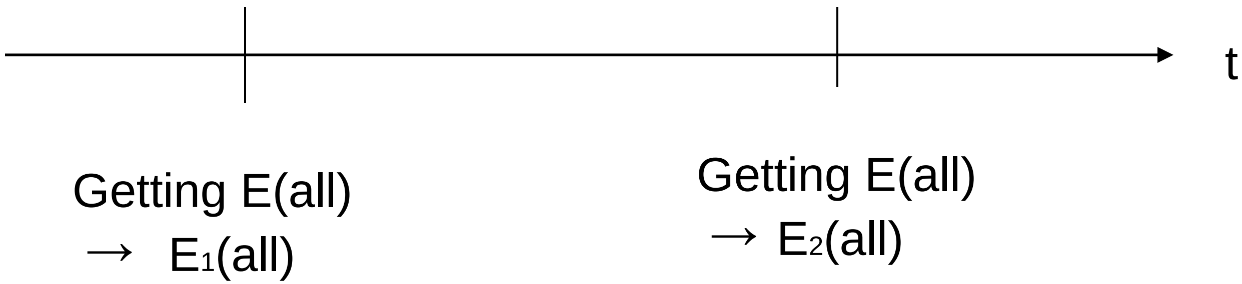
- VMM provides VM(s) fully-virtualized environment
- VMM provides IA32-based environment
- VM(s) on VMM has independent TSC (important!)
- RDTSC instruction can be executable on ring level 3 (important!)
  – Popular OSs enables to run RDTSC on ring level 3

TSC measuring for VM detecting architecture



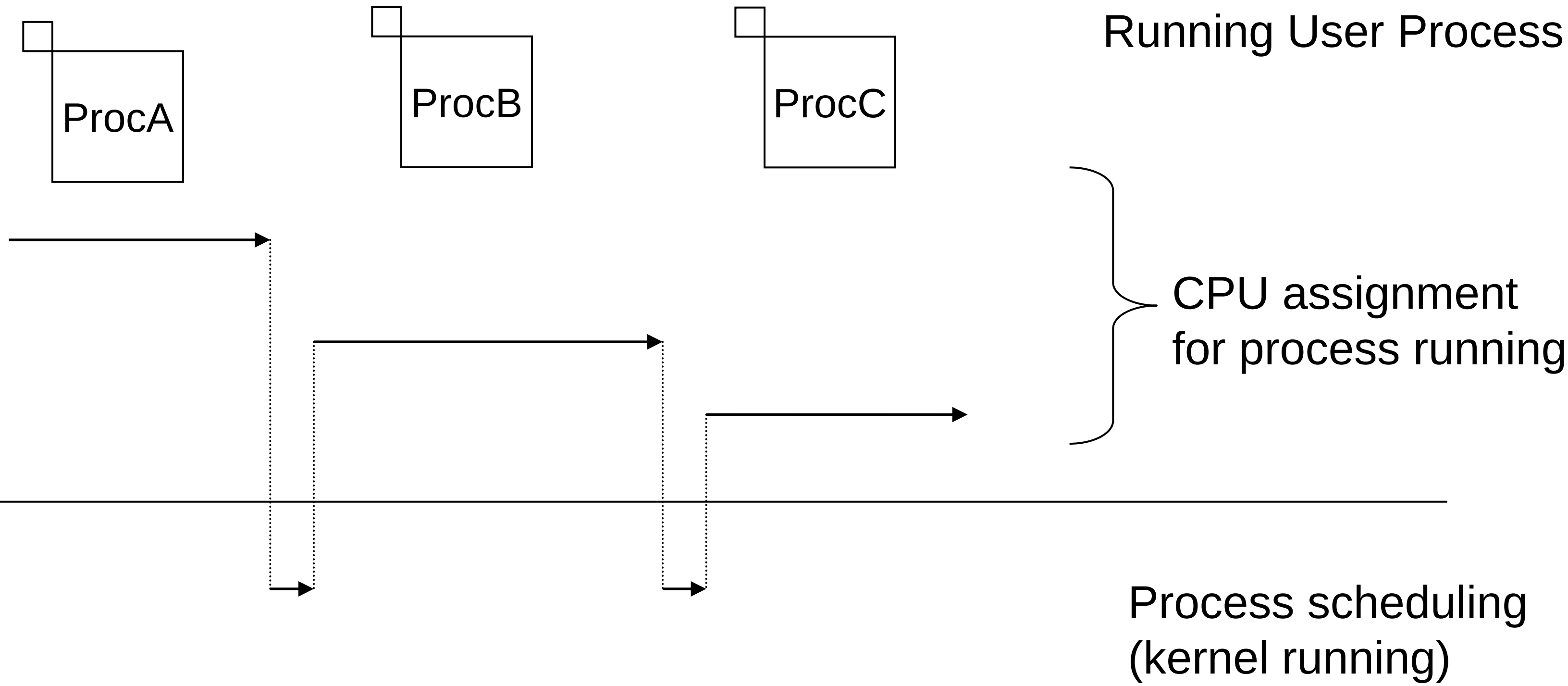
- On the Real Hardware, E(all) is available and always same value
  - On the Virtual Machine, E(all) differs per timing of getting E(all)

Judging method
VM or Real Machine

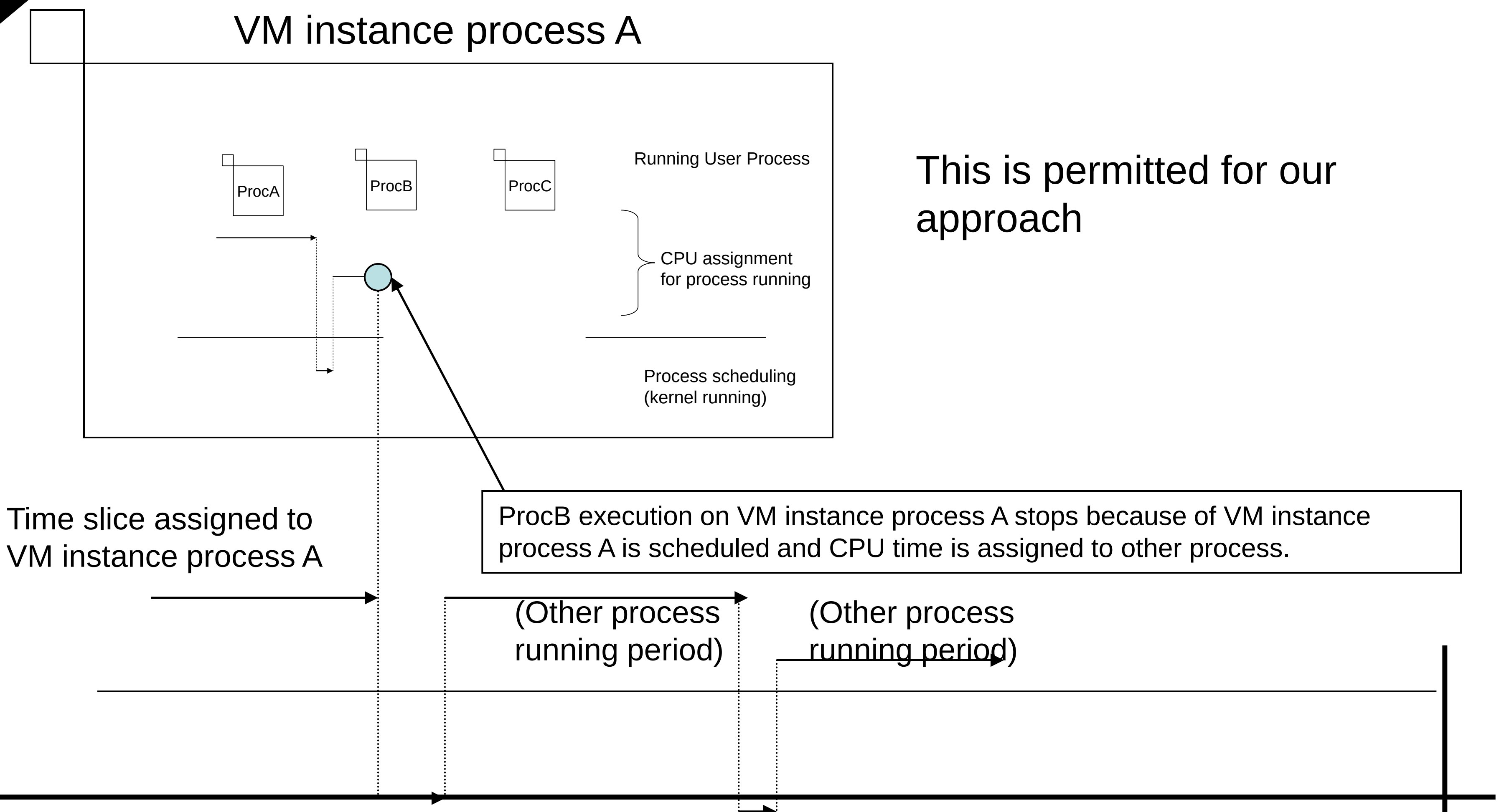


- If E_1(all)=E_2(all) then Program is running on Real Machine
- If E_1(all) != E_2(all) then Program is running on VM

Process Scheduling and time slice assignment



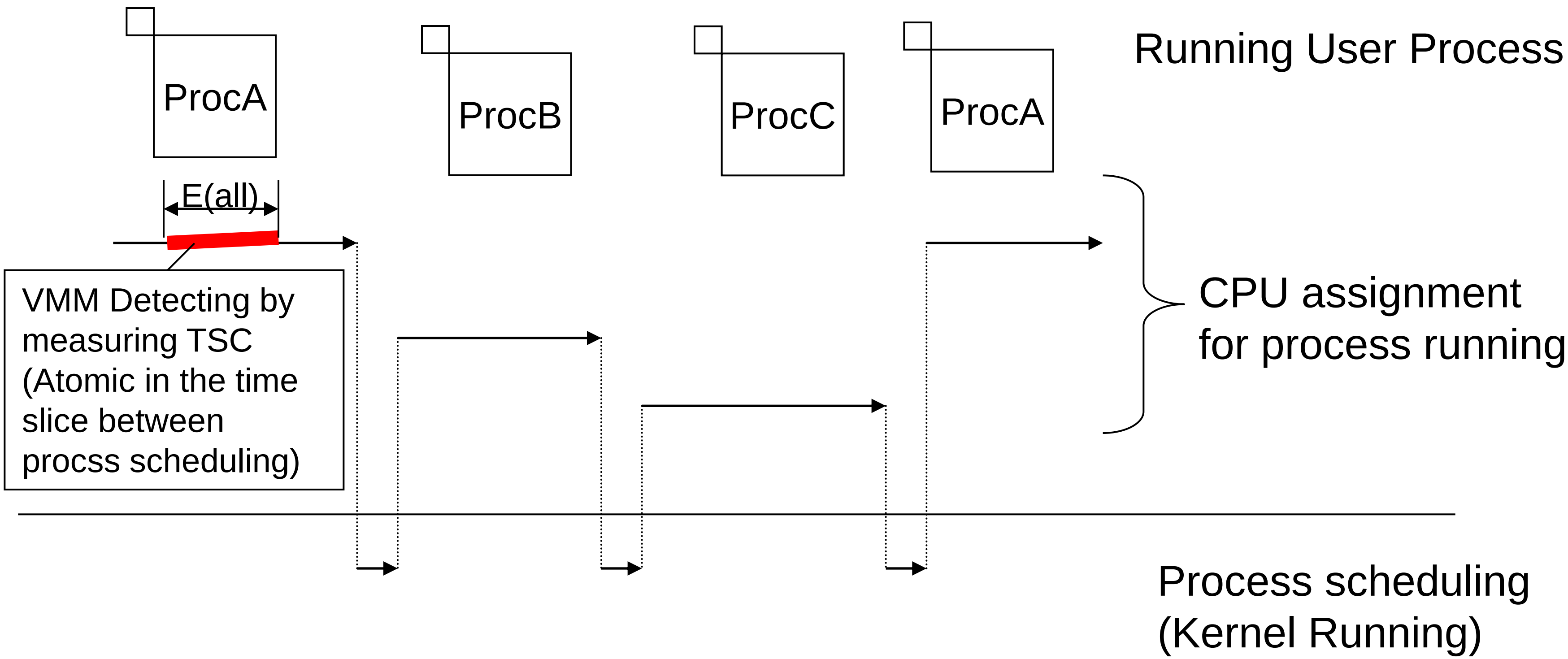
Process executing in VM is suspended by whole VM preemption



This is permitted for our approach

Copyright by Kunio Miyamoto

VMM detecting process between process dispatch timing.



Sample Implementation using C and Assembly Language

Simple!

CPUID
RDTSC
MOV EBX, EAX
MOV ECX, EDX
RDTSC
(EDX:EAX – ECX:EBX)

- CPUID resets Out-of-Order execution in IA32
- These instructions makes RDTSC execution clock value
  - And this value is not stable on the VMM

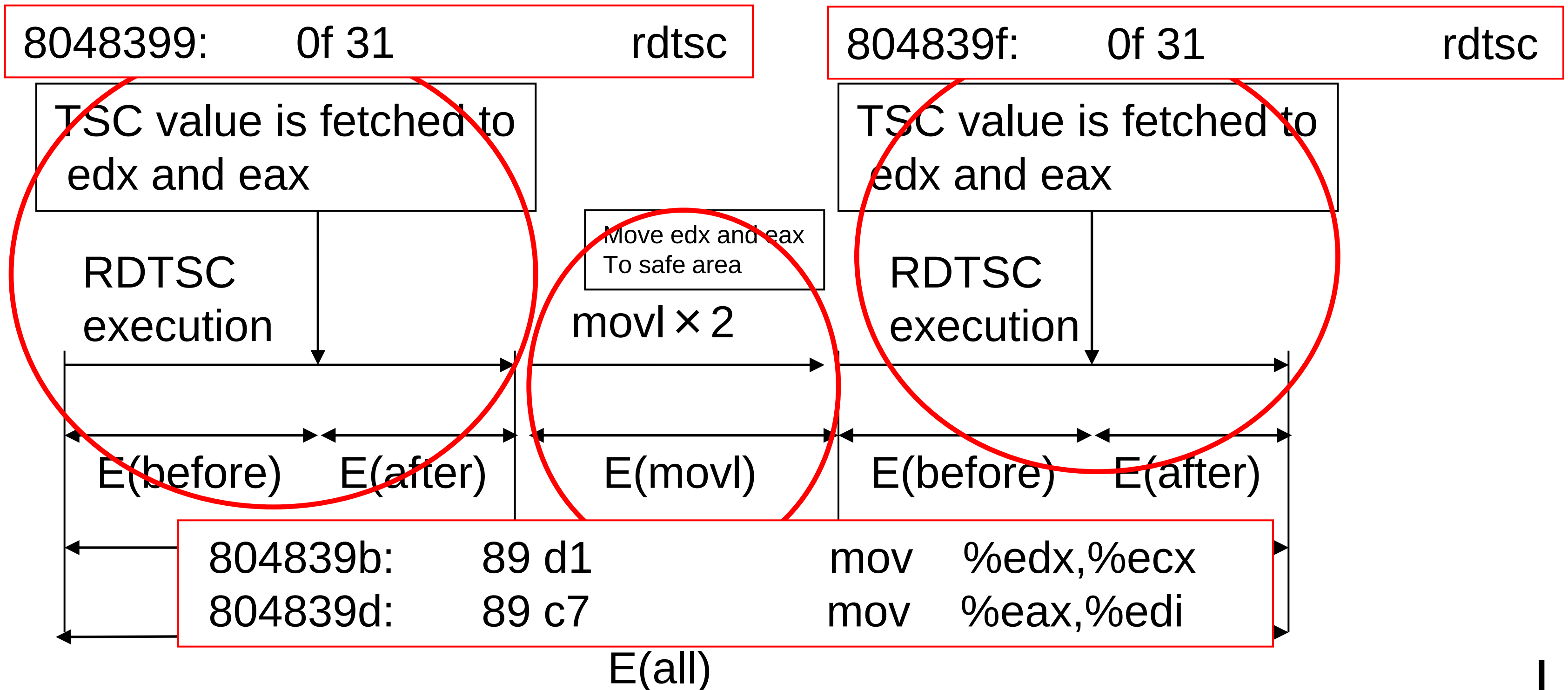
Real Code
(C and Inline Assembler)

```
#include <unistd.h>
#include <sys/types.h>
main()
{
        unsigned  long long  before,after;
        char *area;
        register unsigned long bhi,blo,ahi,alo;
        unsigned long long bhi64,blo64,ahi64,alo64;

        __asm__("cpuid" : );
        __asm__(".byte 0x0f,0x31" : "=a" (blo),"=d" (bhi));
        __asm__(".byte 0x0f,0x31" : "=a" (alo),"=d" (ahi));
        blo64 = blo;
        bhi64 = bhi;
        alo64 = alo;
        ahi64 = ahi;
        before = bhi64 << 32 | blo;
        after = ahi64 << 32 | alo;
        printf("%lld\n",after - before);
}
```

```
8048397:    0f a2     cpuid
8048399:    0f 31     rdtsc
804839b:    89 d1     mov    %edx,%ecx
804839d:    89 c7     mov    %eax,%edi
804839f:    0f 31     rdtsc
```

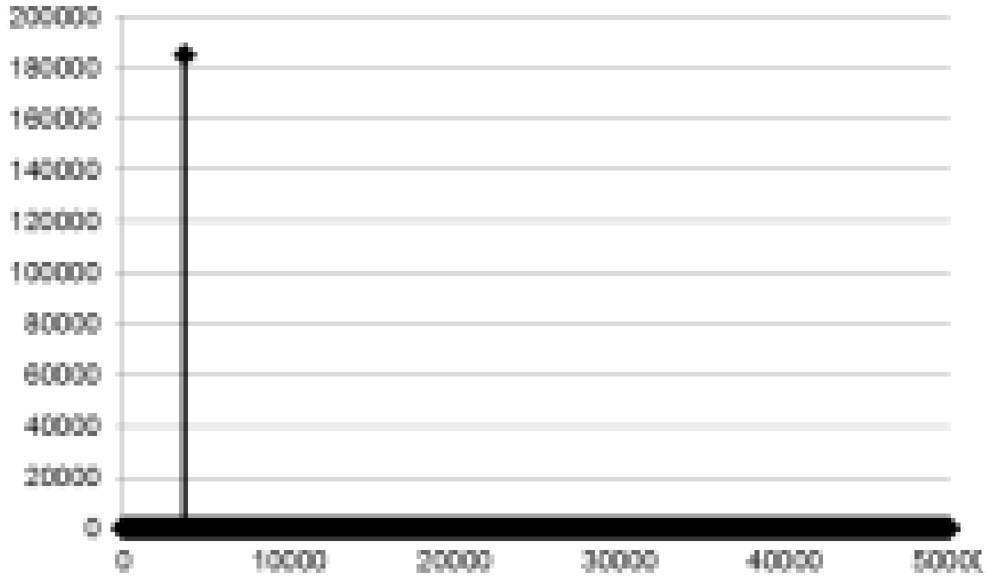
TSC measuring for VM detecting architecture



- On the Real Hardware, E(all) is available and always same value
  - On the Virtual Machine, E(all) differs per timing of getting E(all)

Result on Real Hardware

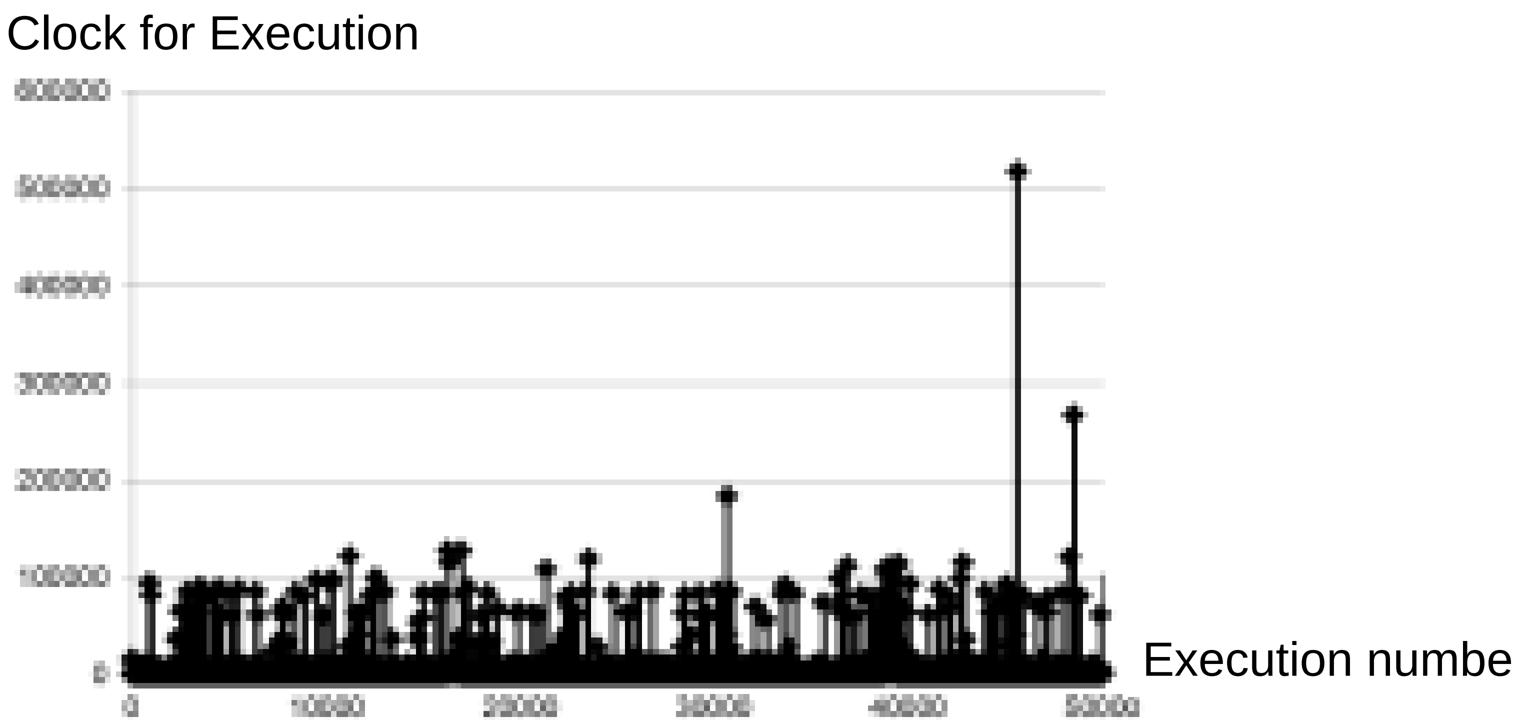
Clock for Execution



Execution number

Stable ☺

Result on Virtual Machine (on VMM)



Not Stable ☹

In case of Intel VT/AMD-V

- This approach cannot be applied
  - Need to be modified little a bit ☺
- Trapped Instruction in Intel VT/AMD-V is:
  - CPUID ☺
- Modify the Code ! (little a bit ☺)

Simple!

```
CPUID
RDTSC
MOV ESI, EAX
MOV EDI, EDX
XOR EAX, EAX
INC EAX
CPUID
RDTSC
(EDX:EAX – EDI:ESI)
```

Added code (3 lines) ☺

- 1st CPUID resets Out-of-Order execution in IA32
- These instructions makes CPUID + RDTSC(+α) execution clock value
  - And this value is not stable(and/or too large) on the VMM execution.

Strength and Weakness

- Strong Point
  - Detect underlying VMM (or other software like VMM)
  - Detect unknown(or newer) VMM running
  - Small and Simple Code
    - Can be included in many software
- Weak Point
  - Unable to know the name of VMM(or other software like VMM)

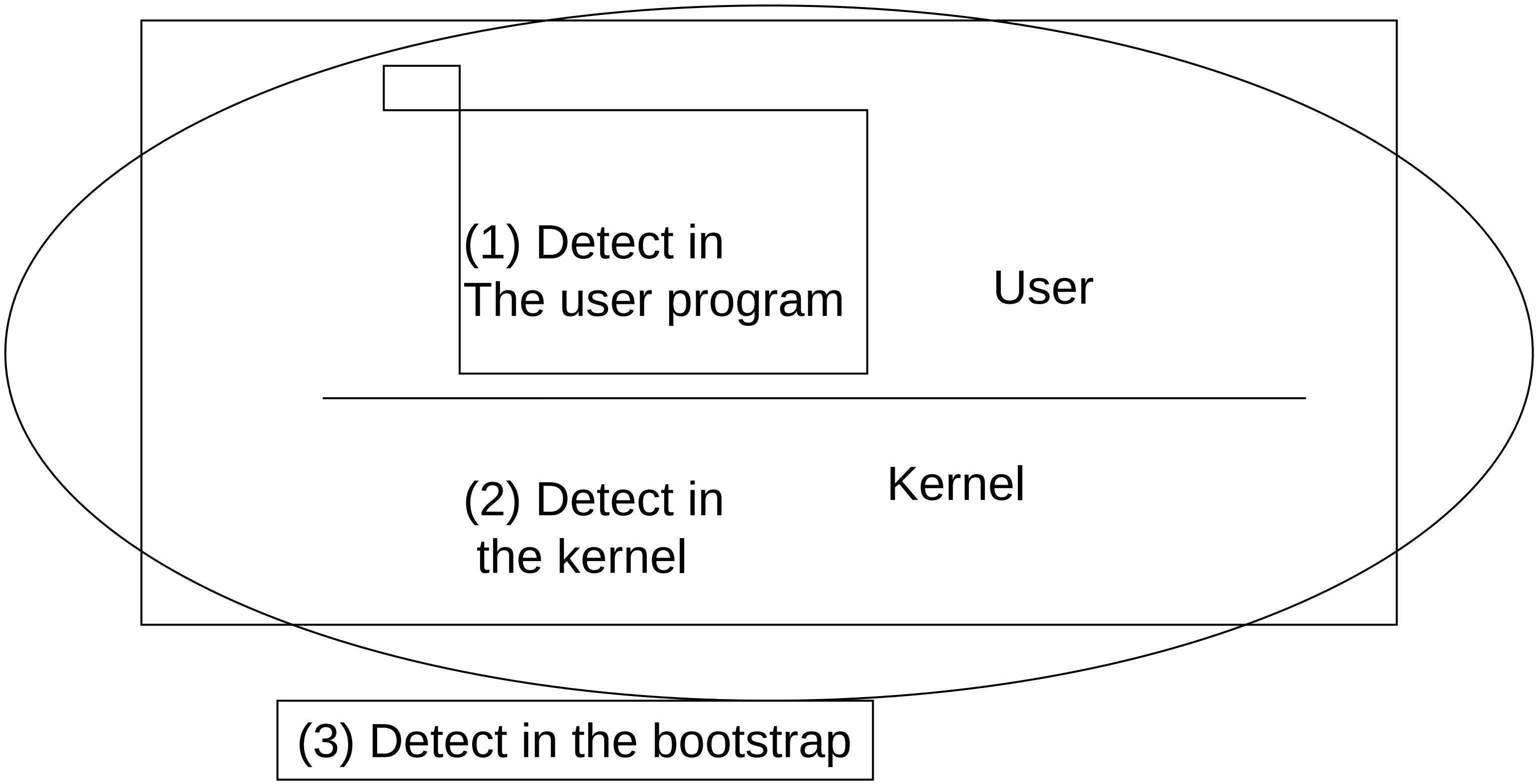
Caution: many kind of TSC

- I know at least 3 kinds of TSC
  - (normal) TSC
    - Normal TSC
    - Count up by CPU cycle
  - Constant TSC
    - Count up interval is fixed time ☹
      - Not related to the CPU cycle.
    - Interval Specified in the boottime clock
  - Invariant TSC
    - Don’t stop when CPU sleeped ☺

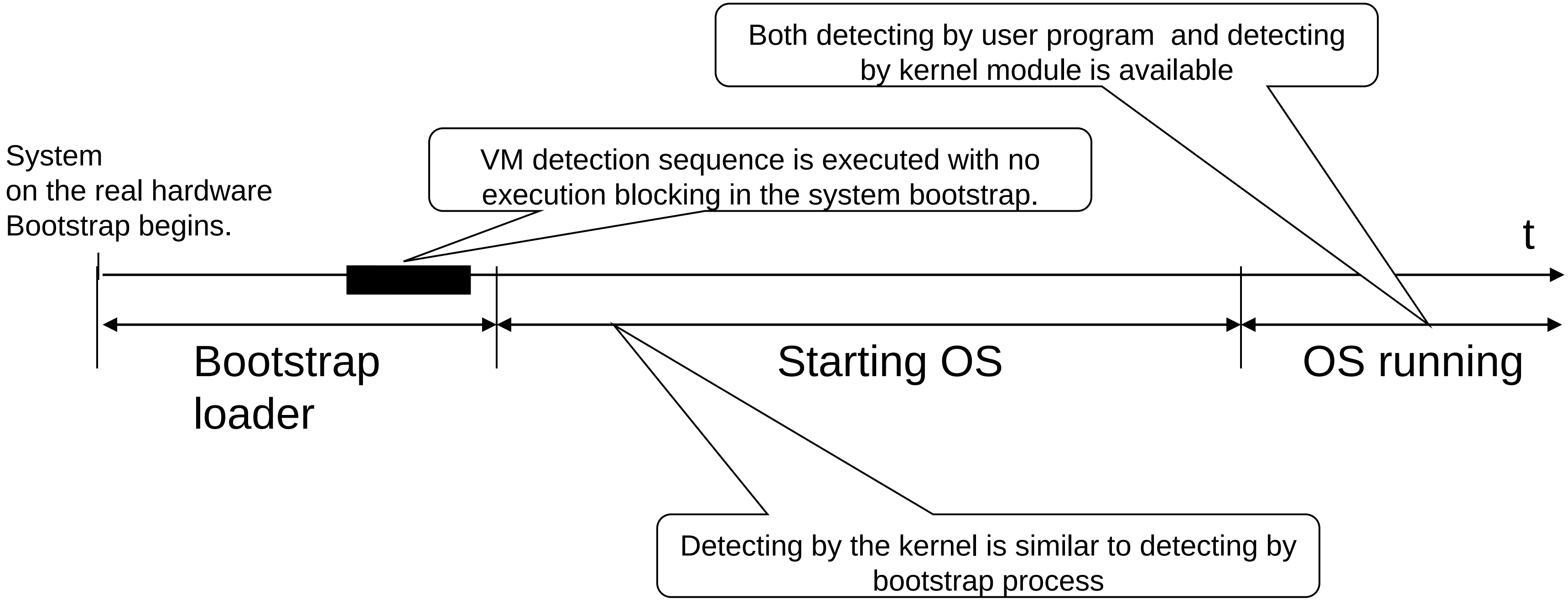
In this case,
Code in this program
Returns various clock
Because of CPU clock is modified
Dynamically.

VM detecting approach in bootstrap process

VM Detecting Approach



Timeline from bootstrap to running OS



Point of VMM detection

- VMM detection by kernel module
  - One of most reliable approach
  - Some restrictions
    - Runnability of kernel module depends kind of OSs and these versions
      - e.g. Linux kernel module for 2.4.x cannot be used for Linx kernel 2.6.x
- VMM detection by user process
  - Easy to use from user programs
  - Less reliable than by kernel module
  - Suppressing user process preemption is not practical in the general OS.
- VMM detection by bootstrap process
  - Running no process
  - If Underlying VMM exists, any of preemption is caused because of VM and other processes scheduling

Focus

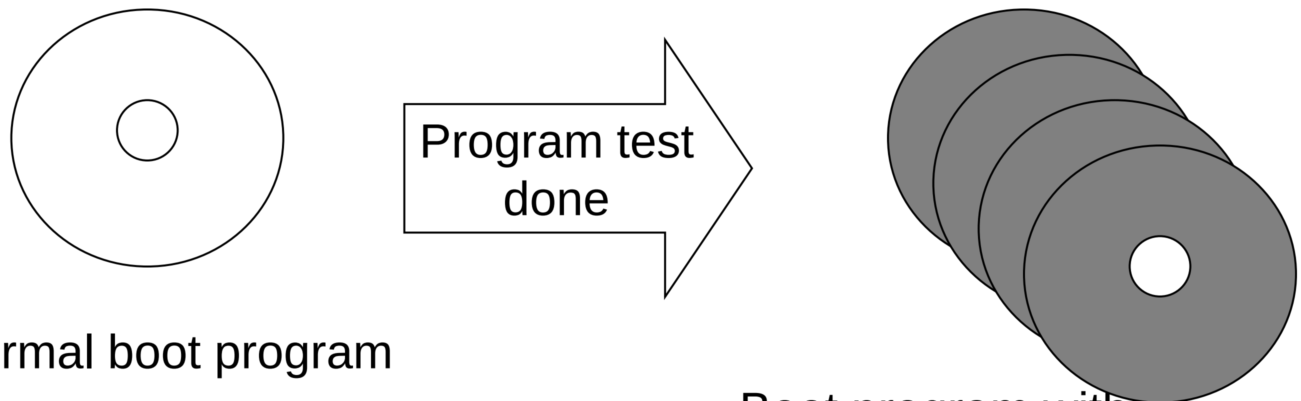
Benefit of Detection in Bootstrap Process?

- Hardware Stability
  - HW processing speed is stable just after powered on.
  - VM processing speed is not stable just after (VM) invoked
- (Real) Hardware Occupation
  - Real HW is occupied by bootstrap loader. →Stable in processor speed.
  - HW is not occupied by bootstrap loader. →Unstable in processor speed.

Practical use of bootstrap VM detector

Program Developing
and Testing(includes OS)

Program Deploying
or Distributing(includes OS
or distributed as an appliance HW)

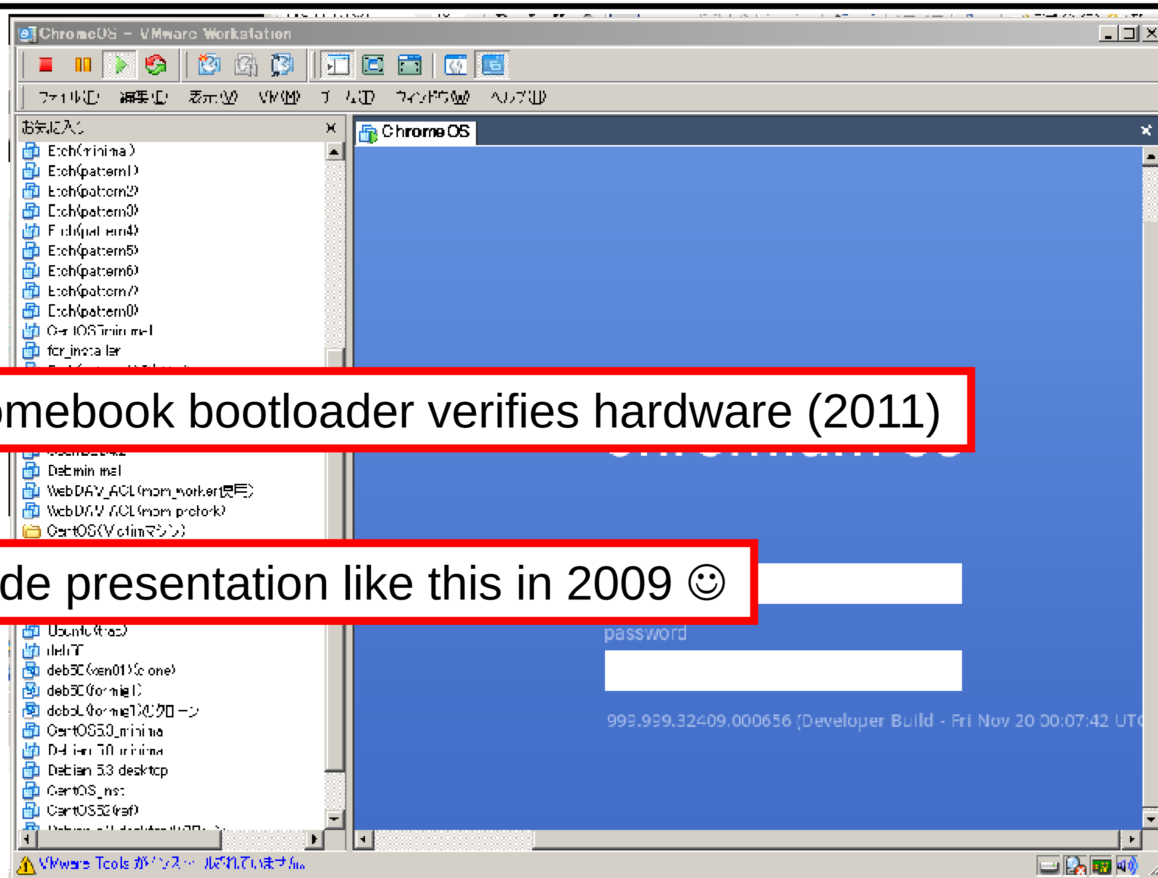


Normal boot program

Boot program with
VM detection.

- Usable for appliance hardware development and deployment
  - Bootstrap VM detector completes in the boot process, and not affects OS initialization processing

Example(2)



Chromebook bootloader verifies hardware (2011)

I made presentation like this in 2009 ☺

- Chrome OS(Chromium OS) will initially be targeted and the netbook class or products (in Google Chrome OS press conference)

Implementation

- Assuming to use GNU GRUB
  - GNU GRUB is the bootloader to use Generic Operating System boot
- Now in progress
  - Runs detection mechanism on custom-made GRUB 1.x (not yet 2.x)

Conclusion

- I proposed new VMM detection approach
  - Smaller Code, and Useful Results
  - And now in progress to develop VMM detection software usable everywhere.
- Bootstrap VM detection is more useful than VM detection in each application if possible.
- I pointed that VMM detection in boottime is useful for System-Wide structure assurance.

Thank you!