CS223
Lecture 03: System Virtual Machines -- CPU Virtualization

Kasidit Chanchio
Department of Computer Science,
Faculty of Science and Technology,
Thammasat University
1st Semester 2025

Trends: CPUs for Cloud Data Center

- CPU จะมี Cores เพิ่มมากขึ้น

AMD EPYC CPU Codenames[11][12]

Gen	Year	Name	Cores
1st	2017	Naples	32 × Zen 1
2nd	2019	Rome	64 × Zen 2
3rd	2021	Milan	64 × Zen 3
	2022	Milan-X	64 x Zen 3
4th	2022	Genoa	96 × Zen 4
	2023	Genoa-X	96 × Zen 4
		Bergamo	128 × Zen 4c
		Siena	64 × Zen 4
5th	2024	Turin	
6th	2025	Venice	

Intel Xeon Sapphire Rapids-WS Specifications*

Processor	Stepping	Cores / Threads	Clock Speed	L3 Cache	TDP
Xeon W9-3495X	Xeon W9-3495X	56C/112T	1.9 GHz	105 MB	350W
SR-WS ES2	D0 (QYQU)	56C/112T	?	105 MB	350W

https://www.tomshardware.com/news/intel-xeon-sapphire-rapids-ws-specs-leaked-up-to-56-cores-350w-tdp

NVIDIA Grace CPU Superchip

NVIDIA Grace CPU Superchip uses the NVLink-C2C technology to deliver 144 Arm® v9 cores and 1 TB/s of memory bandwidth.

- High performance CPU for HPC and cloud computing
- Super chip design with up to 144 Arm v9 CPU cores

https://www.nvidia.com/en-us/data-center/grace-cpu/

Genoa, the first Zen 4 based Epyc CPUs will be built on a TSMC 5nm process node and support up to 96 cores and 192 threads per socket, alongside 12 channels of DDR5,[7] 128 PCIe 5.0 lanes, and Compute Express Link 1.1.[8] AMD also shared information regarding the sister-chip of Genoa,

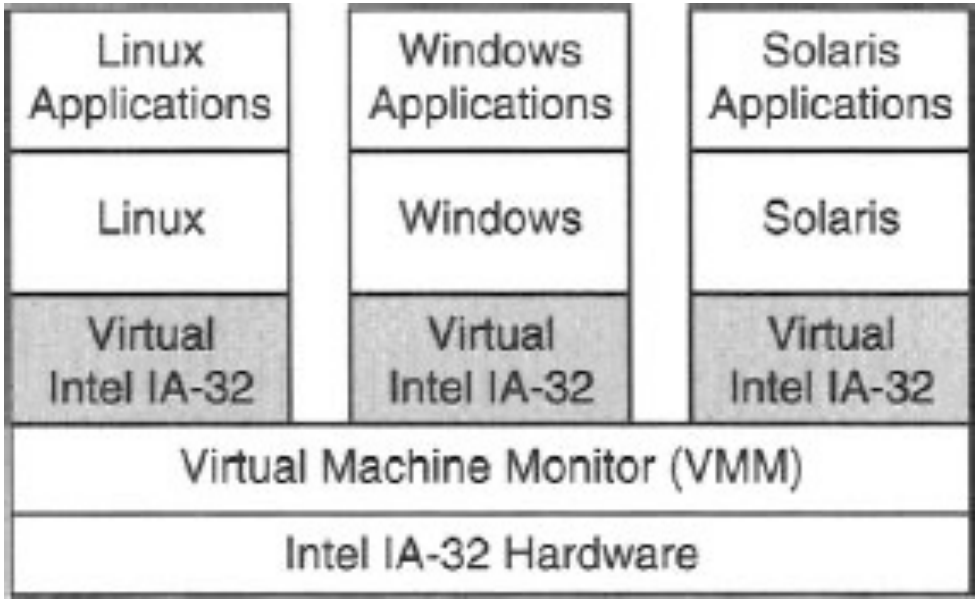
https://en.wikipedia.org/wiki/Epyc

Outline

- Usages
- Theory
- Processor Virtualization (Software-based)

VMM

- Virtual machine monitor (VMM) (หรือ Hypervisor) คือ software ที่บริหารจัดการ resources บนเครื่อง Host computer เพื่อให้บริการๆประมวลผลของ VMs



VMM

- เนื่องจาก Host มักจะ Idle เป็นส่วนใหญ่ VMM เป็น software ที่ช่วยลด Idle time บน Host โดยการรัน VM หลายๆ VM พร้อมๆกัน
- เรานิยาม Host ให้หมายถึงเครื่องคอมพิวเตอร์จริง
- นิยาม Virtual Machine (VM) ว่าเป็นการประมวลผลของเครื่องคอมพิวเตอร์เสมือนเครื่องหนึ่งๆซึ่งประกอบไปด้วย
  - Guest OS หมายถึง Operating System ของ VM
  - Applications หมายถึง Application Processes ของ Guest OS

Usages

- Multiprogramming ในระดับ VM: คือการที่ VM หลาย VM สามารถ Share Resources ด้วยการทำ context switching ได้
- Multiple secure environments (sandbox): Apps ที่รันอยู่ใน VM แต่ละเครื่องมี OS และสภาพแวดล้อมของตนเอง ไม่สามารถเข้าถึงได้โดยตรงจาก OS ของเครื่อง Host
- Managed App Environments: Apps สามารถรันพร้อมกับสภาพแวดล้อมที่เหมาะสมได้เลยเนื่องจาก App นั้นอาจได้รับการพัฒนามาบน OS ใด OS หนึ่ง เช่น Ubuntu หรือ Windows ที่มี versions หรือ configuration เฉพาะ
- Mixed-OS environments: Host เครื่องเดียวสามารถรัน VMs ที่แต่ละ VM มี OS ที่แตกต่างกันได้ในขณะเดียวกัน
- Multiple Single-Application VMs: สามารถกำหนดให้แต่ละ VM รัน App เดียวได้ เพื่อเพิ่มประสิทธิภาพในการทำงานของ VM (อาจไม่มี OS หรือมี OS ก็ได้) e.g. unikernel

Usages

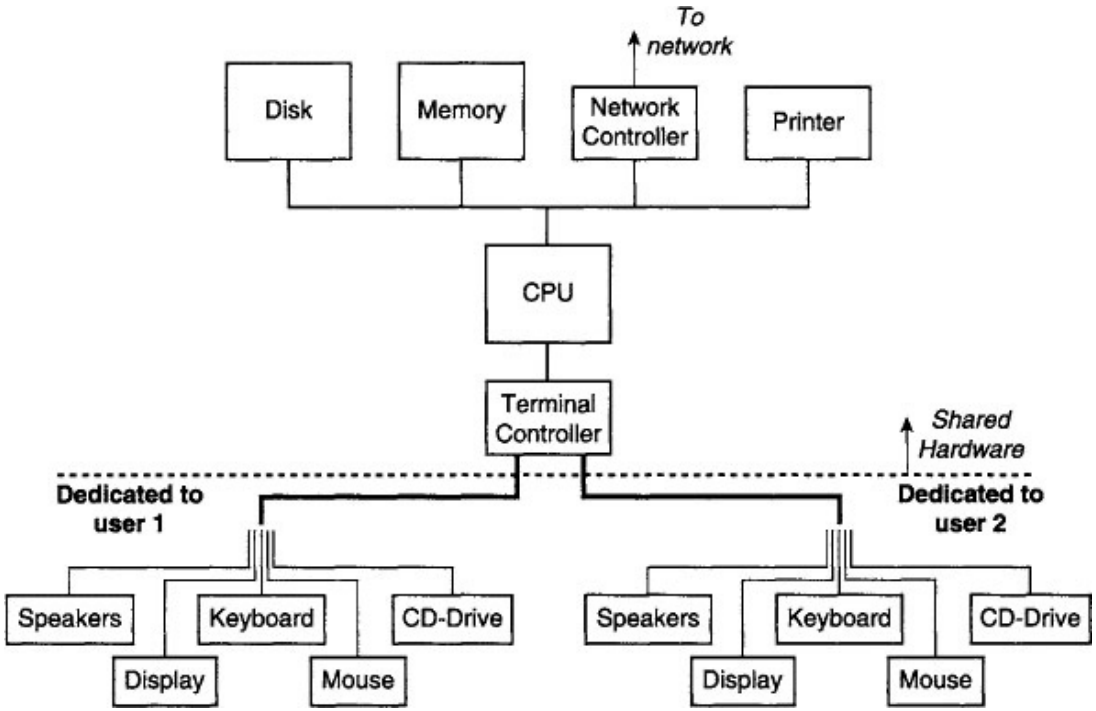
- Legacy Apps: รัน App สมัยเก่าที่ต้องใช้สภาพแวดล้อมเก่าๆ
- Multiplatform App Developments: บนเครื่องเก่า สามารถสร้าง VM รันสภาพแวดล้อมใหม่เพื่อพัฒนา SW สำหรับอนาคตได้
- New System transition: สามารถทดสอบการติดตั้งและใช้งานระบบใหม่บน VM ได้ก่อนที่จะย้ายไปบนเครื่องจริง
- System software development สามารถพัฒนา OS features: ใหม่ๆได้
- OS training: ใช้ VM เพื่อสอน นศ และพนักงานให้รู้จักใช้งาน OS ใหม่ๆ
- Help Desk supports: ใช้ทดสอบสภาพแวดล้อมของเครื่องลูกค้าเพื่อแก้ปัญหาบางอย่างทางไกล
- OS instrumentation: VMM can instrument access to resources of OS สามารถใช้ VMM เก็บสถิติการใช้ง่น resource เพื่อดูพฤติกรรมของ OS และ App

Usages

- Event monitoring for logging trace of execution: เนื่องจาก VMM เป็น software มันจึงสามารถ log คำสั่งที่มันประมวลผลได้ เราสามารถนำ log file มาใช้วิเคราะห์ bug ของ OS และ Apps ที่รันยน VM ได้ โดยการย้อนกลับ (replay) การทำงานของคำสั่งเหล่านั้นทีละคำสั่ง
- System encapsulation: VMM สามารถเก็บสถานะการทำงานของ VM ด้วยการรวบรวมข้อมูลของ context และ memory และ disk image ของ VM ณ. ขณะใดขณะหนึ่ง แล้วเก็บไว้เพื่อเริ่มการทำงานใหม่ หรือย้ายสถานที่เพื่อรันบน Host เครื่องใหม่ได้
  - VMM เราสามารถเปลี่ยนการประมวลผลของ VM เป็นข้อมูลสถานะ (state) ได้
  - เราสามารถข้อมูลสถานะนั้นกลับมาเป็นการประมวลผลของ VM ได้

Single System Illusion

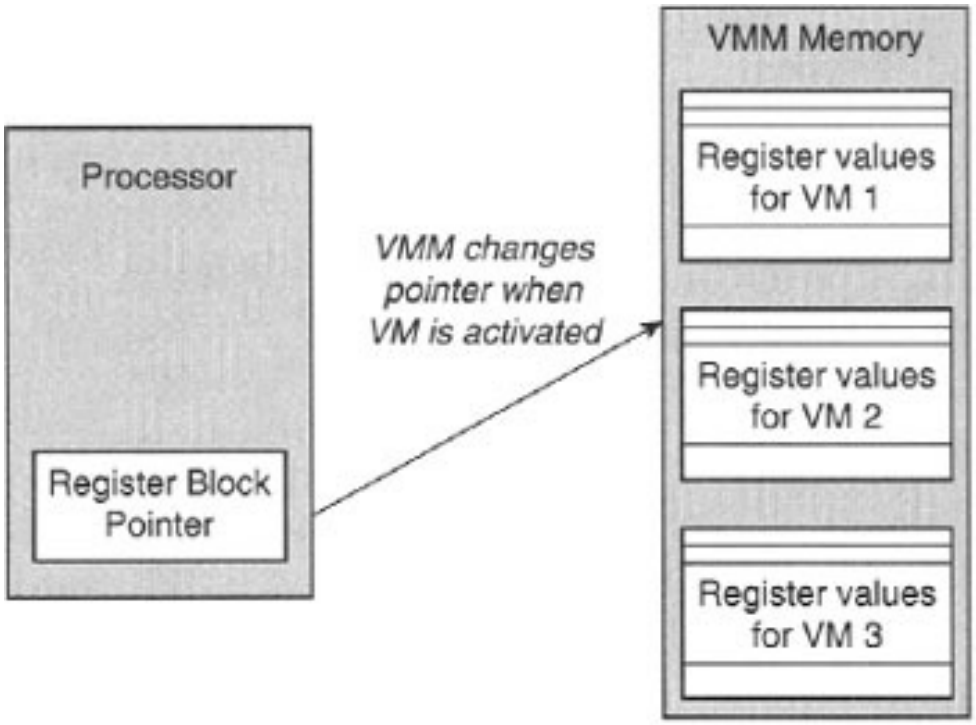
- เราพยายามสร้างภาพลวงว่าคอมพิวเตอร์เป็นของผู้ใช้คนเดียว
- ยกตัวอย่างเช่น การรัน DOS window บนเครื่อง windows 10



State Management

- Context switching ของข้อมูลสถานะของ VM (VM state)
  - การสลับการทำงานของ VM ต้องมีการทำ Context Switching
- State ของ VM ถูกเก็บใน host memory
- State indirection v.s. State copying

State Indirection



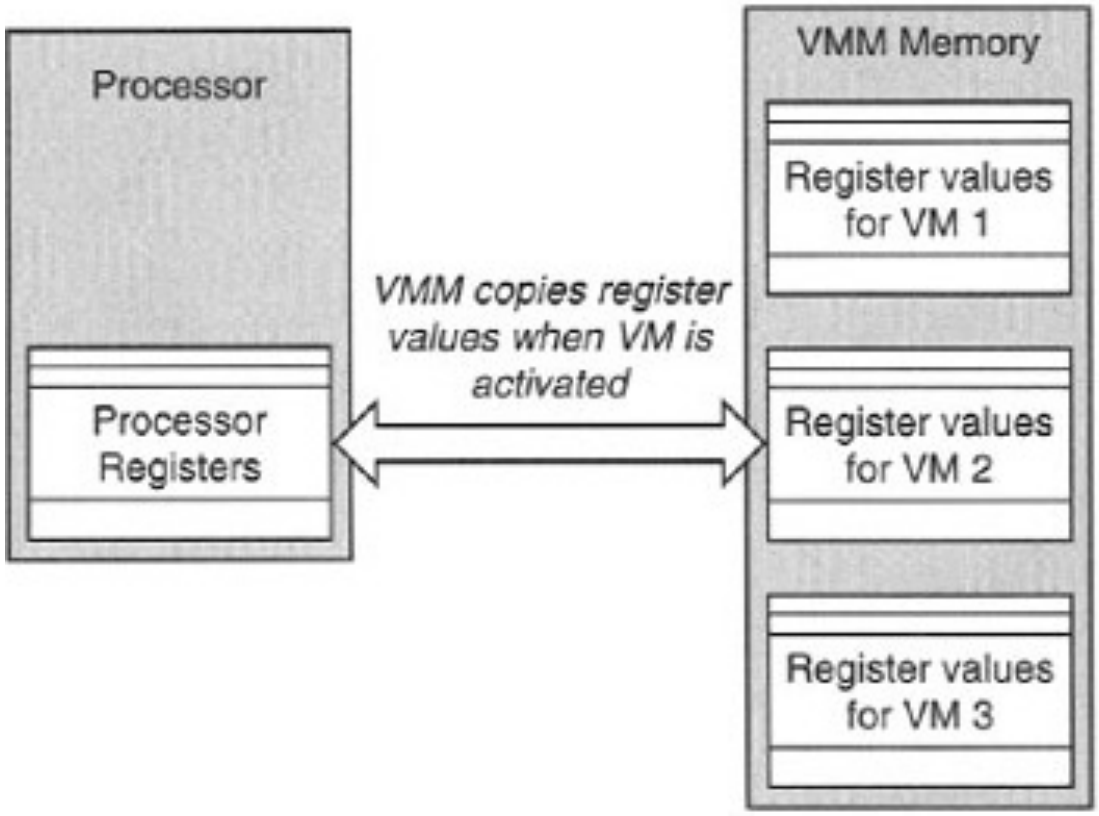
(a)

```
Load register block pointer
to point to VM's registers
in VMM memory

Load program counter to
point to VM program and
start execution
                .
                .
                .
load temp <- reg_pointer, index(A)
store reg_pointer, index(B) <- temp

                .
                .
                .
```

State Copying



(b)

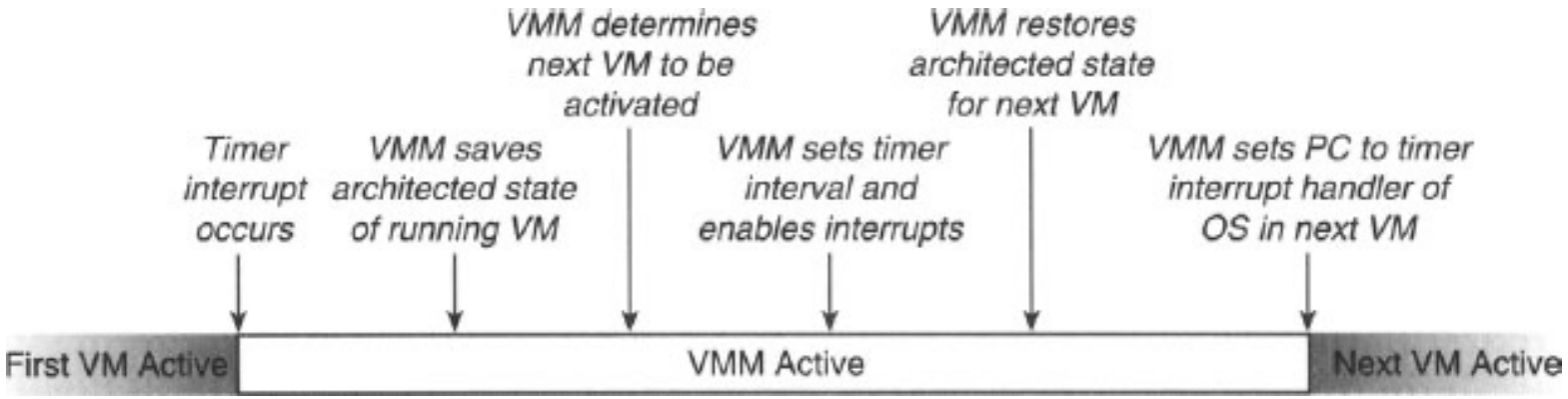
```
Copy register state from
VMM memory

Load program counter to
point to VM program and
start execution
                •
                •
                •
mov reg A -> reg B
                •
                •
                •
Copy register state from
processor back to system
memory
```

Resource Control

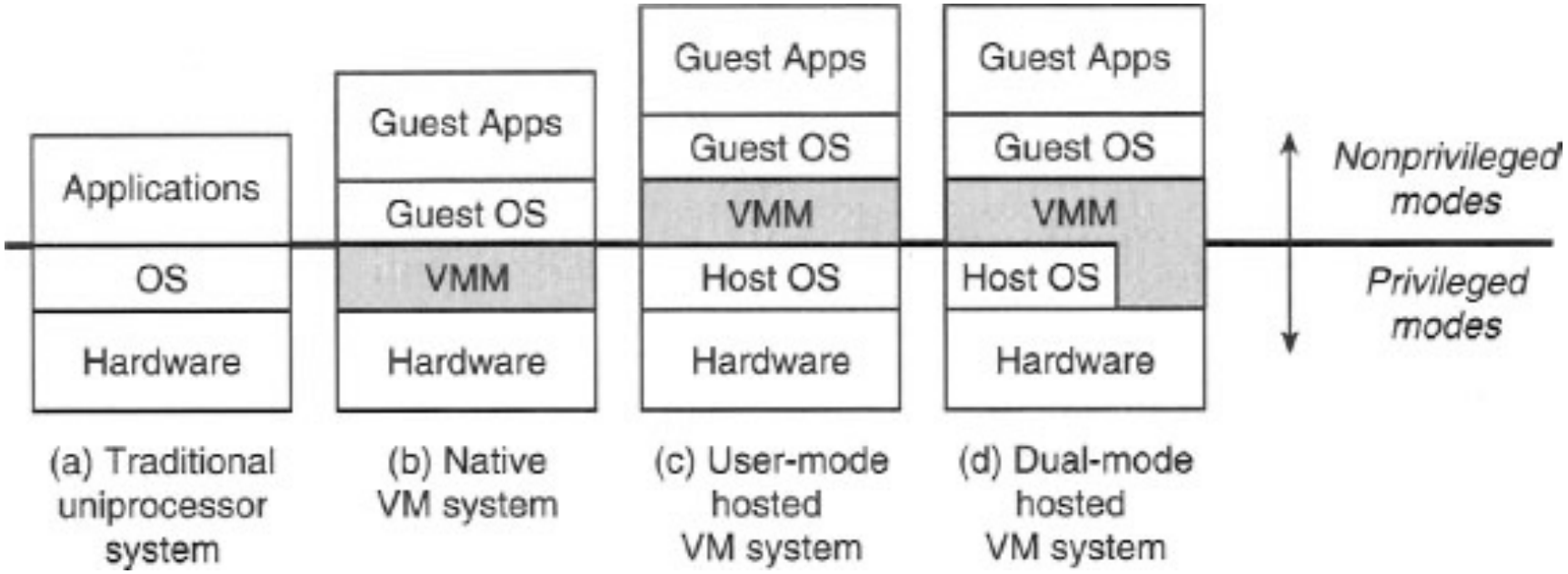
- ถึงแม้ว่าเวลาในการทำ context switching ของการ Copying จะมากกว่าการ Indirection เนื่องจากต้อง copy contexts แต่การทำงานในช่วง Time slice จะมีประสิทธิภาพมากกว่าเพราะประมวลผลบน registers โดยตรงไม่ต้อง Access memory ตลอดเวลา
- VMM จำเป็นต้องใช้ timer interrupt to perform context switching ระหว่าง VMs
- VMM ต้องไม่ให้ Guest OS set ค่า Hardware Timer ได้เองเพราะอาจทำให้เกิดความไม่ fare กับ VM อื่นดังนั้น VMM ต้องจำลอง emulates Virtual Interrupt timer ให้กับแต่ละ Guest

Resource Control



- ในการ switch VM VMM จะกลับมาใช้ CPU ด้วย Timer Interrupt VMM จะเก็บ State ของ VM ก่อนหน้าและเลือกว่าจะนำ VM ใดมารันต่อ
- VMM จะต้องพิจารณาว่าจะ set Timer สำหรับ VM ใหม่อย่างไร และ
  - ถ้ารอบก่อนหน้าของ VM ใหม่ไม่เกิด Virtual Timer Interrupt สำหรับ Guest OS ก็ให้ restore VM state และปล่อยให้ VM ใหม่รันต่อจากเดิม หรือ
  - ไม่เช่นนั้นก็ กำหนดค่า PC ให้ชี้ไปที่ Timer Interrupt handler ของ Guest OS ของ VM ใหม่

Native vs. Hosted VMs



e.g. มี Device Driver ที่ทำให้สามารถรัน Instructions ได้เร็วขึ้น

Native vs. Hosted VM

- แบบ Native (หรือ Classic) VMM กำหนดให้ CPU รัน VM ‘s Instruction จาก Memory โดยตรง
- แบบ Hosted VM VMM ต้อง scan VM’s instructions เพื่อ Interpret หรือ Translate หรือทั้งสองอย่าง เพื่อประมวลผล VM
- แบบ Dual-Mode VMM สามารถส่ง VM’s instructions ผ่าน Special Device หรือ Host OS kernel extension เพื่อให้ CPU รัน VM’s Instructions ส่วนใหญ่ได้โดยตรง

IBM system 360 and 370

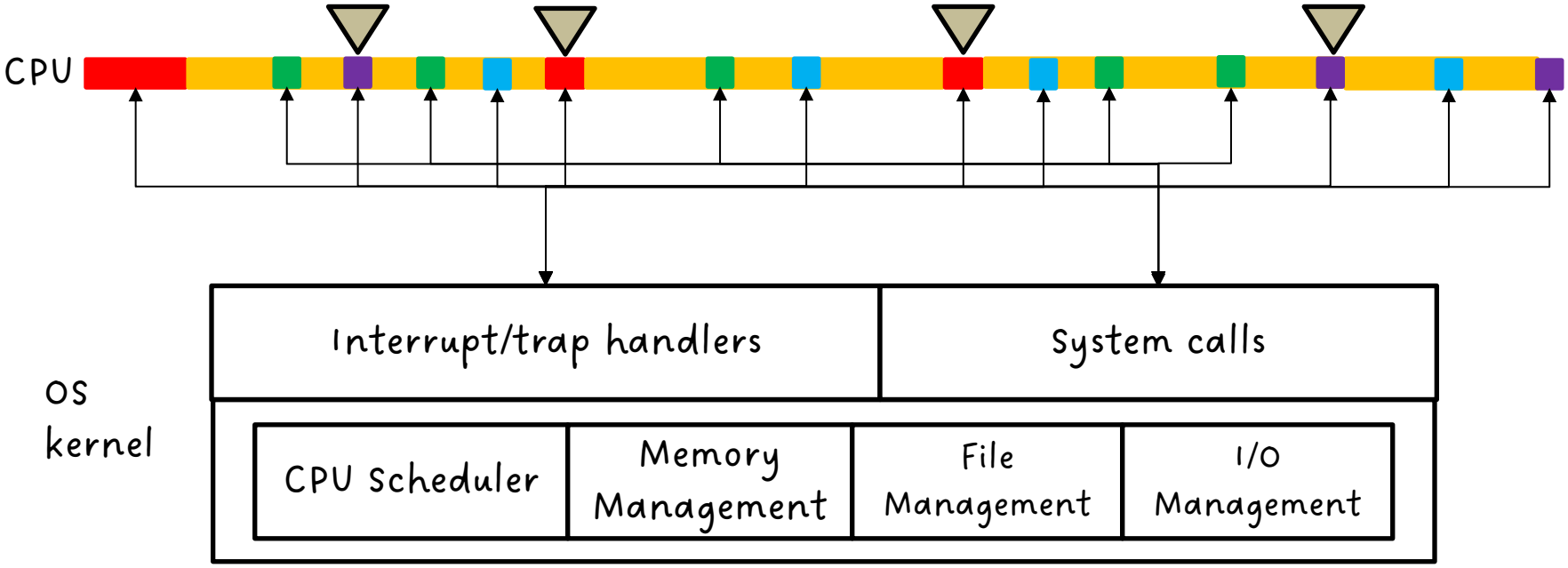
- Popek และ Goldberg ได้เขียน paper “Formal Requirements for Virtualizable Third Generation Architectures” ซึ่งถูกใช้เป็นแนวทางออกแบบ ISA เพื่อสนับสนุน Virtualization
- ในปี 1965 IBM สร้างระบบ Native VMM เพื่อให้บริการ ISA แทนที่จะเป็น ABI ใน IBM System 360 model 40 เนื่องจาก ISA ทำให้มีการ Isolation ระหว่าง VM มากกว่า และ
- VMM ทำให้ IBM 360 model 40 มีความ portability คือสามารถรัน software ในตระกูล 360 ที่ถูกพัฒนาขึ้นก่อนหน้าได้
- ใน IBM 370 IBM พัฒนา Native VMM เรียกว่า Control Program (CP) และพัฒนา single-user Guest OS ชื่อว่า Conversational Monitor System (CMS) เพื่อรันบน CP.
- IBM พัฒนา Hardware supports สำหรับ virtualization ใน IBM 370.

ทบทวนการทำงานของ ระบบคอมพิวเตอร์แบบ Multiprogramming: OS, CPU, Interrupts

OS คือ Event-Driven Program

- ผู้ออกแบบซีพียูและระบบคอมพิวเตอร์ มีนิยามของรายการของเหตุการณ์ที่สามารถเกิดขึ้นได้และตรวจจับเพื่อตอบสนองได้ในระหว่างที่ระบบคอมพิวเตอร์ฮาร์ดแวร์และซอฟต์แวร์ประมวลผล **อย่างชัดเจน** จำนวนหนึ่ง
  - แต่ละเหตุการณ์มีหมายเลขกำหนดอย่างชัดเจน
- โอแสเคอร์เนิล ประกอบไปด้วย code สำหรับจัดการเหตุการณ์เหล่านั้น เรียกว่า Interrupt/Trap handlers และ code อื่นที่ handler เรียกใช้
- เมื่อเกิดเหตุการณ์ขึ้น จะมีคนตะโกนแจ้ง (ส่งสัญญาณ Interrupts) ซีพียู และซีพียูจะหันมาให้ความสนใจ

OS kernel Activities



User program execution

Timer Interrupt handler/CPU Scheduling

I/O interrupt handlers

External Causes

System Calls

Trap handler e.g. page faults, arith errors, etc.

Internal Causes

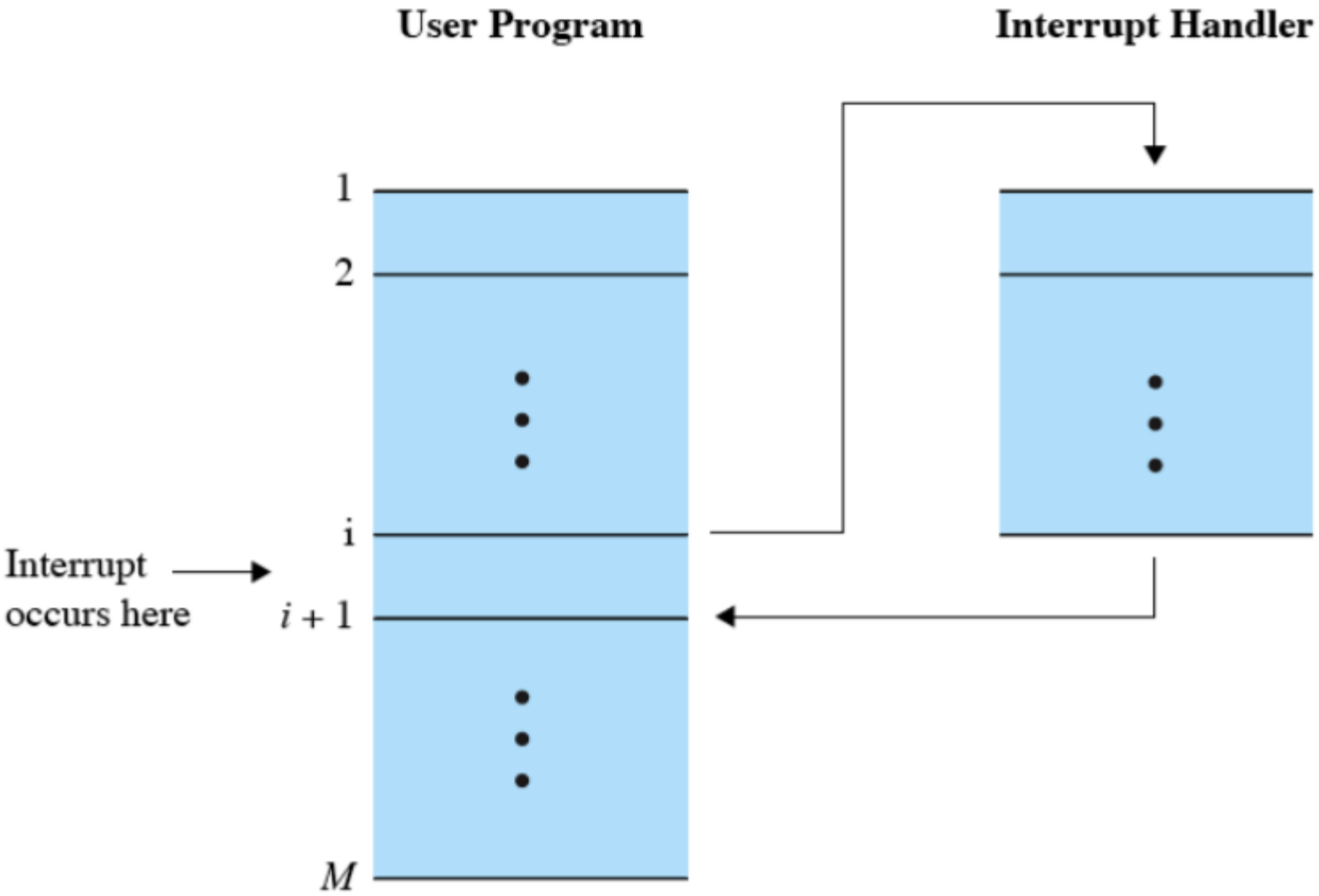
Context Switching เกิดขึ้น เฉพาะเมื่อมีการเปลี่ยน Process ที่ใช้งาน CPU

Interrupt signals

- คือนามธรรมของสัญญานที่เกิดขึ้นในระบบคอมพิวเตอร์ที่ทำให้เกิดการขัดจังหวะการทำงานของ CPU เพื่อให้ CPU ไป execute routine เรียกว่า Interrupt handler ที่ OS เตรียมไว้เพื่อจัดการกับสาเหตุของสัญานเหล่านั้น
- ถ้ามองในอีกมุมหนึ่งสัญญานเหล่านี้เป็นวิธีการที่ทำให้ CPU กลับใช้บริการของ OS
  - เนื่องจาก OS จะ Assign ให้ CPU ไปทำงานให้ Process ต่างๆที่มีอยู่ในระบบ เมื่อมีเหตุการที่จำเป็นต้องให้ OS เข้ามาจัดการ

Interrupt handler

- เมื่อเกิด Interrupt ขึ้นในขณะที่โปรแกรม P กำลัง execute อยู่การทำงานของโปรแกรม P จะถูกขัดจังหวะโดยที่ CPU จะเปลี่ยนไป execute Interrupt Handler
- Interrupt Handler จะ save Context ของ P ใน Process Control Block (PCB) ซึ่งเป็น Data Structure อยู่ใน Memory ของ Kernel ที่ OS ใช้เก็บข้อมูลที่เป็นตัวแทนของ Process P
- เมื่อ Handler ทำงานเสร็จก็จะส่งการทำงานให้ OS Kernel ซึ่งจะพิจารณาว่าจะรัน Process P ต่อหรือเลือก Process อื่นเข้ามาใช้ CPU
- ในกรณีที่ OS เลือกให้ P รันต่อก็จะ load Context ของ P จาก PCB เข้าสู่ Registers ต่างๆ ของ CPU และให้ P รันต่อดังในภาพ



Reference: William Stalling, Operating Systems: Internal and Design Principles

Instruction Cycle With Interrupts

- ในการ Implement การขัดจังหวะดังกล่าว จะเกิดขึ้นใน CPU โดยที่ CPU จะได้รับการออกแบบให้เช็ค Flag ใน Register ชนิดหนึ่งของ CPU ว่ามี Interrupt เกิดขึ้นหรือไม่ก่อนที่จะ fetch คำสั่งใหม่ของ Process P ที่ใช้ CPU อยู่ในปัจจุบันเข้ามา Execute
- ถ้ามี Interrupt Flag ก็ให้ CPU ไป fetch คำสั่งของ Interrupt Handler มา Execute เป็นคำสั่งถัดไป
- แต่ถ้าไม่มีก็ให้ fetch คำสั่งถัดไปของ P เข้ามา Execute
- ดังภาพถัดไป

Instruction Cycle with Interrupts

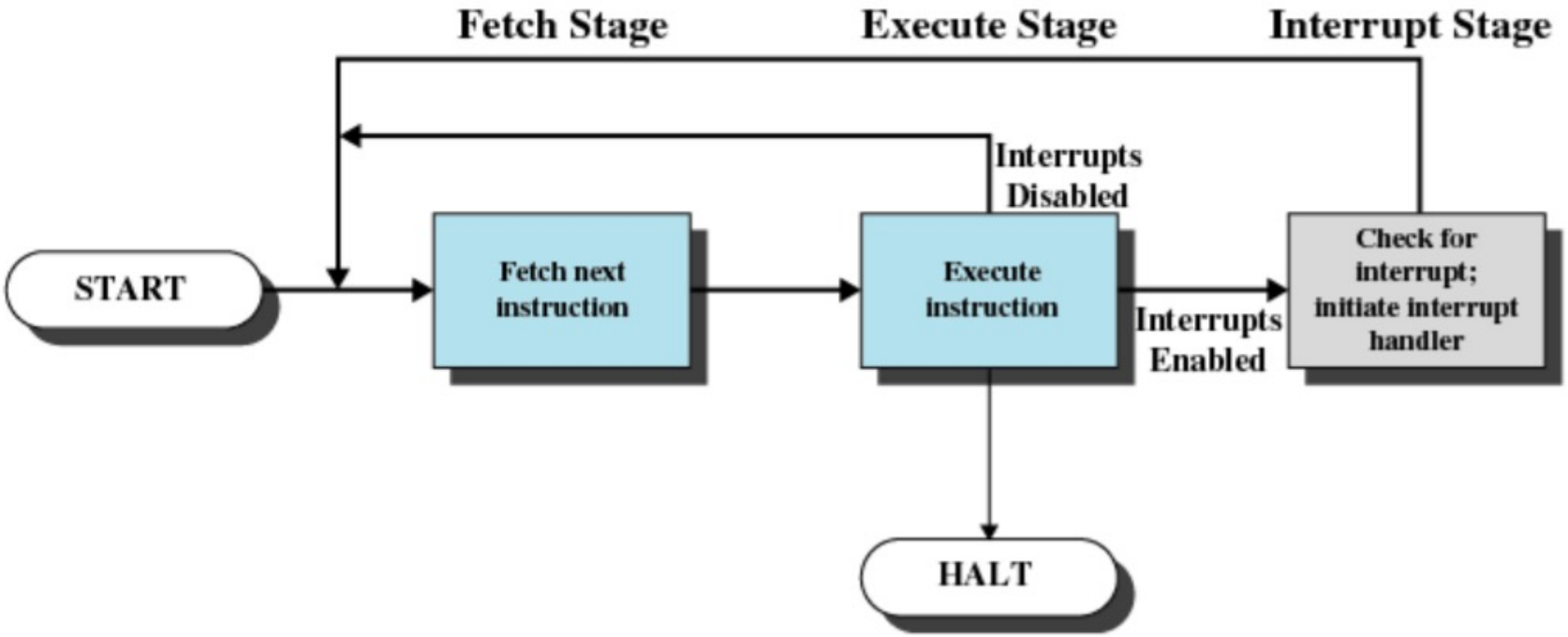


Figure 1.7 Instruction Cycle with Interrupts

Reference: William Stalling, Operating Systems: Internal and Design Principles

Hardware v.s. Software Interrupts

- ในมิติของแหล่งกำเนิดเราแบ่งอินเตอร์รับเป็น 2 กลุ่มได้แก่
- Hardware Interrupts คือสัญญาน Interrupt ที่กำเนิดขึ้นจากอุปกรณ์ Hardware ในระบบคอมพิวเตอร์ เมื่อมีเหตุการณ์สำคัญเกิดขึ้น สัญญานดังกล่าวจะถูกส่งผ่านระบบ Bus ในระบบคอมพิวเตอร์(ดังภาพถัดไป) มายัง CPU
- Software Interrupts คือสัญญาน Interrupt ที่**เกิดขึ้นจากการรันคำสั่ง**เช่น คำสั่ง System Call หรือ**เกิดจากความผิดพลาดในการประมวลผล** (ส่งผลให้เกิดการเปลี่ยน Privilege Level ของ CPU เพื่อรัน Routines หรือ Handler ของ OS kernel)
- จะอธิบายต่อไปใน Slide เรื่อง Interrupt v.s. Trap

Hardware Interrupts

- Hardware Interrupts แบ่งออกเป็น 4 ประเภทได้แก่
  - Maskable Interrupt (IRQ) คือ Interrupt ที่โปรแกรมให้สนใจหรือไม่สนใจได้
  - Non-Maskable Interrupt (NMI) คือ Interrupt ที่ไม่สามารถละเลยได้ ได้แก่ สัญญานที่ hardware แจ้งเมื่อมีเหตุฉุกเฉินเช่น Chipset Errors, Memory Errors (ECC Error), Data Corruption บน Bus, หรือการกดปุ่ม reset เป็นต้น ส่วนใหญ่ระบบจะ Hang หรือเข้าสู่ Recovery Mode
  - Inter-Processor Interrupts (IPI) คือการส่ง Interrupts ระหว่าง Processors ใน Multi-Processors system
  - Spurious Interrupts คือ Interrupt ที่เกิดจากการออกแบบ Hardware ที่ผิดพลาด หรือการมีสัญญานไฟฟ้ารบกวน Interrupt line ในเครื่อง

Interrupt Controller

- Hardware ที่ทำหน้าที่จัดการสัณญาณอินเตอร์รับมีสองชนิด
- Local Advanced Programmable Interrupt Controller (LAPIC) เป็นอุปกรณ์จัดการสัญญาณอินเตอร์รับสำหรับแต่ละซีพียูคอร์ ส่ง timer interrupt ให้โลคอลซีพียูคอร์ และส่ง IPI ให้ LAPIC ของซีพียูคอร์อื่น
- I/O Advanced Programmable Interrupt Controller (IOAPIC) เป็นอุปกรณ์จัดการสัญญาณอินเตอร์รับจากไอโอสับซิสเต็ม (I/O subsystem) และส่งต่อให้ซีพียูคอร์ที่เป็นผู้รับ

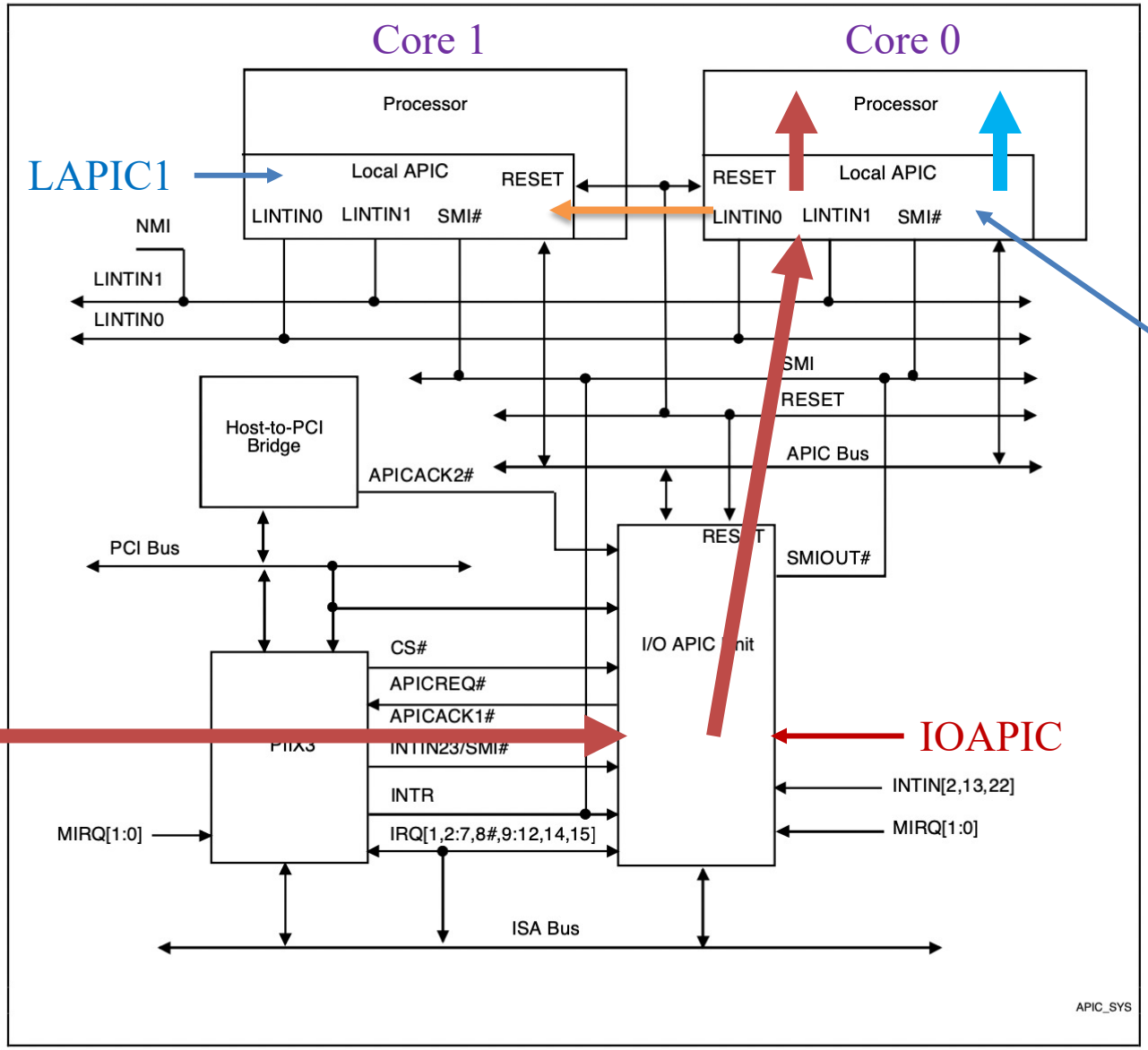


Figure 2. I/O And Local APIC Units

Maskable Interrupt Signals

- Maskable Interrupt (IRQ) คือสัญญาน Interrupt ที่ Programmer สามารถเขียนโปรแกรมให้ OS หรือ Process เลือกที่จะสนใจหรือไม่สนใจได้
  - เนื่องจากในระหว่างที่ OS หรือ Process รัน อาจมี Interrupt เกิดขึ้นมากมาย ดังนั้นจึงมีความจำเป็นต้องสามารถออกคำสั่งให้มีการจัดการ Interrupt เหล่านั้นในอันดับและเวลาที่เหมาะสม
  - ตัวอย่างเช่น ในสถาปัตยกรรม x86 ดั้งเดิมมี IRQ 0 – 15 ดัง Slides ถัดไป
- มี Hardware ควบคุมได้แก่ Programmable และ Advanced Programmable Interrupt Controller (PIC และ APIC)

Interrupt Requests (IRQ)

- **บนเครื่อง IRQ พื้นฐานได้แก่**

IRQ 0 – system timer
IRQ 1 – keyboard controller
IRQ 2 – cascaded signals from IRQs 8–15
IRQ 3 – serial port 2
IRQ 4 – serial port 1
IRQ 5 – parallel port 2 and 3 or sound card
IRQ 6 – floppy disk controller
IRQ 7 – parallel port 1.

IRQ 8 – real-time clock (RTC)
IRQ 9 – Advanced Configuration and Power Interface
IRQ 10 – open interrupt, SCSI or NIC
IRQ 11 – open interrupt, SCSI or NIC
IRQ 12 – mouse on PS/2 connector
IRQ 13 – CPU co-processor or integrated floating point unit or inter-processor interrupt
IRQ 14 – primary ATA channel (ATA interface usually serves hard disk drives and CD drives)
IRQ 15 – secondary ATA channel

Traps v.s. Interrupts

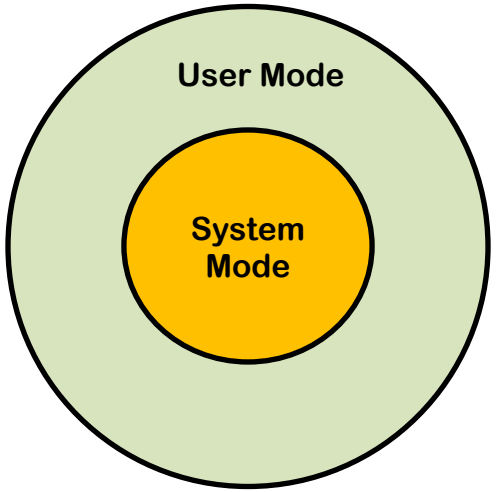
- ในมิติของความสัมพันธ์กับยูเซอร์โปรแกรม เราแบ่งอินเตอร์รับเป็น 2 กลุ่ม
- Trap คือการที่ CPU ต้องเปลี่ยนการประมวลผลจากชุดคำสั่งของ user โปรแกรมที่กำลังประมวลผลในปัจจุบันไปประมวลผลชุดคำสั่งของ OS เพื่อตอบสนองกับ เหตุการณ์พิเศษที่เกิดขึ้นอันเป็นผลข้างเคียงของการประมวลผลคำสั่งของ user โปรแกรมนั้น
  - มักจะเกิดขึ้นเมื่อมี exception หรือ error ในการรันคำสั่งของโปรแกรม
  - User program สามารถเรียก system call เพื่อทำให้เกิด trap ขึ้นได้
  - E.g. Arith overflow, page faults, violation of privilege level, violation of mem prot., illegal opcode
- Interrupt เกิดจากเหตุการณ์ที่เกิดขึ้นบนระบบคอมพิวเตอร์ภายนอกการประมวลผล User โปแกรม
  - E.g. I/O interrupt and timer interrupts

Examples of Trap

- เป็นเหตุการแบบ Synchronous คือ Trap เกิดขึ้นเพราะการรันคำสั่งใดคำสั่งหนึ่งของ Process และคำสั่งนั้นต้องรอรับผลจากการเกิด Trap
- System Call: โปรแกรมรันคำสั่งพิเศษเช่น SYSCALL/SYSRET หรือ INT (x86) เพื่อเปลี่ยน Privilege เป็น System mode เพื่อใช้บริการ services ต่างๆของ OS เพื่อบริหารจัดการทรัพยากรของคอมพิวเตอร์
- Arithmetic Overflow: การเพิ่มหรือลดค่าของตัวเลขมากกว่าที่ตัวแปรชนิดที่ใช้อยู่จะแทนค่าได้ หรือค่าที่มากกว่าที่การประมวลผลตัวเลขจะทำได้
- Arithmetic Error: การหารเลขด้วย 0
- Page Fault: Memory page ที่ต้องการใช้งานไม่อยู่ใน Physical Memory
- Violation of CPU Privilege Levels: การรัน System ISA ในขณะที่ CPU ประมวลผลใน User Mode
- Violation of Memory Protection: การพยายามเปลี่ยนแปลงค่าใน Memory ส่วนที่ไม่ได้รับอนุญาตให้เปลี่ยนแปลงค่าเช่น Code Segment
- Illegal Opcode: การส่งรหัสคำสั่ง binary ที่ไม่ใช่ชุดคำสั่งที่ถูกต้องให้ CPU

Examples of Interrupts

- มักจะเป็นเหตุการณ์ที่เกิดแบบ Asynchronous คือสามารถเกิดเมื่อไรก็ได้ในระหว่างที่ Process กำลังประมวลผลคำสั่งใดๆก็ได้
- System Timer Interrupt: เมื่อ Timer นับถอยหลังถึง 0
- I/O devices: เมื่อมีการกด Keyboard, เคลื่อน Mouse,
- Disk Interrupts: ส่งสัญญานเมื่อการอ่านข้อมูลจาก Disk หรือเขียนข้อมูลลง Disk เสร็จสิ้น
- Network Interrupts: ส่งสัญญานเมื่อการรับข้อมูลจาก Network หรือส่งข้อมูลสู่ Network เสร็จสิ้น
- NMI Interrupts ที่ยกตัวอย่างไปก่อนหน้า



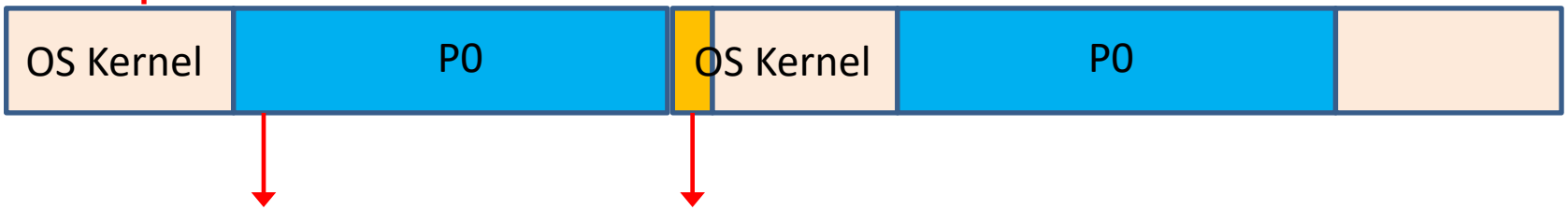
CPU Privilege Levels

- เพื่อป้องกันไม่ให้ยูเซอร์โปรแกรมสามารถจัดสรรทรัพยากรได้ตามใจ จึงได้มีการกำหนดให้ซีพียูมี Privilege Levels โดยมีข้อกำหนดว่า ซีพียูจะรันคำสั่ง System ISA (เช่นกำหนดค่า clock timer) ได้เมื่อรันในซิสเต็มโหมดเท่านั้น
- โอเอสเริ่มต้นในซิสเต็มโหมด และเมื่อจะรันยูเซอร์โปรแกรมจะเปลี่ยนโหมดเป็นยูเซอร์โหมดแล้วจึงรันยูเซอร์โปรแกรม
- เมื่อเกิดอินเตอร์รับหรือแทรบซีพียูจะเปลี่ยนโหมดเป็นซิสเต็มเพื่อรันอินเตอรับแฮนด์เลอร์ และโอเอสจะเปลี่ยนโหมดเมื่อเลือกยูเซอร์โปรแซสมาใช้ซีพียู
- ถ้ายูเซอร์โปรแกรมใช้ System ISA ในยูเซอร์โหมด จะเกิด Trap ขึ้น

Review: Multi-programming CPU Execution

ทุกอย่างรันโดย CPU
CPU ทำงานในปัจจุบัน และทำทีละคำสั่ง

1. ก่อนจะรัน P0 OS kernel สร้าง PCB ของ P0 และ load P0 เข้าสู่ memory
2. Kernel กำหนดให้ Program Counter ชี้ไปที่ Addressเริ่มต้นของ P0
3. OS kernel set Timer Interrupt ให้ = Time Slice (eg. 1000 ns)
   hardware จะ count down จนถึง 0



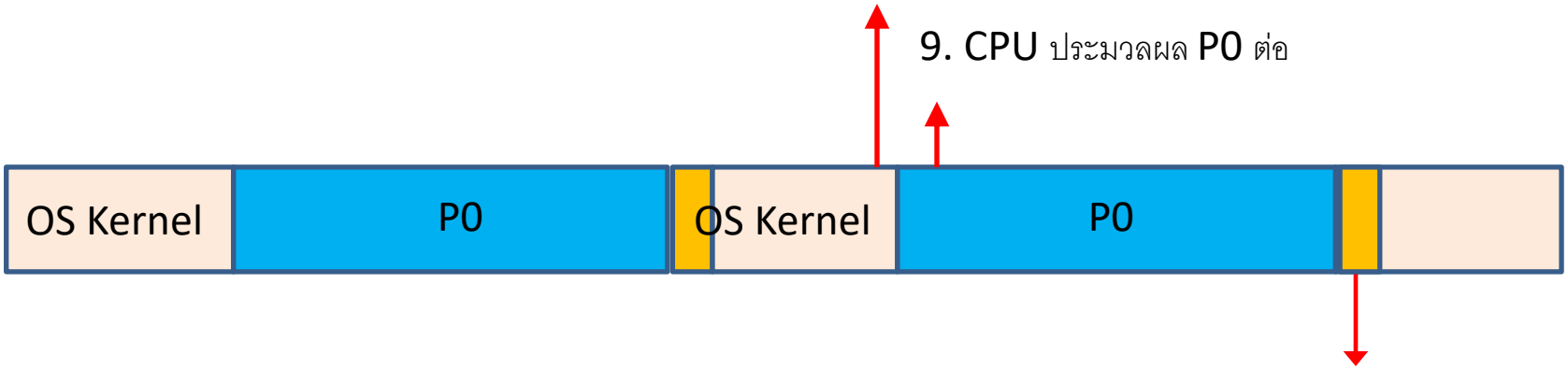
4. CPU ประมวลผล P0
5. Hardware ปล่อย CPU Timer Interrupt ขัดจังหวะการทำงานของ P0 และรัน Interrupt Handler (ส่วนหนึ่งของ OS kernel)
6. Interrupt Handler save context ของ P0 และ รับ code ของ Hander และรัน OS ให้ OS พิจารณาว่าจะทำอะไรต่อ

Review: Multi-programming CPU Execution

6. OS kernel เลือกจะรัน P0 และ copy CPU Context ของ P0 จาก memory มา CPU

7. ซึ่งจะทำให้ Program Counter Register ใน CPU เปลี่ยนชี้ไปที่ Address ล่าสุดของ P0

8. OS kernel set Timer Interrupt ให้ = Time Slice (eg. 1000 ns) hardware จะ count down จนถึง 0



10. Hardware ปล่อย CPU Timer Interrupt ขัดจังหวะการทำงานของ P0 และรัน Interrupt Handler (ส่วนหนึ่งของ OS kernel)

11. Interrupt Handler save context ของ P0 และ รับ code ของ Hander และรัน OS ให้ OS พิจารณาว่าจะทำอะไรต่อ

จะเป็นอย่างไรถ้า User Process Set Timer Interrupt ได้ ???

CPU Virtualization

Processor Virtualization

- How to execute VM's Instruction?
  - โดยเฉพาะอย่างยิ่ง Guest OS instructions
- Emulation VS. Direct native execution
- Emulation คือการแปลงคำสั่ง ทำได้สองวิธีได้แก่
  - Interpretation: VMM อ่านทีละคำสั่งและประมวลผลทีละคำสั่ง
  - Binary translation: VMM แปลงคำสั่งทีละ block ให้เป็นภาษาเครื่องของ Host หรือเป็นคำสั่งที่รันบน Host OS ได้ แล้วจึงรัน block นั้น
- Emulation มักจะใช้สำหรับ VM ที่ ISA แตกต่างจาก Host ISA

Processor Virtualization

- Emulation มีประสิทธิภาพต่ำ
  - โดยเฉพาะอย่างยิ่งในการ scan คำสั่งและประมวลผลทีละคำสั่งในระดับ High-level หรือแม้กระทั่งในเรื่องของการแปลงคำสั่งทีละ block
- Direct native execution มักใช้กับ VM กับ Host มี same ISA.
- Direct native execution ใช้วิธีส่งคำสั่งไปรันบน CPU โดยตรงและใช้ Trap และ Trap Handler เพื่อจัดการคำสั่งที่มีปัญหา โดยให้ VMM แปลและรันคำสั่งเหบ่านั้นให้
  - เหมือนกับทุ่มงานให้ CPU รันไปก่อน งานที่มีปัญหาค่อยแก้เอาเป็นรายๆไป ถ้าคำสั่งหรืองานที่มีปัญหามีเปอร์เซนต์น้อย ประสิทธิภาพโดยรวมก็จะดี

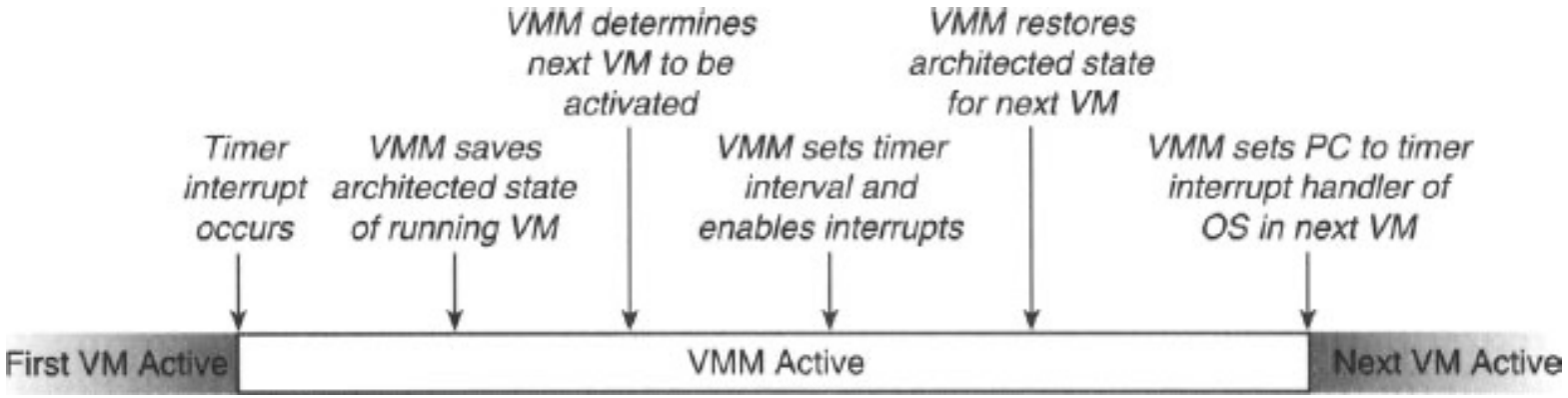
Conditions for ISA Virtualization

- ในระบบแบบ Native VMM นั้น VMM เป็นเหมือนกับ OS ซึ่ง CPUประมวลผล VMM ใน system mode และ CPU ประมวลผล Guest OS ใน user mode.
- ปัญหาเกิดขึ้นเพราะ Guest OS คือโปรแกรมที่ประกอบไปด้วยชุดคำสั่งที่หลายๆ คำสั่งเป็น privilege instructions (หรือ system ISA)
- สมมุติฐาน Assumptions:
  - Hardware ประกอบไปด้วย processor และ Memory แบบ Uniformed Memory Access (UMA)
  - Processor ประมวลผลสอง mode ได้แก่ System & User modes
  - CPU รัน Subset ของ ISA (ได้แก่ System ISA) ใน System mode เท่านั้น
  - VMM กำหนดตำแหน่งเริ่มต้นของ Memory ของแต่ละ VM ใน relocation register

Privilege Instructions

- **Privilege Instructions: ถ้า cpu ได้รับคำสั่งแบบนี้มันจะมีพฤติกรรมสองแบบคือ (1) มันจะรันคำสั่งนี้ให้ก็ต่อเมื่อ mode ของ cpu ในขณะนั้นเป็น system mode และ (2) มันจะส่งสัญญาน traps ถ้า mode ของ cpu ในขณะนั้นเป็น user mode ยกตัวอย่างเช่น**
  - **LoadPSW:** load processor status words เป็นคำสั่งที่ใช้ในการเปลี่ยน configuration ต่างๆของ cpu รวมทั้งเปลี่ยนค่า mode ของการทำงานด้วย
  - **Set CPU Timer:** กำหนดค่าระยะเวลา count down ก่อนที่ hardware จะ generate สัญญาน timer interrupt ครั้งถัดไป ใน timer register
- **Non-privilege:** คำสั่งแบบนี้รันได้ในทุก cpu mode

VMM operation

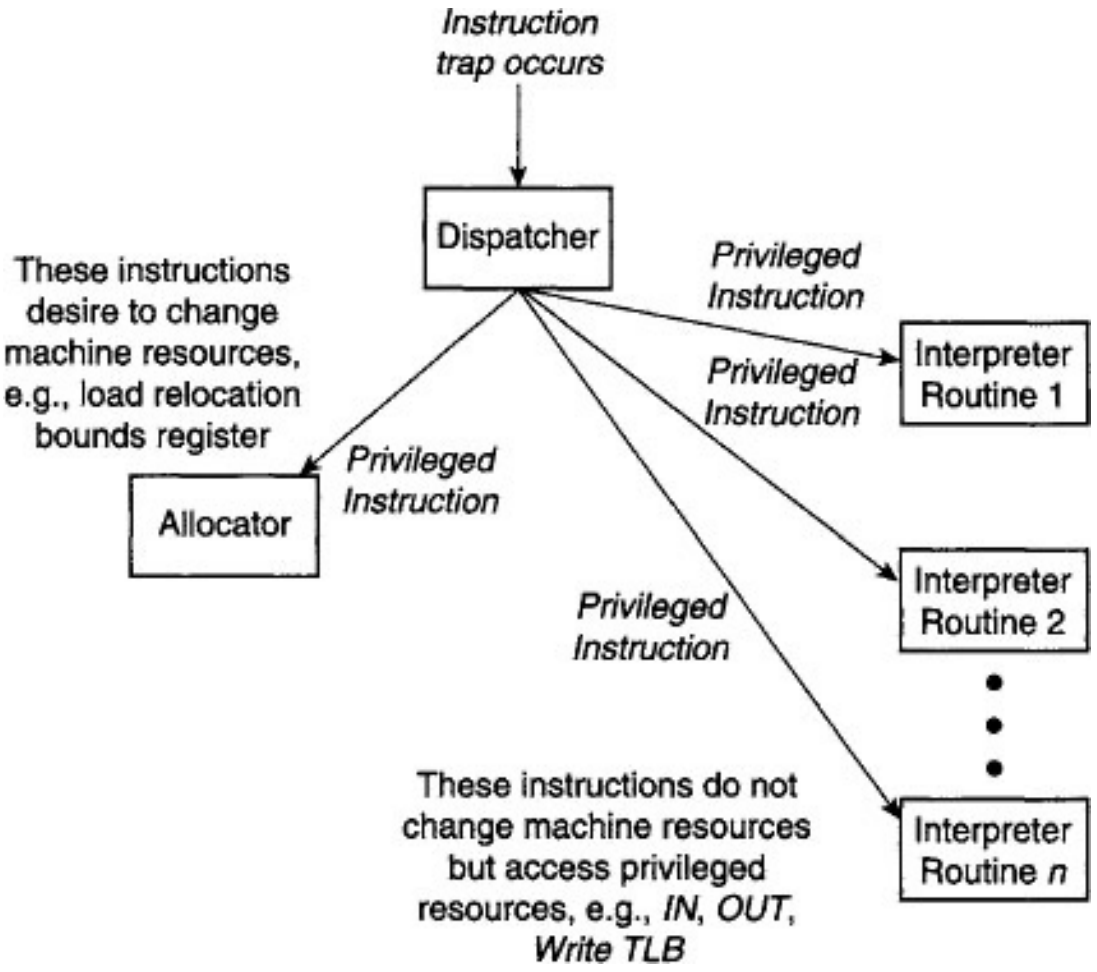


- เมื่อ VMM กำหนดค่า register ต่างๆของ cpu ให้เริ่มทำงานให้ vm หนึ่ง ถ้า vm นั้น รันเป็นครั้งแรก VMM ก็จะหนดค่า PC ของ cpu ให้ชี้ไปที่ timer interrupt handler
- แต่ถ้า vm ที่จะรันใน time slice ถัดไป เป็น vm ที่เคยรันมาแล้วมันก็จะโหลด context ของ vm จาก memory มาที่ cpu และให้ cpu รันต่อ
- Vmm (ทำงานใน system mode) จะเปลี่ยน mode ของ cpu ให้เป็น user mode ก่อนที่จะปล่อยให้ cpu รัน vm นั้น

VMM operation

- CPU จะนำคำสั่งของ vm เข้ามาทำงานทีละคำสั่ง ใน user mode
- แต่ปัญหาเกิดขึ้นเนื่องจากคำสั่งของ vm มีทั้งส่วนที่เป็น OS และส่วนที่เป็น Apps ซึ่งในส่วนที่เป็น OS อาจมี privilege instructions อยู่
- เมื่อ privilege instruction ถูกรันใน user mode ก็จะเกิด trap ขึ้น
- CPU ก็จะเปลี่ยนไปรัน Trap handler ของ VMM ทันที และเปลี่ยน mode กลับเป็น system mode โดยอัตโนมัติ
- ผู้ออกแบบ VMM ใช้ trap เป็นเครื่องมือ กล่าวคือ Trap handler จะพิจารณาว่าคำสั่ง privilede instruction ใดทำให้เกิด trap ขึ้นและจะไปรันคำสั่งนั้นให้ แล้วหลังจากนั้นก็จะเปลี่ยน mode ให้เป็น user mode และกำหนด PC ให้ cpu ไปรันคำสั่งของ vm คำสั่งถัดไป ดัง slide ถัดไป

Component of VMM



e.g. Load new page table

Trap Handler of VMM

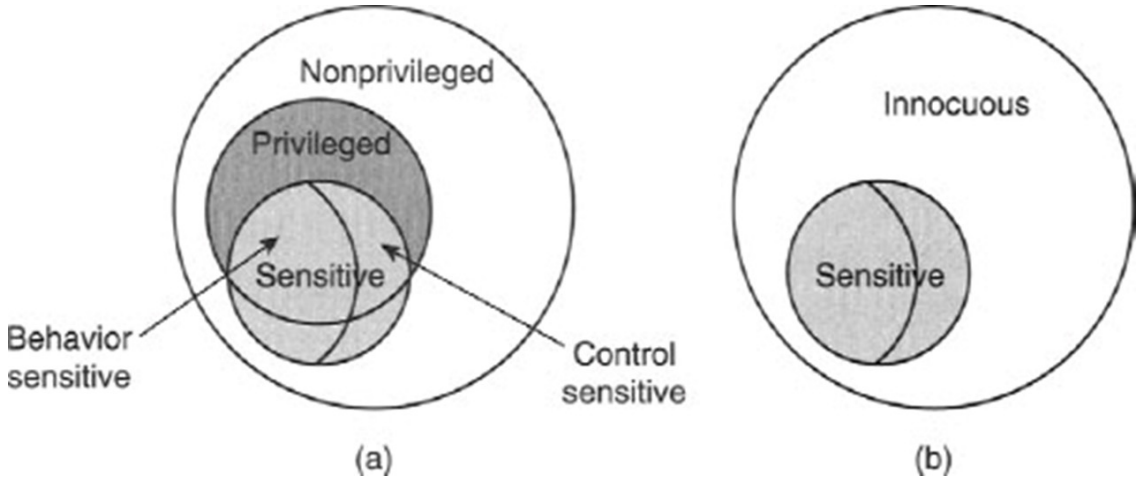
- Dispatcher เป็น code ที่พิจารณาว่า instruction ที่ทำให้เกิด trap คืออะไร และต้องไปเรียก routine ใดรันคำสั่งนั้นให้
- คำสั่ง privilege instruction ที่ trap handler ของ VMM จะรันให้มีสองแบบ
- Allocator: เป็น privilege instruction ที่กำหนดค่าของระบบคอมพิวเตอร์ เช่น load relocation register หรือกำหนดให้ register นี้ชี้ไปยัง page table ของ App process นั่นเอง
- Interpreter routing คือ routine ที่ VMM จะรันให้แทน Guest OS ของ VM เช่นคำสั่ง IN OUT อ่านเขียนข้อมูลจาก I/O ต่างๆ

Critical Instructions

- แต่อุปสรรคคือ กลไก trap ไม่เพียงพอที่จะทำให้ VMM ทำงานแทนคำสั่งของ guest OS ทุกคำสั่งได้อย่างถูกต้อง
- Critical Instruction คือคำสั่งไอเอสเอที่รันได้เมื่อ CPU อยู่ใน system mode และ User Mode ซึ่งให้ผลลัพธ์ต่างกันขึ้นอยู่กับโหมดของซีพียู เช่นคำสั่ง POPF ของ x-86
- VMM ดักจับและรัน privilege instruction ให้ guest os ได้เพราะ cpu รันคำสั่งเหล่านี้ใน user mode
- แต่ไม่สามารถตรวจจับ Critical Instruction ได้ เนื่องจากสามารถรันได้ทั้งใน system mode และ user mode แต่จะมีพฤติกรรมหรือให้ผลลัพธ์ในแต่ละสถานะการณ์ที่แตกต่างกัน

Instruction Comparisons

instructions	Privilege/ Non-privilege	Sensitive/ Innocuous
LoadPSW (LPSW) (IBM)	Privilege	Sensitive (control)
Set CPU Timer (SPT) (IBM)	Privilege	Sensitive (control)
Pop Stack Into Flag (x86)	Non-privilege	Sensitive (behavior)
add (x86)	Non-privilege	Innocuous



Virtualization Requirements

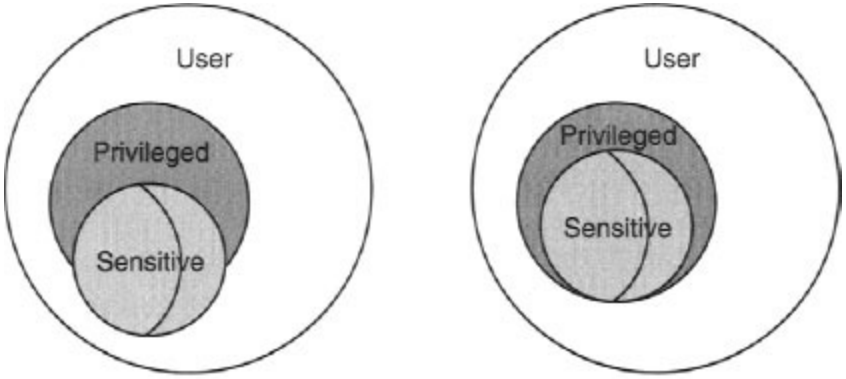
- Popek & Goldberg กำหนดคุณสมบัติของ VMM ไว้ดังนี้
- Efficiency
  - คำสั่ง privilege ทุกคำสั่งจะต้องถูก Trap เมื่อรันในขณะที่ซีพียูประมวลผลใน User Mode
- Resource Control
  - Guest VM ไม่สามารถเปลี่ยน configuration ของ Host ได้

Virtualization Requirements

- Equivalence: OS และ App บน VM จะต้องมีพฤติกรรมการประมวลผลเหมือนรันบน Hardware จริง
- ยกเว้น :
  - VM execution overheads เวลาในการประมวลผลโดยรวมของ App ใดๆ อันจากการประมวลผลของ code ใน VMM ที่แทรกเข้ามาระหว่างการประมวลผลของ VM แต่ละ VM
  - VM resource limitation ข้อจำกัดเช่นขนาดของ Host Memory มีจำกัด การสร้าง VM แต่ละอันต้องใช้ Memory ซึ่งจำกัด
  - VM และ Hardware time lag เนื่องจาก VM อาจถูกขัดจังหวะด้วยการทำงานของ VM อื่น ดังนั้นเวลาระหว่างการ Execute คำสั่งสองคำสั่งของ VM อาจไม่เท่ากันได้ เวลาที่ OS รันบน hardware เวลาระหว่างสองคำสั่งจะเท่ากัน แต่ในกรณีของ Guest OS จะคาดหวังเช่นนั้นไม่ได้

Theorem 1: Efficient VMM

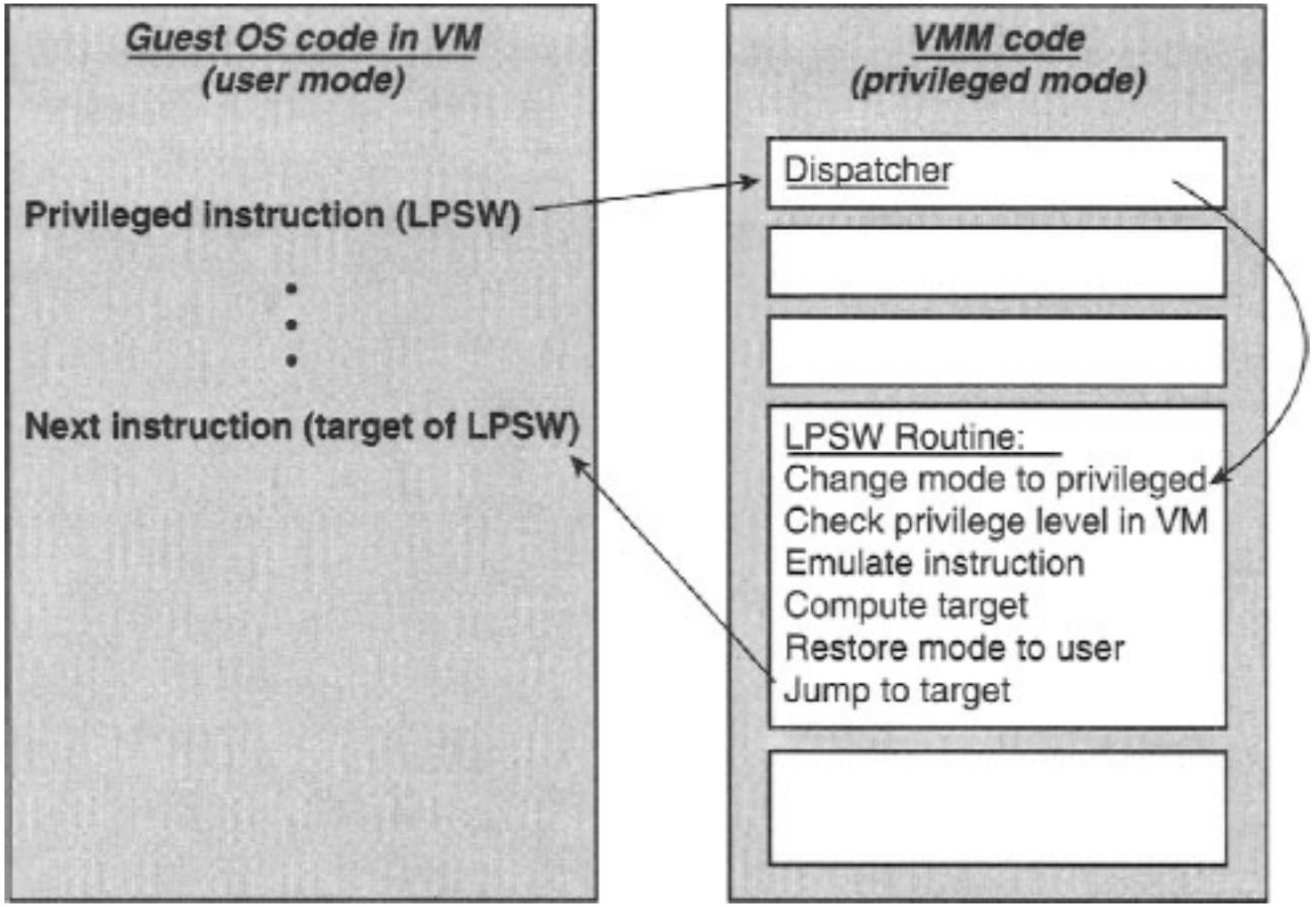
- **Theorem 1**
  - สำหรับ conventional third-generation computer เราสามารถสร้าง VMM ที่มีประสิทธิภาพได้ ถ้าคำสั่ง Set ของ Sensitive Instructions ของ CPU นั้นเป็น Subset ของ Set ของ Privilege Instructions
  - ไม่มีคำสั่งที่เปลี่ยน configuration หรือที่ได้รับผลจากการเปลี่ยน configuration ใดๆของ VM ที่ไม่ถูก trap และจัดการโดย VMM



(a) Does not satisfy condition

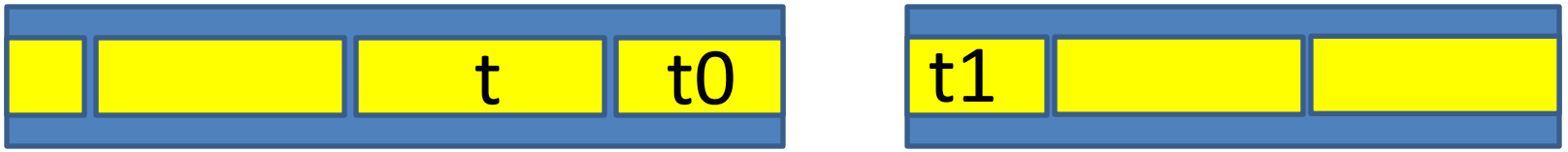
(b) Satisfies condition — efficiently virtualizable

Example 1: Running LPSW



Example 2: Running SPT

- Set timer
  - VMM มี table สำหรับเก็บค่า timer interrupt ของแต่ละ VM
  - VMM กำหนดให้ VM แต่ละเครื่องมี time slice = T และ Guest OS กำหนดให้ time slice ของแต่ละ Process คือ t ซึ่งมีค่า <= T
  - ถ้า Process หนึ่งรันบน VM ไปได้ t0 แล้ว VM หมด Time Slice เมื่อ VMM resume VM นั้นเข้ามาใช้งาน CPU อีกครั้ง Process นั้นก็จะทำงานต่อในเวลาที่เหลืออยู่ (t1 ในภาพ)

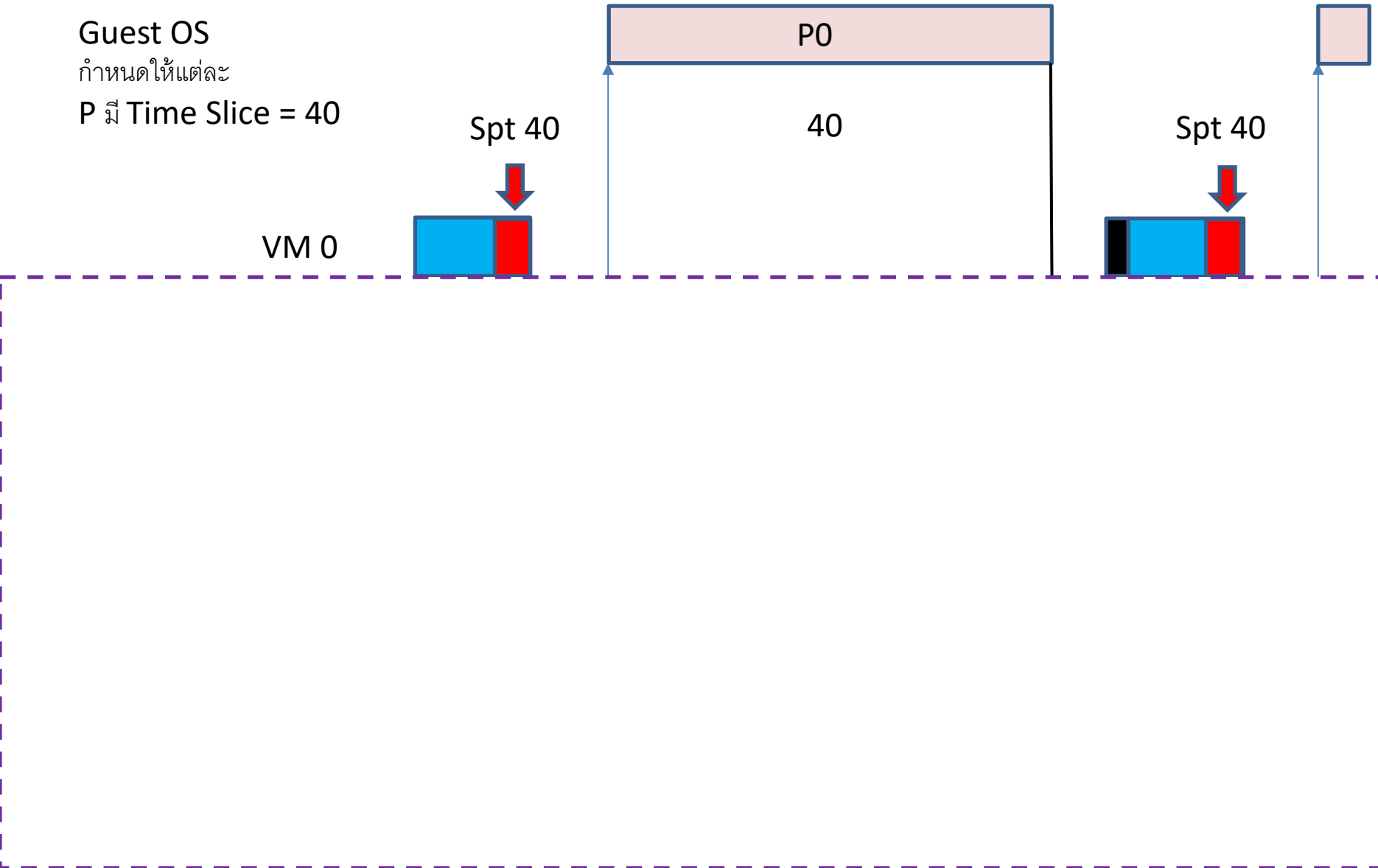


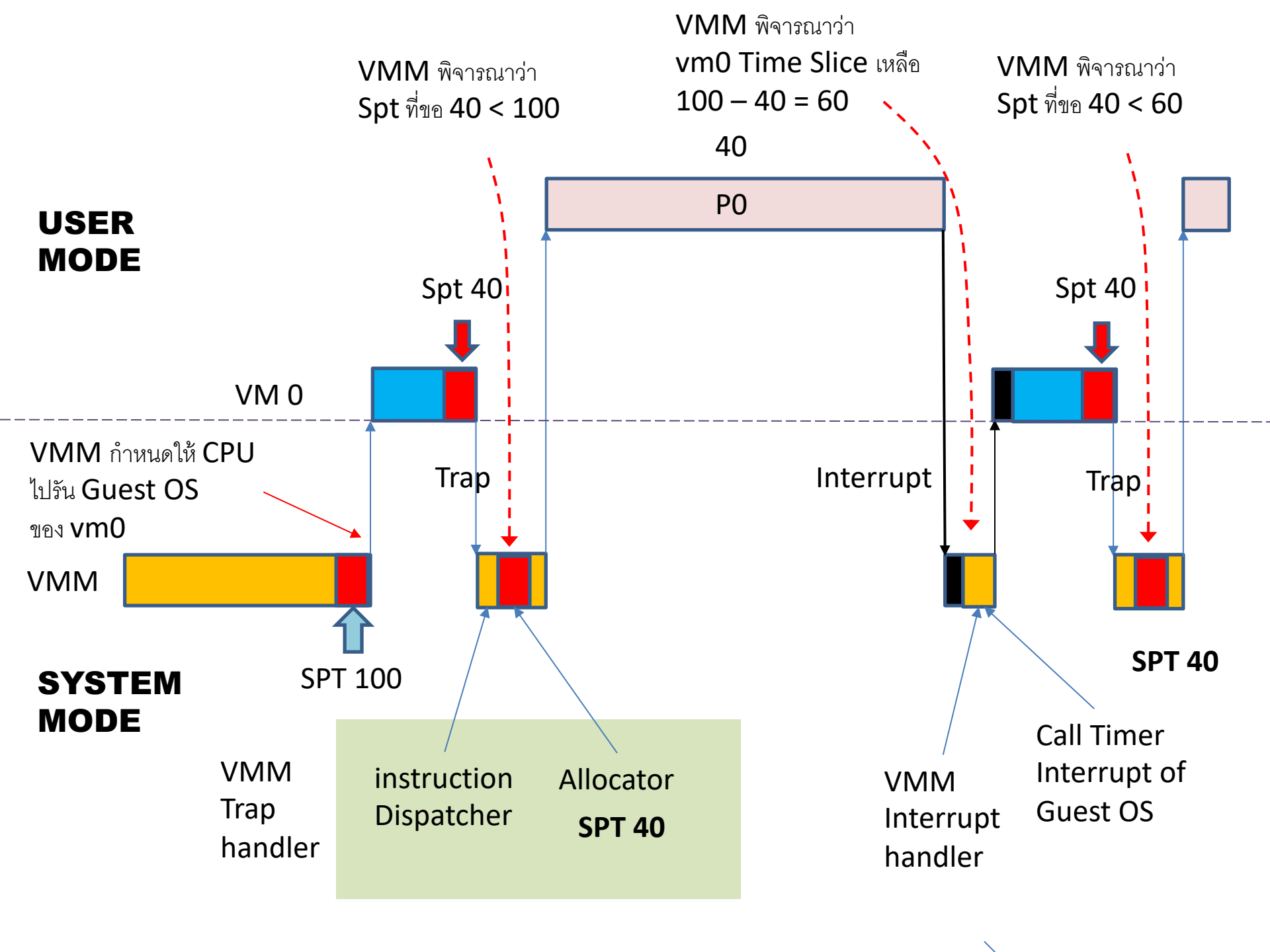
t <= T t = t0 + t1

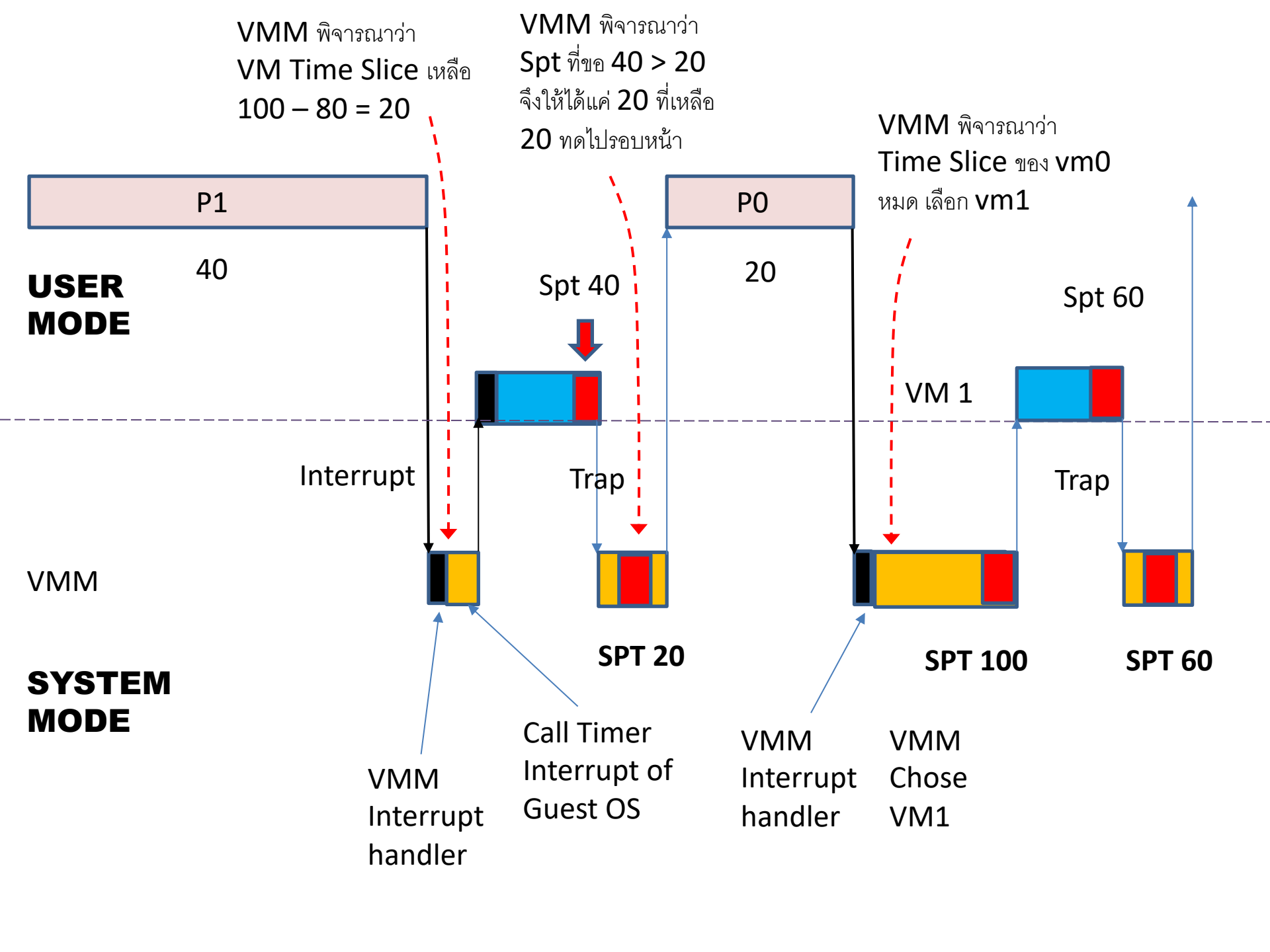
Guest OS

กำหนดให้แต่ละ

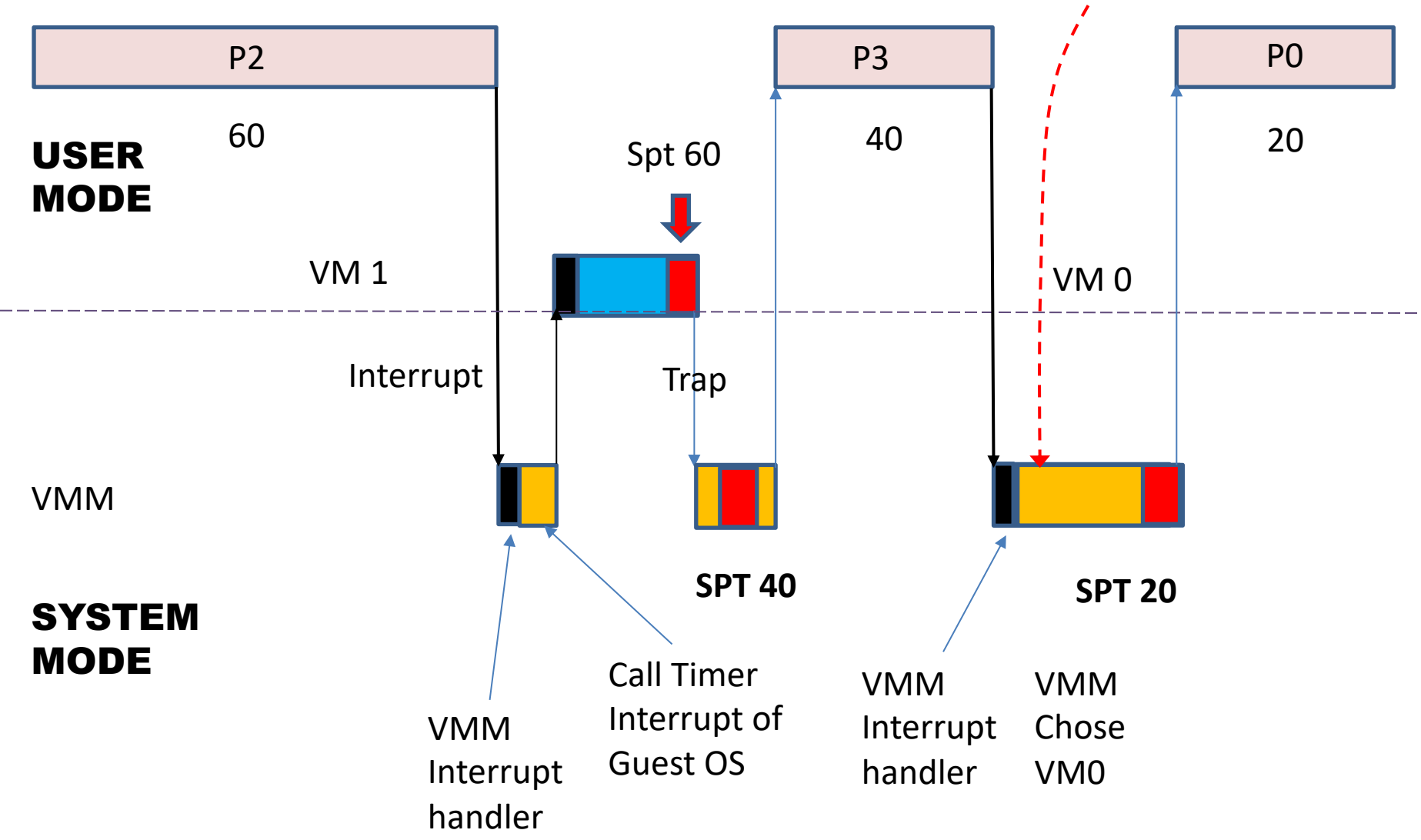
P มี Time Slice = 40

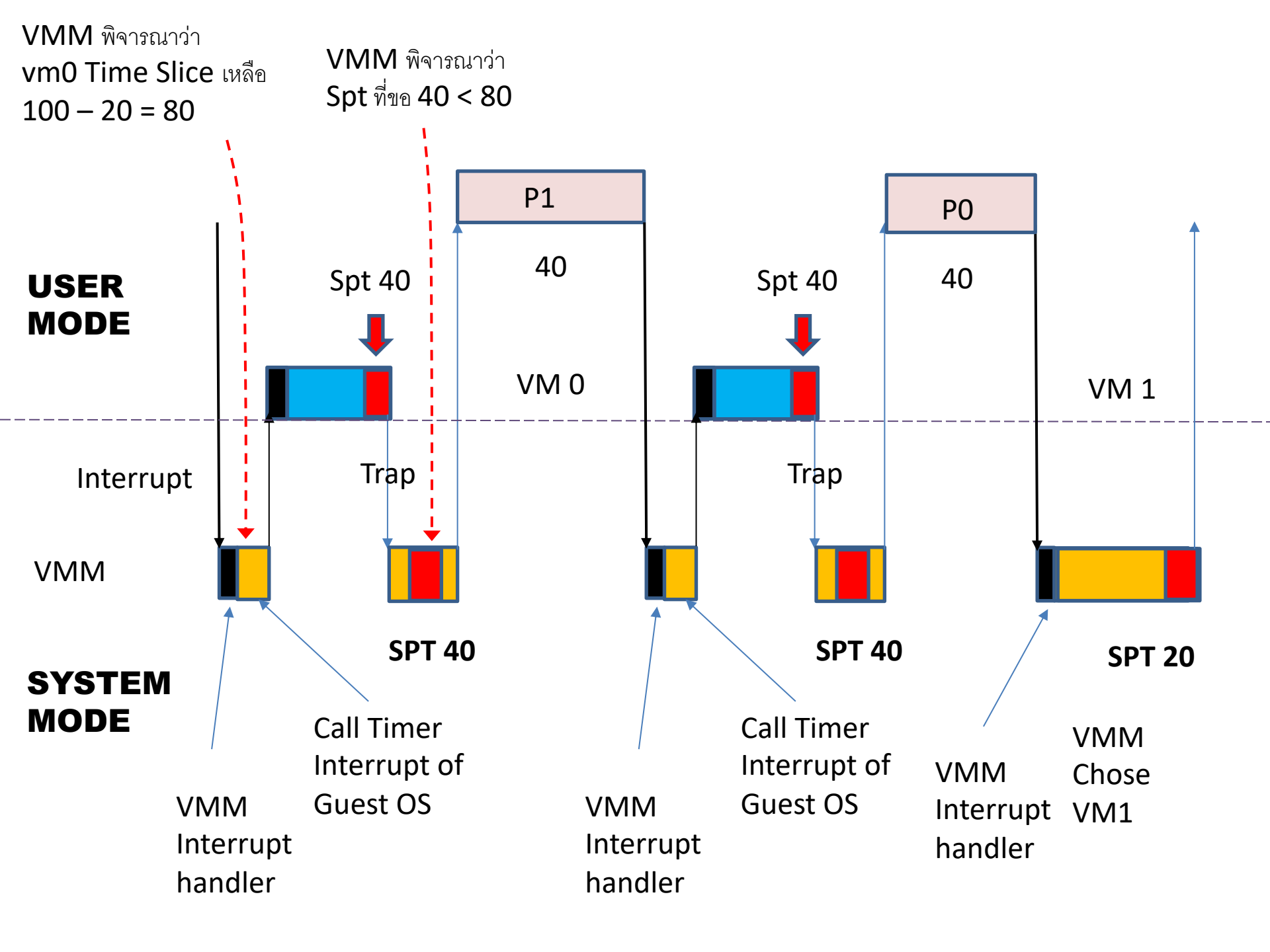


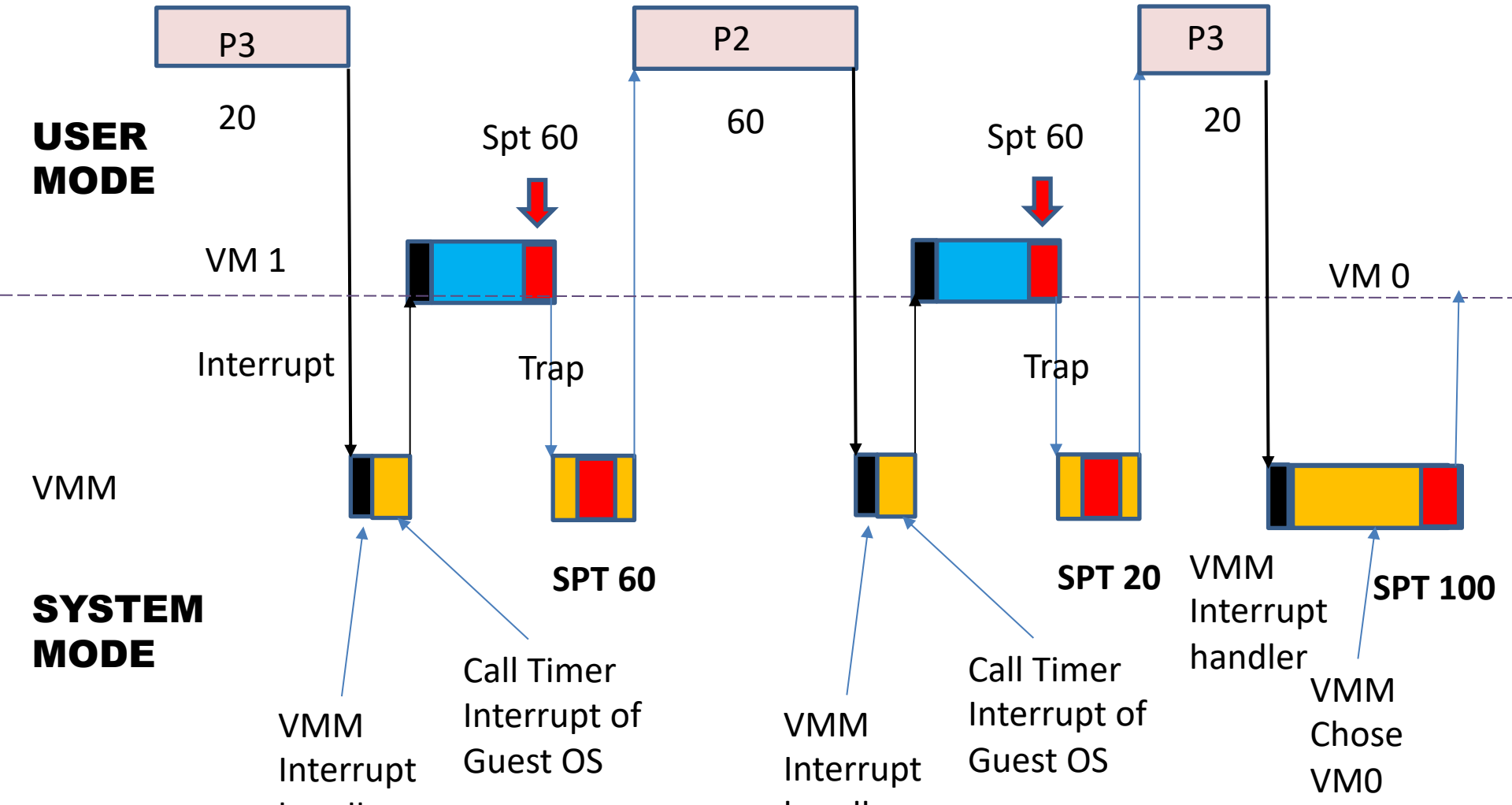

VMM พิจารณาว่า
Spt ที่ขอ 40 < 100
VMM พิจารณาว่า
vm0 Time Slice เหลือ
100 – 40 = 60
VMM พิจารณาว่า
Spt ที่ขอ 40 < 60
40
USER MODE
P0
Spt 40
Spt 40
VM 0
VMM กำหนดให้ CPU ไปรัน Guest OS ของ vm0
Trap
Interrupt
Trap
VMM
SPT 100
SPT 40
SYSTEM MODE
VMM Trap handler
instruction Dispatcher
Allocator
SPT 40
VMM Interrupt handler
Call Timer Interrupt of Guest OS

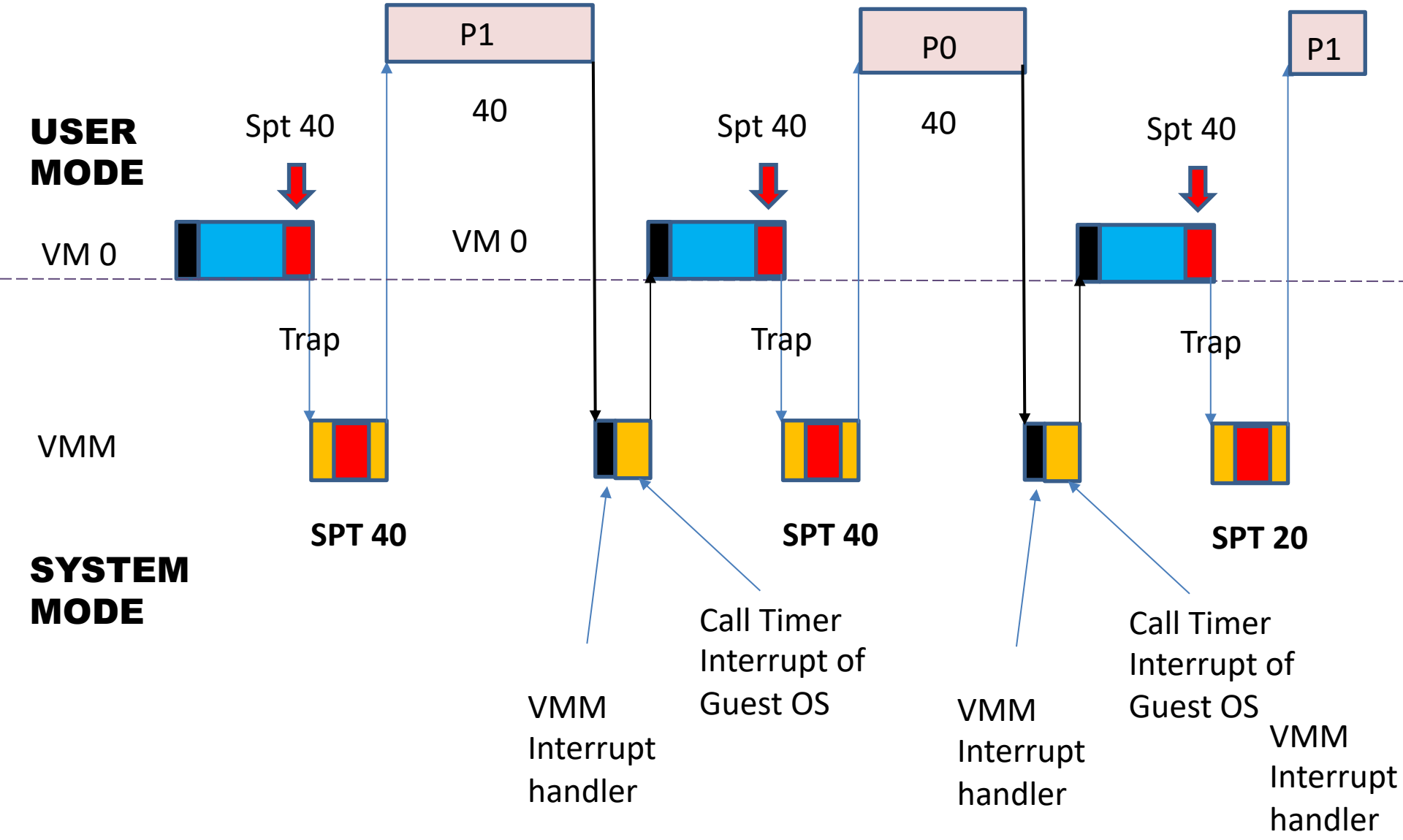

VMM พิจารณาว่า
VM Time Slice เหลือ
100 – 80 = 20
VMM พิจารณาว่า
Spt ที่ขอ 40 > 20
จึงให้ได้แค่ 20 ที่เหลือ
20 ทดไปรอบหน้า
VMM พิจารณาว่า
Time Slice ของ vm0
หมด เลือก vm1
P1
P0
40
20
USER MODE
Spt 40
Spt 60
VM 1
Interrupt
Trap
Trap
VMM
SPT 20
SPT 100
SPT 60
SYSTEM MODE
VMM Interrupt handler
Call Timer Interrupt of Guest OS
VMM Interrupt handler
VMM Chose VM1

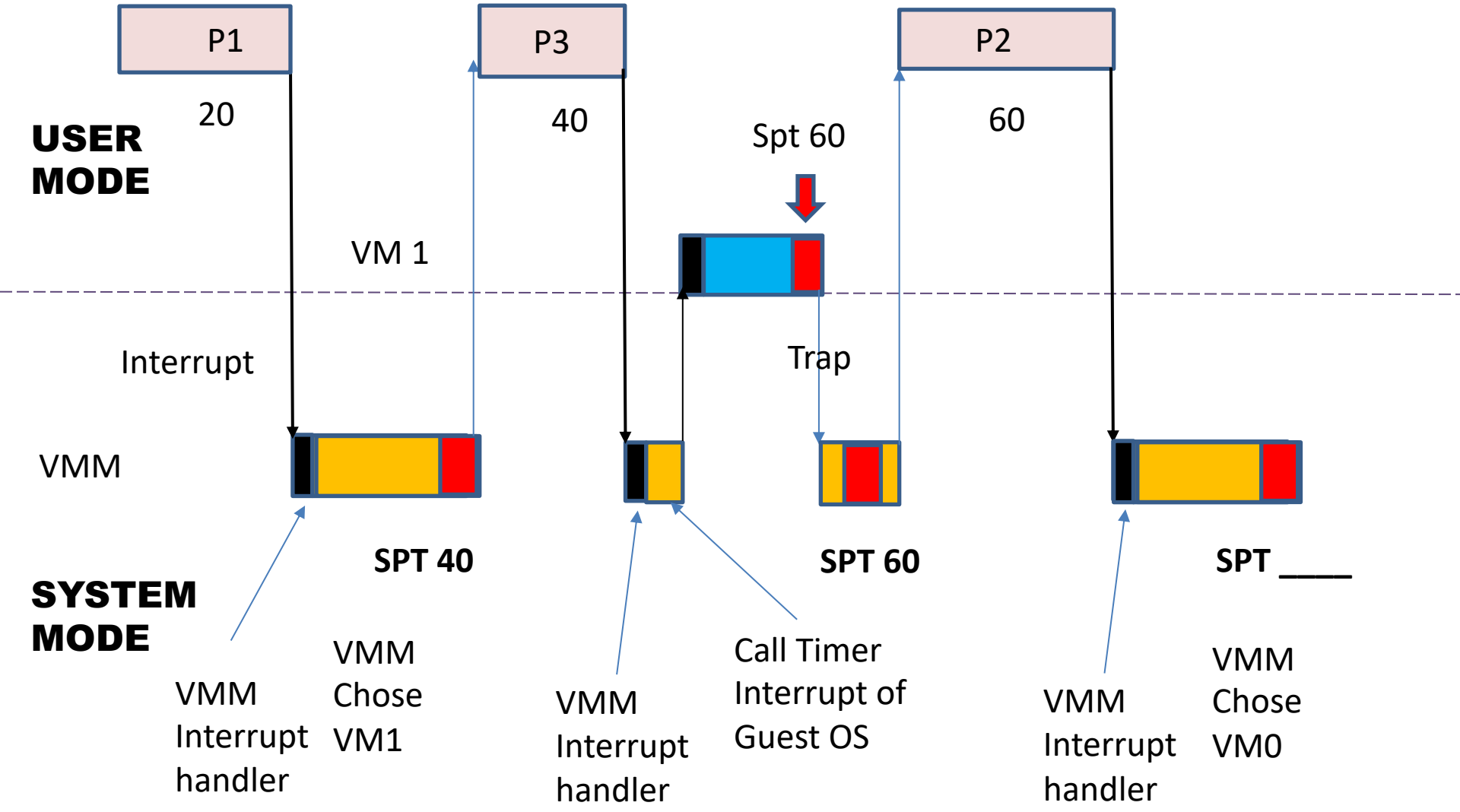
เมื่อเลือก vm0 แล้ว VMM พิจารณาว่า P0
มีเวลาเหลือจากรอบที่แล้ว = 20 < 100

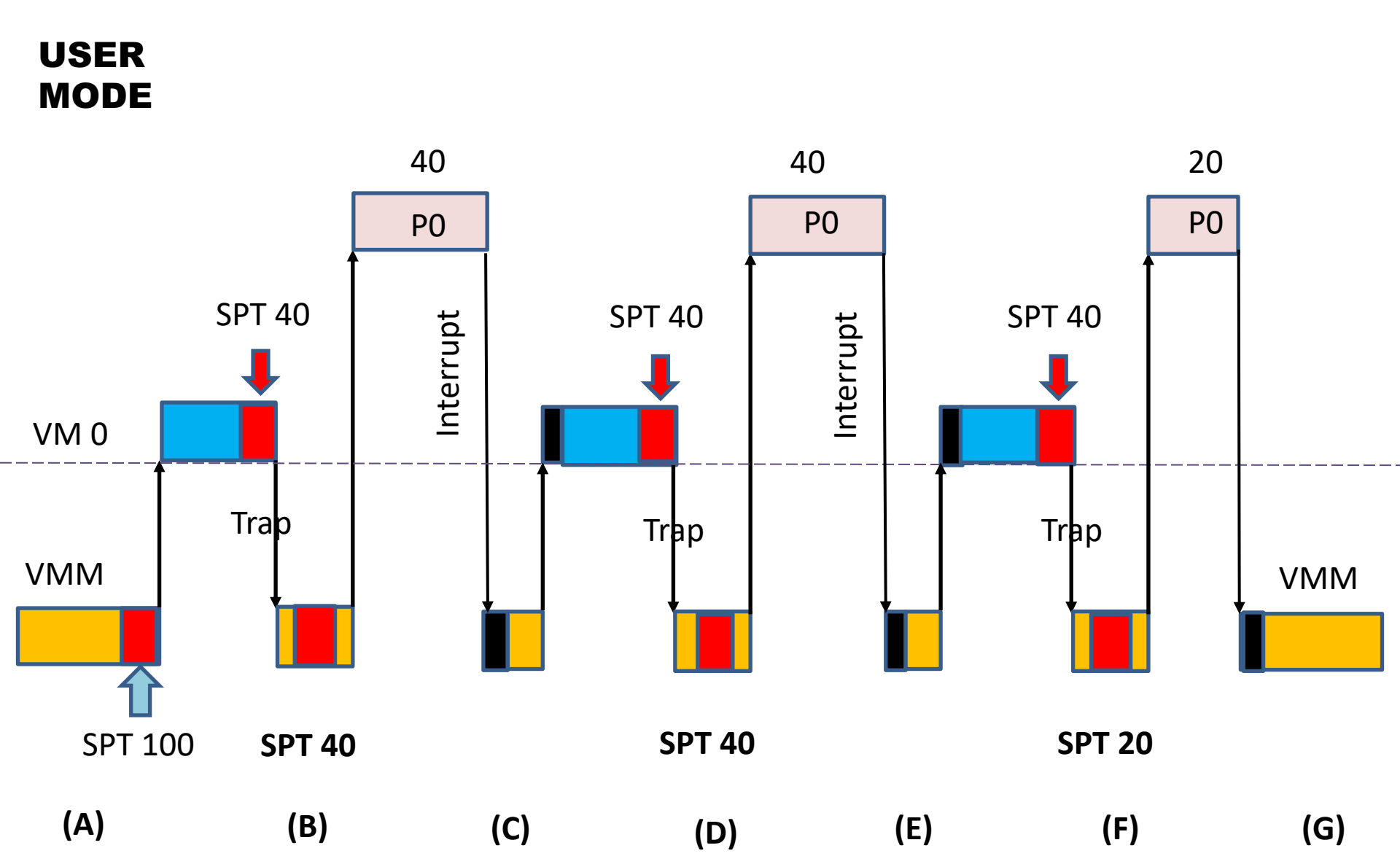



VMM พิจารณาว่า
vm0 Time Slice เหลือ
100 – 20 = 80
VMM พิจารณาว่า
Spt ที่ขอ 40 < 80
P1
40
P0
40
USER MODE
Spt 40
Spt 40
VM 0
VM 1
Interrupt
Trap
Trap
VMM
SPT 40
SPT 40
SPT 20
SYSTEM MODE
Call Timer Interrupt of Guest OS
Call Timer Interrupt of Guest OS
VMM Chose VM1
VMM Interrupt handler
VMM Interrupt handler
VMM Interrupt handler


P3
20
P2
60
P3
20
USER MODE
Spt 60
Spt 60
VM 1
VM 0
Interrupt
Trap
Trap
VMM
SPT 60
SPT 20
SPT 100
SYSTEM MODE
VMM Interrupt handler
Call Timer Interrupt of Guest OS
VMM Interrupt handler
Call Timer Interrupt of Guest OS
VMM Interrupt handler
VMM Chose VM0

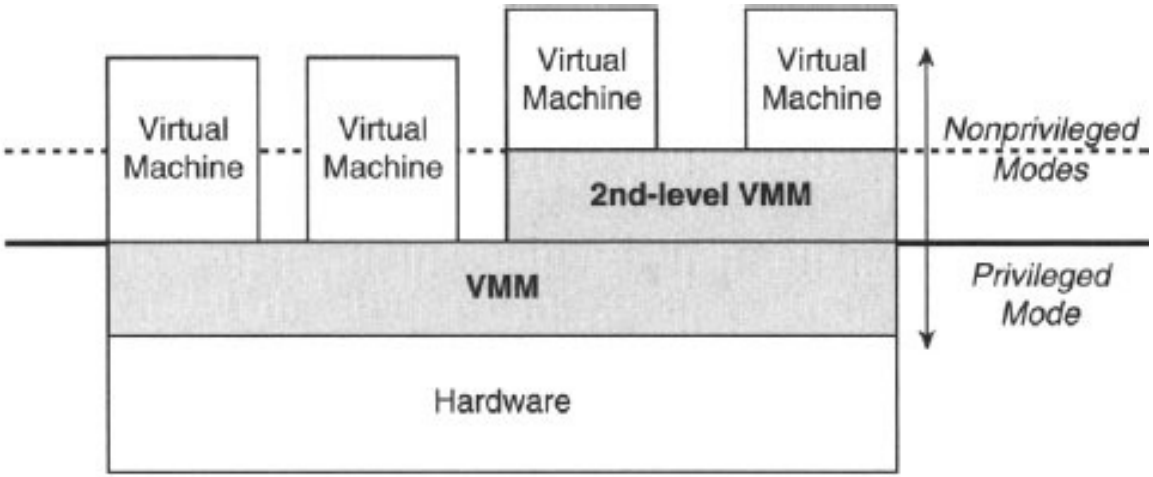

P1
P0
P1
40
40
Spt 40
Spt 40
Spt 40
USER MODE
VM 0
VM 0
Trap
Trap
Trap
VMM
SPT 40
SPT 40
SPT 20
SYSTEM MODE
VMM Interrupt handler
Call Timer Interrupt of Guest OS
VMM Interrupt handler
Call Timer Interrupt of Guest OS
VMM Interrupt handler


P1
P3
P2
20
40
60
Spt 60
USER MODE
VM 1
Interrupt
Trap
VMM
SPT 40
SPT 60
SPT ____
SYSTEM MODE
VMM Interrupt handler
VMM Chose VM1
VMM Interrupt handler
Call Timer Interrupt of Guest OS
VMM Interrupt handler
VMM Chose VM0


USER MODE
40
40
20
P0
P0
P0
SPT 40
SPT 40
SPT 40
Interrupt
Interrupt
VM 0
Trap
Trap
Trap
VMM
VMM
SPT 100
SPT 40
SPT 40
SPT 20
(A)
(B)
(C)
(D)
(E)
(F)
(G)
SYSTEM MODE

Theorem 2: Recursive virtualization

- **Theorem 2**
  - สำหรับ conventional third-generation computer จะมีคุณสมบัติ recursively virtualization ถ้า (1) คอมพิวเตอร์นั้นสามารถปฏิบัติการแบบ virtualization ได้และ (2) สามารถรัน 2^{nd}-level VMM ที่ไม่มีความต้องการเรื่องเวลา (เช่น real-time applications) บน VMM ที่รันบน host ได้



Recursive virtualization

- ขนาดของทรัพยากรที่มีให้ใช้บน Host จะลดลงเมื่อระดับชั้นของการทำ virtualization เพิ่มขึ้น
- ใช้ Verify correctness ของ VMM
- ข้อเสีย: VM ที่รันบน 2^{nd}-level VMM มี Performance ลดลง

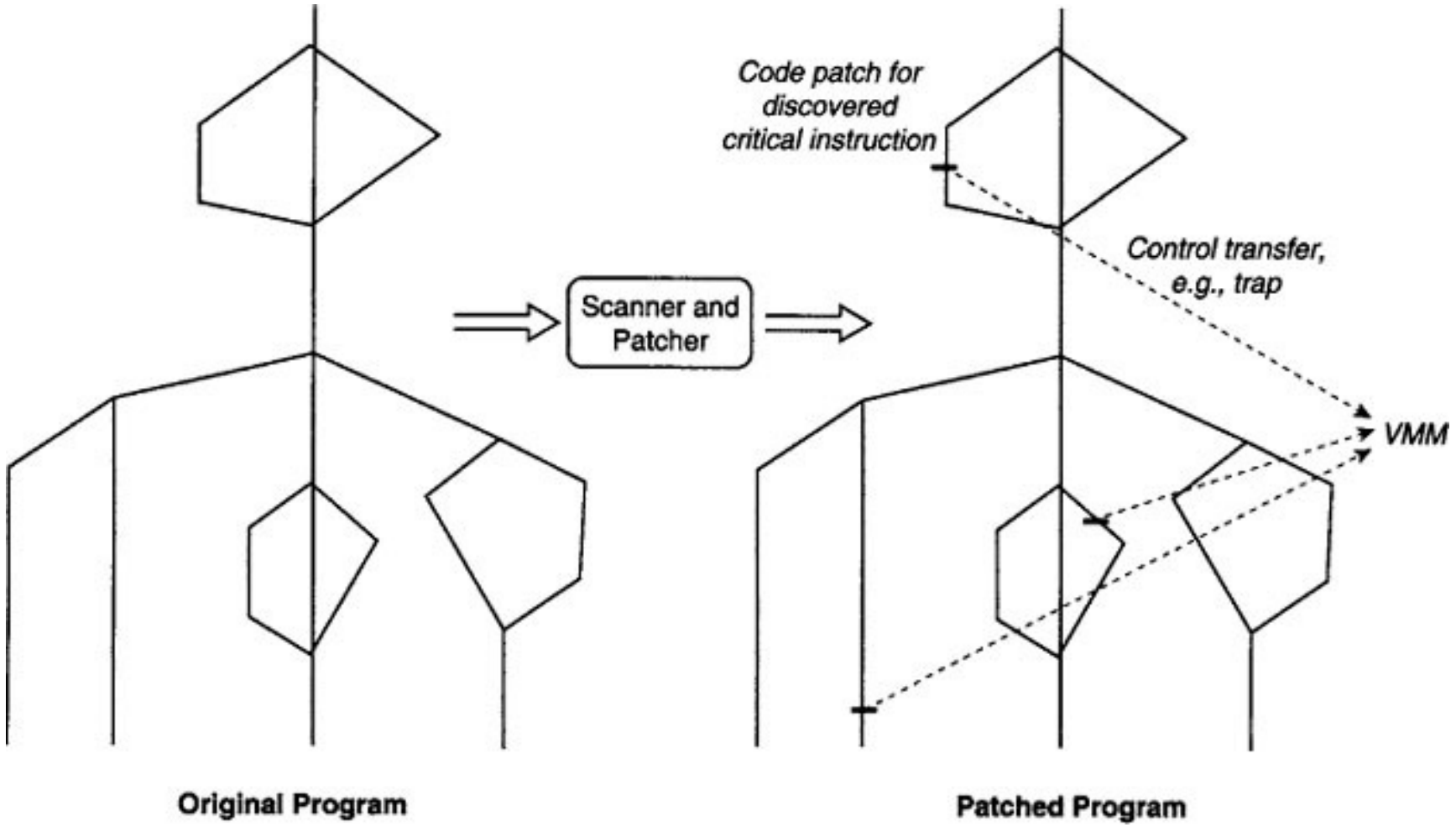
Handling of critical instructions

- ไม่ใช่ทุกคำสั่งจะเป็นไปตามทฤษฎีที่ 1 ของ Efficient VMM
- นิยาม Critical Instructions
  - คำสั่งที่เมื่อรันบน user mode จะไม่ trap ดังนั้น VMM จึงใช้ trap handler emulate คำสั่งให้ไม่ได้
- วิธีการแก้ปัญหา (1) Interpretation คือ VMM อ่านคำสั่งทุกคำสั่งก่อนส่งให้ CPU รัน ถ้าเจอคำสั่ง critical VMM ก็รันคำสั่งนั้นให้
  - ประสิทธิภาพต่ำเพราะต้องเช็คทุกคำสั่ง

Handling of critical instructions

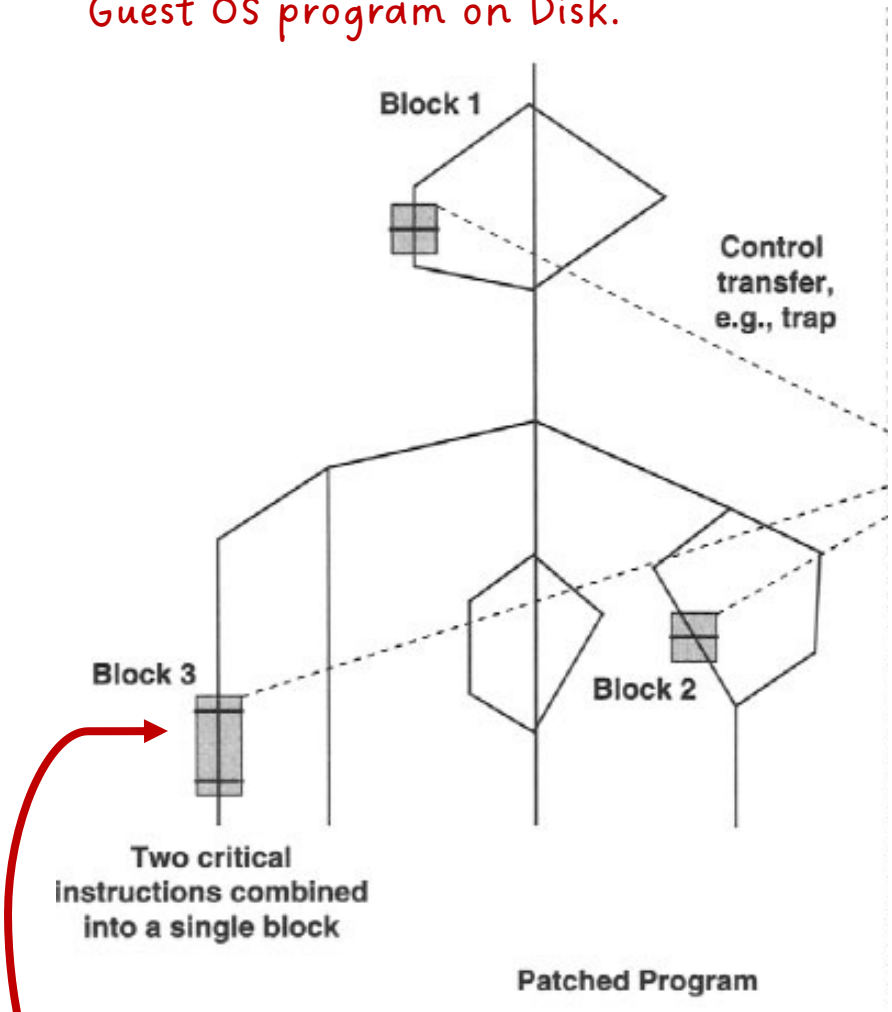
- วิธีการแก้ปัญหา (2) การ Patching Code ของ Guest OS
  - ใช้วิธีการแปลง Guest OS code ใน VM image แบบ offline ก่อนรัน VM
  - Scan code ของ Guest เพื่อหา Critical section
  - เมื่อพบแล้วให้ Patch คำสั่ง trap พร้อมทั้งข้อมูลที่จำเป็นสำหรับ VMM แทนที่คำสั่ง Critical Instruction ที่เจอ
- ใช้วิธีการ Code Caching เพื่อเพิ่มประสิทธิภาพ

Guest OS Patching



Code caching

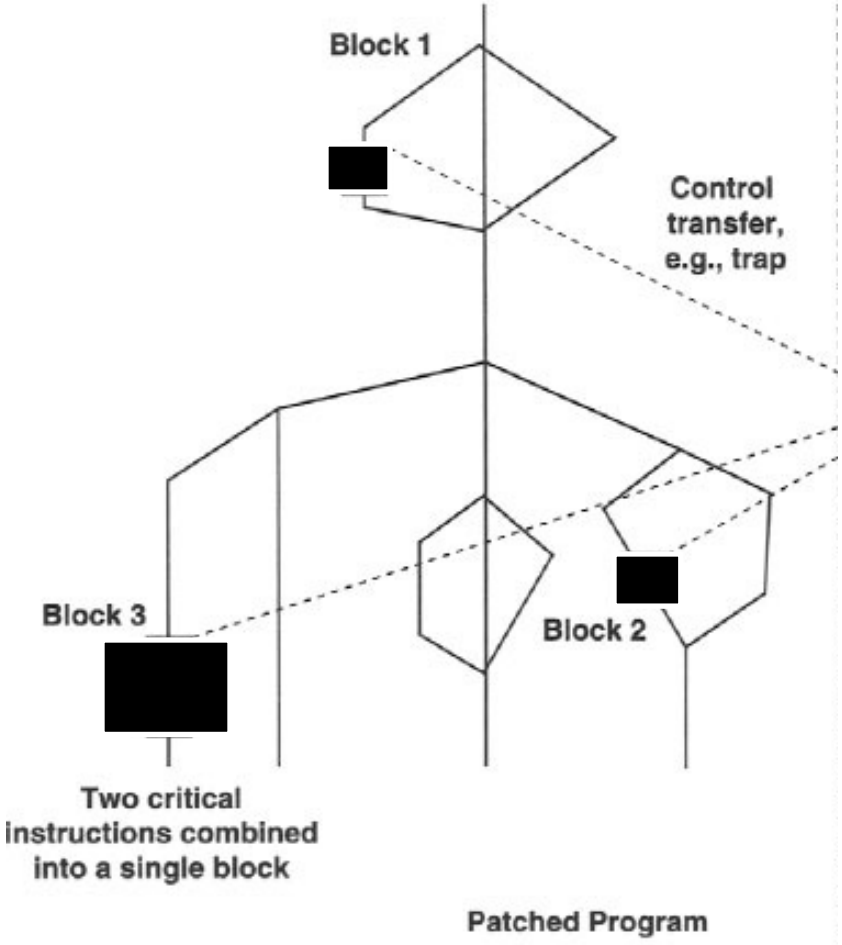
Guest OS program on Disk.



Block 3 แสดง ตัวอย่างที่ 1 block ครอบคลุม code ที่มีการ trap 2 ครั้ง

Code caching

Everything is on Disk.

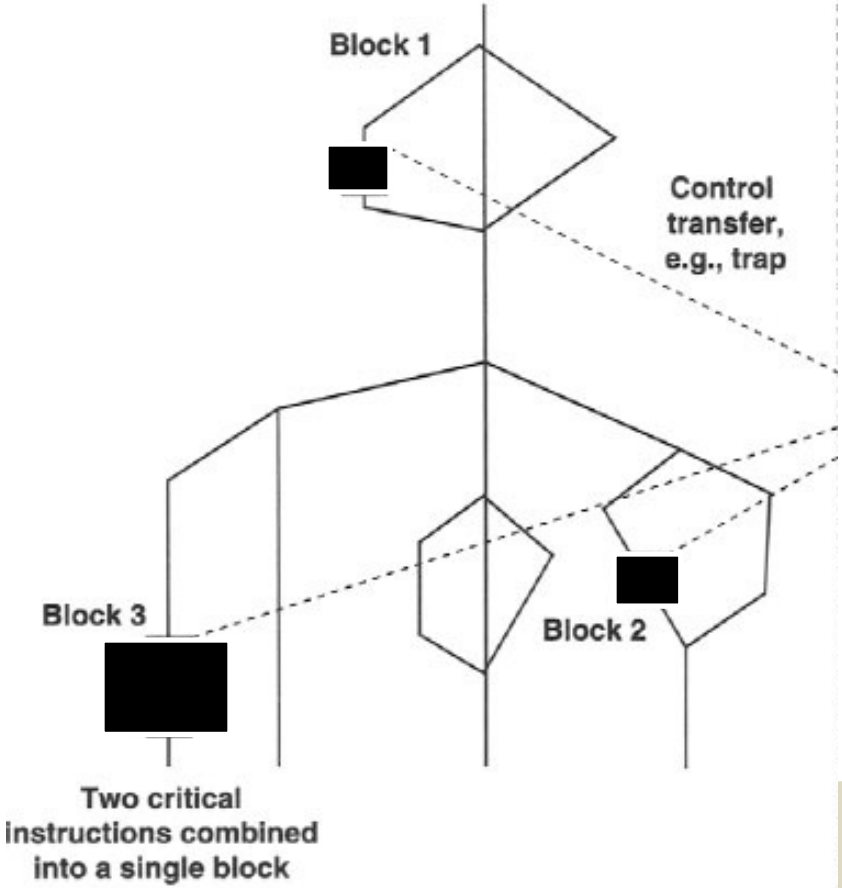


index	
1	block1
2	block2
3	block3
...	...
n	block n

1. ตัด code ส่วนที่มีการ patching traps แทน critical section ออกทั้ง block ซึ่งอาจ cover คำสั่ง trap หลายคำสั่ง มาไว้ใน table[*Z*]
2. แทนที่ code block นั้นด้วย trap และ hash tag (หรือ index *Z*) ของ code ใน table[i]

Code caching

Patched program is loaded to memory

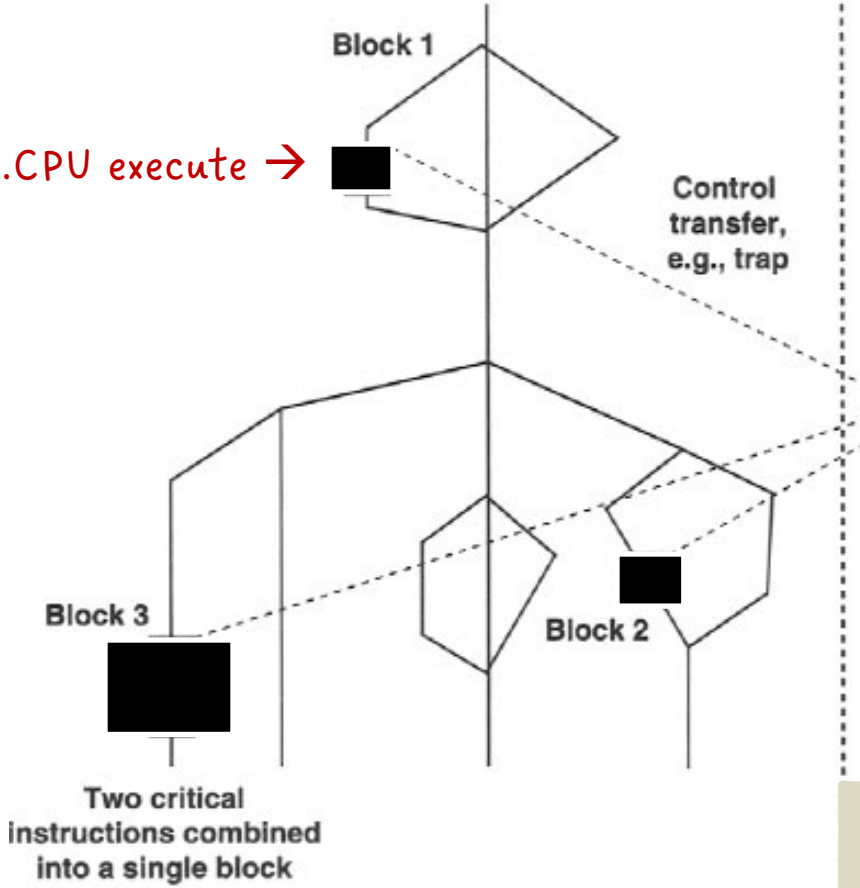


Table[z] is on Disk.

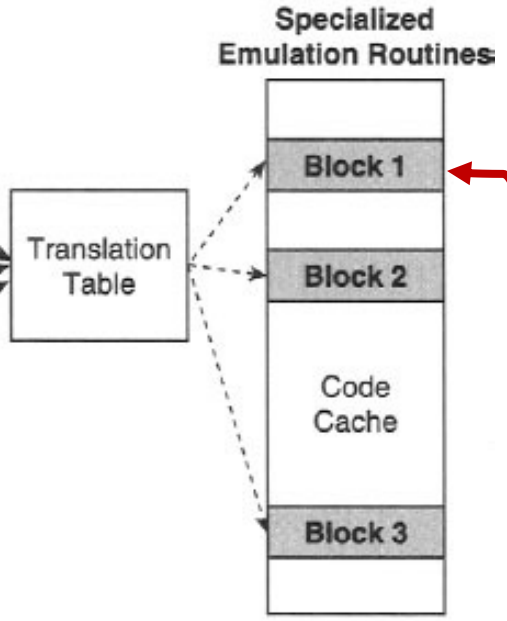
index	
1	block1
2	block2
3	block3
...	...
n	block n

Code caching

Patched program is loaded to memory



Code cache is in memory



2. Trap occurs. VMM loads Block 1 to **code Cache** in Memory

index	
1	block1
2	block2
3	block3
...	...
n	block n

Table[z] is on Disk.

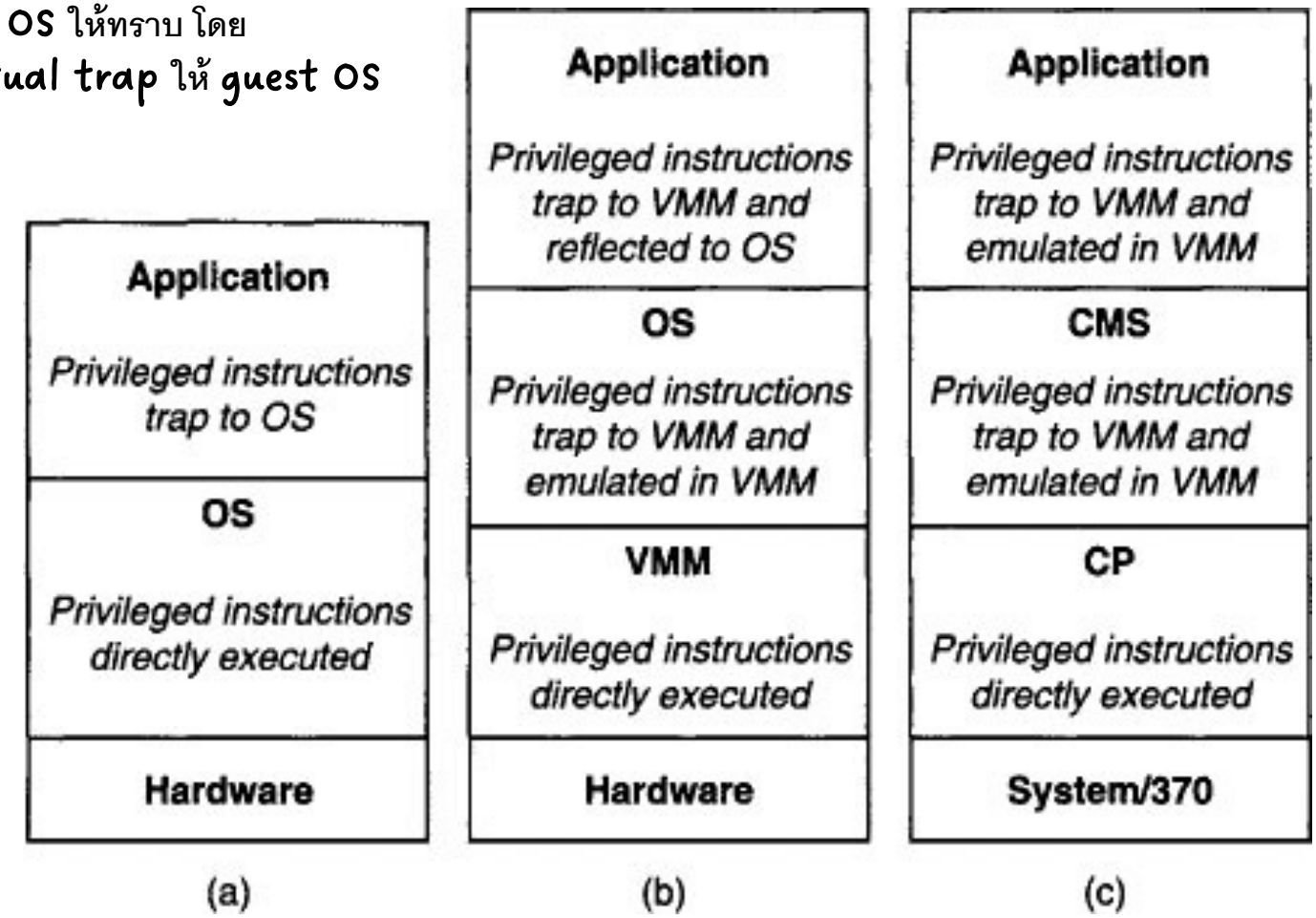
3. If CPU run block 1 again, Trap handler will execute Block 1 in code cache right away.

Code caching

1. ตัด code ส่วนที่มีการ patching traps แทน critical section ออกทั้ง block ซึ่งอาจ cover คำสั่ง trap หลายคำสั่ง มาไว้ใน table[*Z*]
2. แทนที่ code block นั้นด้วย trap และ hash tag (หรือ index *Z*) ของ code ใน table[i]
3. เมื่อ VM รันไปเจอคำสั่ง trap ของ code block นั้น CPU ก็จะเรียก trap handler ของ VMM เข้ามารันและ handler จะพิจารณาว่า hash tag ที่เจอ เป็นของ code block ใด ใน table[i]
4. VMM จะเช็คจาก memory code cache ว่า code block *Z* นี้เคยถูก load และมีอยู่ ในmemory cache หรือไม่ ถ้ามีก็ให้ CPU fetch code block นั้นจาก memory cache ถ้าไม่มีก็จะ load code นั้นจาก Disk ไปไว้ใน memory cache และให้ CPU รัน code นั้น
5. ช่วยเพิ่มประสิทธิภาพ เพราะ (1) code ที่มีการใช้งานบ่อยจะอยู่ใน code cache และ (2) ตอน CPU พบ trap ของ block 3 จะ trap ครั้งเดียว (แทนที่จะเป็น 2 ครั้ง)

IBM ออกแบบ CP และ CMS
ให้ CP สามารถจัดการกรณีที่
Apps รัน privilege inst ให้

Apps พยายามรันคำสั่งของ
OS VMM จะต้องแจ้ง
Guest OS ให้ทราบ โดย
ส่ง virtual trap ให้ guest OS



สรุป

- แนะนำ System Virtual Machine
- เงื่อนไขของการทำ Processor Virtualization
- Processor Virtualization