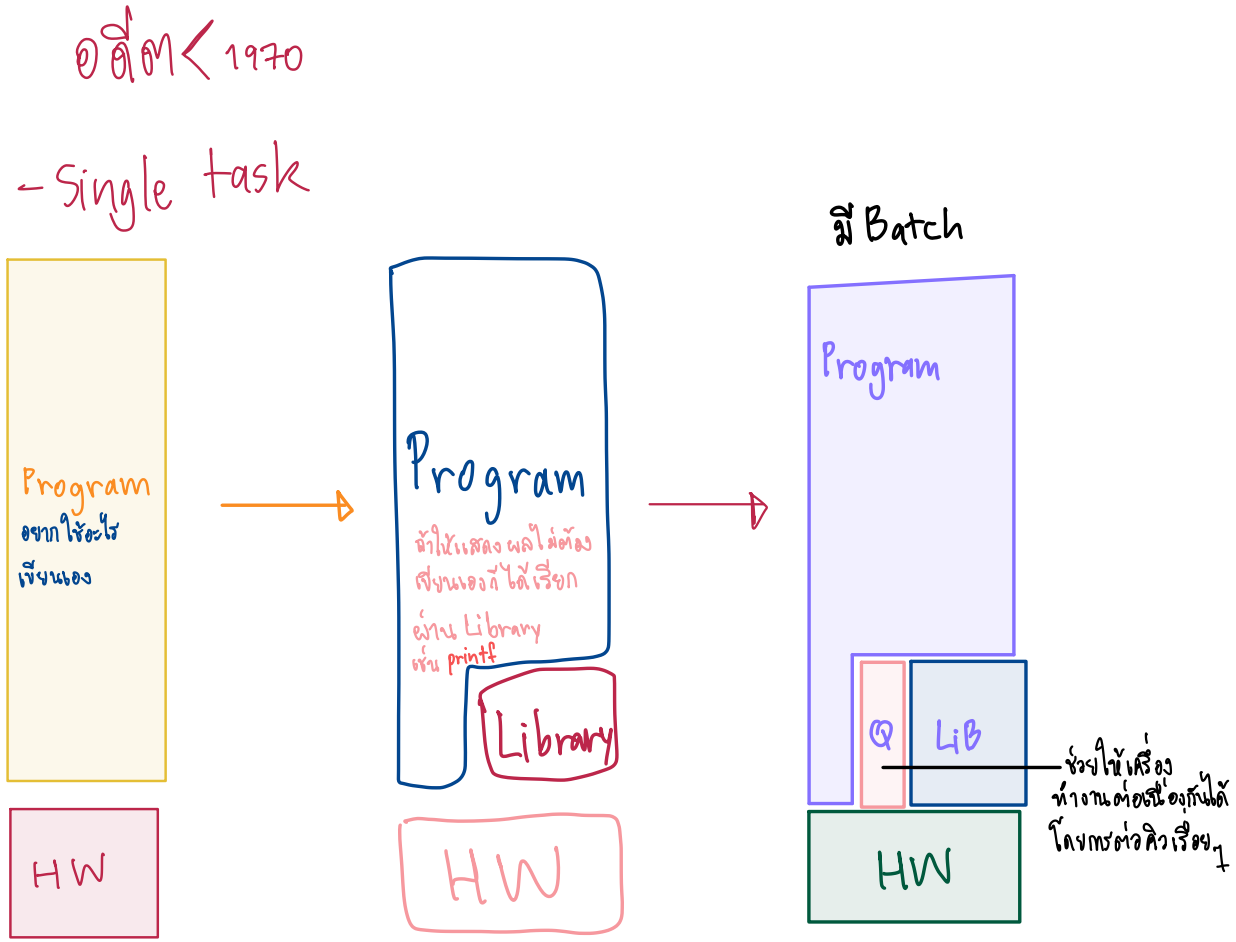
The Kernel Abstraction

Slides adopted from CSE 451 class at UW, CS162 class at Berkeley and CSE 421/521 class at UB

อดีต

- Single task system



ทำทั้งหมด เพื่อให้เกิด "Protection"

ปัจจุบัน

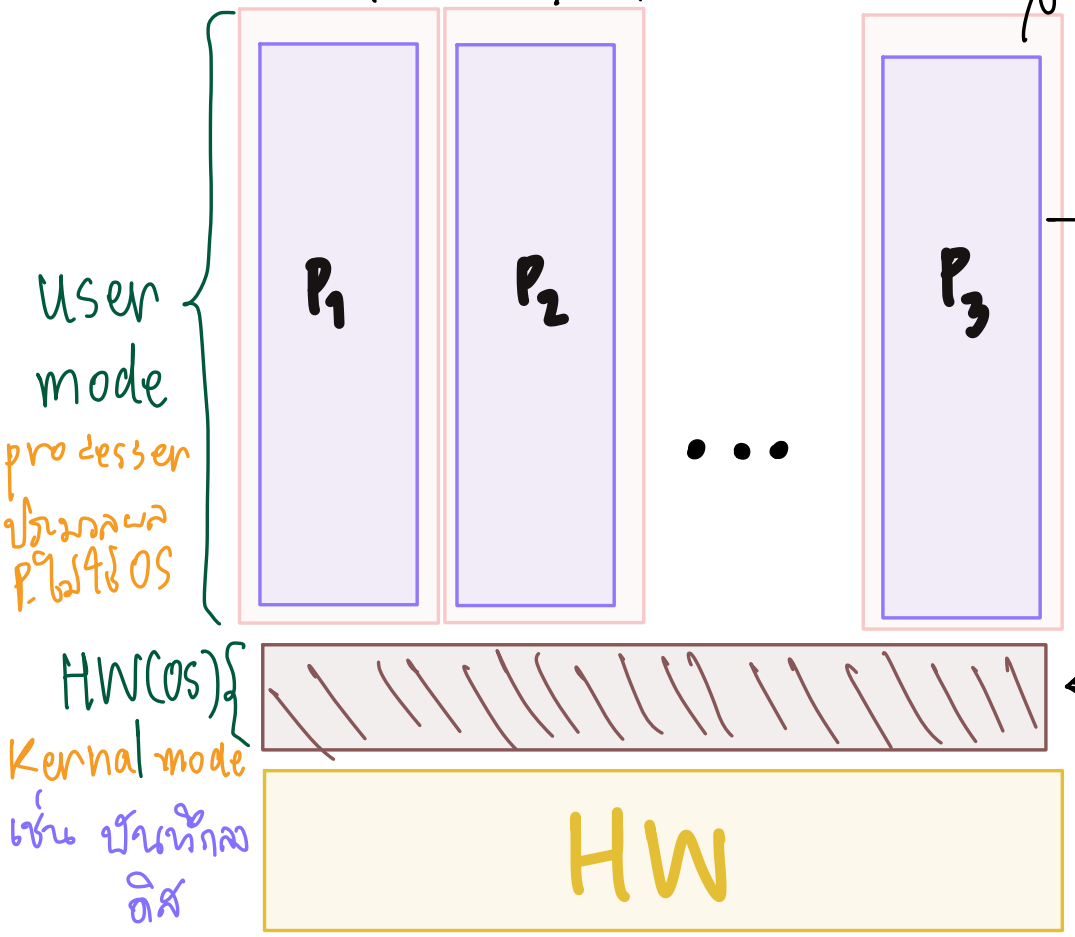
ทำได้หลายงานในเวลาเดียวกัน เช่น เปิด word พร้อม Chome

- Multitasking system

modern > 1980

Process

GOAL
ต้องทำให้ ทำงานพร้อมกัน



→ Process คือป้องกันให้แต่ละ Programe ไม่ให้มากวนกัน

❤ โปรแกรมทุกอันรันบน Process ออกมายุ่งกับ HW ไม่ได้

Q (พัฒนา) → Resouse Manager (Load Manager)

Lib → System Library

เพิ่ม Hardware access control
(ควบคุมการเข้าถึง Hardware)

- Isolation (SW)

Activity #1 13,57

- การเปลี่ยนแปลงจากระบบแบบ single task ไปเป็นระบบแบบ Multitask ... สิ่งที่จะต้องมีการปรับแต่งหรือเพิ่มเติมเข้ามาใน OS ได้แก่...

สร้าง Protection ขึ้นมา

Q (พ้องมา) → Resouse Manager (Load Manager)

Lib → System Library

เพิ่ม Hardware access control

(ควบคุมการเข้าถึง Hardware)

- Isolation (SW)

What does an OS do...

- Hiding Complexity ADD-ON ของ process
  - Variety of HW
    - E.g. different CPU, amout of RAM, I/O devices

เขียนแล้ว RUN ใน window อื่นได้ ทุกการคำสั่งของ Program เห็น HW ที่อื่นเป็นสเปคเดียวกันหมดเลย

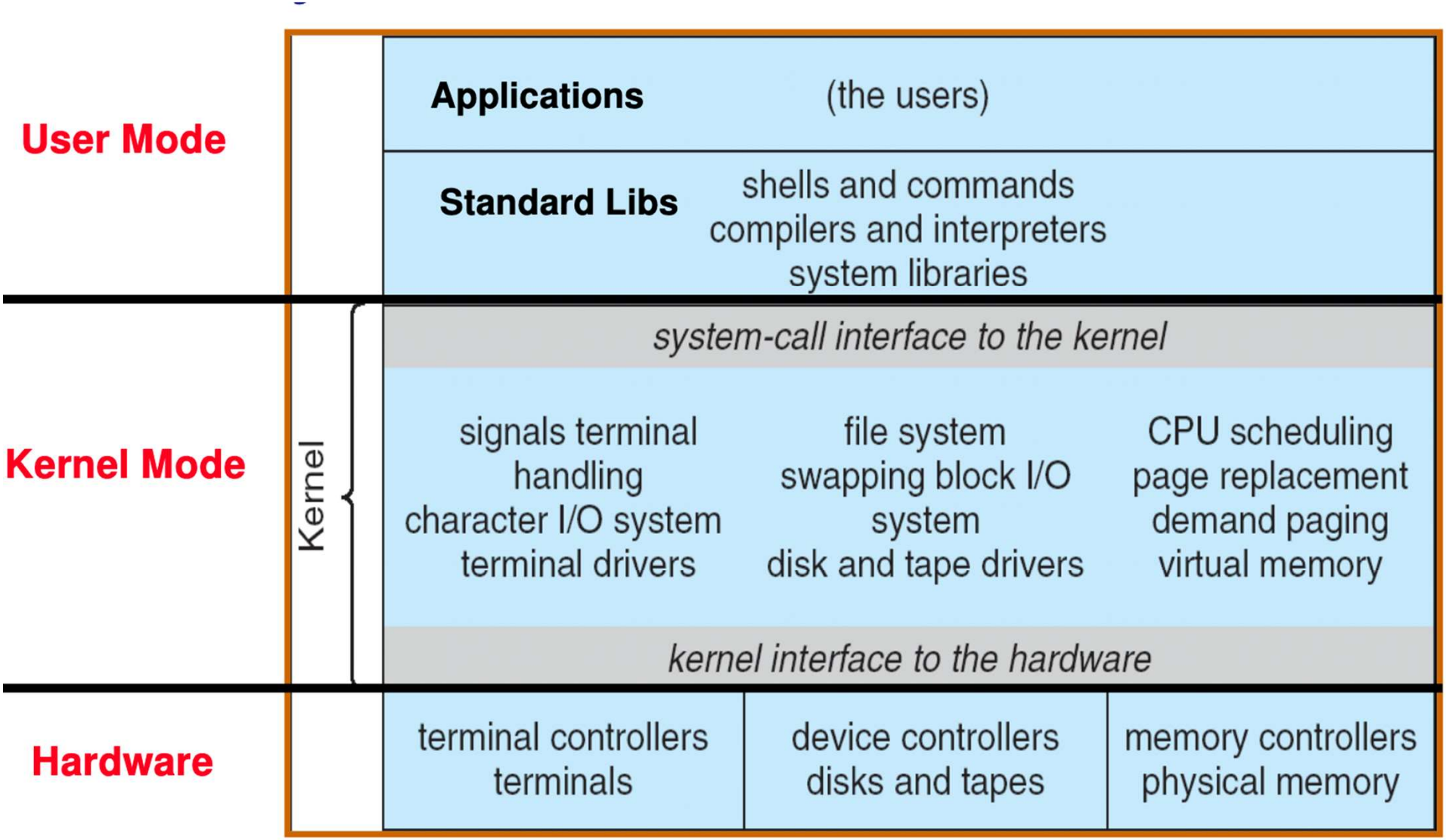
- Kernel is the part of the OS that running all the time on the computer

set of SW อยู่ใน memory ตลอดเวลา (แกนระบบปฏิบัติการ) ตัวอื่นมีเข้า-ออก memory

  - Core part of the OS
  - Manages system resources
  - Acts as a bridge between apps and HW

UNIX Structure

สรุปแล้วทำแบบนี้



ระบบกลางเก่ากลางใหม่ ตระกูล Window ธรรมดา 95/98/xp/ME

User/Kernel (Privileged) Mode

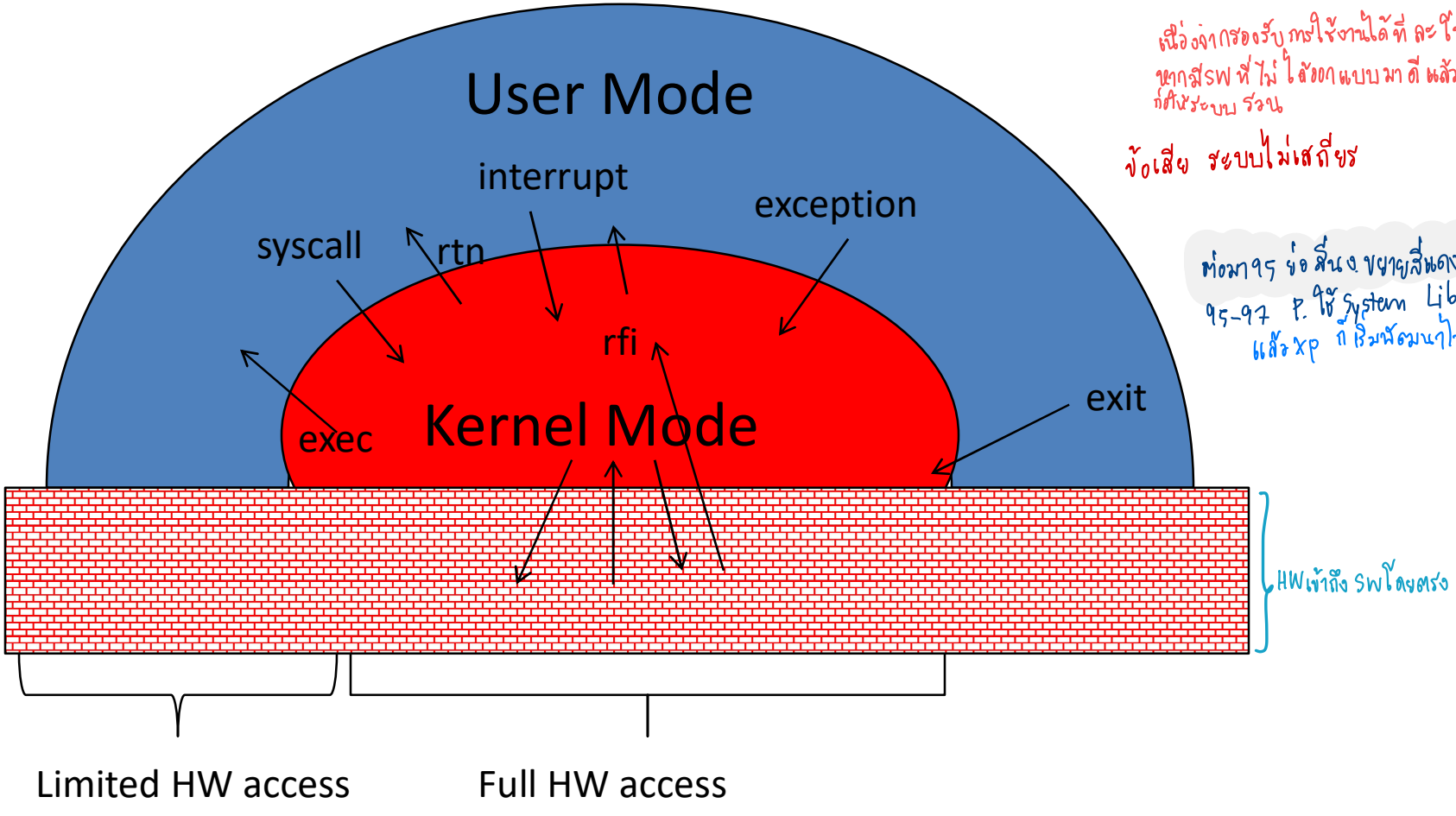
ใช้งานเยอะ เพราะใช้งาน Program ยุคเก่าได้ (Dos) Programe เก่า เข้าถึง HW โดยตรง

คนใช้งานเยอะ → มี Program พัฒนามาขายเยอะ

เนื่องจากรองรับการใช้งานได้ที่ละโปรแกรม หากมีรพที่ไม่ใช้ออกแบบมาดี แล้วทำงานพร้อมกันก็ทำให้ระบบร่วน

ข้อเสีย ระบบไม่เสถียร

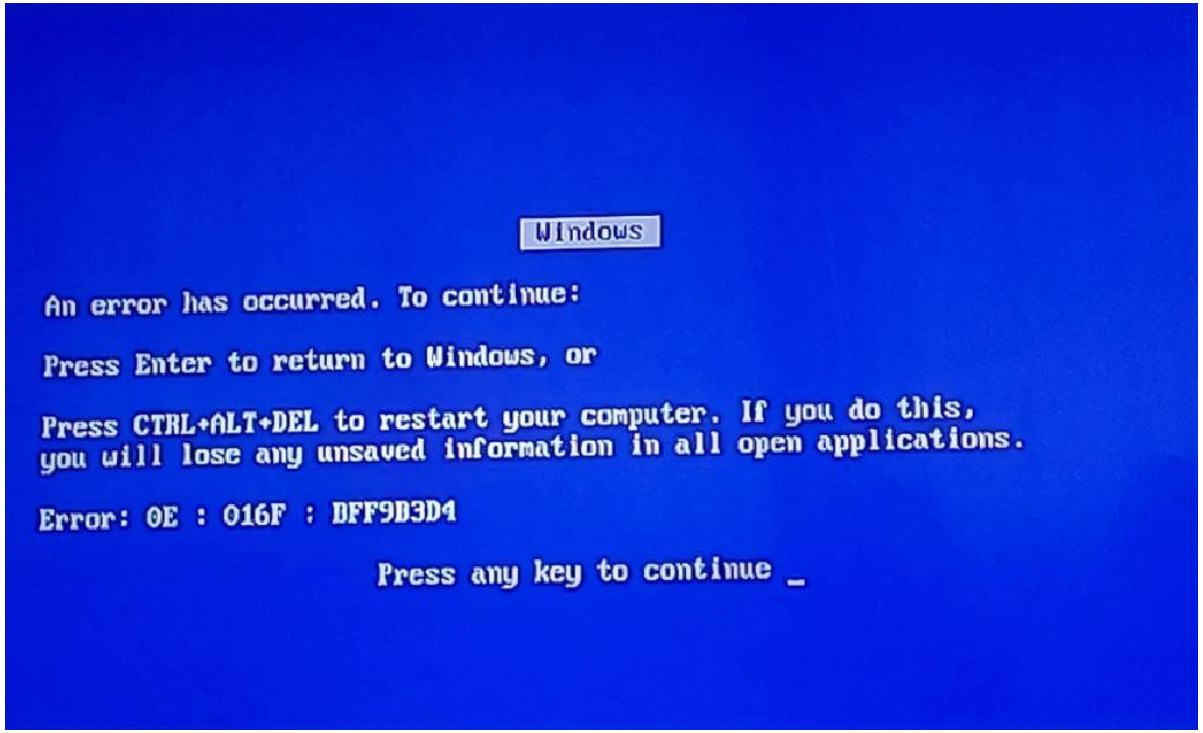
ต่อมา 95 ย่อสิ่นง ขยายสิ่นอง 95-97 P. ใช้ System Lib มากขึ้น แล้ว xp ก็เริ่มพัฒนาไปอีก



One of the major goals of OS is...

ที่มาของฟ้า
interfear = ทำผิดกติกา/ผิดกฎ
ปิดตั้งแต่ P_1 พยายามบันทึกข้อมูลของมัน
ตน.ของ P_2 ถ้าปิดไม่ได้ระบบจะปิด
ระบบตัวเองทำให้จอฟ้า

- Protecting Process and the Kernel
  - Running multiple programs
    - Keep them from interfering (การแทรกแซง) with the OS - kernel
    - Keep them from interfering with each other



Activity #2: Protection: WHY?

เวลา 10 นาที

การ protect Process และ Kernel ทำให้เกิด impact อะไรกับระบบบ้าง และยังต้อง protect อะไรอีกบ้าง เพื่ออะไร

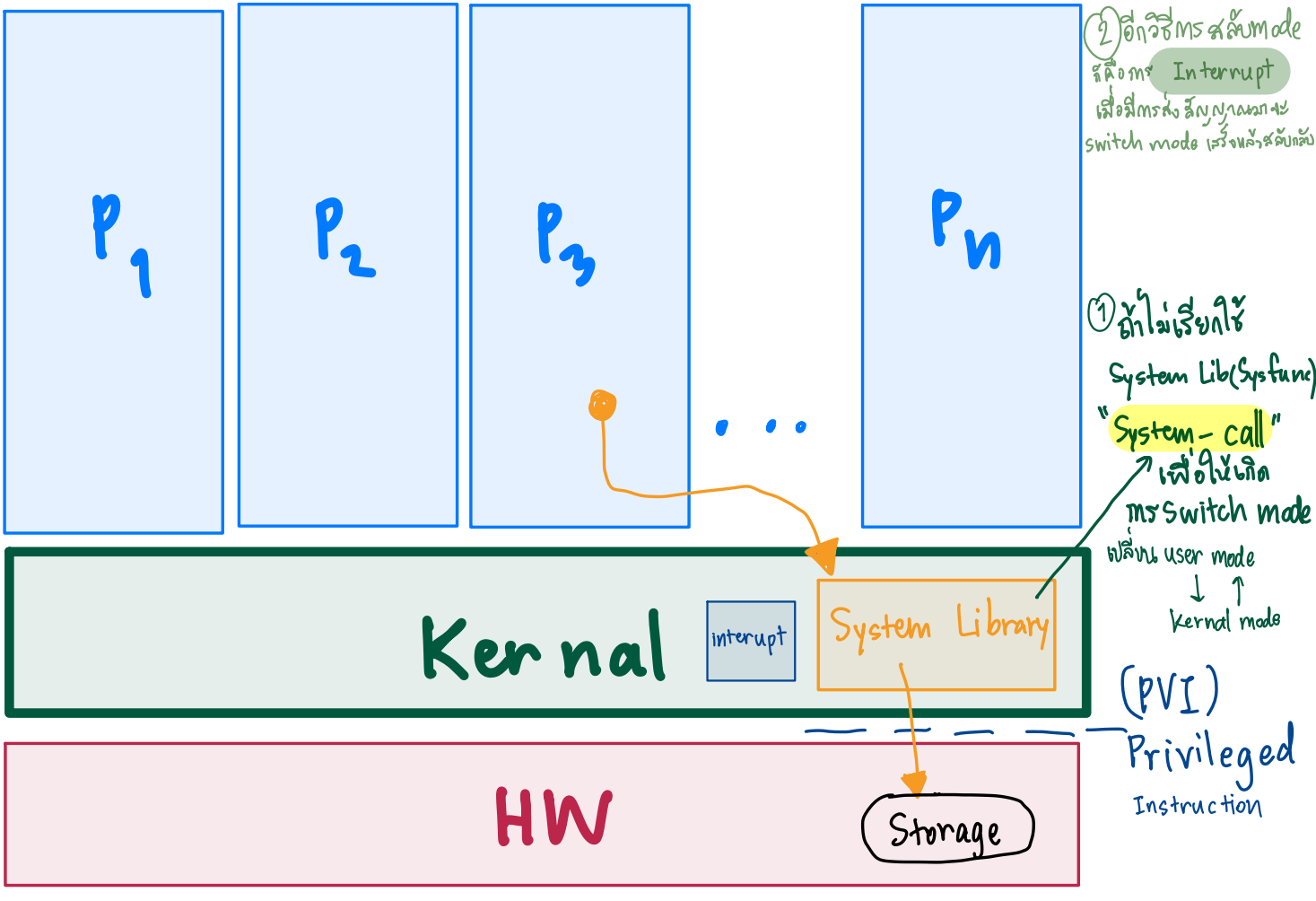
- Reliability: buggy programs only hurt themselves
  เกิดบัค แค่เฉพาะโปรแกรมนั้น
- Security and privacy: trust programs less
  ↳ process ครอบได้ความปลอดภัย อ่านข้อมูลคนอื่นไม่ได้ privacy ค.เป็นส่วนตัวมีตั้งแต่ ๆ 80
- Fairness: enforce shares of disk, CPU
  protect hardware ด้วย ฮีทรอัดคิว ฮิสิทธ์เข้าไปใช้ได้กี่อิ

Protection: How? (HW/SW)

เวลา 10 นาที

- 2 Main HW mechanism
  - Memory address translation → Virsual add. ข้อมูลใช้ เวลาเกิด pointer / แกรกเซ็ง เข้าถึงข้อมูลจริงไม่ได้ add. ซึ่งเวลาใช้งานต้องแยกของ จริง/ปลอม
  - Dual mode operation
    - Privileged (Kernal mode) เกี่ยวข้องกับการ access HW โดยตรง
    - non-privileged mode (ทุก mode Kernal, User)
      - System calls (SW เริ่มต้นขบวนการ) → ไม่เกี่ยว HW/ทรัพยากร / I/O ใน process จะใช้ User mode ได้แค่นี้

- SW
  - Process
  - System calls (SW interupts)


P1
P2
P3
Pn
. . .
Kernal
interupt
System Library
HW
Storage
(PVI)
Privileged
Instruction
② อีกวิธีการสลับmode
คือการ Interrupt
เมื่อมีการส่ง สัญญาณมาจะ
switch mode เสร็จแล้วสลับกลับ
① ถ้าไม่เรียกใช้
System Lib(Sysfunc)
"System - call"
เพื่อให้เกิด
การ switch mode
เปลี่ยน user mode
↓ ↑
Kernal mode

Hardware Support: Dual-Mode Operation

- Kernel mode
  - Execution with the full privileges of the hardware
  - Read/write to any memory, access any I/O device, read/write any disk sector, send/read any packet
- User mode
  - Limited privileges
  - Only those granted by the operating system kernel
- On the x86, mode stored in EFLAGS register
- On the MIPS, mode in the status register

Hardware Support: Dual-Mode Operation

- Privileged instructions
  - Available to kernel
  - Not available to user code
- Limits on memory accesses
  - To prevent user code from overwriting the kernel
- Timer
  - To regain control from a user program in a loop
- Safe way to switch from user mode to kernel mode, and vice versa

Privileged instructions

- Examples?

- What should happen if a user program attempts to execute a privileged instruction?

โปรแกรม error ล่ะ ปิดไปเอง
เลวร้ายสุด จอฟ้า

User Mode

- Application program
  - Running in process

Virtual Machine:VM

คอมพิวเตอร์เสมือน

- Software emulation of an abstract machine
  - Give programs illusion they own the machine
  - Make it look like HW has feature you want
- 2 types of VM
  - Process VM หยาบๆ เช่น window เหมือนกัน ใช้ด้วยกันได้
    - Supports the execution of a single program (one of the basic function of the OS)
  - System VM ละเอียด จำลอง HW ทำให้ OS รันใน window ได้
    - Supports the execution of an entire OS and its applications

Process VMs

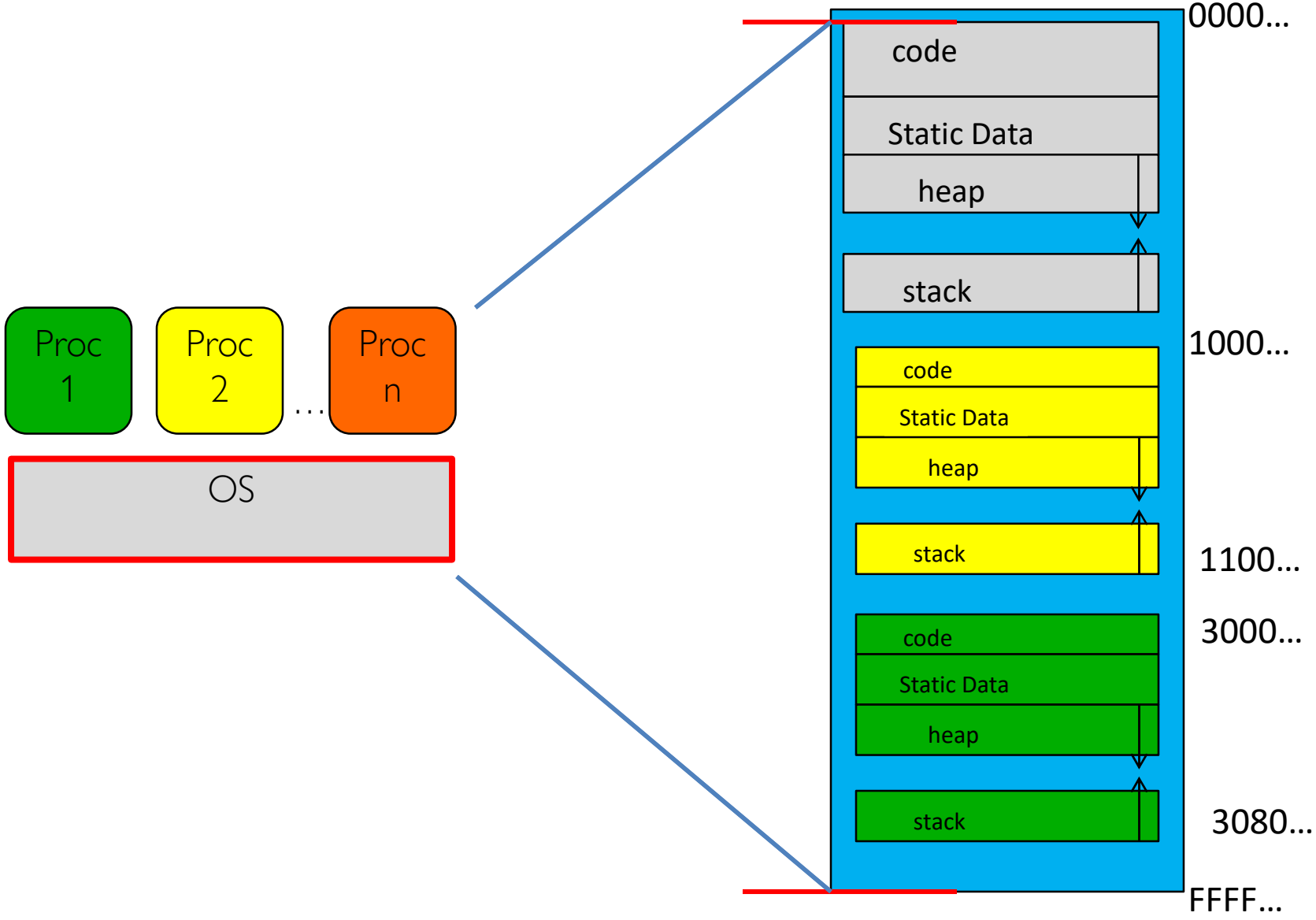
- GOAL:
  - Provide an isolation to a program

    - Processes unable to directly impact other processes
      - Boundary to the usage of a memory
    - Fault isolation
      - Bugs in program cannot crash the computer

  - Portability (Program)

    - Write the program for the OS rather the HW

Kernel mode & User mode

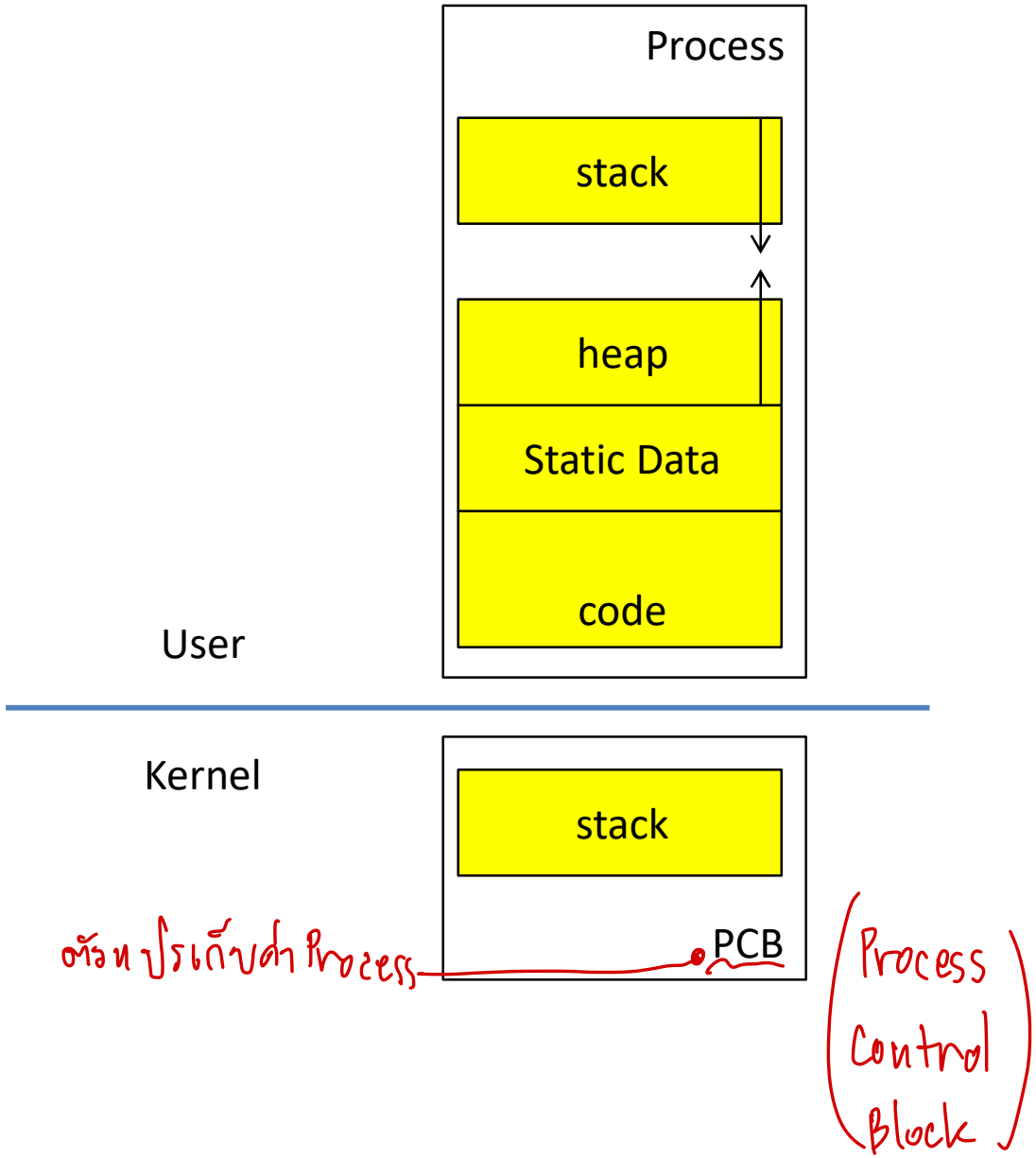


Process Abstraction

- Process: an *instance* of a program, running with limited rights
  - Thread: a sequence of instructions within a process
    - Potentially many threads per process (for now 1:1)
  - Address space: set of rights of a process
    - Memory that the process can access
    - Other permissions the process has (e.g., which system calls it can make, what files it can access)

Process

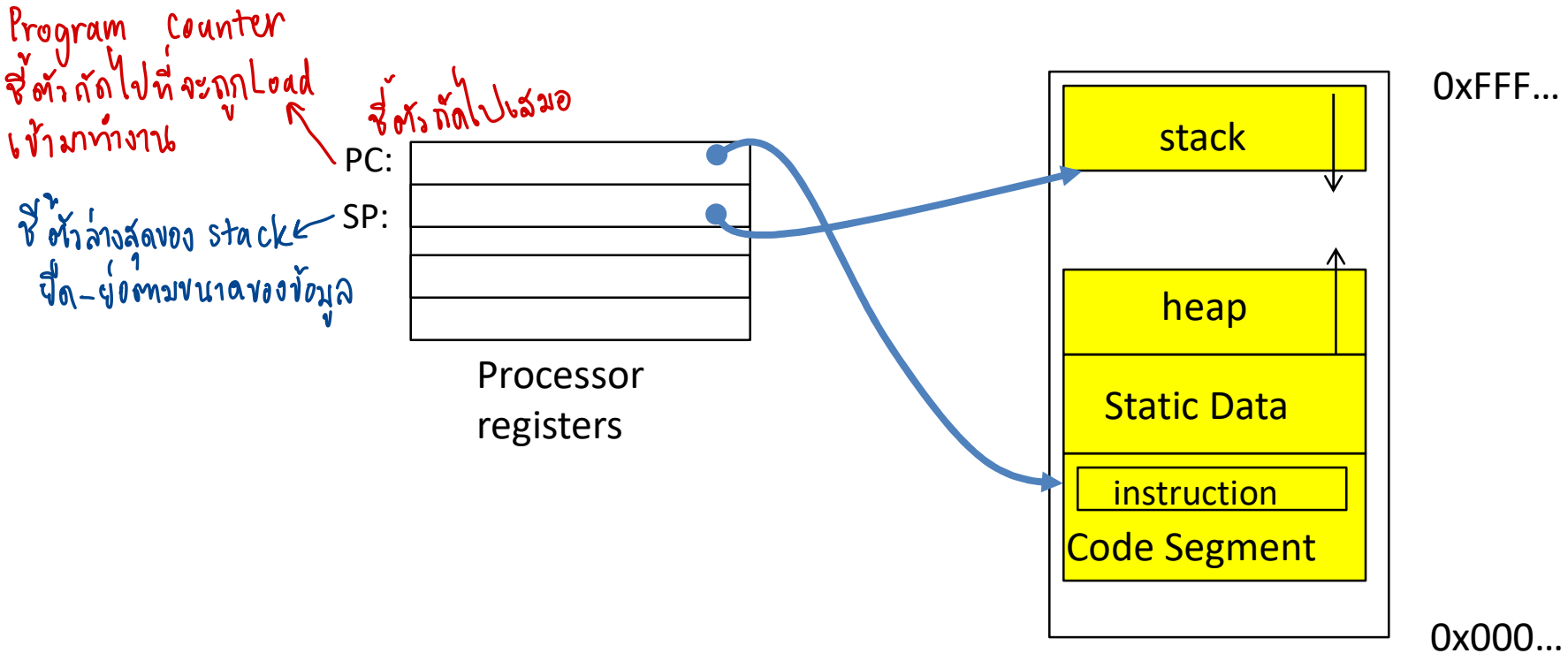
- 2 parts
  - PCB in kernel
  - Others in user



Process Control Block: PCB

- Kernel represents each process as a process control block (PCB)
  - Status (running, ready, blocked, ...)
  - Registers, SP, ... (when not running)
  - Process ID (PID), User, Executable, Priority, ...
  - Execution time, ...
  - Memory space, translation tables, ...
- Kernel Scheduler maintains a data structure containing the PCBs
- Scheduling algorithm selects the next one to run

Address Space: In a Picture



Break