政大資科系

作業系統

Operating System

廖峻鋒

cfliao@nccu.edu.tw

Operating System

Introduction

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

這一章說些什麼?

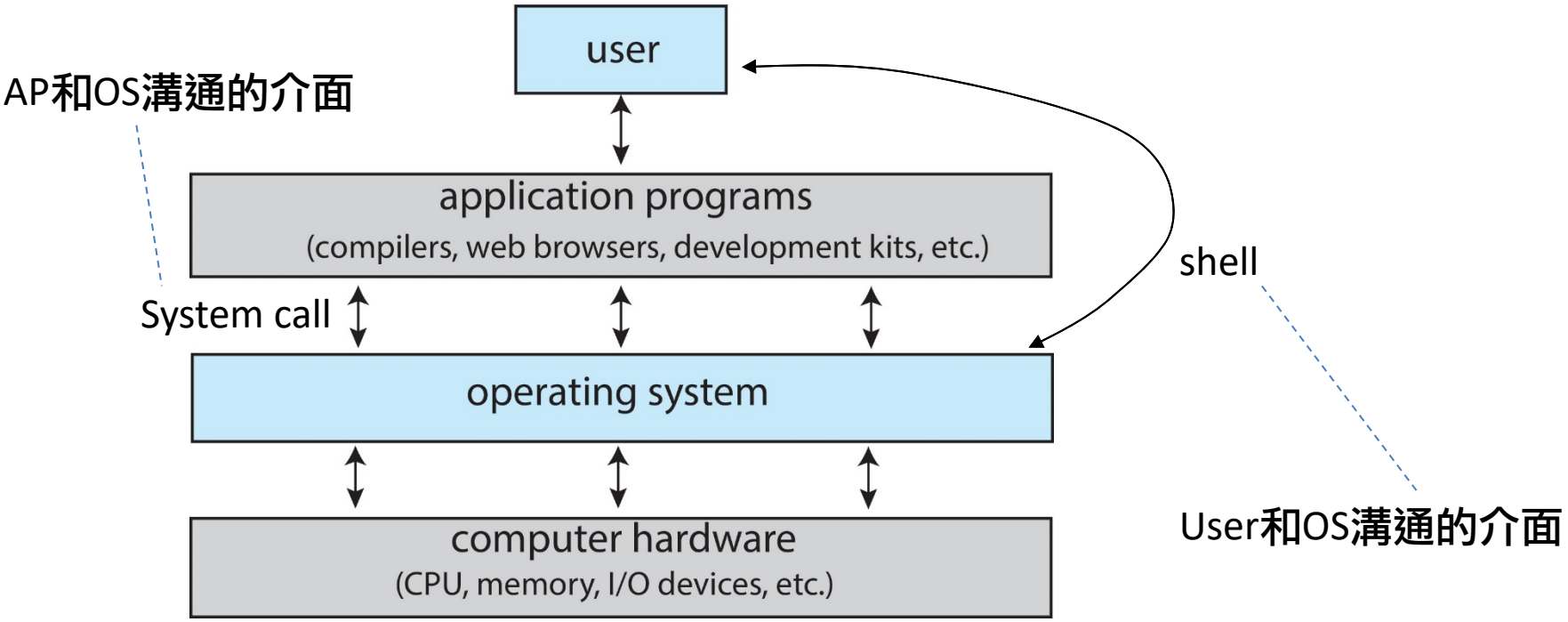
- 當代作業系統一般來說具備那些功能?
- 這些功能由那些元件實現?
- 這些元件又是如何分工合作來實現上述功能?

電腦系統的主要元素

- User
  - People, machines, other computers
- Application
  - Use the resources to fulfill users' needs
- Operating System
  - Control and coordinates the use of resources
- Hardware
  - Provides the resources
  - Ex: CPU time, memory, I/O devices

電腦系統 (Computer System)

- Main components
  - Hardware, OS, AP, User
  - Interfaces



作業系統 (Operating System, OS)

- Role
  - Abstractions of resources　例: UNIX中的File與Stream抽象
  - Manager of the resources
  - Controller of AP execution
- Objective
  - Convenient
  - Efficient/ Real-time
  - Robust
  - Secure

不同的情境對OS有不同要求

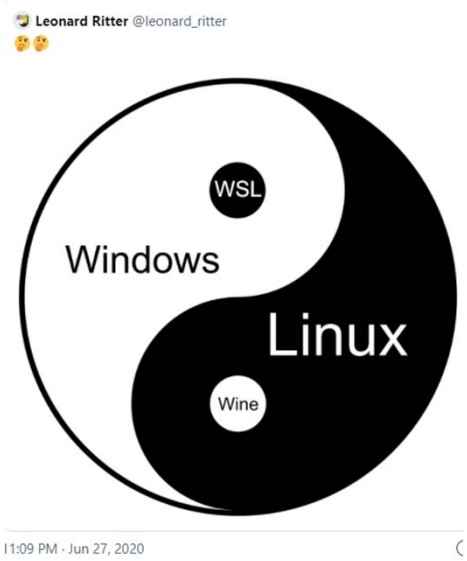
- PC (Single user)
  - Ease of use
  - Response time
- Mainframe (Multiuser)
  - Throughput
  - Fairness
  - Resource utilization
- Mobile devices (Mobile user)
  - Small UI / Voice recognition
  - Power consumption
- Robot/ satellite (mostly no user)
  - Minimal user intervention
  - Long running
- Container OS
  - Ease of deploy (container tools)
  - Ease of scale (etcd/fleet)
  - automatically-updating
  - compact
  - Ex: Fedora CoreOS

OS的定義

- No universally accepted definition
- Possible definitions
  - The software that provides common supporting operations
    - Managing and abstracting the resources: I/O, CPU, Memory
    - Controlling the program execution: security, error handling
- Main components
  - Kernel
  - System programs (系統程式)
  - Applications
  - (optional) middleware
    - Addition services to developers such as databases, multimedia, graphics

Operating System Examples

- Windows系列
  - DOS, Windows NT, Windows 10
  - 2019年6月之後，Windows 10中亦內建Linux核心 Subsystem for Linux version 2 (WSL2)
- UNIX系列
  - Solaris, AIX, HP-UX, BSD
  - Linux
  - Mac
  - Android



Why Study OS?

- Most SW engineers are not involved in the creation or modification of an OS
- Mechanical sympathy
  - Comes from J. Stewart (3 times F1 champion):

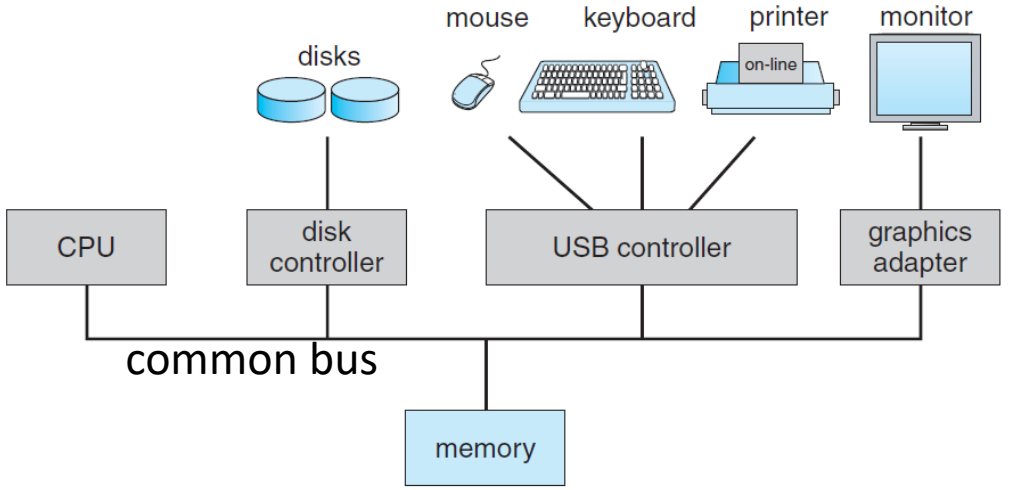
  "The best drivers had enough understanding of how a machine worked so they could work in harmony with it."

  - Thus, understanding OS is crucial to proper, efficient, effective, and secure programming

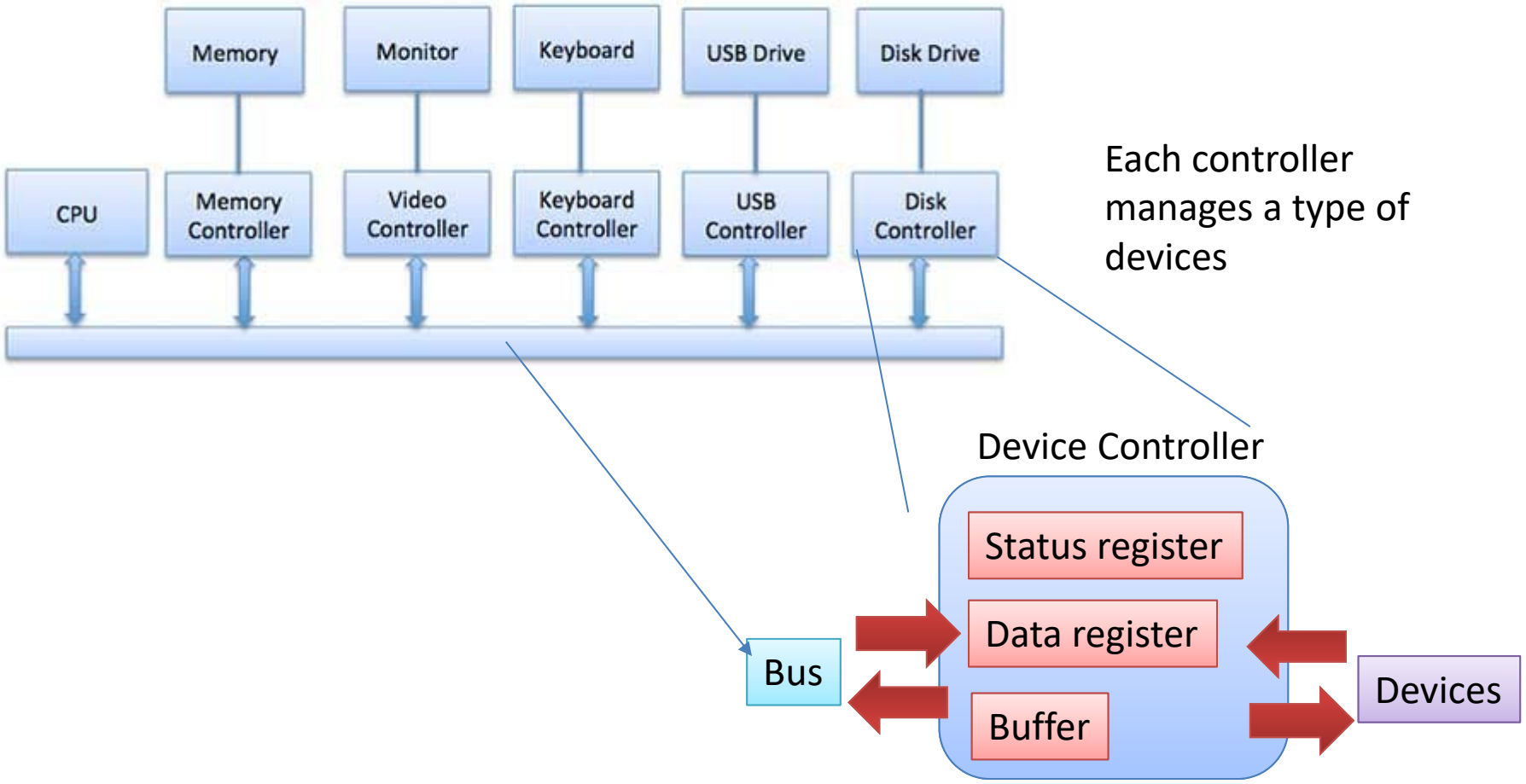
Computer System Organization

- CPU and device controllers
  - Each controller manages a type of devices
  - Device driver provides interfaces to the OS
- Connected by a common bus
  - The bus provides access to the shared memory
  - CPUs and devices competes for memory cycles

1.2重點:
- Interrupt
- 儲存系統
- DMA

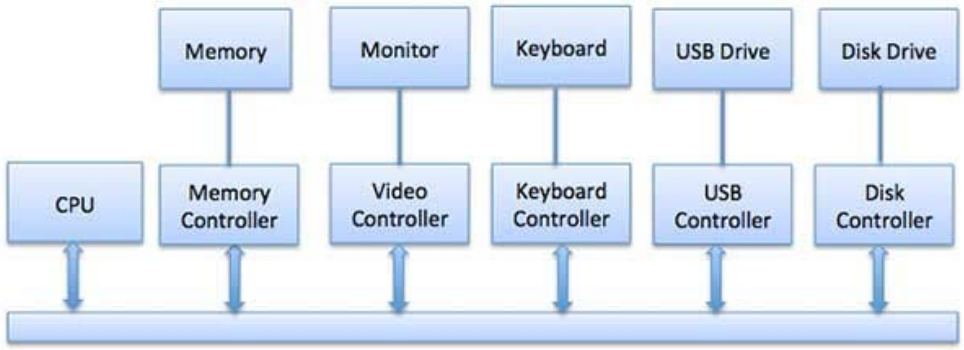


I/O Architecture



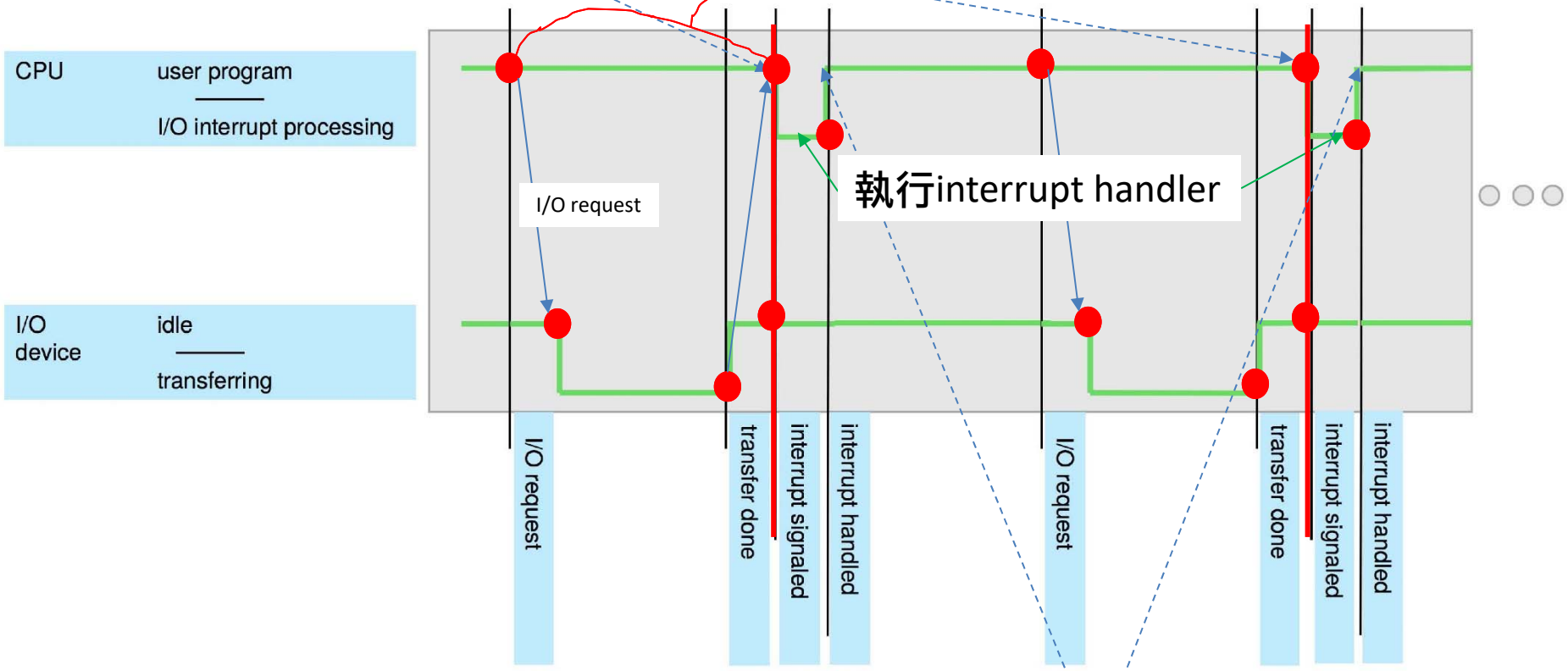
I/O Architecture

- 議題
  - CPU平常負責循序處理程式，也要處理來自I/O裝置的要求
    - 例如: 檔案讀取完成、記憶體錯誤、重大系統事件
  - 問題: CPU不知道I/O裝置那時候會發來要求?
    - 解法:「乾等」或「我先做別的事，好了叫我」
- 解法:
  - Poll (busy waiting)
    - I/O完成才做下一件事
  - Interrupt (event) driven
    - I/O後，就先做別的
    - Modern OSs are typically interrupt (event) driven



When the CPU is interrupted, it stops what it is doing and immediately transfers execution to the interrupt handler

如果是用poll的話，
這段時間都會浪費掉

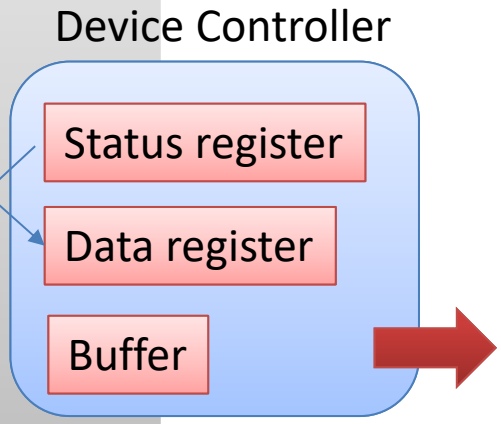


on interrupt completion, the CPU resumes the original computation

Poll (Busy Waiting)

- 先寫入一個char，接下來等它做完
- 等的過程: 不斷poll，直到status register顯示動作已完成

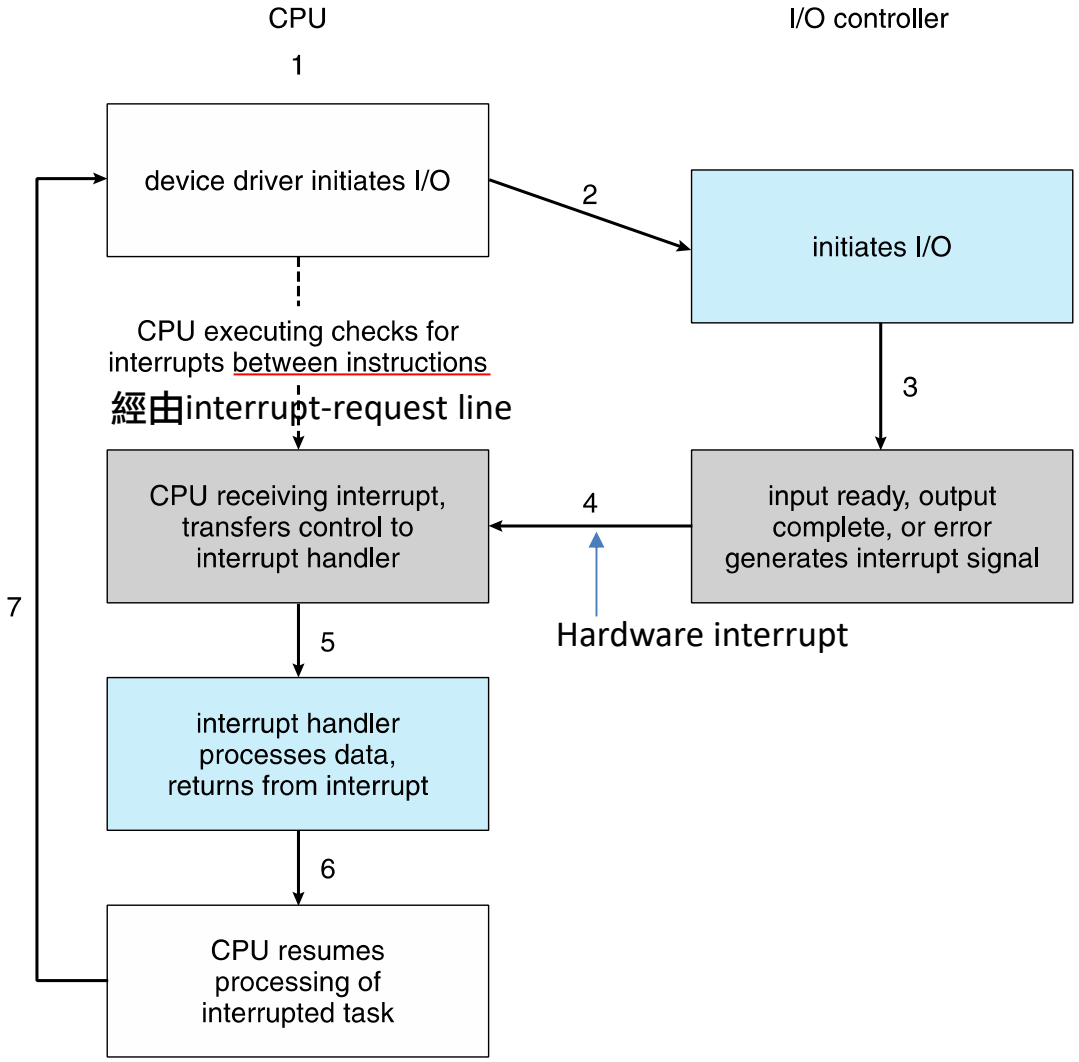
```
#define OUT_CHAR 0x1000 // device data register
#define OUT_STATUS 0x1001 // device status register
current_char = …(a string);
while (*current_char != '\0') {
    poke(OUT_CHAR,*current_char);  // 寫入資料
    while (peek(OUT_STATUS) != 0); // 等待回應
    current_char++;
}
```



Interrupt

- Mechanism
  - An event is characterized by an interrupt
    - HW: Devices send signals to CPU Via interrupt request lines (attached to CPU)
    - SW: Programs cause exceptions or issue system calls a.k.a. trap
  - Interrupt handling
    - Interrupt vector (IV): an array of pointers
    - IV contains pointers to the interrupt handlers (a.k.a. interrupt service routine, ISR)
    - IV is indexed by interrupt number; **目的**: speedup
      - Windows**與**Unix**都屬於此種方式**
  - Resuming
    - Save and restore the states of CPU
      - **正在執行的程式，會對**CPU**暫存器進行更動，後臨時切換到**Interrupt
      - **因為執行**interrupt service routine**時會動到**CPU**內暫存器的值，所以需要回復**
    - Save the address of the interrupted instruction (for return)

Interrupt 運作機制



Interrupt Implementation

- CPU 執行完每道指令後檢查interrupt request line
- 若有interrupt發生
  - (catch) Read interrupt number
  - (dispatch-1) Looking up the routine in interrupt vector
  - (dispatch-2) Jump to the interrupt-handler routine
  - (clear) 執行interrupt-handler routine

```
Head: Do one job (instruction)
while there are still events to process:
    e = get the next event
    handle e by jumping to IV functions
Goto Head
```

2. Lookup and run the interrupt handler

Interrupt vector (a table of pointers to the interrupt handler)

0	1	2	3	…	n
*Interrupt handler 0	*Interrupt handler 1	*Interrupt handler 2	*Interrupt handler 3		*Interrupt handler n

只包含特定interrupt handler的位址

1. Interrupt occurs

3. Perform I/O

```
int main() {
   …
   …
   int devfd = open(“/dev/random”, 0_RDONLY);
   …
}
```

當代Interrupt Handling的特色

- Realized by modern CPU and interrupt-controller HW
  - Prioritized and deferrable interrupt processing
    - 例如: critical section – 一批指令做完不能中斷，不然資料會錯誤
    - Maskable interrupts can be turned off by CPU temporarily
    - Non-maskable interrupts can not be preempted
  - Keeping interrupt vector small
    - 在一般32-bits PC中，記憶體前1024bytes用來放IV
      - 每個4bytes，共256個欄位
    - 超過的部份，使用interrupt chaining
      - 愈常用的放愈前面
  - Interrupt chaining
    - Interrupt vector points to the head of an ISRP list

ISRP=Interrupt Service Routine Pointers

Demo

- 看一下Linux中的Interrupts:
  - cat /proc/interrupts

```
$cat /proc/interrupts
           CPU0
  0:         51   IO-APIC   2-edge      timer
  1:          9   IO-APIC   1-edge      i8042
  4:       3154   IO-APIC   4-edge      ttyS0
  8:          0   IO-APIC   8-edge      rtc0
  9:          0   IO-APIC   9-fasteoi   acpi
 12:        156   IO-APIC  12-edge      i8042
 14:          0   IO-APIC  14-edge      ata_piix
 15:          0   IO-APIC  15-edge      ata_piix
 19:      84797   IO-APIC  19-fasteoi   enp0s3
 20:      96809   IO-APIC  20-fasteoi   ioc0, vboxguest
```

次數　Interrupt Controller　裝置名稱　介面

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Non-maskable

需要儘快處理

maskable

可配合排程

Figure 1.5 Intel processor event-vector table.

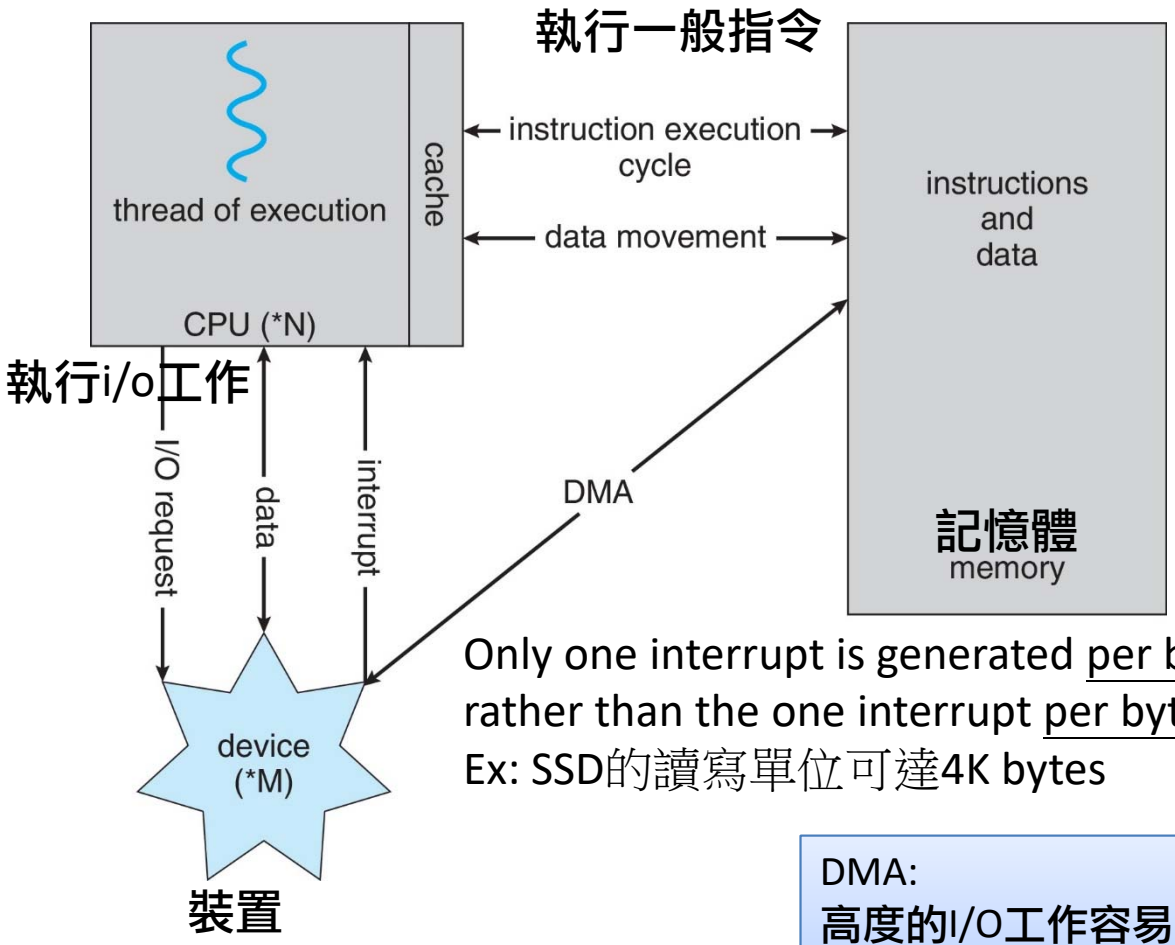
Interrupt Context

- 如果「中斷」被「中斷」怎麼辦?
- ISR會在稱為Interrupt Context的特殊環境下被執行
  - Interrupt Context是atomic context，在此環境下執行的程式會直接執行完，不被打斷

Top half interrupt/ bottom half interrupt

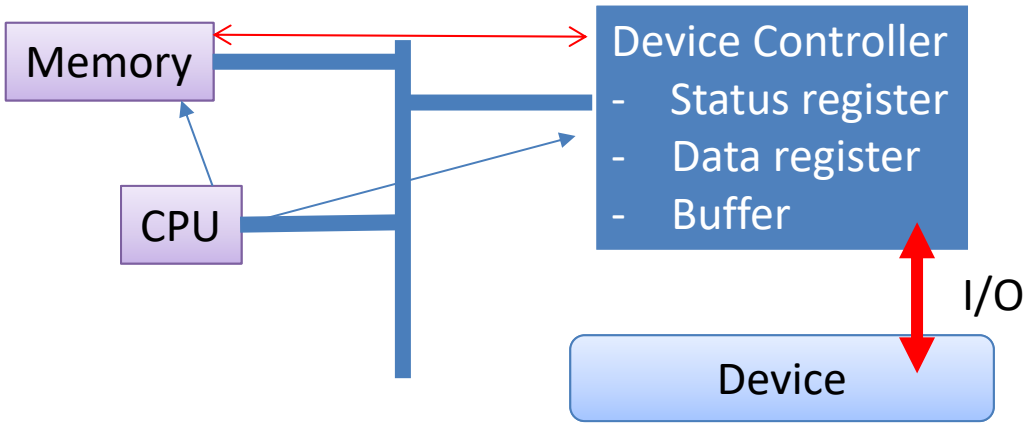
- 很多ISR所處理的事情難以在短時間內處理完成
  - 如: 網路卡: 回應硬體、複製封包到記憶體、傳送封包以特定的格式到特定應用程式、處理buffering
  - 中斷需要很快被處理，才能不打斷原進行工作太久
  - 做「很多」事和做事「很快」是trade-offs
- 解法
  - ISR分二段: 前段很快處理，後段在方便時再處理
    - 類似設計概念: L1、L2 cache (L1快而小;L2大但較慢)

DMA-based I/O Structure

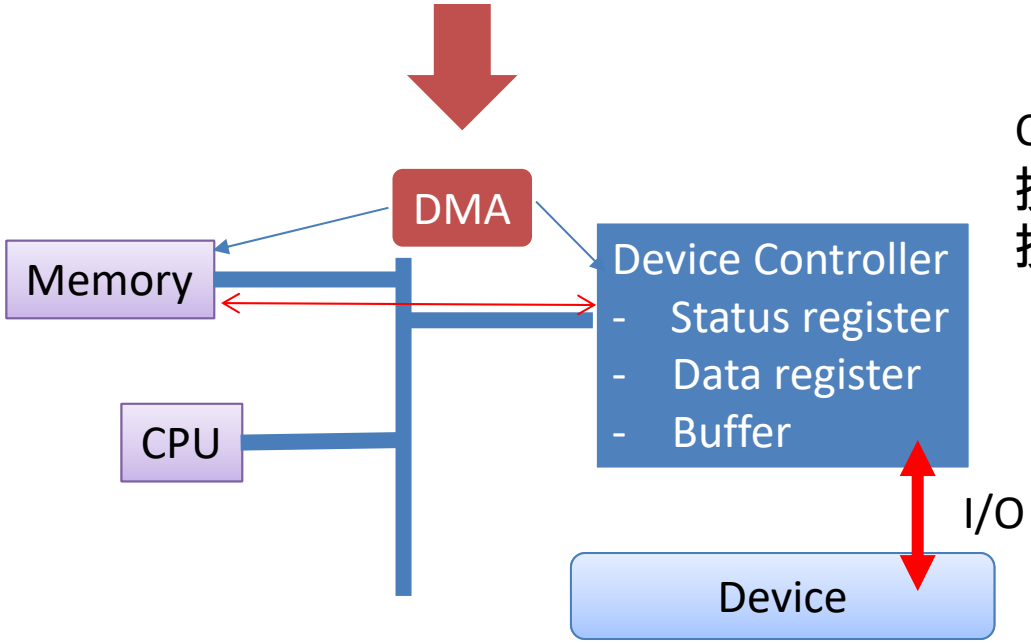


Only one interrupt is generated per block, rather than the one interrupt per byte
Ex: SSD的讀寫單位可達4K bytes

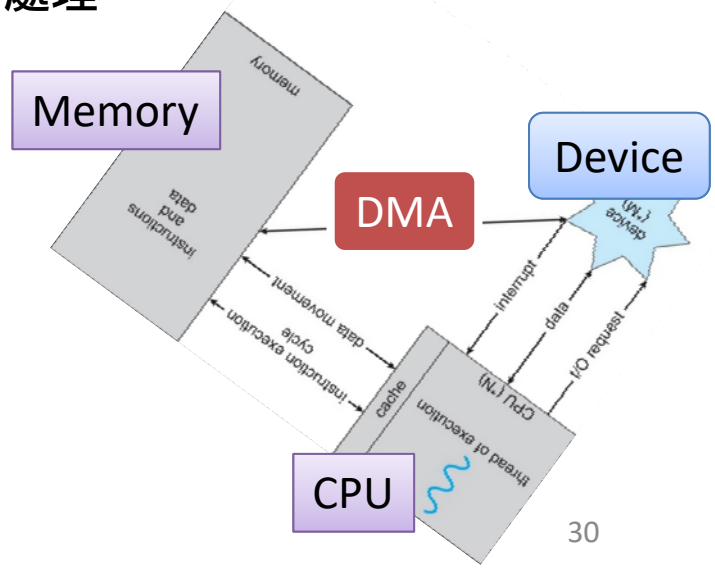
DMA:
高度的I/O工作容易耗掉寶貴CPU Time，因此，在CPU設定好相關參數後，裝置-記憶體間傳送資料的控制，由Device controller來代勞



原本每個byte都需要CPU處理



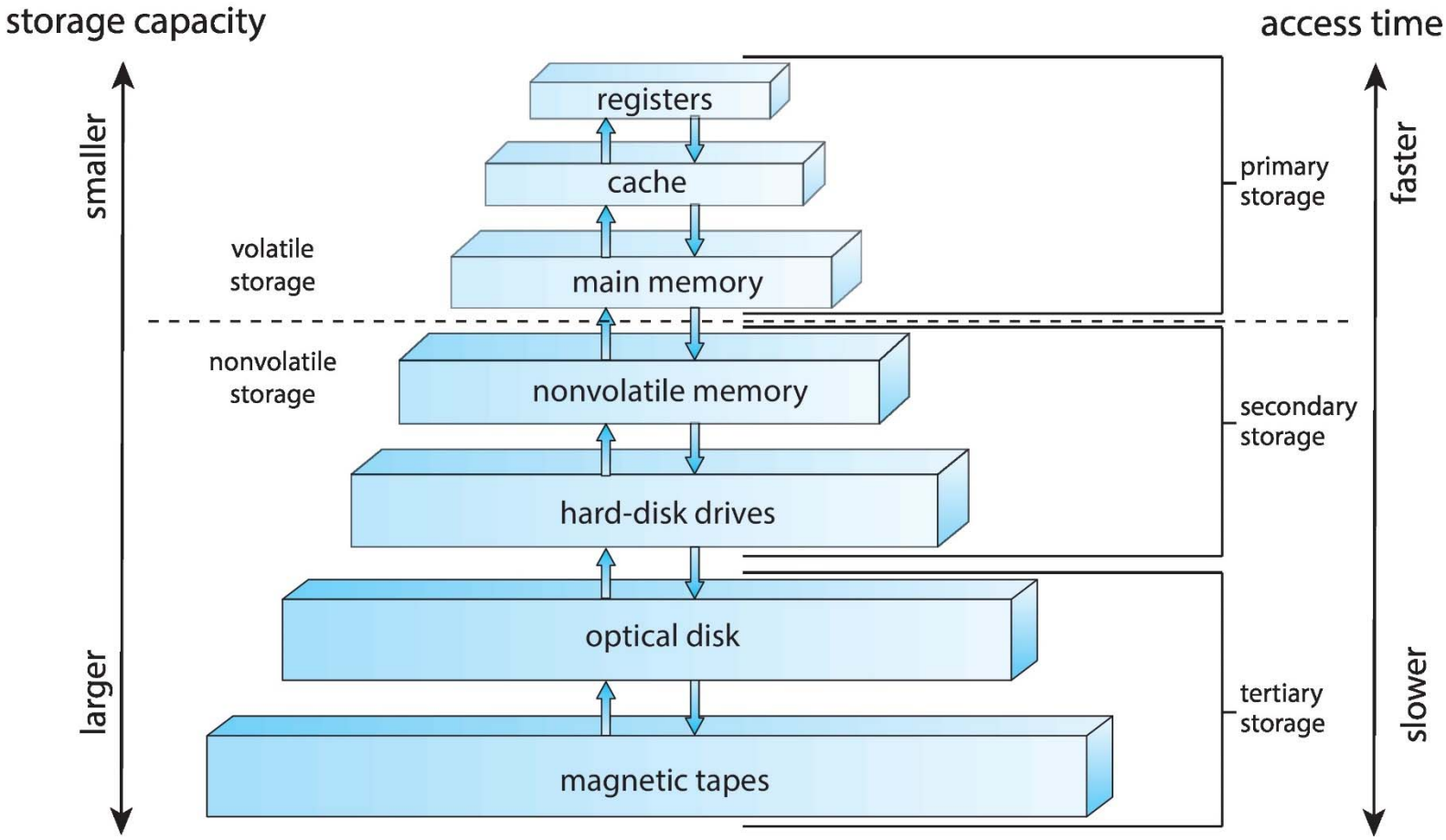
CPU先設定好從x位址，連讀y bytes;
接下來這y bytes (一個block) 都由DMA接手處理



Storage-Device Hierarchy

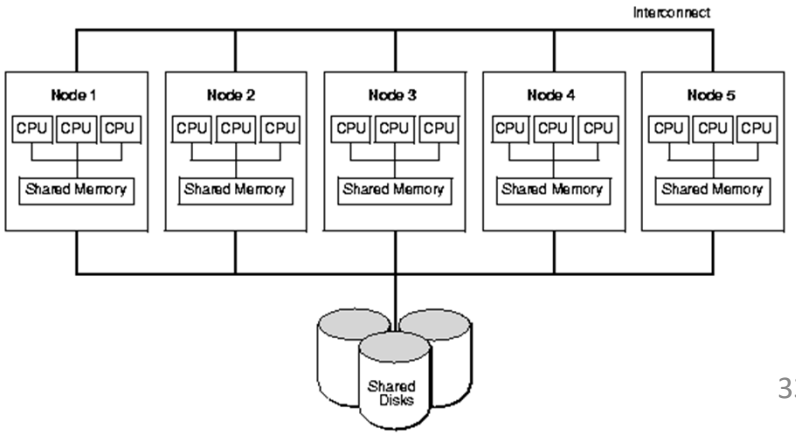
- Storage systems organized in hierarchy
  - Speed, cost, volatility
- Primary storage: register, cache, main memory
  - Small, expensive, and volatile
  - Ex: SRAM, DRAM
- Secondary storage: NVM and HDD
  - NVM: nonvolatile memory; HDD: hard-drive disk
  - Ex: Magnetic disk and SSD

Storage-Device Hierarchy

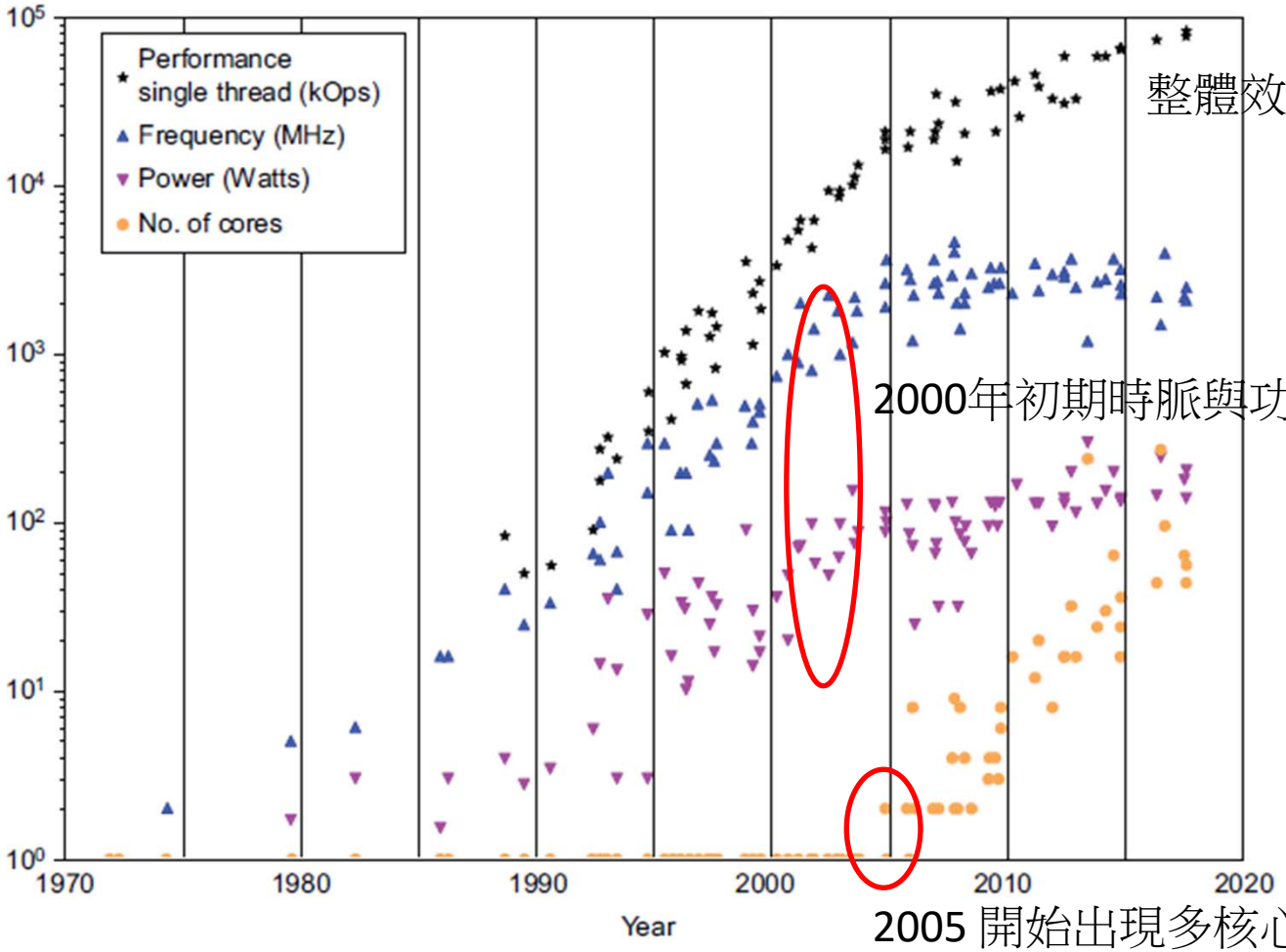


Computer-System Architecture

- Processor
  - A physical chip that (may) contains one or more CPUs
  - CPU: The hardware that executes instructions
- Categories
  - Single-Processor Systems
    - 1 processor; 1 CPU;
  - Multiprocessor Systems
    - One core (each p has 1 CPU)
    - Multicore (each p has n CPUs)
  - Clustered Systems



Technological Trends

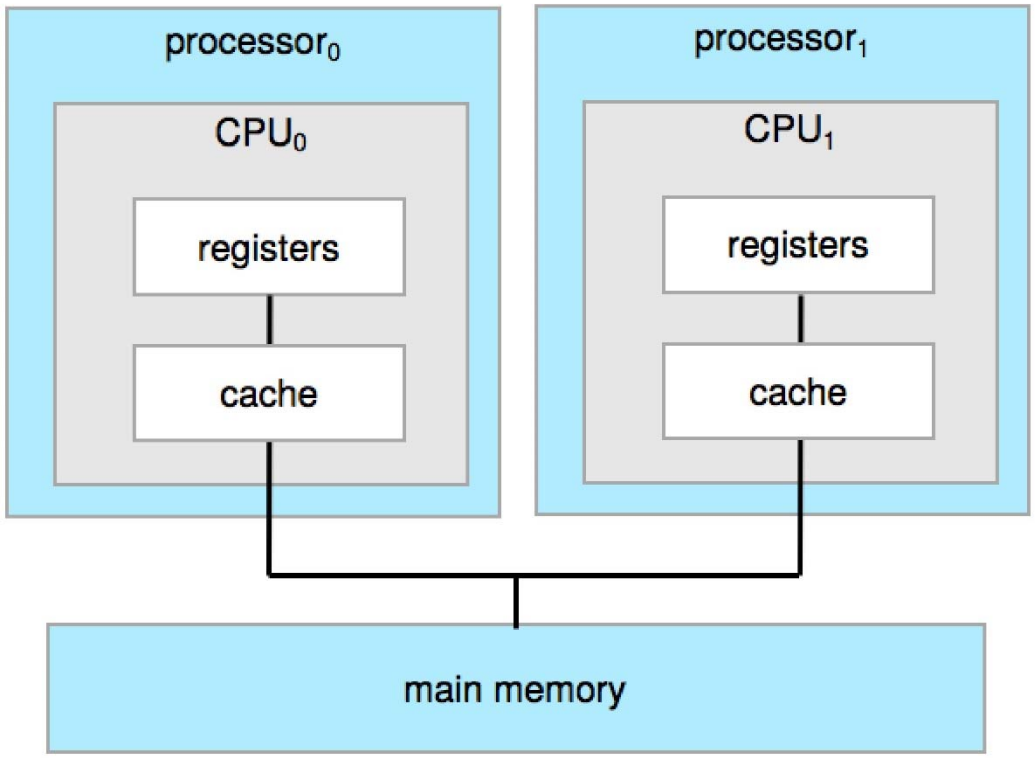


整體效能還是不斷有所提升

2000年初期時脈與功率就不再增加

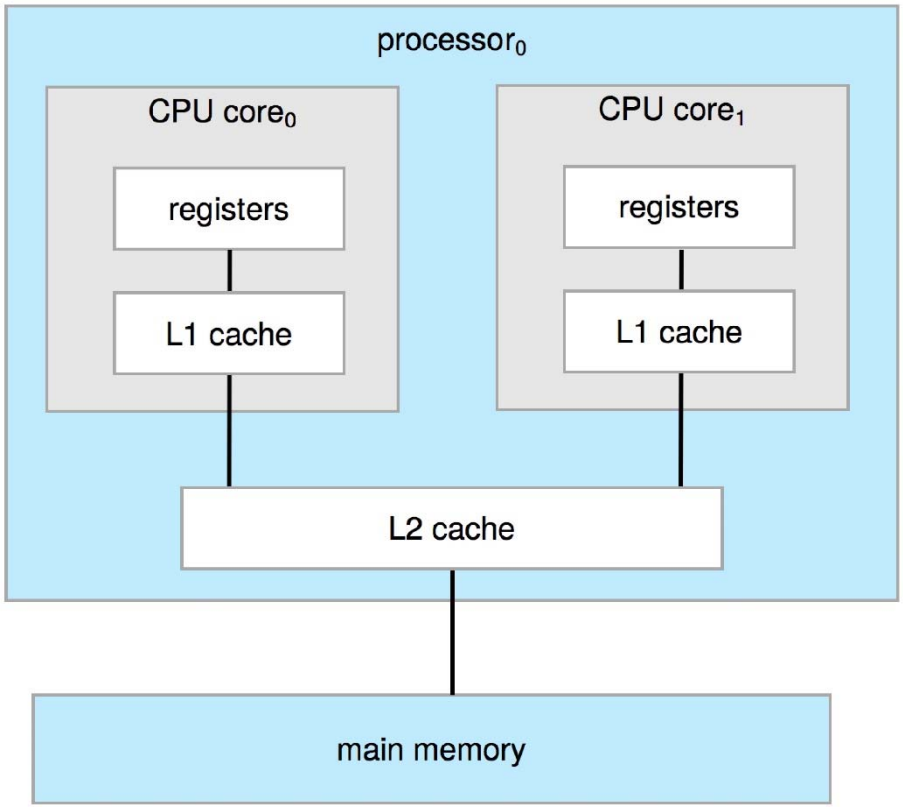
2005 開始出現多核心

Symmetric Multiprocessing (SMP) Architecture



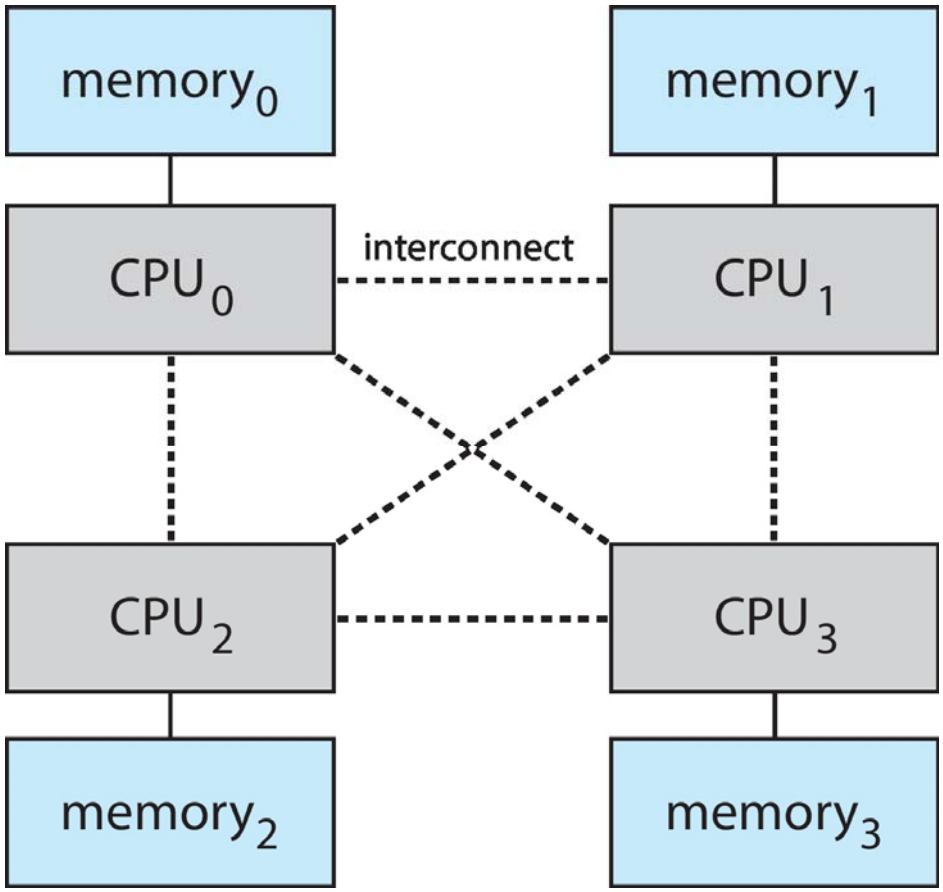
計算工作分散，但二個Processor會競爭同一資源(Main Memory)

Multi-core Architecture



計算工作分散，二個core仍會競爭同一資源(Main Memory)
但因為有Cache，所以效能比SMP好

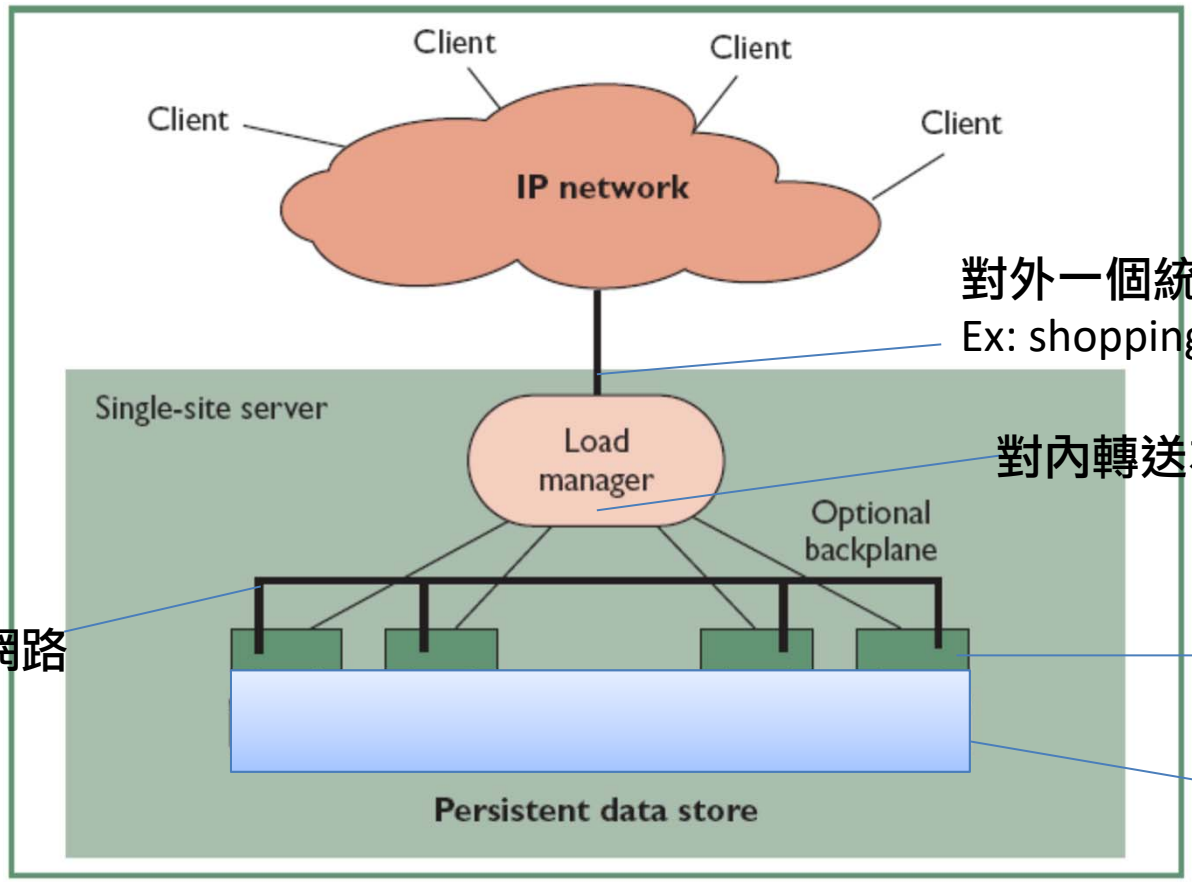
Non-Uniform Memory Access (NUMA) Architecture



每個CPU保有自己專屬memory，因此可解決SMP和multicore的競爭memory問題
但當CPU要用到另一CPU的memory中的資料時，就要跨CPU傳輸，因此記憶體管理設計要非常小心

Basic Model of a Clustered System

Client: 別如瀏覽器、手機程式等等，透過標準協定如JSON或SOAP存取服務



對外一個統一domain name
Ex: shopping.pchome.com.tw

對內轉送為多個內部機器IP

高速內部網路

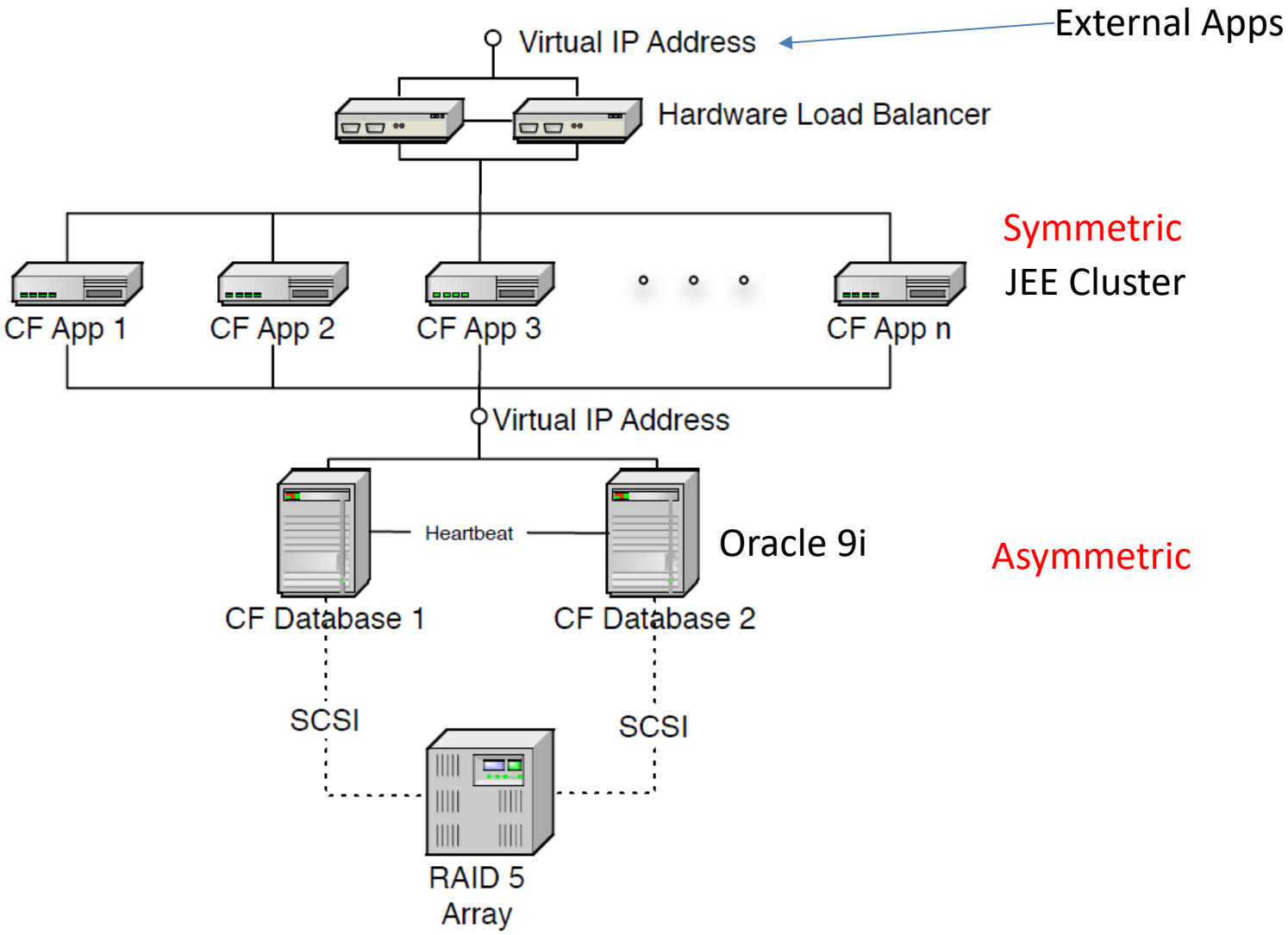
真正的伺服器

資料庫/SAN

Clustered Systems

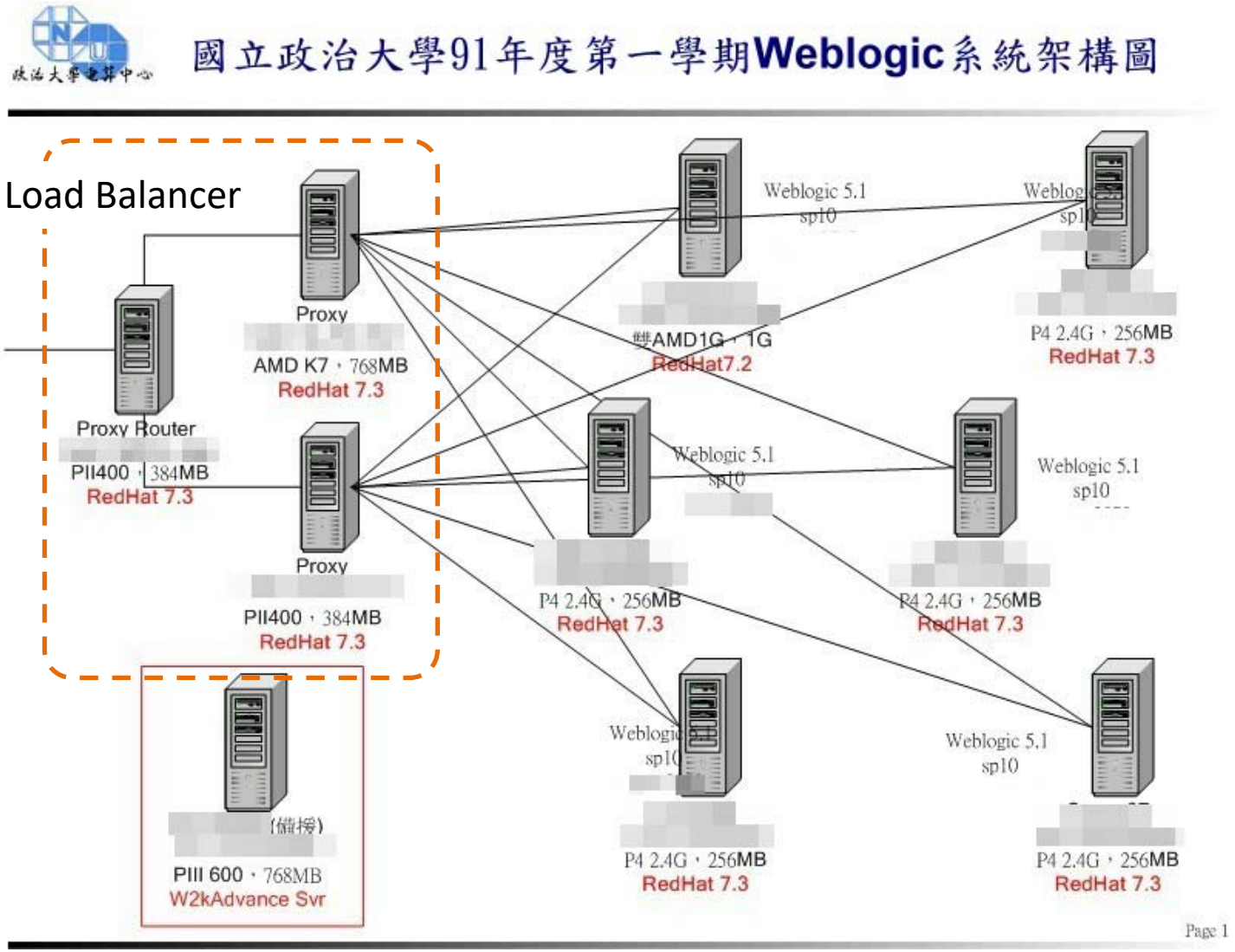
- Usually sharing storage via a storage-area network (SAN)
- Provides a high-availability service which survives failures
  - Asymmetric: one machine in hot-standby mode
  - Symmetric: multiple nodes monitoring each other
- Uses distributed lock manager (DLM) to avoid conflicting operations

Clustering Example



Example: Symmetric Clustering

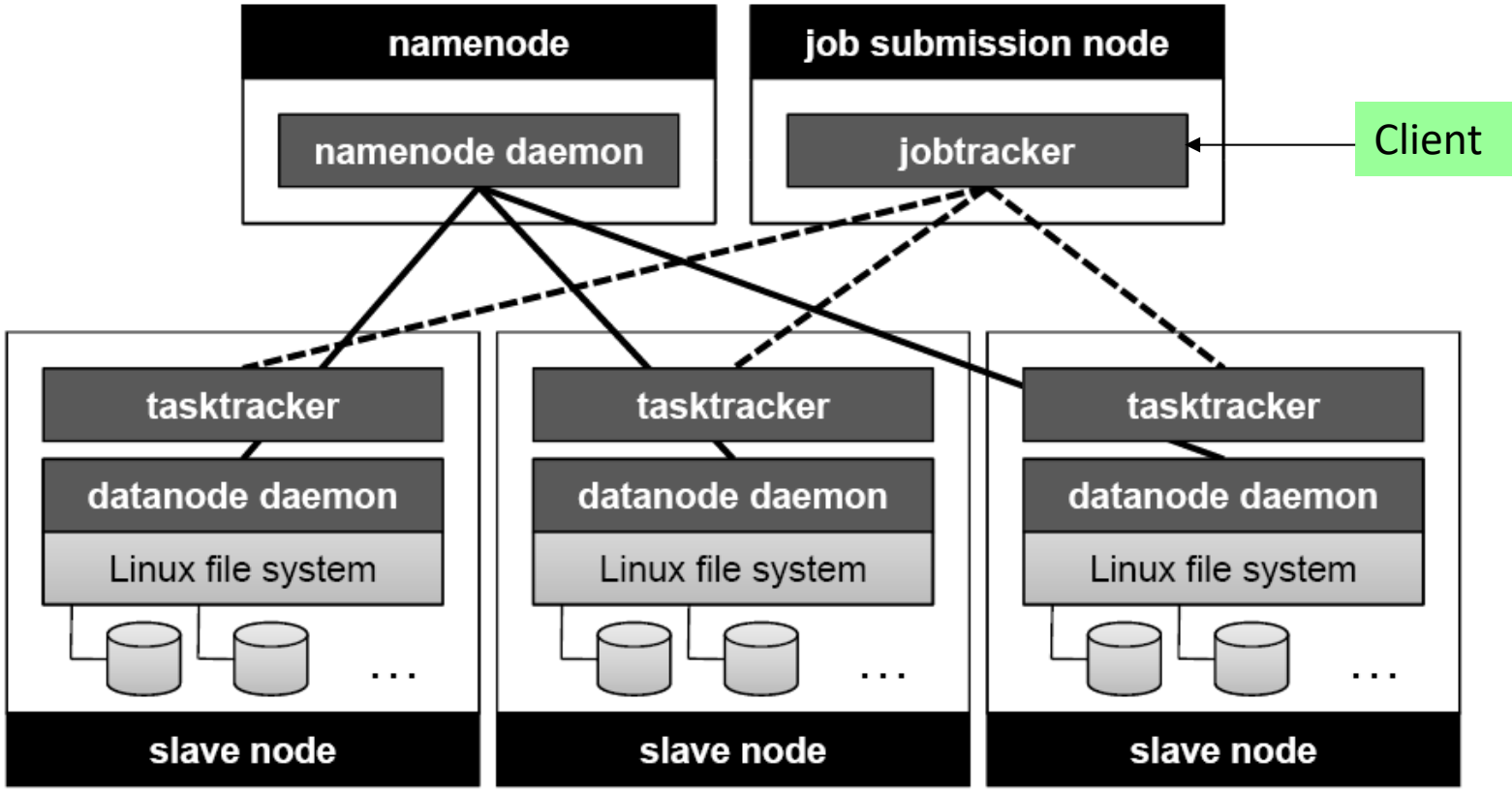
Software Load Balancer



Advantages of Clusters

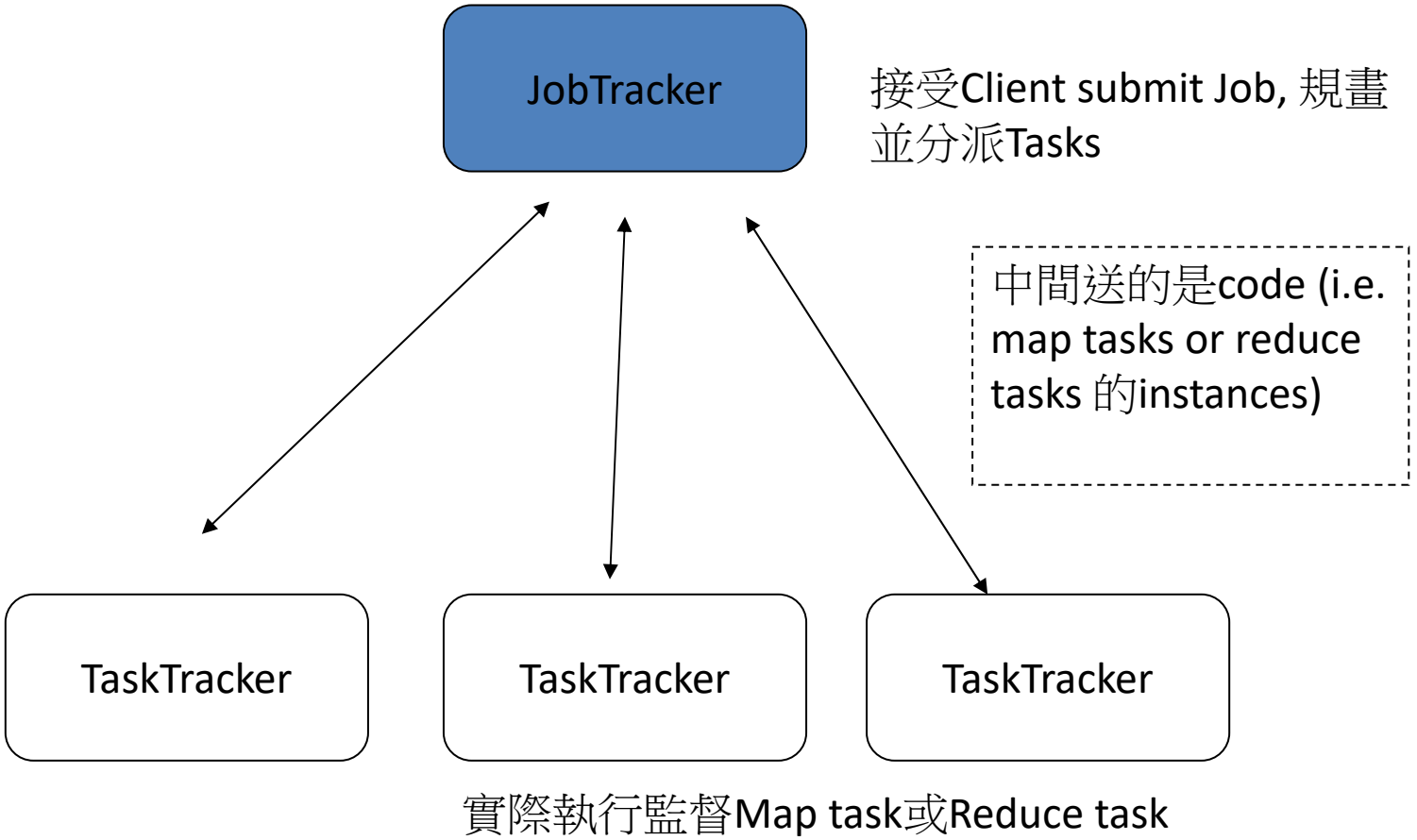
- 為何要cluster
  - 高擴充性 scalability
    - CPU/IO速度跟不上資料處理的需求
    - 因應漸增需求逐次擴充
  - 高可用性 High availability
    - 將節點失效的影響降到最低
  - 低成本 Low cost
    - 用較低價格PC或低階伺服器來達到高擴充性

Hadoop Cluster Architecture

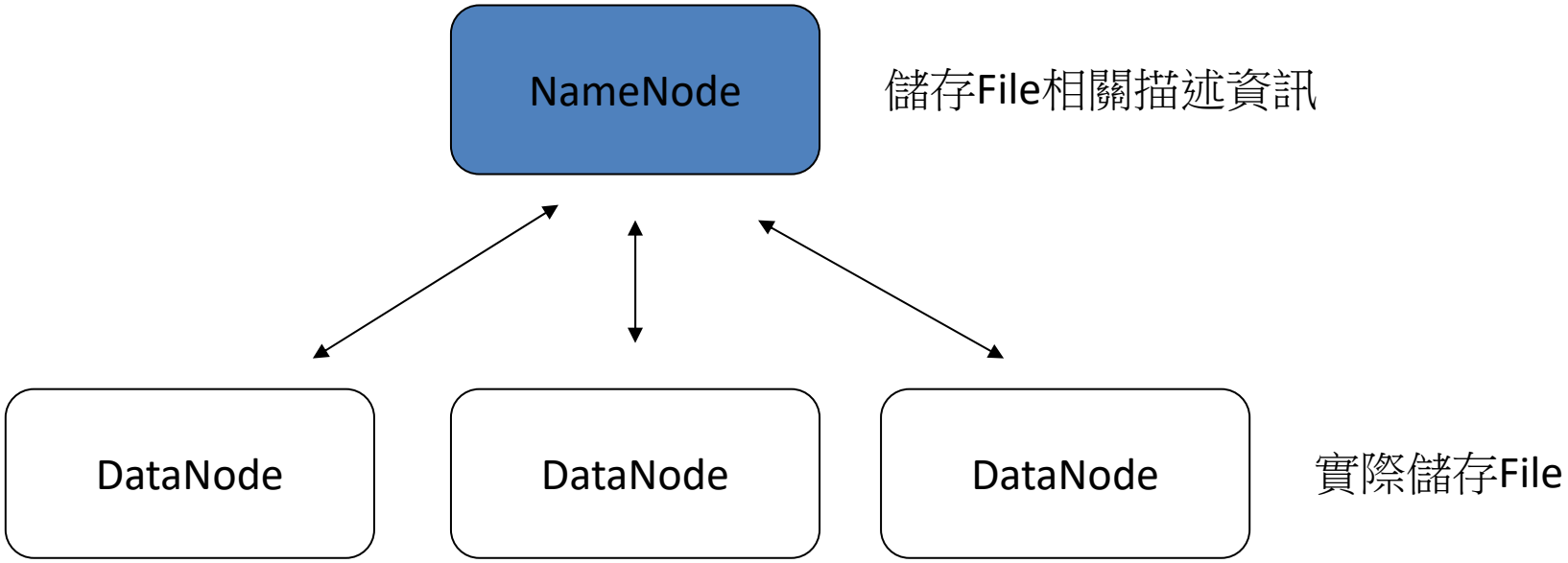


Jobtracker會儘可能讓map tasks處理local data
4000節點以上無法負荷 (c.f. YARN)

MR Architecture (Simplified)



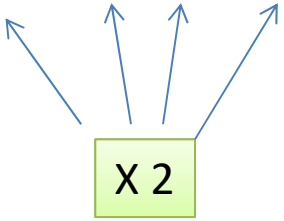
HDFS Architecture (Simplified)



Map

- Map
  - 對集合中的每個元素做特定的處理
  - 例

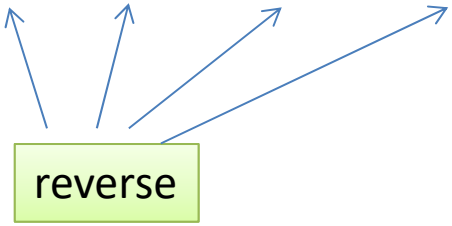
(1, 2, 3, 4)



結果

(2, 4, 6, 8)

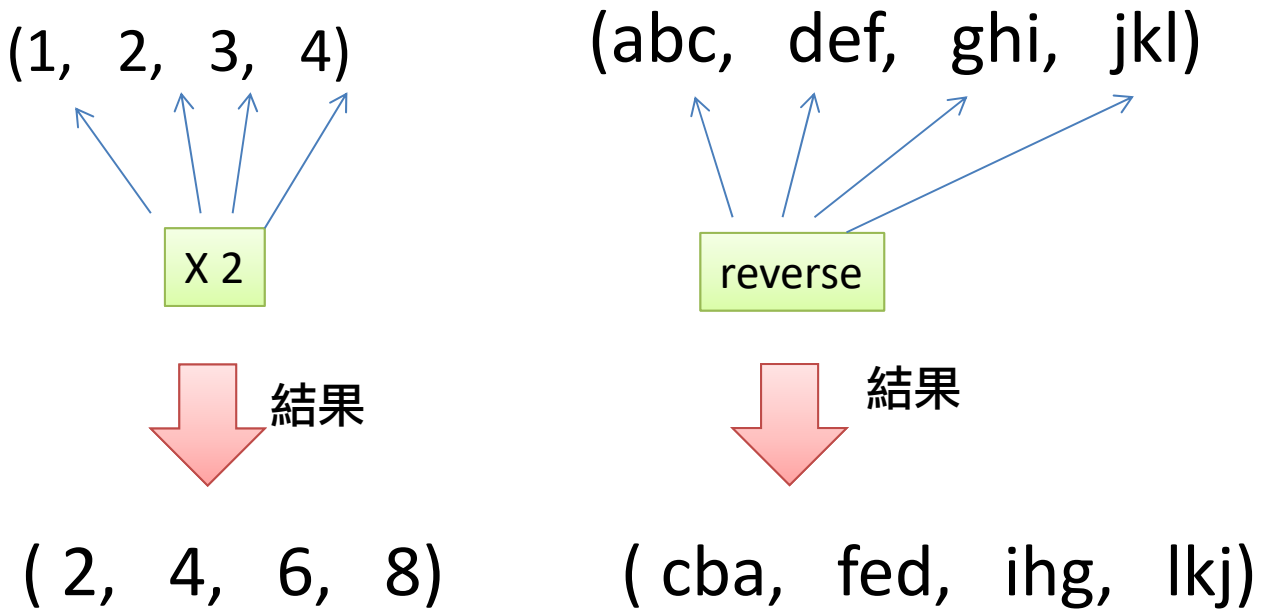
(abc, def, ghi, jkl)



結果

(?, ?, ?, ?)

Map



觀察: 將特定動作apply到每個元素時，先apply那個元素有沒有差?

例如 : 先做2x2 或是先做 2x4 對結果有影響嗎?

如果沒有影響，就代表這些工作是可以拆開來做 (平行處理)

Map

- Map

(006701199099999**1950**051507004...9999999N9**+0000**1+99999999999...,
0043011990999991**950**051512004...9999999N9**+0022**1+99999999999...,
0043011990999991**950**051518004...9999999N9**-0011**1+99999999999...,
0043012650999991**949**032412004...0500001N9**+0111**1+99999999999...,
0043012650999991**949**032418004...0500001N9**+0078**1+99999999999...)

取出每個row的二個字串，其中第二個字串化為整數

結果

([1950, 0],
[1950, 22],
[1950, -11],
[1949, 111],
[1949, 78])

Parallelizing Map

假設處理一行要花t時間, 處理5行要多久？

(0067011990999991950051507004...9999999N9+00001+99999999999...,
0043011990999991950051512004...9999999N9+00221+99999999999...,
0043011990999991950051518004...9999999N9-00111+99999999999...,
0043012650999991949032412004...0500001N9+01111+99999999999...,
0043012650999991949032418004...0500001N9+00781+99999999999...)

取出特定位置的二個字串，其中第二個字串化為整數

結果

([1950, 0],
[1950, 22],
[1950, -11],
[1949, 111],
[1949, 78])

Parallelizing Map

- Map

(006701199099999**1950**051507004...9999999N9**+0000**1+99999999999...,
 004301199099999**1950**051512004...9999999N9**+0022**1+99999999999...,
 004301199099999**1950**051518004...9999999N9**-0011**1+99999999999...,
 004301265099999**1949**032412004...0500001N9**+0111**1+99999999999...,
 004301265099999**1949**032418004...0500001N9**+0078**1+99999999999...)

取出特定位置的二個字串，其中第二個字串化為整數

結果

([1950, 0],
 [1950, 22],
 [1950, -11],
 [1949, 111],
 [1949, 78])

觀察: 將特定動作apply到每個元素時，先apply那個元素有沒有差?

Parallelizing Map

假設處理一行要花t時間, 5個CPU同時處理5行要多久？

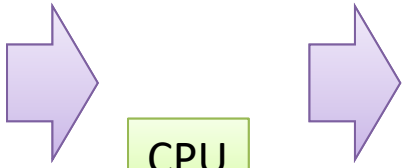
0067011990999991950051507004...9999999N9+00001+99999999999

0043011990999991950051512004...9999999N9+00221+99999999999...

0043012650999991949032412004...0500001N9+01111+99999999999...

0043012650999991949032418004...0500001N9+00781+99999999999...

0043011990999991950051518004...9999999N9-00111+99999999999...



CPU

CPU

CPU

CPU

[1950, 0]

[1950, 22]

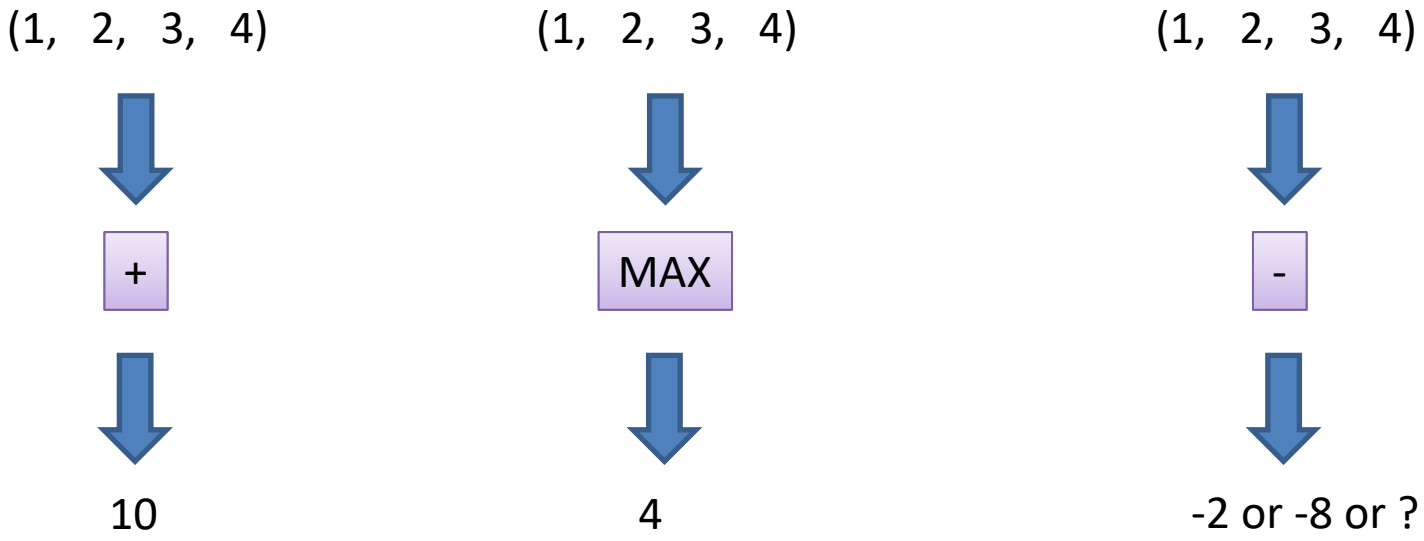
[1950, -11]

[1949, 111]

[1949, 78]

Reduce

- Reduce
  - 透過「特定的行為」，將集合中所有元素合併為較少元素



觀察: apply的先後次序有沒有關係?

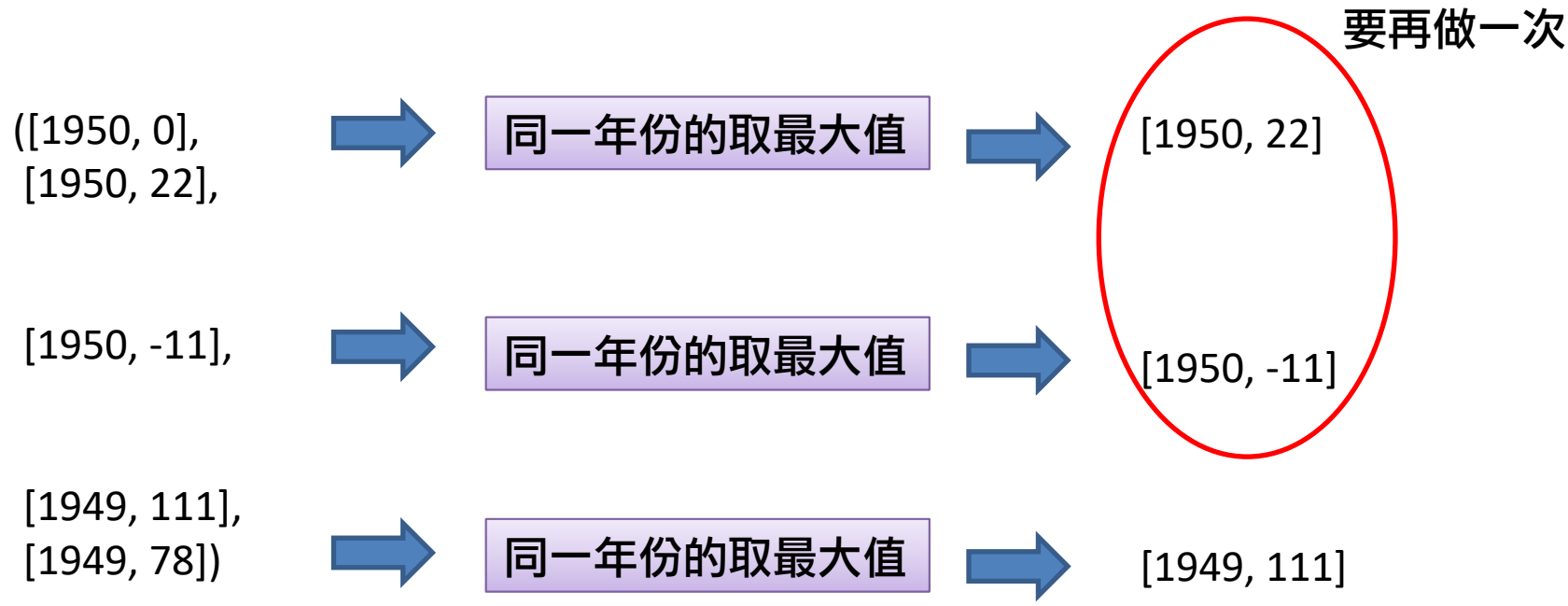
Reduce Example

([1950, 0],
[1950, 22],
[1950, -11],
[1949, 111],
[1949, 78])

同一年份的取最大值

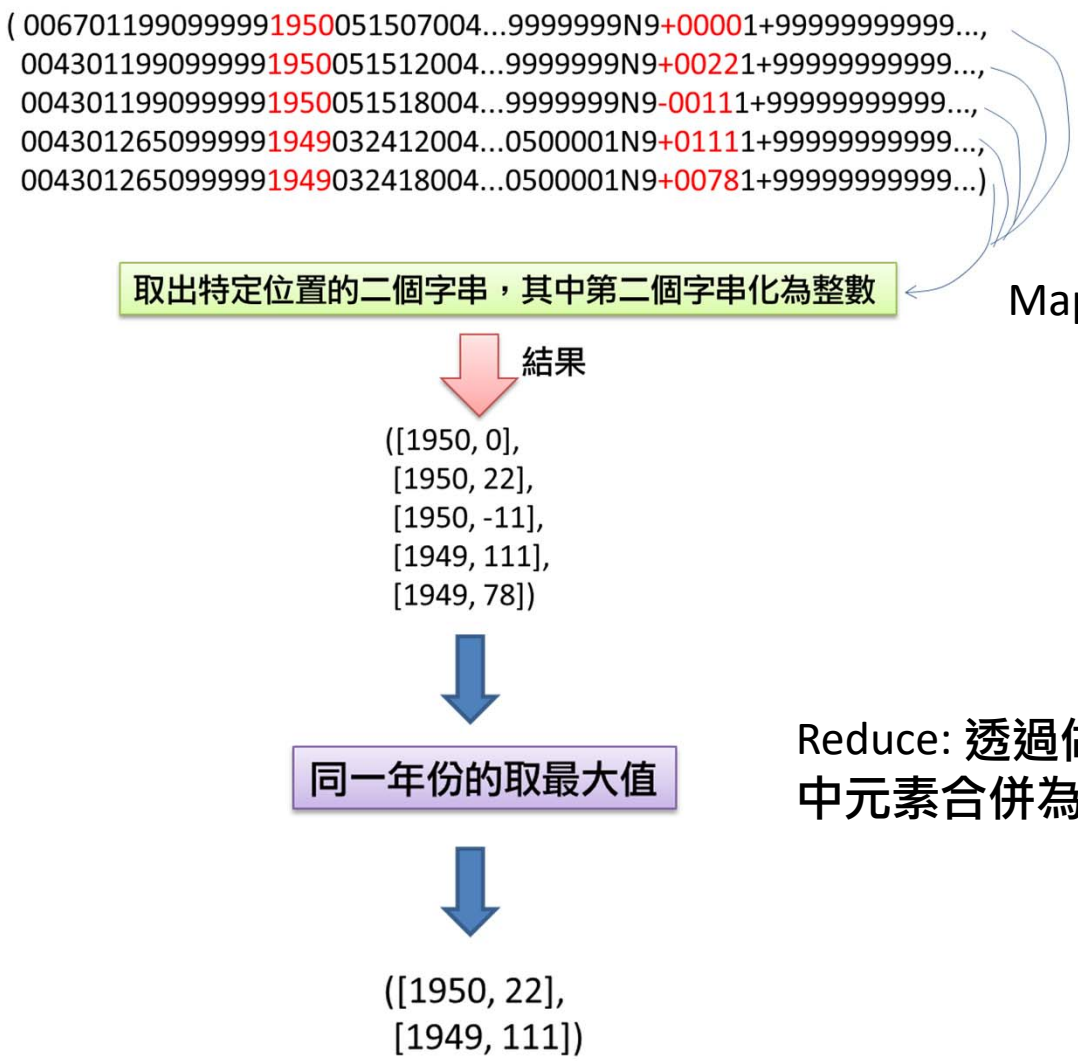
([1950, 22],
[1949, 111])

Parallelizing Reduce



所執行的動作必須符合結合律 (和結合的先後次序無關)

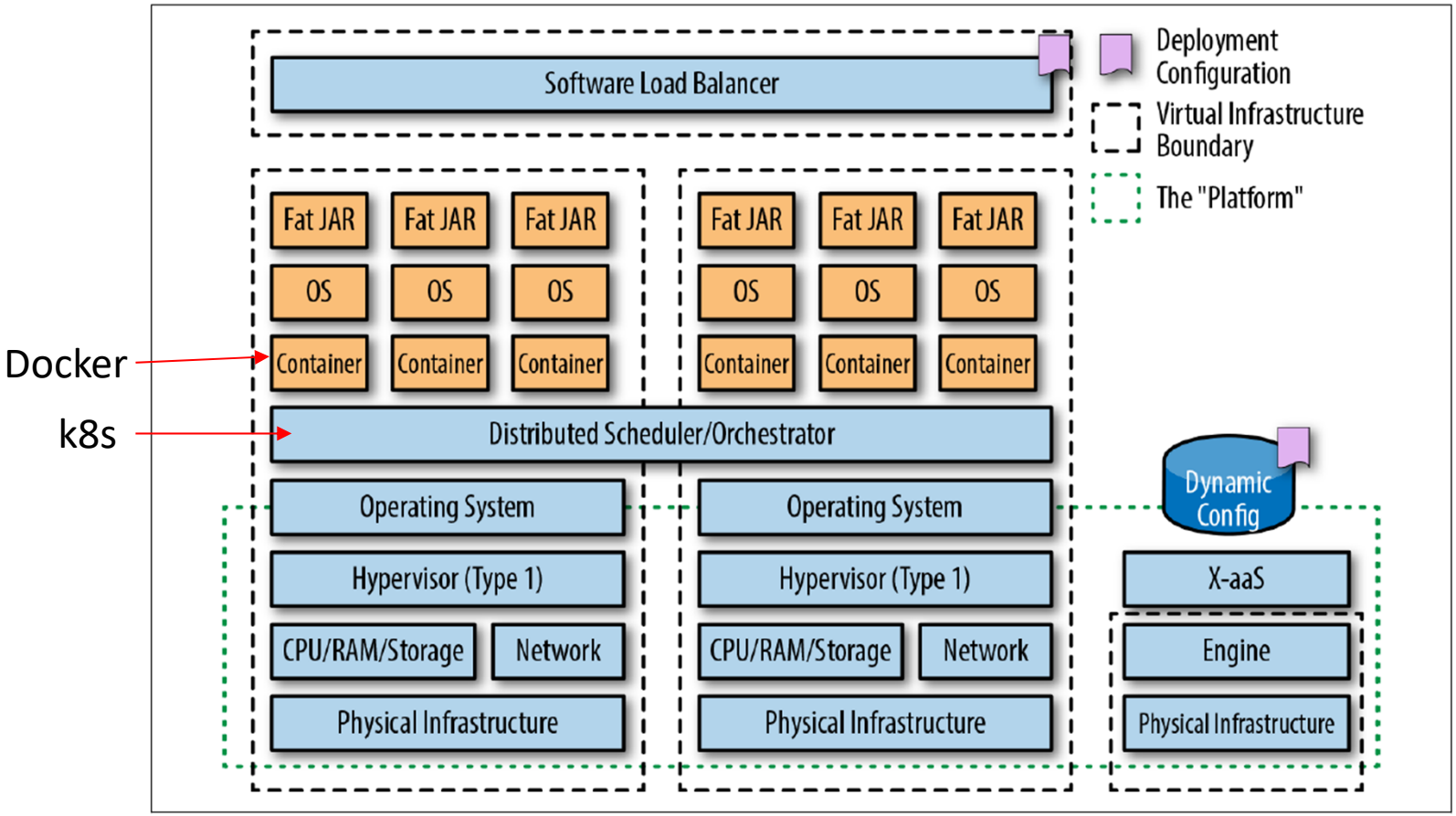
結合Map和Reduce



Map: **對每個元素做一件事**

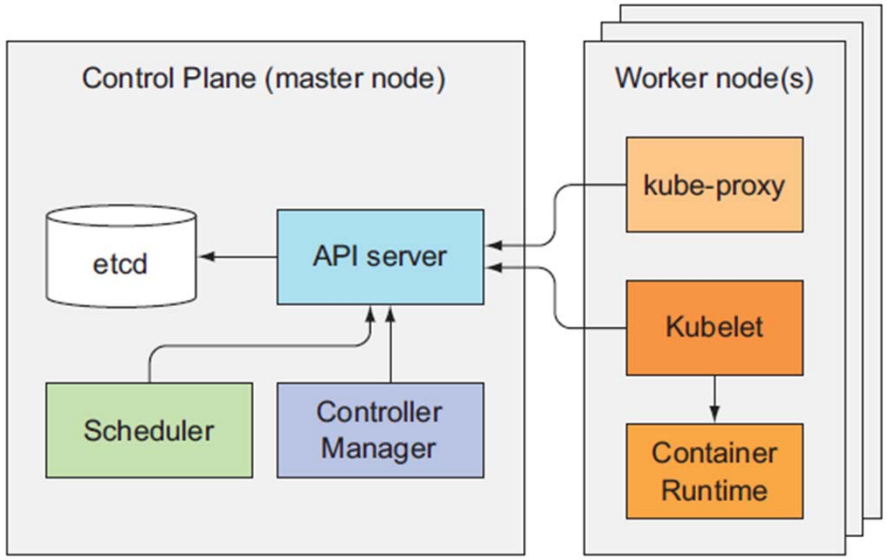
Reduce: **透過做「特定的事」，將集合中元素合併為較少元素**

Cloud Native 系統服務的運行



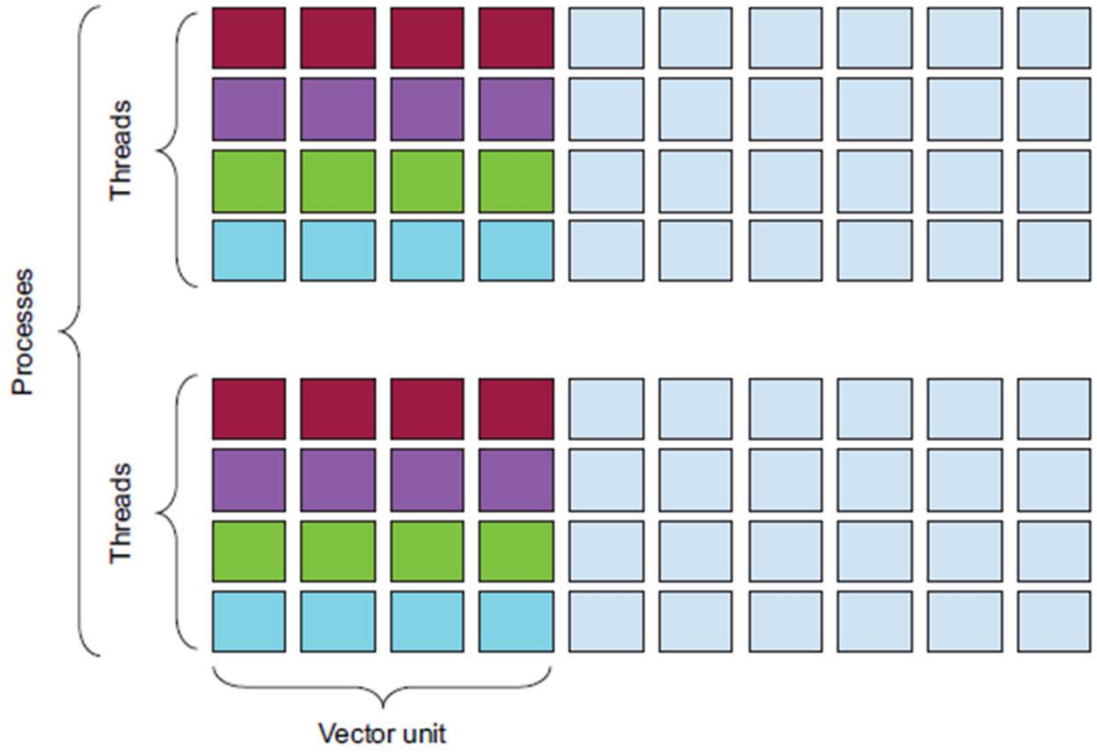
Kubernetes Infrastructure

- Master Node: container cluster control pane
  - Kube-apiserver
  - etcd (KV store)
  - kube-scheduler (deploy new container to pods)
  - Kube-controller-manager
- Worker Node
  - Kubelet (local manager of pods)
  - Kube-proxy (network mapping)
  - Container runtime



K8s上所有的系統管理元件不直接溝通; 而是透過API Server溝通
Etcd也只被API Server維護
Master node上的系統元件，也可以變成pod方式運行
(此時master node上也要運行kubelet)

Process, Threads, and Vector

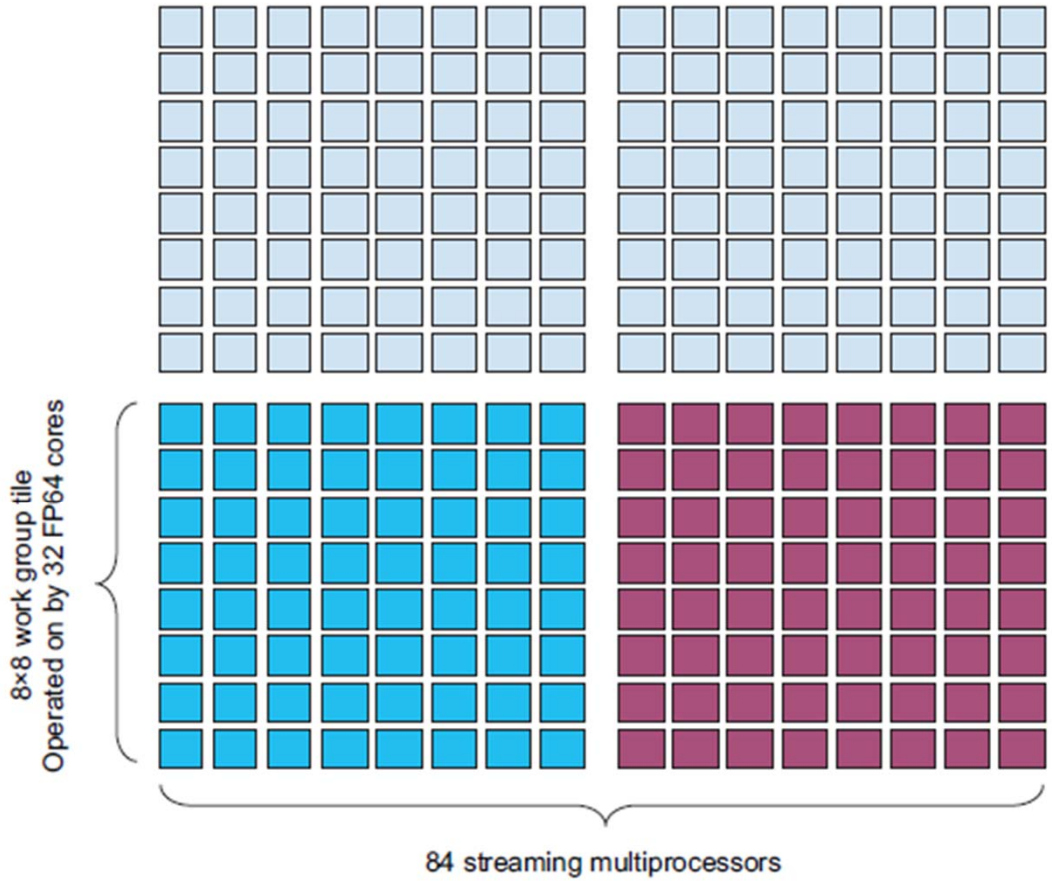


Process: 透過不同Process，可將工作分配到不同的實體電腦節點(Node)

Thread: 一個Thread 有一個PC，可配置至一個core

Vector: 同時執行多道指令

GPU



GPU具有相當大的vector
可一次平行處理大量運算指令

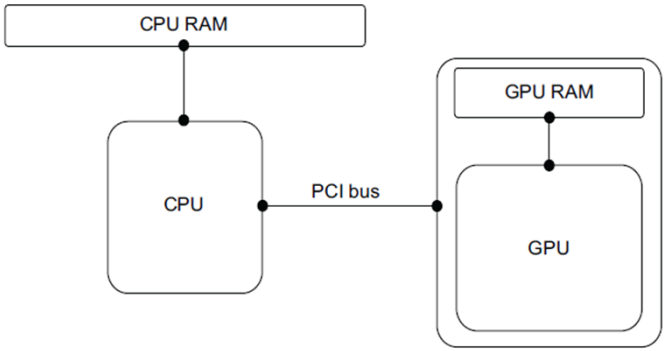
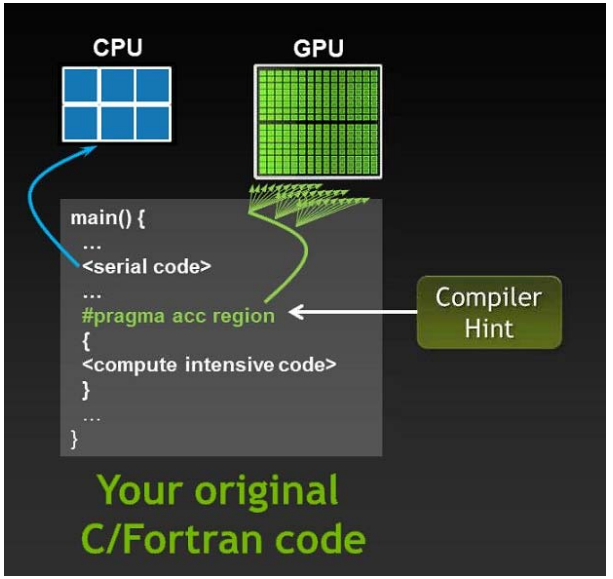


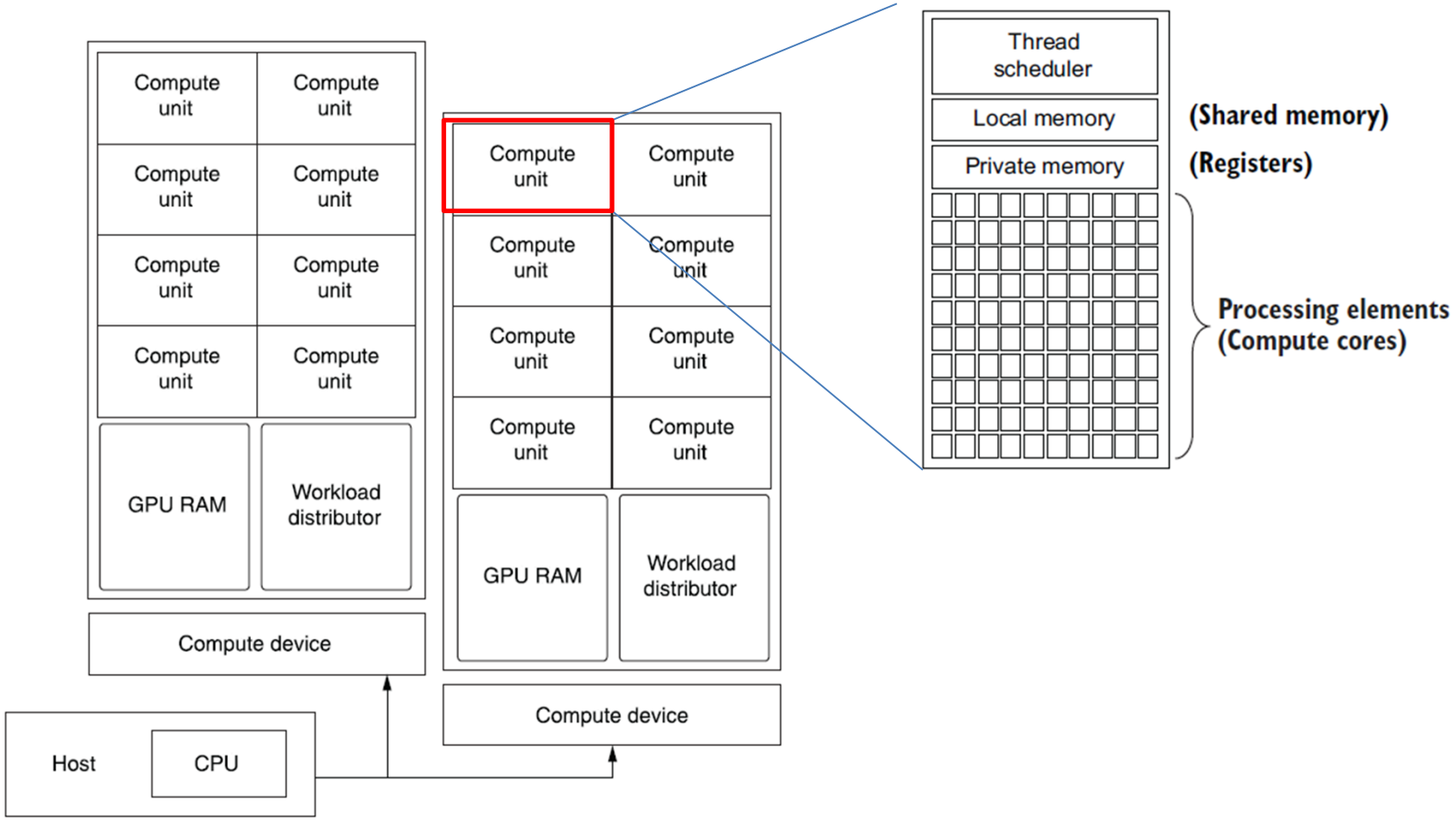
Figure 1.14 On a GPU, the vector length is much larger than on a CPU. Here, 8×8 tiles are distributed across GPU work groups.

GPU用來處理圖形運算以外的工作

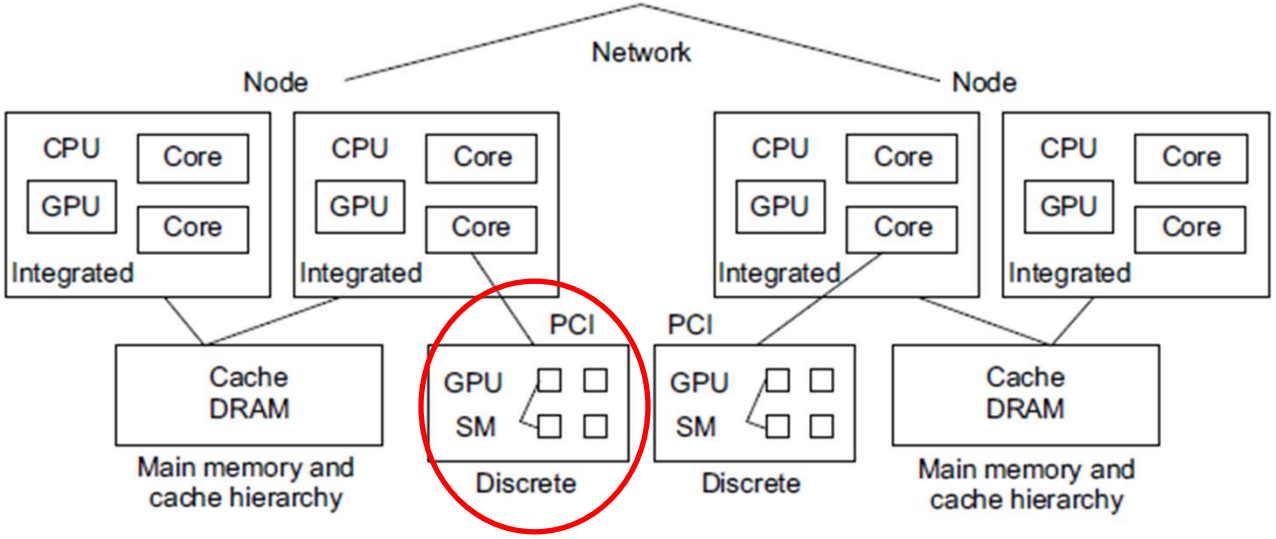
- GPGPU
  - General-Purpose Graphics Processing Unit
  - 將一般性、非圖形處理的工作也交給GPU處理
- 問題
  - 要先將非圖形處理的工作轉換成類圖形處理的型式
  - 需要不同的Programming Model
    - CUDA (Compute Unified Device Architecture)
      - A special purpose language for NVIDIA GPUs
    - OpenCL (Open Computing Language)
      - A group led by Apple
    - Directive-based API
      - OpenACC : Open Accelerators
      - OpenMP: Open Multi-Processing



Overview of a GPU System



平行運算硬體配置



一個GPU含多個SM (Streamed Multiprocessor)

Operating-System Operations

- Daemons/ system calls
- System startup
- Multiprogramming/ multitasking
- Dual-mode/ multi-mode
- Security and protection
- Virtualization

Terms

- Daemons (Services)
  - System services provided outside of the kernel
  - Loaded at boot time
  - Run the entire time when the kernel is running
  - Ex: linux initd (now systemd)
- System calls and API
  - A specific piece of code and APIs provided by OS to raise “traps”
    - Traps: software generated interrupts
  - Purpose: access services provided by OS
    - Ex: Create processes, access files, control devices, communications
  - More on Section 2.3 (P.62-P.74)

BIOS與UEFI

- BIOS
  - 最早在1982年為Intel 8088 16bits CPU(記憶體256K)電腦所設計
  - 限制
    - Boot Sector限制512bytes以內，難以支援GUI、滑鼠
- UEFI
  - 1998年: Intel Boot Initiative (IBI)→EFI;
  - UEFI(Unified Extensible Firmware Interface)本身就是一個(小)OS
    - 可驅動Audio/Video
    - 可驅動網路卡
    - 可驅動USB (Mouse)
    - 提供Rich GUI

系統啟動 (1)

More on Section 2.9.1, P.94-P.95

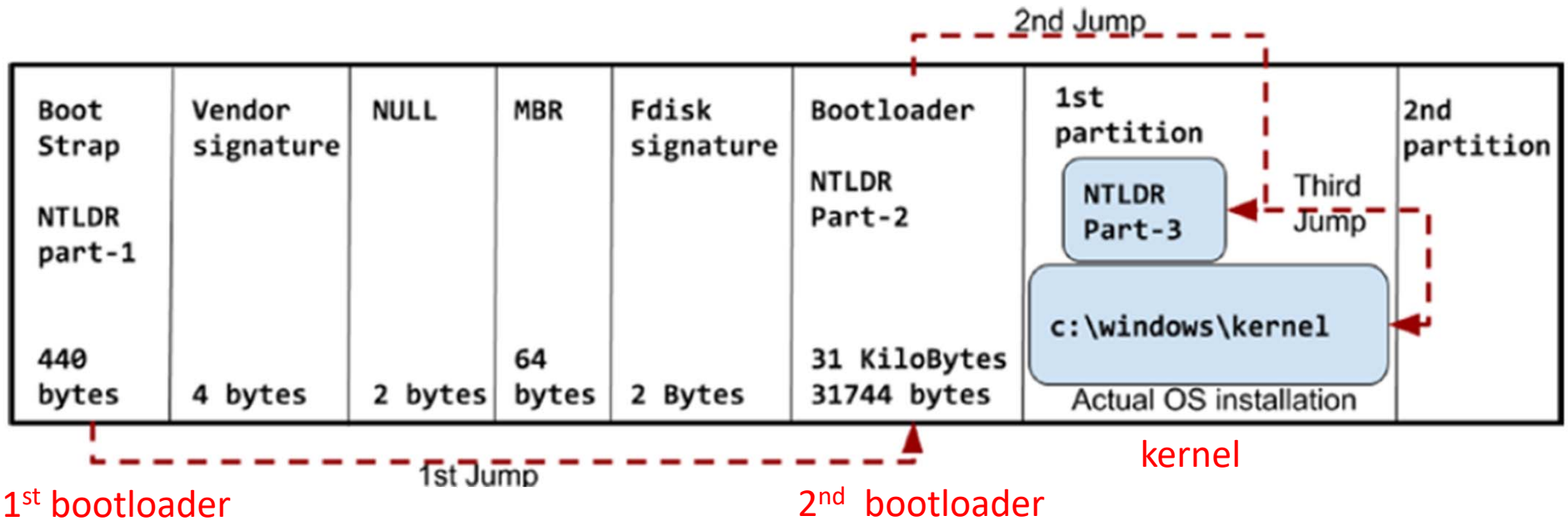
- Bootloader
  - First bootloader **系統啟動時第一個執行的程式**
    - Boot firmware (BIOS): Stored in ROM or EEPROM
    - Boot Sector
      - **載入**First bootloader (fixed location)
    - UEFI (Unified Extensible Firmware Interface): **建立**EFI file system **直接讀取**
      - efi/grub, efi/apple, efi/microsoft…
  - Second bootloader: **載入**kernel (Ex: GRUB、NTLDR、LK)

GRUB: GRand Unified Bootloader

    - **找到**kernel image
    - **取得**kernel **參數**
    - **啟動並將參數傳入**kernel

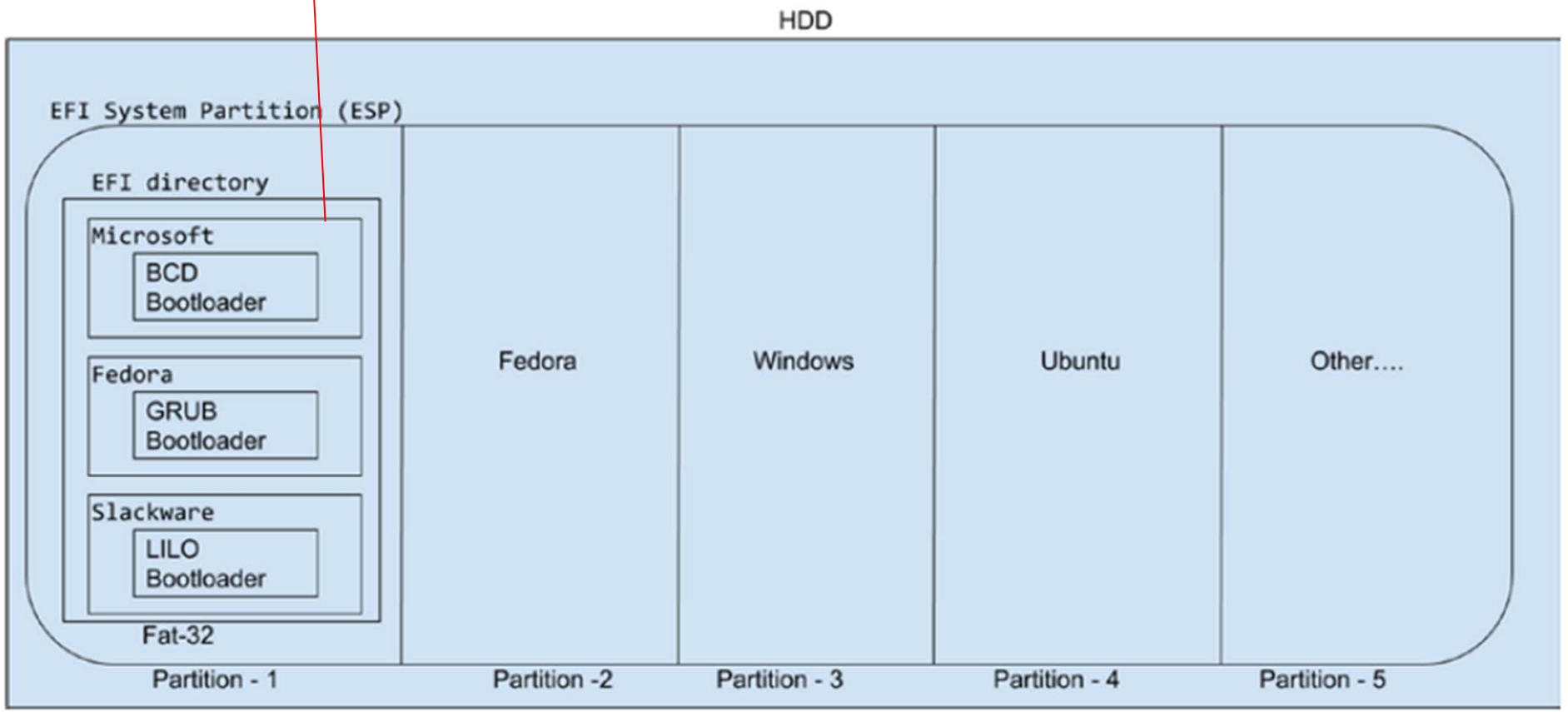
```
BOOT_IMAGE=/boot/vmlinuz-4.15.0-45-generic root=UUID=a0001b43-bfa0-4d3b-a4d5-c61
d490f7a98 ro quiet splash
```

Boot Sector (WinXP案例)



UEFI

```
EFI System Partition
├── EFI
│   ├── <OS_vendor_name>
│   │   ├── <boot_loader_files>
```



Demo

- cat /proc/cmdline

```
BOOT_IMAGE=/boot/vmlinuz-4.15.0-45-generic root=UUID=a0001b43-bfa0-4d3b-a4d5-c61
d490f7a98 ro quiet splash
```

Kernel image與參數都存在file system中，因此通常要在file systems 還沒mount之前，在最低階的層次直接讀取檔案

Vmlinuz本身是壓縮檔→誰來解壓縮?

Vmlinuz = Header + kernel setup code + vmlinux (actual kernel)

這段才是真壓縮

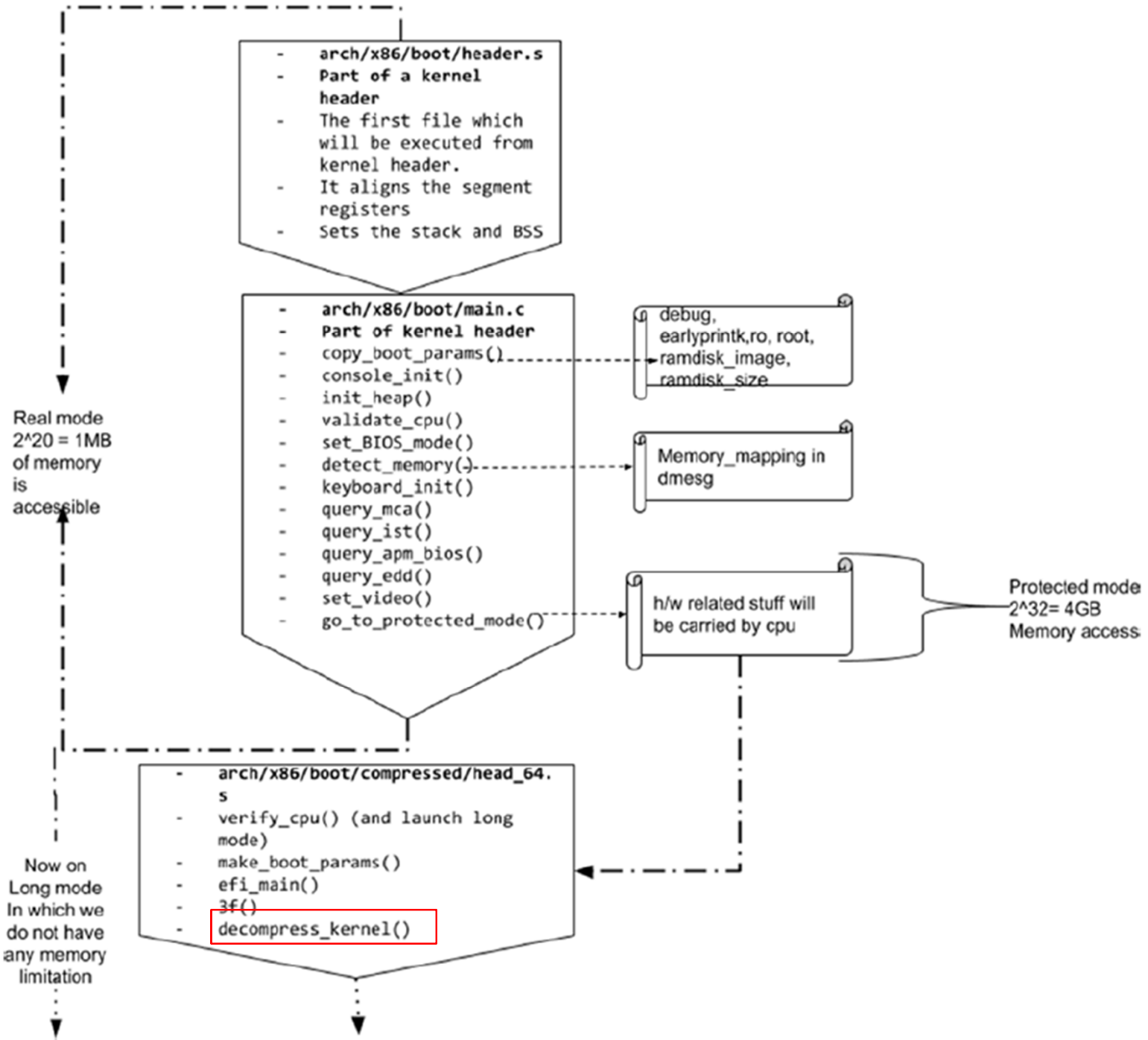
arch/x86/boot/header.S

arch/x86/boot/main.c

arch/x86/boot/compressed/head_64.S

arch/x86/boot/compressed/misc.c　　//解壓程式碼內含於此

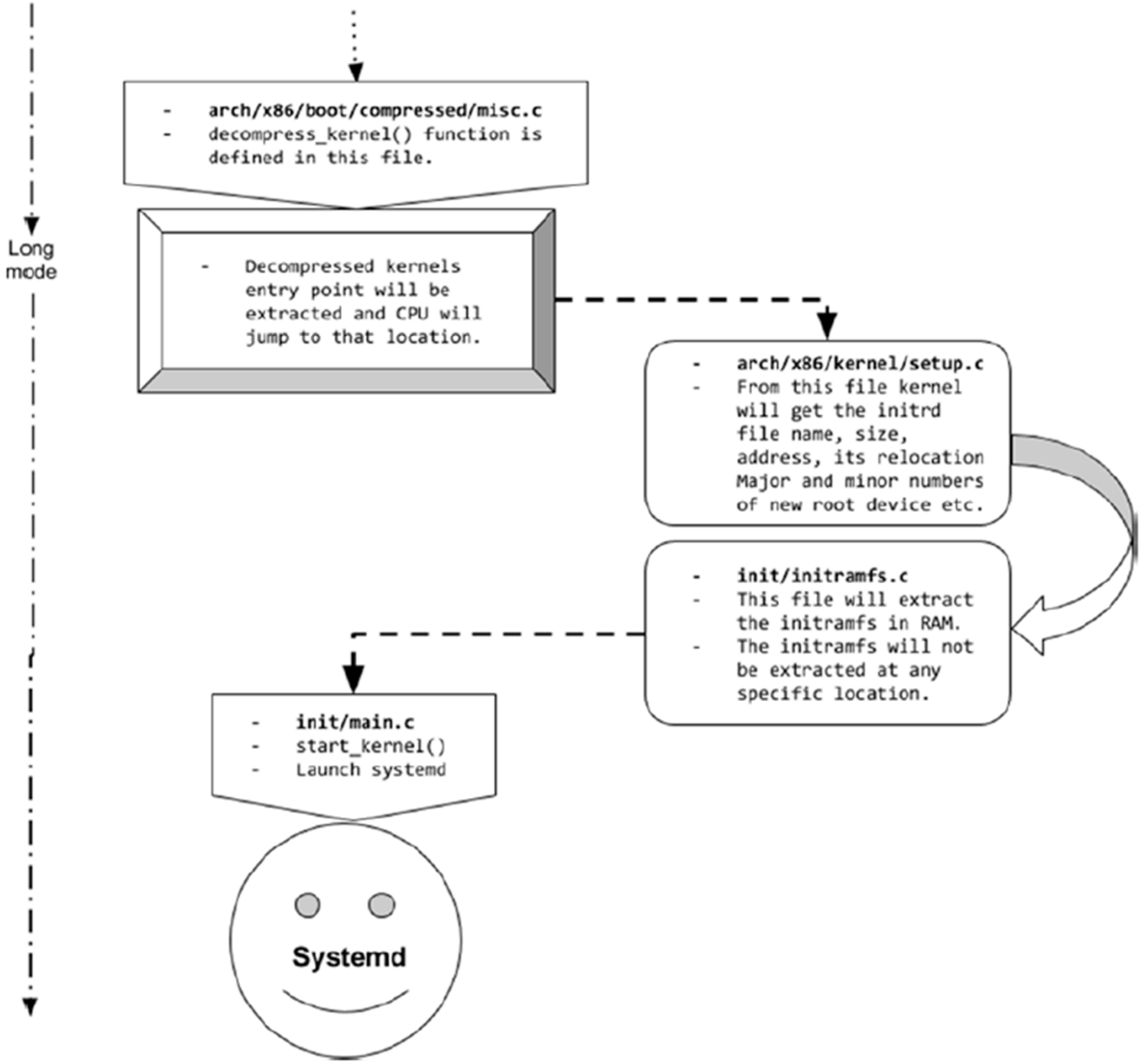
https://github.com/torvalds/linux/tree/master/arch/x86/boot



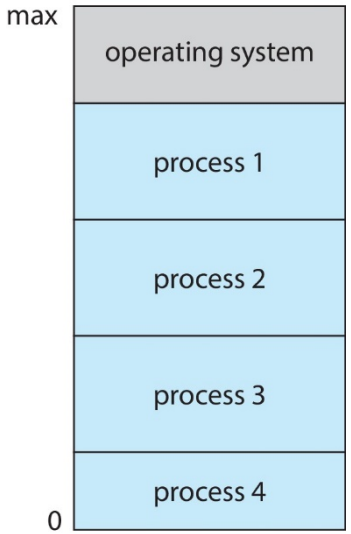
沒有空間限制，但有存取權管制

系統啟動 (2)

- Kernel
  - 硬體初始化
    - CPU registers, device controllers, memory
  - 啟動作業系統
    - Mounts the root file systems (initramfs)
    - 準備好interrupt機制
    - Executes init (systemd)
    - Starts other daemons
    - 系統啟動之後，等待interrupt，並加以處理
      - SW Interrupts透過system call來觸發


Long mode
- arch/x86/boot/compressed/misc.c
- decompress_kernel() function is defined in this file.
- Decompressed kernels entry point will be extracted and CPU will jump to that location.
- arch/x86/kernel/setup.c
- From this file kernel will get the initrd file name, size, address, its relocation Major and minor numbers of new root device etc.
- init/initramfs.c
- This file will extract the initramfs in RAM.
- The initramfs will not be extracted at any specific location.
- init/main.c
- start_kernel()
- Launch systemd
Systemd

Multiprogramming and Multitasking



- Multiprogramming
  - Batch system
  - Single user cannot keep CPU busy at all times
  - Multiprogramming organizes jobs so CPU always busy
- Multitasking
  - Timesharing system
  - CPU switches jobs so frequently that users can interact with each job while it is running
  - Key technologies
    - CPU scheduling
    - Memory management/ Virtual memory

Dual Mode

- Purpose
  - 確保AP不要亂搞系統
    - 尤其不能影響到作業系統本身的正常運作
    - CPU分不出來一道指令到底來自OS還是AP
- Use HW to enforce dual mode execution

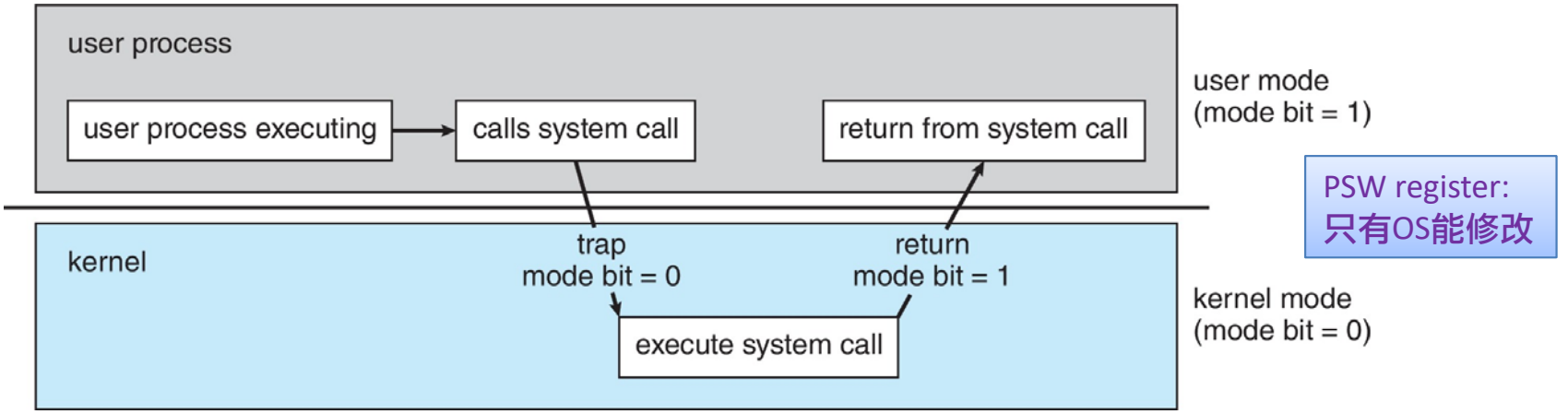
PSW register:
只有OS能修改

  - User mode: executed on behalf of the user
    - AP平常使用user mode運作
  - Kernel mode: executed on behalf of OS
    - 執行重大指令時(privileged instructions)， 切換到kernel mode

PSW=Program Status Word, 在MIPS中可以用K0, K1來做

Dual Mode

- User application平常使用user mode運作，需要執行重大指令時(privileged instructions)，需透過OS 切換到kernel mode
  - Privileged instructions executed only in kernel mode
  - Typically via system calls (to cause traps)
    - Kernel mode下，只能執行OS現成寫好的system call 內的程式來做事
    - 不透過system call，應用程式無法執行privileged instructions



執行system call →發出trap →OS改PSW進入kernel mode → 執行privilege instructions→ CPU檢查PSW，發現是kernel mode→執行

```c
#include <sys/types.h>
#include <unistd.h>

int main(void){

 while(1){}

}
```

```c
#include <sys/types.h>
#include <unistd.h>

int main(void){

 while(1){getppid();}

}
```

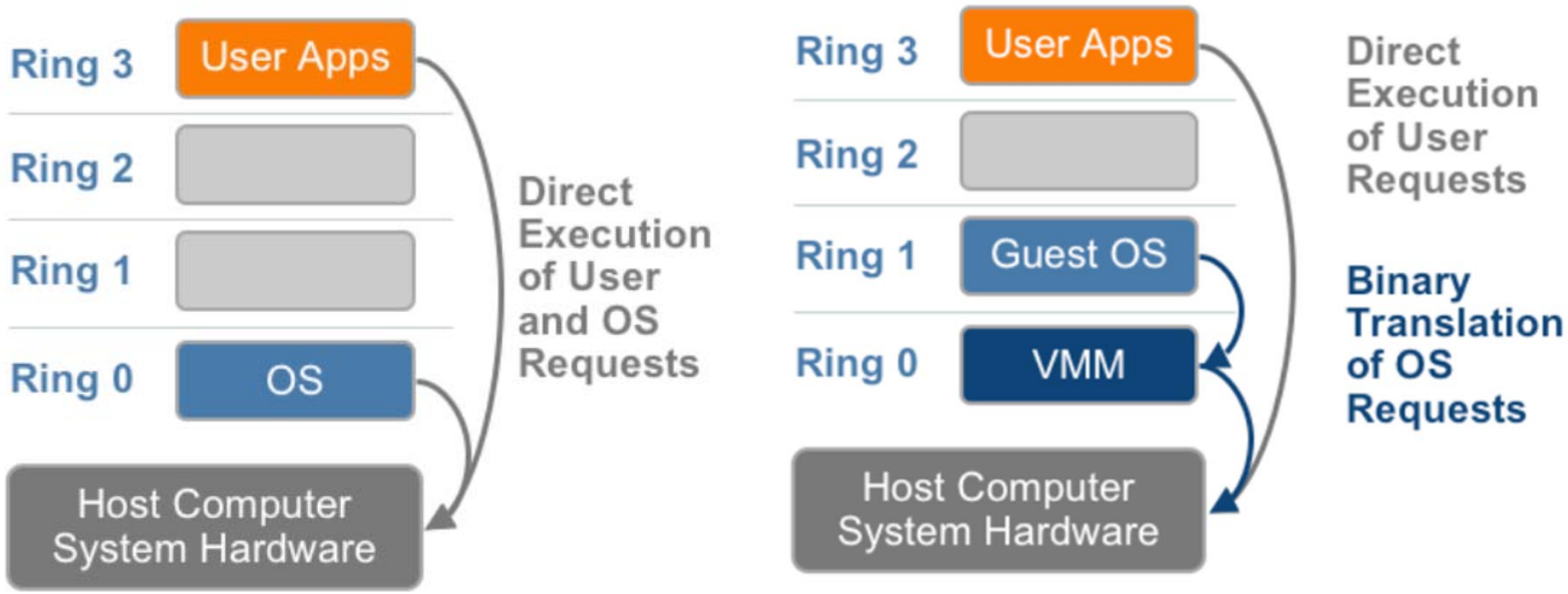
```
14時18分58秒     CPU      %user     %nice   %system   %iowait    %steal     %idle
14時18分59秒     all      69.81      0.00     30.19      0.00      0.00      0.00
14時18分59秒       0      69.81      0.00     30.19      0.00      0.00      0.00

平均時間：      CPU      %user     %nice   %system   %iowait    %steal     %idle
平均時間：      all      69.81      0.00     30.19      0.00      0.00      0.00
平均時間：        0      69.81      0.00     30.19      0.00      0.00      0.00
root@try-VirtualBox:~# sar -P ALL 1 1
Linux 4.15.0-70-generic (try-VirtualBox)        2020年08月26日  _x86_64_        (1 CPU)

14時21分06秒     CPU      %user     %nice   %system   %iowait    %steal     %idle
14時21分07秒     all      30.00      0.00     70.00      0.00      0.00      0.00
14時21分07秒       0      30.00      0.00     70.00      0.00      0.00      0.00

平均時間：      CPU      %user     %nice   %system   %iowait    %steal     %idle
平均時間：      all      30.00      0.00     70.00      0.00      0.00      0.00
平均時間：        0      30.00      0.00     70.00      0.00      0.00      0.00
```

有呼叫到system call的程式，在kernel model花費比較多的時間

sar –P ALL 1 1 (1秒鐘取樣1次)

Multimode

- Extended form the concept of dual mode
  - Intel: 4-ring model
    - 0: kernel; 3: user; other: for VM or VMM
  - ARMv8: 7-mode model



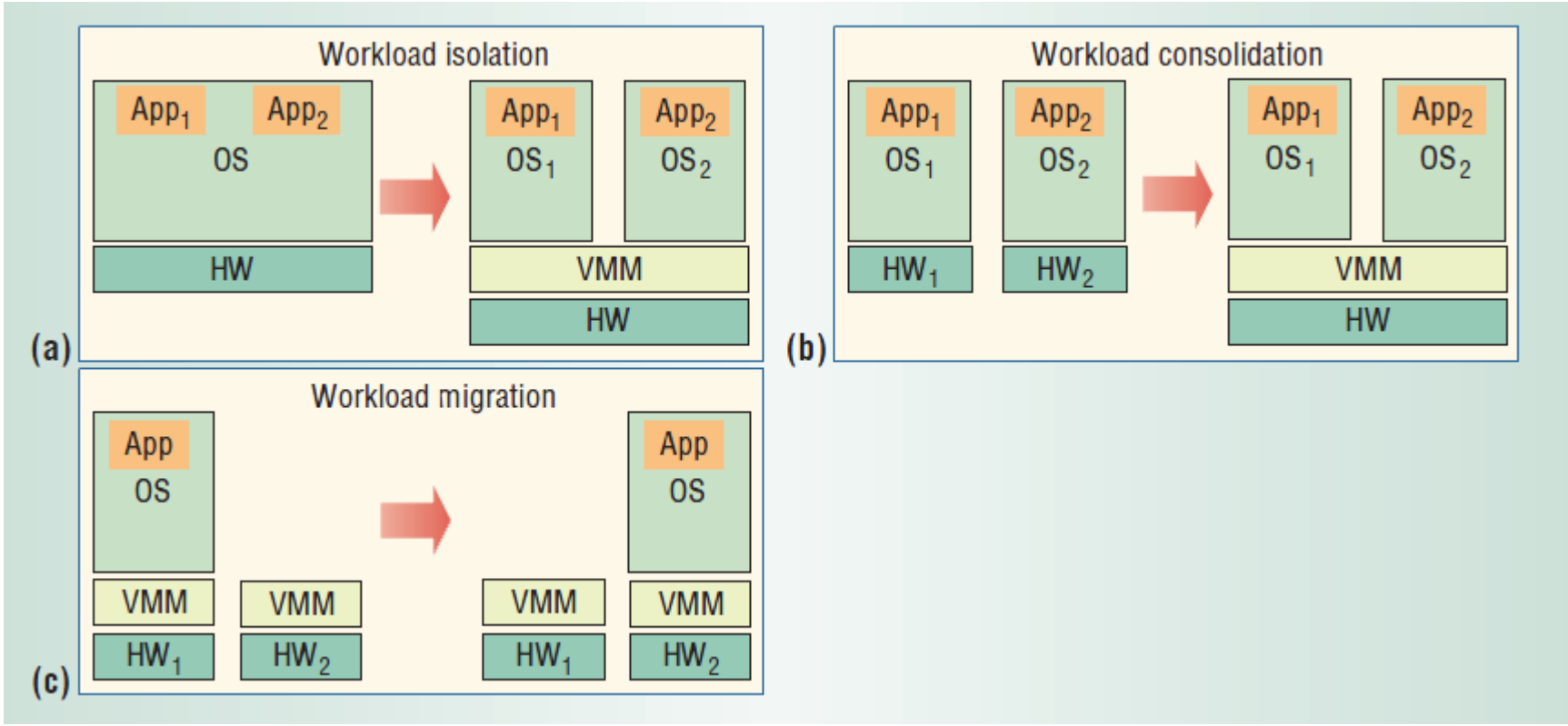
Security and Protection

- Protection
  - Any mechanism for controlling access of processes or users to resources defined by the OS
- Security
  - Defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

Virtualization Technology

- Virtualization (as code)
  - To create a software-based version of something
    - Something = OS, Database, Server, Storage, Network…
- Virtual Machine
  - A software-based implementation of some real (hardware-based) computer
- Virtual Machine Monitor (VMM, or called Hypervisor)
  - The software that creates and manages the execution of virtual machines
  - Essentially an operating system

為何要虛擬化?


Workload isolation
App1
App2
OS
HW
App1
OS1
App2
OS2
VMM
HW
(a)
Workload consolidation
App1
OS1
App2
OS2
HW1
HW2
App1
OS1
App2
OS2
VMM
HW
(b)
Workload migration
App
OS
VMM
HW1
VMM
HW2
App
OS
VMM
HW1
VMM
HW2
(c)

VM系統的分類

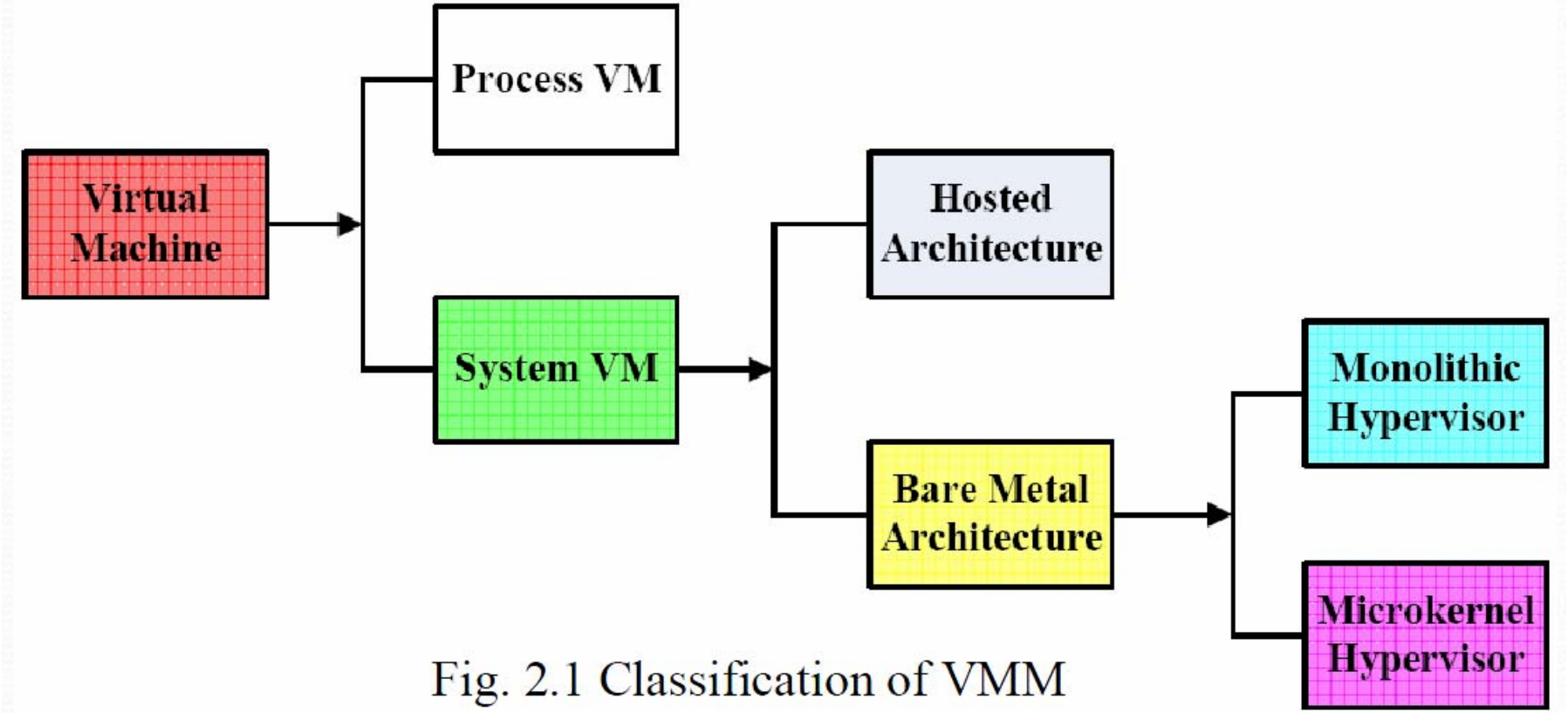
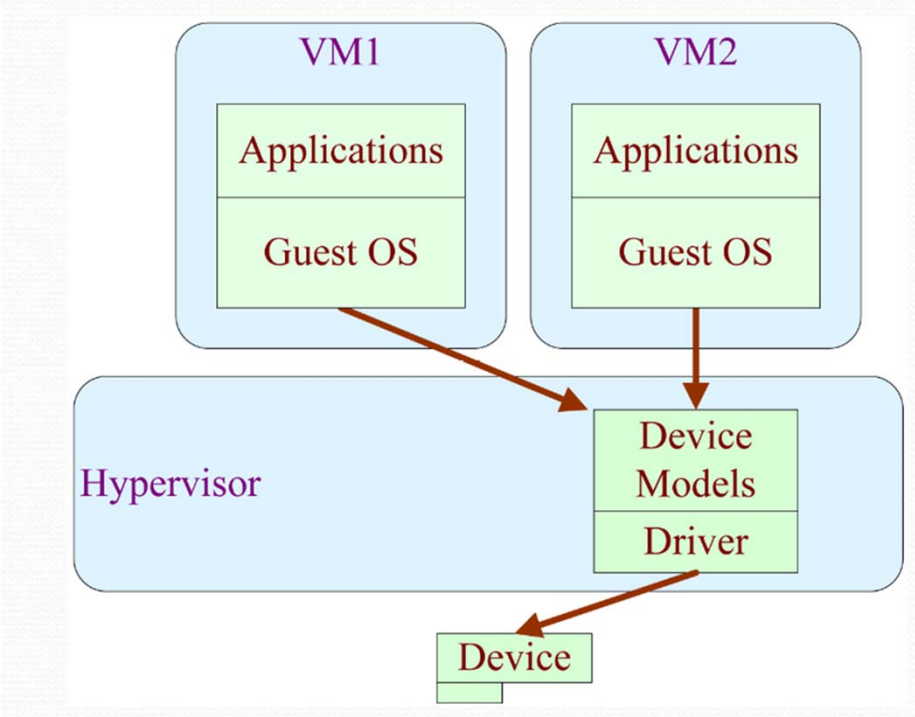


Fig. 2.1 Classification of VMM

Monolithic Hypervisor

- VMM可以看作為了虛擬化而設計出來的一個完整OS，它掌控並管理所有硬體資源
- VMM尚需負責VM的建立與管理

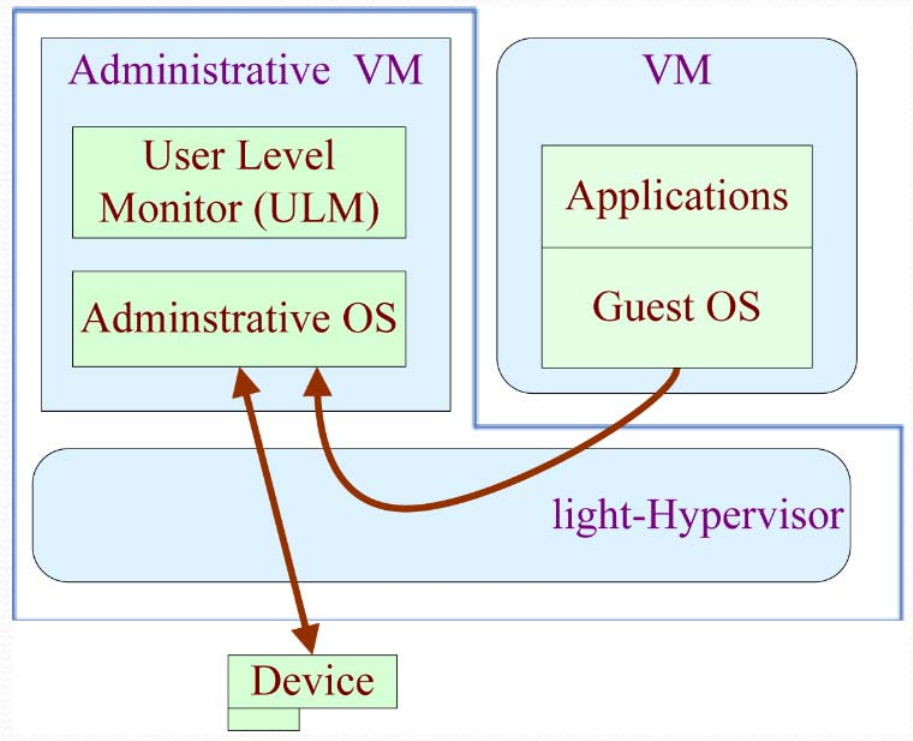


Monolithic Hypervisor

- 優點：因VMM同時具有硬體資源的管理功能和虛擬化功能，故效能較高
- 缺點：VMM開發商需提供所有IO設備驅動程式
- 產品：採用該結構的VMM有VMWare ESX Server、Wind River Hypervisor、KVM（後期）
- 對象：企業層級虛擬化、大型伺服器虛擬化

Microkernel Hypervisor

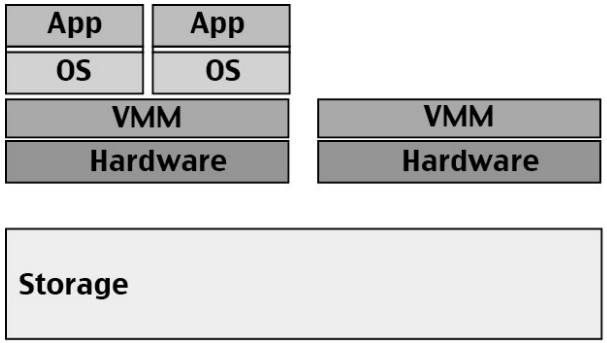
- VMM是一個輕量型Hypervisor，讓出大部分I/O設備的控制權，給Administrative VM中的 Administrative OS來控制
- VMM只負責CPU和Memory的虛擬化，I/O設備的虛擬化由VMM和Administrative OS共同完成



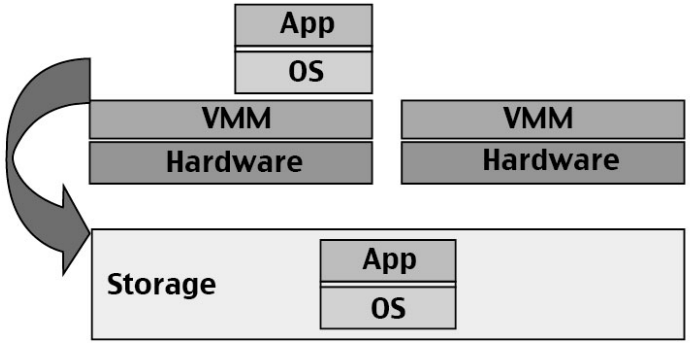
Microkernel Hypervisor

- 優點：可利用現有OS的I/O設備驅動程式，避免在VMM上開發I/O設備驅動程式
- 缺點：低效能
  - Admin OS也運行於VM上，當需要其OS提供服務時，VMM需要切換到管理OS，這裡面就產生Context Switch的時間浪費
  - 對象：研發或小型辦公室虛擬化、一般伺服器或PC虛擬化

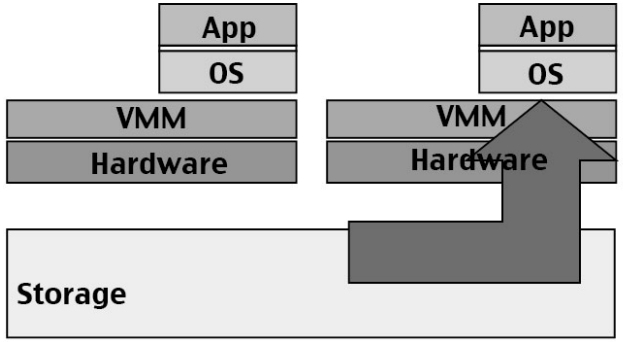
Primitive Operations in Virtual Machines



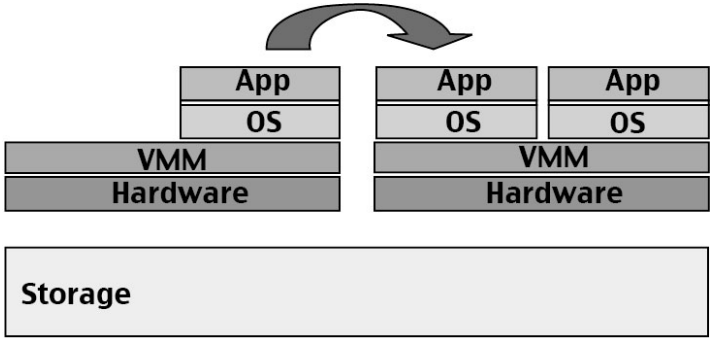
(a) Multiplexing



(b) Suspension (storage)



(c) Provision (resume)



(d) Life migration

Case: VMware

- 1998 : VMware成立; 2001 : 推出產品
  - 市佔率: 70%
- 架構
  - ESX: Bare Metal; GSX: Hosted (已終止)
  - ESX為Linux-based Admin+Hypervisor架構
    - 面臨資源耗用過多與安全性問題
  - ESXi將兩者整合
    - Monolithic架構
    - 2011後只保留ESXi產品線

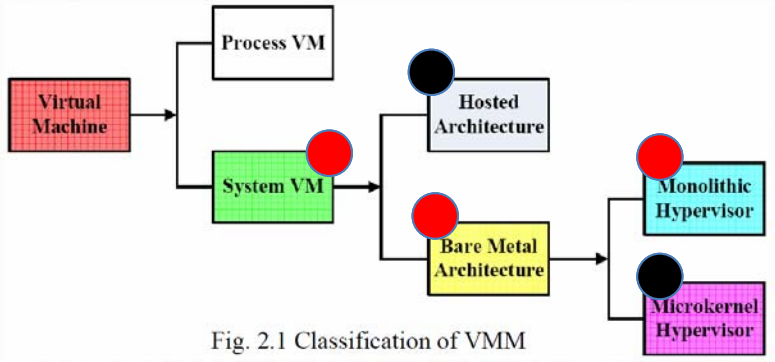
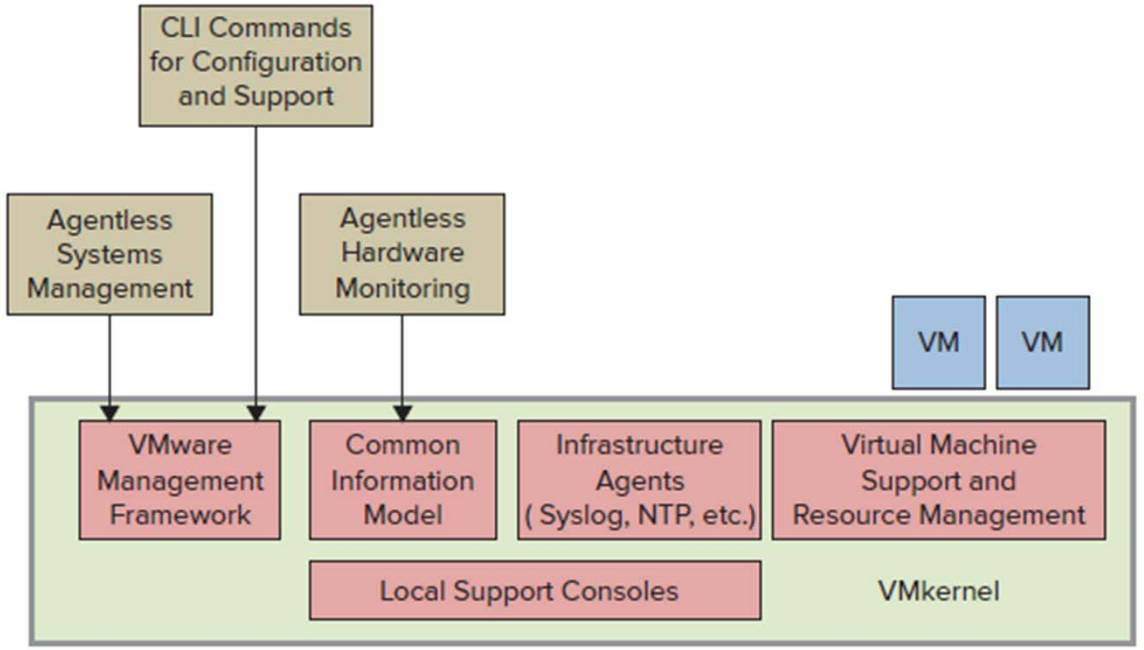


Fig. 2.1 Classification of VMM

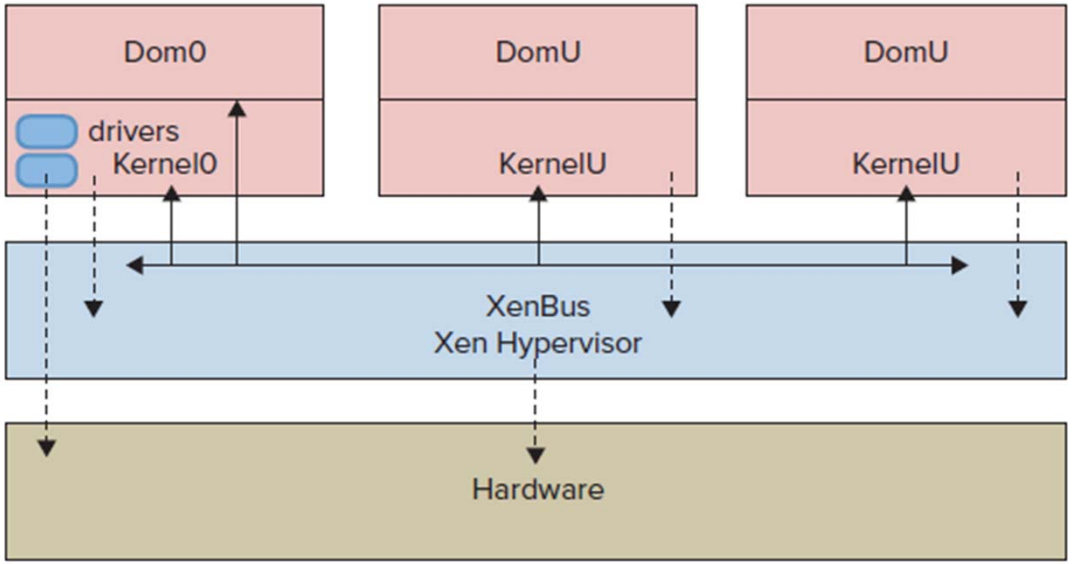
VMWare ESXi Hypervisor



重要服務全部整合進單體的大kernel中

Case: Xen

- 2002: OSS project by University of Cambridge
- 2013: Become Linux foundation project
- Microkernel hypervisor architecture
  - Dom0有效能與availability議題



虛擬化的優勢

- 充分利用現有資源的程度
- 透過縮減實體基礎架構和提升伺服器/管理員比率以降低運轉成本
- 提高硬體和應用程式的可用性，進而提高事務連貫性
- 呈現運營方面靈活性
- 提升桌面的可管理性和安全性

虛擬化的顧慮

- 以下兩點是虛擬化技術未來繼續發展過程中必需要解決的問題：
  - 硬體使用效率: 如何在多虛擬機模式下，充分發揮硬體的極緻能力
  - 安全及可靠性: 安全性是最重要的，然而可靠性也是同等重要

Review (1)

- What is a computer system?
- What is an operating system?
- Why we need interrupt and how it works?
- Why we need DMA and how it works?
- What are the benefits of and differences among multiprocessor, muti-core, NUMA, and clustered architectures?

Review (2)

- How an OS starts?
- The differences of application programs and system daemons
- Why we need system calls?
- Why we need Dual-mode/ multi-mode
- The benefits of virtualization technologies

Q & A