

Introduction to Parallel Computing (I)

Cheng-Hung Lin

課程大綱

- ▶ 課程名稱: 平行計算
- ▶ 英文名稱: Parallel Computing
- ▶ 授課教師: 林政宏 brucelin@ntnu.edu.tw
- ▶ 課程大綱:
 - ▶ 本課程將介紹平行計算與程式設計, 課程內涵包括平行計算的背景、平行計算的軟體架構、平行程式設計。其中平行計算的軟體架構包含訊息傳遞架構(message passing)、共享記憶體架構(shared-memory system)、向量/單一指令多重資料架構(vector/SIMD)、與網路溝通架構(communication networks)。而平行程式設計部分包含OpenMP, Pthreads, CUDA, and MPI.

參考進度

1. Introduction /Theoretical background

Amdahl's Law, Gustafson's Law, Limits of parallel computing, load balancing

2. Parallel computing systems

Hardware classification(SISD, SIMD, MISD, MIMD), Hardware for parallel computing, memory architecture (shared and distributed memory), hybrid system, multicore system, accelerated systems (GPGPU and MIC)

3. Parallel programming models

Data parallelism, task parallelism, SPMD(Single Program Multiple Data)

4. Introduction of shared-memory programming with OpenMP

5. OpenMP scheduling, caches, cache coherence, and false sharing

6. Case study using OpenMP: Sorting, Odd-even transposition sort

7. Introduction of GPGPU programming using CUDA

8. Thread cooperation, shared memory and synchronization

9. Atomic operation, texture and constant memory
10. Performance evaluation, case study using CUDA
11. Midterm Presentation
12. Introduction of shared-memory programming with Pthreads
13. Case study using Pthread: Matrix-vector multiplication
14. Introduction of distributed memory programming using MPI
15. Collective vs. point-to-point Communication, Tree-structure communication, Broadcast, data distributions
16. Case study using MPI, Performance evaluation of MPI
17. Demo of Final projects

評分方式

- ▶ 作業 30%
- ▶ 期中報告 30%
- ▶ 期末專題 40%

平行計算(Parallel Computing)

- ▶ 一般是指許多指令得以同時進行的計算模式。
- ▶ 同時使用多種計算資源解決計算問題的過程。
- ▶ 在同時進行的前提下，可以將計算的過程分解成小部份，之後以並行方式來加以解決。
- ▶ 電腦軟體可以被分成數個運算步驟來執行。為了解決某個特定問題，軟體採用某個演算法，以一連串指令執行來完成。
- ▶ 為執行平行計算，計算資源應包括一臺配有多處理機(並行處理)的計算機、或一個與網路相連的計算機系統，或者兩者結合使用。平行計算的主要目的是**快速**解決大型且複雜的計算問題。

為什麼需要平行計算?

- ▶ 節省時間
 - ▶ 使用更多的資源來縮短執行時間，可以節省花費



- ▶ 縮短執行時間代表我們有更多的機會來優化工作

	DUAL XEON CPU server	DGX-1 GPU server (8 GPUs)
FLOPS	3TF	170TF
Node Mem BW	76GB/s	768GB/s
Alexnet Train Time	150 Hr	2Hr
Train in 2Hr	>250Nodes	1Node

為什麼需要平行計算?

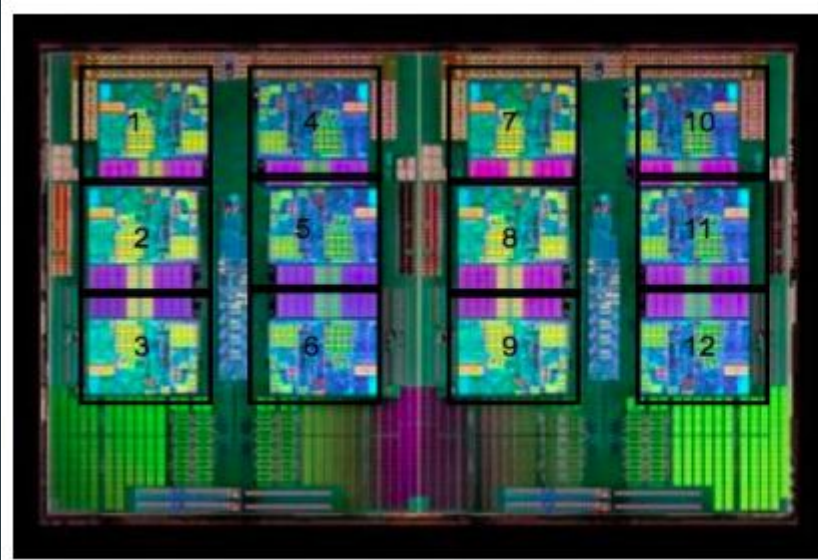
- ▶ 解決大型問題
 - ▶ 解決難以在單一電腦上解決的大問題
- ▶ 科學計算(Scientific computing):
 - ▶ Trillion particles
 - ▶ Tens and hundreds of parameters
 - ▶ TBs of data to be processed/analyzed
 - ▶ Several hours of execution using millions of cores (PetaFLOPS)



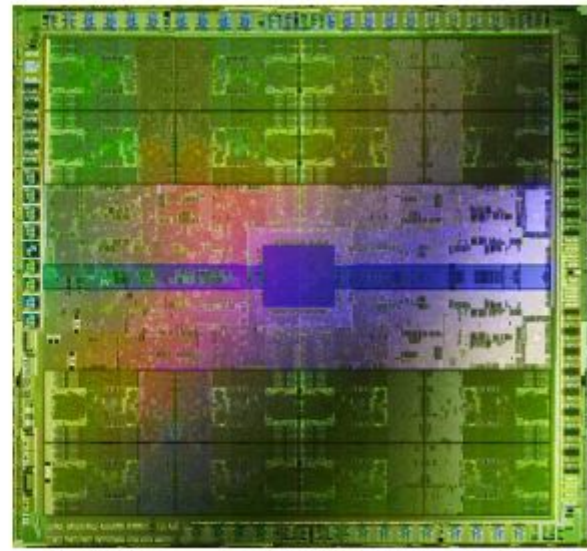
The world has been driven by science research!

為什麼需要平行計算?

- ▶ 平行計算可以有效運用多核心的硬體
 - ▶ Multicore CPU and GPU



12 Cores IBM Blade Multi-core CPU



512 Cores NVIDIA Fermi GPU

平行計算(Parallel Computing)

- ▶ 平行計算可以劃分成時間並行(temporal parallelism)和空間並行(spatial parallelism)。
 - ▶ 時間並行即流水線(pipelining)技術，將任務分解成子任務的級聯(cascade)，可以實現在單一處理器。
 - ▶ 空間並行性是指多個處理單元同時執行任務。。



時間並行



空間並行

時間並行的例子

- ▶ 比如說工廠生產食品的時候步驟分為：
 1. 清洗:將食品沖洗乾淨。
 2. 消毒:將食品進行消毒處理。
 3. 切割:將食品切成小塊。
 4. 包裝:將食品裝入包裝袋。
- ▶ 如果不採用流水線，一個食品完成上述四個步驟後，下一個食品才進行處理，耗時且影響效率。
- ▶ 採用流水線技術，就可以同時處理四個食品。
- ▶ 這就是並行算法中的時間並行，在同一時間啟動兩個或兩個以上的操作，大大提高計算性能。
- ▶ 可以實現在單一處理器。

Classic RISC Pipeline

- ▶ Instruction fetch
- ▶ Instruction decode and register fetch
- ▶ Execute
- ▶ Memory access
- ▶ Register write back

Instr. No.	Pipeline Stage						
	IF	ID	EX	MEM	WB		
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

空間並行的例子

- ▶ 小李準備在植樹節種三棵樹，如果小李1個人需要6個小時才能完成任務，植樹節當天他叫來了好朋友小紅、小王，三個人同時開始挖坑植樹，2個小時後每個人都完成了一顆植樹任務。
- ▶ 這就是並行算法中的空間並行，將一個大任務分割成多個相同的子任務，來加快問題解決速度。
- ▶ 必須具備多個處理器。

平行計算(Parallel Computing)

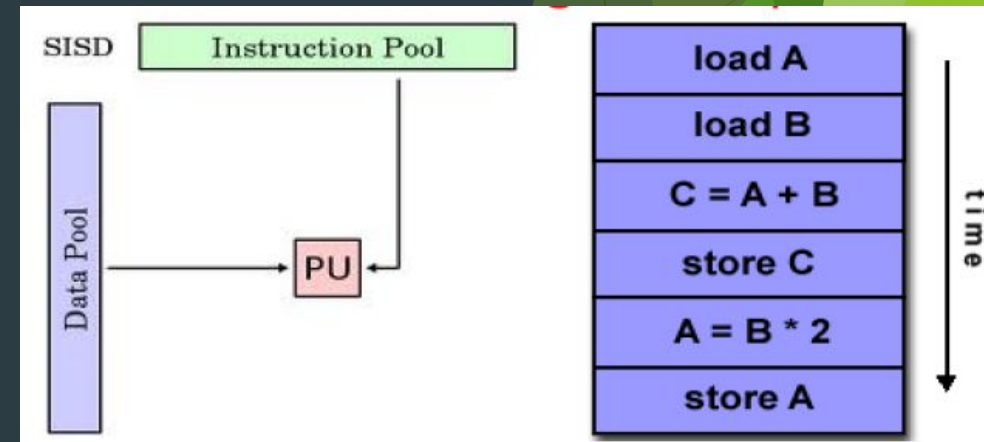
- ▶ 平行計算又可分為資料並行(data parallelism)和任務並行(task parallelism)。
- ▶ 資料並行把大的任務化解成若干個相同的子任務，處理起來比任務並行簡單。
- ▶ 空間上的並行導致兩類並行機的產生。
 - ▶ Michael Flynn (費林分類法)，高效能計算機的分類方式。
 - ▶ 單指令流多資料流(SIMD)和多指令流多資料流(MIMD)

Parallel Computer Classification

- ▶ 費林分類法 (Flynn's Taxonomy)
- ▶ 是一種高效能計算機的分類方式。1972年費林 (Michael J. Flynn) 根據資訊流 (information stream) 可分成指令 (Instruction) 和資料 (Data) 兩種。據此又可分成四種計算機類型：
 - ▶ 單一指令流單一資料流計算機 (SISD)
 - ▶ 單一指令流多資料流計算機 (SIMD)
 - ▶ 多指令流單一資料流計算機 (MISD)
 - ▶ 多指令流多資料流計算機 (MIMD)

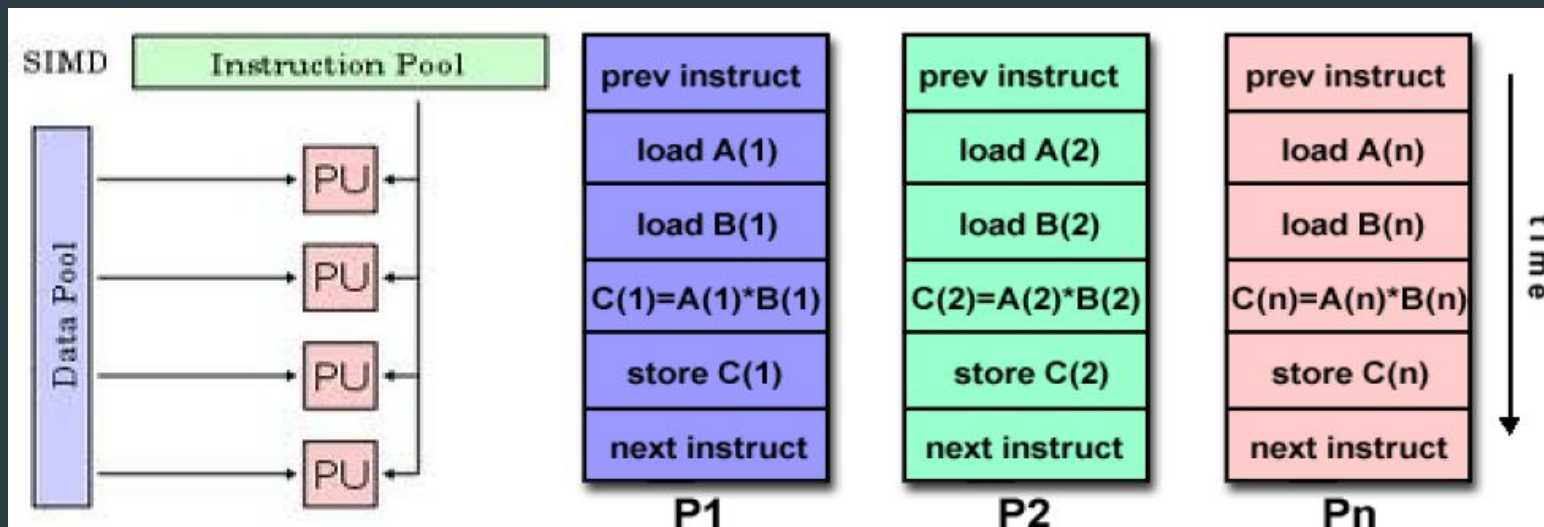
Single Instruction, Single Data (SISD)

- A serial (non-parallel) computer
- Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle
- Single Data: Only one data stream is being used as input during any one clock cycle
- Example: Old mainframes, single-core processor



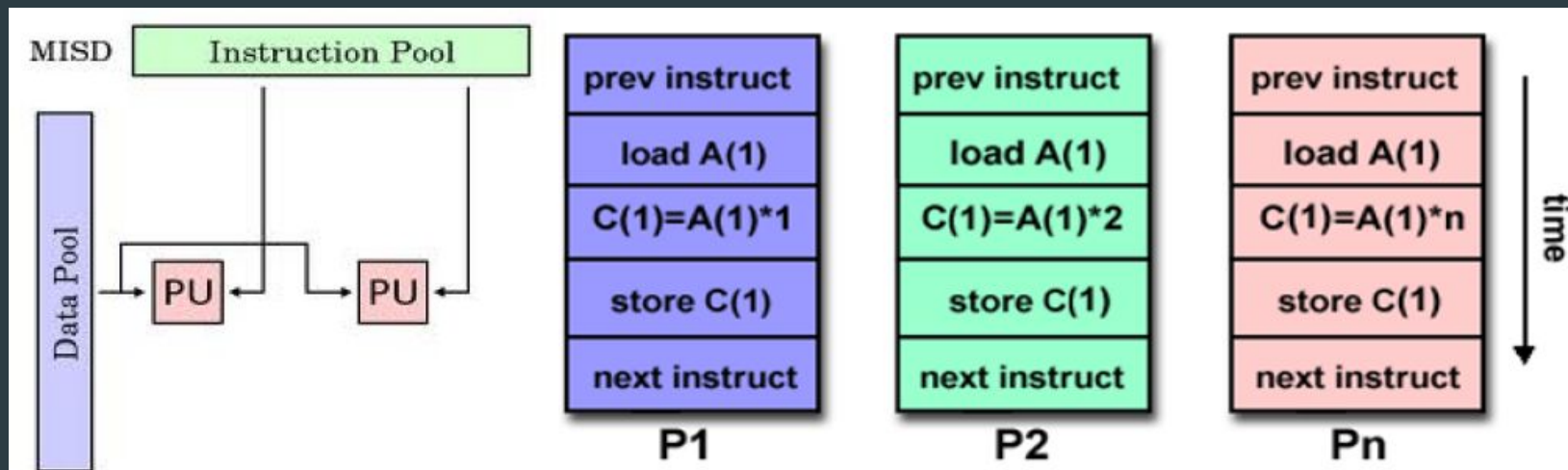
Single Instruction, Multiple Data (SIMD):

- ▶ Single Instruction: All processing units execute the same instruction at any given clock cycle
- ▶ Multiple Data: Each processing unit can operate on a different data element
- ▶ Example: GPU, vector processor (X86 AVX instruction)



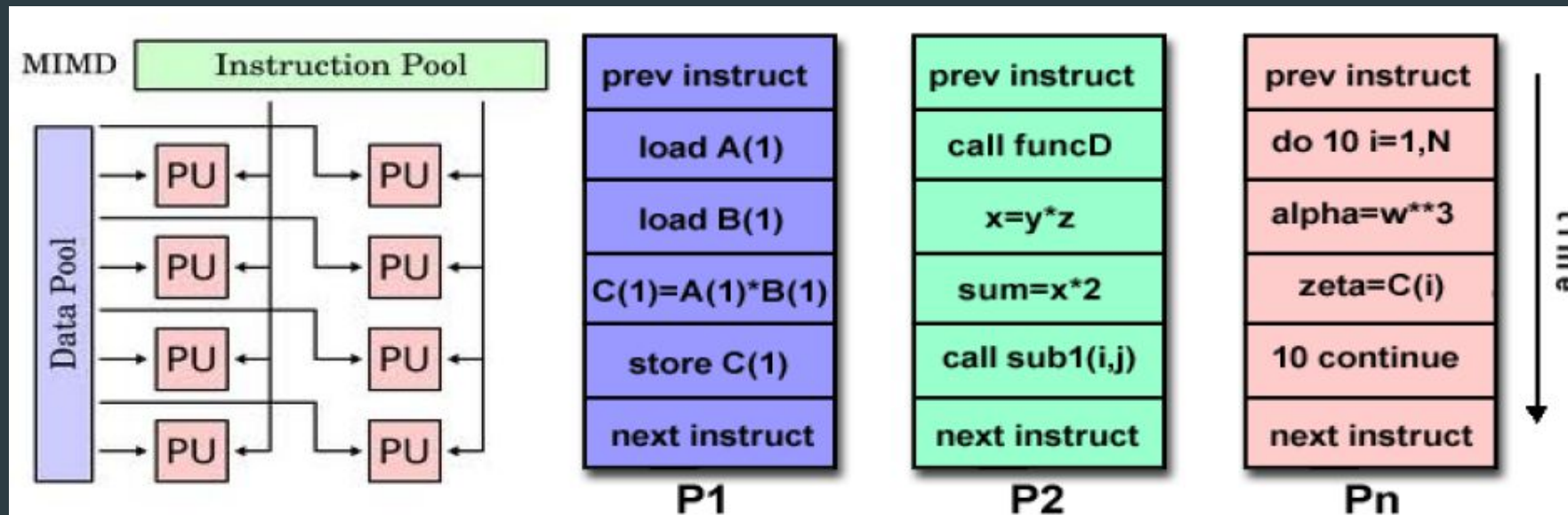
Multiple Instruction, Single Data (MISD):

- ▶ Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
- ▶ Single Data: A single data stream is fed into multiple processing units.
- ▶ Example: Only experiment by CMU in 1971; Could be used for fault tolerance



Multiple Instruction, Multiple Data (MIMD):

- ▶ Multiple Instruction: Every processor may be executing a different instruction stream
- ▶ Multiple Data: Every processor may be working with a different data stream
- ▶ Example: Most modern computers, such as multi-core CPU



平行計算的特徵

- ▶ 將工作分割成離散部分, 有助於同時解決
- ▶ 即時執行多個程式指令
- ▶ 多計算資源下解決問題的耗時要少於單個計算資源下的耗時

平行計算的性能度量

- ▶ 基本指標
 - ▶ 執行時間
 - ▶ 工作負載
 - ▶ 儲存性能
- ▶ 加速比評測
 - ▶ Amdahl's law
 - ▶ Gustofson's law
 - ▶ Sun-Ni's law

Amdahl's Law

➤ 加速倍率 $speedup = \frac{1}{(1-p) + p/s}$

➤ p : 可以被平行化的比例

➤ s : 加速的倍率

➤ 假設有一隻程式，可以被平行的部分占80%

➤ 4個核心可以平行處理

➤ $speedup = \frac{1}{(1-0.8) + 0.8/4} = 2.5$ 倍

➤ 如果有無限多個核心

➤ $speedup = \frac{1}{(1-0.8) + 0.8/\infty} = 5$ 倍，最多可以提升五倍

➤ Speedup的最大的倍率被串列的比例所決定，再多的核心數都無濟於事!!!!

1 processor

100

100

100

100

100

4 processors

100

100

100

100

100

$$speedup = \frac{500}{200} = 2.5$$

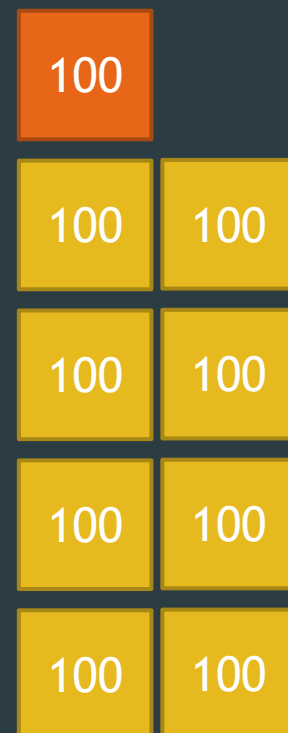
Gustafson's Law

- ▶ 直至1988年, Gustafson 對於 Amdahl定律提出不同的看法
- ▶ 如果有2個核心, 我們把可平行計算的部分工作量增加一倍, 可以完成900個單位工作, 效能提升80%($400/500=0.8$)
- ▶ 如果有4個核心, 我們把可平行計算的部分工作量增加4倍, 可以完成1700個單位工作, 效能提升2.4倍($1200/500$)

1 processor



2 processors



4 processors



Sun-Ni's Law

- ▶ 記憶體限制加速模型(Memory-bounded speedup model)隨著計算能力的增加, 加速的大小受到系統記憶體容量的限制。
- ▶ 隨著 CPU 速度和記憶體存取延遲之間的差距越來越大, 應用程序的執行時間通常取決於系統的記憶體的速度。
- ▶ Sun-Ni's定律指出運算速度的大小應該是可擴展的, 但受系統記憶體容量的限制。
- ▶ 有時, 記憶體的頻寬可能比記憶體容量更重要。

Difference of Amdahl's Law, Gustafson's Law, and Sun-Ni's Law

- ▶ Amdahl's定律側重於給定**固定大小問題**的時間縮減。
 - ▶ Amdahl's law定律指出, 隨著系統資源的增加, 問題(算法)的**循序部分(sequential part)**限制了可以實現的總加速。
- ▶ Gustafson's 定律表明構建大規模並行系統是有益的, 因為如果擴大問題規模並以保持固定的執行時間, 加速比可以隨系統規模線性增長。
 - ▶ 然而, 由於**記憶體延遲**通常成為應用程序執行時間的主要因素, 應用程序可能無法擴展以滿足時間限制。
- ▶ Sun-Ni's定律不是通過時間來限制問題的大小, 而是通過系統的記憶體容量來限制問題, 或者換句話說, 基於記憶體的界限。

分散式計算 (Distributed Computing)

- ▶ 透過多個計算機系統 (通常稱為節點) 的協同工作, 以完成一個共同的任務或解決一個共同的計算問題。這些節點可能分布在世界各地, 通過網絡互相連接。分散式計算的目的是利用多個計算資源來提高計算速度, 增加數據存儲能力, 提高系統的可靠性和容錯能力。
 - ▶ 并行處理: 分散式計算允許多個進程同時運行在不同的節點上, 從而加快計算速度。
 - ▶ 容錯性: 由於系統分布在多個節點上, 單個或多個節點的失效不會導致整個系統失效, 從而增加了系統的可靠性。
 - ▶ 擴展性: 隨著計算需求的增加, 可以通過增加更多的節點來擴展系統的計算能力。
 - ▶ 資源共享: 分散式計算使得不同地理位置的資源能夠被共享和利用。

平行計算與分散式計算的區別

- ▶ 平行計算和分散式計算都是用於提高計算效率和處理大規模計算問題的技術，但它們在設計、實現以及應用場景上有著本質的區別。
- ▶ **1. 計算環境**
 - ▶ **平行計算** 通常發生在單個計算機系統中，該系統擁有多個處理器(或核心)。這些處理器共享系統的記憶體和資源。
 - ▶ **分散式計算** 涉及多個獨立的計算機(或節點)，這些計算機通過網絡連接。每個節點都有自己的處理器和本地記憶體，並且運行可能不相同的操作系統。
- ▶ **2. 記憶體共享**
 - ▶ **平行計算** 中的處理器通常訪問共享的記憶體空間，這使得數據交換和同步相對簡單。
 - ▶ **分散式計算** 中的節點擁有獨立的記憶體，因此數據共享和通信必須通過網絡進行，這可能引入額外的延遲和複雜性。
- ▶ **3. 計算目標**
 - ▶ **平行計算** 旨在通過多個處理器協同工作來加速單一計算任務的執行速度。
 - ▶ **分散式計算** 不僅旨在提高計算速度，還旨在處理大規模的數據集，提高數據存儲能力，增強系統的可靠性和可用性。

平行計算與分散式計算的區別

▶ 4. 計算模型

- ▶ **平行計算** 模型著重於數據的并行處理和任務的分解, 以實現在多個處理器上同時計算。
- ▶ **分散式計算** 模型更加關注於節點之間的通信和協作, 以及如何有效地分配和管理跨多個計算資源的任務。

▶ 5. 應用領域

- ▶ **平行計算** 常用於科學計算、圖像處理、大規模模擬等需要大量計算資源的領域。
- ▶ **分散式計算** 廣泛應用於雲計算、大數據分析、分散式數據庫、網絡服務等需要高度可擴展性和可靠性的場景。

叢集計算(Cluster Computing)

- ▶ 叢集計算(Cluster Computing)是一種通過將多台獨立的計算機(稱為節點)連接在一起,形成一個統一的系統來進行計算的技術。這些節點通過快速的局域網(LAN)緊密連接,共同工作以完成計算任務。叢集系統可以看作是一種特殊的分散式計算系統,
- ▶ **高性能**:通過結合多台計算機的計算能力,叢集可以處理比單個計算機更複雜、更大規模的計算任務。
- ▶ **可擴展性**:當需要更多的計算能力時,可以通過增加更多的節點來擴展叢集的規模。
- ▶ **高可用性**:叢集計算通常具有容錯功能,如果一個節點失效,其他節點可以接管其工作,從而提高系統的整體可靠性。
- ▶ **成本效益**:叢集計算可以使用標準的、低成本的硬件和開源軟件,相比於傳統的大型計算機,它們提供了一種經濟高效的計算方式。

網格計算(Grid Computing)

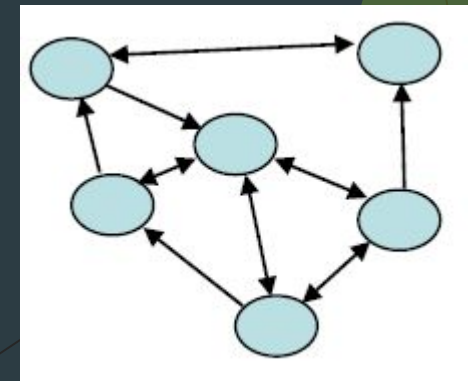
- ▶ 網格計算是分散式計算的一種，也是一種與**叢集**計算非常相關的技術。
- ▶ **異構性**：網格由多種不同類型的計算資源組成，包括不同操作系統和硬件架構的計算機。
- ▶ **地理分散性**：網格中的計算資源分佈在不同的地理位置，通過互聯網或其他通信網絡連接。
- ▶ **動態性**：網格中的資源可以根據需求加入或離開網格，資源的可用性和性能可能會變化。
- ▶ **資源共享**：網格計算使不同組織和用戶能夠共享計算能力、數據存儲和應用程序。

雲端計算(Cloud Computing)

- ▶ 雲端計算是最新開始的新概念，它不只是負責計算，還有營運服務等概念了。
- ▶ 它是分散式計算、平行計算和網格計算的發展，或者說是這些概念的商業實現。
- ▶ 美國國家標準和技術研究院的雲端運算定義中明確了三種服務模式：^[12]
- ▶ 軟體即服務(Software as a Service, SaaS): 消費者使用應用程式，但並不掌控作業系統、硬體或運作的網路基礎架構。是一種服務觀念的基礎，軟體服務供應商，以租賃的概念提供客戶服務，而非購買，比較常見的模式是提供一組帳號密碼。例如：Adobe Creative Cloud，Microsoft CRM與Salesforce.com。
- ▶ 平台即服務(Platform as a Service, PaaS): 消費者使用主機操作應用程式。消費者掌控運作應用程式的環境(也擁有主機部分掌控權)，但並不掌控作業系統、硬體或運作的網路基礎架構。平台通常是應用程式基礎架構。例如：Google App Engine。
- ▶ 基礎設施即服務(Infrastructure as a Service, IaaS): 消費者使用「基礎運算資源」，如處理能力、儲存空間、網路元件或中介軟體。消費者能掌控作業系統、儲存空間、已部署的應用程式及網路元件(如防火牆、負載平衡器等)，但並不掌控雲端基礎架構。例如：Amazon AWS、Rackspace。

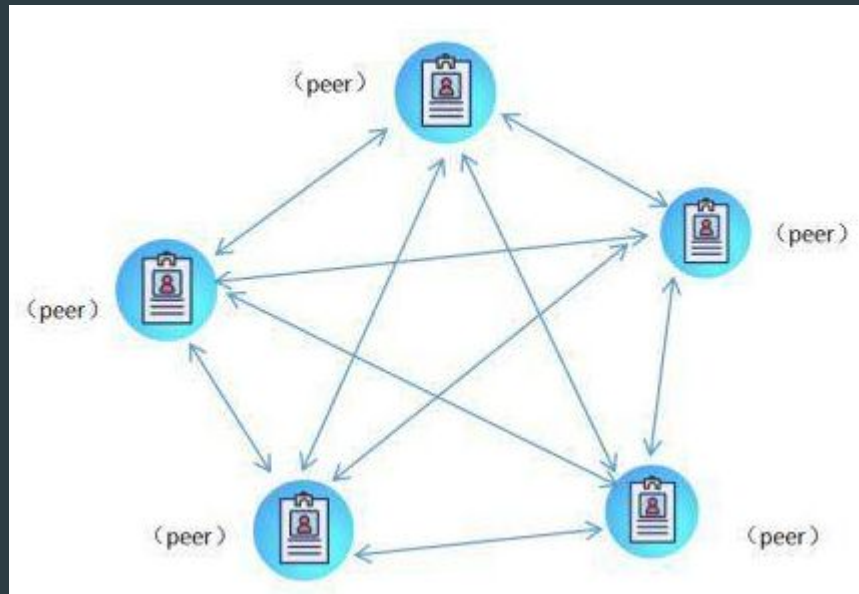
對等式運算 (peer to peer)

- ▶ **P2P (Peer-to-Peer) Network** 點對點網路又稱同儕網路、對等網路，概念源自於音樂分享服務Napster，被Intel喻為第三代網路革命的「點對點分散式網路架構」。
- ▶ 網路中每個節點既是 Server Side 服務端，亦是 Client Side 用戶端；節點可以選擇隨時加入，隨時退出；
- ▶ P2P系統讓個人電腦和工作站可透過internet以及其他的網路達到資料的運用以及資源的計算；而不是單獨管理的server，將資源集中在一起服務。因此，services不可依賴在信任的單一主機，可藉由多台具有相同資源主機尋求服務以解決此問題。
- ▶ P2P系統主要特徵
 - ▶ 每個用戶貢獻資源給其他用戶
 - ▶ 每個節點具有相同的功能能力和責任
 - ▶ 不依賴任何中央管理系統



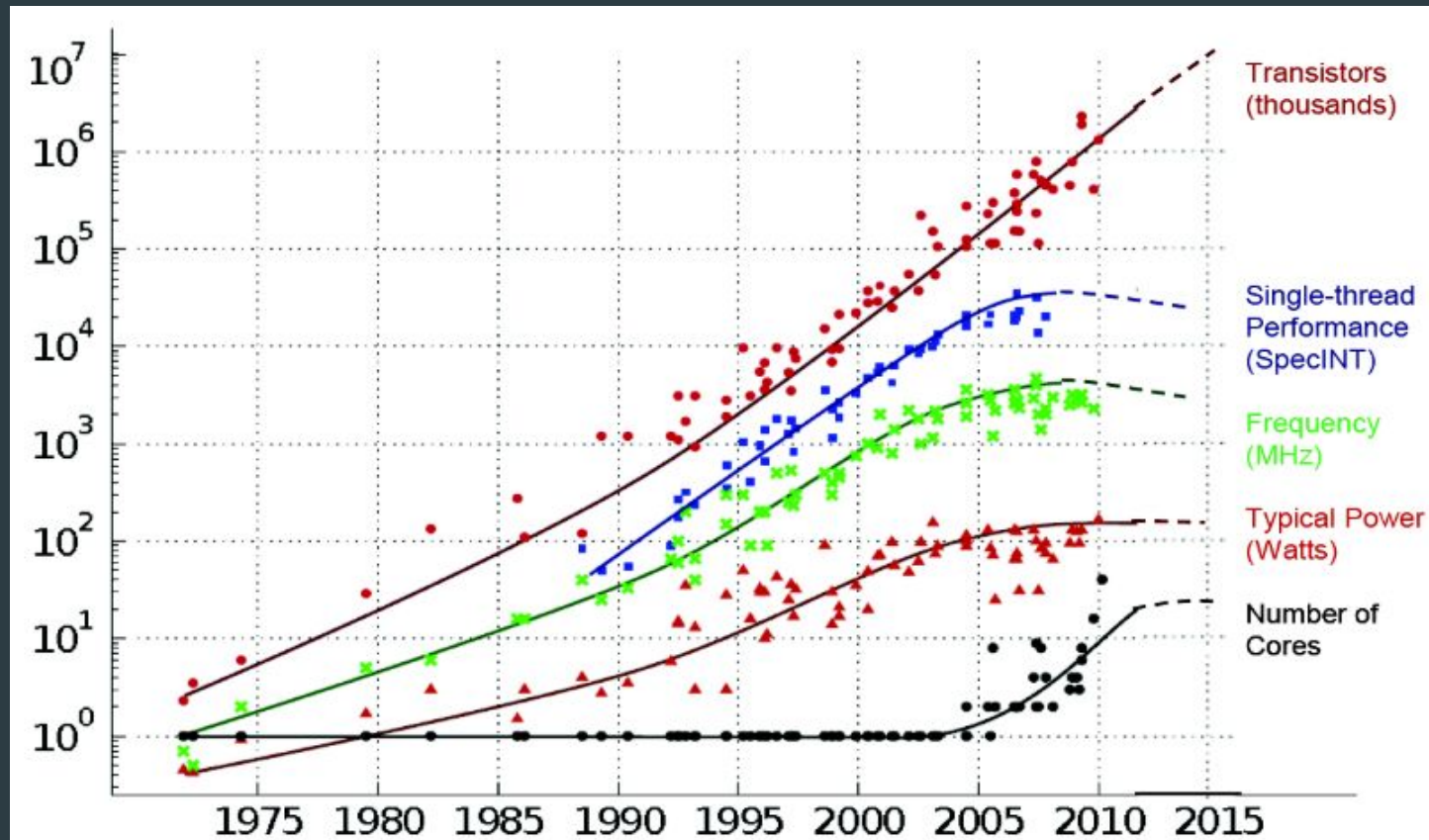
區塊鏈 Blockchain

- ▶ 比特幣白皮書 (Bitcoin: A peer-to-peer Electronic Cash System) 中，使用 Peer-to-Peer (P2P) 描述區域鏈技術及加密貨幣
- ▶ 區塊鏈是 P2P 構成的去中心化系統，以保持各個節點間的數據記錄一致來建立分佈式“賬本”。
- ▶ P2P 網絡是無中心服務器、依靠用戶羣交換信息的互聯網體系。



The Death of CPU Scaling

- ▶ Increase of transistor density \neq performance
 - ▶ The power and clock speed improvements collapsed

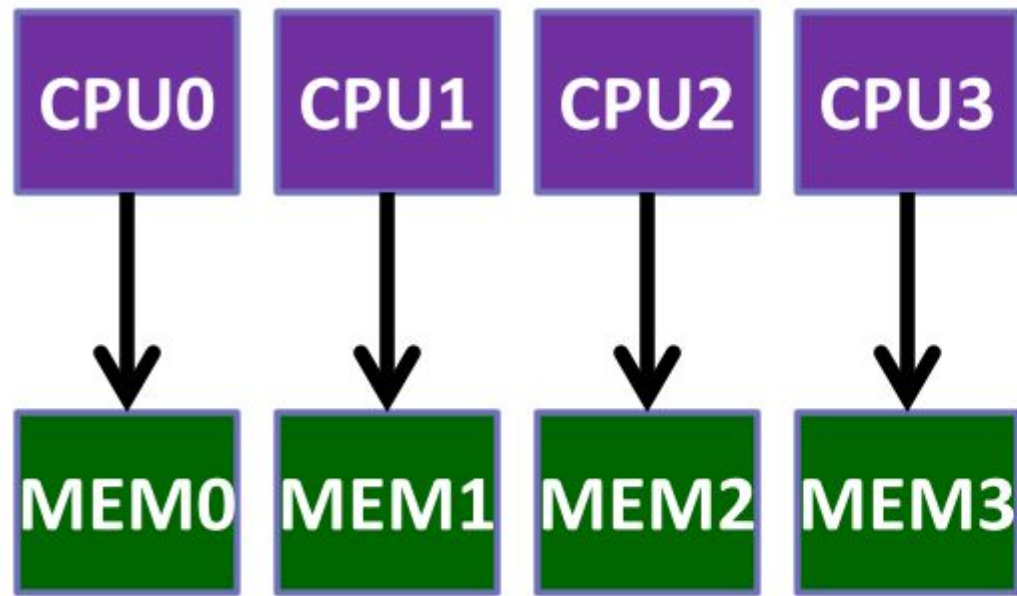


“Parallel Computing
is a trend and
essential tools in
today’s world!”

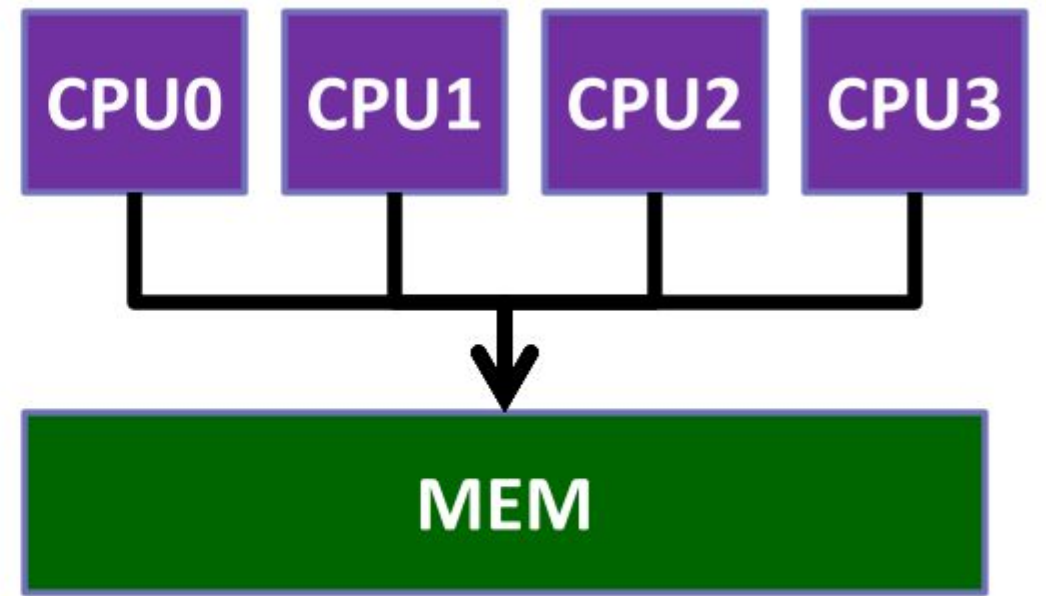
Outline

- ▶ Parallel Computing Introduction
 - ▶ What is parallel computing
 - ▶ Why need parallel computing
- ▶ **Parallel Computers**
 - ▶ Flynn's classic taxonomy
 - ▶ **Shared memory vs. Distributed memory parallel computer systems**
 - ▶ Interconnection networks & InfiniBand
 - ▶ Supercomputers

Shared memory vs. Distributed memory



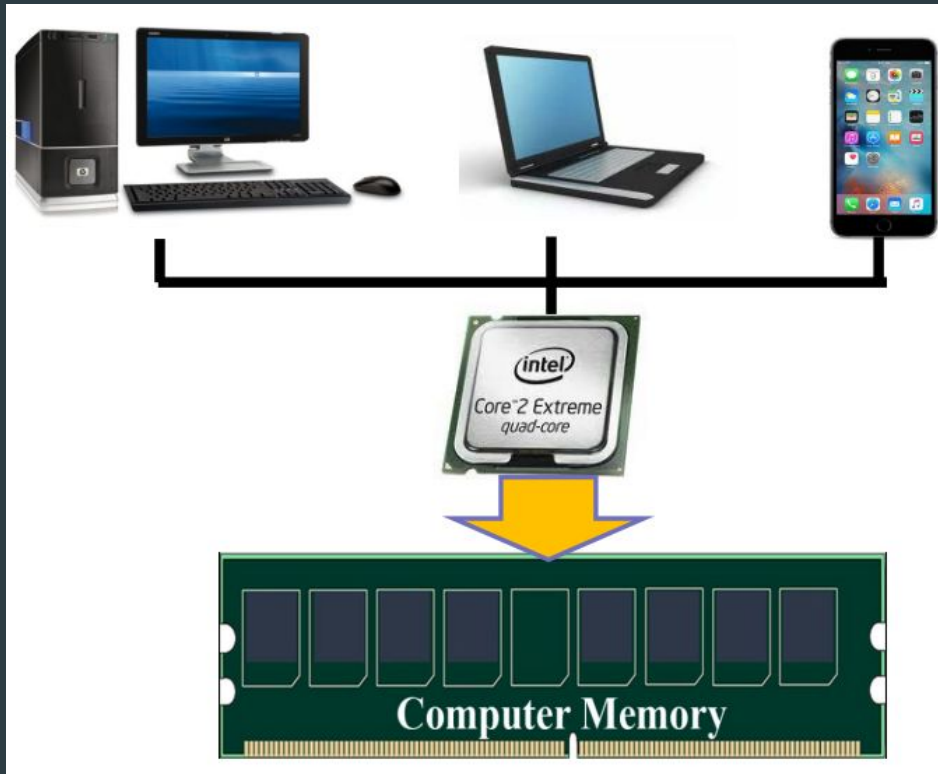
Distributed memory



Shared memory

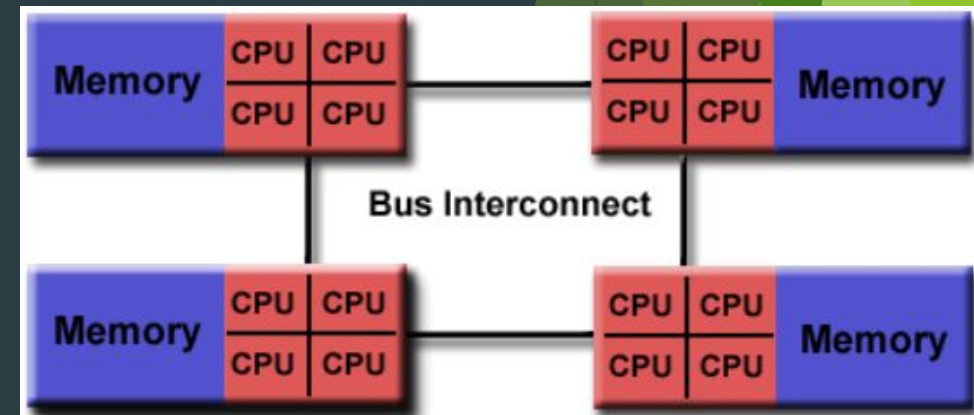
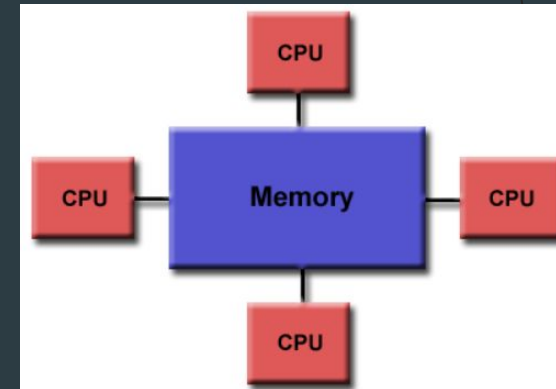
Shared Memory Multiprocessor Computer System

- ▶ Single computer with multiple internal multi-core processors



Shared Memory Computer Architecture

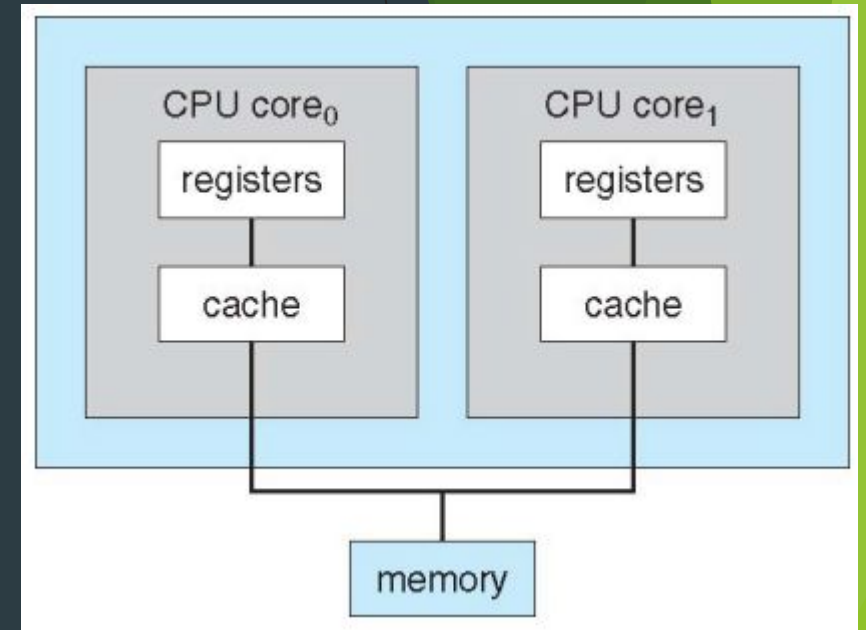
- ▶ Uniform Memory Access (UMA):
 - ▶ Most commonly represented today by Symmetric Multiprocessor (SMP) machines
 - ▶ Identical processors
 - ▶ Equal access times to memory
 - ▶ Example: commercial servers
- ▶ Non-Uniform Memory Access (NUMA):
 - ▶ Often made by physically linking two or more SMPs
 - ▶ One SMP can directly access memory of another SMP
 - ▶ Memory access across link is slower
 - ▶ Example: HPC server



Cache Coherence & Data Consistency

- ▶ **Cache coherence** problem
 - ▶ Cache invalidation / cache copy
 - ▶ It is difficult because of the non-zero latencies of the interconnect network
 - ▶ Solved by hardware solutions
- ▶ **Data consistency** problem
 - ▶ Synchronization protocol
 - ▶ Locking is required
 - ▶ Avoid by programmers

```
P1:  
While(1){  
    Counter++;  
}  
P2  
While(1){  
    Counter--;  
}
```

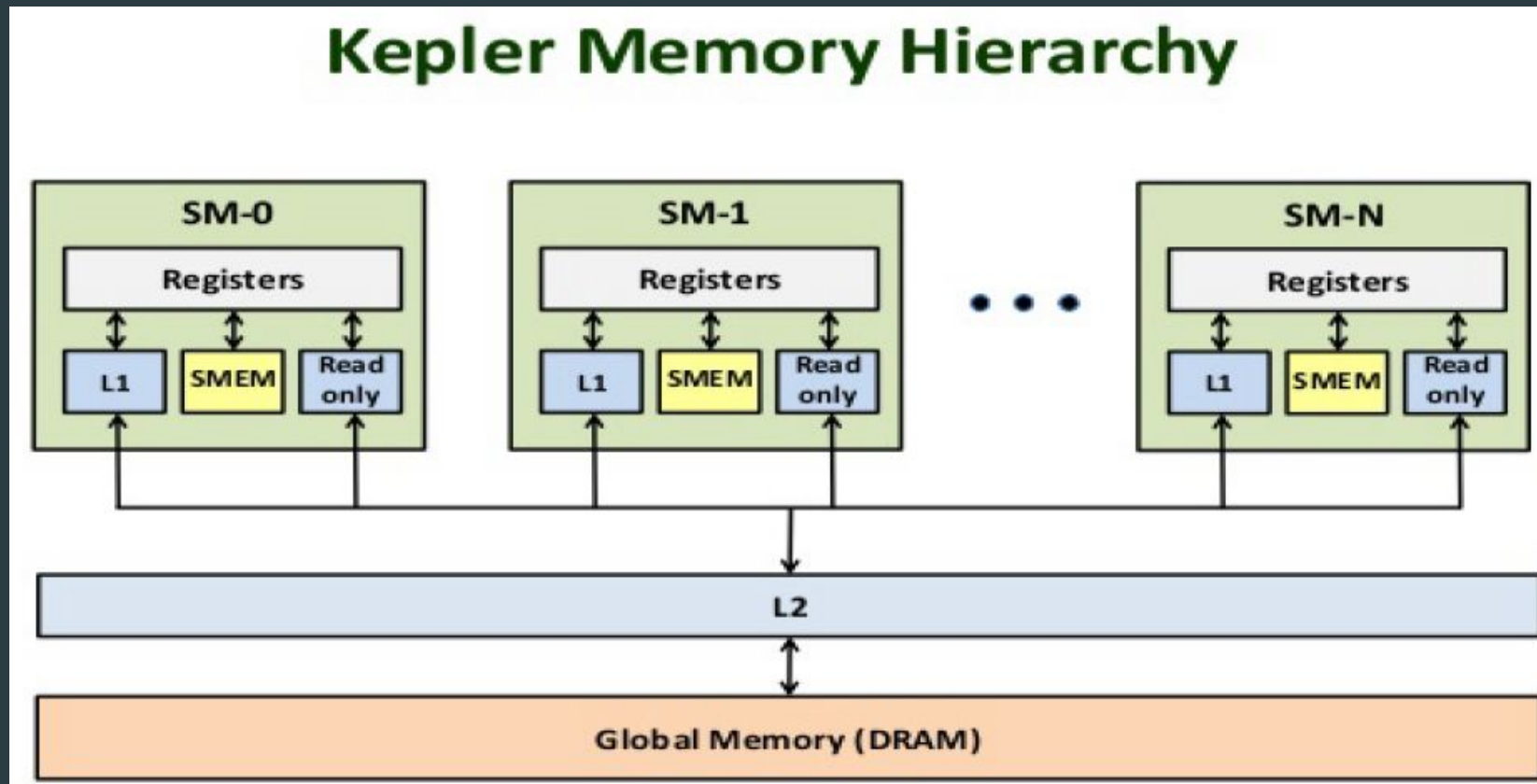


Many-core co-processor/accelerator

	Intel Xeon E5-2697 v3 CPU (Haswell)	NVIDIA Tesla K80 GPU (Kepler)	Intel Xeon Phi 7120P (Knight's Corner)
Cores	14	32	61
Logical Cores	28	2496(CUDA cores)	244
Frequency	2.6GHz	562 MHz	1.238GHz
GFLOPS(double)	583	1,455	1208
Max memory	768GB	12GB	16GB
Max Mem BW	68GB/s	240GB/s	352GB/s
Price	2700USD	5000USD	4000USD

Memory Hierarchy in GPU

- Only global memory is shared among all processors, but the speed is 10x slower than local shared memory



Distributed Memory Multicomputer

- Connect multiple computers to form a computing platform without sharing memory



Cluster: tens of servers



Supercomputer:
hundreds of servers

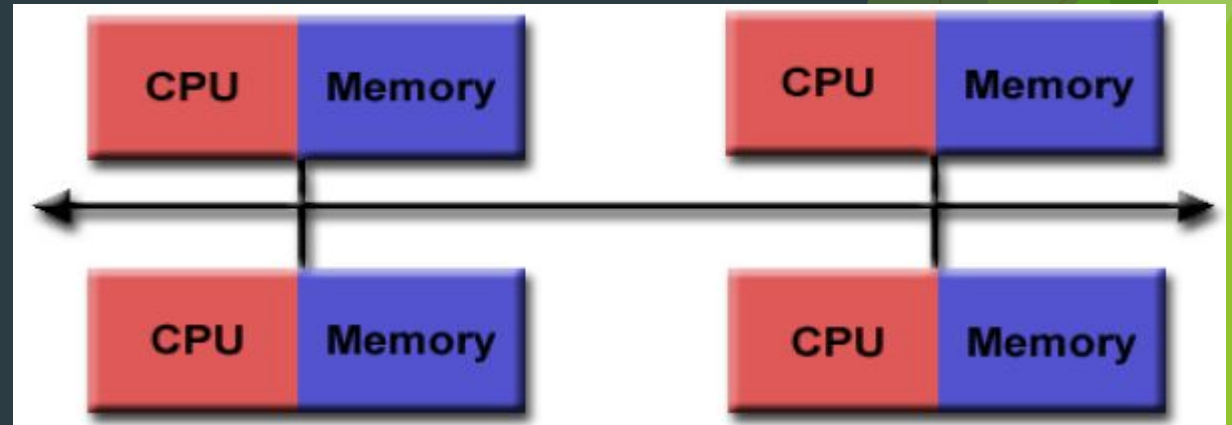


Datacenter: thousands of servers

Distributed Memory Multicomputer

- ▶ Require a communication network (i.e. not bus) to connect inter-processor(處理器間) memory
- ▶ Processors have their own memory & address space
- ▶ Memory change made by a processor has **NO** effect on the memory of other processors
- ▶ Programmers or programming tools are responsible to explicitly define how and when data is communicated between processors

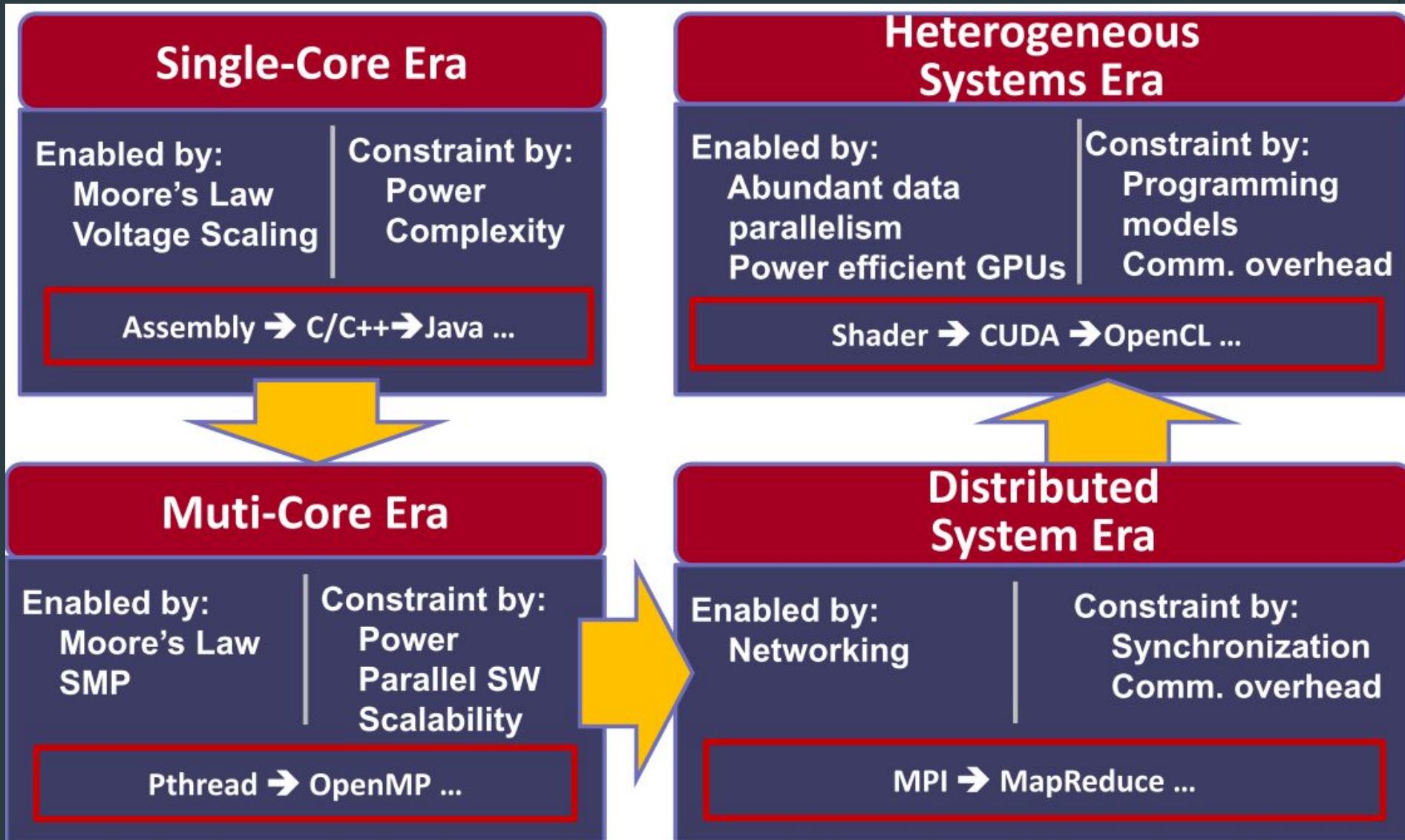
Network fabric:
Ethernet,
InfiniBand



Summary

- ▶ Shared memory
 - ▶ Global address space (全域的記憶體空間) provides a user-friendly programming perspective to memory
 - ▶ Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs (記憶體靠近CPU)
- ▶ Distributed memory
 - ▶ Memory is scalable with the number of processors
 - ▶ No cache coherence problem
 - ▶ Cost effectiveness: can use commodity, off-the-shelf (現成的) processors and networking

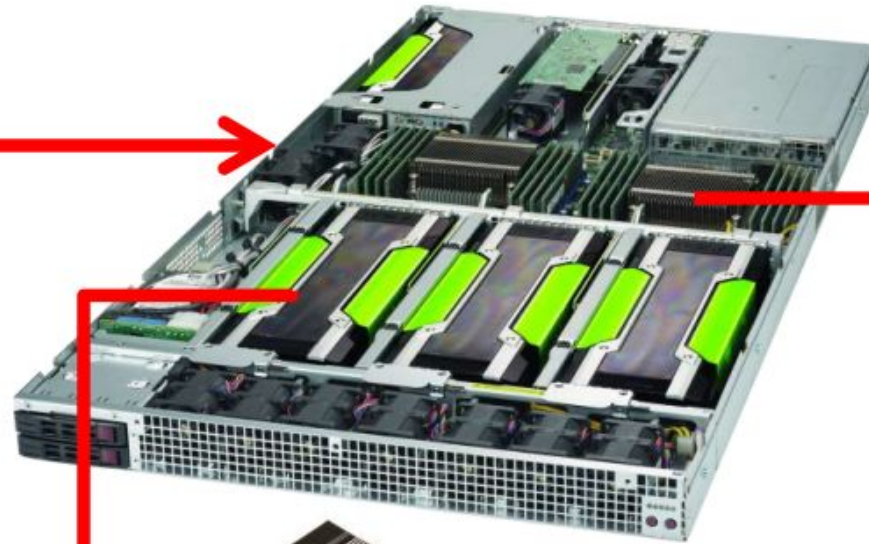
Trend of Parallel Computers



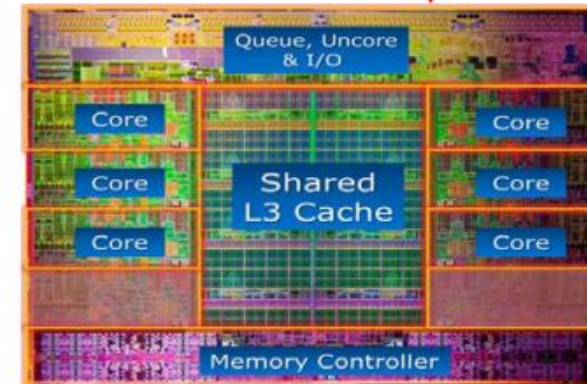
Today's Typical Parallel Computers

Racks: 16~42U

Node/Server: 1~4U



**Multi-core Processor:
100x cores/1000xthreads**



Co-Processor: 4~12 cores

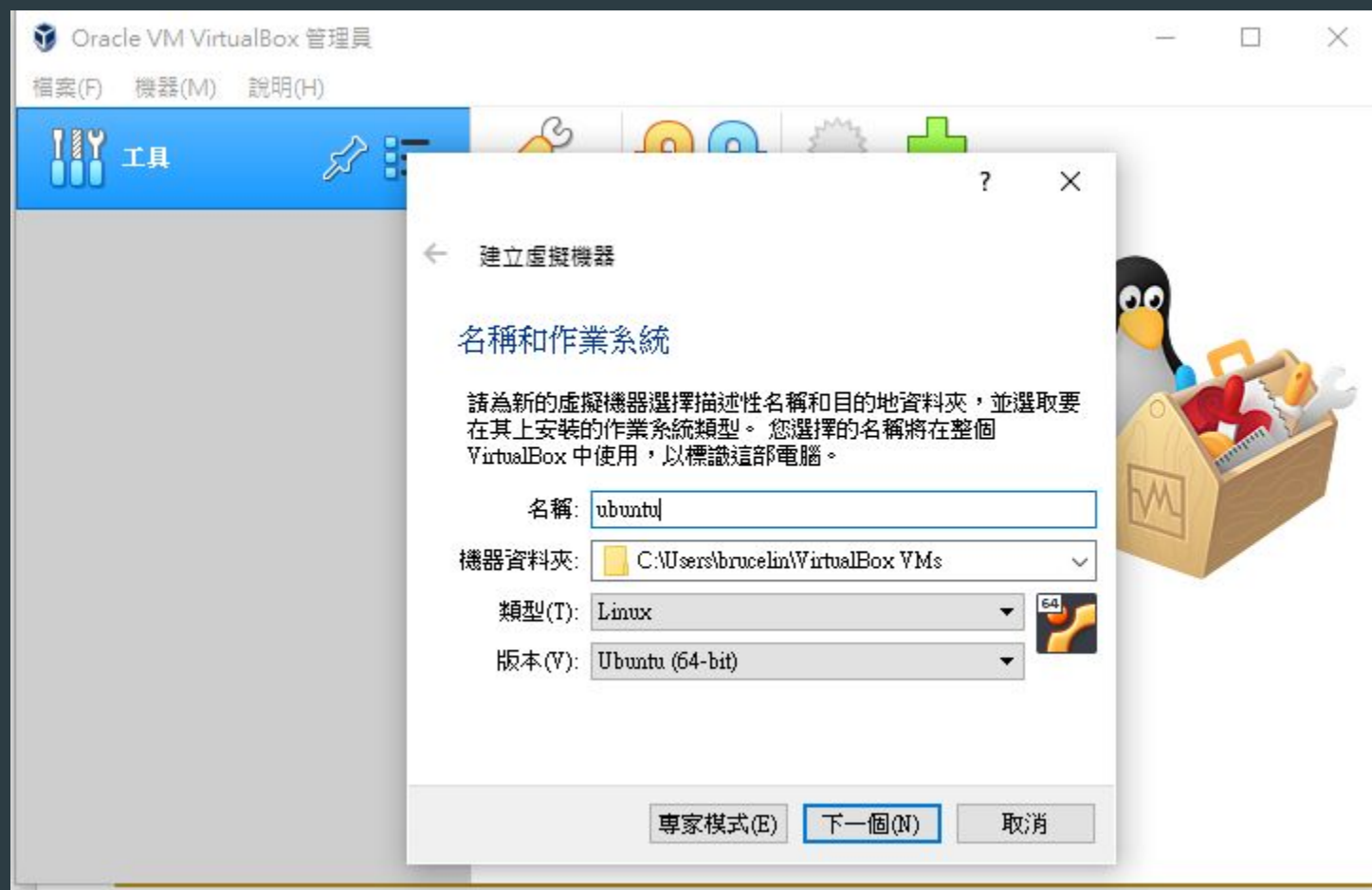
Lab 1

- ▶ 安裝VirtualBox(<https://www.virtualbox.org/>)
- ▶ 下載Ubuntu(<http://free.nchc.org.tw/ubuntu-cd/bionic/>)
 - ▶ ubuntu-18.04.4-desktop-amd64.iso
- ▶ 安裝Ubuntu on VirtualBox
- ▶ 安裝vim, gcc

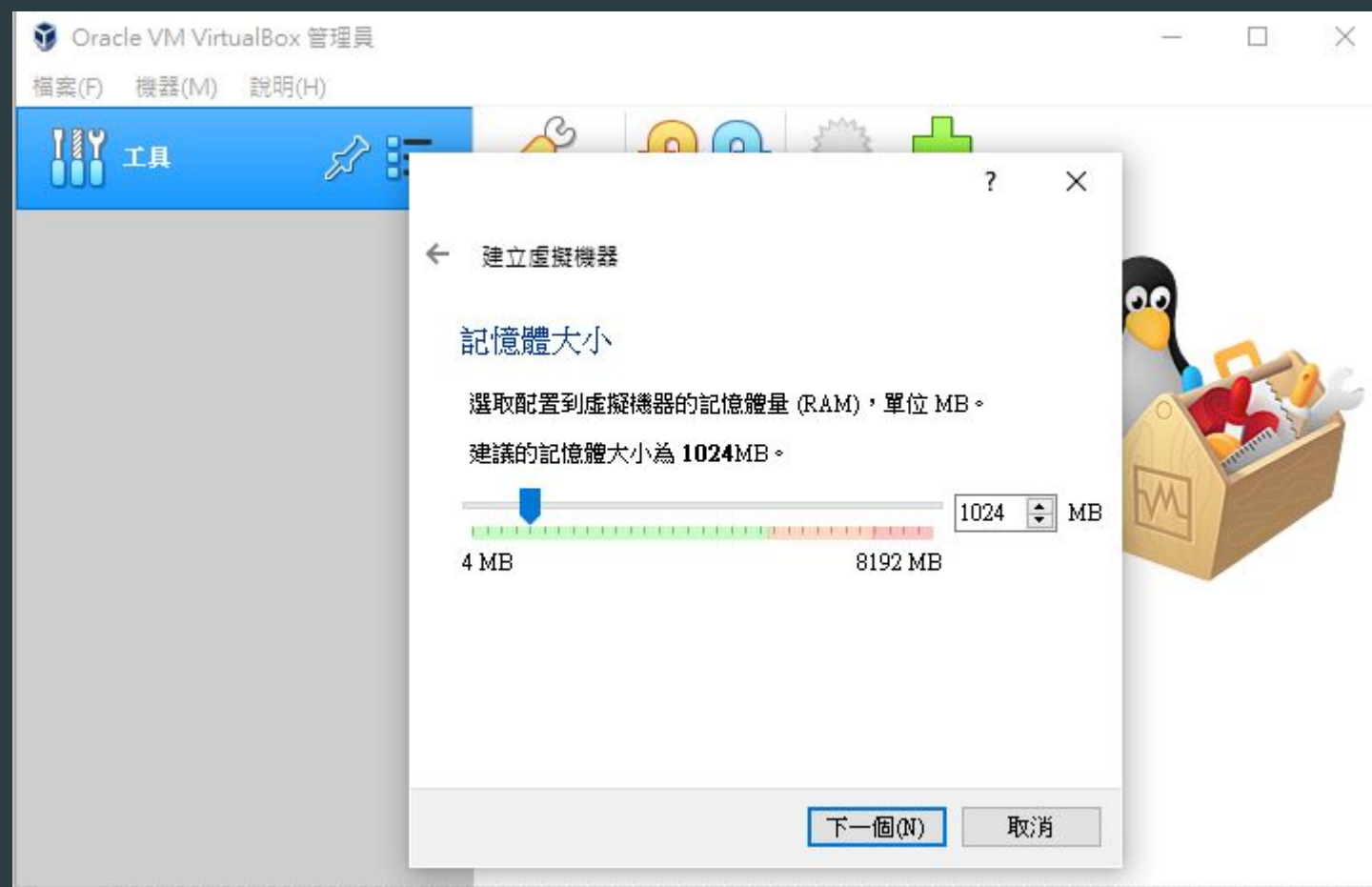
The steps of installing ubuntu on virtual box



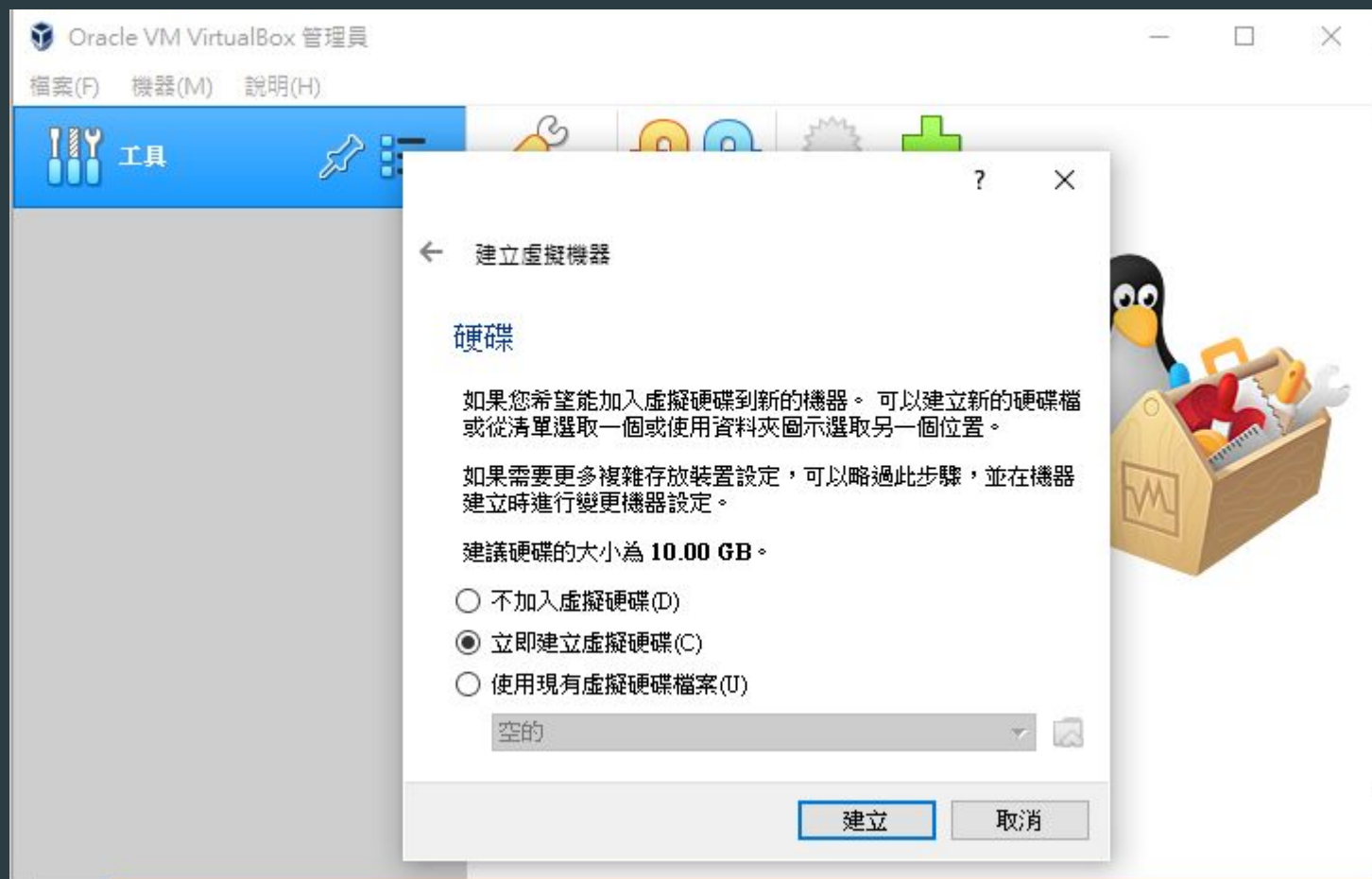
新增



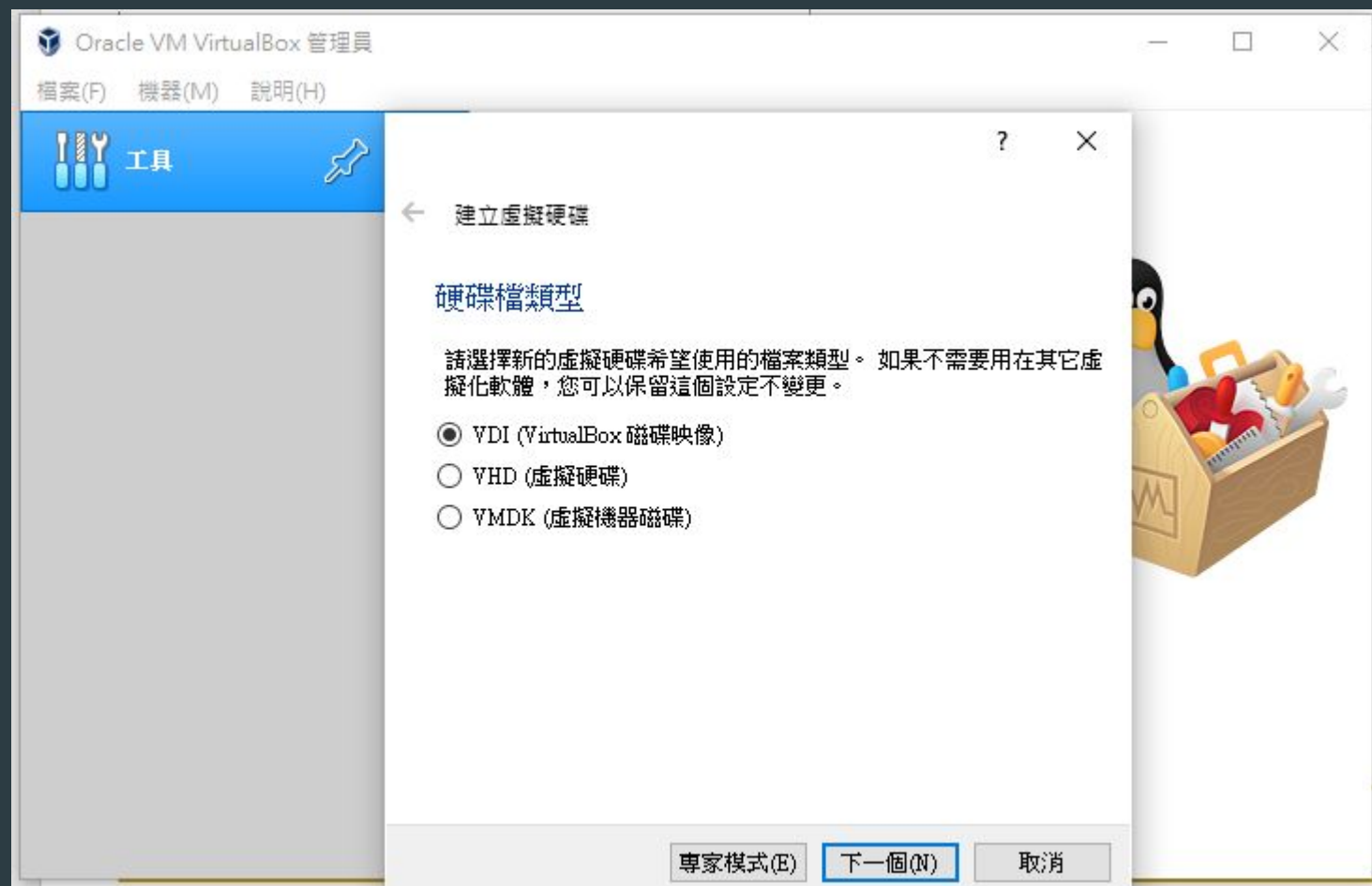
Setup memory size

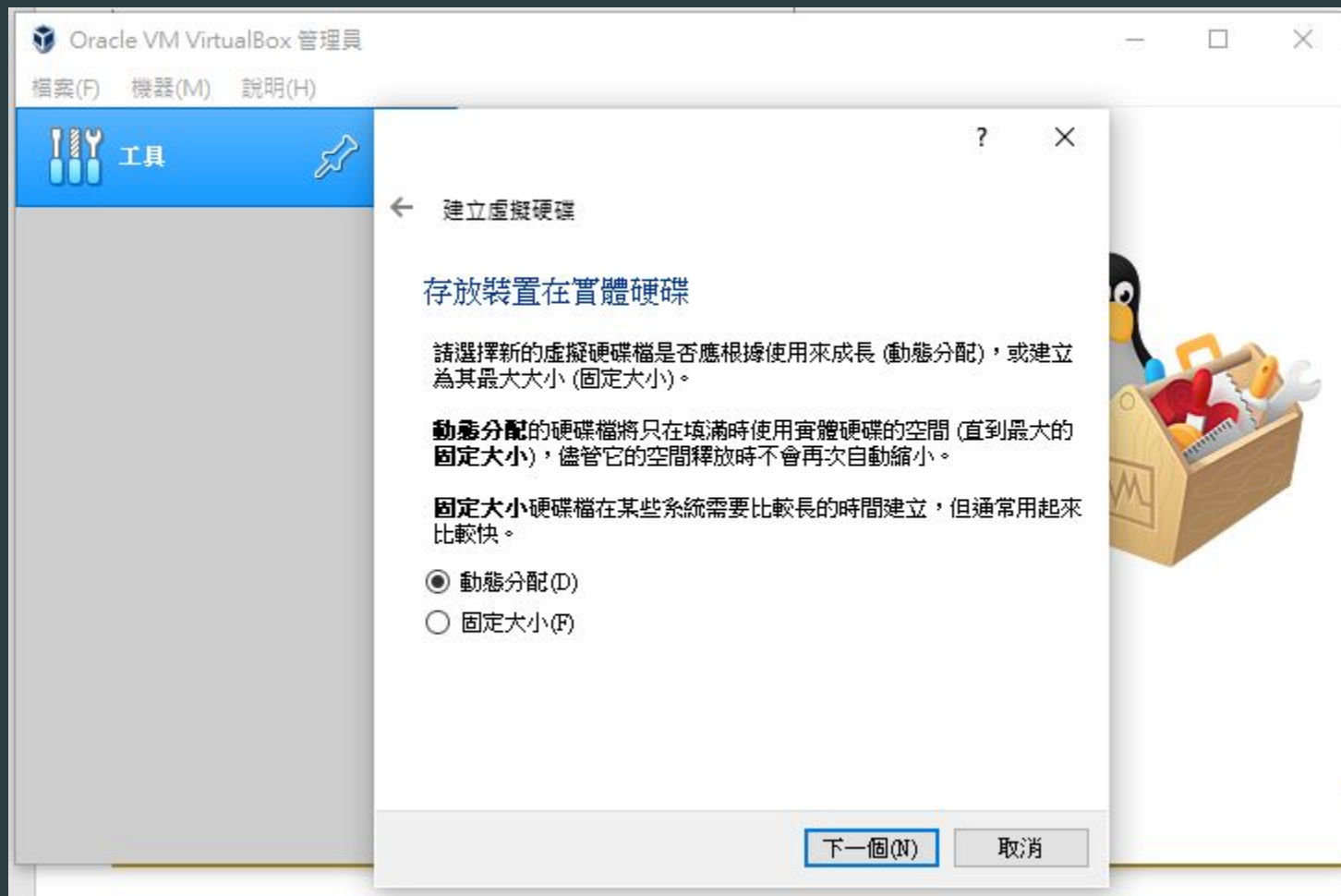


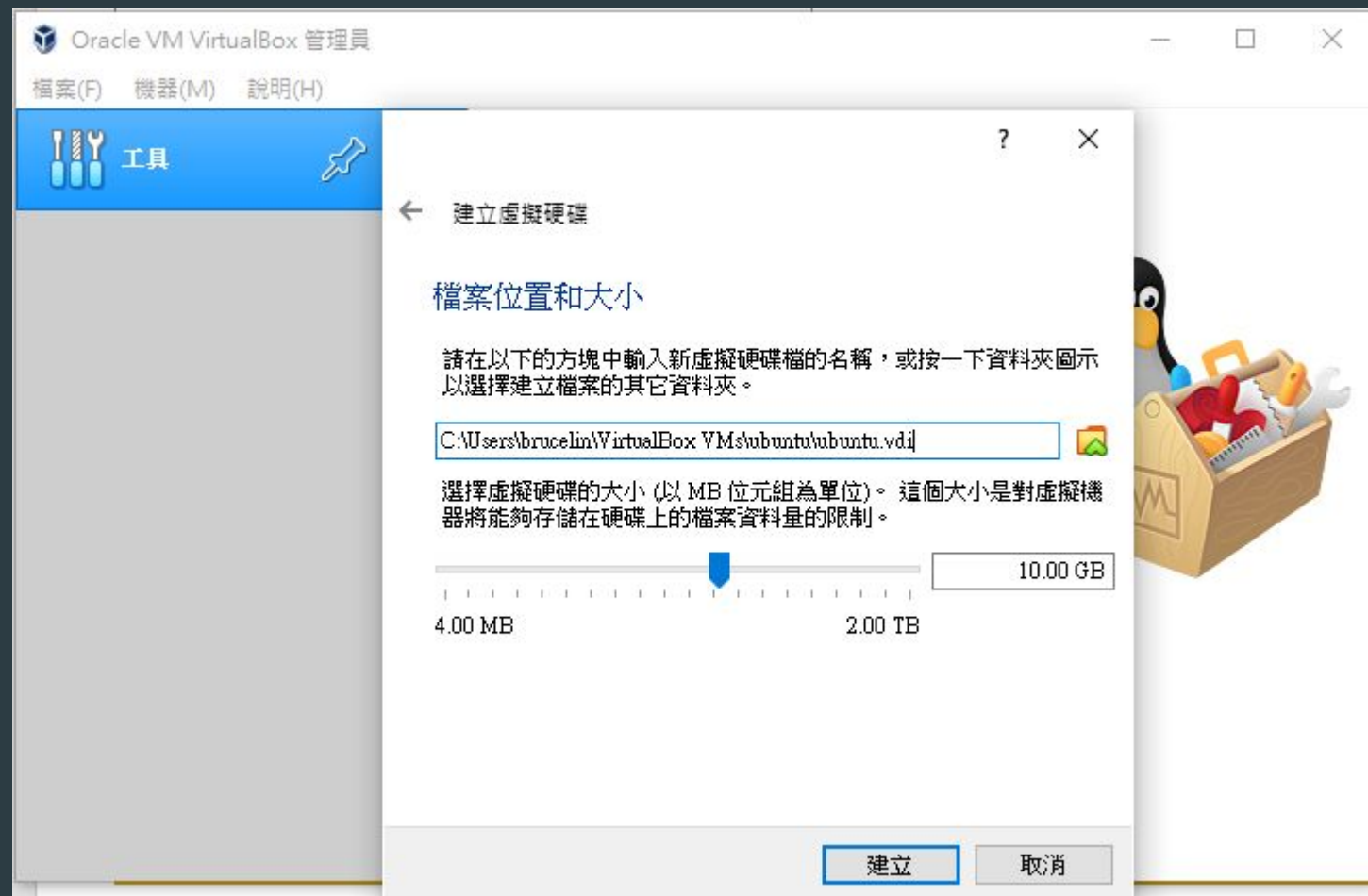
Setup HD size



設定硬碟型態









您已開啟「自動擷取鍵盤」選項。這將造成虛擬機器在每次啟動 VM 視窗時自動「擷取」鍵盤，並使其對主機上執行的其它應用程式不可

虛擬機器報告客體作業系統支援滑鼠指標整合。這意味著您不需要解放滑鼠指標就可以在客體作業系統中使用它--當滑鼠指

Welcome

English

Español

Esperanto

Euskara

Français

Gaeilge

Galego

Hrvatski

Íslenska

Italiano

Kurdî

Latviski

Lietuviškai

Magyar

Nederlands

No localization (UTF-8)

Norsk bokmål

Norsk nynorsk



Try Ubuntu

You can try Ubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

You may wish to read the [release notes](#).



Install Ubuntu



Trash

[Show Applications](#)



🔍 Type to search...



Start...



Sudoku



Syste...



Termi...



Text ...



Thun...



To Do



Trans...



Ubun...



Utilit...



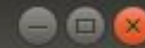
Videos

Frequent

All



bruce@bruce-VirtualBox: ~



File Edit View Search Terminal Help

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

bruce@bruce-VirtualBox:~\$

Lab1

- ▶ 安裝vim: `sudo apt install vim`
- ▶ 安裝gcc: `sudo apt install gcc`
- ▶ 建立子目錄lab1: `mkdir lab1`
- ▶ 進入子目錄: `cd lab1`
- ▶ 啟動vim編輯test.c: `vim test.c`
- ▶ 查看目錄內容: `ls`
- ▶ 編譯test.c: `gcc test.c -o test`
- ▶ 執行test: `./test`
- ▶ 編譯test.c: `gcc -Wall -g -o test test.c`
- ▶ 啟動gdb: `gdb -q test`

編譯與連結 C 程式

- ▶ 假設我們寫好一個 C 程式，常程式碼儲存在 `hello.c` 這個檔案中，而其內容如下：

```
// hello.c
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

- ▶ 若要將 C 的原始碼編譯成執行檔，可以執行 `gcc` 並指定 C 語言的原始碼檔案：
 - ▶ `gcc hello.c`
- ▶ GCC 預設會執行編譯與連結，直接產生一個可以執行的程式，輸出至 `a.out` 這個檔案，完成編譯之後，即可執行這個程式：
 - ▶ `./a.out`

- ▶ 若要指定輸出的執行檔名稱, 可以加上 `-o` 參數, 並指定輸出的檔案名稱:
- ▶ # 編譯 C 程式, 指定輸出檔名
- ▶ `gcc -o hello hello.c`
- ▶ # 執行編譯好的程式
- ▶ `./hello`

GCC 常用參數

▶ 除錯用參數

```
// bug.c
#include <stdio.h>
int main() {
    int v; // 未初始化
    printf("v = %d\n", v);
    return 0;
}
```

▶ -Wall:編譯器顯示所有的警告訊息

```
bruce@bruce-VirtualBox:~/lab1$ gcc -Wall test2.c -o test2
test2.c: In function 'main':
test2.c:5:2: warning: 'v' is used uninitialized in this function [-Wuninitializ
ed]
    printf("%d\n", v);
    ^~~~~~
```

- ▶ 而通常在開發與除錯的階段, 都會使用 `gdb` 來除錯, 所以也會加上 `-g` 參數
- ▶ 所以在開發階段, 會很常用到像這樣的指令來編譯程式:
- ▶ # 顯示警告訊息、加入除錯資訊
- ▶ `gcc -Wall -g -o hello hello.c`

定義巨集(Macro)

- ▶ 在開發中大型的程式時，開發者通常會使用一些巨集來處理各種的問題，例如除錯時會需要輸出詳細的訊息，但正式版的程式又不需要，這時候就可以這樣寫：

```
// macro.c
#include <stdio.h>
int main() {
    printf("Hello, world!\n");

    #ifdef DEBUG
        printf("DEBUG is defined.\n");
    #endif

    return 0;
}
```

- ▶ 這段程式中，我們用 `#ifdef` 判斷 `DEBUG` 這個變數是否有被定義，若有被定義的話，就加入輸出除錯的程式碼，這樣一來我們就可以透過 `DEBUG` 的定義，來決定是否要產生有除錯訊息的執行檔。
- ▶ 若要定義 `DEBUG` 這個變數，可將定義直接加在程式碼中：

```
// 定義 DEBUG
```

```
#define DEBUG
```

- ▶ 也可以直接透過 GCC 的 `-D` 參數來定義：

```
# 定義 DEBUG
```

```
gcc -DDEBUG test.c
```

最佳化

- ▶ GCC 可以根據 CPU 的架構，進行最佳化處理，編譯出效能更好的執行檔，可用的選項有 -O (跟 -O1 相同)、-O2 與 -O3，數字越高代表最佳化的程度越高，許多專案在編譯正式版的時侯，都會使用 -O2 進行最佳化。

進行最佳化

```
gcc -O2 -o hello hello.c
```

```
#include <stdio.h>
int main(){
    int a, b, c;
    scanf("%d%d", &a, &b);
    c = a + b;
    printf("%d\n", c);
    return 0;
}
```


- ▶ 要使用gdb前, 用gcc編譯程式時, 要加上-g
- ▶ `$gcc -Wall -g test.c -o test`

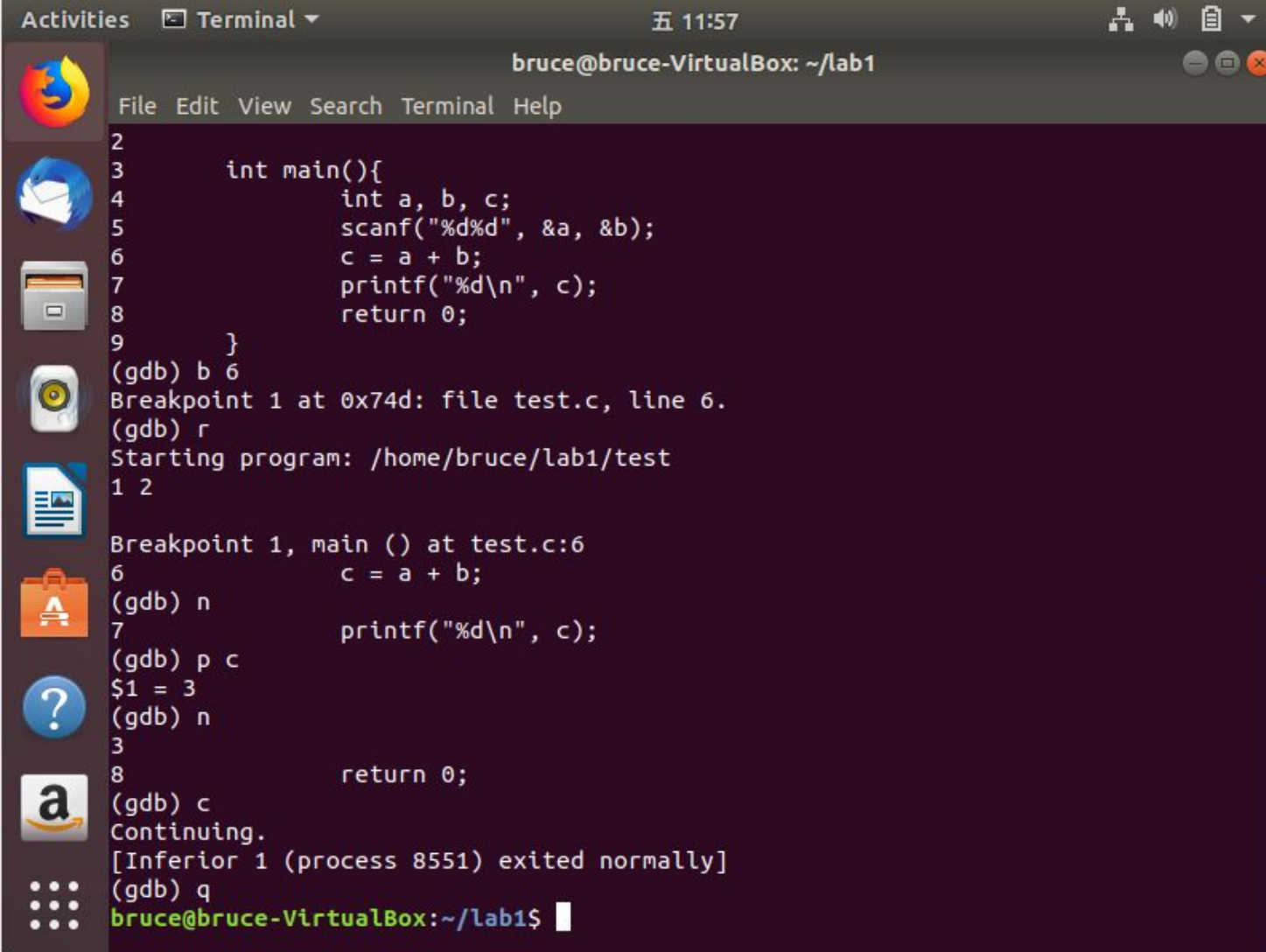
```
brucelin@brucelin-VirtualBox: ~/lab1
File Edit View Search Terminal Help
brucelin@brucelin-VirtualBox:~/lab1$ ls
test.c
brucelin@brucelin-VirtualBox:~/lab1$ gcc -Wall -g test.c -o test
brucelin@brucelin-VirtualBox:~/lab1$ ls
test  test.c
brucelin@brucelin-VirtualBox:~/lab1$ ./test
1 2
3
brucelin@brucelin-VirtualBox:~/lab1$
```

- ▶ list or l
- ▶ break or b
- ▶ run / r

```
Activities  Terminal  五 11:56  bruce@bruce-VirtualBox: ~/lab1
File Edit View Search Terminal Help
bruce@bruce-VirtualBox:~/lab1$ gdb -q test
Reading symbols from test...done.
(gdb) l
1      #include <stdio.h>
2
3      int main(){
4          int a, b, c;
5          scanf("%d%d", &a, &b);
6          c = a + b;
7          printf("%d\n", c);
8          return 0;
9      }
(gdb) b 6
Breakpoint 1 at 0x74d: file test.c, line 6.
(gdb) r
Starting program: /home/bruce/lab1/test
1 2

Breakpoint 1, main () at test.c:6
6          c = a + b;
(gdb) n
7          printf("%d\n", c);
(gdb) p c
$1 = 3
(gdb) n
3
8          return 0;
(gdb) c
Continuing.
```

- ▶ next / n
- ▶ continue / c
- ▶ print / p



```
Activities Terminal 五 11:57
bruce@bruce-VirtualBox: ~/lab1
File Edit View Search Terminal Help
2
3     int main(){
4         int a, b, c;
5         scanf("%d%d", &a, &b);
6         c = a + b;
7         printf("%d\n", c);
8         return 0;
9     }
(gdb) b 6
Breakpoint 1 at 0x74d: file test.c, line 6.
(gdb) r
Starting program: /home/bruce/lab1/test
1 2

Breakpoint 1, main () at test.c:6
6         c = a + b;
(gdb) n
7         printf("%d\n", c);
(gdb) p c
$1 = 3
(gdb) n
3
8         return 0;
(gdb) c
Continuing.
[Inferior 1 (process 8551) exited normally]
(gdb) q
bruce@bruce-VirtualBox:~/lab1$
```

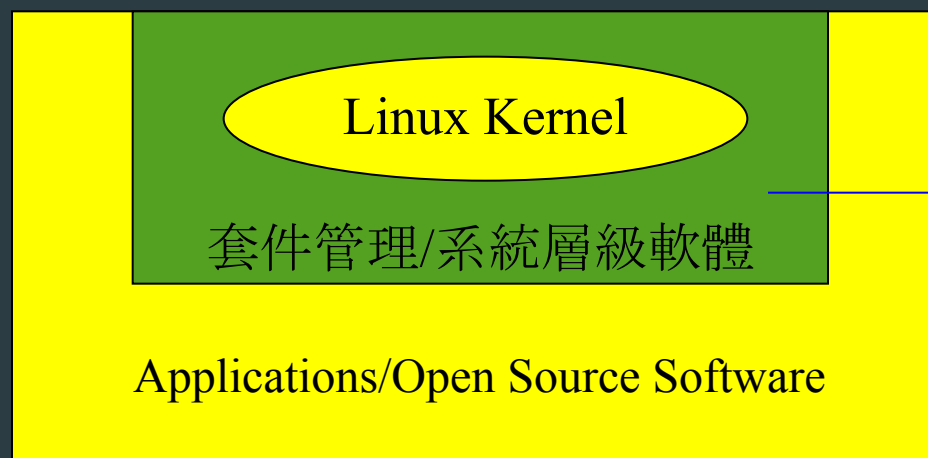
什麼是 Linux?

- ▶ Linux 是一種自由開放原始碼的類 Unix 的作業系統，其廣泛運用於伺服器 and 嵌入式系統中。
- ▶ Linux的由來
 - ▶ UNIX: 1970年代由Bell Lab發展出來，並廣泛使用於當時的minicomputers和workstations上
 - ▶ 但當時的 UNIX只能在特定的主機上運行，並且有版權的問題
 - ▶ Andrew Tanenbaum為了OS之教學，根據 UNIX重新撰寫一套可在PC上運行的系統 Minix (1986)
 - ▶ 1991年Linus Torvalds又改寫Minix成為Linux
 - ▶ GNU又為Linux kernel加上許多公用程式

Linux Distributions

- ▶ 目前主流的 Linux 發佈版本 (Linux distributions, 係指可完整安裝使用的套件版本) 包括: Debian、Fedora、CentOS、Ubuntu 等
 - ▶ 核心都是Linux kernel, 而公用程式也大都是GNU 軟體及其他開放源碼軟體 (Open Source Software)
 - ▶ 最主要的差別在於專屬工具及套件管理方式

Linux Distribution 簡單示意圖



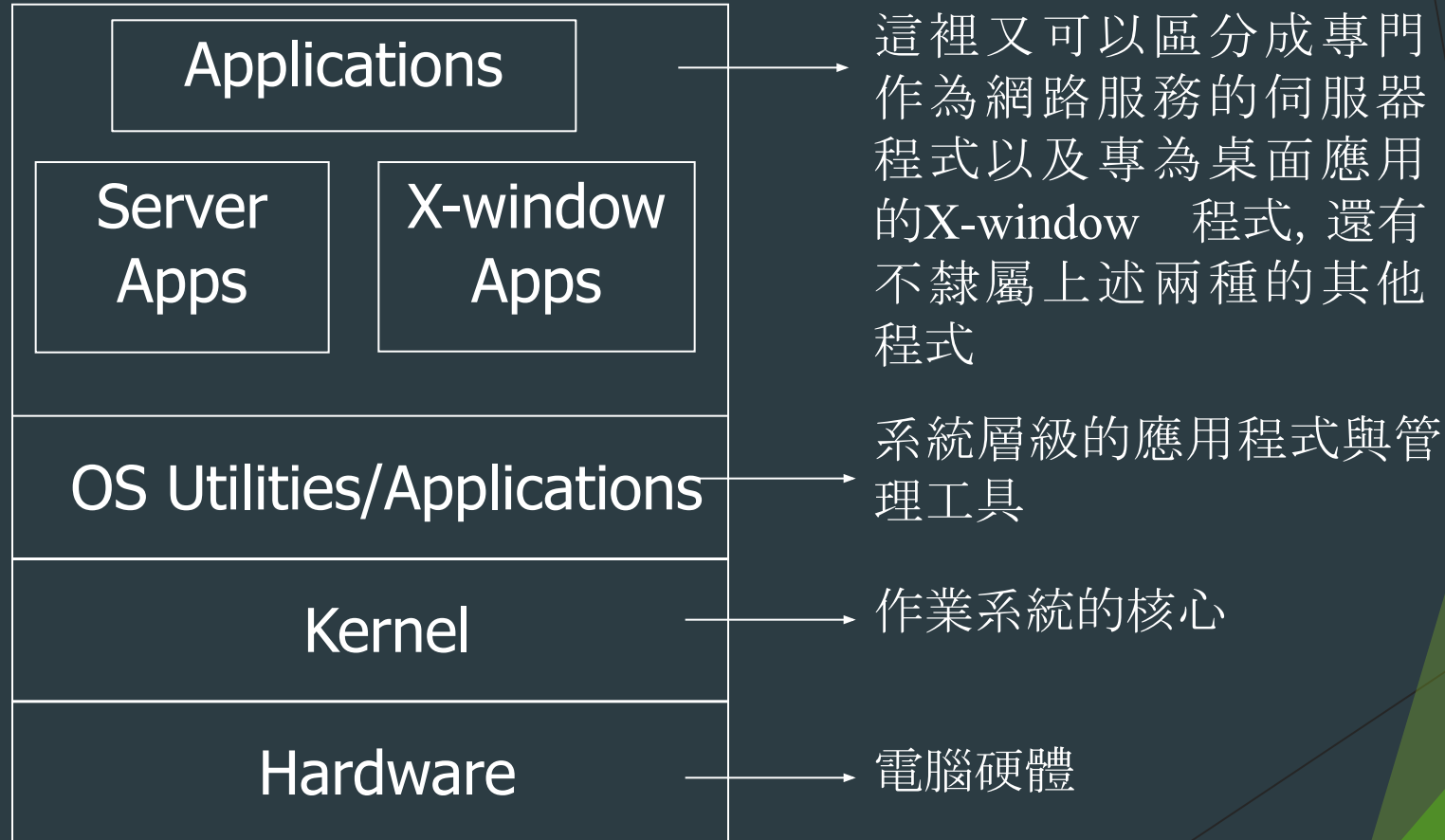
除了『系統層級』的軟體會有所不同以外, 系統的核心及最上層的應用程式, 基本上都是一樣的。

Why Linux?

- ▶ Used in many servers and embedded systems
 - ▶ Web servers, mail servers, firewalls, printer servers, wifi access points, disk servers
 - ▶ Google phone (Android), OpenMoko
 - ▶ Note: Many do not have GUI, but text mode interface

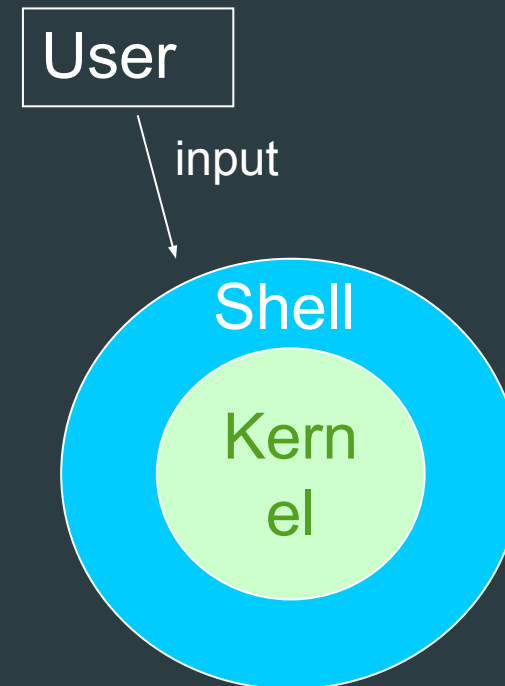
Linux系統結構

- Linux作業系統由以下幾個部分所組成：



Linux from Users' Perspective

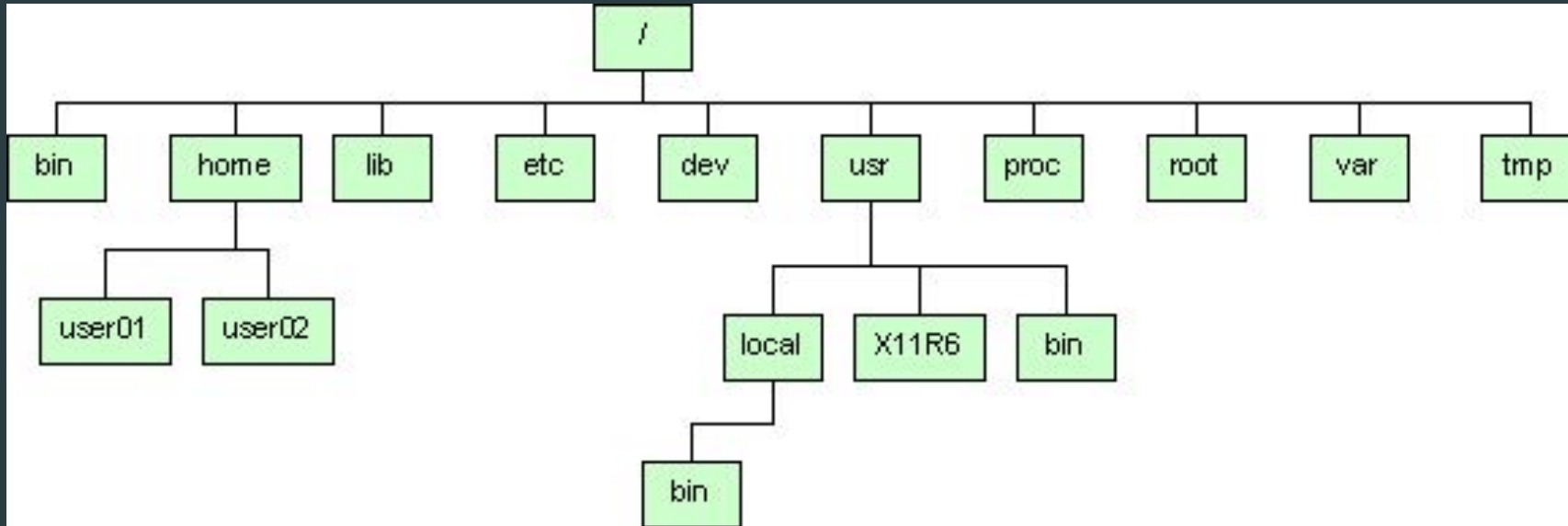
- ▶ Underneath the windows GUI system, users interact with Linux kernel through *shell*
 - ▶ Shell interprets user's input as commands and pass them to kernel
 - ▶ Operates in a simple loop:
 - ▶ accepts a command
 - ▶ interprets the command
 - ▶ executes the command
 - ▶ displays a "prompt," to notify user that it is ready to accept the next command
- ▶ 使用終端機程式



Directory Structure

- ▶ Linux統一由/ (root)作為根目錄, 系統內所有的目錄都依附在/ 底下
 - ▶ 不同的硬碟或分割區 (partition), 也都在/ 底下, 透過mount (掛載)的方式, 掛載起來使用
 - ▶ Windows則是由C:\, D:\的順序編排而成
 - ▶ All directories are in a **hierarchical structure (tree structure)**
 - ▶ Users have the own directory (**home** directory)
 - ▶ The *path* is the location of a file or directory within the tree, e.g. /usr/sbin/bzip2, ../../john, /home/john
 - ▶ Support access control by defining permissions for read, write, and execution

Directory Structure



Linux 檔案系統架構

- ▶ / 根目錄
- ▶ /bin, /sbin
 - ▶ /bin 主要放置一般使用者可以操作的指令, /sbin 放置系統管理員可以操作的指令。
- ▶ /boot
主要放置開機相關檔案
- ▶ /dev
放置 device 裝置檔案, 包括滑鼠鍵盤等
- ▶ /etc
主要放置系統檔案
- ▶ /home, /root
 - ▶ /home 主要是一般帳戶的家目錄, /root 為系統管理者的家目錄

- ▶ **/lib, /lib64**

主要為系統函式庫和核心函式庫, 若是 64 位元則放在 /lib64

- ▶ **/proc**

將記憶體內的資料做成檔案類型

- ▶ **/sys**

與 /proc 類似, 但主要針對硬體相關參數

- ▶ **/usr**

/usr 全名為 **unix software resource** 縮寫, 放置系統相關軟體、服務 (注意不是 **user** 的縮寫喔!)

▶ **/var**

全名為 variable, 放置一些變數或記錄檔

▶ **/tmp**

全名為 temporary, 放置暫存檔案

▶ **/media, /mnt**

放置隨插即用的裝置慣用目錄, /mnt 為管理員/使用者手動掛上 (mount) 的目錄

▶ **/opt**

全名為 optional, 通常為第三方廠商放置軟體處

▶ **/run**

系統進行服務軟體運作管理處

▶ **/srv**

通常是放置開發的服務 (service), 如: 網站服務 www 等

Other Things to Know

- ▶ Normal user and super user:
 - ▶ There is one special user for administrator, which can do anything and is called **root** or **superuser**
- ▶ Case sensitivity:
 - ▶ Linux is case-sensitive
- ▶ Multi-user and multi-process:
 - ▶ Many people can use a machine at the same time
- ▶ File and process
 - ▶ Almost everything (data, directory, process, hard disk) are expressed as a file
 - ▶ *Process* is a running program identified by a unique id (PID)

Basic Commands

- ▶ When you interact with shell, you will see the prompt, something like **king@my_pc:~\$**
- ▶ A command consists of three parts, i.e. command name, options, arguments

king@my_pc:~\$ command-name optionA optionB arg1 arg2

- ▶ Options always start with "-"
- ▶ Example:

```
cd    ..  
ls    -l    .bashrc  
mv    fileA  fileB
```

Basic Commands

- ▶ `ls` show files in current position
- ▶ `cd` change directory
- ▶ `cp` copy file or directory
- ▶ `mv` move file or directory
- ▶ `rm` remove file or directory
- ▶ `pwd` show current position
- ▶ `mkdir` create directory
- ▶ `rmdir` remove directory
- ▶ `less, more` display file contents
- ▶ `man` display online manual

Basic Commands

- ▶ `su` switch to superuser
- ▶ `passwd` change password
- ▶ `mount` mount file system
- ▶ `umount` unmount file system
- ▶ `df` show disk space usage
- ▶ `ps` show current processes
- ▶ `shutdown` reboot or turn off machine
- ▶ `grep XXX` show lines matching pattern XXX in File

Relative and Absolute Path

- ▶ Path means a position in the directory tree
- ▶ Two ways to express a path
 - ▶ Relative path expression: the path depends on the current directory
 - ▶ . : the current directory
 - ▶ .. : the parent directory
 - ▶ pwd: gives the current path
 - ▶ Absolute path expression: the path is defined uniquely from the root
 - /home/linux/
 - ~/linux (home directory/linux)

檔案與目錄管理指令

► ls

```
brucelin@brucelin-VirtualBox: ~  
brucelin@brucelin-VirtualBox:~$ ls  
darknet  Documents  examples.desktop  Pictures  Templates  
Desktop  Downloads  Music            Public    Videos  
brucelin@brucelin-VirtualBox:~$ ls -l  
total 48  
drwxrwxr-x 13 brucelin brucelin 4096 +- 12 2018 darknet  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Desktop  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Documents  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Downloads  
-rw-r--r--  1 brucelin brucelin 8980 +- 12 2018 examples.desktop  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Music  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Pictures  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Public  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Templates  
drwxr-xr-x  2 brucelin brucelin 4096 +- 12 2018 Videos  
brucelin@brucelin-VirtualBox:~$
```

- // -l 列出詳細資料 -a 列出隱藏資料

```
brucelin@brucelin-VirtualBox:~$ ls -a
.          Downloads      .sudo_as_admin_successful
..         examples.desktop Templates
.bash_history .gconf          .vboxclient-clipboard.pid
.bash_logout .gnupg          .vboxclient-display.pid
.bashrc      .ICEauthority   .vboxclient-draganddrop.pid
.cache       .local          .vboxclient-seamless.pid
.config      .mozilla        Videos
darknet      Music           .wget-hsts
.dbus        Pictures        .Xauthority
Desktop      .profile        .xsession-errors
Documents    Public          .xsession-errors.old
brucelin@brucelin-VirtualBox:~$
```

- pwd: print work directory, 印出目前工作目錄: \$ pwd
- cd: change directory, 移動進入資料夾
- 移動到家目錄: \$ cd ~
- 移動到上一層目錄: \$ cd ..
- 移動到根目錄: cd /

- ▶ `mkdir`: make directory, 創建新資料夾
- ▶ `cp`: copy, 複製檔案
- ▶ `mv`: move (rename) files, 移動檔案或是重新命名檔案

- ▶ 移動檔案:

- ```
$ mv README.md /examples/README.md
```

- ▶ 重新命名

- ```
$ mv README.md README_MV.md
```

- ▶ `rm`: remove file, 刪除檔案

- ▶

```
$ rm README.md
```

- ▶ 刪除目前資料夾下副檔名為 `.js` 檔案:

- ▶

```
$ rm *.js
```

- ▶ 刪除資料夾和所有檔案:

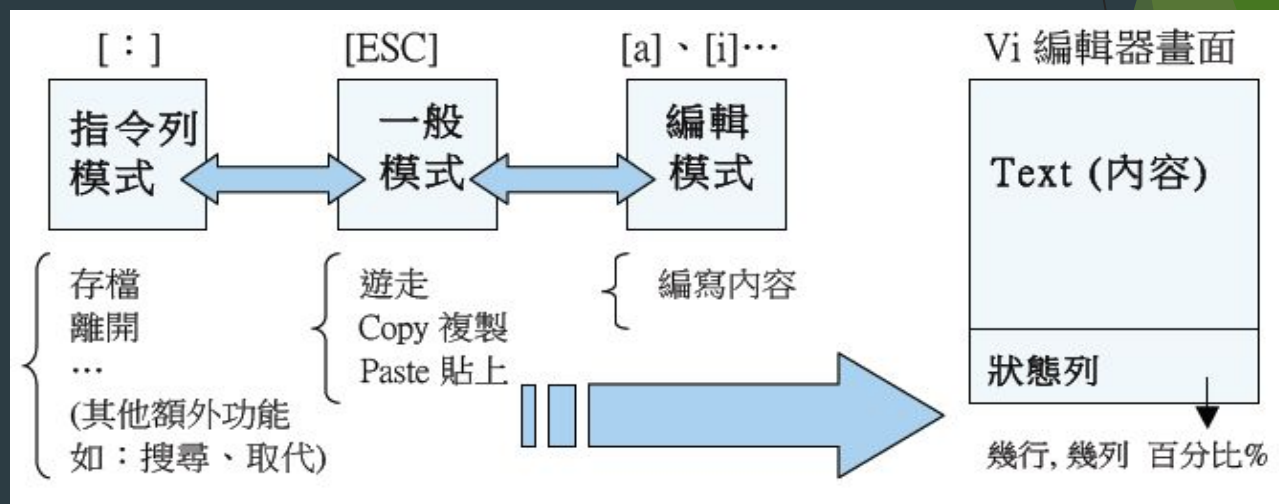
- ▶

```
$ rm -f examples
```


- ▶ `more`: 將檔案一頁頁印在終端機上
 - ▶ 可以使用上下移動換頁, 按 `q` 離開:
 - ▶ `$ more README.md`

編輯文字檔案：vim

- ▶ vim: 在終端機編輯文字檔案
- ▶ `$ vim README.md`
- ▶ 啟動後，使用 `i`或`a` 進入編輯，`esc` 離開編輯模式，`:q` 不儲存離開，`:wq` 儲存離開，`:q!` 強制離開
- ▶ vi 共分為三種模式：
 - ▶ 一般模式
 - ▶ 編輯模式
 - ▶ 指令列命令模式



vim編輯 (插入) 模式

[i]	從『目前位置』開始編輯
[I]	從『行首』開始編輯
[a]	從『下個字元』開始編輯
[A]	從『行尾』開始編輯
[o]	產生『下一行』空白行
[O]	在『游標位置』建立空白行
[ESC]	切換回『一般模式』

一般模式

[0]	到『行首』
[1G]	到『檔頭』
[\$]	到『行尾』
[G]	到『檔尾』
[w]	跳到『下一個字』
[x]	刪除『目前游標字元』
[X]	刪除『游標前一個字元』

一般模式

[:] + [NUM]	跳到『NUM』行
[y]	複製
[y][y]	複製一整行
[d]	刪除
[d][d]	刪除一整行
[d][0]	刪除至『行首』
[d][\$]	刪除至『行尾』

一般模式

[d] [1G]	刪除至『檔頭』
[NUM] + [..] + [..]	[3] + [d] + [d] 刪除三行
[..] + [NUM] + [..]	[d] + [3] + [w] 刪除三個字
[r]	取代『一個』字元
[x]	剪下 / 刪除
[p]	貼上
[u]	『復原』上一個動作

指令列模式

[:] [e]	編輯新檔
[:] [w]	儲存檔案
[:] [w] [q]	儲存檔案後離開 (『:x』或『ZZ』)
[:] [w] [q] [!]	『強制』儲存檔案後離開
[:] [q]	不儲存離開
[:] [q] [!]	『強制』不儲存離開

搜尋與取代

[/]	向下搜尋 (n 至下一個)
: s/[old]/[new]/g	將所有的 [old] 替換成 [new]
:1,30s/[old]/[new]/g	將『第一行』至『第三十行』的 [old] 替換成 [new]
:1,\$s/[old]/[new]/g	將『整個檔案』的 [old] 替換成 [new]
:1,\$s/^/ /g	將每行的開頭都往後移四個空白

檔案權限設定(Permission)

► 在 Linux 系統中，每一個檔案都具有四種存取權限：

1. 可讀取(r, Readable), 用數字 4 表示
2. 可寫入(w, writable), 用數字 2 表示
3. 可執行:(x, eXecute), 用數字 1 表示
4. 無權限(-), 用數字 0 表示

► Example:

```
ls -l .bash_profile
```

```
-rw-r--r-- 1 cnotred 191 Jan 4 13:11 .bash_profile
```

► r:readable, w:writable, x: executable

- ▶ 系統管理者依據使用者需求來設定檔案權限，若我們想檢視檔案權限可以使用 `$ ls -l` 來查看

The image shows a terminal window with the command `ls -al` executed. The output is as follows:

```
n100@N100: ~  
n100@N100:~$ ls -al  
total 220  
drwxr-xr-x 22 n100 n100 4096 2012-08-18 18:09 .  
drwxr-xr-x  3 root root 4096 2012-08-18 04:36 ..  
-rw-----  1 n100 n100  117 2012-08-18 18:12 .bash_history
```

Handwritten annotations with arrows point to various parts of the output:

- # of HardLinks**: Points to the number '22' in the first line.
- owner of file**: Points to 'n100' in the first line.
- Size in Bytes**: Points to '4096' in the first line.
- Directory or File Name**: Points to '.' in the first line.
- File type and Access Permissions**: Points to 'drwxr-xr-x' in the first line.
- Usergroup**: Points to 'n100 n100' in the first line.
- Date & Time**: Points to '2012-08-18 18:09' in the first line.

- ▶ 由 10 個字元組成，第一個字元表示檔案型態（- 為檔案，d 表示目錄，l 表示連結檔案）。字元 2、3、4 表示檔案擁有者的存取權限。字元 5、6、7 表示檔案擁有者所屬群組成員的存取權限。字元 8、9、10 表示其他使用者的存取權限
- ▶ 舉例來說 `-rwxrwxr--`，代表這是一格檔案，擁有者和群組都具備讀取、寫入和執行權限，其他使用者只擁有讀取權限

- ▶ 第二欄:檔案數量
- ▶ 第三欄:擁有者
- ▶ 第四欄:群組
- ▶ 第五欄:檔案大小
- ▶ 第六欄:檔案建立時間
- ▶ 第七欄:檔案名稱

- ▶ **chmod: 修改檔案權限**
 - ▶ 將權限設為 `rw-rw-r--`:
 - ▶ `$ chmod 664 filename`
 - ▶ 將檔案的使用者(user)和群組(group)加入執行權限
 - ▶ `$ chmod ug+x filename`
 - ▶ 對所有使用者(all)更改權限為writable
 - ▶ `chmod a+w filename`
 - ▶ 對其他使用者(others)移除執行(executable)權限
 - ▶ `chmod o-x filename`
 - ▶ 對使用者(users)增加執行(executable)權限
 - ▶ `chmod u+x`
 - ▶ `u: user (owner); g: group; o: others; a: all`

系統管理

- ▶ **sudo**: 使用最高權限 (superuser) 執行指令, 會要求輸入自己密碼, 使用上必須非常小心
 - ▶ `$ sudo apt install vim`
- ▶ **ps**: 顯示process資訊
 - ▶ `$ ps`
- ▶ **kill**: 根據 Process ID 指定要終止程式
 - ▶ `$ kill PID`
 - ▶ 立即強制執行:
 - ▶ `$ kill -9 PID`

套件管理

- ▶ `apt-get`: 套件管理工具(`-get`可省略)
- ▶ 更新套件資料庫列表:
 - ▶ `$ sudo apt-get update`
- ▶ 升級套件並下載安裝套件:
 - ▶ `$ sudo apt-get upgrade`
- ▶ 安裝套件:
 - ▶ `$ sudo apt-get install xxxx`
- ▶ 移除套件:
 - ▶ `$ sudo apt-get remove xxxx`

網際網路相關操作

- ▶ **ping**: 網路檢測工具, 透過發送 ICMP ECHO_REQUEST 的封包, 檢查自己與特定設備之間的網路是否暢通, 速度是否正常
 - ▶ 可輸入 hostname 或是 IP:
 - ▶ `$ ping google.com`
- ▶ **tracert**: 檢查從你的電腦到網路另一端的主機是走的什麼路徑
 - ▶ `$ traceroute google.com`
- ▶ **nslookup**: 查詢 DNS 回應是否正常
 - ▶ `$ nslookup google.com`

工作控制job control

- ▶ Linux中一個login Session只能有一個前景程式存在
- ▶ 讓某一程式進入背景執行為兩以上的程式同時執行的方法
- ▶ 讓程式進入背景的方法為
 - ▶ 在一般的指令後面加個 &
- ▶ 常用的指令
 - ▶ 中斷指令:Ctrl+c
 - ▶ 放入背景, 並暫停其指令:Ctrl+z
 - ▶ 讓編號1的工作在背景執行:bg %1
 - ▶ 讓編號1的工作在前景執行:fg %1

Example

- ▶ `vim hello.sh`
- ▶ 放入背景: `Ctrl + z`
- ▶ 跳回去vim編輯介面: `$fg`

history 的運用與設定

- 觀看歷史指令: `history`
- 執行上一個指令: `!!`
- 執行第n個指令: `!n`
- 搜尋以某字串string為開頭的指令並加以執行: `!string`
 - 搜尋時以event數字大到小
- 搜尋過去指令中有某個字串的指令: `!?string?`

Practice

```
bruce@bruce-VirtualBox: ~/lab1
File Edit View Search Terminal Help
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char*argv[]){
    if(argc!=3){
        printf("Please use command %s [integer1] [integer2]\n", argv[0]);
        exit(1);
    }
    printf("%d\n", atoi(argv[1])+atoi(argv[2]));

    return 0;
}
~
~
~
~
~
~
~
"add.c" 12L, 248C 12,1 All
```