

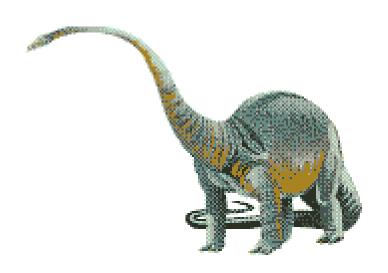


作業系統(Operating Systems)

Course 10: 磁碟管理 (Disk Management)

授課教師: 陳士杰

國立聯合大學 資訊管理學系



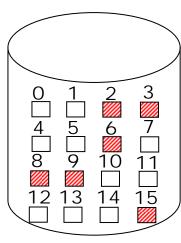


- Free Space Management (可用空間管理)
- Allocation Methods (檔案配置方式)
- Disk Access Time
- Disk Scheduling Algorithms
- RAID
- Solid State Disk



■ Free Space Management

●基本概念: Disk Allocation/Free Space單位是以"區塊" (Block) 為主。



- Allocation
- □ Free

- ●方法:
 - Bit Vector (Bit Map)
 - Link List
 - Combination
 - Counting





Bit Vector

- Def: 用一組Bits來代表Blocks配置與否,一個Bit對應一
 - 個Block。
 - Bit值 = 0,表示Free
 - Bit值 = 1,表示Allocate
- Allocation
- ☐ Free
- ●範例:如右圖,請列出其Bit Vector。
 - **0011001011000001**
- ●優點:
 - Simple
 - 易於找到連續可用空間(即:找到連續的"0")
- 缺點:
 - Bit Vector儲存佔用Memory空間,所以此法不適用於大型Disk (∵若Block有上百萬個,則Bit Vector會非常大, Memory可能會容納不下)



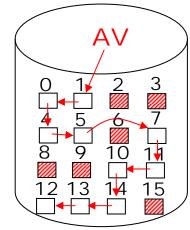
6 20 10



Link List

● Def: 利用Link List將Free Block串接,形成一個AV List。

- Allocation
- ☐ Free



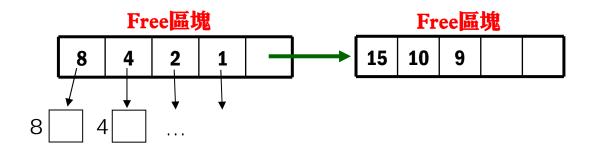
- ●優點:
 - ™ 加入/刪除Block容易
- ●缺點:
 - 型 效率不佳。因為在找尋可用區塊時,需從頭檢視此串列,I/O Time過長(∵檢視一個Block就須做一次I/O)





Combination (組合法)

取某個可用區塊,記錄其它可用區塊之編號。若此區塊不足以容納所有的Free Block編號,則再以其它區塊記錄,並用 Link 串連。

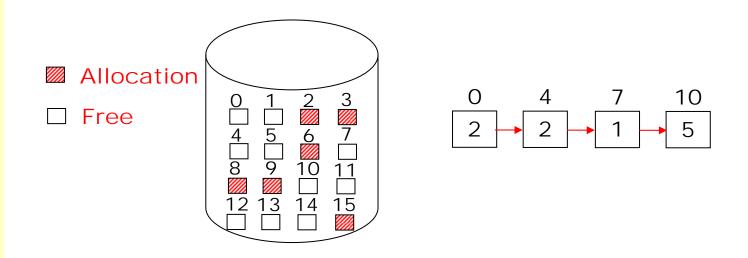


- ●優點:
 - □ 加入/刪除Block容易
 - ☑ 可迅速取得大量可用區塊
- 缺點:
 - 效率不佳,I/O Time過長





適用於連續區塊較多之情況。在一個Free Block中,記錄其後的連續可用區塊數目(含自已)。



● 如果連續區塊數目夠多,則Link List長度可大幅縮短。





■ Disk結構

- Track (磁軌)
- Sector (磁區)
- Cylinder (磁柱)
 - 不同面之相同Tracks形成之 集合
- Read/Write Head (讀寫頭)

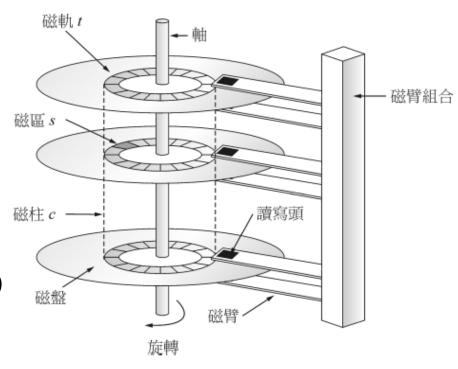


圖 12.1 移動磁頭的磁碟機制



Disk Capacity

若某硬碟有10個Disks,正反面皆可存Data,但最上、最下一面不存。每一個表面上有1024條Tracks,每個Track有256個Sectors。一個Sector可存512 Byte資料,則磁碟的容量為何?

Ans:

$$(10 \times 2 - 2) \times 1024 \times 256 \times 512 \text{ Bytes} = 18 \times 2^{10} \times 2^8 \times 2^9 \text{ Bytes}$$

= $18 \times 2^7 \text{ MB}$





Disk Access Time

- 由下列3個時間加總:
 - Seek Time
 - · 將Read/Write Head移到指定Track上方所花的時間
 - Latency (Rotation) Time
 - · 將欲Access的Sector轉到Read/Write Head下方所花的時間
 - Transfer Time
 - ·資料在Disk及Memory之間的傳輸時間
- 上述三者以Seek Time耗時最多
 - ∵是機械動作
 - 與 "Disk Scheduling Algorithm" (磁碟排班演算法)及 "Track service request arrival order" (Track 服務請求之到達順序)

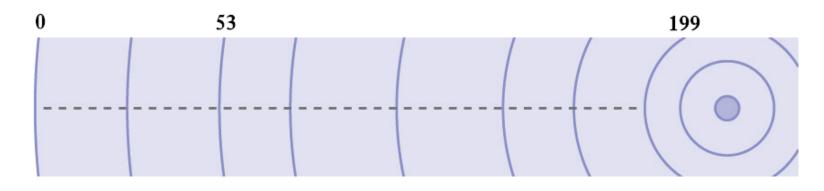




Disk Scheduling Algorithms

- FCFS
- SSTF
- SCAN
- C-SCAN
- LOOK
- C-LOOK
- 説明範例:
 - 目前的磁頭位於53軌。磁碟緩衝區內的磁軌讀、寫要求依序為:

98, 183, 37, 122, 14, 124, 65, 67







FCFS演算法 (first-come, first-serve)

- 先到達的Track Request優先服務
 - 範例:有下列的Track請求於佇列當中,採用FCFS,則Track移動總數為何?

佇列=98,183,37,122,14,124,65,67 讀寫頭自 53 啟始

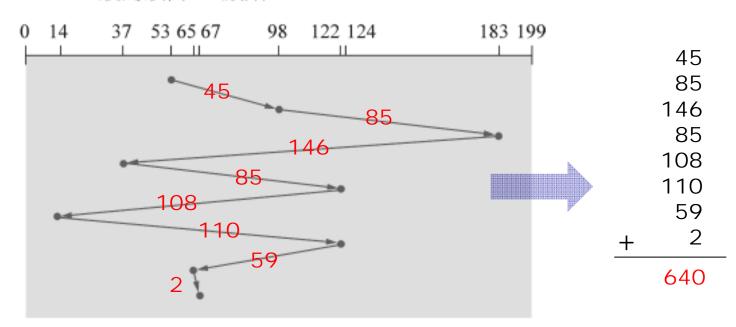




圖 12.4 先來先服務的磁碟排序



- Simple
- 公平的 (不會發生Starvation)
- 效果不佳 (Avg. Seek Time ↑)





SSTF演算法 (short-seek time first)

- 距離目前R/W Head最近的Track優先服務
 - 範例:有下列的Track請求於佇列當中,採用SSTF,則Track移動總數為何?

佇列=98,183,37,122,14,124,65,67 讀寫頭自 53 啓始

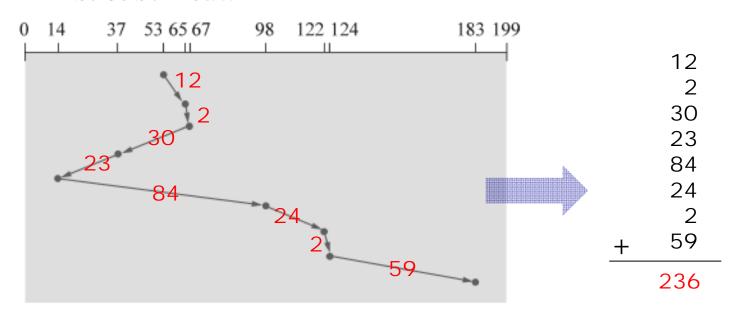


圖 12.5 最短尋找時間的優先服務之磁碟排序



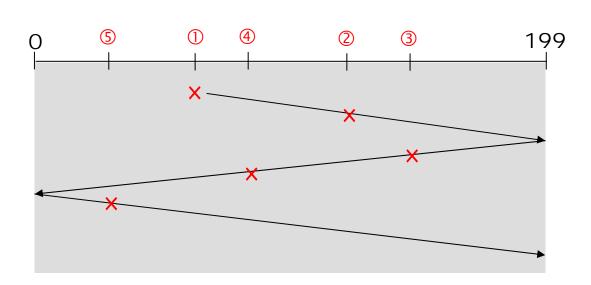


- 並非是最佳的('.'平均移動的Track總數量無法保証為最少)
- 不公平,可能會發生Starvation (若後續進來的Track請求皆是在讀寫頭附近的話,則距離讀寫頭最遠的Track請求可能會餓死)



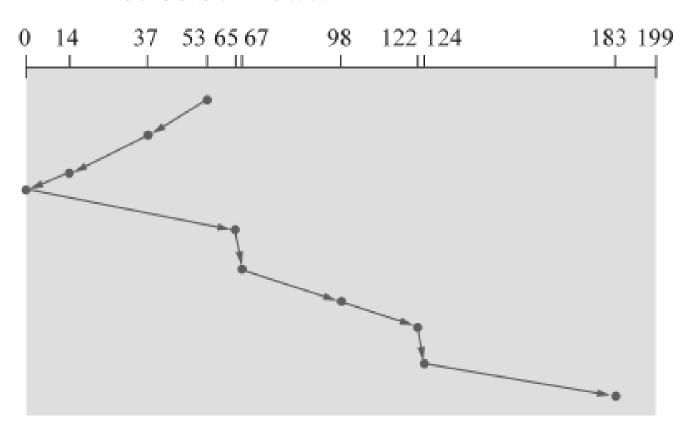


- 讀寫頭來回不停地掃描,若有Track請求,則服務。
 - ₩ 提供雙向服務
 - 遇到Track起頭與尾端才折返。
- 範例:



- 特點:
 - 因為要碰到起頭或尾端才會折返,會有一些不必要的磁軌移動產生
- 1





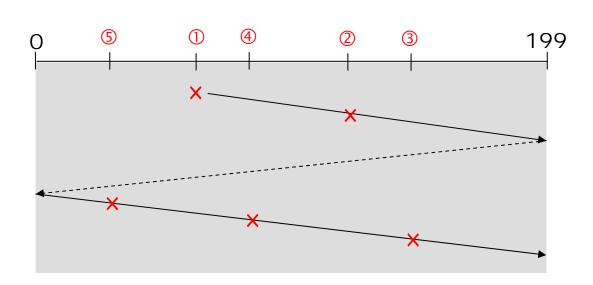






C-SCAN演算法

- 讀寫頭來回不停地掃描,若有Track請求,則服務。
 - 提供單向服務
 - 遇到Track起頭與尾端才折返。
- 範例:







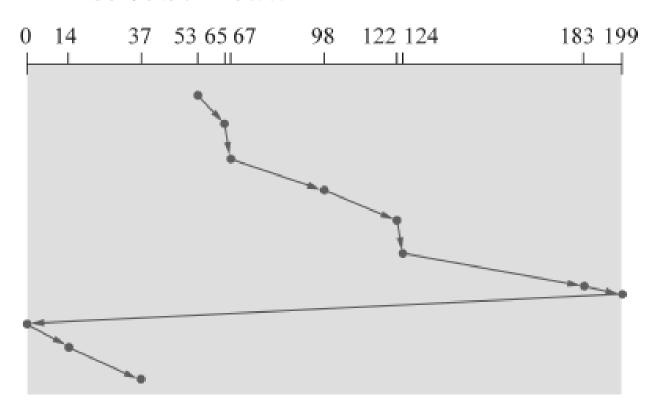
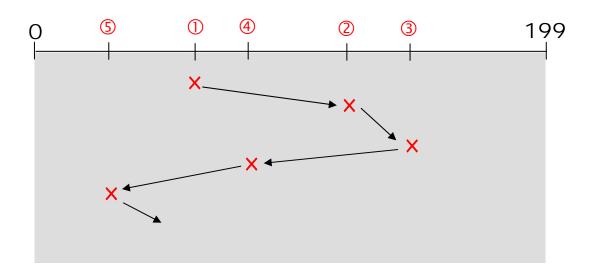


圖 12.7 C-SCAN 磁碟排序

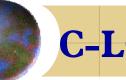




- 讀寫頭來回不停地掃描,若有Track請求,則服務。
 - ₩ 提供雙向服務
 - 處理完某方向的最後一個Track請求,即折返服務,不需遇到 Track起頭與尾端才折返。
- 範例:

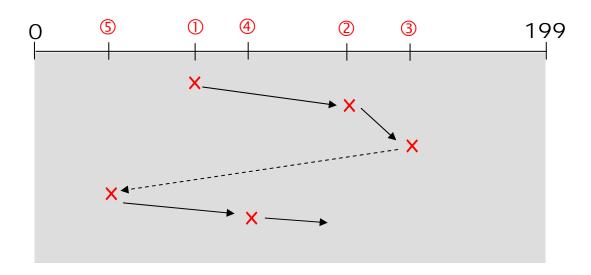






C-LOOK演算法

- 讀寫頭來回不停地掃描,若有Track請求,則服務。
 - ₩ 提供單向服務
 - 處理完某方向的最後一個Track請求,即折返服務,不需遇到 Track起頭與尾端才折返。
- 範例:







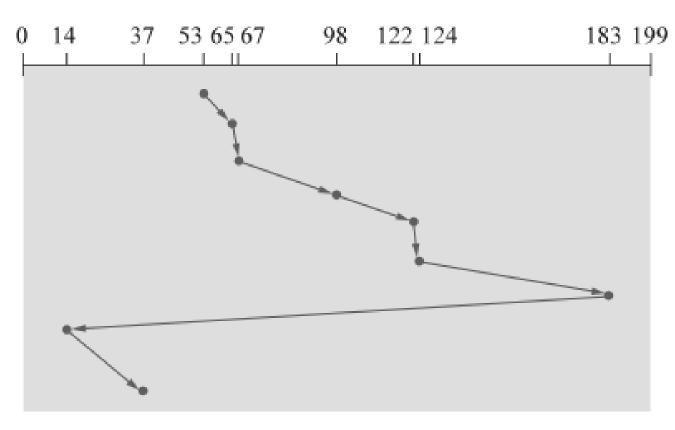


圖 12.8 C-LOOK 磁碟排序



22



■ Allocation Methods (檔案配置方式)

- Disk配置給File所需之區塊的方式有四:
 - Contiguous Allocation
 - Linked Allocation
 - FAT
 - Index Allocation



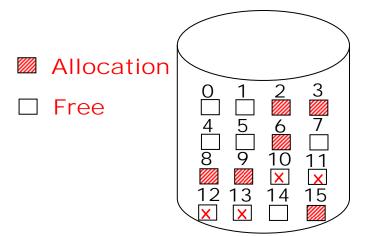


Contiguous Allocation (連續性配置)

- 若File大小為n個Blocks,則OS必須在Disk上找到大於等於n個連續Free Blocks才能配置。
- Physical Directory的記錄方式:

File Name, Size, Starting Block #

- Physical Directory是給OS看的
- Logical Directory是給User看的(如:dir或是檔案總管中的資訊)
- 範例:若File 1要求4個Blocks大小的空間。



File Name	Size	Starting Block #
File 1	4	10



● 優點:

- 可支援Sequential Access及Random Access
- Seek Time 通常較短 (: File所佔的連續區塊大都會落在同一個 Track或鄰近的Track上)

缺點:

- 易遭受External Fragmentation Problem
 - ·以前例來說,若有一個File要7個Block, Disk中可用的Block有10個, 卻因為空間不連續而無法配置給該File使用。
- 檔案大小無法任意增/刪
 - · 有些File是動態的,如:交易檔。但連續配置時,若某一已被配置之動態檔案,其位置前後的空間都已被配置給其它檔案,此時就無法動態增加該檔案的資料了。
- 建檔時須事先宣告大小





Link Allocation (連結配置)

- 若File 大小為n個區塊,則OS必須在Disk上找到 ≥ n個可用區塊即可配置(不一定要連續)。各配置區塊之間以Link來做串連。
- Physical Directory的記錄方式:

File Name, Start Block #, End Block #

● 範例: 若File 2要求5個Blocks大小的空間。

File Name	Star Block #	End Block #
File 2	1	14

⇒Block內會有幾個byte是存放連結 的。最後一個block有無存null皆無 妨!!∵有End Block #的記錄在。



● 優點:

- ≅ 沒有External Fragmentation Problem
- File可任意增/刪空間
- 建檔時,無需事先宣告大小

缺點:

- 不支援Random Access
 - ·要讀某個File的某個區塊時,需從頭找起。二只支援Sequential Access。
- Seek Time比Contiguous配置方式長
 - · :: 非連續性的Blocks可能散落在許多不同的Track上。
- Reliability差
 - ·若Link斷裂,則Data block lost。





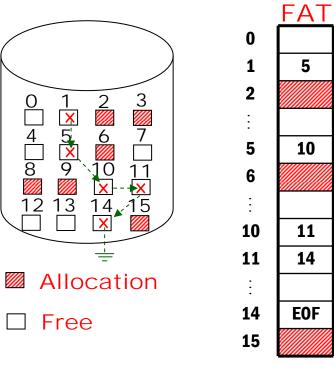
FAT (File Allocation Table)

- 為Link Allocation的變形。將所有配置區塊之間的Link及 Free Block資訊全部記錄在一個Table中,此Table稱之為 FAT。
 - 若Disk有n個Blocks,則FAT有n個項目(不論是Free的還是使用中的Block皆需記錄)
 - FAT中的項目資訊:
 - ·空白:表示Free Block
 - · 某Block #:有Link指向此Block,即使用中的Block
 - EOF: End of File
 - MS-DOS及OS/2採用
- Physical Directory的記錄方式:

File Name, FAT Index







File Name	FAT Index
File 3	1

⇒FAT Index = 1表示由第1個Block 開始。

- Block可完全都存放Data,不像Link Allocation會有幾個 Byte用來存放連結。
- 優缺點和Link Allocation差不多



29

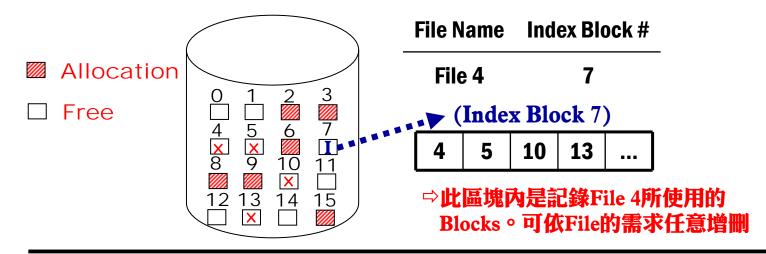


Index Allocation (索引式配置)

- 每個File皆會多配置一些額外的Blocks作為Index Blocks,
 內含各配置的Data Block編號。採非連續性配置方式。
- Physical Directory的記錄方式:

File Name, Index Block #

- 範例:若File 4要求4個Blocks大小的空間。
 - 會配到5個Blocks (1個Index Block + 4個Free Blocks)
 - 假設1個Index Block可存10個Block #







- ※ 沒有External Fragmentation Problem
- 支援Random Access及Sequential Access
- File可任意增/刪空間
- 🛮 建檔時,無需事先宣告大小

● 缺點:

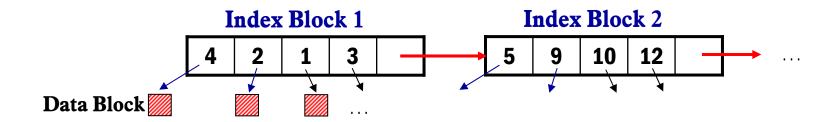
- Index Blocks佔用額外空間
 - · 索引區塊佔用了Free Block的空間
- Index Block大小設計亦是難題
 - ·太小,不夠存放一個file的所有Block之編號;太大,又形成浪費



解決Index Block不夠大的三個方法

● 【方法1】

- 利用多個(≥1)Index Blocks儲存,且Index Block之間以Link方式串連。
- 範例:

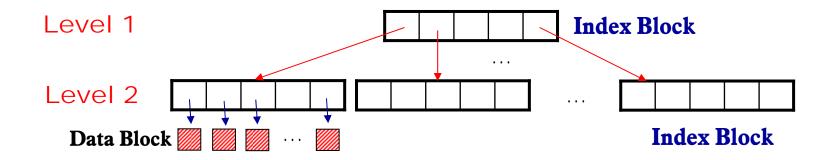


- 適用於小檔案
- □ Link斷裂, Data lost



● 【方法2】

- Multilevel Index (多層索引)
- 範例: 2 level index



📱 僅適用於大型檔案,小檔案不適合

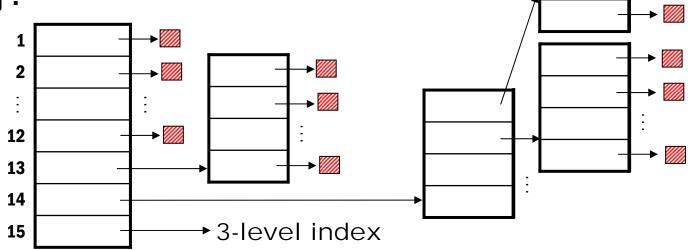
· ::一旦系統的索引決定是2 level的,則不論檔案的大小,所使用的都是2 level的index!!當存到小檔案時,搞不好index的大小會大於檔案。



● 【方法3】

- 混合法,即 Unix 之 i-Node Structure。有15個項目 (Pointer),可逐級累加使用。
 - ・1~12號指標:儲存Data Block # (檔案的Block需求≤12個時適用)
 - 13號指標:指向Single-level Index
 - 14號指標:指向Two-level Index
 - 15號指標:指向Three-level Index

■ 範例:







■磁碟陣列(RAID)

- Why RAID?
 - ■電腦系統運作中,就以Disk效率最差(∵機械式運作)
 - ₩安全性、可靠性議題日益重要
- RAID是由多顆磁碟機組成一個陣列(邏輯磁碟; Logical Disk),將資料以切割(striping)的方式,同時對不同的磁碟做讀寫的動作。
 - ■1個Byte的資料原本存放於1個磁碟上。若切割成8個bit分散存放於 8個Disk上,且一次存取1個Byte資料,則可獲得8倍的傳輸速率。
 - ■採"平行I/O"以提升效率。
 - ■安全性、可靠性提升
 - ■由OS進行資料的Striping與分散存取,故User僅看到一個大且速度快的邏輯磁碟空間(Logical Disk/Logical Device)。

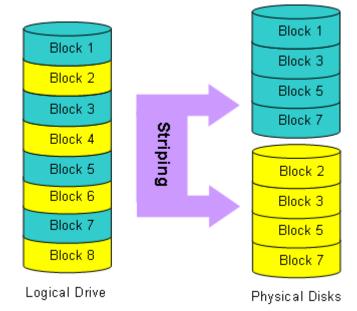


- 依所使用的技術不同,定義出多種具有不同方式與效能的 RAID 代號,簡稱RAID Level。
 - RAID 0, RAID 1, RAID 0+1, RAID 1+0, RAID 2, RAID 3, RAID 4, RAID 5, RAID 6, ...
 - Level高低不代表系統效能或可靠度的高低,端視管理員的需求。



• RAID 0 :

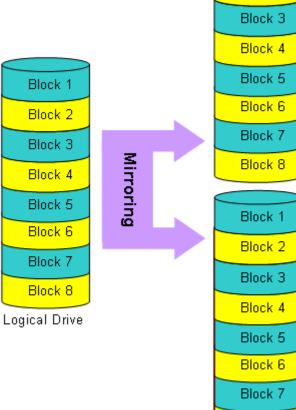
- 又稱Striping RAID
- ■最少需要2顆磁碟
- ■將資料切割存放到多部硬碟中,且不會重複存放。
- 以<mark>效能</mark>導向為主,可平行讀寫,適用於具有高效能需求的系統。
 - ·理論上,硬碟越多效能就等於[單一 硬碟效能]×[硬碟數]。事實上,受 限於匯流排I/O瓶頸及其它硬體因 素,RAID 效能會隨之遞減。
 - · 所以,兩個硬碟的RAID 0最能感受 到效能的提升。
- □ 缺點是完全沒有容錯能力,只要有 一個磁碟故障,就會導致所有資料 損毀,無法挽回。





• **RAID** 1:

- 又稱Mirror RAID
- 最少需要2個硬碟
- □ 有資料寫入時,直接同時寫入兩個硬 碟,故其內部資料是完全一樣的。
- 其中一個硬碟可做為備份。當有一個 硬碟故障時,另一個硬碟可讓系統正 常運作,故可靠性高。
- Read效能佳,但Write效能差。
- 優點:系統可靠度很高,實作容易。
- 缺點:效率最差,且成本高。



Physical Disks

Block 8

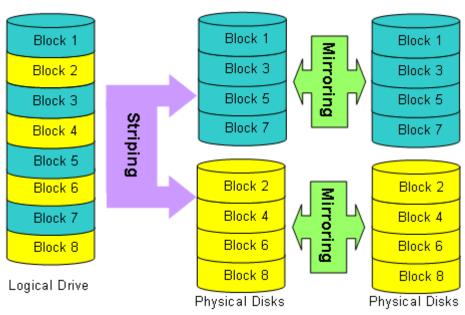
Block 1

Block 2



■ RAID 0+1 (RAID 01): Striping + Mirror

- 點 結合了RAID 0與RAID 1兩種模式。在這樣的架構下,至少要4顆磁碟,且硬碟總數需為偶數。
- 由2個硬碟先進行RAID 0的Striping,再由RAID 1的Mirror做複製。
- 優點:Speed與安全性 兼顧。
- 缺點:一次最少要用上 4顆硬碟,而且也要浪 費一半的總容量,成本 很高。





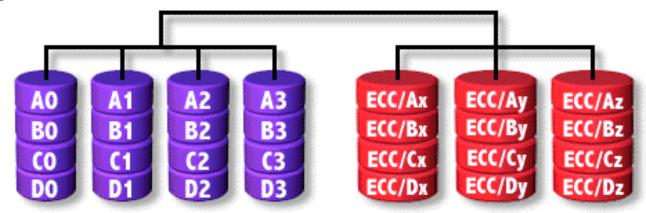


- 點 結合了RAID 0與RAID 1兩種模式。在這樣的架構下,至少要4顆以上的磁碟,且硬碟總數需為偶數。
- 由2個硬碟先進行RAID 1的Mirror,再由RAID 0的Striping做複製。
- 可靠性較RAID 01高,但Speed較差。



RAID 2

- 點 為RAID 0的改良版,以<mark>漢明碼(Hamming Code</mark>)的方式進行資料編碼。
- 不同的位元儲存在不同的硬碟中。例如:4個Bits的資料需要7部硬碟來儲存,4部存資料,3部存Hamming Code。
- □ 因為加入了錯誤修正碼(ECC, Error Correction Code),具錯誤檢查與更正能力。所以當一部硬碟壞掉時,可由其它硬碟的內容來更正。



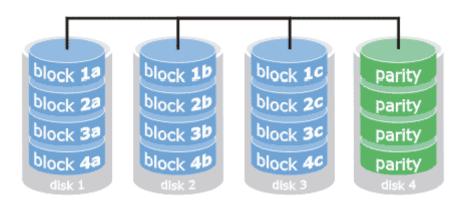
AO to A3=Word A; BO to B3 = Word B; CO to C3=Word C; DO to D3 = Word D ECC/Ax to Az=Word A ECC; ECC/Bx to Bz = Word B ECC; ECC/Cx to Cz=Word C ECC; ECC/Dx to Dz = Word D ECC



- RAID 3: Parallel with Parity (平行同位元檢查)
 - □ 資料切割以bit為單位
 - 以同位元檢查為容錯機制
 - 所有的同位元資料皆存放於同一個硬碟(Parity Disk)上
 - □ 允許同時間有一個disk損壞,且仍可回復
 - □ 問題:
 - · Parity Disk會成為效能上的瓶頸 (RAID 4亦有此問題)。

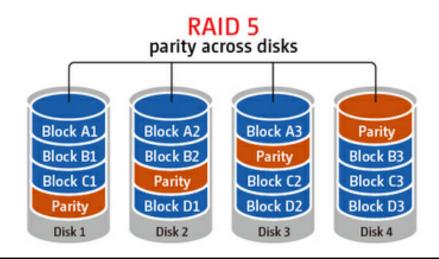
RAID 3

parity on separate disk





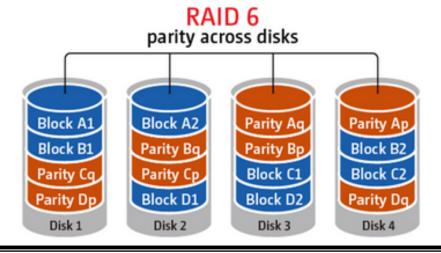
- RAID 4: Parallel with Parity (平行同位元檢查)
 - 點 跟RAID 3 相似,不同之處為資料切割是以Block 為單位。
 - 效率較RAID 3佳。
- **RAID** 5:
 - 與RAID 4相似,資料切割是以Block為單位,但是RAID 5把同位 元分散儲存於陣列中的每顆磁碟內。





RAID 6:

- RAID 6 儲存2 份同位元資料(Double Parity),讓系統即使損壞兩顆硬碟仍能進行資料回復。
- RAID 6的Write效能較RAID 5差,是因為資料寫入時的同位元校 正機制必須執行兩次,即:計算同位元值和驗證資料正確性所 花費的時間較多。
- ≌ 資料重建時,回復的時間也較RAID 5要多一倍。







■固態硬碟(SSD)

- Solid State Disk
 - ™相較於傳統旋轉式H.D.,為較進代的儲存裝置。
 - ™採用SATA 3.0傳輸規格。





•與傳統硬碟的比較:

	固態硬碟	傳統硬碟		
優點	電子式資料讀寫 ⇒抗震性佳 ⇒適用於行動裝置	機械式資料讀寫 ⇒抗震性差 ⇒適用於桌機或NB	缺	
	Random Access ⇒Speed快	Header移動 ⇒Speed慢	點	
缺點	成本高	成本低	優	
	容量小	容量大	點	



補充





■錯誤碼的偵測與更正

- Parity Check (同位元檢查) -具偵測錯誤能力
- Hamming Code (漢明碼) -具偵測錯誤與更正錯誤能力





Parity Check (同位元檢查)

可分為:

- 偶同位 (Even Parity): 訊息加上Parity Check Bit所形成的傳輸碼中,位元值為 "1" 之個數必須是偶數,才算正確。
- □ 奇同位 (Odd Parity): 訊息加上Parity Check Bit所形成的傳輸碼中,位元值為 "1" 之個數必須是奇數,才算正確。

● 例:

- 求下列傳送訊息所需補上的Parity check bit之值(採Even Parity):
 - · 訊息: 1101000 ⇒ Parity Check Bit = 1
 - · 訊息: 0110011 ⇒ Parity Check Bit = 0
- 下列何者傳輸碼不符合Odd Parity?
 - · 1111111 (\sqrt{)
 - · 1011100 (×)
 - · 10000100 (×)





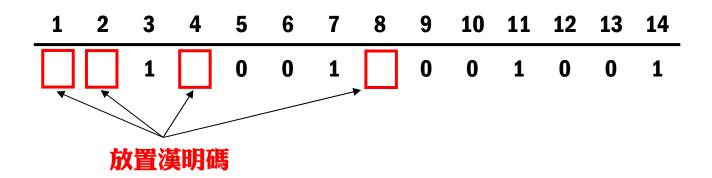
- Simple, Easy Implementation •
- 用於ASCII,RAM資料傳輸檢視,RAID (磁碟冗餘陣列: RAID-3, RAID-4, RAID-5)等。
- 器 若有<mark>偶數個bits</mark>同時出錯,則收方無法偵測出Error,會誤判傳輸資料為正確。
- 不具更正錯誤能力。





Hamming Code (漢明碼)

- 傳輸碼 = 傳送訊息 + Hamming Code Bits (H.C. Bit)
 - 規定:H.C.Bit一定要出現在2的冪次方位置 (2⁰, 2¹, 2², 2³, ...)
 - 例:訊息為 1001001001,則傳輸碼:



● H.C.Bit數目:

Solution $2^k \ge n + k + 1$ (n: 訊息長度; k: H.C.Bit數, 取最小的k)

・例:n = 10 ⇒ $2^k \ge 10 + k + 1$,則最小的k = 4。



●如何決定H.C.Bit的值?

■ 作法:

①填入H.C.Bit的位置碼 1 © 中立値中立値表③各位置的二進制■ □ □ □ □ □

④作偶同位

■ 例:訊息為 1001001001,

則傳輸碼為:

10100010001001

	8	4	2	1	
3	0	O	1	1	
7	0	1	1	1	作偶同位
11	1	Ο	1	1	
14	1	1	1	0	Ļ
	0	0	0	1	_



- 收方收到傳輸碼,如何判定有無錯誤?若錯誤,是哪個 Bit有錯?
 - 作法:



- 零:傳輸碼正確

·非零:傳輸碼有錯,此值表示錯誤的位置



■ 例: 收方收到 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1 之H.C.Bit傳輸碼,請問有無error? 若有,請問錯第幾個?

	8	4	2	1	
1	0	О	0	1	
3	0	O	1	1	
7	0	1	1	1	作偶同位
9	1	0	0	1	
11	1	0	1	1	
14	1	1	1	0	
	轉成	O 10 進位	O	1 [-

- ·正確的H.C.Bit傳輸碼應為:10100010011001
- ·正確的傳送訊息為:100100101 (把H.C.Bit拿掉)

