CH9、Disk

硬碟的介紹

目錄：

Disk System 組成

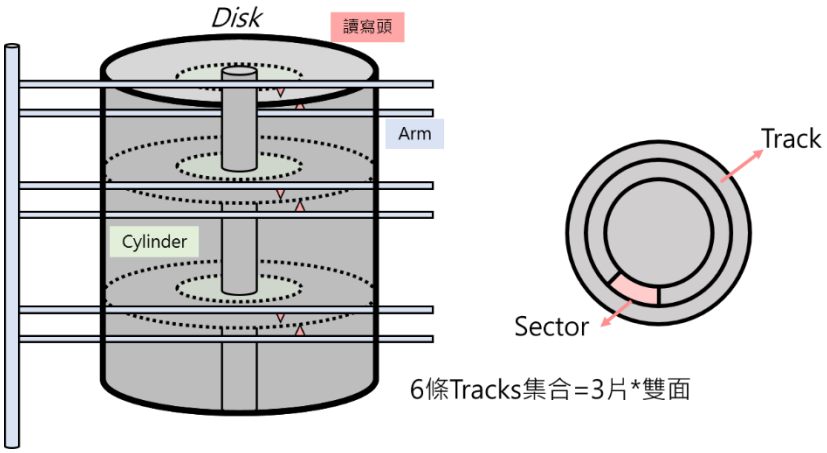
一、Disk System 由多片 Disks 組成

每片 Disk 通常雙面(double sided)可存 Data

每一面(Surface)劃分為多個同心圓軌道，叫磁軌(Track)

每條 Track 由多個 Sector(磁區)組成

不同面之相同 Track Number 組成之 Tracks 集合，叫作 Cylinder(磁柱)



例：

Disk System 有 10 片 Disks

每片皆雙面可存

每面有 2048 條 Tracks

每條 Track 有 4096 個 Sectors

每個 Sector 可存 16KB data

求 Disk System size ?

*10*2*2048*4096*16KB = 20*2^{11} * 2^{12} * 2^{14} bytes = 10*2^{38} bytes = 2.5*2^{40} bytes = 2.5 TB*

Disk Access Time 組成

一、Disk Access Time 由下列 3 個時間加總而來

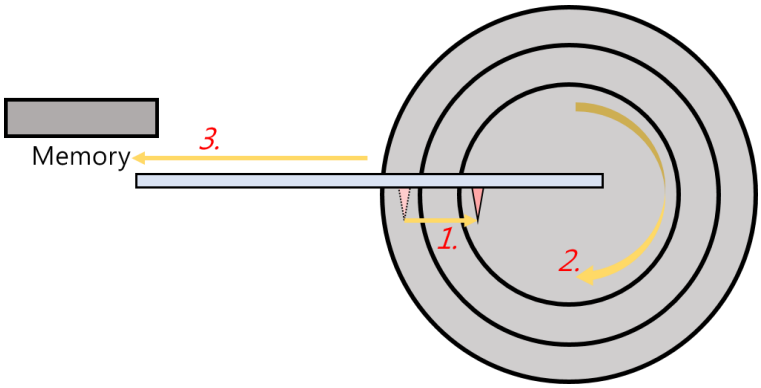
1. Seek Time
2. Latency Time (or Rotation Time)
3. Transfer Time

分述如下：

1. Seek Time：將 Head(磁頭)移到欲存取之 Track 上方所花的時間
2. Latency Time (or Rotation Time)：將欲存取之 Sector 轉到 Head 下方所花的時間
3. Transfer Time：Data 在 Disk 與 Memory 之間的傳輸時間，與傳輸量成正比

此外，上述 3 者通常以 Seek Time 佔比較大*(Seek > Transfer > Rotate)*

(Seek Time + Rotational Latency 稱為定位時間 Positioning Time)



二、計算

例 1：Disk 轉速 7200rpm，求 Average Latency (Rotation) Time？

rpm = Rotations per minutes
因此 7200 rpm = 1 分鐘可轉 7200 圈 => 1 秒可轉 7200/60 = 120
也等於轉一圈花 1/120 秒
*因此：平均 Latency Time = 1/2 * 1/120 秒 = 1/240 秒*

例 2：Disk system 有 3 片 Disks、雙面可存、每面有 1024 條 Tracks、每條 Track 有 40%個 Sectors、每個 Sector 可存 32KB、轉速 6000rpm，求 Transfer rate(即每秒可傳輸多大量 Data)？

6000rpm = 1 秒可轉 6000/60 = 100 圈
*每轉一圈可傳輸 1 個 cylinder 容量(多面時)，因此一條 Track 容量 = 40% * 32KB*
*因此一條 Cylinder 容量 = 6 條 Track * 40% * 32 KB = 6*128MB*
*Transfer Rate = 100*6*128MB/sec = 600*128MB/sec*

例 3：承上題，若 Disk 平均 Seek Time=10ms，今有一個 File 大小=2MB，欲 Read this file，要花多少 IO Time

IO Time = Seek Time + Latency Time + Transfer Time
*= 10ms + 1/2*1/100 秒 + 2MB/(600*128MB)sec*
*= 10ms + 5ms + 10/(3*128)ms*

Disk Free Space Management(可用空間的管理)

有四種方式

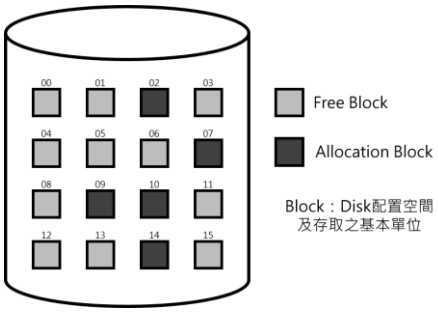
1. Bit Vector (或 Bit Map)
2. Link List
3. Grouping
4. Counting

3、4 皆為 2 之變型

Bit Vector (Bit Map)位元向量/地圖

一、Def：每一個 Block 皆用 1 個 Bit 表示 Free 與否：0 表 Free、1 表 Allocated
Disk 有 n 個 Blocks，則 Bit Vector 大小 = n bits

二、例：



0010 0001 0110 0010

三、優點：

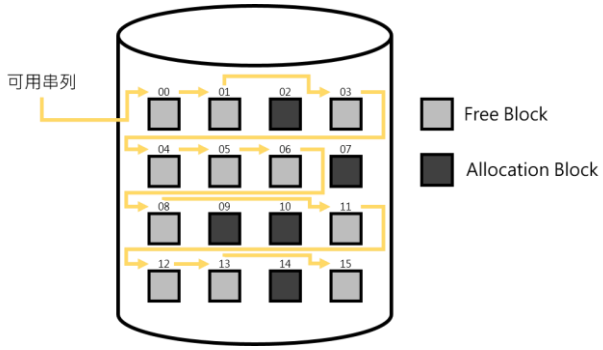
1. 簡單易實施
2. 容易找到連續的 Free Blocks(即找連續的 0)

缺點：

小型 Disk 適用，但大型 Disk 不適用，因為 Blocks 數目龐大，造成 Bit Vector size 很大，很佔 Memory Space，甚至被迫置於 Disk 中

Linked List

一、Def：OS 直接在 Disk 上，將這些 Free Blocks 以 Linking 的方發串接，進行管理



二、優點：

1. 大型 Disk 適用
2. 插入/刪除 Free Block 方便

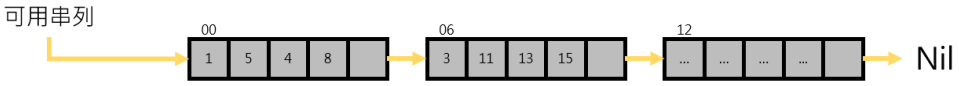
三、缺點：

1. 不夠迅速來找大量可用 Block，因為在 Disk 上讀取 Link Information 耗時(IO)。*Note：用 Grouping 法改善*
2. 不容易找到連續的 Free Blocks。*Note：用 Counting 法改善*

Grouping 法

一、Def：是 Linked List 法之變型，在 Free Block 內除了記錄 Link Information 之外，另額外記錄其他 Free Blocks 之 Number(Address)

二、例：Assume One Block 可記 5 個欄位

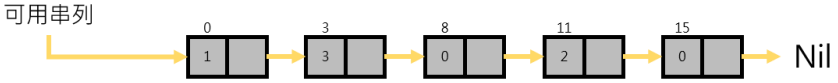


三、優點：可迅速找到大量的 Free Blocks

Counting 法

一、Def：利用連續性配置及歸還之特性，改變 Linked List 記錄方式，Free Block 內除了記錄 Linking Information 以外，另外記錄在此 Free Block 之後的連續 Free Blocks 之個數

二、例



三、優點：適用於連續性配置，方便找到『連續的』 Free Blocks，且若連續的 Free Blocks 很多，Linked List 長度也可大幅縮短

File Allocation Methods

1. Contiguous Allocation(連續性)
2. Linked Allocation(鏈結式)
   (變型)FAT Method
3. Index Allocation(索引式)：例：UNIX 的 I-Node Structure

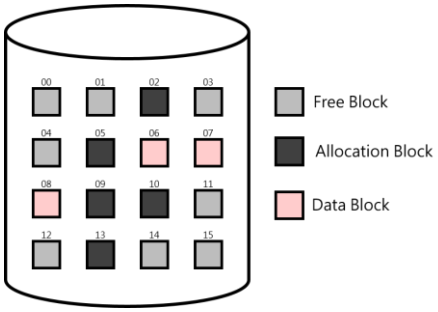
Contiguous Allocation

一、Def：若 File 大小=n 個 Blocks，則 OS 必須在 Disk 中找到 n 個"連續的"Free Blocks，才能配置給它。此外，OS 在 Physical Directory 會記錄下列資訊：

File Name	Start Block Number	Size(區塊數)

二、例：

三、



若 File 1 大小=3 個 Blocks，則 OS 配置 6、7、8 號 Blocks 給它，且 Physical Directory 記錄如下：

File Name	Start Block Number	Size
File 1	6	3

四、優點：

1. 平均 Seek Time 較小(因為連續的 Blocks 大都落在同一條 Track 或鄰近 Track 上)
2. 可支持 Random(Direct) Access(任意存取 i^{th} Block of the file)及 Sequential Access => 即 i^{th} Block Number = Start Number + i-1
3. 可靠度較高(相對於 Linked Allocation)
4. 循序存取速度較快(相對於 Linked Allocation)

缺點：

1. 會有外部碎裂。*(Note：Disk 用 Repack(磁碟重組)方式來解決，類似 Memory 的 Compaction，兩者差別：1.速度快慢、2.位置在 Memory 或 Disk)*

> **Note：所有 File 配置方法皆會有內部碎裂。例：**
> *Block size = 10KB、File 大小=44KB、所以配 5 個 Blocks => 內部碎裂 = 5*10-44 = 6KB*

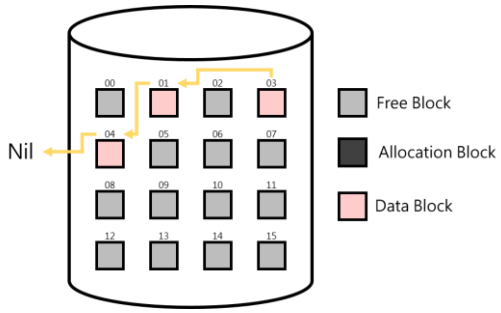
2. File 大小不易動態擴充
3. 建檔之前需事先宣告大小

Linked Allocation

一、Def：若 File 大小= n 個 Blocks，則 OS 只需在 Disk 中找到 n 個 Free Blocks(不需連續)即可配置，且 Allocated Blocks 之間以 Link 方式串連，另外 OS 的 Physical Directory 記錄如下：

File Name	Index Block Number
File 3	15

二、例：



若 File 2 大小=3 個 Blocks，則 OS 可配置：3、1、4 號 Blocks 給它，且 Physical Directory 記錄如下：

File Name	Start Block Number	Size
File 2	3	4

三、優缺點：與 Contiguous 相反

優點：

1. 沒有外部碎裂
2. File 大小容易動態擴充
3. 建立 File 之前，無需事先宣告大小

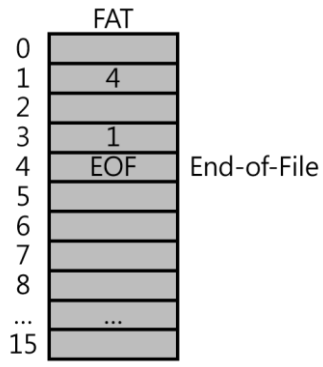
缺點：

1. Seek Time 較長*(因為不連續的 Blocks 可能散落在許多不同 Track 上)*
2. 不支援 Random Access
3. 可靠度差*(因為萬一 Link 斷裂，Data Lost)*
4. 循序存取速度慢*(因為要在 Disk 上讀取 Link 資訊，才知道下一個 Block 為何)*

FAT(File Allocation Table)方法

一、Def：是"Linked Allocation"之變型，主要差異：Allocated Blocks 之間的 Linking Information 存於 OS Memory Area 中的一個表格，叫 FAT，而非存於 Disk 中

二、例：



Physical Directory

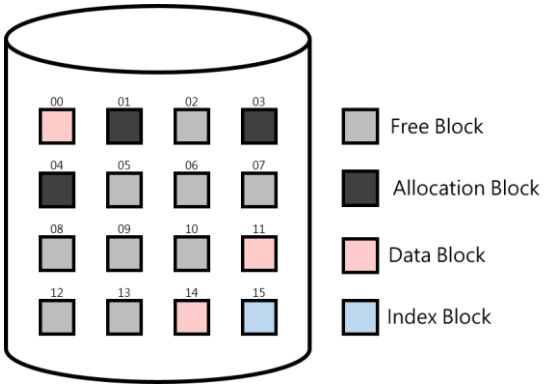
File Name	FAT entry
File 2	e

三、優點：想要加速 Random Access in the Linked Allocation，因為可以在 Memory 中的 FAT 迅速找到第 i^{th} Block 之 Number，然後再到 Disk Access i^{th} Block 即可，不用在 Disk 中 traverse Linking Information

Index Allocation(Linux)

一、Def：若 File 大小=n 個 Blocks，則 OS 除了配置 n 個 Blocks(無需連續)，另需額外配置 Index Block，儲存所有 Data Blocks 之 Number(Address)

二、例：



若 File 3 大小=3 個 Blocks，則 OS 可配置：11、0、14 號 Blocks 給它存放 Data，另外配置 15 號 Block 作為 Index Block，而 Index Block 內容為：

11	0	14	-	-

三、優點：

1. 無外部碎裂
2. 支援 Random Access 及 Sequential Access
3. File 大小容易動態擴充
4. 建立 File 之前，無需事先宣告大小

缺點：

1. Index Block 佔用額外空間
2. Linking Space 浪費(overhead)比 Linked Allocation 大許多
3. 若 File 很大，則單一個 Index Block 可能無法容納(保存)File 的所有 Data Block Number，此問題需被解決

解決單一個 Index Block 不夠存放所有 Data Blocks Number 之問題

方法有三：

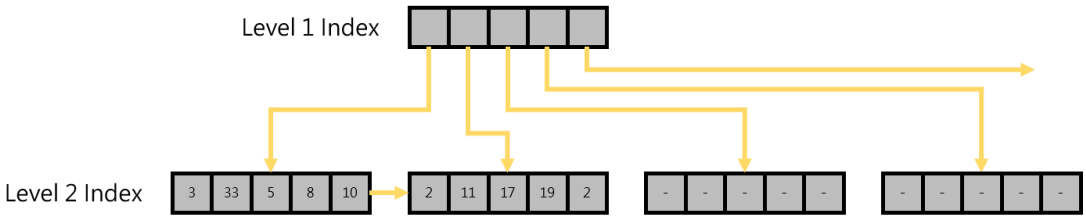
[法一]：使用多個 Index Blocks，且彼此以 Link 方式串連

例：(Assume 一個 Index Block 可存 5 個 Number)

缺點：Random Access of i^{th} Block 之平均 IO 次數大幅增加

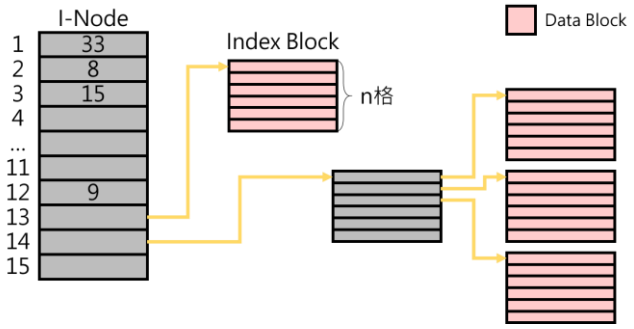
[法二]：使用階層式 Index Structure

例：2-level Index Structure



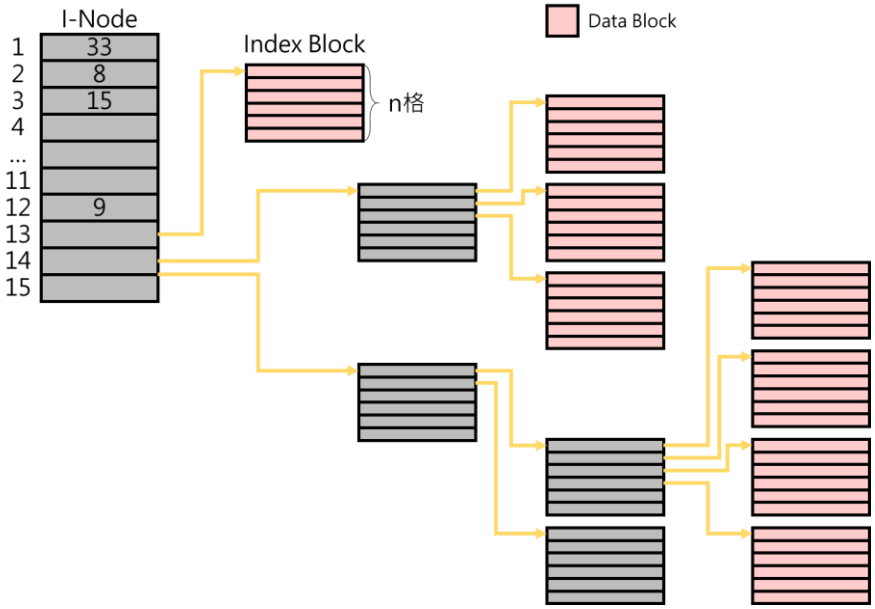
優點：Random Access of i^{th} Block 之 IO 次數是固定值，例：2 層=3 次 IO

缺點：對小型檔案極不合適，因為 Index Blocks 太佔空間，甚備多於 Data Blocks 數目



[法三]：混合法，即 UNIX 的 I-Node 結構
例：I-Node 有 15 格(entry)
其中：

1. 第 1~12 格：直接記 Data Blocks Number
2. 第 13 格：pointer to Single-Level Index
3. 第 14 格：pointer to Double-Level Index
4. 第 15 格：pointer to Triple-Level Index



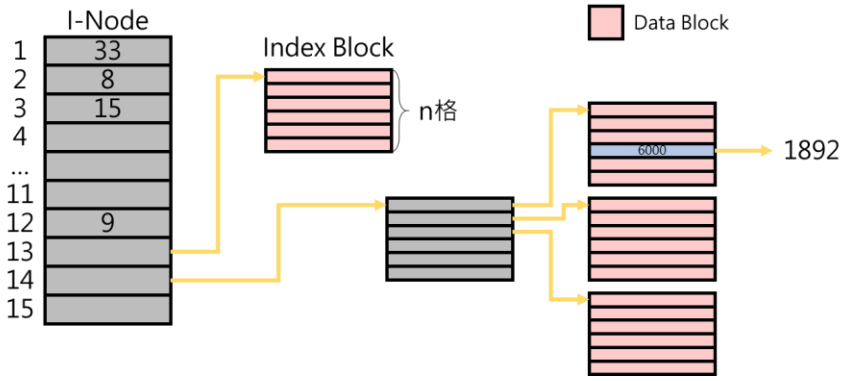
MAX File Size = $(12 + n + n^2 + n^3)$個 Data Blocks
優點：小型、大型 File 皆合適

例 1：
I-Node 之定義同先前所述(15 entry)
Block size=16KB
Block Number(Address)佔 4
求 MAX File size 多大？

1 個 index Block 可存：16KB/4bytes = 212 個 Data Block Number
所以 MAX File size = $(12 + 2^{12} + (2^{12})^2 + (2^{12})^3)$ 個 Blocks
= $(12 + 2^{12} + (2^{12})^2 + (2^{12})^3) * 16KB = 2^{36} * 16KB = 2^{50}$ Bytes = 1 PB

例 2：承上，若 File 大小=8000 個 Blocks，假設 I-Node 已在 Memory 中，則要存取此 File 的第 6000 個 Data Block 需要幾次 IO？

(6000-12=5988)
(5988-4096=1892)
⇨ *6000+3=6003 次 IO*



[補充]：

	Contiguous	Linked	Indexed
外部碎裂	有	無	無
內部碎裂	有	有	有
Sequential Access	有	有	有
Direct Access	支援	不支援	支援
指標浪費	無	中	嚴重
Seek Time	快	慢	慢
Index Blocks 額外空間損耗	無	無	嚴重

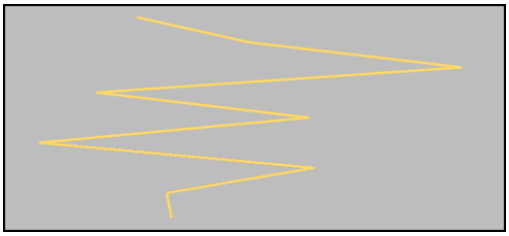
Disk Scheduling Algorithm

1. FCFS
2. SSTF
3. SCAN
4. C-SCAN(C：Circular)
5. Look
6. C-Look

FCFS(First-Come-First-Service)

一、Def：最早到達的 Track Request 優先服務

二、例：Disk 有 200 軌，編號 0~199，Head 目前停在第 53 軌，剛剛服務完第 40 軌，現在 Disk Queue 中有下列 Track Request 依序為：98, 183, 37, 122, 14, 124, 65, 67，求 Tracks 移動總數？



(183-53)+(183-37)+(122-37)+(122-14)+(124-14)+(124-65)+(67-65) = 640 軌

三、分析：

1. 排班效果不是很好，Track 移動數較多，Seek Time 較長*(但 SSD 多採此法)*

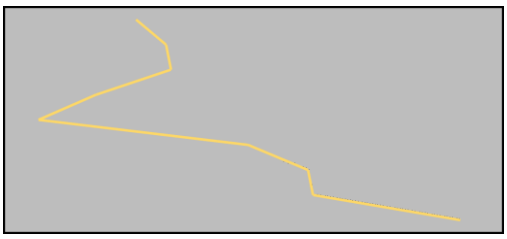
   Note：Disk Scheduling 法則：既無最佳、也無最佳

2. 公平，No Starvation

SSTF(Shortest Seek-Time First)

一、Def：距離 Head 目前位置最近的 Track Request，優先服務

二、例：

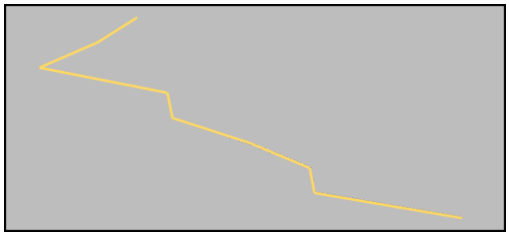


(67-53)+(67-14)+(183-14) = 230 軌

三、分析：

1. 排班效果不錯，Track 移動數少，Seek Time 小，但並非是 Optimal

   同樣例子，但是 FCFS：

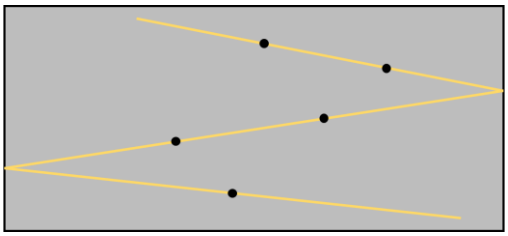
   

   (53-28)+(183-14) = 236-28 < 236，比 SSTF 好

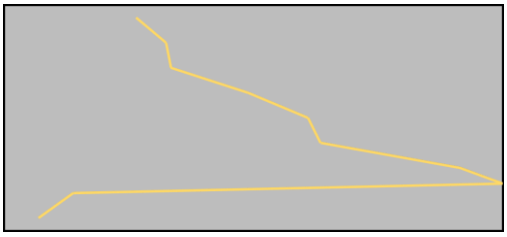
2. 不公平，有可能 Starvation

SCAN 法則

一、Head 來回雙向移動掃瞄，遇有 Track Request，即行服務，Head 遇到 Track 開端或盡頭時，才折返提供服務



二、例：



(199-53)+(199-14) = 331 軌

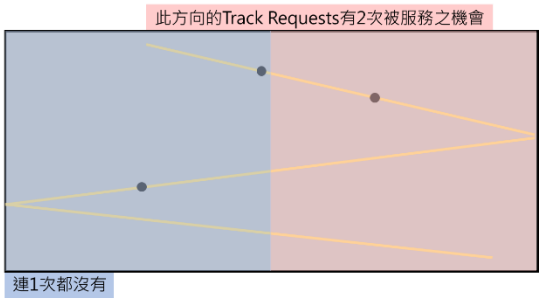
三、分析：

1. 排班效能可接受，適用於大量負載之情況(因為 Track Request 有較均勻一致之等待時間)

   Note：Look 也是

2. 在某些時刻，對某些 Track Request，似乎不盡公平

   例：



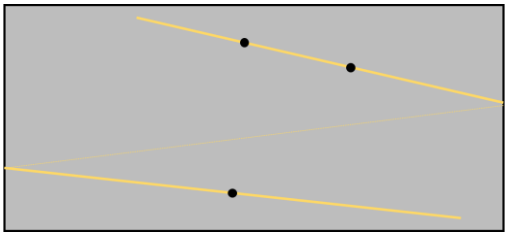
Note：用 C-SCAN 解

3. Head 需要遇到 Track 開端或盡頭才折返，此舉耗費不必要的 Seek Time

Note：用 Look 解

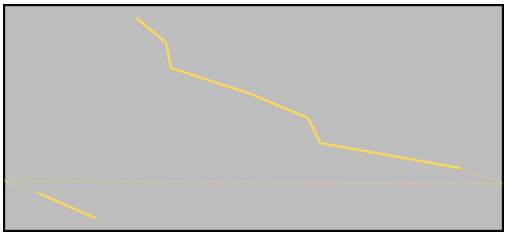
C-SCAN

一、Def：是 SCAN 變型；差別只是提供單向服務，折返回程不提供服務



爭議在計算上：折返的 Track 移動數是否列入？*(個人意見為不列入)*

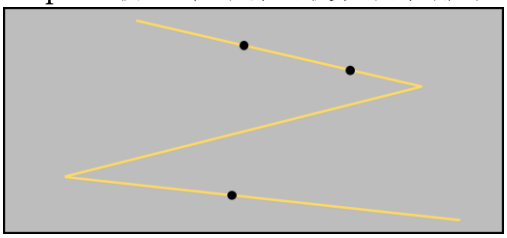
二、例：



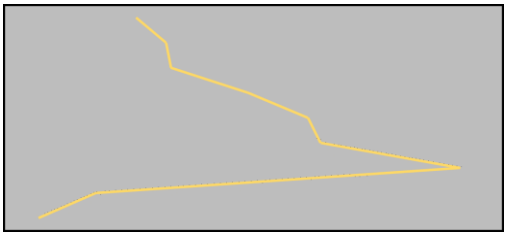
*(199-53)+(37-0)或(199-53)+(37-0)+**(199-0)***

Look

一、Def：是 SCAN 之變形；差別：Head 服務完該方向之最後一個 Track Request 後，即可折返提供回程服務



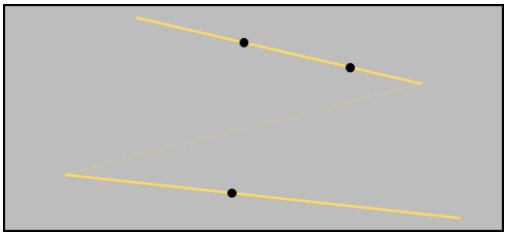
二、例：



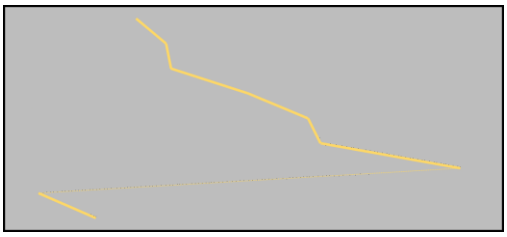
(183-53)+(183-14) = 299 軌

C-Look

一、Def：Look 之變形；差別：只提供單向服務



二、例：



(183-53)+(183-14)+(37-14) = 322 軌

[補充]

Elevator???

[恐]	[Modern]與其他版本
SCAN	X
C-SCAN	X
Look	SCAN
C-Look	C-SCAN

Elevator 法則

其他名詞

Formatting(格式化)(p9-18)

一、Physical Format (Low-level Format)：

1. 工廠生產 Disk System 時執行
2. 主要是劃分出 Disk Controller 可以存取的 Sector

Head	Data area	Trail
控制已訊		控制資訊

Ex：Sector Number, Parity-check…等

3. 偵測有無 BAD Sectors

二、Logical Format：user 在使用 Disk 之前，必須作 2 件事：

1. Partition：切割分區，即 Logical Drive(ex：C, D, E 磁碟機)
2. Logical Format：OS 製作(寫入)File Management System 所需之資料結構
   (1) Free Space 管理(ex：Bit Vector)
   (2) FAT, I-Node
   (3) 空的 Physical Directory

RAW-IO(p9-19)

Def：將 Disk 視為大型 Array 在使用，1 個 Sector 好比是 Array 的一格

無 File System 之支援

優點：存取速度快速

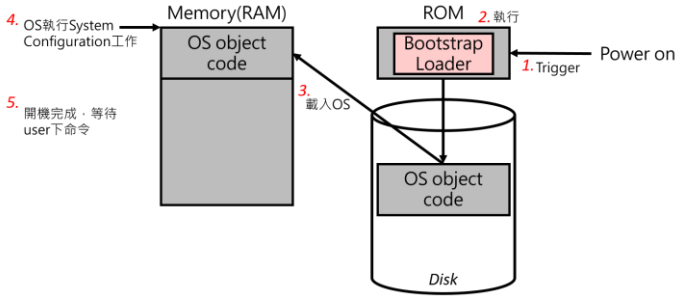
缺點：user 不易使用

通常在 Database Main Structure(DBMS)『底層』使用

Bootstrap Loader

一、主要目的：開機時，用以從 Disk 載入 OS object code 到 Memory(RAM)的特殊 Loader

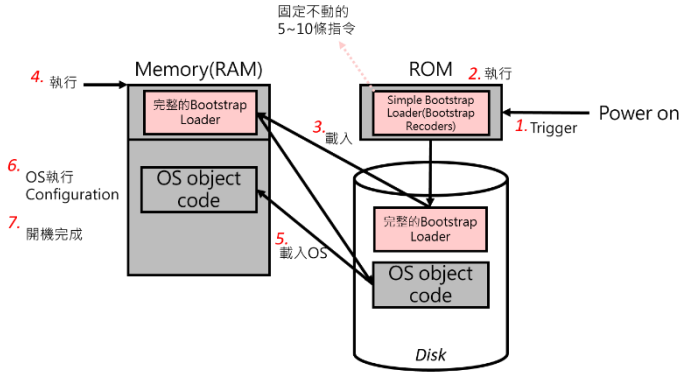
二、[早期的作法]



缺點：

1. Bootstrap Loader 無法任意變更
2. ROM size 有限，Bootstrap Loader 無法很大

三、[近代/現今的作法]



Bootstrap Loader 是放在 Disk 的固定 Blocks，稱為 Boot Blocks

擁有 Boot Blocks 之 Disk 叫作 Boot Disk 或 System Disk

BAD Sectors 之處理方法

一、Sector 會壞之原因

1. 工廠生產時已壞
2. 正常使用一段時間後損壞

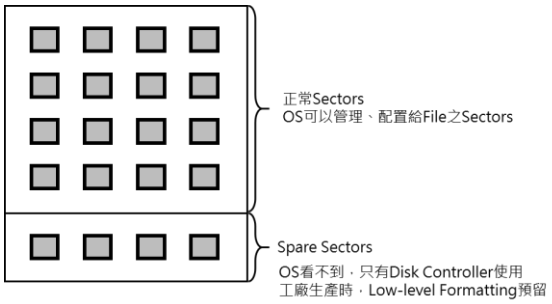
二、處理方法：

[法一]：Mark(標示)Bad Sectors：以後不用

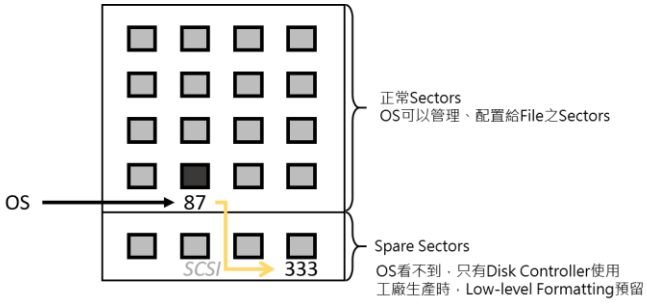
例：IDE Disk Controller

[法二]：Spare Sectors 方法

例：SCSI Disk Controller 採用

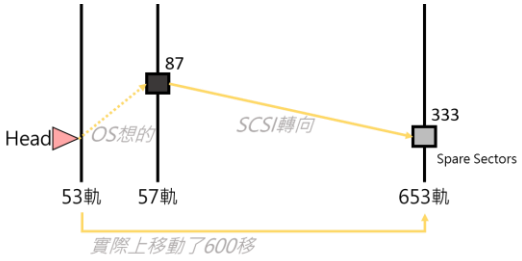


一旦有 Bad Sector(ex：87 號 Sector Bad)，則 Disk Controller 會從 Spare Sectors 選擇一個 Spare Sector(ex：333 號 Sector)來替代 Bad Sector，將來 OS 在存取 87 號 Sector 時，SCSI Controller 會將它轉向成對的 Sector(333 號 Sector)來存取(但 OS 不知道此事)



缺點：此一 Sector 轉向存取之動作，可能會破壞掉 OS 『Disk Scheduling』之最佳效益

例：

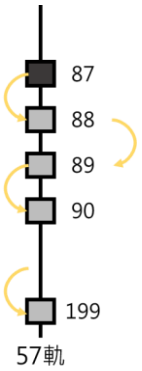


改善作法：

將 Spare Sectors 分散到每條 Track (or Cylinder)上，不要集中存放；若 Sector 壞掉，則用相同或鄰近的 Track 上的 Spare Sector 作替代

[法三]：Sector Slipping 法

例：



依序將：

199 號 Sector 內容 Copy 到 200 號 Sector、

198 號 Sector 內容 Copy 到 199 號 Sector、

…

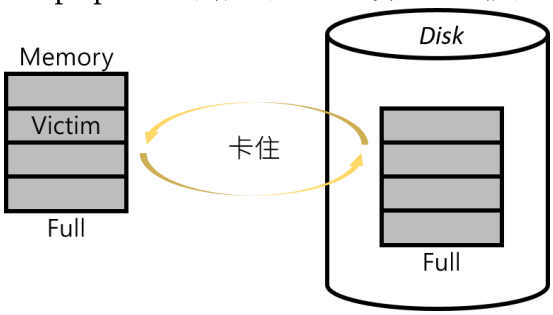
88 號 Sector 內容 Copy 到 89 號 Sector、

Then，88 號 Sector 作為 87 號之 Spare Sector

SWAP Space Management

一、在 Virtual Memory, Medium-Term Scheduler，會將 Disk 作為 Swap out 的 Page 或 Process Image 之暫存處

二、Swap Space 空間大小，宜超估，比較安全



三、Swap Space Management 方法

(用什麼方式保存 Swap out Page/Process Image ?)

[法一]：用 File System：仍然以 File 型式保存：

優點：Easy

缺點：

1. 有外部碎裂*(因為大都採 Contiguous Allocation)*
2. 效能比較差

[法二]：獨立的"Partition"來保存

優點：效能佳(因為 RAW-IO，無 File System 支持)

缺點：

1. 內部碎裂
2. 若 Partition 不夠大，則需重新 Re-partition

提升 Disk Data Access performance 之技術

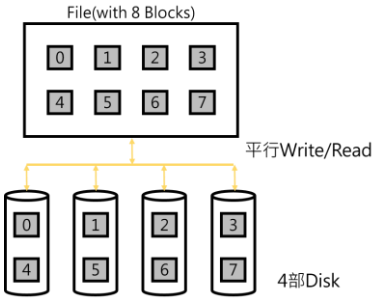
Data (Disk) Striping/Interleaving 技術

Def：將多部 Physical Disks 組成單一的 Logical Disk，運用平行存取的技巧，來提升效能

一般分為 2 種：

1. Bit-Level Striping
2. Block-Level Striping

例：File(大小=8 個 Blocks)



只需 2 次

提升 Disk Reliability(可靠度)之技術

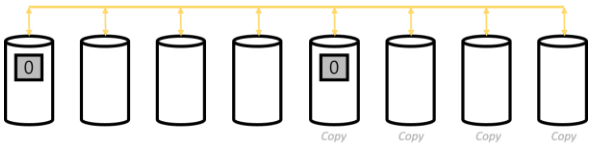
(當 Block 壞了、Data Lost，如何作 Data Recovery？)

1. Mirror(或 Shadow)技術
2. Parity-check 技術

Mirror

Def：每一部正常的 Disk 均配備有對應的 Mirror Disk。資料需同時存入正常 Disk 與其 Mirror Disk。將來若正常 Disk 損壞，則用其 Mirror Disk 替代。

例：4 部正常 Disk



優點：

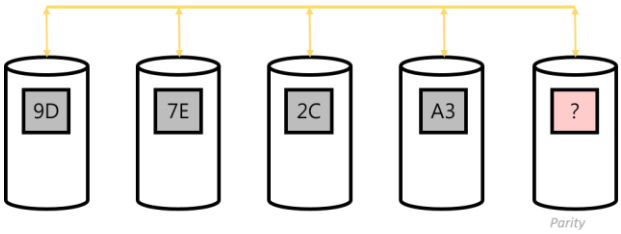
1. 可靠度最高
2. Data Recovery 速度最快

缺點：成本高(貴)

Parity-check 技術

Def：多準備一部 Disk 作為儲存 Parity-check Blocks 之用。資料寫入時，需額外算出 Parity-check Block 內容。將來若某一個 Block 損壞，只要用其他 Blocks 及 Parity Block 作"偶同位"，即可 Recovery Data

例：



1001 1101
0111 1110
0010 1100
1010 0011
0110 1100 = 6C

偶同位(Even)

優點：成本較 Mirror 便宜許多

缺點：

1. 可靠度低於 Mirror(若多個 Blocks 同時損壞，則無法修復)
2. Data Recovery 速度慢於 Mirror
3. 資料寫入速度也慢

RAID 規格(Redundant Array of Independent Disks)

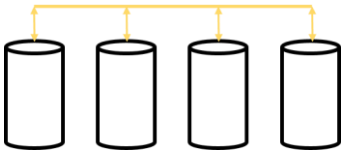
(磁碟冗餘陣列)

RAID 0~6

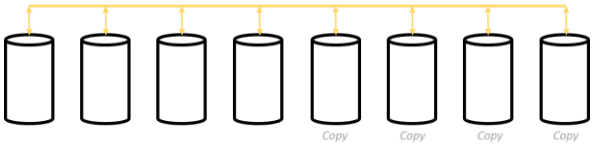
RAID 0 與 RAID 1 組合

例子皆以 4 部正常使用 Disk 為例(p9-28)

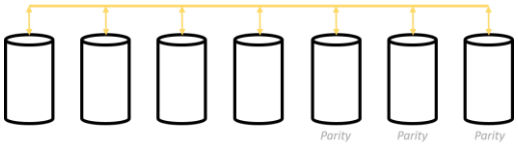
1. RAID 0：只提供 Block-level Striping(Interleaving)而已，未提供任何可靠度技術*(ex：Mirror, Parity-check)*，用在強調存取效能要求高、但可靠度不重要之場合*(ex：VOD Server)*



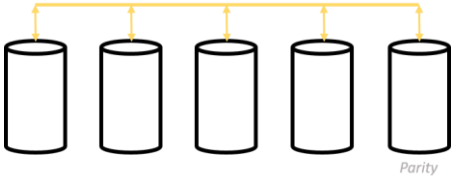
2. RAID 1：就是 Mirror



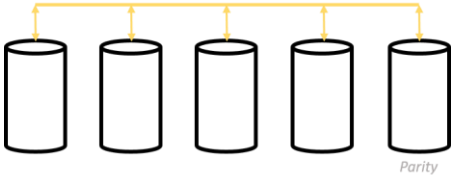
3. RAID 2：採用 Memory 的 DCC 技術來改善可靠度，希望降低 Mirror 成本，但是成本降低有限(只比 Mirror 少一部 Disk 而已)。此外，與 RAID 3 相比，可靠度一樣，但成本高於 RAID 3。因此無實際產品



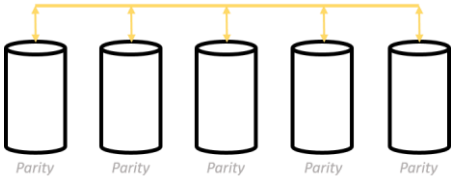
4. RAID 3：採用 Bit-level Striping 及 Parity-check 技術(額外多一部 Disk 即可)



5. RAID 4：採用 Block-level Striping 及 Parity-check 技術



6. RAID 5：採用 Block-level Striping 及 Parity-check 技術，與 RAID 4(RAID 3)主要差別在於：將 Parity Blocks 分散存於不同 Disks，並非集中在一部 Disk，以避免對單一 Disk 之過度使用*(適用於大量資料儲存)*

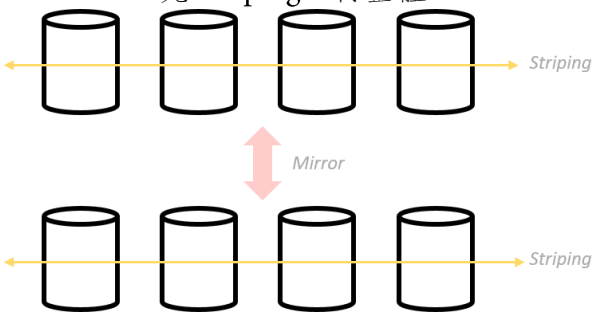


7. RAID 6：不用 Parity-check 技術，改用類似 Read-Solomon 技巧，可以作到 2 部 Disk Block 同時出錯，還能 Recovery，但成本太高(>=Mirror 成本)，也無實際產品

Note：RAID 2 與 RAID 6 皆無實際產品。口訣：2266

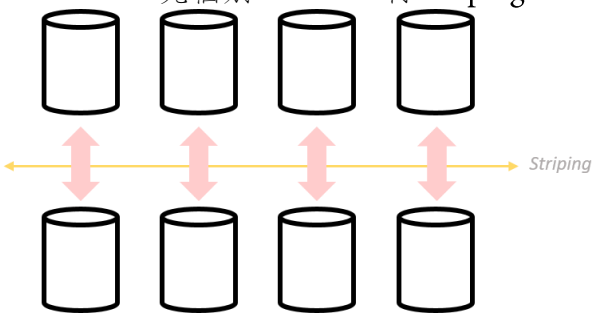
8. RAID 0 與 RAID 1 組合：用在高效能及高可靠度要求之場合，雖然成本很高，組合方式有 2：RAID 0+1 與 RAID 1+0(較好)

(1) RAID 0+1：先 Striping，再整體 Mirror



缺點：一部 Disk 損壞，則需整組替換

(2) RAID 1+0：先個別 Mirror，再 Striping



P9-31 小結：RAID 選擇