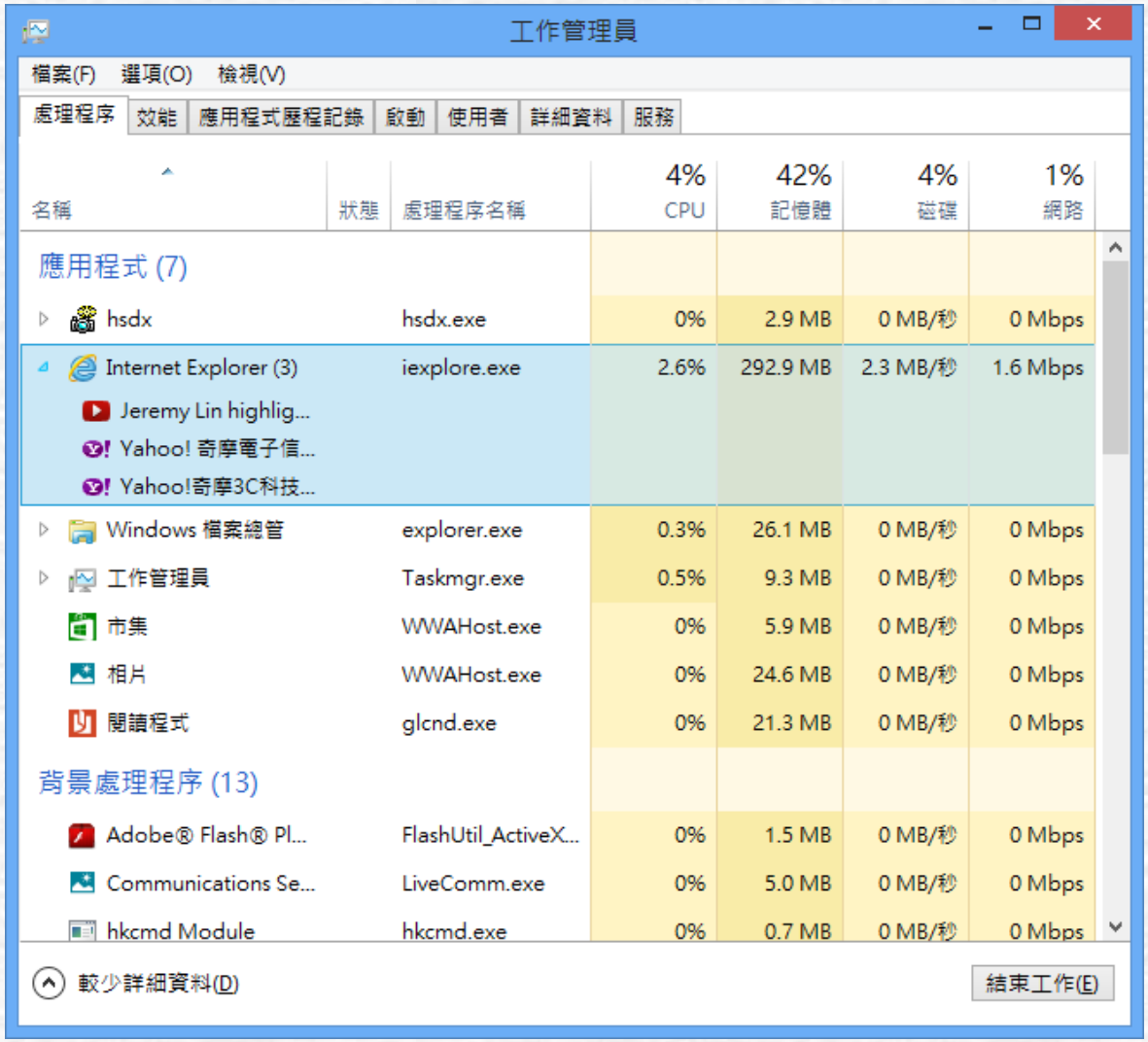
第 2 章 行程管理
Process Management

2-1 何謂行程

- 電腦系統的"程式指令"在未執行時，通常是以靜態的"檔案"形式儲存在輔助記憶體(例如:硬碟)中
- 當程式被載入系統執行時，就稱為行程

程式(Program) V.S. 行程(Process)

圖2-1 IE行程範例


工作管理員
檔案(F) 選項(O) 檢視(V)
處理程序 效能 應用程式歷程記錄 啟動 使用者 詳細資料 服務
名稱 狀態 處理程序名稱 4% CPU 42% 記憶體 4% 磁碟 1% 網路
應用程式 (7)
hsdx hsdx.exe 0% 2.9 MB 0 MB/秒 0 Mbps
Internet Explorer (3) iexplore.exe 2.6% 292.9 MB 2.3 MB/秒 1.6 Mbps
Jeremy Lin highlig...
Yahoo! 奇摩電子信...
Yahoo!奇摩3C科技...
Windows 檔案總管 explorer.exe 0.3% 26.1 MB 0 MB/秒 0 Mbps
工作管理員 Taskmgr.exe 0.5% 9.3 MB 0 MB/秒 0 Mbps
市集 WWAHost.exe 0% 5.9 MB 0 MB/秒 0 Mbps
相片 WWAHost.exe 0% 24.6 MB 0 MB/秒 0 Mbps
閱讀程式 glcnd.exe 0% 21.3 MB 0 MB/秒 0 Mbps
背景處理程序 (13)
Adobe® Flash® Pl... FlashUtil_ActiveX... 0% 1.5 MB 0 MB/秒 0 Mbps
Communications Se... LiveComm.exe 0% 5.0 MB 0 MB/秒 0 Mbps
hkcmd Module hkcmd.exe 0% 0.7 MB 0 MB/秒 0 Mbps
較少詳細資料(D)
結束工作(E)

為什麼需要行程？

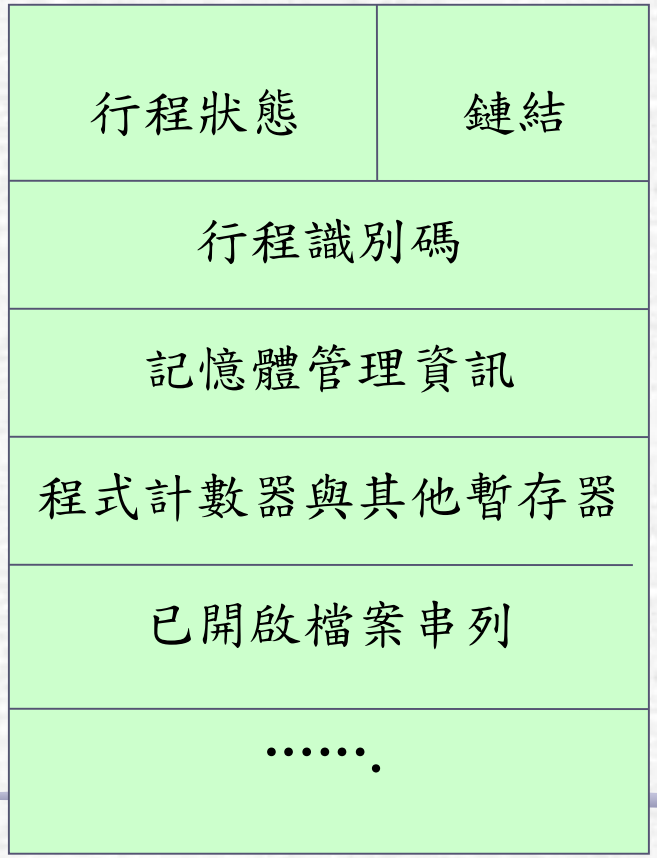
- 行程=程式嗎？
- 行程包含：
  - 程式區段
  - 資料區段
  - 堆疊區段
  - 控制資訊

行程與程式的主要差異

- “行程”是執行中的”程式”
- 行程是主動的個體，程式是被動的個體
- 行程是暫時的，程式是長存的
- 行程與程式的組成內容不同

行程控制區段（Process Control Block）

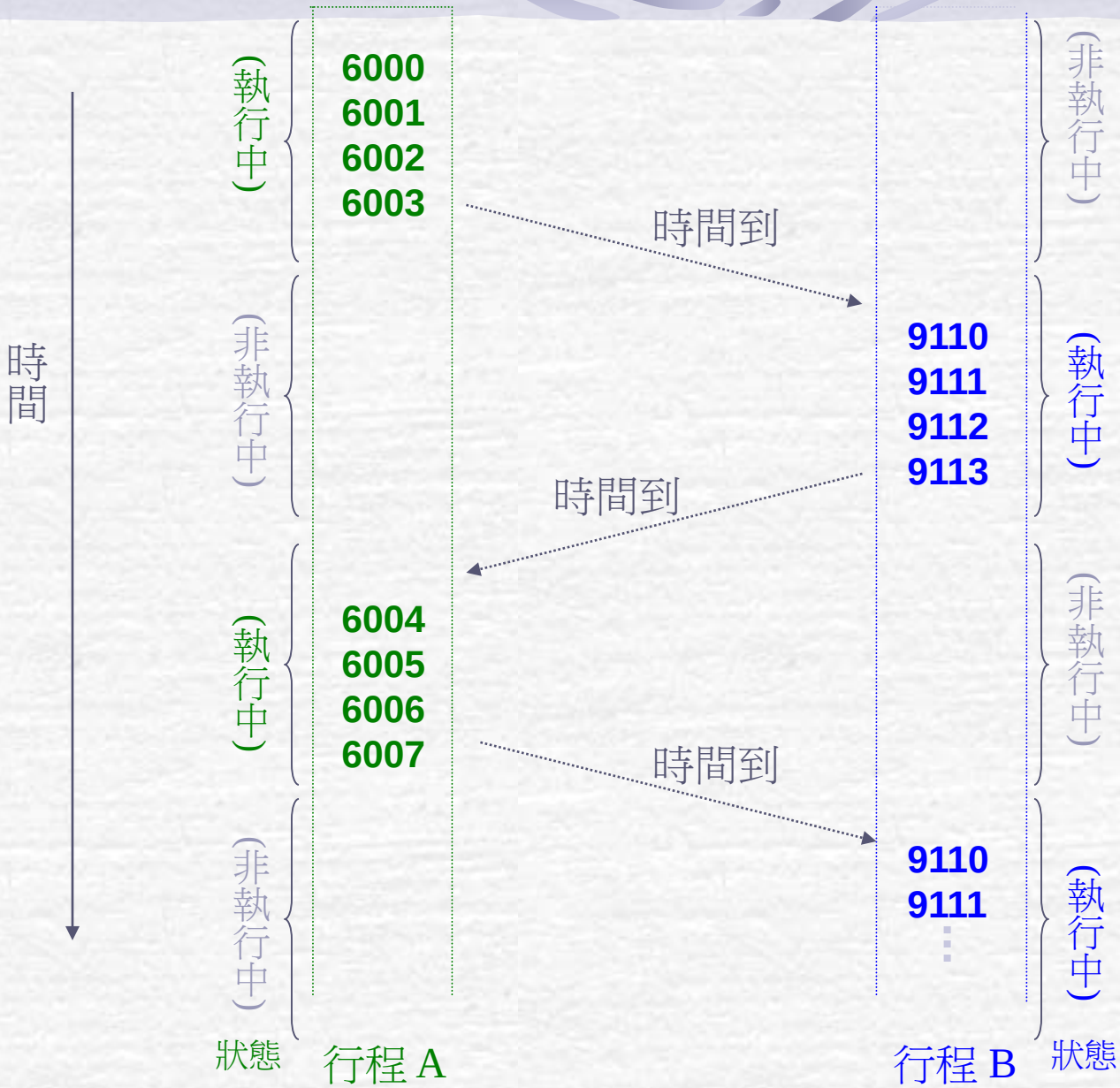
- 作業系統會將行程的控制資訊存放在行程控制區段（PCB）中



2-2 行程狀態

- 行程執行的過程中，可能會經歷多個不同的狀態
- 行程的狀態反應出行程目前的動作，以及能做的事情
  - 在每個狀態下能做的事情是不一樣的
  - 不同狀態之間會因為某些條件而發生轉換

最簡單的行程狀態模型-雙狀態模型範例



雙狀態模型的狀態轉換

- 在實際情況下，是由分派程式來決定接著是哪個行程能進入執行中的狀態
- 在雙狀態模型中，當新行程建立後，首先會以非執行中狀態進入系統佇列，等待機會執行
- 當目前正在執行中的行程被中斷後，分派程式會從在佇列內等待的行程中選擇一個來執行

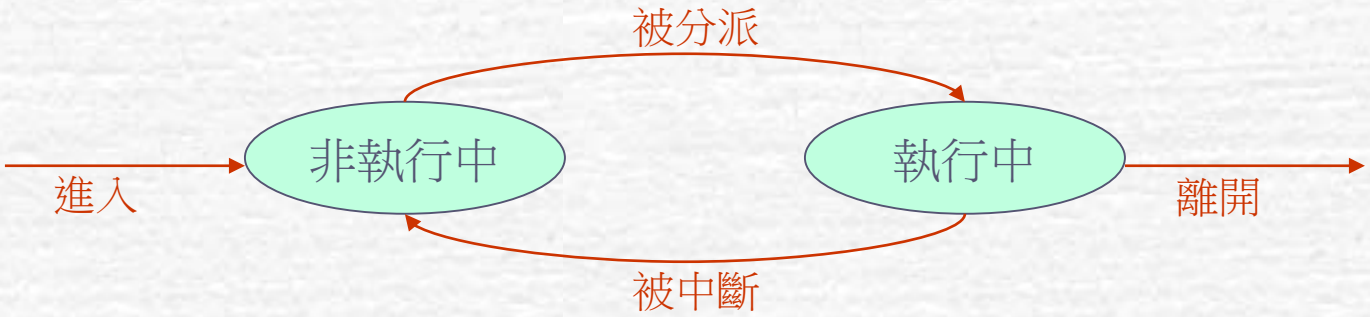
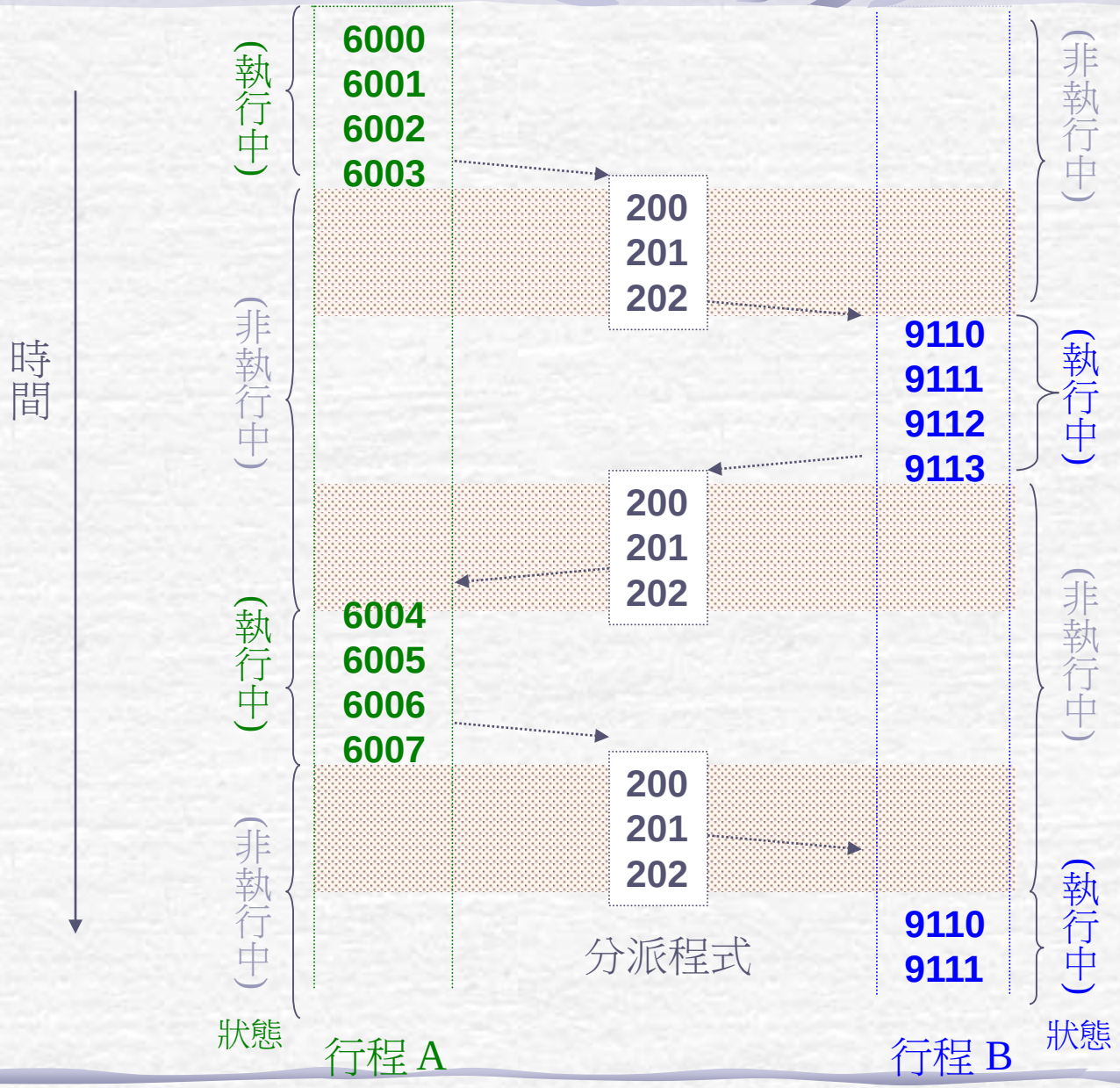


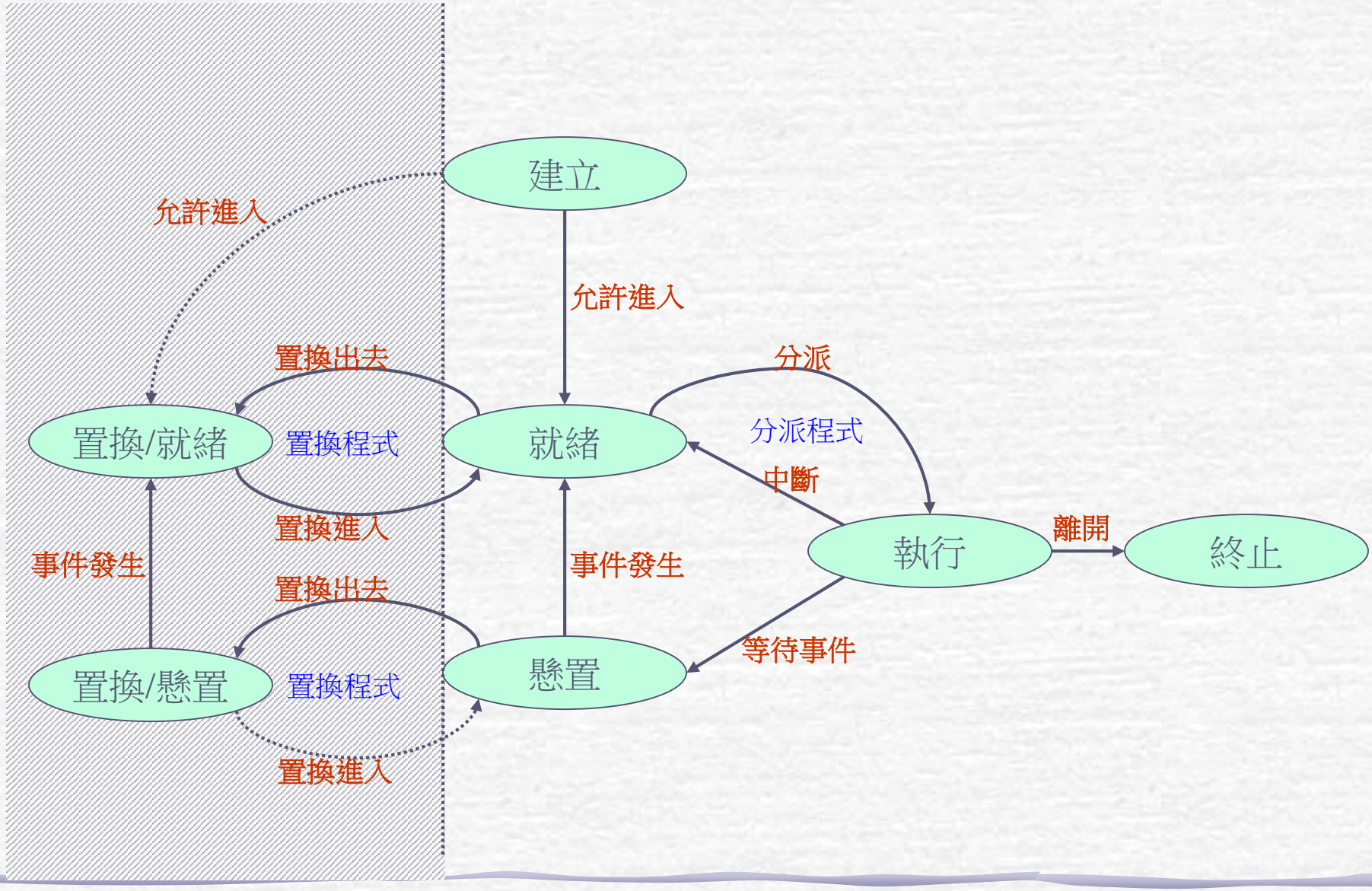
圖2-3 加上分派程式之雙狀態模型範例

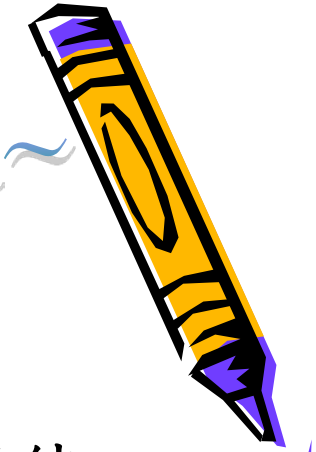


長短期排程

- 排程：在多工系統中，決定哪個行程能夠取得CPU控制權的過程
- 長期排程器：
  - 用來決定哪些任務（行程）可以參與系統資源的競爭；常見於傳統的批次作業系統
- 短期排程器：也稱為分派程式
  - 負責將CPU分配給已經進入記憶體，並且已經就緒可以執行的行程使用
- 中期排程器：也稱為置換程式

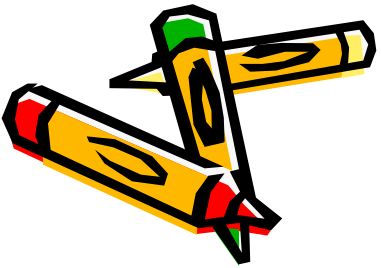
圖2-5 完整的行程狀態轉換圖





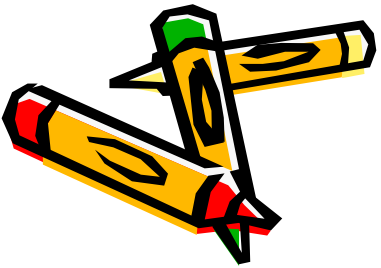
課堂練習-行程狀態轉換

- 假設有一行程在執行了一個I/O運算。在等待運算完成的過程中，因為系統的多工程度過高，被置換程式移到輔助記憶體中。
- 當這個行程還停留在輔助記憶體的時候，它的I/O運算就完成了。
- 請問，直到它被重新執行的時候，一共經歷了哪些狀態？



練習解答

- 執行→懸置→置換/懸置→置換/就緒→就緒→執行



2-3 行程的建立

- 當行程處於建立狀態時，表示作業系統已經完成行程控制資訊的建立，但是還沒有將行程移入記憶體中
  - 這樣的設計技巧有助於作業系統區分行程控制與記憶體配置的工作

產生新行程的時機：

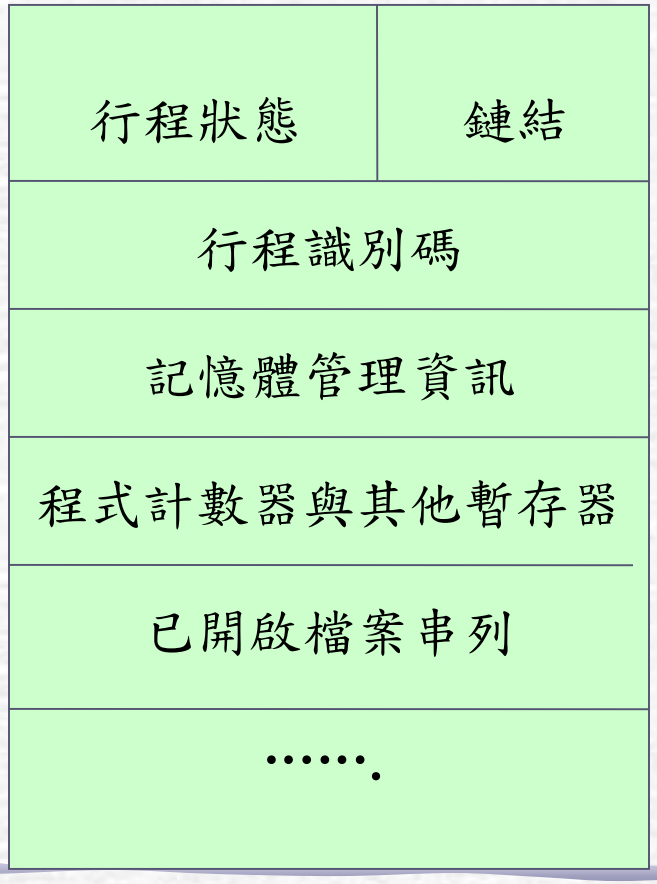
- 使用者登入
- 使用者執行某個程式
- 請求系統提供服務
- 由現有行程產生→父行程

行程的結束

- 當行程處於終止狀態，代表行程已經不能執行，但是它的一些相關資訊與表格仍然可以暫時保存在作業系統中
- 行程結束的可能原因：
  - 任務完成
  - 保護錯誤
  - 運算錯誤
  - I/O失敗
  - 外力終止
  - 父行程終止
  - 父行程要求

2-4 行程的控制

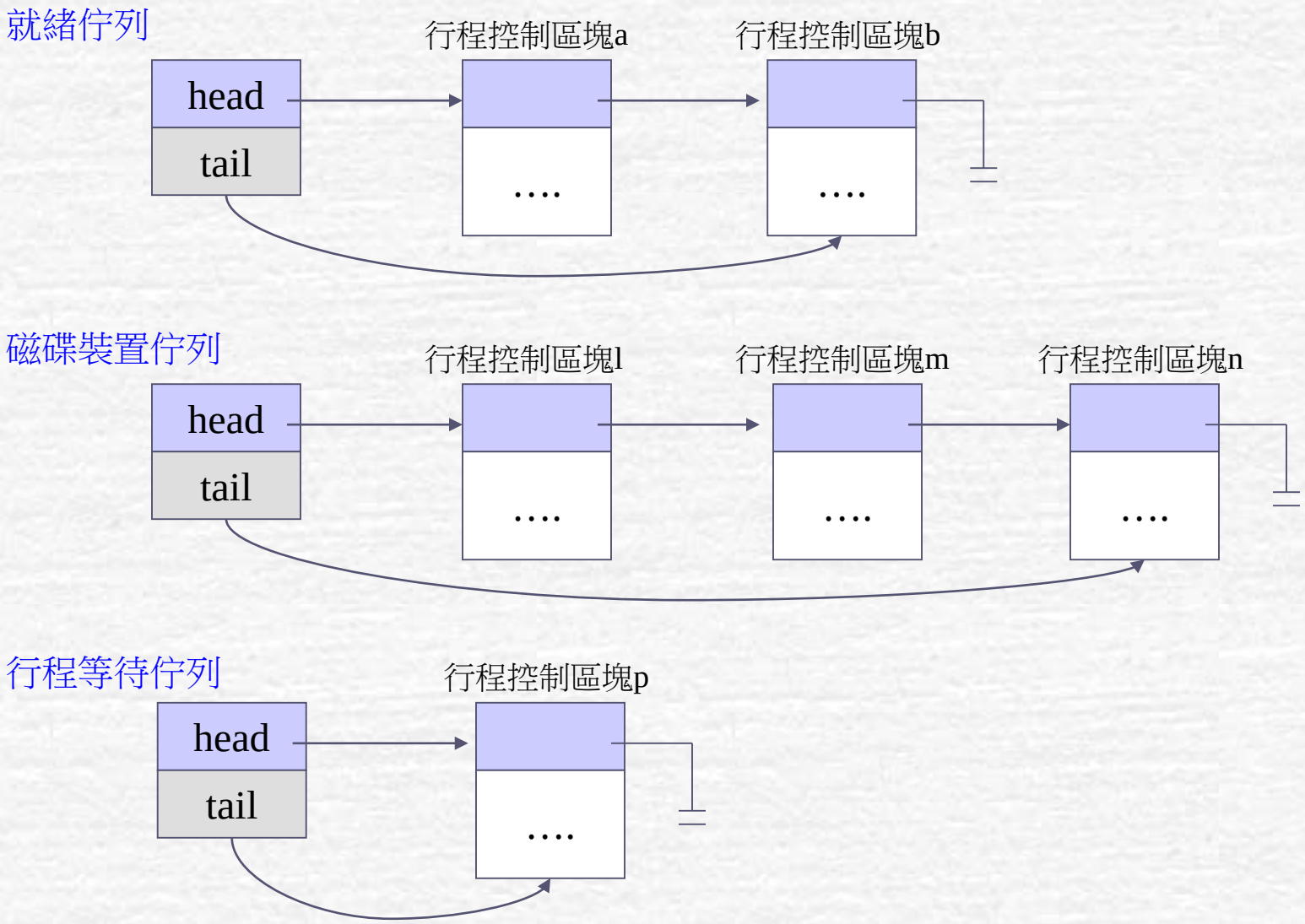
- 行程是作業系統管理與分配資源的基本單位
- 作業系統會將行程的控制資訊存放在行程控制區段（PCB）中



行程的佇列

- 作業系統使用佇列來協助進行排程
- 工作佇列：用來記錄系統中所有的行程
- 就緒佇列：用來存放就緒狀態的行程
- 裝置佇列：為了等待I/O完成所產生的佇列，每個裝置都有各自對應的裝置佇列
- 等待佇列：存放正在執行系統呼叫或是等待事件發生的行程

圖2-8 就緒佇列與裝置佇列



內文切換(Context Switch)

- 內文切換：當CPU的使用權由一個行程切換另一個行程時，作業系統必須將行程的相關資訊儲存在該行程的行程控制區段中，並且將另一個行程的程式控制區段載入系統，以便將執行環境復原為後者當初被中斷時的狀態

內文切換

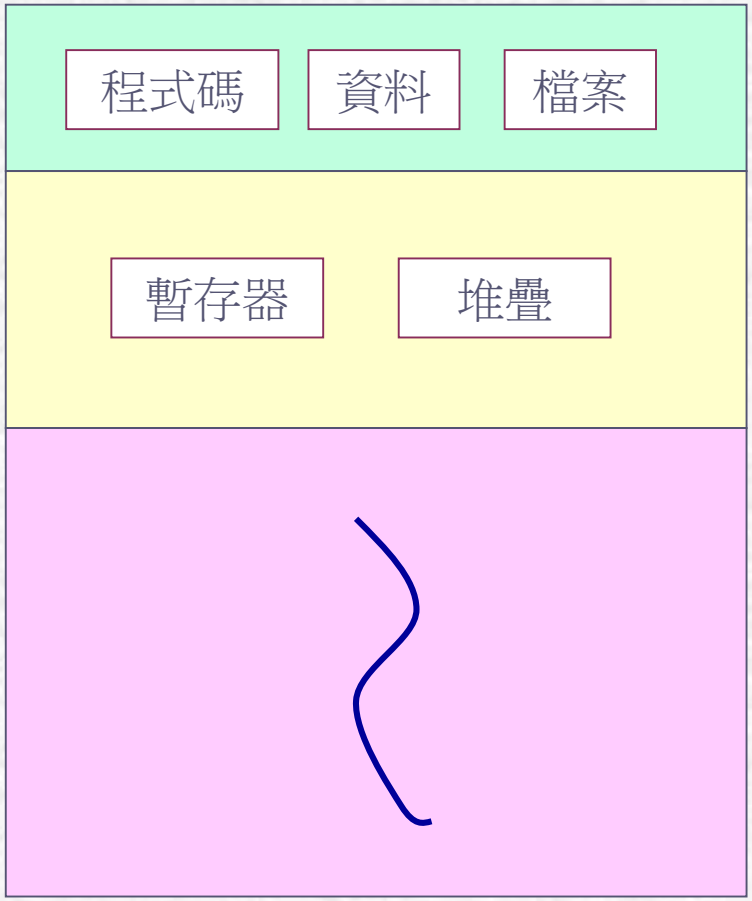

行程A
作業系統
行程B
執行中
中斷或系統呼叫
閒置
內文切換
將狀態儲存到PCB A
由PCB B載入狀態
閒置
執行中
中斷或系統呼叫
將狀態儲存到PCB B
內文切換
由PCB A載入狀態
閒置
執行中

內文切換與中斷

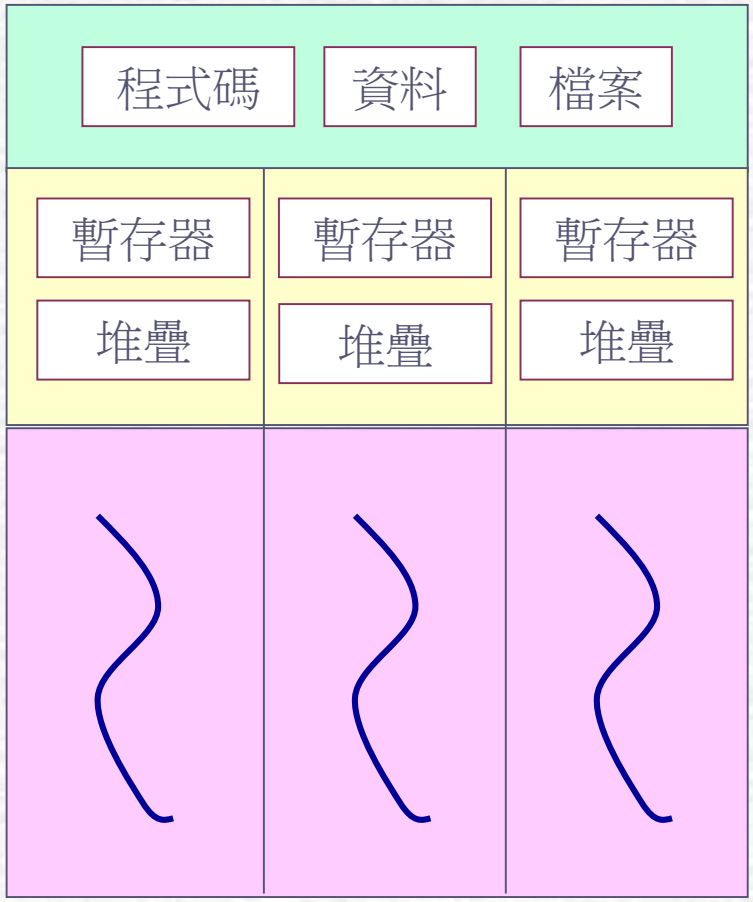
- 行程的內文切換需要作業系統先重新取得控制權
- 作業系統通常是透過中斷來重新取得控制權
  - 與目前執行中行程無關的事件所引起的，通常就直接稱為中斷，例如時鐘中斷與輸入/輸出中斷
  - 與目前執行中的行程相關的事件所引起的，包括例外中斷與軟體中斷
- 就系統的利用率而言，內文切換所花費的時間可以說是純粹的浪費，因為它做的並不是具有生產力的工作
  - 內文切換的速度會隨著電腦硬體架構而有所不同
  - 作業系統越複雜，內文切換時所需執行的工作也會越多

執行緒-輕量級行程

- 執行緒：CPU配置的基本單位
  - 擁有自己的程式計數器、暫存器、與堆疊空間
  - 和同一行程的其他執行緒共享相同的記憶體位址空間、程式區段、資料區段，和一些系統資源
- 傳統的行程相當於是只有一個執行緒的單執行緒行程
  - 單執行緒行程的缺點在於當它同時接收到多項要求的時候，如果不是採取循序執行的方式，就是必須產生多個子行程，以提供較佳的互動
  - 這樣的做法需要頻繁地建立大量的行程與執行內文切換，不僅耗費系統處理時間，而且每個行程都要佔用一塊記憶體空間
- 目前許多的伺服器程式都是使用多執行緒來改善效能



單執行緒行程



多執行緒行程

圖2-10 單執行緒與多執行緒的行程

行程與執行緒的主要差異

- 位址空間：行程間的位址空間是相互獨立的，而屬於同一行程的執行緒間則是共用相同的位址空間。
- 通訊方式：行程間的通訊必須利用作業系統所提供的機制進行；而執行緒間因為共享相同的位址空間，可以透過直接讀寫其全域資料來進行溝通。
- 內文切換：同一行程中的執行緒間進行內文切換的時間遠小於行程間的內文切換。

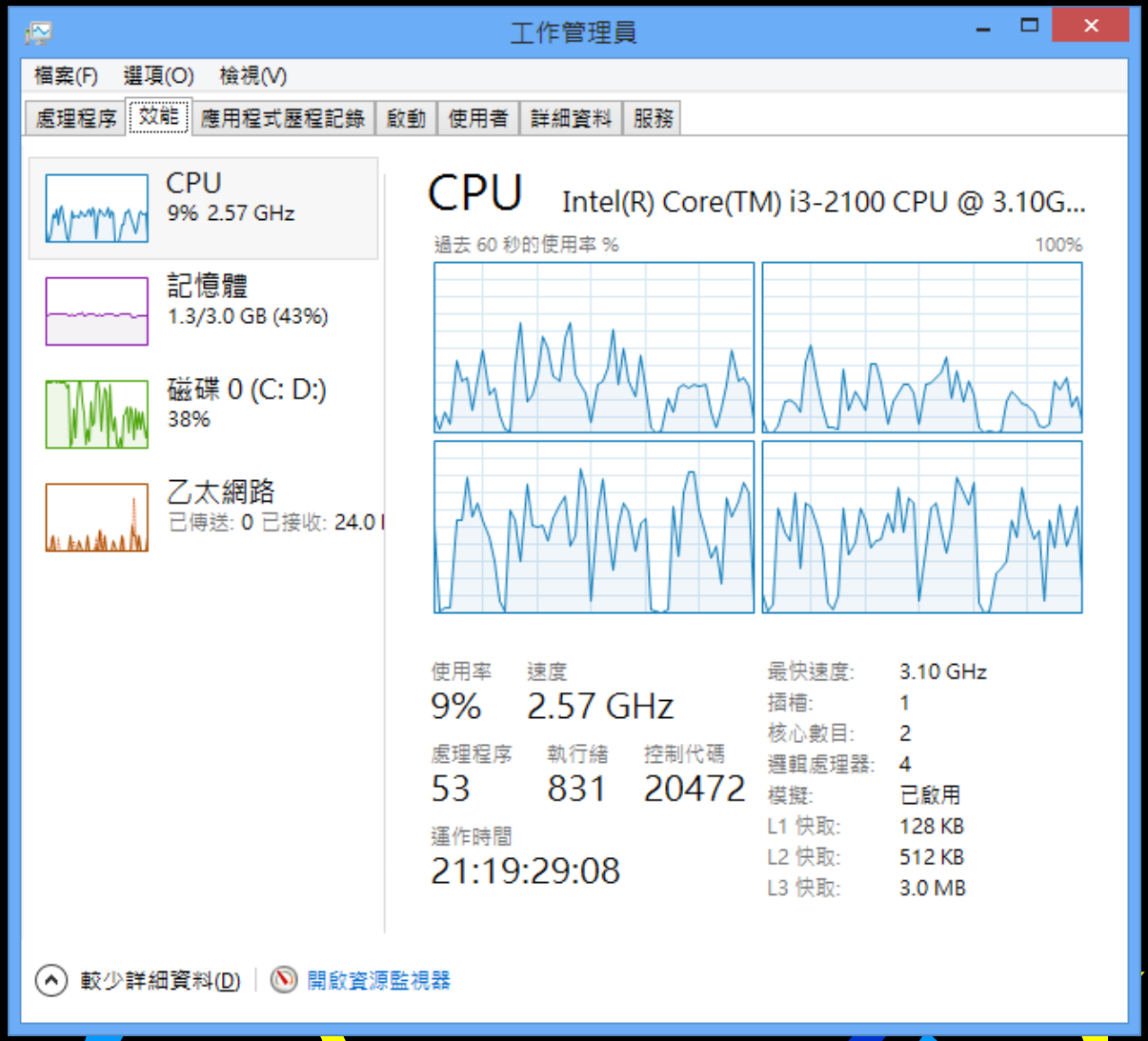
使用多執行緒的優點

- 快速回應
- 資源共享
- 效率
- 平行處理

動手做做看
觀察系統內的行程與執行緒數量

- 按下Ctrl-Alt-Del叫出Windows工作管理員
- 切換到效能頁籤

圖2-11 Windows工作管理員所顯示的行程與執行緒資訊



2-5 行程的排程

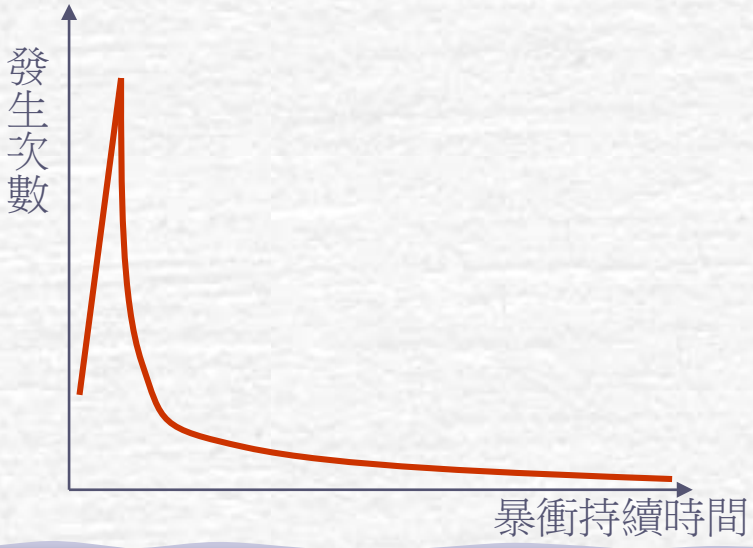
- 行程排程是多工系統的基礎
- 排程的概念：當行程在等待某個事件或I/O運算時，因為無法使用到CPU，所以可以將CPU讓出來給其他需要執行的行程使用
- 行程的執行通常都是在兩種狀況間不斷切換：
  - CPU暴衝：持續地使用CPU
  - I/O暴衝：專注在I/O運算上

行程排程是多工系統的基礎

當發生I/O暴衝的時候，行程會進入懸置狀態

CPU暴衝的分布形式

- 不同電腦架構與不同行程出現的CPU暴衝時間通常有相當大的差異
- 但原則上大都具有大量極短的CPU暴衝，而較長的CPU暴衝數量則少得多
- 這種CPU暴衝的分布形式，對於如何選擇或設計適當的CPU排程演算法而言，是非常重要的資料
  - 以CPU為主的程式：需要大量CPU運算
  - 以I/O為主的程式：通常會有很多很短的CPU暴衝



排程效能的衡量

- 排程的目的就是希望在所有就緒的行程中挑出一個能帶來最佳效能者
- 常見的一些排程標準
  - CPU使用率：CPU真正在執行行程的時間比例
  - 產量：系統在單位時間內所能完成的行程數目
  - 回覆時間：單一行程執行完畢平均所需的時間
  - 等待時間：行程花費在等待的平均時間
  - 反應時間：在互動式系統中，當使用者輸入之後，系統開始做出回應所需的平均時間
  - 可預測性：用來評估系統回應的一致程度
  - 公平性：所有行程具有相同執行機會的程度

可搶先與不可搶先式排程

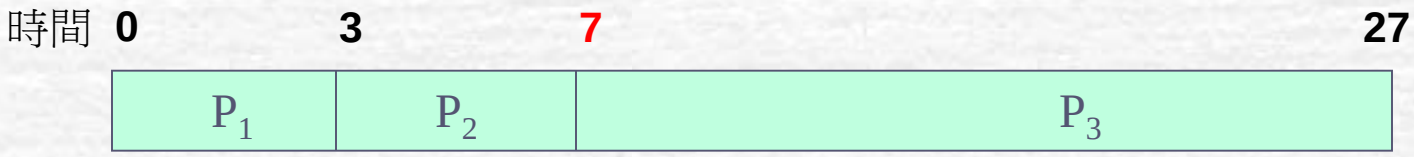
- 不可搶先式排程：只有當執行中的行程進入懸置狀態或是終止的時候，其他行程才有可能取得CPU的控制權
  - 可以讓執行中的行程完成整個CPU暴衝
  - 在現代的互動式系統中，這樣的做法可能會讓其他的使用者或應用程式等待過久
- 可搶先式排程：排班程式可以在行程進入等待或結束之前就先將行程趕出CPU，以便將CPU配置給另一個行程
  - 目前系統採用的做法
  - 需要比較複雜的CPU設計

先到先做排程（FCFS Scheduling）

- 不可搶先式排程法
- 優點：簡單
- 缺點：等待時間變動很大，而且平均等待時間並不短→可預測性低
- FCFS排程對以CPU為主的程式比較有利
- 可能會發生護送現象：所有行程都在等待一個大行程離開CPU的情況

圖2-16 先到先做排程範例

行程	CPU暴衝時間（毫秒）
P_1	3
P_2	4
P_3	20



(a) 以P_1、P_2、P_3順序提出請求　　平均等待時間 = (0+3+7)/3 = 3.3毫秒

(b) 以P_3、P_2、P_1順序提出請求　　平均等待時間 = (0+20+24)/3 = 14.7毫秒

最短工作優先排程（SJF Scheduling）

- 不可搶先式排程
- 希望選出的是下一次CPU暴衝最短的行程
- 最早是出現在批次處理的系統，依賴批次工作所提供的預估時間來做排程
- 現在的做法是利用行程過去暴衝時間的算術平均值或指數平均值，作為對下次CPU暴衝的估計值

$$e_{n+1}=\alpha t_n + (1-\alpha)e_n$$

t_n：第n次暴衝的時間，e_n：第n次暴衝的預期時間

α：最近一次暴衝時間的權重

圖2-17 最短工作優先排程範例

行程	CPU暴衝時間（毫秒）
P_1	20
P_2	4
P_3	3

平均等待時間=3.3毫秒（FCFS的最佳情況）

SJF排程的特性

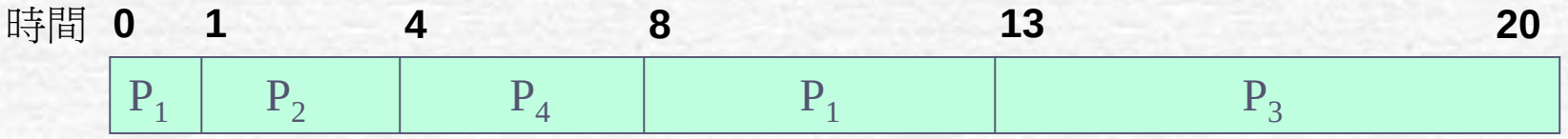
- 所有不可搶先排程法中平均等待時間最短的一個
- 將CPU暴衝時間較長的行程延後執行，藉此降低平均等待時間
- 如果不斷有CPU暴衝較短的行程進入就緒佇列，就可能會導致CPU暴衝較長的行程發生飢餓現象→一直無法取得CPU的控制權

最短剩餘時間排程（SRT Scheduling）

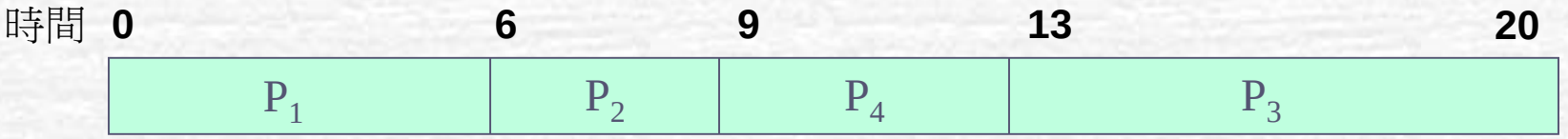
- SJF排程的可搶先版
- 當有新的行程進入就緒佇列時，如果比現在執行中行程所剩下的CPU暴衝時間更短，則現在執行的行程會被趕出去，由新行程搶先執行
- CPU暴衝較長的行程同樣可能會發生飢餓現象

圖2-18 最短剩餘時間排程範例

行程	CPU暴衝時間（毫秒）	抵達時間
P_1	6	0
P_2	3	1
P_3	7	2
P_4	4	3



(a) SRT排程結果　　平均等待時間 = ((8-1) + (1-1) + (13-2) + (4-3)) / 4 = 4.75毫秒



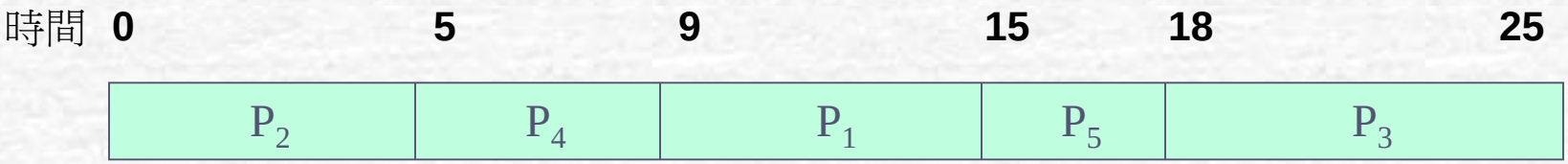
(b) SJF排程結果　　平均的等待時間 = ((0-0) + (6-1) + (13-2) + (9-3)) / 4 = 5.5毫秒

優先權排程（Priority Scheduling）

- 不可搶先式  /可搶先式 
- 排程器依據優先權的高低來排程
- 優先權的指定標準：
  - 行程的特性
  - 使用者等級
  - 指定的某些參數
- 有些指定優先權的機制會讓優先權產生動態變化，有些則是固定的優先權設計
- SJF排程法也可以視為是優先權排程的一個特例
- 可能造成飢餓現象
- 可以利用老化機制解決飢餓現象

圖2-19 優先權排程範例

行程	CPU暴衝時間（毫秒）	優先權
P_1	6	2
P_2	5	0
P_3	7	3
P_4	4	1
P_5	3	2



平均等待時間 = (9+0+18+5+15)/5 = 9.4毫秒

循環分時排程（Round-Robin Scheduling）

- 特別針對分時系統所設計的排程法
- 類似FCFS的小單位搶先版
- 原則上會讓在佇列中等待最久的行程進入CPU執行，可是在執行一段固定時間之後，排程器會將執行中的行程中斷
- 行程可能在CPU暴衝的過程中被中斷，這個切割的時間單位，就稱為時間切片

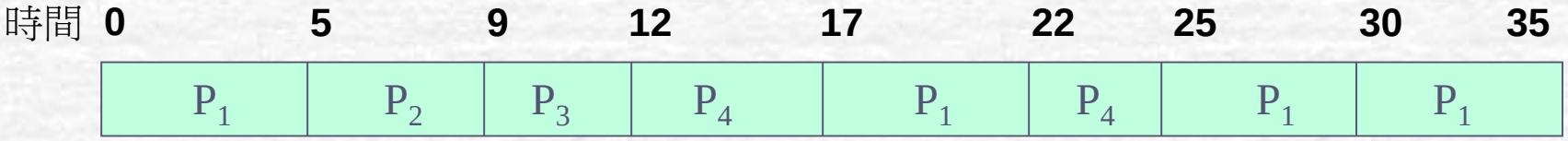
RR排程的優點

- 當系統中存在CPU為主的行程時，不會出現護送現象
- 平均等待時間比較長，但是能提供較好的反應時間
- 時間切片的長度：
  - 切片時間太短，會造成內文切換的頻率過高，而影響系統效能
  - 切片時間太長，則會近似FCFS排程的效果，造成類似護送現象
  - 一般經驗法則：80%的CPU暴衝時間應該要小於一個時間切片的長度

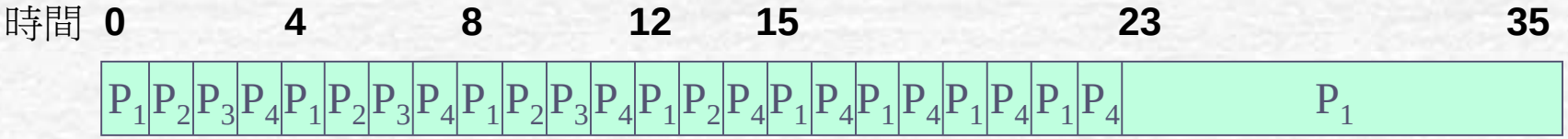
圖2-20 循環分時排程範例

行程	CPU暴衝時間（毫秒）
P_1	20
P_2	4
P_3	3
P_4	8

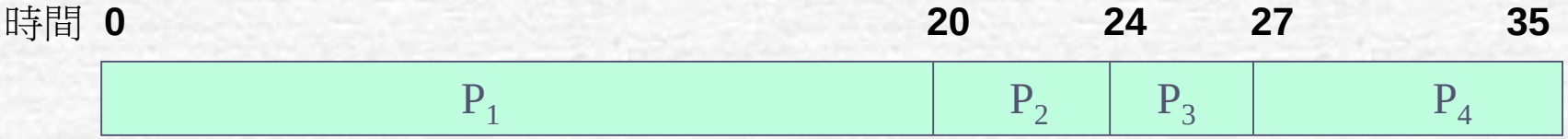
a.時間切片 = 5（毫秒）　平均等待時間 = ((25-10) + 5 + 9 + (22-5)) / 4 = 11.5毫秒



b.時間切片 = 1（毫秒）



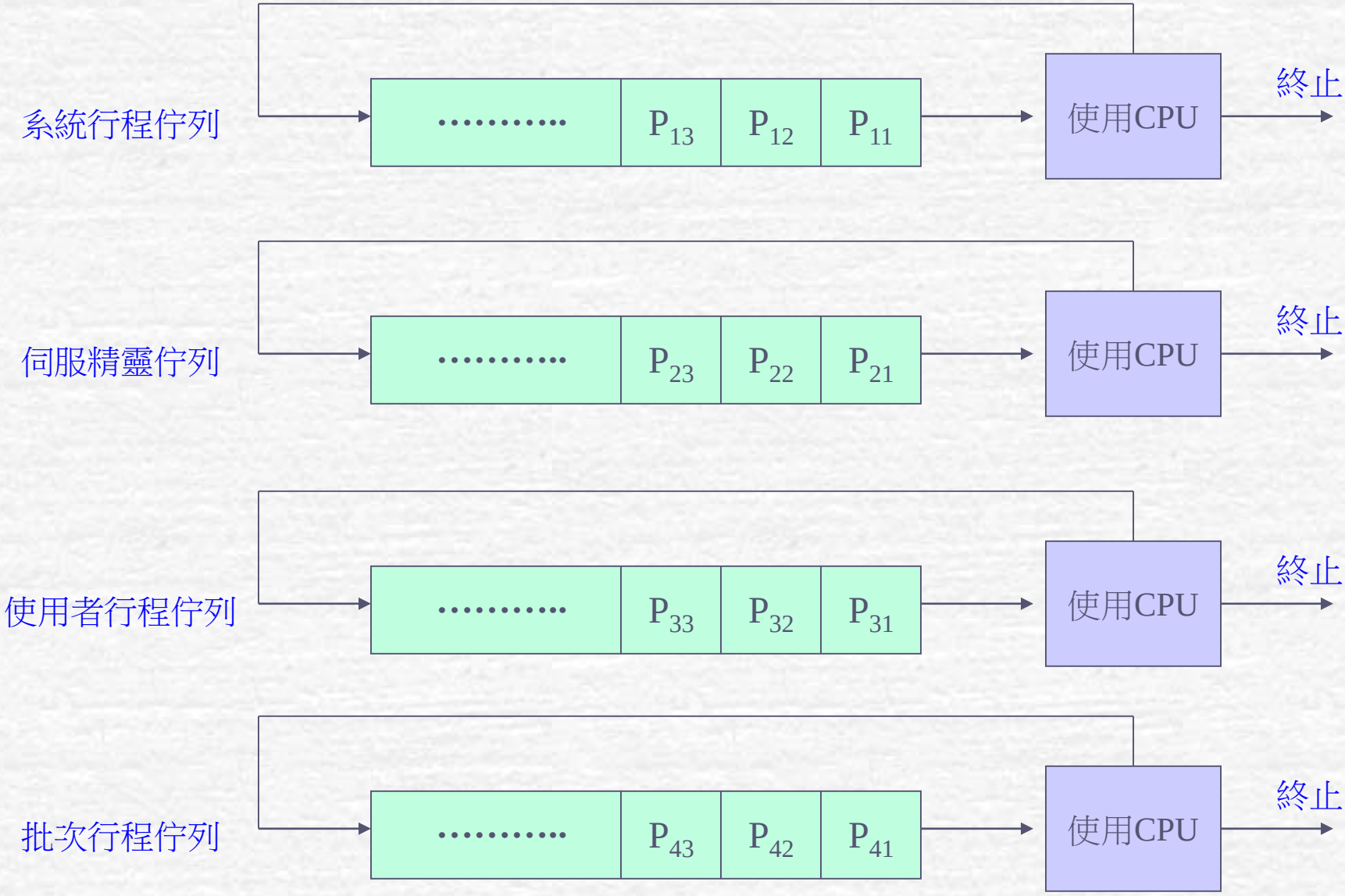
c.時間切片 = 無限大→FCFS



多層佇列排程（Multilevel Queue Scheduling）

- 對行程進行分類，將不同類型的行程放入不同佇列中，在每個佇列內使用最適合該類行程的排程方法
  - 例如將行程分成前景行程與背景行程
- 除了佇列內的排程方式之外，各佇列間還需要一個整體性的排程方式
- 常用的佇列間排程法有：
  - 可搶先式的固定優先權排程法
  - 類似RR的排程法

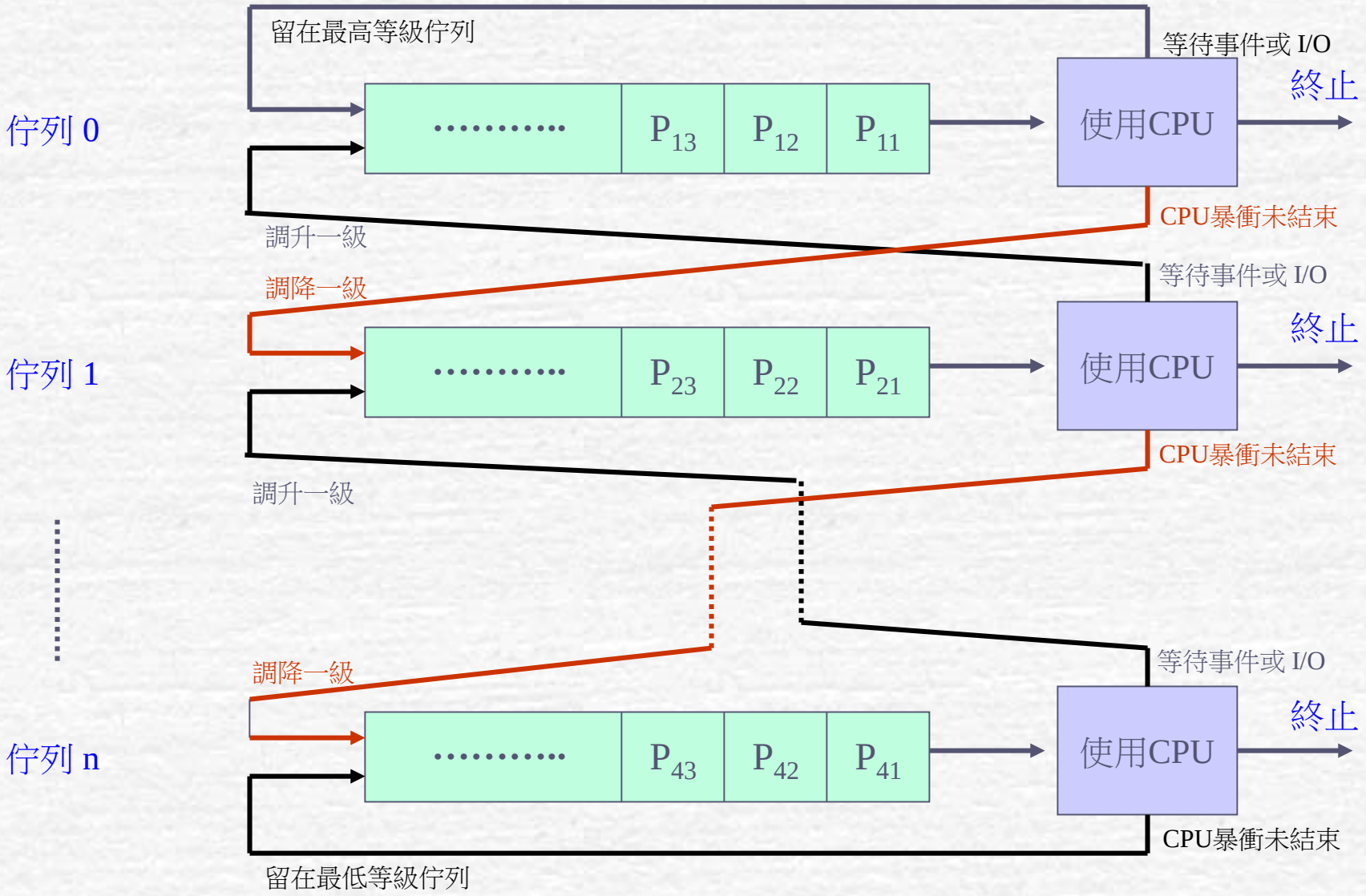
圖2-21 多層佇列排程示意圖



多層反饋佇列排程

- 可搶先的優先權排程
- 根據CPU暴衝的時間，將行程在各佇列間動態移動
- 以I/O為主的行程通常會停留在較高等級的佇列
- 以CPU為主的行程一旦取得CPU控制權，就可以分配到較長的時間切片
- 可能發生飢餓現象，可利用老化機制來避免

圖2-22 多層回饋佇列排程示意圖



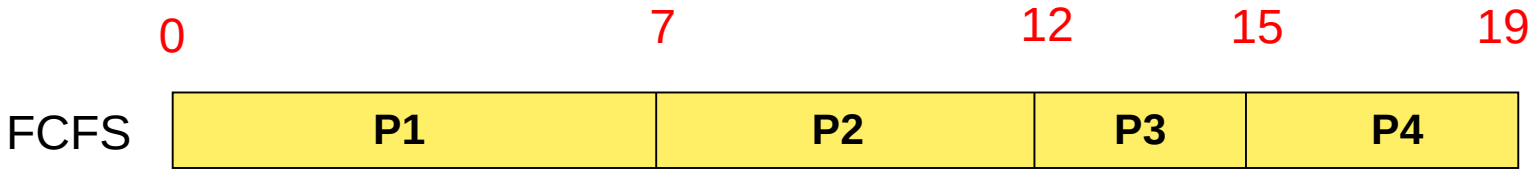
課堂練習：排程法（一）

範例補充包-Extra~

- 假設有四個行程 P1、P2、P3 和 P4，都在時間 0 到達，順序為 P1、P2、P3、P4
- 請針對 FCFS、SJF、和不可搶先的優先權排程法：
  1. 請畫出與課文類似的行程執行順序圖
  2. 計算所有行程的平均等待時間
  3. 每個行程的回覆時間各為何？

行程	CPU 暴衝時間（ms）	優先權 （0的優先權最高）
P1	7	1
P2	5	0
P3	3	3
P4	4	1

練習一解答-FCFS

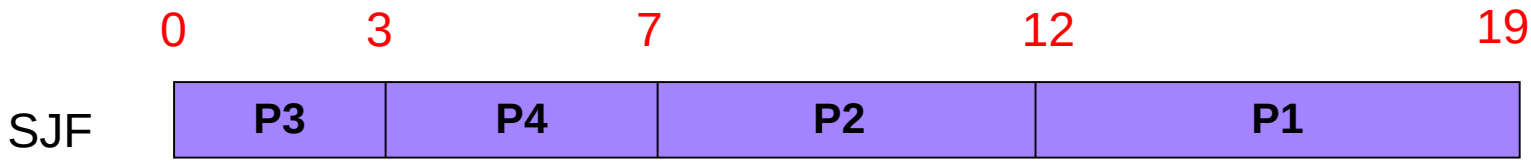


平均等待時間：(0 + 7 + 12 + 15) / 4 = 8.5

回覆時間：

P1	7
P2	12
P3	15
P4	19

練習一解答：SFJ

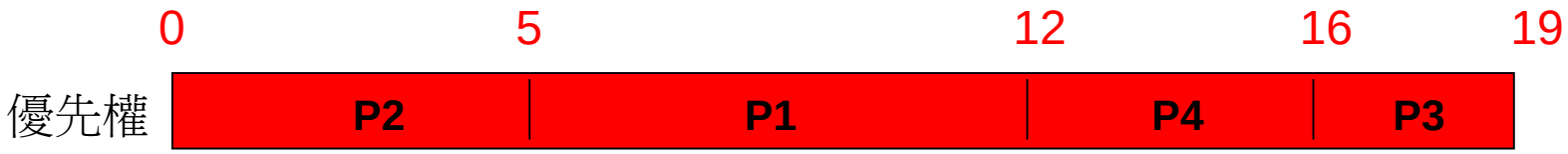


平均等待時間：(0 + 3 + 7 + 12) / 4 = 5.5

回覆時間：

P1	19
P2	12
P3	3
P4	7

練習一解答：優先權



平均等待時間：(0 + 5 + 12 + 16) / 4 = 8.25

回覆時間：

P1	12
P2	5
P3	19
P4	16

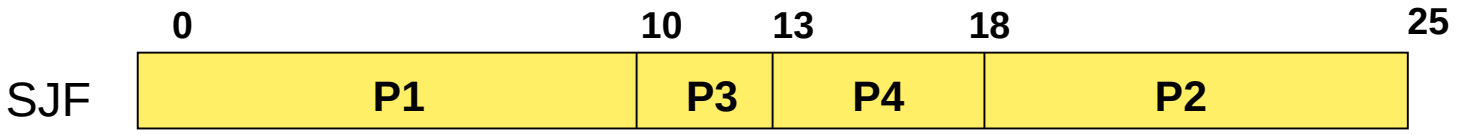
範例補充包-Extra~

課堂練習：排程法（二）

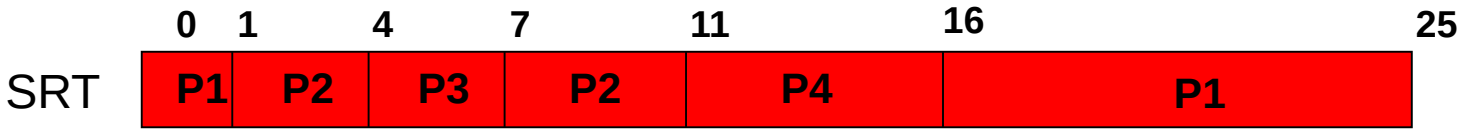
- 假設有四個行程 P1、P2、P3 和 P4，分別在不同時間抵達
- 請針對 SJF和SRT排程法：
  1. 請畫出與課文類似的行程執行順序圖
  2. 計算所有行程的平均等待時間

行程	CPU 暴衝時間（ms）	抵達時間（0的優先權最高）
P1	10	0
P2	7	1
P3	3	4
P4	5	4

練習二解答



平均等待時間：(0 + 7 + 12 + 15) / 4 = 8.5



平均等待時間：(15 + 3 + 0 + 7) / 4 = 6.25

範例補充包-Extra~

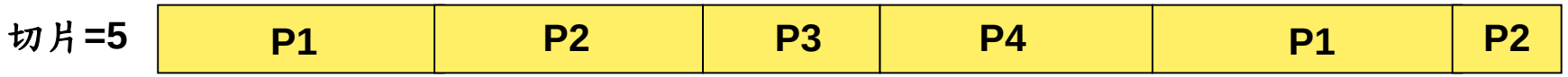
課堂練習-排程法（三）

- 假設有四個行程 P1、P2、P3 和 P4都在時間 0 到達，順序為 P1、P2、P3、P4
- 請針對 時間切片 為2、5、10的RR排程法：
  1. 請畫出與課文類似的行程執行順序圖
  2. 計算所有行程的平均等待時間

行程	CPU 暴衝時間 (ms)
P1	10
P2	7
P3	3
P4	5

練習三解答

平均等待時間：(15 + 10 + 15 + 16) / 4 = 14



平均等待時間：(13 + 18 + 10 + 13) / 4 = 13.5

平均等待時間：(0 + 10 + 17 + 20) / 4 = 11.75