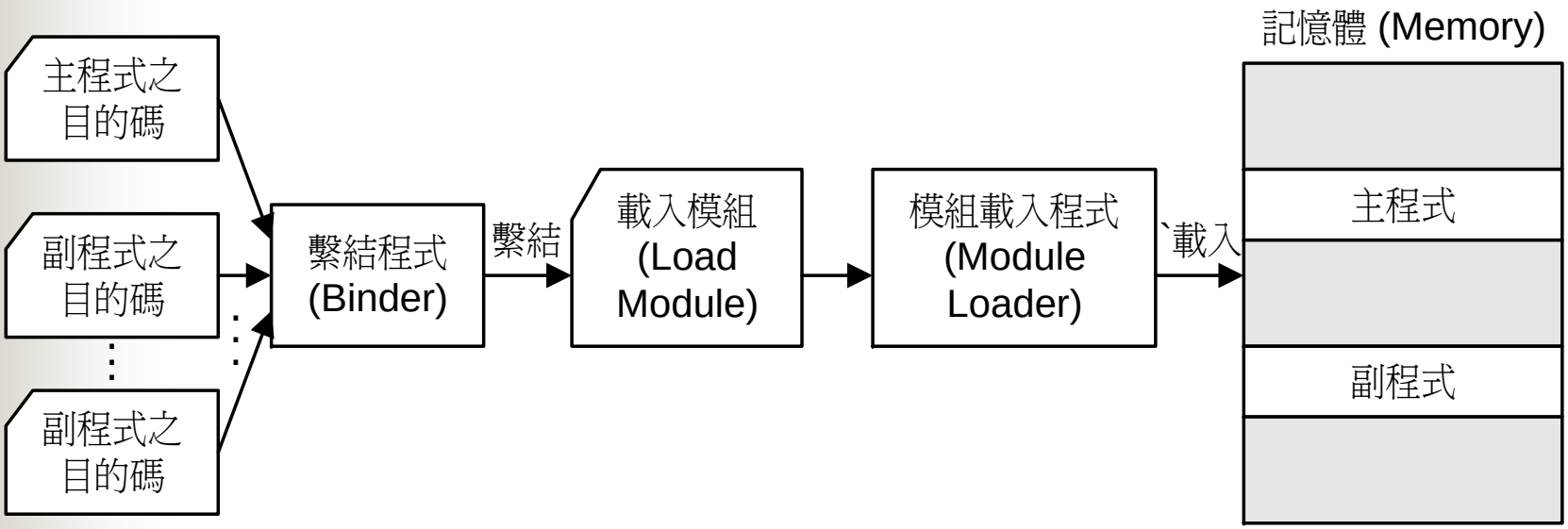
連結程式

繫結程式 (Binder) 之功能 —

- 對主程式與副程式間做位址之重定位，並對程式間做連結，連結後會產生一個載入模組。然後，再將此載入模組存入次儲存體中。綜合上述，繫結程式只負責記憶體配置，重定位及連結三項工作，至於將載入模組載入記憶體中執行的工作，則由模組載入程式另行負責。


主程式之
目的碼
副程式之
目的碼
副程式之
目的碼
繫結程式
(Binder)
繫結
載入模組
(Load
Module)
模組載入程式
(Module
Loader)
、載入
記憶體 (Memory)
主程式
副程式

- 繫結程式之類型 —
  繫結程式一般可分成兩類：
- (1) 磁蕊影像繫結程式 (Core Image Binder)
- (2) 連結編輯程式 (Linkage Editor)

- 磁蕊影像繫結程式 ：

  爲一個能產生絕對位址模組之繫結程式。其記憶體配置的工作在繫結所有副程式時便已完成。然後，模組載入程式便根據磁蕊影像繫結程式所產生之絕對位址，直接將載入模組載入其對應之記憶體位址中。

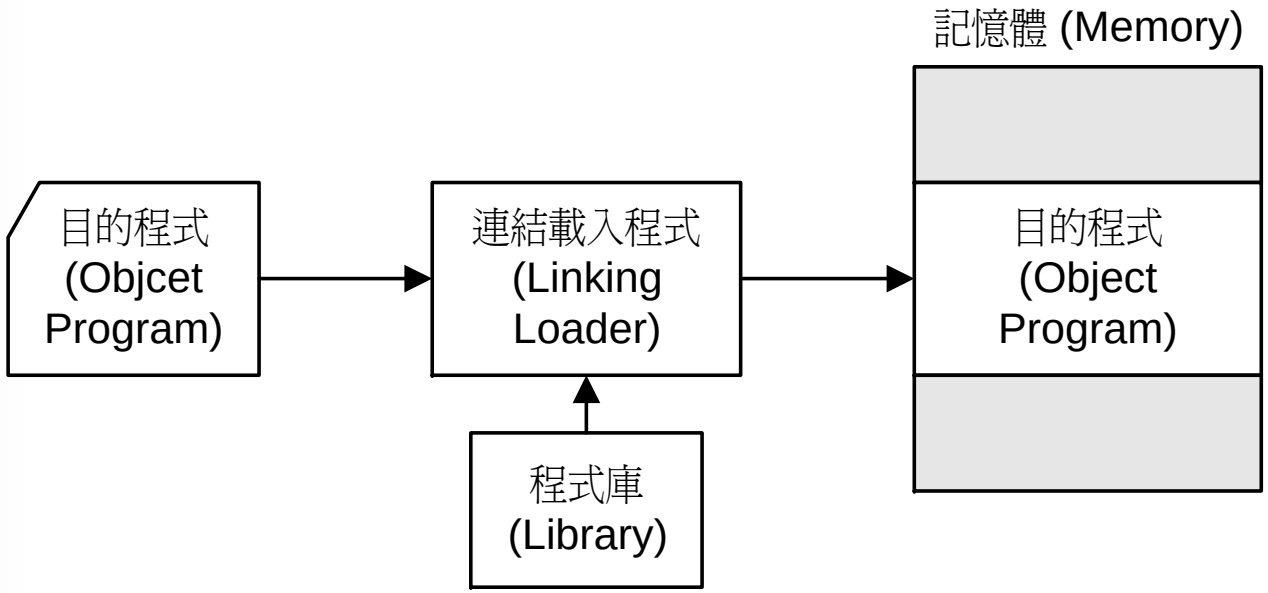
- 連結編輯程式 ：

  爲一個能產生可重定位的相對位址模組。然後，再由模組載入程式進行記憶體配置，重定位及載入的工作。

連結載入程式與連結編輯程式

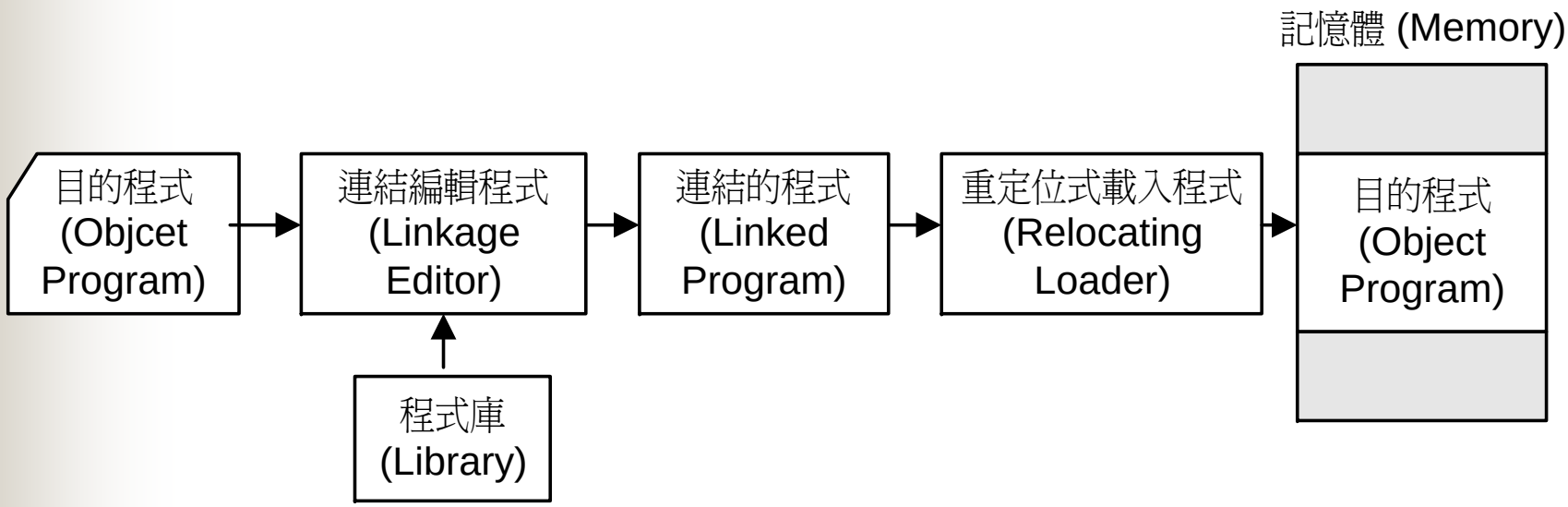
- 連結載入程式 (Linking Loader) —

  對目的程式進行連結與重定位的動作(包括至程式庫中搜尋會使用到的函式，並且將已經完成連結的程式載入記憶體中準備執行)。


記憶體 (Memory)
目的程式
(Objcet
Program)
連結載入程式
(Linking
Loader)
目的程式
(Object
Program)
程式庫
(Library)

- 2. 連結編輯程式 (Linkage Editor) —

  對目的程式進行連結與重定位，並且將已經完成連結的載入模組儲存於次儲存體中。當要執行程式時，再由重定位式載入程式將載入模組由次儲存體中載入主記憶體中執行。


記憶體 (Memory)
目的程式
(Objcet
Program)
連結編輯程式
(Linkage
Editor)
連結的程式
(Linked
Program)
重定位式載入程式
(Relocating
Loader)
目的程式
(Object
Program)
程式庫
(Library)

- 連結載入程式與連結編輯程式之差異

  (1) 連結載入程式：

  外部符號的參考與程式庫的搜尋都僅需要做一次即可。

  (2) 連結編輯程式：

  每次執行程式時，都需要再次處理外部符號的參考與程式庫的搜尋。

重疊技術與動態載入

- 重疊(Overlay) 結構

  重疊的技術是僅把程式執行時要使用到的程式段落由次儲存體中載入主記憶體內，而目前程式在執行時尚不會使用到的程式段落則仍然保留於次儲存體中。所以程式內所有的程式段並不需要同時被存放於主記憶體內，那麼即使實際的記憶體空間較程式所需求之空間小，該程式仍然可以順利的執行。

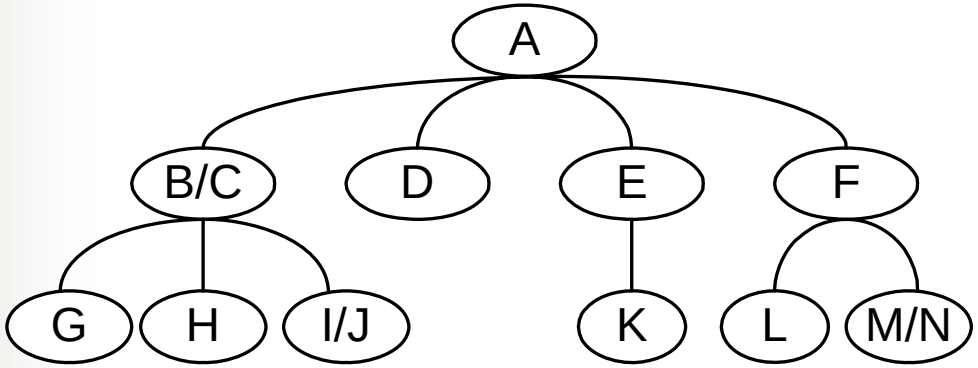
- 與動態載入(Dynamic Loading)

  程式執行時，當成是需要使用時才載入程式段落的方式稱之爲動態載入。

- 重疊技術 (Overlay Technique)

  重疊技巧是利用各個副程式間的互斥性；也就是以其呼叫程式段與被呼叫程式段間的關係，將它們的關係畫成一個樹狀結構。在此一樹狀結構中，在上層的程式段可以呼叫處於其路徑中下一層的程式段。由根部開始至目前所處理的程式段爲止，其路徑上所有的程式段都必須同時存在記憶體中。對屬於同一層的程式段，由於其間沒有順序性與關聯性的關係存在。因此，同一層之程式段不會同時被使用到，因此它們可以於不同時候使用相同之記憶體位址。

範例：下圖中，字母 A～N 分別表示程式段或資料段。



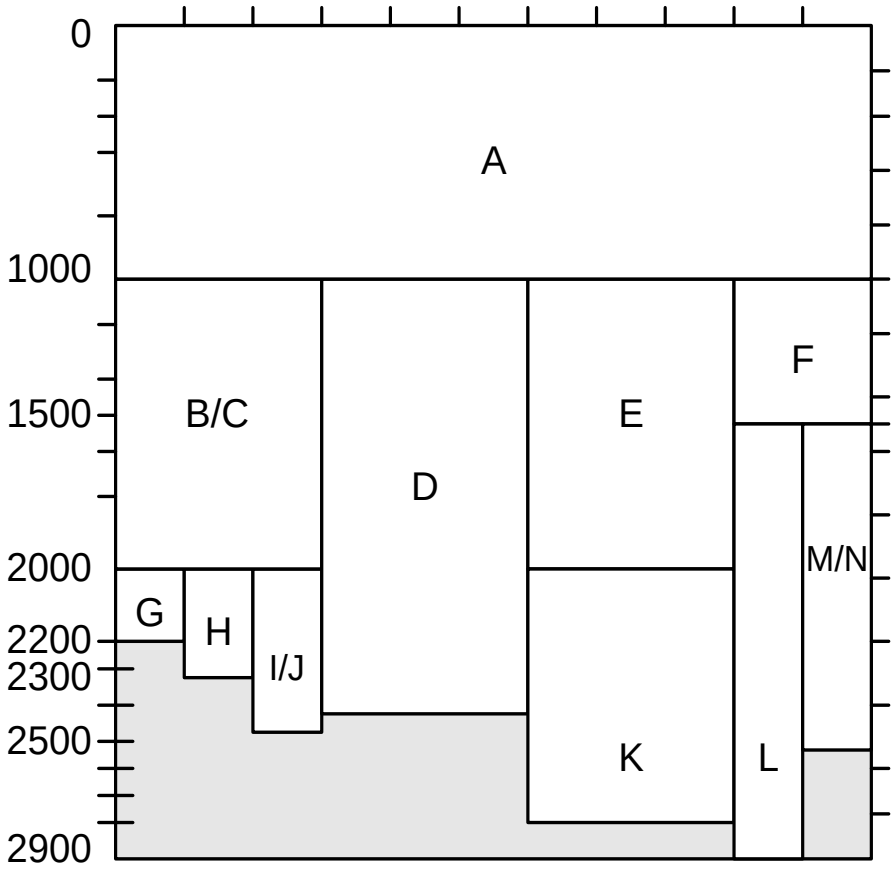
程式段	長度 (Bytes)
A	1200
B	600
C	400
D	1200
E	1000
F	500
G	200
H	300
I	400
J	100
K	600
L	1400
M	700
N	300

- A 是根部，它可以呼叫 B/C、D、E 或 F。
- B/C 可以呼叫 G、H 或 I/J，而不會去呼叫 D、E、F、K、L 或 M/N。
- B/C 代表 B 與 C 雖然是獨立編譯，但是 C 是屬於 B 的資料段，故於重疊處理時，必須視爲一個單位。

- A 是根部，表示程式開始執行時便會載入記憶體中，且一直保留到程式結束，其它程式段(B～N) 則只有在被呼叫時才被載入記憶體中。例如：A 呼叫 B，B 便隨即被載入記憶體中。如果 A 呼叫 B 後又呼叫 D，則 D 也被載入記憶體中，此時，D 可以覆蓋於 B 所佔用的記憶體位置，因爲在同一層的程式段沒有順序性與關聯性，故 B/C、D、E 與 F 四者可任由 A 呼叫。但若呼叫L時，便是因爲 A 呼叫 F，F 再呼叫 L，故此時 A、F 與 L 必須同時存在於記憶體中，而且 L 執行後之結果必須傳回 F，F 執行後之結果必須傳回 A。

- 由此重疊結構可看出最長的路徑是由A(1200)、F(500) 與 L(1400)所組成，故記憶體的最大使用量爲 2900 個位元組(Bytes)。如果不採用重疊的技巧則需將所有程式段同時載入記憶體中，則需要佔用 8900 個位元組。

- 各程式段可能配置的記憶位置



動態連結

- 先前介紹的載入的方式皆有一重大的缺點，即是僅被參考但未被實際執行的副程式仍然需要事先做好連結的工作。而所謂的動態連結 (Dynamic Linking) 則是將連結與載入的動作延遲至程式執行時才處理。

動態連結的優缺：

- 優點：

  (1) 真正使用到的程式才予以連結並載入記憶體中，可以避免記憶體的浪費。

  (2) 程式結構可以做動態性的修改。

  (3) 節省連結的時間。

- 缺點：

  將大部分的繫結工作延遲至程式執行才做，使得在處理程式執行時的工作變得較爲複雜，降低程式執行的效率。