並行計算

EP07: 軟體解法（一之二）

C很容易讓你拿石頭砸自己的腳，
C++是比較不容易些，不過一旦發生了，
你整條腿都會被炸掉。

Bjarne Stroustrup

本集內容

- 上一講（EP06）我們提到使用某個共用變數時得把它的敘述放到一個臨界區中，使得臨界區至少得滿足互斥條件。
- 臨界區有個入口和出口，所以臨界區問題就是如何設計這個入口和出口，使得可以滿足互斥（和其它）條件。
- 本講開始討論不必用到硬體和系統支援的方式的解答。因為這完全只用到程式，我們叫做「軟體」解法 。
- 本講的基本原則是提供一些最基本的、導致正確解答的思維所需要的元素，下一講會給出正確的解法。
- 這些「純」軟體解法在目前系統中是很少使用的，但是討論這些解法卻可以提供給我們更深入的理解。下一講會講解正確的答案。

請看影片的說明，那兒有取得投影片和程式的連結

第一波頭痛

請務必先熟悉上一講（**EP06**）中的名詞和觀念
並且需要了解如何做証明

純軟體解法(只討論兩個Process)

- 我們只討論兩個process的情況；假設它們是P_0 和 P_1 。
- 一般而言，若一個process是 P_i 、另一個是 P_j ，於是i 和 j滿足 $j = 1- i$ 。所以，若 $i = 0$， $j = 1$；若 $i = 1$， $j = 0$ 。
- 我們要為一個臨界區的入口和出口設計能夠滿足三個條件的「好」解法（互斥、有進展、有界等待）。
- 我們會透過一系列不太成功的嚐試、一步一步地走向正確的解法。
- 此地的解法只用軟體，不會用到特殊的系統硬體和軟體，所以這是任何人都可以用得上的，不過它有缺點!

一個非常重要的假設: 1/3

把值存入一個共用變數和測試一個共用變數的值
是兩個無法分割、而且是沒有干擾的作業*

*E. W. Dijkstra, Co-operating sequential processes, in F. Guneys (editor), *Programming Languages*, pp. 43-112, New York, Academic Press, 1968.

一個非常重要的假設: 2/3

- **解釋**
  1. 若兩個proces同時把值存入同一個共用變數時，這兩個儲存的作業會順序地執行，哪個先哪個後無法預知。
  2. 當一個process在檢查一個共用變數的值，而同時另一個process又往該共用變數存入新值，這時檢查該變數值的process既可能看到舊值、也可能會看到新值。
  3. 這個共用變數可能在記憶體、也可能在暫存器中。
  4. 計算一道運算式時卻並非不能分割；也就是說，運算式算到一半可能被打斷，控制權交給另一個process。在第六講開頭我們曾經用不少時間說明這個觀念。

一個非常重要的假設: 3/3

Edsger W. Dijkstra的里程碑文章

Co-operating Sequential Processes

E. W. DIJKSTRA

Department of Mathematics, Technological University, Eindhoven, The Netherlands

INTRODUCTION

This chapter is intended for all those who expect that in their future activities they will become seriously involved in the problems that arise in either the design or the more advanced applications of digital information processing equipment; they are further intended for all those who are just interested in information processing.

The applications are those in which the activity of a computer must include the proper reaction to a possibly great variety of messages that can be sent to it at unpredictable moments, a situation which occurs in process control, traffic control, stock control, banking applications, automatization of information flow in large organizations, centralized computer service, and,

Edsger Wybe Dijkstra
（1930年5月11日 — 2002年8月6日）

再加幾個較次要的假設

- 以下是幾個對參與競爭的process的行為的假設
  1. 在臨界區之外各process除了不能干涉其它process之外，就沒有其它假設。
  2. 在臨界區之外不可以使用在入口和出口中用到的共用變數。
  3. 在執行入口、臨界區、和出口的敘述時不可以無限循環、也不可以出錯。只要被安排執行，就一定得出動。
  4. 一個process在臨界區和出口內只能執行有限個步驟，不可以無限循環。
  5. 固然參與競爭的process是並行執行，但每一個process在執行時都是順序的。

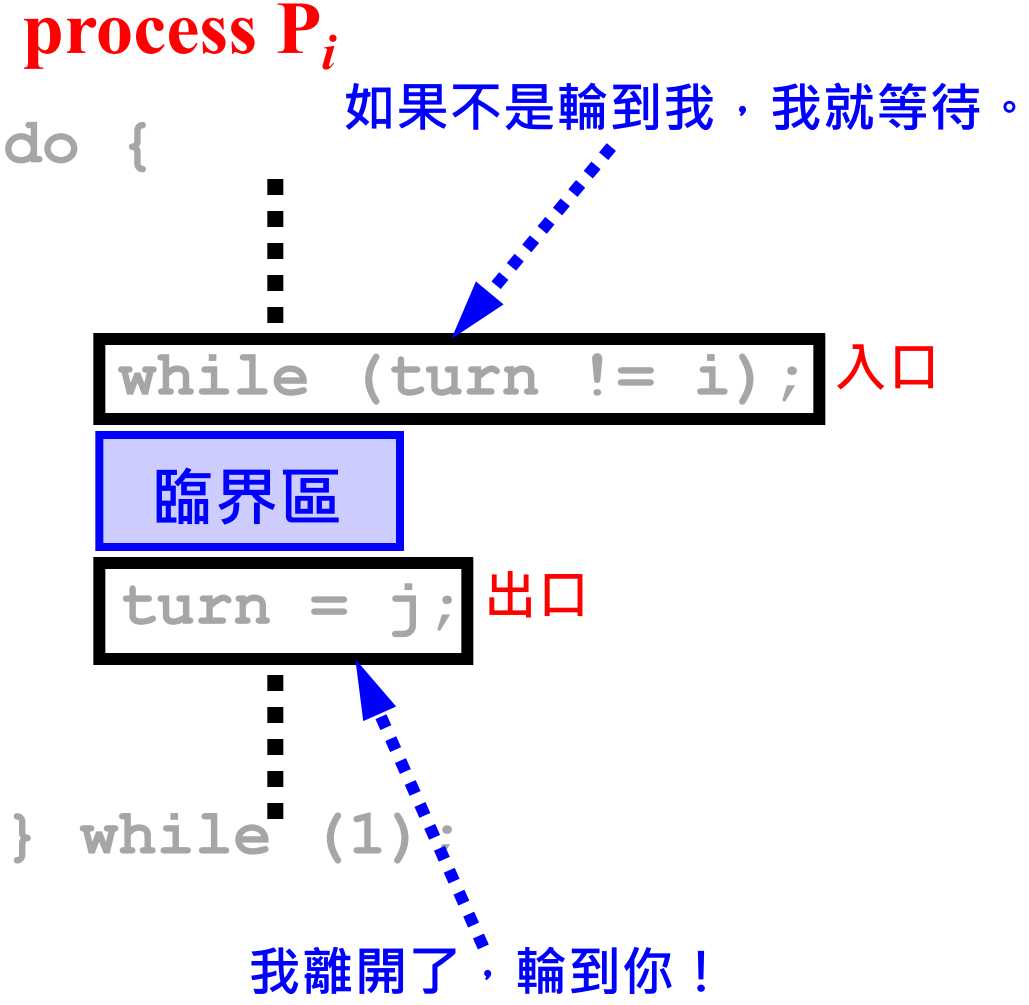
往後的討論方式

- 接下來的幾個嚐試會逐步地發展出一個好解法，每一次嚐試都會引入一個新想法，後面的嚐試使用前面的若干想法而得到新的結果，直到我們能導出一個正確的解法為止。
- 對每一個嚐試，我們都會驗証互斥、有進展、有界等待三個要件。
- 證明互斥的好方式是「反證」，步驟通常如下：
  1. 找出P_i能進入臨界區的條件C_i，找出P_j能進入臨界區的條件C_j；
  2. 若P_i和P_j都在臨界區內，則條件C_i和C_j都為真；
  3. 檢查在此情況下用來控制誰可以進入臨界區的變數；
  4. 其中若干變數必定會得到矛盾的值，所以原假設（ P_i和P_j 都在臨界區內）為假，因此它們不能同時在臨界區內。

嚐試一

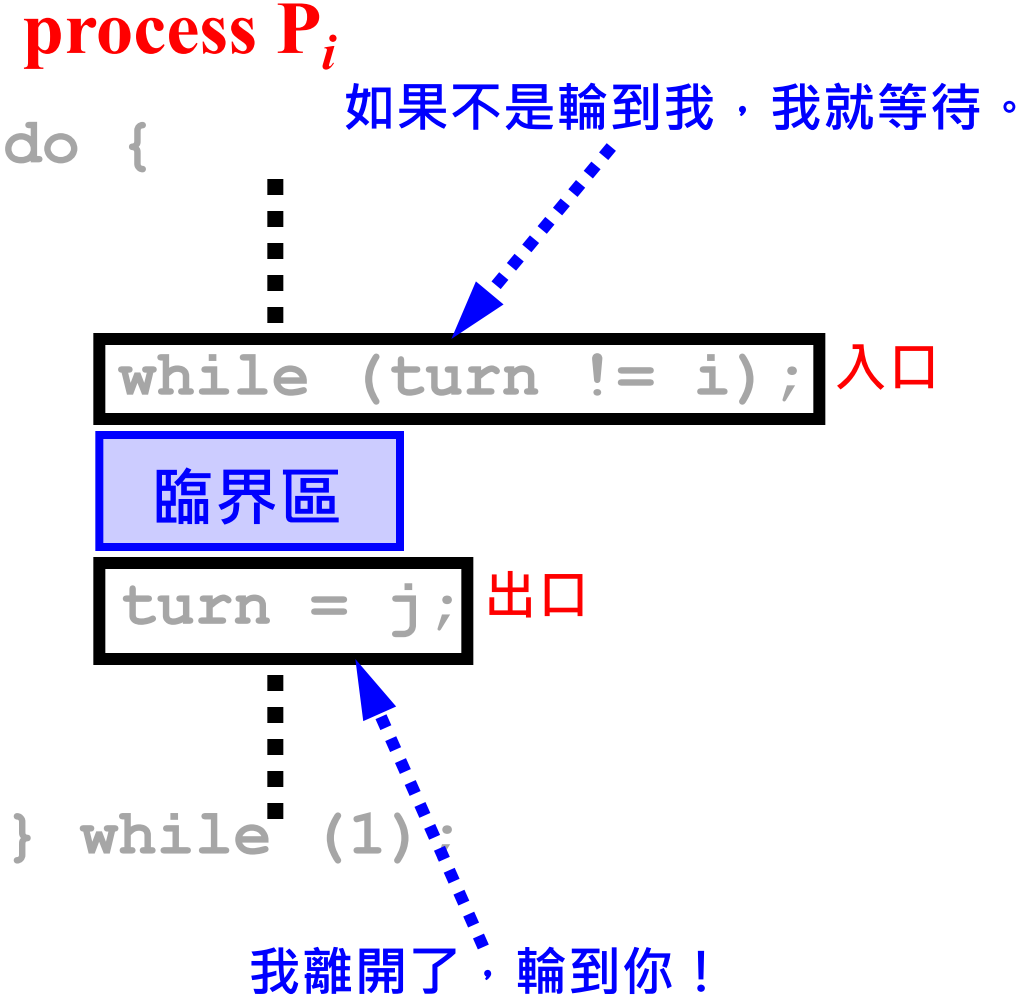
用一個變數指出誰可以進入臨界區

嚐試一: 1/6



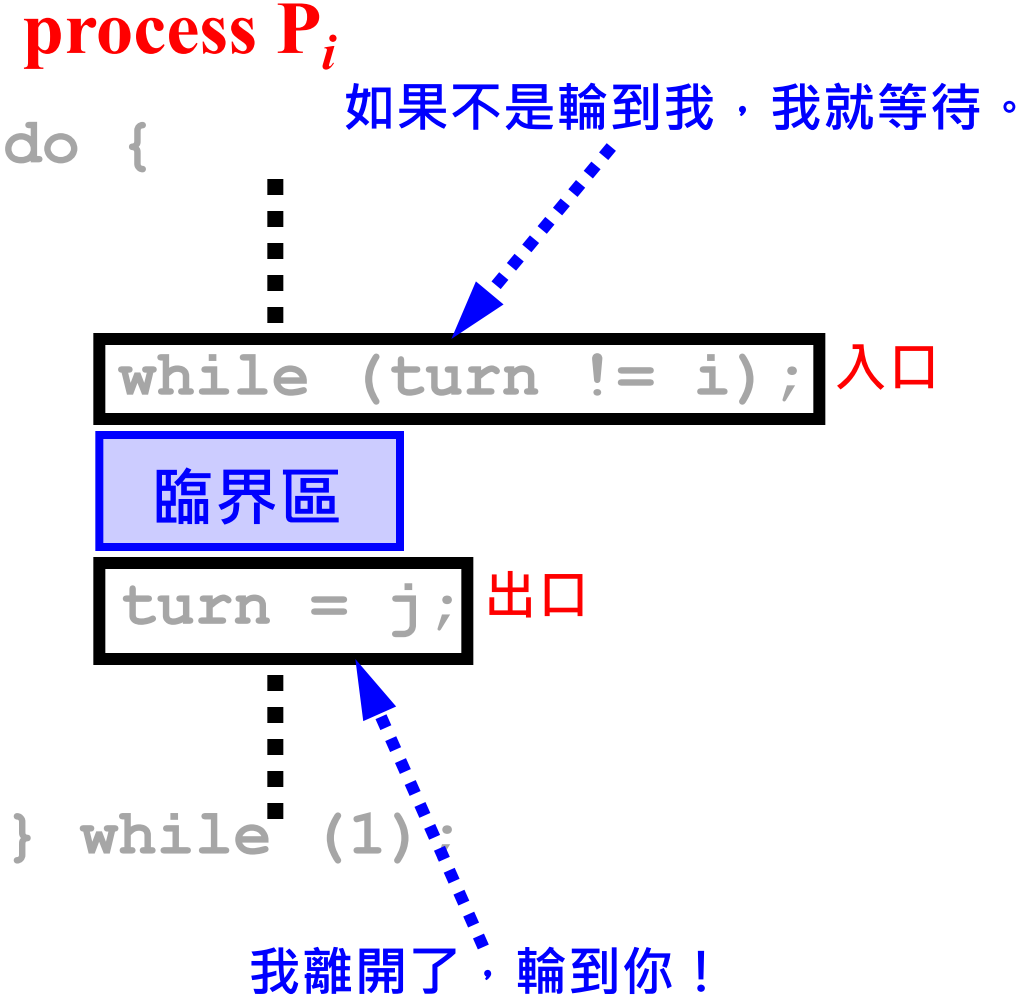
- 假設共用變數 `turn` 的初值為 `i` 或`j`，這是用來控制誰可以進入臨界區。
- 因為 `turn` 的值不是 `i` 就是 `j`，所以只會有一個**process**可以進入。
- 不過，誰可以進入卻有固定方式：P_i、P_j、P_i、P_j、…等等。
- 這不是個好解法。

嚐試一: 2/6



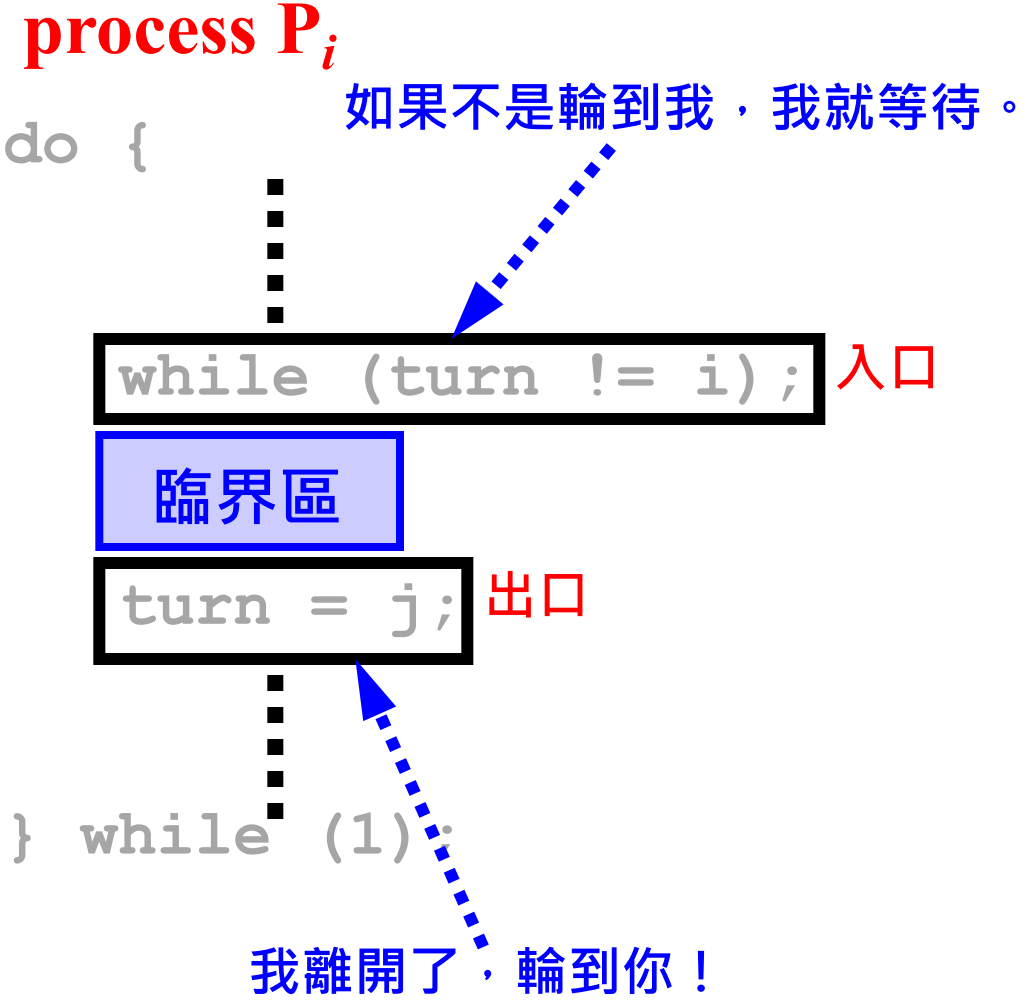
- 互斥
- 若turn = i，P_i進入。
- 若turn = j，P_j進入。
- 若 P_i和 P_j都在臨界區，於是turn = i 和 turn = j 同時成立。
- 這不可能，因為變數只能存一個值，turn 不可能同時有 i 和 j。所以，P_i和 P_j不可能同時在臨界區內。

嚐試一: 3/6



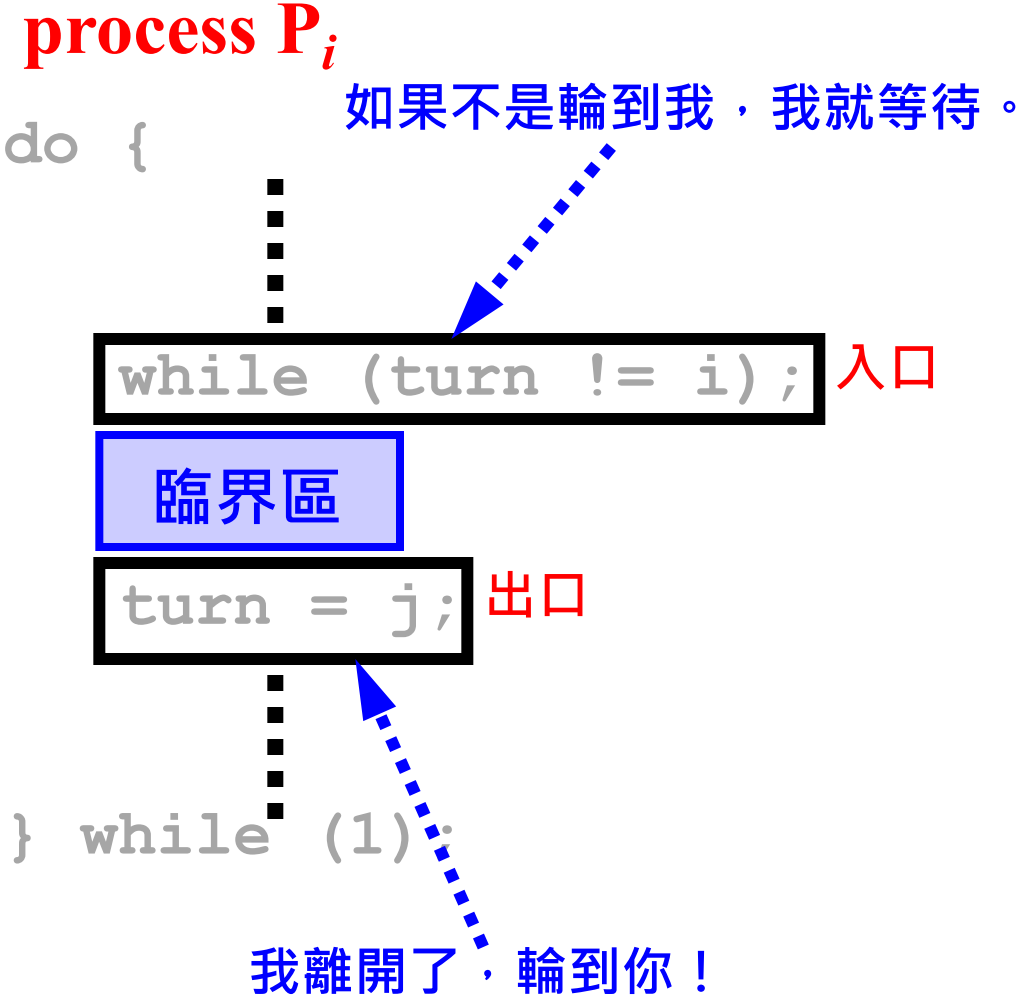
- **有進展: 1/2**
- 若while在有限時間內發現turn!=i為假，P_i進入。
- 若P_i和P_j都到達while，於是一次比較就知道turn是i還是j，因此有限時間內就可以挑出P_i或P_j。

嚐試一: 4/6



- **有進展: 2/2**
- 但若P_j離開時把turn定成i而且從此就不再使用該臨界區，於是P_i到達時因為turn != i為假，P_i進入。
- P_i離開時把turn定成j，然而因為P_j不再使用該臨界區，從而turn就沒有任何機會被定回i。因此，當P_i再次到達入口而臨界區又是空的時候，P_i永遠無法進入，所以有限決策時間不成立。
- 不但如此，P_j（不參加等待）影響了決策。所以有進展不成立。

嚐試一: 5/6



- **有界等待**
- 請回想前幾頁的說明，這個解法強制地建立了一個固定的順序。
- 當P_i和P_j都在等待，若turn是i，P_i立即進入；若turn為j，P_j進入。P_j離開時會把turn定成i，於是P_i進入。
- 是故，P_i最多等一次就可以進入。
- 要注意的是，若P_j離開後就不再進入臨界區，於是P_i最多就進入一次，往後就卡在入口。
- 然而，這是有進展不成立，而不是有界等待不成立，因為在P_i進入之前「沒有」其它process進入！

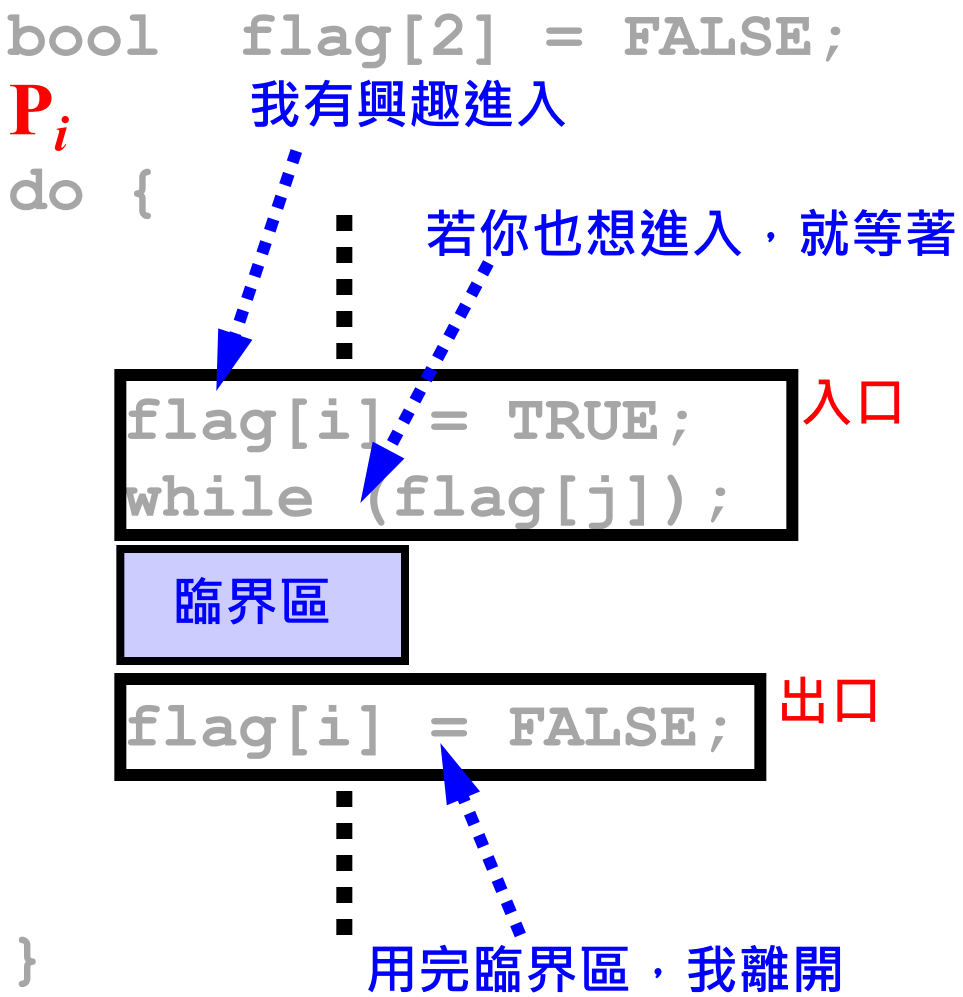
嚐試一: 6/6

- 我們學到什麼？
  1. 這個簡單的解法做到了互斥和有界等待，但不滿足有進展。
  2. 可以用一個變數指明誰可以進入。
  3. 然而，一個process離開臨界區時，在出口處得透過turn的值指定下一個可以進入的process。
  4. 如果給turn的是基於某道式子算出來（譬如都是下一個），那麼若被指定的那個process不再使用臨界區時，就失去了由它指定的下一個process，有進展的條件就十分有可能失敗。
  5. 結論：除了turn之外，還得有其它機制使上述情況不會發生。

嚐試二

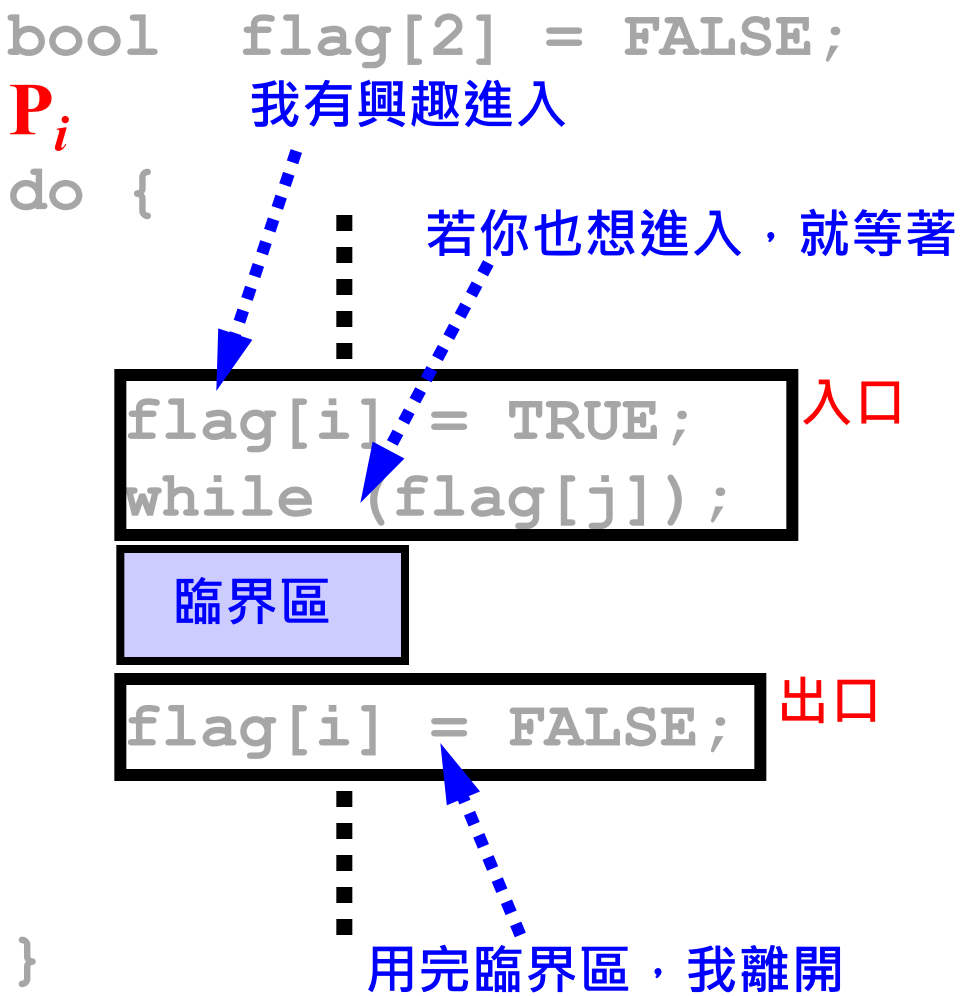
用一個邏輯變數表示誰在臨界區中

嚐試二: 1/5



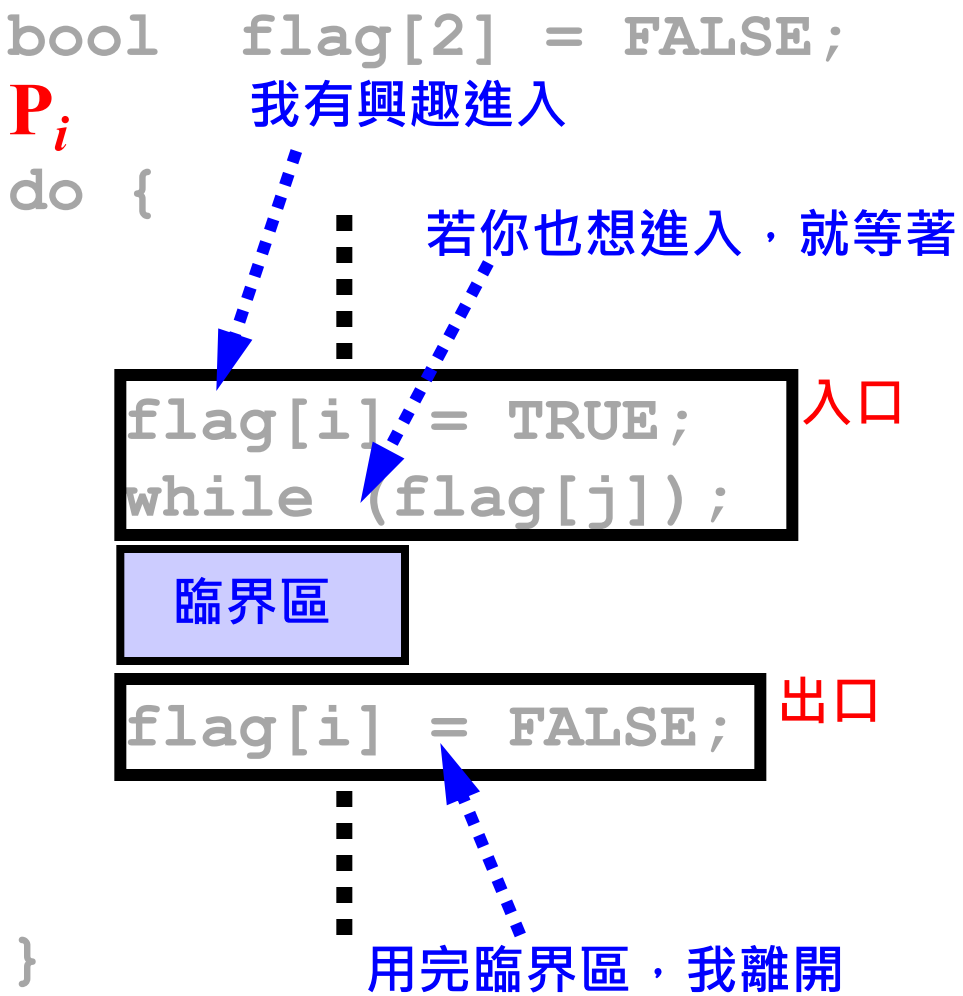
- 邏輯變數flag[i]表示P_i對臨界區的使用狀態。
- flag[i]為TRUE時，表示P_i有「興趣」或「已經」進入臨界區； flag[i]為FALSE表示P_i不感興趣（譬如已經離開臨界區）。
- P_i先把flag[i]定成TRUE，表示想進入臨界區；若P_j也有興趣，P_i等待。
- 到P_j不感興趣之後，P_i進入臨界區。
- 在離開臨界區時，P_i把flag[i]定成FALSE，表示自己離開了。

嚐試二: 2/5



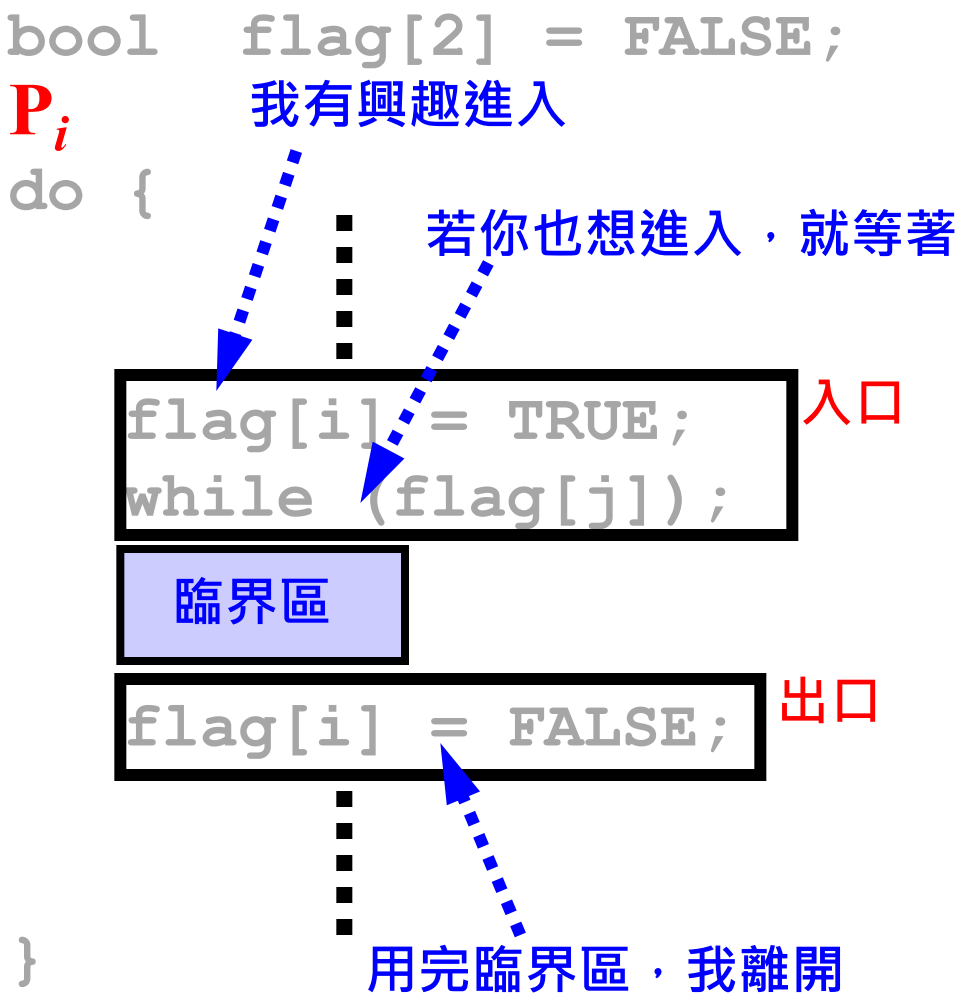
- 互斥
- P_i 在臨界區中的條件是：flag[i]為TRUE而flag[j]為FALSE。
- P_j 在臨界區中的條件是：flag[j]為TRUE而flag[i]為FALSE。
- 若P_i和P_j同時在臨界區中，flag[i]和flag[j]同時為TRUE和FALSE。但這不可能，因為flag[i]和flag[j]只能有一個值，不能同時有這兩個值。
- 因此， P_i和P_j不可能同時在臨界區中；因此互斥成立。

嚐試二: 3/5



- **有進展**
- 若P_i和P_j同時把flag[i]和flag[j]定成TRUE，兩者都在while進入無限循環，因為flag[i]和flag[j]都是TRUE。
- 當P_i和P_j都同時在入口處的while進入無限循環，無法在有限時間內選擇可以進入臨界區的**process**，所以不滿足有限決策時間。
- 因此，無死鎖的條件（當然有限決策時間）就失敗了。

嚐試二: 4/5



- **有界等待**
- 假設P_i和P_j同時到達入口，但P_i在執行`flag[i] = TRUE`之前被暫停。
- P_j卻通行無阻（`flag[i]`為`FALSE`）、進入臨界區、把`flag[j]`定成`FALSE`。然而P_j卻是快腿，立即回到入口打算進入臨界區。
- 若P_i還是在暫停狀態，P_j可以進入。於是，在P_i可以進入臨界區之前P_j可以進入很多次。於是有界等待不成立!

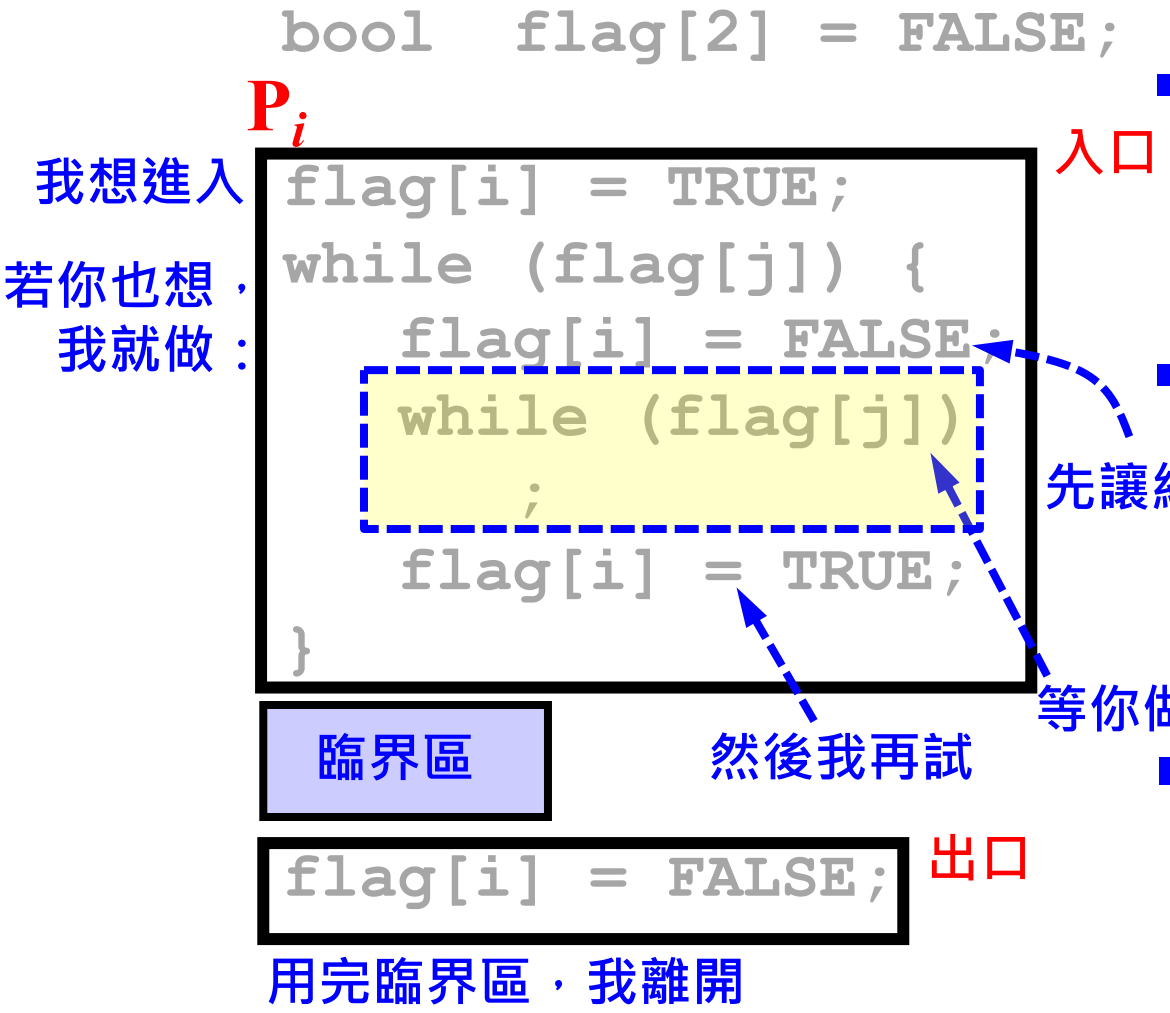
嚐試二: 5/5

- 我們學到什麼？
- 固然我們可以用一個邏輯變數記錄誰在臨界區中，但也有可能參與競爭的兩個process完全同步、同時表達想進入臨界區的志願而同時等待對方不想進入的設想，於是我們會到達兩個process可能都想進入等待對方不想進入的願景，於是進入無限循環狀態。
- 結論：能否有一個比較好、比較細膩的做法呢？請看嚐試三。

嚐試三

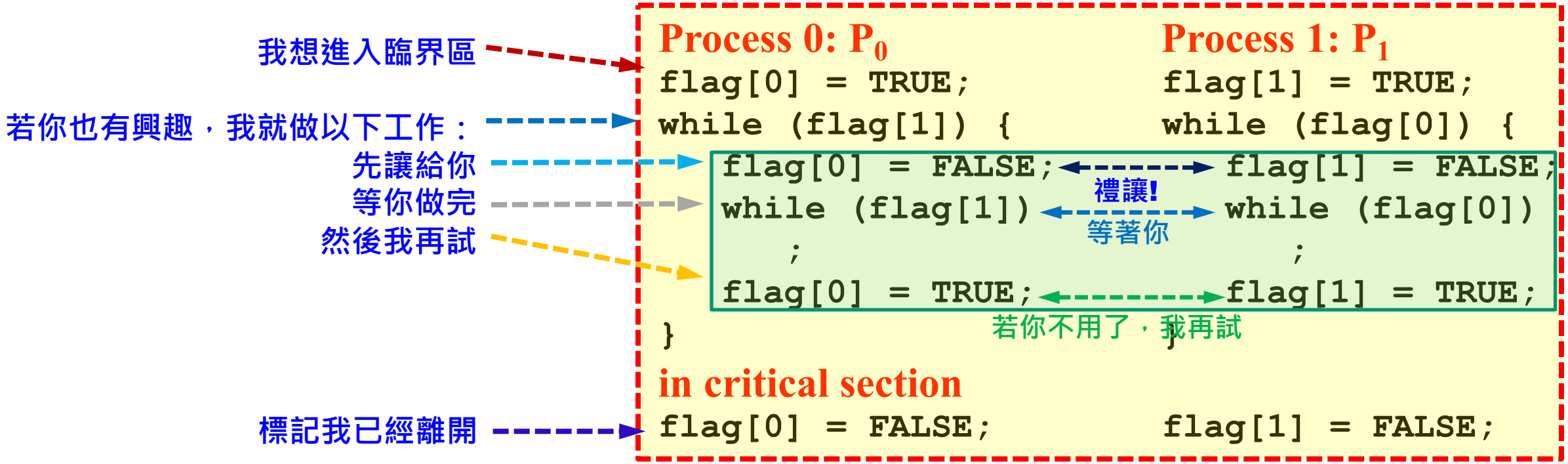
在等待時退一步、讓一讓如何？

嚐試三: 1/10

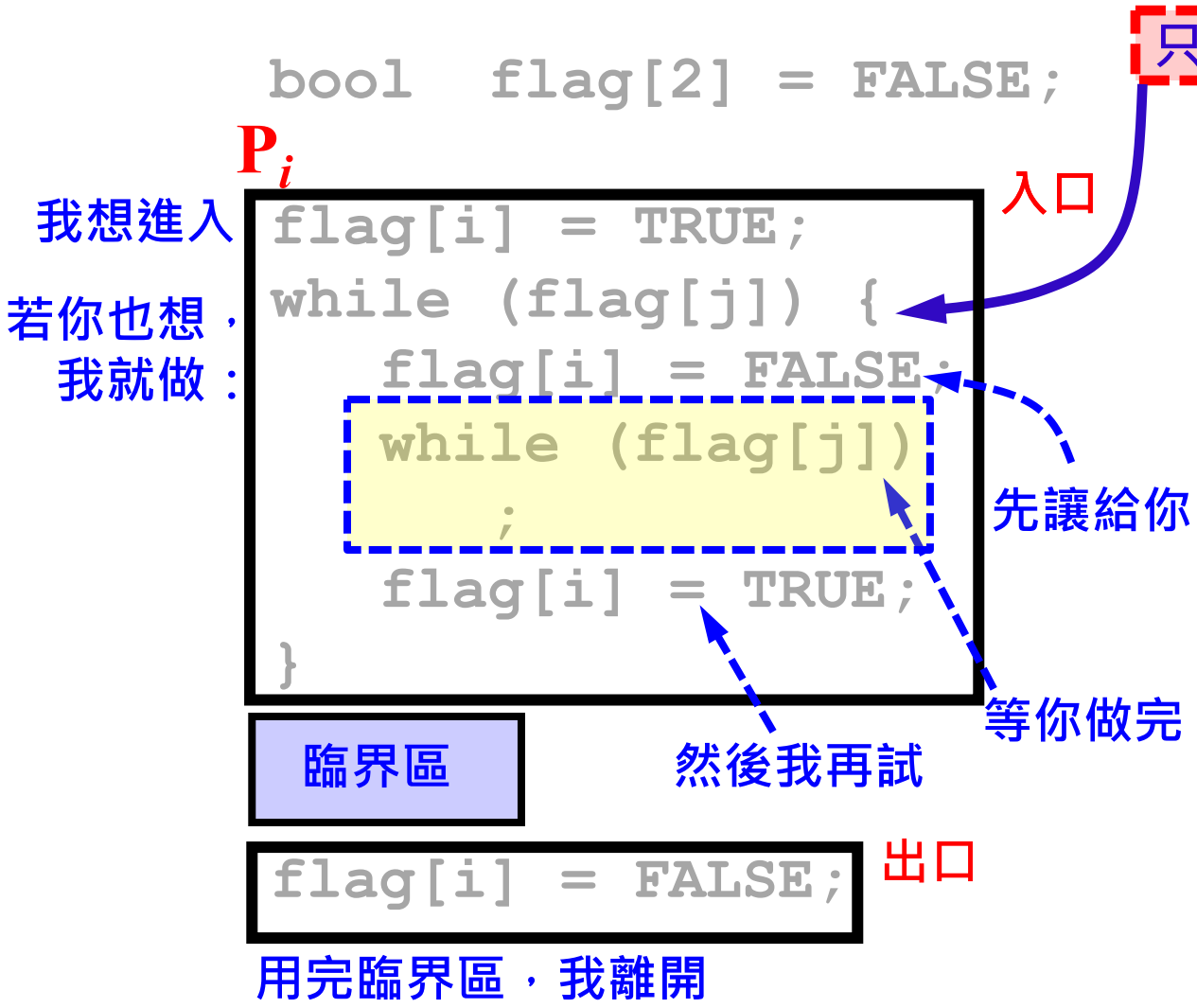


- 在嚐試二中，兩個process各不相讓、一直等到對方不感興趣為止。這種各不相讓的方式造成誰也進不了臨界區。
- 於是，我們嚐試：一旦知道對方有興趣，己方就退讓、直到對方沒有興趣為止，然後再表達己方有興趣、再回頭測試對方是否沒有興趣。
- 如果對方沒有興趣，就進入臨界區！

嚐試三: 2/10

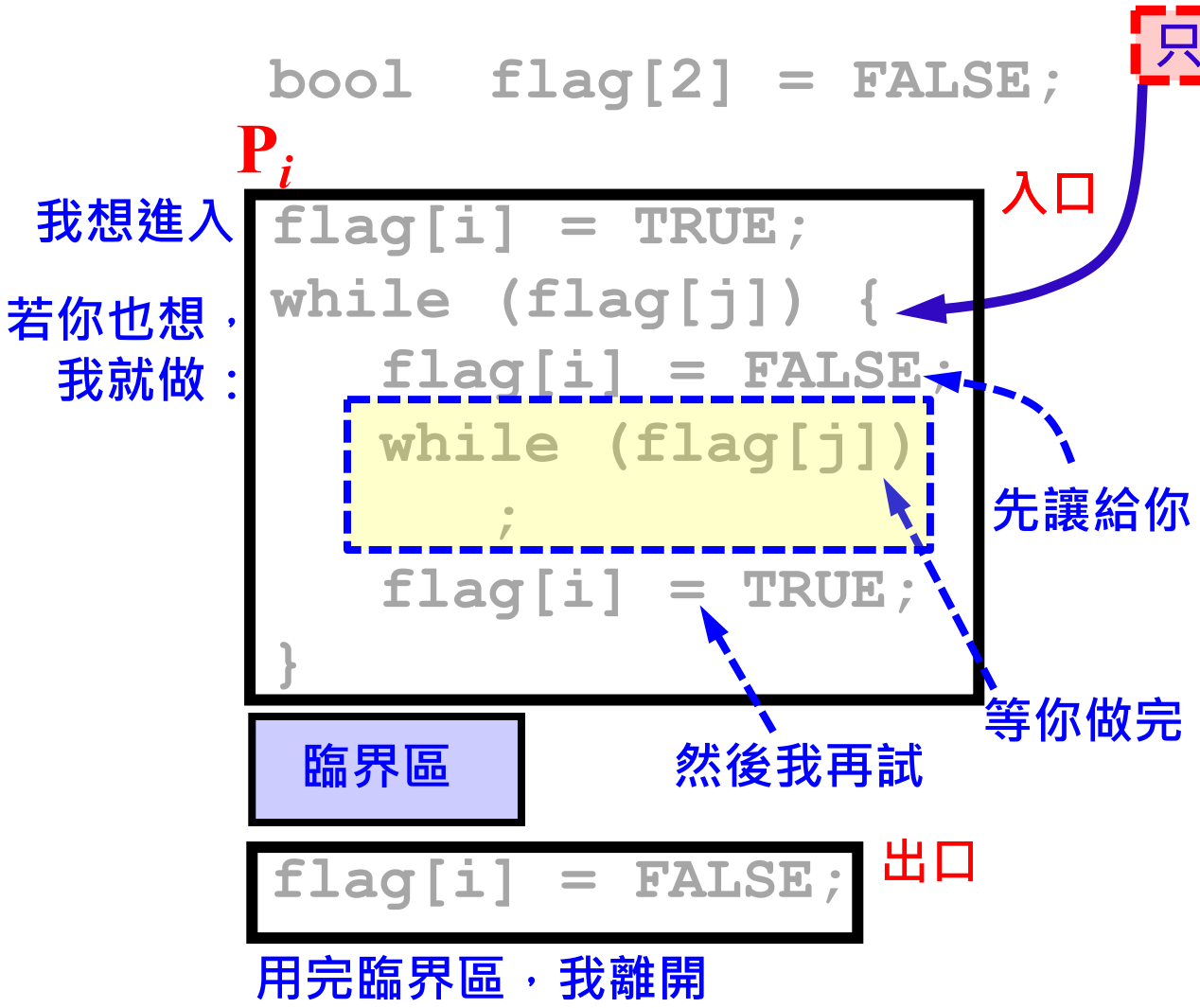

我想進入臨界區
若你也有興趣，我就做以下工作：
先讓給你
等你做完
然後我再試
標記我已經離開
Process 0: P0
Process 1: P1
flag[0] = TRUE;
flag[1] = TRUE;
while (flag[1]) {
while (flag[0]) {
flag[0] = FALSE;
flag[1] = FALSE;
禮讓!
while (flag[1])
while (flag[0])
等著你
;
;
flag[0] = TRUE;
flag[1] = TRUE;
若你不用了，我再試
}
}
in critical section
flag[0] = FALSE;
flag[1] = FALSE;

嚐試三: 3/10



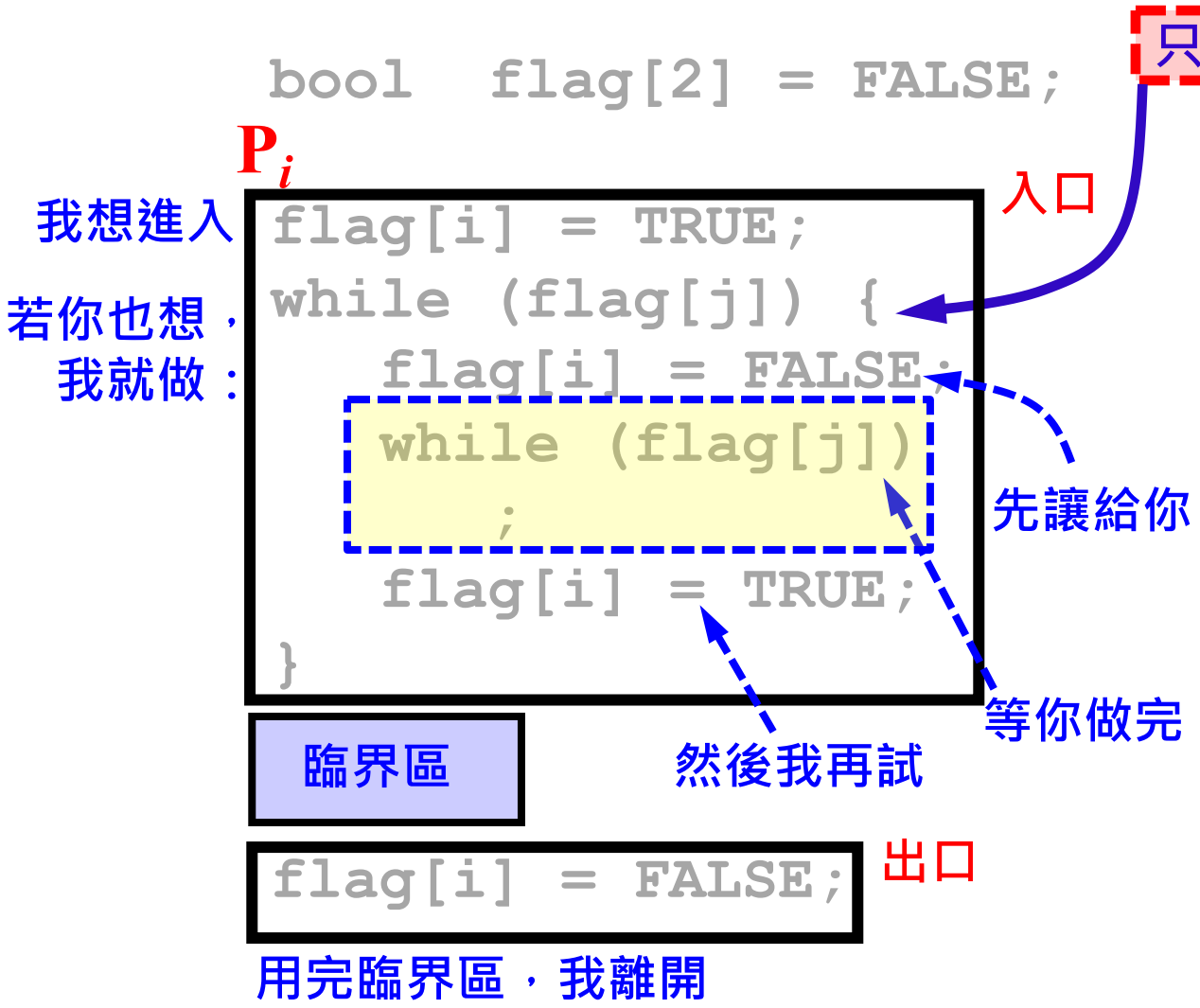
- **互斥**
  - ✓ 到達while之前，flag[i]都會被定成TRUE 。一旦進入臨界區，flag[i]為TRUE而flag[j]為FALSE 。
  - ✓ 若P_i在臨界區，則flag[i]為TRUE，flag[j]為FALSE 。
  - ✓ 若P_j在臨界區，則flag[j]為TRUE，flag[i]為FALSE 。
  - ✓ 若P_i和P_j同時在臨界區中，則flag[i]（和flag[j]）同時為TRUE也為FALSE。 **這是不可能的！**
  - ✓ 所以P_i和P_j不會同時在臨界區中。

嚐試三: 4/10



- **有進展: 1/2**
- 沒在等待的process不會影響決策
  - ✓ 假設P_i等待進入，而P_j在它處執行
  - ✓ 若P_j在它處，則P_j從未進入過或已經離開臨界區，flag[j]為FALSE。
  - ✓ 當P_i到達入口時會發現while處的條件為FALSE，於是立刻可以進入。
  - ✓ 因此，沒在等待的process不會影響決策。

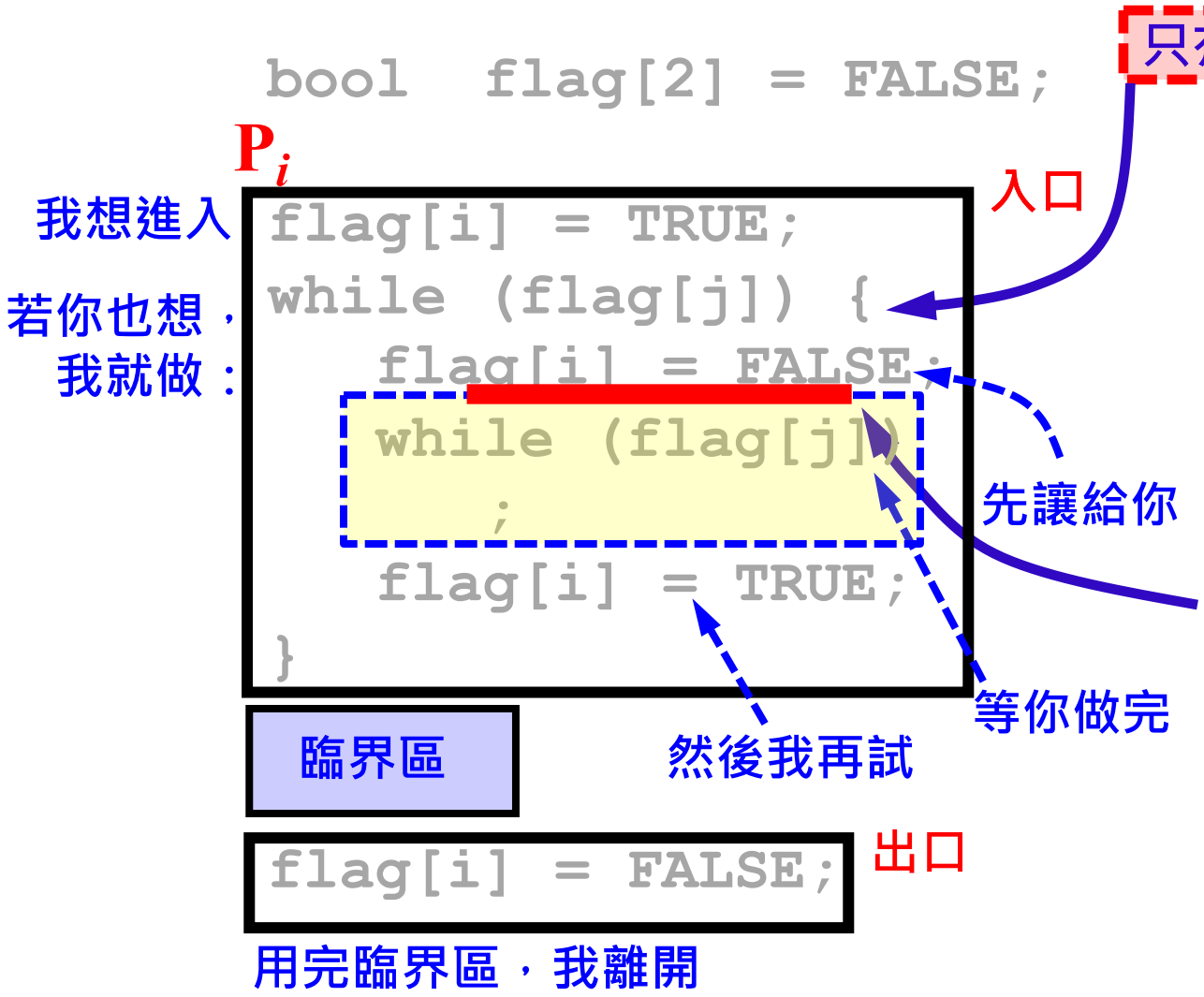
嚐試三: 5/10



只有離開這個while才能進入臨界區

- **有進展: 2/2**
- 有限決策時間
  - ✓ 若兩個process完全同步、同時間執行同一道敘述，於是：
    1. 兩者都想進入，從而到達外while。
    2. 兩者都發現對方想進入，因此把自己的flag[ ]定成FALSE，進入內while。
    3. 因為對方的flag[ ]為FALSE，兩者都同時離開while、把自己的flag[ ]定成TRUE，進入下一個循環。
    4. 這可以一再重複而演變成無限循環，無法在有限時間內挑選一個process。
    5. 因此，有限決策時間不成立

嚐試三: 6/10



- 有界等待
- P_i把flag[i]定成FALSE之後，P_j有機會跳出外while進入臨界區。
- 但若P_i才把flag[i]定成FALSE之後就被暫停，這給P_j一個反覆地進入臨界區的機會。
- P_j能進入多少次，就看P_i會被暫停多久而定。請看下幾頁。
- 因此，有界等待不成立。

嚐試三: 7/10

有界等待不成立

P_0

```
flag[0] = TRUE;
while (flag[1]) {
    flag[0] = FALSE;
    while (flag[1])
        ;
    flag[0] = TRUE;
}
```

被暫停

P_1

```
flag[1] = TRUE;
while (flag[0]) {
    flag[1] = FALSE;
    while (flag[0])
        ;
    flag[1] = TRUE;
}
```

繼續

可以反覆進入

...臨界區...

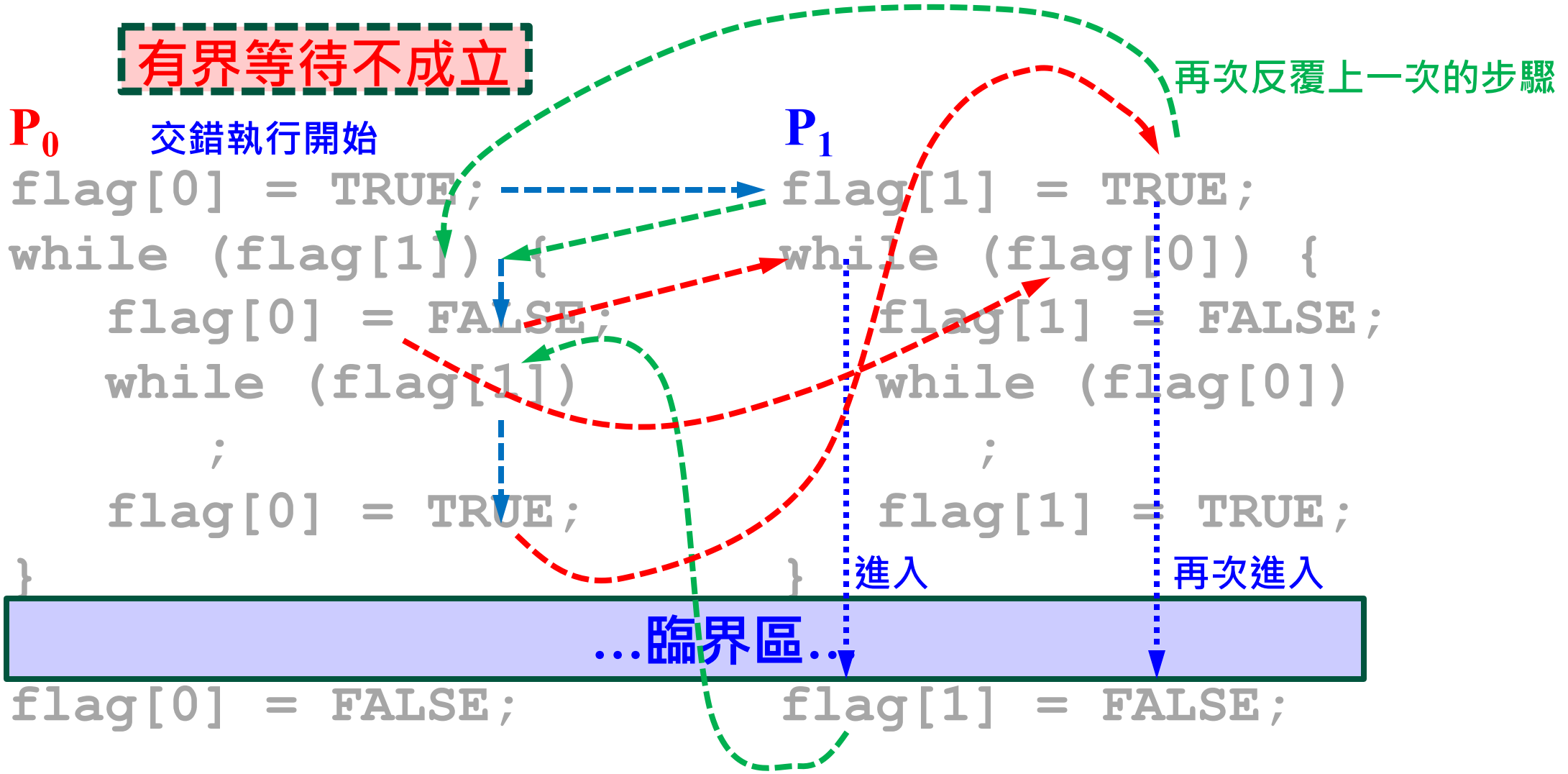
```
flag[0] = FALSE;
```

```
flag[1] = FALSE;
```

嚐試三: 8/10

比較復雜的例子

有界等待不成立



嚐試三: 9/10

為了省空間，我們儘可能地讓兩個process部份平行處理

用執行序列說明比較清楚

```
1.  flag[i] = TRUE;
2.  while (flag[j]) {
3.    flag[i] = FALSE;
4.    while (flag[j])
5.      ;
6.    flag[i] = TRUE;
7.  }
      臨界區
8.  flag[i] = FALSE;
```

P_1 進入兩次
並且可以一再進入

P_0	P_1	flag[0]	flag[1]	說明
兩者同時開始		F	F	
f[0] = T	f[1] = T	T	T	
while(f[1])		T	T	P_0 的外 while
f[0] = F	while(f[0])	F	T	P_1 的外 while
while(f[1])	P_1 進入 CS	F	T	P_0 的內 while
	P_1 離開CS	F	T	
	f[1] = F	F	F	P_1 重設 f[1]
f[0] = T		T	F	P_0 到達第6列
	f[1] = T	T	T	P_1 回頭有興趣進入
f[0] = F		F	T	P_0 進入下一個循環
	while(f[0])	F	T	P_1的外 while
while(f[1])	P_1 進入CS	F	T	P_0 的內while
	P_1 離開CS	F	T	
	f[1] = F	F	F	P_1重設f[1]
f[0] = T		T	F	P_0到達第6列
	f[1] = T	T	T	P_1回頭有興趣進入

嚐試三: 10/10

- 我們學到什麼？
- 只有嚐試一做到有界等待，但卻是透過固定順序完成。
- 嚐試二和嚐試三克服了不等待process不干預決策，但卻做不到有限決策時間。
- 嚐試三期望通過「禮讓」方式避免同時齊一步伐的競爭，從而使有限決策時間的失敗；然而，不正確的「禮讓」也可能使自已完全失去進入臨界區的機會。
- 從嚐試一的turn（指定下一個可以進入的）變數來實現「禮讓」、用flag[ ]表示誰在（或不在）臨界區內的比較細膩分工是否會比較好呢？請看下回分解。

綜合講評

- 在只有兩個process的前提下，反証法是証明互斥條件的最簡明扼要的辦法。
- 證明在「任何」執行序列下都會滿足互斥是個全稱命辭（universal proposition），切記「不可以」列出幾個滿足互斥的執行序列做為互斥的証明（這叫做用例子做證明）。
- 反之，證明不滿足某個條件時，用一個反例就足夠。
- 下一講會用本講提到的觀念討論「好」的解法，包含有歷史上第一個正確解答、和到目前為止最「簡單」的解答。

想一想

幾道練習題

習題一

```
bool  flag[2] = FALSE;
```

process P_i

```
while (flag[j])
    ;
flag[0] = TRUE;
```

入口

臨界區

```
flag[i] = FALSE;
```

出口

- 這是個簡化的嚐試二。
- 請就互斥、有進展、有界等待三點分析這個解法。

習題二

```
bool   flag[2] = FALSE;
int    turn = i 或 j;
```

process P_i

```
flag[i] = TRUE;
while (turn != i) {
    while (flag[j])
        ;
    turn = i;
}
```

入口

臨界區

```
flag[i] = FALSE;
```

出口

- 這是在**1966**年時發表的解法*，它把`turn`和`flag[ ]`合併使用。
- 請用一個執行序列證明這個解法不滿足互斥條件。正因為如此，有進展和有界等待就沒多大意義了。

*Harris Hyman, Comments on a Problem in Concurrent Programming Control, *Communications of the ACM*, Vol. 9 (1966), No. 1 (January), p.45

我們學到了什麼?

- 我們講解完全不用硬體和系統支援如何解決臨界區問題的技巧，但本講只限制在兩個process的特殊情況，因為超過兩個process就會變得很複雜。
- 本講主要的內容是使用一個共用變數turn指出誰可以進入臨界區，以及一個邏輯陣列flag[ ]記錄下誰沒有興趣進入、又誰有興趣進入（或根本就在臨界區內）。
- 我們用三個嚐試，每一個都用到上述變數，並且分析它們是否滿足互斥、有進展、和有界等待這三個條件。
- 三個嚐試都不能做到完全滿足所有條件，但都可以做到互斥。
- 本講的內容為下一講的正確解法舖平道路。

結束，謝謝收看！
期望您再次觀看下一集

請看影片的說明，那兒有取得投影片和程式的連結