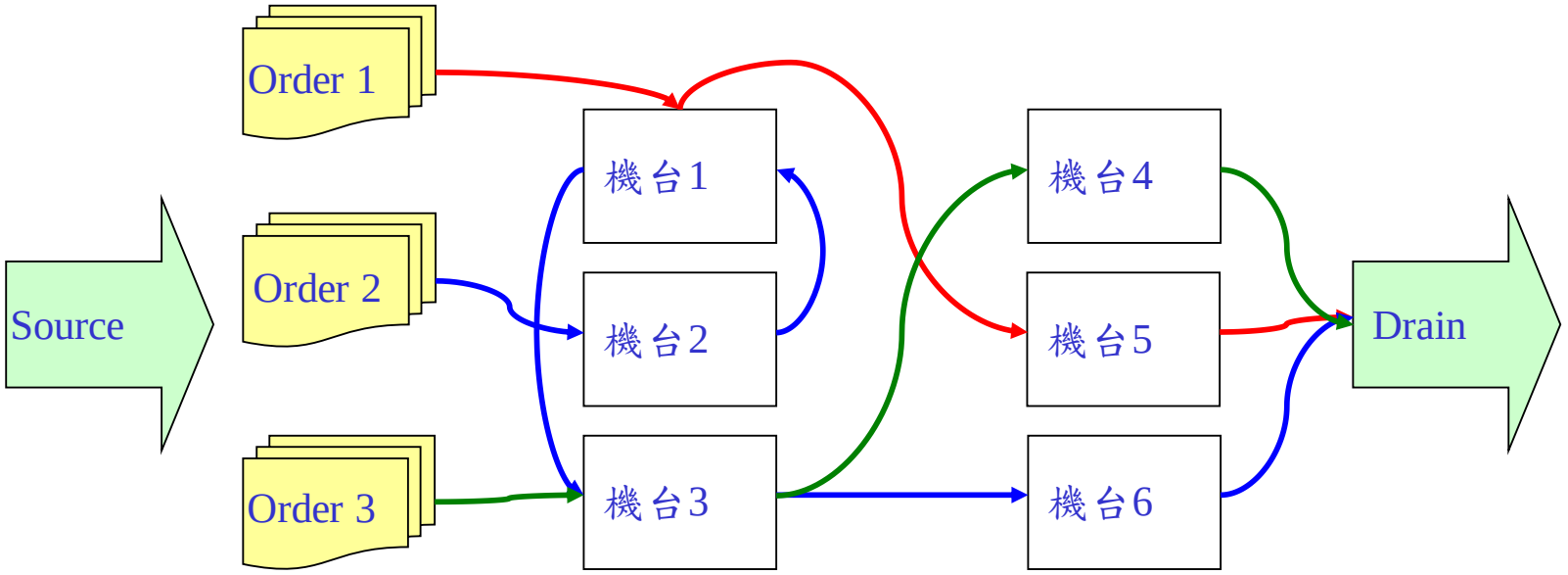
Job shop scheduling
step by step algorithm example

零工式生產排程問題情境

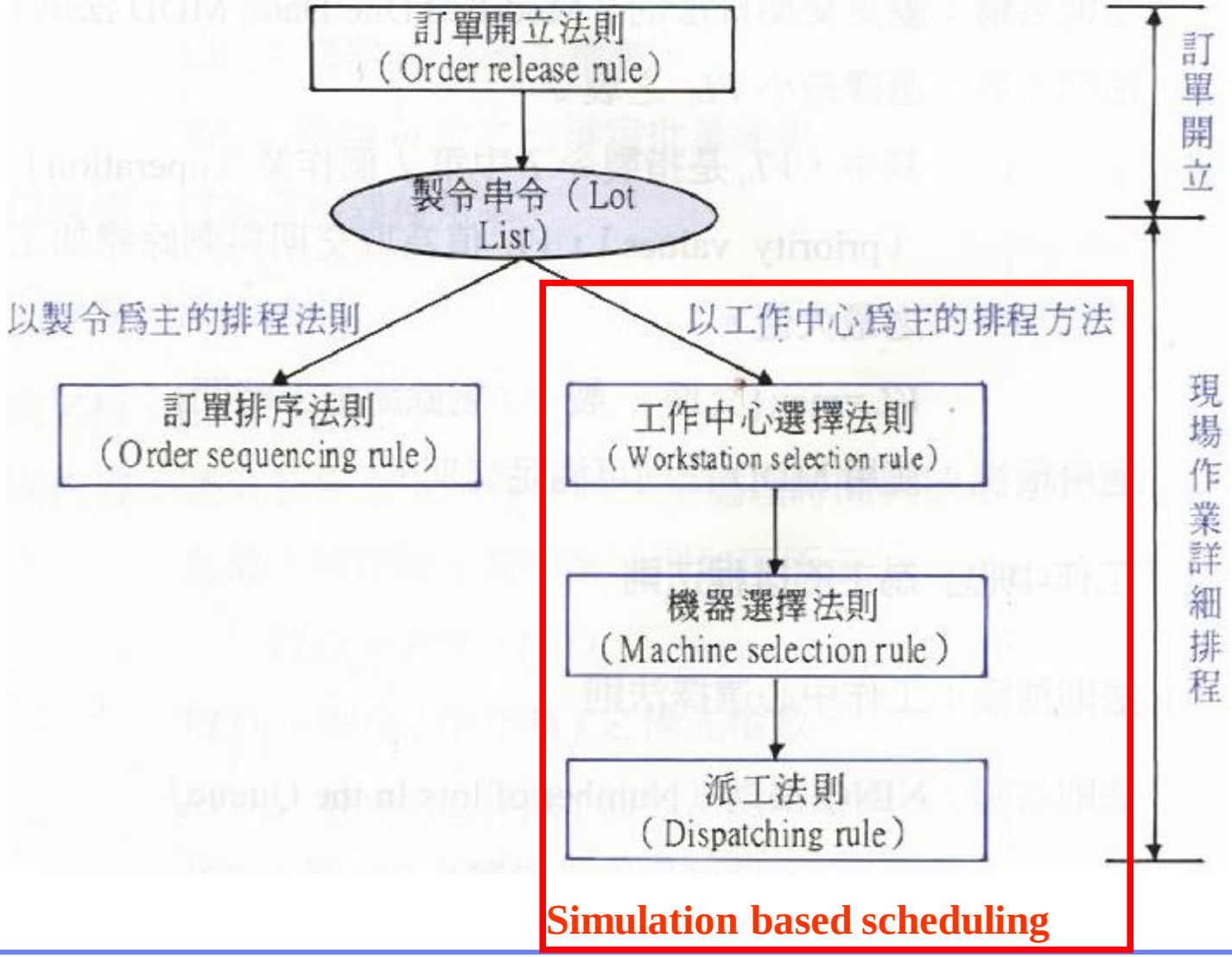
- 典型的零工式行生產（job shop）系統排程環境中，其生產特性極為複雜的加工作業典型，所有工作或製令單在系統內流動有其各自的作業排程。
  - 每種工作或製令單有不同的來到時間（release time）與交期（due date）
  - 每種工作或製令單有各自的加工途程（routing）與加工時間（process time）
  - 主要決定工作或製令單的於各加工中心的開始時間點與投料數量



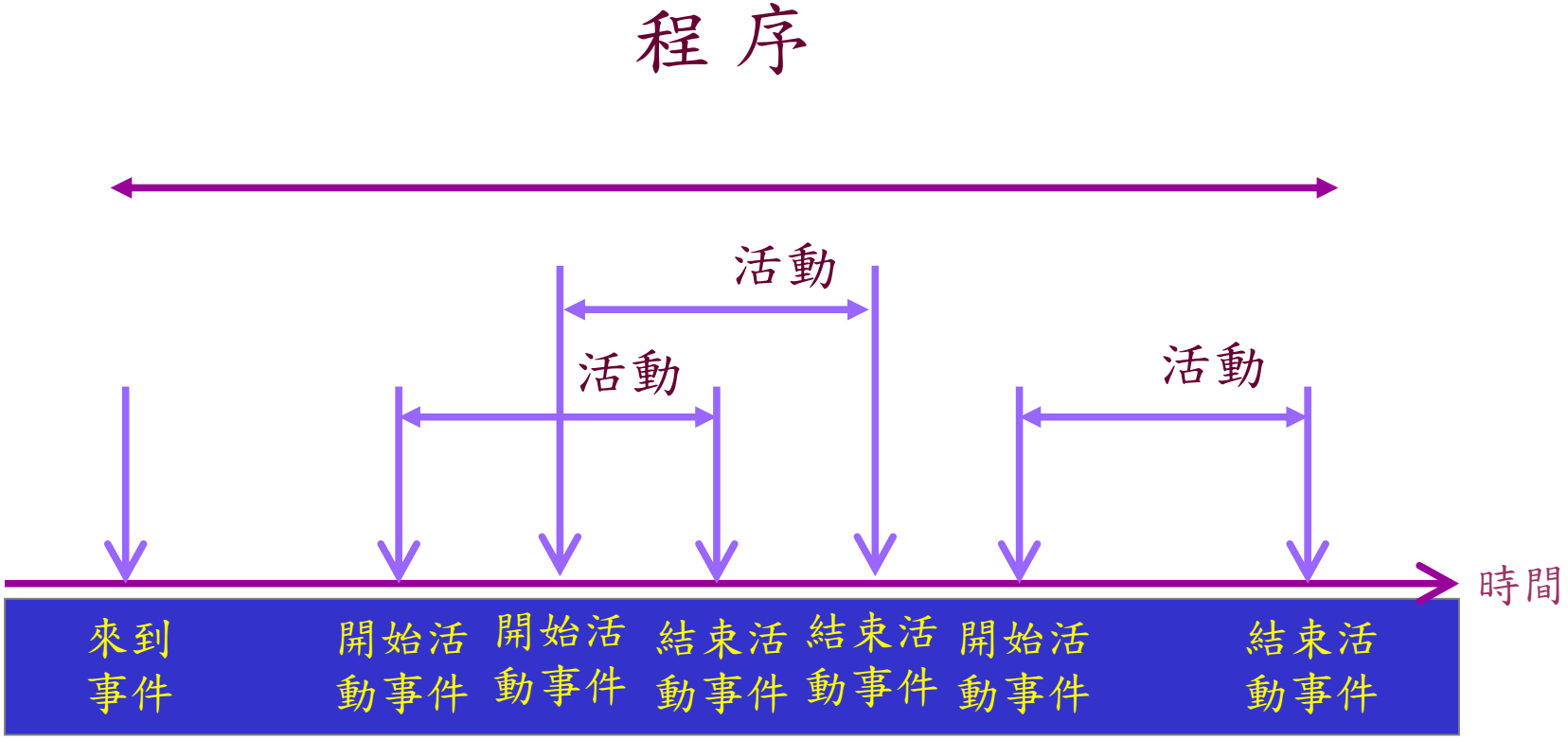
零工式生產排程問題

- 一般而言，有限產能排程問題主要可分成下列幾種：
  - 以「製令」為主的排程方法(Job-based Scheduling或Order-based Scheduling)：利用訂單排序法則決定訂單或製令的加工優先順序，再按順序高低逐一安排各製令的詳細作業排程。
  - 以「工作中心」為主的排程方法(Event-based Scheduling)：運算邏輯是利用事件導向(Event-Driven)的模擬觀念來描述製造系統的實際運作流程。
    - 先進先出派工法則（FIFO）：以先來到等候線的工作為優先
    - 最短作業時間派工法則（SPT）：以等候線中加工時間做短的作業優先
    - 最早交期派工法則（EDD）：以等候線中最早交期的工作為優先
    - LSF, LWR,……

以「製令」為主及以「工作中心」為主的排程方法



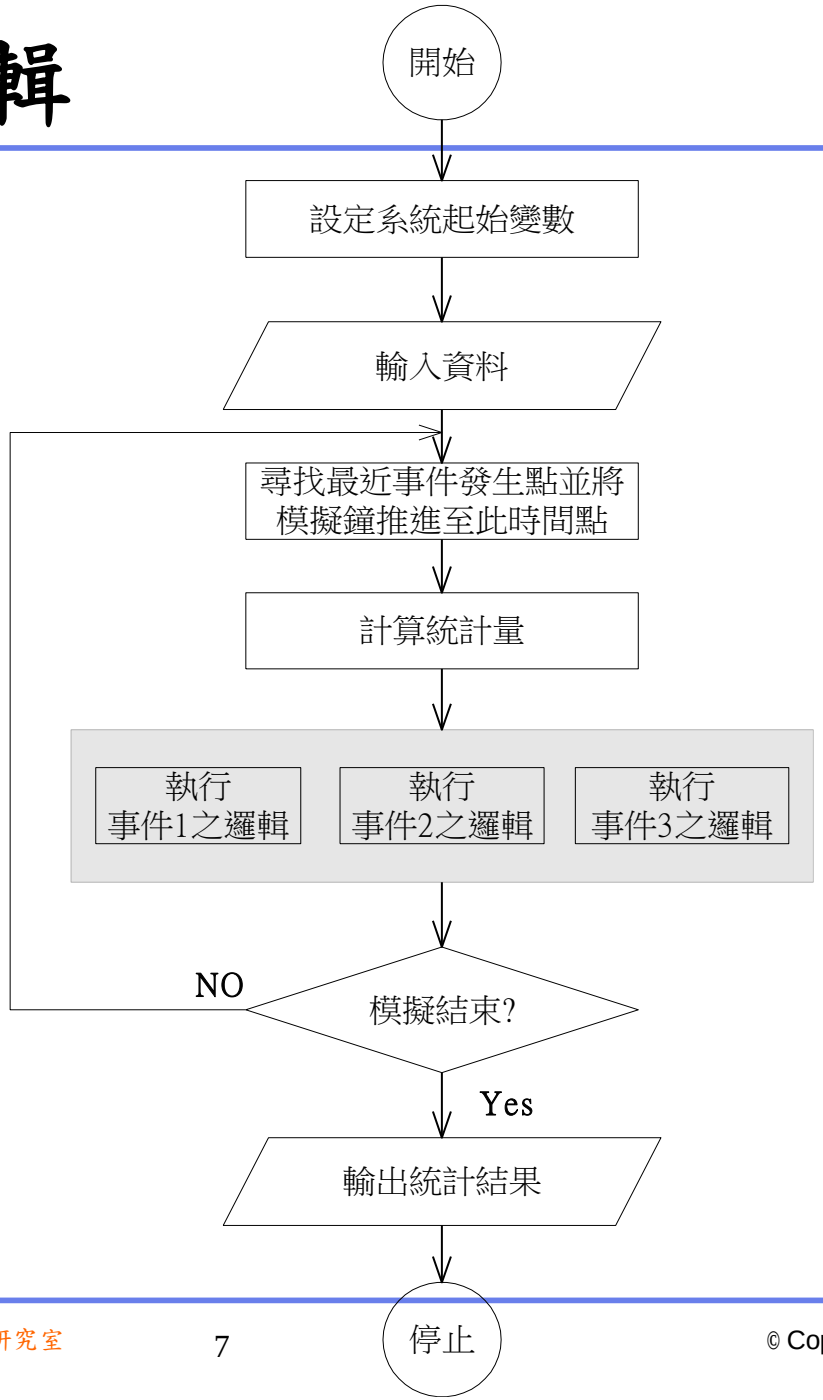
事件(Event)、活動(Activity)與程序(Process)關係



事件排程法(Event Scheduling)

- 在「事件排程法模擬」中，主要為描述並記錄事件點上系統狀態的變化。
- 模式構建的任務即在決定有那些改變系統狀態的事件，以及定義事件與事件間的邏輯關係。
- 模擬即以虛擬的時間推進機構將事件依時間發生先後順序負責執行其相關邏輯，從而改變系統狀態，收集相關統計資料。

事件排程法邏輯



零工式生產排程範例

1.零工式生產排程系統由三台加工機台以及個別機台前的等候暫存區（Buffer）所組成。

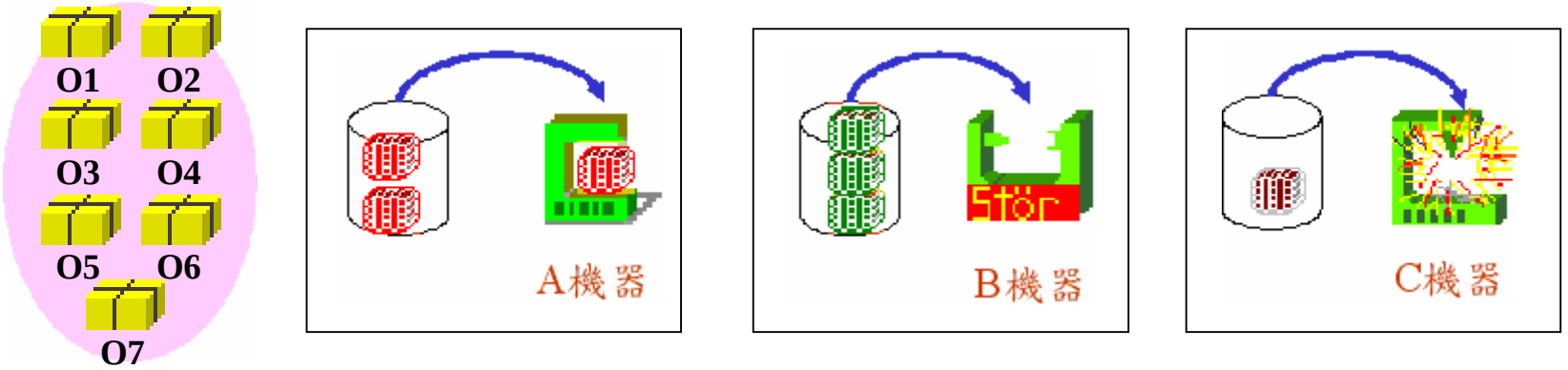
2.機台前的等候暫存區則可以無限容納多張製令等候加工 。

3.製令來到時間：每張製令依來到時間，來到後才能開始加工。

4.機台加工處理時間：製令在系統內流動有其各自不同的作業途程以及加工時間，例如製令七只有經過B機器與C機器加工，且分別在這兩台機器上的加工時間為5小時與2小時。

5.機台有空：製令馬上接受處理 ；機台正忙：製令排在等候線的最後。

6.從等候線中選擇下一張製令的規則：「製令交期越早優先加工」(EDD）。



零工式生產排程例的時間數據

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

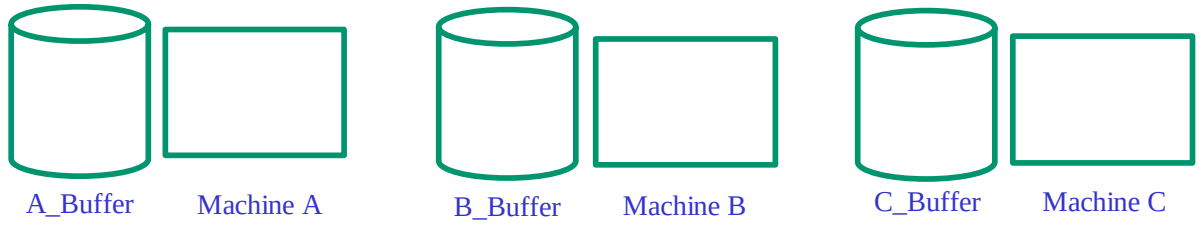
表1. 零工式排程系統的製令數據

零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

Future Event List	
Event type	**Time**
Arrival	0
A_Complete	infinite
B_Complete	infinite
C_Complete	infinite

Event type
initialize
Time

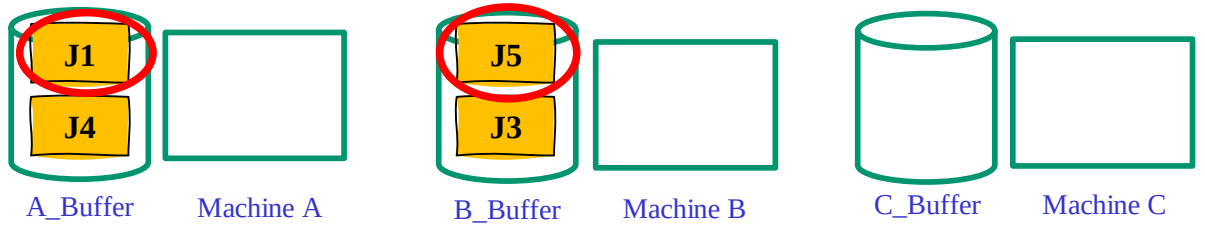


零工式生產排程範例

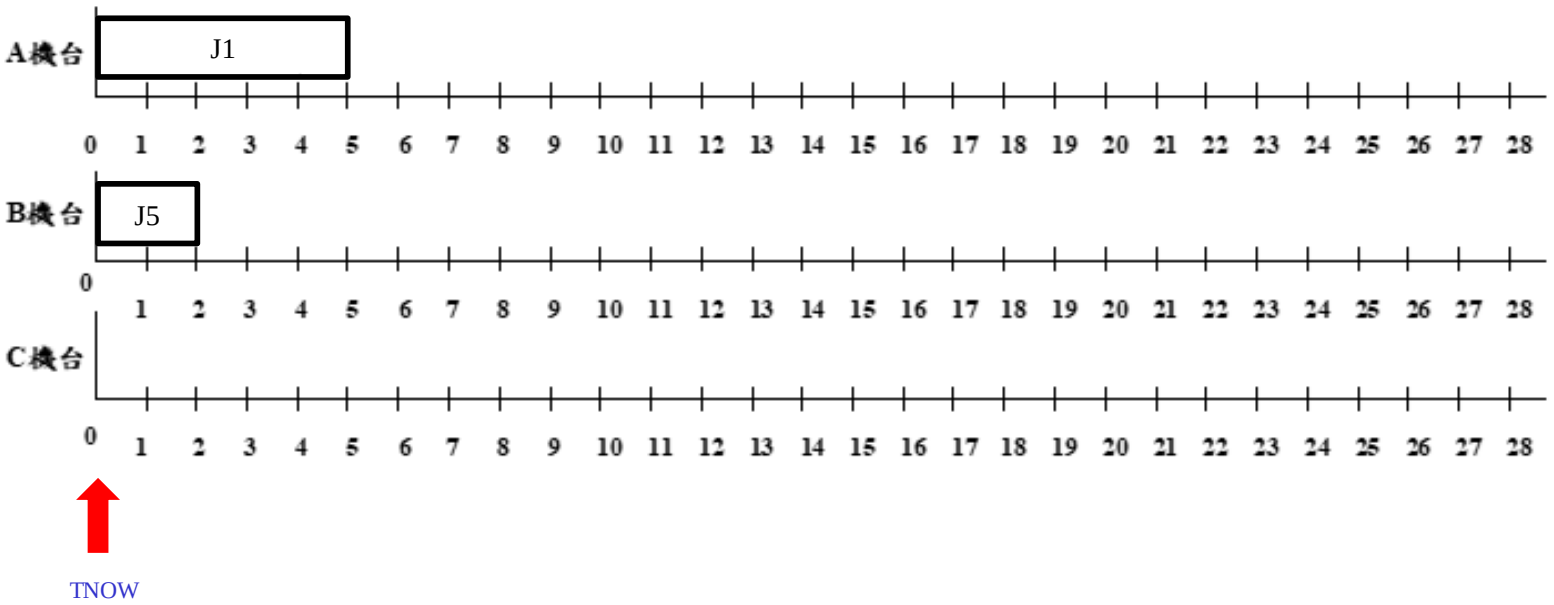
製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

Future Event List	
Event type	**Time**
Arrival	4
A_Complete	5
B_Complete	2
C_Complete	infinite

Event type
Arrival event
Time
0



零工式生產排程範例

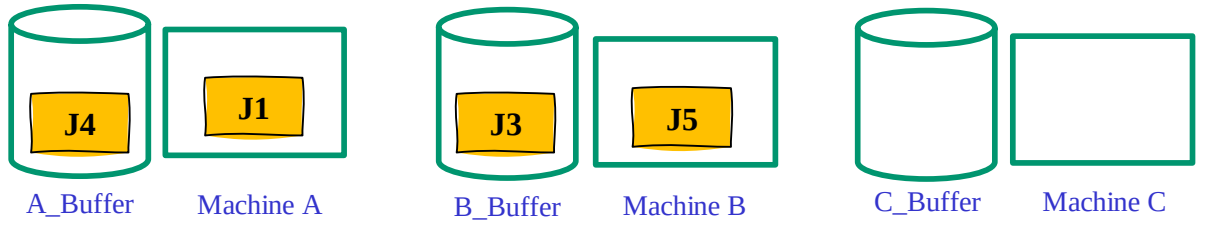


零工式生產排程範例

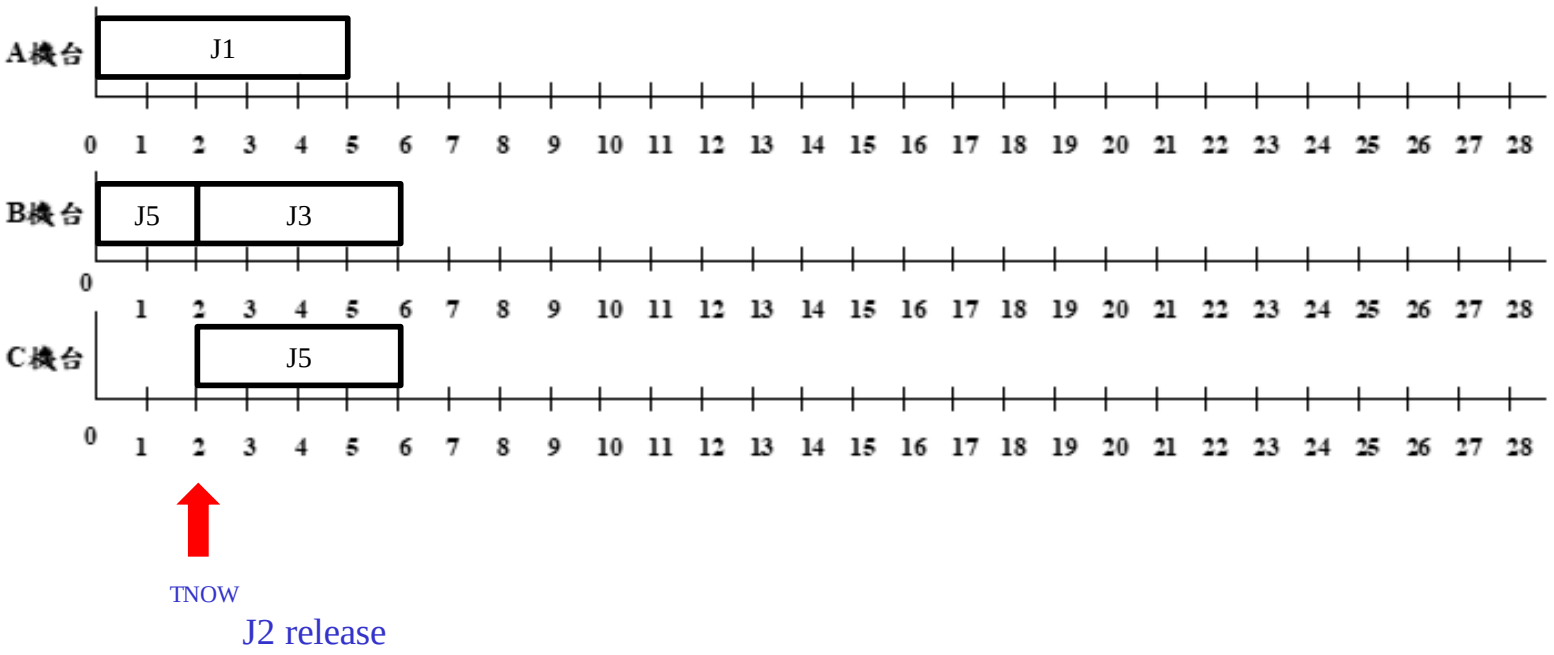
製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

Future Event List	
Event type	**Time**
Arrival	4
A_Complete	5
B_Complete	6
C_Complete	6

Event type
B_complete
Time
2



零工式生產排程範例

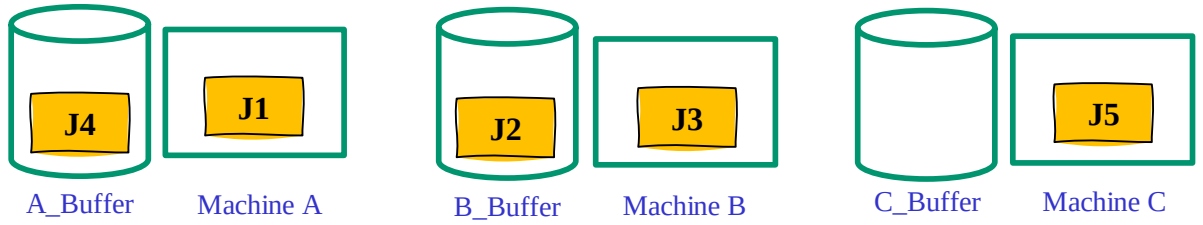


零工式生產排程範例

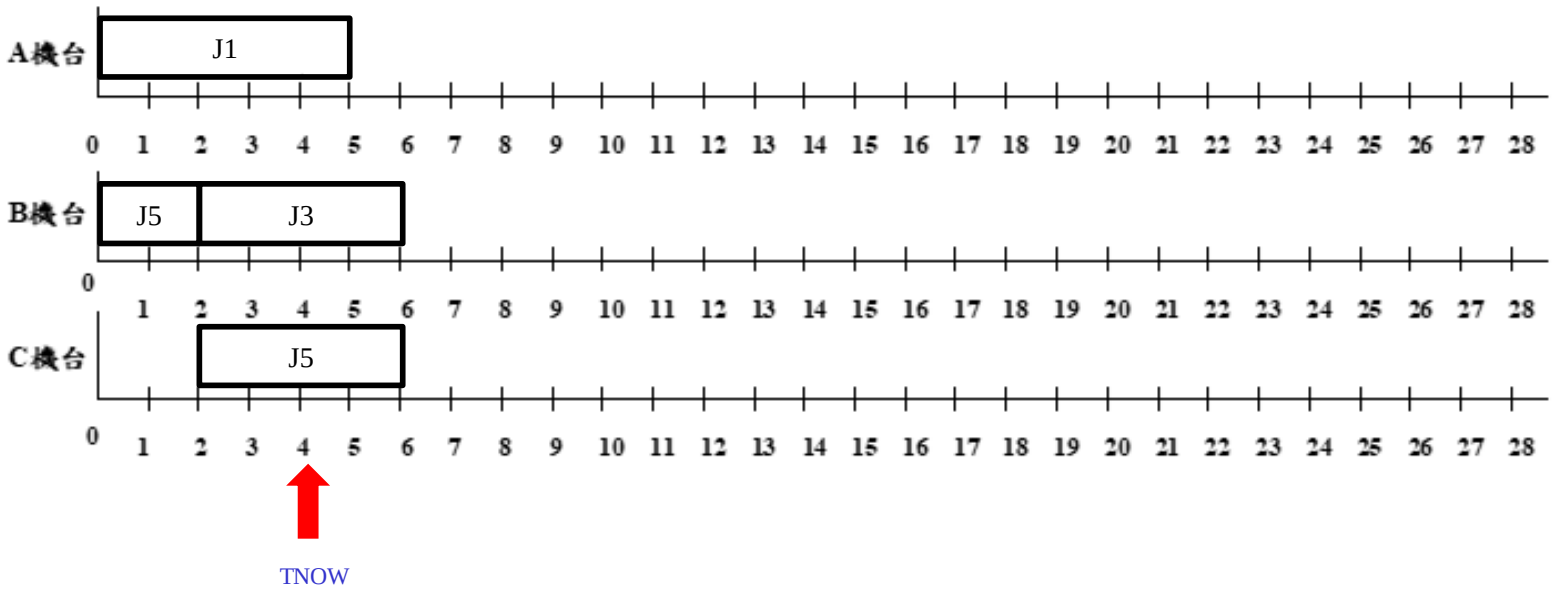
製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

Future Event List	
Event type	**Time**
Arrival	7
A_Complete	5
B_Complete	6
C_Complete	6

Event type
Arrival event
Time
4



零工式生產排程範例

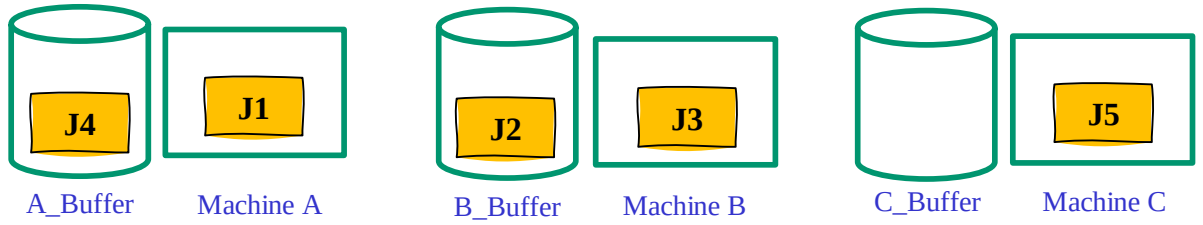


零工式生產排程範例

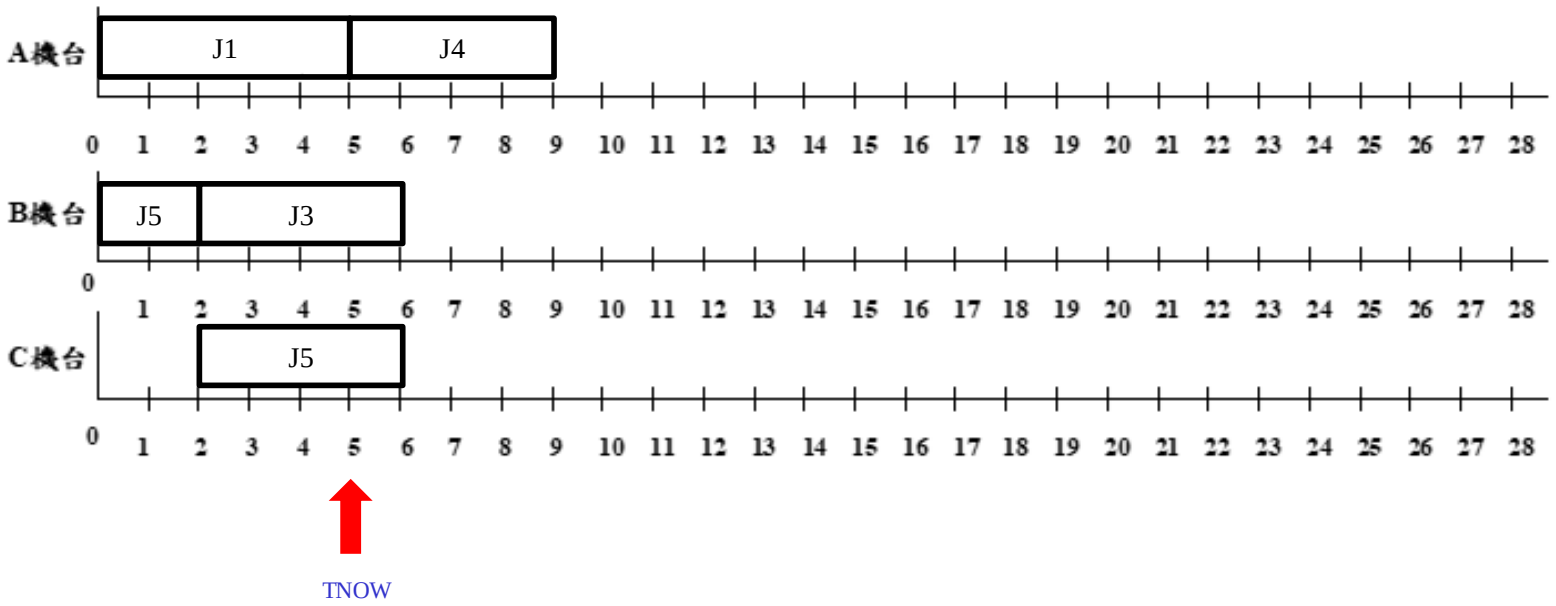
製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

Future Event List	
Event type	**Time**
Arrival	7
A_Complete	9
B_Complete	6
C_Complete	6

Event type
A_complete
Time
5



零工式生產排程範例

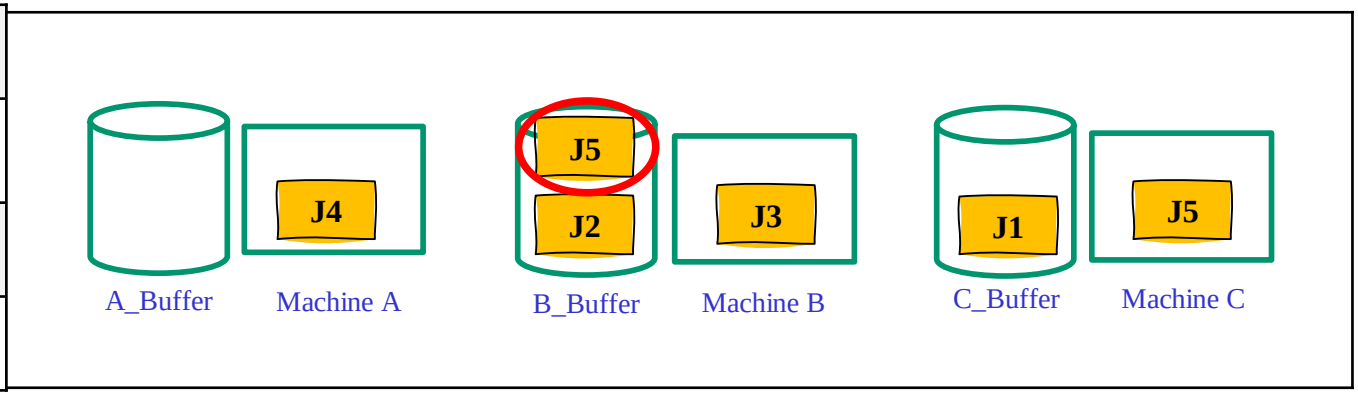


零工式生產排程範例

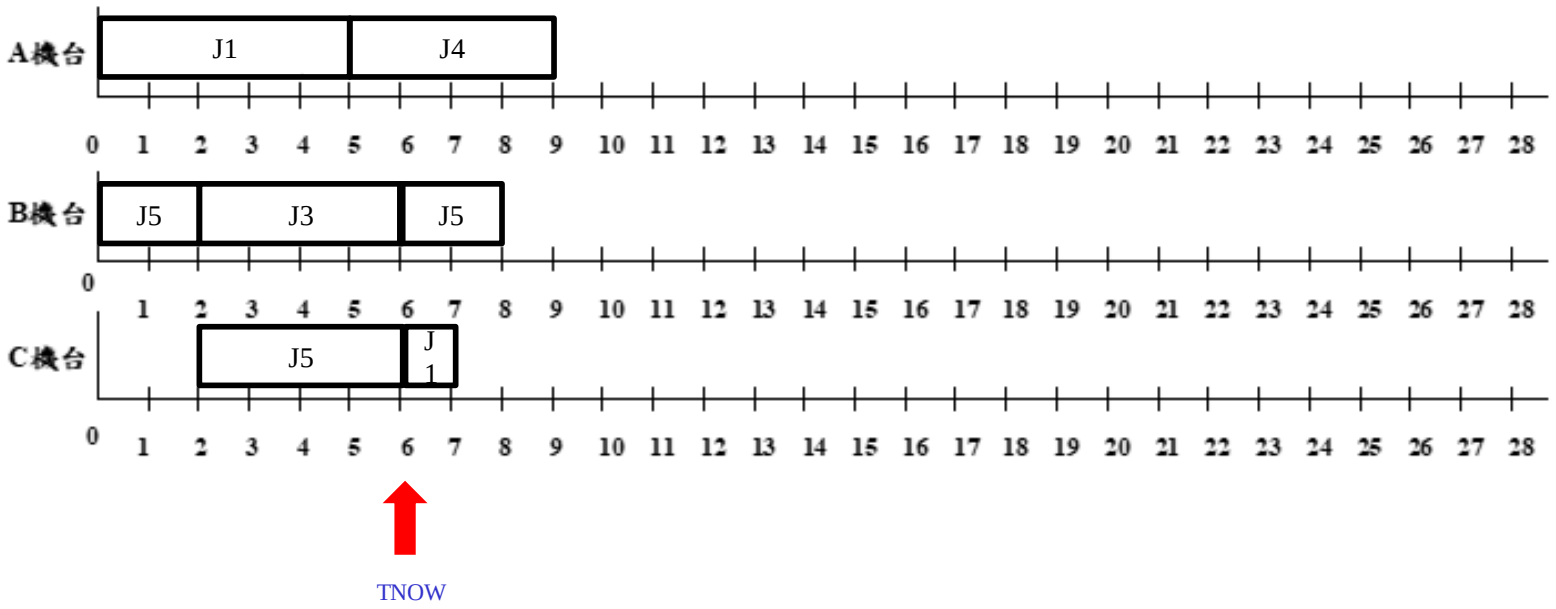
製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

Future Event List	
Event type	**Time**
Arrival	7
A_Complete	9
B_Complete	8
C_Complete	7

Event type
B_complete & C_complete
Time
6



零工式生產排程範例

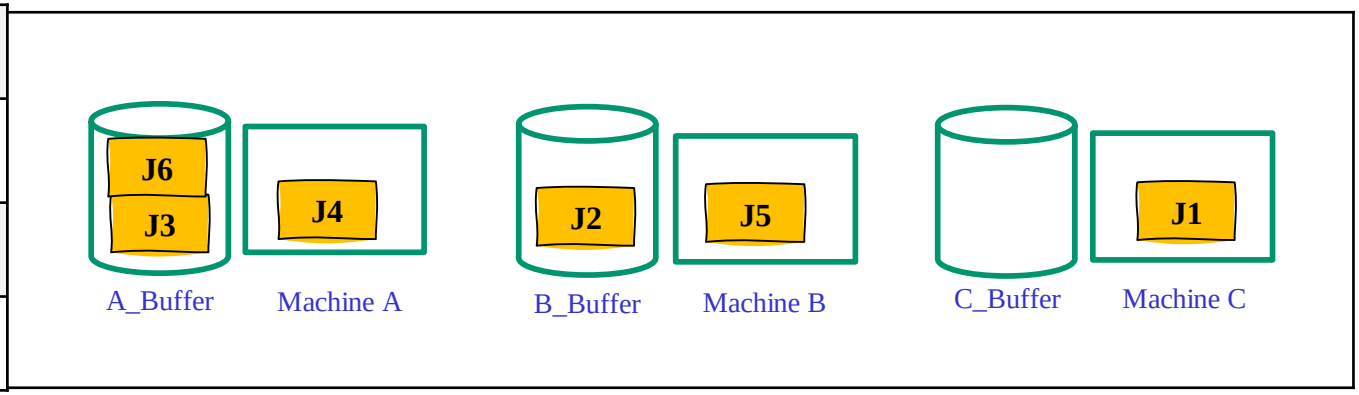


零工式生產排程範例

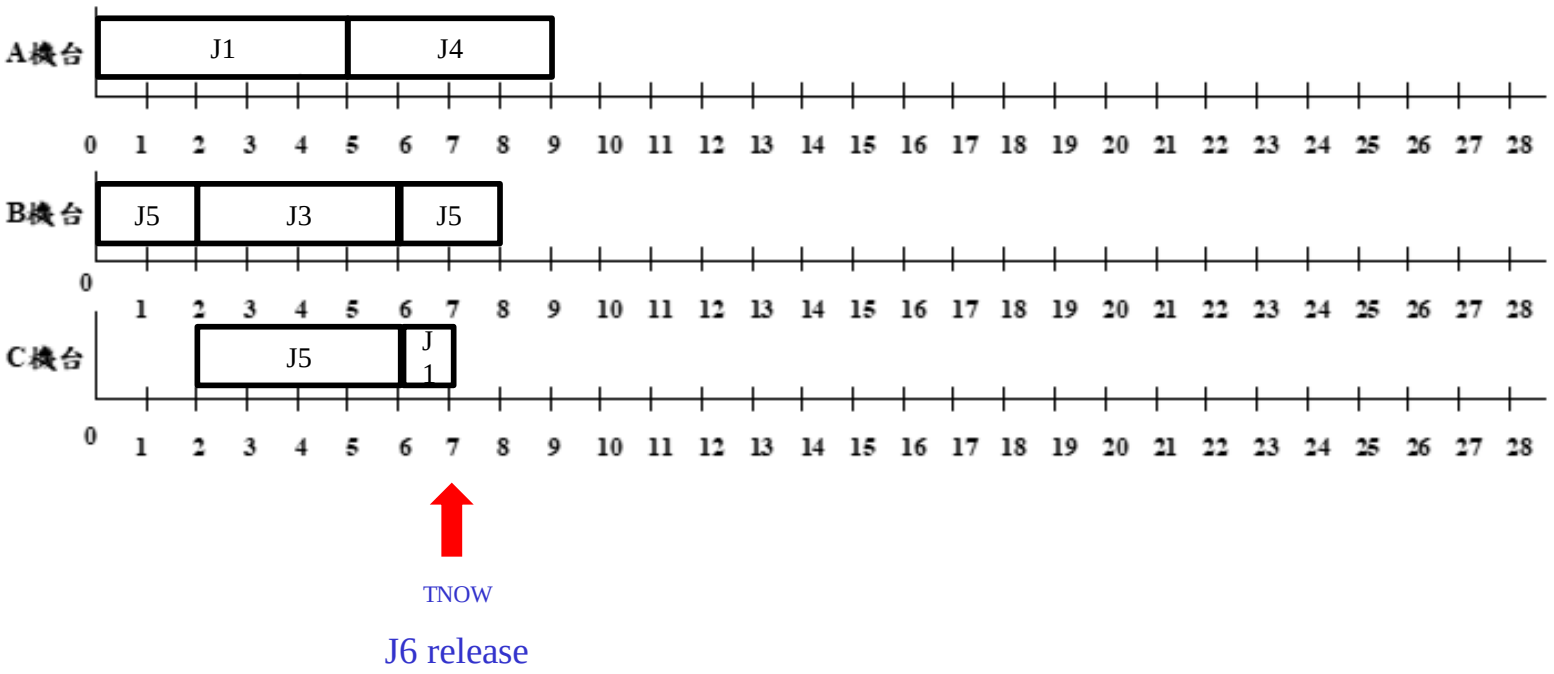
製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

Future Event List	
Event type	**Time**
Arrival	10
A_Complete	9
B_Complete	8
C_Complete	infinite

Event type
Arrival & C_complete
Time
7



零工式生產排程範例

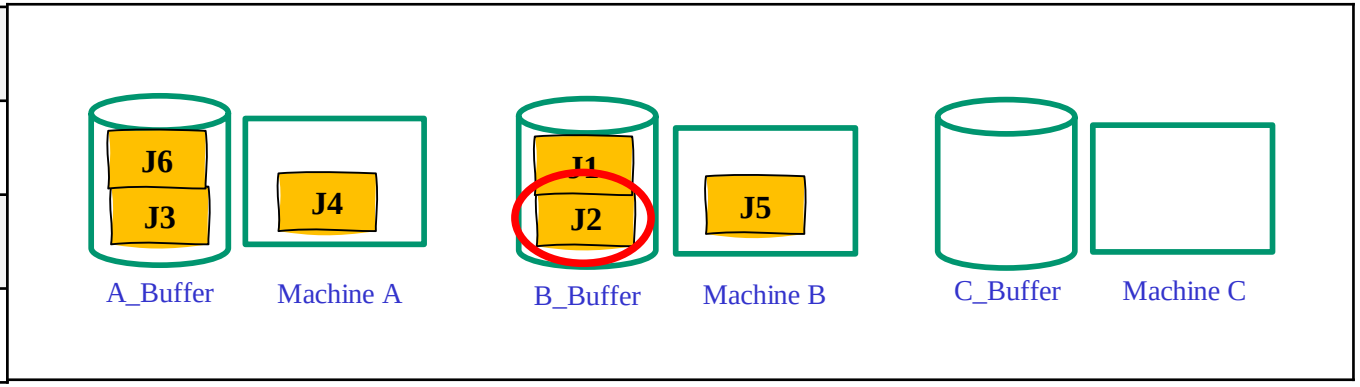


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

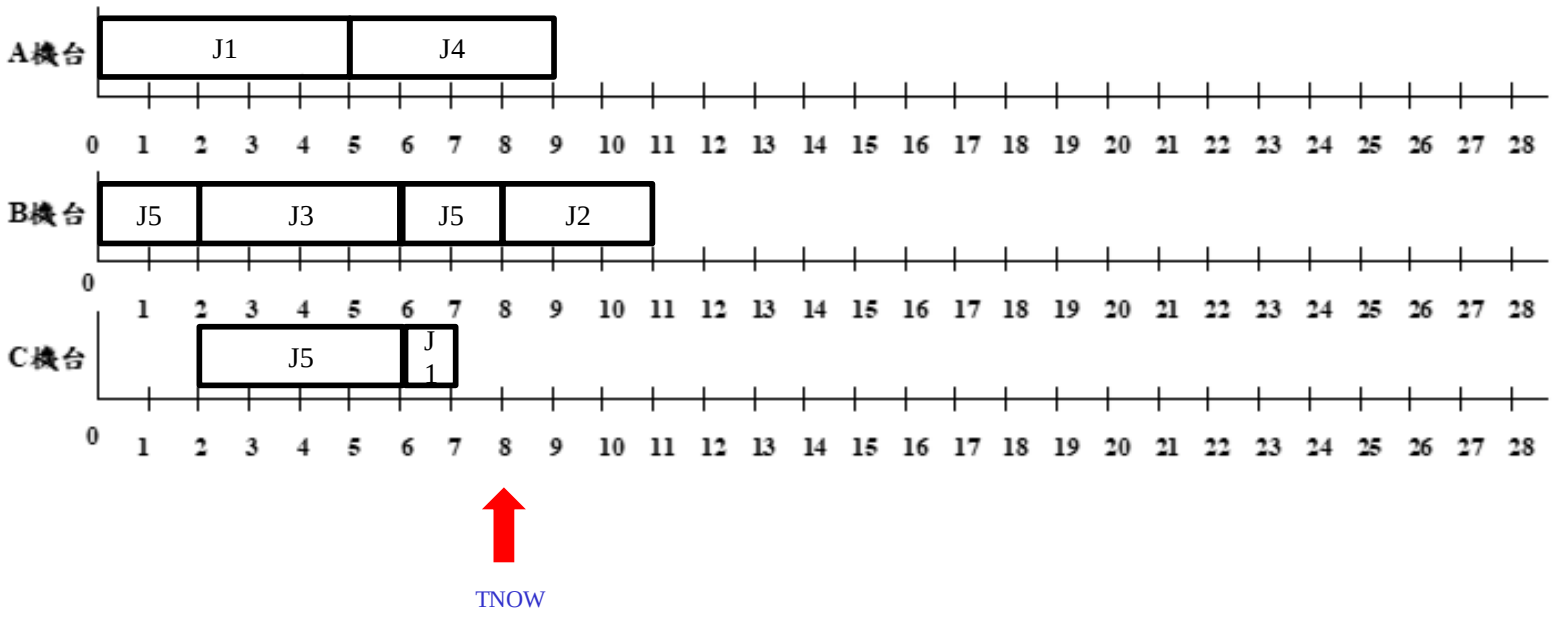
Future Event List	
Event type	**Time**
Arrival	10
A_Complete	9
B_Complete	11
C_Complete	infinite

Event type
B_complete
Time
8



Throughput : 1

零工式生產排程範例

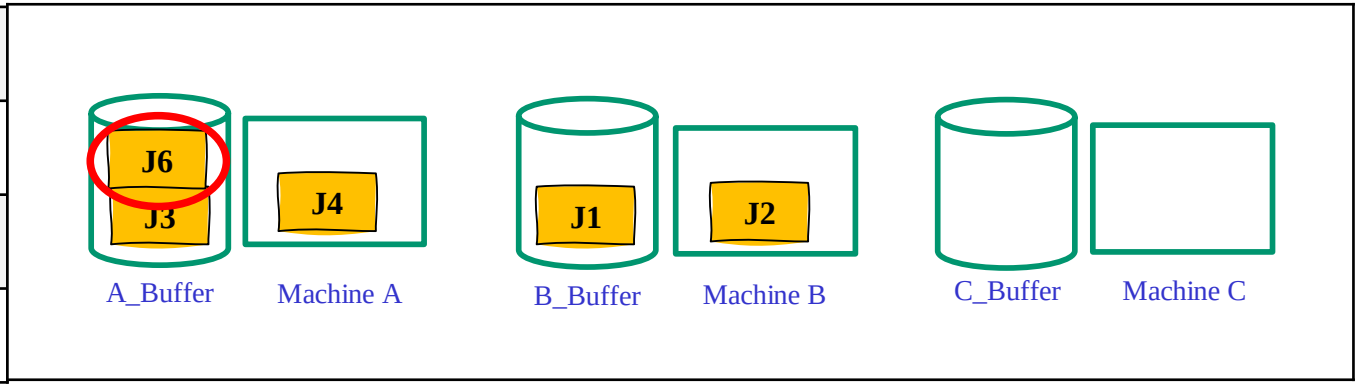


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

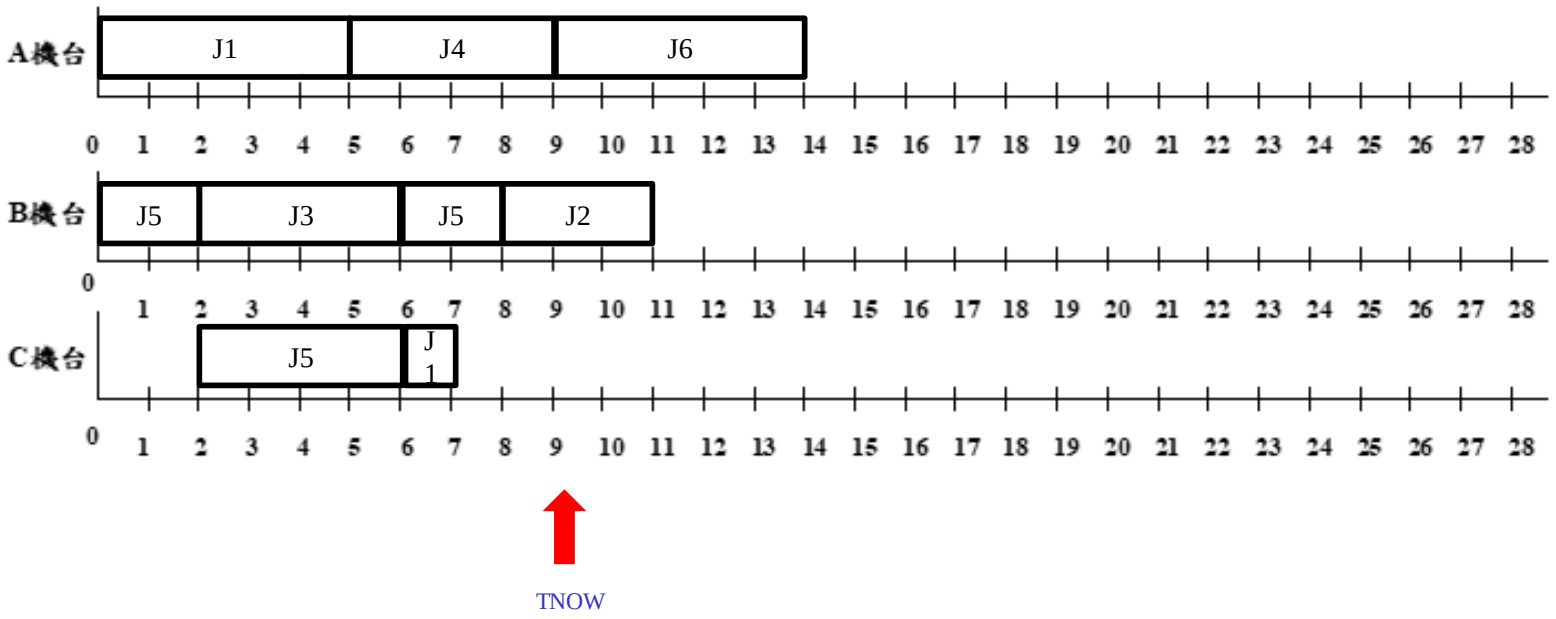
Future Event List	
Event type	**Time**
Arrival	10
A_Complete	14
B_Complete	11
C_Complete	infinite

Event type
A_complete
Time
9



Throughput : 1

零工式生產排程範例

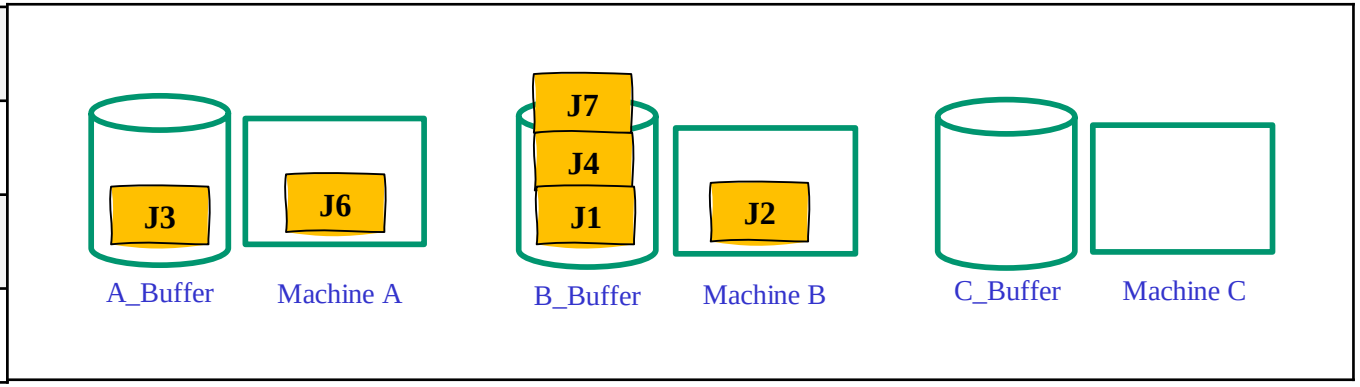


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

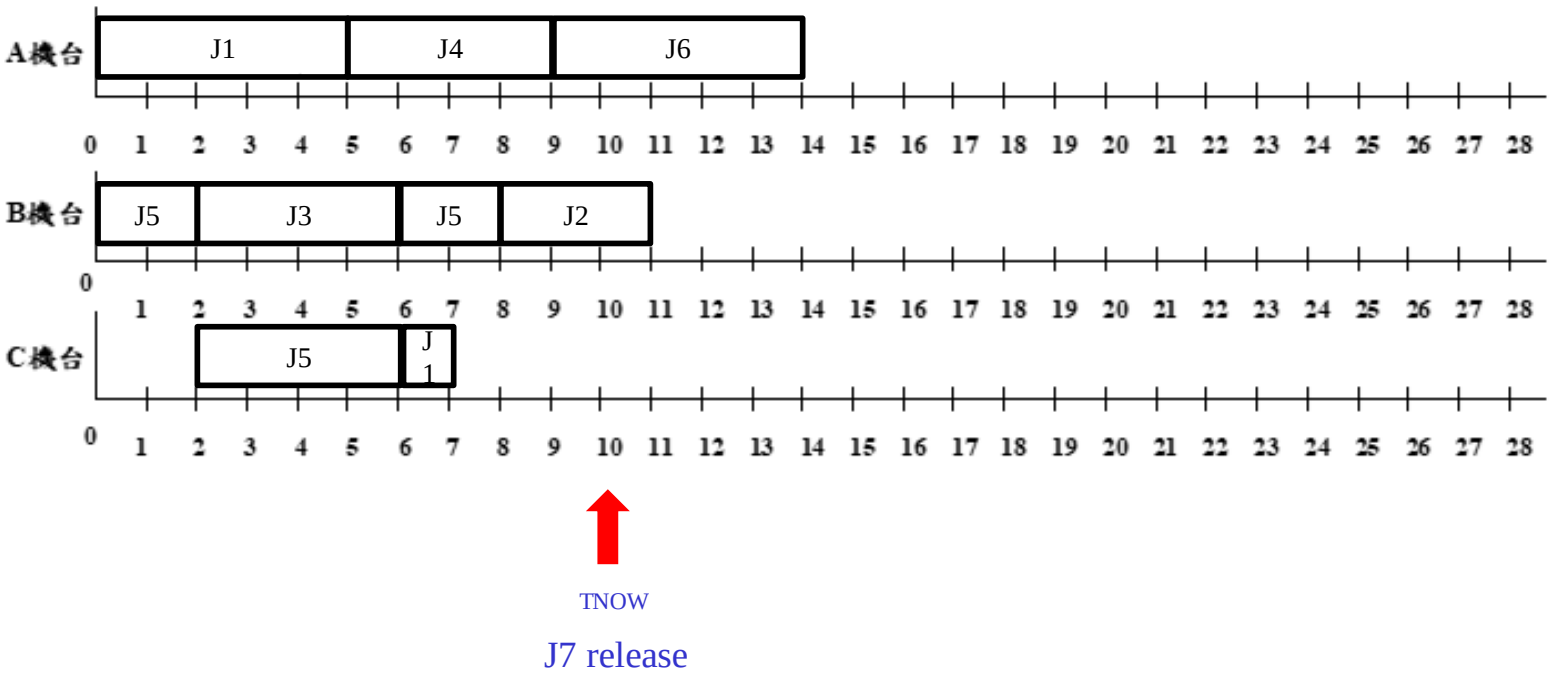
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	14
B_Complete	11
C_Complete	infinite

Event type
Arrival_event
Time
10



Throughput : 1

零工式生產排程範例

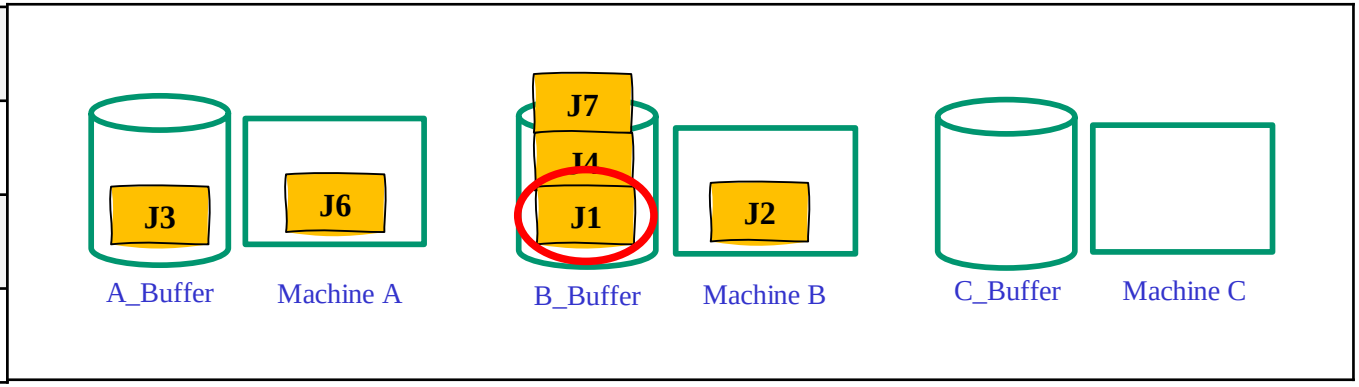


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

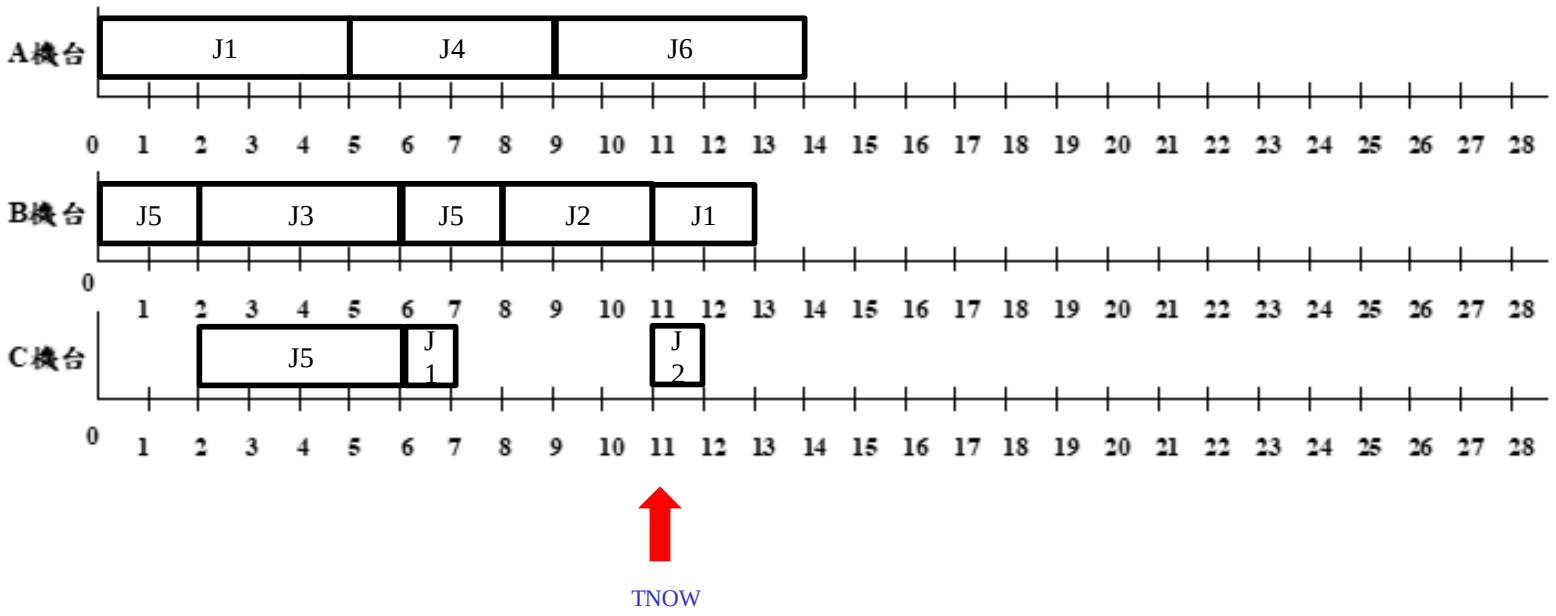
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	14
B_Complete	13
C_Complete	12

Event type
B_complete
Time
11



Throughput : 1

零工式生產排程範例

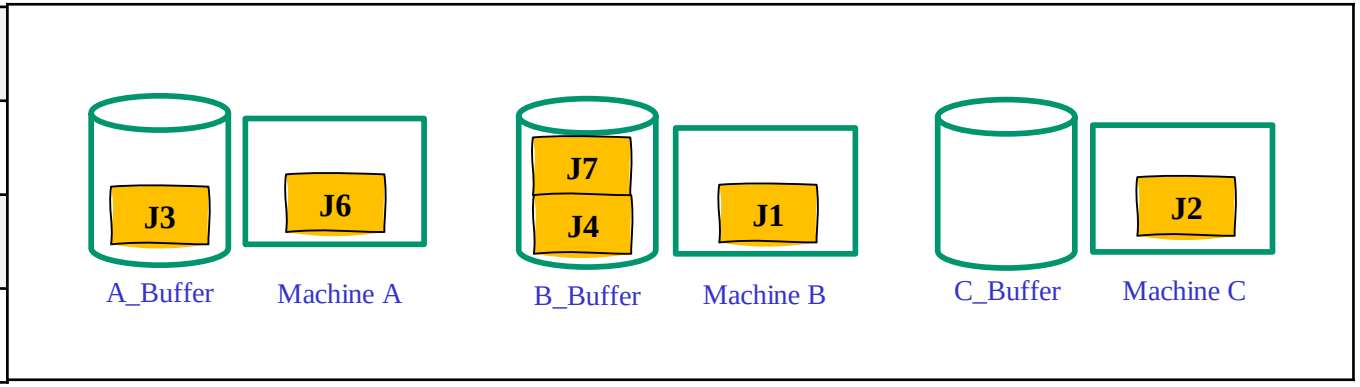


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

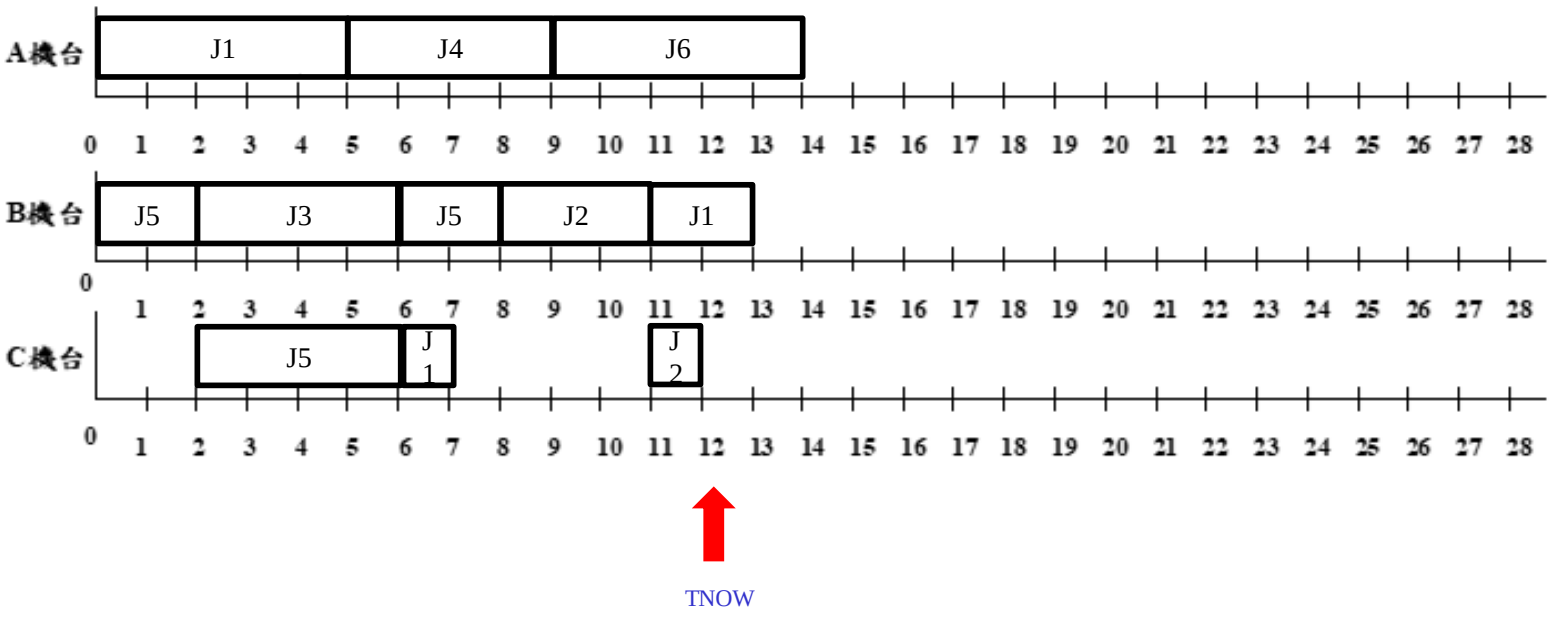
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	14
B_Complete	13
C_Complete	infinite

Event type
C_complete
Time
12



Throughput : 1

零工式生產排程範例

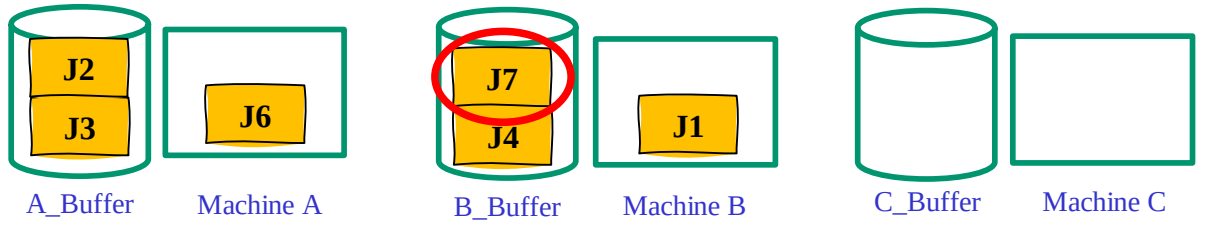


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

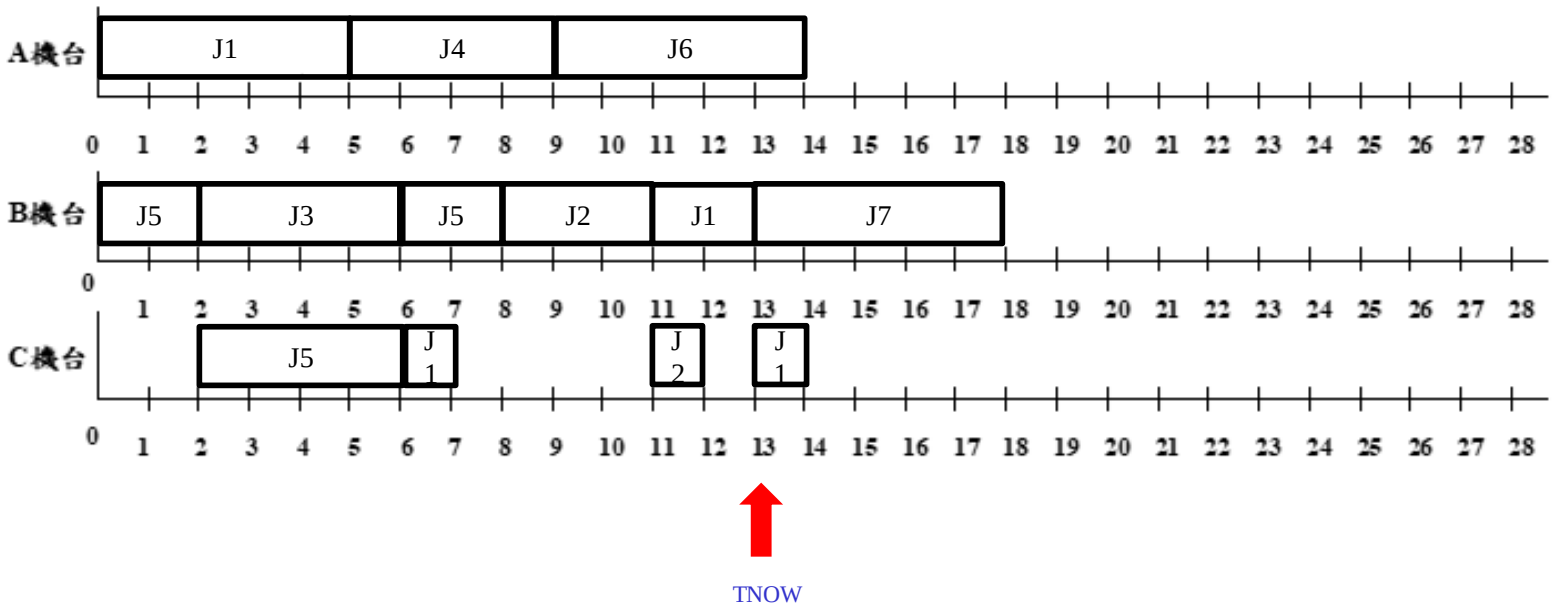
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	14
B_Complete	18
C_Complete	14

Event type
B_complete
Time
13



Throughput : 1

零工式生產排程範例

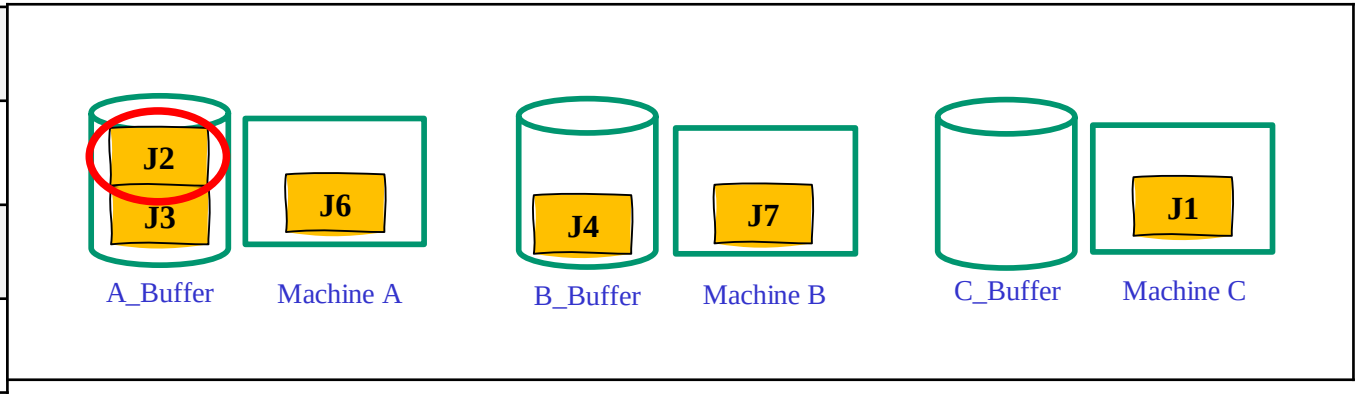


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

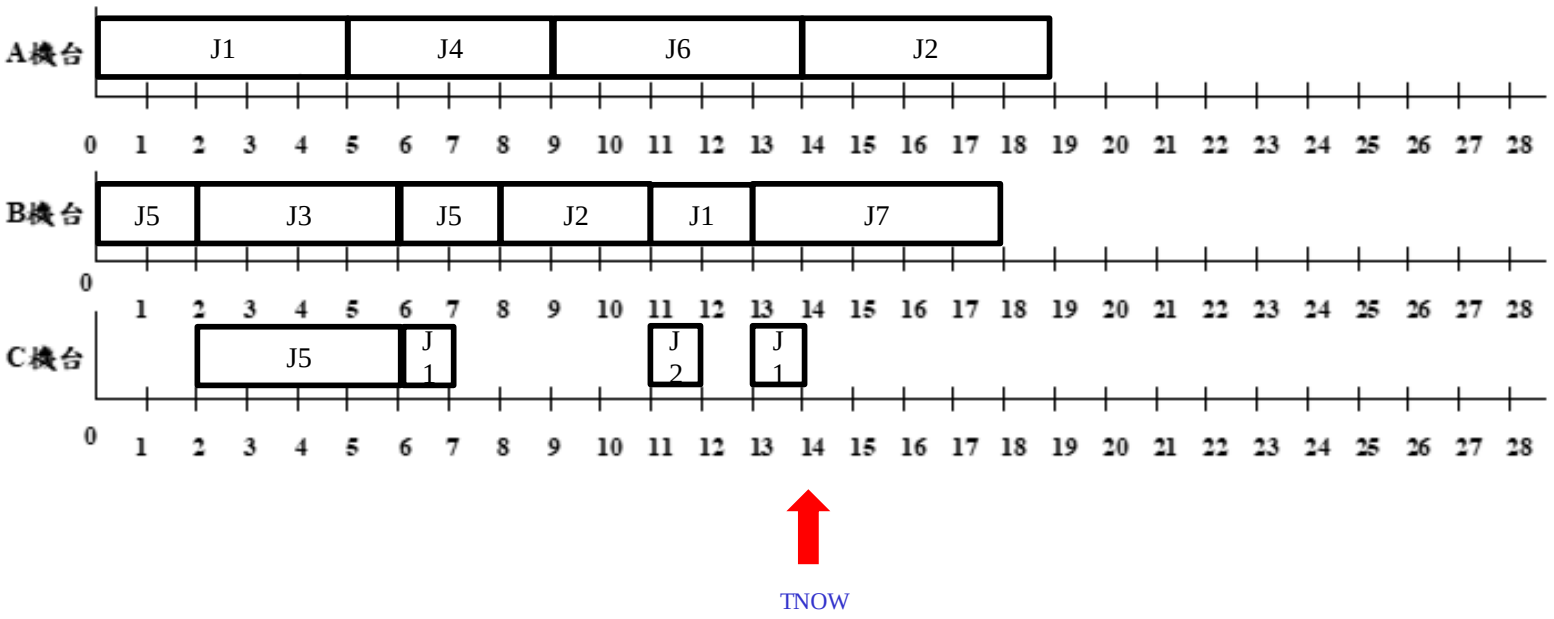
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	19
B_Complete	18
C_Complete	infinite

Event type
A_complete & C_complete
Time
14



Throughput : 2

零工式生產排程範例

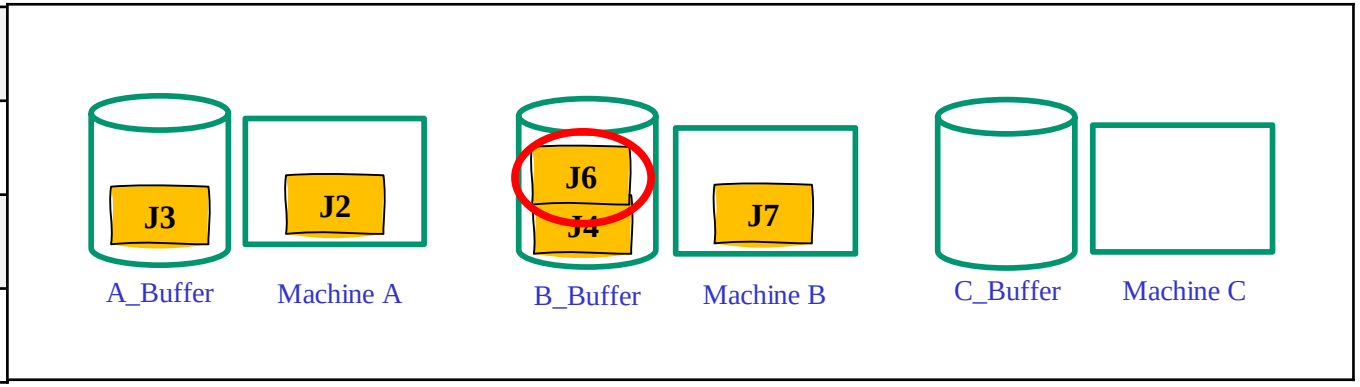


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

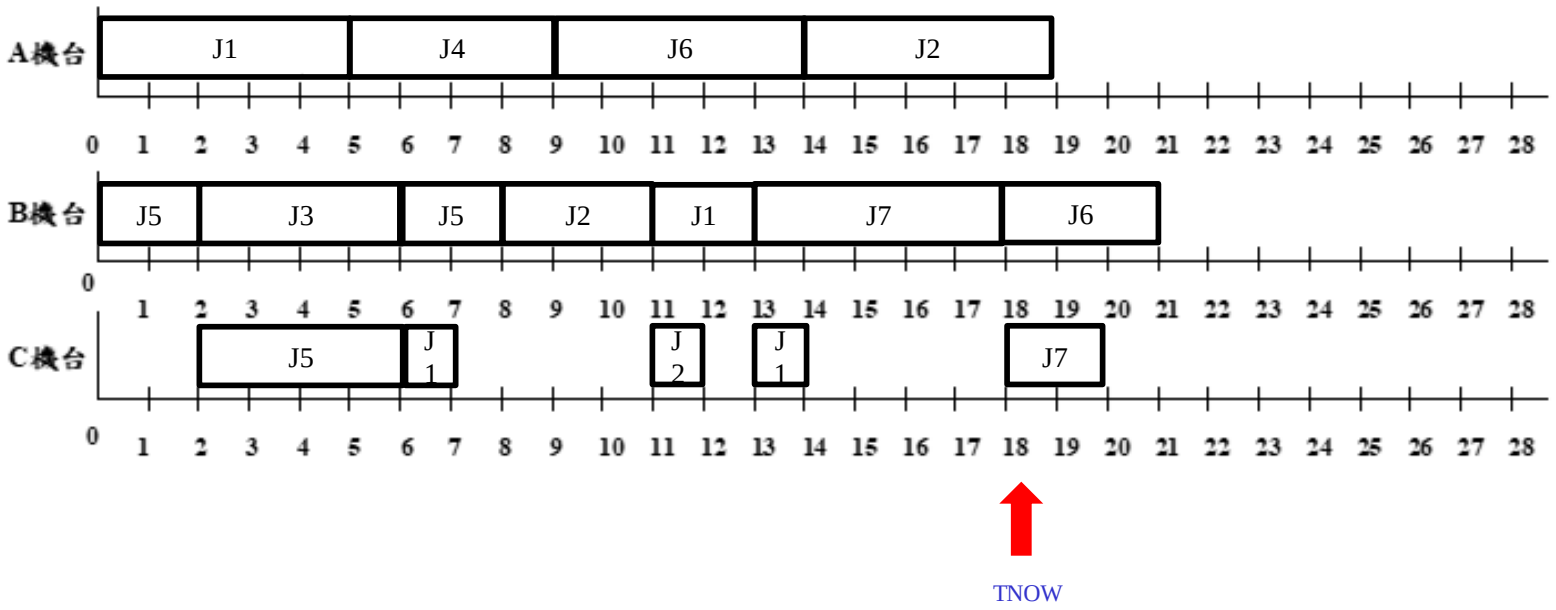
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	19
B_Complete	21
C_Complete	20

Event type
B_complete
Time
18



Throughput : 2

零工式生產排程範例

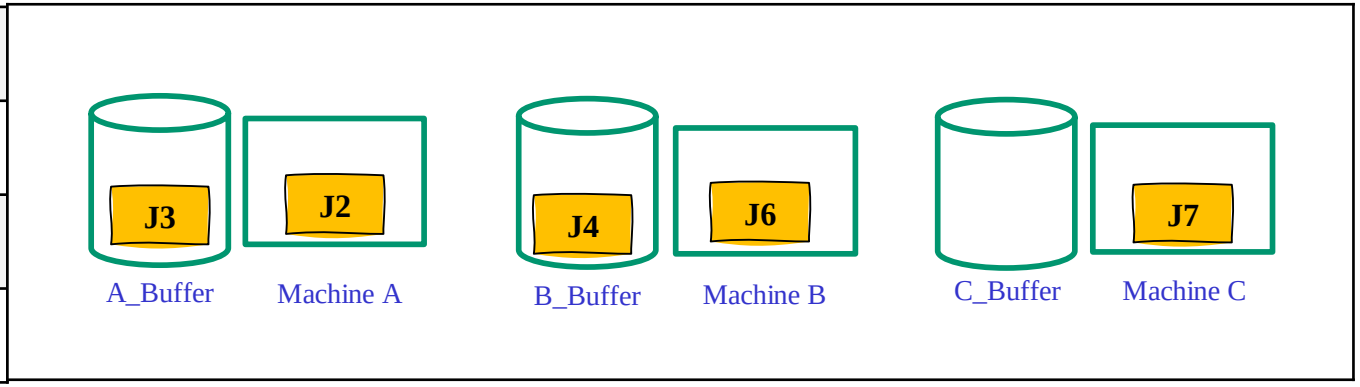


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

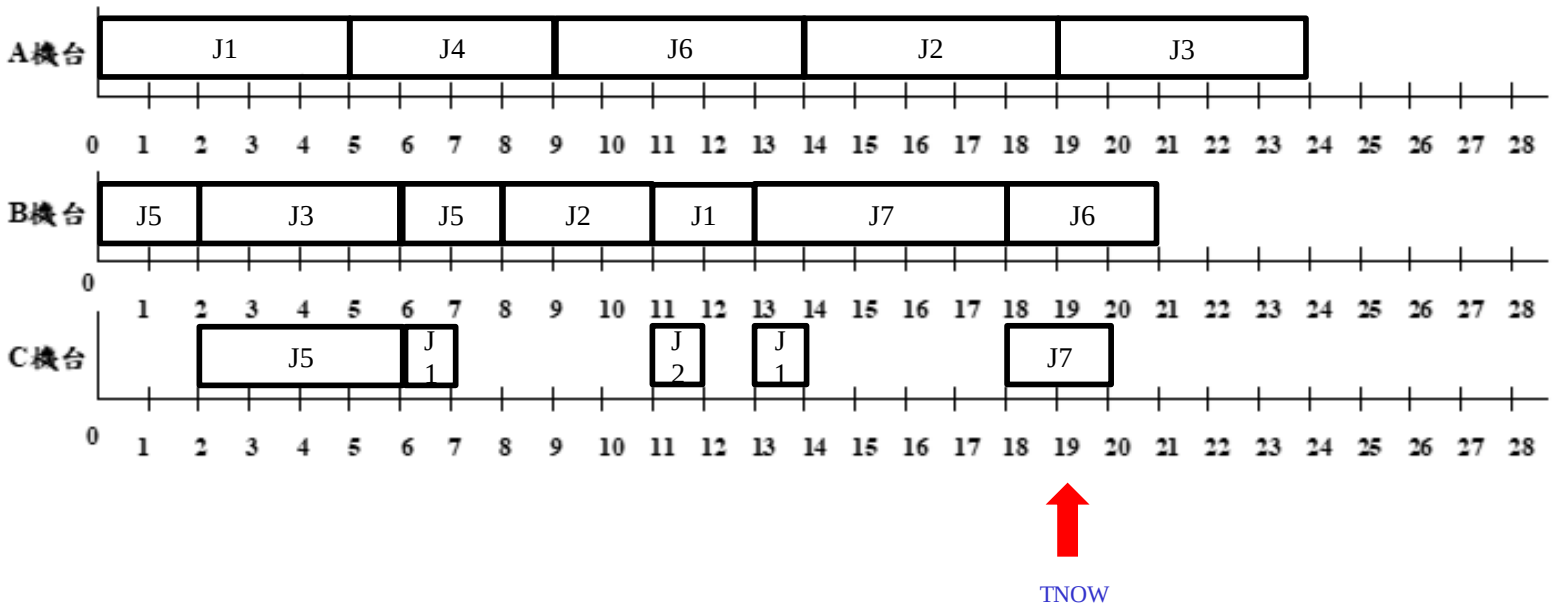
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	24
B_Complete	21
C_Complete	20

Event type
A_complete
Time
19



Throughput : 2

零工式生產排程範例

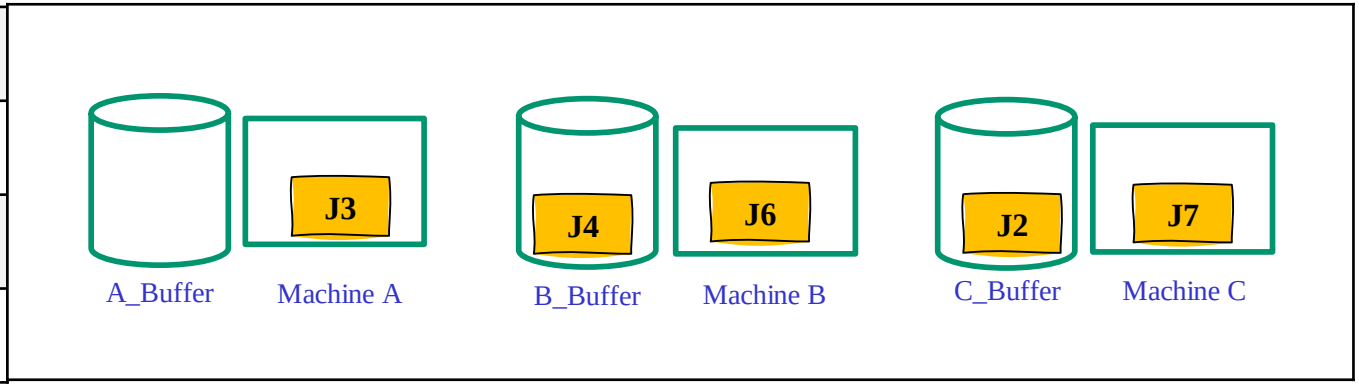


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

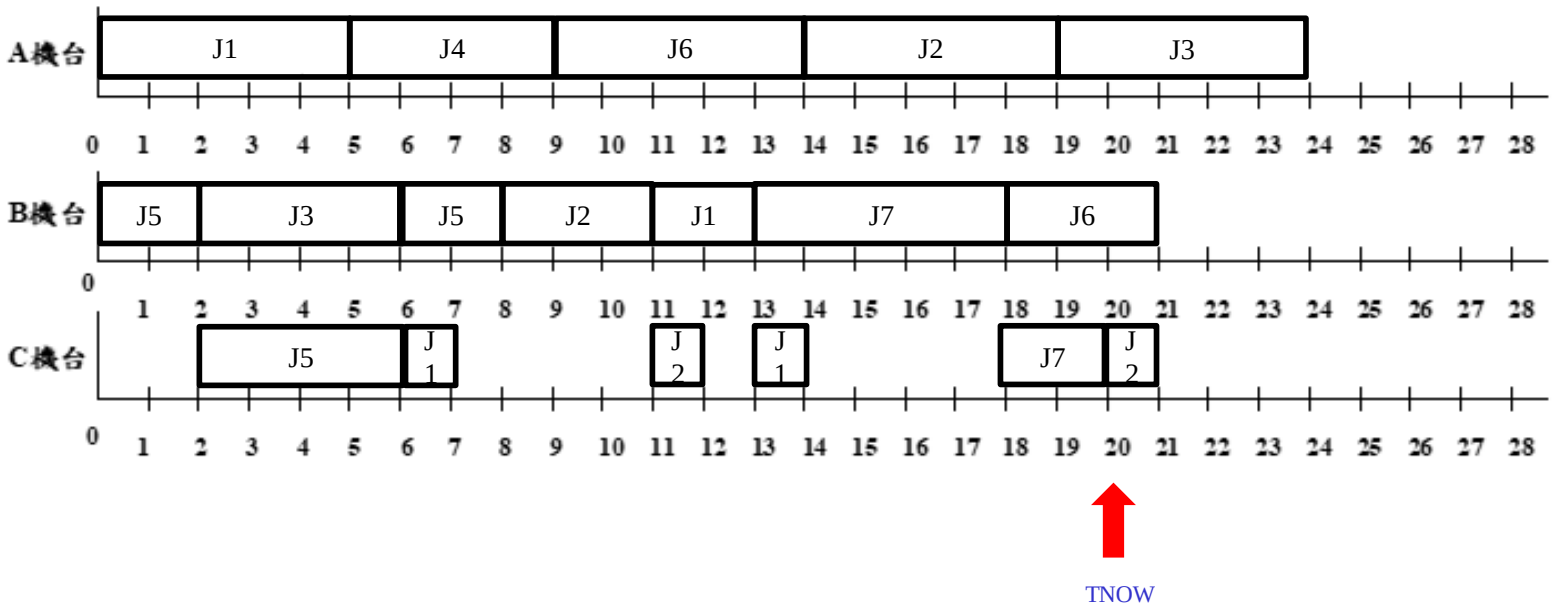
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	24
B_Complete	21
C_Complete	21

Event type
C_complete
Time
20



Throughput : 3

零工式生產排程範例

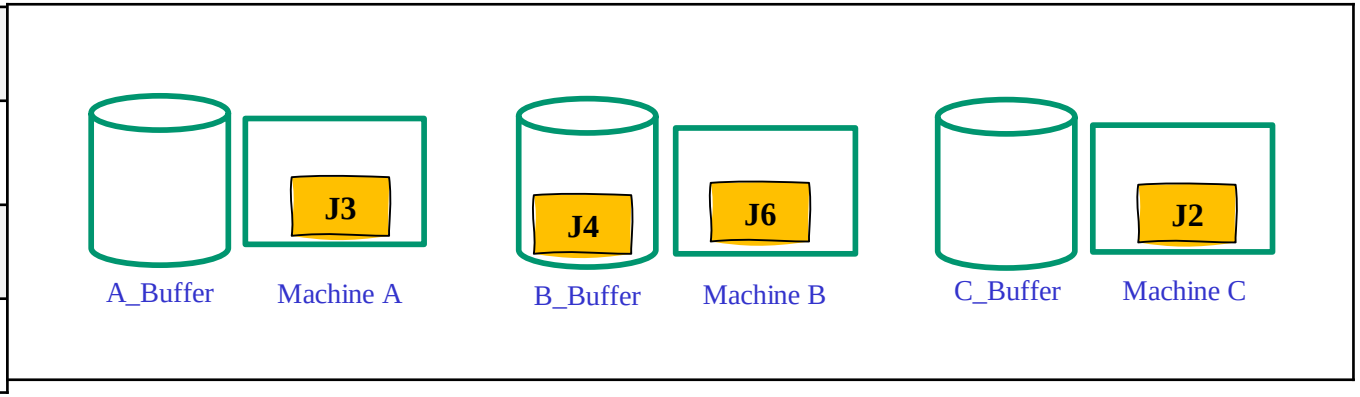


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

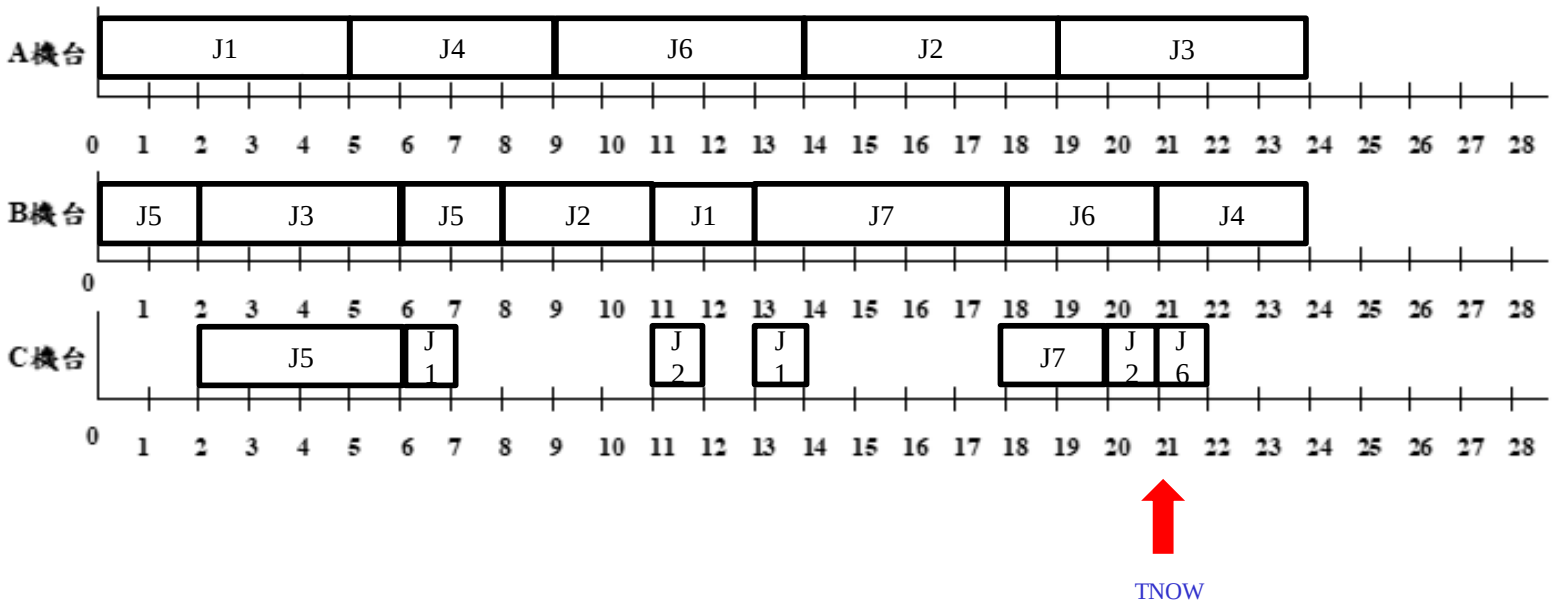
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	24
B_Complete	24
C_Complete	22

Event type
B_complete & C_complete
Time
21



Throughput : 4

零工式生產排程範例

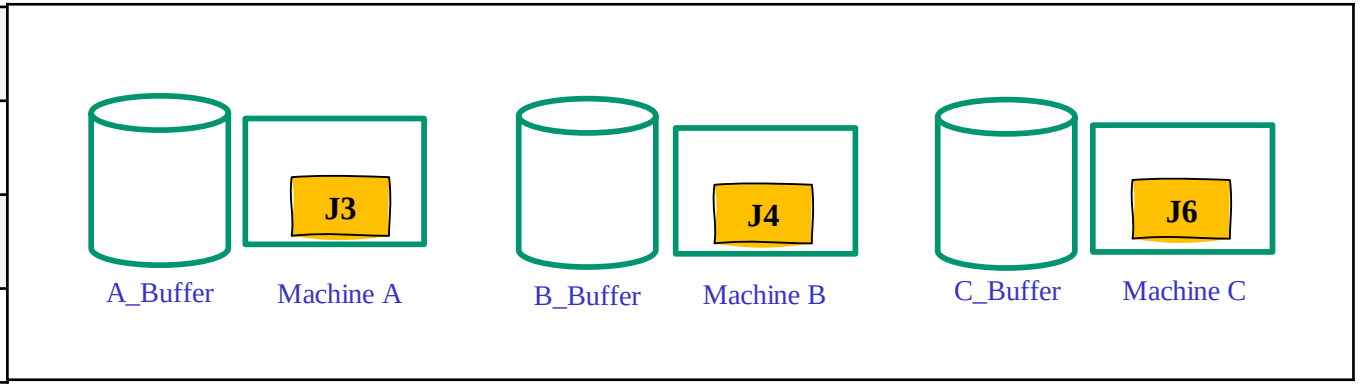


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

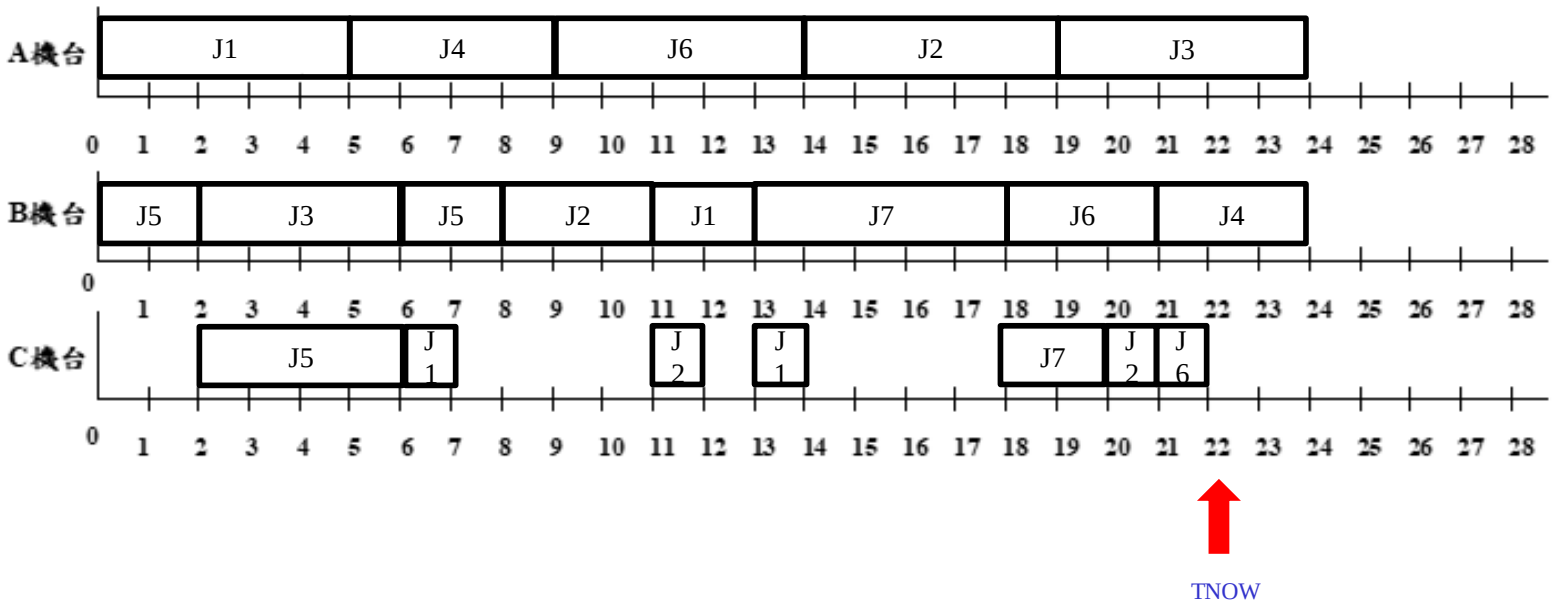
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	24
B_Complete	24
C_Complete	infinite

Event type
C_complete
Time
22



Throughput : 5

零工式生產排程範例

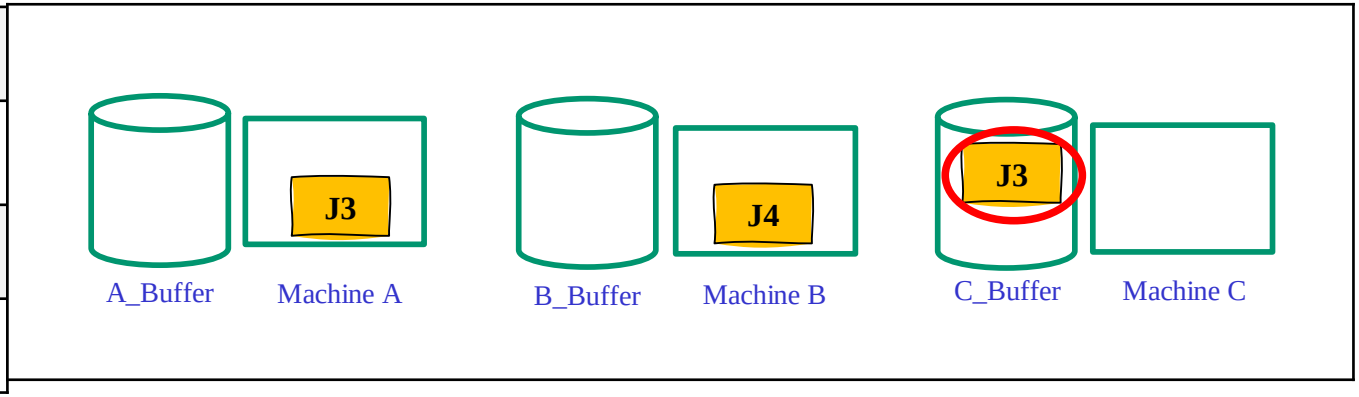


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

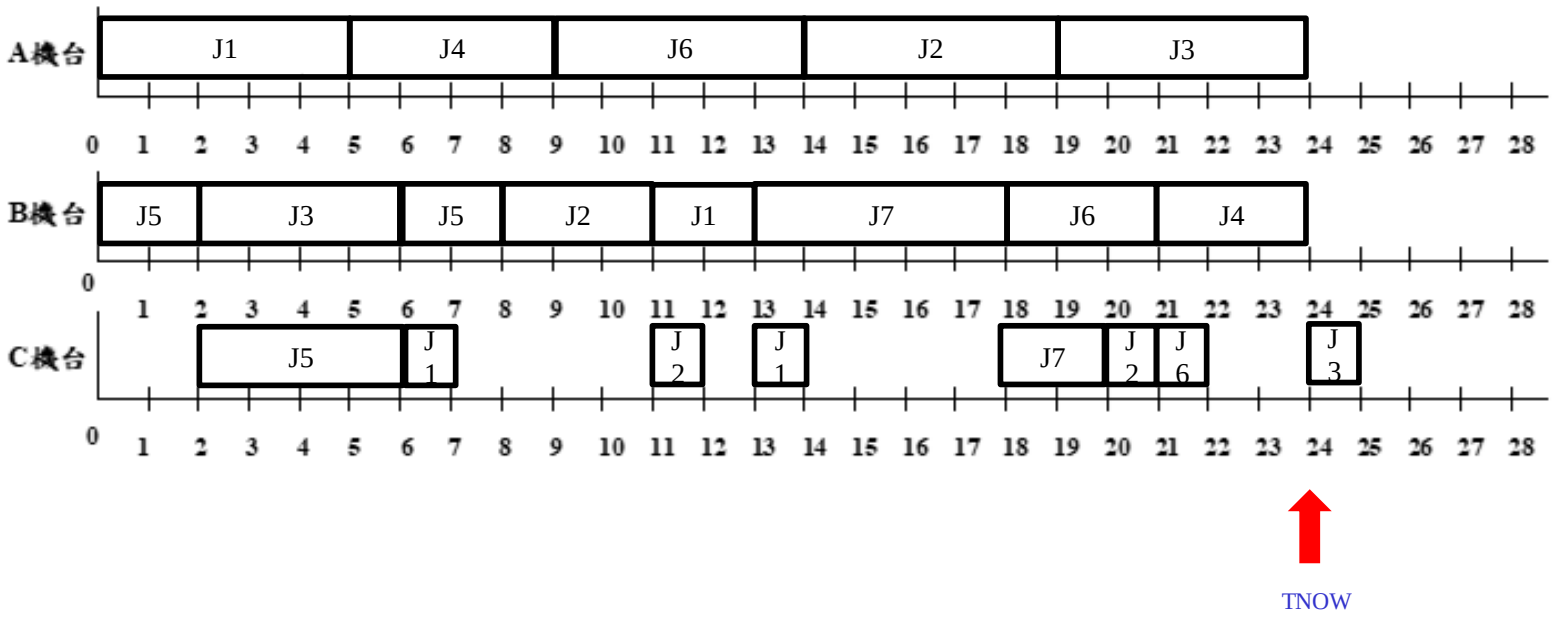
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	infinite
B_Complete	infinte
C_Complete	25

Event type
A_complete & B_complete
Time
24



Throughput : 5

零工式生產排程範例

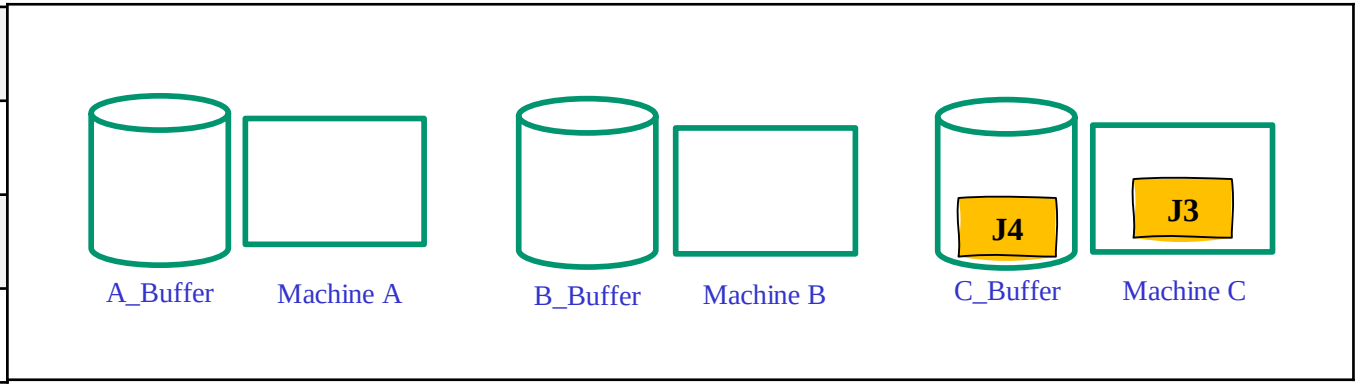


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

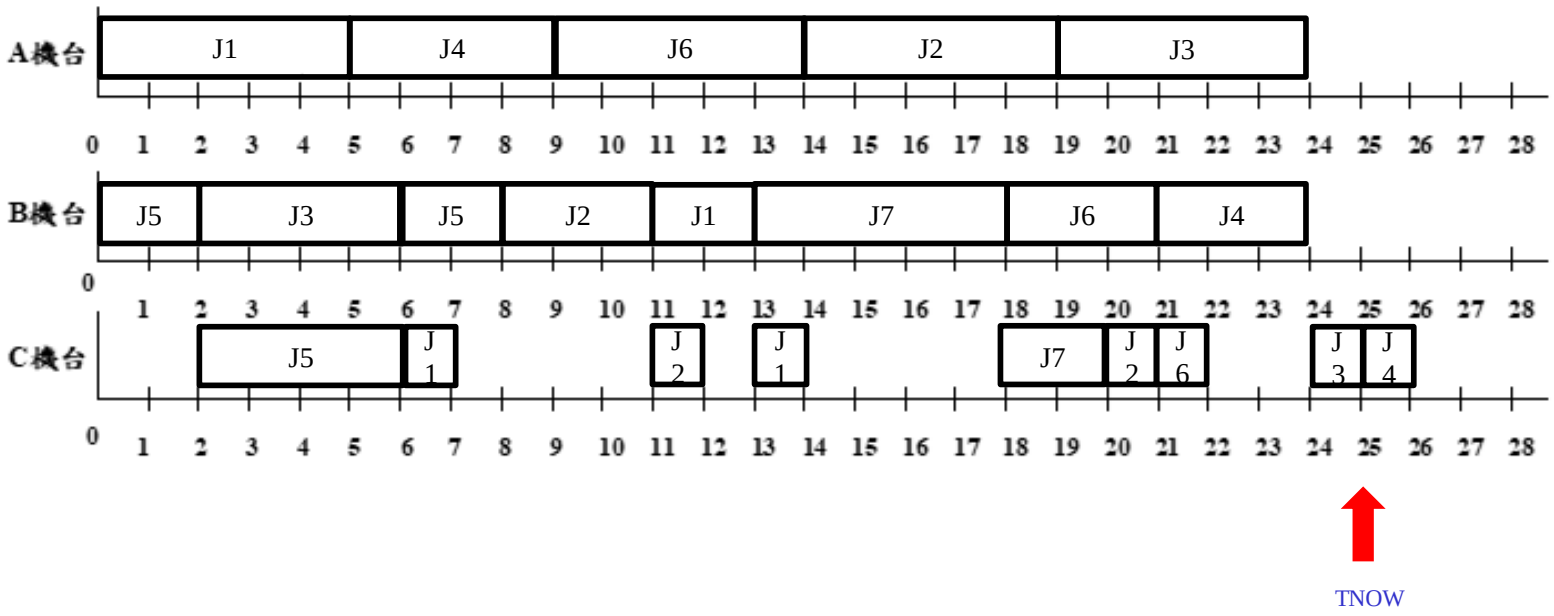
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	infinite
B_Complete	infinte
C_Complete	26

Event type
C_complete
Time
25



Throughput : 6

零工式生產排程範例

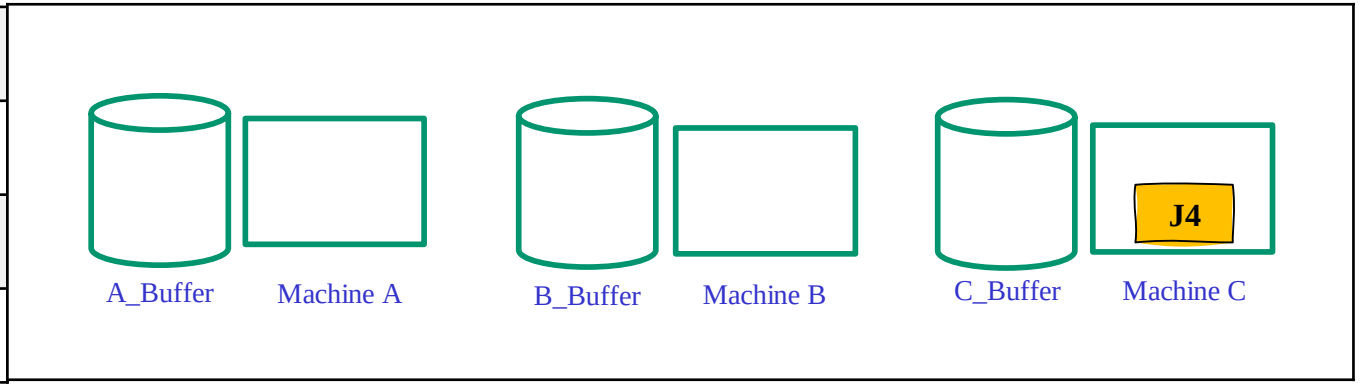


零工式生產排程範例

製單	來到時間	處理程序	交期(hrs)
J1	0	A(5hrs)→C(1hrs) →B(2hrs)→C(1hrs)	15
J2	4	B(3hrs)→C(1hrs)→A(5hrs)→C(1hrs)	14
J3	0	B(4hrs) → A(5hrs) → C(1hrs)	20
J4	0	A(4hrs) → B(3hrs) → C(1hrs)	24
J5	0	B(2hrs)→C(4hrs)→B(2hrs)	10
J6	7	A(5hrs) → B(3hrs) → C(1hrs)	16
J7	10	B(5hrs)→C(2hrs)	20

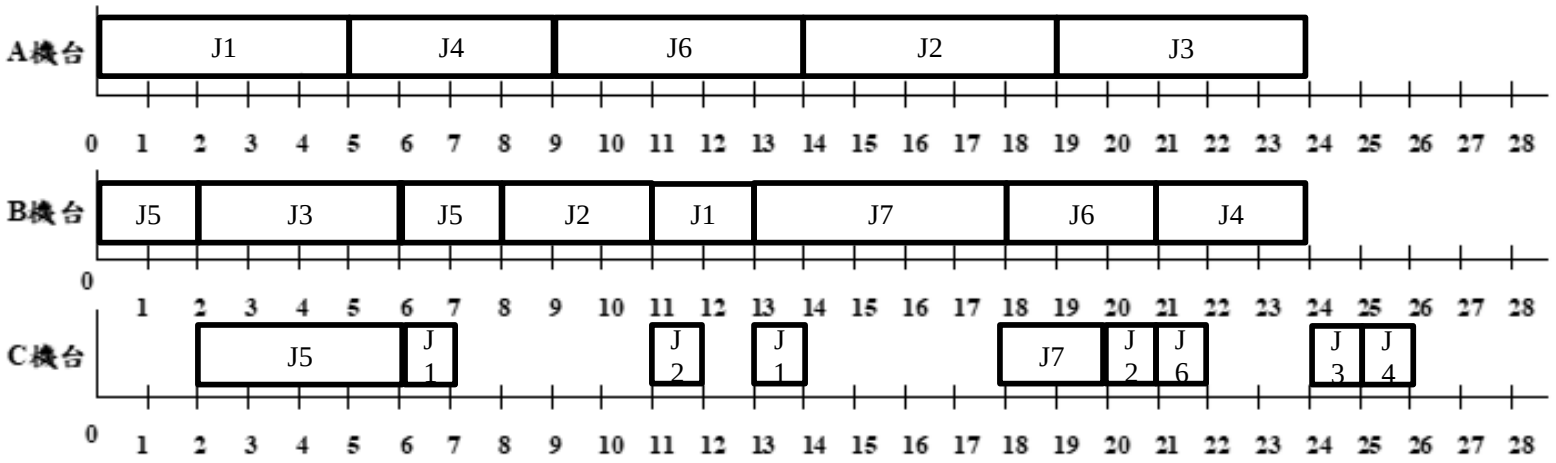
Future Event List	
Event type	**Time**
Arrival	infinite
A_Complete	infinite
B_Complete	infinte
C_Complete	infinite

Event type
C_complete
Time
26



Throughput : 7

零工式生產之排程結果



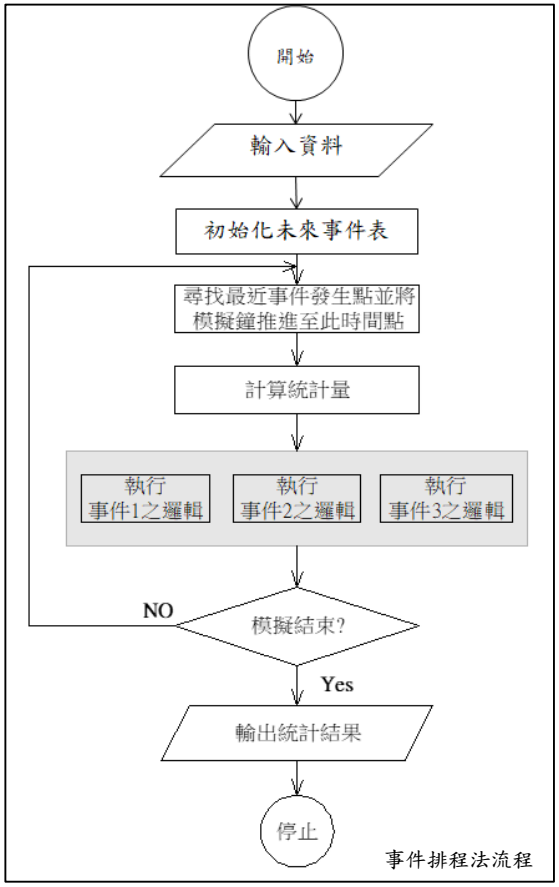
零工式生產排程手算之結果

製令編號	來到時間	完成加工時間	交期	在系統停留時間	Lateness	Tardiness
O1	0 hrs	14 hrs	15 hrs	14 hrs	-1 hrs	0 hrs
O2	4 hrs	21 hrs	14 hrs	17 hrs	7 hrs	7 hrs
O3	0 hrs	25 hrs	20 hrs	25 hrs	5 hrs	5 hrs
O4	0 hrs	26 hrs	24 hrs	26 hrs	2 hrs	2 hrs
O5	0 hrs	8 hrs	10 hrs	8 hrs	-2 hrs	0 hrs
O6	7 hrs	22 hrs	16 hrs	15 hrs	6 hrs	6 hrs
O7	10 hrs	20 hrs	20 hrs	10 hrs	0 hrs	0 hrs

Job shop scheduling
Step by step algorithm

Step by step Algorithm

1. 決定投料順序
2. 選擇派工法則
3. 事件排程法:
   1. 輸入資料
   2. 初始化未來事件表
   3. 尋找最近事件發生點並將時間推進
   4. 計算統計量
   5. 執行該事件種類之邏輯
      - ✓ 來到事件
      - ✓ 開始加工事件
      - ✓ 結束加工事件
   6. 判斷是否達成終止條件，否則重回 3.找最近事件
4. 輸出結果並繪製甘特圖



事件排程法流程

Implementation with Python

物件導向程式設計風格

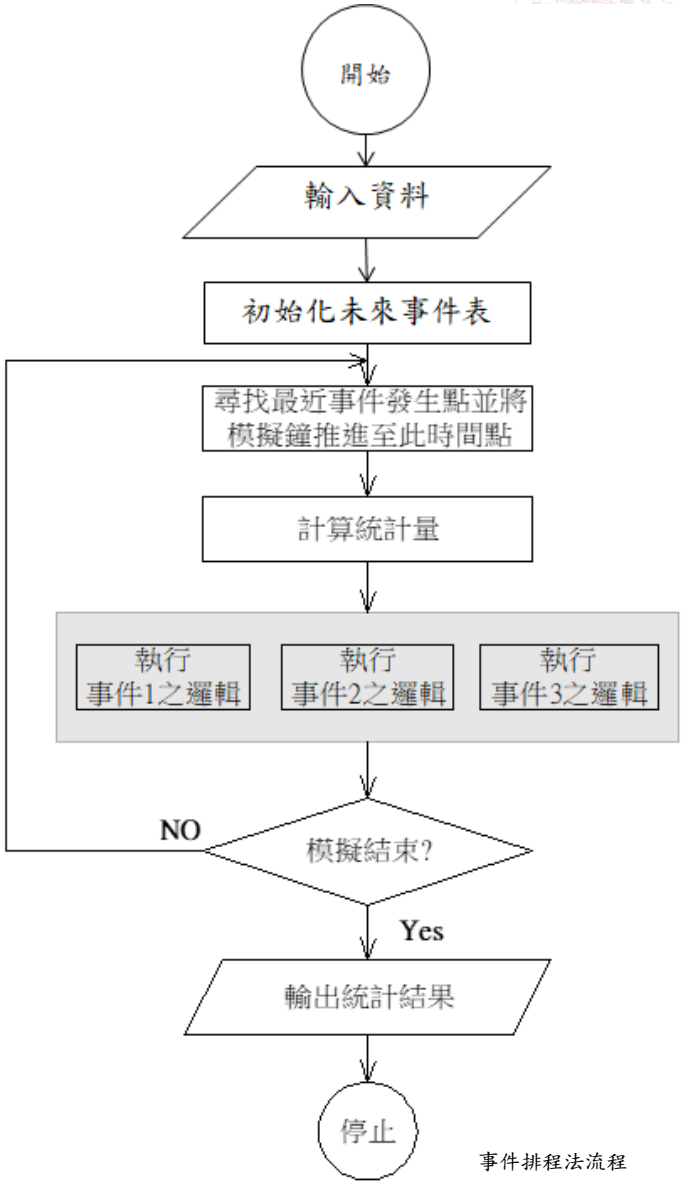
- **本範例之事件排程法使用物件導向程式設計風格**
  - 可參考 Python 學習筆記 #007：物件導向、檔案存取

- **物件導向目的**
  1. 可維護性：要更新時，只需改要改的東西
  2. 可複用性：可以後來重複使用
  3. 可擴展性：如果需增加條件，只需要另外再增加就可以
  4. 靈活性：可透過條件改變滿足需求

事件排程法邏輯

分析事件類別與對應物件：

1. 來到事件 ----- Source
2. 派工事件(開始加工事件) ----- Machine
3. 結束加工事件 ----- Machine



事件排程法流程

Class類別 & 物件應用

- 本例包含之 class 類別：
  - Order
  - Factory
  - Source
  - Machine

Class類別 & 物件應用

- **Order**:
  - Attribute:
    - Input
      - ID
      - due date
      - routing
    - progress
    - processing time
  - Statistics:
    - arrival time
    - finish time

```
#entity
class Order:
    def __init__(self, ID, AT, DD, routing, PT):
        self.ID    = ID
        self.AT    = AT          #AT: arrival time
        self.DD    = DD          #DD: due date

        self.PT       = PT       #PT: processing time
        self.routing  = routing
        self.progress = 0
```

- 先設定好需收集的統計
- 紀錄每張製令的來到時間, 完成時間

Class類別 & 物件應用

- **Source**:
  - Attribute:
    - order information
  - Statistics:
    - output
  - Method:
    - order arrival(arrival_event)

Input

```
class Source:
    def __init__(self, order_info):
        self.order_info = order_info
        self.output = 0

    def arrival_event(self, fac):
        raise NotImplementedError
```

- 此物件的屬性含所有製令資訊(表1)
- 此站需紀錄投料數量
- Method 執行各製令的來到事件

Class類別 & 物件應用

- Machine:
  - Attribute:
    - Input
      - ID
      - dispatching rule
    - state
    - buffer
    - working space
  - Method:
    - start processing
    - end processing

```
class Machine:
    def __init__(self, ID, DP_rule):
        self.ID     = ID
        self.state  = 'idle'
        self.buffer = []
        self.wspace = [] #wspace: working space
        self.DP_rule = DP_rule

    def start_processing(self, fac):
        raise NotImplementedError

    def end_process_event(self, fac):
        raise NotImplementedError
```

- 在此物件的屬性中建立兩個list，一作機台前之暫存區(buffer)，用以存放系統之半成品，另一個為機台的working space。
- 機台執行開始加工與結束工作事件。

Class類別 & 物件應用

- Factory:
  - Attribute:
    - Input
      - order information
      - dispatching rule
    - (future) event list
  - Statistics:
    - 系統產出
    - 各製令模擬結果
  - Method:
    - 建立Factory
    - 依event type執行事件
    - 執行下次事件直到stop
    - 更新統計量

```
class Factory:
    def __init__(self, order_info, DP_rule):
        self.order_info = order_info
        self.DP_rule    = DP_rule
        self.event_lst = pd.DataFrame(columns=["event_type", "time"])

        #statistics
        self.throughput = 0
        self.order_statistic = pd.DataFrame(columns = ["ID", "release_time",
                                                       "complete_time", "due_date",
                                                       "flow_time", "lateness",
                                                       "tardiness"])
        #[Plug in] tool of gantt plotting
        self.gantt_plot = Gantt()

        #build ur custom factory
        self.__build__()

    def __build__(self):
        raise NotImplementedError

    def initialize(self, order_info):
        raise NotImplementedError

    def next_event(self, stop_time):
        raise NotImplementedError

    def event(self, event_type):
        raise NotImplementedError

    def update_order_statistic(self, order):
        raise NotImplementedError
```

Class類別 & 物件應用

- Factory:
  - __build__(): 設定建造一個工廠所需的物件，例如：一個Source、三部機台。
  - 需提供更新order statistic的功能, 以計算平均flow time、lateness、tardiness
  - 使用gantt_plot，繪製排程結果之甘特圖。
- Components:
  - Source
  - Machine A
  - Machine B
  - Machine C

```
class Factory:
    def __init__(self, order_info, DP_rule):
        self.order_info = order_info
        self.DP_rule    = DP_rule
        self.event_lst = pd.DataFrame(columns=["event_type", "time"])

        #statistics
        self.throughput = 0
        self.order_statistic = pd.DataFrame(columns = ["ID", "release_time",
                                                       "complete_time", "due_date",
                                                       "flow_time", "lateness",
                                                       "tardiness"])
        #[Plug in] tool of gantt plotting
        self.gantt_plot = Gantt()

        #build ur custom factory
        self.__build__()
```

```
def __build__(self):
    self.source   = Source(self.order_info)
    self.machines = {'A': Machine('A', self.DP_rule),
                     'B': Machine('B', self.DP_rule),
                     'C': Machine('C', self.DP_rule)}
```

```
    def event(self, event_type):
        raise NotImplementedError

    def update_order_statistic(self, order):
        raise NotImplementedError
```

python語言之模擬模式構建

主程式依照step by step演算法實現:

```
if __name__ == '__main__':
    #read the input data sheet
    data_dir = os.getcwd() + "/data/"
    order_info = pd.read_excel(data_dir + "order_information.xlsx")

    #data preprocessing
    order_info = order_info.sort_values(['arrival_time']).reset_index(drop=True)

    #choose the dispatching policy u'd like
    DP_rule = 'EDD' # 'SPT' #

    #build the factory
    fac = Factory(order_info, DP_rule)

    #start the simulation
    fac.next_event(stop_time)

    #output result
    print(fac.order_statistic)
    print("Makespan = ", fac.makespan)
    fac.gantt_plot.draw_gantt()
```

1. 讀取製令資訊表，將製令依照來到時間排序
2. 選擇適合的派工法則
3. 建立欲模擬之生產系統，並開始事件排程法
4. 輸出排程結果與系統績效

python語言之模擬模式構建

離散事件排程法:

```
def next_event(self, stop_time):
    global T_NOW, T_LAST
    T_NOW, T_LAST = infinity, infinity
    self.initialize(self.order_info)
    T_NOW      = self.event_lst.min()["time"]
    event_type = self.event_lst['time'].astype(float).idxmin()

    while T_NOW < stop_time:
        self.event(event_type)
        T_LAST     = T_NOW
        T_NOW      = self.event_lst.min()["time"]
        event_type = self.event_lst['time'].astype(float).idxmin()

    T_NOW = stop_time
    self.makespan = T_LAST
```

```
def initialize(self, order_info):
    self.event_lst.loc[0] = ["Arrival", order_info.loc[0, "arrival_time"]]
    self.event_lst.loc[1] = ["A_complete",  infinity]
    self.event_lst.loc[2] = ["B_complete",  infinity]
    self.event_lst.loc[3] = ["C_complete",  infinity]
    self.event_lst.loc[4] = ["dispatching", infinity]
    self.event_lst = self.event_lst.set_index('event_type')
```

```
def event(self, event_type):
    #Arrival event
    if event_type == 'Arrival':
        self.source.arrival_event(self)

    #Complete event
    elif event_type == 'A_complete':
        self.machines['A'].end_process_event(self)
    elif event_type == 'B_complete':
        self.machines['B'].end_process_event(self)
    elif event_type == 'C_complete':
        self.machines['C'].end_process_event(self)

    #Dispatch event
    else:
        for mc in self.machines.values():
            mc.start_processing(self)
        self.event_lst.loc["dispatching"]['time'] = infinity
```

python語言之模擬模式構建

來到事件:

```
def arrival_event(self, fac):
    order_num = self.order_info.shape[0] #num of total orders

    #generate and release the order
    ID      = self.order_info.loc[self.output, "ID"]
    routing = self.order_info.loc[self.output, "routing"].split(',')
    PT      = [int(i) for i in self.order_info.loc[self.output, "process_time"].split(',')]
    DD      = self.order_info.loc[self.output, "due_date"]
    AT      = T_NOW
    order   = Order(ID, AT, DD, routing, PT)
    if LOG == True:
        print("{} : order {} release.".format(T_NOW, order.ID))

    self.output += 1

    #update the future event list - next order arrival event
    if self.output < order_num:
        fac.event_lst.loc["Arrival"]["time"] = self.order_info.loc[self.output, "arrival_time"]
    else:
        fac.event_lst.loc['Arrival']['time'] = infinity

    #send order to correlated station
    target = order.routing[order.progress]
    machine = fac.machines[target]
    machine.buffer.append(order)

    #update the future event list - dispatch machines to process the jobs
    if machine.state == 'idle':
        fac.event_lst.loc["dispatching"]['time'] = T_NOW
```

從資訊表讀取將來到之製令資訊(ID, routing, DD etc.)
並生成Order
更新投料數量

預排下一張製令的來到事件

判斷來到之製令的第一站為哪部機台，送往其暫存區，並執行以下邏輯：

IF <機台有空> {
<安排派工事件> }
ELSE {
<製令在暫存區等待> }

python語言之模擬模式構建

派工事件:

```
#Dispatch event

    for mc in self.machines.values():
        mc.start_processing(self)
    self.event_lst.loc["dispatching"]['time'] = infinity
```

待所有工件都來到後，各機台執行**開始加工事件**，並更新 future event list

python語言之模擬模式構建

開始加工事件:

```
def start_processing(self, fac):
    #check state
    if self.state == 'idle':
        #get a new order from buffer by DP_rule
        if len(self.buffer) > 0:
            if self.DP_rule == "FIFO":
                order = self.buffer[0]
            elif self.DP_rule == "EDD":
                idx = np.argmin([j.DD for j in self.buffer])
                order = self.buffer[idx]
            elif self.DP_rule == "SPT":
                idx = np.argmin([j.PT[j.progress] for j in self.buffer])
                order = self.buffer[idx]

            #remove order from buffer
            self.buffer.remove(order)

            #start processing the order
            self.wspace.append(order)
            self.state = 'busy'
            processing_time = order.PT[order.progress]

            #[Gantt plot preparing] udate the start/finish processing time of machine
            fac.gantt_plot.update_gantt(self.ID, T_NOW, processing_time, order.ID)
            if LOG == True:
                print("{} : machine {} start processing order {} - {} progress".format(T_NOW, se

            #update the future event list - job complete event
            fac.event_lst.loc["{}_complete".format(self.ID)]['time'] = T_NOW + processing_time
            order.progress += 1
```

IF<機台閒置>
IF<暫存區有製令待加工>
{
<依派工法則選擇製令>
<將該製令從buffer移除>
<將該製令加入工作區>
<讀取製令之加工時間>
<更改機台狀態為**忙碌**>
}

紀錄機台的加工資訊，以繪製甘特圖

預排該機台的結束加工事件與更新製令的加工進度

python語言之模擬模式構建

結束加工事件:

```
def end_process_event(self, fac):
    order = self.wspace[0]


    self.wspace.remove(order)
    self.state = 'idle'

    #send the processed order to next place
    if order.progress >= len(order.routing):
        #update factory statistic
        fac.throughput += 1
        #update order statistic
        fac.update_order_statistic(order)
    else:
        #send the order to next station
        target = order.routing[order.progress]
        next_machine = fac.machines[target]
        next_machine.buffer.append(order)

    #update the future event list - wait for the dispatching to get a new job
    fac.event_lst.loc["dispatching"]['time'] = T_NOW
    fac.event_lst.loc["{}_complete".format(self.ID)]["time"] = infinity
```

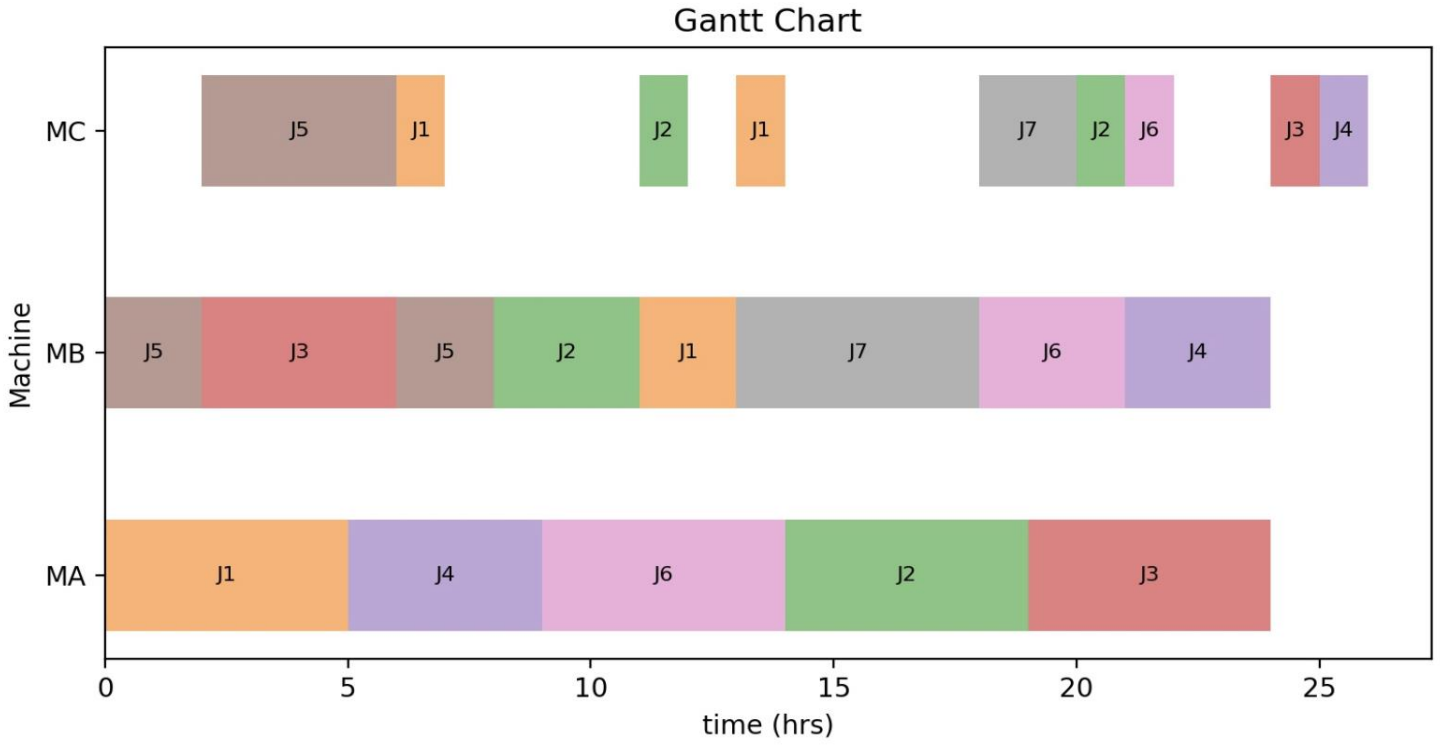
<工作區移除完成加工之製令>
<更改機台狀態為**閒置**>

IF<該製令完成所有加工進度>
{
  <更新系統產出量>
  <計算製令之加工結果>
}
ELSE
{
  <判斷製令的下一加工站>
  <將其送往目標機台暫存區>
 }

<安排派工事件>
<更新future event list>

Result

Output a gantt chart and the result of statistics:



ID	release_time	complete_time	due_date	flow_time	tardiness	lateness
5.0	0.0	8.0	10.0	8.0	0.0	-2.0
1.0	0.0	14.0	15.0	14.0	0.0	-1.0
7.0	10.0	20.0	20.0	10.0	0.0	0.0
2.0	4.0	21.0	14.0	17.0	7.0	7.0
6.0	7.0	22.0	16.0	15.0	6.0	6.0
3.0	0.0	25.0	20.0	25.0	5.0	5.0
4.0	0.0	26.0	24.0	26.0	2.0	2.0

演算法之優缺點

- 優點：

  利用離散事件模擬邏輯來排程，因其時間推進的概念，使得驗證步驟變更容易。

- 缺點：

  為了實現時間推進機制，需要預先建立未來事件表，且一個事件類別便需要一列(如圖)。若機台變多，事件表也會隨著變大，coding時需逐項列出，較麻煩且無彈性。

Future Event List	
Event type	**Time**
Arrival	4
A_Complete	5
B_Complete	2
C_Complete	infinite

- 改善方法：

  利用simPy套件配合timeout與callback function實現時間推進機制，可以省去手動建表步驟，並使model更加彈性。