Lý thuyết hệ điều hành

Giảng viên: TS. Hà Chí Trung

Bộ môn: Khoa học máy tính

Khoa: Công nghệ thông tin

Học viện Kỹ thuật quân sự

Email: hct2009@yahoo.com

Mobile: 01685.582.102

Chương 2. Tiến trình và luồng

2.1. Tổng quan về tiến trình và luồng

2.2. Các trạng thái của tiến trình

2.3. Điều khiển tiến trình

2.4. Tắc nghẽn (deadlock) và tránh tắc nghẽn

2.5. Lập lịch cho CPU

2.6. Bài tập phần Quản lý tiến trình

2.1. Tổng quan về tiến trình và luồng

2.1.1. Tài nguyên hệ thống

2.1.2. Lời gọi hệ thống (System Calls)

2.1.3. Khái niệm về tiến trình (Proccess)

2.1.4. Phân loại tiến trình

2.1.5. Sự thực thi của tiến trình

2.1.1. Tài nguyên hệ thống

- **Tài nguyên hệ thống (System Resources)**:
  - **Tài nguyên không gian:** là các không gian lưu trữ của hệ thống như đĩa, bộ nhớ chính, quan trọng nhất là không gian bộ nhớ chính, nơi lưu trữ các chương trình đang được CPU thực hiện.
    - **VD:** Bộ nhớ, CPU, tài nguyên ảo, máy in,…
  - **Tài nguyên thời gian:** chính là thời gian thực hiện lệnh của processor và thời gian truy xuất dữ liệu trên bộ nhớ.

2.1.1. Tài nguyên hệ thống

- Trên khía cạnh cấp phát tài nguyên, tài nguyên hệ thống được chia thành 2 loại:
  - **Tài nguyên phân chia được:** tại một thời điểm nó có thể cấp phát cho nhiều tiến trình khác nhau, các tiến trình song song có thể đồng thời sử dụng chúng.
    - VD: Bộ nhớ chính và processor, bởi tại một thời điểm có thể có nhiều tiến trình cùng chia nhau sử dụng không gian lưu trữ của bộ nhớ chính và thời gian xử lý của processor.
  - **Tài nguyên không phân chia được:** tại một thời điểm nó chỉ có thể cấp phát cho một tiến trình duy nhất. VD: máy in.

2.1.2. Lời gọi hệ thống (System Calls)

- Để tạo môi trường giao tiếp giữa chương trình và hệ điều hành, hệ điều hành đưa ra các **lời gọi hệ thống (system calls).** Chương trình dùng các lời gọi hệ thống để liên lạc với hệ điều hành và yêu cầu các dịch vụ từ hệ điều hành.
- Lời gọi hệ thống có thể được chia thành các loại:
  - điều khiển tiến trình;
  - thao tác trên tập tin;
  - thao tác trên thiết bị vào/ ra;
  - thông tin liên tiến trình, ...

2.1.2. Lời gọi hệ thống (System Calls)

- **Lời gọi hệ thống điều khiển tiến trình:**
  - Kết thúc, huỷ bỏ tiến trình (**end, abort**);
  - Nạp và thực hiện tiến trình (**load, execute**);
  - Tạo và dừng tiến trình (**create, terminate**)
  - Đọc, đặt thuộc tính của tiến trình (**get/set attributes**);
  - Đợi (**wait**);
  - Cấp và thu hồi bộ nhớ (**allocate/ deallocate**)
- Quá trình hoạt động của tiến trình là quá trình chuyển từ trạng thái này sang trạng thái khác do sự tác động từ bên ngoài, cụ thể là **bộ điều phối tiến trình (dispatcher)** của HĐH.

2.1.2. Lời gọi hệ thống (System Calls)

- Mỗi lời gọi hệ thống tương ứng với một **thủ tục** trong thư viện của hệ điều hành (**DLL- Dynamic Link Liblary**).
- Ví dụ, trong window9x:
  - **Kernel32.DLL:** Hạt nhân của windows, hỗ trợ cho những chức năng ở mức thấp mà một ứng dụng cần để chạy, chẳng nếu ứng dụng cần bộ nhớ.
  - **GDI32.DLL:** Giao diện thiết bị đồ họa của windows, nó thực hiện các chức năng về Font chữ, máy in, màn hình, ...
  - **User32.DLL:** Giao tiếp người sử dụng.

2.1.3. Khái niệm về tiến trình (proccess)

- **Tiến trình** là một chương trình đang được thực thi (đã được nạp vào bộ nhớ chính).
- ***Phân biệt:*** *Chương trình là một tập tin thụ động nằm trên đĩa, tiến trình là trạng thái động của chương trình.*
- Để một tiến trình đi vào trạng thái hoạt động thì HĐH phải cung cấp và duy trì đầy đủ **tài nguyên hệ thống** cho nó trong suốt quá trình hoạt động: CPU, bộ nhớ chính, các tập tin và thiết bị nhập/xuất,…
- Để có nhiều dòng xử lý công việc có thể cùng chia sẻ 1 không gian địa chỉ và hoạt động song song với nhau, các hệ điều hành hiện nay đưa ra một cơ chế thực thi (các chỉ thị trong chương trình), gọi là tiểu trình (**thread**).

2.1.3. Khái niệm về tiến trình (proccess)

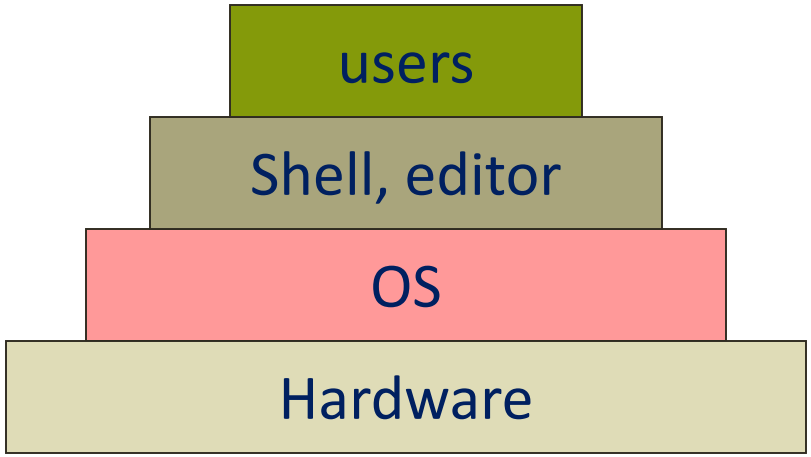
- Tiến trình bao gồm 3 thành phần: **Code, Data, Stack**
  - **Code**: Thành phần câu lệnh thực hiện;
  - **Data**: Thành phần dữ liệu;
  - **Stack**: Thành phần lưu thông tin tạm thời.
- Các câu lệnh trong code chỉ dùng data và stack riêng của mình ngoại trừ các vùng dùng chung.
- Tiến trình được hệ thống phân biệt bằng số hiệu **pid** (**proccess indentification**).
- Tiến trình là đơn vị làm việc cơ bản của hệ thống. Trong một tiến trình có thể có nhiều tiểu trình (**thread**: luồng).
- Tiểu trình là một đơn vị xử lý cơ bản trong hệ thống, nó hoàn toàn tương tự như tiến trình.

2.1.3. Khái niệm về tiến trình (proccess)

- **Thread**: Các tiểu trình trong một tiến trình chia sẻ một không gian địa chỉ chung:
  - Có các trạng thái như một tiến trình thật;
  - chia sẻ các biến toàn cục của tiến trình;
  - có thể truy xuất đến stack của tiểu trình khác trong cùng tiến trình…
- **Bộ xử lý lệnh (Shell):**
  - **Shell** là một bộ phận hay một tiến trình đặc biệt của hệ điều hành, có nhiệm vụ nhận lệnh của người sử dụng, phân tích lệnh và phát sinh tiến trình mới để thực hiện yêu cầu của lệnh, tiến trình mới này được gọi là tiến trình đáp ứng yêu cầu.
  - **VD**: tập tin command.com là Shell của **MS DOS**.

2.1.4. Phân loại tiến trình

- **Đối với người sử dụng:**
  - Các tiến trình của HĐH hoạt động trong chế độ đặc quyền, có thể truy xuất vào các vùng dữ liệu được bảo vệ.
  - Các tiến trình người sử dụng hoạt động trong chế độ không đặc quyền. Các tiến trình người sử dụng chỉ có thể truy xuất vào hệ thống thông qua các tiến trình của hệ điều hành bằng cách thực hiện một **lời gọi hệ thống.**



Chế độ không đặc quyền

Chế độ đặc quyền

2.1.4. Phân loại tiến trình

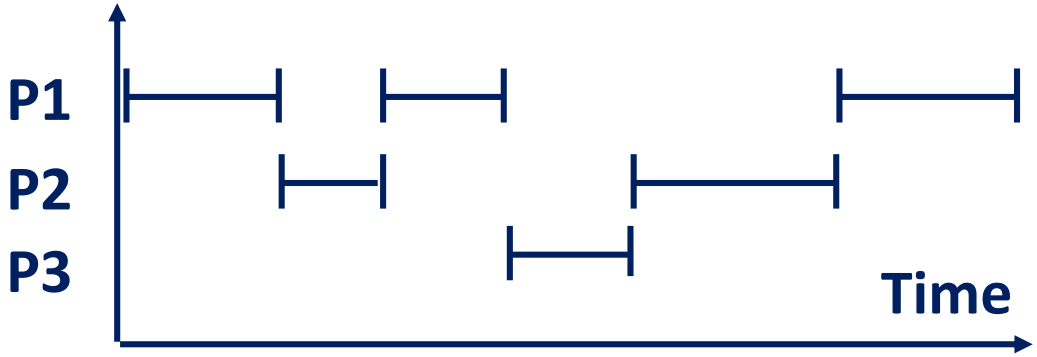
- **Đối với hệ điều hành:**
  - **Tiến trình tuần tự:** điểm khởi tạo của tiến trình này là điểm kết thúc của tiến trình trước đó.
  - **Tiến trình song song:** điểm khởi tạo của tiến trình này có thể nằm ở thân của các tiến trình khác.
- Tiến trình song song được chia thành nhiều loại:
  - Tiến trình song song độc lập;
  - Tiến trình song song có quan hệ thông tin;
  - Tiến trình song song phân cấp;
  - Tiến trình song song đồng mức;

2.1.5. Sự thực thi của tiến trình

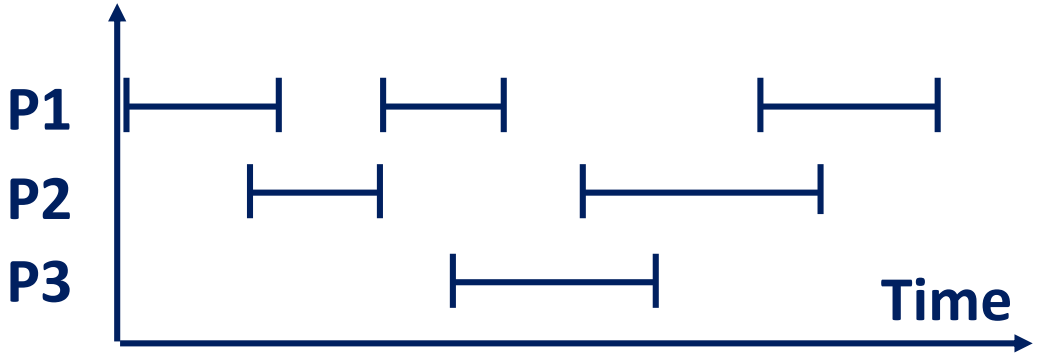
- Về bản chất, việc thực hiện một tiến trình là việc điều khiển **processor** để nó thực hiện xen kẽ các chỉ thị (**machine instrustion**) bên trong tiến trình. Điều này có thể thực hiện bằng cách thay đổi hợp lý giá trị của con trỏ lệnh (**vd**: **cặp thanh ghi CS:IP** trong các **processor** thuộc kiến trúc Intel) để con trỏ lệnh chỉ đến các chỉ thị cần thực hiện trong các tiến trình.

2.1.5. Sự thực thi của tiến trình

- Sự thực thi của các tiến trình song song trong hệ thống một nhân và đa nhân:



a. Trong hệ thống uniprocessor



b. Trong hệ thống Multiprocessor

2.2. Các trạng thái của tiến trình

2.2.1. Tiến trình hai trạng thái

2.2.2. Tiến trình ba trạng thái

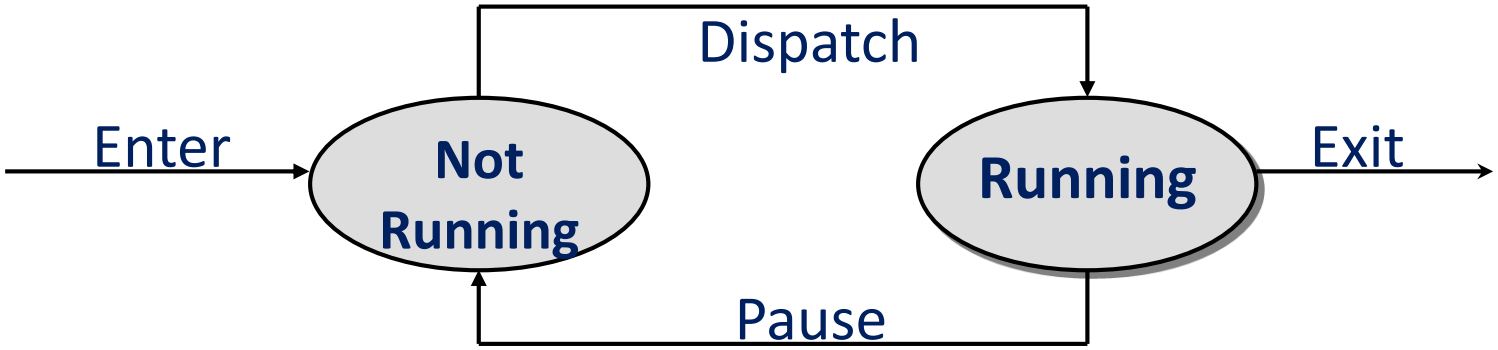
2.2.3. Tiến trình bốn trạng thái

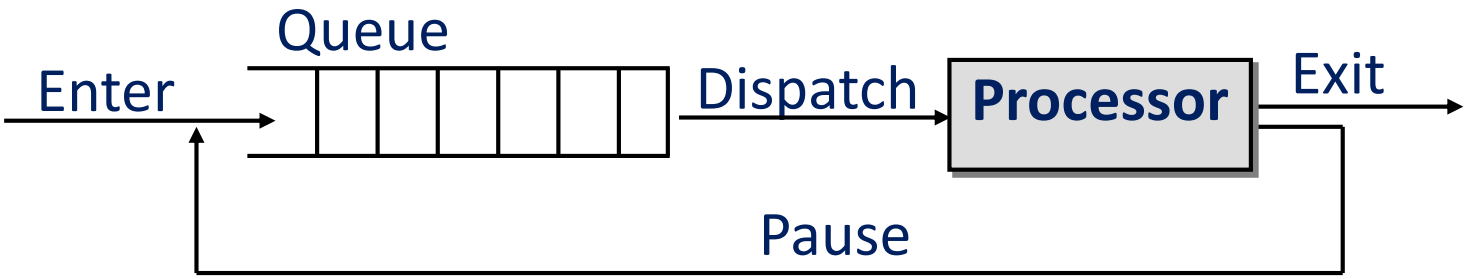
2.2.4. Tiến trình năm trạng thái

2.2.5. Các trạng thái của tiến trình

2.2.1. Tiến trình hai trạng thái

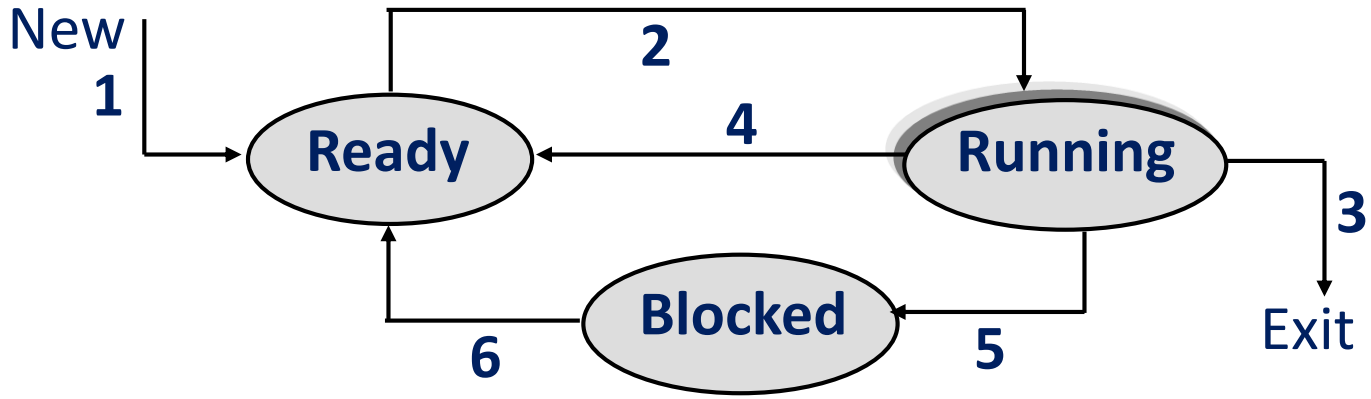
- Một số hệ điều hành chỉ cho phép tiến trình có 2 trạng thái: **thực thi** và **không thực thi.**





2.2.2. Tiến trình ba trạng thái

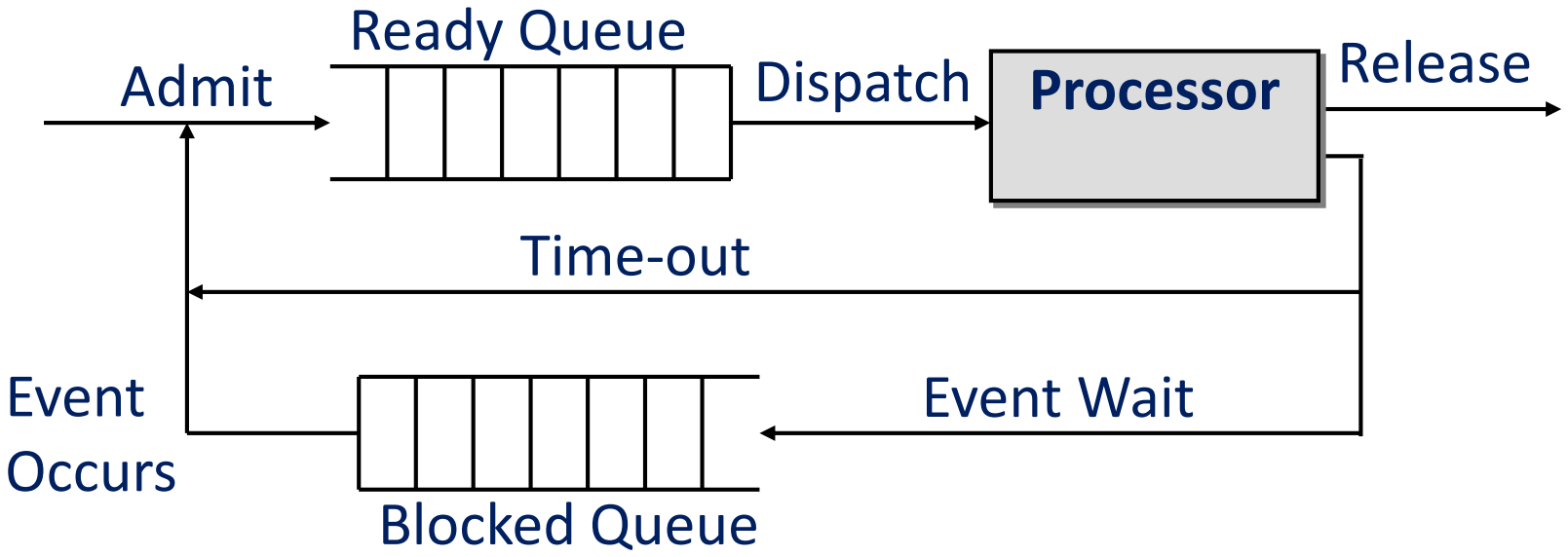
- Đa số hệ điều hành đều cho phép tiến trình tồn tại ở một trong ba trạng thái, đó là: **ready, running, blocked.**



- Bộ phận điều phối thu hồi proccessor trong các trường hợp:
  - Tiến trình đang thực hiện hết thời gian (time-out) được quyền sử dụng processor;
  - Có một tiến trình mới phát sinh và có độ ưu tiên cao hơn tiến trình hiện tại hoặc có thời gian thực thi nhỏ hơn nhiều so với khoảng thời gian còn lại mà tiến trình hiện tại cần.

2.2.2. Tiến trình ba trạng thái

- Tại một thời điểm xác định có thể có nhiều tiến trình đang ở trạng thái **Ready** hoặc **Blocked** nhưng chỉ có một ở trạng thái **Running**. Các tiến trình ở trạng thái **Ready** và Blocked được chứa trong các hàng đợi (**Queue**) riêng.

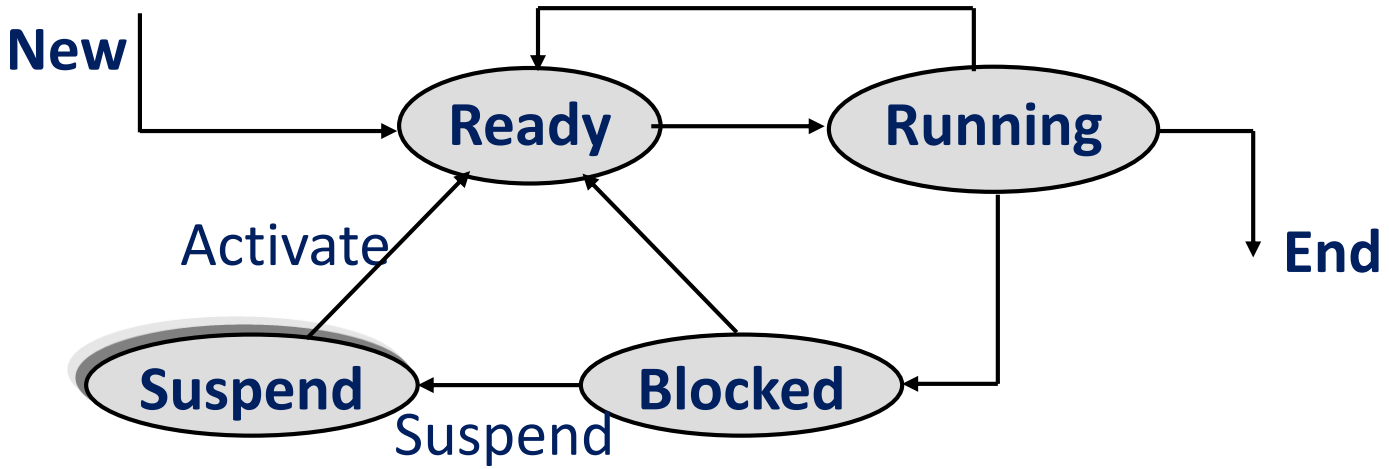


2.2.2. Tiến trình ba trạng thái

- **Trong sơ đồ trên:**
  1. (**Admit**) Tiến trình được khởi tạo, được đưa vào hệ thống, được cấp phát đầy đủ tài nguyên (trừ processor);
  2. (**Dispatch**) Tiến trình được cấp processor để bắt đầu thực hiện/ xử lý.
  3. (**Release**) Tiến trình hoàn thành xử lý và kết thúc.
  4. (**Time_out**) Tiến trình bị bộ điều phối tiến trình thu hồi processor, do hết thời gian được quyền sử dụng processor, để cấp phát cho tiến trình khác.
  5. (**Event wait**) Tiến trình chờ sự kiện nào đó xảy ra.
  6. (**Event Occurs**) Sự kiện mà tiến trình chờ đã xảy ra.

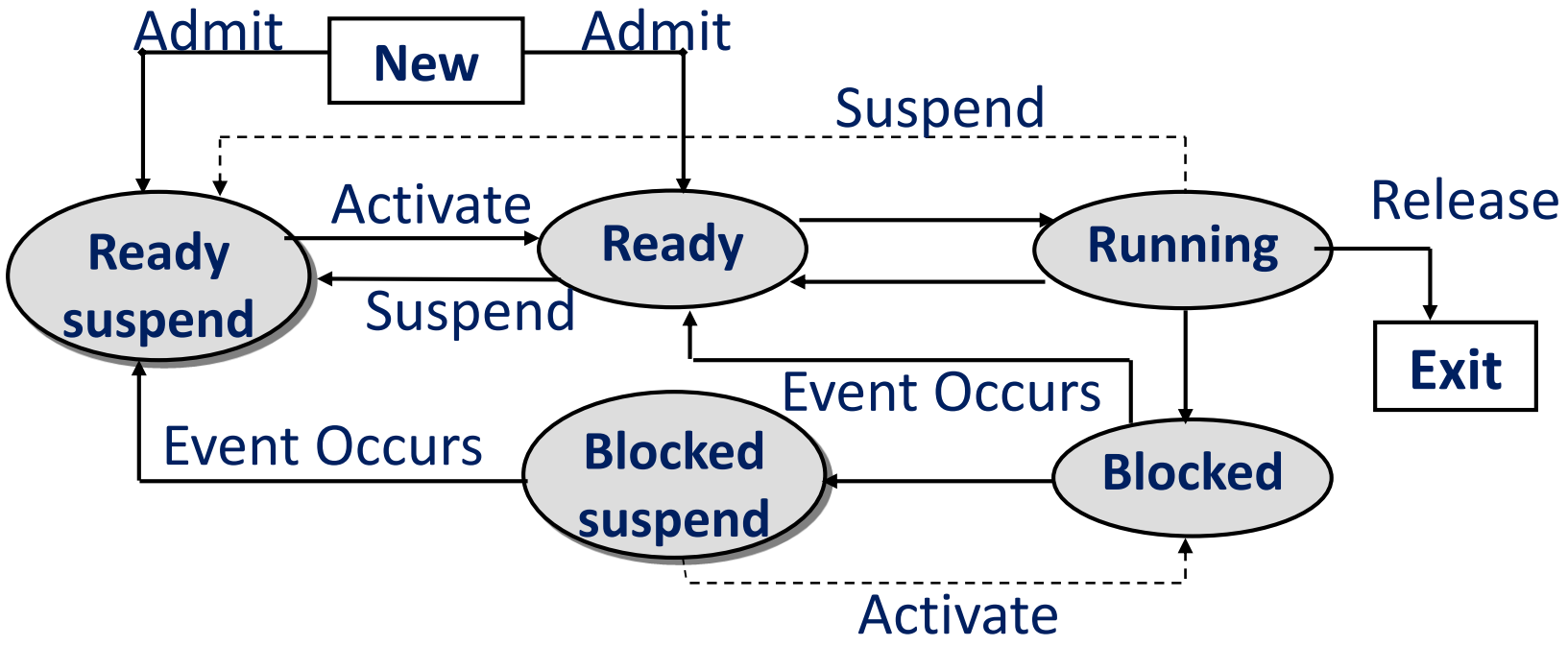
2.2.3. Tiến trình bốn trạng thái

- **Tiến trình 4 trạng thái:** các hệ thống sử dụng kỹ thuật **Swap** trong việc cấp phát bộ nhớ cho các tiến trình.



2.2.4. Tiến trình năm trạng thái

- **Tiến trình 5 trạng thái:** phân biệt 2 trạng thái suspend:
  - một trạng thái suspend dành cho các tiến trình từ blocked chuyển đến, được gọi là **blocked-suspend;**
  - một trạng thái suspend dành cho các tiến trình từ ready chuyển đến, được gọi là **ready-suspend.**



2.2.5. Các trạng thái của tiến trình

- **Trạng thái của tiến trình**:
  - **Khởi tạo (New)**: tiến trình vừa được tạo lập;
  - **Sẵn sàng (Ready)**: tiến trình đã được cấp phát đầy đủ tài nguyên (trừ processor) và đang đợi trong hàng đợi (**ready queue**) chờ chiếm dụng CPU để tiếp tục;
  - **Thực thi (Running)**: tiến trình đang được xử lý bởi CPU (đang chiếm dụng CPU);
  - **Bị chặn - đợi (Blocked, Waiting)**: tiến trình tạm dừng và chờ vì thiếu tài nguyên hay chờ 1 sự kiện nào đó;
  - **Đình chỉ, treo (Suspend):** là trạng thái của một tiến trình khi nó đang được lưu trữ trên bộ nhớ phụ.

2.2.5. Các trạng thái của tiến trình

- **Trạng thái của tiến trình**:
  - **Blocked-suspend:** tiến trình đang bị chứa trên bộ nhớ phụ (đĩa) và đang đợi một sự kiện nào đó.
  - **Ready-suspend:** tiến trình đang bị chứa trên bộ nhớ phụ nhưng sẵn sàng thực hiện ngay sau khi được nạp.
  - **Dừng (Halt)**: Tiến trình đã hoàn tất.

2.3. Điều khiển tiến trình

2.3.1. Khối điều khiển tiến trình

2.3.2. Các thao tác điều khiển tiến trình

2.3.1. Khối điều khiển tiến trình

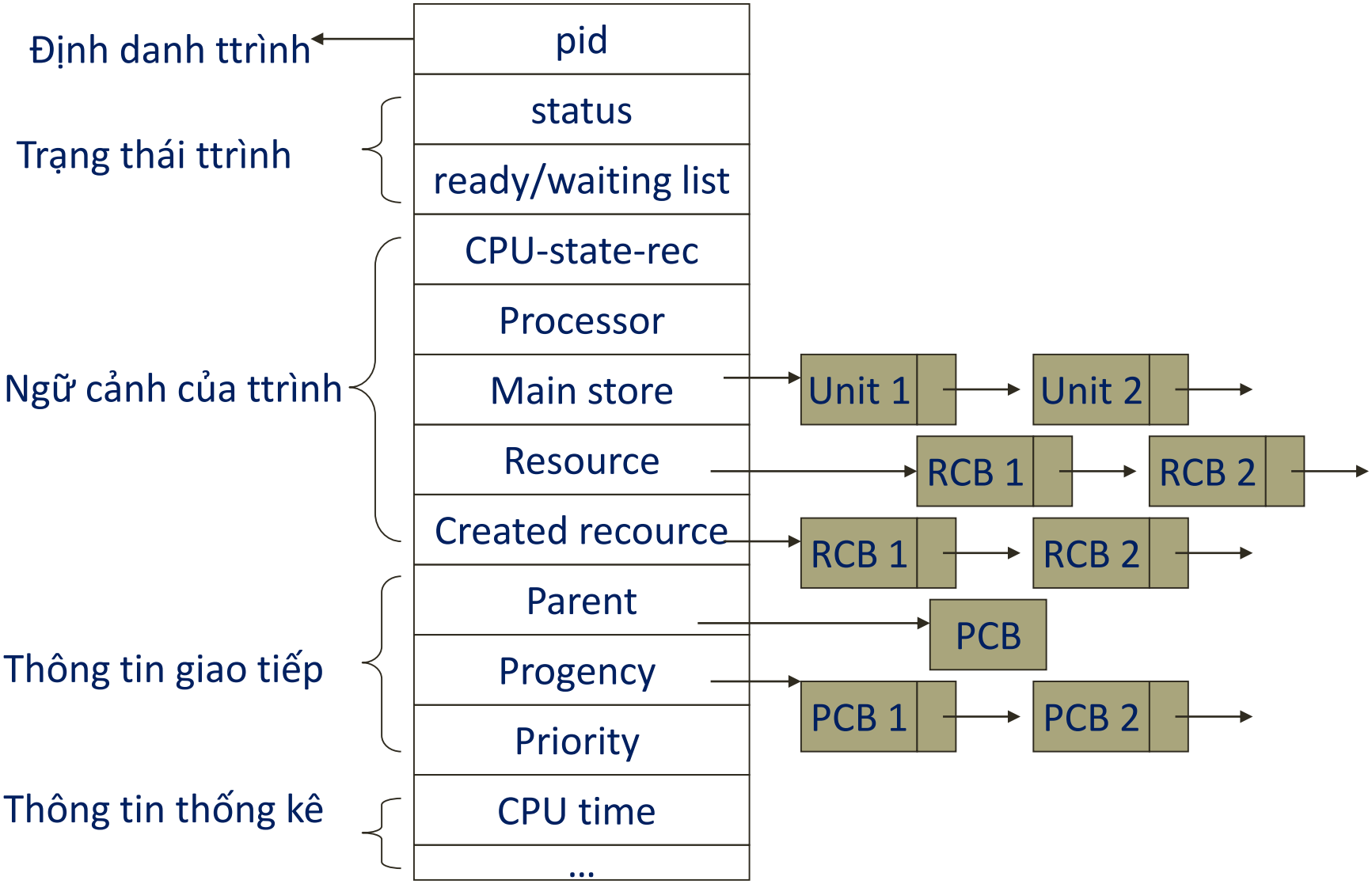
- Để quản lý các tiến trình và tài nguyên hệ thống, HĐH phải có các thông tin về trạng thái hiện thời của mỗi tiến trình và tài nguyên:
  - **memory table** cho đối tượng bộ nhớ (Không gian bộ nhớ chính dành cho tiến trình; Không gian bộ nhớ phụ dành cho tiến trình; Các thuộc tính bảo vệ bộ nhớ chính và bộ nhớ ảo…)
  - **I/O table** cho đối tượng thiết bị vào/ra;
  - **file table** cho đối tượng tập tin;
  - **process table** cho đối tượng tiến trình.

2.3.1. Khối điều khiển tiến trình

- **Khối điều khiển tiến trình (process control block -PCB)** có nhiệm vu quản lý mọi hoạt động của tiến trình.

- **Cấu trúc dữ liệu PCB:**
  - Định danh tiến trình;
  - Trạng thái của tiến trình;
  - Ngữ cảnh của tiến trình (PC - Program Counter value, trạng thái CPU, các thanh ghi, bộ xử lý, bộ nhớ, tài nguyên…);
  - Thông tin giao tiếp (tiến trình cha, tiến trình con, độ ưu tiên,…);
  - Thông tin thống kê (thời gian sử dụng, thời gian chờ…)

2.3.1. Khối điều khiển tiến trình



2.3.2. Các thao tác điều khiển tiến trình

a. Khởi tạo tiến trình

- HĐH gán PID và đưa vào ds quản lý của hệ thống;
- Cấp phát không gian bộ nhớ;
- Khởi tạo các thông tin cần thiết cho khối điều khiển tiến trình: Các PID của tiến trình cha (nếu có), thông tin trạng thái, độ ưu tiên, ngữ cảnh của processor;
- Cung cấp đầy đủ các tài nguyên (trừ processor);
- Đưa tiến trình vào danh sách tiến trình nào đó: ready list, suspend list, waiting list…

2.3.2. Các thao tác điều khiển tiến trình

b. Kết thúc tiến trình:

- Thu hồi tài nguyên đã cấp phát cho tiến trình;
- Loại bỏ tiến trình ra khỏi danh sách quản lý của hệ thống;
- Hủy bỏ khối điều khiển tiến trình.

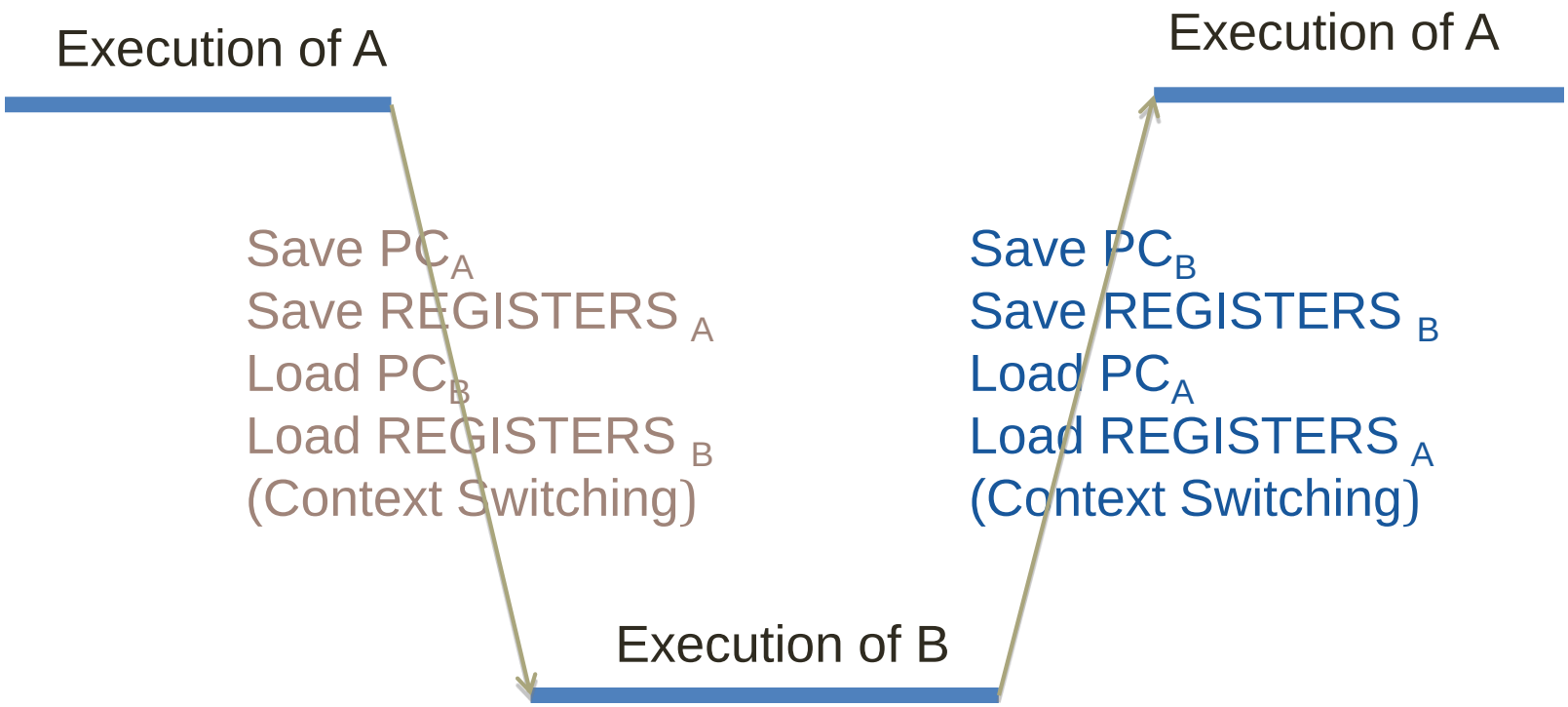
2.3.2. Các thao tác điều khiển tiến trình

c. Thay đổi trạng thái của tiến trình:

- Lưu ngữ cảnh của processor;
- Cập nhật PCB (process control block) của tiến trình sao cho phù hợp với trạng thái của nó;
- Di chuyển PCB của tiến trình đến 1 hàng đợi thích hợp
- Chọn tiến trình khác để cho phép nó thực hiện;
- Cập nhật PCB của tiến trình vừa thực hiện;
- Cập nhật thông tin liên quan đến quản lý bộ nhớ;
- Khôi phục lại ngữ cảnh của processor.

2.3.2. Các thao tác điều khiển tiến trình

- **c. Thay đổi trạng thái của tiến trình:**
  - Program Counter (PC) value,



2.4. Tài nguyên găng

2.4.1. Tài nguyên găng (Critical Resource)

2.4.2. Đoạn găng (Critical Section)

2.4.3. Tắc nghẽn (Deadlock)

2.4.4. Đói tài nguyên (Stravation)

2.4.1. Tài nguyên găng

- **Khái niệm tài nguyên găng:** Những tài nguyên được HĐH chia sẻ cho nhiều tiến trình hoạt động đồng thời dùng chung mà có nguy cơ tranh chấp giữa các tiến trình này khi sử dụng chúng.
- Tài nguyên găng có thể là tài nguyên phần cứng hoặc phần mềm, có thể là tài nguyên phân chia được hoặc không phân chia được.
- Để các tiến trình hoạt động đồng thời mà không xung đột với nhau, HĐH phải tổ chức cho các tiến trình này được độc quyền truy xuất/ sử dụng trên các tài nguyên dùng chung.

2.4.1. Tài nguyên găng

- **Ví dụ:** Giả sử có 2 tiến trình sau hoạt động song song.

```
P1:   Begin                    P2:   Begin
        L1 :=  Count;                  L2 :=  Count;
        L1 := L1 + 1;                  L2 := L2 + 1;
        Count := L1;                   Count := L2;
      End;                           End;
```

- Chương trình có thể thực hiện như sau:
  - P1 ghi nội dung biến toàn cục Count vào biến cục bộ L1
  - P2 ghi nội dung biến toàn cục Count vào biến cục bộ L2
  - Tiến trình P1 thực hiện L1:= L1 + 1 và Count := L1
  - Tiến trình P2 thực hiện L2:= L2 + 1 và Count := L2

2.4.1. Tài nguyên găng

- **VD:** Giả sử có 2 tiến trình cùng thực hiện đoạn mã sau.

```
if  (Tài khoản - Tiền rút >= 0)                {kiểm tra tài khoản}
        Tài khoản := Tài khoản - Tiền rút      {thực hiện rút tiền}
else
        Thông báo lỗi                          {không thể rút tiền}
endif;
```

- **Vấn đề?**
- **Nguyên nhân:** hai thao tác kiểm tra tài khoản và thực hiện rút tiền của các tiến trình này có thể bị tách rời nhau do song song hóa.
- **Giải pháp?**

2.4.2. Đoạn găng (Critical Section)

- **Critical Section:** Các đoạn code trong các chương trình dùng để truy cập đến tài nguyên găng được gọi là đoạn găng.
- HĐH có cơ chế **điều độ tiến trình qua đoạn găng**:
  - Để hạn chế lỗi có thể xảy ra do sử dụng tài nguyên găng, tại 1 thời điểm HĐH chỉ cho phép 1 tiến trình nằm trong đoạn găng;
  - Nếu có nhiều tiến trình cùng muốn vào (thực hiện) đoạn găng thì chỉ có một tiến trình được vào, các tiến trình khác phải chờ, một tiến trình khi ra khỏi (kết thúc) đoạn găng phải báo cho hệ điều hành và/hoặc các tiến trình khác biết để các tiến trình này vào đoạn găng, vv.

2.4.2. Đoạn găng (Critical Section)

- Yêu cầu của công tác điều độ tiến trình qua đoạn găng:
  - Tại 1 thời điểm chỉ cho phép 1 tiến trình nằm trong đoạn găng;
  - Nếu có nhiều tiến trình đồng thời cùng xin được vào đoạn găng thì chỉ có một tiến trình được phép;
  - Tiến trình chờ ngoài đoạn găng không được ngăn cản các tiến trình khác vào đoạn găng;
  - Không có tiến trình nào phải chờ lâu để được vào đoạn găng;
  - Đánh thức các tiến trình trong hàng đợi để tạo điều kiện cho nó vào đoạn găng khi tài nguyên găng được giải phóng.

2.4.3. Tắc nghẽn (Deadlock)

- **Ví dụ:** Giả như có hai tiến trình P1 và P2, và hai tài nguyên găng R1 và R2, mỗi tiến trình đều cần truy xuất đến để mã thực hiện một hàm của nó. Xét trường hợp sau:
- R1 đang được giao cho P2, R2 được giao cho P1. Mỗi tiến trình đều chờ đợi được sử dụng tài nguyên thứ hai;

2.4.3. Tắc nghẽn (Deadlock)

- **Khái niệm:** Sự xung đột về tài nguyên của các tiến trình hoạt động đồng thời trong hệ thống
- **Điều kiện hình thành tắc nghẽn:**
  - Sử dụng tài nguyên không thể chia sẻ;
  - Chiếm giữ và yêu cầu tài nguyên;
  - Không thu hồi tài nguyên từ tiến trình đang chiếm giữ chúng;
  - Đợi vòng tròn;

2.4.4. Đói tài nguyên (Stravation)

- **Ví dụ:** Giả sử rằng có 3 tiến trình P1, P2, P3 đều cần truy xuất định kỳ đến tài nguyên R. Xét trường hợp sau:
  - P1 đang sở hữu tài nguyên. P2, P3 đang chờ;
  - Khi P1 thoát khỏi đoạn găng, cả P2 lẫn P3 đều có thể được chấp nhận truy xuất đến R;
  - Giả sử rằng P3 được truy xuất R, sau đó trước khi P3 kết thúc đoạn găng, P1 lại một lần nữa cần truy xuất;
  - nếu như P1, P3 thay nhau nhận được quyền truy xuất thì P2 hầu như không thể truy cập đến tài nguyên, cho dù không có sự tắc nghẽn nào xảy ra.

Chương 2. Tiến trình và luồng

2.1. Tổng quan về tiến trình và luồng

2.2. Các trạng thái của tiến trình

2.3. Điều khiển tiến trình

2.4. Tắc nghẽn (deadlock) và tránh tắc nghẽn

2.5. Lập lịch cho CPU

2.6. Bài tập phần Quản lý tiến trình

2.5. Lập lịch cho CPU

- **Lập lịch cho CPU (Processor Scheduling):**
- Trong các hệ thống đa chương, tiến trình được thực thi theo cơ chế song song giả lập (mỗi chương trình như có 1 CPU riêng). Trên thực tế đó là sự chuyển đổi quyền chiếm dụng CPU liên tục giữa các tiến trình.
- Một vài khái niệm:
  - Thực thi (*execution*) tiến trình: CPU thực hiện các chỉ thị lệnh của tiến trình, thay đổi các thông số thanh ghi… Ta gọi quá trình này là *"***processor (CPU) burst***"*
  - Nhập xuất (*I/O work*): CPU thực hiện các thao tác I/O, quá trình này gọi là *"***I/O burst***"*.
  - Tương ứng, có thể phân loại các chương trình thành **Processor-bound programs, I/O-bound programs**

2.5. Lập lịch cho CPU

- **Mục tiêu điều phối bằng lập lịch:**
  - Sự công bằng;
  - Tính hiệu quả;
  - Thời gian đáp ứng hợp lý;
  - Cực tiểu hóa thời gian lưu lại trong hệ thống
  - Thông lượng tối đa (cực đại hóa số công việc được xử lý trong 1 khoảng thời gian)

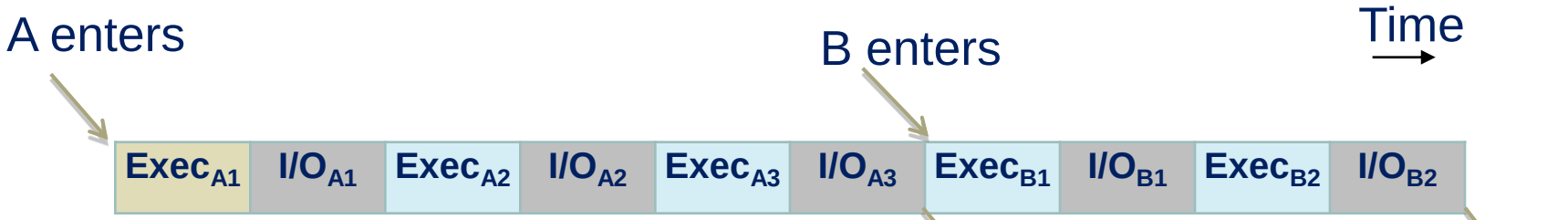
2.5. Lập lịch cho CPU

- **Cơ chế điều phối:**
  - **Độc quyền (nonpreemtive):** Tiến trình toàn quyền sử dụng processor cho đến khi kết thúc hoặc tự động trả lại CPU
    - Quyết định điều phối khi tiến trình chuyển từ Running sang Waiting (blocked) hoặc kết thúc.
  - **Không độc quyền (preemtive):** Tiến trình đang xử lý thì bị thu hồi processor để cấp cho tiến trình khác.
    - Quyết định điều phối khi tiến trình chuyển từ Running sang Waiting (blocked) hoặc ready hoặc kết thúc hoặc từ Waiting sang ready

2.5. Lập lịch cho CPU

- Giả sử ta có 2 tiến trình A bà B, mỗi tiến trình mất 1 giây để thực thi CPU và 1 giây để nhập xuất dữ liệu. A thực thi 3 lần, B 2 lần:

- Nếu các tiến trình thực hiện một các tuần tự, sơ đồ thực hiện có thể như sau:

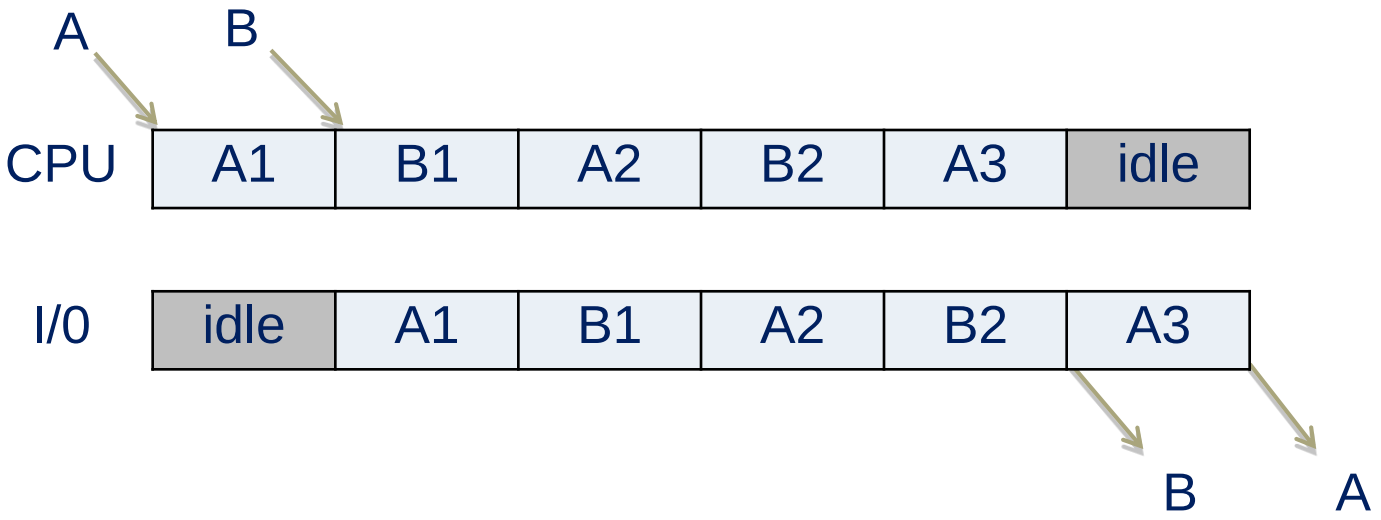


$$Utilization = \frac{5}{10} \times 100 = 50\%$$

2.5. Lập lịch cho CPU

- Đa chương:

$$Utilization = \frac{5}{6} \times 100 = 83\%$$



- Các tiến trình yêu cầu thực thi có thể nhiều hơn so với khả năng thực thi cũng như lưu trữ trong RAM, vì vậy một phần có thể nằm ở trạng thái **suspend.**
- **Lập lịch dài hạn - The long-term scheduler** lựa chọn tiến trình (từ **suspend**) đưa vào bộ nhớ (sang trạng thái **ready** hoặc **blocked**).

2.5. Lập lịch cho CPU

- **Lập lịch ngắn hạn - The short-term scheduler** lựa chọn tiến trình từ trạng thái ready sang running.
  - Lập lịch ngắn hạn thực hiện thường xuyên (khoảng 10 milliseconds). Vì vậy có khả năng nhanh chóng điều phối để đạt hiệu quả sử dụng CPU.
  - Lập lịch dài hạn ít thực hiện hơn, thông thường được gọi đến khi có 1 tiến trình kết thúc công việc.
  - Hệ điều hành cung cấp 1 Chương trình lập lịch (chạy ở chế độ ưu tiên). Chương trình chiếm dụng khoảng 5 ->10% thời gian xử lý của CPU.
  - Hầu hết các hệ chia sẻ thời gian không đòi hỏi lập lịch dài hạn.
- **Lập lịch trung hạn - Medium-term scheduler** Trong một số trường hợp hệ điều hành điều chuyển một số tiến trình từ các trạng thái như running sang suspend, sử dụng cơ chế swapping.

2.5. Lập lịch cho CPU

- **Một vài tiêu chuẩn đánh giá hiệu suất (Performance Criteria) của điều phối bằng lập lịch:**
- **Hiệu suất CPU - Processor Utilization:** tần suất sử dụng CPU với thời gian thực tế:

  Processor Utilization = (Processor busy time) / (Processor busy time + Processor idle time)

- **Thông lượng - Throughput:** Khối lượng công việc hoàn thành trên 1 đơn vị thời gian.

  Throughput = (Number of processes completed) / (Time Unit)

- **Thời gian quay vòng - Turnaround Time (tat):** Thời gian chờ đợi trong trạng thái ready, thời gian thực thi và nhập xuất dữ liệu

  tat = t(tiến trình hoàn tất) – t(tiến trình xuất hiện)

2.5. Lập lịch cho CPU

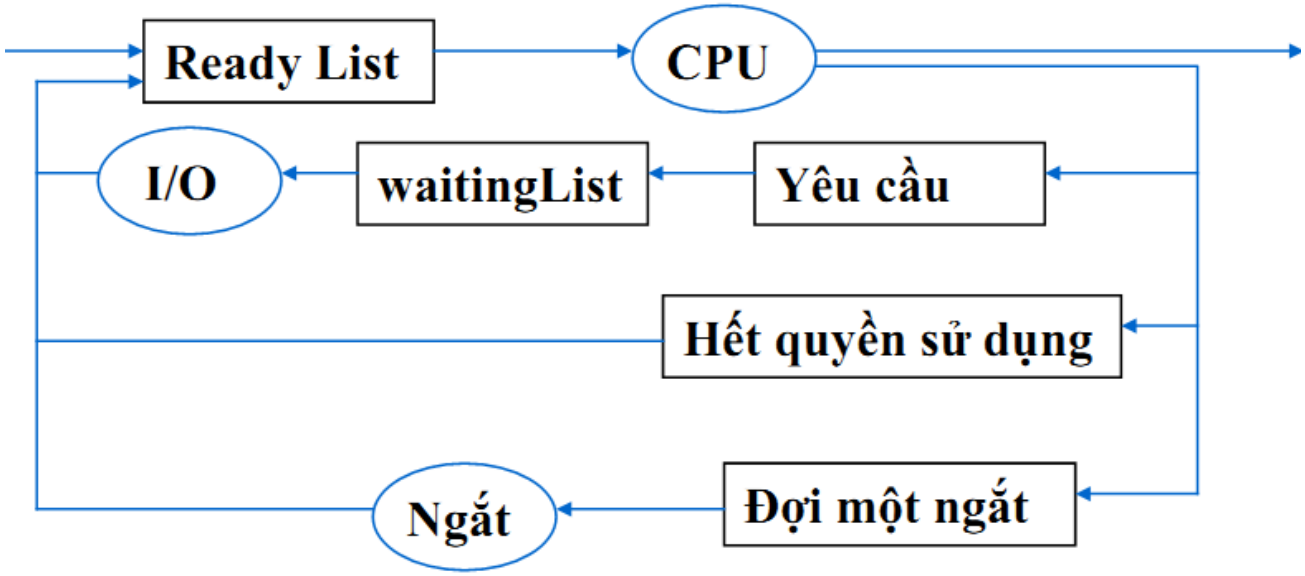
- **Waiting Time (wt):** Lập lịch cho tiến trình chỉ xét đến thời gian đợi trong hàng đợi ready.
- **Response Time (rt):** thời gian chờ để bắt đầu đáp ứng 1 yêu cầu. Tiêu chí này quan trọng với các hệ thống tương tác.

$$rt = t(\text{bắt đầu thực thi}) - t(\text{đệ trình truy vấn})$$

- Lập lịch cho tiến trình đòi hỏi cực đại hóa hiệu suất CPU và thông lượng, cực tiểu hóa tat, wt, và rt. Tuy vậy tùy thuộc vào đòi hỏi của hệ thống mà có thể có cách đánh giá khác.

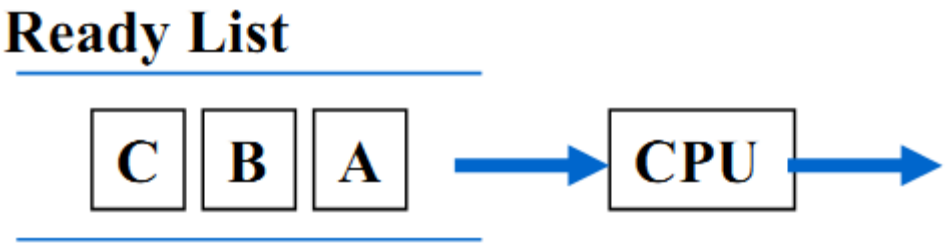
2.5. Lập lịch cho CPU

- Giả sử rằng HĐH sử dụng 2 loại danh sách để tổ chức lưu trữ các tiến trình:
  - Danh sách Ready: Chỉ tồn tại 1 danh sách này
  - Danh sách Waiting: Có thể tồn tại nhiều danh sách này
  - Hệ thống chỉ có 1 I/O server;
  - Bỏ qua thời gian chuyển ngữ cảnh



2.5.2. Chiến lược FIFO (FCFS)

- Chiến lược **FIFO (FCFS)**: Tiến trình nào được đưa vào danh sách ready trước sẽ được cấp Processor trước



Tiến trình	Thời điểm vào	t/g xử lý
P1	0	24
P2	1	3
P3	2	3

Thời điểm cấp processor		
P1	P2	P3
0	24	27

Thời gian chờ:
P1: 0; P2: 23; P3: 25

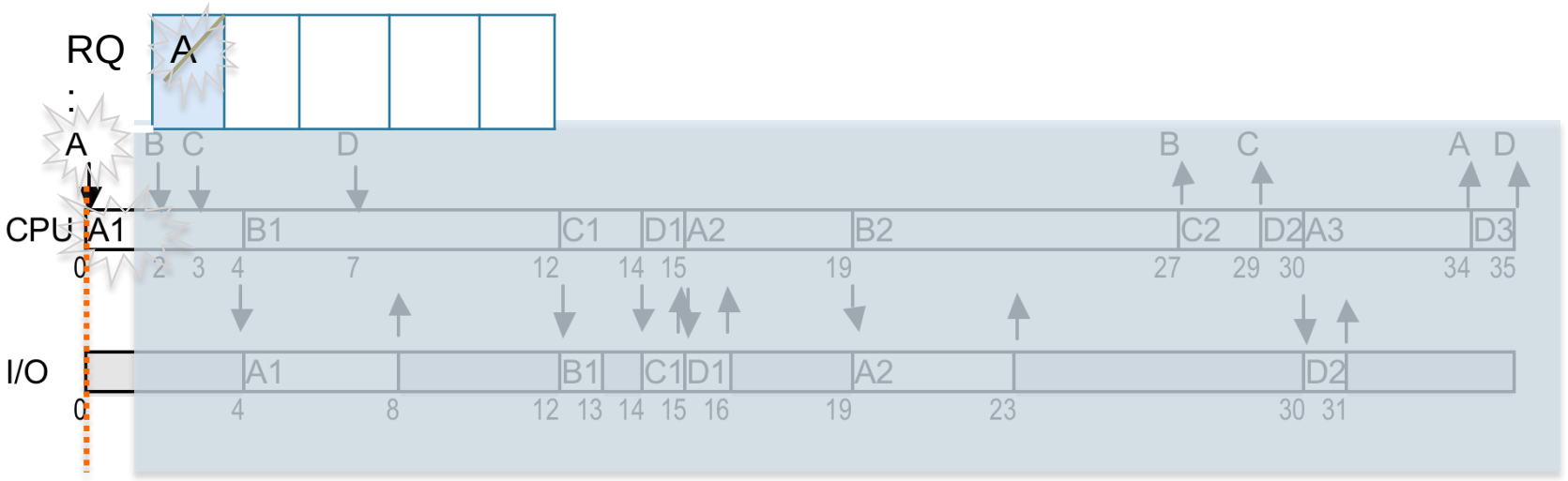
2.5.2. Chiến lược FIFO (FCFS)

- Ta xét ví dụ sau:

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

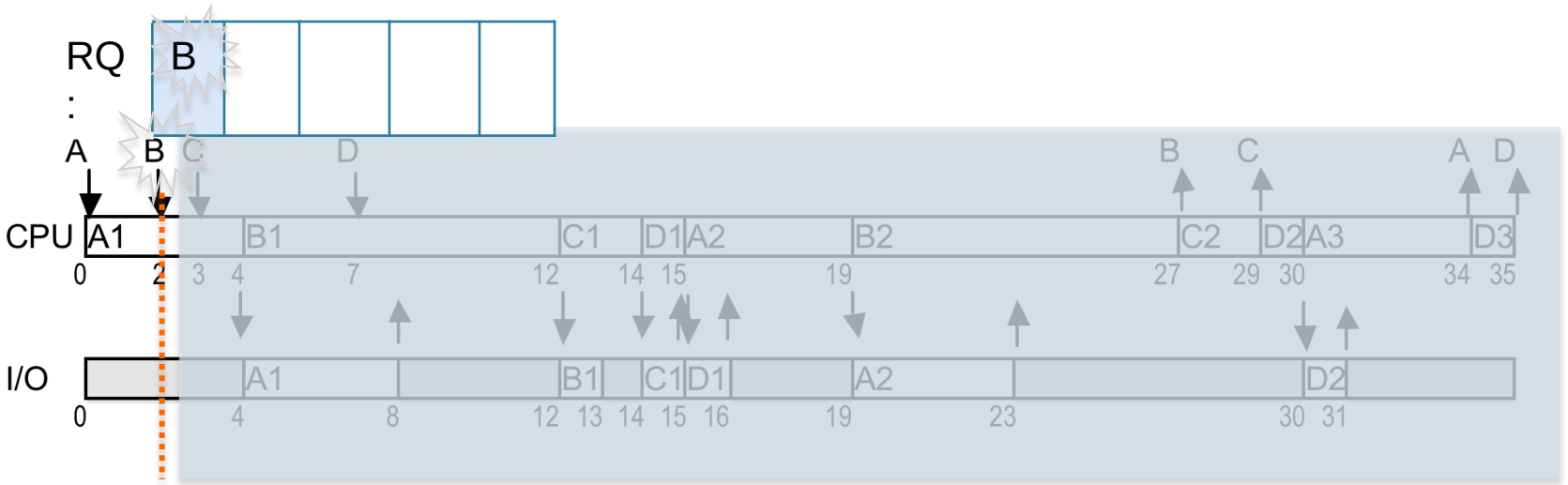
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



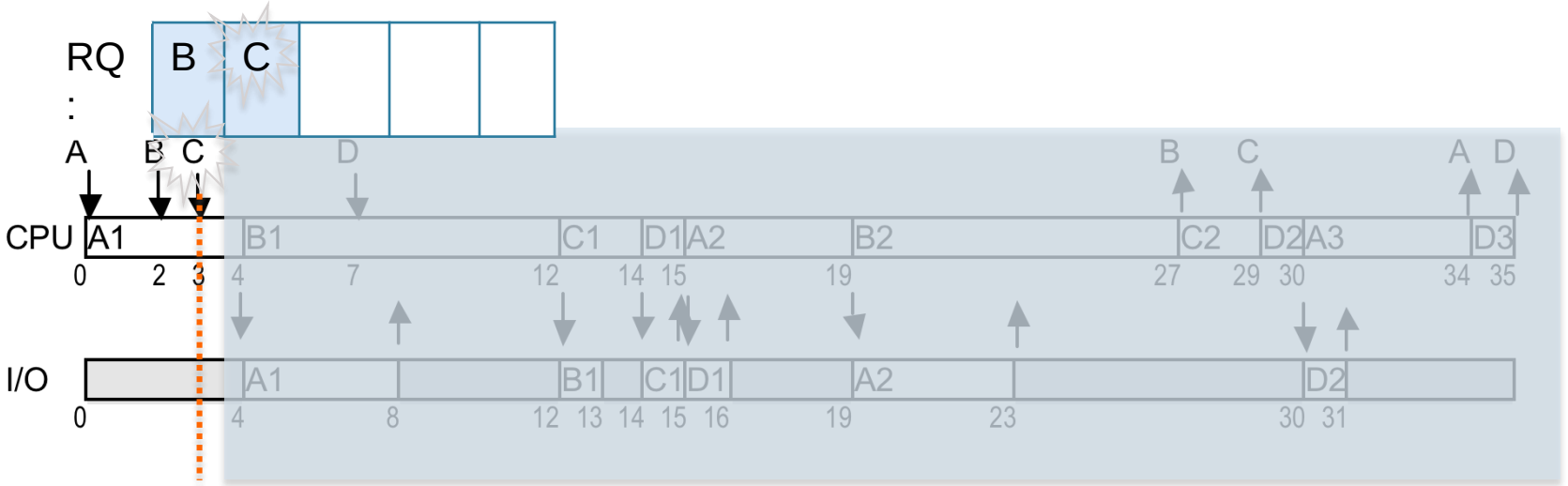
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



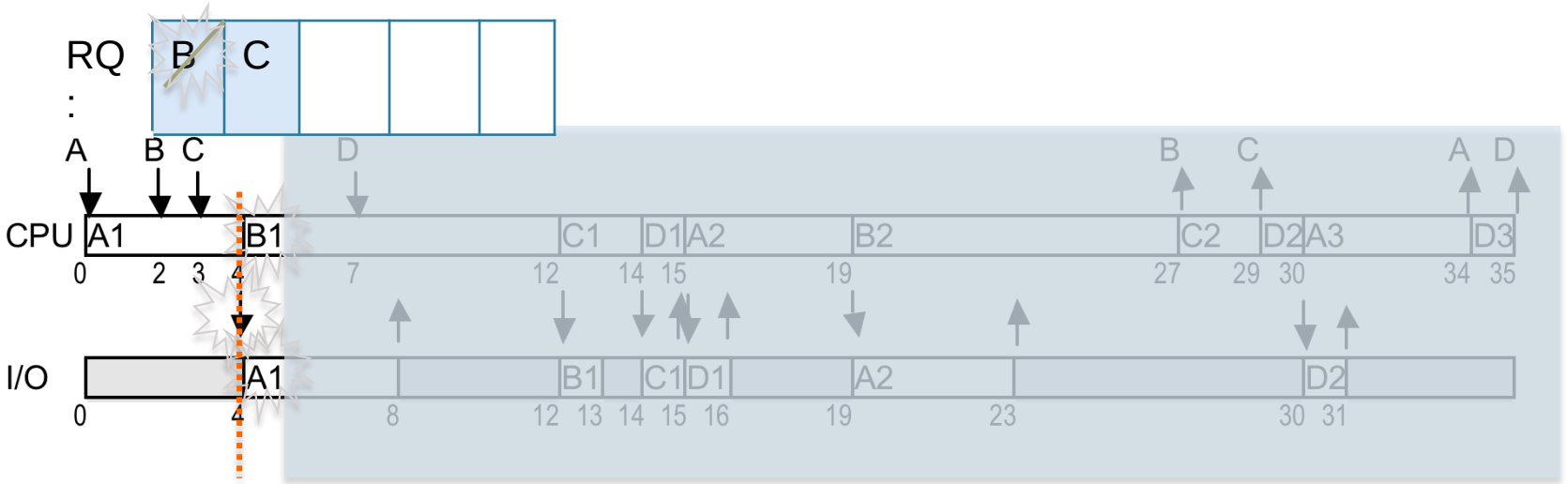
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



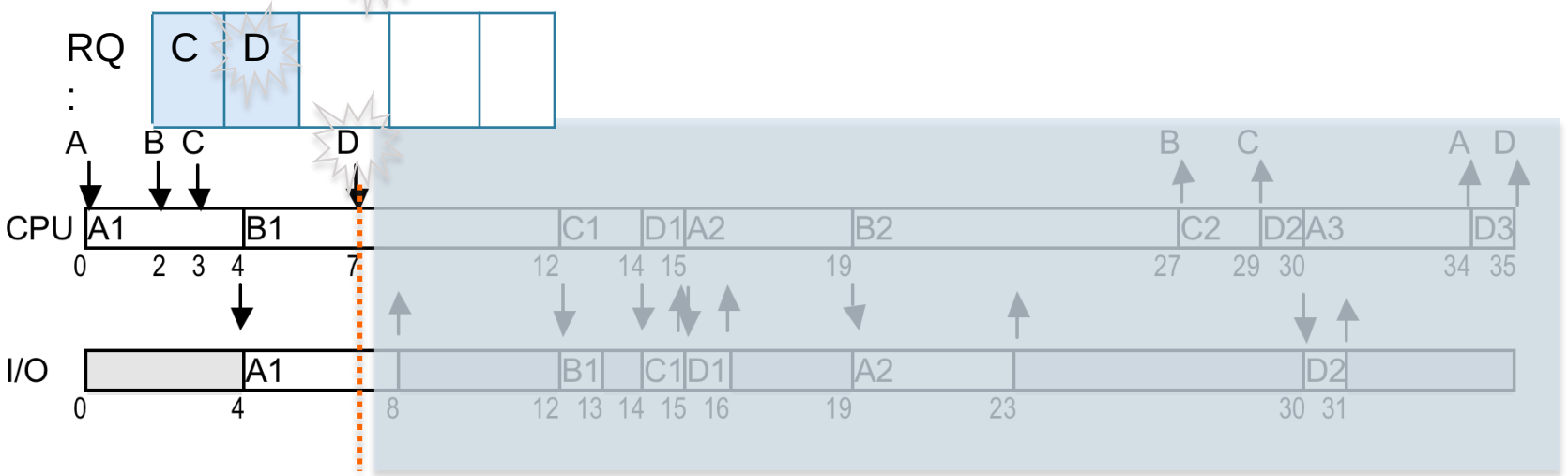
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



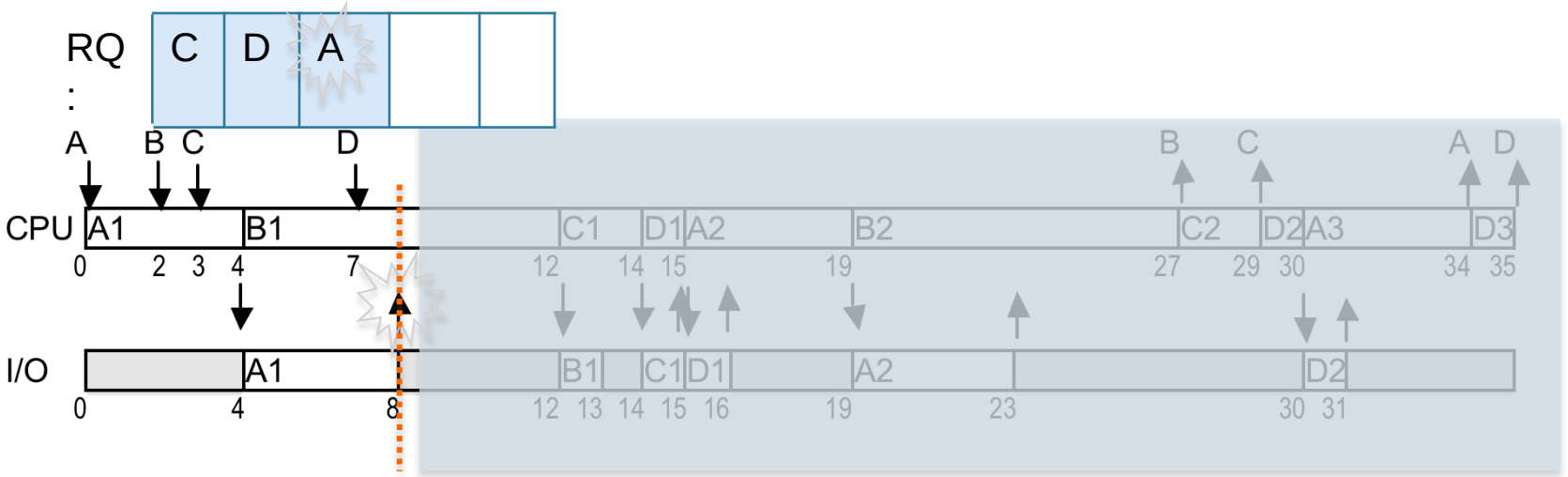
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



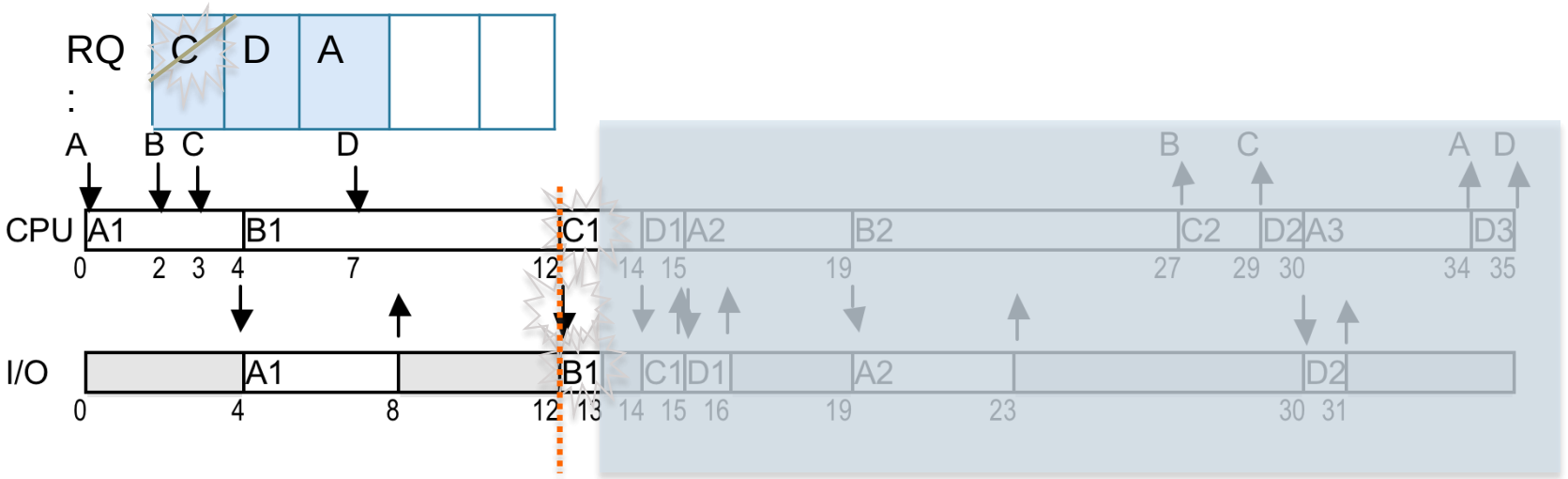
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



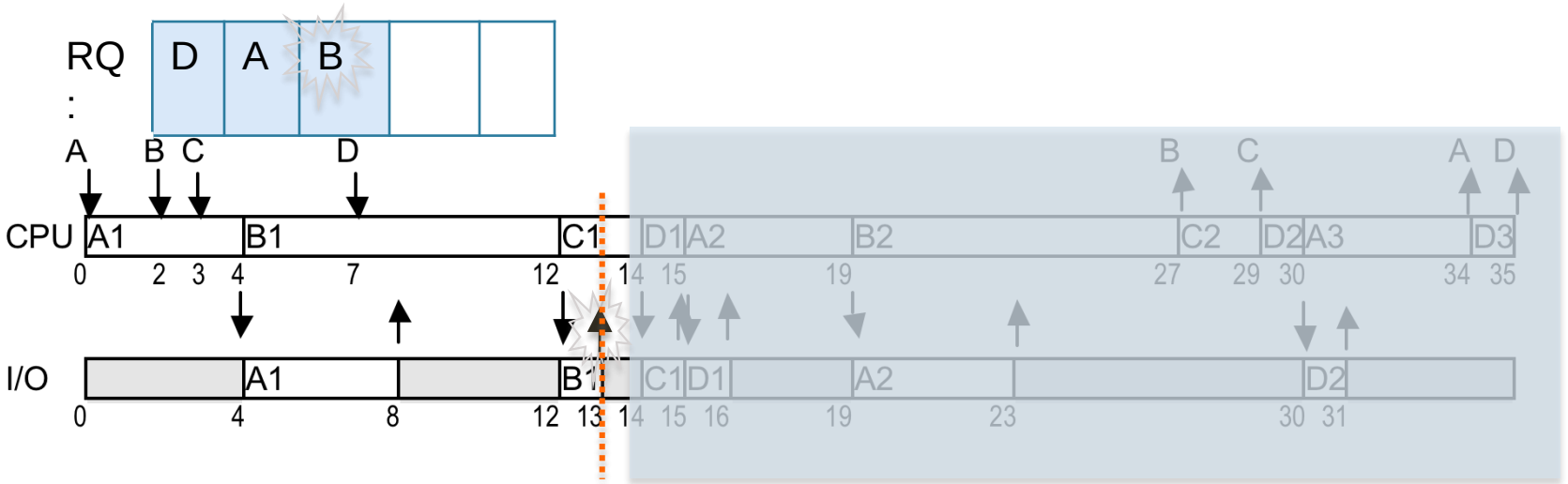
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



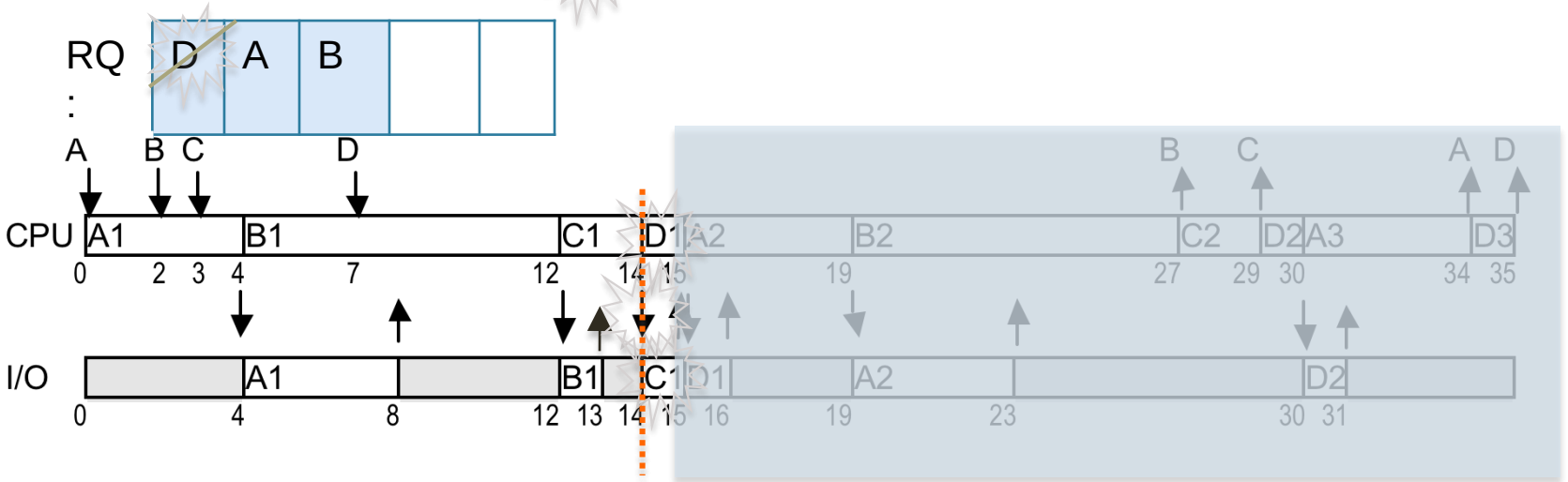
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



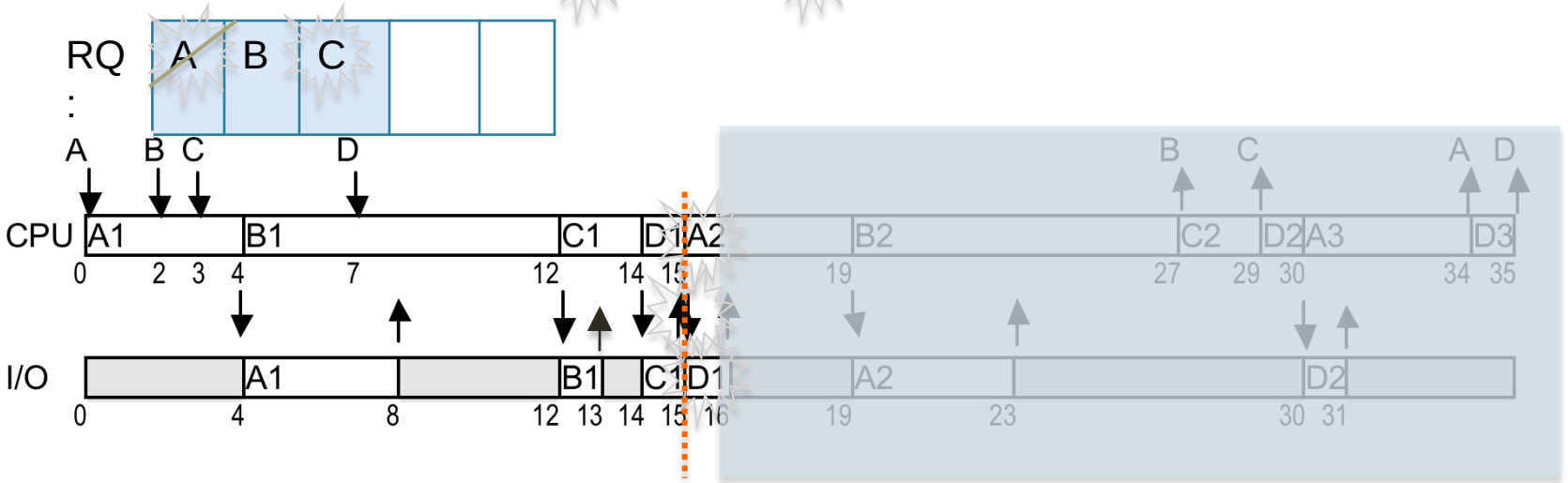
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



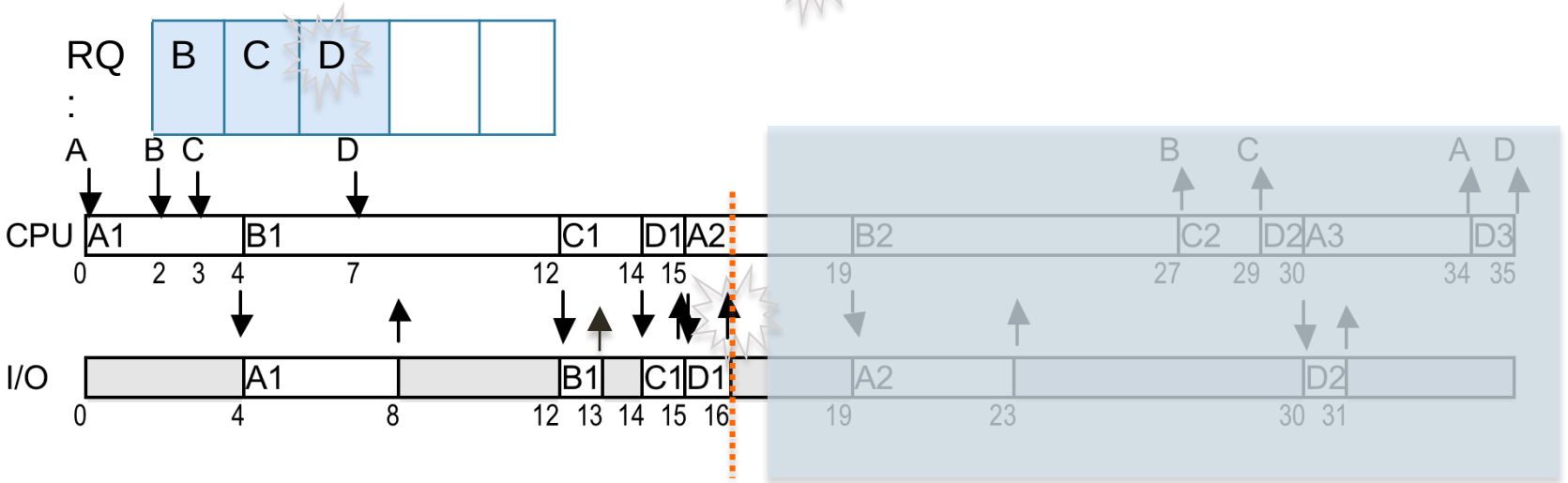
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



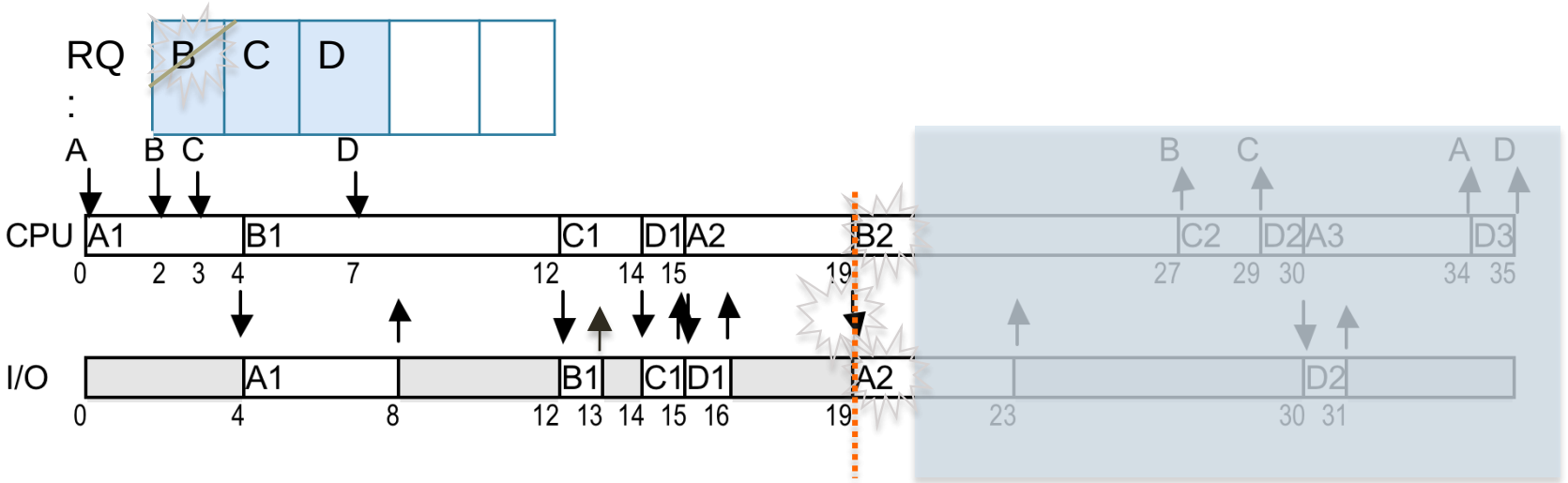
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



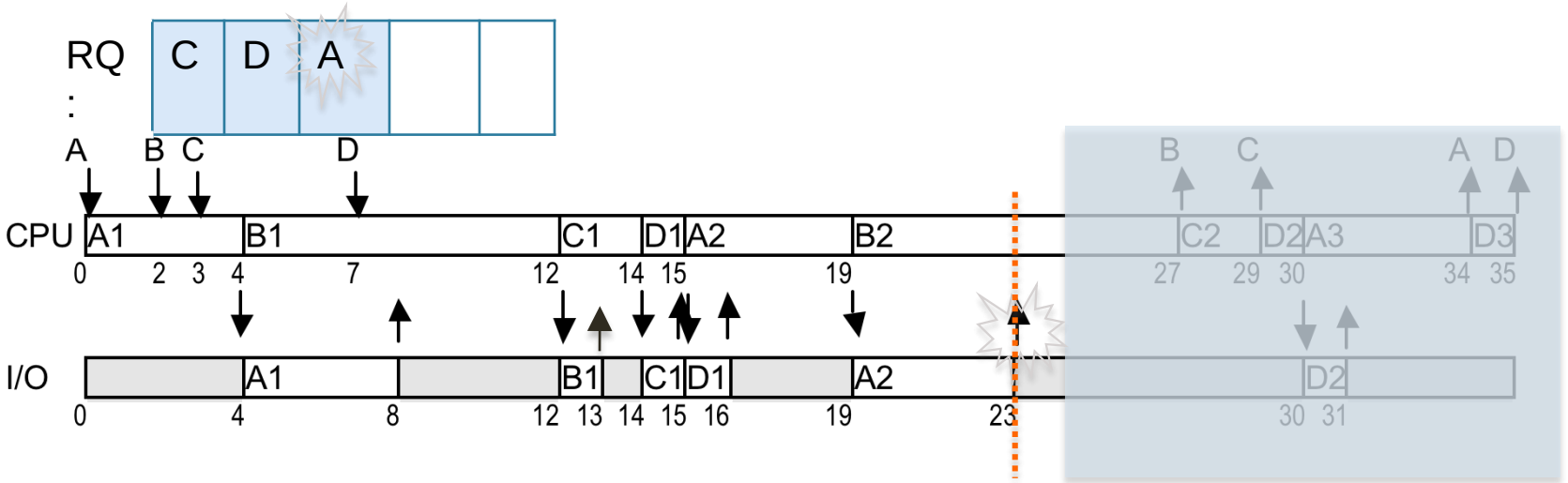
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	6	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



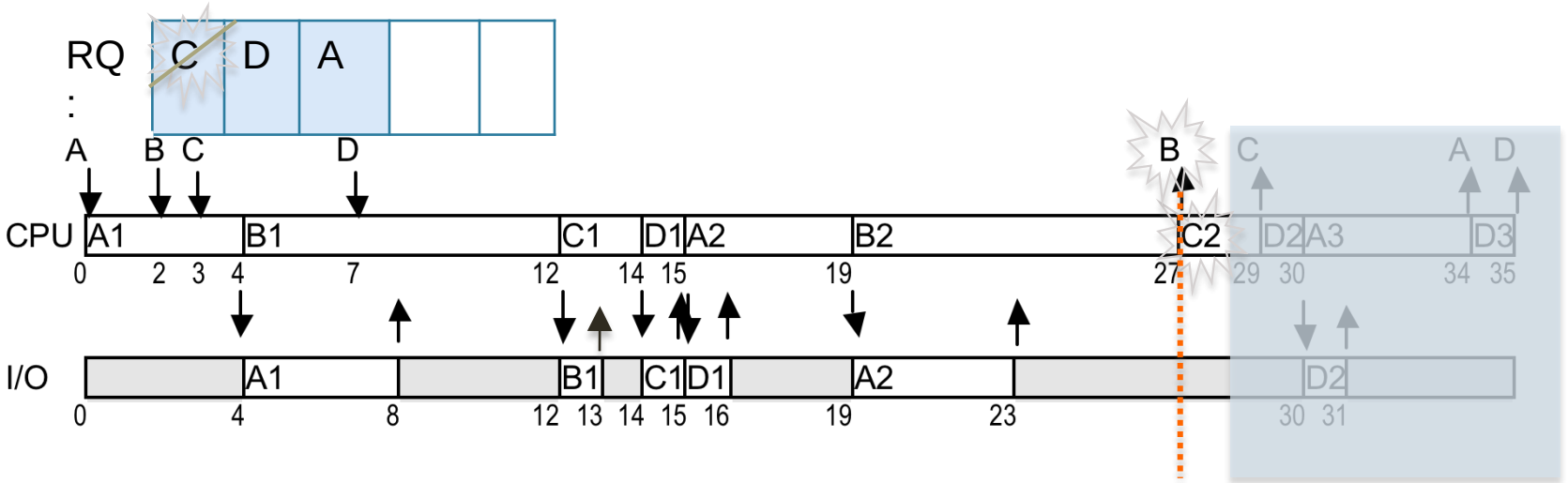
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



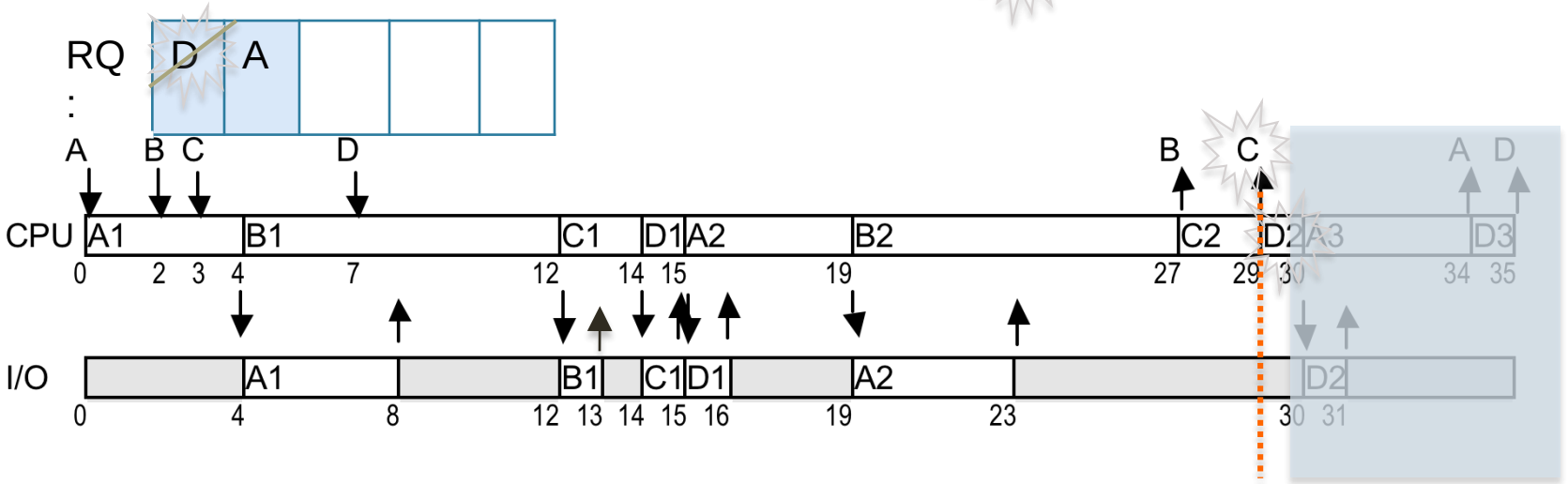
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



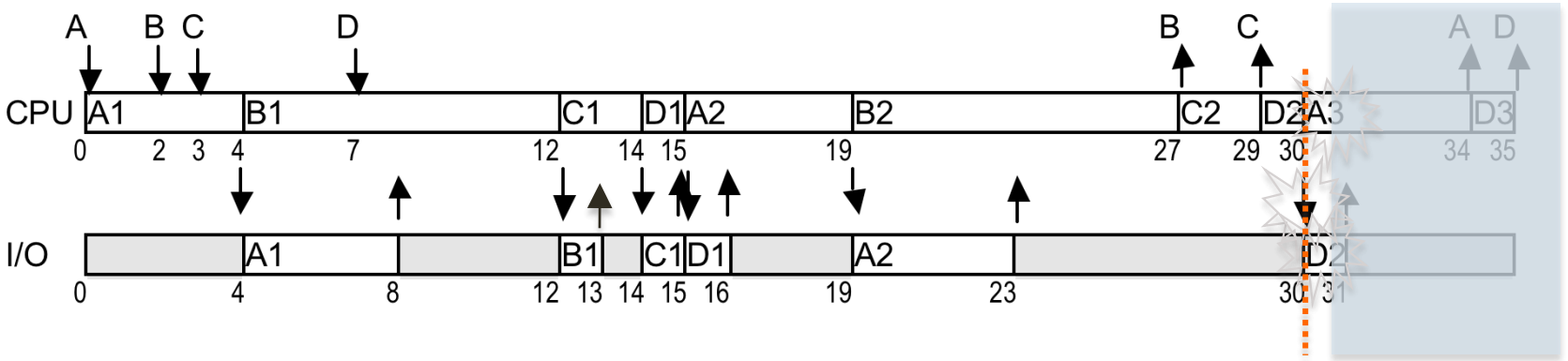
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



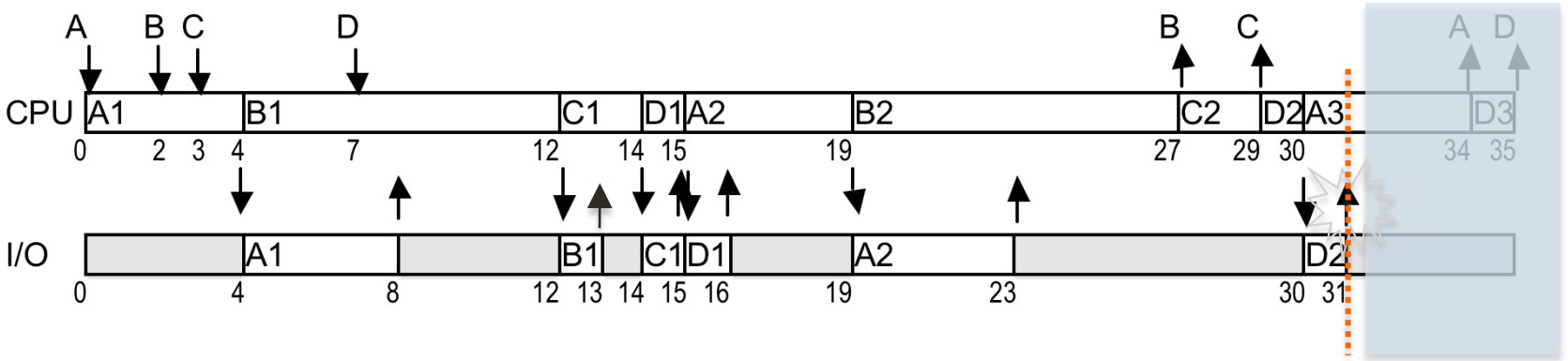
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



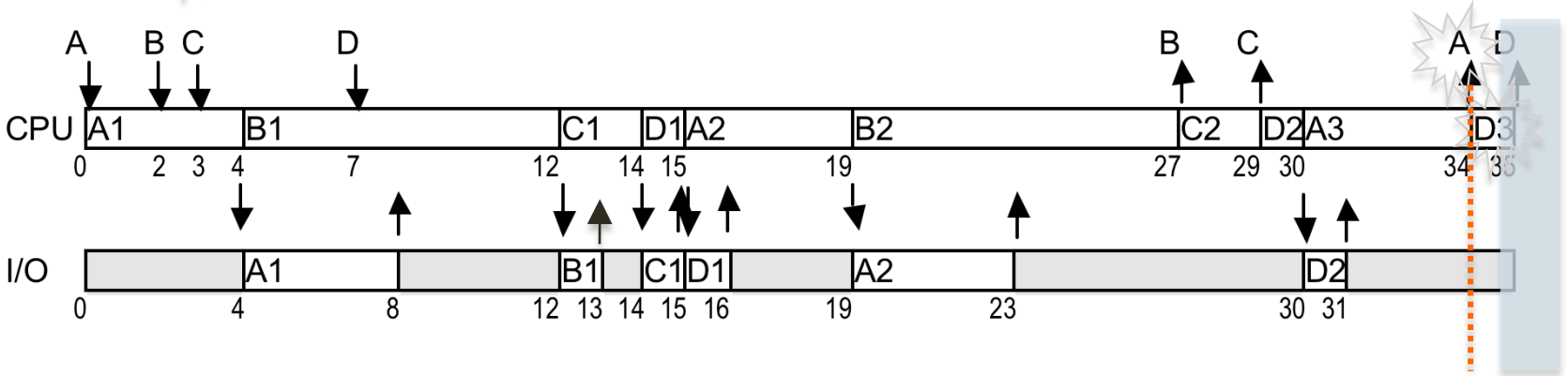
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



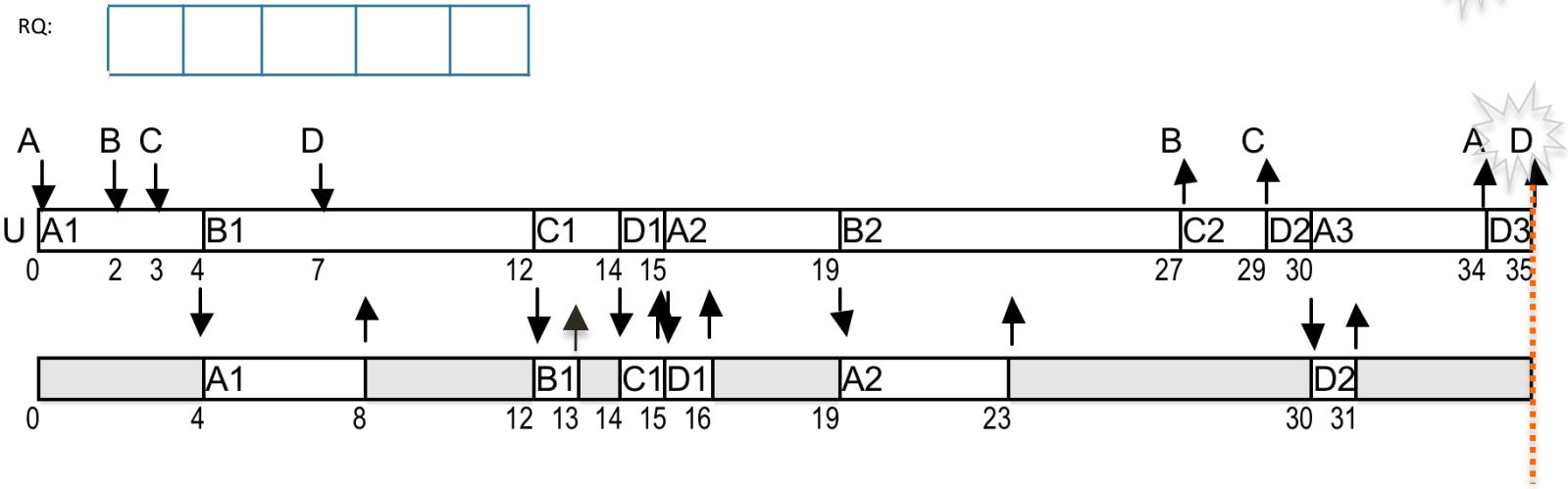
2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

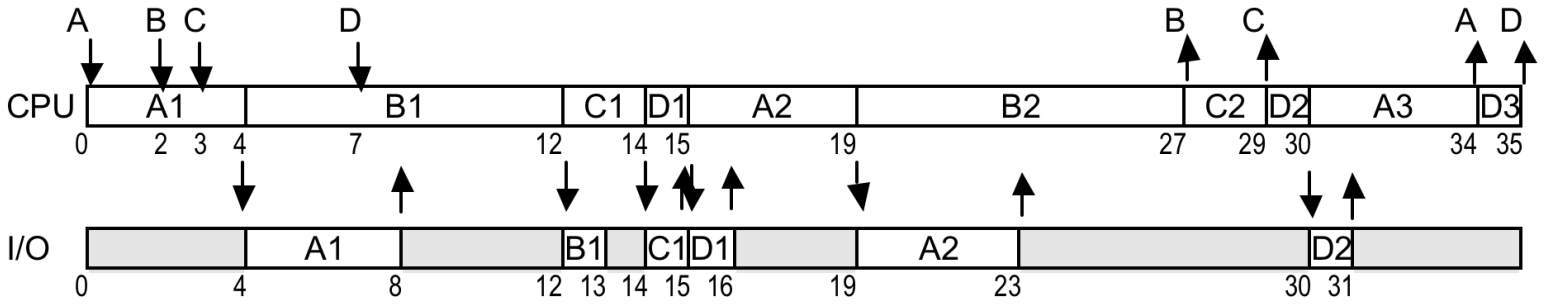


2.5.2. Chiến lược FIFO (FCFS)

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

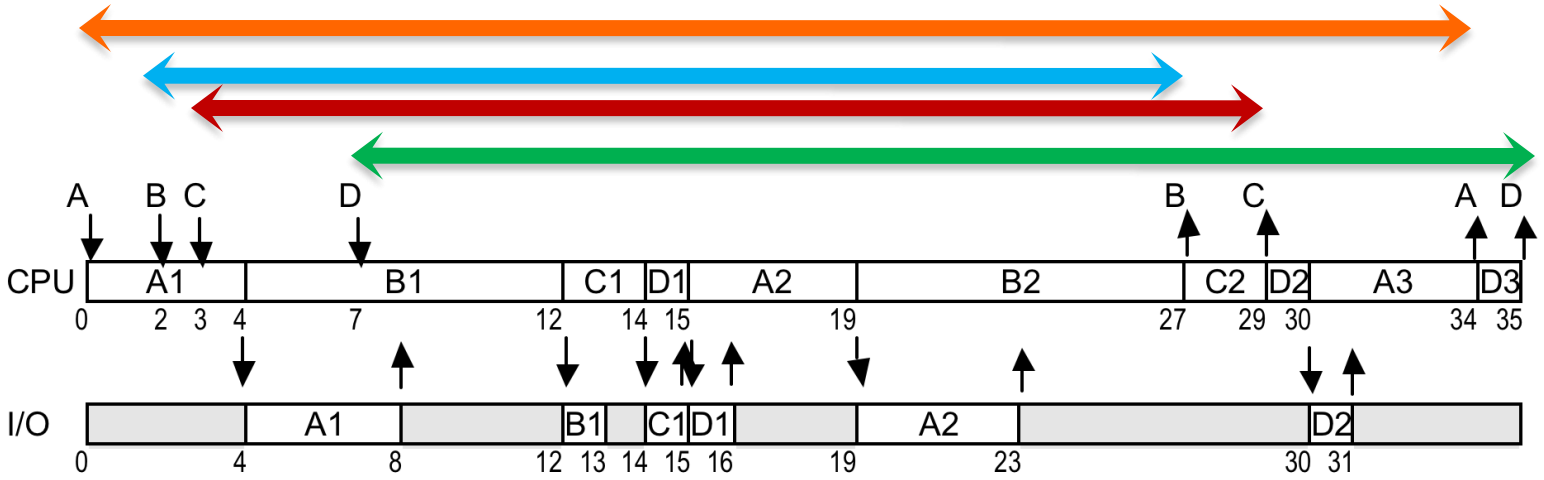


2.5.2. Chiến lược FIFO (FCFS)



- Processor utilization = (35 / 35) * 100 = 100 %
- Throughput = 4 / 35=0.11

2.5.2. Chiến lược FIFO (FCFS)



- Turn around time:

$tat_A = 34 - 0 = 34$

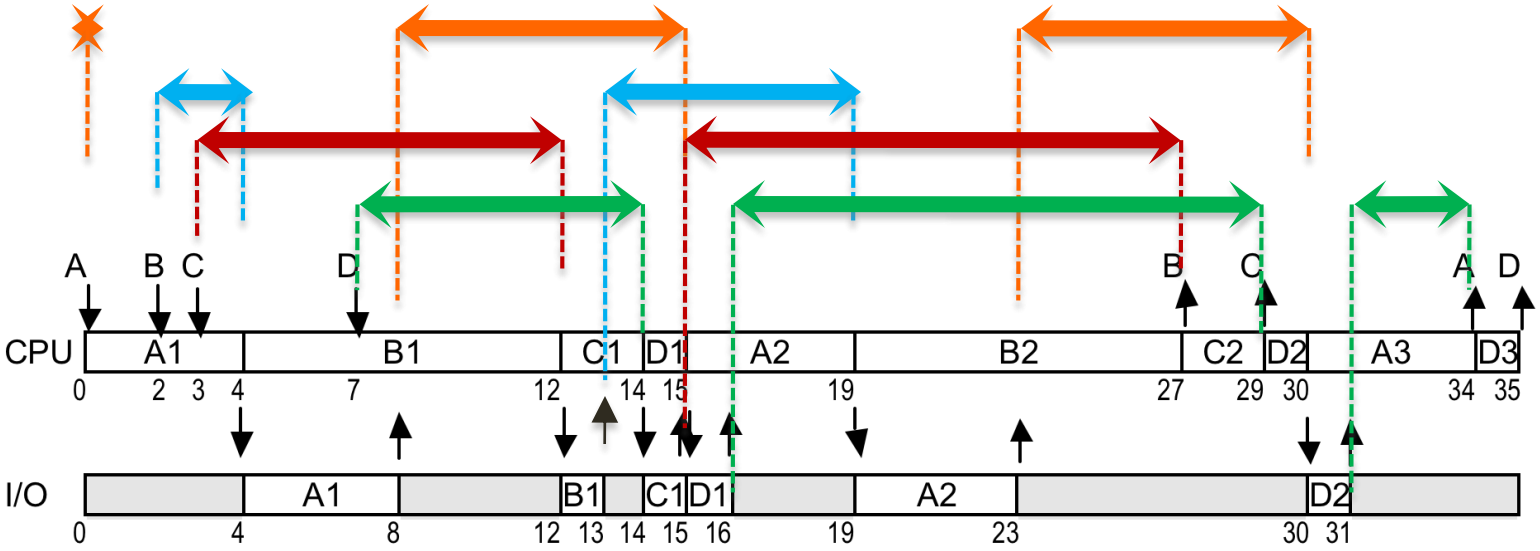
$tat_B = 27 - 2 = 25$

$tat_C = 29 - 3 = 26$

$tat_D = 35 - 7 = 28$

$tat_{AVG} = (34 + 25 + 26 + 28) / 4 = 28.25$

2.5.2. Chiến lược FIFO (FCFS)



- Waiting time:

$wt_A = (0 - 0) + (15 - 8) + (30 - 23) = 14$

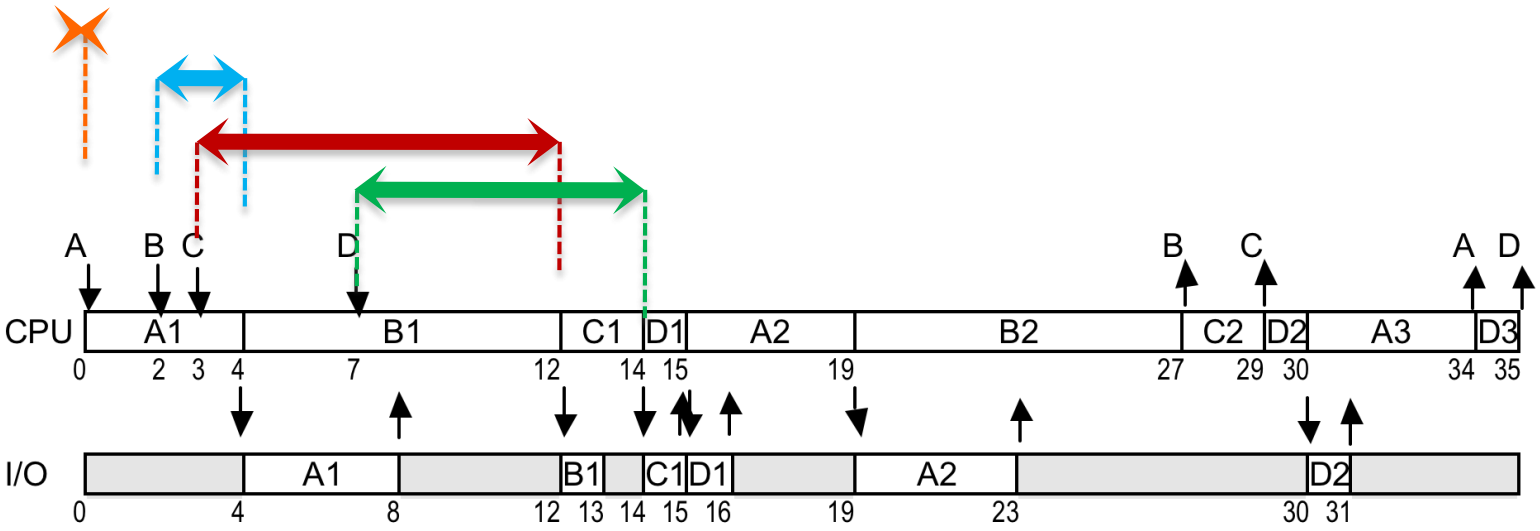
$wt_B = (4 - 2) + (19 - 13) = 8$

$wt_C = (12 - 3) + (27 - 15) = 21$

$wt_D = (14 - 7) + (29 - 16) + (34 - 31) = 23$

$wt_{AVG} = (14 + 12 + 21 + 23) / 4 = 16.5$

2.5.2. Chiến lược FIFO (FCFS)



- Response time:

$rt_A = 0 - 0 = 0$

$rt_B = 4 - 2 = 2$

$rt_C = 12 - 3 = 9$

$rt_D = 14 - 7 = 7$

$rt_{AVG} = (0 + 2 + 9 + 7) / 4 = 4.5$

2.5.3. Chiến lược theo độ ưu tiên

- **Chiến lược theo độ ưu tiên (Priority Scheduling):**
  - Mỗi tiến trình được gán cho một độ ưu tiên tương ứng, tiến trình có độ ưu tiên cao nhất sẽ được chọn để cấp phát CPU đầu tiên;
  - Độ ưu tiên của tiến trình do HĐH gán và có thể bị thay đổi;
  - Giải thuật điều phối với độ ưu tiên có thể theo nguyên tắc độc quyền hay không độc quyền;
  - Điều phối với độ ưu tiên và không độc quyền sẽ thu hồi processor từ tiến trình hiện hành để cấp cho tiến trình mới nếu độ ưu tiên của tiến trình này cao hơn
  - Điều phối với độ ưu tiên và độc quyền sẽ chỉ chèn tiến trình mới vào danh sách sẵn sàng tại vị trí phù hợp;

2.5.3. Chiến lược theo độ ưu tiên

- **Nhược điểm:** Tiến trình có độ ưu tiên thấp dễ rơi vào trạng thái chờ vô hạn => Cần giảm độ ưu tiên của tiến trình sau mỗi lần được cấp processor

Tiến trình	Độ ưu tiên	t/g xử lý
P1	3	24
P2	2	3
P3	1	3

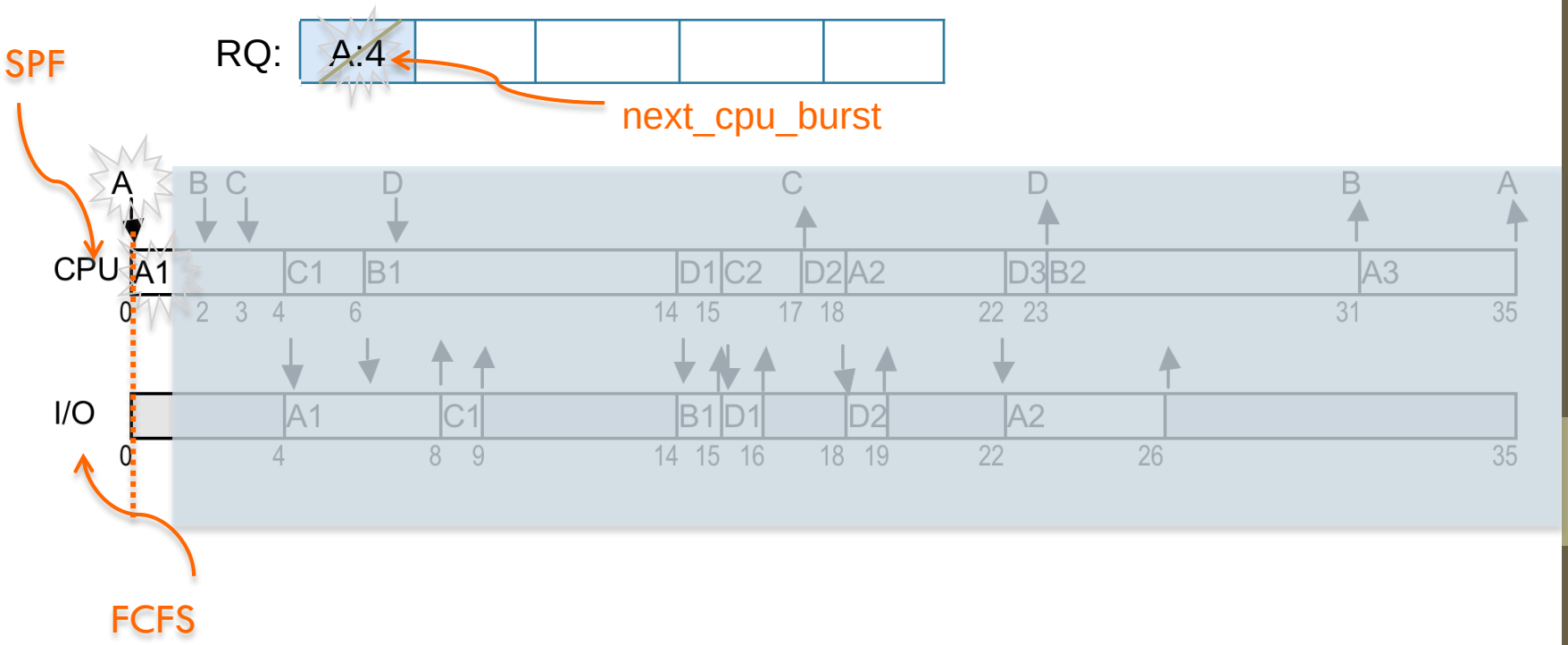
Thời điểm cấp processor		
P1	P2	P3
0	24	27

2.5.4. Chiến lược công việc ngắn nhất

- **Chiến lược công việc ngắn nhất (shortest job first – SJF hoặc shortest process first - SPF):**
  - Đây là một trường hợp đặc biệt của giải thuật điều phối với độ ưu tiên
  - Độ ưu tiên p được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý t mà tiến trình yêu cầu : p = 1/t
  - CPU được sẽ được cấp phát cho tiến trình yêu cầu ít thời gian nhất để kết thúc tiến trình
  - Giải thuật này cũng có thể độc quyền hoặc không độc quyền

2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

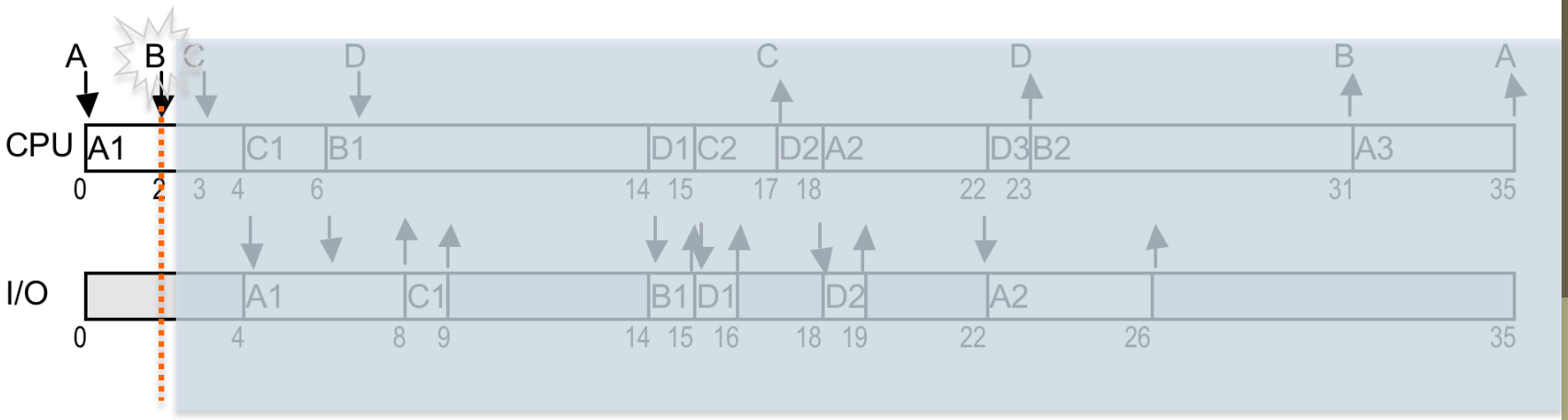


2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

RQ: B:8

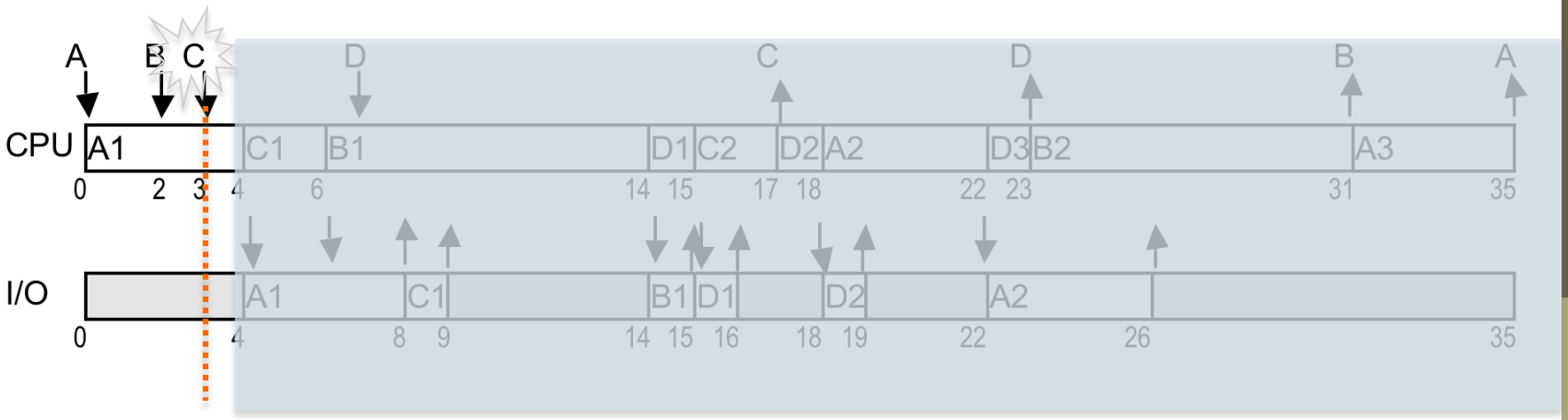
Proc_id : next_cpu_burst



2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

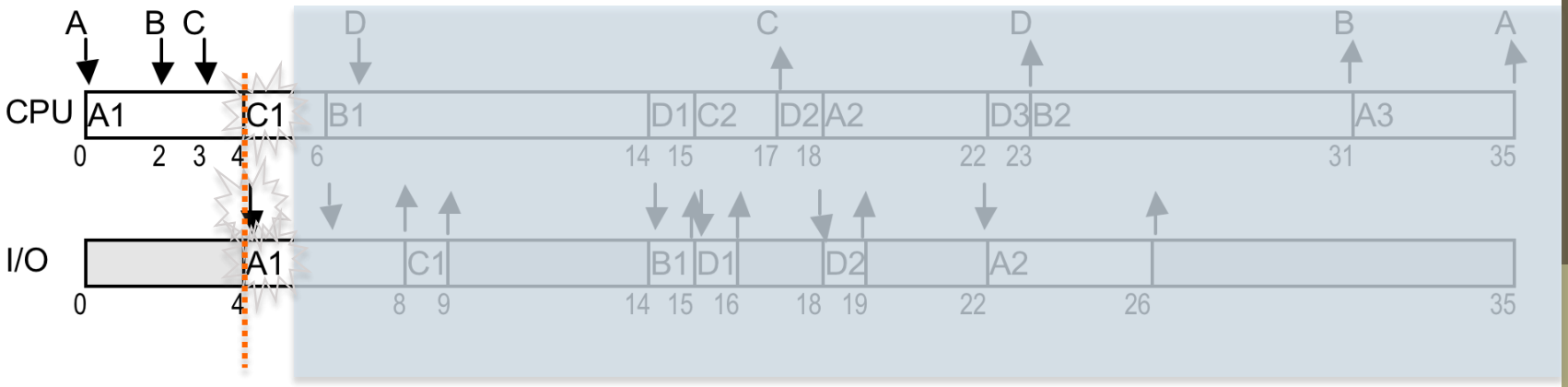
RQ: B:8 C:2



2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

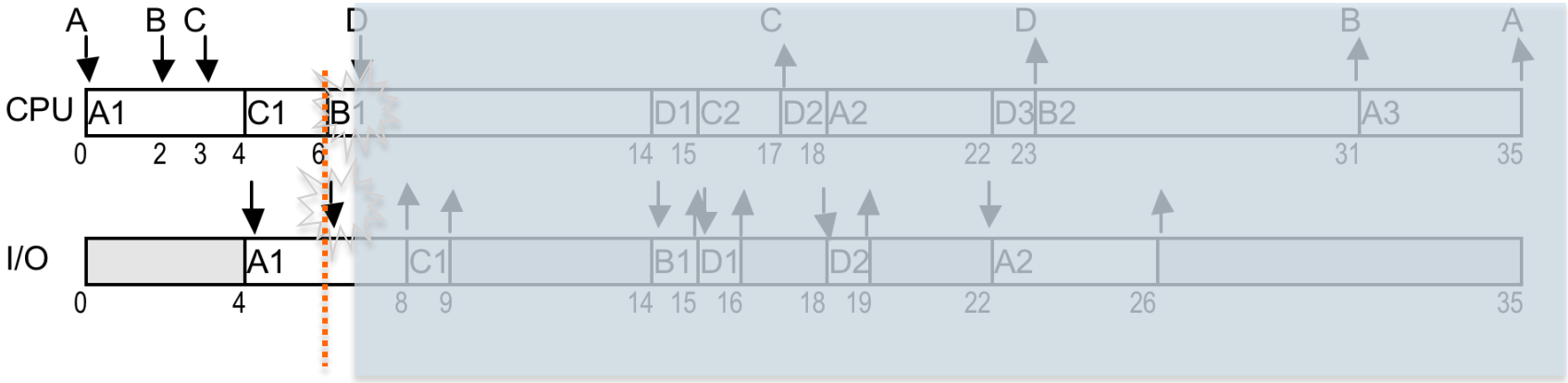
RQ: B:8 C:2



2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

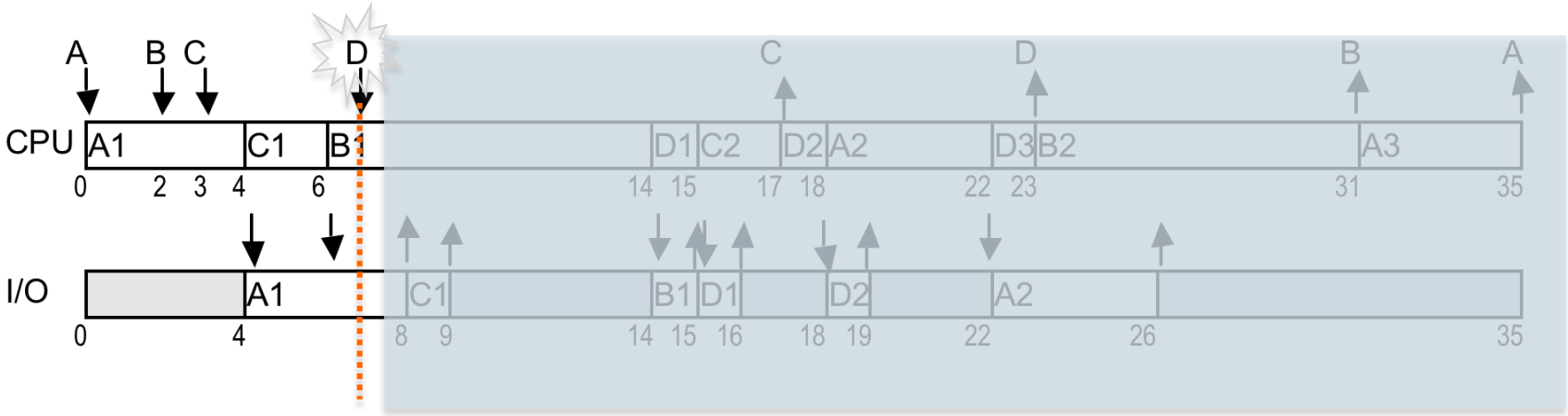
RQ: B.8



2.5.4. Chiến lược công việc ngắn nhất

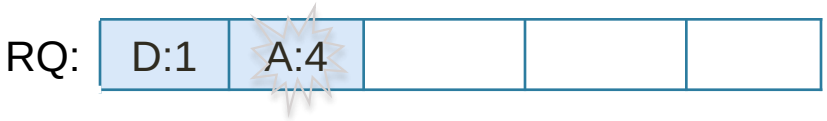
Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

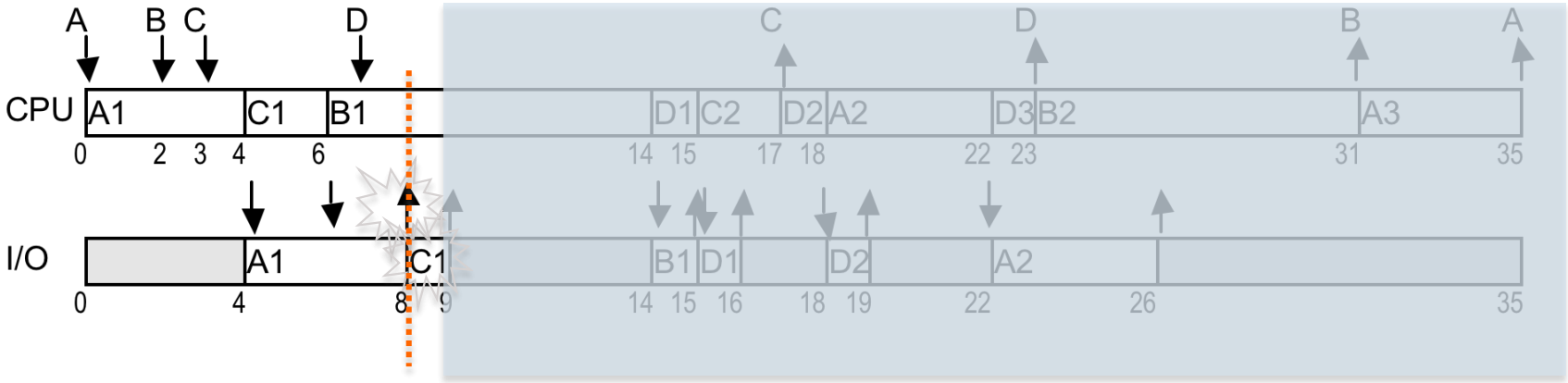
RQ: D:1



2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

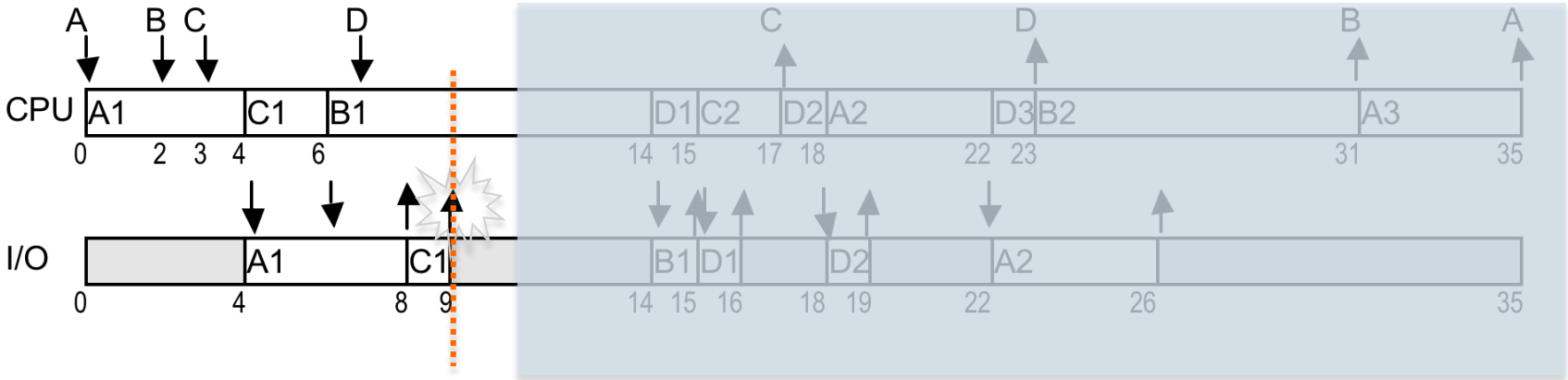




2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

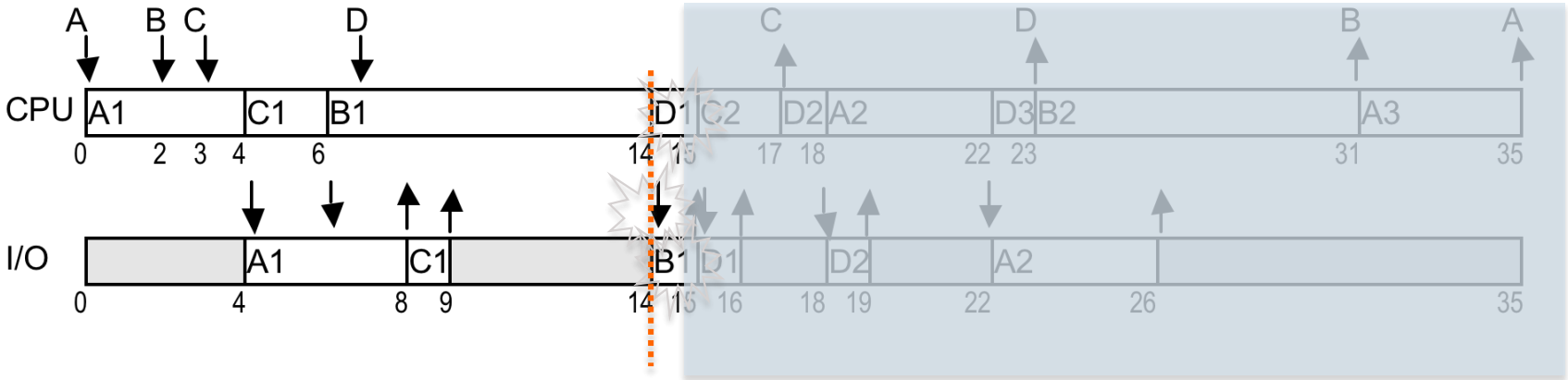
RQ: D:1 | A:4 | C:2



2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

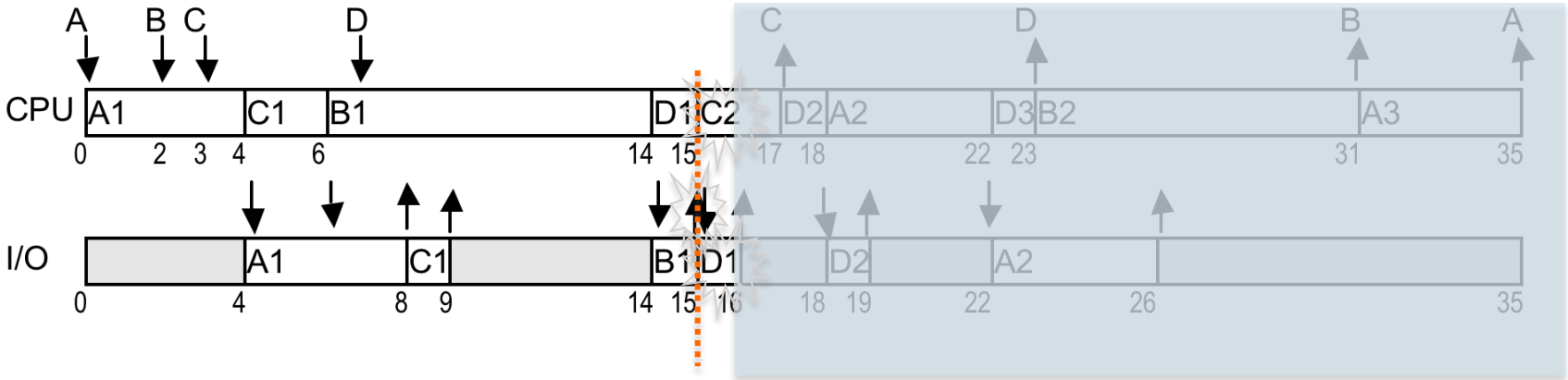
RQ: D:1 | A:4 | C:2



2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

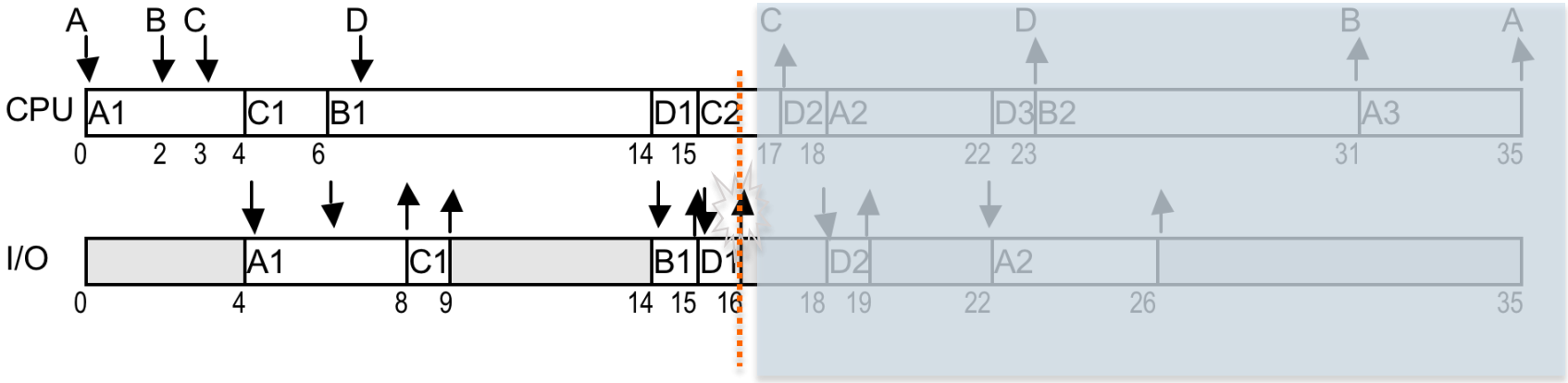
RQ: A:4 | C:2 | B:8



2.5.4. Chiến lược công việc ngắn nhất

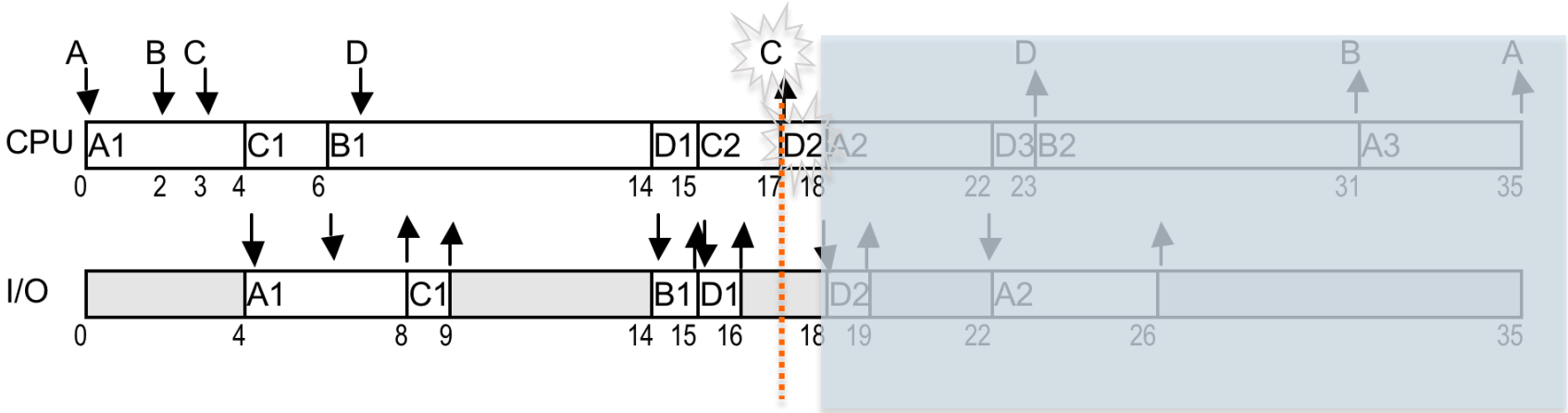
Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

RQ: A:4 | B:8 | D:1



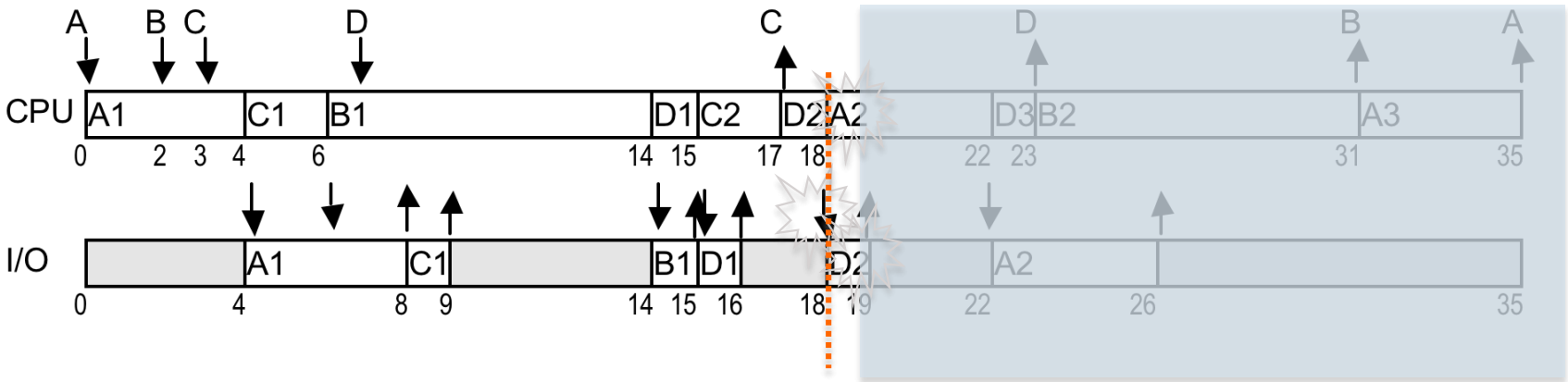
2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



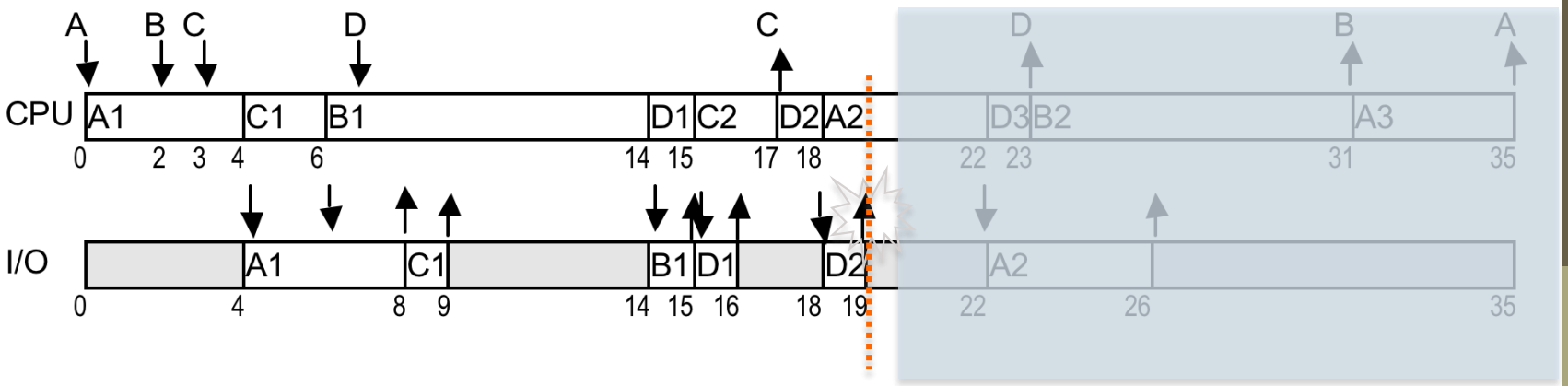
2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



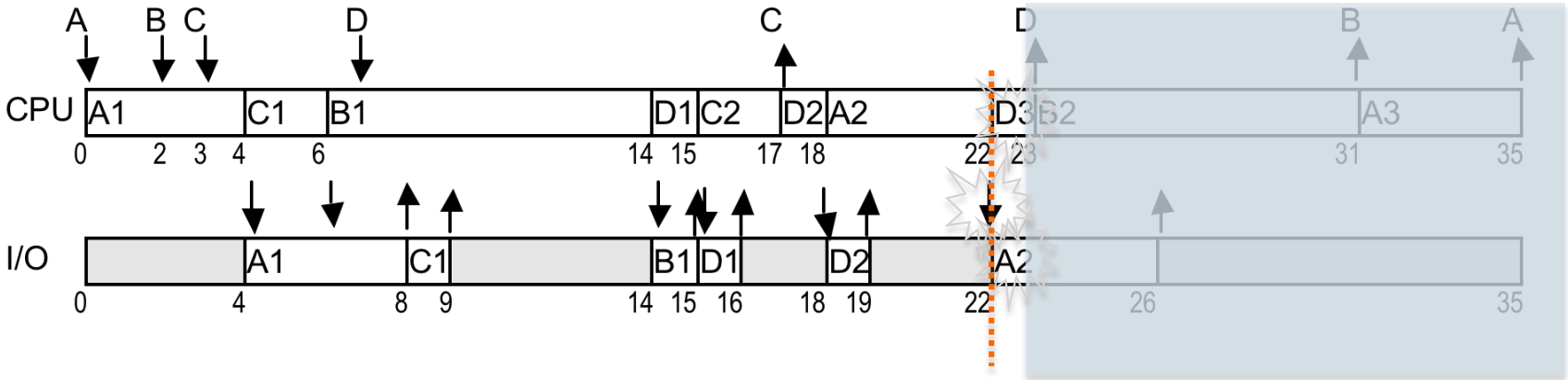
2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



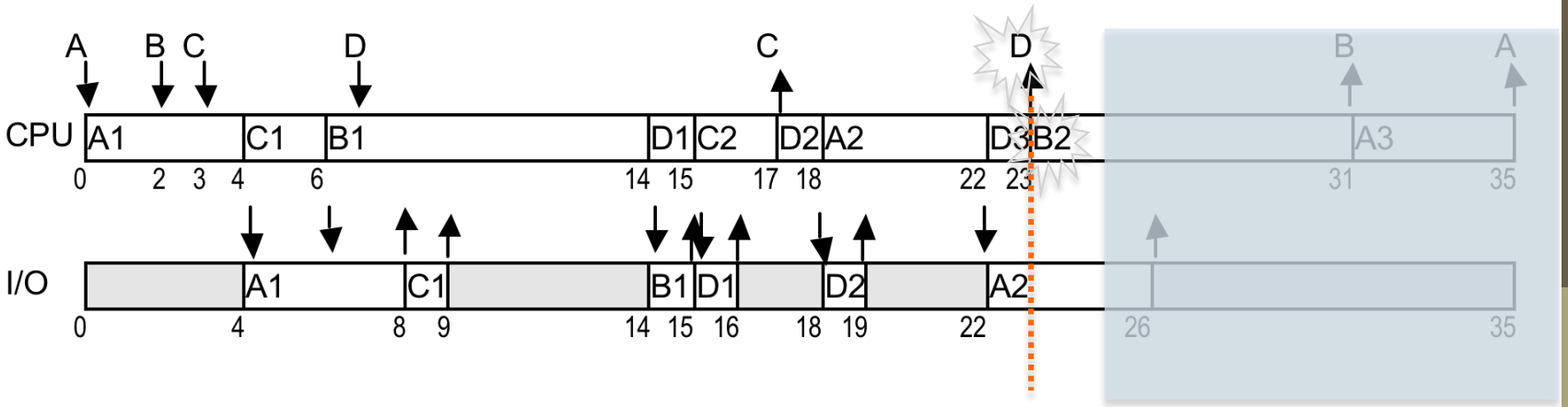
2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



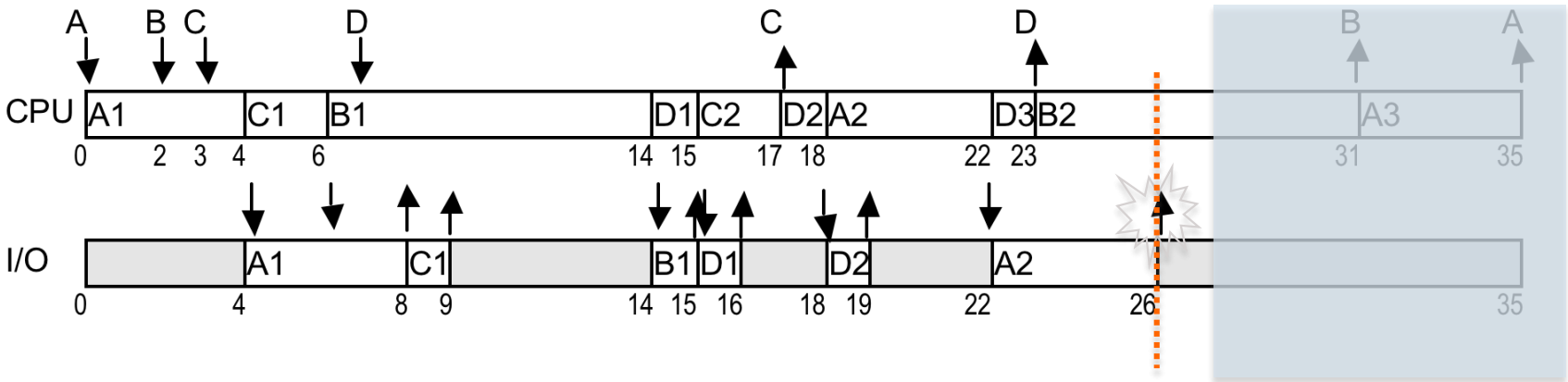
2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



2.5.4. Chiến lược công việc ngắn nhất

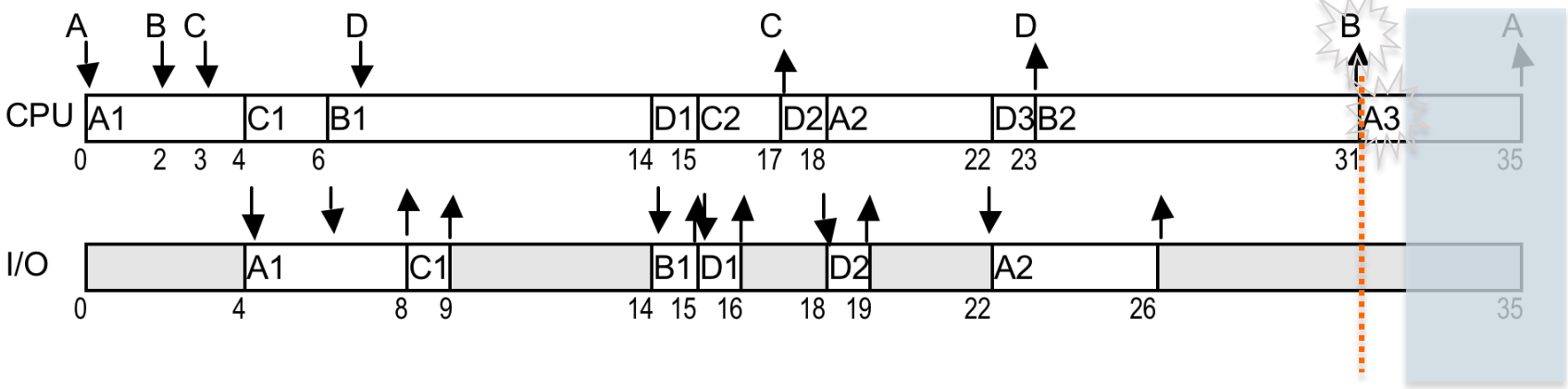
Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1



2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

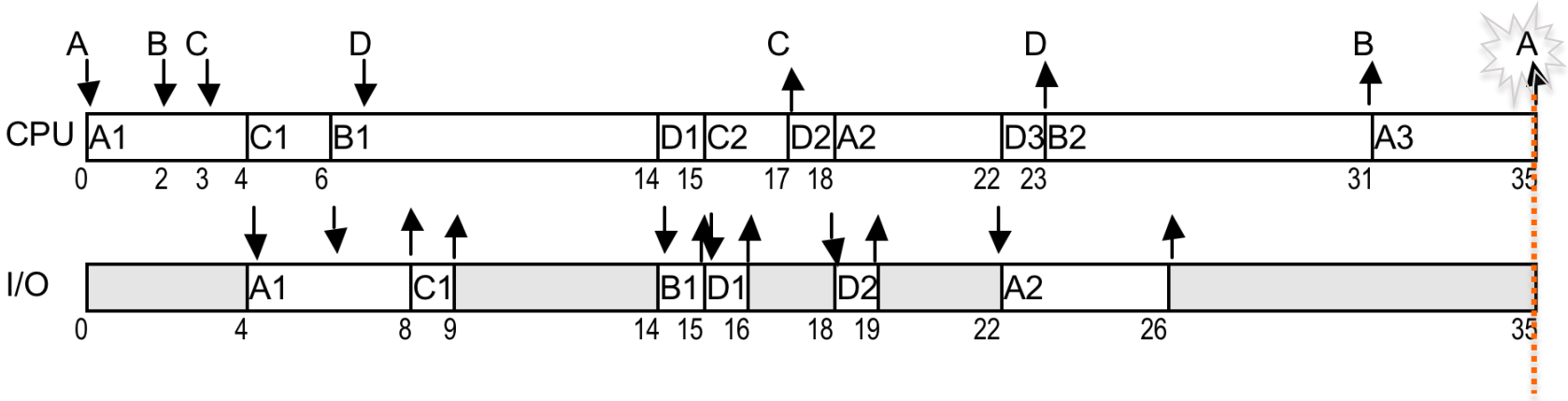
RQ: A.4



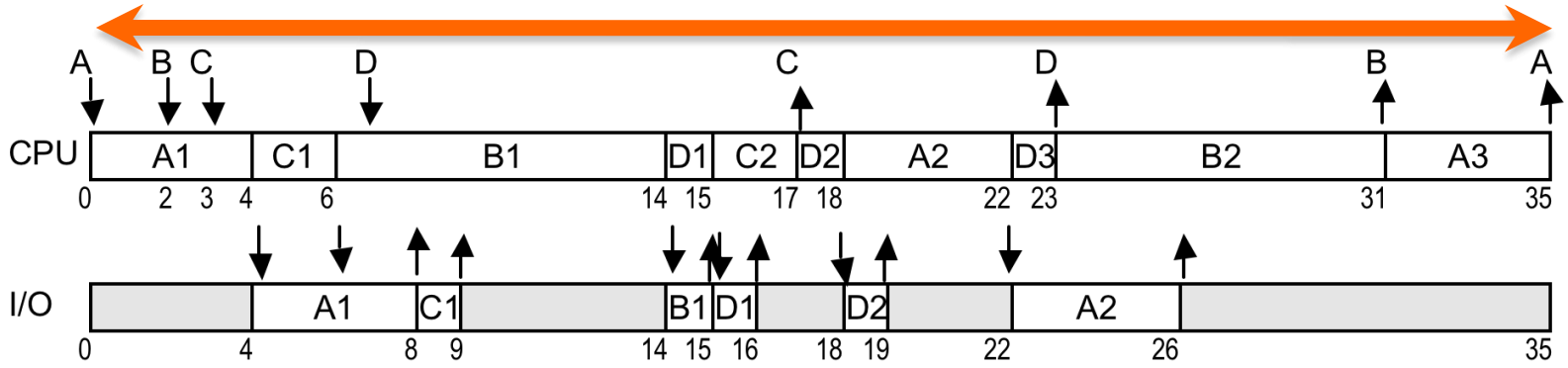
2.5.4. Chiến lược công việc ngắn nhất

Process	Arrival time	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4	4	4	4	4
B	2	8	1	8	-	-
C	3	2	1	2	-	-
D	7	1	1	1	1	1

RQ:

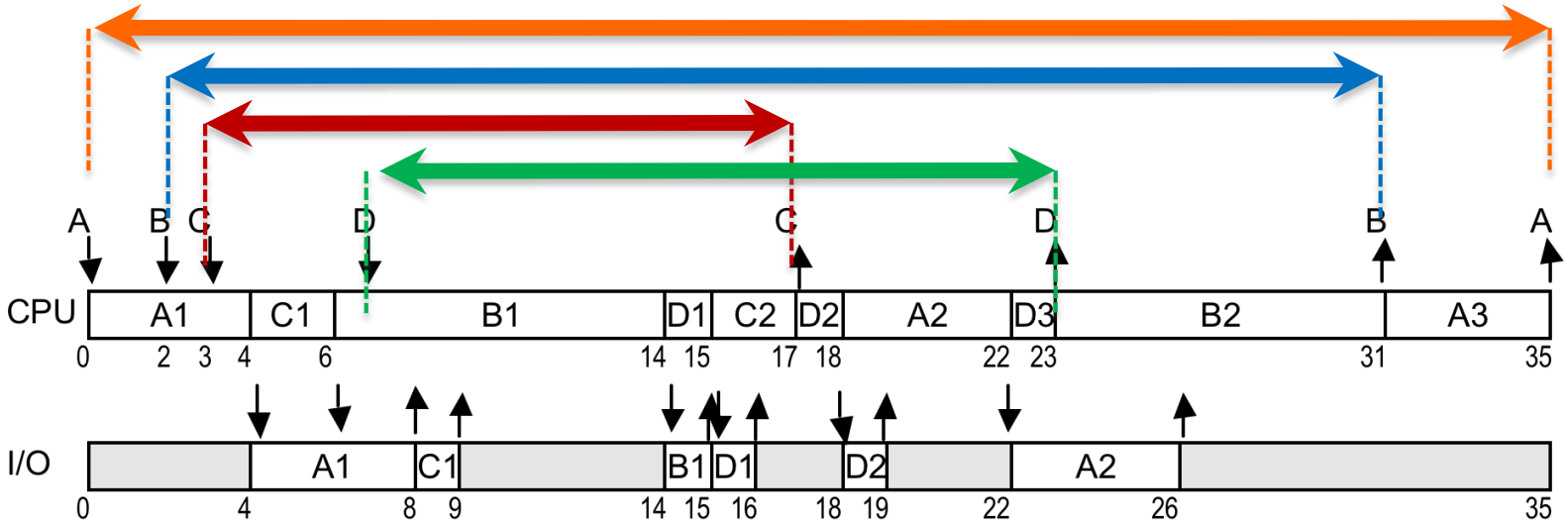


2.5.4. Chiến lược công việc ngắn nhất



- Processor utilization = (35 / 35) * 100 = 100 %
- Throughput = 4 / 35 = 0.11

2.5.4. Chiến lược công việc ngắn nhất



- Turn around time:

$tat_A = 35 - 0 = 35$

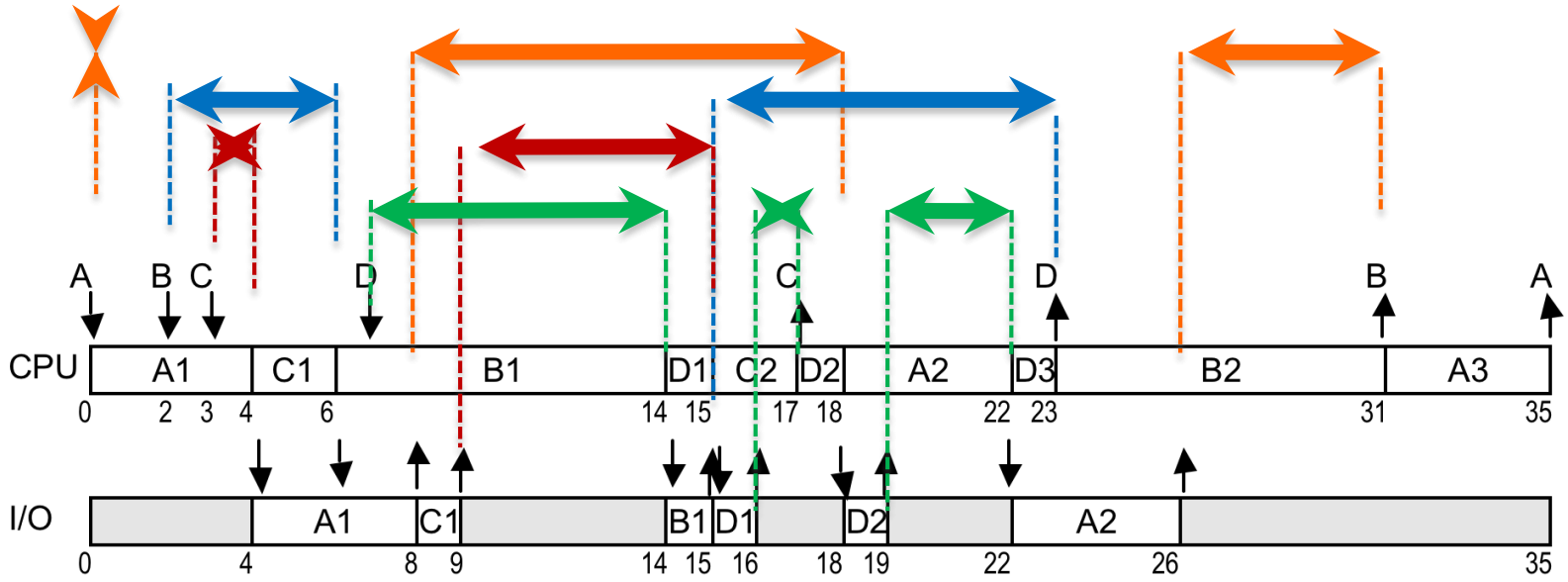
$tat_B = 31 - 2 = 29$

$tat_C = 17 - 3 = 14$

$tat_D = 23 - 7 = 16$

$tat_{AVG} = (35 + 29 + 15 + 16) / 4 = 23.5$

2.5.4. Chiến lược công việc ngắn nhất



- Waiting time:

$wt_A = (0 - 0) + (18 - 8) + (31 - 26) = 15$

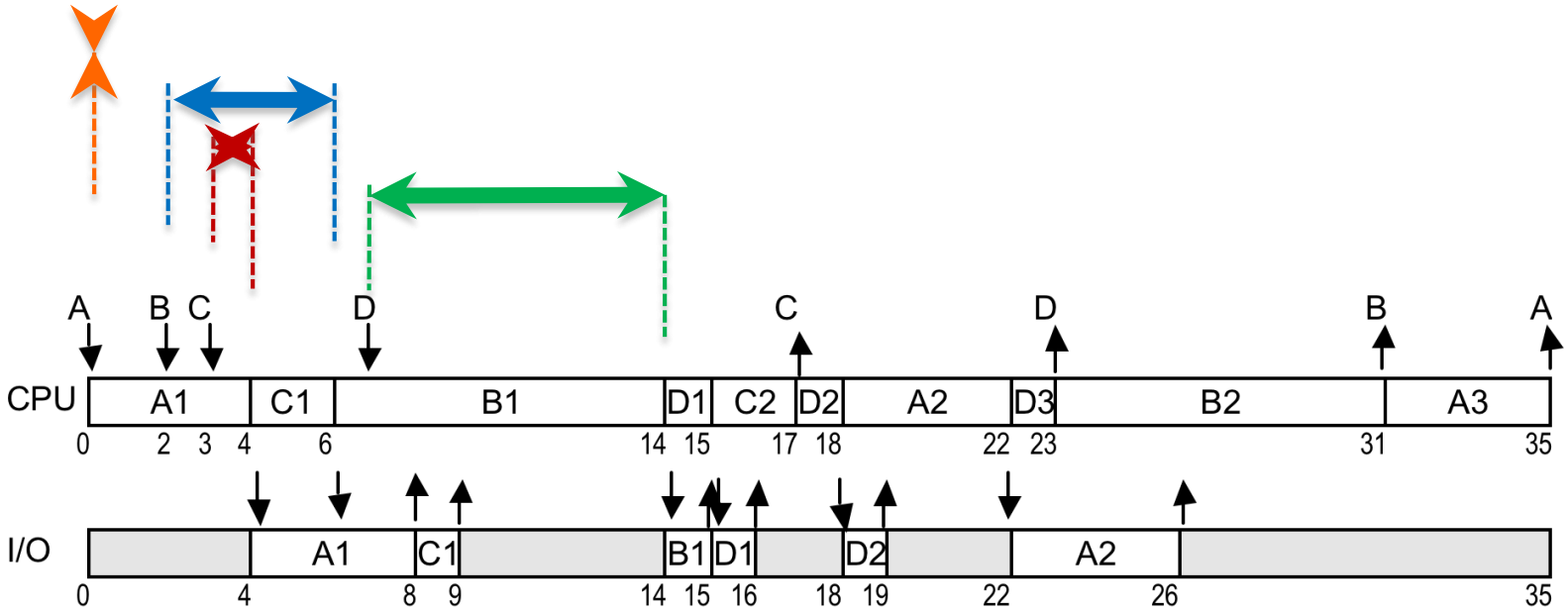
$wt_B = (6 - 2) + (23 - 15) = 12$

$wt_C = (4 - 3) + (15 - 9) = 7$

$wt_D = (14 - 7) + (17 - 16) + (22 - 19) = 11$

$wt_{AVG} = (15 + 12 + 7 + 11) / 4 = 11.25$

2.5.4. Chiến lược công việc ngắn nhất



- Response time:

  $rt_A = 0 - 0 = 0$

  $rt_B = 6 - 2 = 4$

  $rt_C = 4 - 3 = 1$

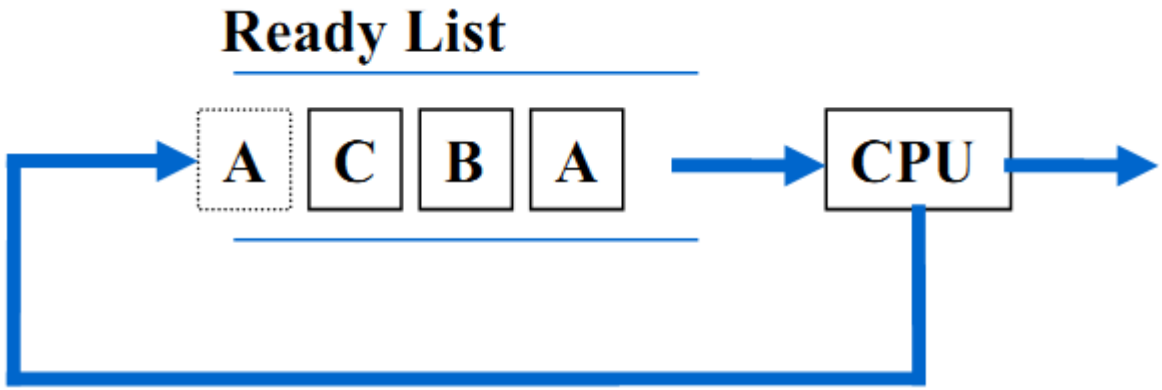
  $rt_D = 14 - 7 = 7$

  $rt_{AVG} = (0 + 4 + 1 + 7) / 4 = 3$

2.5.5. Chiến lược xoay vòng (RR)

- Các chiến lược đã xem xét là các chiến lược non-preemptive.
- Các chiến lược Preemptive:
  - Shortest-Remaining-Time-First (SRTF);
  - Round-Robin Scheduling.
- Chiến lược phân phối xoay vòng (**Round Robin**):
  - Tiến trình nào vào danh sách Ready trước được cấp processor trước
  - Mỗi tiến trình chỉ được sử dụng processor trong 1 khoản thời gian bằng nhau được gọi là Quantum

2.5.5. Chiến lược xoay vòng (RR)



Tiến trình	Thời điểm vào	t/g xử lý
P1	0	24
P2	1	3
P3	2	3
Quantum = 4;		

Tiến trình	P1	P2	P3	P1	P1	P1	P1	
Thời điểm	0	4	7	10	14	18	22	

2.5.6. Chiến lược nhiều cấp độ ưu tiên

- **Chiến lược nhiều cấp độ ưu tiên:**
  - Phân lớp các tiến trình tùy theo độ ưu tiên của chúng để có cách thức điều phối thích hợp cho từng nhóm;
  - Mỗi danh sách bao gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật điều phối thích hợp;
  - Ngoài ra, cần có một giải thuật điều phối giữa các nhóm;
  - Một tiến trình thuộc về danh sách ở cấp ưu tiên i sẽ chỉ được cấp phát CPU khi các danh sách ở cấp ưu tiên lớn hơn i đã rỗng;

2.6. Các giải pháp điều phối qua đoạn găng

- **Các giải pháp phần cứng**
  - Dùng cặp chỉ thị STI & CLI
  - Dùng chỉ thị TSL (Test and set)
- **Các giải pháp phần mềm (dùng biến khoá)**
  - Dùng biến khoá chung
  - Dùng biến khoá riêng
- **Các giải pháp HĐH và ngôn ngữ lập trình**
  - Giải pháp dùng Semaphore (sự đánh tín hiệu bằng cờ) Giải pháp dùng Monitors;
  - Giải pháp trao đổi Message (thông điệp);
  - …

Một vài ví dụ về tiến trình

- Tạo tiến trình con trong UNIX, LINUX:

```
#include <stdio.h>
#include <unistd.h>
int main (int argc, char *argv[]){
  int       pid;
  /* create a new process */
  pid = fork();

  if (pid > 0){
    printf("This is parent process");
    wait(NULL);
    exit(0);
  }
  else if (pid == 0)
  {
    printf("This is child process");
    execlp("/bin/ls", "ls", NULL);
    exit(0);
  }
  else {
    printf("Fork error\n");
    exit(-1);
  }
}
```