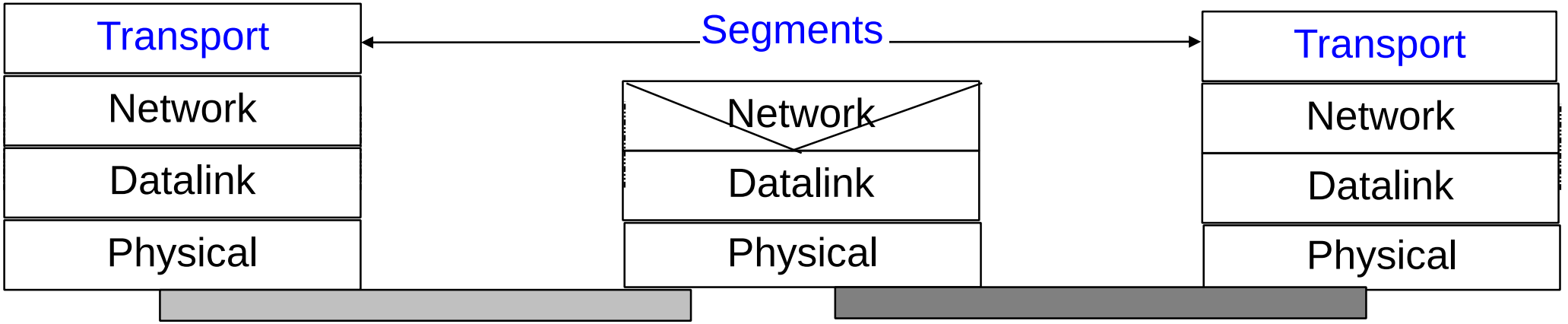
Mạng máy tính : Nguyên lý, Giao thức và luyện tập

Phần 3 : Tầng Giao vận

Phần 3 : Tầng Giao vận

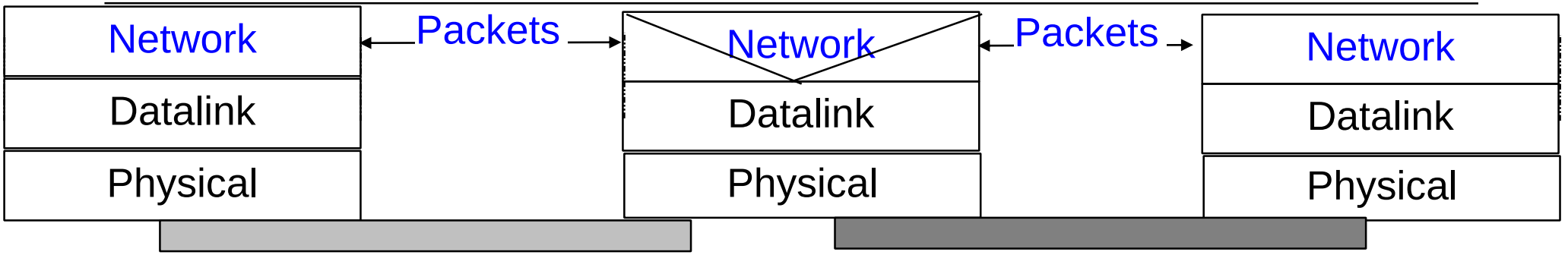
- Kiến thức cơ bản
- Xây dựng tầng giao vận tin cậy:
  - Truyền dữ liệu tin cậy
  - Thiết lập kết nối
  - Giải phóng kết nối
- UDP : giao thức giao vận đơn giản không kết nối
- TCP : giao thức truyền tin tin cậy có kết nối

Tầng Giao vận



- Mục đích
  - Cải thiện dịch vụ cung cấp bởi tầng Mạng cho phép các ứng dụng có thể sử dụng được
    - Đảm bảo tính tin cậy
    - Dồn kênh
- Các dịch vụ của tầng giao vận
  - Dịch vụ không kết nối, không tin cậy
  - Dịch vụ có kết nối, tin cậy

Tầng Mạng



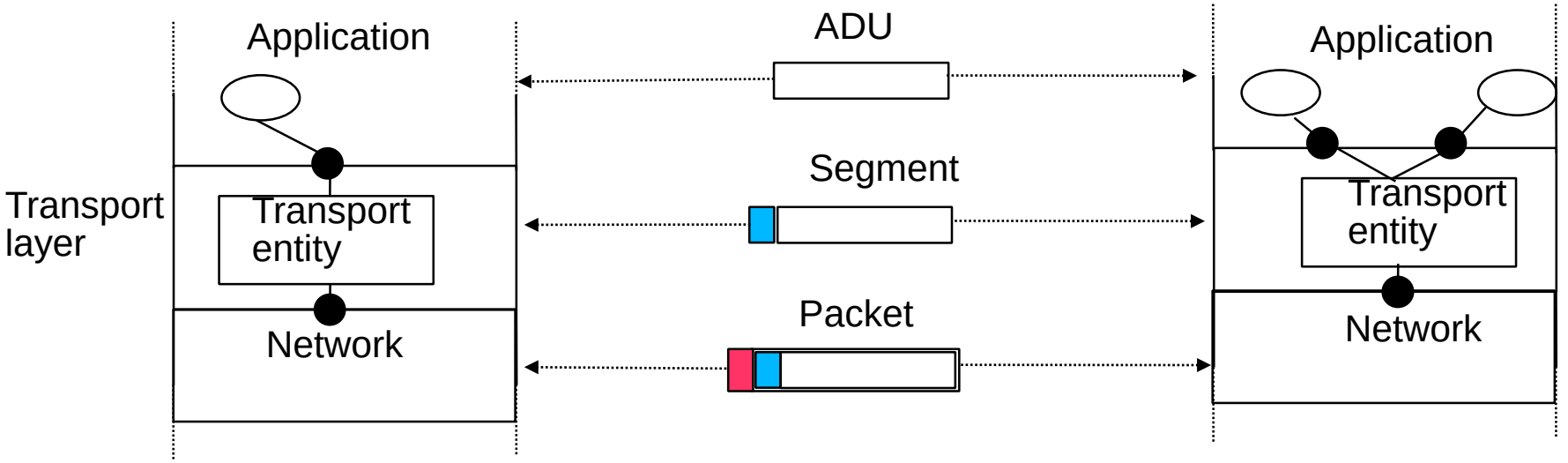
- Các dịch vụ ở tầng mạng
  - Dịch vụ không kết nối, không tin cậy:
    - Gói tin có thể bị mất
    - Gói tin bị lỗi do đường truyền
    - Thứ tự gói tin không được đảm bảo
    - Gói tin có thể bị trùng
    - Kích cỡ gói tin bị hạn chế trong 64 KBytes
  - Vấn đề là làm sao giải quyết được các khó khăn trên để cung cấp dịch vụ khả dụng cho các ứng dụng ?

Tầng Giao vận

- Các vấn đề cần giải quyết ở tầng giao vận
  - Tầng giao vận phải cho phép hai ứng dụng có thể truyền tin được với nhau
    - Điều này đòi hỏi phải định danh các ứng dụng
  - Dịch vụ tầng giao vận phải khả dụng với ứng dụng
    - Phát hiện lỗi truyền
    - Sửa lỗi truyền
    - Phục hồi gói tin bị mất hoặc bị trùng
    - Và nhiều dịch vụ khác như
      - Không cần kết nối
      - Có kết nối
      - Gửi yêu cầu-phản hồi

Tầng giao vận

- Tổ chức bên trong của tầng giao vận
  - Tầng giao vận sử dụng dịch vụ cung cấp bởi tầng mạng
  - Tầng giao vận sử dụng các thực thể để trao đổi các đoạn tin (segments)



Phần 3 : Tầng Giao vận

- Kiến thức cơ bản
- Cơ chế truyền tin tin cậy
  - Truyền tin tin cậy
  - Thiết lập kết nối
  - Giải phóng kết nối
- UDP : giao thức giao vận đơn giản không kết nối
- TCP : giao thức giao vận tin cậy, có kết nối

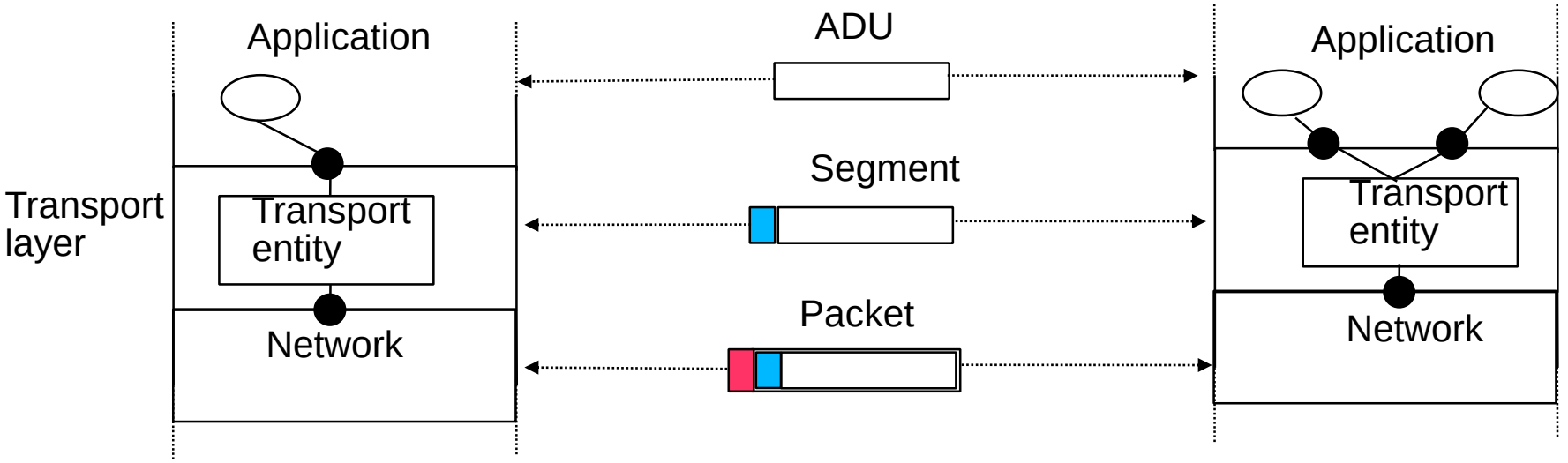
Các giao thức tầng giao vận

- Làm thế nào để cung cấp dịch vụ truyền tin tin cậy trên tầng giao vận

  - Giả thiết ban đầu
    1. Ứng dụng gửi các SDUs có kích cỡ nhỏ
    2. Tầng Mạng cung cấp một dịch vụ hoàn hảo
       1. Không có lỗi truyền trong gói tin
       2. Gói tin không bị mất
       3. Các gói tin không bị mất thứ tự
       4. Các gói tin không bị trùng
    3. Việc truyền tin là đơn công (một chiều)

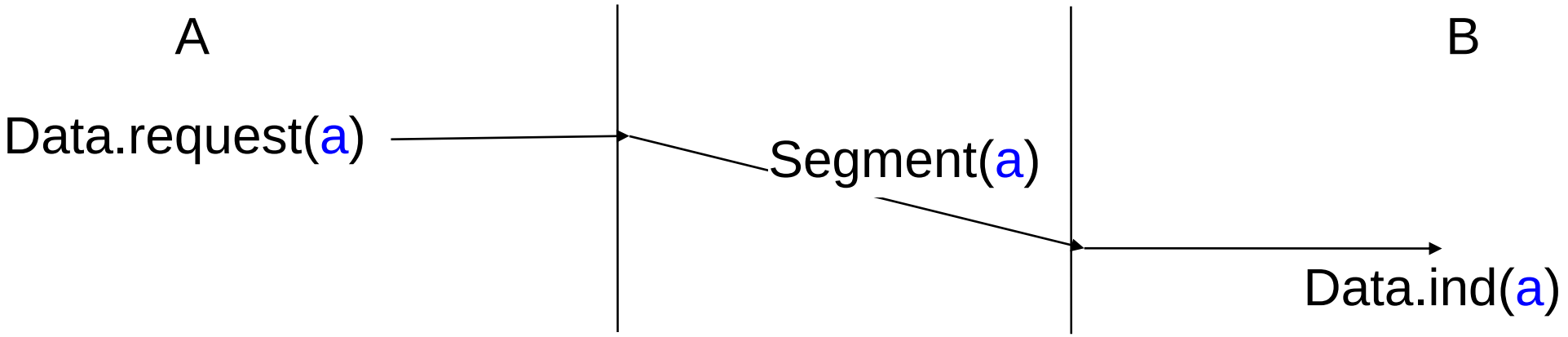
Các giao thức tầng giao vận (2)

- Môi trường tham chiếu của giả thuyết



- Kí pháp
  - `data.req` và `data.ind` là các hàm nguyên thủy để tương tác giữa các tầng ứng dụng và giao vận
  - `recv()` và `send()` là các tương tác giữa tầng giao vận và mạng

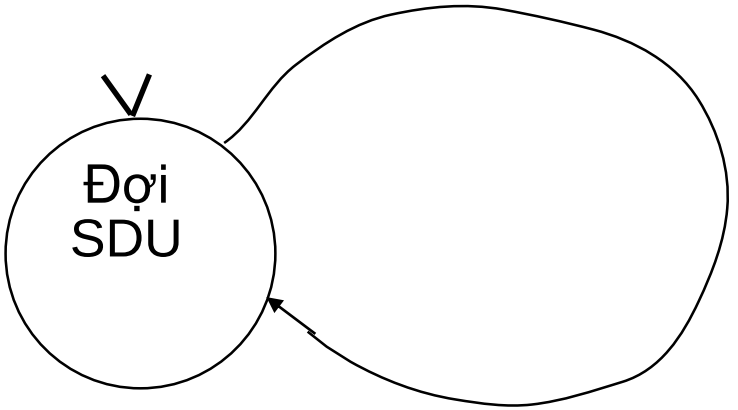
Giao thức 1 : Kiến thức căn bản



- Nguyên lý
  - Khi nhận được `data.request(SDU)`, thực thể tầng giao vận gửi một đoạn tin (segment) chứa SDU xuống tầng mạng (`send(SDU)`)
  - Khi nhận được nội dung của gói tin (packet) từ tầng mạng (`recv(SDU)`), thực thể giao vận gửi SDU tìm được trong goi tin tới người dùng bằng cách sử dụng `data.ind(SDU)`

Giao thức 1 dưới dang FSM (mô hình trạng thái)

- Bên gửi

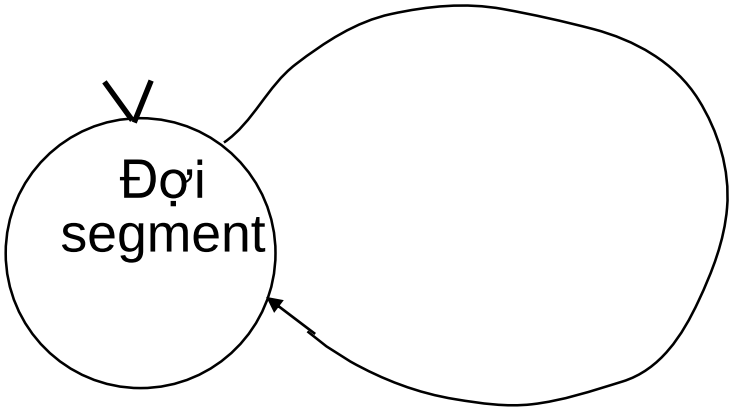


Data.req(SDU)

---

send(SDU)

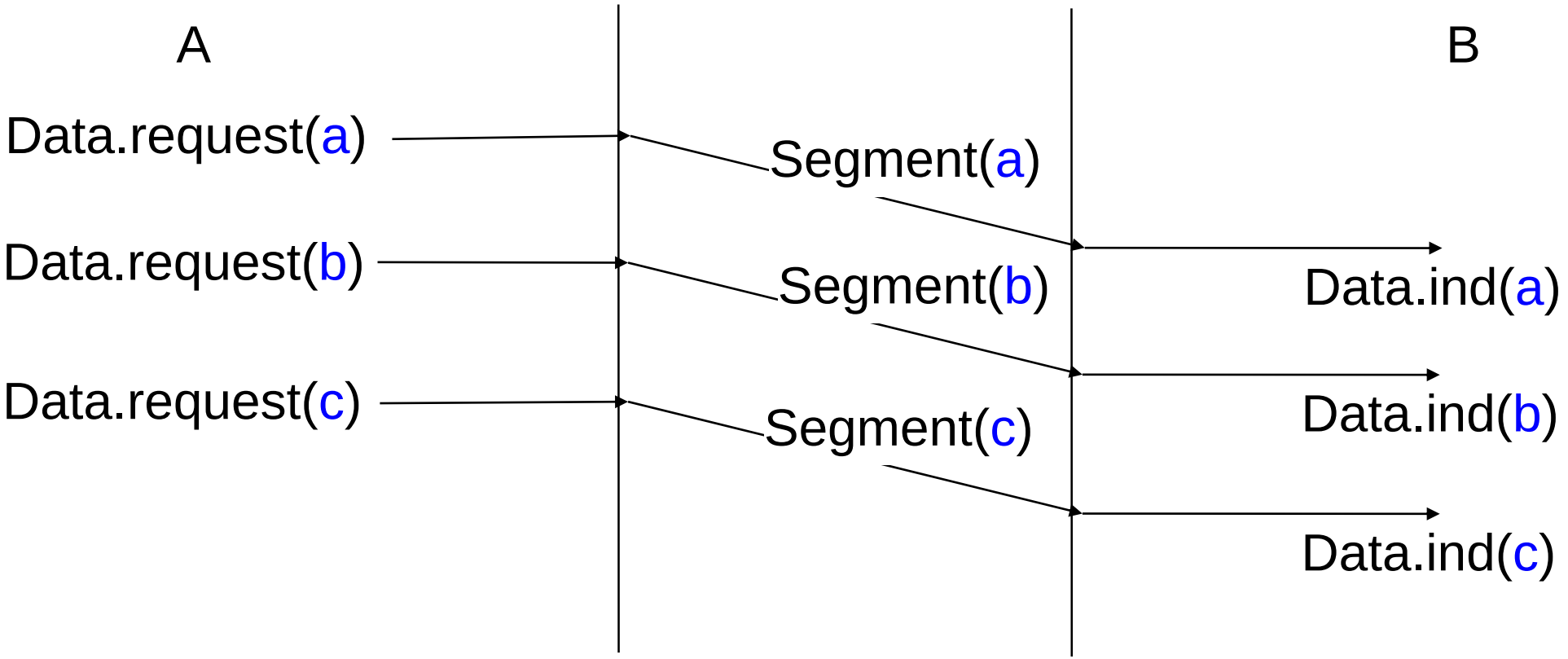
- Bên nhận



recvd(SDU)

Data.ind(SDU)

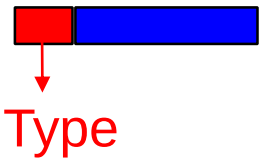
Giao thức 1 : Ví dụ



- Ví dụ
  - Điều gì xảy ra nếu bên nhận chậm hơn rất nhiều bên gửi?
    - e.g. Bên nhận có thể xử lý 1 segment trên giâu trong khi bên gửi có thể gửi 10 segments trên giây ?

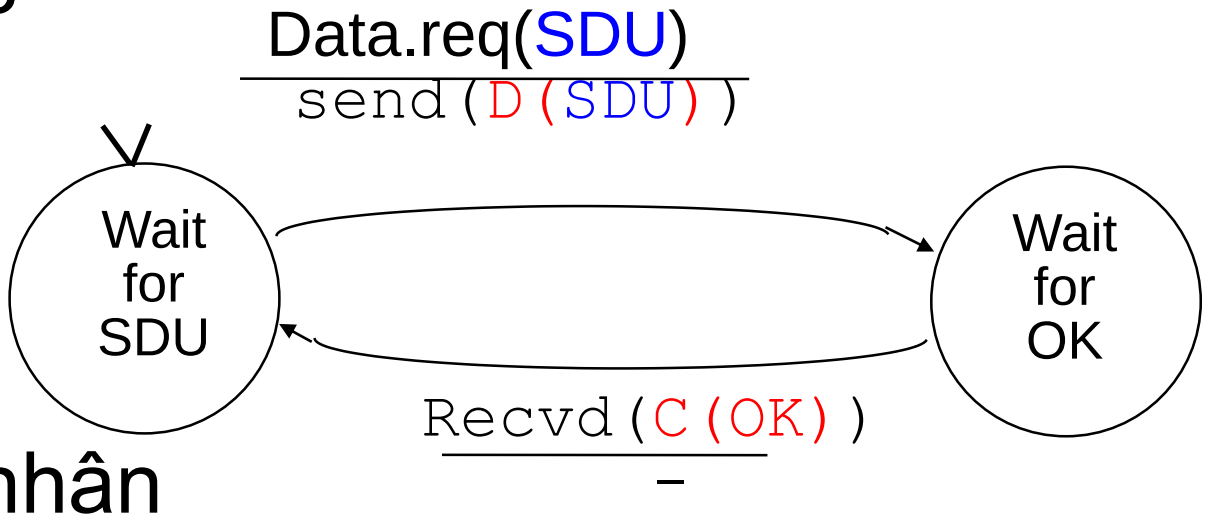
Giao thức 2

- Nguyên lý
  - Sử dụng một segment điều khiển (OK) được gửi bởi bên nhận sau khi đã nhận xong một
  - Có thể tạo ra lặp trả lời giữa bên gửi và bên nhận
- Kết quả là
  - Có hai loại segment
    - Segment dữ liệu chưa SDU
      - Kí pháp : D(SDU)
    - Segment điều khiển
      - Kí pháp : C(OK)
  - Khuôn dạng Segment như sau
    - Phải có ít nhất một bit trong tiêu đề chỉ ra kiểu của segment

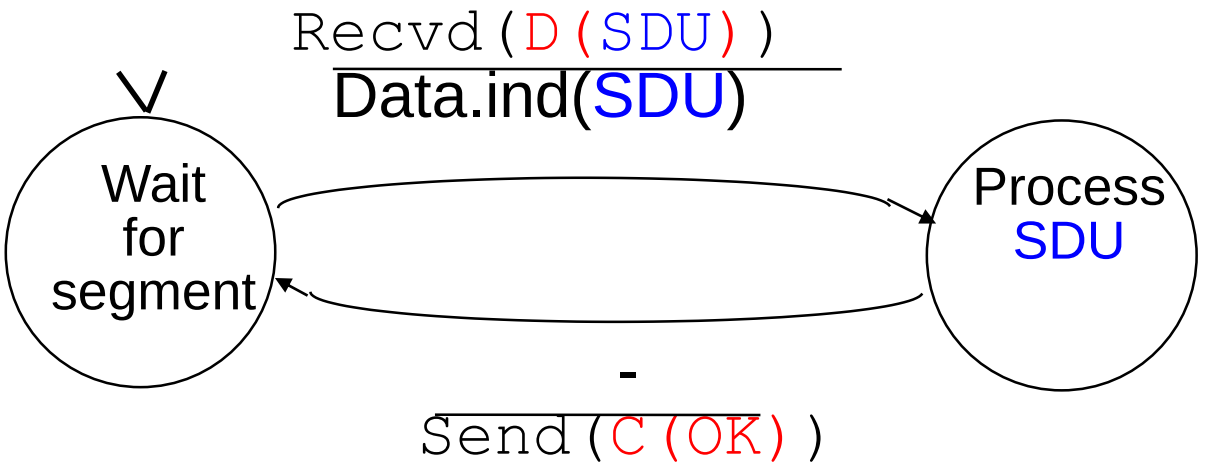


Giao thức 2 (tiếp tục.)

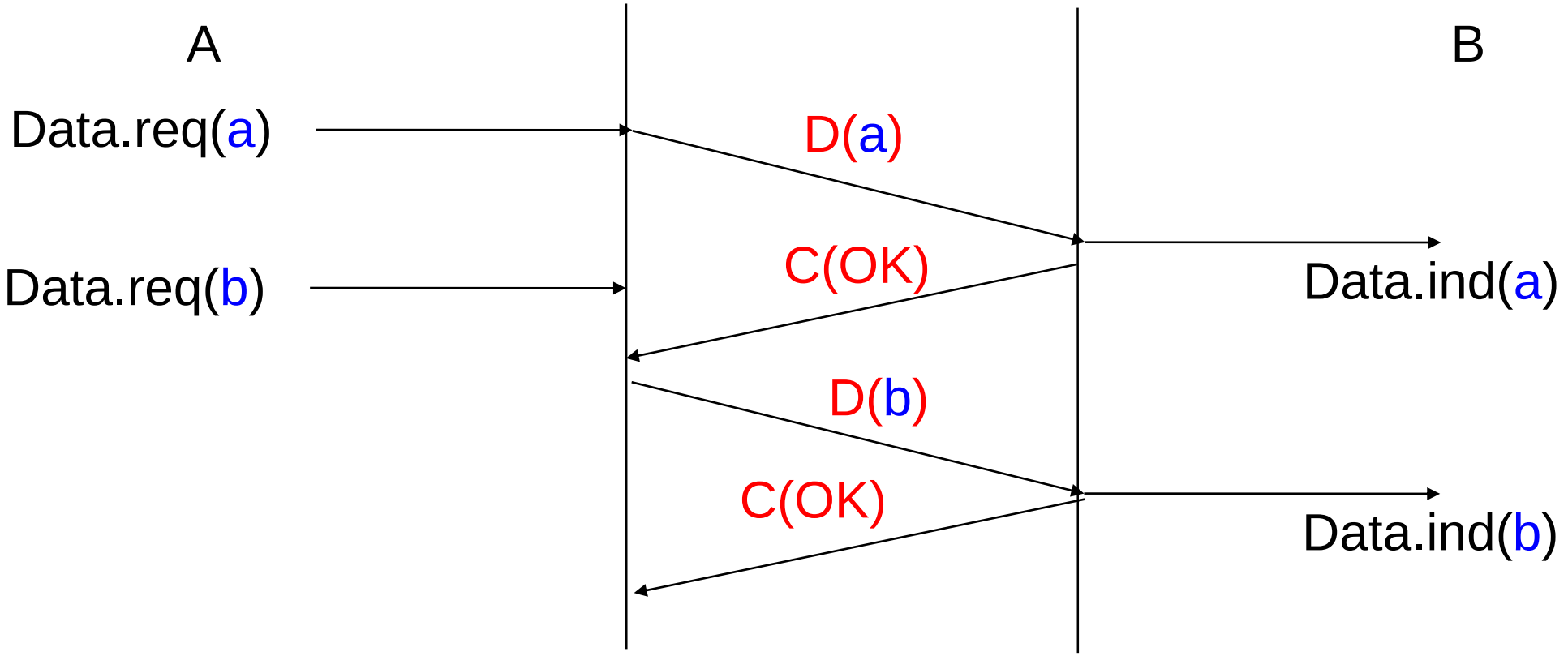
- Bên gửi



- Bên nhận



Giao thức 2 : Vi dụ



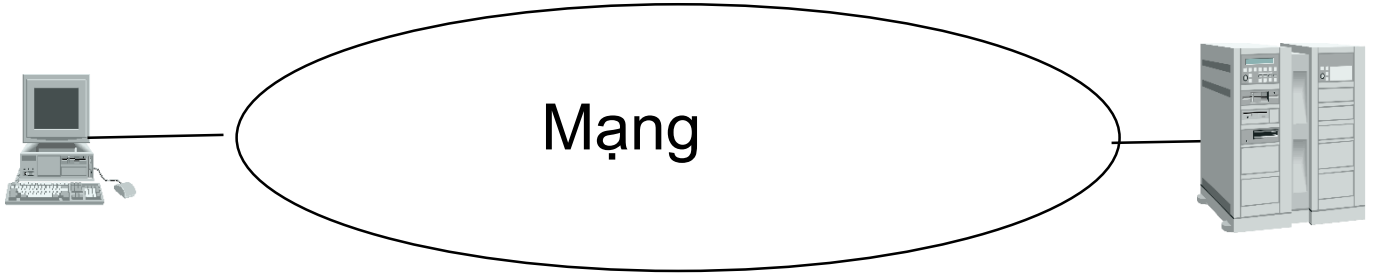
- Phía gửi chỉ gửi segment sau khi có sự xác nhận của phái nhận

Giao thức 3

- Làm sao cung cấp dịch vụ truyền tin tin cậy trên tầng giao vận
- Giả thiết
  1. Ứng dụng gửi các SDUs có kích cỡ nhỏ
  2. Tầng mạng cung cấp dịch vụ hoản hảo
     1. Có thể có lỗi đường truyền
     2. Không bị mất gói tin
     3. Không bị mất thứ tự gói tin
     4. Không bị trùng gói tin
  3. Truyền dữ liệu đơn công

Lỗi trên đường truyền

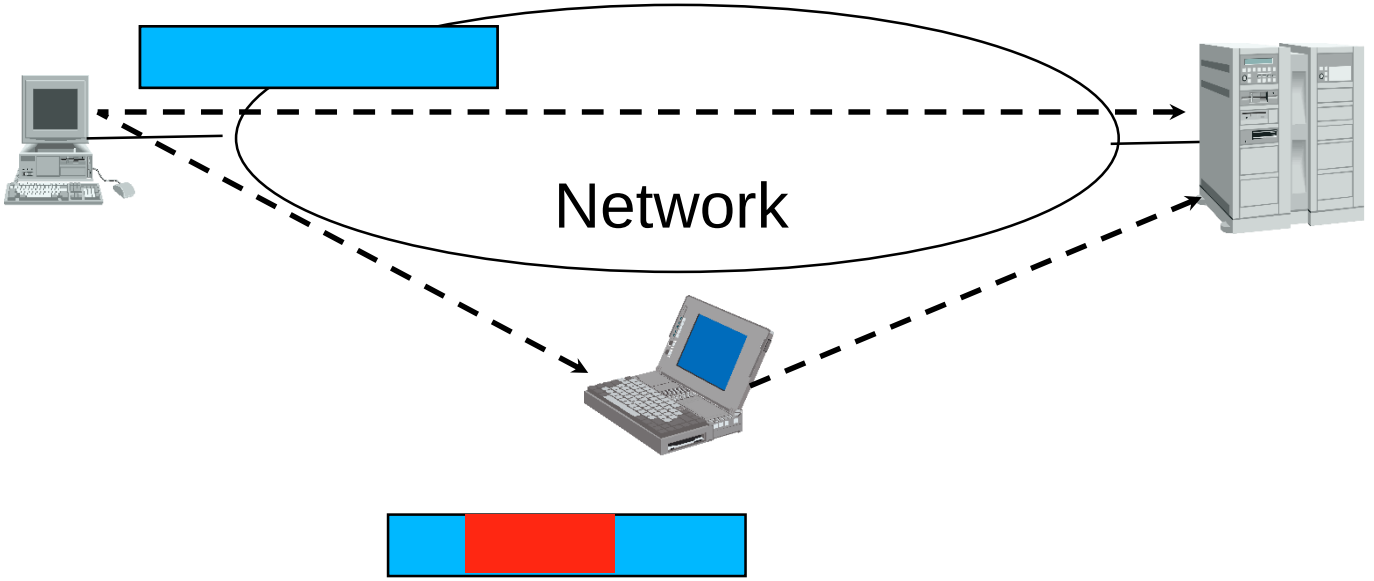
- Kiểu lỗi truyền nào được xem xét trên tầng giao vận ?



- Lỗi truyền tầng vật lý gây ra bởi bản chất của đường truyền
  - Lỗi đơn ngẫu nhiên
    - 1 bít bị mất trong 1 segment
  - Lỗi loạt ngẫu nhiên
    - Một nhóm n bit trong segment bị lỗi
      - Đa số bit trong nhóm bị sai

Vấn đề an ninh và lỗi đường truyền

- Thông tin được gửi qua mạng có thể bị xâm phạm bởi kẻ xâm nhập chứ không phải bị lỗi



- Trường hợp này nên giải quyết bằng các giao thức đảm bảo an ninh chứ không phải cơ chế của tầng giao vận

Làm sao để phát hiện lỗi truyền ?

- Nguyên lý
  - Bên gửi gửi một số thông tin điều khiển bên trong segment
    - Thông tin điều khiển được tính toán trên toàn bộ segment và được đặt ở đầu hoặc cuối segment

  - Bên nhận kiểm tra xem thông tin điều khiển có chính xác không bằng cách tính lại thông tin đó

Mã chẵn lẻ

- Là giải pháp đơn giản để phát hiện lỗi truyền
- Được sử dụng trên các đường truyền tốc độ thấp
  - e.g. modems nối với mạng điện thoại

- Mã lẻ
  - Với mỗi nhóm n bits, bên gửi tính bít thứ n+1th sao cho nhóm n+1 chứa số lượng lẻ các bit 1

    - **Ví dụ**

      0011010 0 11011001

- Mã chẵn

Mã checksum

- Ý tưởng
  - Vì Internet được cài đặt bằng phần mềm nên chúng ta muốn các giải thuật phát hiện lỗi đường truyền hiệu quả dễ cài đặt
- Giải pháp
  - Internet checksum
    - Bên gửi tính toán cho mỗi segment và trên toàn bộ segment số bù 1 của tổng các số có 16 bit trên segment đó
    - Bên nhận tính toán lại checksum trên mỗi segment nhận được và thẩm định lại xem có đúng không.

Internet checksum : Ví dụ

- Giả sử một segment bao gồm 48 bits

```
0110011001101100 0101010101010101 0000111100001111

1011101110111011

1100101011001010                  0011010100110101
```

Mã dư thừa vòng
Cyclical Redundancy Check (CRC)

- Nguyên lý
  - Cải thiện hiệu năng của Checksum bằng cách sử dụng mã đa thức

    - Bên gửi và bên nhận đồng thuận một mẫu r+1 bits gọi là Generator (G)
    - Bên gửi cộng mã CRC r bít với d bits dữ liệu của segment tạo ra d+r bits phải chia hết G theo modulo 2 arithmetic
      - D * 2^r XOR R = n*G

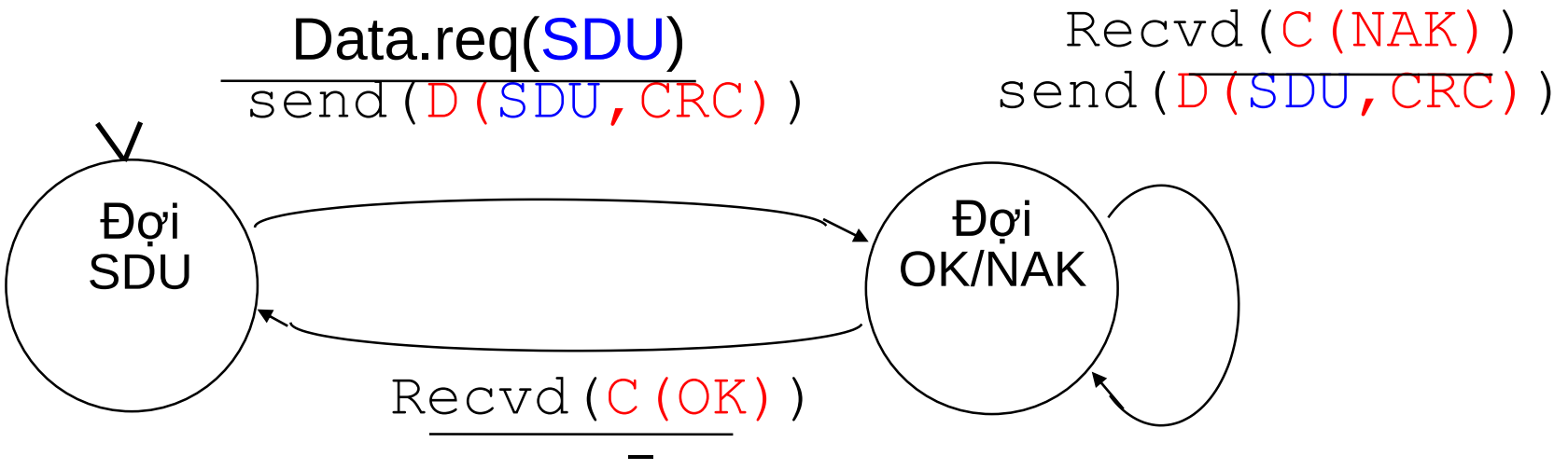
    - Các tính toán đều dựa trên modulo 2 và phép XOR
      - 1011 + 0101 = 1110 1001 + 1101 = 0100
      - 1011 - 0101 = 1110 1001 - 1101 = 0100

Phát hiện lỗi bít (2)

- Cách xử lý bên nhận
  - Nếu checksum chính xác
    - Gửi segment điều khiển OK tới bên nhận để
      - Xác nhận đã nhận segment dữ liệu
      - Cho phép bên gửi gửi segment tiếp theo
  - Nếu checksum không chính xác
    - Nội dung của segment bị xâm phạm và segment sẽ bị loại bỏ
    - Gửi một gói tin điều khiển đặc biệt (NAK) tới bên gửi để hỏi xem bên gửi có truyền lại gói tin bị xâm phạm không

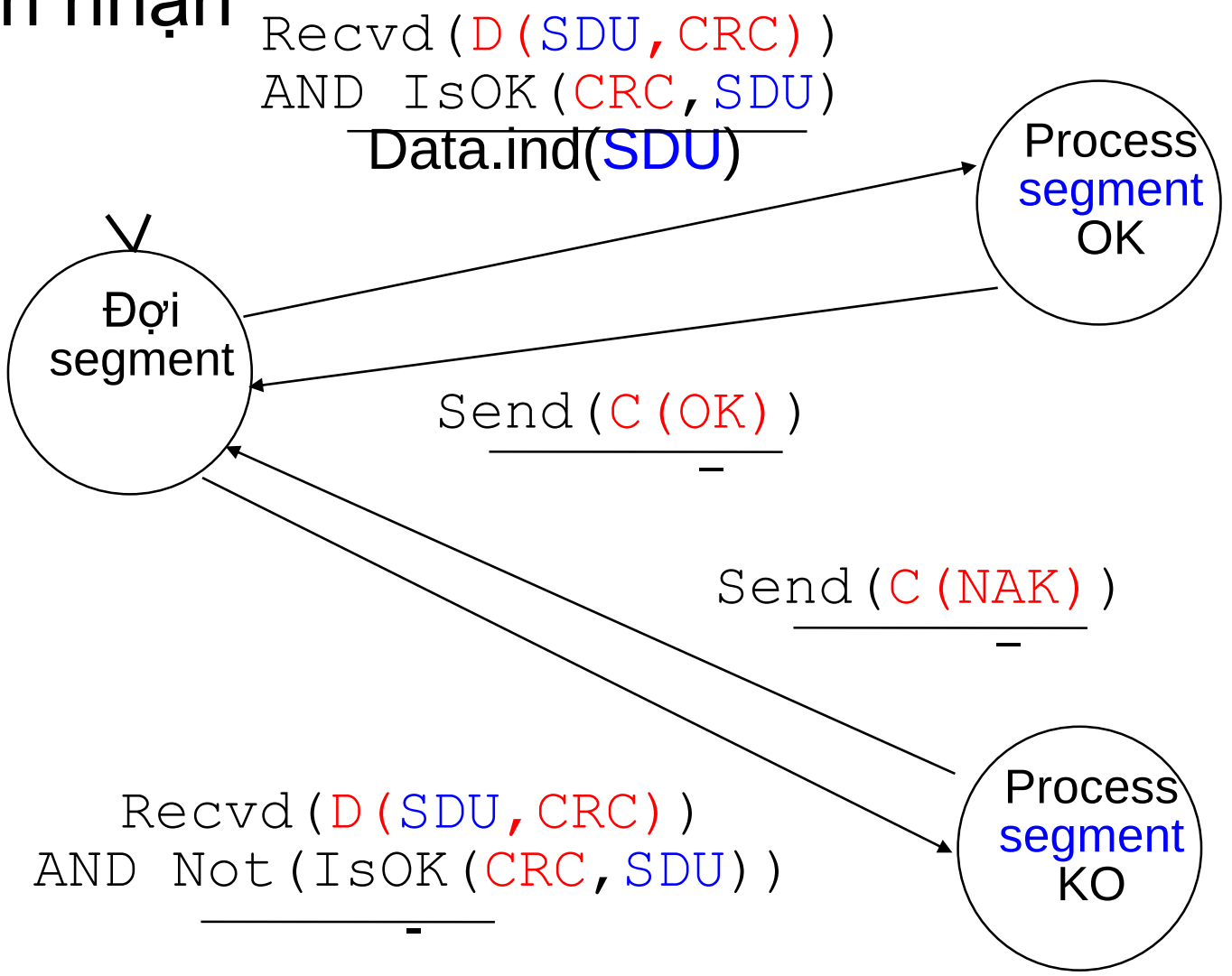
Giao thức 3a : Bên gửi

- Bên gửi

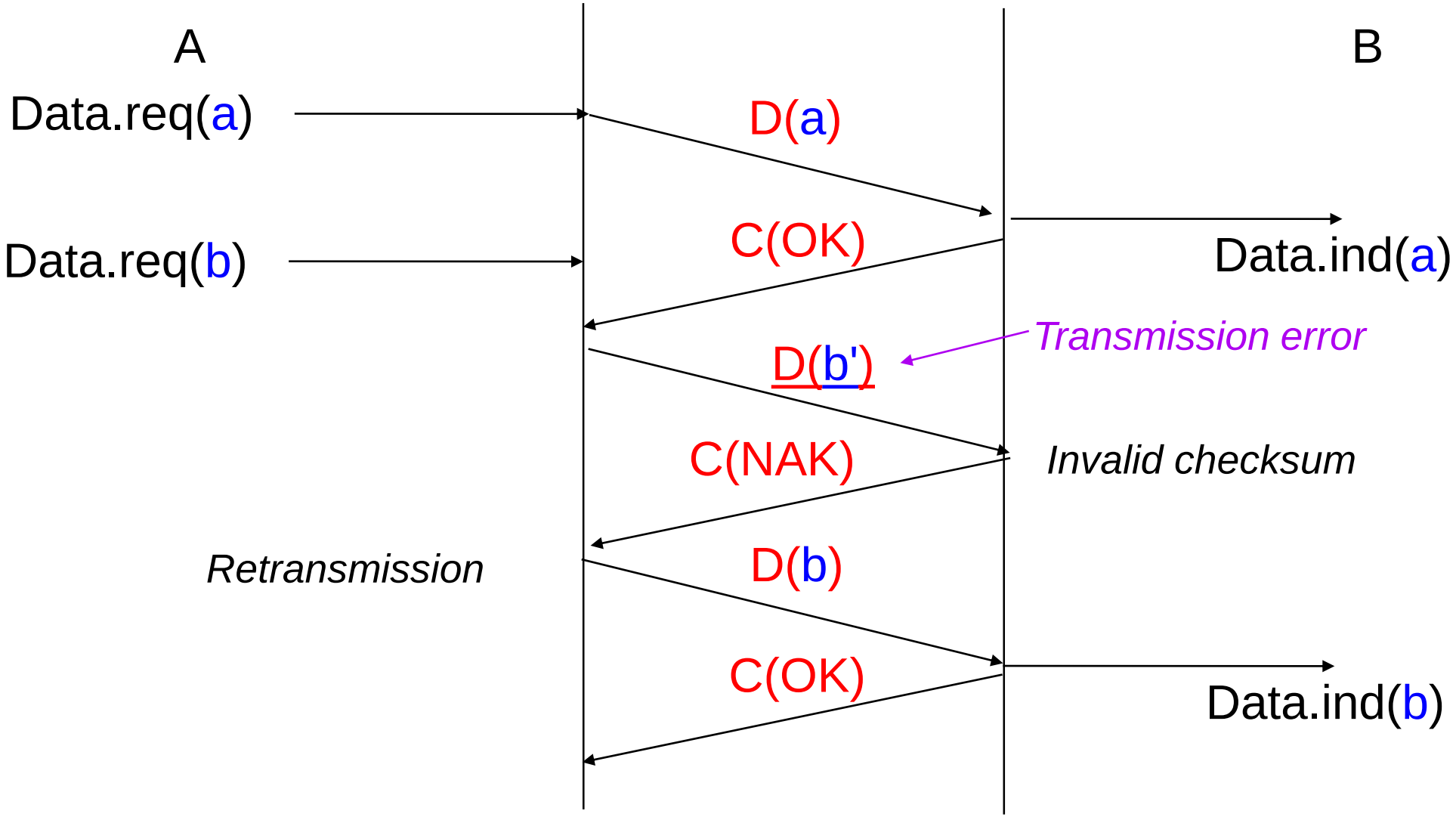


Giao thức 3a : Bên nhận

- Bên nhận



Giao thức 3a : Vi dụ

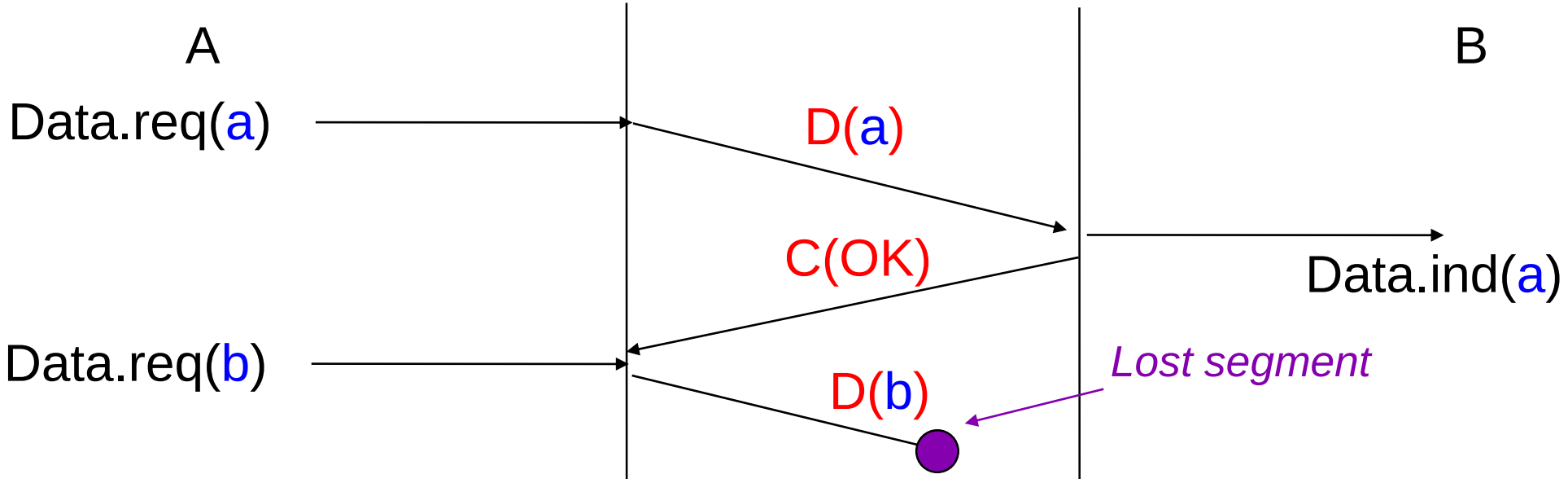


Giao thức 3b

- Làm sao để cung cấp dịch vụ tin cậy trên tầng giao vận ?
  - Giả thuyết
    1. Các ứng dụng gửi SDUs có kích cỡ nhỏ
    2. Tầng Mạng cung cấp một dịch vụ hoàn hảo
       1. Có thể có lỗi truyền
       2. Gói tin có thể bị mất
       3. Không bị mất thứ tự gói tin
       4. Không bị trùng gói tin
    3. Truyền dữ liệu đơn công
  2. Giải quyết các vấn đề này như nào ?

Giao thức 3a và sự mất mát segement

- Việc mất mát segment ảnh hưởng thế nào đến giao thức 3a ?



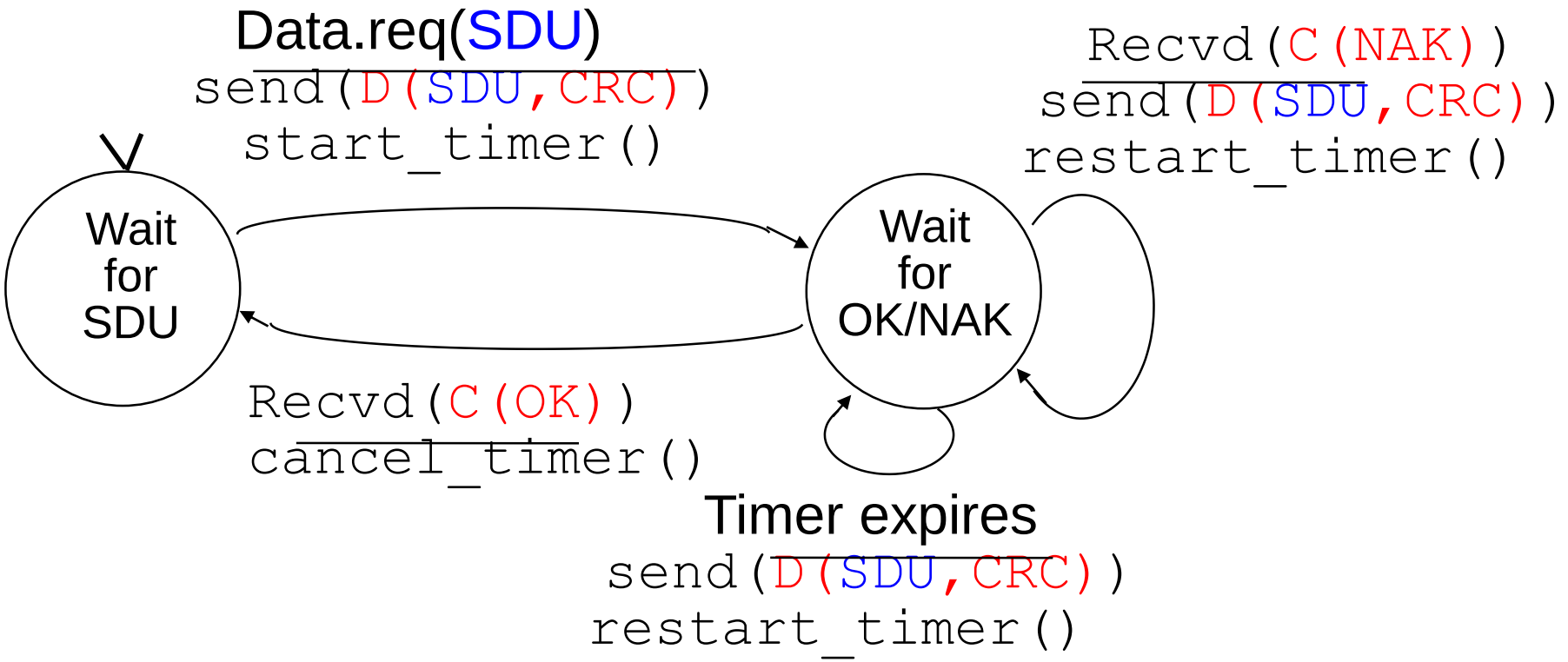
DEADLOCK

A đang đợi gói tin điều Khiển

B đang đợi gói tin dữ liệu

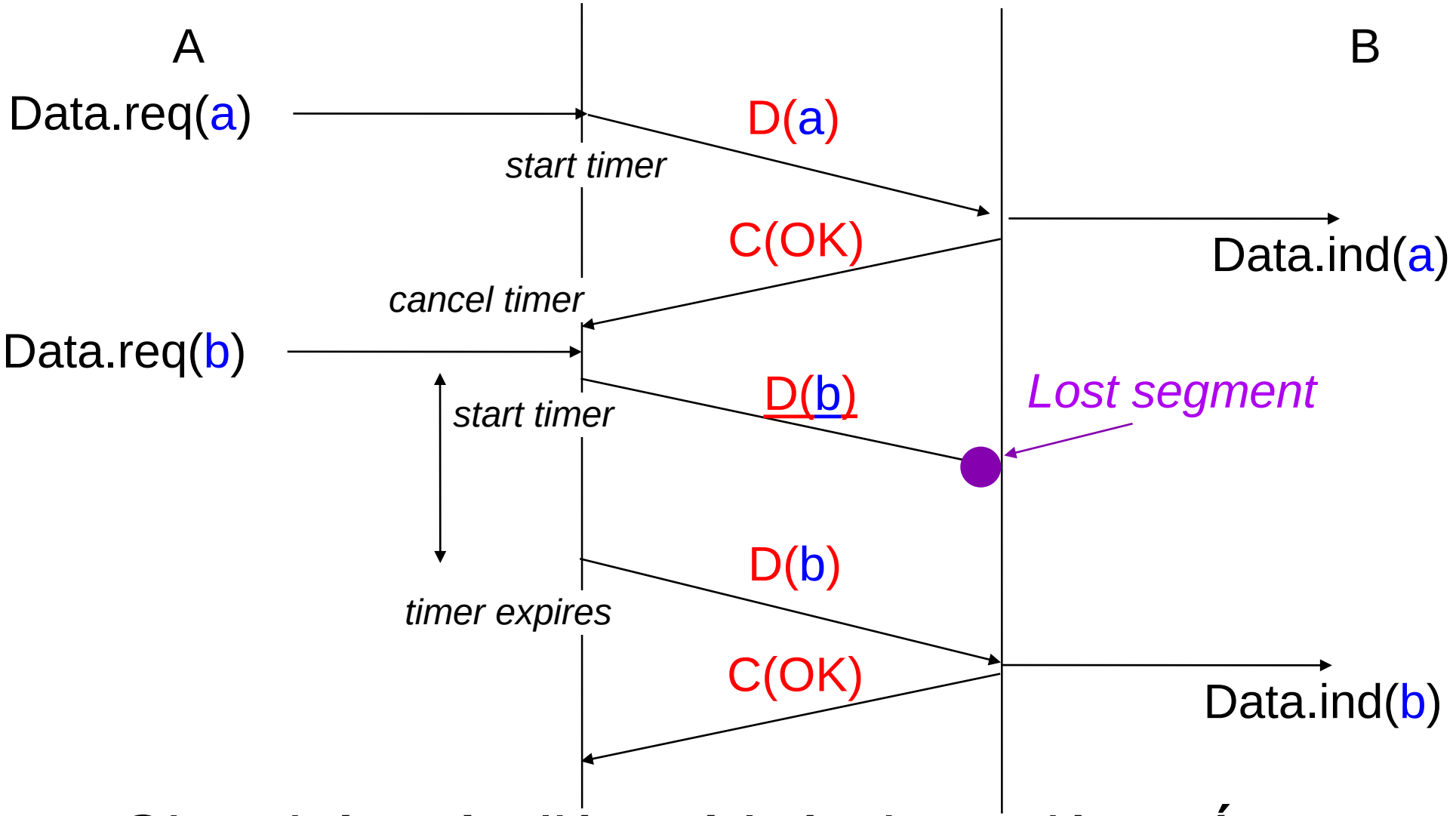
Giao thức 3b

- Thay đổi phía bên gửi
  - Thêm một đồng hồ truyền lại để truyền lại segment bị mất trước đó



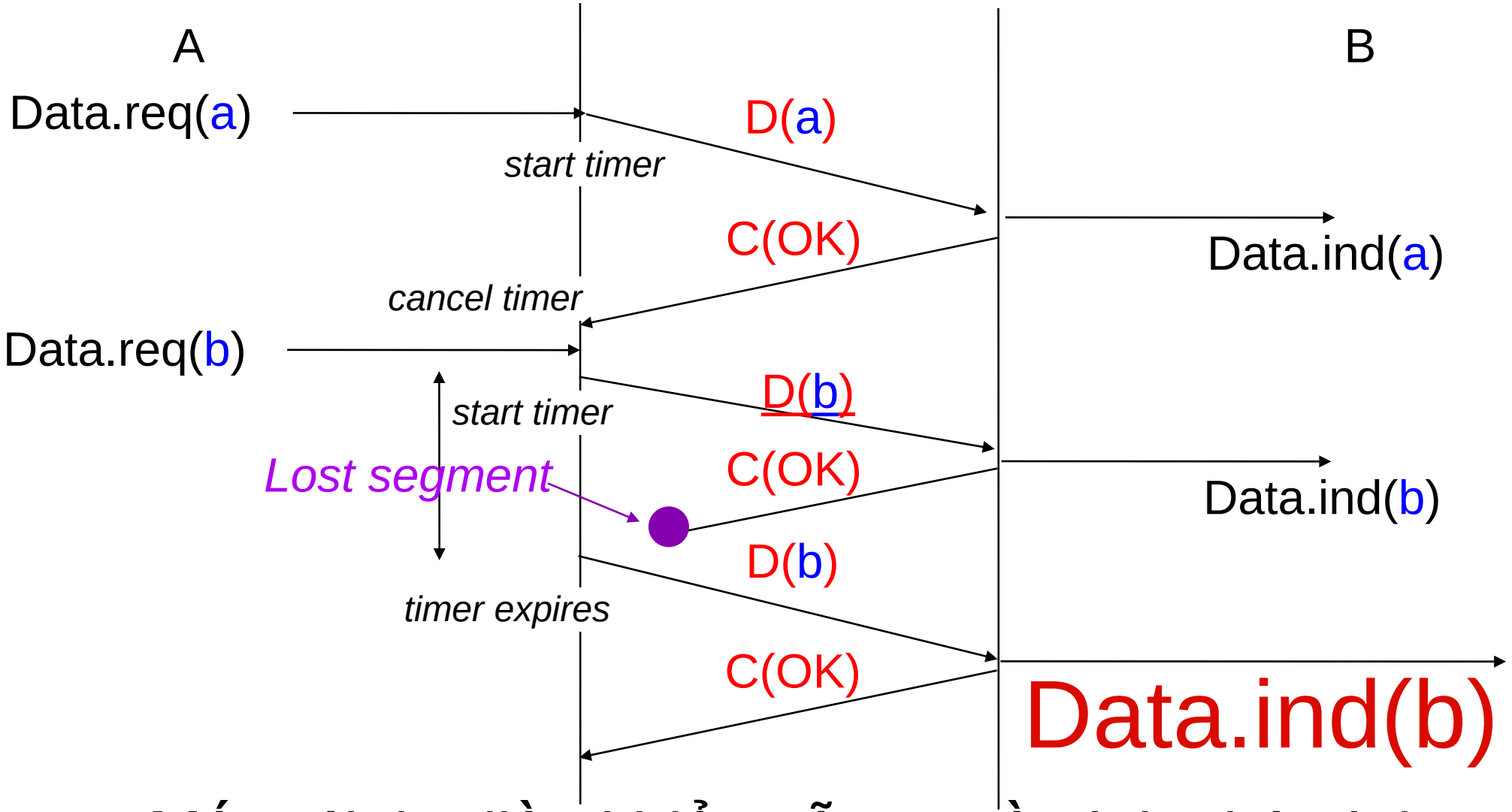
- Không thay đổi phía bên nhận

Giao thức 3b : Vi dụ



- Giao thức này liệu có luôn hoạt động tốt ?

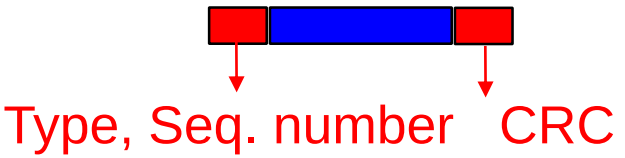
Giao thức 3b : Ví dụ

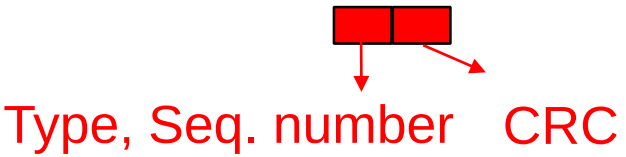


- Mất gói tin điều khiển vẫn truyền lại, nhận lại gói tin cũ. Làm sao giải quyết ?

Giao thức bít thay thế (alternating bit protocol)

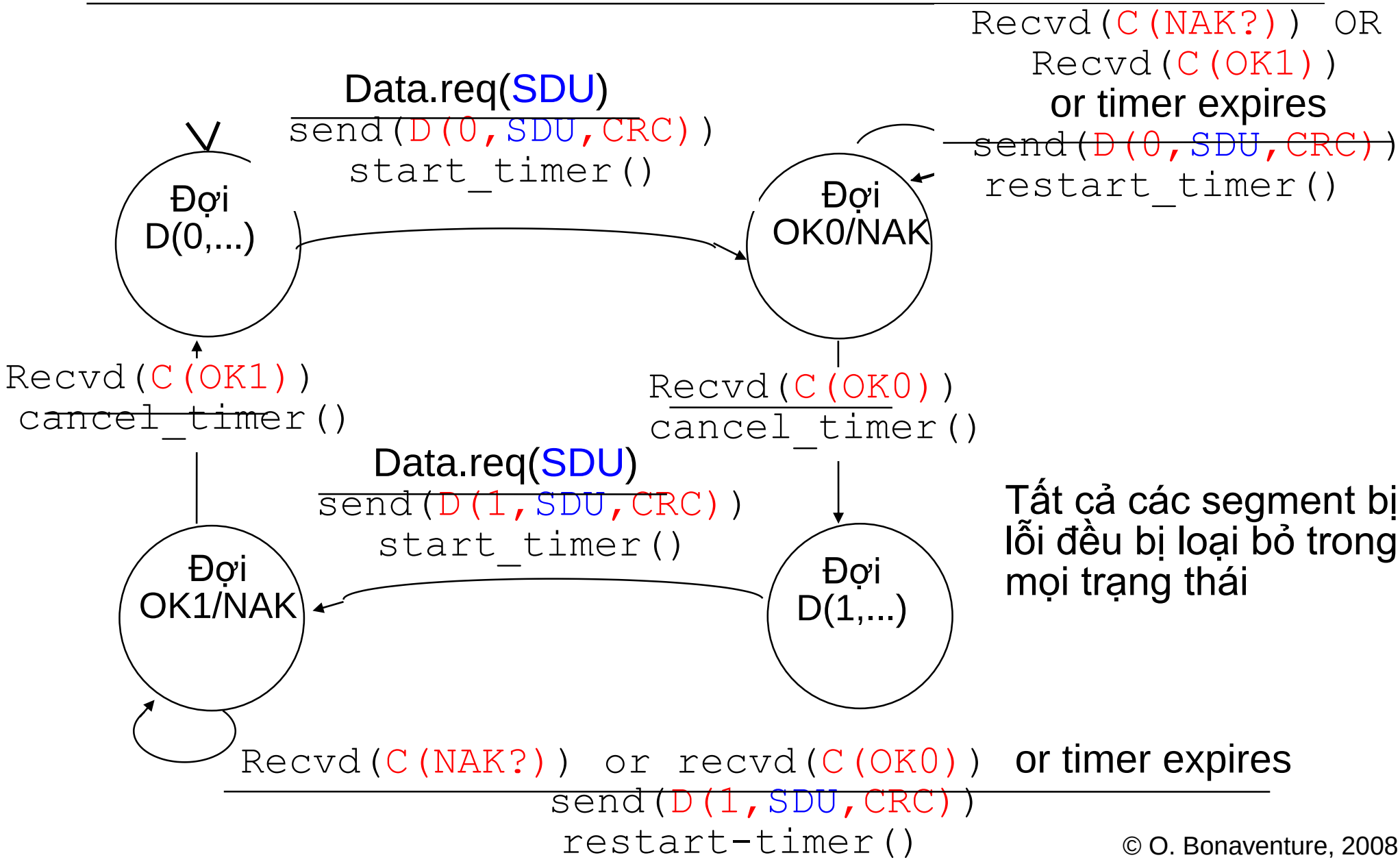
- Nguyên lý của giải pháp
  - Thêm số thứ tự vào mỗi segment dữ liệu được gửi từ bên gửi
  - Bằng cach gửi số thứ tự, bên nhận có thể kiểm định xem nó đã nhận được gói tin ấy chưa
- Nội dung của mỗi segment
  - segment dữ liệu
  - segment điều khiển





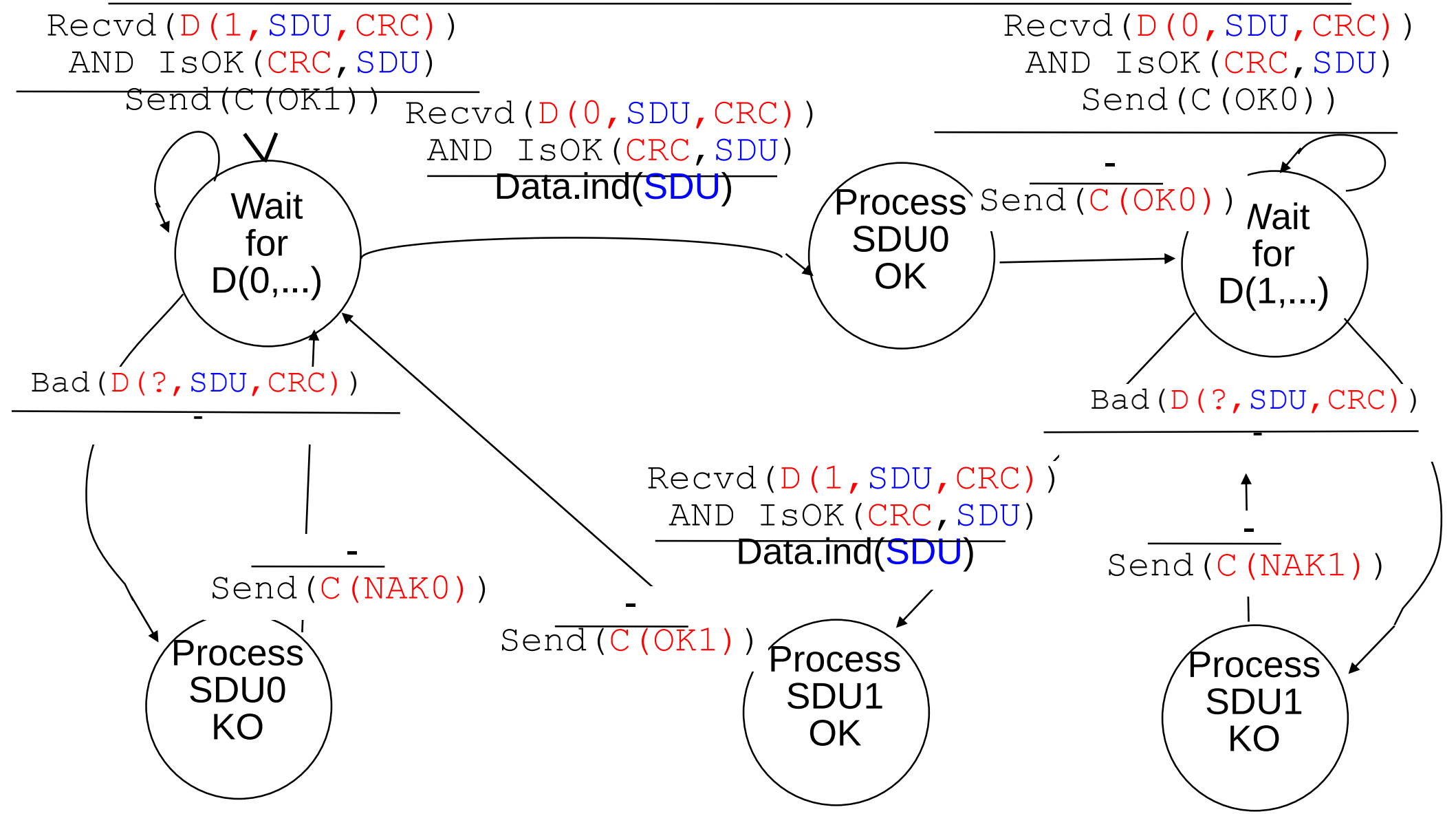
  - Chúng ta cần bao nhiêu bit cho số thứ tự?
    - Một bít là đủ

Giao thức bít thay thế
Bên gửi



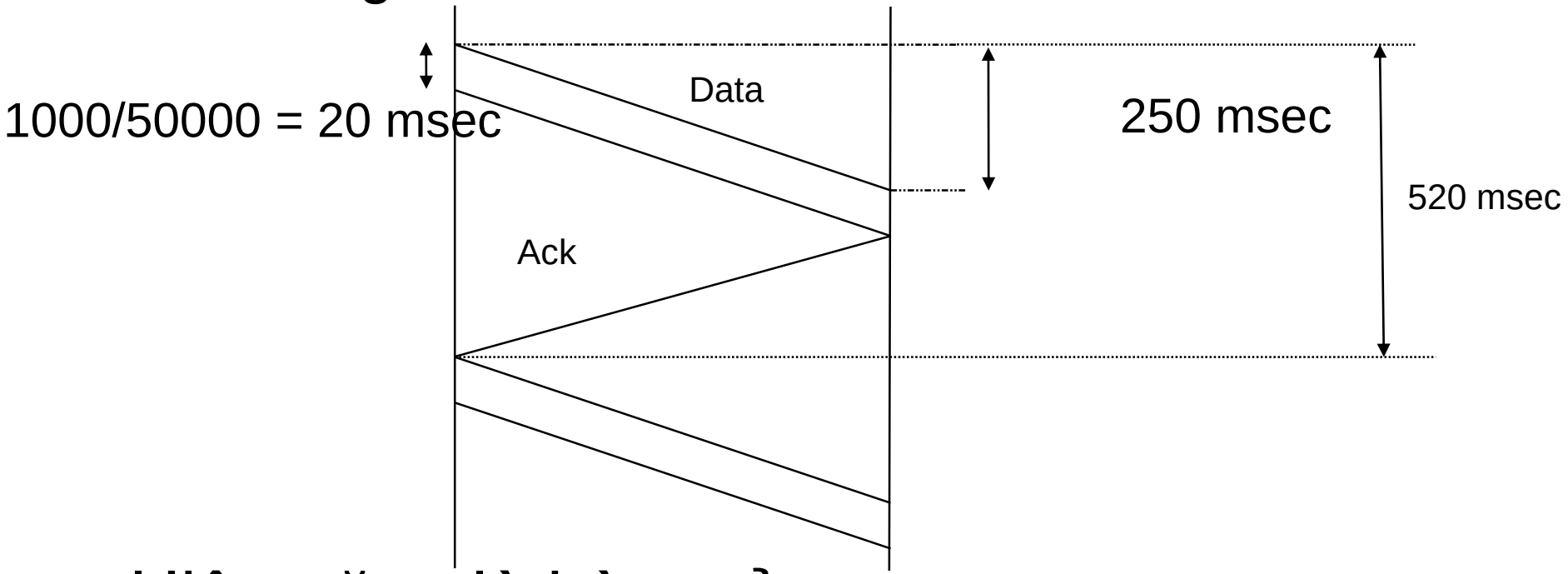
Tất cả các segment bị lỗi đều bị loại bỏ trong mọi trạng thái

Giao thức bit thay thế
Bên nhận



Hiệu năng của giao thức bit thay thế (ABP)

- Hiệu năng ABP như thế nào trong trường hợp này
  - Độ trễ (một chiều) : 250 msec
  - Băng thông tầng vật lý : 50 kbps
  - Kích cỡ segment : 1000 bits

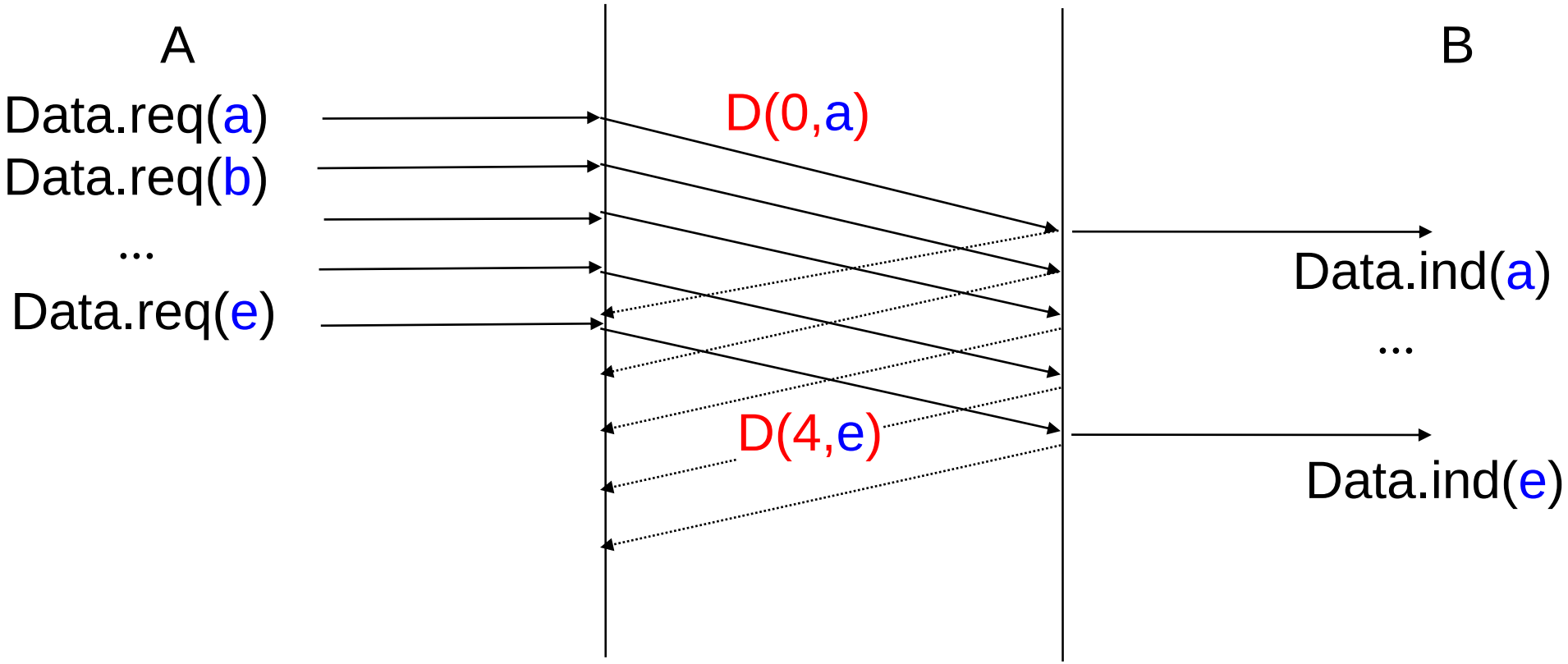


- -> Hiệu năng là hàm của

*bandwidth * round-trip-time*

Làm sao để cải thiện hiệu năng của ABP ?

- Sử dụng pipeline
- Nguyên lý
  - Bên gửi nên cho phép gửi nhiều hơn một segment khi đợi xác nhận từ bên nhận



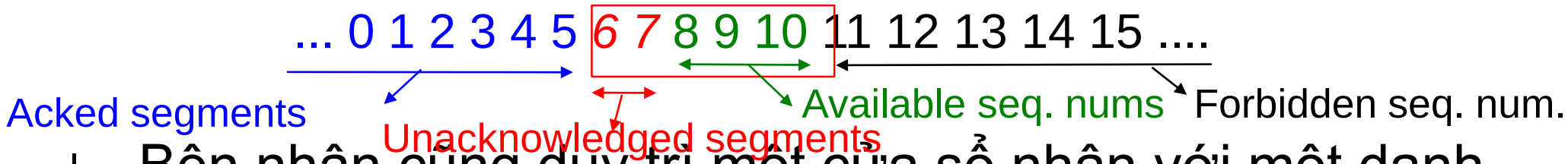
Làm sao để cải tiến hiệu năng của ABP ? (2)

- Thay đổi ABP
  - Số thứ tự trong mỗi segment
    - Mỗi segment dữ liệu chứa số thứ tự của riêng nó
    - Mỗi segment điều khiển chỉ ra số thứ tự của segment dữ liệu mà nó xác nhận (OK/NAK)
  - Bên gửi
    - Cần đủ bộ nhớ đệm để lưu trữ segment dữ liệu mà chưa được xác nhận để có thể truyền lại nếu được yêu cầu
  - Bên nhận
    - Cần đủ bộ nhớ đệm để lưu trữ các segment bị sai số thứ tự

Làm sao để tránh tràn bộ nhớ đệm bên nhận?

Cửa sổ trượt

- Nguyên lý
  - Bên gửi giữ một danh sách các segment được phép gửi
    - `sending_window`

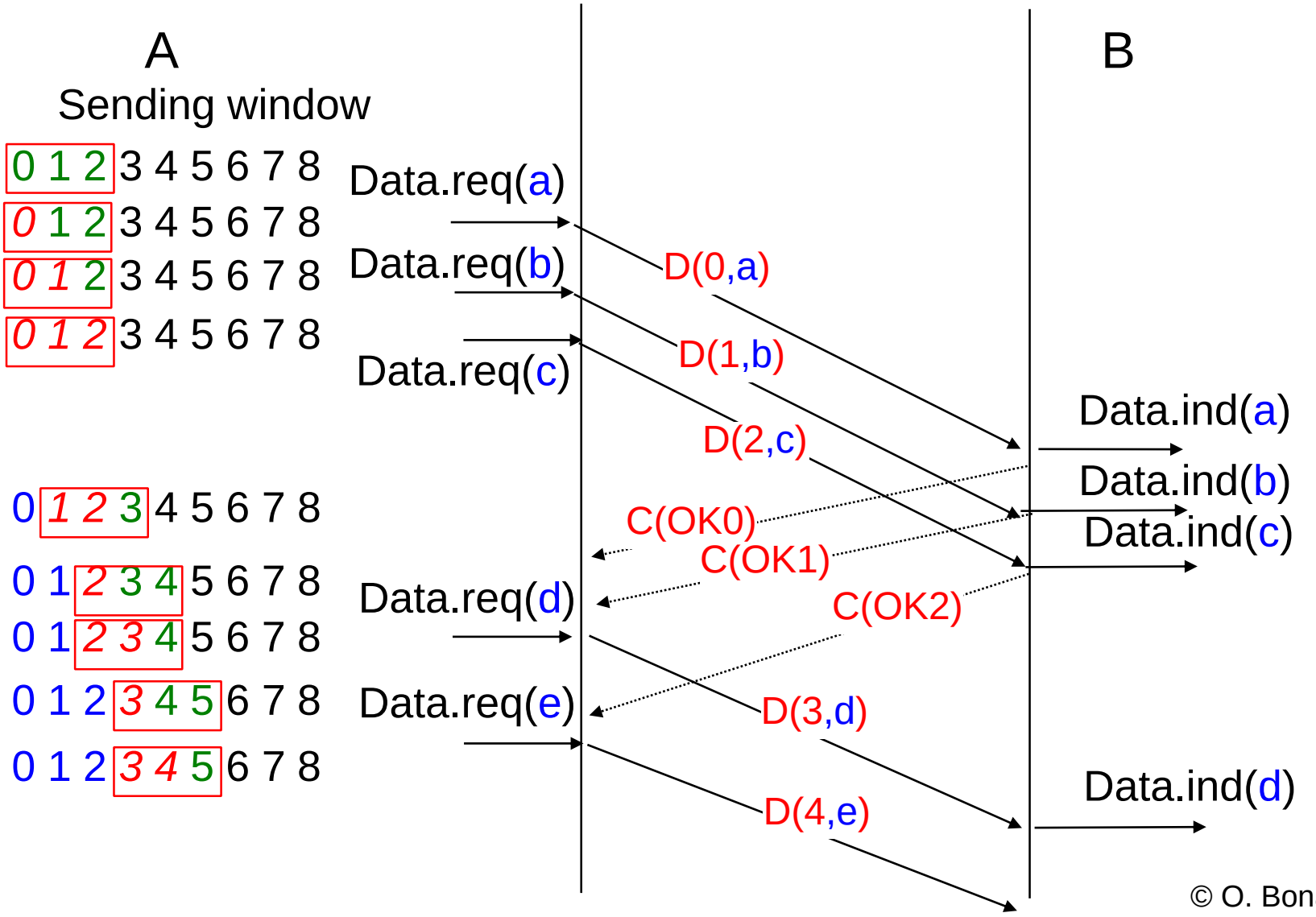


  - Bên nhận cũng duy trì một cửa sổ nhận với một danh sách các số thứ tự có thể được chấp nhận
    - receiving_window

  - Bên gửi và bên nhận phải sử dụng các của sổ tương thích nhau
    - sending_window δ receiving window
      - Chẳng hạn kích cỡ cửa sổ là hằng số với một giao thức cho trước hoặc được thương lượng giữa các bên trong pha thiết lập kết nối

Cửa sổ trượt : ví dụ

- Cửa sổ gửi và nhận : 3 segments

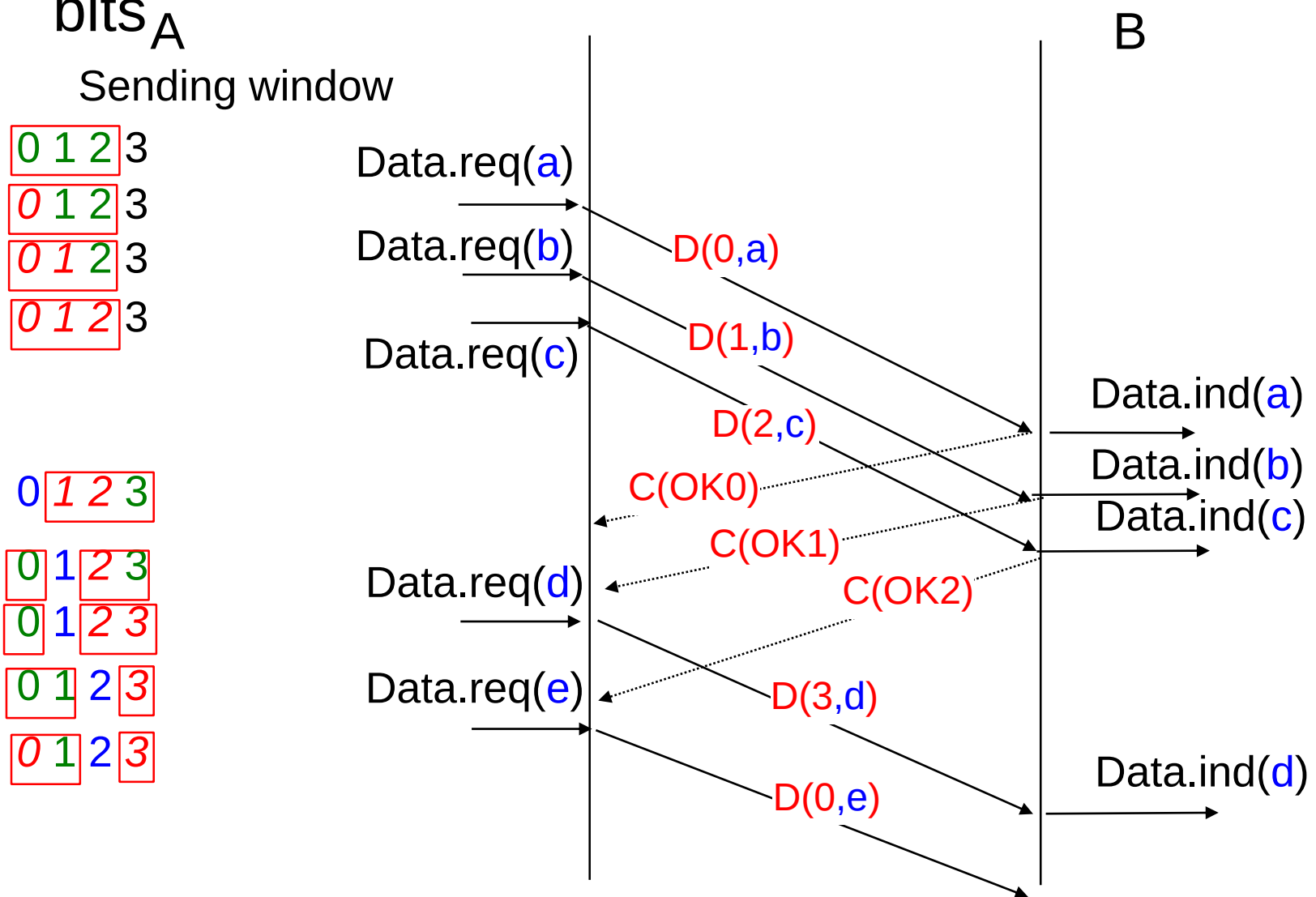


Mã hóa số thứ tự

- Vấn đề
  - Chúng ta phải sử dụng bao nhiêu bít trong phần tiêu đề của
    - N bits nghĩa là mã hóa được 2^N số thứ tự khác nhau
- Giải pháp
  - Đặt trong mỗi segment được truyền đi một số thứ tự là modulo của 2^N
  - Cùng một số thứ tự có thể được dung cho nhiều segment khác nhau
    - Tuy nhiên, điều này có thể gây ra các vấn đề khác...
  - Cửa sổ trượt
    - Là danh sách các số thứ tự liên tiếp (là modulo 2^N) mà bên gửi được phép truyền

Cửa sổ trượt : ví dụ tiếp theo

- Cửa sổ gửi và cửa sổ nhận 3 segments
  - Các số thứ tự được mã hóa bằng trường thông tin 2 bits



Truyền tin tin cậy với cửa sổ trượt

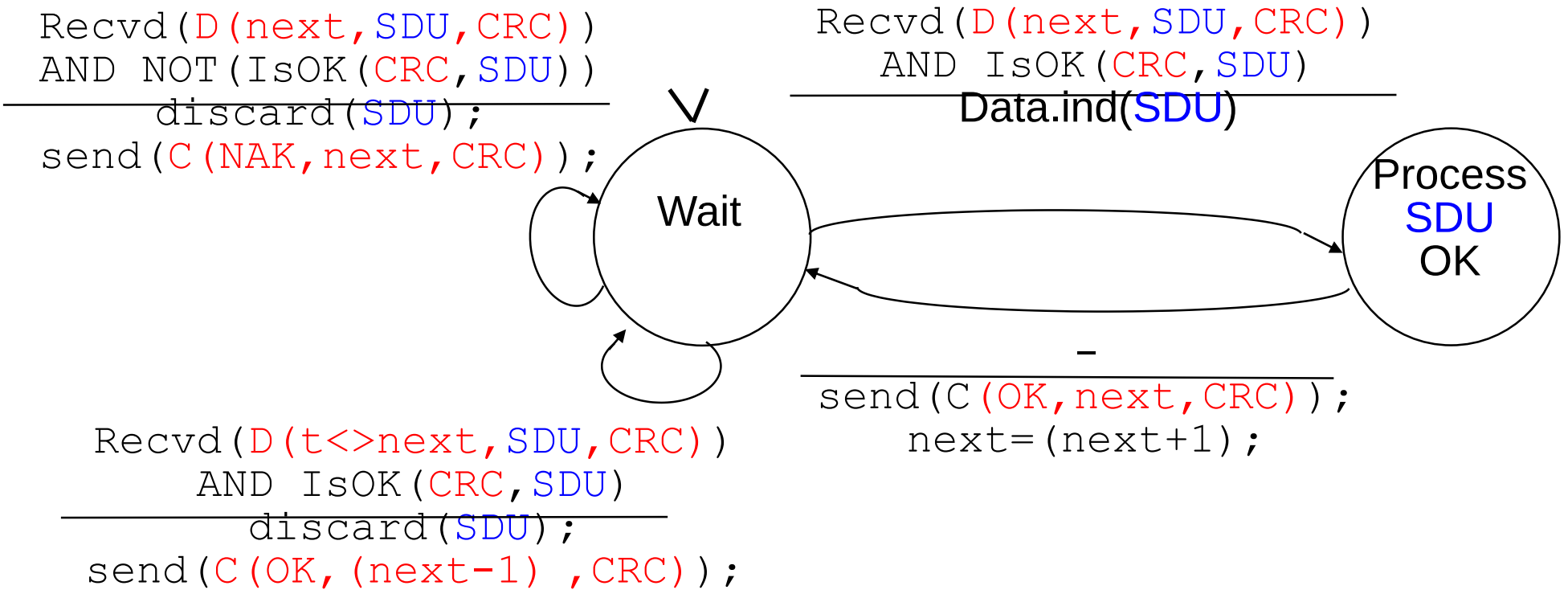
- Làm sao để truyền tin tin cậy với một cửa sổ trượt
  - Phản ứng với việc nhận gói tin điều khiển như nào?
  - Cách ứng xử của bên nhận và bên gửi
- Các giải pháp cơ bản
  - Go-Back-N
    - Cài đặt đơn giản nhất là phía nhận
    - thông lượng bị hạn chế khi có mất mát gói tin
  - Selective Repeat
    - Cài đặt khó hơn
    - Thông lượng vẫn có thể cao kkhi xảy ra mất mát gói tin

GO-BACK-N

- Nguyên lý
  - Phía nhận phải càng đơn giản càng tốt
  - Phía bên nhận
    - Chỉ chấp nhận các segment dữ liệu có số thứ tự liên tiếp
    - Giải nghĩa các segment điều khiển
      - Khi nhận được segment điều khiển
        - OKX nghĩa là tất cả các segment dữ liệu, có số thứ tự và bao gồm cả X đã được nhận chính xác
        - NAKX nghĩa là segment dữ liệu có số thứ tự X có lỗi hoặc bị mất
  - Bên gửi
    - Dựa trên đồng hồ truyền lại (retransmission timer) để phát hiện segment bị mất
    - Khi hết thời gian truyền lại hoặc nhận được NAK : truyền lại tất cả các segment dữ liệu chưa được xác nhận
      - Như vậy, bên gửi sẽ truyền lại một segment đã nhận được chính xác nhưng không đúng thứ tự ở phía nhận

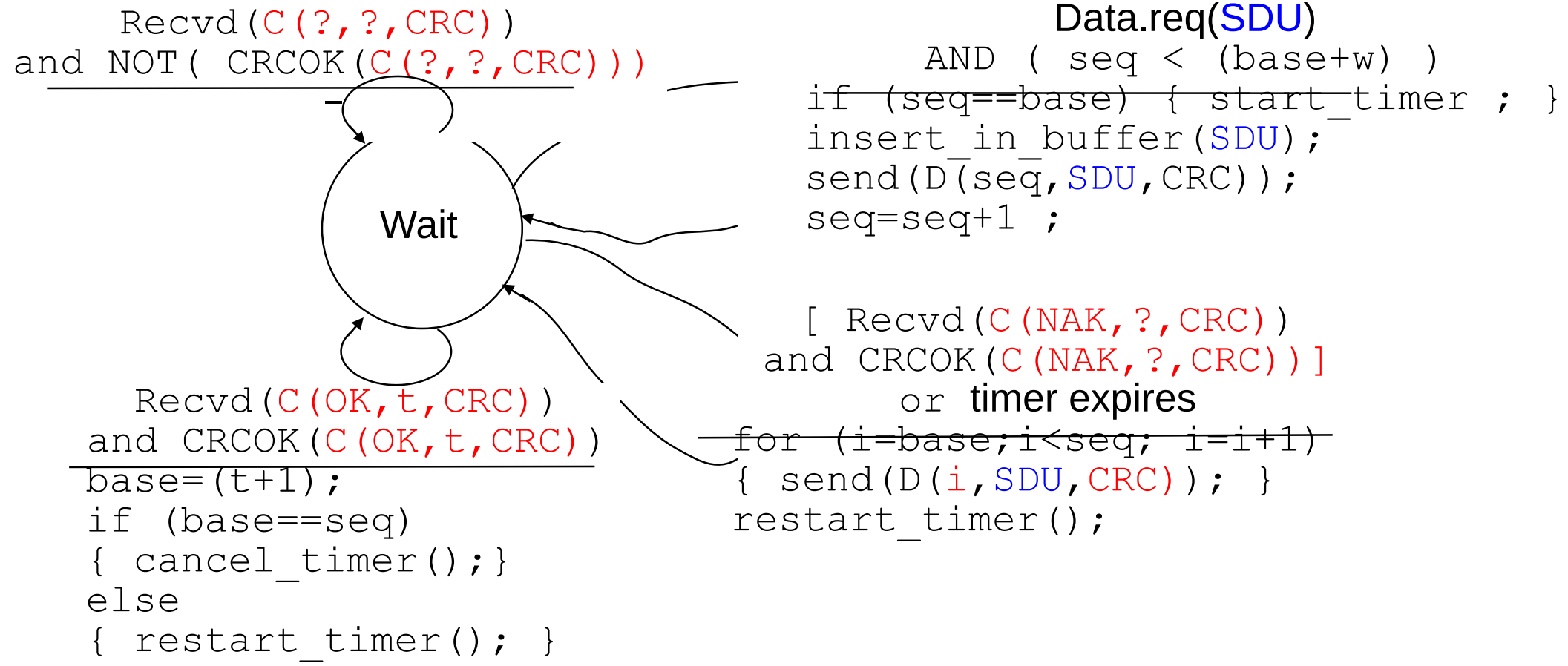
Go-Back-N : Bên nhận

- Biến trạng thái
  - `next` : là số thứ tự của segment dữ liệu mong muốn nhận

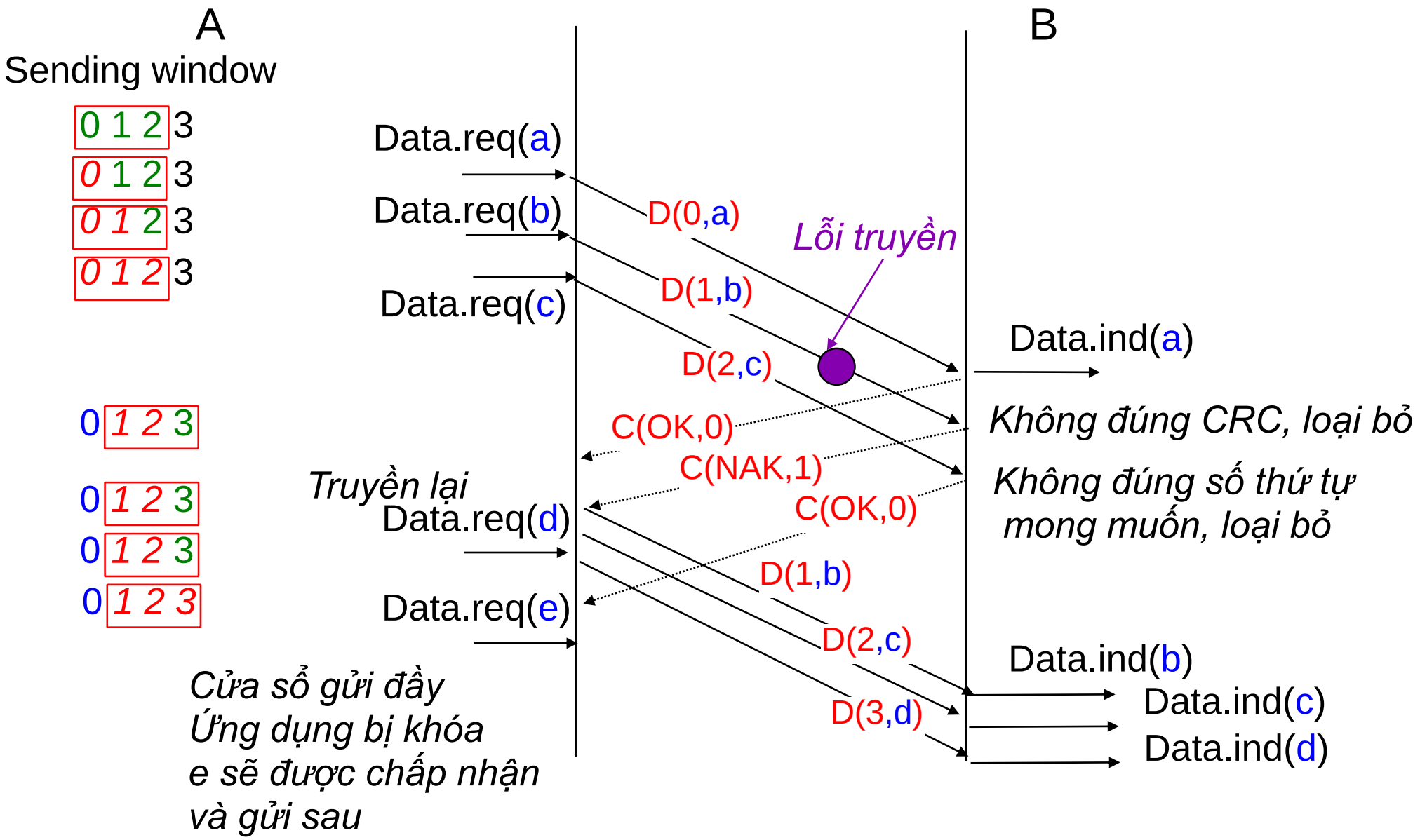


Go-Back-N : Bên gửi

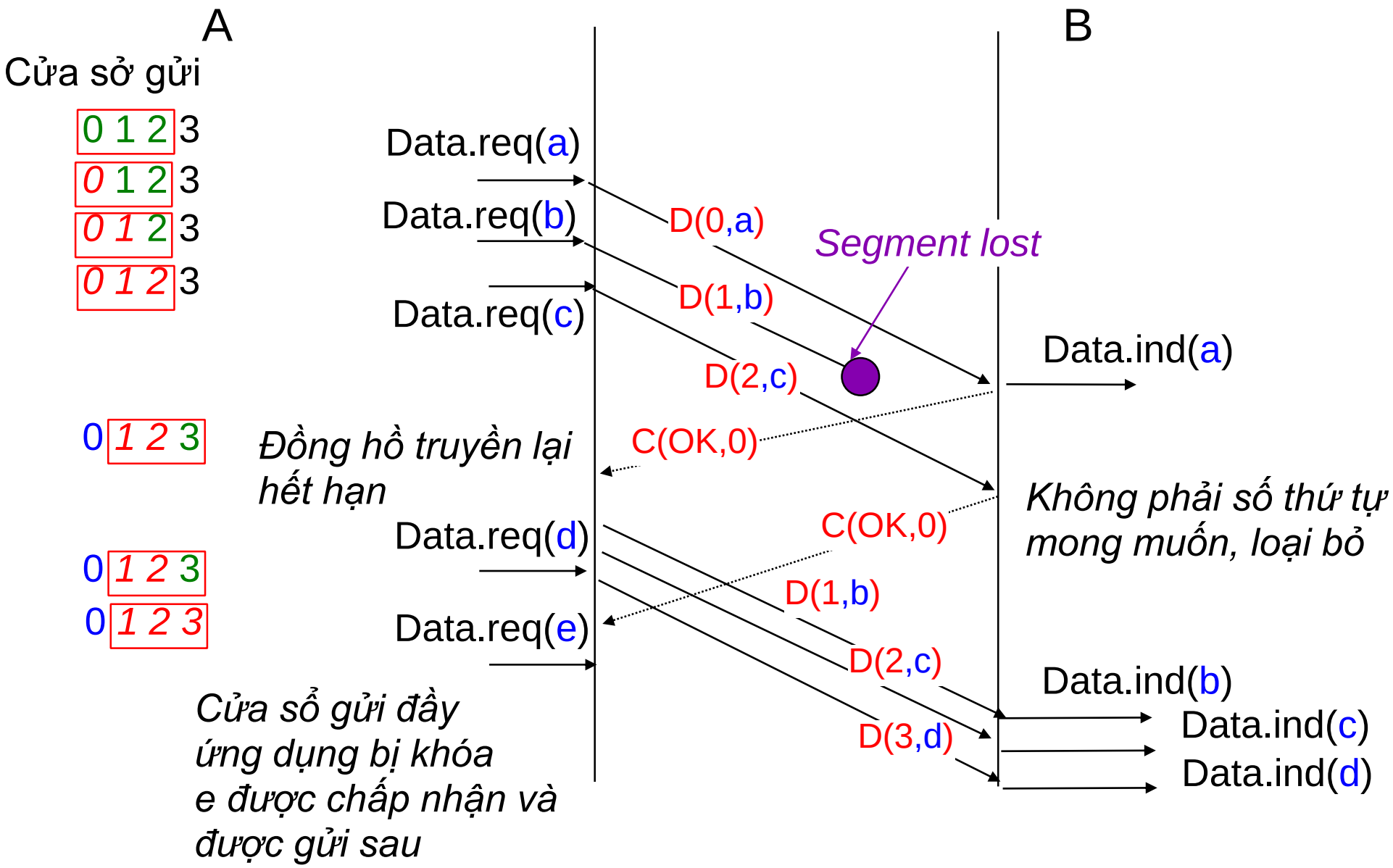
- Các biến trạng thái
  - `base` : số thứ tự của segment dữ liệu đã gửi đi lâu nhất
  - `seq` : số thứ tự sẵn có đầu tiên
  - `W` : kích cỡ cửa sổ trượt



Go-Back-N : Ví dụ

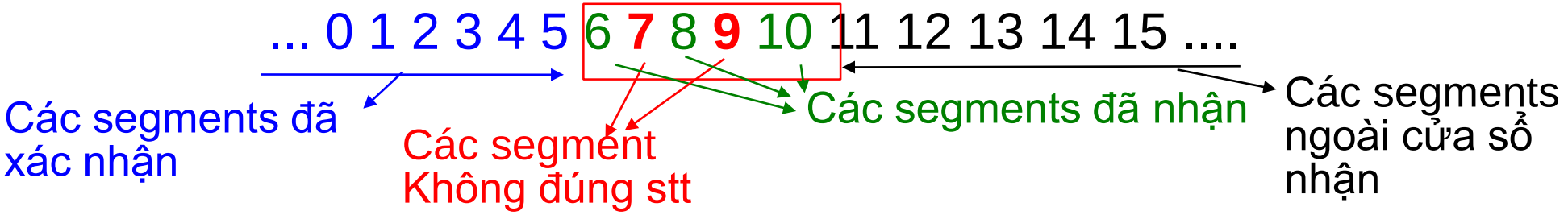


Go-Back-N : Ví dụ (2)



Selective Repeat

- Bên nhận
  - Sử dụng bộ đệm để lưu các segement đã nhận nhưng không đúng số thứ tự và sắp xếp lại chúng
  - Có cửa sổ nhận



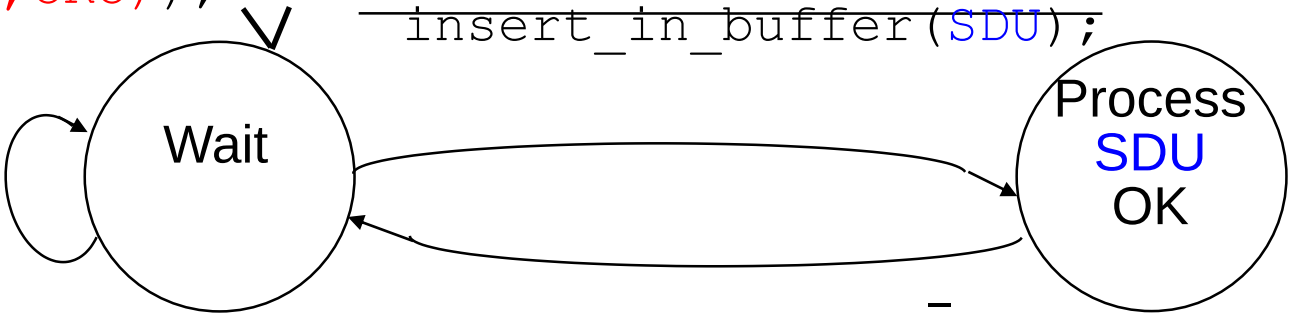
  - Giải nghĩa các segment điều khiển
    - OKX
      - Các segments có thứ tự đến X và gồm cả X đã được nhận chính xác
    - NAKX
      - Segment có số thứ tự X bị lỗi
- Bên gửi
  - Khi phát hiện segment bị mất hay bị lỗi, bên gửi chỉ gửi lại segment đó
    - Có thể cần sử dụng một đồng hồ truyền lại cho một segment

Selective-Repeat : Bên nhận

- Các biến trạng thái
  - `next` : stt của segment dữ liệu mong muốn nhận
  - `Last` : segment được nhận đúng số thứ tự cuối cùng

```
Recvd(D(t,SDU,CRC))
AND NOT(IsOK(CRC,SDU))
discard(SDU);
send(C(NAK,t,CRC));
```

```
Recvd(D(t,SDU,CRC))
AND IsOK(CRC,SDU)
insert_in_buffer(SDU);
```

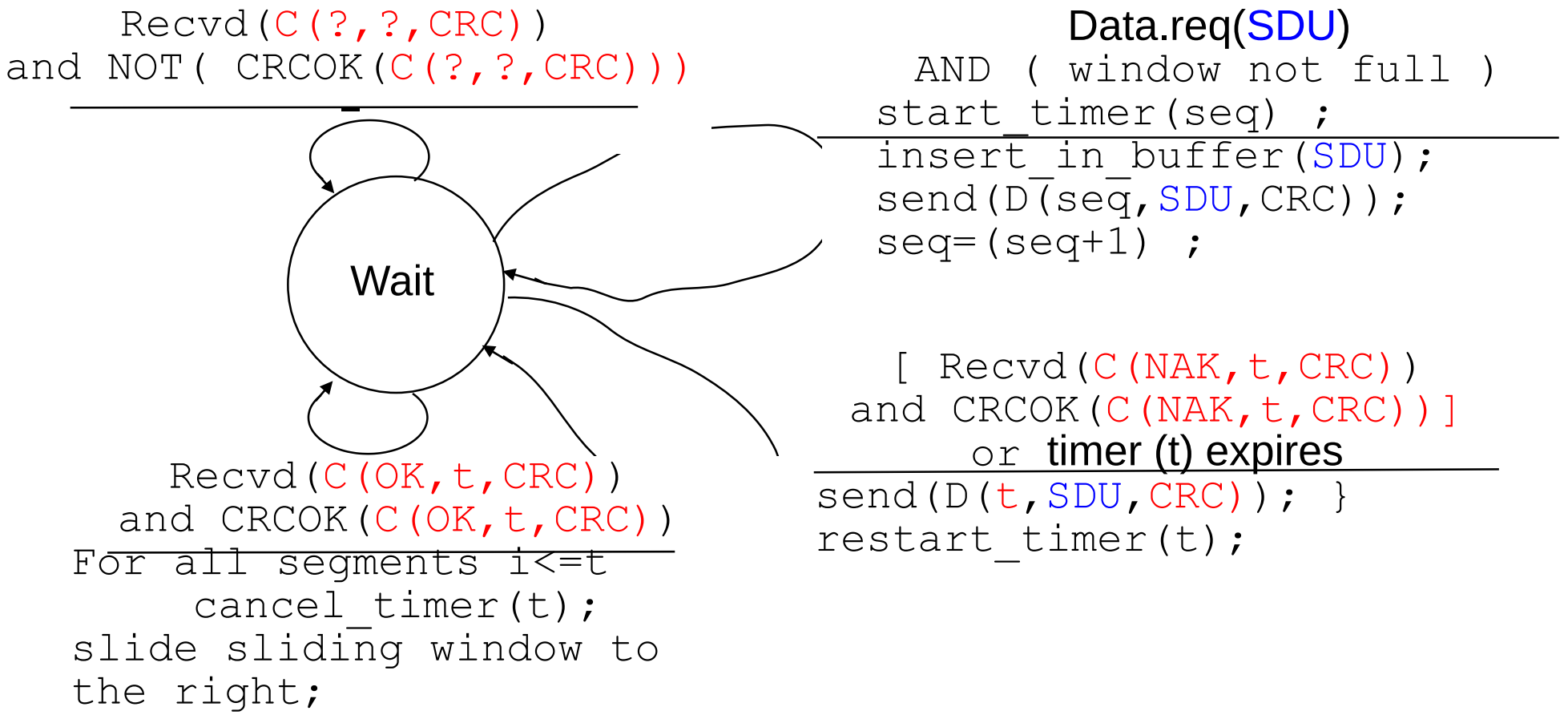


For all in sequence segments inside buffer
Data.ind(SDU);

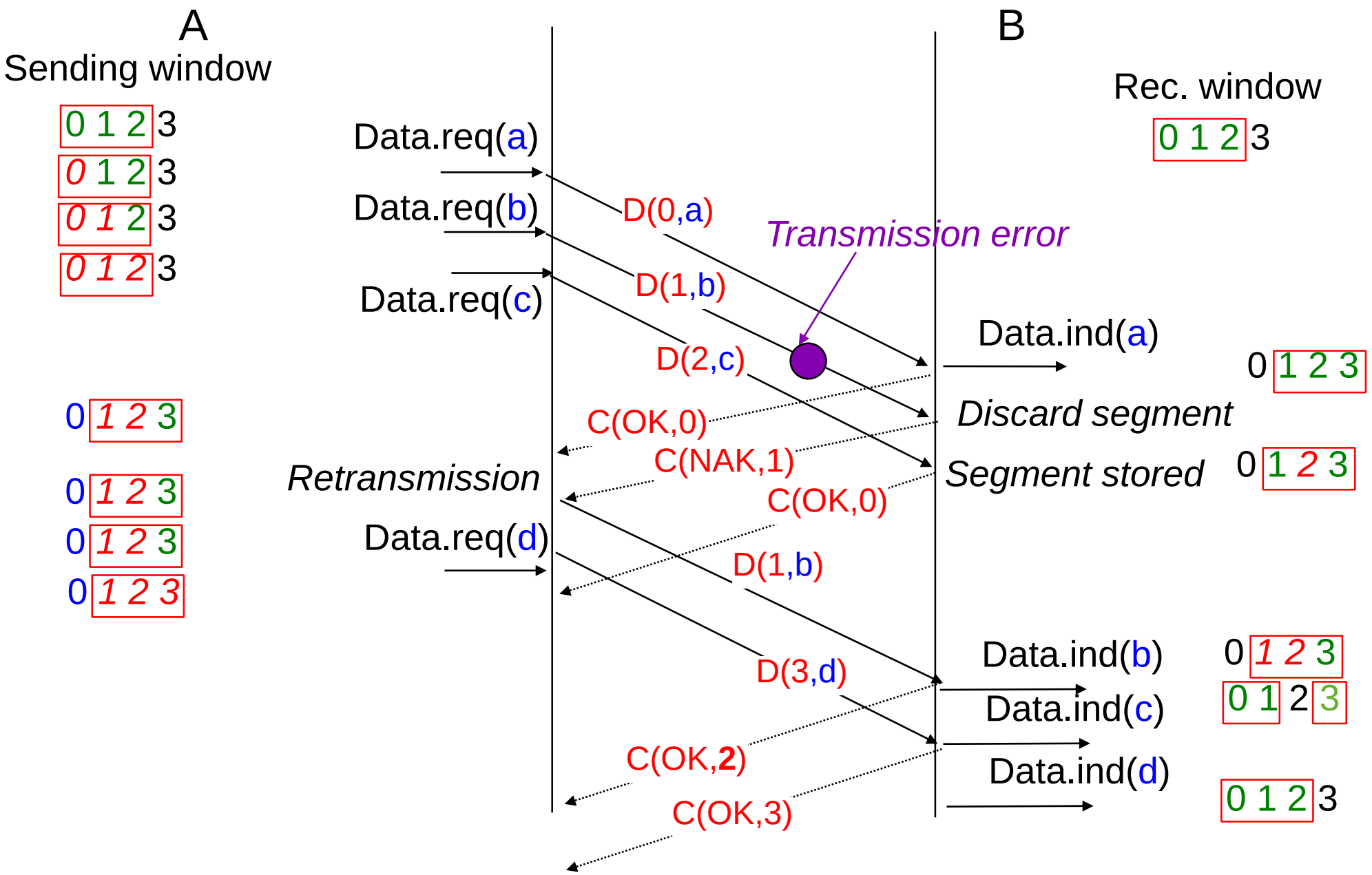
```
slide the sliding window;
update next and last
send(C(OK,(next-1)));
```

Selective Repeat : Bên gửi

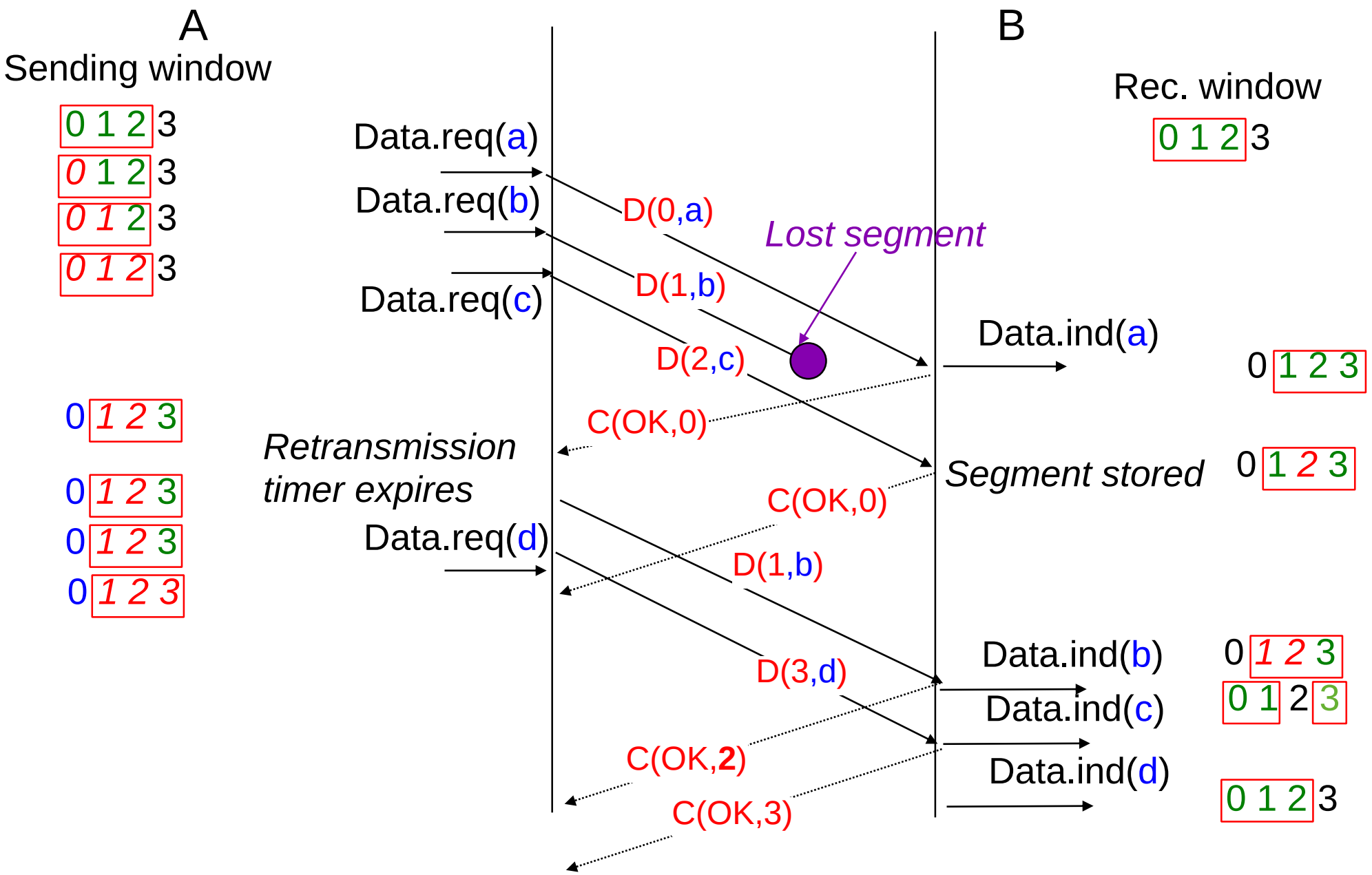
- Các biến trạng thái
  - `base` : số thứ tự của segment đã gửi đi chưa được xác nhận lâu nhất
  - `seq` : số thứ tự sẵn có đầu tiên
  - `W` : kích cỡ cửa sổ



Selective Repeat : Ví dụ

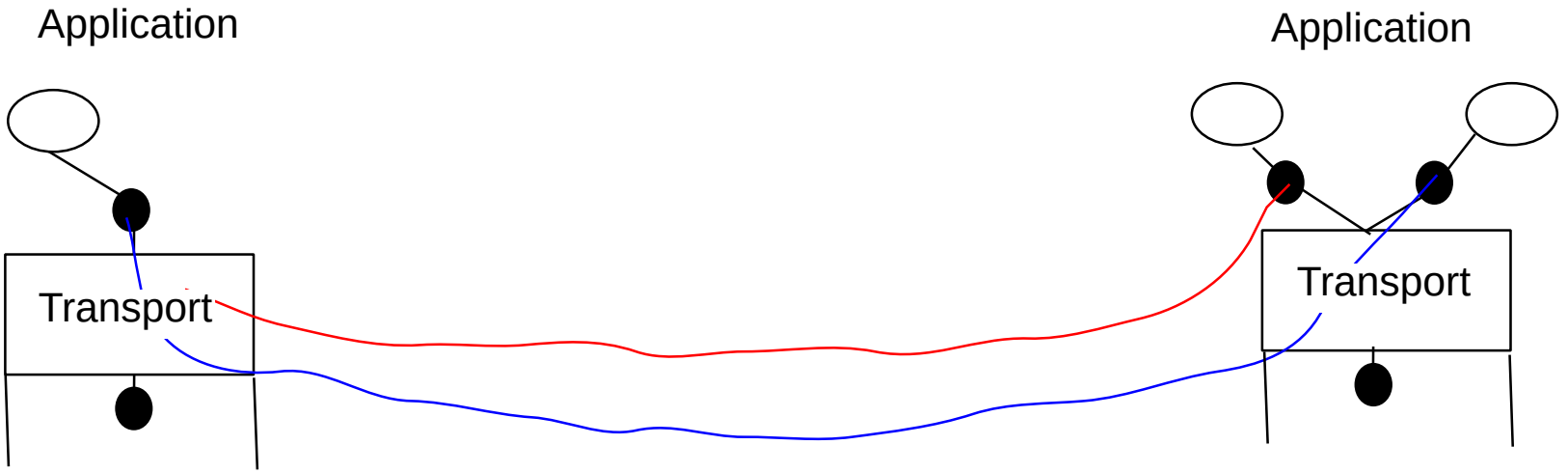


Selective Repeat : Ví dụ (2)



Quản lý bộ nhớ đệm

- Đặt vấn đề



- Thực thể tầng giao vận có thể phải phục vụ nhiều kết nối đồng thời
  - Làm sao chia sẻ bộ nhớ đệm cho nhiều kết nối?
  - Số lượng kết nối lại thay đổi theo thời gian
  - Một số kết nối yêu cầu nhiều bộ nhớ đệm trong khi số khác lại yêu cầu ít hơn
    - `ftp` versus `telnet`

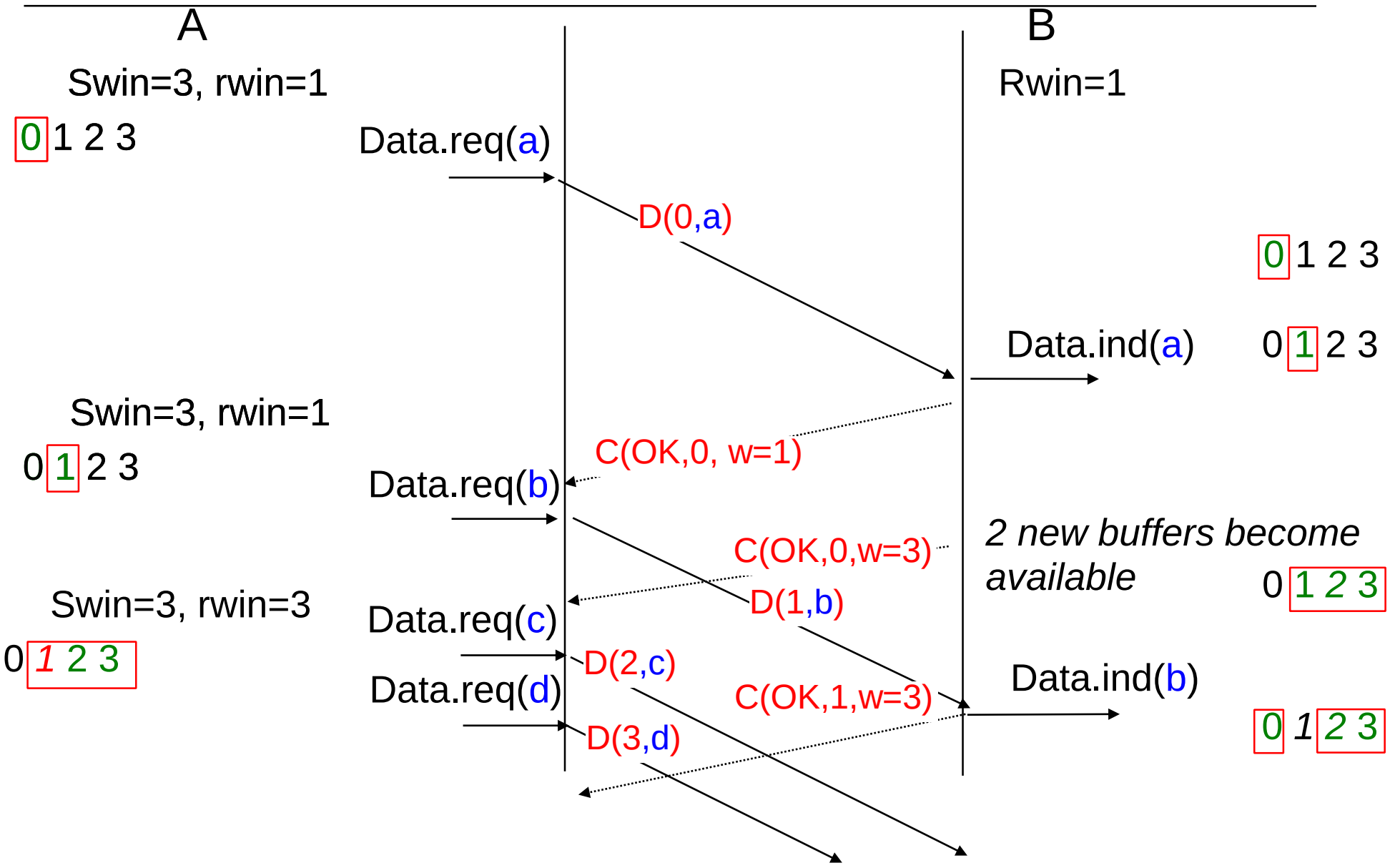
Quản lý bộ nhớ đệm (2)

- Nguyên lý
  - Điều chỉnh kích cỡ của cửa sổ nhận theo dung lượng bộ nhớ đệm sẵn có bên nhận
  - Cho phép bên nhận quảng bá kích cỡ cửa sổ nhận đến phía gửi

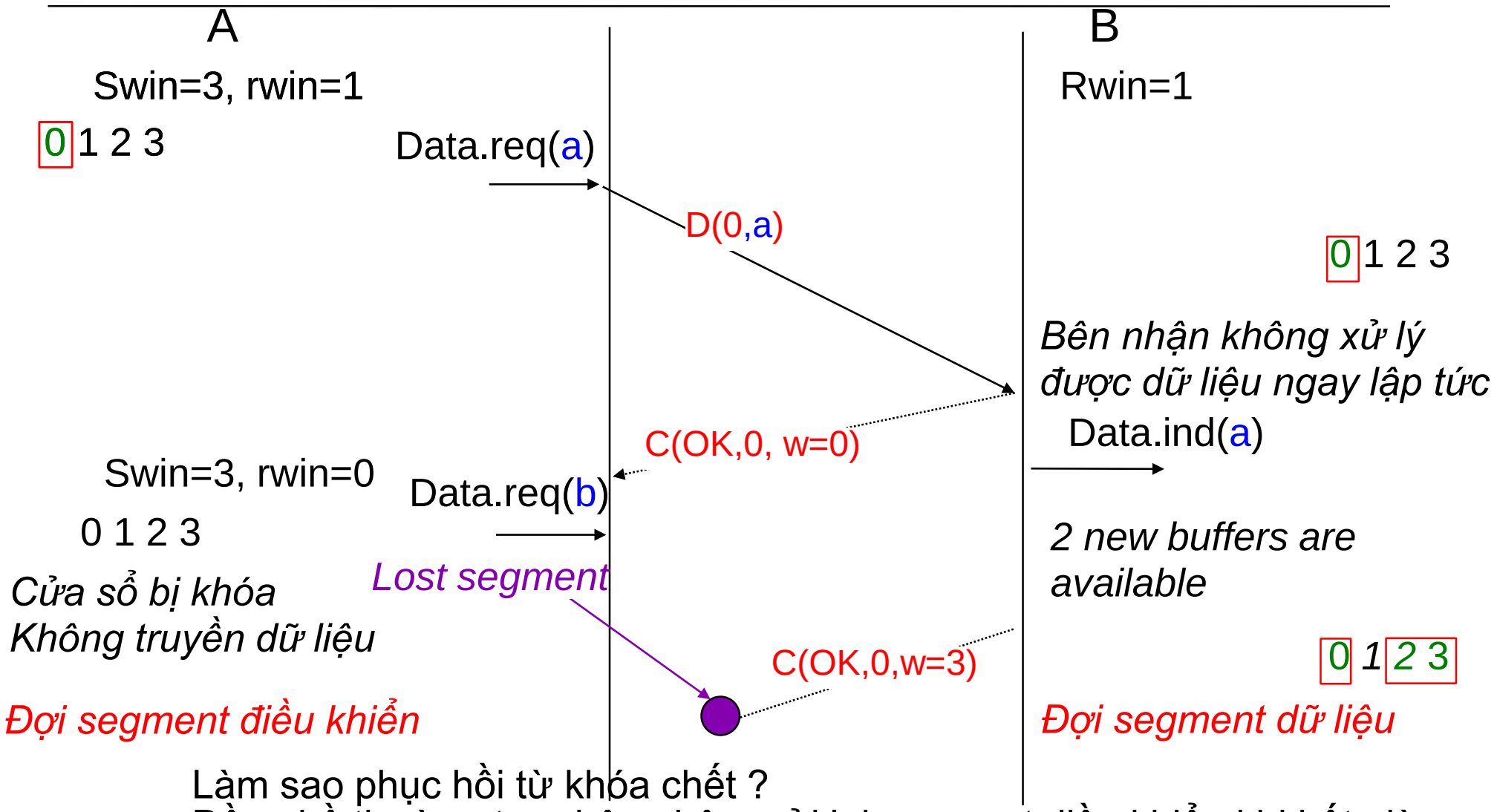
  - Thông tin mới them vào segments điều khiển
    - `win` chỉ kích cỡ cửa sổ nhận hiện tại

  - Thay đổi bên gửi
    - Cửa sổ gửi : `swin` (là hàm của bộ nhớ sẵn có)
    - Lưu trữ trạng thái của của sổ nhận nhận được từ phía nhận bằng biến : `rwin`
    - Tại bất kì thời điểm nào, bên gửi chỉ được cho phép gửi các segment dữ liệu có số thứ tự nhỏ hơn
      min(`rwin`, `swin`)

Quản lý bộ nhớ đệm (3)



Quản lý bộ nhớ đệm (4)



Làm sao phục hồi từ khóa chết ?
Đồng hồ thường trực bên nhận, gửi lại segment điều khiển khi hết giờ

Trùng gói tin và sai thứ tự

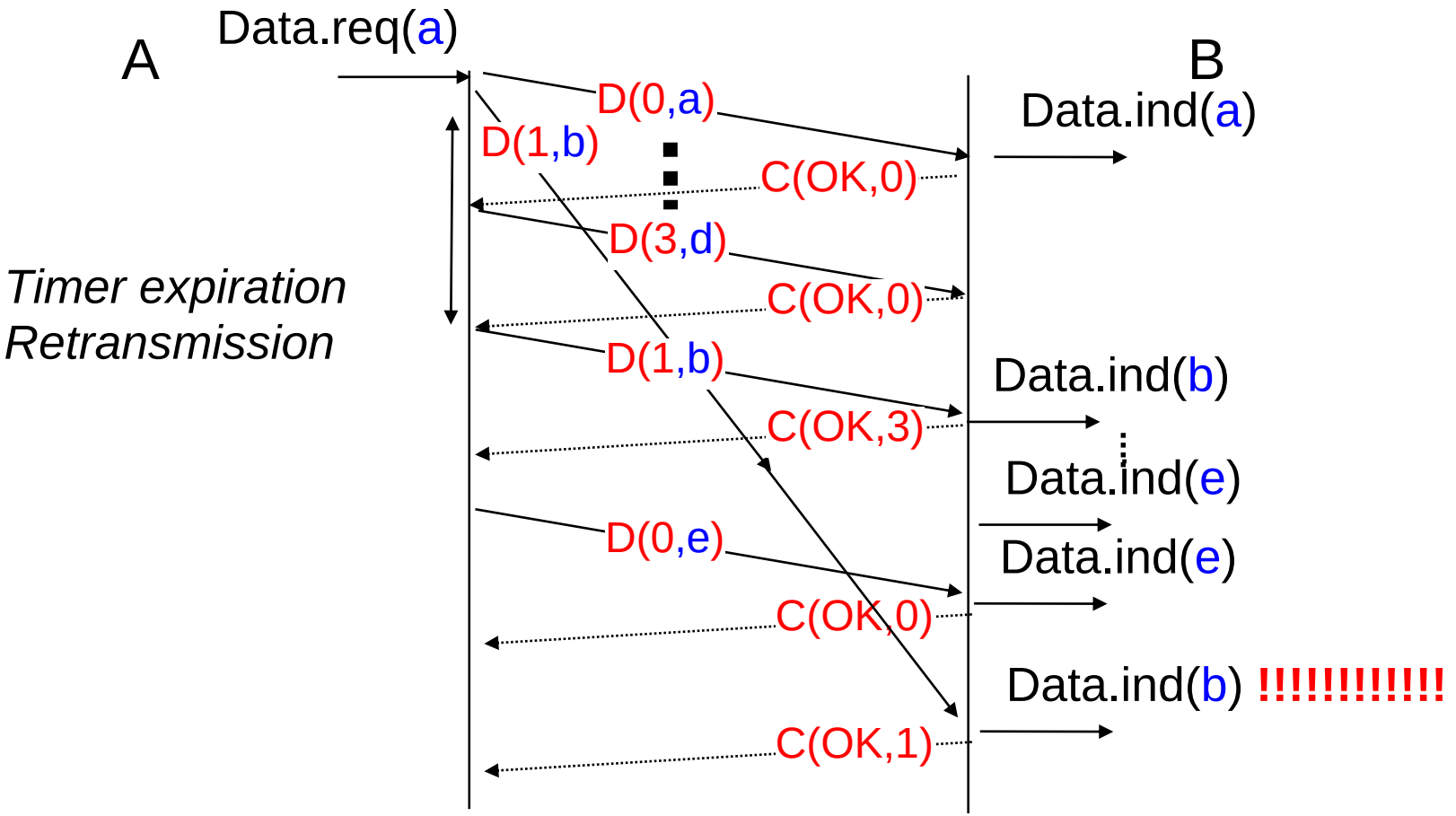
- Làm sao cung cấp dịch vụ tin cậy trên tầng giao vận?

- Giả thuyết
  1. Ứng dụng gửi các SDU nhó
  2. Tầng Mạng cung cấp dịch vụ hoàn hảo
     1. Nhưng vẫn có thể có lối truyền
     2. Vẫn có thể mất gói tin
     3. Vẫn có thể sai thứ tự
     4. Gói tin vẫn có thể trùng
  3. Truyên tin đơn công (1 chiều)

2. Làm sao giải quyết các vấn đề này ?

Trùng gói tin và sai thứ tự (2)

- Vấn đề
  - Segment đến muộn có thể bị nhầm với segment đã nhận đúng



Trùng gói tin và sai thứ tự (3)

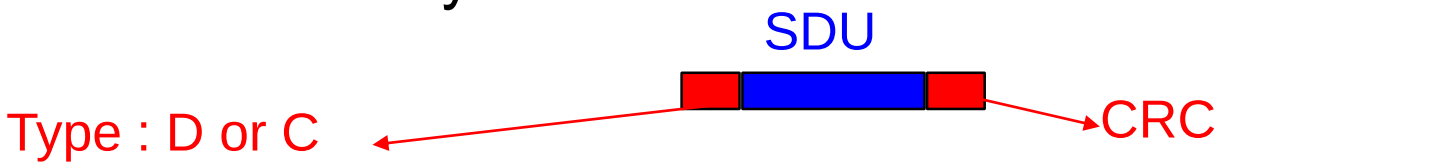
- Làm sao giải quyết trùng gói tin và sai thứ tự?
  - Segment không nên tồn tại vĩnh viễn trên mạng
  - Đặt ràng buộc trên tầng mạng
    - Gói tin không thể tồn tại trong mạng lâu hơn MSL (maximum segment life time) giây (2 phut trong RFC 793)
- Nguyên lý của giải pháp
  - Segment với số thứ tự x chỉ có thể được truyền trong thời gian MSL seconds

Luồng song công

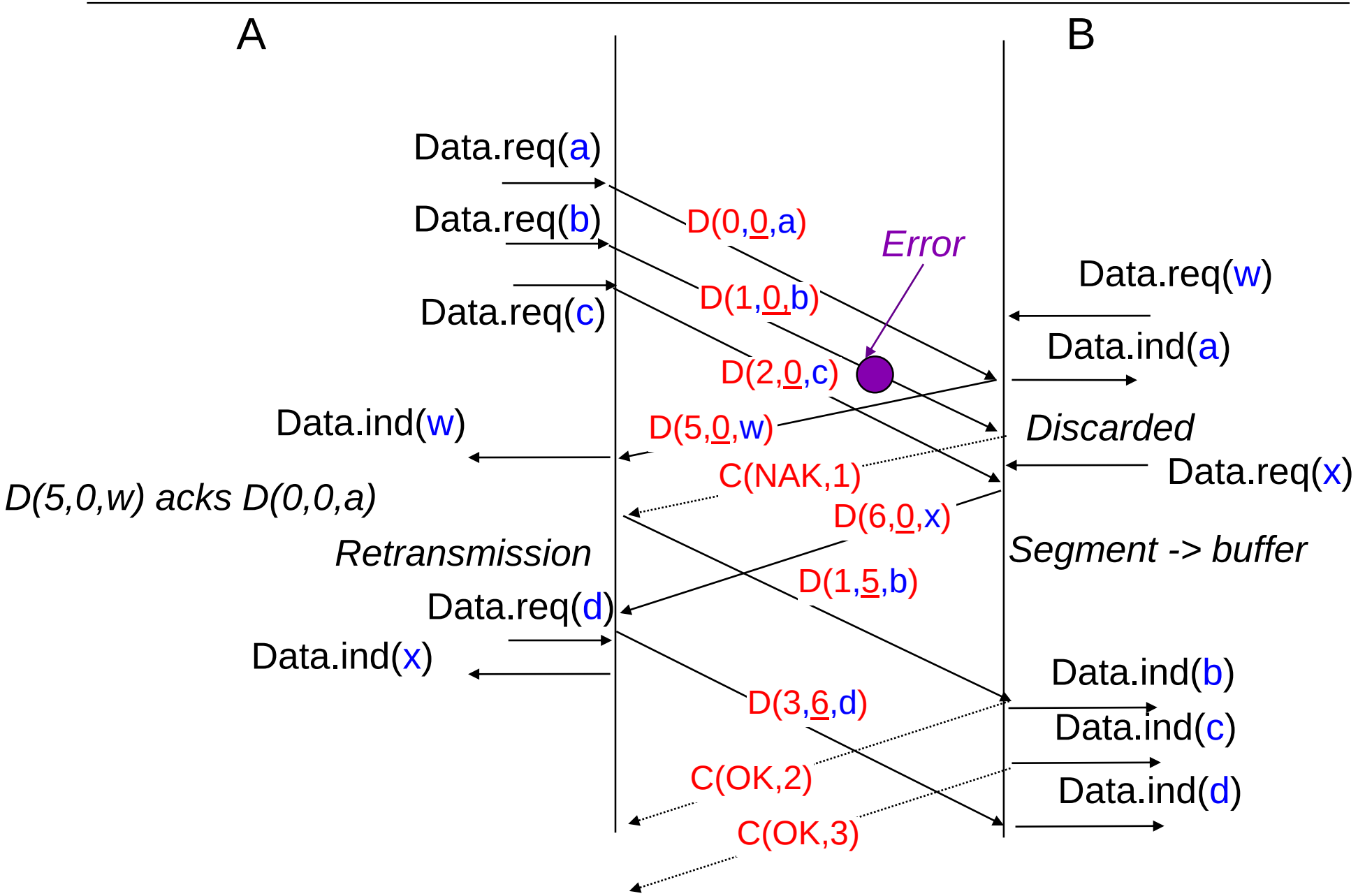
- Làm sao cho phép hai máy có thể truyền dữ liệu ?

- Nguyên lý
  - Mỗi máy đều gửi segment điều khiển và segment dữ liệu

  - Piggybacking
    - Đặt trường điều khiển trong gói tin dữ liệu (e.g. window, ack number) sao cho segment dữ liệu cũng mang thông tin điều khiển
    - Giảm tải truyền tin



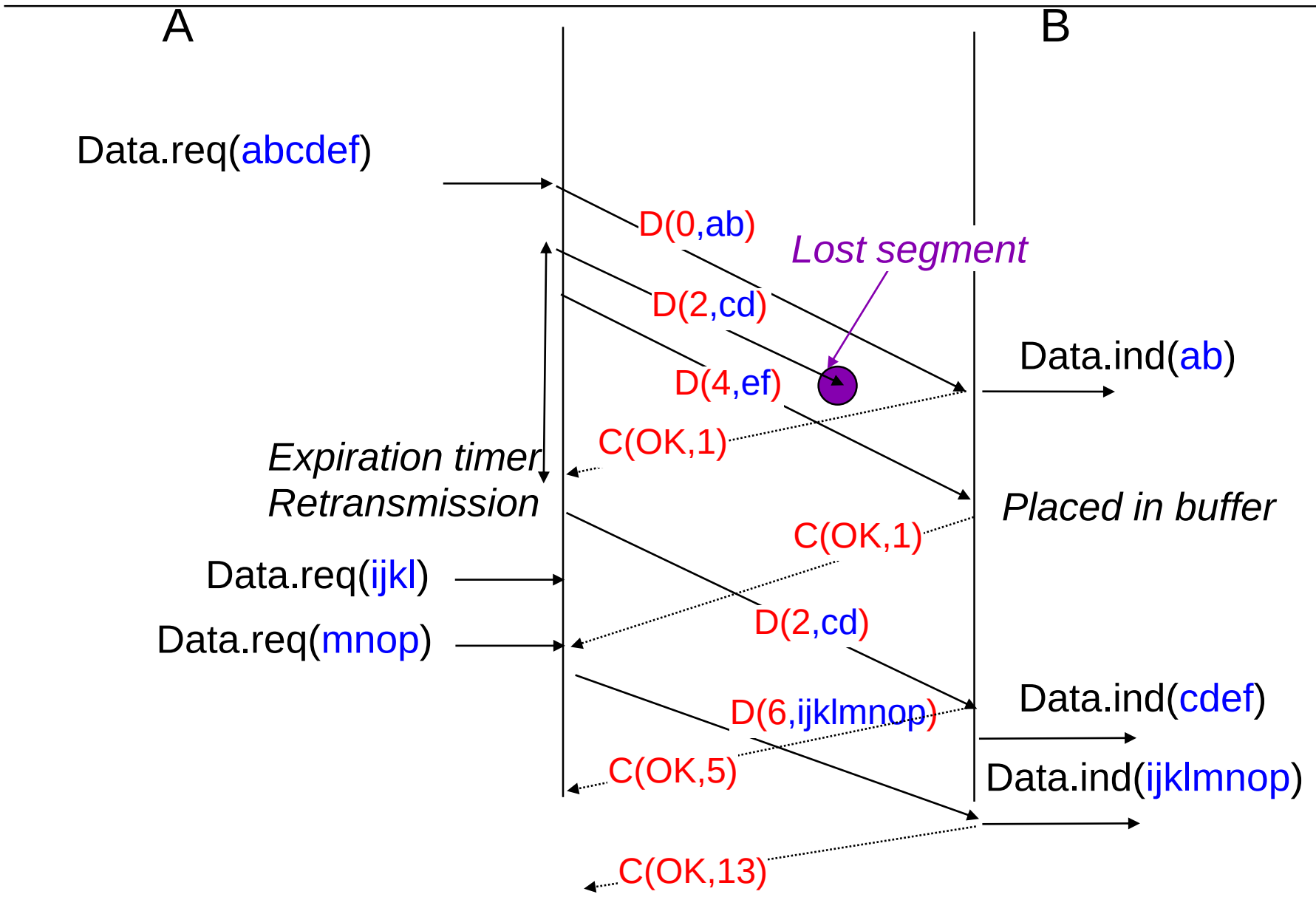
Luồng song công: ví dụ



Truyền luồng byte

- Làm sao cung cấp dịch vụ truyền luồng byte ?
  - Nguyên lý
    - Bên gửi chia luồng byte thành các segment
    - Bên nhận gửi chuyen khối dữ liệu của các segment đã nhận đúng số thứ tự cho người dùng
    - Thông thường, mỗi octet của luồng byte có một số thứ tự riêng, tiêu đề của segment chứa số thứ tự octet đầu của khối dữ liệu
      - Trong trường hợp này, kích cỡ cửa sổ được đo bằng byte

Dịch vụ luồng byte (2)



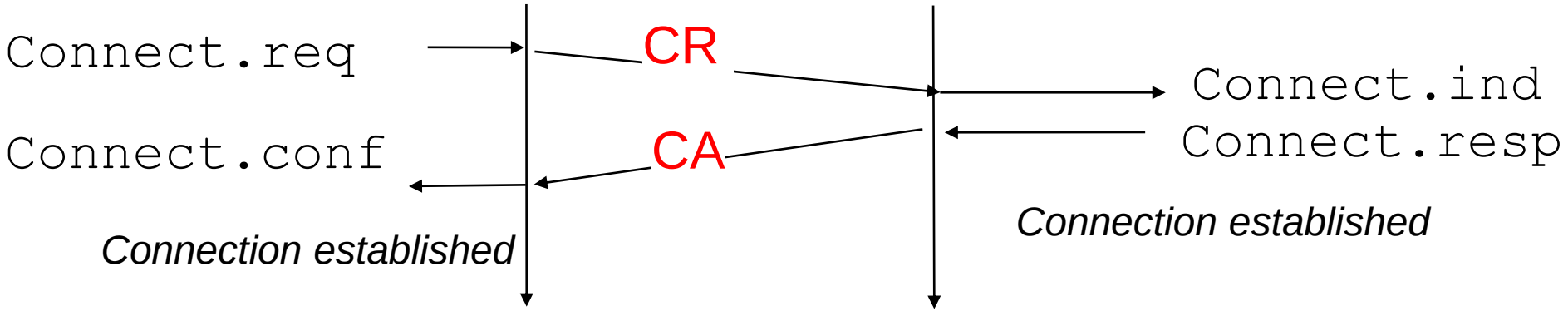
Phần 3 : Tầng giao vận

- Cơ bản
- Xây dựng cơ chế truyền tin tin cậy trên tầng giao vận
  - Truyền tin tin cậy
  - Thiết lập kết nối
  - Giải phóng kết nối
- UDP : giao thức không kết nối đơn giản
- TCP : giao thức có kết nối, tin cậy

Thiết lập kết nối giao vận

- Làm thế nào để mở kết nối giữa hai thực thể giao vận ?
  - Tầng mạng cung cấp dịch vụ không hoàn hảo
    - Đường truyền có thể có lỗi
    - Segment có thể bị mất
    - Segment có thể bị mất thứ tự
    - Segments có thể bị trùng
  - Giả thuyết
    - Chúng ta giả thuyết là cần một kết nối đơn giữa hai thực thể giao vận

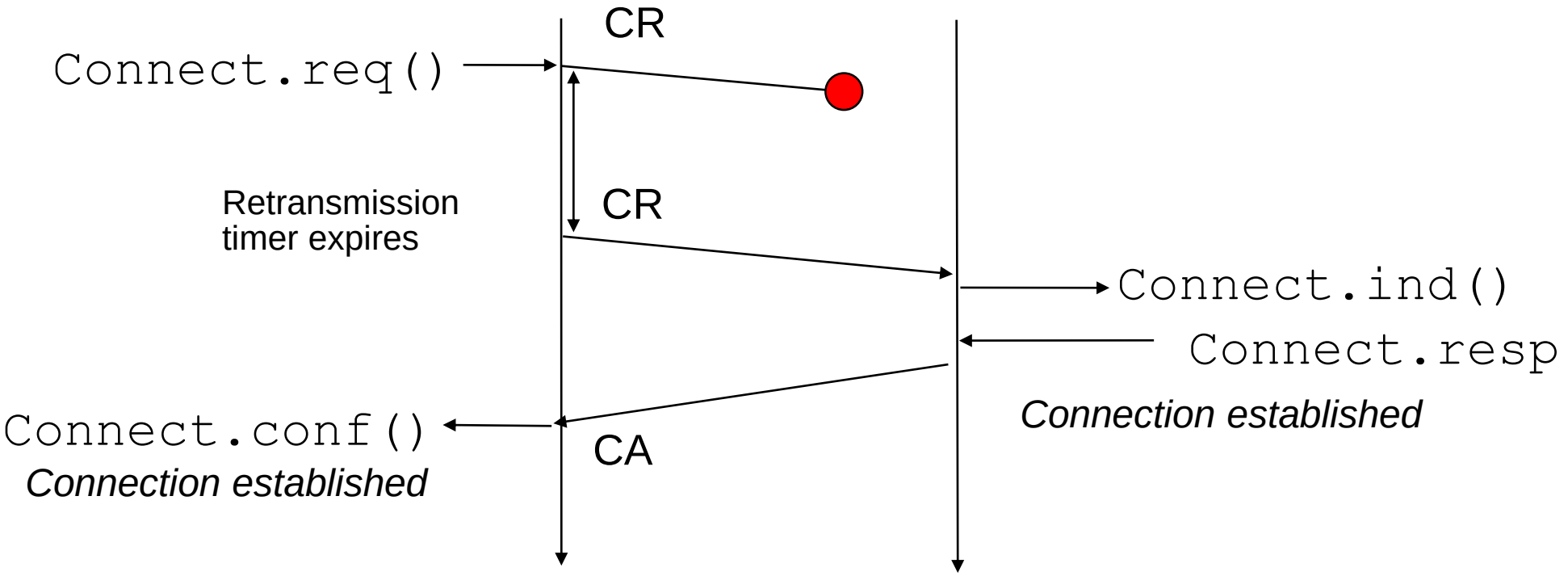
Giải pháp đơn giản



- Nguyên lý
  - 2 segment điều khiển
    - CR là để gửi yêu cầu kết nối
    - CA là để xác nhận kết nối
  - Liệu điều này có đủ với dịch vụ không hoàn hảo được cung cấp bởi tầng mạng?

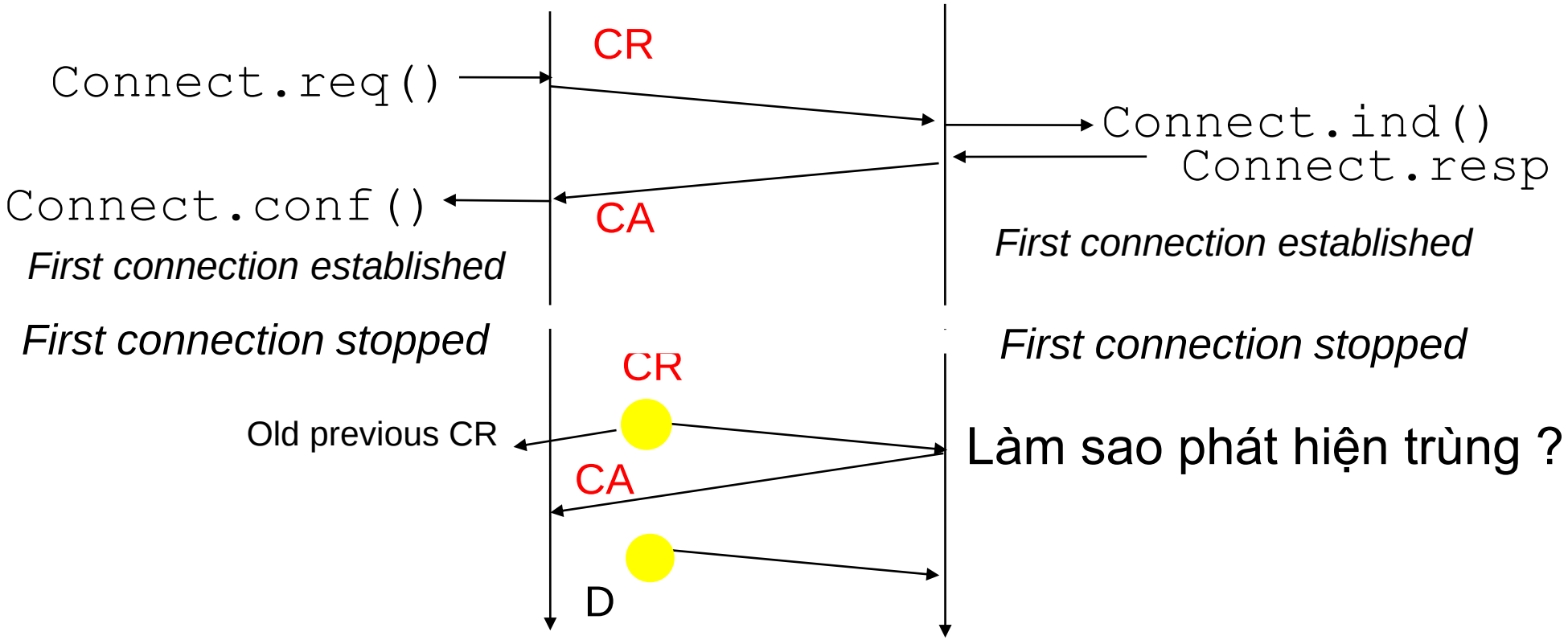
Giải pháp đơn giản (2)

- Làm sao giải quyết được mất mát và lỗi truyền ?
  - Phải sử dụng CRC hoặc checksum trong các segment điều khiển
  - Phải sử dụng đồng hồ truyền lại để chống lại mất mát



Thiết lập kết nối

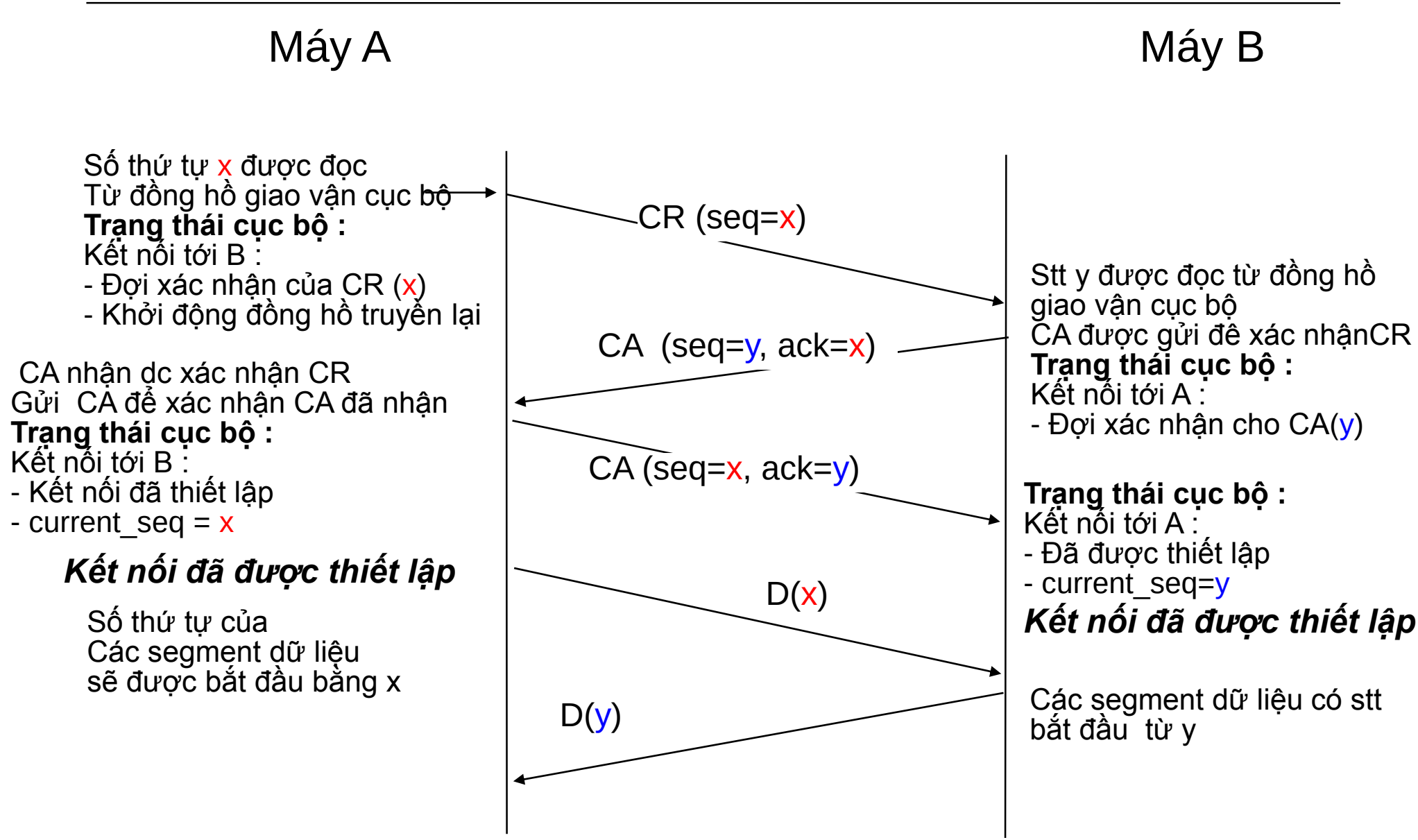
- Làm sao giải quyết trùng hay mất thứ tự segment điều khiển ?
  - Một segment CR bị trùng sẽ không dẫn đến việc thực hiện được hai kết nối thay vì một kết nối



Thiết lập kết nối (2)

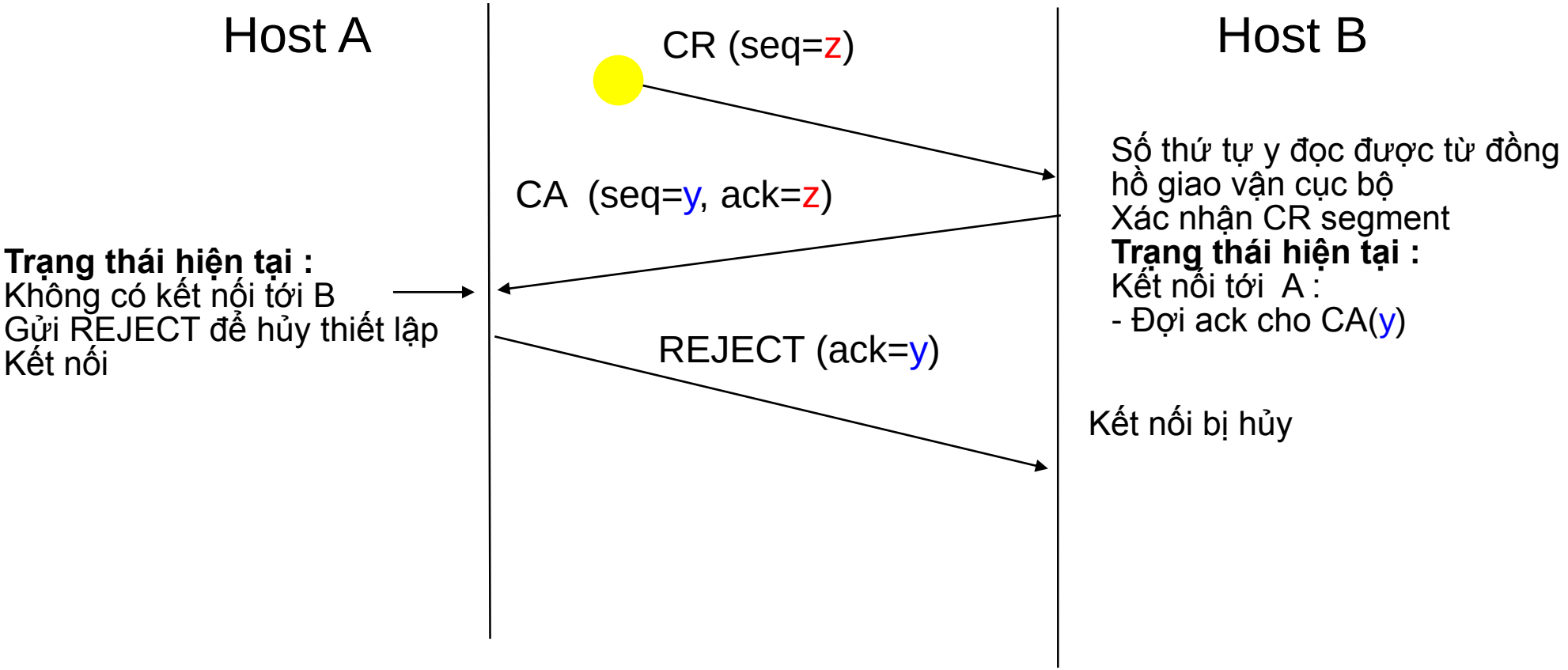
- Làm sao phát hiện kết nối ?
- Nguyên lý
  - Tầng mạng phải đảm bảo là gói tin không được tồn tại vĩnh viễn trong mạng
    - Không gói tin nào được sống quá MSL seconds
  - Tầng giao vận dựa trên đồng hồ của nó để phát hiện trùng gói tin thiết lập kết nối

Bắt tay ba bước



Bắt tay ba bước (2)

- Điều gì xảy ra nếu trùng
  - Trùng CR



No connection is established

Bắt tay ba bước(3)

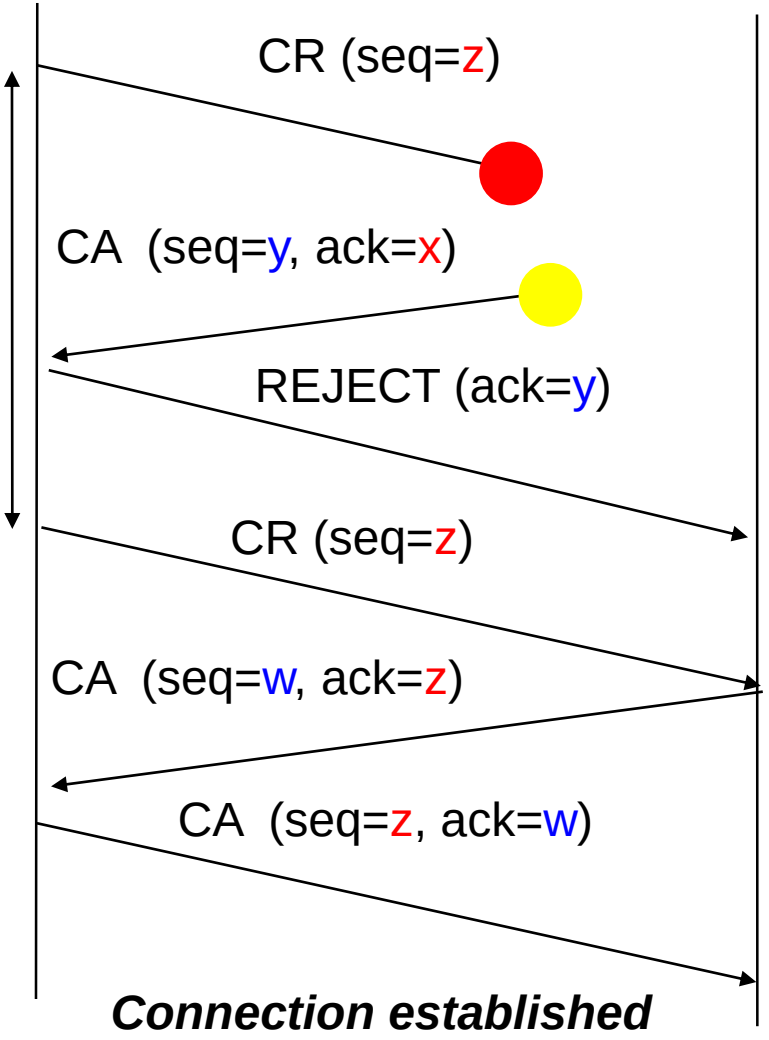
Host A

Host B

Số thứ tự z đọc được từ đồng hồ giao vận cục bộ

Trạng thái cục bộ :
Kết nối tới B :
- Đợi ack cho CR (z)
- Khởi tạo đồng hồ truyền lại

Trạng thái hiện tại không chứa CR với seq=x



Retransmission timer expires

Trạng thái hiện tại không chứa segment với seq=y
REJECT bị bỏ qua

CA nhận được xác nhận cho CR
Gửi CA để xác nhận CA đã nhận

Trạng thái cục bộ :
Kết nối tới B :
- Đã được kết nối
- current_seq = z

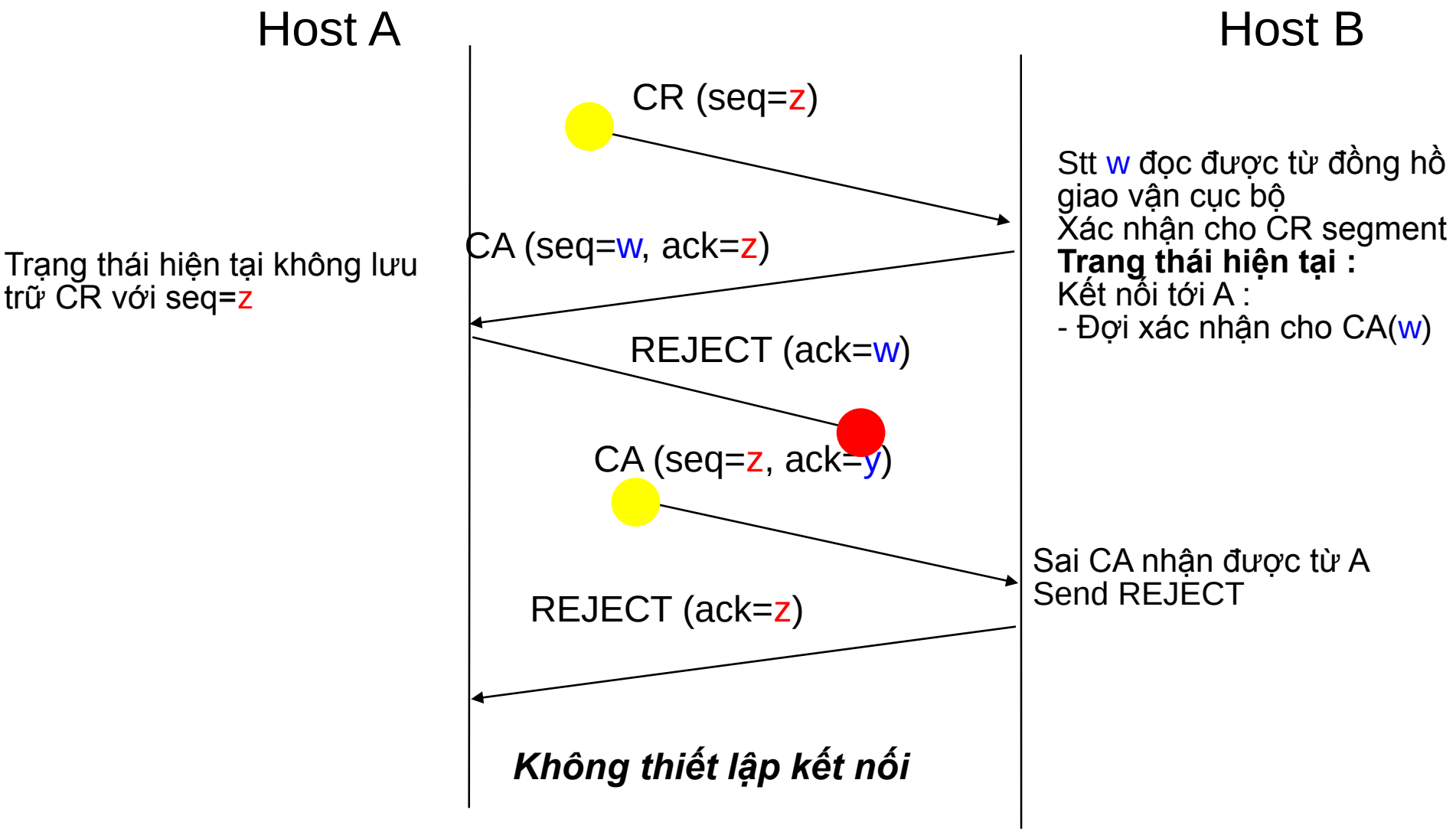
Số thứ tự w đọc được từ đồng hồ giao vận cục bộ
CA được gửi để xác nhận CR

Trạng thái cục bộ :
Kết nối tới A :
- Đợi ack cho CA(w)

Connection established

Bắt tay ba bước(4)

- Kịch bản khác



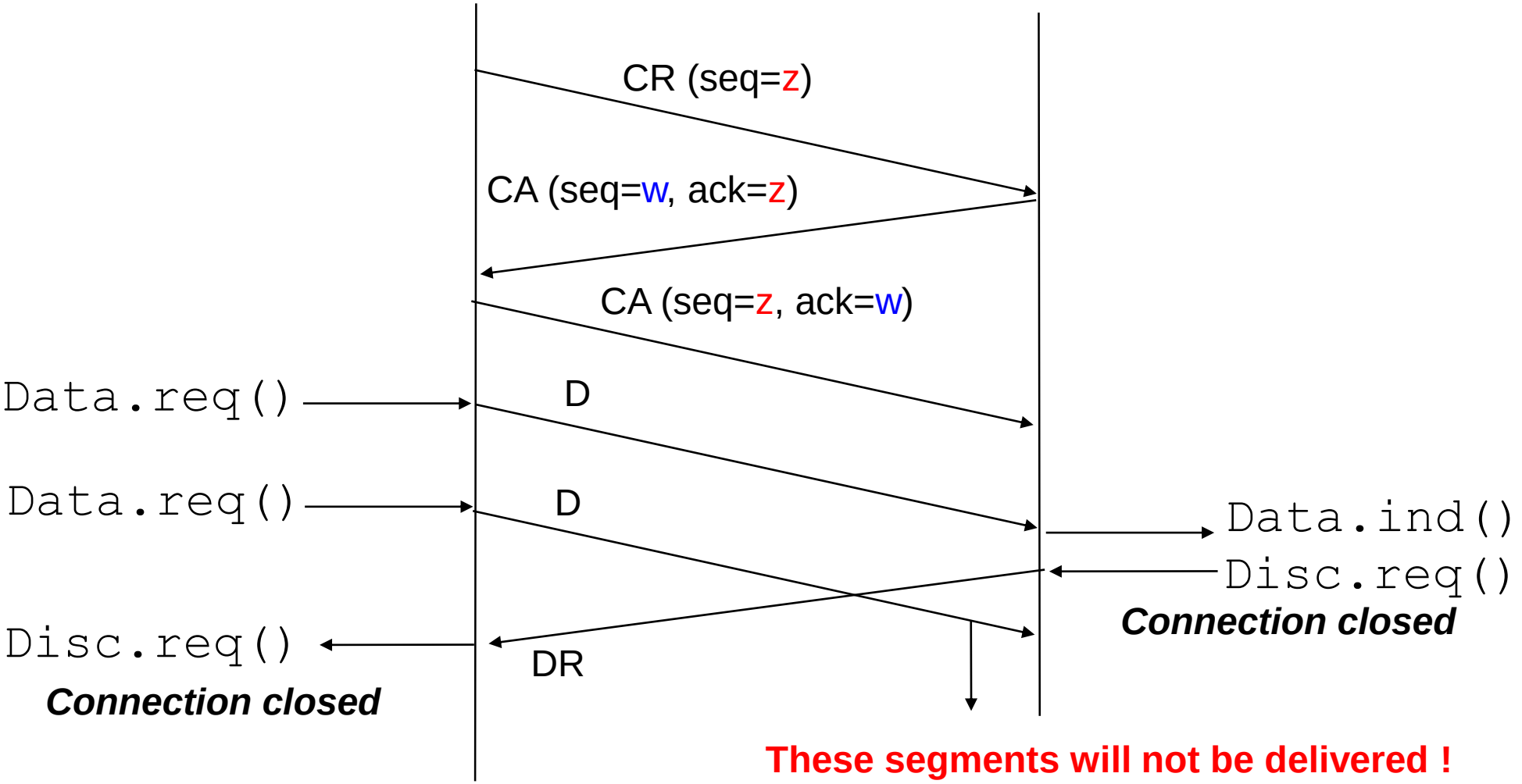
Phần 3 : Transport Layer

- Basics
- Building a reliable transport layer
  - Reliable data transmission
  - Connection establishment
  - Connection release
- UDP : a simple connectionless transport protocol
- TCP : a reliable connection oriented transport protocol

Giải phóng kết nối

- Một kết nối có thể được thực hiện ở cả hai phía
- Các kiểu giải phóng kết nối
  - Giải phóng kết nối đột ngột
    - Một trong những thực thể giao vận đóng cả hai hướng truyền dữ liệu
    - Có thể dẫn đến mất mát dữ liệu
  - Giải phóng kết nối khoan thai
    - Mỗi thực thể giao vận đóng hướng truyền tin của nó
    - Kết nối được đóng khi dữ liệu đã được truyền chính xác

Giải phóng đột ngột

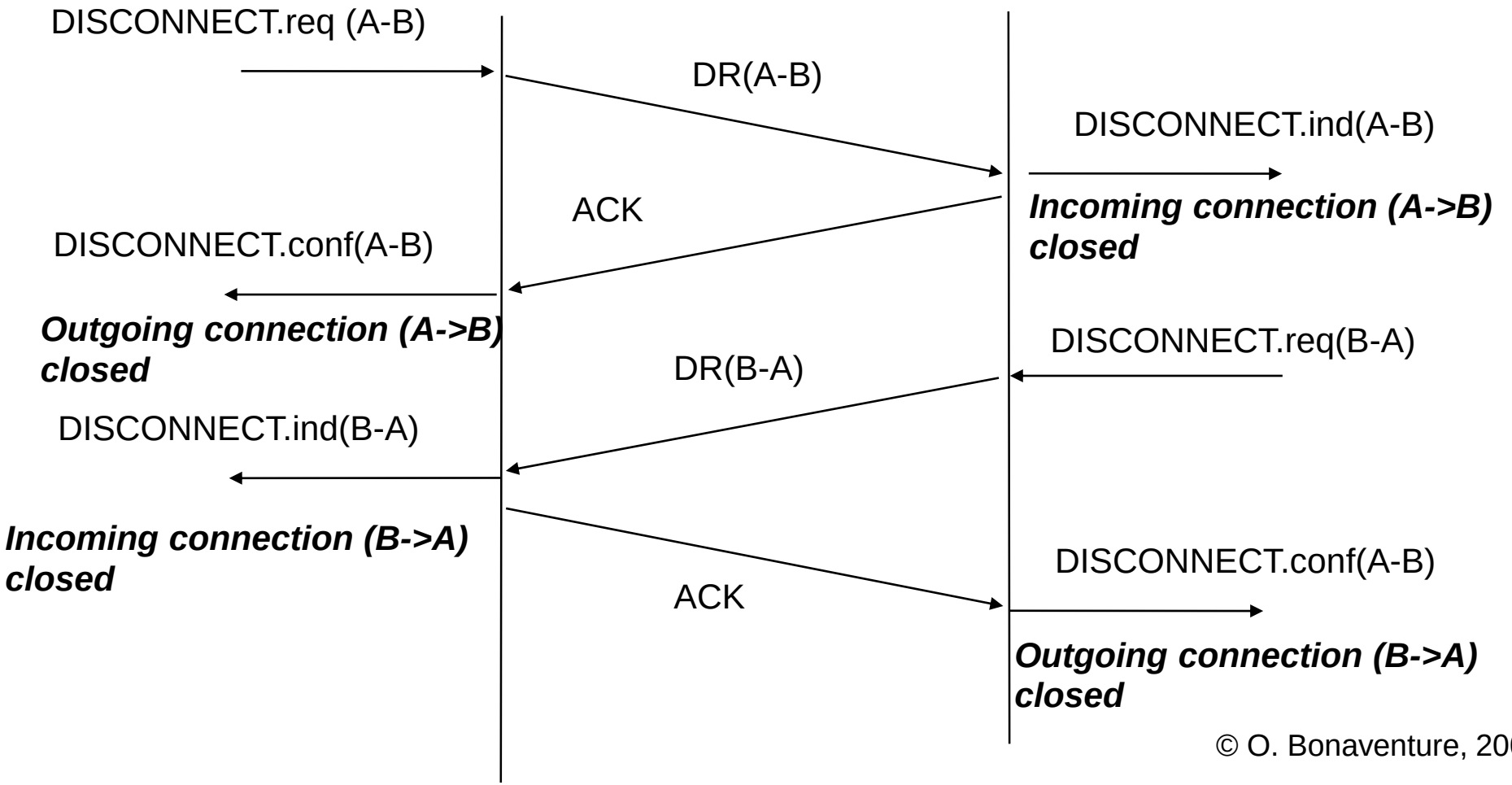


Giải phóng đột ngột (2)

- Thực thể ứng dụng có thể bị bắt buộc đóng kết nối
  - Cùng một segment dữ liệu có thể bị truyền lại nhiều lần mà không nhận được báo nhận
  - Tầng mạng thông báo là máy đích không thể kết nối
  - Thực thể tầng mạng không còn đủ tài nguyên để phục vụ kết nối này (e.g. không còn đủ bộ nhớ)
- Trong trường hợp này, thực thể tầng giao vận buộc phải đóng kết nối đột ngột

Đóng kết nối khoan thai

- Nguyên lý
  - Mỗi thực thể đóng hướng truyền của nó khi hết dữ liệu truyền



Đánh giá độ tin cậy của tầng giao vận

- Giới hạn
  - Tầng giao vận đảm bảo tính tin cậy trong suốt thời gian sống của kết nối giao vận
    - Nếu kết nối được đóng một cách khoan thai thì tất cả dữ liệu gửi đi sẽ được nhận chính xác
    - việc truyền dữ liệu sẽ có thể không còn tin (e.g. mất segment) nếu kết nối bị đóng đột ngột
- Tầng giao vận không tự hồi phục nếu có giải phóng kết nối đột ngột
  - Giải pháp khả dĩ
    - Ứng dụng mở lại kết nối và truyền lại dữ liệu
    - Nên có tầng Phiên
    - Phải có cơ chế quản lý giao dịch

Module 3 : Transport layer

- Basics
- Building a reliable transport layer
- UDP : a simple connectionless transport protocol
- TCP : a reliable connection oriented transport protocol

Giao thức giao vận đơn giản

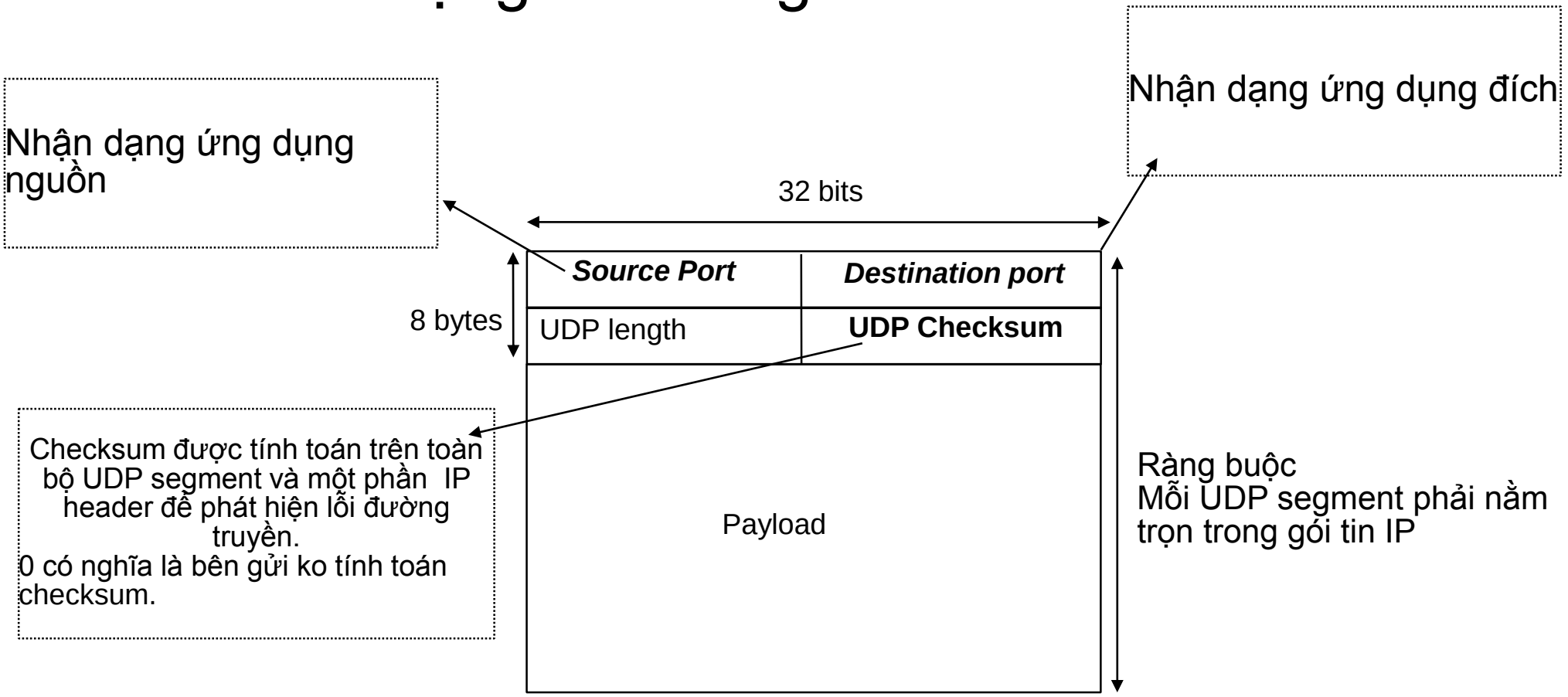
- User Datagram Protocol (UDP)
  - Giao thức giao vận đơn giản
- Mục đích
  - Cho phép các ứng dụng trao đổi các SDUs nhỏ dựa trên các dịch vụ của tầng IP
    - Trên đa số các hệ điều hành, việc gửi các gói tin IP thô cần có sự ưu tiên đặc biệt trong khi đó các ứng dụng có thể sử dụng trực tiếp các dịch vụ giao vận
- Ràng buộc
  - Cài đặt thực thể giao vận UDP càng đơn giản càng tốt

UDP : các lựa chọn thiết kế

- Cơ chế nào cho UDP ?
  - Định danh ứng dụng
    - Các ứng dụng chạy trên cùng một máy phải có thể dử dụng dịch vụ UDP
  - Giải pháp
    - Cổng nguồn để định danh ứng dụng gửi
    - Cổng đích để định danh ứng dụng đích
    - Mỗi segment UDP chứa cả cổng nguồn và cổng đích
  - Phát hiện lỗi truyền

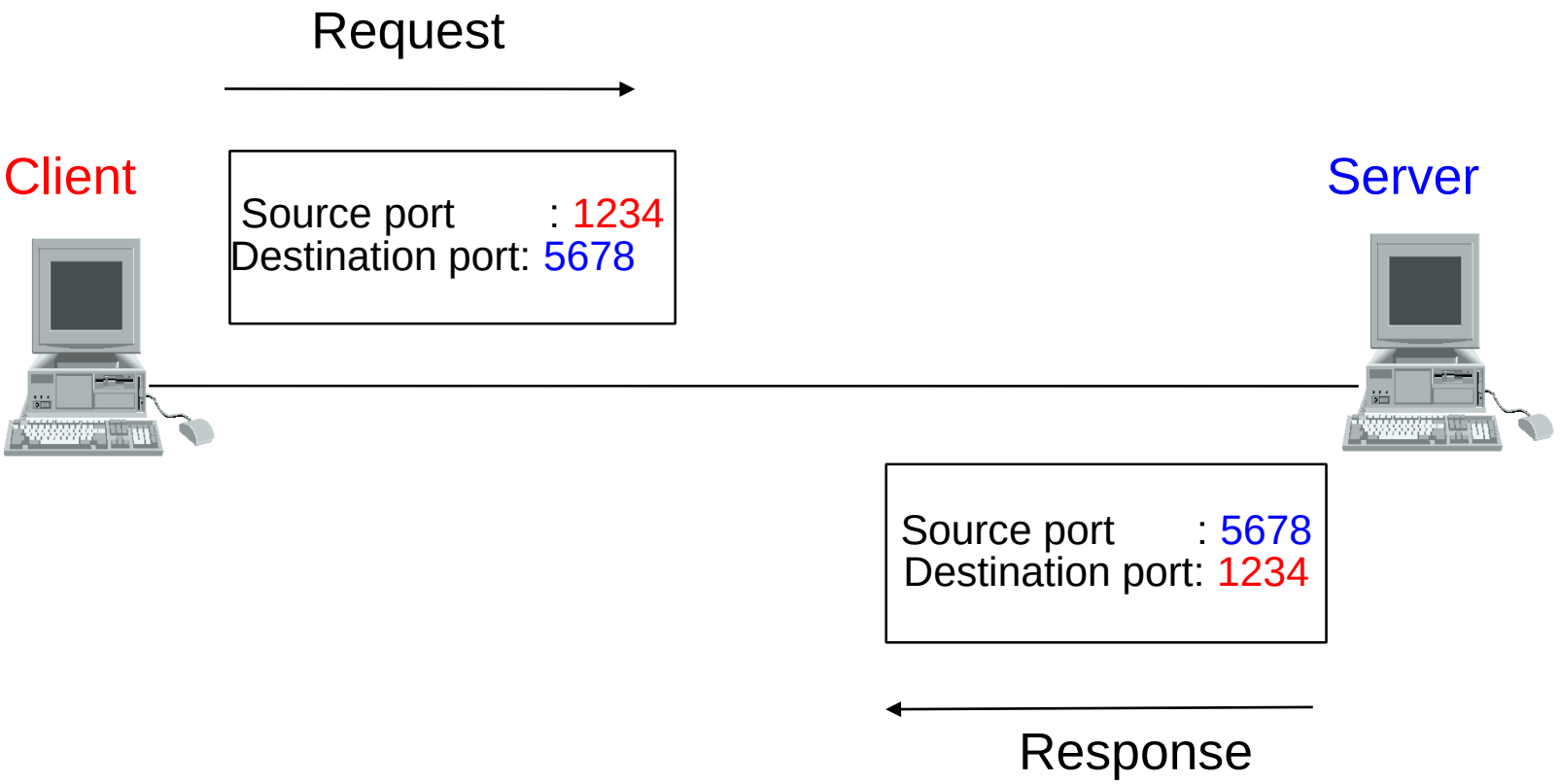
Giao thức UDP

- 2 thực thể UDP trao đổi UDP segments
- Khuôn dạng UDP segment



Giao thức UDP (2)

- Sử dụng cổng UDP



Hạn chế của dịch vụ UDP

- Hạn chế
  - Độ dài tối đa của UDP SDUs phụ thuộc vào độ dài tối đa của IP packets
  - Dịch vụ truyền tin không kết nối không tin cậy
    - SDUs có thể bị mất nhưng lỗi truyền sẽ phát hiện được
    - UDP không hỗ trợ đánh số thứ tự gói tin
    - UDP không phát hiện hay khắc phục trùng gói tin

Sử dụng UDP

- Các ứng dụng request-response applications với các yêu cầu và phản hồi ngắn, độ trễ ngắn nhất là trong môi trường
  - DNS
  - Remote Procedure Call
  - NFS
  - Games
- Truyền tin đa phương tiện nới mà truyền tin tin cậy không thực sự cần thiết và việc truyền lại có thể gây ra độ trễ lớn
  - Voice over IP
  - Video over IP

Module 3 : Transport Layer

- Basics
- Building a reliable transport layer
- UDP : a simple connectionless transport protocol
- TCP : a reliable connection oriented transport protocol
  - TCP connection establishment
  - TCP connection release
  - Reliable data transfer
  - Congestion control

TCP

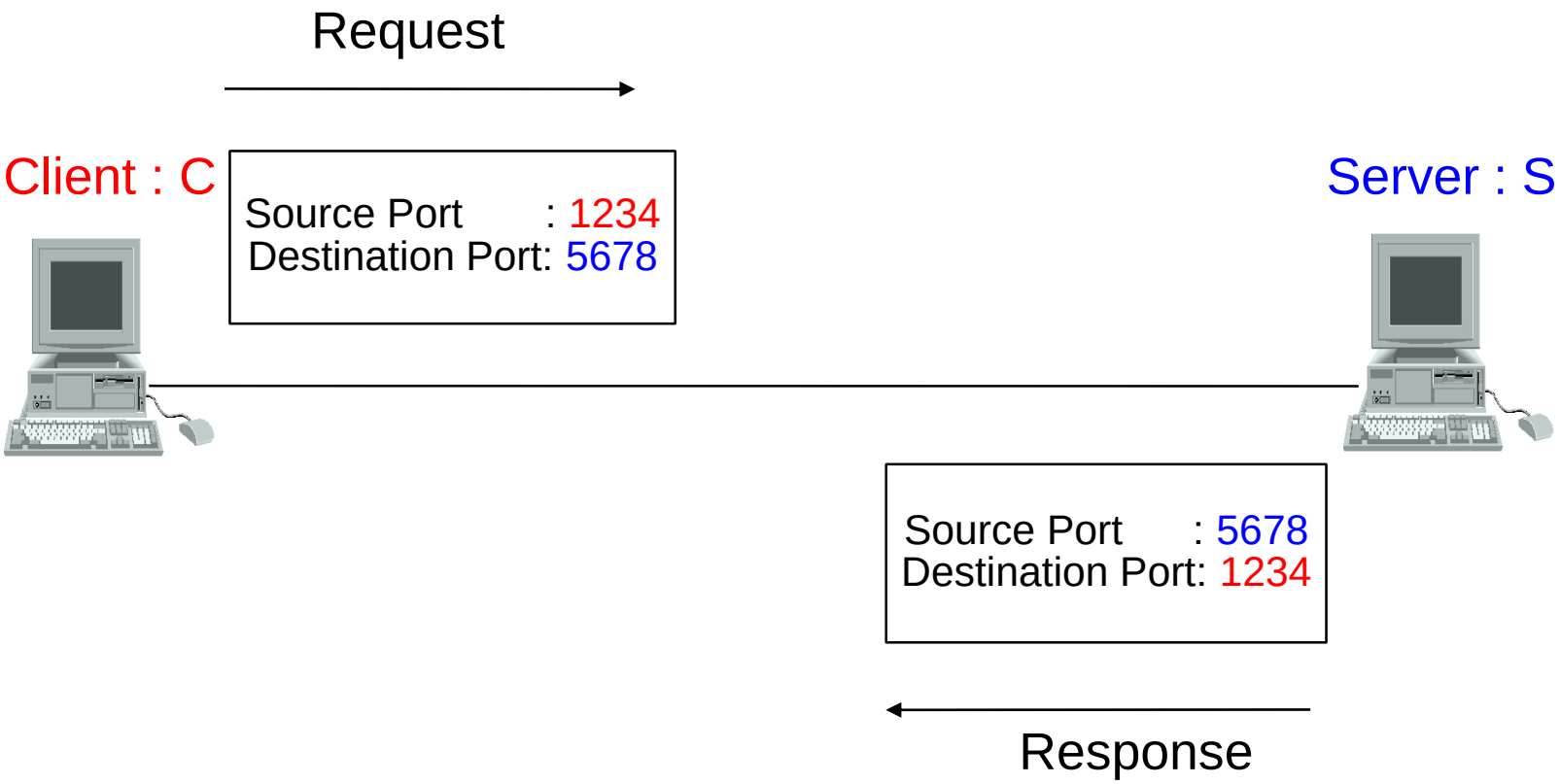
- Transmission Control Protocol
- Cung cấp dịch vụ truyền luồng thông tin tin cậy
- Đặc điểm của UDP
  - Kết nối TCP
  - Truyền tin tin cậy
    - Không mất mát gói tin
    - Không lỗi đường truyền
    - Không trùng lặp gói tin
  - Truyền tin song sông
  - TCP dựa trên dịch vụ của IP
  - TCP chỉ truyền tin điểm tới điểm (unicast)

Kết nối TCP

- Làm sao định danh kết nối TCP
  - Địa chỉ ứng dụng nguồn
    - IP Address của máy nguồn
    - Số cổng TCP của ứng dụng trên máy nguồn
  - Địa chỉ của ứng dụng đích
    - IP Address của máy đích
    - Số cổng TCP của ứng dụng trên máy đích
- Mỗi TC segment chứa định danh của kết nối mà nó thuộc về

Kết nối TCP (2)

- Sử dụng số cổng TCP



Established TCP connections on client

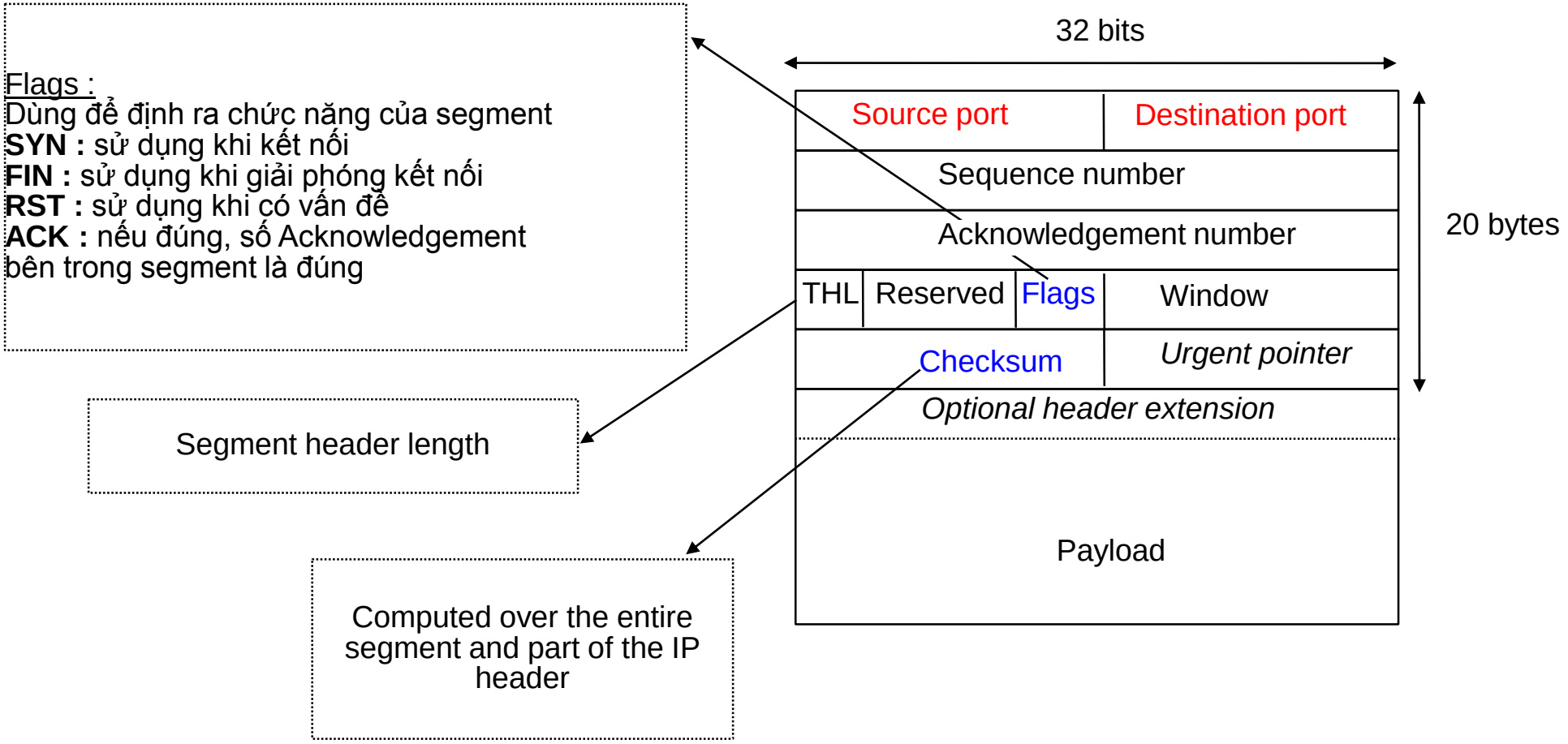
Local IP	Remote IP	Local Port	Remote Port
C	S	1234	5678

Established TCP connections on server

Local IP	Remote IP	Local Port	Remote Port
S	C	5678	1234

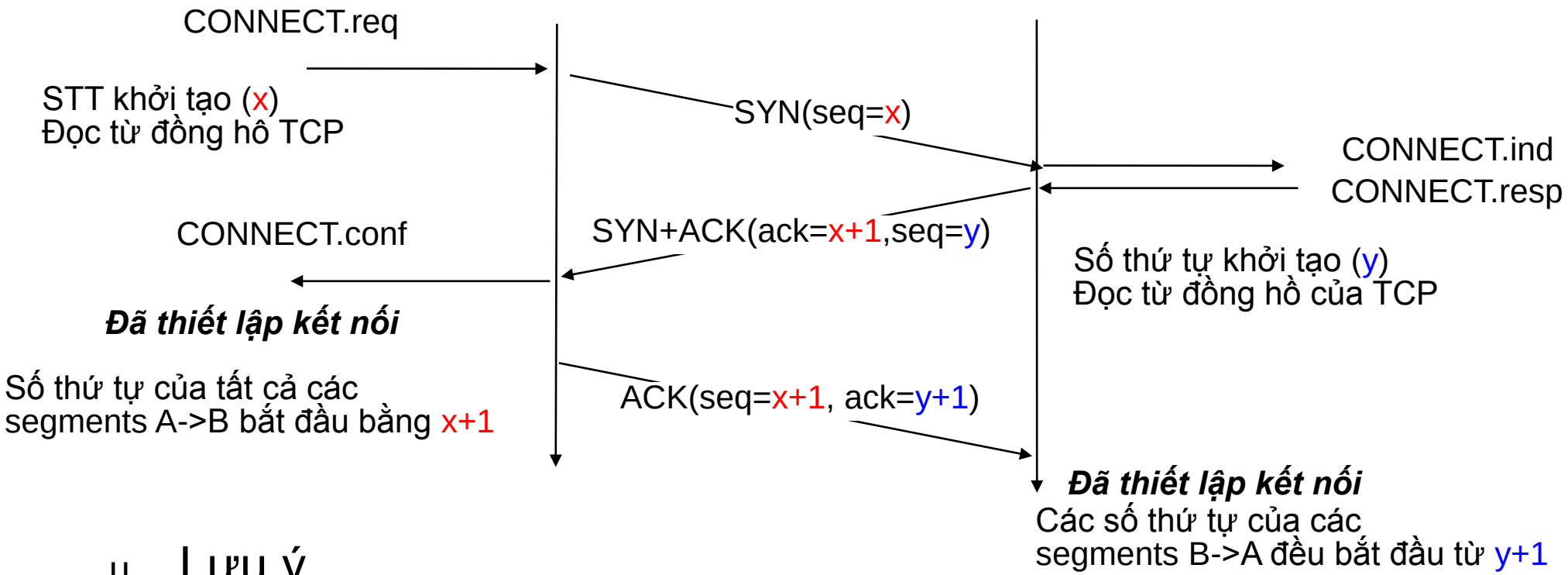
Giao thức TCP

Khuôn dạng segment TCP



Thiết lập kết nối TCP

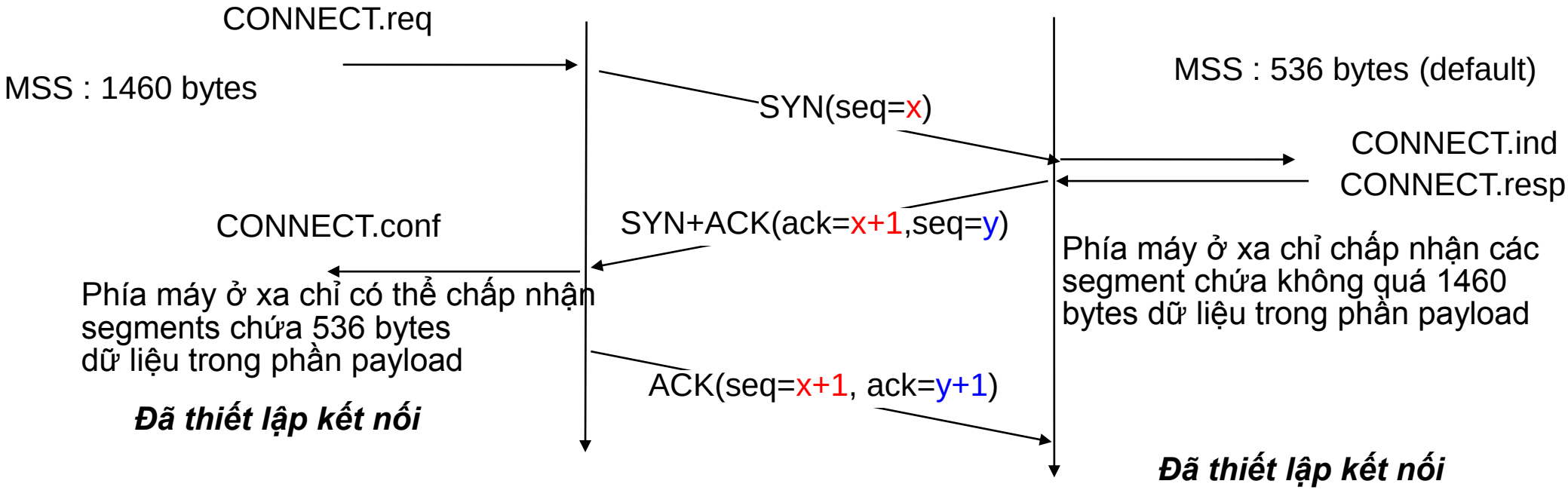
Bắt tay ba bước



- Lưu ý
  - Thiết lập cờ SYN flag trong một segment tiêu tốn một số thứ tự
  - Cờ ACK chỉ được thiết lập khi trường acknowledgement chứa một giá trị đúng
  - Khuyến nghị ngầm định cho đồng hồ TCP là tăng ít nhất sau 4 microsecon và sau khi kết nối TCP được thiết lập

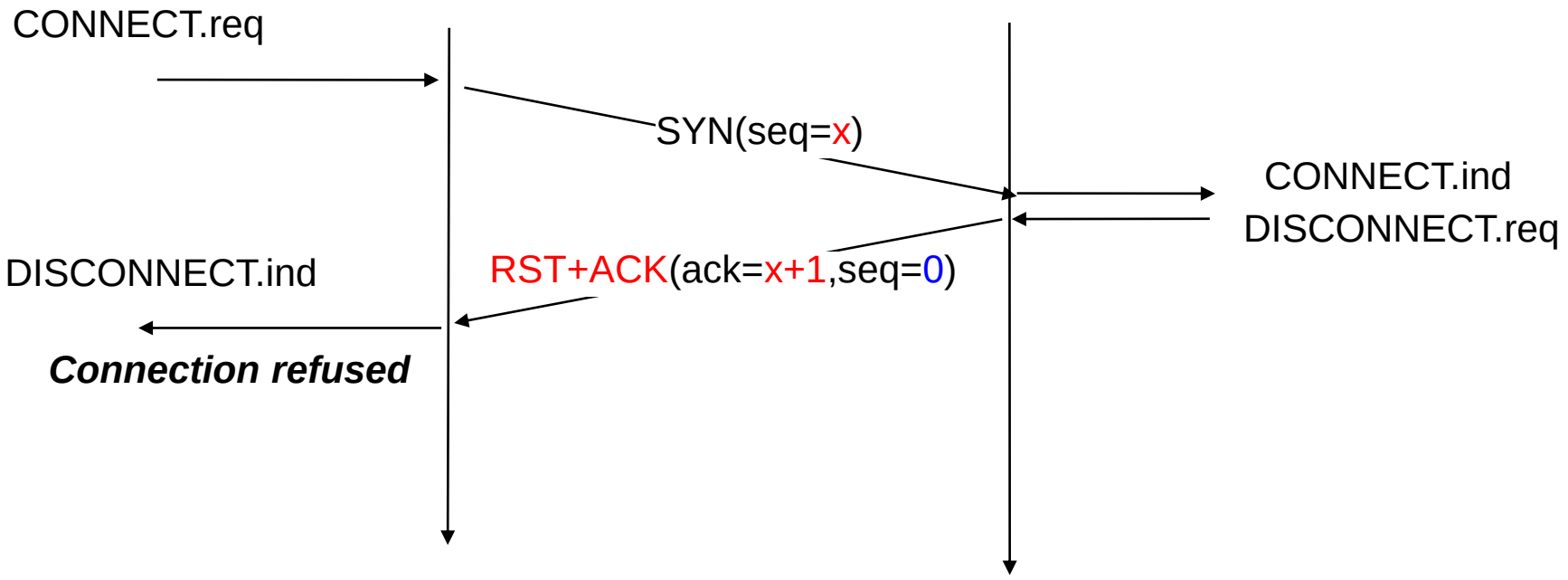
Thiết lập kết nốiTCP (2)

- Thương lượng các tùy chọn
  - Trong suốt quá trình thiết lập kết nối, hai bên có thể thương lượng một số tùy chọn mở rộng TCP
    - Các tùy chọn được mã hóa bên trong phần tùy chọn của TCP header
      - Maximum segment size (MSS)
      - RFC1323 timestamp extensions
      - Selective Acknowledgments



Thiết lập kết nối TCP (3)

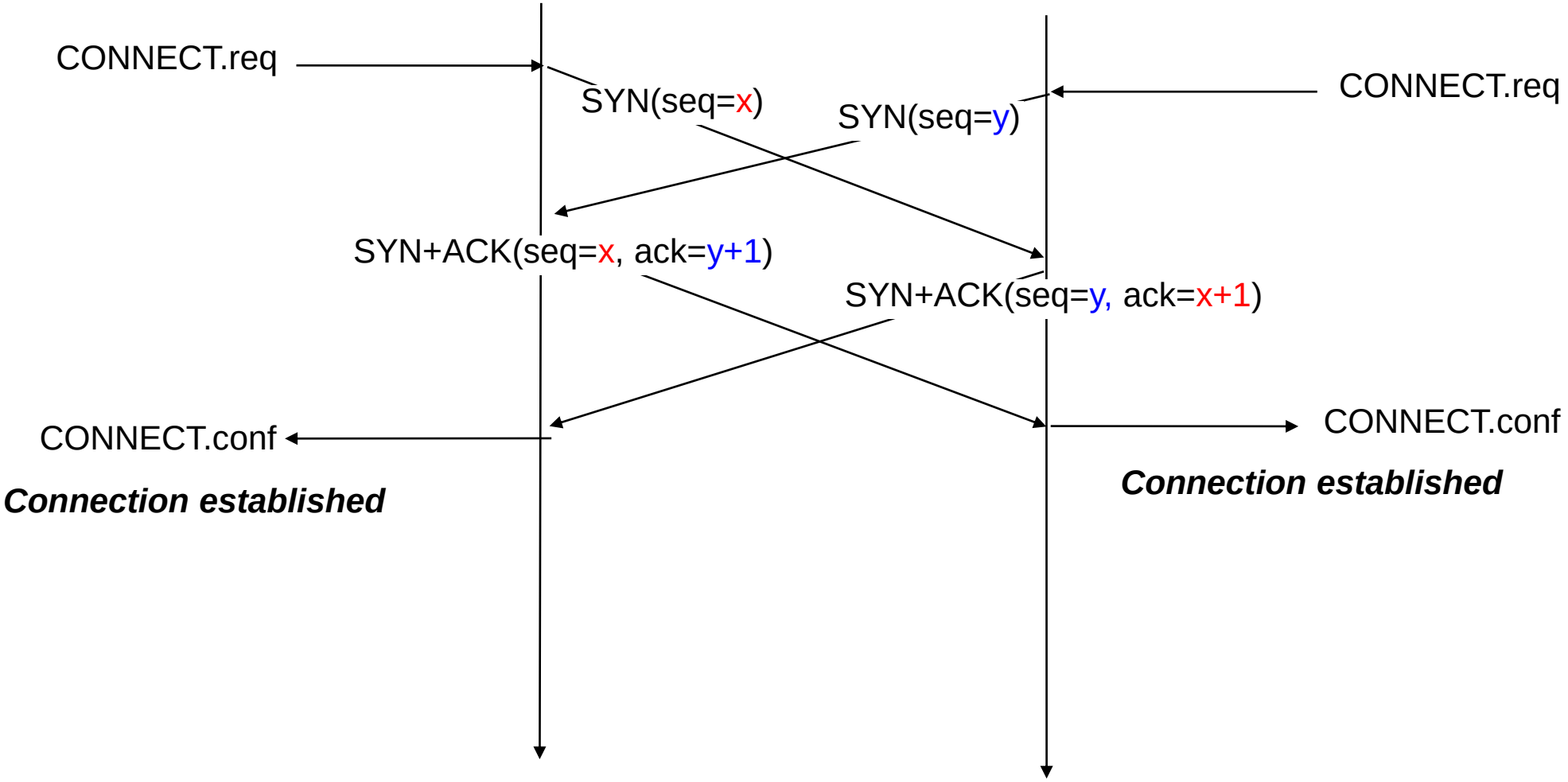
- Từ chối thiết lập kết nối



Thực thể TCP không bao giờ gửi segment RST khi nhận được segment khác

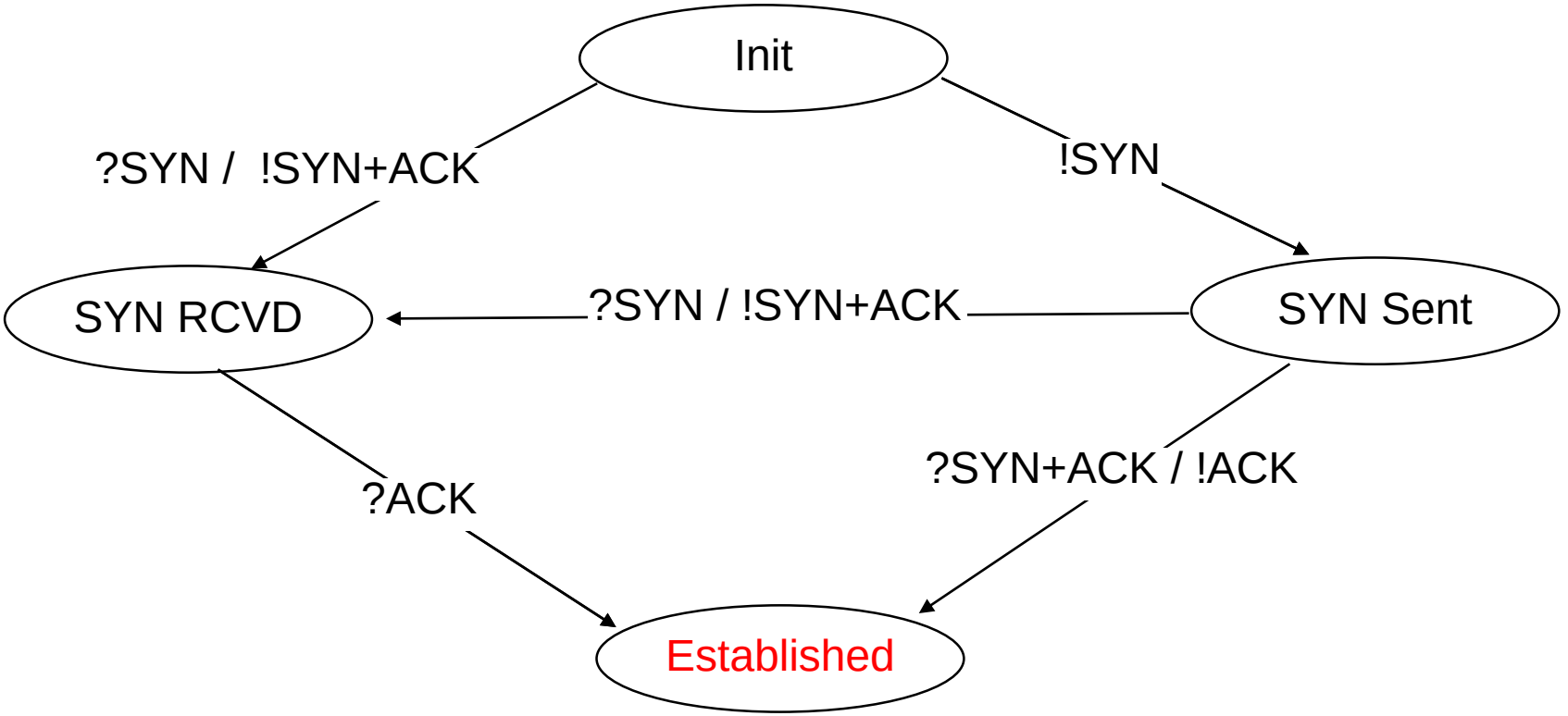
Thiết lập kết nối TCP (4)

- Thiết lập đồng thời



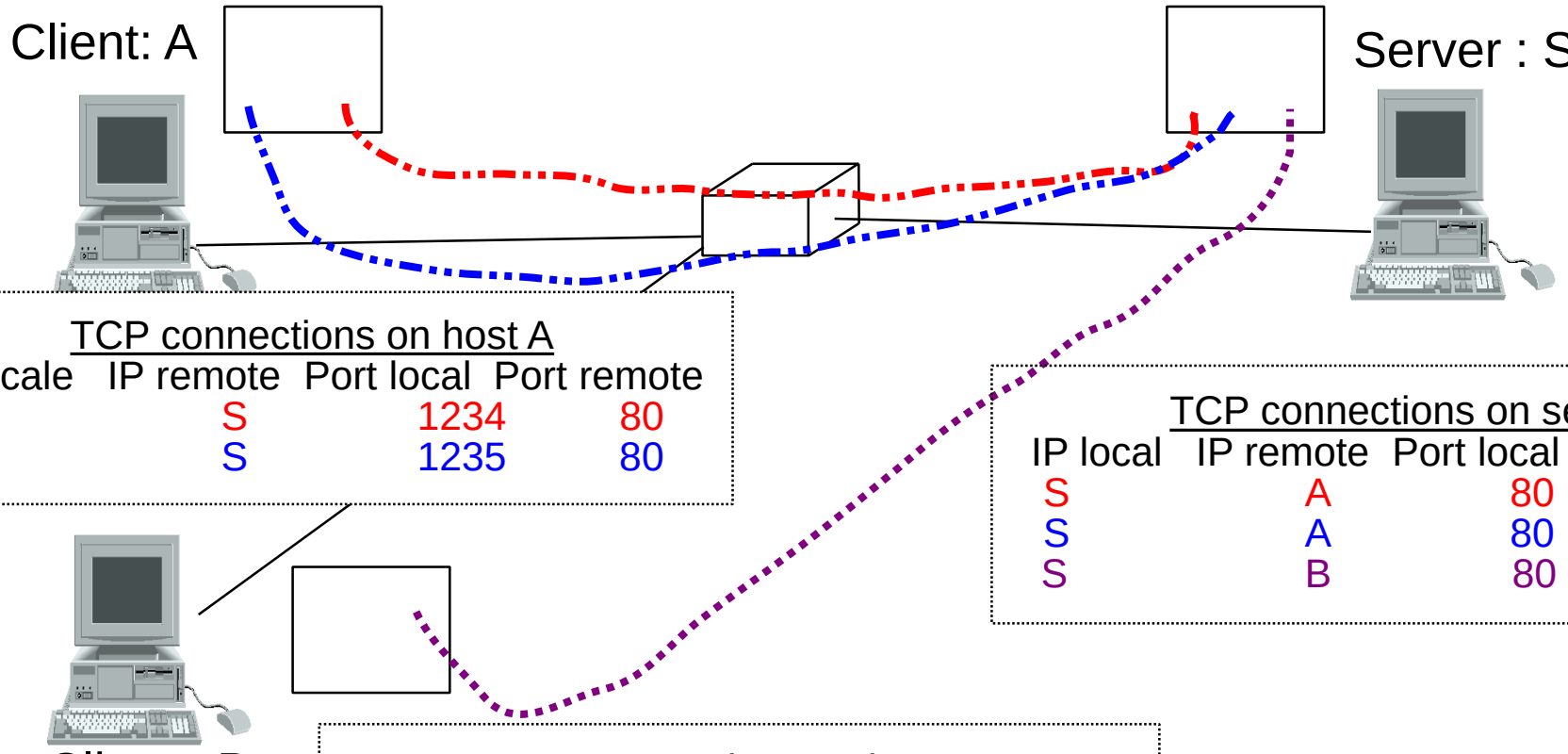
Thiết lập kết nối TCP (5)

- Biểu diễn dưới dạng máy trạng thái hữu hạn



Thiết lập kết nốiTCP (6)

- Làm sao mở nhiều kết nối TCP đồng thời ?



TCP connections on host A

IP locale	IP remote	Port local	Port remote
A	S	1234	80
A	S	1235	80

TCP connections on server

IP local	IP remote	Port local	Port remote
S	A	80	1234
S	A	80	1235
S	B	80	1235

TCP connections on host B

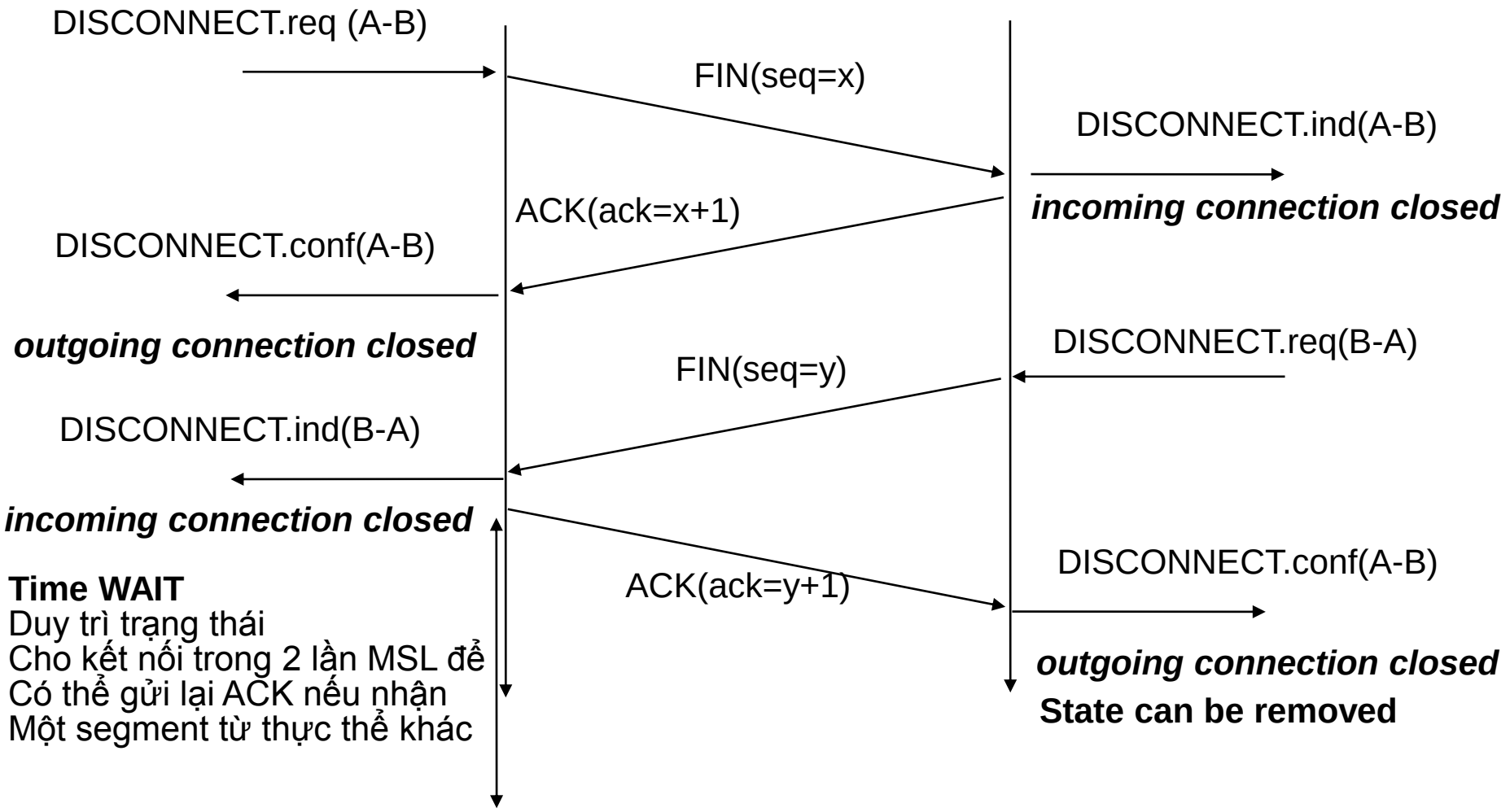
IP locale	IP remote	Port local	Port remote
B	S	1235	80

Module 3 : Transport Layer

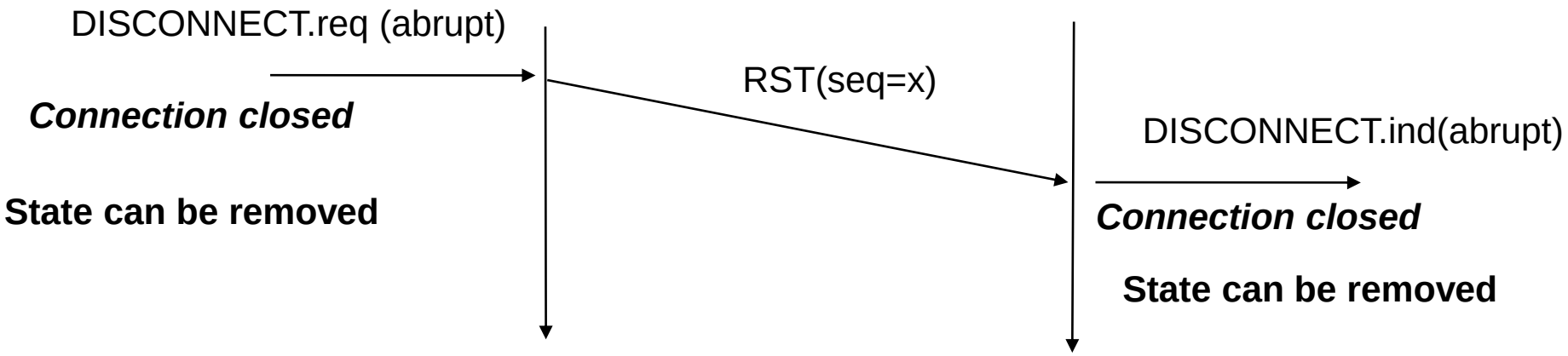
- Basics
- Building a reliable transport layer
- UDP : a simple connectionless transport protocol
- TCP : a reliable connection oriented transport protocol
  - TCP connection establishment
  - TCP connection release
  - Reliable data transfer
  - Congestion control

Giải phóng kết nốiTCP

- Đóng kết nối TCP chậm

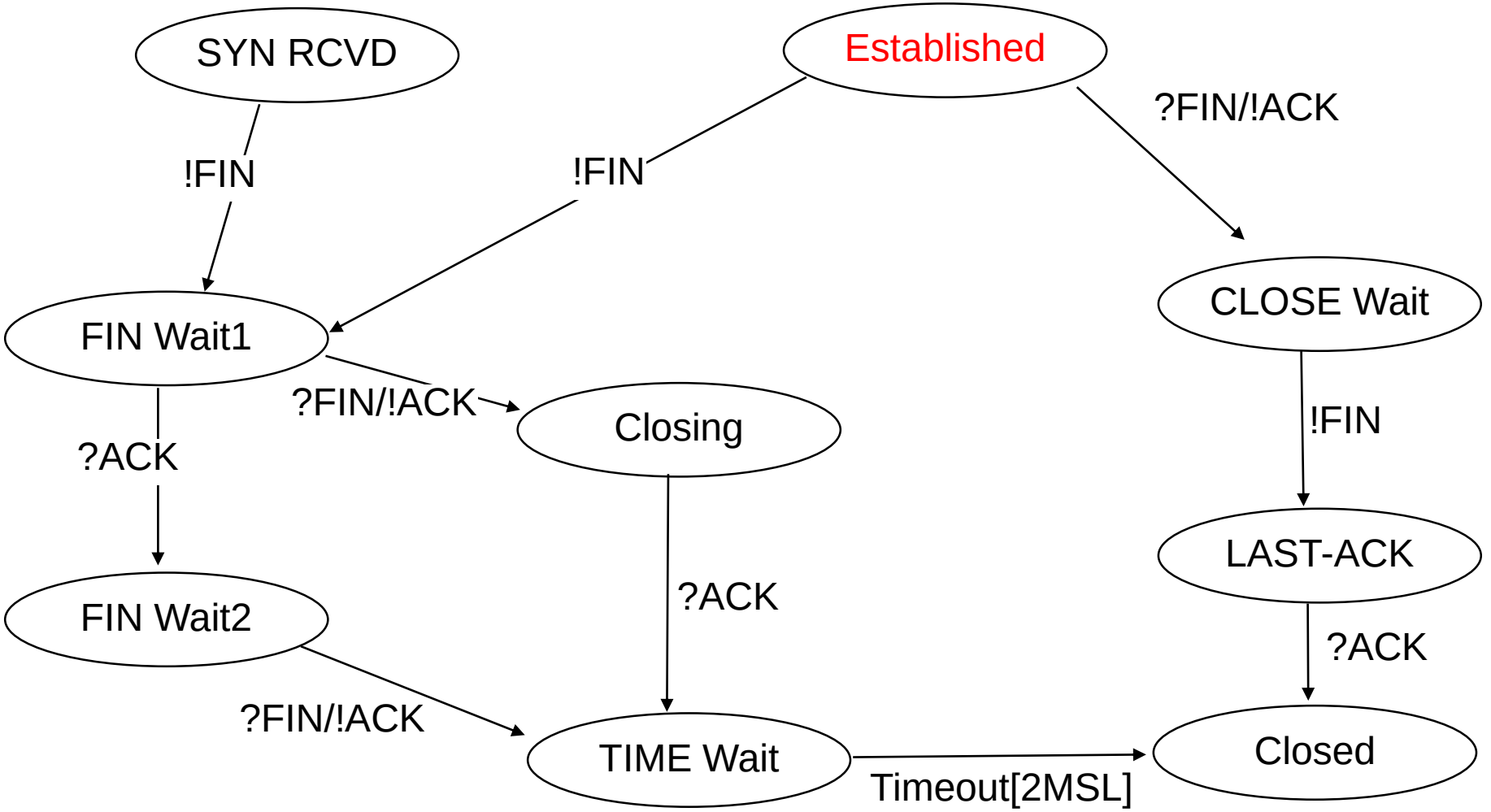


Giải phóng kết nối TCP đột ngột



- Segment dữ liệu có thể bị mất khi có ngắt kết nối đột ngột
- Không có thực thể nào cần đợi TIME_WAIT state sau khi giải phóng kết nối
  - Dẫu sao, mọi segment nhận được khi không có trạng thái nào gây ra việc truyền RST segment

Giải phóng kết nối TCP

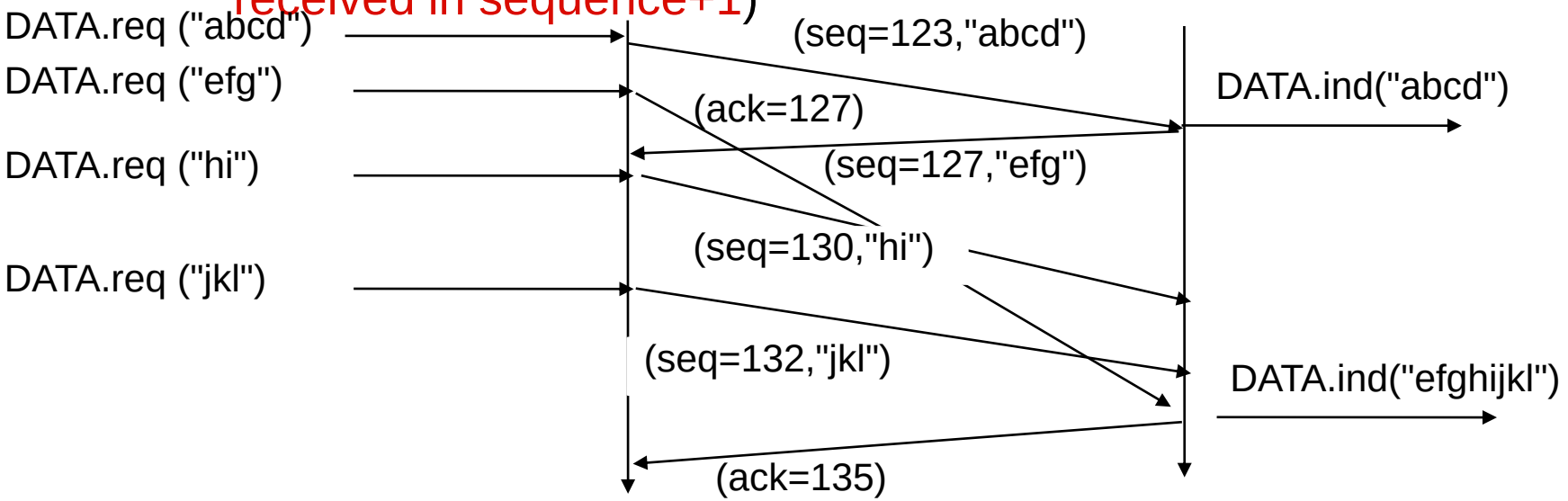


Module 3 : Transport Layer

- Basics
- Building a reliable transport layer
- UDP : a simple connectionless transport protocol
- TCP : a reliable connection oriented transport protocol
  - TCP connection establishment
  - TCP connection release
  - Reliable data transfer
  - Congestion control

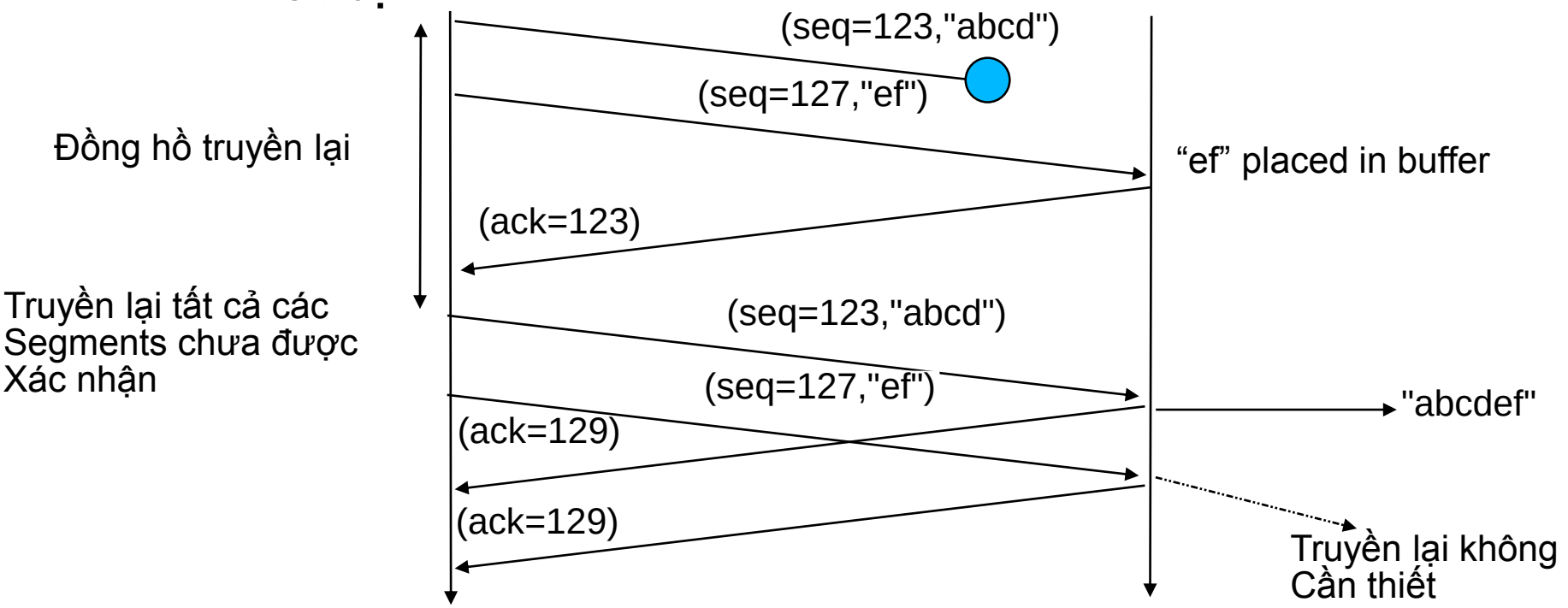
Truyền tin tin cậy

- Mỗi segment TCP chứa
  - 16 bits checksum
    - Sử dụng để phát hiện lỗi đường truyền ảnh hưởng đến payload
  - 32 bits sequence number (one byte=one seq. number)
    - Được dùng bởi bên gửi để phác họa segment đã gửi
    - Được dung bởi bên nhận để sắp đúng thứ tự các segment nhận được
    - 32 bits acknowledgement number
    - Được dùng (khi cờ ACK flag được thiết lập 1) bởi bên nhận để thông báo số thứ tự của byte mong muốn nhận tiếp theo (last byte received in sequence+1)



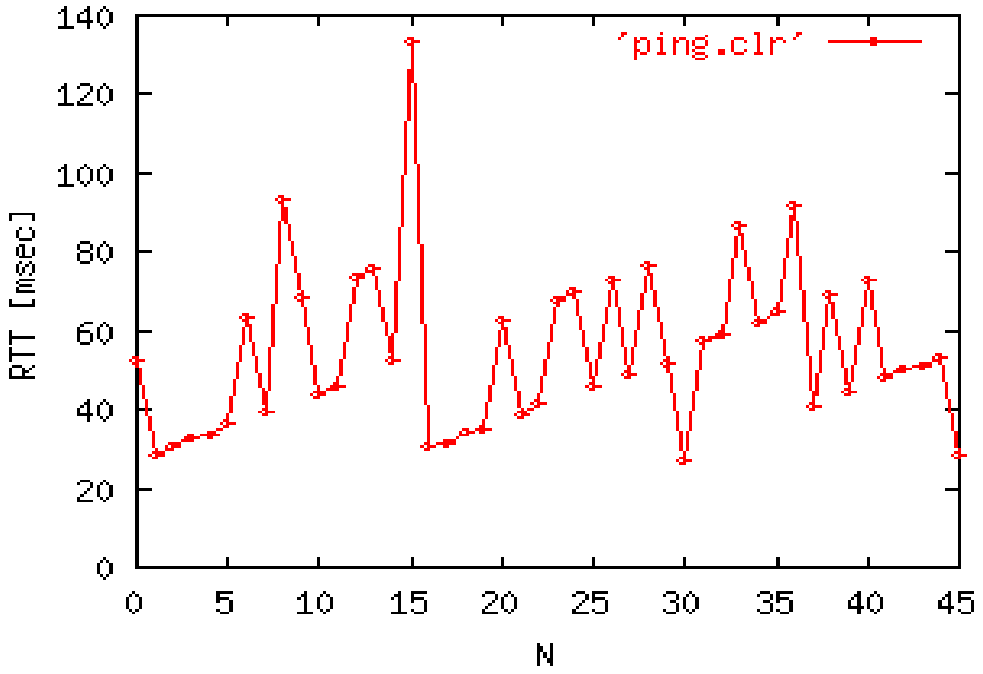
Truyền tin tin cậy

- Giải quyết mất gói tin như nào ?
  - TCP sử dụng đồng hồ truyền lại
    - Nếu đồng hồ truyền lại hết hạn, TCP thực thi go-back-n và truyền lại tất cả các segments chưa được xác nhận
    - Thông thường đồng hồ truyền lại đơn chạy tại thời điểm nhất định



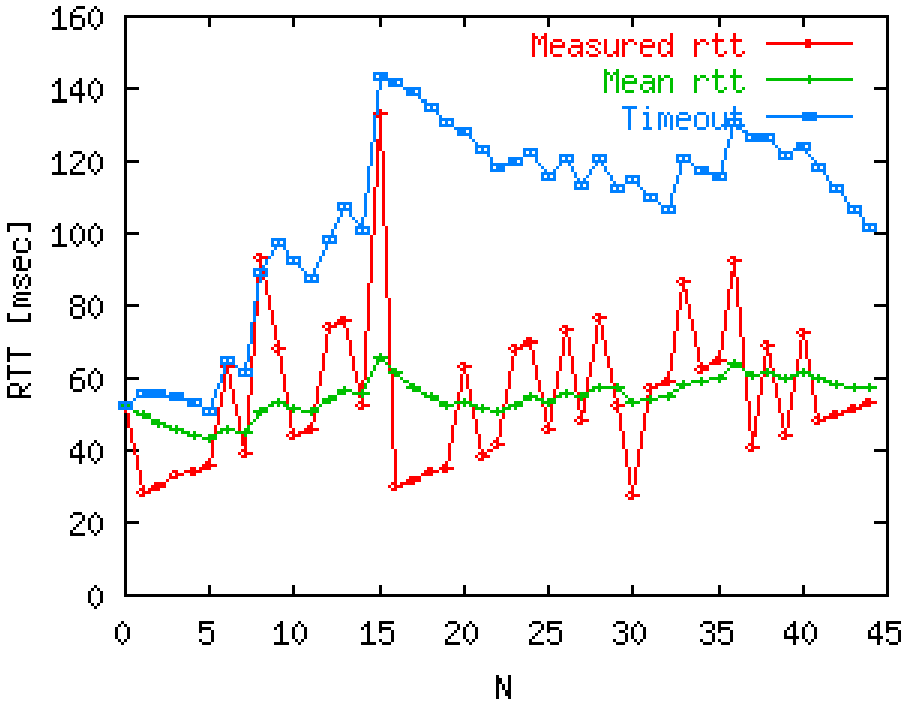
Đồng hồ truyền lại

- Làm sao tính toán ?
  - Vấn đề
    - round-trip-time có thể thay đổi thường xuyên trong suốt thời gian sống của TCP connection



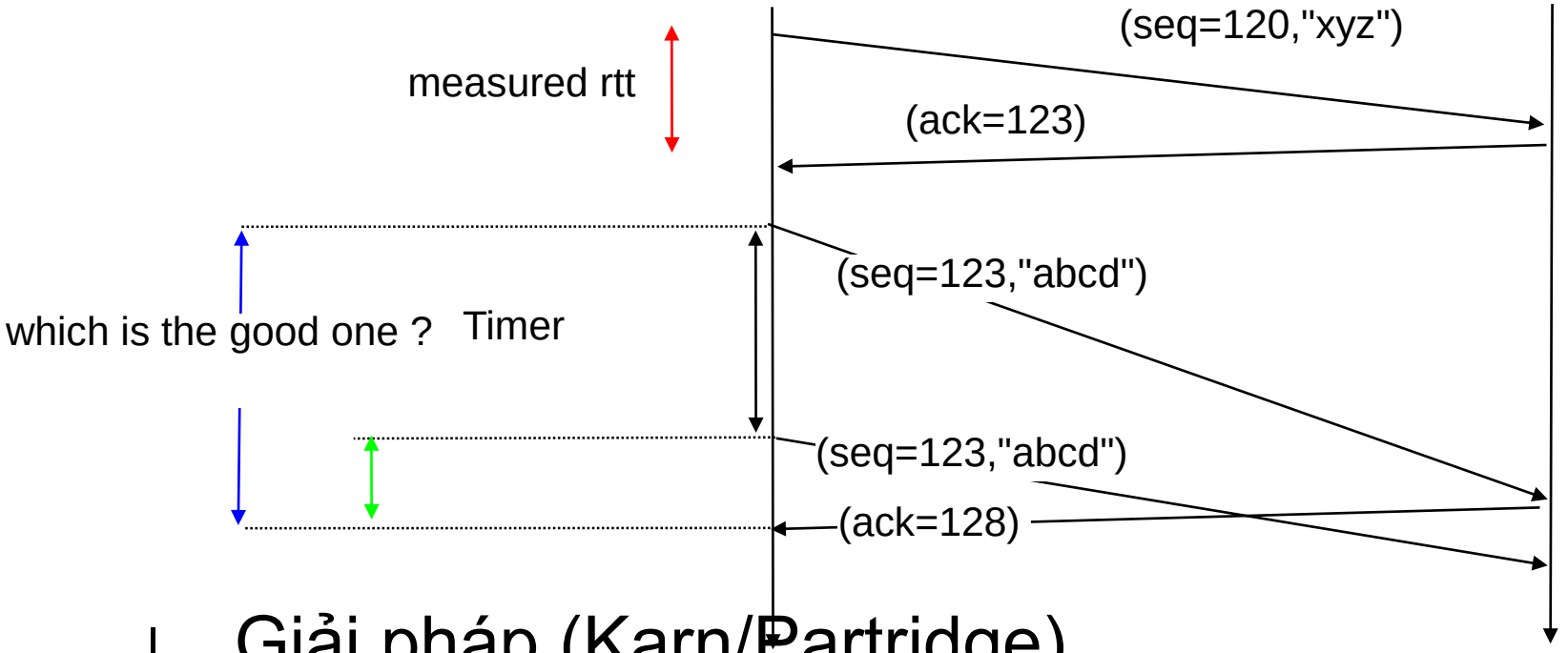
Đồng hồ truyền lại

- Đồng hồ truyền lại củaTCP
  - Mỗi đồng hồ trên một kết nối
    - timer = mean(rtt) + 4*std_dev(rtt)
    - Dự đoán giá trị trung bình
      - est_mean(rtt) = (1-〈)*est_mean(rtt) +〈 *rtt_measured
    - Dự đoán độ lệch chuẩn của rtt
      - est_std_dev=(1-®)*est_std_dev+®*|rtt_measured - est_mean(rtt)|



Dự đoánRound-trip-time

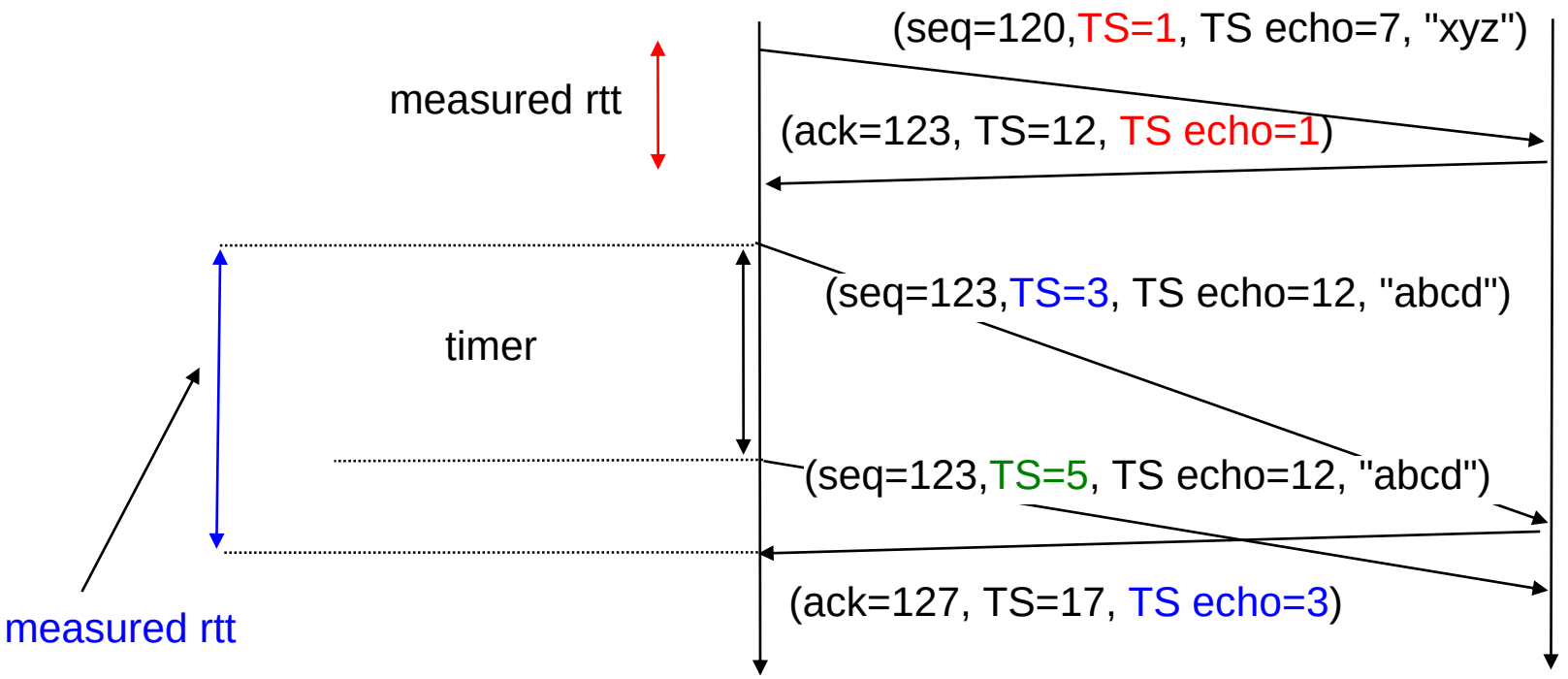
- Vấn đề
  - Làm sao tính toán rtt sau khi truyền lại ?



- Giải pháp (Karn/Partridge)
  1. Không do rtt của những segments bị truyền lại

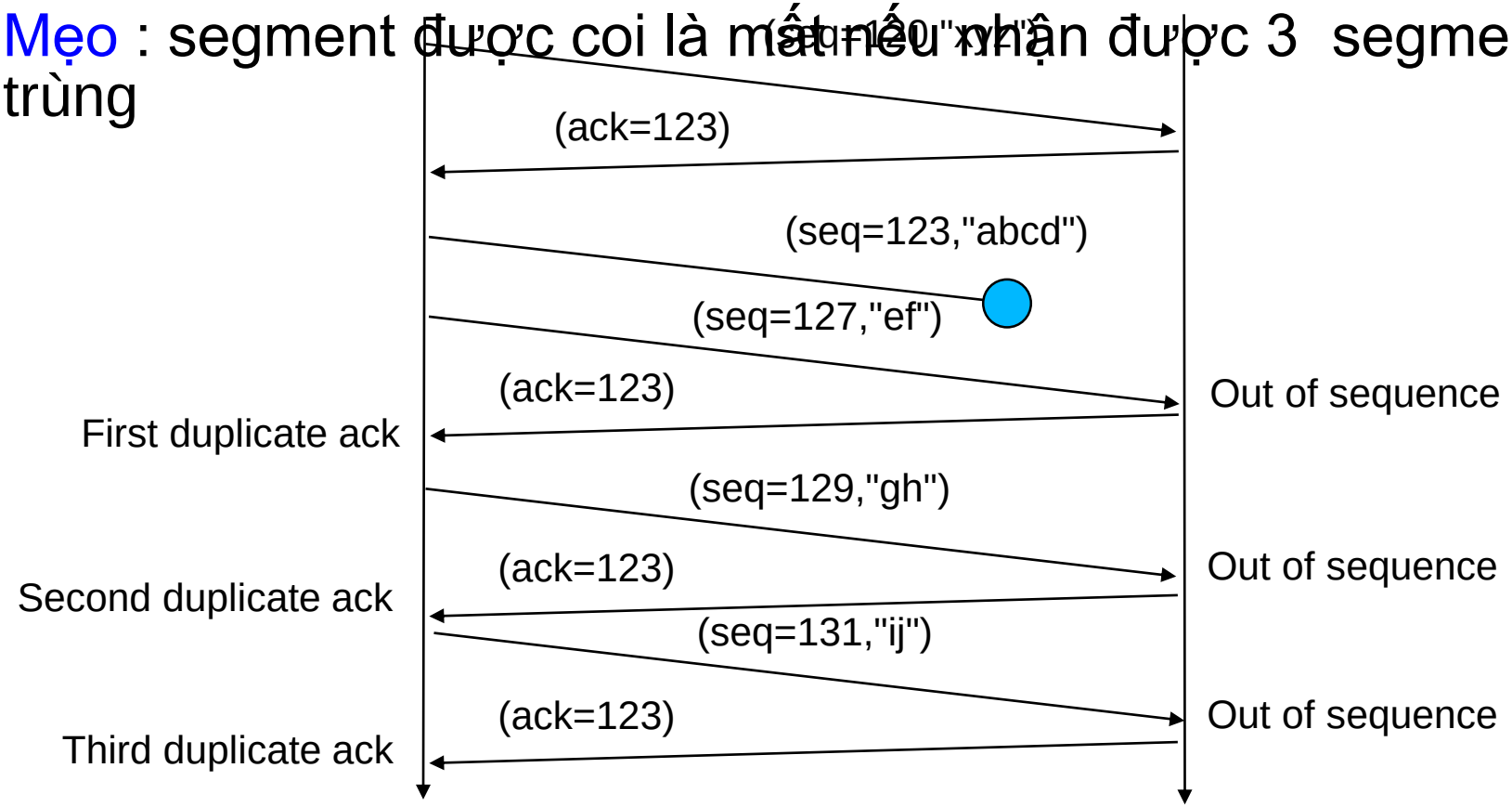
Dự đoán Round-trip-time (2)

- Cải tiến Karn/Partridge
  - Thêm nhãn thời gian vào mỗi segment được gửi
    - TS và TSEcho (RFC1323)



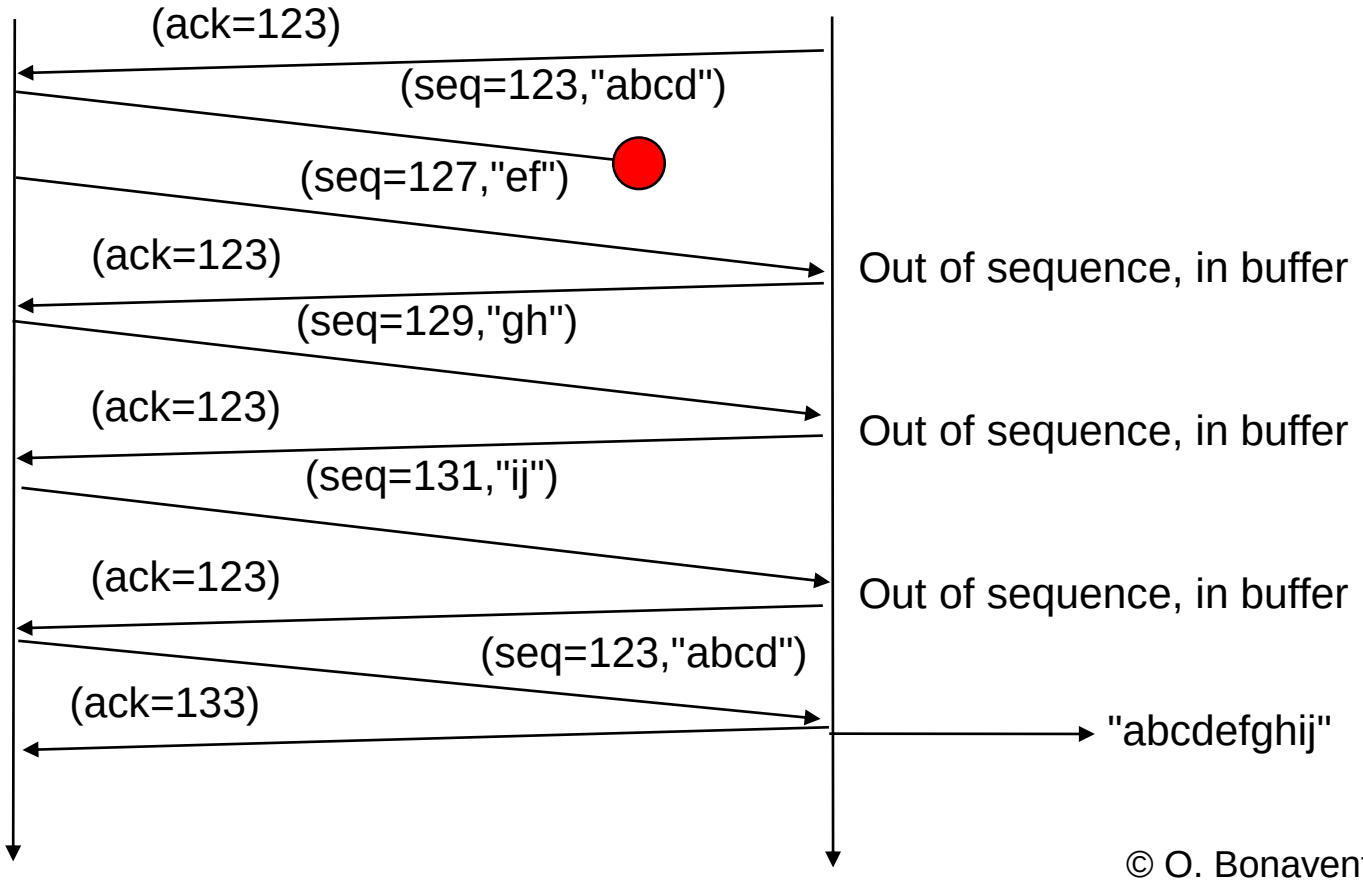
Cải tiến việc truyền tin tin cậy

- Làm sao để cải thiện cách phản ứng với mất mát gói tin ?
  - Bên nhận TCP gửi ack mỗi lần có sự mất thứ tự gói tin
    - Mẹo : segment được coi là mất nếu nhận được 3 segment trùng



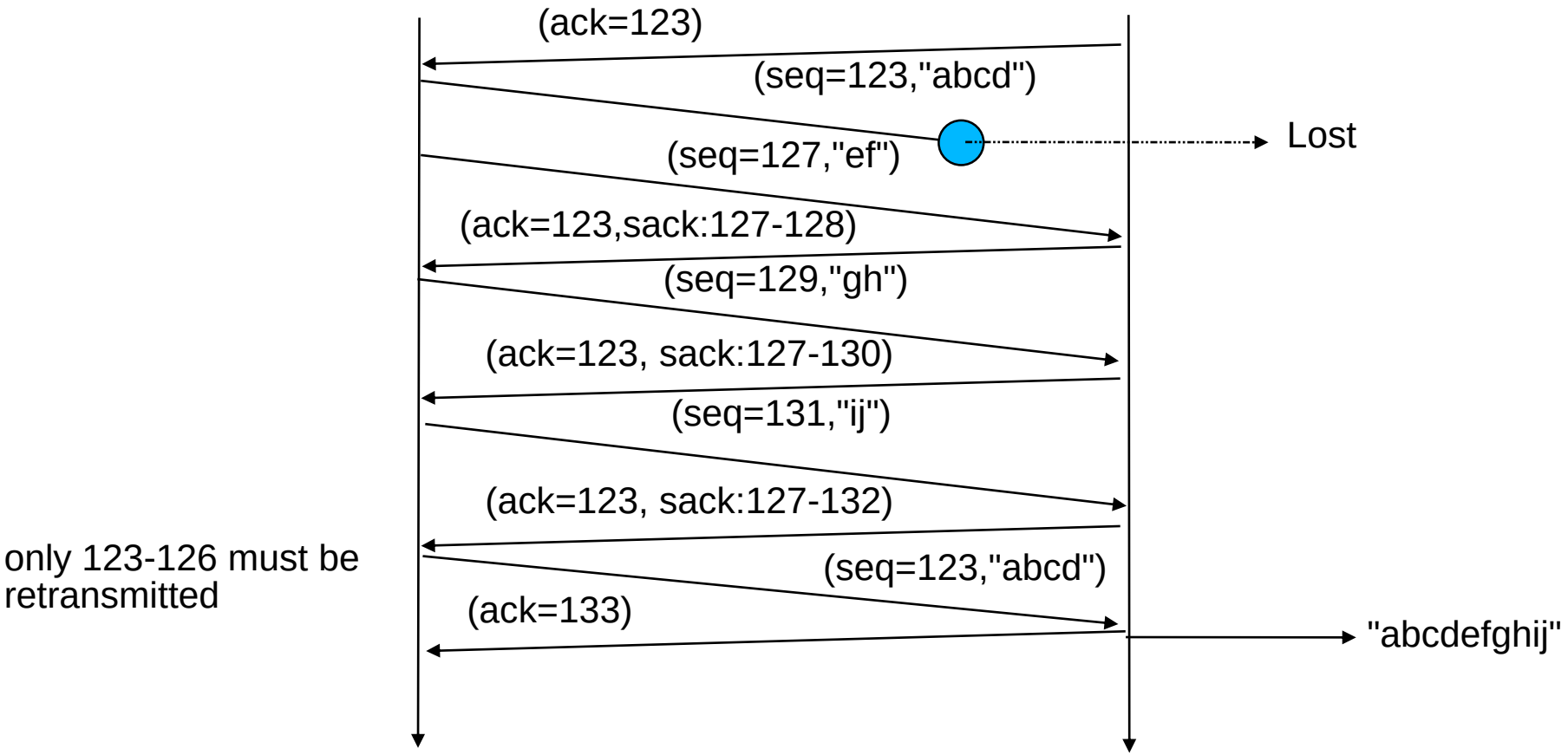
Cải thiện truyền tin tin cậy

- Làm sao truyền lại gói tin bị mất?
  - Khi nhận được 3 ack trùng, truyền lại segment không được xác nhận đầu tiên
    - Fast retransmit, được cài đặt bởi đa số các cài đặt TCP



Cải thiện truyền tin tin cậy

- Selective acknowledgement (xác nhận lựa chọn)
  - sack:[seq1-seq2];[seq3-seq4]



Gửi xác nhận

- Khi nào gửi ACK đơn thuần ?
  - Khi nhận segment dữ liệu
  - Bên trong segment dữ liệu theo hướng khác (piggyback)

- Ưu nhược điểm TCP
  - Các segment đến đúng thứ tự
    - Nếu không có ack đang chờ được gửi , khởi động đồng hồ ack timer (50 msec) và đợi cho đến khi
      - Truyền segment dữ liệu (piggyback)
      - Đồng hồ ack hết hạn
    - Nếu có ack đang chờ đợi được gửi
      - Gửi ack thuần túy ngay tức thì
  - Các segment đến không đúng thứ tự
    - Gửi ack ngay lập tức

Điều khiển luồng

- Mục đích : bảo vệ các bộ đệm gửi
- Nguyên lý
  - Gửi cửa sổ nhận trong tất cả segments
  - Biến trạng thái lưu trữ bởi các thực thể TCP
    - last_ack, swin, rwin

Last_ack=122, swin=100, rwin=4
To transmit : abcdefghijklm

Last_ack=122, swin=96, rwin=0

Last_ack=126, swin=100, rwin=0

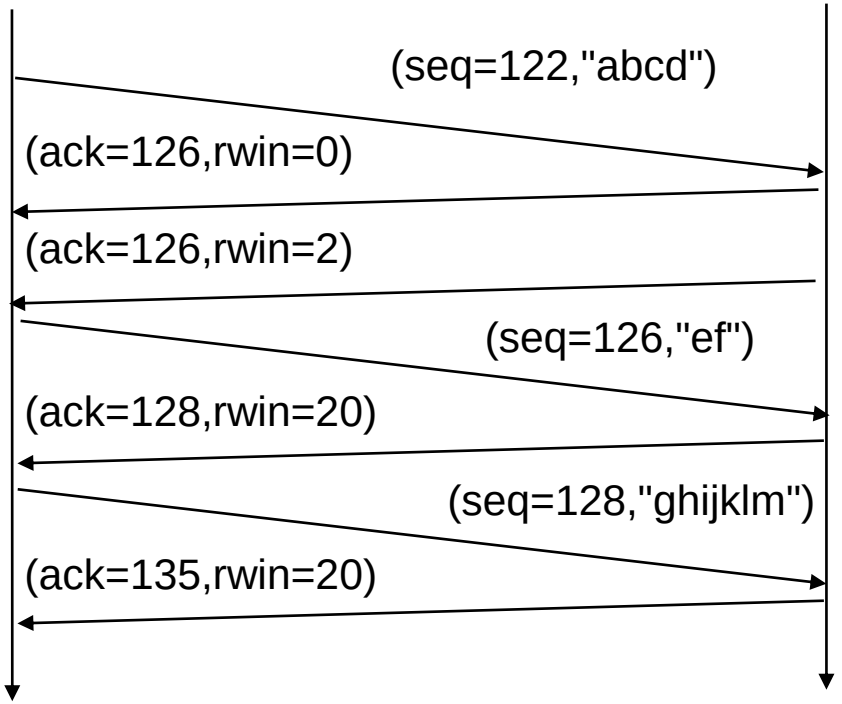
Last_ack=126, swin=100, rwin=2

Last_ack=126, swin=98, rwin=0

Last_ack=128, swin=100, rwin=20

Last_ack=128, swin=93, rwin=13

Last_ack=135, swin=100, rwin=20



Điều khiển luồng (2)

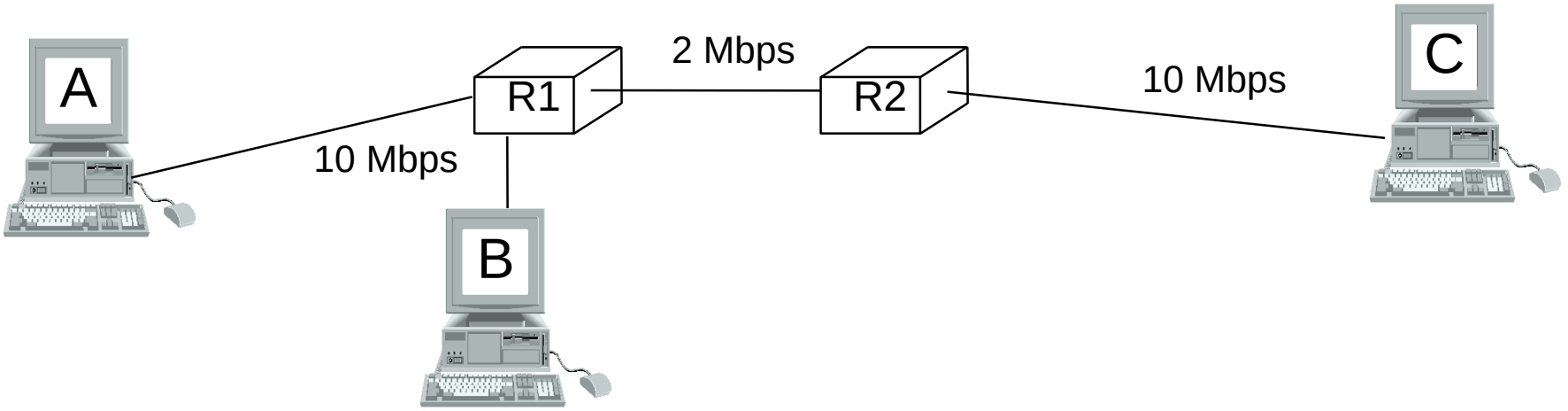
- Hạn chế
  - TCP sử dụng trường cửa sổ 16 bits trong segment header
    - Maximum window size for normal TCP : 65535 bytes
    - Extension RFC1323 for larger windows
  - Sau khi truyền cửa sổ đầy dữ liệu, bên gửi TCP phải giữ yên (idle) để nhận ack
  - Thông lượng cực đại của kết nối TCP
    - ~ window / round-trip-time

rtt / Window	1 msec	10 msec	100 msec
8 Kbytes	65.6 Mbps	6.5 Mbps	0.66 Mbps
64 Kbytes	524.3 Mbps	52.4 Mbps	5.2 Mbps

Module 3 : Transport Layer

- Basics
- Building a reliable transport layer
- UDP : a simple connectionless transport protocol
- TCP : a reliable connection oriented transport protocol
  - TCP connection establishment
  - TCP connection release
  - Reliable data transfer
  - Congestion control

Tắc nghẽn trong mạng TCP/IP

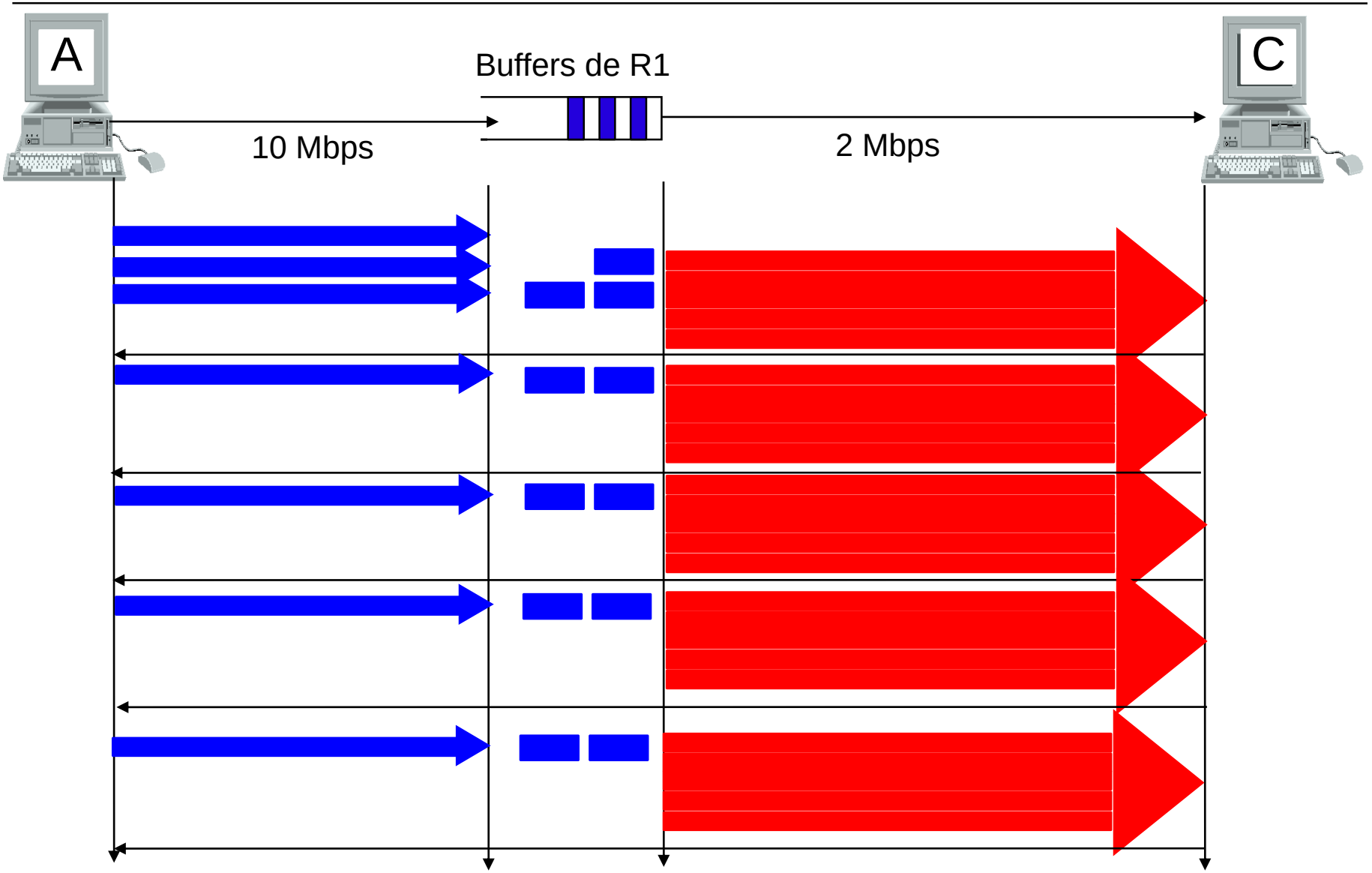


- Mạng TCP/IP không đồng nhất
  - A có thể gửi với 10 Mbps to B
  - B có thể gửi với 2 Mbps to C
- Làm sao chia sẻ mạng với nhiều máy ?
  - A và B gửi dữ liệu C đồng thời

Tắc nghẽn trong mạng TCP/IP

- Các giải pháp khả dĩ
  - Mạng chỉ ra rõ ràng bang thông giành cho mỗi máy
    - Mạng gửi thông tin điều khiển thường kì tới các máy
    - Ví dụ Available Bit Rate in ATM networks
  - Máy đo trạng thái của mạng và thích nghi bang tần với trạng thái của mạng
    - Máy phải có khả năng đo lượng tắc nghẽn trong mạng
    - Solution used by TCP in the Internet

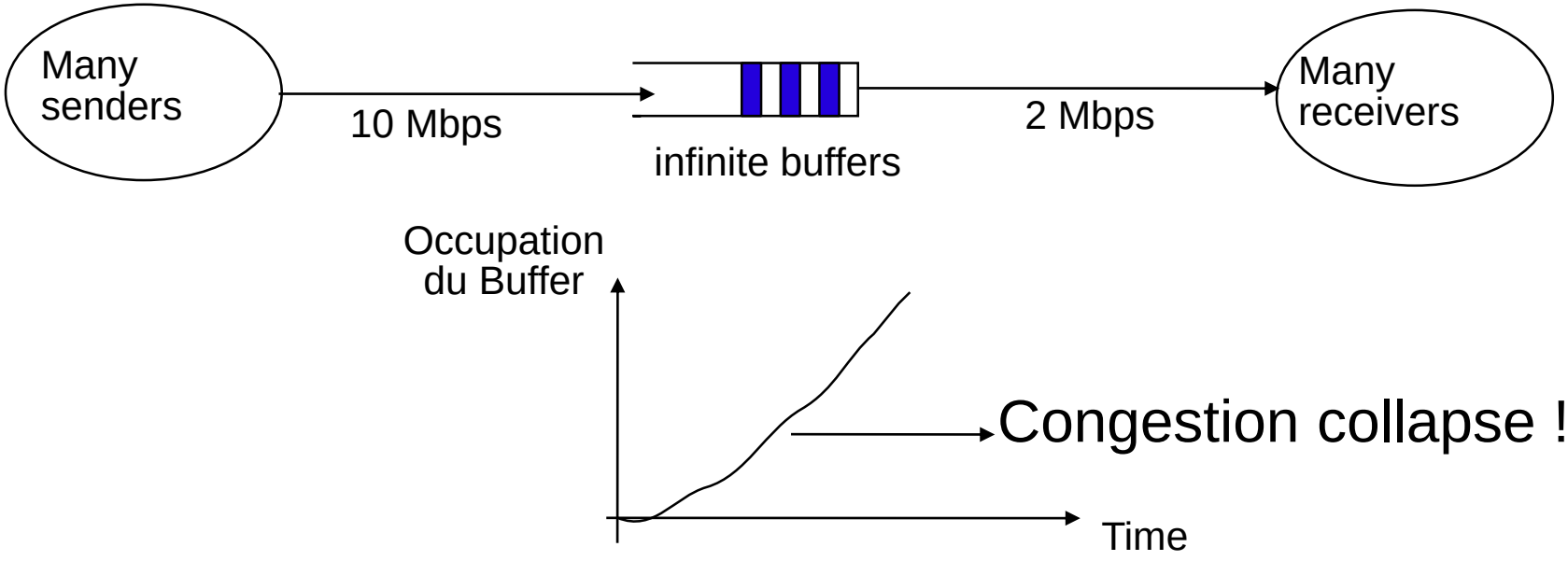
Tắc nghẽn đơn giản



- TCP self-clocking

Tắc nghẽn đơn giản

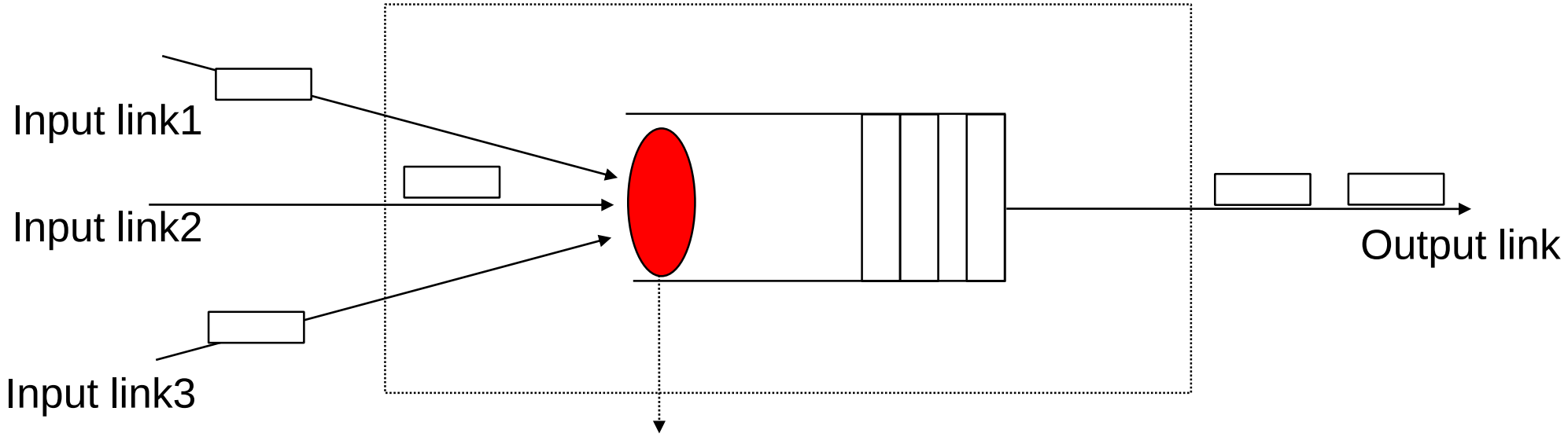
- TCP self-clocking
  - Có thể đủ khi kết nối TCP đơn sử dụng bằng tần thấp nêu bộ đệm trung gian có thể lưu trữ cửa sổ đầy segment
  - Điều gì xảy ra nếu nhiều kết nối TCP chia sẻ một kết nối



Điều khiển tắc nghẽn TCP

- Làm sao thích nghi kết nối TCP với trạng thái mạng ?
  - Làm sao đo trạng thái tắc nghẽn hiện tại ?
    - TCP sử dụng tỷ lệ mất gói tin trên router làm chỉ báo tắc nghẽn
      - Điều này đúng trong đa số các môi trường trừ mạng không dây nơi mà lỗi truyền có thể làm mất gói tin
  - Thích nghi băng thông của kết nối TCP
    - TCP thích nghi tốc độ truyền của nó bằng cách sử dụng của sổ tắc nghẽn (cwnd) được điều khiển bởi phía gửi dựa trên trạng thái tắc nghẽn hiện tại

Router đơn giản



Thuật toán điều khiển bộ nhớ đệm
Buffer acceptance algorithm
Khi gói tin đến bộ đệm ra, phải quyết định chấp nhận hay loại bỏ gói tin

Điều khiển tắc nghẽn TCP

- Làm sao để lựa chọn khi kết nối bắt đầu ?
  - Tránh tắc nghẽn từ từ

Slowstart

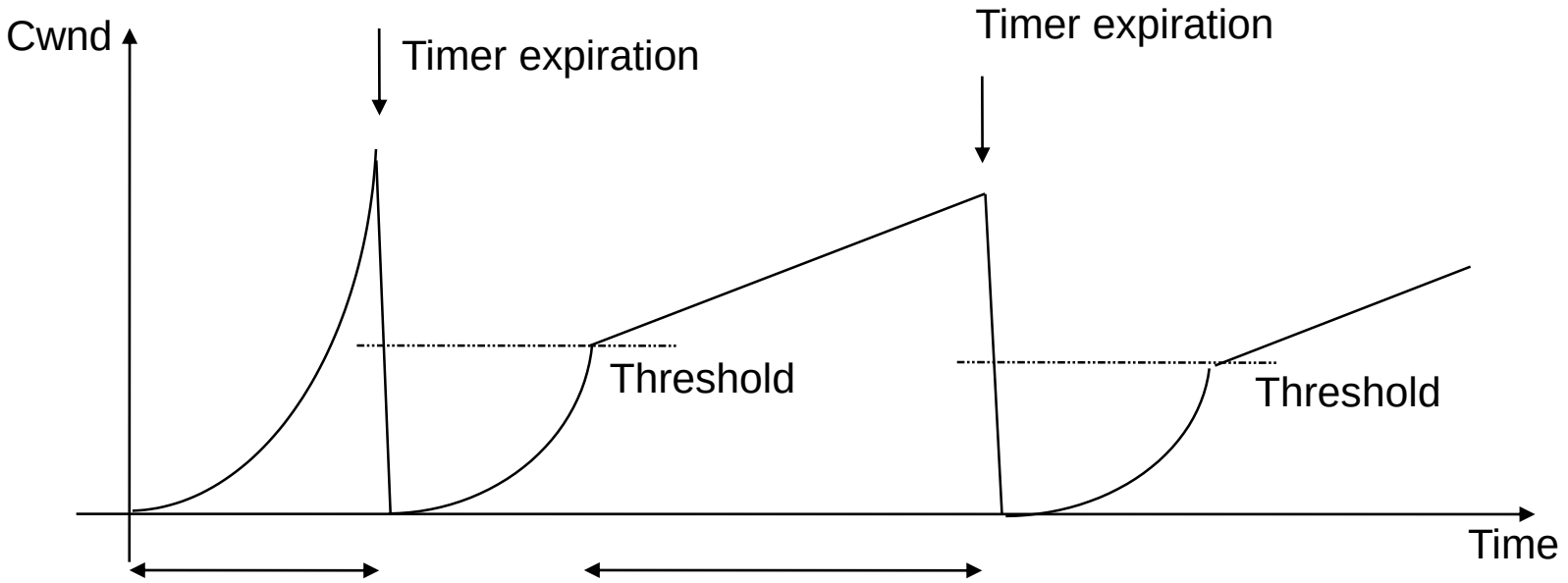
Congestion avoidance

```
Initialisation :
cwnd = MSS;
ssthresh= swin;

Ack reception :
if (network not congested ) // no segment losses
{
  if (cwnd < ssthresh)
   {  // increase quickly cwnd
      // double cwnd  every rtt
      cwnd = cwnd+ MSS;
   }
   else
   {  // increase slowly cwnd
       // increase cwnd by one mss every rtt
       cwnd = cwnd+ mss*(mss/cwnd);
   }
}
```

Điều khiển tắc nghẽn TCP

Ví dụ



Slow-start
exponential increase of cwnd

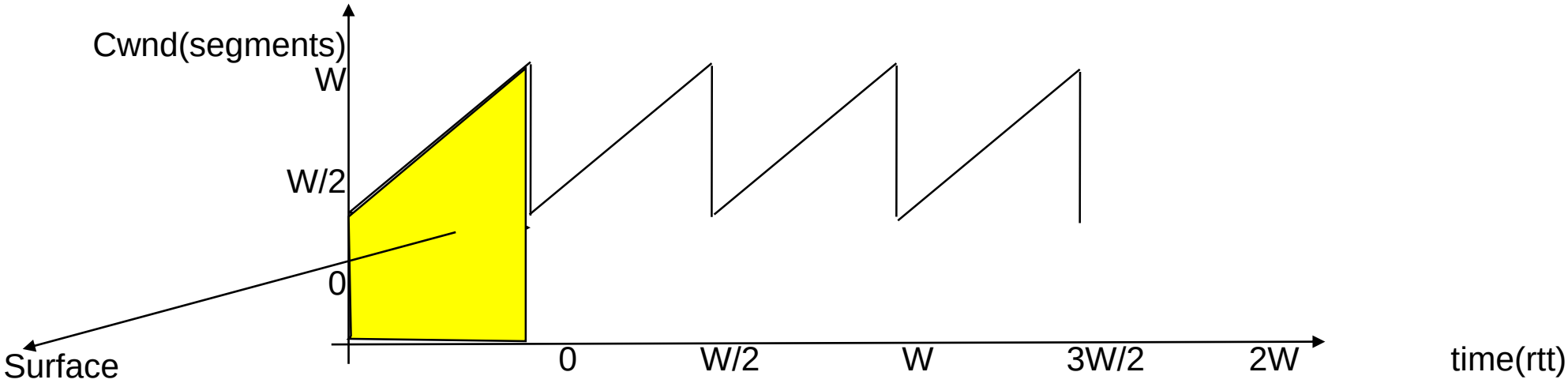
Congestion avoidance
linear increase of cwnd

Điều khiển tắc nghẽnTCP

- Làm sao phản ứng với mất gói tin ?
  - Hai kiểu giảm cấp số nhân (multiplicative decrease)
  - Mất mát nghiêm trọng [several lost segments]
    - Đợi cho đến khi hết hạn đồng hồ truyền lại
      - sstresh=ssthresh/2
      - retransmit lost segments
      - slow-start (until cwnd=ssthresh)
      - congestion avoidance
  - Mất mát đơn lẻ [a single lost segment]
    - fast retransmit can recover from lost segment
    - If a single segment was lost : fast recovery
      - retransmit lost segment
      - sstresh=cwnd / 2
      - cwnd=ssthresh ; congestion avoidance

Điều khiển tắc nghẽn TCP

- Mô hình đơn giản hóa
  - Giả sử rằng mất mát gói tin là thường kì và với xác suất với xác suất 1/p



$$(\frac{W}{2})^2 + \frac{1}{2}(\frac{W}{2})^2 = \frac{1}{p}$$

It can be shown that the throughput of a TCP connection can be approximated by :

$$BW < Min[\frac{Window}{RTT}, (\frac{MSS}{RTT})\frac{k}{\sqrt{p}}]$$

Maximum throughput without losses

Throughput with losses/congestion