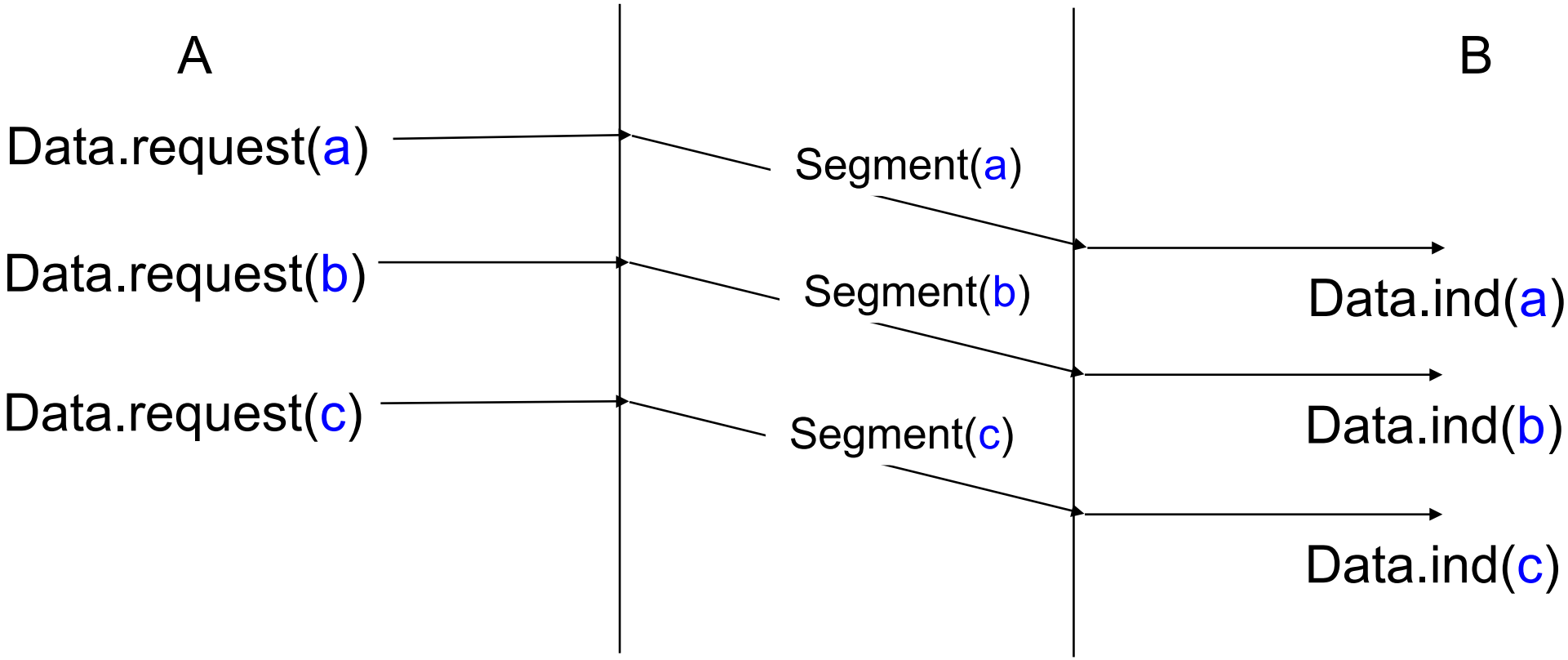
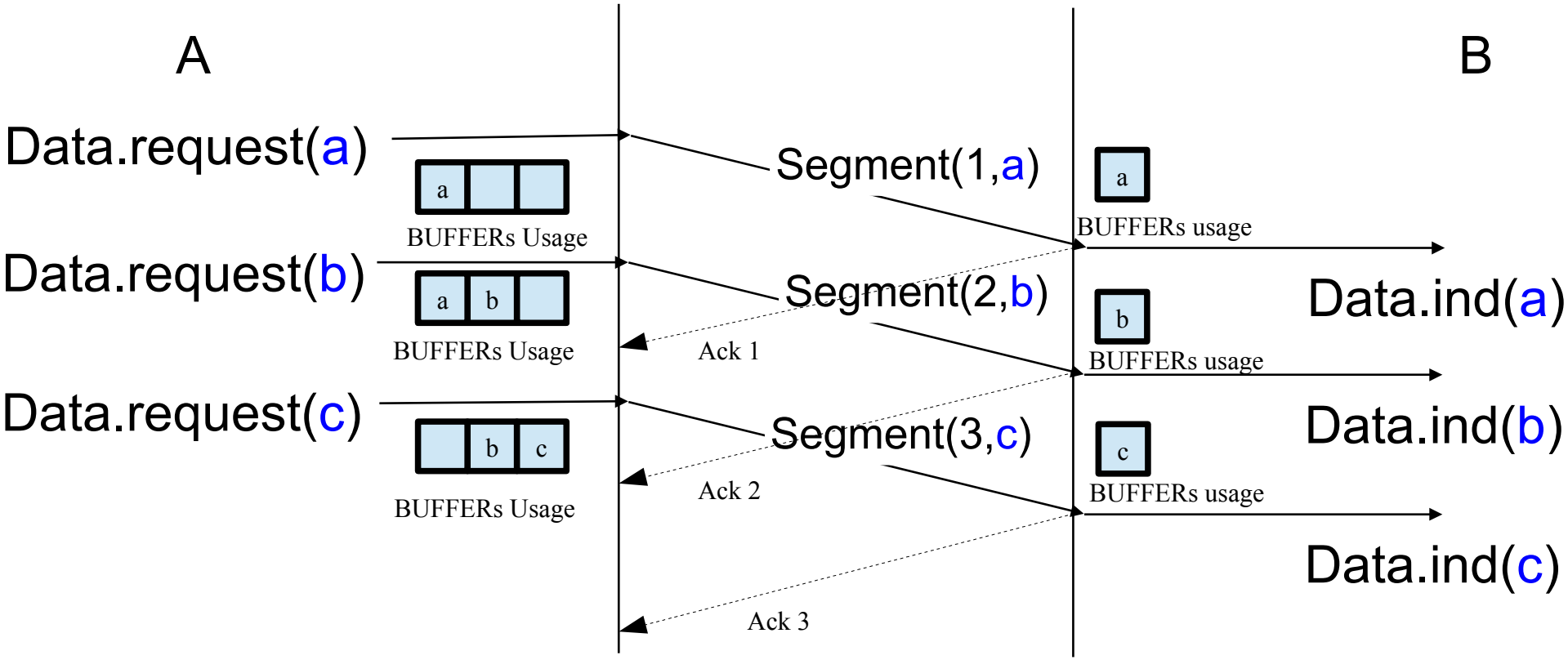
How to improve the alternating bit protocol ? (2)

- Modifications to alternating bit protocol
  - Sequence numbers inside each segment
    - Each data segment contains its own sequence number
    - Each control segment indicates the sequence number of the data segment being acknowledged (OK/NAK)
  - Sender
    - Needs enough buffers to store the data segments that have not yet been acknowledged to be able to retransmit them if required
  - Receiver
    - Needs enough buffers to store the out-of-sequence segments, but this is optional (see Go-Back-N).

Remember protocol 1 ? It was pipelined



How to improve the alternating bit protocol ? (3)

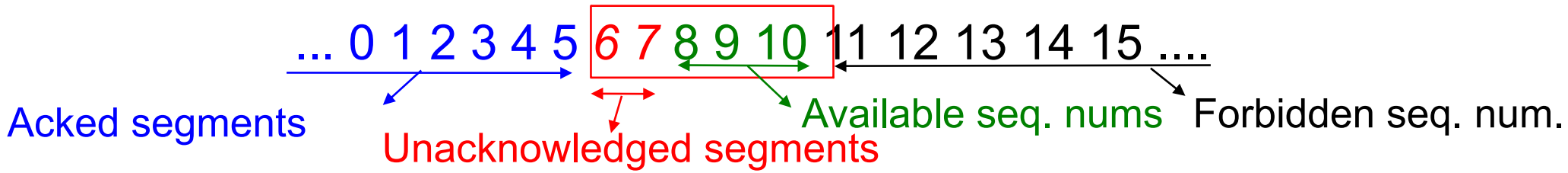


We want the best of two worlds : pipelining + feedback loop.

- Issue ?
  - What happens if the receiver is much slower than the sender ?
    - e.g. receiver can process one segment per second while sender is producing 10 segments per second ?
    - *How to avoid an overflow of the receiver's buffers ?*

Sliding window

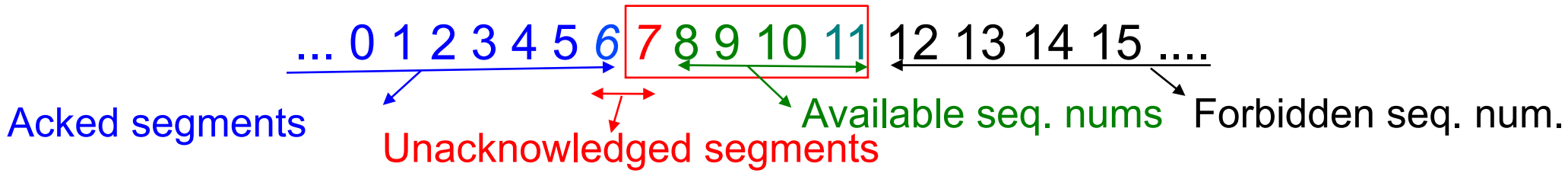
- Principle
  - Sender keeps a list of all the segments that it is allowed to send
    - `sending_window`



  - Receiver also maintains a receiving window with the list of acceptable sequence number
    - receiving_window
  - Sender and receiver must use compatible windows
    - sending_window ≤ receiving window
      - For example, window size is a constant for a given protocol or negotiated during connection establishment phase

Sliding window

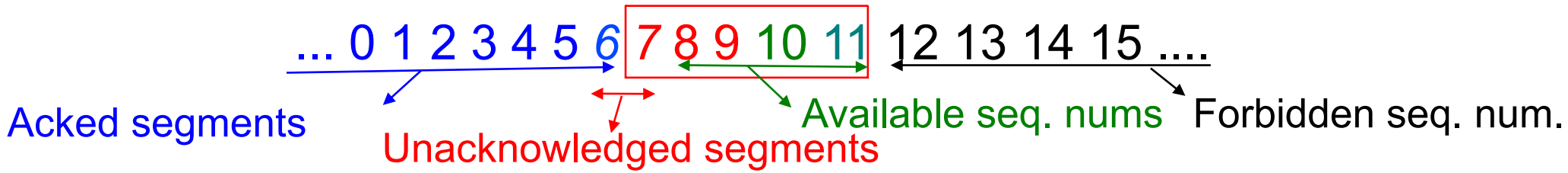
- Principle
  - Sender keeps a list of all the segments that it is allowed to send
    - `sending_window`



  - Receiver also maintains a receiving window with the list of acceptable sequence number
    - receiving_window
  - Sender and receiver must use compatible windows
    - sending_window ≤ receiving window
      - For example, window size is a constant for a given protocol or negotiated during connection establishment phase

Sliding window

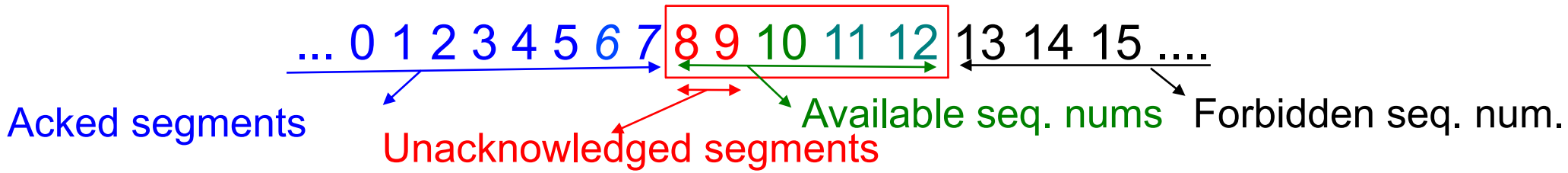
- Principle
  - Sender keeps a list of all the segments that it is allowed to send
    - `sending_window`



  - Receiver also maintains a receiving window with the list of acceptable sequence number
    - receiving_window
  - Sender and receiver must use compatible windows
    - sending_window ≤ receiving window
      - For example, window size is a constant for a given protocol or negotiated during connection establishment phase

Sliding window

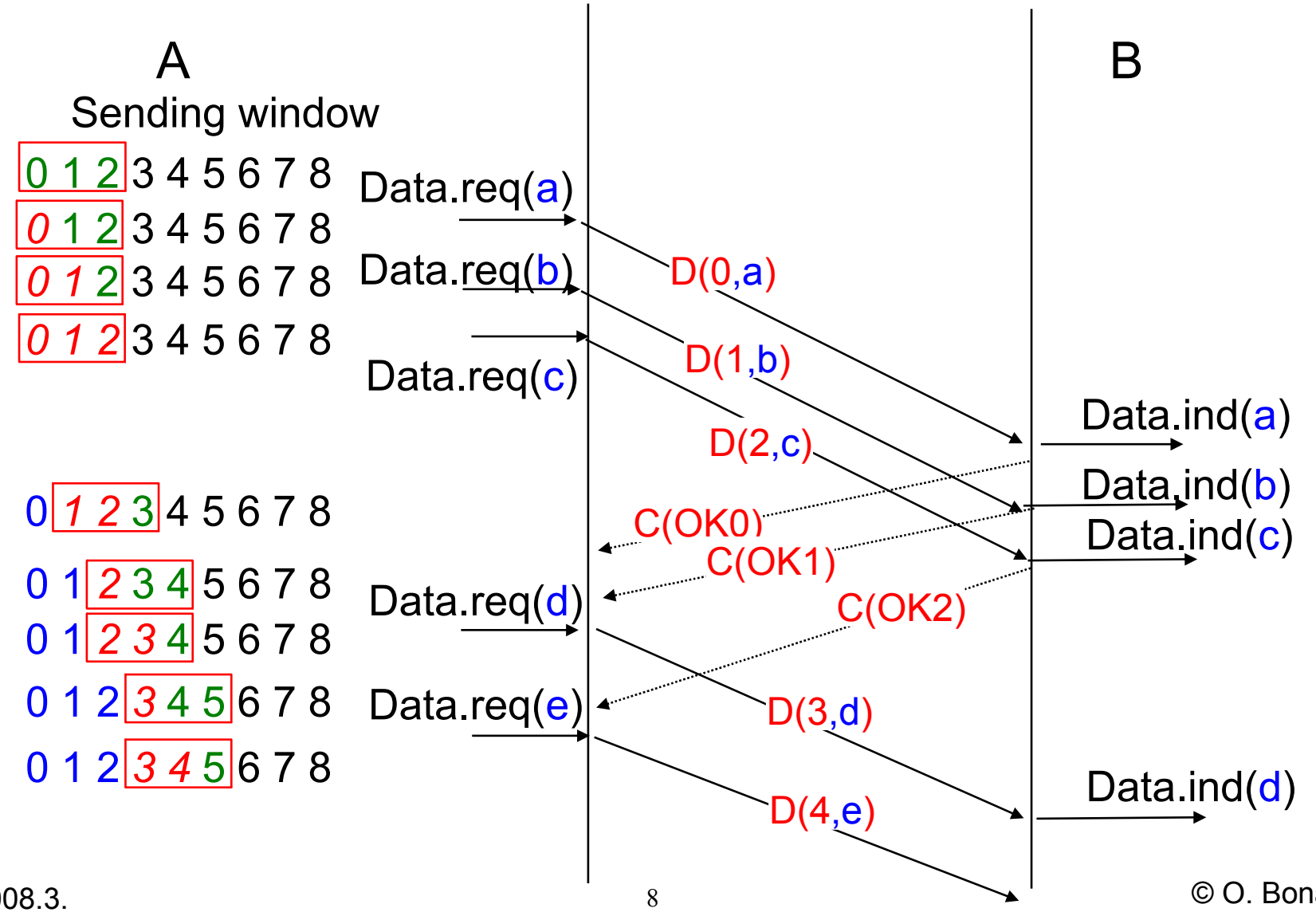
- Principle
  - Sender keeps a list of all the segments that it is allowed to send
    - `sending_window`



  - Receiver also maintains a receiving window with the list of acceptable sequence number
    - receiving_window
  - Sender and receiver must use compatible windows
    - sending_window ≤ receiving window
      - For example, window size is a constant for a given protocol or negotiated during connection establishment phase

Sliding windows : example

- Sending and receiving window : 3 segments



Encoding sequence numbers

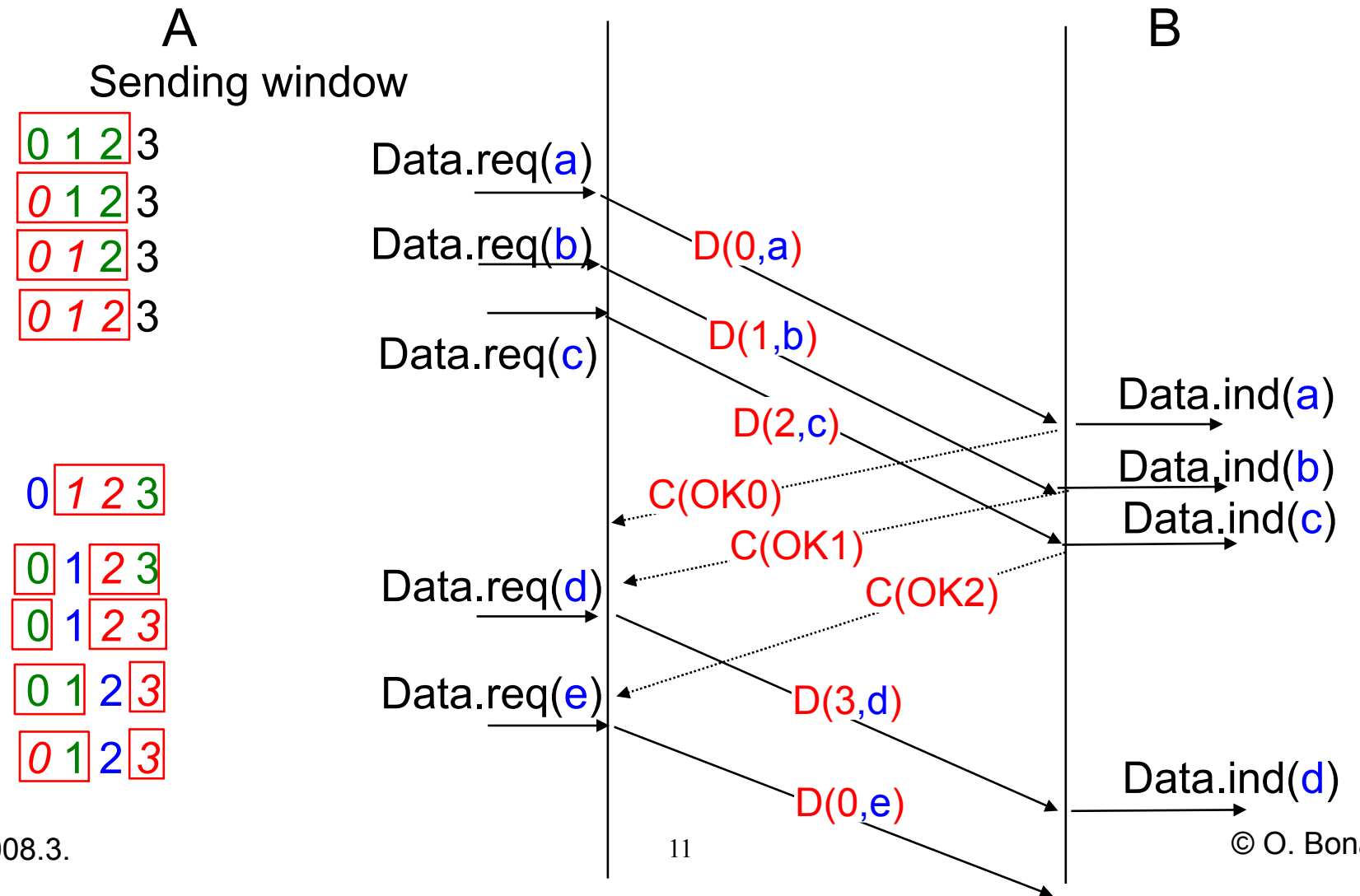
- Problem
  - How many distinct sequence numbers can we have with a N bits sequence number ?
    - N bits means 2^N different sequence numbers
- Solution ?

Encoding sequence numbers

- Problem
  - How many distinct sequence numbers can we have with a N bits sequence number ?
    - N bits means 2^N different sequence numbers
- Solution
  - place inside each transmitted segment its sequence number modulo 2^N
  - The same sequence number will be used for several different segments
    - be careful, this could cause problems...
  - Sliding window
    - List of consecutive sequence numbers (modulo 2^N) that the sender is allowed to transmit

Sliding window : second example

- 3 segments sending and receiving window
  - Sequence number encoded as 2 bits field



Reliable transfer with a sliding window

- How to provide a reliable data transfer with a sliding window
  - How to react upon reception of a control segment ?
  - Sender's and receiver's behaviours
- Basic solutions, using ARQ* methods
  - Go-Back-N
    - simple implementation, in particular on receiving side
    - throughput will be limited when losses occur
  - Selective Repeat
    - more difficult from an implementation viewpoint
    - throughput can remain high when limited losses occur

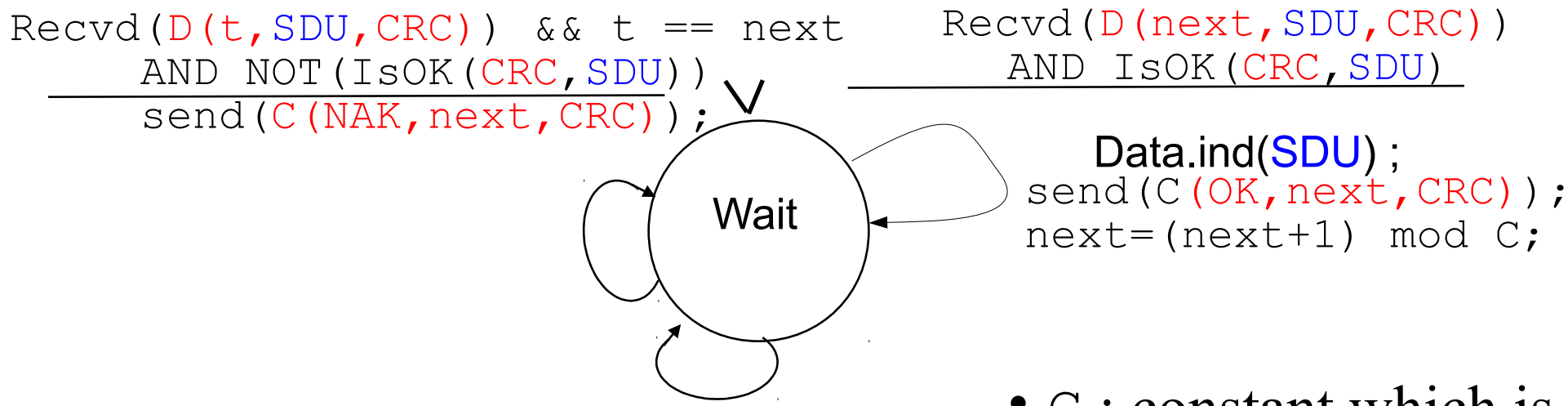
* Automatic Repeat reQuest see http://en.wikipedia.org/wiki/Automatic_repeat_request

GO-BACK-N

- Principle
  - Receiver must be as simple as possible
  - Receiver
    - Only accepts consecutive in-sequence data segments
    - Meaning of control segments
      - Upon reception of data segment
        - OKX means that all data segments, up to and including X have been received correctly
        - NAKX means that the data segment whose sequence number is X contained an error.
  - Sender
    - Relies on a retransmission timer to detect segment losses
    - Upon expiration of retransmission time or arrival of a NAK segment : retransmit all the unacknowledged data segments
      - the sender may thus retransmit a segment that was already received correctly but out-of-sequence at destination

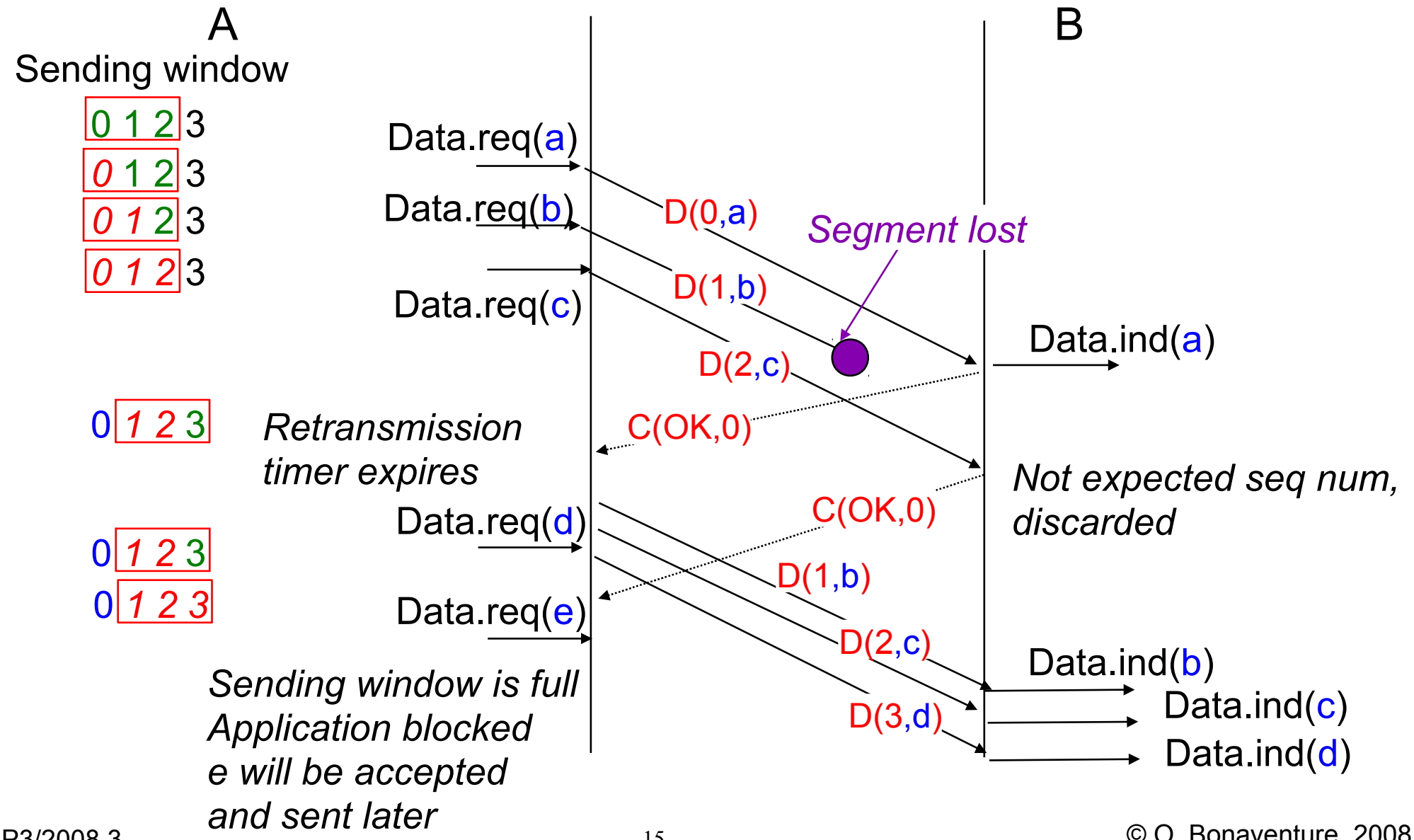
Go-Back-N : Receiver

- State variable
  - `next` : sequence number of expected data segment



- C : constant which is the total number of available sequence numbers

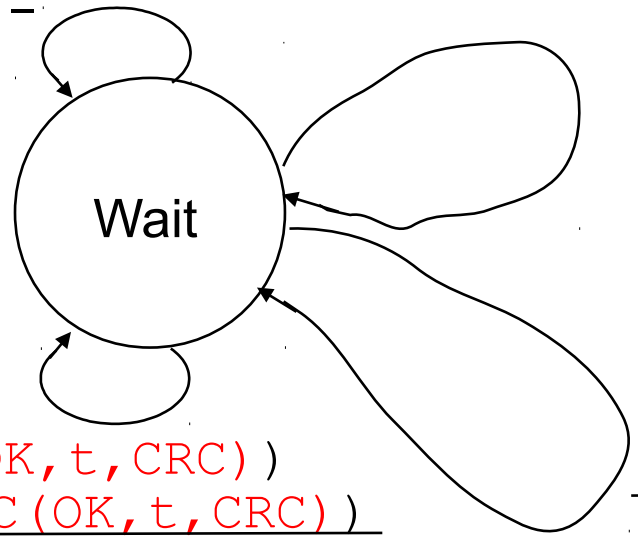
Go-Back-N : Example (2)



Go-Back-N : Sender

- State variables
  - `base` : sequence number of oldest unacked data segment
  - `seq` : first available sequence number
  - `w` : size of sending window

```
Recvd(C(?,?,CRC))
and NOT( CRCOK(C(?,?,CRC)))
```



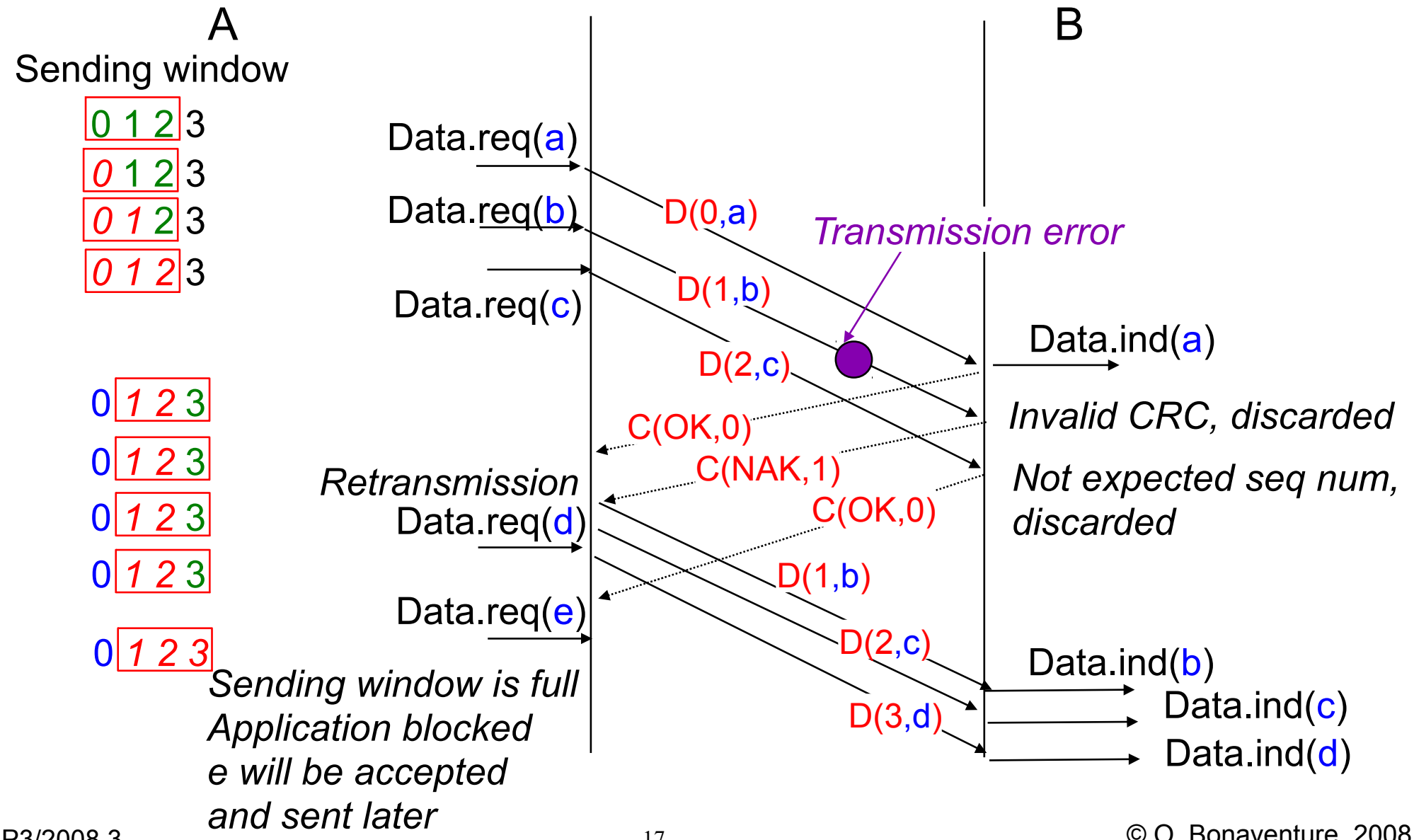

```
Recvd(C(OK,t,CRC))
and CRCOK(C(OK,t,CRC))
base=(t+1);
if (base==seq)
{ cancel_timer();}
else
{ restart_timer(); }
```

Data.req(SDU)
AND (seq < (base+w))

```
send(D(seq,SDU,CRC));
if (seq==base) {
start_timer ;
}
seq=(seq+1)mod C ;
```

```
[ Recvd(C(NAK,?,CRC))
and CRCOK(C(NAK,?,CRC))]
or timer expires
for (i=base;i<seq; i=i+1)
{ send(D(i,SDU,CRC)); }
restart_timer();
```

Go-Back-N : Example



Go Back-N : a bit of practice

Consider a go-back-n sender and a go-back receiver that are directly connected with a 10 Mbps link that has a propagation delay of 100 milliseconds. Assume that the retransmission timer is set to 3 seconds. If the window has a length of 4 segments, draw a time-sequence diagram showing the transmission of 10 segments (each segment contains 10000 bits):

- when there are no losses
- when the third and seventh segments are lost
- when the second, fourth, sixth, eighth, ... acknowledgements are lost
- when the third and fourth data segments are reordered (i.e. the fourth arrives before the third)

Go Back-N : a bit of practice

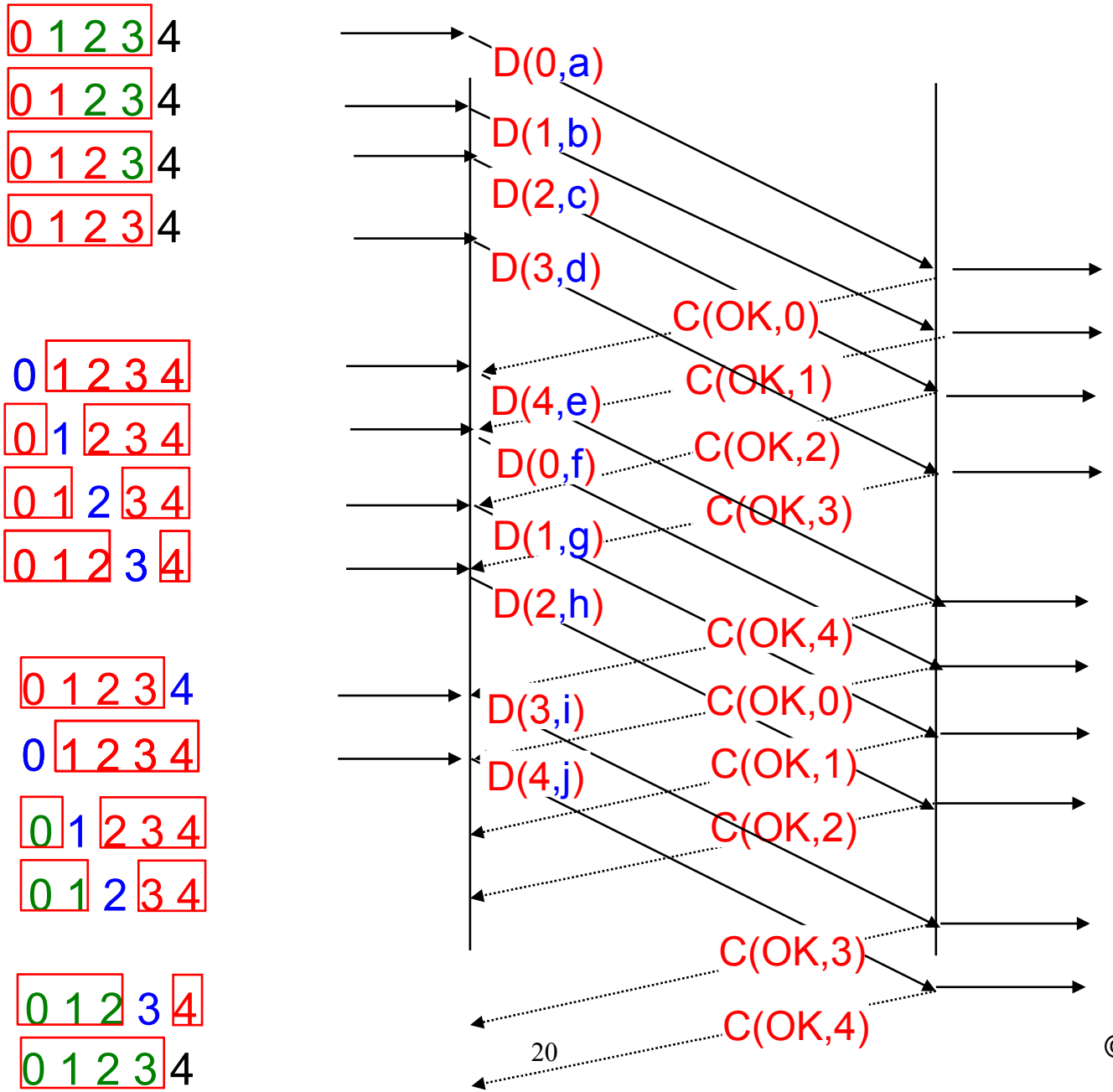
Consider the following situation. A go-back-n sender with Sequence number encoded as 2 bits has a sending windows of 4 segments.

The sender sent a full window of data segments. All the Segments have been received correctly and in-order by the receiver, but all the returned acknowledgements have been lost.

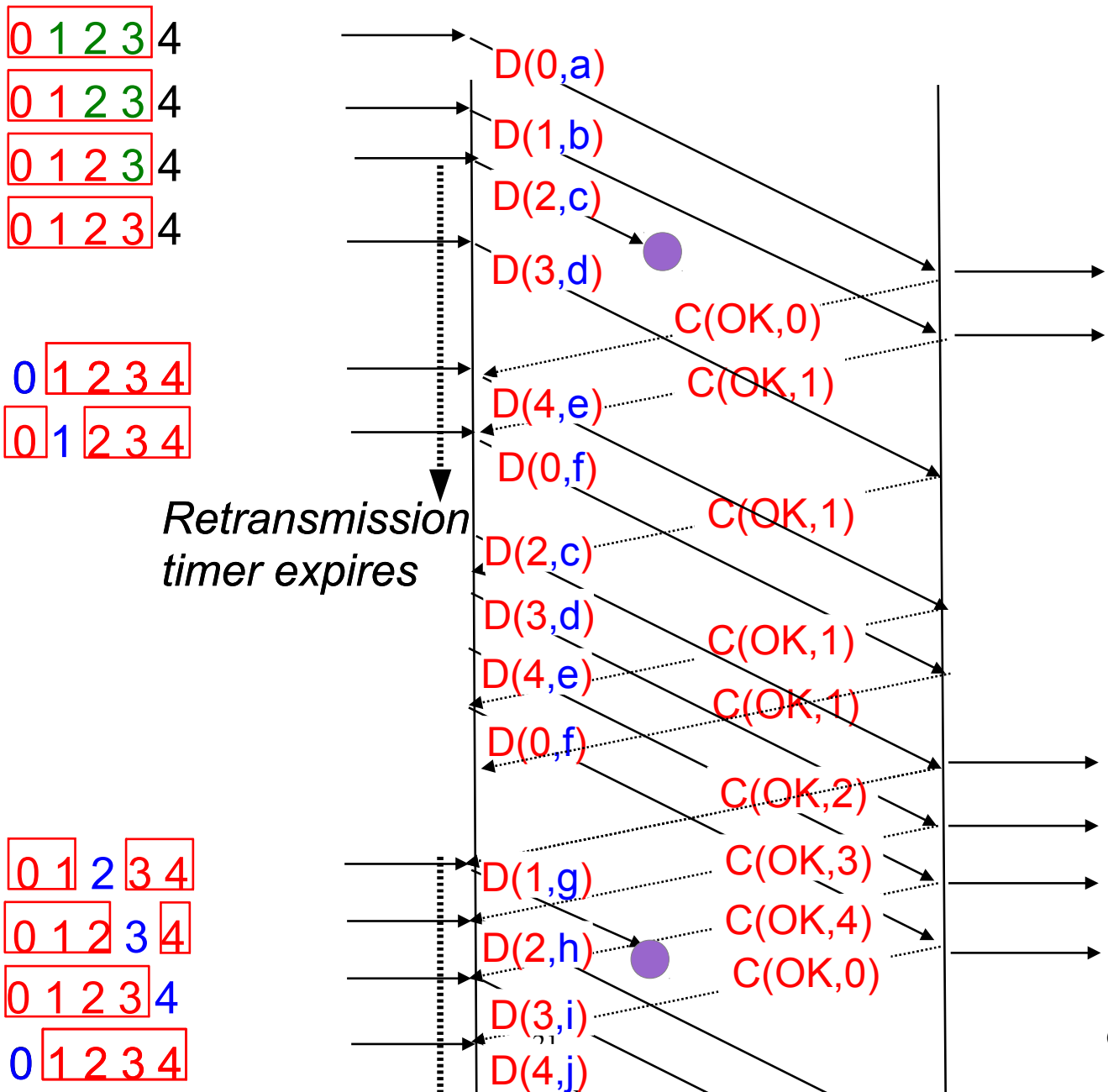
Show by using a time sequence diagram (e.g. by considering a window of four segments) what problem can happens in this case.

Can you fix the problem on the go-back-n sender ?

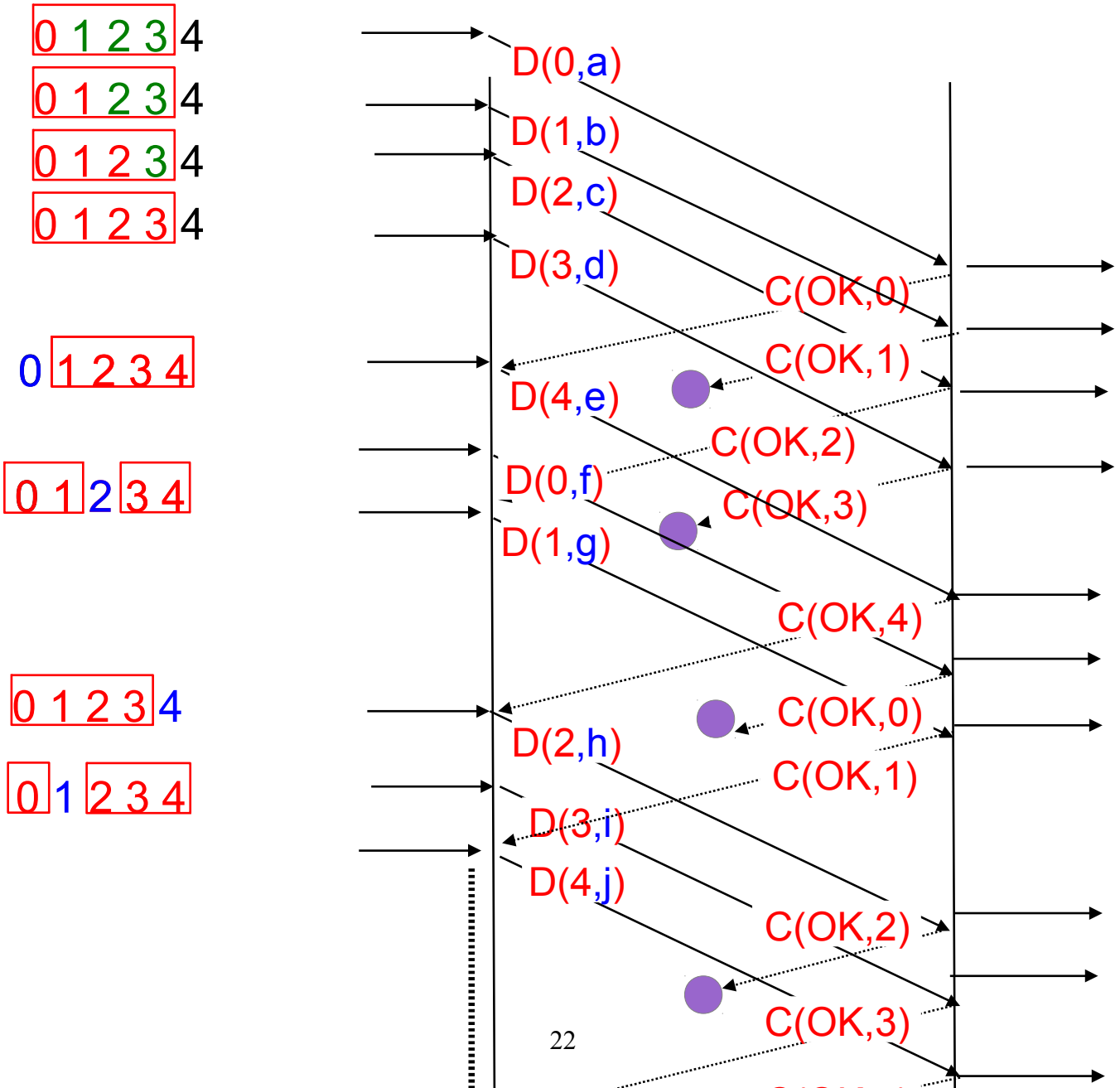
Go-Back-N : 10 segments with no loss



Go-Back-N : 10 segments, 3rd and 7th lost



Go-Back-N : 10 segments, 2,4,6... acks lost



Go-Back-N : 10 segments, invert 3 and 4