Internet, Principes et Protocoles (IPP)

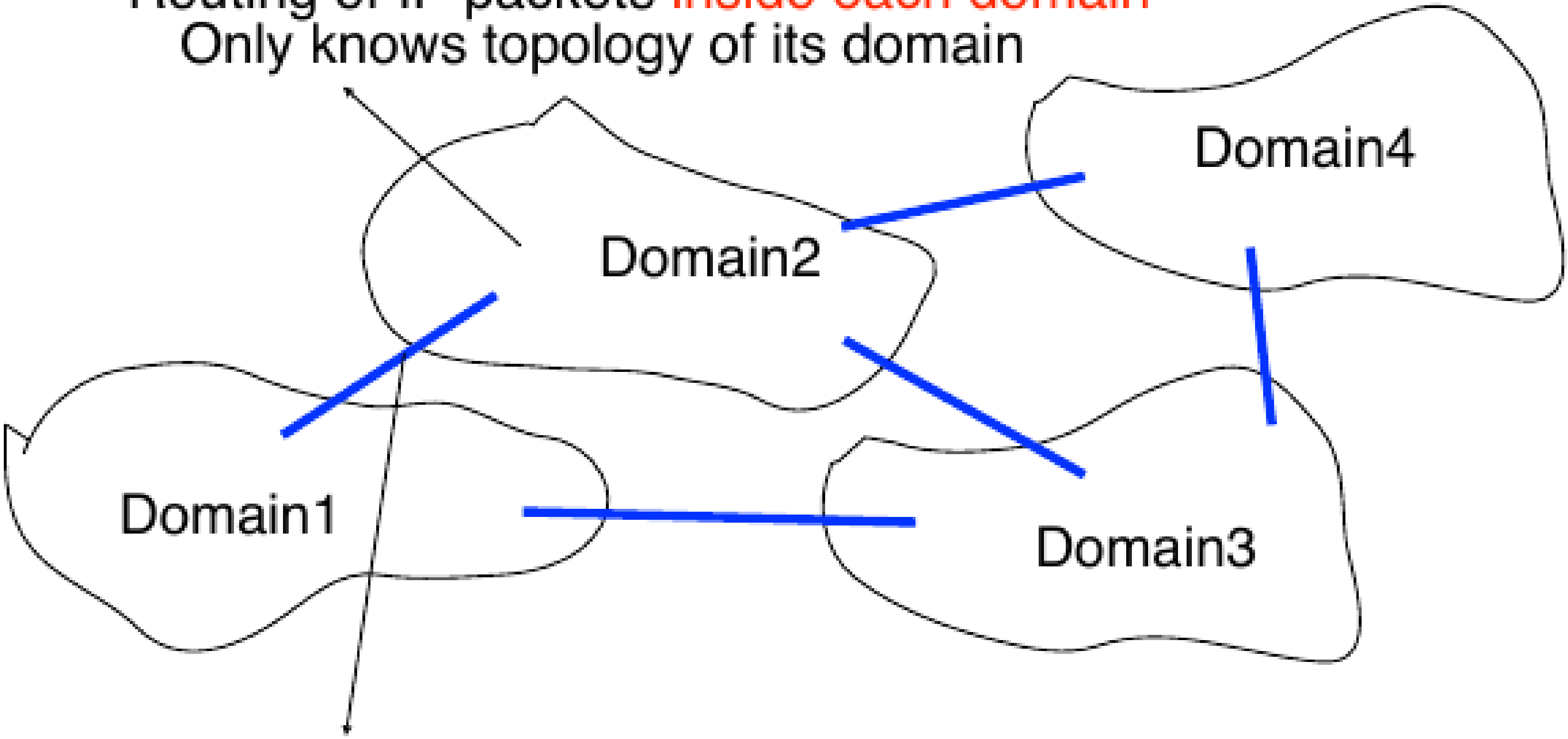
Table of Contents

- Recap
- Transport Layer
  - UDP
  - TCP

Interior Gateway Protocol (IGP)

Routing of IP packets inside each domain

Only knows topology of its domain



Exterior Gateway Protocol (EGP)

Routing of IP packets between domains

Each domain is considered as a blackbox

Border Gateway Protocol (BGP)

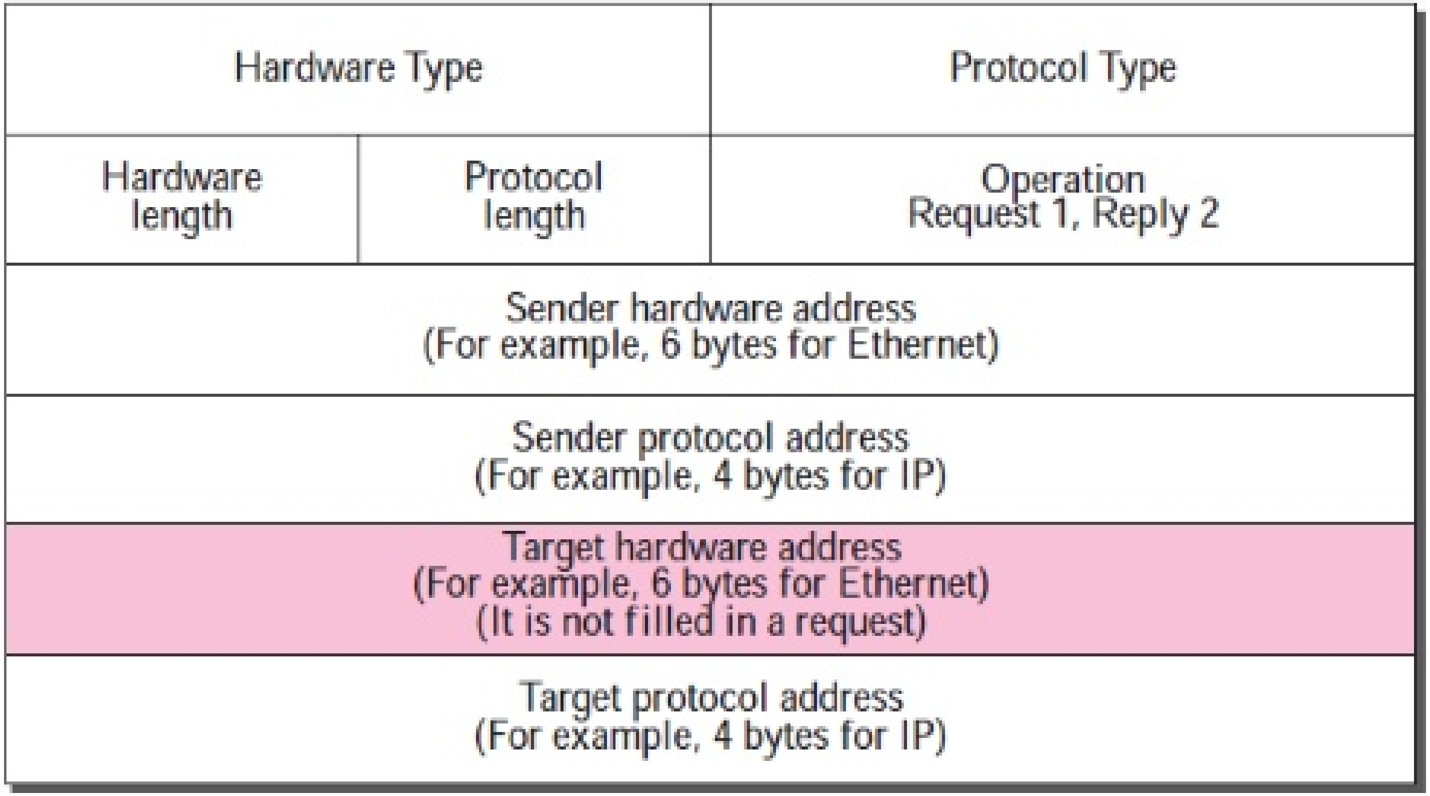
- Used as routing protocol between different domains.
- BGP may be used for routing within an autonomous system. (Interior Border Gateway Protocol/ iBGP).
- In contrast, the Internet application of the protocol may be referred to as Exterior Border Gateway Protocol, or eBGP.

Adress Resolution Protocol

- A host A wants to send a packet to 192.168.1.23
- IP layer, OK, but Datalink layer does not understand IP addresses. It uses MAC addresses (hardware address)
- Address Resolution Protocol (ARP) “translates” IP addresses into MAC, and *vice-versa.*

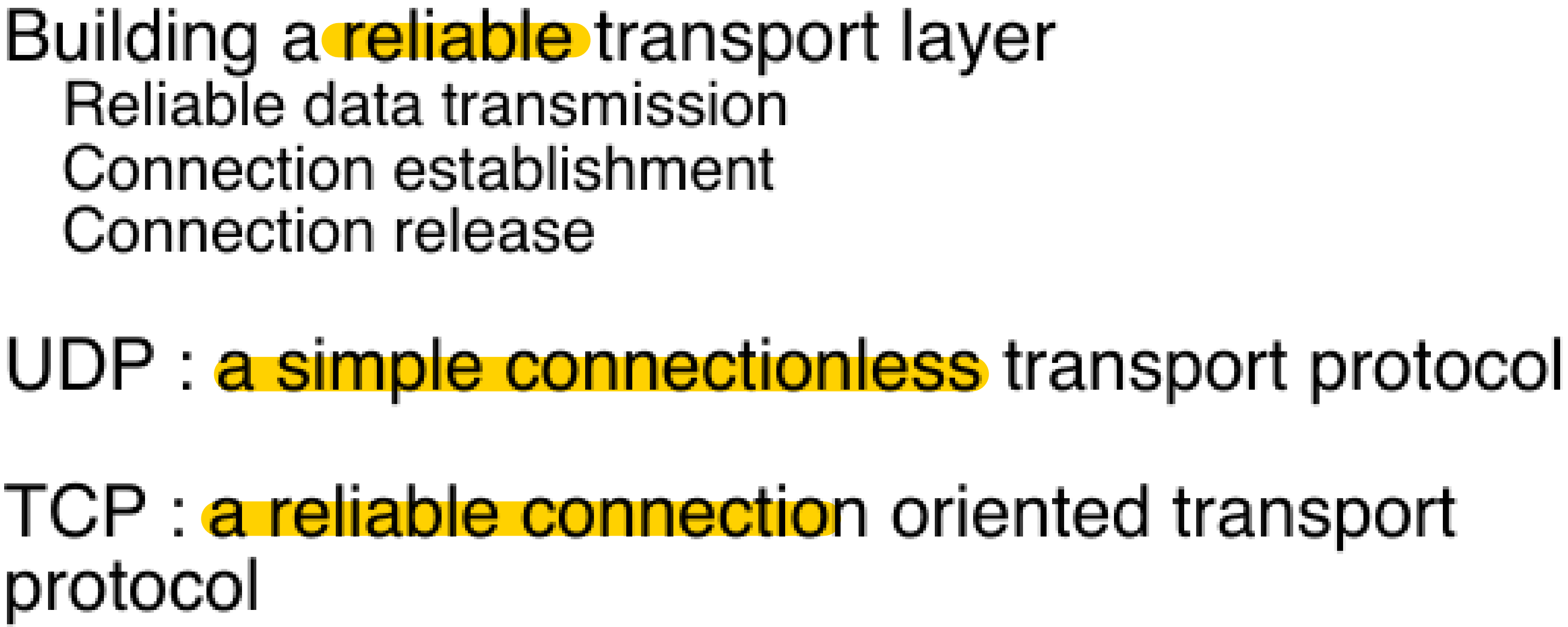
ARP

- Source broadcasts message “Who has IP 192.168.1.23”
- The host with the right IP answers with his MAC in unicast mode.



<table>
<tr><td colspan="2">Hardware Type</td><td>Protocol Type</td></tr>
<tr><td>Hardware length</td><td>Protocol length</td><td>Operation
Request 1, Reply 2</td></tr>
<tr><td colspan="3">Sender hardware address
(For example, 6 bytes for Ethernet)</td></tr>
<tr><td colspan="3">Sender protocol address
(For example, 4 bytes for IP)</td></tr>
<tr><td colspan="3">Target hardware address
(For example, 6 bytes for Ethernet)
(It is not filled in a request)</td></tr>
<tr><td colspan="3">Target protocol address
(For example, 4 bytes for IP)</td></tr>
</table>

Transport Layer



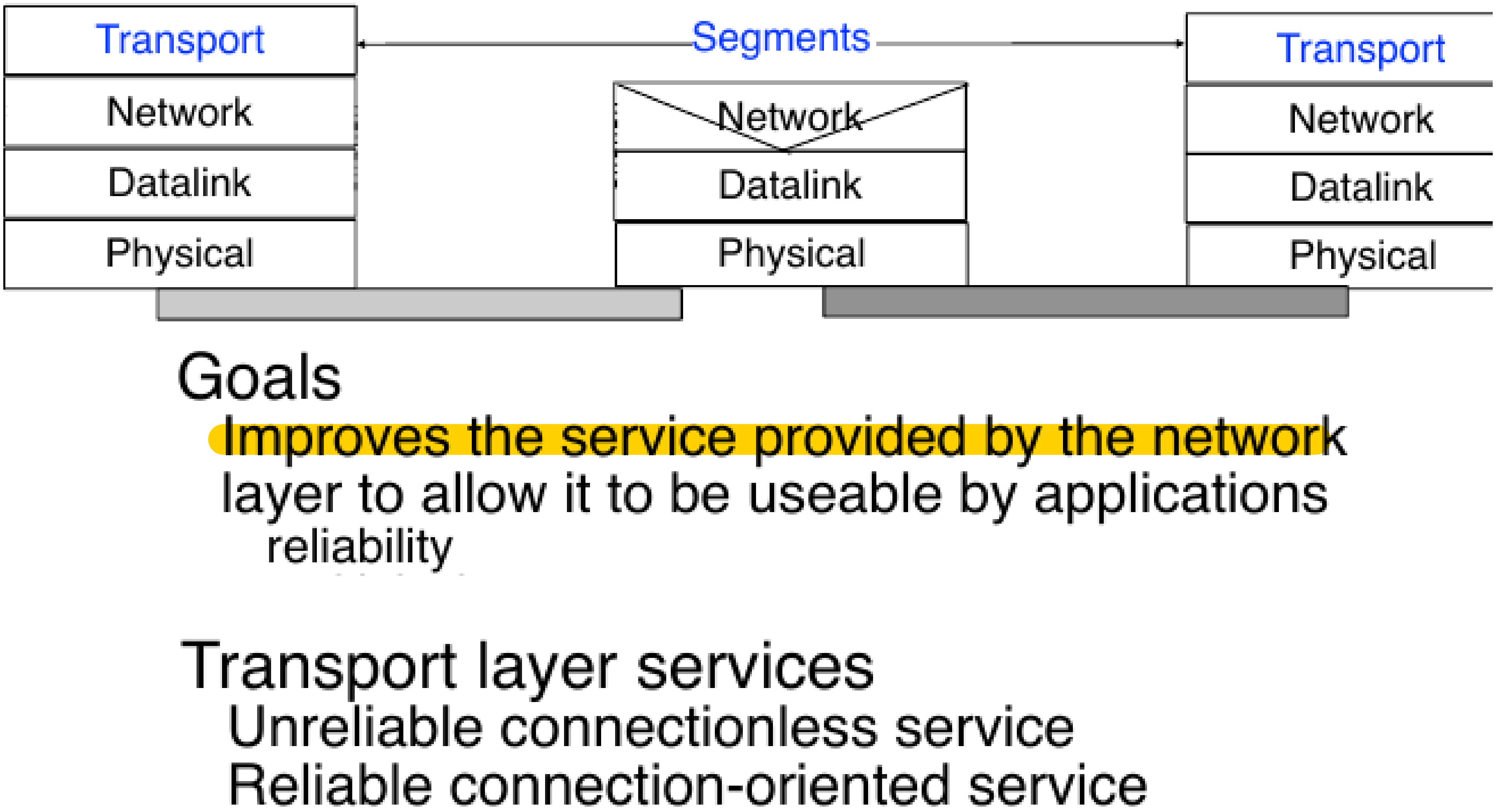
Building a reliable transport layer

- Reliable data transmission
- Connection establishment
- Connection release

UDP : a simple connectionless transport protocol

TCP : a reliable connection oriented transport protocol

Transport Layer



Goals

Improves the service provided by the network layer to allow it to be useable by applications

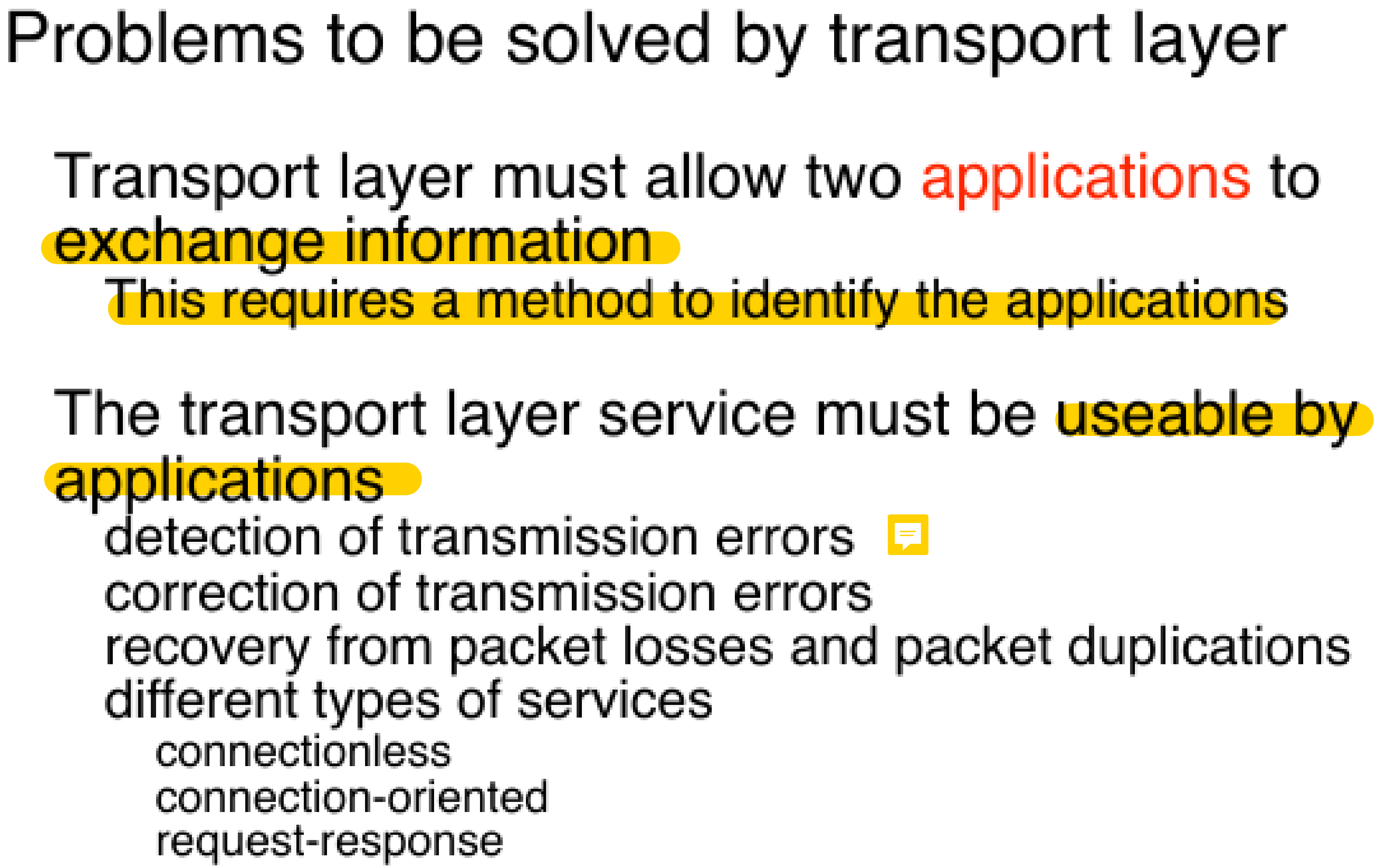
reliability

Transport layer services

Unreliable connectionless service

Reliable connection-oriented service

Transport Layer



Problems to be solved by transport layer

- Transport layer must allow two applications to exchange information
  - This requires a method to identify the applications
- The transport layer service must be useable by applications
  - detection of transmission errors
  - correction of transmission errors
  - recovery from packet losses and packet duplications
  - different types of services
    - connectionless
    - connection-oriented
    - request-response

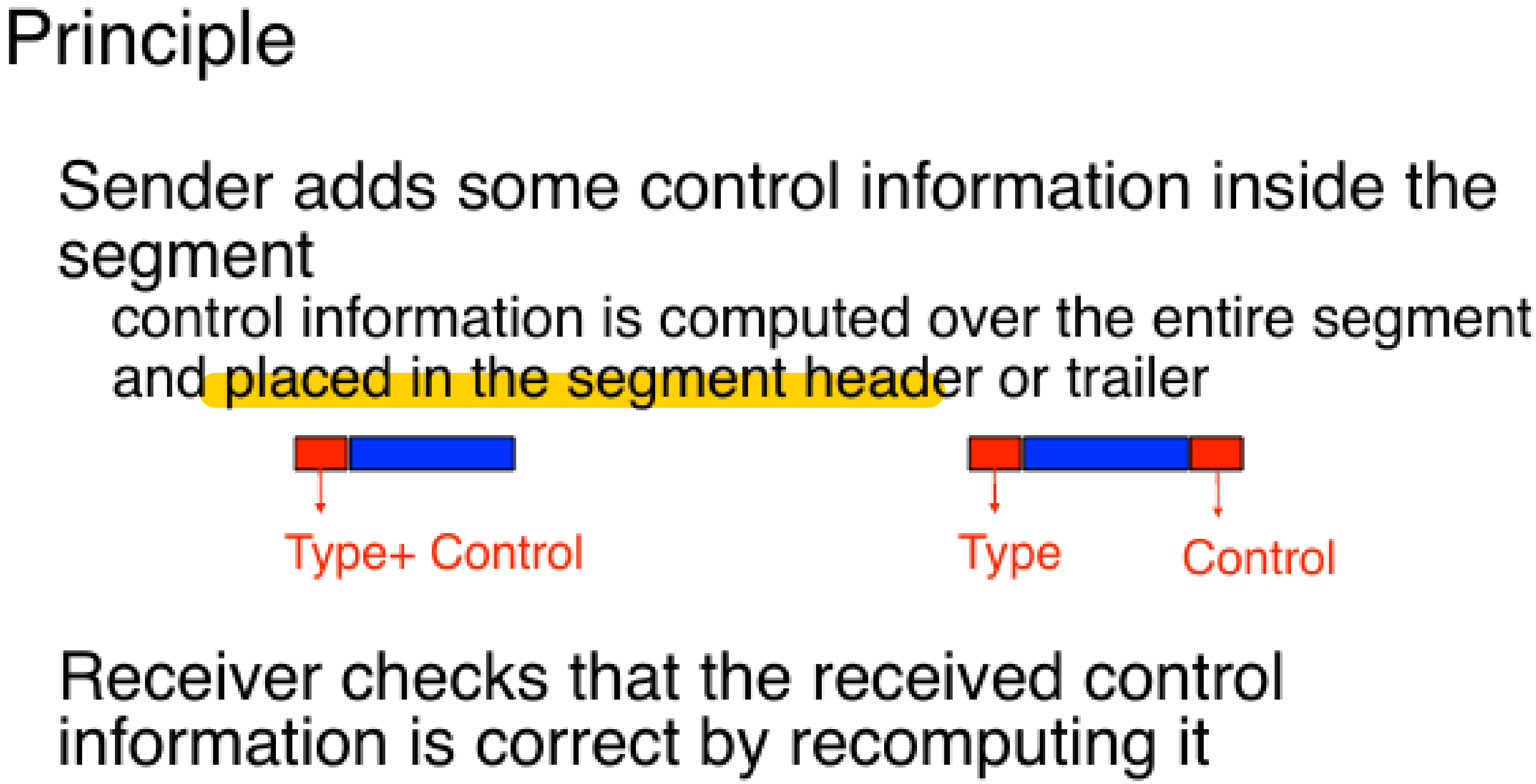
Transmition Errors

Which types of transmission errors do we need to consider in the transport layer ?

Physical-layer transmission errors caused by nature

- Random isolated error
  - one bit is flipped in the segment
- Random burst error
  - a group of n bits inside the segment is errored
    - most of the bits in the group are flipped

Transmition Errors



Principle

- Sender adds some control information inside the segment
  - control information is computed over the entire segment and placed in the segment header or trailer

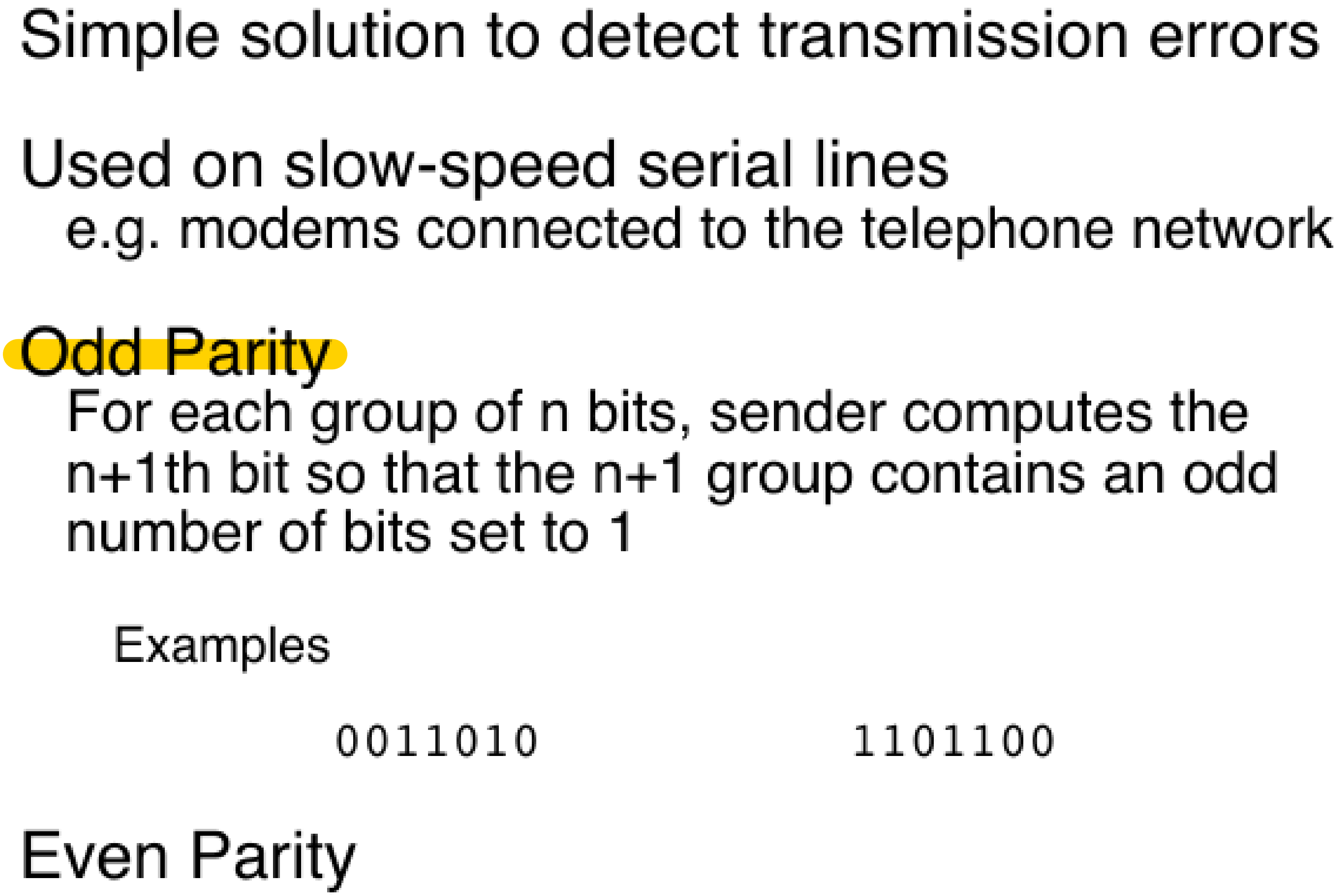
Type+ Control

Type

Control

- Receiver checks that the received control information is correct by recomputing it

Transmition Errors



Simple solution to detect transmission errors

Used on slow-speed serial lines

- e.g. modems connected to the telephone network

Odd Parity

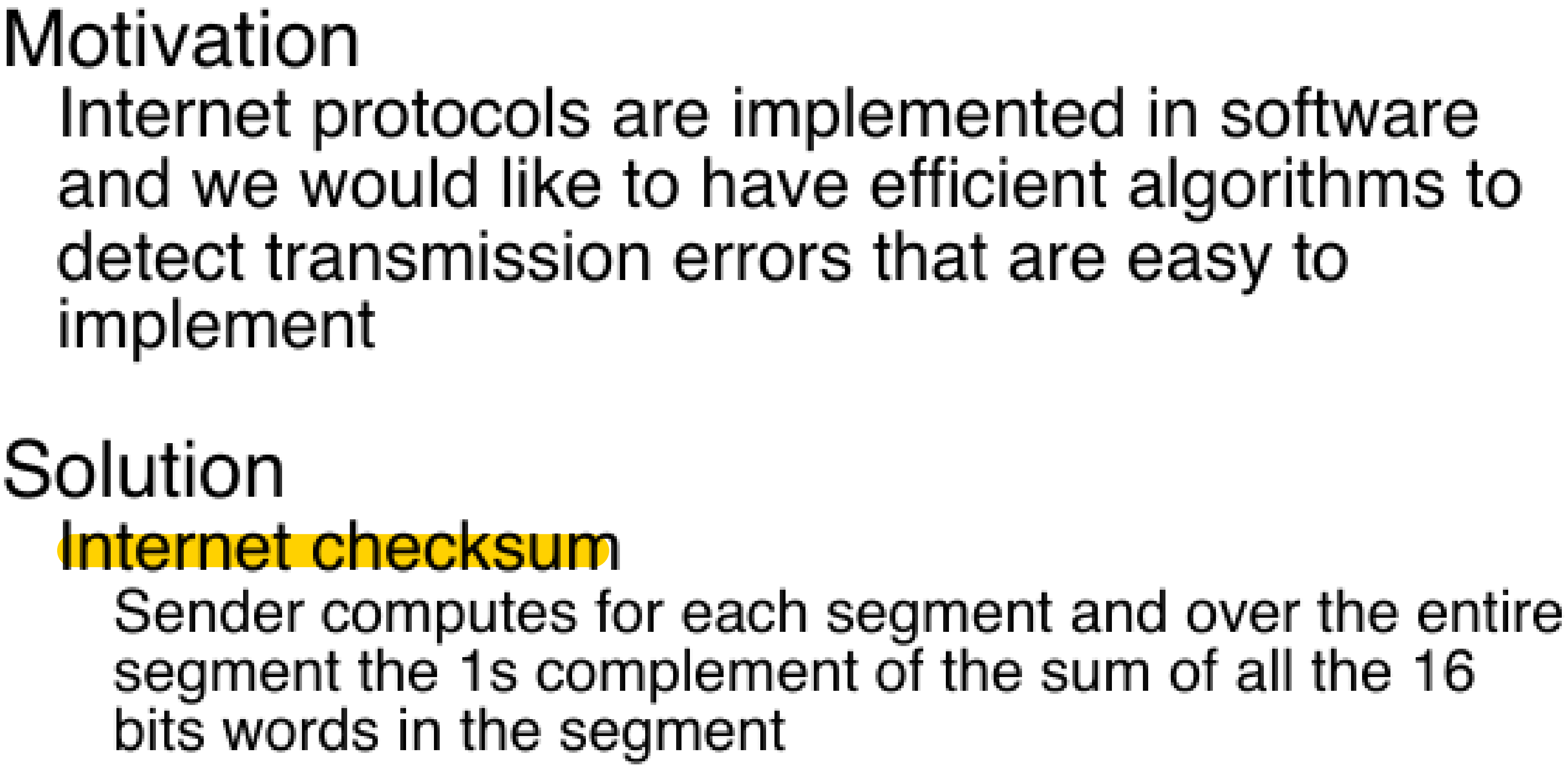
- For each group of n bits, sender computes the n+1th bit so that the n+1 group contains an odd number of bits set to 1

Examples

```
0011010          1101100
```

Even Parity

Transmition Errors



Motivation

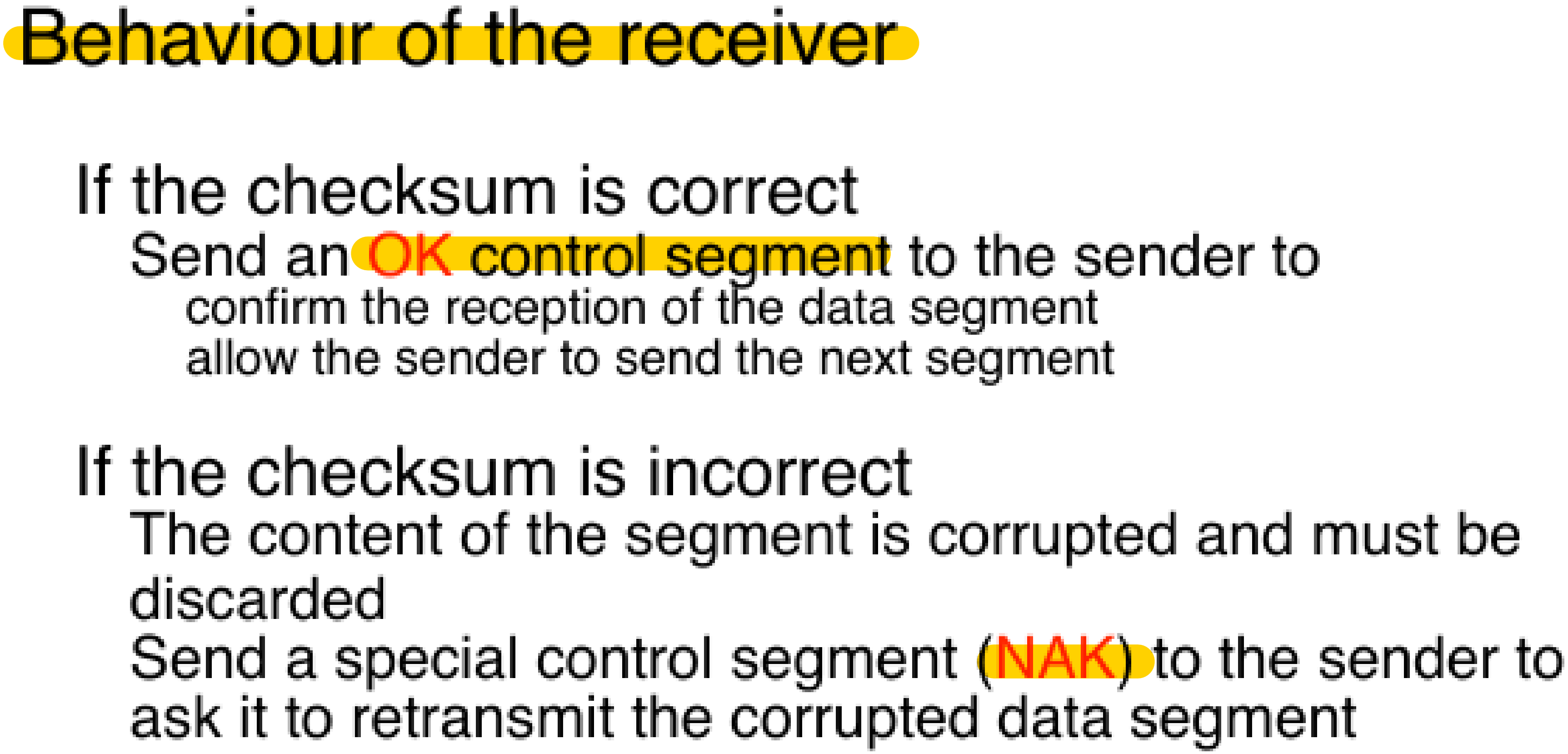
Internet protocols are implemented in software and we would like to have efficient algorithms to detect transmission errors that are easy to implement

Solution

Internet checksum

Sender computes for each segment and over the entire segment the 1s complement of the sum of all the 16 bits words in the segment

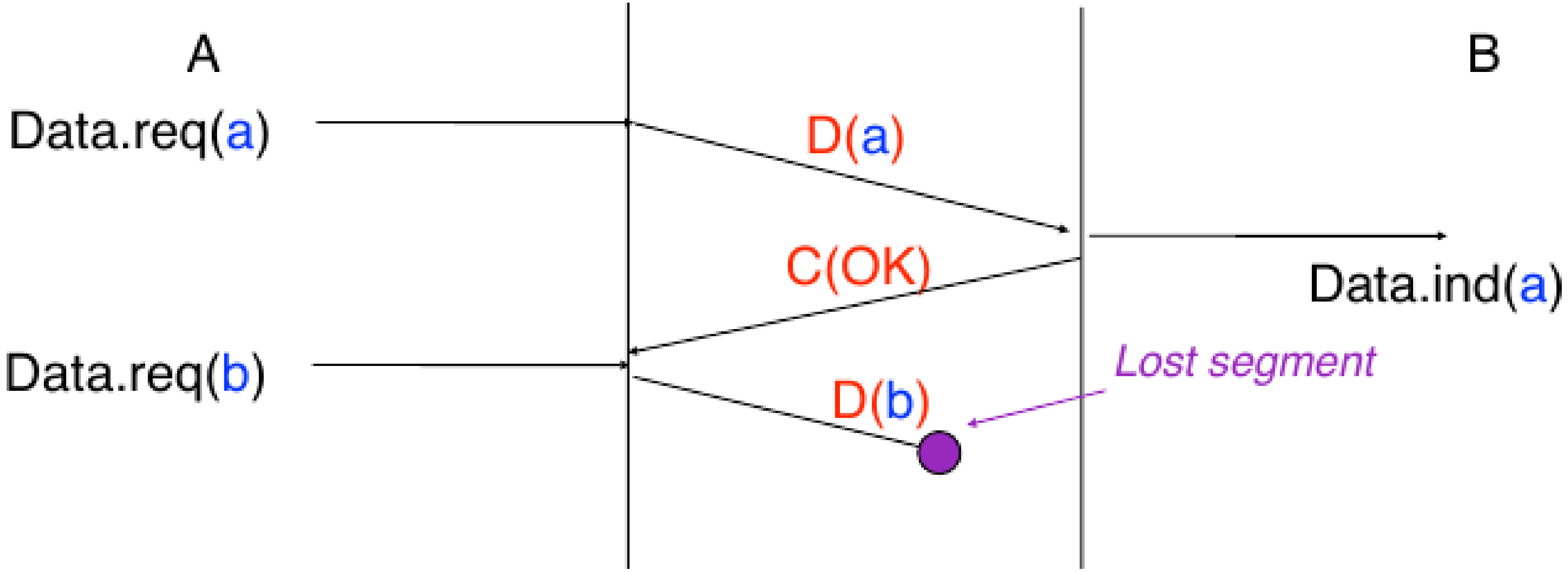
Transmition Errors



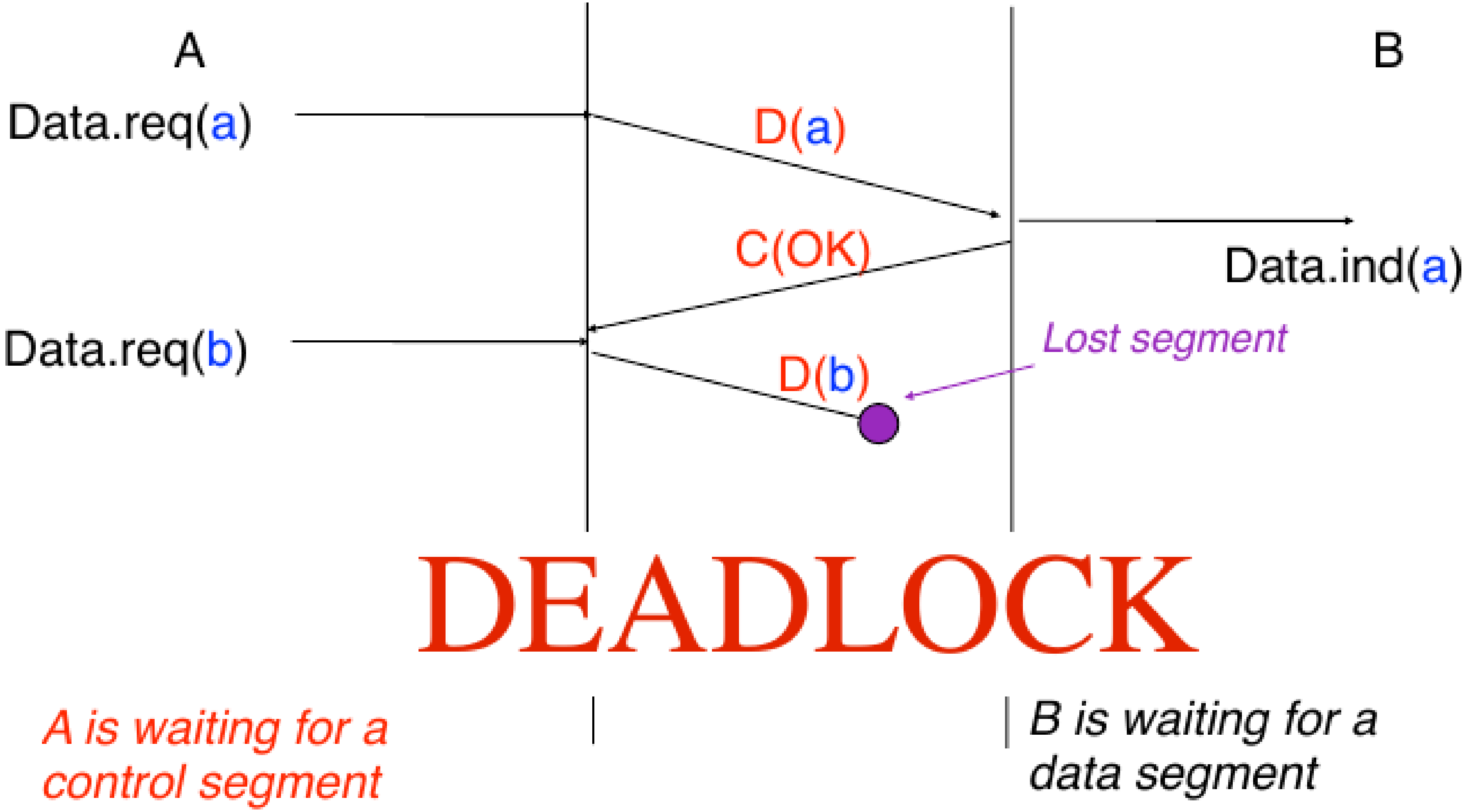
Behaviour of the receiver

- If the checksum is correct
  - Send an OK control segment to the sender to
    - confirm the reception of the data segment
    - allow the sender to send the next segment
- If the checksum is incorrect
  - The content of the segment is corrupted and must be discarded
  - Send a special control segment (NAK) to the sender to ask it to retransmit the corrupted data segment

Transmition Errors



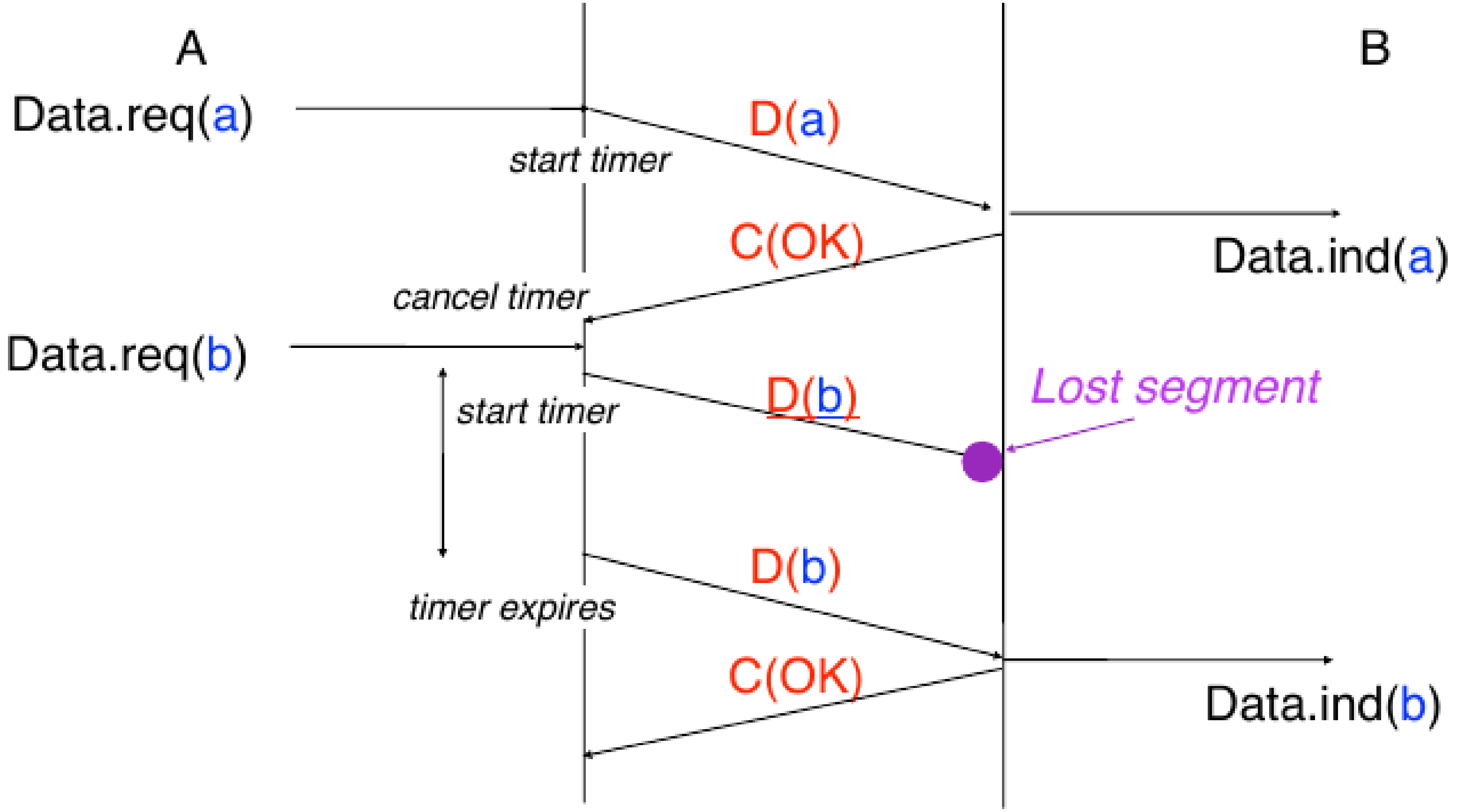
Transmition Errors



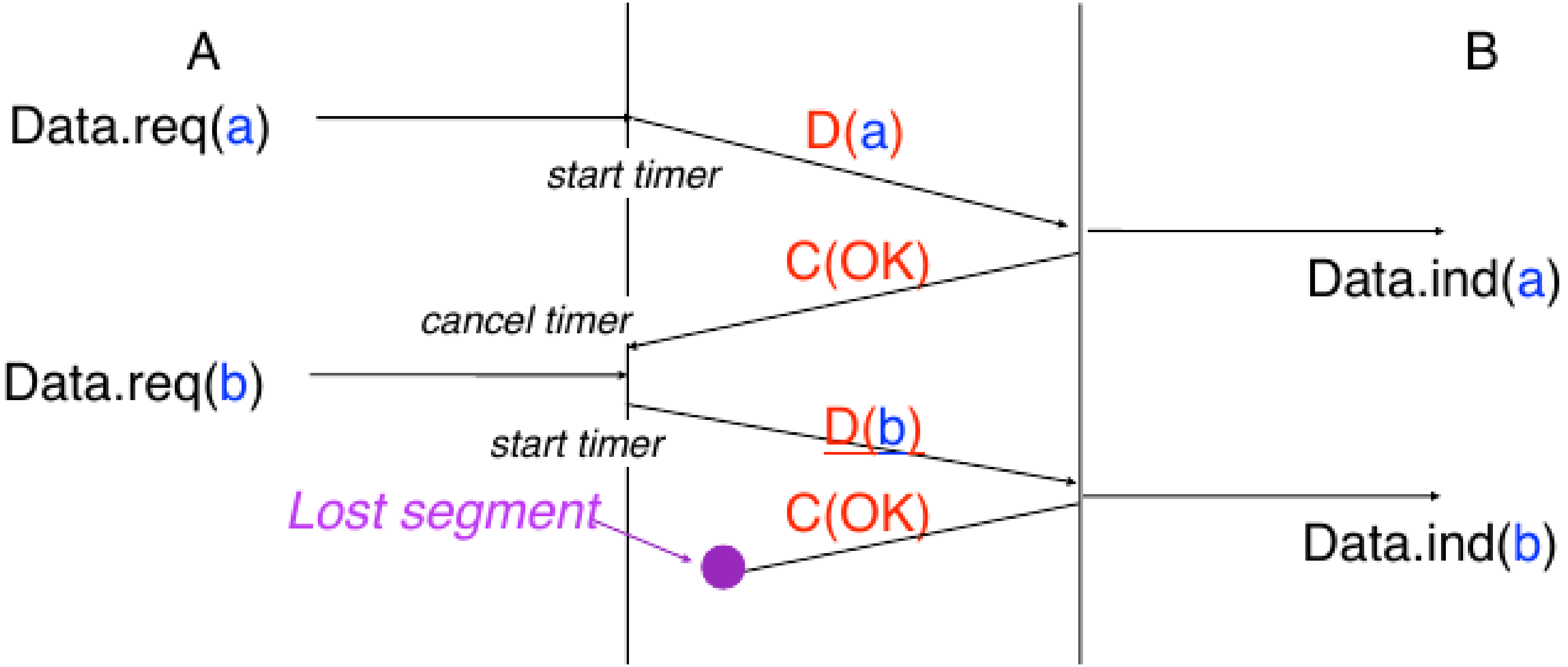
Transmition Errors

Modification to the sender

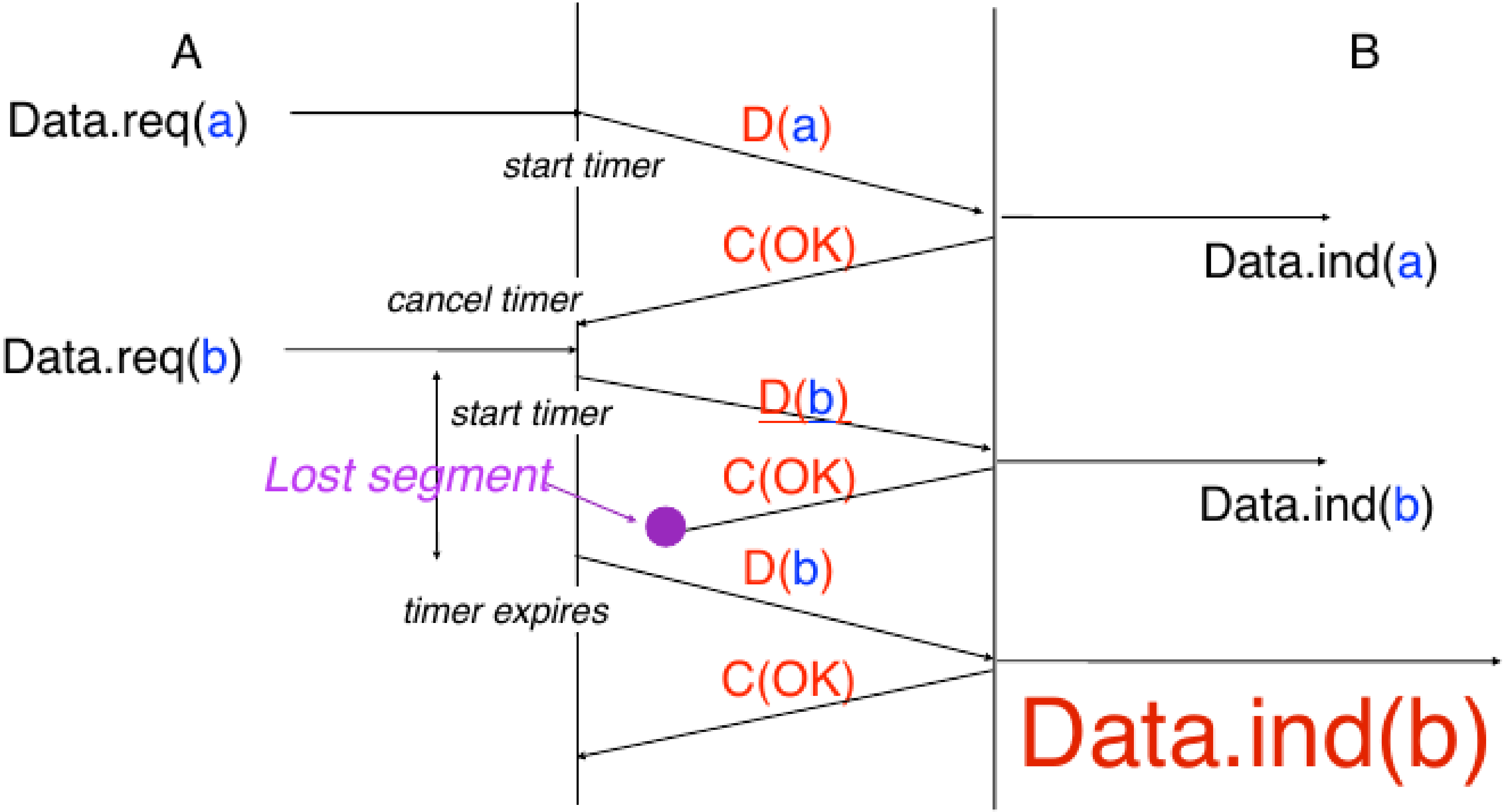
- Add a retransmission timer to retransmit the lost segment after some time



Transmition Errors



Transmition Errors



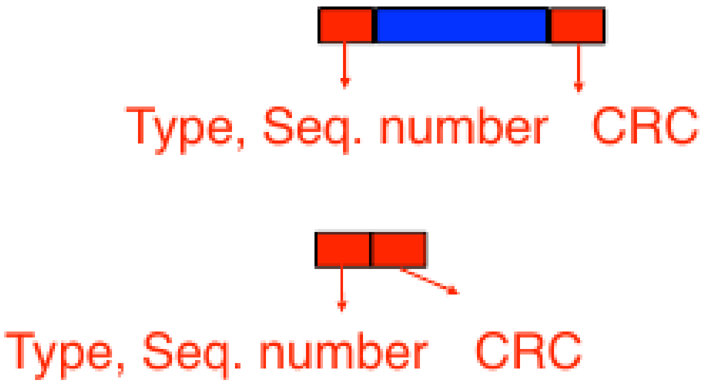
Transmition Errors

Principles of the solution

- Add sequence numbers to each data segment sent by sender
- By looking at the sequence number, the receiver can check whether it has already received this segment

Contents of each segment

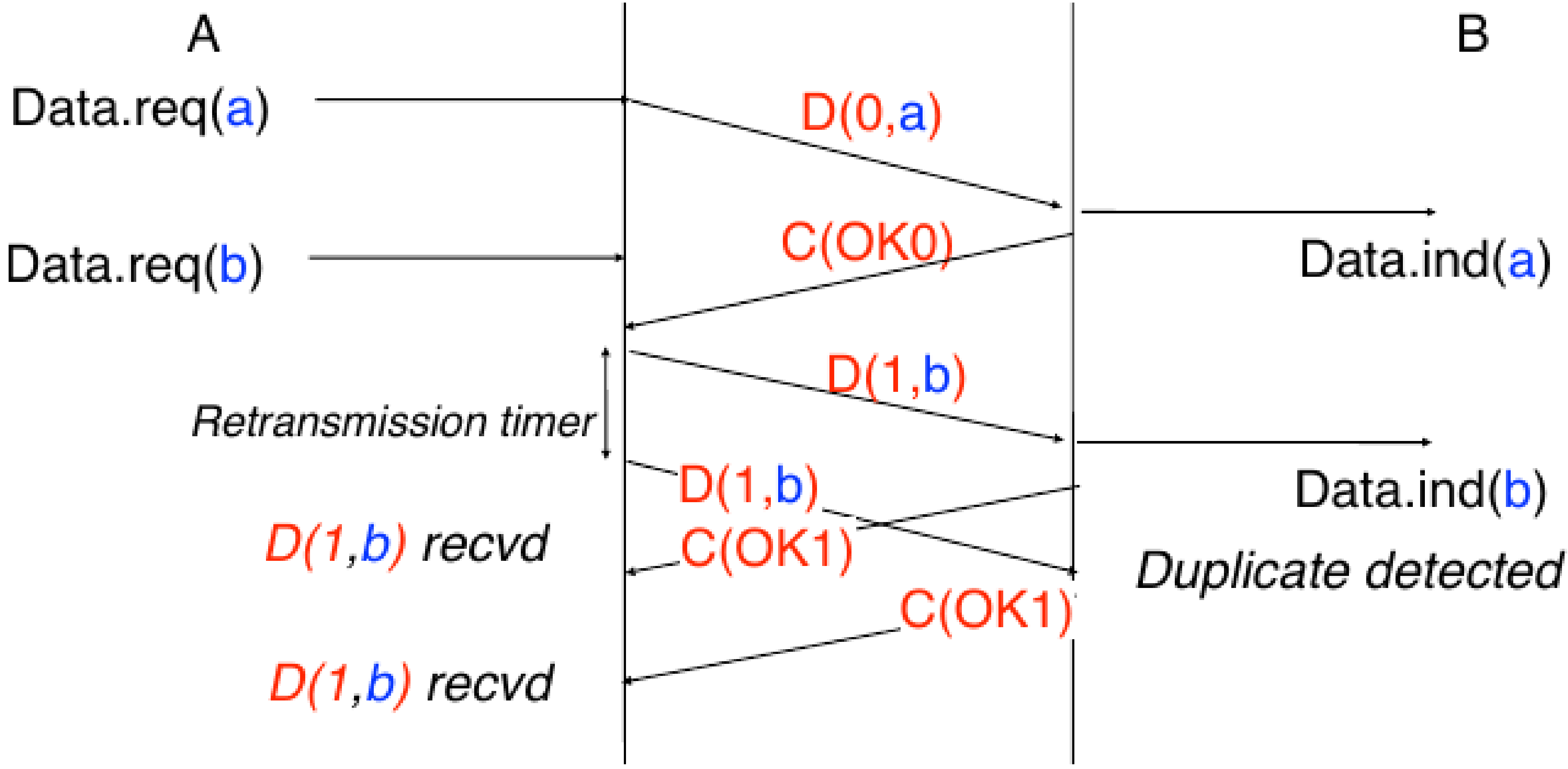
- Data segments
- Control segments



Type, Seq. number CRC

Type, Seq. number CRC

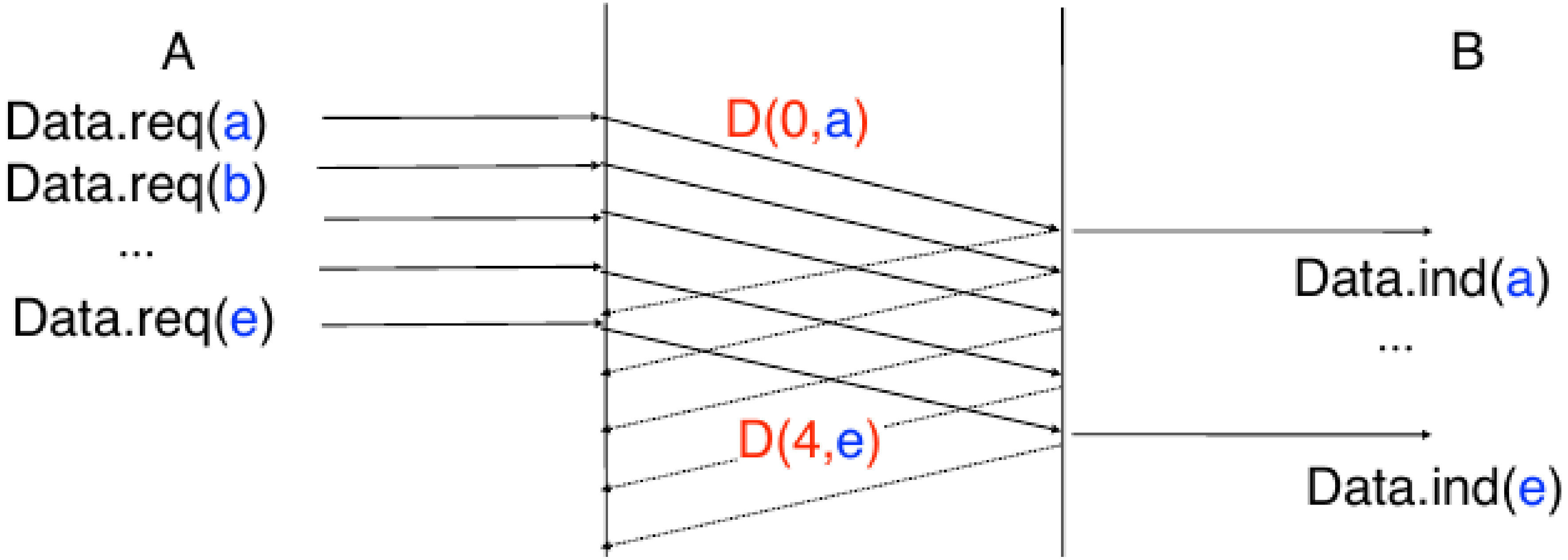
Transmition Errors



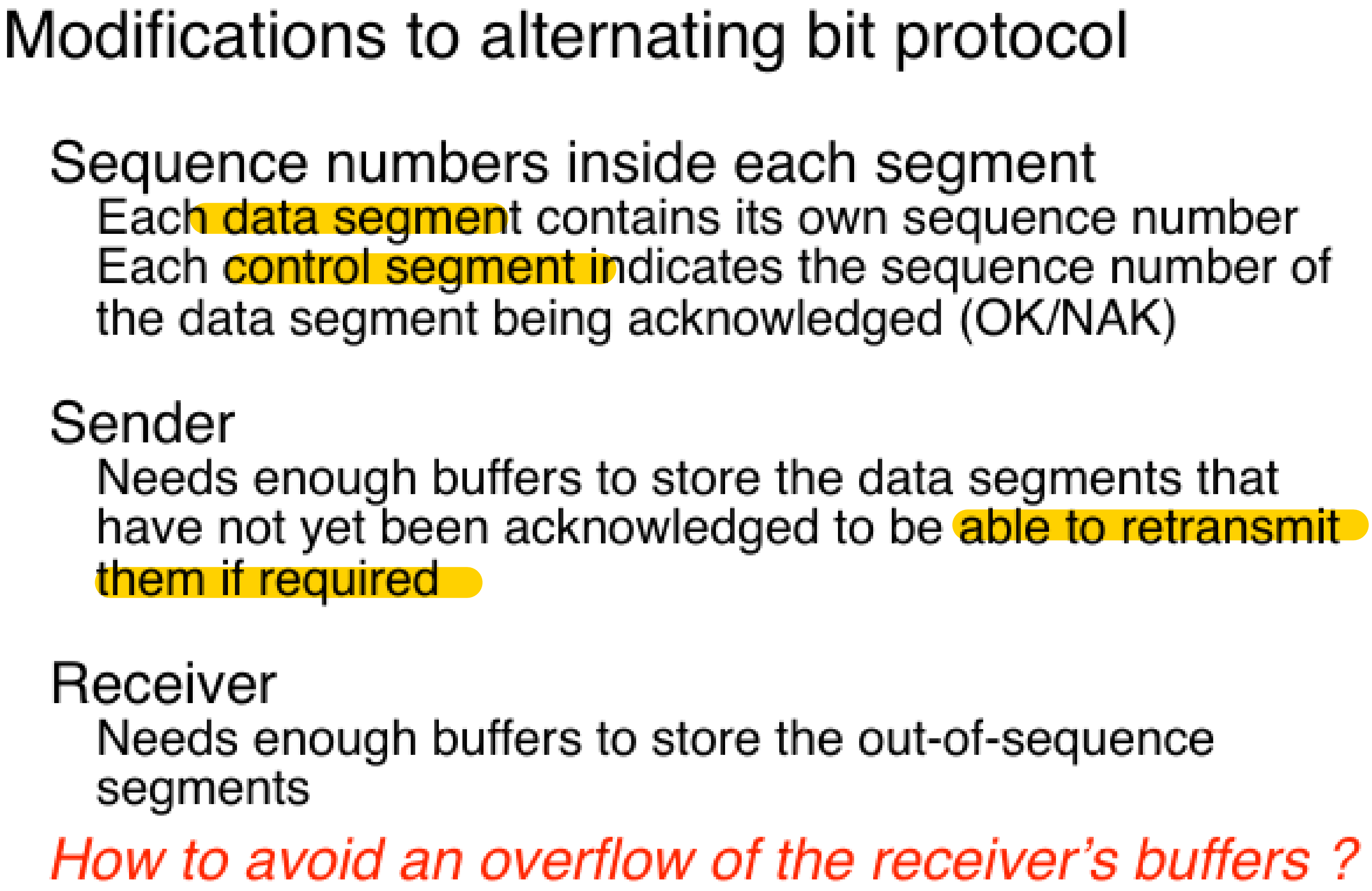
Improvements

Principle

The sender should be allowed to send more than one segment while waiting for an acknowledgement from the receiver



Improvements



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

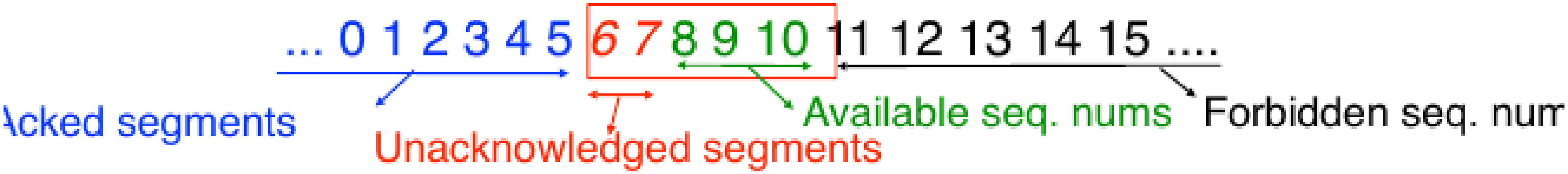
How to avoid an overflow of the receiver's buffers ?

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

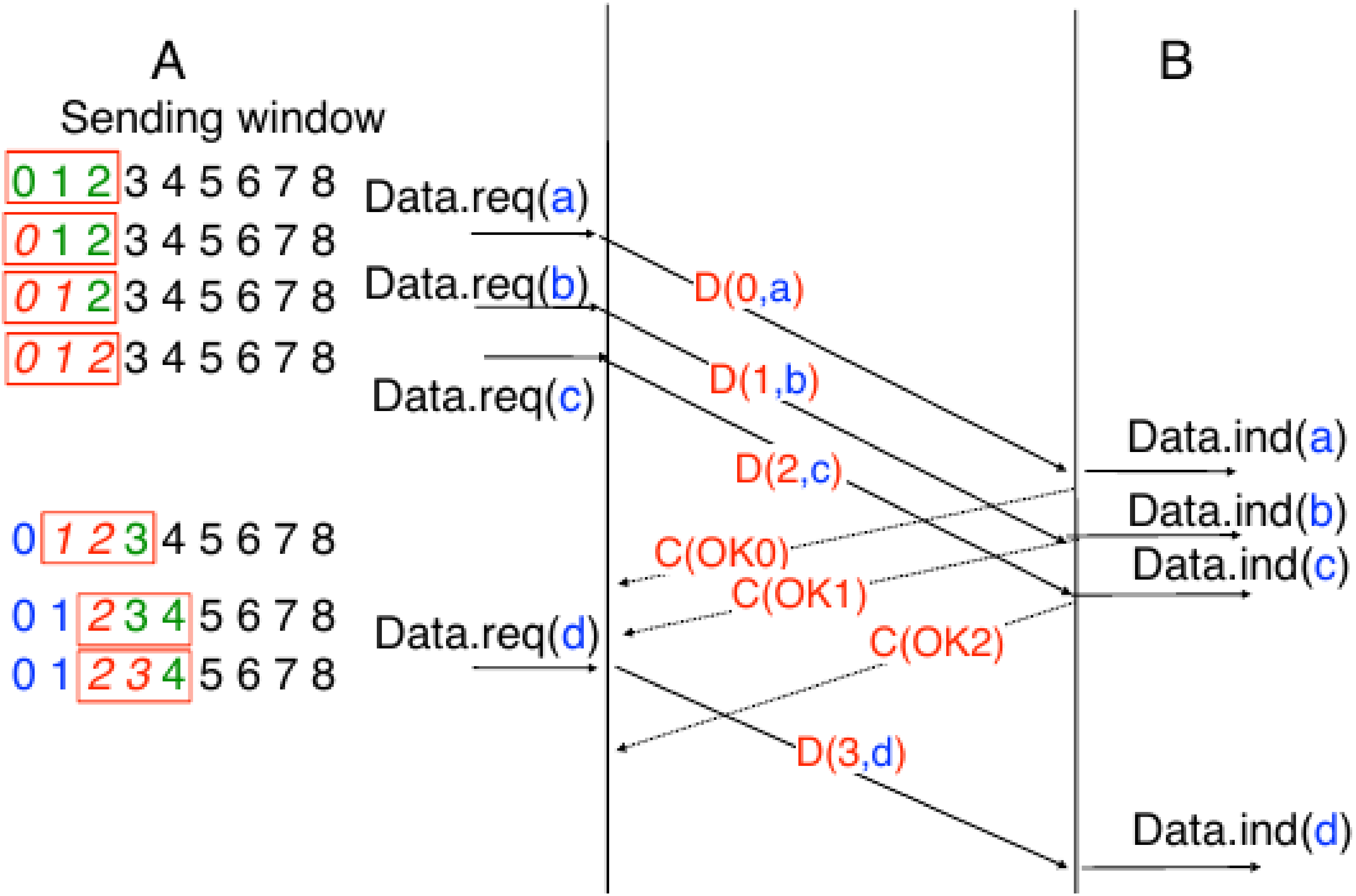
receiving_window

Sender and receiver must use compatible windows

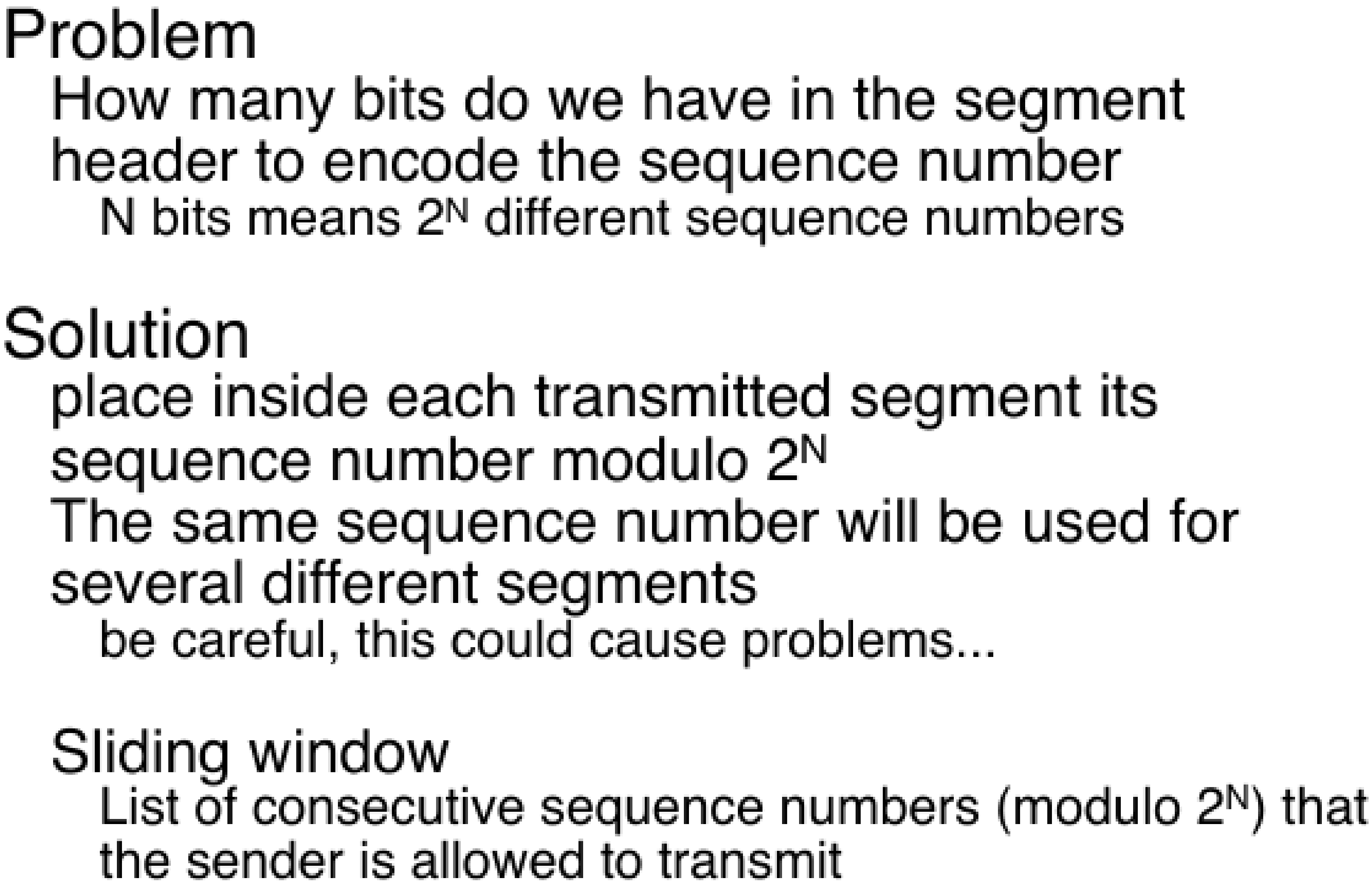
sending_window ≤ receiving window

For example, window size is a constant for a given protocol or negotiated during connection establishment phase

Sliding Window



Sliding Window



Problem

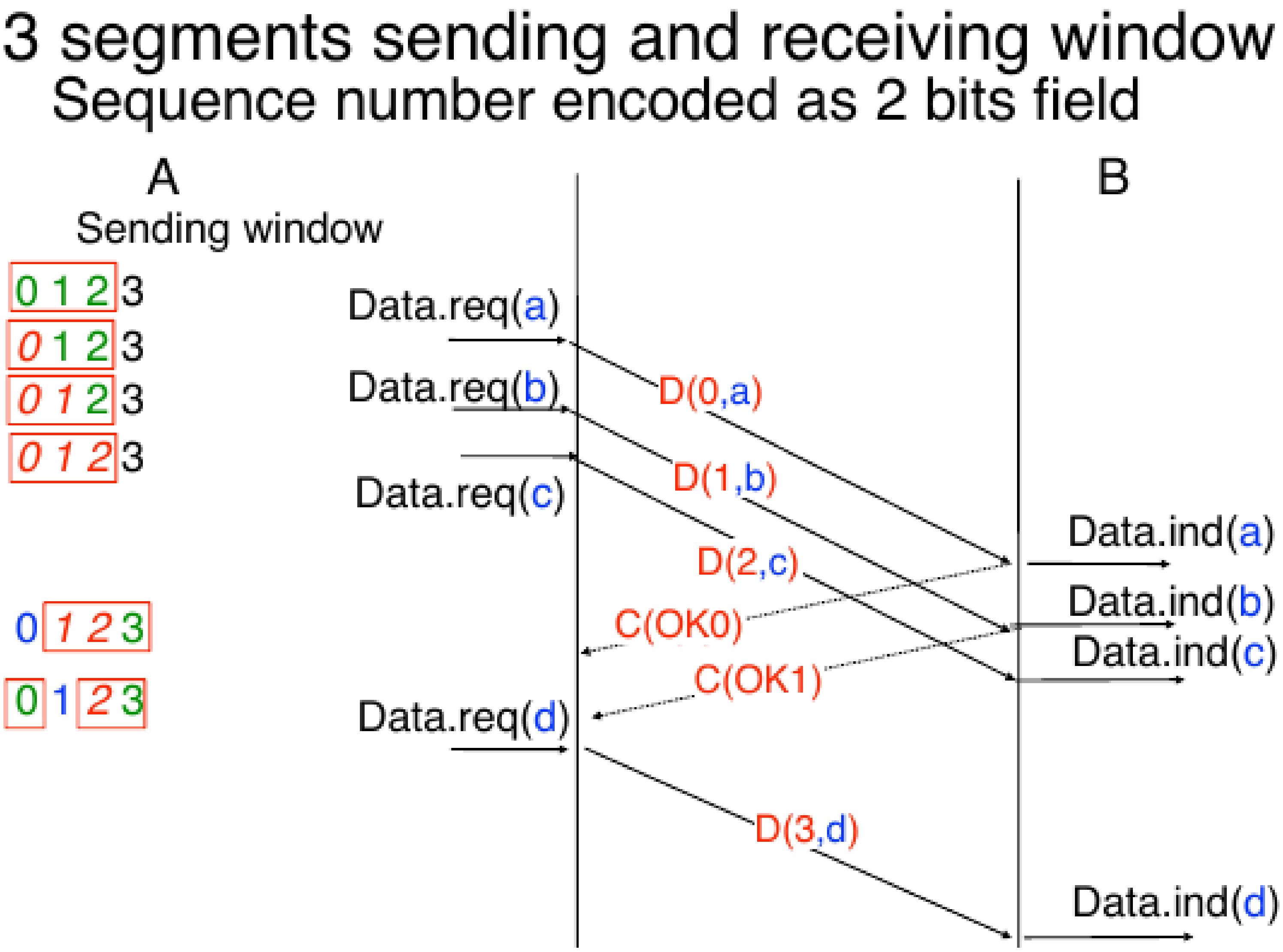
- How many bits do we have in the segment header to encode the sequence number
  - N bits means 2^N different sequence numbers

Solution

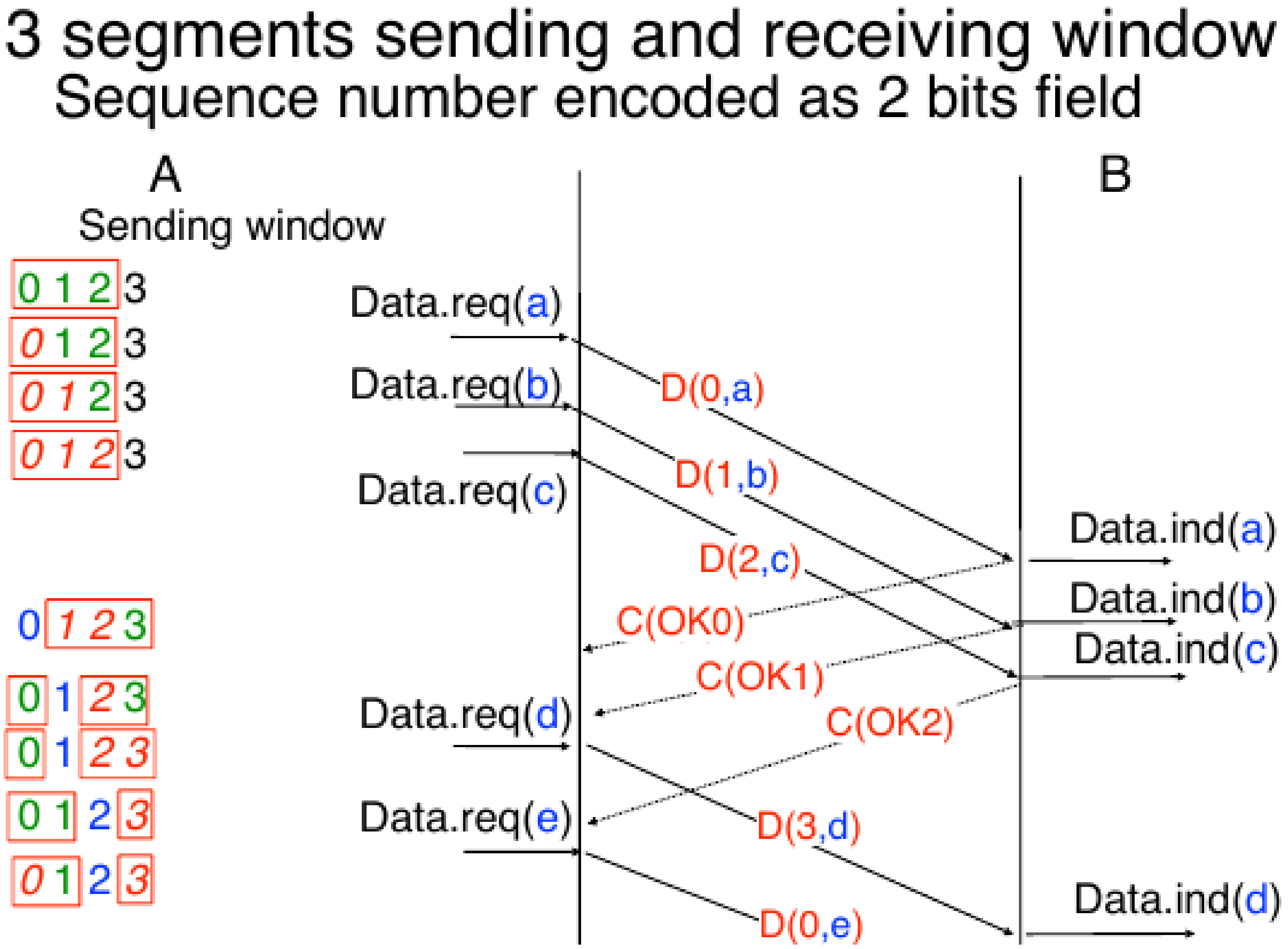
- place inside each transmitted segment its sequence number modulo 2^N
- The same sequence number will be used for several different segments
  - be careful, this could cause problems...

- Sliding window
  - List of consecutive sequence numbers (modulo 2^N) that the sender is allowed to transmit

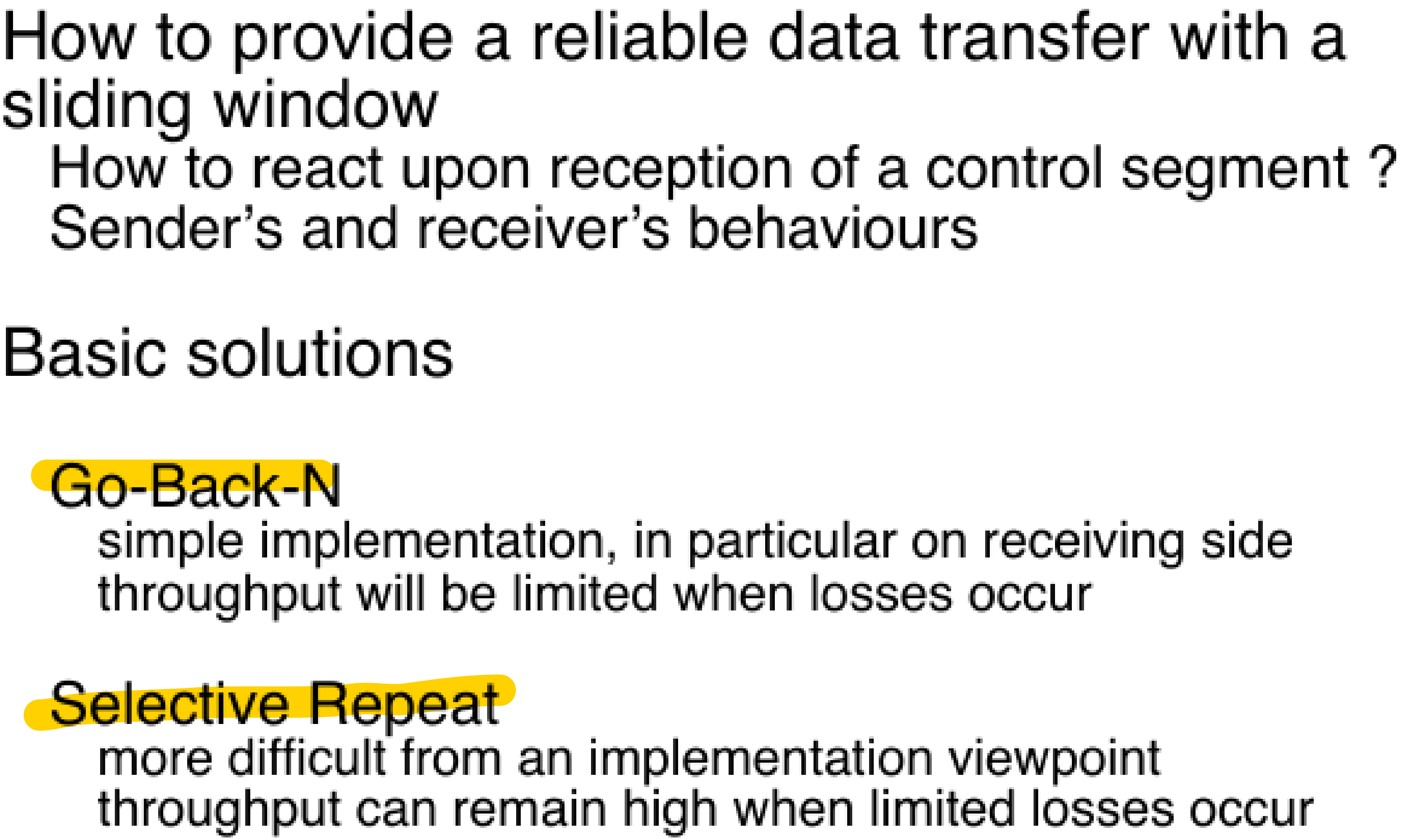
Sliding Window



Sliding Window

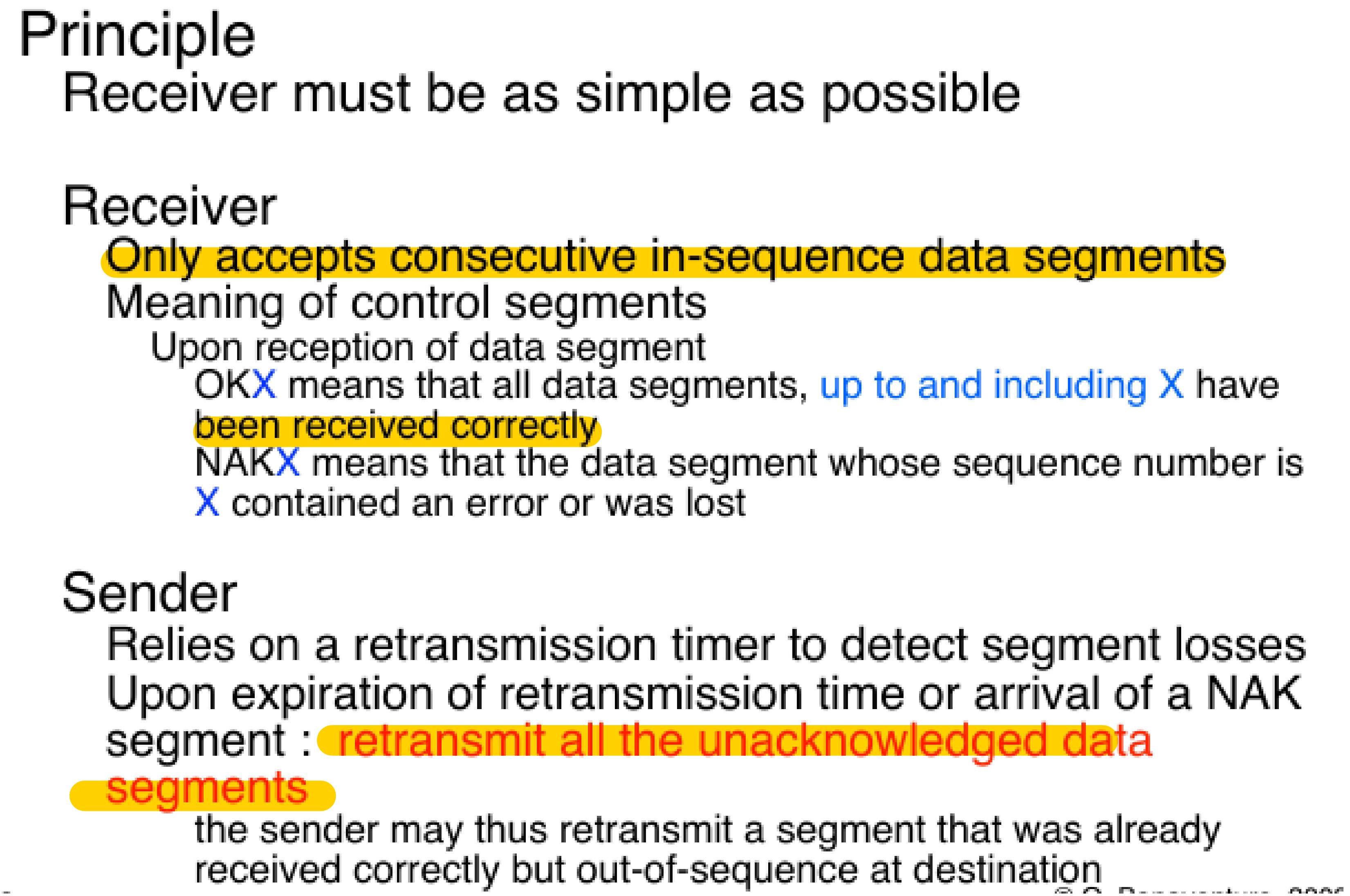


Reliable transfer with Sliding Window



- 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
  - 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

Go-Back-N



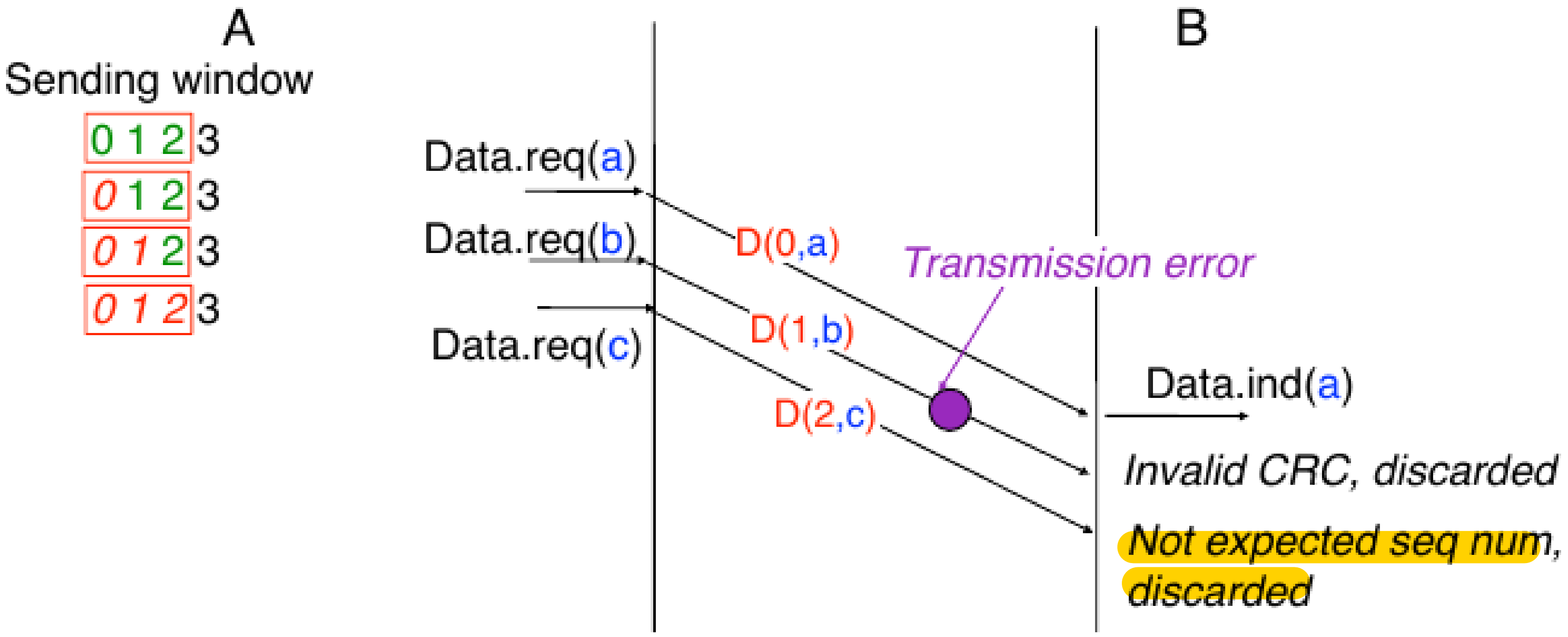
Principle

- Receiver must be as simple as possible

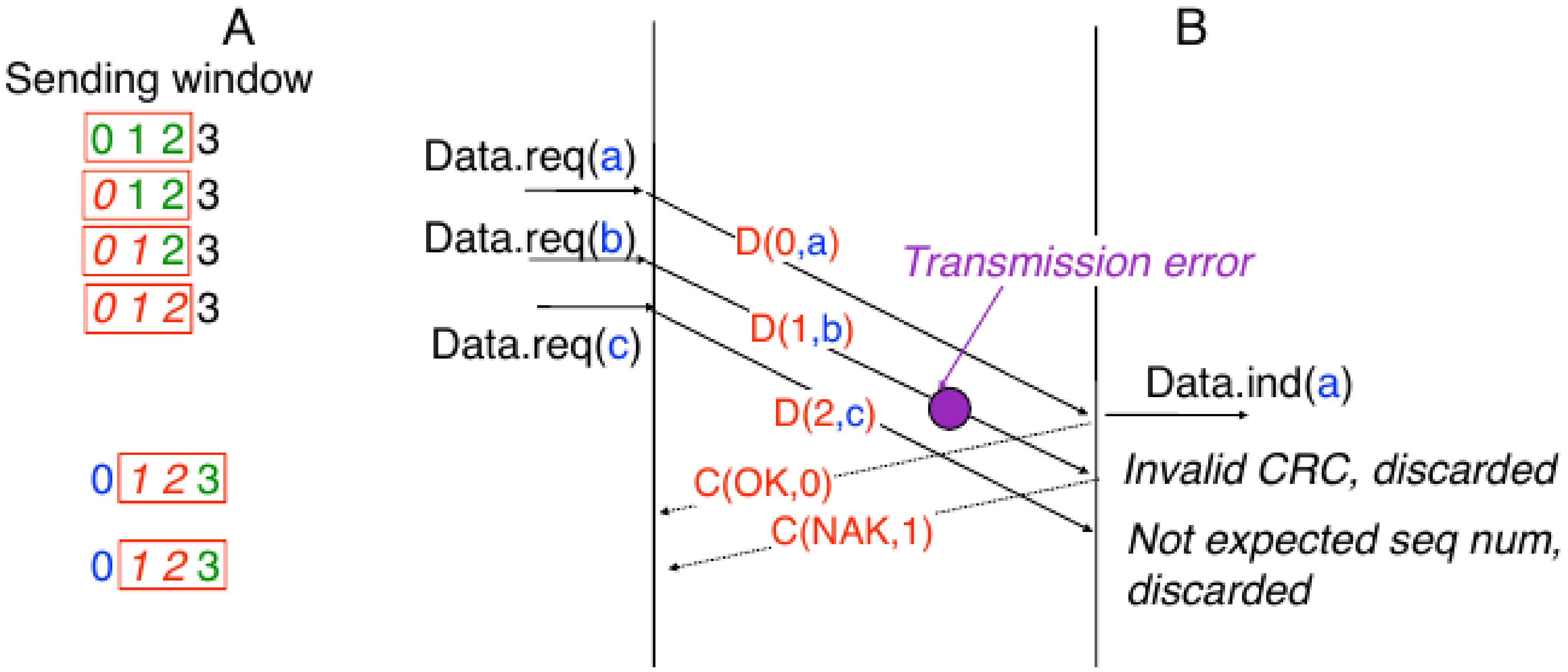
- Receiver
  - Only accepts consecutive in-sequence data segments
  - Meaning of control segments
    - Upon reception of data segment
      - OKX means that all data segments, up to and including X have been received correctly
      - NAKX means that the data segment whose sequence number is X contained an error or was lost

- 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

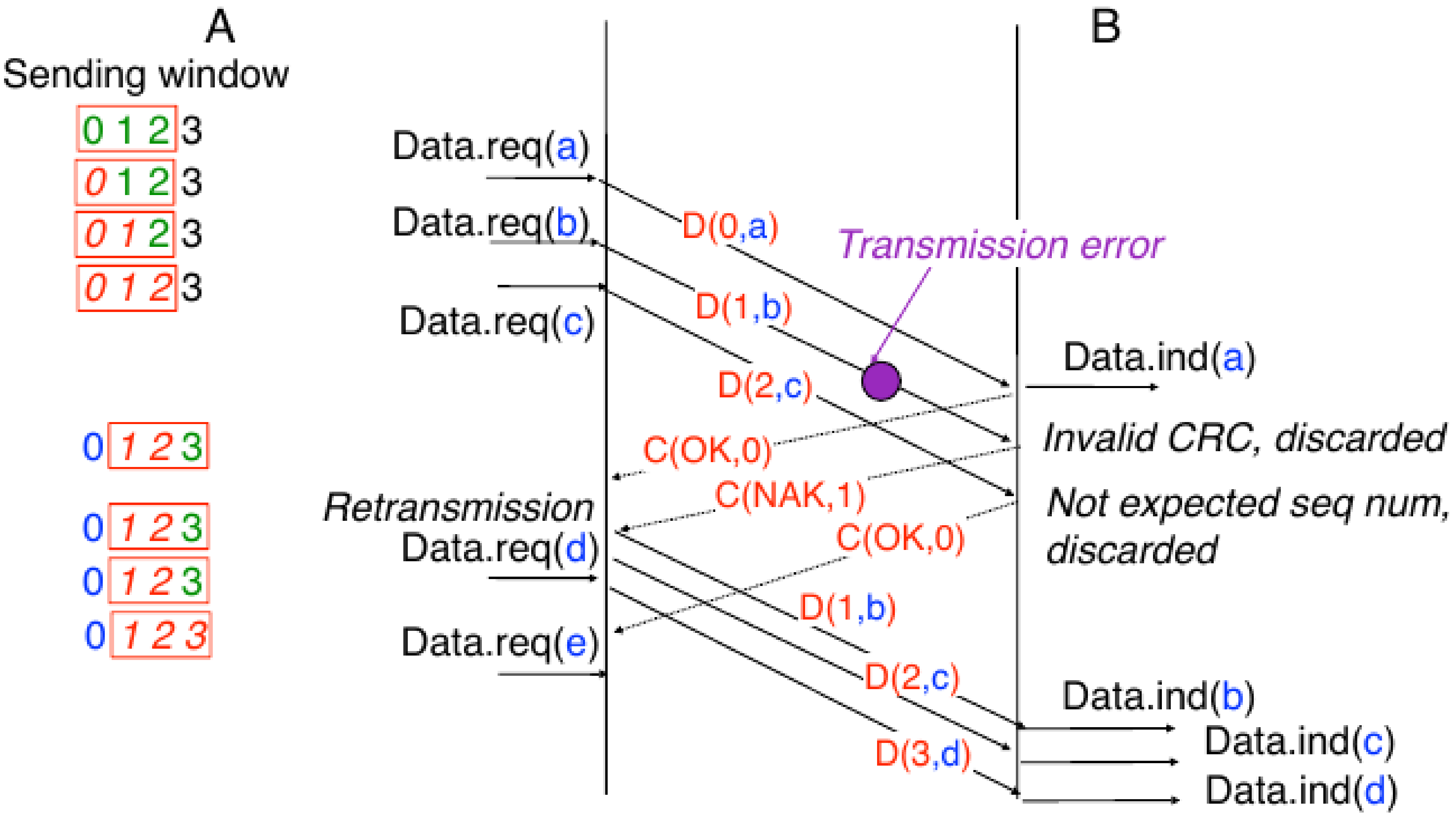
Go-Back-N


A
Sending window
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
Data.req(a)
Data.req(b)
Data.req(c)
D(0,a)
D(1,b)
D(2,c)
Transmission error
B
Data.ind(a)
Invalid CRC, discarded
Not expected seq num, discarded

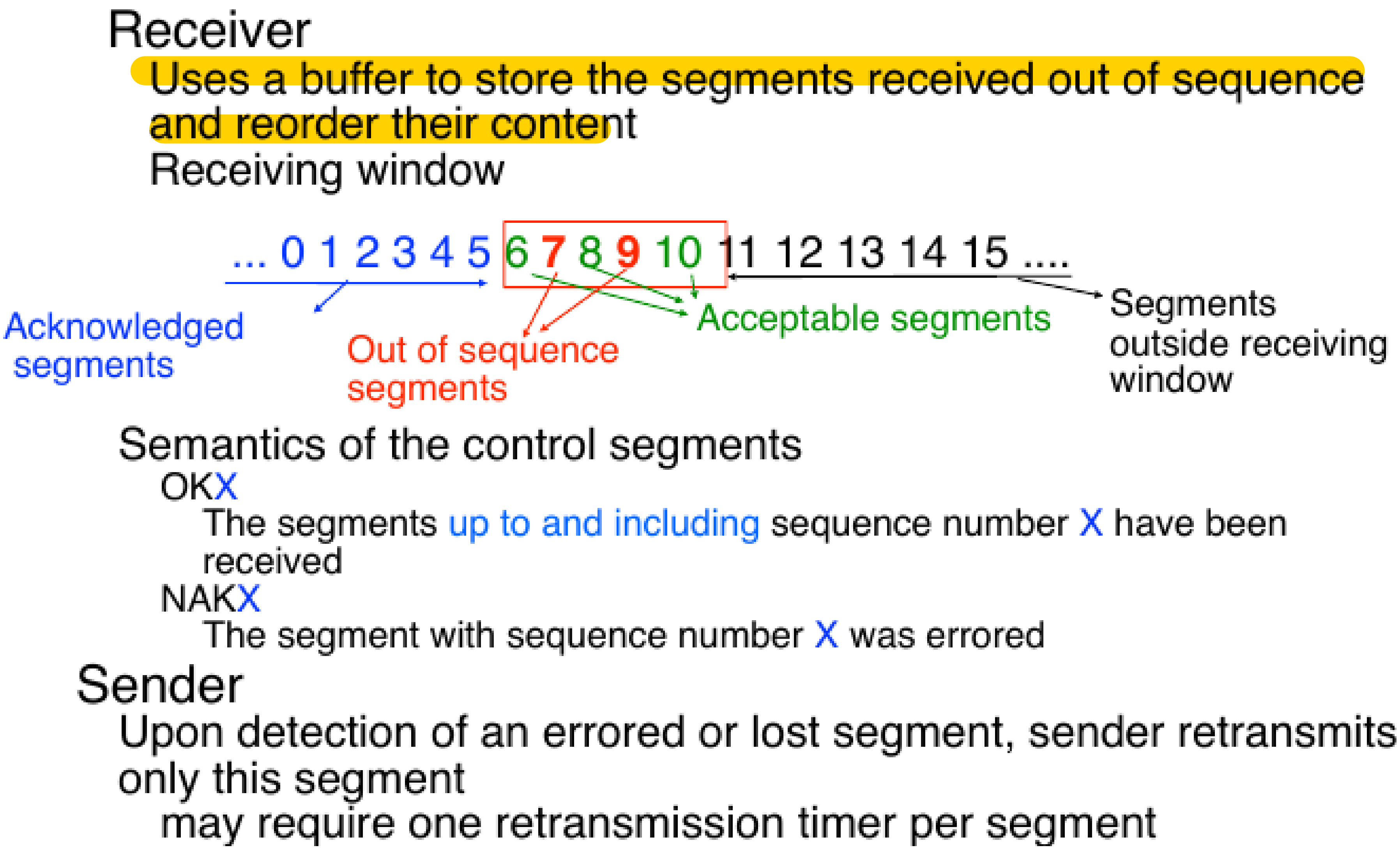
Go-Back-N


A
Sending window
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
B
Data.req(a)
Data.req(b)
Data.req(c)
D(0,a)
D(1,b)
D(2,c)
Transmission error
Data.ind(a)
Invalid CRC, discarded
Not expected seq num, discarded
C(OK,0)
C(NAK,1)

Go-Back-N


A
Sending window
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
0 1 2 3
B
Data.req(a)
Data.req(b)
Data.req(c)
D(0,a)
D(1,b)
D(2,c)
Transmission error
Data.ind(a)
Invalid CRC, discarded
C(OK,0)
C(NAK,1)
Not expected seq num, discarded
Retransmission
Data.req(d)
C(OK,0)
D(1,b)
Data.req(e)
D(2,c)
D(3,d)
Data.ind(b)
Data.ind(c)
Data.ind(d)

Selective Repeat



Receiver

- Uses a buffer to store the segments received out of sequence and reorder their content
- Receiving window

... 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ....

Acknowledged segments

Out of sequence segments

Acceptable segments

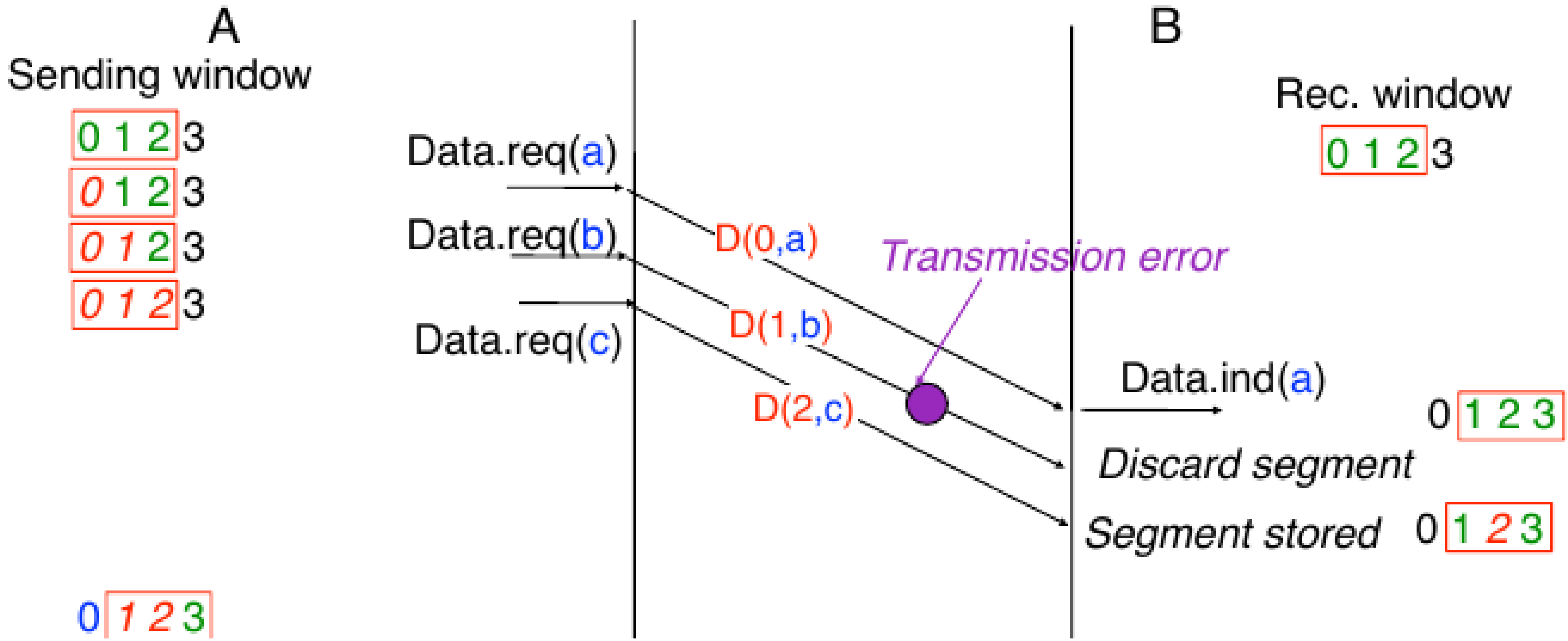
Segments outside receiving window

- Semantics of the control segments
  - OKX
    - The segments up to and including sequence number X have been received
  - NAKX
    - The segment with sequence number X was errored

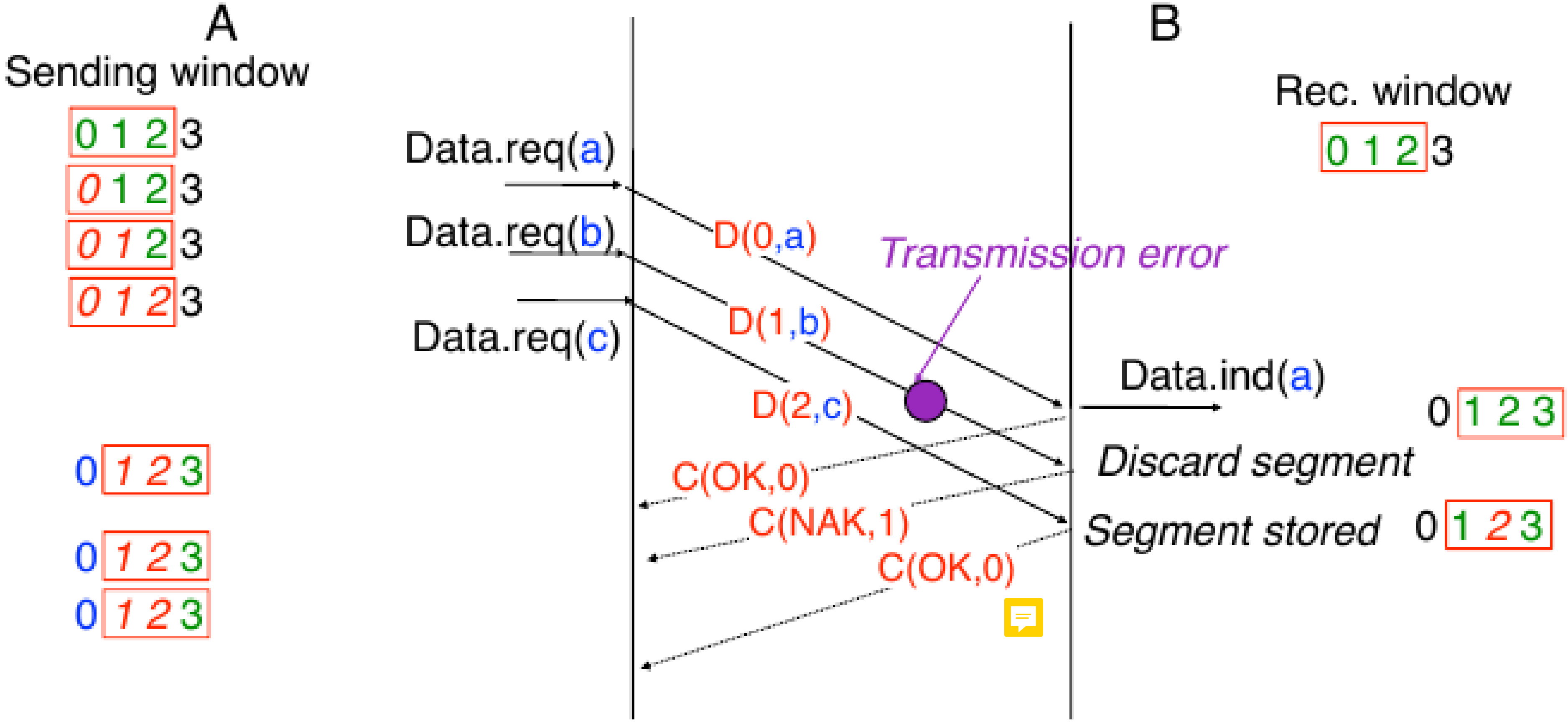
Sender

- Upon detection of an errored or lost segment, sender retransmits only this segment
  - may require one retransmission timer per segment

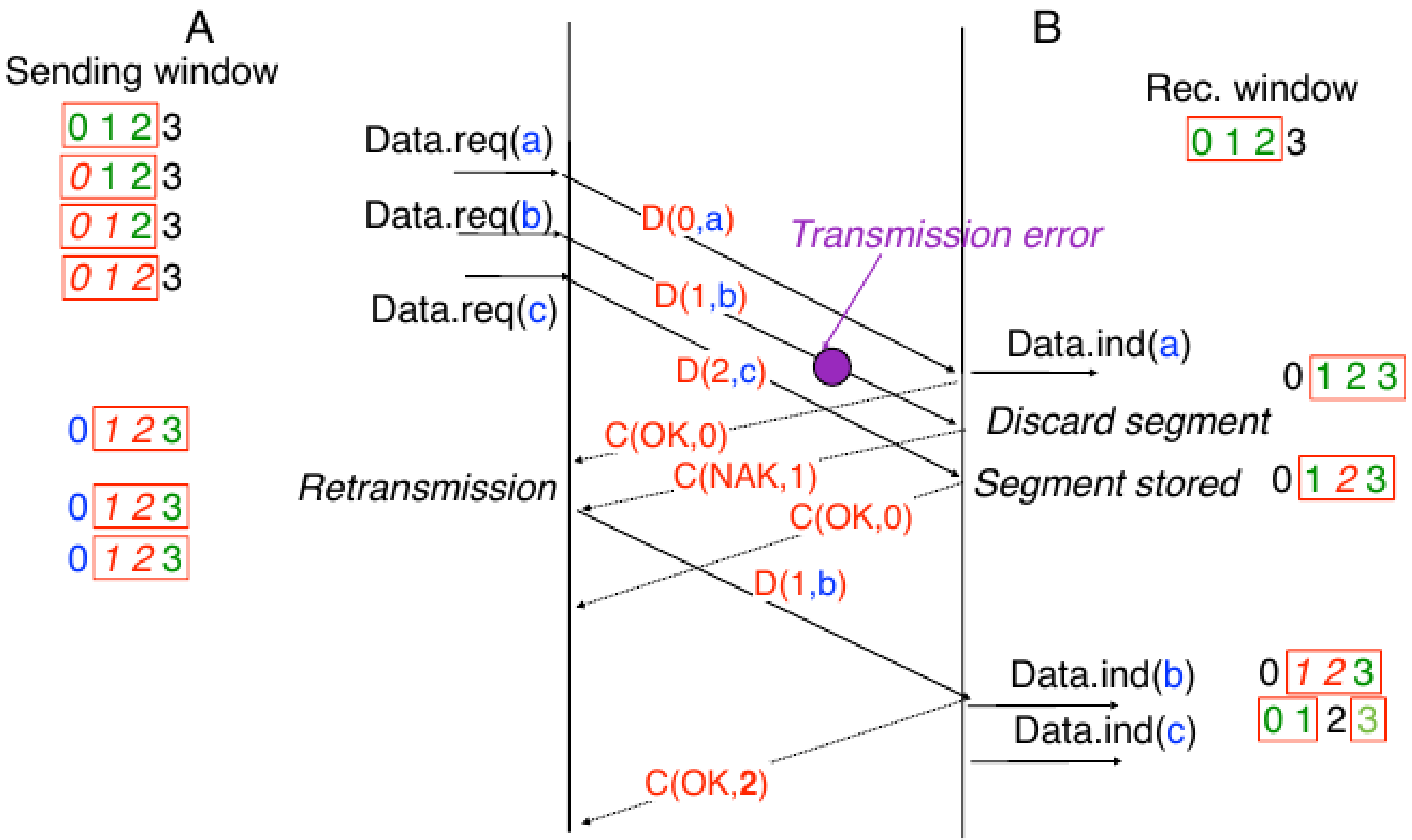
Selective Repeat



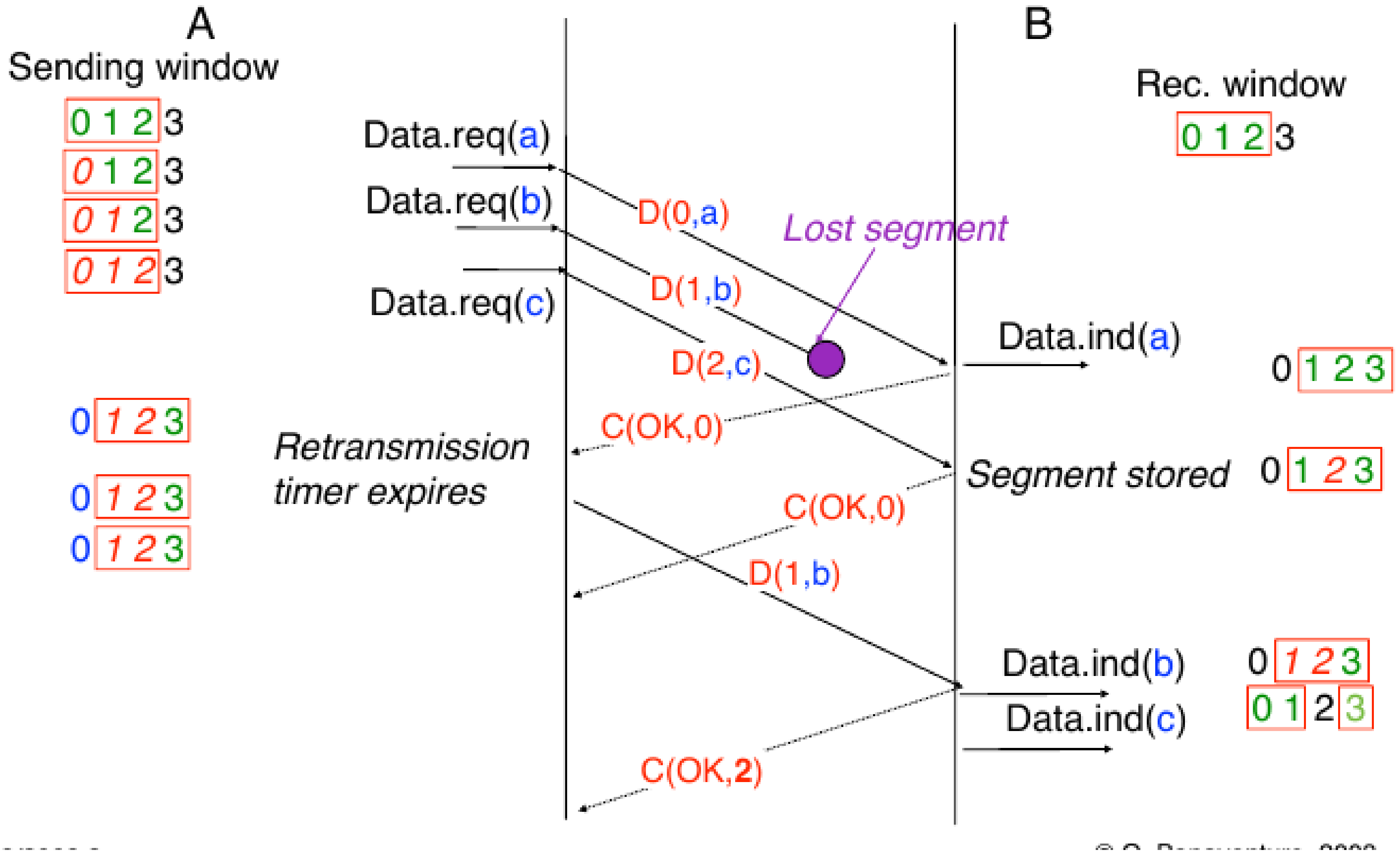
Selective Repeat



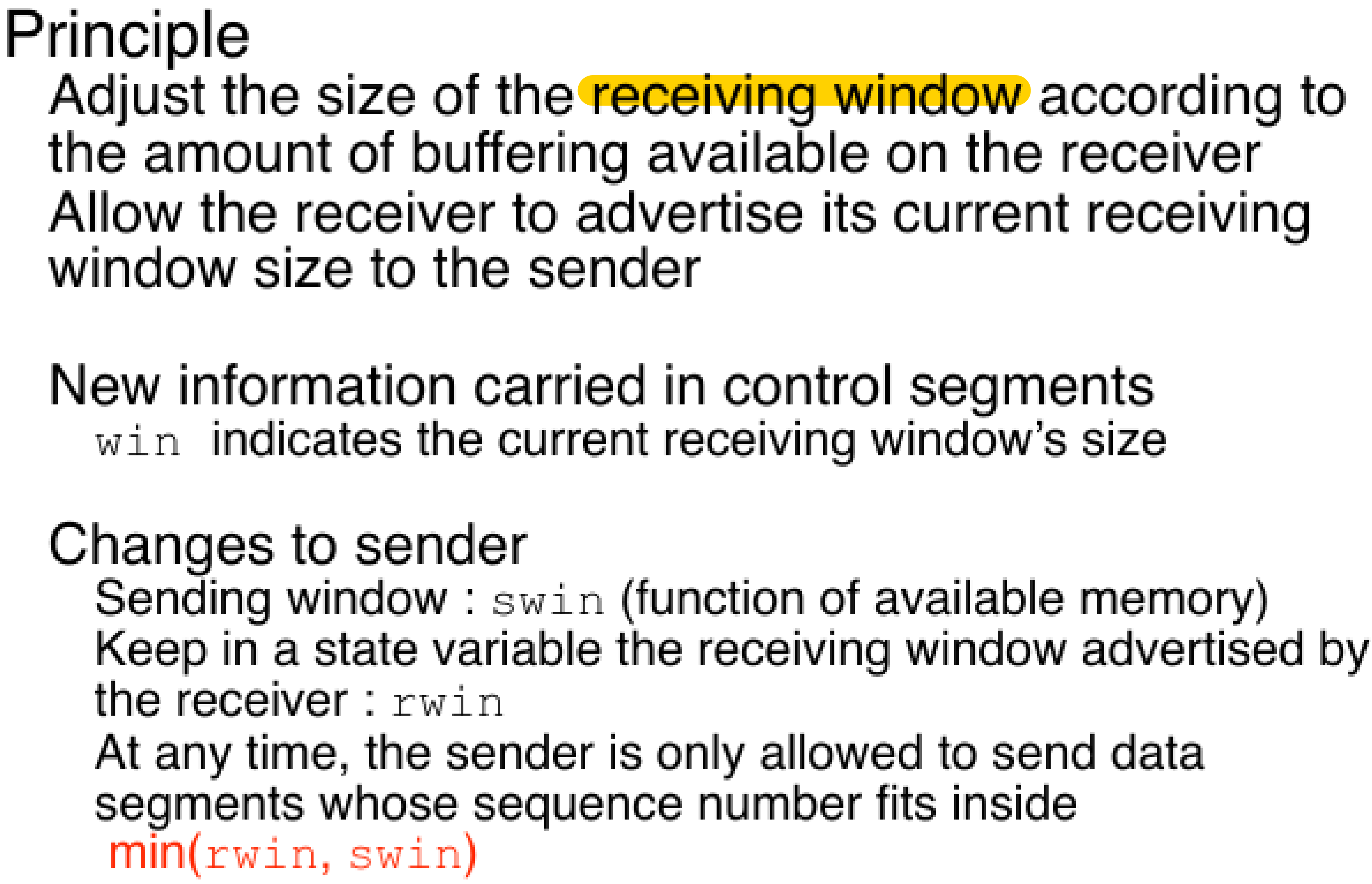
Selective Repeat



Selective Repeat



Window Size management



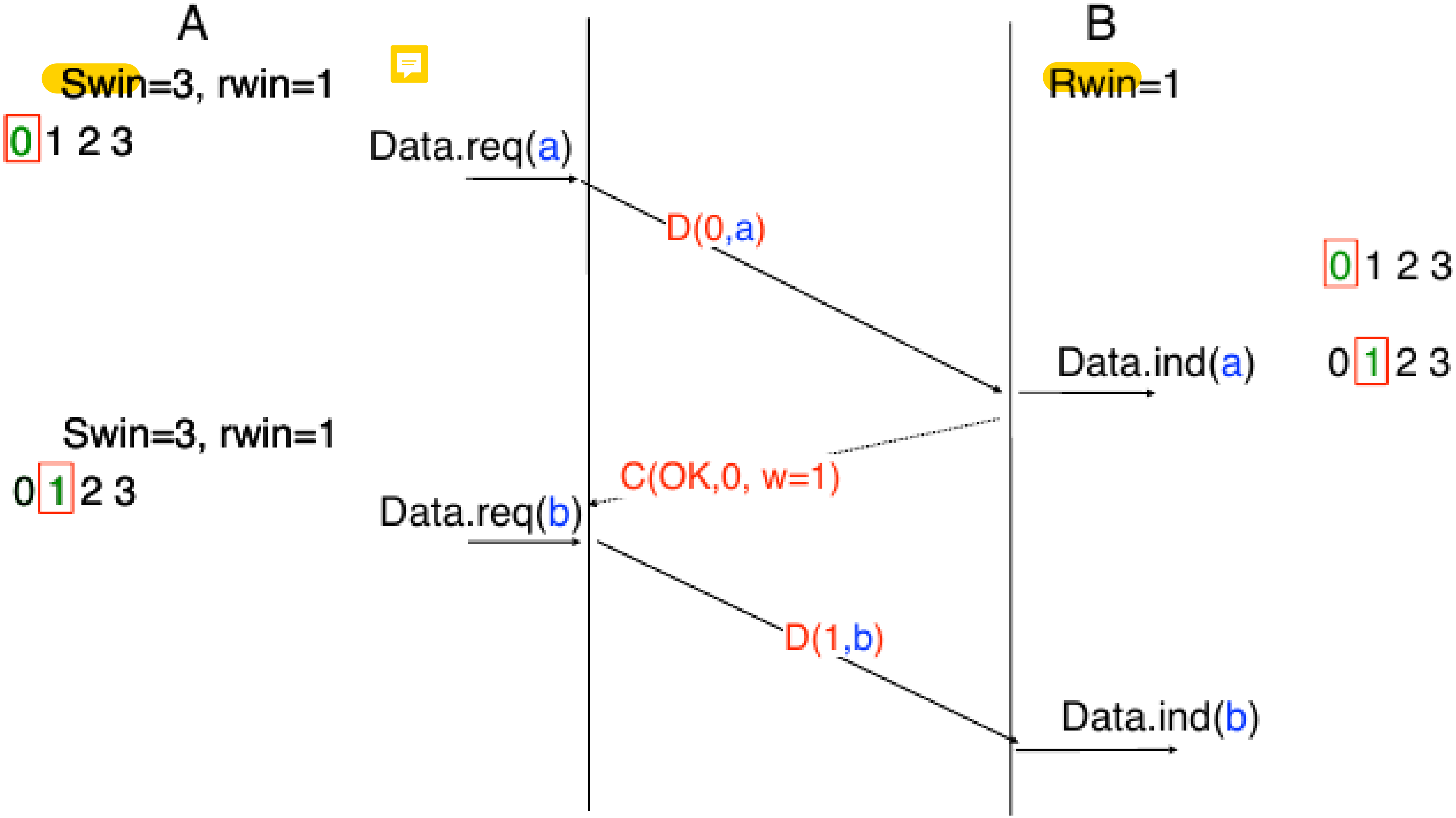
Principle

- Adjust the size of the receiving window according to the amount of buffering available on the receiver
- Allow the receiver to advertise its current receiving window size to the sender

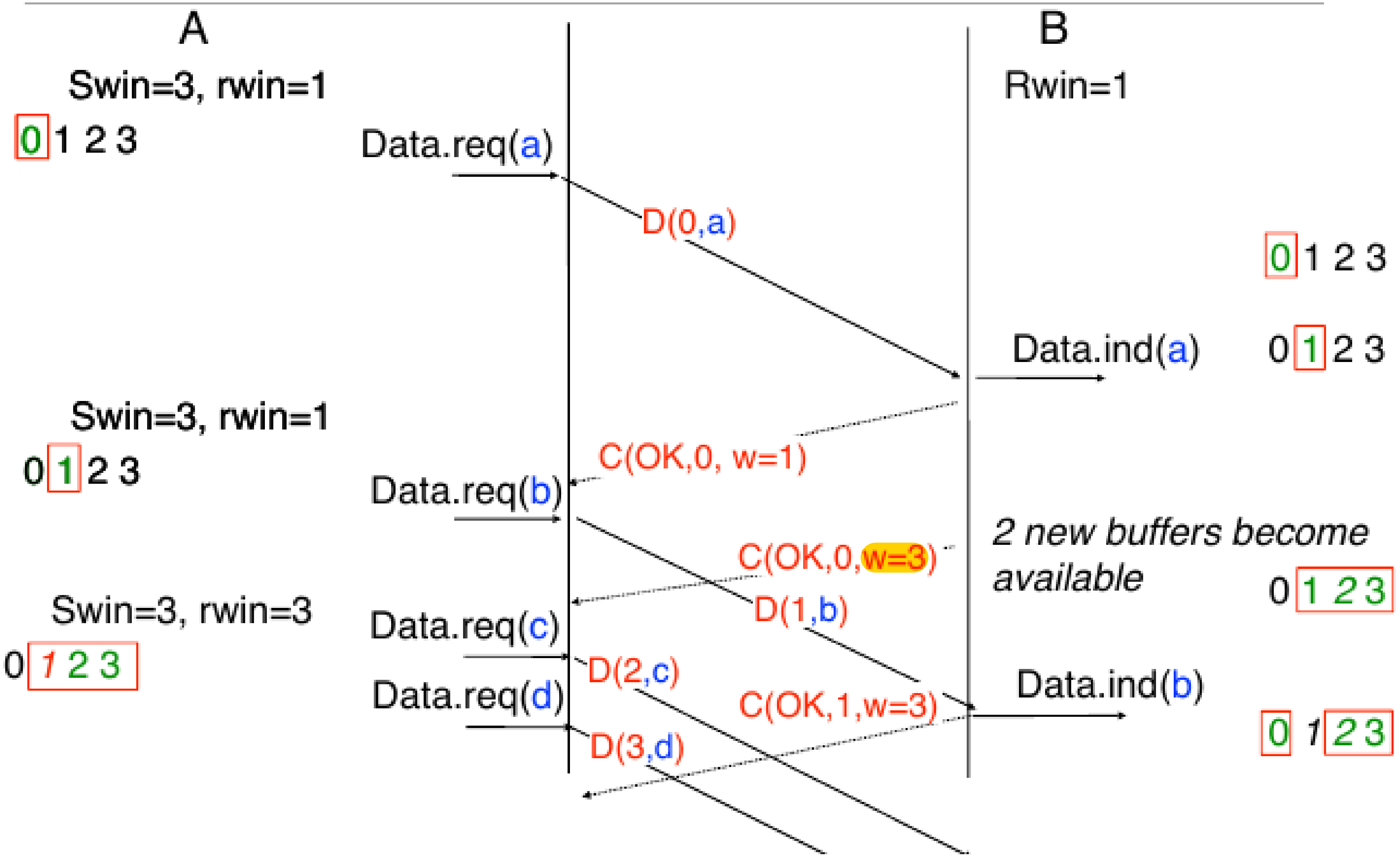
- New information carried in control segments
  - `win` indicates the current receiving window’s size

- Changes to sender
  - Sending window : `swin` (function of available memory)
  - Keep in a state variable the receiving window advertised by the receiver : `rwin`
  - At any time, the sender is only allowed to send data segments whose sequence number fits inside min(`rwin`, `swin`)

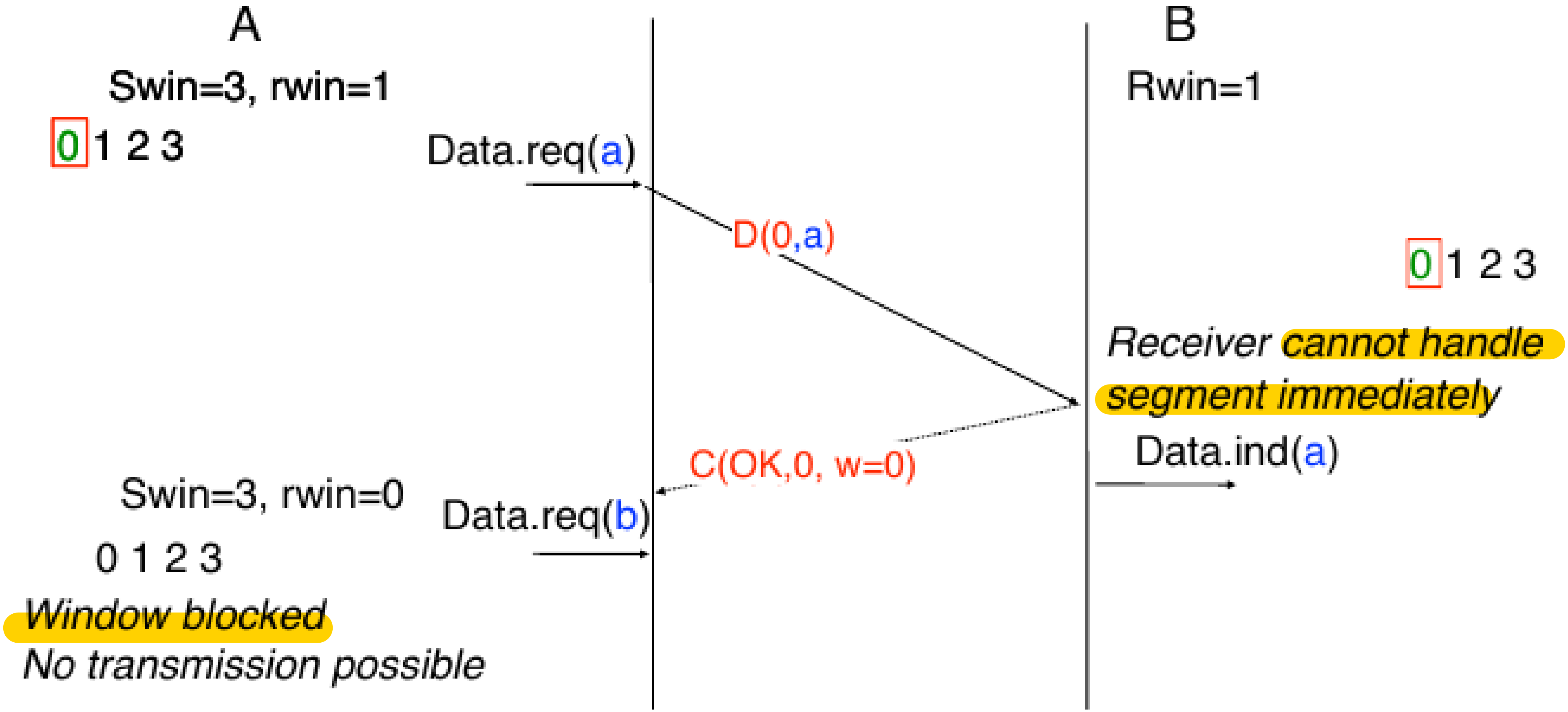
Window Size management



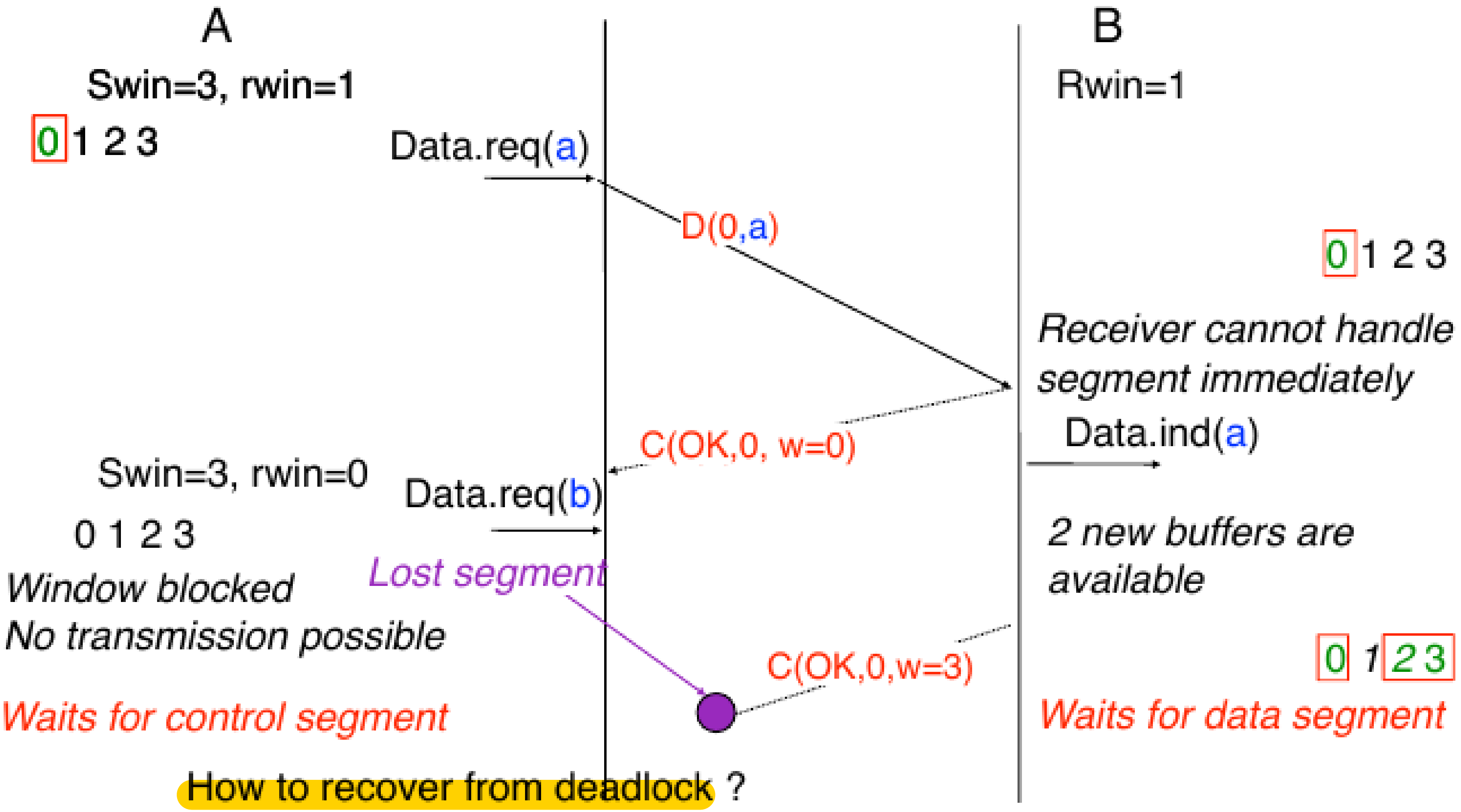
Window Size management



Windows Size management



Windows Size management



A

Swin=3, rwin=1

0 1 2 3

Data.req(a)

D(0,a)

B

Rwin=1

0 1 2 3

Receiver cannot handle segment immediately

Data.ind(a)

C(OK,0, w=0)

Swin=3, rwin=0

Data.req(b)

0 1 2 3

Window blocked
No transmission possible

Lost segment

2 new buffers are available

C(OK,0,w=3)

0 *1* 2 3

Waits for control segment

Waits for data segment

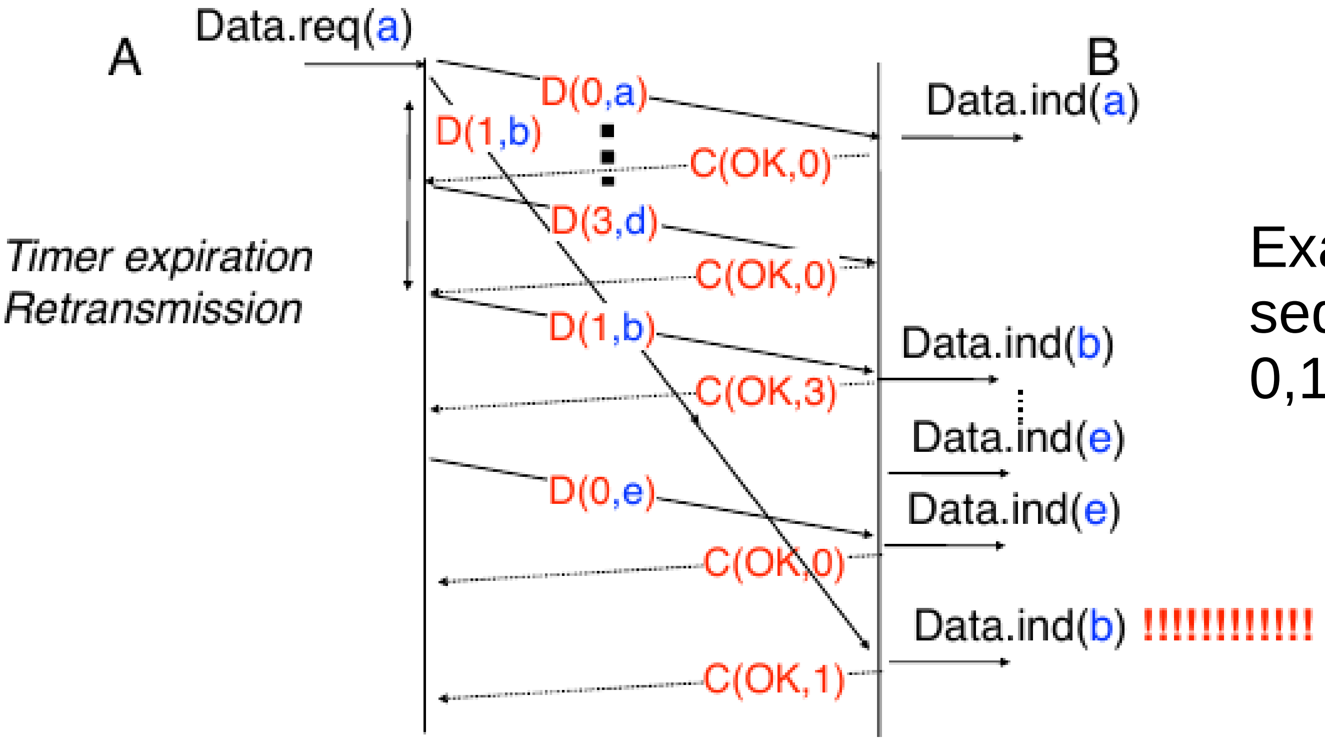
How to recover from deadlock ?
Persitence timer on receiver, resend control segment after timer expiration

Transmition Errors

- Errors in Payload – CheckSum (CRC)
- Paquets can be lost – Timer
- Paquets can arrive out-of-order – Sequence Number
- Paquets can be duplicated – Sequence Number

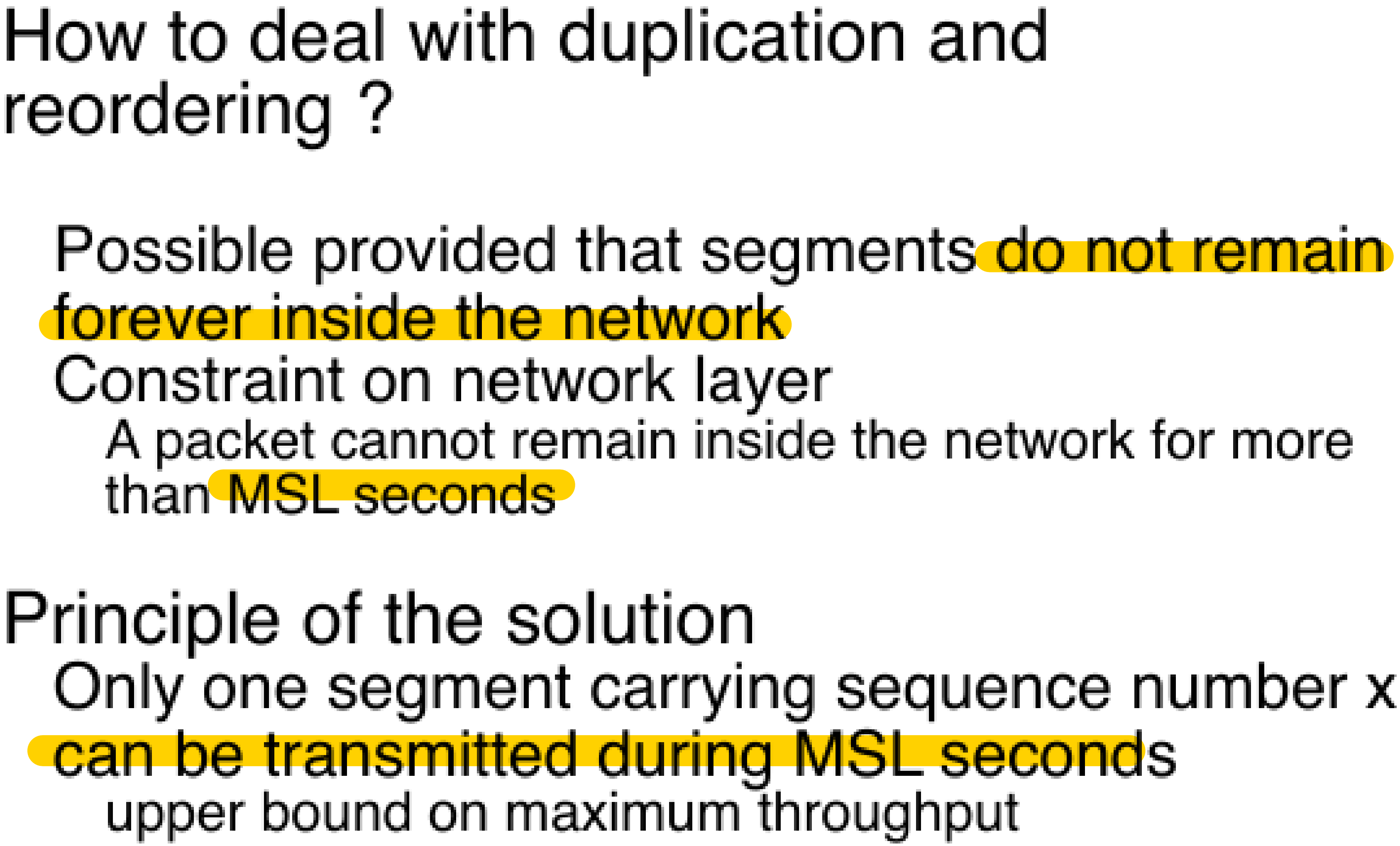
Duplication and reordering

- Because the sequence numbers are limited, a paquet that is late can have the right sequence number, despite not being in-order:



Example Possible sequence numbers: 0,1,2,3

Duplication and Reordering



How to deal with duplication and reordering ?

- Possible provided that segments do not remain forever inside the network
- Constraint on network layer
  - A packet cannot remain inside the network for more than MSL seconds

Principle of the solution

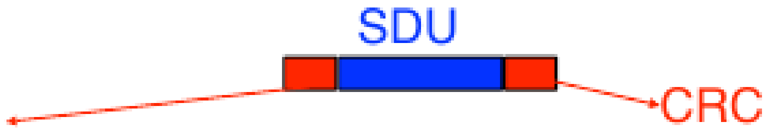
- Only one segment carrying sequence number x can be transmitted during MSL seconds
  - upper bound on maximum throughput

Bi-directional flow

How can we allow both hosts to transmit data ?

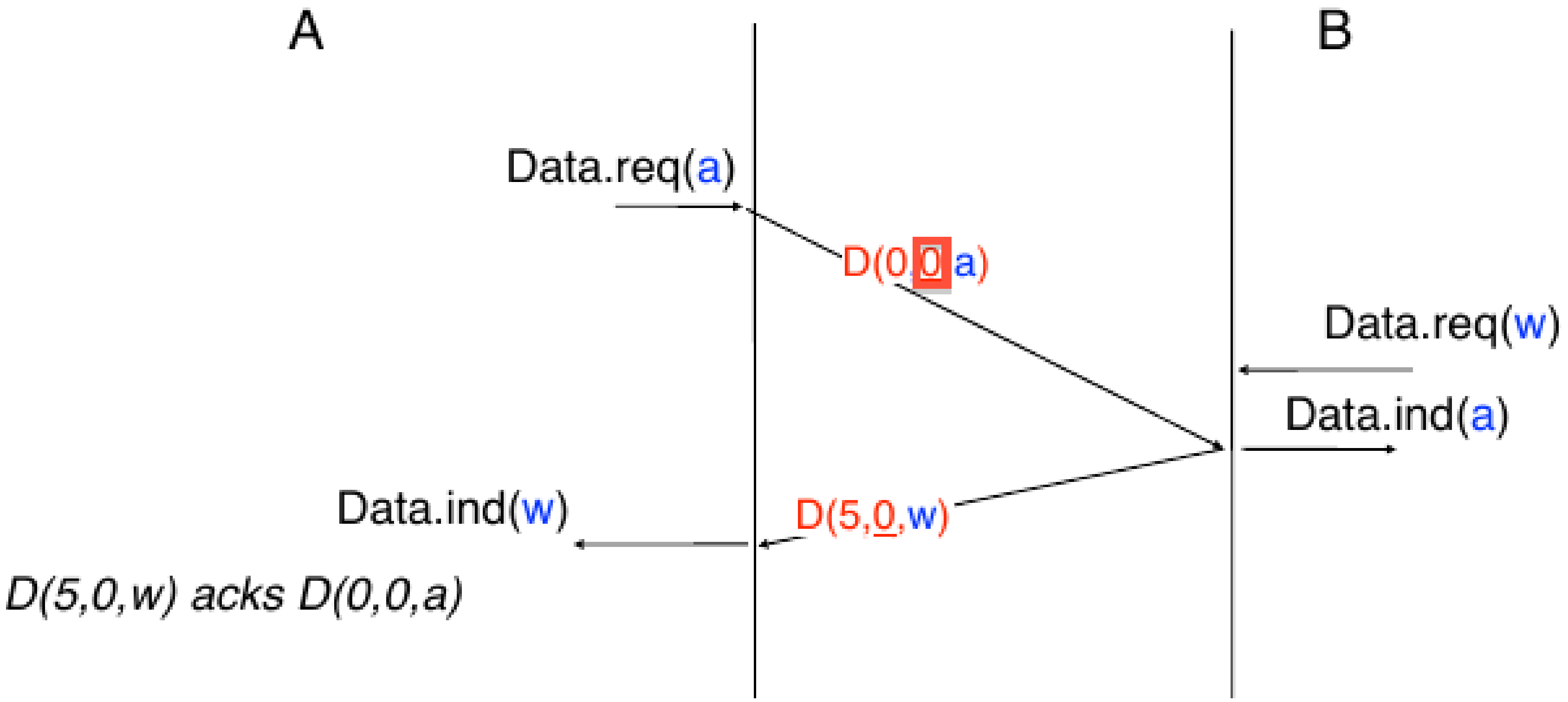
Principle

- Each host sends both control and data segments
  - Piggybacking
    - Place control fields inside the data segments as well (e.g. window, ack number) so that data segments also carry control information
    - Reduces the transmission overhead



Type : D or C
Seq : segment's sequence number
Ack : sequence number of the last received in-order segment

Bi-Directional Flow


A
B
Data.req(a)
D(0,0,a)
Data.req(w)
Data.ind(a)
Data.ind(w)
D(5,0,w)
D(5,0,w) acks D(0,0,a)