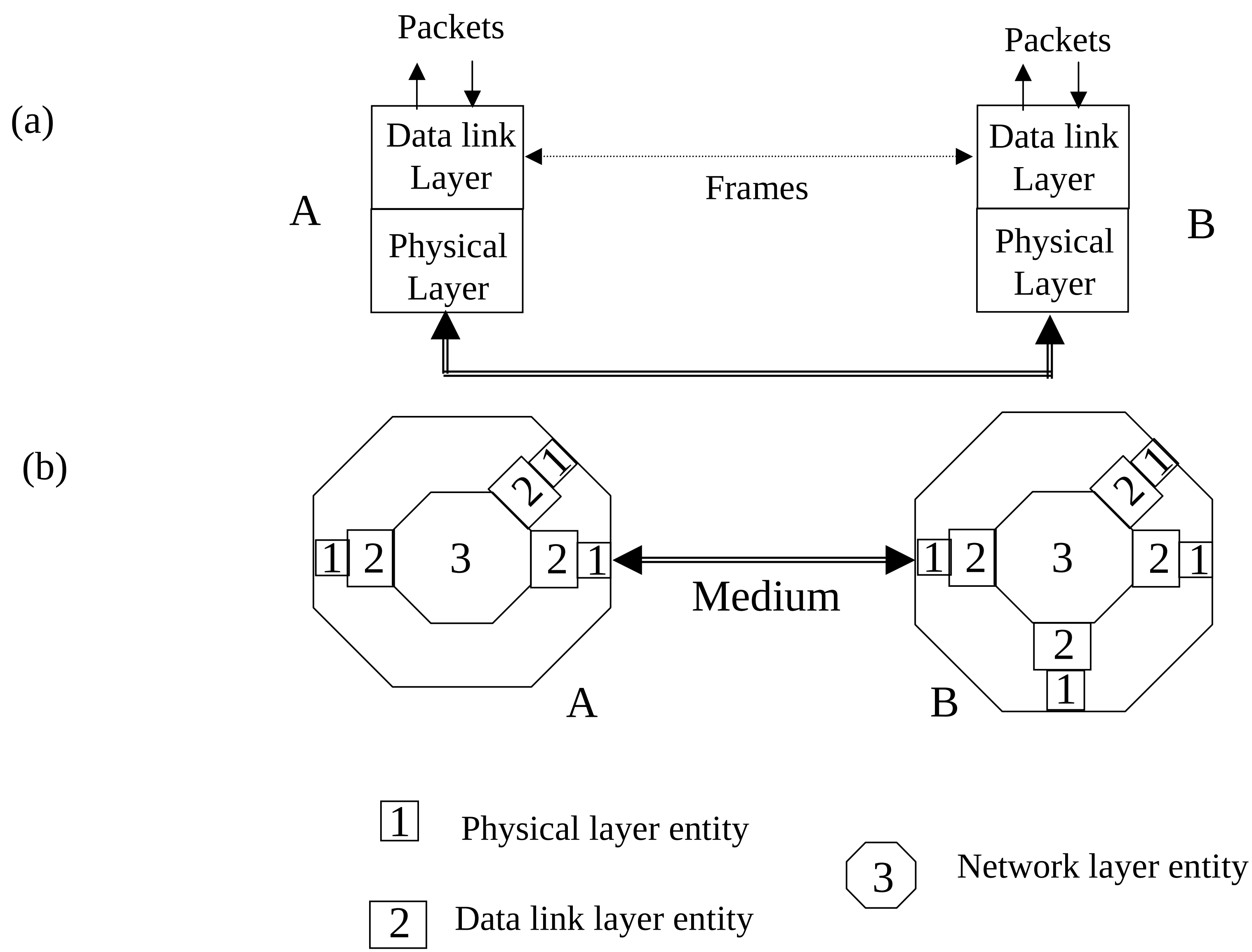
Data Link Layer

Data Link Layer

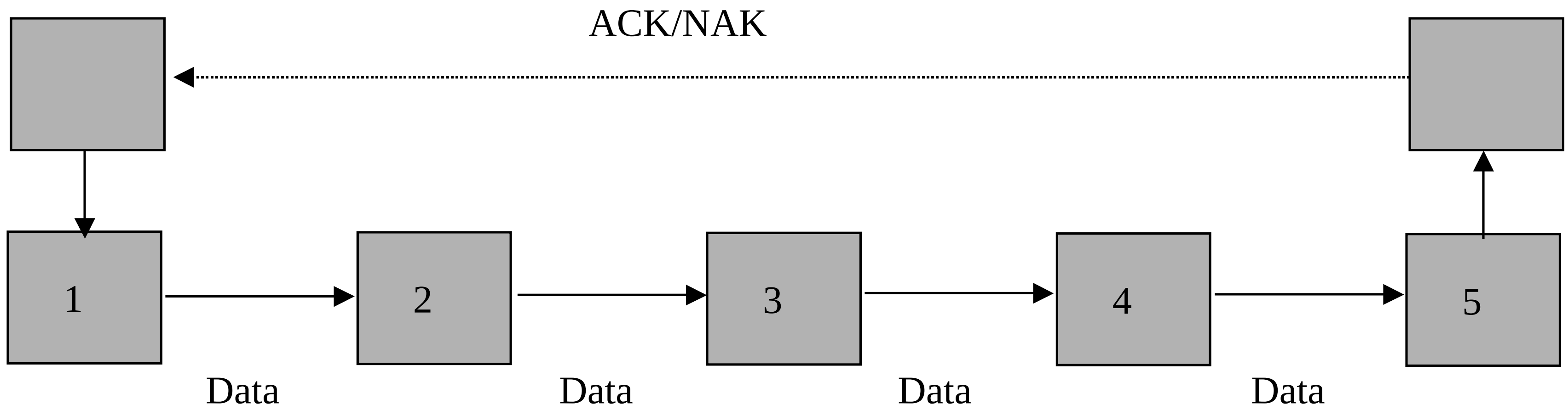
- Provides a *well-defined service interface* to the network layer.
- Determines how the bits of the physical layer are grouped into frames *(framing)*.
- Deals with transmission errors *(CRC and ARQ)*.
- Regulates the flow of frames.
- Performs general link layer management.



(a)
Packets
Packets
A
Data link Layer
Physical Layer
Frames
Data link Layer
Physical Layer
B
(b)
1
2
3
2
1
2
1
Medium
1
2
3
2
1
2
1
2
1
A
B
1
Physical layer entity
3
Network layer entity
2
Data link layer entity

Figure 5.2

End to End



Hop by Hop

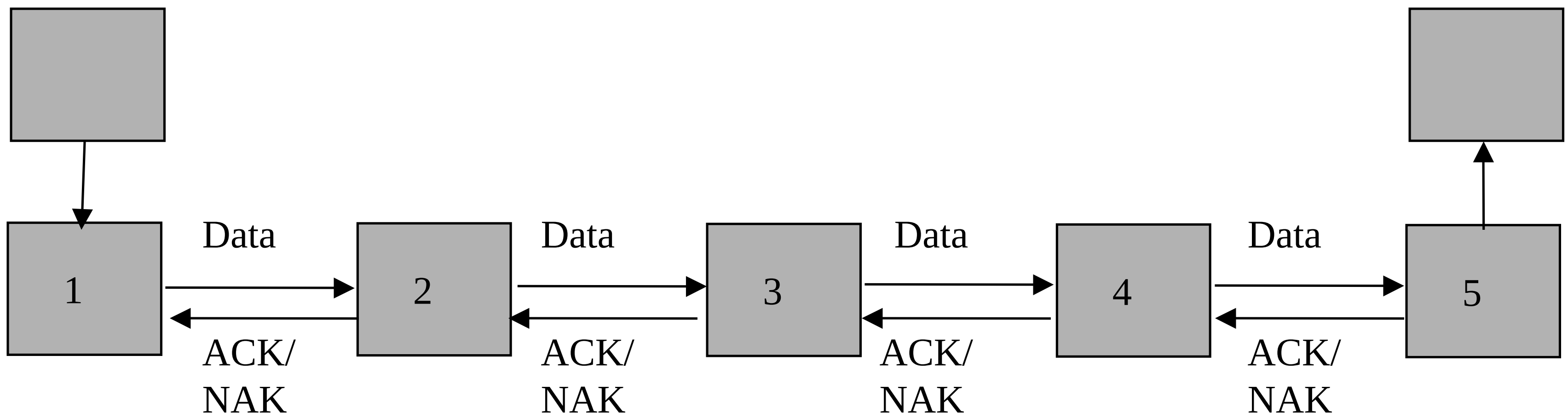
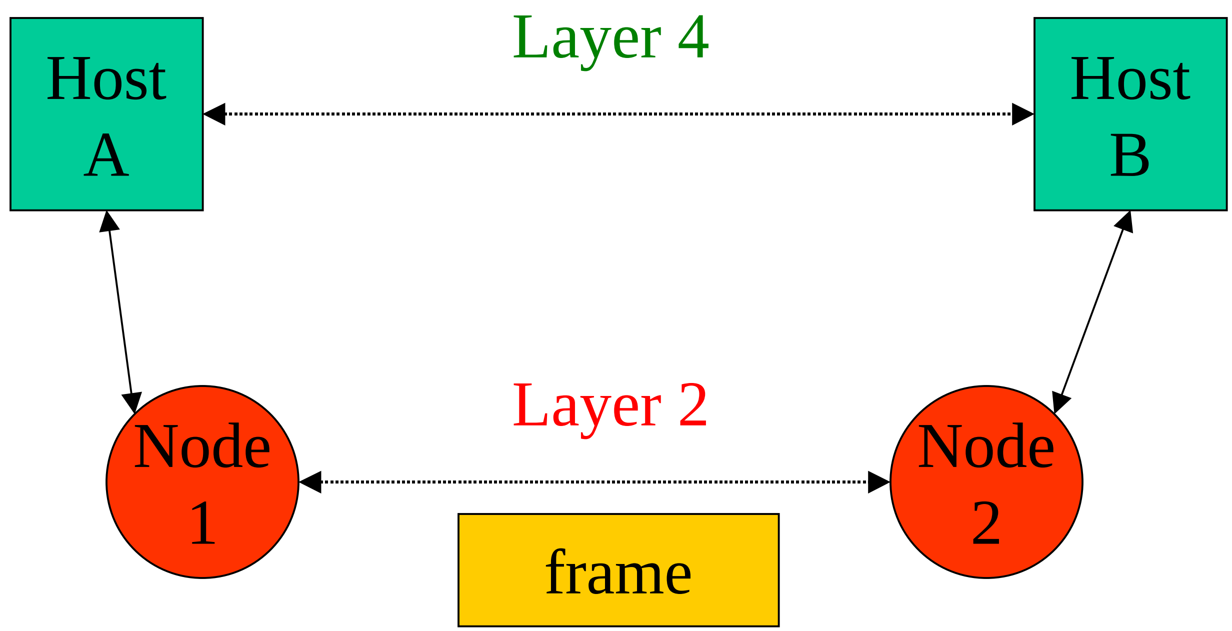


Figure 5.7

Tanenbaum’s Data Link Treatment

- Concerned with communication between two adjacent nodes in the subnet (IMP to IMP).
- Assumptions:
  - Bits delivered in the order sent
  - **Rigid interface** between the HOST and the node ➔ *the communications policy and the Host protocol (with OS effects) can evolve separately*.
  - uses a simplified model



Data Link Layer Model

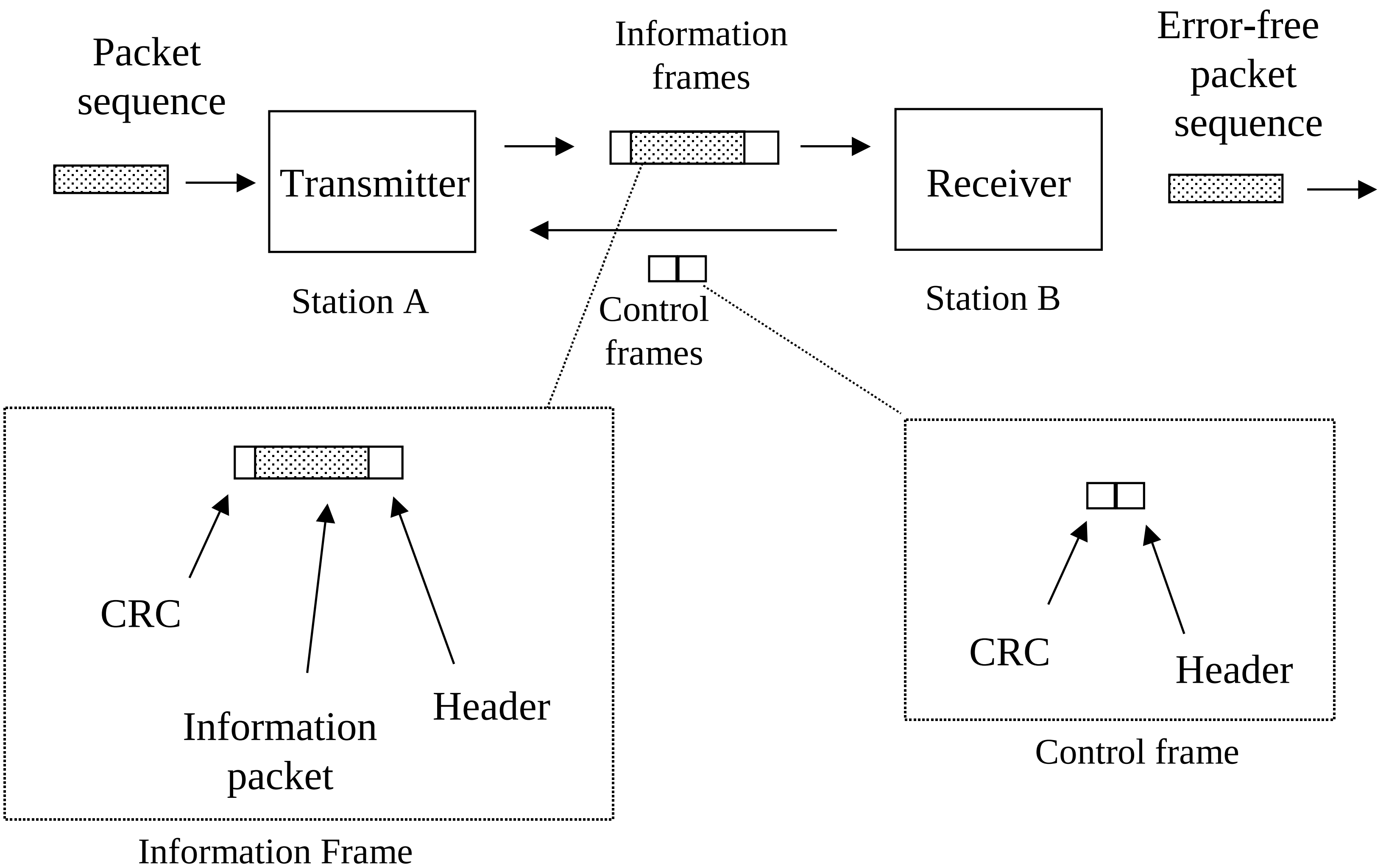
Assume Host has *infinite* supply of messages.

Node constructs frame from a single packet message.

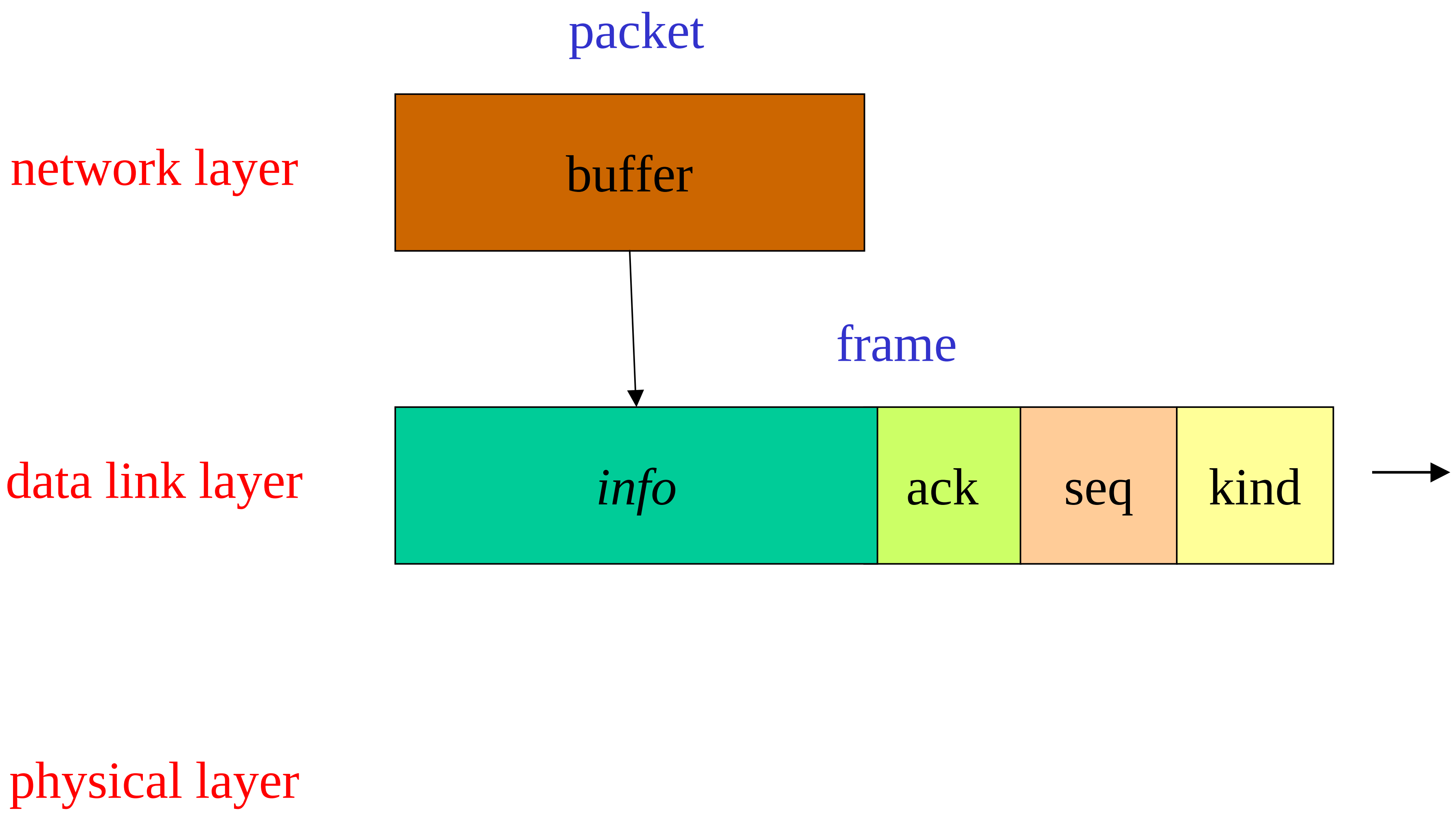
Checksum is automatically appended in the hardware.

Protocols are developed in increasing complexity to help understand the data link layer issues.

Basic Elements of ARQ



 Leon-Garcia & Widjaja: *Communication Networks* Figure 5.8


packet
network layer
buffer
frame
data link layer
info
ack
seq
kind
physical layer

Protocol 3 (PAR) Positive ACK with Retransmission [Old Tanenbaum Version]

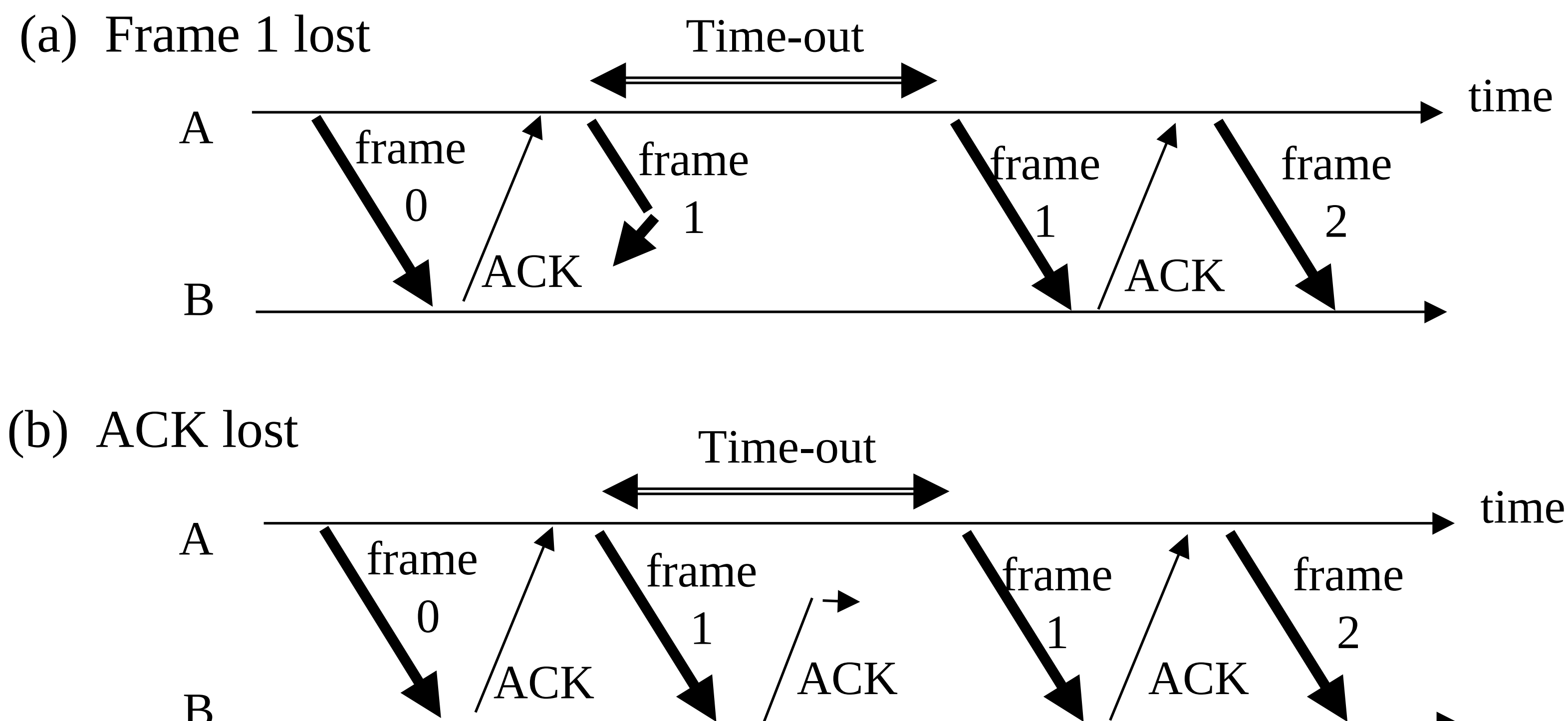
```
#define MAX_SEQ 1
typedef enum {frame_arrival, cksum_err, timeout} event_type;
include “protocol.h”

void sender_par (void)
{
  seq_nr  next_frame_to_send;
  frame s;
  packet buffer;
 event_type event;
 next_frame_to_send = 0;
 from_network_layer (&buffer);
 while (true)
 {     s.info = buffer;
       s.seq = next_frame_to_send;
       to_physical_layer (&s);
       start_timer (s.seq);
       wait_for_event(&event);
       if (event == frame_arrival) {
                 from_network_layer (&buffer);
                 inc (next_frame_to_send);
       }
 }
}
```

Protocol 3 (PAR) Positive ACK with Retransmission [Old Tanenbaum Version]

```
void receiver_par (void)
{
  seq_nr  next_frame_to_send;
  frame r, s;
 event_type event;
 frame_expected = 0;
 while (true)
 {    wait_for_event (&event);
      if (event == frame_arrival)
      {         from_physical_layer (&r);
                if (r.seq == frame_expected) {
                          to_network_layer(&r.info);
                          inc (frame_expected);
                }
                to_physical_layer (&s);              /* Note – no sequence number on ACK */
       }
 }
}
```

Ambiguities with Stop-and-Wait [unnumbered frames]

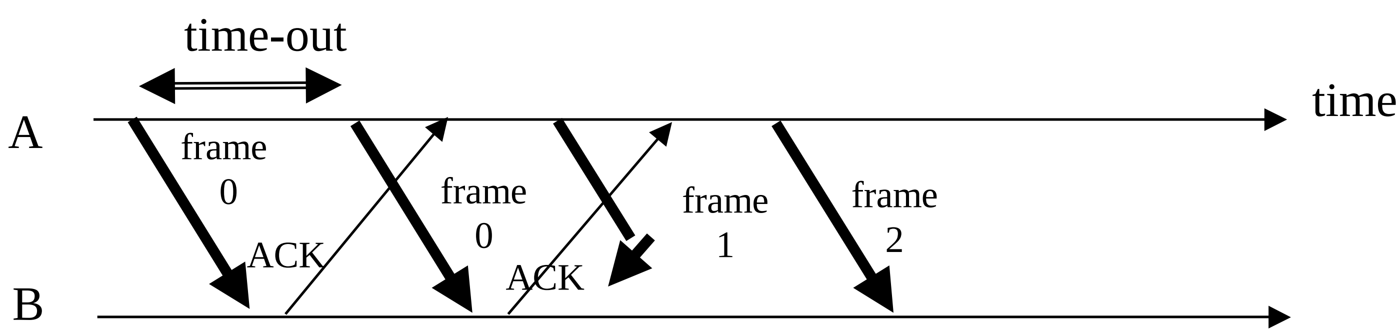


In parts (a) and (b) transmitting station A acts the same way, but part (b) receiving station B accepts frame 1 twice.

 Leon-Garcia & Widjaja: *Communication Networks* Figure 5.9

PAR [OLD] problem

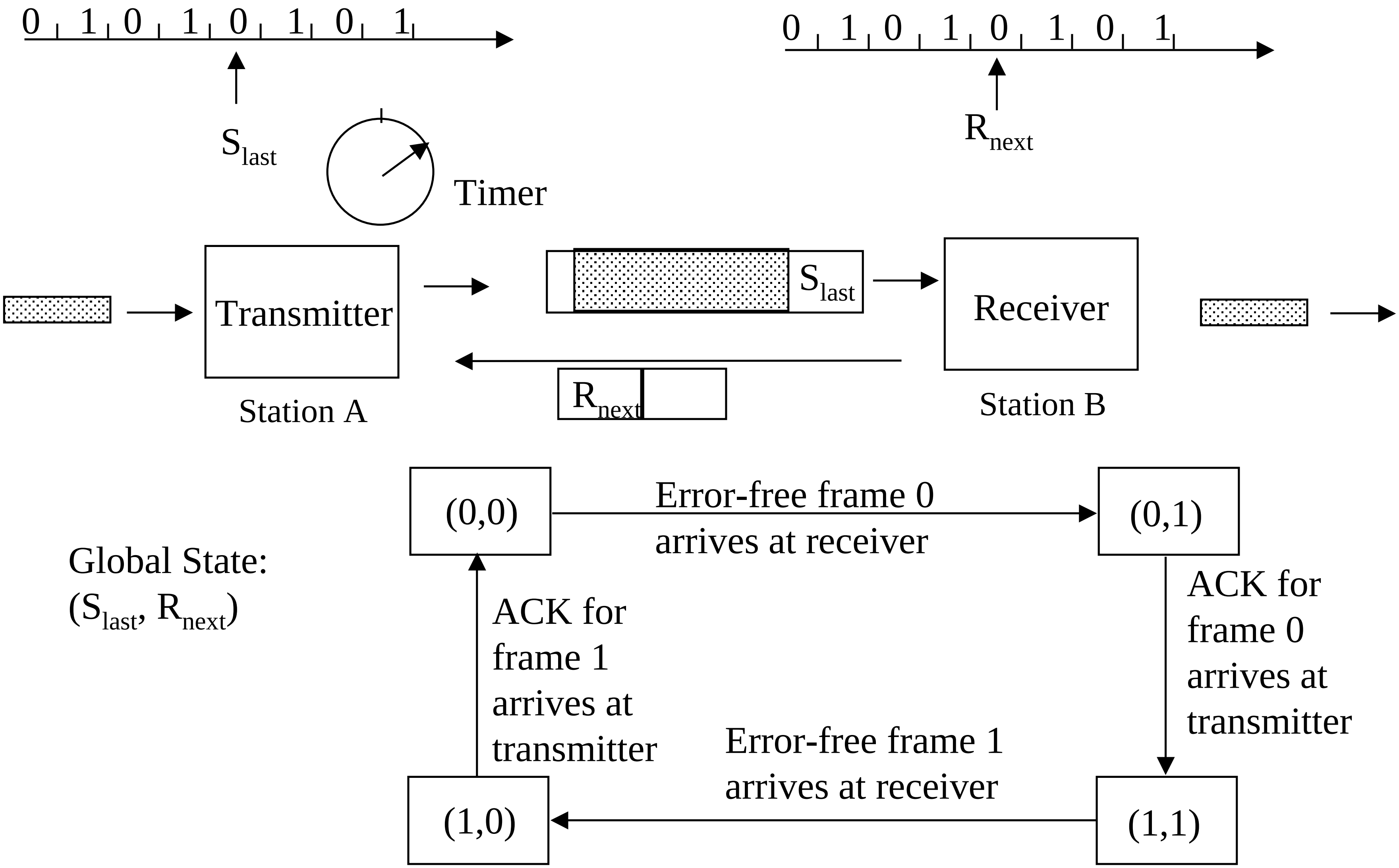
Ambiguities when ACKs are not numbered



Transmitting station A misinterprets duplicate ACKs

Leon-Garcia & Widjaja: *Communication Networks*
Figure 5.10

State Machine for Stop-and-Wait



Leon-Garcia & Widjaja: *Communication Networks*
Figure 5.11

Sliding Window Protocols

[Tanenbaum]

- Must be able to transmit data in <u>both</u> directions.
- Choices for utilization of the reverse channel:
  - mix DATA frames with ACK frames.
  - *Piggyback* the ACK
    - Receiver waits for DATA traffic in the opposite direction.
    - Use the ACK field in the frame header to send *sequence number* of frame being ACKed.
  - ➔ better use of the channel capacity.

Sliding Window Protocols

- ACKs introduce a new issue – how long does receiver wait before sending ONLY an ACK frame.
  - ➔We need an *ACKTimer!!*
  - ➔ sender *timeout period* needs to set longer.
- The protocol must deal with the premature timeout problem and be “robust” under pathological conditions.

Sliding Window Protocols

Each outbound frame must contain a *sequence number*. With *n* bits for the sequence number field, maxseq = 2**n - 1 and the numbers range from 0 to maxseq.

Sliding window :: sender has a window of frames and maintains a list of consecutive sequence numbers for frames that it is permitted to send without waiting for ACKs.

receiver has a window that is a list of frame sequence numbers it is permitted to accept.

Note – sending and receiving windows do NOT have to be the same size.

Windows can be fixed size or dynamically growing and shrinking.

Sliding Window Protocols

- Host is oblivious, message order at transport level is maintained.

sender's window :: frames sent but not yet ACKed.

- – new packets from the Host cause the upper edge inside sender window to be incremented.
- – ACKed frames from the receiver cause the lower edge inside window to be incremented.

Sliding Window Protocols

- All frames in the sender's window must be saved for possible retransmission and we need one timer per frame in the window.
- If the maximum sender window size is B, the sender needs B buffers.
- If the sender window gets full (i.e., reaches its maximum window size, the protocol must shut off the Host (the network layer) until buffers become available.

Sliding Window Protocols

receiver window

- Frames received with sequence numbers outside the *receiver window* are not accepted.
- The receiver window size is normally *static*.

  The set of acceptable sequence numbers is rotated as "acceptable" frames arrive.

a receiver window size = 1 ➔ *the protocol only accepts frames in order.*

There is referred to as Go Back N.

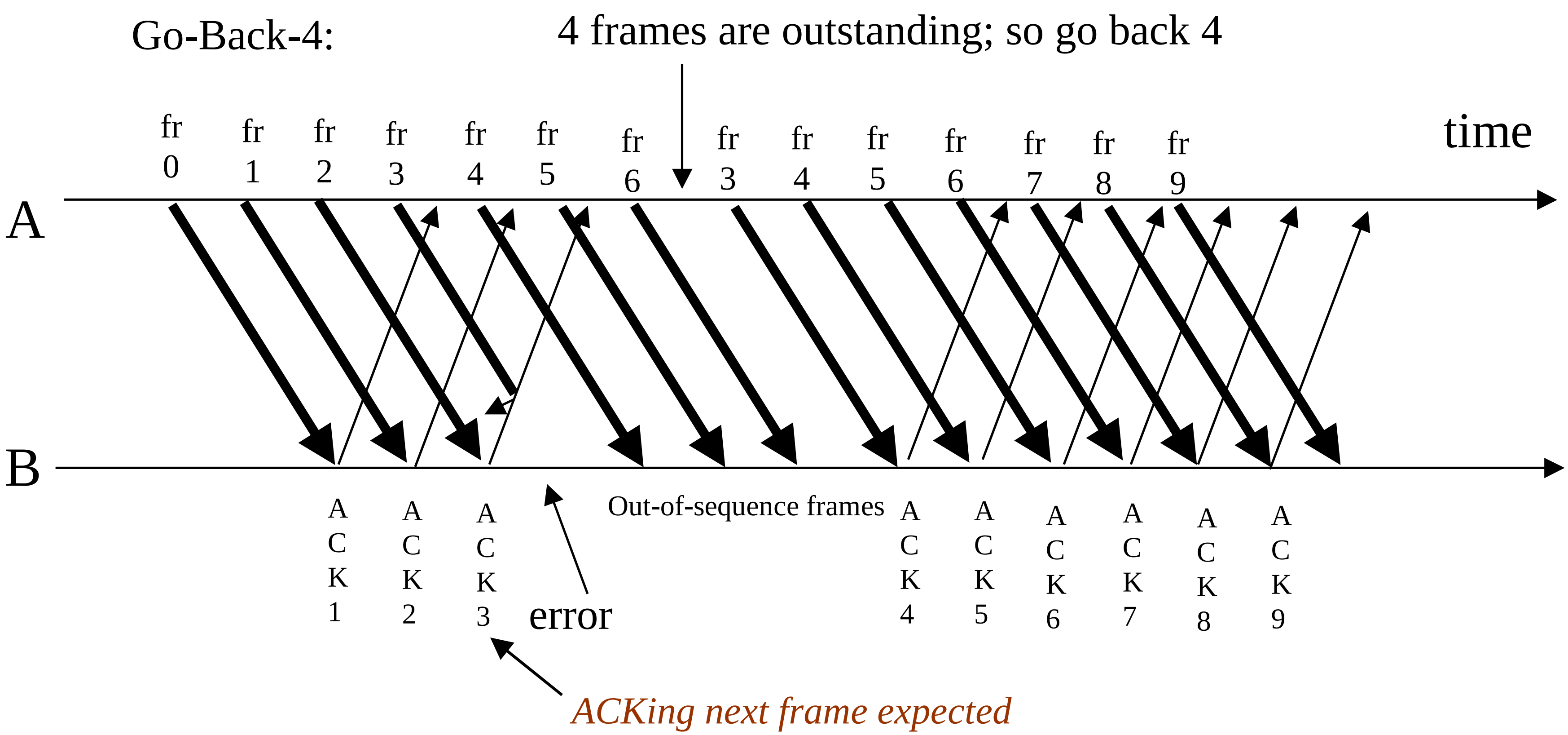
Standard Ways to ACK

1. ACK sequence number indicates *the last frame successfully received.*
2. ACK sequence number indicates *the next frame the receiver expects to receive.*

Both of these can be strictly *individual ACKs or represent* *cumulative ACKing.*

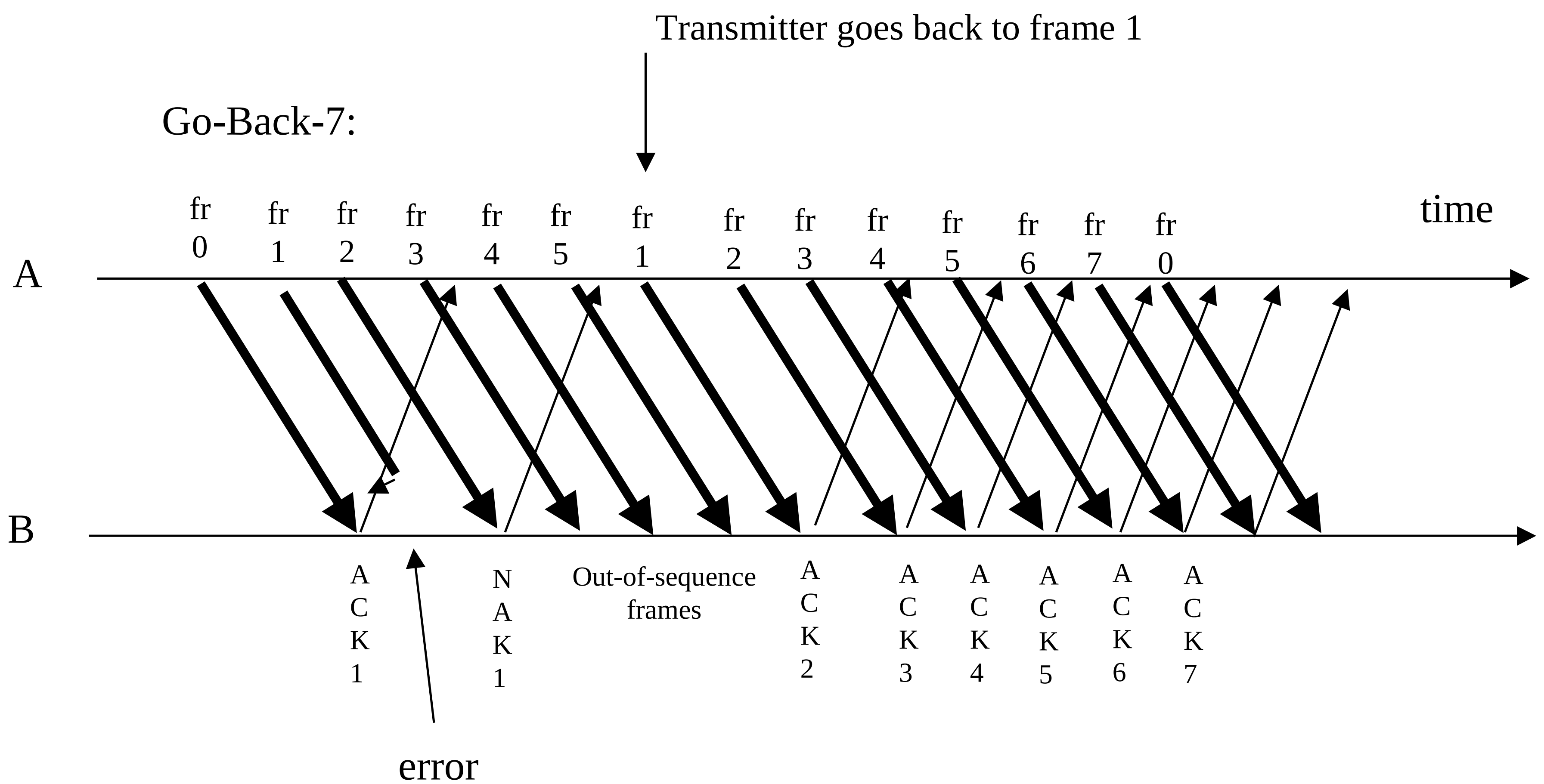
Cumulative ACKing is the most common technique.

Go Back N



 Leon-Garcia & Widjaja: *Communication Networks* Figure 5.13

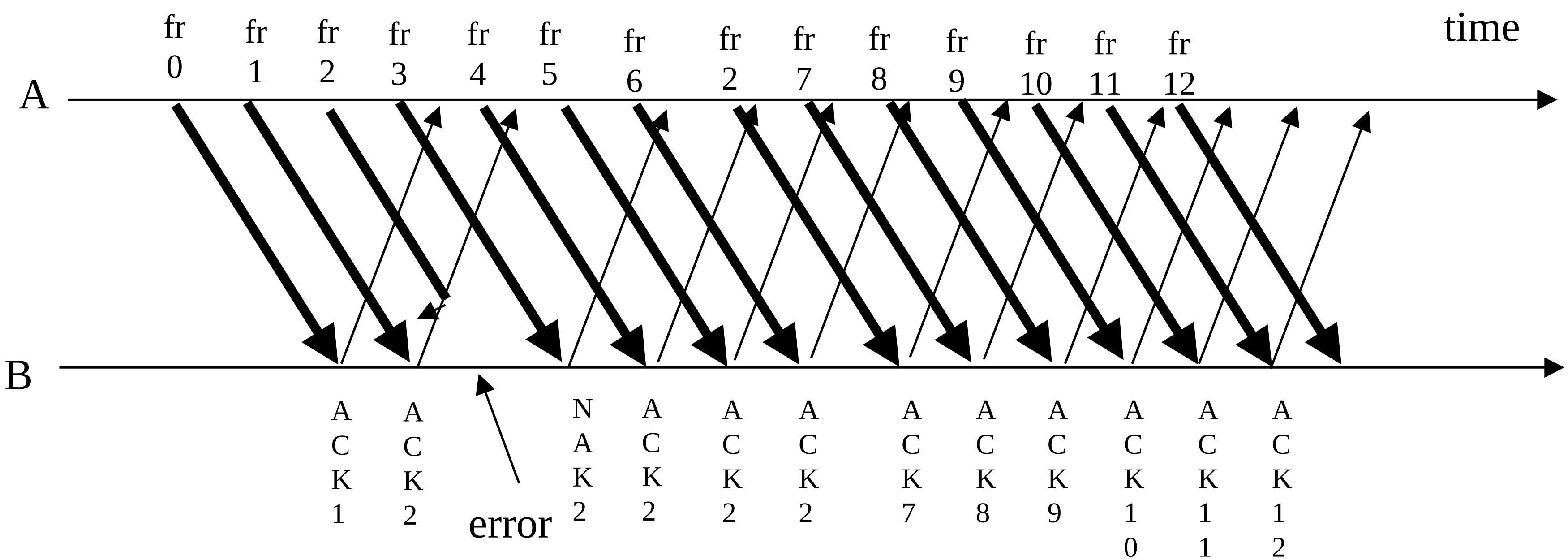
Go Back N
with NAK error recovery



Leon-Garcia & Widjaja: *Communication Networks*

Figure 5.17

Selective Repeat
with NAK error recovery



 Leon-Garcia & Widjaja: *Communication Networks* Figure 5.21