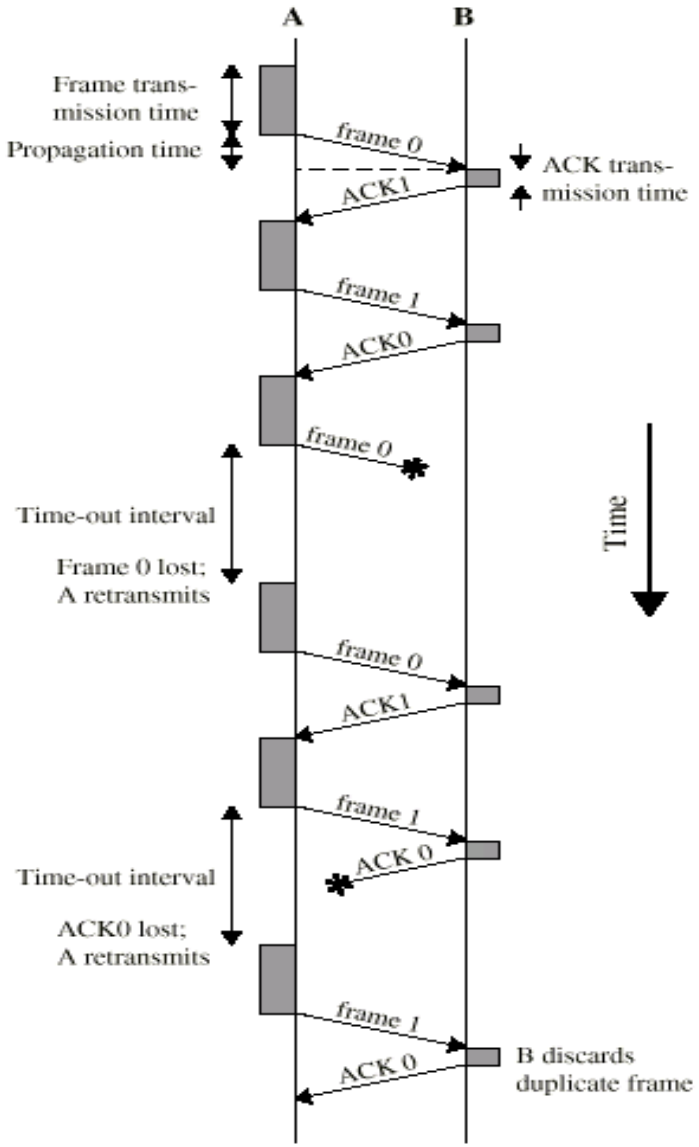
Data Link Control

Flow Control for NOISY CHANNELS

- *Although the Stop-and-Wait Protocol gives us an idea of how to add flow control, noiseless channels are nonexistent.*
- *3 protocols that use error control*
  - Stop-and-Wait Automatic Repeat Request
    Go-Back-N Automatic Repeat Request
    Selective Repeat Automatic Repeat Request
- Transmission time (t_{frame})
  - Time taken to emit all bits into medium
- Propagation time (t_{prop})
  - Time for a bit to traverse the link

Stop and Wait Operation

- Simple
- Inefficient



Further assumptions dropped

- The channel is noisy, frames may be damaged or lost
- Good scene : data frame reaches intact, ack sent back and received, next frame sent
- Bad scene :
  - Data frame damaged or lost ..hence no ack – sender times out and resends .. No problems
  - Data frame reaches intact but Ack lost .. Times out ..resends.. Receiver receives duplicate frames..Problem

Problem of Duplicate frames

- Solution :
  - Keep a sequence number for each frame to distinguish between the new frame and a duplicate frame.
- How large should be the sequence number? Or
- What should be minimum number of bits required for the sequence number?
  - The only ambiguity at the receiver is between two successive frames.. Say m and m+1 and never between m-1 and m +1 ..
  - Only after the ack for m-1 reaches back intact mth frame is sent.. And once mth frame reaches intact at the receiver's end .. Story of m-1th frame is over ..
  - However depending upon whether ack of mth frame reaches back intact or not either mth or m+1 th frame is sent .. hence only 1 bit is sufficient

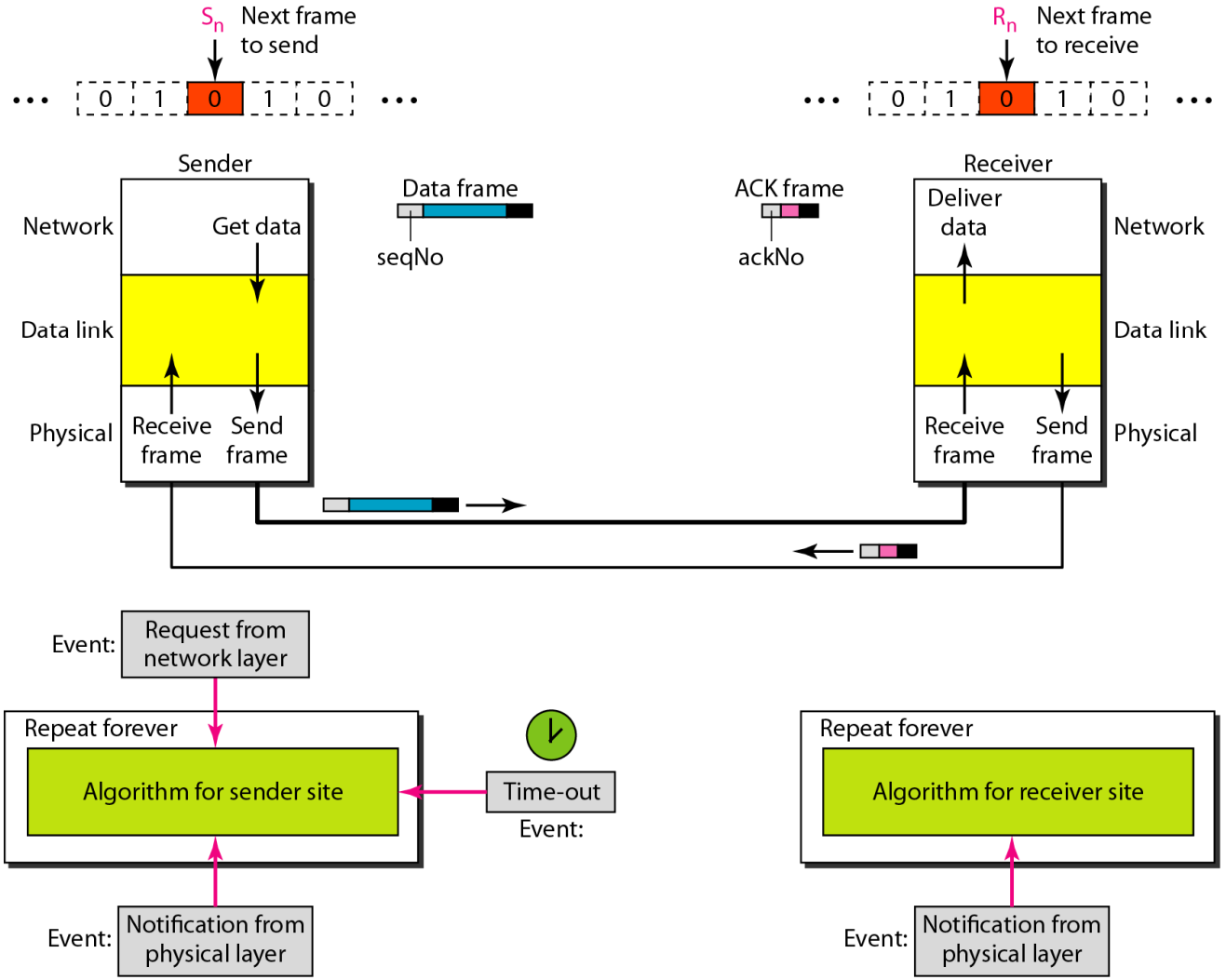
Sliding Window with Window Size W

- With a window size of 1 (stop-and-wait ARQ)
  - sender waits for an ACK before sending another frame
  - This protocol behaves identically to stop and wait for a noiseless channel!
- With a window size of W, the sender can transmit up to W frames before "being blocked"
- We call using larger window sizes **Pipelining**
  - **Go-Back-N ARQ**
  - **Selective Repeat ARQ**

Stop-and-Wait Automatic Repeat Request

- Error correction in Stop-and-Wait ARQ
  - keep a copy of the sent frame and retransmit the frame when the timer expires.
- Use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.
- Acknowledgment (ACK) number always announces the sequence number of the next frame expected (modulo-2)

Design of the Stop-and-Wait ARQ Protocol



Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ*

```
1  Sn = 0;                                  // Frame 0 should be sent first
2  canSend = true;                          // Allow the first request to go
3  while(true)                              // Repeat forever
4  {
5    WaitForEvent();                        // Sleep until an event occurs
6    if(Event(RequestToSend) AND canSend)
7    {
8      GetData();
9      MakeFrame(Sn);                       //The seqNo is Sn
10     StoreFrame(Sn);                      //Keep copy
11     SendFrame(Sn);
12     StartTimer();
13     Sn = Sn + 1;
14     canSend = false;
15   }
16   WaitForEvent();                        // Sleep
```

(continued)

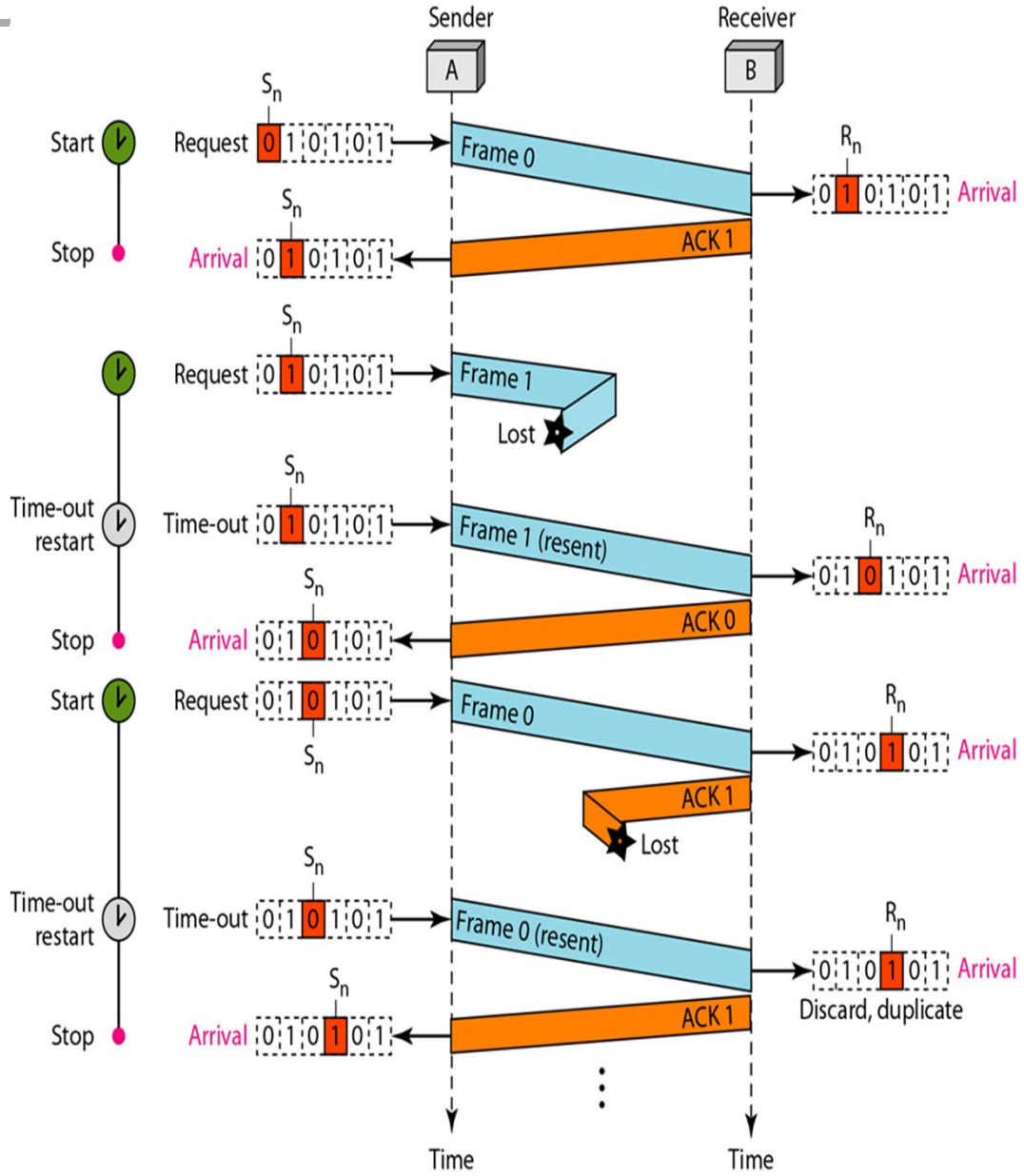
Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ* (continued)

```
17      if(Event(ArrivalNotification)          // An ACK has arrived
18      {
19        ReceiveFrame(ackNo);                 //Receive the ACK frame
20        if(not corrupted AND ackNo == Sn)    //Valid ACK
21          {
22            Stoptimer();
23            PurgeFrame(Sn-1);                //Copy is not needed
24            canSend = true;
25          }
26       }
27
28      if(Event(TimeOut)                      // The timer expired
29      {
30       StartTimer();
31       ResendFrame(Sn-1);                    //Resend a copy check
32      }
33  }
```

Algorithm 11.6 *Receiver-site algorithm for Stop-and-Wait ARQ Protocol*

```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7       ReceiveFrame();
8       if(corrupted(frame));
9          sleep();
10      if(seqNo == Rn)              //Valid data frame
11      {
12       ExtractData();
13        DeliverData();             //Deliver data
14        Rn = Rn + 1;
15      }
16       SendFrame(Rn);             //Send an ACK
17   }
18 }
```

Stop-and-Wait ARQ. Frame 0 is sent and acknowledged.
Frame 1 is lost and resent after the time-out. The resent frame 1 is acknowledged and the timer stops.
Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.



A Simplex Protocol for a Noisy

A positive acknowledgement with retransmission protocol.

```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;                        /* seq number of next outgoing frame */
  frame s;                                          /* scratch variable */
  packet buffer;                                    /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;                           /* initialize outbound sequence numbers */
  from_network_layer(&buffer);                      /* fetch first packet */
  while (true) {
      s.info = buffer;                              /* construct a frame for transmission */
      s.seq = next_frame_to_send;                   /* insert sequence number in frame */
      to_physical_layer(&s);                        /* send it on its way */
      start_timer(s.seq);                           /* if answer takes too long, time out */
      wait_for_event(&event);                       /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
          from_physical_layer(&s);                  /* get the acknowledgement */
          if (s.ack == next_frame_to_send) {
              stop_timer(s.ack);                    /* turn the timer off */
              from_network_layer(&buffer);          /* get the next one to send */
              inc(next_frame_to_send);              /* invert next_frame_to_send */
          }
      }
  }
}
```

Continued →

A Simplex Protocol for a Noisy Channel

```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);                  /* possibilities: frame_arrival, cksum_err */
       if (event == frame_arrival) {            /* a valid frame has arrived. */
            from_physical_layer(&r);            /* go get the newly arrived frame */
            if (r.seq == frame  expected) {     /* this is what we have been waiting for. */
                 to_network_layer(&r.info);     /* pass the data to the network layer */
                 inc(frame_expected);           /* next time expect the other sequence nr */
            }
            s.ack = 1 – frame_expected;         /* tell which frame is being acked */
            to_physical_layer(&s);              /* send acknowledgement */
       }
  }
}
```

A positive acknowledgement with retransmission protocol.

Flow Control Effect on Network Performance

- Bandwidth Delay Product (BDP):
  - The product of a data link's capacity (in bits per second) and its round-trip delay time (in seconds).
  - The result, an amount of data measured in bits (or bytes), **is equivalent to the maximum amount of data on the network circuit at any given time, i.e., data that has been transmitted but not yet acknowledged**.

Example

Assume that, in a Stop-and-Wait system, the bandwidth of the line is 1 Mbps, and propagation time is 10 ms. What is the bandwidth-delay product? If the system data frames are 1000 bits in length

Solution

The bandwidth-delay product is

$(1 \times 10^{6}) \times (10 \times 10^{-3}) = 10{,}000$ bits

What is the time needed for an ACK to arrive? (ignore the ACK frame transmission time)?

=Frame Transmission time + 2*Propagation time = $1000/ 10^{6} + 2x\ 10 \times 10^{-3} = 0.021$ sec

How many frames can be transmitted during that time?

= (time * bandwidth) / (frame size in bits)

$0.021 \times (1 \times 10^{6}) = 21000$ bits / 1000 = 21 frames

What is the Link Utilization if stop-and-wait is used?

Link Utilization = (number of actually transmitted frame/ # frames that can be transmitted)*100

$(1/21) \times 100 = 5\ \%$

Go-Back-N Automatic Repeat Request

- In the Go-Back-N Protocol, the sequence numbers are modulo 2^m,
- m is the size of the sequence number field in bits.
- The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: Sf, Sn, and S_{size}.
- The send window can slide one or more slots when a valid acknowledgment arrives.

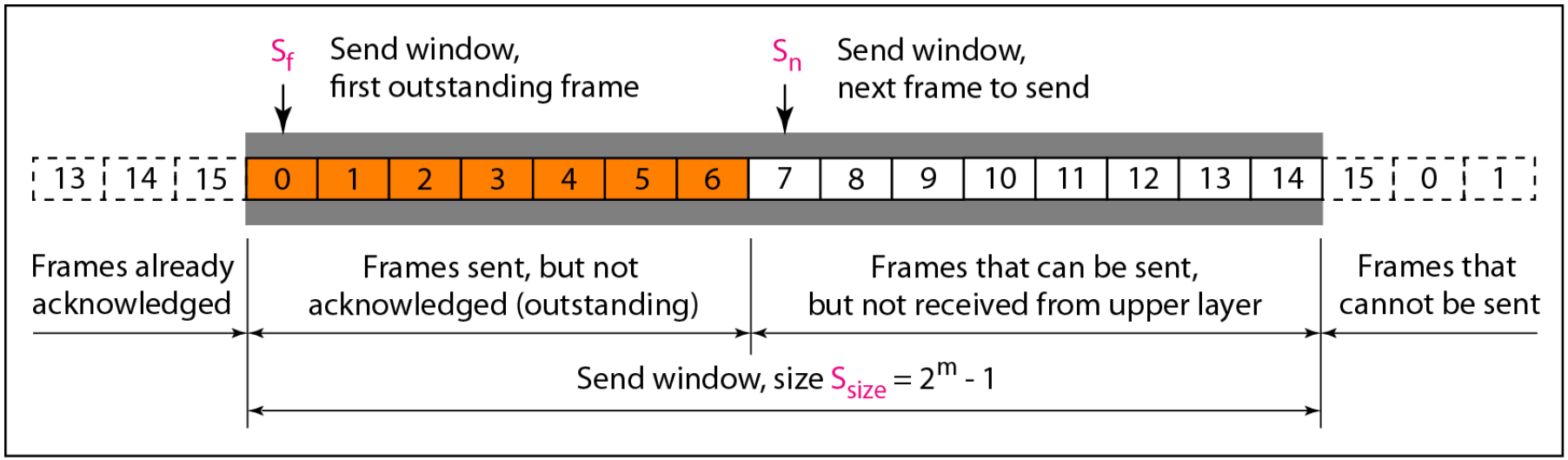
Sliding Window Flow Control

- Allow multiple frames to be in transit
- Transmitter can send up to W frames without ACK (sender window)
- Each frame is numbered (sequence number)
- ACK includes number of next frame expected
  - Ack for frame n = I am expecting frame n+1
    (not "I received fame n")
- Sequence number bounded by size of field
  - e.g. k bits: Frames are numbered modulo 2^k

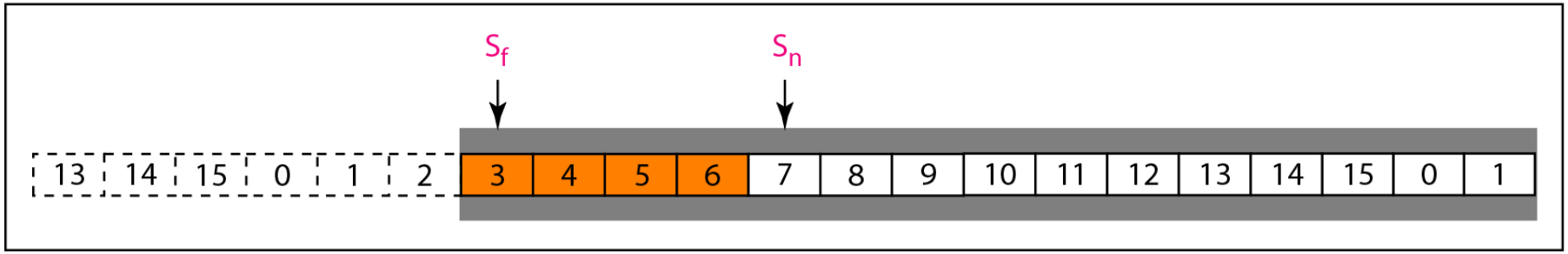
Operations

- Based on sliding window
- If no error, ACK as usual with next frame expected
- Use window to control number of outstanding frames
- If error, reply with rejection
  - Discard that frame and all future frames until error/lost frame received correctly
  - Transmitter must go back and retransmit that frame and all subsequent frames

Figure 11.12 *Send window for Go-Back-N ARQ*



a. Send window before sliding



b. Send window after sliding

receive window

- The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n.
- The window slides when a correct frame has arrived;
- sliding occurs one slot at a time.

Figure 11.13 *Receive window for Go-Back-N ARQ*

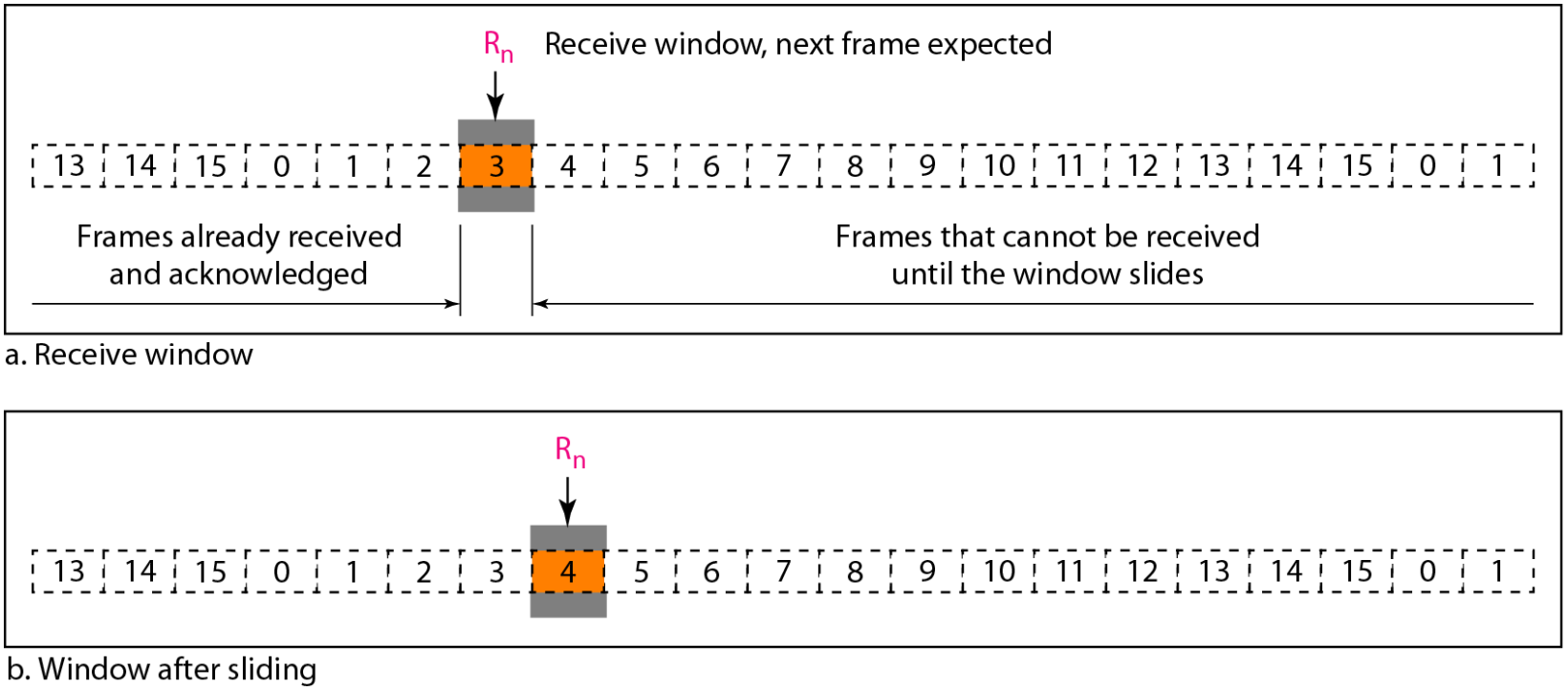


Figure 11.14 *Design of Go-Back-N ARQ*

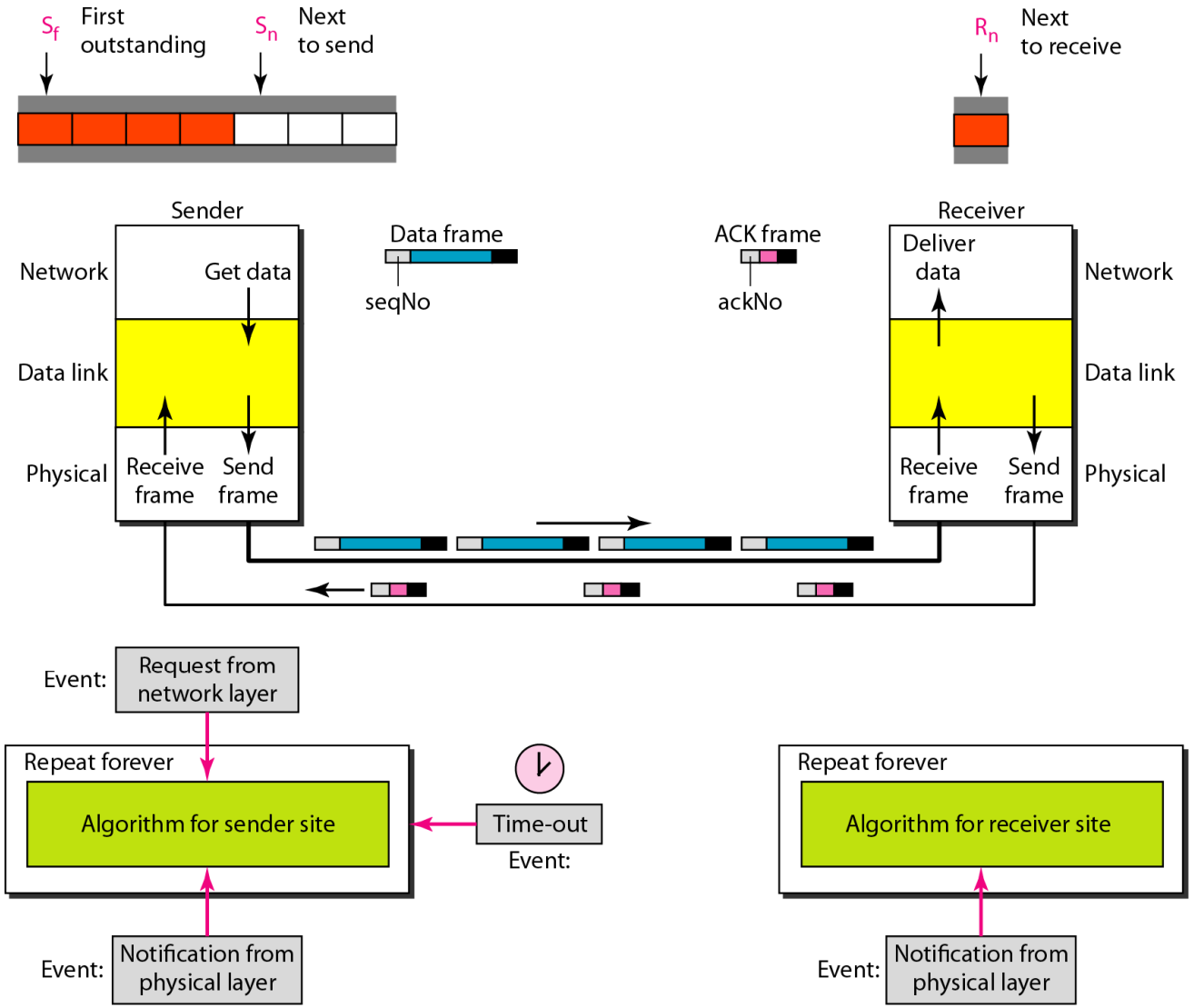
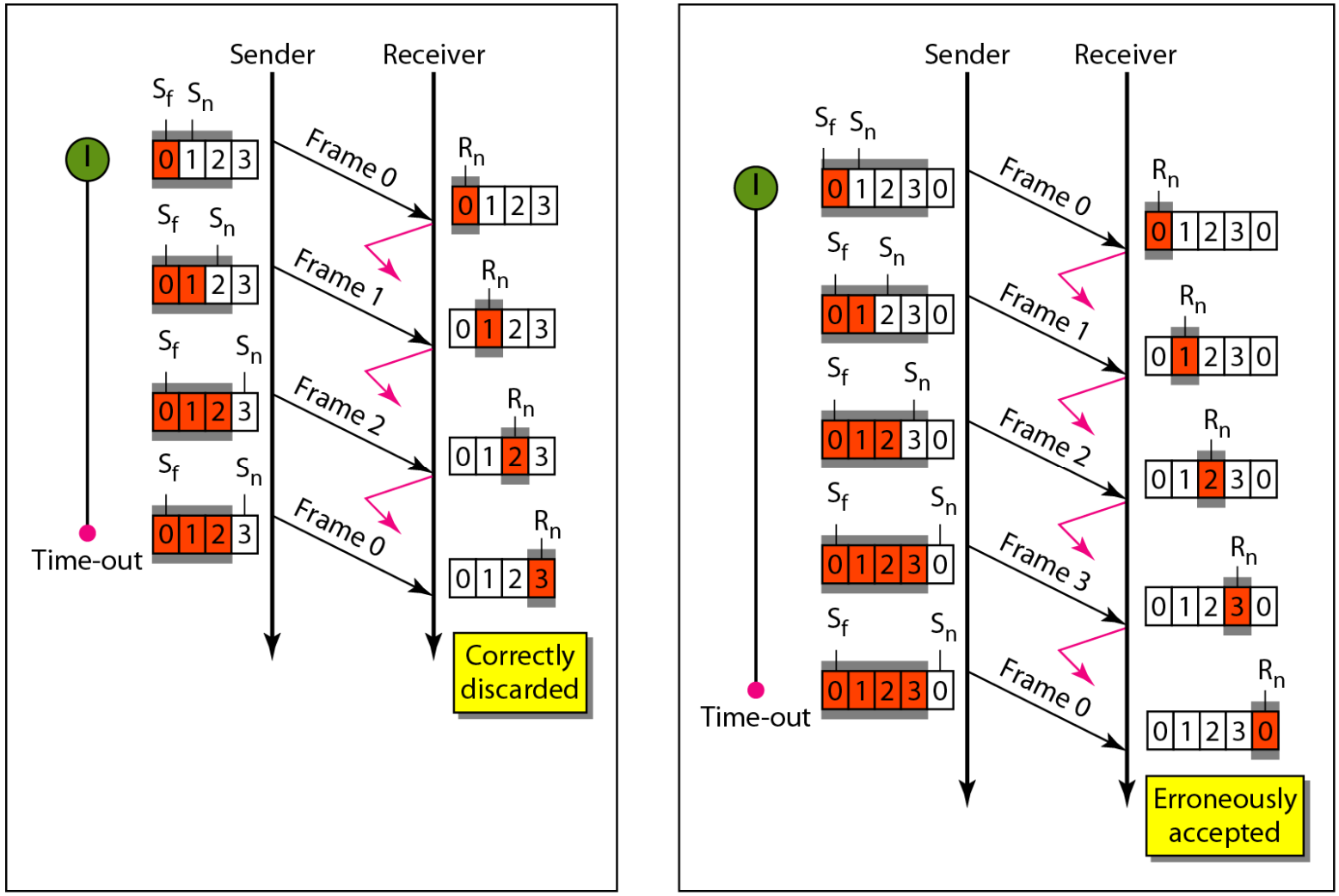


Figure 11.15 *Window size for Go-Back-N ARQ*



a. Window size < 2^m

b. Window size = 2^m

Note

In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.

Algorithm 11.7 *Go-Back-N sender algorithm*

```
1  Sw = 2^m - 1;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7   WaitForEvent();
8     if(Event(RequestToSend))           //A packet to send
9     {
10       if(Sn-Sf >= Sw)                //If window is full
11             Sleep();
12       GetData();
13       MakeFrame(Sn);
14       StoreFrame(Sn);
15       SendFrame(Sn);
16       Sn = Sn + 1;
17       if(timer not running)
18            StartTimer();
19    }
20
```

(continued)

Algorithm 11.7 *Go-Back-N sender algorithm* (continued)

```
21    if(Event(ArrivalNotification)) //ACK arrives
22    {
23       Receive(ACK);
24       if(corrupted(ACK))
25            Sleep();
26       if((ackNo>Sf)&&(ackNo<=Sn)) //If a valid ACK
27       While(Sf <= ackNo)
28        {
29         PurgeFrame(Sf);
30         Sf = Sf + 1;
31        }
32        StopTimer();
33    }
34
35    if(Event(TimeOut))                  //The timer expires
36    {
37     StartTimer();
38     Temp = Sf;
39     while(Temp < Sn);
40       {
41        SendFrame(Sf);
42        Sf = Sf + 1;
43       }
44    }
45 }
```

Algorithm 11.8 *Go-Back-N receiver algorithm*

```
1  Rn = 0;
2
3  while (true)                          //Repeat forever
4  {
5    WaitForEvent();
6
7    if(Event(ArrivalNotification)) /Data frame arrives
8    {
9       Receive(Frame);
10      if(corrupted(Frame))
11           Sleep();
12      if(seqNo == Rn)                  //If expected frame
13      {
14        DeliverData();                 //Deliver data
15        Rn = Rn + 1;                   //Slide window
16        SendACK(Rn);
17      }
18   }
19 }
```

Go-Back-N. Assume forward link is reliable,reverse not.

No data frames are lost, some ACKs delayed, 1 is lost.

See how cumulative ACKs can help if ACKs are delayed/lost.

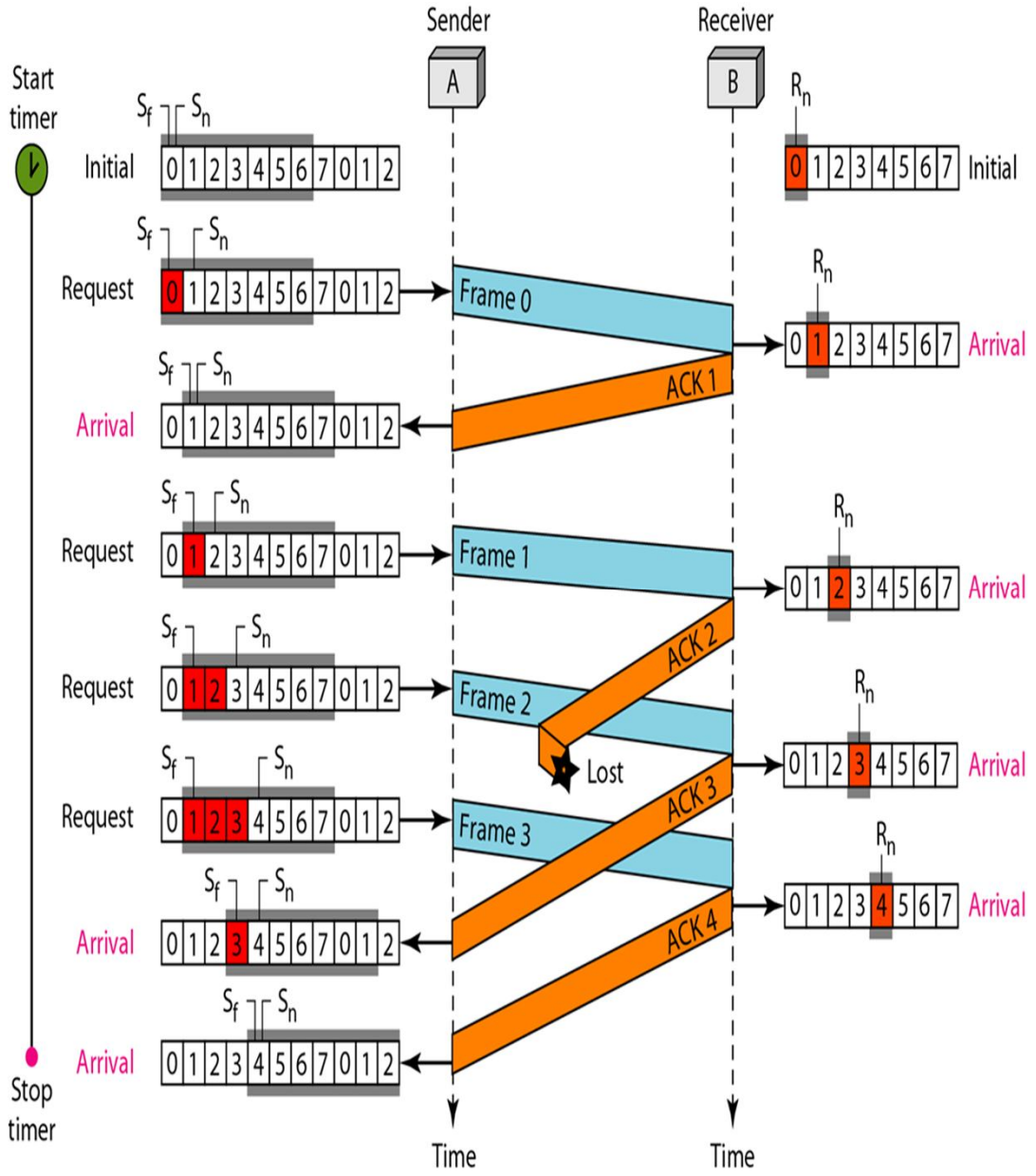
After initialization, there are 7 sender events.

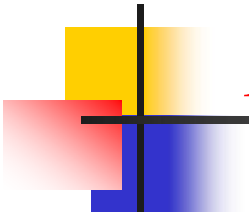
Request events: triggered by data from network layer;

Arrival events: triggered by ACKs from the physical layer.

There is no time-out event here b/c all outstanding frames are ACKed before timer expires.

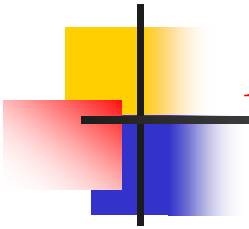
Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.





Example 11.7

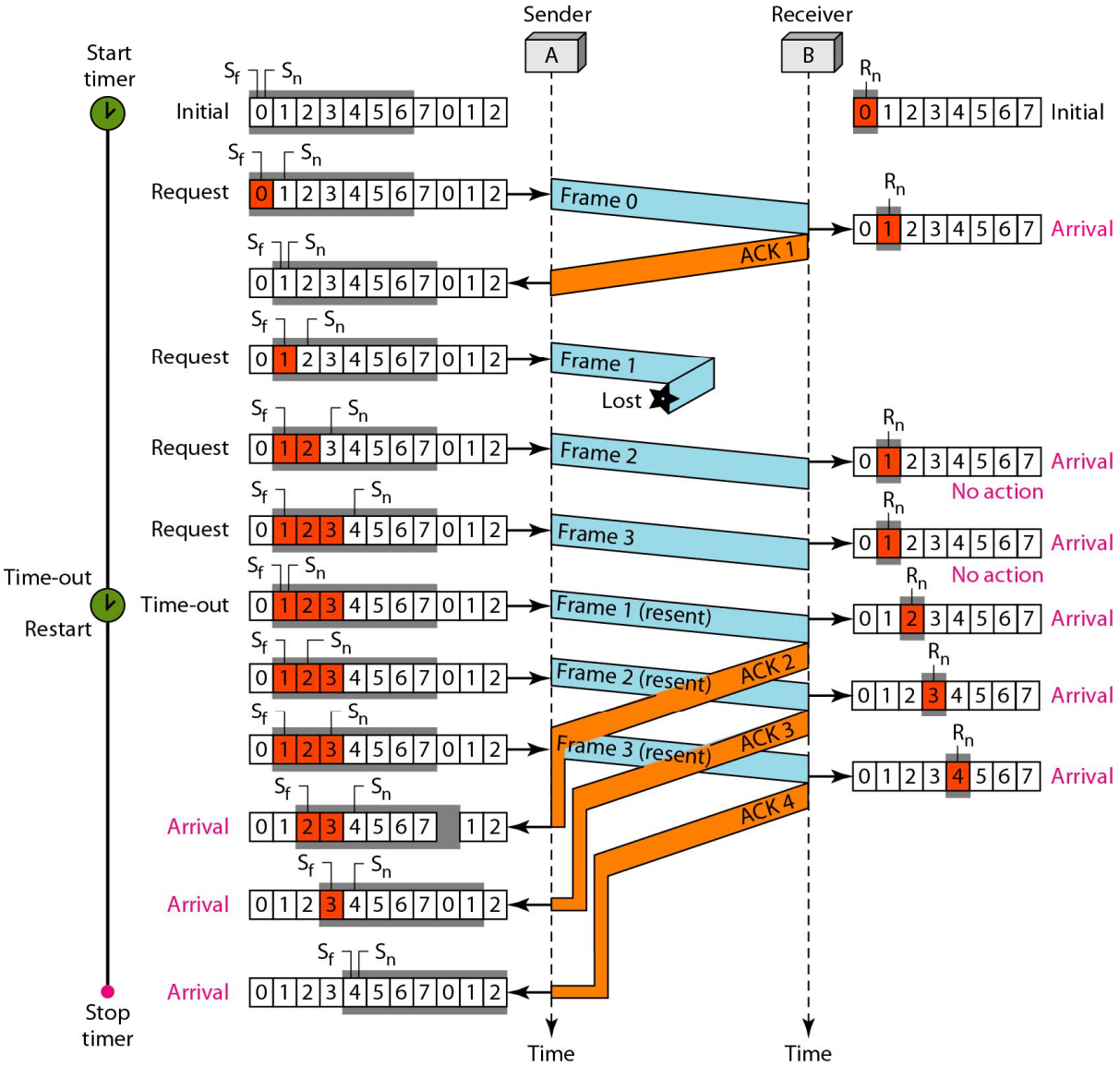
Figure 11.17 shows what happens when a frame is lost. Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out of order. The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong. Note that the resending of frames 1, 2, and 3 is the response to one single event. When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3.



Example 11.7 (continued)

The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state. We have shown a vertical line to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again when ACK 4 arrives. Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.

Figure 11.17 *Flow diagram for Example 11.7*



Note

Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

Selective Repeat Automatic Repeat Request

- In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m
- m is the size of the sequence number field in bits.

Features

- Allow multiple frames to be in transit
- Receiver has buffer W long (receiver window)
- Transmitter can send up to W frames without ACK (sender window)

Operations

- The sender retransmits only the frame with errors
- The receiver stores all the correct frames that arrive following the bad one (receiver window > 1)
- Requires a significant amount of buffer space at the receiver
- When the sender notices that something is wrong, it just retransmits the one bad frame, not all its successors
- Might be combined with Negative Acknowledgment (NACK)

Figure 11.18 *Send window for Selective Repeat ARQ*

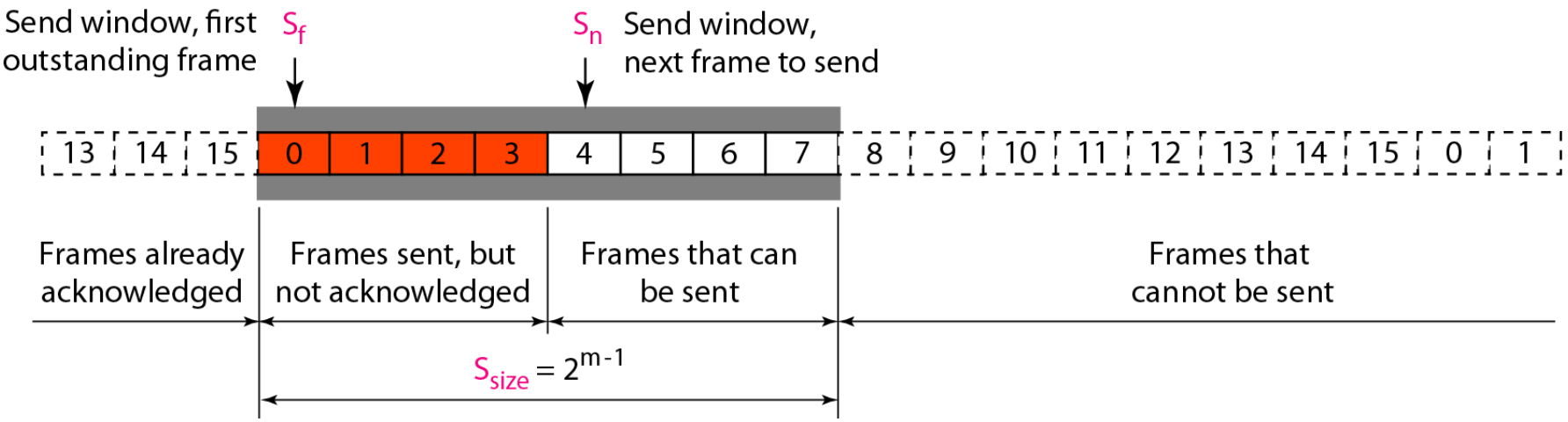


Figure 11.19 *Receive window for Selective Repeat ARQ*

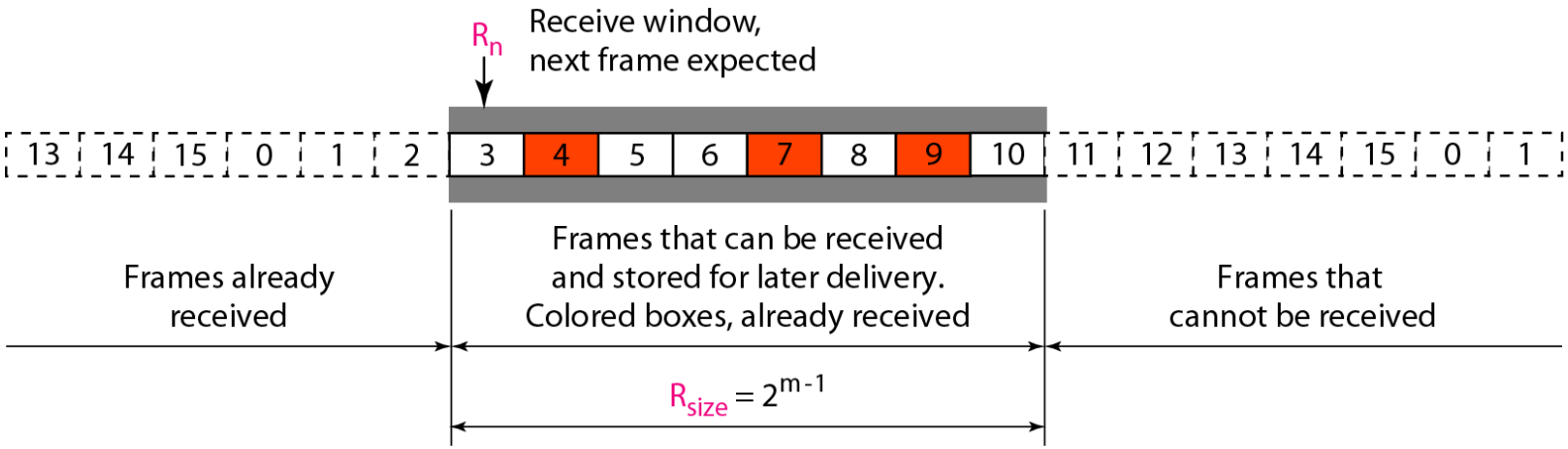


Figure 11.20 *Design of Selective Repeat ARQ*

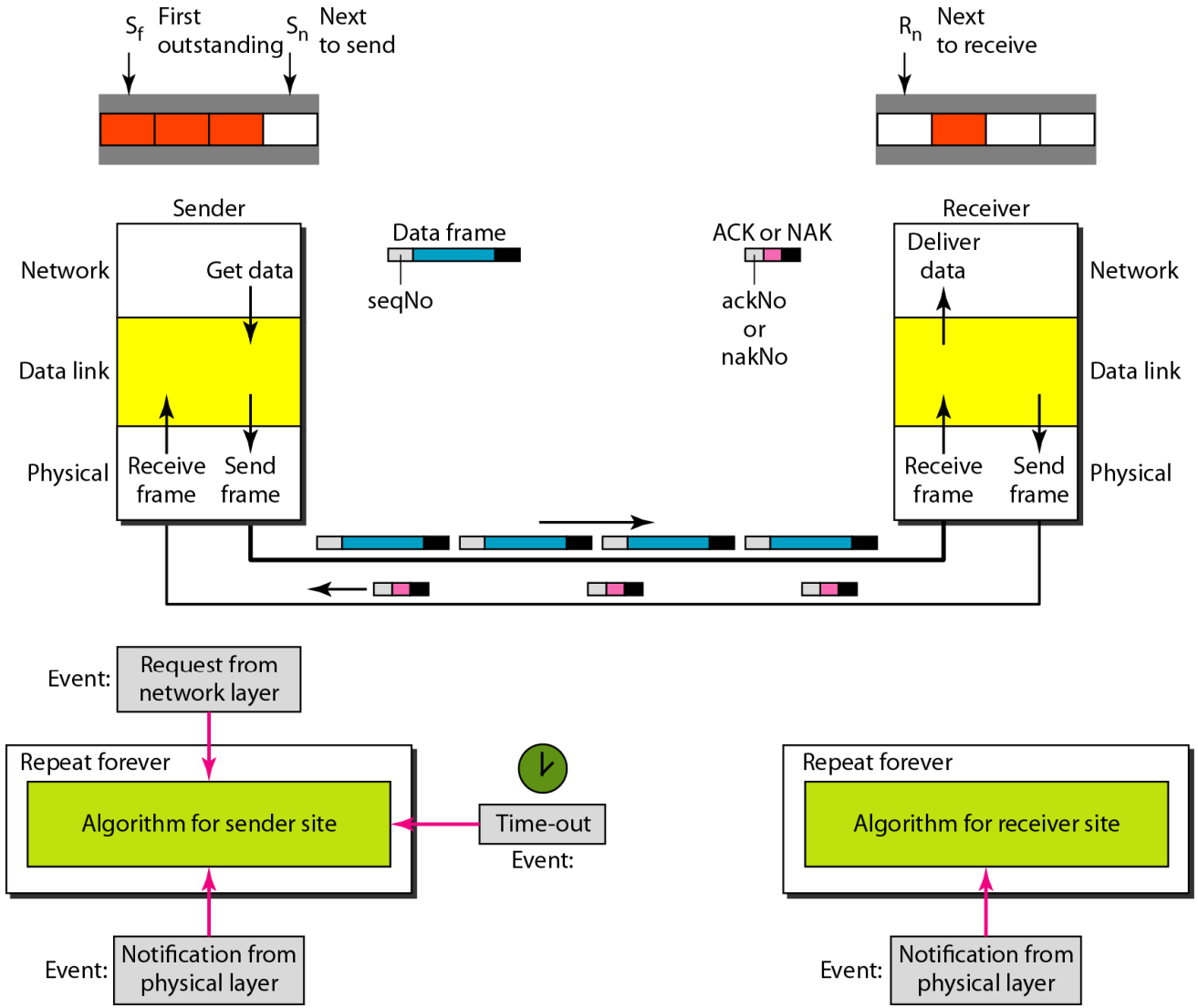
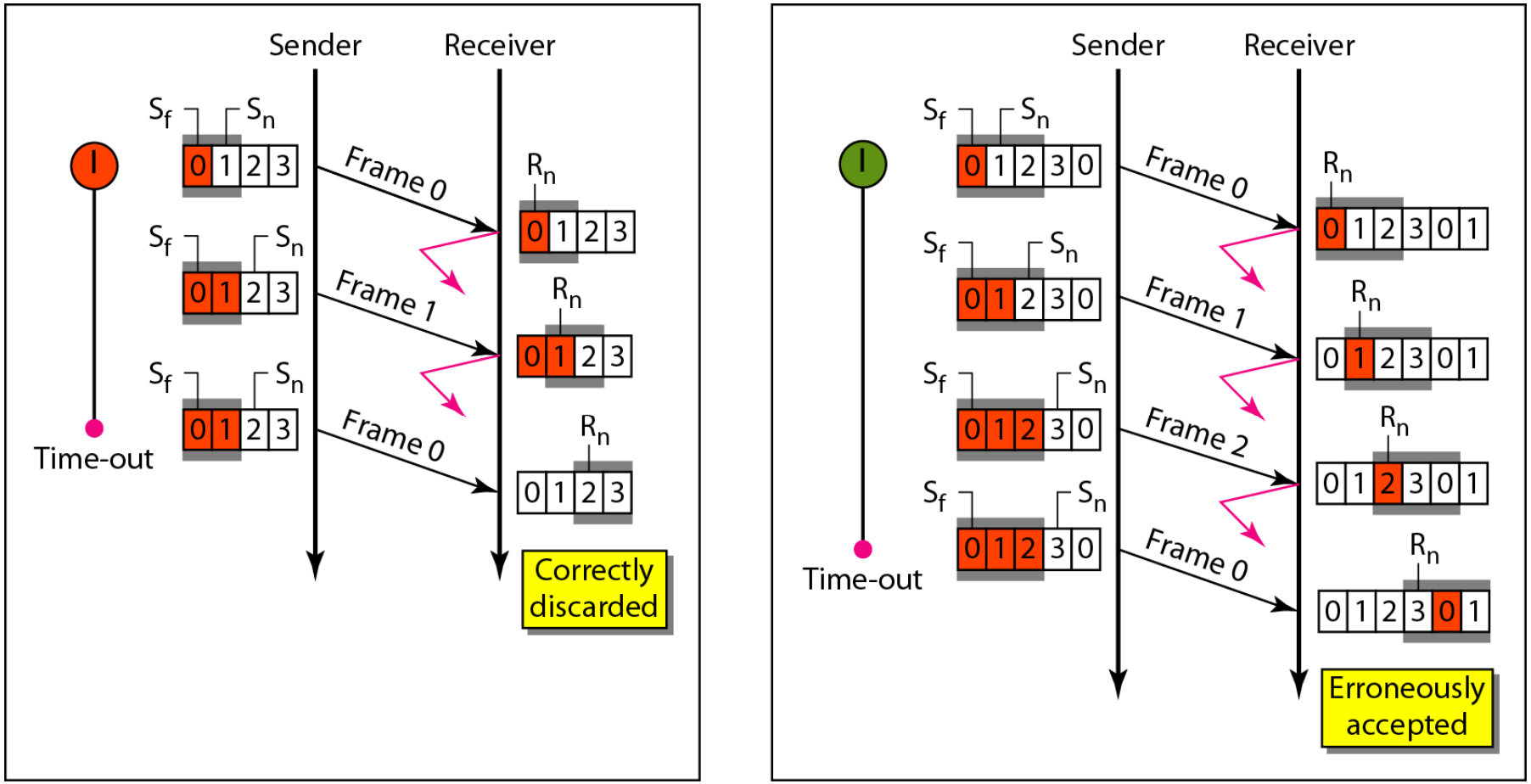


Figure 11.21 *Selective Repeat ARQ, window size*



a. Window size = 2^{m-1}

b. Window size > 2^{m-1}

Algorithm 11.9 *Sender-site Selective Repeat algorithm*

```
1  Sw = 2^(m-1) ;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7    WaitForEvent();
8    if(Event(RequestToSend))            //There is a packet to send
9    {
10     if(Sn-Sf >= Sw)                   //If window is full
11           Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     StartTimer(Sn);
18   }
19
```

(continued)

Algorithm 11.9 *Sender-site Selective Repeat algorithm* (continued)

```
20  if(Event(ArrivalNotification)) //ACK arrives
21  {
22     Receive(frame);              //Receive ACK or NAK
23     if(corrupted(frame))
24          Sleep();
25     if (FrameType == NAK)
26        if (nakNo between Sf and Sn)
27        {
28         resend(nakNo);
29         StartTimer(nakNo);
30        }
31     if (FrameType == ACK)
32        if (ackNo between Sf and Sn)
33        {
34          while(sf < ackNo)
35          {
36           Purge(sf);
37           StopTimer(sf);
38           Sf = Sf + 1;
39          }
40        }
41  }
```

(continued)

Algorithm 11.9 *Sender-site Selective Repeat algorithm* (continued)

```
42
43    if(Event(TimeOut(t)))              //The timer expires
44    {
45     StartTimer(t);
46     SendFrame(t);
47    }
48  }
```

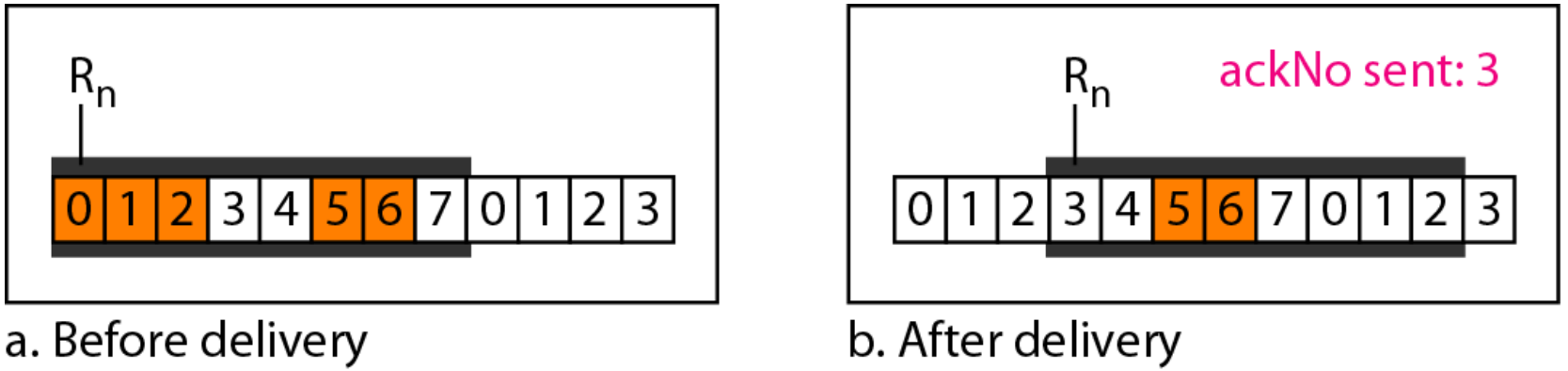
Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

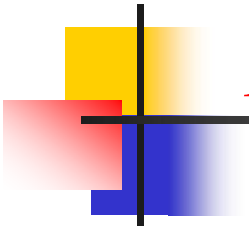
```
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5      Marked(slot) = false;
6
7  while (true)                                  //Repeat forever
8  {
9    WaitForEvent();
10
11   if(Event(ArrivalNotification))              /Data frame arrives
12   {
13      Receive(Frame);
14      if(corrupted(Frame))&& (NOT NakSent)
15      {
16       SendNAK(Rn);
17       NakSent = true;
18       Sleep();
19      }
20      if(seqNo <> Rn)&& (NOT NakSent)
21      {
22       SendNAK(Rn);
```

Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
23          NakSent = true;
24          if ((seqNo in window)&&(!Marked(seqNo))
25          {
26           StoreFrame(seqNo)
27           Marked(seqNo)= true;
28           while(Marked(Rn))
29           {
30            DeliverData(Rn);
31            Purge(Rn);
32            Rn = Rn + 1;
33            AckNeeded = true;
34           }
35            if(AckNeeded);
36            {
37            SendAck(Rn);
38            AckNeeded = false;
39            NakSent = false;
40            }
41         }
42        }
43    }
44  }
```

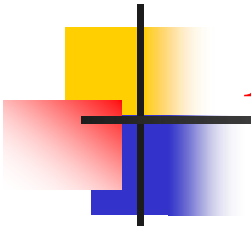
Figure 11.22 *Delivery of data in Selective Repeat ARQ*





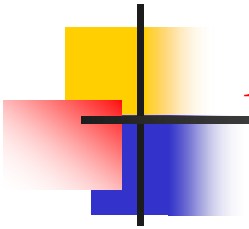
Example 11.8

This example is similar to Example 11.3 in which frame 1 is lost. We show how Selective Repeat behaves in this case. Figure 11.23 shows the situation. One main difference is the number of timers. Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1, 2, and 3). The timer for frame 0 starts at the first request, but stops when the ACK for this frame arrives. The timer for frame 1 starts at the second request, restarts when a NAK arrives, and finally stops when the last ACK arrives. The other two timers start when the corresponding frames are sent and stop at the last arrival event.



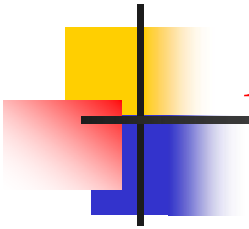
Example 11.8 (continued)

At the receiver site we need to distinguish between the acceptance of a frame and its delivery to the network layer. At the second arrival, frame 2 arrives and is stored and marked, but it cannot be delivered because frame 1 is missing. At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered. Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer. There are two conditions for the delivery of frames to the network layer: First, a set of consecutive frames must have arrived. Second, the set starts from the beginning of the window.



Example 11.8 (continued)

Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same. The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames. The second NAK would still be NAK1 to inform the sender to resend frame 1 again; this has already been done. The first NAK sent is remembered (using the nakSent variable) and is not sent again until the frame slides. A NAK is sent once for each window position and defines the first slot in the window.



Example 11.8 (continued)

The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames. In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to n frames are delivered in one shot, only one ACK is sent for all of them.

Figure 11.23 *Flow diagram for Example 11.8*

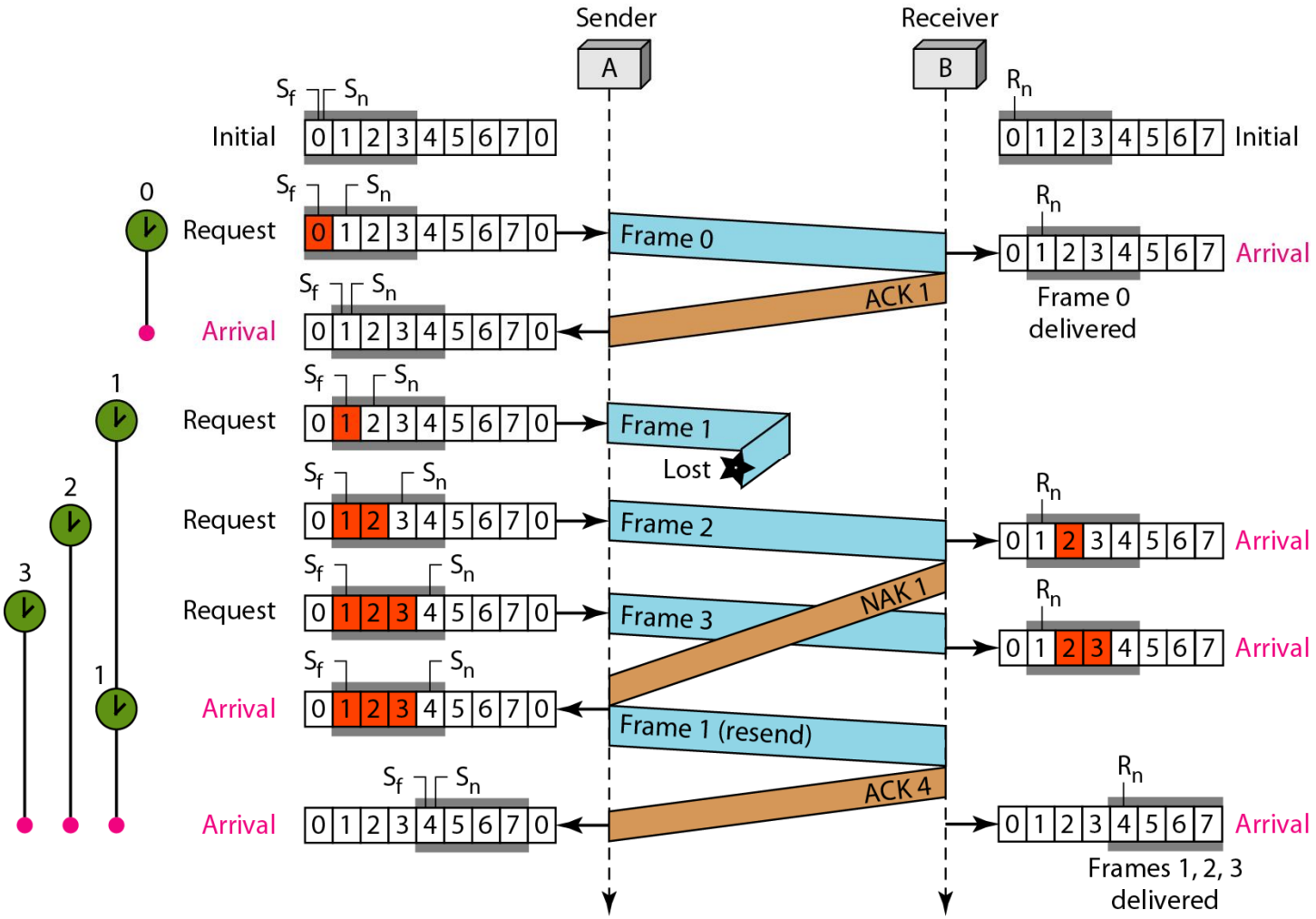
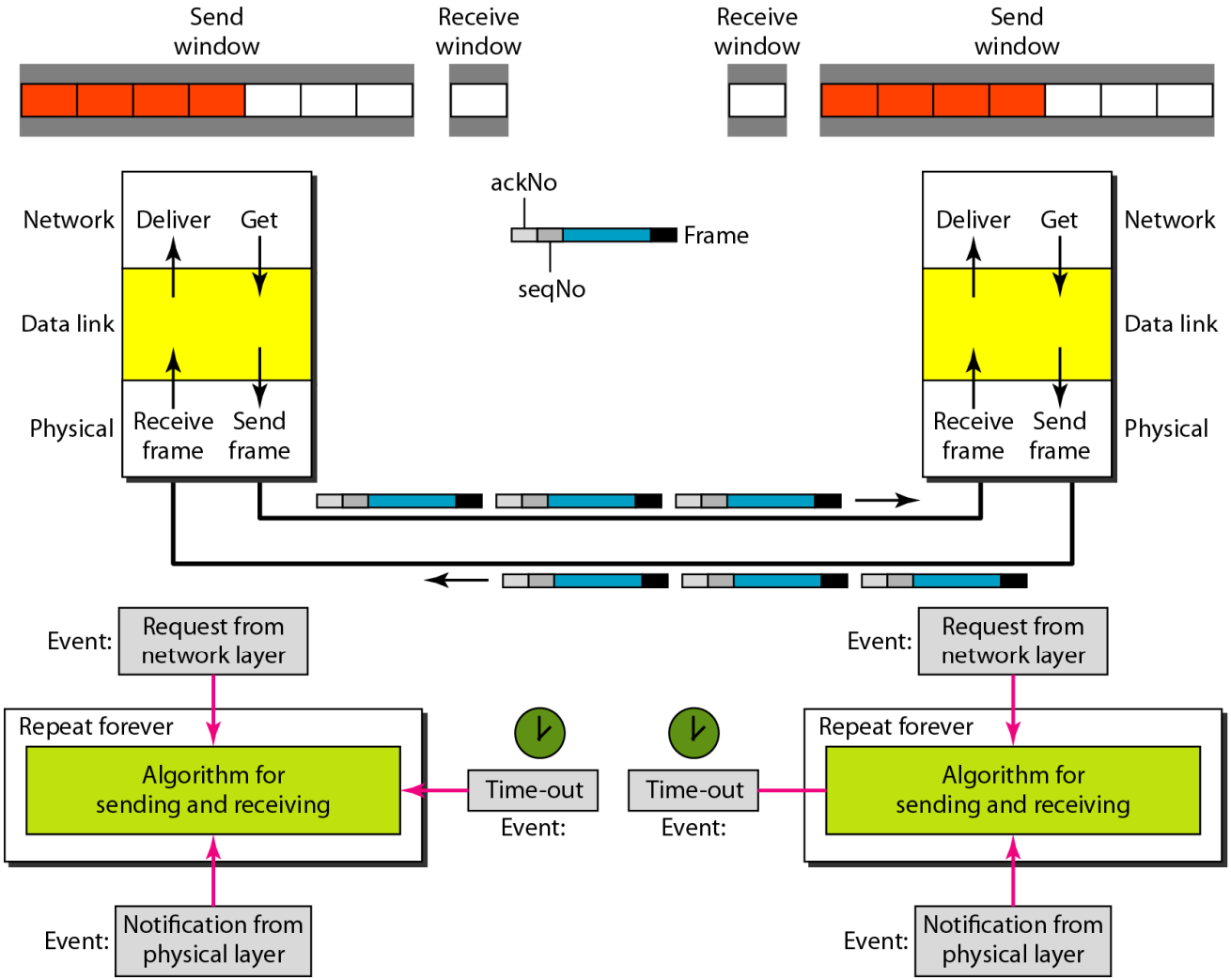


Figure 11.24 *Design of piggybacking in Go-Back-N ARQ*



HDLC

High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the ARQ mechanisms