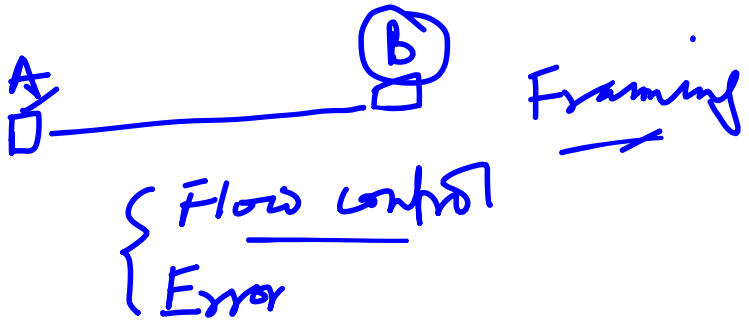
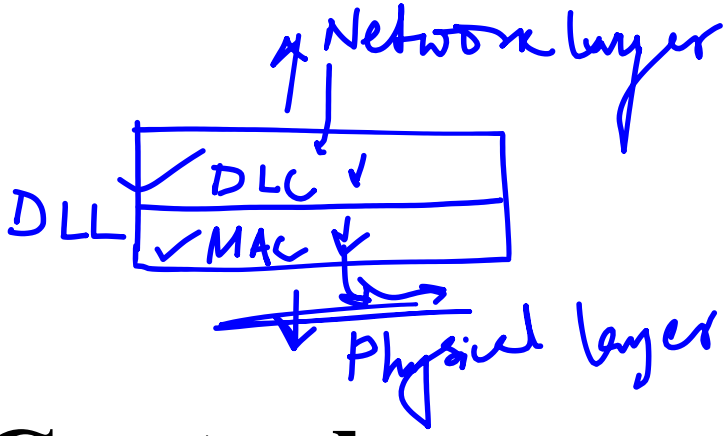
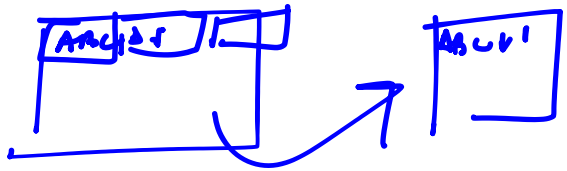


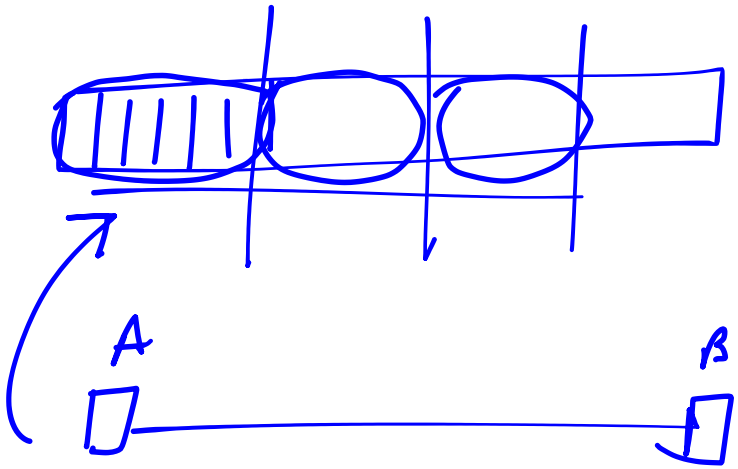


Data Link Control (DLC)

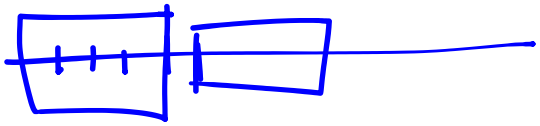
Data Link Control



- Deal with the procedure for node-to-node communication
- Functions include
  - Framing
  - Flow control
  - Error control



Framing



- The data link layer needs to pack bits into frames, so that each frame is distinguishable from another.
- Framing separates a message from source to destination by adding a sender address and a destination address
  - Example:

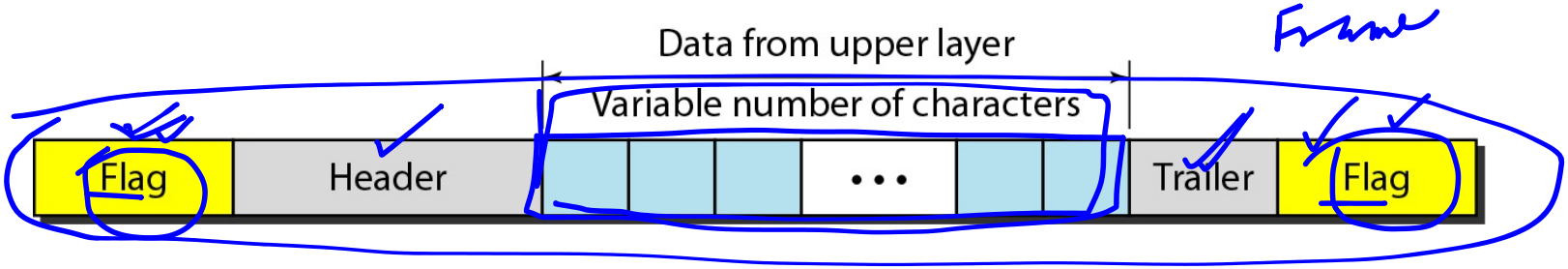
    - Our postal system practices a type of framing.
    - The simple act of inserting a letter into an envelope separates one piece of information from another.
    - The envelope serves as the delimiter.

Frame size

- Frames can be of
  - Fixed size:
    - No need of defining the boundaries of the frame
    - Size itself can be used as a delimiter
  - Variable size framing
    - Need a way to define the end of one frame and start of the next
    - Two approaches are followed for this
      - Character oriented
      - Bit-oriented

Character oriented Framing

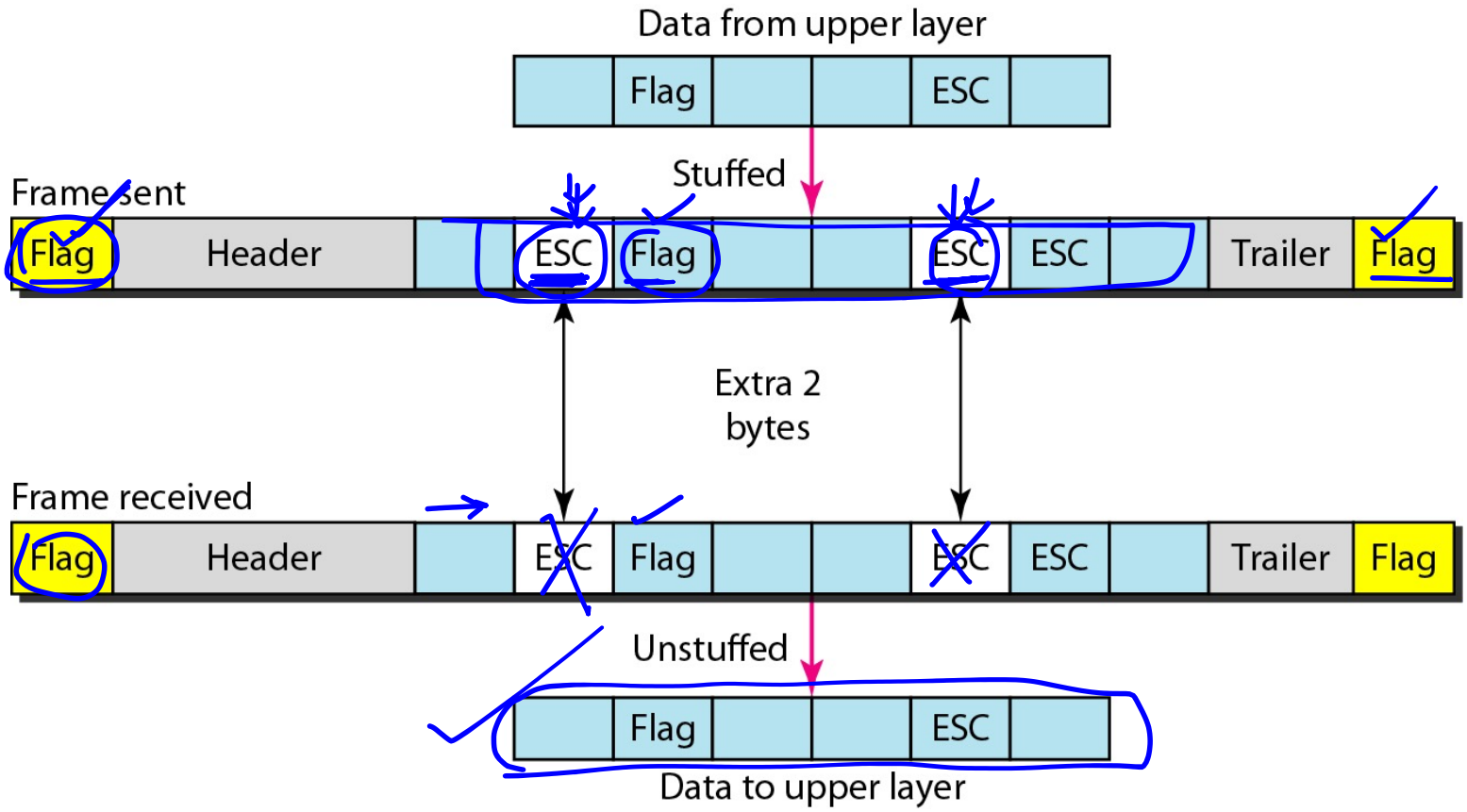
- **Data:** Characters are used; 8-bit ASCII
- **Header:**
  - Carries the source and destination address
  - Other control information
- **Trailor:**
  - Carries error detection redundant bits
- **Flag:**
  - Added at the beginning and at the end of the frame
  - Composed of protocol dependent special characters, signals the start and end of the frame



Character oriented Framing

- Popular when text was exchanges
- Flag could be any character not used for text communication
- But information can be graphs, audio, video
- Any character used as flag can be a part of information
- To fix this problem, a *byte stuffing(character stuffing)* strategy is followed

Byte stuffing and unstuffing

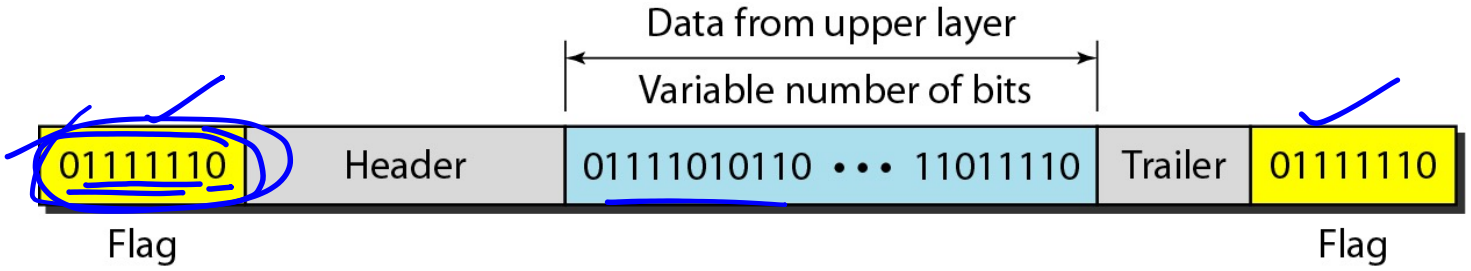


Note

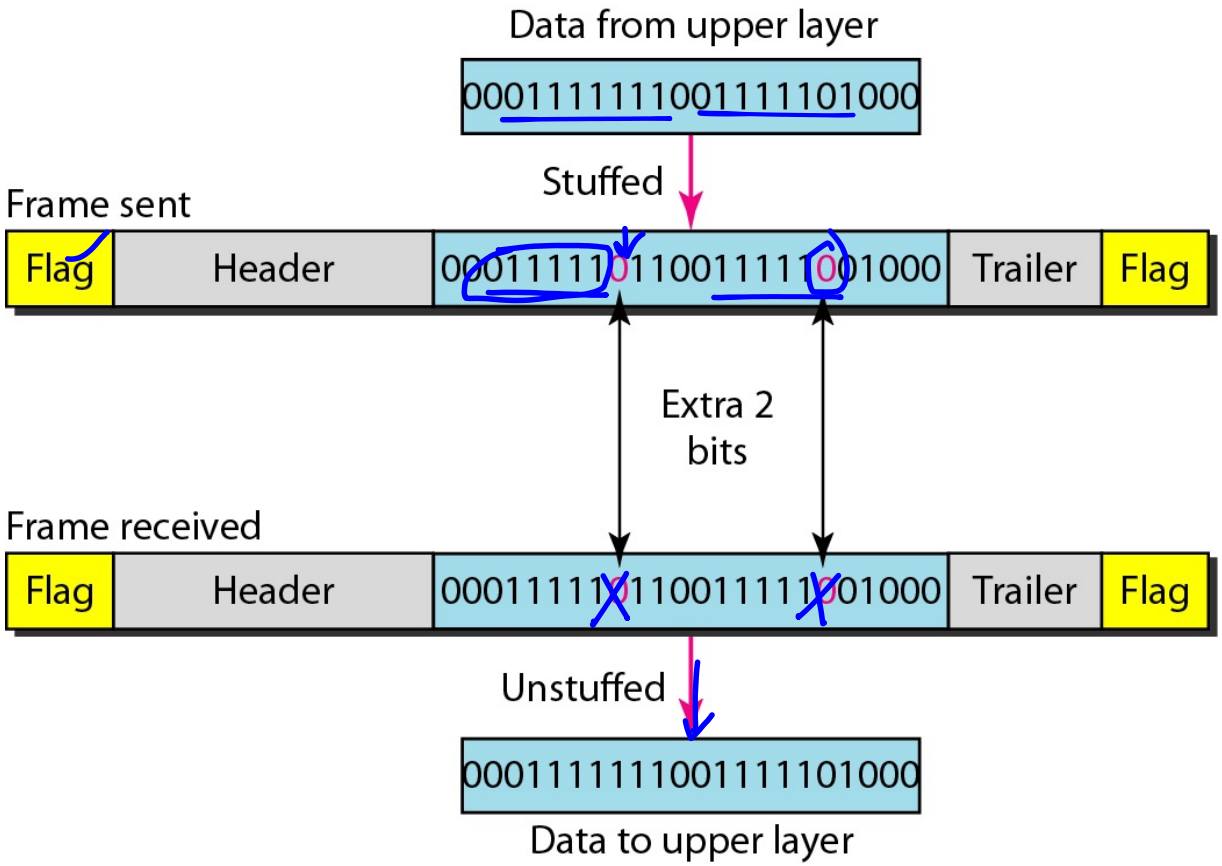
Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.

A frame in a bit-oriented protocol

- Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.



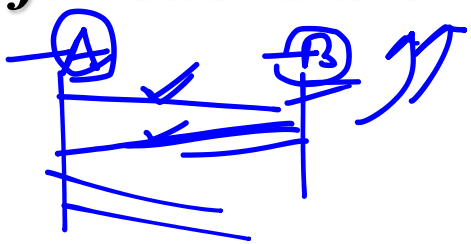
Bit stuffing and unstuffing



Flow and Error Control

- ***The most important responsibilities of the data link layer are flow control and error control.***

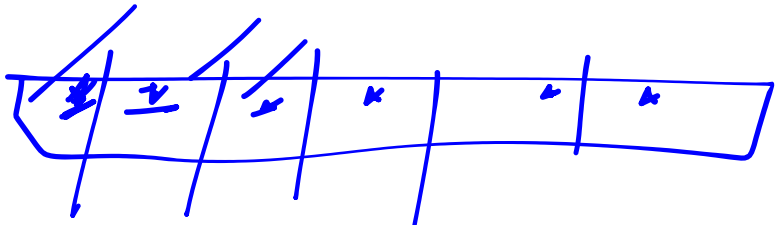
- ***Collectively, these functions are known as data link control***



- ***Flow Control:***
  - **Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment (ACK).**

- ***Error Control:***
  - Prevent the receiving node from delivering corrupted packets to its network layer

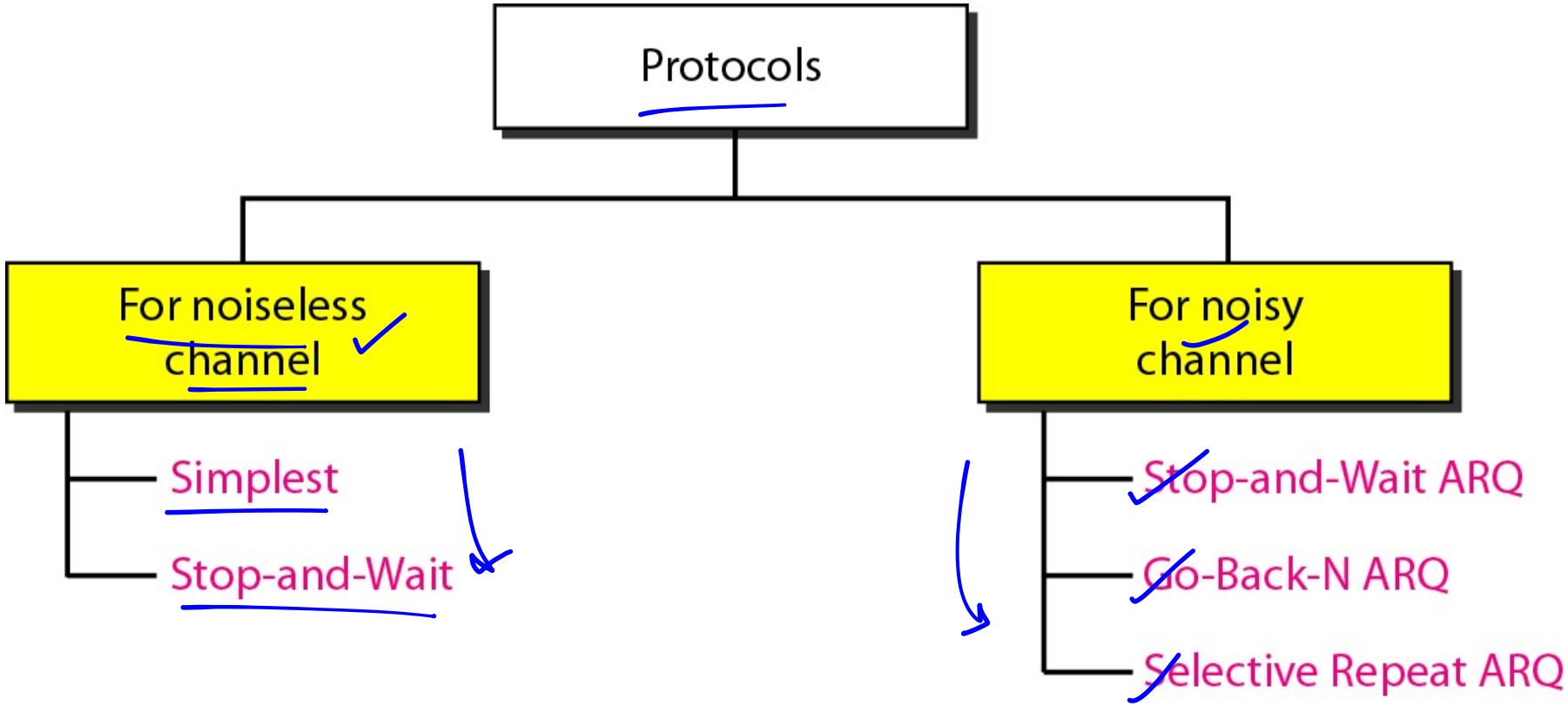
Protocols



- A DLC protocol can be either *connectionless or connection oriented*
- **Connectionless Protocol:**
  - Frames are sent from one node to the next without any relationship in between the frames
    - Each frame is independent
  - Here the frames are not numbered and hence no sense of ordering
- **Connection oriented Protocol:**

  - A logical connection should first be established between the two nodes (setup phase)
  - After all frames that are somehow related to each other are transmitted (transfer phase)
  - The logical connection is terminated (teardown phase)

Taxonomy of protocols

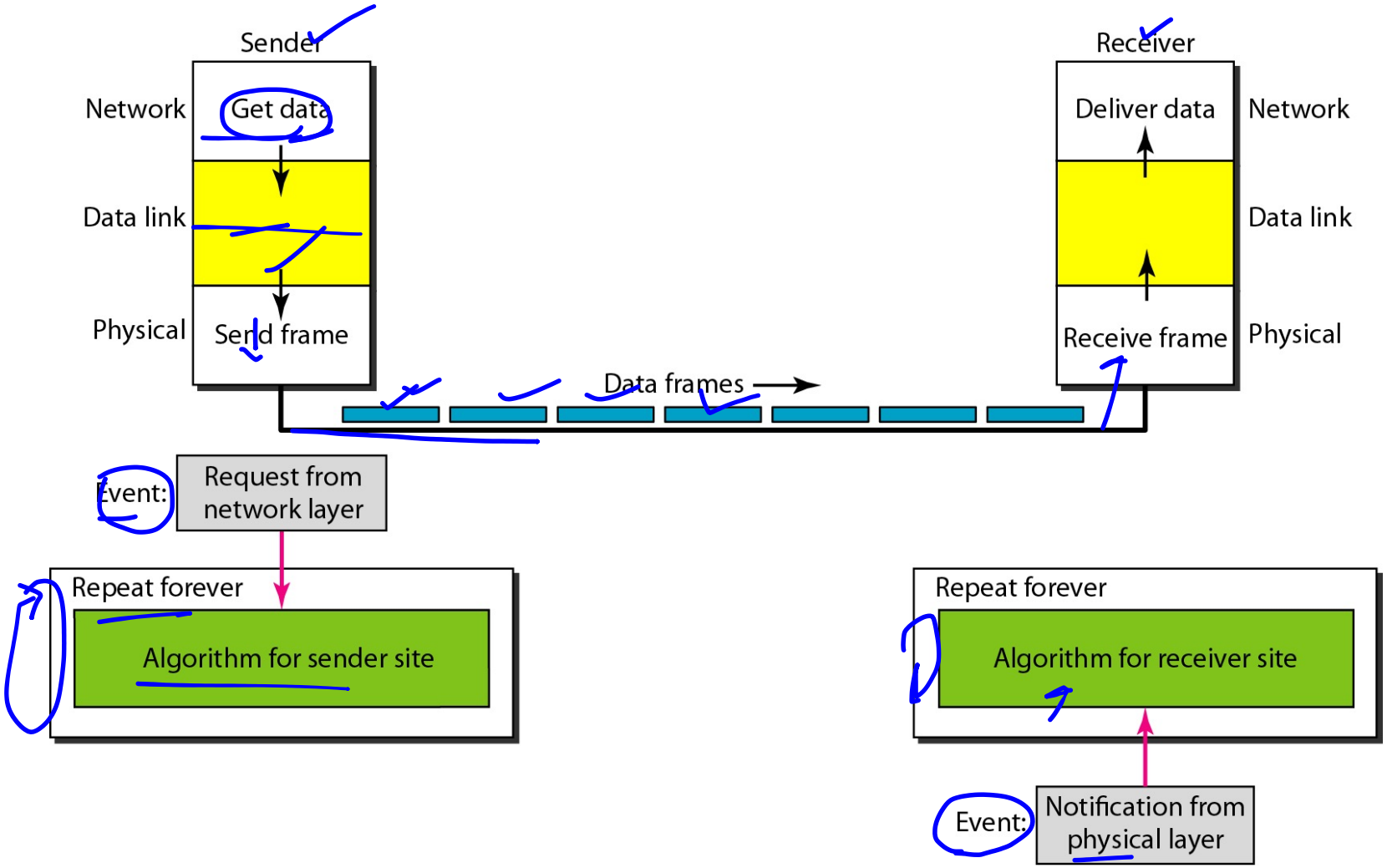


Noiseless Channels

- *It assumes we have an ideal channel in which no frames are lost, duplicated, or corrupted.*

- *We introduce two protocols for this type of channel.*
  - **Simplest Protocol**
  - **Stop-and-Wait Protocol**

Simplest protocol *with* *no flow or error control*



Algorithm : *Sender-site algorithm for the simplest protocol*

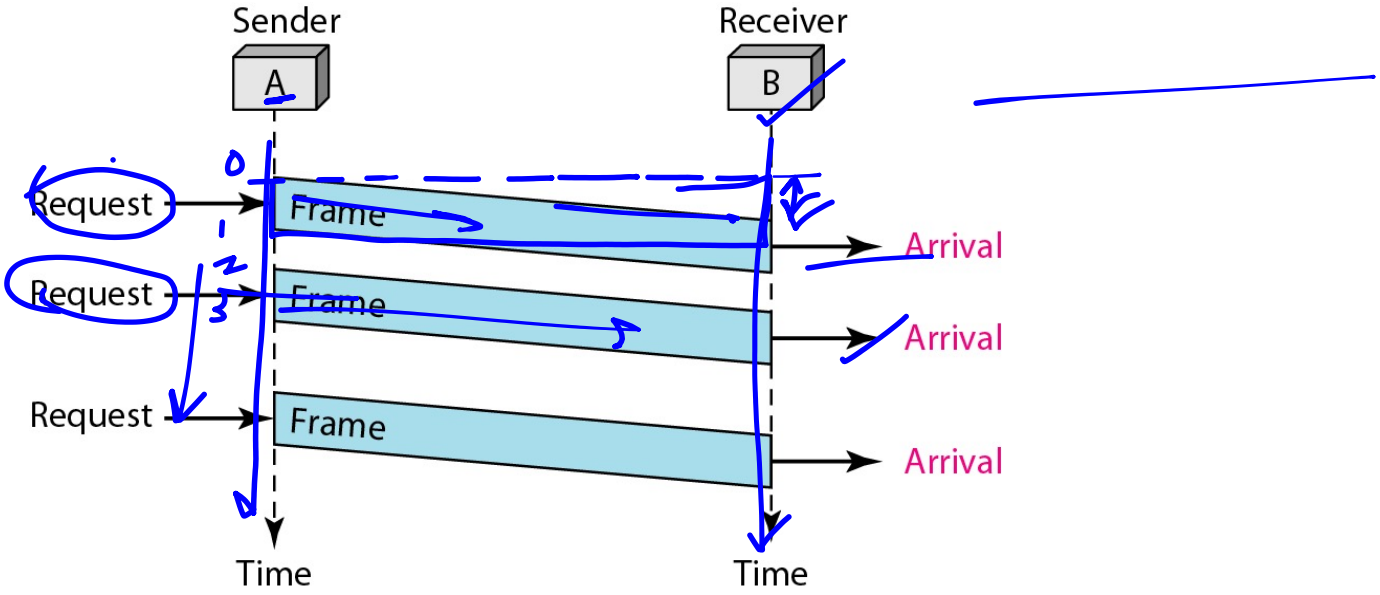
```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(RequestToSend))           //There is a packet to send
5    {
6        GetData();
7        MakeFrame();
8        SendFrame();                   //Send the frame
9    }
10 }
```

Algorithm : *Receiver-site algorithm for the simplest*

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrived
5    {
6        ReceiveFrame();
7        ExtractData();
8        DeliverData();                 //Deliver data to network layer
9    }
10 }
```

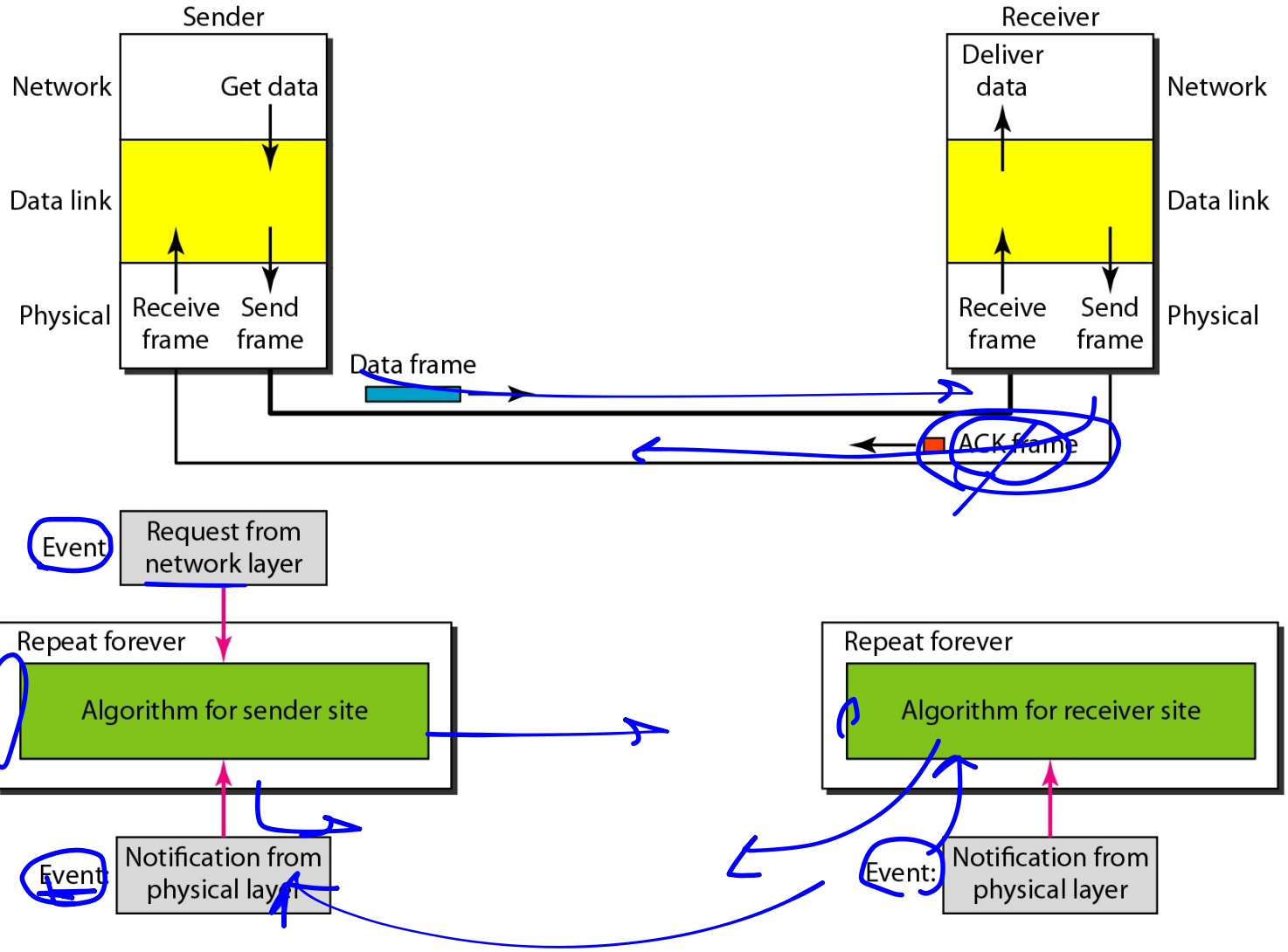
Example

- It is very simple.
- The sender sends a sequence of frames without even thinking about the receiver.
- To send three frames, three events occur at the sender site and three events at the receiver site.
- Note that the data frames are shown by tilted boxes; the height of the box defines the transmission time difference between the first bit and the last bit in the frame.



Flow diagram for Example

Design of Stop-and-Wait Protocol



Algorithm: *Sender-site algorithm for Stop-and-Wait Protocol*

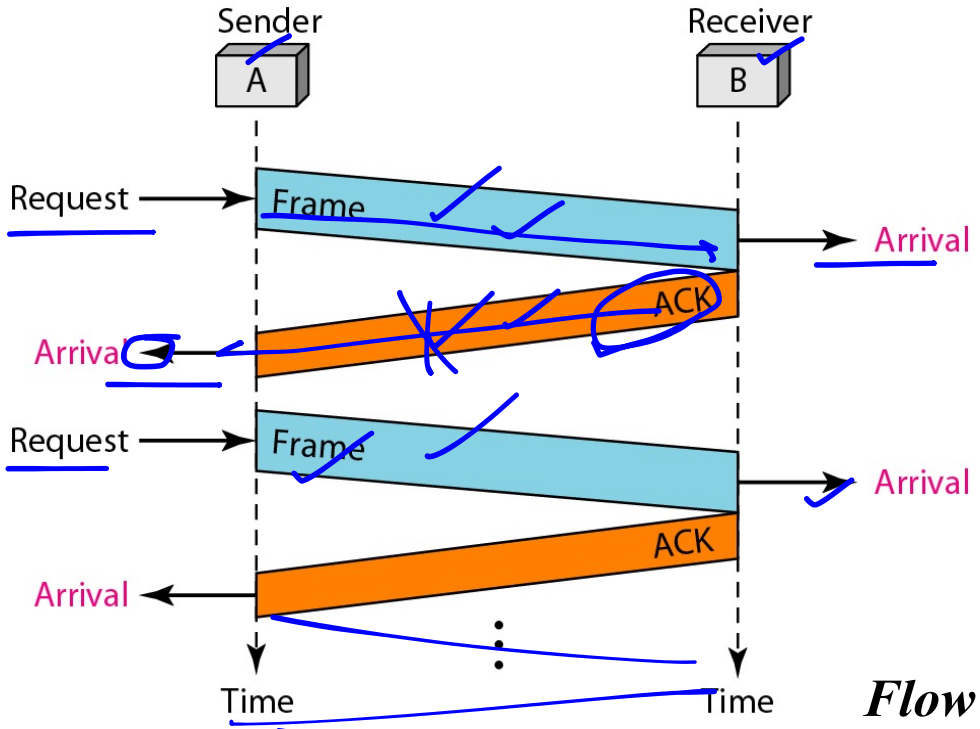
```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

Algorithm 11.4 *Receiver-site algorithm for Stop-and-Wait Protocol*

```
1  while(true)                          //Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrives
5    {
6       ReceiveFrame();
7       ExtractData();
8       Deliver(data);                  //Deliver data to network layer
9       SendFrame();                    //Send an ACK frame
10   }
11 }
```

Example

- It is still very simple.
- The sender sends one frame and waits for feedback from the receiver.
- When the ACK arrives, the sender sends the next frame.
- Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.

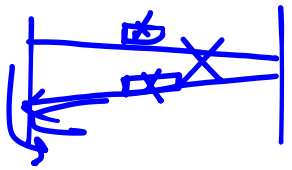


Flow diagram

Noisy Channels

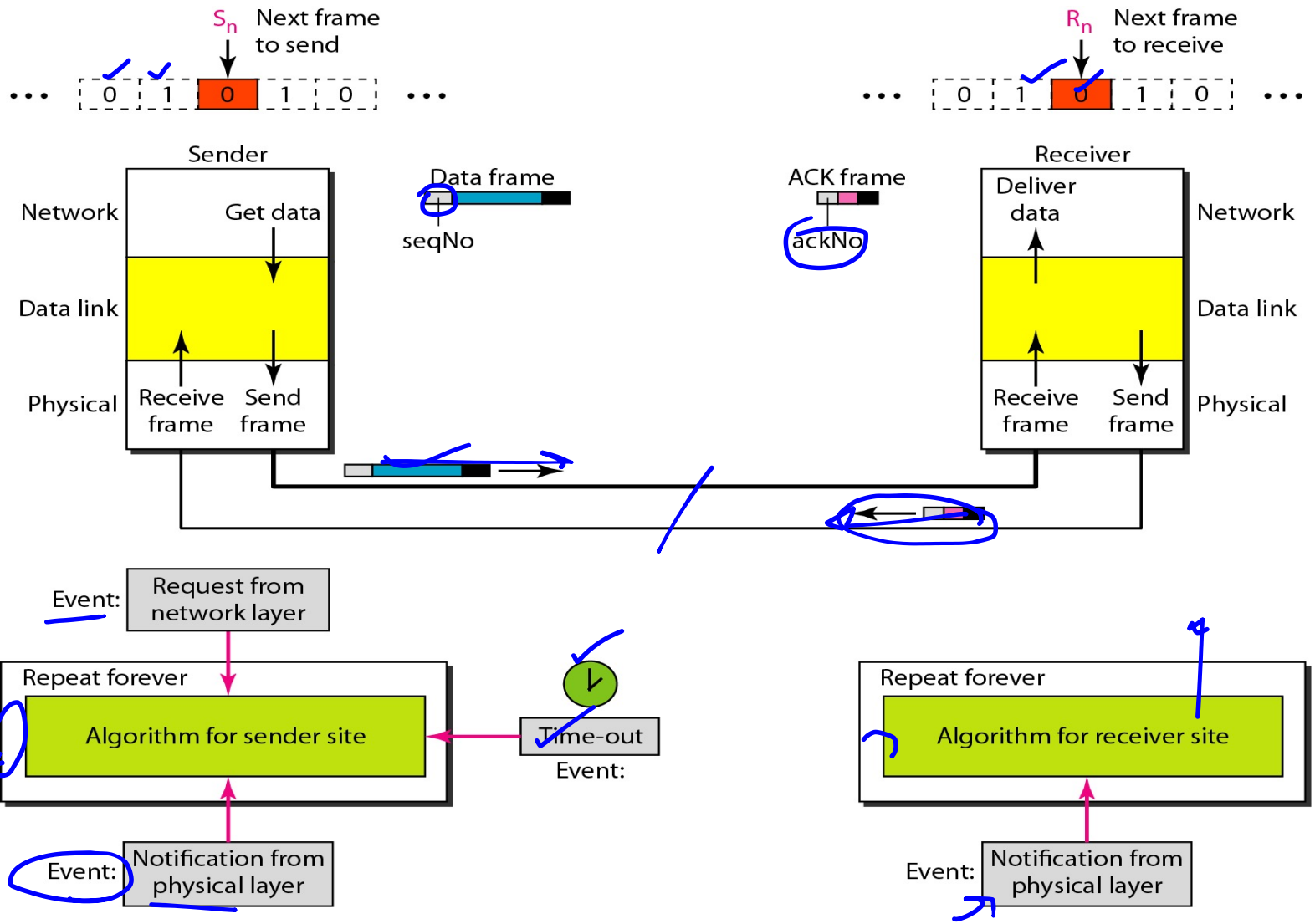
- *Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent.*
- *We discuss three protocols that use error control.*
  - **Stop-and-Wait Automatic Repeat Request**
  - **Go-Back-N Automatic Repeat Request**
  - **Selective Repeat Automatic Repeat Request**

Stop-and-Wait ARQ



- Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.
- we use sequence numbers to number the frames.
  - The sequence numbers are based on modulo-2 arithmetic.
- The acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Design of the Stop-and-Wait ARQ Protocol



Algorithm : *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)

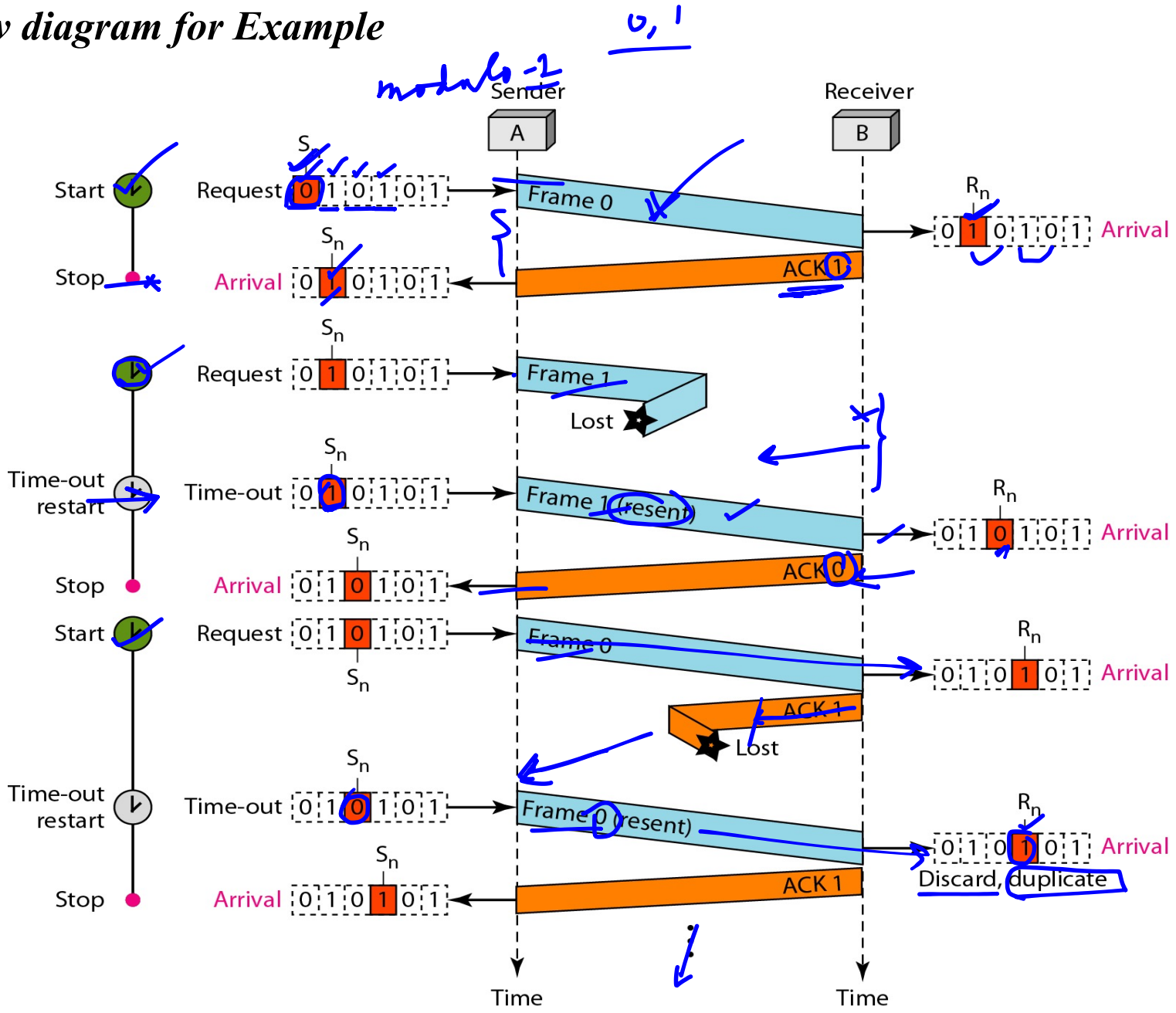
Algorithm : *Sender-site algorithm for Stop-and-Wait ARQ* (continued)

```
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 : *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 }
```

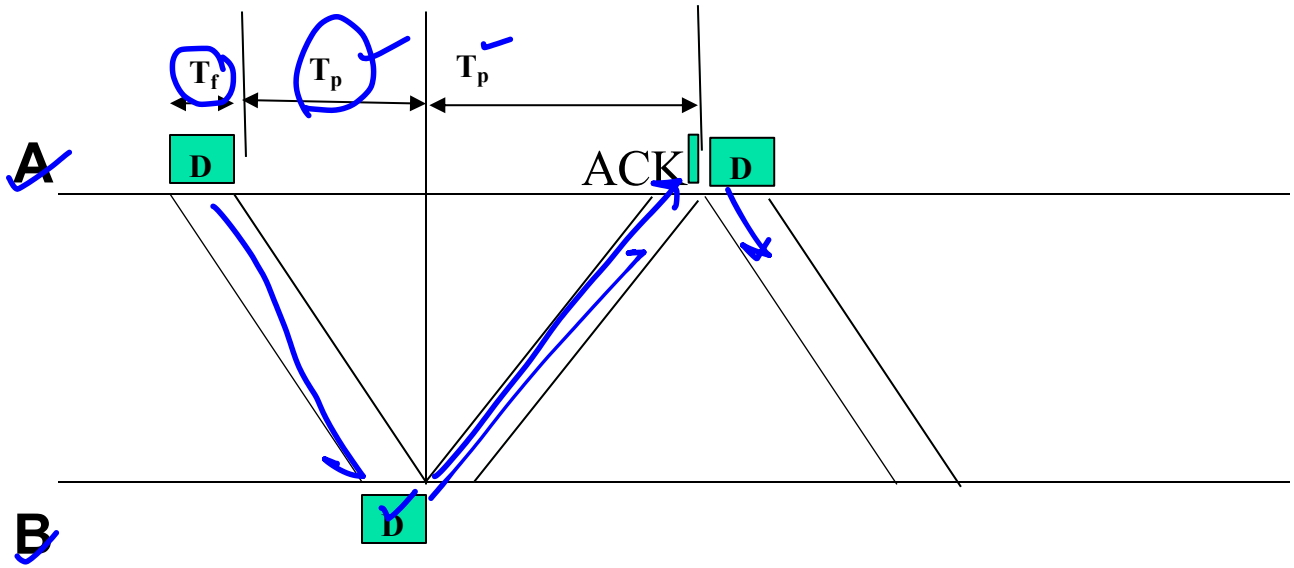
Flow diagram for Example



Link Utilization in Stop-and-Wait ARQ

- Only data frame is sent at a time and each frame is individually acknowledged.
- The data frame and acknowledgement take a certain amount of propagation time to travel across the link
- Large propagation time makes the mechanism very inefficient from the point of view of link utilization

Sequence of events and associated time instants



Link Utilization

- Let average size of a frame is L and the data rate is R
  - Complete transmission time T_f for one frame is $T_f = L/R$
- If T_p is the propagation time, the frame is completely received by receiving device (B) after $T_f + T_p$.

Link Utilization

- To calculate the best possible utilization of the link
  - The receiver sends back acknowledgment immediately on receipt of a data frame
  - The size of the acknowledgement frame is very small, and
  - There are no errors in the data frame or its acknowledgement

Link Utilization

- The sender receives the acknowledgement after time $T_f+T_p+T_p=T_f+2T_p$
- Out of the total time T_f+2T_p the sender has utilized the time Tf amount only.
- Therefore link utilization efficiency U is

$$U=\frac{T_f}{T_f+2T_p}=\frac{1}{1+\frac{2T_p}{T_f}}=\frac{1}{1+\frac{2T_pR}{L}}=\frac{1}{1+2\alpha}$$

$\alpha=\frac{R}{L}$

Bandwidth-delay product:

- Stop-and-wait protocol is very inefficient if our channel is thick and long
- Thick means our channel has a large bandwidth (high data rate)
- By long means round trip delay is long
- The product of these two is called *bandwidth-delay* product

Example

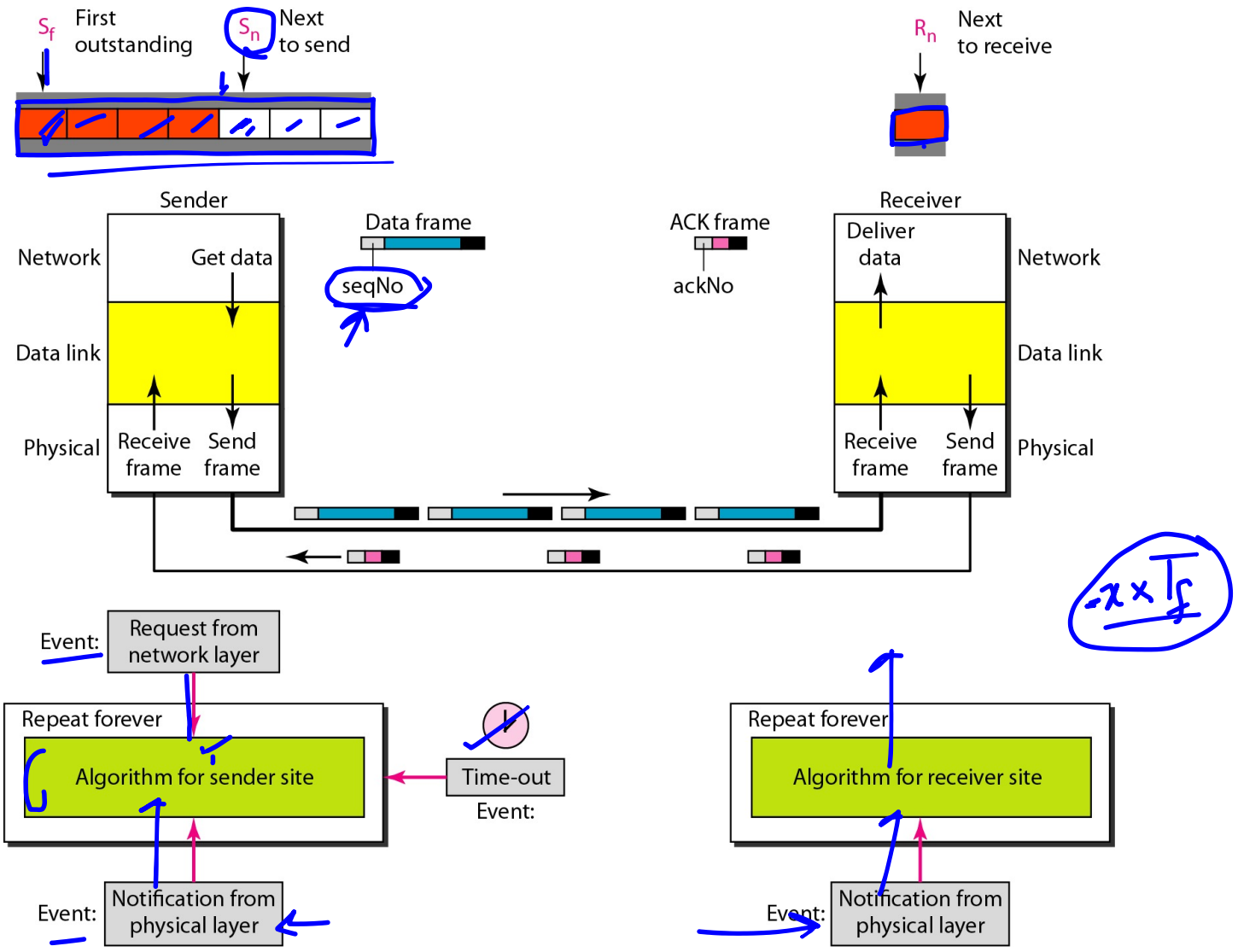
- Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip.
  - What is the bandwidth-delay product?
  - If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

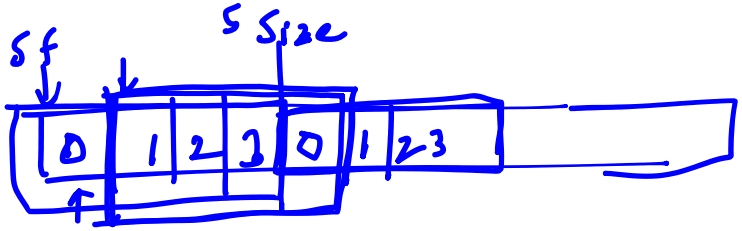
The bandwidth-delay product is

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}$$

Go-Back-N ARQ


S_f First outstanding
S_n Next to send
R_n Next to receive
Sender
Receiver
Network
Get data
Data link
Physical
Receive frame
Send frame
Data frame
seqNo
ACK frame
ackNo
Deliver data
Network
Data link
Physical
Receive frame
Send frame
Event:
Request from network layer
Repeat forever
Algorithm for sender site
Time-out
Event:
Event:
Notification from physical layer
Repeat forever
Algorithm for receiver site
Event:
Notification from physical layer

Go-Back-N ARQ

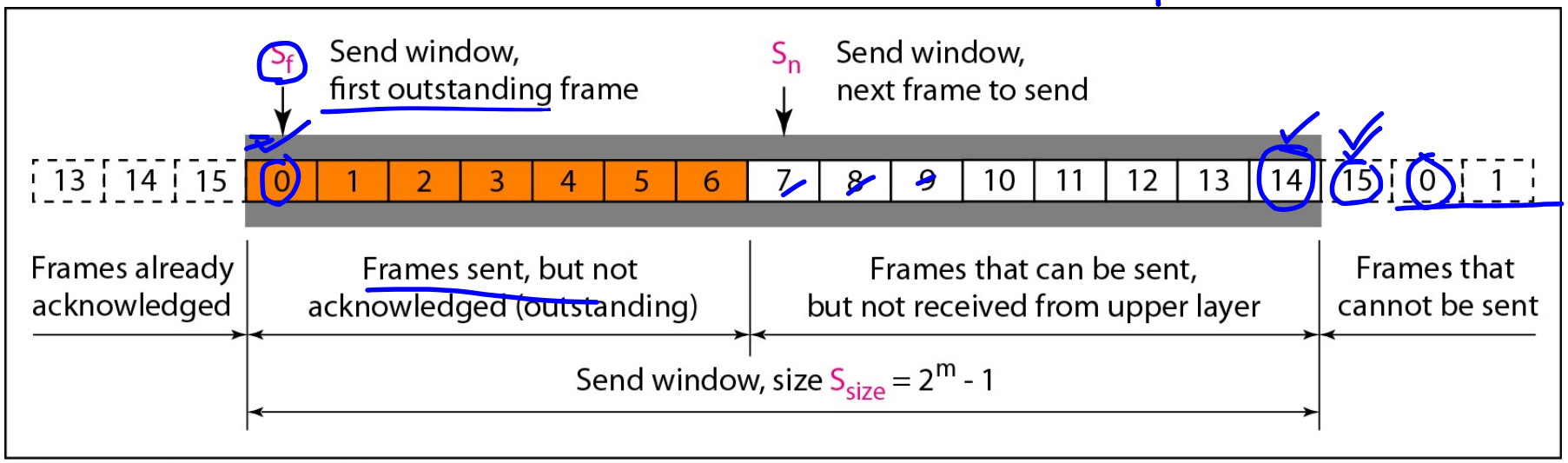


- It improves the efficiency of transmission
  - Multiple packets must be in transition while the sender is waiting for acknowledgement
- The receiver can only buffer one packet
- Here, a sequence number is maintained among the frames
  - the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.
- The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}.
- The send window can slide one or more slots when a valid acknowledgment arrives.

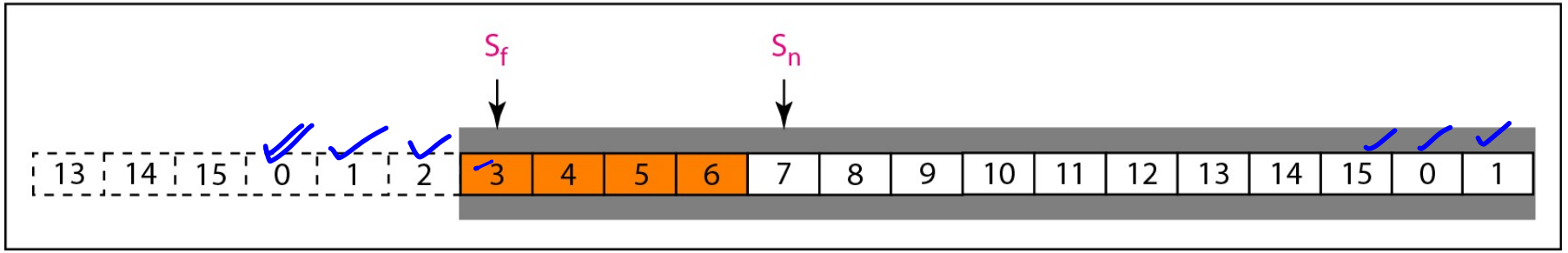
Go-Back-N ARQ

- There can be a timer for each packet that is sent
  - But one clock is maintained
  - This is due to timer for the first outstanding packet always expires first
  - Sender resends all outstanding packets when the timer expires

Send window for Go-Back-N ARQ



a. Send window before sliding

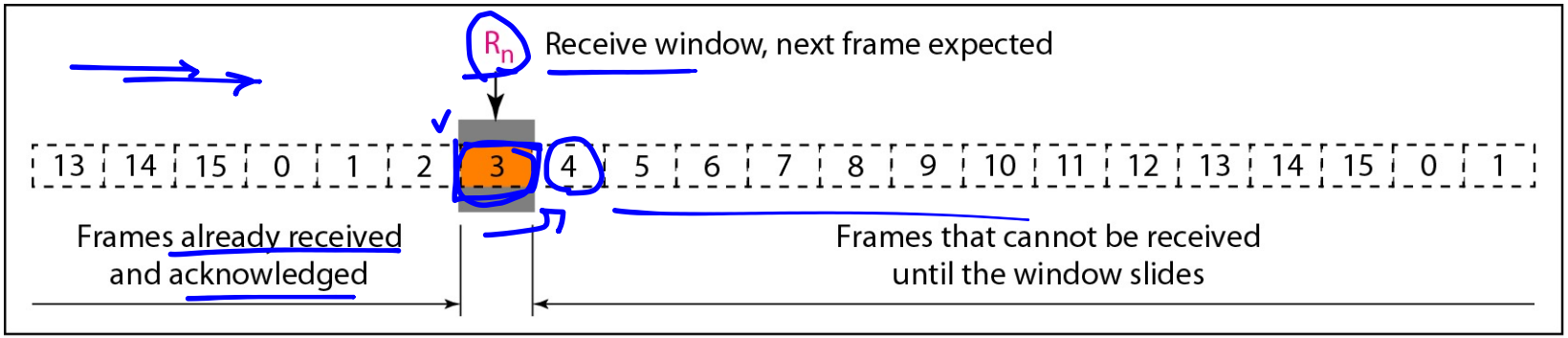


b. Send window after sliding

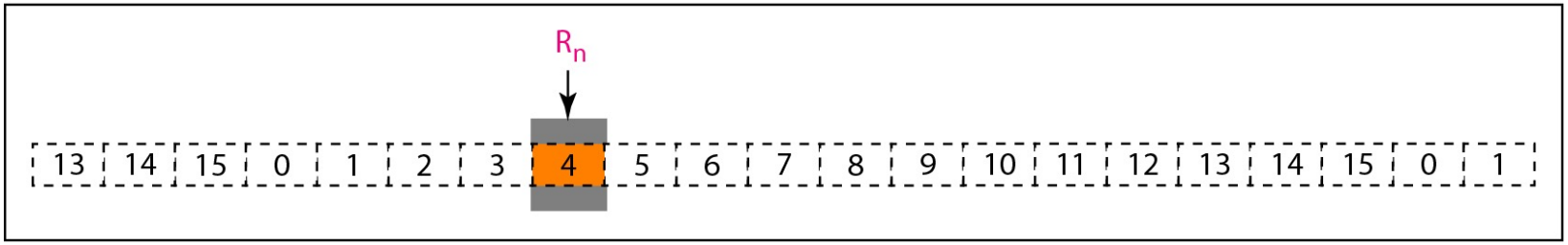
Receive window for Go-Back-N ARQ

- 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.

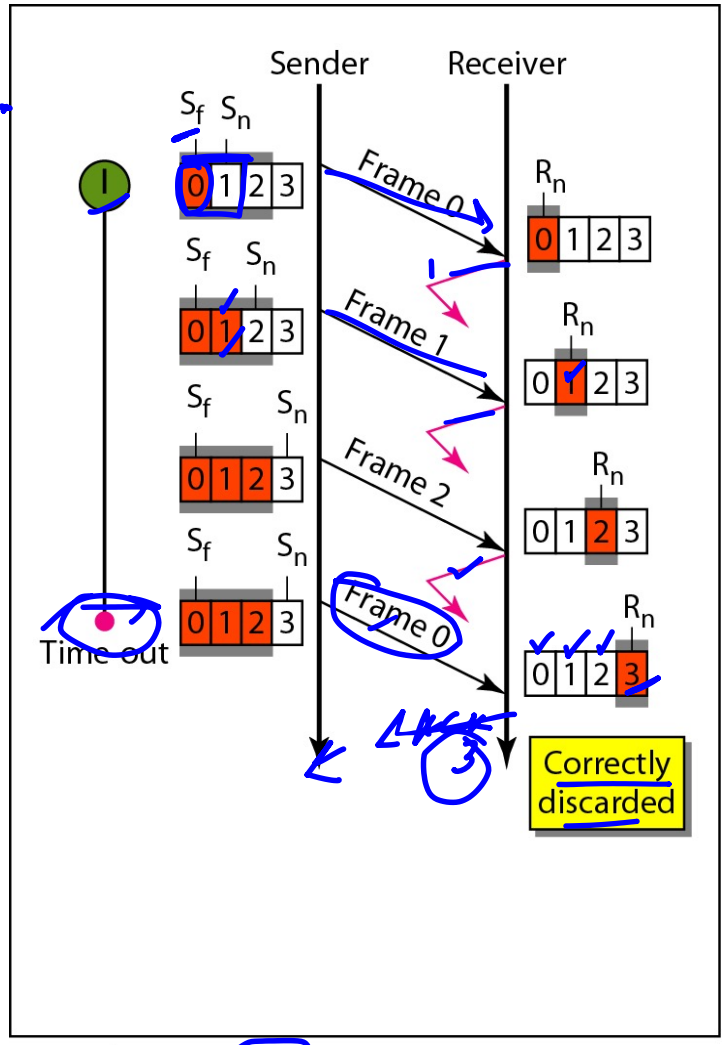


a. Receive window

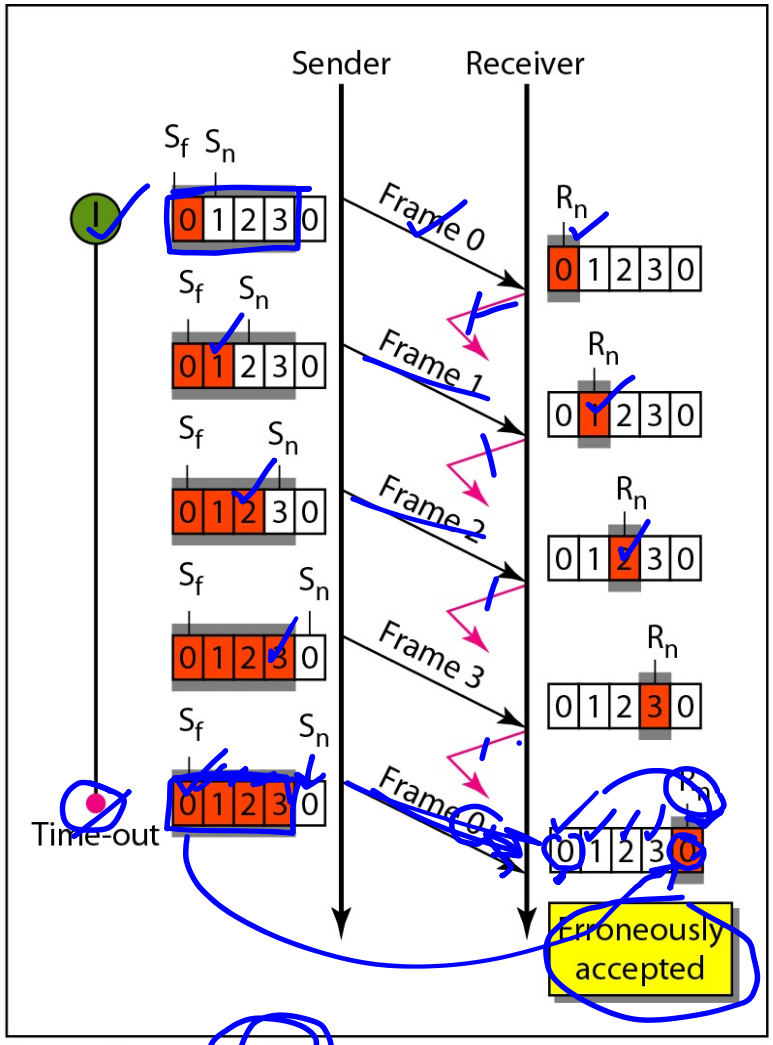


b. Window after sliding

Window size for Go-Back-N ARQ



a. Window size $< 2^m$



b. Window size $= 2^m$

Algorithm: *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 : *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 : *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  }
```

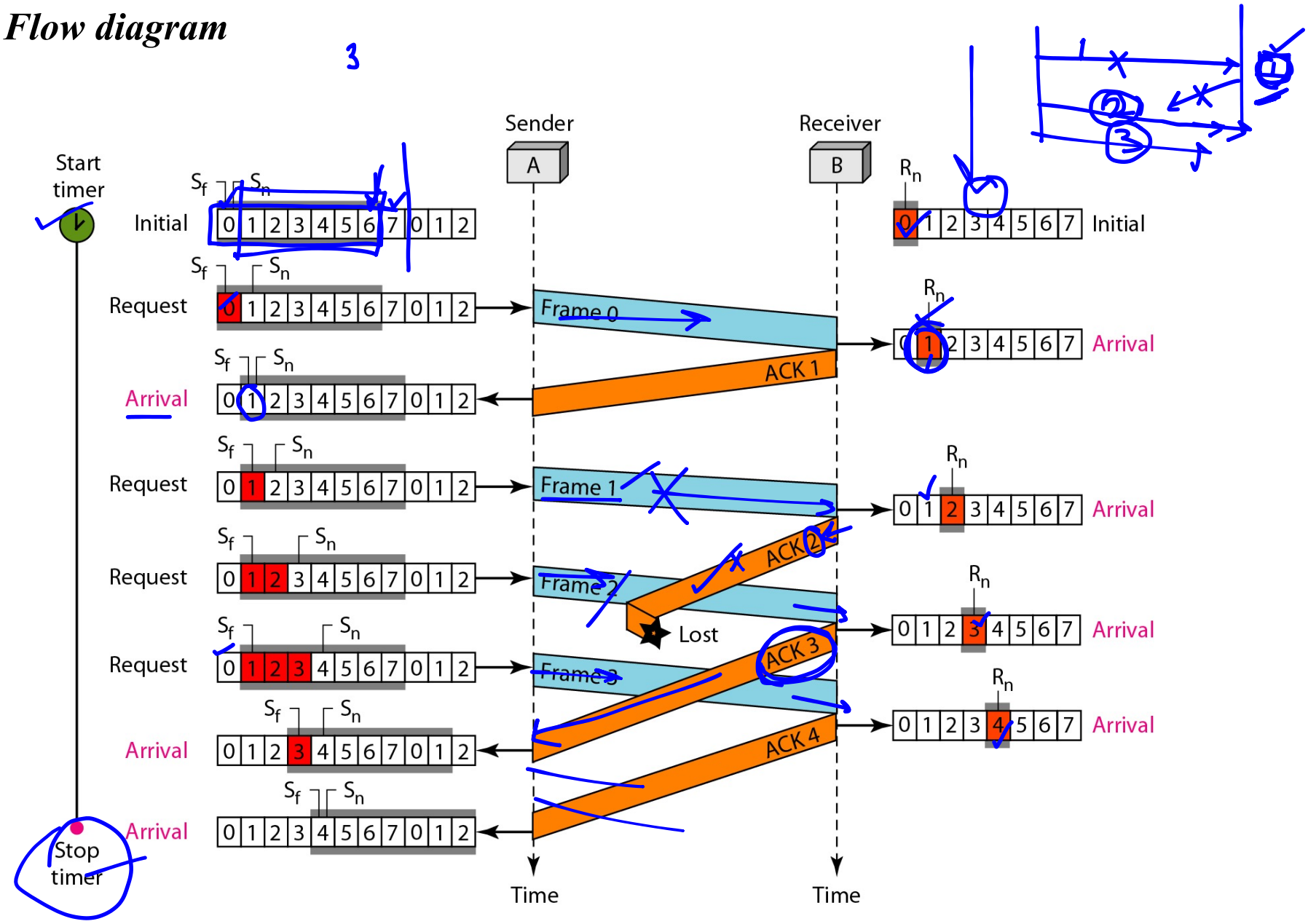
Example

- This is an example of a case where the forward channel is reliable, but the reverse is not.
- No data frames are lost, but some ACKs are delayed and one is lost.

- The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.
- After initialization, there are seven sender events.
  - Request events are triggered by data from the network layer
  - Arrival events are triggered by acknowledgments from the physical layer.

- There is no time-out event here because all outstanding frames are *acknowledged before the timer expires.*
  - Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK 3.

Flow diagram



Example

- Figure 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.

Note

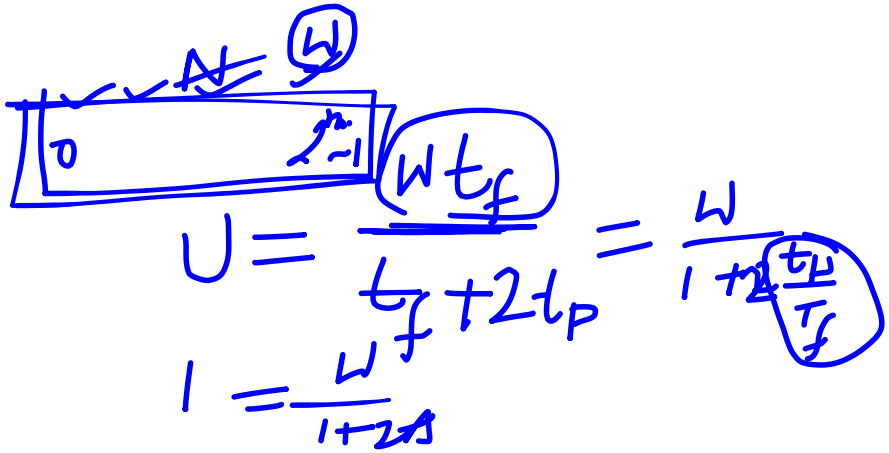
Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

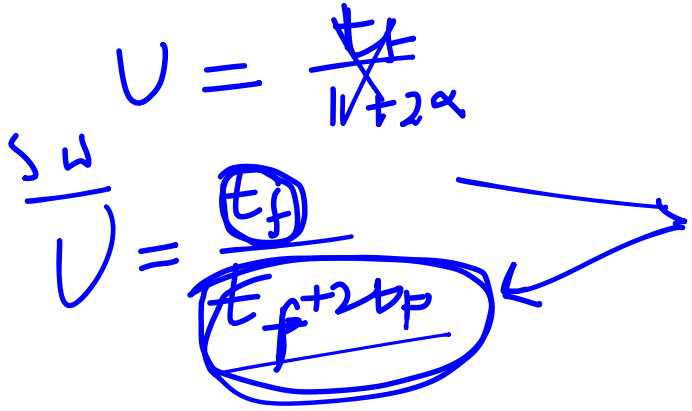
Link utilization: Go-Back-N

- **Without error:**
  - The maximum possible link utilization for no error in frame transmission can be written as

$$U = \begin{cases} 1 & for\ W \geq 1 + 2A \\ \frac{W}{1+2A} & for\ W < 1 + 2A \end{cases}$$

. Here W is the window size and $A = t_p/t_f$.





Link Utilization : Go-Back-N

- **With error:**
  - If error occurs the number of retransmissions (N_r) required to send a data frame correctly for given frame error rate (P_f) is obtained as $\mathbf{N_r = 1/(1-P_f)}$
  - When $W \geq 1 + 2A$, the link utilization is

$$U = \frac{1-P_f}{1+2P_fA}$$

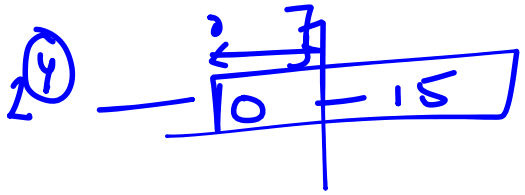
  - When W<1+2A, the link utilization is

$$U = \frac{W(1-P_f)}{(2A+1)(1-P_f+WP_f)}$$

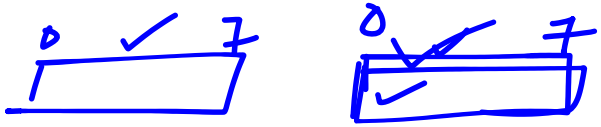
Problem in Go-Back-N ARQ

- The Go-Back-N ARQ protocol is inefficient if the underlying network protocol loses a lot of packets
- Each time a single packet is lost or corrupted, the sender resends all outstanding packets
  - Even though some of the packets have been received safe and sound but out of order.
  - This is other effects on network like congestion

Selective Repeat ARQ



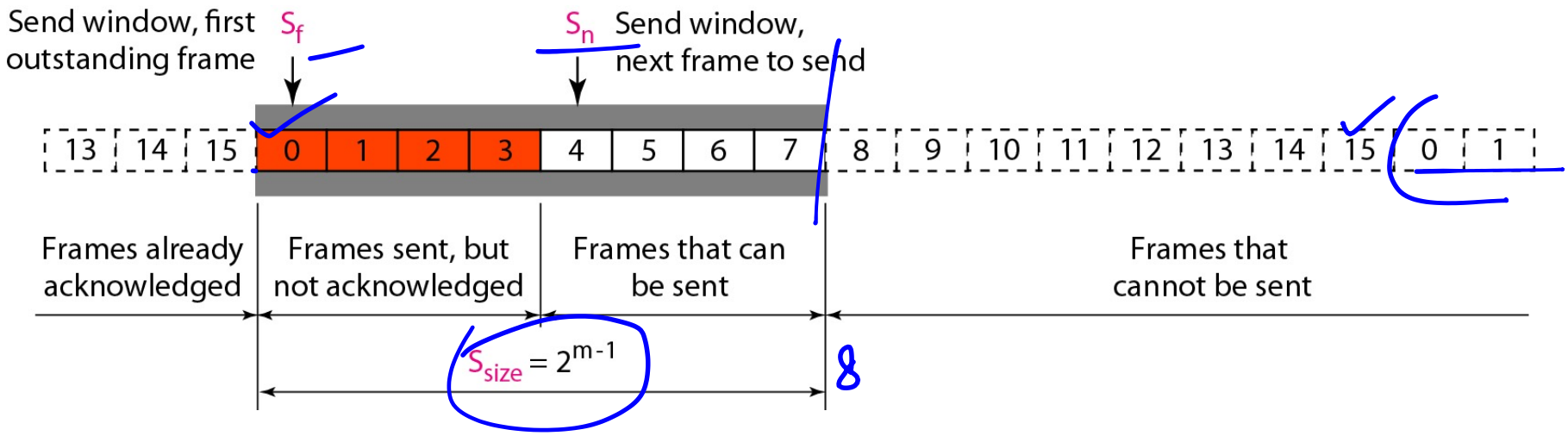
- Improved the performance of Go-Back-N ARQ
- This mechanism resends only selective packets, those that are actually lost
- It uses two windows:
  - A send window and a receive window
  - There are differences between the windows used in Go-Back-N ARQ and this protocol



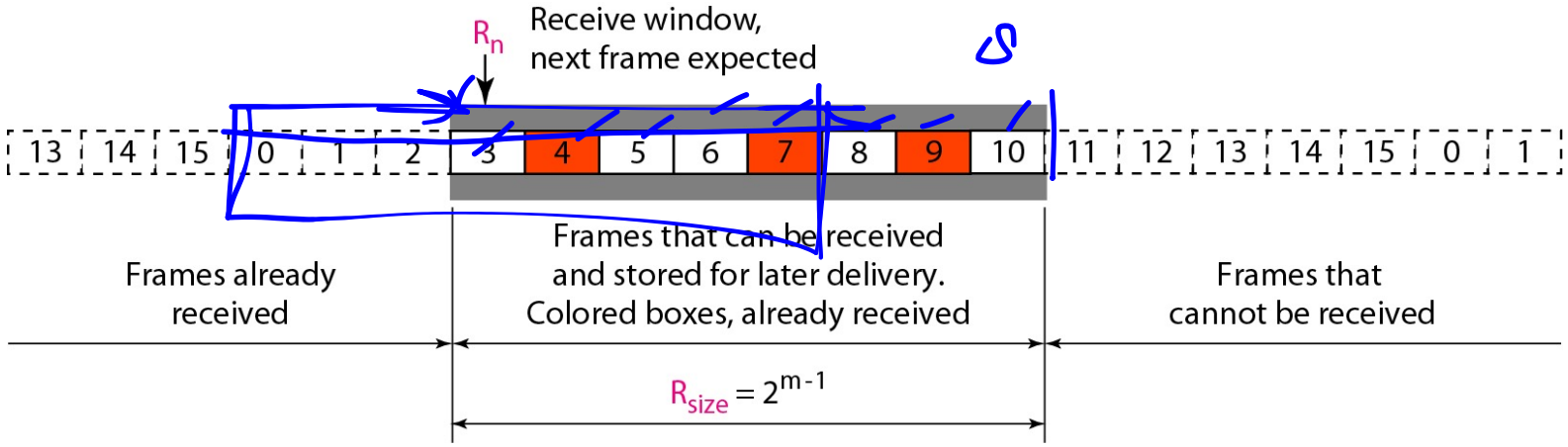
Selective Repeat ARQ

- Differences in windows:
  - First, maximum size of the send window is much smaller
  - The receive window size is the same size as the send window
- The Selective repeat ARQ allows as many packets as the size of the receive window to arrive out of order and kept a copy of those

Send window for Selective Repeat ARQ



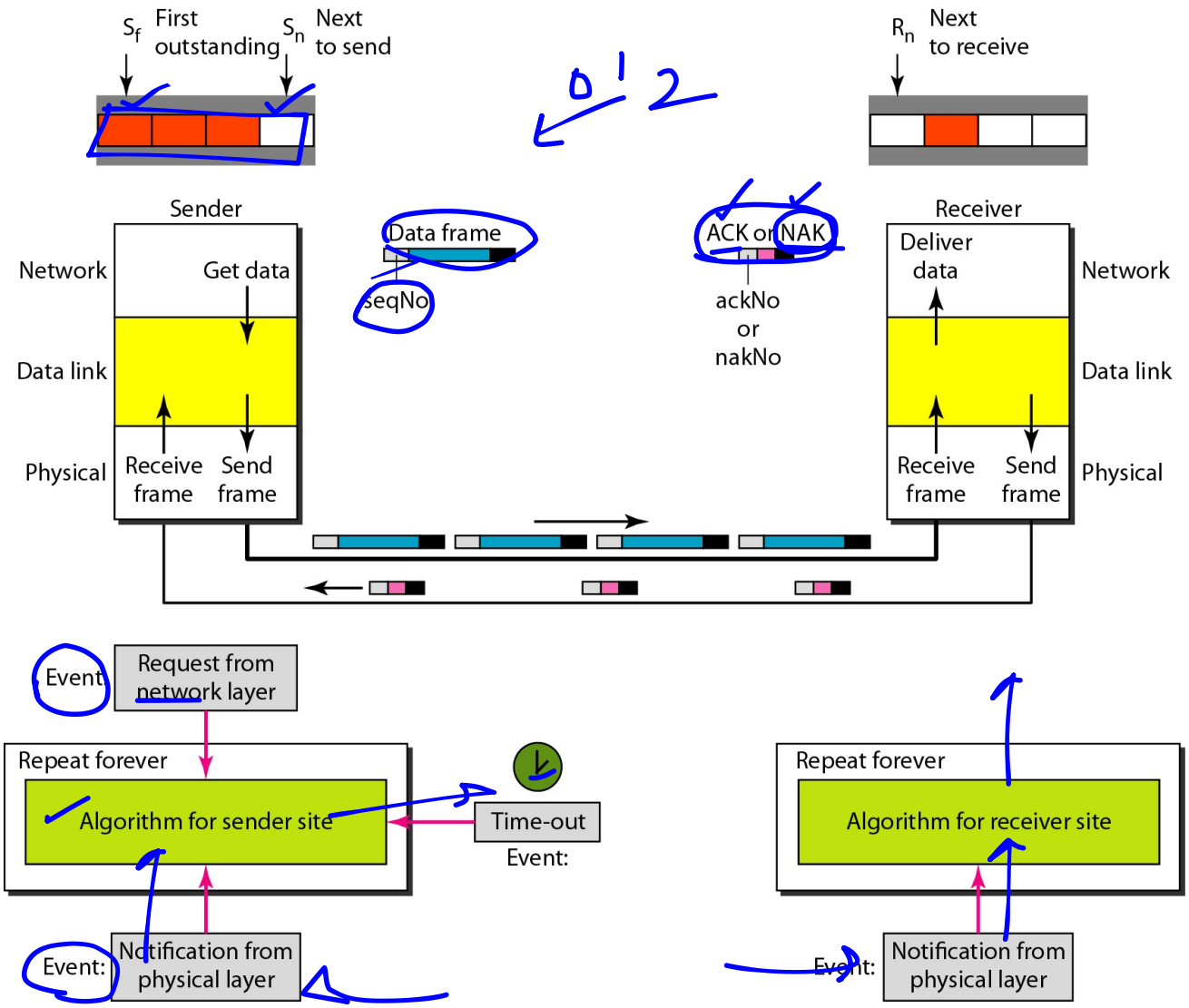
Receive window for Selective Repeat ARQ



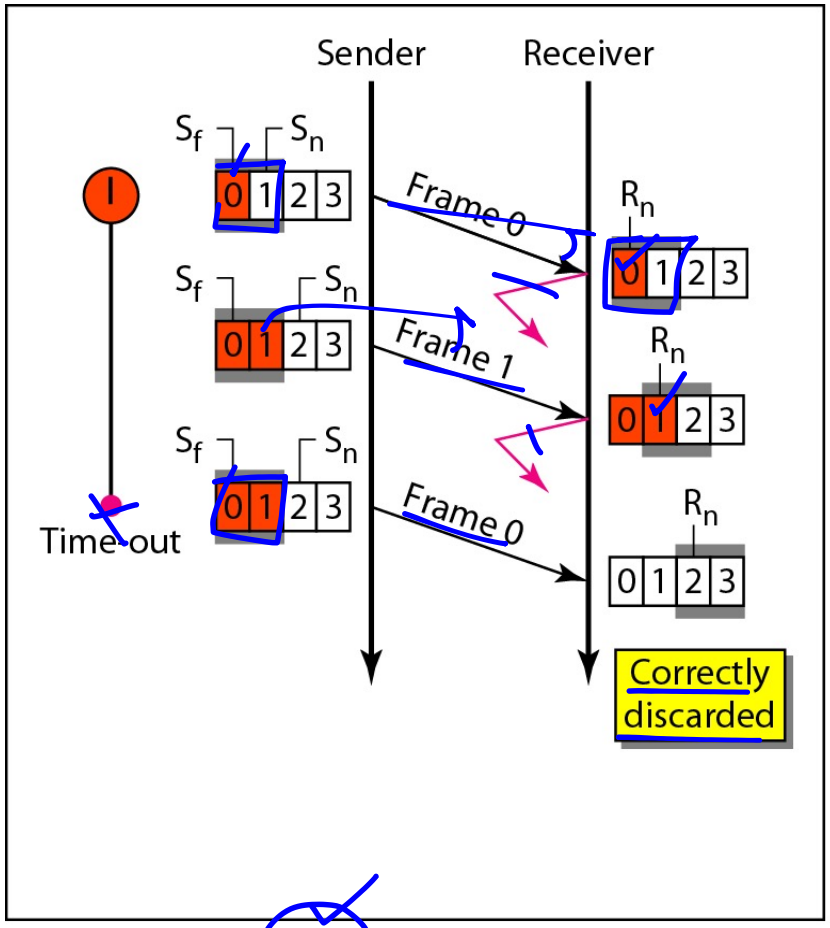
Timer used

- Uses one timer for each outstanding packet
- When a timer expires, only the corresponding packet is resent
- In Go-Back-N ARQ treats outstanding packets as a group
  - In this case each packet is treated individually
- But implementable using a single timer

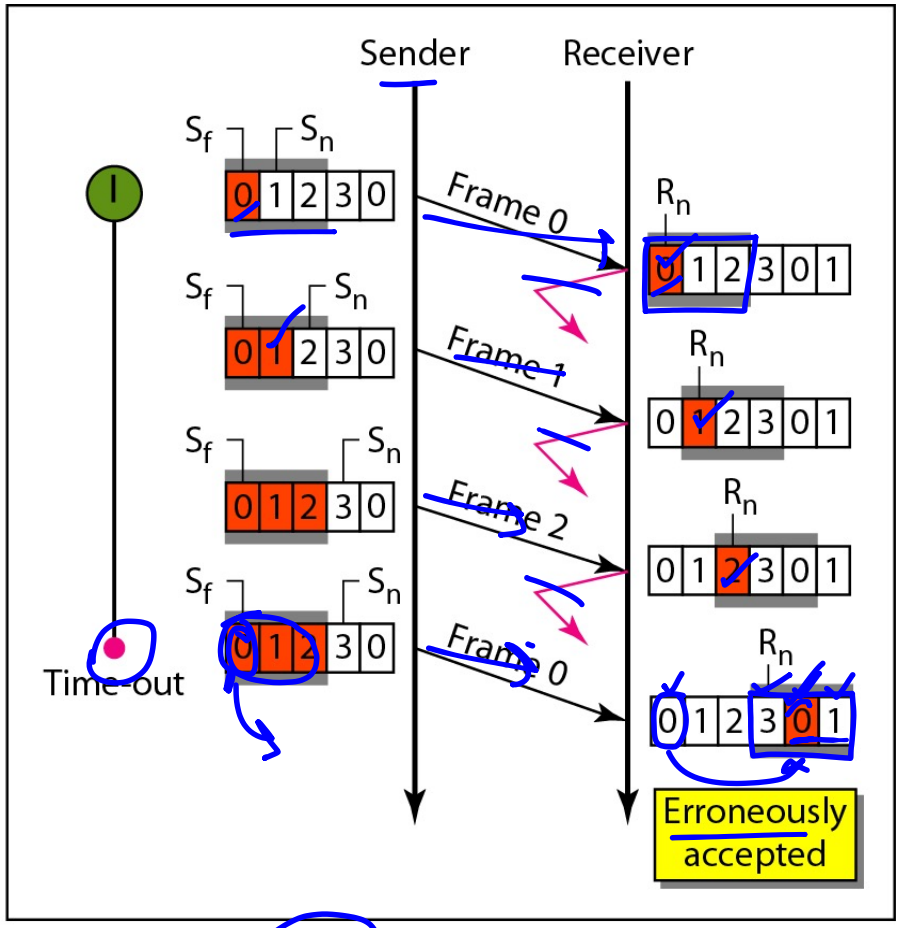
Design of Selective Repeat ARQ



Selective Repeat ARQ, window size



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



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

Note

In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.

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

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

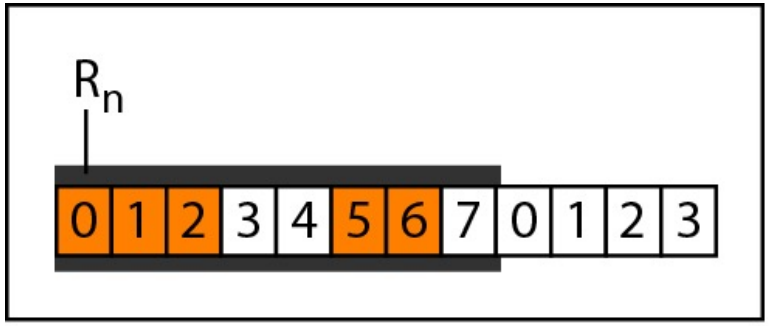
Algorithm : *Receiver-site Selective Repeat algorithm*

```
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);
```

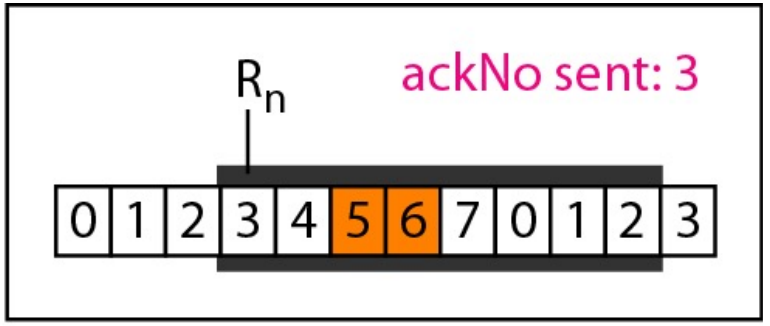
Algorithm :*Receiver-site Selective Repeat algorithm*

```
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  }
```

Delivery of data in Selective Repeat ARQ



a. Before delivery



b. After delivery

Example

- 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 (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.

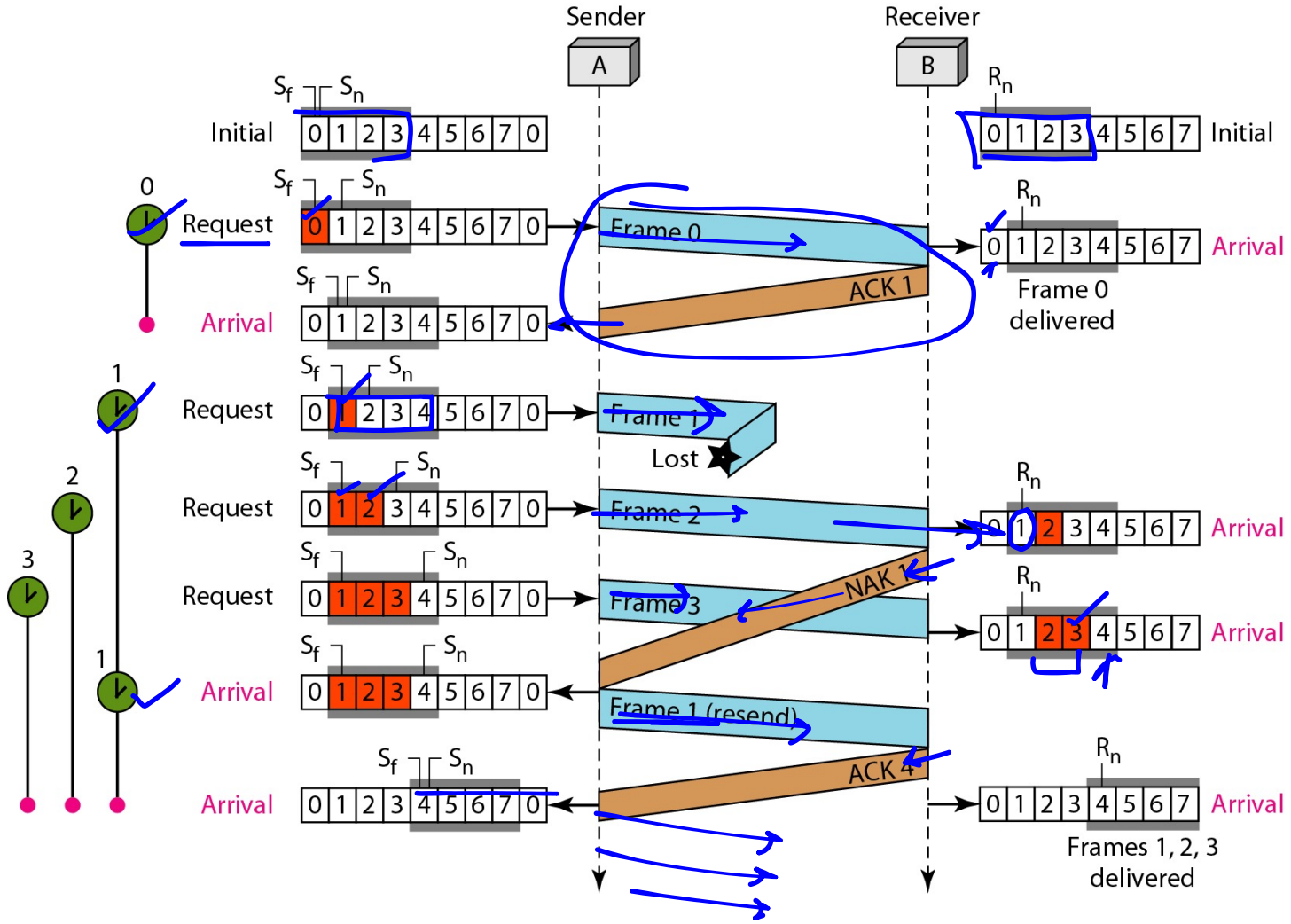
Example (continued)

- 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.

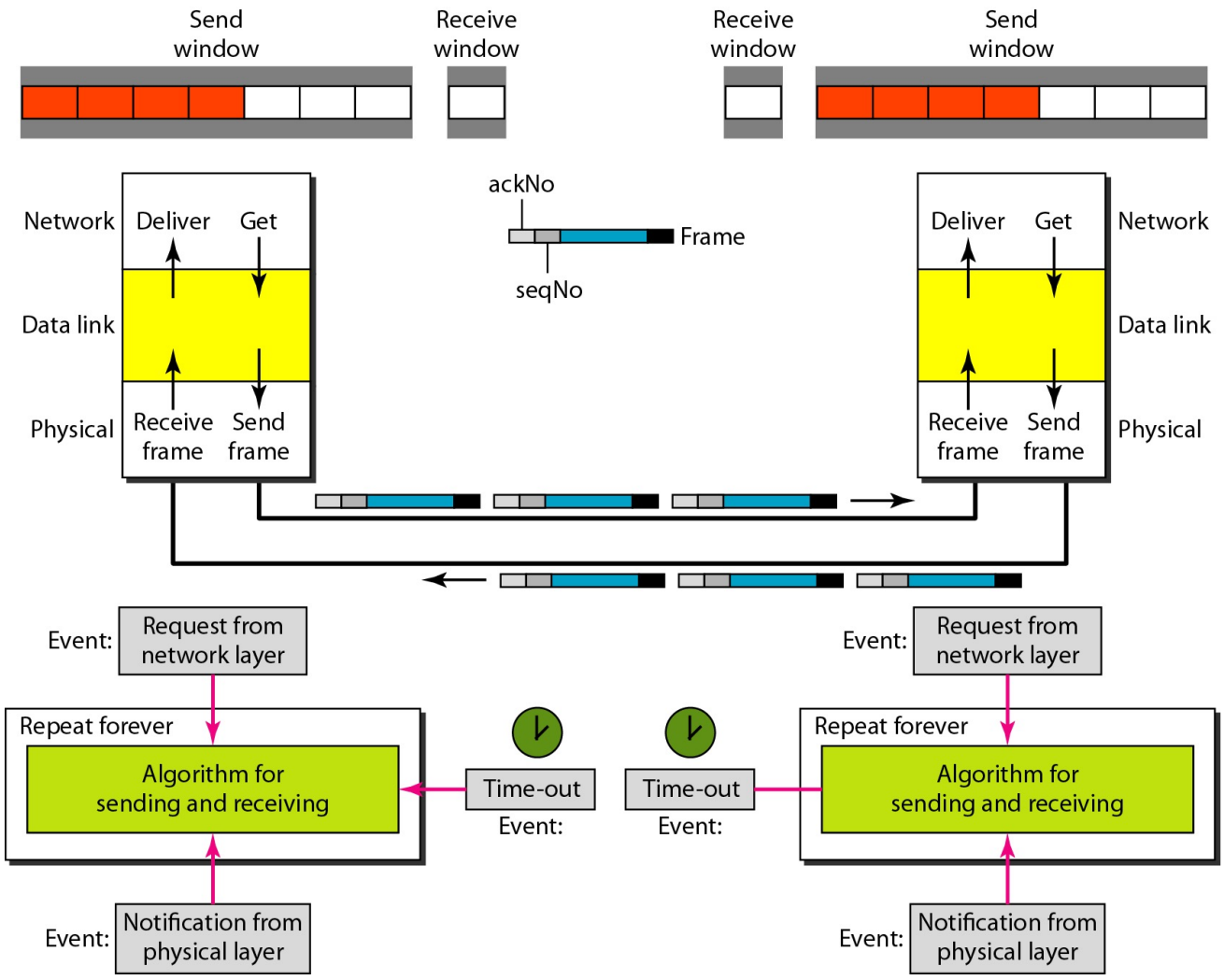
Example (continued)

- 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.

Flow diagram: Case study

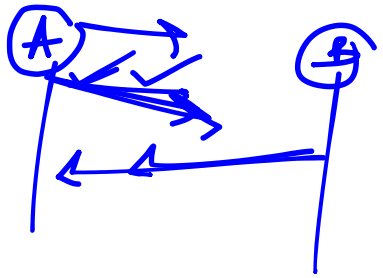


Design of piggybacking in Go-Back-N ARQ

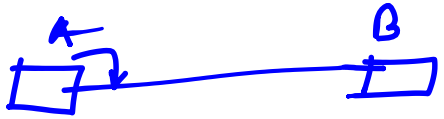


Bidirectional Protocol: Piggybacking

- All the protocols discussed earlier are unidirectional as the data packets are flowing in one direction and only acknowledgement flowing in other direction
- In real life data packets are normally flowing in both the directions
  - Means both data packets and acknowledgements should flow in both the directions
  - ***Piggybacking*** helps to improves the efficiency by flowing data and acknowledgement in both directions



Mathematical Examples:



- ***Example 1:*** Suppose that the stop-and-wait ARQ protocol is used on a link with bit rate of 64Kbps and 20ms propagation delay. Assume that the transmission time for the acknowledgement and the processing time at nodes are negligible. What is the minimum frame size in bytes to achieve a link utilization of at least 50%?

Ans:

Bit rate (B)= 64 Kbps. Propagation delay T_p= 20 ms.

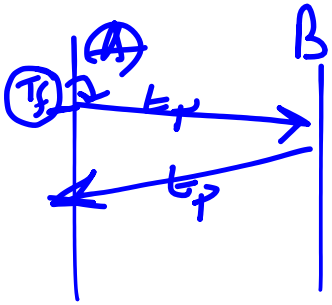
Link utilization U= 1/(1+2A), where A= T_p/T_f

➔ ½ = *1/(1+2A)* ➔ *A = ½*

Hence, $T_f = 2T_p$

If length of the frame is L bits, then $L/B = 2T_p$ ➔ $L = 2 \times B \times T_p$

So, L= 2 × 20 × 10^{-3} × 64 × 10^3 = 2560 bits = 320 bytes



Mathematical Examples:

- ***Example 2:*** A 20Kbps satellite link has a propagation delay of 400ms. The transmitter employs the 'Go-Back-N ARQ' scheme with n set to 10. Assuming that each frame is 100 bytes of long, what is the maximum data rate possible?

Ans:

Data rate = Throughput = U × ***Bandwidth***

$= [W_s / (1 + 2A)] \times$ ***Bandwidth***

$A = \frac{T_p}{T_f}$

Given $W_s = 10$, Length of each frame (L)=100 bytes = 800 bits

Propagation delay(T_p) = 400ms = 400 ms

Bandwidth (B) = 20Kbps = 20×10^3 bps

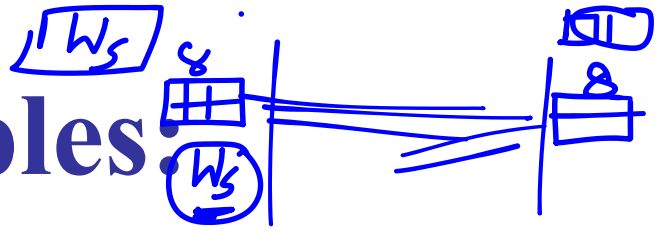
So, $T_f = L/B = (800/20)$ ms =40 ms and

$A = T_p/T_f = 400 / 40 = 10$

Now, Throughput = $10/(1+20) \times 20 \times 10^3$ bps = 9.52 Kbps

≈ 10 Kbps

Mathematical Examples



- ***Example 3:*** Consider a selective repeat sliding window protocol that uses a frame size of 1 KB to send data on a 1.5 Mbps link with a one-way latency of 50 msec. To achieve a link utilization of 60%, what is the minimum number of bits required to represent the sequence number field?

Ans:

L ✓ $\quad T_p$ ✓

B ✓

$T_f = \frac{L}{B} = \frac{1 \times 1000 \times 8}{1.5 \times 10^6} = 5.33\ ms$

$U = \frac{W_S}{1 + 2A}$ ✓ $\quad A = \frac{T_p}{T_f}$ ✓ $= \frac{50}{5.33}$

$0.6 = \frac{W_S}{1 + 2 \times \frac{50}{5.33}}$

$\Rightarrow W_S = 11.856$

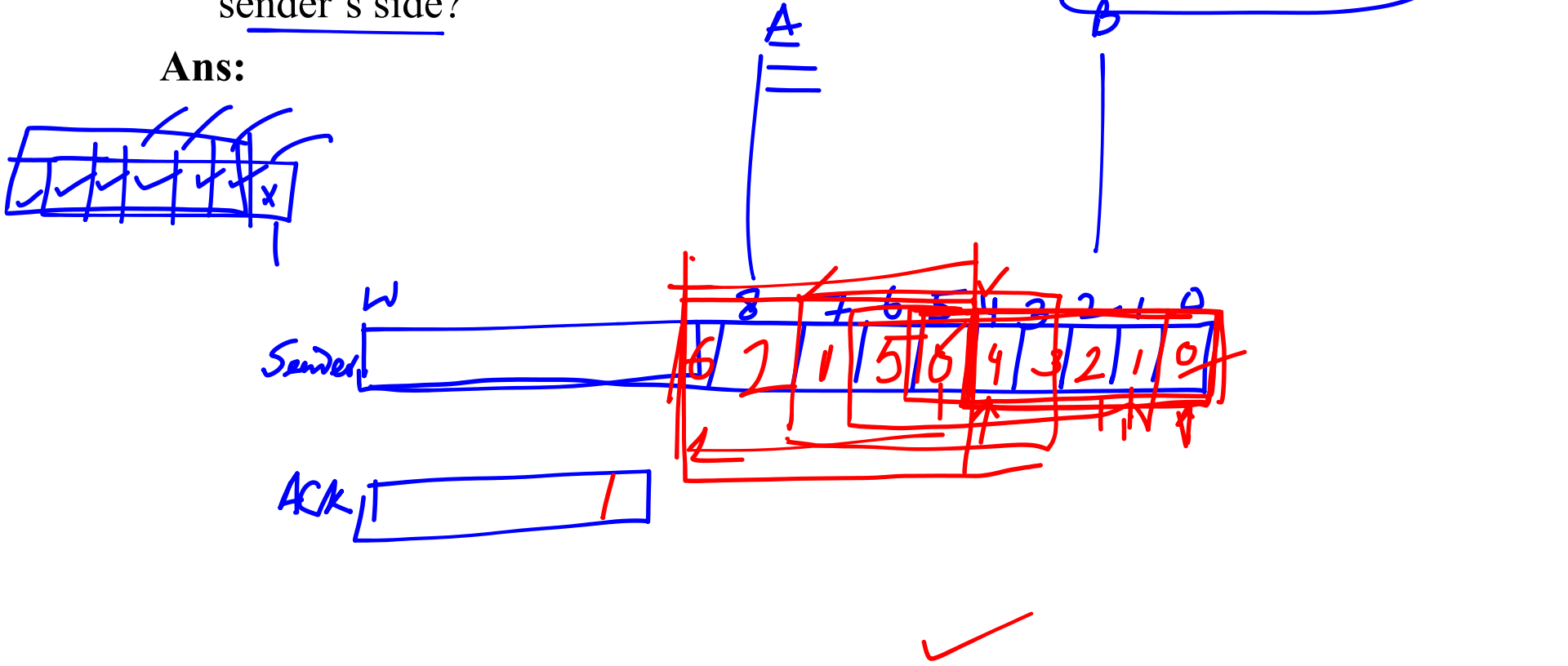
≈ 12

$2 \times 12 = 24 \rightarrow \log_2 24 \approx 5$ bit

Mathematical Examples:

- ***Example 4:*** In SR protocol, suppose frames through 0 to 4 have been transmitted. Now, imagine that 0 times out, 5 (a new frame) is transmitted, 1 times out, 2 times out, and 6 (another new frame) is transmitted. At this point, what will be the outstanding frames in sender's side?

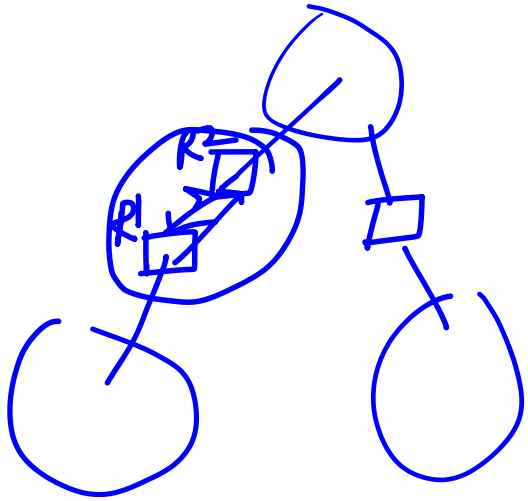
Ans:



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***
- ***We will discuss the following issues about HDLC***
  - **Configurations and Transfer Modes**
  - **Frames**
  - **Control Field**

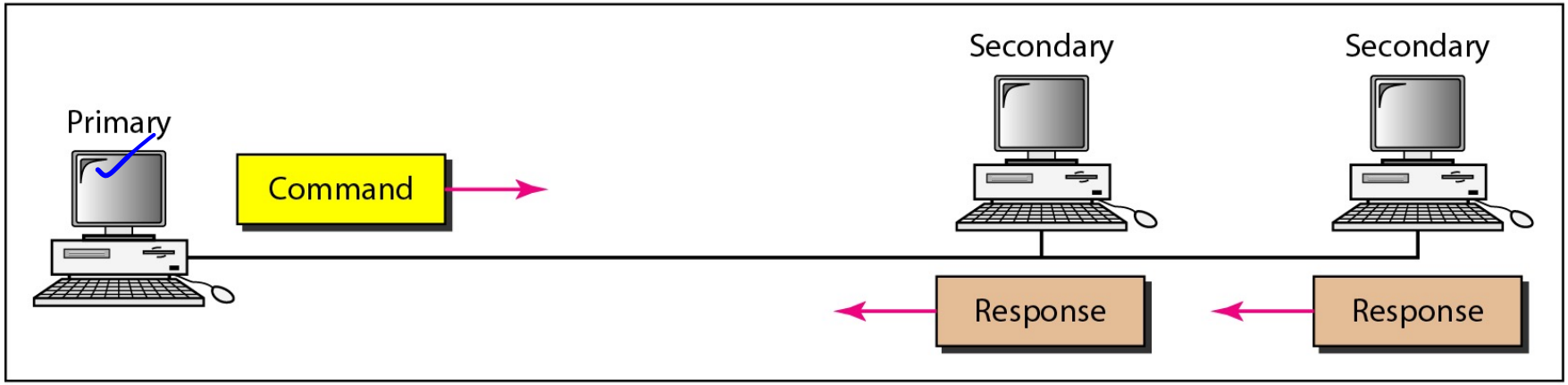
- HDLC works in two modes
  - Normal response mode (NRM)
  - Asynchronous balanced mode (ABM)



Normal response mode

- The station configuration is unbalanced
- We have one primary station and multiple secondary stations

a. Point-to-point



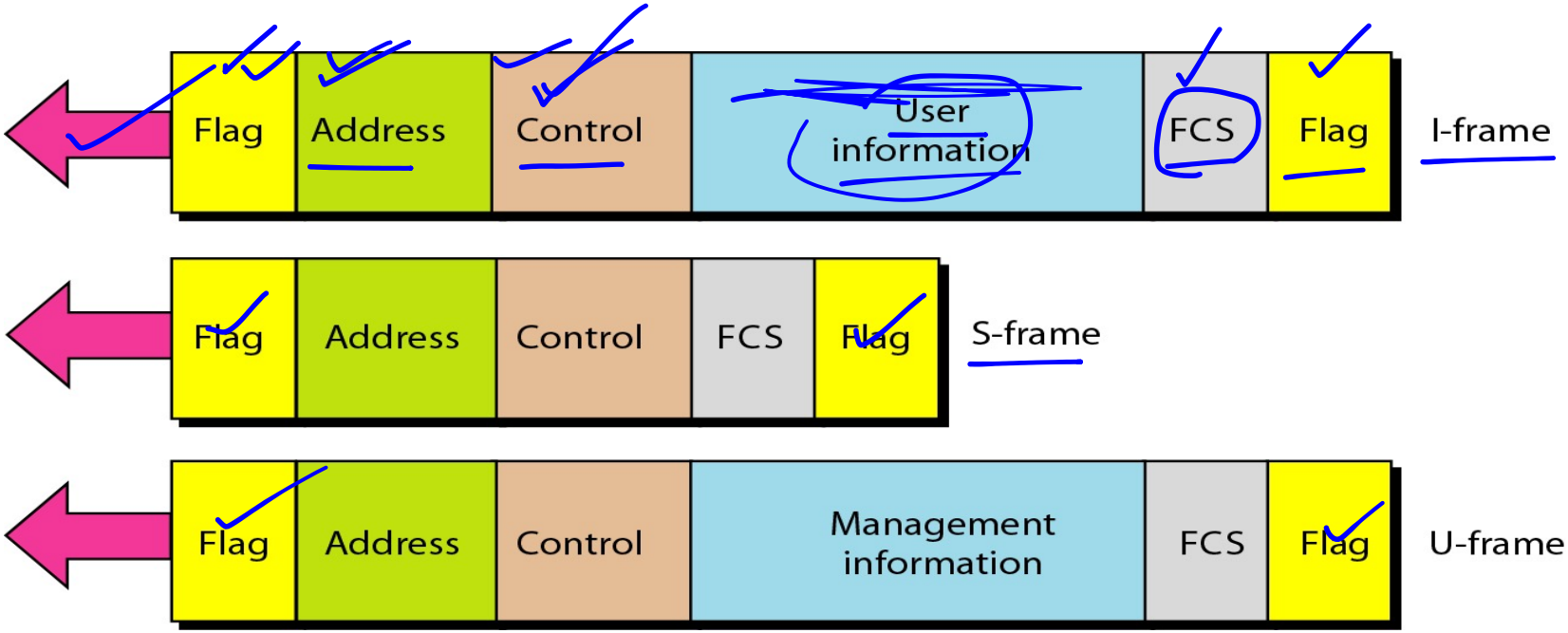
b. Multipoint

Asynchronous balanced mode

- The configuration is balanced
- The link is point-to-point and each station can function as a primary and a secondary
- This is common in todays scenario

HDLC frames

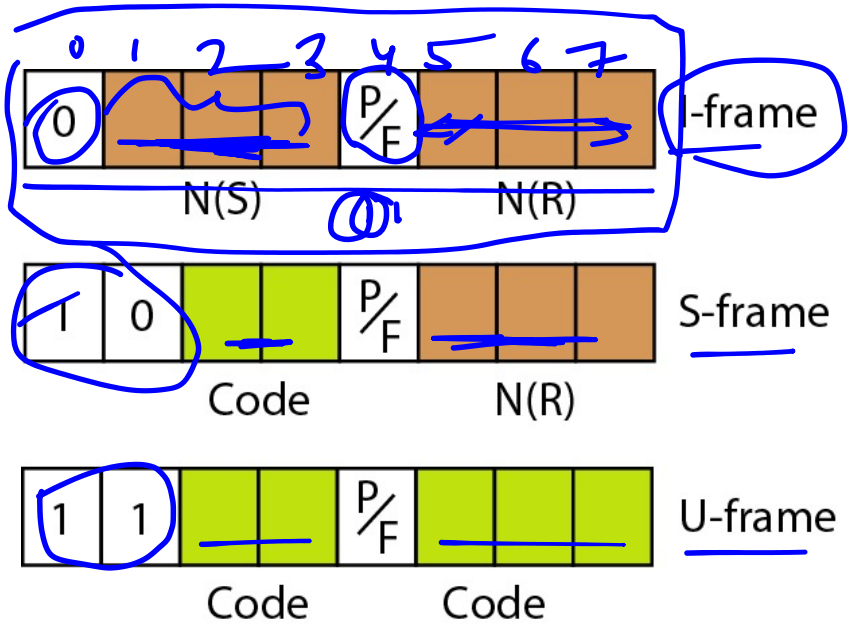
- **Supports three types of frames**
  - Information frames (I-frame) : For data link user data and control information relating to user data (piggybacking)
  - Supervisory frames (S-frames): For transport control information
  - Unnumbered frames (U-frames): Intended for managing the link itself
- **Each type of frames serves as an envelop for the transmission of a different types of messages**



Frame fields

- **Flag field:**
  - Contains synchronization patterns 01111110,identifies the beginning and end of a frame
- **Address field:**
  - Contains the address of a secondary station
  - Can be one byte or more than one byte long, depending on the need of the network
- **Control field:**
  - One or two bytes used for flow and error control

Control field format for the different frame types



- **S-Frame types: 2 bits code**
  - 00: Receive ready (RR)
  - 10: Receive not ready(RNR)
  - 01:Reject (Rej)
  - 11: Selective Reject (SREJ)

- **First one or two bits:** Types of frame
- **N(S):** Sequence number of the frame
- **N(R):** For ACK or NAK when piggybacking
- **P/F:** Poll or final:
  - Poll: When frame is sent by a primary station
  - Final: When the frame is sent by a secondary station
- **Code:** Defines the types of the S/U-Frame

Frame fields

- **Information fields**
  - Contains the user data from network layer
  - Length can vary from one network to another
- **FCS field:**
  - Frame check sequence(FCS) is the HDLC error detection field
  - Contains either a 2-byte or 4-byte CRC

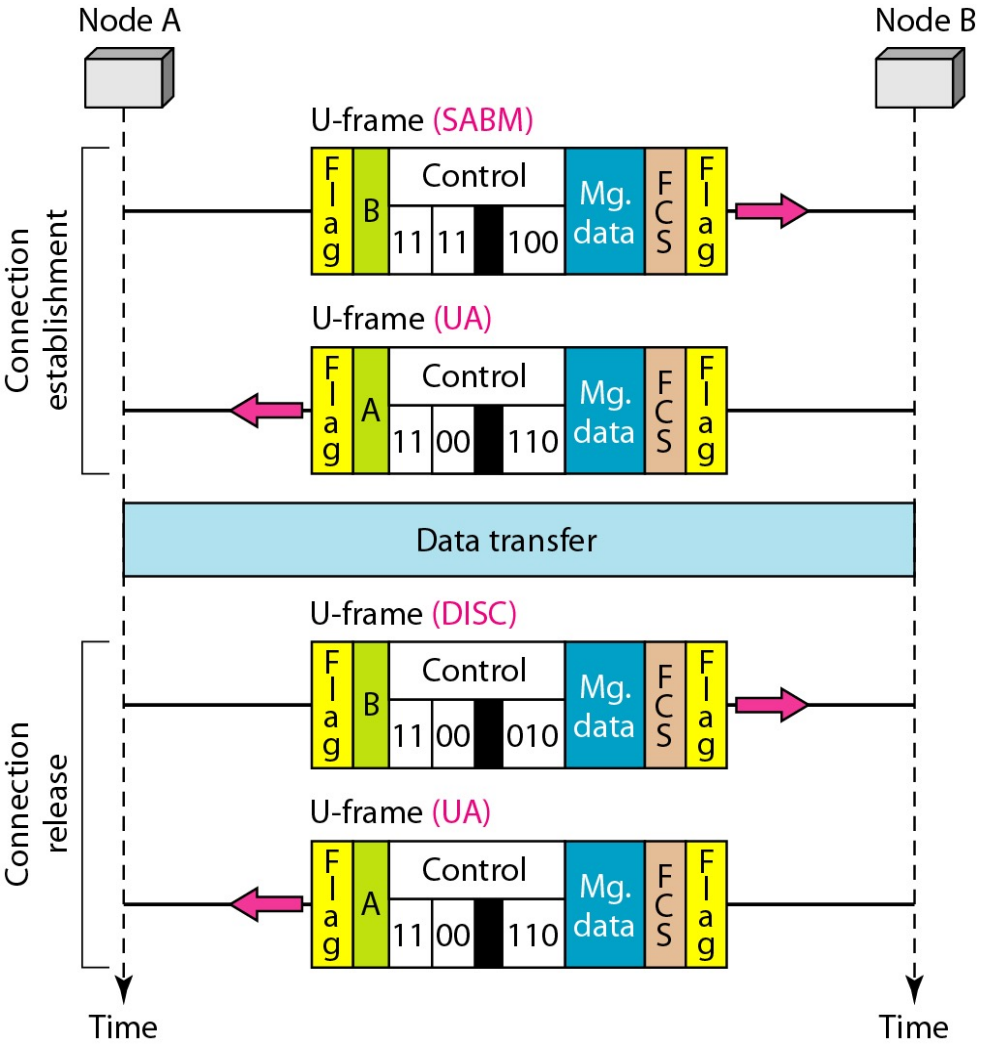
U-frame control command and response

Code	*Command*	*Response*	*Meaning*
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	**DM**	Set asynchronous balanced mode or **disconnect mode**
11 110	SABME		Set asynchronous balanced mode, extended
00 000	UI	**UI**	Unnumbered information
00 110		**UA**	**Unnumbered acknowledgment**
00 010	DISC	**RD**	Disconnect or **request disconnect**
10 000	SIM	**RIM**	Set initialization mode or **request information mode**
00 100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	**XID**	Exchange ID
10 001	FRMR	**FRMR**	Frame reject

Example

- Figure shows how U-frames can be used for connection establishment and connection release.
  - Node A asks for a connection with a set asynchronous balanced mode (SABM) frame;
  - Node B gives a positive response with an unnumbered acknowledgment (UA) frame.
  - After these two exchanges, data can be transferred between the two nodes (not shown in the figure).
  - After data transfer, node A sends a DISC (disconnect) frame to release the connection
  - It is confirmed by node B responding with a UA (unnumbered acknowledgment).

Example of connection and disconnection



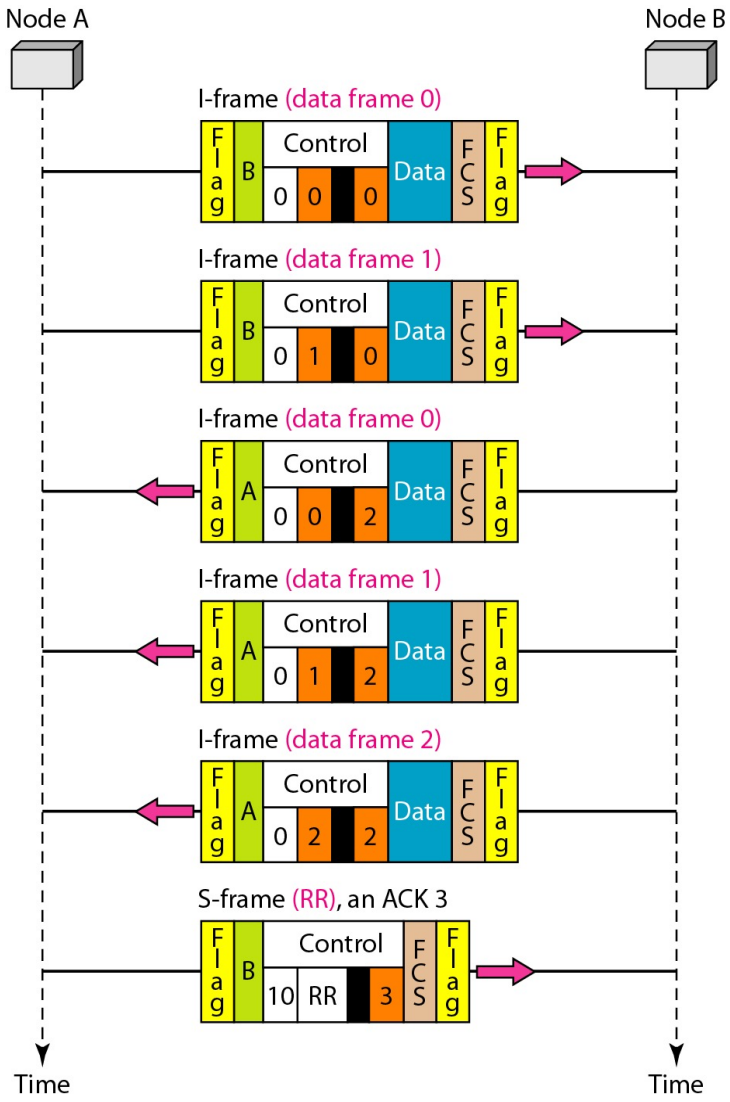
Example

- Figure shows an exchange using piggybacking. Node A begins the exchange of information with an I-frame numbered 0 followed by another I-frame numbered 1.
- Node B piggybacks its acknowledgment of both frames onto an I-frame of its own.
- Node B's first I-frame is also numbered 0 [N(S) field] and contains a 2 in its N(R) field, acknowledging the receipt of A's frames 1 and 0 and indicating that it expects frame 2 to arrive next.
- Node B transmits its second and third I-frames (numbered 1 and 2) before accepting further frames from node A.

Example (continued)

- Its N(R) information, therefore, has not changed: B frames 1 and 2 indicate that node B is still expecting A's frame 2 to arrive next.
- Node A has sent all its data.
- Therefore, it cannot piggyback an acknowledgment onto an I-frame and sends an S-frame instead.
- The RR code indicates that A is still ready to receive.
- The number 3 in the N(R) field tells B that frames 0, 1, and 2 have all been accepted and that A is now expecting frame number 3.

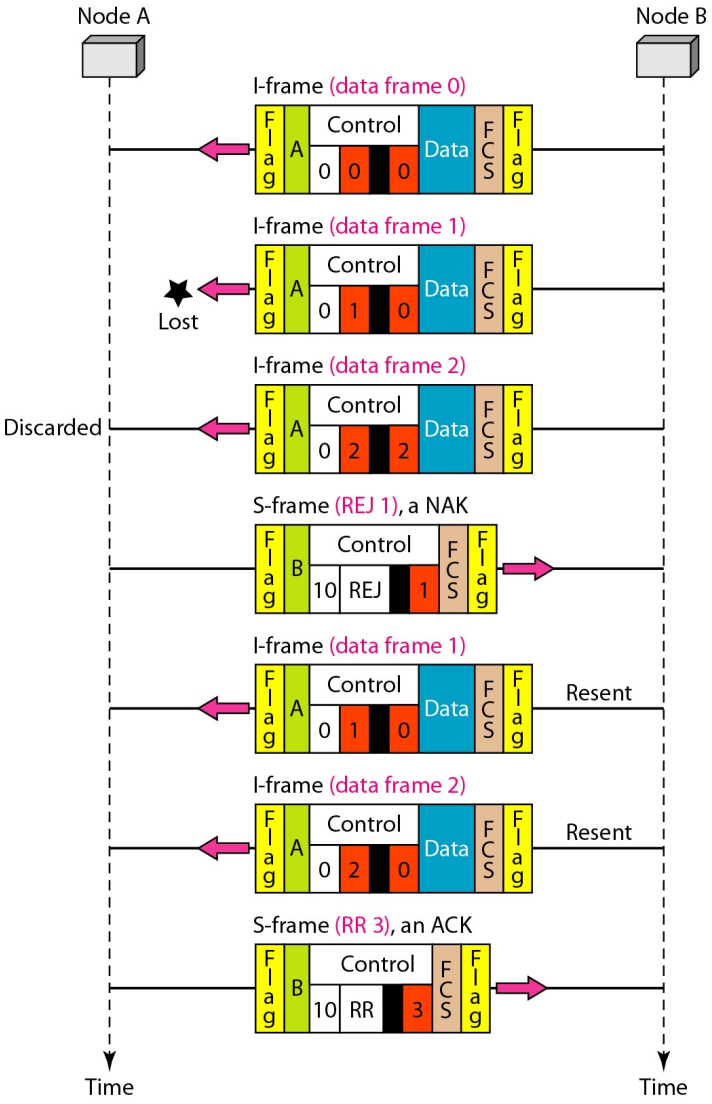
Example of piggybacking without error



Example

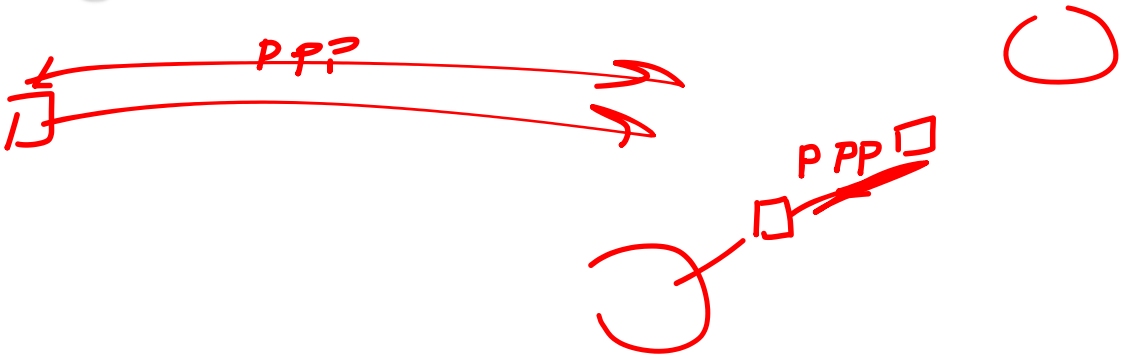
- Figure shows an exchange in which a frame is lost. Node B sends three data frames (0, 1, and 2), but frame 1 is lost.
- When node A receives frame 2, it discards it and sends a REJ frame for frame 1.
- Note that the protocol being used is Go-Back-N with the special use of an REJ frame as a NAK frame.
- The NAK frame does two things here: It confirms the receipt of frame 0 and declares that frame 1 and any following frames must be resent.
- Node B, after receiving the REJ frame, resends frames 1 and 2.
- Node A acknowledges the receipt by sending an RR frame (ACK) with acknowledgment number 3.

Example of piggybacking with error



Point-To-Point Protocol (PPP)

- *Although HDLC is a general protocol that can be used for both point-to-point and multipoint configurations*
- *One of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP).*
- *PPP is a byte-oriented protocol.*
  - *Example: Home computer connects the server of an ISP using PPP*



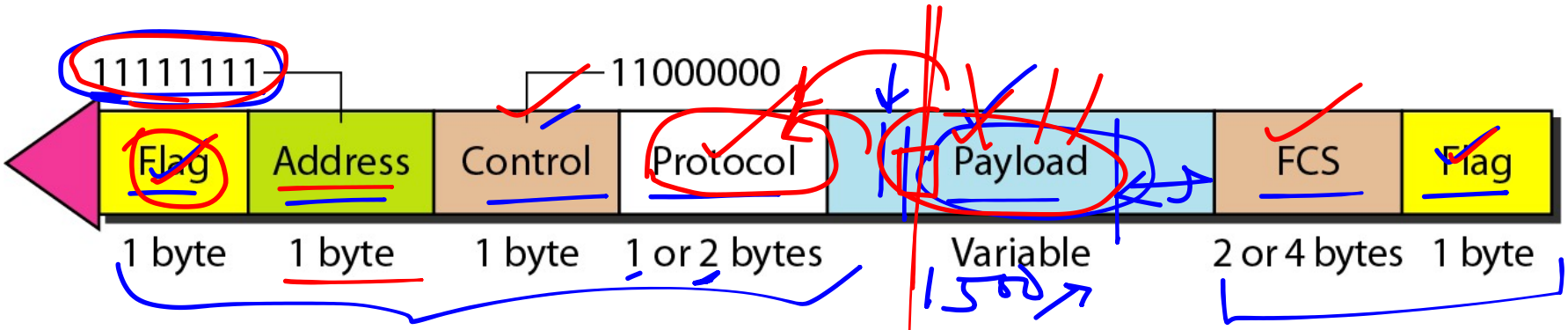
PPP

- Services provided:
  - Defines the format of the frame to be exchanged
  - How two devices negotiate the establishment of the link and the exchange of data
  - Accept payloads from several network layers(not only IP)
  - Authentication is also provided, but optional
  - New version called Multilink PPP
    - Provides connection over multiple links
    - Provides network address configuration

PPP

- Services not provided:
  - Error control
  - Has very simple mechanism for error control
    - Uses CRC to detect error
    - If the frame is corrupted → frame discarded
      - Upper layer protocol takes care the issue
  - Sequence number
  - Sophisticated addressing mechanism to handle multipoint configuration

PPP frame format: Character oriented frame

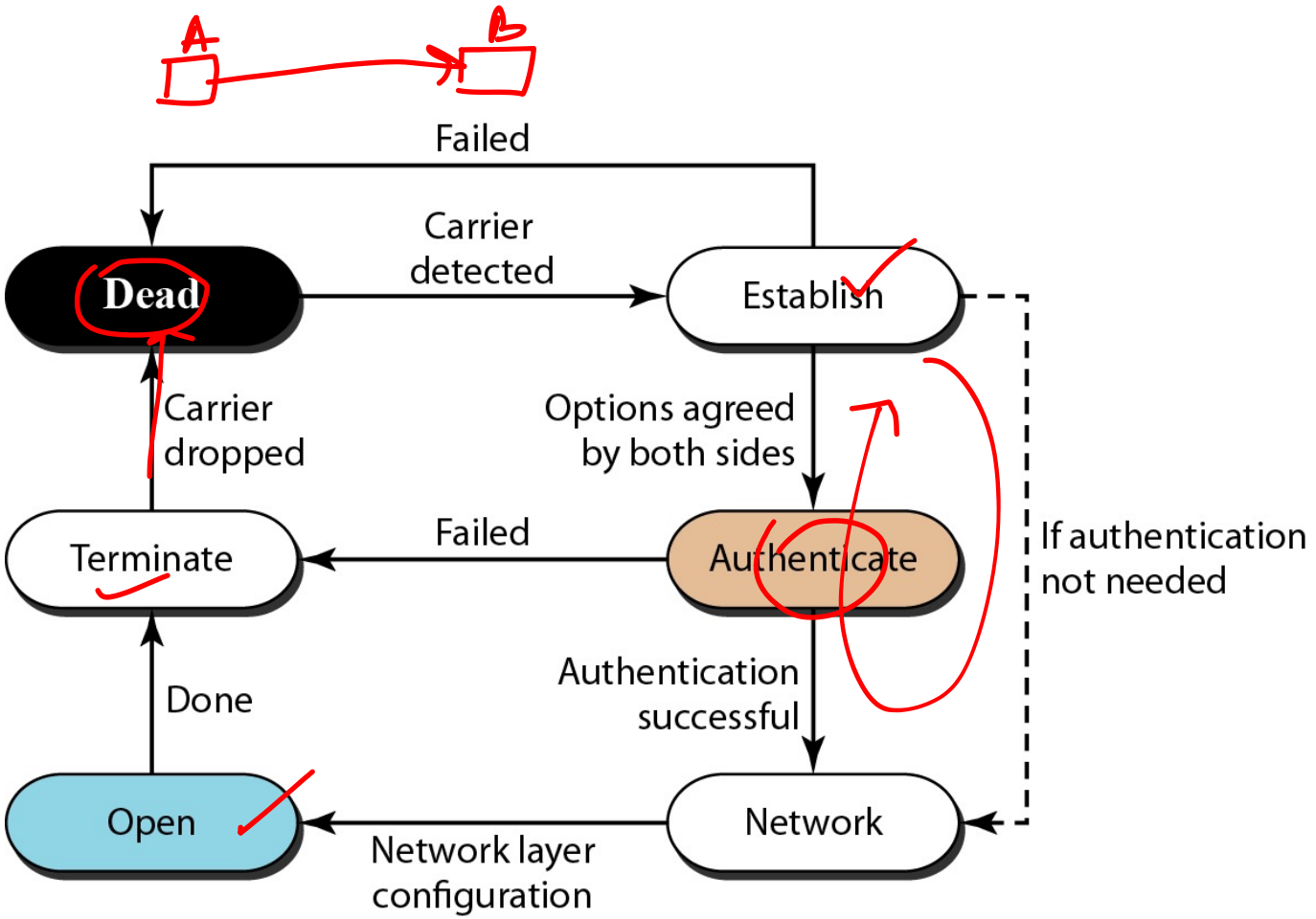


- **Flag:** 1-byte flag with the bit pattern 01111110
- **Address:** Constant value 11111111 set. Broadcast address
- **Control:** Set with constant value 00000011
- **Protocol:** Defines what is being carried in data field. Either user data or other information
- **Payload field:**
  - Data field is of maximum 1500bytes but can be changed with negotiation
  - Data field is byte stuffed if flag presents in the data
  - Padding is needed if the size is less than the maximum byte
- **FCS:** A 2-byte or 4-byte standard CRC

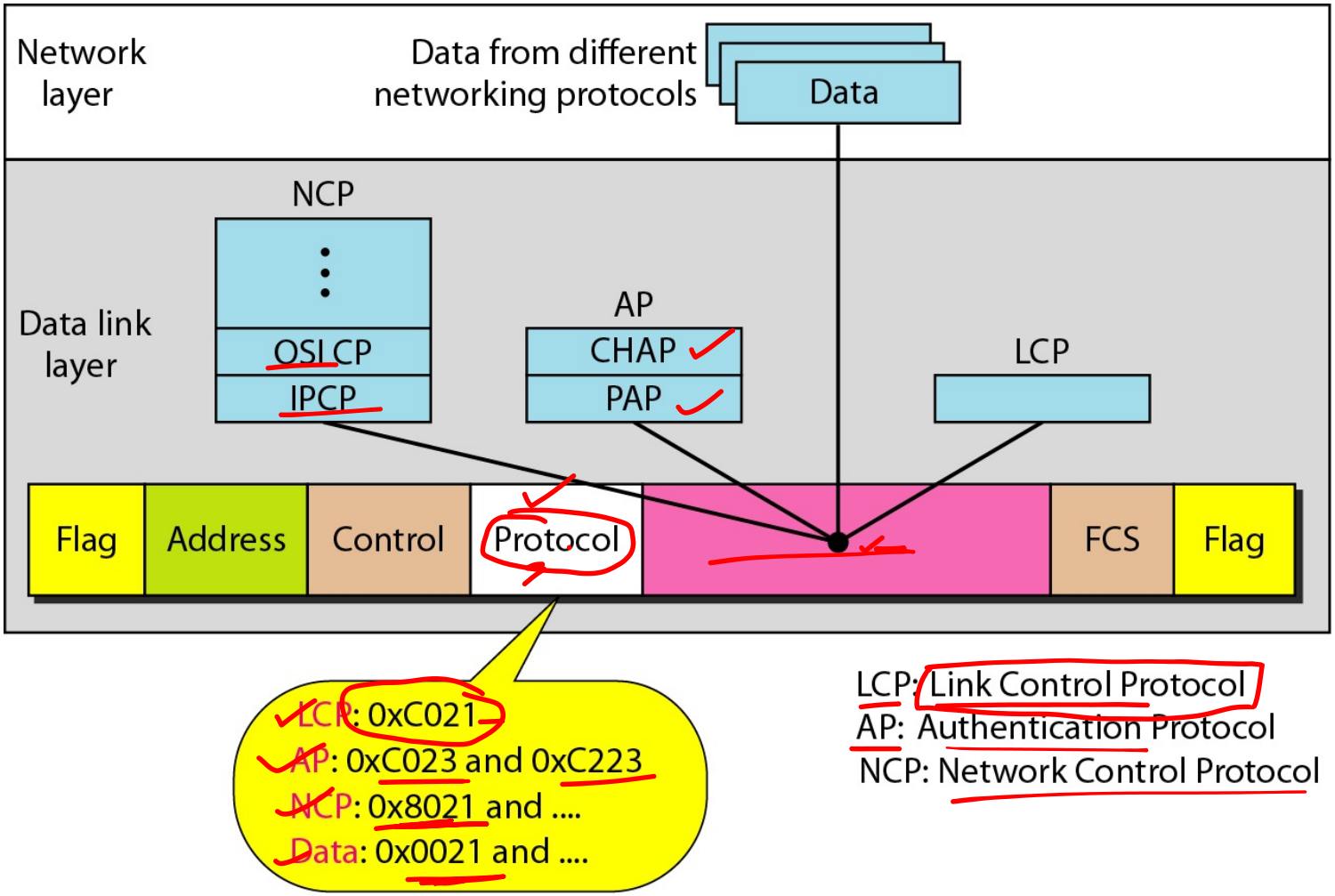
Note

PPP is a byte-oriented protocol using byte stuffing with the escape byte 01111101.

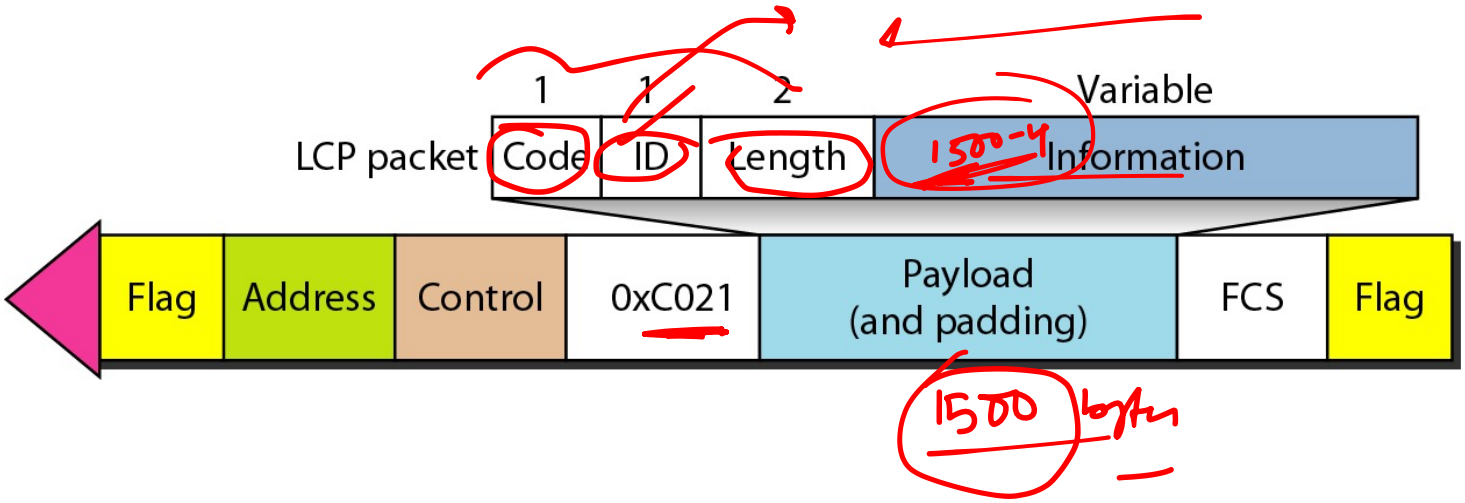
Transition phases



Multiplexing in PPP



LCP packet encapsulated in a frame



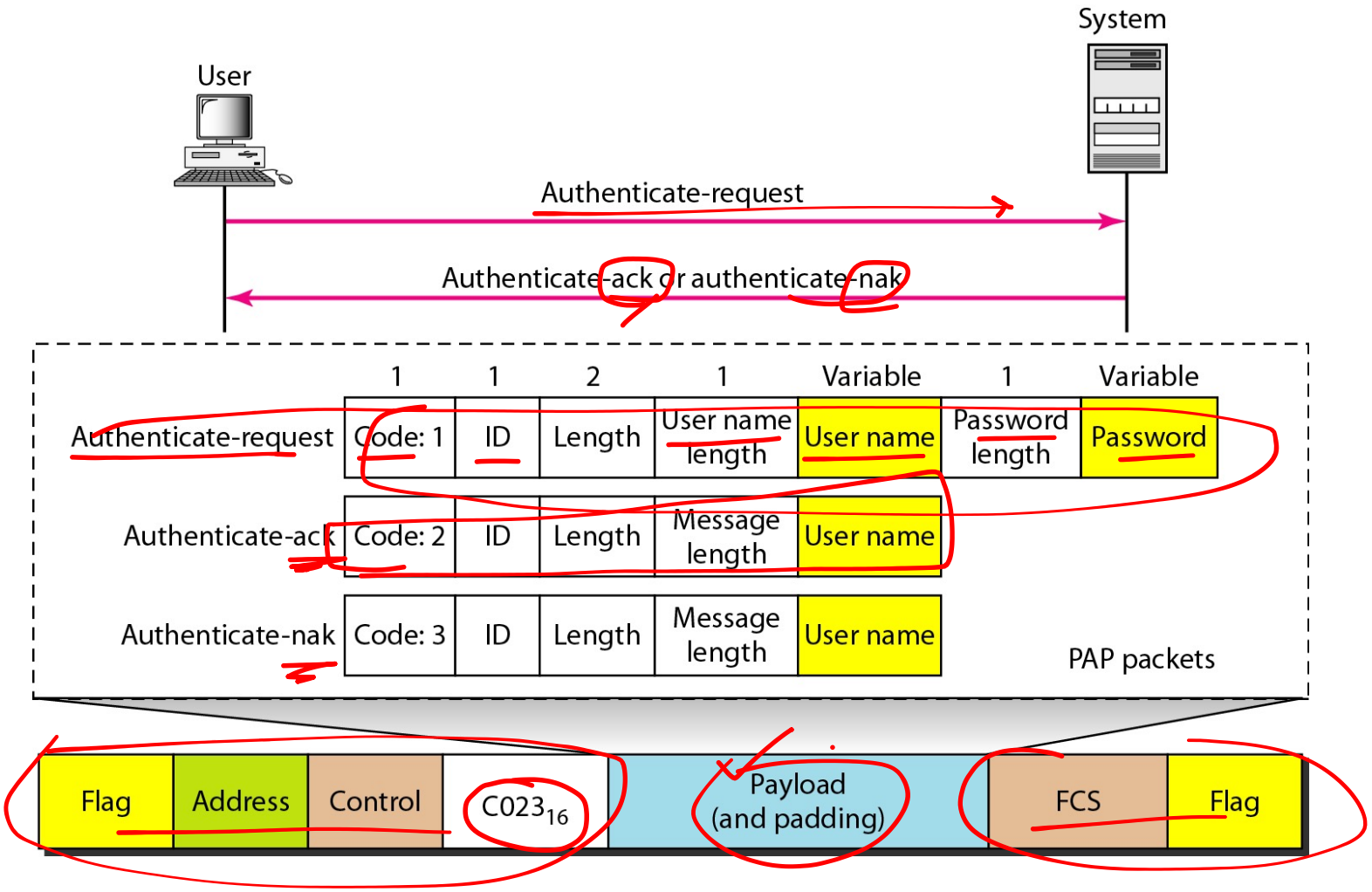
LCP packets

Code	*Packet Type*	*Description*
0x01	Configure-request	Contains the list of proposed options and their values
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptable
0x04	Configure-reject	Announces that some options are not recognized
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other end is alive
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet

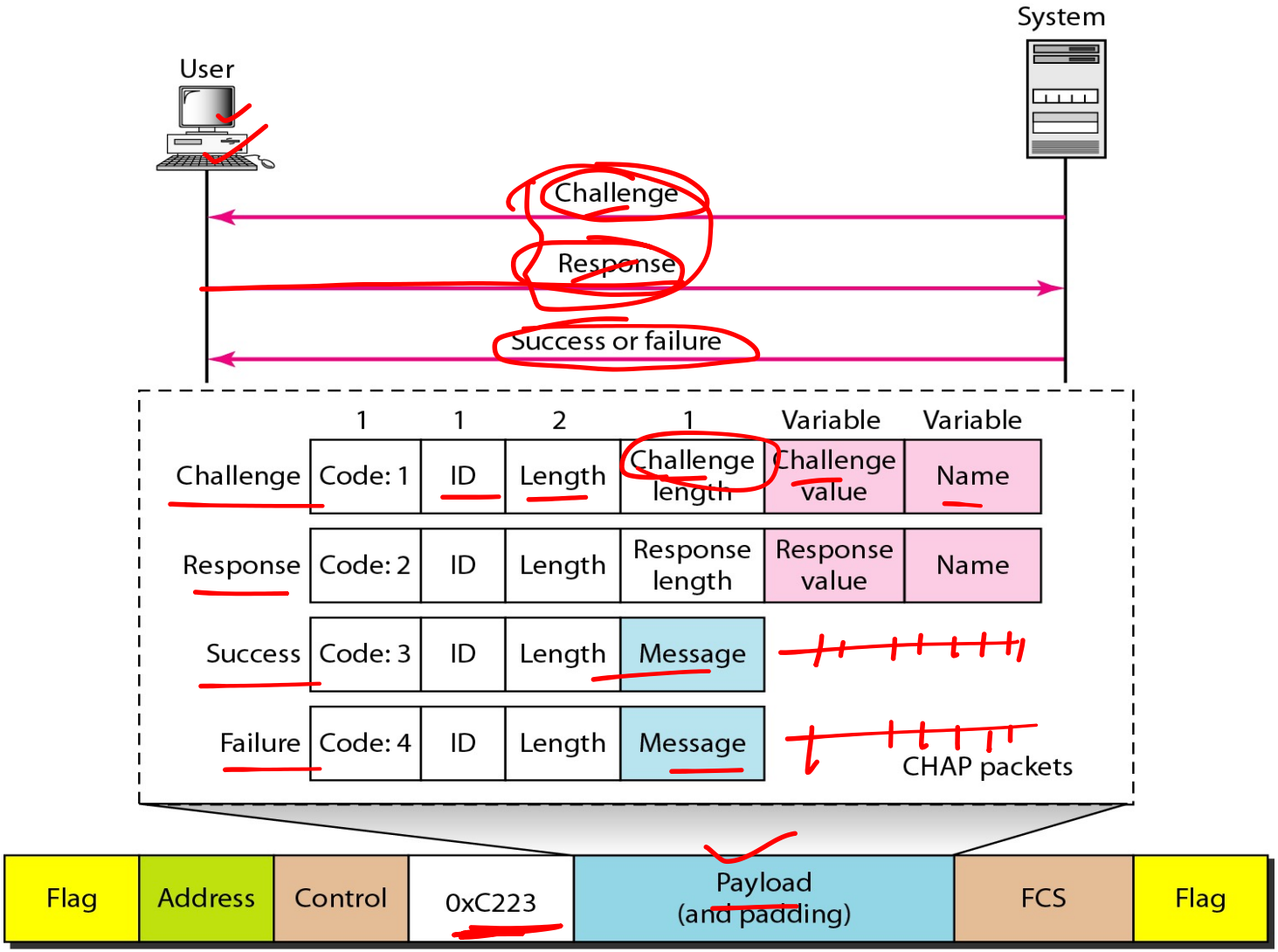
Common options

Option	*Default*
Maximum receive unit (payload field size)	1500
Authentication protocol	None
Protocol field compression	Off
Address and control field compression	Off

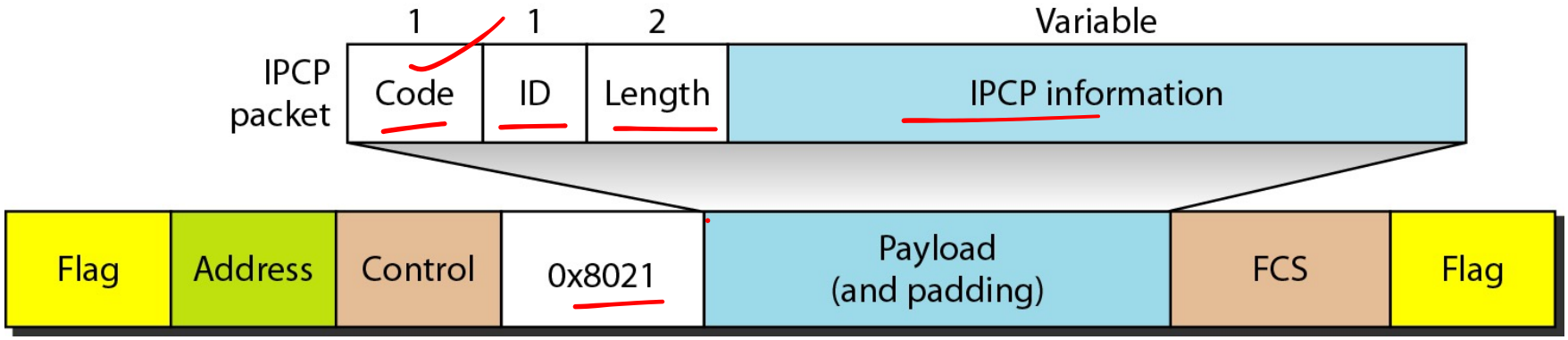
PAP packets encapsulated in a PPP frame



Challenge Handshake Authentication protocol (CHAP) packets encapsulated in a PPP frame



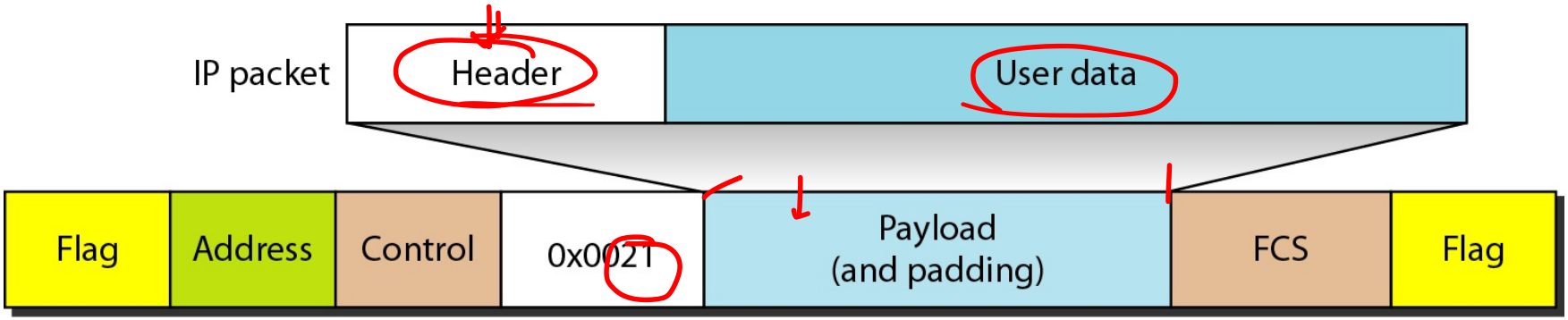
Internet Protocol Control protocol(IPCP) packet encapsulated in PPP frame



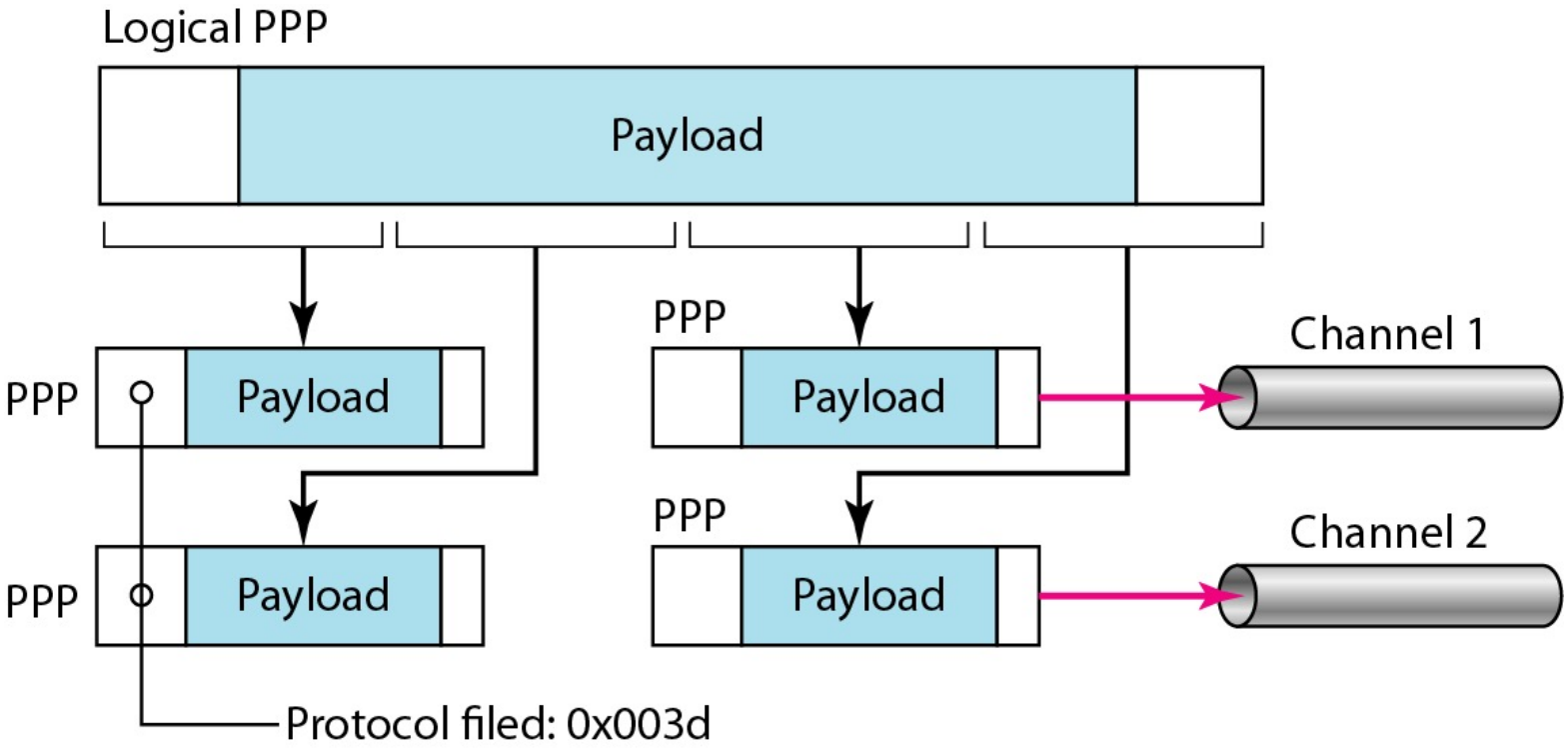
Code value for IPCP packets

Code	*IPCP Packet*
0x01	Configure-request
0x02	Configure-ack
0x03	Configure-nak
0x04	Configure-reject
0x05	Terminate-request
0x06	Terminate-ack
0x07	Code-reject

IP datagram encapsulated in a PPP frame



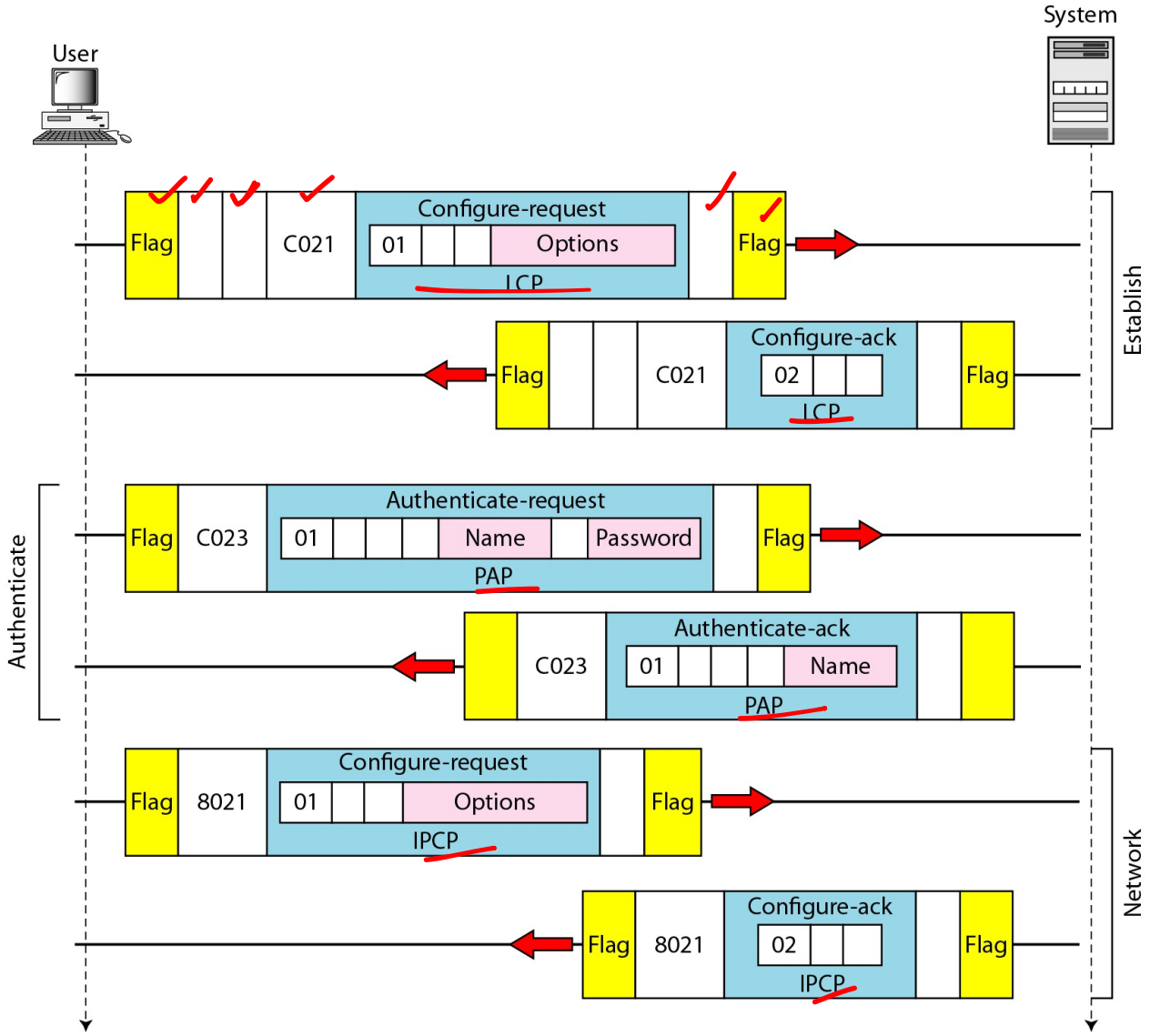
Multilink PPP



An example

- Shows the phases followed by a network layer packet as it is transmitted though a PPP connection
- We assume unidirectional movement of data from the user site
  - Ex: Sending an email through an ISP

An example:


System
User
Flag
C021
Configure-request
01
Options
LCP
Flag
Flag
C021
Configure-ack
02
LCP
Flag
Establish
Authenticate
Flag
C023
Authenticate-request
01
Name
Password
PAP
Flag
C023
Authenticate-ack
01
Name
PAP
Flag
8021
Configure-request
01
Options
IPCP
Flag
Flag
8021
Configure-ack
02
IPCP
Flag
Network

An example (continued)