Chapter 03
Data Link Layer

Associate Prof. Hong Zheng （郑宏）
Computer School
Beijing Institute of Technology

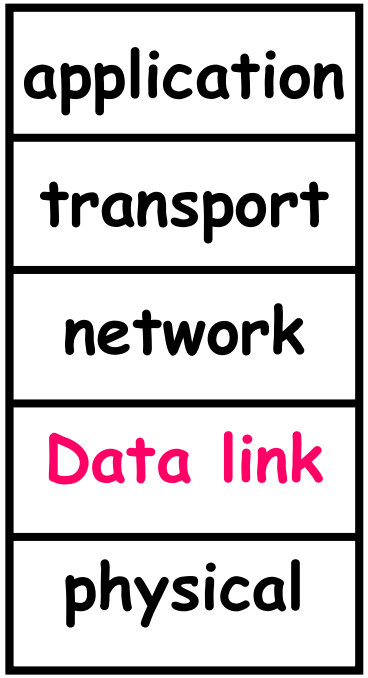
Key Points

功能及机制	功能	掌握
	组帧	熟练掌握
	差错控制	熟练掌握
流量控制与可靠传输	停等协议	熟练掌握
	滑动窗口协议	
协议	HDLC	理解
	PPP	掌握

Chapter 3: Roadmap

- 3.1 Introduction and services
- 3.2 Framing
- 3.3 Error Detection and Correction
- 3.4 Stop-and-Wait Protocols
- 3.5 Sliding Window Protocols
- 3.6 HDLC and PPP

Introduction

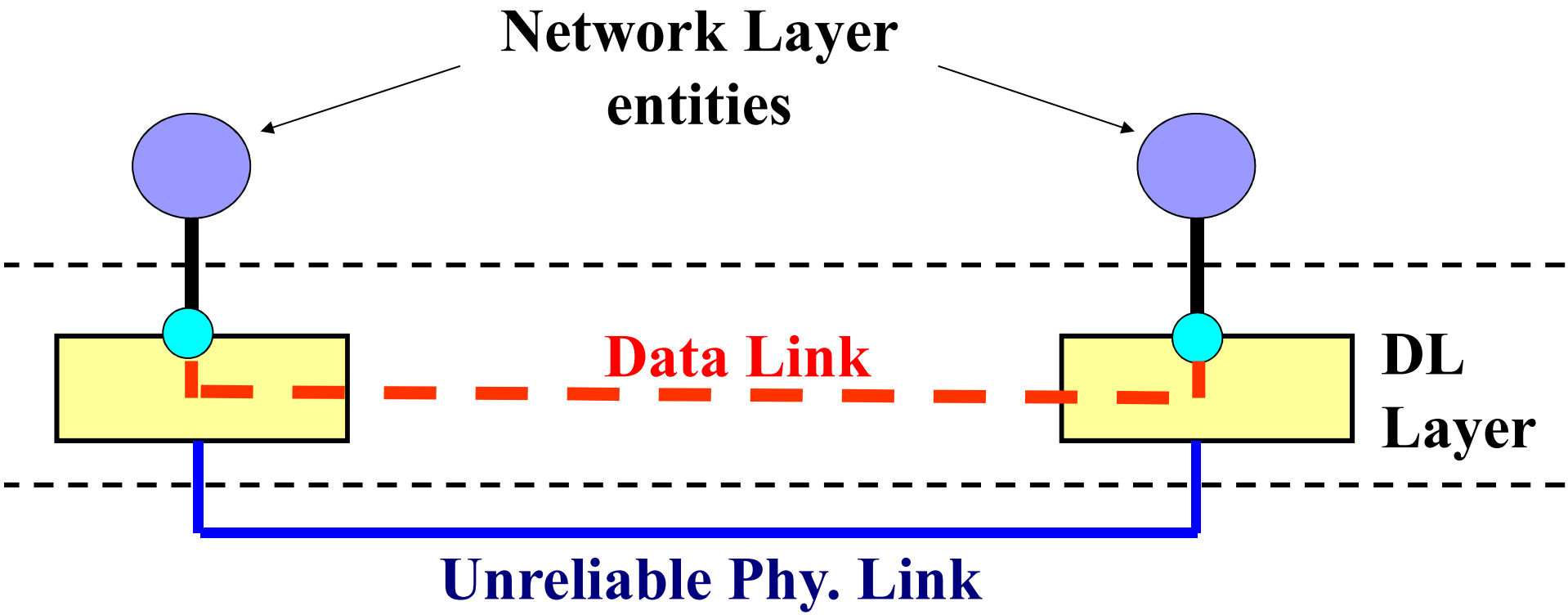


Services:
Deliver a data link frame between two physically connected (adjacent) machines

Functions:

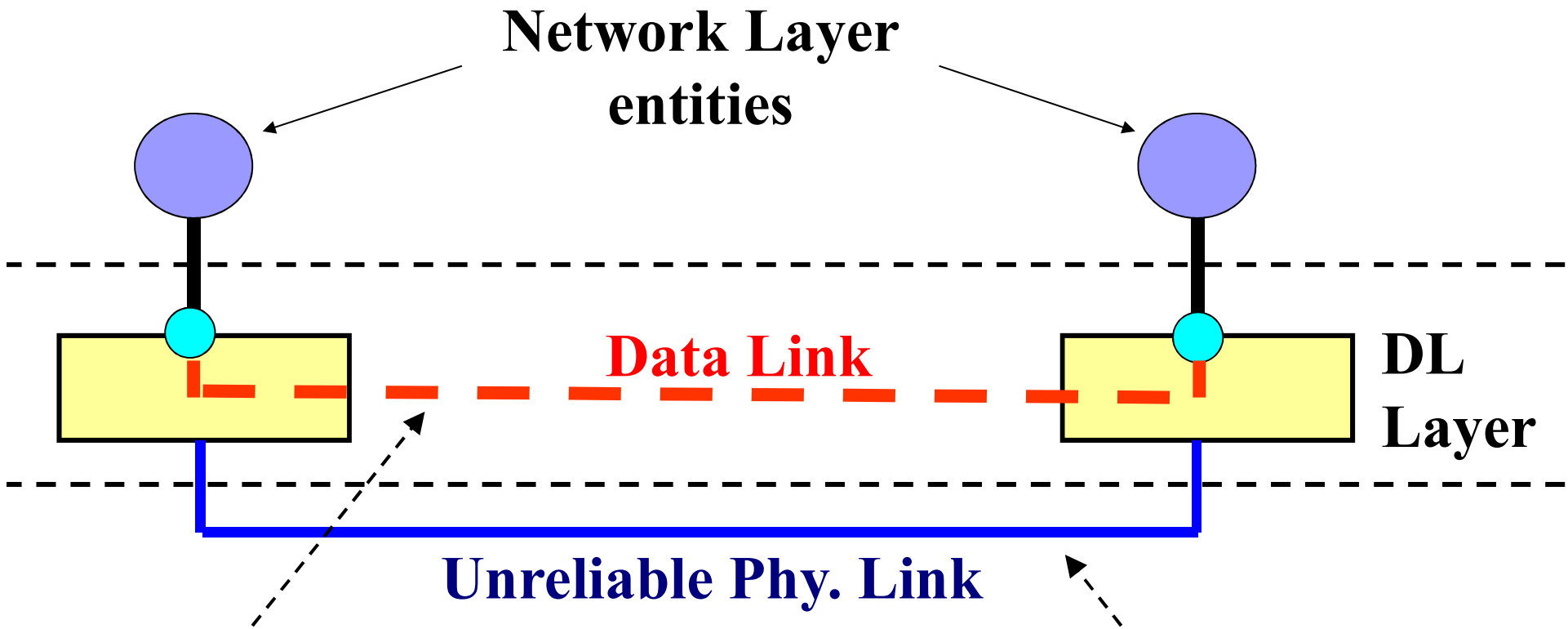
- Providing a well-defined service interface to the network layer.
- Dealing with transmission errors.
- Regulating data flow so that slow receivers are not swamped by fast senders.

Data Link Model



Data Link Model

Data Link Model

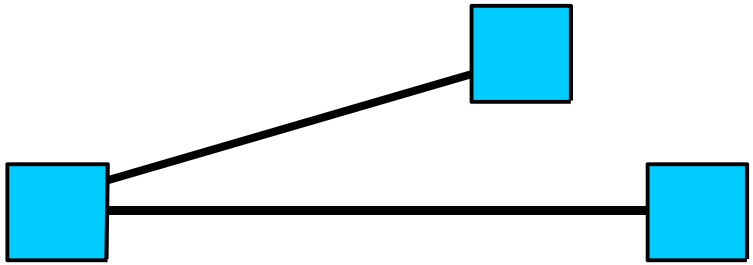


Data Link: make two devices communicate with each other and transfer data.

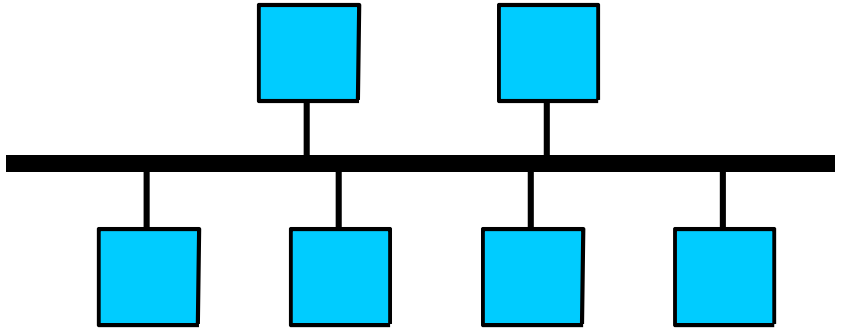
Phy. Link: physical medium connecting two or more devices.

Link Types

- 2 link types:
  - point-to-point
  - Multi-point (broadcast)



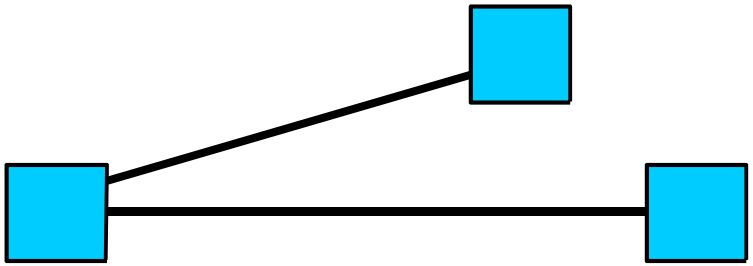
point-to-point link



multipoint link

Point-to-Point Link

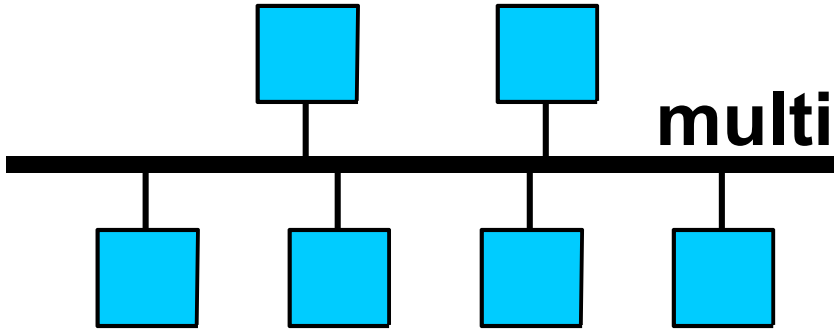
- Two hosts connected directly, one sender, one receiver.



  - No issues of contention, routing, …
  - Framing: recognizing bits on the wire as frames
  - Error detection and correction
  - Flow control

Multi-point (Broadcast) Link

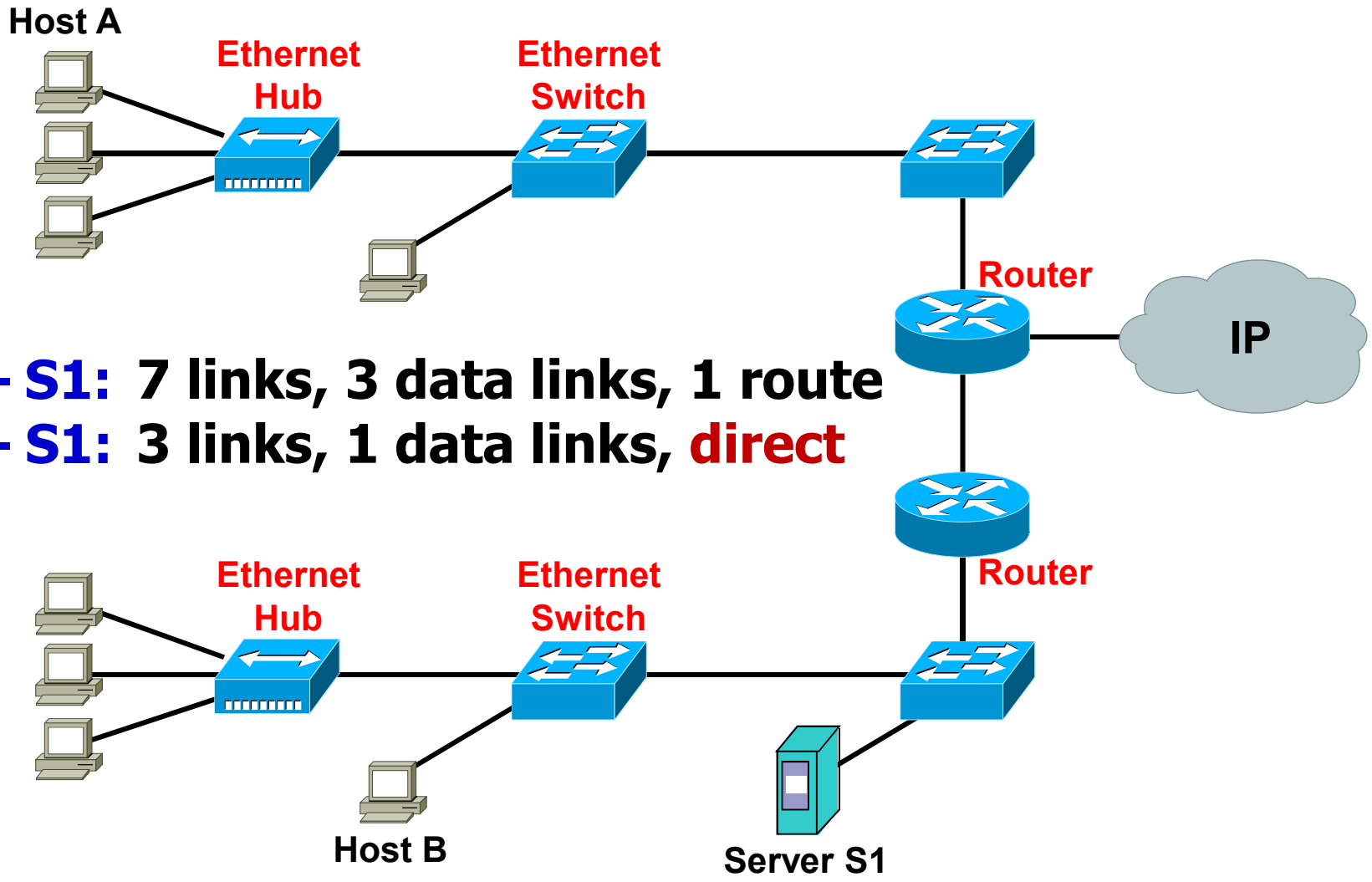
- Multiple hosts sharing the same medium, many sender, many receiver.



What are the new problems?

- Framing
- Error detection and correction
- Flow control
- Addressing
- Media access control

How many links, data links and routes?



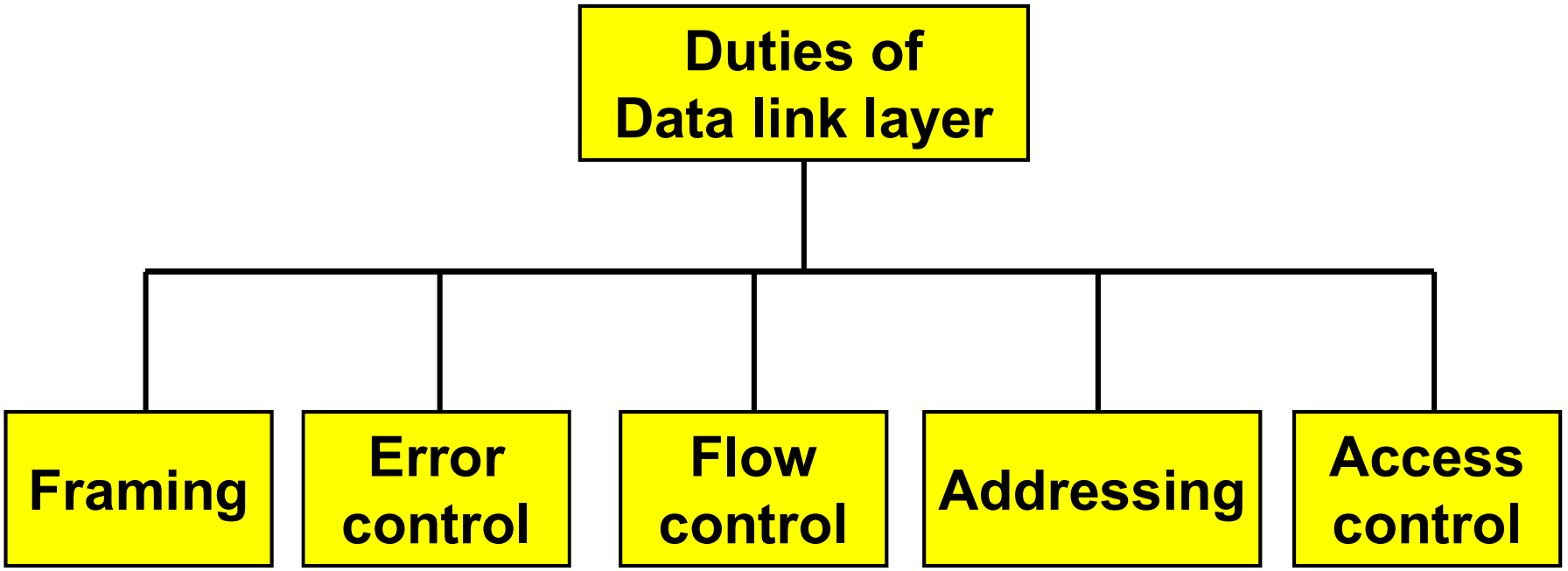
A – S1: 7 links, 3 data links, 1 route

B – S1: 3 links, 1 data links, direct

Services Provided to the Network Layer

- **Unacknowledged connectionless service**
  - **No connection is established beforehand or released afterward.**
  - **Source machine sends independent frames to the destination without having the destination acknowledge them.**
  - **No attempt is made to recover lost data.**
- **Acknowledged connectionless service**
  - **No logical connection used.**
  - **Each frame sent is individually acknowledged. If a frame has not arrived within a specified time interval, it can be sent again.**
- **Acknowledged connection-oriented service**
  - **Transfers go through three distinct phases, each frame sent over the connection is numbered and indeed received.**

Data link layer duties

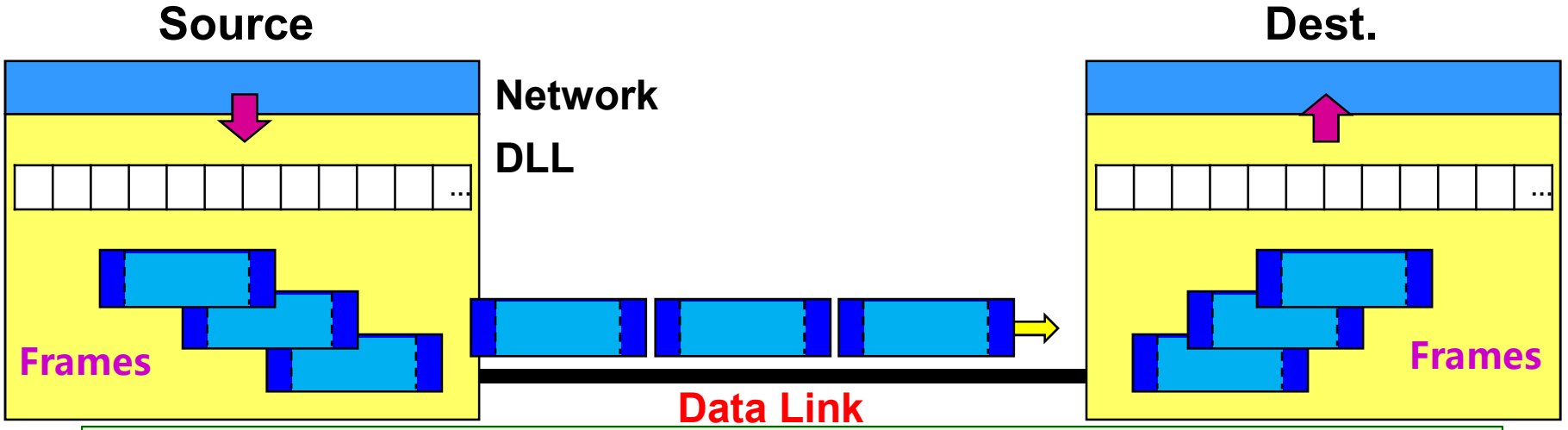


Chapter 3: Roadmap

- 3.1 Introduction and services
- 3.2 Framing
- 3.3 Error Detection and Correction
- 3.4 Stop-and-Wait Protocols
- 3.5 Sliding Window Protocols
- 3.6 HDLC and PPP

Framing

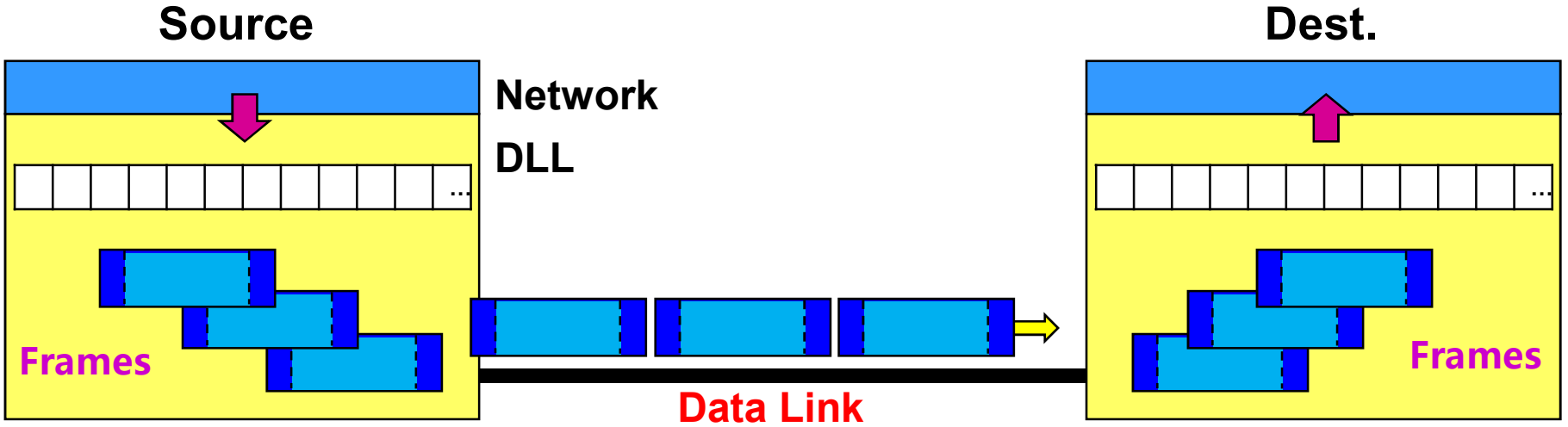
- **Frame:** unit of DLL data transfer.
- Large block of data may be broken up into small frames at the source. **(WHY?)**



- Limit buffer size at the receiver.
- Detect errors sooner.
- Retransmit small amount of data.
- Prevent from occupying medium for long periods.

Framing

- **Frame:** unit of DLL data transfer.
- Large block of data may be broken up into small frames at the source. **(WHY?)**

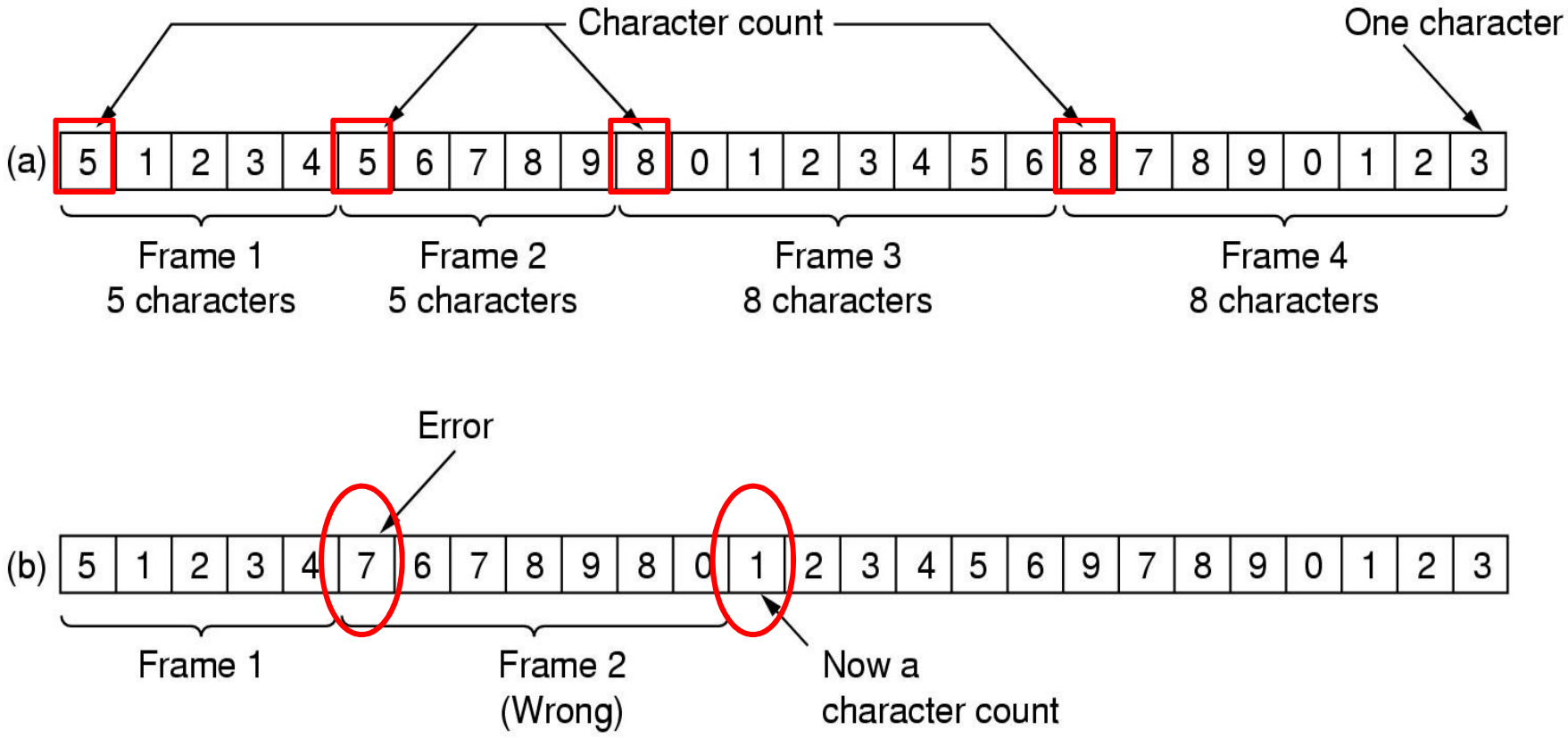


Need to indicate the start and end of a block of data - Frame synchronization.

Framing

- **How can one determine the beginning/end of a frame?**
- **Solutions:**
  - Character count
  - Flag bytes with byte or character stuffing
  - Starting and ending flags with bit stuffing
  - Physical layer coding violations
  - Clock-Based Framing

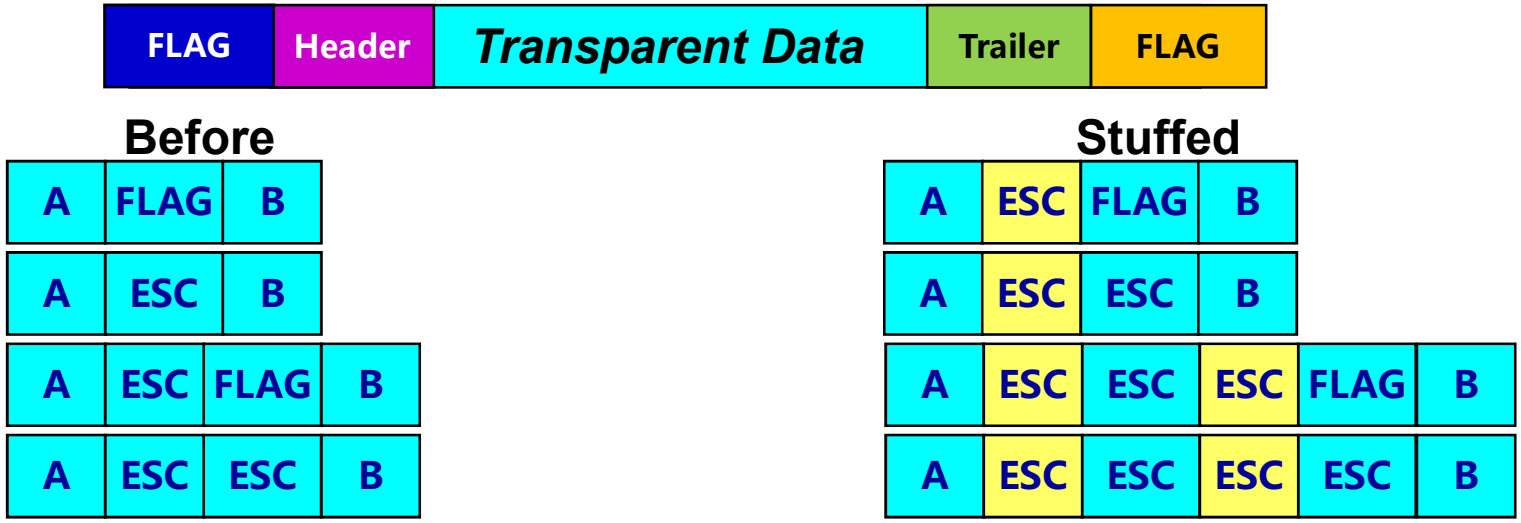
Character Count



A character stream. (a) Without errors. (b) With one error.

Flag byte with byte/Char stuffing

- **Flag byte:** a special byte or character, marking the beginning and end of a frame.
- **byte stuffing:** The **sender** inserts a special escape byte (e.g. ESC) just before each "accidental" flag byte in the data. ⇒ **Transparent Transmission**



The data link layer on the receiving end unstuffs the ESC before giving the data to the network layer.

Starting and ending flags with bit stuffing

- **Flag:** a special bit sequence (e.g. 01111110), marking the beginning and end of a frame.
- **bit stuffing:** the sender automatically stuffs a 0 bit into the outgoing stream whenever encountering five consecutive '1' in the data stream.

Data Stream

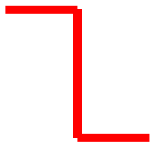
0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0

0 1 0 1 1 1 1 1 **0** 0 1 1 1 1 1 **0** 1 0 1 1 1 1 1 **0** 1 1 1 1 1 **0** 1 1 0 1 0

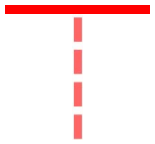
When the receiver sees 5 consecutive incoming ones followed by a 0 bit, it automatically destuffs the 0 bit before sending the data to the network layer.

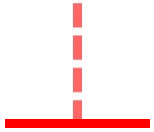
Physical layer coding violations

- **Manchester encoding:**


= “1”


= “0”


= start flag

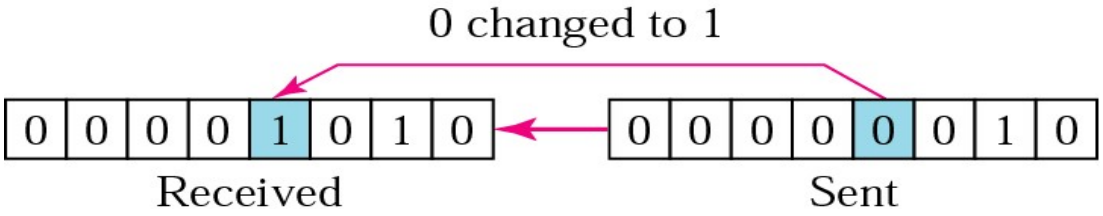

= end flag

Chapter 3: Roadmap

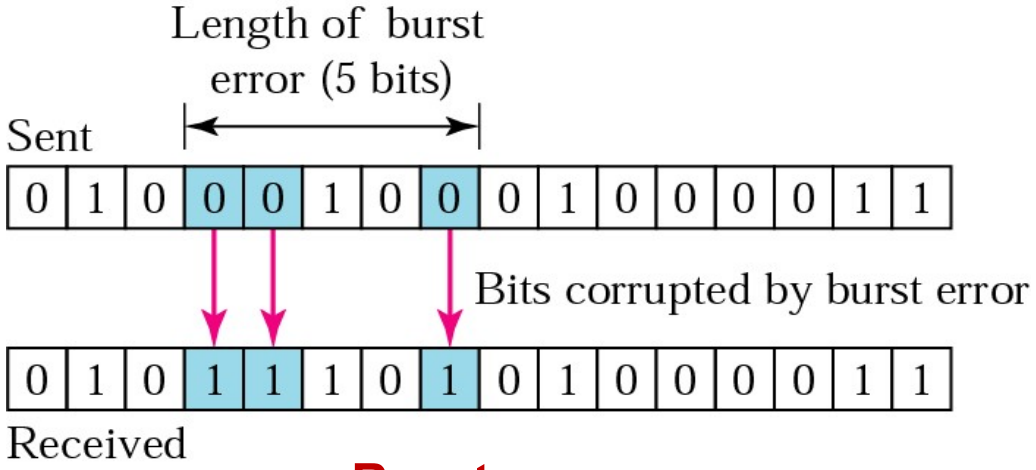
- 3.1 Introduction and services
- 3.2 Framing
- 3.3 Error Detection and Correction
- 3.4 Stop-and-Wait Protocols
- 3.5 Sliding Window Protocols
- 3.6 HDLC and PPP

Error Detection and Correction

- **An error occurs when a bit is altered between transmission and reception**



Single-Bit Error



Burst error

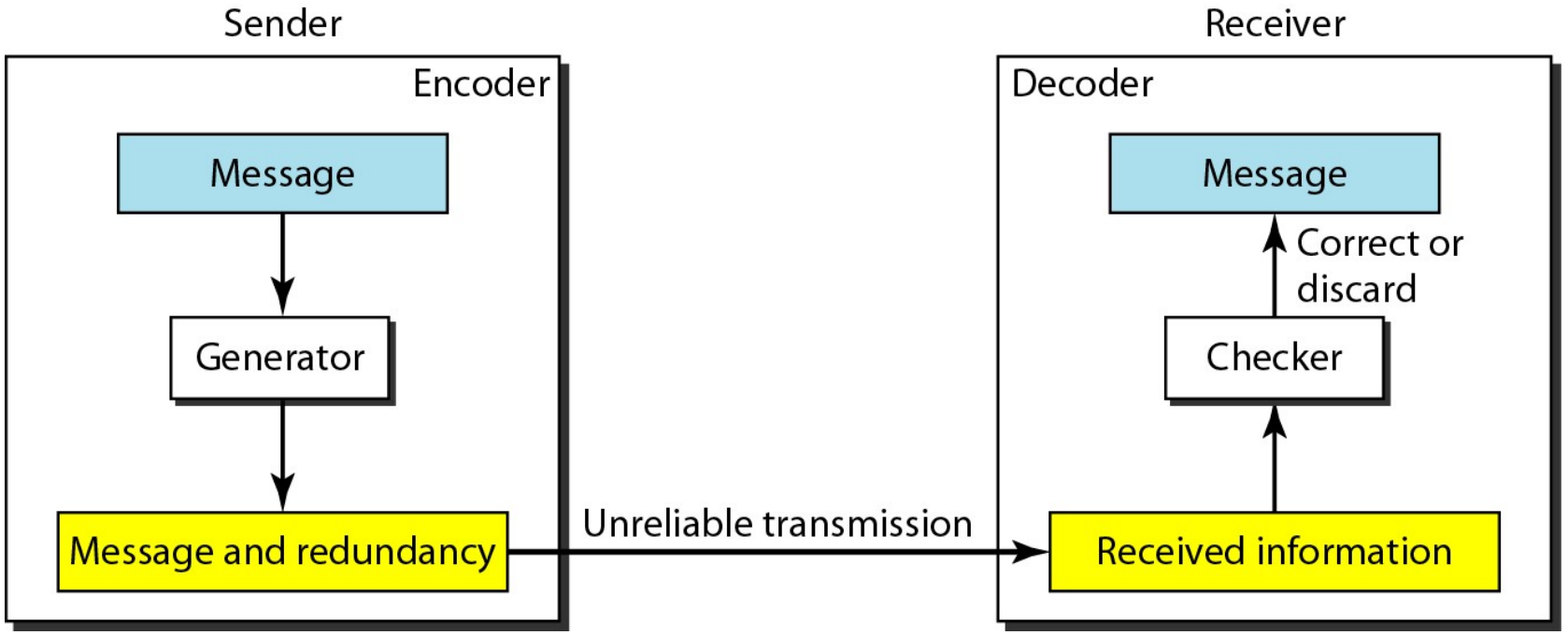
does NOT necessarily mean that the errors occur in consecutive bits

Error Detection and Correction

- **Problem:** How do we actually *notice* that something's wrong, and can it be corrected *by the receiver*?

To detect or correct errors, we need to send extra (redundant) bits with data.

Error Detection and Correction



- **Two general approaches:**
  - **error-correcting codes (forward error correction).**
  - **error-detecting codes + an error correction mechanism when errors are detected.**

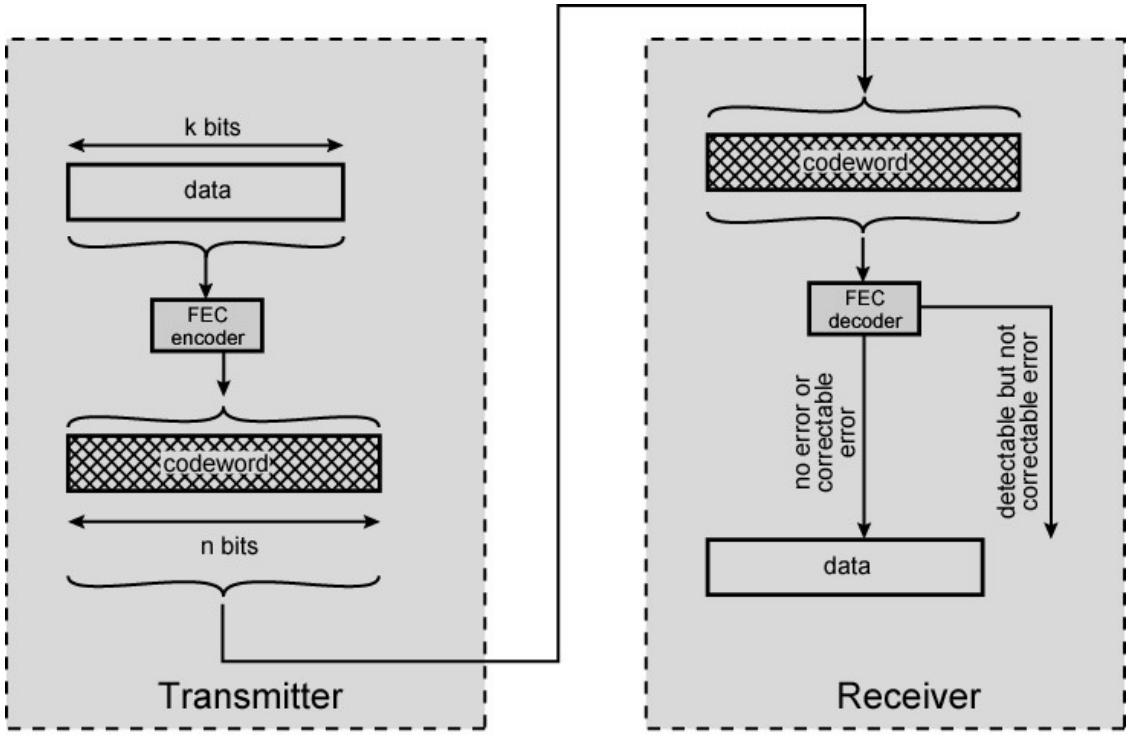
Error Correction

- To enable the receiver to correct errors in an incoming transmission on the basis of the bits in that transmission.
- FEC: forward error correction

Error-Correcting Codes

- A frame consists of m data (i.e., message) bits and r redundant, or check, bits.
- Let the total length be n (i.e., $n = m + r$). An n-bit unit containing data and check bits is often referred to as an **n-bit codeword**.

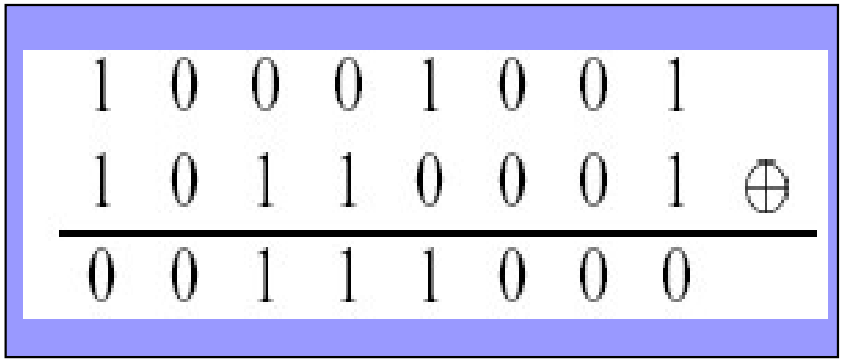
Error Correction Process Diagram



- A frame consists of *m* data (i.e., message) bits and *r* redundant, or check, bits.
- Let the total length be *n* (i.e., *n* = *m* + *r*). An *n*-bit unit containing data and check bits is often referred to as an *n*-bit codeword.

Error Correction: Hamming Code

- **Hamming distance :** The hamming distance between two bit string *a* and *b* is the number of bits at the same position that differ.



Error Correction: Hamming Code

- To *detect* all sets of d or fewer errors, it is necessary and sufficient that the Hamming distance between any two frames is $d+1$ or more.
- To *correct* all sets of d or fewer errors, it is necessary that the Hamming distance between any two frames is $2d+1$ or more.

Error Correction: Hamming Code

- Every bit at position 2^k, k>=0 is used as a parity bit for those positions to which it contributes
- If a check bit at position p is wrong upon receipt, the receiver increments a counter v with p; the value of v will, in the end, give the position of the wrong bit, which should then be swapped.

	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11
1	X		X		X		X		X		X
2		X	X			X	X			X	X
4				X	X	X	X				
8								X	X	X	X
	1	0	1	1	1	0	0	1	0	0	1

Error Correction: Hamming Code

Example: "H" = 1001000, even parity check,

2^0: 3, 5, 7, 9, 11; 2^1: 3, 6, 7, 10, 11;

2^2: 5, 6, 7; 2^3: 9, 10, 11;

2^0	2^1		2^2				2^3			
1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	0	0	1	0	0	0	0

if received:

0	0	1	1	0	0	0	0	0	0	0

Σ = 1 + 2 + 4 = 7, bit 7 is in error, change to 1;

if received:

0	0	1	0	1	0	0	1	0	0	1

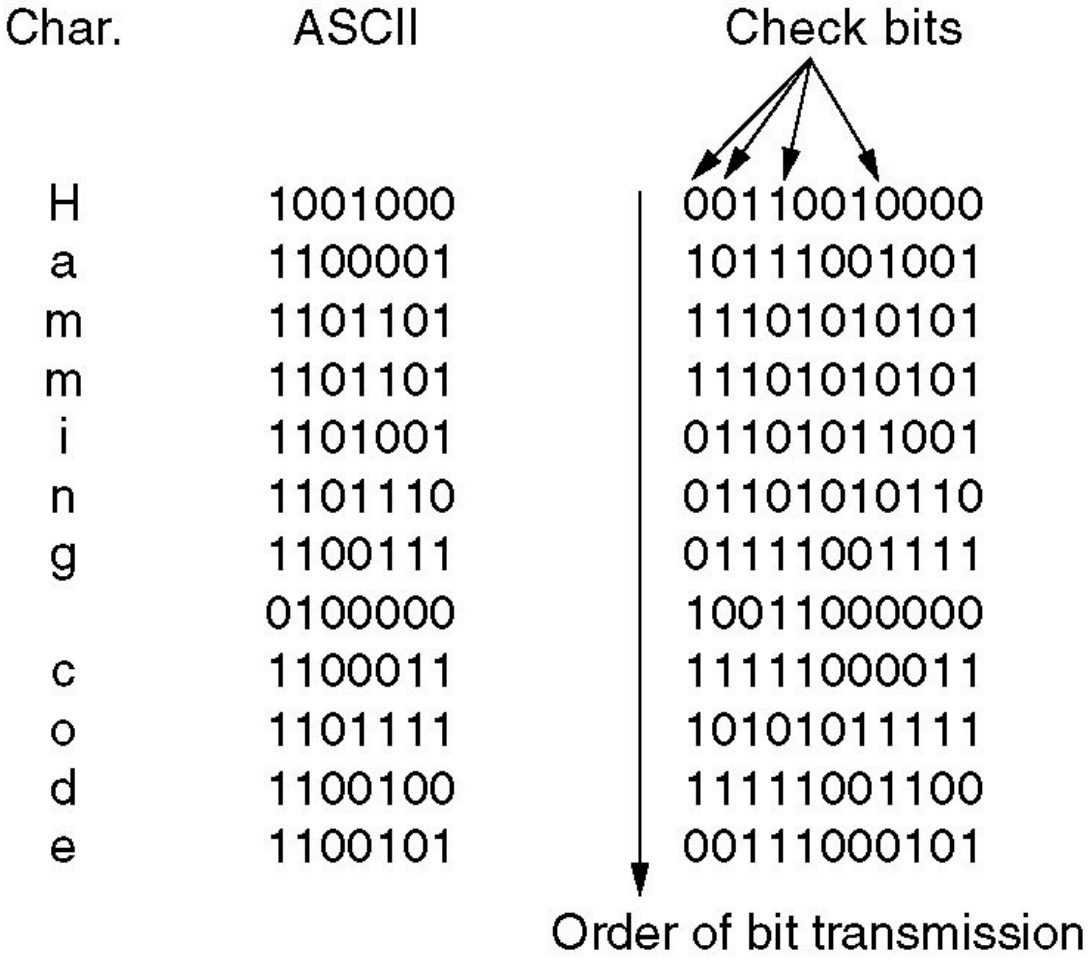
then

Σ = 1 + 4 = 5, bit 5 is in error, change to 0. ???

Error Correction: Hamming Code

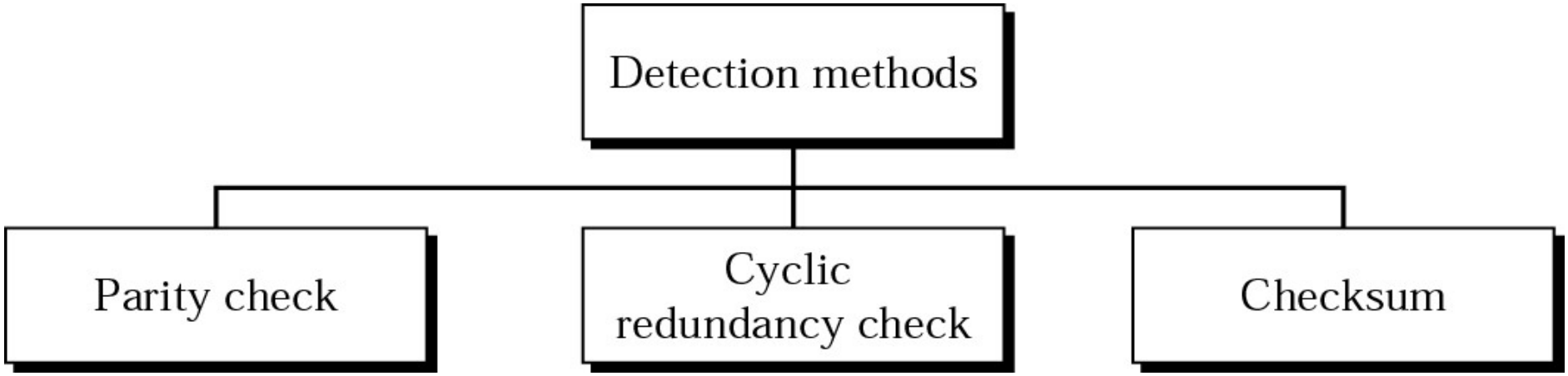
- Imagine that we want to design a code with m message bits and r check bits that will allow all single errors to be corrected.
- Since the total number of bit patterns is 2^n, we must have $(n+1)2^m <= 2^n$. Using $n=m+r$, this requirement becomes $(m+r+1) <= 2^r$.

Use of a Hamming code to correct burst errors



A sequence of k consecutive codewords are arranged as a matrix, one codeword per row. Normally the data would be transmitted one codeword at a time, from left to right.
This method uses *kr* check bits to make blocks of *km* data bits immune to a single burst error of length *k* or less.

Error-Detecting Codes



Common methods:

- **Cyclic redundancy check (CRC)**
- **Checksum**

Parity Check: Single

Info Bits: $b_1, b_2, b_3, \ldots, b_k$

Check Bit: $b_{k+1} = b_1 + b_2 + b_3 + \ldots + b_k$ modulo 2

Codeword: $[b_1, b_2, b_3, \ldots, b_k, b_{k+1}]$

Minimum Hamming Distance d_{min} = ?

for ALL parity check codes, d_{min} = 2

In parity check, a parity bit is added to every data unit so that the total number of 1s is even (or odd for odd-parity).

Parity Check: Single

- **Receiver**
  - **CAN DETECT** all single-bit errors & burst errors with odd number of corrupted bits
  - single-bit errors **CANNOT** be **CORRECTED** – position of corrupted bit remains unknown
  - all even-number burst errors are **UNDETECTABLE** !!!

Parity Check: Single

- **Effectiveness**

$$P(\textit{error detection failure}) = \binom{n}{2} p_b^2 (1-p_b)^{n-2} + \binom{n}{4} p_b^4 (1-p_b)^{n-4} + \binom{n}{6} p_b^6 (1-p_b)^{n-6} + \ldots$$

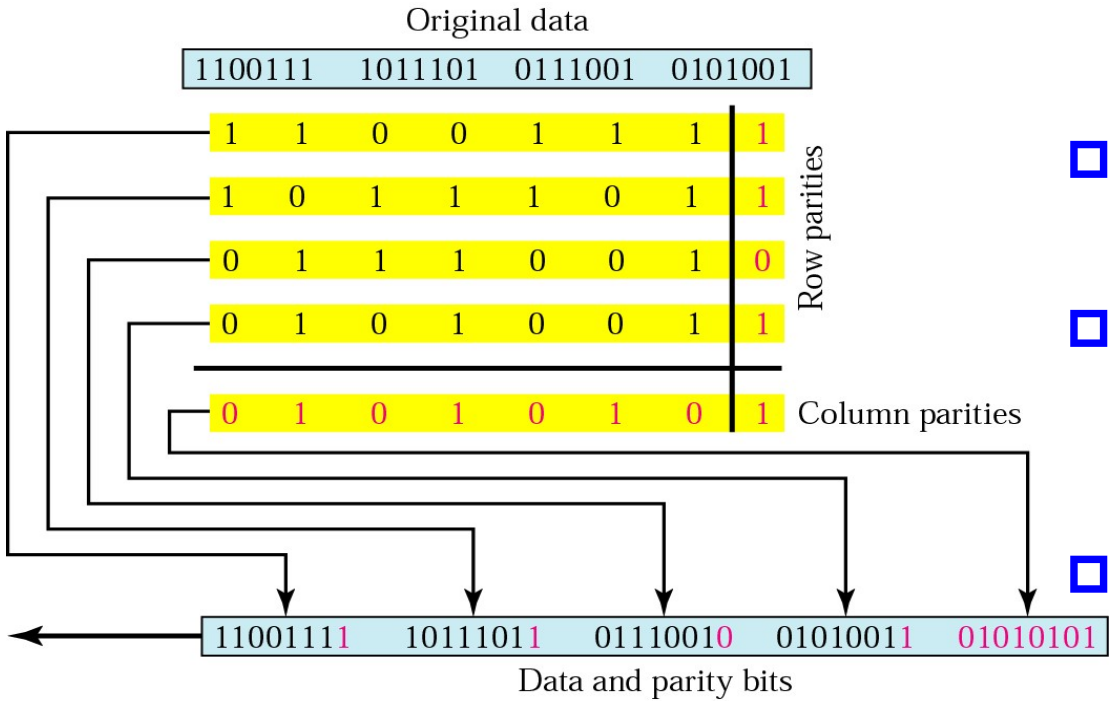
Example: Assume there are n = 32 bits in a codeword (packet). Probability of error in a single bit transmission $p_b = 10^{-3}$. Find the probability of error-detection failure.

$$P(\textit{error detection failure}) = \binom{32}{2} p_b^2 (1-p_b)^{30} + \binom{32}{4} p_b^4 (1-p_b)^{28} + \binom{32}{6} p_b^6 (1-p_b)^{26} + \ldots$$

$$P(\text{error detection failure}) = 496 * 10^{-6} = 4.96 * 10^{-4} \approx \frac{1}{2000}$$

Parity Check: 2-Dimension

- **Increasing the likelihood of detecting burst errors.**



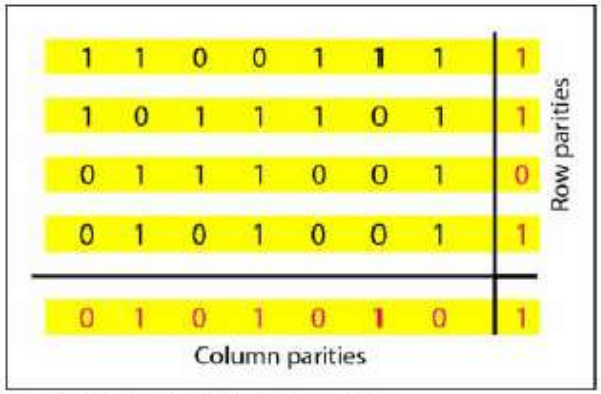
- **all 1-bit errors CAN BE DETECTED and CORRECTED.**
- **all 2-, 3- bit errors can be DETECTED.**
- **4- and more bit errors can be detected in some cases.**
- **Drawback: too many check bits !!!**

Parity Check: 2-Dimension

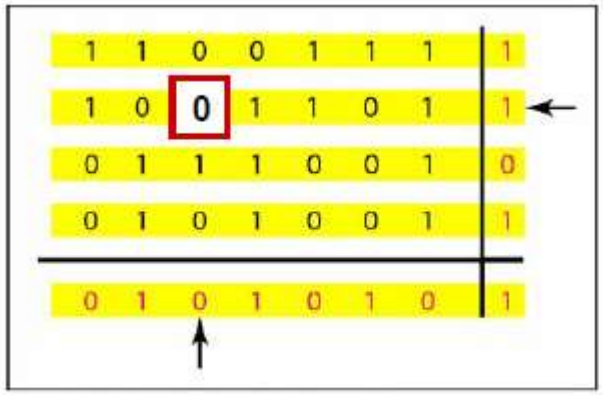
In two-dimensional parity check, a block of bits is divided into rows and a redundant row of bits is added to the whole block.

Parity Check: 2-Dimension

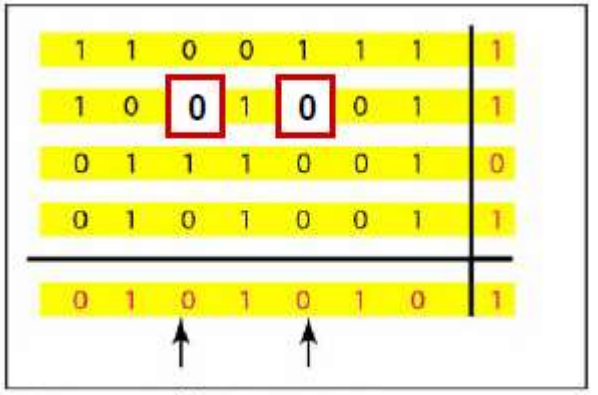
- **Effectiveness**



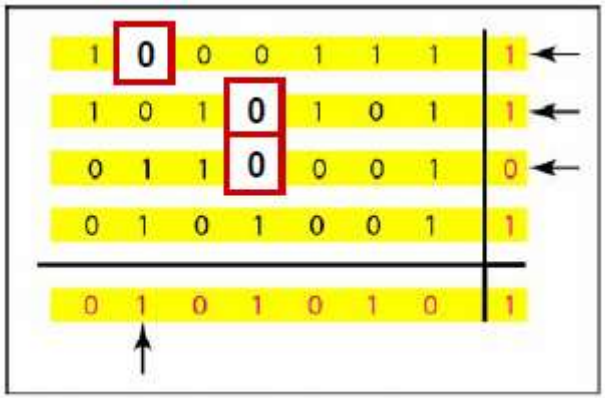
a. Design of row and column parities



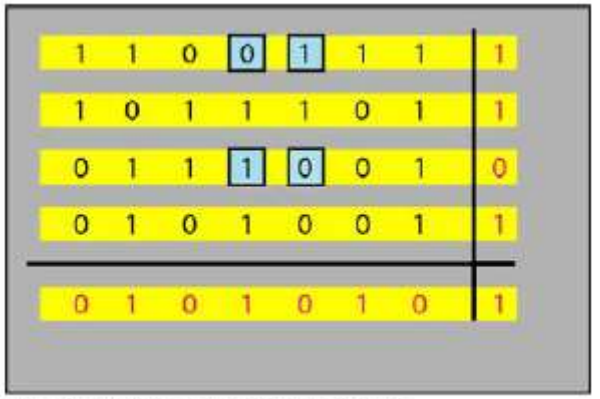
b. One error affects two parities



c. Two errors affect two parities



d. Three errors affect four parities



e. Four errors cannot be detected

Parity Check: 2-Dimension

Example: Suppose the following block of data, error-protected with 2-D parity check, is sent:

10101001 00111001 11011101 11100111 10101010

However, the block is hit by a burst noise of length 8, and some bits are corrupted.

10100011 10001001 11011101 11100111 10101010

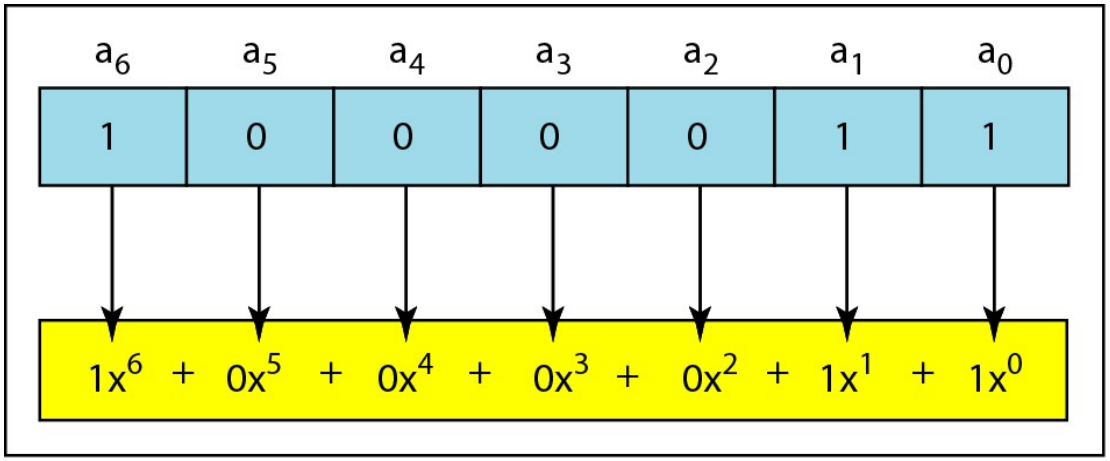
Will the receiver be able to detect the burst error in the sent data?

1010100	1
0011100	1
1101110	1
1110011	1
1010101	0

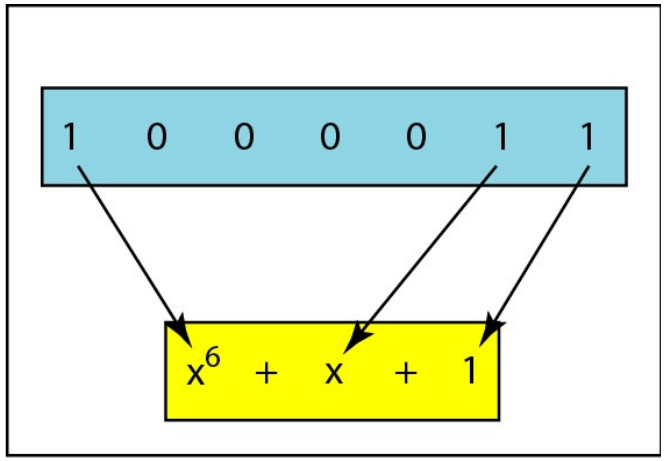
1010001	1
1000100	1
1101110	1
1110011	1
1010101	0

Cyclic Redundancy Check

- A ***k+1*** bit number is represented as a **polynomial** of degree ***k*** in some dummy variable, with leftmost bit being most significant.



a. Binary pattern and polynomial



b. Short form

A polynomial to represent a binary word

Cyclic Redundancy Check

- Sender and receiver agree on the "generator" of the code, another polynomial G(x), with 1s in msb and lsb
- All arithmetic operation is

  EXCLUSIVE OR (XOR)

  Ex: 10011011+11001010=01010001

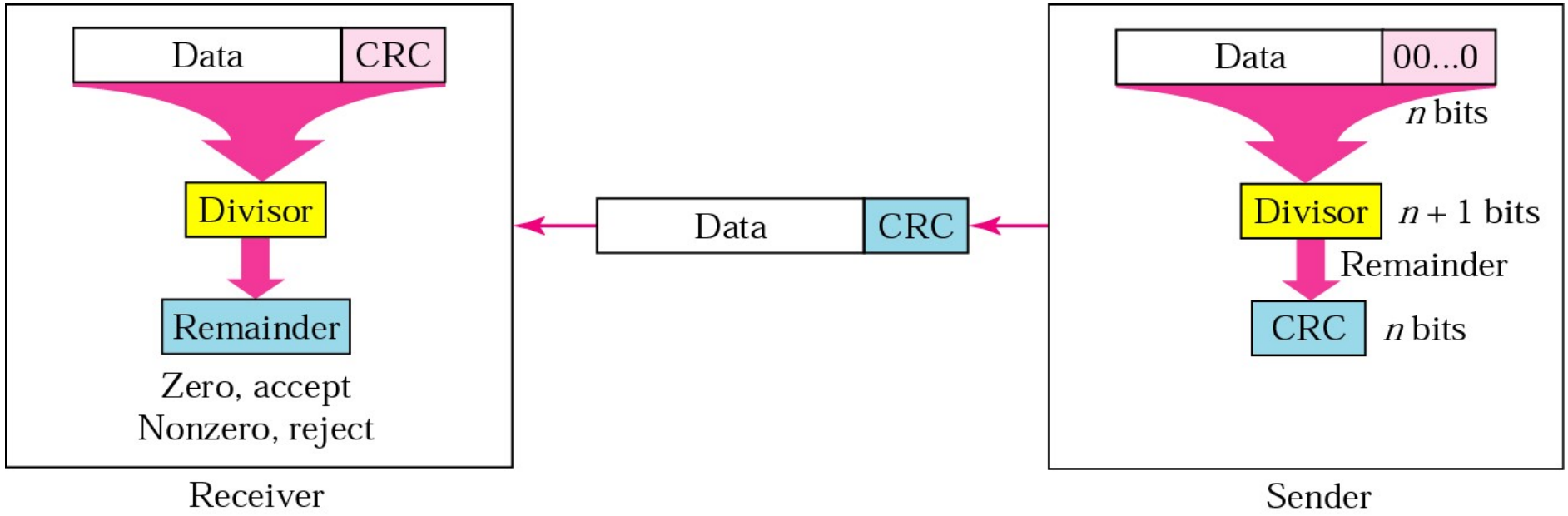
  01010101 -10101111=11111010

  For division, is done at same length.

Cyclic Redundancy Check

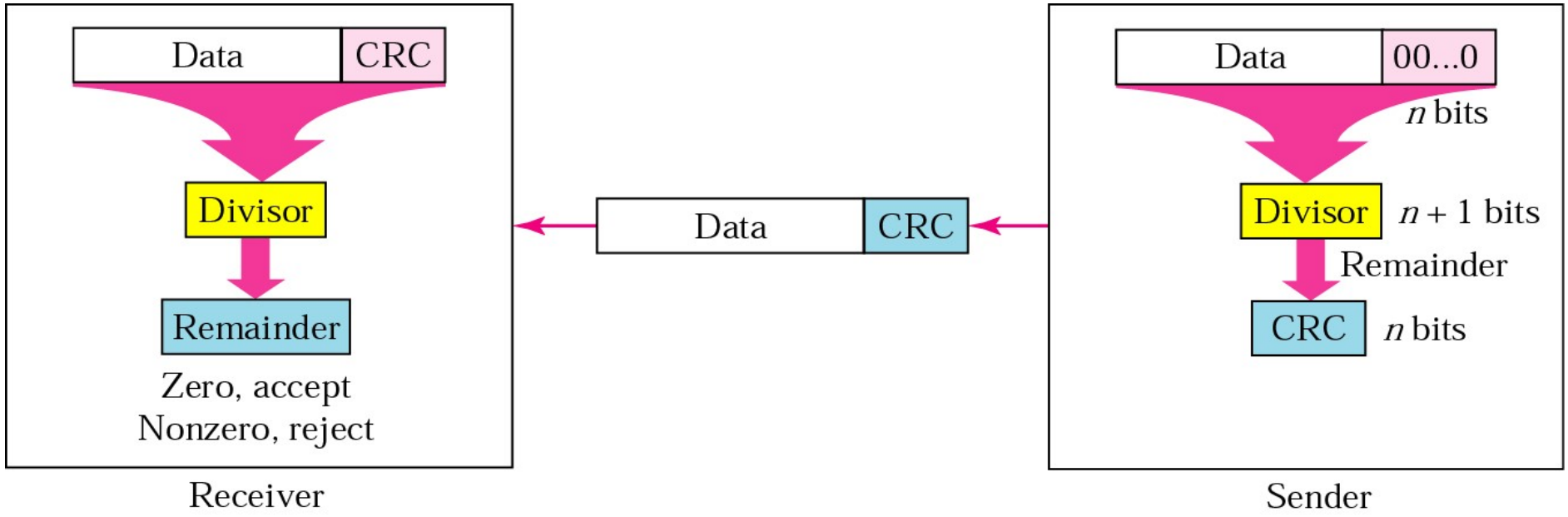
- Append a "CRC" to the end of the msg
  - If G(x) is of degree *r*, then append *r* 0s to end of the m msg bits. The new number is now $x^r M(x)$.
  - Divide this by G(x) modulo 2
  - Subtract the remainder from $x^r M(x)$ modulo 2, The result is the checksummed frame to be transmitted T(x).
  - This number is now divisible by G(x)

Cyclic Redundancy Check



- **Transmit T(x), and have receiver check if the received data is divisible by G(x).**
- **Remainder = 0, no error**
- **Remainder != 0, error detected**

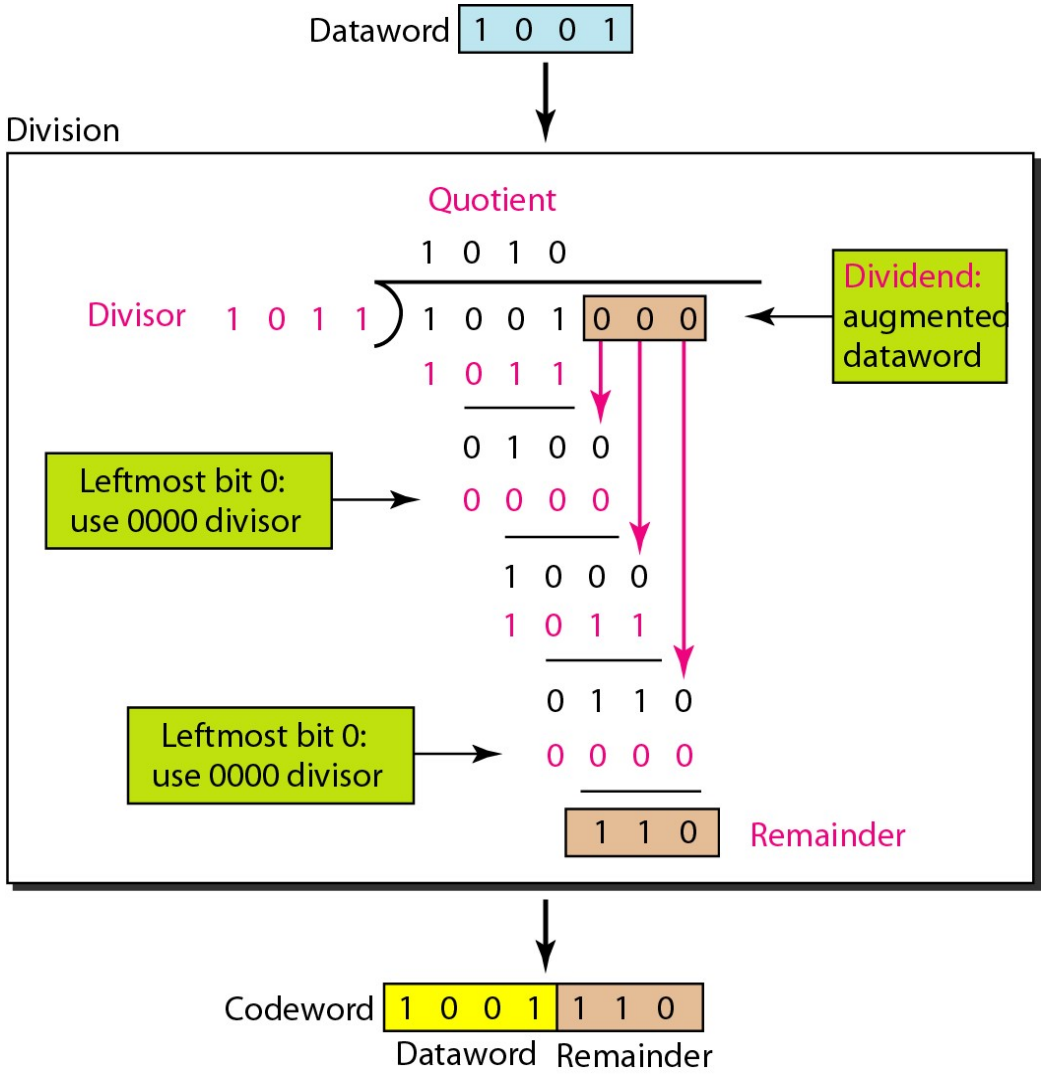
Cyclic Redundancy Check



Remainder = 0 implies no errors

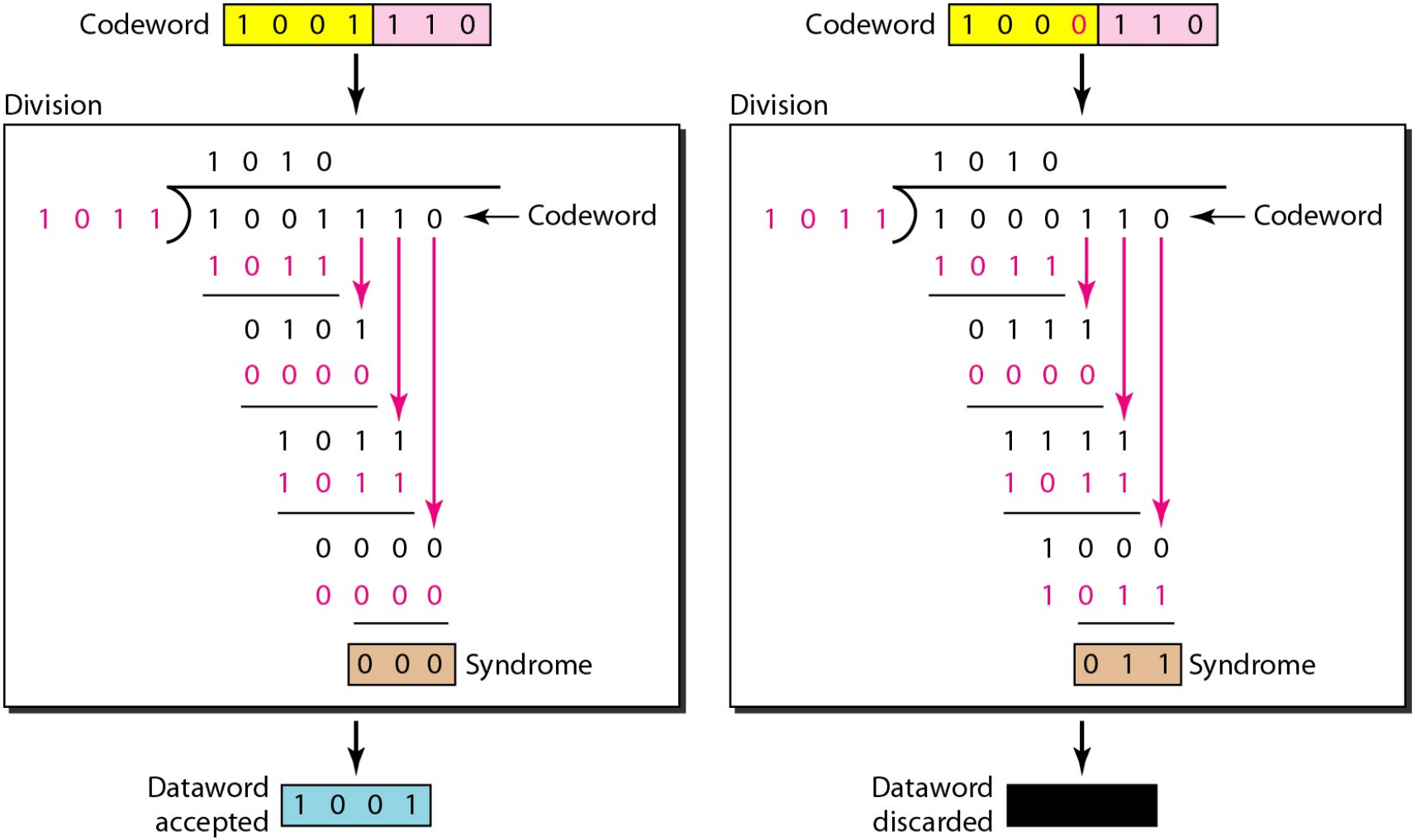
- **No bit is corrupted. OR**
- **Some bits are corrupted, but the decoder failed to detect them.**

Cyclic Redundancy Check



Division in CRC encoder

Cyclic Redundancy Check



Division in the CRC decoder for two cases

Selecting G(x)

- **All single-bit errors,** as long as the x^k and x^0 terms have non-zero coefficients.
- **All double-bit errors,** as long as G(x) contains a factor with at least three terms
- **Any odd number of errors,** as long as G(x) contains the factor (x + 1)
- **Any 'burst' error** (i.e., sequence of consecutive error bits) for which the length of the burst is less than ***k*** bits.
- **Most burst errors** of larger than ***k*** bits can also be detected

Example

- **Which of the following g(x) values guarantees that a single-bit is caught? For each case, what is the error that cannot be caught?**

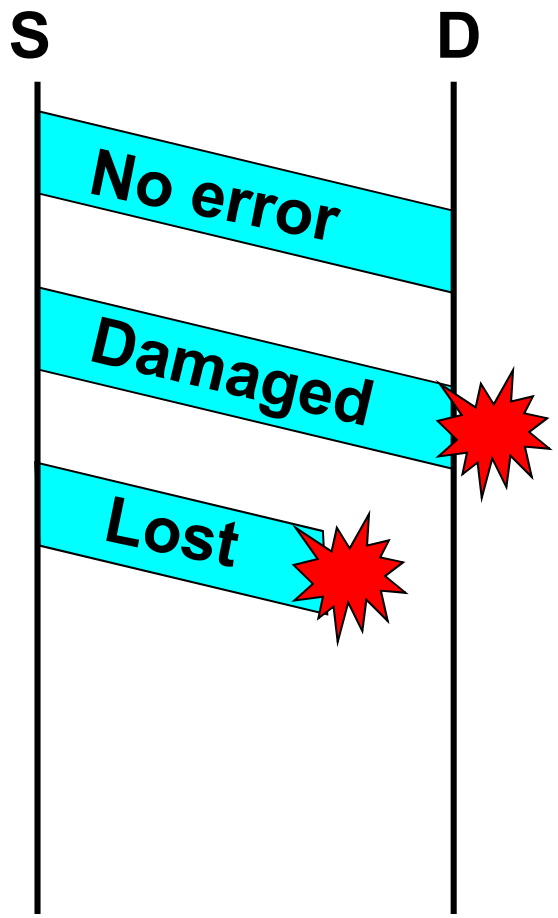
  a. $x + 1$ **b.** x^3 **c.** 1

- **Solution**
  - **a. No x^i can be divisible by $x + 1$. Any single-bit error can be caught.**
  - **b. If i is equal to or greater than 3, x^i is divisible by g(x). All single-bit errors in positions 1 to 3 are caught.**
  - **c. All values of i make x^i divisible by g(x). No single-bit error can be caught. Useless.**

CRC polynomials

- **(ATM header) CRC-8:** $x^8+x^2+x^1+1$
- **(ATM AAL)CRC-10:** $x^{10}+x^9+x^5+x^4+x^1+1$
- **CRC-12:** $x^{12}+x^{11}+x^3+x^2+1$
- **CRC-16:** $x^{16}+x^{15}+x^2+1$
- **(HDLC) CRC-CCITT:** $x^{16}+x^{12}+x^5+1$
- **(LAN) CRC-32:** $x^{32}+x^{26}+x^{23}+x^{22}+$
  $x^{16}+x^{12}+x^{11}+x^{10}+x^8 +$
  $x^7+x^5+x^4+x^2+1$

Error and Flow Control



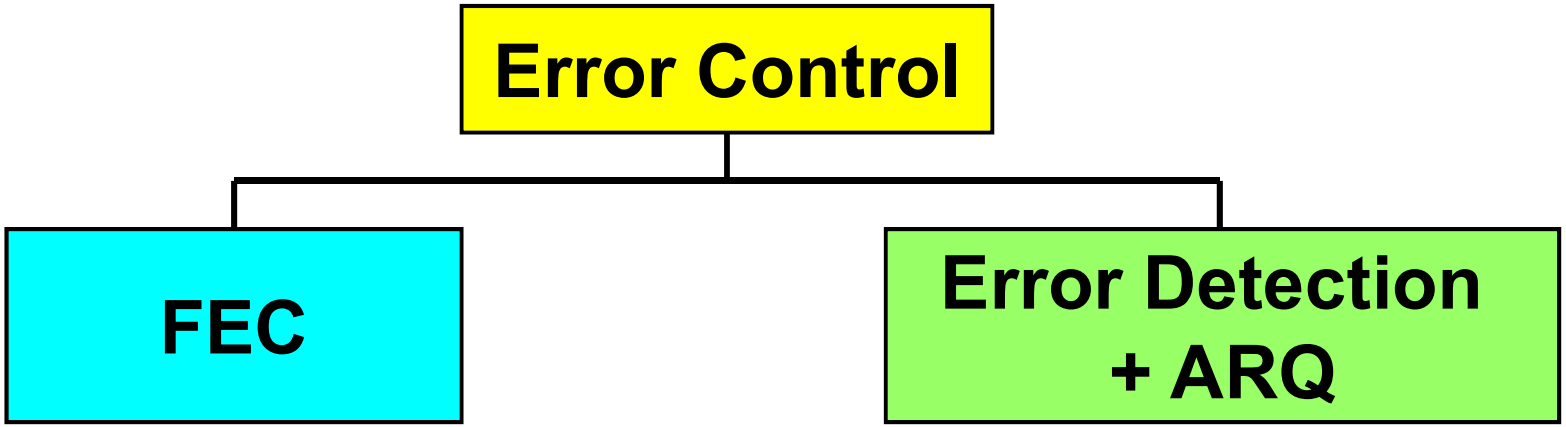
- **Damaged frames**
  - The receiver can recognize the frame, but some bits are in error.
- **Lost frames**
  - The receiver cannot recognize that this is a frame.

Error Control

- **Error control:**

  detection and correction of errors

- **Two approaches for error control:**



FEC

- **FEC: forward error correction**
  - **Error-correcting codes.**
  - **Does not guarantee reliable transmission**
  - **Not cost effective**

Error Detection + ARQ

- **Error detection in a corrupt frame:**
  - Two-dimensional parity
  - Checksum
  - CRC
  - Acknowledgement
  - Sequence
- **Lost frames:**
  - Timeout
- **Correction:**
  - Retransmission

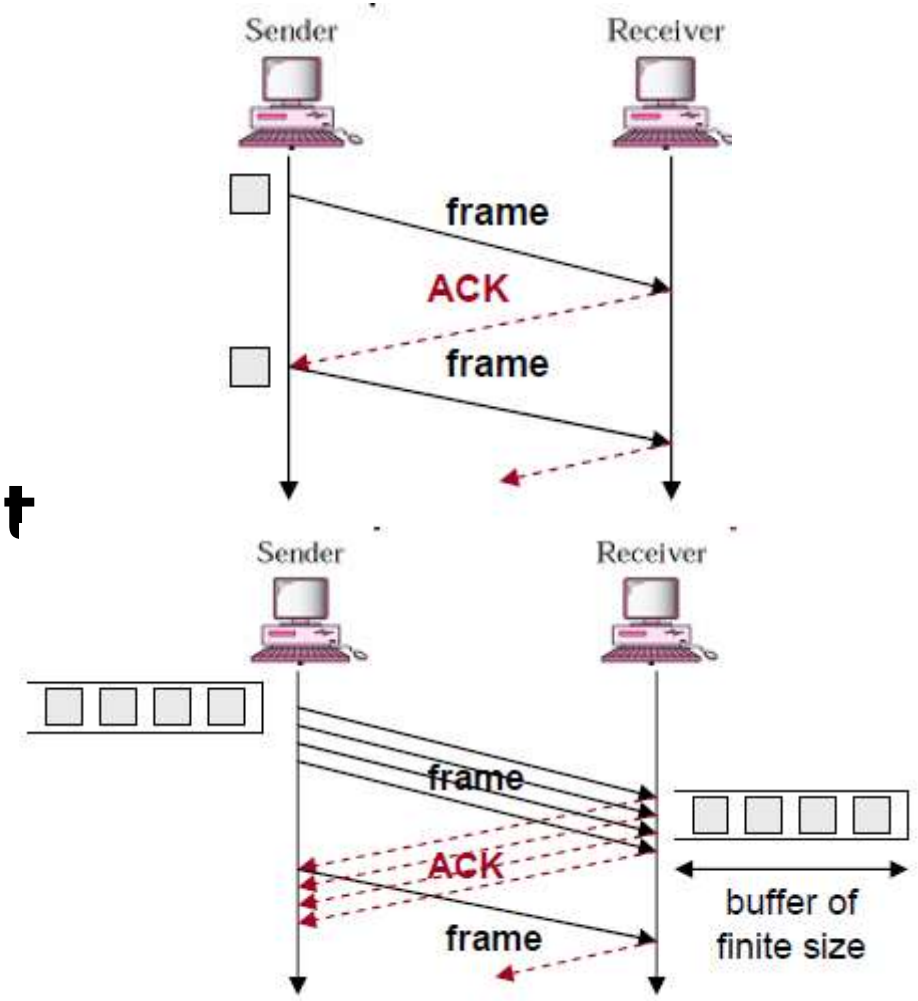
Automatic Repeat Request (ARQ)

- Error control.
- ARQ ensure that transmitted data satisfies the followings:
  - Error free
  - Without duplicates
  - Same order in which they are transmitted
  - No loss

ARQ turns an unreliable data link into a reliable one.

Challenges of ARQ-based Error Control

- Send **one frame** at the time, wait for ACK.
  - **Easy** to implement, **but inefficient** in terms of channel usage.
- Send **multiple frames** at once
  - **Better channel usage, but more complex** - sender must keep (all) sent but unACKed frame(s) in a buffer, as such frame(s) may have to be retransmitted.



How many frames should be sent at any point in time?
How should frames be released from the sending buffer?

Flow Control

- Restrict the amount of data that sender can send while waiting for acknowledgment.
- 2 main strategies
  - **Stop-and-Wait:** sender waits until it receives ACK before sending next frame.
  - **Sliding Window:** sender can send W frames before waiting for ACKs.

Error + Flow Control

- **Versions of ARQ:**
  - (1-bit sliding widow) **Stop-and-Wait ARQ**
  - (sliding widow) **Go-Back-N ARQ**
  - (sliding widow) **Selective Repeat ARQ**

Chapter 3: Roadmap

- 3.1 Introduction and services
- 3.2 Framing
- 3.3 Error Detection and Correction
- 3.4 Stop-and-Wait Protocols
- 3.5 Sliding Window Protocols
- 3.6 HDLC and PPP

Some basic assumptions

- 1. DLL and Network layer are independent processes that communicate by passing messages back and forth trough the physical layer.
- 2. a. Machine A wants to send a long stream of data to machine B, using a reliable, connection-oriented service.

  b. Consider the case where B also wants to send data to A simultaneously.
- 3. Machines do not crash.

Protocol Definitions (1/3)

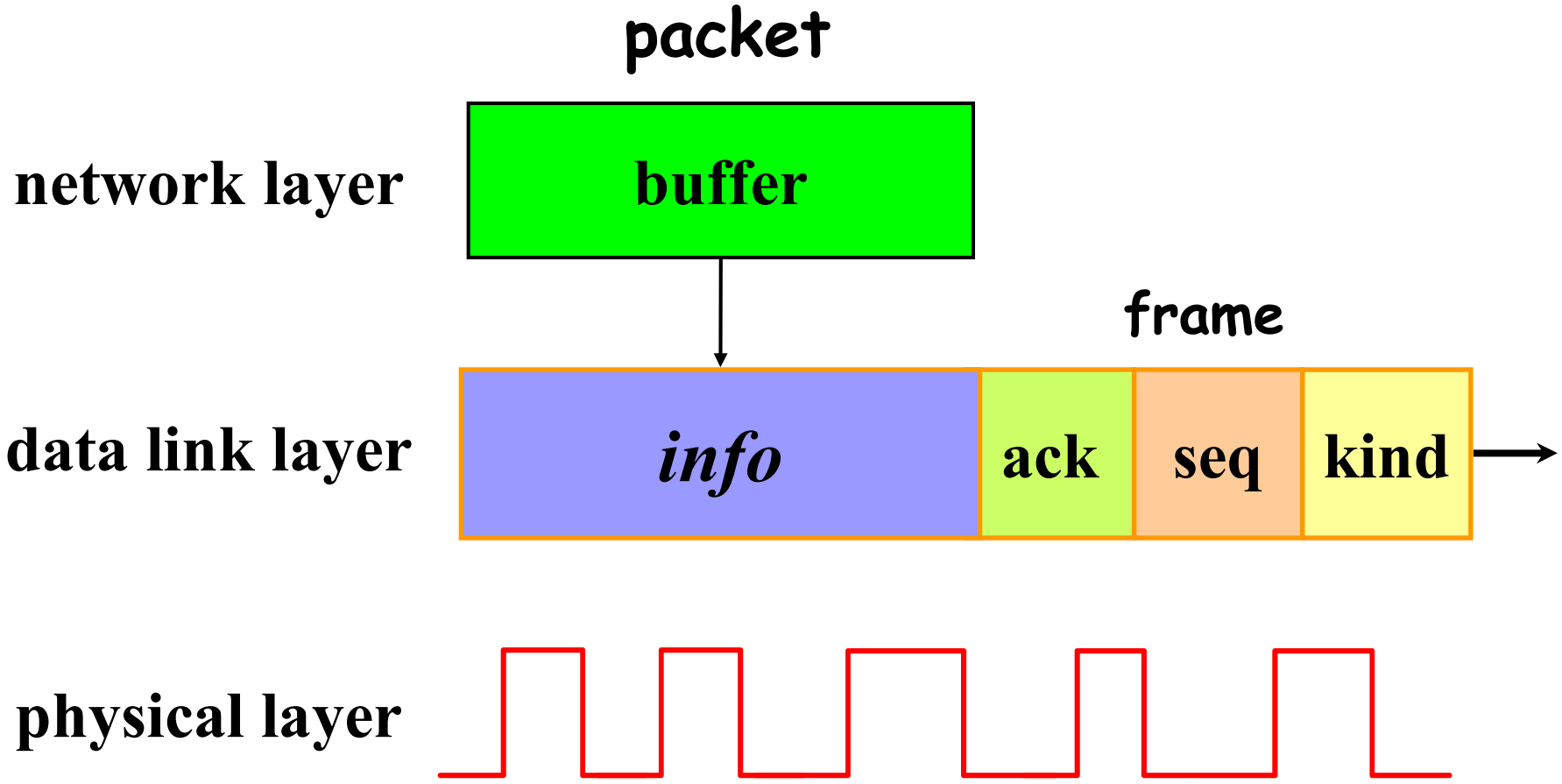
```
#define MAX_PKT 1024                                  /* determines packet size in bytes */

typedef enum {false, true} boolean;                   /* boolean type */
typedef unsigned int seq_nr;                          /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;             /* frame_kind definition */

typedef struct {                                      /* frames are transported in this layer */
  frame_kind kind;                                    /* what kind of a frame is it? */
  seq_nr seq;                                         /* sequence number */
  seq_nr ack;                                         /* acknowledgement number */
  packet info;                                        /* the network layer packet */
} frame;
```

Continued →

Protocol Definitions (2/3)

```
/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Protocol Definitions (3/3)



Stop-and-Wait ARQ

- An Unrestricted Simplex Protocol (protocol 1)
- A Simplex Stop-and-Wait Protocol (protocol 2)
- A Simplex Stop-and-Wait Protocol for a Noisy Channel (protocol 3)

Stop-and-Wait ARQ

- Source transmits a frame.
- Destination receives the frame, and replies with a small frame called acknowledgement (ACK).
- Source waits for the ACK before sending the next frame.
  - This is the core of the protocol !
- Destination can stop the flow by not sending ACK (e.g., if the destination is busy …).

An Unrestricted Simplex Protocol

- **Protocol 1** (utopia):
  - Data are transmitted in one direction only.
  - Both the transmitting and receiving networks are always ready.
  - Processing time can be ignored.
  - Infinite buffer space is available.
  - Communication channel never damages or loses frames.

An Unrestricted Simplex Protocol

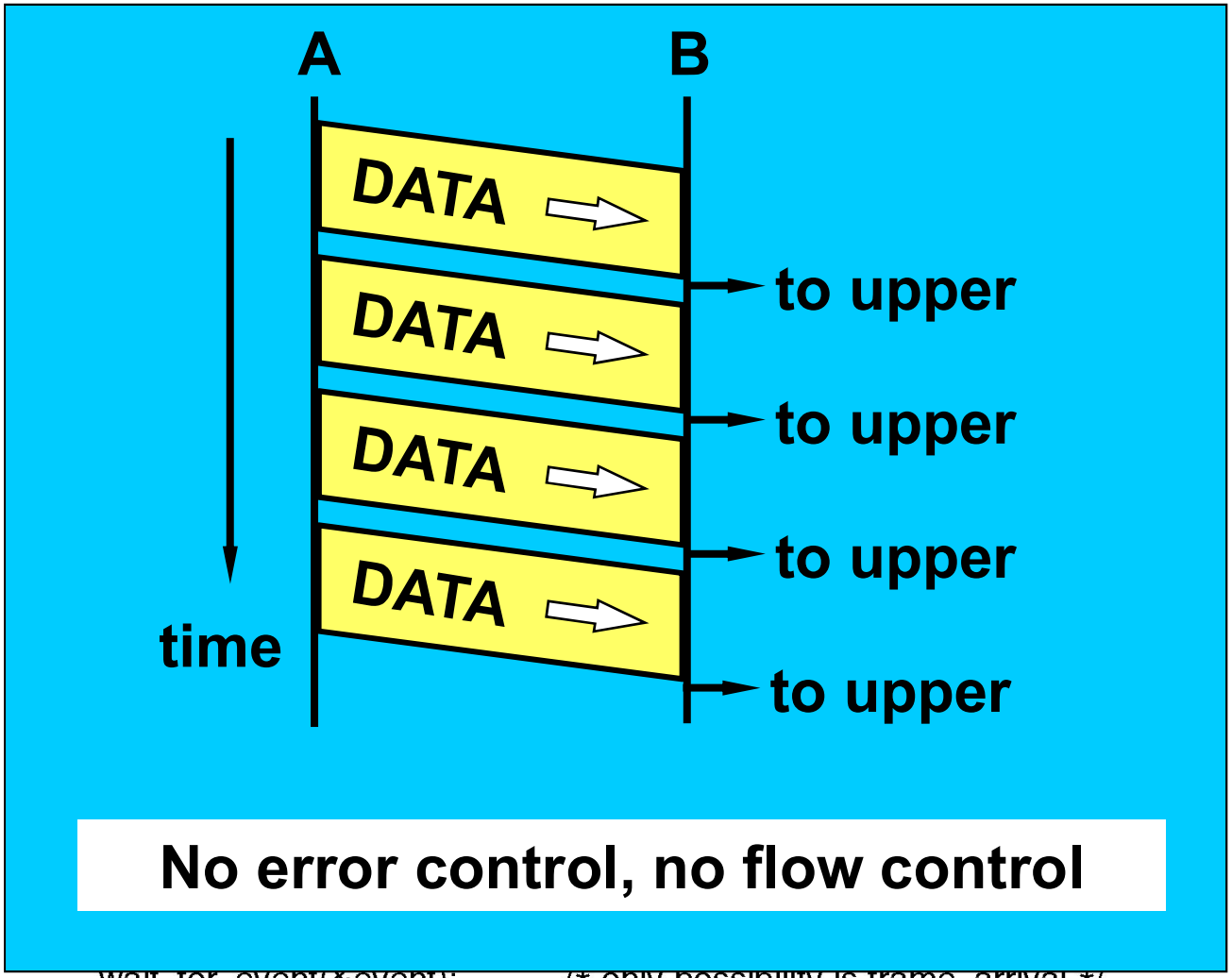
- **Protocol 1 (utopia):**
  - The protocol consists of two distinct procedures, a sender and a receiver.
  - No sequence number or acknowledgements are used, *MAX_SEQ* is not needed.
  - The only event type possible is *frame_arrival*.
  - The sender is in an infinite while loop just pumping data out onto the line as fast as it can. The receiver is equally simple.

Protocol 1 Unrestricted Simplex Protocol (utopia)

```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free,
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* send it on its way */
  }                                     /          * Tomorrow, and tomorrow, and tomorrow,
                                                     Creeps in this petty pace from day to day
                                                     To the last syllable of recorded time
                                                         - Macbeth, V, v */
}

void receiver1(void)
{
  frame r;
  event_type event;                     /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event);           /* only possibility is frame_arrival */
      from_physical_layer(&r);          /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
  }
}
```

Protocol 1 Unrestricted Simplex Protocol (utopia)

/* Protocol 1 (utopia) provides for data transmission in one direction only, from sender to receiver. The communication channel is assumed to be error free, and the receiver is assumed to be able to process all the input infinitely quickly. Consequently, the sender just sits in a loop pumping data out onto the line as fast as it can. */




```
        wait_for_event(&event);        /* only possibility is frame_arrival */
        from_physical_layer(&r);       /* go get the inbound frame */
        to_network_layer(&r.info);     /* pass the data to the network layer */
    }
}
```

A Simplex Stop-and-Wait Protocol

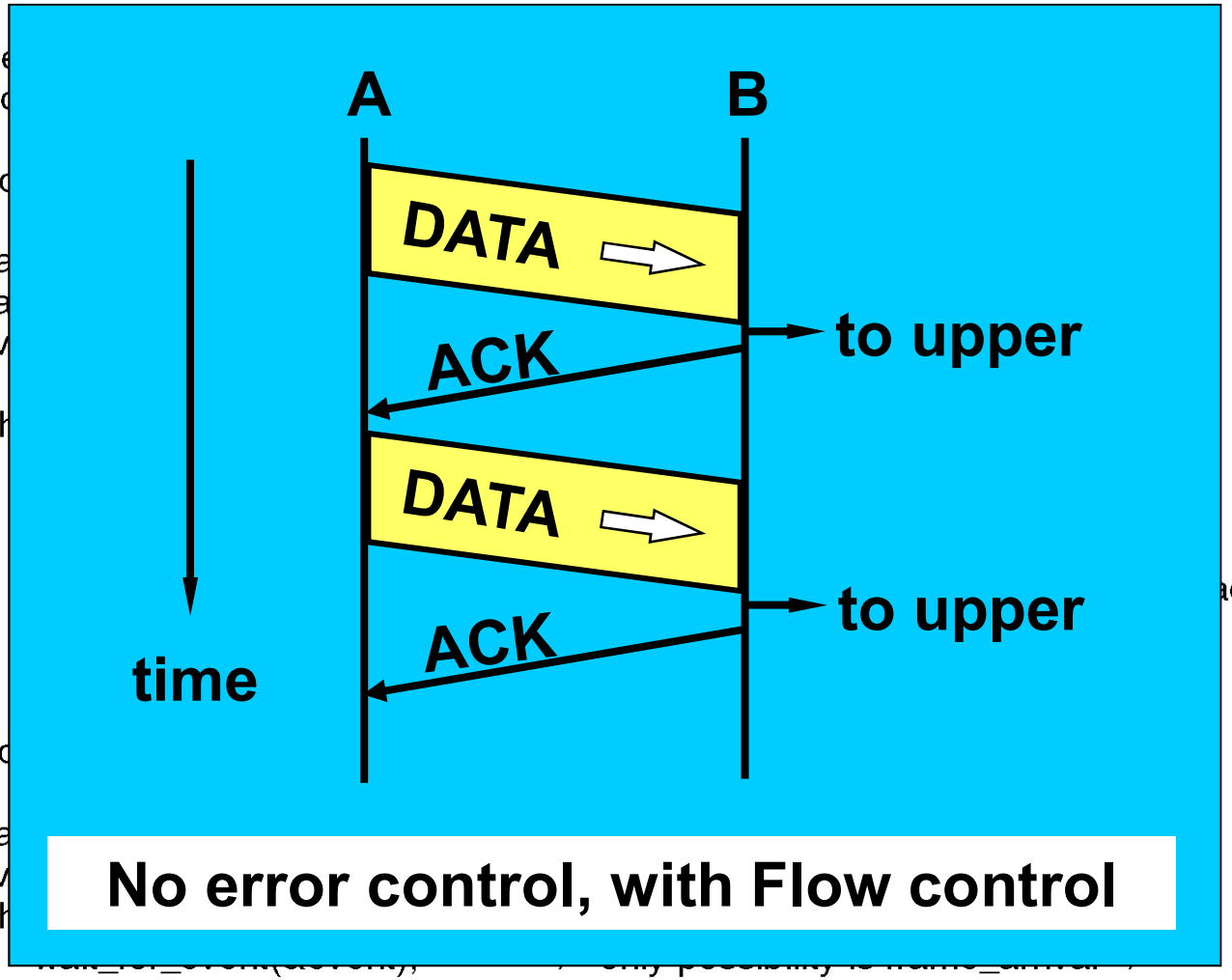
- **Protocol 2:** Communication channel is assumed to be error free and the data traffic is simplex.
  - The problem： how to prevent the sender from flooding the receiver with data faster than the latter is able to process it.
  - The solution: having the receiver provide feedback to the sender. Sender has to wait for a acknowledgment from receiver before transmit the next. → Stop-and-wait.

Protocol 2 Simplex Stop-and-Wait Protocol

```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                          /* buffer for an outbound frame */
  packet buffer;                    /* buffer for an outbound packet */
  event_type event;                 /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);  /* go get something to send */
      s.info = buffer;              /* copy it into s for transmission */
      to_physical_layer(&s);        /* bye bye little frame */
      wait_for_event(&event);       /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                       /* buffers for frames */
  event_type event;                 /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);       /* only possibility is frame_arrival */
      from_physical_layer(&r);      /* go get the inbound frame */
      to_network_layer(&r.info);    /* pass the data to the network layer */
      to_physical_layer(&s);        /* send a dummy frame to awaken sender */
  }
}
```

Protocol 2 Simplex Stop-and-Wait Protocol

/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free, as in protocol 1. However, this time, the receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled. */




```
type
#inc

voic
{
  fra
  pa
  ev

  wh

  }
}

voic
{
  fra
  ev
  wh
        from_physical_layer(&r);          /* go get the inbound frame */
        to_network_layer(&r.info);        /* pass the data to the network layer */
        to_physical_layer(&s);            /* send a dummy frame to awaken sender */
  }
}
```

A Simplex Stop-and-Wait Protocol for a Noisy Channel

- **Protocol 3:** We do assume that damaged frames can be detected, but also that frames can get lost entirely.
  - **Problem 1:** A sender doesn't know whether a frame has made it (correctly) to the receiver.

    Solution: Let the receiver acknowledge undamaged frames.

A Simplex Stop-and-Wait Protocol for a Noisy Channel

- **Protocol 3:**
  - **Problem 2:** Acknowledgments may get lost.

    Solution: let the sender use a timer by which it simply retransmits unacknowledged frames after some time.

A Simplex Stop-and-Wait Protocol for a Noisy Channel

- **Protocol 3:**
  - **Problem 3:** The receiver cannot distinguish duplicate transmissions.

    Solution: use sequence numbers.
  - **Problem 4:** Sequence numbers cannot go on forever.

    Solution: We can (and need) only use a few of them. [In stop-and-Wait protocol, we need only two (0 & 1)]

Protocol 3 A Simplex Protocol for a Noisy Channel

```
/* 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 */
           }
      }
 }
}
```

A positive acknowledgement with retransmission protocol.

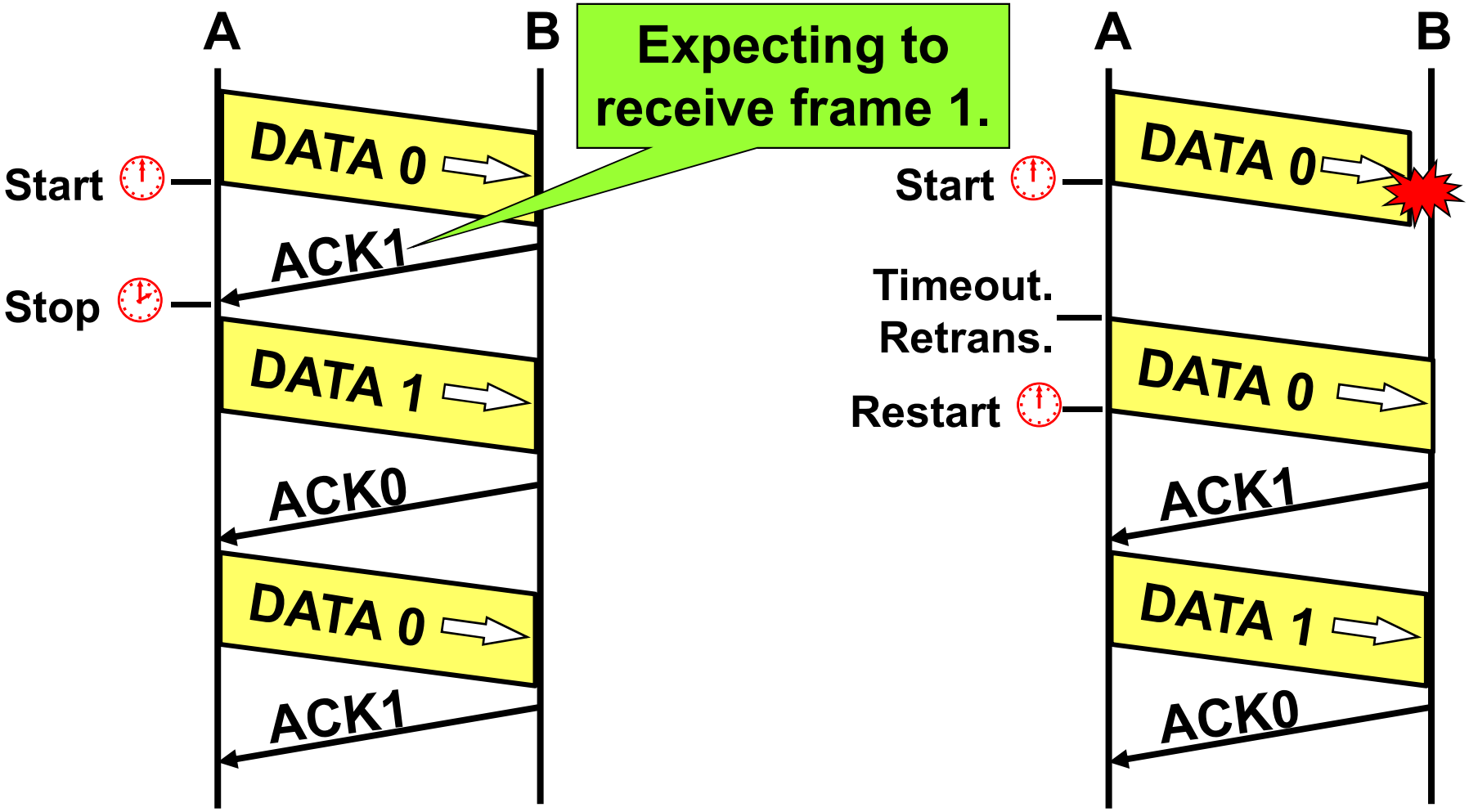
Continued →

A Simplex Stop-and-Wait 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 */
       }
  }
}
```

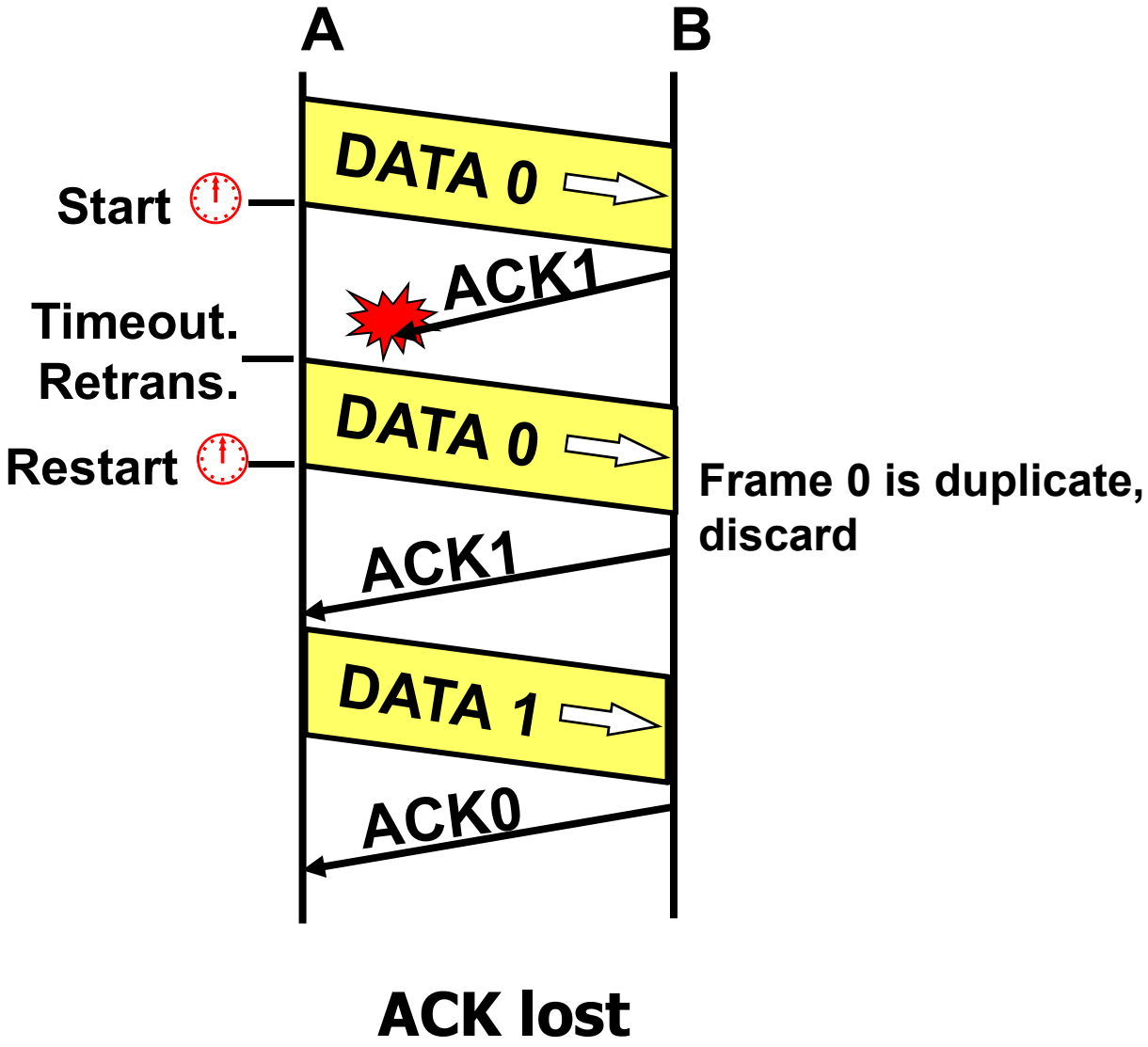
A positive acknowledgement with retransmission protocol.

Protocol 3 in action



Normal operation **Data frame lost**

Protocol 3 in action



ACK lost

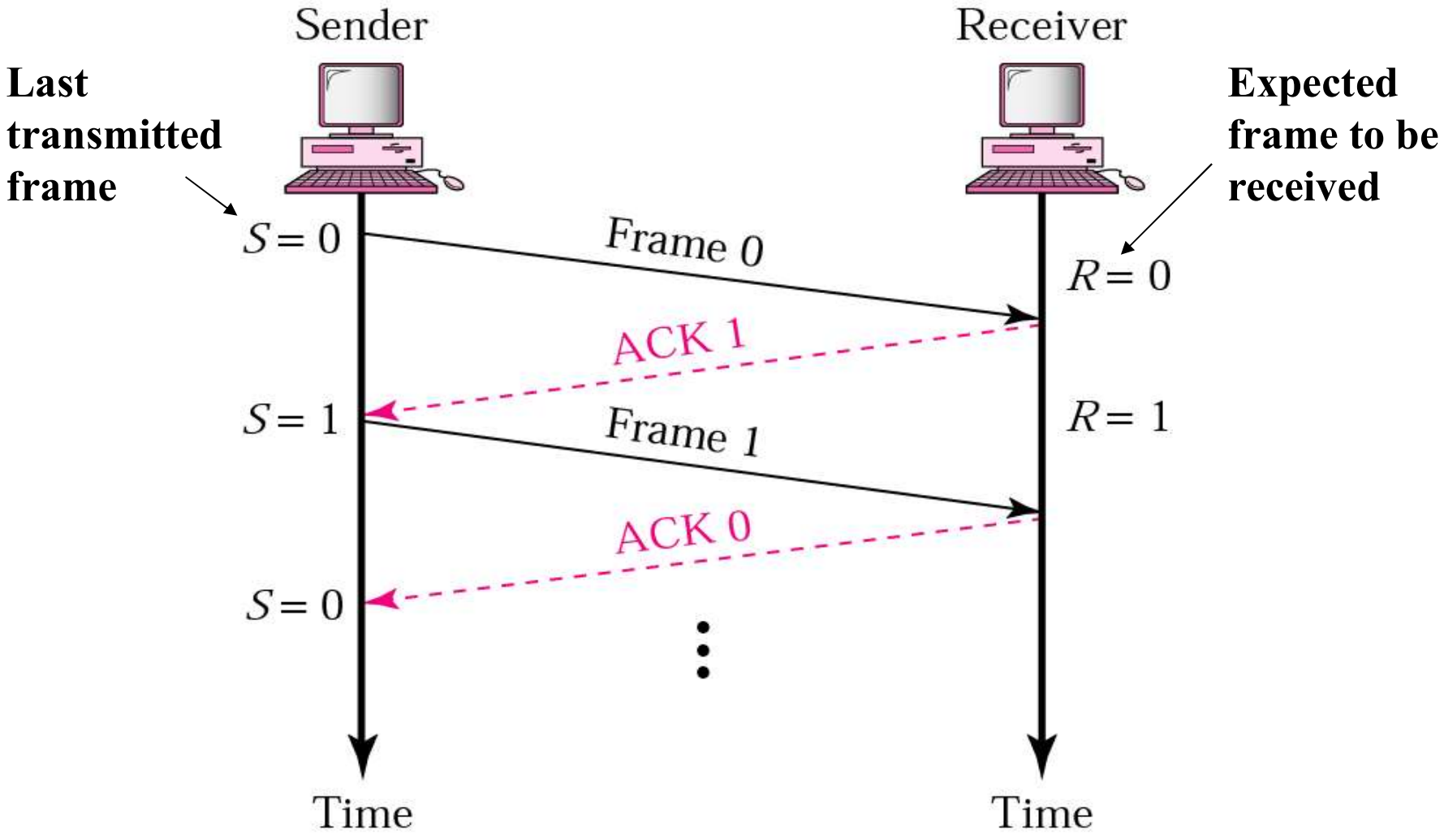
Stop-and-Wait ARQ

- Simplest flow and error control protocol
- Data frames and ACK frames are numbered alternately (0,1)
- When receiver sends ACK1: it acknowledges data frame 0 and is expecting data frame 1, ACK0 acknowledges data frame 1 and is expecting data frame 0
- Sequence Numbers are incremented modulo 2.
- Receiver has a counter (R) which hold the number of the expected frame to be received. R is incremented by 1 modulo 2 when the expected frame is received.
- The sender keeps a counter (S) which holds the number of the last transmitted frame. S is incremented by 1 modulo 2 when the acknowledgement of the last transmitted frame is received and the next frame is transmitted

Stop-and-Wait ARQ

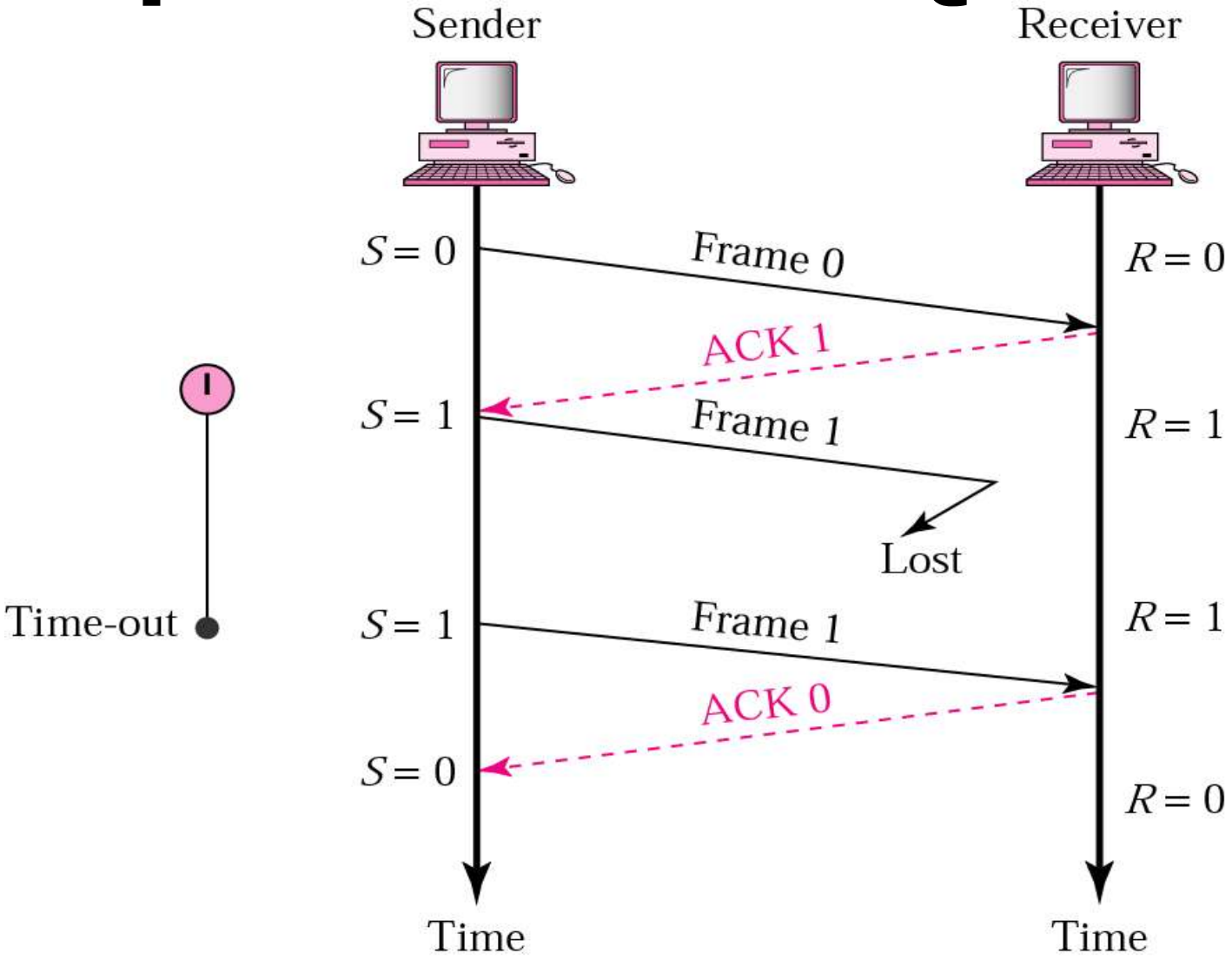
- If the receiver detects an error in the frame it discards it
- If the receiver receives an out-of-order frame (frame 0 instead of frame 1 or vice versa), it knows that the expected frame is lost or damaged and discards the out-of-order frame and resend the previous ACK
- If the sender receives an ACK with a different number than the current value of S+1, it discards it.
- The sending device keeps a copy of the last frame transmitted until it receives the right acknowledgment (ACK) for the frame
- The sender starts a timer when it sends a frame. If an ACK is not received within the allocated time, the sender assumes that the frame was lost or damaged and resends it

Stop-and-Wait ARQ



Normal operation

Stop-and-Wait ARQ



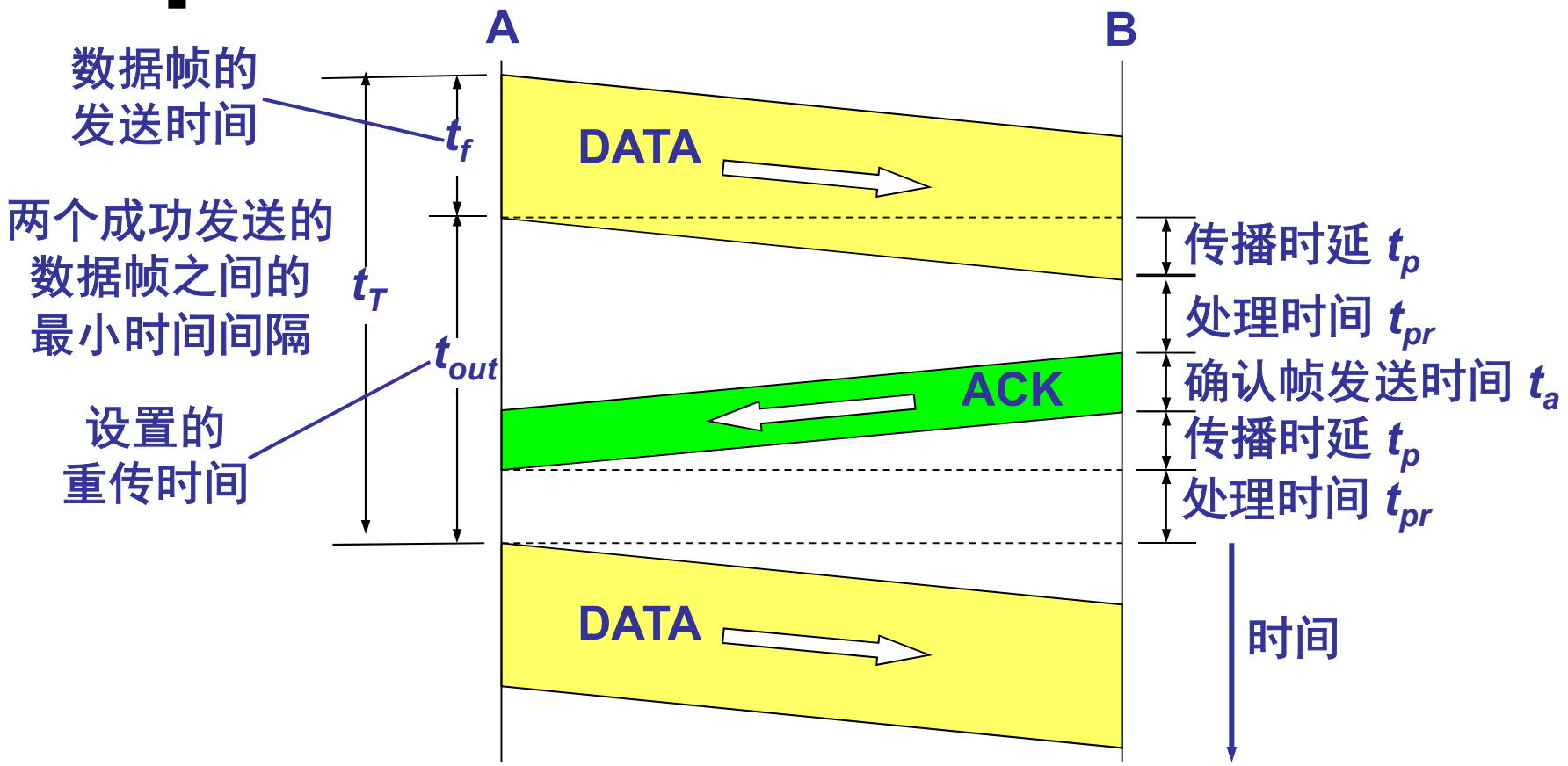
Stop-and-Wait ARQ, lost frame

Stop-and-Wait ARQ

In Stop-and-Wait ARQ, numbering frames prevents the retaining of duplicate frames.

Numbered acknowledgments are needed if an acknowledgment for frame is delayed and the next frame is lost.

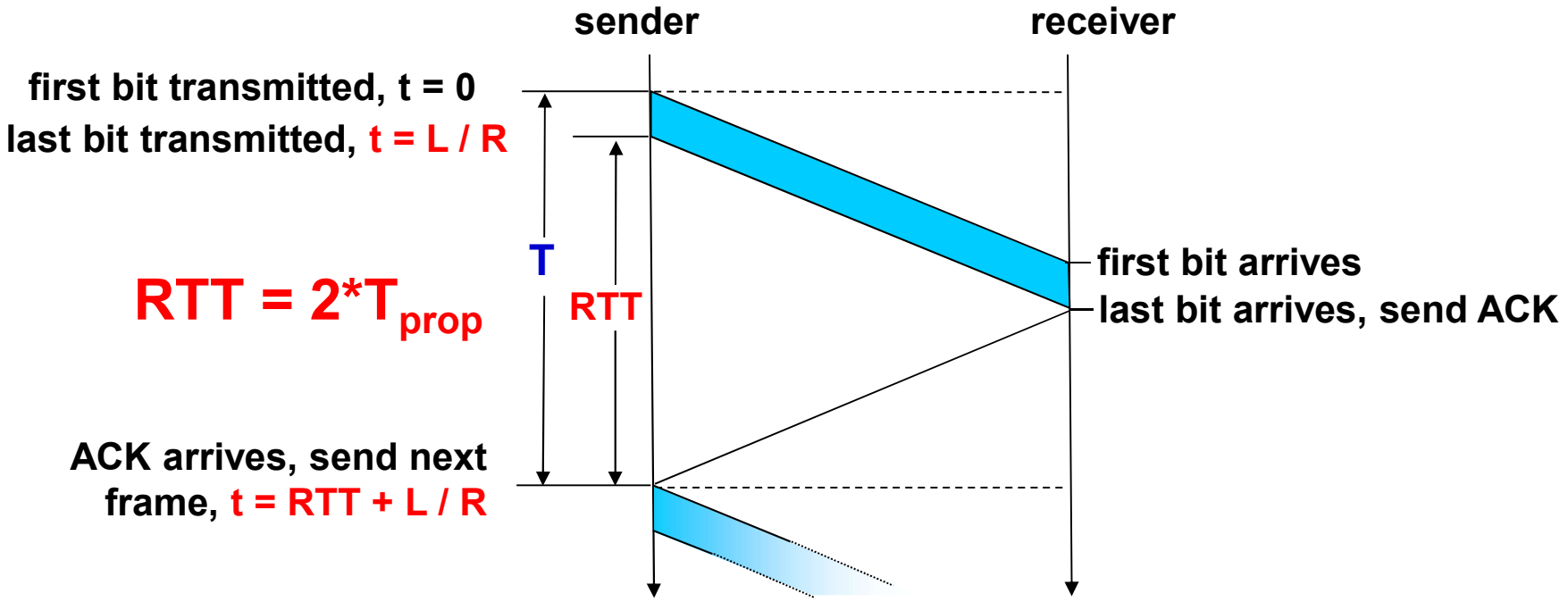
Stop-and-Wait Link Utilization



channel utilization

$$U = \frac{t_f}{t_T} = \frac{t_f}{t_p + t_{pr} + t_a + t_p + t_{pr} + t_f} = \frac{t_f}{2t_p + t_a + t_f}$$

Stop-and-Wait Link Utilization



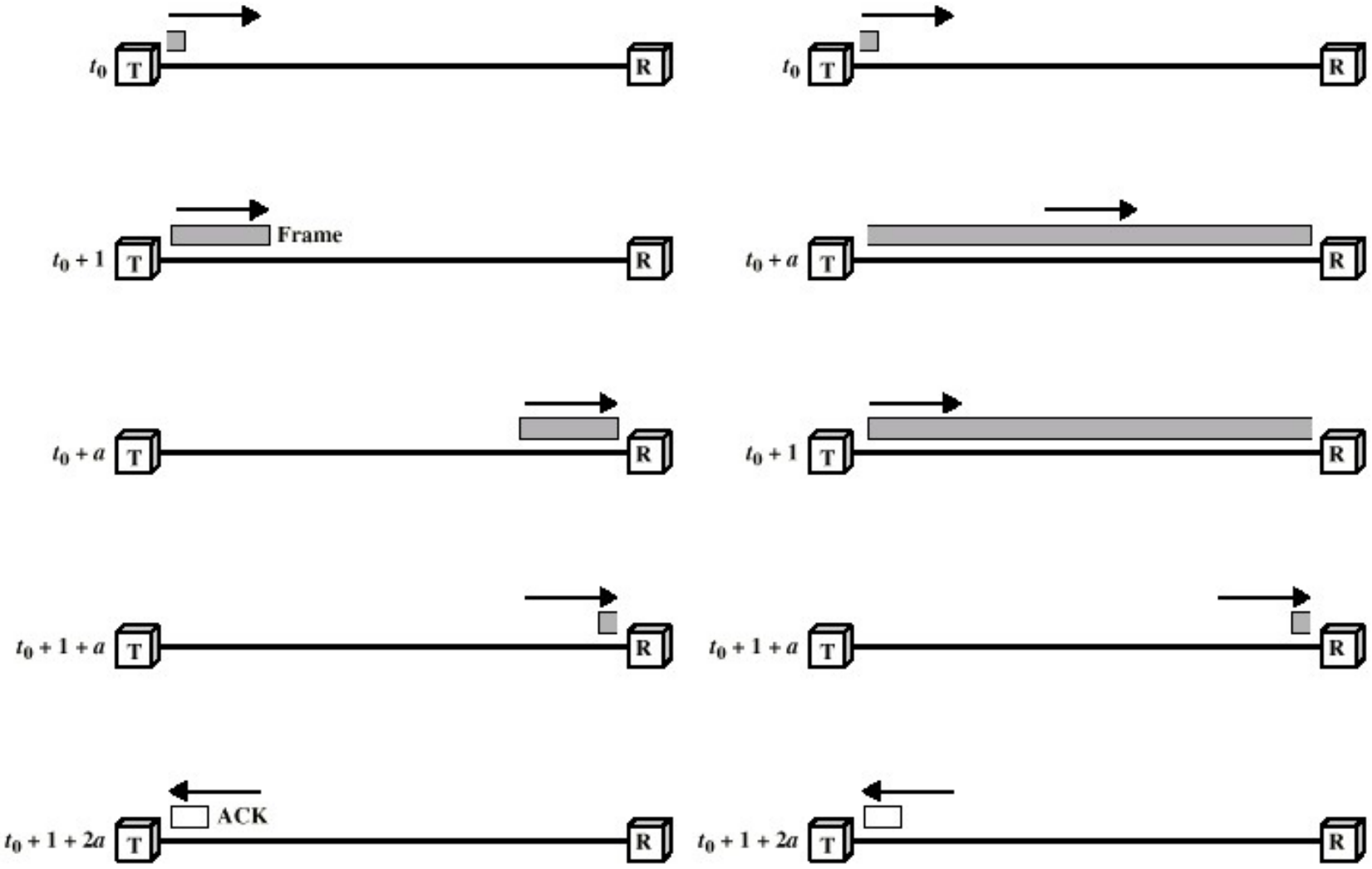
Assumptions:

- Transmission time of the ACK frame is 0.
- Error-free transmission.

$$U = \frac{T_{trans}}{T} = \frac{L/R}{2 \times T_{prop} + L/R} = \frac{1}{1 + 2\alpha}$$

$$\alpha = t_{prop} / t_{trans}$$

Stop-and-Wait Link Utilization



$\alpha > 1$ $\alpha < 1$ (the pipe is full)

Stop-and-Wait Link Utilization

- Example 1
  - Distance between to nodes is d = 0.1~10km;
  - Link data rate is R = 10~100Mbps;
  - Signal propagation speed V = $2*10^8$ m/s;
  - Frame length L = 1000b;

R = 10Mbps, t_{trans} = 1000b/$10*10^6$bps=10^{-4} s

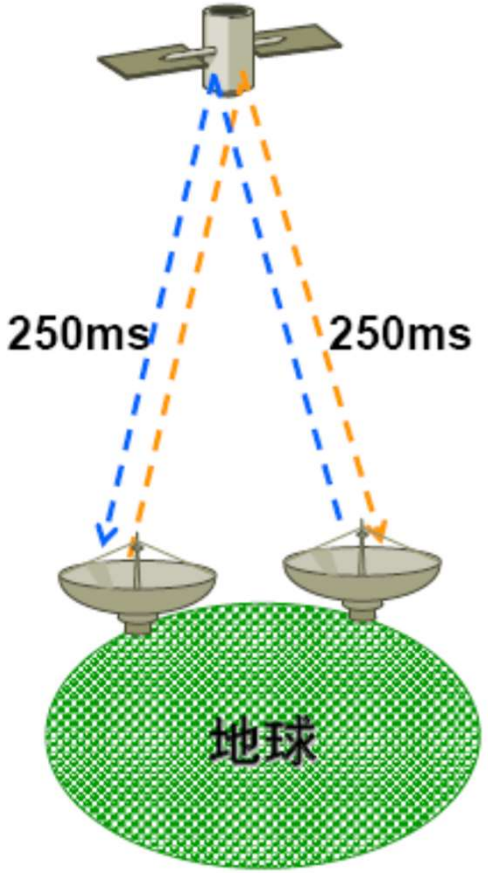
t_{prop} = 100~10000m/$2*10^8$ m/s = $0.5*10^{-6}$ ~ $0.5*10^{-4}$ s

a = t_{prop}/t_{trans} = 0.005 ~ 0.5 (0.1~10km)

U = 0.99 ~ 0.5 (0.1~10km)

Stop-and-Wait Link Utilization

- **Example 2**
  - **L = 1000b; R = 50kps;**
  - **Propagation delay = 250ms;**



t_{trans} = 1000b/5*10^{5}bps=20ms

U = 20/(2*250 + 20) = 3.8%

Stop-and-Wait Link Utilization

- **Example 3:**

  1 Gbps link, 15 ms prop delay. 1KB frame. How many bits per time unit can this protocol achieve?

  1KB every 30 msec -> 33kB/sec thruput over 1 Gbps link. 1024*8 / 30ms = 33kBps = 270kbps

$$T_{transmit} = \frac{L \text{ (length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

network protocol limits use of physical resources!

Stop-and-Wait Link Utilization

- **Example 4:**

  Assume that, in a Stop-and-Wait ARQ 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.

Stop-and-Wait Link Utilization

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 = $1000/ 10^6 + 2 \times 10 \times 10^{-3} = 0.021$ 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 ARQ is used?

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

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

Stop-and-Wait Link Utilization

Data Frame = 1250 bytes = 10000 bits
ACK = frame header = 25 bytes = 200 bits

Utilization	200 km (t_{prop} = 1 ms)	2000 km (t_{prop} = 10 ms)	20000 km (t_{prop} = 100 ms)	200000 km (t_{prop} = 1 sec)
1 Mbps	10^3 88%	10^4 49%	10^5 9%	10^6 1%
1 Gbps	10^6 1%	10^7 0.1%	10^8 0.01%	10^{49} 0.001%

Stop-and-Wait does NOT work well for very high speeds or long propagation delays.

Stop-and-Wait Link Utilization

- **How to improve efficiency?**

Send more frames while sender is waiting for ACK. ⇒

sliding window algorithms.

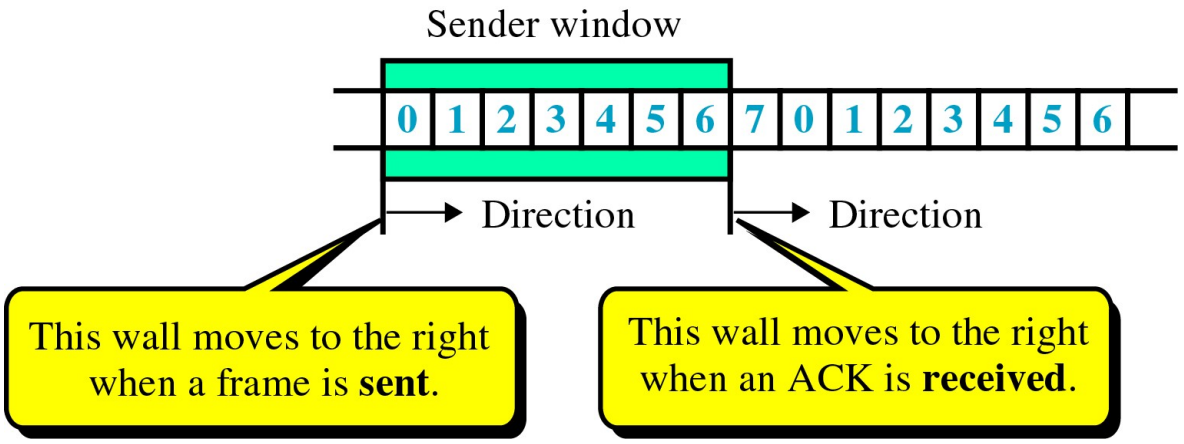
Chapter 3: Roadmap

- 3.1 Introduction and services
- 3.2 Framing
- 3.3 Error Detection and Correction
- 3.4 Stop-and-Wait Protocols
- 3.5 Sliding Window Protocols
- 3.6 HDLC and PPP

Sliding window Protocols

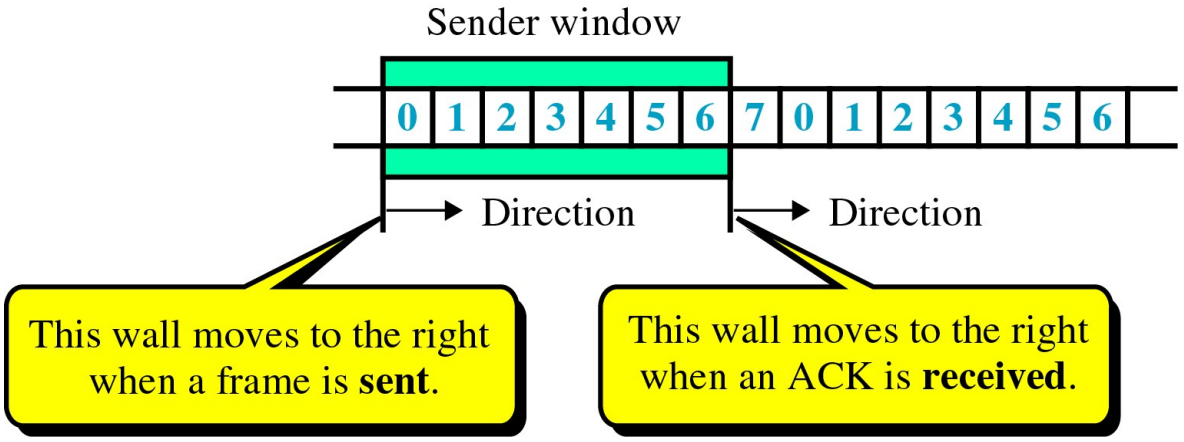
- **Allow multiple frames to be in transit.**
- **Receiver has buffer W long.**
- **Transmitter can send up to W frames without ACK.**
- **Each frame is numbered.**
- **Sequence number bounded by size of field sequence (n).**
  - **Frames are numbered modulo 2^n $(0 \sim 2^n-1)$**

Sending Window



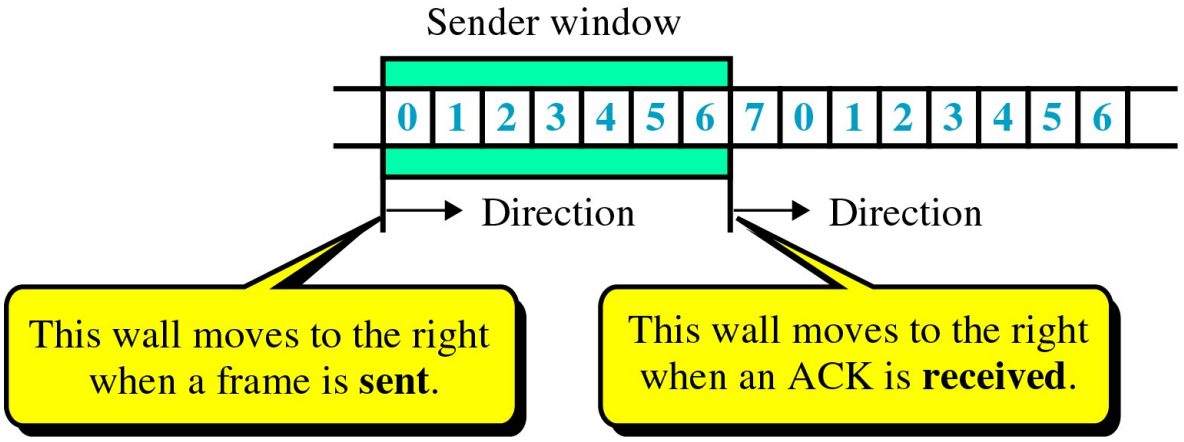
- Sender maintains a set of sequence numbers corresponding to frames it is permitted to send, called the sending window.
- Contains frames that can be sent or have been sent but not yet acknowledged – *outstanding frames*.

Sending Window



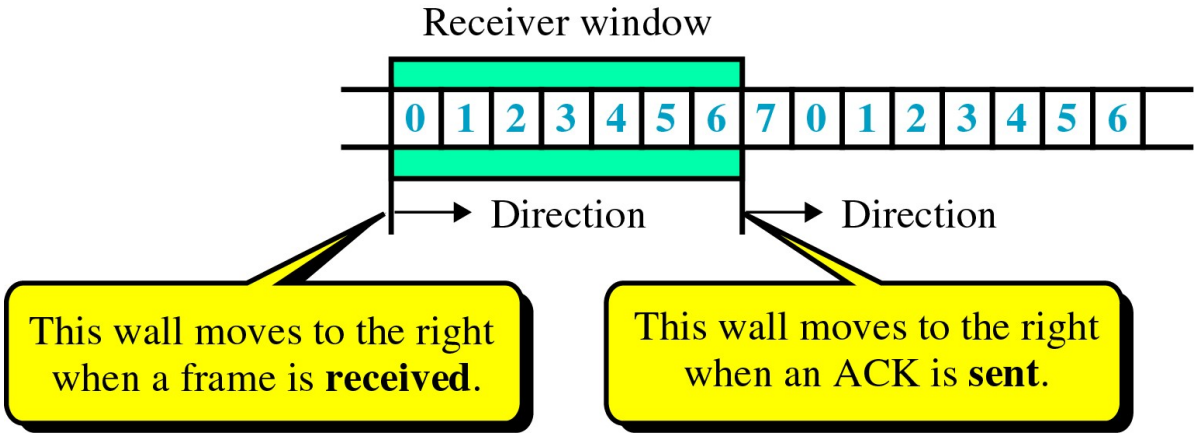
- When a packet arrives from network layer
  - Next highest sequence number assigned
  - Upper edge of window advanced by 1
- When an acknowledgement arrives
  - Lower edge of window advanced by 1

Sending Window



- If the maximum window size is w, w buffers is needed to hold unacknowledged frames.
- If window is full (maximum window size reached)
  - shut off network layer, stop sending.

Receiving Window



- The receiver has a receiving window containing frames it is permitted to receive.
- Frame outside the window → discarded
- When a frame's sequence number equals to lower edge
  - Passed to the network layer
  - Acknowledgement generated
  - Window rotated by 1

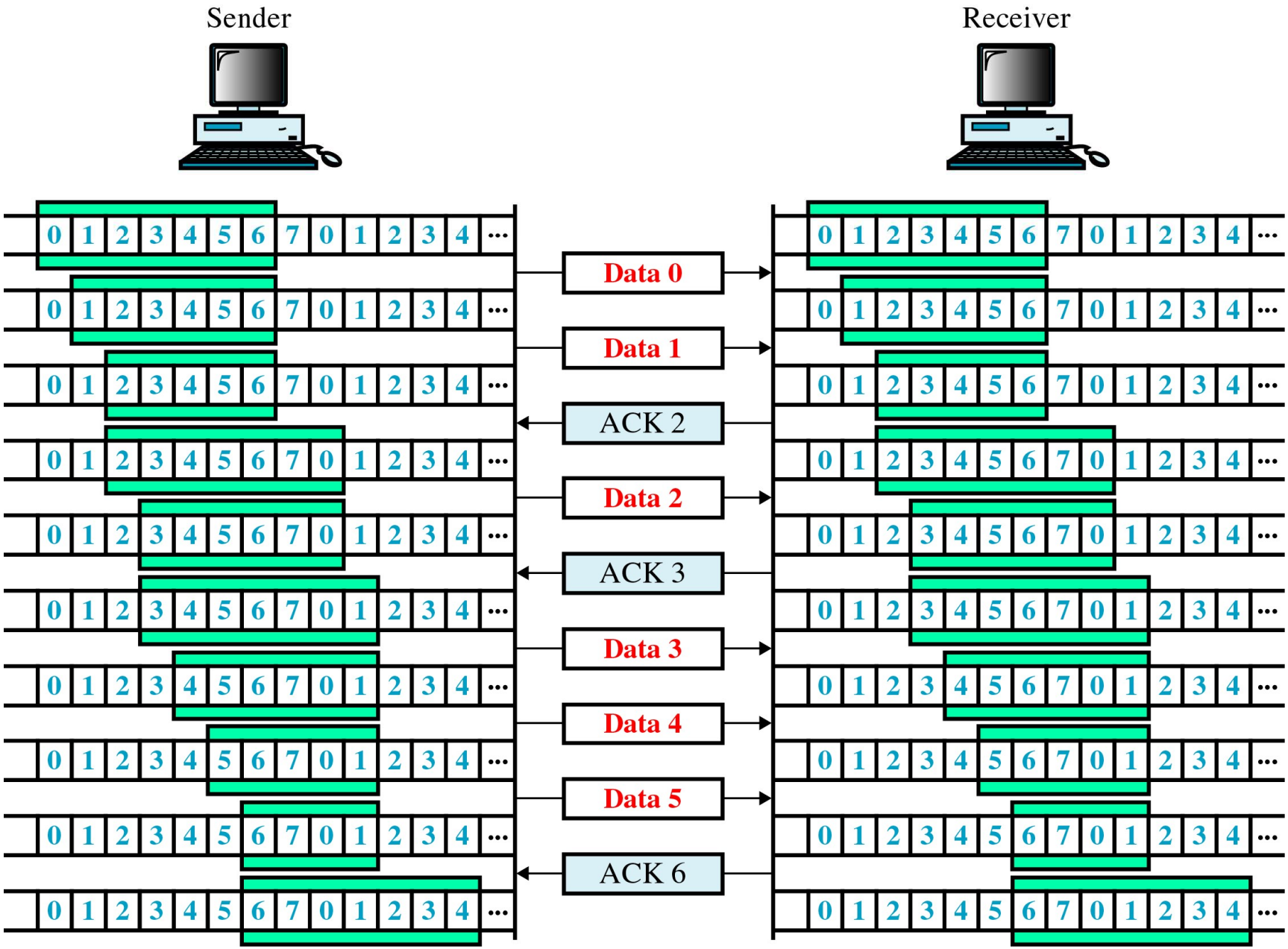
Receiving Window

- Always remains at initial size (different from sending window)
- Size
  - =1 means frames only accepted in order
  - >1 not so

The order of packets fed to the receiver's network layer must be the same as the order packets sent by the sender's network layer.

Receiving Window

- *A damaged frame is kept in the receiving window until a correct version is received.*
- *Also, frame #N is kept in the window until frame #N-1 has been received.*


Sender
Receiver
Data 0
Data 1
ACK 2
Data 2
ACK 3
Data 3
Data 4
Data 5
ACK 6

Standard Ways to ACK

1. ACK sequence number indicates *the last frame successfully received.*

- OR -

2. ACK sequence number indicates *the next frame the receiver expects to receive.*

Standard Ways to ACK

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

Cumulative ACKing is the most common technique.

Cumulative ACKing:
If acknowledgement number is N+1, the receiver has received everything prior to the N+1, and the next expected sequence number is N+1.
One ACK can acknowledge multiple frames.

Full Duplex Communication

- Unidirectional assumption in previous elementary protocols not general.
- Full-duplex:

  allow symmetric frame transmission between two communicating parties.

Full Duplex Communication

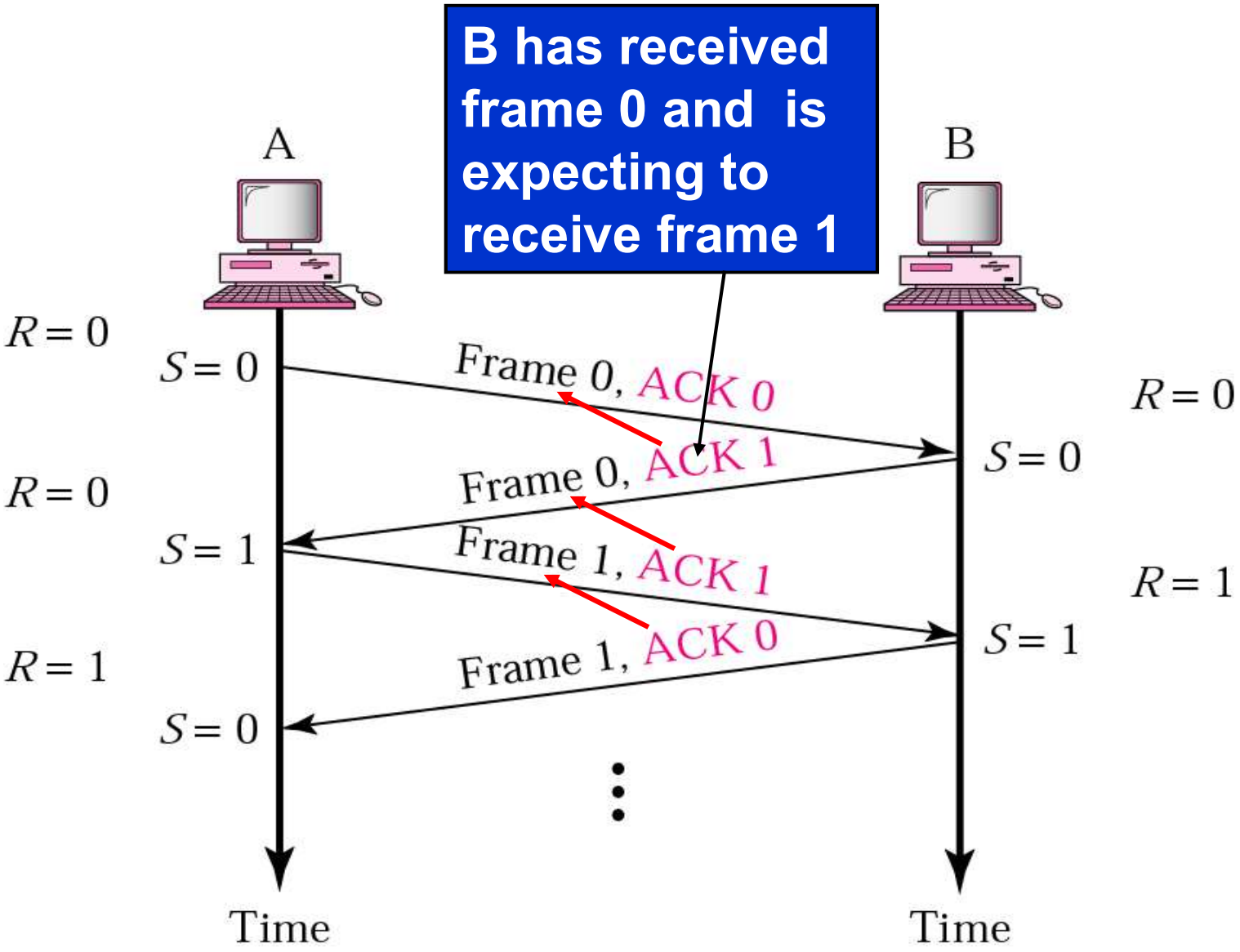
- **Approaches:**
  - Two separate communication channels.
  - Same circuit for both directions.
    - Data and acknowledgement are intermixed
    - How to tell acknowledgement from data?

      kind field tells data or acknowledgement.

  - Piggybacking

    Attaching acknowledgement to outgoing data frames.

Piggybacking



Piggybacking

- Solution for timing complexion
  - If a new frame arrives quickly
    - ⇒ Piggybacking
  - If no new frame arrives after a receiver ack timeout
    - ⇒ Sending a separate ACK

Sliding Window Protocols

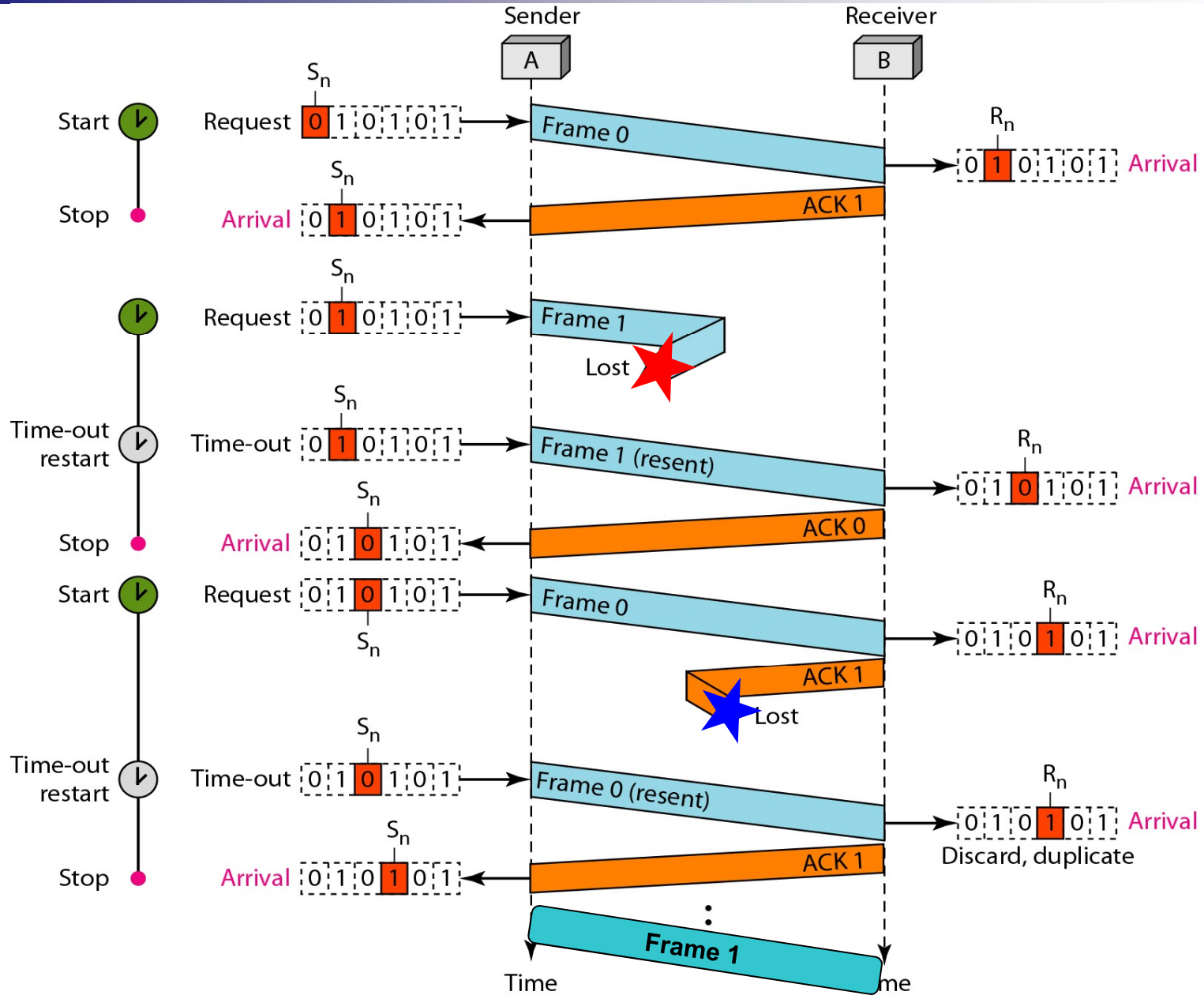
- 3 ***bidirectional sliding window protocols***
  (**max** sending window size, receiving window size)
  - One-Bit Sliding Window Protocol ARQ (1,1)
  - Go-Back-N ARQ (>1, 1)
  - Selective Repeat ARQ (>1, >1)

One-Bit Sliding Window Protocol

- Max sending window size = 1
- receiving window size = 1
- Stop-and-wait protocol
- Acknowledgement =

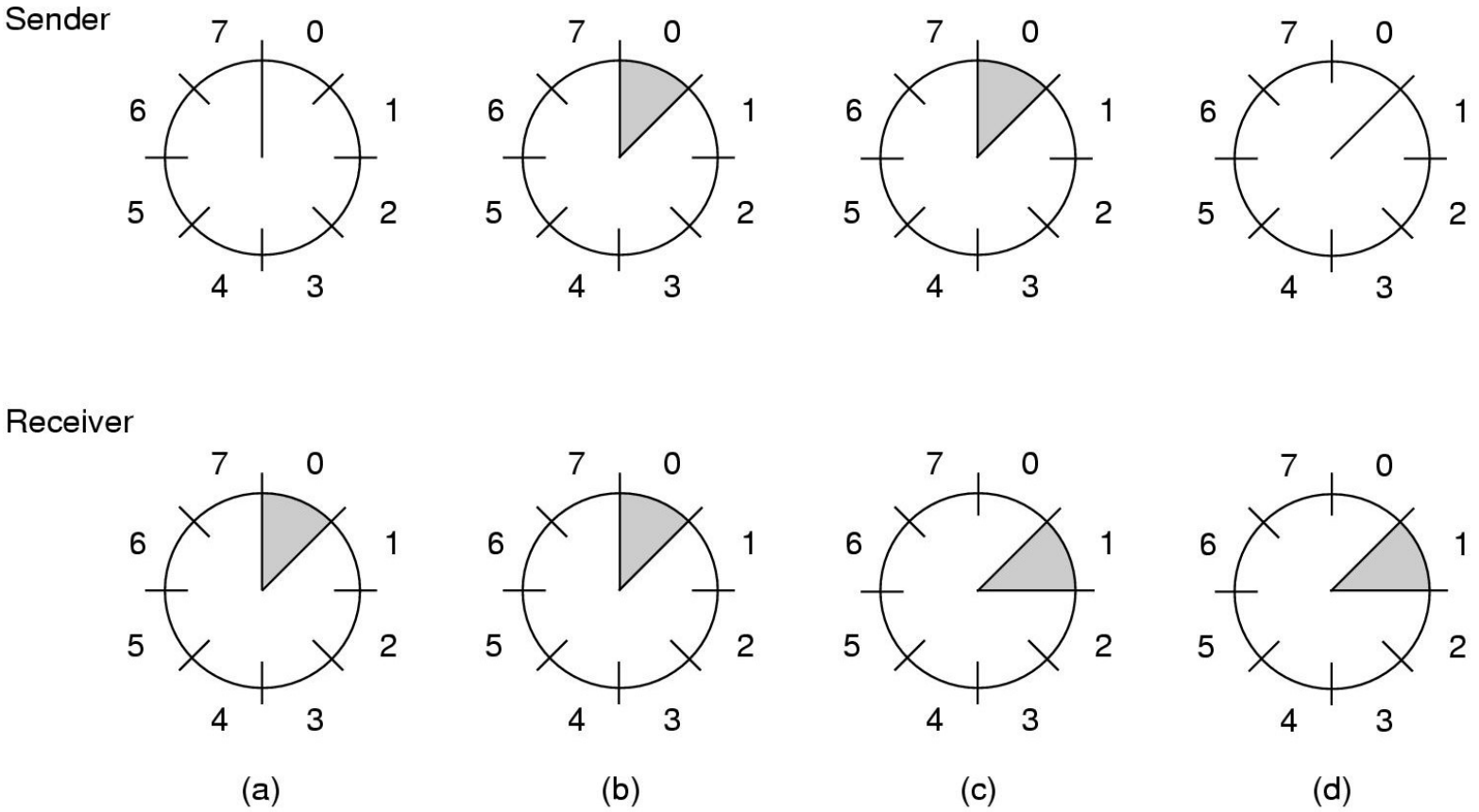
  last frame received w/o error.

  Sequence number expected to receive
- Refer to Protocol 4: A stop-and-wait sliding window protocol with a maximum window size of 1.



Flow diagram when data lost and ACK lost

A Sliding Window of Size 1



A sliding window of size one, with a 3-bit sequence number.

a: Initially. b: After sending frame #0.

c: After receiving frame #0. d: After receiving ack for frame #0.

One-Bit Sliding Window Protocol

- Two scenarios for protocol 4
  - All things go well, but behavior is a bit strange when A and B transmit simultaneously, abnormal case;
  - We are transmitting more than once, just because the two senders are more or less out of sync, normal case.

One-Bit Sliding Window Protocol

```
/* Protocol 4 (sliding window) is bidirectional. */
#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 0 or 1 only */
  seq_nr frame_expected;                            /* 0 or 1 only */
  frame r, s;                                       /* scratch variables */
  packet buffer;                                    /* current packet being sent */
  event_type event;
  next_frame_to_send = 0;                           /* next frame on the outbound stream */
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network layer */
  s.info = buffer;                                  /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                       /* insert sequence number into frame */
  s.ack = 1 – frame_expected;                       /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
```

Continued →

One-Bit Sliding Window Protocol

```
while (true) {
    wait_for_event(&event);                  /* frame_arrival, cksum_err, or timeout */
    if (event == frame_arrival) {            /* a frame has arrived undamaged. */
        from_physical_layer(&r);             /* go get it */

        if (r.seq == frame_expected) {       /* handle inbound frame stream. */
            to_network_layer(&r.info);       /* pass packet to network layer */
            inc(frame_expected);             /* invert seq number expected next */
        }

        if (r.ack == next_frame_to_send) {   /* handle outbound frame stream. */
            stop_timer(r.ack);               /* turn the timer off */
            from_network_layer(&buffer);     /* fetch new pkt from network layer */
            inc(next_frame_to_send);         /* invert senderís sequence number */
        }
    }
    s.info = buffer;                         /* construct outbound frame */
    s.seq = next_frame_to_send;              /* insert sequence number into it */
    s.ack = 1 – frame_expected;              /* seq number of last received frame */
    to_physical_layer(&s);                   /* transmit a frame */
    start_timer(s.seq);                      /* start the timer running */
  }
}
```

Performance of One-Bit Sliding Window Protocol

- If channel capacity = R, frame size = L, and round-trip propagation delay = p, then max. bandwidth utilization = (L/R)/[(L/R)+p]
- Conclusion:

  Long propagation time + high bandwidth + short frame length ⇒ disaster

Performance of One-Bit Sliding Window Protocol

- **Solution: Pipelining**
  - Allowing *w* frames sent before blocking.
- **Problem: errors**
- **Solutions:**
  - Go-back-N ARQ protocol (GBN)
  - Selective-Repeat ARQ protocol (SR)

Pipelining

- **RTT = Round-Trip Time**
- **Long round-trip transmission time can have important implications for the efficiency of the bandwidth utilization.**

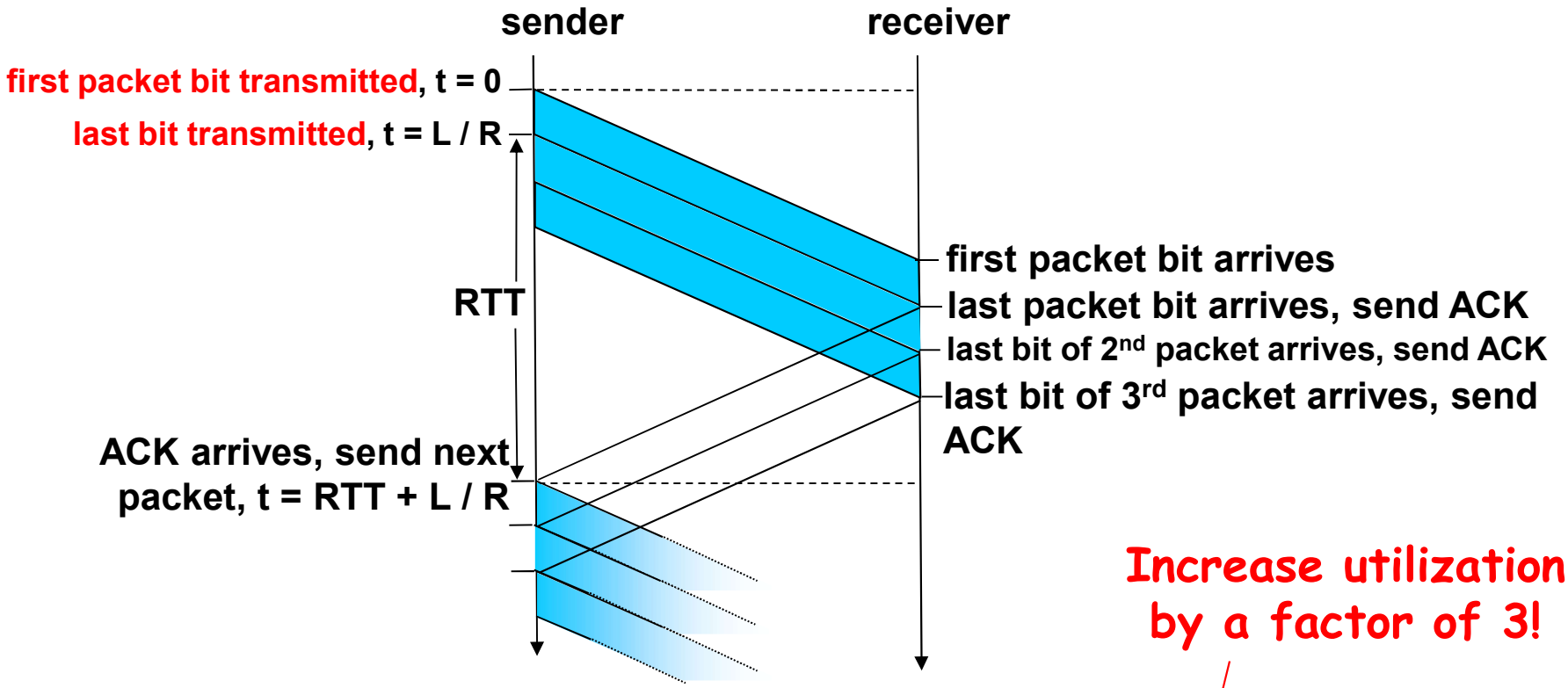
Channel utilization

- If the channel capacity is *R* bits/sec, the frame size is *L* bits, and the round-trip propagation time(delay) is *RTT* sec, the time required to transmit a single frame is *L*/*R* sec.
- The max. channel utilization is

$$L/R \;/\; (L/R + RTT)$$

- If $L/R < RTT$, the efficiency will be less than 50%.

Pipelining: increased utilization



Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

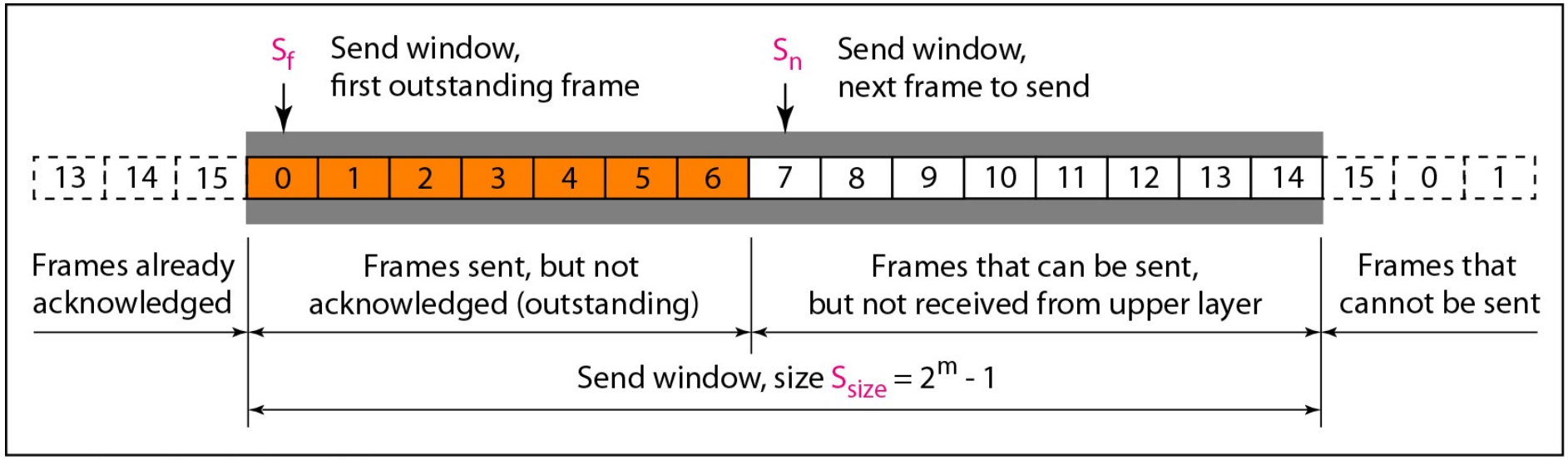
Go-Back-N ARQ

- Based on sliding-window flow control
- Frames are numbered sequentially using n-bit in the frame header
  - n=3 ⇒ (0,1,2,3,4,5,6,7,0,1,2,3,4,5,6,7, …)
- **Sending window size > 1:** to control the number of unacknowledged frames outstanding. Sender can send multiple frames while waiting for ACK.
- **Receiving window size = 1:** frames must be accepted **in the order** they were sent.

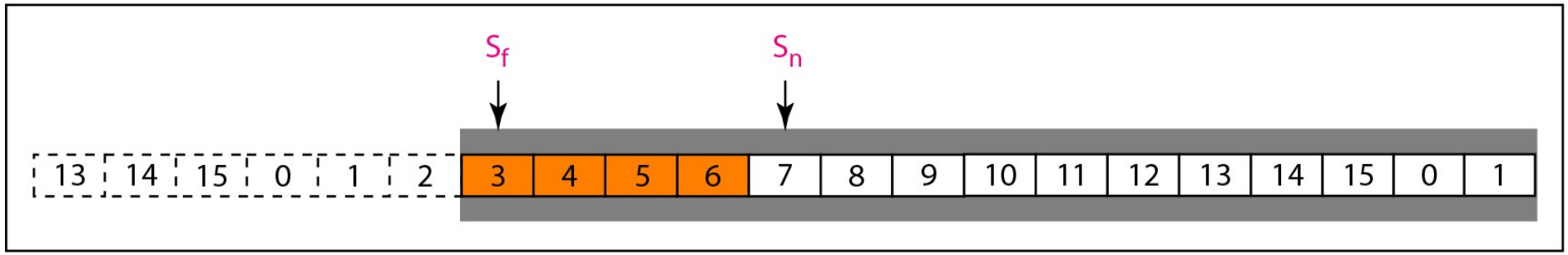
Go-Back-N ARQ

- If no error, the receiver will send ACK as usual with next frame expected.
- If error occurs,
  - Receiver will discards that frame and all subsequent frames following an error one, until the erroneous frame is received correctly.
  - Sender must go back and retransmit that frame and all succeeding frames that were transmitted in the interim.

Go-Back-N ARQ

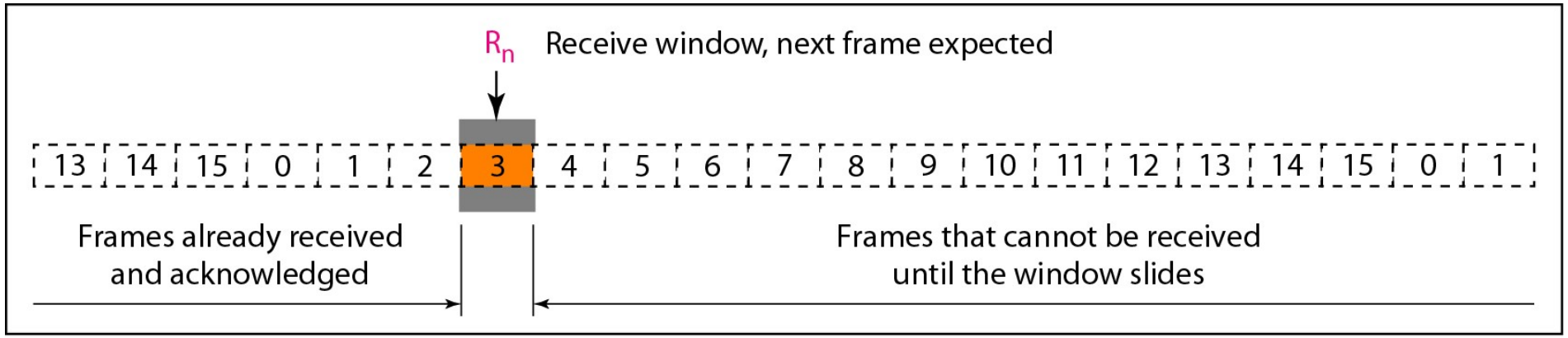


a. Send window before sliding

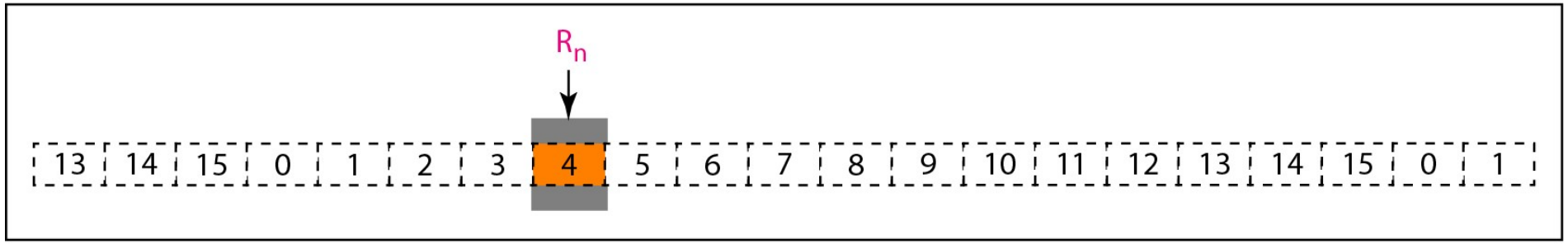


b. Send window after sliding

Go-Back-N ARQ



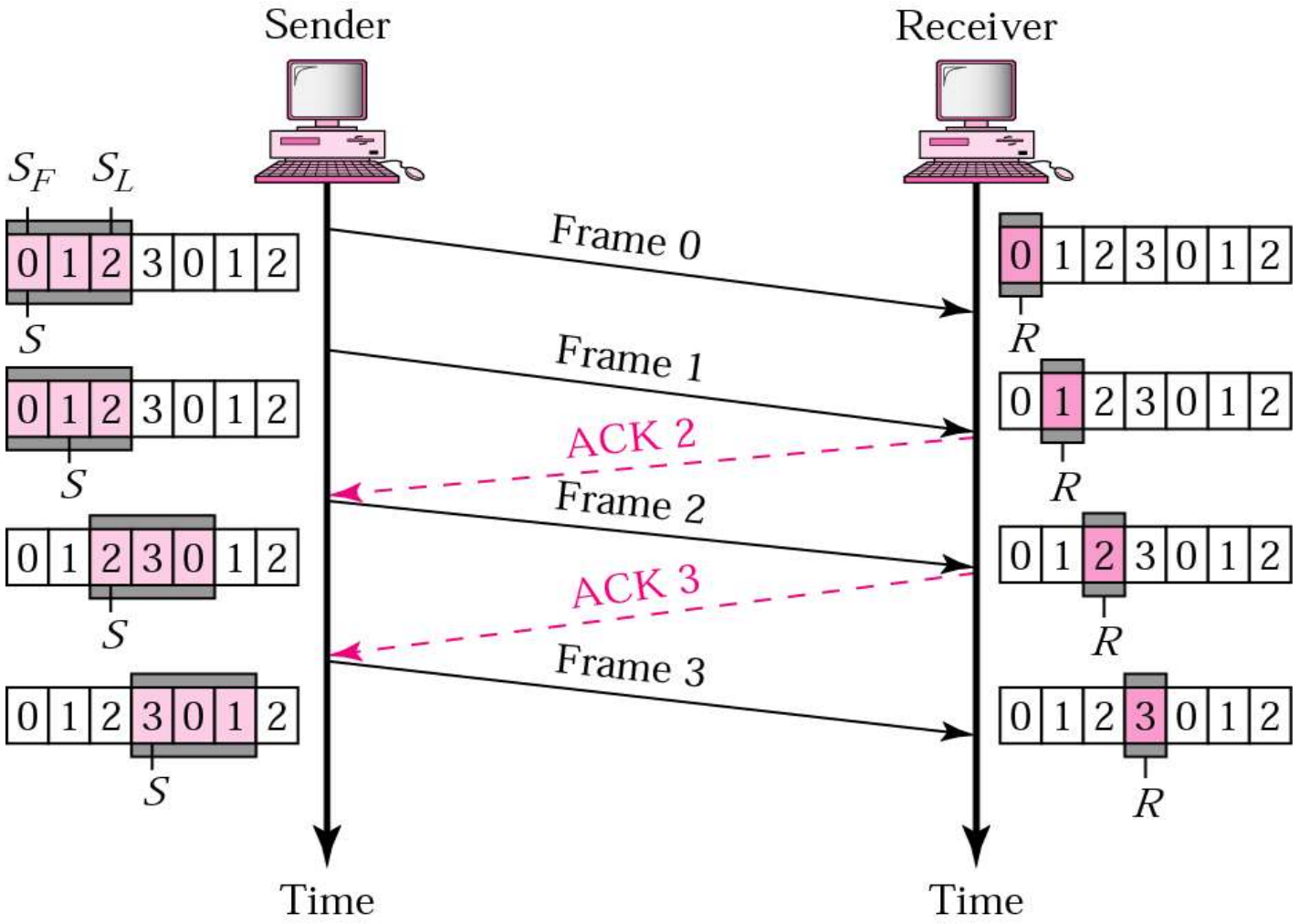
a. Receive window



b. Window after sliding

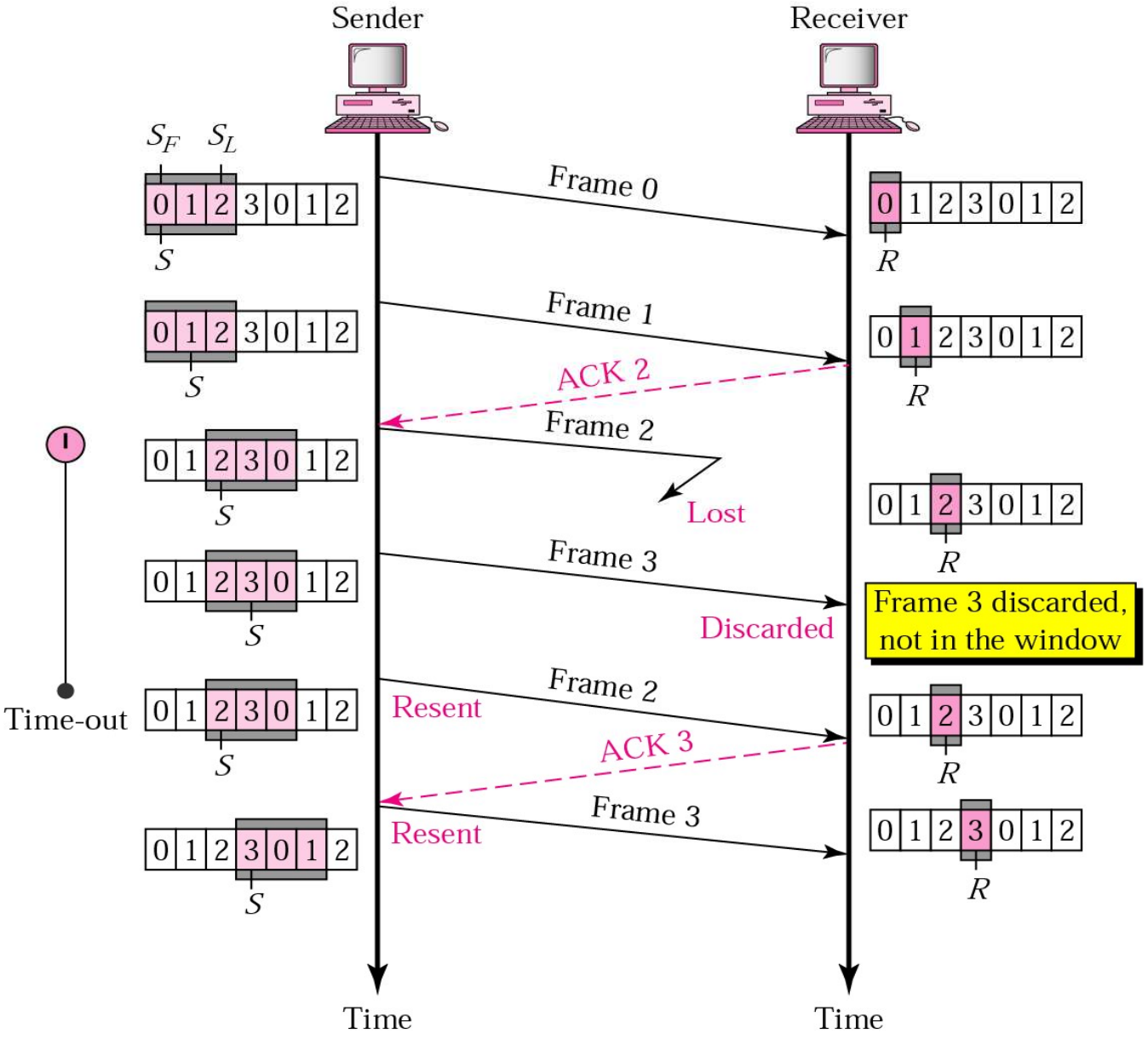
Once a frame is received without error and in order, the window will slides over to the next placeholder.

Go-Back-N ARQ, normal operation

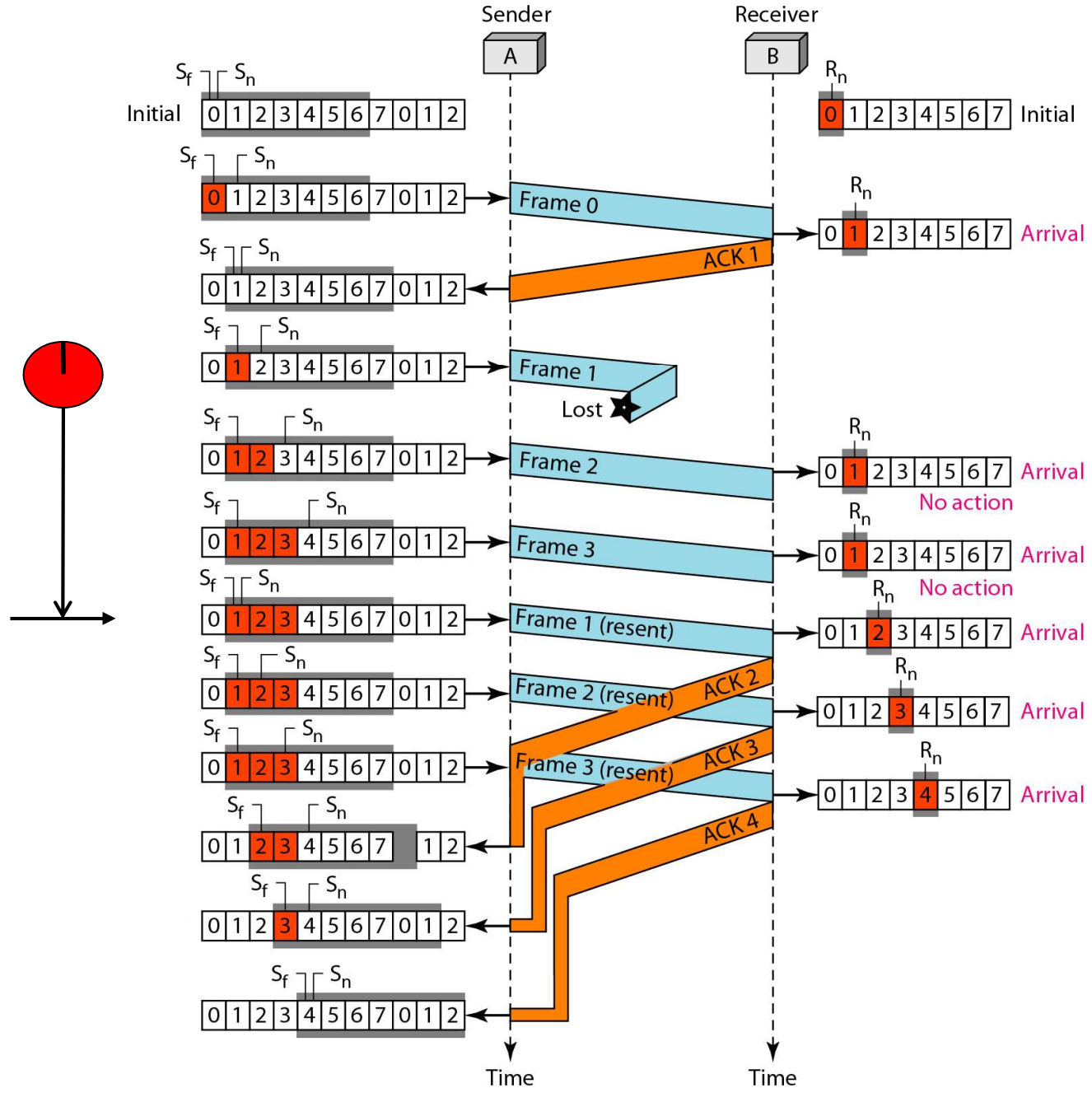


Sender window will slide to frame number (i) when (ACK i) is received.

Go-Back-N ARQ, lost frame

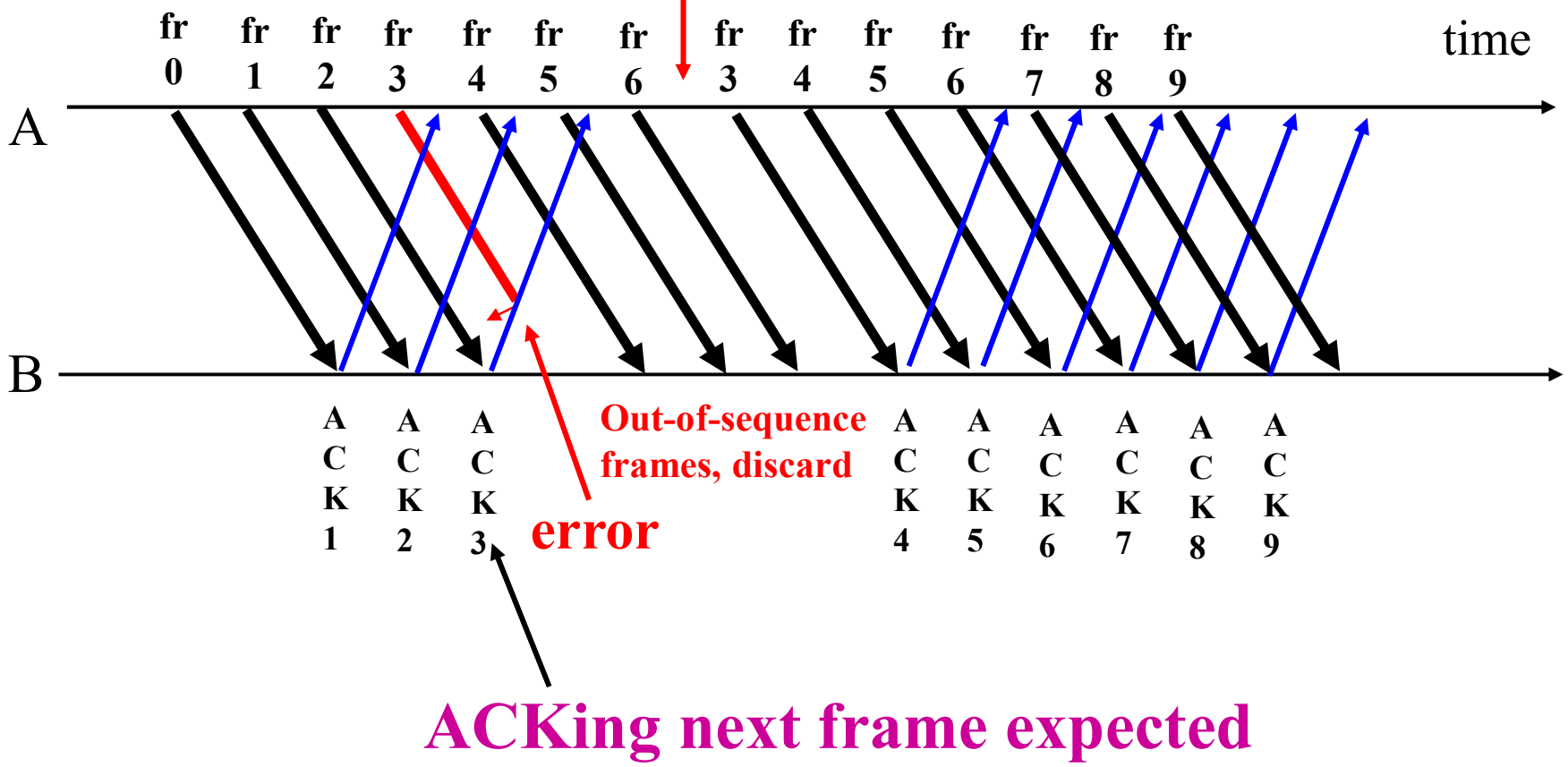


Timeout for frame 1 expired



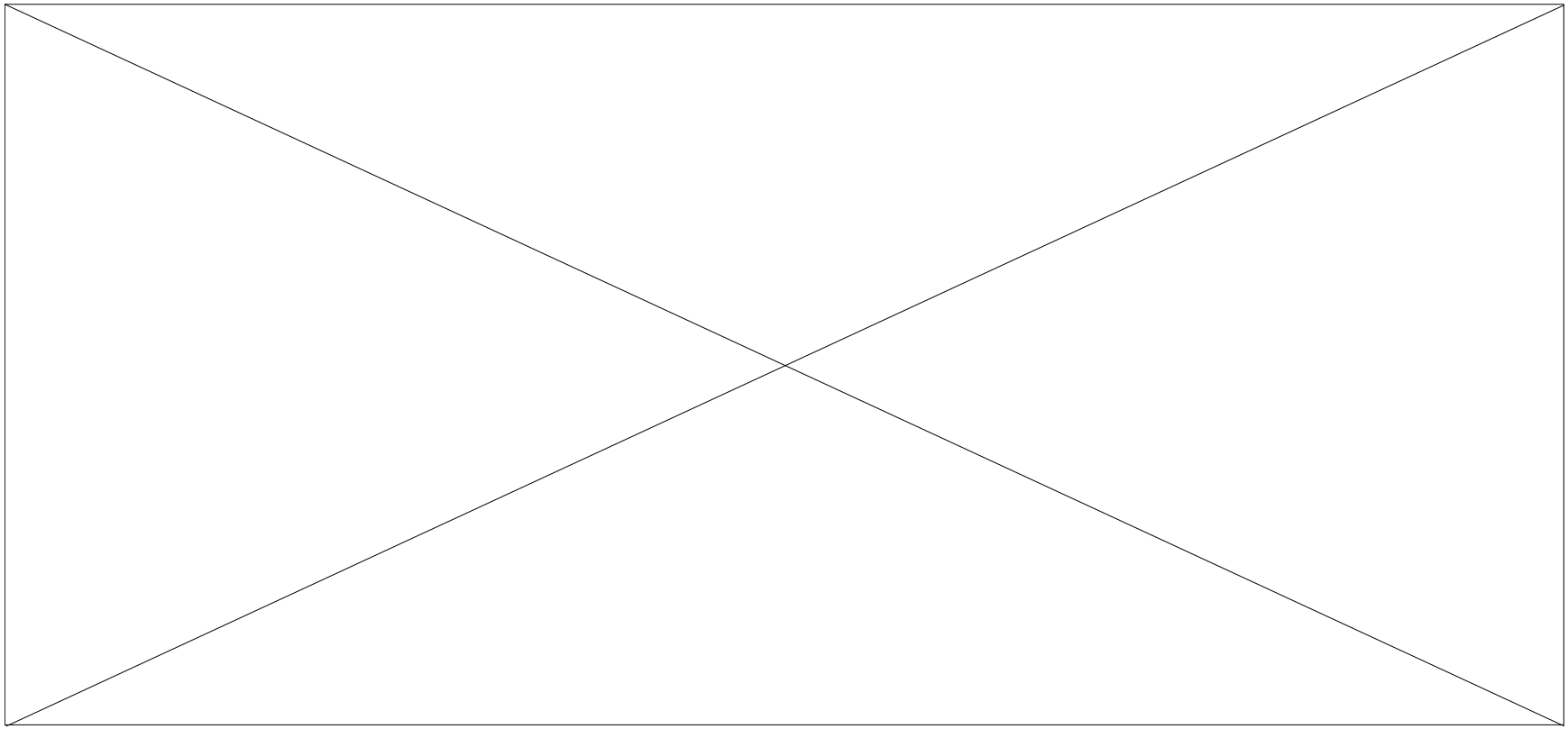
Go-Back-N ARQ

Go-Back-4: 4 frames are outstanding; so go back 4, retransmit frame 3,4,5,6.



Go-Back-N ARQ

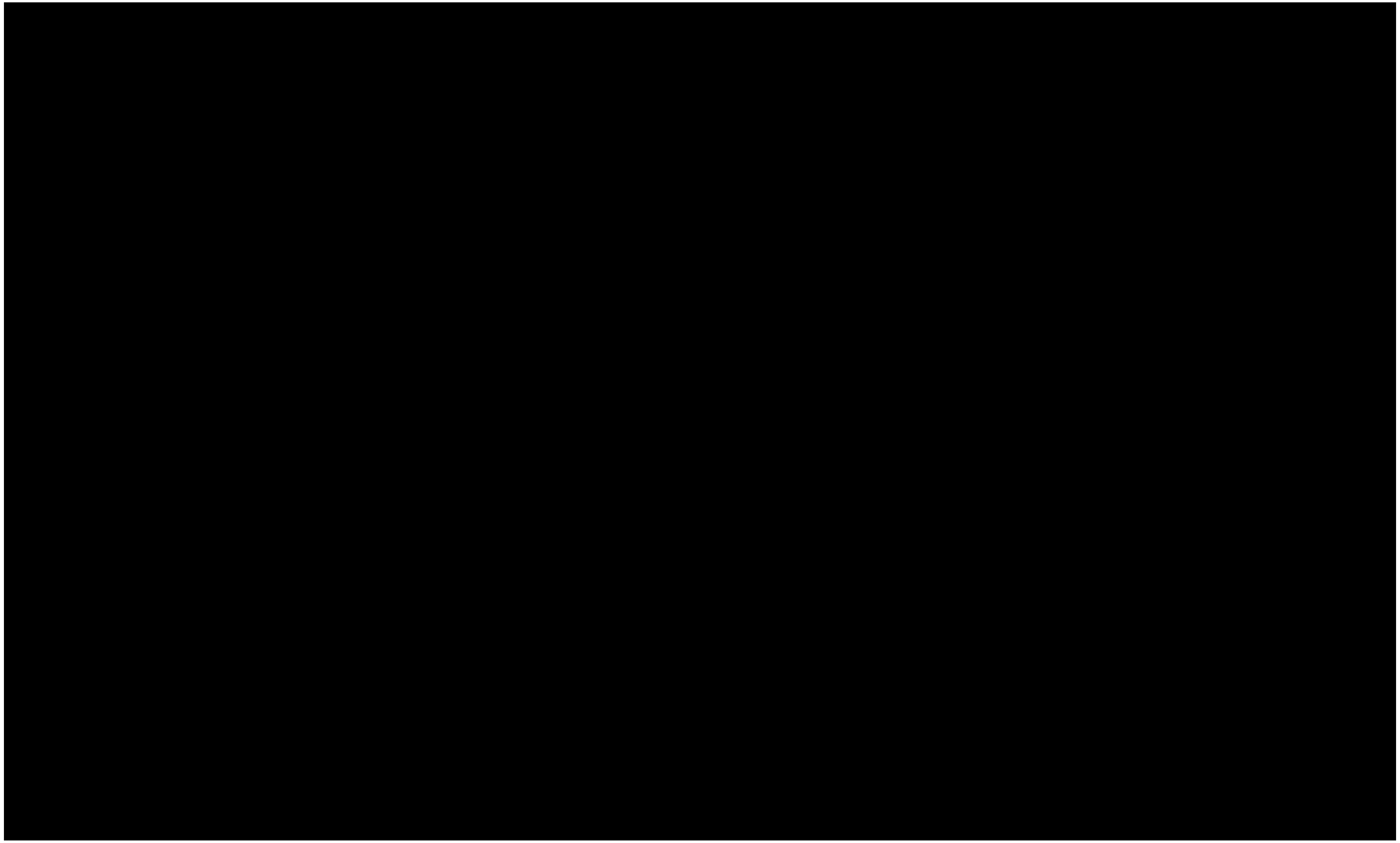
- Here is a simulation of Go-Back-N ARQ **without any errors.** This is pretty much just a sliding window. (Go-Back-N_ARQ.swf)



Go-Back-N ARQ

- Here is a simulation of Go-Back-N ARQ **with a damaged frame. The 2nd frame 5 gets damaged during transmission,** so it is ignored by the receiver.
- When frame 6 arrives, however, the receiver sender a REJ 5, telling the sender to start sending again at frame 5.
- The sender starts over at frame 5 and also retransmits frame 6 (Damaged_Frame_A_Go-Back-N_ARQ.swf)

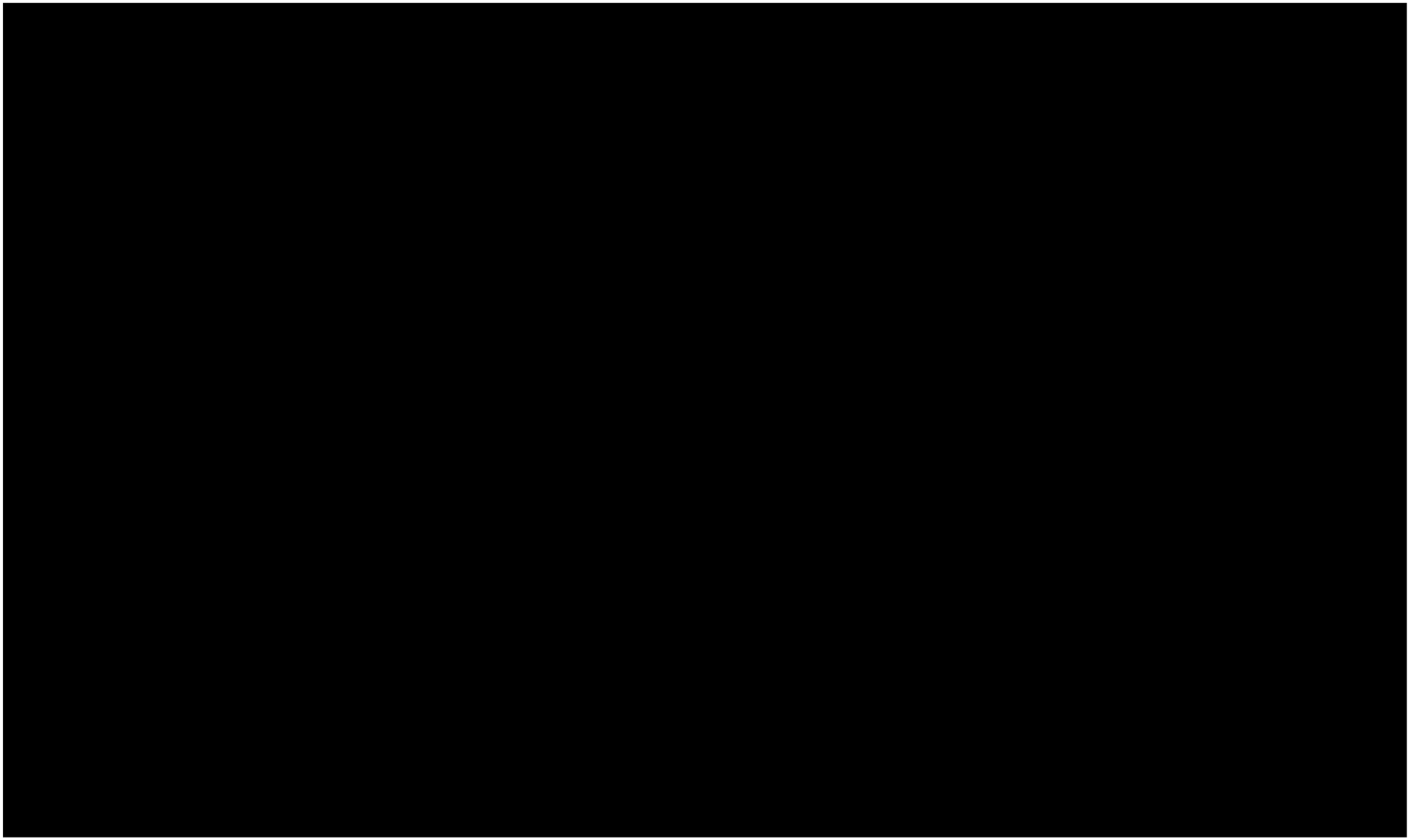
Go-Back-N ARQ



Go-Back-N ARQ

- Here is a simulation of Go-Back-N ARQ **with a damaged RR.** In this simulation, **the RR for the 2nd frame 4 is lost in transit or damaged.**
- But when the sender receive the next RR (for frame 5) indicating that the receiver is ready for frame 6, it is able to assume that frame 4 was received properly, since RRs are cumulative (Damaged_RR_A_Go-Back-N_ARQ.swf)

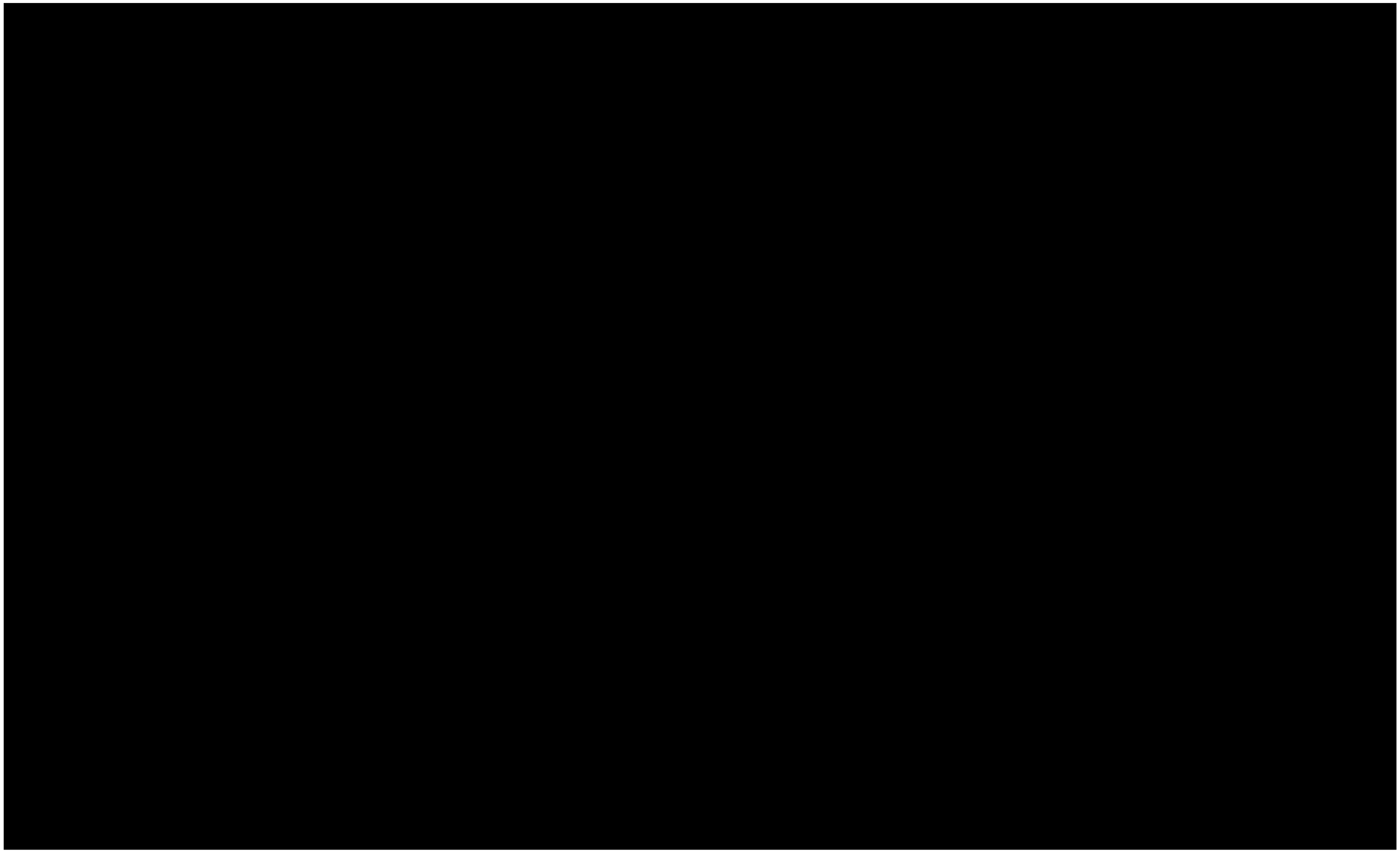
Go-Back-N ARQ



Go-Back-N ARQ

- Here is another simulation of Go-Back-N ARQ **with multiple damaged RRs.** In fact, **three in a row don't make it back to the sender.**
- After a predetermined amount of time, the timer goes off and sends an RR with a P-value of 1 (as in damaged frame simulation b) to tell the receiver to reply with the latest RR sent. The receiver replies with RR 1, indicating that it is ready for the next frame 1. The sender can open its sliding window, and discard the three frames in question from its buffer of sent frames. (Damaged_RR_B_Go-Back-N_ARQ.swf)

Go-Back-N ARQ



Go-Back-N ARQ

- Here is a simulation of Go-Back-N ARQ **with a damaged REJ. in this simulation, the 2nd frame 5 is damaged and ignored. When the sender sends frame 6, the receiver rejects it with a REJ 5 to tell the sender to start over at frame 5.** Unfortunately, **the REJ packet gets lost on its way back.**
- After a predetermined amount of time, the timer goes off and sends an RR with a P-value of 1 (as in damaged frame simulation b) to tell the receiver to reply with the latest RR sent. The receiver replies with RR 5, indicating that it still needs frame 5. The sender must retransmit frames 5 and 6. (Damaged_REJ_B_Go-Back-N_ARQ.swf)

Go-Back-N ARQ

Protocol. 5 Sliding Window Protocol Using Go Back N

```
/* Protocol 5 (pipelining) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike the previous protocols,
   the network layer is not assumed to have a new packet all the time. Instead, the
   network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7                       /* should be 2^n – 1 */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <=b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
        return(true);
    else
        return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data frame. */
  frame s;                              /* scratch variable */

  s.info = buffer[frame_nr];            /* insert packet into frame */
  s.seq = frame_nr;                     /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);                /* transmit the frame */
  start_timer(frame_nr);                /* start the timer running */
}
```

Continued →

Protocol. 5 Sliding Window Protocol Using Go Back N

```
void protocol5(void)
{
  seq_nr next_frame_to_send;          /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;              /* next frame expected on inbound stream */
  frame r;                            /* scratch variable */
  packet buffer[MAX_SEQ + 1];         /* buffers for the outbound stream */
  seq_nr nbuffered;                   /* # output buffers currently in use */
  seq_nr i;                           /* used to index into the buffer array */
  event_type event;

  enable_network_layer();             /* allow network_layer_ready events */
  ack_expected = 0;                   /* next ack expected inbound */
  next_frame_to_send = 0;             /* next frame going out */
  frame_expected = 0;                 /* number of frame expected inbound */
  nbuffered = 0;                      /* initially no packets are buffered */
```

Continued →

Protocol. 5 Sliding Window Protocol Using Go Back N

```
while (true) {
  wait_for_event(&event);              /* four possibilities: see event_type above */

  switch(event) {
    case network_layer_ready:          /* the network layer has a packet to send */
         /* Accept, save, and transmit a new frame. */
         from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
         nbuffered = nbuffered + 1;    /* expand the sender's window */
         send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
         inc(next_frame_to_send);      /* advance sender's upper window edge */
         break;

    case frame_arrival:                /* a data or control frame has arrived */
         from_physical_layer(&r);      /* get incoming frame from physical layer */

         if (r.seq == frame_expected) {
              /* Frames are accepted only in order. */
              to_network_layer(&r.info);  /* pass packet to network layer */
              inc(frame_expected);   /* advance lower edge of receiver's window */
          }
```

Continued →

Protocol. 5 Sliding Window Protocol Using Go Back N

```
            /* Ack n implies n – 1, n – 2, etc.  Check for this. */
            while (between(ack_expected, r.ack, next_frame_to_send)) {
                /* Handle piggybacked ack. */
                nbuffered = nbuffered    1; /* one frame fewer buffered */
                stop_timer(ack_expected); /* frame arrived intact; stop timer */
                inc(ack_expected);      /* contract sender's window */
            }
            break;

        case cksum_err: break;              /* just ignore bad frames */

        case timeout:                       /* trouble; retransmit all outstanding frames */
            next_frame_to_send = ack_expected;    /* start retransmitting here */
            for (i = 1; i <= nbuffered; i++) {
                send_data(next_frame_to_send, frame_expected, buffer);/* resend 1 frame */
                inc(next_frame_to_send);  /* prepare to send the next one */
            }

    }

    if (nbuffered < MAX_SEQ)
        enable_network_layer();
    else
        disable_network_layer();
  }
}
```

Go-Back-N ARQ - Window Size

Note

In Go-Back-N ARQ,
the maximum size of the send window is 2^n-1,
the size of the receiver window is always 1.
(where n is the sequence field size in)

Important points about Go-Back-N ARQ

- In Go-Back-N, N determines the sender's window size, and the size of the receiver's window is always 1.
- It does not consider the corrupted frames and simply discards them.
- It does not accept the frames which are out of order and discards them.
- If the sender does not receive the acknowledgment, it leads to the retransmission of all the current window frames.

Performance of Go-Back-N ARQ

- Better than Stop and wait.
- Does not allow error free but out of order frames to be accepted by the receiver.
- Waste a lot of bandwidth if the error rate is high.

Go-Back-N ARQ Example 1

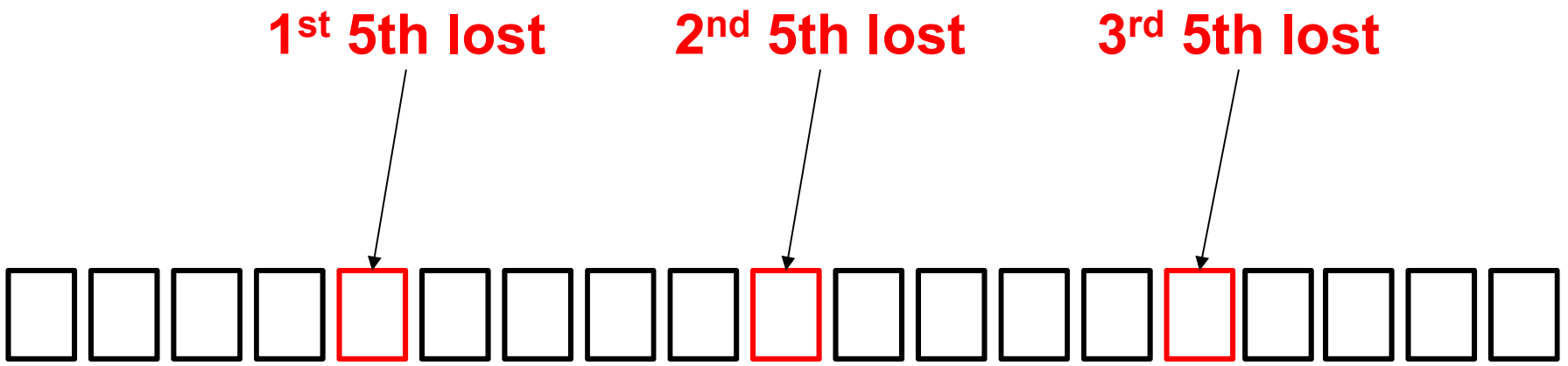
- Station A needs to send a message consisting of a sequence of frames to Station B using a sliding window (window size 7) and go-back-n error control strategy (Go-Back-7). All frames are ready and immediately available for transmission.
- If 5th packet that A transmits gets lost (but no acks from B ever get lost), then how many frames that A will retransmit and how many new frames that A will transmit?
- Solution:

Go-Back-N ARQ Example 2

- Station A needs to send a message consisting of 9 packets to Station B using a sliding window (window size 3) and go-back-n error control strategy (Go-Back-3). All packets are ready and immediately available for transmission.
- If every 5th packet that A transmits gets lost (but no acks from B ever get lost), then what is the number of packets that A will transmit for sending the message to B?

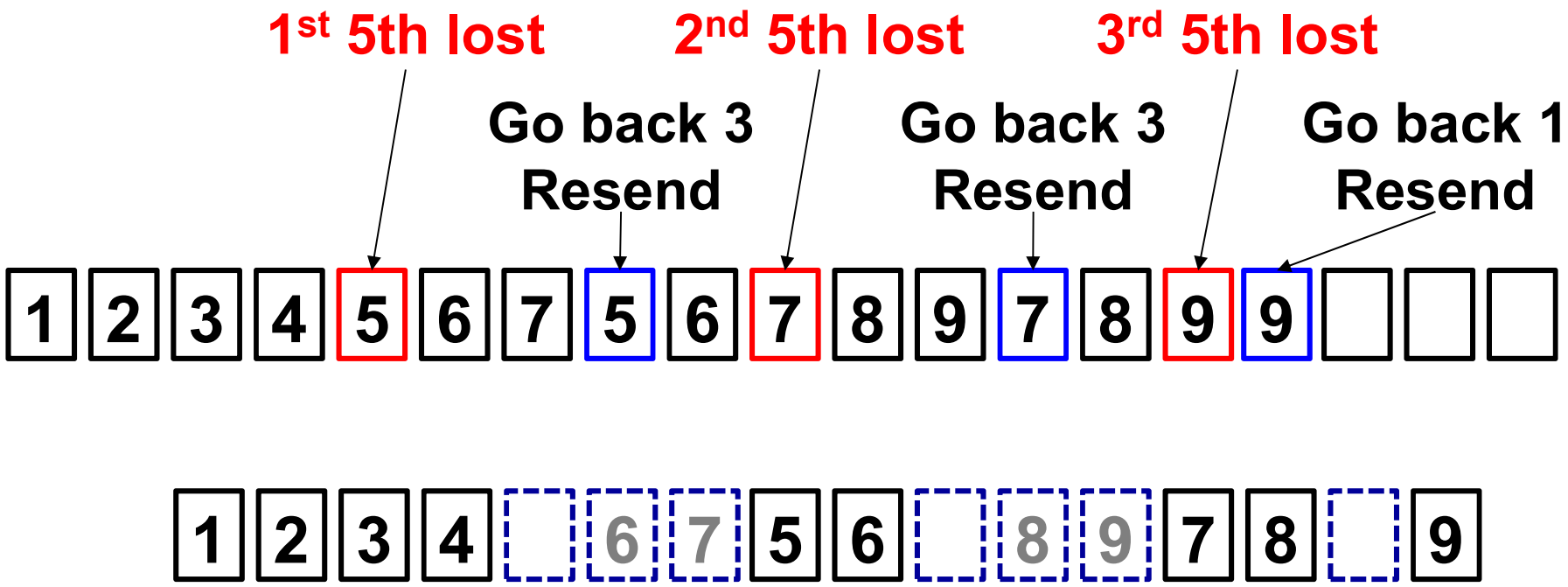
Go-Back-N ARQ Example

- **Solution：**



Go-Back-N ARQ Example

- **Solution：**



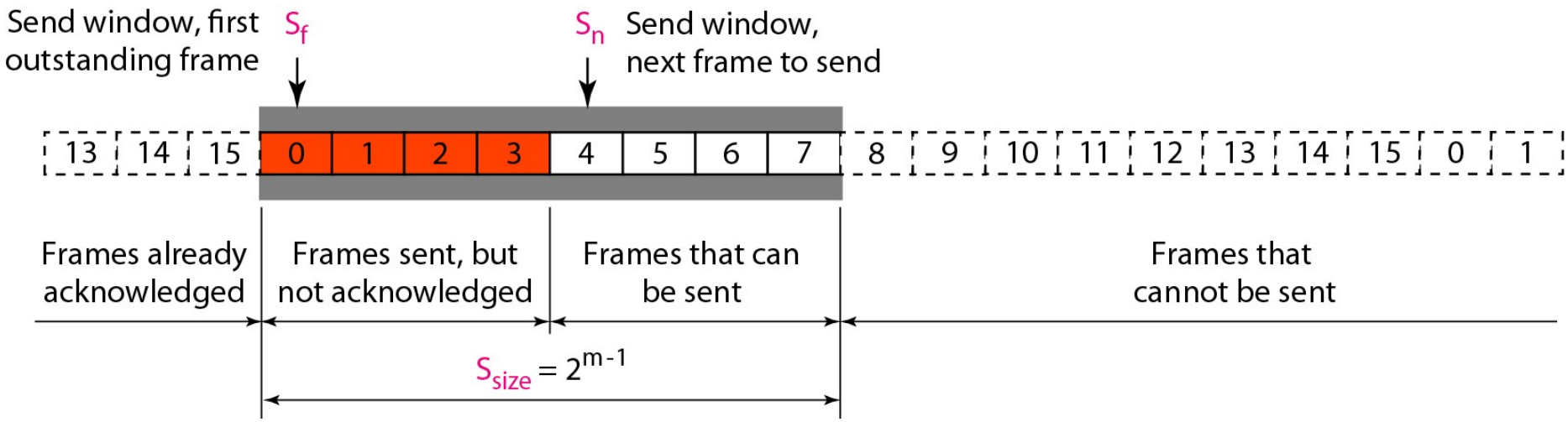
Correct answer is ??

Selective Repeat ARQ

- Both sender and receiver maintain a window of acceptable sequence numbers.
- **Sending window size > 1:** Sender can send multiple frames while waiting for ACK.
- **Receiving window size > 1:** Any frame within the window may be accepted and buffered.

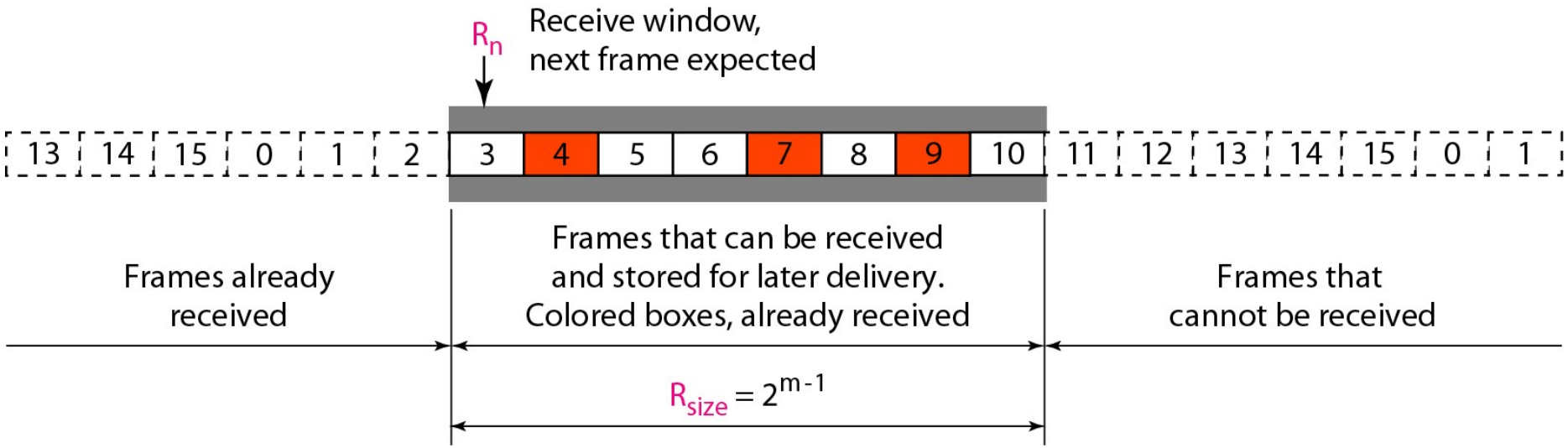
Only the specific damaged or lost frame is retransmitted.

Selective Repeat ARQ



Send window for Selective Repeat ARQ

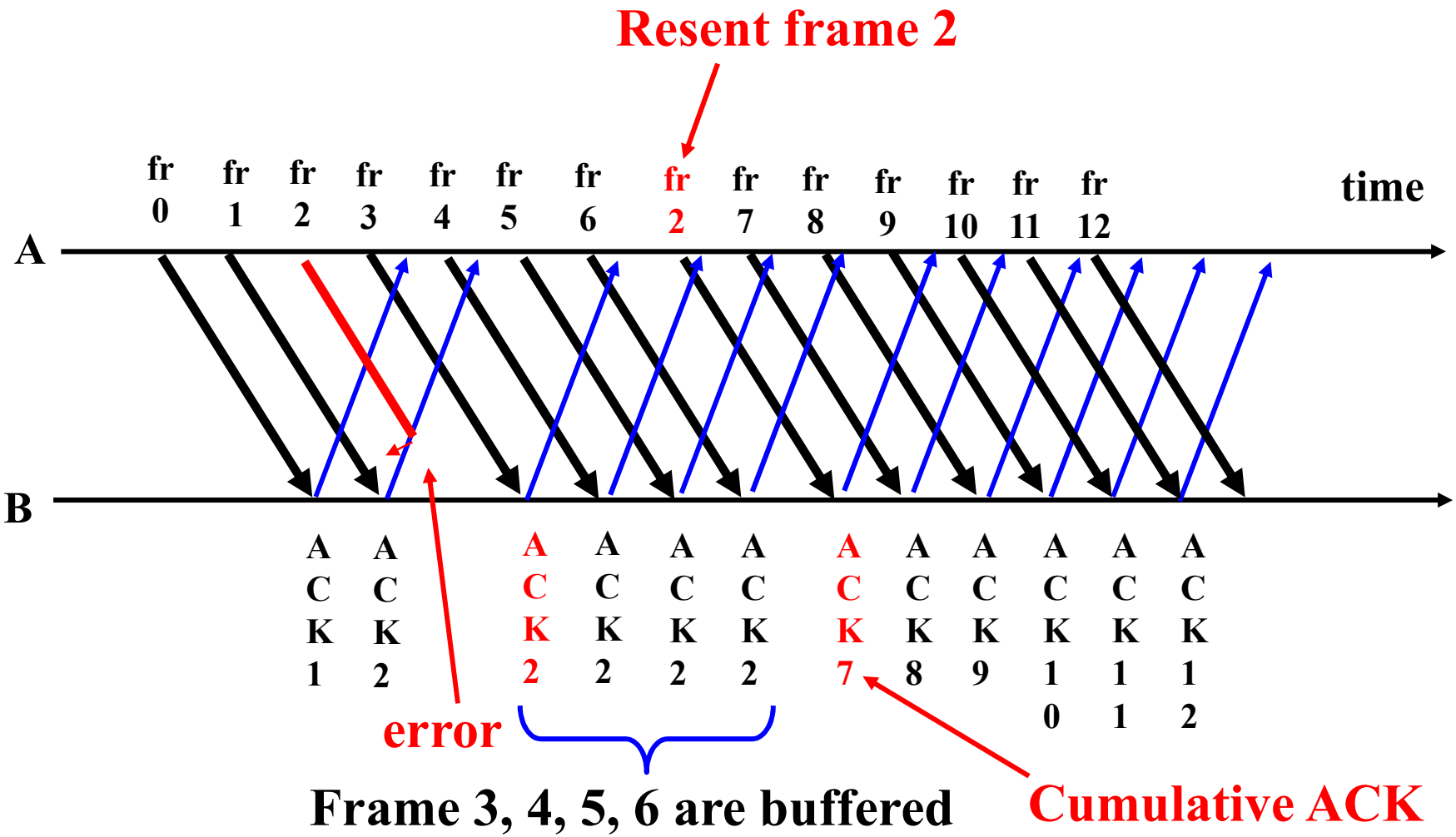
Selective Repeat ARQ



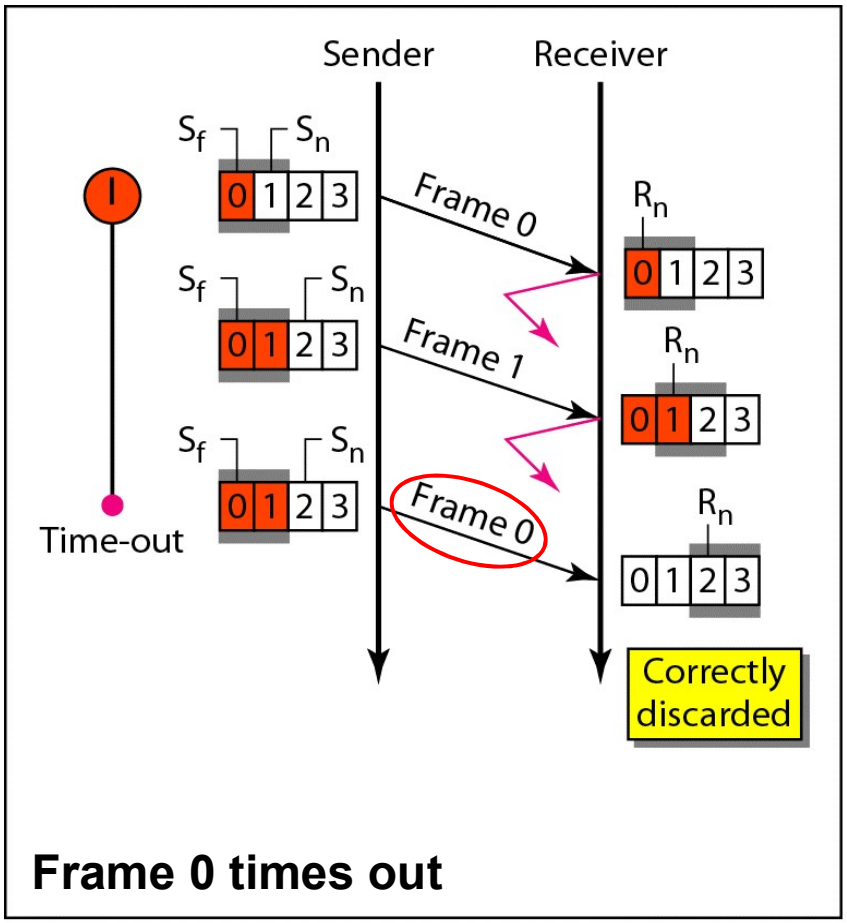
Receive window for Selective Repeat ARQ

- **Receiver window is shifted to position (i) only when receiver sends ACK i**
- **ACK i can be sent if all frames up to (i-1) have been received without error**

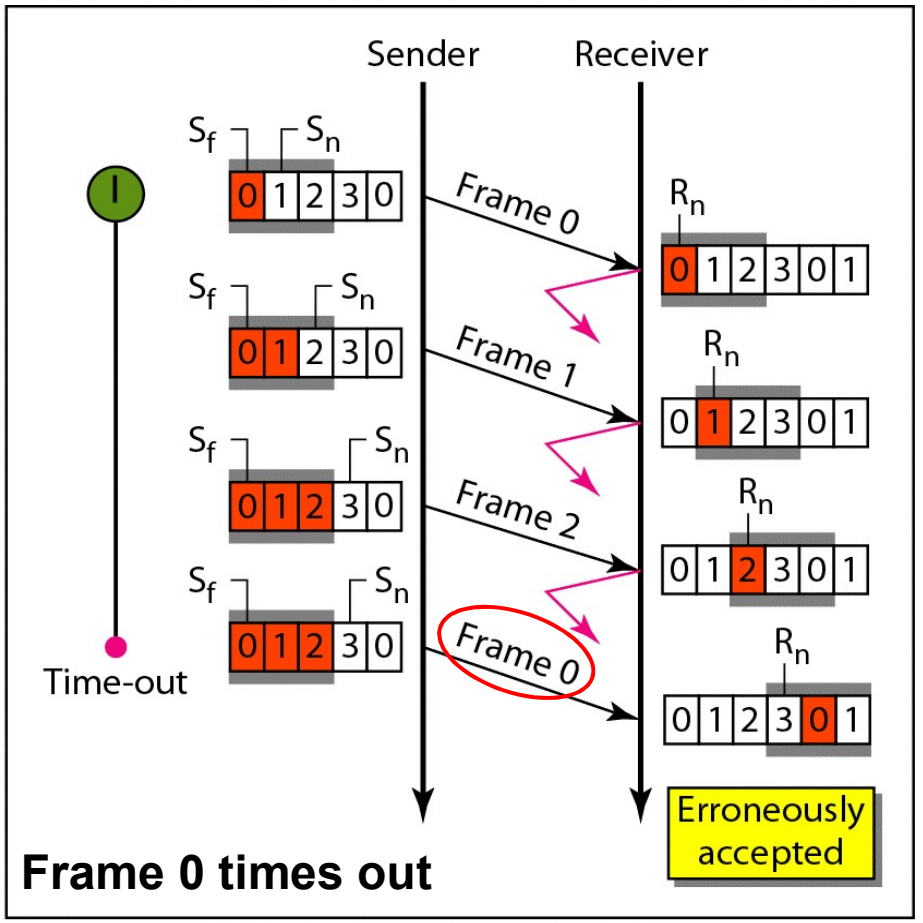
Selective Repeat ARQ



Selective Repeat ARQ - Window Size



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



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

Selective Repeat ARQ - Window Size

Note

In Selective Repeat ARQ,
the maximum size of the send window is
2^{n-1} .
(where n is the sequence field size in bits)

Performance

- **More efficient than the other two protocols.**
- **The receiver and transmitter processing logic is more complex**
  - Receiver **must be able to** reinsert **the retransmitted (lost, delayed, damaged) frame in the proper sequence after it arrives.**
  - The sender **should be able to** send out of order **frame when requested.**
  - Needs **more memory** than Go-Back-N ARQ at the receiver side**. The receiver memory is large if the window size is large**

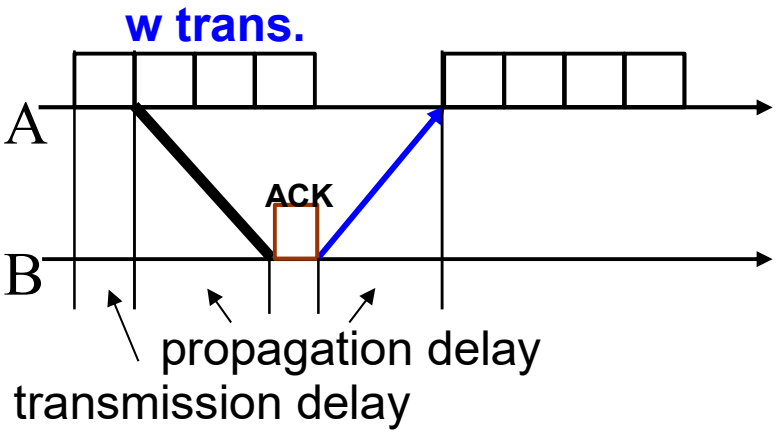
Pipelining

- **Pipelining:** beginning a task before the previous task has ended.
- **No pipelining** in Stop-And-Wait ARQ.
- **Pipelining** in Go-Back-N ARQ and Selective repeat ARQ .

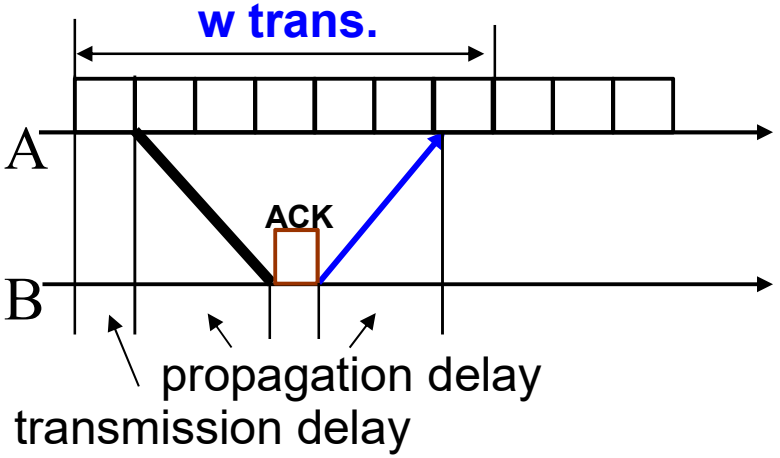
Pipelining increases the efficiency because it keeps the line busy instead of waiting

Example 1

- **Calculate the efficiency (utilization) of Go-Back-N (or Selective Repeat) where the window size is w, there is no transmission error and ACK frames are vary short.**



Efficiency < 100%



Efficiency = 100%

$$\text{Efficiency} = min\left\{\frac{\text{w Trans}}{\text{Trans} + \text{ACK} + 2\text{Prop}}, 1\right\} = min\left\{\frac{\text{w}}{1 + 2\alpha}, 1\right\}$$

Minimum seq numbers: GBN = w + 1, SR = w + w

Example 2

- A channel has a data rate of R bps and a propagation delay of t sec/km. The distance between the sending and receiving nodes is L km. Nodes exchange fixed-size frames of B bits. Find a formula that gives the minimum sequence field size in bits of the frame as a function of R, t, B and L. Assume that ACK frames are negligible in size, the processing at the nodes is instantaneous, and that maximum utilization is required.

Example 2

Solution

- Maximum utilization implies the max window size = $1 + 2a$ ($a = T_{prop}/T_{trans}$)
- $T_{prop} = Lt$
- $T_{trans} = B/R$
- Hence $W = 1 + 2(Lt)/(B/R) = 1 + 2RLt/B$
- Now $W=2^n - 1$, so $n=\log_2(W+1)$
- n = integer greater than $\log_2(2+RLt/B)$

Example 3

- Let the window size be W.
- (1) How many timers are required at the sender side of GBN and how many timers at the sender side of SR?
- (2) How much buffer is required at the receiver side of GBN and how much at the receiver side of SR?
- (3) Suppose W is an even number. If packet W/2 is lost, what packet(s) will be retransmitted in GBN and what packet(s) will be retransmitted in SR?
- (4) Suppose W is an even number. If packet W/2 is lost, how much time does it take to have the first W packets arriving at the receiver in GBN and how much time in SR?

Example 4

- Consider an error-free 64-kbps satellite channel used to send 512-byte data frames in one direction with very short acknowledgements coming back the other way. What is the maximum throughput for window sizes of 1, 7, 15 and 127?

Example 4

Solution

- A 512 byte (4096 bits) data frame has a duration of 4096/64000 seconds – that is 64 msec.
- Assume that the satellite is at 36000 km distance. This leads to roundtrip propagation time of 240 msec. We should also add 64 msec to this to account for transmission time.
- Hence with a window size of 1, 4096 bits can be sent every 240 + 64 = 304 msec. This equates to the throughput of 4096 bits/304 msec or 13.5 kbps.
- For a window size greater than 5, the full 64 kbps is used.

Chapter 3: Roadmap

- 3.1 Introduction and services
- 3.2 Framing
- 3.3 Error Detection and Correction
- 3.4 Stop-and-Wait Protocols
- 3.5 Sliding Window Protocols
- 3.6 HDLC and PPP

Examples of Data Link Protocol

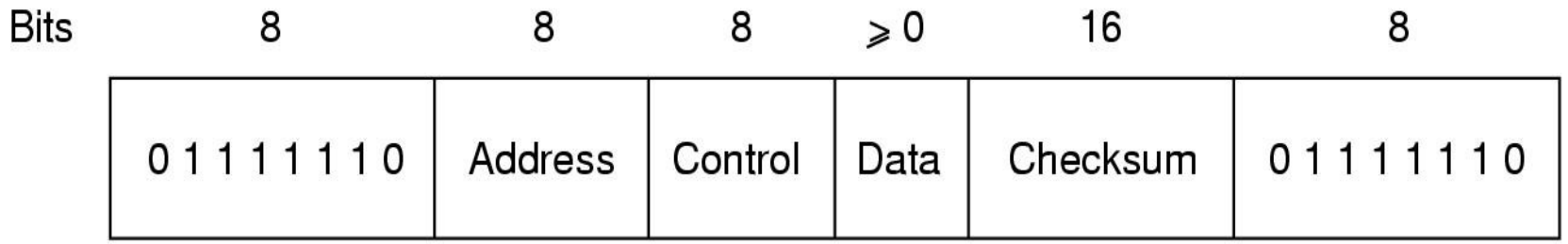
- High-Level Data Link Control (HDLC)
  - ISO standard
- Point-to-Point Protocol (PPP)
  - The Data Link Layer in the Internet

High-Level Data Link Control

- **HDLC: (ISO 33009, ISO 4335)**
  - Modified from SDLC (IBM)
  - A pretty old, but widely used protocol for point-to-point connections. bit-oriented, bit stuffing.
  - subsets:
    - Link Access Procedure for D Channel (LAPD)
    - Advanced Data Communication Control Procedure (ADCCP)
    - Link Access Procedure (LAP).

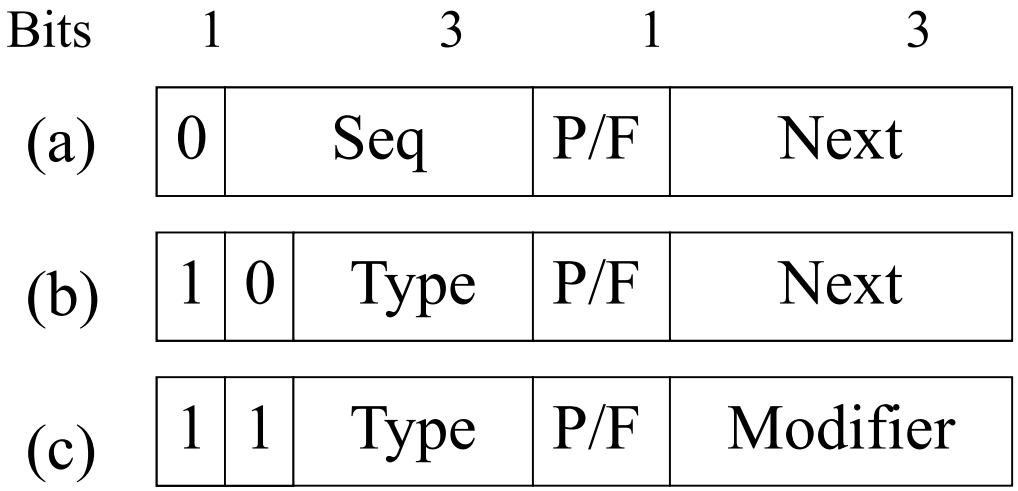
High-Level Data Link Control

Frame Format



- Flag: delimit frame at both ends. 01111110
- Address: identify the frame receiver
- Control: specify different frame types
- FCS: frame check sequence (error detecting code)

High-Level Data Link Control



- The protocol uses a sliding window, with 3-bit sequence number.
- Up to seven unacknowledged frames may be outstanding at any instant.

Control field of

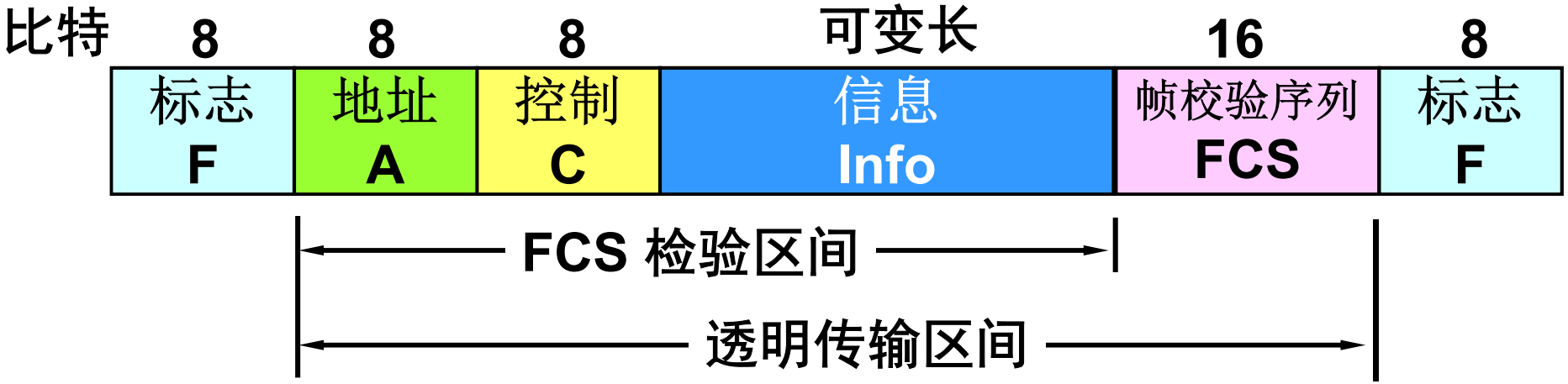
(a) An information frame.

(b) A supervisory frame.

(c) An unnumbered frame.

P-polling data
F-finished polling.
(a)-nACK (reject)
(b)-RNR
(c)-Selective reject-retransmit specified

High-Level Data Link Control



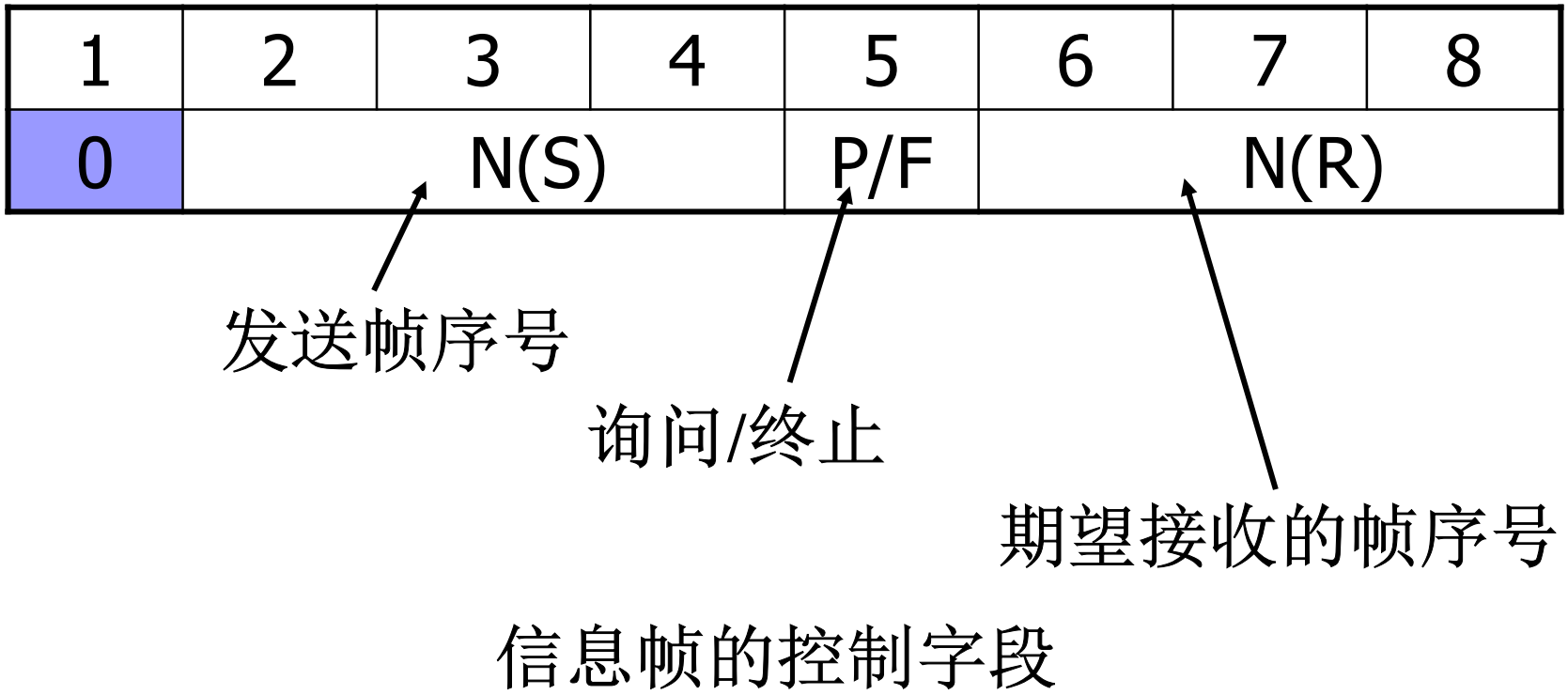
- 标志字段F：8位，01111110 (7Eh)
- 地址字段A：8位，站地址
  - 全1：广播
  - 全0：无效

High-Level Data Link Control

- 控制字段C：8位，根据最前面两位的取值，HDLC帧分成三类：
  - 信息帧（I）
  - 监督帧（S）
  - 无编号帧（U）
- 信息字段Info：若干8位，发送的实际数据
- 帧校验序列FCS：16位、CRC校验，生成多项式：$x^{16}+x^{12}+x^5+1$，即CRC-CCITT

High-Level Data Link Control

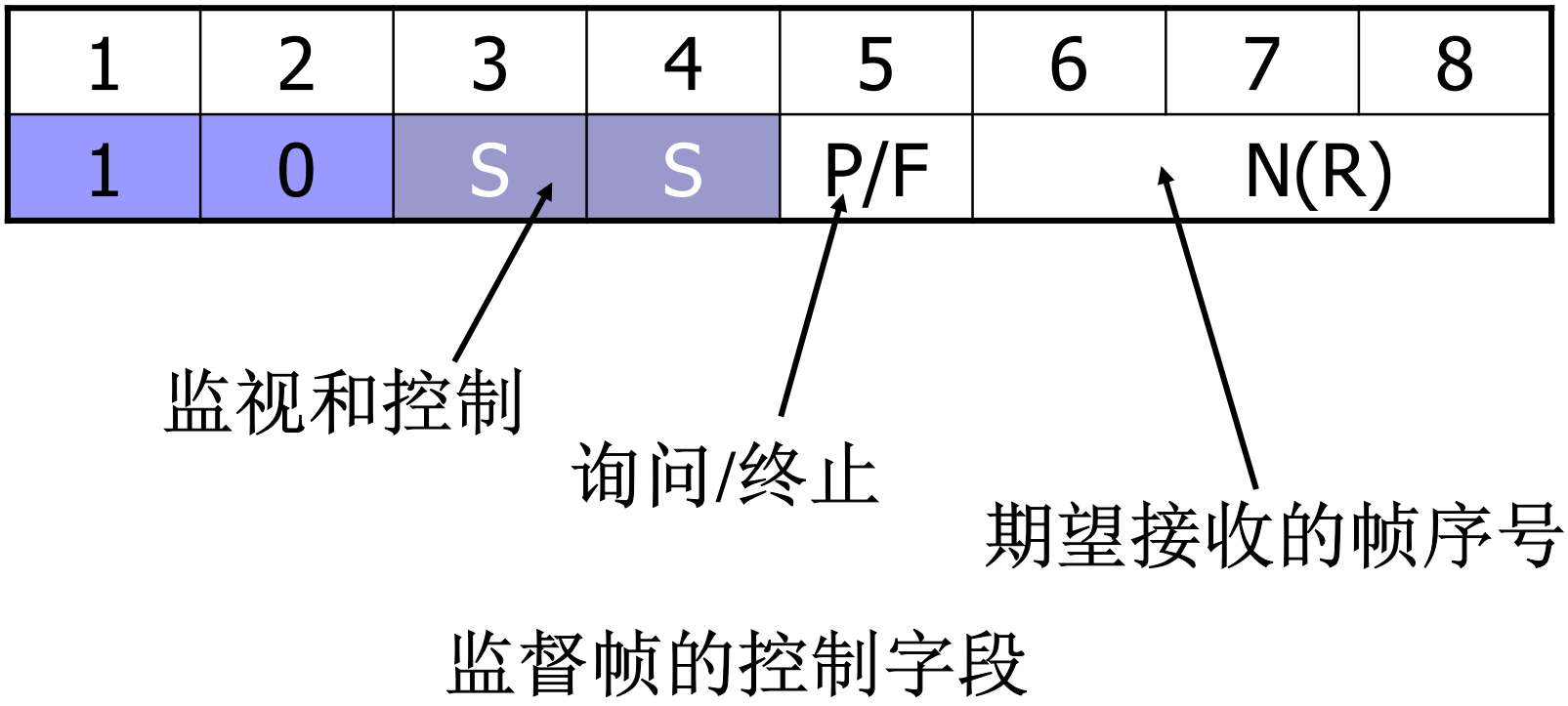
- 信息帧的控制字段



信息帧的控制字段

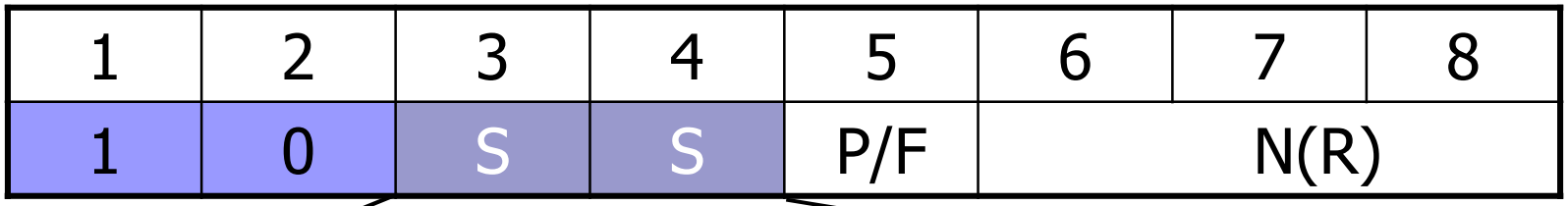
High-Level Data Link Control

- 监督帧的控制字段



监督帧的控制字段

High-Level Data Link Control



SS=00（RR）：表示准备接收下一帧，确认序号为N(R)-1及其以前的帧；

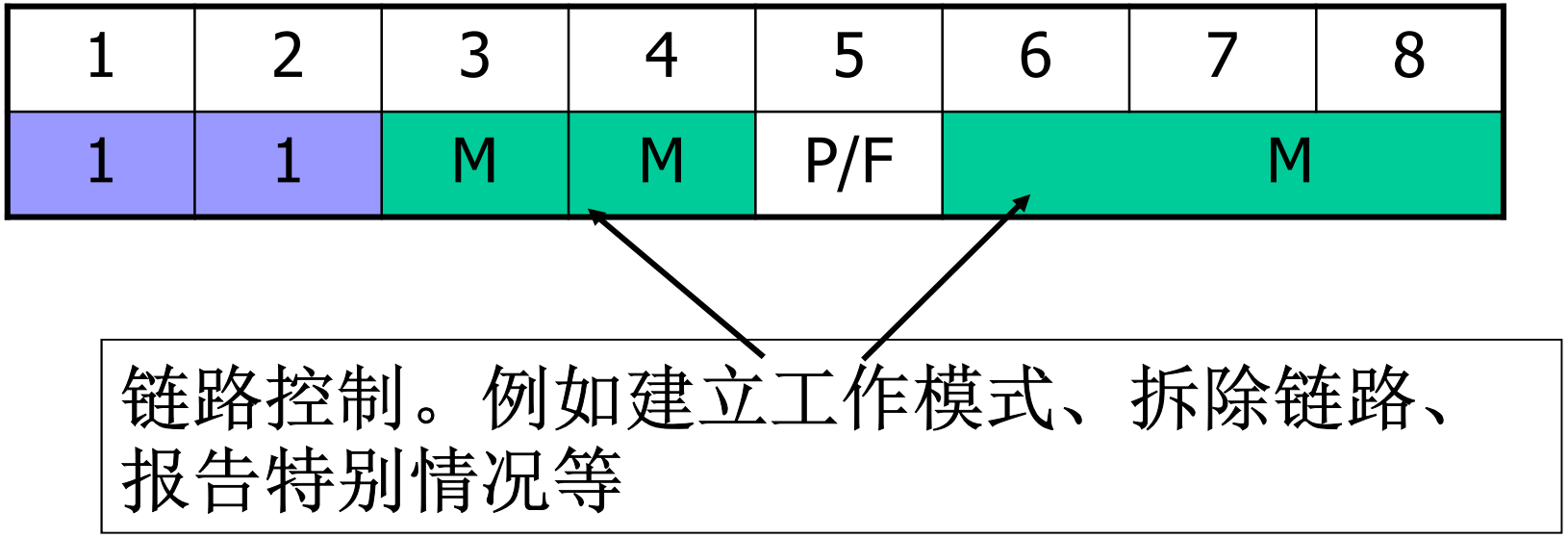
SS=01（REJ）：表示拒绝N(R)开始的所有帧，但确认N(R)-1及其以前的各帧；

SS=10（RNR）：表示暂停接收下一帧，确认N(R)-1及其以前的各帧；

SS=11（SREJ）：表示只拒绝N(R)帧但确认N(R)-1及其以前的各帧

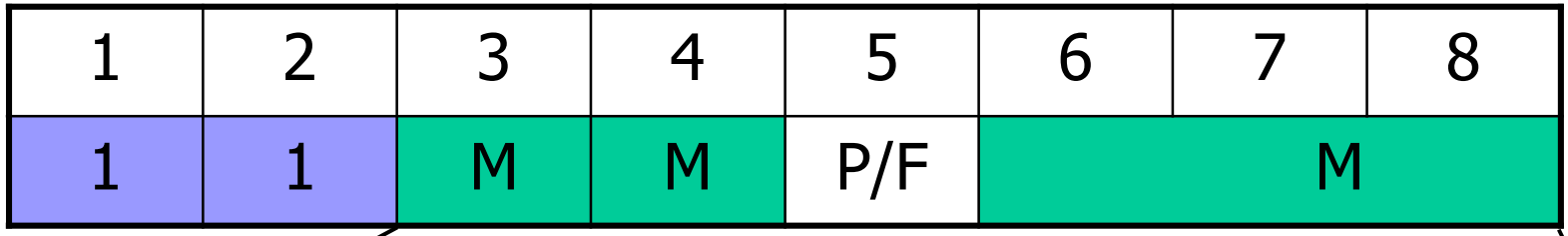
High-Level Data Link Control

- 无编号帧的控制字段



无编号帧的控制字段

High-Level Data Link Control



例如：

M=11100(SABM)：某一复合站置本次链路为异步平衡模式；

M=00010(DISC)：主站请求释放(拆除)本次链路；

M=00110(UA)：次站对主站命令的确认

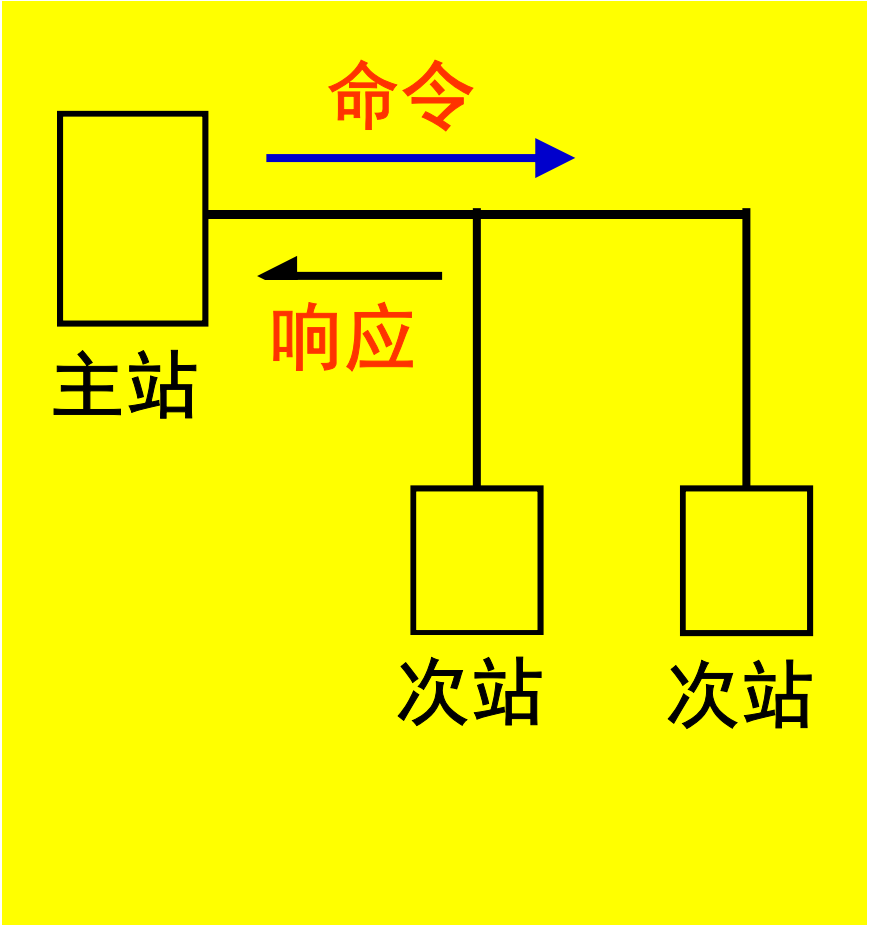
M=10001(CMDR)：次站对主站命令的否认

High-Level Data Link Control

- 一些术语
  - 主站(Primary Station)
    - 控制整个链路的工作，可以发出命令确定和改变链路的状态，包括确定次站、组织数据传输和链路恢复等
  - 次站(Secondary Station，从站)
    - 受主站控制，只能发出响应的站
    - 主站与每个次站都维持一条逻辑链路
  - 组合站(Combined Station，复合站)
    - 兼有主/次站功能的站

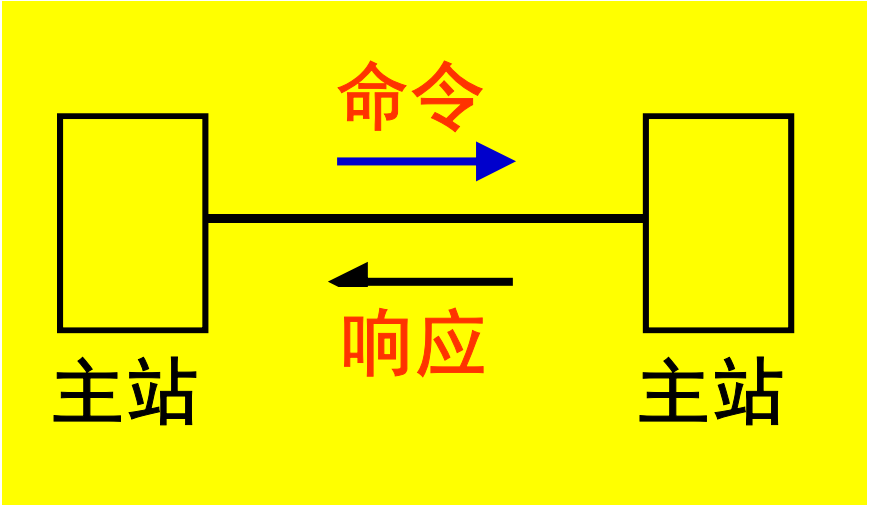
High-Level Data Link Control

- 一些术语（续）
  - 非平衡结构
    - 由一个主站和一个或多个次站组成
    - 适用于点-点、点-多点操作



High-Level Data Link Control

- 一些术语（续）
  - 平衡结构
    - 由两个组合站组成
    - 适用于点-点操作



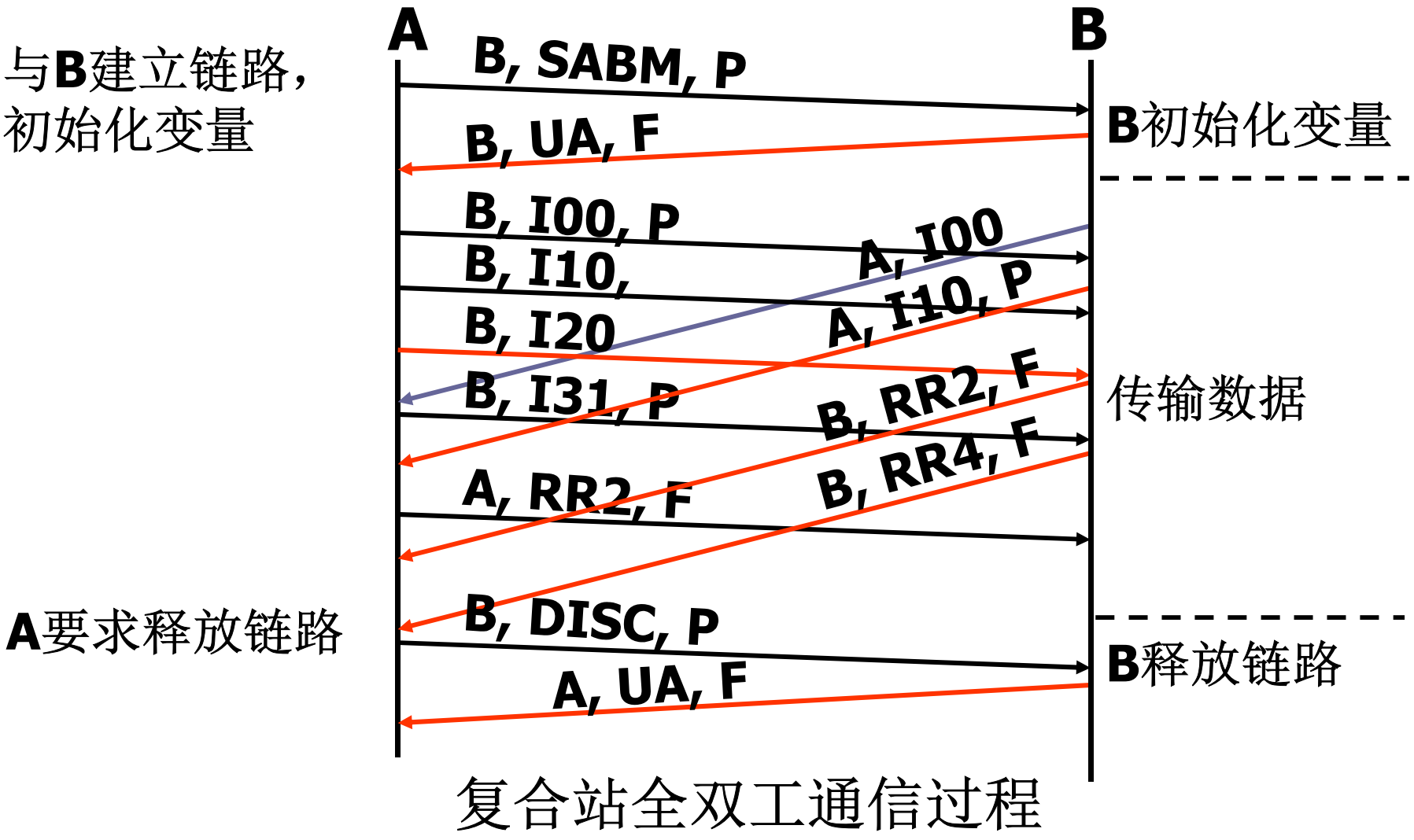
High-Level Data Link Control

- 流量控制
  - 滑动窗口协议
  - 在信息帧中用N(S)和N(R)来表示当前窗口的情况

High-Level Data Link Control

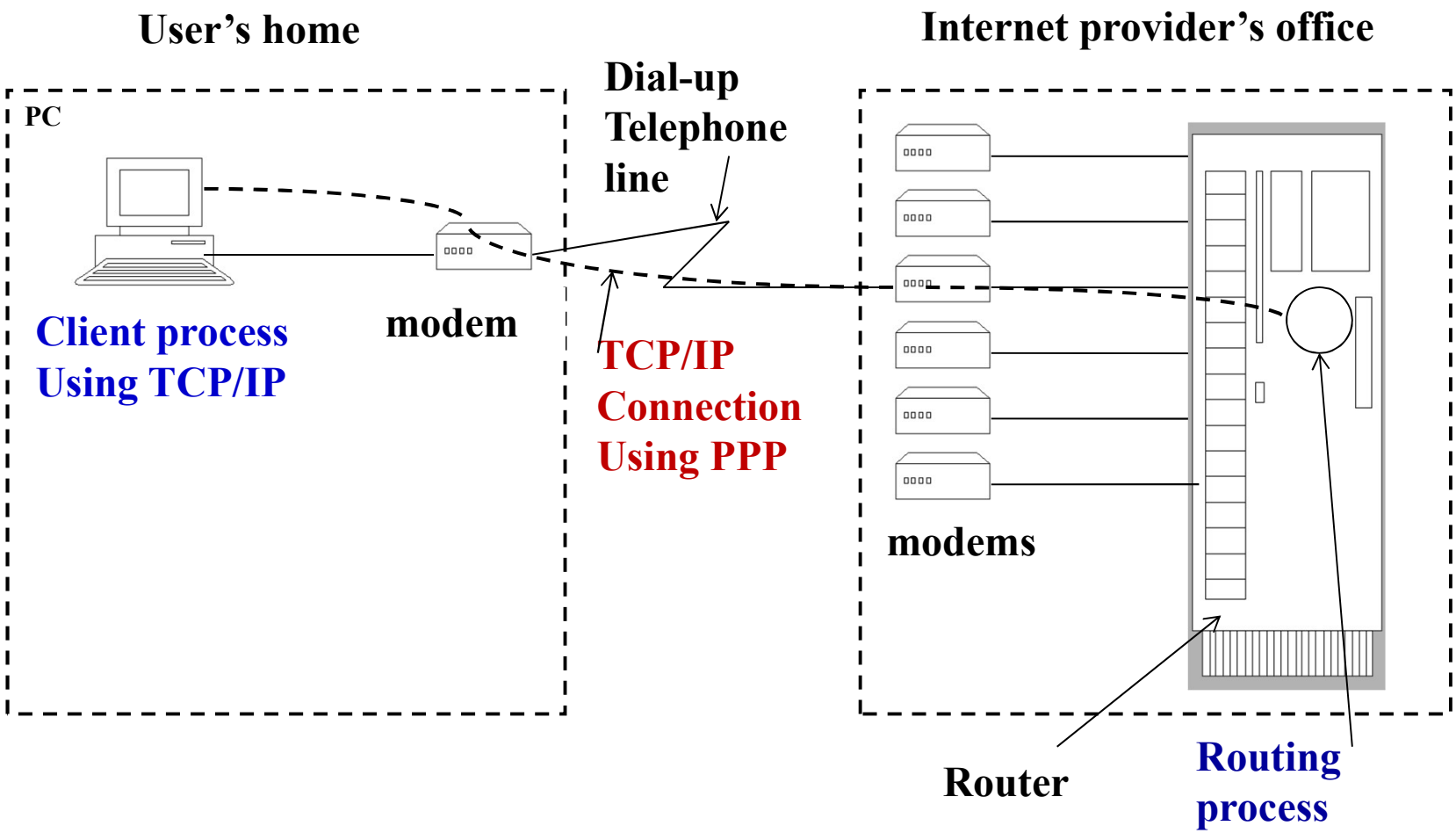
- 超时重发
  - 为了防止发送方无期限地等待接收方的确认，收发双方均设置计时器
  - 发送方在一定的时间内未收到接收方传来的确认，表示传输有故障，准备重发所有未被确认的帧

High-Level Data Link Control



复合站全双工通信过程

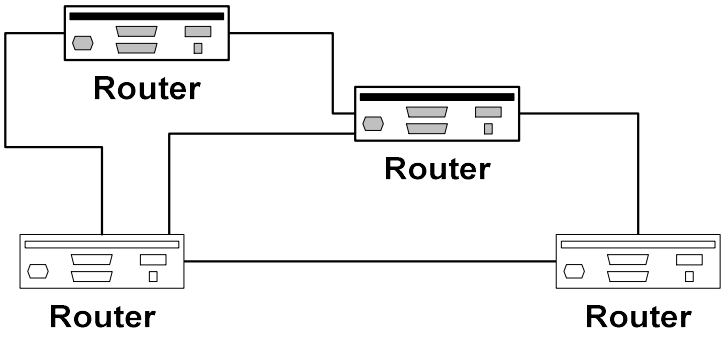
Point-to-Point Protocol (PPP)



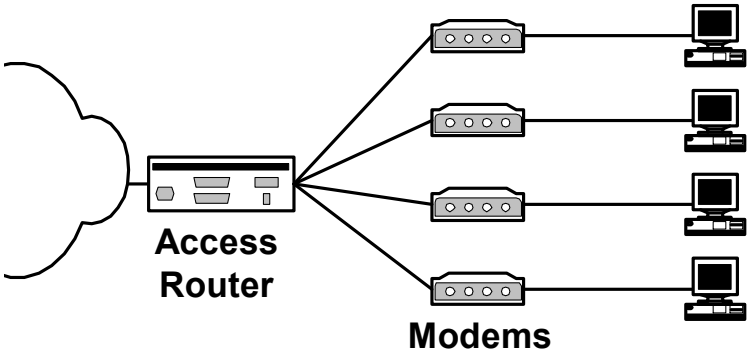
The Data Link Layer in the Internet

Point-to-Point Protocol (PPP)

- Used for Point-to-Point Connections only
- Used as data link control (encapsulates network layer PDUs) to connect :
  - Home users (PC) to ISP using a telephone line and a modem
  - Two routers



Between Routers



Dial-Up Access

PPP Design Requirements [RFC 1557]

- **packet framing:** encapsulation of network-layer datagram in data link frame
  - carry network layer data of any network layer protocol (not just IP) *at same time*
  - ability to demultiplex upwards
- **bit transparency:** must carry any bit pattern in the data field
- **error detection** (no correction)
- **connection liveness:** detect, signal link failure to network layer
- **network layer address negotiation:** endpoint can learn/configure each other's network address

PPP non-requirements

- no error correction/recovery
- no flow control
- out of order delivery OK
- no need to support multipoint links (e.g., polling)

Error recovery, flow control, data re-ordering all relegated to higher layers!

Point-to-Point Protocol (PPP)

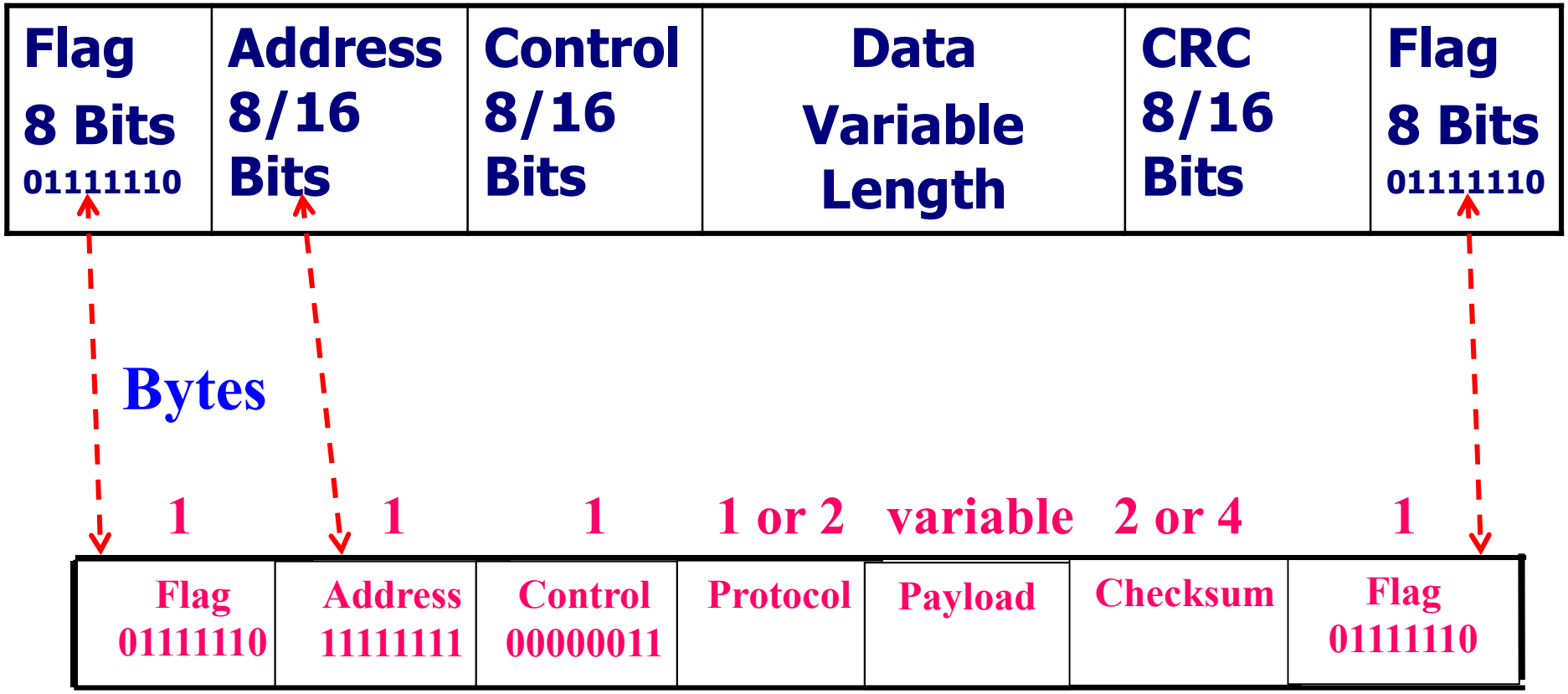
- A data link layer protocol
- Services provided by PPP
  - Defines the format of the frame to be exchanged between devices
  - Supports multiple network layer protocols simultaneously (like in multiprotocol routers)
  - Defines how two devices authenticate each other

PPP provides three features

- A framing method.
- A link control protocol (LCP) for bringing lines up, testing them, negotiating options, and bringing them down again when they are no longer needed. It supports synchronous and asynchronous circuits and byte-oriented and bit-oriented encodings.
- A way to negotiate network-layer options in a way that is independent of the network layer protocol to be used. The method chosen is to have a different NCP (Network Control Protocol) for each network layer supported.

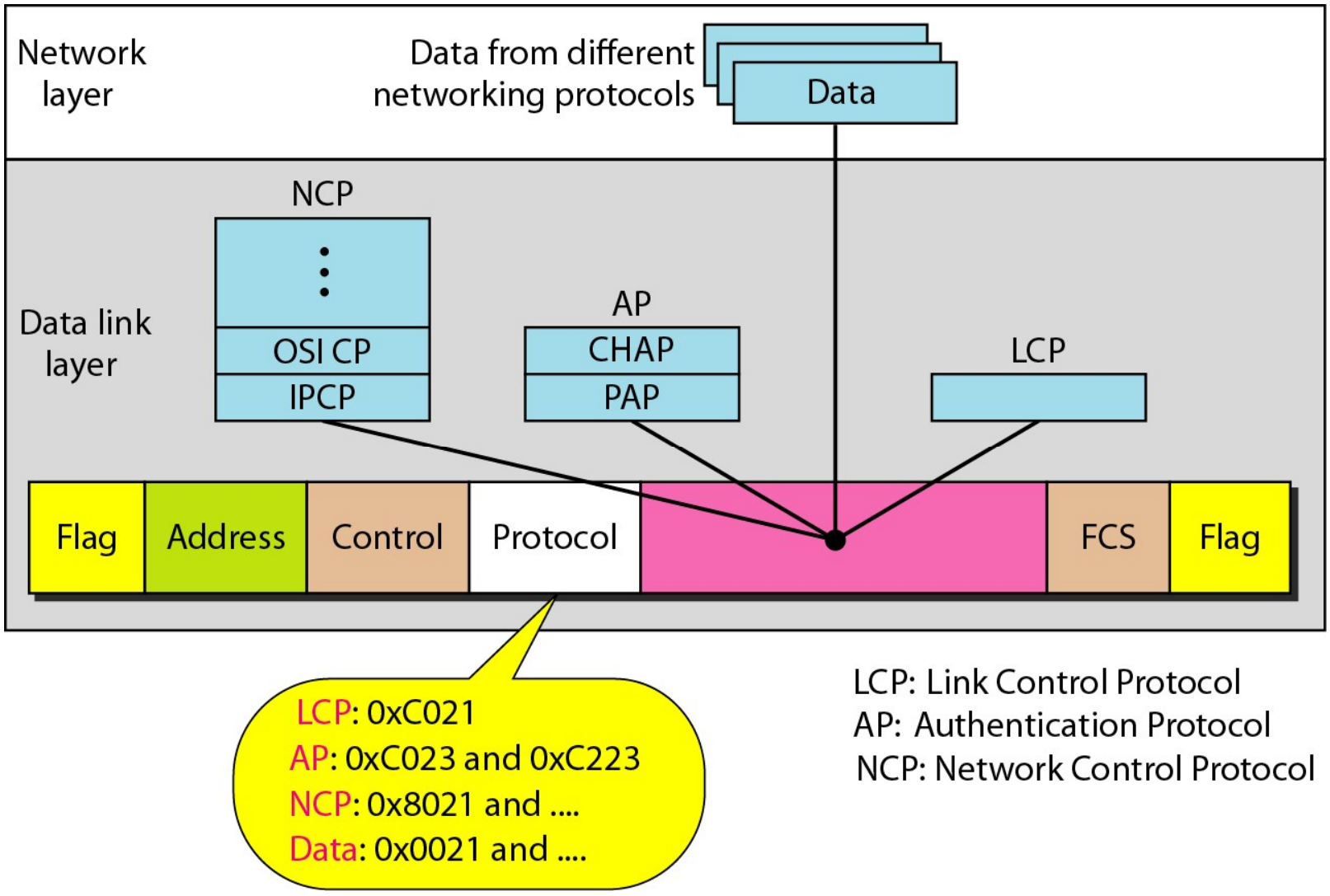
PPP Data Frame

High-Level Data Link Frame; Bit-Oriented



PPP Frame; Byte Oriented

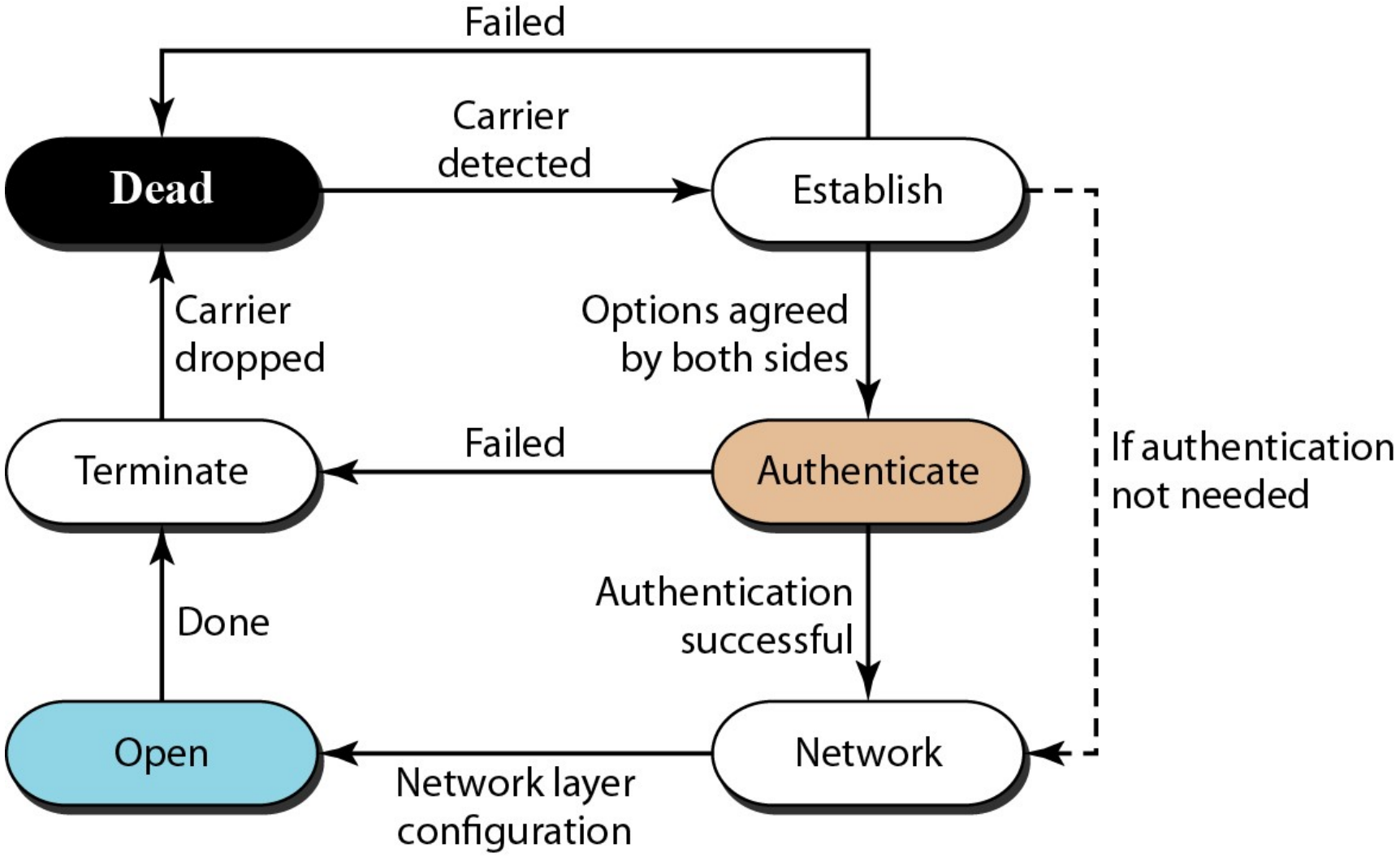
Multiplexing in PPP



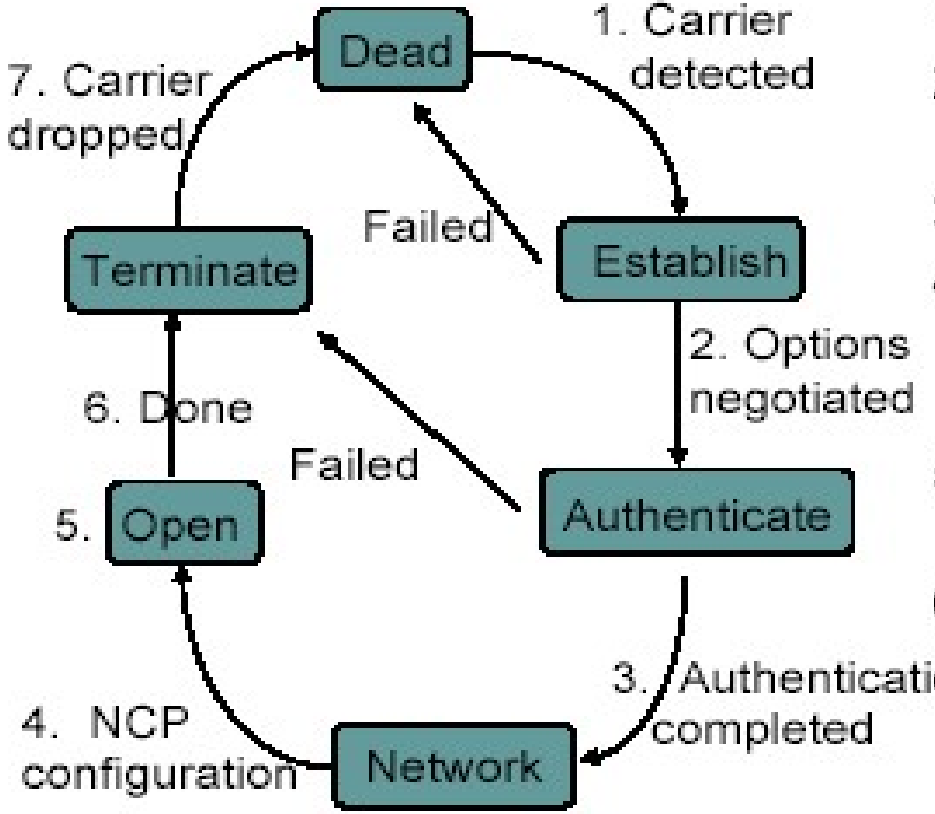
PPP Transition states

- **Dead state:** The link is not being used
- **Establishing state:**
  - Performed by LCP
  - Options are negotiated between endpoints (frame data length, keeping address and control fields, use Authentication or not, etc,)
- **Authentication state:**
  - Performed by Authentication Protocol
  - The user identity is verified
- **Networking state:**
  - Performed by NCP
  - Control (dynamic IP address, Compression of IP packets)
  - Exchange of Data packets
- **Terminating state:** The link is terminated (NCP first then LCP)

Transition phases



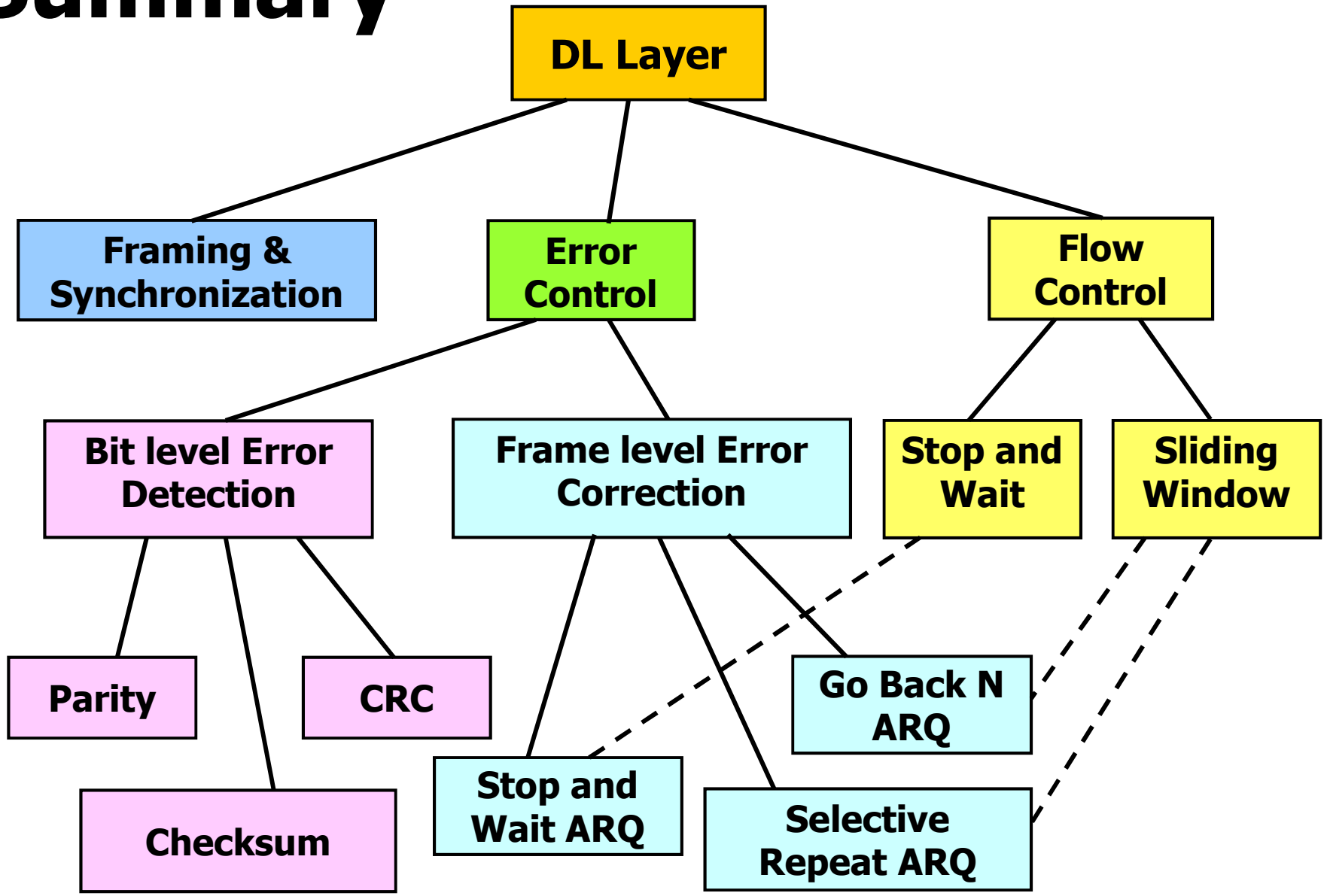
A Typical Scenario



Home PC to Internet Service Provider

1. PC calls router via modem.
2. PC and router exchange LCP packets to negotiate PPP parameters.
3. Check on identities.
4. NCP packets exchanged to configure the network layer, e.g., TCP/IP (requires IP address assignment).
5. Data transport, e.g. send/receive IP packets.
6. NCP used to tear down the network layer connection (free up IP address); LCP used to shut down data link layer connection.
7. Modem hangs up.

Summary


DL Layer
Framing & Synchronization
Error Control
Flow Control
Bit level Error Detection
Frame level Error Correction
Stop and Wait
Sliding Window
Parity
CRC
Checksum
Stop and Wait ARQ
Go Back N ARQ
Selective Repeat ARQ