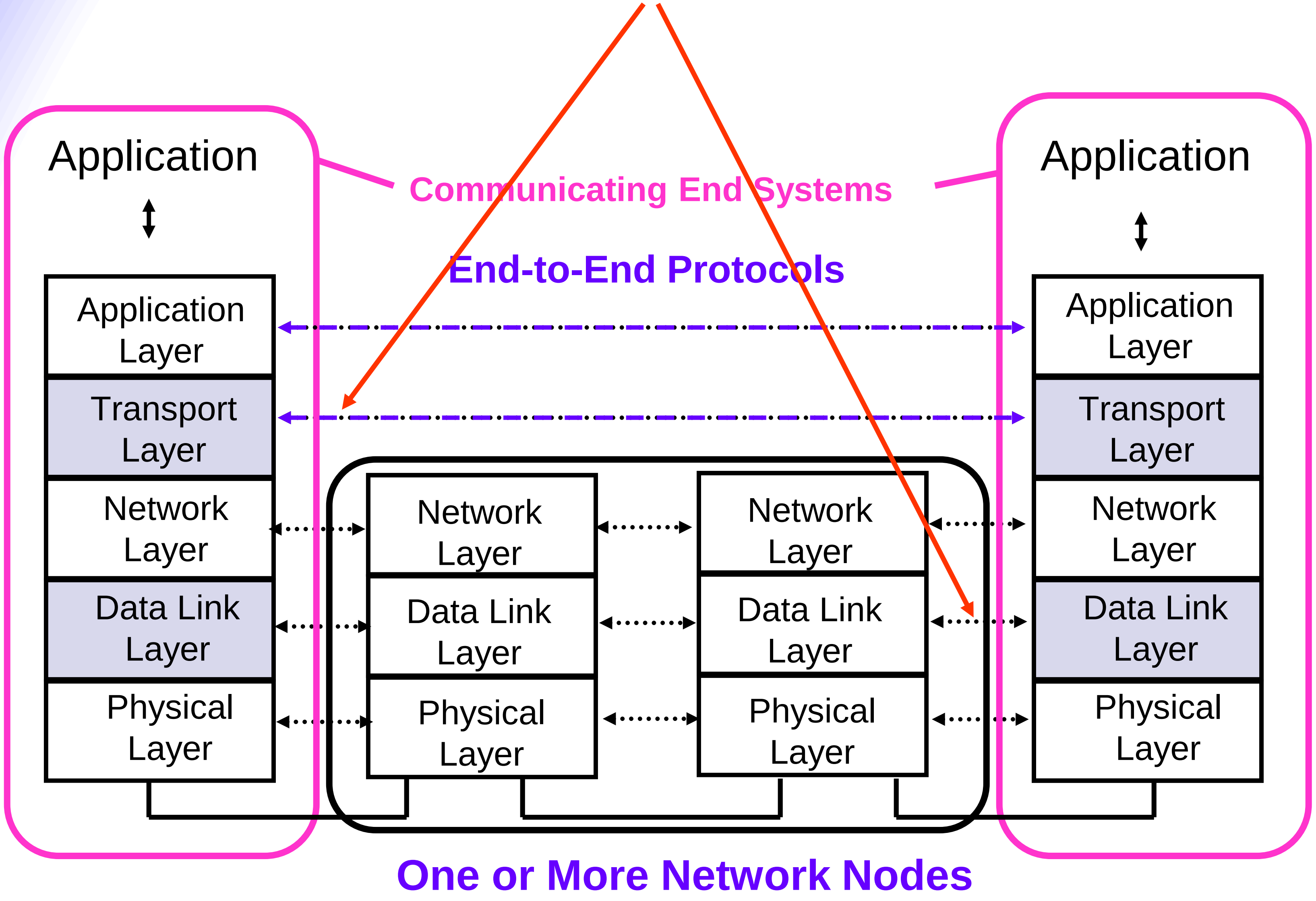
3. Data link layer

- reliable data transfer
  - peer-to-peer protocols
  - error detection
  - Automatic Repeat Request (ARQ)
  - flow control
- data link control
  - framing
  - point-to-point protocol (PPP)
  - high-level data link control (HDLC)

Peer-to-Peer protocols



- each layer provides a service to the layer above
- executes a peer-to-peer protocol that uses the services of the layer below
- how a protocol layer provides a reliable data transfer service across unreliable networks
- what data link layer protocols can be used

3.1 Peer-to-peer protocols

- layer *n*+1 requests a transfer of a service data unit (SDU)

- layer *n* peer processes construct protocol data units (PDUs)

- layer *n* protocol uses the services of layer *n*−1

- peer-to-peer protocol involves the interaction of two or more processes or entities through the exchange of messages (PDUs)

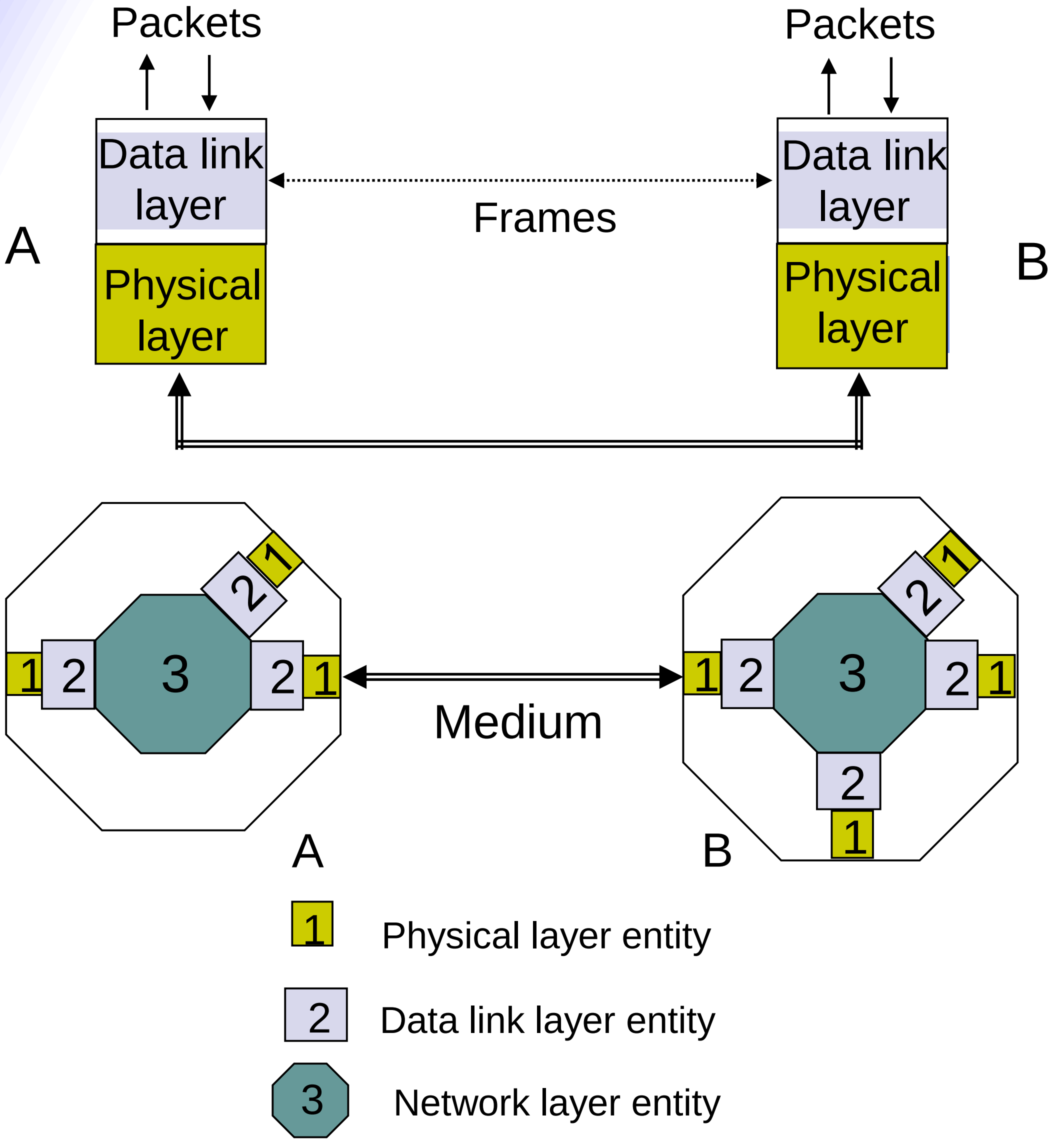
- SDU is delivered to the destination layer *n*+1

- *reliability*: are messages or information stream delivered error-free and without loss or duplication?
- *sequencing*: are messages or information stream delivered in order?
- how does a peer-to-peer protocol provide the service?
- you will learn some examples
- *ARQ protocols* combine error detection, retransmission, and sequence numbering to provide reliability & sequencing
- flow control - Transmission Control Protocol (TCP)
- data link layer protocol - High-Level Data Link Control (HDLC)

End-to-End vs. Hop-by-Hop

- A service feature can be provided by implementing a protocol
  - end-to-end across the network
  - across every hop in the network
- Example:
  - perform error control at every hop in the network or only between the source and destination
  - perform flow control between every hop in the network or only between source & destination
- consider the tradeoffs between the two approaches …

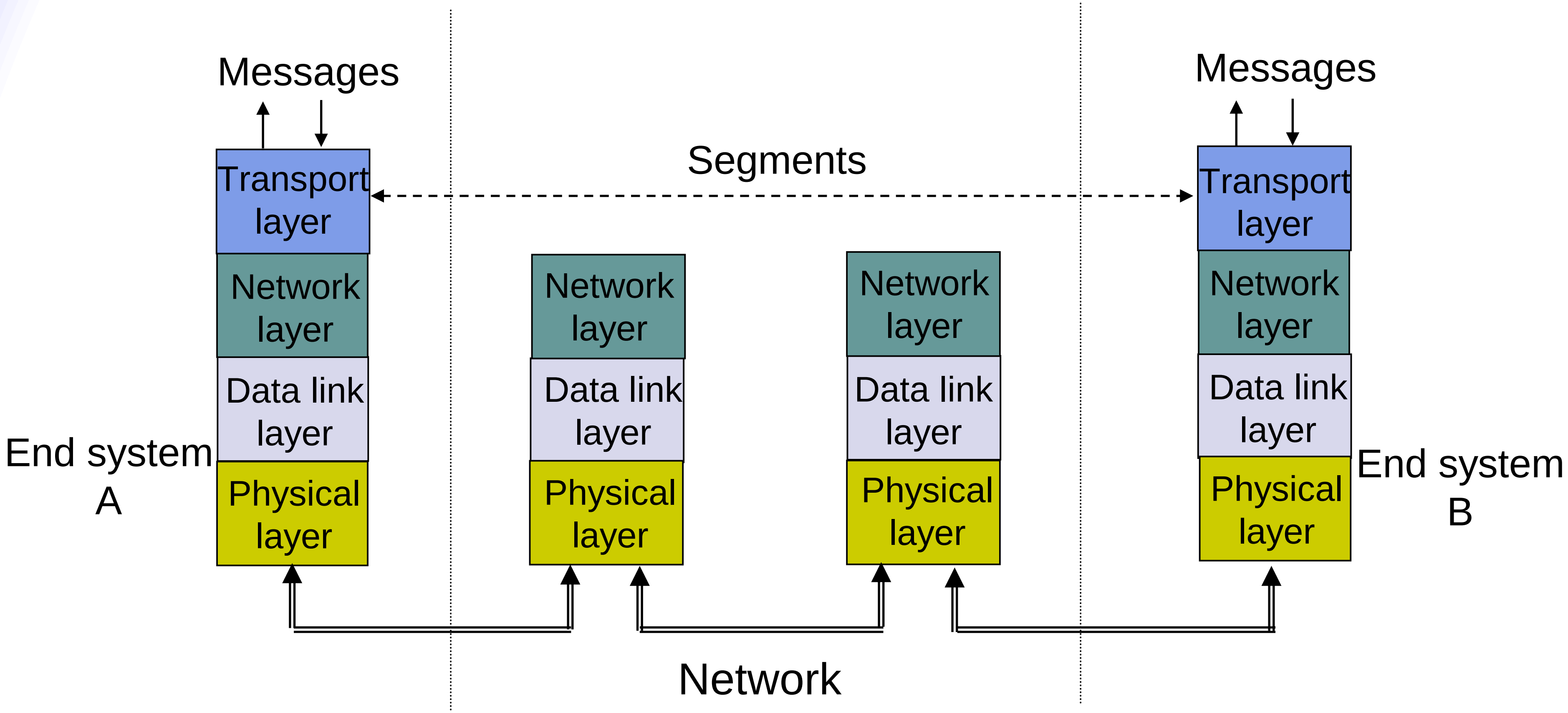
Error control in Data Link Layer



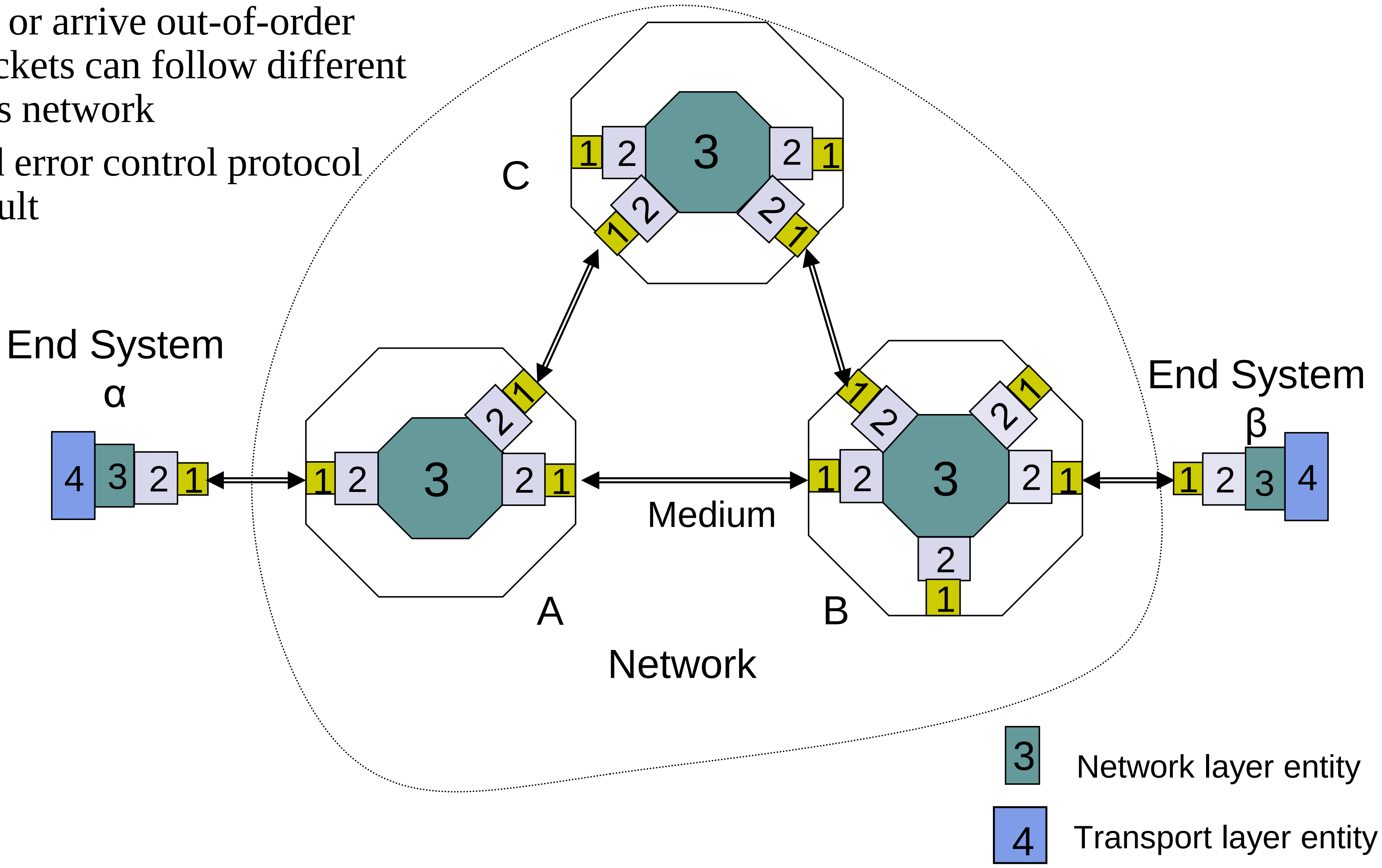
- Data link operates over wire-like, directly-connected systems
- Frames can be corrupted or lost, but arrive in order
- Data link performs error-checking & retransmission
- Ensures error-free packet transfer between two systems

Error control in Transport Layer

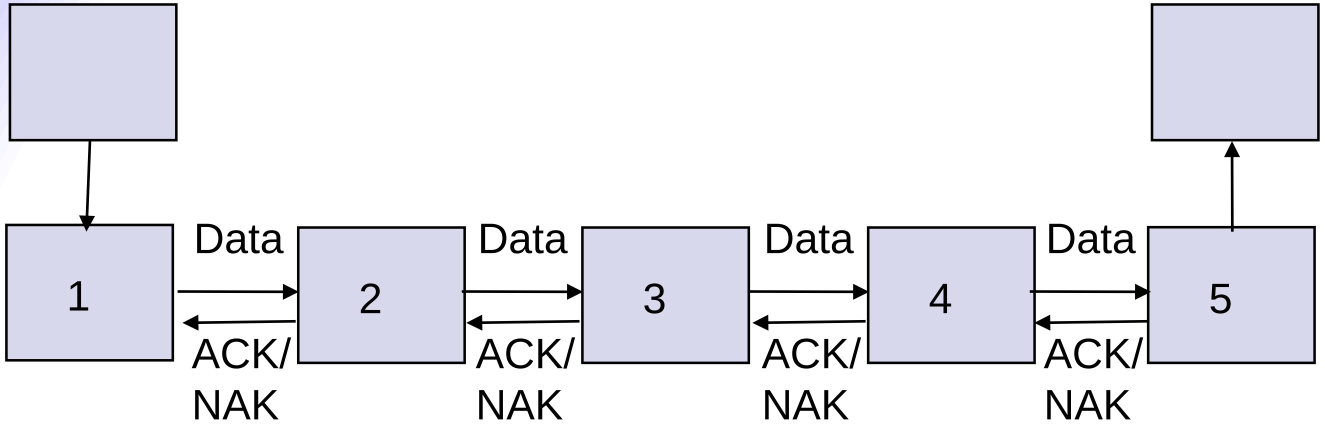
- Transport layer protocol (e.g. TCP) sends segments across network and performs end-to-end error checking & retransmission
- Underlying network is assumed to be unreliable



- Segments can experience long delays, can be lost, or arrive out-of-order because packets can follow different paths across network
- End-to-End error control protocol more difficult



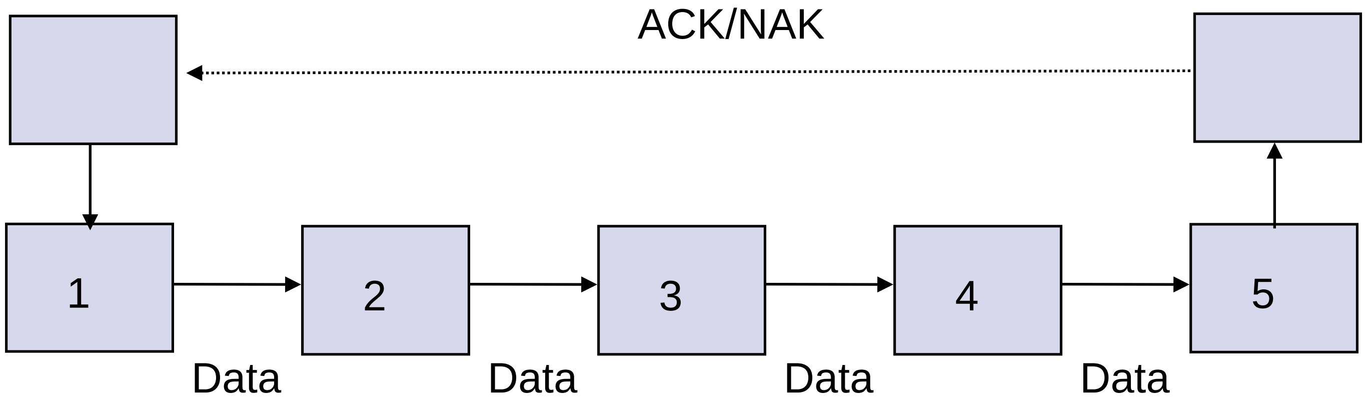
Hop-by-Hop


1 Data 2 Data 3 Data 4 Data 5

ACK/NAK ACK/NAK ACK/NAK ACK/NAK

Faster recovery

End-to-End


ACK/NAK

1 Data 2 Data 3 Data 4 Data 5

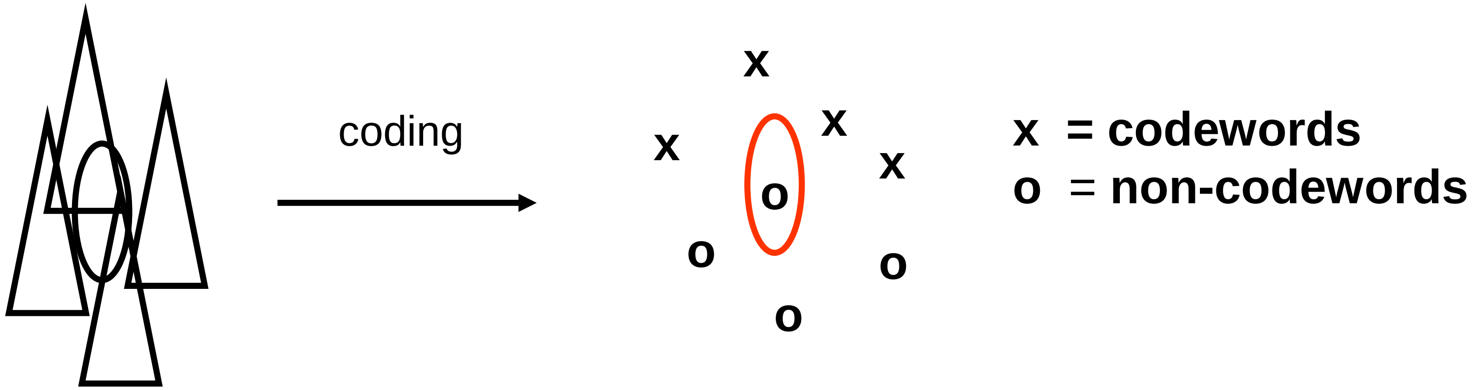
Simple
inside the
network

More scalable,
complexity at
the edge

3.2 Error detection

- digital transmission systems introduce errors – depend on physical media
- applications require certain reliability level – bit error rate acceptability
  - data applications require error-free transfer
  - voice & video applications tolerate some errors
- error control used when transmission system does *not* meet application requirement
- error control ensures a data stream is transmitted to a certain level of accuracy despite errors
- two basic approaches:
  - error detection & ARQ (*with return channel*, *waste bandwidth*)
  - error detection & forward error ***correction*** (FEC) (*no return channel or inefficient to retransmit*, *need more redundancy*, *not covered in this course*)

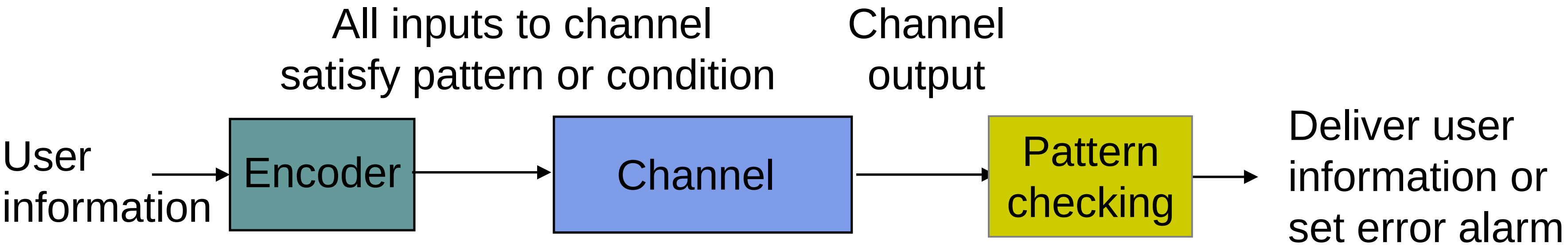
- all transmitted data blocks (“codewords”) satisfy a pattern
- if received block doesn’t satisfy pattern, it is in error
- redundancy: only a subset of all possible blocks can be codewords, e.g. data block length = 2 bits, 1 bit redundancy



2-bit data blocks

2-bit data, 1 bit redundancy

blindspot: when channel transforms a codeword into another codeword


User information

Encoder

All inputs to channel satisfy pattern or condition

Channel

Channel output

Pattern checking

Deliver user information or set error alarm

3.2.1 single parity check code

- append an overall parity check to *k* information bits

information 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: $(b_1, b_2, b_3, \ldots, b_{k,}, b_{k+1})$

- all codewords have even number of 1s
- receiver checks to see if number of 1s is even
  - all error patterns that change an odd number of 1s are detectable
  - all error patterns with even number of 1s are undetectable
- example: ASCII code (*7 bits for character + 1 parity bit*)

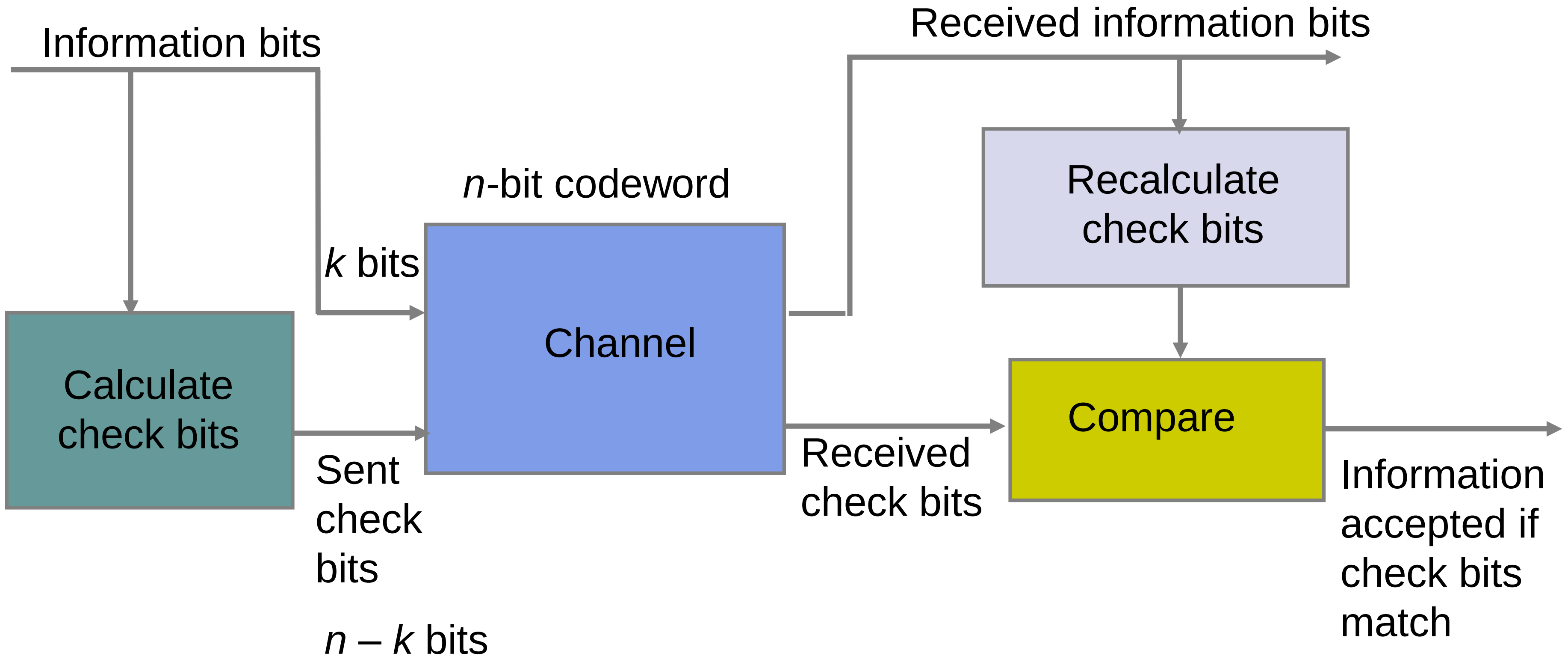
Example

- information (7 bits): (0, 1, 0, 1, 1, 0, 0)
- parity bit: b_8 = 0 + 1 + 0 + 1 +1 + 0 + 0 = 1
- codeword (8 bits): (0, 1, 0, 1, 1, 0, 0, 1)

- if single error in bit 3: (0, 1, 1, 1, 1, 0, 0, 1)
  - number of 1s = 5, odd
  - error detected

- if errors in bits 3 and 5: (0, 1, 1, 1, 0, 0, 0, 1)
  - number of 1s = 4, even
  - error not detected

Error Detection System



How good is the single parity check code?

- Redundancy: single parity check code adds 1 redundant bit per k information bits, overhead = $1/(k + 1)$
- Coverage: all error patterns with odd number of errors can be detected
  - an error pattern is a binary $(k + 1)$-tuple with 1s where errors occur and 0's elsewhere
  - of 2^{k+1} binary $(k + 1)$-tuples, ½ are odd, so 50% of error patterns can be detected
- Is it possible to detect more errors if we add more check bits?
- Yes, with the right codes

- many transmission channels introduce bit errors at random, independently of each other, and with probability p
- some error patterns are more probable than others

$$P[10000000] = p(1-p)^7 = (1-p)^8 \left(\frac{p}{1-p}\right) \text{ and}$$

$$P[11000000] = p^2(1-p)^6 = (1-p)^8 \left(\frac{p}{1-p}\right)^2$$

- in any worthwhile channel $p < 0.5$, and so $(p/(1-p)) < 1$
- it follows that patterns with 1 error are more likely than patterns with 2 errors and so forth
- What is the probability that an undetectable error pattern occurs?

- undetectable error pattern if even number of bit errors:

$$P[\text{error detection failure}] = P[\text{undetectable error pattern}]$$
$$= P[\text{error patterns with even number of 1s}]$$

$$= \binom{n}{2} p^2(1-p)^{n-2} + \binom{n}{4} p^4(1-p)^{n-4} + \ldots$$

- example: evaluate above for $n = 32$, $p = 10^{-3}$

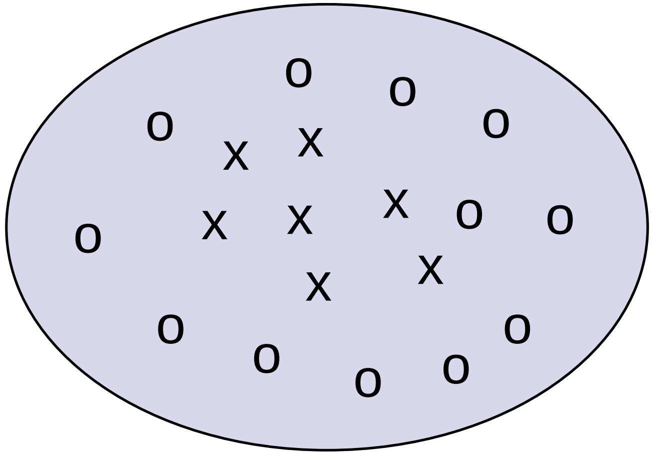
$$P[\text{undetectable error}] = \binom{32}{2}(10^{-3})^2\,(1-10^{-3})^{30} + \binom{32}{4}(10^{-3})^4\,(1-10^{-3})^{28} + \ldots$$

$$\approx 496\,(10^{-6}) + 35960\,(10^{-12}) \approx 4.96\,(10^{-4})$$

- roughly 1 in 2000 error patterns is undetectable

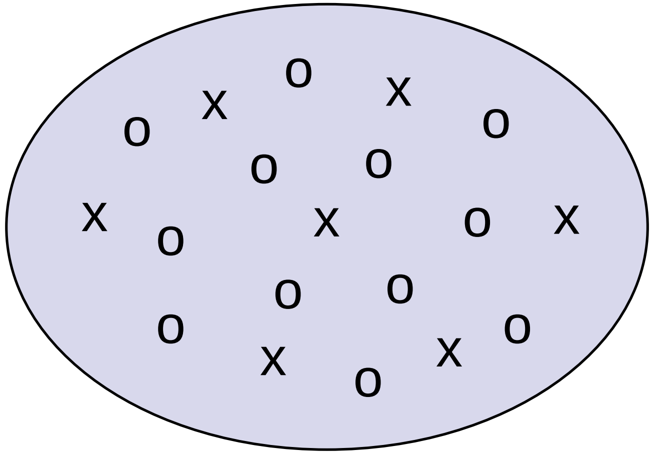
What is a good code?

- many channels have preference for error patterns that have fewer number of errors
- these error patterns map transmitted codeword to nearby *n*-tuple
- if codewords close to each other then detection failures will occur
- good codes should maximize separation between codewords



Poor distance properties

x = codewords
o = non-codewords



Good distance properties

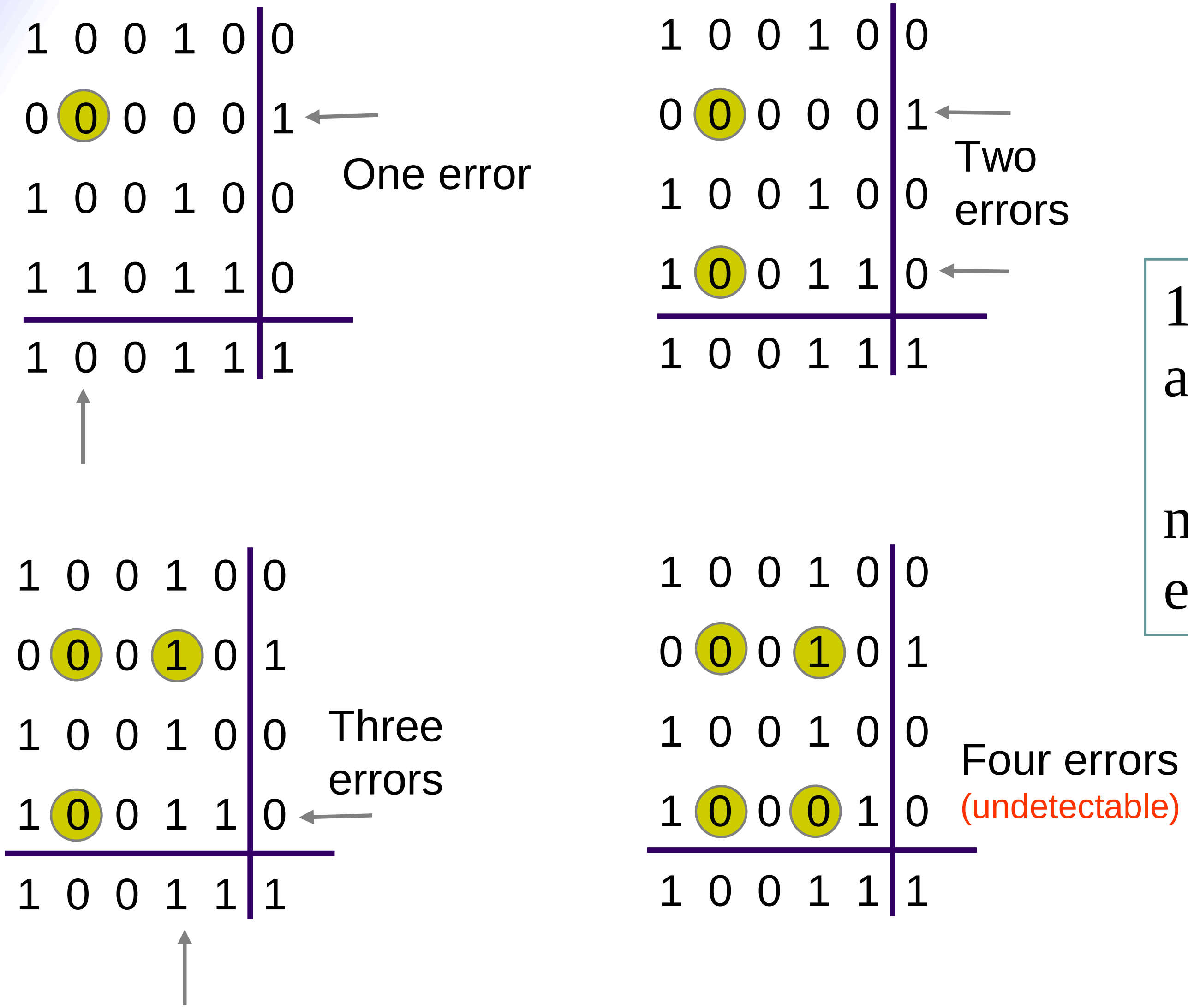
3.2.2 two-dimensional parity check

- more parity bits to improve coverage
- arrange information as columns
- add single parity bit to each column
- add a final “parity” column
- used in early error control systems

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

Last column consists of check bits for each row

Bottom row consists of check bit for each column



One error

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

Two errors

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

Three errors

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

Four errors (undetectable)

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

Arrows indicate failed check bits

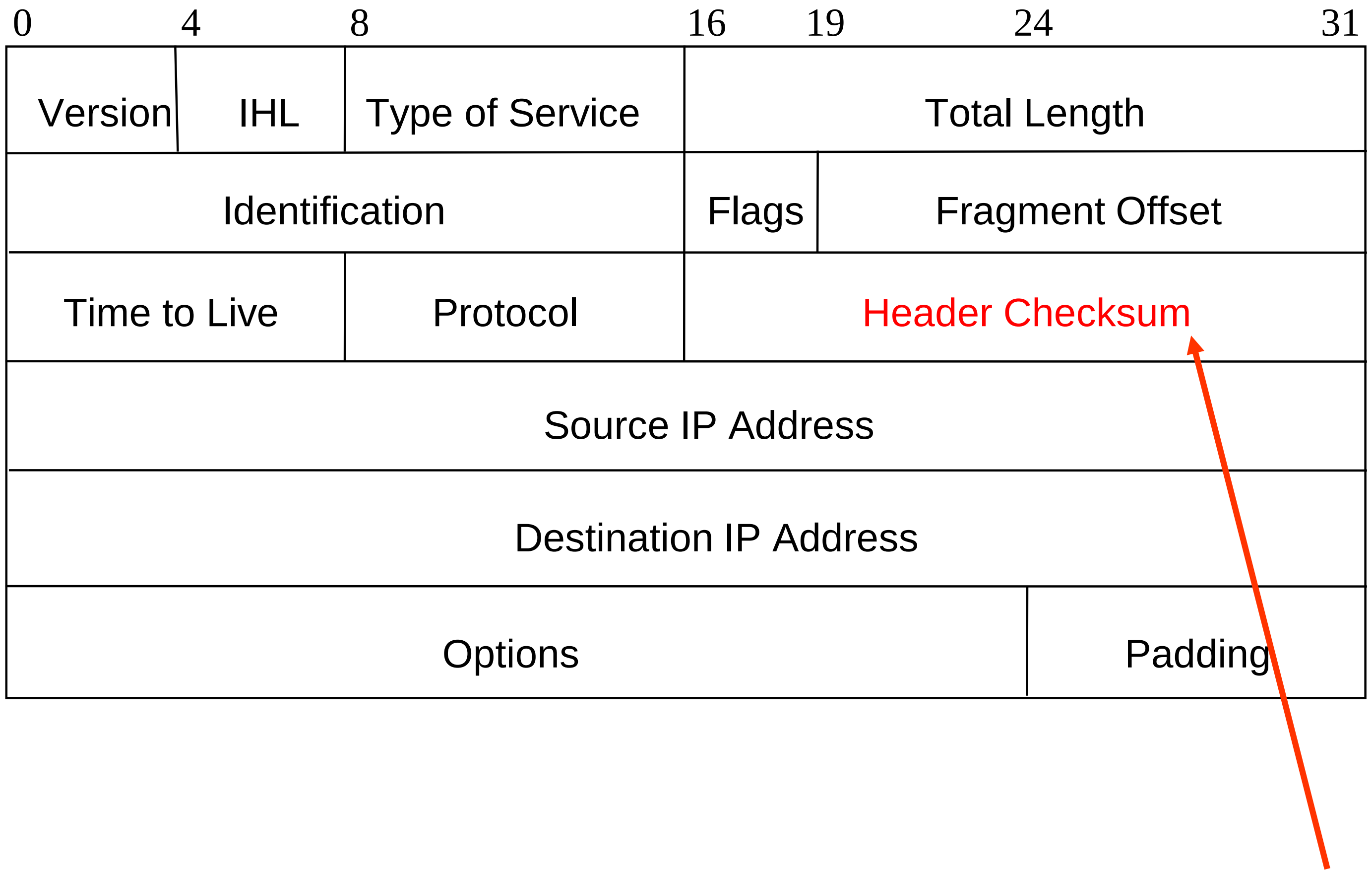
1, 2, or 3 errors can always be detected

not all patterns > 4 errors can be detected

- many applications require very low error rate
- need codes that detect the vast majority of errors
- single parity check codes do not detect enough errors
- two-dimensional parity check codes require too many check bits
- the following error detecting codes used in practice:
  - Internet checksum
  - polynomial codes - also called cyclic redundancy check (CRC)

3.2.3 Internet checksum

used in some Internet protocols, e.g. TCP, IP, …



0 4 8 16 19 24 31

Version | IHL | Type of Service | Total Length

Identification | Flags | Fragment Offset

Time to Live | Protocol | Header Checksum

Source IP Address

Destination IP Address

Options | Padding

how to calculate this?

- IP header uses check bits to detect errors in the ***header***
- a checksum is calculated for header contents
- checksum recalculated at every router, so algorithm selected for ease of implementation in software
- let header consists of L 16-bit words,

  $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{L-1}$
- the algorithm appends a 16-bit *checksum* $\mathbf{b}_L$
- for each received header, the router calculate

  $0 = \mathbf{b}_0 + \mathbf{b}_1 + \mathbf{b}_2 + \mathbf{b}_{L-1} + \mathbf{b}_L \text{ modulo } 2^{16} - 1$

Example

For simplicity, assuming 4-bit words

Use modulo arithmetic

- use modulo 2^4-1 arithmetic
- b_0 = 1100 = 12
- b_1 = 1010 = 10
- b_0 + b_1 = 12 + 10 = 22
- 22 modulo 15 = 7
- b_2 = -7 = 8 modulo 15
- checksum b_2 = 1000

Use binary arithmetic

- 16 = 1 modulo 15
- 10000 = 0001 modulo 15
- leading bit wraps around

$$\begin{aligned} b_0 + b_1 &= 1100 + 1010 \\ &= 10110 \\ &= 10000 + 0110 \\ &= 0001 + 0110 \\ &= 0111 \\ &= 7 \end{aligned}$$

Take 1s complement

b_2 = -0111 = 1000

3.2.4 polynomial codes

- polynomials instead of vectors for codewords
- polynomial arithmetic instead of checksum
- implemented using shift-register circuits
- most data communication standards use polynomial codes for error detection
- polynomial code is also basis for powerful error-correction methods

Binary polynomial arithmetic

- binary vectors map to polynomials

$$(i_{k-1}, i_{k-2}, \ldots, i_2, i_1, i_0) \rightarrow i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_2x^2 + i_1x + i_0$$

addition:

$$(x^7 + x^6 + 1) + (x^6 + x^5) = x^7 + x^6 + x^6 + x^5 + 1$$

$$= x^7 + (1+1)x^6 + x^5 + 1$$

$= x^7 + x^5 + 1$ since 1+1=0 modulo 2

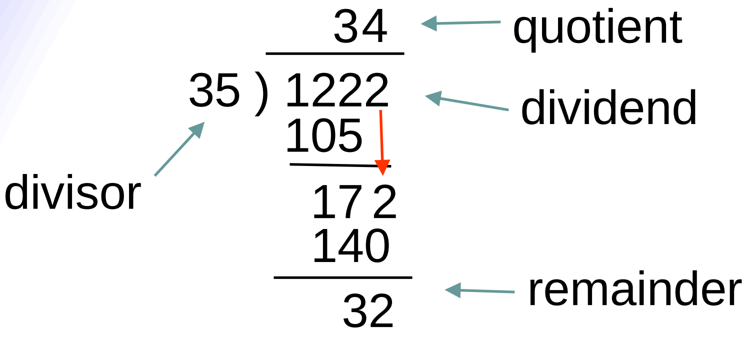
multiplication:

$$(x + 1)(x^2 + x + 1) = x(x^2 + x + 1) + 1(x^2 + x + 1)$$

$$= (x^3 + x^2 + x) + (x^2 + x + 1)$$

$$= x^3 + 1$$

division with decimal numbers:

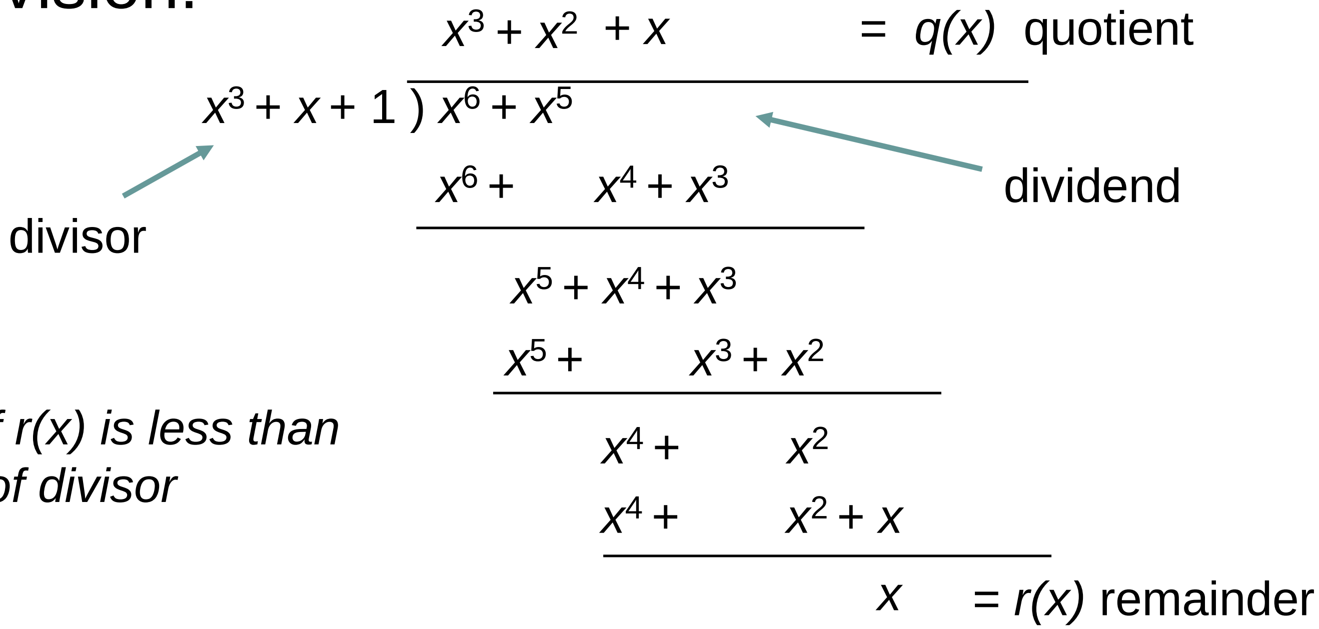


$$
\begin{array}{r}
34 \\
35\ )\ \overline{1222} \\
\underline{105} \\
172 \\
\underline{140} \\
32
\end{array}
$$

dividend = quotient x divisor + remainder

1222 = 34 x 35 + 32

polynomial division:



$$
\begin{array}{l}
\qquad\qquad\qquad\quad x^3 + x^2 + x \qquad\qquad = q(x) \text{ quotient} \\
x^3 + x + 1\ )\ \overline{x^6 + x^5 \qquad\qquad\qquad\qquad\qquad\qquad} \\
\qquad\qquad\qquad\ \underline{x^6 + \qquad x^4 + x^3 \qquad\qquad} \\
\qquad\qquad\qquad\qquad x^5 + x^4 + x^3 \\
\qquad\qquad\qquad\quad\ \underline{x^5 + \qquad x^3 + x^2 \qquad} \\
\qquad\qquad\qquad\qquad\quad x^4 + \qquad x^2 \\
\qquad\qquad\qquad\qquad\quad \underline{x^4 + \qquad x^2 + x \qquad} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad x \quad = r(x) \text{ remainder}
\end{array}
$$

Note: degree of r(x) is less than degree of divisor

Polynomial coding

- code has binary *generator polynomial* of degree $n - k$

$$g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \ldots + g_2x^2 + g_1x + 1$$

- k information bits define polynomial of degree $k - 1$

$$i(x) = i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \ldots + i_2x^2 + i_1x + i_0$$

- find *remainder polynomial* of at most degree $n - k - 1$

$$\begin{array}{r} q(x) \\ g(x)\;\overline{)\;x^{n-k}\,i(x)} \\ r(x) \end{array}$$

$$x^{n-k}i(x) = q(x)g(x) + r(x)$$

- define the *codeword polynomial* of degree $n - 1$

$$\boxed{\underbrace{b(x)}_{n \text{ bits}} = \underbrace{x^{n-k}i(x)}_{k \text{ bits}} + \underbrace{r(x)}_{n-k \text{ bits}}}$$

Example: k = 4, n – k = 3

generator polynomial: $g(x) = x^3 + x + 1$

information: (1,1,0,0) $i(x) = x^3 + x^2$

encoding: $x^3 i(x) = x^6 + x^5$

$$\begin{array}{rl}
 & x^3 + x^2 + x \\
\cline{2-2}
x^3 + x + 1\,) & x^6 + x^5 \\
 & x^6 + x^4 + x^3 \\
\cline{2-2}
 & \phantom{x^6 +{}} x^5 + x^4 + x^3 \\
 & \phantom{x^6 +{}} x^5 + x^3 + x^2 \\
\cline{2-2}
 & \phantom{x^6 + x^5 +{}} x^4 + x^2 \\
 & \phantom{x^6 + x^5 +{}} x^4 + x^2 + x \\
\cline{2-2}
 & \phantom{x^6 + x^5 + x^4 + x^2 +{}} x
\end{array}$$

$$\begin{array}{rl}
 & 1110 \\
\cline{2-2}
1011\,) & 1100000 \\
 & 1011 \\
\cline{2-2}
 & 1110 \\
 & 1011 \\
\cline{2-2}
 & 1010 \\
 & 1011 \\
\cline{2-2}
 & 010
\end{array}$$

transmitted codeword:

$b(x) = x^6 + x^5 + x$

$\Longrightarrow$ $\underline{b}$ = (1,1,0,0,0,1,0)

- all codewords satisfy the following **pattern**:

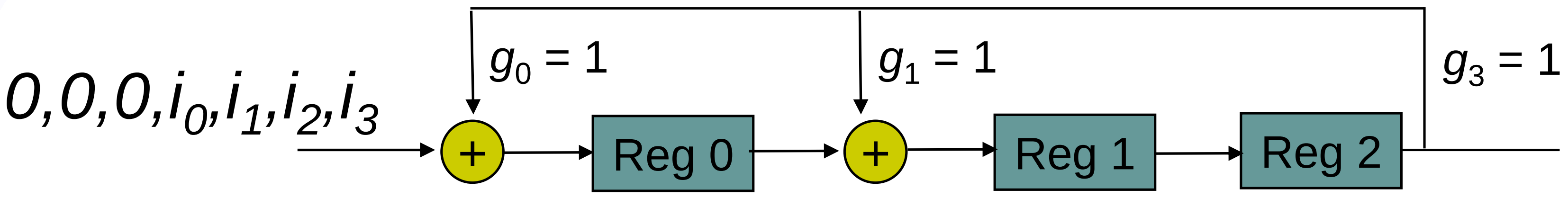
$$b(x) = x^{n-k}i(x) + r(x) = q(x)g(x) + r(x) + r(x) = q(x)g(x)$$

- all codewords are a multiple of *g(x)!*
- receiver should divide received *n*-tuple by *g*(*x*) and check if remainder is zero
- if remainder is non-zero, then received *n*-tuple is not a codeword

Implementation

1. accept information bits $i_{k\text{-}1}, i_{k\text{-}2}, \ldots, i_2, i_1, i_0$
2. append $n - k$ zeros to information bits
3. feed sequence to shift-register circuit that performs polynomial division
4. after n shifts, the shift register contains the remainder

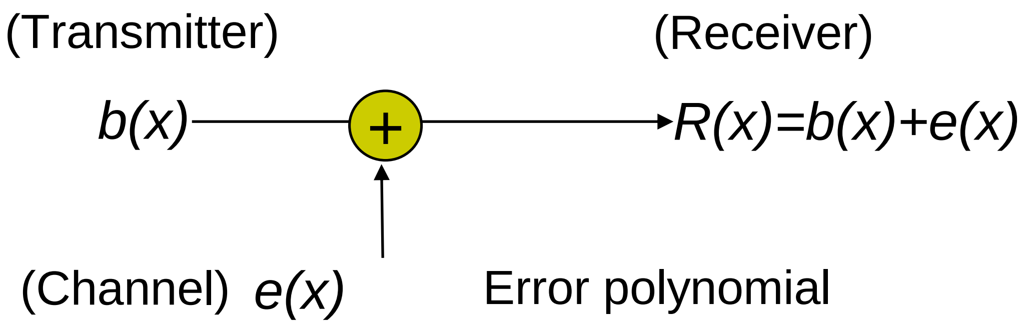
Encoder for $g(x) = x^3 + x + 1$



Clock	Input	Reg 0	Reg 1	Reg 2
0	-	0	0	0
1	$1 = i_3$	1	0	0
2	$1 = i_2$	1	1	0
3	$0 = i_1$	0	1	1
4	$0 = i_0$	1	1	1
5	0	1	0	1
6	0	1	0	0
7	0	**0**	**1**	**0**
	Check bits:	$\boldsymbol{r_0 = 0}$	$\boldsymbol{r_1 = 1}$	$\boldsymbol{r_2 = 0}$

$\Longrightarrow \boldsymbol{r(x) = x}$

Undetectable error patterns



- *e(x)* has 1s in error locations & 0s elsewhere
- receiver divides the received polynomial *R(x)* by *g(x)*
- blindspot: if *e(x)* is a multiple of *g(x)*, that is, *e(x)* is a non-zero codeword, then

$$R(x) = b(x) + e(x) = q(x)g(x) + q'(x)g(x)$$

- the set of undetectable error polynomials is the set of non-zero code polynomials
- choose the generator polynomial so that selected error patterns can be detected

Designing good polynomial codes

- select generator polynomial so that likely error patterns are not multiples of $g(x)$
- detecting single errors
  - $e(x) = x^i$ for error in location $i + 1$
  - if $g(x)$ has more than 1 term, it cannot divide x^i
- detecting double errors
  - $e(x) = x^i + x^j = x^i(x^{j-i}+1)$ *where* $j > i$
  - if $g(x)$ has more than 1 term, it cannot divide x^i
  - if g(x) is a primitive polynomial, it cannot divide $x^m + 1$ for all $m < 2^{n-k} - 1$ (need to keep codeword length less than $2^{n-k} - 1$)
  - primitive polynomials can be found by consulting coding theory books

- detecting odd number of errors
  - for odd number of errors, $e(x)$ evaluated at $x = 1$ is 1, therefore $(x + 1)$ is not a factor of $e(x)$
  - suppose all codeword polynomials have an even number of 1s, $b(x)$ evaluated at $x = 1$ is zero because $b(x)$ has an even number of 1s
  - this implies $x + 1$ must be a factor of all $b(x)$
  - pick $g(x) = (x + 1)\ p(x)$ where $p(x)$ is primitive

Standard generator polynomials

- CRC-8: ATM

  $= x^8 + x^2 + x + 1$

- CRC-16: Bisync

  $= x^{16} + x^{15} + x^2 + 1$

  $= (x + 1)(x^{15} + x + 1)$

- CCITT-16: HDLC, XMODEM, V.41

  $= x^{16} + x^{12} + x^5 + 1$

- CCITT-32: IEEE 802, DoD, V.42

  $= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

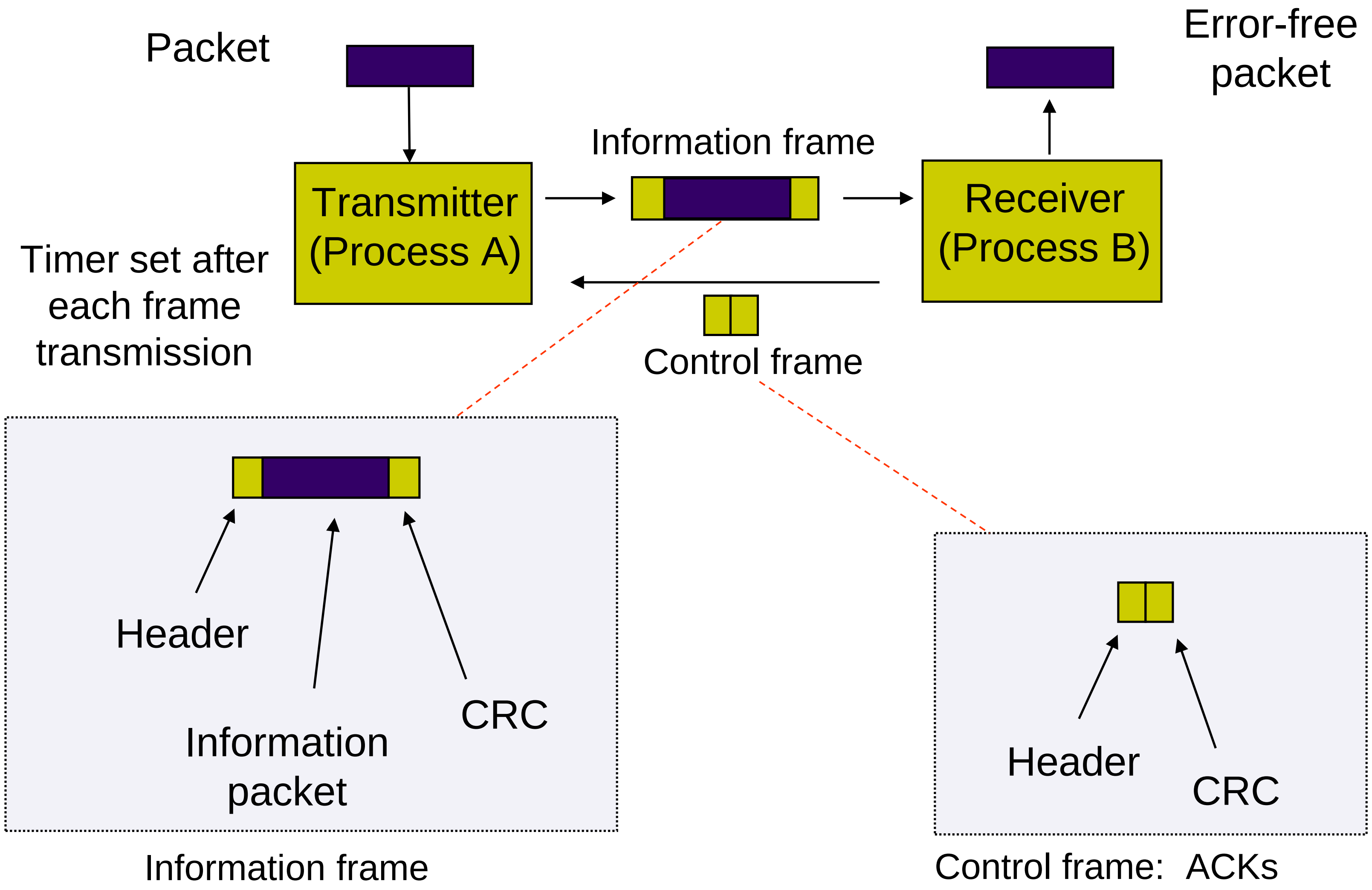
3.3 Automatic Repeat Request (ARQ)

- Purpose:

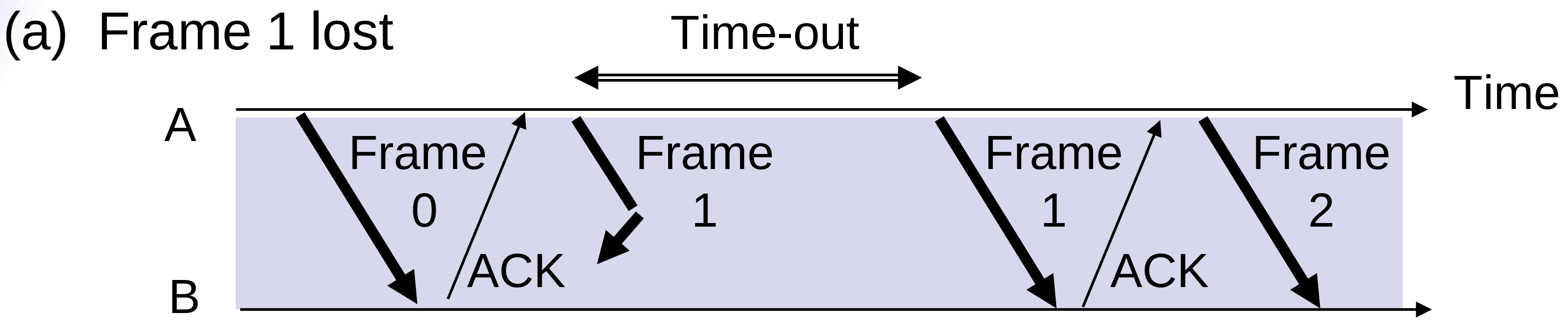
  to ensure a sequence of information packets is delivered in order and without errors or duplications despite transmission errors and losses

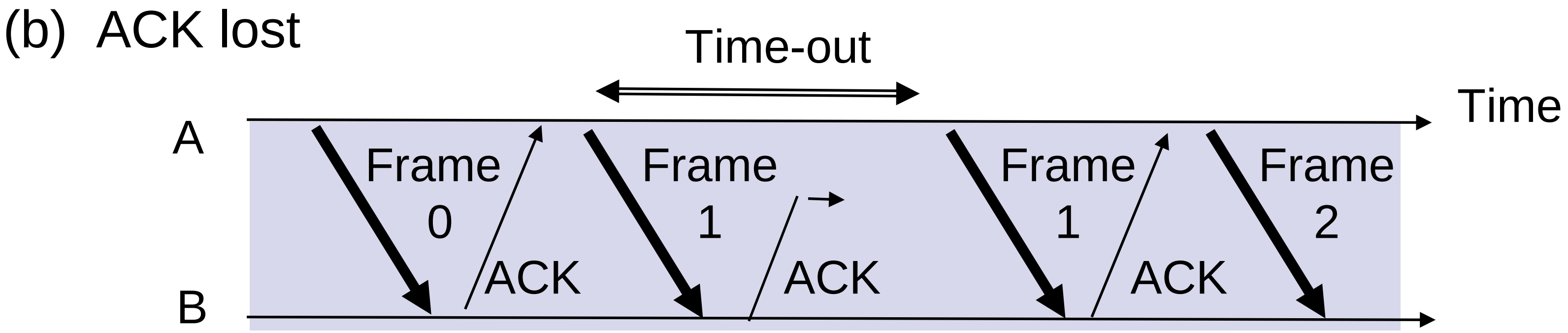
- We will look at:
  - Stop-and-Wait ARQ
  - Go-Back N ARQ
  - Selective Repeat ARQ
- Basic elements of ARQ:
  - error-detecting code with high error coverage
  - ACKs (positive acknowledgments)
  - NAKs (negative acknowledgments)
  - timeout mechanism, and ….

Transmit a frame, wait for ACK



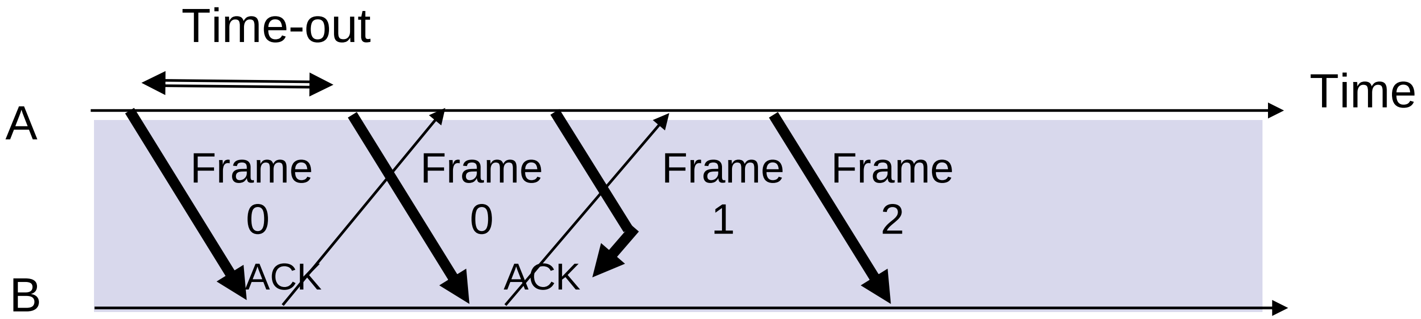
3.3.1 Stop-and-Wait ARQ (SW)





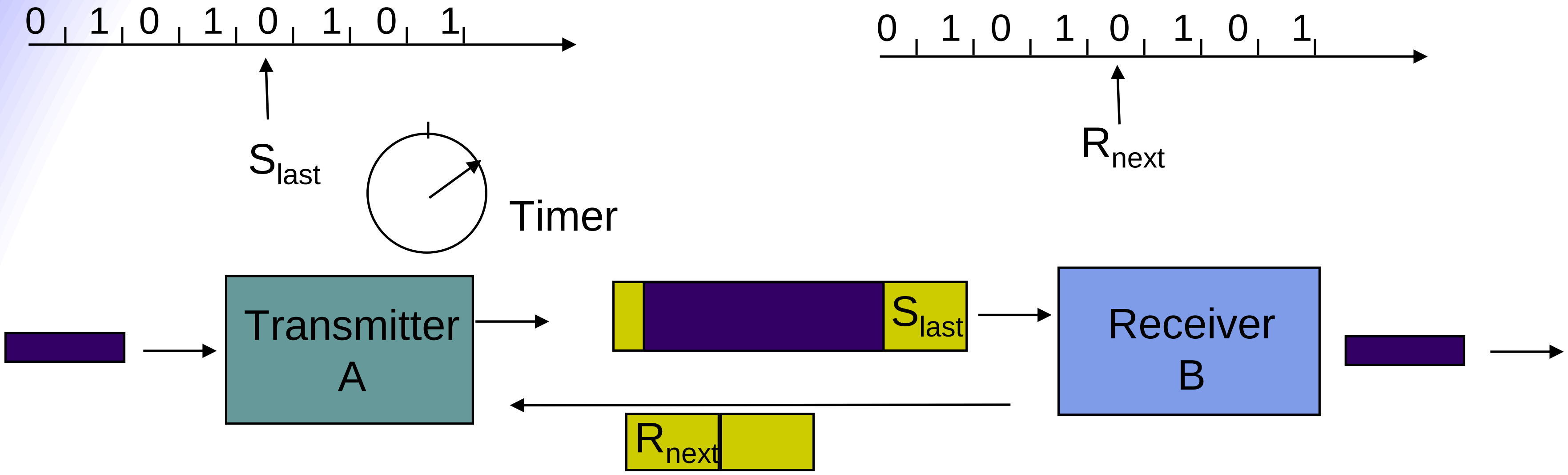
- In cases (a) & (b) the transmitting station A acts the same way
- But in case (b) the receiving station B accepts frame 1 twice
- Question: How is the receiver to know the second frame is also frame 1?
- Answer: ***Add frame sequence number in header***
- S_{last} is sequence number of most recent transmitted frame

(c) Premature Time-out



- The transmitting station A misinterprets duplicate ACKs
- Incorrectly assumes second ACK acknowledges Frame 1
- Question: How is the transmitter to know second ACK is for frame 0?
- Answer: ***Add frame sequence number in ACK header***
- R_{next} is sequence number of next frame expected by the receiver
- Implicitly acknowledges receipt of all prior frames

1-bit sequence numbering



0 1 0 1 0 1 0 1

S_{last}

Timer

0 1 0 1 0 1 0 1

R_{next}

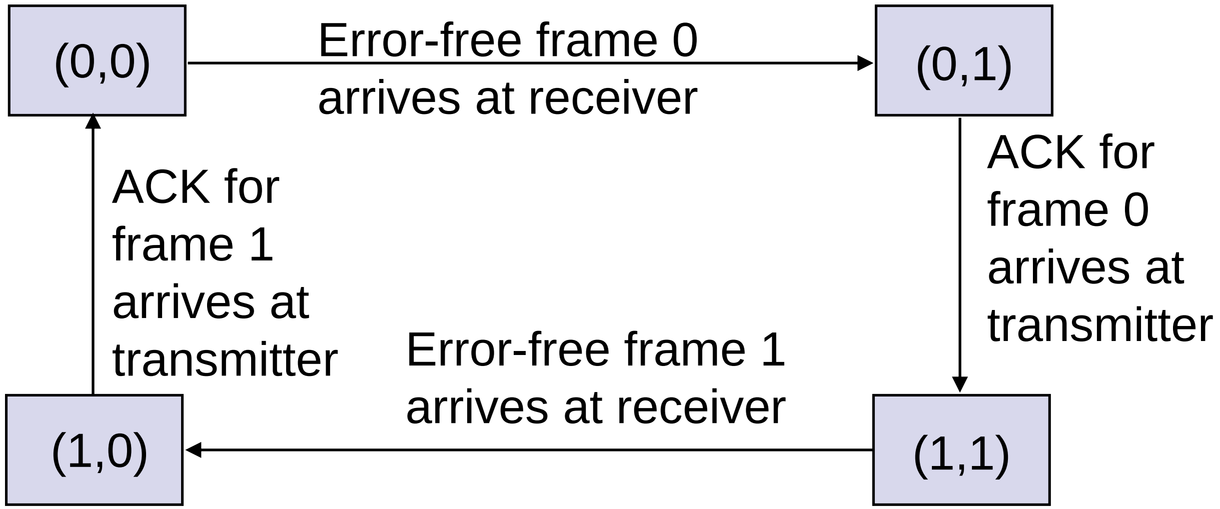
Transmitter A

S_{last}

R_{next}

Receiver B

Global State:
(S_{last}, R_{next})



(0,0)

Error-free frame 0 arrives at receiver

(0,1)

ACK for frame 0 arrives at transmitter

(1,1)

Error-free frame 1 arrives at receiver

(1,0)

ACK for frame 1 arrives at transmitter

protocol

Transmitter

Ready state

- Await request from higher layer for packet transfer
- When request arrives, transmit frame with updated S_{last} and CRC
- Go to Wait State

Wait state

- Wait for ACK or timer to expire; block requests from higher layer
- If timeout expires
  - retransmit frame and reset timer
- If ACK received:
  - If sequence number is incorrect or if errors detected: ignore ACK
  - If sequence number is correct ($R_{next} = S_{last} +1$): accept frame, go to Ready state

Receiver

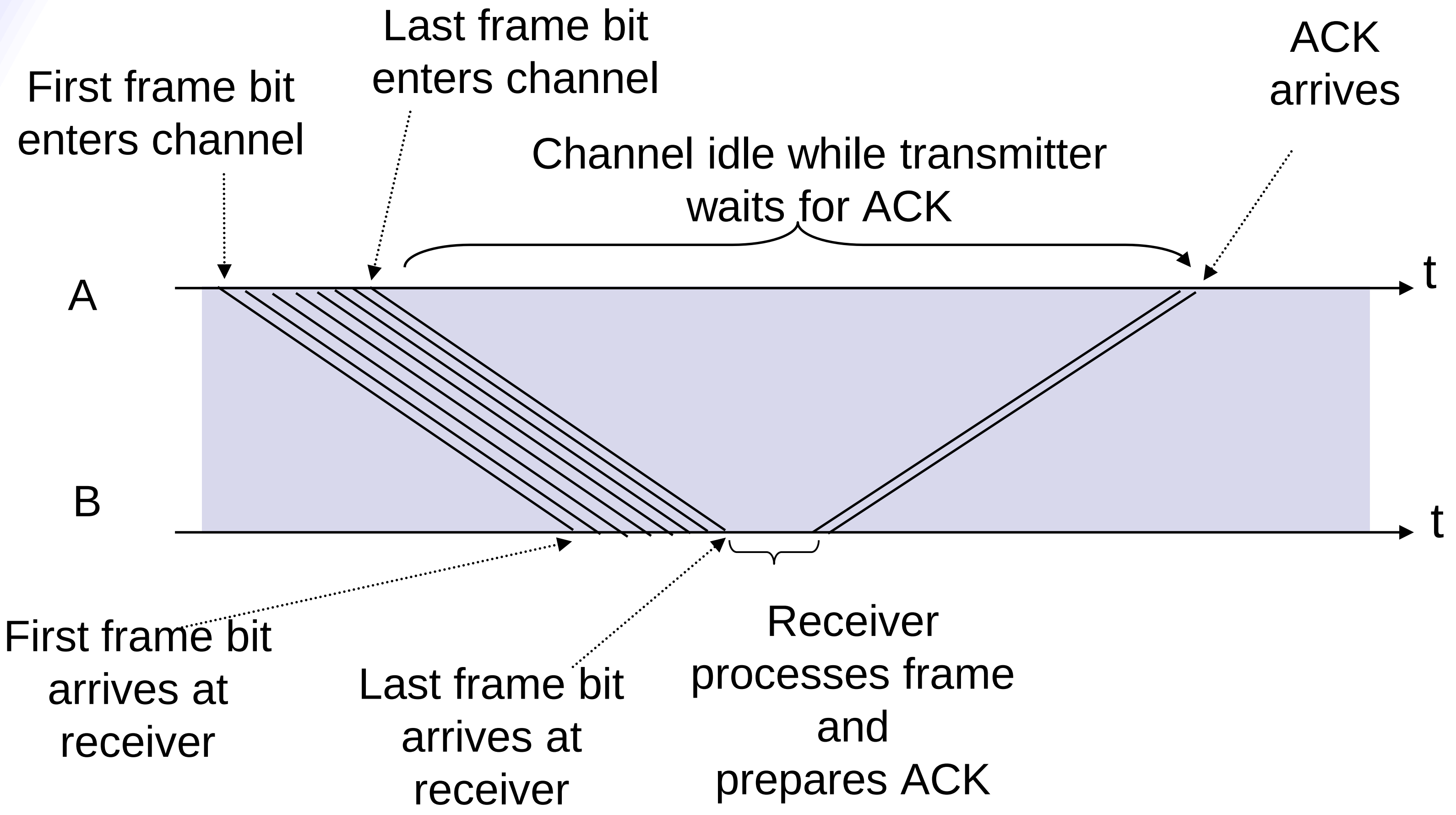
Always in Ready State

- Wait for arrival of new frame
- When frame arrives, check for errors
- If no errors detected and sequence number is correct ($S_{last}=R_{next}$), then
  - accept frame,
  - update R_{next},
  - send ACK frame with R_{next},
  - deliver packet to higher layer
- If no errors detected and wrong sequence number
  - discard frame
  - send ACK frame with R_{next}
- If errors detected
  - discard frame

applications

- IBM Binary Synchronous Communications protocol (Bisync): character-oriented data link control
- Xmodem: modem file transfer protocol
- Trivial File Transfer Protocol (RFC 1350): simple protocol for file transfer over User Datagram Protocol (UDP)

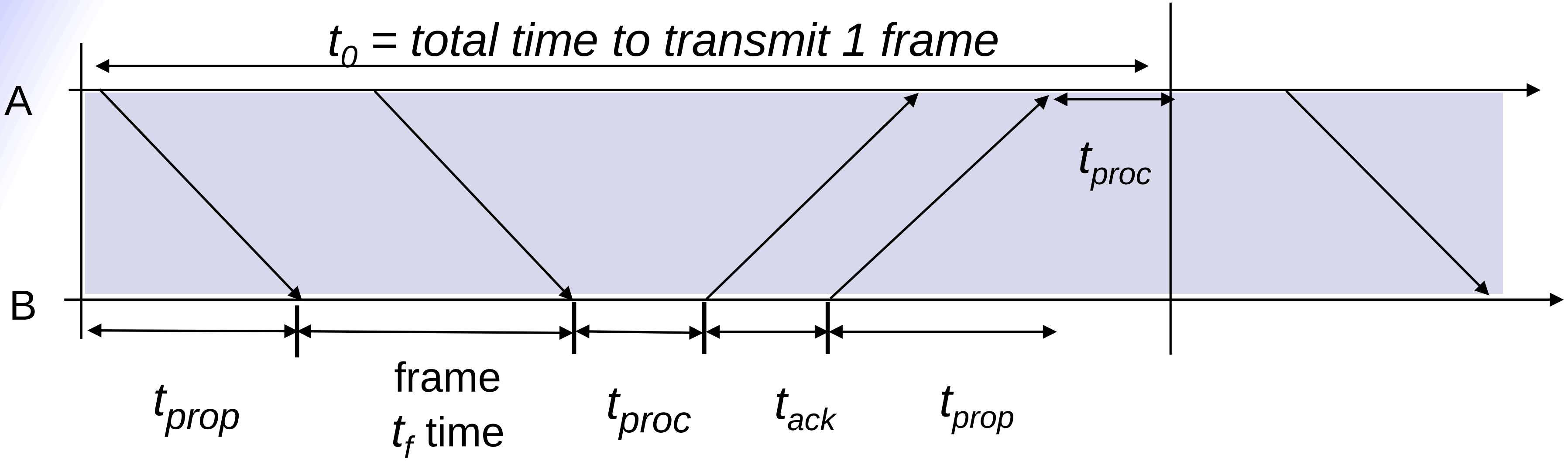
efficiency



Last frame bit enters channel
ACK arrives
First frame bit enters channel
Channel idle while transmitter waits for ACK
t
A
B
t
First frame bit arrives at receiver
Last frame bit arrives at receiver
Receiver processes frame and prepares ACK

- 10000 bit frame @ 1 Mbps takes 10 ms to transmit
- If wait for ACK = 1 ms, then efficiency = 10/11 = 91%
- If wait for ACK = 20 ms, then efficiency =10/30 = 33%

delay components



$$t_0 = 2t_{prop} + 2t_{proc} + t_f + t_{ack}$$

$$= 2t_{prop} + 2t_{proc} + \frac{n_f}{R} + \frac{n_a}{R}$$

bits/info frame

bits/ACK frame

channel transmission rate

efficiency on error-free channel

Effective transmission rate:

bits for header & CRC

$$R_{eff}^{0} = \frac{\text{number of information bits delivered to destination}}{\text{total time required to deliver the information bits}} = \frac{n_f - n_o}{t_0},$$

Transmission efficiency:

$$\eta_0 = \frac{R_{eff}}{R} = \frac{\dfrac{n_f - n_o}{t_0}}{R} = \frac{1-\dfrac{n_o}{n_f}}{1+\dfrac{n_a}{n_f}+\dfrac{2(t_{prop}+t_{proc})R}{n_f}}.$$

Effect of
frame overhead

Effect of
ACK frame

Effect of
delay-bandwidth product

impact of delay-bandwidth product

n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

2xDelayxBW **Efficiency**	1 ms 200 km	10 ms 2,000 km	100 ms 20,000 km	1 sec 200,000 km
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%	10^9 0.001%

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

efficiency in channel with errors

- Let 1 – P_f = probability frame arrives without errors
- Average number of transmissions to first correct arrival is then 1/ (1–P_f)
- “If 1-in-10 get through without error, then average 10 tries to success”
- Average total time per frame is then t_0/(1 – P_f) (a proper derivation is given in next two slides)

$$\eta_{SW}=\frac{R_{eff}}{R}=\frac{\dfrac{n_f-n_o}{{}^{t_0}\!/_{1-P_f}}}{R}=\frac{1-\dfrac{n_o}{n_f}}{1+\dfrac{n_a}{n_f}+\dfrac{2(t_{prop}+t_{proc})R}{n_f}}\boxed{(1-P_f)}$$

Effect of
frame loss

- Let *X* be a random variable assuming the values of x_1, x_2,… with corresponding probabilities p_1, p_2,.... The mean or *expected* value of *X* is defined by:

$$E(X) = \sum_i p_i x_i$$

- e.g. $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, each with probability of 1/3
  $E(X) = 1/3 + 2/3 + 3/3 = (1+2+3)/3 = 2$, which is the simple average formula when *X* takes each value with equal probability

1 successful transmission

i – 1 unsuccessful transmissions

$$E[t_{total}] = t_0 + \sum_{i=1}^{\infty} (i-1) t_{out} P[n_t = i]$$

$$= t_0 + \sum_{i=1}^{\infty} (i-1) t_{out} P_f^{\,i-1} (1-P_f)$$

$$= t_0 + \frac{t_{out} P_f}{1-P_f} = t_0 \frac{1}{1-P_f}.$$

Efficiency:

$$\eta_{SW} = \frac{\dfrac{n_f - n_o}{E[t_{total}]}}{R} = (1-P_f) \frac{1 - \dfrac{n_o}{n_f}}{1 + \dfrac{n_a}{n_f} + \dfrac{2(t_{prop} + t_{proc})R}{n_f}} = (1-P_f)\eta_0.$$

impact of bit error rate

n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

Find efficiency for random bit errors with p = 0, 10^{-6}, 10^{-5}, 10^{-4}

$$1 - P_f = (1 - p)^{n_f} \approx e^{-n_f p} \quad \text{for large } n_f \text{ and small } p$$

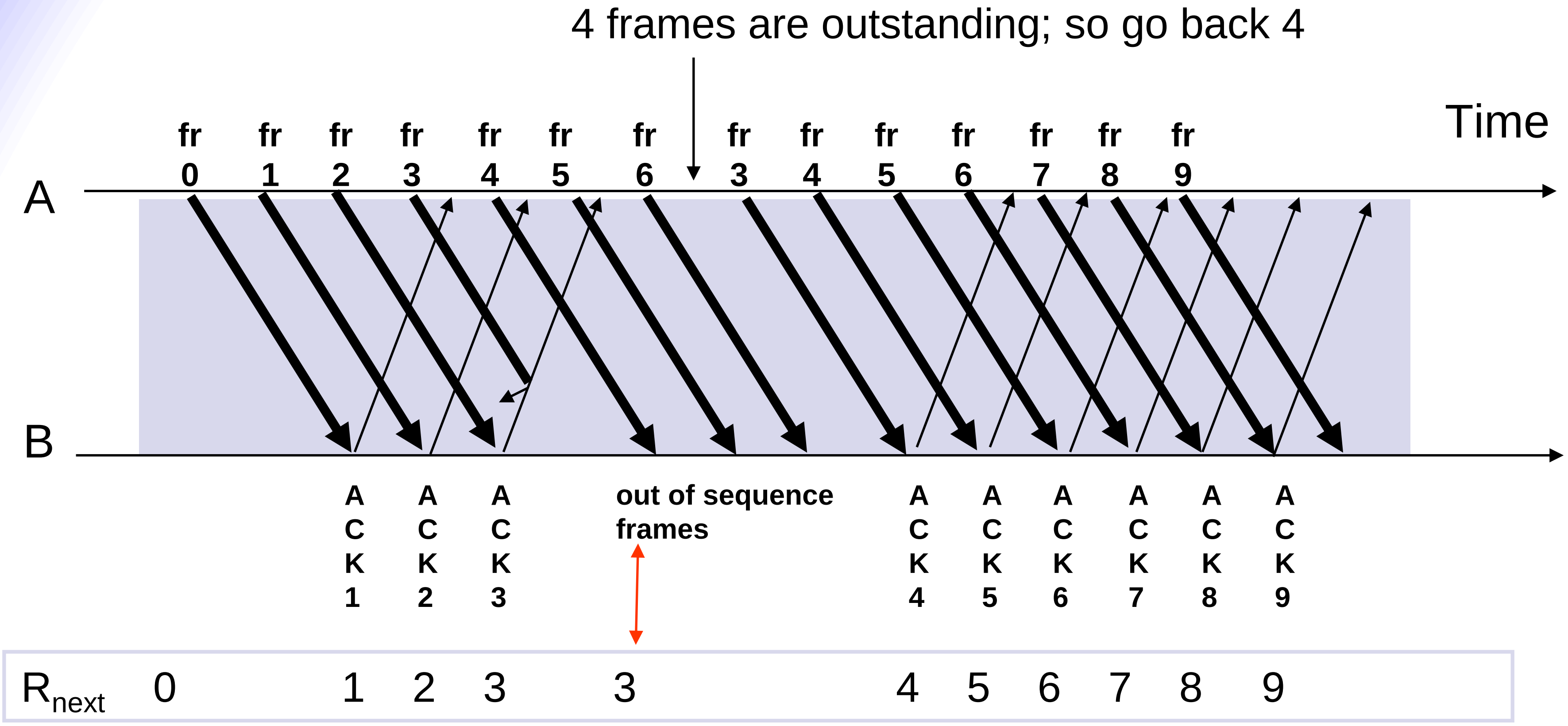
$1 – P_f$ **Efficiency**	0	10^{-6}	10^{-5}	10^{-4}
1 Mbps	1	0.99	0.905	0.368
& 1 ms	88%	86.6%	79.2%	32.2%

bit error impact performance as $n_f p$ approach 1

3.3.2 Go-Back-N ARQ (GBN)

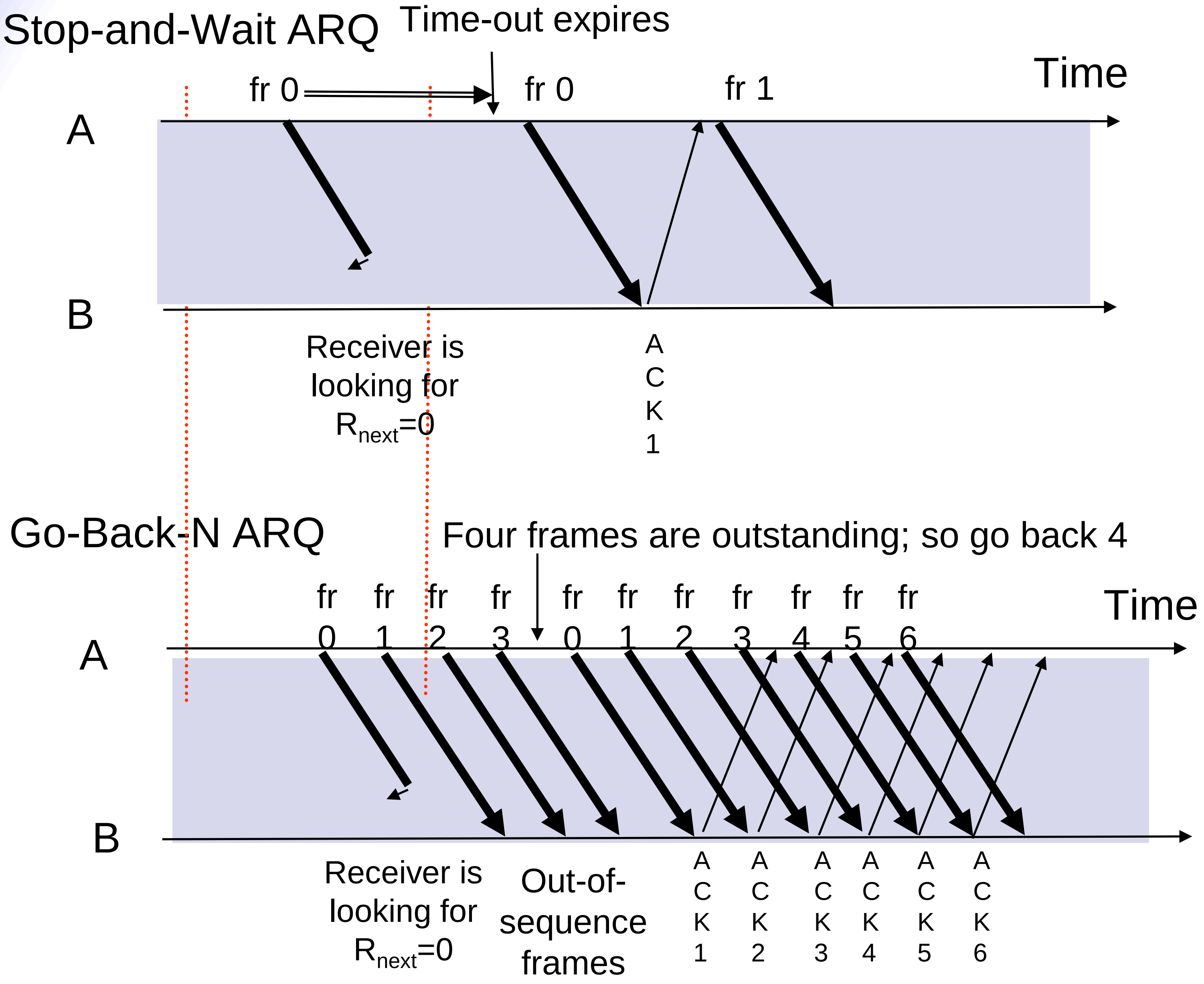
- Improve Stop-and-Wait by not waiting!
- Keep channel busy by continuing to send frames
- Allow a window of up to W_s outstanding frames
- Use *m*-bit sequence numbering
- Primitive version
  - If ACK for oldest frame arrives before window is exhausted, we can continue transmitting
  - If window is exhausted, pull back and retransmit all outstanding frames

Go-Back-4



- Frame transmission are *pipelined* to keep the channel busy
- Frame with errors and subsequent out-of-sequence frames are ignored
- Transmitter is forced to go back when window of 4 is exhausted

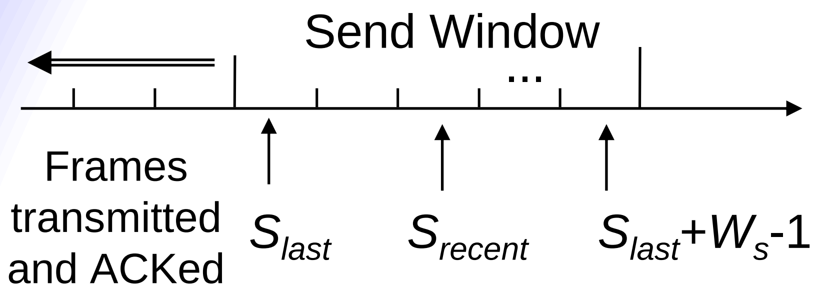
Window size long enough to cover round trip time



Alternative: use timeout

- Problem with the primitive Go-Back-N as presented:
  - If frame is lost and source does not have frame to send, then window will not be exhausted and recovery will not commence
- Use a timeout with each frame
  - When timeout expires, resend all outstanding frames

Transmitter



Send Window

...

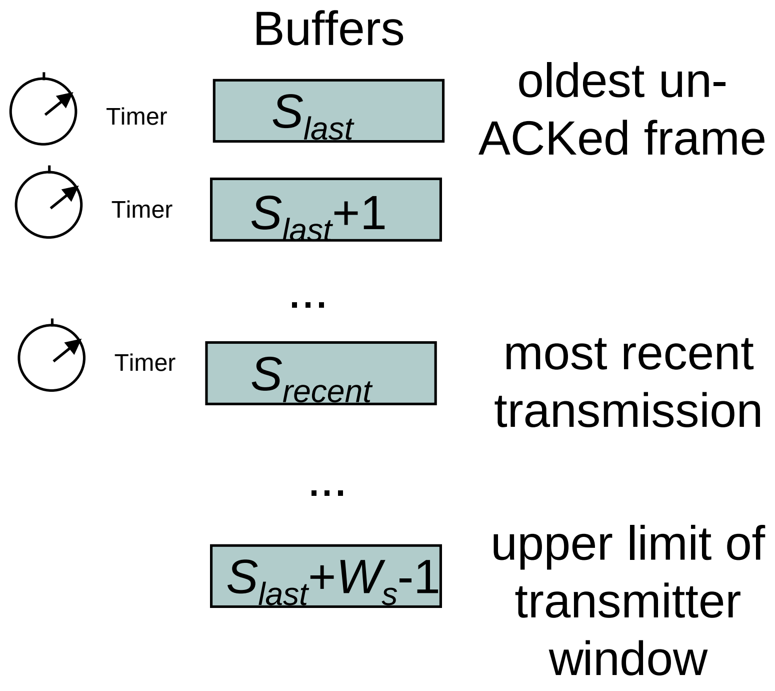
Frames transmitted and ACKed

S_{last}

S_{recent}

$S_{last}+W_s-1$

Buffers



Timer

S_{last}

oldest un-ACKed frame

Timer

$S_{last}+1$

...

Timer

S_{recent}

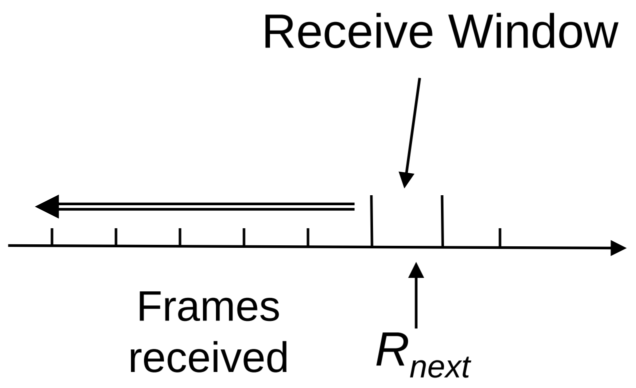
most recent transmission

...

$S_{last}+W_s-1$

upper limit of transmitter window

Receiver



Receive Window

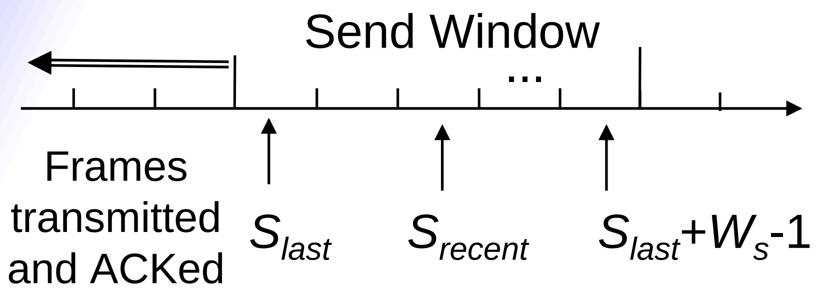
Frames received

R_{next}

Receiver will only accept
a frame that is error-free and
that has sequence number R_{next}

When such frame arrives R_{next} is incremented by one, so the receive window slides forward by one

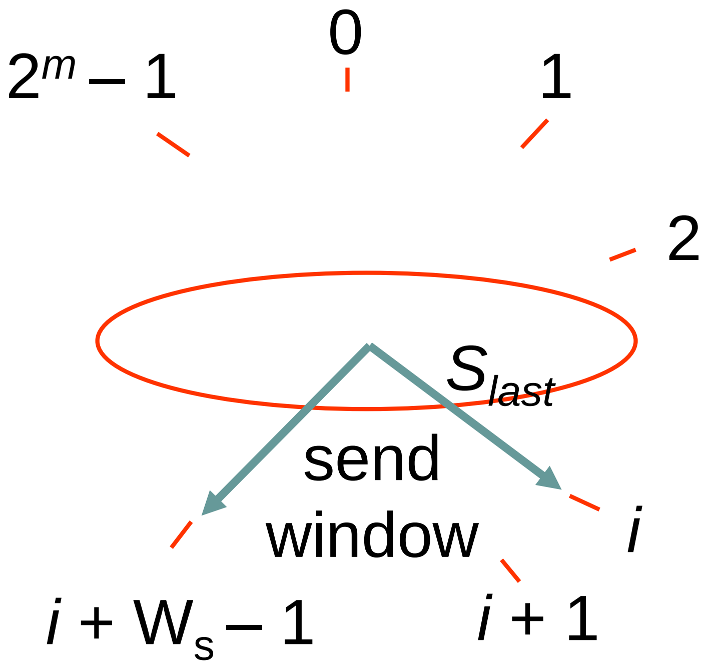
Transmitter



Transmitter waits for error-free ACK frame with sequence number S_{last}

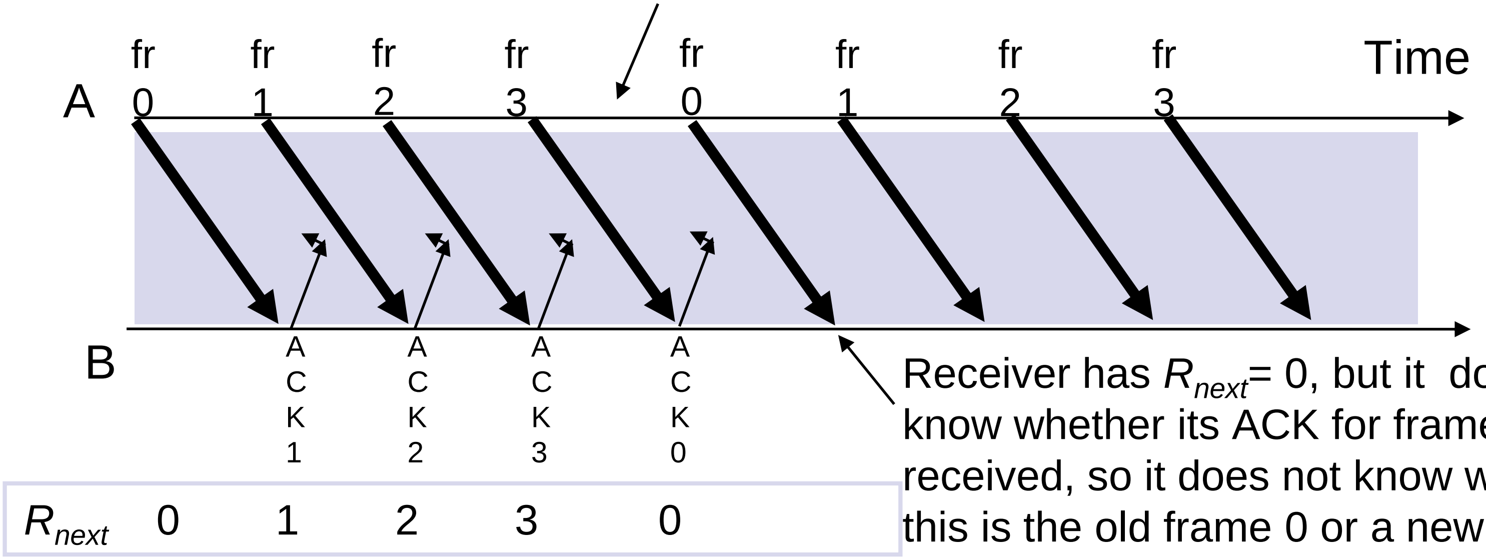
When such ACK frame arrives, S_{last} is incremented by one, and the send window slides forward by one

m-bit Sequence Numbering



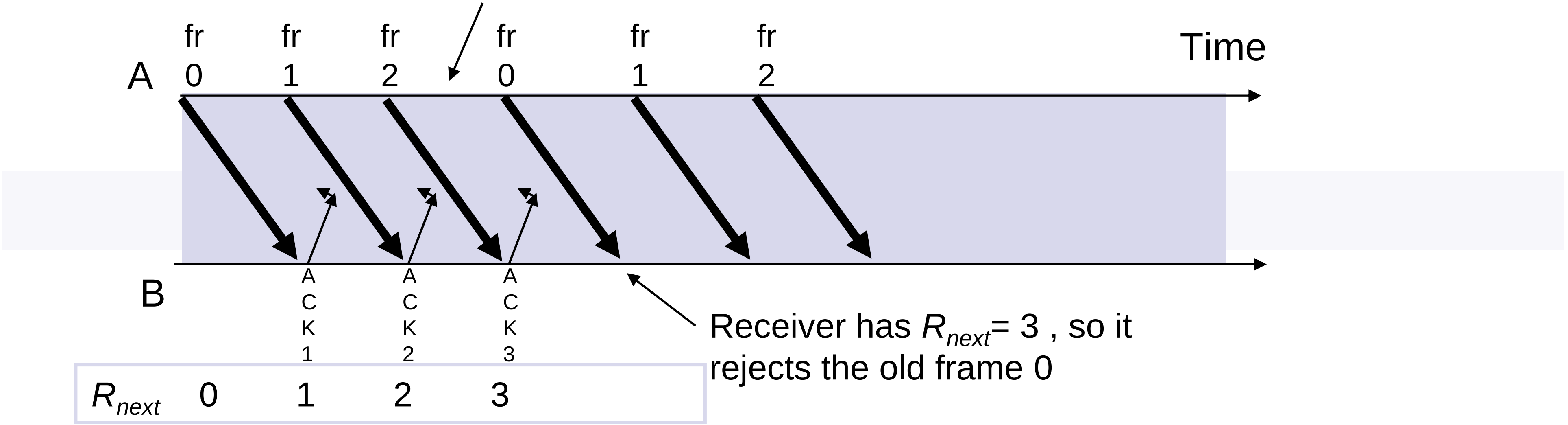
Maximum Allowable Window Size is $W_s = 2^m$-1

M = 2^2 = 4, Go-Back - 4: Transmitter goes back 4

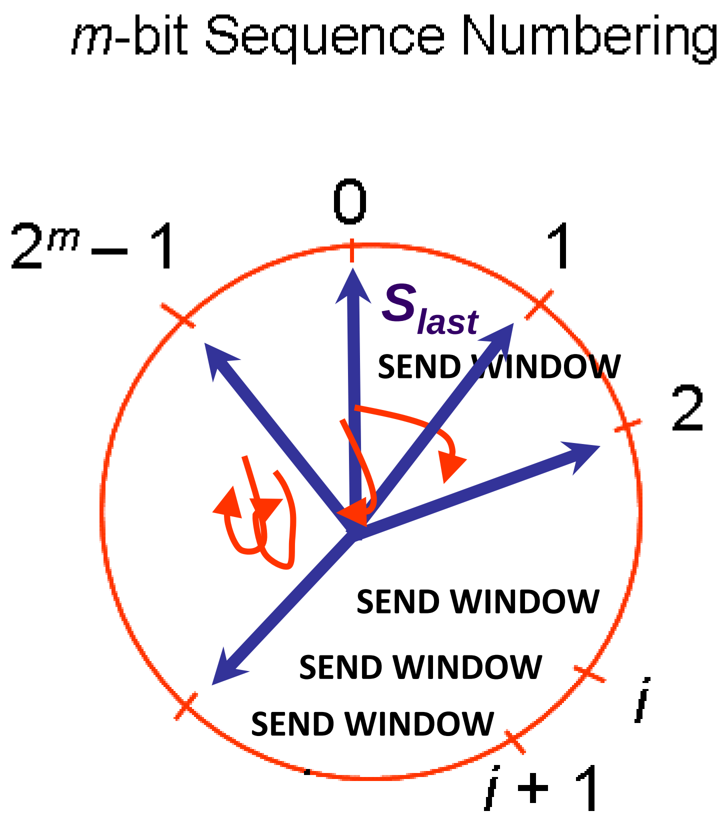


Receiver has R_{next}= 0, but it does not know whether its ACK for frame 0 was received, so it does not know whether this is the old frame 0 or a new frame 0

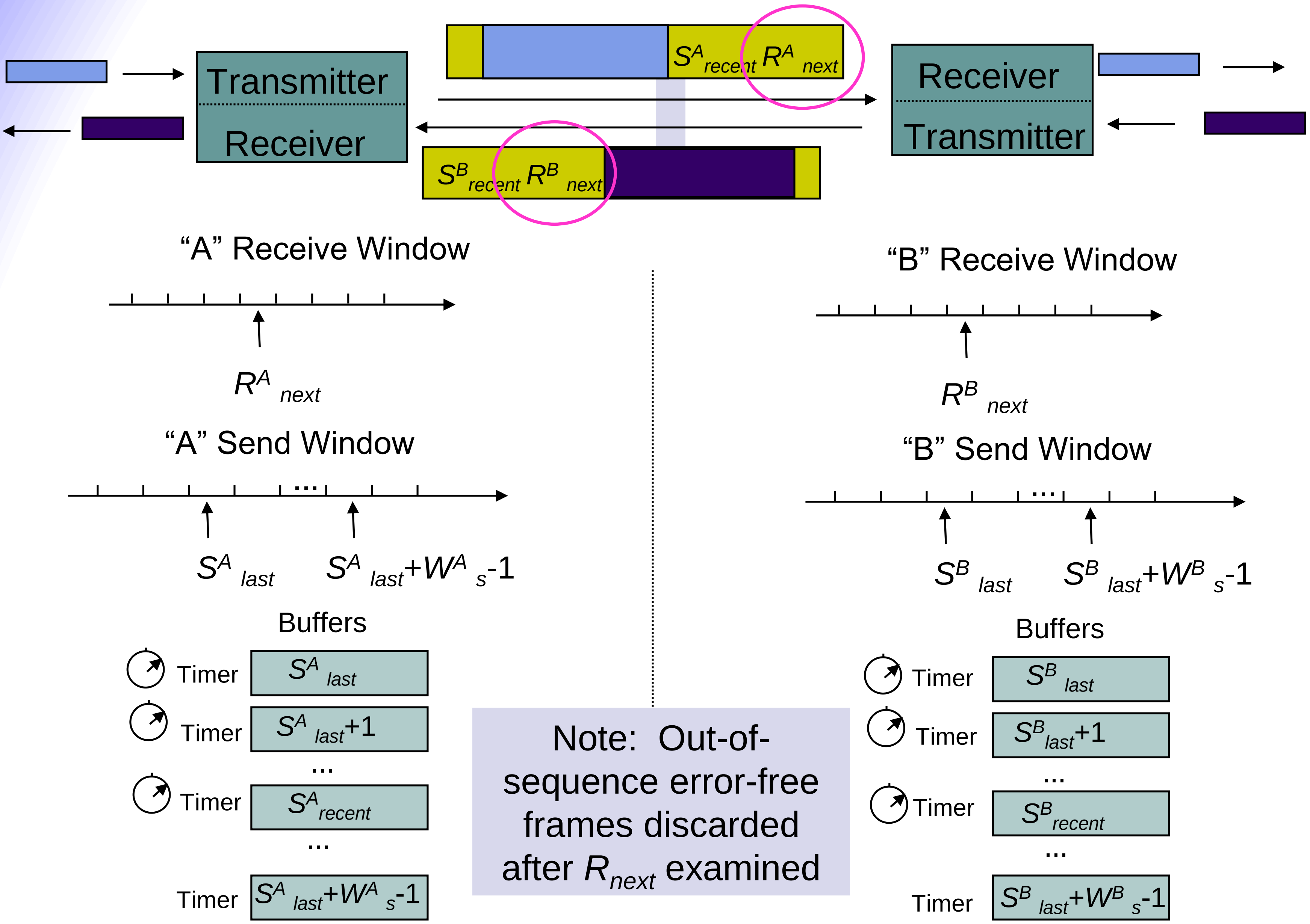
M = 2^2 = 4, Go-Back-3: Transmitter goes back 3



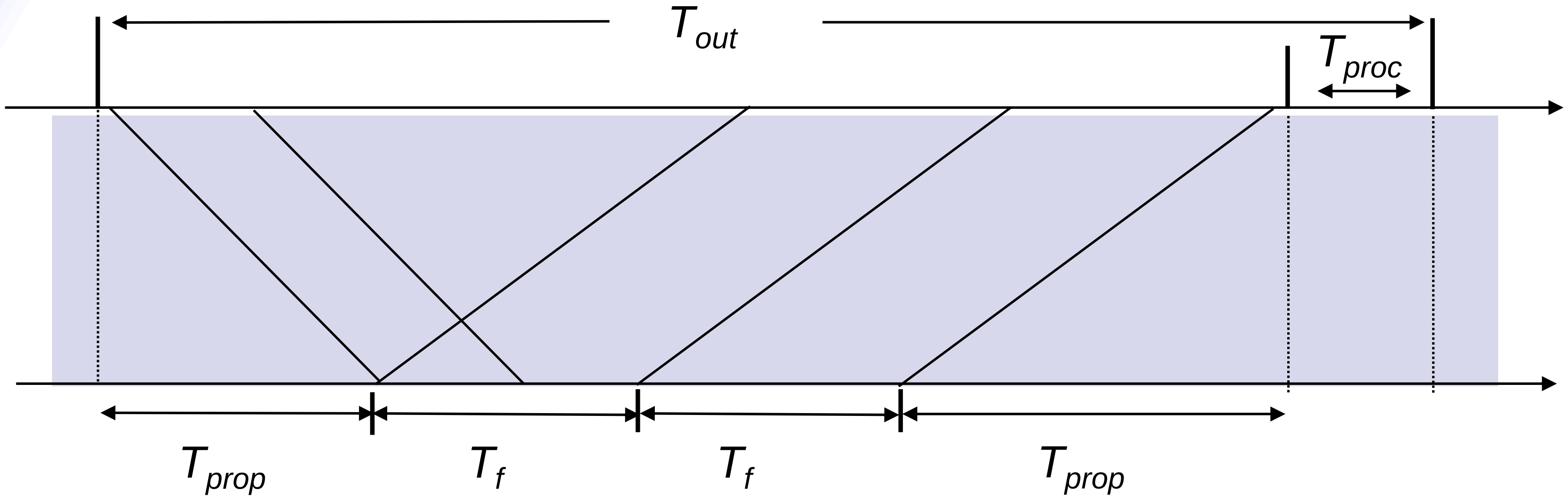
Receiver has R_{next}= 3 , so it rejects the old frame 0


m-bit Sequence Numbering
0
$2^m - 1$
1
2
S_{last}
SEND WINDOW
SEND WINDOW
SEND WINDOW
SEND WINDOW
i
i + 1

ACK Piggybacking in bidirectional GBN



Note: Out-of-sequence error-free frames discarded after R_{next} examined



- Timeout value should allow for:
  - Two propagation times + 1 processing time: 2 T_{prop} + T_{proc}
  - A frame that begins transmission right before our frame arrives T_f
  - Next frame carries the ACK, T_f
- W_s should be large enough to keep channel busy for T_{out}

applications

- High-Level Data Link Control (HDLC): bit-oriented data link control
- V.42 modem: error control over telephone modem links

performance

Frame = 1250 bytes =10,000 bits, *R* = 1 Mbps		
$2(T_{prop} + T_{proc})$	**2 x Delay x BW**	**Window**
1 ms	1000 bits	1
10 ms	10,000 bits	2
100 ms	100,000 bits	11
1 second	1,000,000 bits	101

- GBN is completely efficient, if W_s large enough to keep channel busy, and if channel is error-free
- Assume P_f frame loss probability, then time to deliver a frame is:
  - t_f if first frame transmission succeeds $(1 – P_f)$
  - $t_f + W_s t_f/(1\text{-}P_f)$ if the first transmission does not succeed P_f

$$t_{GBN} = t_f(1 - P_f) + P_f\{t_f + \frac{W_s t_f}{1 - P_f}\} = t_f + P_f \frac{W_s t_f}{1 - P_f} \quad \text{and}$$

$$\eta_{GBN} = \frac{\dfrac{n_f - n_o}{t_{GBN}}}{R} = \frac{1 - \dfrac{n_o}{n_f}}{1 + (W_s - 1)P_f}(1 - P_f)$$

Delay-bandwidth product determines W_s

1 successful transmission

i – 1 unsuccessful transmissions

$$E[t_{total}] = t_f + \sum_{i=1}^{\infty} (i-1) W_s t_f P[n_t = i]$$

$$= t_f + W_s t_f \sum_{i=1}^{\infty} (i-1) P_f^{\,i-1} (1 - P_f)$$

$$= t_f + \frac{W_s t_f P_f}{1 - P_f} = t_f \, \frac{1 + (W_s - 1) P_f}{1 - P_f}.$$

Efficiency:

$$\eta_{GBN} = \frac{\dfrac{n_f - n_o}{E[t_{total}]}}{R} = (1 - P_f) \frac{1 - \dfrac{n_o}{n_f}}{1 + (W_s - 1) P_f}.$$

impact of bit error rate

n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

compare SW with GBN efficiency for random bit errors with p = 0, 10^{-6}, 10^{-5}, 10^{-4} and R = 1 Mbps and 100 ms

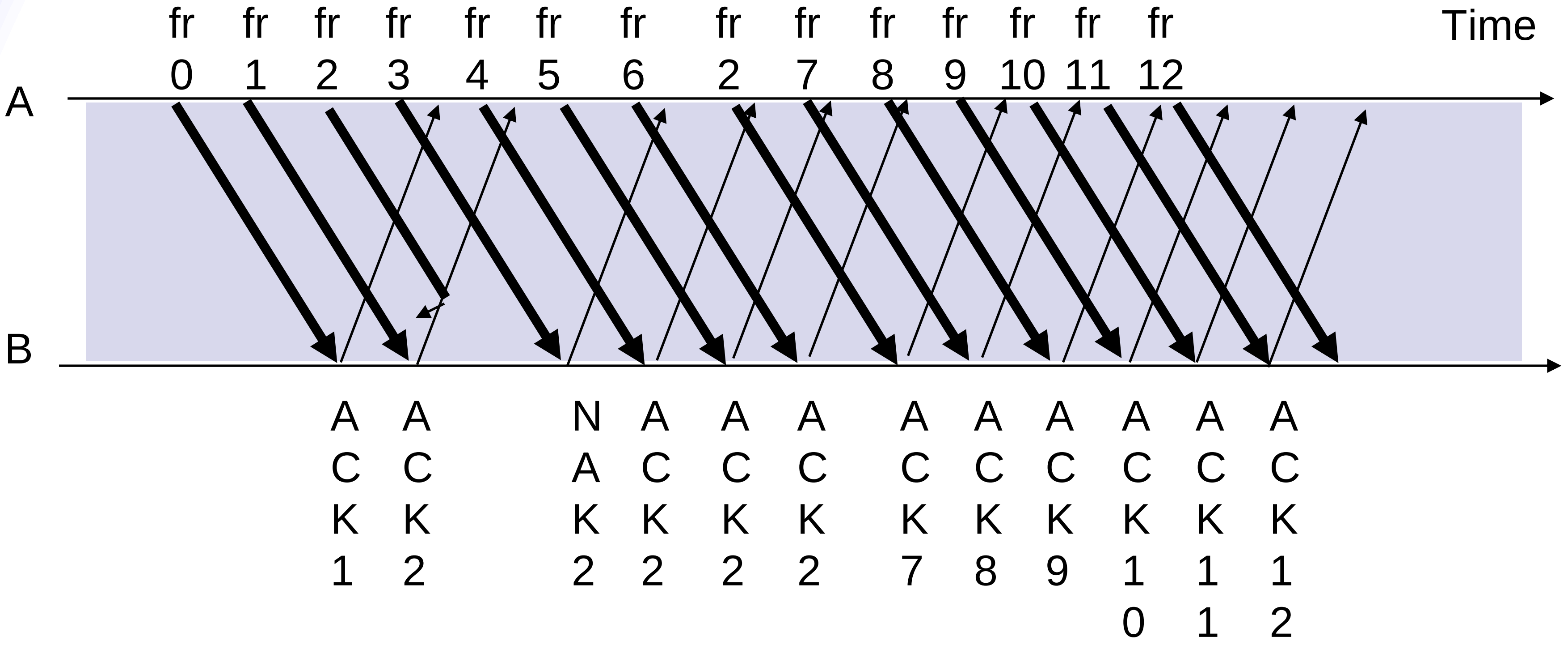
1 Mbps x 100 ms = 100,000 bits = 10 frames → Use W_s = 11

Efficiency	0	10^{-6}	10^{-5}	10^{-4}
SW	8.9%	8.8%	8.0%	3.3%
GBN	98%	88.2%	45.4%	4.9%

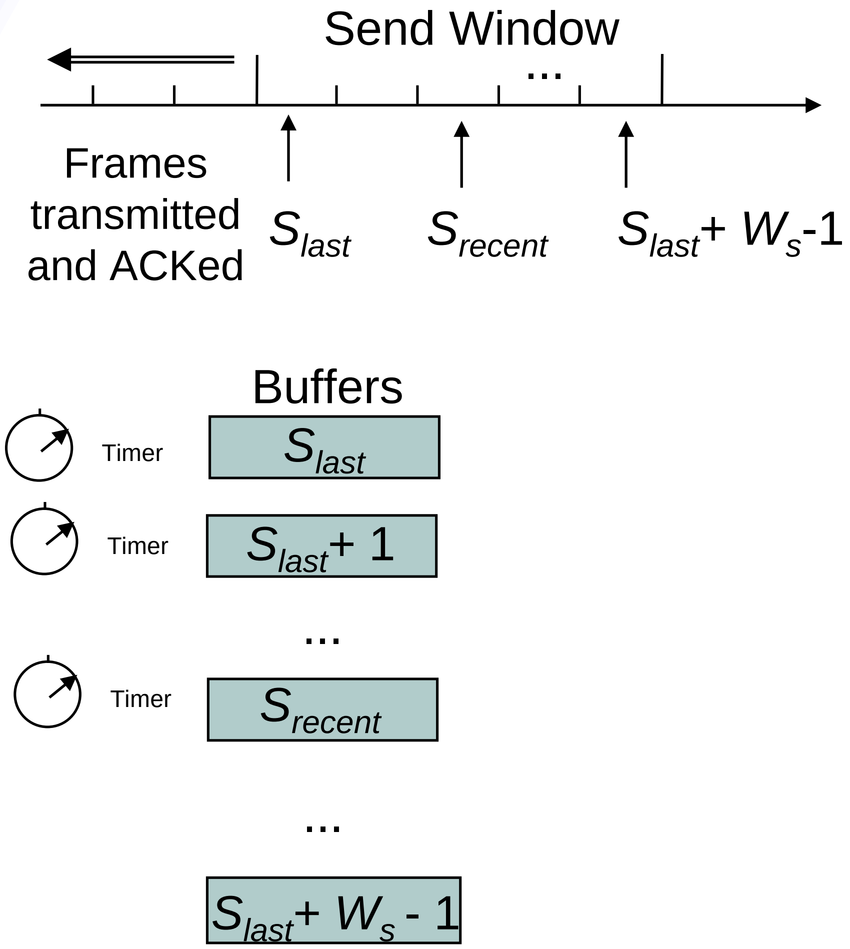
- Go-Back-N significant improvement over Stop-and-Wait for large delay-bandwidth product
- Go-Back-N becomes inefficient as error rate increases

3.3.3 Selective Repeat ARQ (SR)

- Go-Back-N ARQ inefficient because *multiple* frames are resent when errors or losses occur
- Selective Repeat ARQ retransmits *only an individual frame*
  - Timeout causes individual corresponding frame to be resent
  - NAK causes retransmission of oldest un-acked frame
- Receiver maintains a *receive window* of sequence numbers that can be accepted
  - Error-free, but out-of-sequence frames with sequence numbers within the receive window are buffered
  - Arrival of frame with R_{next} causes window to slide forward by 1 or more


fr 0
fr 1
fr 2
fr 3
fr 4
fr 5
fr 6
fr 2
fr 7
fr 8
fr 9
fr 10
fr 11
fr 12
Time
A
B
ACK 1
ACK 2
NAK 2
ACK 2
ACK 2
ACK 2
ACK 7
ACK 8
ACK 9
ACK 10
ACK 11
ACK 12

Transmitter



Send Window

...

Frames transmitted and ACKed

S_{last}

S_{recent}

$S_{last} + W_s$-1

Buffers

Timer S_{last}

Timer S_{last}+ 1

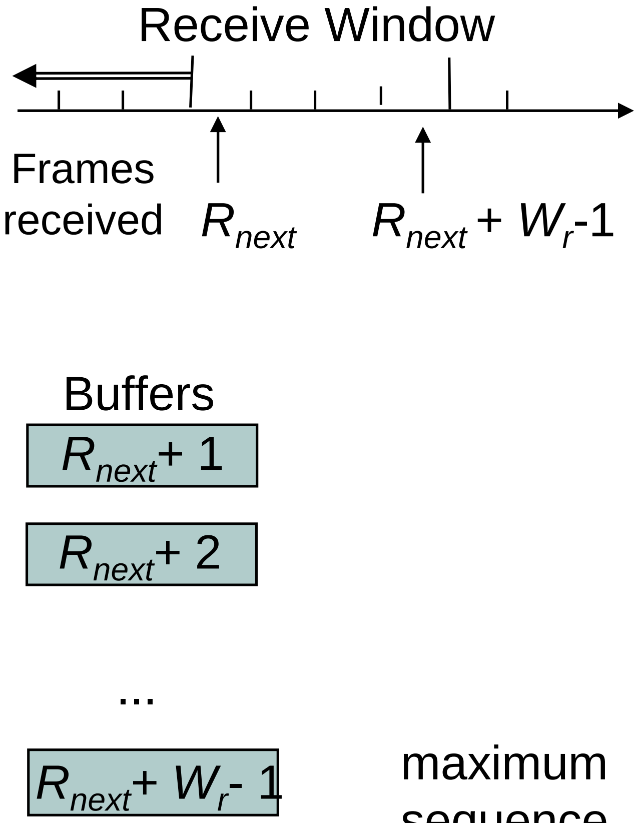
...

Timer S_{recent}

...

S_{last}+ W_s - 1

Receiver



Receive Window

Frames received

R_{next}

R_{next} + W_r-1

Buffers

R_{next}+ 1

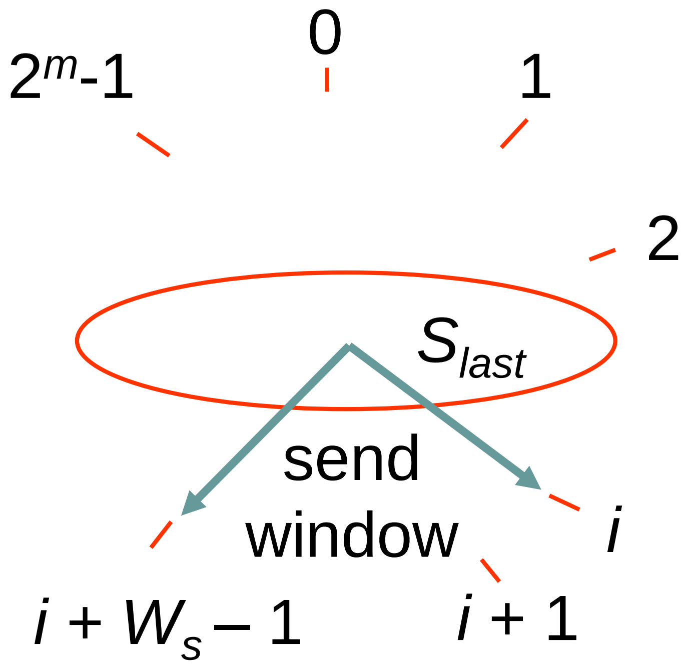
R_{next}+ 2

...

R_{next}+ W_r- 1

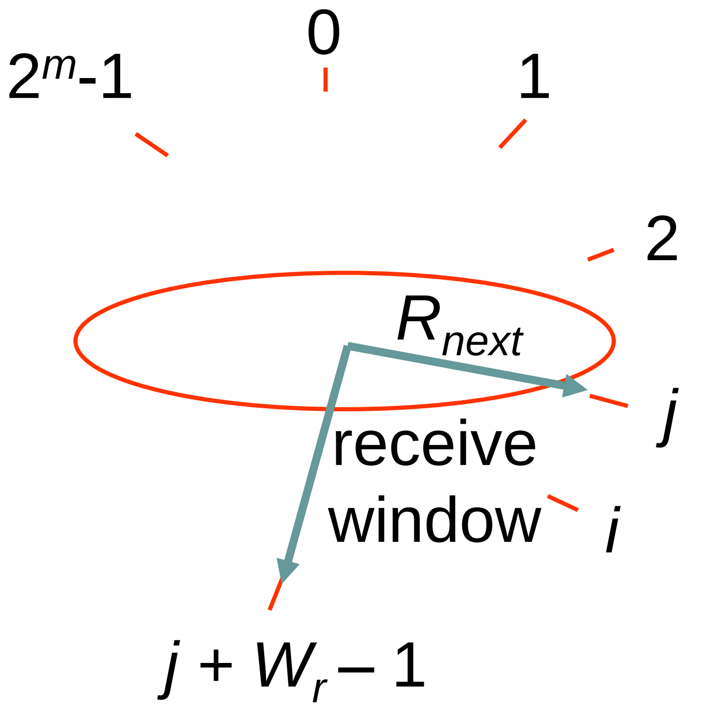
maximum sequence number accepted

Transmitter



Moves k forward when ACK arrives with $R_{next} = S_{last} + k$
$k = 1, \ldots, W_s$-1

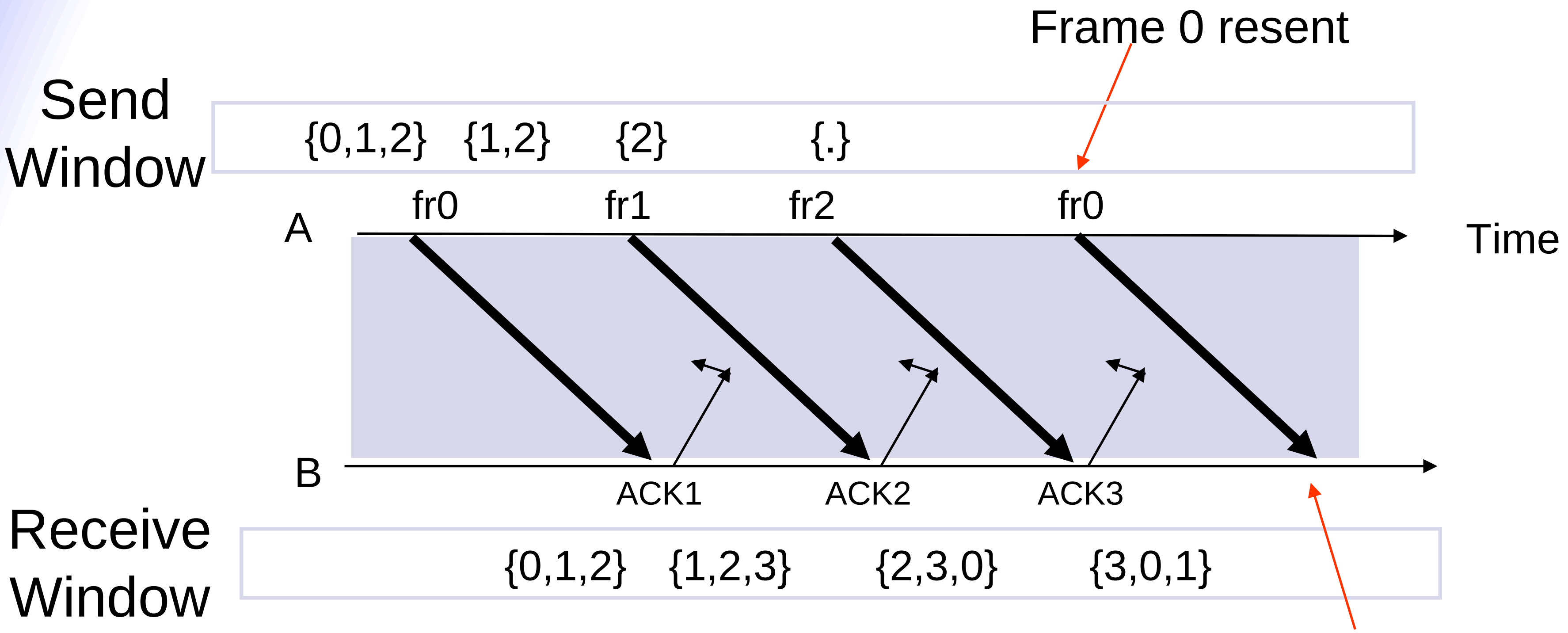
Receiver



Moves forward by 1 or more when frame arrives with sequence number = R_{next}

What size W_s and W_r allowed?

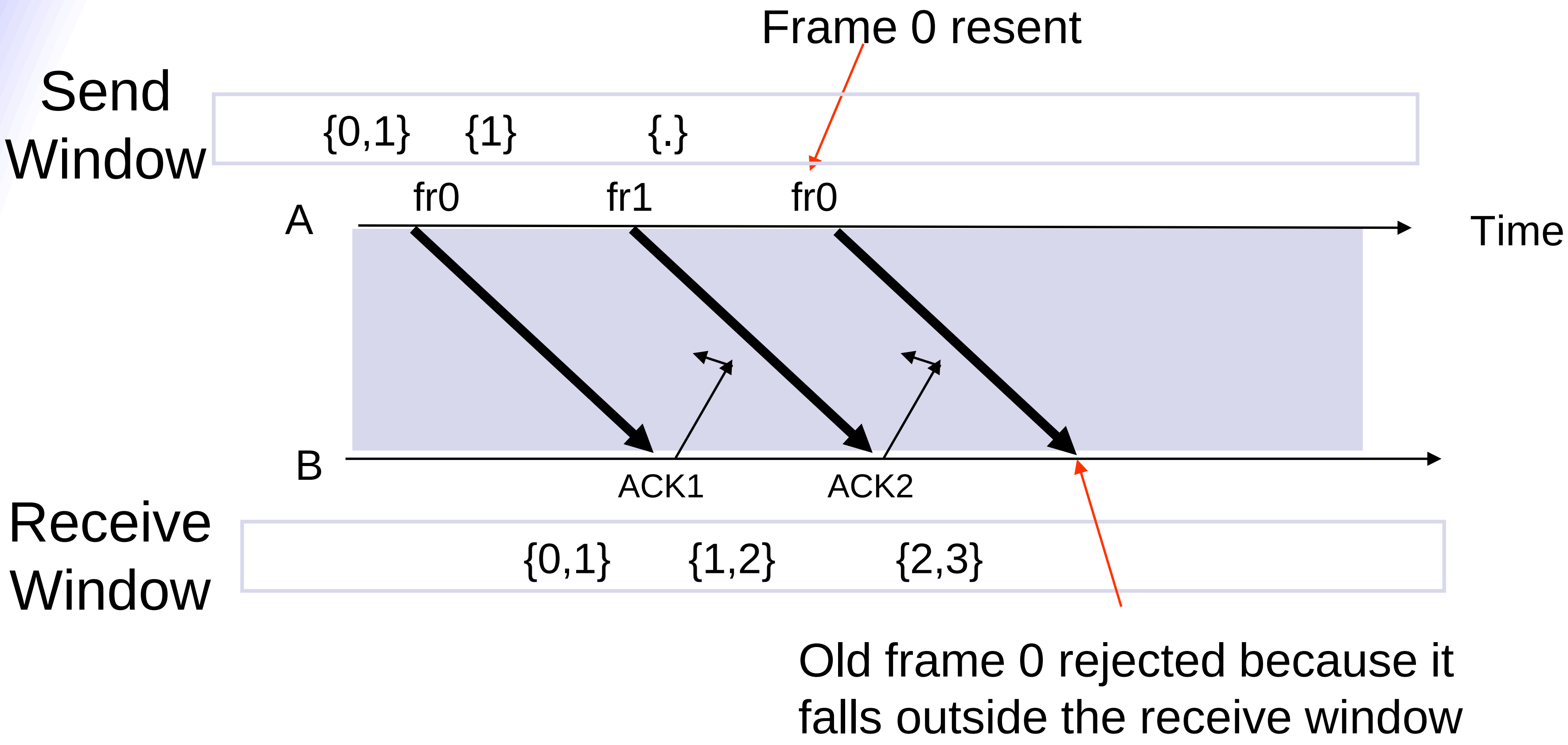
Example: $M=2^2=4$, $W_s=3$, $W_r=3$



Old frame 0 accepted as a new frame because it falls in the receive window

$W_s + W_r = 2^m$ is maximum allowed

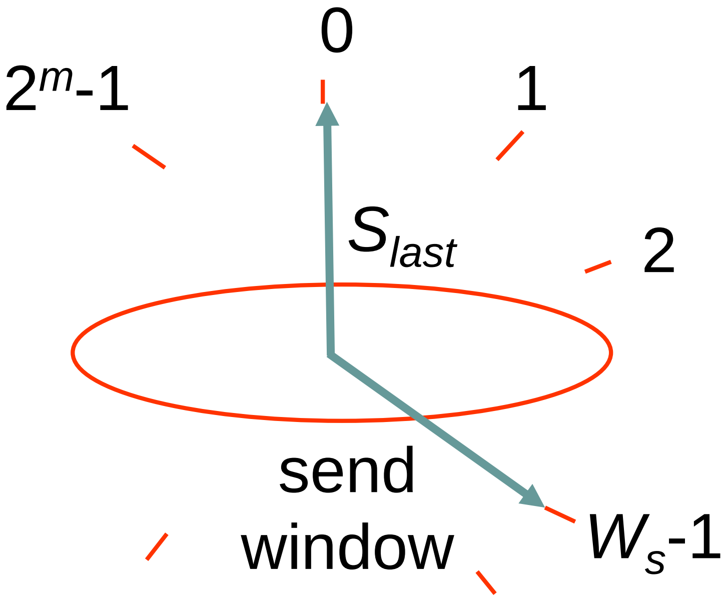
Example: $M=2^2=4$, $W_s=2$, $W_r=2$

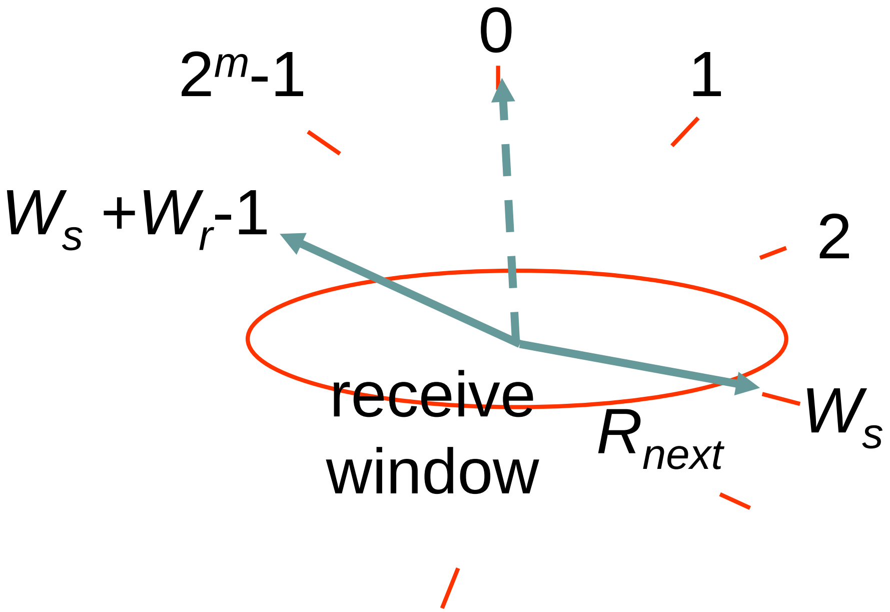


Why $W_s + W_r = 2^m$ works?

- Transmitter sends frames 0 to W_s-1; send window empty
- All arrive at receiver
- All ACKs lost
- Transmitter resends frame 0

- Receiver window starts at {0, …, W_r-1}
- Window slides forward to {W_s,…,W_s+W_r-1}
- Receiver rejects frame 0 because it is outside receive window





applications

- Transmission Control Protocol (TCP): transport layer protocol uses variation of selective repeat to provide reliable stream service
- Service Specific Connection Oriented Protocol: error control for signaling messages in asynchronous transfer mode (ATM) networks

performance

- Assume P_f frame loss probability, then number of transmissions required to deliver a frame is:
  - t_f/ (1-P_f)

$$\eta_{SR} = \frac{\dfrac{n_f - n_o}{t_f /(1 - P_f)}}{R} = (1 - \frac{n_o}{n_f})(1 - P_f)$$

impact of bit error rate

n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

compare SW, GBN and SR efficiency for random bit errors with p=0, 10^{-6}, 10^{-5}, 10^{-4} and R = 1 Mbps and 100 ms

Efficiency	0	10^{-6}	10^{-5}	10^{-4}
SW	8.9%	8.8%	8.0%	3.3%
GBN	98%	88.2%	45.4%	4.9%
SR	98%	97%	89%	36%

- SR outperforms GBN and SW, but efficiency drops as error rate increases

Assume n_a and n_o are negligible relative to n_f, and
$L = 2(t_{prop}+t_{proc})R/n_f$ =(W$_s$-1), then

Selective Repeat:

$$\eta_{SR} = (1-P_f)(1-\frac{n_o}{n_f}) \approx (1-P_f)$$

Go-Back-N:

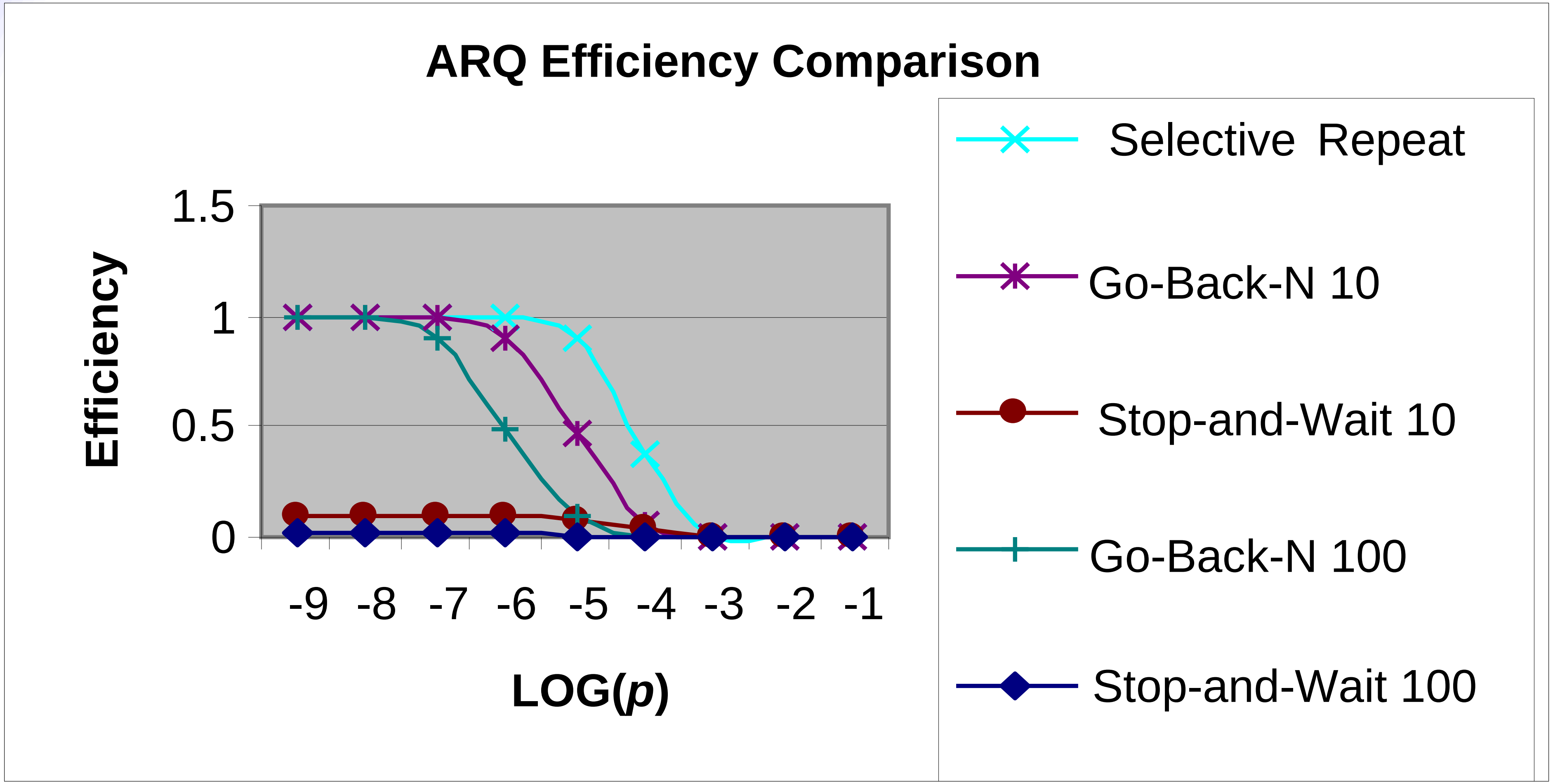
For $P_f\approx 0$, SR and GBN same

$$\eta_{GBN} = \frac{1-P_f}{1+(W_S-1)P_f} = \frac{1-P_f}{1+LP_f}$$

Stop-and-Wait:

For $P_f \longrightarrow 1$, GBN and SW same

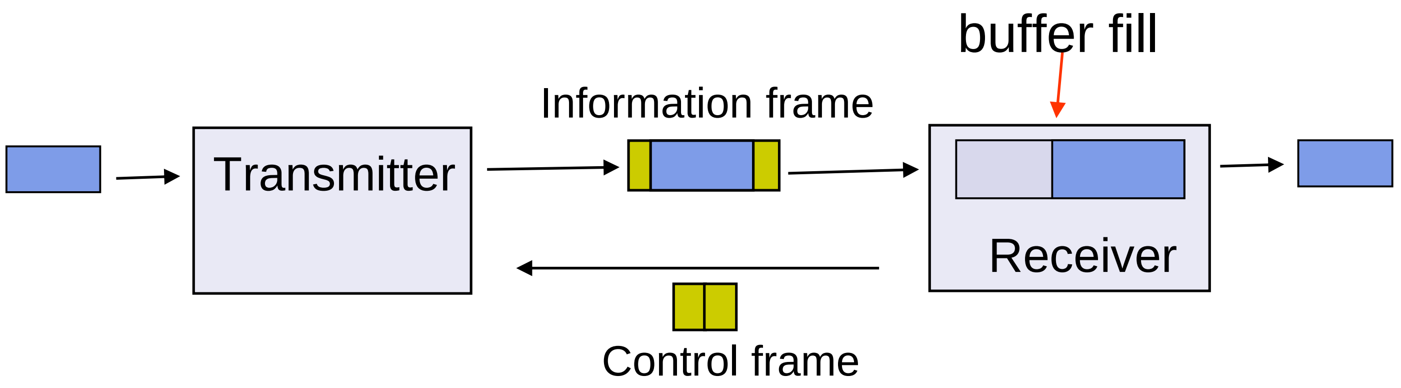
$$\eta_{SW} = \frac{(1-P_f)}{1+\frac{n_a}{n_f}+\frac{2(t_{prop}+t_{proc})R}{n_f}} \approx \frac{1-P_f}{1+L}$$



delay-bandwidth product = 10, 100 frames

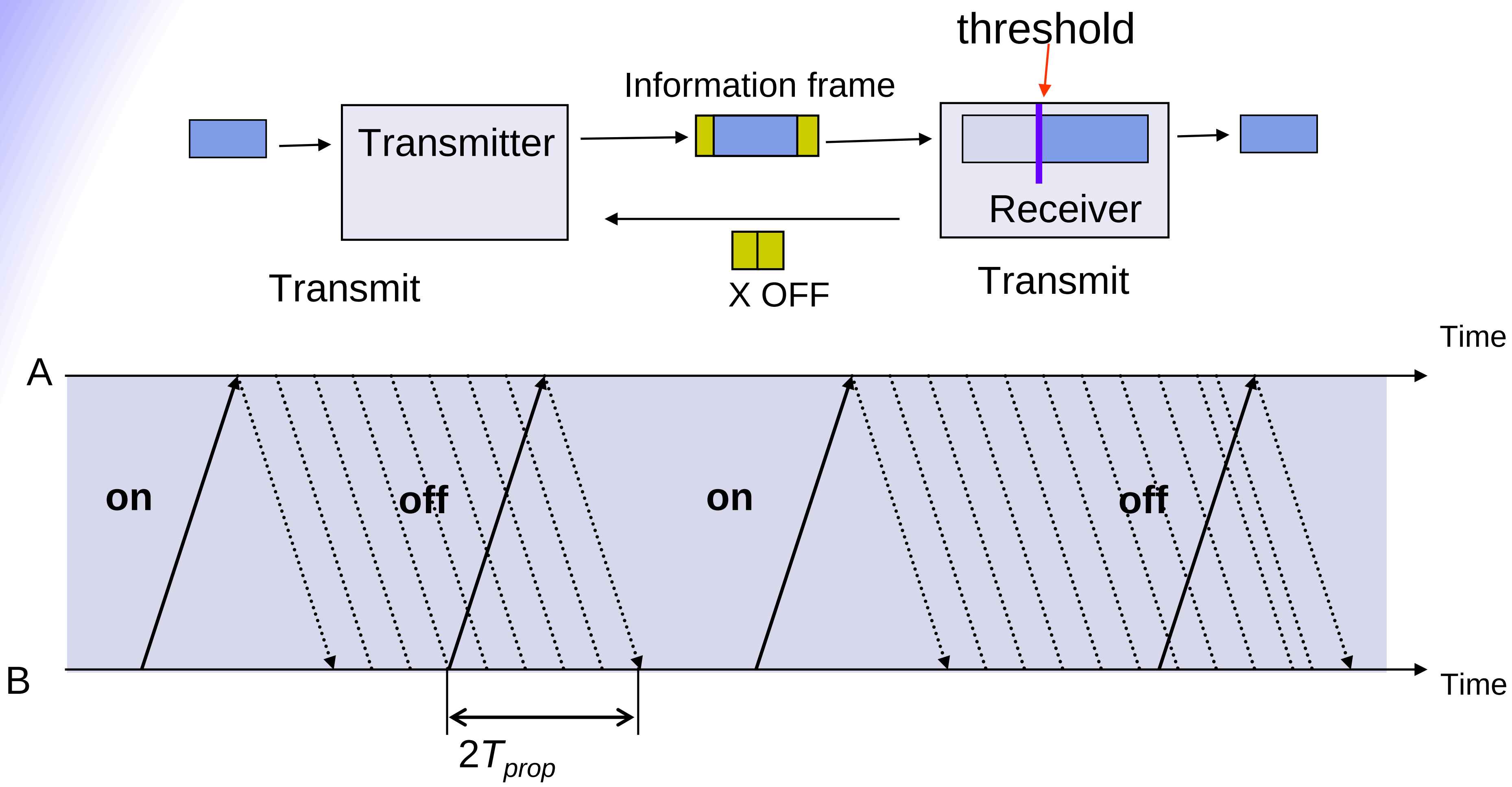
3.4 Flow control

- Messages can be lost if receiving system does not have sufficient buffering to store arriving messages
- If destination layer-(*n*+1) does not retrieve its information fast enough, destination layer-*n* buffers may overflow
- Pacing and flow control provide backpressure mechanisms that control transfer according to availability of buffers at the destination
- Examples: TCP and HDLC



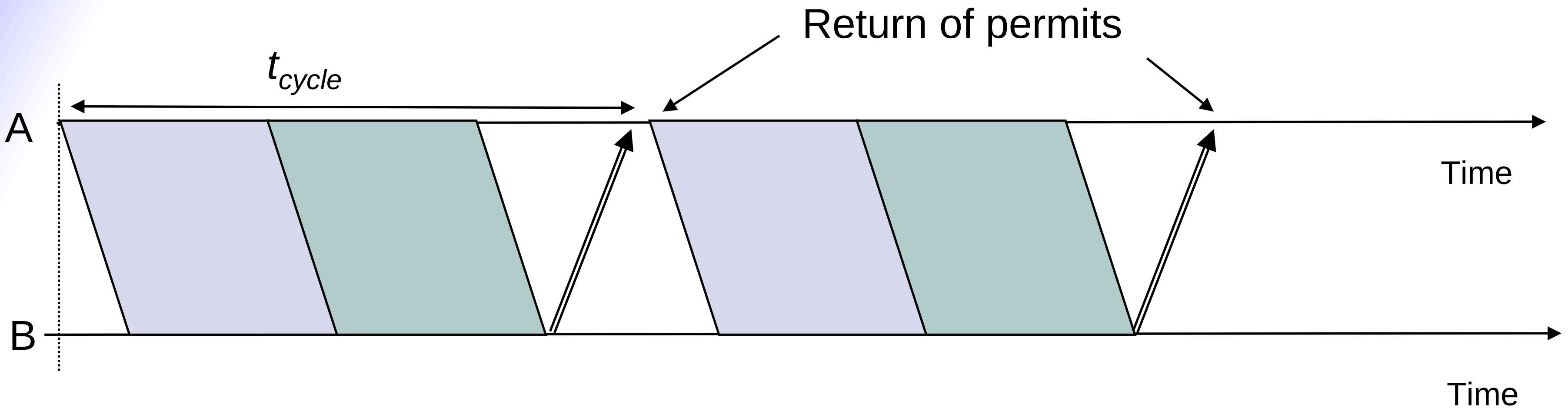
- Receiver has limited buffering to store arriving frames
- Several situations cause buffer overflow
  - Mismatch between sending rate and rate at which user can retrieve data
  - Surges in frame arrivals
- Flow control prevents buffer overflow by regulating rate at which source is allowed to send information

ON-OFF flow control



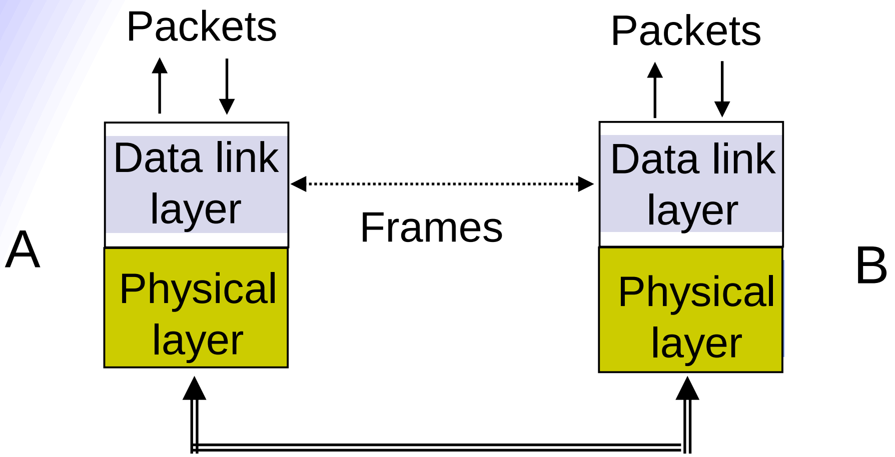
Threshold must activate OFF signal while 2 T_{prop} R bits still remain in buffer

Sliding window flow control



- Sliding window ARQ method with W_s equal to buffer available
  - Transmitter can never send more than W_s frames
- ACKs that slide window forward can be viewed as permits to transmit more
- Can also pace ACKs as shown above
  - Return permits (ACKs) at end of cycle regulates transmission rate

Data Link Protocols



- Directly connected, wire-like
- Losses & errors, but no out-of-sequence frames
- Applications: Direct Links; LANs; Connections across WANs

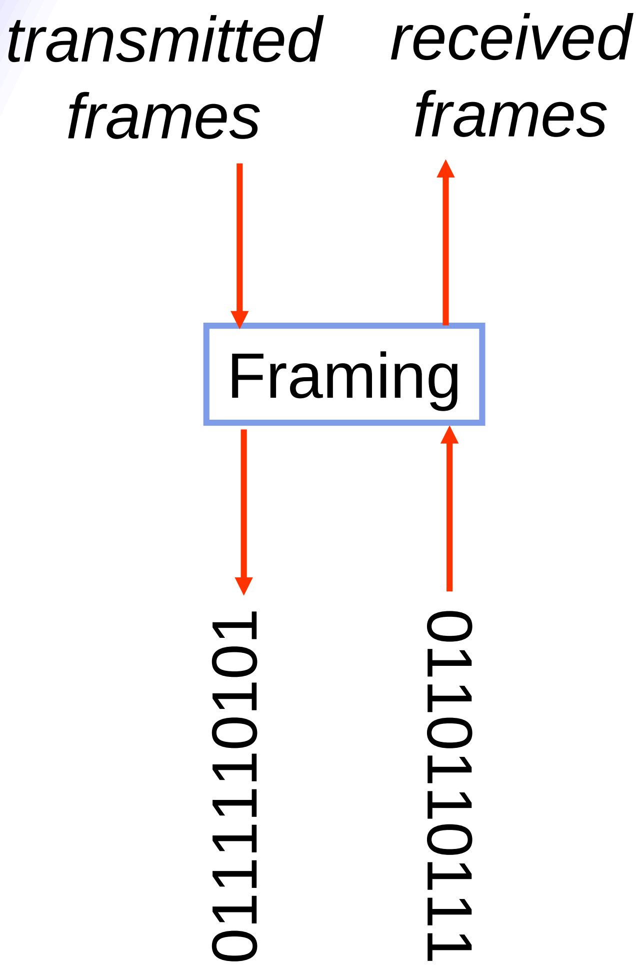
Data Links Services

- Framing
- Error control
- Flow control
- Multiplexing
- Link Maintenance
- Security: Authentication & Encryption

Examples

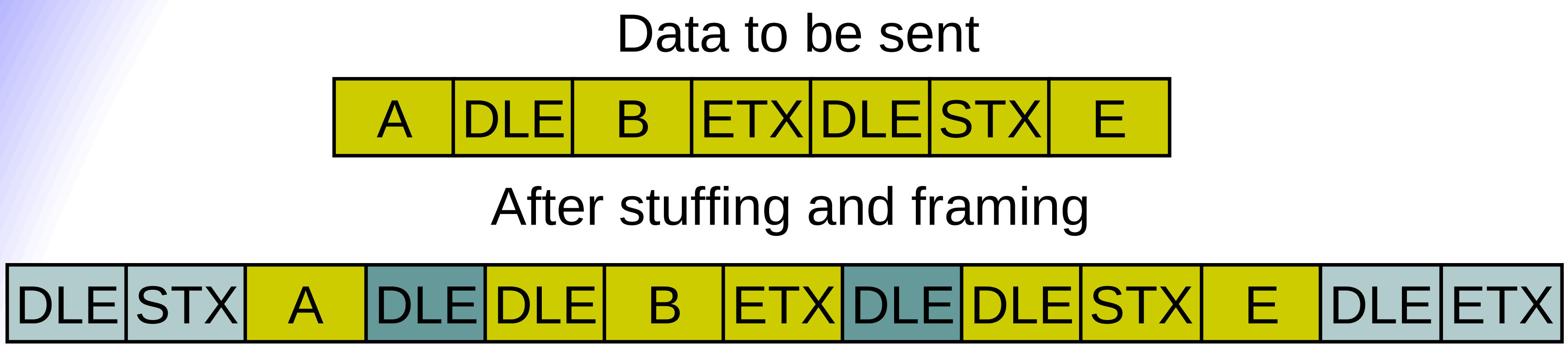
- PPP
- HDLC
- Ethernet LAN
- IEEE 802.11 (Wi Fi) LAN

3.5 Framing



- Mapping stream of physical layer bits into frames
- Mapping frames into bit stream
- Frame boundaries can be determined using:
  - character counts
  - control characters
  - flags
  - CRC checks

byte stuffing



Data to be sent

A	DLE	B	ETX	DLE	STX	E

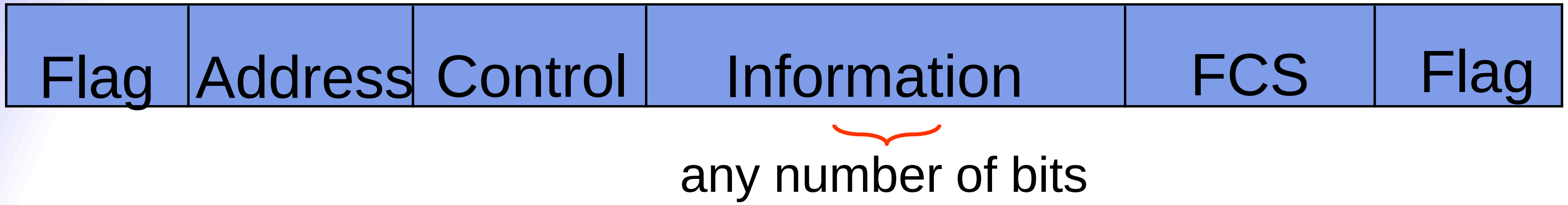
After stuffing and framing

DLE	STX	A	DLE	DLE	B	ETX	DLE	DLE	STX	E	DLE	ETX

- Frames consist of integer number of bytes
  - Asynchronous transmission systems using ASCII to transmit printable characters
  - Octets with HEX value < 20 are nonprintable
- Special 8-bit patterns used as control characters
  - STX (start of text) = 0x02; ETX (end of text) = 0x03
- Byte used to carry non-printable characters in frame
  - DLE (data link escape) = 0x10
  - DLE STX (DLE ETX) used to indicate beginning (end) of frame
  - Insert extra DLE in front of occurrence of DLE in frame – byte stuffing
  - All DLEs occur in pairs except at frame boundaries

bit stuffing

HDLC frame



Flag Address Control Information FCS Flag

any number of bits

- Frame delineated by flag character
- HDLC uses bit stuffing to prevent occurrence of flag 01111110 inside the frame
- Transmitter inserts extra 0 after each consecutive five 1s inside the frame
- Receiver checks for five consecutive 1s
  - if next bit = 0, it is removed
  - if next two bits are 10, then flag is detected
  - If next two bits are 11, then frame has errors

Example

(a)

Data to be sent

$$01101\underbrace{111}11\underbrace{111}100$$

After stuffing and framing

$$\mathit{01111110}011011111\underline{0}11111\underline{0}00\mathit{01111110}$$

(b)

Data received

$$01\underbrace{111}11000011101\underbrace{111}101\underbrace{111}1011001 1\underbrace{111}10$$

After destuffing and deframing

000111011111-11111-110

3.6 Point-to-point protocol (PPP)

- Data link protocol for point-to-point lines in Internet
  - Router-router; dial-up to router

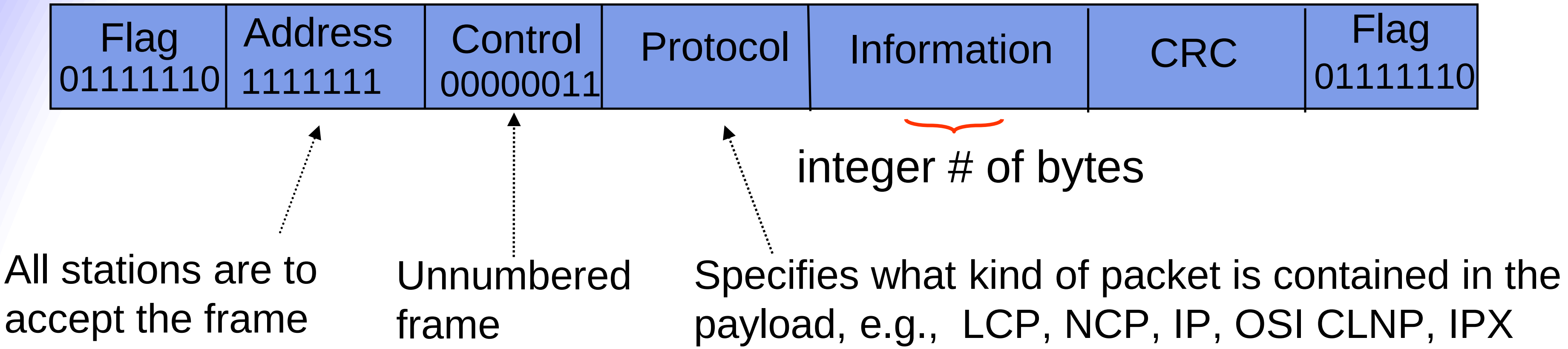
1. Provides framing and error detection
   - Character-oriented HDLC-like frame structure
2. Link control protocol (LCP)
   - Bringing up, testing, bringing down lines, negotiating options
   - **Authentication**: key capability in ISP access
3. A family of Network Control Protocols (NCP) specific to different network layer protocols
   - IP, OSI network layer, IPX (Novell), Appletalk

applications

PPP used in many point-to-point applications

- Telephone Modem Links 30 Kbps
- Packet over Synchronous Optical Network (SONET) 600 Mbps to 10 Gbps
  - IP→PPP→SONET

- PPP is also used over shared links such as Ethernet to provide LCP, NCP, and authentication features
  - PPP over Ethernet (RFC 2516)
  - Used over DSL



Flag	Address	Control	Protocol	Information	CRC	Flag
01111110	1111111	00000011				01111110

integer # of bytes

All stations are to accept the frame

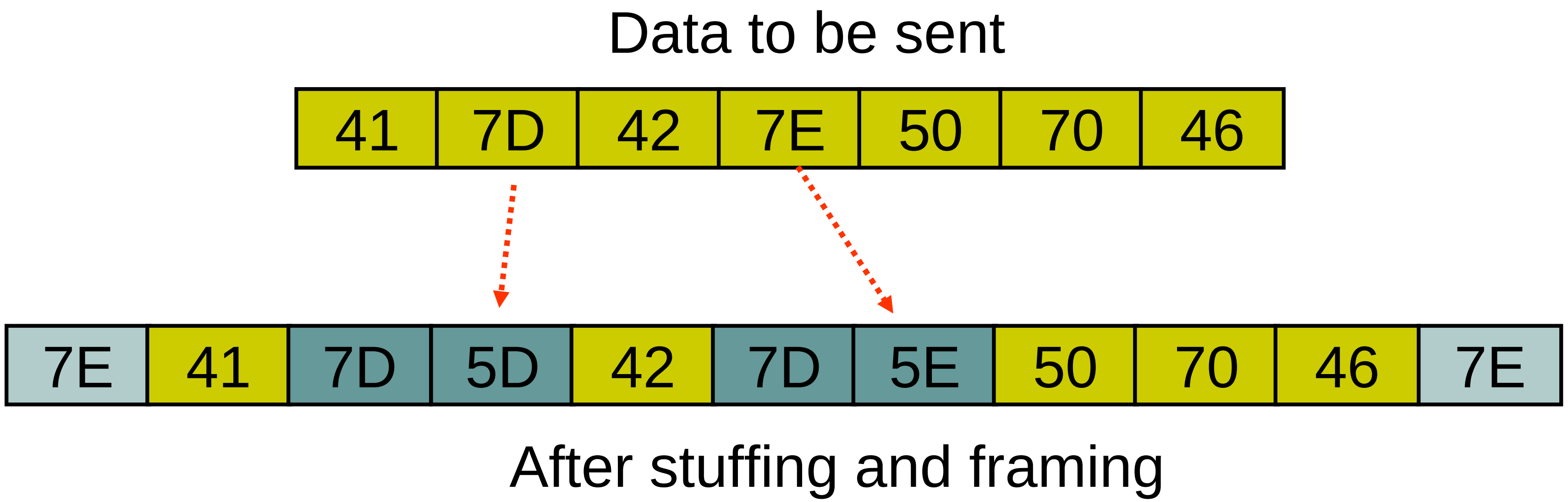
Unnumbered frame

Specifies what kind of packet is contained in the payload, e.g., LCP, NCP, IP, OSI CLNP, IPX

- PPP uses similar frame structure as HDLC, except
  - Protocol type field
  - Payload contains an integer number of bytes
- PPP uses the same flag, but uses byte stuffing

- PPP is character-oriented version of HDLC
- Flag is 0x7E (01111110)
- Control escape 0x7D (01111101)
- Any occurrence of flag or control escape inside of frame is replaced with 0x7D followed by

original octet XORed with 0x20 (00100000)



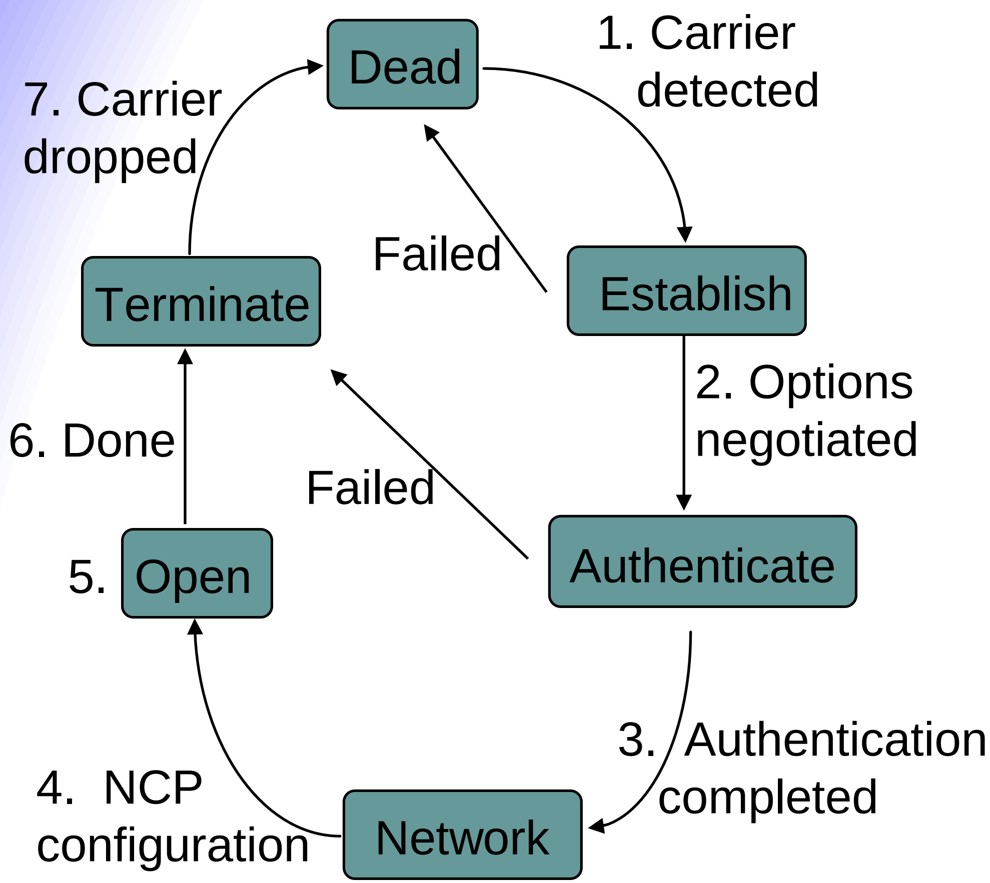
Data to be sent

41	7D	42	7E	50	70	46

7E	41	7D	5D	42	7D	5E	50	70	46	7E

After stuffing and framing

PPP phase diagram



Home PC to Internet Service Provider

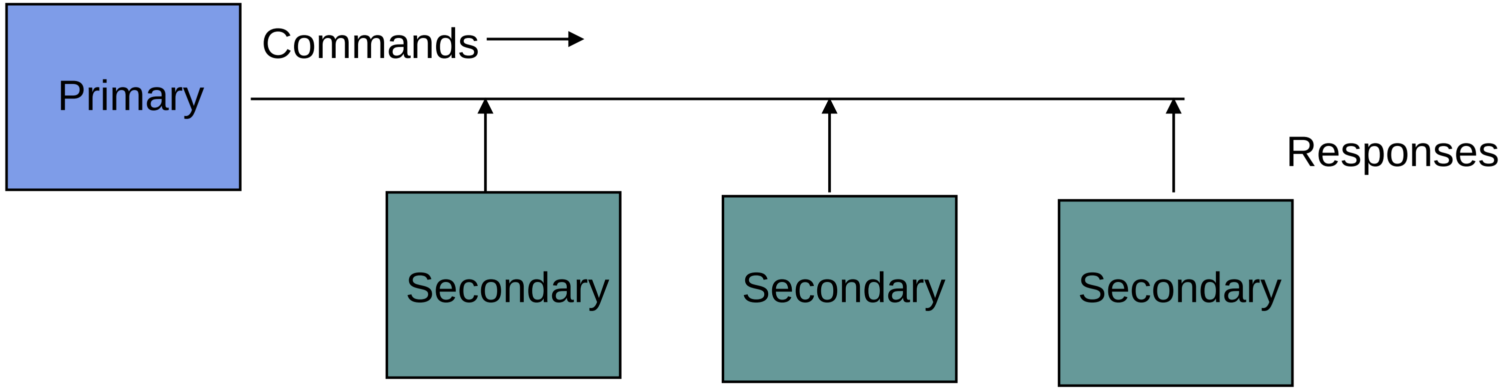
1. PC calls router via modem
2. PC and router exchange LCP packets to negotiate PPP parameters
3. Check on identities
4. NCP packets exchanged to configure the network layer, e.g. TCP/IP (requires IP address assignment)
5. Data transport, e.g. send/receive IP packets
6. NCP used to tear down the network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

- Password Authentication Protocol (PAP)
  - Initiator must send ID and password
  - Authenticator replies with authentication success/fail
  - After several attempts, LCP closes link
  - Transmitted unencrypted, susceptible to eavesdropping
- Challenge-Handshake Authentication Protocol (CHAP)
  - Initiator and authenticator share a secret key
  - Authenticator sends a challenge (random number and ID)
  - Initiator computes cryptographic checksum of random number and ID using the shared secret key
  - Authenticator also calculates cryptographic checksum and compares to response
  - Authenticator can reissue challenge during session

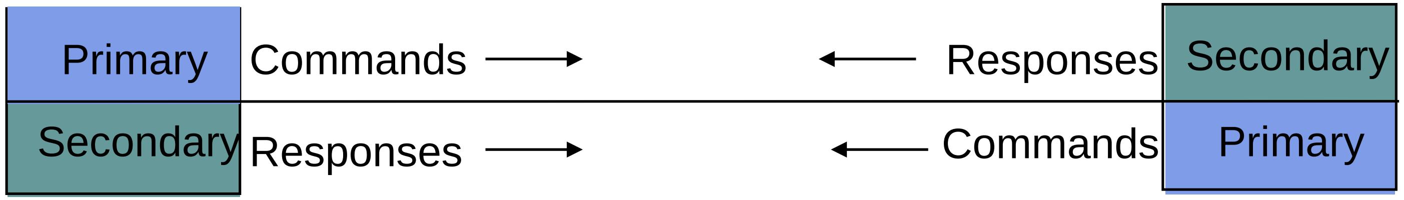
3.7 High-level data link control (HDLC)

- Bit-oriented data link control
- Derived from IBM Synchronous Data Link Control (SDLC)
- Related to Link Access Procedure Balanced (LAPB)
  - LAPD in ISDN
  - LAPM in cellular telephone signaling

- Normal Response Mode (NRM)
  - Used in polling multidrop lines



- Asynchronous Balanced Mode (ABM)
  - Used in full-duplex point-to-point links



- Mode is selected during connection establishment

Flag	Address	Control	Information	FCS	Flag

- Control field gives HDLC its functionality
- Codes in fields have specific meanings and uses
  - Flag: delineate frame boundaries
  - Address: identify *secondary* station (1 or more octets)
    - In ABM mode, a station can act as primary or secondary so address changes accordingly
  - Control: purpose and functions of frame (1 or 2 octets)
  - Information: contains user data; length not standardized, but implementations impose maximum
  - Frame Check Sequence (FCS): 16- or 32-bit CRC

control field format

Information Frame

1	2-4	5	6-8
0	N(S)	P/F	N(R)

Supervisory Frame

1	0	S	S	P/F	N(R)

Unnumbered Frame

1	1	M	M	P/F	M	M	M

- S: Supervisory Function Bits
- N(R): Receive Sequence Number
- N(S): Send Sequence Number

- M: Unnumbered Function Bits
- P/F: Poll/Final bit used in interaction between primary and secondary

information frame

- Each I-frame contains sequence number N(S)
- Positive ACK piggybacked
  - N(R)=Sequence number of *next* frame expected acknowledges all frames up to and including N(R)-1
- 3 or 7 bit sequence numbering
  - Maximum window sizes 7 or 127
- Poll/Final bit
  - NRM: Primary polls station by setting P=1; Secondary sets F=1 in *last* I-frame in response
  - Primaries and secondaries always interact via *paired* P/F bits

- Frames lost due to loss-of-sync or receiver buffer overflow
- Frames may undergo errors in transmission
- CRCs detect errors and such frames are treated as lost
- Recovery through ACKs, timeouts and retransmission
- Sequence numbering to identify out-of-sequence and duplicate frames
- HDLC provides for options that implement several ARQ methods

supervisory frame

Used for error control (ACK, NAK) and flow control (Don’t Send):

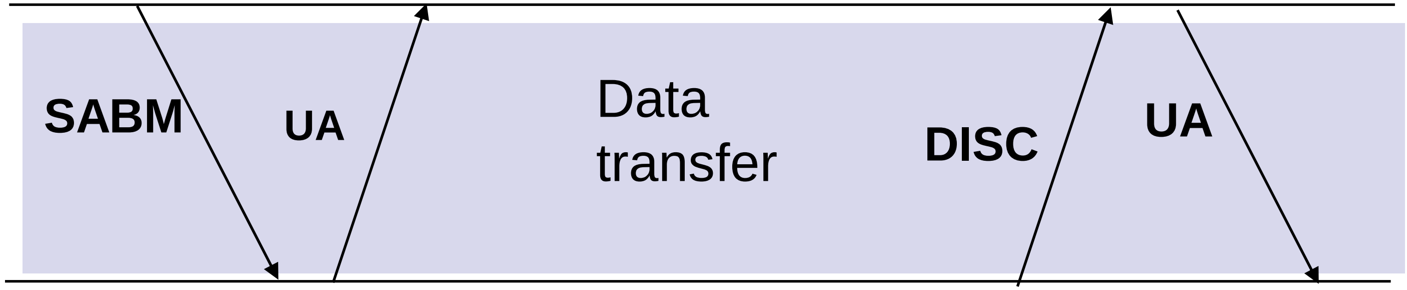
- Receive Ready (RR), SS=00
  - ACKs frames up to N(R)-1 when piggyback not available
- Reject (REJ), SS=01
  - Negative ACK indicating N(R) is first frame not received correctly. Transmitter must resend N(R) and later frames
- Receive Not Ready (RNR), SS=10
  - ACKs frame N(R)-1 and requests that no more I-frames be sent
- Selective Reject (SREJ), SS=11
  - Negative ACK for N(R) requesting that N(R) be selectively retransmitted

unnumbered frame

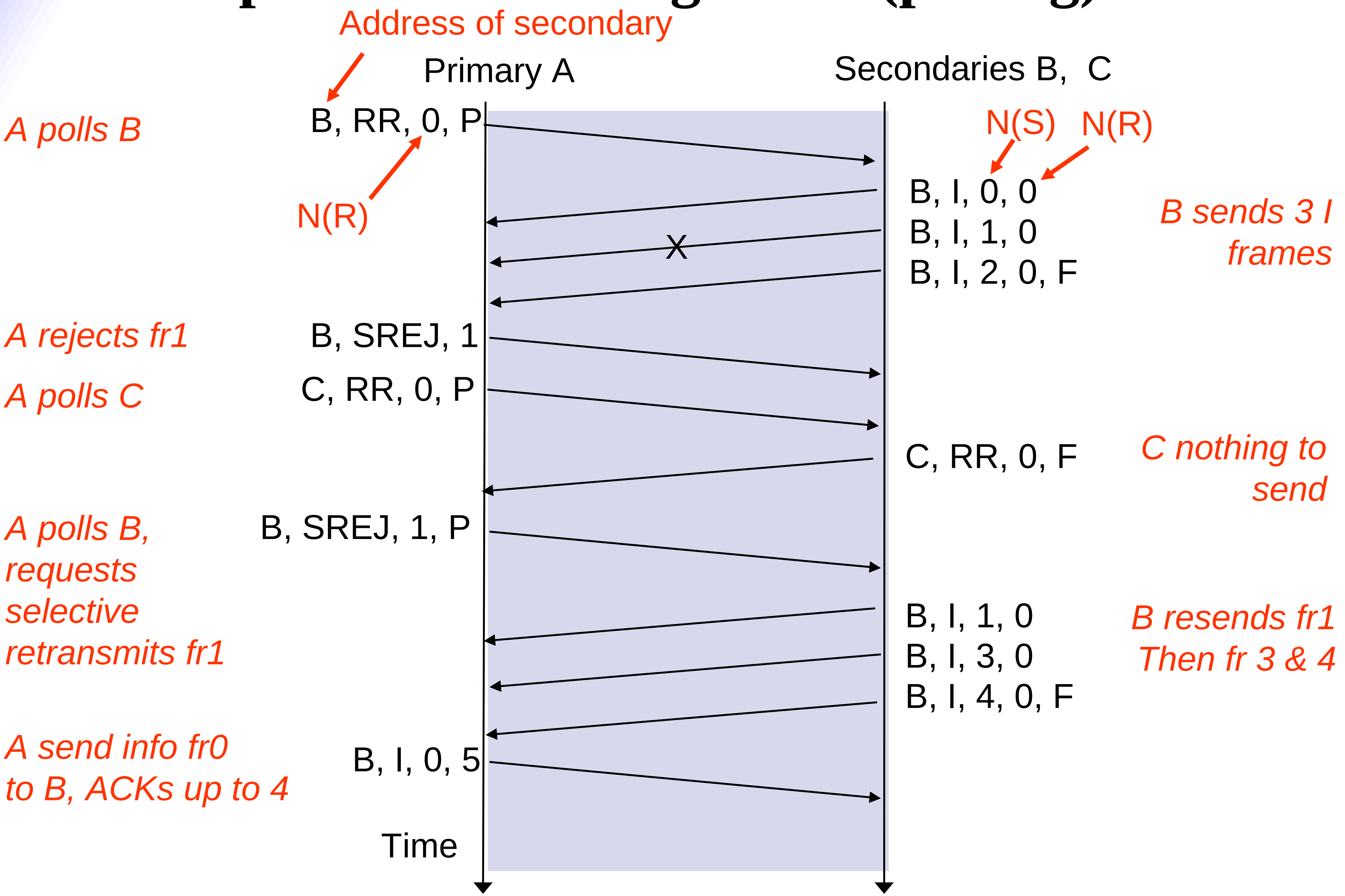
- Setting of modes with M bits:
  - set asynchronous balanced mode (SABM)
  - unnumbered acknowledgment (UA): acknowledges acceptance of mode setting commands
  - disconnect (DISC): terminates logical link connection
- Information transfer between stations
  - unnumbered information (UI)
- Recovery used when normal error/flow control fails
  - frame reject (FRMR): frame with correct FCS but impossible semantics
  - RSET: indicates sending station is resetting sequence numbers
- XID: exchange station id and characteristics

connection establishment and release

unnumbered frames used to establish and release data link connection



Example: HDLC using NRM (polling)



Address of secondary

Primary A

Secondaries B, C

A polls B

B, RR, 0, P

N(R)

N(S) N(R)

B, I, 0, 0

B, I, 1, 0

B, I, 2, 0, F

B sends 3 I frames

X

A rejects fr1

B, SREJ, 1

A polls C

C, RR, 0, P

C, RR, 0, F

C nothing to send

A polls B, requests selective retransmits fr1

B, SREJ, 1, P

B, I, 1, 0

B, I, 3, 0

B, I, 4, 0, F

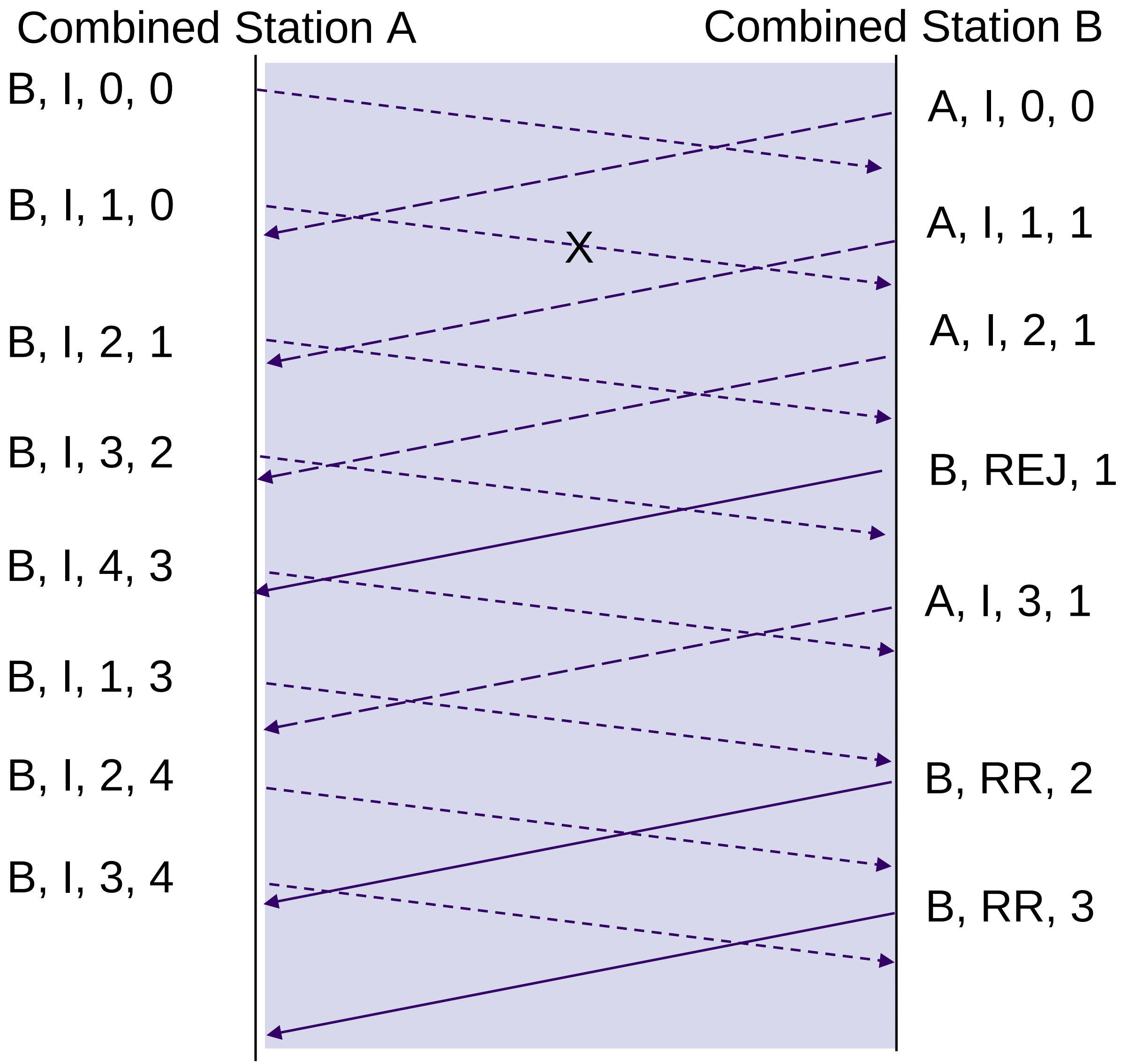
B resends fr1 Then fr 3 & 4

A send info fr0 to B, ACKs up to 4

B, I, 0, 5

Time

Example: frame exchange using ABM



Combined Station A

Combined Station B

B, I, 0, 0

A, I, 0, 0

B, I, 1, 0

A, I, 1, 1

B ACKs fr0

A sends 5 frames

B, I, 2, 1

A, I, 2, 1

B, I, 3, 2

B, REJ, 1

B rejects fr1

B, I, 4, 3

A, I, 3, 1

A goes back to 1

B, I, 1, 3

B, I, 2, 4

B, RR, 2

B ACKs fr1

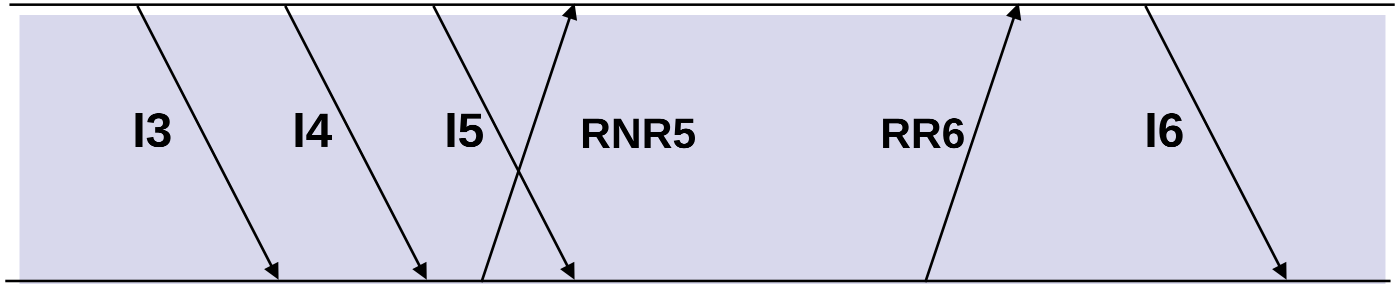
B, I, 3, 4

B, RR, 3

B ACKs fr2

flow control

- Flow control is required to prevent transmitter from overrunning receiver buffers
- Receiver can control flow by delaying acknowledgement messages
- Receiver can also use supervisory frames to explicitly control transmitter
  - Receive Not Ready (RNR) and Receive Ready (RR)



Chapter Summary

- peer-to-peer protocols
- reliable data transfer – error detection + ARQ
- parity check, Internet checksum, polynomial codes
- SW, GBN, SR
- flow control
- framing
- PPP
- HDLC

Reference

Chapters 3 and 5, Communication Networks: Fundamental Concepts and Key Architectures