

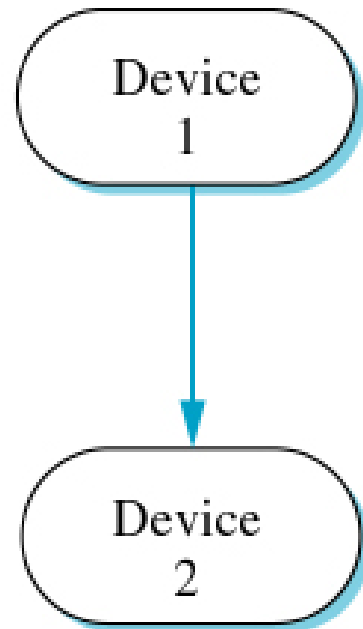
**Lectures 14, 15, 16:**  
**Connections, Protocols,**  
**Link and Flow Control, LANs**

*Brian Carpenter*

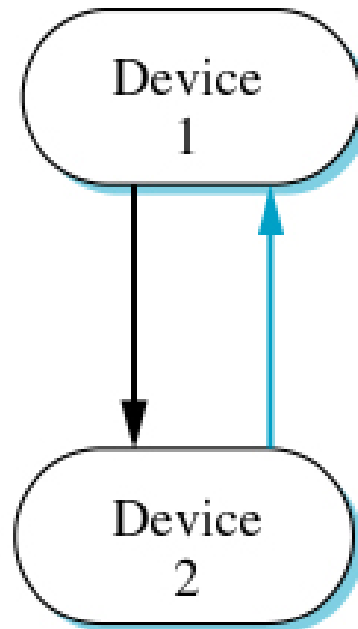
314 S1T 2008

# Transmission Modes (Shay 4.3)

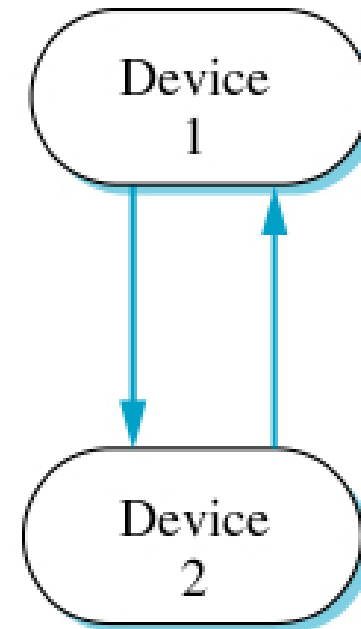
- Parallel (many wires) or Serial (one wire)
- Direction-related



Simplex  
communication  
goes one way only.



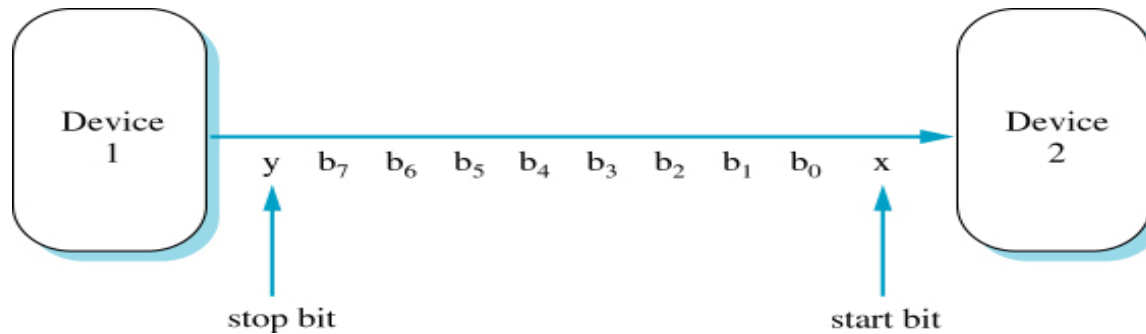
Half-duplex  
communication can go  
both ways, but devices  
must alternate sending.



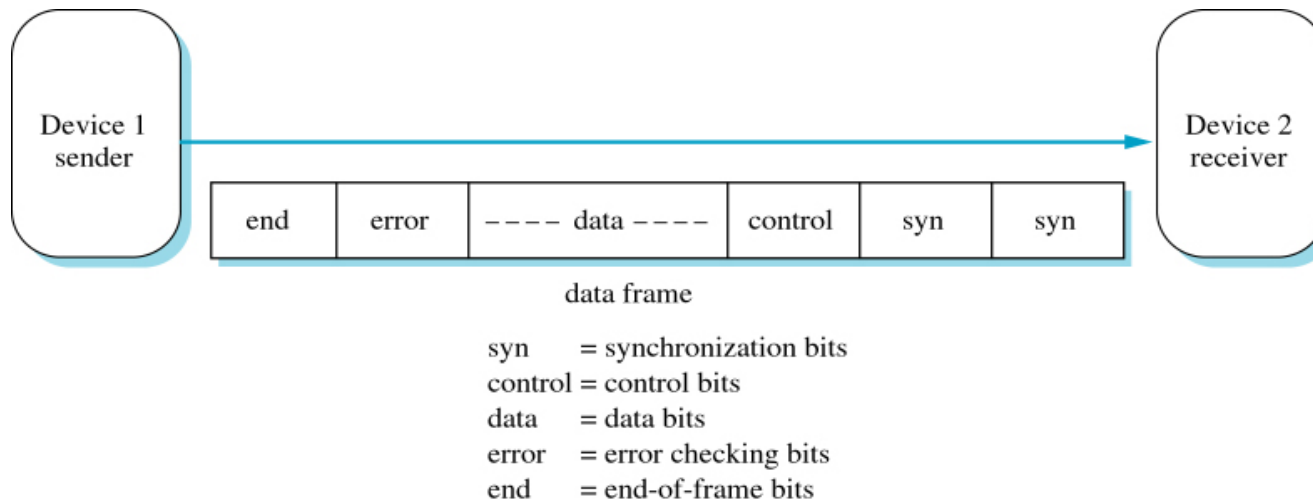
Full-duplex  
communication can  
go both ways  
simultaneously.

# Transmission Modes

- Time-related
  - **asynchronous**: may start/stop at any time



- **synchronous**: uses a continuous clock



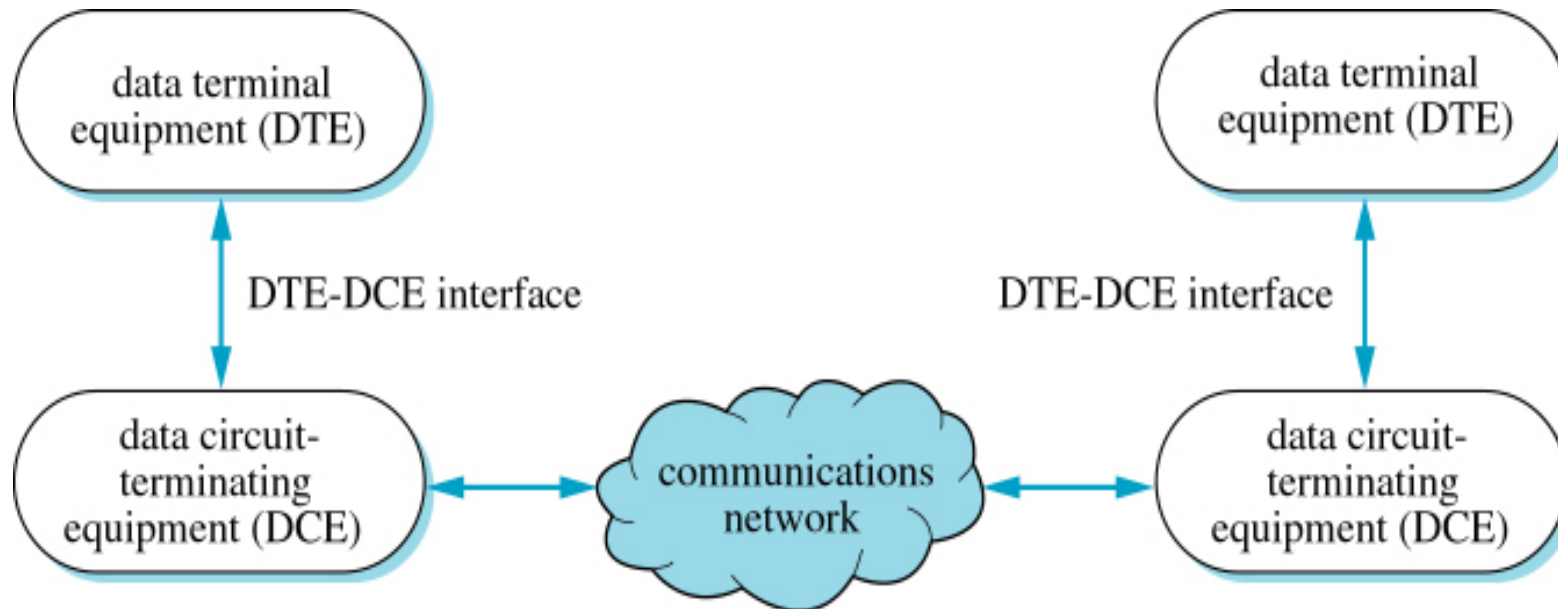
- **isochronous**: imposes gaps to match transmission rates

# Interface Standards (Shay 4.4)

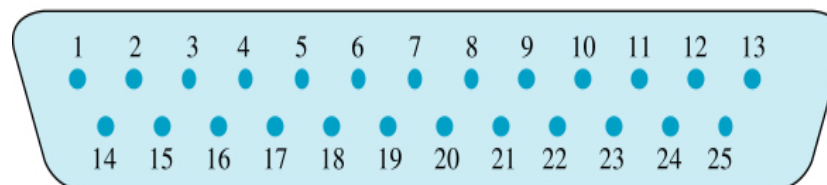
- There are lots of 'standard' interfaces for connecting devices together
- Shay has good descriptions of:
  - EIA-232 (RS-232) <= we only look at this one
  - USB
  - IEEE 1394 (Firewire)
  - X.21

# RS-232 Serial Interface

- Connects DTE (computer) to DCE (modem)

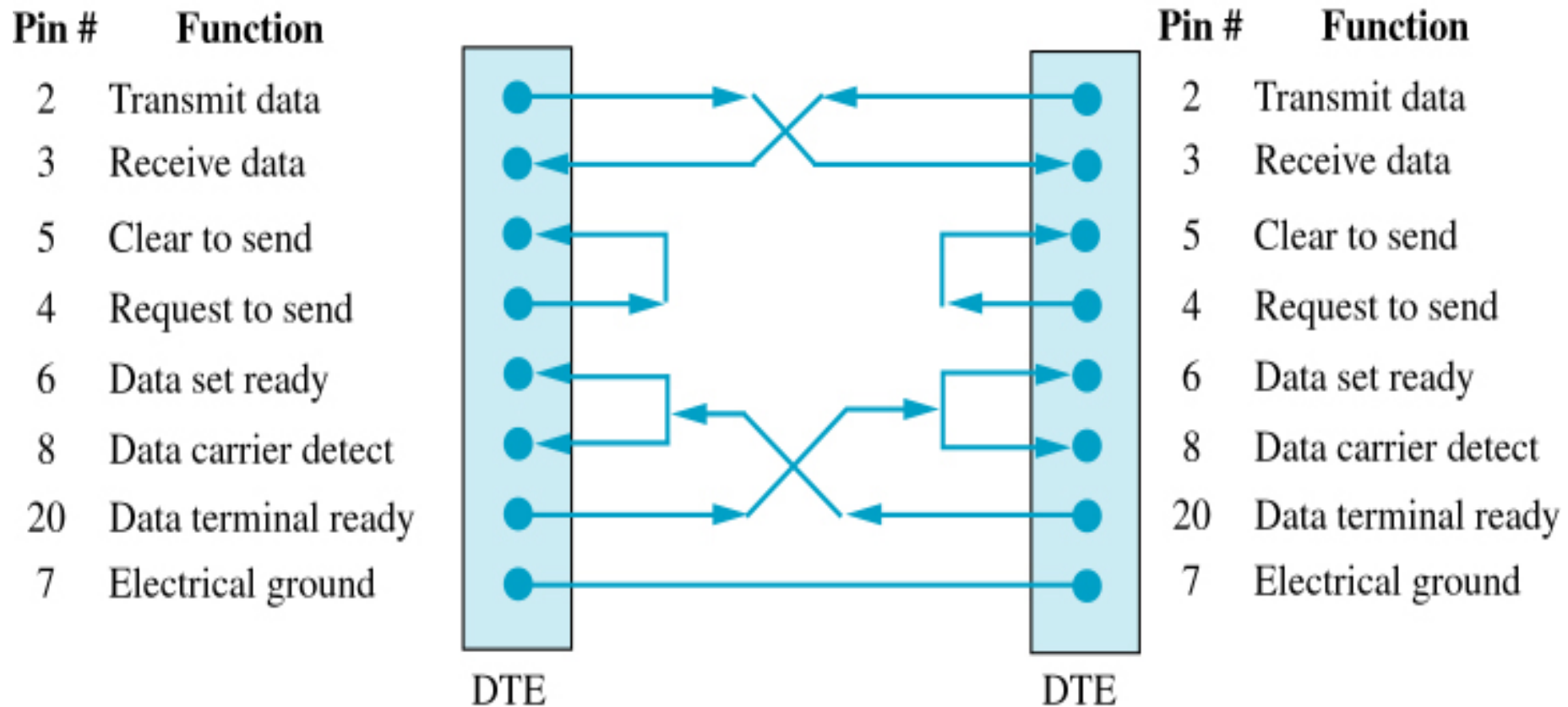


- 25-pin connector, we normally use only 9



# RS-232 Serial Interface

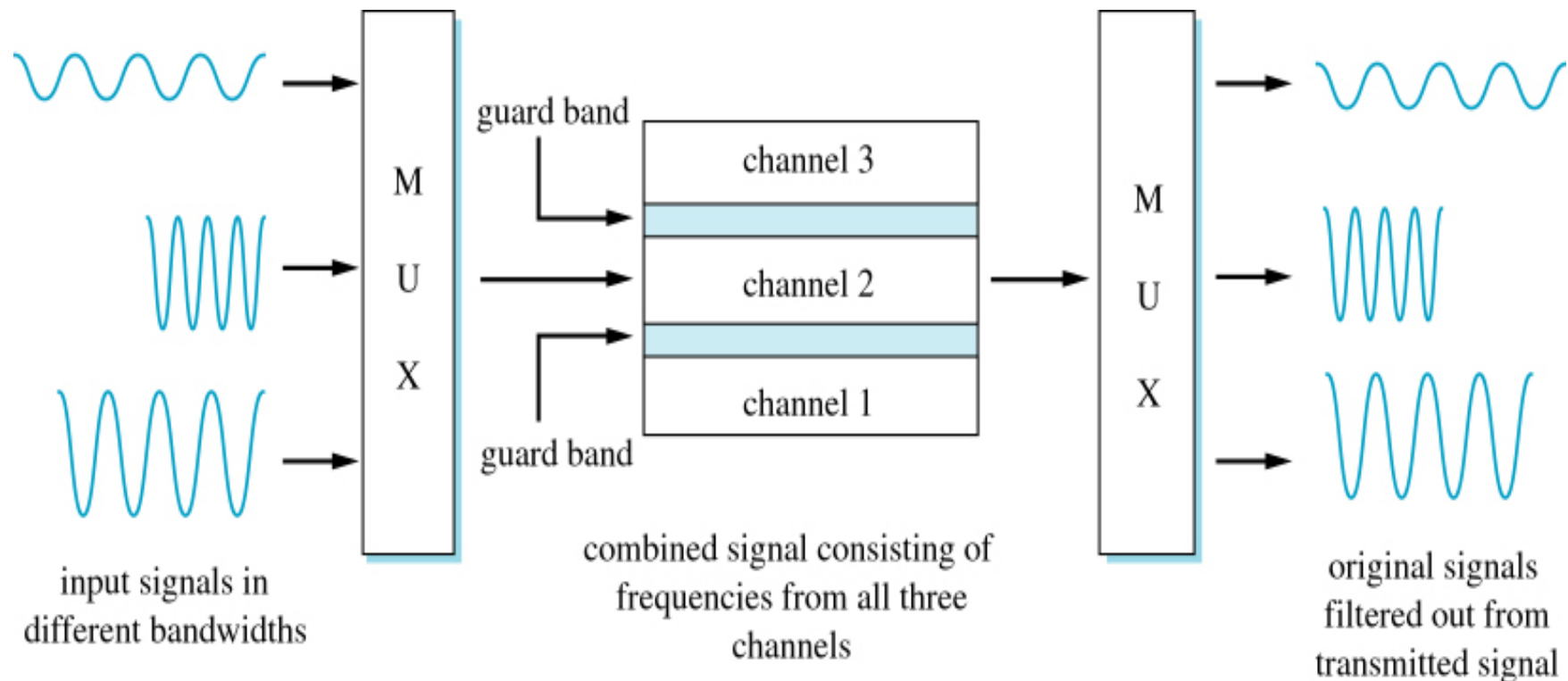
- *Null Modem* for connecting two DTEs



- *Not used here:* pin 22 = Ring Indicator, pin 1 = Protective Earth

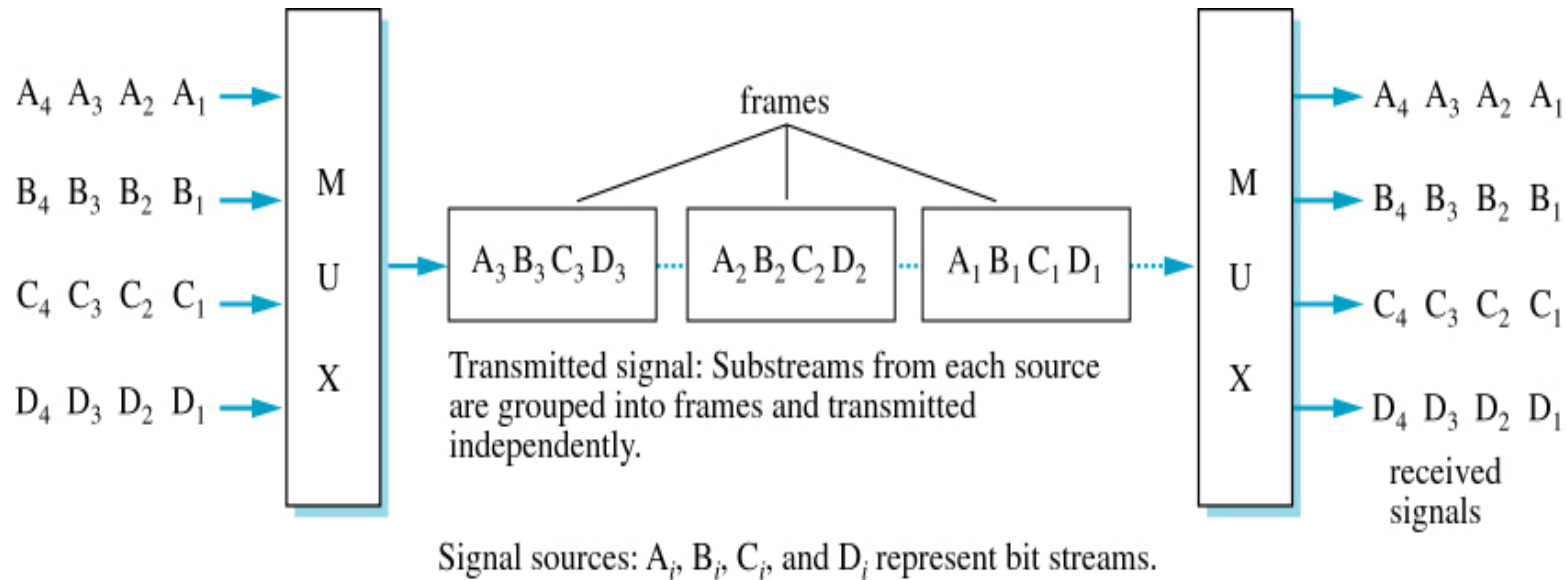
# Multiplexing (Shay 4.5)

- Ways of carrying several different connections over a common link
- Frequency-Division (FDM):



# Multiplexing (2)

- Time-Division (TDM):

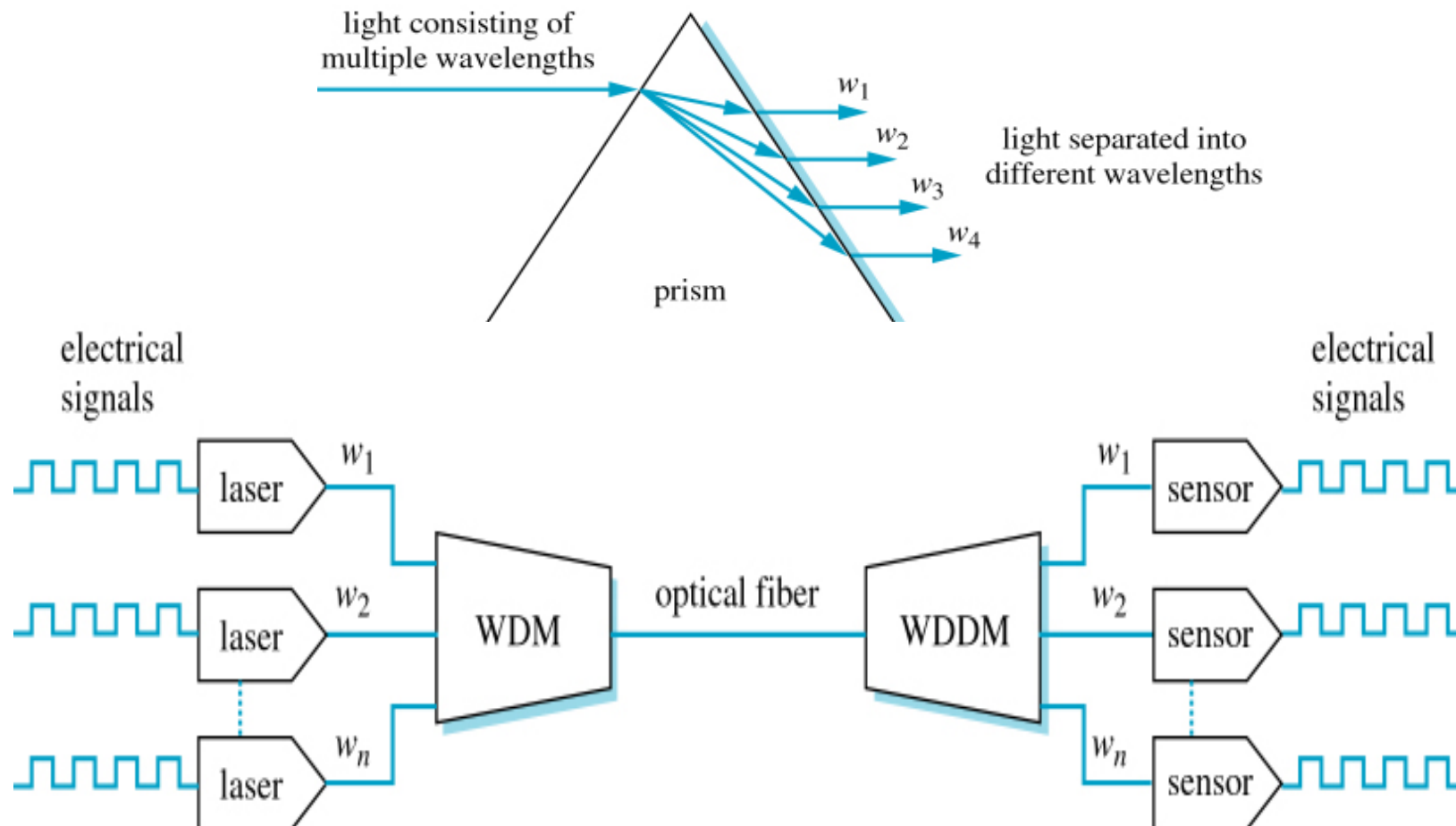


- Statistical Multiplexing
  - Much the same as TDM, but doesn't use fixed time allocations (slots)
  - Receiver must be able to identify incoming frames



# Multiplexing (3)

- Wave-Division (WDM):



# Flow Control (Shay 8.1)

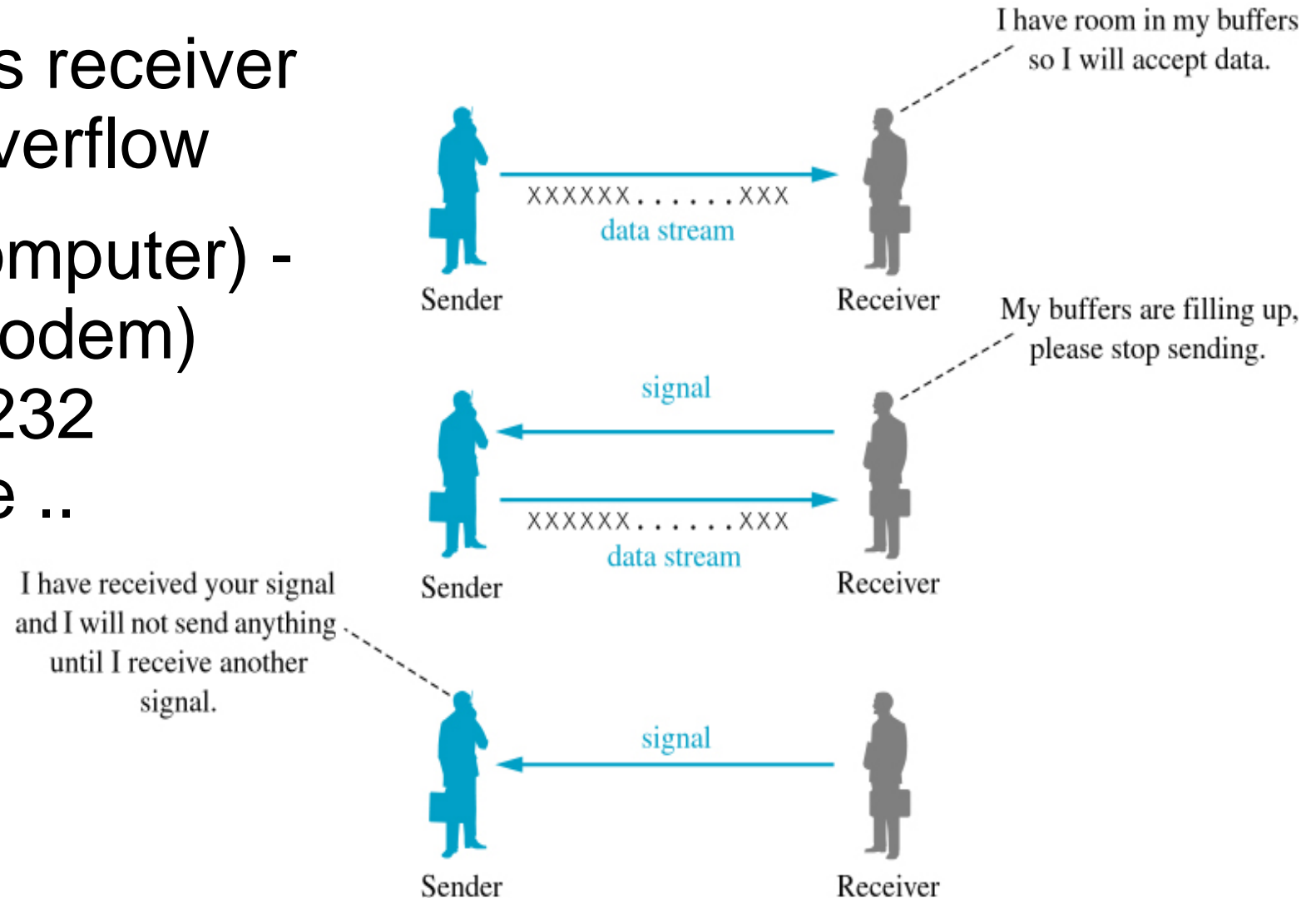
- Need for flow control
  - how can we send long messages, e.g. big files?
  - what happens when messages get lost, or are corrupted when they arrive?
  - what if the receiving *host* is busy, i.e. slow to accept incoming data?
  - how will a sender cope with lost (undelivered) messages?
  - will both hosts be able to send/receive at the same time?

# What *is* Flow Control?

- Messages are broken into *frames*
- Flow Control defines
  - “the way frames are sent, tracked and controlled”
  - may be simple or complex
- Many examples of protocols around us, e.g. traffic rules (Road Code), 'phone conversations
- How can we be sure that a protocol is *correct*?
  - works properly
  - will never suddenly 'freeze'

# Signaling (Shay 8.2)

- *Receiver tells sender when it's ready to receive*
- Prevents receiver buffer overflow
- DTE (computer) - DCE (modem) via RS-232 interface ..



## X-ON/X-OFF

- Over the DTE-DCE path ..
  - send ASCII X-OFF (0x13, ^S) to stop transmission
  - send X-ON (0x11, ^Q) to start it again
- This is *in-band* signalling, i.e. send signal on same path as data
- How quickly does the transmitter stop sending?
- How can we send 0x11 or 0x13 to the receiver?

## Frame-oriented Control (Shay 8.3)

- Idea is to break large sequences of chars into smaller *frames*
- Frames are sent from one *user* (higher protocol layer) to another
- *Unrestricted* protocol
  - simply assume it's always safe to send
  - not really a useable protocol!

## *Stop-and-Wait*

- Sender:
  - send frame, wait for ACK or NAK
  - if NAK, send frame again. Repeat until get ACK
- Receiver:
  - receive frame, check for errors
  - if OK, send ACK. otherwise send NAK
- No way to handle lost (therefore not ACKed) frames

# Protocol Efficiency: Effective data rate

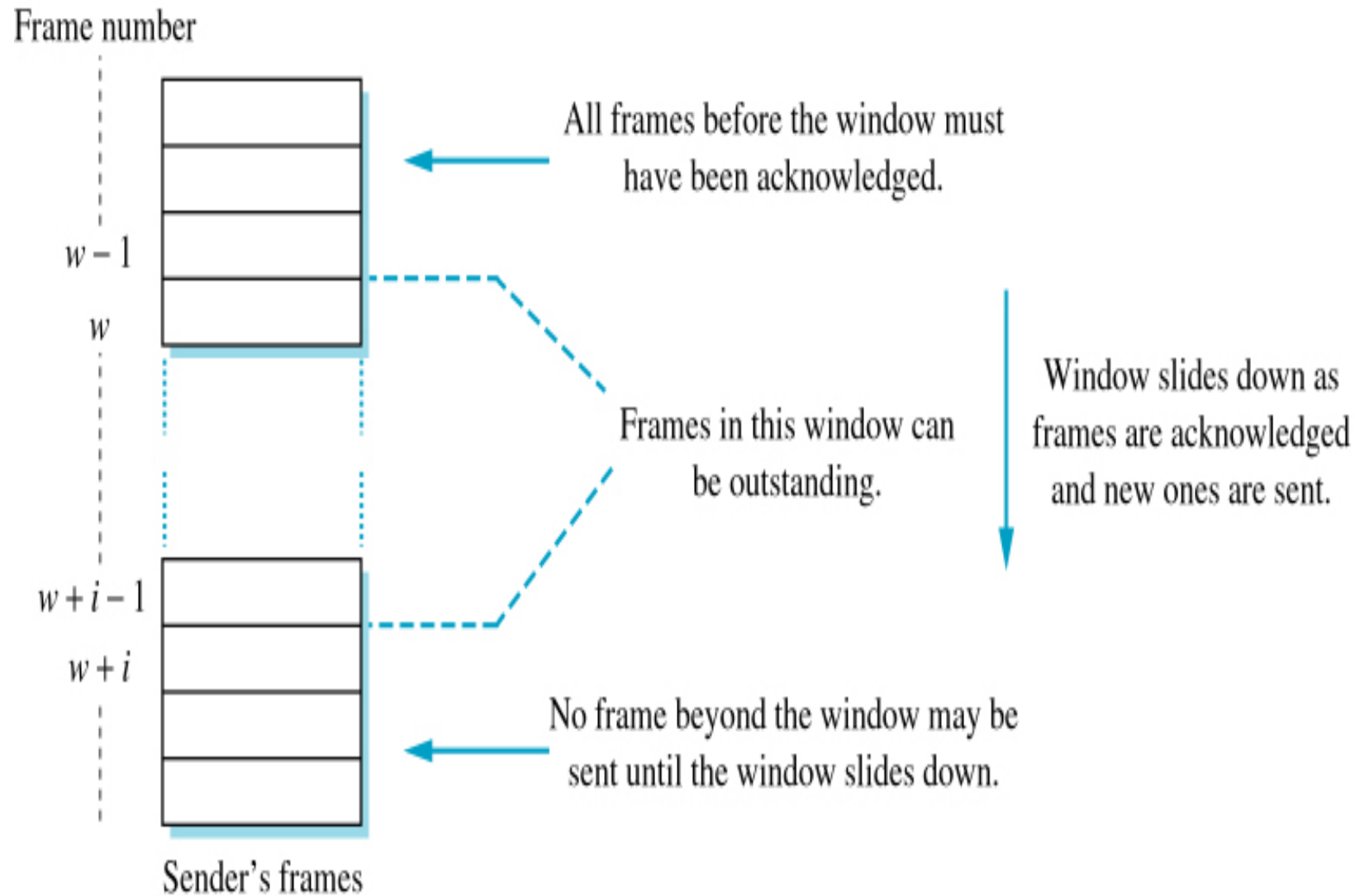
- Shay derives formulae, we “just work it out”
- Remember, *velocity = distance / time*
  - in wire or fibre,  $v$  is  $\sim 2/3$  speed of light, i.e.  $2 \times 10^8$  m/s
  - Auckland-Hamilton is about 120 km, so a byte takes  $(120 \times 10^3) / (2 \times 10^8) = 0.6$  ms to get there
  - If we send a 1500-Byte frame at 10 Mb/s, it will take  $(1500 \times 8) / (10 \times 10^6) = 1.2$  ms to transmit
  - Assume that ACK is a 64-Byte frame, 0.0512 ms
  - Therefore, to send frame and receive ACK takes roughly  $1.2 + 0.05 + 2 \times 0.6 = 2.45$  ms
  - Effective bit rate is  $(1500 \times 8) / (2.45 \times 10^{-3}) = \underline{4.9 \text{ Mb/s}}$
  - *Half the time is wasted waiting for ACKs*



## Side note: a catch in the notation

- Convention:
  - Mb/s for megabits per second
  - MB/s for megabytes per second
- Often leads to confusion, especially with marketing people, journalists, and politicians.
- If there is any chance of confusion, write "megabits" or "megabytes" in full.
- In data communications, we normally discuss megabits. But when considering application throughput, megabytes are more useful.

# Sliding Window (Shay 8.4)

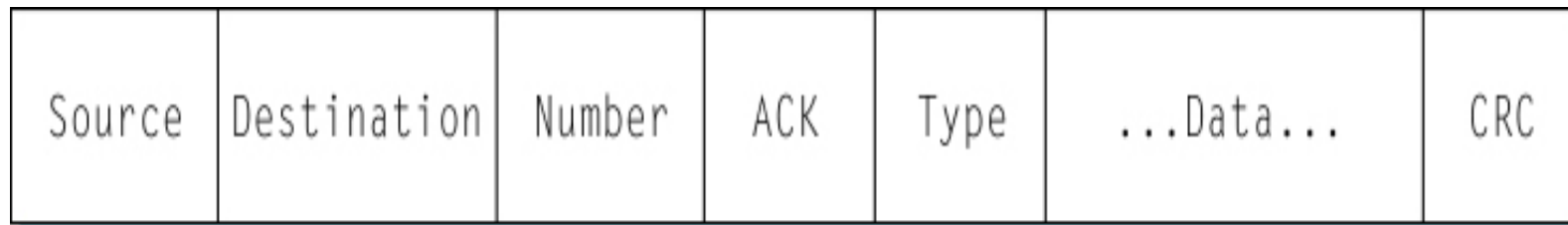


## Sliding Window / *Go-back-n*

- Idea here is to have a maximum of  $i$  frames on the wire at any time.  $i$  is the *window size*
- Each frame has a sequence number, sender must hold each frame until it is ACKed
- Sender keeps track of  $w$ , sequence number of first (of  $i$  frames) in window. When frame  $w$  is ACKed, sender can forget it
- Window does not move until earliest frame has been ACKed

## Go-back-n

- Shay develops a frame format for two-way communication

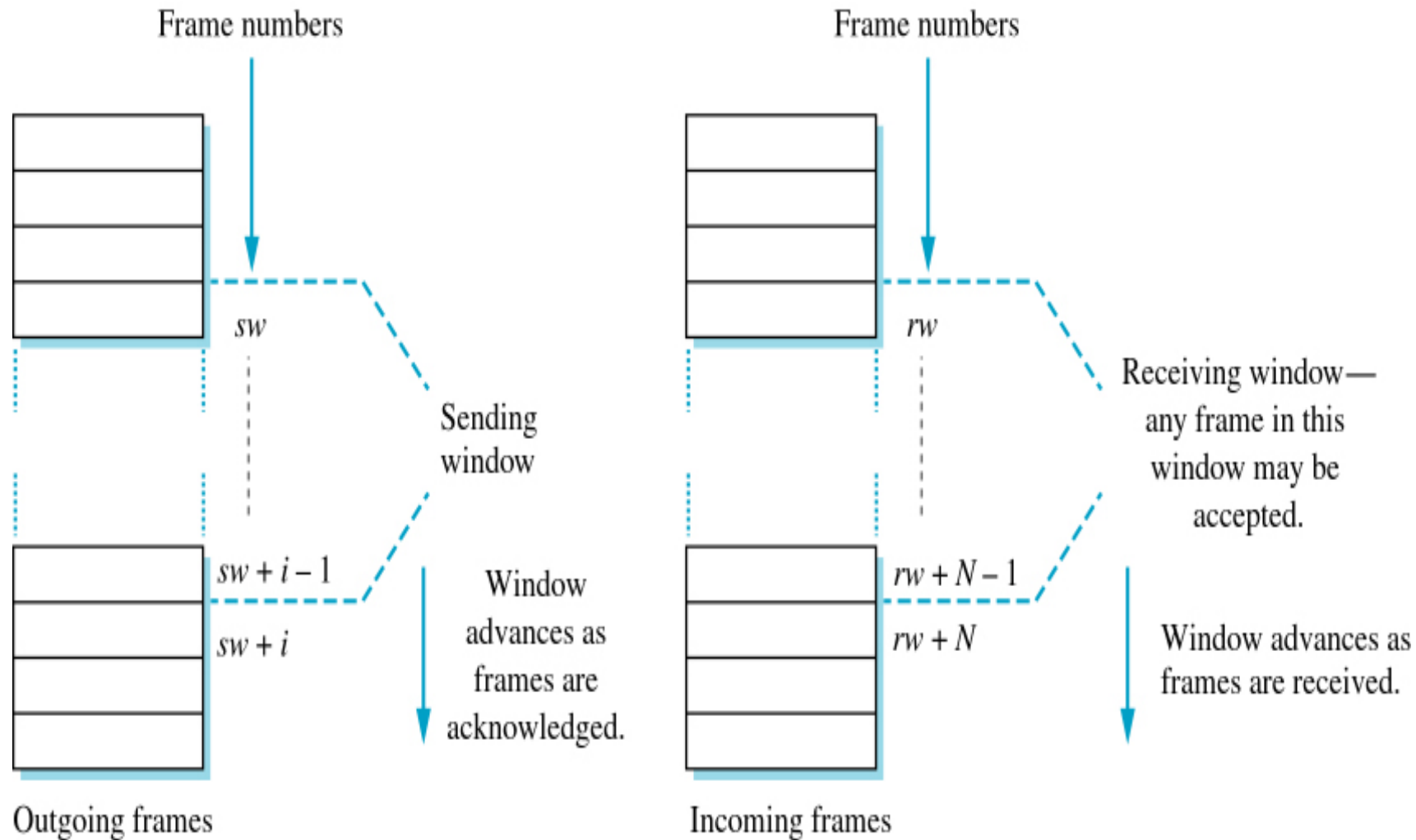


- Data frame in one direction can carry an ACK for the other direction, i.e. a *piggy-backed ACK*
- To handle lost frames, he has an *ACK timer* at the receiver ..
- and a *frame timer* at the transmitter

# Sequence Numbers

- Sequence Numbers fit in a  $K$ -bit field; there can be at most  $2^K$  frames in the window
- $K$  should be big enough to handle the maximum window size we expect to use
- They are *unsigned* numbers, and can *wrap*, i.e. count through  $2^K-2, 2^K-1, 0, 1, 2, \dots$   
You can think of the sequence numbers as being arranged in a circle
- What happens if a host crashes and restarts?
- Some protocols used *lollipop sequence numbering* to handle restarts! (see Wikipedia)

# Selective Repeat (Shay 8.5)



## Selective Repeat (2)

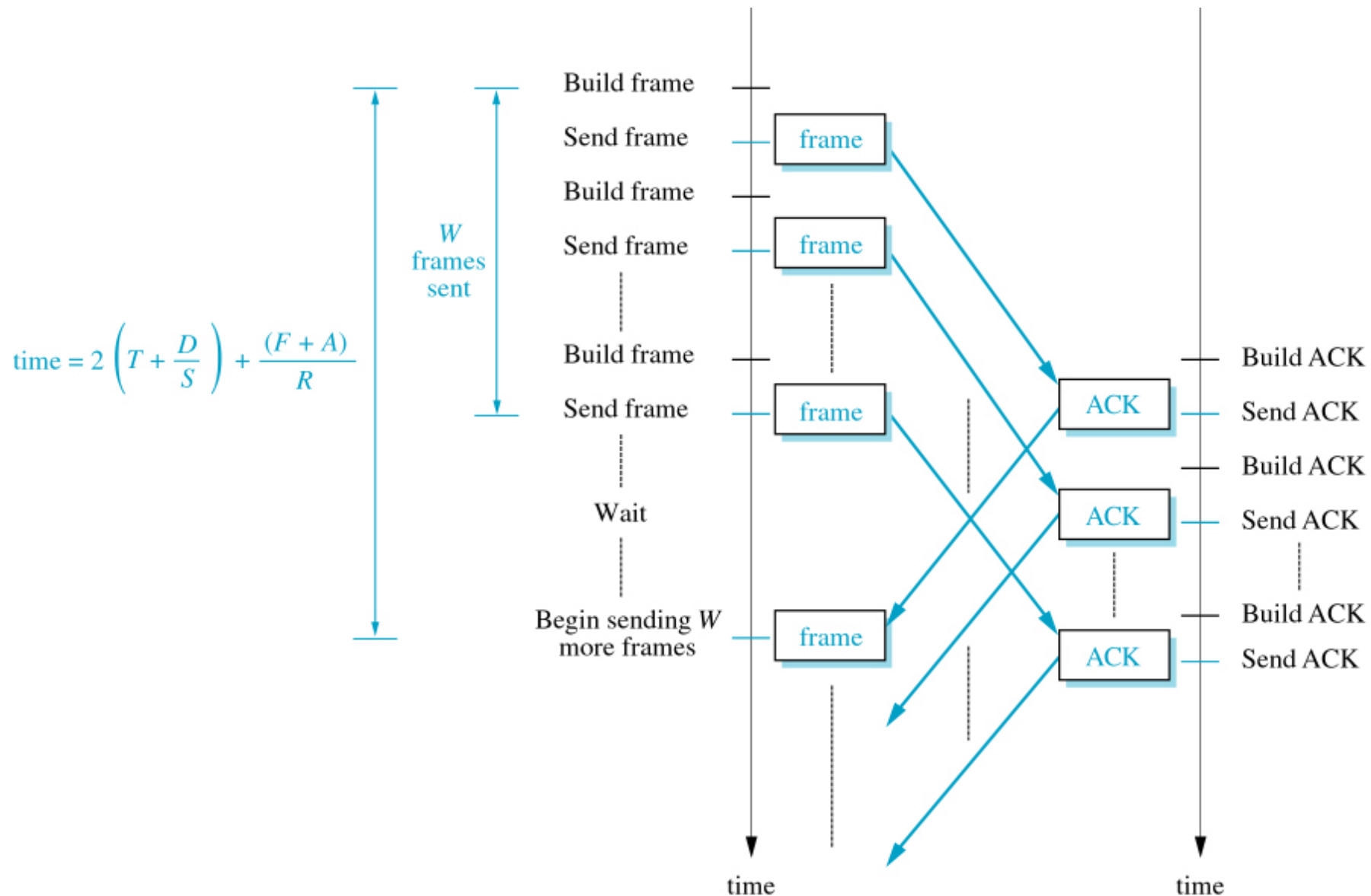
- Any frame can be ACKed, specifying its sequence number
- Frames arriving out of sequence are *buffered* until earlier frames have been ACKed
- When a NAK is received, only the NAKed frame is resent (Go-Back-n resent the whole window!)
- If a frame timer expires (no ACK or NAK), only the timed-out frame is resent
- Piggy-backed ACK acknowledges the *last frame delivered to the user*, so the sender knows that all frames up to that one have been safely received

## Efficiency of Sliding Window Protocols (8.6)

- For a particular window size, message size, transmission speed and link distance, we can “just work it out,” as we did for stop-and-wait
- *We assume no lost or damaged packets !*
- Two cases
  - we get our first message ACKed before we've sent a whole window. That allows us to keep sending at full link speed
  - we have to wait for an ACK after sending a window, then we can send another window. Shay has a diagram illustrating this ..



# Sending whole window and waiting



# Numerical examples

- Sending 100x 1500B frames in 20-frame windows, Auckland-Hamilton on a 10 Mb/s link
  - as for Stop-and-Wait: 1.2ms to send frame, 1.2ms round-trip time.  
Any window > 2 frames can run at full speed, 10 Mb/s
- As above, but with 64B frames
  - send time is  $(64 \times 8)/(10 \times 10^6) = 0.0512 \text{ ms}$
  - time to send 20 frames =  $20 \times 0.0512 = 1.024 \text{ ms}$
  - first ACK returns after  $1.2 + 2 \times 0.0512 = 1.3024 \text{ ms}$
  - effective bit rate is  $(20 \times 64 \times 8)/1.3024 = 7.862 \text{ Mb/s}$
  - note the effect of using a *small frame size* !

# Bandwidth-Delay Product (BDP)

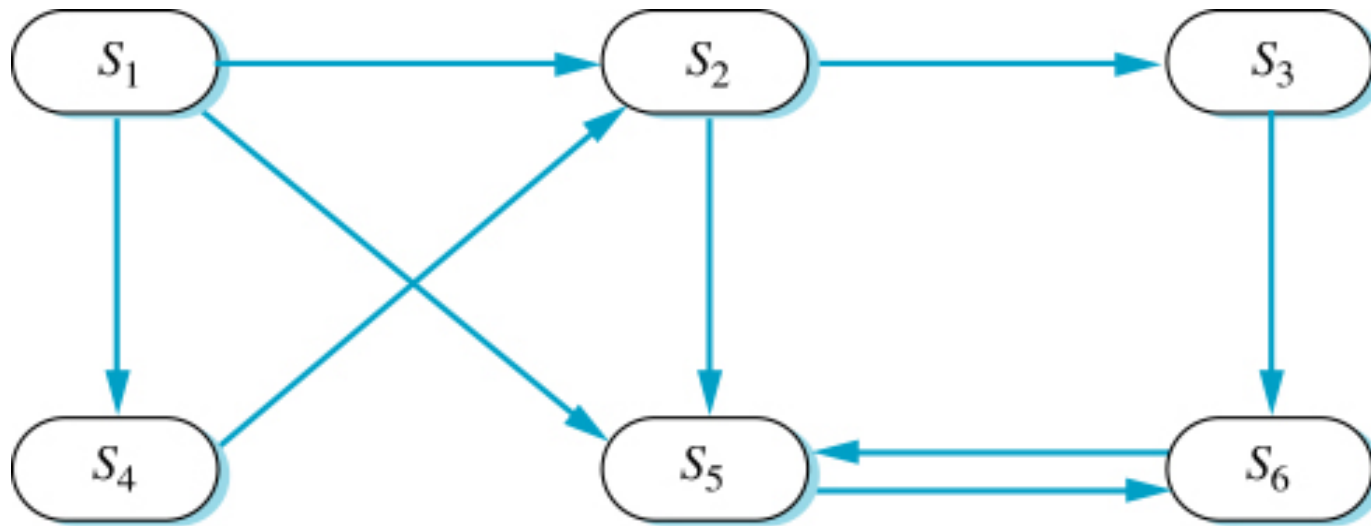
- BDP for a link = data rate x link delay\*
- Auckland-Hamilton at 10 Mb/s:  
$$\text{BDP} = 10 \text{ Mb/s} \times 0.6 \text{ ms} = \mathbf{6000 \text{ bits}}$$
$$= \mathbf{750 \text{ B}}$$
- This is the maximum number of bits we can have 'on the wire'
- Need to have buffers **at least double this** so that transport protocol can keep the link busy
  - **fill the wire once, and then again before first ACK returns**
- Bigger frames sizes help to keep the link busy – less *protocol overhead*

*\*one-way delay, not round-trip time*

# Protocol Correctness (Shay 8.7)

- Shay discusses two ways to describe systems:
  - Finite State Machines
  - Petri nets
- Finite State Machine models a system as being in one of a finite set of *states*
- State Transition Diagrams (STDs) are graphs, each vertex represents a state, and each edge a transition between states
- Petri nets are more detailed, we won't discuss them further

# State Transition Diagrams

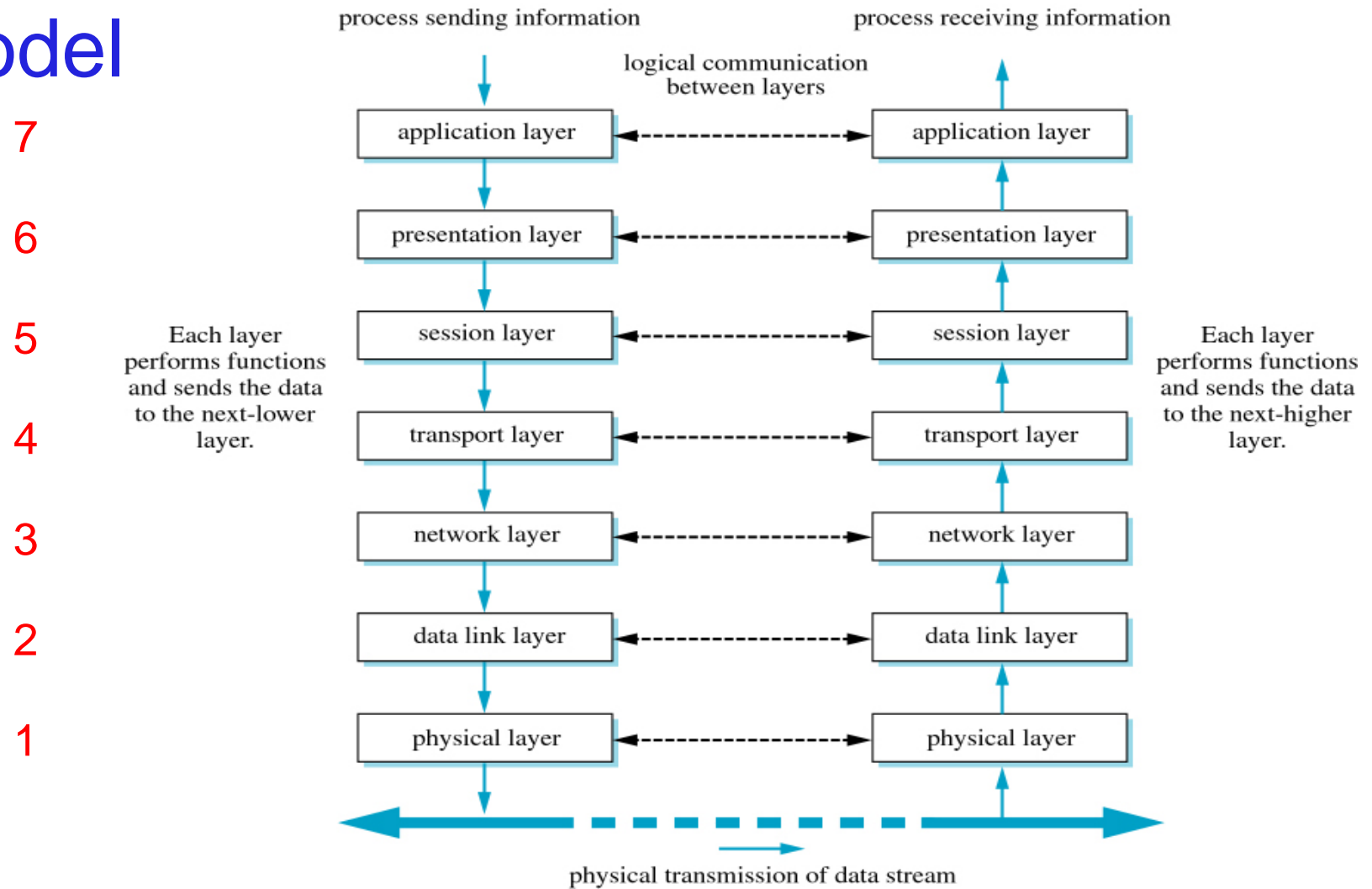


- Look for problems on graph
  - No edges pointing to  $S_1$
  - $S_5 - S_6$  is an infinite loop
- This kind of analysis helps find flaws
  - *it doesn't prove correctness!*

# Protocol Layers, the OSI Model (reminder)

- Layers are an abstraction, they provide a simple view of what happens in a communication system
- Layer  $n$ 
  - provides services to layer  $n+1$
  - uses services from layer  $n-1$
- Generally we implement systems this way, but sometimes we may find it useful to peek between layers, or 'break layer purity'

# OSI Model



- OSI has 7 layers, TCP/IP collapses 5-7 into 5
- So far, we've mainly discussed layer 1

# Introduction to LANs (Shay 9.1)

- LANs connect many hosts (devices) together
- Link medium may be copper (coax or UTP), fibre or wireless
- Topology may be
  - *bus*: hosts share the medium by taking turns
  - *ring*: access is controlled by passing a token
- Ethernet – today's most common LAN physical layer – uses a bus topology
- Point-to-point link is a LAN with only two hosts

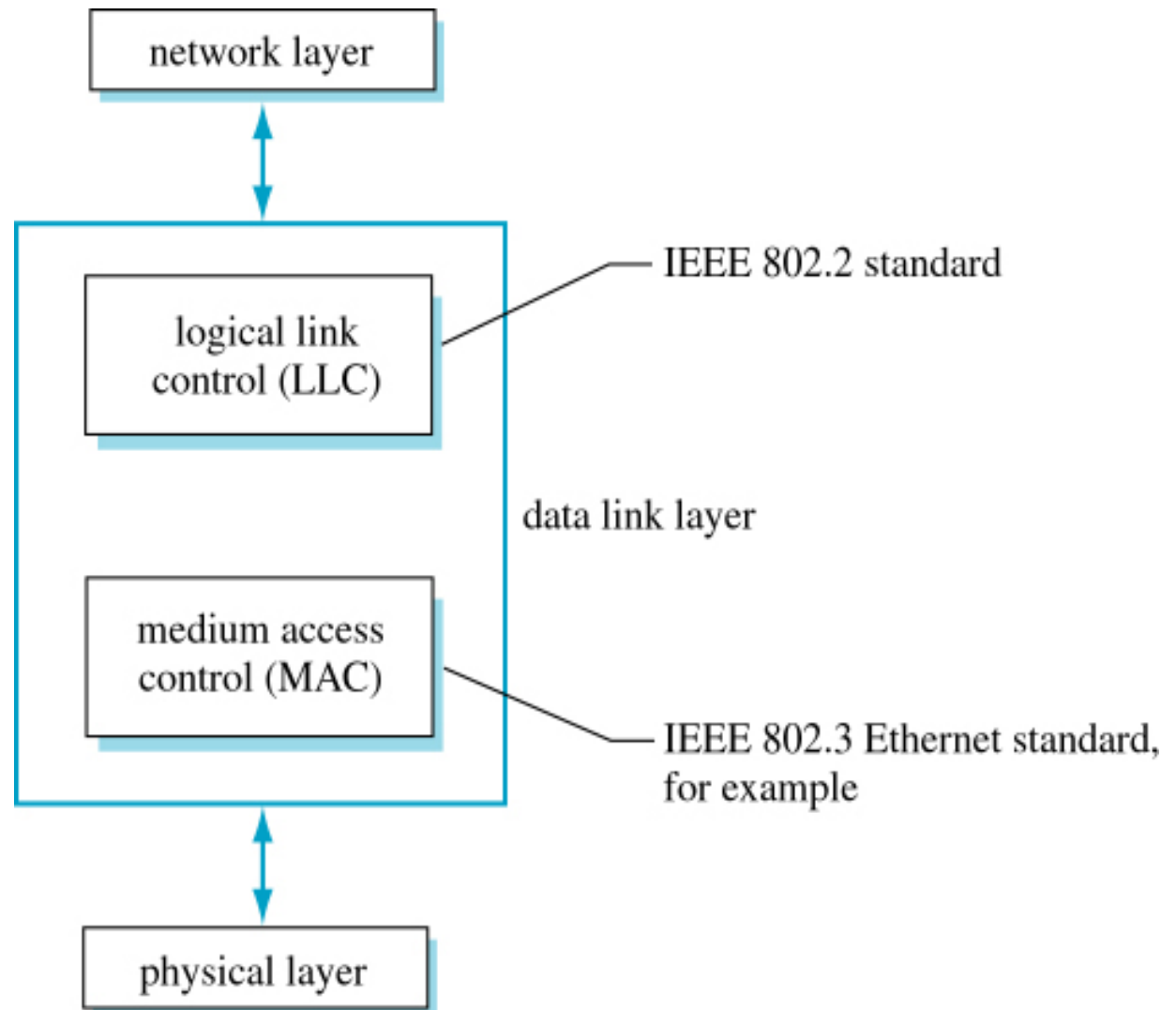


# LAN Layers

- Layer **1** is the **Physical** layer. On this layer, you've already looked at signaling and modulation methods
- Layer **2**, the **Link** layer, is where hosts talk to each other. Protocols here send frames (packets) to other hosts, and receive frames in response
- Layer **3**, the **Network** layer, is used to pass packets between LANs. For example, we often use IP to pass frames between Ethernet-connected hosts

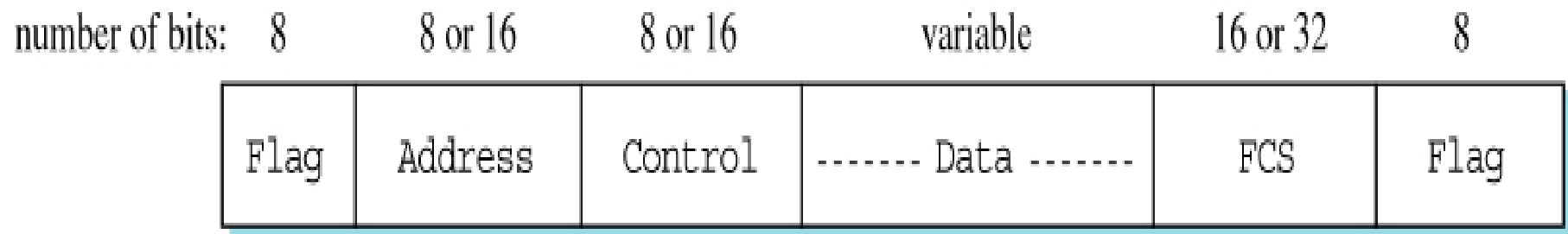
# Data Link Control (Shay 9.2)

- Link layer is divided in two – LLC and MAC
- Shay presents HDLC, a fore-runner of IEEE 802.2
- These are bit-oriented protocols



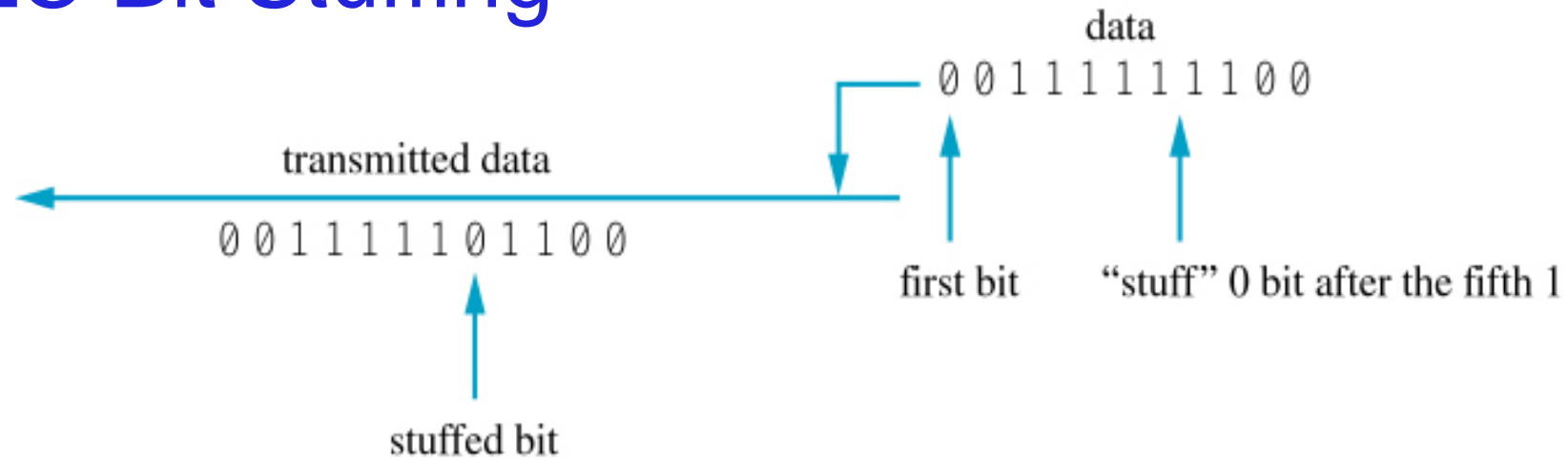
# HDLC Frame Format

- *Flag* pattern, 01111110 (six 1s) marks start and end of frame. Receiver watches medium for flags

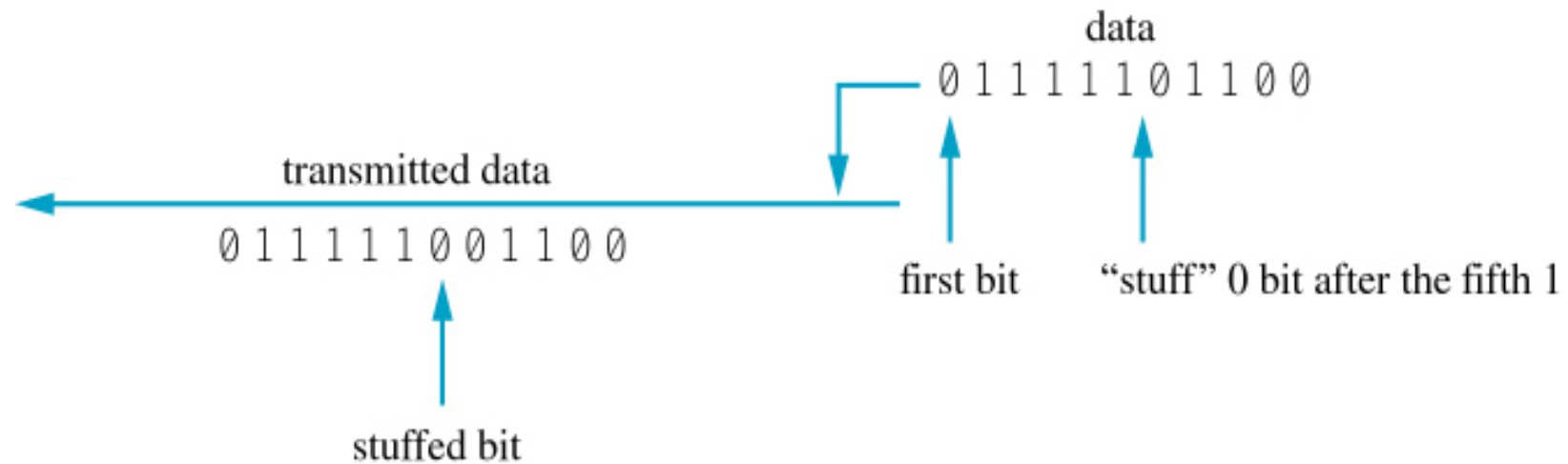


- How do we send the flag pattern within the data part of the frame?

# HDLC Bit Stuffing

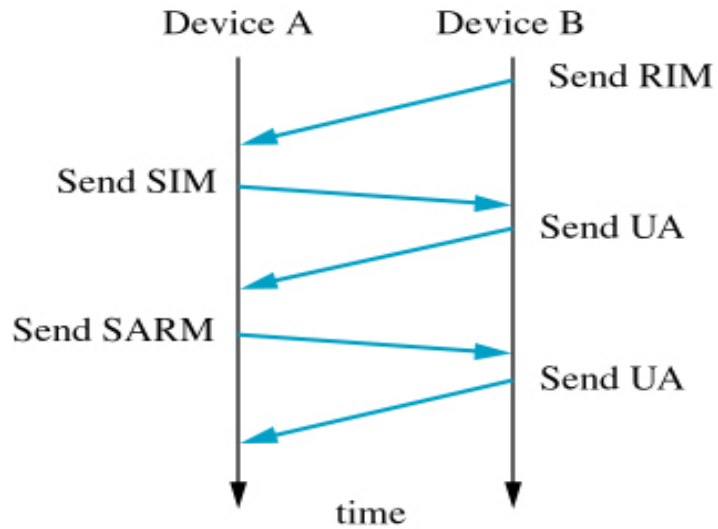


(a) More than five consecutive 1s

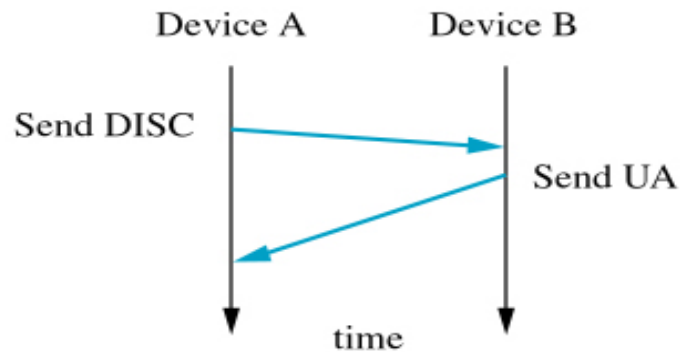


(b) Five consecutive 1s

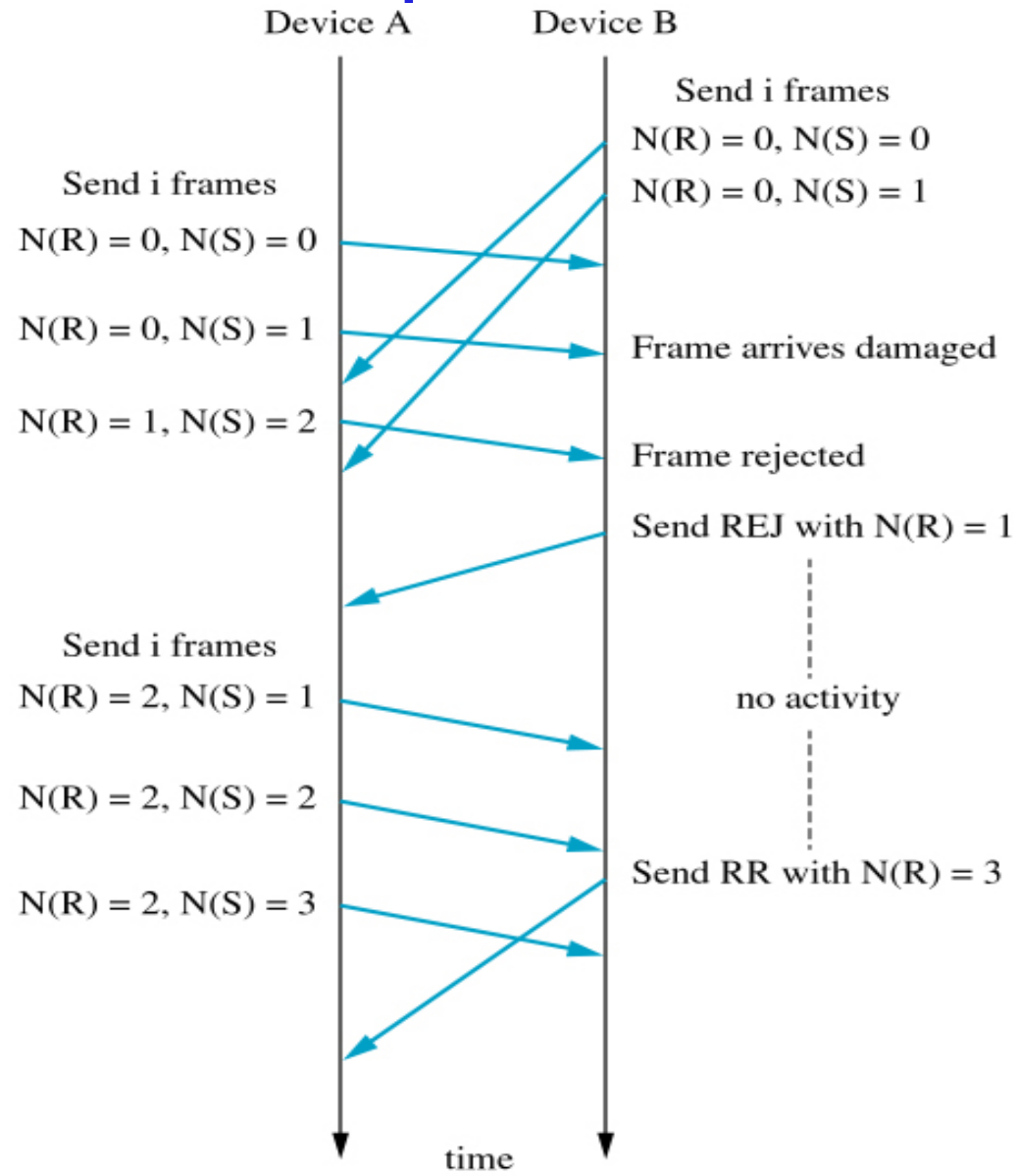
# HDLC communication example



(a) Establishing link



(c) Terminating link



(b) Exchanging frames

## 802.2 LLC Header Formats (if used)

DSAP address 8 bits	SSAP address 8 bits	Control field 8 or 16 bits	Information field N*8 bits
------------------------	------------------------	-------------------------------	-------------------------------

General form  
of LLC header

- DSAP, SSAP are Service Access Point addresses
  - 04 = IBM SNA, 06 = IP,  
AA = SNAP (Subnetwork Attachment Point)

AA   AA   03 LLC	00   00   00 3 octet OUI	08   00 _____
---------------------	-----------------------------	------------------

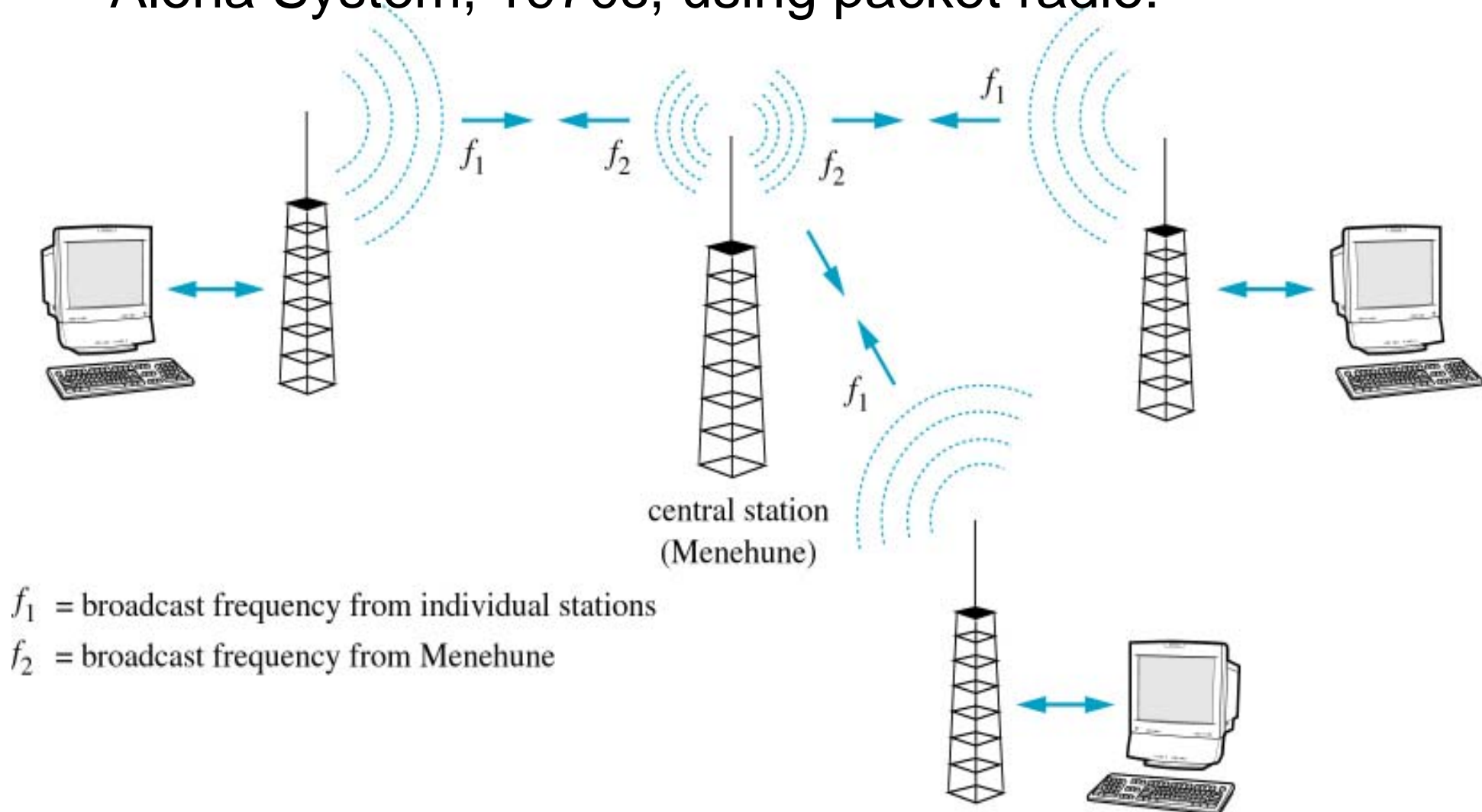
SNAP header  
(8 bytes)

2-octet type field

- OUI = Organisation Unique Identifier
- Type field values are Ethernet type (Ethertype) values
  - $0800_{16}$  = IP, 0806 = ARP, 6003 = DECnet phase IV, ...

# Contention Protocols (Shay 4.7)

- Basic idea: Hosts must *share* the medium
- Aloha System, 1970s, using packet radio:



# Aloha Protocol

- Any host can broadcast a message to Menehune at any time
- If the message is received correctly, Menehune ACKs it (on a different frequency)
- If two host transmissions overlap (and interfere) the message is lost
- If a message is not ACKed the host assumes it was lost, waits a random time, then resends
- Worked and was simple, but not a very efficient use of the medium

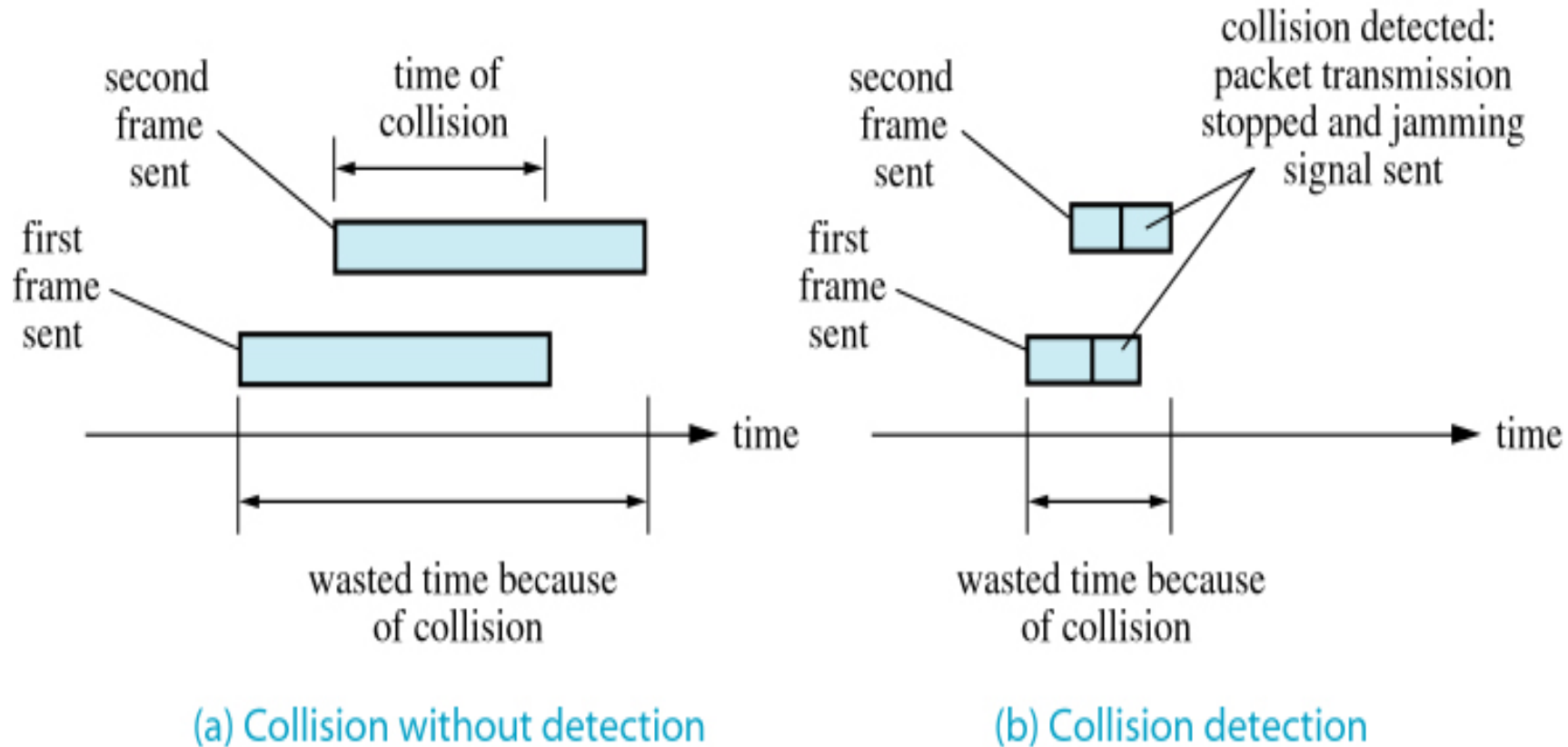


# Carrier Sense Multiple Access (CSMA)

- Like Aloha, *listen to medium* for any activity
- If no activity, transmit; otherwise wait
- Can still get collisions, various ways to reduce them:
  - use 'slot time,' hosts can only transmit at start of a slot
  - random choice, probability  $p$ , to decide whether to transmit or wait for next slot
  - Fig. 4.44 compare various schemes

# Collision Detection

- Start transmitting any time, but watch medium for a collision
- When collision detected, stop transmitting, send jam signal
- This is CSMA/CD



# How to exit a stop sign using CSMA/CD



Oops!



Oops \*!#&\*



OK! 3<sup>rd</sup>  
time  
lucky