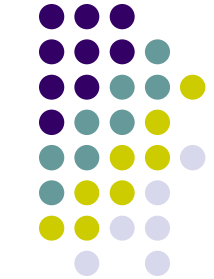
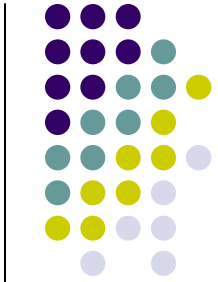
Lecture 4: Reliable Transmission (State Machine Models)



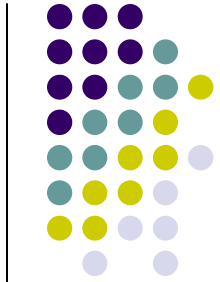
Timothy A. Gonsalves
Professor and Head
Dept. of CSE, IIT Madras

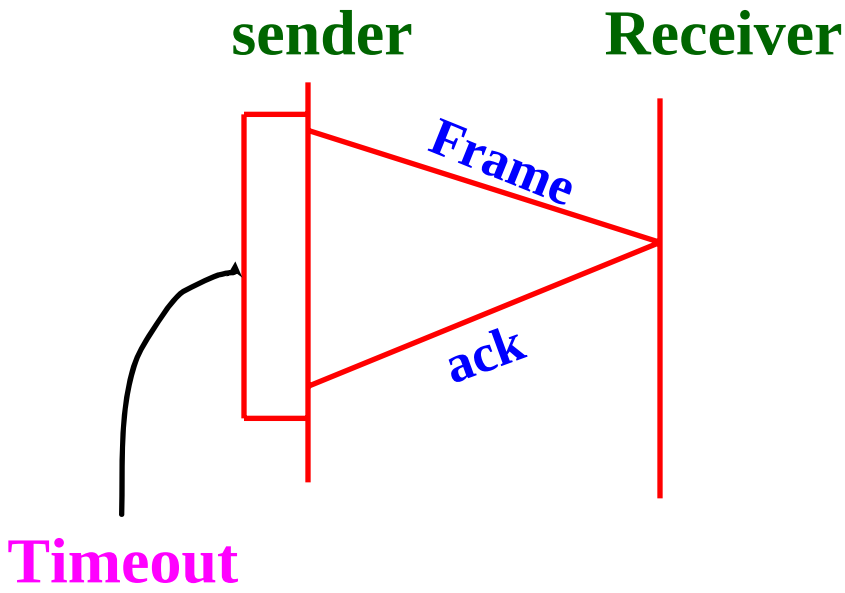
Short Term Course on “Teaching Computer Networks Effectively”. Sponsored by AICTE.

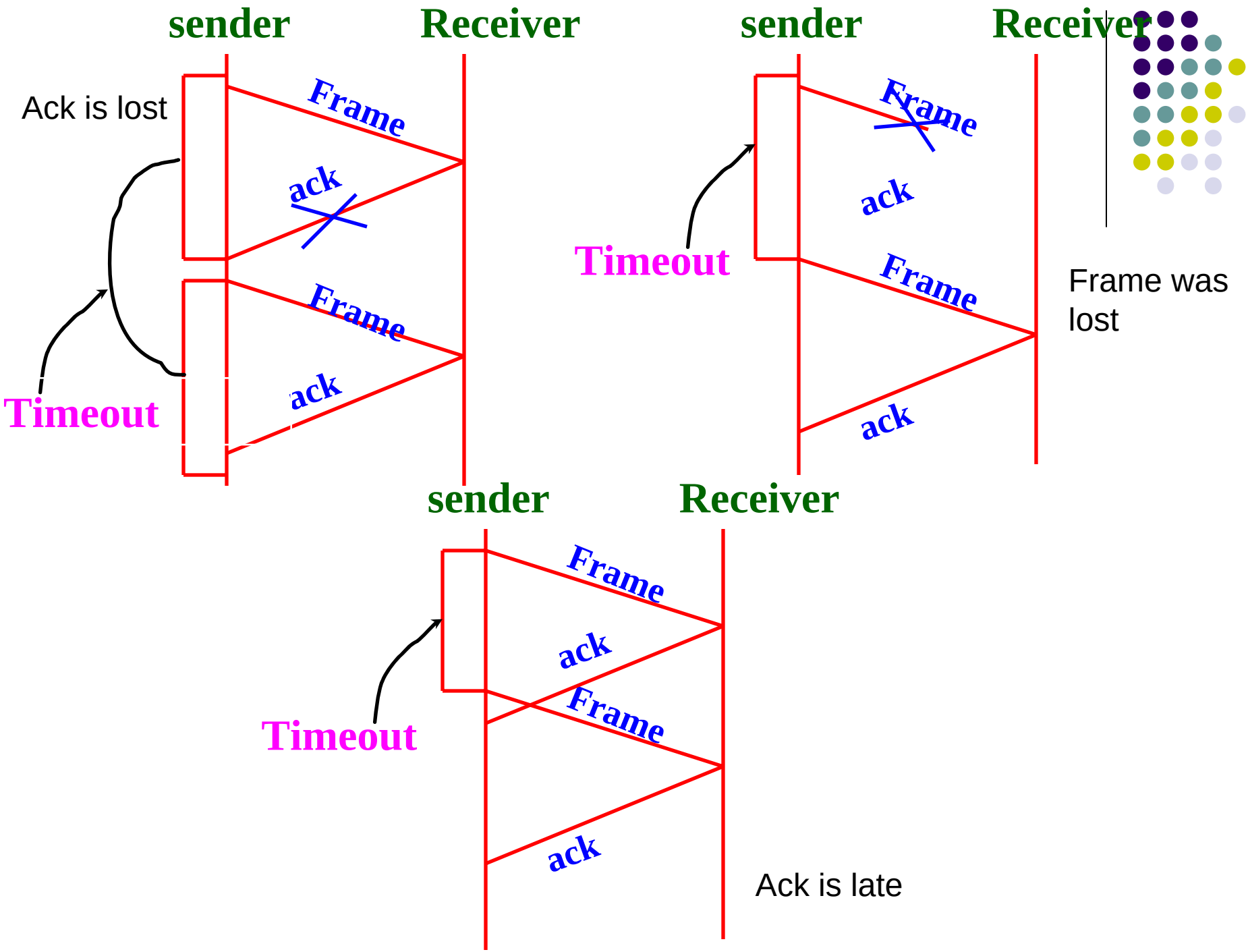
Error control / Reliable Transmission



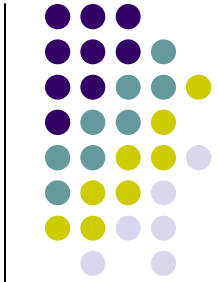
- Acknowledgements (acks)
- Timeouts
- **acks:** a short control frame (header without data)
- **timeout:** sender does not receive ack within finite time retransmit
- **Using acks & timeout:**
- **- Automatic Repeat Request (ARQ)**



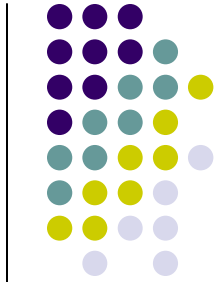

sender
Receiver
Frame
ack
Timeout


sender
Receiver
Ack is lost
Frame
ack
Frame
ack
Timeout
sender
Receiver
Frame
ack
Timeout
Frame
ack
Frame was lost
sender
Receiver
Frame
ack
Frame
ack
Timeout
Ack is late

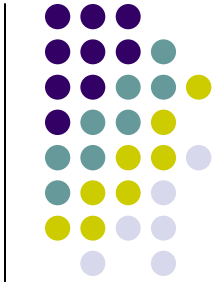
Services



- Sender Process
- Receiver Process
- Service primitives
  - sv = Send(buf, Size, srcSAP, destSAP)
  - rv = Receive(buf, Size, srcSAP, destSAP)

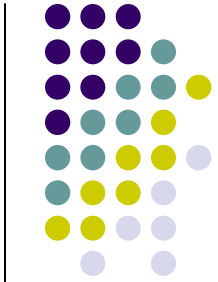


Unrestricted Simplex



- Transport Layer – message
- Network Layer – packetises
  - Packet is sent to Data Link Layer
- Data Link Layer - frames and transmits
  - Two issues to deal with:
    - Fast sender slow receiver
    - Sender swamps receiver

Solution



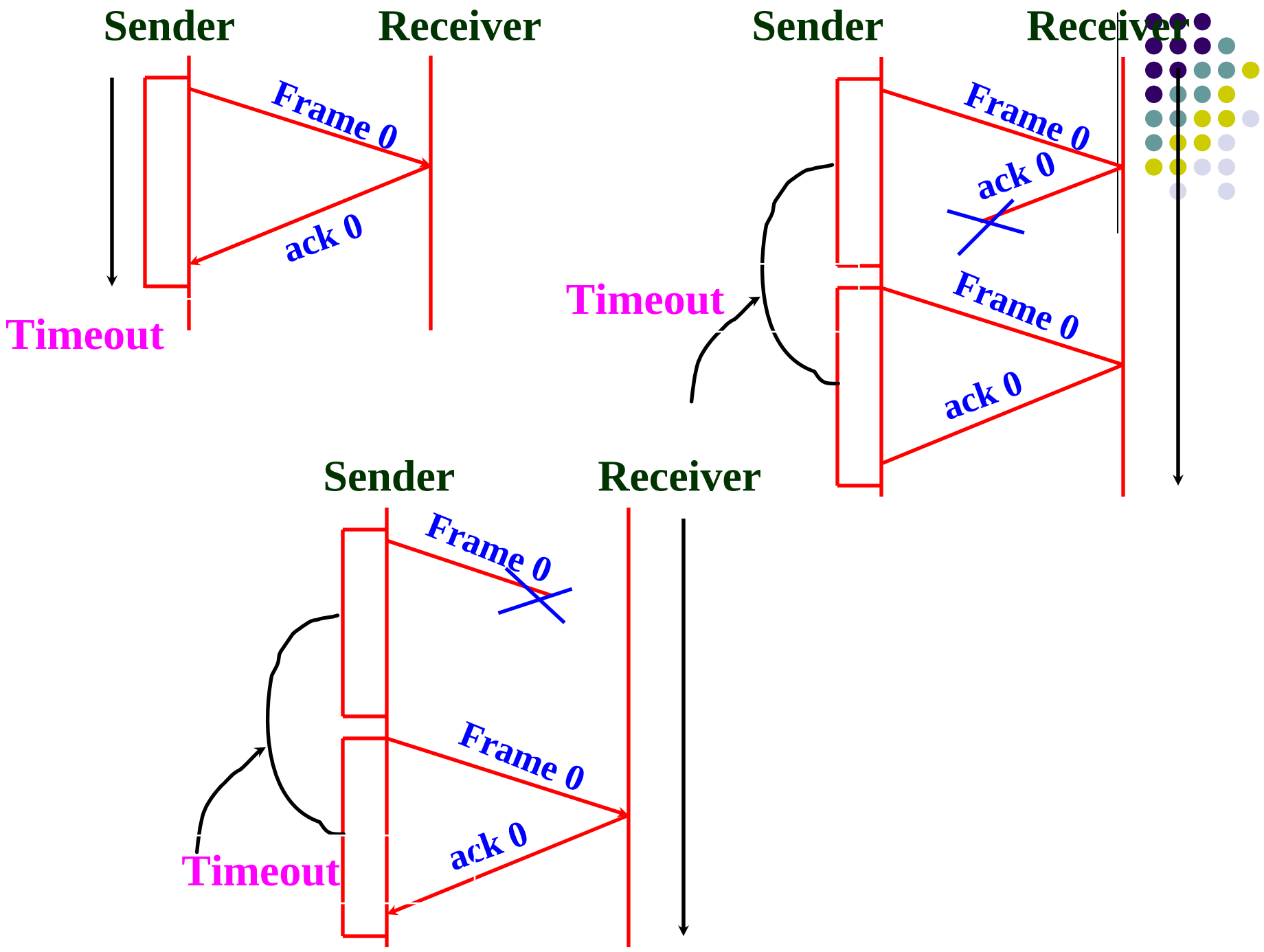
- Slow down sender
  - insert delay in sender (device drivers for plotters, printers)
- Use feed back from receiver
  - send only after acknowledgement is received.

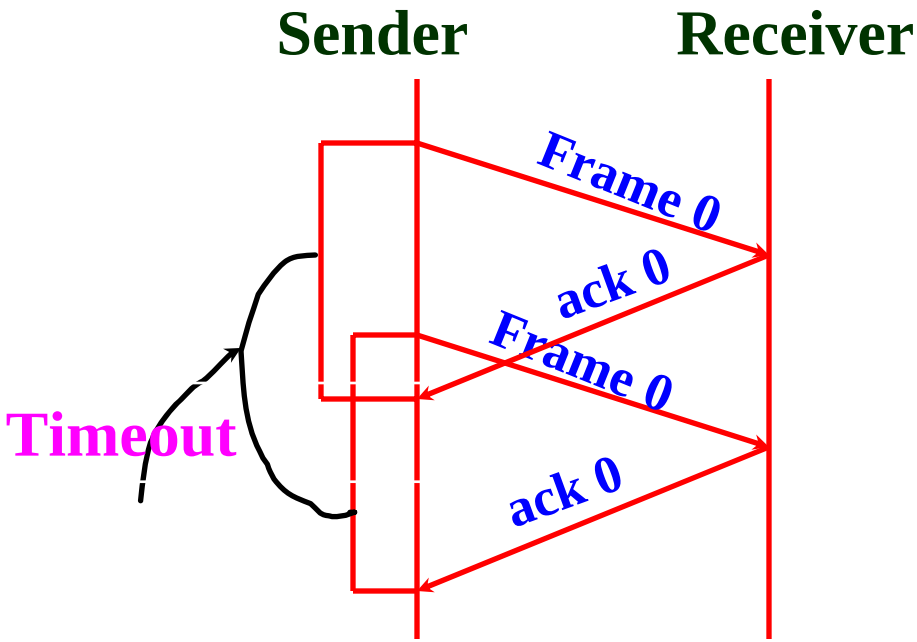
4.1 Stop and Wait Protocol

- Sender sends one frame waits for an ack before proceeding.
  - What if ack lost – sender hangs, therefore timeout.
  - What if receiver is not able to receive: still hangs - number of tries!

Stop and Wait Protocol

- A simple mechanism
  - A frame lost must be resent – to recover from channel characteristics
  - receiver must reply to the event.


Sender
Receiver
Frame 0
ack 0
Timeout
Sender
Receiver
Frame 0
ack 0
Timeout
Frame 0
ack 0
Sender
Receiver
Frame 0
Frame 0
ack 0
Timeout



Require that the sender and receiver take care of all these situations

Sequence number:

Header includes sequence number

modulo 2 counters at receiver and sender

How good is the bandwidth usage with the stop and wait protocol?

- **Example:** 1.5 Mbps link

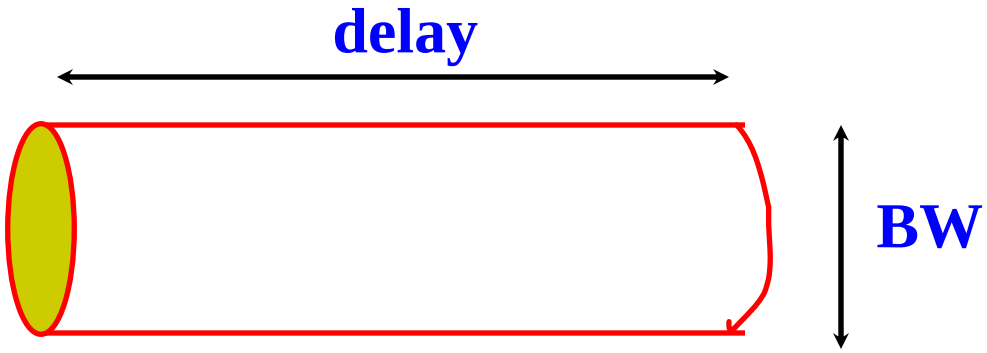
  - RTT – 0.045 s

Propagation delay:

- **delay * BW =** 67.5 kbps

  = delay BW product

- **volume of a link**



delay * BW = volume

How many bits fit in the pipe?

Suppose frame size is **1 KB**

maximum sending rate:

(bits / frame) / (time / frame)

$$= \frac{1024 \times 8}{0.045} = 182 \text{ kbps}$$

$$= \frac{1.5 \times 10^3}{182} = \frac{1500}{182}$$

$$\approx \frac{1}{8} \text{ of link capacity}$$

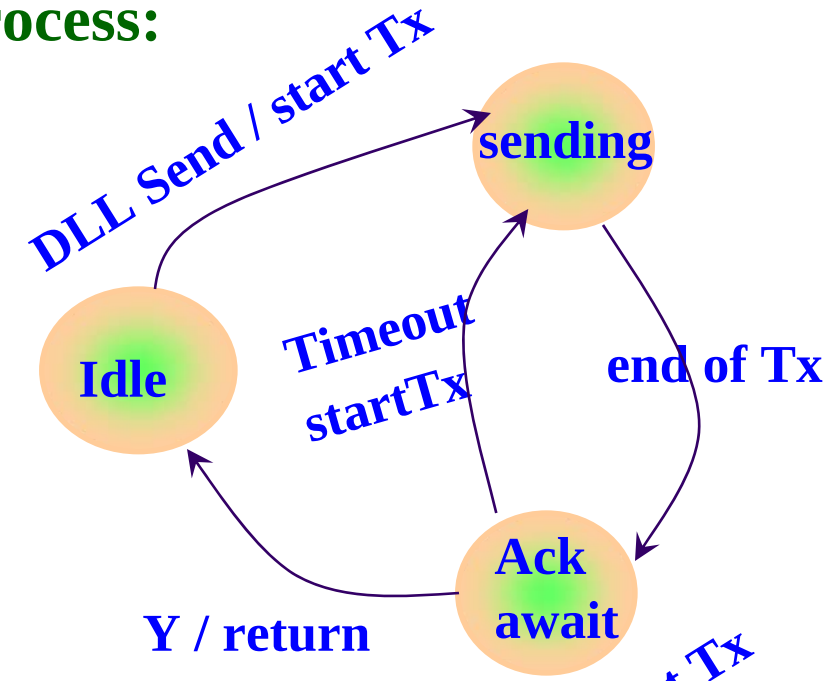
What does delay * BW tell us?

67.5 kbps can be transmitted until an ack is expected.

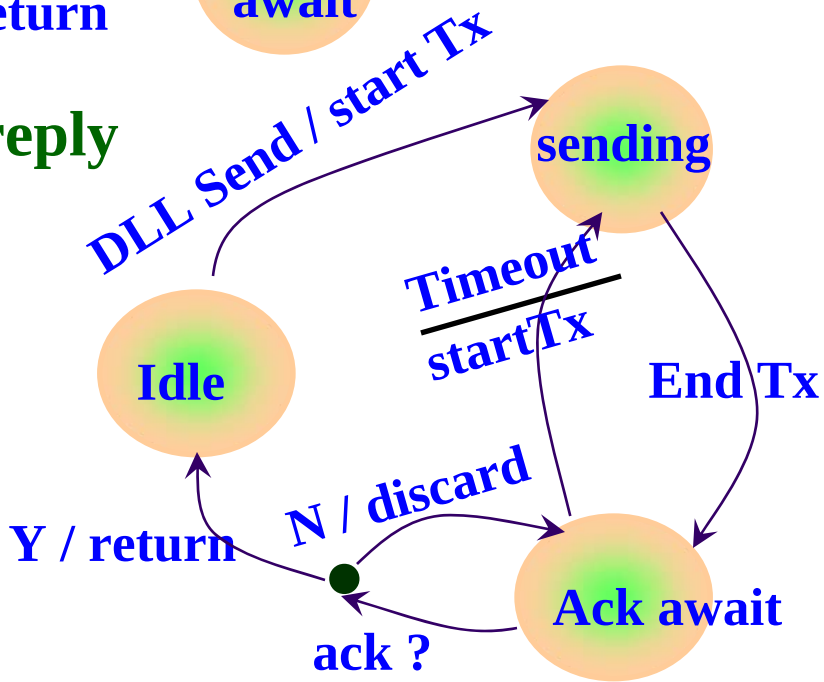
Program as an FSM:

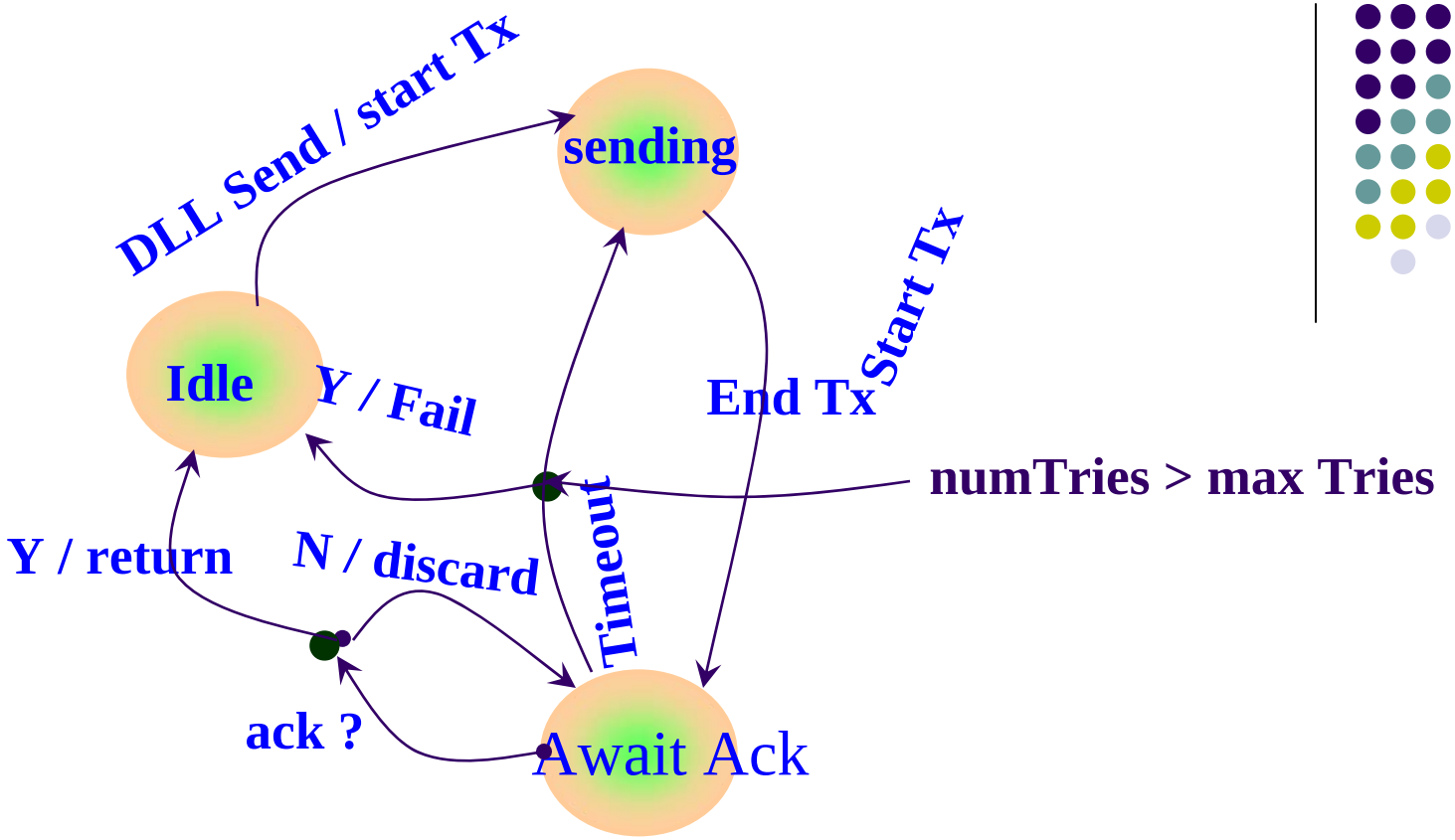
FSM = { states, events, actions}

Sender Process:



What if spurious ----reply





Sending Process (event)

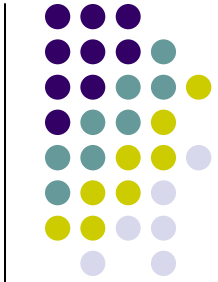
while (event)

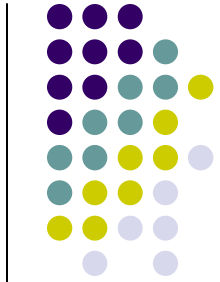
case DLLState **if:**

Idle: if event **=** DLLSend **then**

GetFrame **From** NWL **(buffer)**

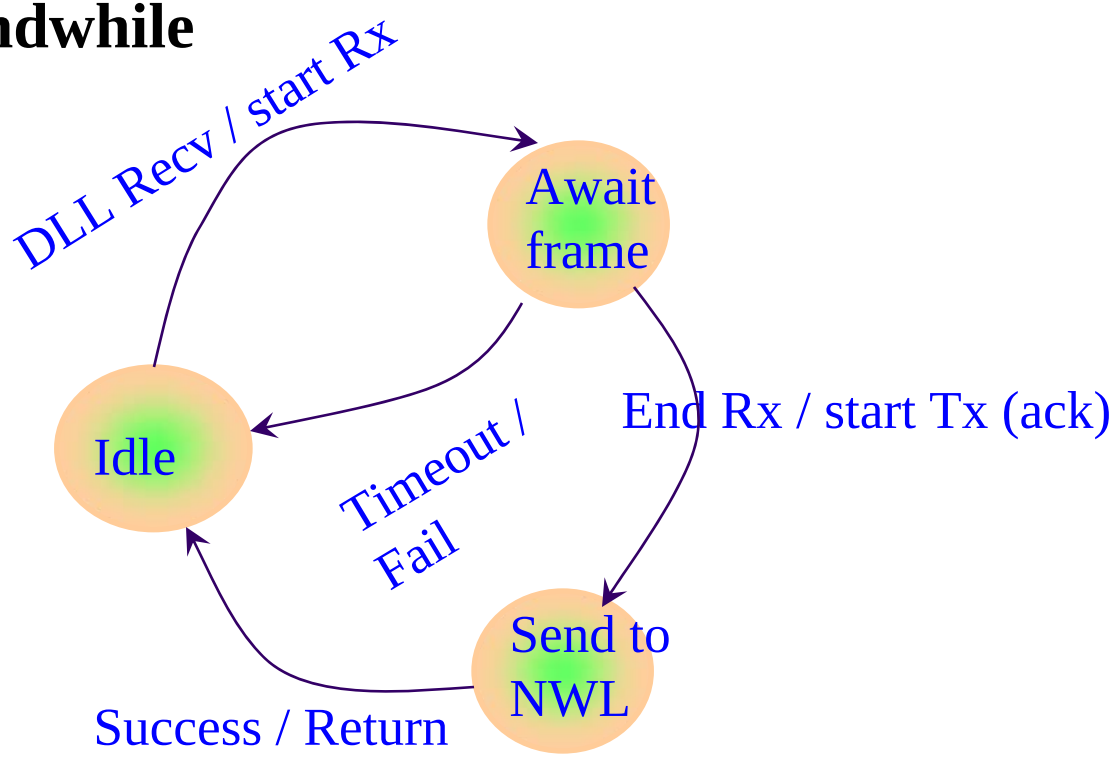
```
            MakeAFrame(buffer, s)
            SendToPhysLayer(s)
            DLLState ← Sending
        else
            error
        endif
Sending: if event = EndTx then
            DLLState ←AwaitAck
        endif
AwaitAck: if event = TimeOut then
            increment numTries
            if numTries > MaxTries then
```





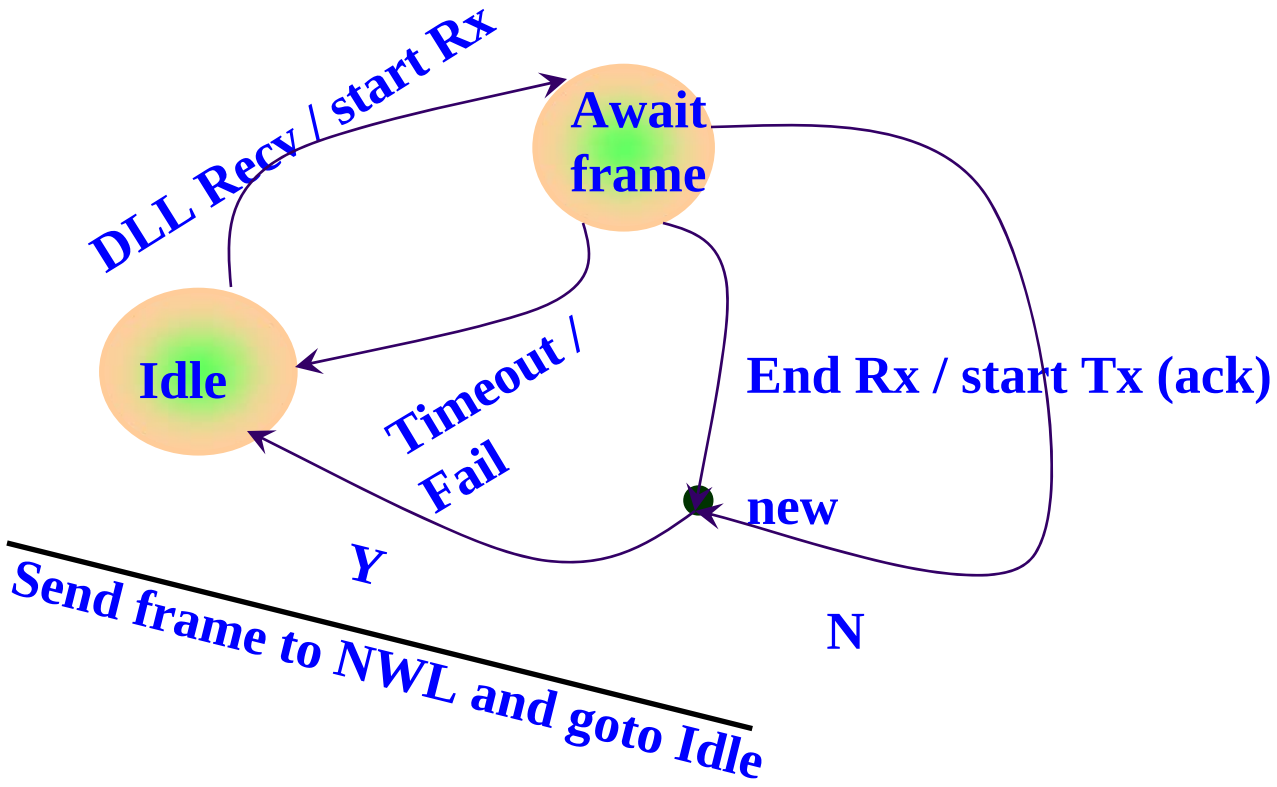
```
        DLLState  ← Idle
        DLLReturn ← Fail
else
        SendToPhysLayer(s)
        DLLState ← Sendif
 endif
 else if event = EndRcv then
       if isAck and SegNo = ExpectedNo then
              DLLState ← Idle
              send Success to upper layer
       else
              discard ack
               DLLstate ← AwaitAck
        endif
```

```
        endif
end case
    wait for Event( )
endwhile
```

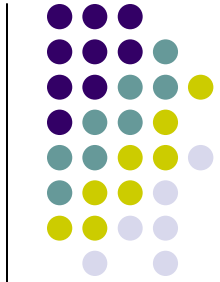


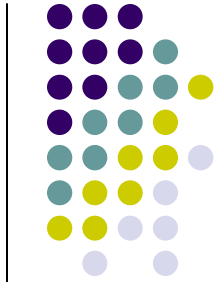
Problem with Duplicate frame:

- if ack lost, sender sends frame again.
- Positive Acknowledgement with Retransmission
- required sequence number on frame



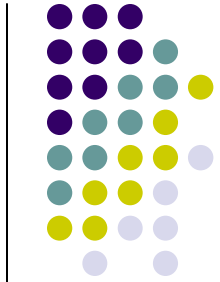
```
pmodule Sender(event – eventType)
    s – frame
  buffer – packet
  DLLStack – state of DLL
   while (event) do
     case DLLState if:
        Idle : if event = DLLSend then
                   getFrame from NWL (buffer)
                   MakeAFrame(buffer, s)
                   DLLState  ←  sending
                   SendTophysLayer(s)
```





```
    else
        error
    endif
Sending: if event = EndTx then
            DLLState ← Idle
        endif
endcase
    wait for An event( )
endwhile
```

```
pmodule Receiver (event)
    r – frame
    event – eventType
    buffer – packet
    while (event) do
        case DLLState if:
            Idle:  if event = DLLRecv then
                        GetFrameFromPhysLayer(s)
                        DLLState ← receiving
                   else
                        error
                   endif
```



Receiving: if event = EndTx then

Make Pkt of Frame(s, buffer)

SendToNWL(buffer)

DLLState ← idle

else

error

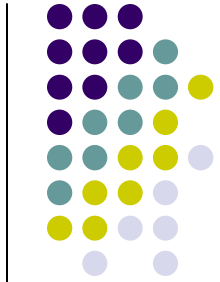
endif

event ← wait for an event( )

event: Check Sum error

instead of DLL Recv

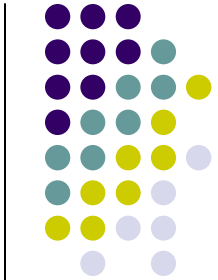
endwhile



4.2 Analysis of Stop and Wait Protocol

- Frame number to be included
- What is the minimum of bits required?
- ambiguity between m and m+1
  - 1 bit sequence number
- sender: knows which frame to send next

Stop and Wait Protocol

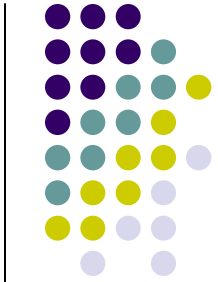


- receiver: knows which frame to expect next
- counters: incremented modulo 2
- Sending process:

  if event = DLL Send then

      increment next FrameNo modulo 2

Stop and Wait Protocol



- Receiver Process:

if event = DLLRecv then

    if recv.Seqnum = expected Seqnum then

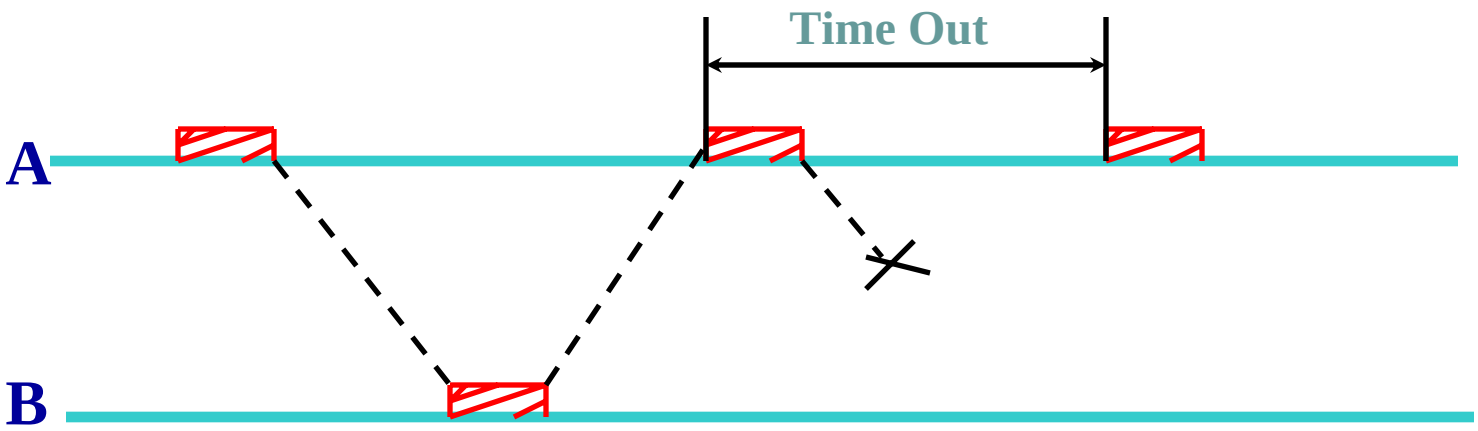
        DLL State = receiving

        getFrameFrom PhysLayer(r, buffer)

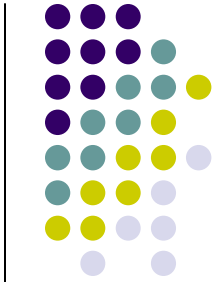
        Sent To NWL(buffer)

        increment NextFrame Expected modulo 2

Stop and Wait: Timing diagram



Throughput



- Error Free Case: Throughput is :

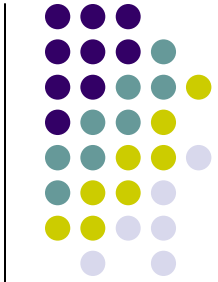
$$U = \frac{T_f}{T_t}$$

T_f - Time take to transmit a frame

T_t - Total time engaged in the transmission of a frame

$$T_t = T_f + T_{prop} + T_{ack} + T_{proc} + T_{prop}$$

Example



- Error free case:
  - Frame size = 10 KB
  - RTT = 100ms = 0.1s
  - Bandwidth = 1 Mbps

$$T_f = 10 \times 8 \times 1024 / (10^6)$$

$$= 0.08192$$

$$T_f + 0.1 = 0.18912$$

$$U = \frac{0.08192}{0.18912} = 0.43$$

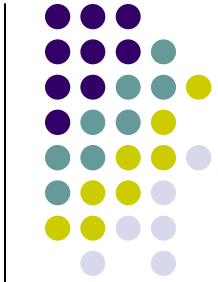
$$Throughput = 430 \ \ kbps$$

Errors in transmission

- Let N_r = E [number of retransmissions]

$$U = \frac{T_f}{N_r T_t}$$

Stop and Wait: Analysis



T_{prop} -- is propagation delay

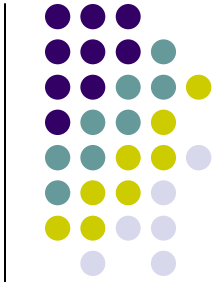
T_{ack} -- time take for acknowledgement

T_{proc} -- time taken for processing at the receiver

If T_{ack}, T_{proc} are negligible then

$$U = \frac{1}{1+2a}, a = T_f / T_p$$

Expected Number of Retransmissions



$$N_r = \sum_{i=1}^{\infty} iP_r[i \text{ transmissions}]$$

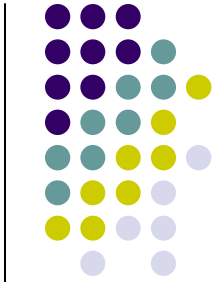
$$= \sum_{i=1}^{\infty} iP^{i-1}(1-P)$$

$$= \frac{1}{1-P}$$

$$U = \frac{(1-P)}{1+2a}$$

where P is the probability of a frame being in error

Error Analysis

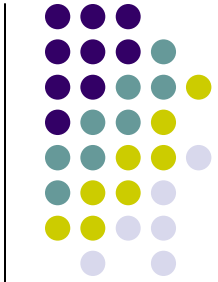


Let p be the probability that a bit is in error

Let F be the number of bits in a frame

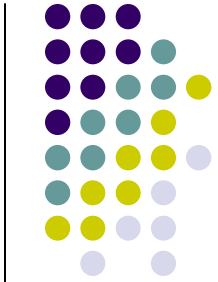
$$P = 1 - (1 - p)^F$$

4.3 Sliding Window Protocol



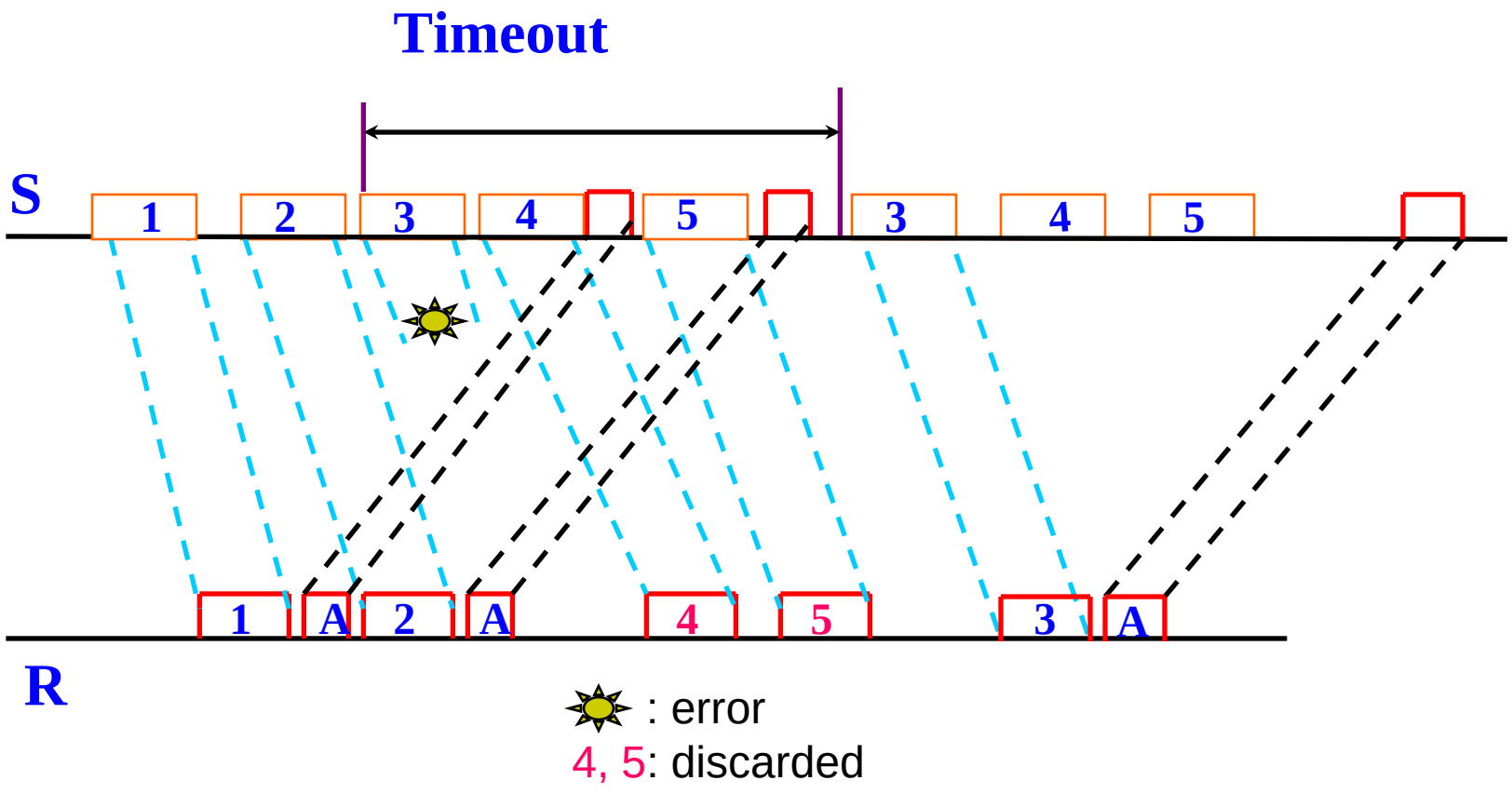
- **Stop & Wait:** inefficient if a is large.
- **Data:** - stream of bulk data
  - data can be pipelined
  - transmit window of data
  - do not worry about getting ack immediately

Sliding Window Protocol

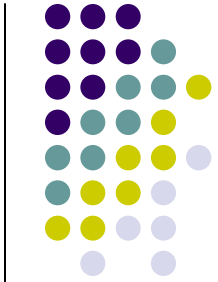


- What should be the size of pipeline?
- How do we handle errors:
  - Sender and receiver maintain – buffer space
  - **Receiver window = 1**
  - **Sender window = n**

Timing Diagram: Go back-N

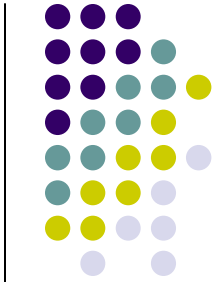


Go-Back N



- Discard if correct frame not received
- Use same circuit for both directions
  - Intermix data frames from both S → R with ack frames from R→ S
- Use type field in header:
  - decide whether data or ack
  - piggy back ack on outgoing frame for R→ S
  - Ack field in frame
  - If frame not available for piggybacking → Timeout

Sliding Window Protocol

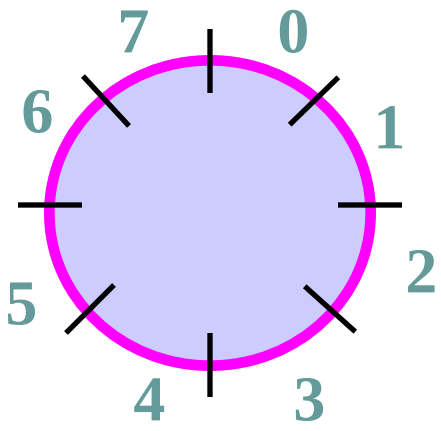


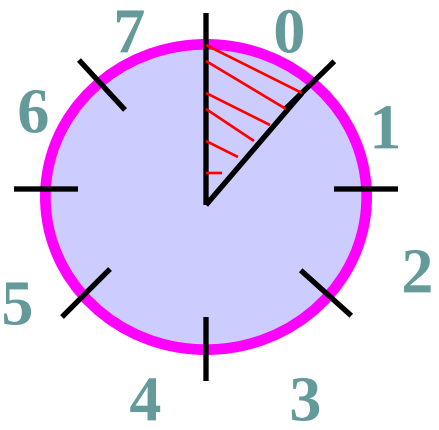
- Outbound frame sequence number
- Range - 0 – 2^n-1
- n bit field
- Stop & Wait is Sliding window with n = 1
- Sender – maintain sequence number of frames it is permitted to send
  - sending window
- Receiver – maintain sequence number of frames it is expected to accept
  - Receiver window

Sliding Window Protocol – An example (Tanenbaum)

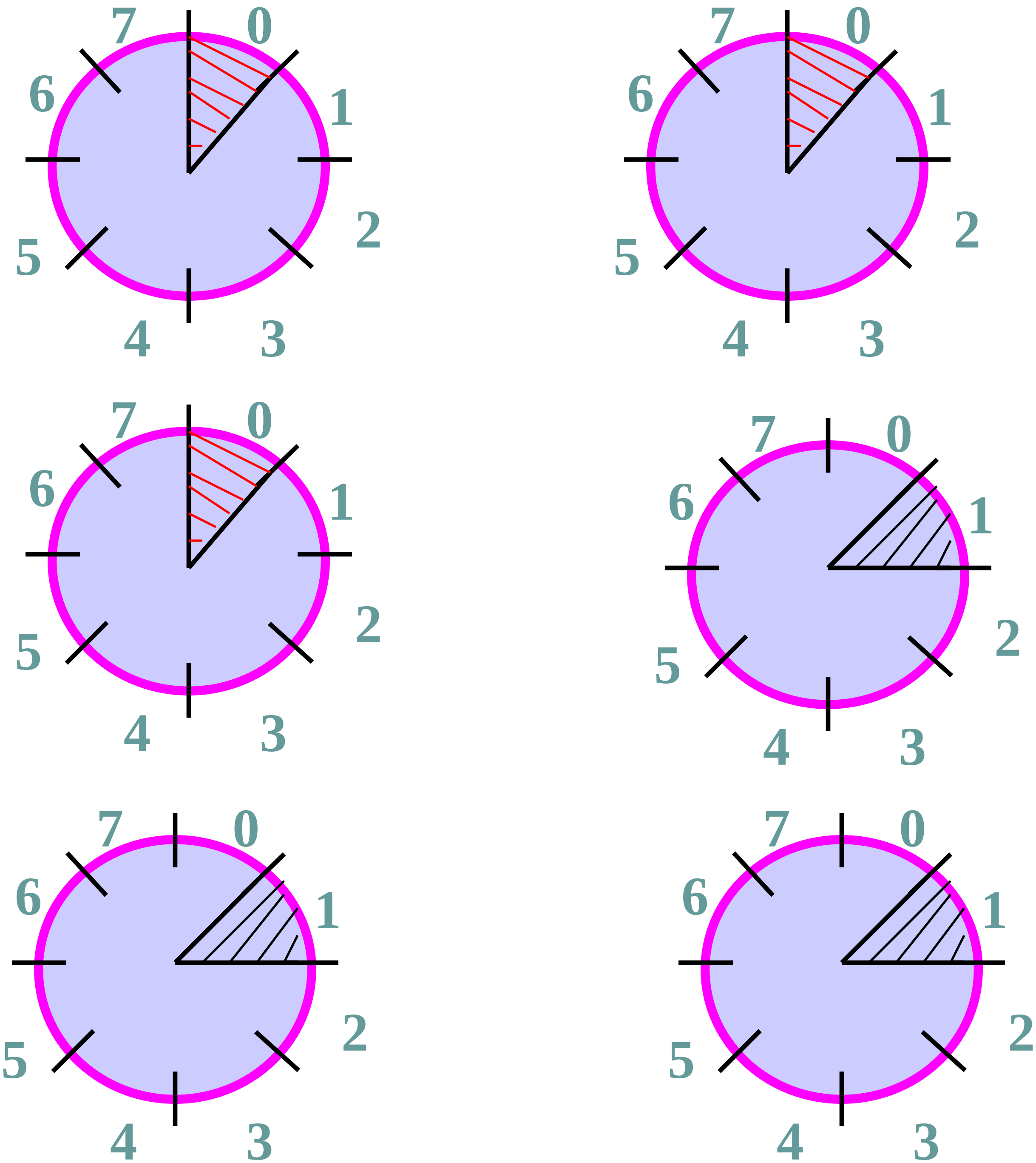
Example: SWP: sequence number: Sender 0 - 7
seqno – 3 bit

Sender





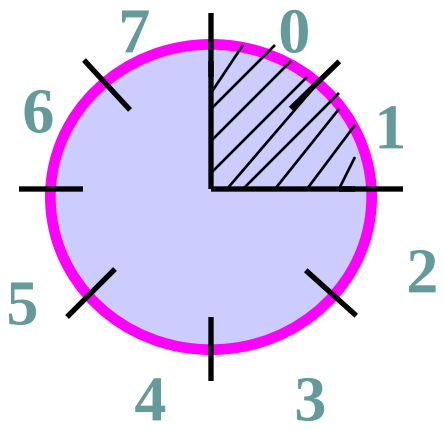
Sender



Receiver

SWP -- Example

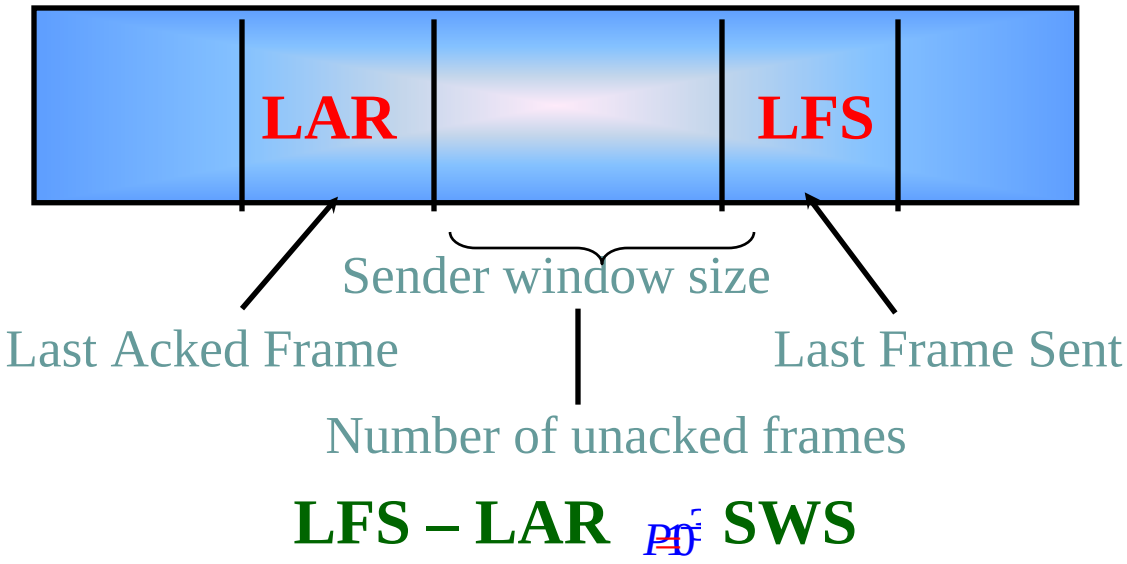
- Larger Sender Window Size



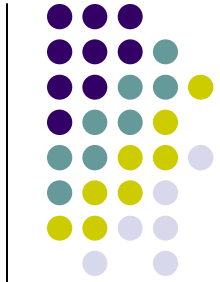
Different Window Sizes: Receiver, Sender (Peterson et al.)

If Sender Window is **n**

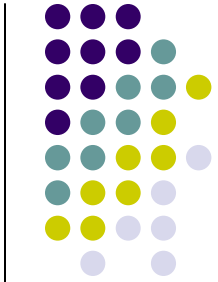
How large can the Receiver Window be?



LFS – LAR ≤ SWS

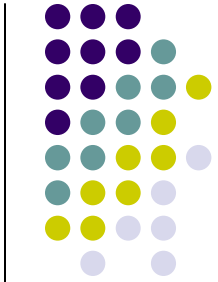


Example: Larger RWS



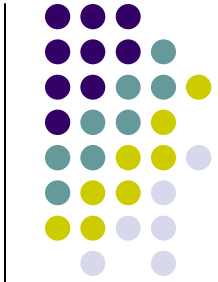
- Example: LFS = 5, RWS = 4, LAF = 9
- If frame 7 & 8 arrive
  - buffered
  - but ack not sent since 6 not arrived.
  - 7 & 8 out of order.
- If frame 6 delayed –
  - Retransmitted, received later
- Notice no NAK for 6.
  - primarily timeout on 6 – retransmit 6.

SWP – Go back-N – a variation



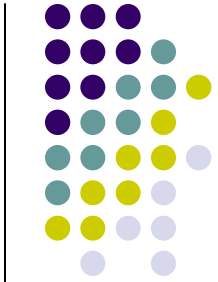
- largest Sequence Number not yet acked.
- receiver only acks SequenceNumberAck even if higher numbered frames are received.
- set LFR = SequenceNumberToAck
- LAF = LFR + RWS

Selective Repeat Protocol



- Variation SWP:
  - selective ack for frame
  - sender knows what to send
  - problem – complicated
  - can RWS > SWS ?

SWS, RWS, Max Sequence Number

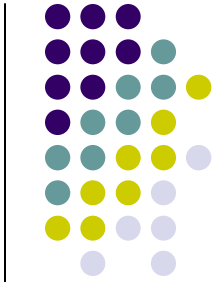


- SWS ≤ MaxSeqNum – 1
- Why? Suppose MaxSeqNum = 7
- Frames sent: 0, 1, 2, 3, 4, 5, 6, 7
- Suppose acks losts
  - Frames resent
- receiver expects 0, 1, 2, 3, .., 7
  - second batch but get duplicate avoid
- 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3

SWS, RWS, Max Sequence Number

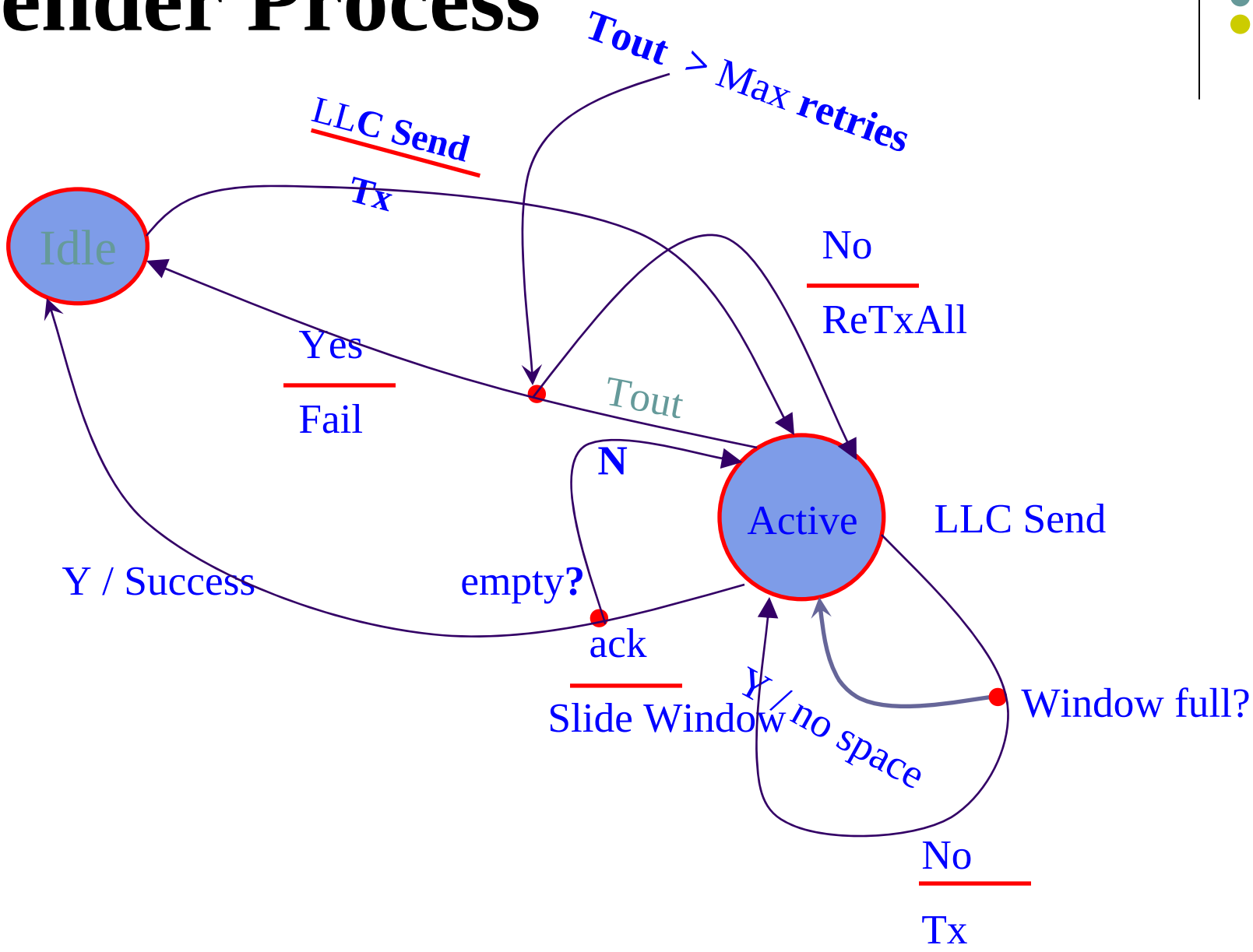
- receiver knows there is a problem when RWS = 1
- what if RWS = SWS = 7
- Sender sends 0,1, 2, …, 6 successfully received – acks lost

SWS and RWS, Max Sequence Number

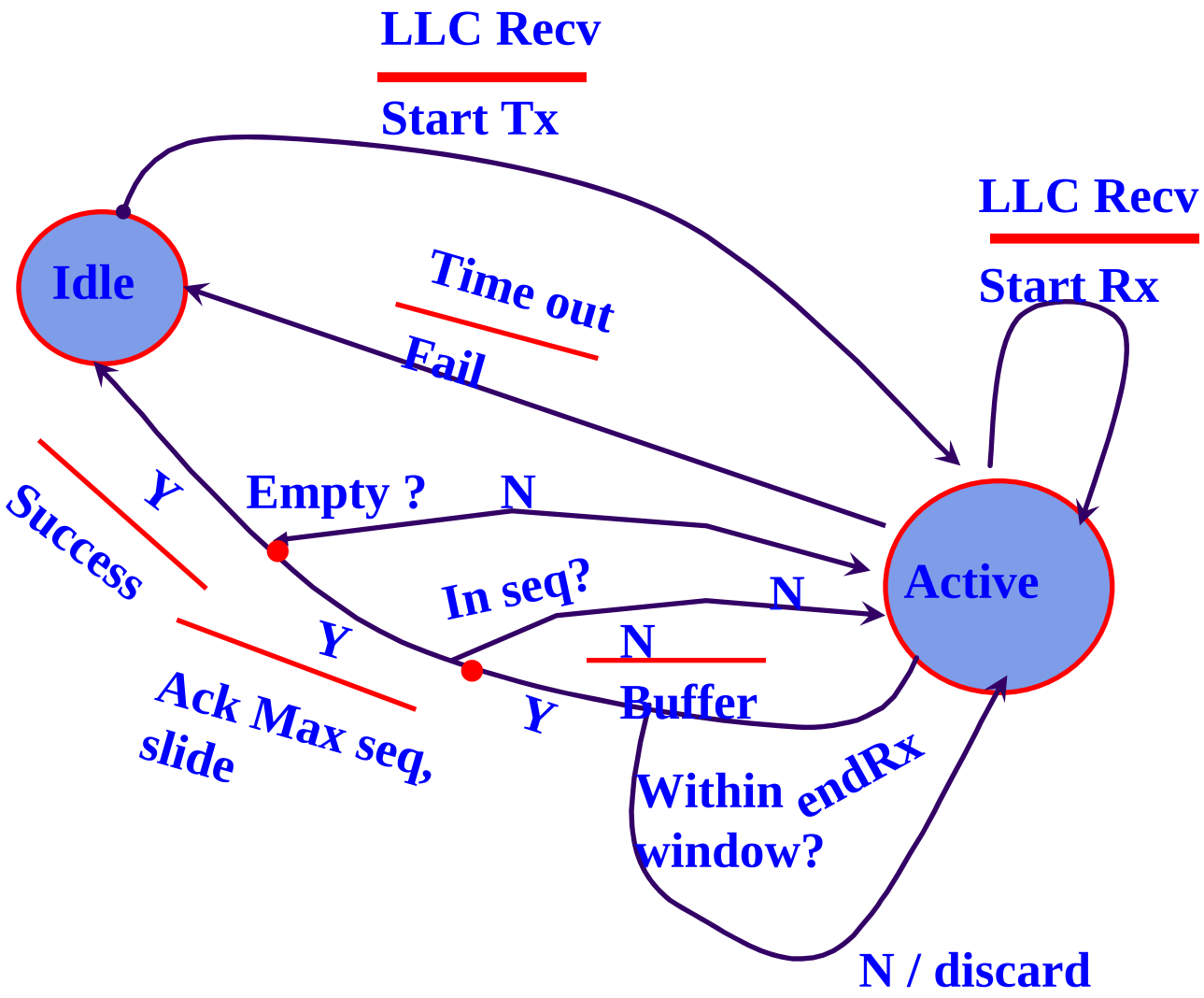


- Receiver expects 7, 0, …, 5
- Sender timeout – sends 0, ..., 6
- Receiver expects second batch
- Sender sends first batch 0, 1, 2, 3
- SWS $\leq$ (MaxSeqNum +1) / 2
- 0, 1, 2, 3 successfully received.
- Next sender sends 4, 5, 6, 7
- What is the rule for RWS < SWS in general?

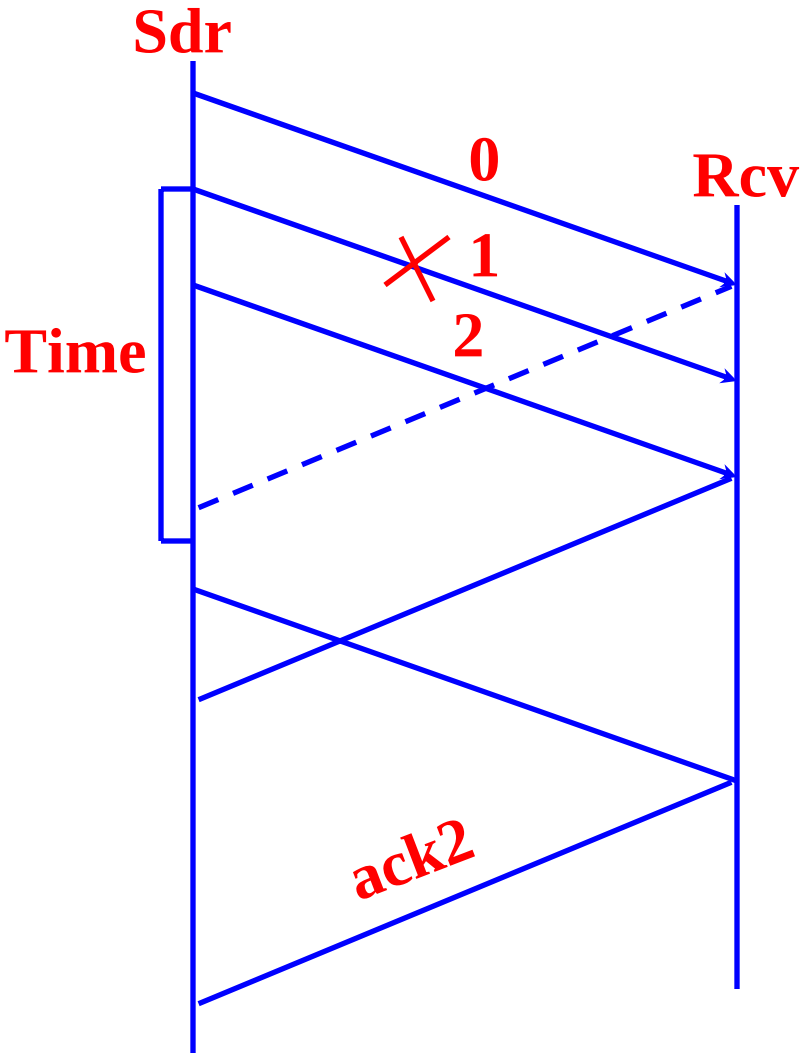
FSM: Sliding Window Protocol: Sender Process



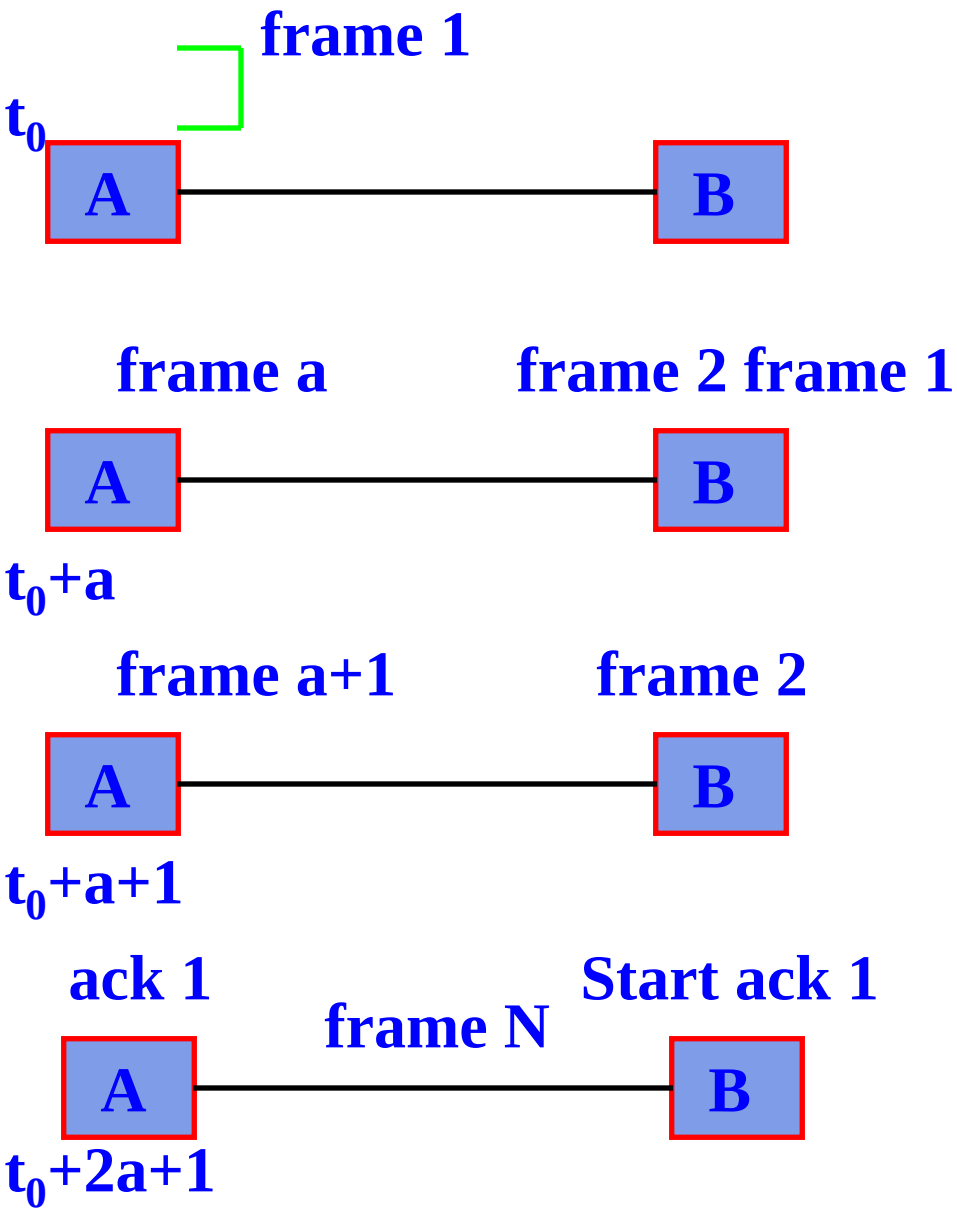
FSM: Sliding Window Protocol: Receiver process:



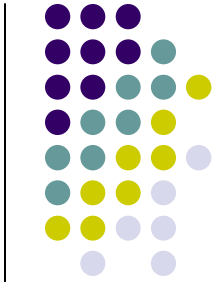
SWP – Timing Diagram



Sliding Window efficiency:

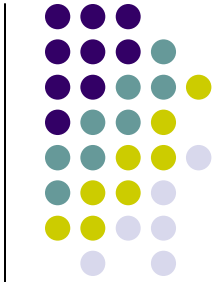


SWP: Efficiency



- Case 1: *N > 2a+1*
- A transmits continuously without pause
  - *U = 1*
- Case 2: *N < 2a+1*
  - *U = N / 2a+1*

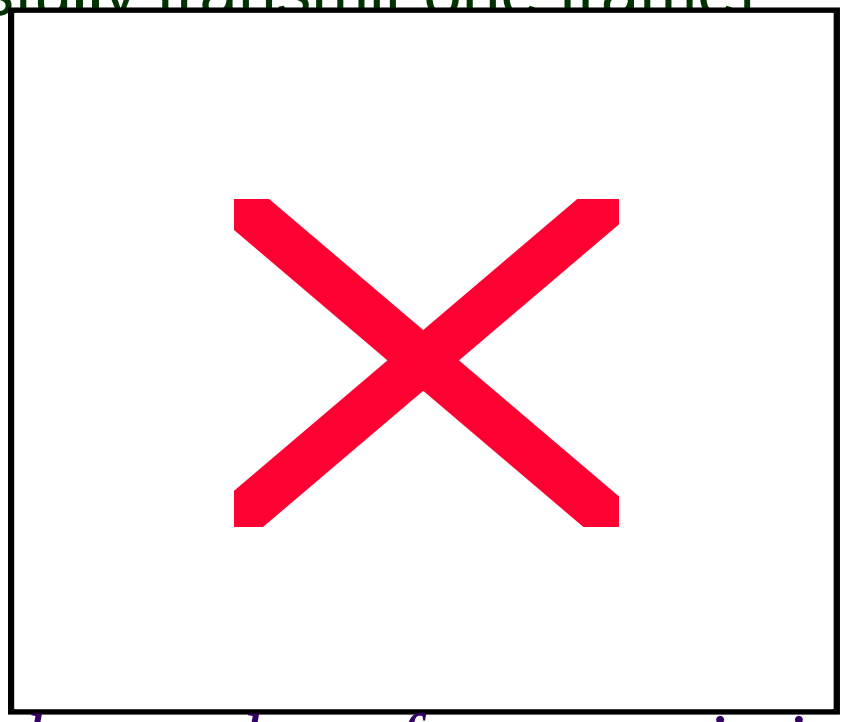
Summary



- Stop and wait -- simple but inefficient for large *a*
- Sliding window -- efficient
- Window size must be large enough to fill pipe
- FSM -- a powerful technique for design and implementation of protocols
- Space-time diagrams for protocol design and analysis

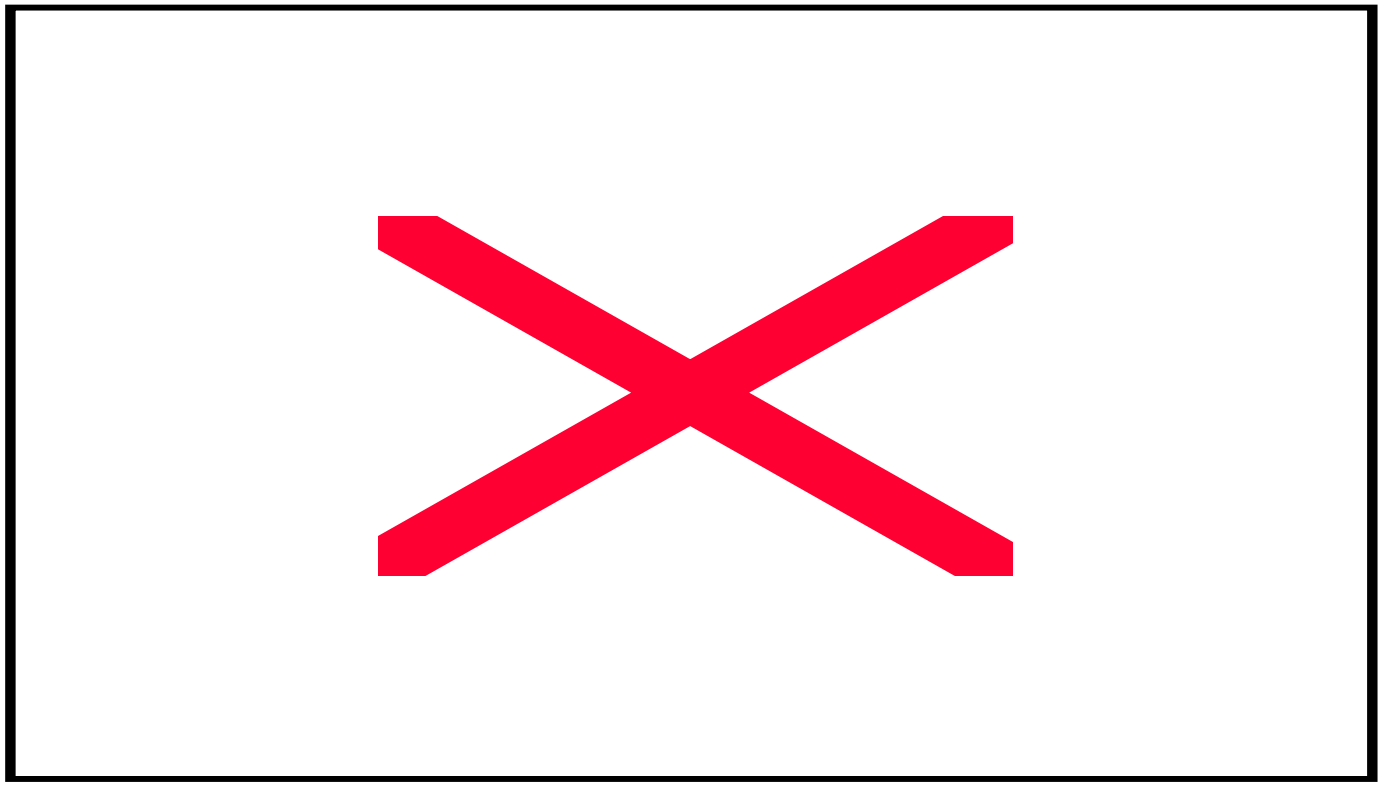
SWP: Transmission with errors

- N_r = E [number of transmitted frames to successfully transmit one frame]



k is the number of retransmissions of a frame

Approximation for k



Utilisation for different protocols (Stallings)