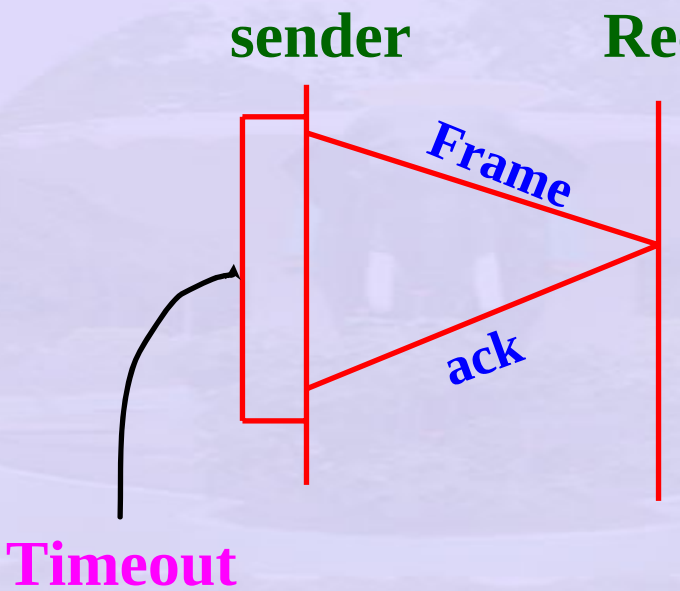
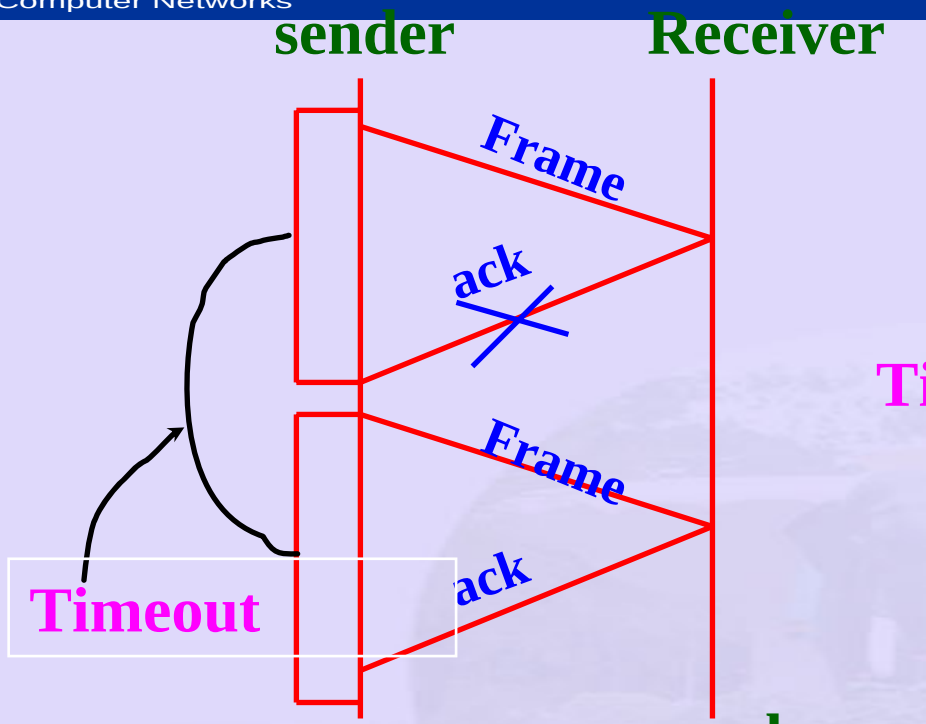
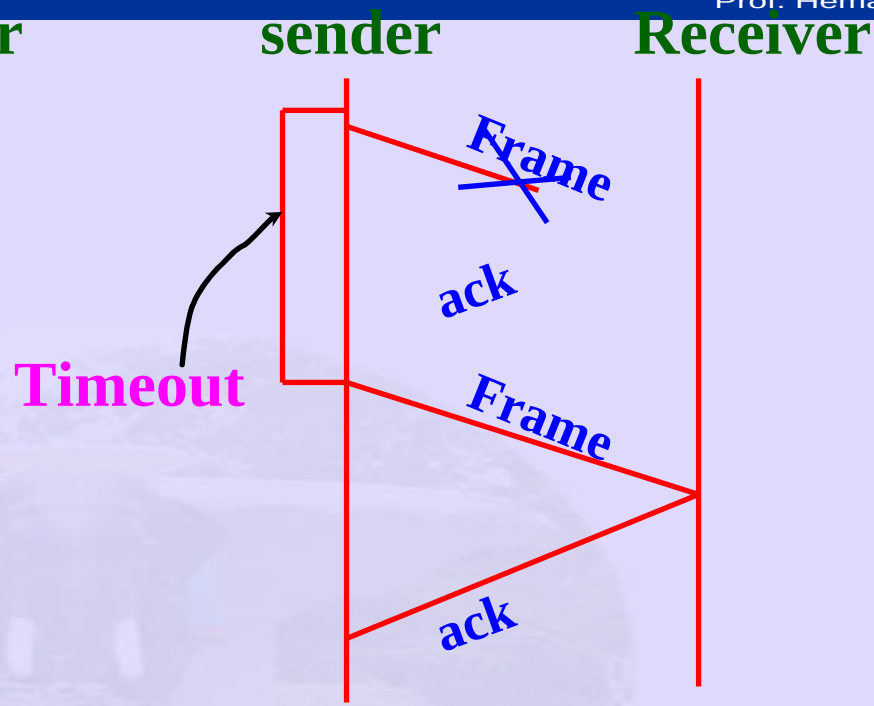
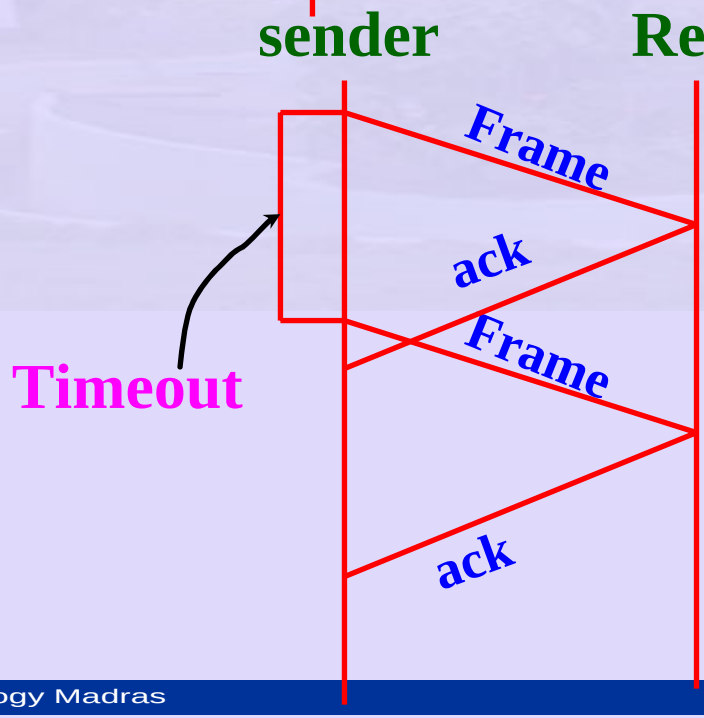
Error control / Reliable Transmission

- 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
Frame
ack
Frame
ack
Timeout

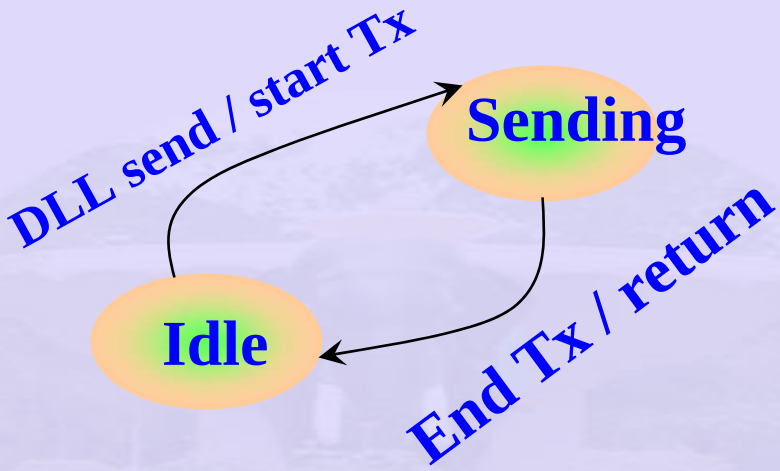

sender
Receiver
Frame
ack
Timeout
Frame
ack


sender
Receiver
Frame
ack
Frame
ack
Timeout

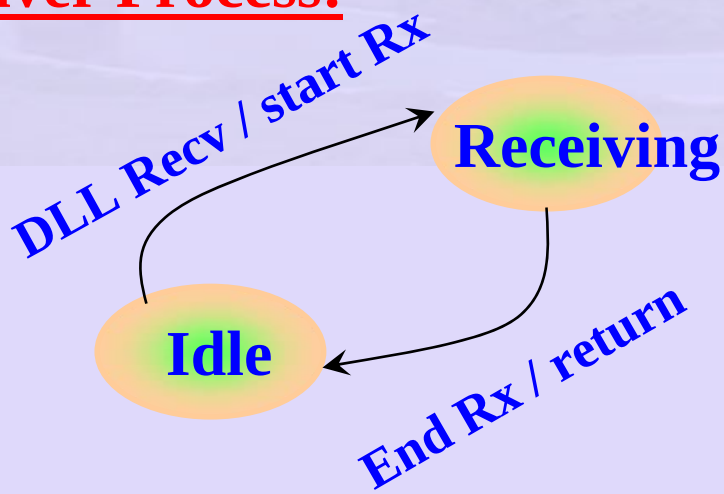
Services

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

Sender process:



Receiver Process:



Unrestricted Simplex

- Transport Layer – message
- Network Layer – packetises
- packet – send to Data Link Layer
- Data Link Layer - frames and transmits
  - Fast sender slow receiver
  - Sender swamps receiver

Solution

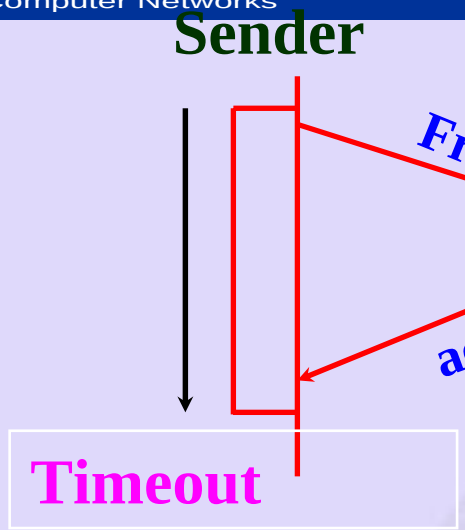
- Slow down sender
  - insert delay in sender (device drivers for plotters, printers)
- Use feed back from receiver
  - send only after acknowledgement is received.

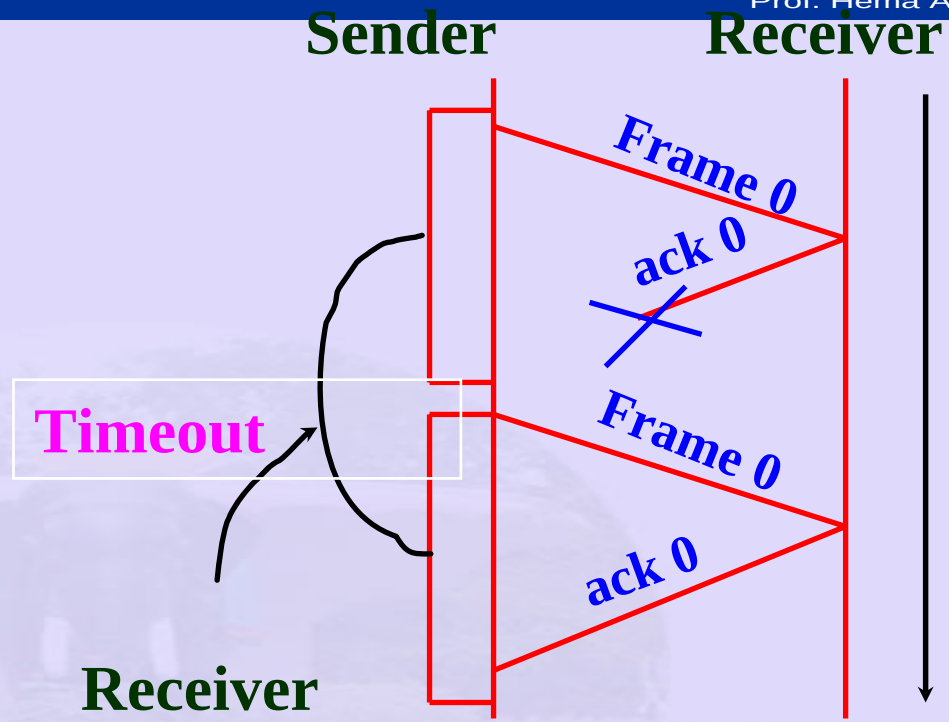
Stop and Wait Protocol

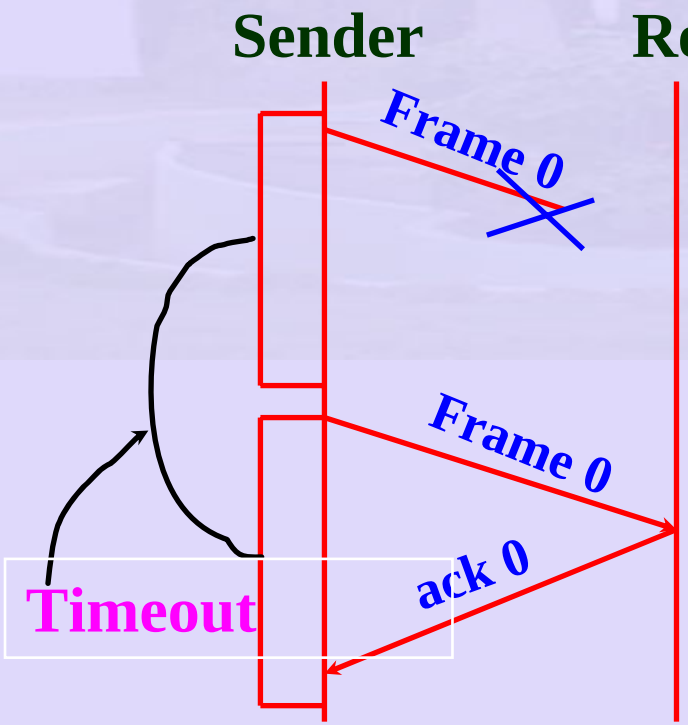
- Sender sends one frame waits for an ack before proceeding.
  - What if ack lost – sender hangs, therefore timeout.
  - What if receiver is not able to receive: still hangs - number of tries!

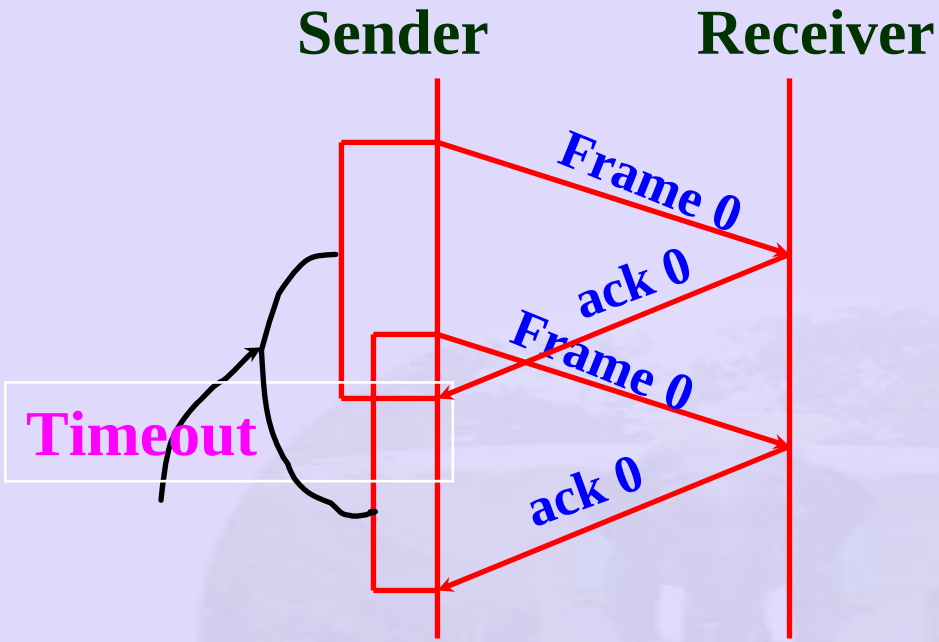
Stop and Wait Protocol

- 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



Basically require that the sender and receiver take care of all these situation.

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

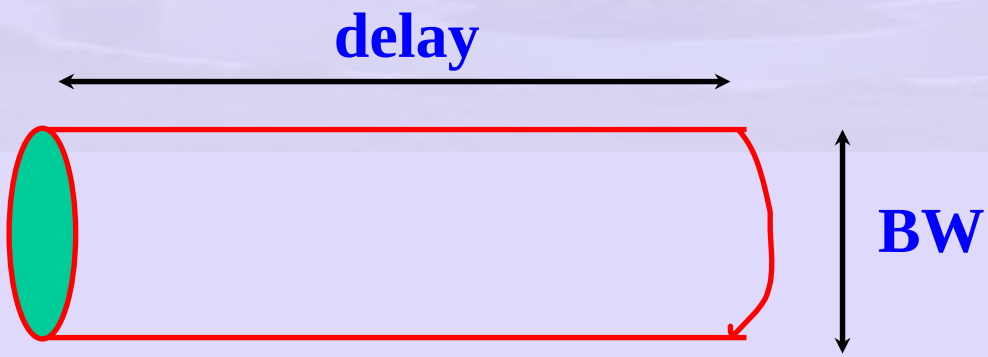
  - RTT – 0.045 s

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)

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

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

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

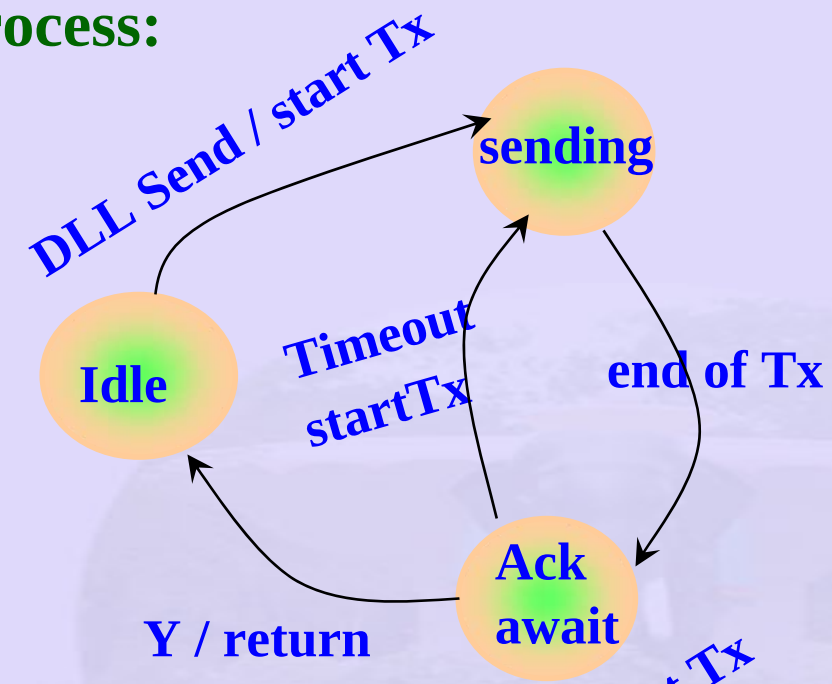
What does delay * BW tell us?

67.5 kbps can be transmit until an ack is expected.

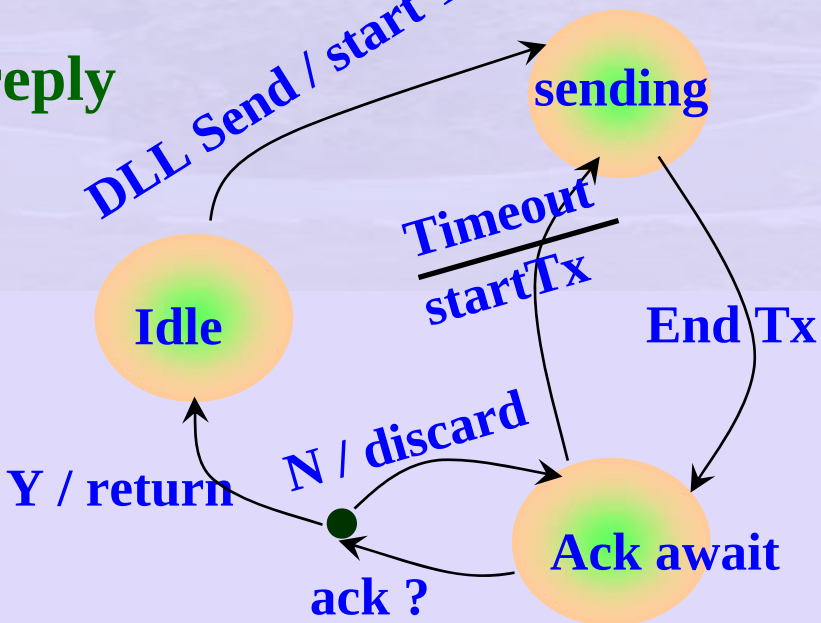
Program as an FSM:

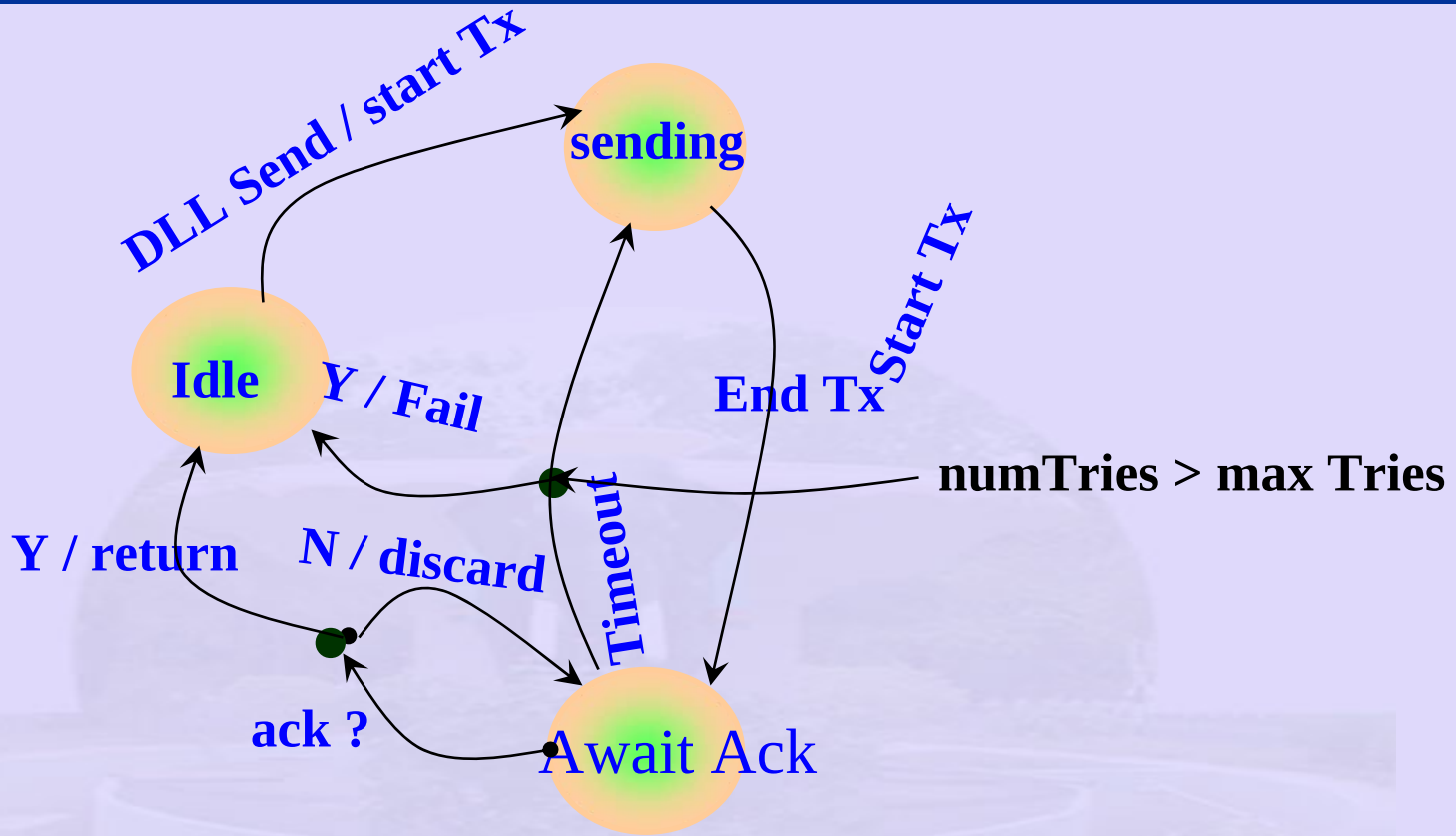
FSM = { states, events, actions}

Sender Process:



What if spurious ----reply





Sending Process (event)

while (event)

case DLLState if:

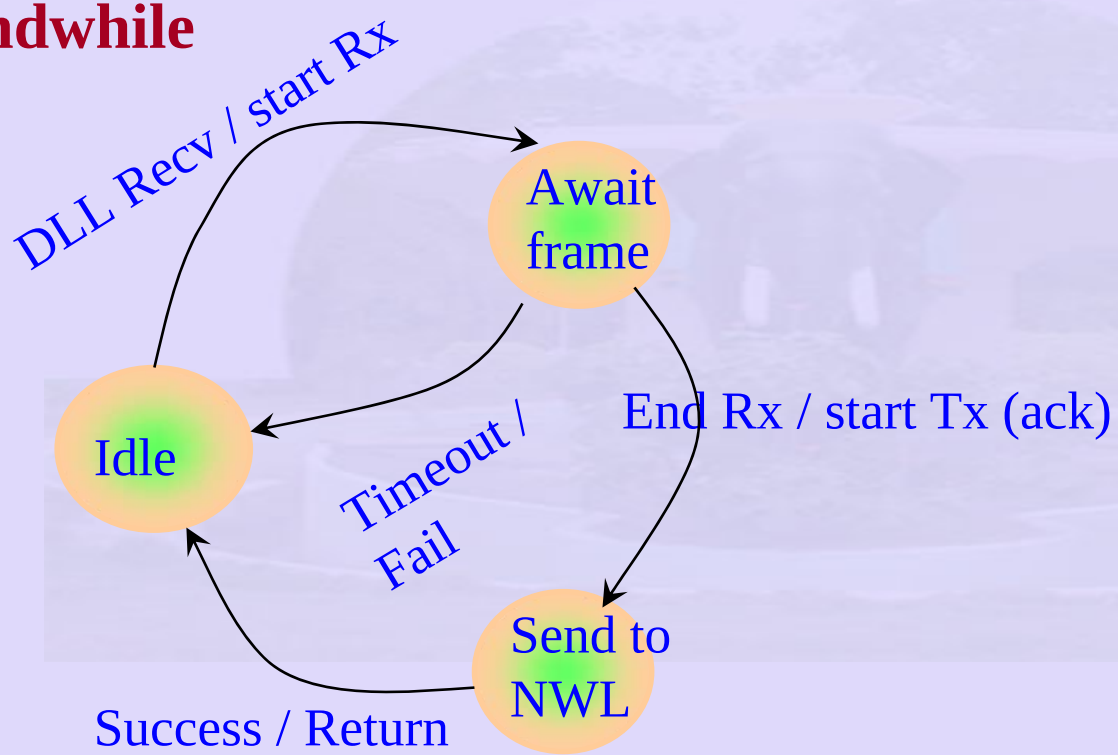
Idle: if event = DLLSend then

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
```

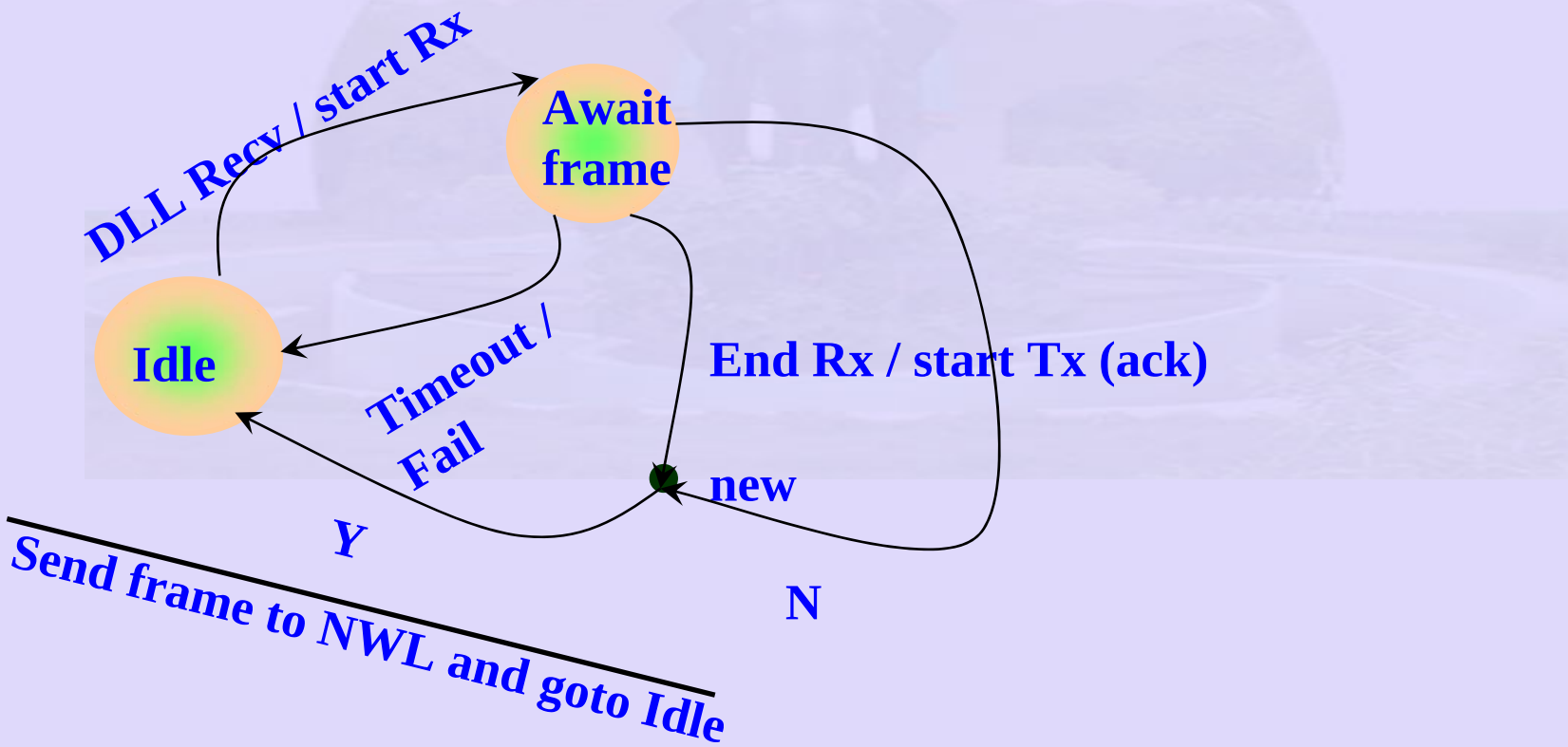
```
            DLLState  ← Idle
            DLLReturn ← Fail
    else
            SendToPhysLayer(s)
            DLLState ← Sendif
     endif
     else if event = EndRcv then
          if isAck and SegNo = ExpectedNo then
                DLLState ← Idle
                send Success to upper layer
 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