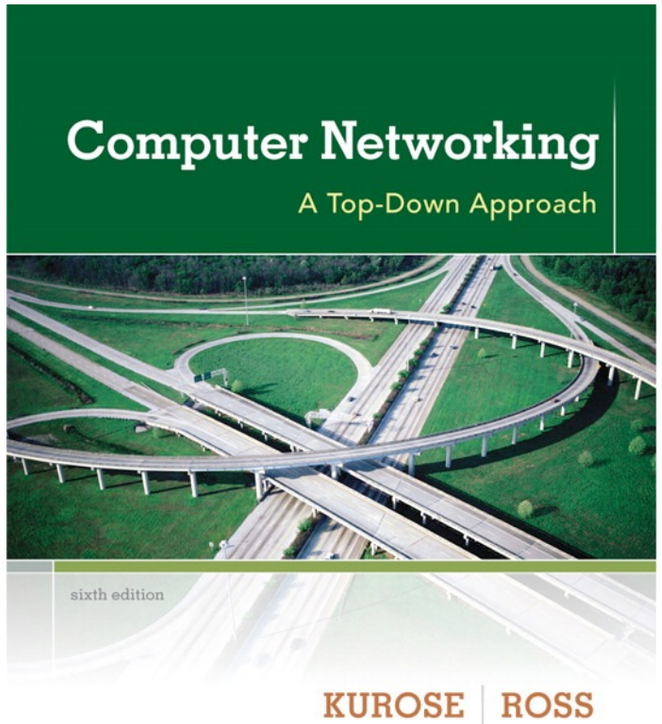
CN-Basic
L20

Reliable Data Transfer

Dr. Ram P Rustagi
rprustagi@ksit.edu.in
http://www.rprustagi.com
https://www.youtube.com/rprustagi

Chapter 3
Transport Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

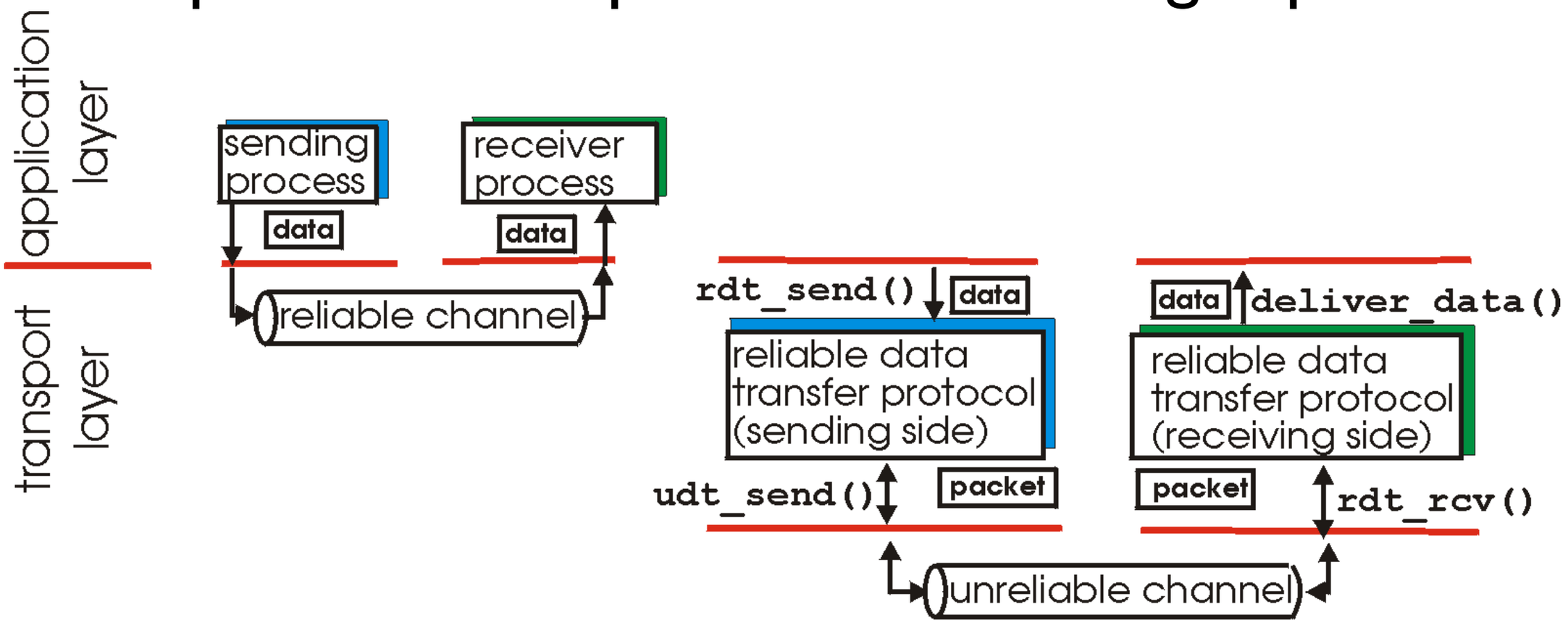
All material copyright 1996-2012
J.F Kurose and K.W. Ross, All Rights Reserved

©

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Principles of Reliable Data Transfer

- Important in application, transport, link layers
  - Top-10 list of important networking topics!



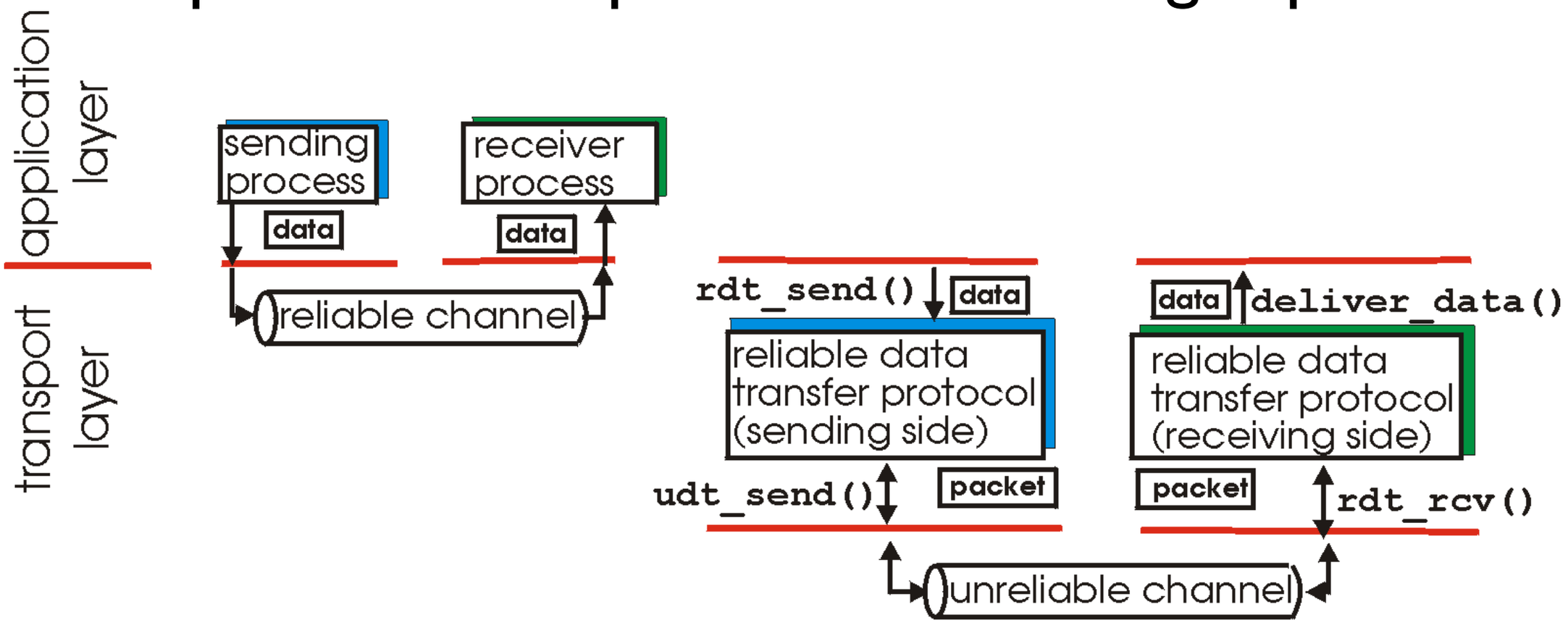
(a) provided service

(b) service implementation

- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable Data Transfer

- Important in application, transport, link layers
  - Top-10 list of important networking topics!

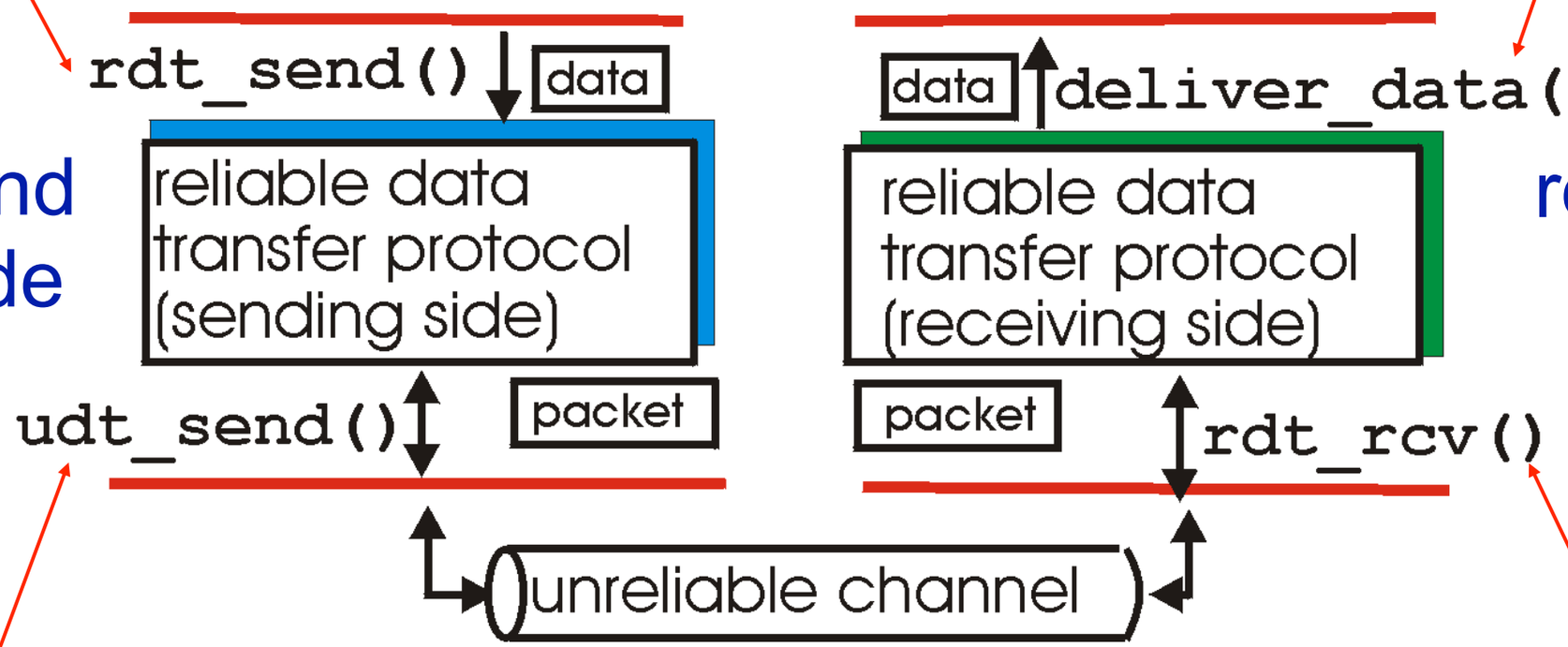


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable Data Transfer: Getting Started

`rdt_send()`: called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

`deliver_data()`: called by **rdt** to deliver data to upper



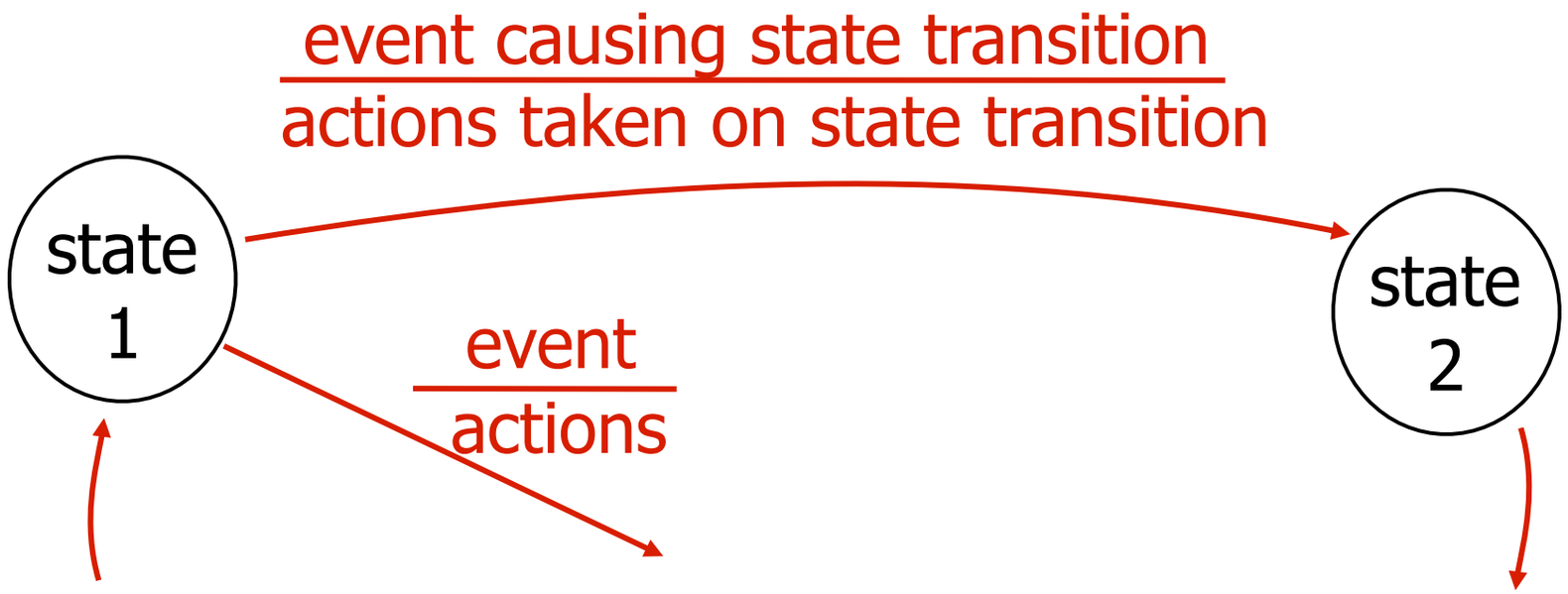
`udt_send()`: called by rdt, to transfer packet over unreliable channel to receiver

`rdt_rcv()`: called when packet arrives on rcv-side of channel

Reliable Data Transfer: Getting Started

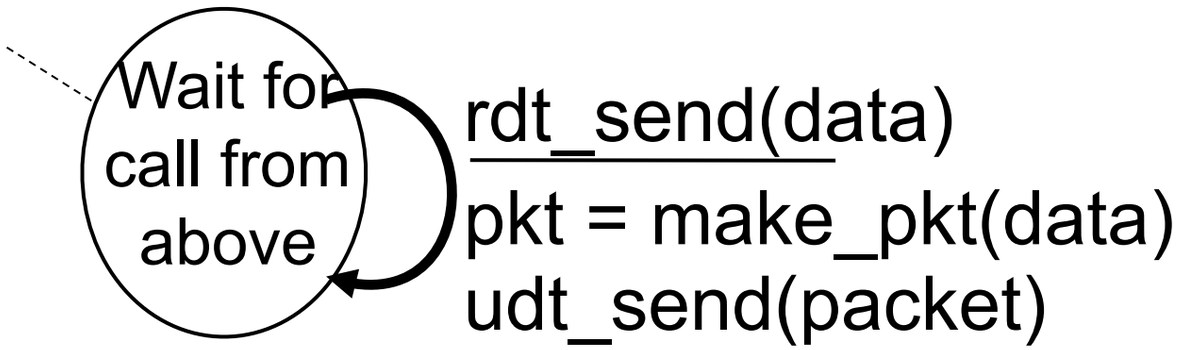
- we'll:
  - incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
  - consider only unidirectional data transfer
    - but control info will flow on both directions!
  - use finite state machines (FSM) to specify sender, receiver

state: when in this "state" next state uniquely determined by next event

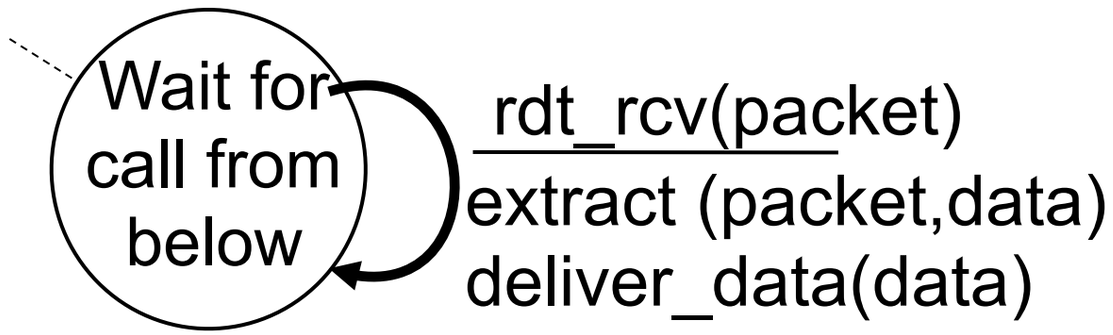


rdt1.0: Reliable Transfer over a Reliable Channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
  - ordered delivery
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel



sender



receiver

rdt2.0: Channel with Bit Errors

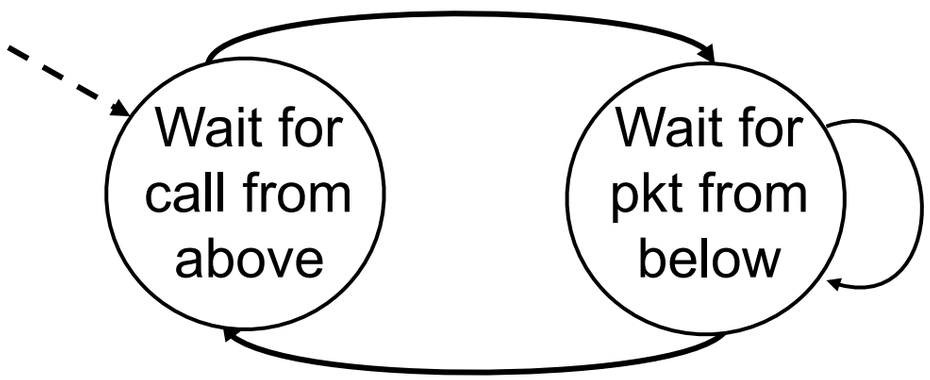
- Underlying channel may flip bits in packet
  - checksum to detect bit errors
  - but no packet loss
- Question: how to recover from errors:?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- New mechanisms in `rdt2.0` (beyond `rdt1.0`):
  - Error detection
  - Receiver feedback:
    - Control msgs (ACK,NAK) rcvr->sender

How do humans recover from "errors" in conversation?

rdt2.0: Channel with Bit Errors

- What are ACKs and NAKs?
  - Control Messages
  - allows receiver to inform sender
    - if packet is received correctly
    - if packet is corrupted and needs retransmission
- Protocols based on retransmission are called
    - ARQ (Automatic Repeat reQuests) protocols
- ARQ requires following capabilities
  - Error Detection - checksum can be used
  - Receiver feedback - ACK or NAK
  - Retransmission - done by sender
- What are state m/cs for sender and receiver?

rdt2.0: FSM Specification



rdt_send(data)
sndpkt = make_pkt(data, checksum)
udt_send(sndpkt)

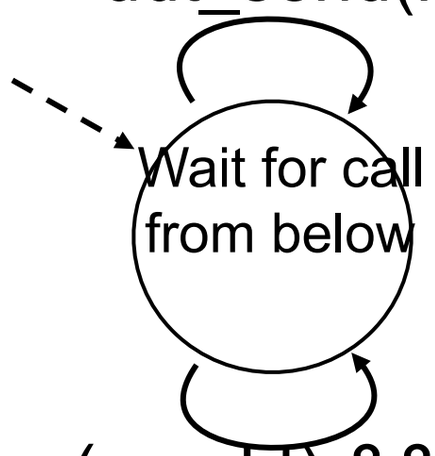
rdt_rcv(rcvpkt) && isNAK(rcvpkt)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && isACK(rcvpkt)
Λ

sender
(2 states)

can't accept packet from higher layer in 2nd state

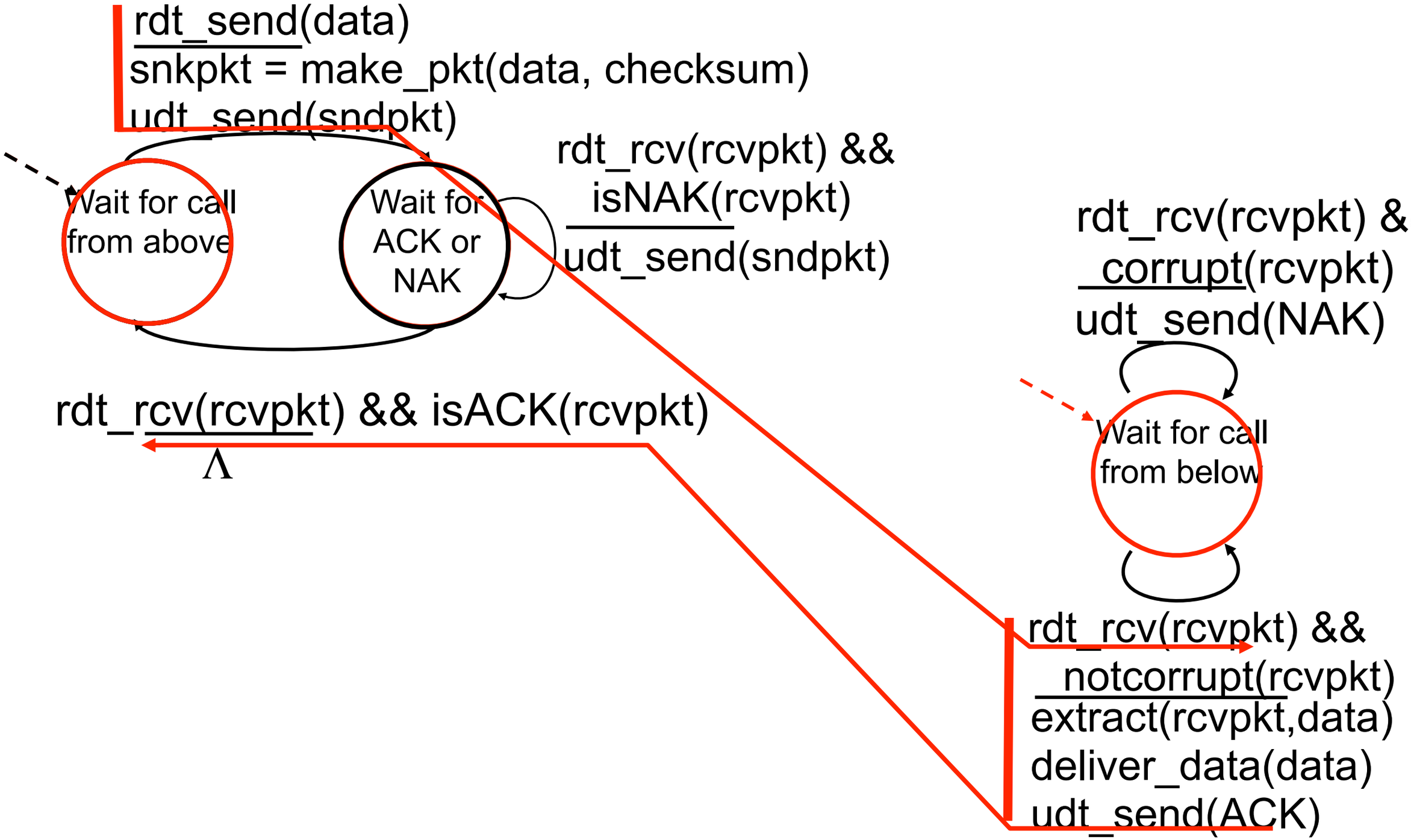
receiver
(1 state only)



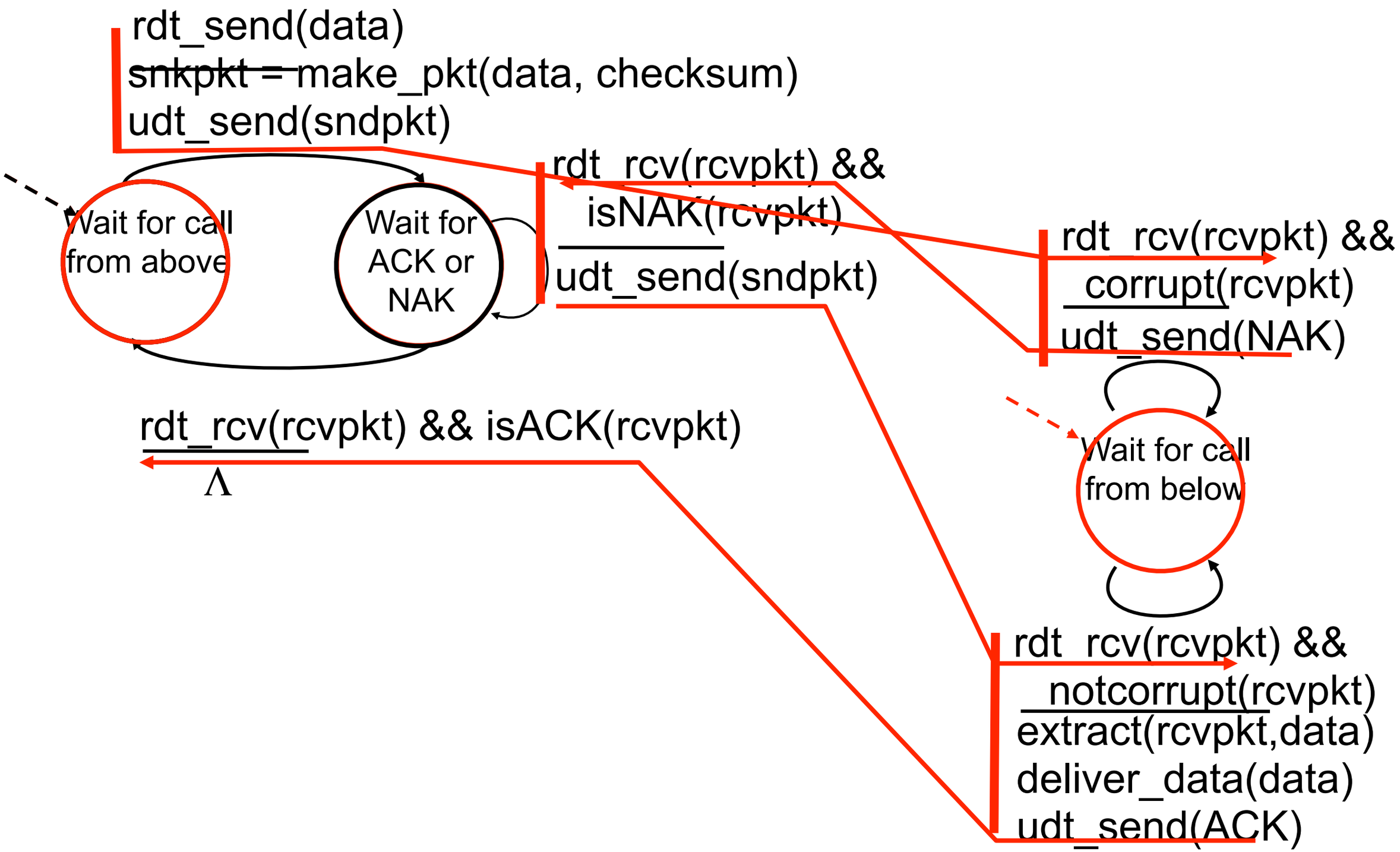
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
udt_send(NAK)

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
udt_send(ACK)

rdt2.0: Operation with No Errors



rdt2.0: Error Scenario



Protocols based on **rdt**2.0

- Known as **Stop-and-Wait**
  - why?
    - Sender does not send a new packet till
    - It is sure previous packet is received
      - Correctly by receiver
      - it does not even get a packet from upper layers
- Are they any issues with rdt2.0?
  - what can go wrong?

rdt2.0 has a Fatal Flaw!

- what happens if ACK/ NAK corrupted?
- sender doesn't know what happened at receiver!
- two possibilities to handle
  - add more checksum bits to recover from error
  - can we just retransmit: possible duplicate pkts?
- handling duplicates:
- sender retransmits current pkt if ACK/NAK corrupted
- sender adds *sequence number* to each pkt
  - Is `1` bit seq num ok?
- receiver discards (doesn't deliver up) duplicate pkt
- does ACK/NAK require seq number?

stop and wait

sender sends one packet,
then waits for receiver response

Summary

- Application layer
- Tranport layer channel
- RDT 1.0
- RDT 2.0
- Requirements for RDT 2.1
  - Handle error for NAK and ACK