CS 348

Computer Networks

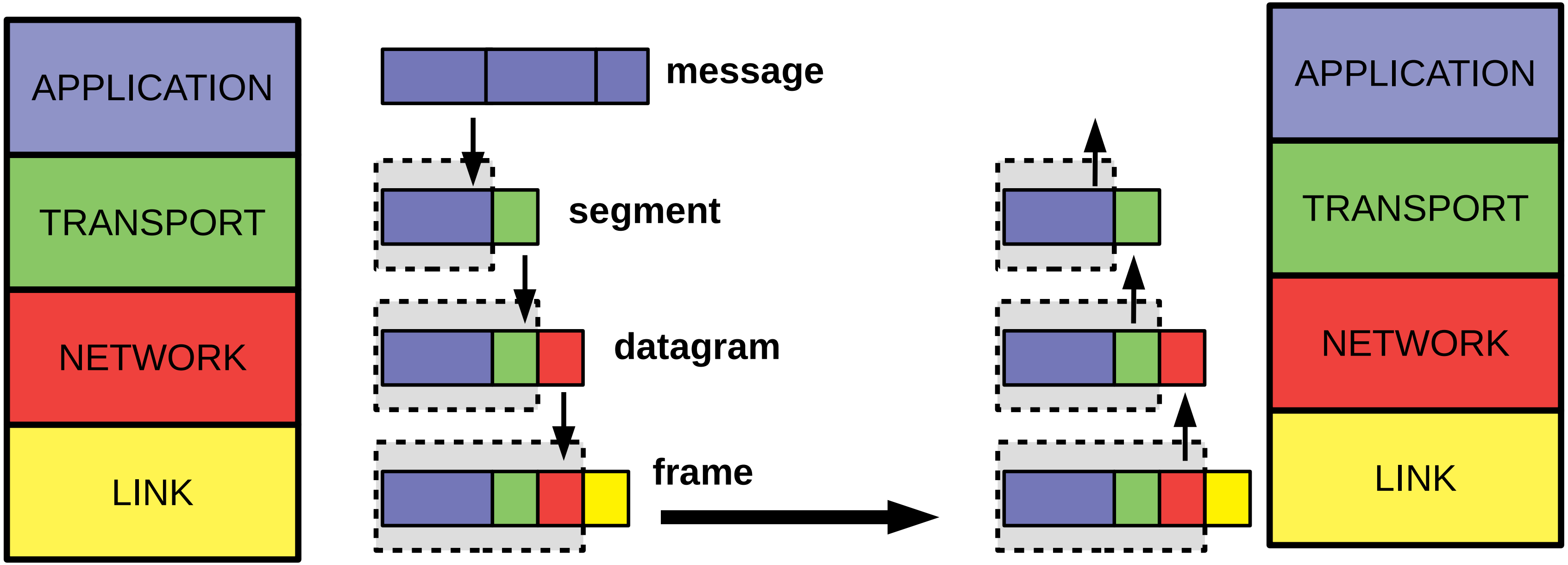
Lec 15
Principles of Reliable Data Transfer

Spring 2020 IIT Goa

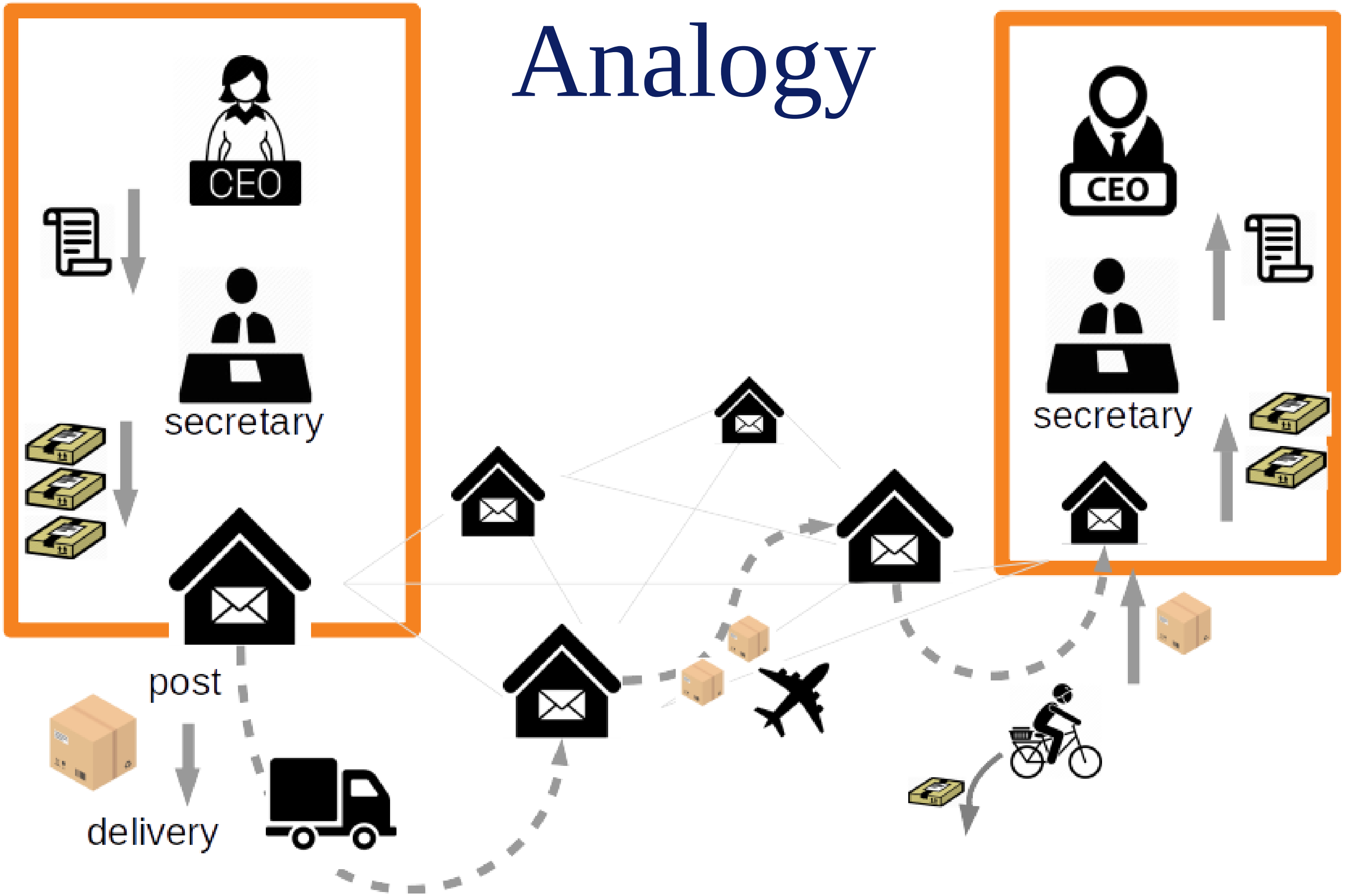
Course Instructor: Dr. Neha Karanjkar

Note: These slides are adapted from “Computer Networking: A Top-down Approach” by Kurose & Ross, 7^{th} ed

Recall: The TCP/IP Model



A Postal Analogy



Recall: Service Models

- **Network Layer:** Best-effort host-to-host delivery of a single datagram

  NW layer: “Give me a payload and a destination IP address. I will create a single datagram containing the payload and make the best effort to deliver it to the destination, but can offer no guarantee of delivery (the packet can get corrupted or lost).”

- **Transport Layer:** Process-to-process message delivery service for applications, which can be reliable and in-order (TCP) or best-effort (UDP)

  TCP: “Give me a message (as a sequence of Bytes) and a <dest IP, dest port> address, and I will split it into segments and deliver them reliably and in-order, to the correct process (socket) at the destination host.”

  UDP: “Give me a message and a <dest IP, dest port> address, and I will make a best-effort delivery of that message as a single packet to the correct process (socket) at the destination.

Reliable Data Transfer

- How can reliable data transfer be possible using the unreliable delivery service provided by the Network layer?

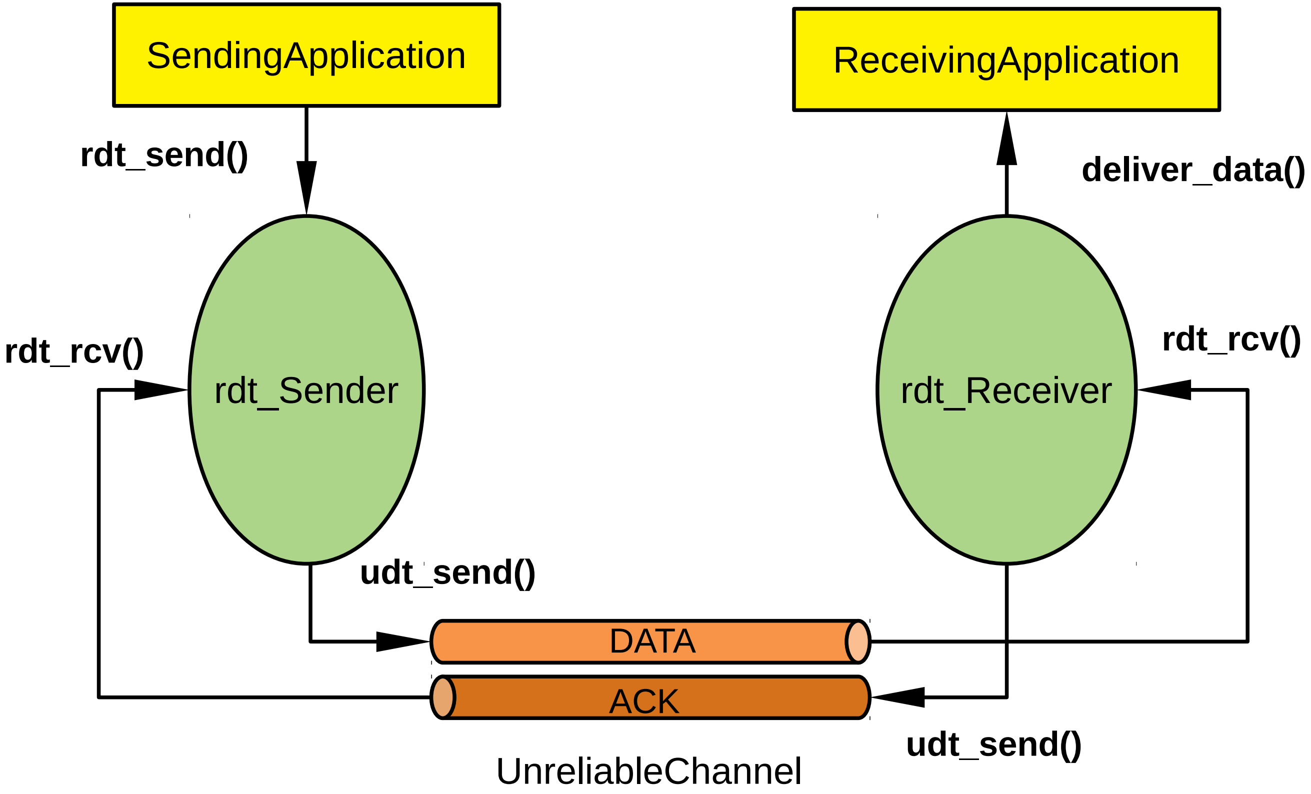
Reliable Data Transfer

- How can reliable data transfer be possible using the unreliable delivery service provided by the Network layer?
  - using acknowledgements, sequence numbers, timers etc

Reliable Data Transfer

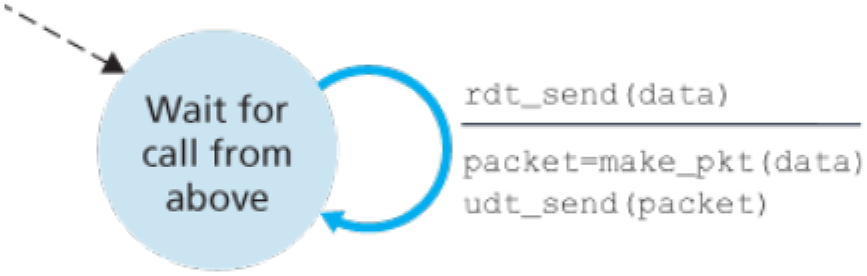
- Some simplfying **assumptions** we make (to being with)
  - One side is the sender and the other is the receiver
  - The sender sends “data”, the receiver sends back “acknowledgements”
  - There’s a lossy, unreliable channel
    - A packet can get corrupted (with probability P_corruption)
    - A packet can get lost (with probability P_loss)
    - Packets are delivered **in-order** (if at all). Packets don’t get re-ordered.

A Template

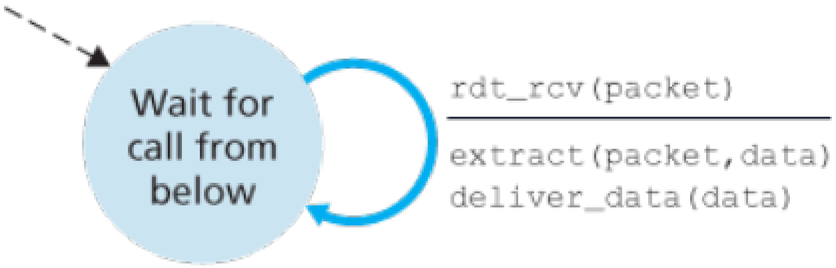

SendingApplication
ReceivingApplication
rdt_send()
deliver_data()
rdt_rcv()
rdt_Sender
rdt_Receiver
rdt_rcv()
udt_send()
DATA
ACK
udt_send()
UnreliableChannel

rdt1.0 A perfectly reliable channel

(P_corruption=0, P_loss=0)



rdt1.0: sending side



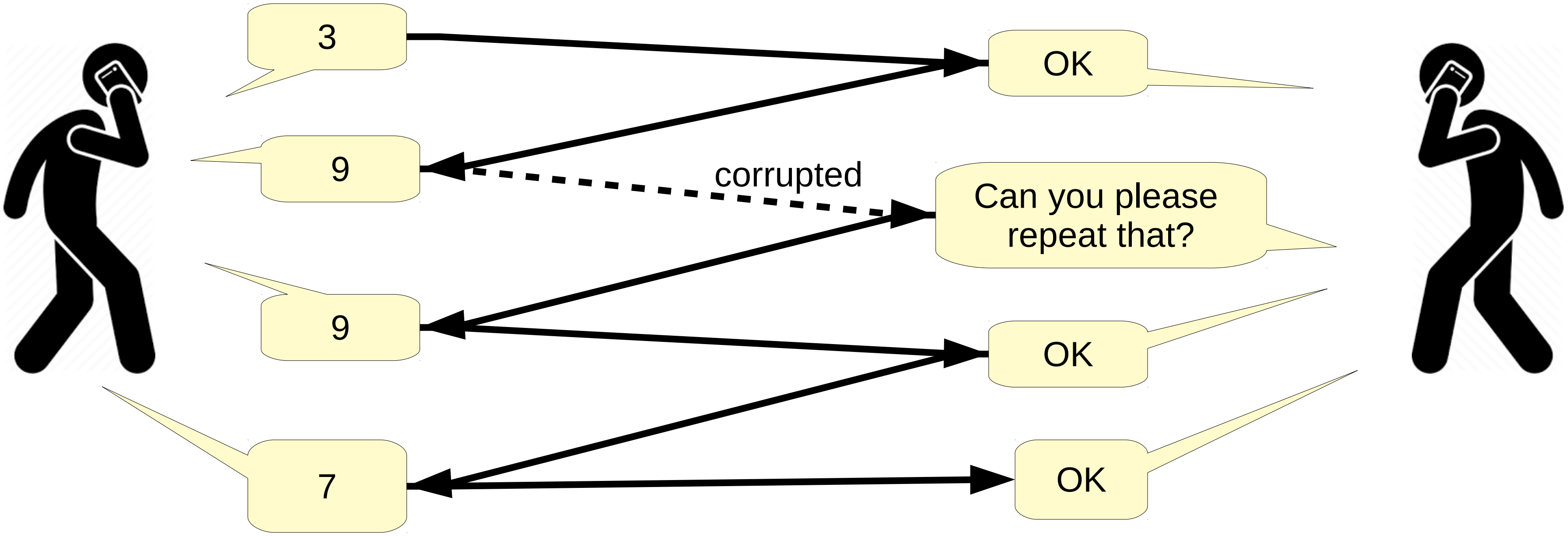
rdt1.0: receiving side

rdt2.0 Bit errors are possible

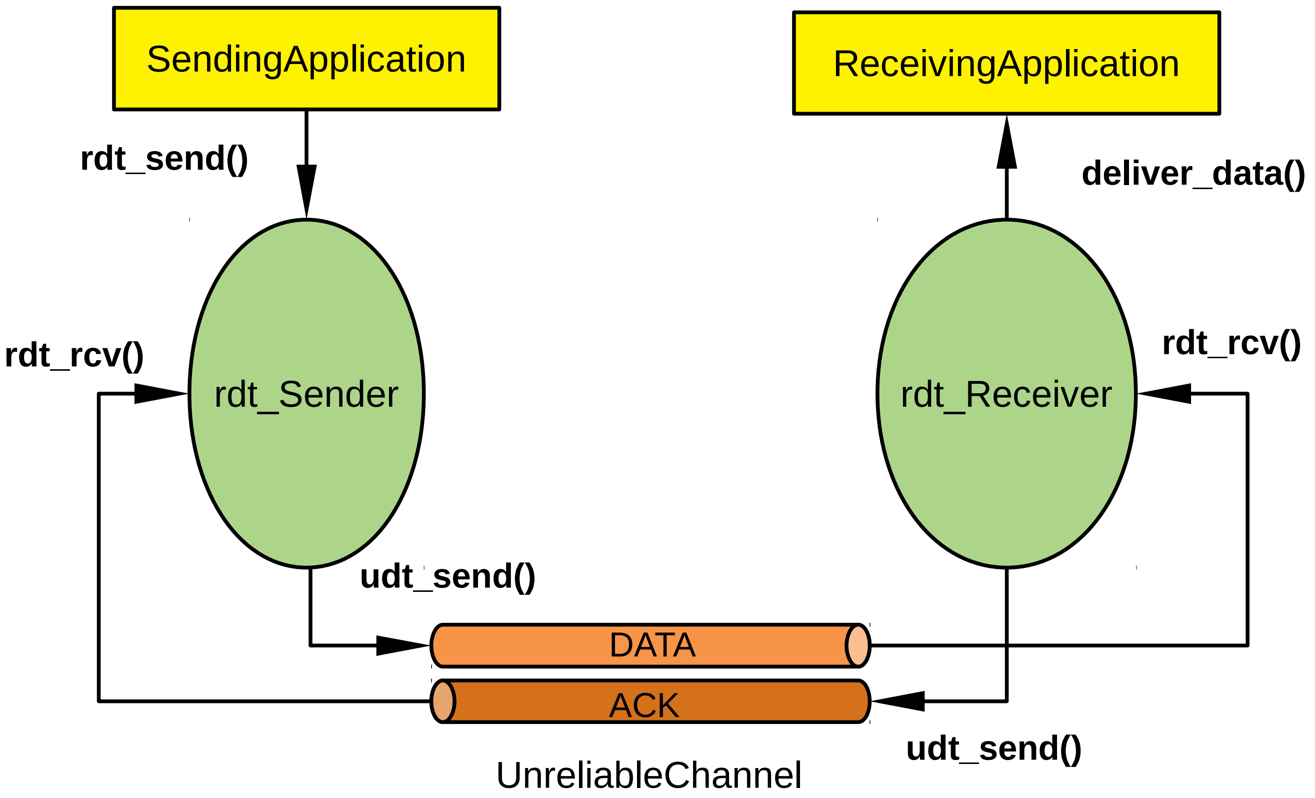
(P_corruption≠0, P_loss=0)

- The receiver needs:
  - Some mechanism to find out if the packet was corrupted
  - Send ACK/NAK
- The sender should re-transmit upon receiving a NAK

Analogy: person A is dictating a phone number to person B over a noisy phone connection

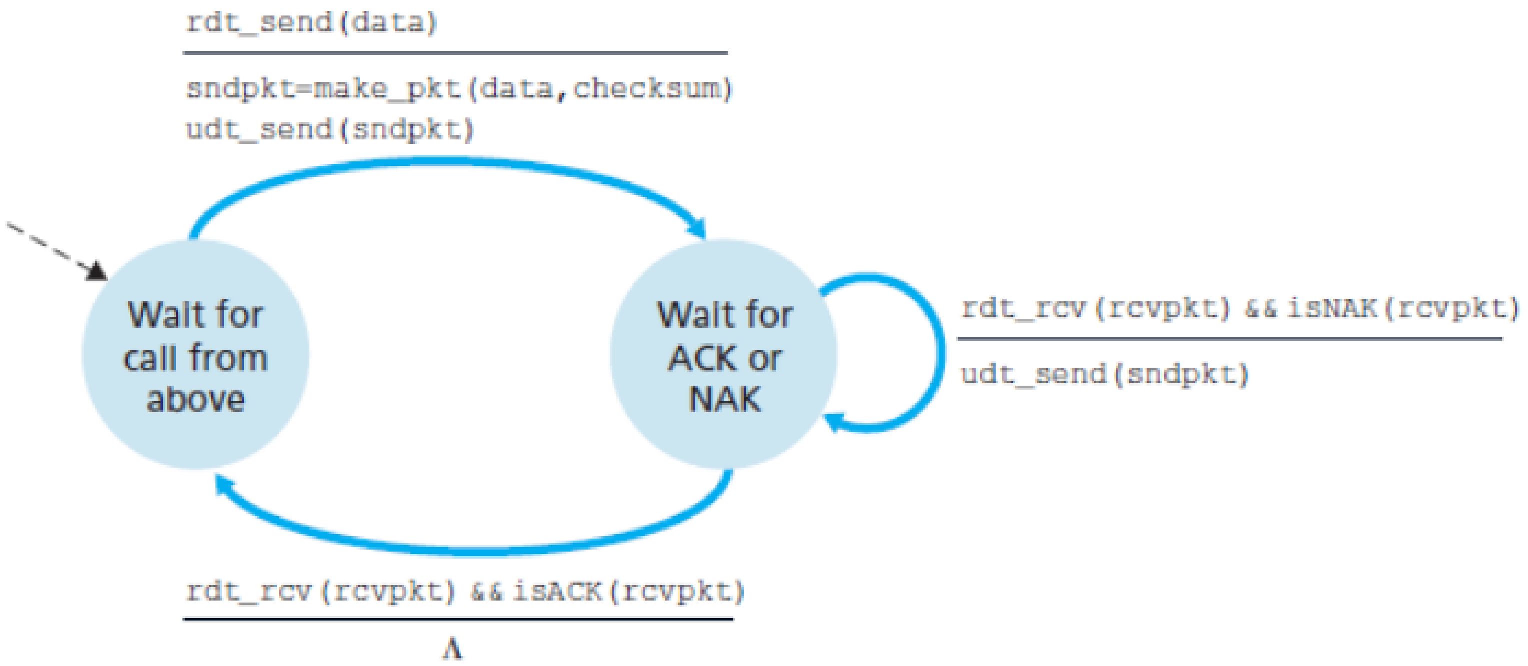


A Template


SendingApplication
ReceivingApplication
rdt_send()
deliver_data()
rdt_rcv()
rdt_Sender
rdt_Receiver
rdt_rcv()
udt_send()
DATA
ACK
udt_send()
UnreliableChannel

rdt2.0 Bit errors are possible

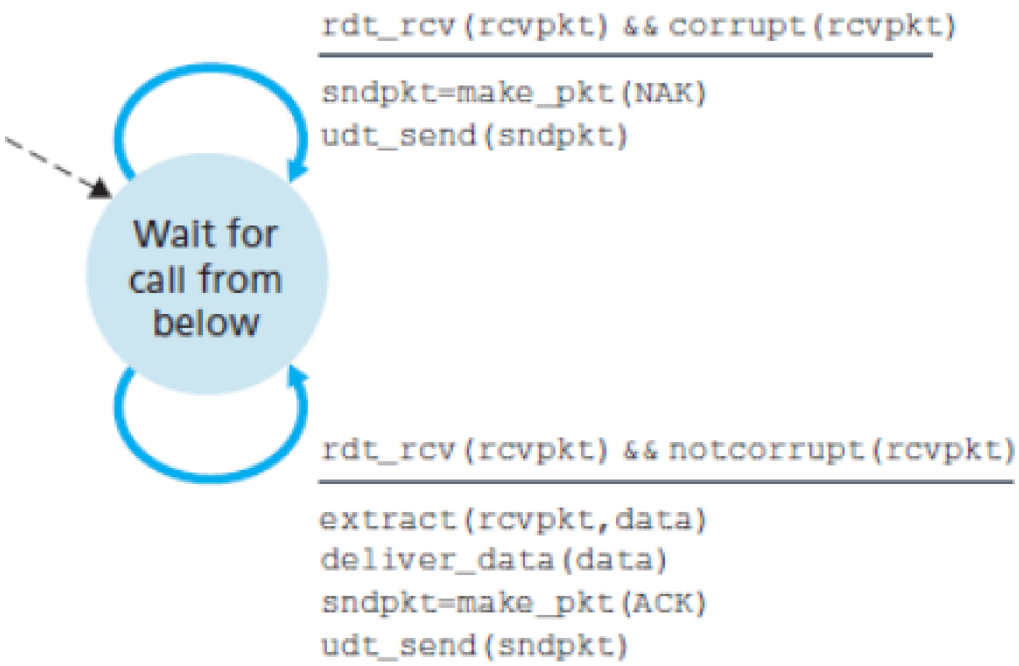
(P_corruption≠0, P_loss=0)



rdt2.0: sending side

rdt2.0 Bit errors are possible

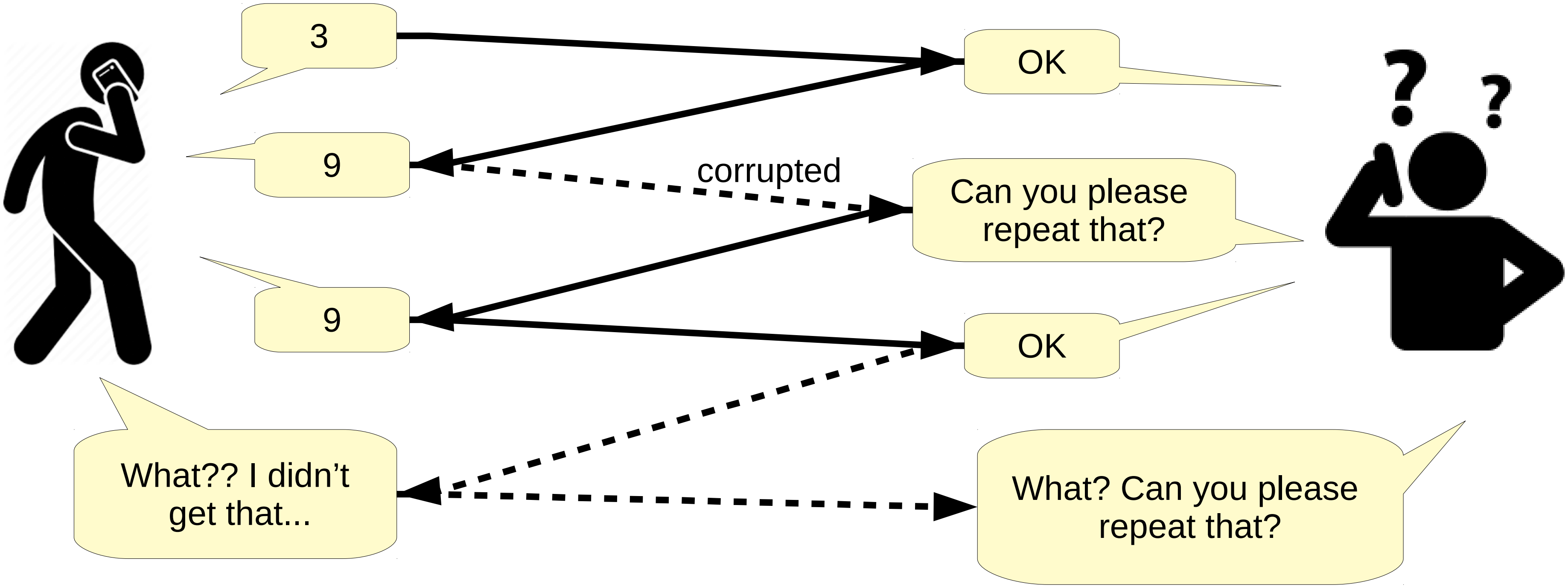
(P_corruption≠0, P_loss=0)



rdt2.0: receiving side

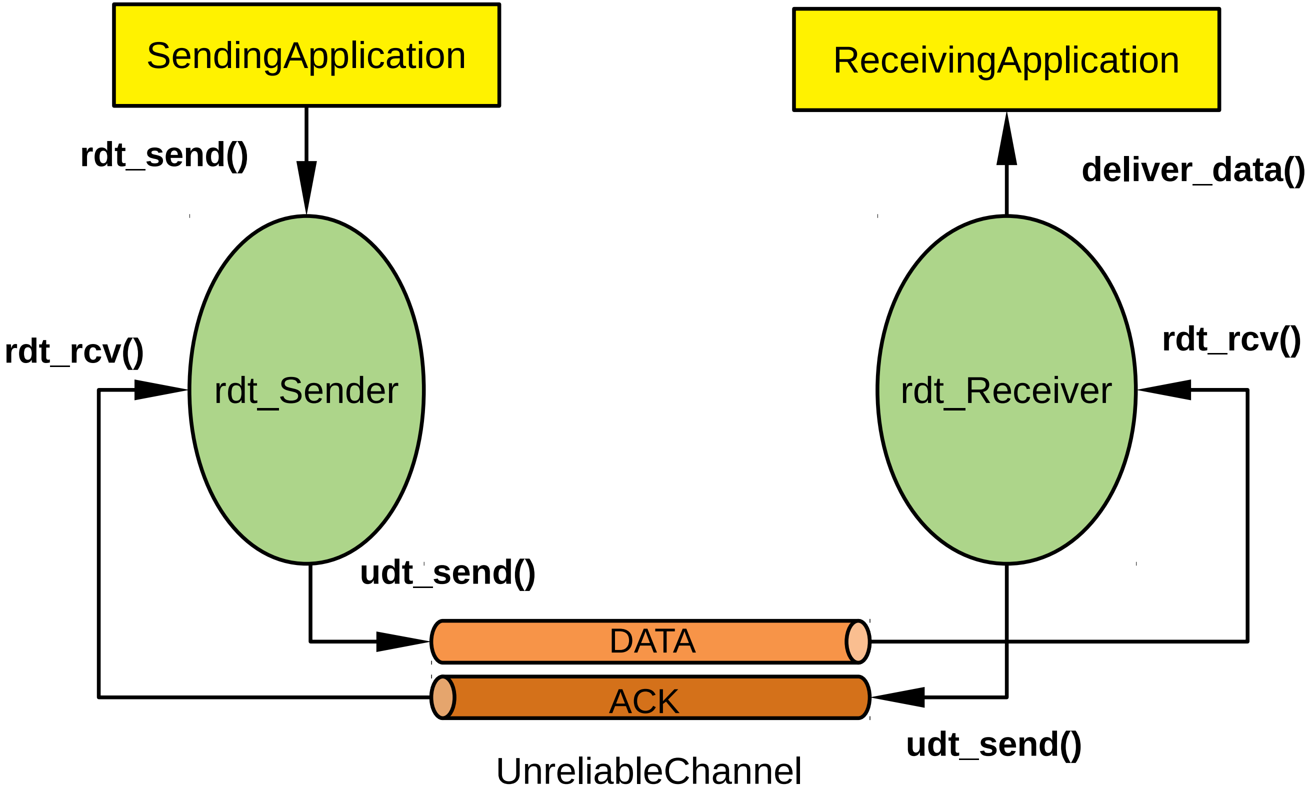
rdt2.1 Bit errors are possible (even in the ACK/NAK packets)

Analogy: person A is dictating a phone number to person B over a noisy phone connection



The ACK/NACK can also get corrupted!

A Template


SendingApplication
ReceivingApplication
rdt_send()
deliver_data()
rdt_rcv()
rdt_Sender
rdt_Receiver
rdt_rcv()
udt_send()
DATA
ACK
udt_send()
UnreliableChannel

rdt2.1 sender

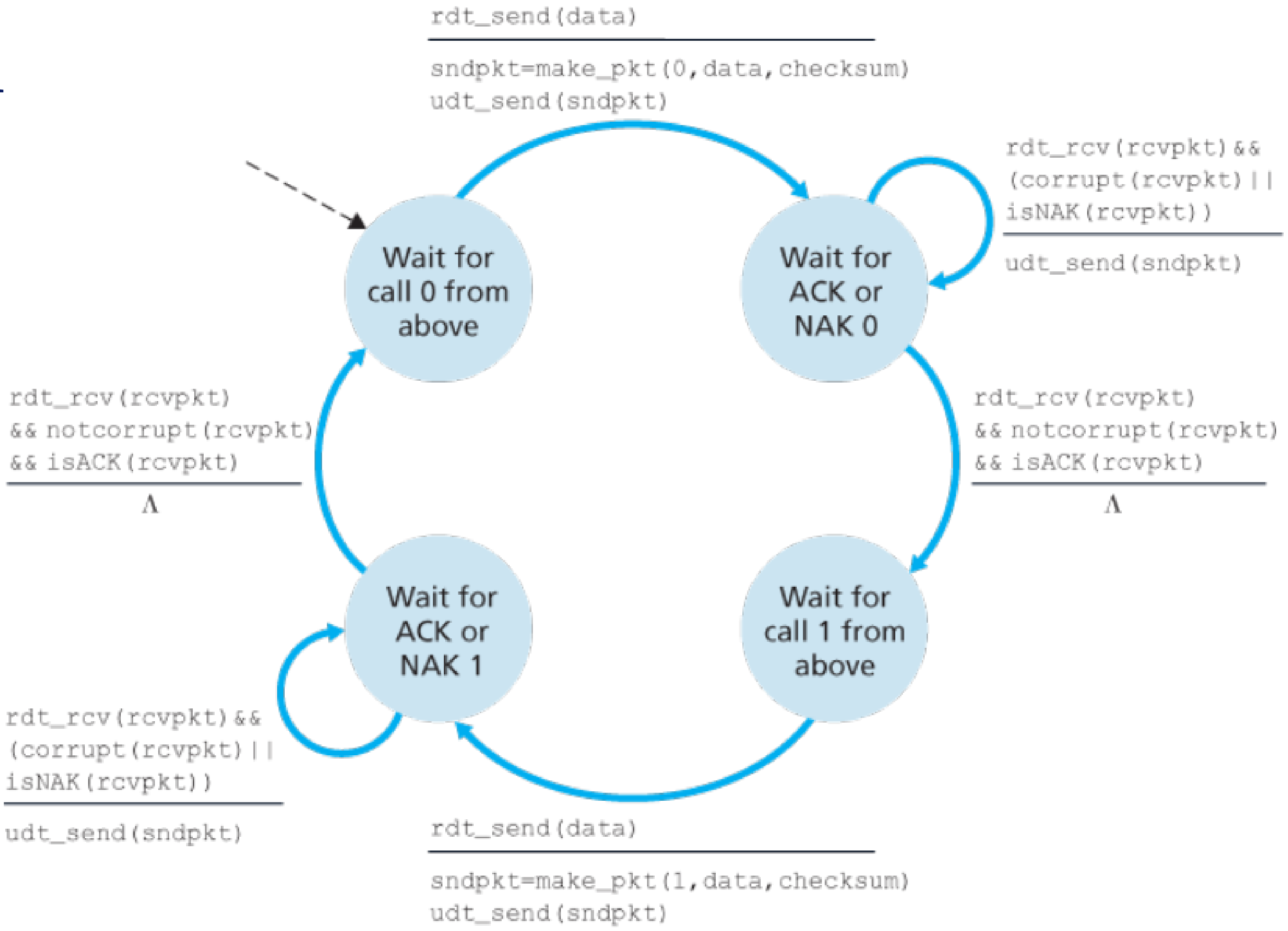


Figure 3.11 *rdt2.1* **sender**

rdt2.1 receiver

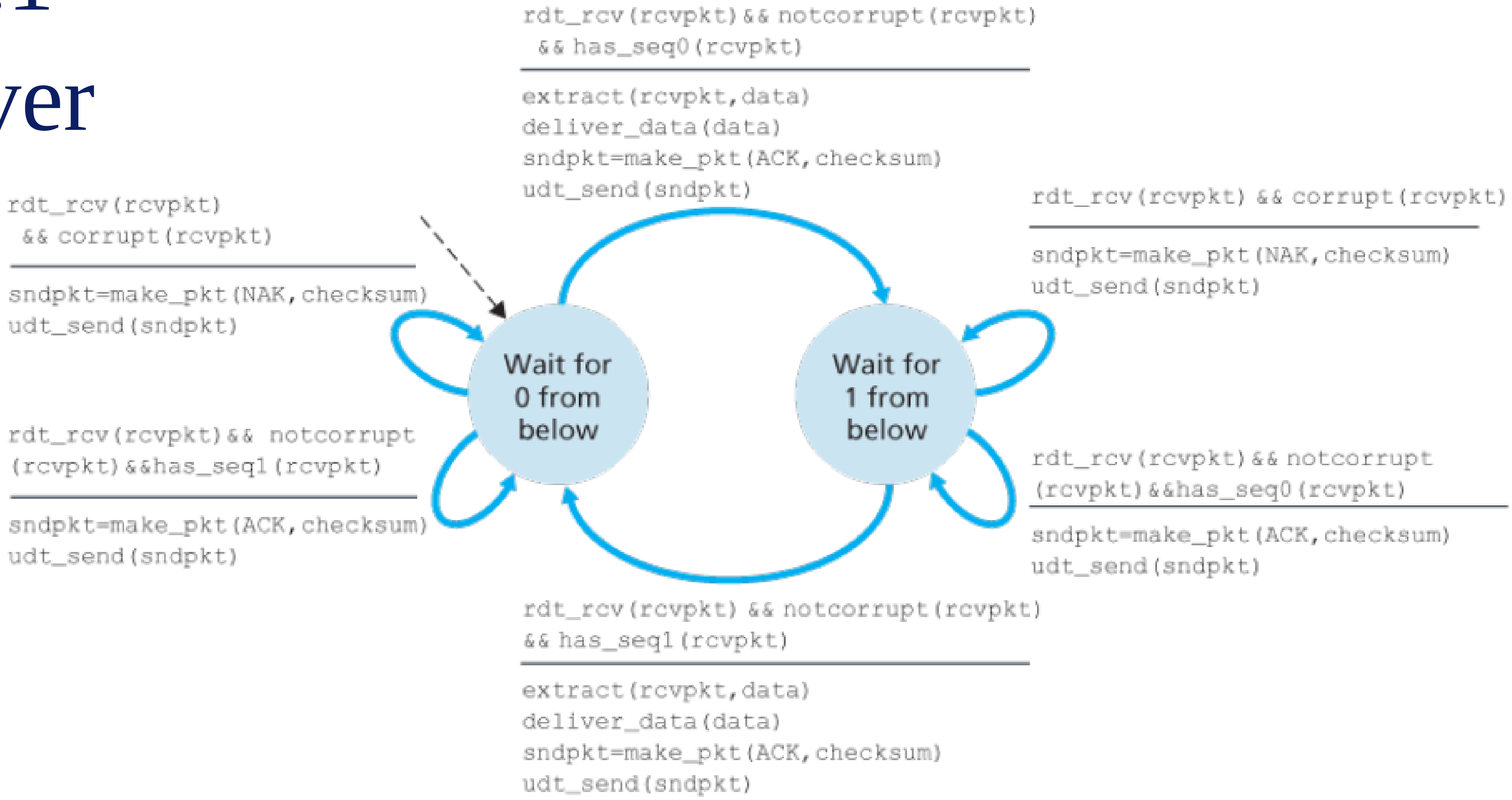


Figure 3.12 *rdt2.1* **receiver**

rdt2.2: We can use ACK 0 /ACK 1 to acknowledge the last successfully received packet. No need of a NAK.

rdt2.2 sender

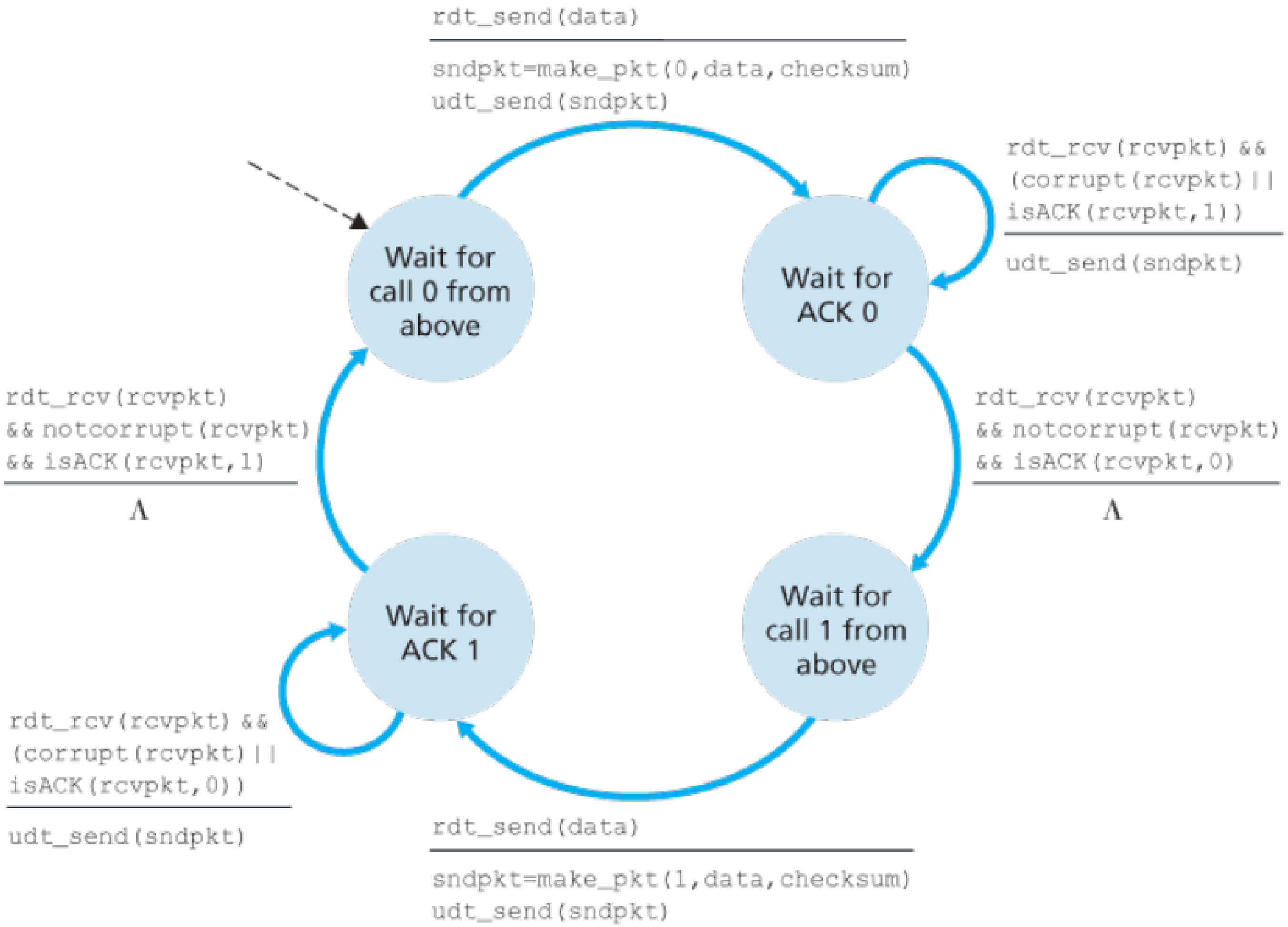


Figure 3.13 `rdt2.2` **sender**

rdt2.2 receiver

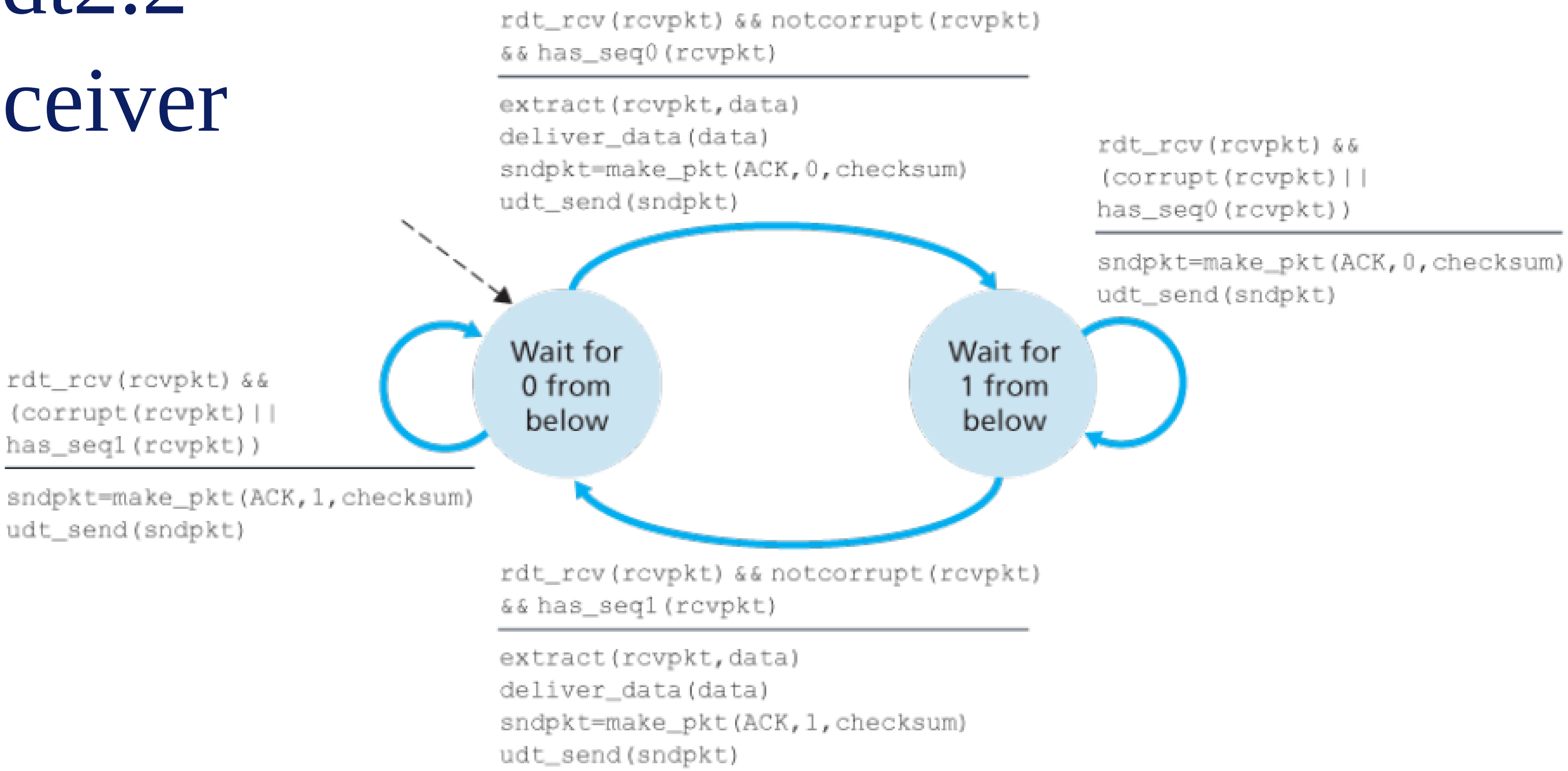
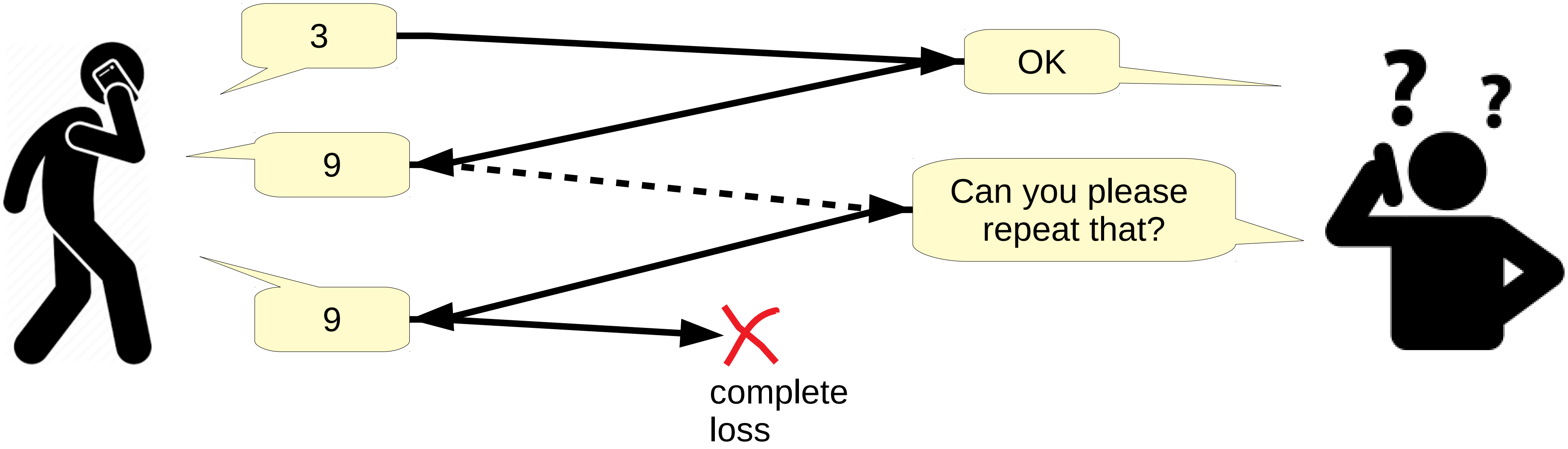


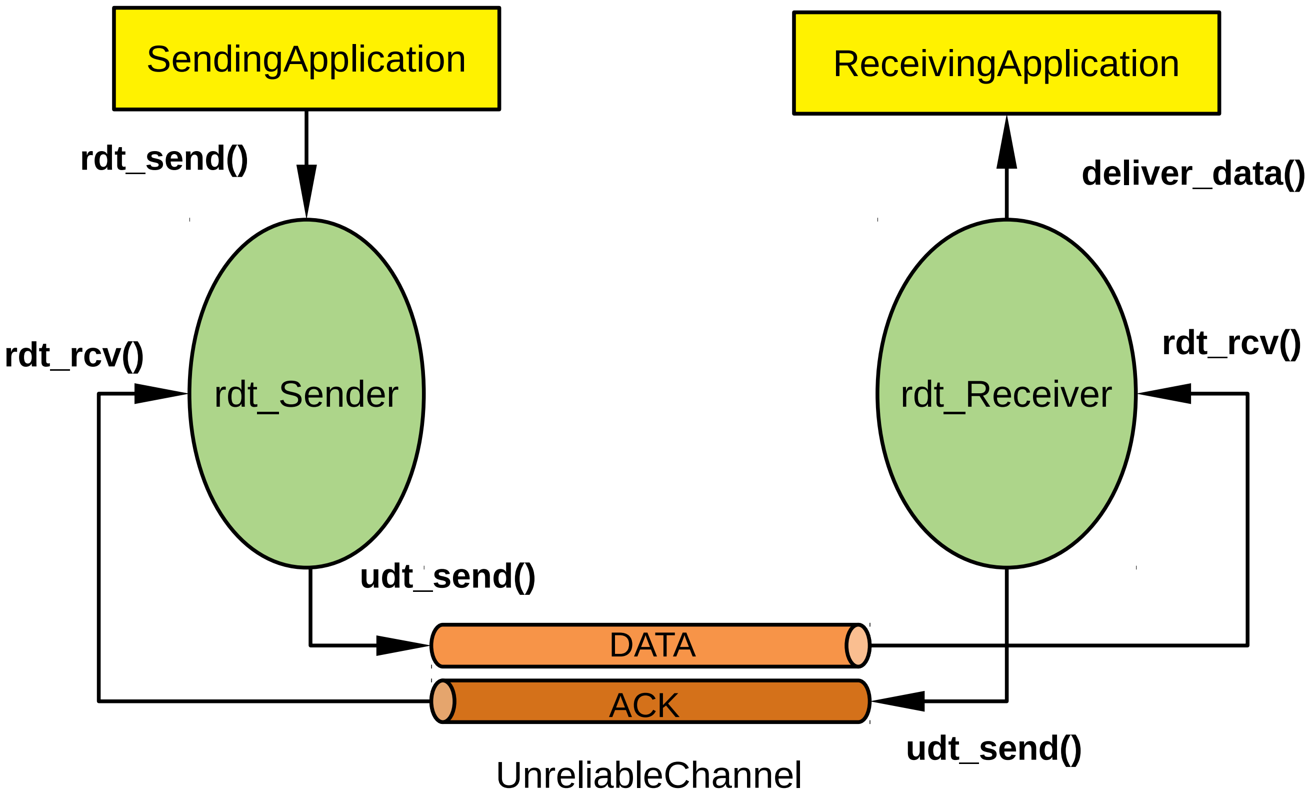
Figure 3.14 `rdt2.2` **receiver**

rdt3.0 Bit errors are possible and packet loss is also possible

(P_corruption≠0, P_loss≠0)


3
OK
9
Can you please repeat that?
9
complete loss

A Template


SendingApplication
ReceivingApplication
rdt_send()
deliver_data()
rdt_rcv()
rdt_Sender
rdt_Receiver
rdt_rcv()
udt_send()
DATA
ACK
udt_send()
UnreliableChannel

rdt3.0 sender

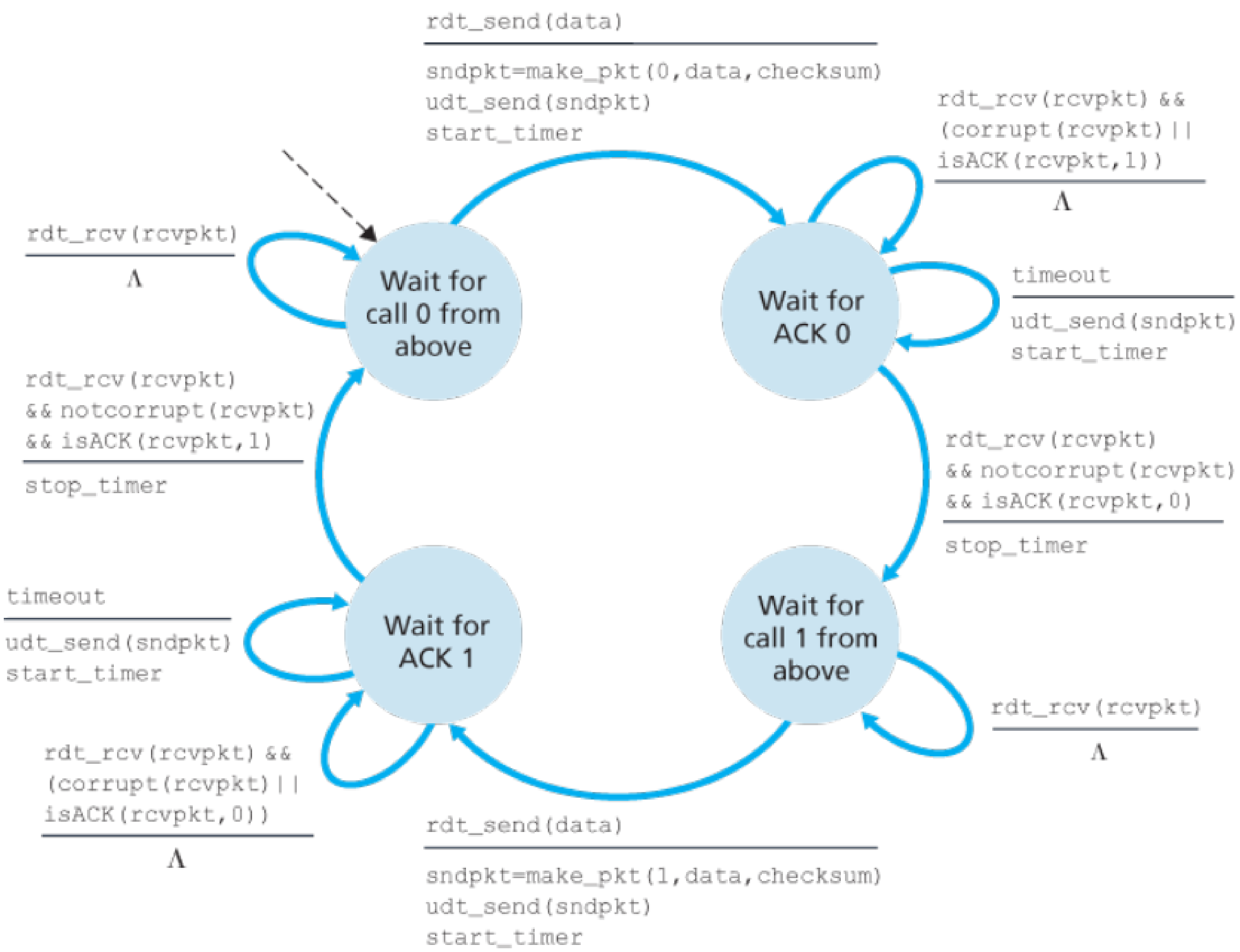


Figure 3.15 *rdt3.0* **sender**

rdt3.0 receiver ??

Exercise

- **[LAB] Design, simulate and test these protocols**

  Using Python+SimPy

  https://nehakaranjkar.github.io/ProtocolSimulation.html

References and Reading Assignment

- **Kurose and Ross 6th ed:** Section 3.4

So far...

- Structure and Physical components of the Internet
- Design of the Internet: Layering and Encapsulation
- The Applications Layer:
  - Sockets Interface
  - The Web and HTTP
  - DNS
- **The Transport Layer: how it works**
  - **Basic services, UDP**
  - **Principles of Reliable Data Transfer**
  - **Pipelined data transfer (Sliding window protocols)**
  - **TCP details**
  - **Congestion and Flow control**