CSCI 466: Networks

Reliable Data Transfer (RDT)

Reese Pearsall

Fall 2024

*All images are stolen from the internet

PA1 Due **Sunday**

NO CLASS ON FRIDAY

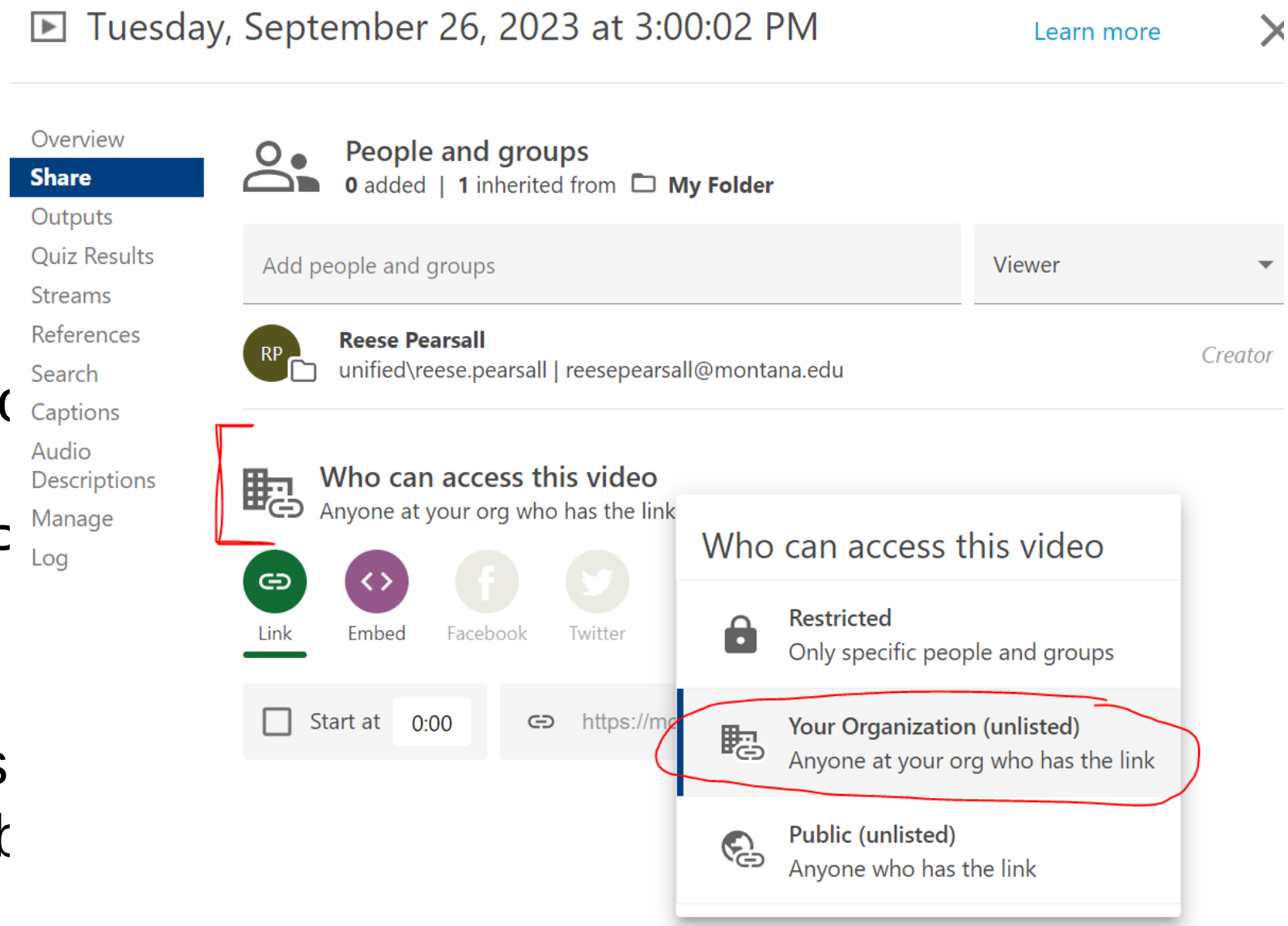
Quiz 2 on Friday

- DNS
- SMTP
- FTP
- P2P
- CDNs/Caches
- Transport Layer
- Reliable Data Transfer

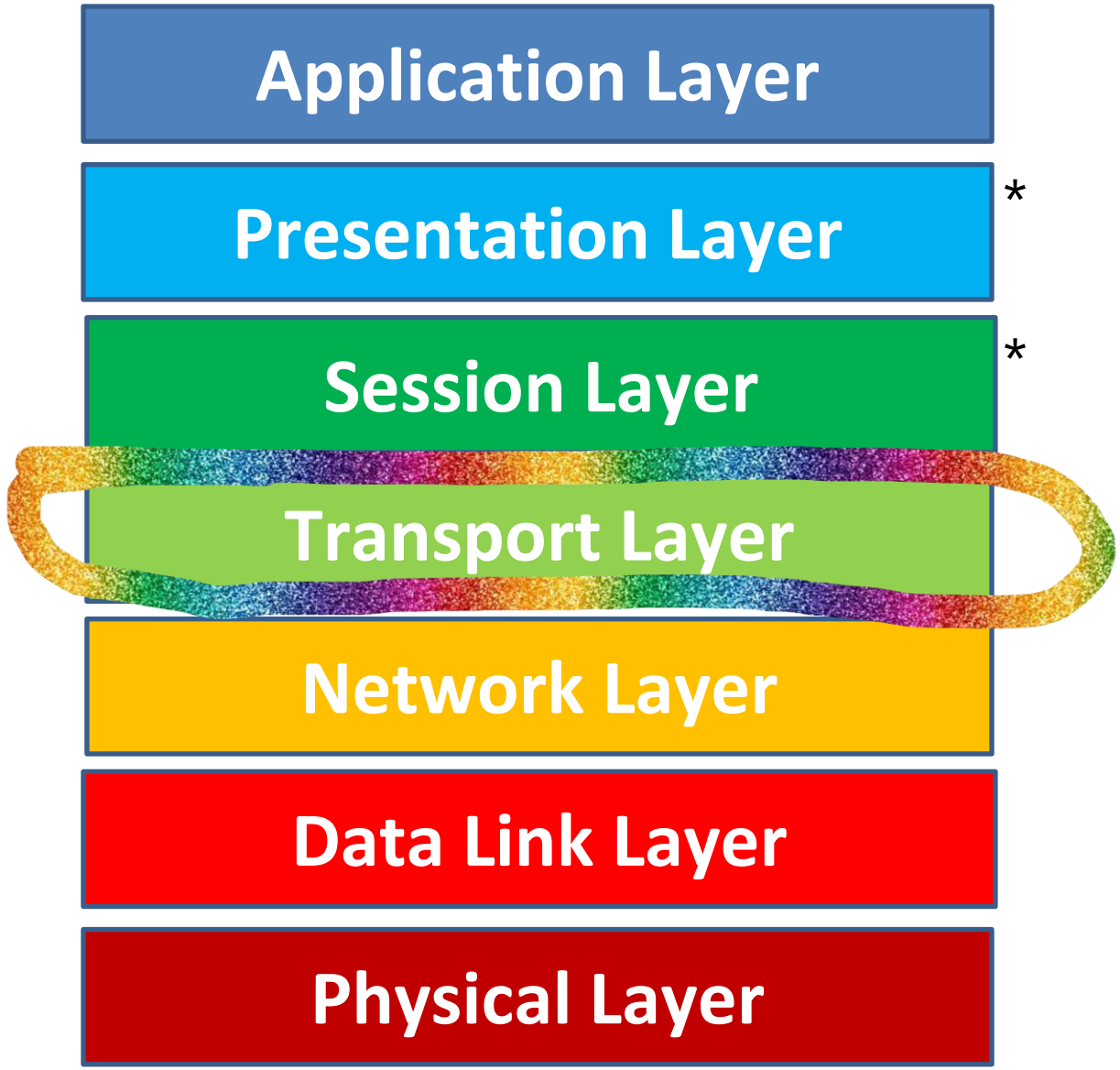
Announcements

Quiz 3 (

- DNS
- SMTF
- DTP
- P2P
- Trans
- Reliab



OSI Model



Application Layer

Messages from Network Applications

Physical Layer

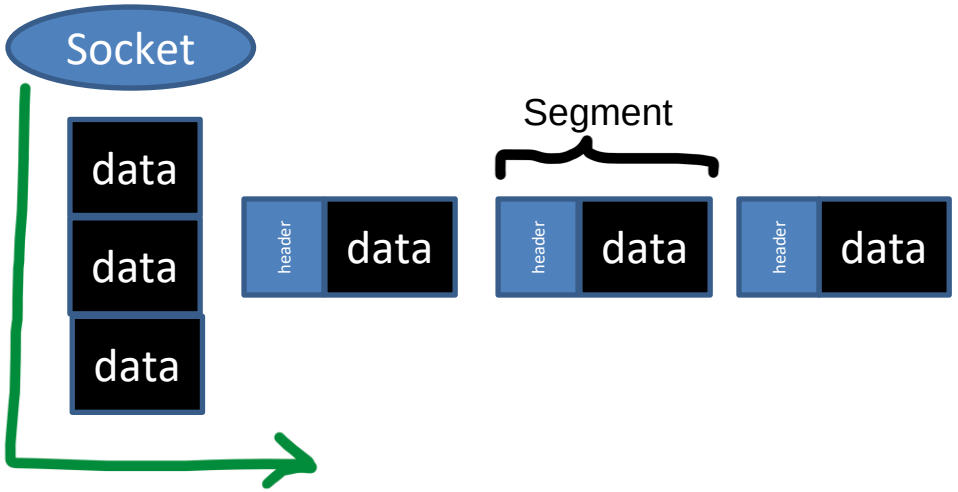
Bits being transmitted over some medium

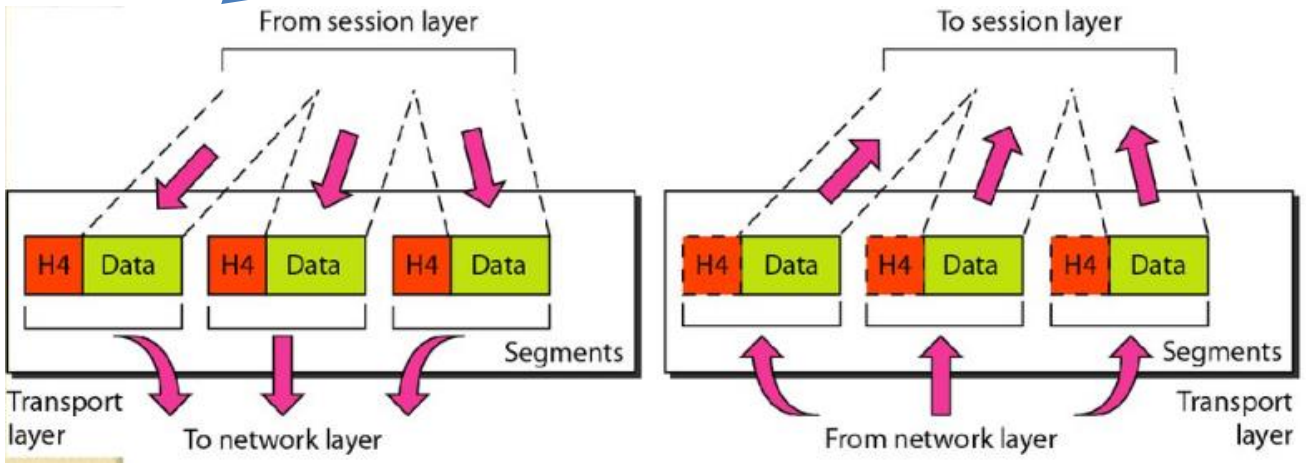
**In the textbook, they condense it to a 5-layer model, but 7 layers is what is most used*

Transport Layer

Multiplexing is the process of gathering chunks from sockets, encapsulating chunks with header information, and passing the segment into the network layer

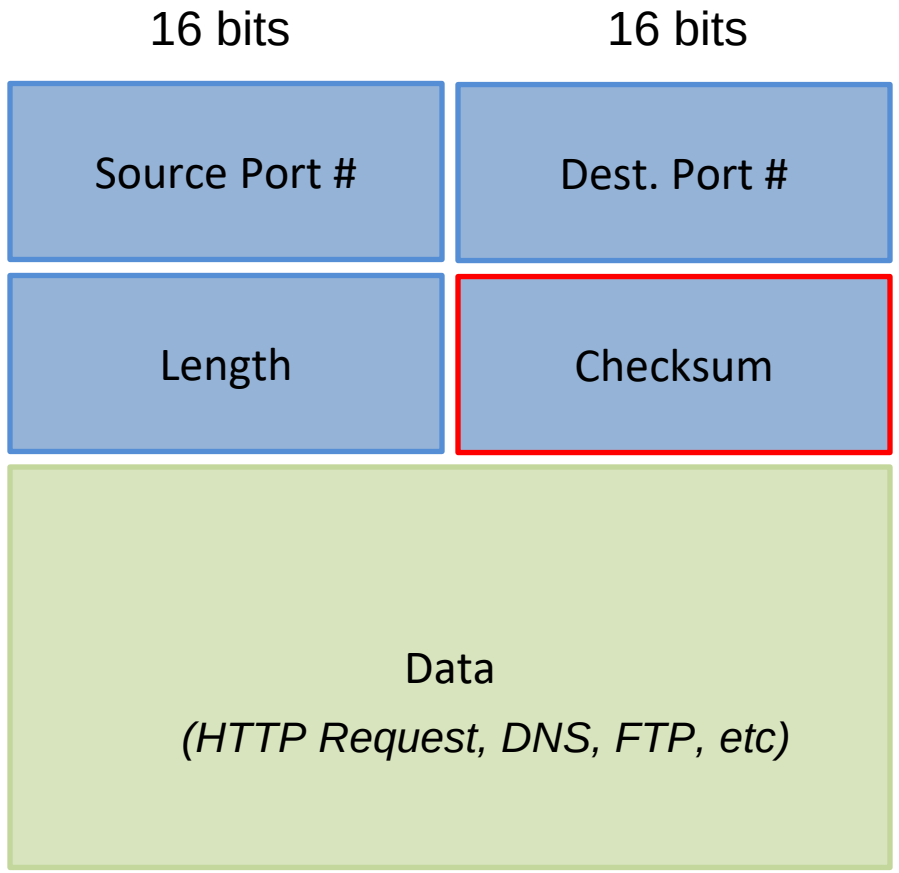
Demultiplexing is the receiving segments from the transport layer and delivering the segment to the correct socket.



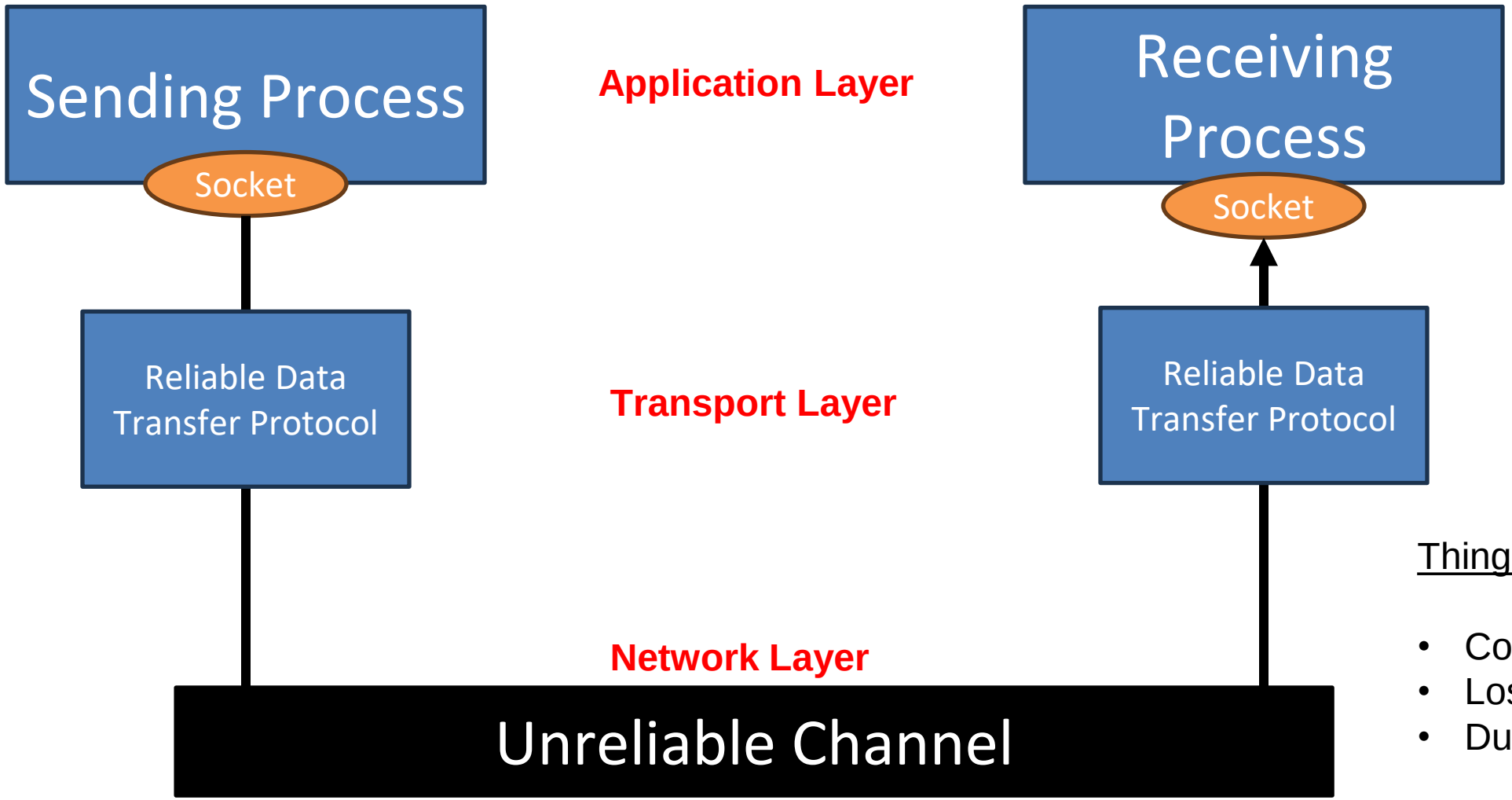


Transport Layer

Transport Layer provides a **checksum** that is used to determine whether bits within a segment have been altered



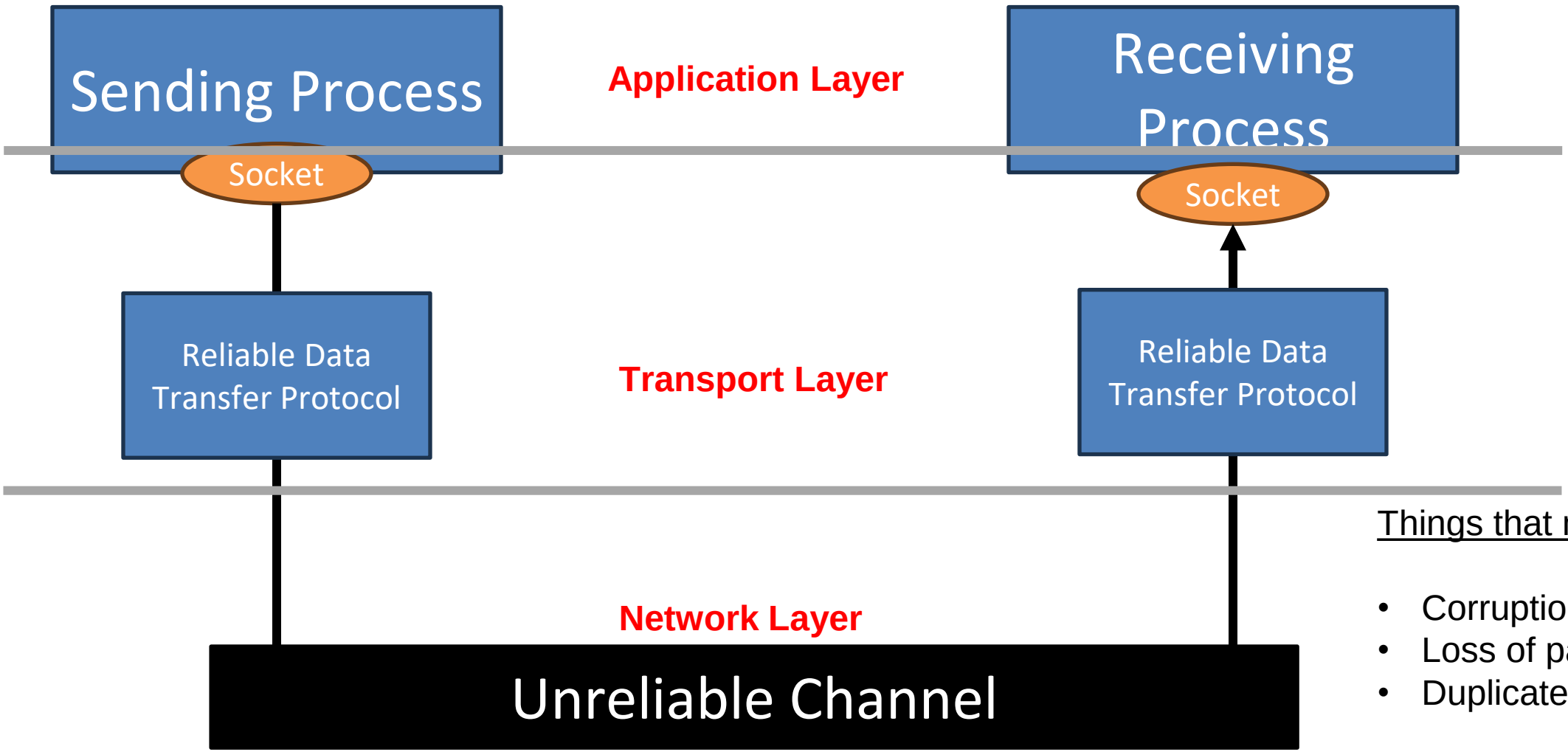
Transport Layer



Things that may happen

- Corruption
- Loss of packets
- Duplicate Delivery

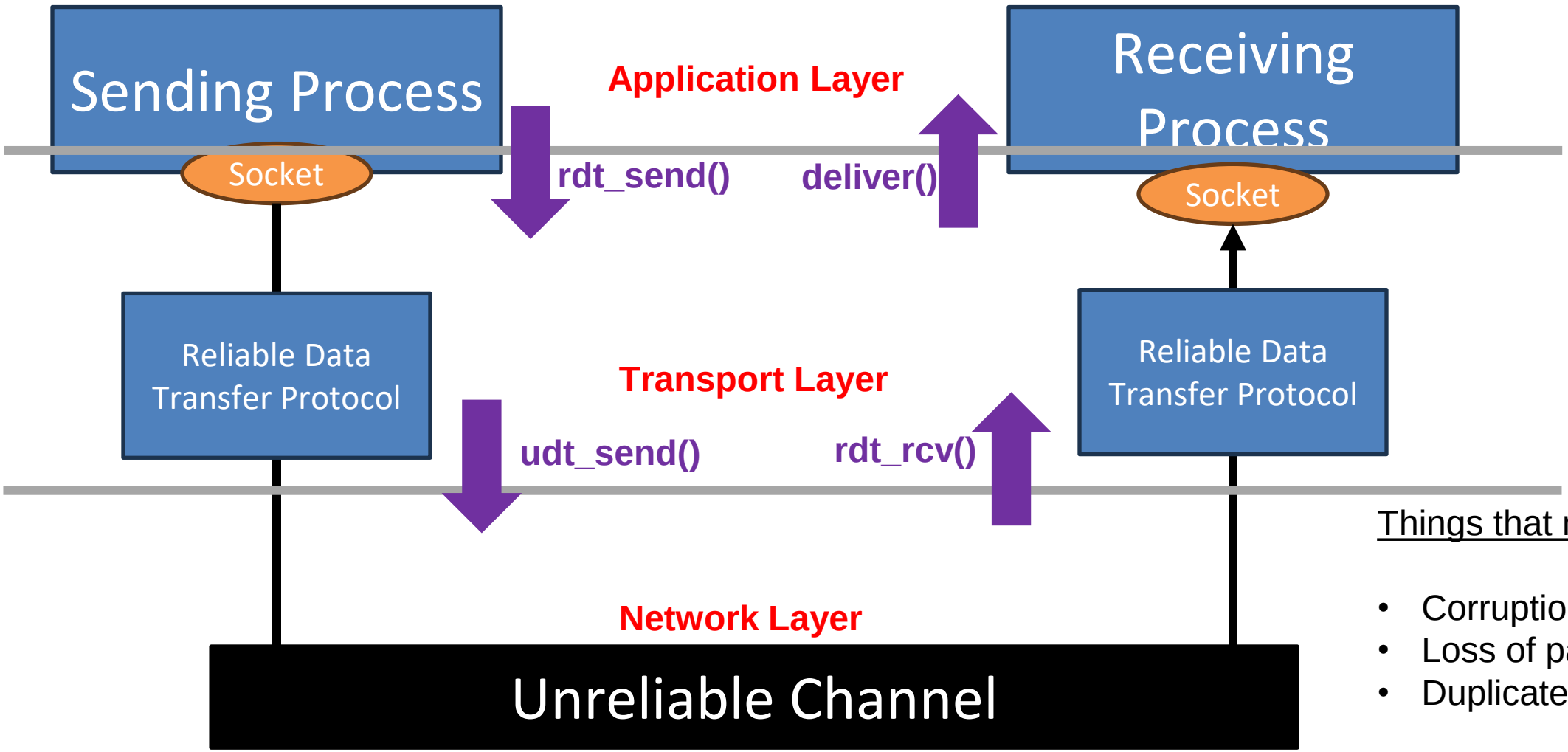
Transport Layer



Things that may happen

- Corruption
- Loss of packets
- Duplicate Delivery

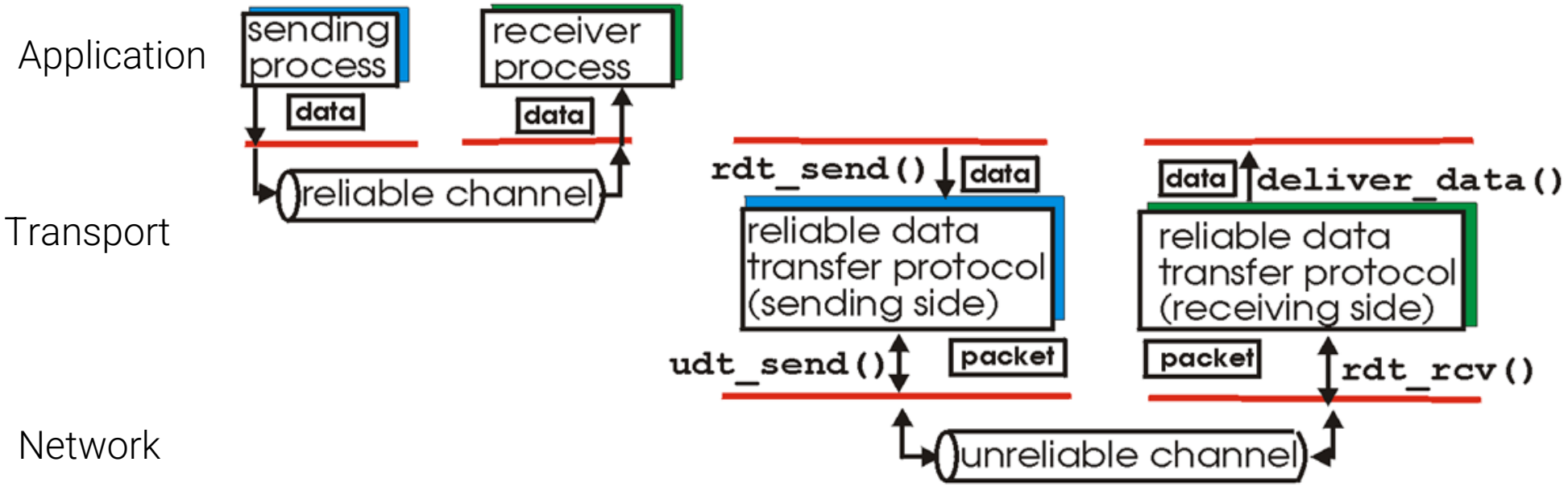
Transport Layer



Things that may happen

- Corruption
- Loss of packets
- Duplicate Delivery

Reliable Data Transfer



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

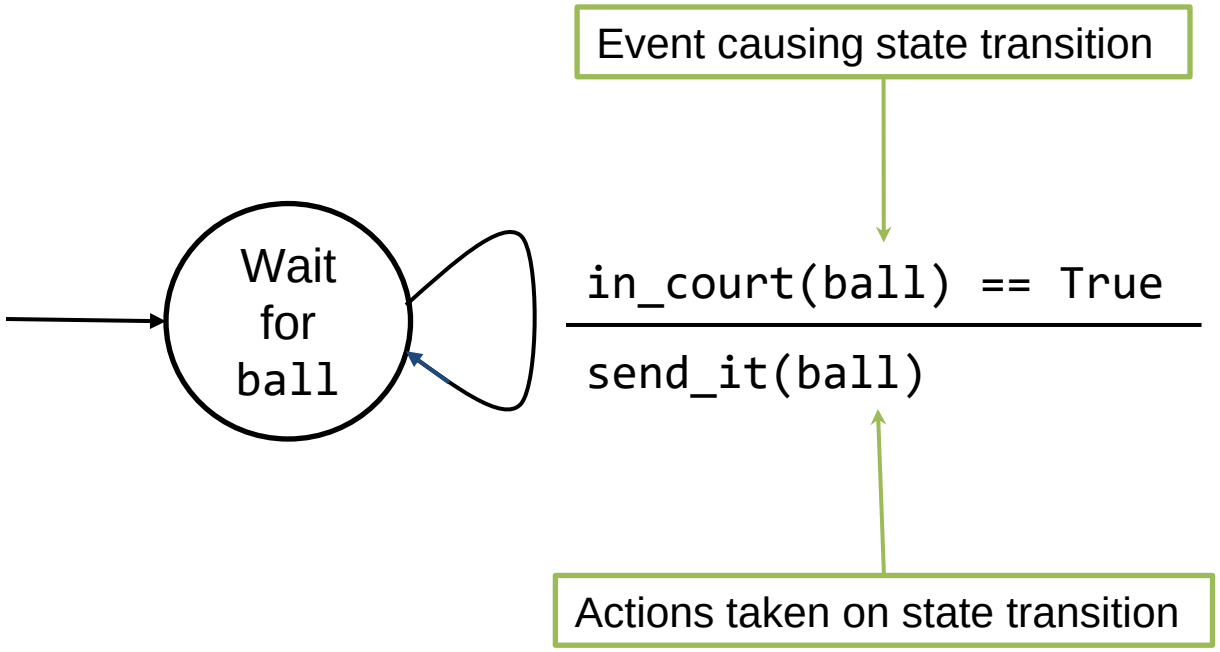
What are some ways in which the network channel can be unreliable?

Things to consider:
Corruption
Loss of packets
Duplicate delivery

Reliable Data Transfer

Reliable Data Transfer

Bruce Lee FSM



Reliable Data Transfer 1.0

RDT 1.0

Assumptions:

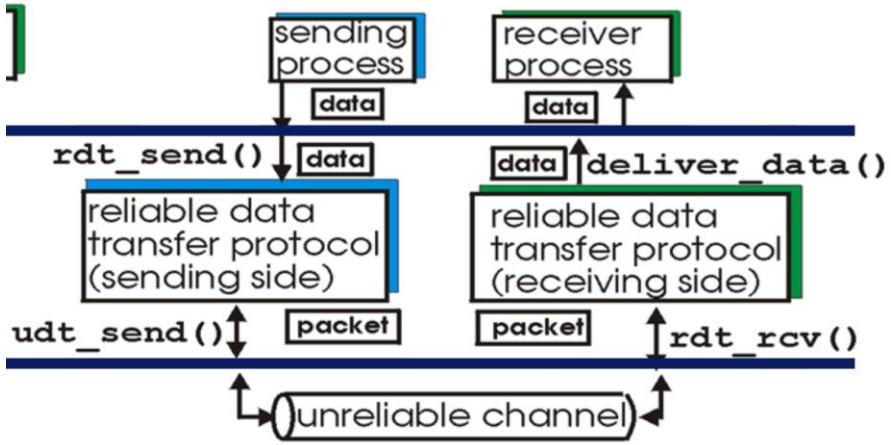
- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

Reliable Data Transfer 1.0

RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

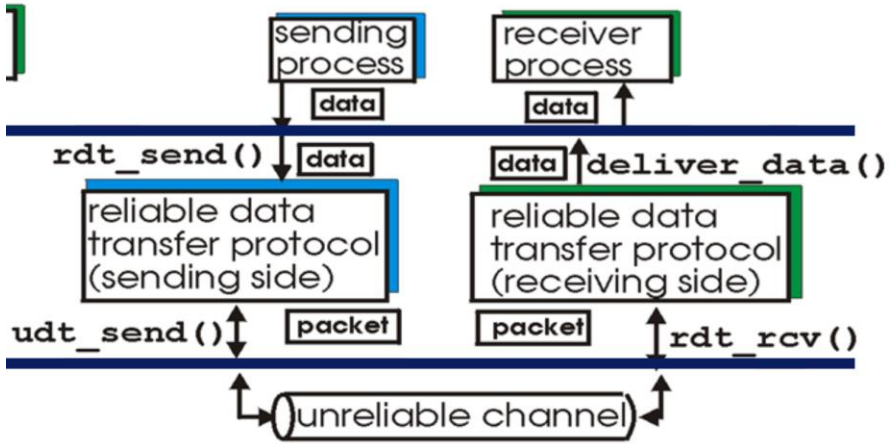


Reliable Data Transfer 1.0

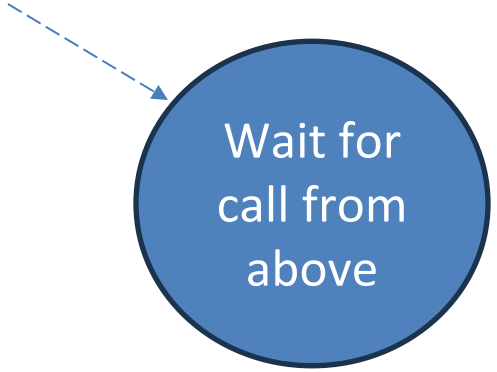
RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering



Sender

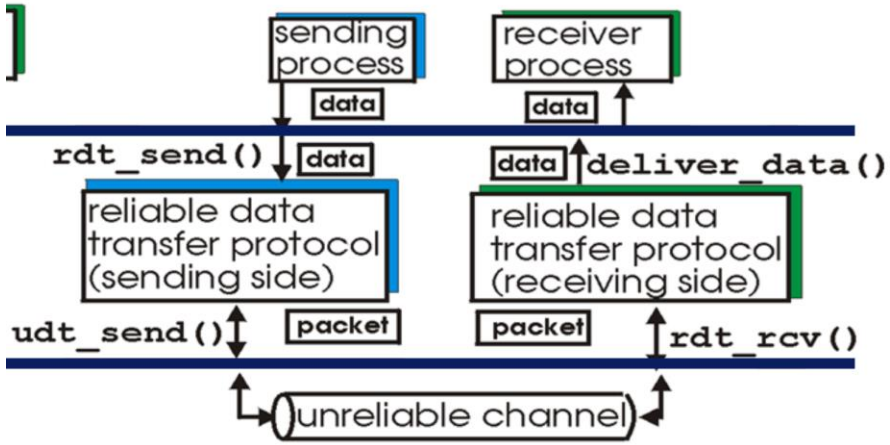


Reliable Data Transfer 1.0

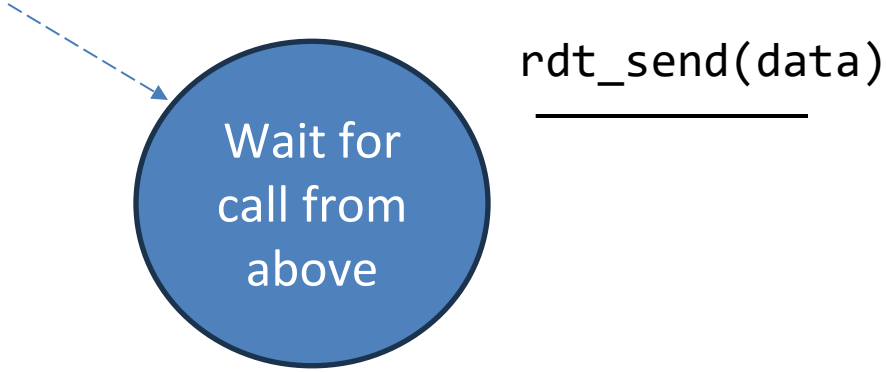
RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering



Sender

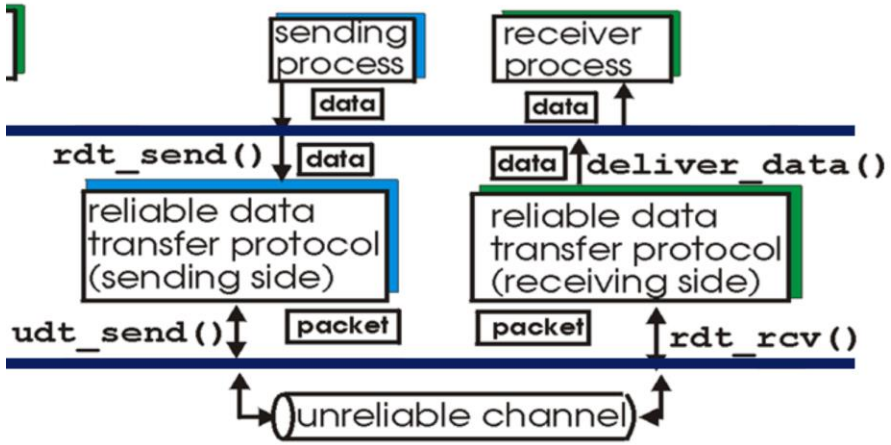


Reliable Data Transfer 1.0

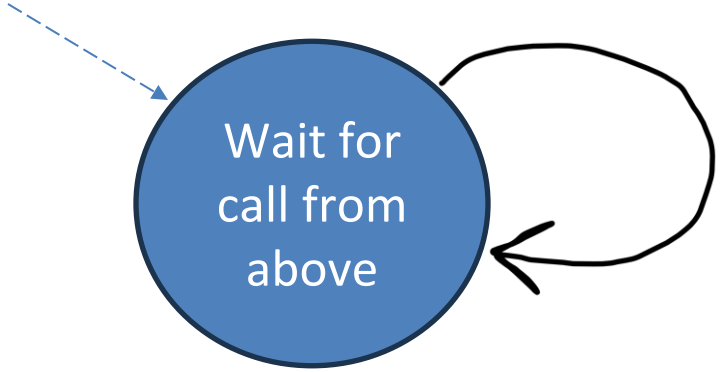
RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering



Sender



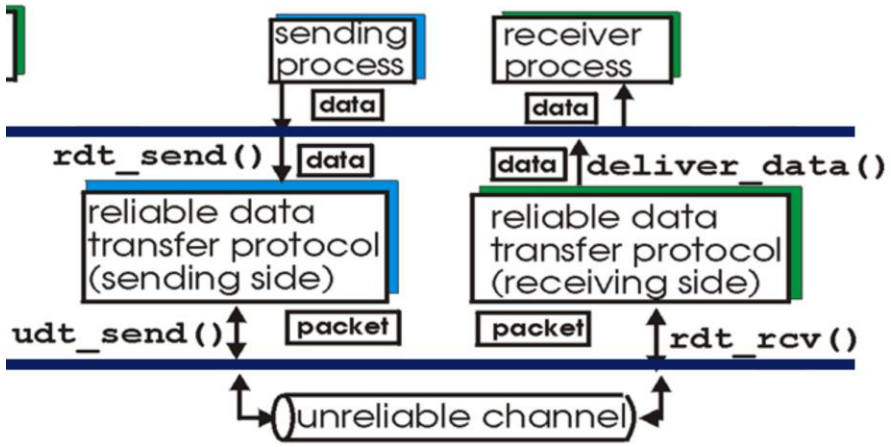

```
rdt_send(data)
_____________

packet = make_pkt(data)
udt_send(packet)
```

Reliable Data Transfer 1.0

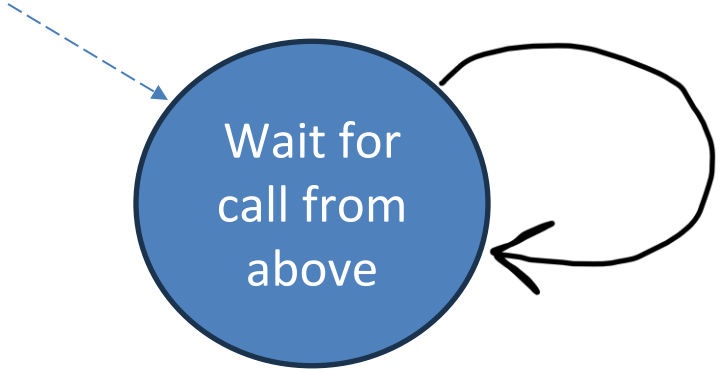
RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

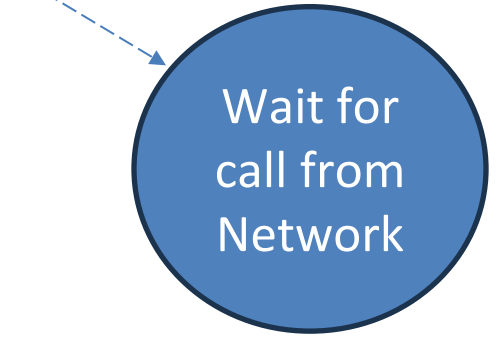


Sender




```
rdt_send(data)
packet = make_pkt(data)
udt_send(packet)
```

Receiver

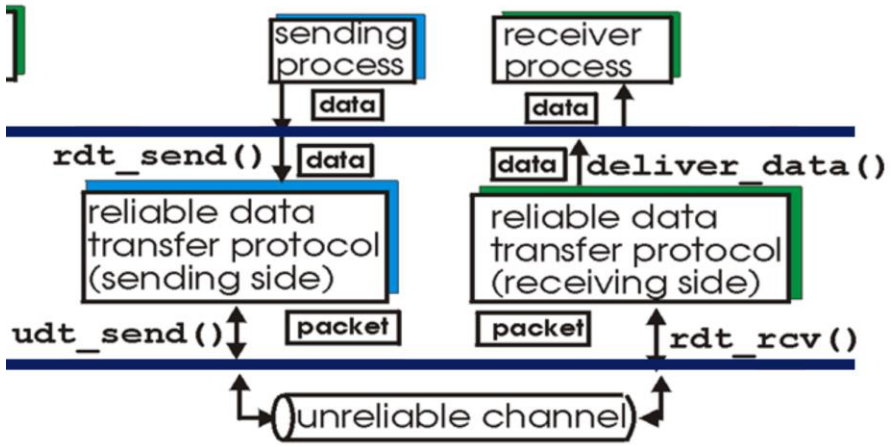


Reliable Data Transfer 1.0

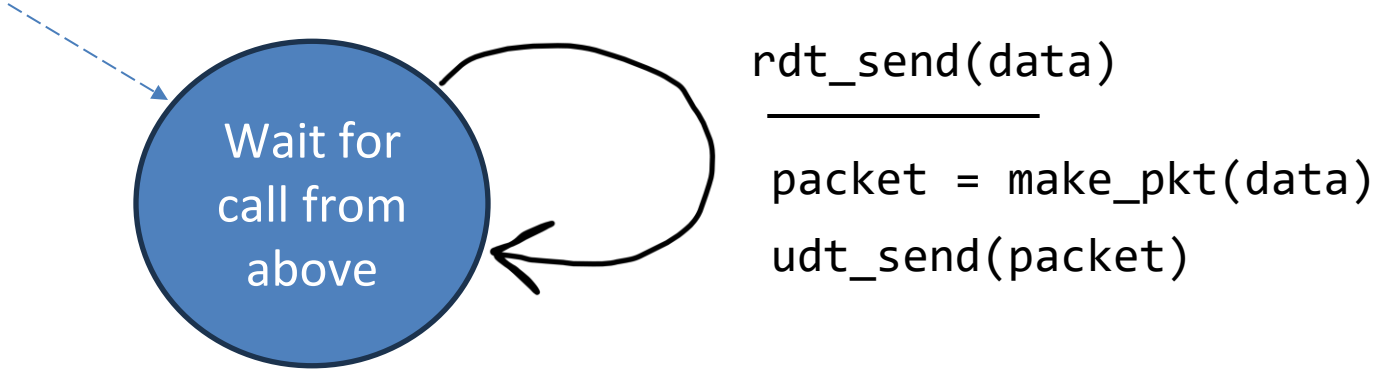
RDT 1.0

Assumptions:

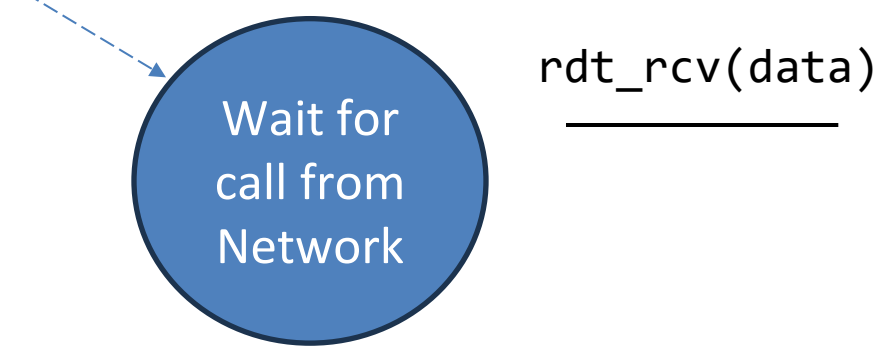
- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering



Sender



Receiver

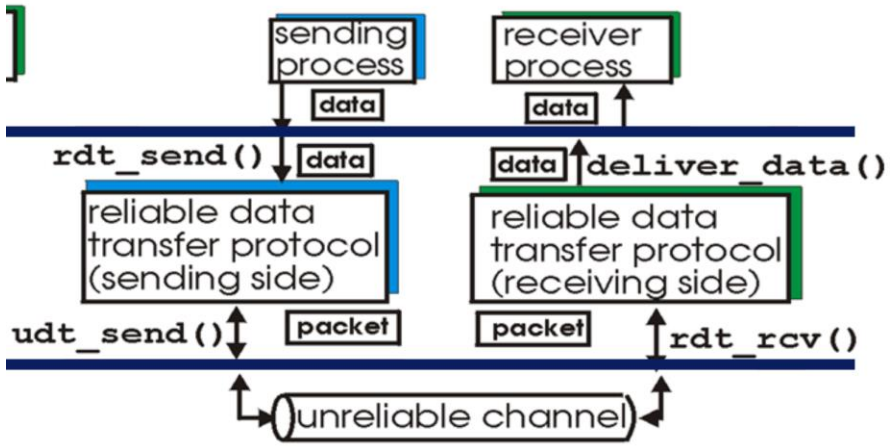


Reliable Data Transfer 1.0

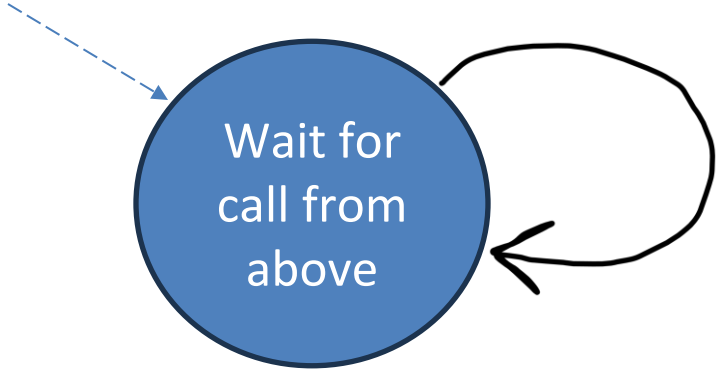
RDT 1.0

Assumptions:

- Unidirectional long data flows
- Perfectly reliable channel
- No bit errors
- No packet loss
- No packet reordering

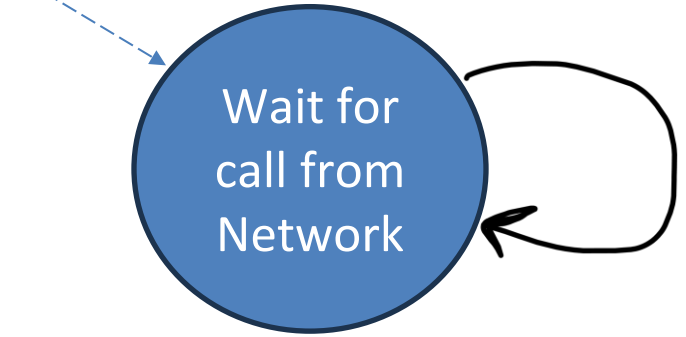


Sender




```
rdt_send(data)
packet = make_pkt(data)
udt_send(packet)
```

Receiver



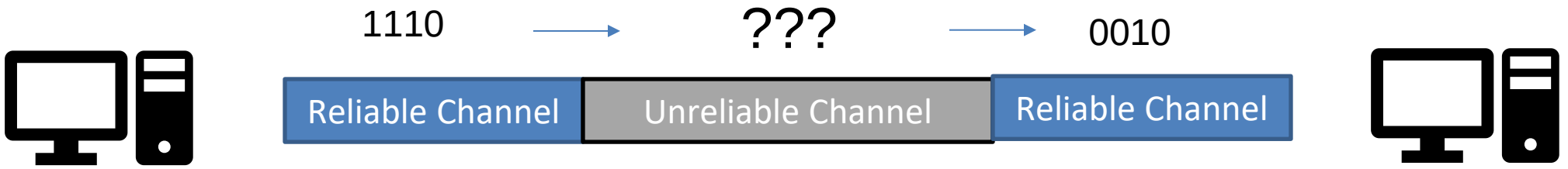

```
rdt_rcv(data)
data = extract(packet)
deliver_data(data)
```

Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?

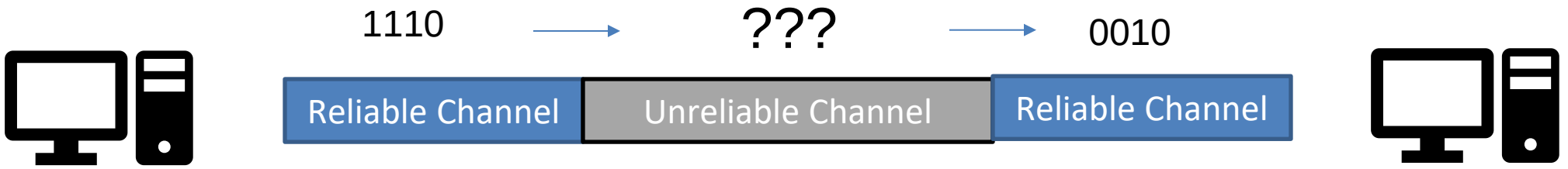


Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?
-Checksum



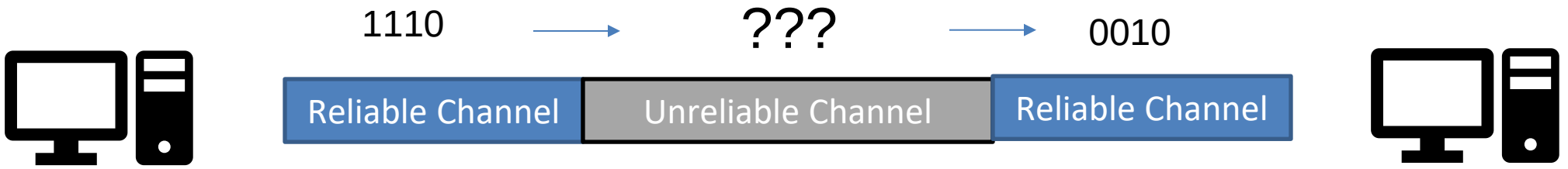
Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?
-Checksum

What is a good way to handle and prevent errors?



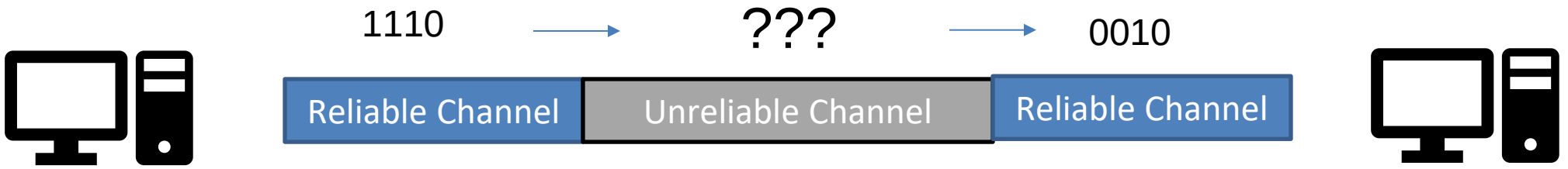
Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?
-Checksum

What is a good way to handle and prevent errors?
- Acknowledged packet, Ask for retransmit if needed



Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?
-Checksum

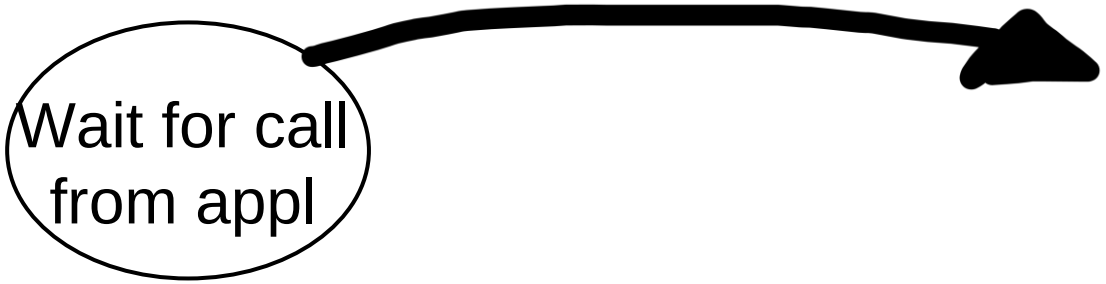
What is a good way to handle and prevent errors?
- Acknowledged packet, Ask for retransmit if needed

Stop-and-wait: sender sends one packet, then waits for receiver response

Sender

```
rdt_send(data)
```

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



Reliable Data Transfer 2.0

RDT 2.0

Potential for bit errors

How can we detect errors?
-Checksum

What is a good way to handle and prevent errors?
- Acknowledged packet, Ask for retransmit if needed

Stop-and-wait: sender sends one packet, then waits for receiver response

Sender

```
rdt_send(data)
```

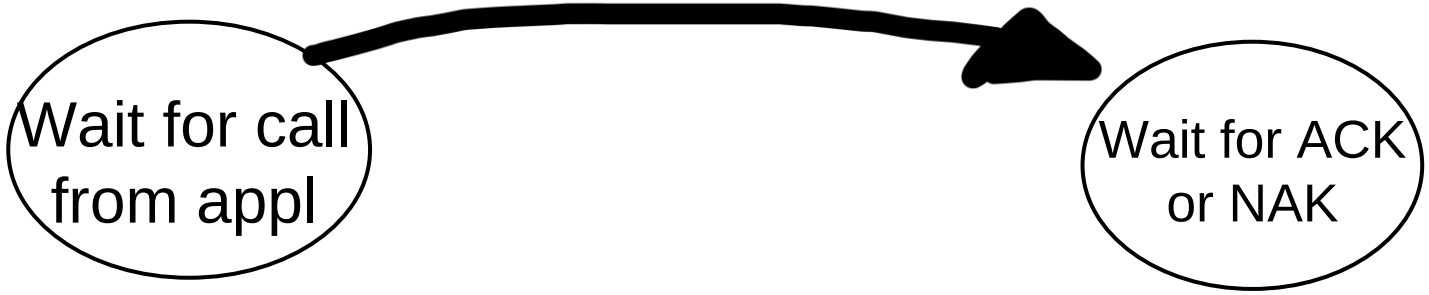
```
sndpkt = make_pkt(data, checksum)
udt_send(sndpkt)
```

Reliable Data Transfer 2.0

Sender

```
rdt_send(data)
```

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



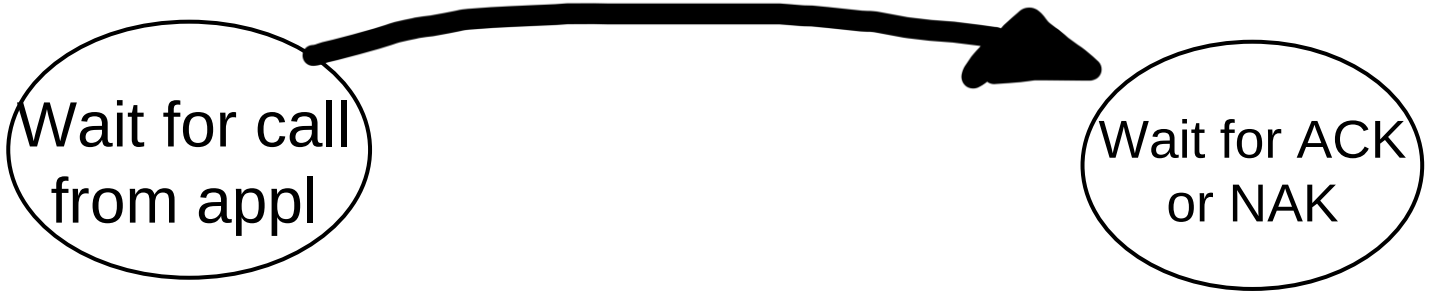
Receiver

Reliable Data Transfer 2.0

Sender

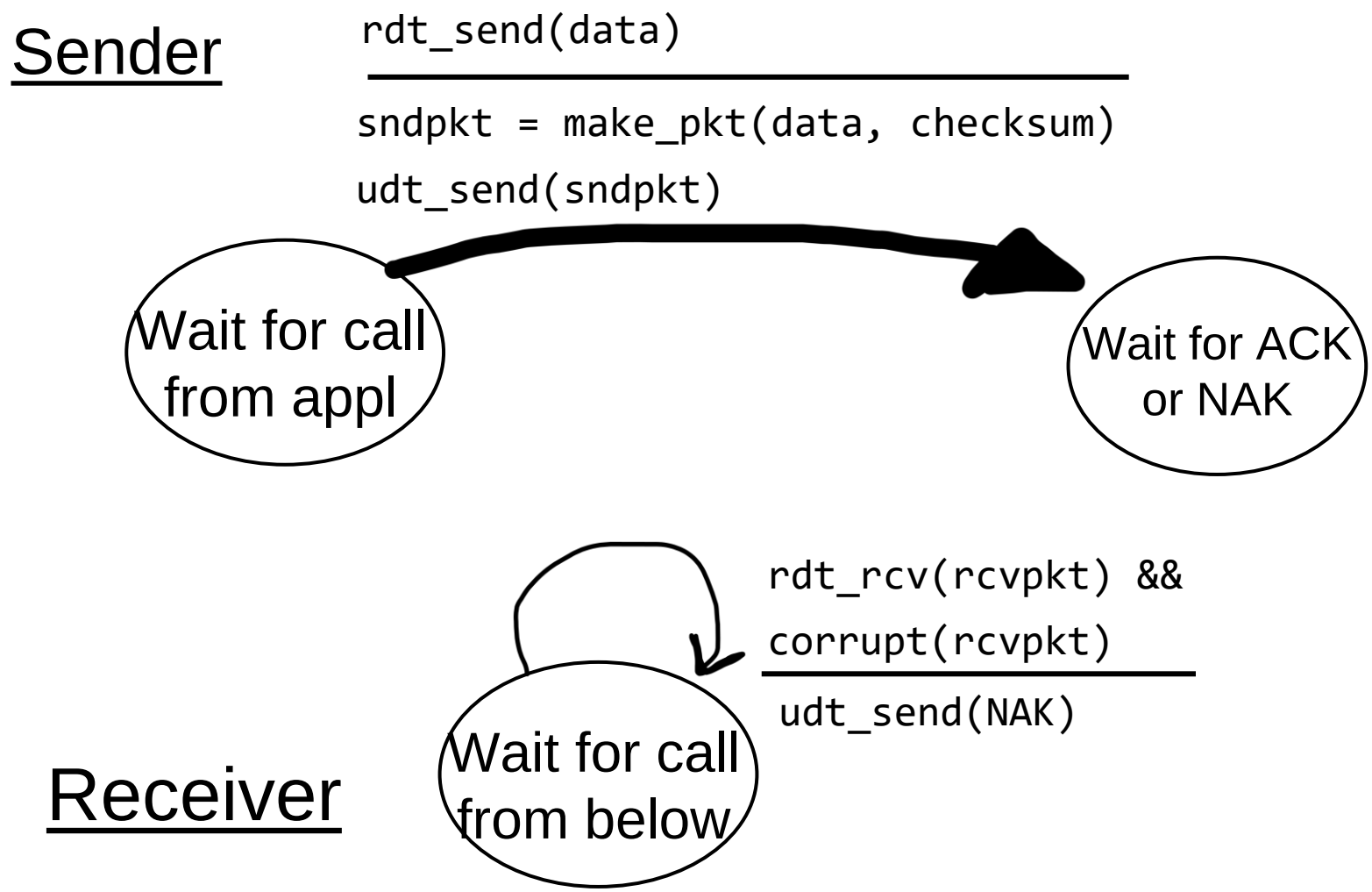
```
rdt_send(data)
```

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



Receiver

Reliable Data Transfer 2.0

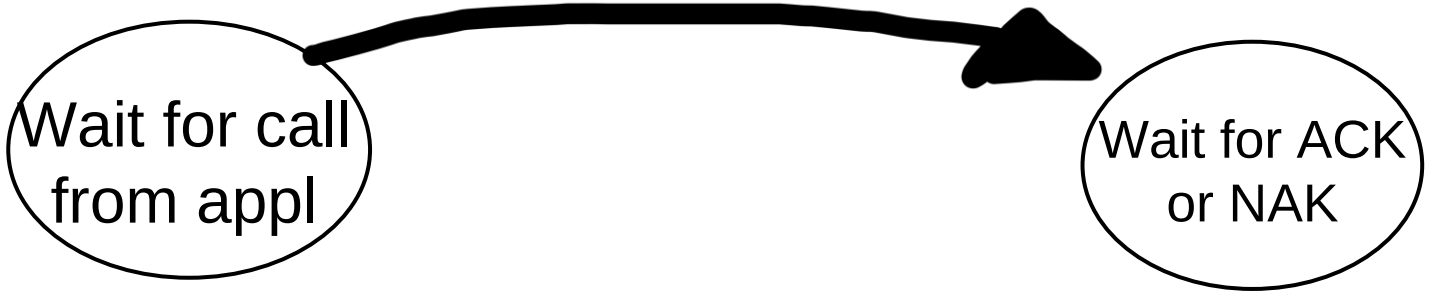


Reliable Data Transfer 2.0

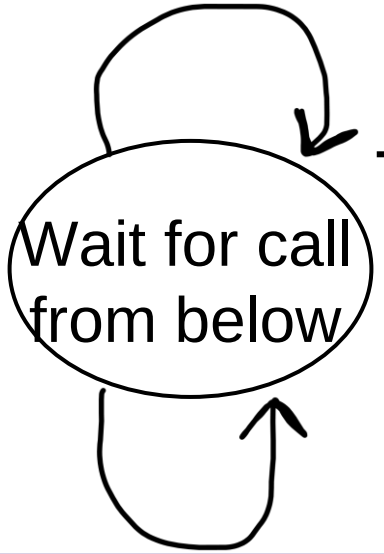
Sender

```
rdt_send(data)
```

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



Receiver




```
rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)
```

```
udt_send(NAK)
```

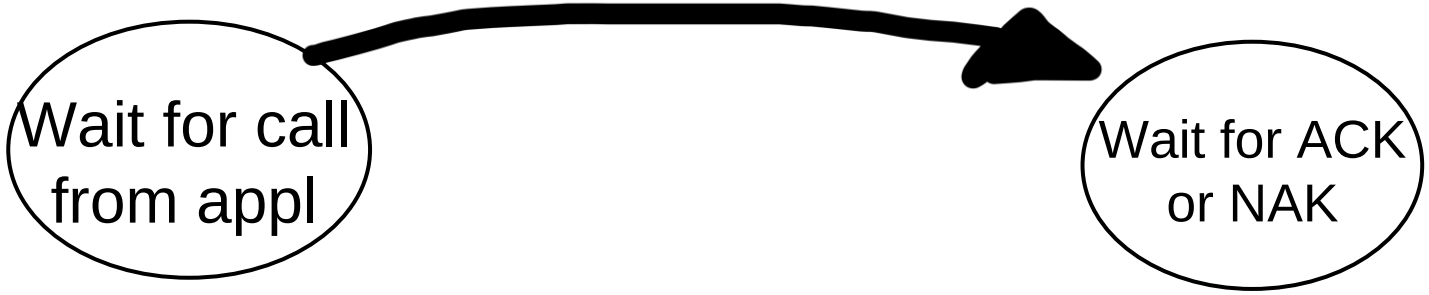
```
rdt_rcv(rcvpkt) &&
!corrupt(rcvpkt)
```

```
data = extract(packet)
deliver_data(data)
udt_send(ACK)
```

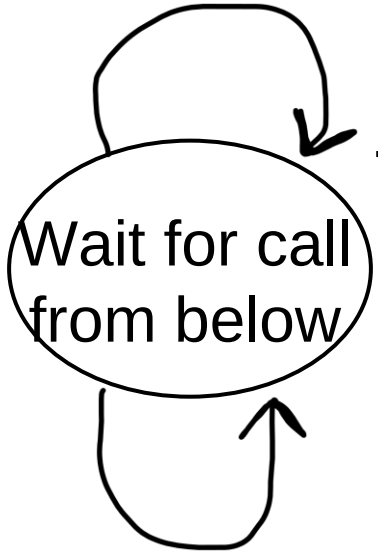
Reliable Data Transfer 2.0

Sender

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



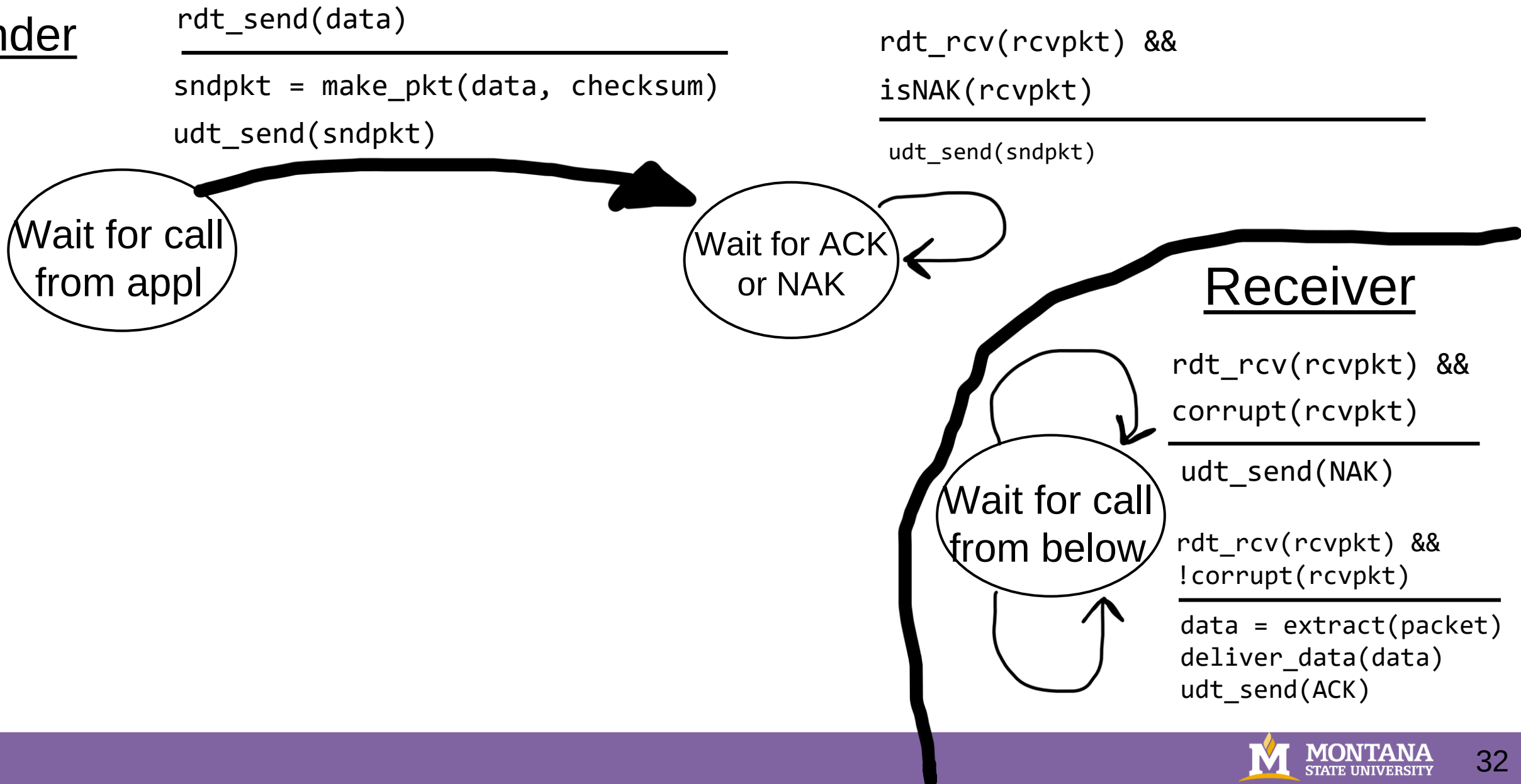
Receiver



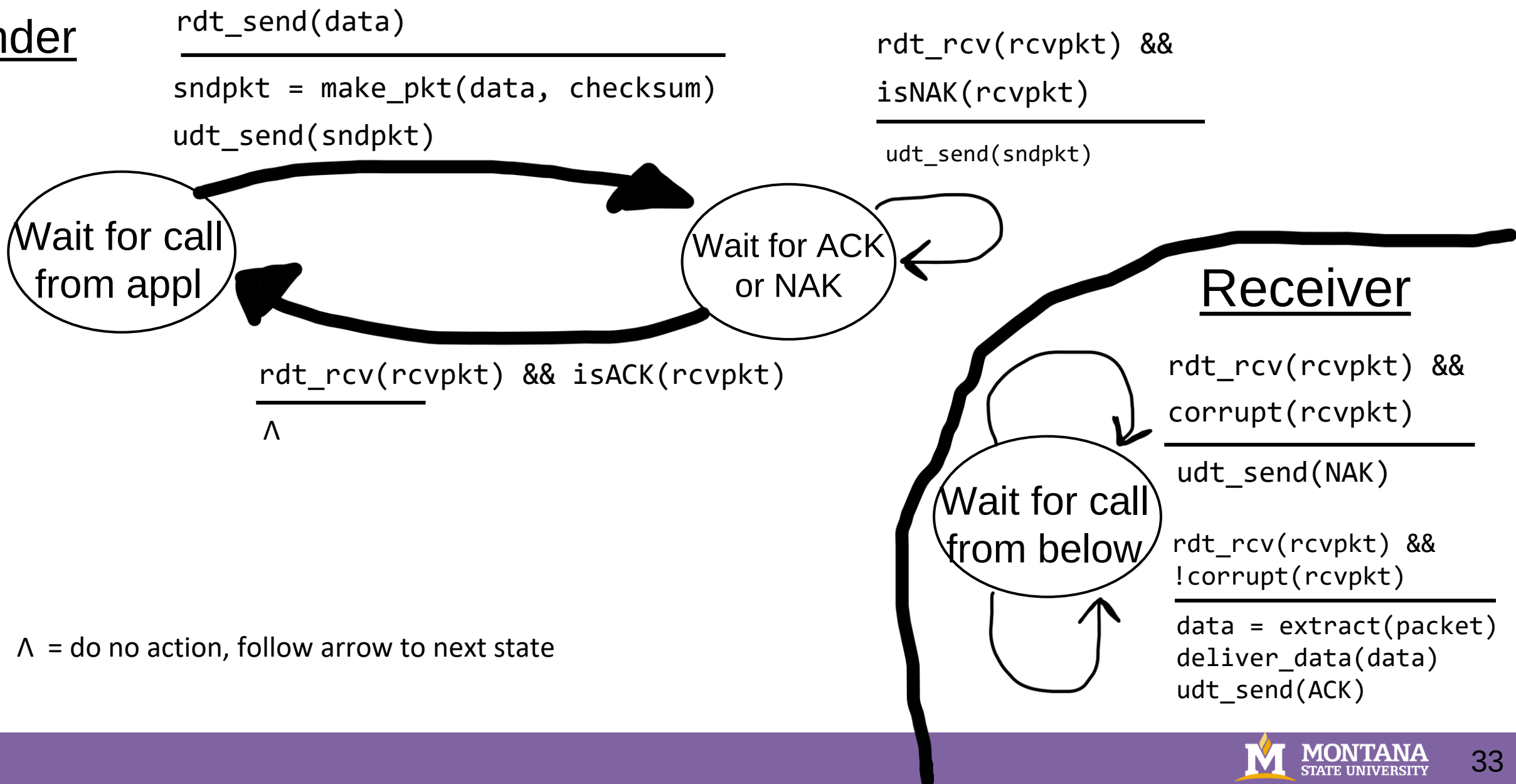

```
rdt_rcv(rcvpkt) &&
corrupt(rcvpkt)
-----------------------------
udt_send(NAK)
```

```
rdt_rcv(rcvpkt) &&
!corrupt(rcvpkt)
-----------------------------
data = extract(packet)
deliver_data(data)
udt_send(ACK)
```

Reliable Data Transfer 2.0



Reliable Data Transfer 2.0



Λ = do no action, follow arrow to next state

Reliable Data Transfer 2.0

What happens if ACK/NAK Corrupted?

→Duplicate delivery, or no retransmission

Solution?

→Retransmit if CORRUPT packet received

How to deal we duplicate Packets?

→ ???

Sender

Wait for
from ap

Λ = do no a

call
low

Receiver

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

rdt_rcv(rcvpkt) &&
!corrupt(rcvpkt)
---
data = extract(packet)
deliver_data(data)
udt_send(ACK)
```

Reliable Data Transfer 2.0

What happens if ACK/NAK Corrupted?

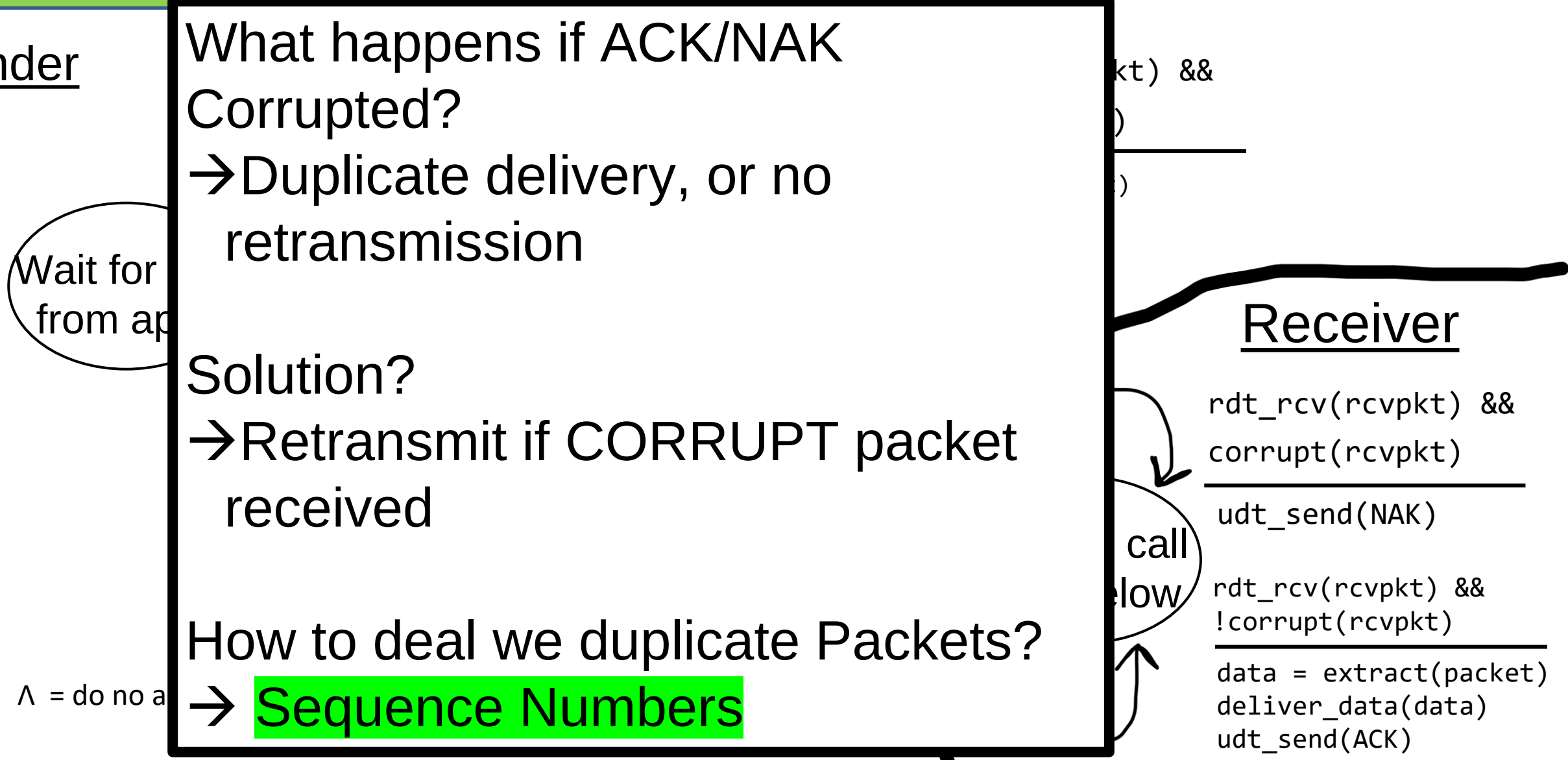
→Duplicate delivery, or no retransmission

Solution?

→Retransmit if CORRUPT packet received

How to deal we duplicate Packets?

→ Sequence Numbers



Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

(We only need to use 0 or 1 for the sequence number)

Sender

Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

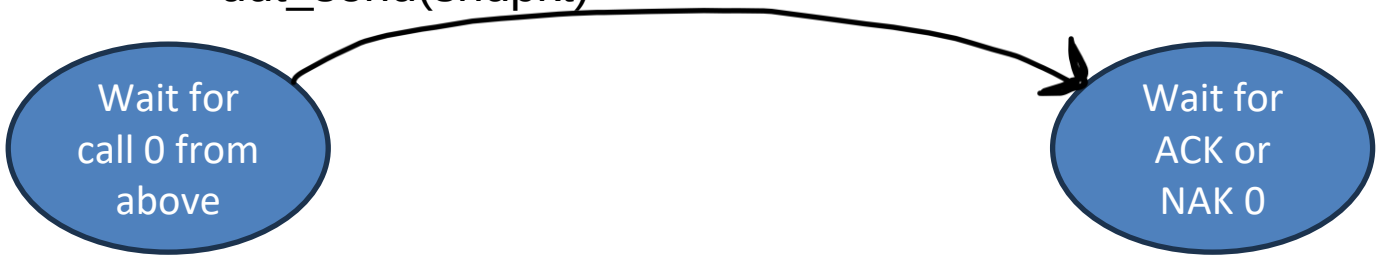
(We only need to use 0 or 1 for the sequence number)

Sender

rdt_send(data)

---

sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)

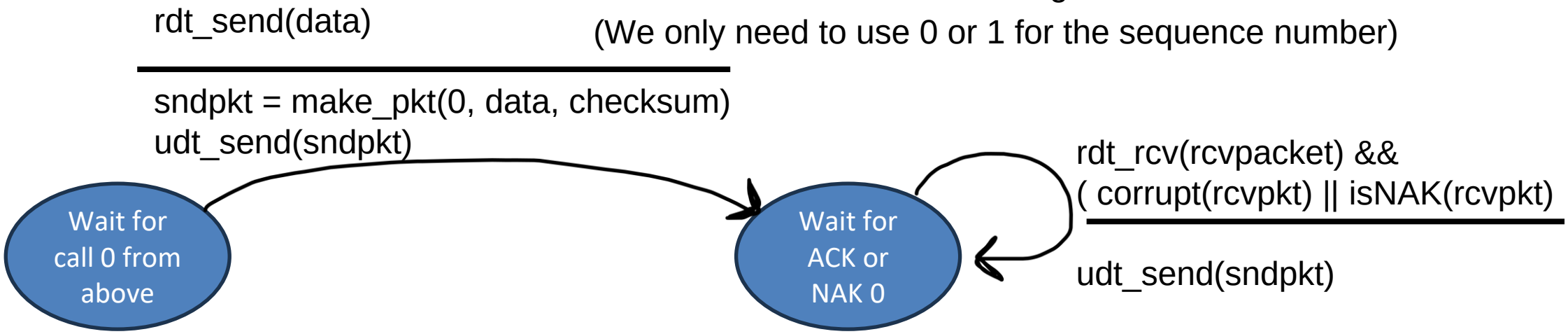


Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

(We only need to use 0 or 1 for the sequence number)

Sender

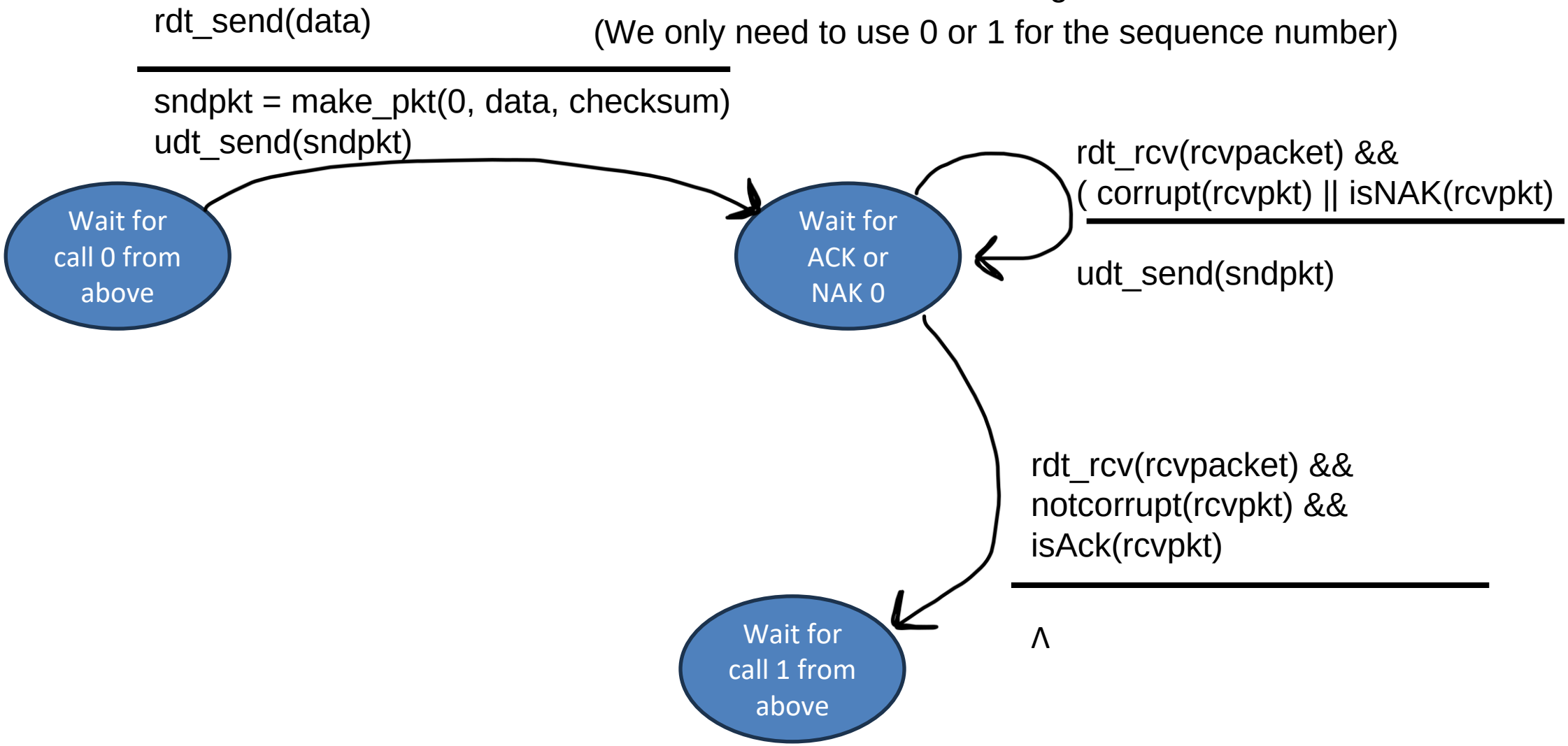


Reliable Data Transfer 2.1

Potential for bit errors and garbled ACKs

(We only need to use 0 or 1 for the sequence number)

Sender

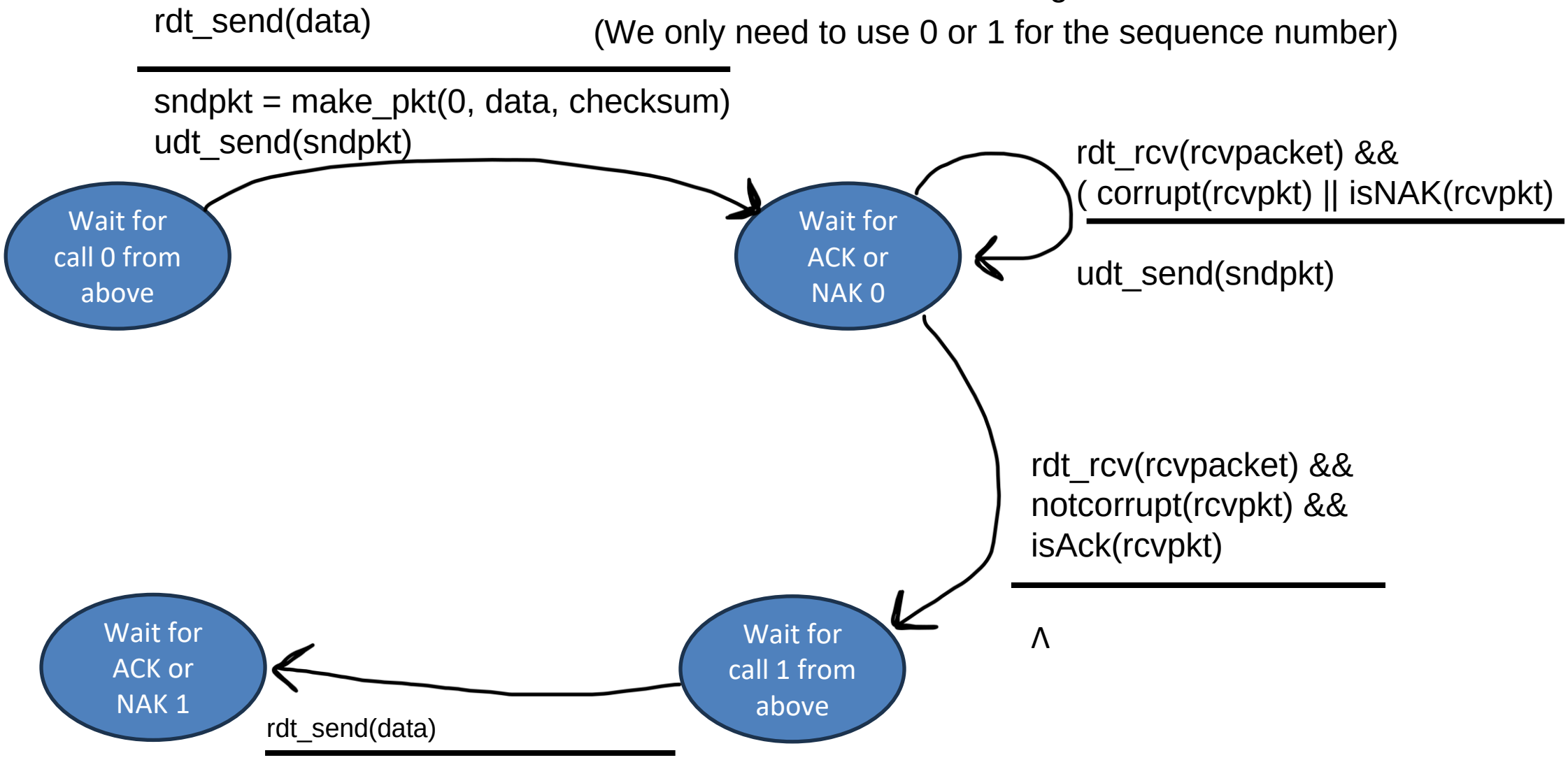


Reliable Data Transfer 2.1

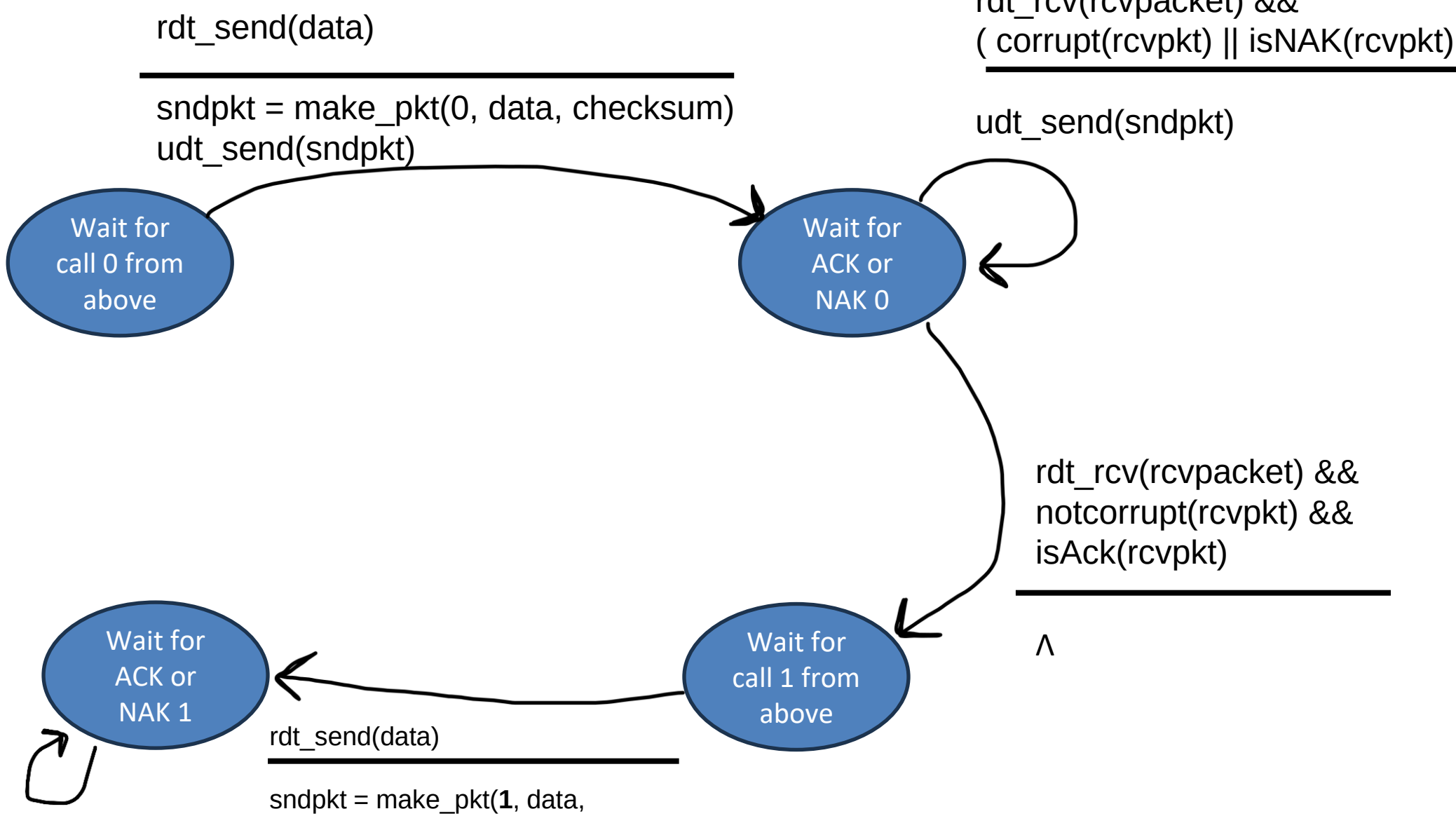
Potential for bit errors and garbled ACKs

(We only need to use 0 or 1 for the sequence number)

Sender

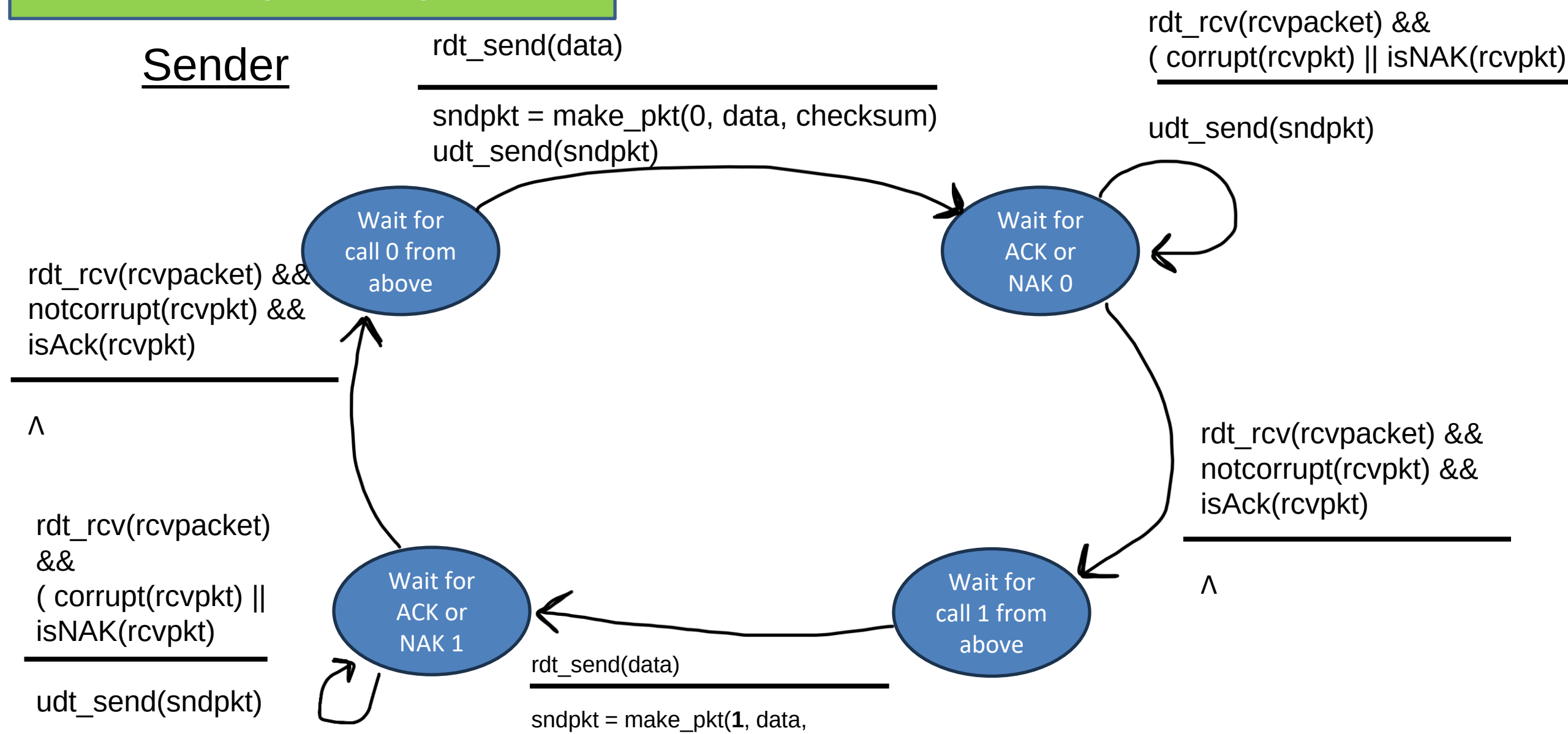


Sender


rdt_send(data)
sndpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK or NAK 0
rdt_rcv(rcvpacket) && (corrupt(rcvpkt) || isNAK(rcvpkt)
udt_send(sndpkt)
rdt_rcv(rcvpacket) && notcorrupt(rcvpkt) && isAck(rcvpkt)
Λ
Wait for call 1 from above
Wait for ACK or NAK 1
rdt_rcv(rcvpacket) && (corrupt(rcvpkt) || isNAK(rcvpkt)
udt_send(sndpkt)
rdt_send(data)
sndpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)

Transport Layer

Sender



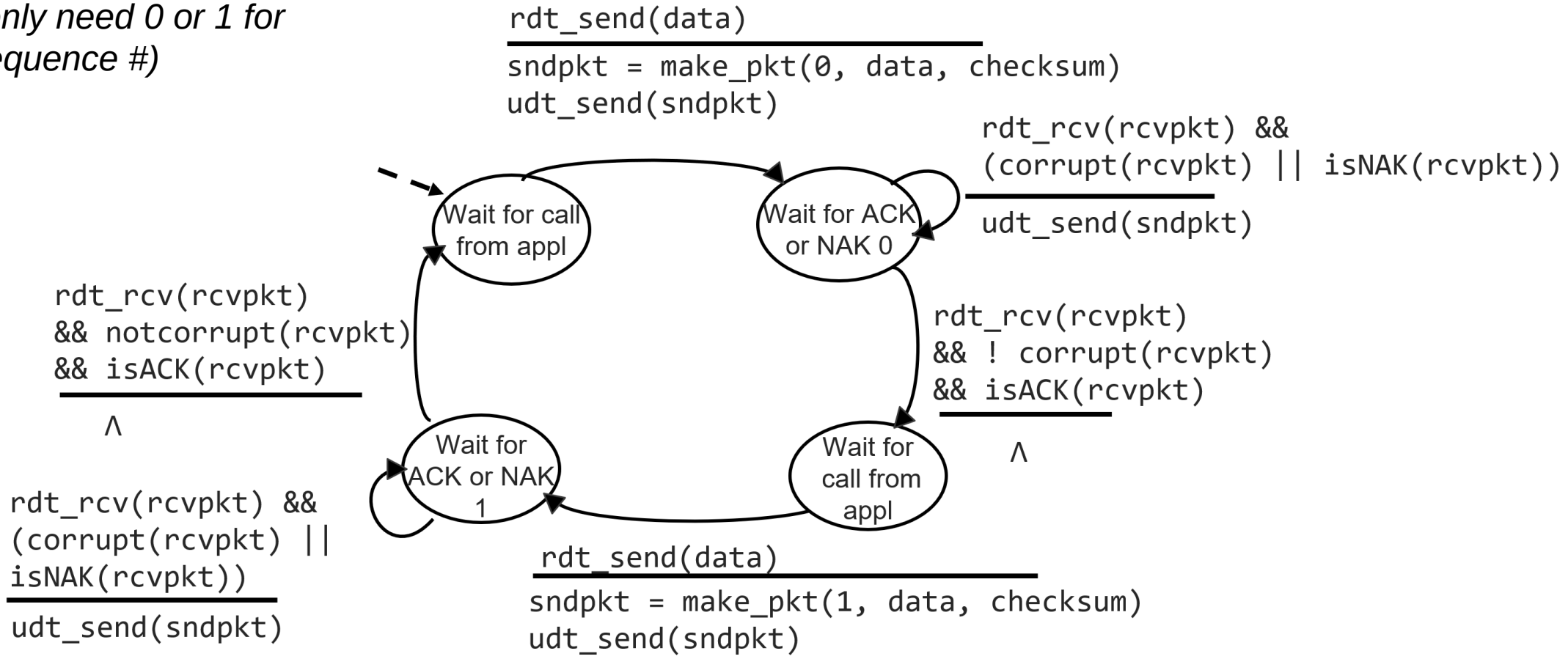
Reliable Data Transfer 2.1

RDT 2.1

Potential for bit errors and garbled ACKs

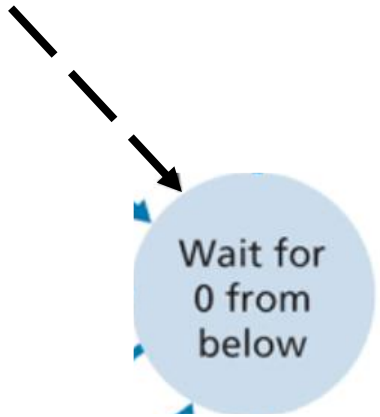
Stop-and-wait: sender sends one packet, then waits for receiver response

(We only need 0 or 1 for the sequence #)



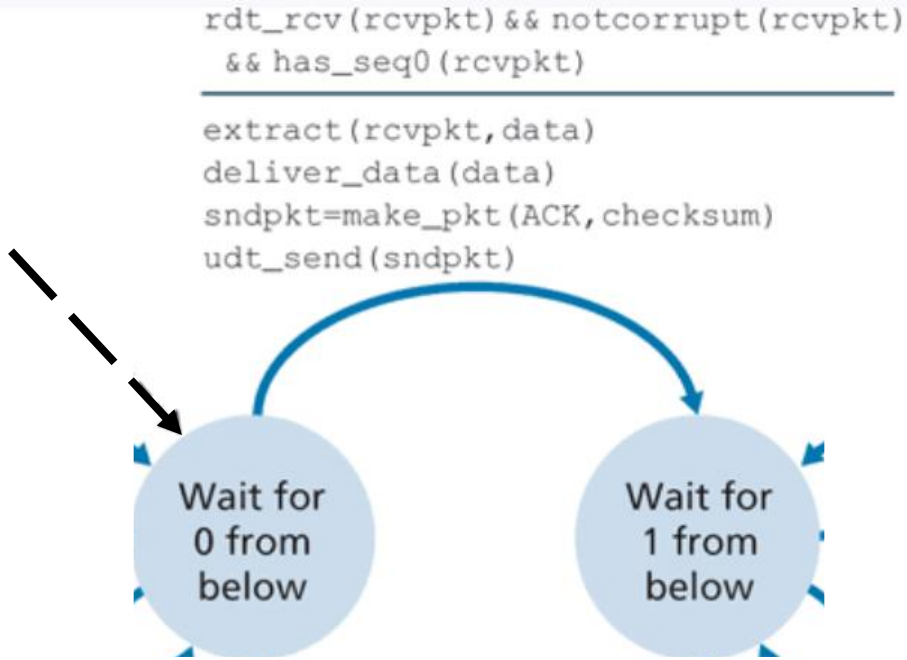
Reliable Data Transfer 2.1

Receiver



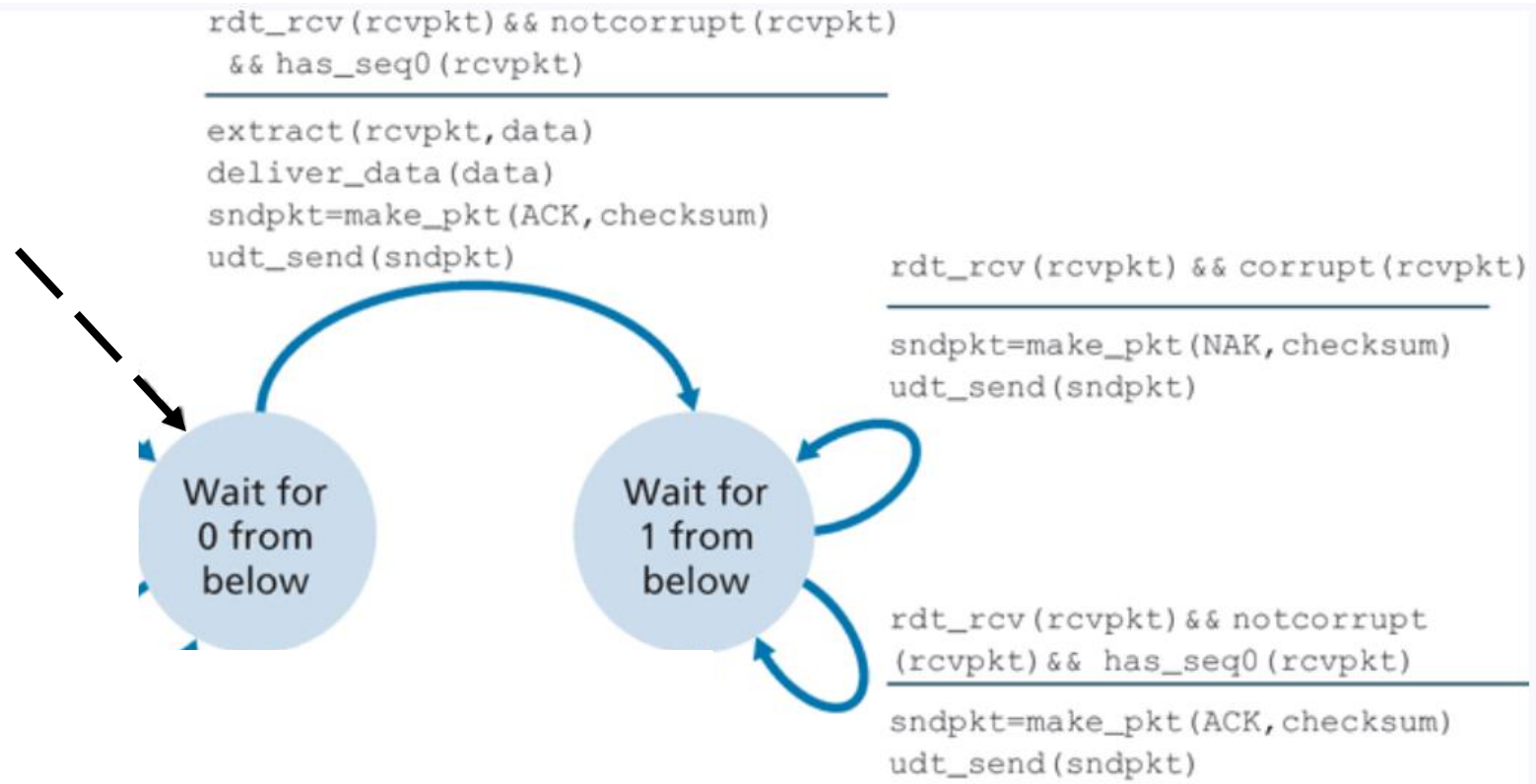
Reliable Data Transfer 2.1

Receiver



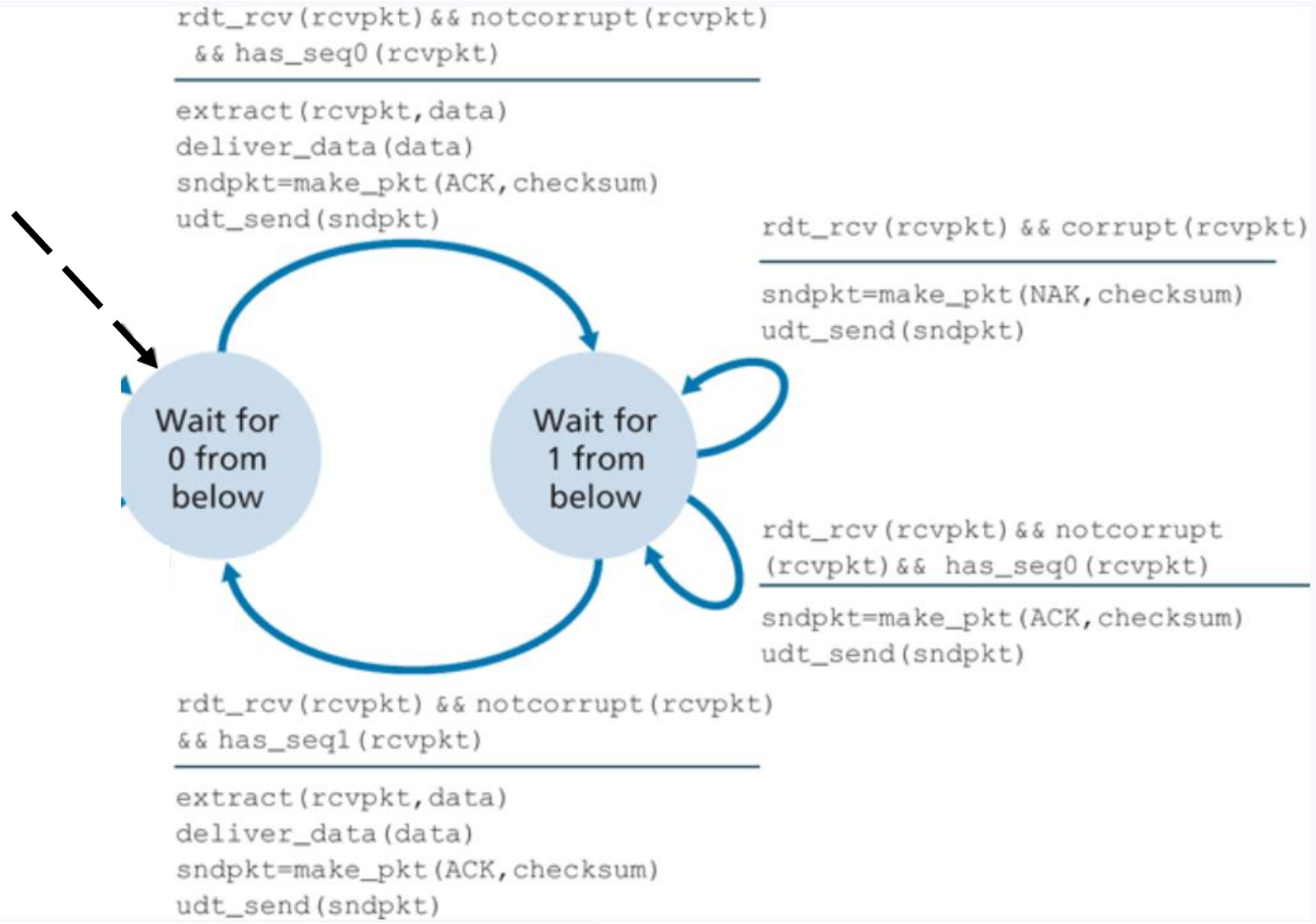
Reliable Data Transfer 2.1

Receiver



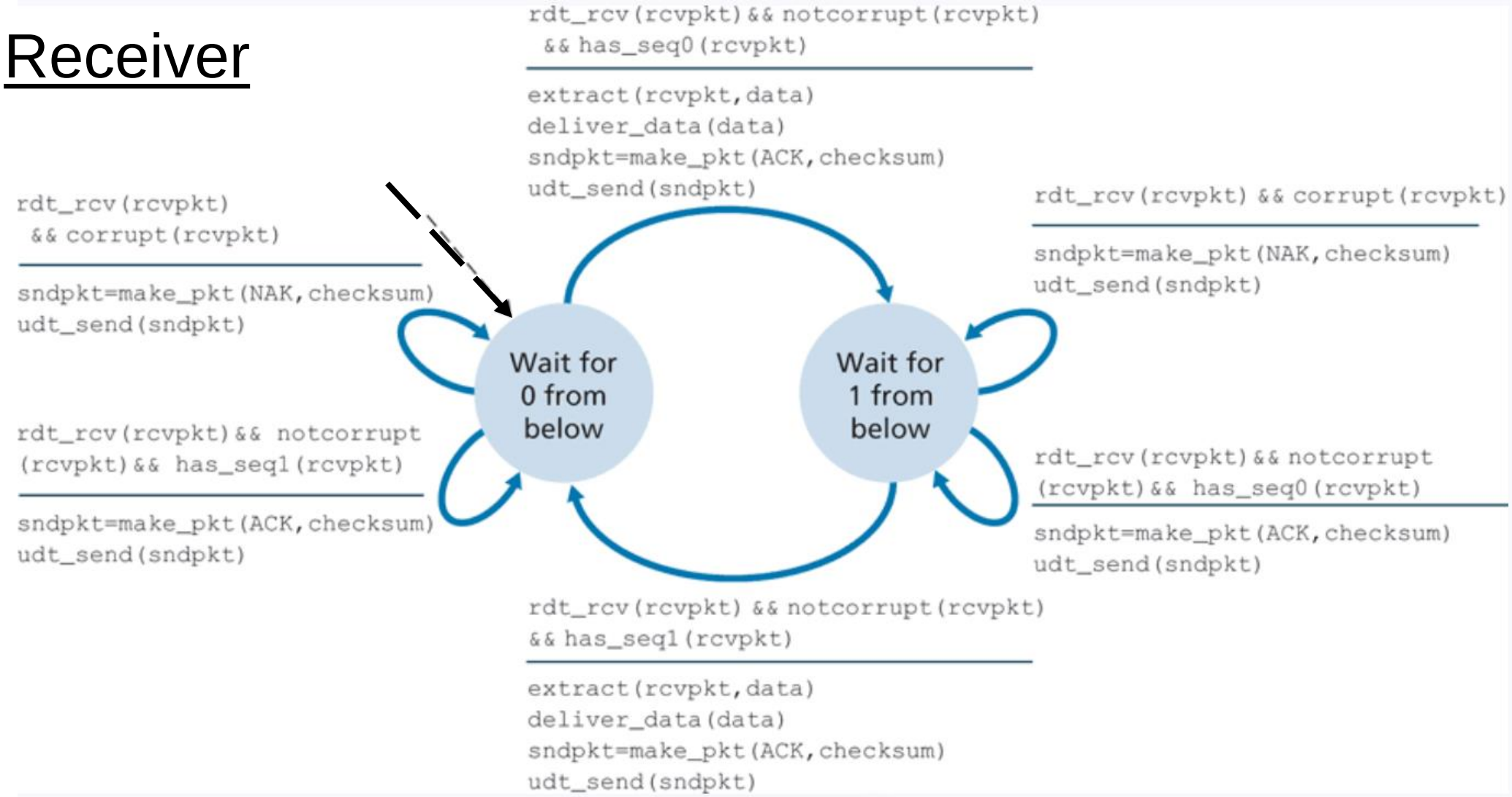
Reliable Data Transfer 2.1

Receiver



Reliable Data Transfer 2.1

Receiver

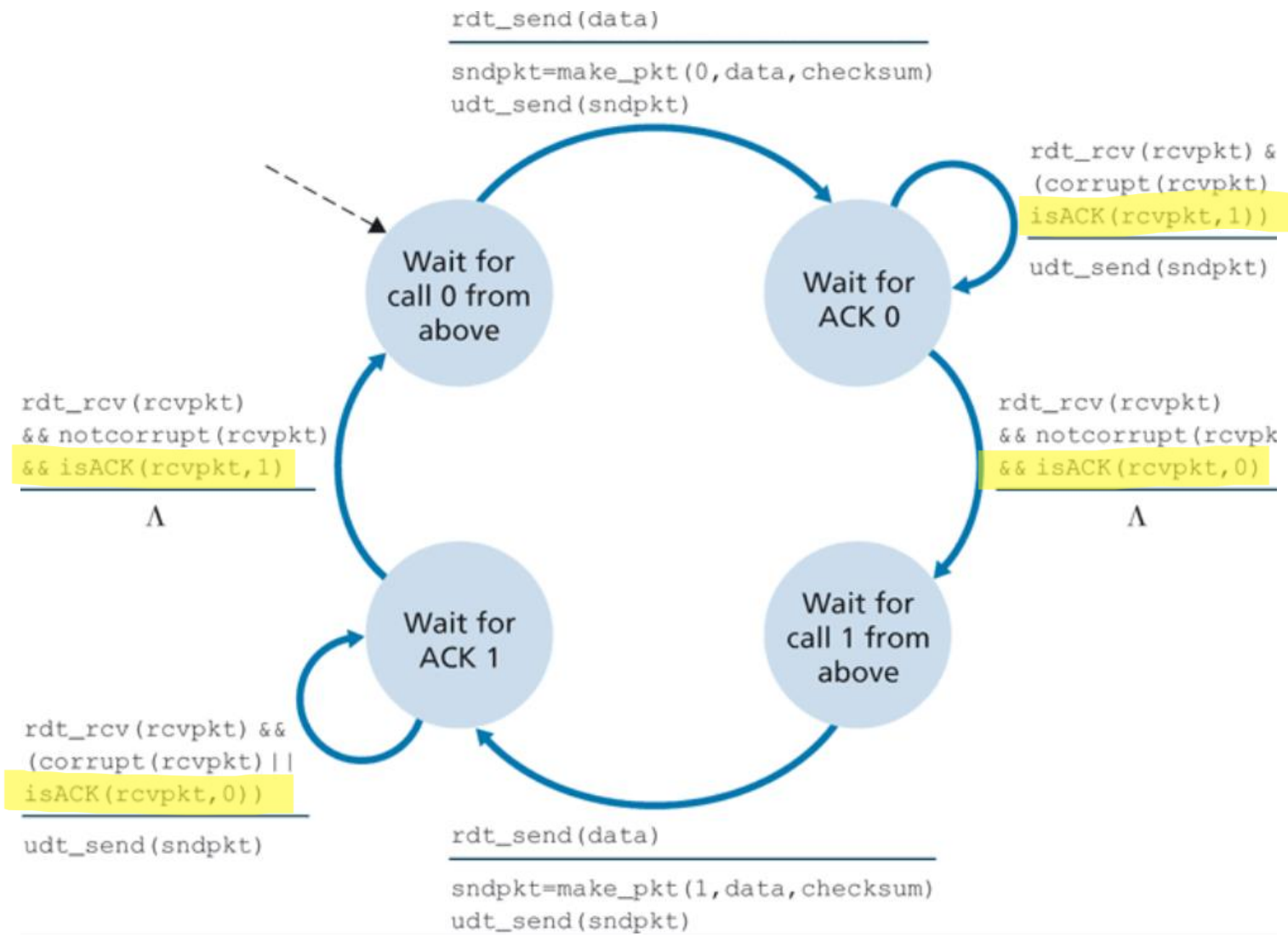


Reliable Data Transfer 2.2

Sender

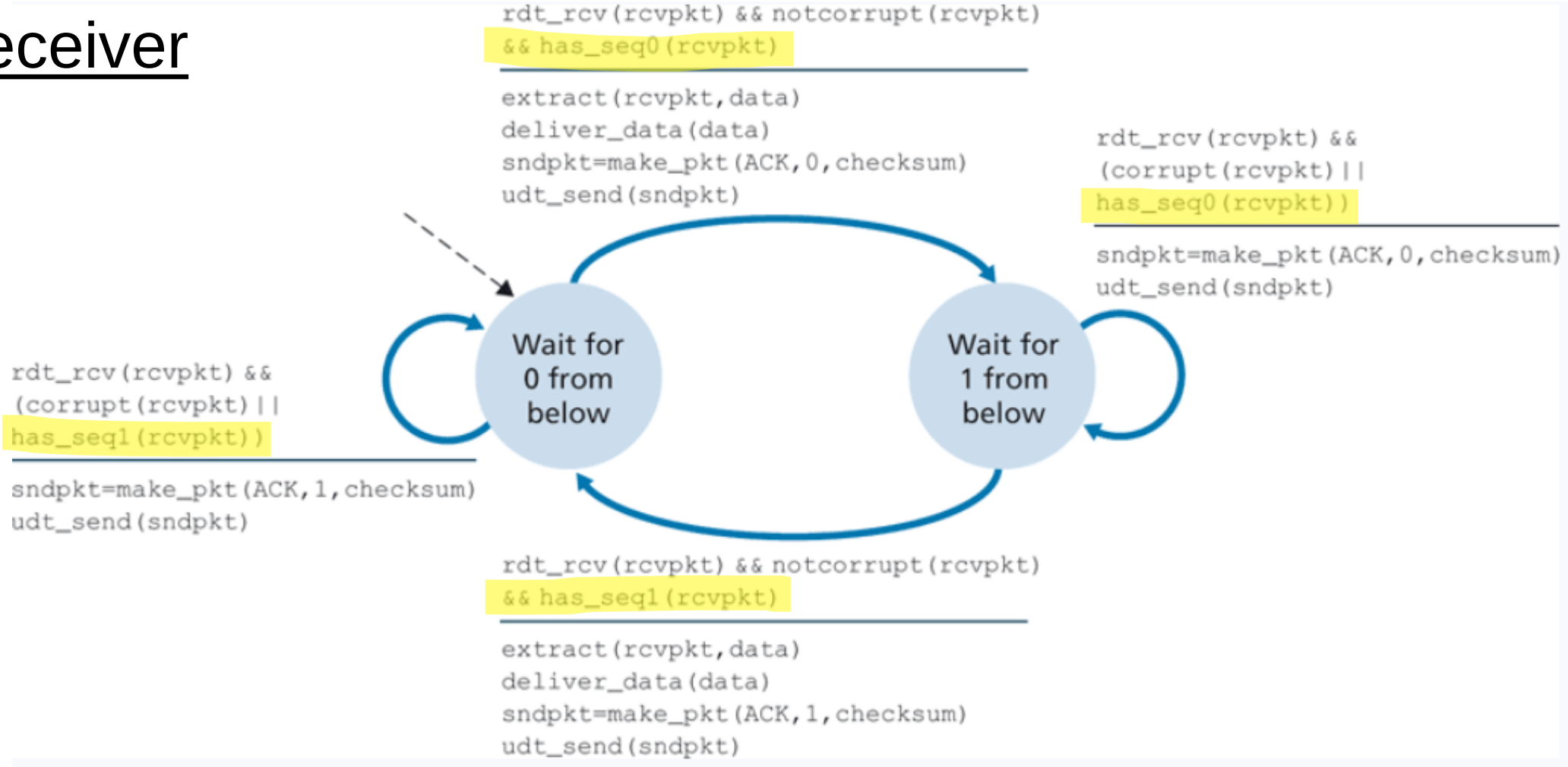
Instead of NAKs, we provide the sequence number of the packet for our ACK (ACK1, ACK0)

Duplicate ACK at sender results in same action as NAK: retransmit current pkt



Reliable Data Transfer 2.2

Receiver



Transport Layer

A quick meatball break…

Reliable Data Transfer 3.0

We will still need checksums, Seq #'s, ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

Reliable Data Transfer 3.0

We will still need checksums, Seq #'s, ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

Solution?

Reliable Data Transfer 3.0

We will still need checksums, Seq #'s, ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

A timer

Retransmit if ACK received in X amount of time

What if the ACK is just taking a really long time to arrive?

Reliable Data Transfer 3.0

We will still need checksums, Seq #'s, ACKS, but we need more

Potential for:

- bit errors
- garbled ACKs
- **Lost/dropped packets**

What if an ACK gets dropped? Sender is stuck waiting forever

A timer

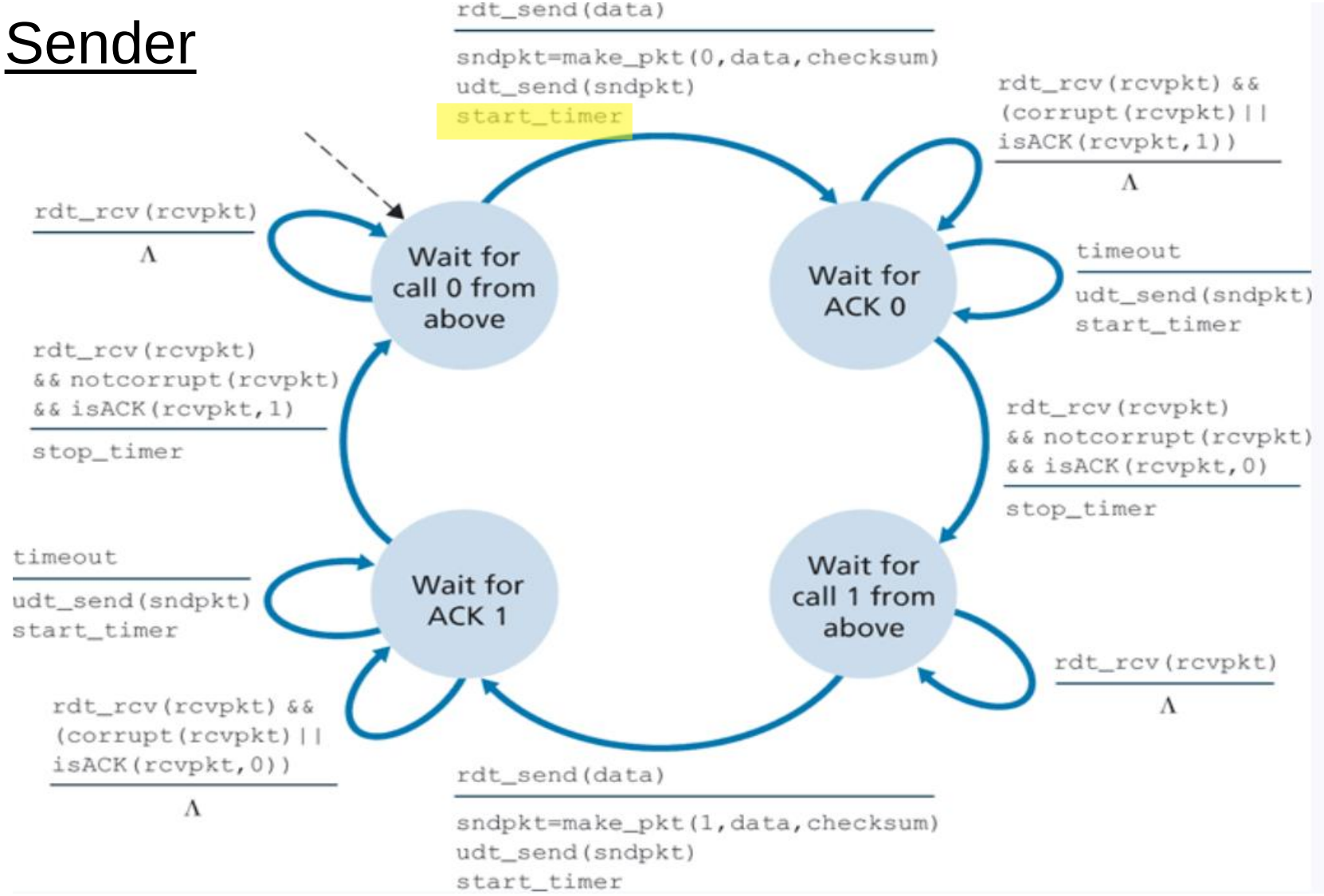
Retransmit if ACK received in X amount of time

What if the ACK is just taking a really long time to arrive? This will be duplicate data, but we have a Seq # to handle that

Reliable Data Transfer 3.0

(Packets can get dropped or lost)

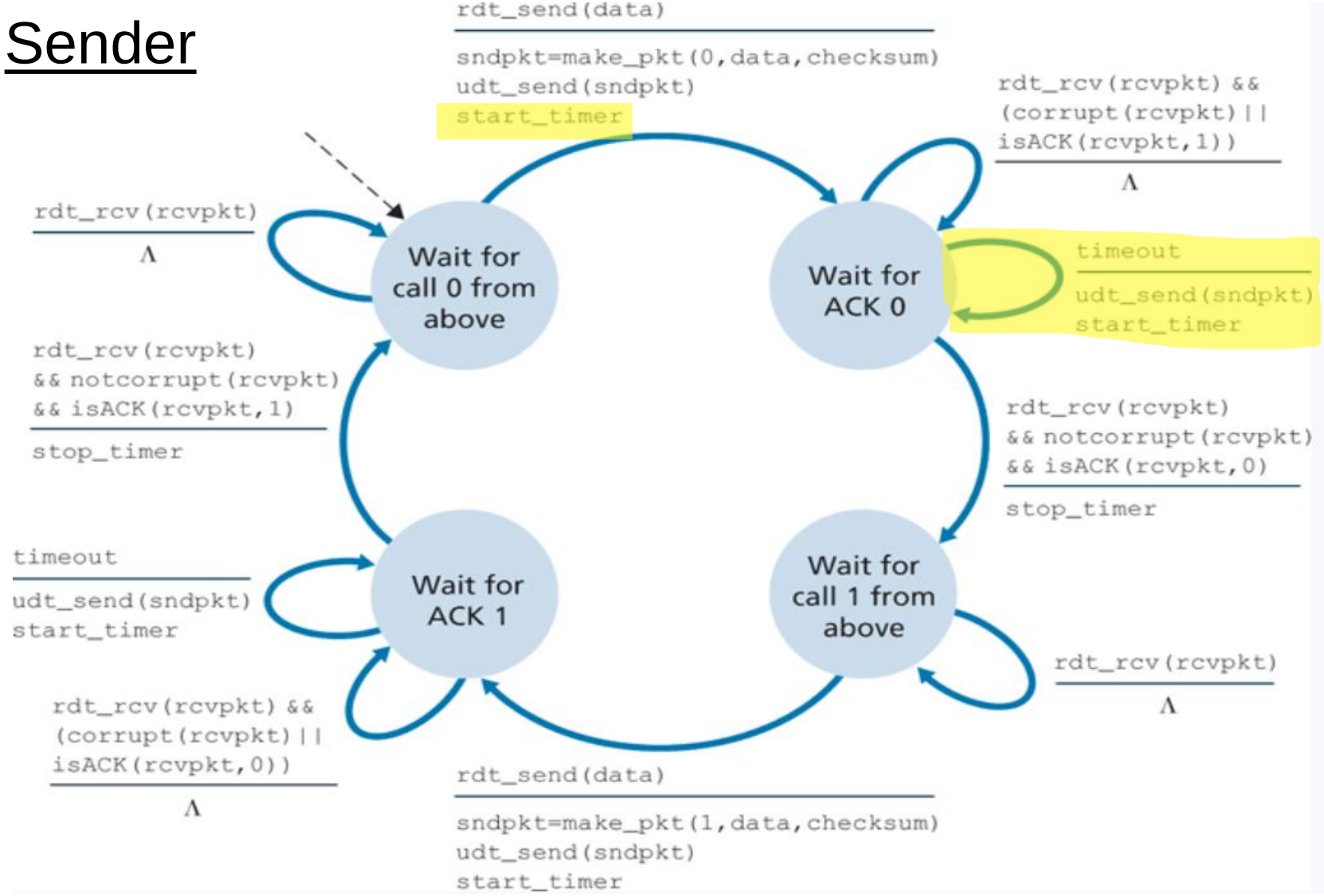
Sender



Reliable Data Transfer 3.0

(Packets can get dropped or lost)

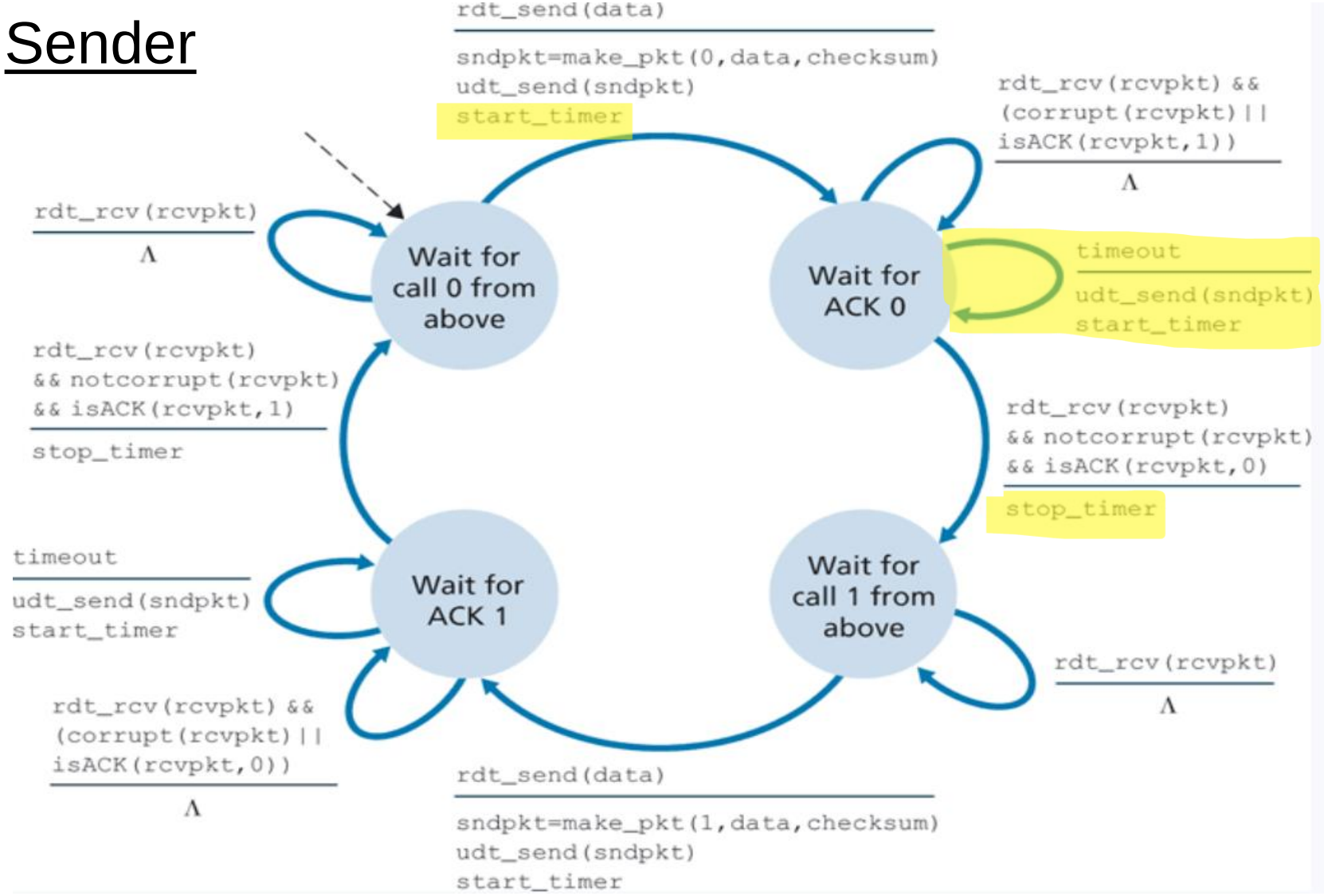
Sender



Reliable Data Transfer 3.0

(Packets can get dropped or lost)

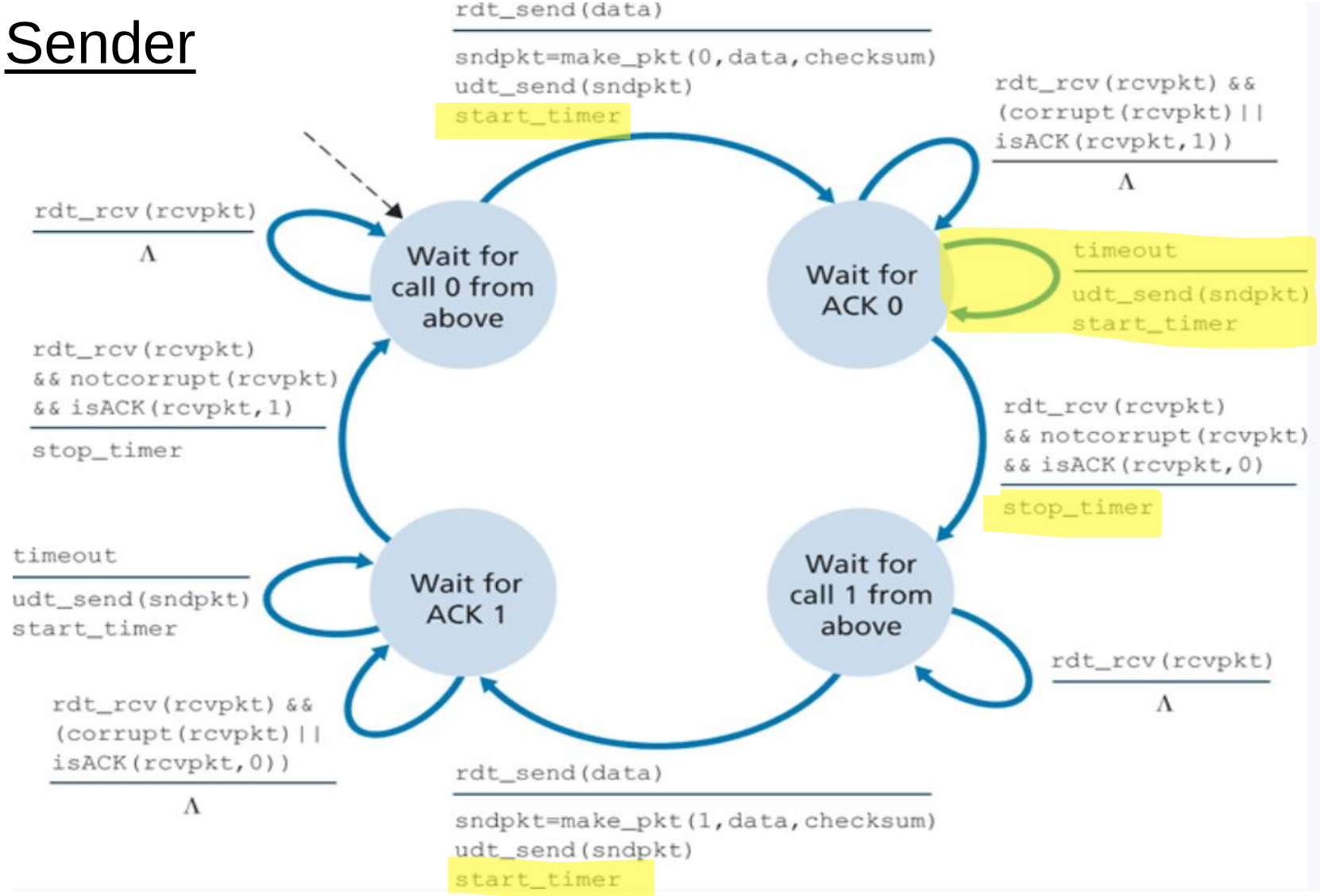
Sender



Reliable Data Transfer 3.0

(Packets can get dropped or lost)

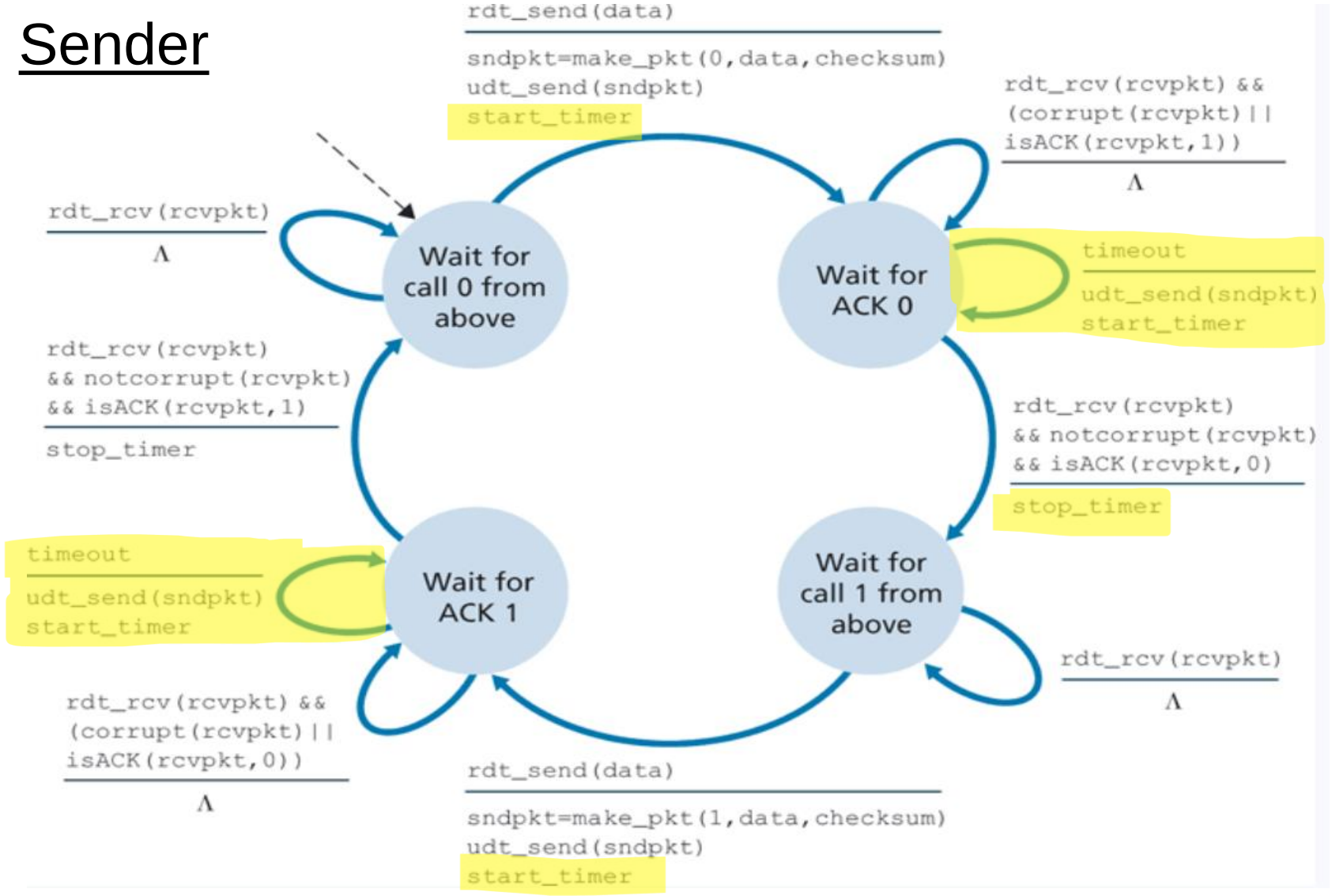
Sender



Reliable Data Transfer 3.0

(Packets can get dropped or lost)

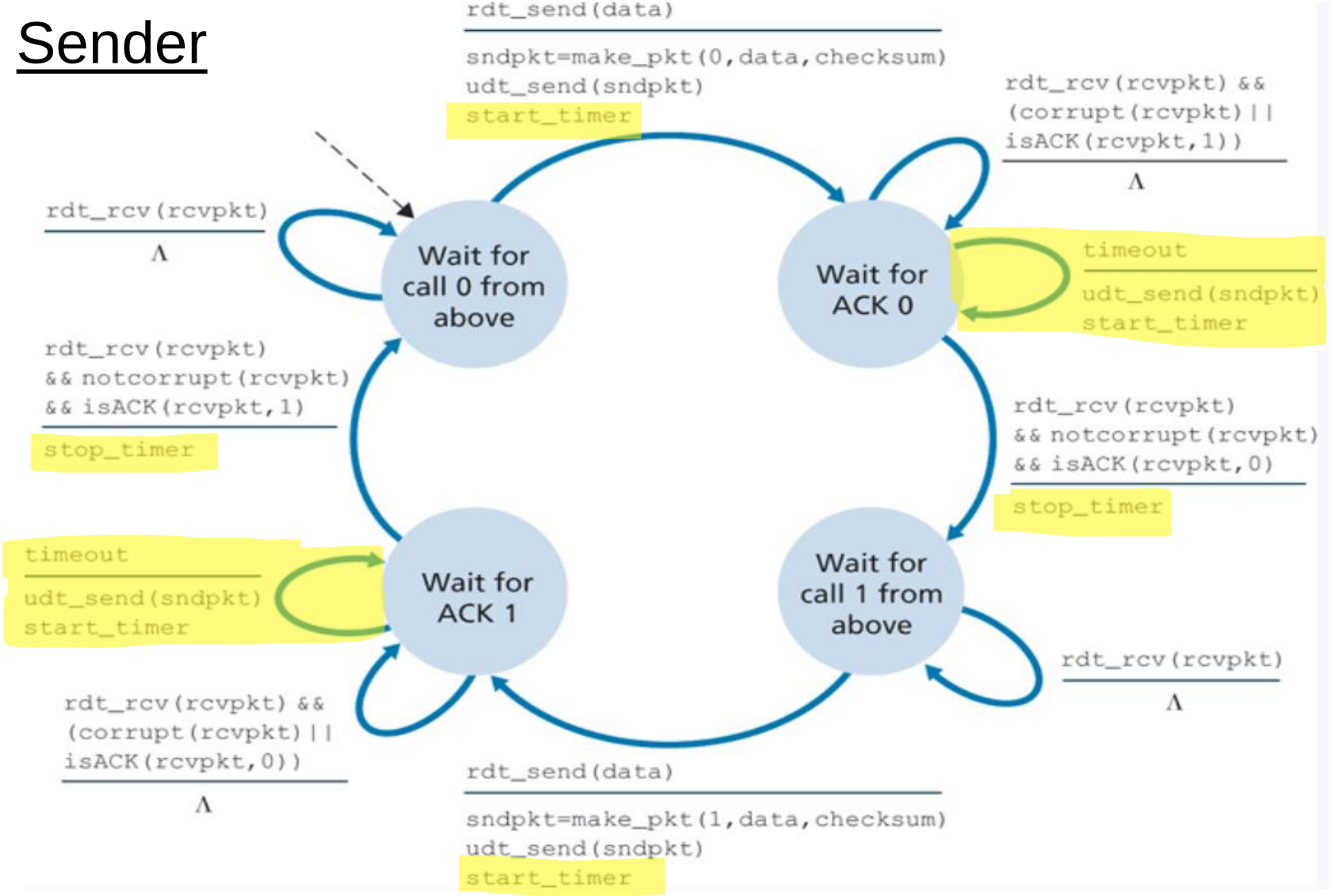
Sender



Reliable Data Transfer 3.0

(Packets can get dropped or lost)

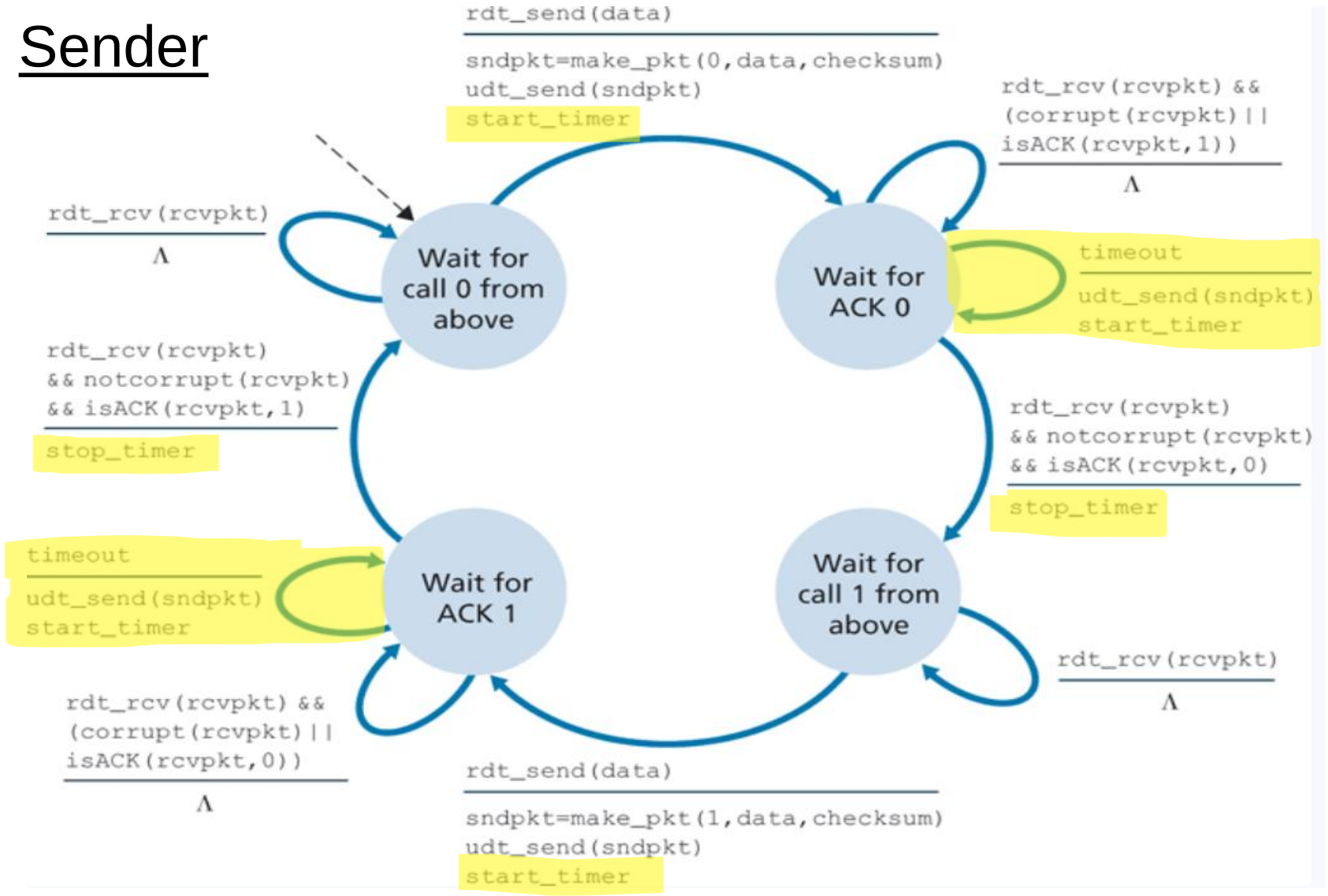
Sender



Reliable Data Transfer 3.0

(Packets can get dropped or lost)

Sender

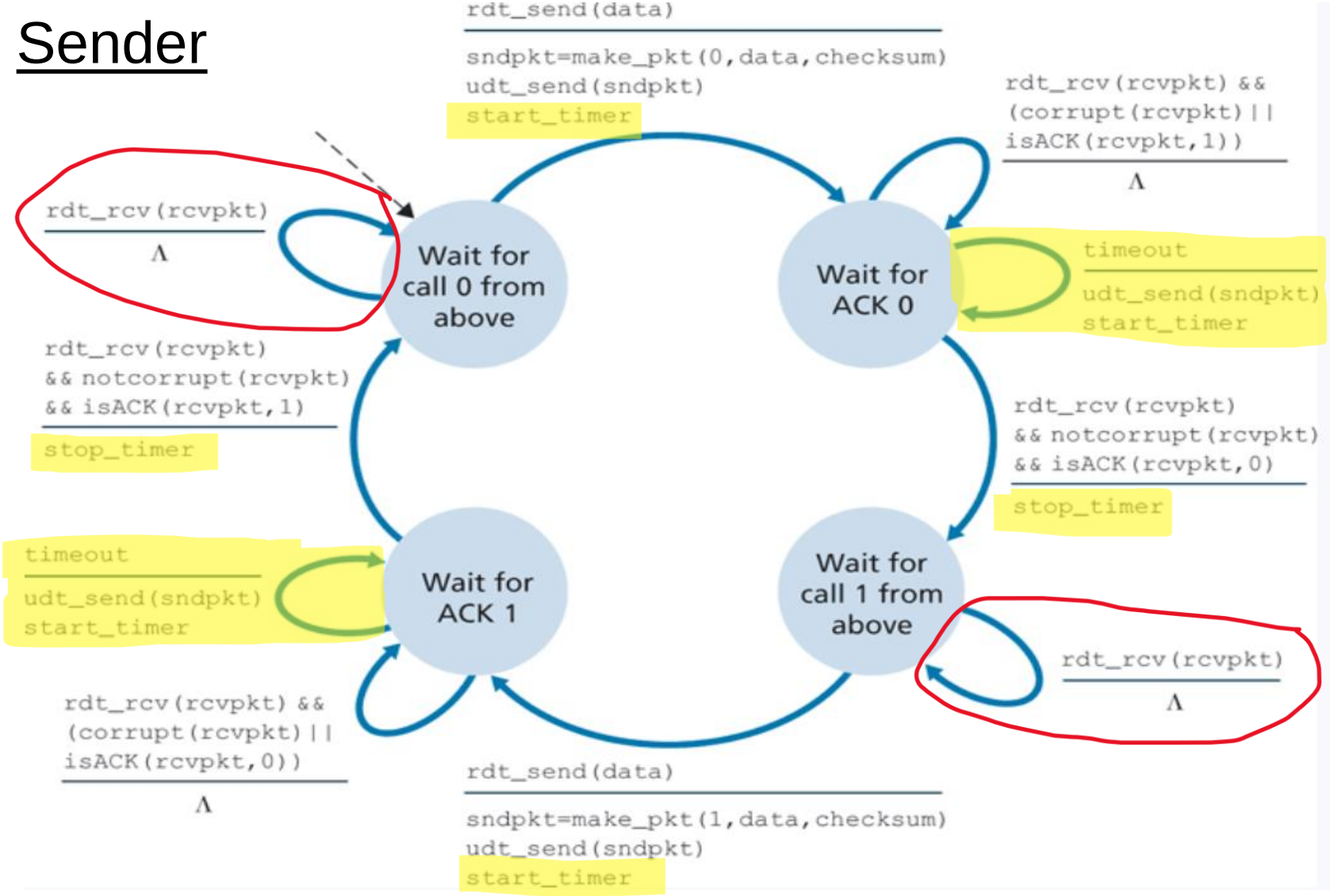


Receiver will be same as 2.2

Reliable Data Transfer 3.0

(Packets can get dropped or lost)

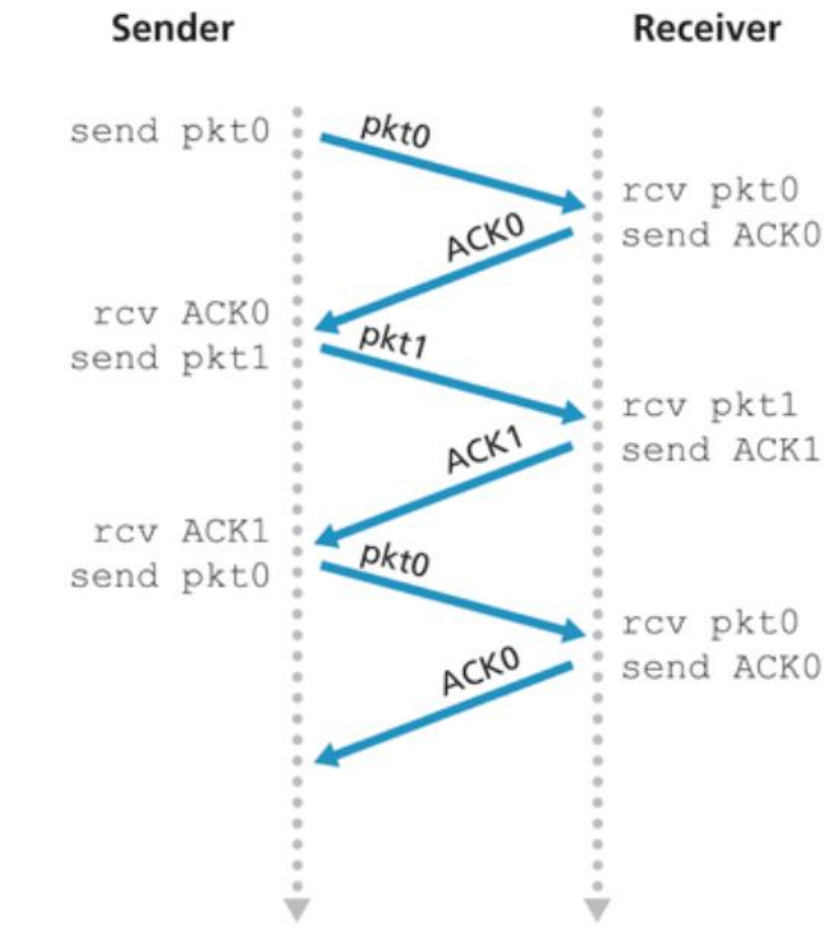
Sender



If we are waiting to send packet, but we receive a packet for some reason, this packet must be a duplicate packet, so do nothing

Reliable Data Transfer 3.0

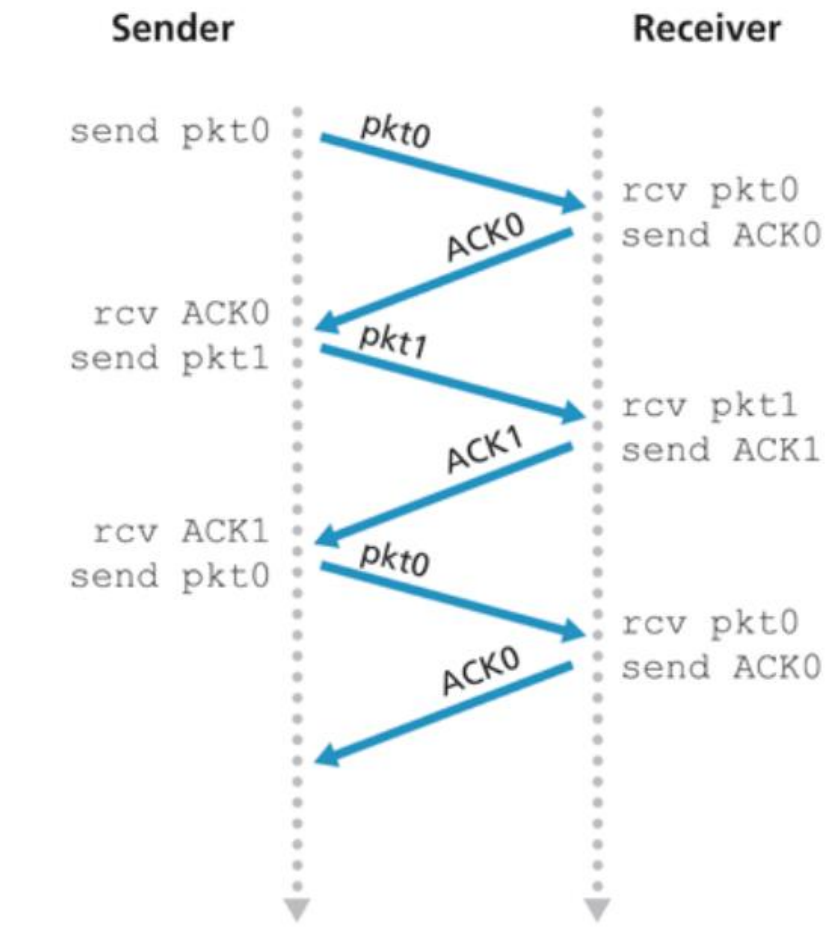
(Packets can get dropped or lost)



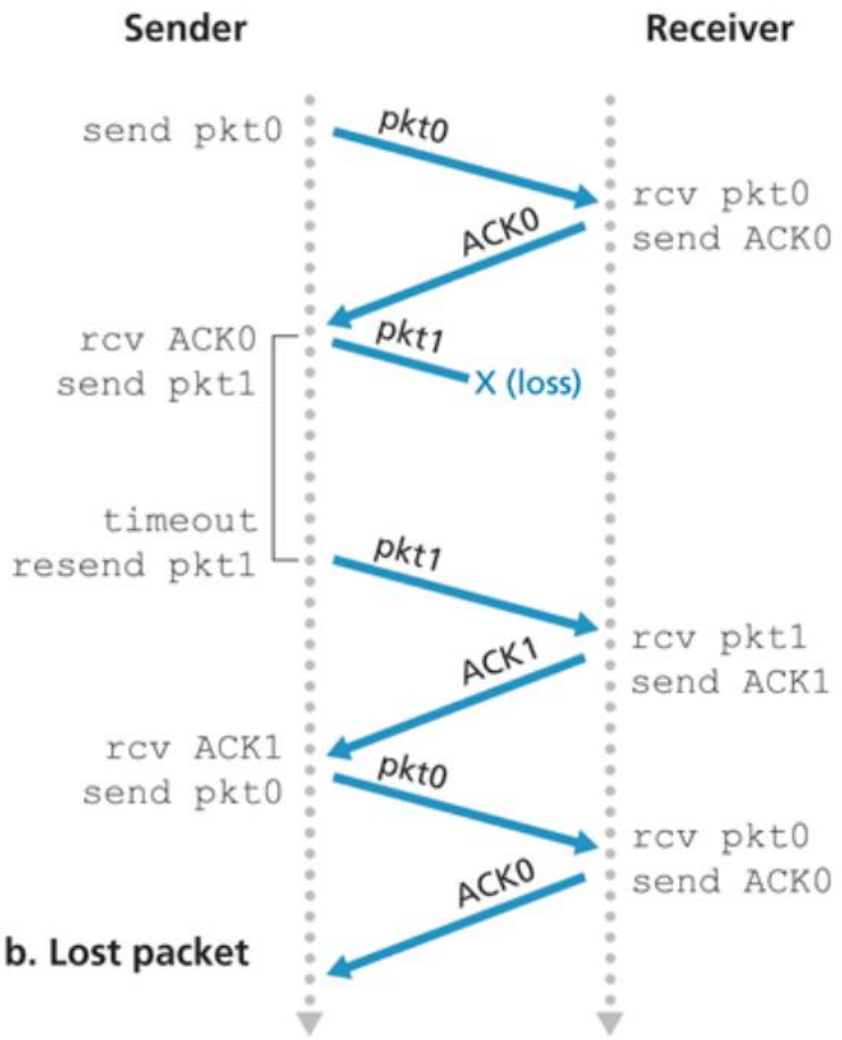
a. Operation with no loss

Reliable Data Transfer 3.0

(Packets can get dropped or lost)



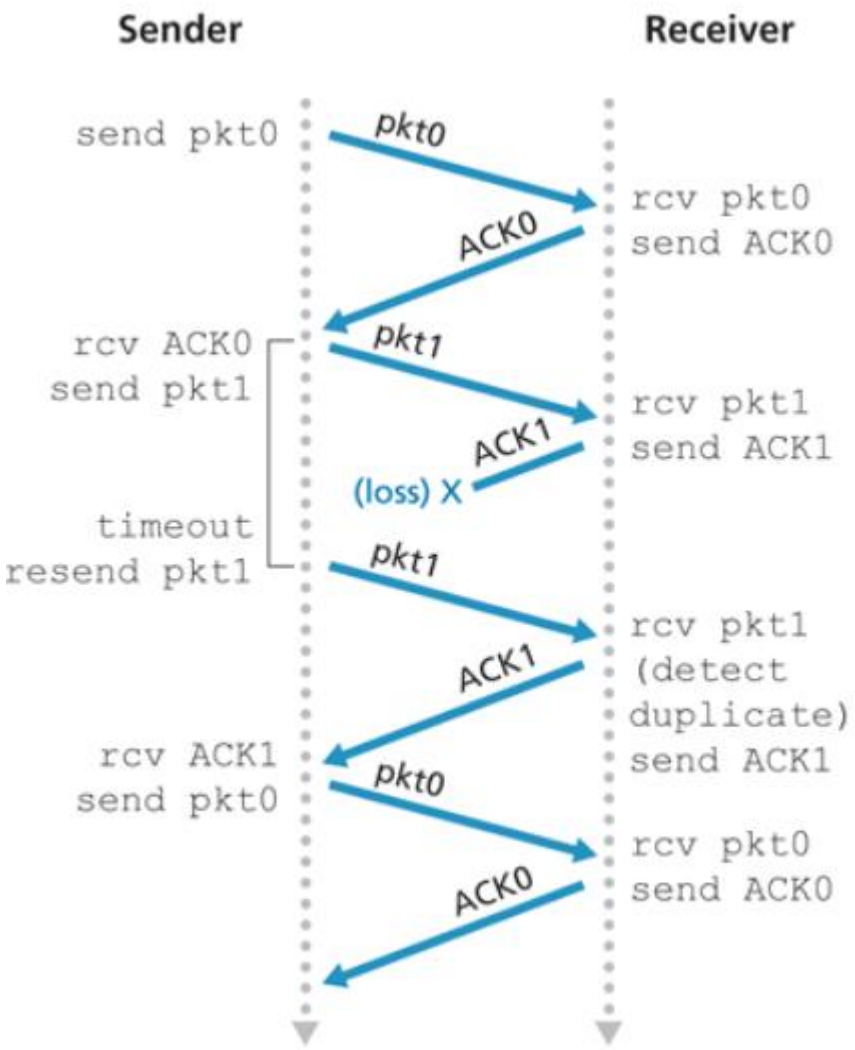
a. Operation with no loss



b. Lost packet

Reliable Data Transfer 3.0

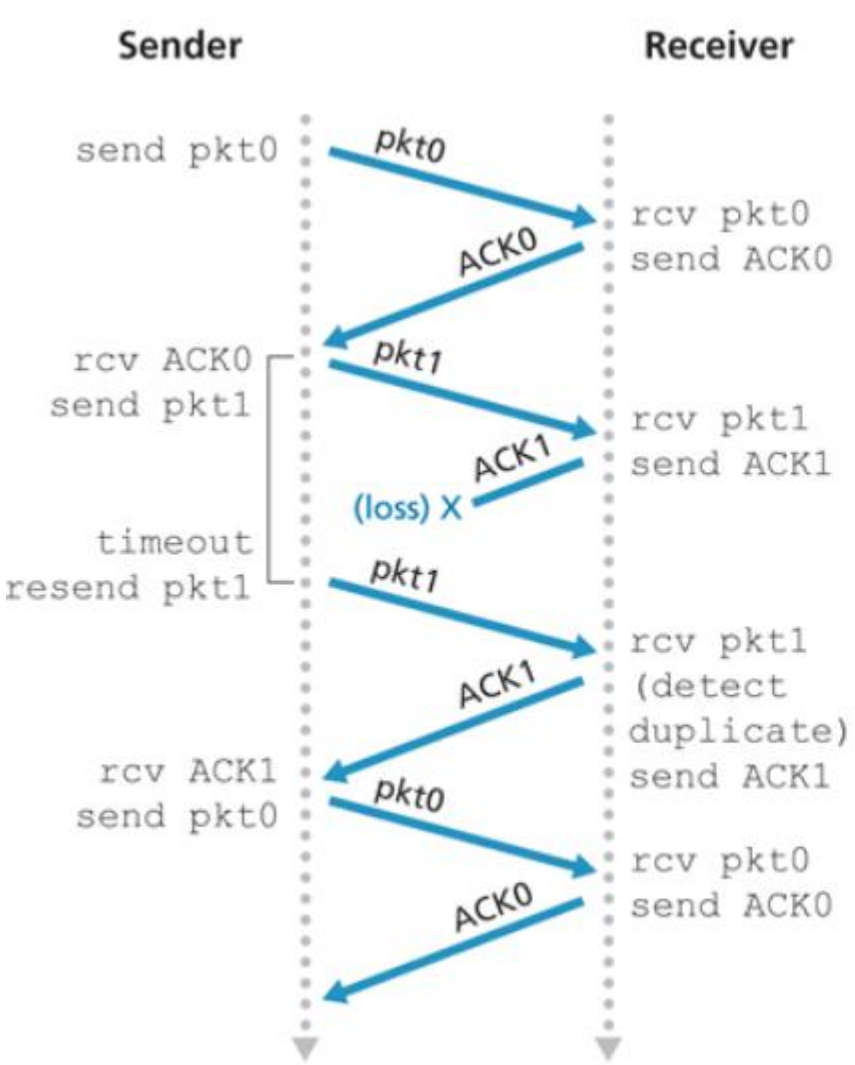
(Packets can get dropped or lost)



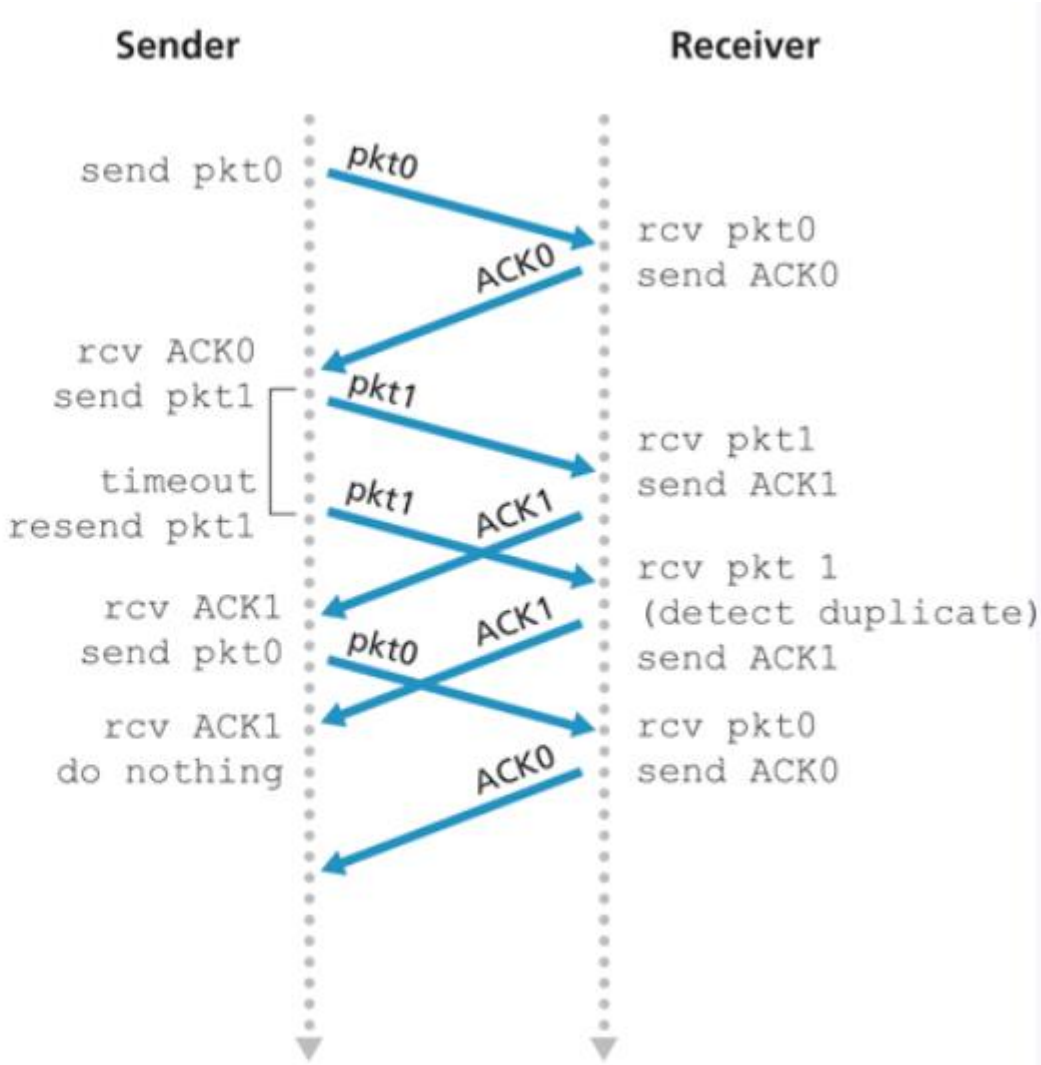
c. Lost ACK

Reliable Data Transfer 3.0

(Packets can get dropped or lost)



c. Lost ACK



d. Premature timeout