Computer Networks X_400487

Lecture 2

Chapter 3: The Data Link Layer Part 1

Lecturer: Jesse Donkervliet

VU VRIJE UNIVERSITEIT AMSTERDAM

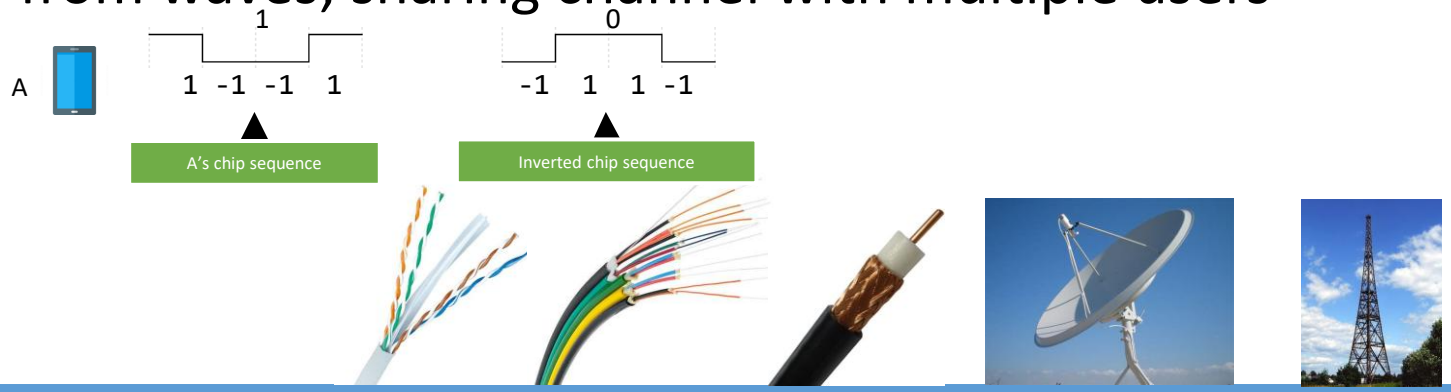
Recap of the Physical Layer

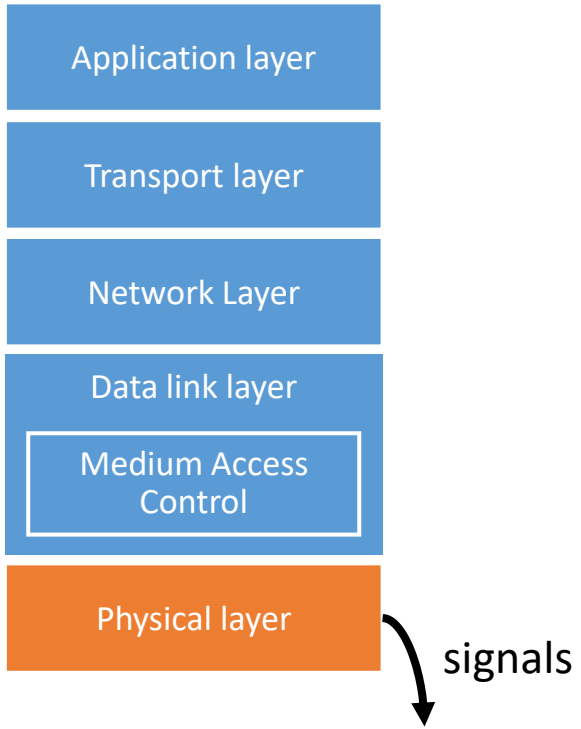
Responsible for transferring *bits* over a *wire-like* medium.

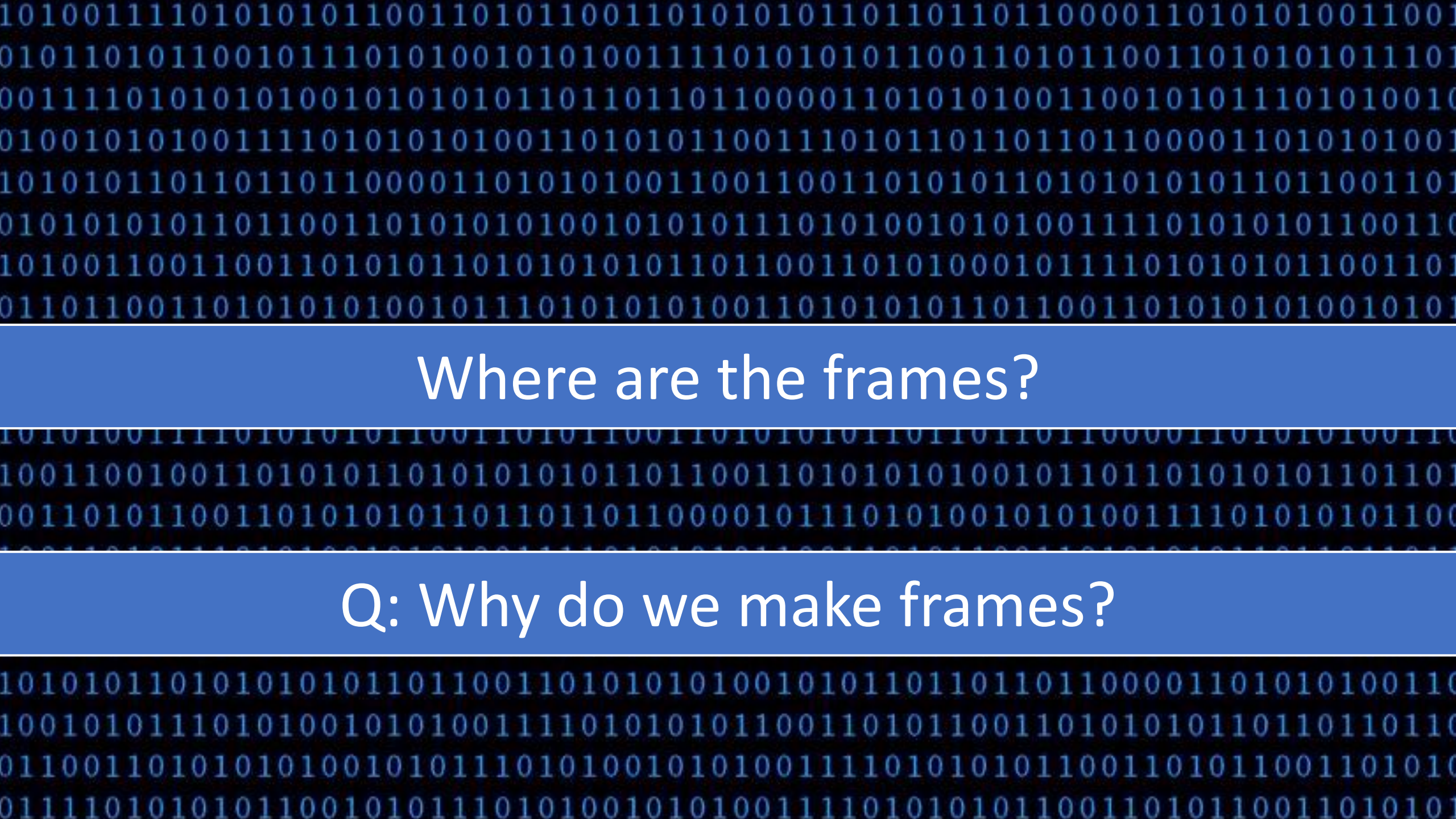
Maximum data rate determined by *bandwidth* and *signal-to-noise ratio.*

$$R = B \times \log_2\left(1 + \frac{S}{N}\right)$$

Physical layer responsible for translating bits to and from waves; sharing channel with multiple users







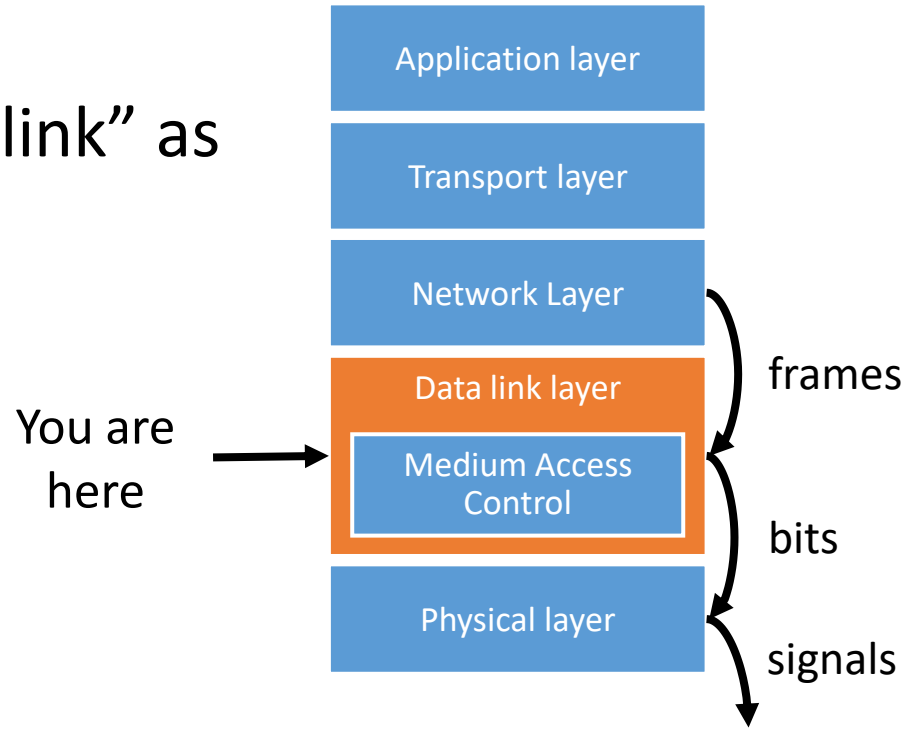
Where are the frames?

Q: Why do we make frames?

The Data Link layer

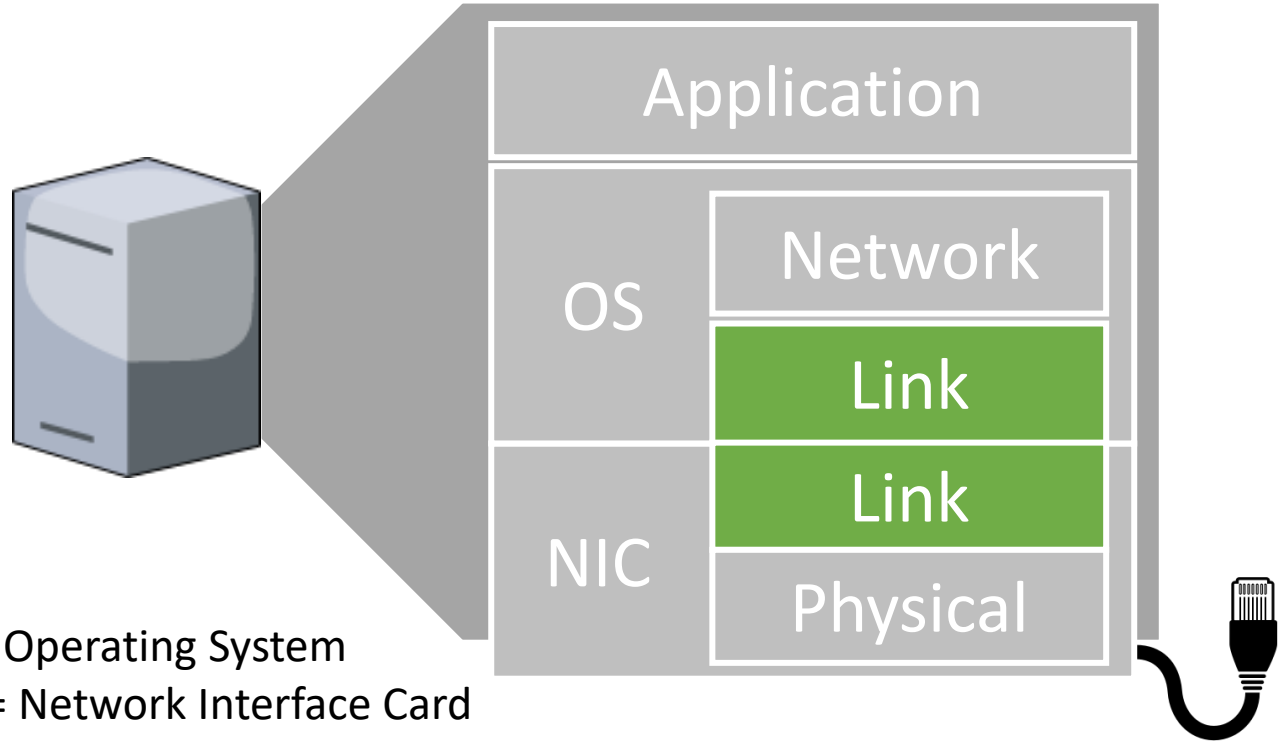
Responsible for transferring *frames* over a single link

1. Groups bits into frames
2. Offers “sending frames over a link” as a *service* to the network layer
3. Handles transmission errors
4. Regulates data flow



Link layer environment

Commonly implemented as NICs and OS drivers; network layer (IP) is often OS software.



OS = Operating System
NIC = Network Interface Card

Data Link Layer — Roadmap

Part 1

1. **Framing → From single bits to discrete data units: *frames***
2. **Flow Control → Prevent overloading the receiver**
3. **Guaranteed Delivery → Because the network can drop frames**
4. **Sliding Window Protocols → Using network resources efficiently**

Part 2

1. Error detection
2. Error correction

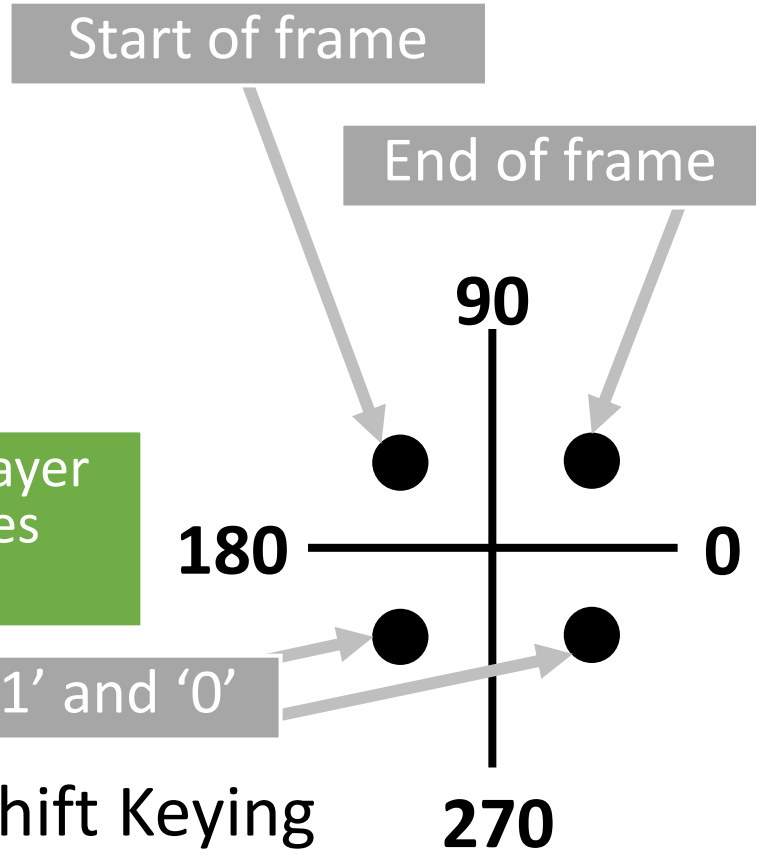
Framing

From Bit Stream to Discrete Units of Information

Framing Methods

1. Byte count.
2. Flag bytes with byte stuffing.
3. Flag bits with bit stuffing.
4. Use special symbols in physical layer.

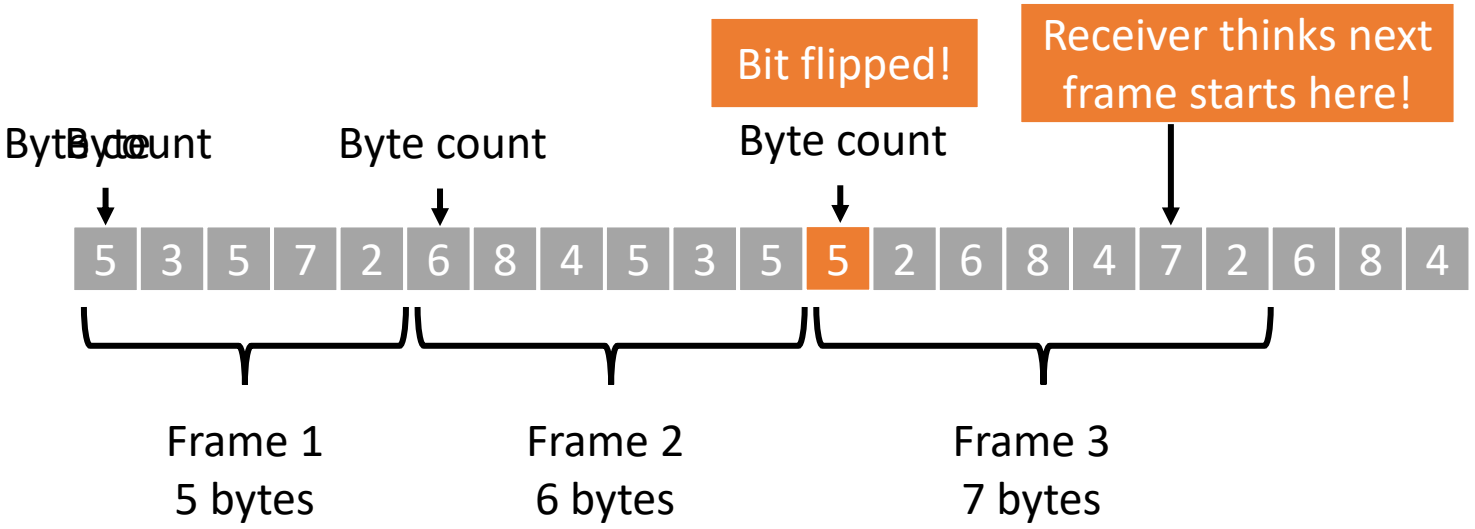
‘Cheating’ because physical layer does not know about frames (according to *our* model)



Example of method 4 using Phase Shift Keying

Framing
Byte count

Q: Advantage?
Disadvantage?

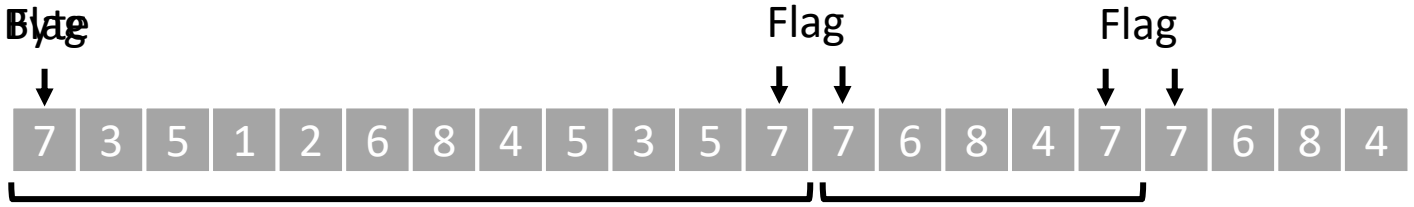


Framing
Byte stuffing

Q: Disadvantage?

Use a ‘flag’ byte to indicate start and end of frame.

Let’s say our flag byte is 00000111_2 (7_{10}).



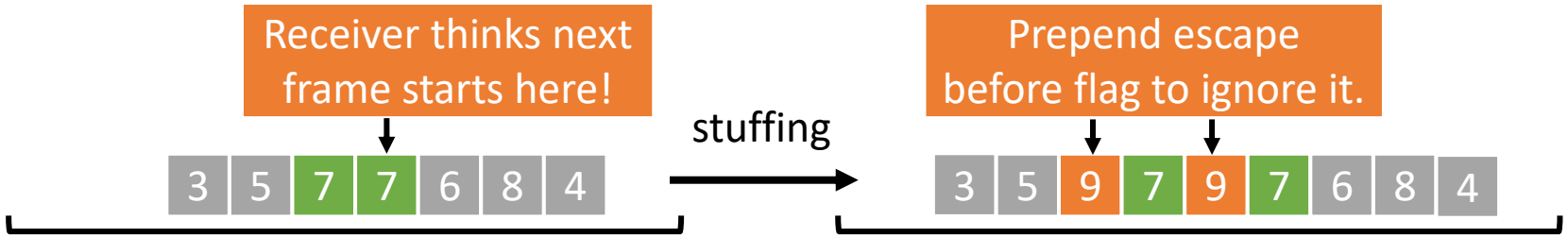
Framing
Byte stuffing

Character	Escape Character
%	%25

What if the data contains a flag byte?

Use an ‘escape’ byte to ignore certain flag bytes.

Let’s say our escape byte is 00001001_2 (9_{10}).



Q: Algorithm on the receiving side?

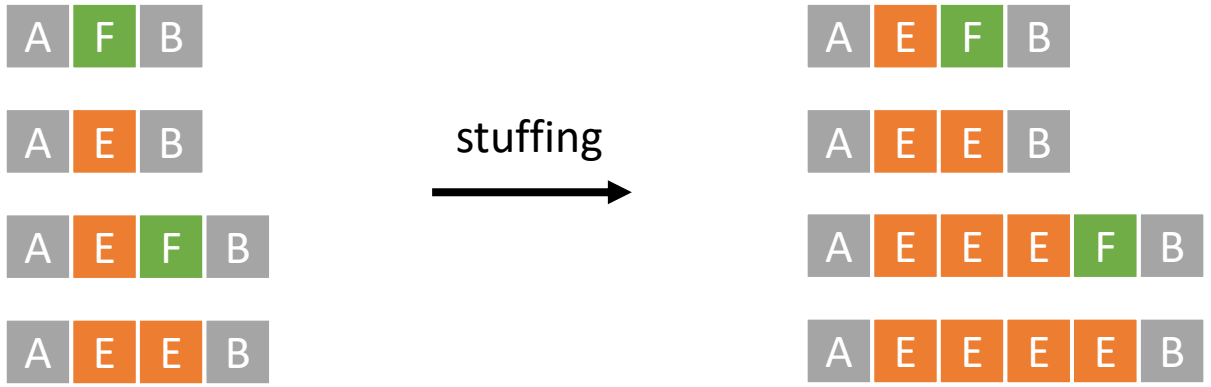
Q: Are we done?

Framing
Byte stuffing

Q: What is the overhead of this approach?

Escape bytes can also occur in data!

Let's use letters for generality.
Flag byte = F, Escape byte = E.



Escape both 'escape' and 'flag' bytes.

Framing
Bit stuffing

Byte stuffing can be space inefficient.

Byte stuffing		**Bit stuffing**
Flag byte	→	Bit pattern
Escape byte	→	Insert single bit

Example:

Bit pattern: `01111110`

Insert bit if pattern in data: `011111**0**10`

Add one extra bit

Bit stuffing example Receiver

Bit pattern: 01111110

```
1100010100011111011110001110011011000000110111000111110100010110111
0100011010100010010100111111010000111001000001100100101000000101111
0101110000111111010011000111001111110011001100000111011111001111001010
0001110111100001110111000110010110011001100011111100100111001101011
0101001110000111111011010100000100011111001101011100110011000010100
0101010000011010010111011010100101001001111100010010100100000010010
1101111101001100100111000011101111101111000010100110100011111110011
1110000000100100001001011001110101101100000011000110101100000000011
0101101001100001100000111110110010010100110011011011001100110110110
0
0101000111101011101000111111011001010100000101001101110011010110100
0
0110101111100011100010000010110101010100101111100101100010010010000
0110101110111011100000100011001000000011111011100100101110110100
10
0010111110010101001011100011001010011001000100011010100100101010011
0111001111000111101000101001010011111000111110001001111101100010000
1010001101010111001011100010010011110101101001111100001111101101001
1011001011111010001010110000111111000111
```

Bit stuffing example Receiver

Q: Are we done?

Bit pattern: 01111110

F1:

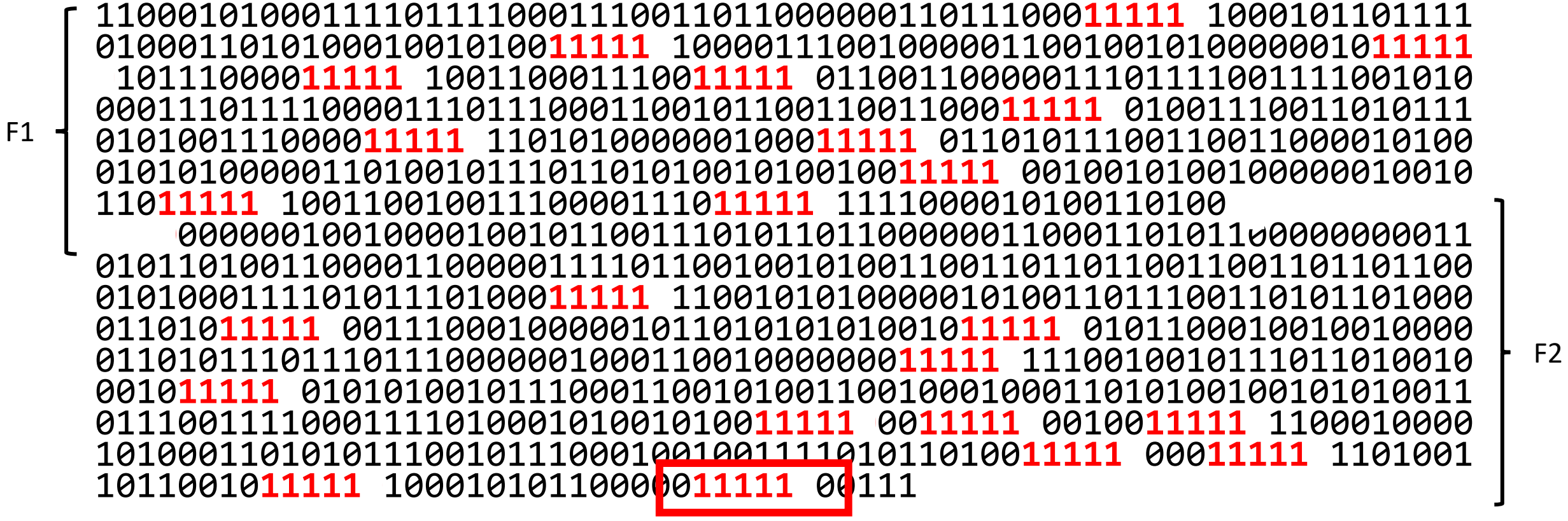
```
1100010100011110111100011100110110000001101110001111101000101101111
0100011010100010010100111110100001110010000011001001010000001011111
0101110000111110100110001110011111001100110000011101111001111001010
0001110111100001110111000110010110011001100011111001001110011010111
01010011100001111101101010000000100011111001101011100110011000010100
010101000000110100101110110101001010010011111000100101001000000010010
11011111010011001001110000111011111101111000010100110100
```

F2:

```
0000001001000010010110011101011011000000110001101011000000000011
0101101001100001100000111101100100101001100110110110011001101101100
01010001111010111010001111101100101010000001010011011100110101101000
01101011111000111000100000010110101010100101111100101100010010010000
01101011101110111000000010001100100000001111101110010010111011010010
0010111110010101001011100011001010011001000100011010100100101010011
0111001111000111101000101001010011111000111110001001111101100010000
1010001101010111001011100010010011110101101001111100001111101101001
1011001011111010001010110000001111000111
```

Bit stuffing example Receiver

Bit pattern: 01111110



Bit pattern in data

Bit stuffing example Receiver

Q: What is the overhead of this approach?

Bit pattern: 01111110

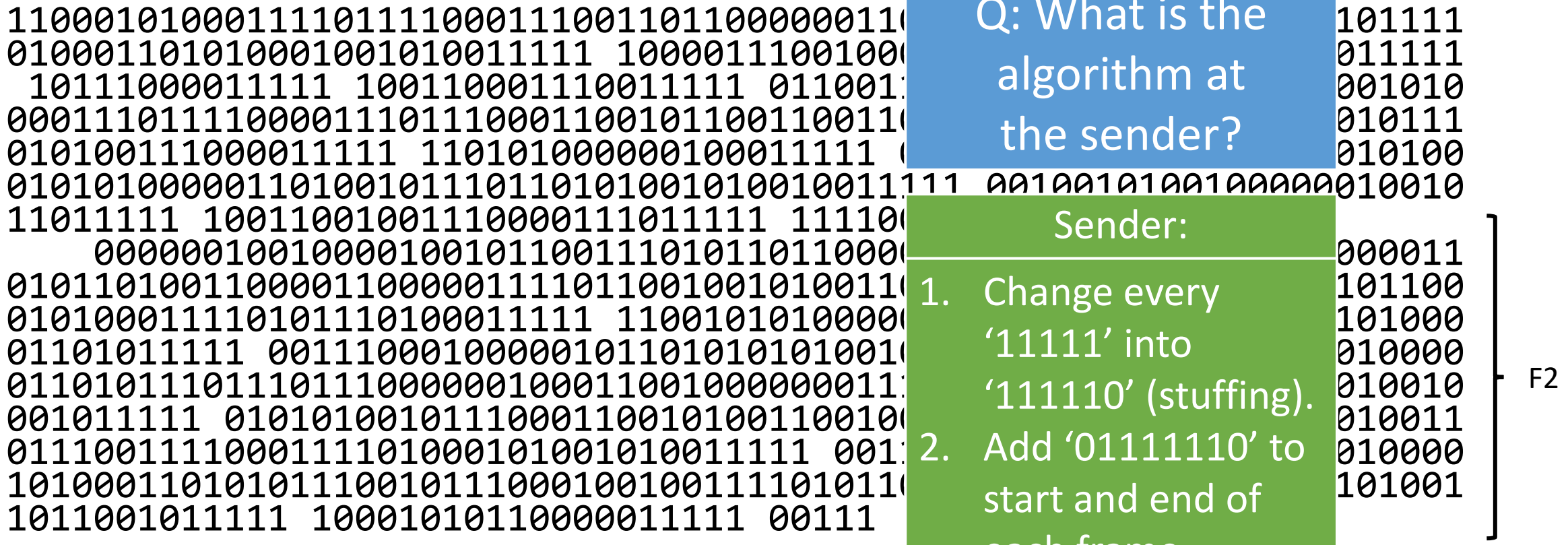
Q: What is the algorithm at the sender?

Sender:

1. Change every ‘11111’ into ‘111110’ (stuffing).
2. Add ‘01111110’ to start and end of each frame.

F1

F2



Data Link Layer — Roadmap

Part 1

1. Framing → From single bits to discrete data units: *frames*
2. **Flow Control → Prevent overloading the receiver**
3. Guaranteed Delivery → Because the network can drop frames
4. Sliding Window Protocols → Using network resources efficiently

Part 2

1. Error detection
2. Error correction

Flow Control

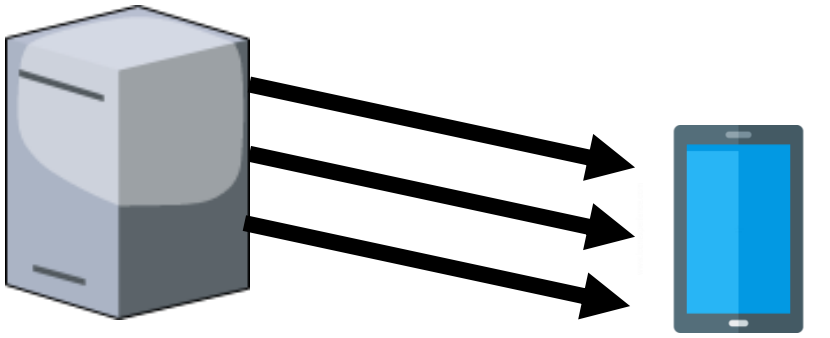
Could you speak more slowly, please?

A *Resource Management* problem

Utopian simplex protocol
The ideal case

…

```
while True:
        packet = from_network_layer()
        frame.payload = packet
        to_physical_layer(frame)
```

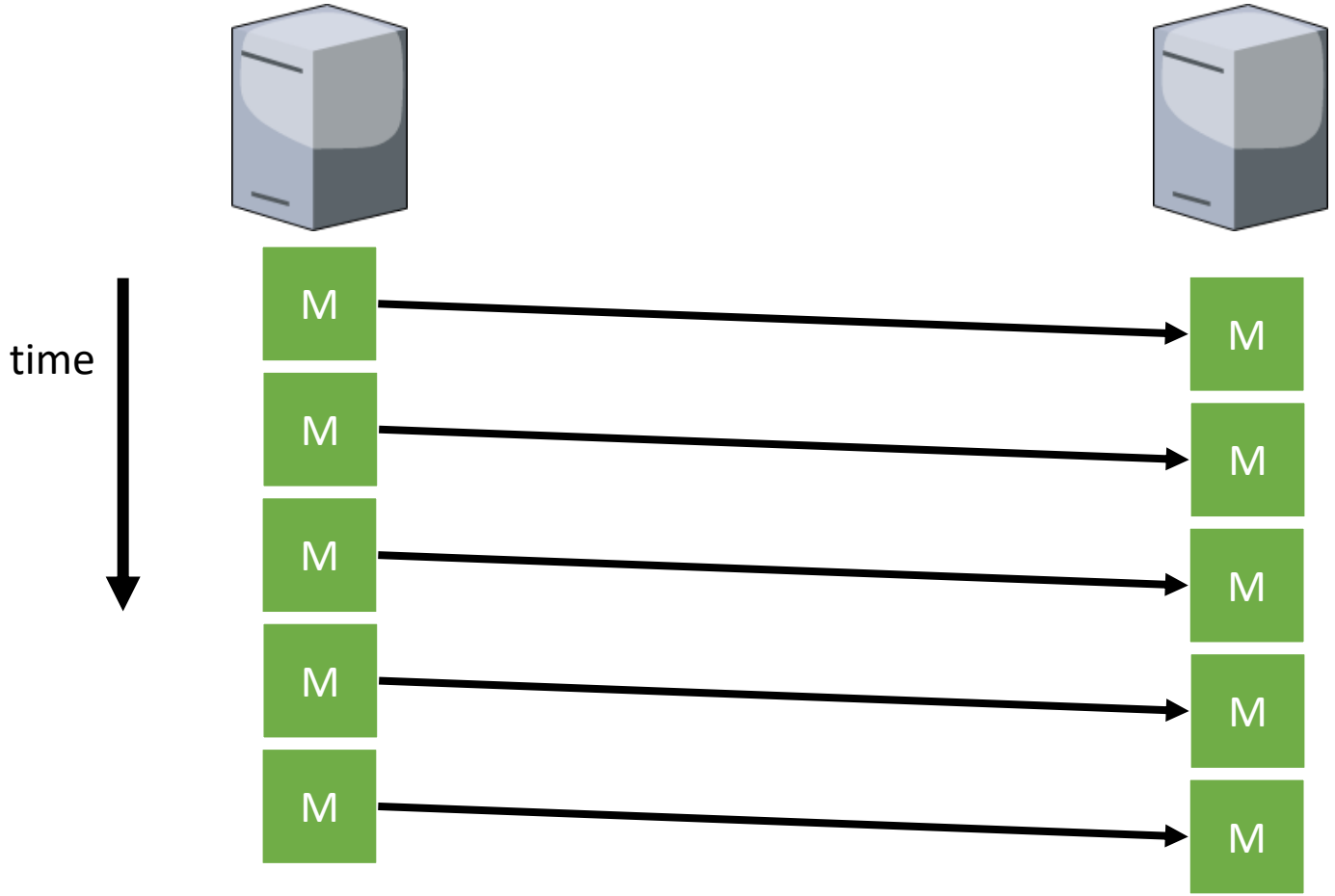


Stop-and-wait for error-free channel

…

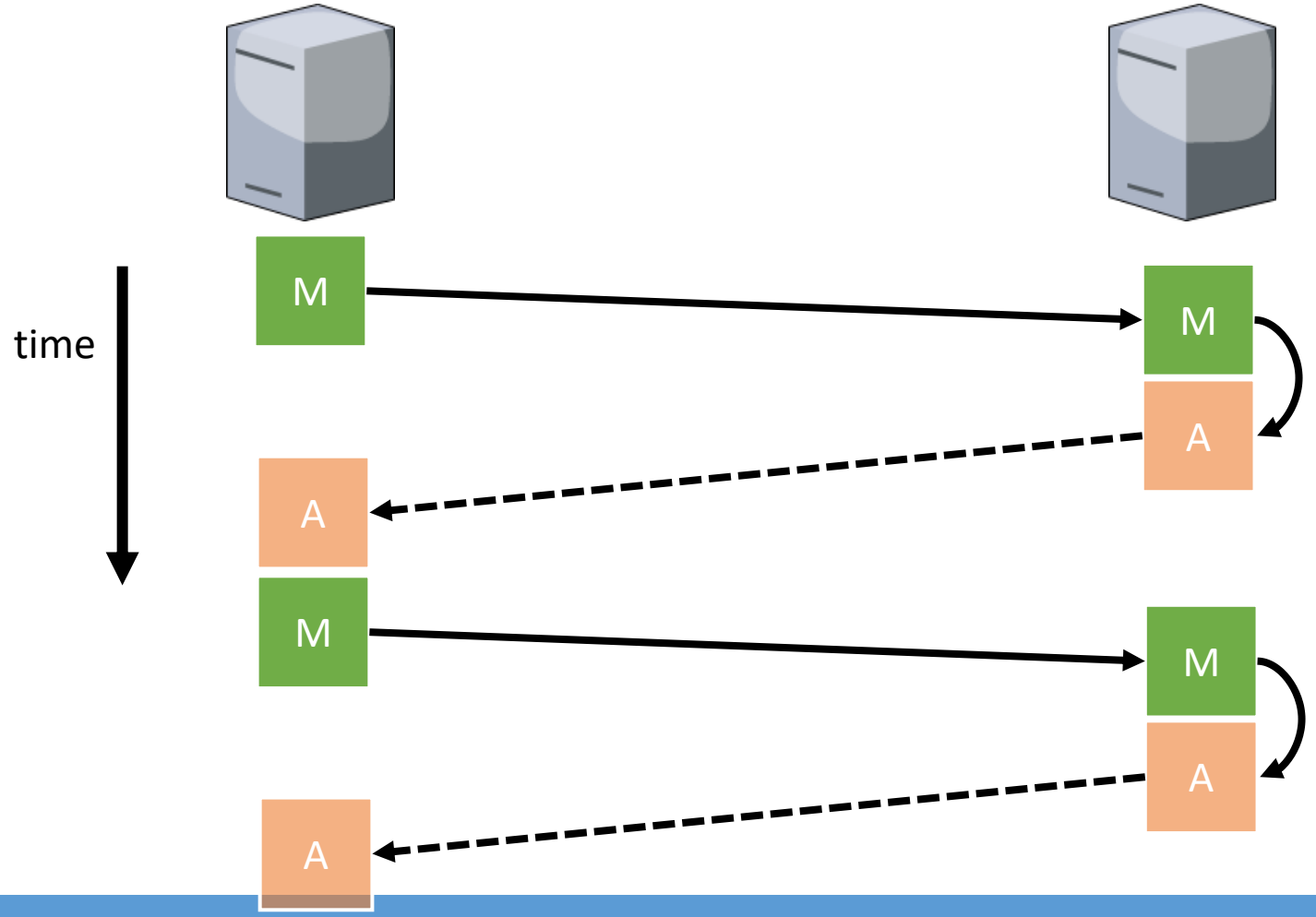
```
while True:
        packet = from_network_layer()
        frame.payload = packet
        to_physical_layer(frame)
        event = wait_for_event()
```

Example of Utopian Simplex Protocol



Example of Stop-and-Wait Protocol

Q: What if a frame gets lost?



Data Link Layer — Roadmap

Part 1

1. Framing → From single bits to discrete data units: *frames*
2. Flow Control → Prevent overloading the receiver
3. **Guaranteed Delivery → Because the network can drop frames**
4. Sliding Window Protocols → Using network resources efficiently

Part 2

1. Error detection
2. Error correction

Guaranteed Delivery

Acknowledgments, Sequence Numbers, and Retransmissions

How Can We Know If a Frame Gets Lost?

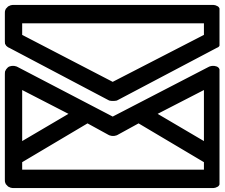
Ask a different question

Q: How can we know if a frame *arrives*?

Send a message back: "I got your message!"

Q: When do we want to retransmit data?

Q: What if the acknowledgment gets lost?



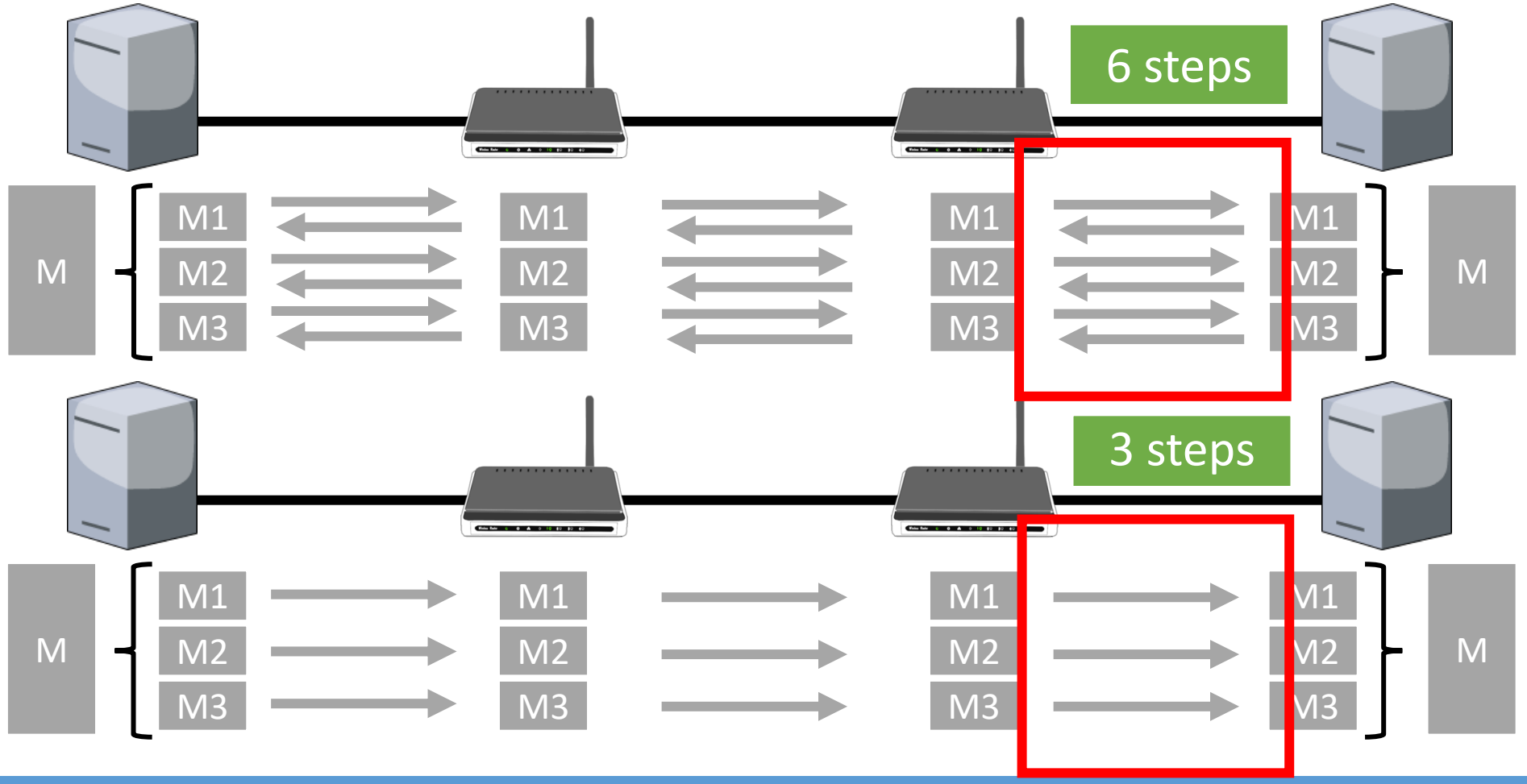
We assume our original message did not arrive

It depends on … the application!

Acknowledgments let the sender know it does ***not*** need to retransmit data.

Many protocols either use or don't use acknowledgments. Different approach: Support acknowledgments, but let the application decide if it needs to use acknowledgments or not.

To acknowledge, or not to acknowledge

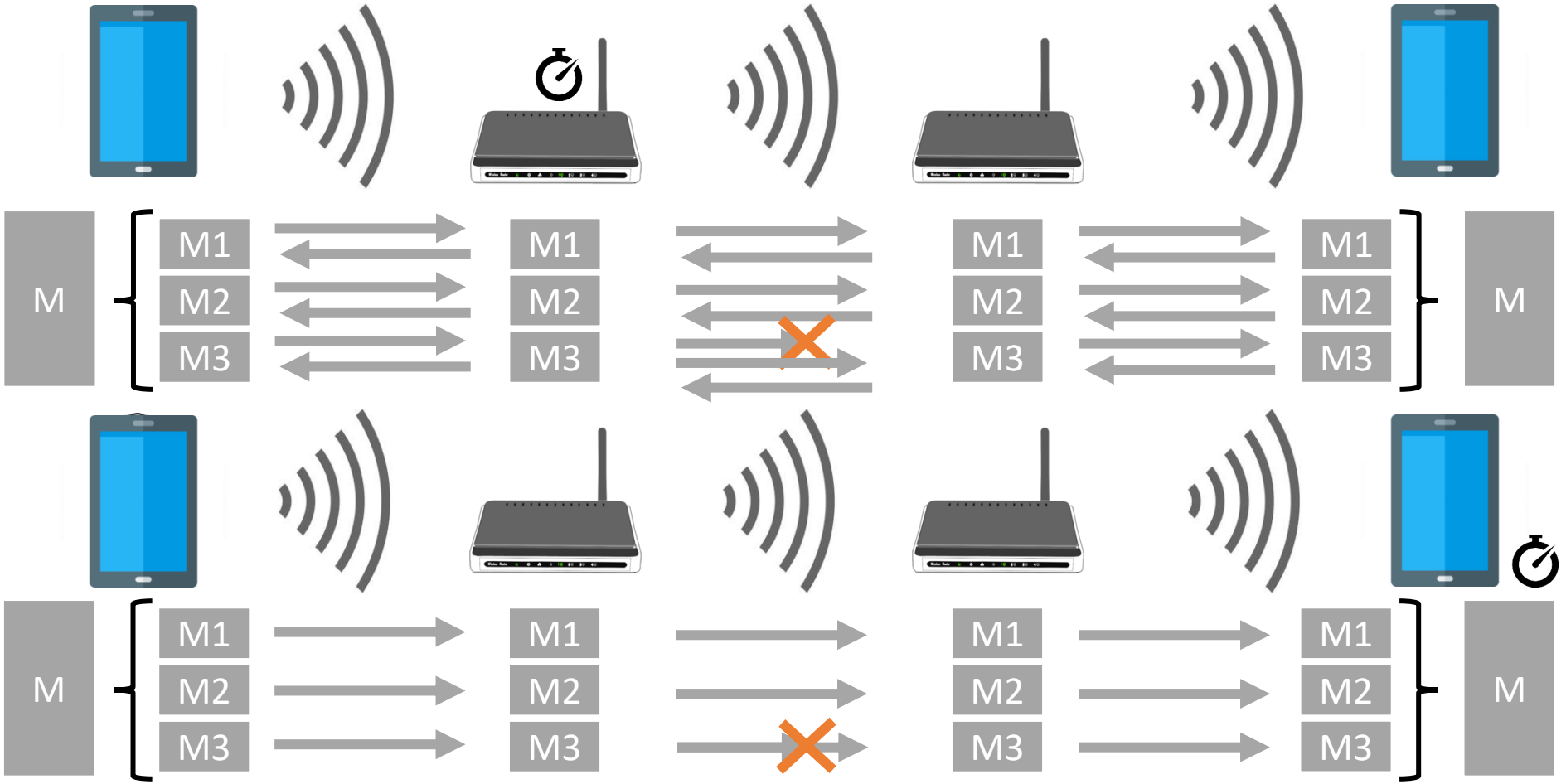


To acknowledge, or not to acknowledge

It depends on …
the physical medium!

It depends on …
the application!

This problem will return later
(in the transport layer)

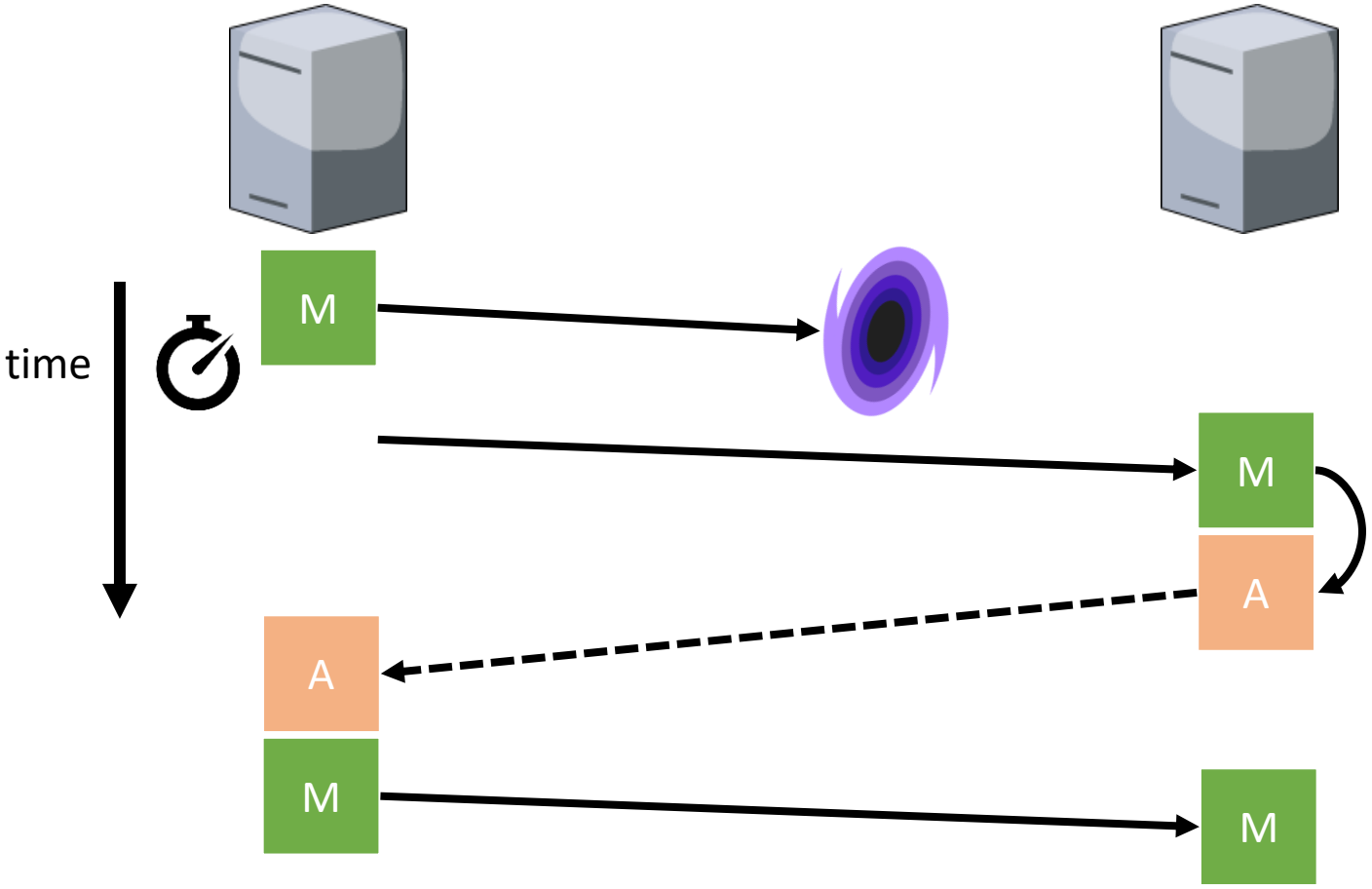


Automatic Repeat reQuest (ARQ) Guaranteed Delivery over Unreliable Channel

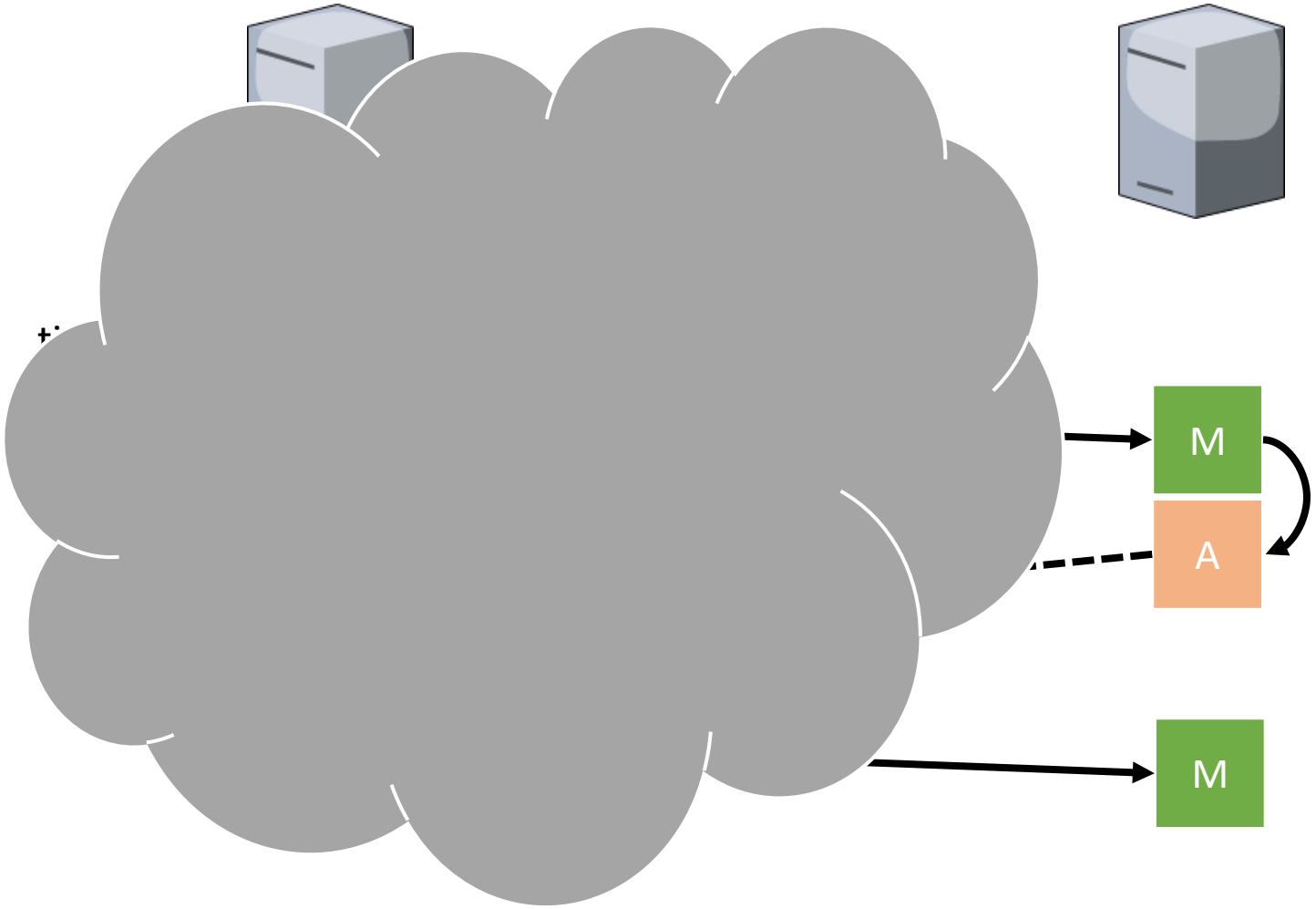
Same as stop-and-wait, except:

1. Keep track of frames using sequence numbers.
2. Wait until previous frame has been accepted.

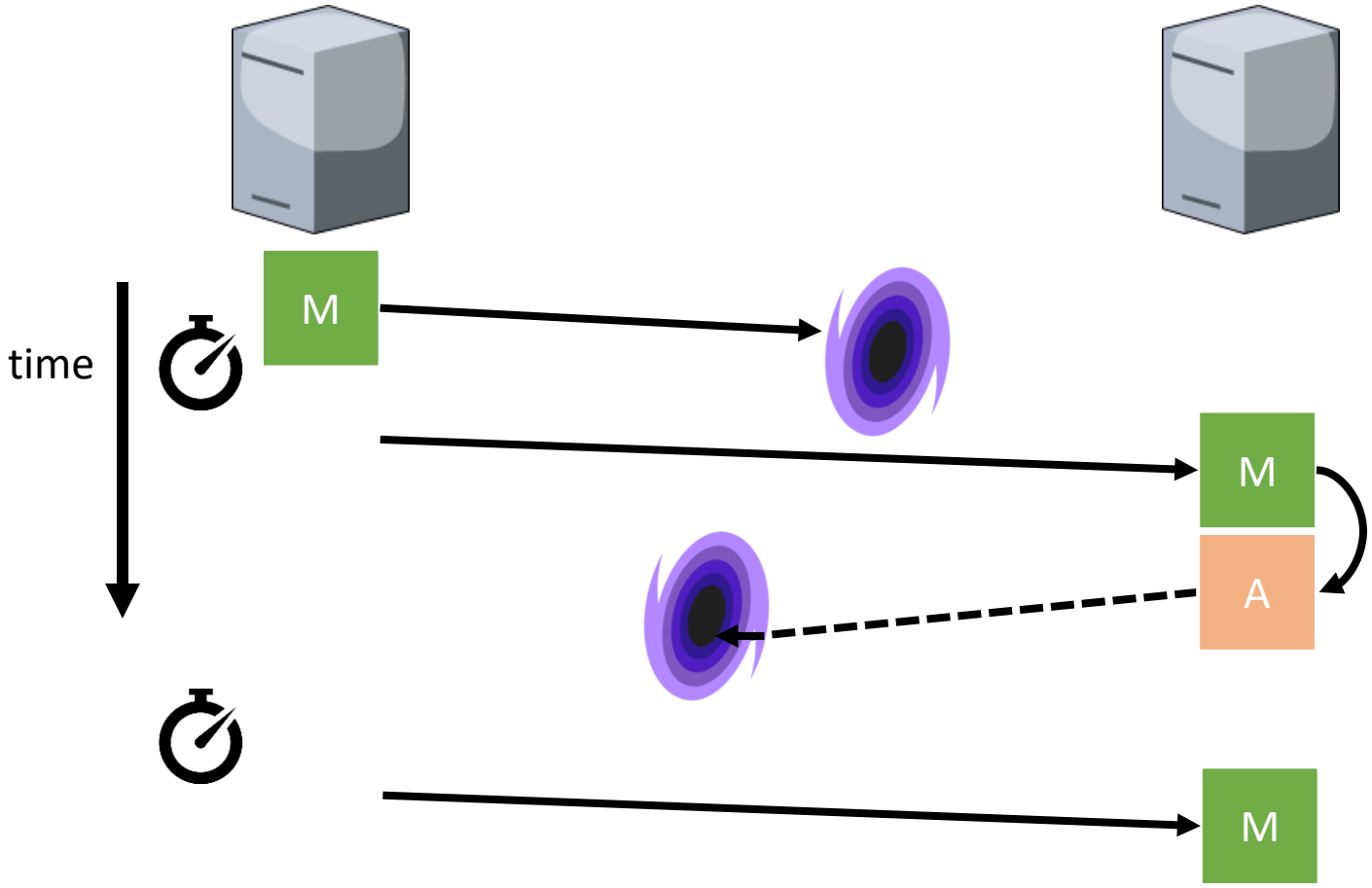
ARQ Example



ARQ Example

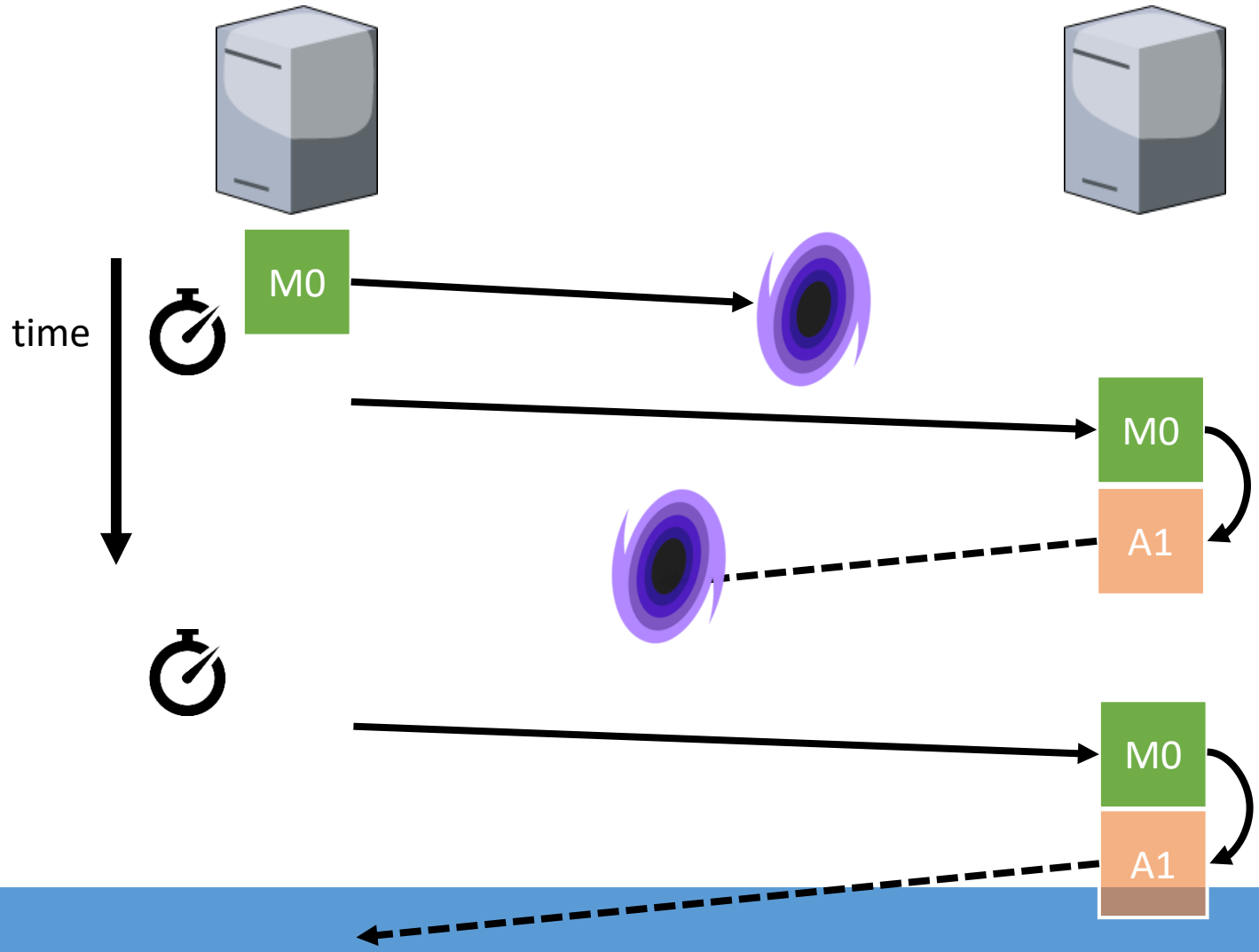


ARQ Example



Sequence numbers needed to differentiate between retransmission and next frame

ARQ Example



Sequence numbers needed to differentiate between retransmission and next frame

Automatic Repeat ReQuest (ARQ) Guaranteed Delivery over Unreliable Channel

ARQ adds error control

Receiver acks frames that are correctly delivered.

Sender sets timer and resends frame if no ack.

Q: How long should we wait?

Q: What can go wrong?

Frames and acks must be identifiable (e.g., with sequence number)

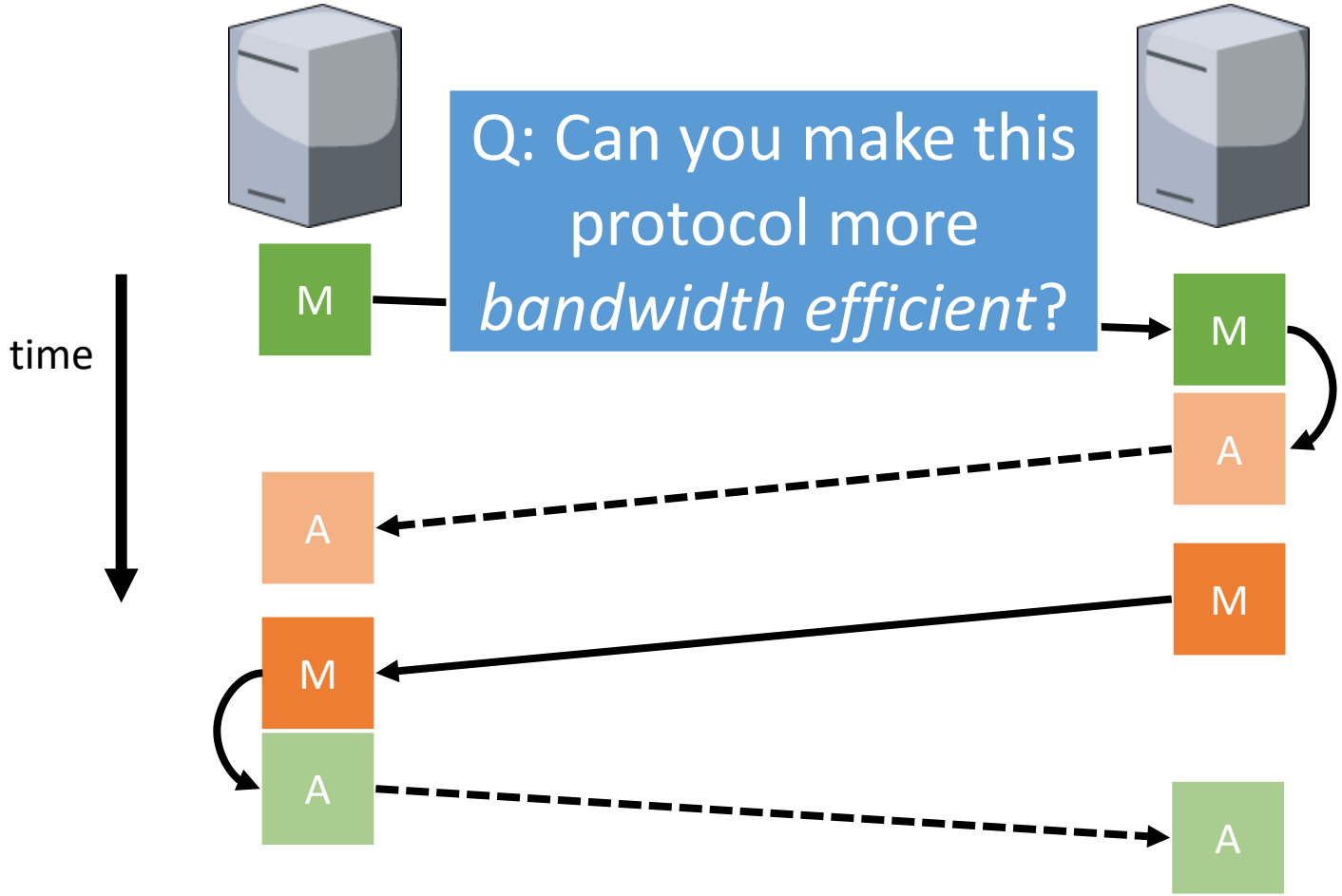
Else receiver cannot tell retransmission (due to lost ack or early timer) from new frame.

For stop-and-wait, 2 numbers (1 bit) are sufficient.

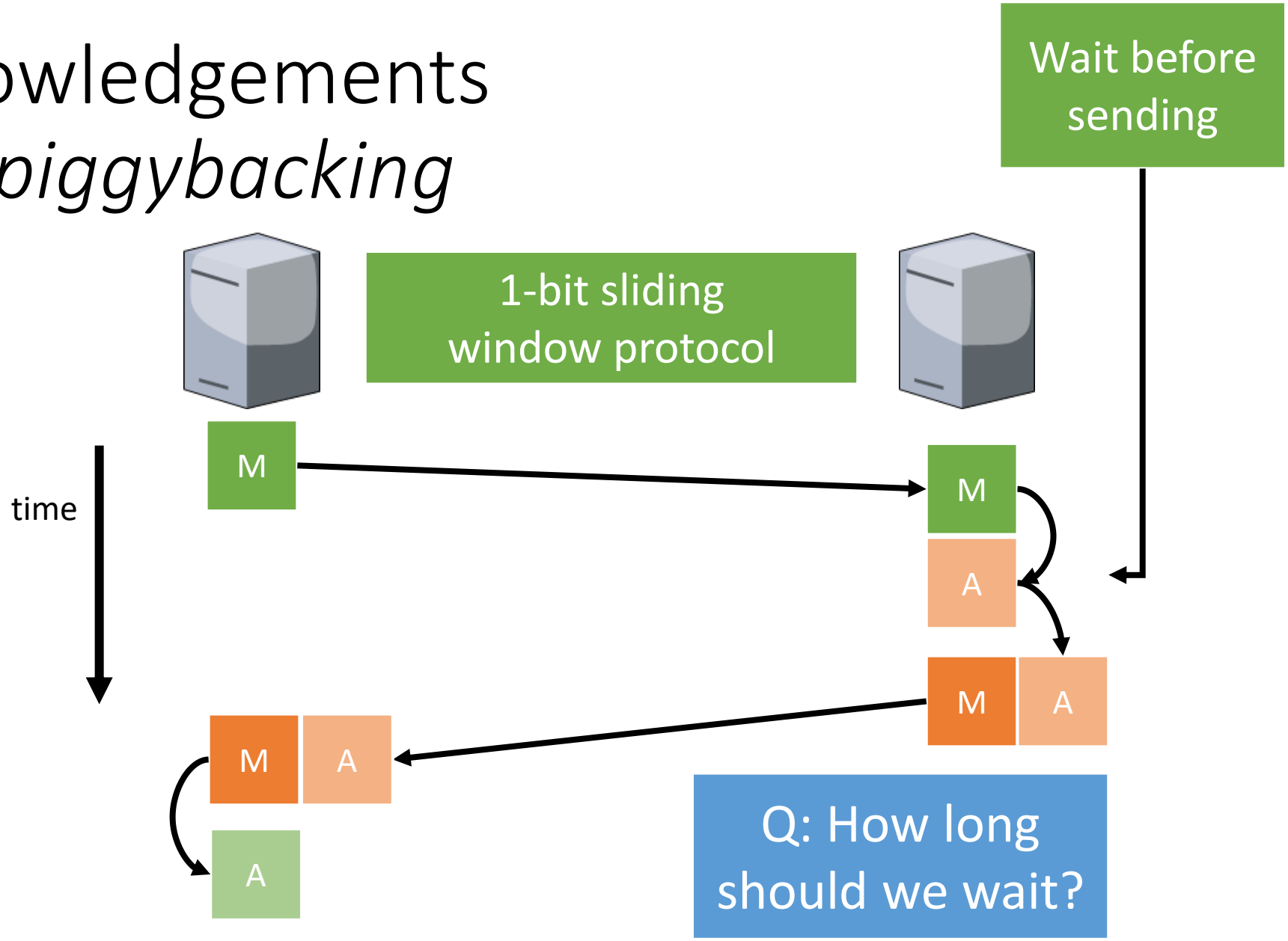
Q: Why sufficient?

Acknowledgements

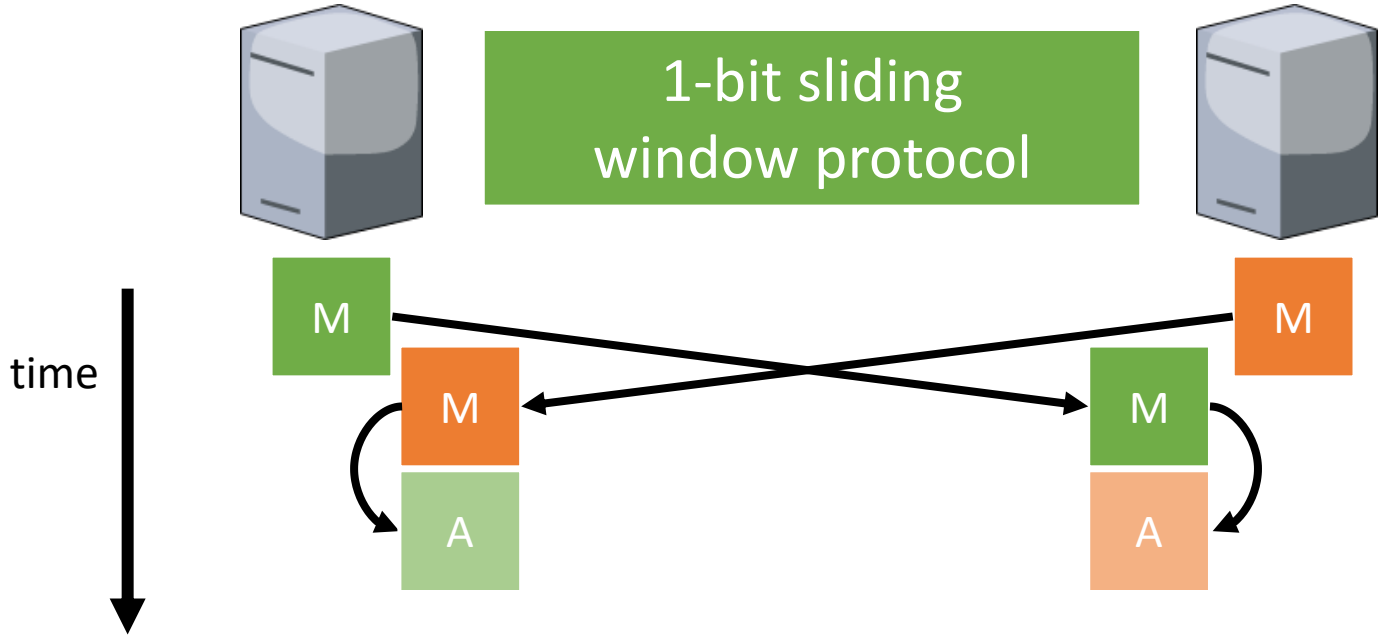
Bidirectional communication



Acknowledgements With *piggybacking*



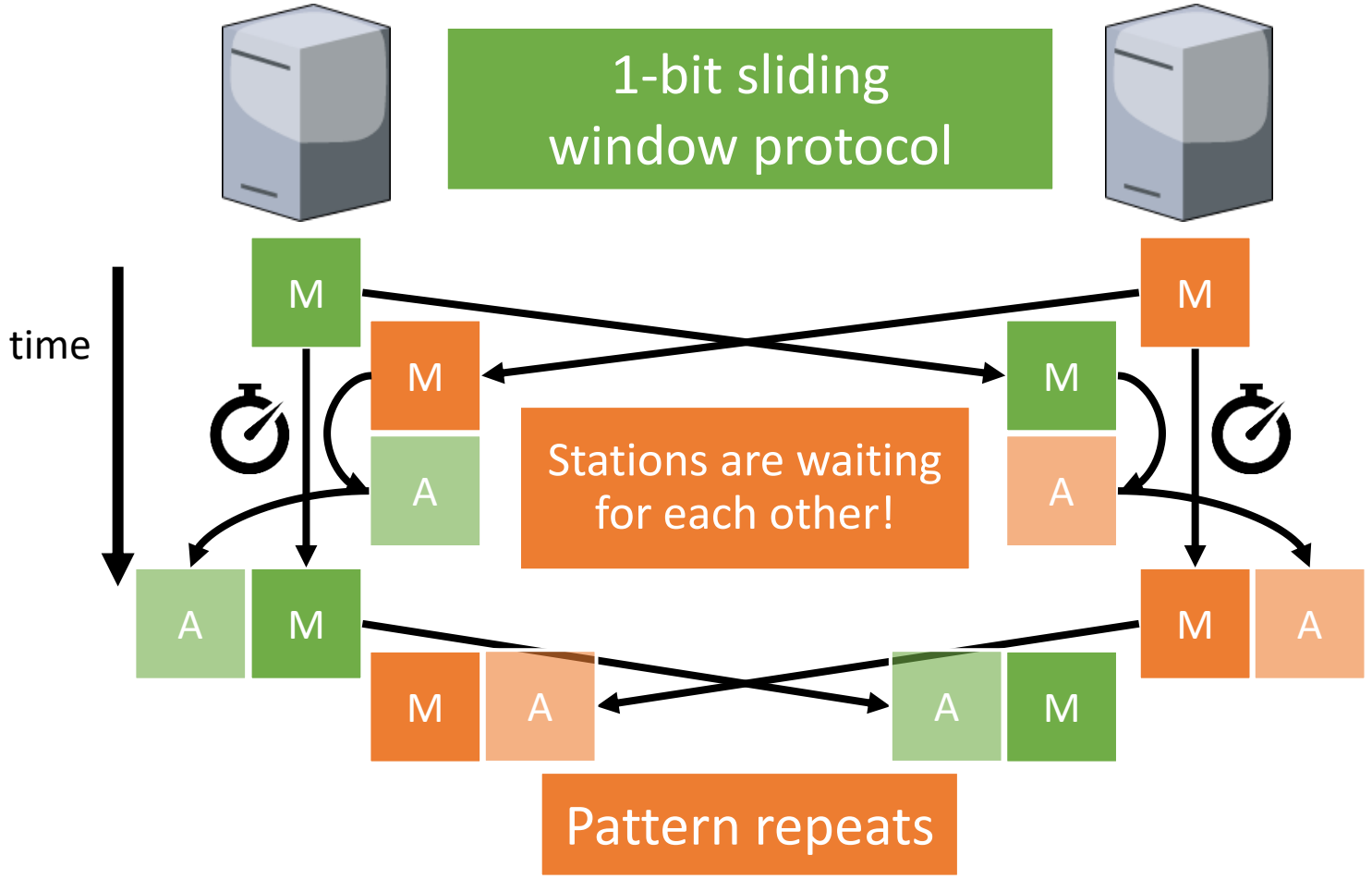
Stop-and-Wait



Q: What happens now?

Stop-and-Wait Special Case

Even without errors, half the bandwidth is wasted on retransmissions



Data Link Layer — Roadmap

Part 1

1. Framing → From single bits to discrete data units: *frames*
2. Flow Control → Prevent overloading the receiver
3. Guaranteed Delivery → Because the network can drop frames
4. **Sliding Window Protocols → Using network resources efficiently**

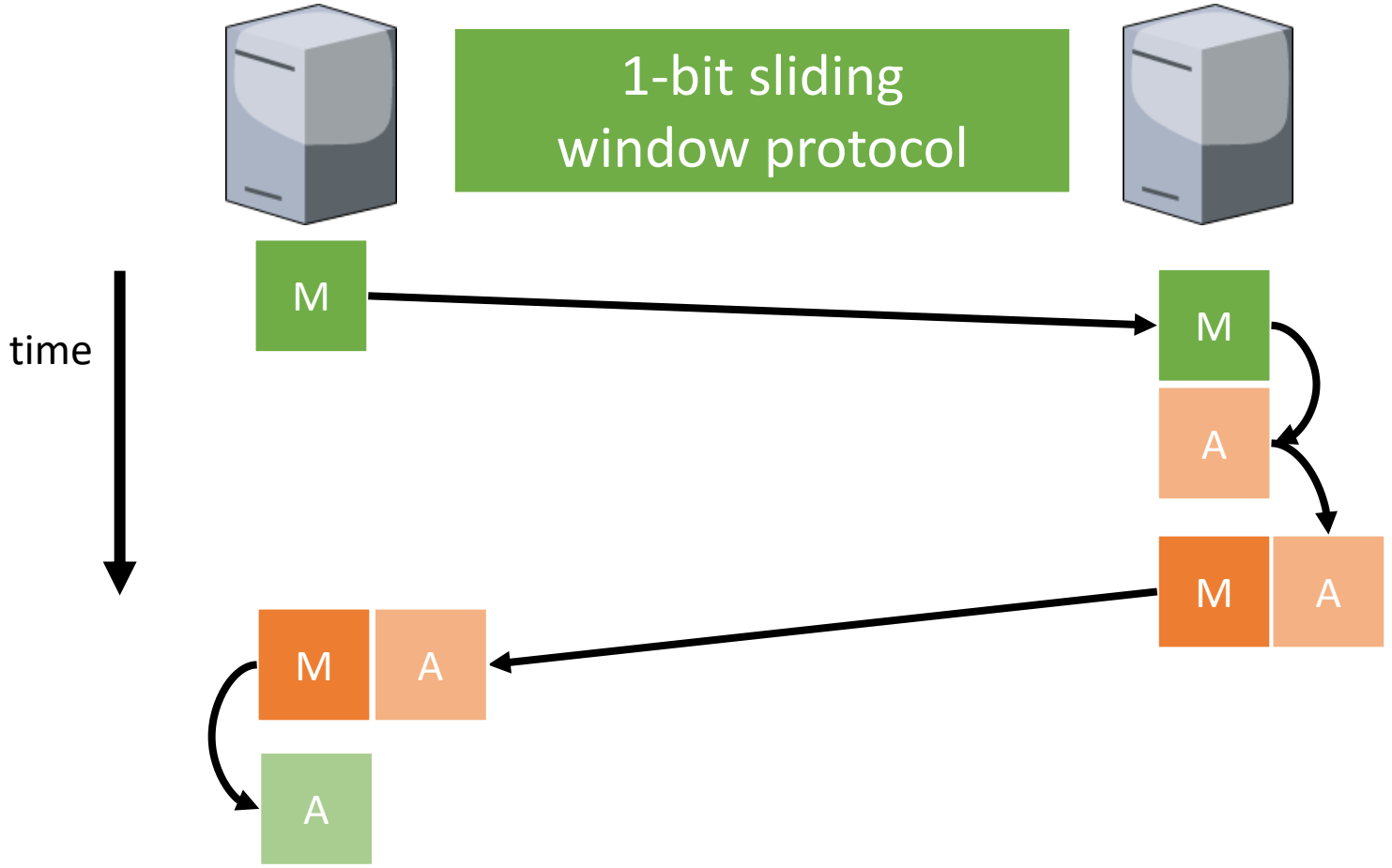
Part 2

1. Error detection
2. Error correction

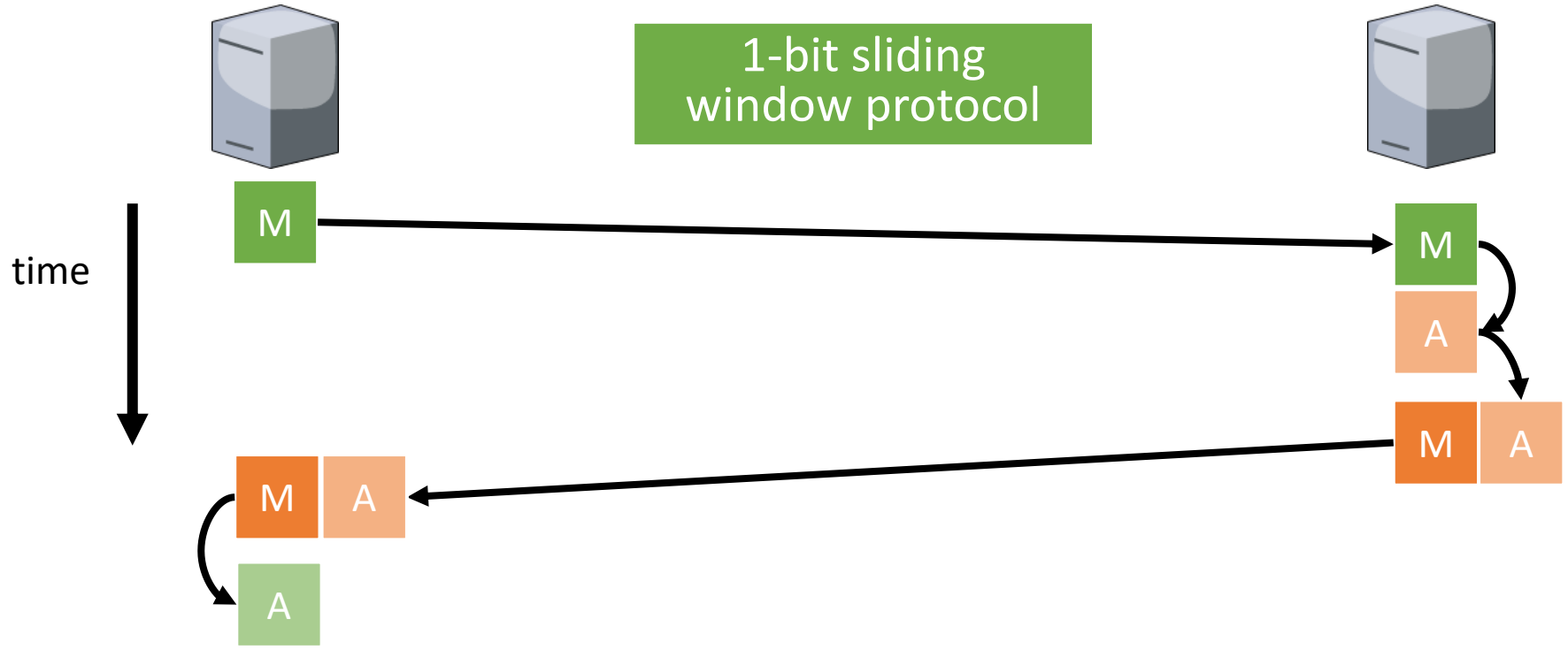
Sliding Window Protocols

Improving Performance using *Pipelining*

Stop-and-Wait:
A 1-Bit Sliding Window Protocol

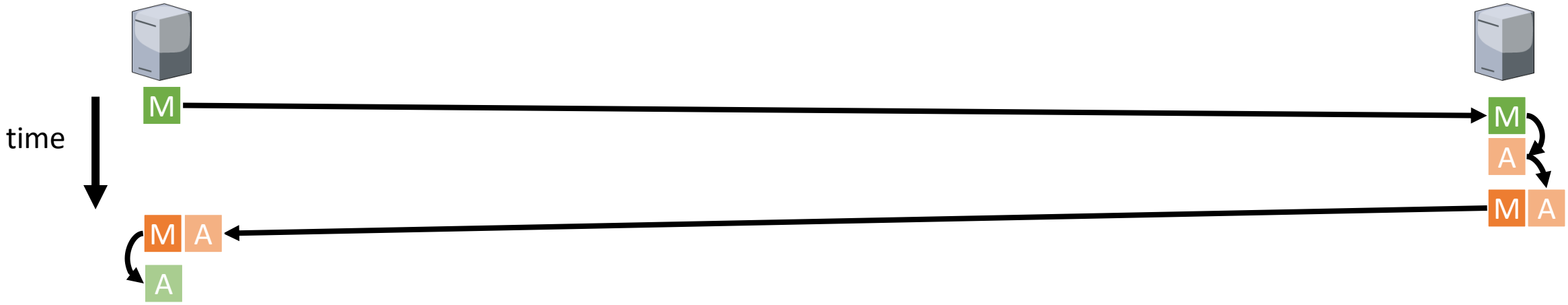


Stop-and-Wait:
A 1-Bit Sliding Window Protocol



Stop-and-Wait:
A 1-Bit Sliding Window Protocol

1-bit sliding window protocol



Bandwidth inefficient for high-latency channels

Sliding window protocols

Q: Which properties cause performance to decrease?

When using stop-and-wait, *data rate decreases* when:

- Latency increases
- Frame size decreases

Solution

Send next frame while waiting for acknowledgment of current frame

Sender window specifies how many frames a sender is allowed to send before waiting for an acknowledgement.

Receiver window specifies the range of frames that the receiver is allowed to accept.

Stop-and-Wait / ARQ: A 1-Bit Sliding Window Protocol

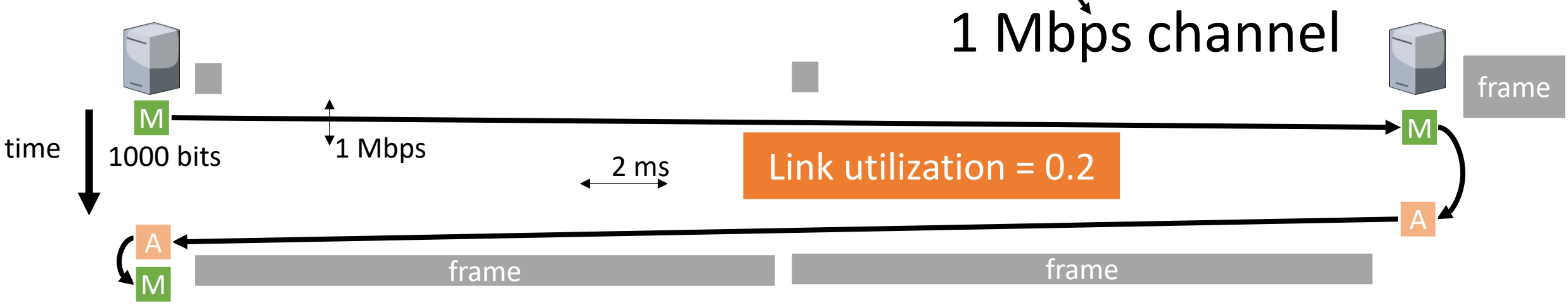
Q: What is the *link utilization*?

It takes $\frac{1{,}000}{1{,}000{,}000} = 0.001$ seconds to send frame

It takes 2 ms for the frame to arrive at the receiver, takes 2 ms for the (0-bit) acknowledgment to come back at the sender

1 frame per 0.001+0.002+0.002 seconds = 200 kbps

Small window reduces performance



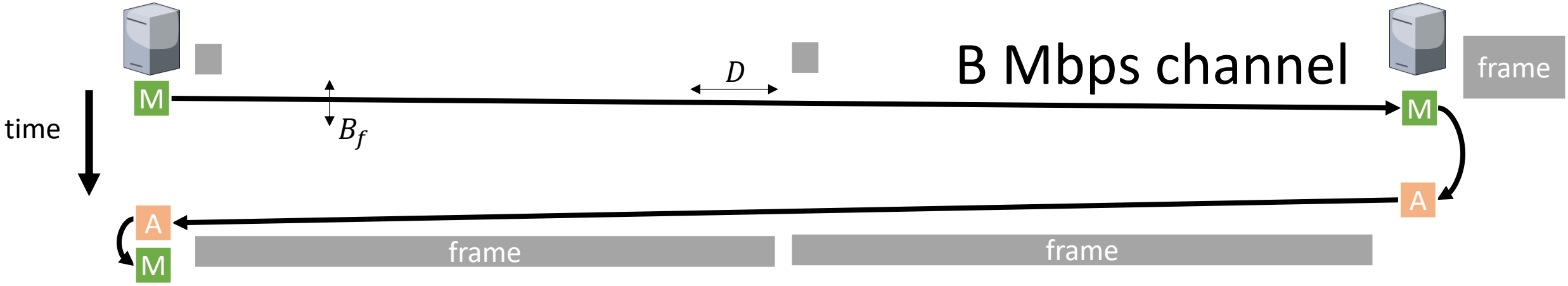
Link Utilization

$$\text{Link utilization} \leq \frac{w}{1+2B_f D}$$

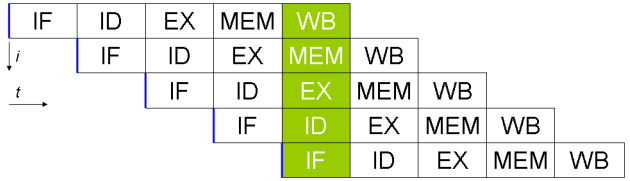
- Frame size (in bits/bytes): f
- Window size (in frames): w
- Bandwidth (max. data rate of physical channel): B_p
- Bandwidth (frames per second): B_f
- Propagation delay (in seconds): D

Sliding window protocol with window size of w frames can send w frames before stopping to wait for acks

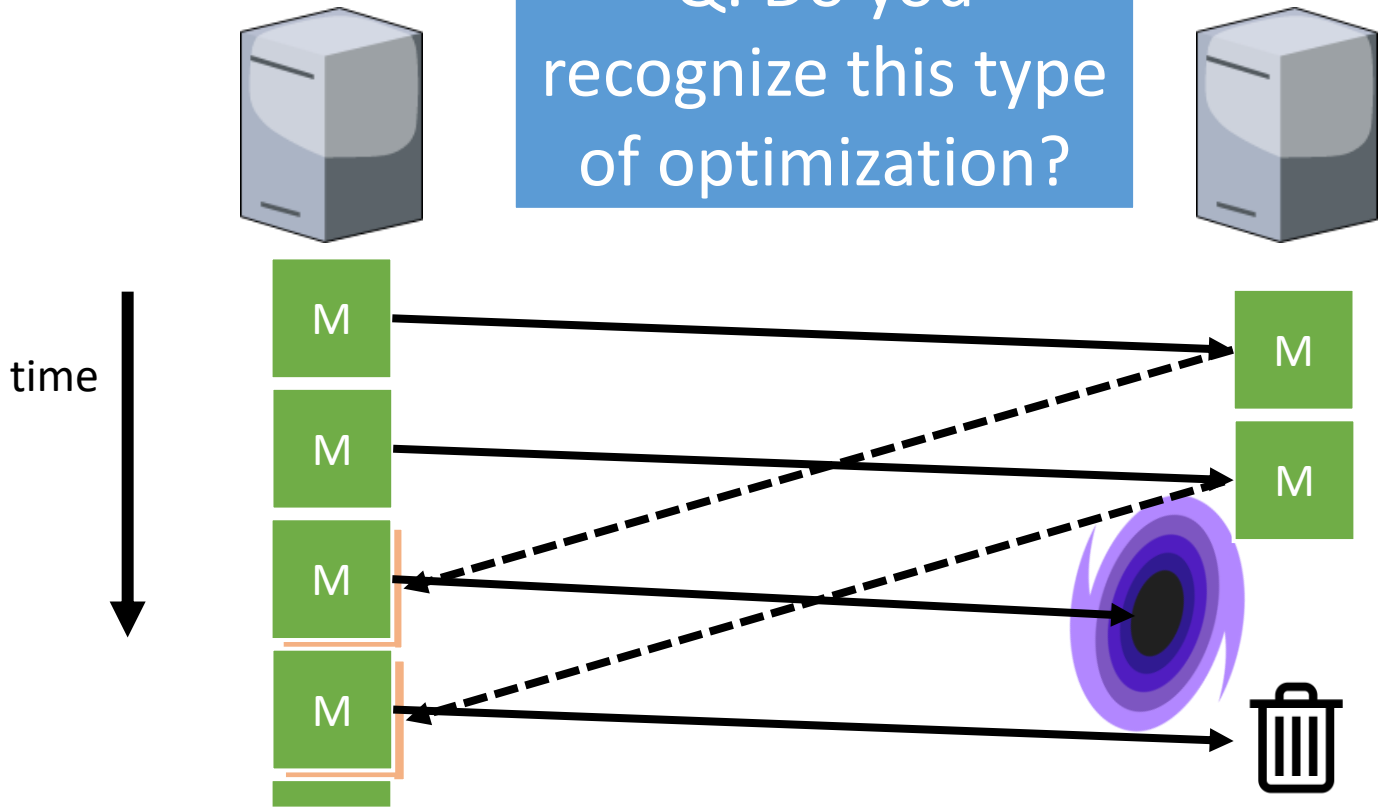
Link bandwidth of B_f frames per second, and delay of D seconds → the link can hold $2B_f D$ frames. The receiver can hold and process 1 additional frame → at most $1 + 2B_f D$ frames in transit



Go-Back-N



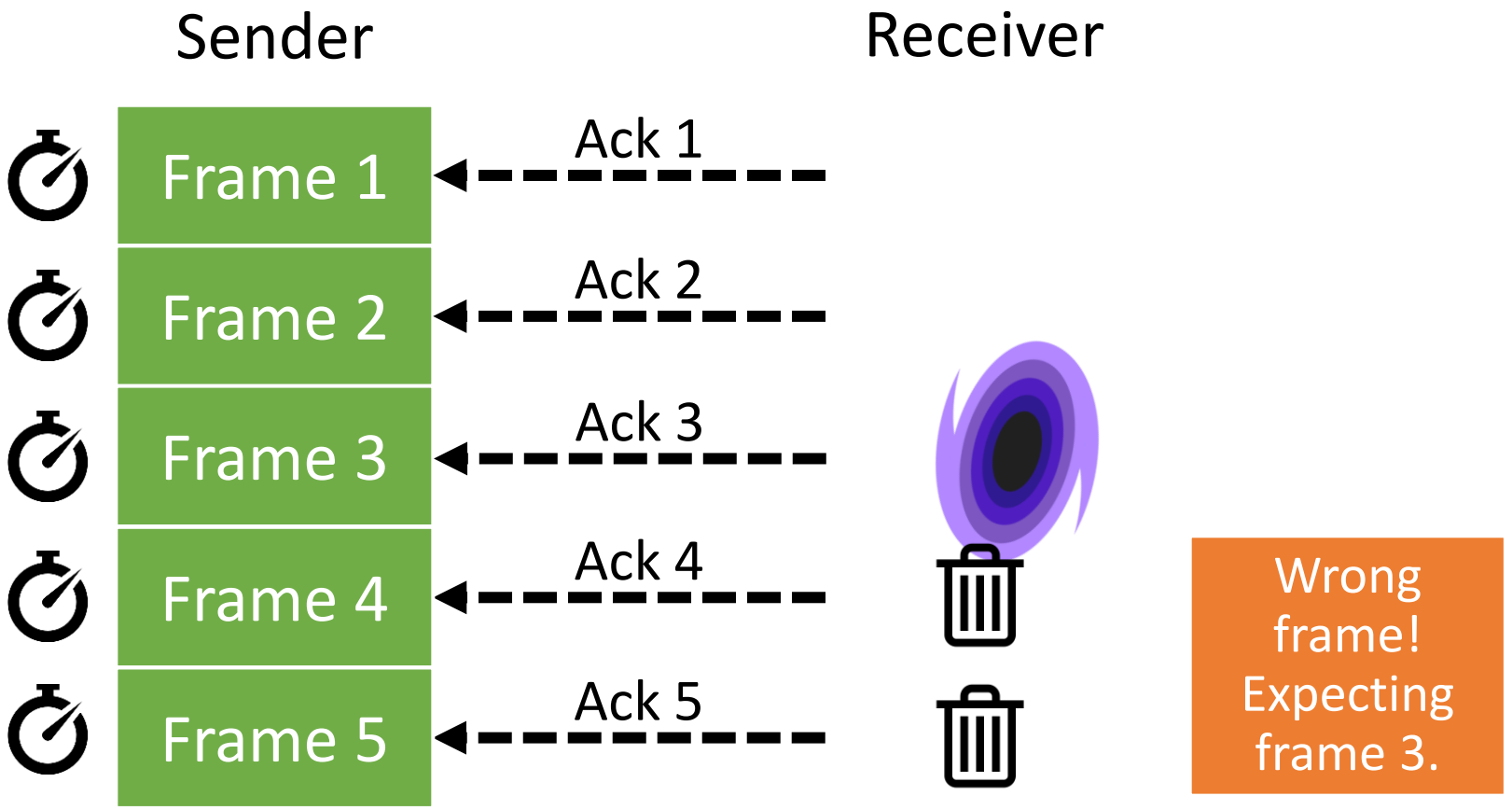
Q: Do you recognize this type of optimization?



Q: What is the size of the receiver window?

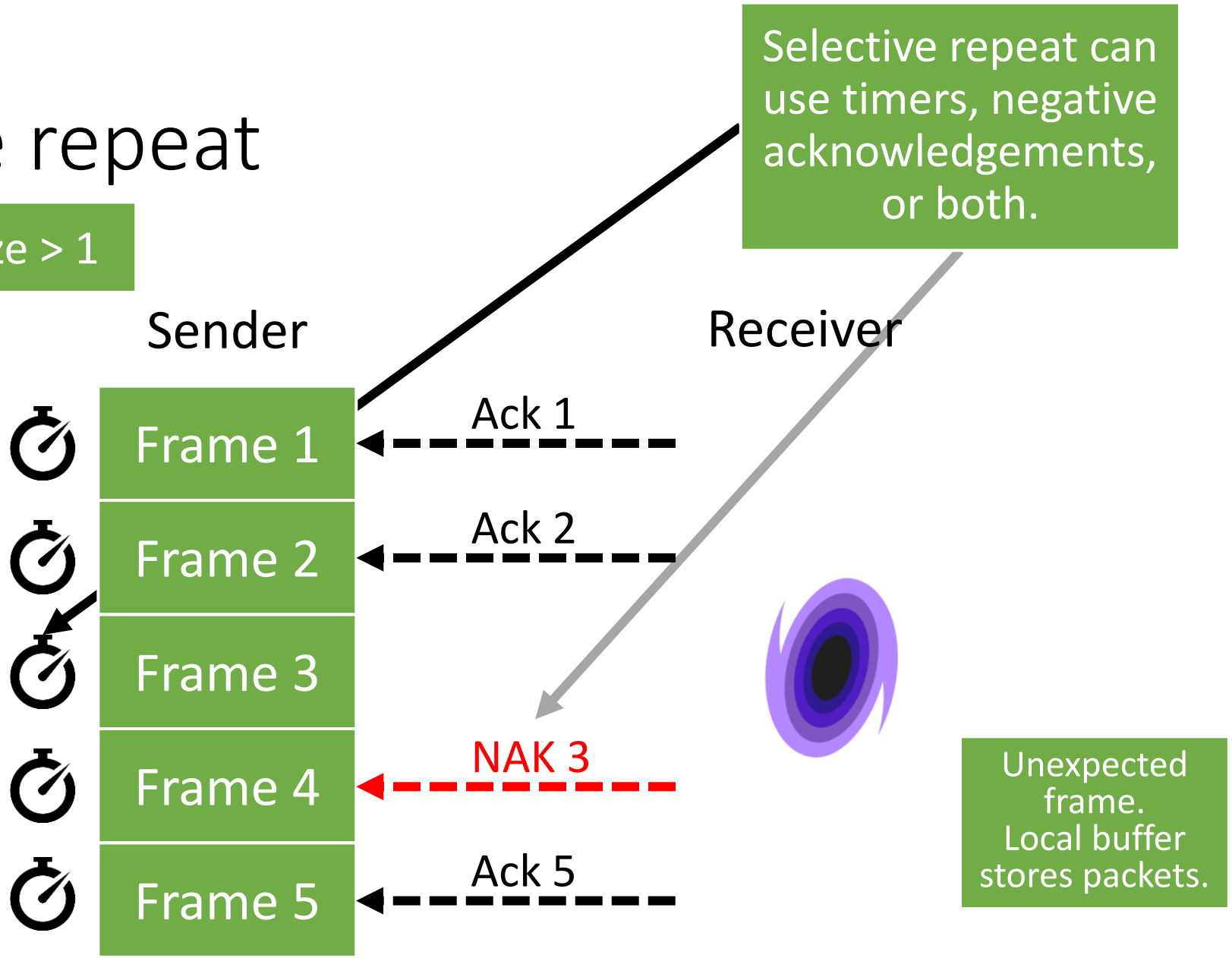
Go-Back-N

Receive window size = 1



Selective repeat

Receive window size > 1



Delivering frames in the order in which they were sent is easy after adding sequence numbers.

Data Link Layer — Roadmap

Part 1

- **Framing**
- **Flow Control**
- **Guaranteed Delivery**
- **Sliding Window Protocols and Link Utilization**

Part 2

- Error detection
- Error correction