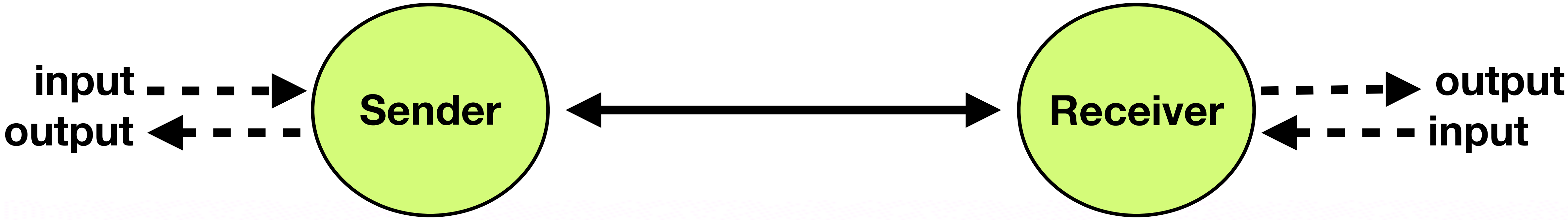
Basic Protocols and Error Control Mechanisms

Nicola Dragoni
Embedded Systems Engineering
DTU Informatics

- ACK/NACK Protocol
- Polling Protocol
- PAR Protocol
- Exchange of State Information
  - Two-Way Handshake Protocol
  - Three-Way Handshake Protocol

} Error Control Mechanisms

Simple ACK/NACK Protocol



- M = domain of correct messages
- M’ = domain of corrupted messages
- $M \cap M' = \{\ \}$
- “checksum” model: $x \in M$ or $x \in M'$

Polling

- In the previous simple ACK/NACK protocol:
  - it is the sender who takes the initiative for sending a message
  - the receiver merely responds to this.
- Effectively, this obliges the receiver to be able to receive data at any time after it has send an acknowledgment.

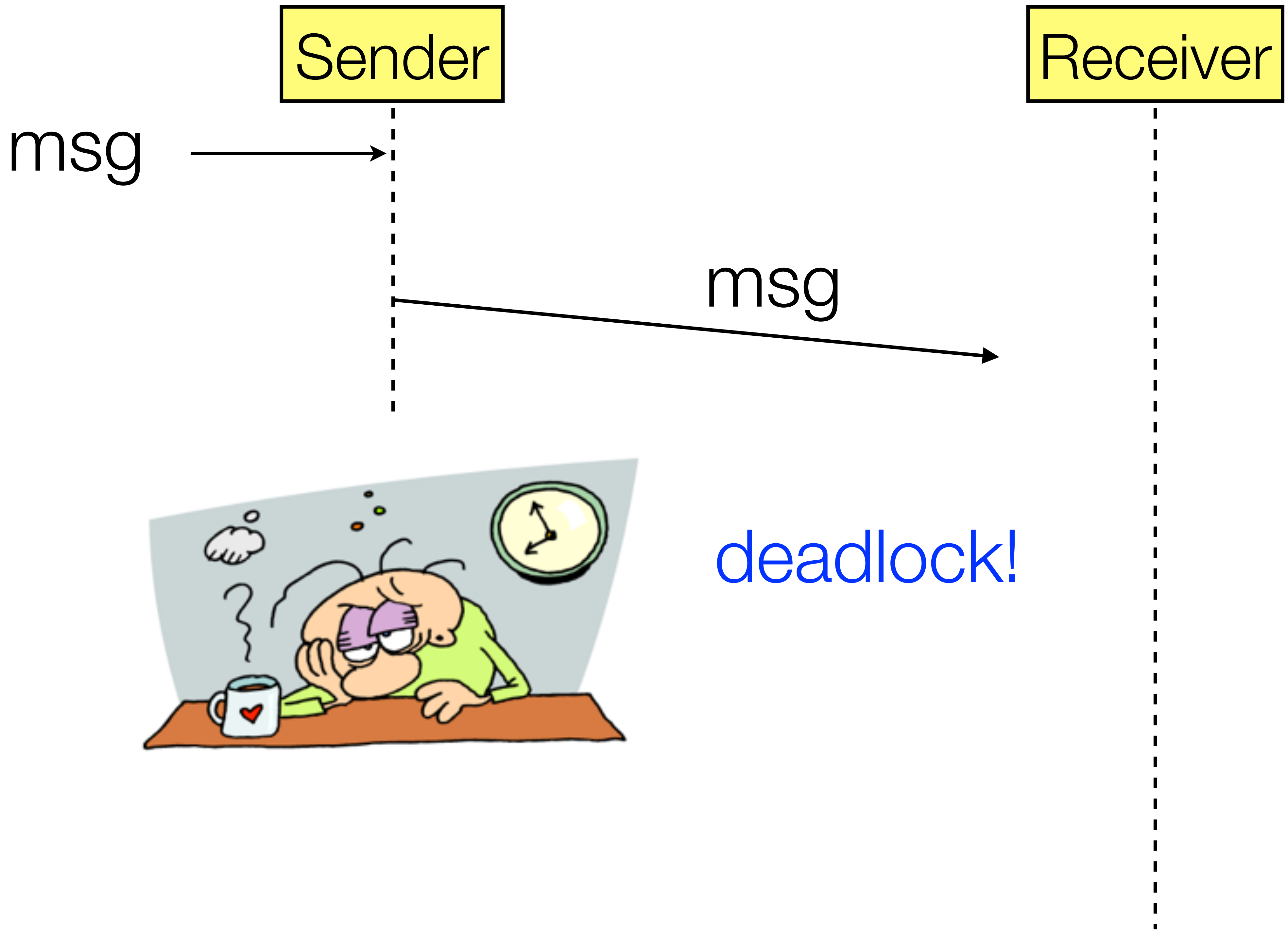
- Alternative strategy (POLLING):
  - the receiver explicitly takes the initiative, requesting data when it is able to receive them.

Simple Polling Protocol

- Receiver has initiative!

- Messages:
  - POLL : request to send data
  - REPT: request to repeat transmission of data received with errors

ACK/NACK Problem



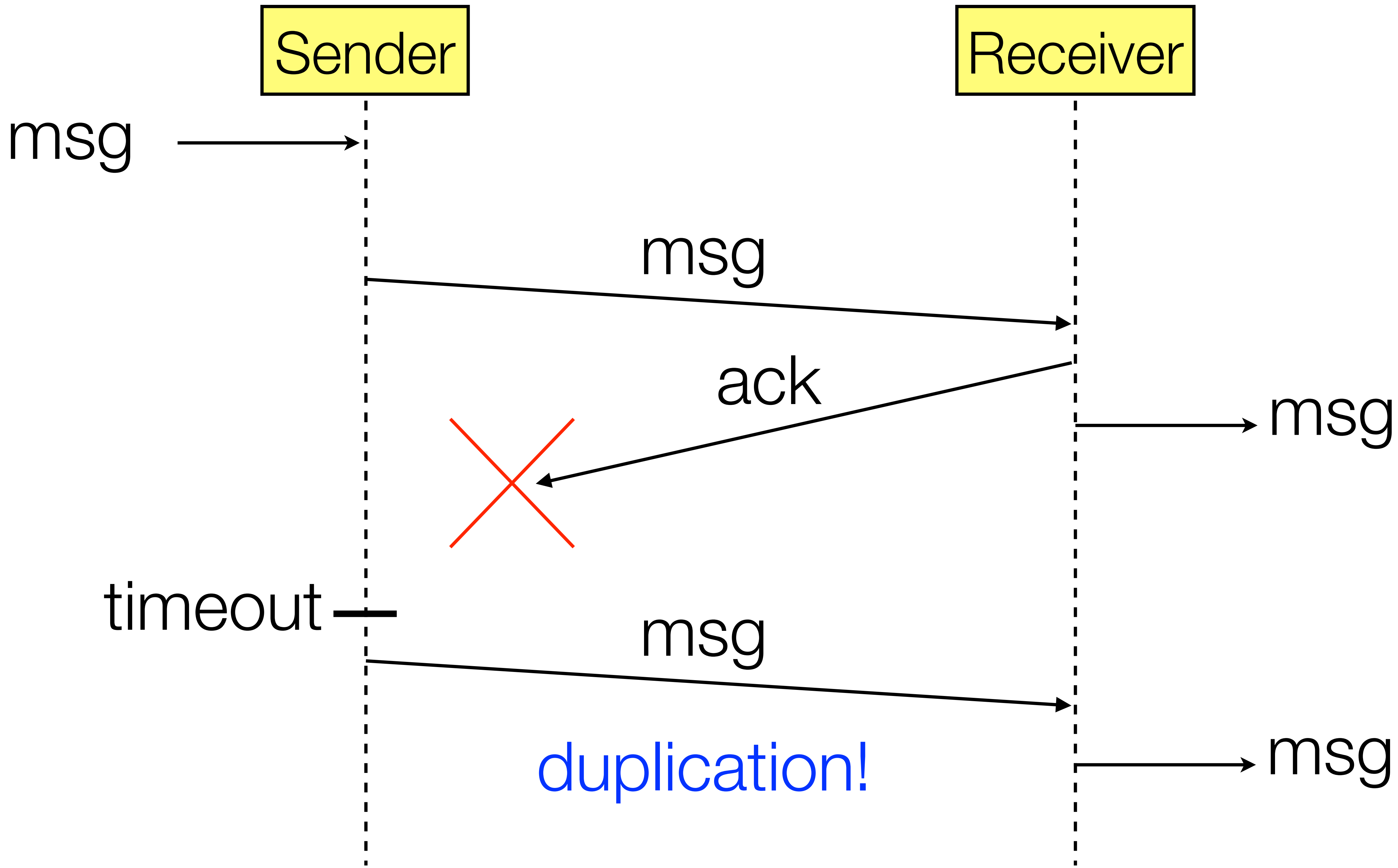
ACK/NACK + TIMEOUT

- Deadlock caused by loss of the acknowledgment message.
- Corrected by retransmission after a *certain time* with no acknowledgment.

ACK/NACK + TIMEOUT - Duplication Problem

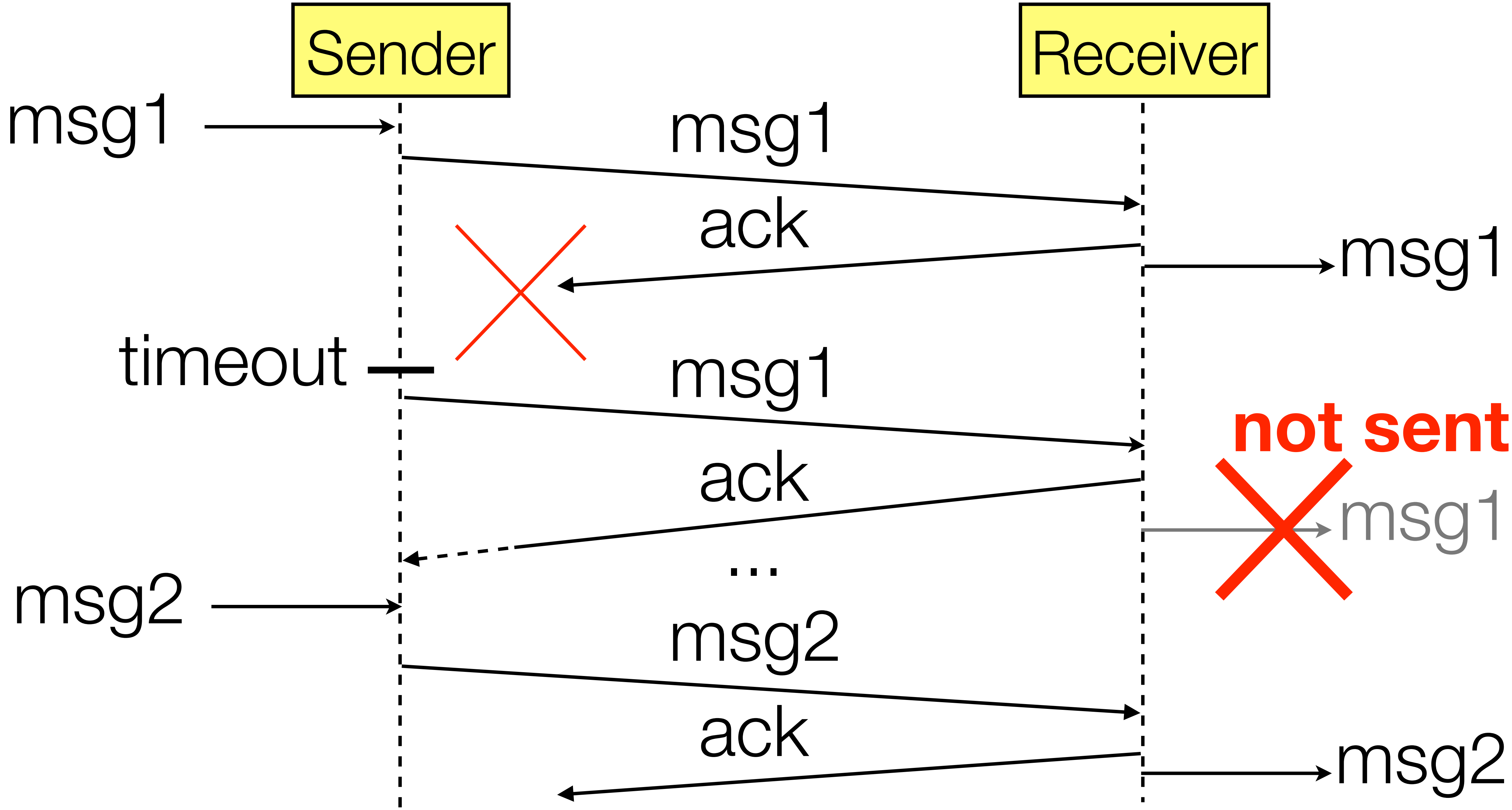
- Consider the following situation:
  - the receiver receives a correct message via its channel left and then sends a positive acknowledgment
  - this acknowledgment message gets lost
  - the sender will eventually time out, and retransmit the same message to the receiver
  - so the receiver receives the message twice and passes it on to the user (output) twice.

ACK/NACK + TIMEOUT - Duplication Problem



Possible Solution: Numbering Scheme

- Introducing a numbering scheme for the messages: duplicated messages can be filtered off by the receiver before messages are passed to the user.



HOMEWORK

Exercise

- Write a specification of an ACK/NACK protocol able to handle the following failures:

  1. deadlock caused by the loss of the acknowledgment message

  2. duplication of messages sent to the user

PAR Protocols

- We can also **remove the NACK type of acknowledgment**. Why?
  - When a timeout mechanism is used, **negative acknowledgments only have an effect on the response time of the protocol**, since they can be used to provoke retransmission before the timeout period runs out.
  - *Negative acknowledgments do not affect the logical properties of the protocol in any way*.
- Protocols with:
  - only positive acknowledgments +
  - using a timeout mechanism to control retransmission

ACK

are often called Positive Acknowledge and Retransmission (PAR) protocols.

PAR Protocol (ACK + TIMEOUT + NUMBERING SCHEME)

- PAR Protocol with Timeout and Sequence Numbers

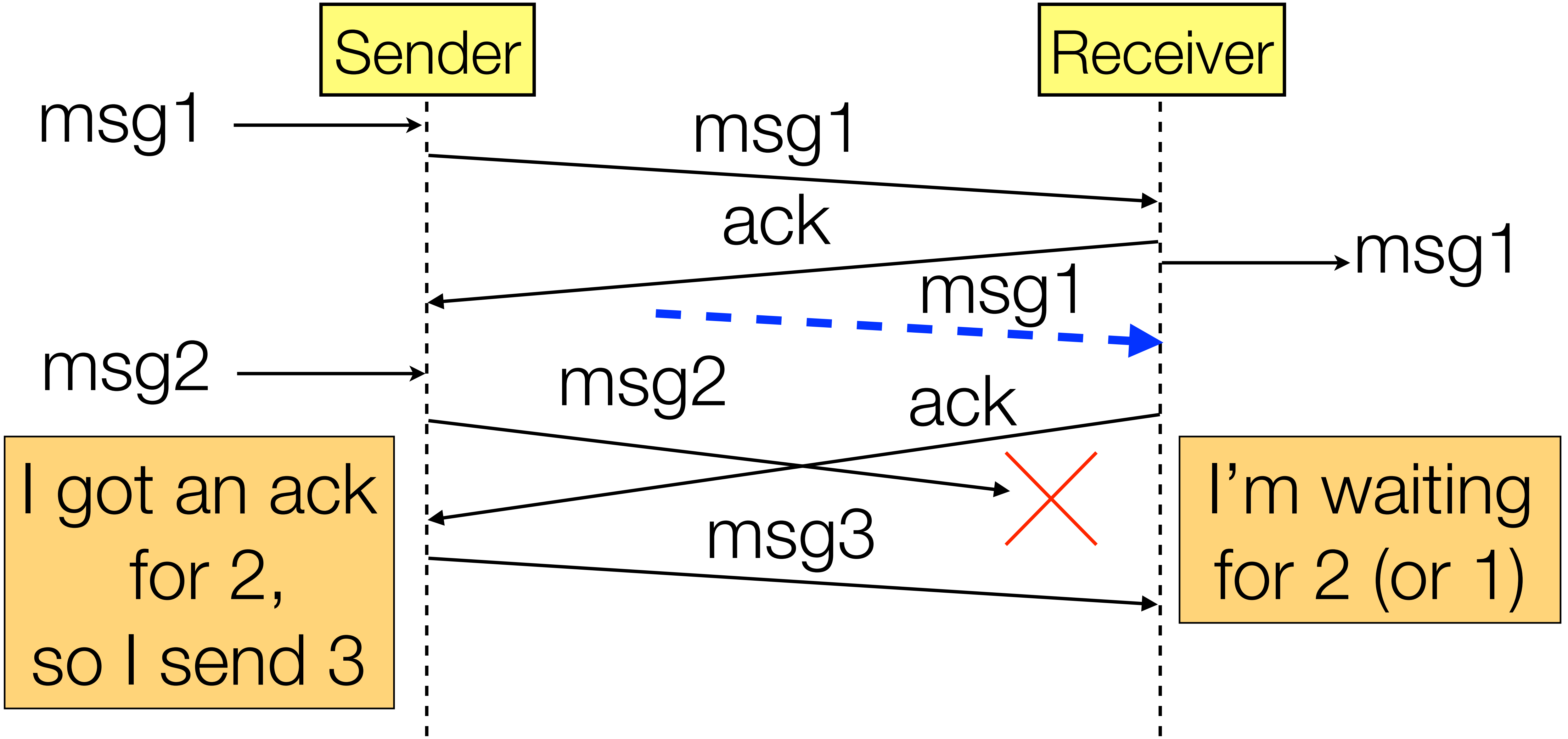
Exercise: Polling Protocol

HOMEWORK

1. Extend the polling protocol with sequence numbers and timeout.

2. Analyze your proposal to see which problems (if any) the protocol might still have.

PAR Protocol - Problem



The acknowledgements are anonymous!

Problem: Anonymous ACK

- Anonymous Acknowledgement Problem: all protocols we have seen so far rely on *anonymous messages*.
  - ACK messages:
    - just tell the sender that the other party has received the data which came in the right order
    - the sender has no means of knowing exactly which data is referred to.

This reflects a general problem in distributed systems: the cooperating parties do not in general know what their collective **global state** is.

- Parties have to make decisions on the basis of
  - whatever information they locally have available or
  - the information their cooperators have sent them.

Solution: Sequence Numbers in ACKs

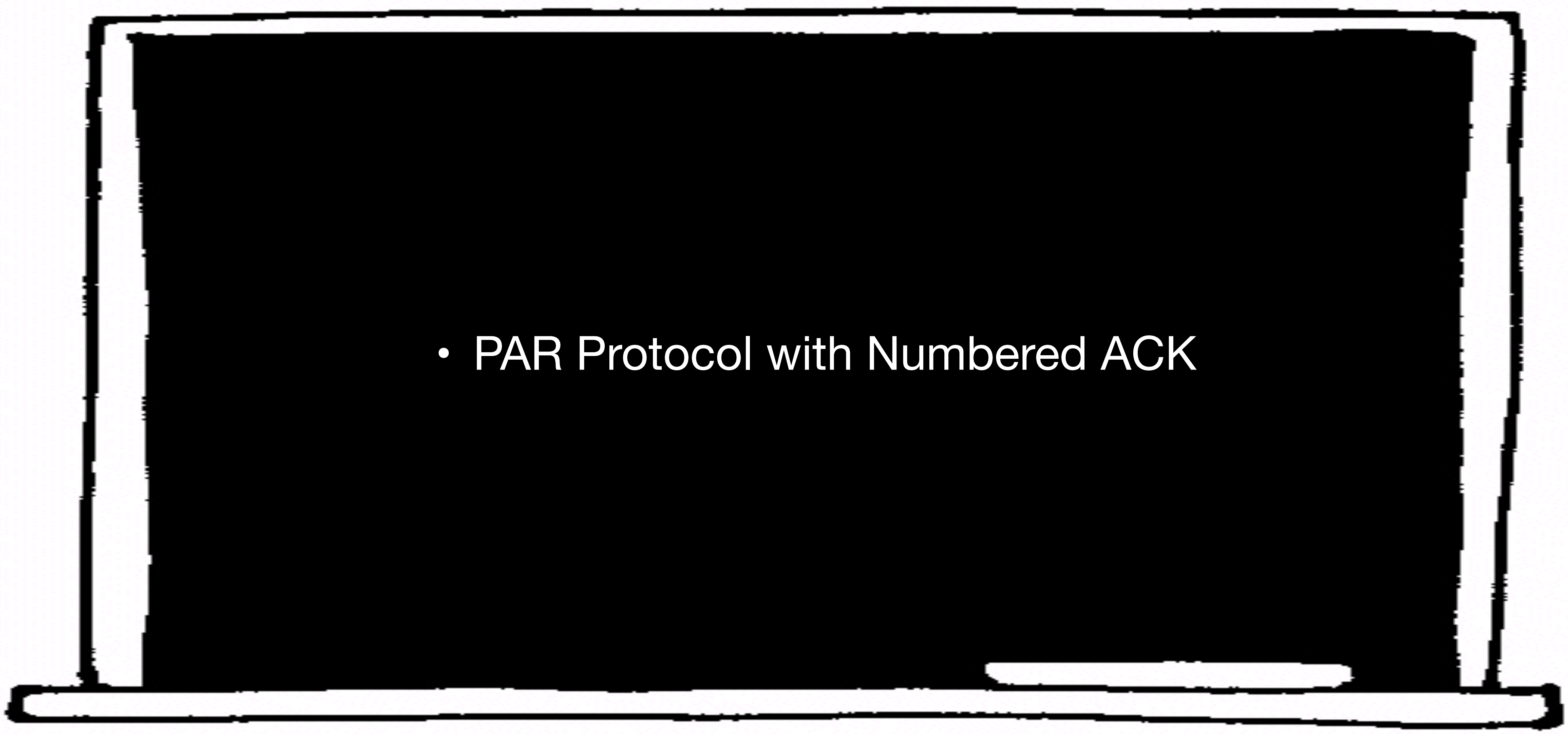
- We include an identification on the acknowledgments, indicating the sequence number of the latest correctly received data.

- Sender:
  - repeats message with number *n* until it receives an acknowledgment explicitly denoting *n*.

- Receiver:
  - replies to each correct incoming data with an acknowledgment that includes the sequence number of the *last correctly received message* (which of course may be the message just received or a previous one).

Example: PAR Protocol + NUMBERED ACK

- The ack message now consists of the NUMBER OF THE LATEST CORRECTLY RECEIVED data message.



Sequence Numbers?

- Simple idea: Sequence numbers are successive natural numbers 0, 1, 2, 3, ...
- Problem: Only a finite number can be represented in a real message.
- New idea: If acknowledgment is received within relatively short time, it is only necessary to count modulo some small value S_{mod}, so

$$succ(n) \stackrel{\text{def}}{=} (n+1) \bmod S_{mod}$$

- Example [PAR protocol with numbered ACK]: Sender always waits for positive ACK for latest transmitted message before using next sequence number. OK to count modulo 2 ("Alternating Bit Protocol").
- If more messages can be outstanding (sent but not acknowledged), S_{mod} must be larger.

ESSENTIAL RULE: messages with number n must be guaranteed to be "dead" before n is re-used.

PAR Protocol + NUMBERED ACK

- Protocol now gives both parties sufficient knowledge of what is happening, so it protects against
  - loss
  - duplication
  - corruption

  of both data messages and ack messages.

But it can still fail. How?

Floating Corpses

But it can still fail. How?

- Imagine a system where msgs can get lost for a considerable period of time.
- In our protocols:
  - The sender eventually times out, declares the messages "dead", and retransmits them.
  - The receiver accepts the retransmitted messages.
- All seems well!!
- But at this moment the corpses come floating up to the top of the service, as it were, and arrive at the receiver.
- Total confusion arises, as most protocols are unable to counteract this form for masquerading.

Class of Error: Masquerading

- Masquerading: introduction by the underlying service (channel) of **false messages** which look as though they are correct ones.
  - For instance: because they have appropriate sequence numbers and belong to the set of correct messages.

- Possible solutions?
  - Never re-use sequence number! Not realistic...
  - Use ENORMOUS sequence number space! After a crash it is extremely difficult to guarantee that we can remember where we got in the sequence numbers.
  - Explicit limits to message lifetime! Several techniques are possible. In practice, combinations of these techniques are often used.

Exchange of State Information

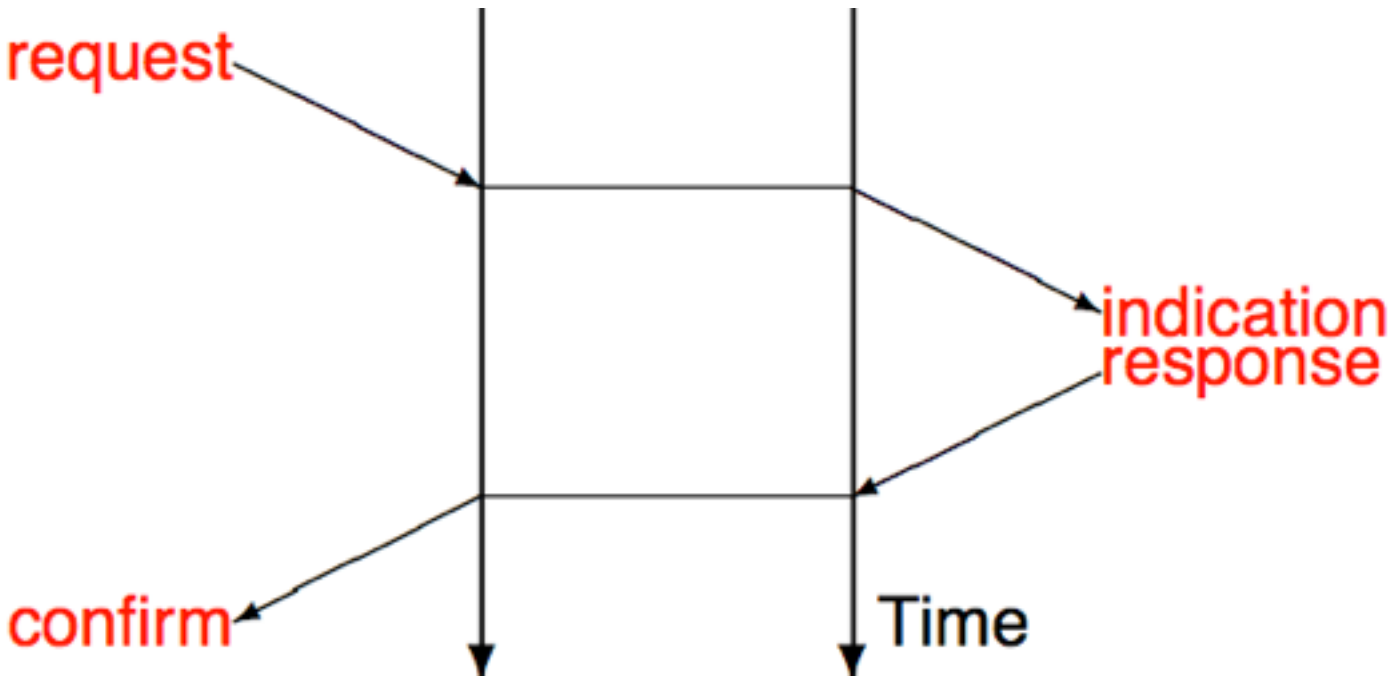
Protocols for Exchange of State Information

- Can be necessary, for example:
  - To agree on an initial state.
  - To indicate a change of state.
  - To set up or break a connection.
  - To perform an atomic action.

TCP

TSL/SSL

- Reliable exchange requires at least exchanging a message in each direction (CONFIRMED EXCHANGE).
- Often depicted by TIME-SEQUENCE DIAGRAM



Two-Way Exchange (or Handshake) Protocol

Two-Way Handshake Protocol

- Req: requests
- Accept: positive replies
- Refuse: negative replies
  ERROR ∊ Refuse: *internal* message indicating refusal
- Accept and Refuse are DISJOINT SETS
- At **(. . .)**, both parties are sufficiently finished to go on with the next part of their tasks.

Exchanges in the Presence of Errors

- We might use the same techniques adopted before (i.e., retransmission, sequence numbers in data and acknowledgments) but...

... how to avoid the FLOATING CORPSES?

- It is not always possible to add sequence numbers to messages used for administrative purposes (for instance, actually establishing connection).
  - The initial sequence number for messages is one of the components of the global state which we wish to establish!
- So we must find some other information which can be exchanged and which will enable us to distinguish false messages from genuine ones during connection establishment.
- In particular, we need another exchange: three-way handshake.

Three-Way Handshake... in a Nutshell

- Used for the connection establishment phase of the Internet TCP Transport layer protocol.

- More generally, the protocol finds uses in all situations where a confirmed service is required over an unreliable underlying service.

- General scheme:
  - the initiating protocol entity sends a request message carrying an arbitrary value x
  - the responding entity replies with a response message bearing (x, y)
  - the initiating entity repeats this message as an extra confirmation.

Analogy: Exchange of Letters

- An analogy is the use of "our reference" and "your reference" fields in an exchange of letters.
  - If you get a letter with an unknown reference on it, you throw it straight in the wastebin.

- Normal run of the protocol:

A				B	
$\rightarrow$	$< req,$	$ourref = x >$		$\rightarrow$	A initiates.
$\leftarrow$	$< accept,$	$ourref = y,$	$yourref = x >$	$\leftarrow$	B responds.
$\rightarrow$	$< check,$	$ourref = x,$	$yourref = y >$	$\rightarrow$	A confirms.

Three-Way Handshake...

- Three-Way Handshake Protocol

What Happens with Floating Corps?

A				B	
$\ldots$	$< req,$	$ourref = x >$		$\rightarrow$	delayed req-PDU
$\leftarrow$	$< accept,$	$ourref = y,$	$yourref = x >$	$\leftarrow$	B responds.
					A gives up.
					(B times out.)

- B responds to a false request message
- A is unable to match B's reference x to any exchange which A is currently taking part

==> A gives up and (in our version of the protocol) B subsequently times out and therefore also gives up.

What Happens with Floating Corps?

A				B	
$\ldots$	$< req,$	$ourref = x >$		$\rightarrow$	delayed req-PDU
$\leftarrow$	$< accept,$	$ourref = y,$	$yourref = x >$	$\leftarrow$	B responds.
$\ldots$	$< check,$	$ourref = x,$	$yourref = z >$	$\rightarrow$	delayed check-PDU
					A and B give up.

- B responds to a false request message
- but when it receives the false check message from A it finds an incorrect reference z instead of the value y which it itself had generated

==> A and B give up without timeout.

Exercise: 3-Way Handshake

HOMEWORK

- The protocol should survive receipt of out-dated request/response/check messages.
  - Analyze the protocol to check whether or not this is true.

Could the protocol still fail in some other situation?