DN-2/V-2

Notiztitel 23.02.2005

Networking

Prof. Dr.-Ing. Holger Hermanns

Dependable Systems & Software
Saarland University

Lecture 0: Prelude

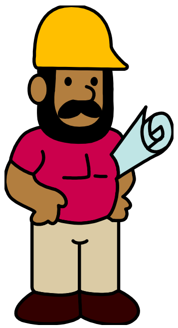
Prelude: Who am I?

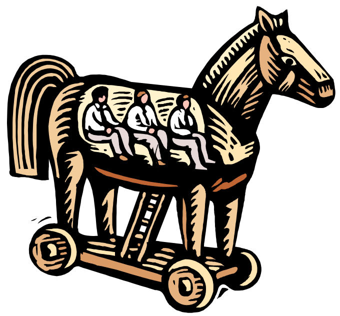
- Prof. Dr.-Ing. Holger Hermanns

- Bldg. 45, Office 501

- e-mail: hermanns@cs.uni-sb.de

Prelude: Who are you?





Prelude: What is this course about?

1. It's about Computer Networks.

2. It's about the how and why of the Internet.

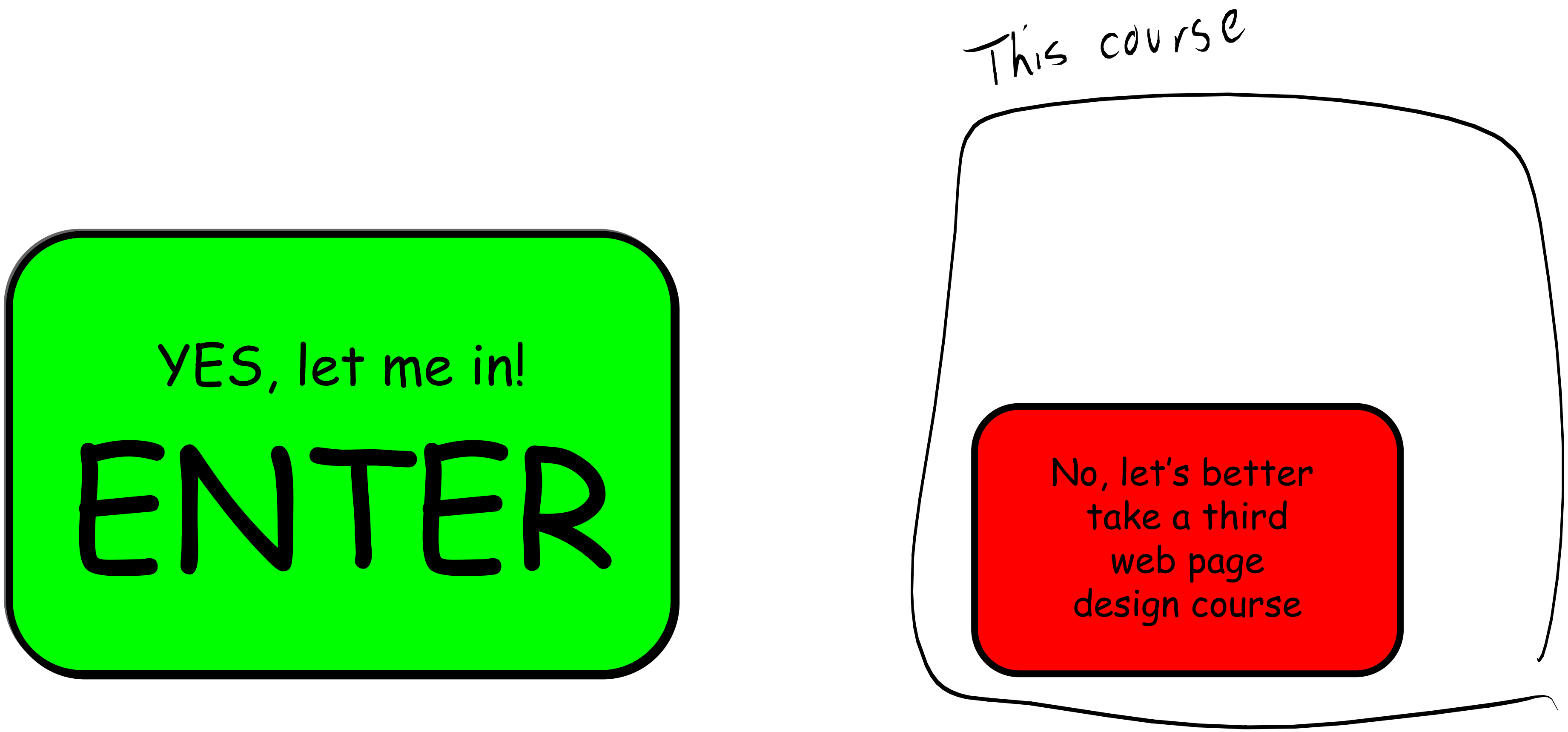
3. It's about foundations of networking beyond the Internet.

Prelude: Course Mater

http://spinroot.com/spin/Doc/Book91.html

Prelude: Why not to take this course?

Disclaimer: This lecture may contain material which is considered offending by students who believe that intellectual challenges are to be avoided.



Networking

Prof. Dr.-Ing. Holger Hermanns

Dependable Systems & Software
Saarland University

Lecture 1: Introduction

Part I: Introduction

Important: read chapter 1 in [Kurose-Ross]

Our goal today:

- get context, overview, "feel" of networking
- more depth, detail *later* in course
- approach:
  - descriptive
  - use Internet as example

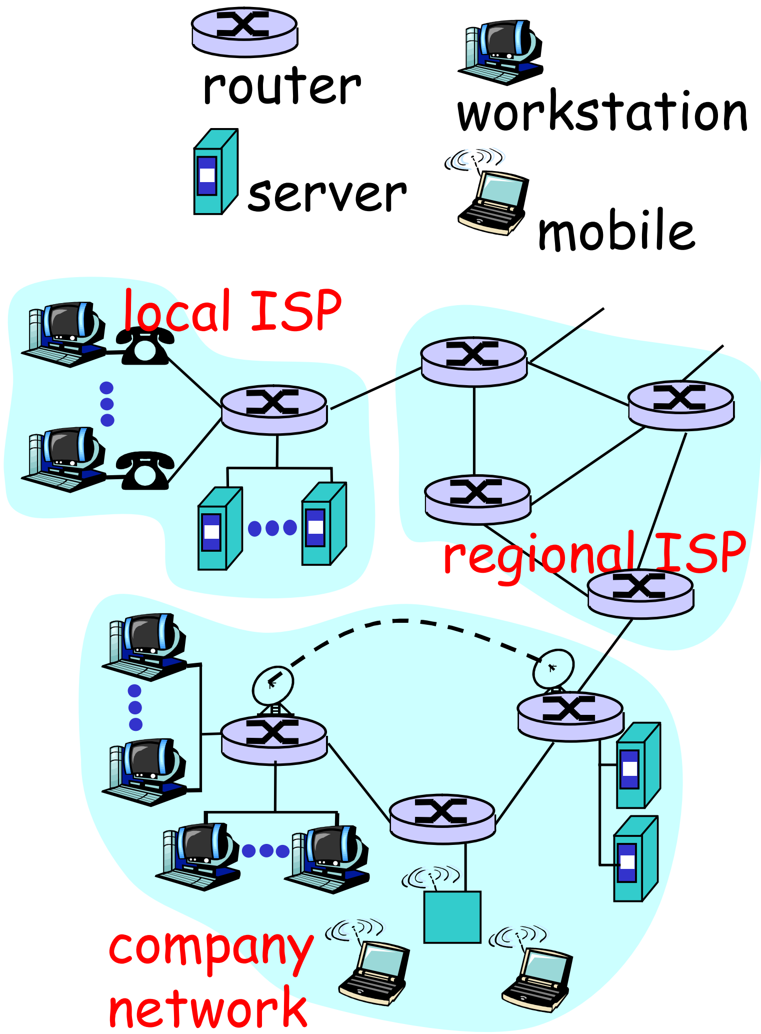
Overview:

- what's the Internet
- what's a protocol?
- network edge
- network core
- access net, physical media
- performance: loss, delay
- protocol layers, service models
- backbones, NAPs, ISPs
- history

What's the Internet: "nuts and bolts" view

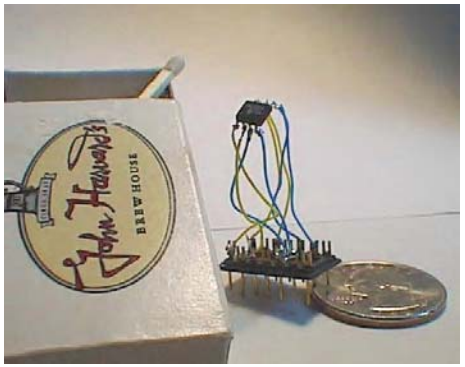
- millions of connected computing devices: *hosts, end-systems*
  - pc's workstations, servers
  - PDA's phones, toasters

  running *network apps*
- *communication links*
  - fiber, copper, radio, satellite
- *routers:* forward packets (chunks) of data thru network



“Cool” internet appliances

Digital photo receiver frame
http://www.ceiva.com/

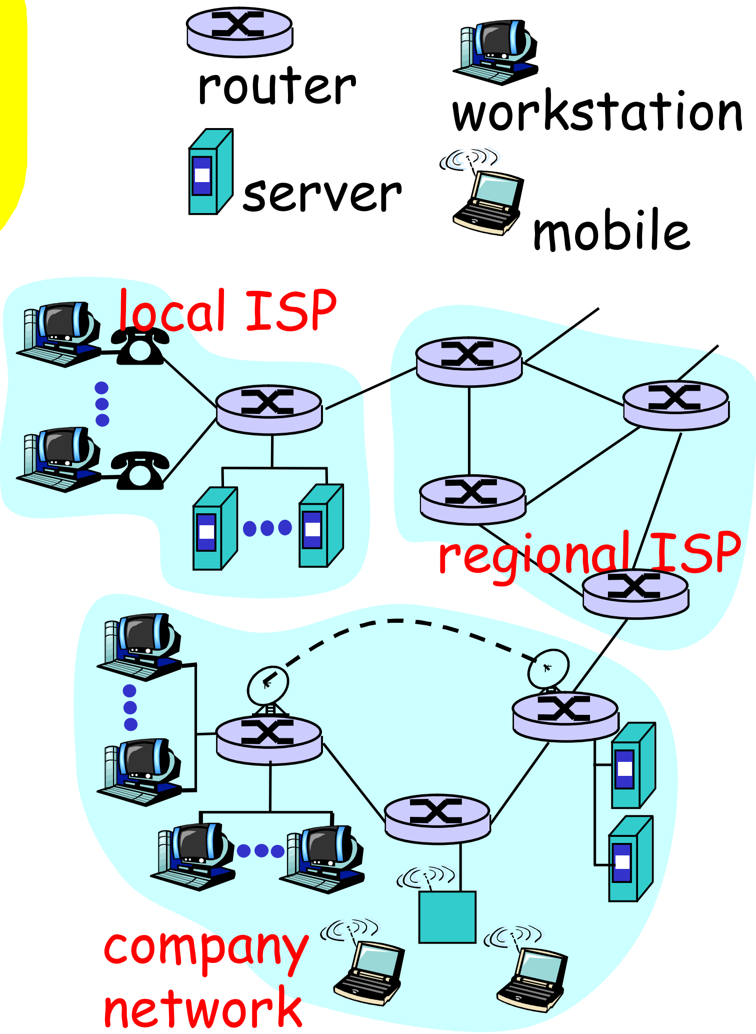


World’s smallest web server
http://www-ccs.cs.umass.edu/~shri/iPic.html

Web-enabled toaster+weather forecaster
http://dancing-man.com/robin/toasty/

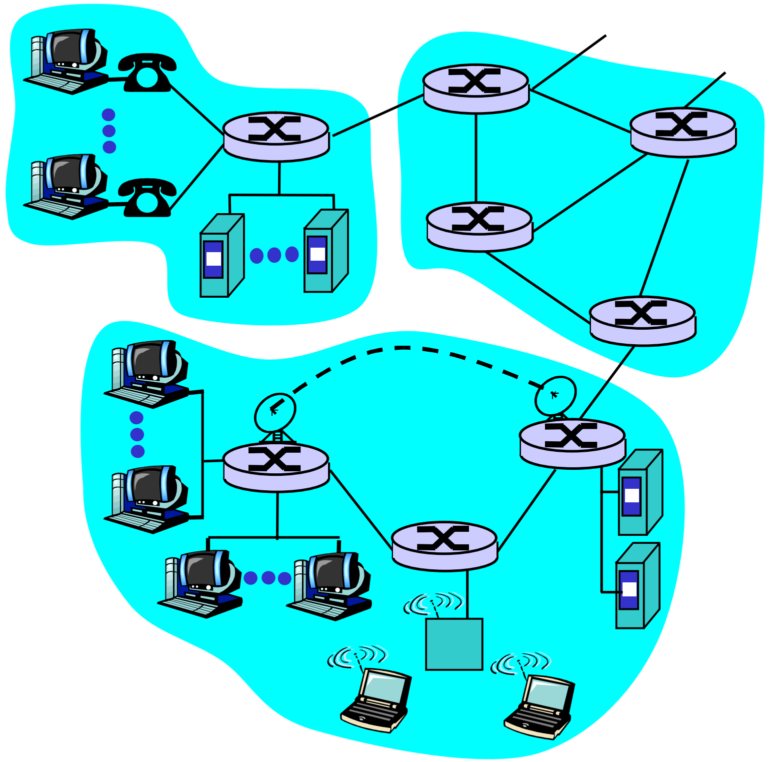
... "nuts and bolts" view

- *protocols*: control sending, receiving of msgs
- ... networks"
  - loosely hierarchical
  - public Internet versus private intranet
- Internet standards
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force



What's the Internet: a service view

- communication *infrastructure* enables distributed applications:
  - WWW, email, games, e-commerce, database, voting, file (MP3) sharing
- communication services provided:
  - connectionless
  - connection-oriented



- cyberspace [Gibson]:

  "a consensual hallucination experienced daily by billions of operators, in every nation, ..."

What's a protocol?

human protocols:

- "what's the time?"
- "I have a question"
- introductions

... specific msgs sent

... specific actions taken when msgs received, or other events

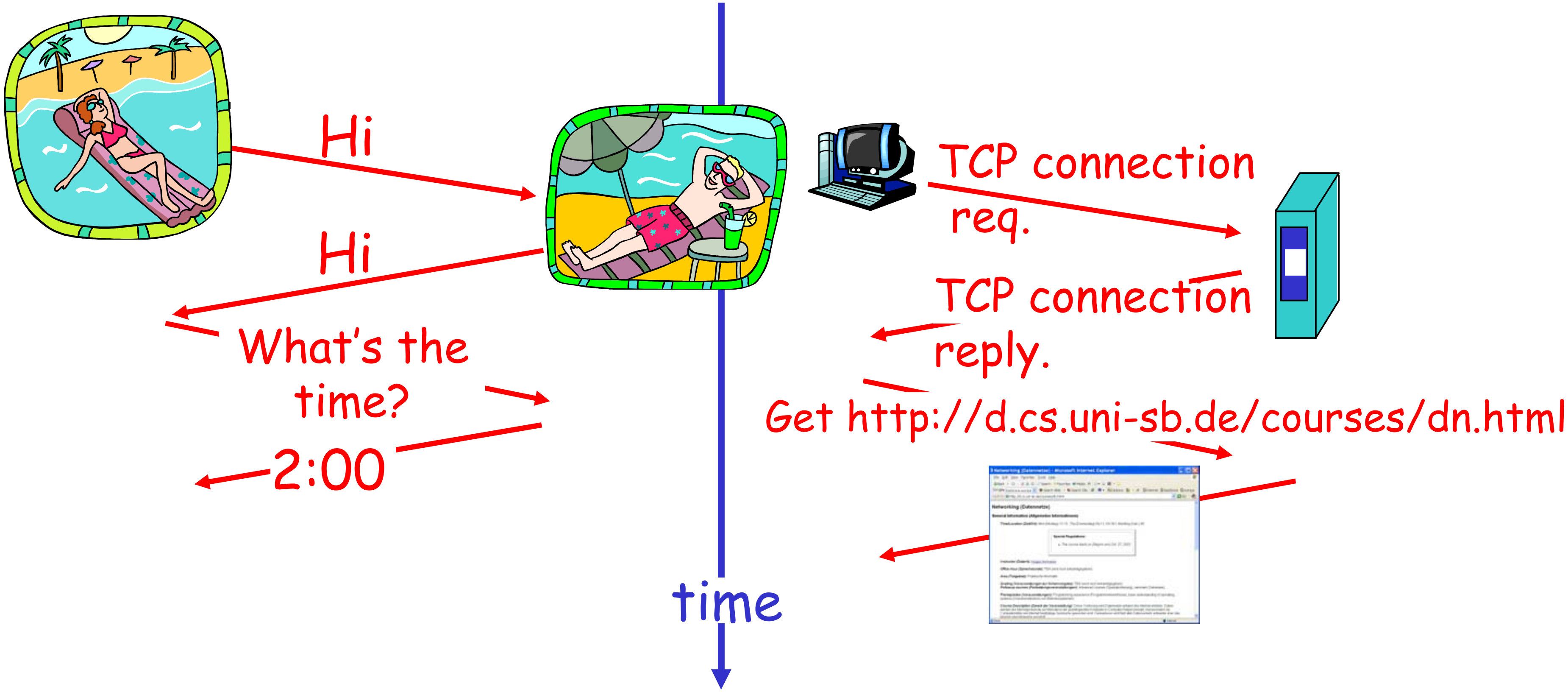
network protocols:

- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt

What's a protocol?

a human protocol and a computer network protocol:



Q: Other human protocol?

Networking

Prof. Dr.-Ing. Holger Hermanns

Dependable Systems & Software
Saarland University

Summer 04

Session A: Protocol Conventions

Protocol Conventions

Suggested:
re
o

Our goal today:

- illustrate the principal elements of any protocol
  - service
  - environmental assumptions
  - vocabulary
  - encoding
  - behavioural rules
- appreciate their event-driven nature
- learn about protocol notations

Overview

- Some history
- Elements of a protocol
- Sequence diagrams and MSCs
- State-transition diagrams and LTS
- Protocol flaws

Recall: What's a protocol?

human protocols:

- "what's the time?"
- "I have a question"
- introductions

... specific msgs sent

... specific actions taken when msgs received, or other events

network protocols:

- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt

Some history of protocols

- Ok, Internet has quite an interesting history.

- But protocol history dates back a little longer, at least to 458 B.C.:

According to Aeschylus (in the play *Agamemnon*), fire signals were used to to communicate the fall of Troy to ~~Athens~~ over a distance of more than 450 km.

Argos / Mykene

A detailed account

[Polybius, 2nd century B. C.]

"It is evident to all that in every matter, and especially in warfare, the power of acting at the right time contributes very much to the success of enterprises, and fire signals are the most efficient of all the devices which aid us to do this. [...] anyone [...] even if he is at a distance of three, four or even more days' journey can be informed."

More on the problems of past protocols

[Polybius, 2nd century B. C.]

*"[...] it was possible [...] to convey information that a fleet had arrived at Oreus, Peparethus, or Chalcis, but when it came to some of the citizens having changed sides or having been guilty of treachery or a massacre having taken place in the town, or anything of the kind, things that often happen, but cannot all be foreseen — **and it is chiefly unexpected occurrences which require instant consideration and help** — all such matters defied communication by fire signal. For it was quite impossible to have a preconcerted code for things which there was no means of foretelling."*

So, what is a protocol?

A set of rules governing communication

- there are at least two parties.
- they have some mutual concern, e.g.
  - selling/buying bread
  - transferring an mp3
  - making Troy surrender
- they have something in common.
- they are communicating in some physical environment.

Service provided by a protocol

Transfer of *information* (or bread)
between a source and one or more destinations

- Some Issues:
  - naming and addressing of the source and destination
  - naming and addressing of the channel (logical or physical)
  - properties of the underlying channel
  - initiation and termination of the connection
  - interpretation of the information
  - error handling

Some concerns

- How do we get started?
- What are we trying to communicate?
- Do we care whether the data/information is received?
- What is the penalty for failure?
- How do we finish?

The five elements of a protocol

A protocol specification consists of five distinct parts. To be complete, each specification should include explicitly:

1. The *service* to be provided by the protocol

2. The *assumptions* about the environment in which the protocol is executed

3. The *vocabulary* of messages used to implement the protocol

4. The *encoding* (format) of each message in the vocabulary

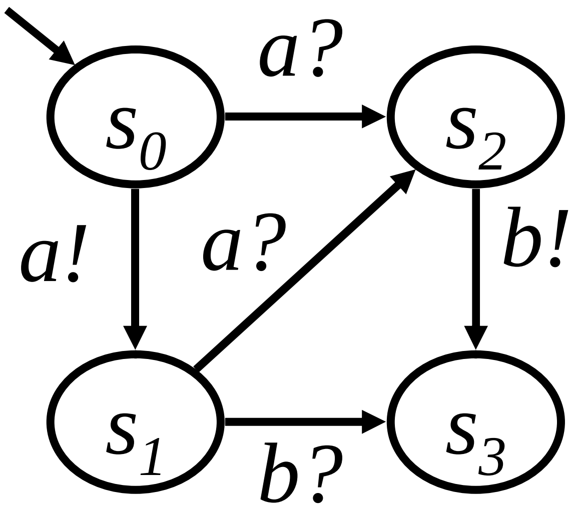
The *behavioural rules* guarding the consistency of message exchanges

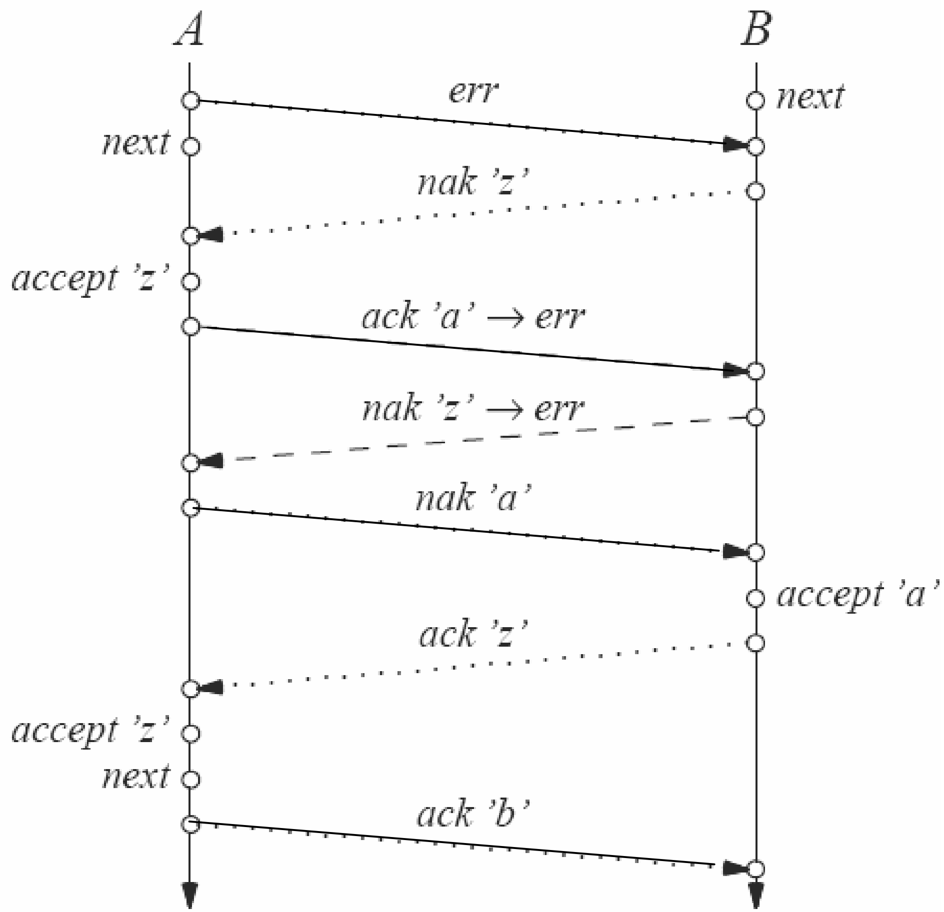
most difficult

Protocol Behaviour Notation

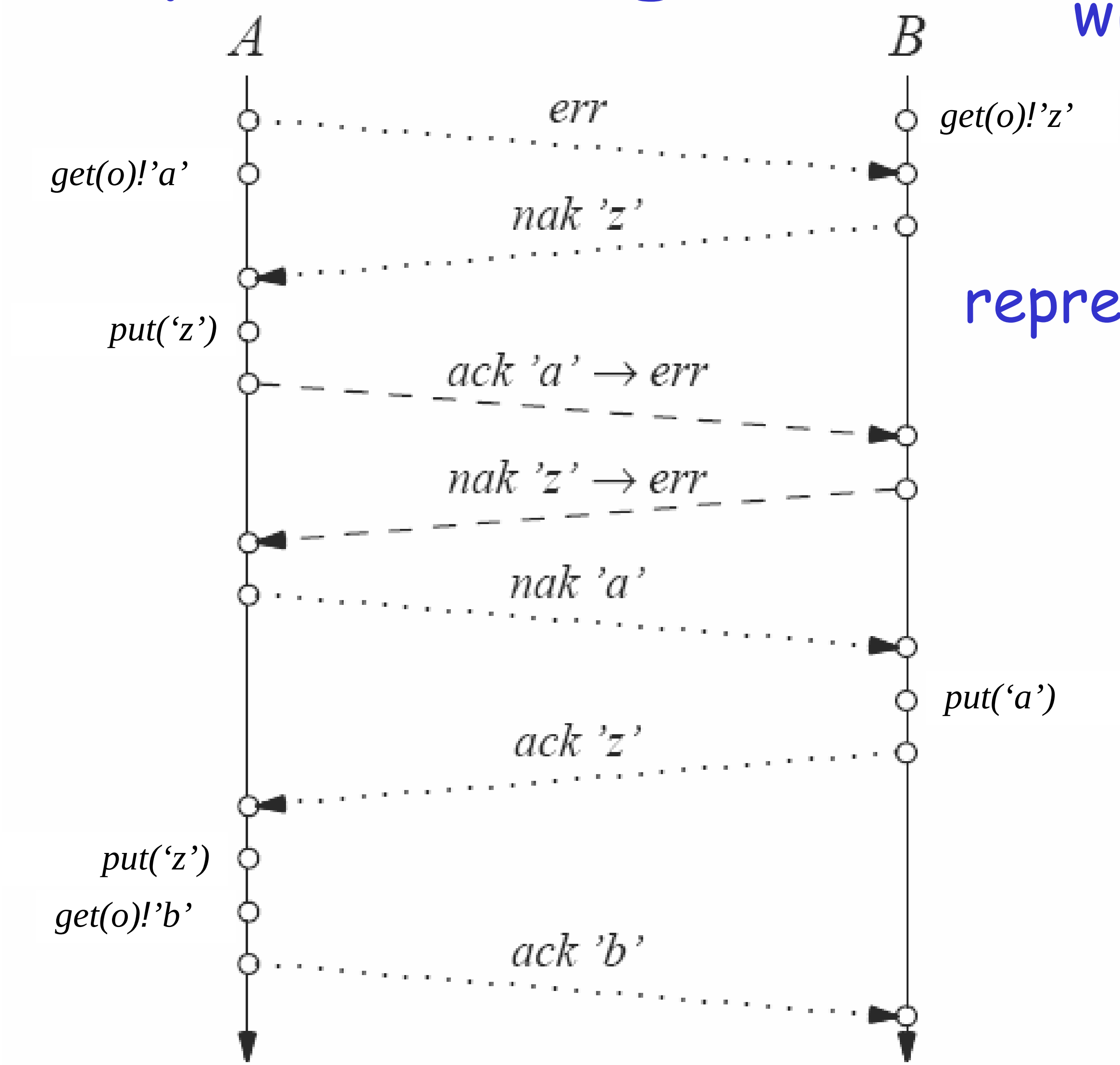
The unambiguos description of protocol behaviour is essential, but difficult. There is no universal notation. Frequently used are

- message sequence charts/ sequence diagrams/ use cases
- state-transition diagrams





Protocol Notation: Sequence diagrams

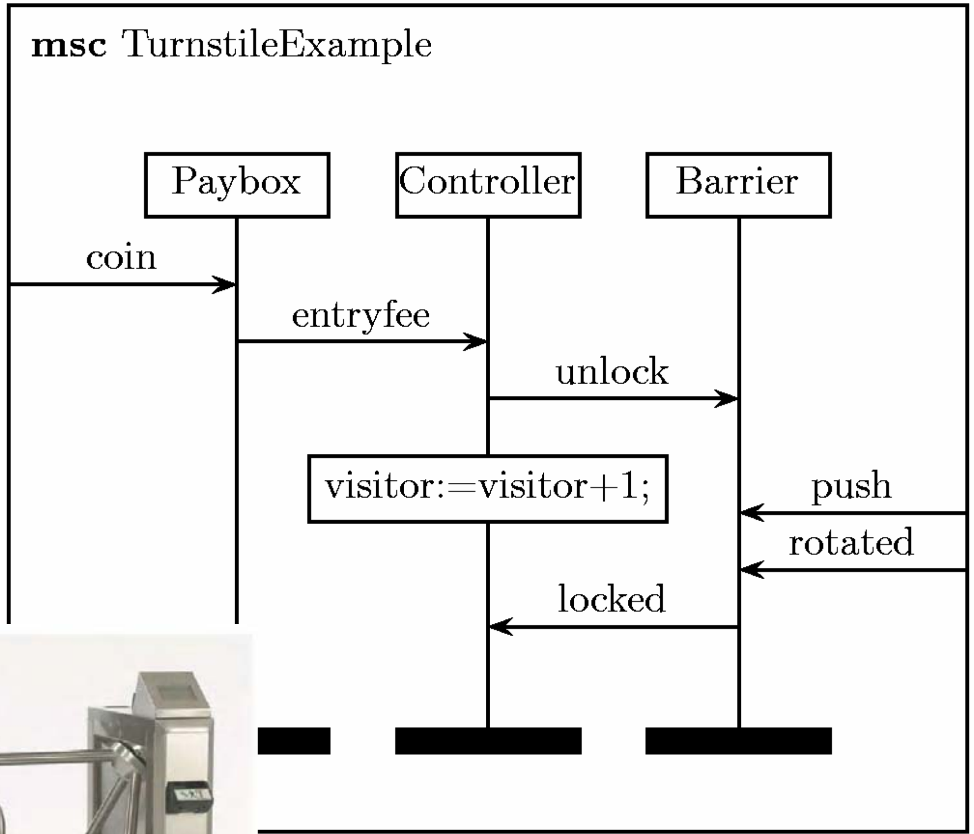


... are precedence graphs with locality information

each vertical line represents a protocol entity (or the environment)

arrows represent signals/messages

MSC: Message Sequence Charts



... are sequence diagrams

have been standardized by the ITU (International Telecomm. Union)

each vertical line represents a protocol entity (or the environment)

arrows represent signals/messages

blocks represent (internal) process activities

Definition: Basic MSC

- A (basic) MSC M is a tuple $(P,E,L,c,<)$
  - a set P of process labels (labelling the instance axis),
  - a finite set E events $E = S \cup R \cup A$, consisting of
    - send events S (buh/)
    - receive events R (/buh)
    - action events A (task executions etc)
  - a labeling function $L: E \rightarrow P$ (putting events on the instance axis),
  - a bijection $c: S \rightarrow R$ (for send-receive edges)
  - precedence relation $< \subseteq E \times E$
    - Send of a message occurs before its receipt
    - Events on the same instance are totally ordered
- Must be well-formed: no cycles in precedence graph

Semantics of Basic MSC

- [illegible] the transitive closure o[illegible]defines a partial order on E

- A trace of MSC M is a linearization of the partial order $<^*$.
  - every trace is a finite sequence of events that "obeys" the precedence.
  - each event occurs exactly once in a trace and only after all its preceding events have already occurred in the trace so far.
  - always finite.

- Semantics of MSC M
  - is the set of all possible traces.
  - can be represented as a finite LTS ([redacted]).

Need to introduce partial orders?

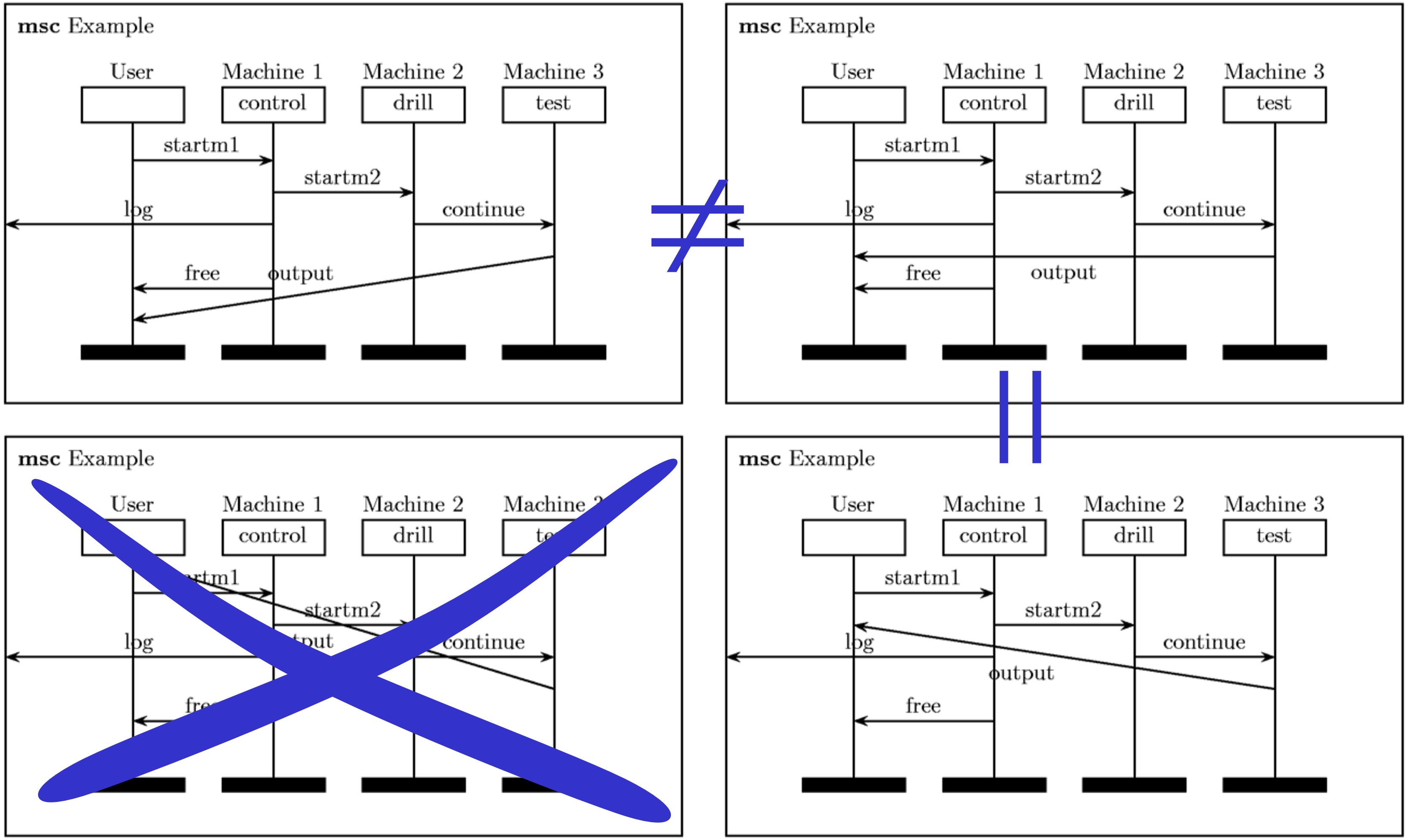
- A relation is a set of pairs drawn from some set, say E.
- A reflexive relation is a relation that contains the pair (e,e) for each element e of E.
- A transitive relation is a relation which contains the pair (e,g) whenever it contains both (e,f) and (f,g).
- A partial order is a reflexive and transitive relation.

The exercises of DN are partially ordered (in time).

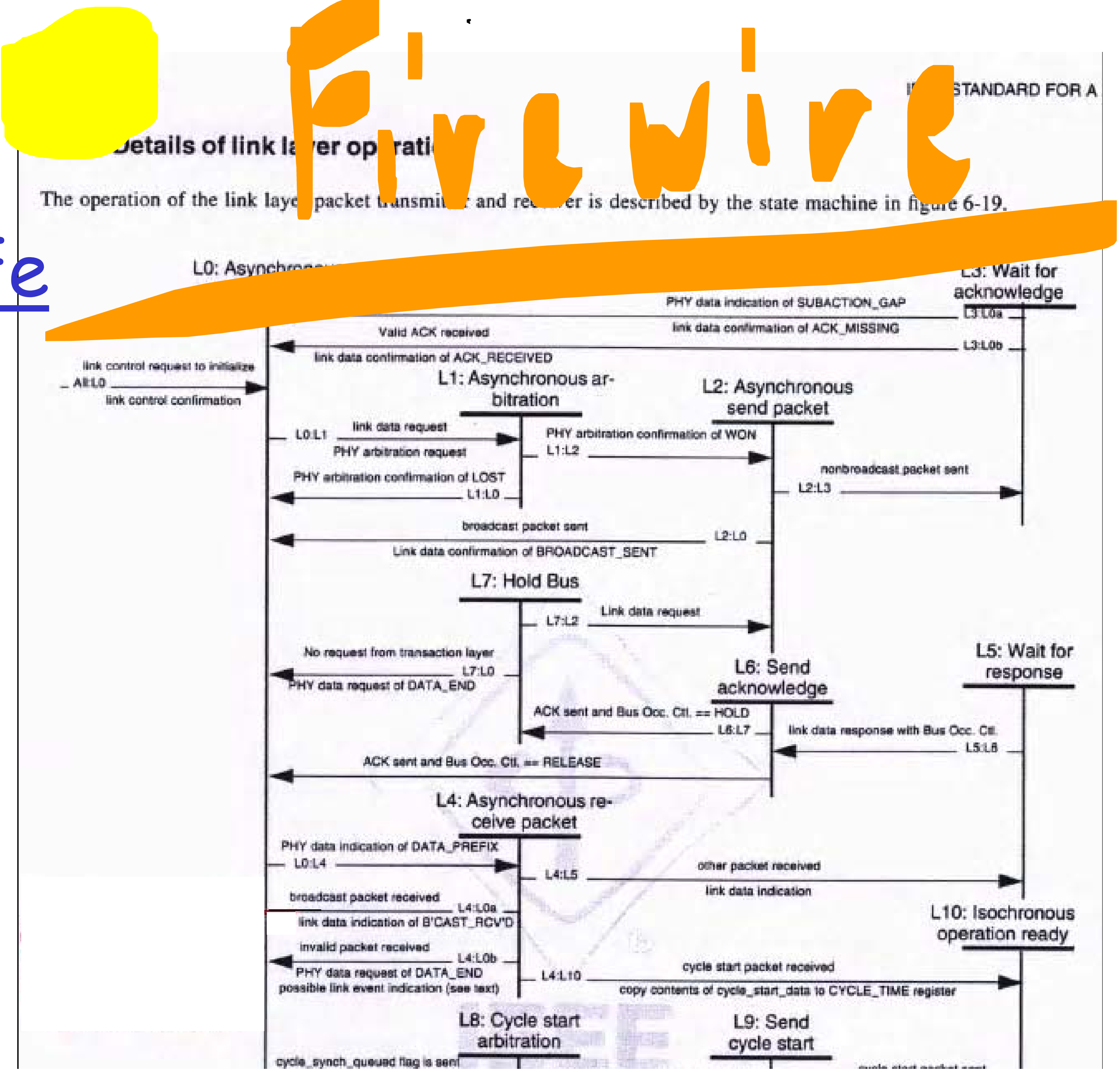
- A total order is a partial order which for each pair (e,f) of E (with $e \neq f$) does either contain (e,f) or (f,e) - but not both.

The lectures of DN are totally ordered (in time).

MSC - How simple!



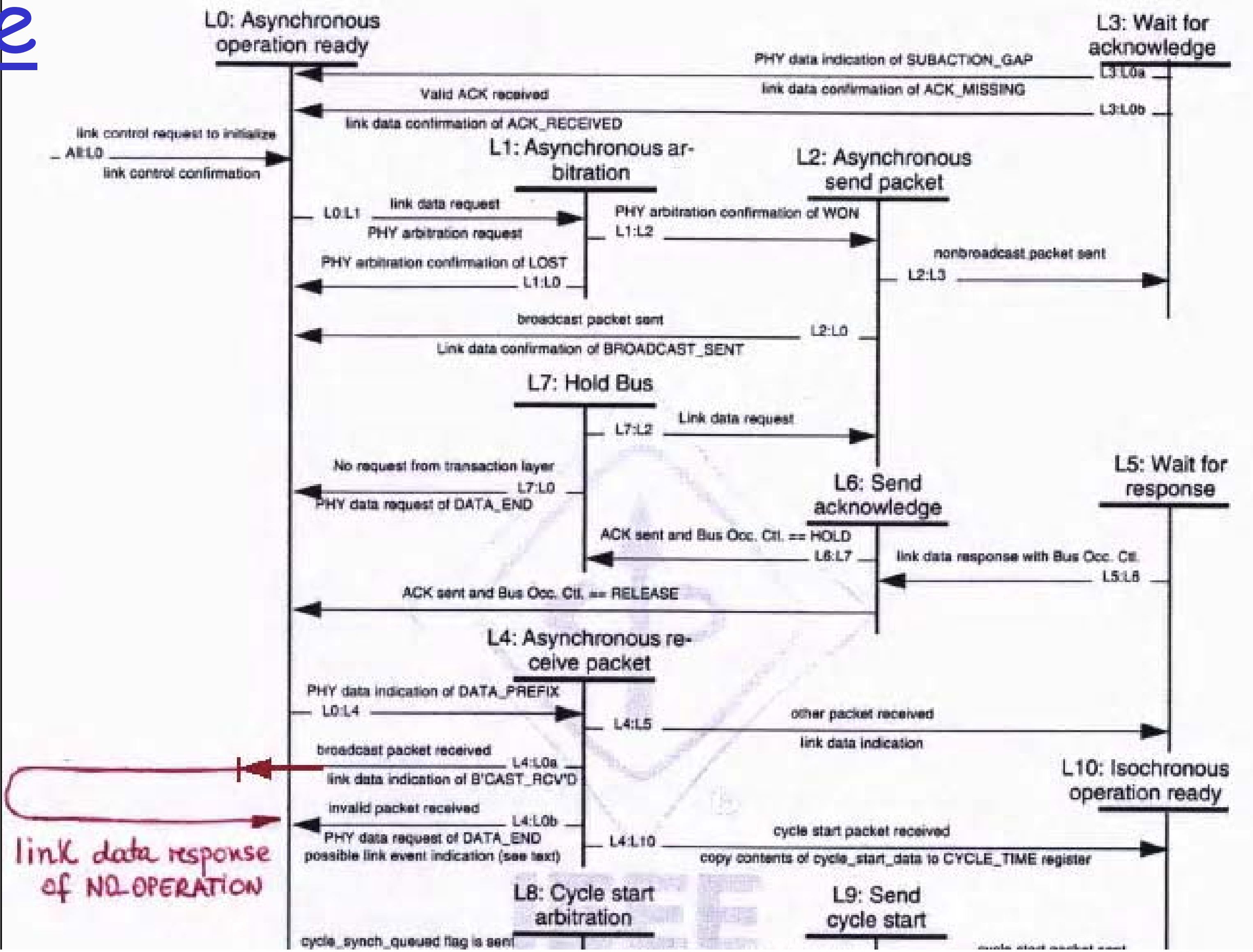
A real-life MSC



A real-life MSC

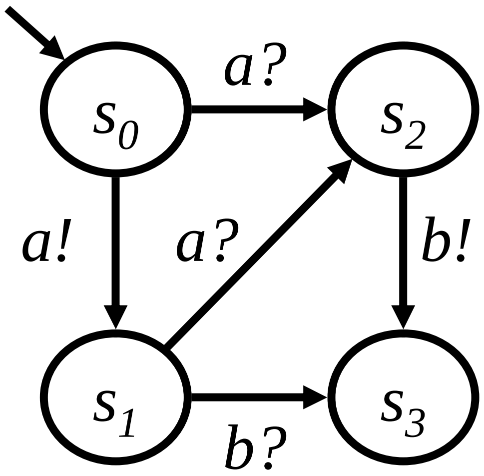
6.3.3 Details of link layer operation

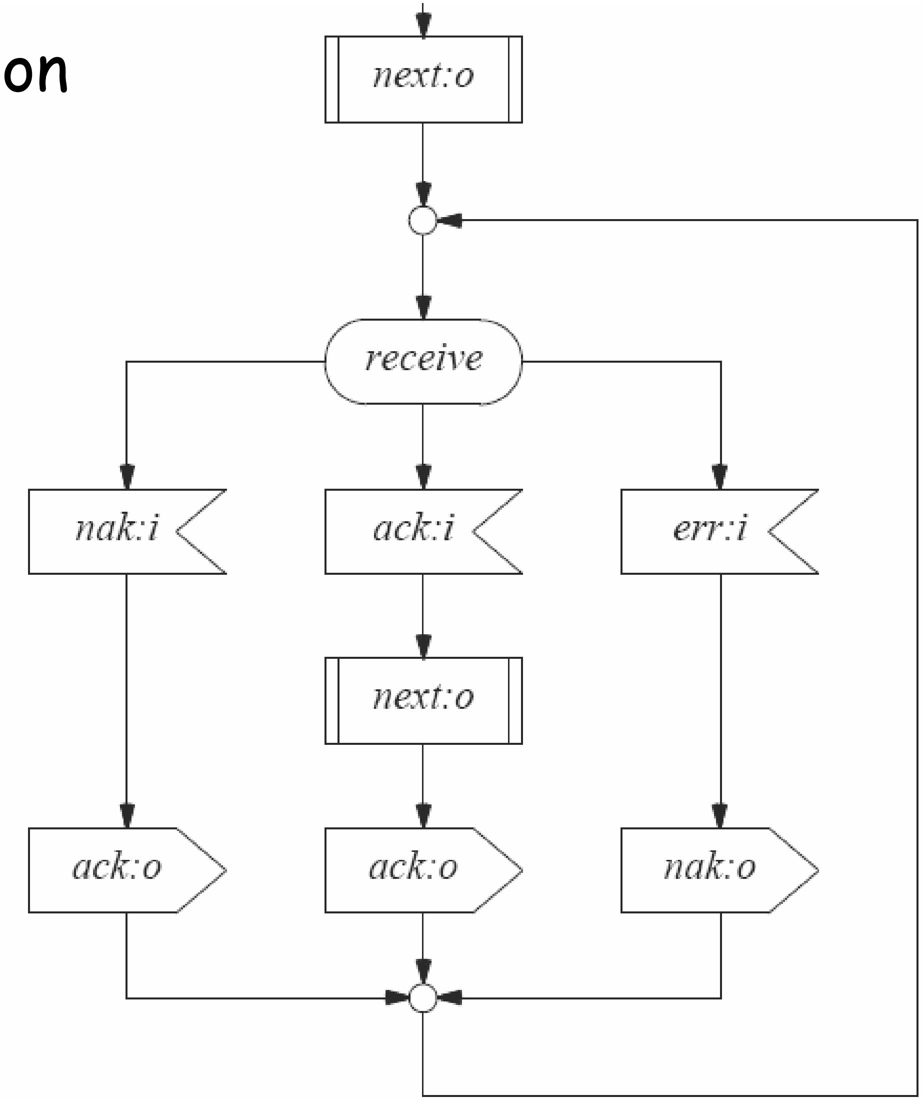
The operation of the link layer packet transmitter and receiver is described by the state machine in figure 6-19.



Protocol Notation: State transition diagrams

we call them labelled transition systems (LTS) in the sequel.



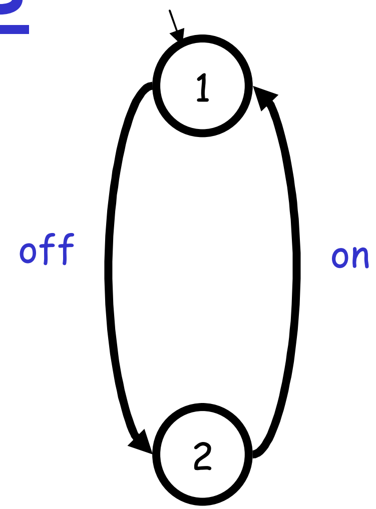


A flowchart like notation for LTS has been adopted by the ITU ('SDL').

Labelled transition systems

An LTS is a quadruple (S, L, T, s) where

- S is a set of states,
- L is a set of labels,
- T is a set of transitions,
  $T \subseteq (S \times L \times S)$,
- $s \in S$ is the initial state.



(a light switch)

Grandma's telephone

What can happen?

- Well, grandma can take off the phone from the hook, and put it back on the hook
- Grandma may spin the dial (these days: press buttons) to 'dial' a number.
- Also, grandma may witness the 'bell', a 'ring tone', a 'dial tone', a 'busy tone'.

- Anything forgotten?

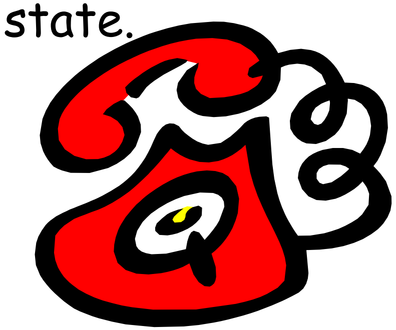
Grandma's telephone as an LTS

This is set L

- hookOFF
- hookON
- dial
- ring_toneON
- ring_toneOFF
- busy_toneON
- busy_toneOFF
- dial_toneON
- dial_toneOFF
- bellON
- bellOFF
- connect
- disconnect

An LTS is a quadruple (S, L, T, s) where

- S is a set of states,
- L is a set of labels,
- T is a set of transitions, $T \subseteq (S \times L \times S)$,
- $s \in S$ is the initial state.



Your first exercise

Complete the LTS describing grandma's telephone

- Do this with pencil and paper.
- Choose meaningful names for the states in S.
- You may assume that grandma can only call a single partner (which is ***you***, her grandchild, of course). This is performed (in one shot) with 'dial'.

Sequence Diagrams vs. State-Transition Diagrams

- Sequence diagrams show the interaction of protocol peer entities - by example (use cases), or by counterexample (misuse cases).

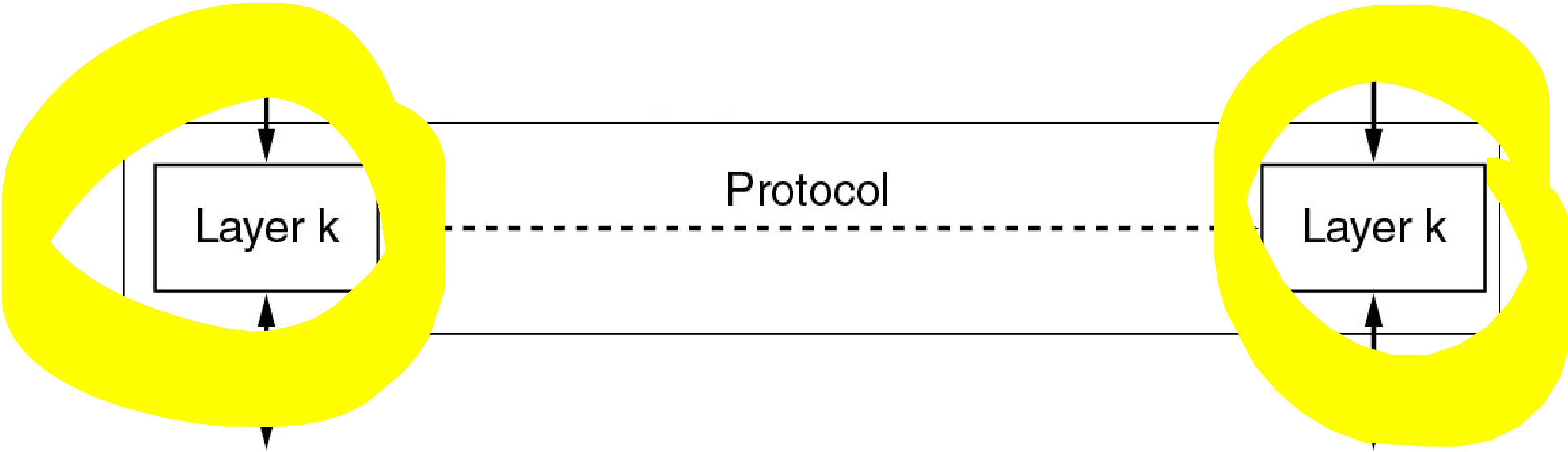
or: LTS

- State-transition diagrams show the behaviour of one protocol entity, possibly the complete behaviour.

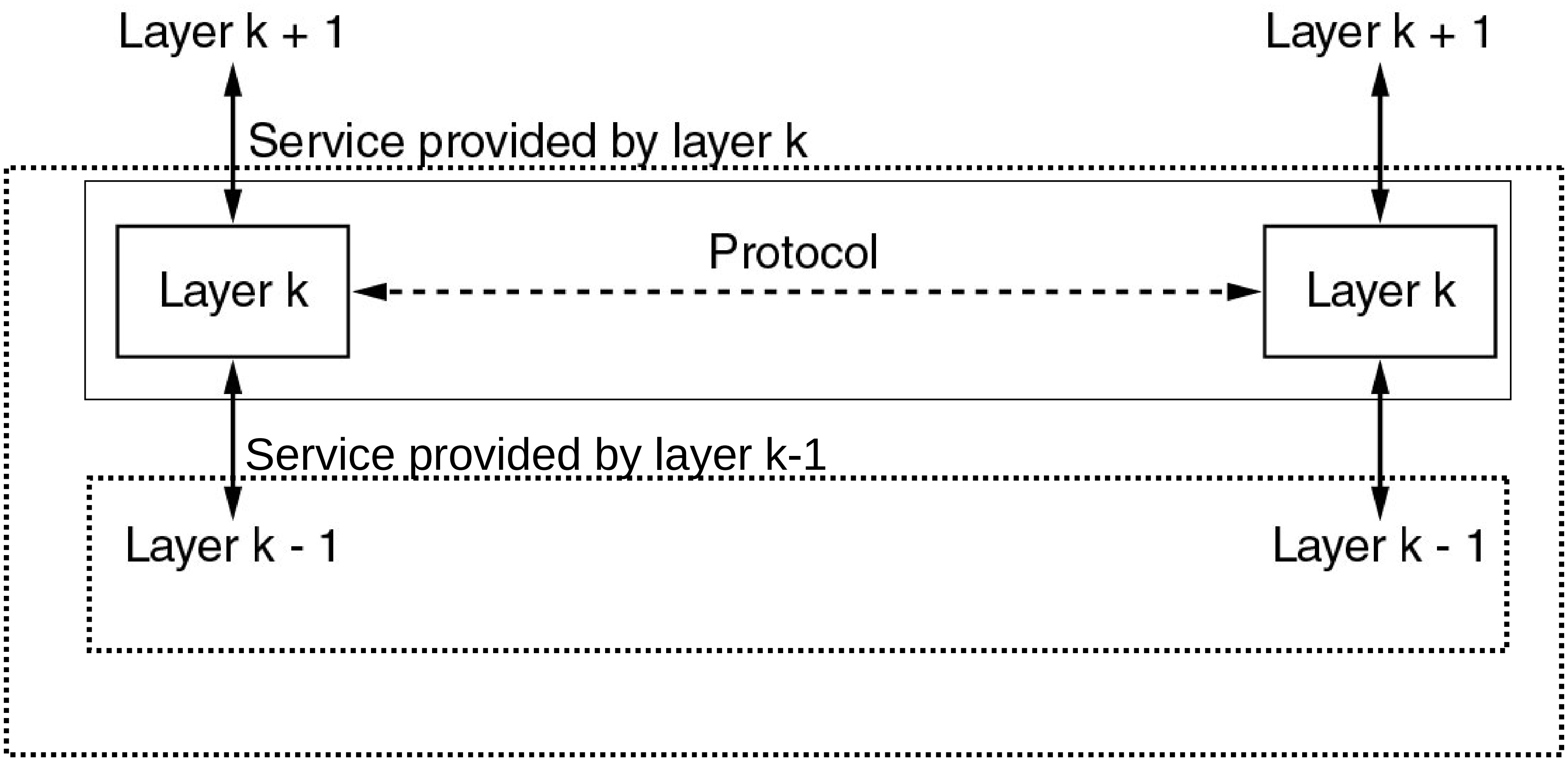
Events in Protocol Behaviour

- Protocol behaviour is driven by *events:*
  - arrival of signals/messages
  - timeouts
- Events induce *state* changes.
- A *state* is identified by the program location where the protocol entity waits for events (or generates events).

Protocol, Environment and Services



Protocol, Environment and Services



The five elements of a protocol

A protocol specification consists of five distinct parts. To be complete, each specification should include explicitly:

1. The *service* provided by the protocol
2. The *assumptions* about the environment in which the protocol is executed
3. The *vocabulary* of messages used to implement the protocol
4. The *encoding* (format) of each message in the vocabulary
5. The *behavioural rules* guarding the c
message exchanges

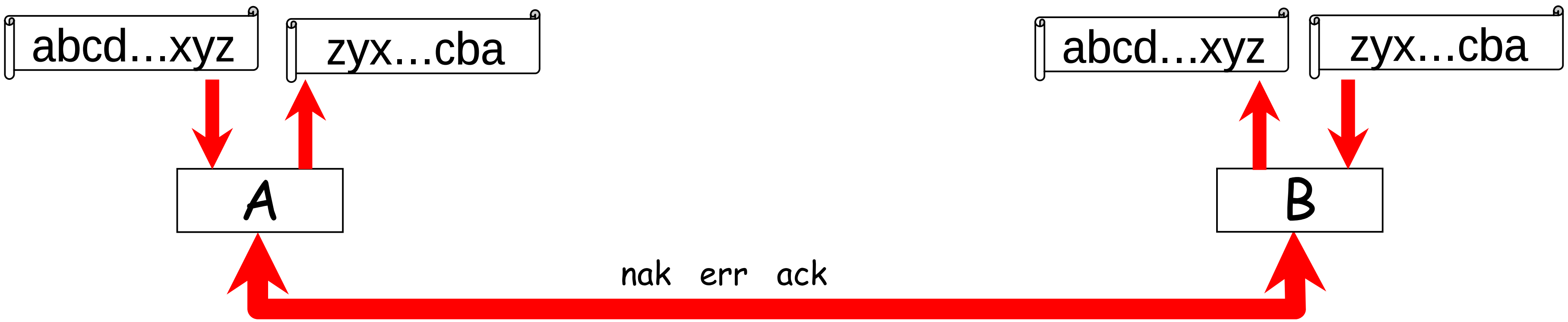
most diffic

1. Service

An example

The purpose of the protocol:

- *transfer text files* as *sequences of characters*
- *across a telephone line*
- *protect against transmission errors.*
- *in ``full-duplex" file transfer*, that is bidirectional simultaneously.



- positive and negative acknowledgments for traffic from *A* to *B* are sent on the channel from *B* to *A*, and vice versa.
- every message contains two parts:
  - a message part, and
  - a control part that applies to traffic on the reverse channel.

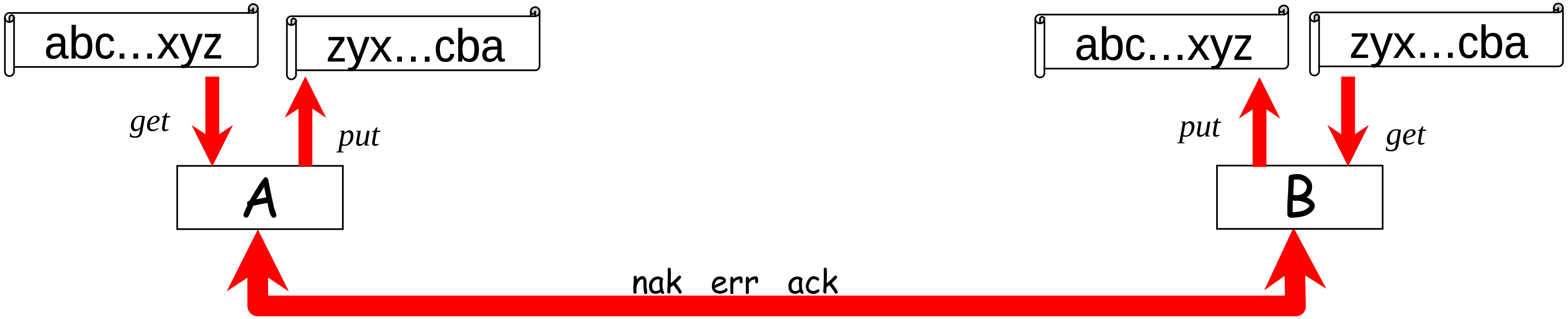
'piggybacking'

2. Environmental Assumptions

An example

The "environment" in which the protocol is to be executed consists of two users and a transmission channel.

- The users can be assumed to simply submit a request for file transfer and await its completion.



- The transmission channel is assumed to cause arbitrary message *distortions*, but *not to lose, duplicate, insert, or reorder* messages.

  We will assume that a lower-level module is used to catch all distortions and change them into undistorted messages of type '*err*'.

3. Protocol Vocabulary

An example

The protocol vocabulary defines three distinct types of messages:

- *ack* for a message combined with a positive acknowledgment,
- *nak* for a message combined with a negative acknowledgment, and
- *err* for a message with a transmission error.

The vocabulary can be succinctly expressed as a set:

$$V = \{ ack, err, nak \}.$$

4. Message Format

An example

Each message consists of a control field identifying

- the message type and
- a data field with the character code.

This gives a simple structure of two fields:

{control tag, data}

in a C-like notation:

```
enum control {ack, nak, err};

struct message {
enum control tag;
unsigned char data;
};
```

5. Behavioural rules

An example

The behavioural rules for the protocol are informally described as follows:

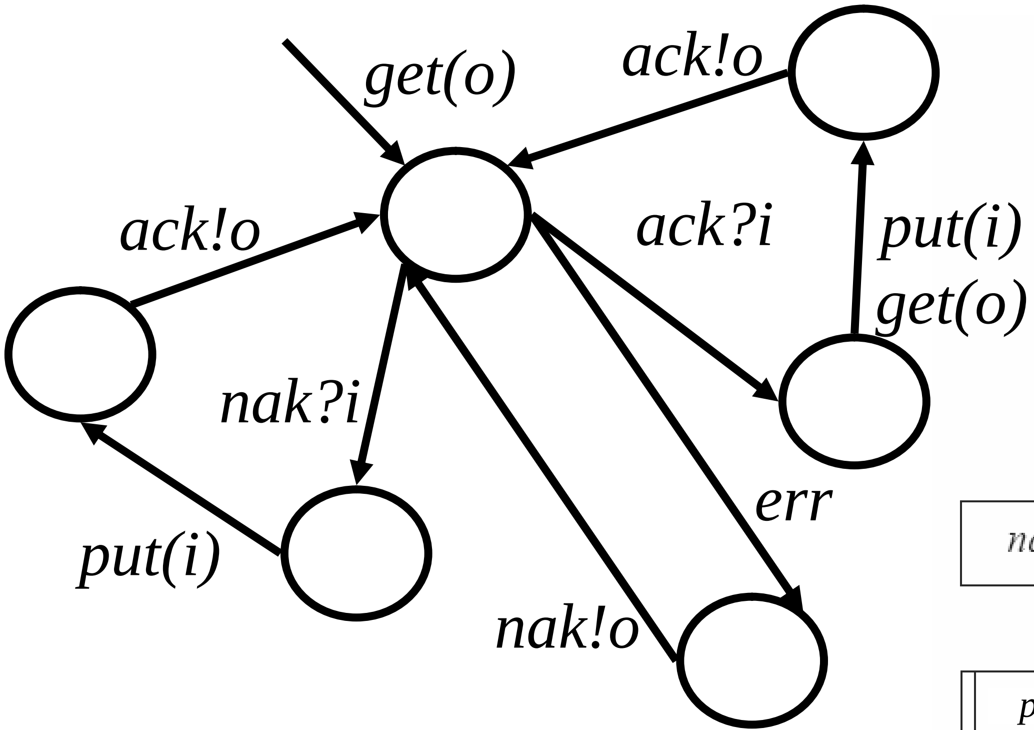
Control: If

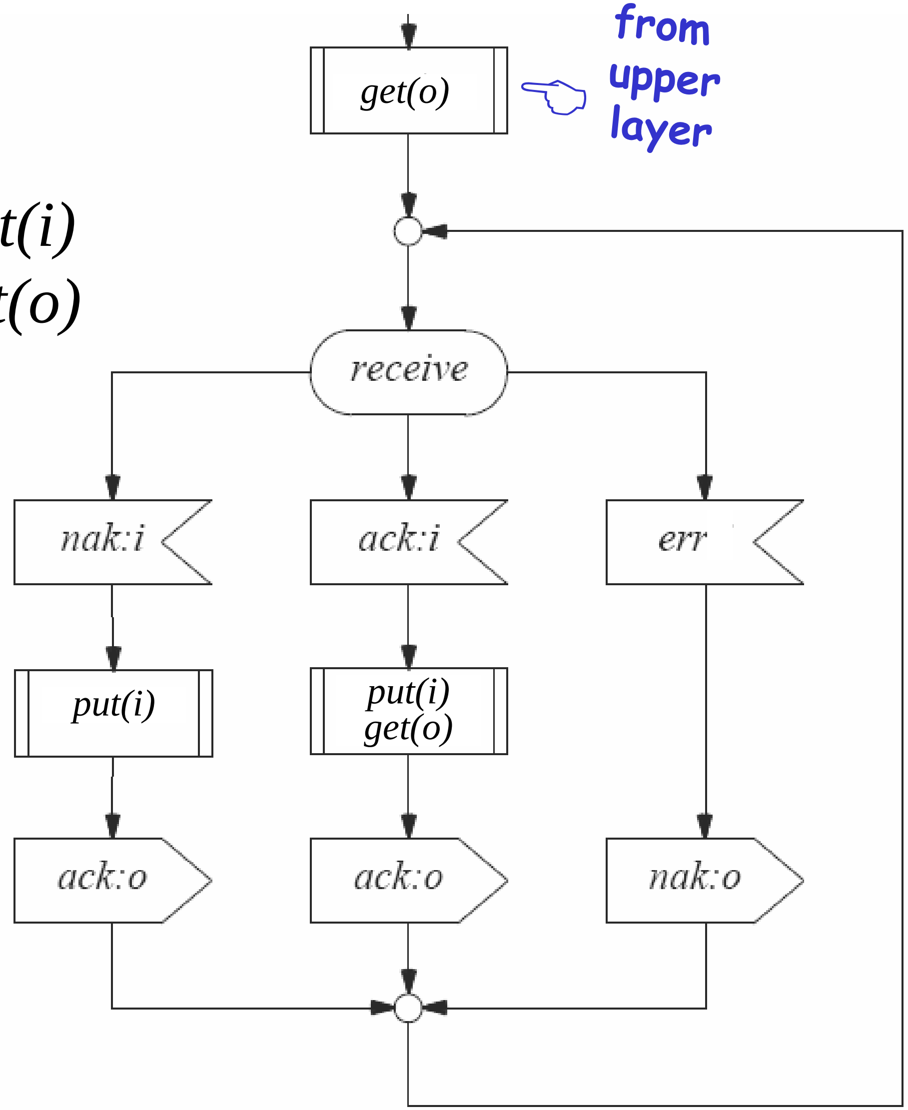
- *the previous reception was error-free,*
  - *the next message on the reverse channel will carry an 'ack' ;*
- *the previous reception was in error,*
  - *it will carry a 'nak'.*

Data: If

- *the previous reception carried a 'nak', or*
  the previous reception was in error,
  - *retransmit the old message;*
- *otherwise ('ack') fetch a new message for transmission.*

Procedure Rules as Diagrams



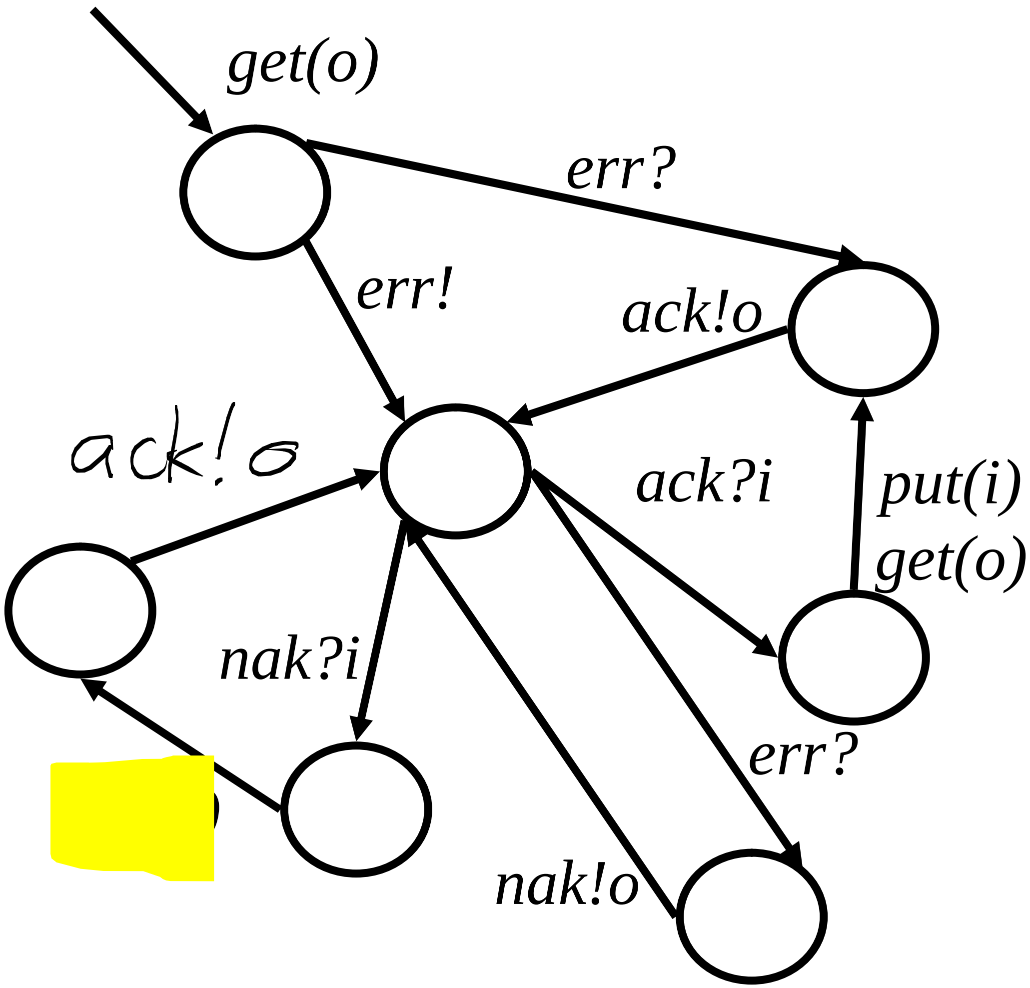


accept next event from upper layer

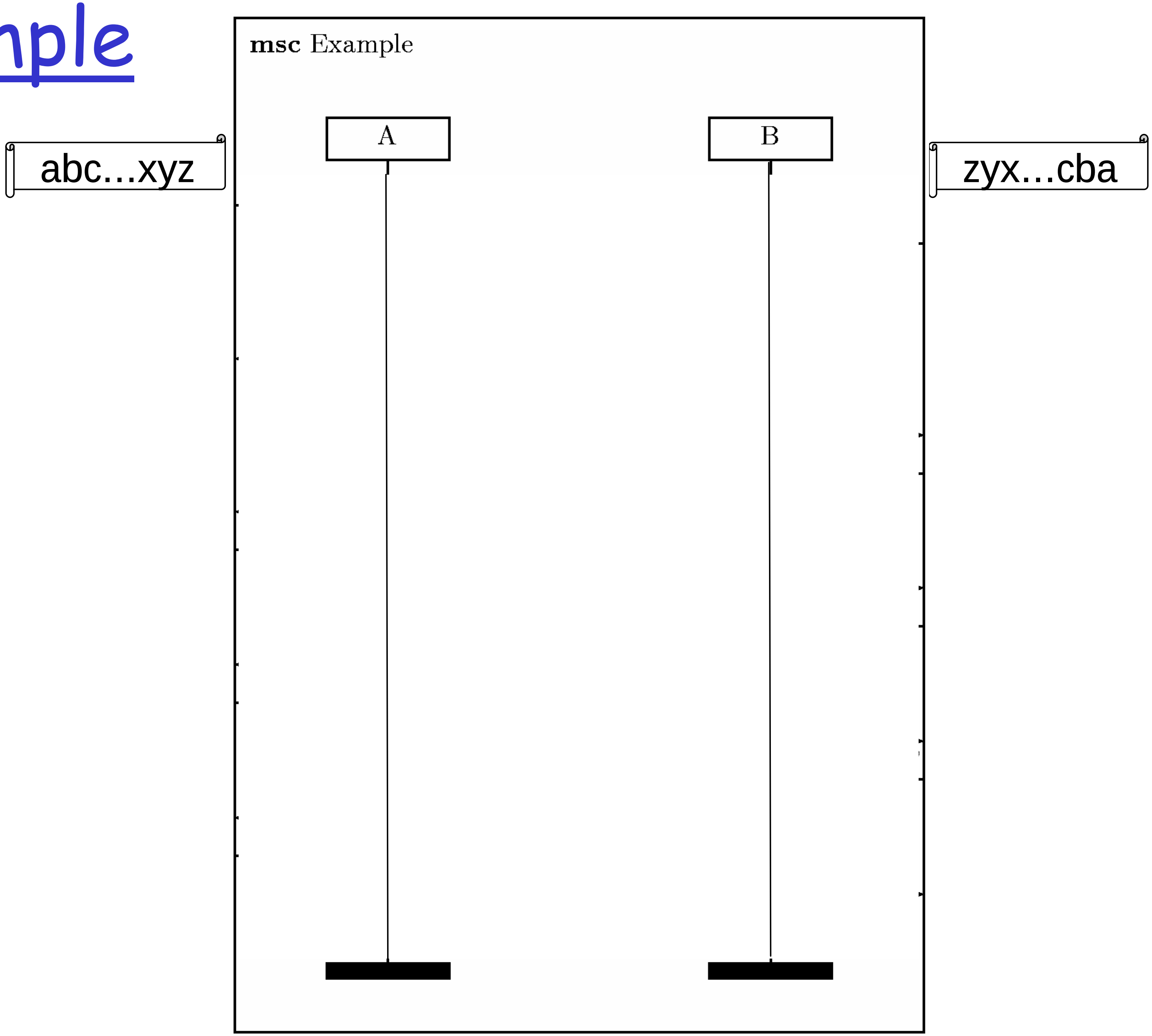
Design Flaws

- The above simple, informal protocol description is convincing, yet the protocol has several flaws.
  - Data flows only if both processes have something to send
  - The protocol does not terminate.

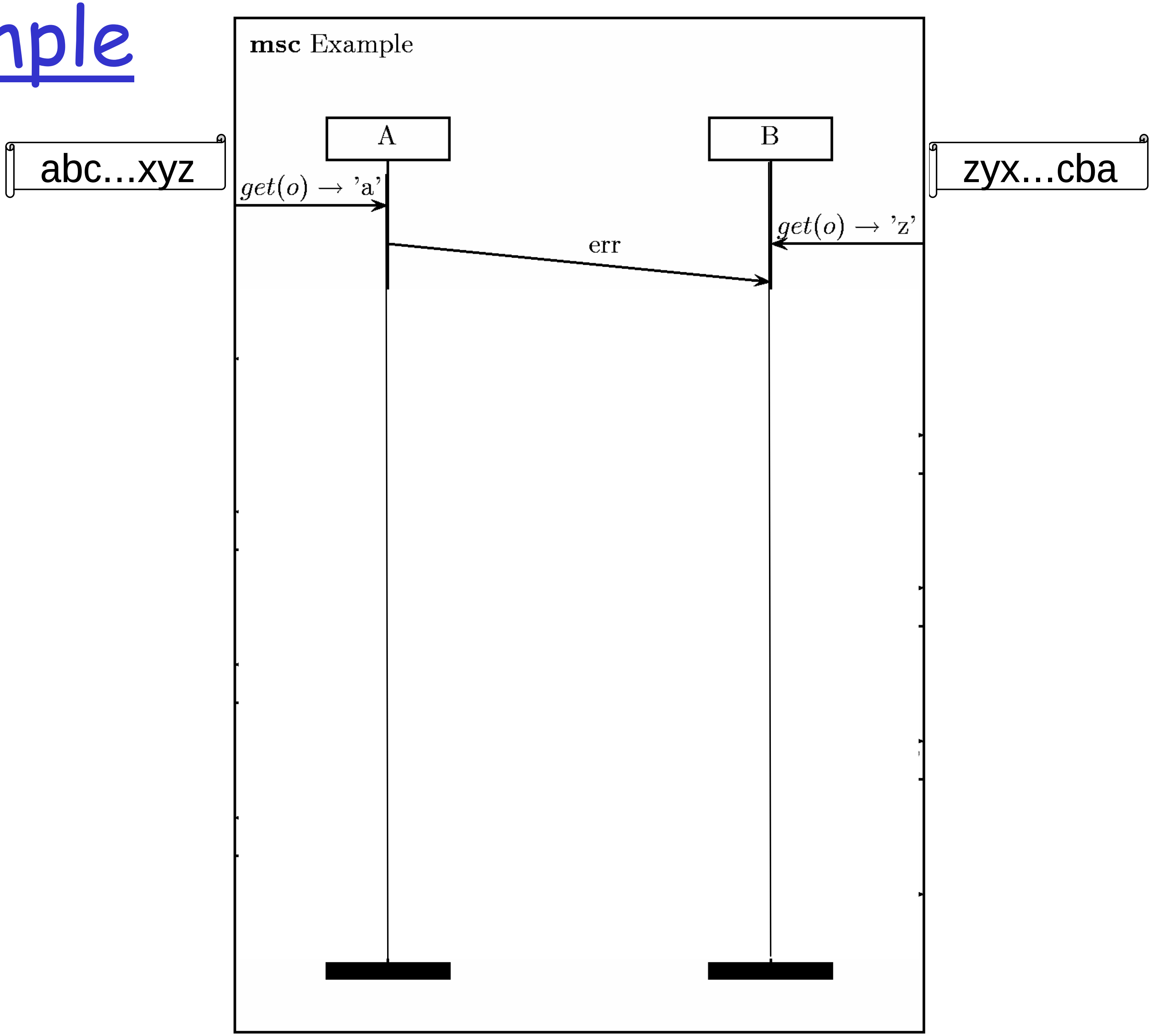
As a quick fix, let's assume we can start up the protocol by faking an error message if we have data to send


get(o)
err?
err!
ack!o
ack!o
ack?i
put(i)
get(o)
nak?i
err?
nak!o

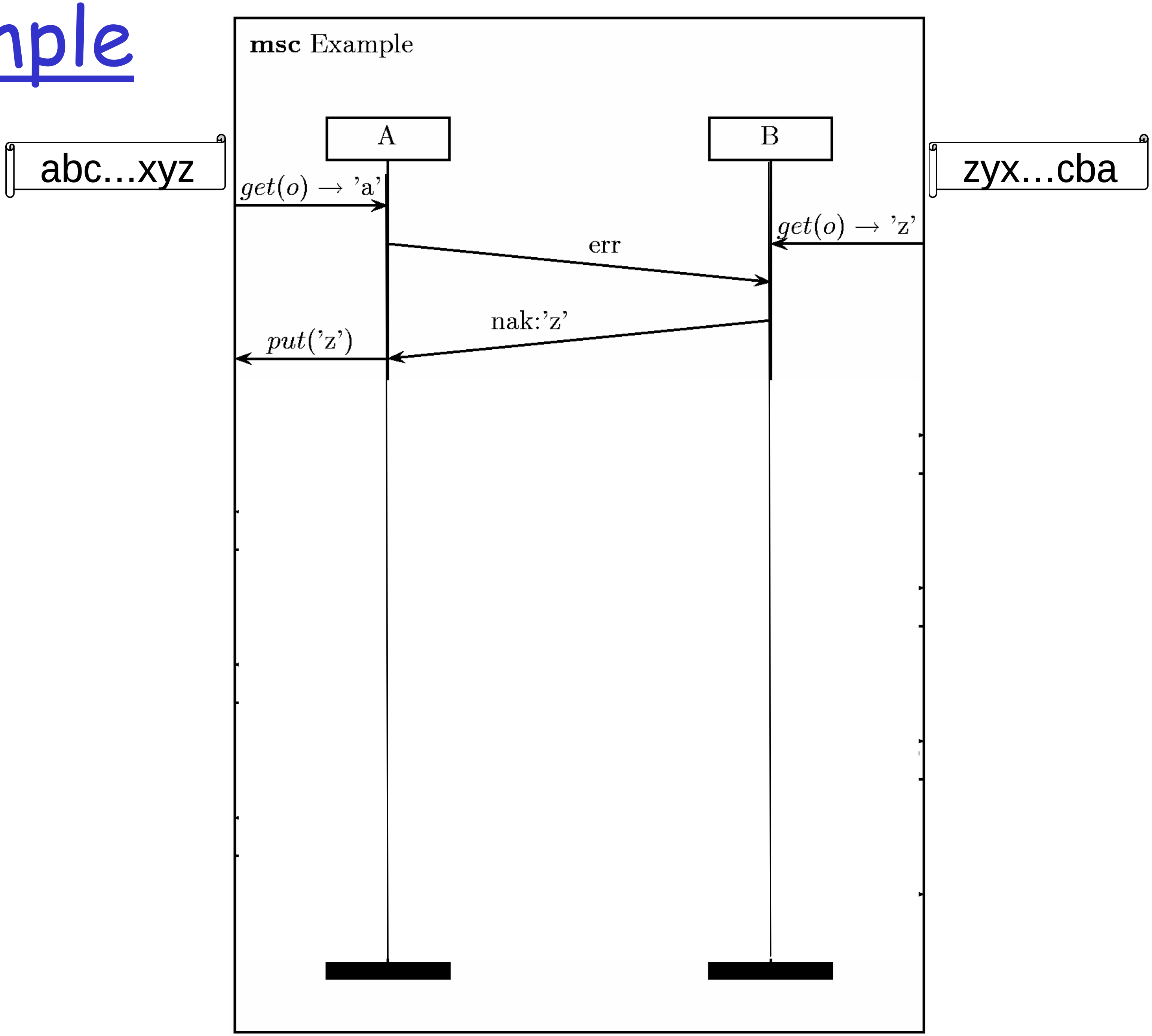
Example runs



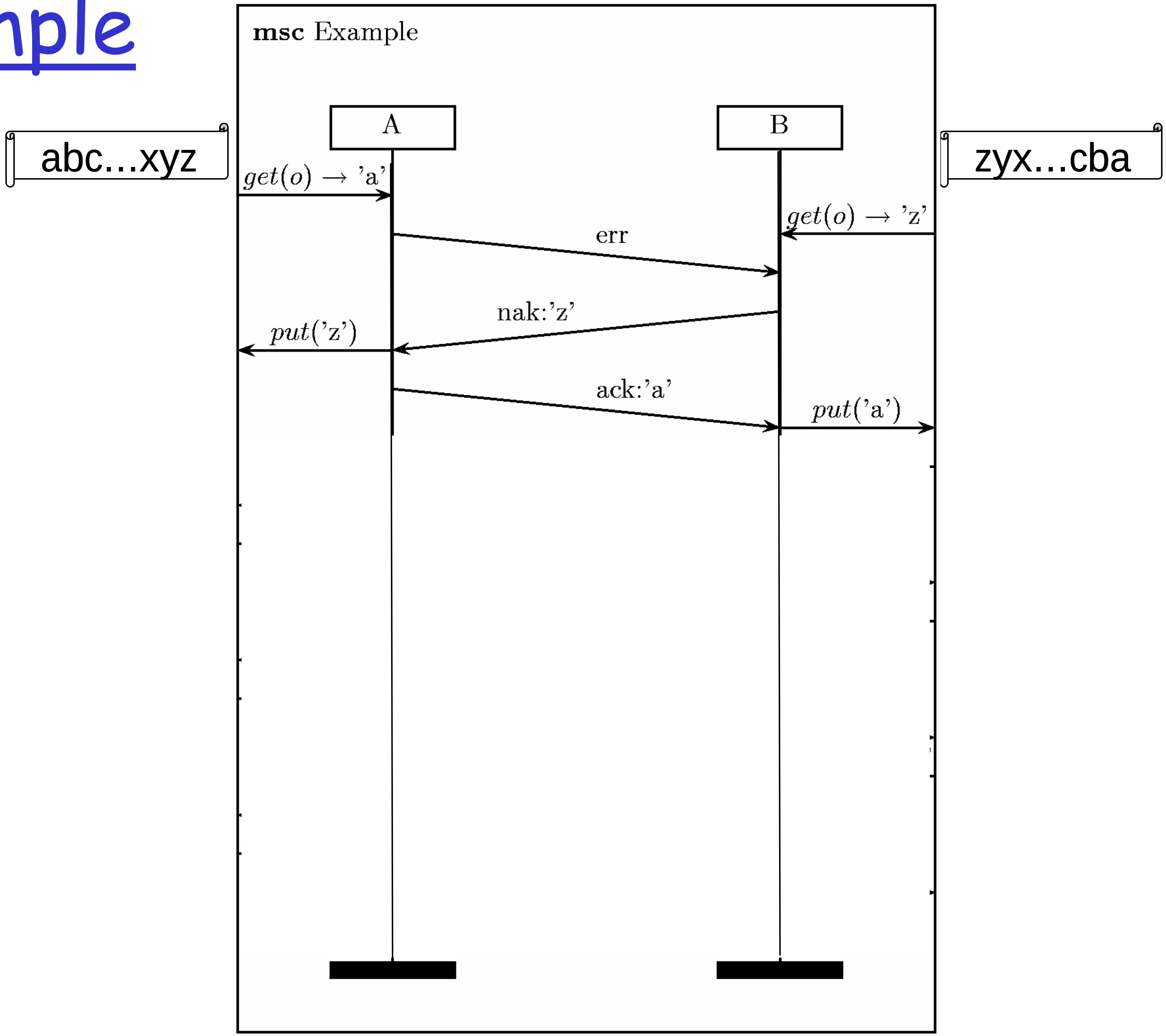
Example runs



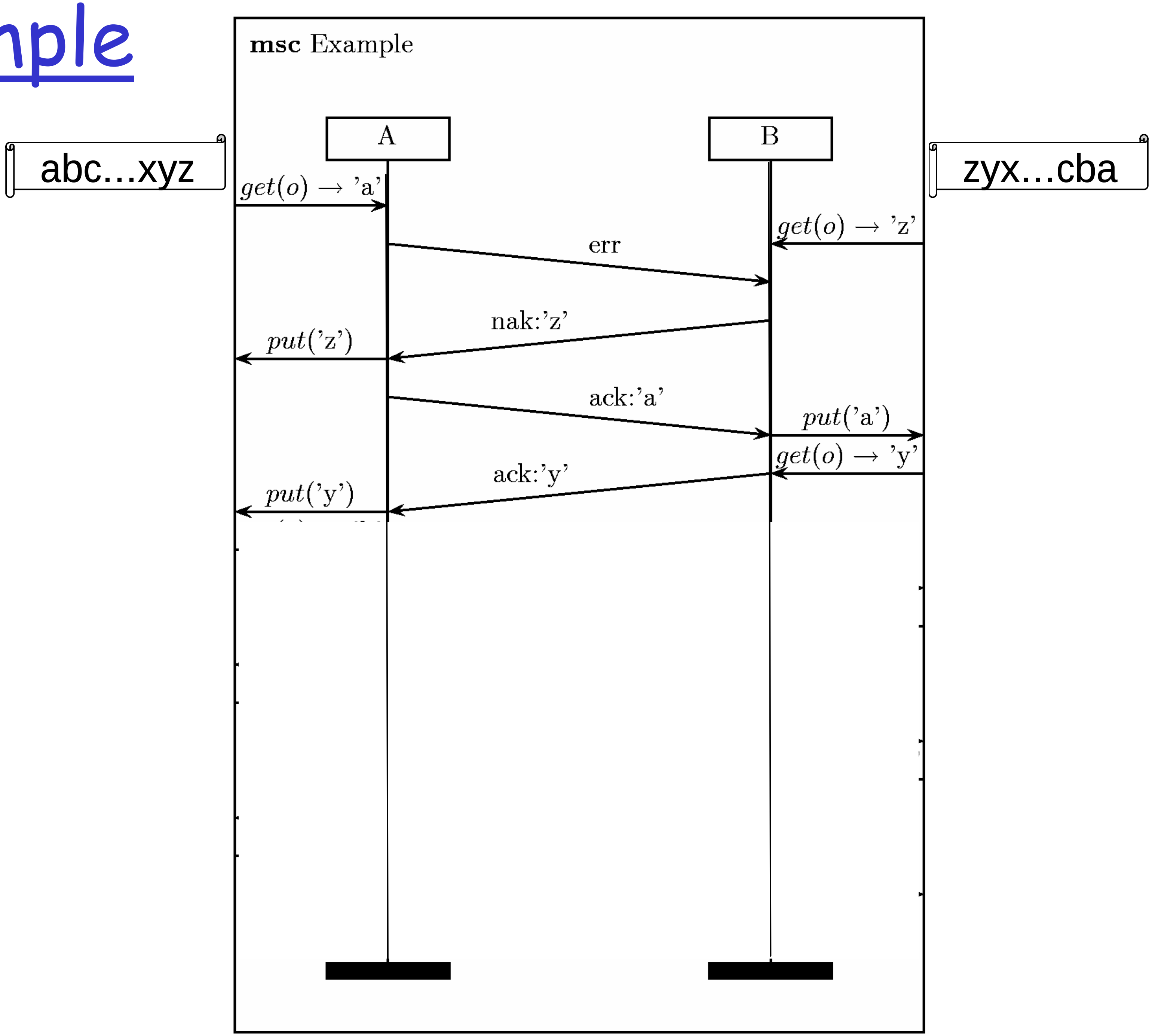
Example runs



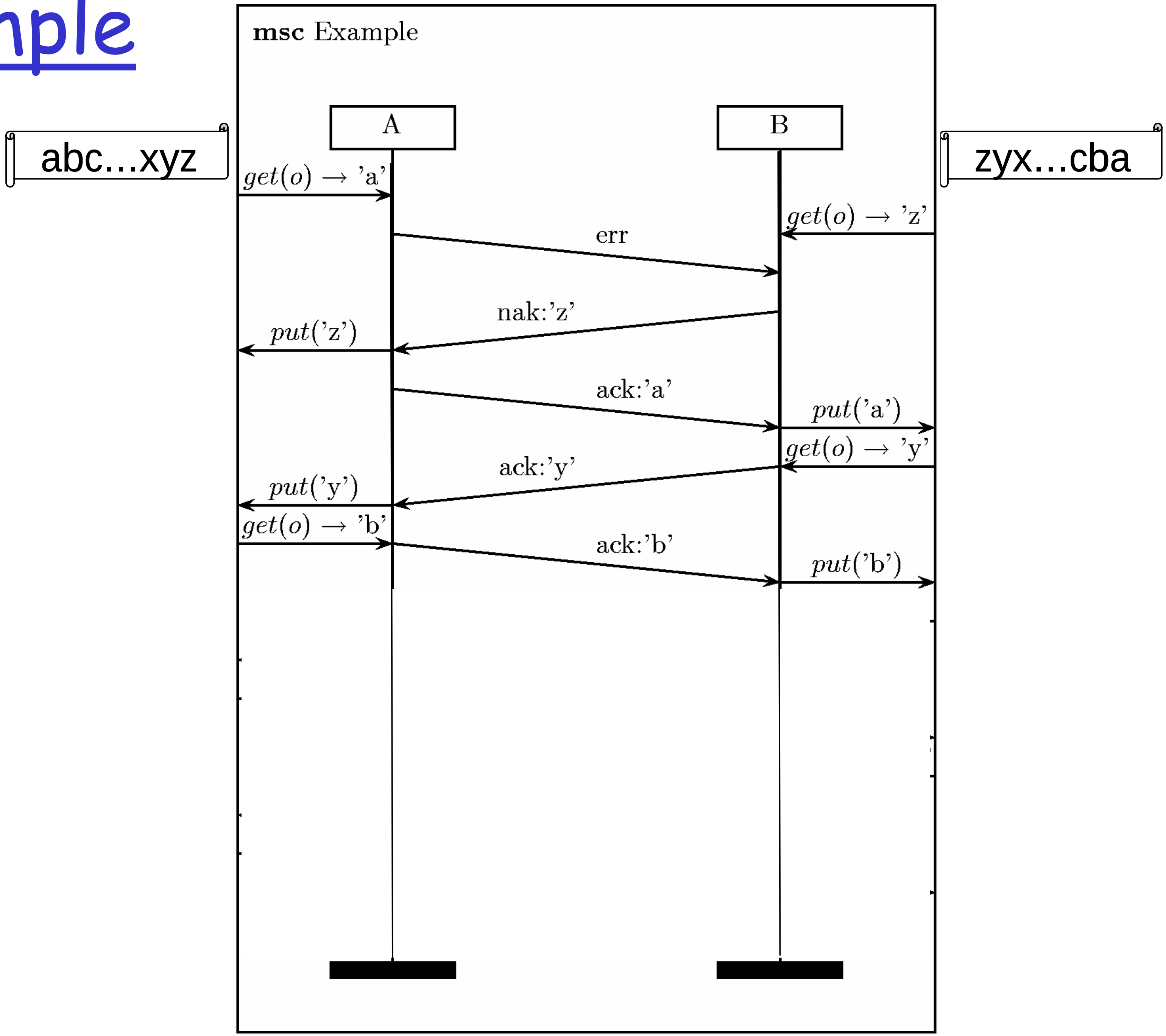
Example runs



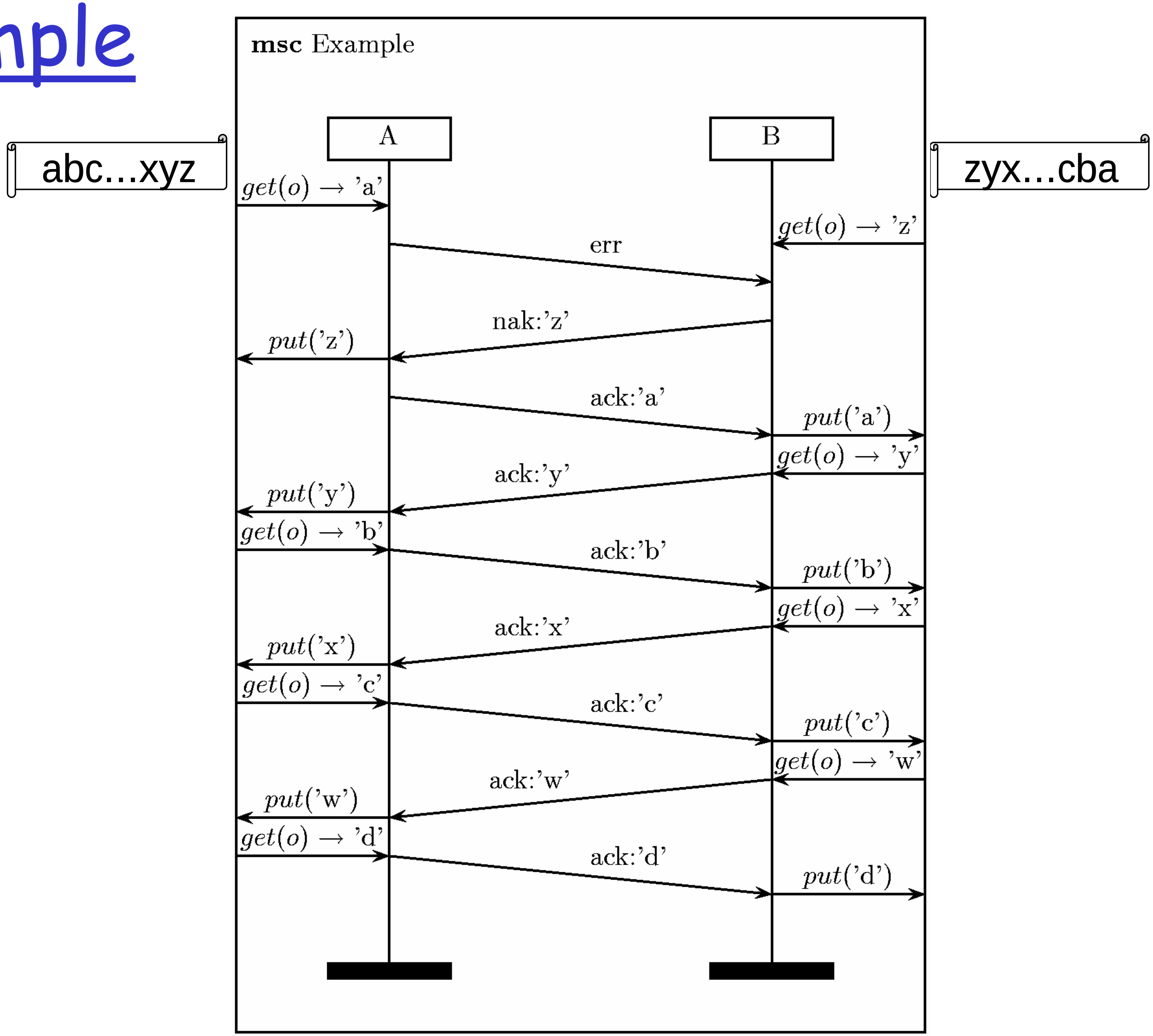
Example runs



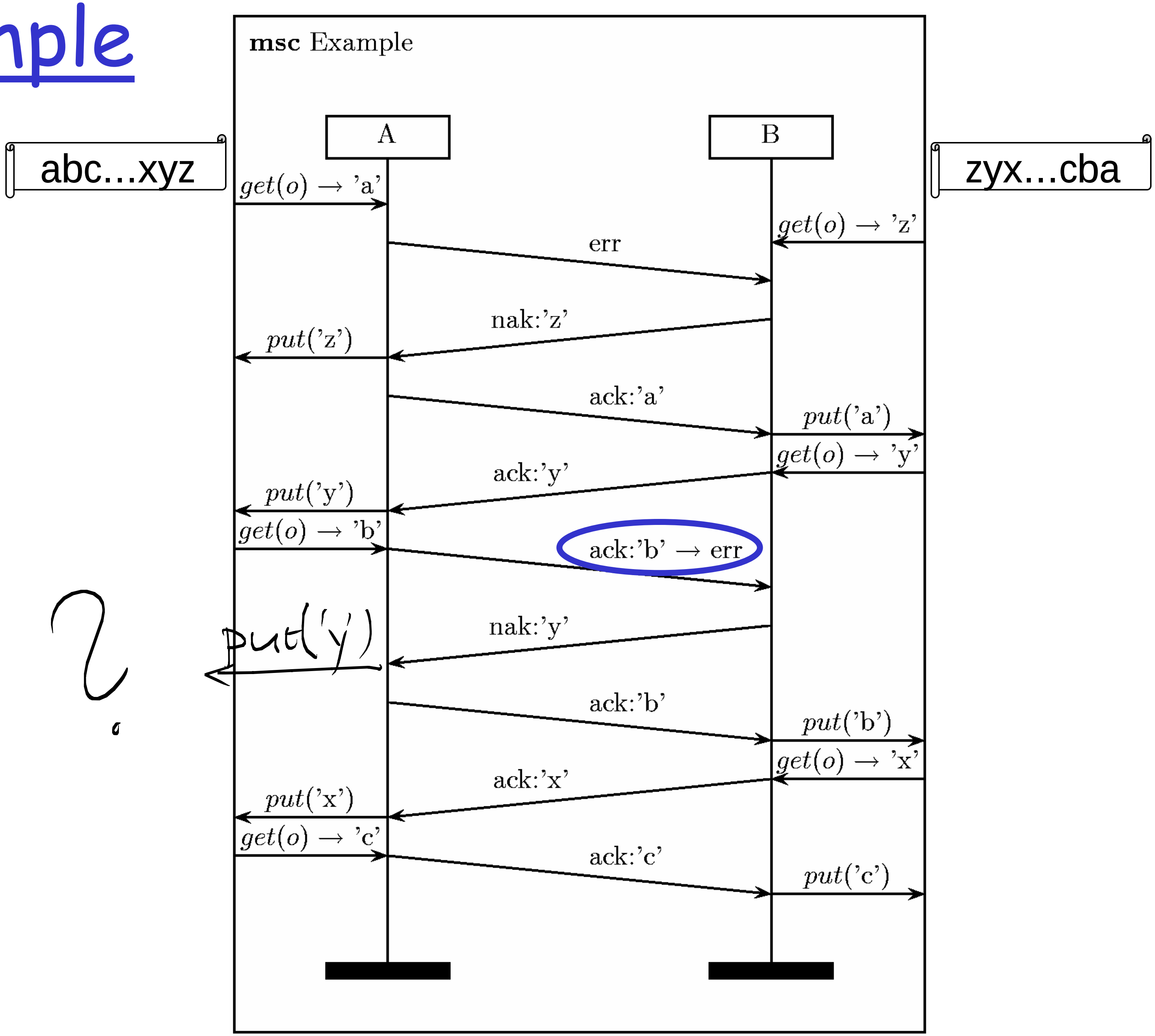
Example runs



Example runs



Example runs

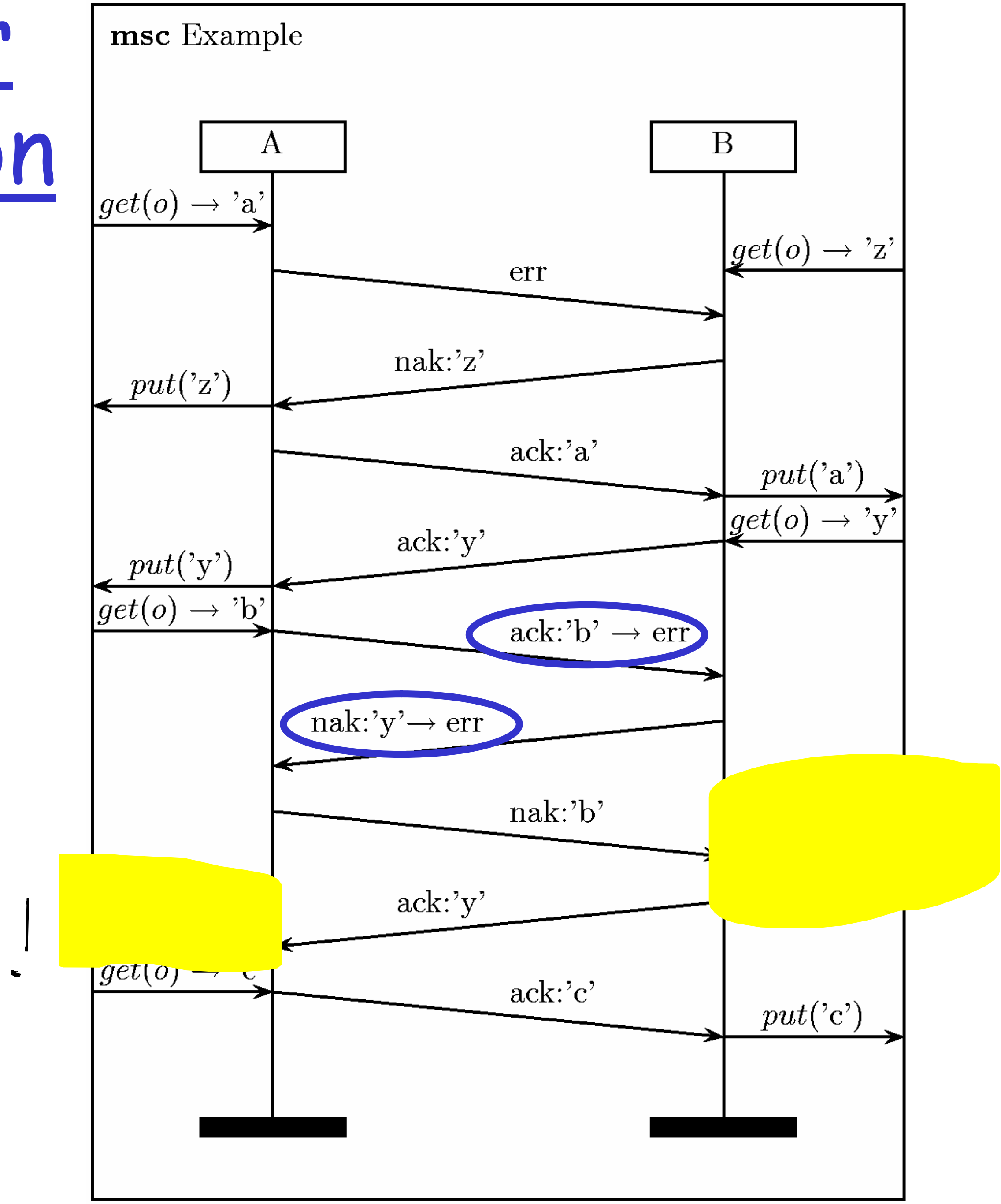


Design Flaws

- The above simple, informal protocol description is convincing, yet the protocol has several flaws.
  - Data flows only if both processes have something to send.
  - The protocol does not start up.
  - The protocol does not terminate.
  - The protocol may not deliver the correct message, it may duplicate characters.

  ...

Character Duplication

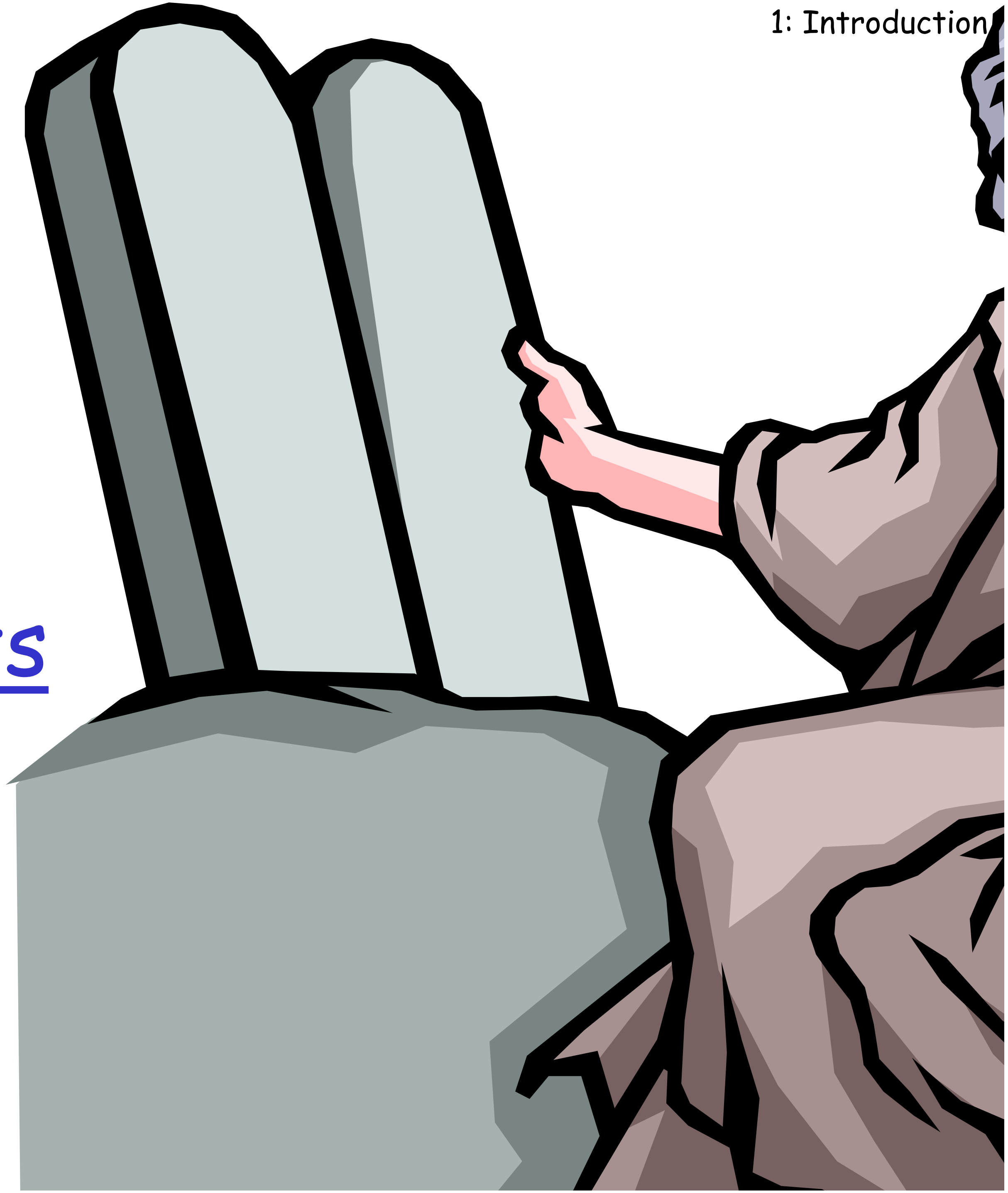


And now

for the

ten

commandments

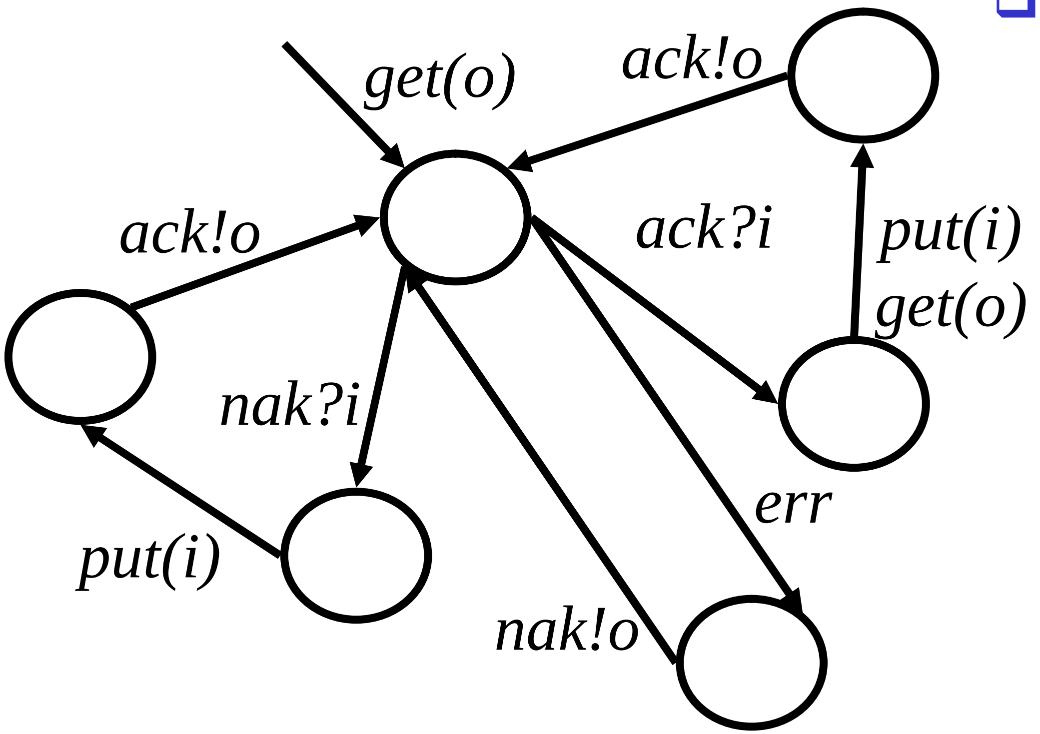


Ten rules of protocol design

1. Make sure that the problem is well-defined. All design criteria, requirements and constraints, should be enumerated before a design is started.
2. Define the service to be performed at every level of abstraction before deciding which structures should be used to realize these services (*what* comes before *how*).
3. Design *external* functionality before *internal* functionality.
4. Keep it simple.
5. Do not connect what is independent. Separate orthogonal concerns.
6. Do not introduce what is immaterial. Do not restrict what is irrelevant.
7. Before implementing a design, build a high-level prototype and verify that the design criteria are met.
8. Implement the design, measure its performance, and if necessary, optimize it.
9. Check that the final optimized implementation is equivalent to the high-level design that was verified.
10. Don't skip Rules 1 to 7.

The most frequently violated rule, clearly, is Rule 10.

A high-level prototype



- And a formal language to generate such prototypes

```
proc Dumb(){
get(o);
  do{
    alt{
      ::ack?i ;
          put(i);
            get(o);
              ack!o;
      ::err;
          nak!o;
      ::nak?i ;
          put(i);
            ack!o;
    }
  }
}
```

How to validate such protocols

1. Formalise the *five elements of a protocol* in a formal notation.

2. Unless you dare a manual proof, let a tool explore all possible event sequences and check for inconsistencies

   ☞ 'model checking'

3. if you think you cannot do 2., do 1. anyhow

- Typical inconsistencies:
  - unspecified message arrivals
  - safety violations (something bad happens),
    - e.g. deadlock (protocol stops unintentionally)
  - liveness violations (nothing good happens),
    - e.g. livelock (protocol entities continue to exchange messages, but no service is provided)