

Sequence Diagrams

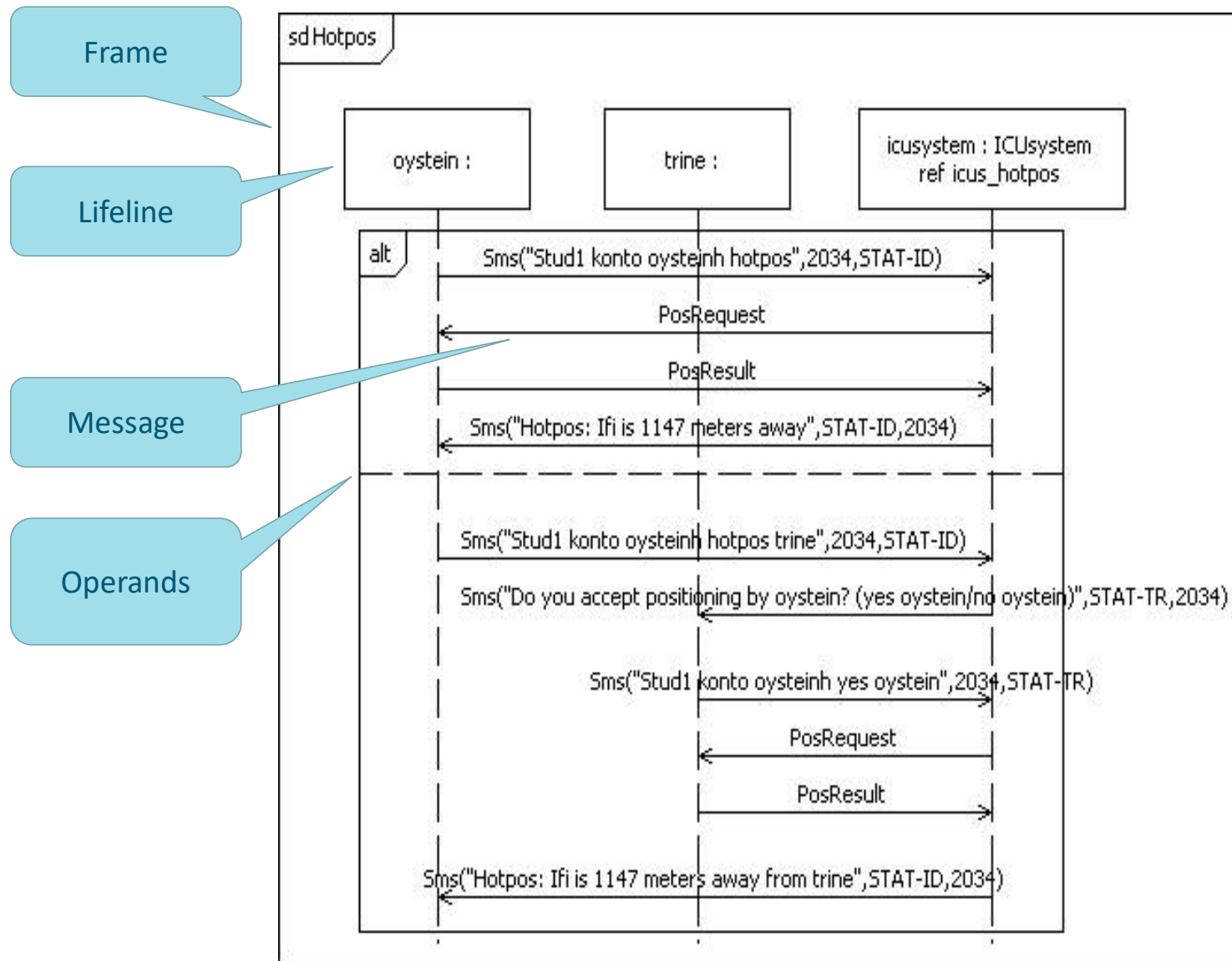
Ketil Stølen

Partly based on slides prepared by Prof. Øystein Haugen, HiØ

Overview of lecture

- Sequence Diagrams
 - What are they for?
 - What is their role in software engineering?
- Basic sequence diagrams
- Interaction Overview Diagram
 - Structuring mechanisms

This is a Sequence Diagram



Exercise: What makes sequence diagrams *fundamentally different* from program code?

Sequence Diagrams in a nutshell

- Sequence Diagrams are
 - simple
 - powerful
 - readable
- Emphasizes the interaction between objects
- Describes often only a small portion of the total behavior
 - improves the individual understanding of an interaction problem

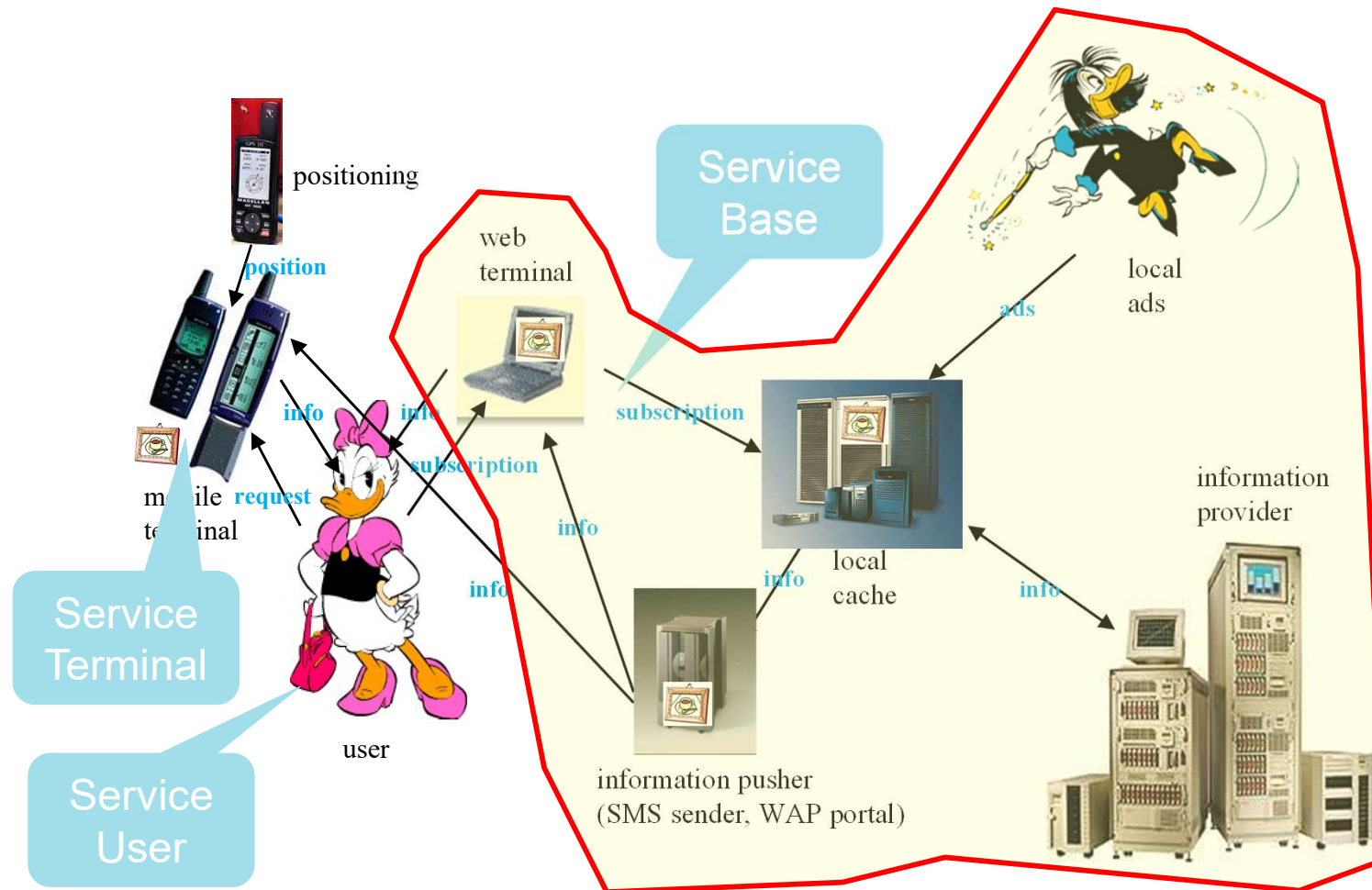
Sequence Diagrams are used to ...

- document protocol situations
- exemplify behavior situations
- verify interaction properties relative to a specification
- describe test cases
- document simulation traces

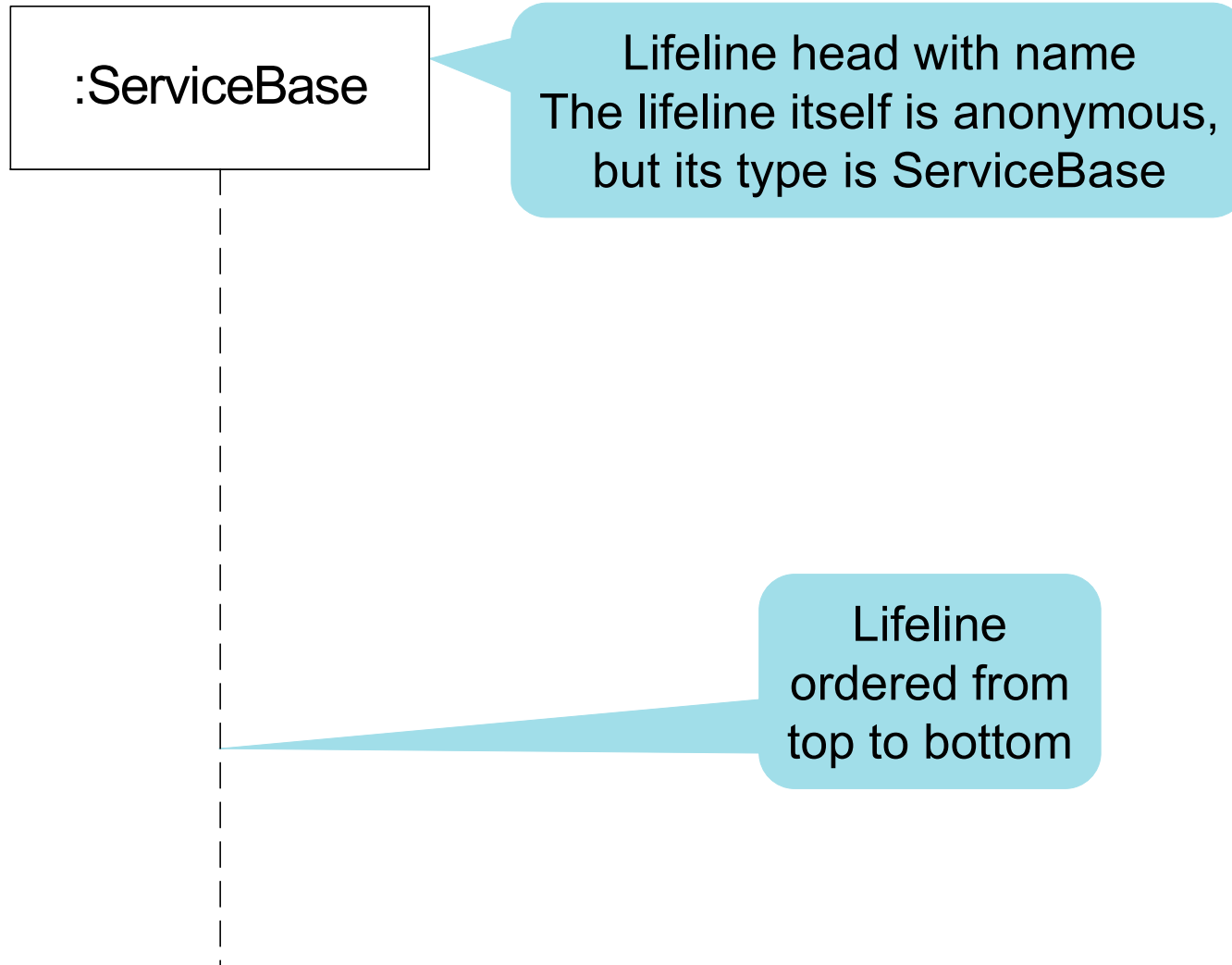
The example context: Dolly Goes To Town

- Dolly is going to town and
 - wants to subscribe for bus schedules back home
 - given her current position, and
 - the time of day
- The service should not come in effect until a given time in the evening

The informal architecture



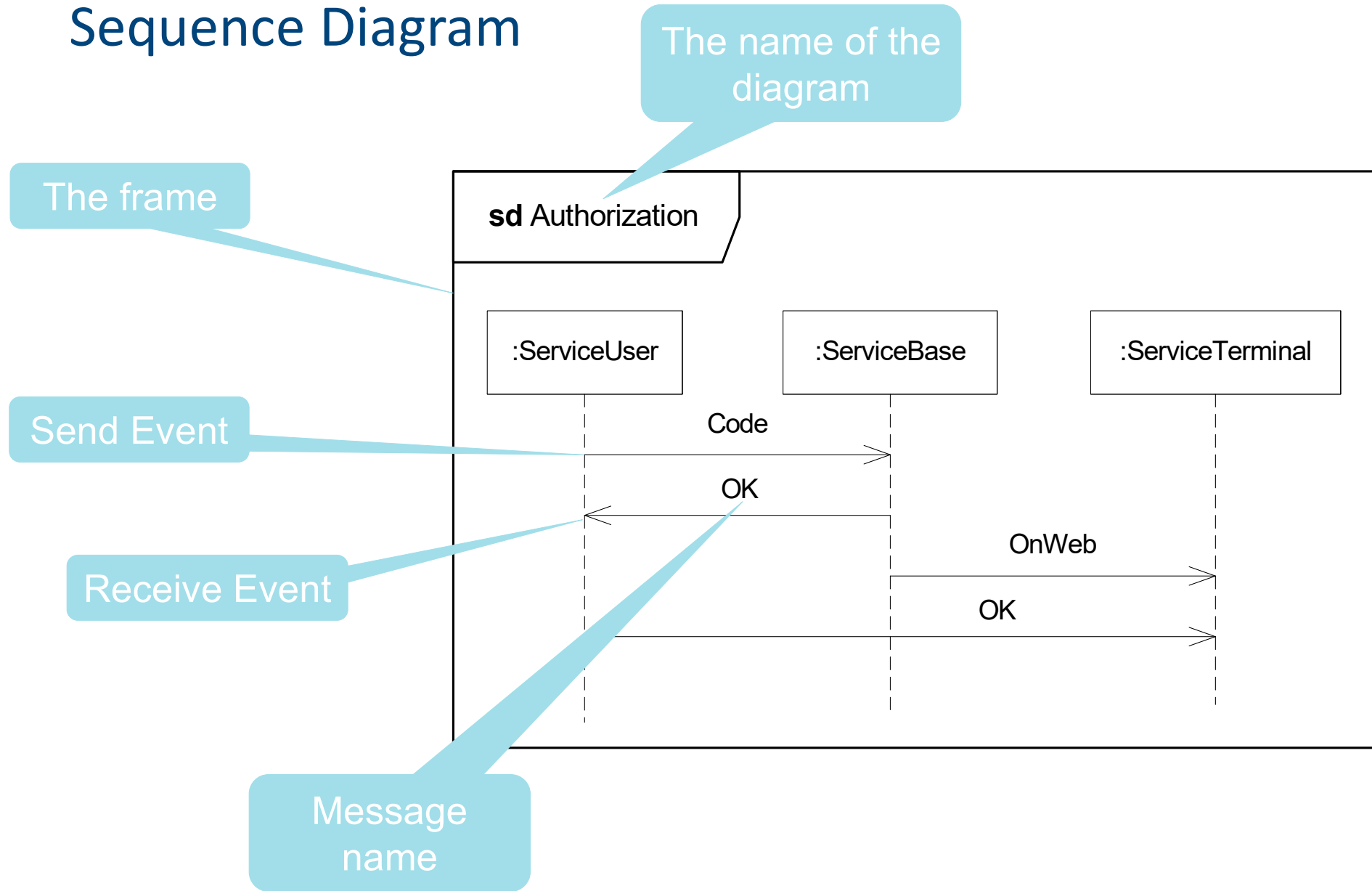
Lifeline – the “doer”



Messages

- Lifelines communicate messages
- A message has one send event and one receive event
- A message is always sent before it can be received
- Events are strictly ordered in time along a lifeline from top to bottom
- For any message name M we use $!M$ to denote its send event and $?M$ to denote its receive event

Sequence Diagram



Trace

- A trace is a sequence of events ordered in time
- A trace represents a possible execution of a sequence diagram
- A trace may be finite or infinite
- A sequence diagram describes at least one trace
- A sequence diagram may describe infinitely many traces

Two rules

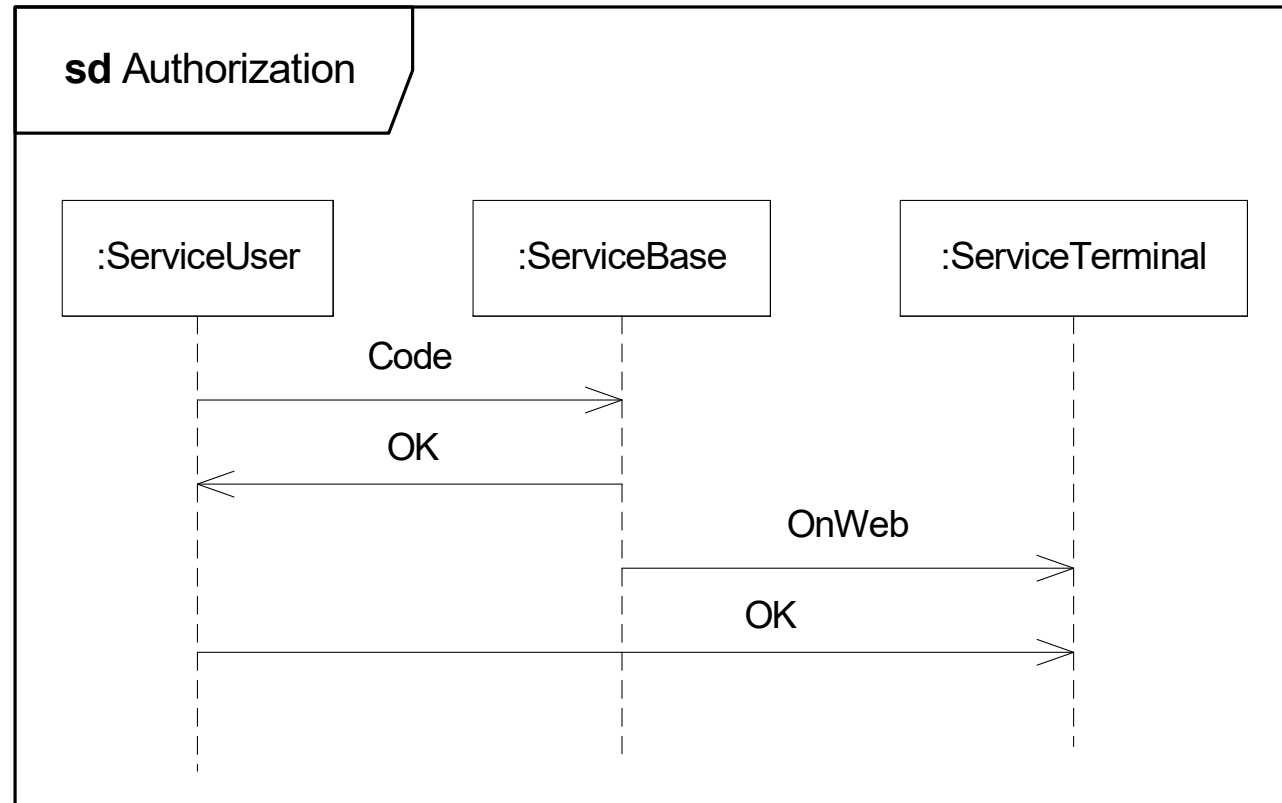
Causality

- a message can never be received before it has been transmitted
- the transmission event for a message is therefore always ordered before the reception event for the same message

Weak sequencing

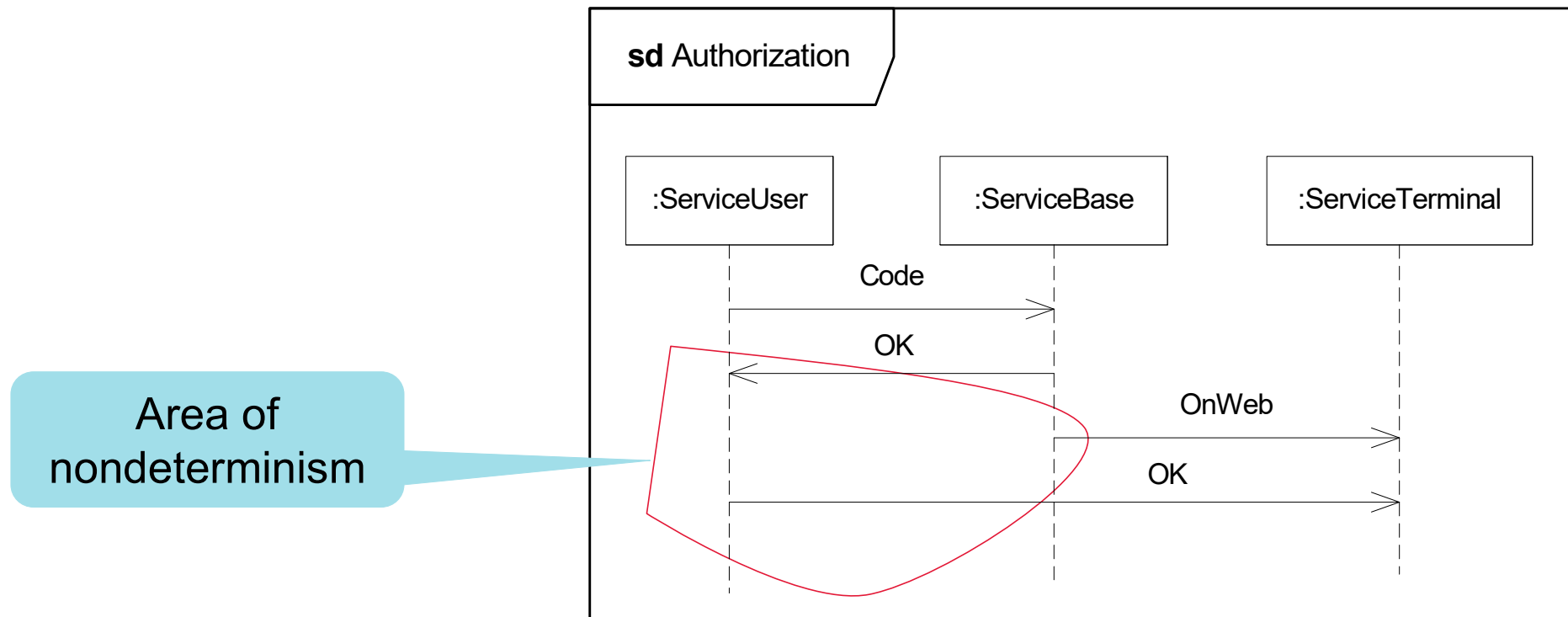
- events from the same lifeline are ordered in the trace in the same order as on the lifeline (from top to bottom)

One possible trace of Authorization

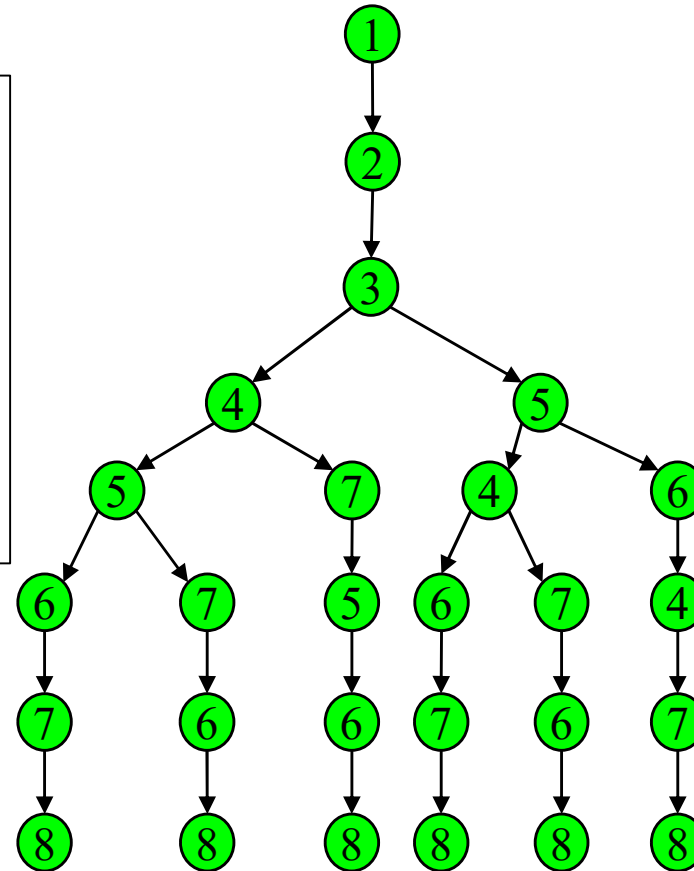
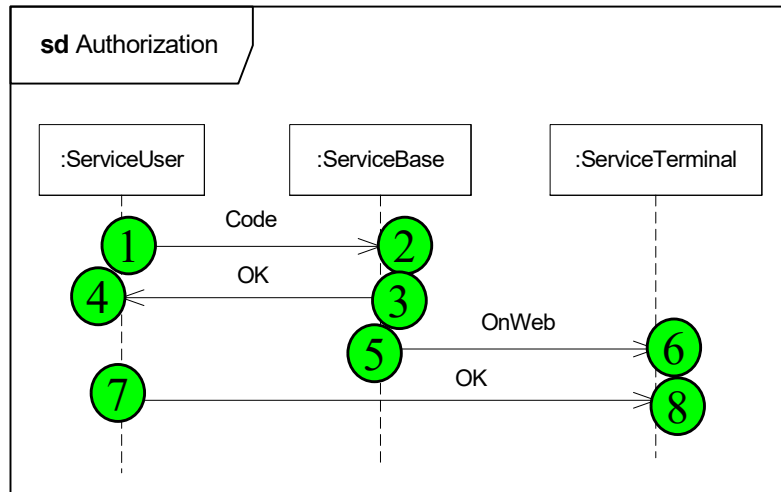


< !Code, ?Code, !OK, ?OK, !OnWeb, ?OnWeb, !OK, ?OK >

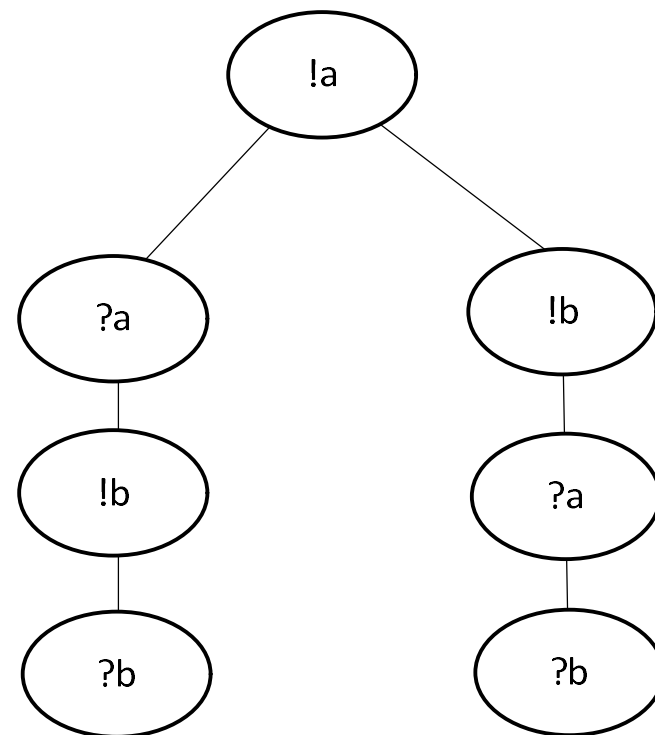
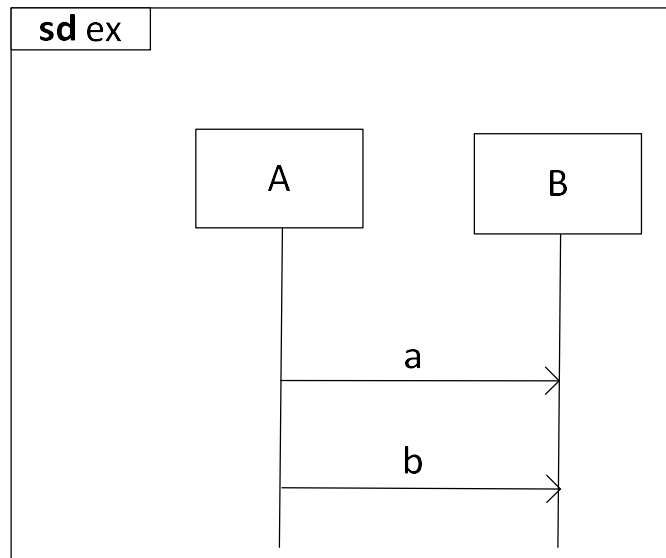
But there is more than one



In fact, there are six traces



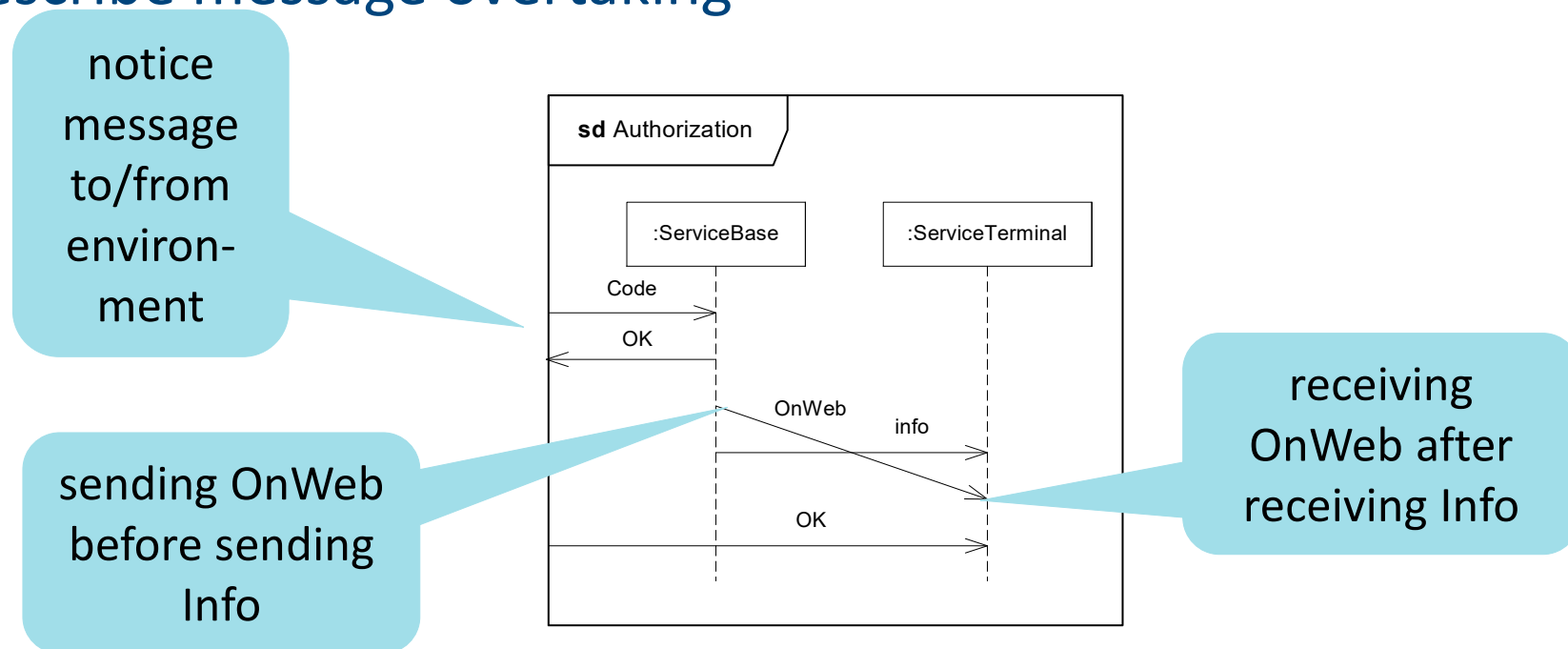
Another example



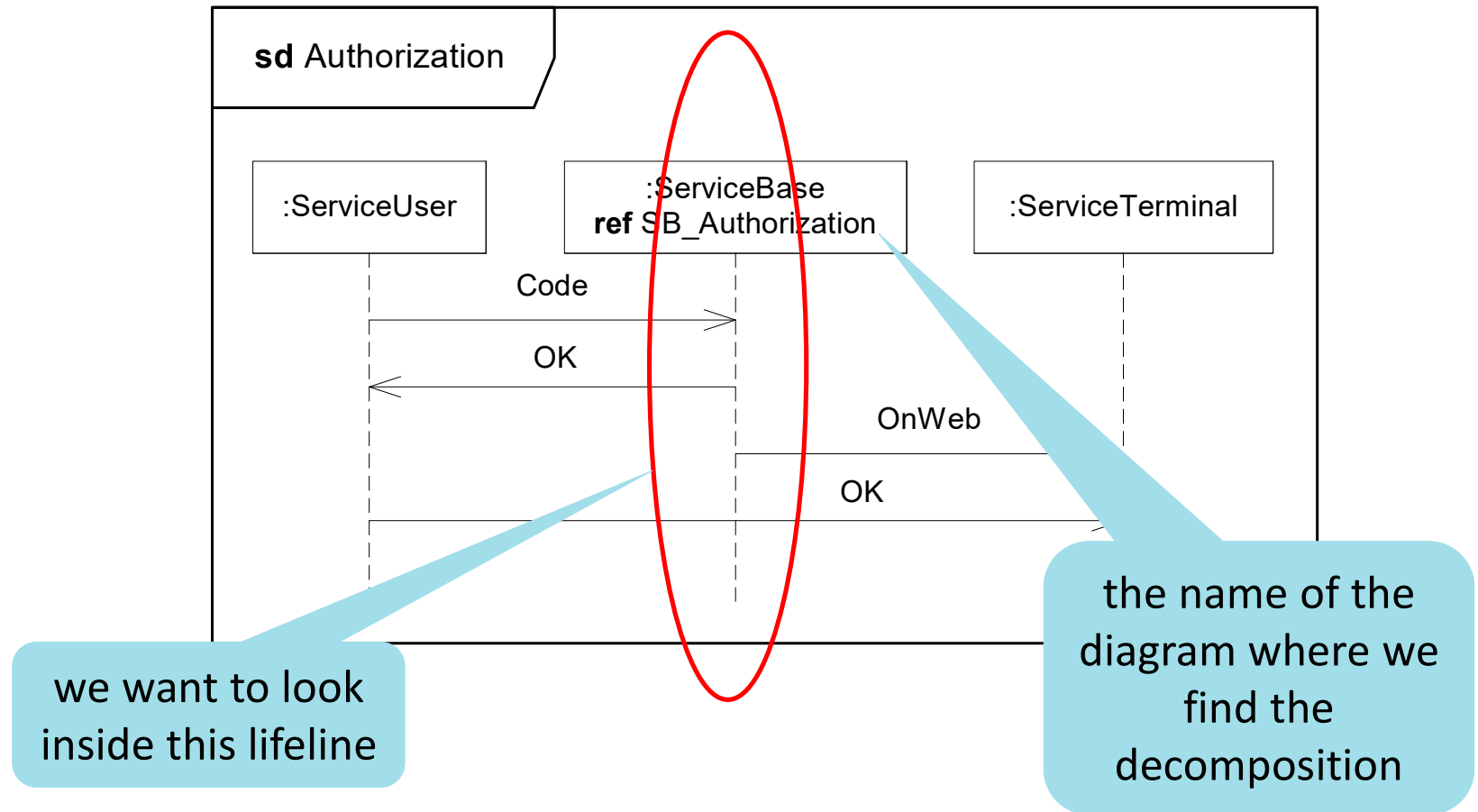
Exercise: Explain how the number of traces in the previous diagram can be reduced by adding messages

Asynchronous communication

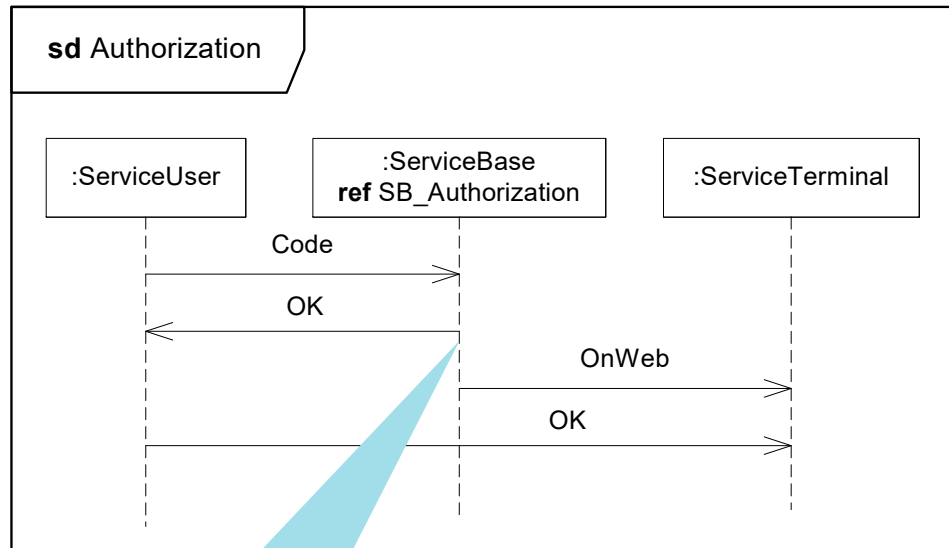
- asynchronous communication = the sender does not wait for the reply to the message sent
- reception is normally interpreted as consumption of the message
- when messages are asynchronous, it is important to be able to describe message overtaking



Decomposing a lifeline

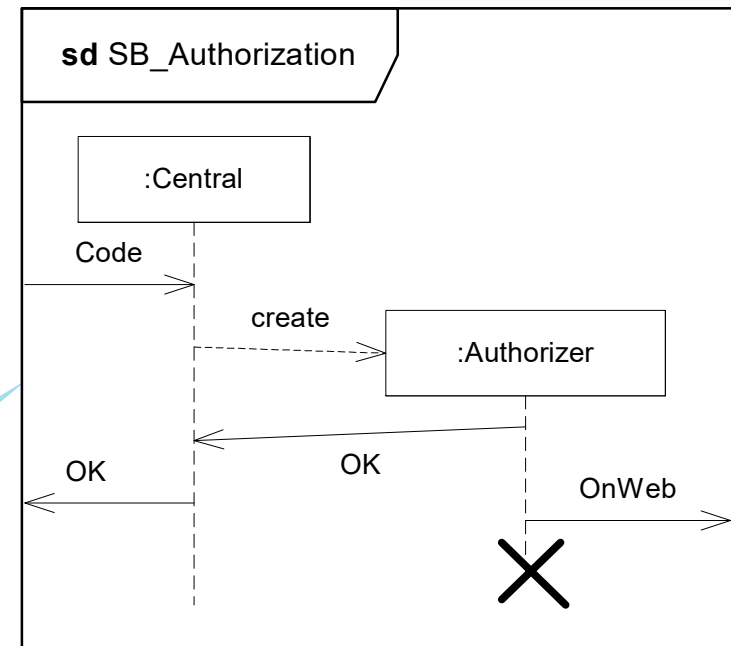


The decomposition



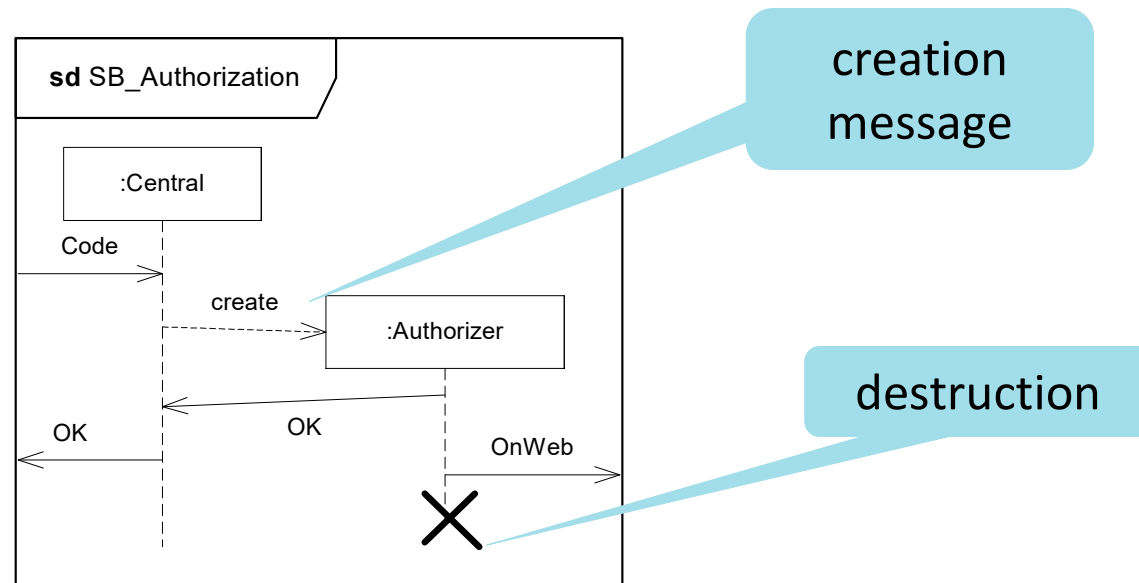
notice the *event* correspondence!

notice the *gate* correspondence!



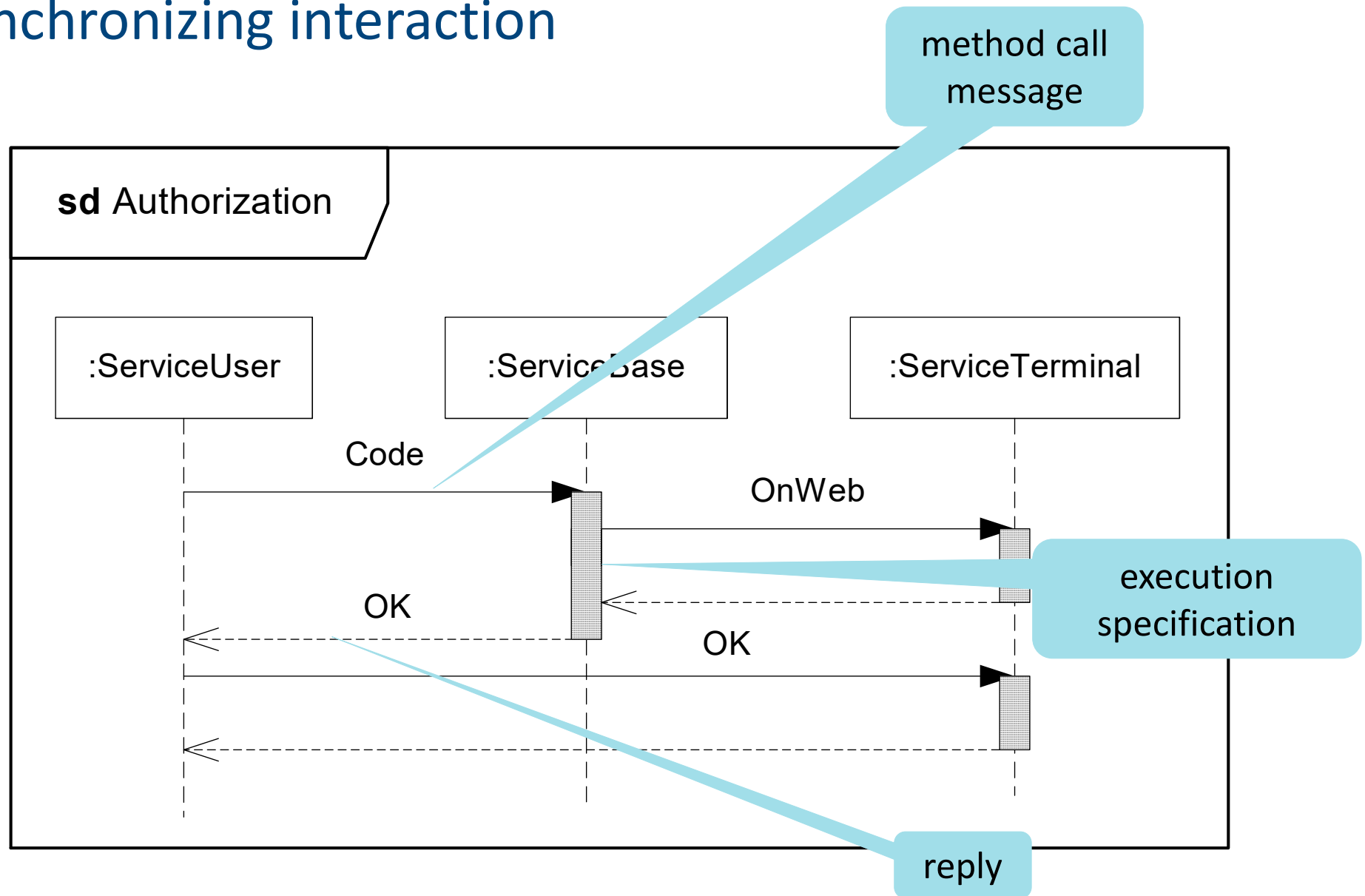
Lifeline creation and destruction

The idea here (though rather far fetched) is that the ServiceBase needs to create a new process in the big mainframe computer to perform the task of authorizing the received Code



Exercise: How many global traces are there in the decomposed diagram?

Synchronizing interaction



Summary of basic sequence diagrams

- We consider mostly messages that are asynchronous
 - the sending of one message must come before the corresponding reception
- The events on a lifeline are strictly ordered
- The distance between events is not significant.
- A lifeline (within an interaction) may be detailed in a decomposition
- We may dynamically create and destruction of lifelines