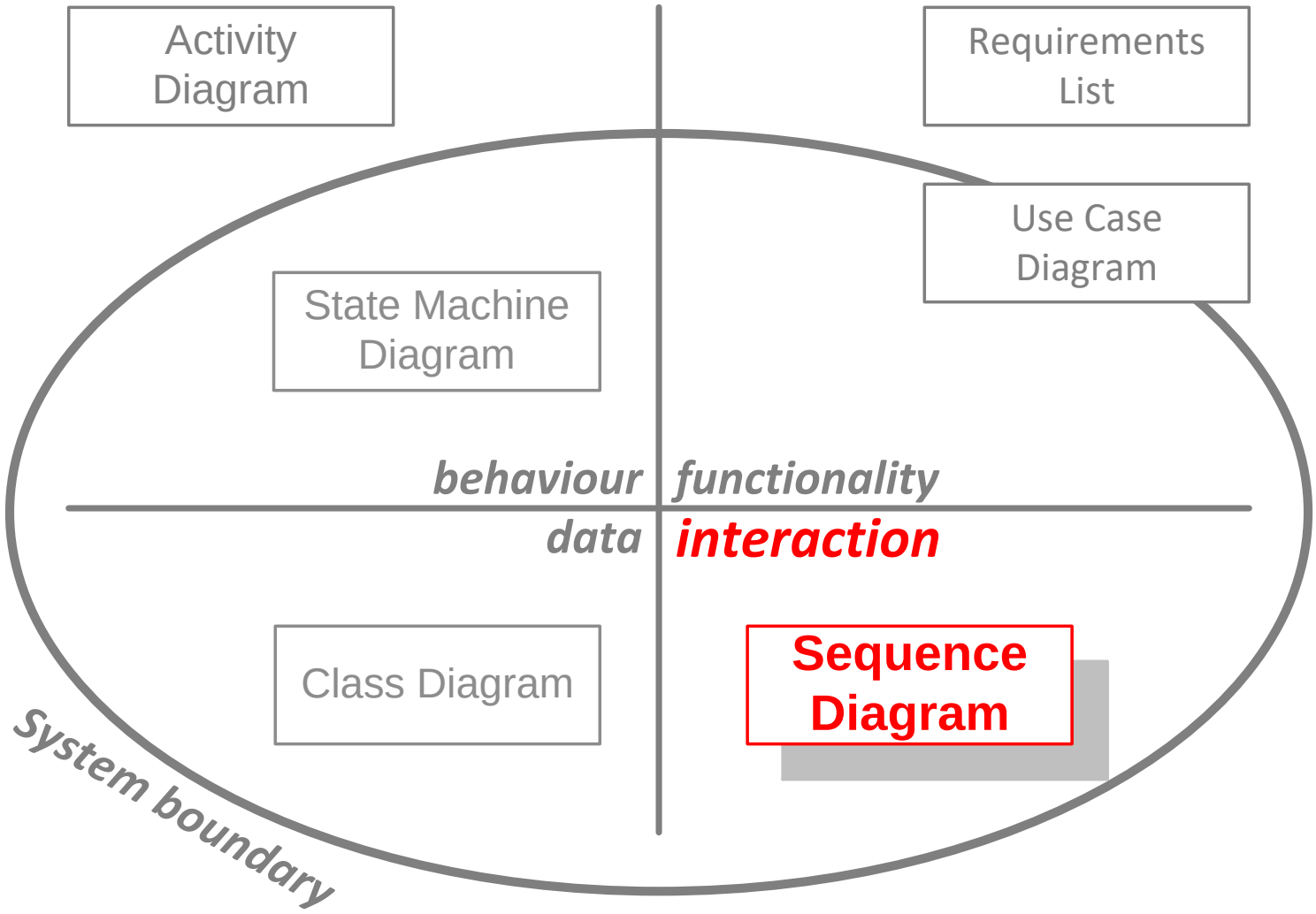
Software Systems

Design – 2B
Sequence Diagrams

Joke van Staalduinen

Credits to Klaas Sikkel

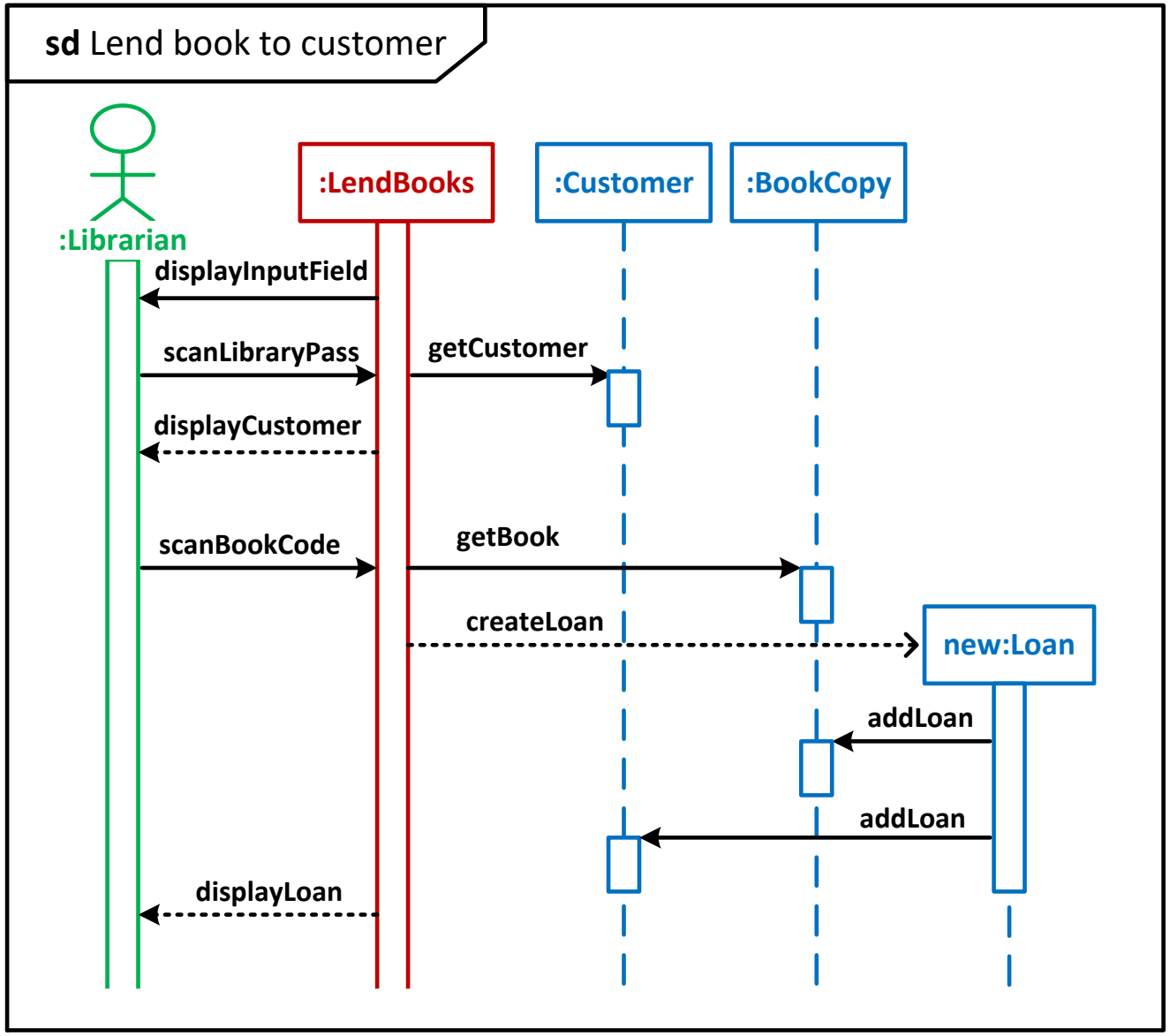
Today's topic



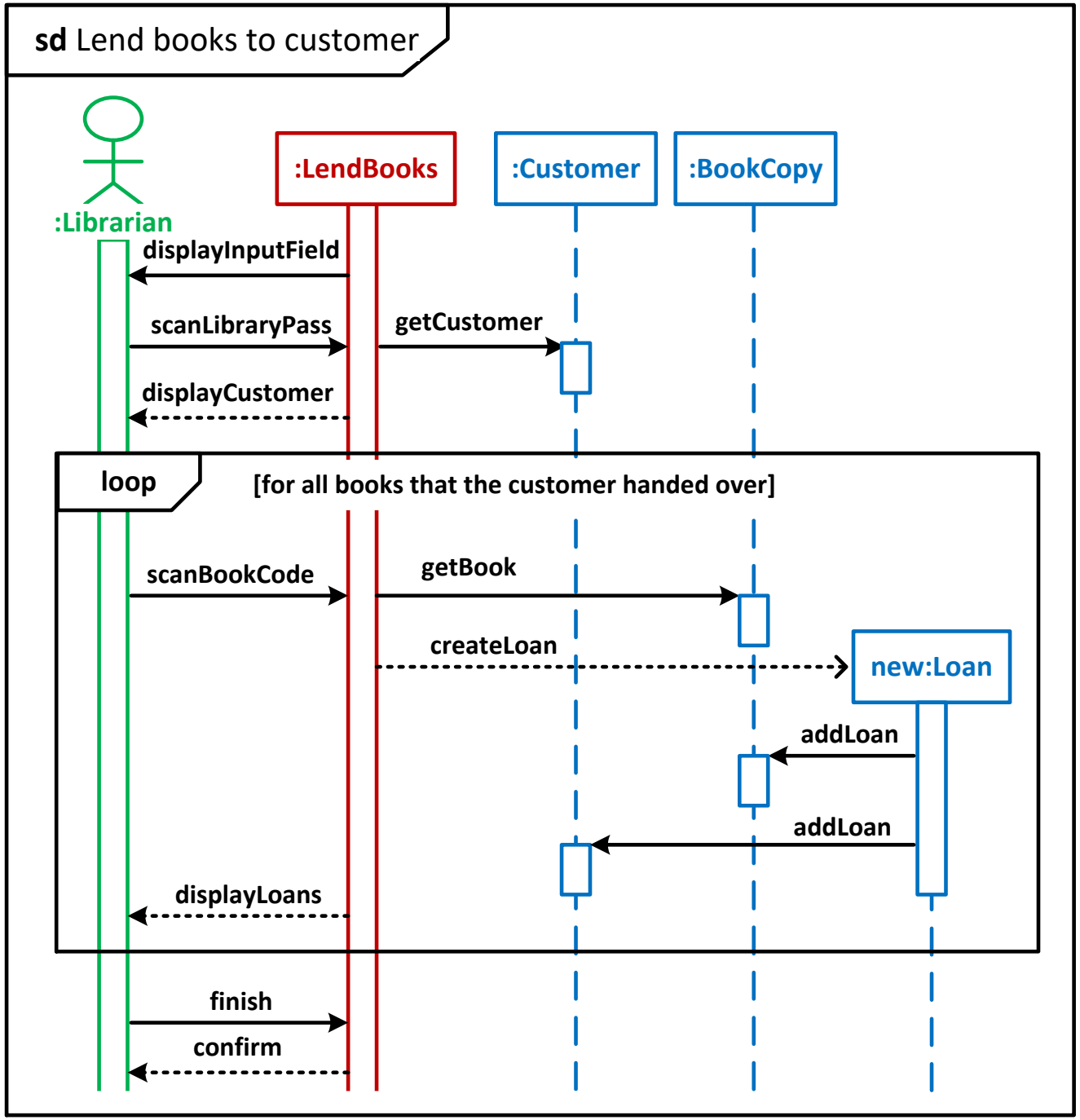
Contents

1. Introduction (running example)

Example of a sequence diagram



Example of a sequence diagram



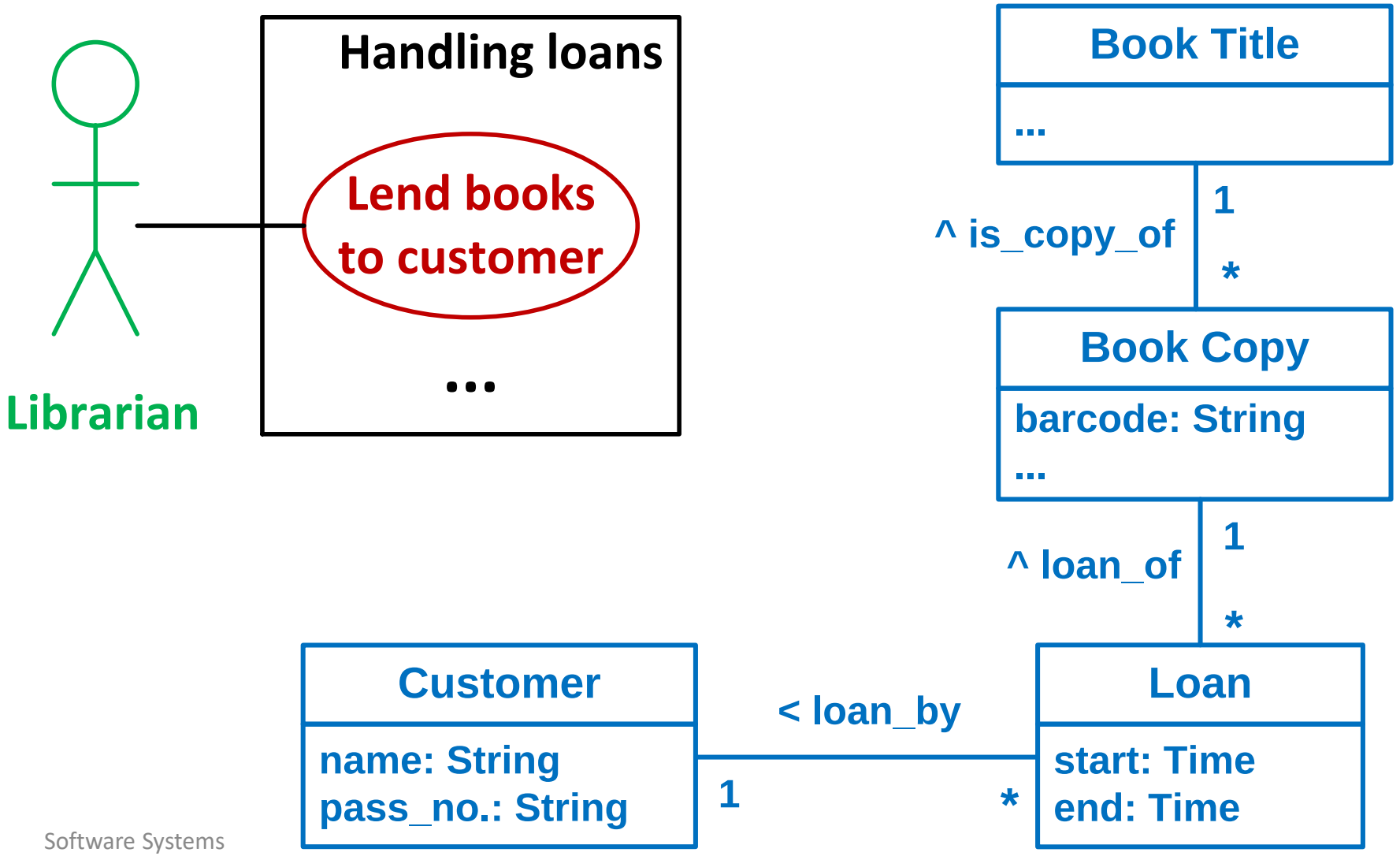
Execution of a use case

- Sequence diagrams describe how a use case is executed
- Elements of a sequence diagram:
  - Actor (green in these slides)
  - Control object (brown in these slides)
  - Data objects (blue in these slides)
  - Various types of messages back and forth
  - Control structures

Running example: new loans for library books

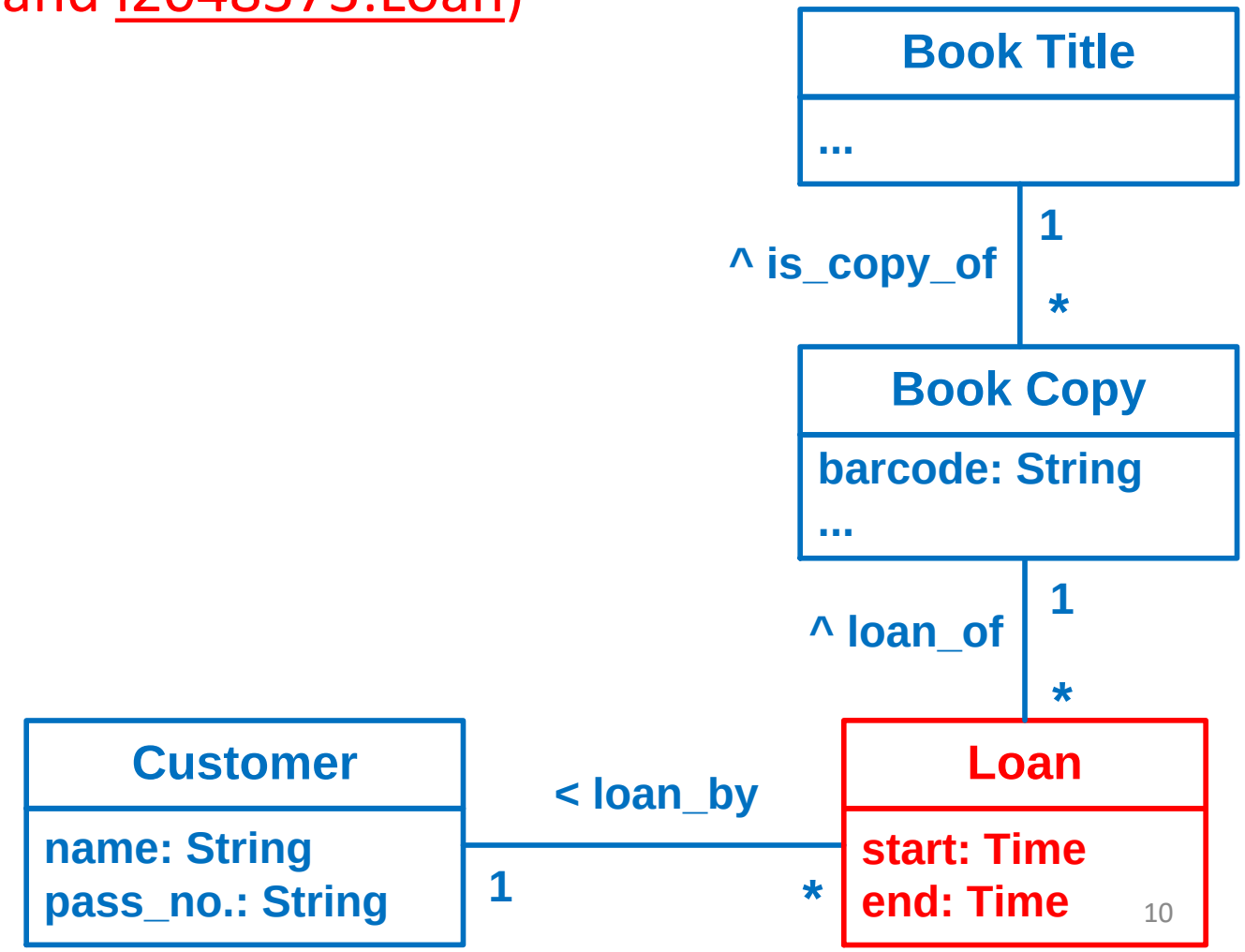
- A customer goes to the library counter with two books, picked up from the shelves
- The librarian scans the customer's library pass
- The librarian scans each of the books
- The librarian completes the transaction
  (i.e. the computer system knows that no more books for this customer will follow)

Use case Diagram / Class Diagram



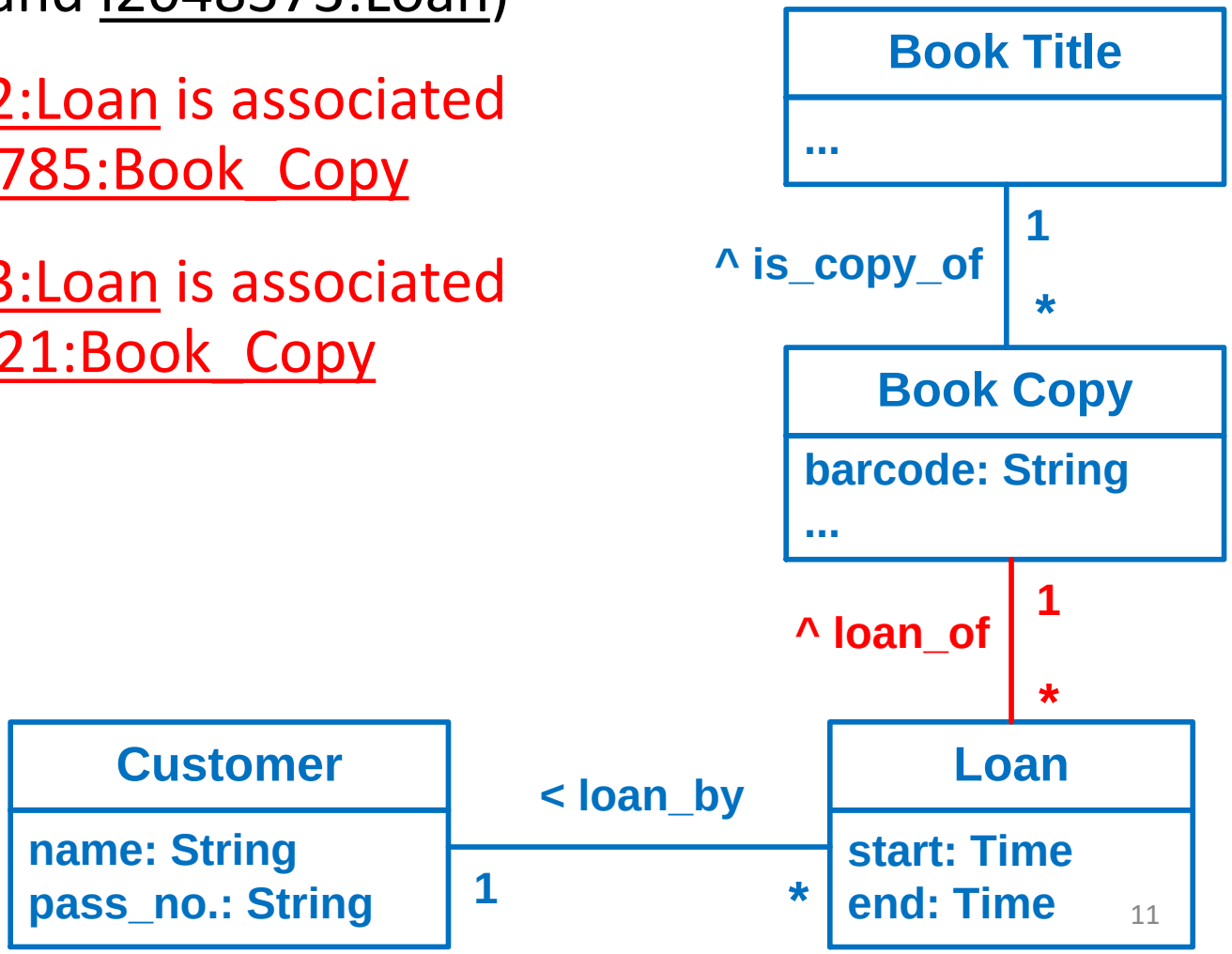
Executing the use case

- Two new loan objects are created (e.g. l2048372:Loan and l2048373:Loan)



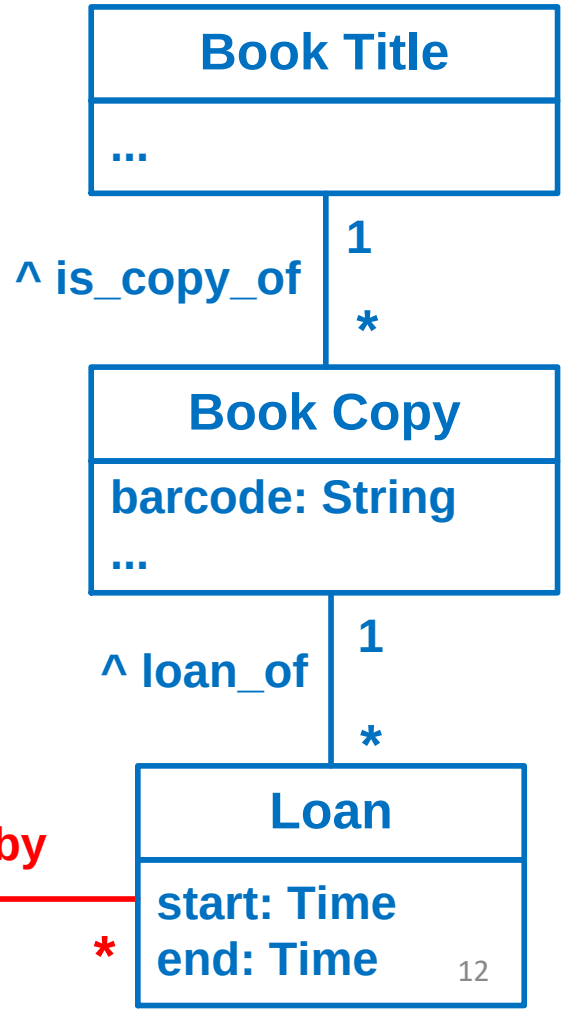
Executing the use case

- Two new loan objects are created (e.g. l2048372:Loan and l2048373:Loan)
- Object l2048372:Loan is associated with object b34785:Book_Copy
- Object l2048373:Loan is associated with object b4021:Book_Copy

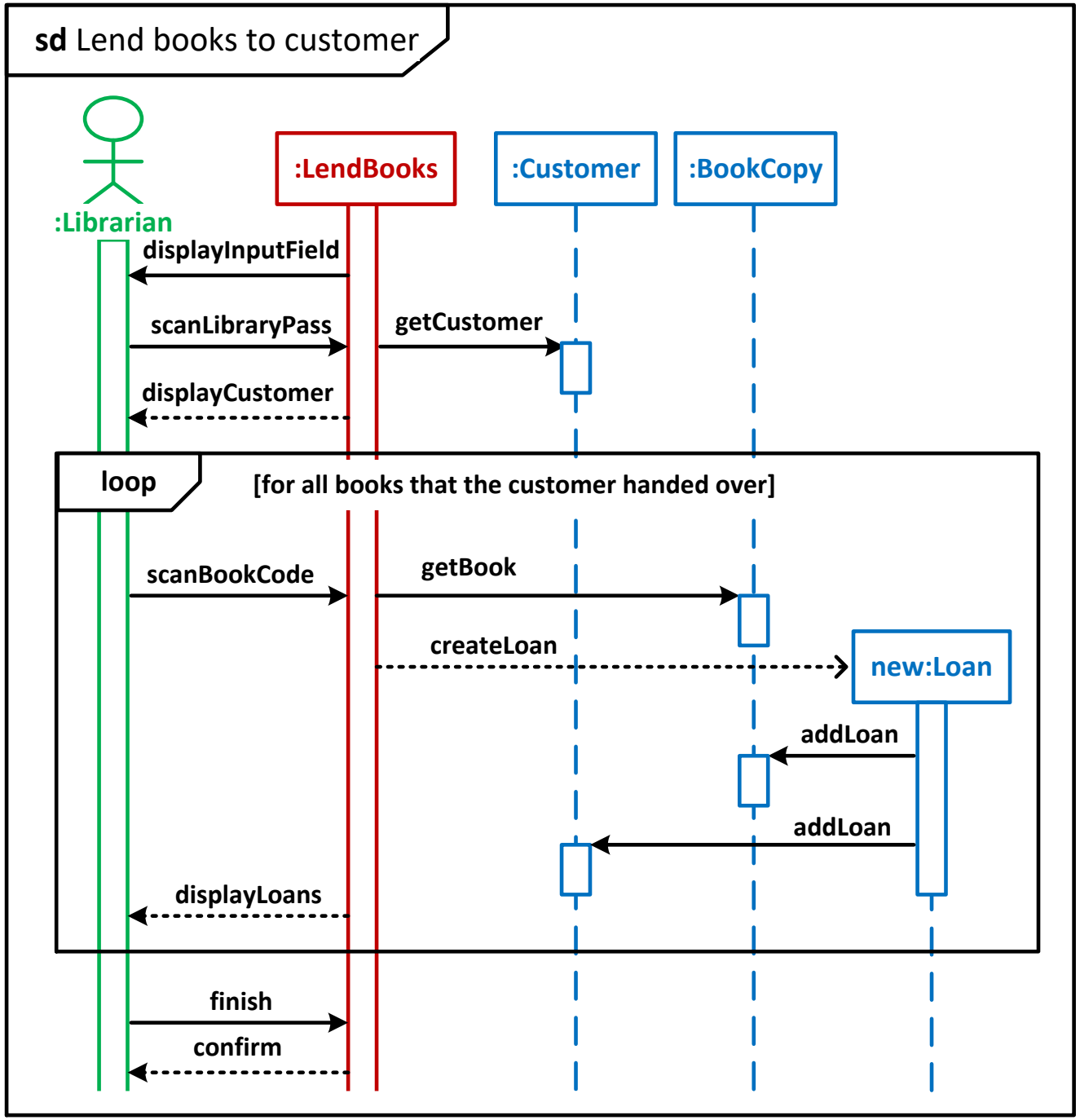


Executing the use case

- Two new loan objects are created (e.g. l2048372:Loan and l2048373:Loan)
- Object l2048372:Loan is associated with object b34785:Book_Copy
- Object l2048373:Loan is associated with object b4021:Book_Copy
- Both loan objects are associated with object c19814:Customer



Sequence diagram for “lend books to customer”



2. Basics of Seqence Diagram construction

Heuristic:
Two parts of a sequence diagram

1. Step-by-step description of the interaction between the user and the system
   (more precisely:
   between the user and the control object, mediated by the user interface)

2. Actions to create/read/update/delete objects in the system

(The easy part)

Step-by-step description of the inter-action between the user and the system

- We know how to make these ☺
  - Extended Use Case Description gives a verbal description
  - This can be mapped one-to-one on a graphical description

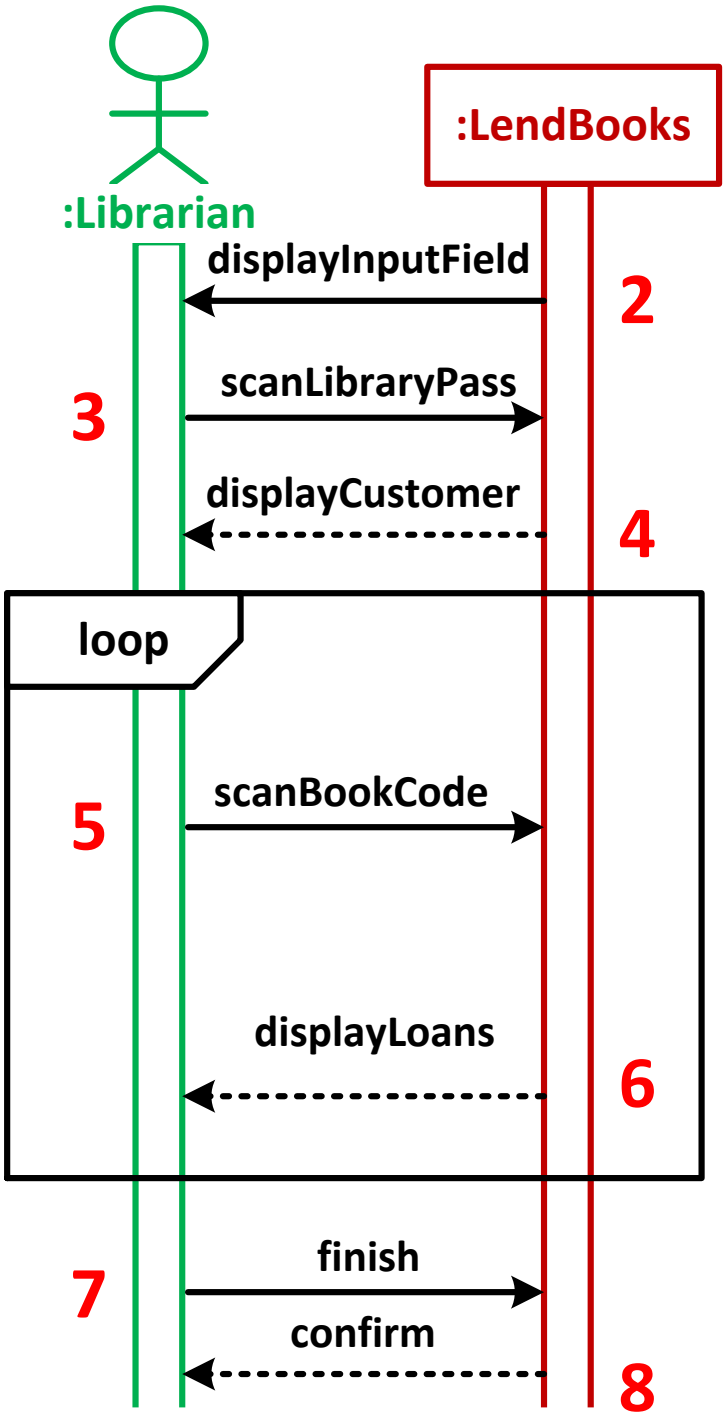
Extended Use Case Description

User		System	
1		2	Display input field
3	Scan library pass	4	Display (empty) loan data for this customer
5	Scan book barcode	6	Display (updated) loan data for this custmer
7	Finish	8	Display confirmation

Alternatives:

Step 1: start lending function if not yet active

Step 5, 6: repeat as appropriate



User		System	
1		2	Display input field
3	Scan library pass	4	Display loan data for this customer
5	Scan book barcode	6	Display loan data for this customer
7	Finish	8	Display confirmation

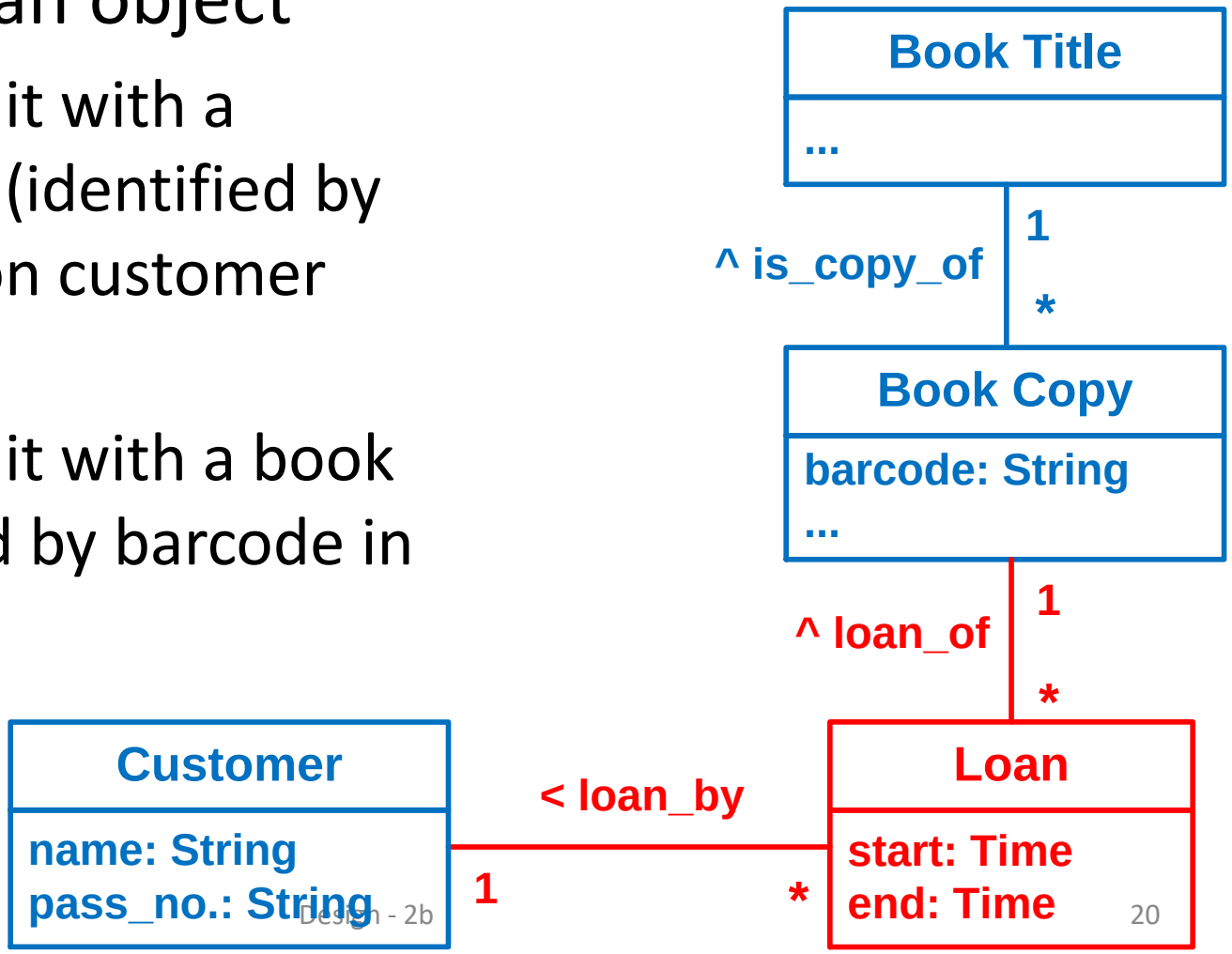
(The extension)

Actions to create/read/update/delete objects in the system

- Requires
  - Knowledge of what objects exists (class diagram!)
  - Knowledge of what has to be modified

Add a new loan

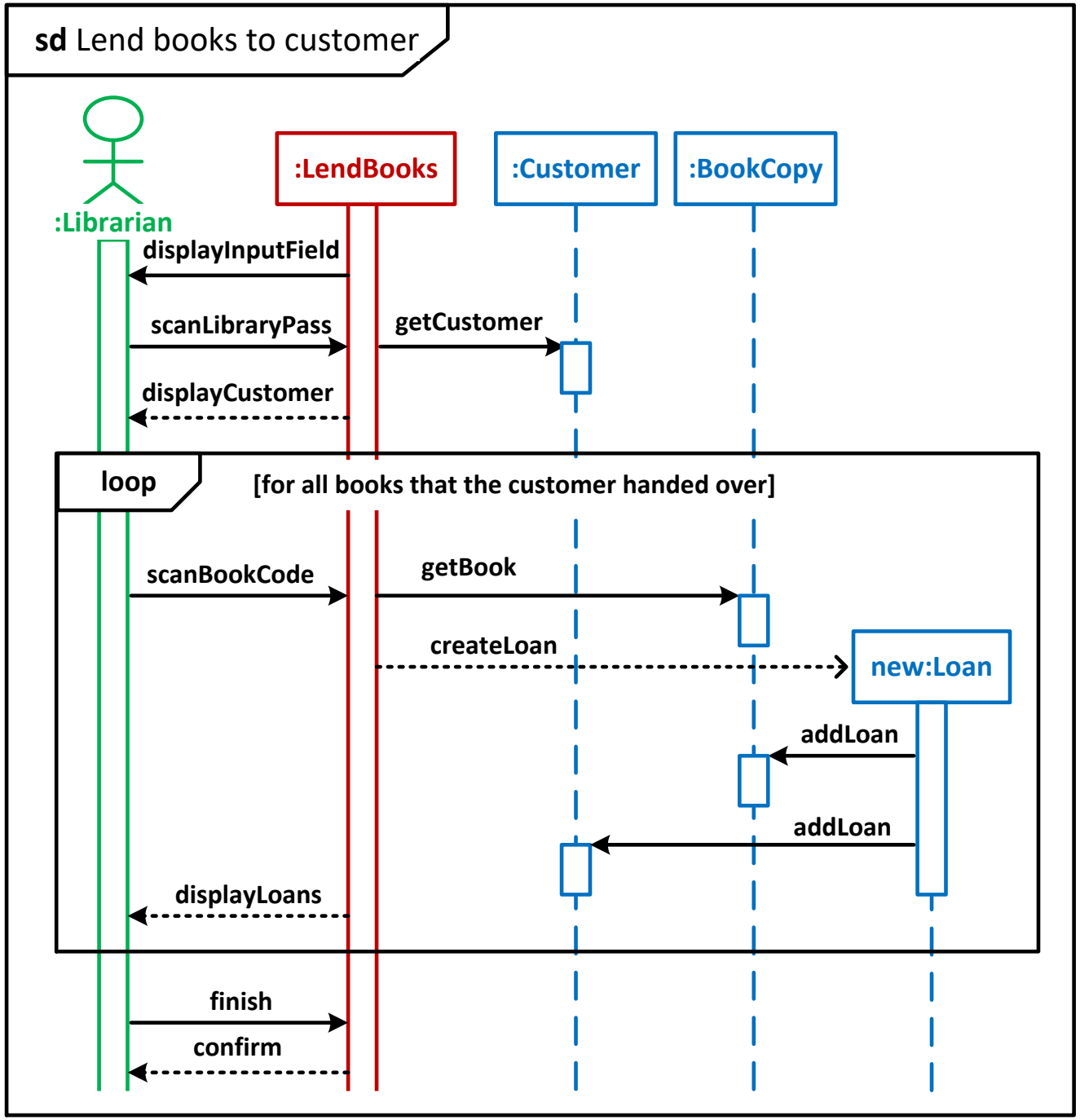
- Create a loan object
  - Associate it with a customer (identified by barcode on customer pass)
  - Associate it with a book (identified by barcode in book)



Add a new loan

- Create a loan object
  - Associate it with a customer (identified by barcode on customer pass)
  - Associate it with a book (identified by barcode in book)
- Procedure:
  - first identify customer (scan pass)
  - Then deal with all the books for this customer

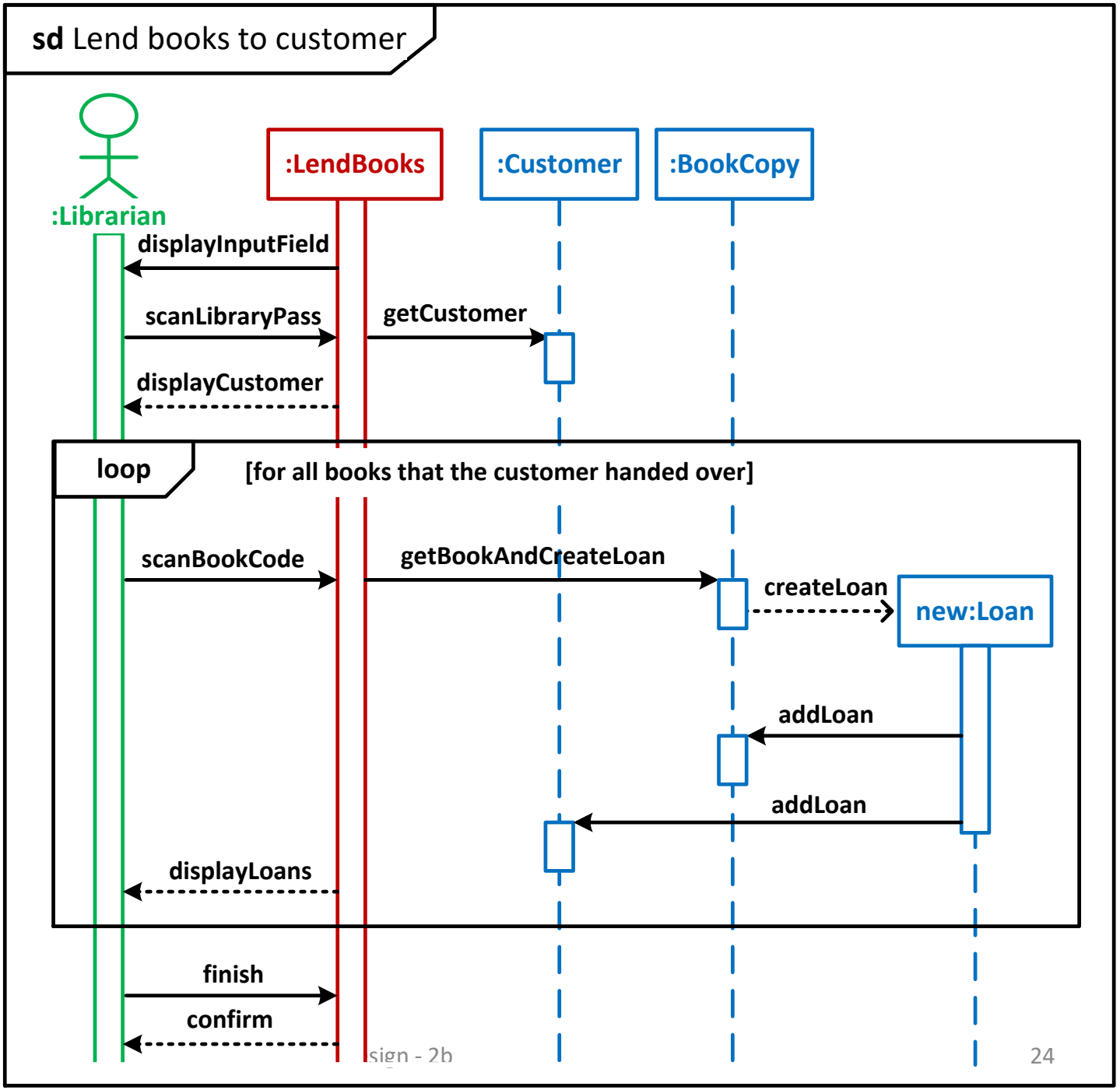
Complete sequence diagram



Object interaction can be designed in different ways

- In the example:
  Control object passes book id and customer id to new Loan object, which has program logic to create associations with Book Copy and Customer objects
- Alternative:
  Control object passes customer id to Book Copy object, which has program logic to create a new Loan object

Alternative Sequence Diagram



Elements of a SD (1)

A sequence diagram describes interactions between objects

- SD always contains an ***actor object*** and usually contains a ***control object***
- The ***lifeline*** of an object is denoted by a dashed line
- An ***activation*** (period in which the object is active) is denoted by a vertical bar

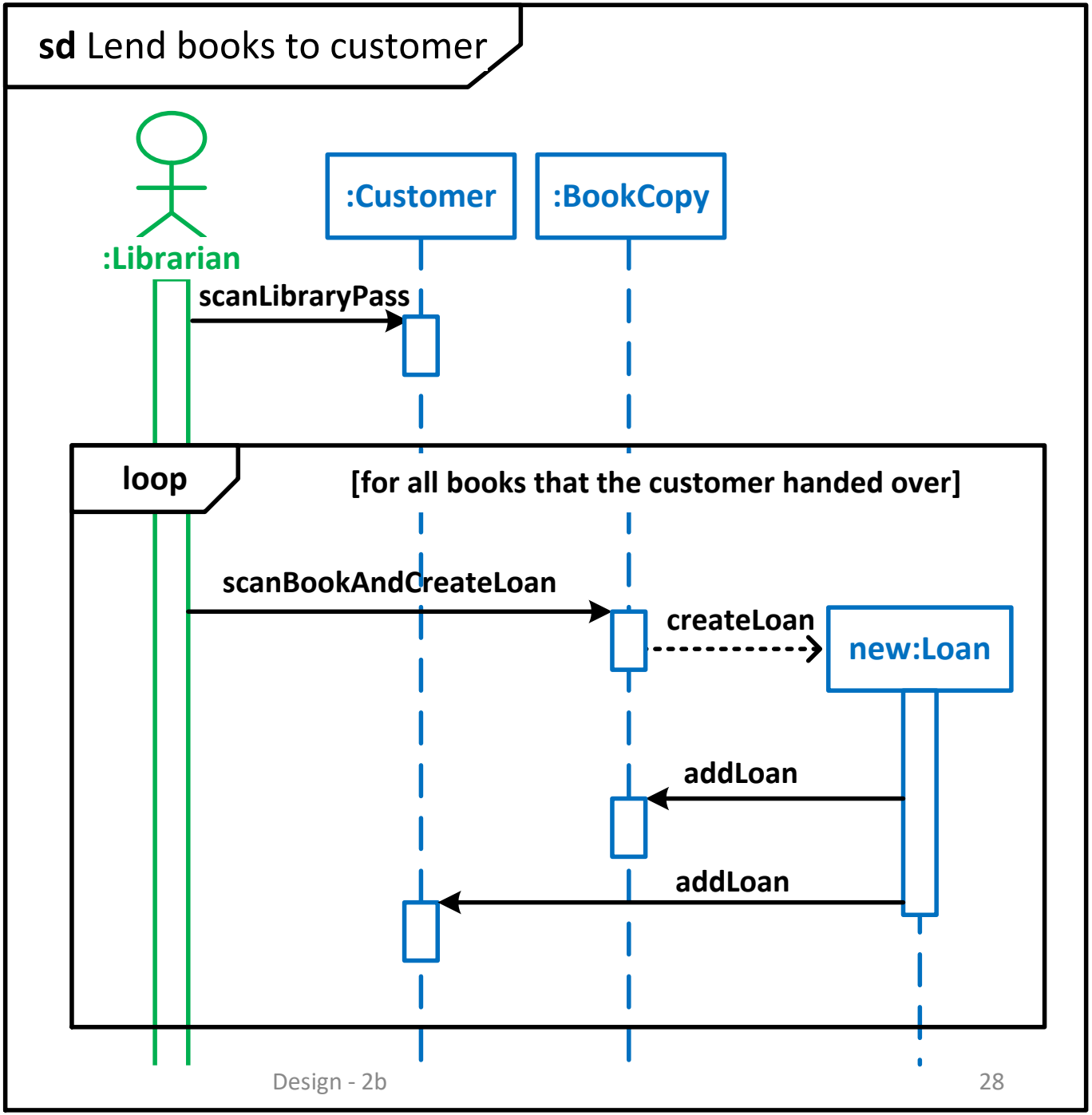
Elements of a SD (2)

- (normal) ***Messages*** from one object to another are represented by a solid arrow
- ***Return messages*** (to the calling object) are represented by a dashed arrow
  - Trivial return messages do not need to be included
  - Do include them if you want to make it explicitly clear that something is returned
- ***Create messages*** (creating a new object) are represented by a dashed arrow

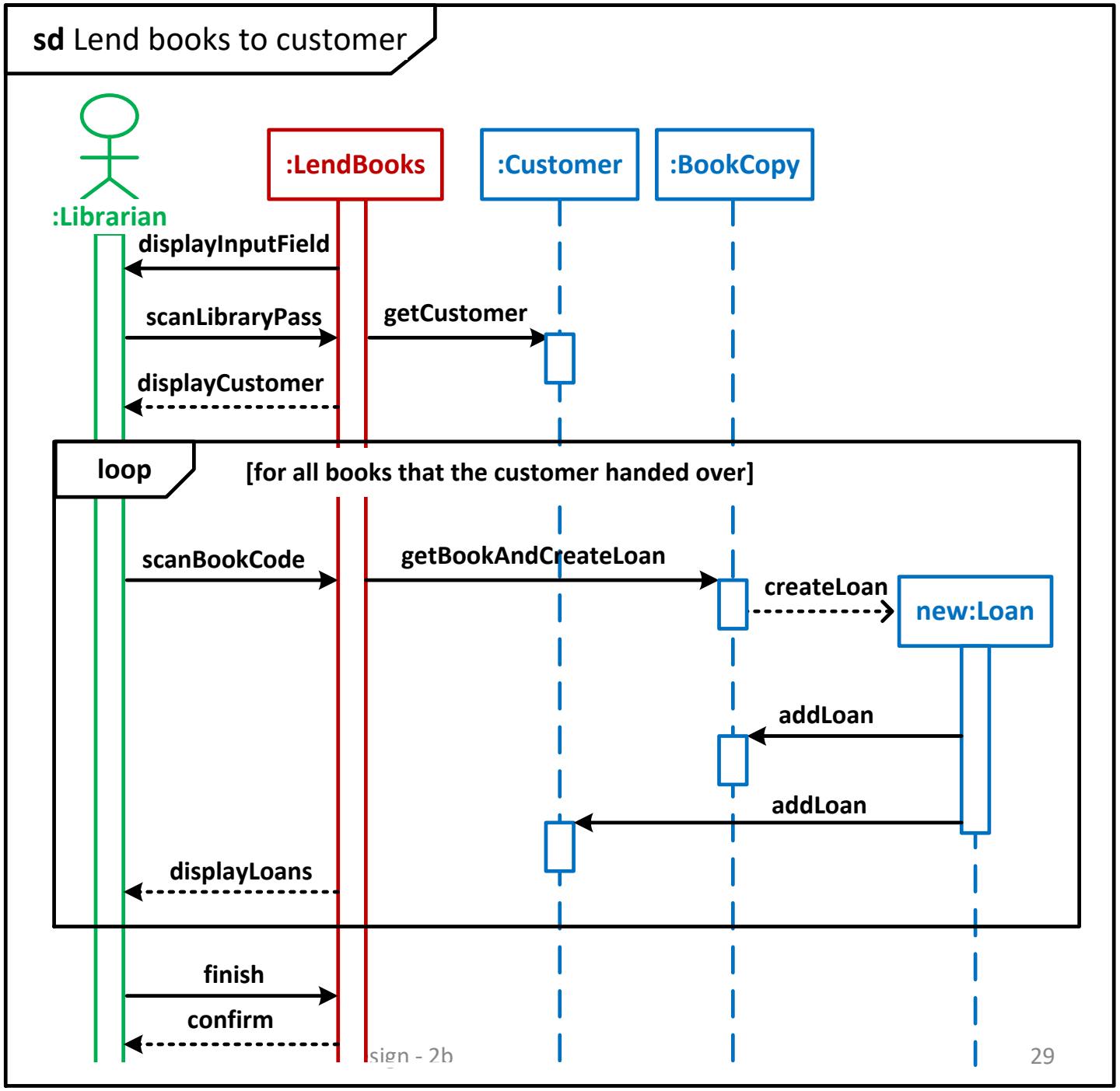
A simplified variant

- It there is little or no program logic in the control object, it can be deleted from the sequence diagram
  - Creates a simpler model, that better highlights the essential structure
  - Loses some details about user interaction
    (no longer identical to Extented Use Case Description)

Alternative Sequence Diagram without control object



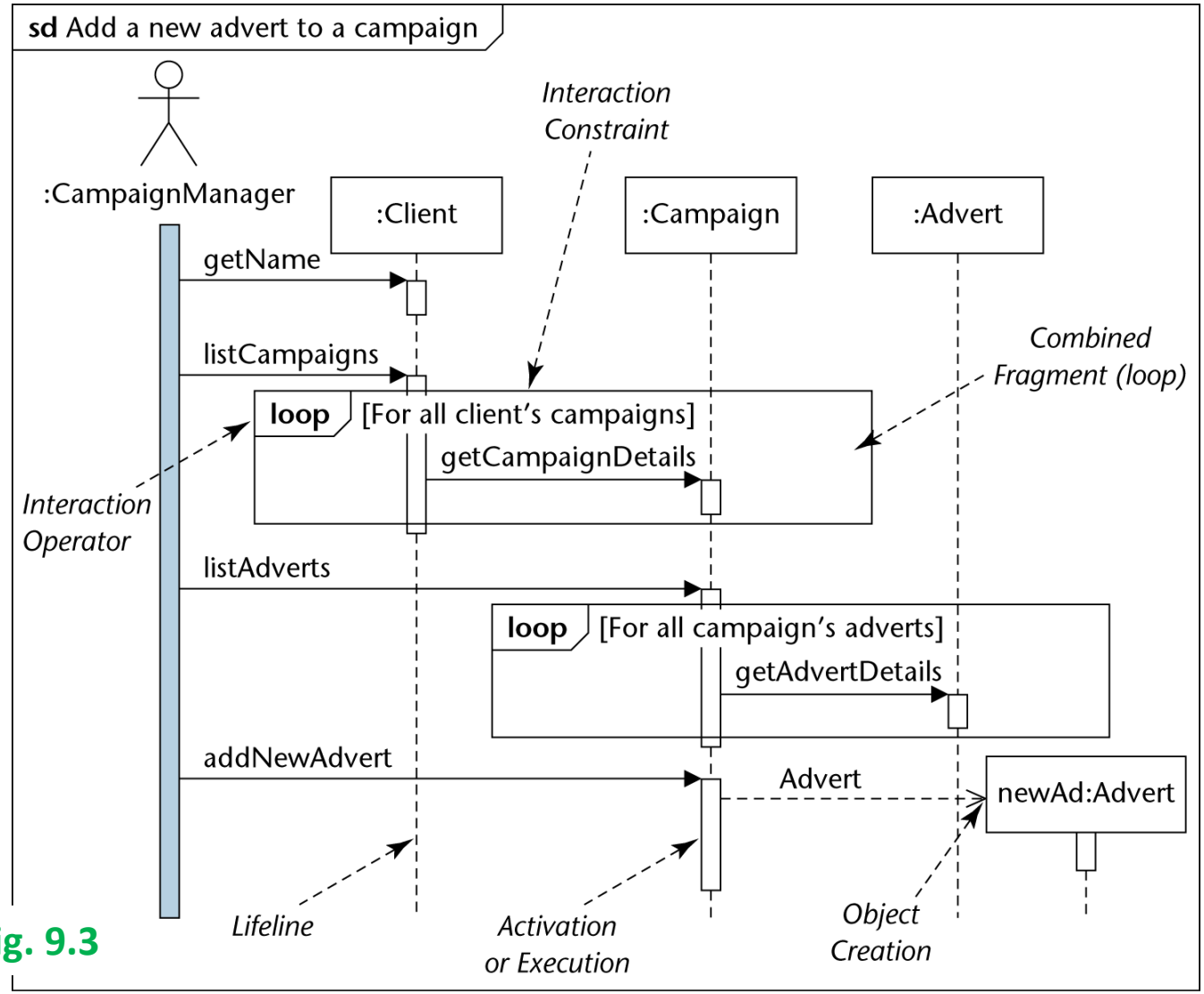
(same SD with control object)



3. Extensions

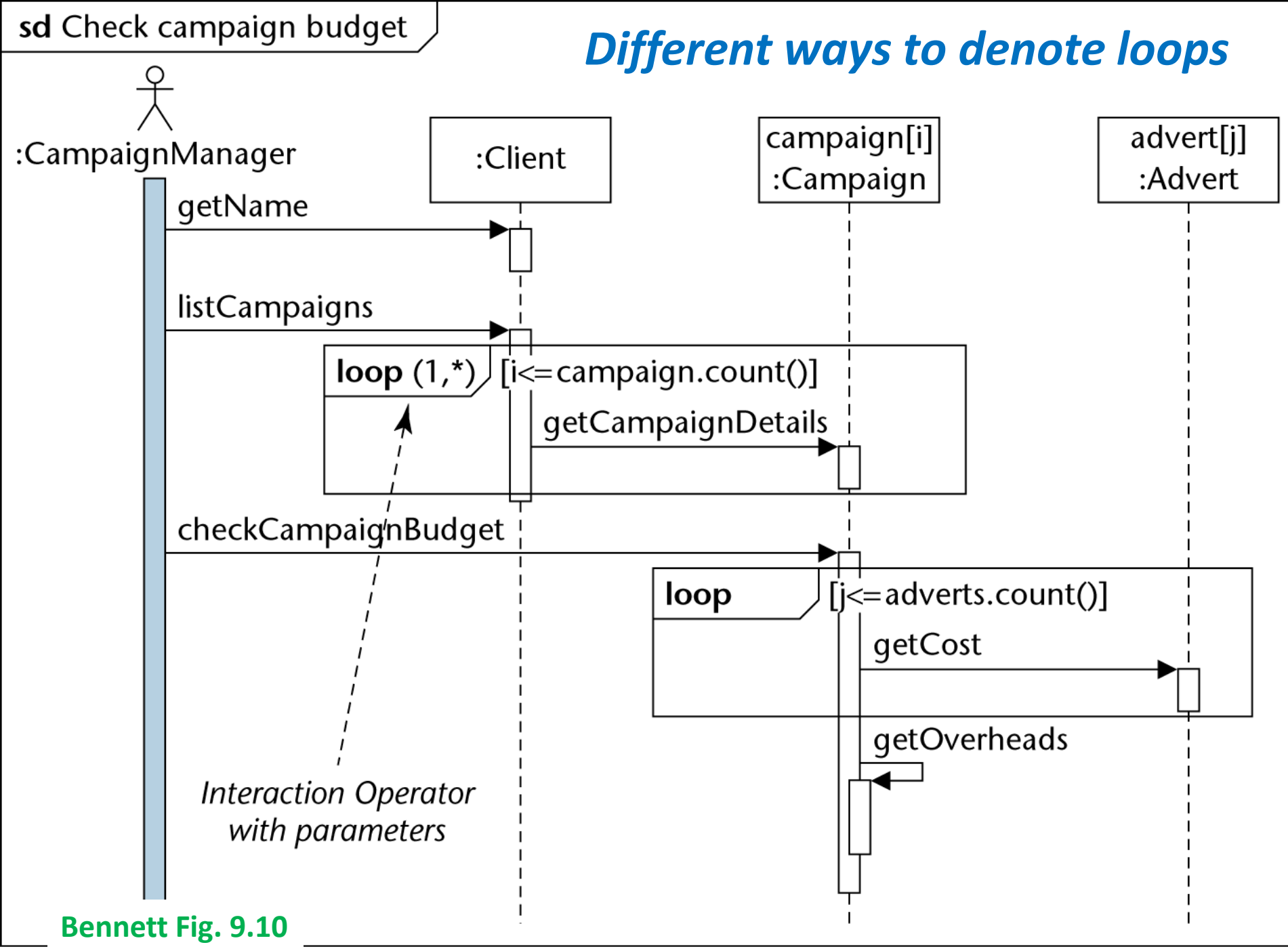
The examples are from Bennett et al. (handout wk. 1)

(without control object)



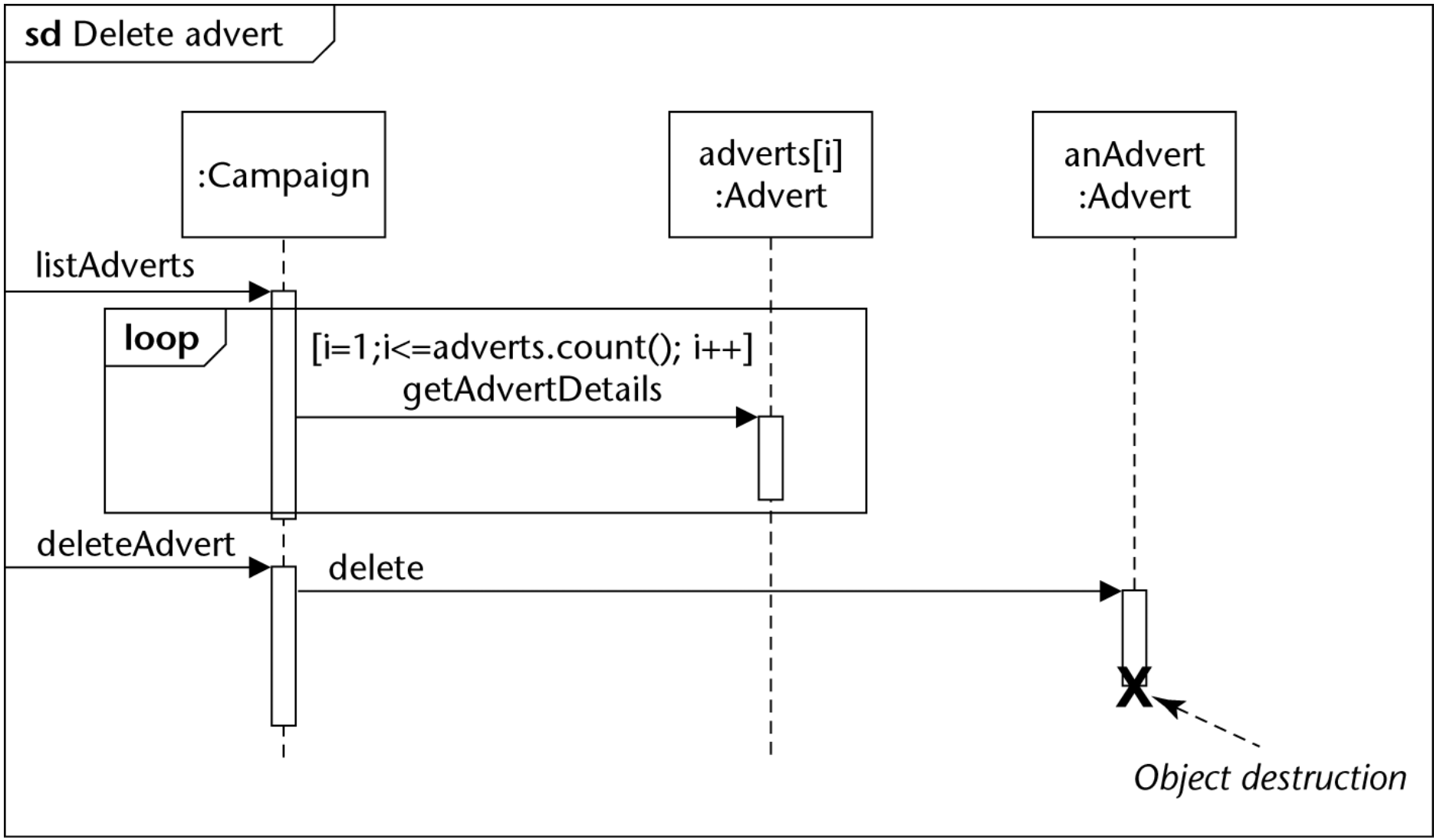
Bennett Fig. 9.3

Different ways to denote loops



Bennett Fig. 9.10

Object destruction

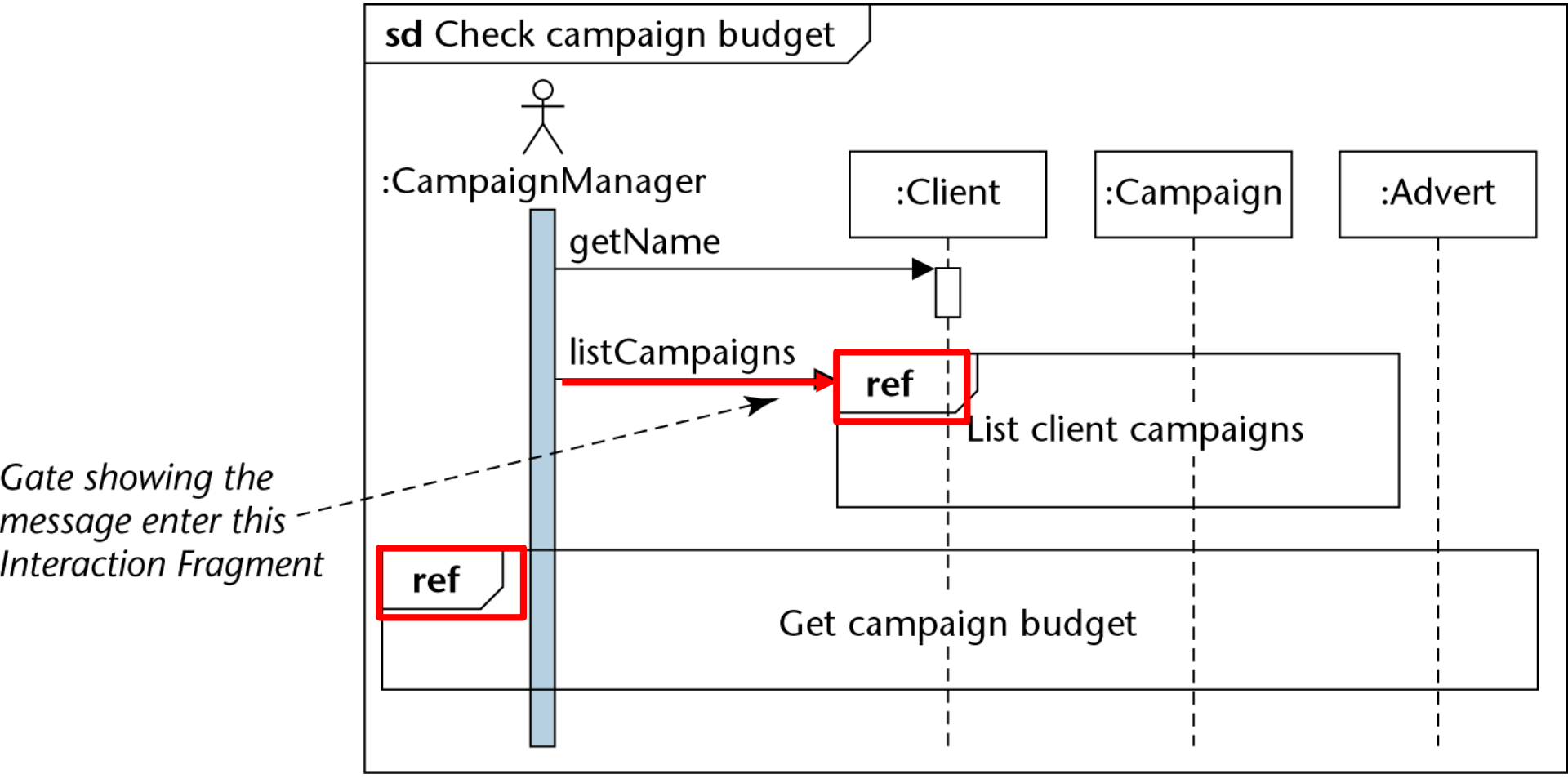


Bennett Fig. 9.6

Elements of a SD (3)

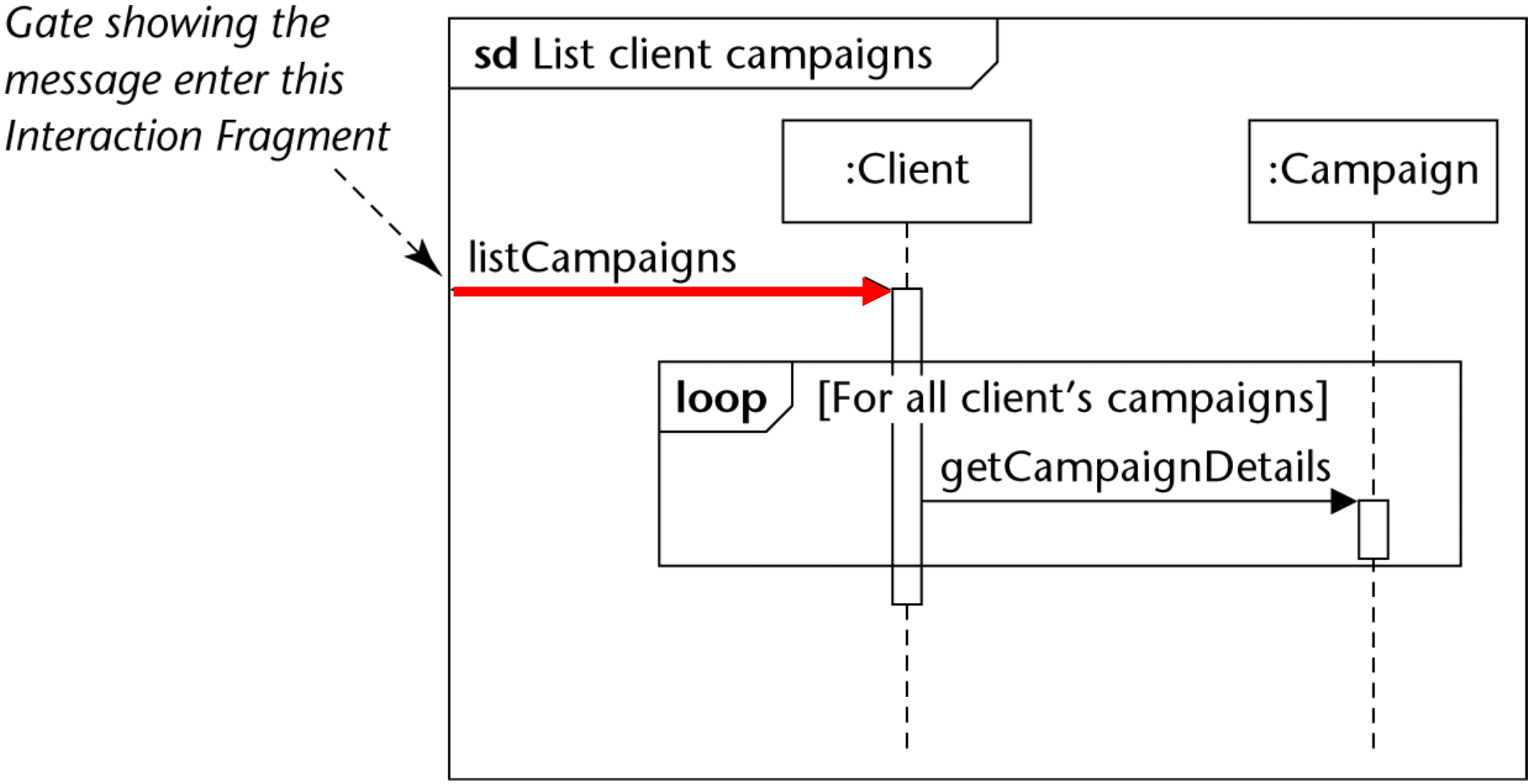
- ***Object destruction*** is indicated by a black cross at the end of an activation (and the lifeline ends)
  - The object is deleted from the system

Interaction fragments (subdiagrams)

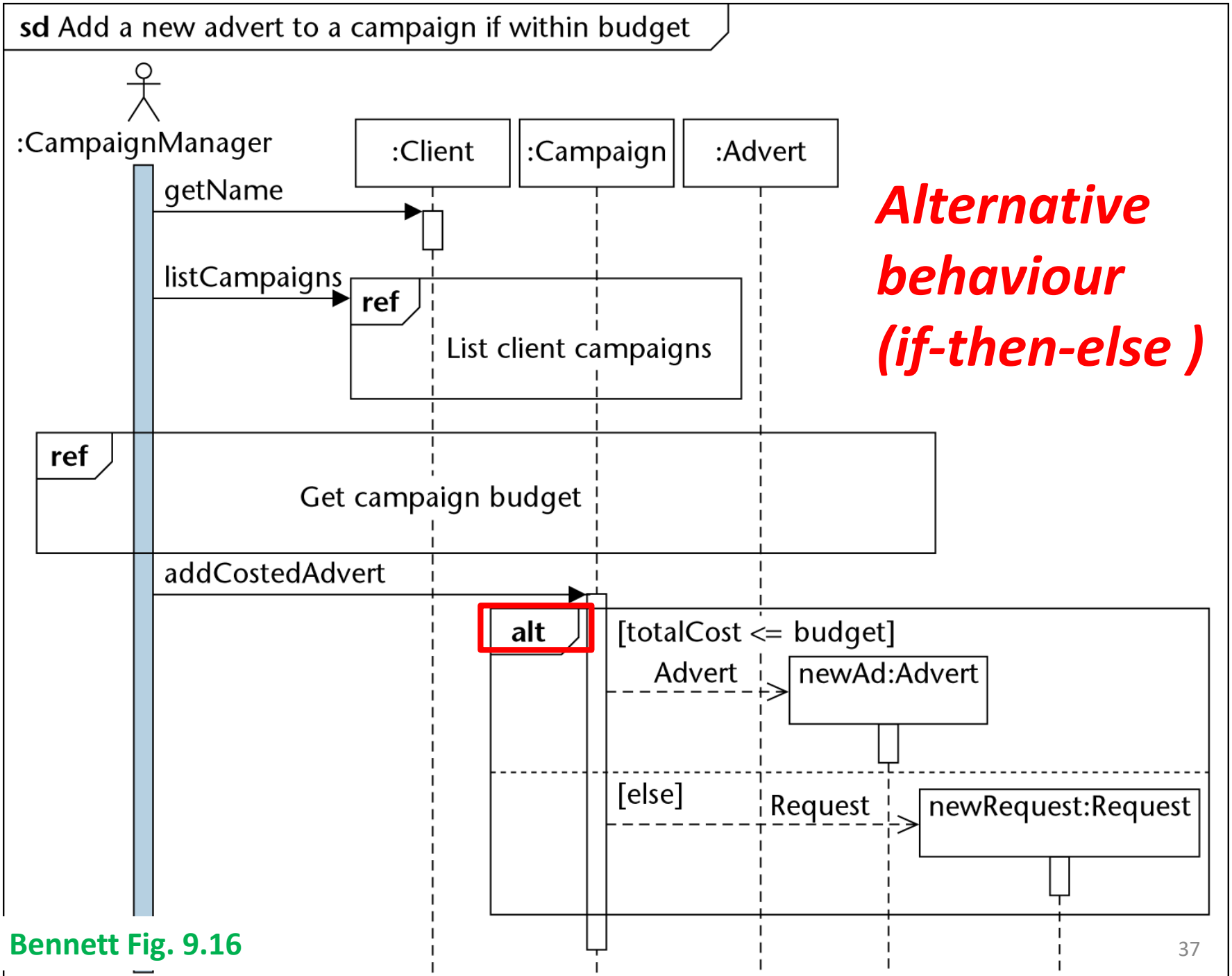


Bennett Fig. 9.11

Interaction fragments (subdiagrams)



Bennett Fig. 9.12



Alternative behaviour (if-then-else)

Bennett Fig. 9.16

Interaction operator	Use
loop	Repeat until the interaction constraint for the loop is no longer true
alt	Alternative behaviours, depending on stated conditions
opt	Will only execute if interaction constraint is true
ref	Refers to an interaction fragment described in a different diagram

Elements of a SD (4)

- An ***Interaction Operator*** (loop, alt, opt) is a kind of control structure
  - Denoted by a box with the operator in the top left corner
  - Different sections of an “alt” are separated by dashed lines

- The ***Interaction Contraint*** indicates whether / how often the enclosed part is executed
  - Denoted “[…]”, use any syntax you like to specify the constraint

Elements of a SD (5)

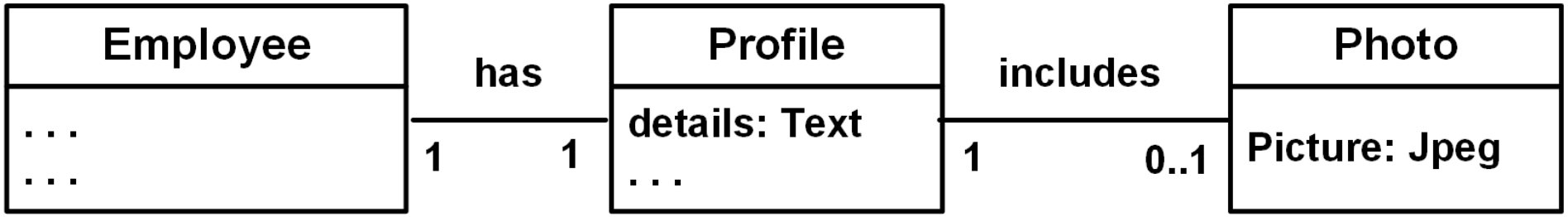
- An ***interaction fragment*** is a sub-diagram that can be referred to from other SDs (using "**ref**")
- Two different ways to do this:
  - With a ***gate***:
    - In the main diagram: arrow to the gate
    - In the interaction fragment: arrow from the gate
  - Without a gate: ← **Easiest to use in the exercises**
    - The **ref** box is a placeholder, during execution it is substituted by interaction fragment

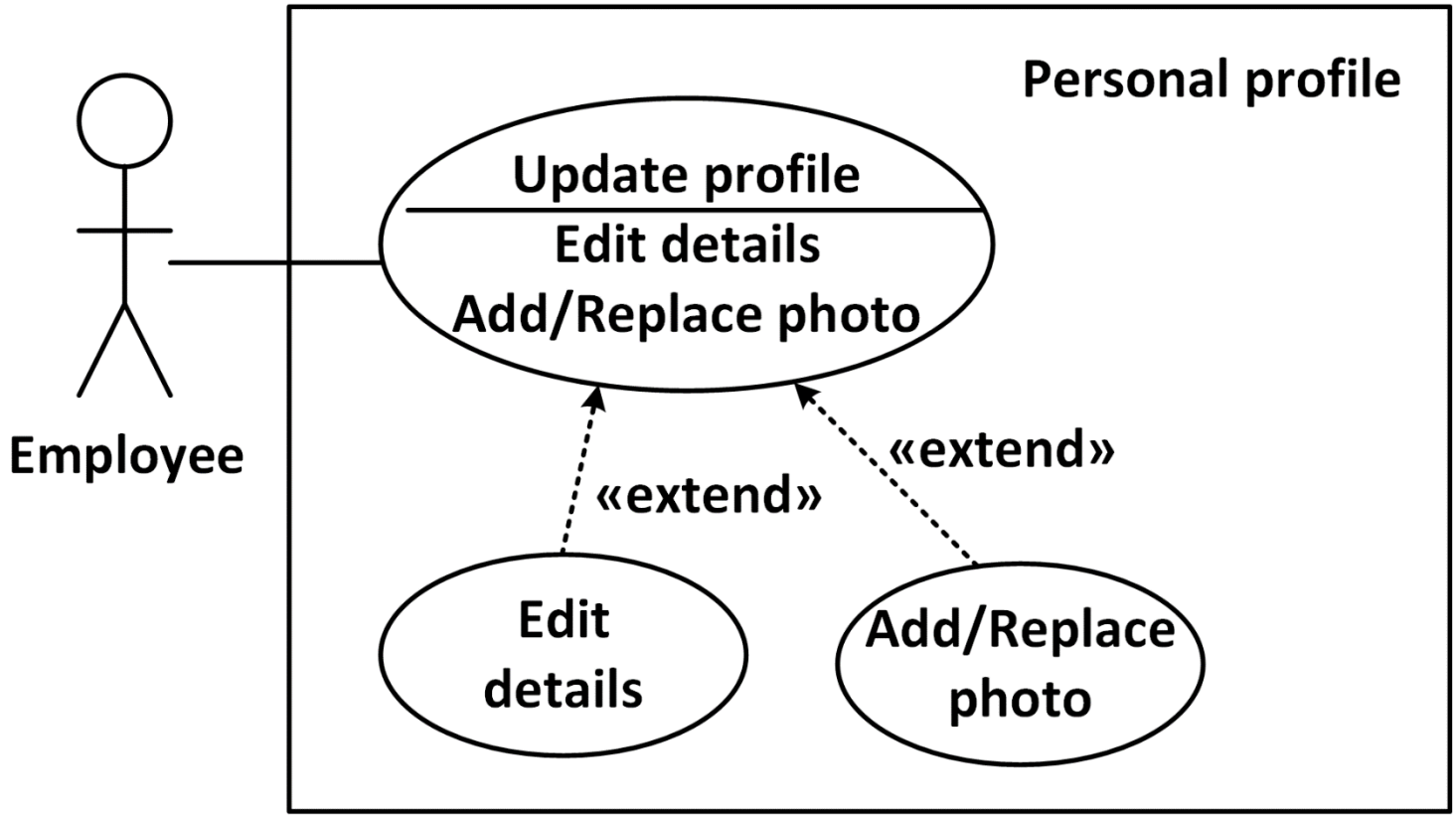
4. Exercise

Company ‘Yellow Pages’

All employees have a personal profile on the company intranet. The profile contains: function; contact info; a description of the employee’s competences; possibly some personal information and a photo.

Employees can update their description and their photo themselves.

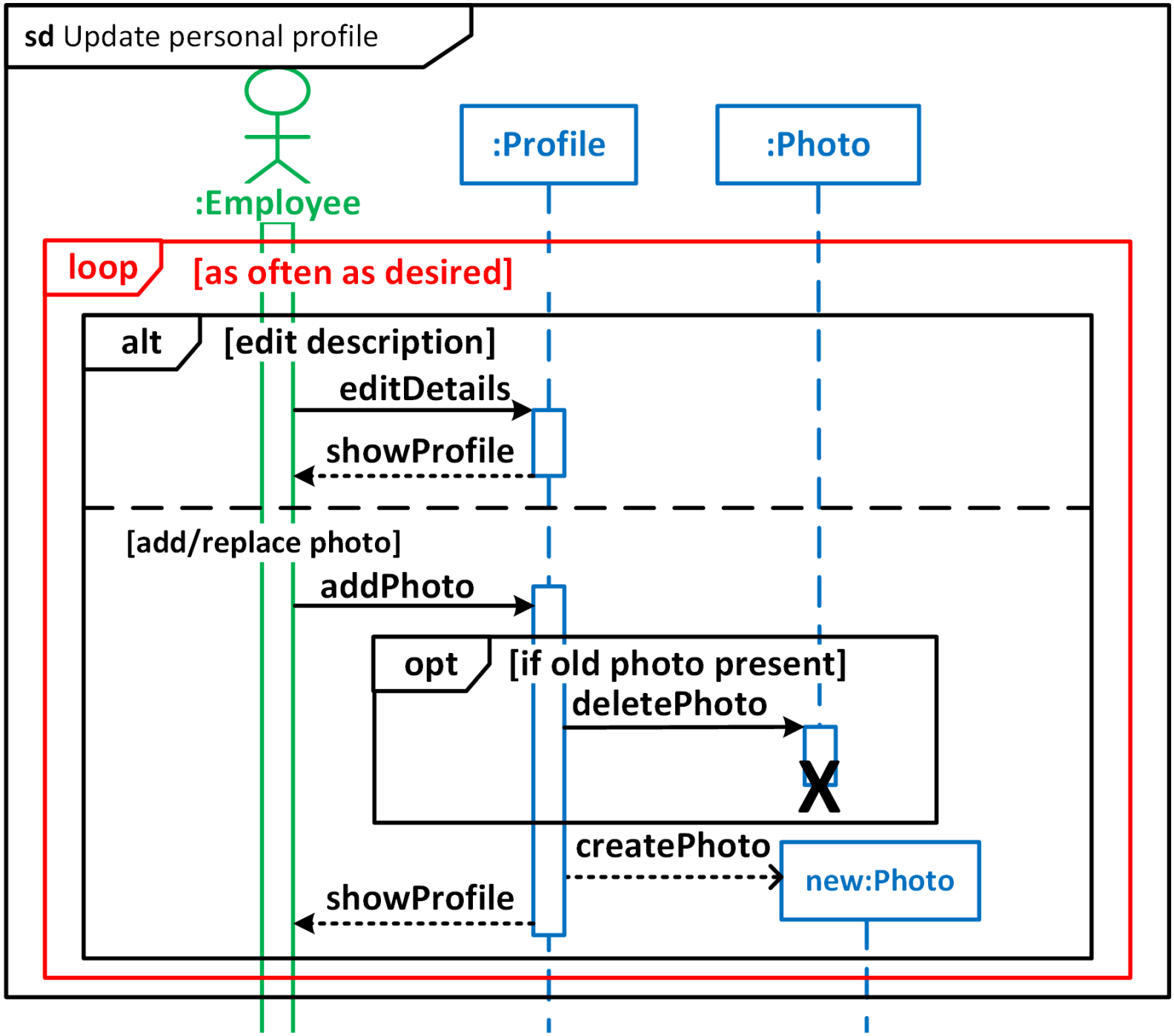

Employee
. . .
. . .
has
1
1
Profile
details: Text
. . .
includes
1
0..1
Photo
Picture: Jpeg


Personal profile
Update profile
Edit details
Add/Replace photo
Employee
«extend»
«extend»
Edit details
Add/Replace photo

- Make a SD* for updating one's personal profile. This could include editing the description and/or oploading a new photo**. Changes can be done in any order and multiple times.

* Don't include a control object

** If an old photo was present, this is deleted when a new photo is uploaded


sd Update personal profile
:Employee
:Profile
:Photo
loop
[as often as desired]
alt
[edit description]
editDetails
showProfile
[add/replace photo]
addPhoto
opt
[if old photo present]
deletePhoto
X
createPhoto
new:Photo
showProfile

Remarks about this solution

- Please note that the existing photo object is deleted (life lines represent objects, not classes)
- The return messages “showProfile” need not be present here. In the library example with the simplified SD, all return messages were abstracted from. But there is no harm in adding it if you want to make explicitly clear that something is shown to the user.

Preview

Lecture 3a:

- State Machine Diagrams
- Retrospective overview of UML

Lecture 3b: Version management (GIT)

- Short lecture, simple exercises
- Gives some room to catch up backlog

Lecture 4: Software Metrics