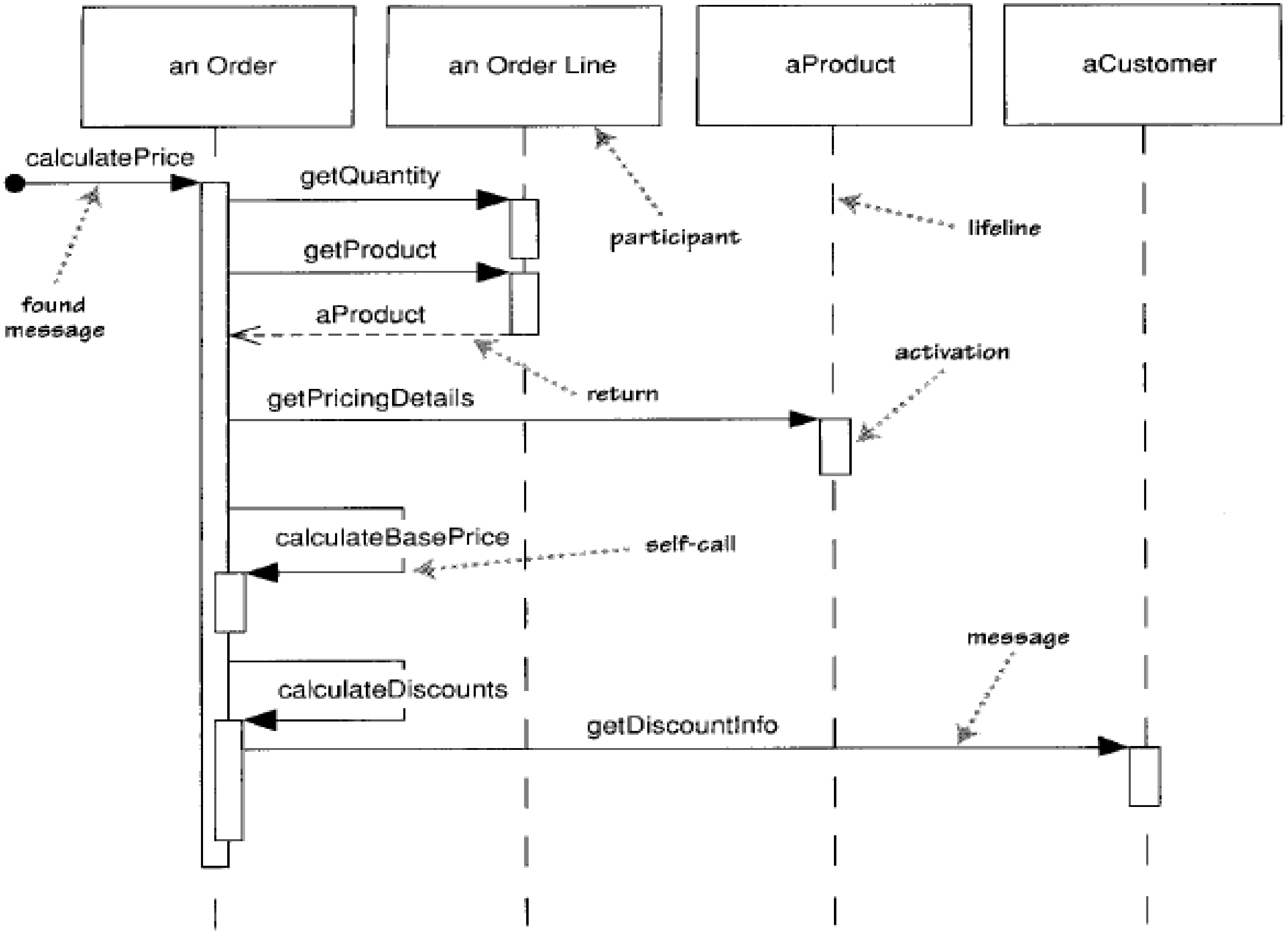
SEQUENCE DIAGRAMS

SEQUENCE DIAGRAMS

- A sequence diagram captures the behavior of a single scenario.
- The diagram shows a number of example objects and the messages that are passed between these objects within the use case.
- Sequence diagrams show the interaction by showing each participant with a lifeline that runs vertically down the page and the ordering of messages by reading down the page.

Example Scenario

- We have an order and are going to invoke a command on it to calculate its price.
- To do that, the order needs to look at all the line items on the order and determine their prices, which are based on the pricing rules of the order line's products.
- Having done that for all the line items, the order then needs to compute an overall discount, which is based on rules tied to the customer.

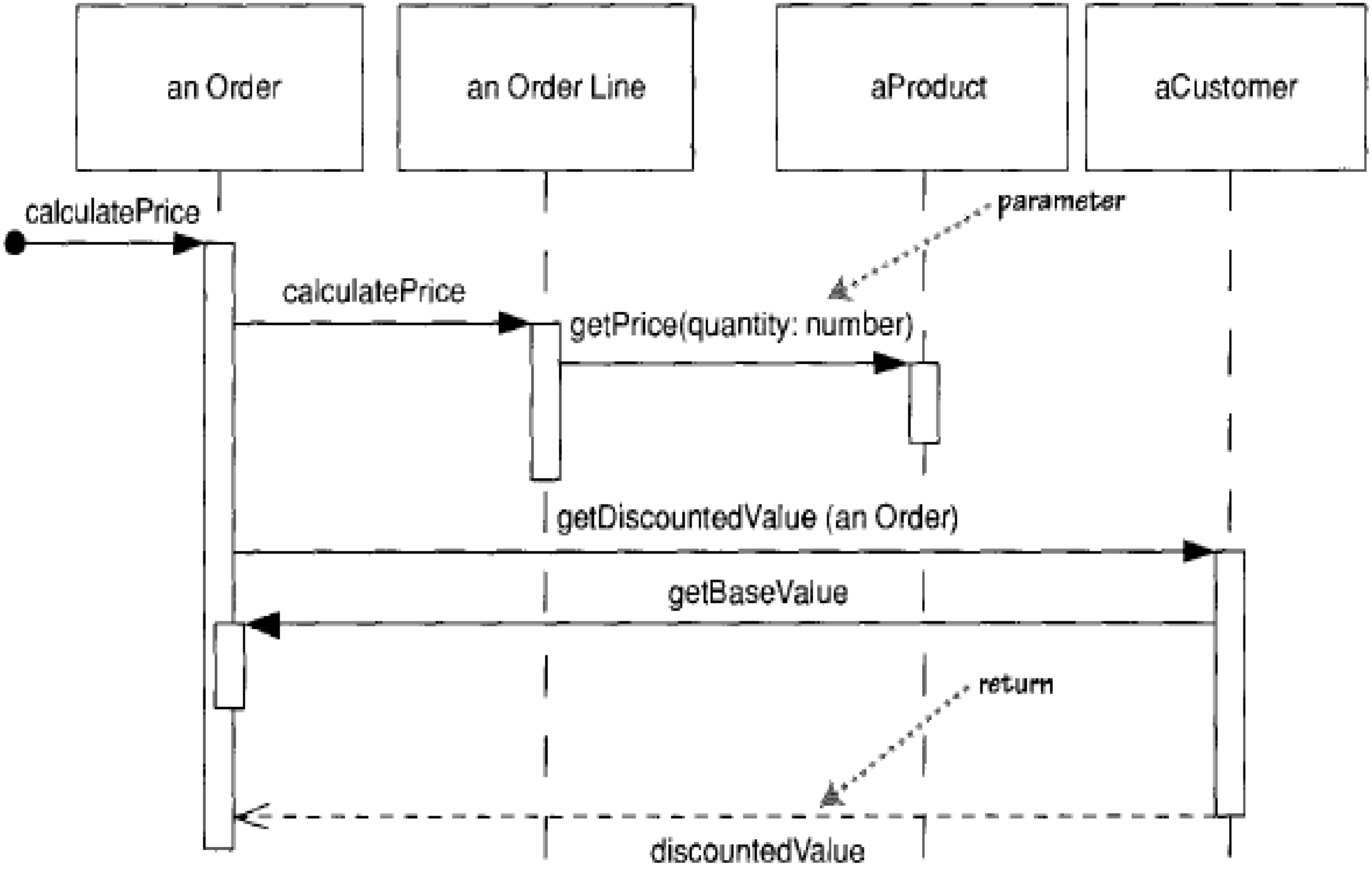

an Order
an Order Line
aProduct
aCustomer
calculatePrice
found message
getQuantity
getProduct
aProduct
participant
lifeline
return
activation
getPricingDetails
calculateBasePrice
self-call
calculateDiscounts
message
getDiscountInfo

Example Scenario

- You can see that, an instance of order sends *getQuantity* and *getProduct* messages to the order line.
- You can also see how we show the order invoking a method on itself and how that method sends getDiscountInfo to an instance of customer.
- The diagram does not show everything very well.
  - The sequence of messages *getQuantity, getProduct, getPricingDetails*, and *calculateBasePrice* needs to be done for each order line on the order, while *calculateDiscounts* is invoked just once.

SEQUENCE DIAGRAMS

- Each lifeline has an activation bar that shows when the participant is active in the interaction.
- This corresponds to one of the participant's methods being on the stack .


an Order
an Order Line
aProduct
aCustomer
calculatePrice
parameter
calculatePrice
getPrice(quantity: number)
getDiscountedValue (an Order)
getBaseValue
return
discountedValue

Another Sequence Diagram

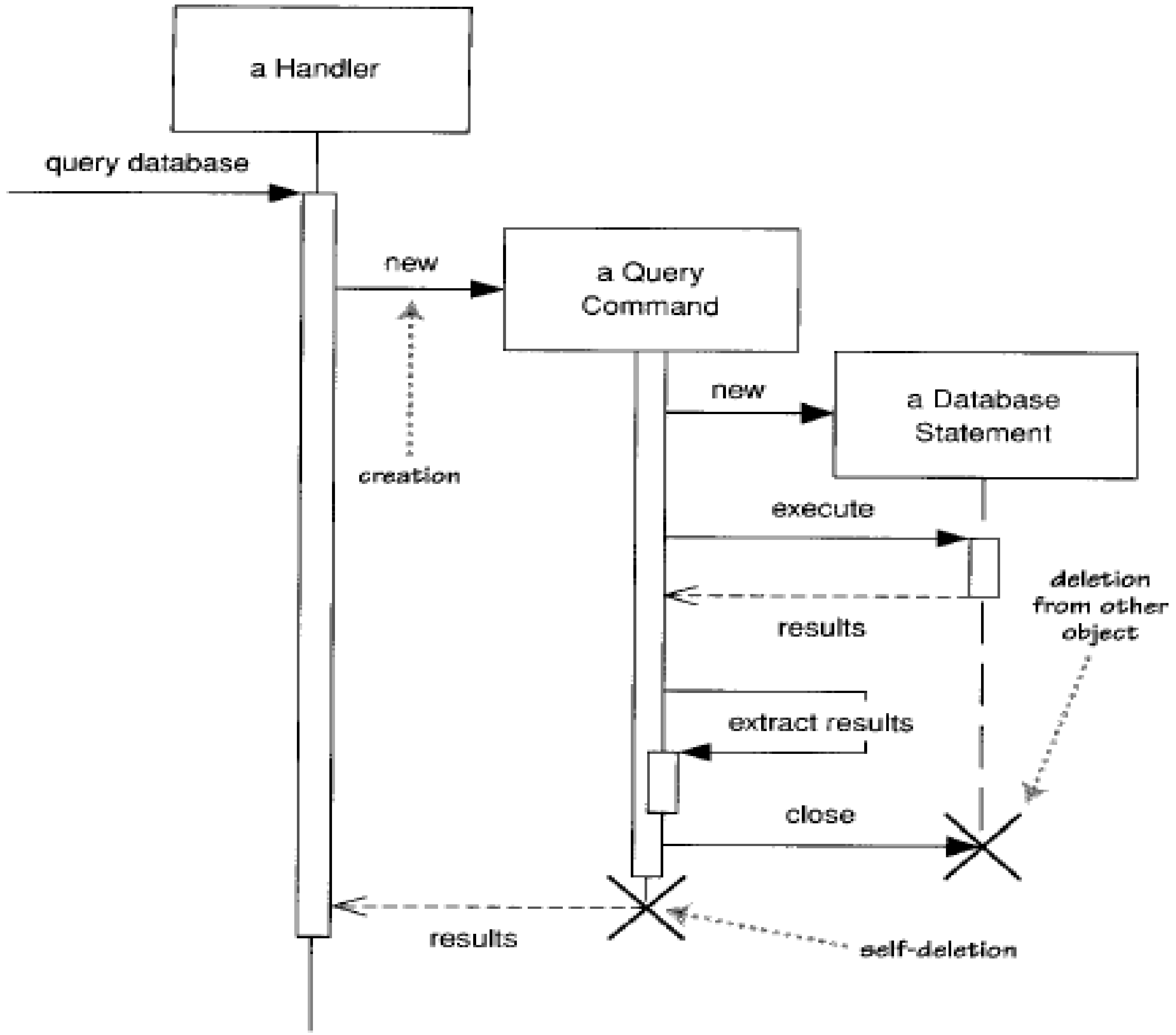
- The Order asks each Order Line to calculate its own Price.
- The Order Line itself further hands off the calculation to the Product;
  - note how to show the passing of a parameter.
- Similarly, to calculate the discount, the Order invokes a method on the Customer.
- Because it needs information from the Order to do this, the Customer makes a reentrant call (*getBaseValue*) to the Order to get the data .

Centralized vs. Distributed Control

- The first exmple uses **centralized control**, with one participant pretty much doing all the processing and other participants there to supply data.
- The second example uses **distributed control**, in which the processing is split among many participants, each one doing a little bit of the algorithm.

Create/Delete Objects

- Sequence diagrams show some extra notation for creating and deleting participants.
- To create a participant, you draw the message arrow directly into the participant box.
- A message name is optional here if you are using a constructor, but I usually mark it with "new" in any case.
- If the participant immediately does something once it's created, such as the query command, you start an activation right after the participant box.
- Deletion of a participant is indicated by big X.
- A message arrow going into the X indicates one participant explicitly deleting another; an X at the end of a lifeline shows a participant deleting itself.


a Handler
query database
new
creation
a Query Command
new
a Database Statement
execute
results
deletion from other object
extract results
close
results
self-deletion

When to Use Sequence Diagrams

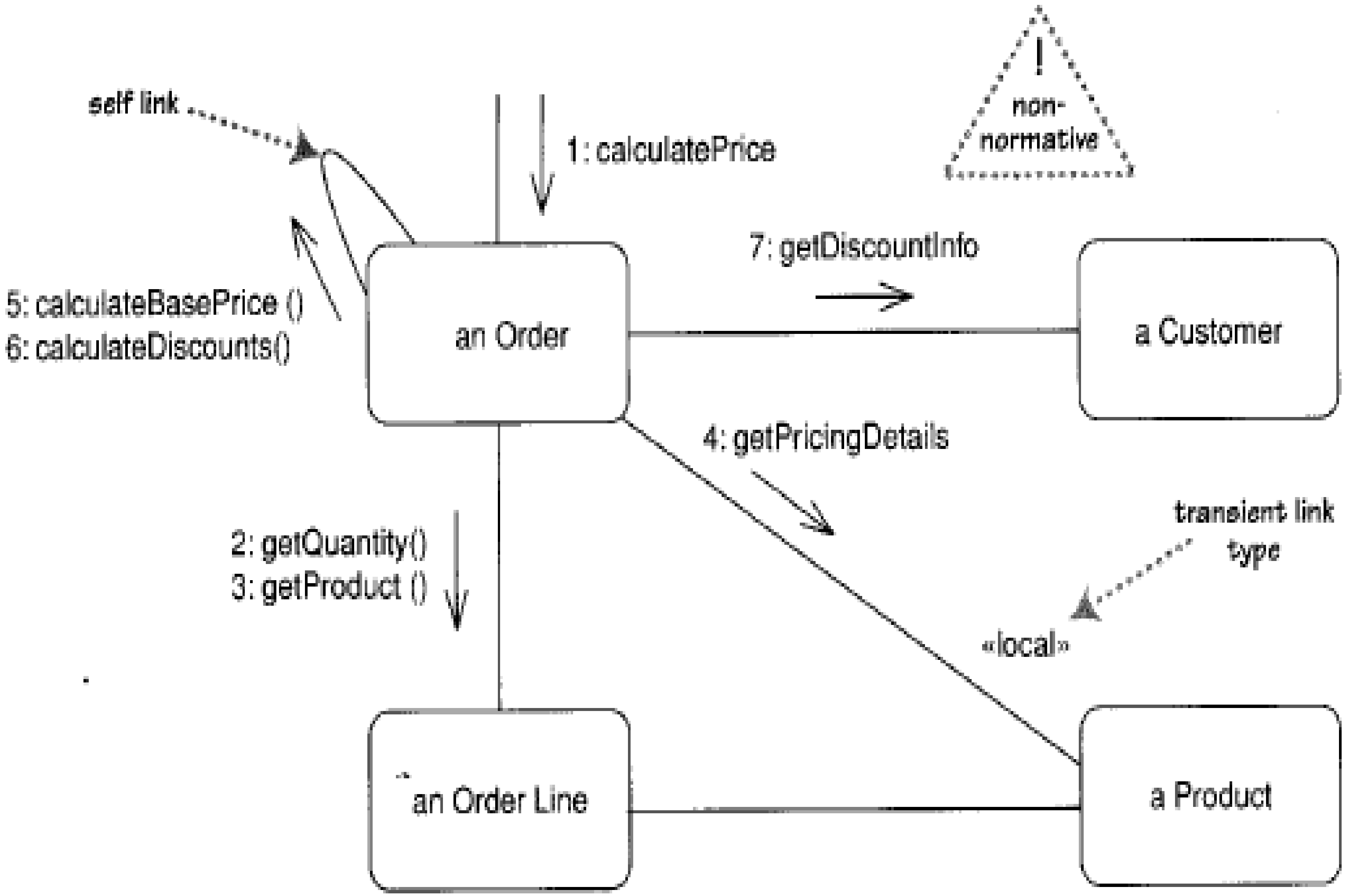
- You should use sequence diagrams when you want to look at the behavior of several objects within a single use case.
- Sequence diagrams are good at showing collaborations among the objects.
- They are not so good at precise definition of the behavior.

- If you want to look at the behavior of a single object across many use cases,
  - use a state diagram
- If you want to look at behavior across many use cases or many threads,
  - use an activity diagram

COMMUNICATION DIAGRAMS

Communication vs. Sequence Diagrams

- Communication diagrams, emphasize the data links between the various participants in the interaction.
- Instead of drawing each participant as a lifeline and showing the sequence of messages by vertical direction as the sequence diagrams does,
  - the communication diagram allows free placement of participants, allows you to draw links to show how the participants connect, and use numbering to show the sequence of messages.
- With a communication diagram, we can show how the participants are linked together.


self link
1: calculatePrice
non-normative
7: getDiscountInfo
5: calculateBasePrice ()
6: calculateDiscounts()
an Order
a Customer
4: getPricingDetails
transient link type
2: getQuantity()
3: getProduct ()
«local»
an Order Line
a Product

When to Use Communication Diagrams

- The main question with communication diagrams is when to use them rather than the more common sequence diagrams.
  - A strong part of the decision is personal preference
- A more rational approach says that
  - sequence diagrams are better when you want to emphasize the sequence of calls and
  - communication diagrams are better when you want to emphasize the links