

Interaction Models: Sequence and Collaboration

- Consider the family of UML diagrams
- Dynamic models (a UML view)
- Interaction models
 - Collaboration diagrams
 - Sequence diagrams
 - Alternatives?
- Conclusions

Some UML Models

- Use cases
- Class diagrams (& object diagrams)
- State charts
- Interaction diagrams (sequence and collaboration)
(Considered dynamic)
- Activity diagrams *(Considered dynamic)*
- Implementation diagrams (component & deployment). Packages and subsystems

Dynamic in UML?

- Use case modelling.
- Interaction modelling:
 - Collaboration diagram
 - Sequence diagram
- State modelling.
 - Statechart
 - Activity Diagram

Process modellers may consider all static

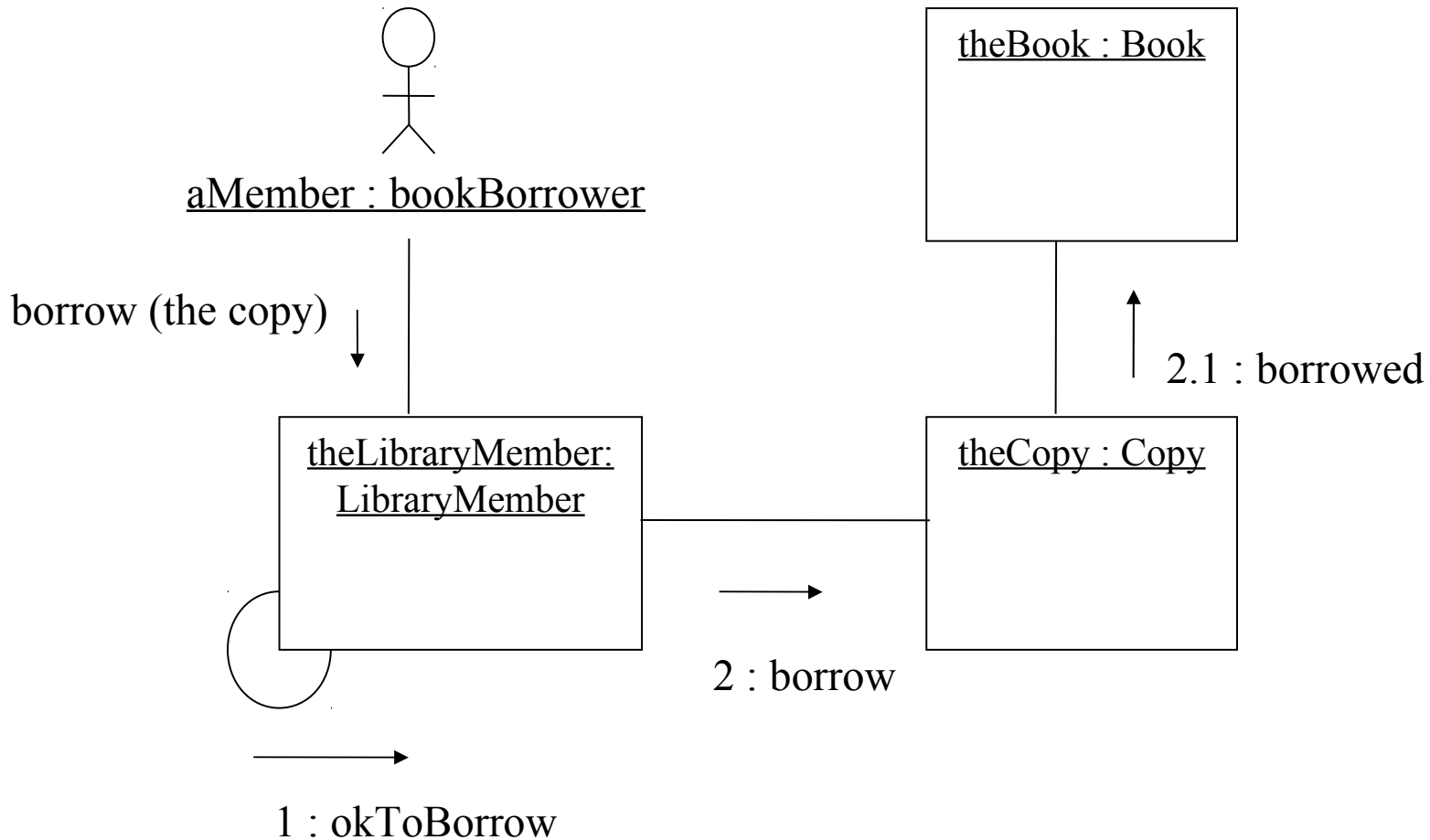
Collaboration

- Objects within systems communicate with each other; they send messages to each other.
- A message is typically just an operation (method) call that one object invokes on another object.
- Communication among a set of objects in order to generate some function is called an *interaction*.

Collaboration Diagram

- Collaboration diagrams describe how objects interact.
- Collaboration diagrams focus on space.
- This means that the relationships (links) between the objects (in space) are of particular interest.
- A collaboration diagram is isomorphic (to a degree) with a sequence diagram.

Collaboration Diagram Example



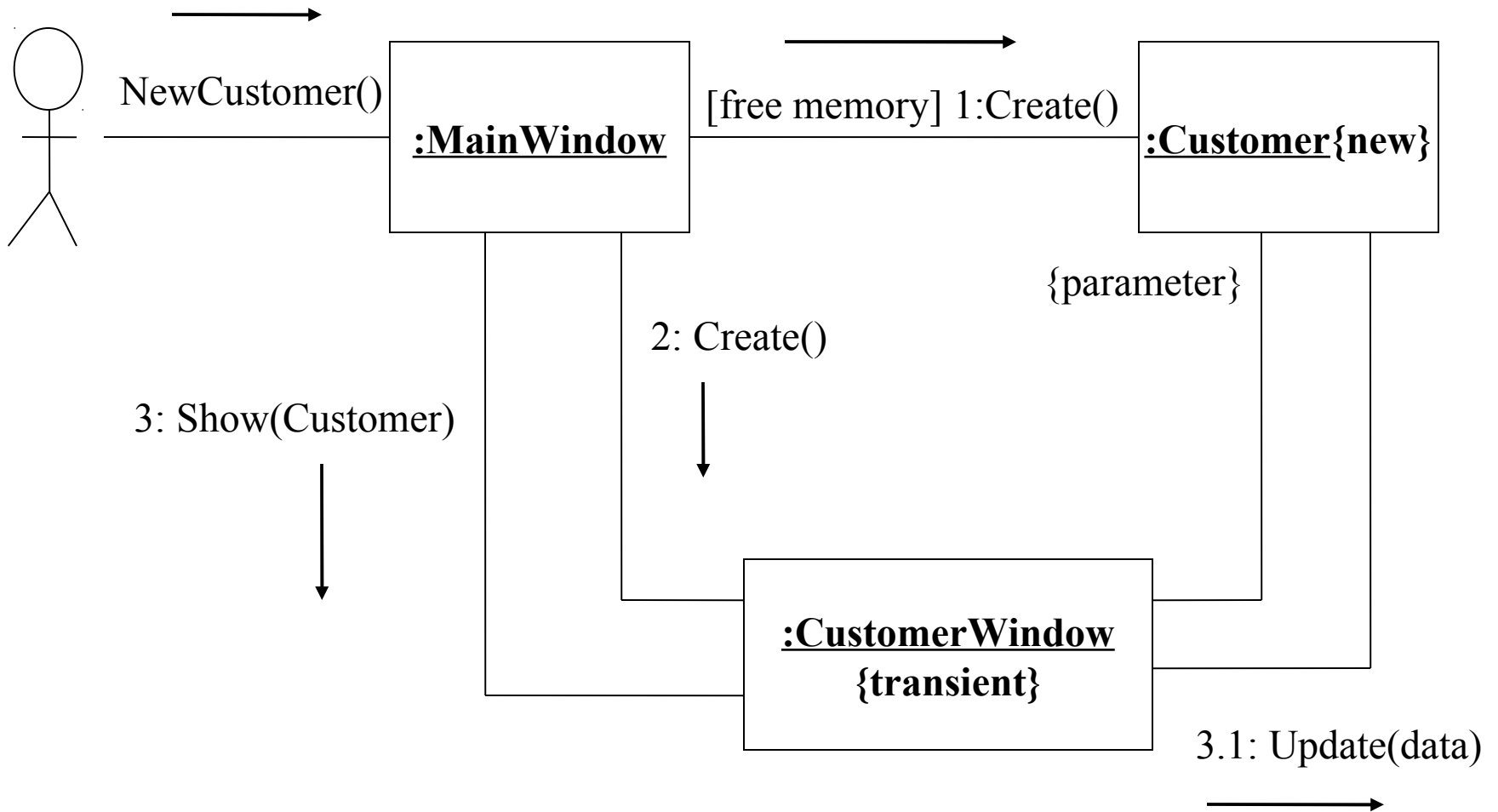
Activations

- For procedural interactions:
 - One object active (computing) at a time.
 - Object activated on receipt of message.
 - Will (eventually) send response. During this live activation period may:
 - Carry out computation OR
 - Send further messages (passing control)
 - Reflected in nested numbering

Collaboration: Object existence


- Objects that are created during a collaboration are designated with **{new}** and objects that are destroyed in a collaboration are designated by the constraint **{destroyed}**.
- Objects that are both created and destroyed during the same collaboration are designated as **{transient}** which is the same as both **{new}** **{destroyed}**


Existence Example





Arrows:

Message types notation

 Synchronous

 Asynchronous

 Simple

 Synchronous with
immediate return

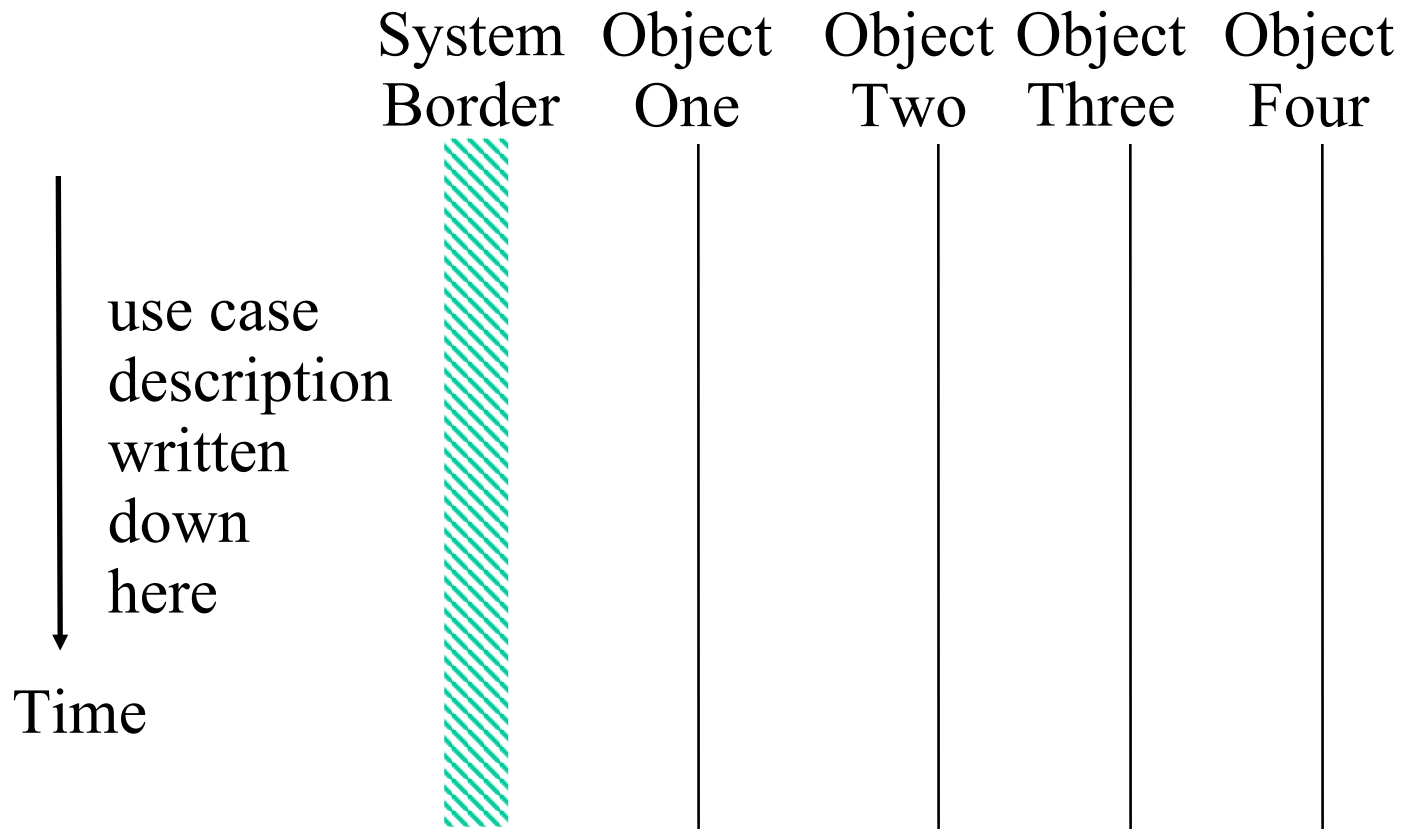
Meaning

- Synchronous: a nested flow of control, typically implemented as an operation call.
 - The operation that handles the message is completed before the caller resumes execution.
- Asynchronous: There is no explicit return to the caller.
 - The sender continues to execute after sending the message without waiting for it to be handled. Used in real-time systems.
- Simple: Shows that control is passed from one object to another without showing any detail.
 - Also used to show the return of a synchronous message.

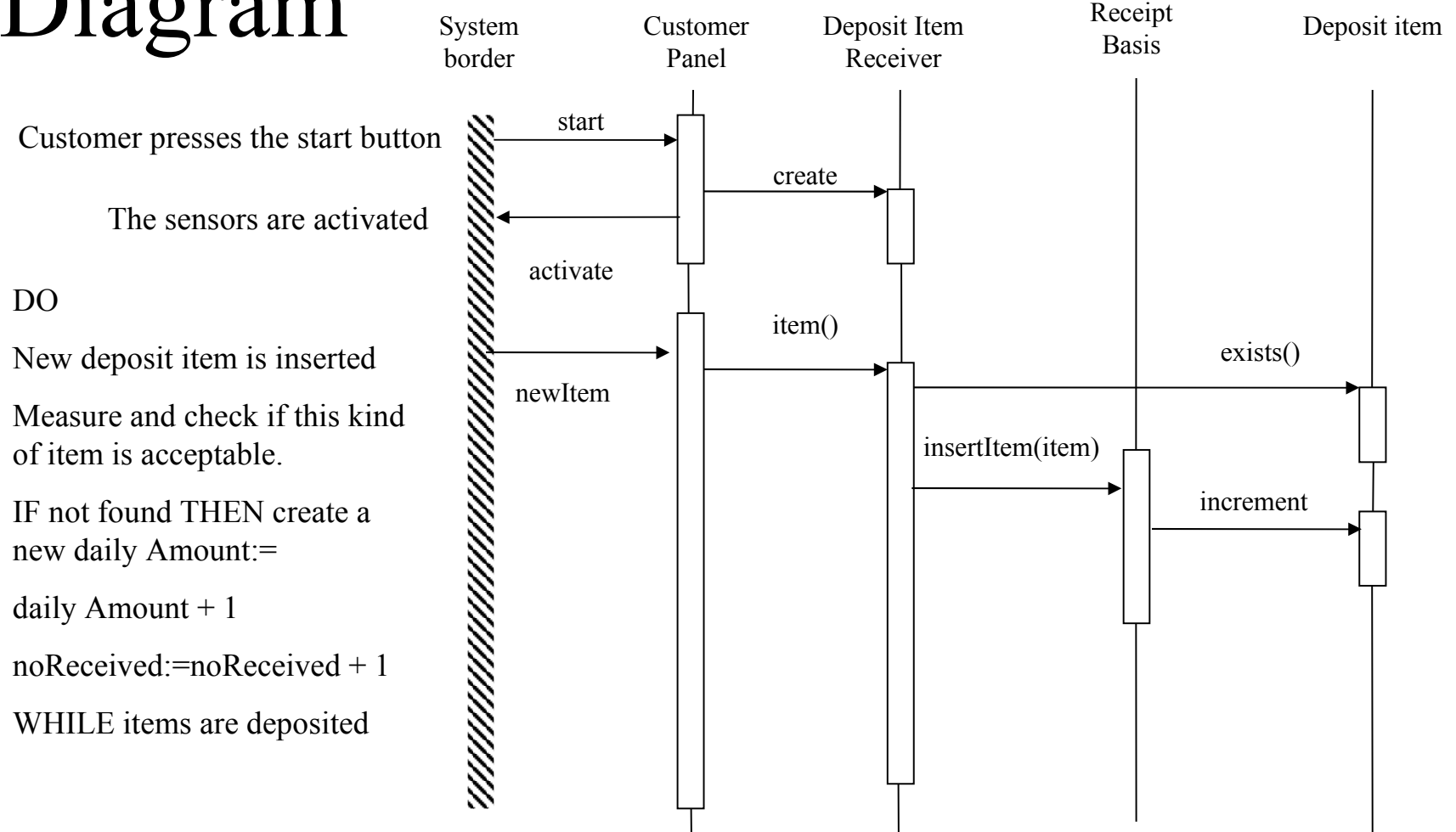
Sequence

- Sequence diagrams illustrate how objects interact with each other to perform some function.
- They focus on message sequences - how messages are sent and received between a number of objects.
- Sequence diagrams have 2 axes:
 1. Horizontal - the set of interacting objects
 2. Vertical - time order
- *Sequence diagrams can be used to “design” use case descriptions (OOSE approach)!*

Sequence diagrams and use cases



Use Case Design - Sequence Diagram



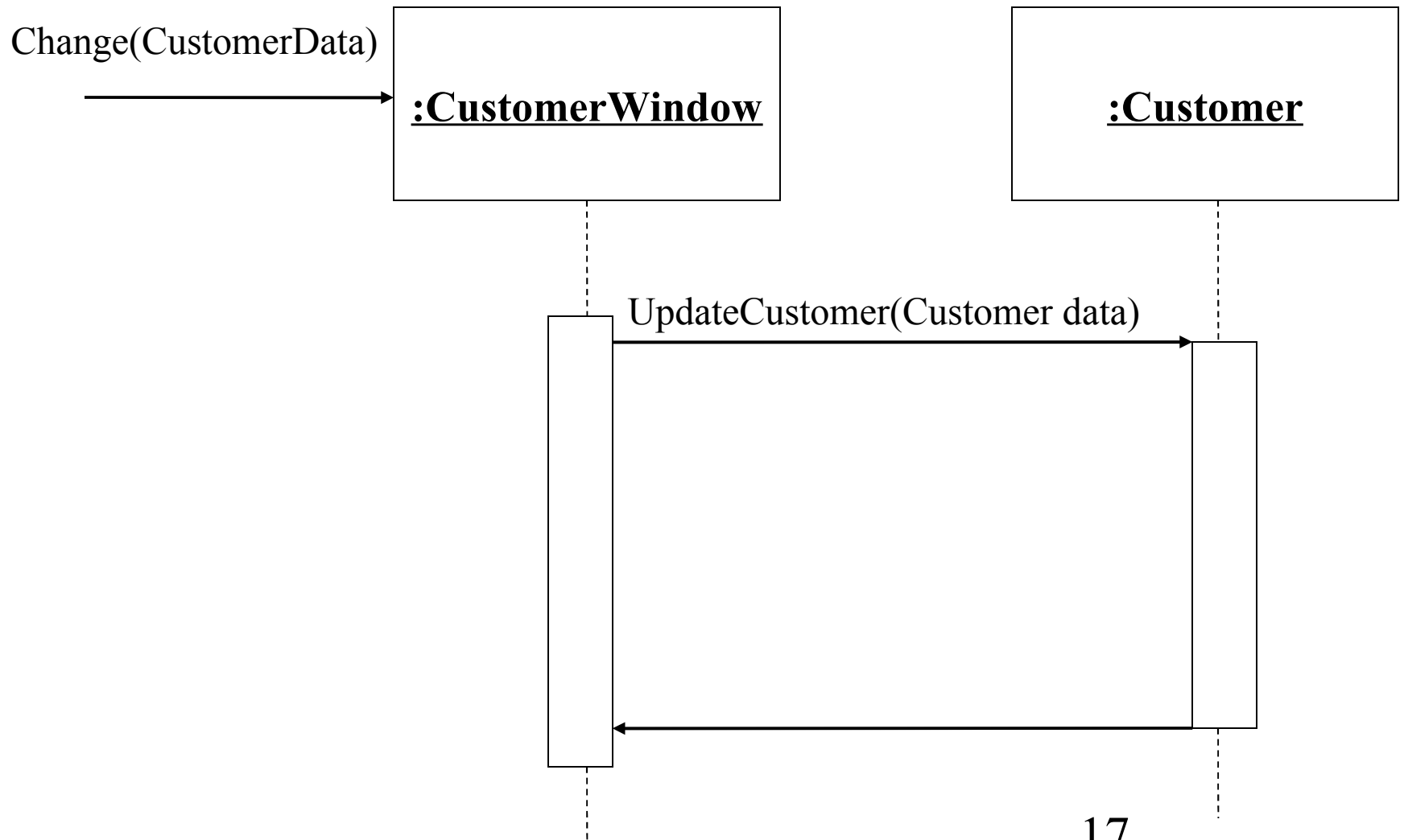
Sequence and Use case comments

- Interesting that even OOSE, and by implication UML sees the need for sequencing issues to be addressed for use cases
 - even though descriptions can't address them.
- *Of course we might suggest other ways (e.g., process models) to design our use cases, and include similar issues.*

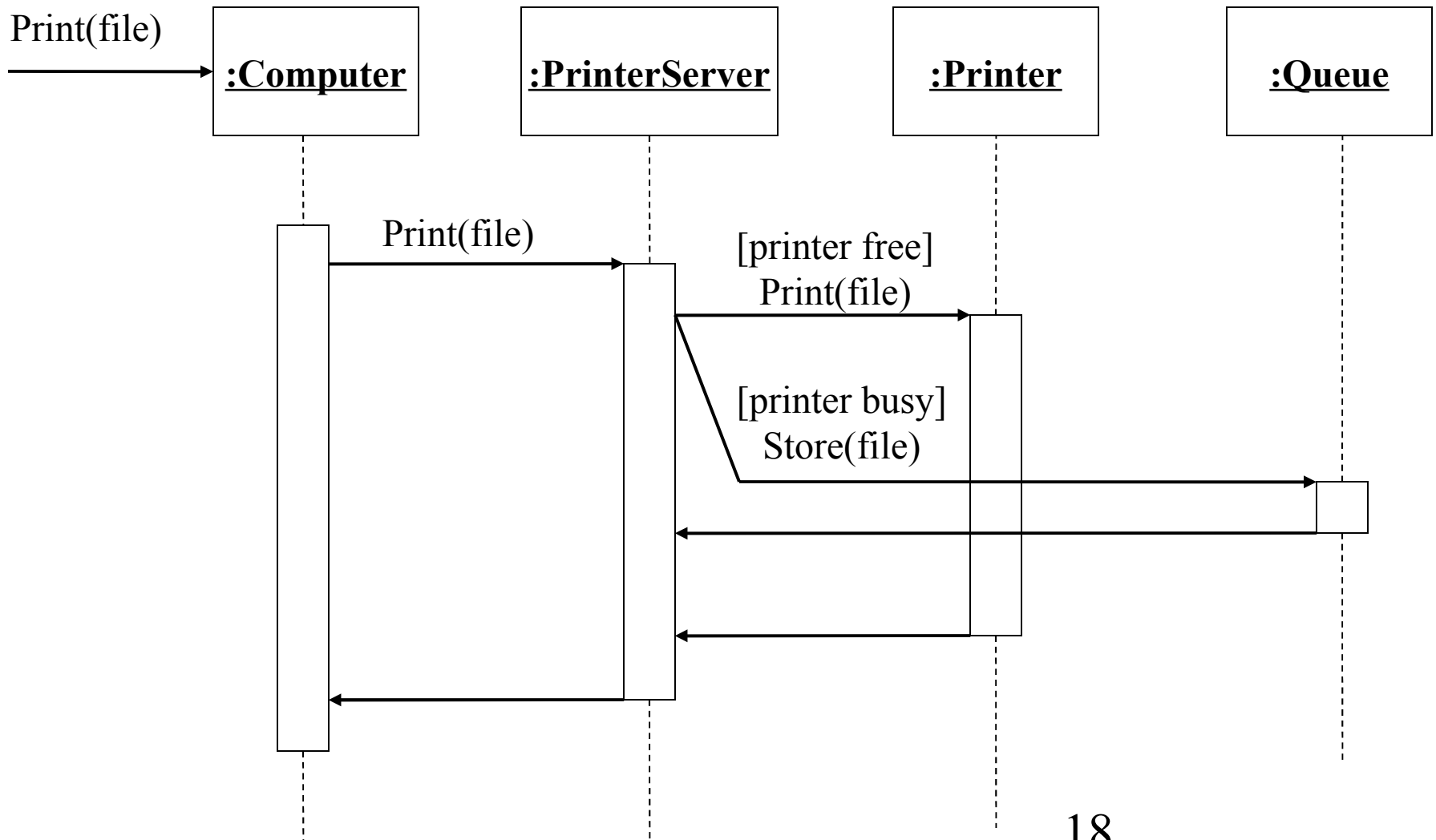
Sequence Forms

- Sequence diagrams can be used in 2 forms:
 1. Instance or 2. Generic
- Instance Sequence Diagram:
 - describes a specific scenario in detail
 - it documents one possible interaction.
 - The instance form does not have any conditions, branches or loops.
- Generic Sequence Diagram:
 - a use case description with alternative courses.

Instance Sequence Diagram



Generic Sequence Diagram

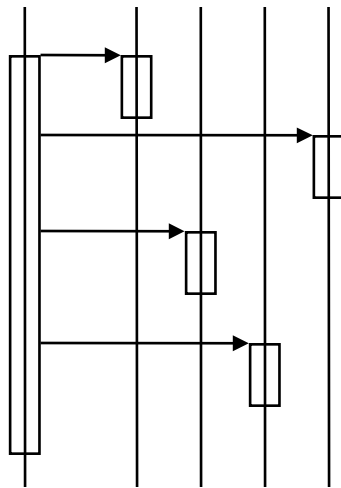


Stair versus Fork

- **Stair:** contain all necessary functionality in only one control object and reuse the object either by inheritance or polymorphism.
- **Fork:** Or make things slightly easier to the novice and just call the necessary object with its localised behaviour.
- Some might call it nit-picking or irrelevant but it is a design issue to be thought about.

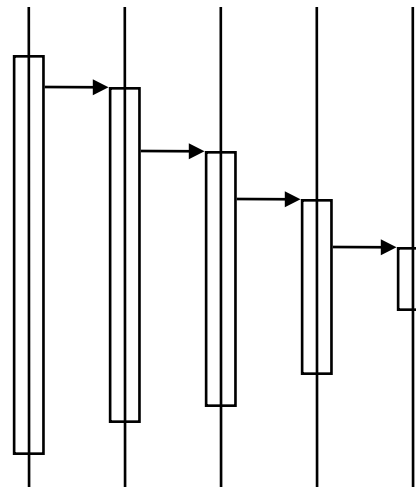
Sequence diagram shapes

Fork - centralised



- operations can change order
- new operations may be added

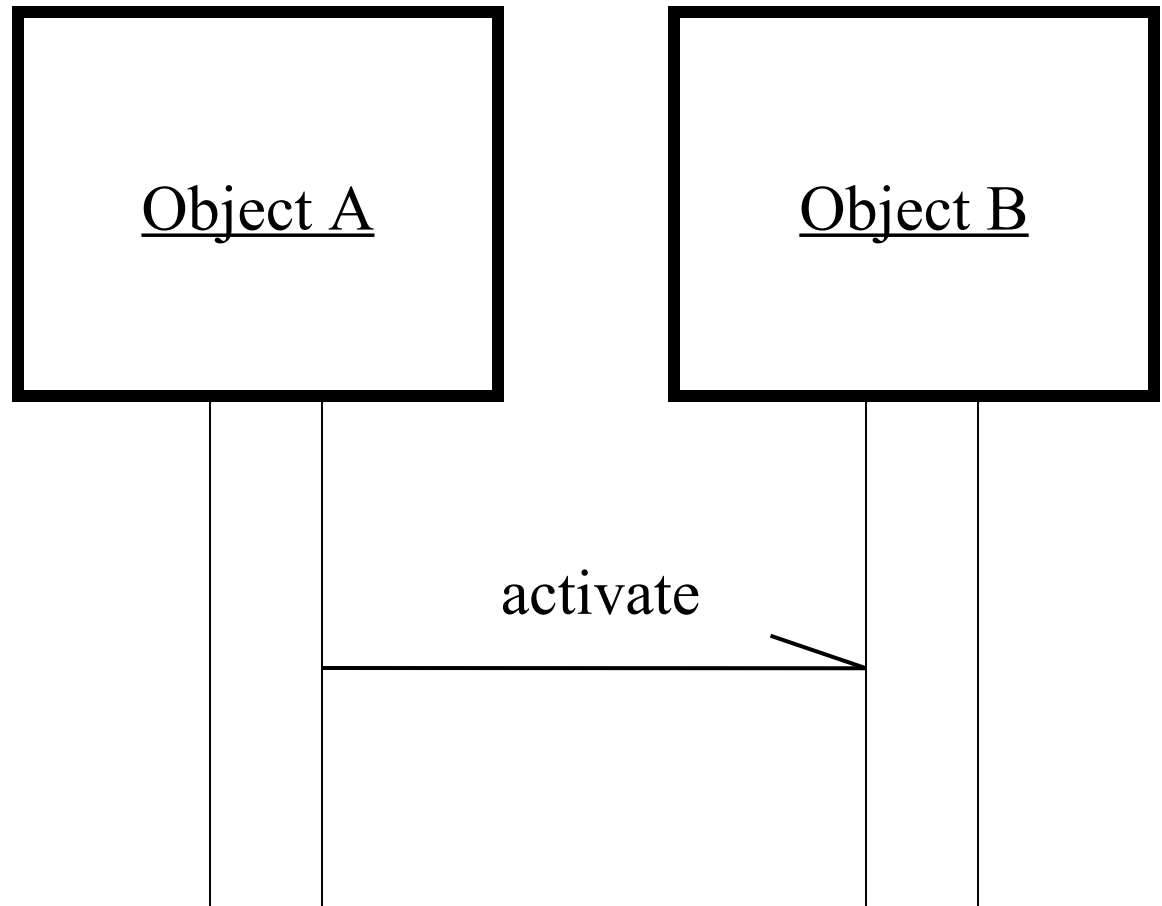
Stair - decentralised



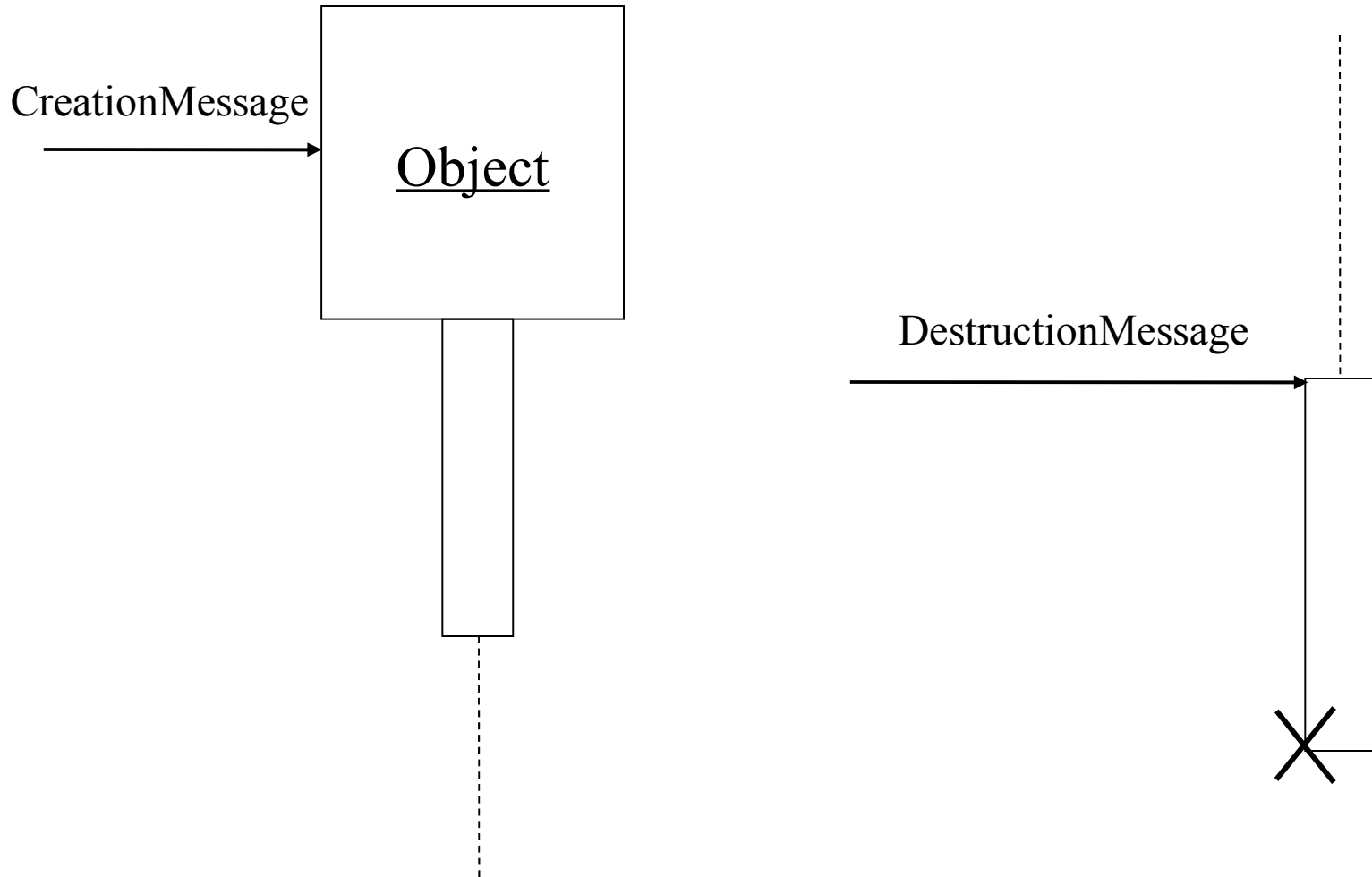
- Operations have strong connections
- performed in same order
- behaviour is encapsulated

Concurrency

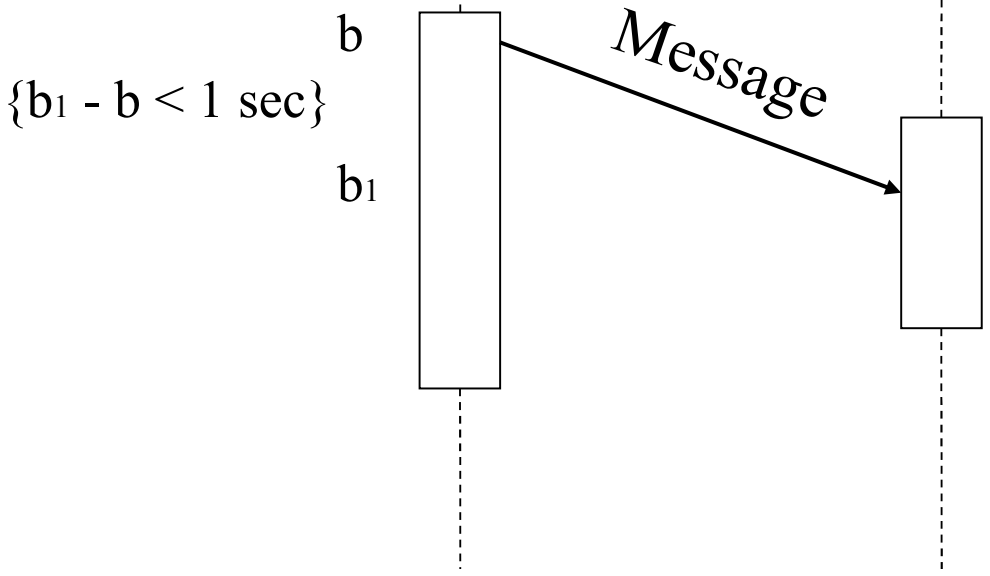
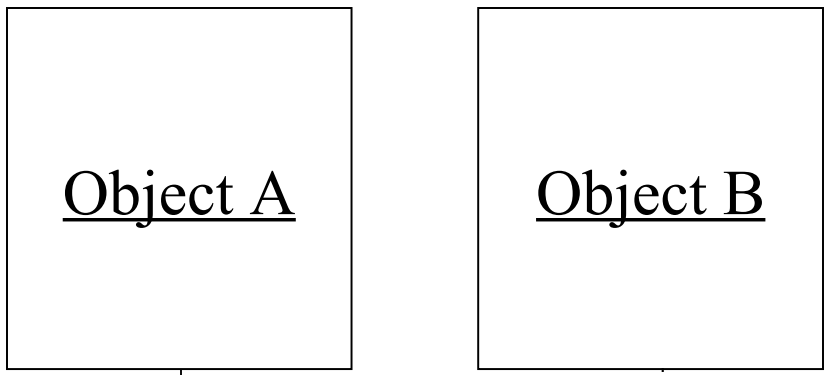
In some systems, objects run concurrently, each with its own thread of control. If the system uses concurrent objects, it is shown by activation, by asynchronous messages and by active objects.



Creation and Destruction

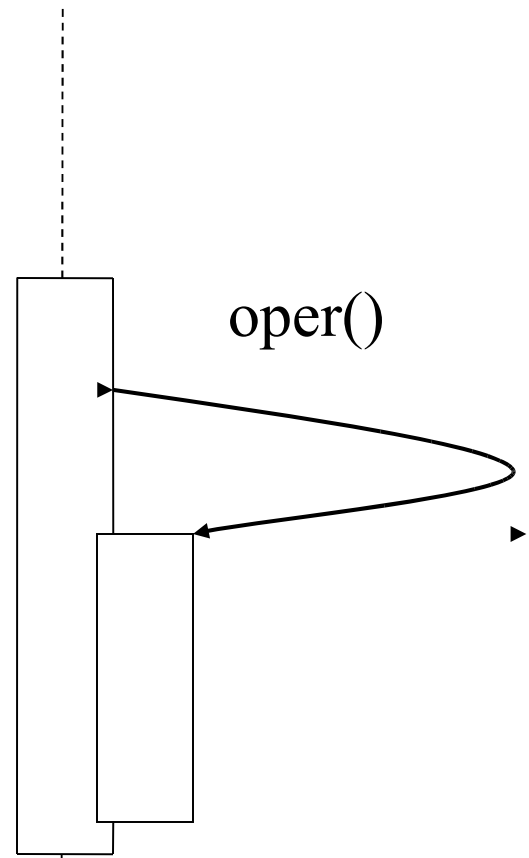


Other stuff



Message with Transmission Time

Recursion:



Conclusion: Interaction Diagrams

- Collaboration diagrams show the links between objects and the numbered sequence of messages - they show SPACE.
- Sequence diagrams are isomorphic to Collaboration diagrams in that they show messages between objects. But sequence diagrams are more concerned with TIME.
- Both diagrams are important to modelling the behavioural design of the system and are excellent for validation and testing.