2IW80 Software specification and architecture

# Interaction diagrams

Alexander Serebrenik

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

# Before we start

True or False?

a) Provided interface is represented by 

b) Provided interface is represented by 

c) Manifestation maps artifacts to nodes

# Before we start

True or False?

a) Provided interface is represented by  True

b) Provided interface is represented by  True

c) Manifestation maps artifacts to nodes

False

Manifestation maps artifacts to use cases / classes / components / packages, relation between artifacts and nodes is called deployment

# Before we start

True or False?

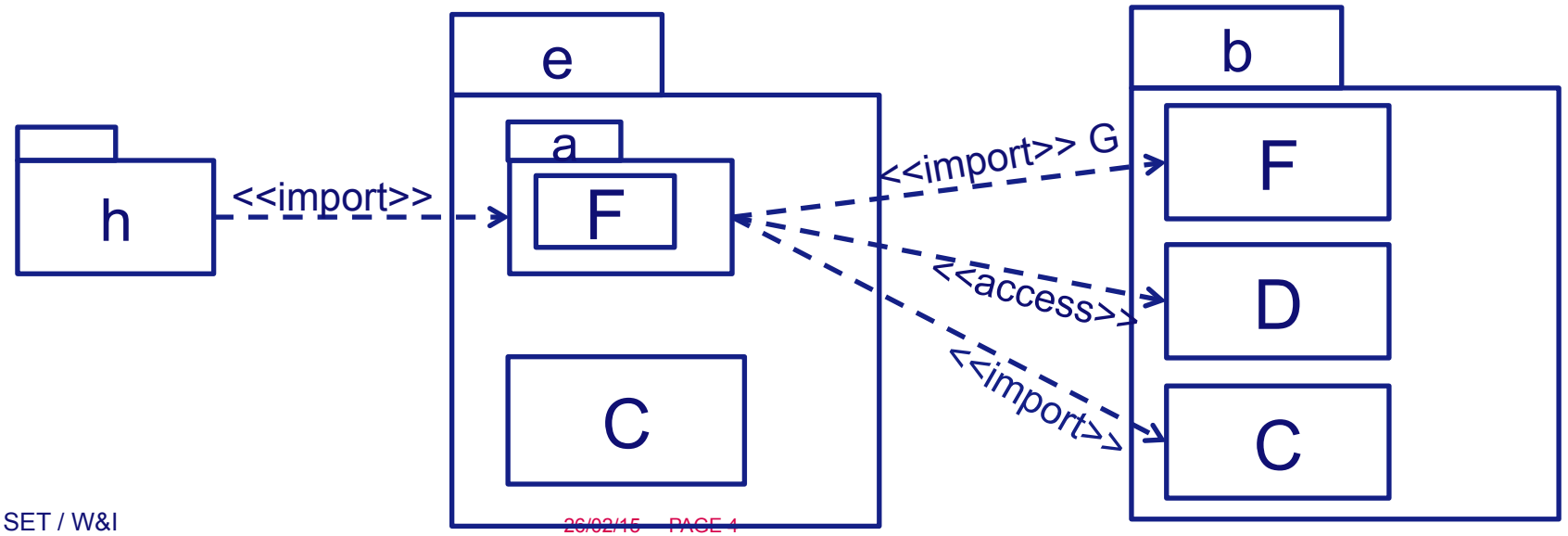
a) Provided interface is represented by ○——

b) Provided interface is represented by ◁- - - -

c) Manifestation maps artifacts to nodes

d) b::C is accessible in h

e) b::D is accessible in h



# Before we start

True or False?

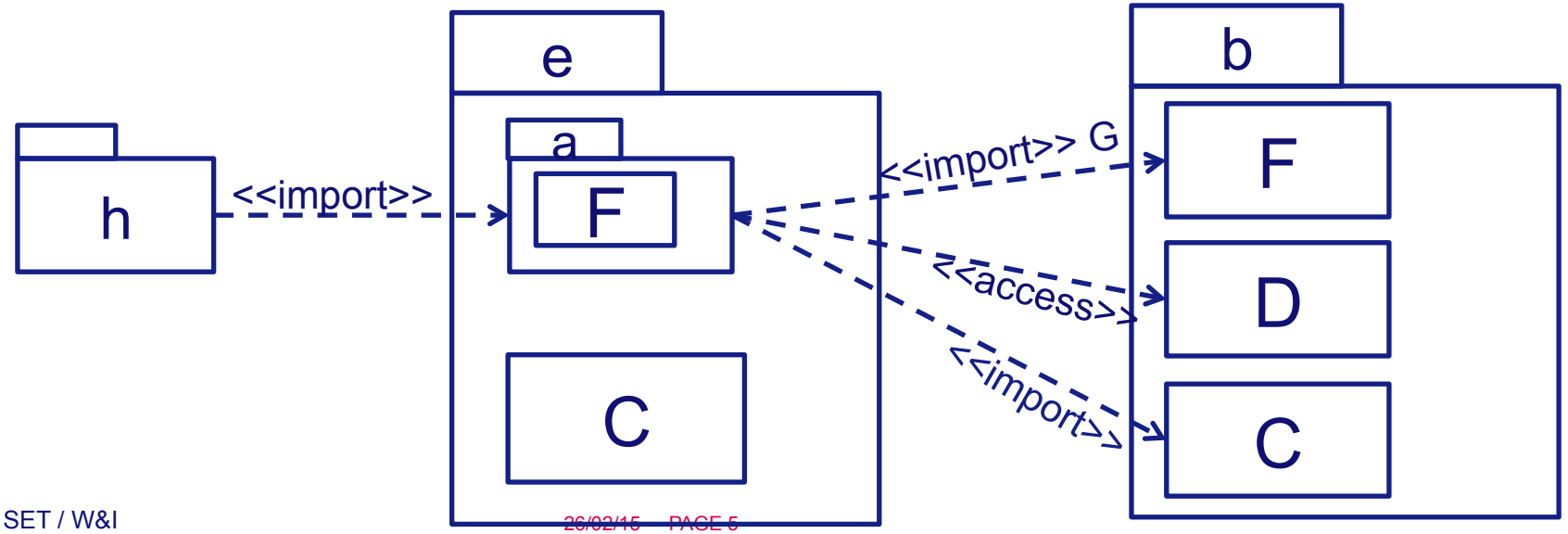
a) Provided interface is represented by 

b) Provided interface is represented by 

c) Manifestation maps artifacts to nodes

d) b::C is accessible in h Yes

e) b::D is accessible in h No



# This week sources

OMG Unified Modeling Language ™ (OMG UML)

*Version 2.5*

## Slides by

David Meredith,
Aalborg University, DK

Marie-Elise Kontro,
Tampere University, FI

## Site by

Kirill Fakhroutdinov
GE Healthcare, USA

# Missing link

- **Use case diagram** describes tasks that system must help actors to perform
- **Class diagram** describes classes required to realize use cases and relationships between these classes
- **Interaction diagrams** describe how objects interact to realize the use case scenarios
- **Interaction diagrams should be consistent with the corresponding class diagrams and use case diagrams**

# Interaction diagrams

Sequence Diagram

TODAY

Communication Diagram

Interaction Overview Diagram

Timing Diagrams

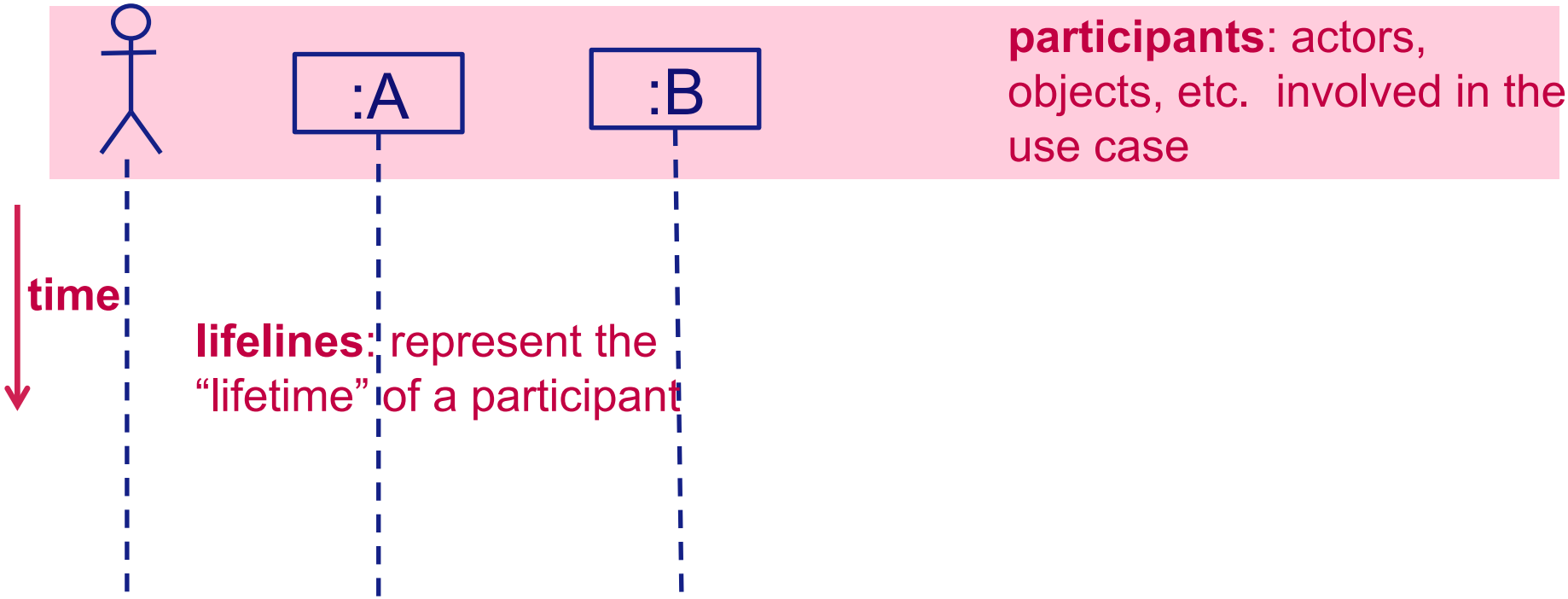
# Sequence diagrams

- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines

- You draw a sequence diagram if, e.g.:
  - you have a use case diagram, to describe how the main components of the system interact
  - you have identified messages arriving at an interface of a component, to describe how the internal parts of the component interact.

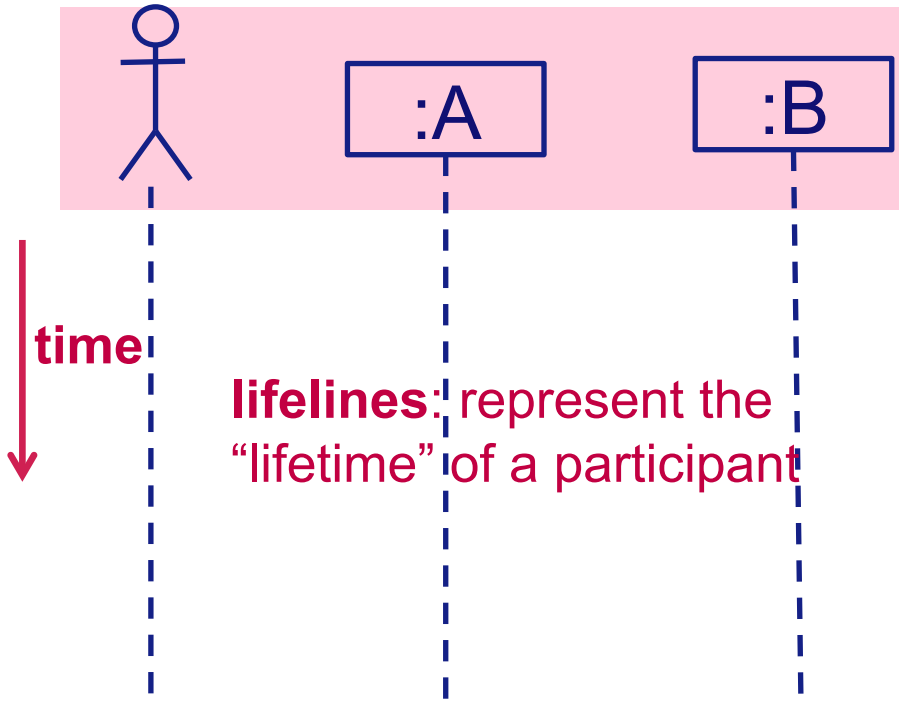
# Sequence diagrams

- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines



# Sequence diagrams

- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines



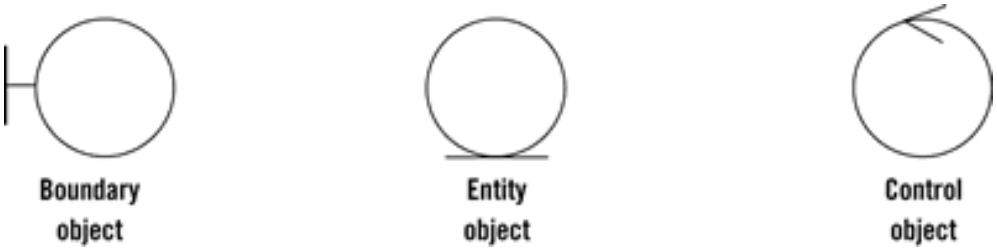

**participants**: actors, objects, etc. involved in the use case.

**Notation** object:Class
- :Class = anObject:Class

Kinds of participants:
- actor, class, database
- boundary, entity, control

http://i.stack.imgur.com/rQPvd.gif

# What are all these things?

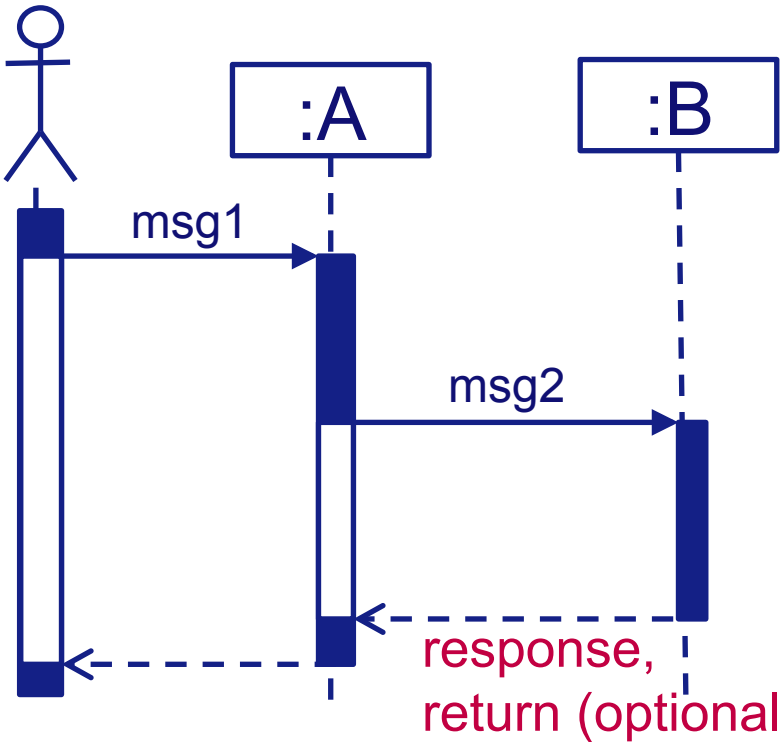



- **Boundary** objects interface with actors.
- **Entity** objects represent system data, often from the domain.
- **Control** objects glue boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.

- Not every system should have boundary/entity/control objects.

# Back to sequence diagrams

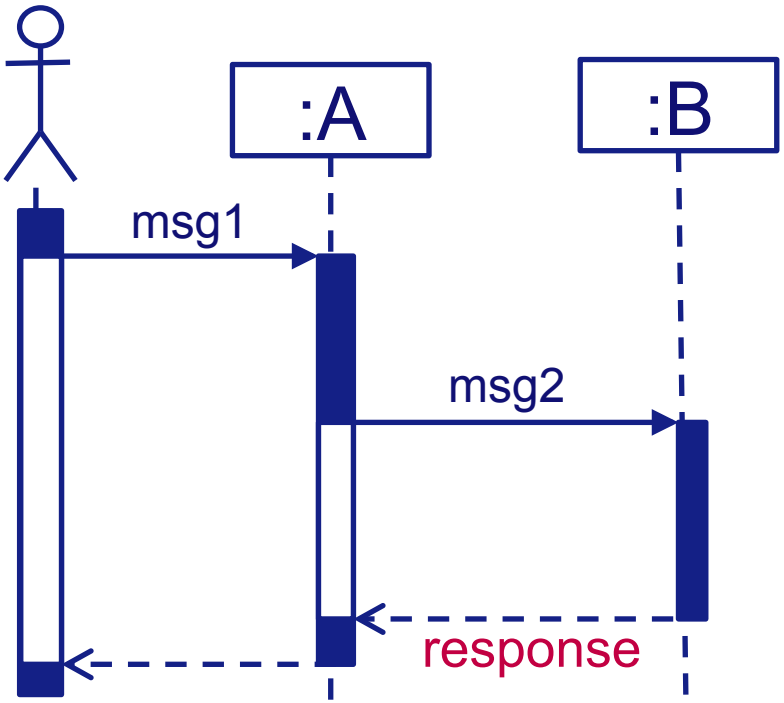
- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines




**:A** sends message **msg2** to **:B** and waits for **:B**'s response: ***synchronous communication***

# Sequence diagrams

- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines

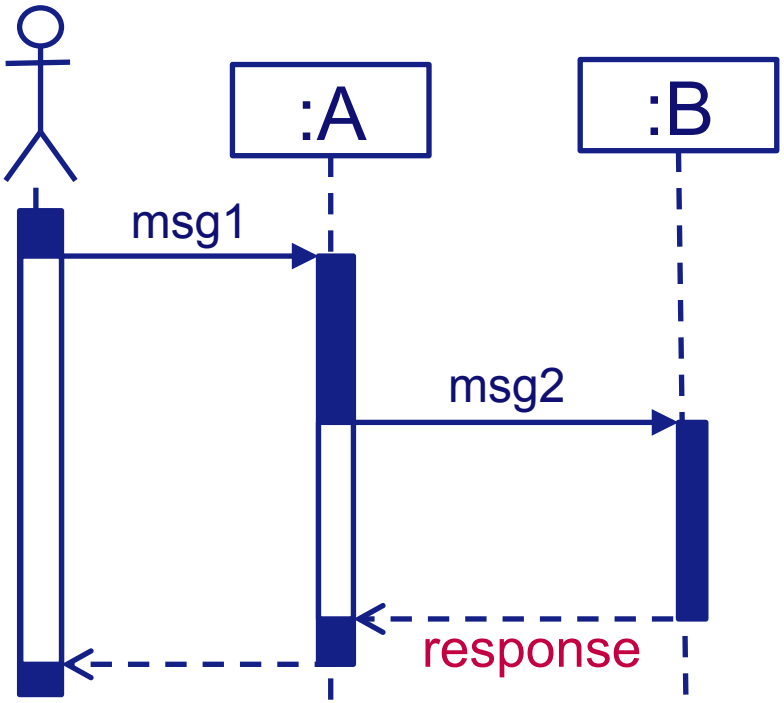



Order of participants = order of the first participation in the interaction

Most of the message sends on arrows from **left to right**

# Sequence diagrams

- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines

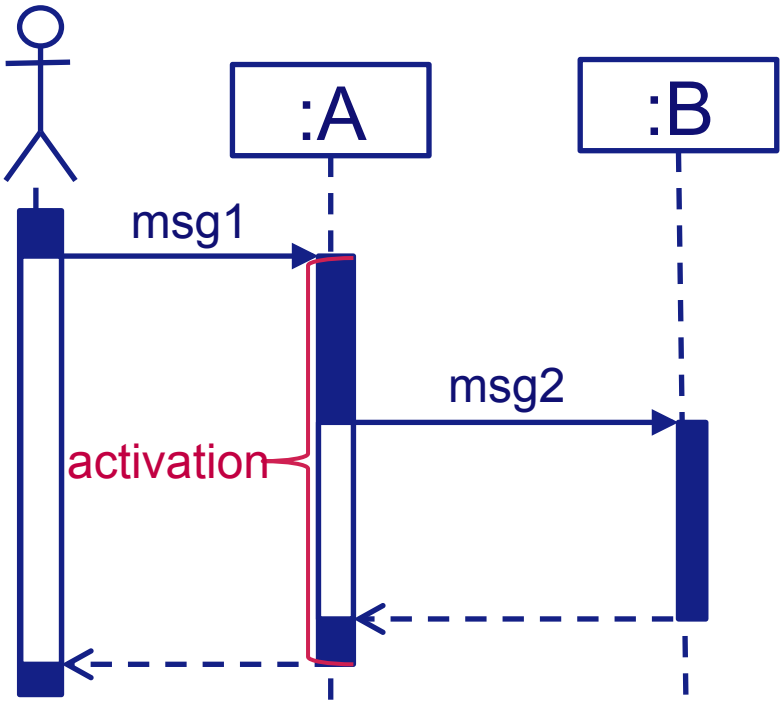



Intuition: Messages are **sent** to objects of classes with the **corresponding operations**.

- Class **A** has method **msg1**
- Class **B** has method **msg2**

# Sequence diagrams

- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines

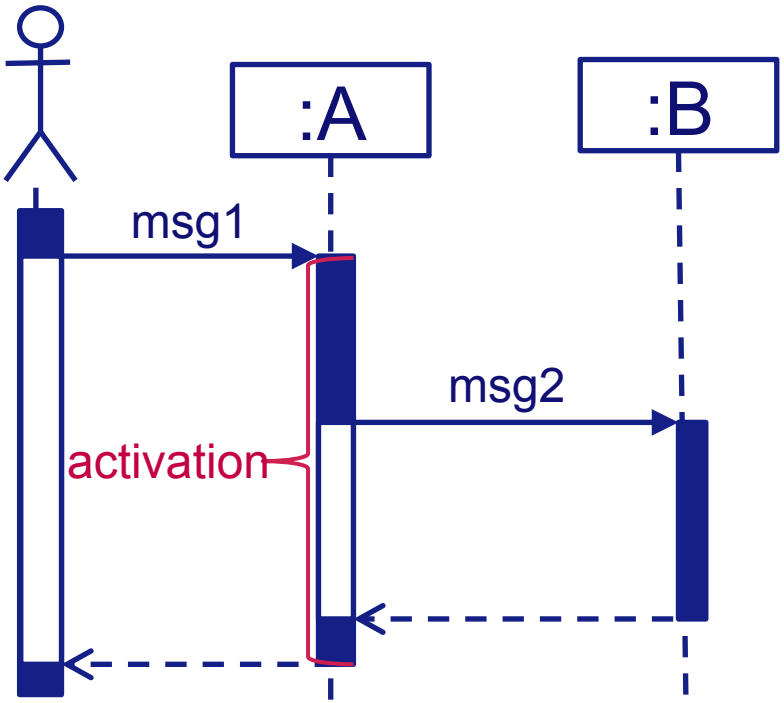



When a message is received by an object, a new **activation** is started.

Activation is also known as **execution specification**.

# Sequence diagrams

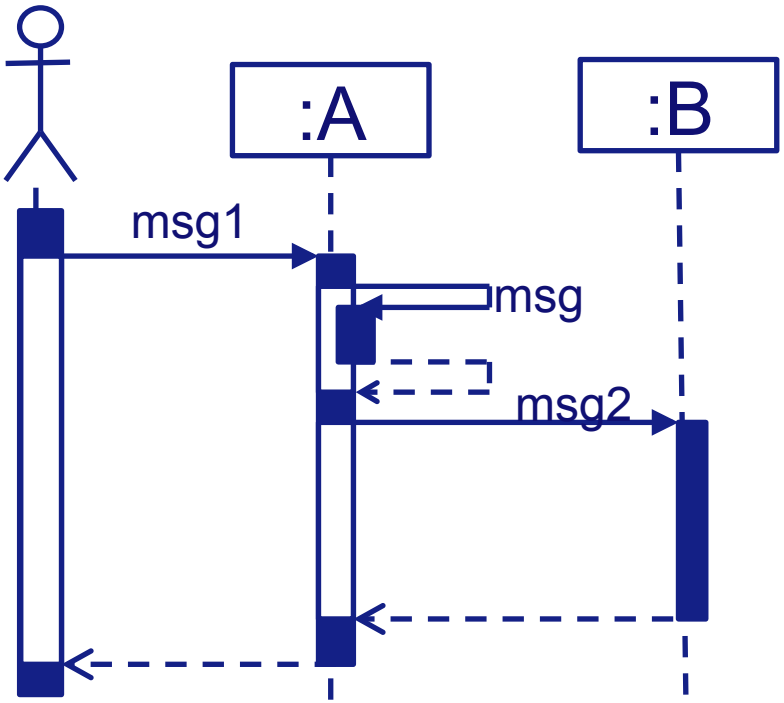
- Can we have multiple activations of the same object?




When a message is received by an object, a new **activation** is started

# Sequence diagrams

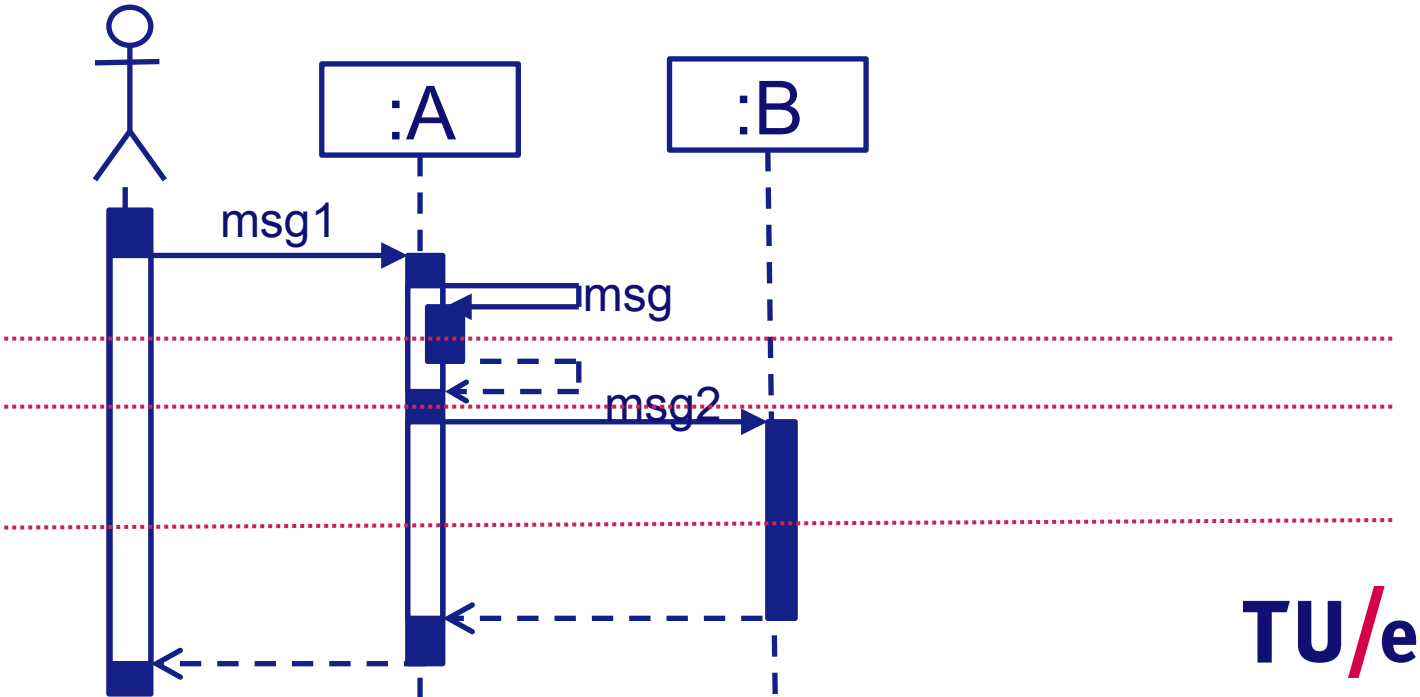
- Can we have multiple activations of the same object?
  - Yes!
  - Activation resulting from message to self is slightly offset from older activation



# Sequence diagrams

If there is no concurrency, **exactly one object** is computing at any given instant

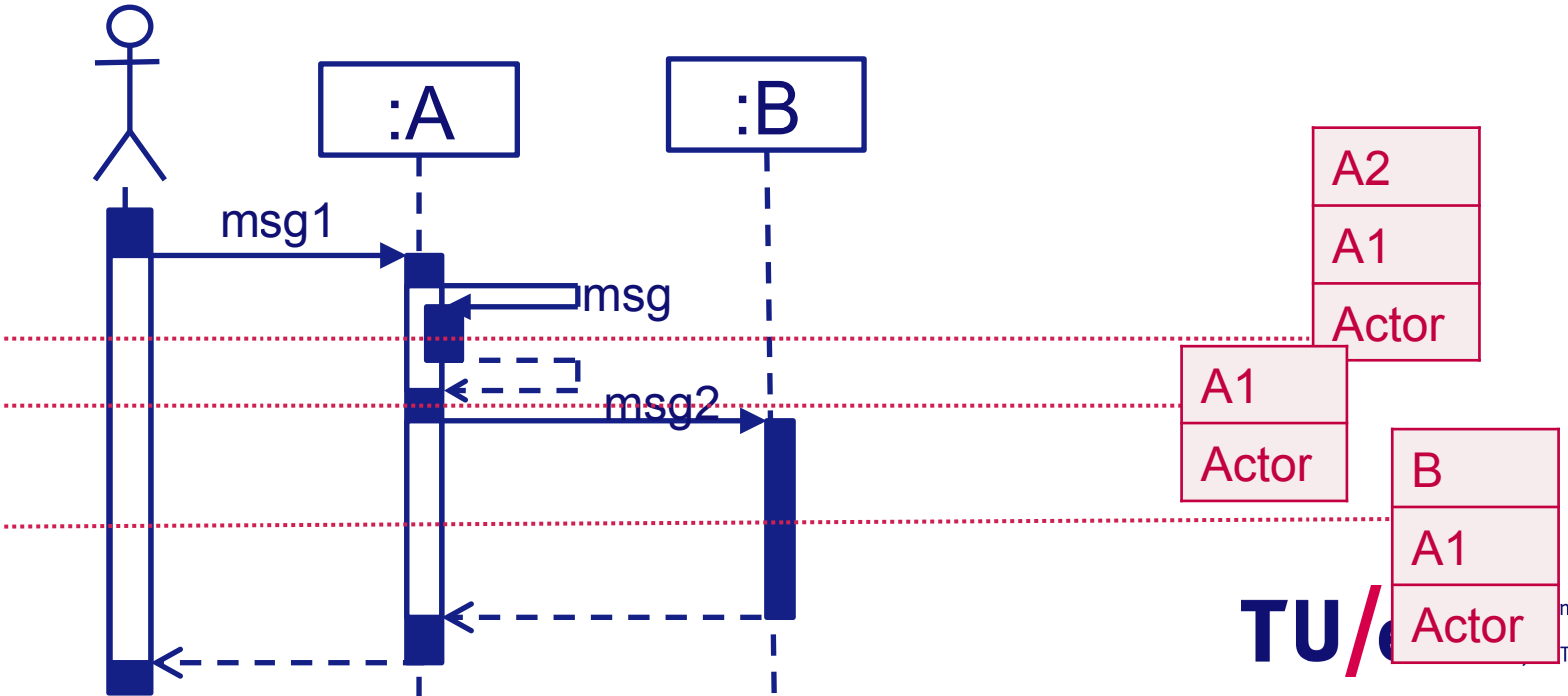
- (Imaginary) horizontal line can cross only one “dark” area



# Sequence diagrams

If there is no concurrency, **exactly one object** is computing at any given instant

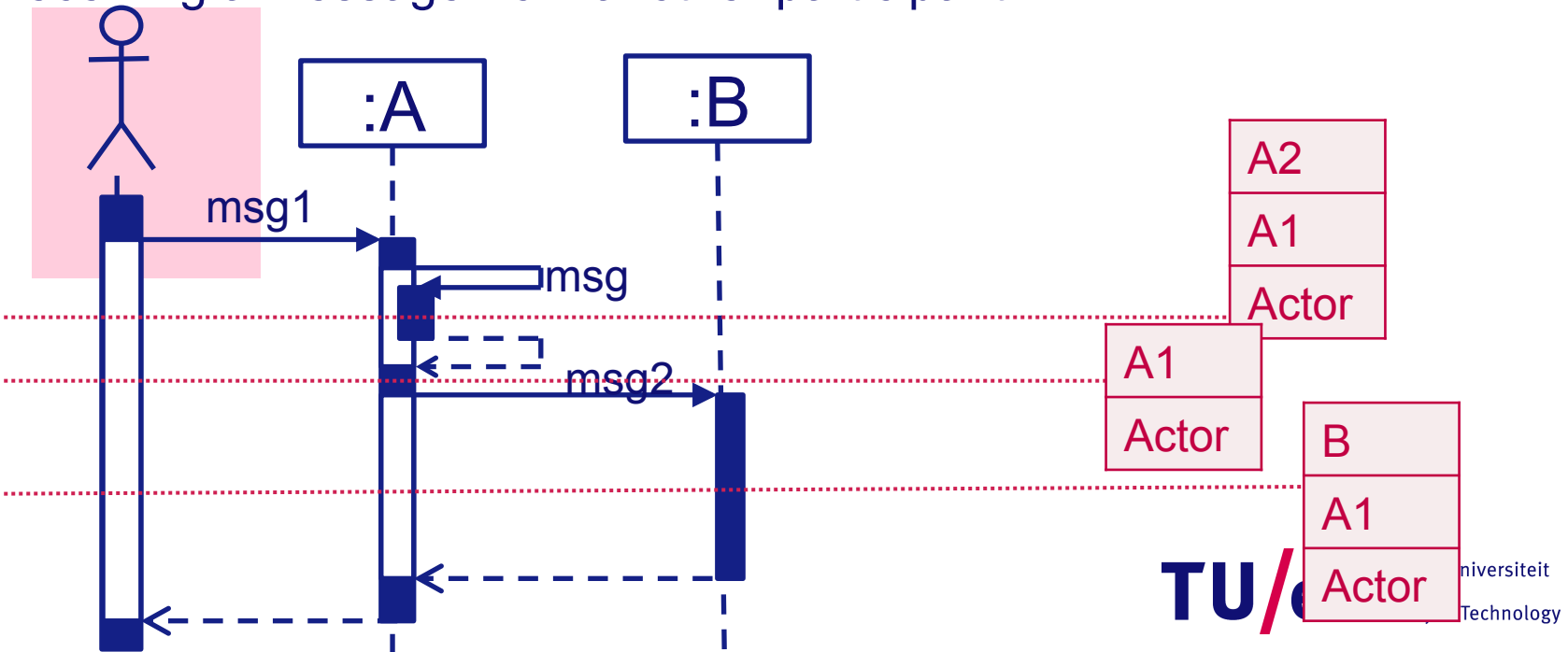
- (Imaginary) horizontal line can cross only one “dark” area
- Activations behave as a stack



# Sequence diagrams

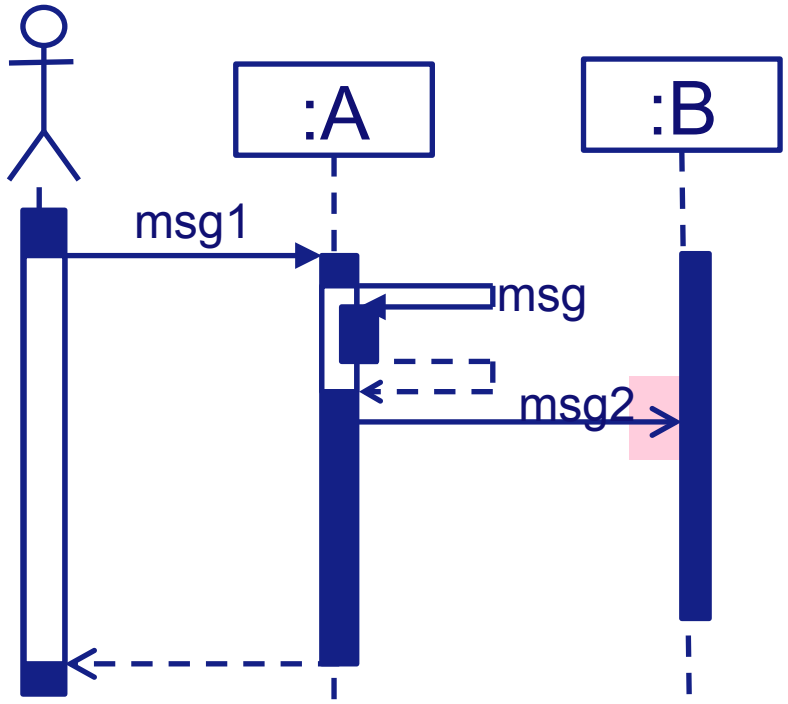
If there is no concurrency, **exactly one object** is computing at any given instant

- (Imaginary) horizontal line can cross only one “dark” area
- Activations behave as a stack
- Only actors can send messages “out of the blue”
  - Objects send messages only when they have been made active by receiving a message from another participant



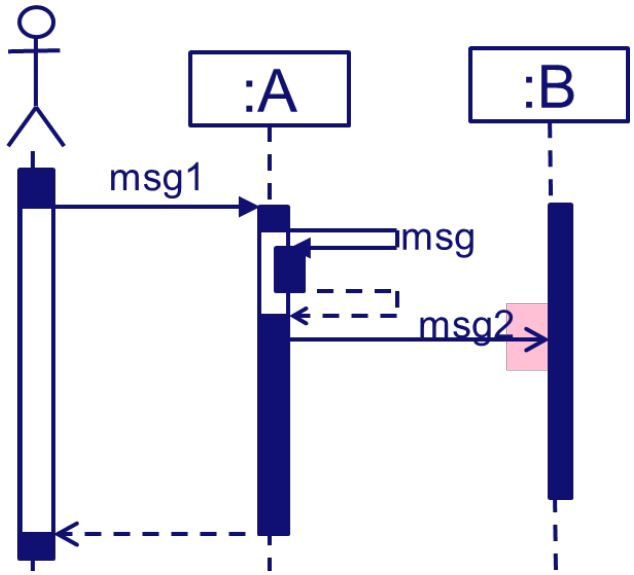
# What if we want concurrency?

Some communication might be asynchronous



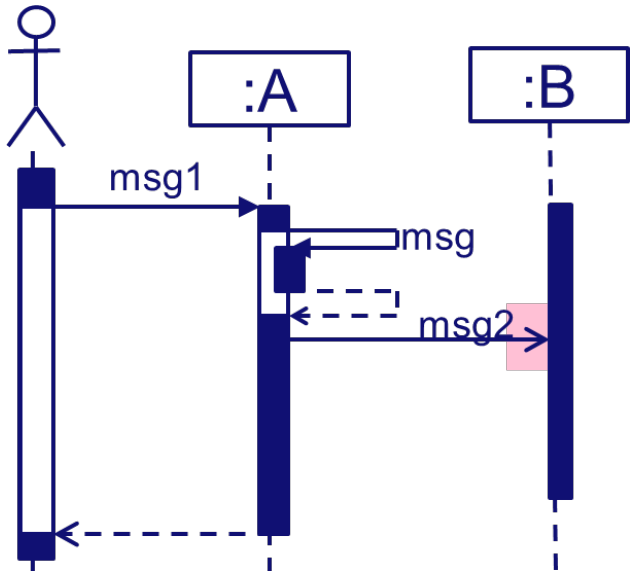
# Question for you

- Messages are partially ordered (before/after)
  - Response to msg should be received before msg2 is sent.
- Why isn't the order is total?



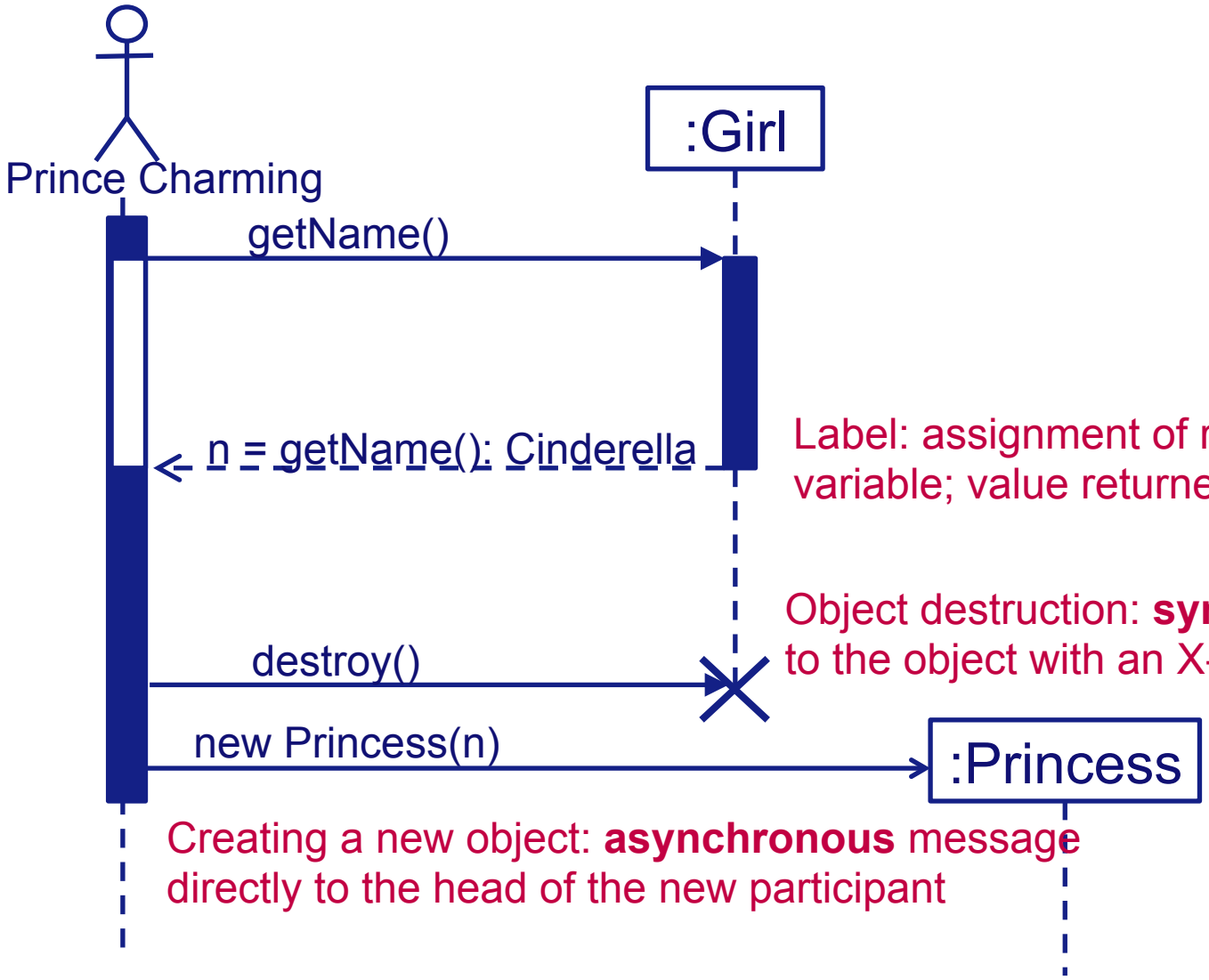
# Question for you

- Messages are partially ordered (before/after)
  - Response to msg should be received before msg2 is sent.




- Why isn't the order is total?
  - independent events on different lifelines are not comparable
    - e.g., concurrent execution resulting from asynchronous messages

# Creating and destroying objects

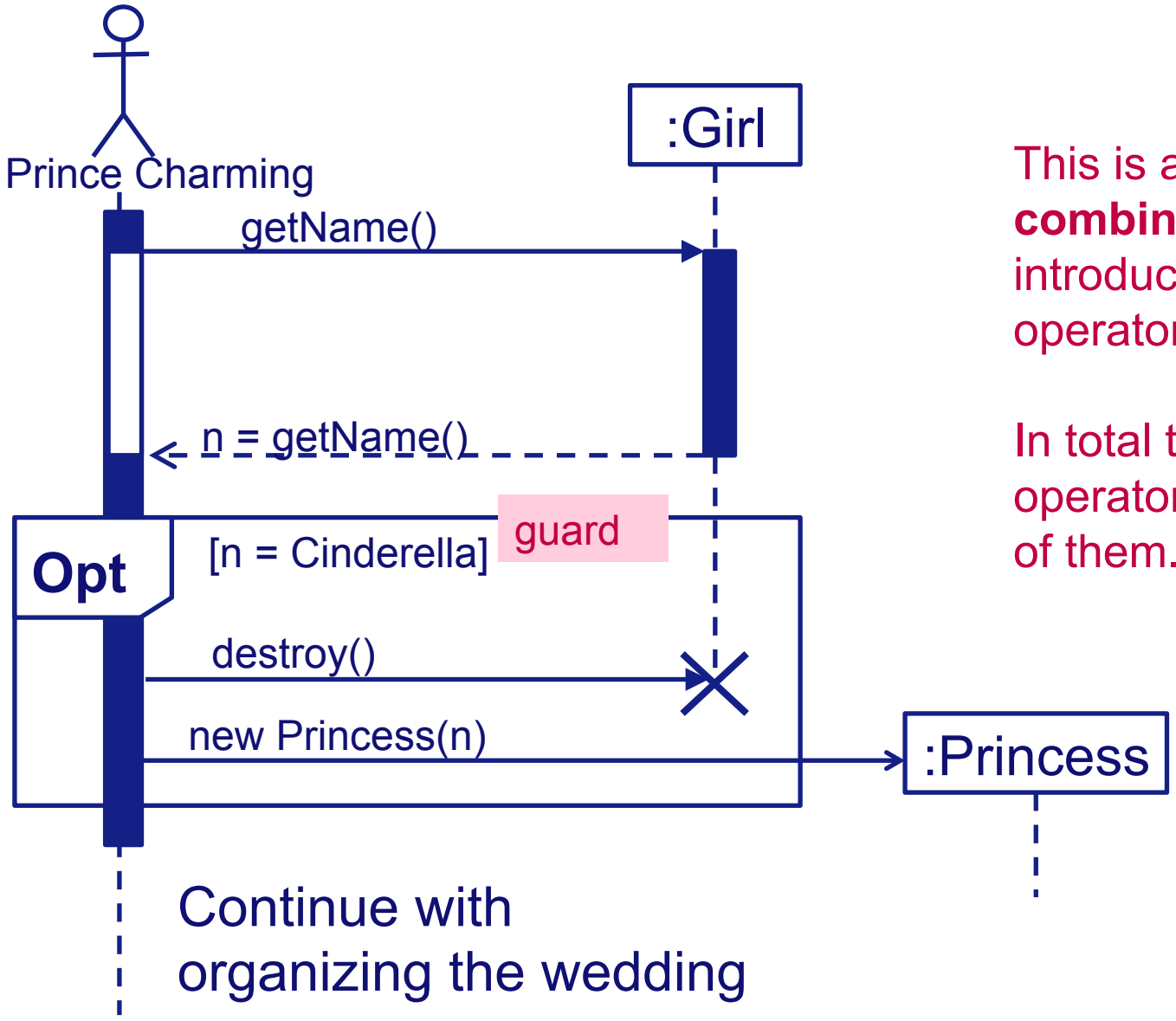



Label: assignment of return value to a variable; value returned shown after colon

Object destruction: **synchronous** message to the object with an X-terminating lifeline

Creating a new object: **asynchronous** message directly to the head of the new participant

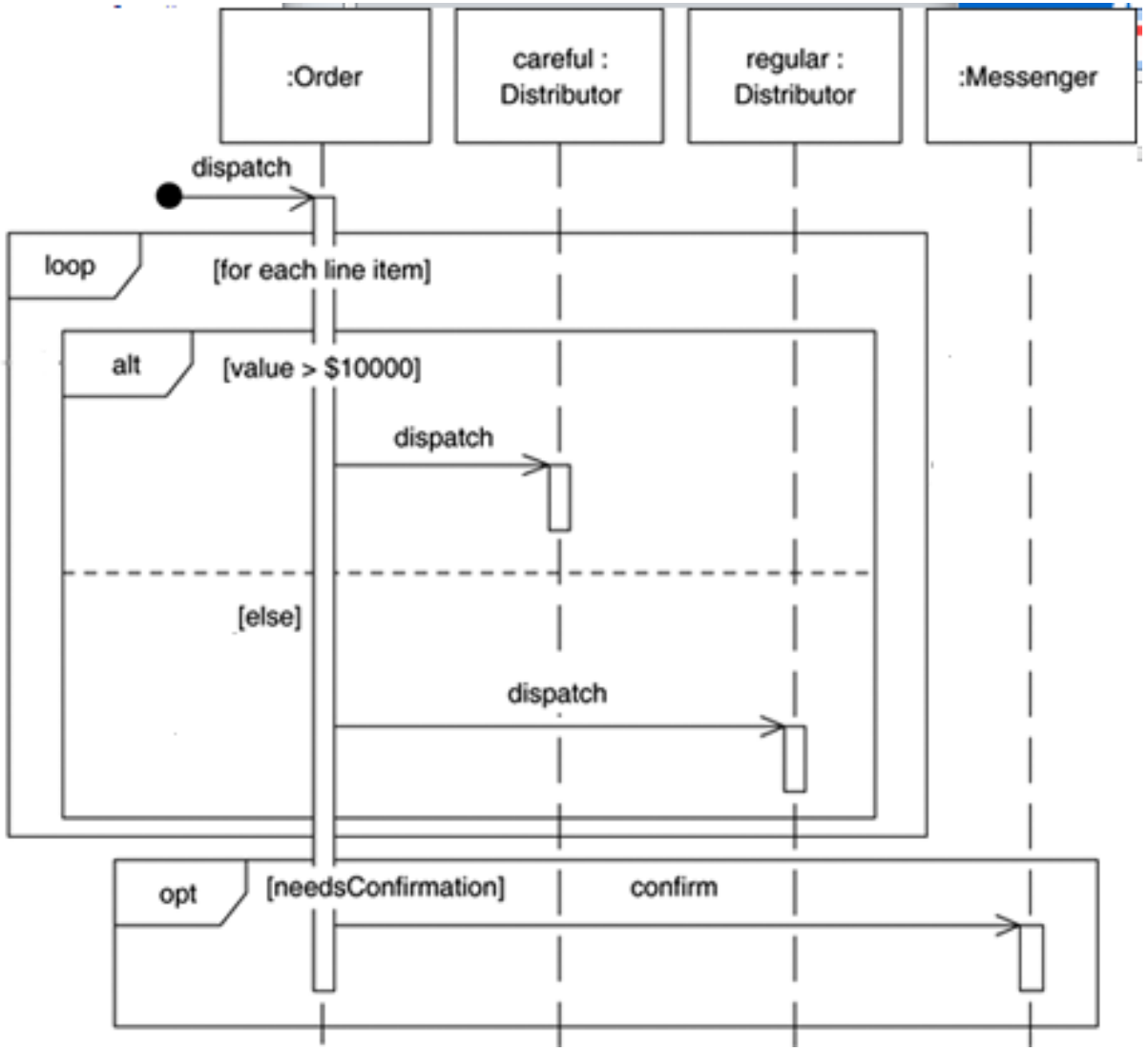
# Optional behavior




This is an example of a **combined fragment** introduced by the operator **Opt** (option).

In total there are 13 operators. We'll see 9 of them.

# Combined fragments: loop and alternatives

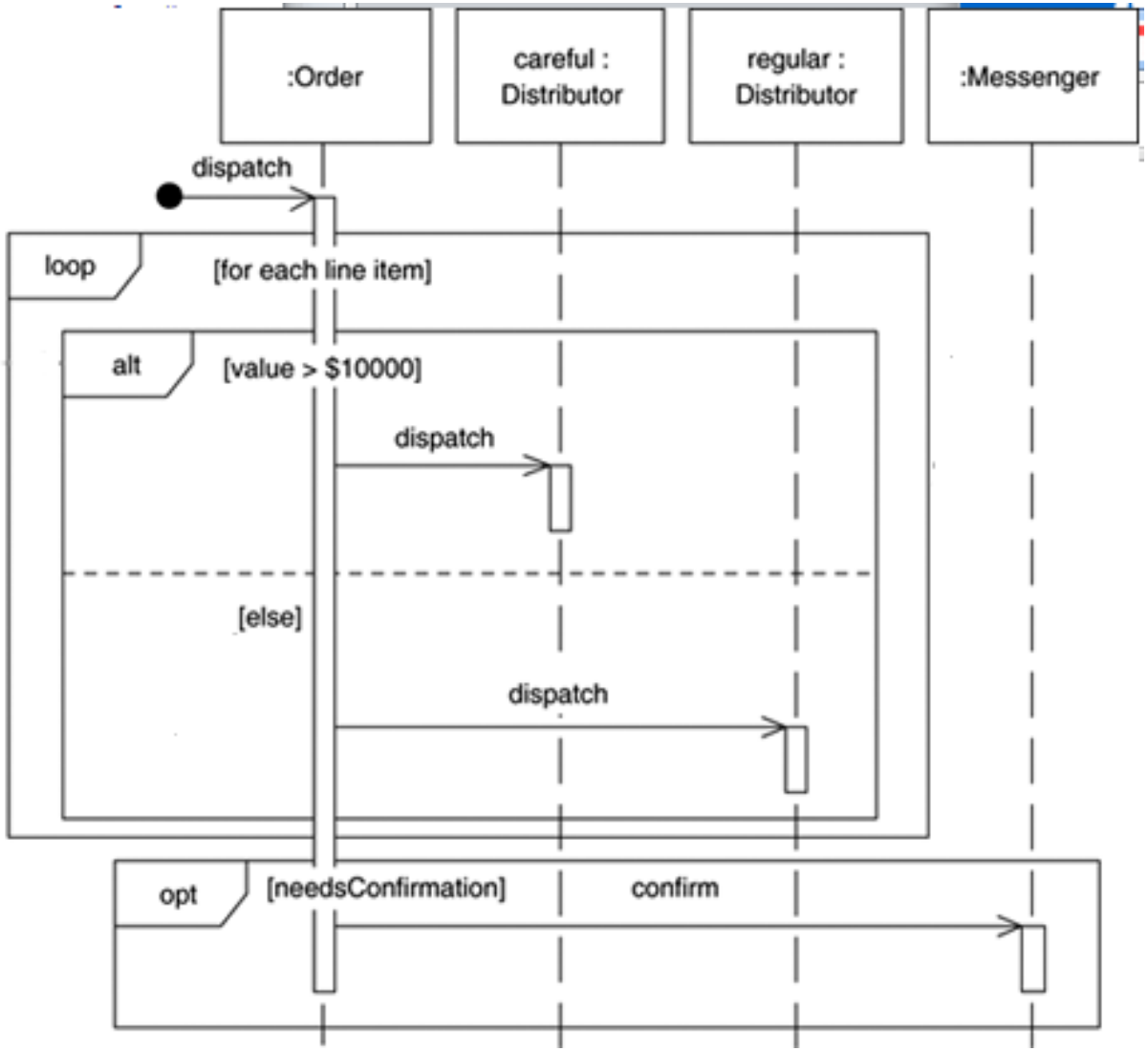



**loop** is potentially infinite.

**loop(10)** indicates 10 iterations.

http://ptgmedia.pearsoncmg.com/images/chap4_0321193687/elementLinks/04fig04.gif

# Combined fragments: loop and alternatives




One can also **break** from the loop.

**break** requires a guard.

**break** quits only one nesting level.

http://ptgmedia.pearsoncmg.com/images/chap4_0321193687/elementLinks/04fig04.gif

# NB: Java/UML mismatches

- Java semantics does not match UML semantics
- UML break
  - if the selected interaction occurs, the enclosing interaction is abandoned
- Java break

**break** quits only one nesting level.

```
for (j = 0; j < arrayOfInts.length; j++) {
   if (arrayOfInts[j] == searchfor) {
      foundIt = true;
      break;
   }
}
```

Rountev, Volgin, Reddoch
PASTE'05

# NB: Java/UML mismatche

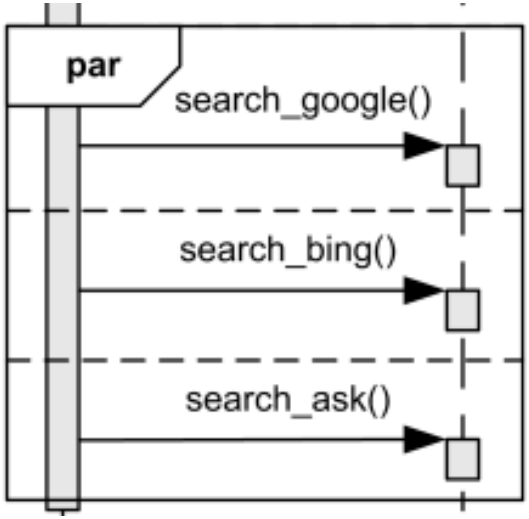
- Java semantics does not match UML semantics
- UML break
  - if the selected interaction occurs, the enclosing interaction is abandoned
- Java break

**break** quits only one nesting level.

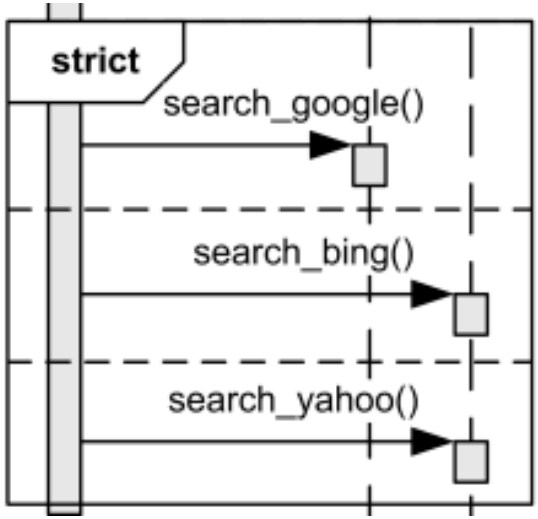
```
search:
  for (i = 0; i < arrayOfInts.length; i++) {
    for (j = 0; j < arrayOfInts[i].length; j++) {
      if (arrayOfInts[i][j] == searchfor) {
        foundIt = true;
        break search;
      }
    }
  }
```

Rountev, Volgin, Reddoch
PASTE'05

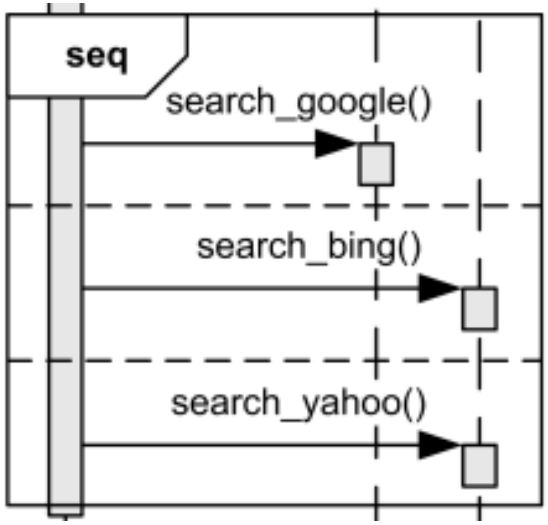
# Combined fragments: par, seq and strict




*Search Google, Bing and Ask in any order, possibly parallel.*




*Search Google, Bing and Yahoo in the strict sequential order.*

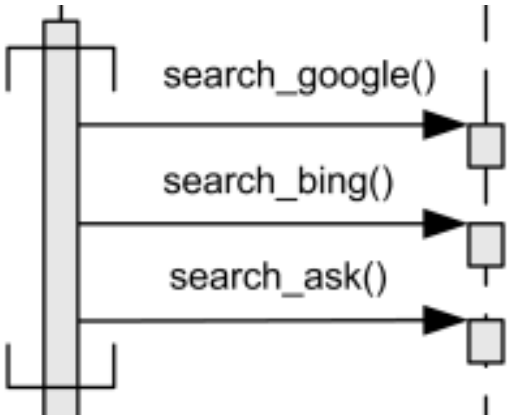


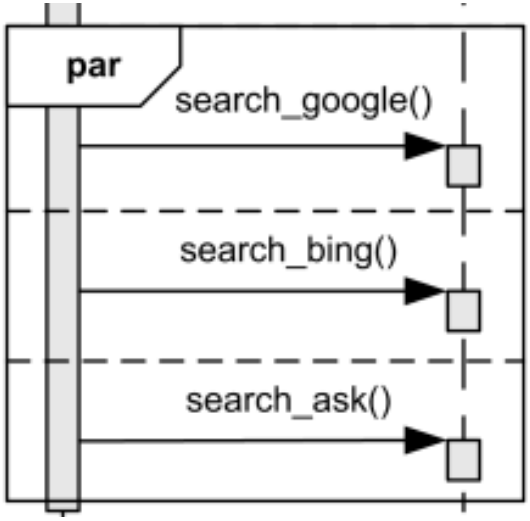

*Search Google possibly parallel with Bing and Yahoo, but search Bing before Yahoo.*

- **seq:**
  - Different lifelines: parallel
  - The same lifeline: sequential

# Par: shorthand

- **Coregion:** par-combined fragment with one lifeline

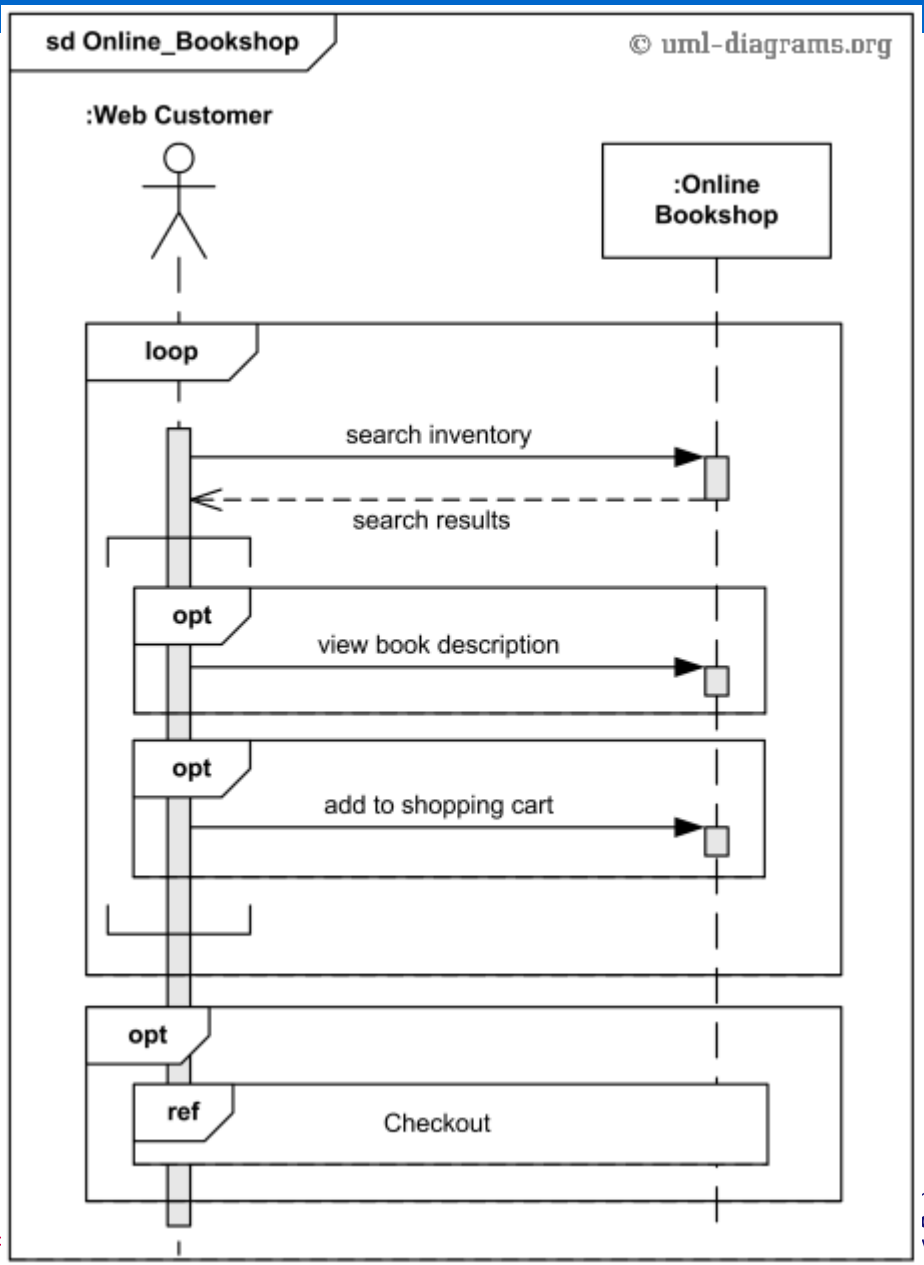



The two fragments are equivalent

# Combining different combined fragments

- **Coregion**
- **Option**
- **Loop**
- **Ref** (interaction use) allows to use (or call) another interaction.




http://www.uml-diagrams.org/examples/sequence-examples-online-bookshop.png

# Interaction use

In general, referencing should be done as

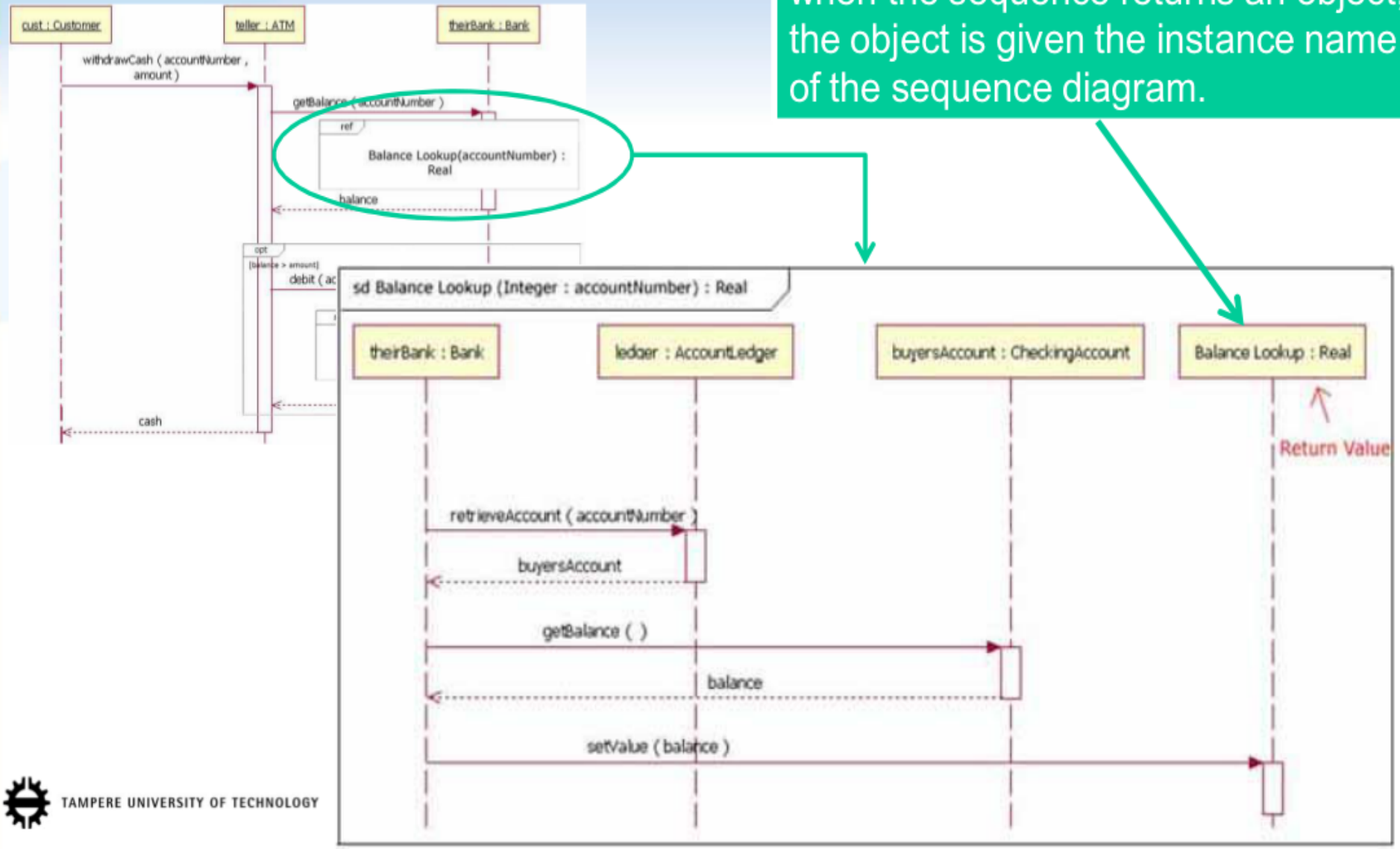
*DiagType DiagName [(ParType:ParName)] : [: RetValType]*

- In practice, types are frequently omitted
  - **Checkout** – no parameters, no return value
  - **Debit Account (accountNumber, amount)** – parameters, no return value
  - **Balance Lookup (accountNumber): Real** – parameters, return value

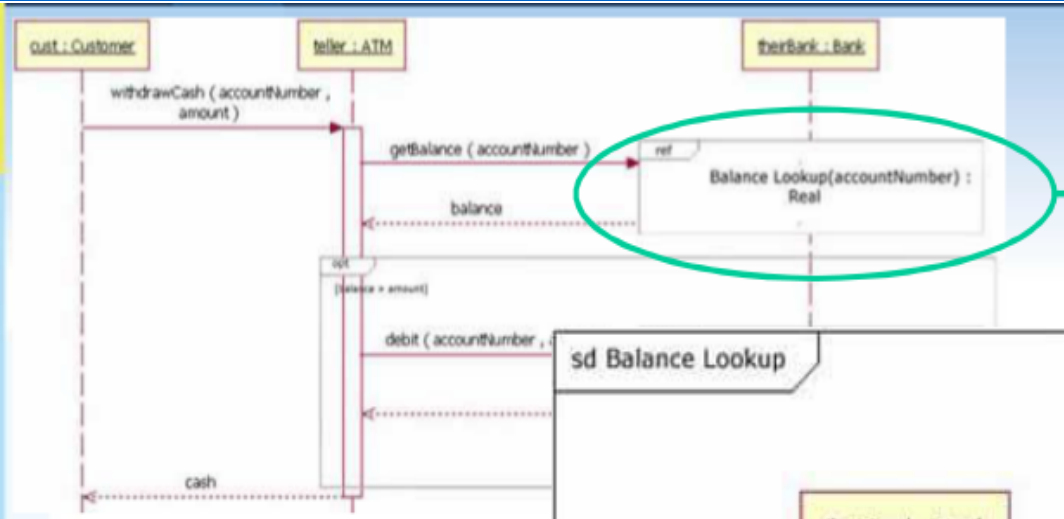
# Interaction use

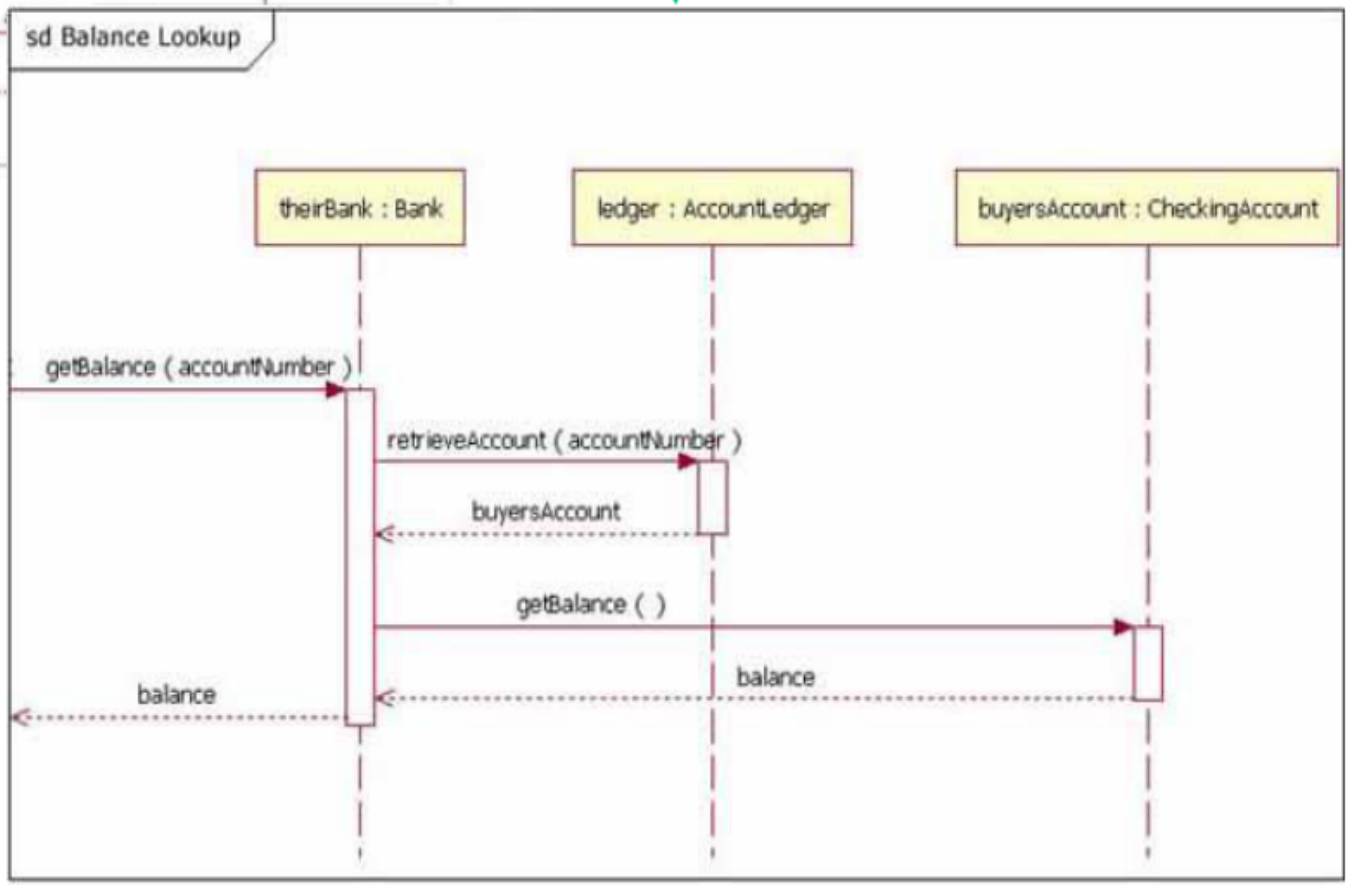
- **Referencing another sequence diagram:**

when the sequence returns an object, the object is given the instance name of the sequence diagram.



# Alternative way to represent a return value




**Gate**: a message with one end connected to the sequence diagram's frame and the other end – to a lifeline.

# Question for you

- How to model **exception handling** in sequence diagram?

- NB: UML provides neither notation to model exception handling in sequence diagrams nor any reasoning why it is absent.

- Still, what would be your solution?

# Question for you

- How to model **exception handling** in sequence diagram?

- NB: UML provides neither notation to model exception handling in sequence diagrams nor any reasoning why it is absent.

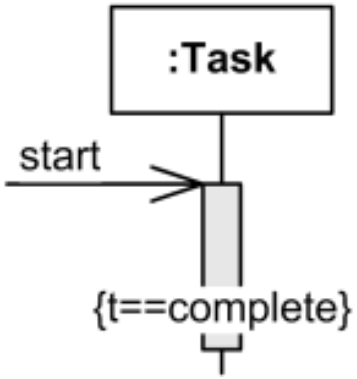
- Still, what would be your solution?

Combined fragments alt/break

# What about constraints?

- **Constraint**: boolean restriction on the participants of the interaction, usually verified at run time.

## State Invariant

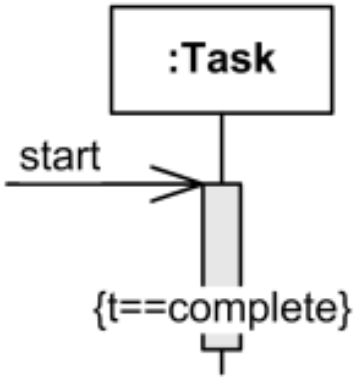



Attribute t of Task should have value “complete”

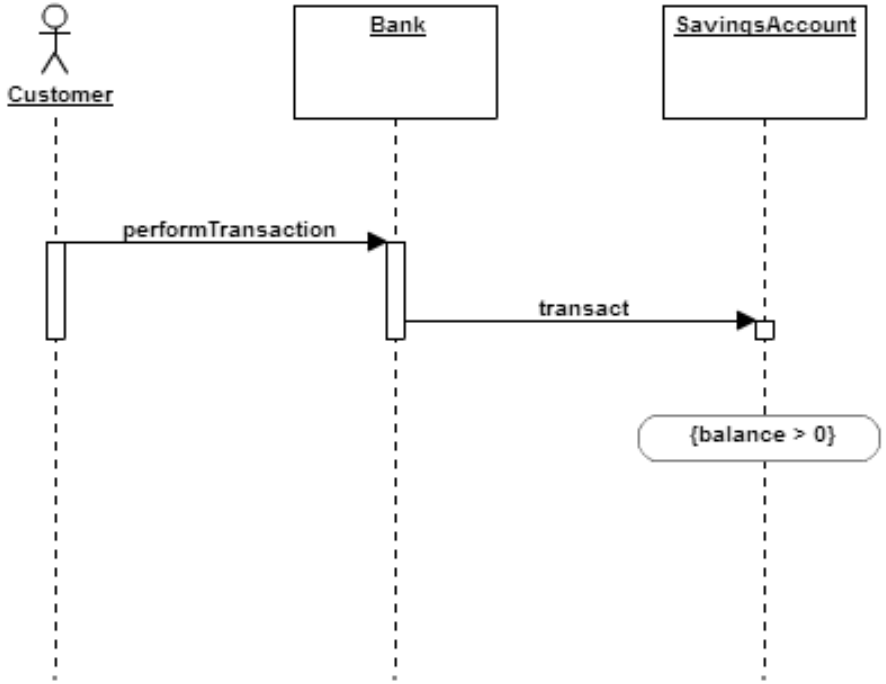
# What about constraints?

- **Constraint**: boolean restriction on the participants of the interaction, usually verified at run time.

## State Invariant




Attribute t of Task should have value “complete”

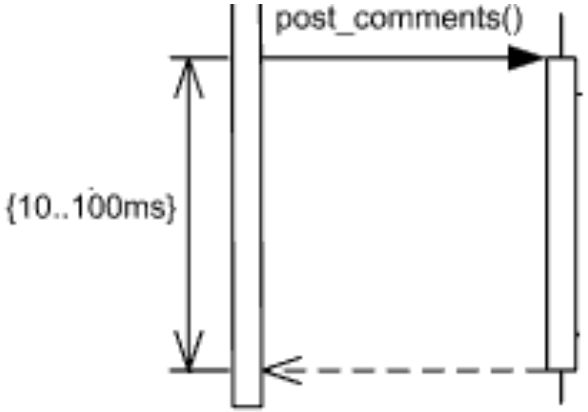



A transaction on SavingsAccount is permitted only if the balance remains positive.

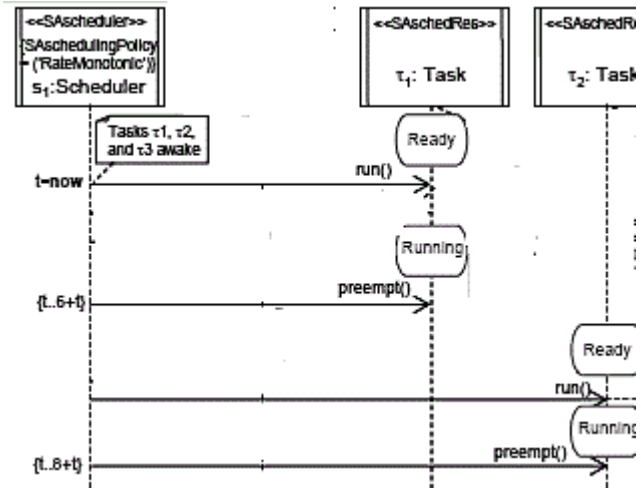
# What about constraints?

- **Constraint**: boolean restriction on the participants of the interaction, usually verified at run time.

## Timing constraints




Response time constraint

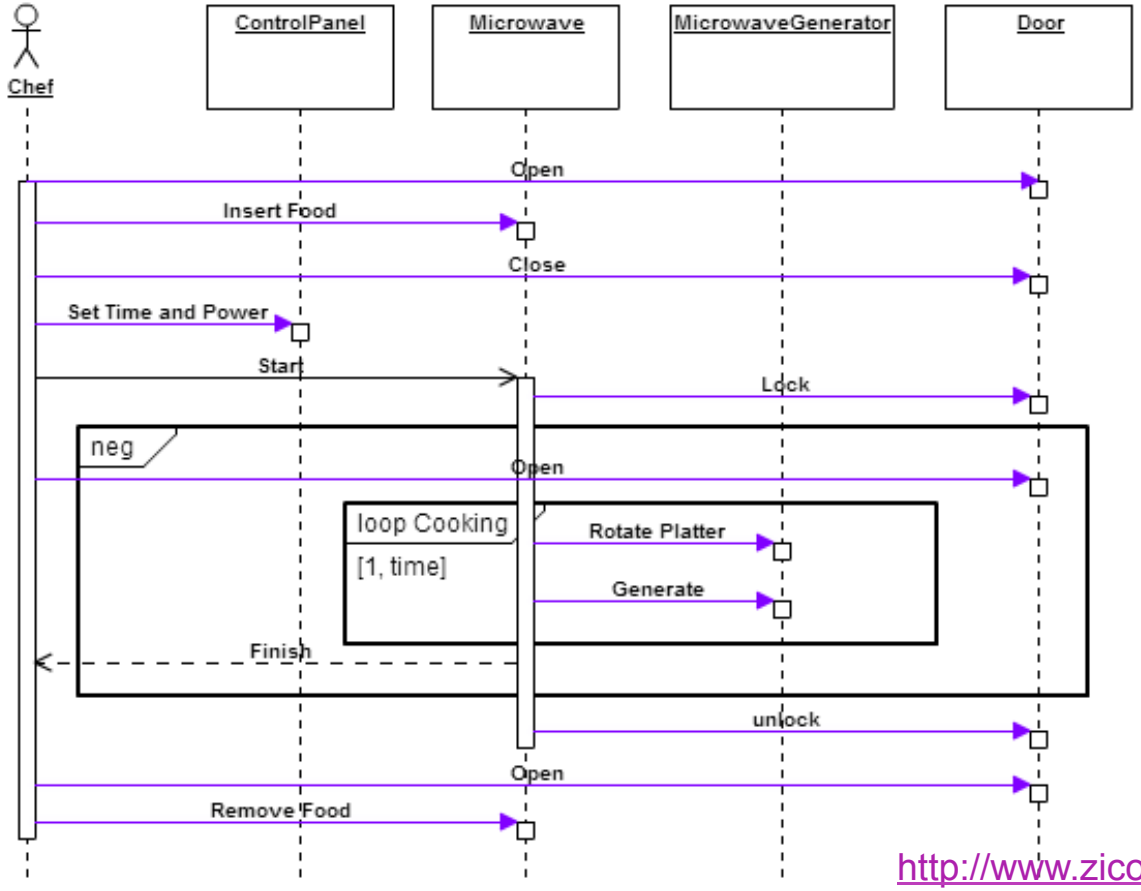



Scheduling constraint

# What about constraints?

- **Constraint**: boolean restriction on the participants of the interaction, usually verified at run time.

## Prohibition: **neg** combined fragment




One **cannot** open the microwave door when it is cooking.

Also notice the “from 1 to time” loop.

http://www.zicomi.com/combinedFragmentNegative.jsp

# Summary of the combined fragments

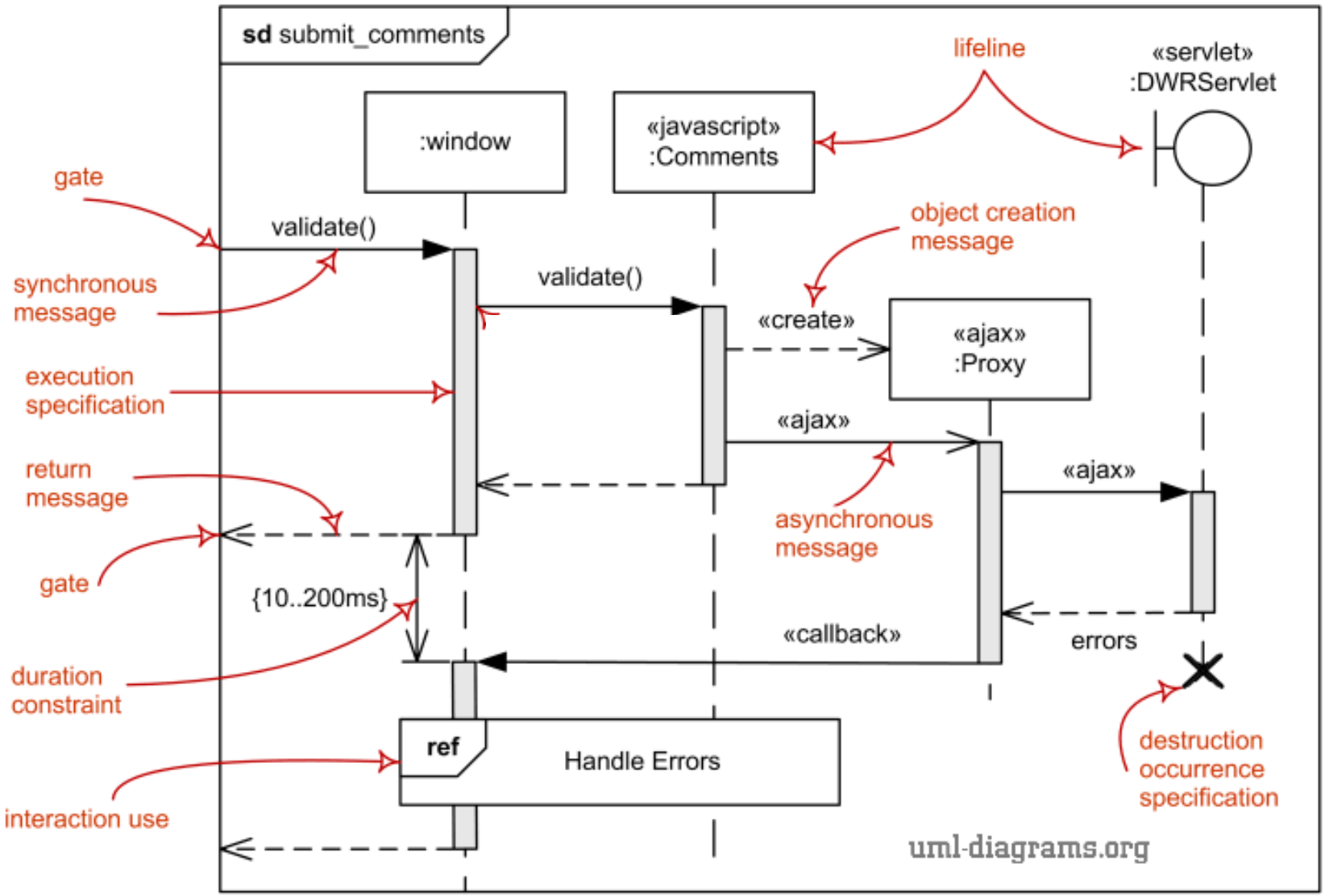
Interaction operator of a combined fragment could be one of:

- alt – alternatives
- opt – option
- loop – iteration
- break – break
- par – parallel
- strict – strict sequencing
- seq – weak sequencing
- *critical – critical region*
- *ignore – ignore*
- *consider – consider*
- *assert – assertion*
- neg – negative
- ref – interaction use

4 operators in *italics* have not been discussed.

You are welcome to read about them in OMG UML version 2.5, Section 17.6.3, pp. 609-611.

# Summary of the syntax




http://www.uml-diagrams.org/sequence-diagrams.html

# Recall…

- You draw a sequence diagram if, e.g.:
  - you have a use case diagram, to describe how the main components of the system interact
  - you have identified messages arriving at an interface of a component, to describe how the internal parts of the component interact.

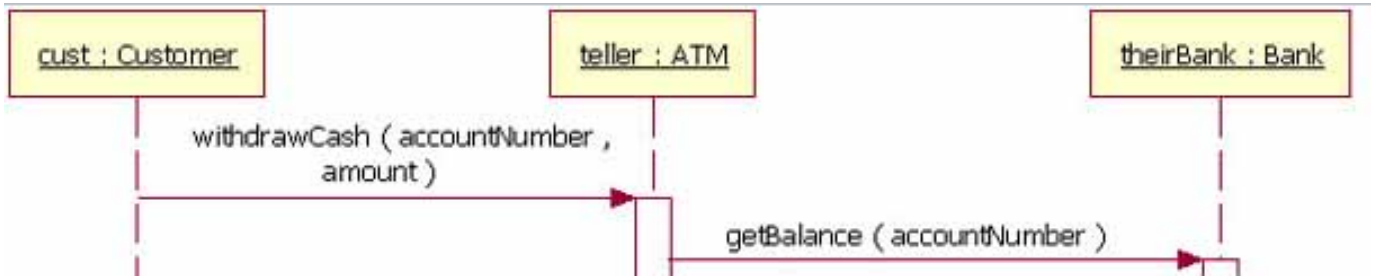
# Building sequence diagrams

- Set the context
- Identify participants that should interact to realize the use-case scenario / to respond to messages arriving at the interface.
  - Place the initiator of the interaction on the left of the diagram
  - Multiple objects of the same class: name each
- Set the lifeline for each participant
- Add messages by drawing arrows
  - Order messages from left to right, top to bottom
  - Show how they are passed from one object to another
  - Include any parameters in parentheses
  - Exclude obvious return values
- Add activations to each participant's lifeline
- Validate the sequence diagram

# Validation

- **Recall:** Interaction diagrams should be consistent with the corresponding class diagrams and use case diagrams

- **Rule:** Objects in [sd] should be instances of classes in [cd]
- **Rule**: Name of the message [sd] should match an operation in the receiver's class [cd]



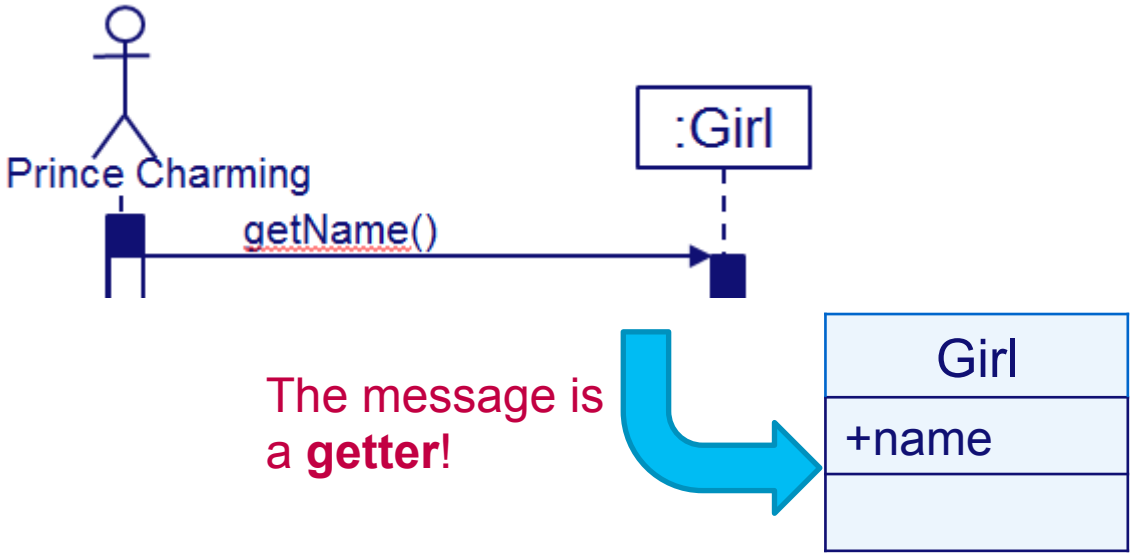


http://www.ibm.com/developerworks/rational/library/3101.html

# Validation

- **Recall:** Interaction diagrams should be consistent with the corresponding class diagrams and use case diagrams

- **Rule:** Objects in [sd] should be instances of classes in [cd]
- **Rule**: Name of the message [sd] should match an operation in the receiver's class [cd]



# Validation

- **Recall:** Interaction diagrams should be consistent with the corresponding class diagrams and use case diagrams

- **Rule:** Objects in [sd] should be instances of classes in [cd]
- **Rule**: Name of the message [sd] should match an operation in the receiver's class [cd]
- **Rule**: If a message is sent from A to B [sd] then there should be an association from the class of A to the class of B [cd]
- **Rule**: If use-case A generalizes use-case B [ucd] then the sequence diagram of A should be a subgraph of the sequence diagram of B [sd].

# Validation

- **Recall:** Interaction diagrams should be consistent with the corresponding class diagrams and use case diagrams

- Briand *et al.* mention 120 consistency rules
  - http://squall.sce.carleton.ca/pubs/tech_report/TR_SCE-03-01.pdf

- Active research domain around 2005
  - Prototype implementations
  - Inconsistency *detection* vs. inconsistency *resolution*

# Group assignment 2: Consistency!

- Are the sequence diagrams consistent with the class diagram?
- Are the state machines consistent with the class diagram?
- Are there any inconsistencies between the behavior described by the behavioral diagrams and the requirements?
- Are the declarations consistent with the classes introduced in the class diagram?

# Sequence diagrams as a specification technique?

*Unambiguous?*

*Realistic?*

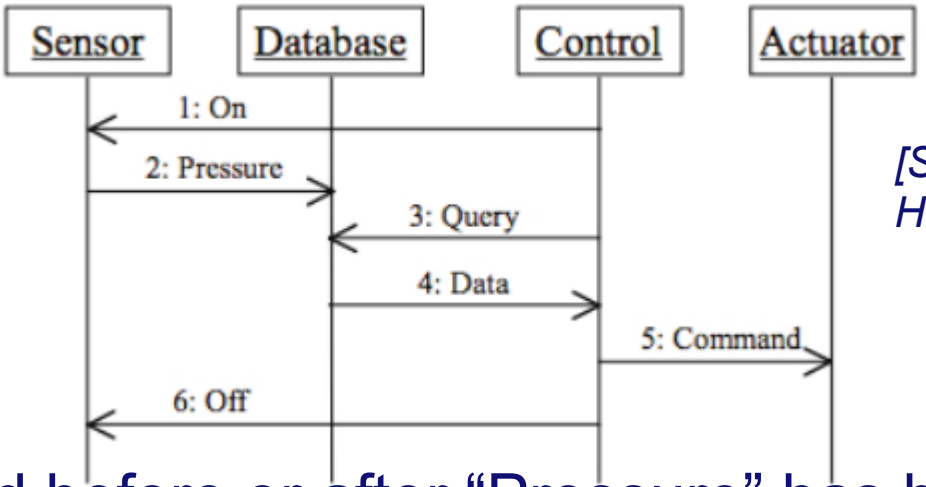
*Verifiable?*

*Evolvable?*

# Sequence diagrams as a specification technique?

*Unambiguous?*

- more advanced features [seq, strict, …] are less clear (and less popular) and might lead to ambiguity
- asynchronous communication is portrayed as if it was simultaneous while it is not




*[Sibertin-Blanc, Hameurlain, Tahir 2008]*

is “Query” send before or after “Pressure” has been received? before or after “Pressure” has been send?

# Sequence diagrams as a specification technique?

*Realistic?*

- allows to model interaction, popular in the industry

*Verifiable?*

- interactions can be replayed through testing
- omissions and inconsistencies are hard to spot
- [Guerra, de Lara 2003] propose a formal verification approach based on Petri nets, but only for a restricted subclass of sequence diagrams

*Evolvable?*

- depends on the complexity/size
- affected by ambiguity

# Interaction diagrams

Sequence Diagram ✔

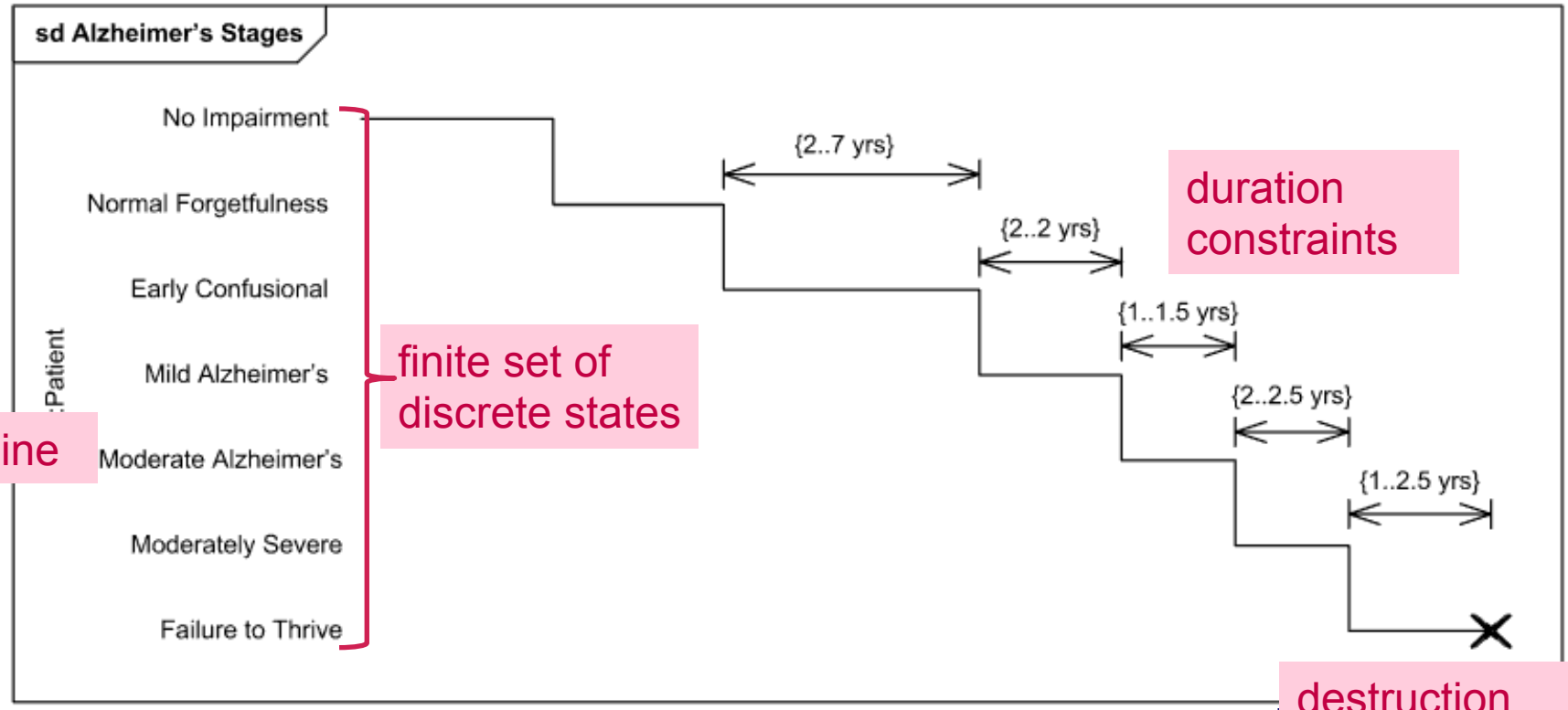
Communication Diagram

Interaction Overview Diagram

Timing Diagrams TODAY

# Timing diagram

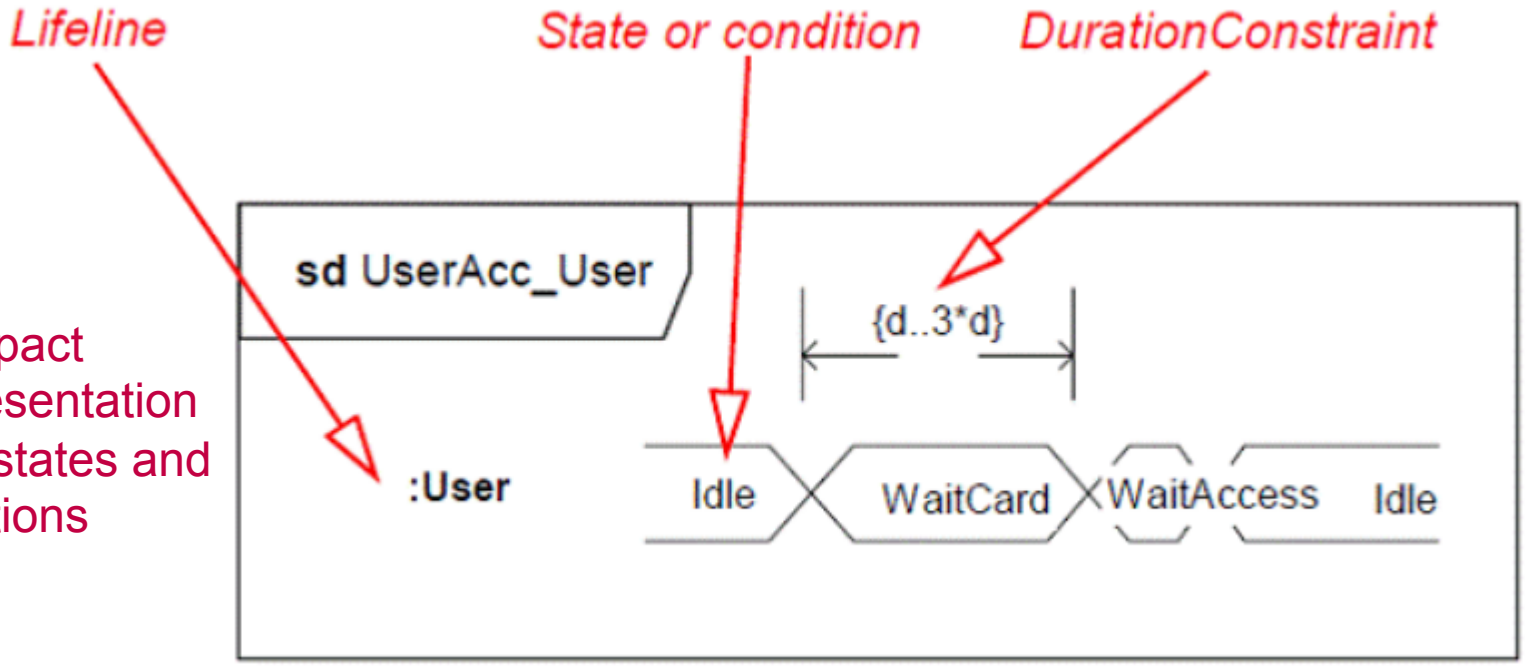
- Interaction diagram for reasoning about time
- **Basic elements**: lifelines, states, duration/time constraints, destruction, events, messages




http://www.uml-diagrams.org/timing-diagrams-examples.html#alzheimers

# Timing diagram

- Interaction diagram for reasoning about time
- **Basic elements**: lifelines, states, duration/time constraints, destruction, events, messages

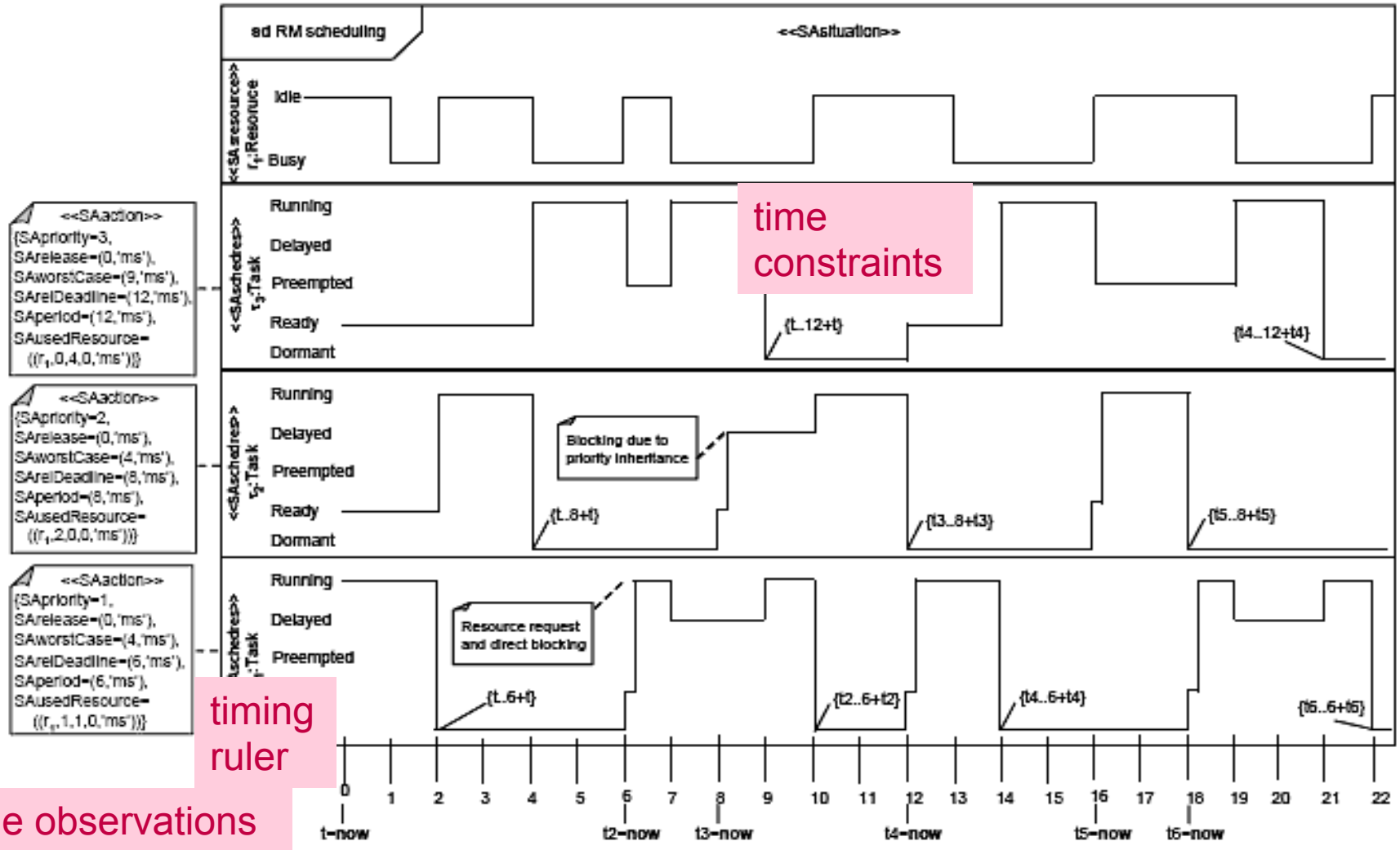



Compact representation with states and durations

# Timing diagram

- Interaction diagram for reasoning about time

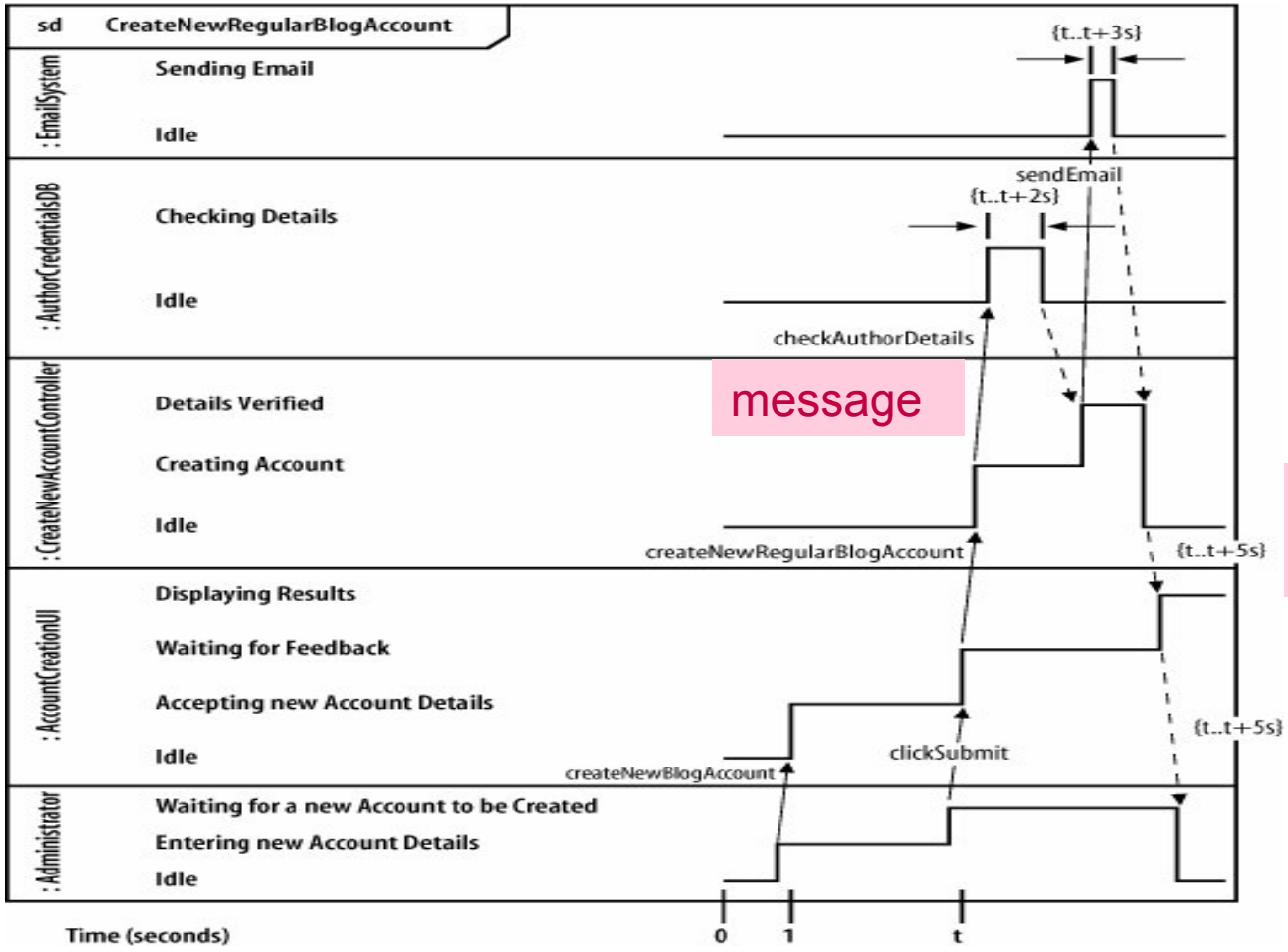
http://www.jot.fm/issues/issue_2006_05/article2/



# Timing diagram

- Interaction diagram for reasoning about time

http://creately.com/blog/diagrams/uml-diagram-types-examples/#TimingDiagram



# Questions

Sketch a timing diagram for the following statements:

a) Ice should melt into water in 1 to 6 minutes.

b) Person should wake up between 5:40 am and 6 am.

c) Lifecycle of a virus: Dormant, Propagation, Triggering and Execution.

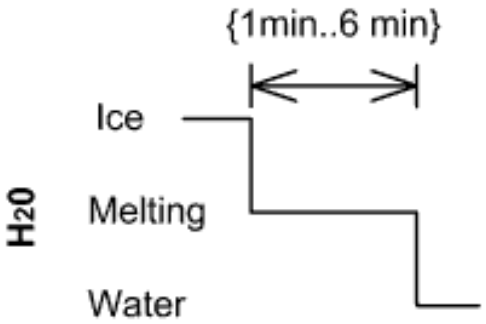
# Questions

Sketch a timing diagram for the following statements:

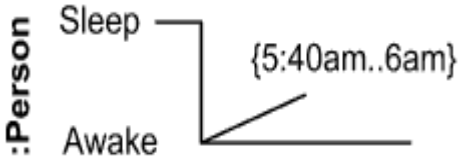
a) Ice should melt into water in 1 to 6 minutes.

b) Person should wake up between 5:40 am and 6 am.

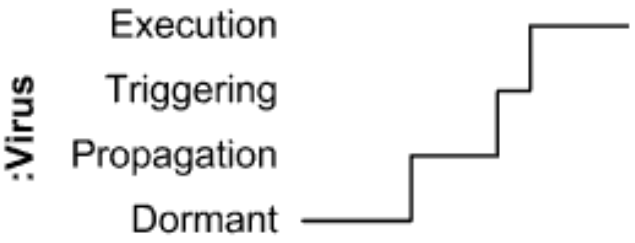
c) Lifecycle of a virus: Dormant, Propagation, Triggering and Execution.




a)




b)




c)

http://www.uml-diagrams.org/timing-diagrams.html

# Timing diagrams as a specification technique?

*Unambiguous?*

*Realistic?*

*Verifiable?*

*Evolvable?*

# Timing diagrams as a specification technique?

*Unambiguous?*

- ranges on messages: do they refer to message being send or being received?

*Realistic?*

- familiar to hardware engineers
- relatively unpopular *[Reggio et al. MODELS 2013]*
  - have a restricted scope
  - alternatives: timed events in state machines and activity diagrams; durations and time intervals may appear in sequence diagrams

# Timing diagrams as a specification technique?

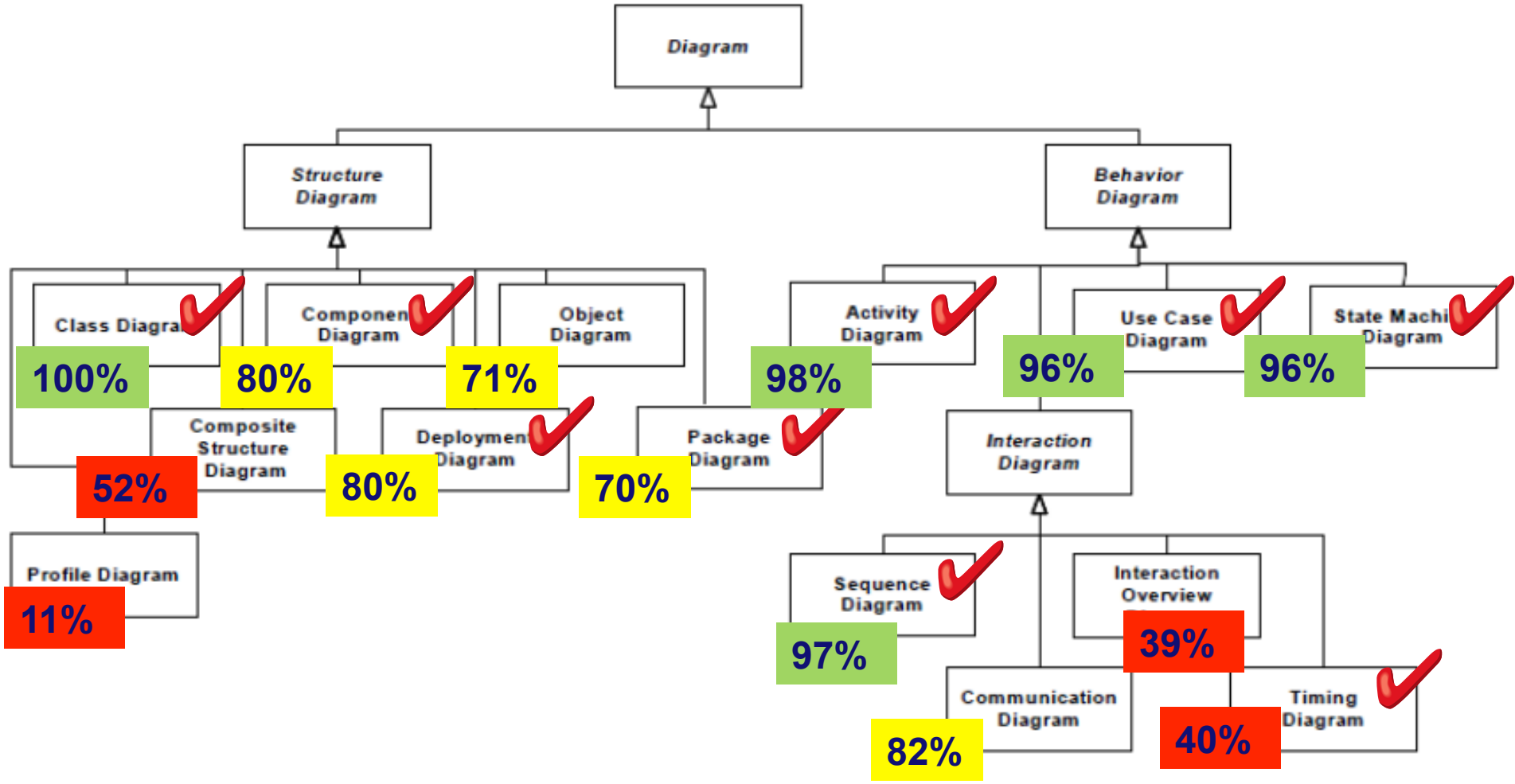
*Verifiable?*

- Cui et al. (2010) convert timing diagrams to timed automata and verify using UPPAAL

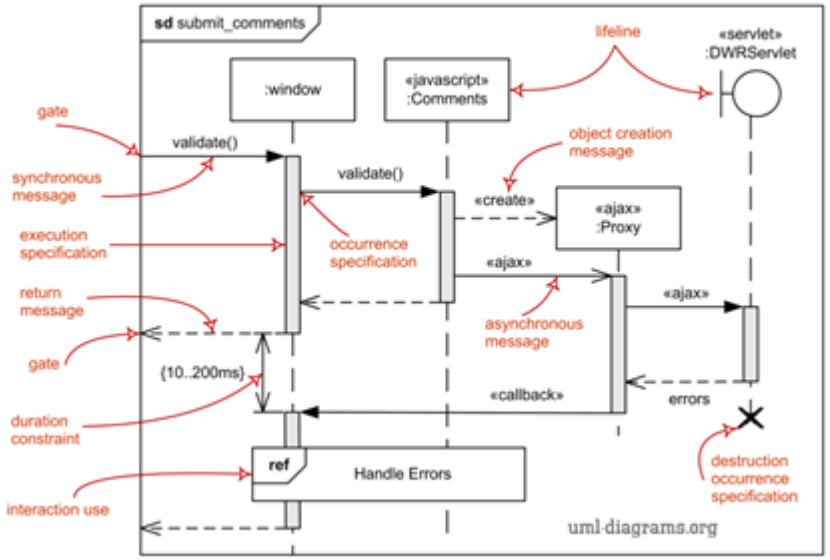
*Evolvable?*

- depends on the complexity/size
- can be affected by lack of popularity

# UML diagram types: popularity



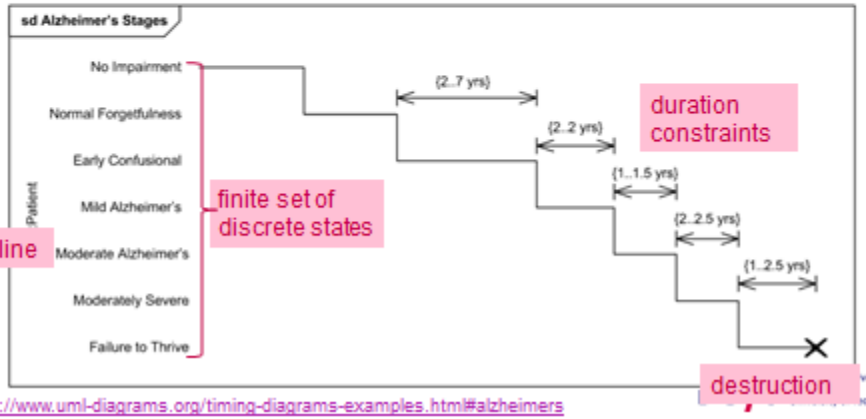
## Summary of the syntax




http://www.uml-diagrams.org/sequence-diagrams.html

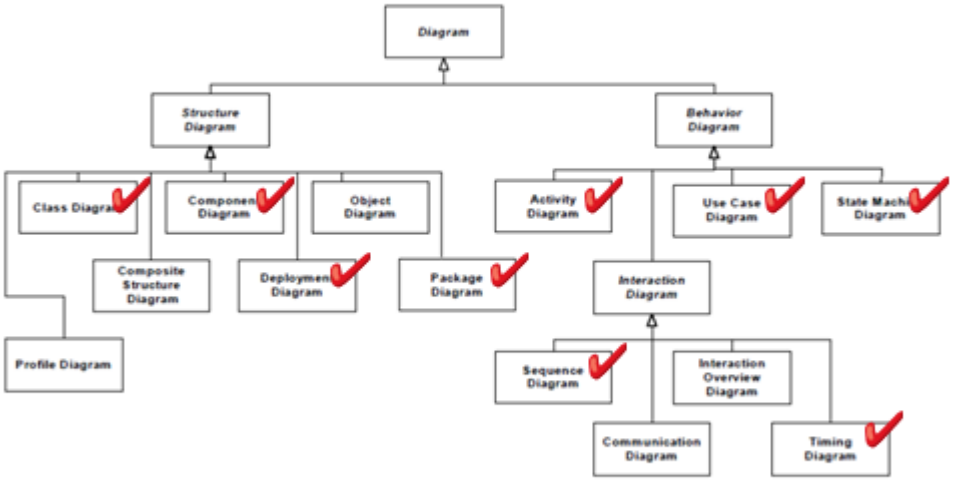
## Timing diagram

- Interaction diagram for reasoning about time
- **Basic elements**: lifelines, states, duration/time constraints, destruction, events, messages




http://www.uml-diagrams.org/timing-diagrams-examples.html#alzheimers

## UML diagram types






/ SET / W&I