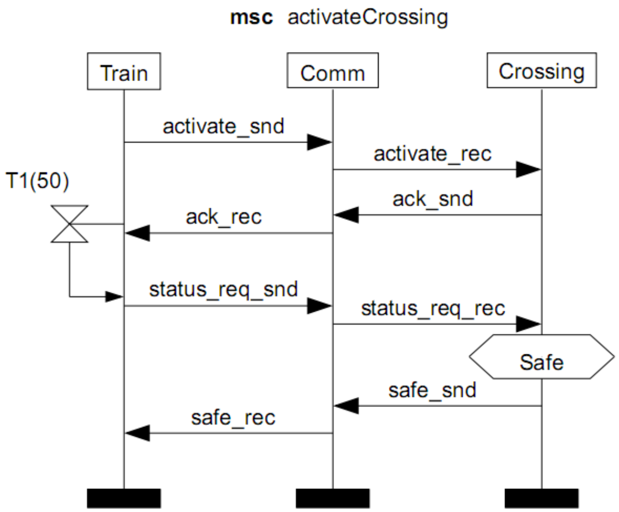
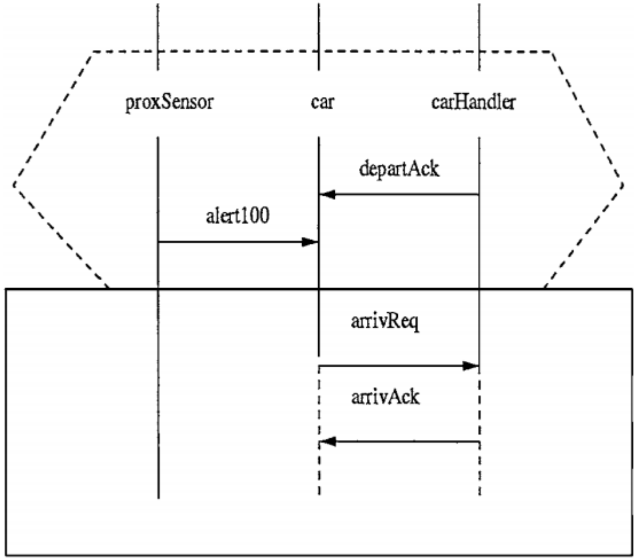
Communication modeling

Systems Engineering BSc Course

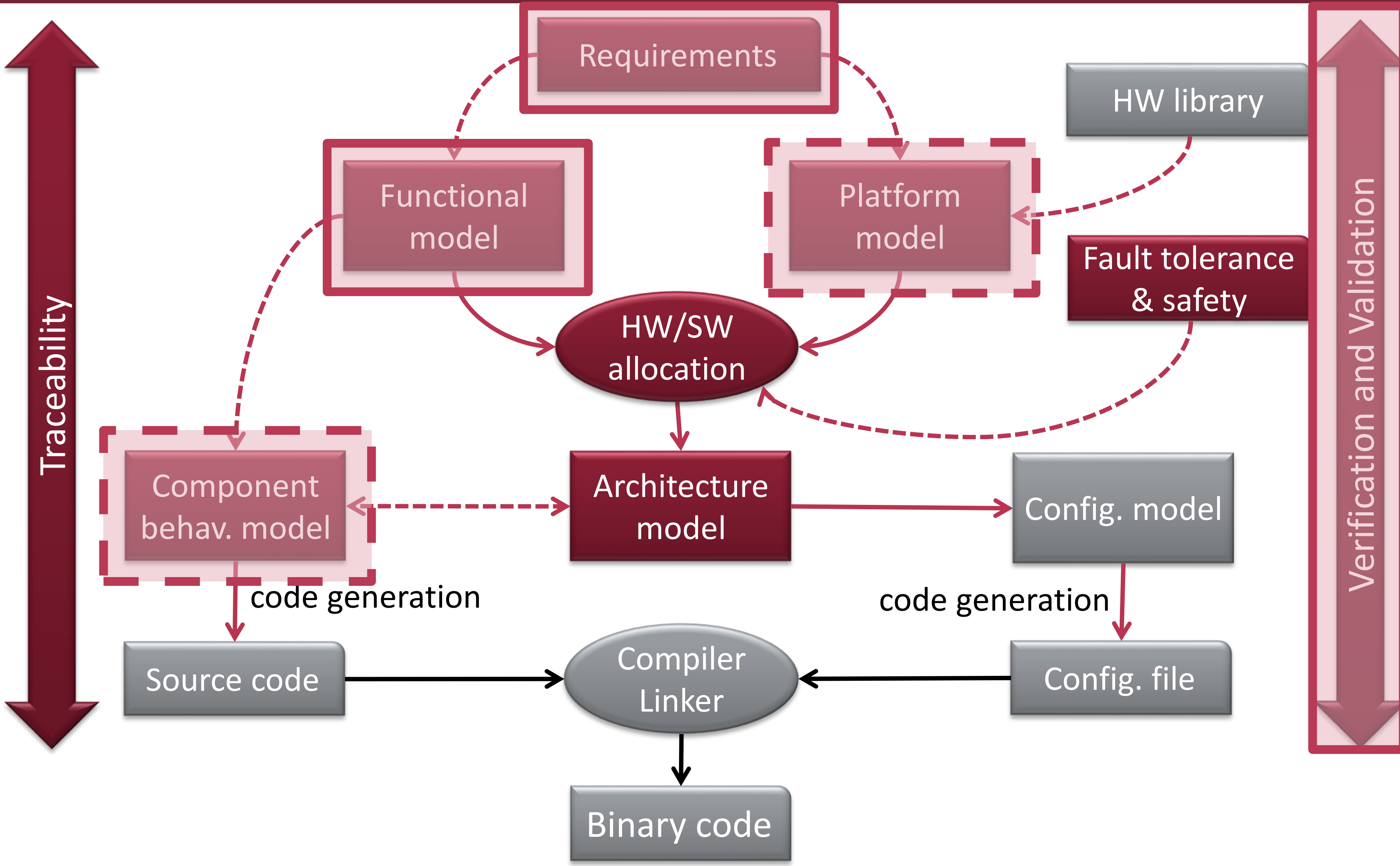
Roots & Relations

- Graphical **scenario** languages
  - Modeling **inter-object behavior**
- Example languages:
  - Message Sequence Charts (MSC)
  - Live Sequence Charts (LSC)
  - …





Platform-based systems design



Learning Objectives

Message-based interaction modeling

- Understand the basic blocks of message-based modeling
- Identify the participants, message types and constraints to describe inter-component behavior
- Understand the syntactic building blocks of UML Sequence Diagrams
- Understand the semantics of UML Sequence Diagrams
- Use Combined fragments to express complex logic and conformance relations
- Avoid ambiguity by fixing the interpretation of models according to a complete and sound semantics

MODELING INTERACTIONS

Objectives

Areas of application

Interaction diagram types

Objectives

Modeling inter-object communication

- **Order** and **type of messages** are important
  - Data and parameters are not the main focus
- *"Interactions do not tell the complete story"*
  - Specification of **certain scenarios only**
  - Samples of behavior rather than internal logic
- Should be applicable on **many levels**
  - Method call sequences of objects in a program
  - Messages between components of a system
  - Communication of nodes in a distributed system

Areas of application

- Refining **use cases**
  - Typical communication between actors and the system
- Modeling and analysis of **method call sequences**
  - *“What calls what and when?”*
- Designing **protocols**
  - Specification of allowed messages and their order
  - Often contains logic
- Visualizing an **execution trace** or log
- Specification of **test cases**
  - Requires assumptions, assertions, etc.

Relations to other diagrams

- **Uses model elements from**
  - **Structure:** Class, Block, Component
  - **Behavior:** Signals, Operations of classes

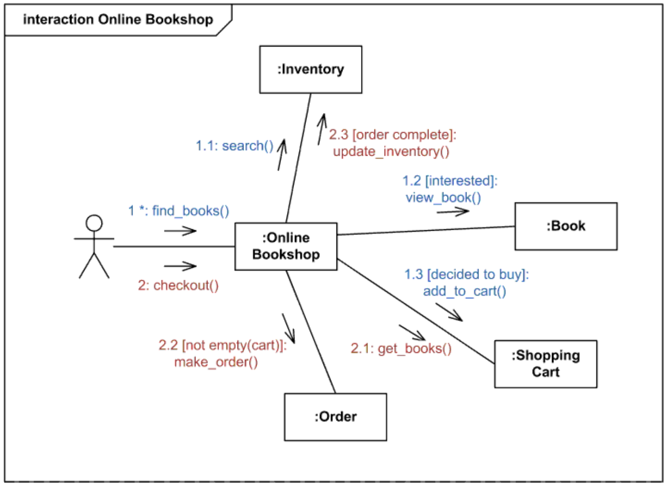
- **Refines**
  - **Use case:** basic and alternate flows
  - **Activity:** high-level activities, provides alternative view

Interaction diagram types

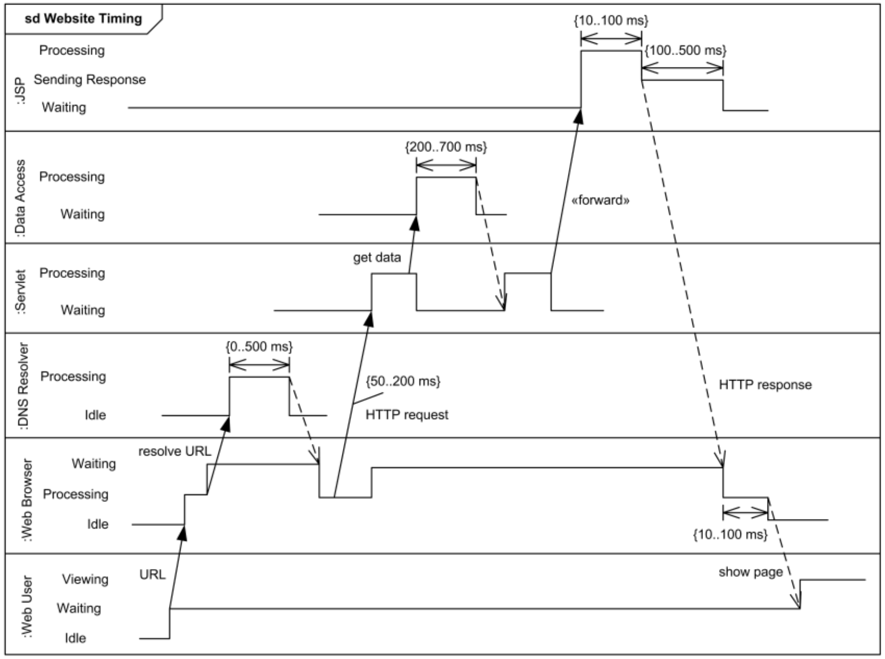
- **Sequence Diagram**
  - Models a sequence of messages between objects
  - Can include logic, timing, parameters, etc.
- **Communication Diagram**
  - Focuses on a single message flow
- **Interaction Overview Diagram**
  - Models control flow between different Interactions
  - Similar to Activity Diagrams
- **Timing Diagram**
  - Focuses on timing

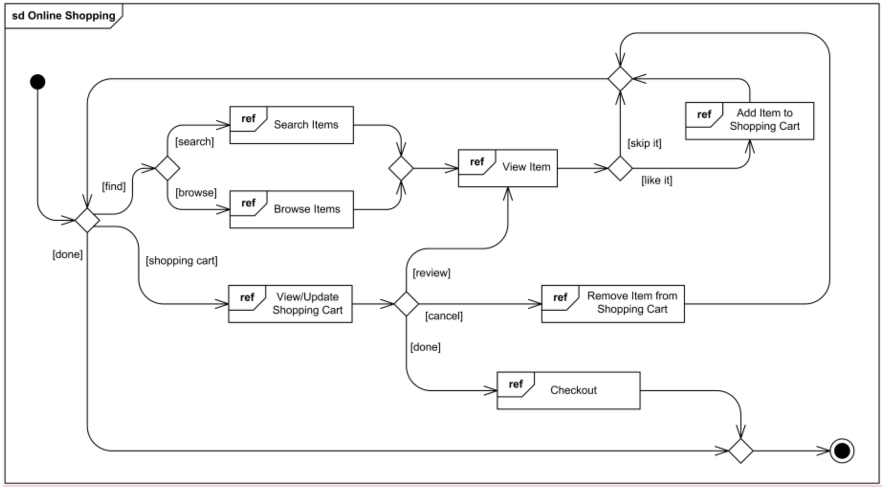
Interaction diagram types

Communication



Timing





Interaction Overview

Source: http://www.uml-diagrams.org

UML SEQUENCE DIAGRAM

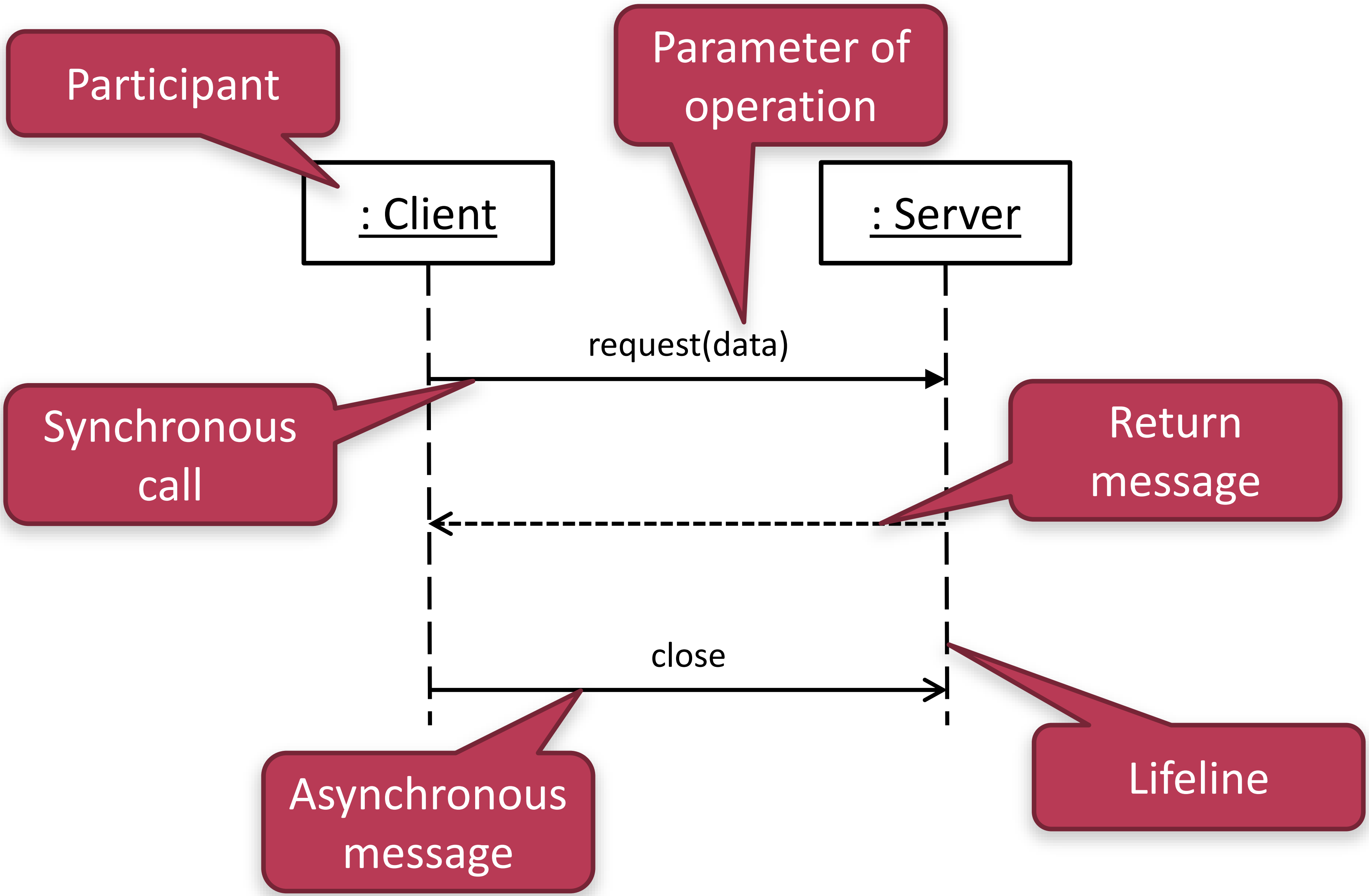
Basic building blocks

Lifecycle & Special messages

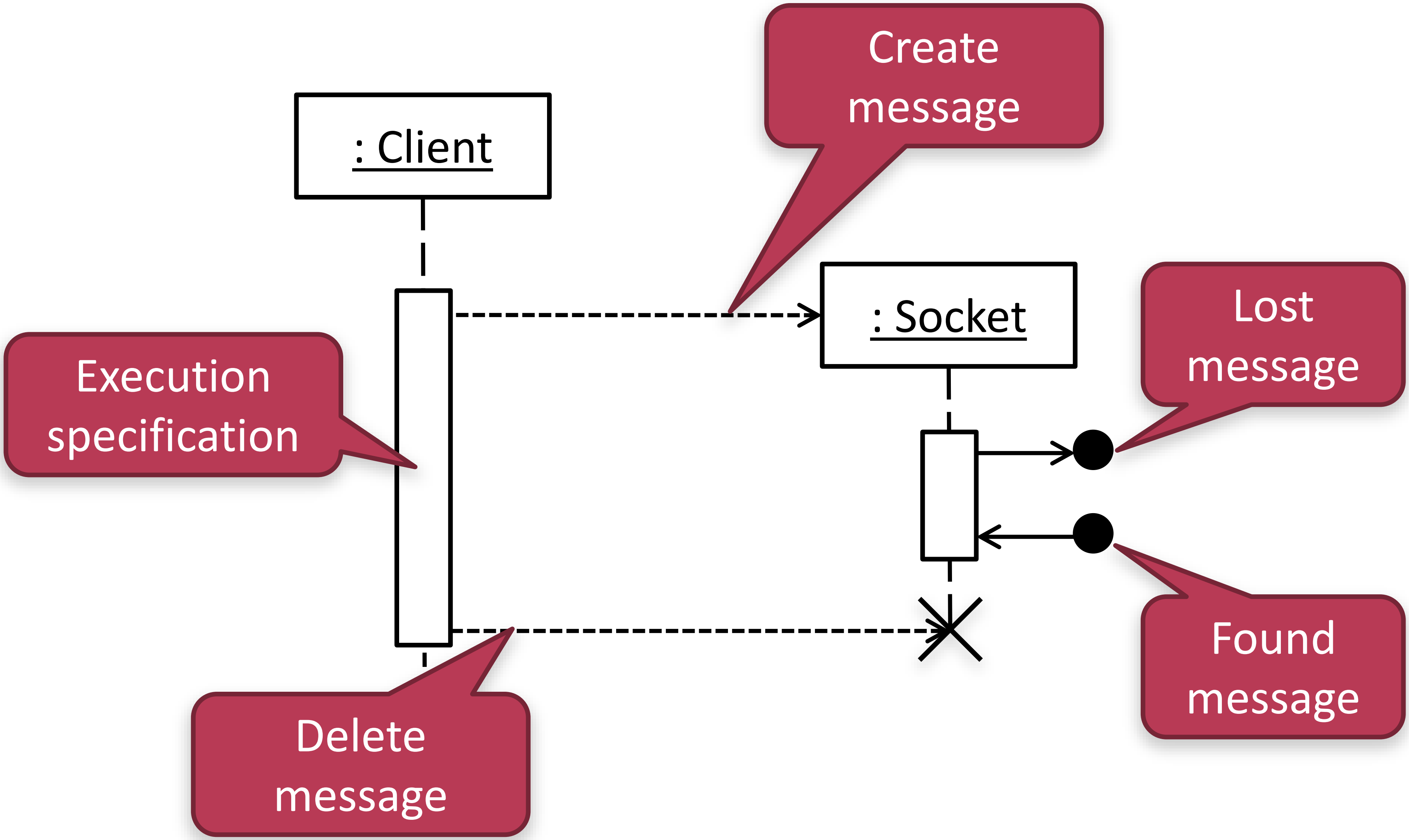
Combined fragments & References

Timing & Invariants

Basic building blocks



Lifecycle & special messages



Basic building blocks

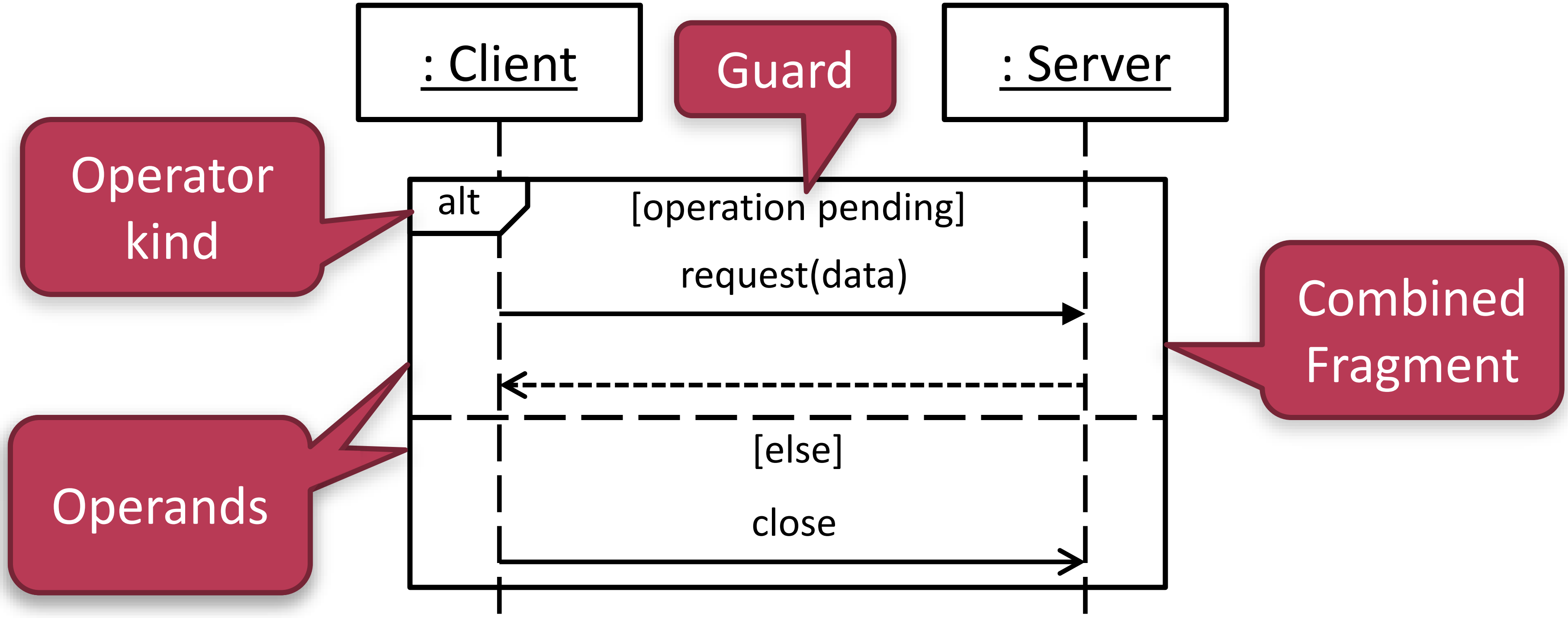
- **Participants**
  - ~~**Instances**~~ **Roles** of a class or block
  - Have a lifeline that denotes the span of their **existence**
  - Can have a name and/or a type
- **Messages**
  - **Synchronous** calls
    - Usually have a return message (optional)
  - **Asynchronous** messages (async. calls or signals)
  - Calls and messages may have arguments
    - A dash ("–") denotes an undefined argument
    - (Arguments are not the strong point in Sequence Diagrams)

Lifecycle & special messages

- **Create & delete message**
  - Denotes the **creation/destruction** of another participant
- **Execution specification**
  - Denotes the duration when a participant is **active**
    - Either processing or waiting for a synchronous response
  - Not mandatory, but tools usually use them
    - Good for one active thread
    - Confusing for more
- **Lost & found messages**
  - Source or target is either not known or **not important**

Combined fragments

- Operators to express complex scenarios
  - Can have several operands
  - Each operand can have a guard

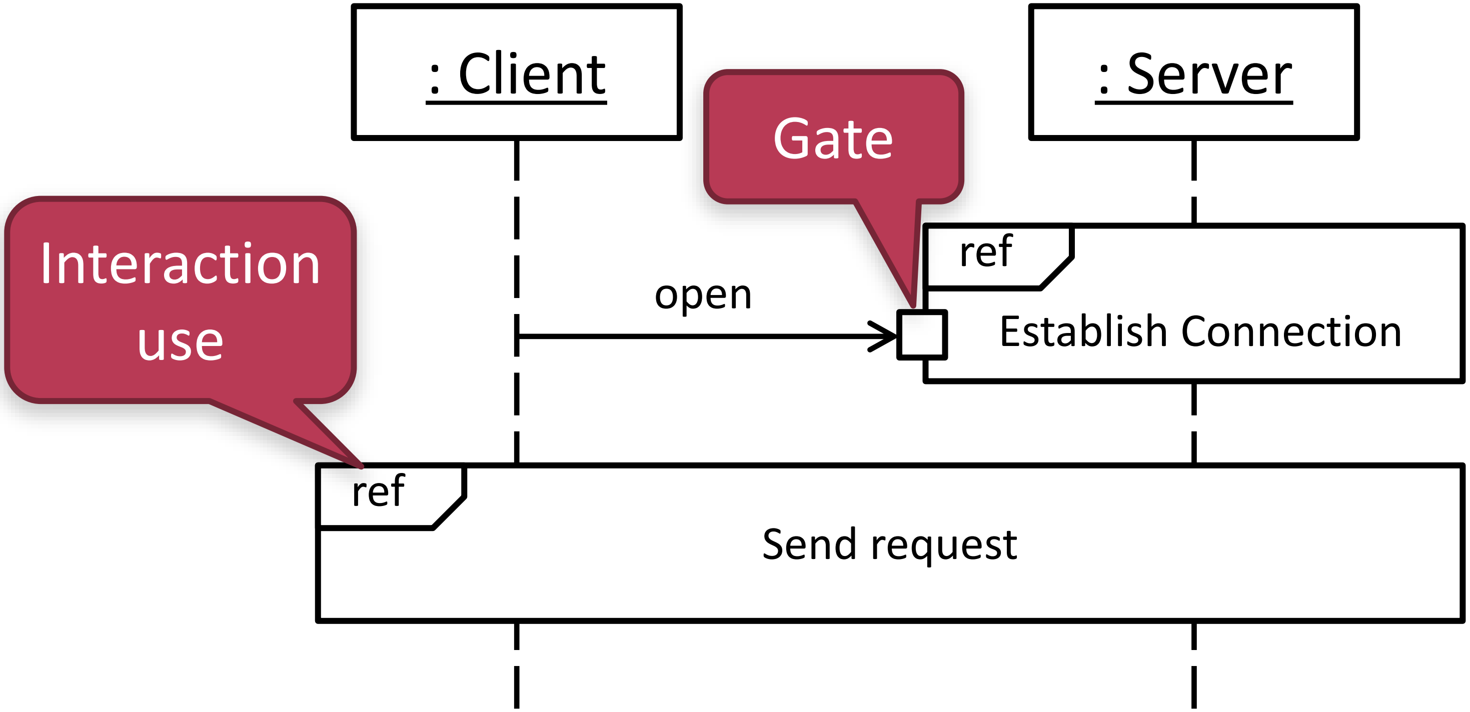


Combined fragments

- Operators for choice and iteration
  - **alt**: choice between the operands
  - **opt**: choice between the sole operand or nothing
  - **loop**: loop with lower or upper bound
  - **break**: represents a breaking scenario
- Operators for parallelization and sequencing
  - **par**, **strict**, **seq**, **critical**
- Operators related to the conformance relation
  - **neg**, **assert**, **ignore**, **consider**
- (See semantics later)

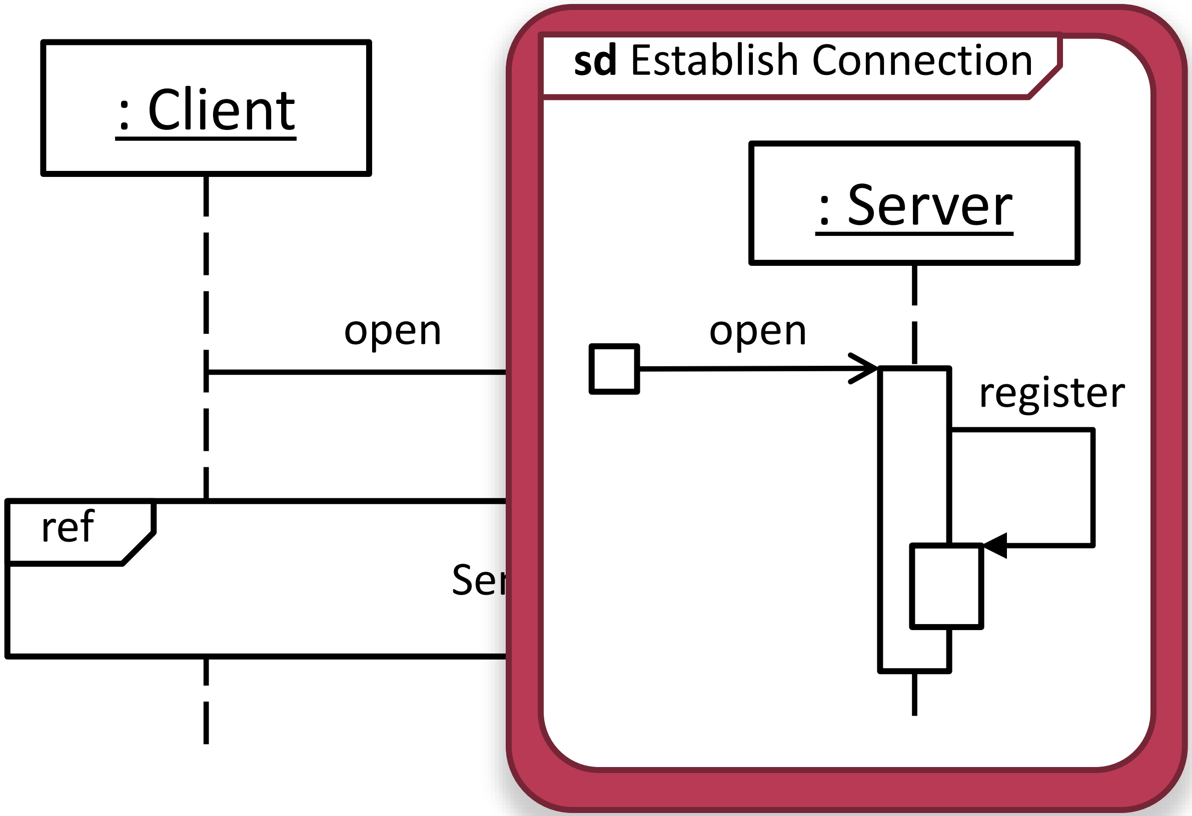
Interaction use

- Interactions support decomposition and reuse



References

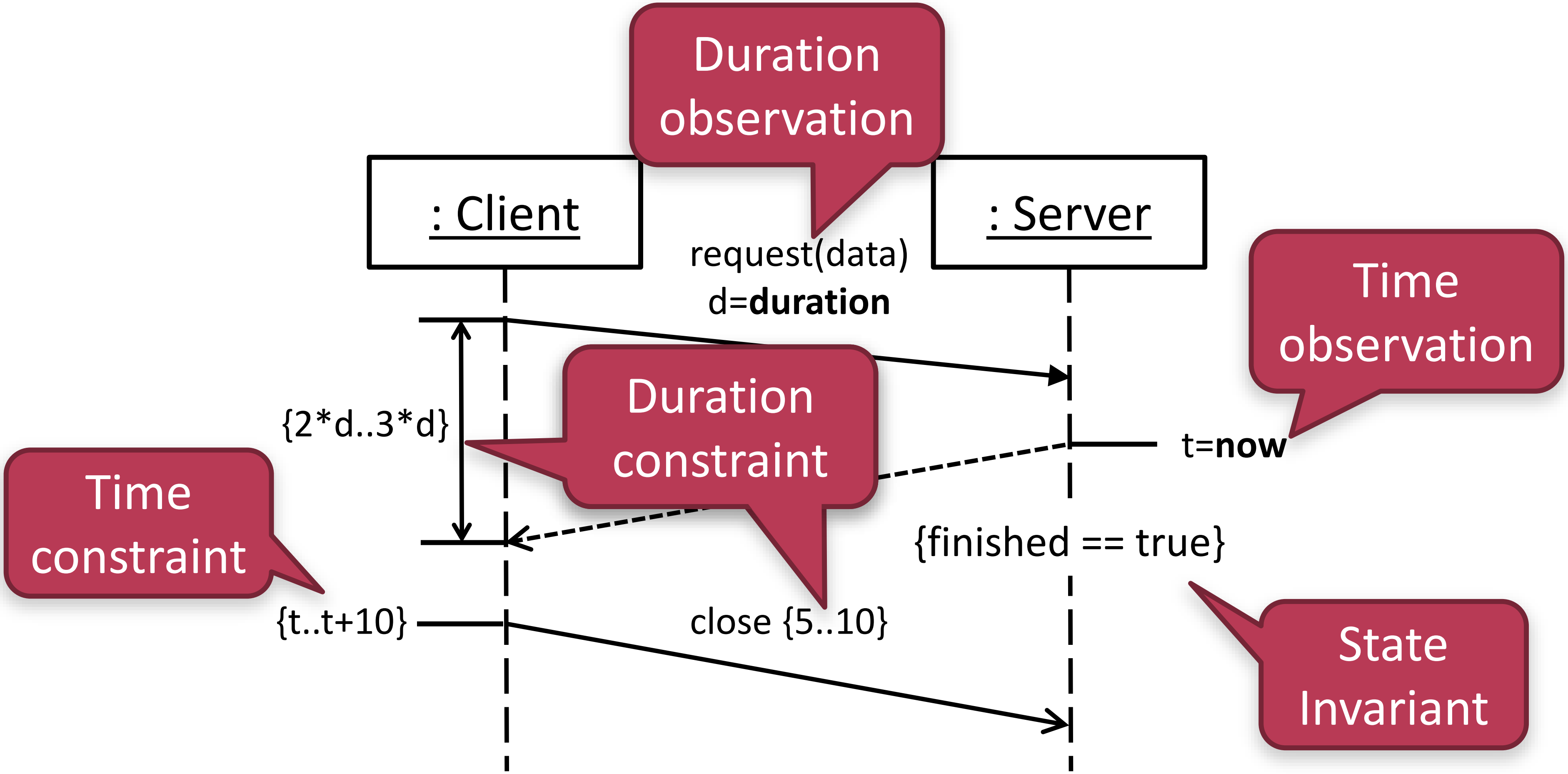
- Interactions support decomposition and reuse



Timing & Invariants

Elapsed time can be expressed and constrained

- **Observations** and **Constraints**



Summary

- **Participants**
  - Lifeline and Execution specification
- **Messages**
  - Synchronous & asynchronous
  - Lost & found
  - Create/delete messages
- **Combined fragments**
  - Logic, parallelism, sequencing, conformance relation
- **Interaction use**
- **Timing and State invariants**

SEMANTICS

Model of semantics

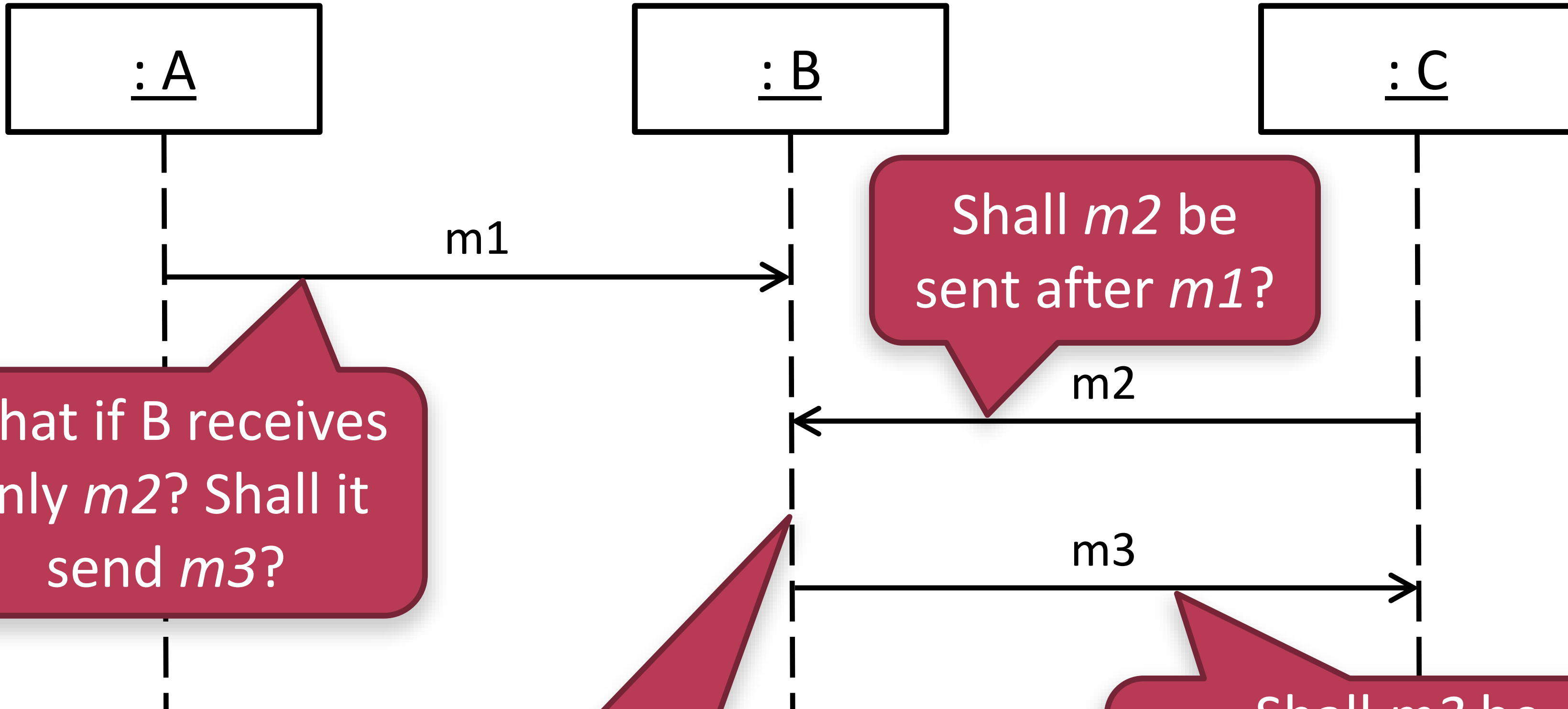
Basic rules

Semantics of Combined Fragments

Final word of caution

Introduction

Is this whole sequence always happening? Sometimes happening? Is it the entire behaviour of the system?



Shall *m2* be sent after *m1*?

What if B receives only *m2*? Shall it send *m3*?

May B send an *m4* message here?

Shall *m3* be definitely sent? Or just optionally?

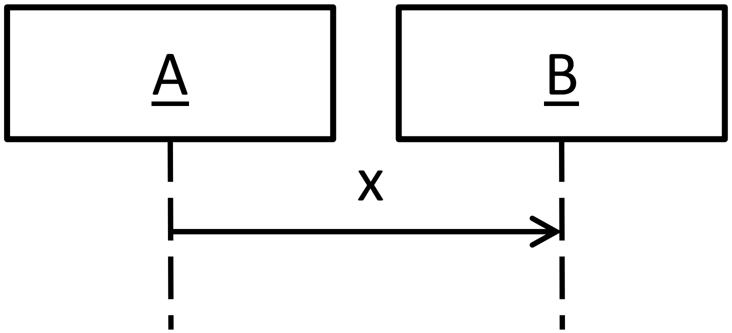
Model of semantics

- Semantics is defined as the **sets of traces** that are
  - valid, invalid, or inconclusive
  - for the Sequence Diagram.
- Elements of a trace: **event occurrences**
  - Sending messages
  - Receiving messages

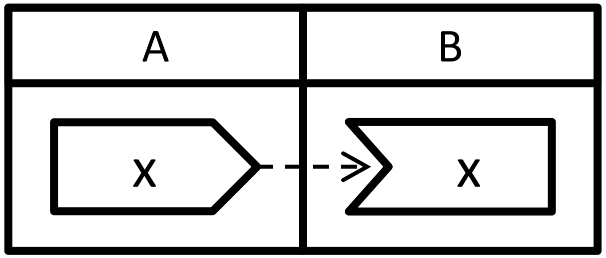
Analogy with regular expressions

- A Sequence Diagram defines a **partial order**
  - Several traces may be valid
- Negative fragments (*neg*), assertions (*assert*) and State Invariants define negative traces

Basic rules

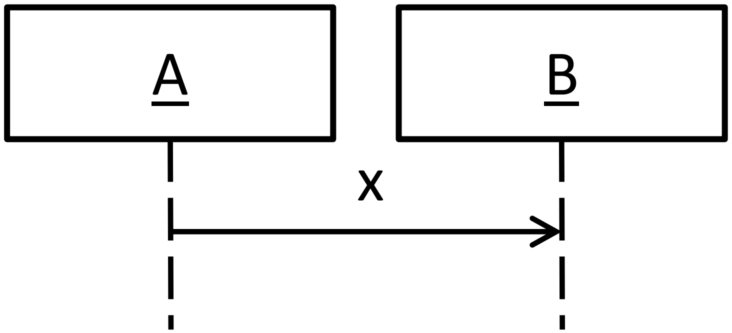


- 2 events
  - Sending *x* in A **!x**
  - Receiving *x* in B **?x**

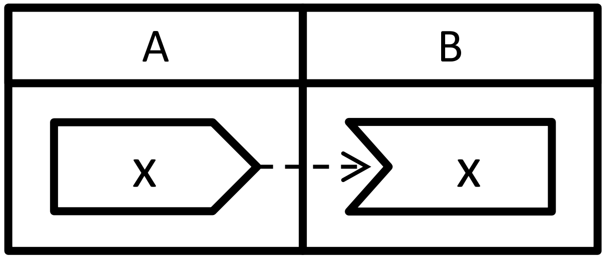


- Weak (partial) ordering: „happens-before”
  - Occurrences on the **same lifeline** are **ordered**
  - Receiving a message occurs **after** sending it (*causality*)
- Valid traces: { **⟨!x, ?x⟩** }

Basic rules



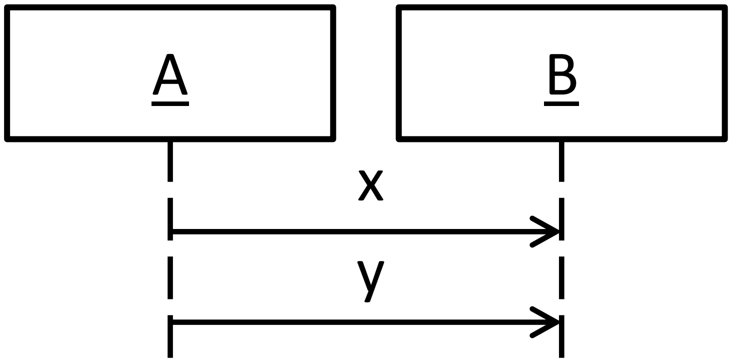
- 2 events
  - Sending *x* in A **!x**
  - Receiving *x* in B **?x**



- Weak (partial) ordering: „happens-before”
  - Occurrences on the **same lifeline** are **ordered**
  - Receiving a message occurs **afte**[…] y)
- Valid traces: { **⟨!x, ?x⟩** }

Every other trace is **inconclusive**

Weak sequencing (default)

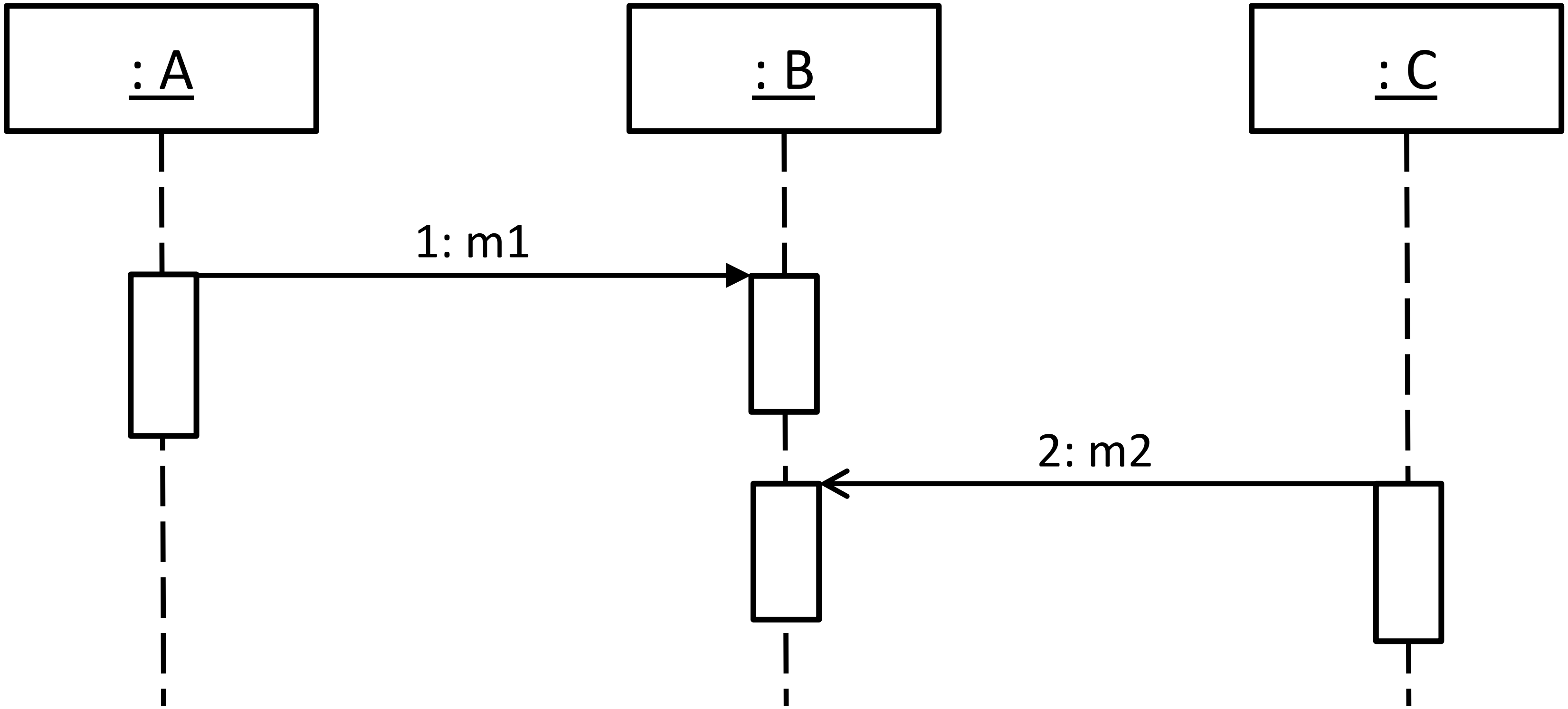


- Weak sequencing: ⟨!x, ?x⟩ **seq** ⟨!y, ?y⟩
  - Preserves the order within the operands
  - Occurrences are ordered **only on the same lifeline**
    - In the order of the operands
    - ?x and !y are **not ordered**
- Valid traces:

  { ⟨!x, ?x, !y, ?y⟩ , ⟨!x, !y, ?x, ?y⟩ }

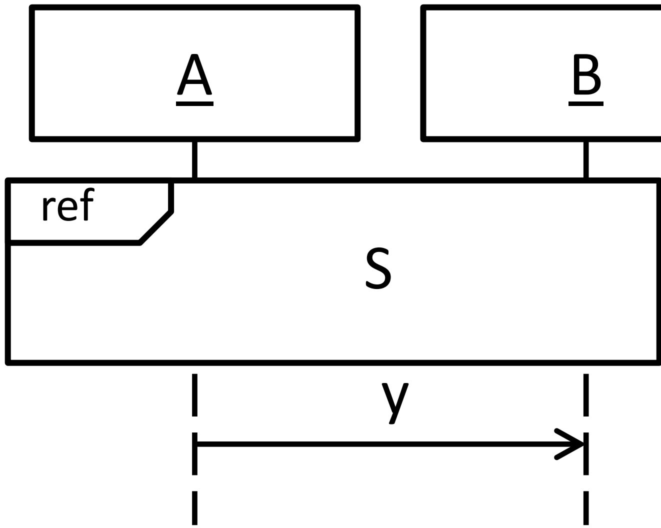
Caution: message sequence numbers

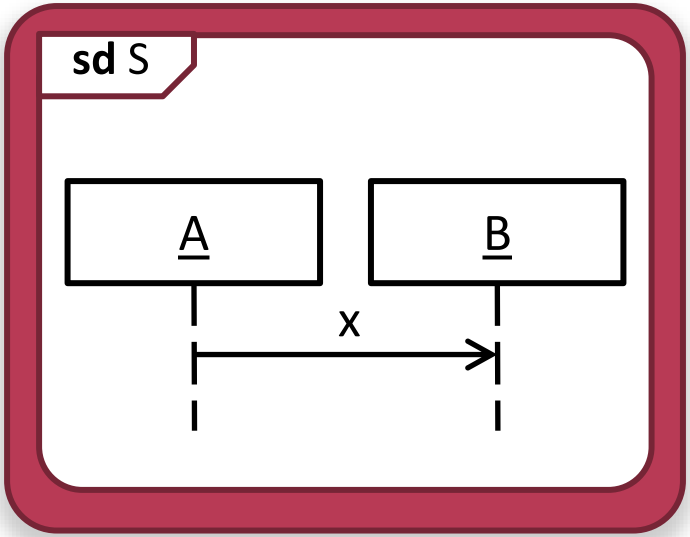
- Some tools use automatic sequence numbers



- Why is this a bad idea?

Caution: Interaction use

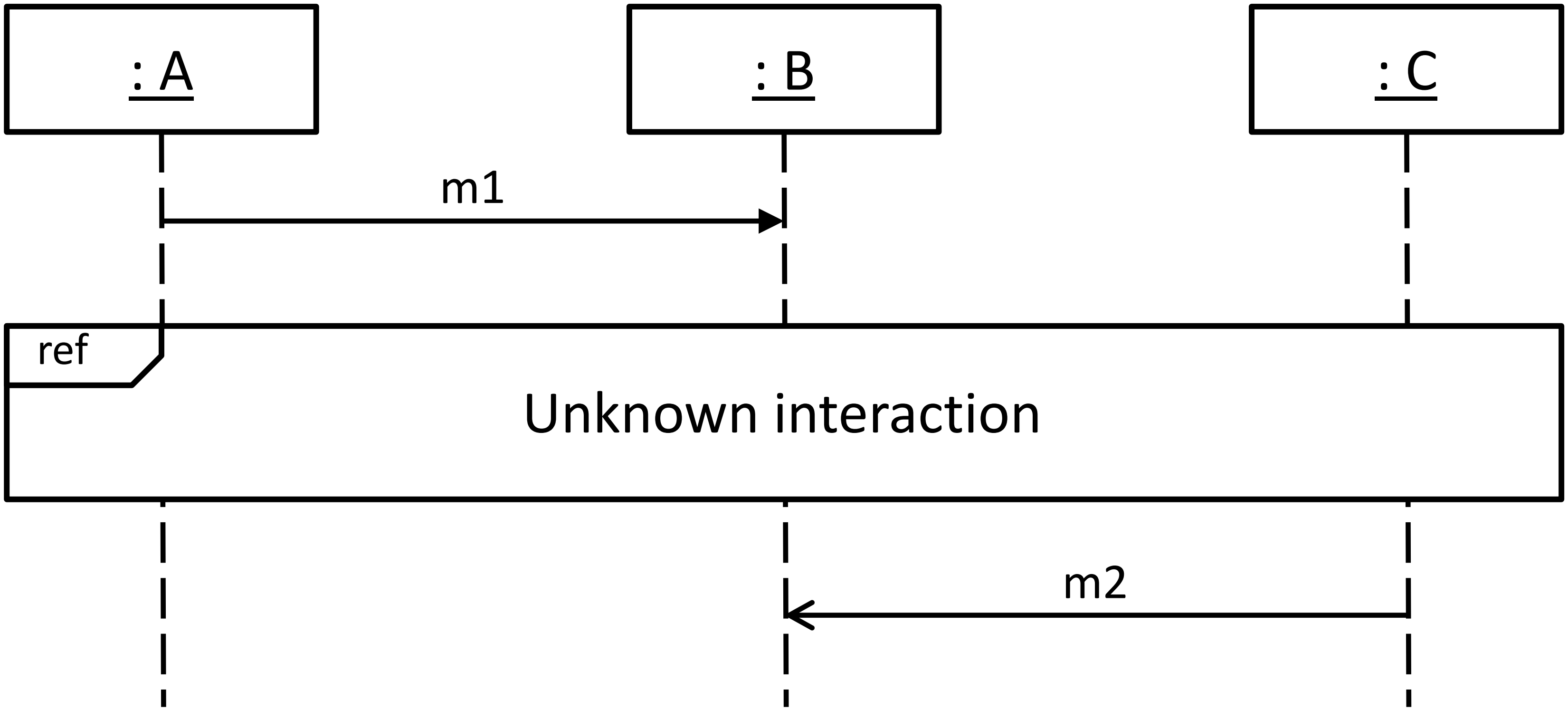




- Interaction occurrence: **S seq ⟨!y, ?y⟩**
  - **Just a shortcut:** equivalent to pasting **S**
- Valid traces:

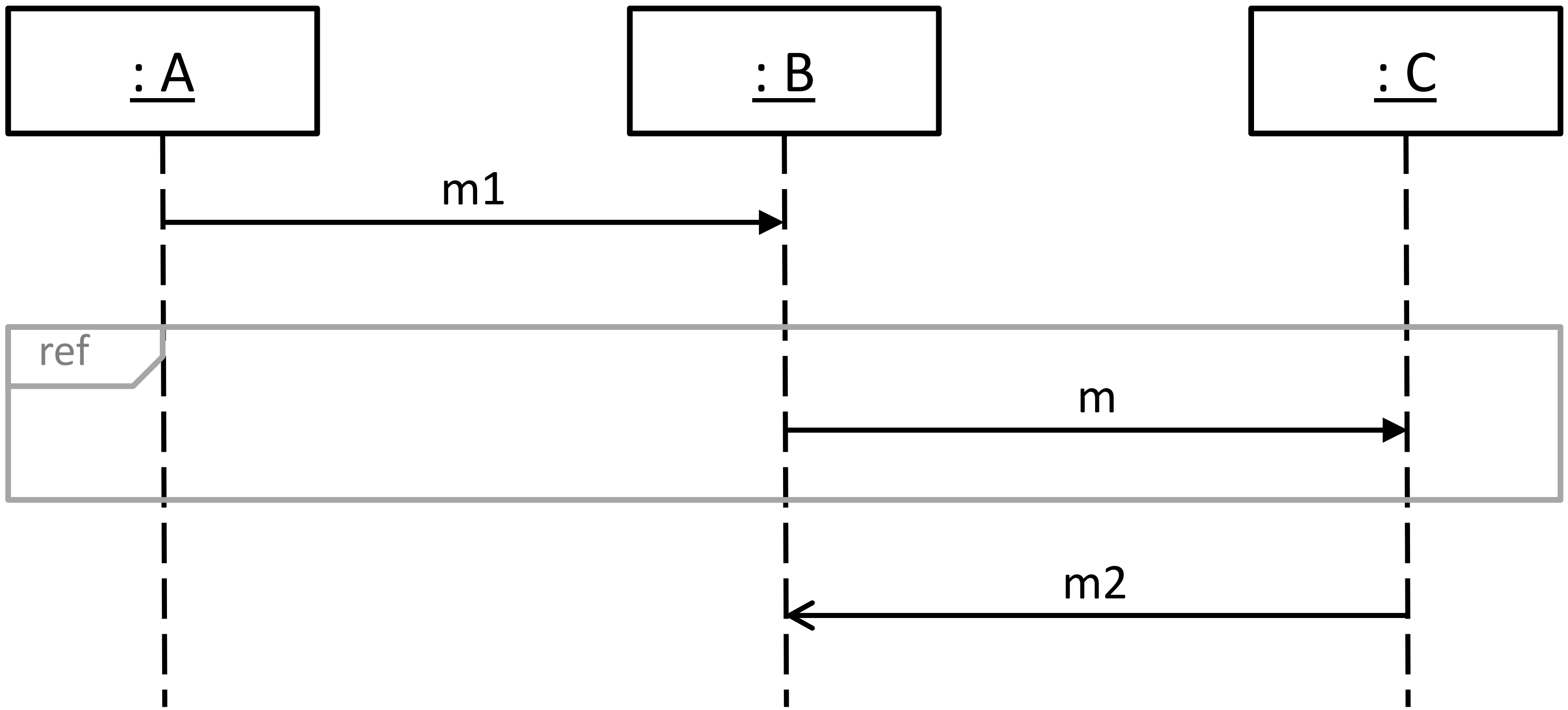
  { **⟨!x, ?x, !y, ?y⟩** , **⟨!x, !y, ?x, ?y⟩** }

Caution: Interaction use



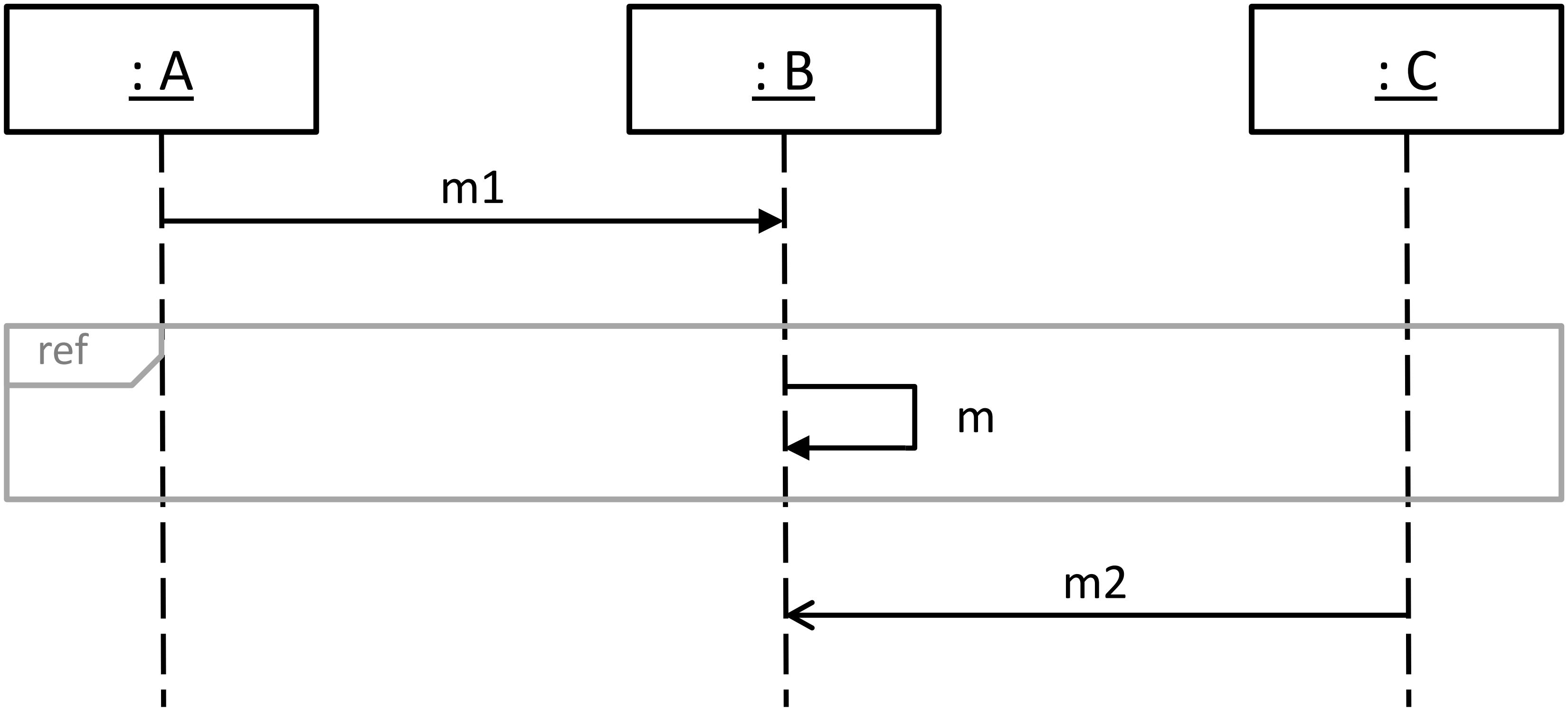
- Which one **may be sent first**?
  - only *m1*, only *m2* or both?

Caution: Interaction use



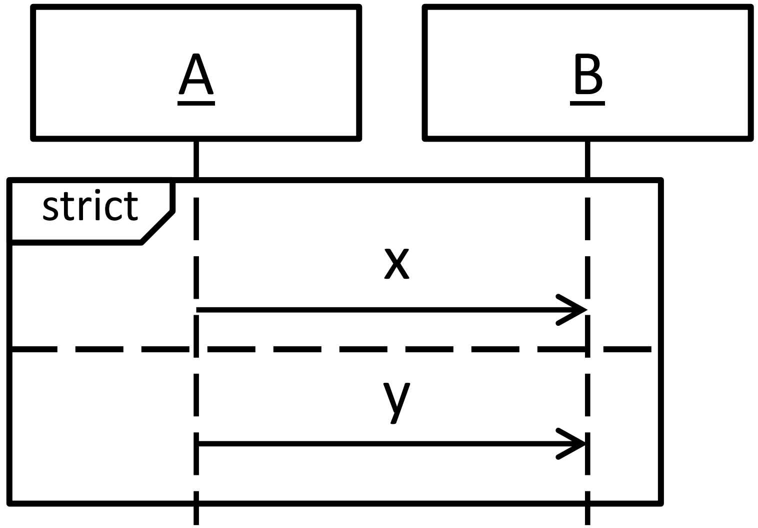
- Which one **may be sent first**?
  - ***only m1***, *only m2* or both?

Caution: Interaction use



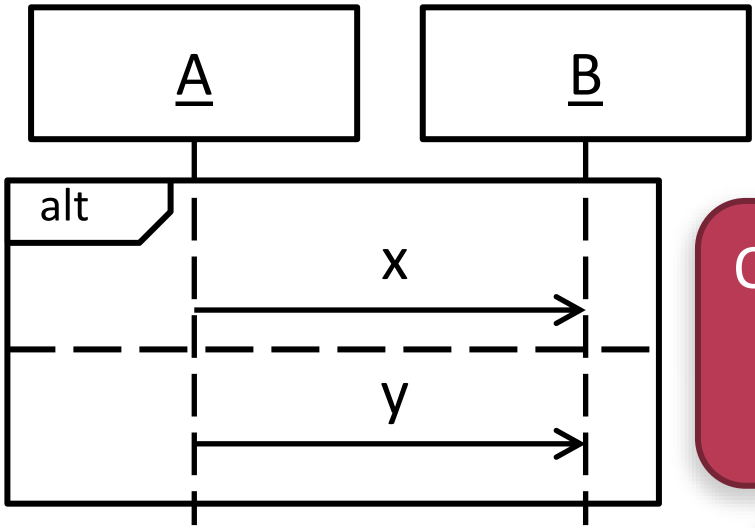
- Which one **may be sent first**?
  - *only m1, only m2* or **both***?*

Strict sequencing



- Strict sequencing: ⟨!x, ?x⟩ **strict** ⟨!y, ?y⟩
  - Preserves the order within the operands
  - Occurrences are ordered **on all lifelines**
    - In the order of the operands
- Valid traces: { ⟨!x, ?x, !y, ?y⟩ }

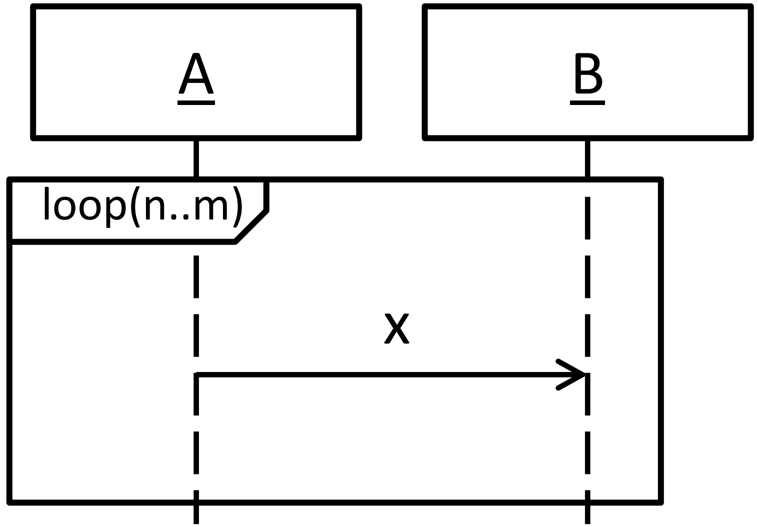
Alternative fragments



Only operands with satisfied guards participate!

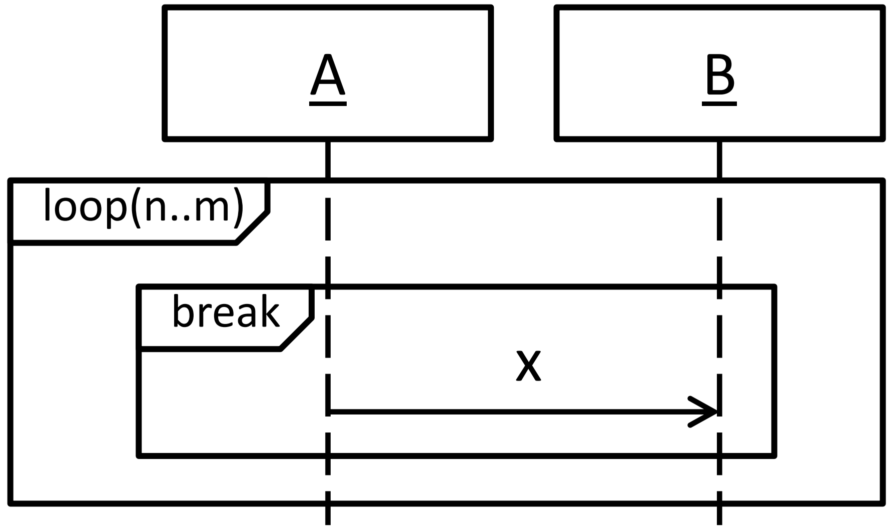
- Alternative fragments: ⟨!x, ?x⟩ **alt** ⟨!y, ?y⟩
  - Union of the valid traces of the operands
  - Optional fragment: **opt** ⟨!x, ?x⟩ = ⟨!x, ?x⟩ **alt** ⟨⟩
- Valid traces: { ⟨!x, ?x⟩, ⟨!y, ?y⟩ }

Loop fragment



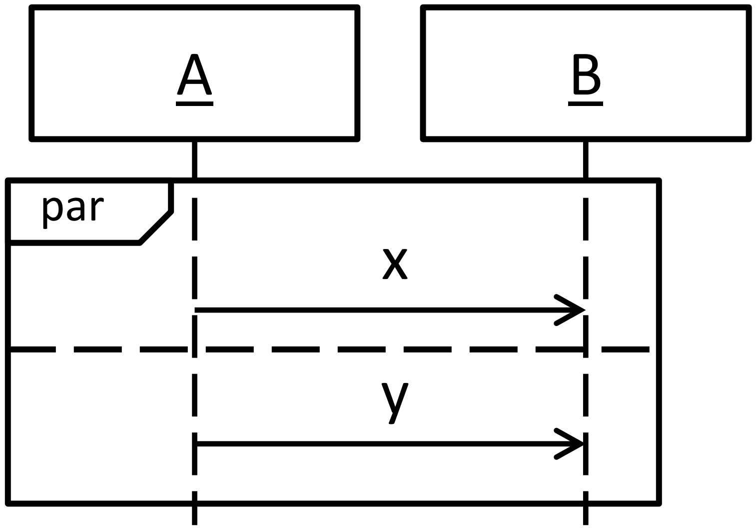
- Loop fragment: $\langle !x, ?x \rangle^{n..m}$
  - Valid traces of operands concatenated *n* to *m* times
  - Only repeats while the (optional) guard is true!
    - Hybrid of a *for* and a *while* loop

Break fragment



- Break fragment:
  - Executes fragment behavior, then
  - **Terminates** the execution of the **enclosing fragment**
    - And only the innermost
  - Only if the guard is true
    - Without a guard: **non-determinism** (UML 2.5.1, Sect. 17.6.3.9.)

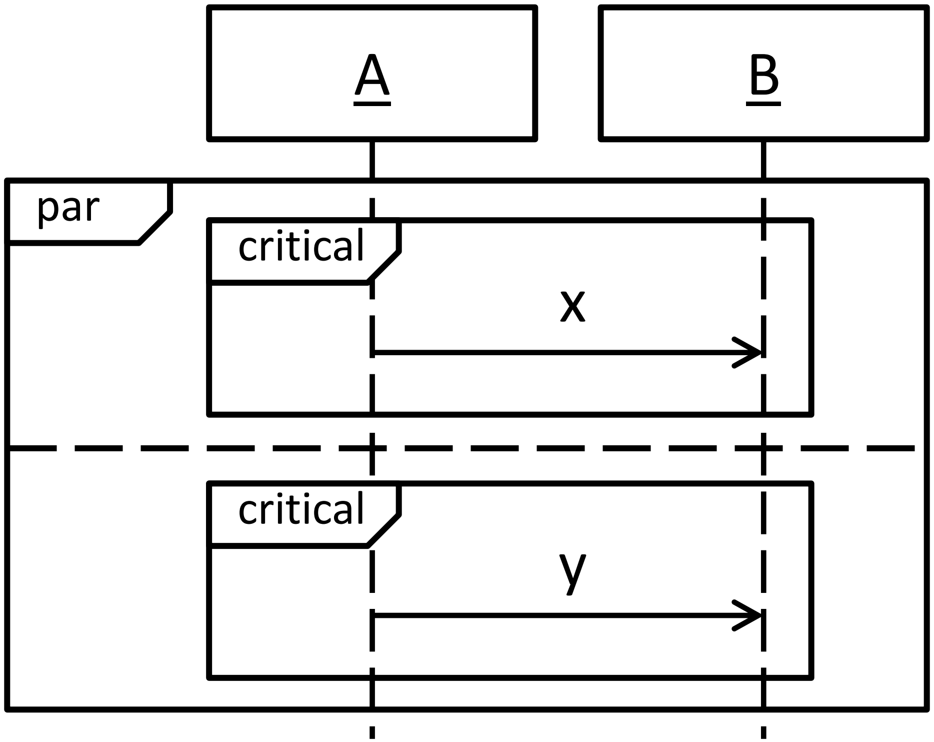
Parallel fragments



- Parallel fragments: ⟨!x, ?x⟩ **par** ⟨!y, ?y⟩
  - Arbitrary interleaving of operand behaviors
- Valid traces:

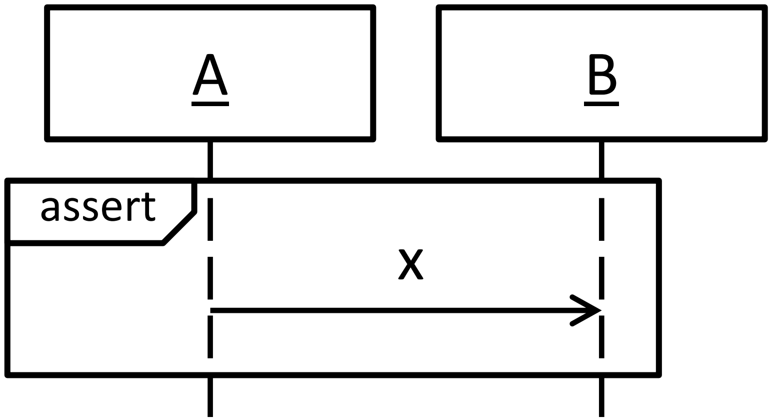
  { ⟨!x, ?x, !y, ?y⟩, ⟨!x, !y, ?x, ?y⟩, ⟨!x, !y, ?y, ?x⟩, ⟨!y, ?y, !x, ?x⟩, ⟨!y, !x, ?y, ?x⟩, ⟨!y, !x, ?x, ?y⟩ }

Critical fragments



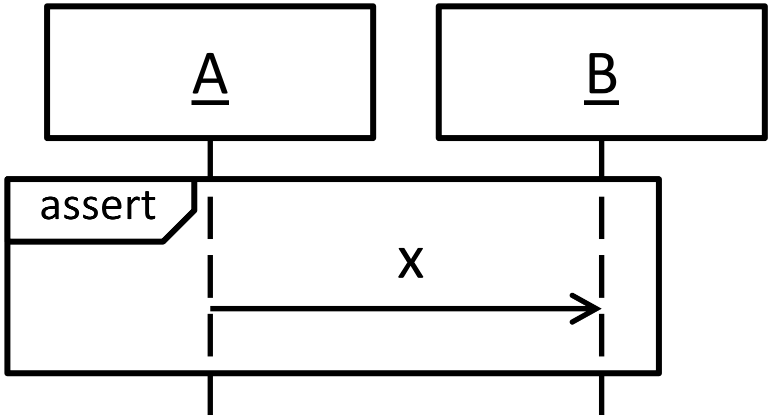
- Critical fragments:
  - Behavior is atomic and cannot be interleaved
- Valid traces: { ⟨!x, ?x, !y, ?y⟩, ⟨!y, ?y, !x, ?x⟩ }

Assertion fragments



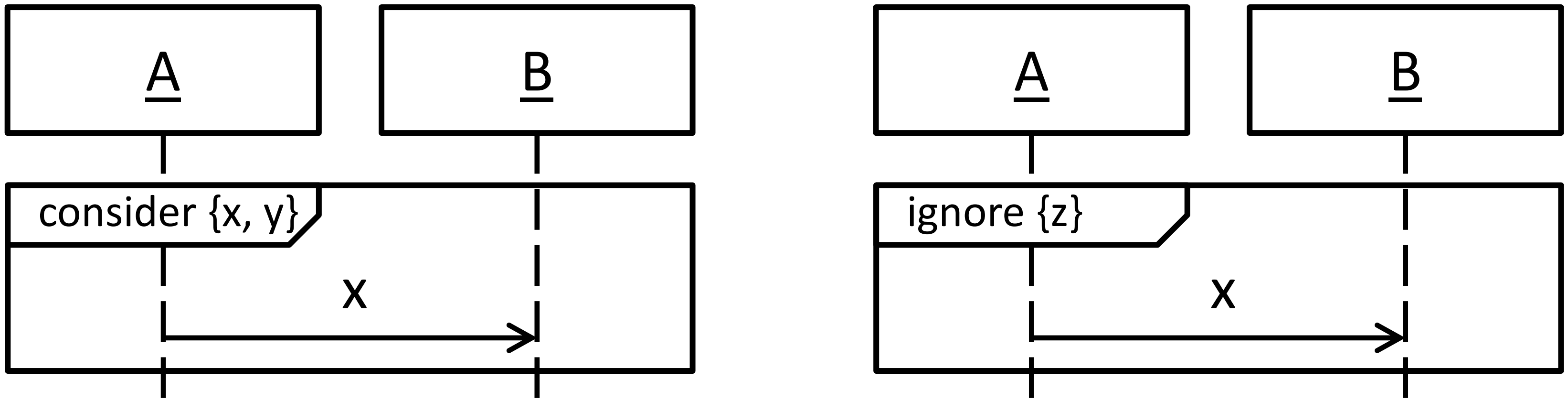
- Assertion fragments: **assert** **⟨!x, ?x⟩**
  - Specifies exactly what **must** happen
- Valid traces: { **⟨!x, ?x⟩** }
- Invalid traces: { **⟨!x, ?x⟩** }^{C}
  - (complement of valid traces)

Negative fragments



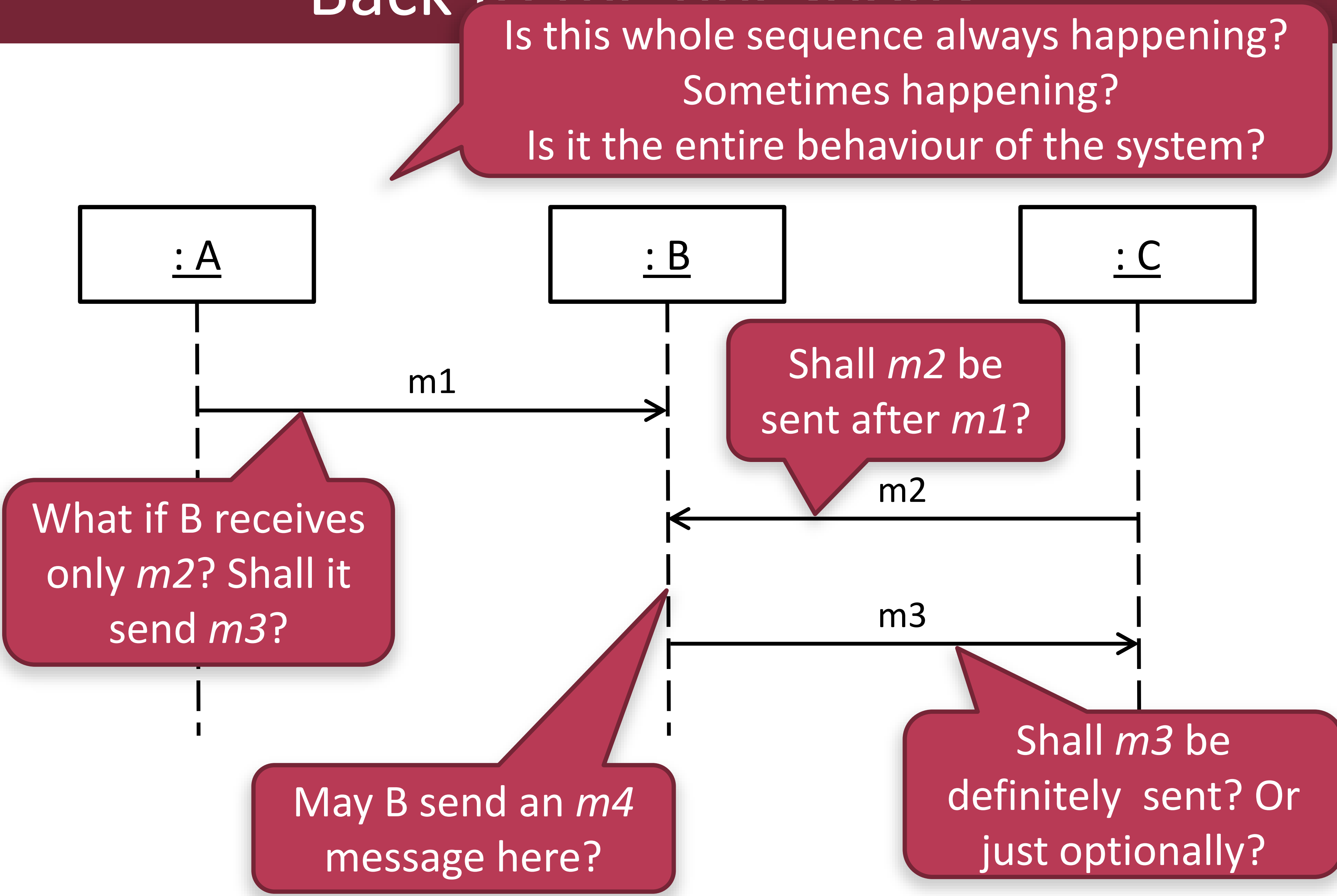
- Negative fragments: **neg** **⟨!x, ?x⟩**
  - Specifies what **must not** happen
- **In**valid traces: { **⟨!x, ?x⟩** }
- Inconclusive traces: { **⟨!x, ?x⟩** }^{C}
  - (complement of invalid traces)

Consider and Ignore fragments



- Assume there are 3 kinds of messages: *x, y* and *z*
  - *Consider* and *Ignore* filter out the **irrelevant** messages
  - The message *z* can appear in any number and any interleaving
- Valid traces: ⟨!x, ?x⟩ **par** ⟨!z, ?z⟩*

Back to the questions



Final word of caution

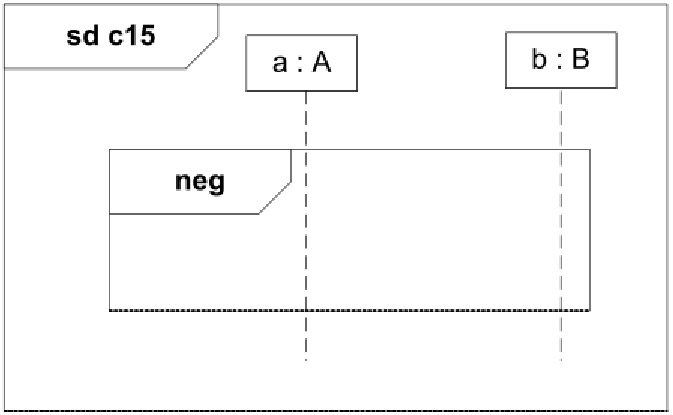
- There are a lot of **variants**
  - Depending on the domain and purpose
- And some **open questions** as well
  - E.g. can traces have pre-/postfixes?

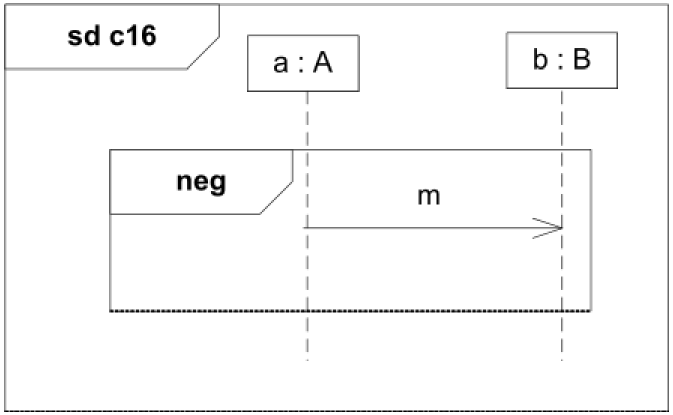
- Conclusion:

Fix your interpretation

prior to using Sequence Diagrams

Possible variations





	c15			c16		
Approach	Valid	Invalid	Inconclusive	Valid	Invalid	Inconclusive
Störrle	$\emptyset$	$\{\epsilon\}$	$\Sigma^* - \{\epsilon\}$	$\emptyset$	$\{!m.?m\}$	$\Sigma^* - \{!m.?m\}$
STAIRS	$\{\epsilon\}$	$\{\epsilon\}$	$\Sigma^* - \{\epsilon\}$	$\{\epsilon\}$	$\{!m.?m\}$	$\Sigma^* - \{\epsilon, !m.?m\}$
Cengarle & Knapp	$\{\epsilon\}$	$\emptyset$	$\Sigma^* - \{\epsilon\}$	$\{\epsilon\}$	$\{!m.?m\}$	$\Sigma^* - \{\epsilon, !m.?m\}$
Grosu & Smolka	$\Sigma^* - \{\epsilon\}$	$\{\epsilon\}$	$\emptyset$	$\Sigma^* - \{!m.?m\}$	$\{!m.?m\}$	$\emptyset$
Cavarra & Filipe, Küster-Filipe	$\emptyset$	Σ^*	$\emptyset$	$\emptyset$	$\{!m.?m\}$	$\Sigma^* - \{!m.?m\}$

For other choices and variations see: Z. Micskei and H. Waeselynck: *The many meanings of UML 2 Sequence Diagrams: a survey*, SoSyM, 10(4):489-514, Springer, 2011.

MODELING WITH UML SEQUENCE DIAGRAMS

Modeling Actor-System interactions

Visualizing traces or typical behavior

Modeling protocols

Defining test cases

Modeling Actor-System interactions

Typically the refinement of **use cases**

- Mostly using **simple elements** only
  - No complex logic (Combined fragments)
  - Semantics is not very important here
- Helps in
  - …the definition of system-level **ports and interfaces**
  - …identifying **data exchanged** between the system and the environment

Visualizing traces or typical behavior

Typically a **single scenario**

- Not to define a behavior, but to **understand** aspects of it
  - Focus is on the order of events and messages
  - Minimal usage of logic (Combined fragments)
  - Often assumes **implicit strict sequencing**
    - Everything happens in vertical order
- Helps in:
  - **Understanding/analyzing** certain behaviors of the system

Modeling protocols

Typically heavy focus on **messages & complex logic**

- One way to define a protocol
  - Use Sequence Diagrams to **design phases**
    - What to send and when (timing)
    - More complex usage of Combined fragments
  - Use Interaction Overview Diagram to **link the phases**
- Alternatives:
  - Activity Diagram if the internal logic is more important
  - State Machine if heavily state-based
    - Still using Sequence Diagrams to visualize communication

Defining test cases

Typically has a **trigger/setup** and an **assertion** phase

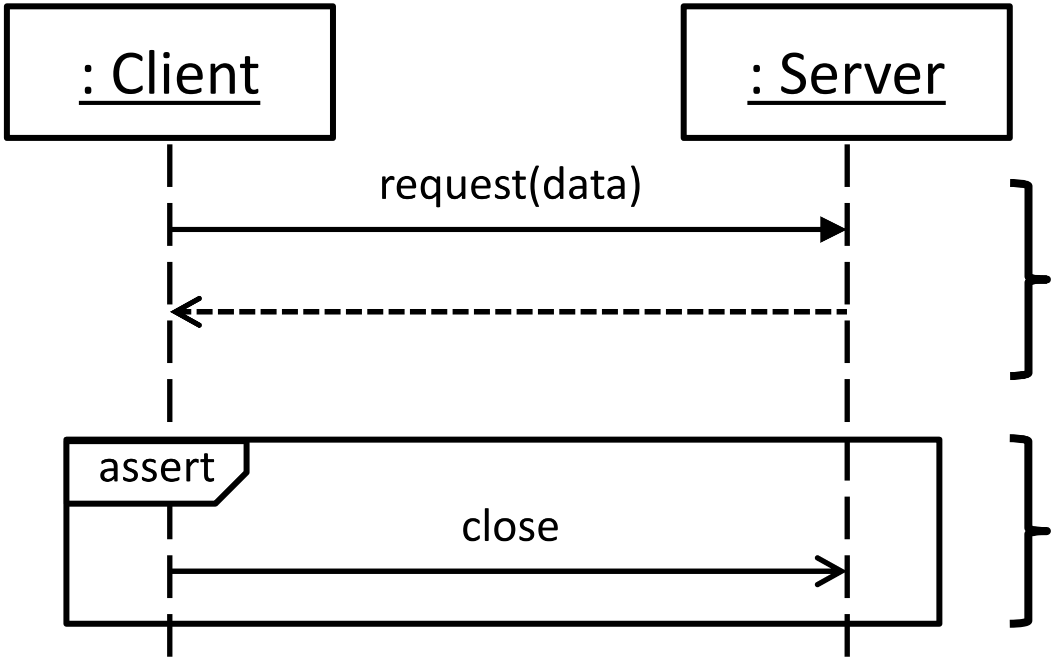
- **Trigger/setup phase** (*may*)
  - Decides if the observed trace **belongs to the test case**
  - Result may be ***inconclusive*** if trace deviates here
    - Otherwise the assertion phase starts
- **Assertion phase** (*must*)
  - The trace is considered ***invalid*** if it deviates here

- Heavy use of **conformance-related fragments**
  - Semantics really matter here

Defining test cases

Test case:

"Once the Client sent a request and the Server replied, the Client must close the connection."



Setup/trigger phase:
Prepares the system

Assertion phase:
Tests the system

If **setup** occurs and **assertion** does not → **invalid** (test failure)

If **setup** does not occur → **inconclusive** (different test case)

Summary

- Interactions model **inter-object behavior**
- Several diagram types in UML
  - **Sequence Diagrams** are used most frequently
- **Powerful language**, many elements
  - Can be used for requirements, design, tests…
- But **interpretation** has to be fixed in the team!