CHRISTIAN HUEMER
MARION SCHOLZ

Object-Oriented Modeling with UML
PART II – Sequence Diagram

Sequence Diagram

The Interaction Diagrams

Christian Huemer and Marion Scholz

Presented by Nicholas Bzowski

Interactions and Messages

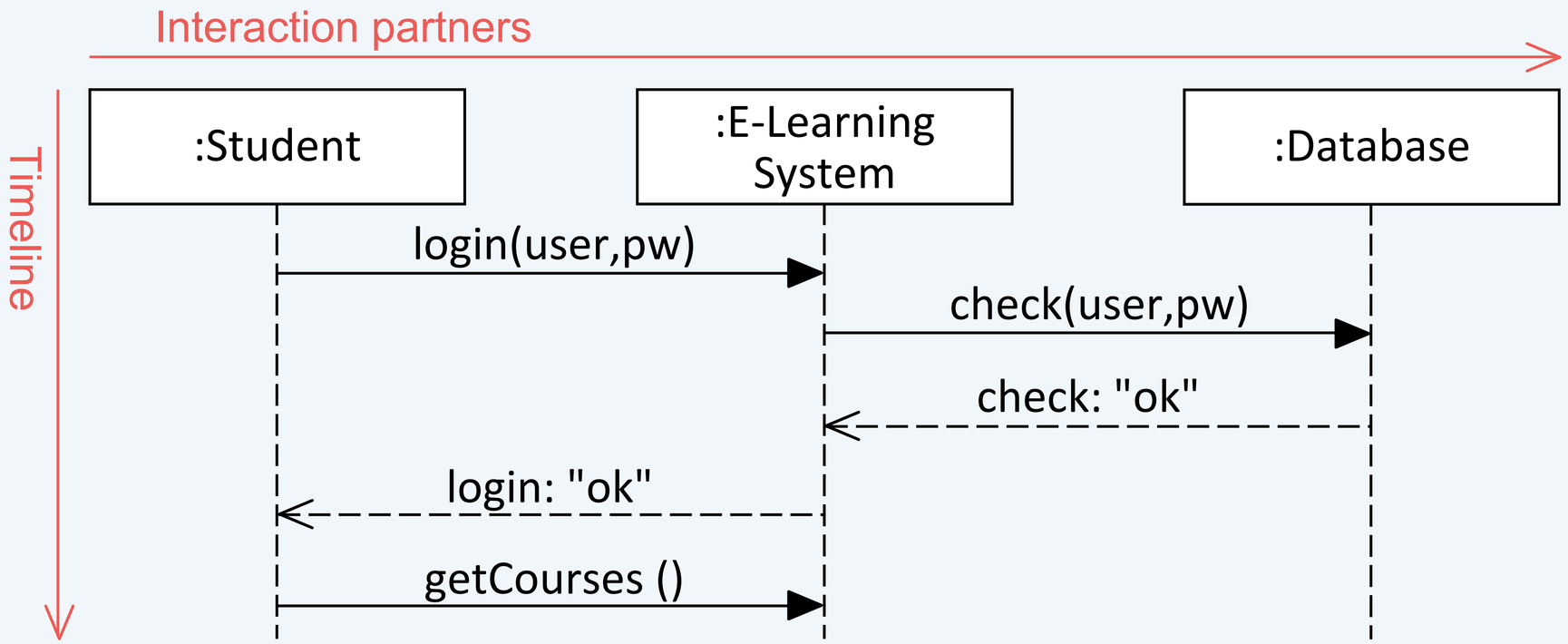
- Interaction
  - Interaction between several communication partners
  - Message and data exchange
- Interactions through
  - Signals
  - Operation calls
    - Calling an operation of a class
    - Response: Result of the called operation
- Control of interactions through
  - Conditions
  - Time events

Interaction Diagrams

- Show how messages are exchanged between different communication partners in a specific context
- Description of communication situations through:
  - Communication partners + their lifelines
  - Interactions
  - Messages
  - Means for flow control
- Four types of interaction diagrams
  - All four are based on similar concepts
  - Different requirements and emphasis on different aspects
  - Semantically equivalent for simple interactions, but different focus

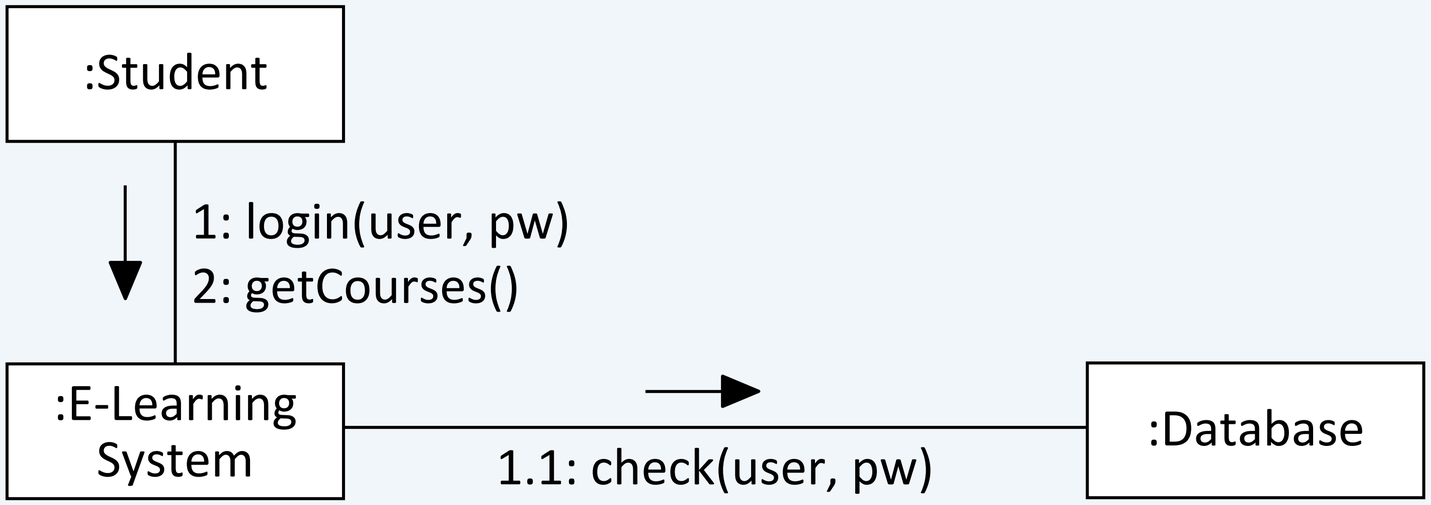
Interaction Diagrams - Types (1/4)

- **Sequence diagram** shows the temporal and logical message flow
  - Time is a separate dimension



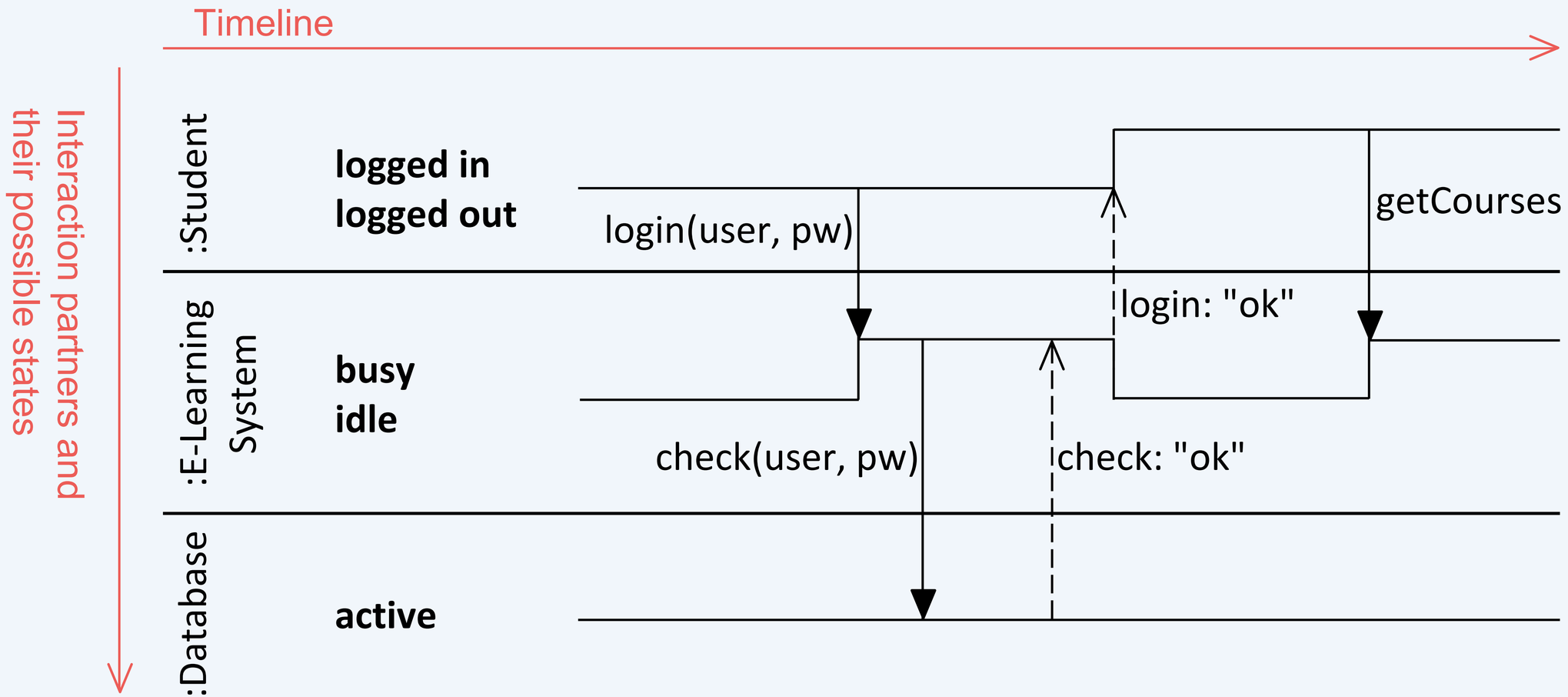
Interaction Diagrams - Types (2/4)

- **Communication diagram** is "structurally" oriented
  - Shows the relationships between interaction partners
  - Focus: Who communicates with whom
  - Time is not a separate dimension
  - Sequence of messages only expressed via decimal classification



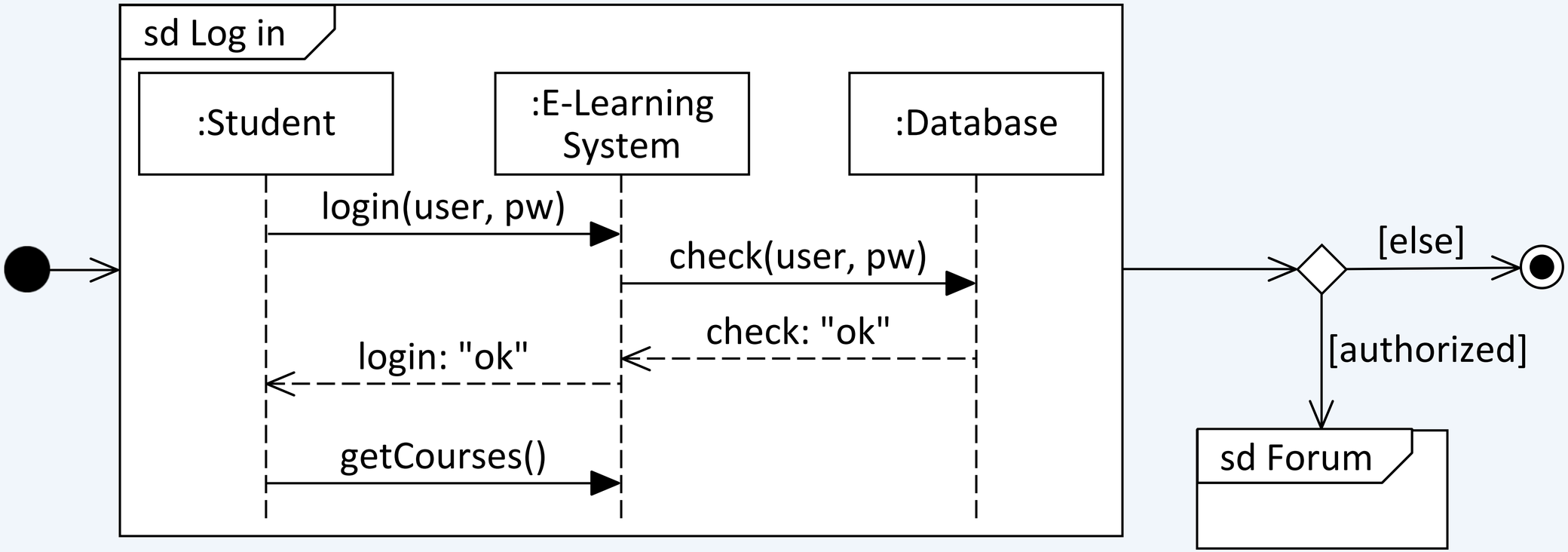
Interaction Diagrams - Types (3/4)

- **Time diagram** shows state changes of the interaction partners due to interactions or time events



Interaction Diagrams - Types (4/4)

- **Interaction overview diagram**
  - shows the interplay of different interactions
  - Visualizes in which order and under which conditions interaction processes take place
  - Notation elements from the activity diagram



Areas of Application

Modeling…

- the **interactions of a system with its environment** (defining system boundaries, system as a black box)
- the **realization of a use case**
- the **interaction of the internal structure** of a class, component or collaboration
- the **operations of the classes**
- the specification of **interfaces between system components** (interaction of provided/used interfaces)

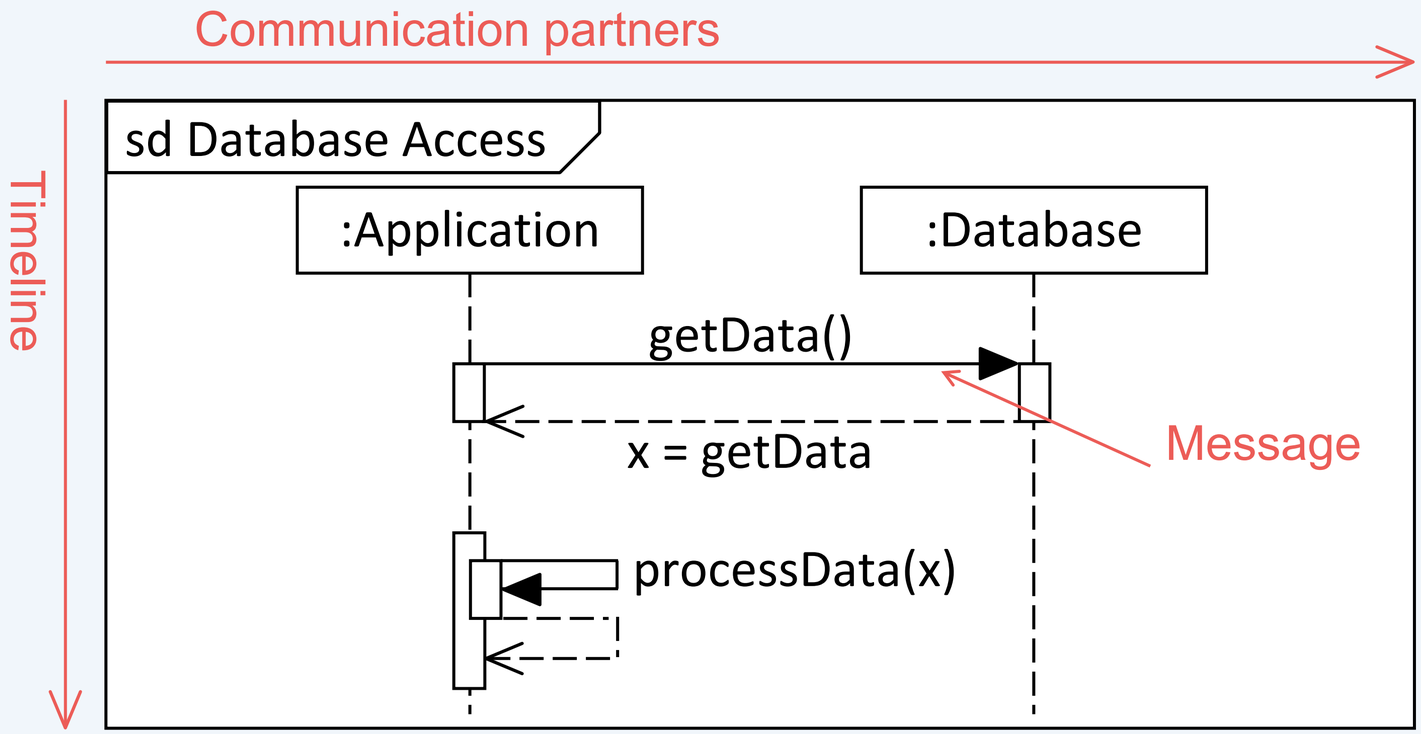
Sequence Diagram
The Lifeline

Christian Huemer and Marion Scholz

Presented by Nicholas Bzowski

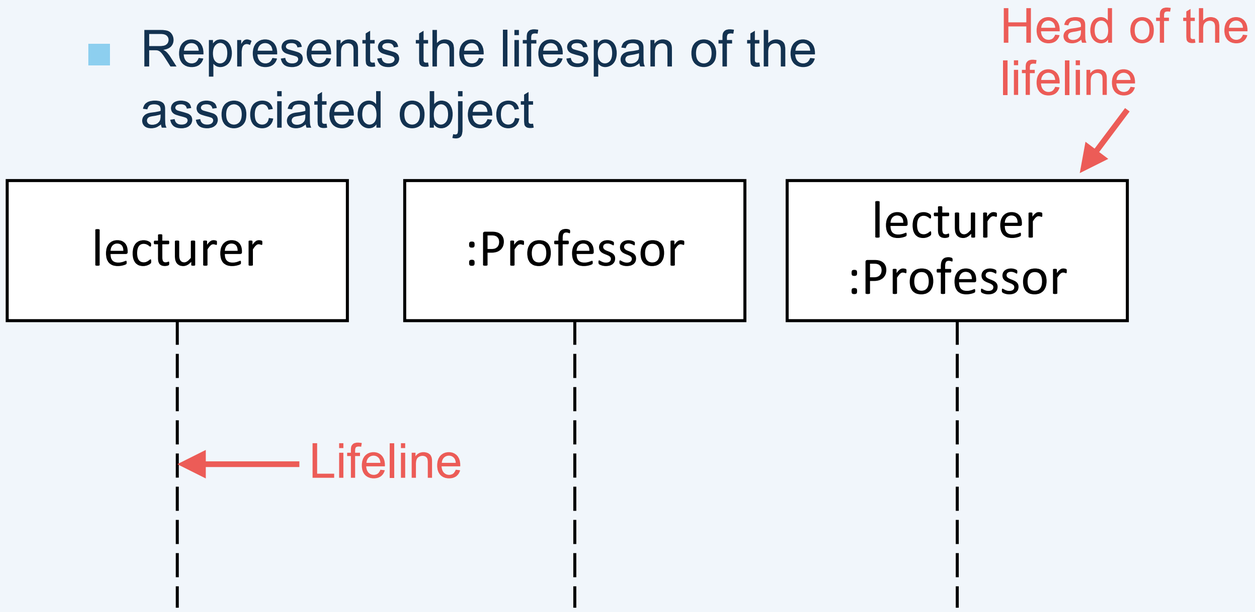
Sequence Diagram

- Representation of interactions in **2 dimensions**:
  - **horizontal**: Communication partners in the form of roles; order is not important
  - **vertical**: Timeline
    Represents the temporal sequence of communication



Lifeline

- A **lifeline** describes **exactly one communication partner**
- Head of the lifeline
  - Rectangle with **`roleName:Class`**
  - Object can take on different roles during its lifetime
- Body of the lifeline
  - Vertical, dashed line
  - Represents the lifespan of the associated object

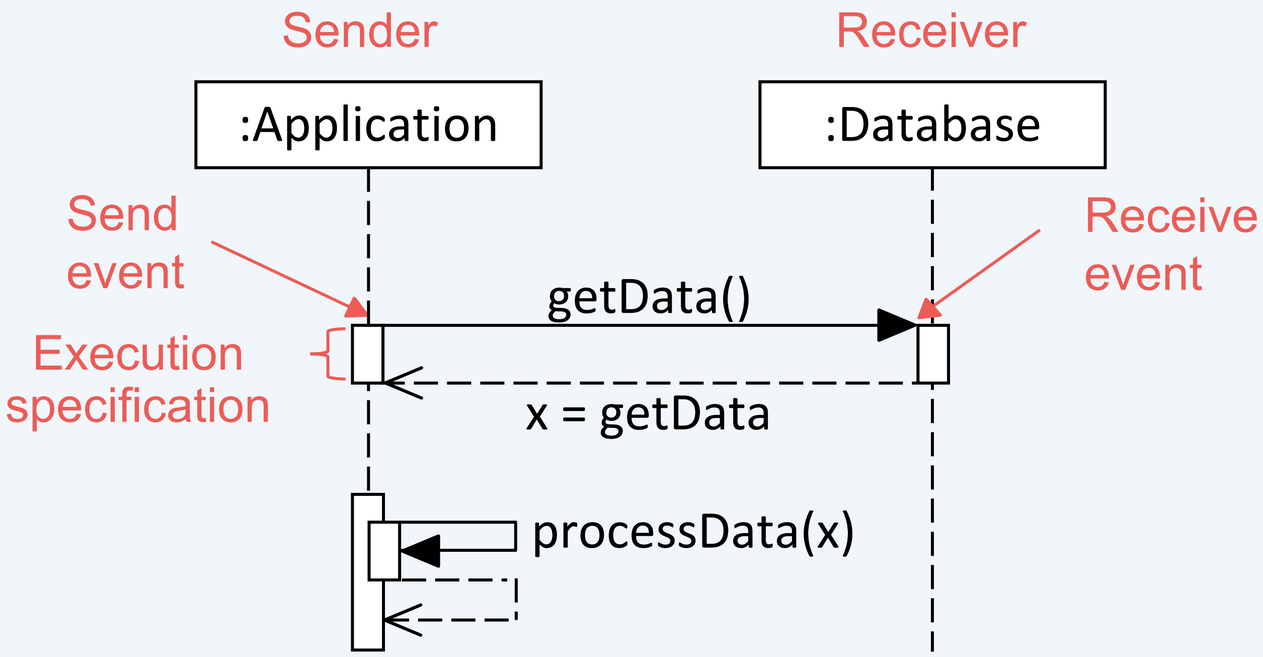


Type vs. Instance Level

- Modeling of message exchange always at **instance level**
  - Roles are representative objects for the context to be modeled
  - Actual interaction takes place between objects at instance level
- Trace
  - Sequence of messages between concrete objects

Lifeline: Design Specification (1/2)

- Interactions are the **result of event specifications** on the lifelines
- Example of event specifications
  - **Sending** and **receiving** messages on different lifelines or the same lifeline



Lifeline: Design Specification (2/2)

Sequence of event specifications

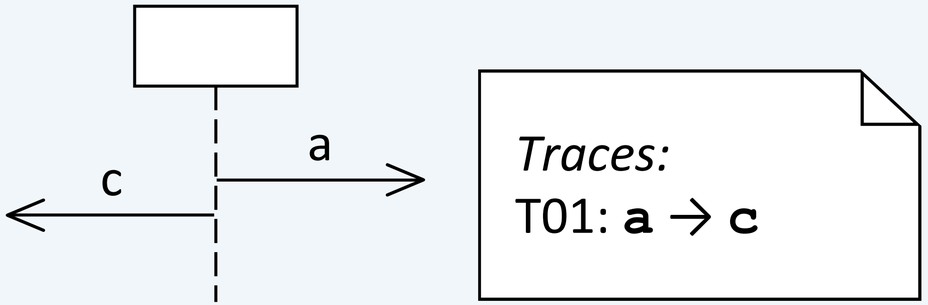
- Vertical time axis determines the order of event occurrences per lifeline

  Does not determine the sequence of event occurrences on different lifelines

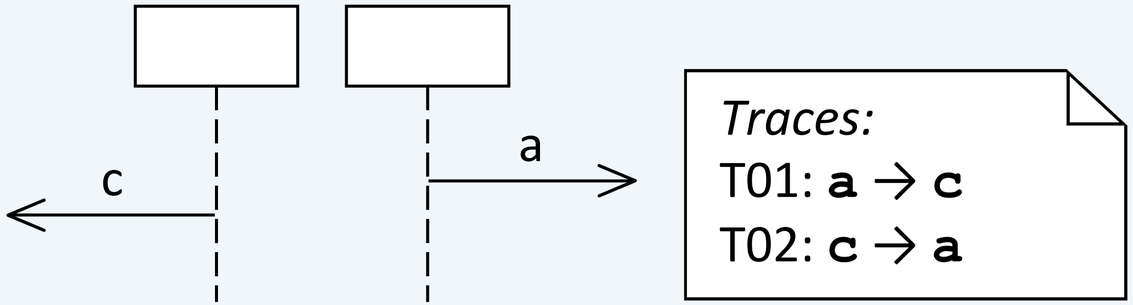
- **Messages between lifelines enforce order** across lifelines

Lifeline: Sequence of event occurrences

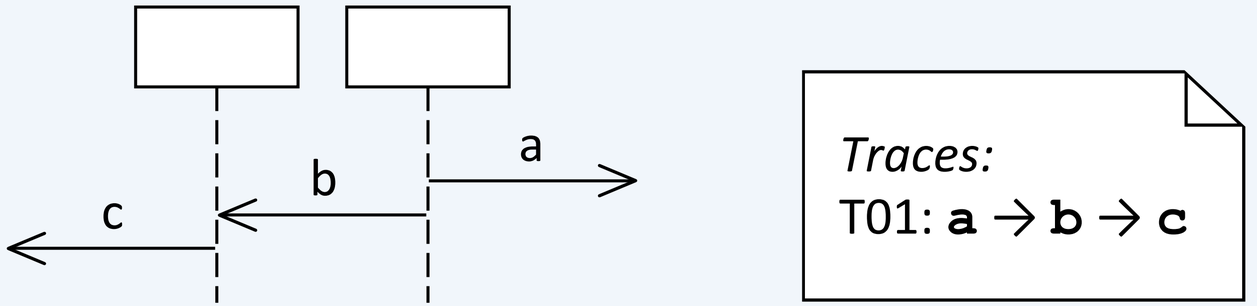
...on a lifeline



... on different lifelines

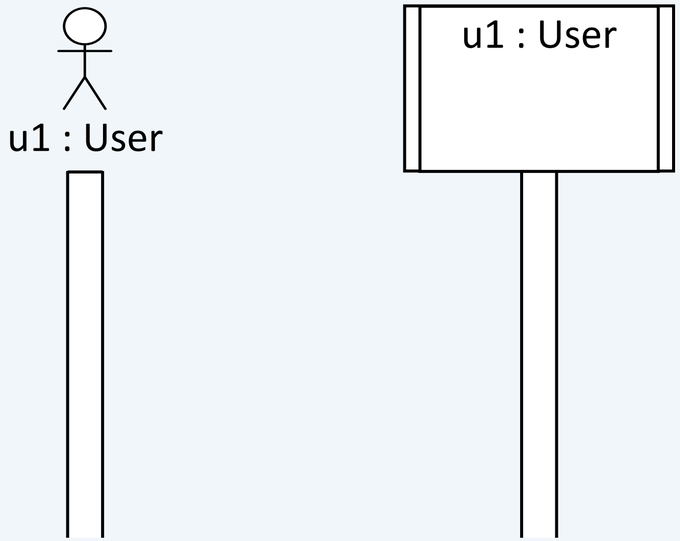


... on different lifelines,
connected by the exchange of messages



Lifeline: Active object

- **Active** objects have their **own control flow** (process or thread)
- Can operate **independently** of other objects
- Notation
  - The head of the lifeline is provided with a double border on the left and right
  - Continuous bar across the entire lifeline



Sequence Diagram

The Message

Christian Huemer and Marion Scholz

Presented by Nicholas Bzowski

Types of Messages

- **Synchronous communication**
  - Sender waits until the end of the interaction triggered by the message

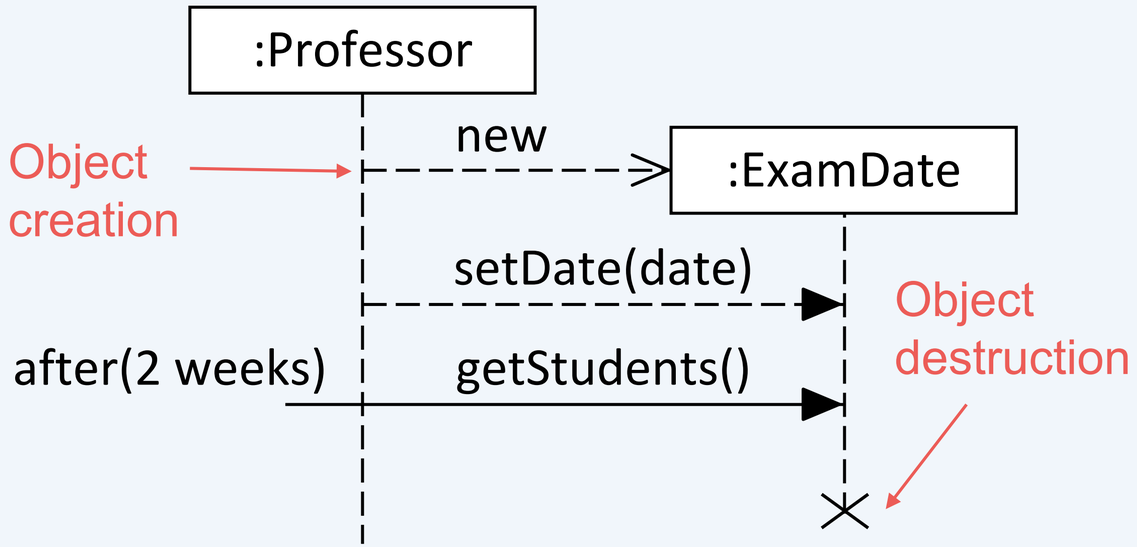
- **Asynchronous communication**
  - Message is a signal
  - Sender does not wait for the end of the interaction

- **Response message (optional)**
  - Syntax:
    `att=msg(par1,par2):val`

`att`: Name of an attribute to which the return value is to be assigned
`msg`: Name of the message to which you are replying
`par`: Parameter list
`val`: Return value

Message: Special message types (1/2)

- Object creation
  - Communication partner is created in the course of the interaction sequence
  - Keyword **`new`**
- Object destruction
  - Object is deleted
  - Large "X" at the end of the lifeline



Message: Special message types (2/2)

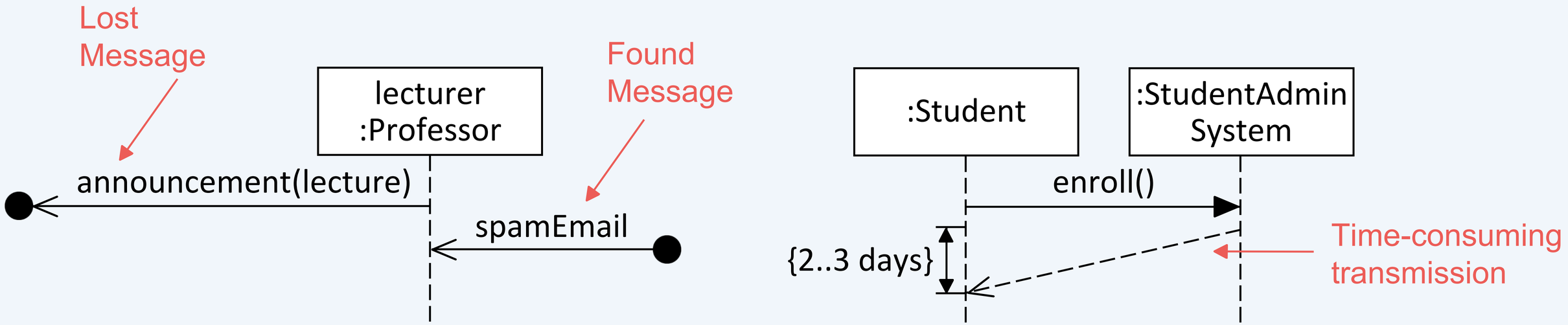
- Lost message

  Sending a message to an unknown/unknown or irrelevant/irrelevant communication partner

- Found message

  Receiving a message from an unknown/unknown or irrelevant/irrelevant communication partner

- Time-consuming transmission

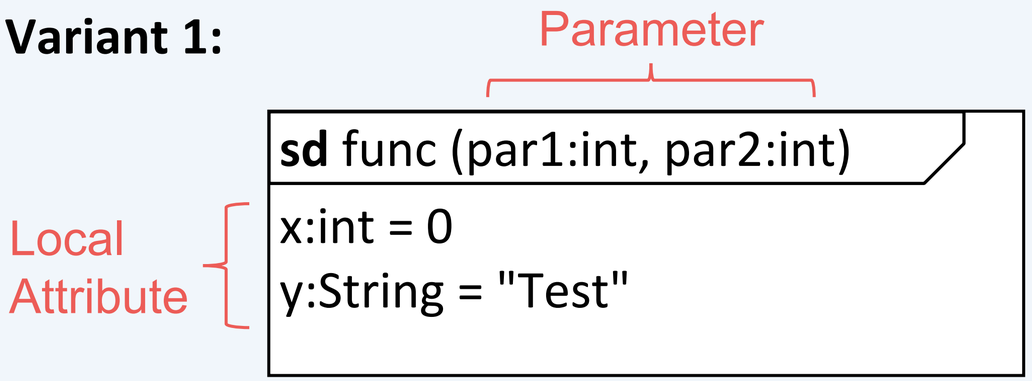


Basic concepts - parameters, local attributes

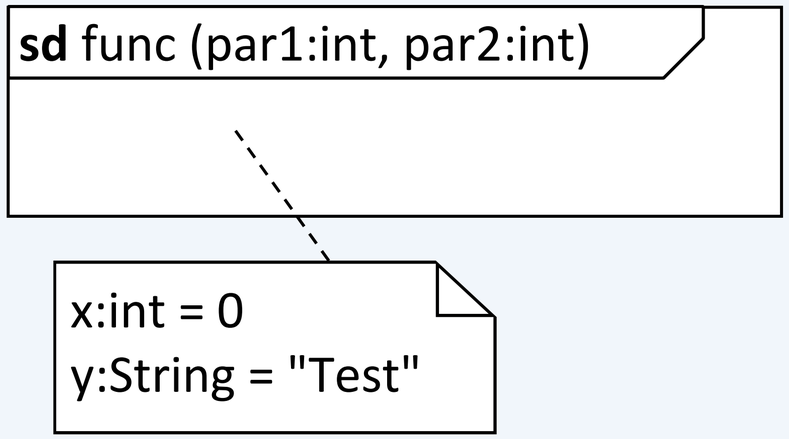
- Representation of parameters and local attributes
- Example: Modeling the operation **`func`**

```
void func (int par1, int par2) {
  int x = 0;
  String y = "Test";
  ...
}
```

Variant 1:



Variant 2:



Sequence Diagram

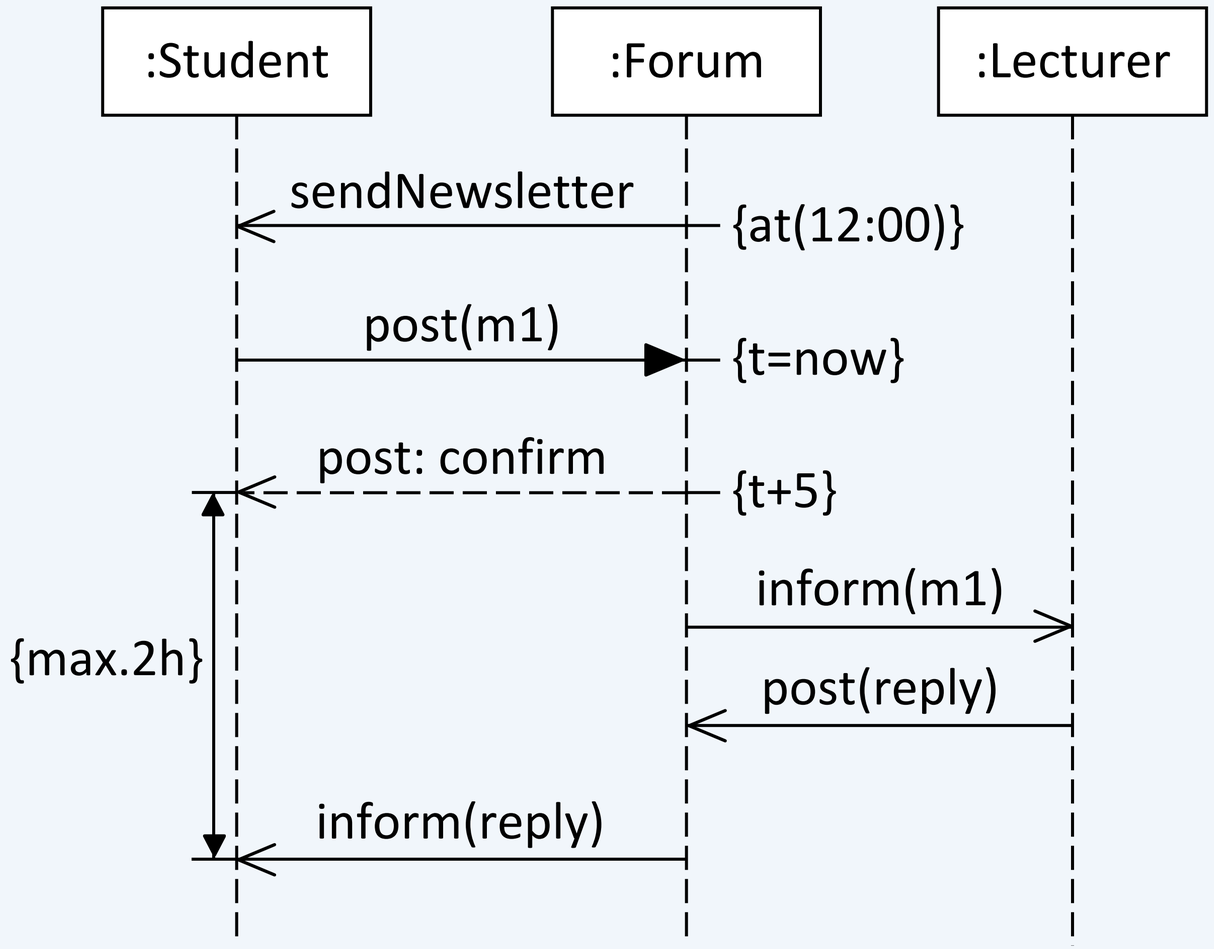
The Time Constraint and The State Invariant

Christian Huemer and Marion Scholz

Presented by Nicholas Bzowski

Time Constraints

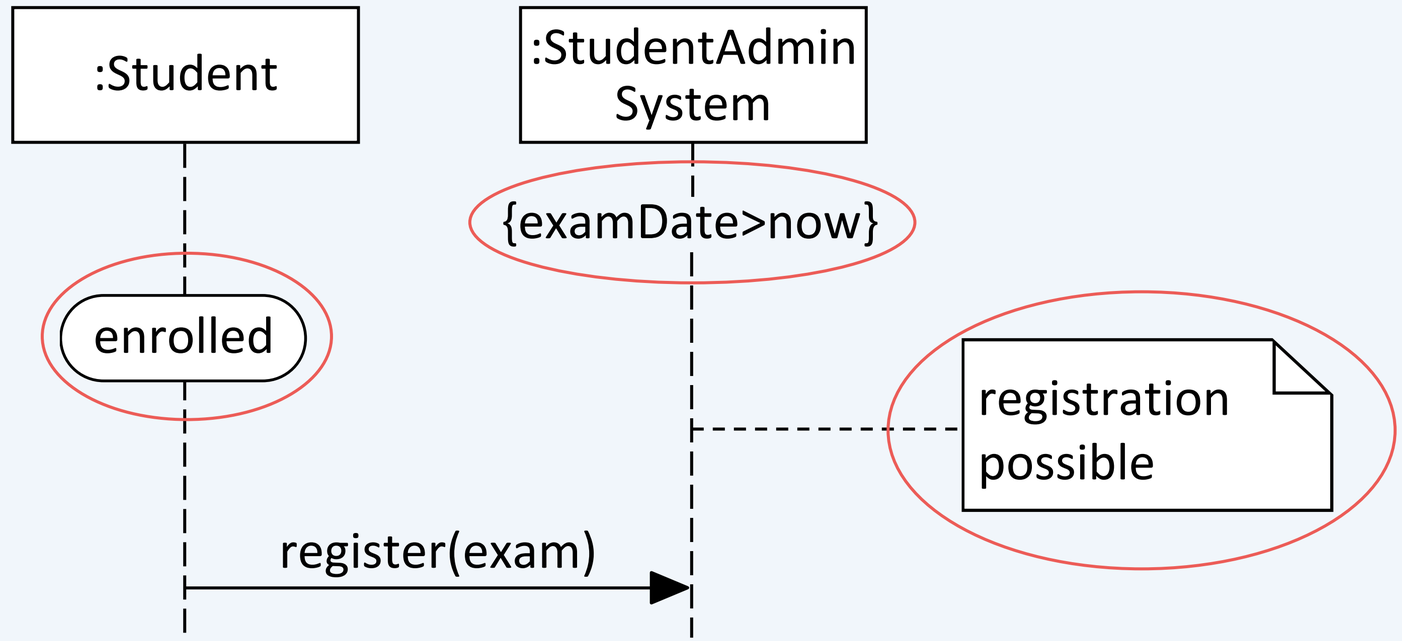
- Types
  - **Point in time** of an event specification
    - absolute: e.g. **`at(12:00)`**
    - relative: e.g. **`after(5sec)`**
  - **Time interval** between two event occurrences
    - e.g. **`{12:00 .. 13:00}`**
- Predefined actions for time calculation
  - **`now`**: current time
  - **`duration`**: Calculation of a message transmission period
  - Values received must be assigned to variables
  - Variables can be used in time expressions



State Invariant

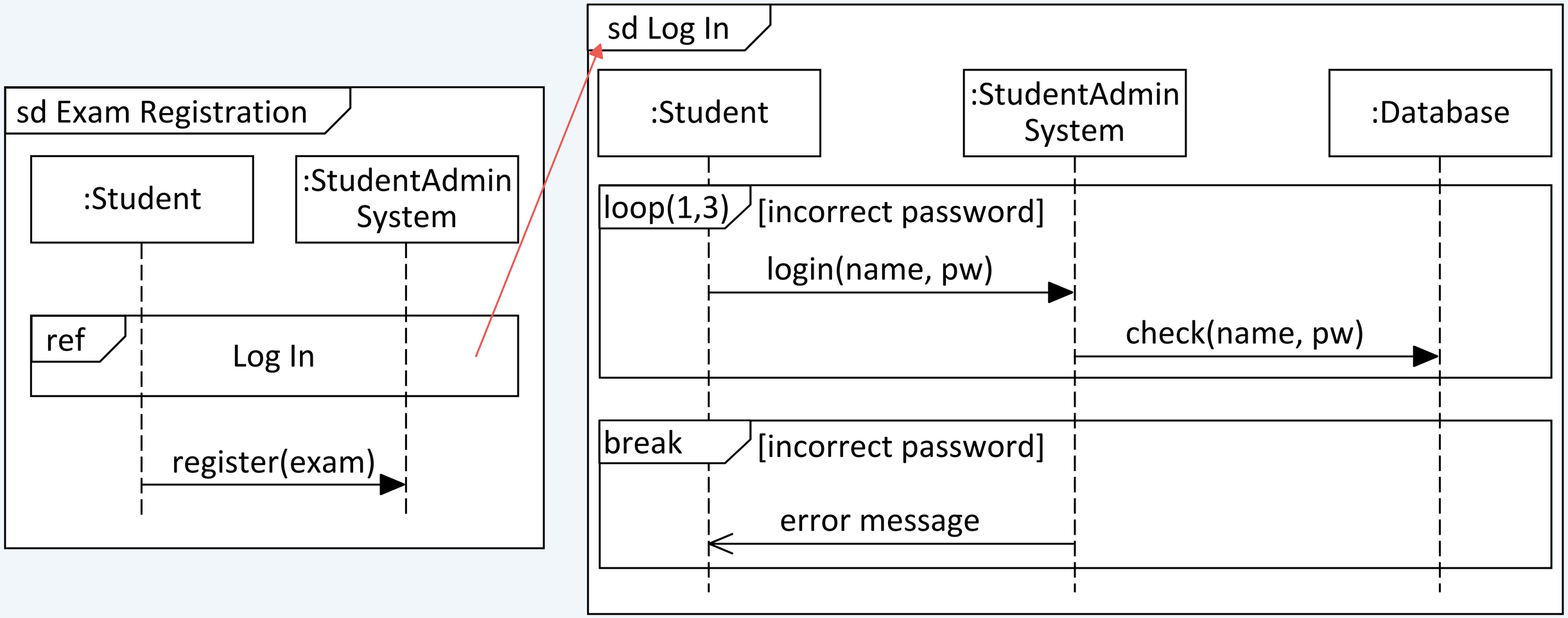
- **Assurance** that a certain **condition** will be **fulfilled** at a certain point in time
- Always refers to a specific lifeline
- Evaluated before the subsequent event occurs
- If state invariant is not fulfilled ⇒ Error

- Three notation variants:



Interaction Reference

- The **`Log In`** sequence diagram is embedded into the **`Exam Registration`** sequence diagram via an interaction reference



Sequence Diagram

The Combined Fragments, Part 1

Christian Huemer and Marion Scholz

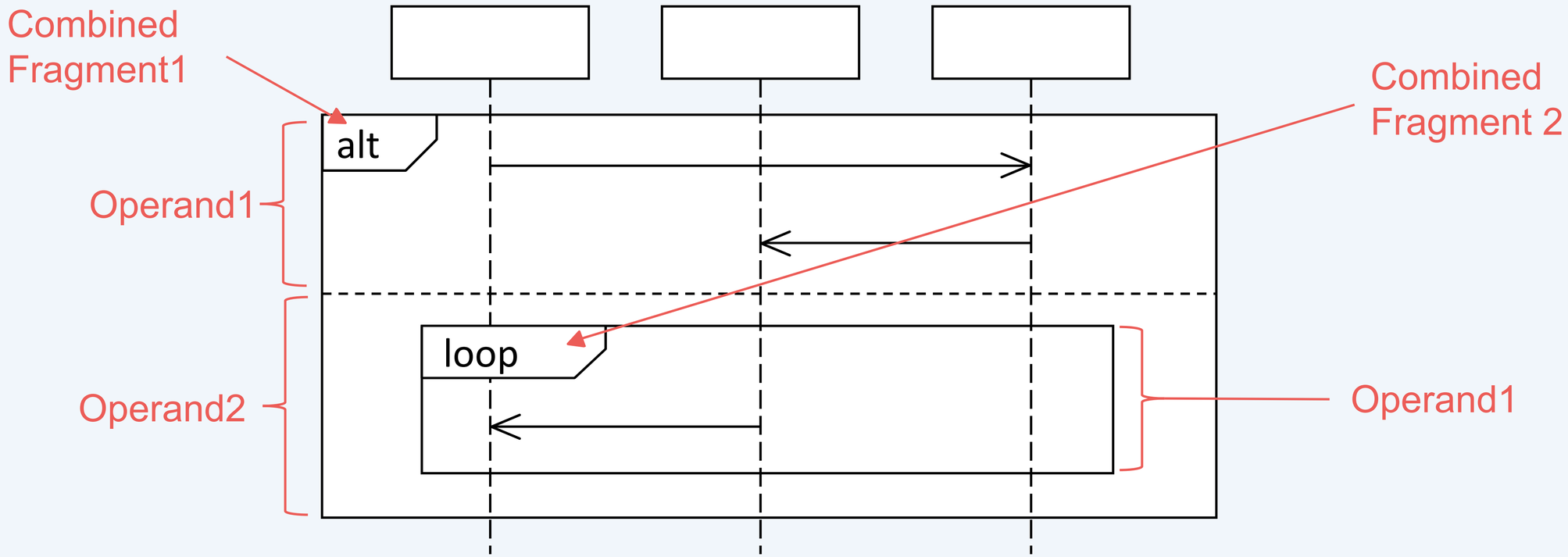
Presented by Nicholas Bzowski

Combined Fragments

- Used for modeling **control structures**
- Components: operator and operands
- **Operator**
  - Defines type of combined fragment
  - 12 predefined operators
- **Operand**
  - An operator contains 1 or more operands, depending on the operator type
  - Includes interactions, combined fragments and references to sequence diagrams

Combined Fragments – Notation

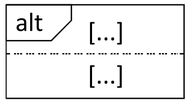
- Combined fragment is represented with a frame
- Type of fragment is determined by the operator in the Pentagon
  - default: **`seq`**
- Operands are separated from each other by dashed lines
- 12 predefined types of operators



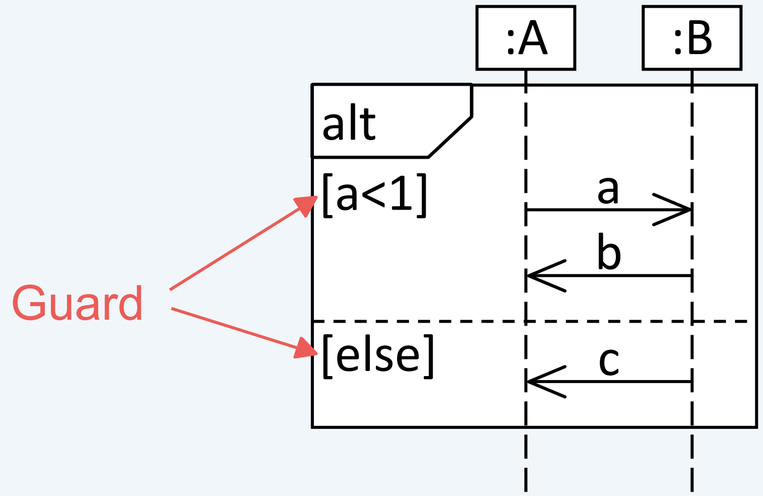
Combined Fragments – Operator Types

	Operator	Purpose
Branches and loops	`alt`	Alternative interactions
	`opt`	Optional interactions
	`break`	Exception Interactions
	`loop`	Iterative interactions
Concurrency and order	`seq`	Sequential interactions with weak order (default)
	`strict`	Sequential interactions with strict order
	`par`	Concurrent interactions
	`critical`	Atomic interactions
Filters and assertions	`ignore`	Irrelevant interactions
	`consider`	Relevant interactions
	`assert`	Asserted interactions
	`neg`	Invalid interactions

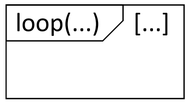
Branches and Loops: `alt` Operator



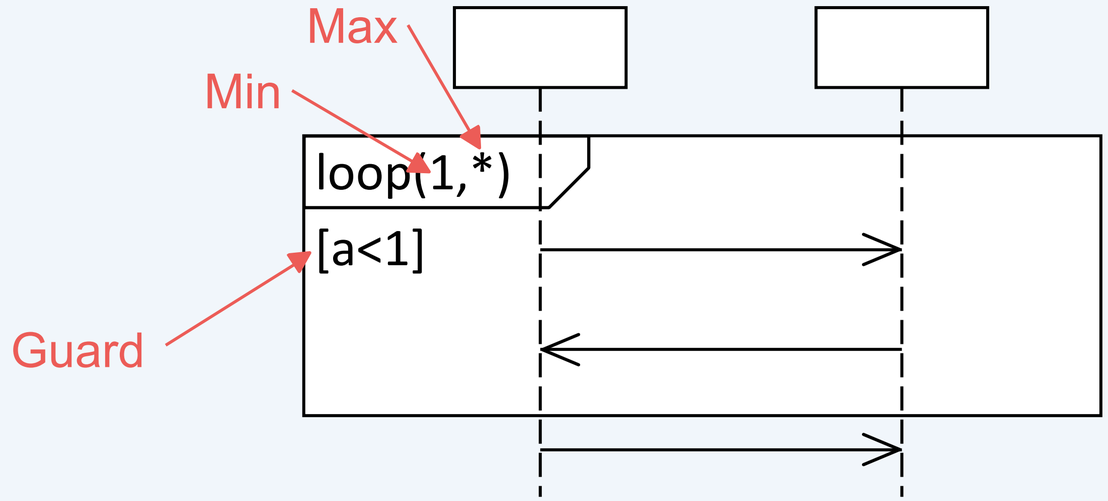
- Representation of two or more **alternative interaction sequences**
- A maximum of one operand is executed at runtime
- Selection of an operand based on guard
- Guards
  - Boolean expression in square brackets
  - Default value : **`[true]`**
  - Predefined : **`[else]`**: Operand is executed if the conditions of all other operands are not fulfilled



Branches and Loops: `loop` Operator



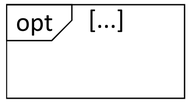
- Representation of a **loop**
  - Fragment contains only one operand
  - Execution count is represented by counters with lower and upper limits
  - Guards
    - optional
    - checked with each run as soon as the minimum number of runs has completed



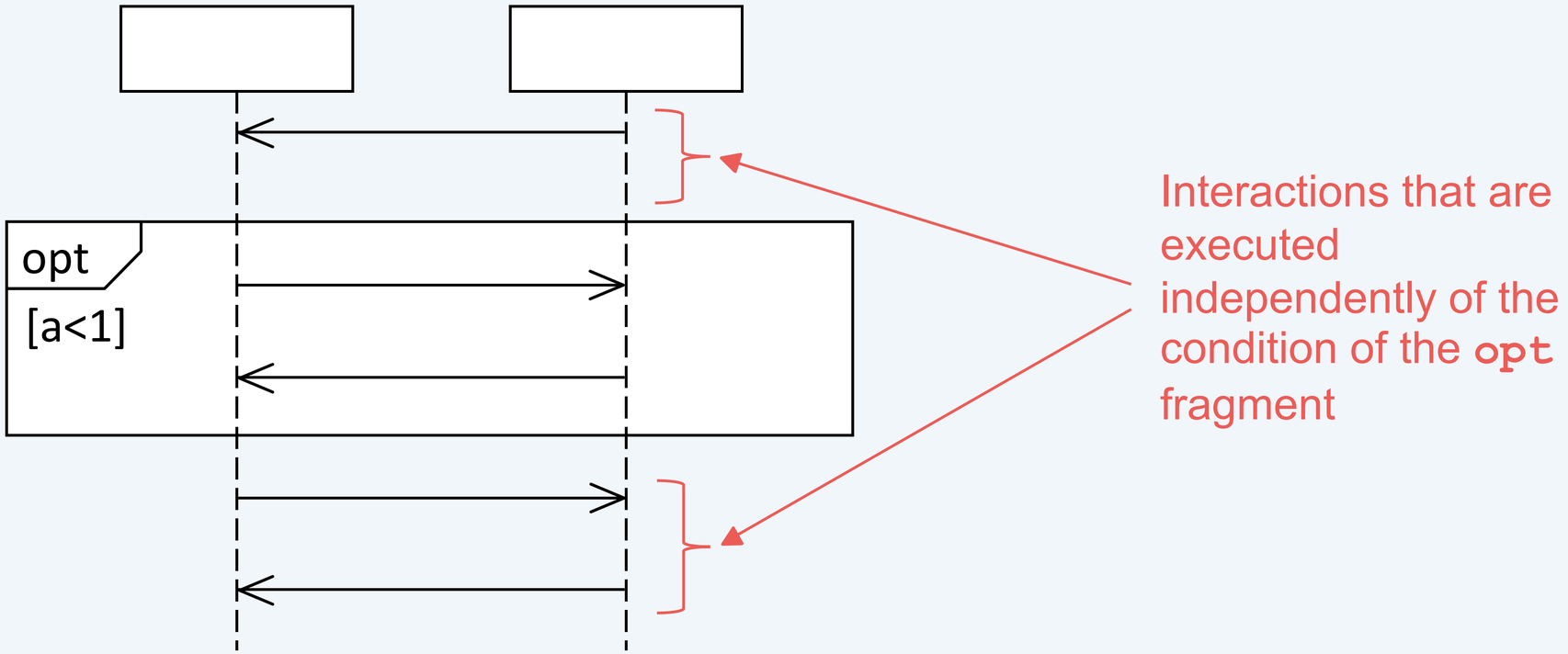
Notation variants :

```
loop(3,8) = loop(3..8)
loop(8,8) = loop (8)
loop = loop (*) = loop(0,*)
```

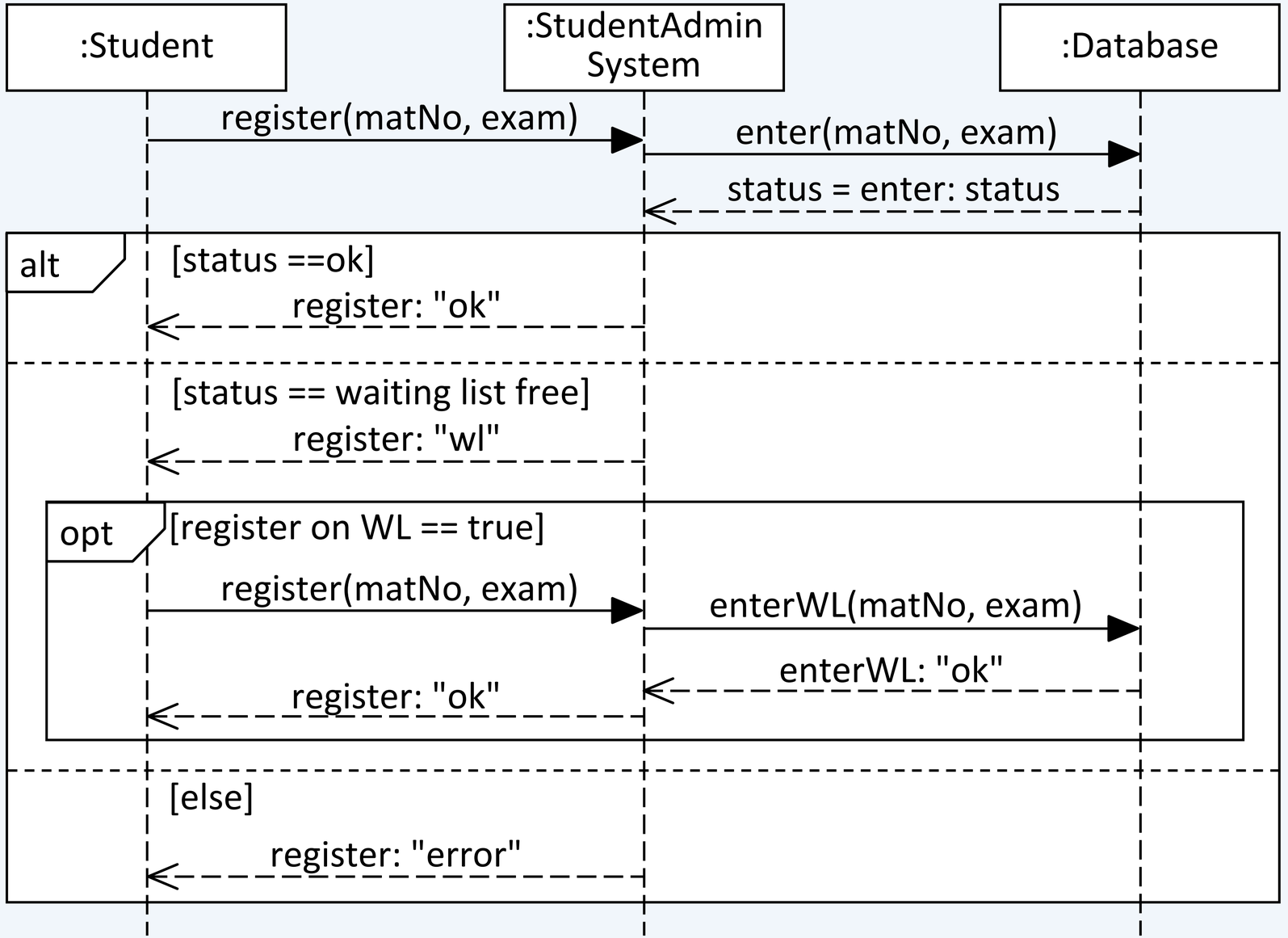
Branches and Loops: `opt` Operator



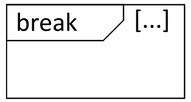
- **Optional** interactions
- Controlled by guard
- Fragment only becomes active if condition is fulfilled
  - Modeling of "if ..., then ..."



`opt` and `alt` Operator – Example

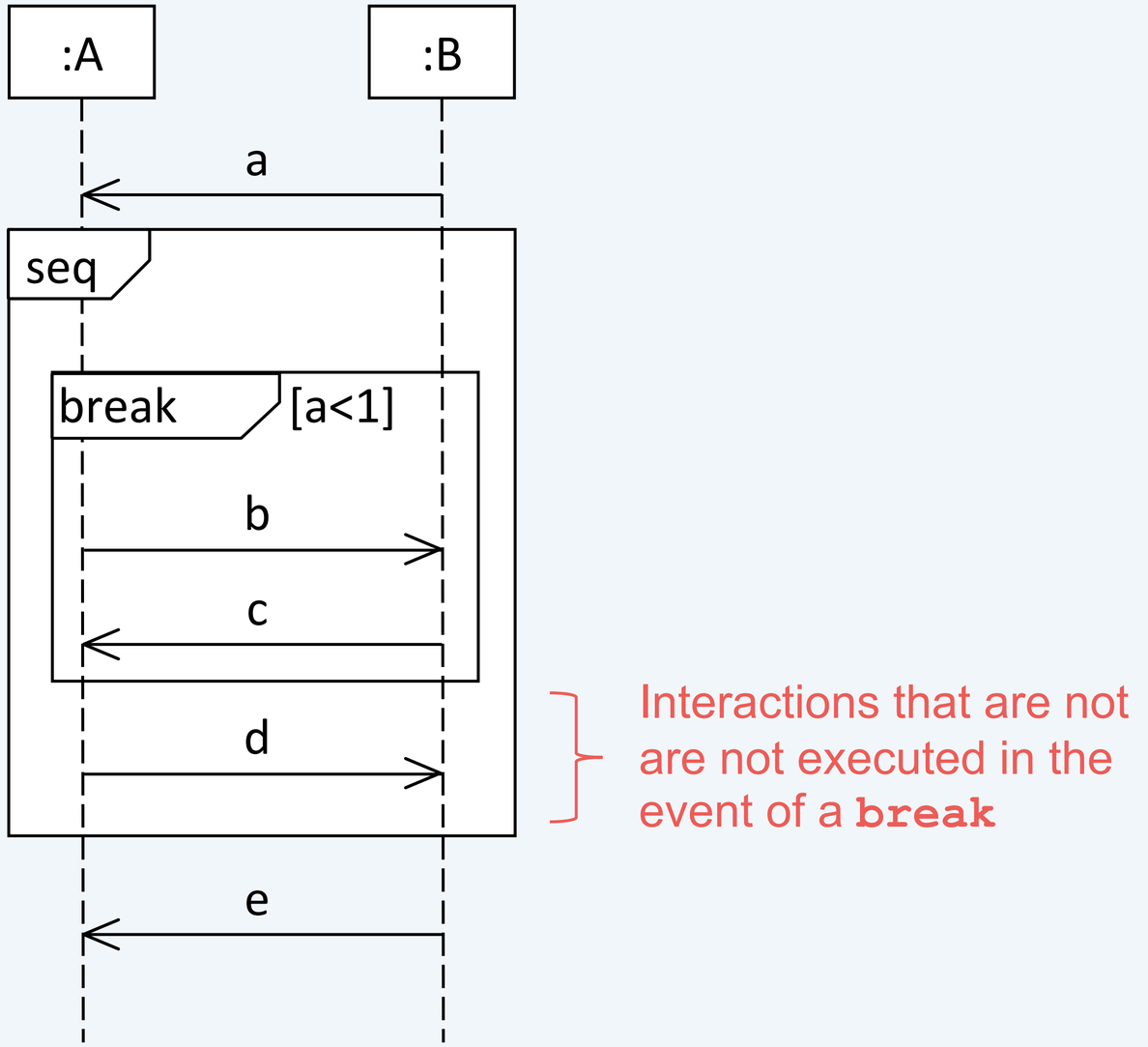


Branches and Loops: `break` Operator

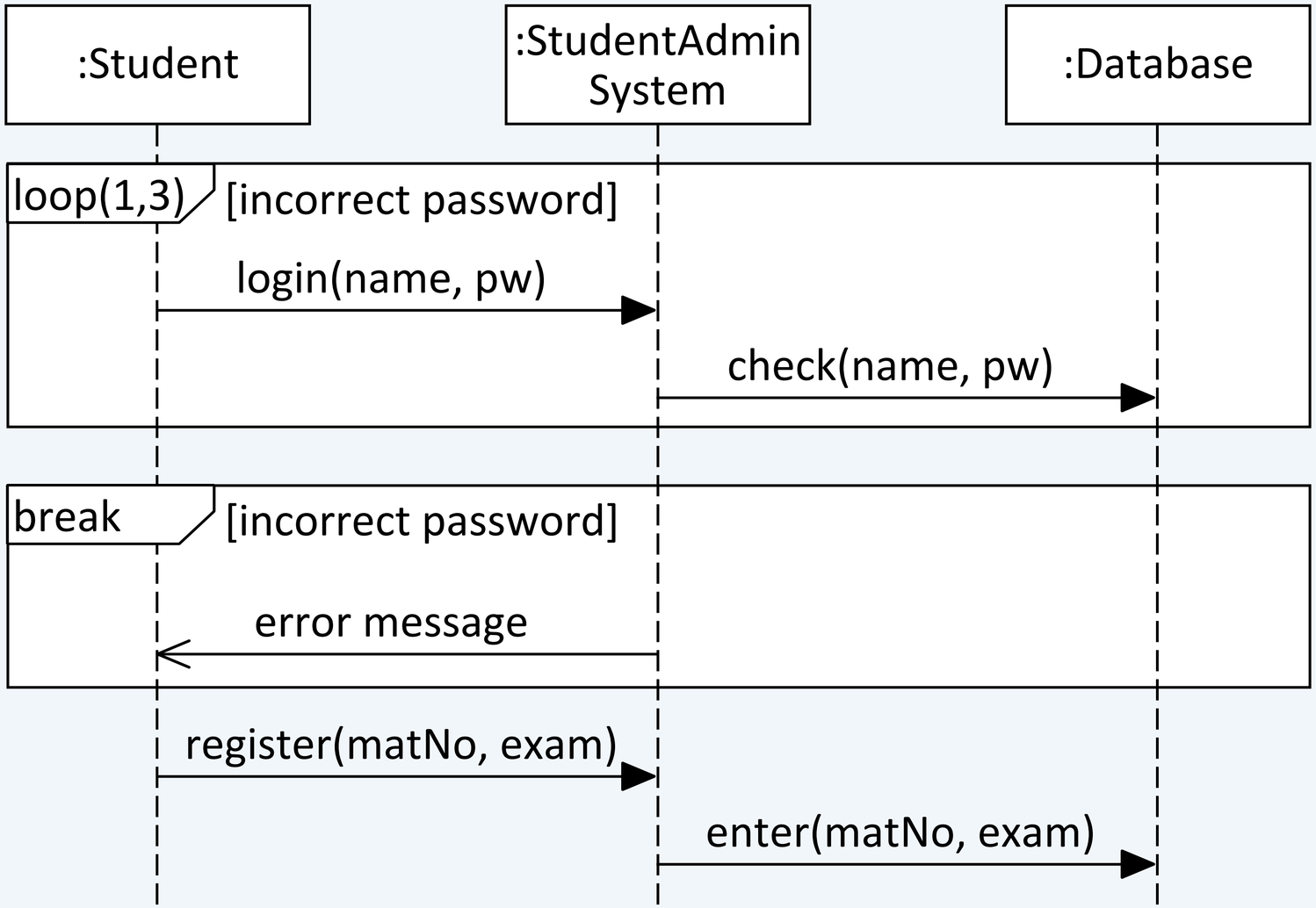


- **Exception** interactions
- Controlled by guard
- Handling of special cases and exceptions
- If the condition is true:
  - Interactions within the break are executed
  - Interactions of the surrounding fragment are discarded
  - Interactions of the higher level fragment are continued

⇒ ≠ `opt`!



`loop` and `break` Operator – Example



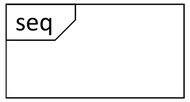
Sequence Diagram

The Combined Fragments, Part 2

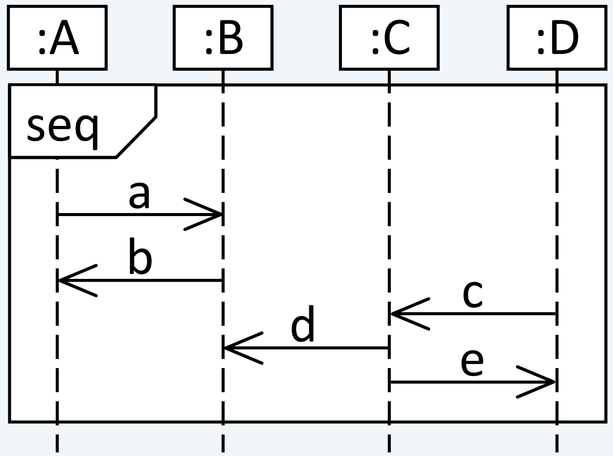
Christian Huemer and Marion Scholz

Presented by Nicholas Bzowski

Concurrency and Order: **`seq`** Operator



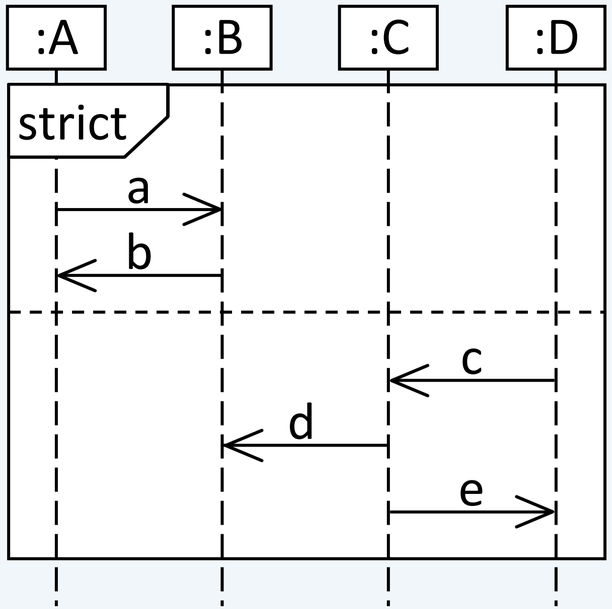
- default
- Sequential interaction with weak order
- min. 1 operand
- Sequence of event occurrences :
  - Fixed sequence of events per lifeline
  - Sequence on different lifelines only significant if a message exchange occurs
  - Operands can be used for "visual grouping", but not to force an order (≠ **`strict`**!)



Traces:
T01: **a** → **b** → **c** → **d** → **e**
T02: **a** → **c** → **b** → **d** → **e**
T03: **c** → **a** → **b** → **d** → **e**

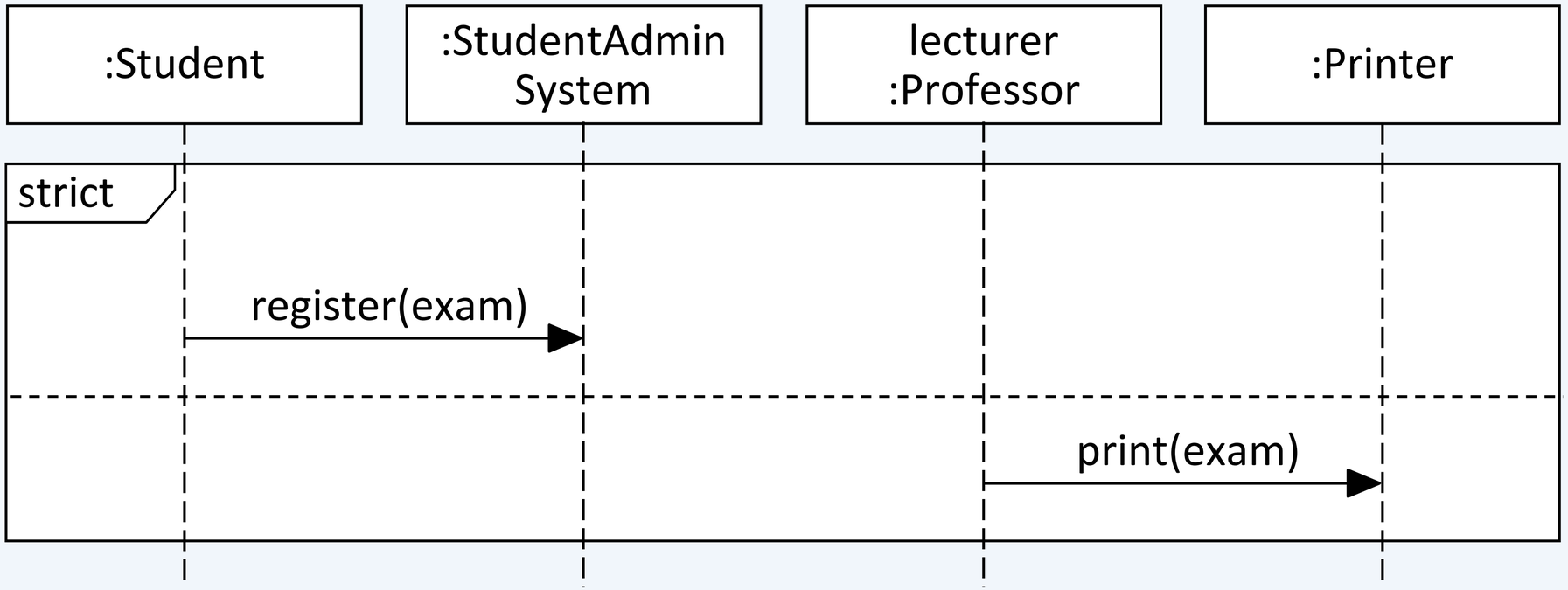
Concurrency and Order: `strict` Operator

- Sequential interaction with **strict** order
- Sequence on different lifelines of different operands is significant

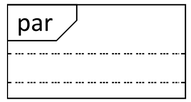


Traces:
T01: **a** → **b** → **c** → **d** → **e**

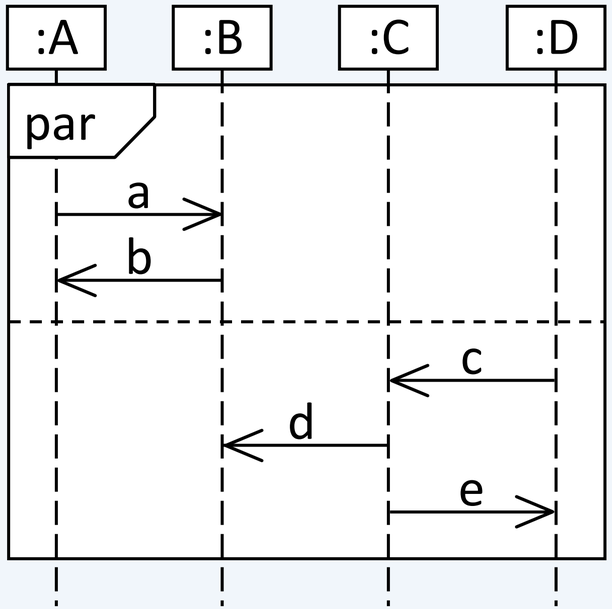
`strict` Operator – Example



Concurrency and Order: `par` Operator



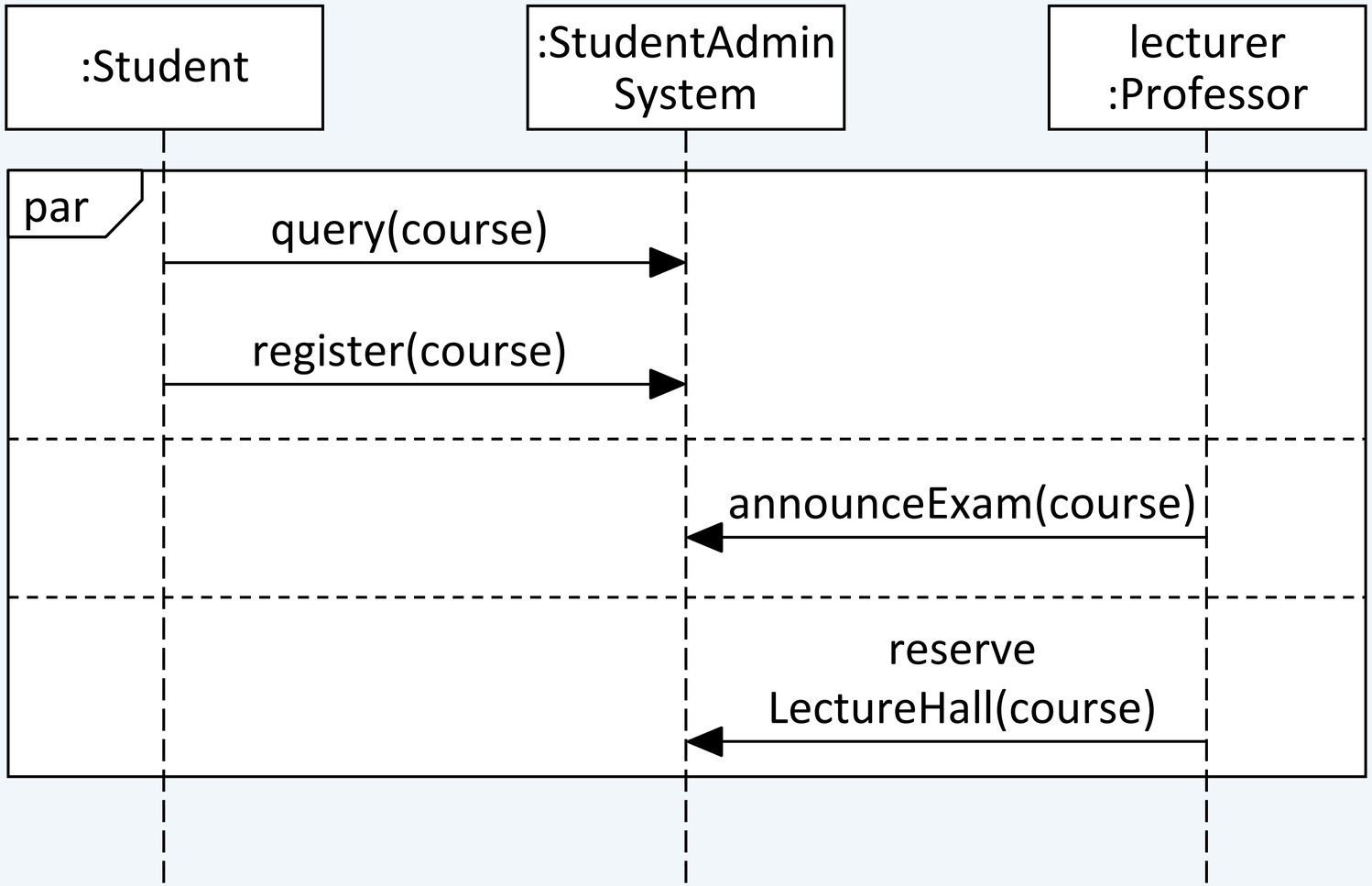
- Concurrent interactions
  - Local sequence per operand must be kept
  - The order of the operands in the diagram is irrelevant!
  - min. 2 operands



Traces:

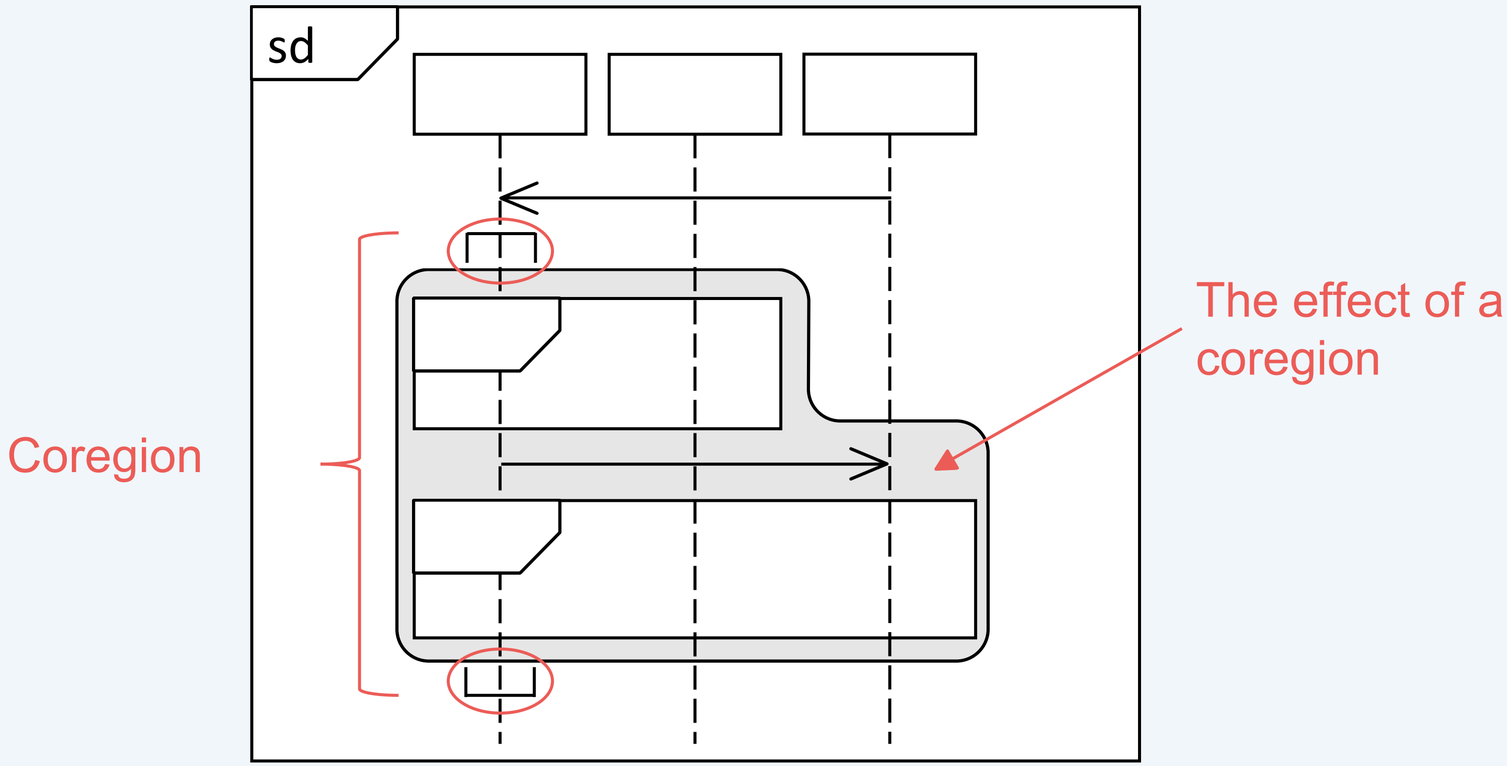
T01: **a → b → c → d → e**
T02: **a → c → b → d → e**
T03: **a → c → d → b → e**
T04: **a → c → d → e → b**
T05: **c → a → b → d → e**
T06: **c → a → d → b → e**
T07: **c → a → d → e → b**
T08: **c → d → a → b → e**
T09: **c → d → a → e → b**
T10: **c → d → e → a → b**

`par` Operator – Example

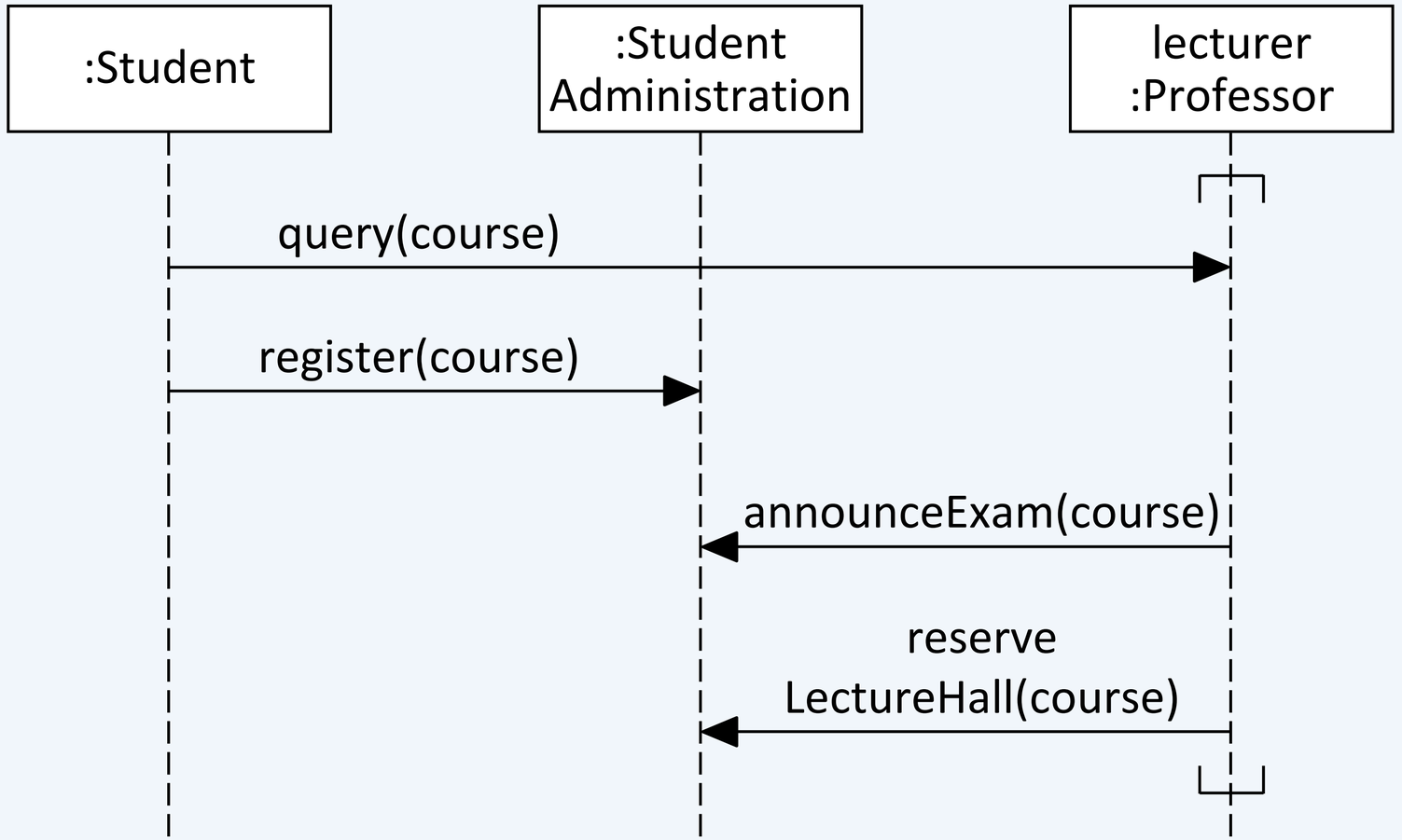


Concurrency and Order: Coregion

- Coregion: Representation of concurrent processes on ONE lifeline
- Sequence of event occurrences within coregions is not limited in any way (“suspension of the time dimension")
- Coregion can contain further combined fragments
  - combined fragments can be executed as a whole in any order



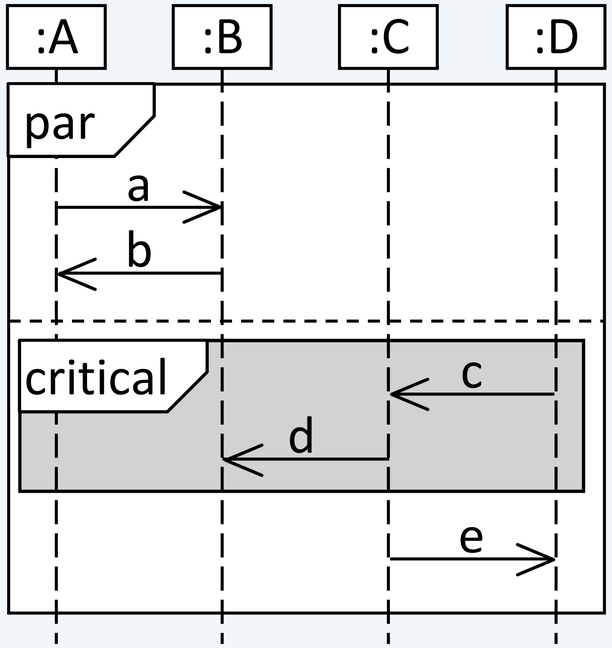
Coregion – Example



Concurrency and Order: `critical` Operator

critical

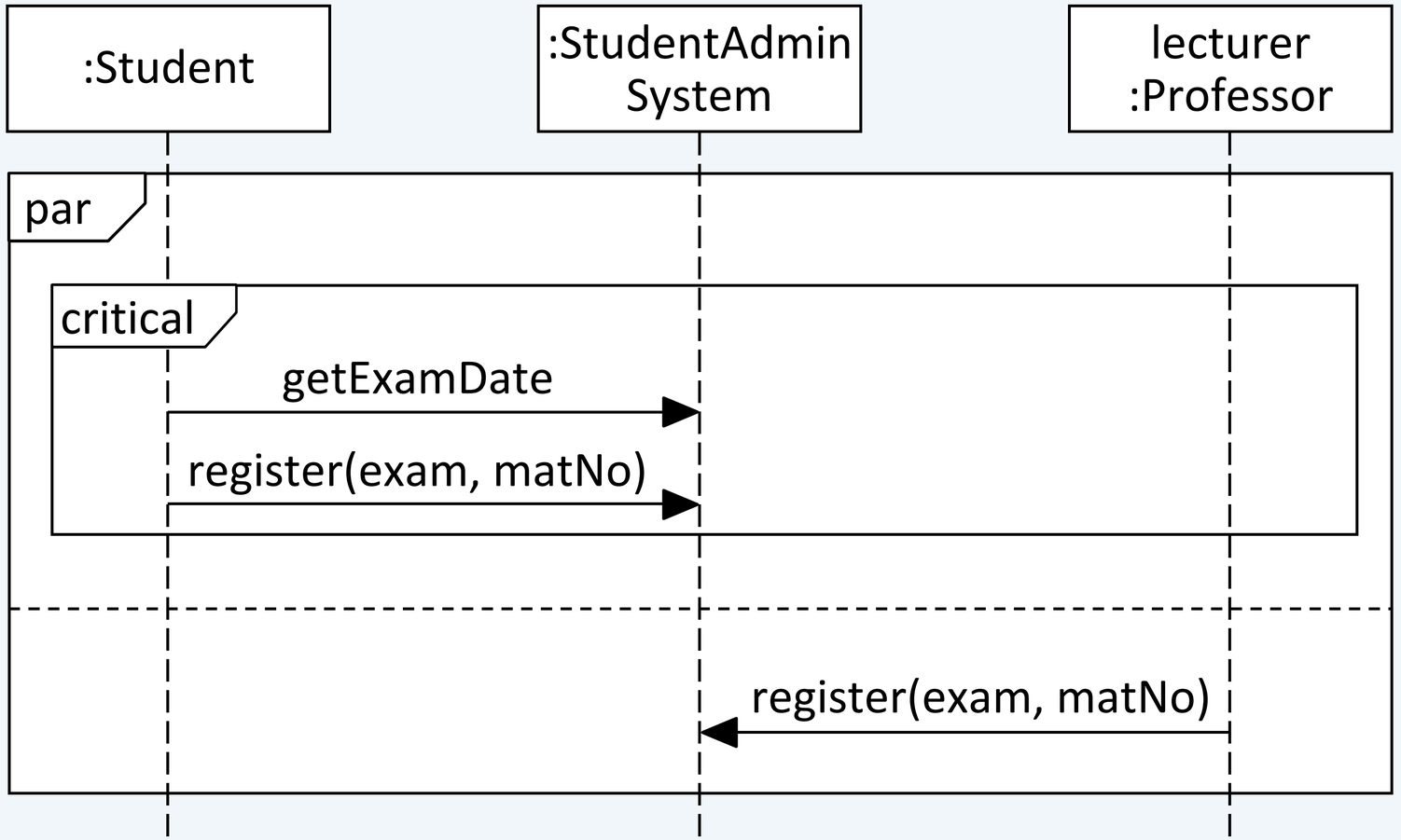
- Critical area: atomic (non-interruptible) interaction process
- No restriction on interactions outside the critical area



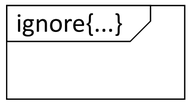
Traces:

T01: **a** → **b** → **c** → **d** → **e**
T02: **a** → **c** → **d** → **b** → **e**
T03: **a** → **c** → **d** → **e** → **b**
T04: **c** → **d** → **a** → **b** → **e**
T05: **c** → **d** → **a** → **e** → **b**
T06: **c** → **d** → **e** → **a** → **b**

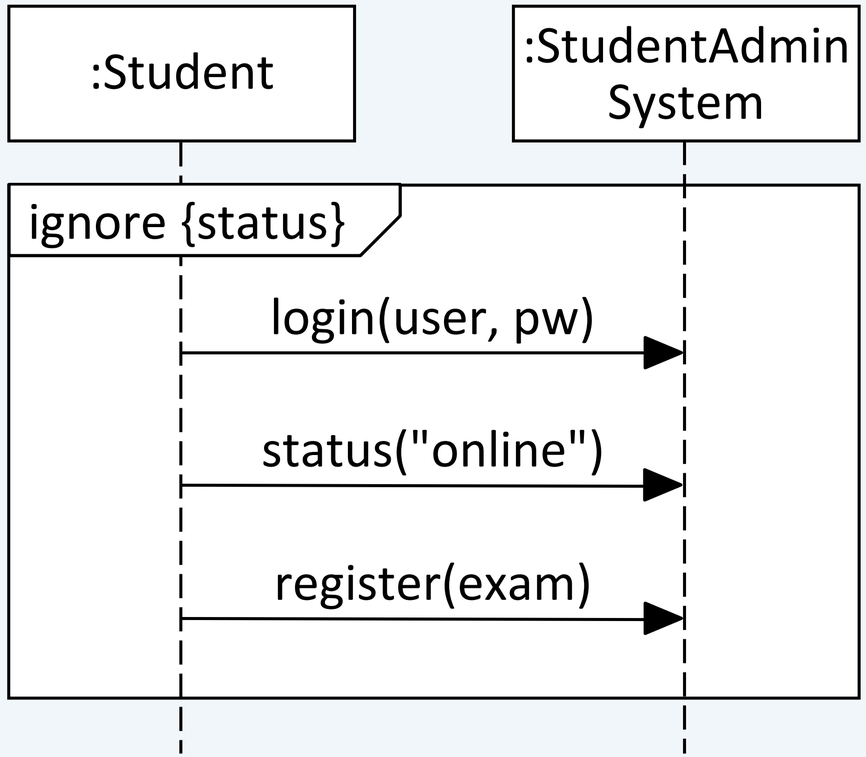
`critical` Operator – Example



Concurrency and Order: `ignore` Operator



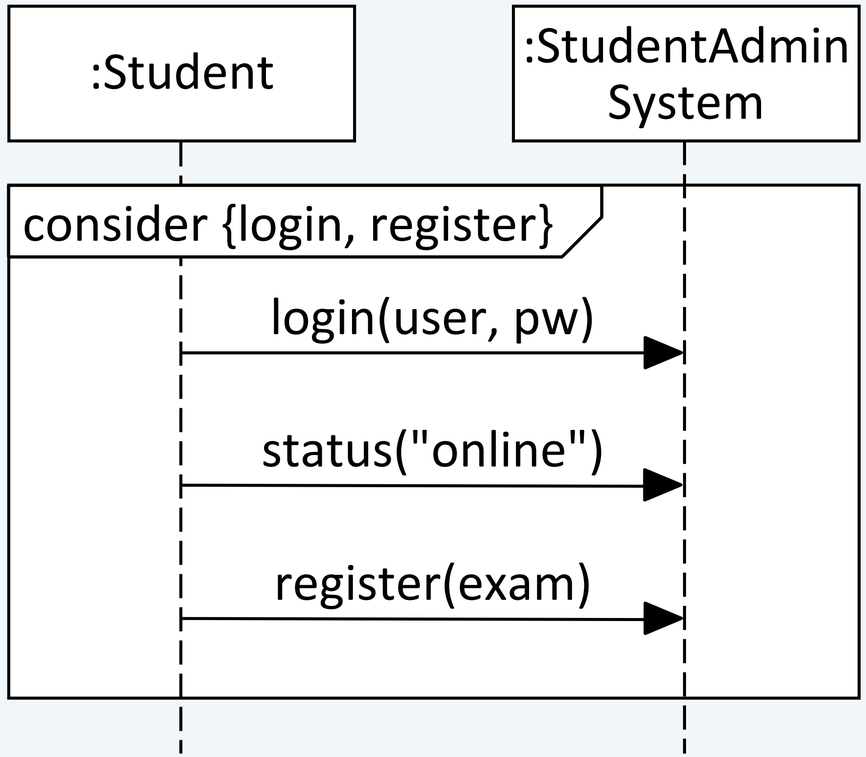
- Denotes irrelevant messages
  - Modeled for technical reasons
  - Modeled for syntactic completeness
  - Messages that can occur at runtime (e.g. keep-alive signals)



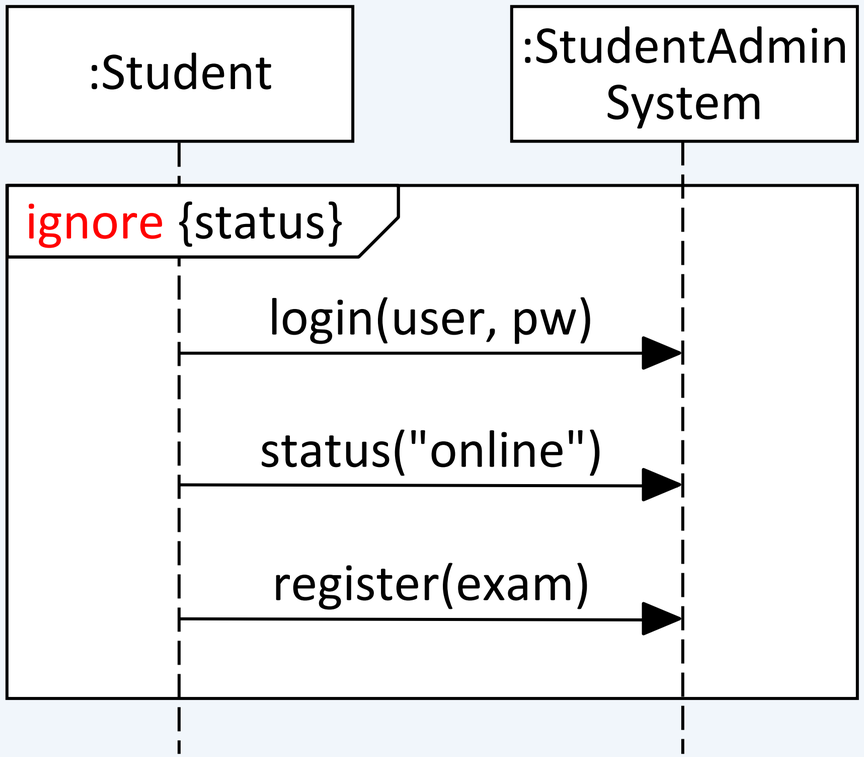
Concurrency and Order: `consider` Operator

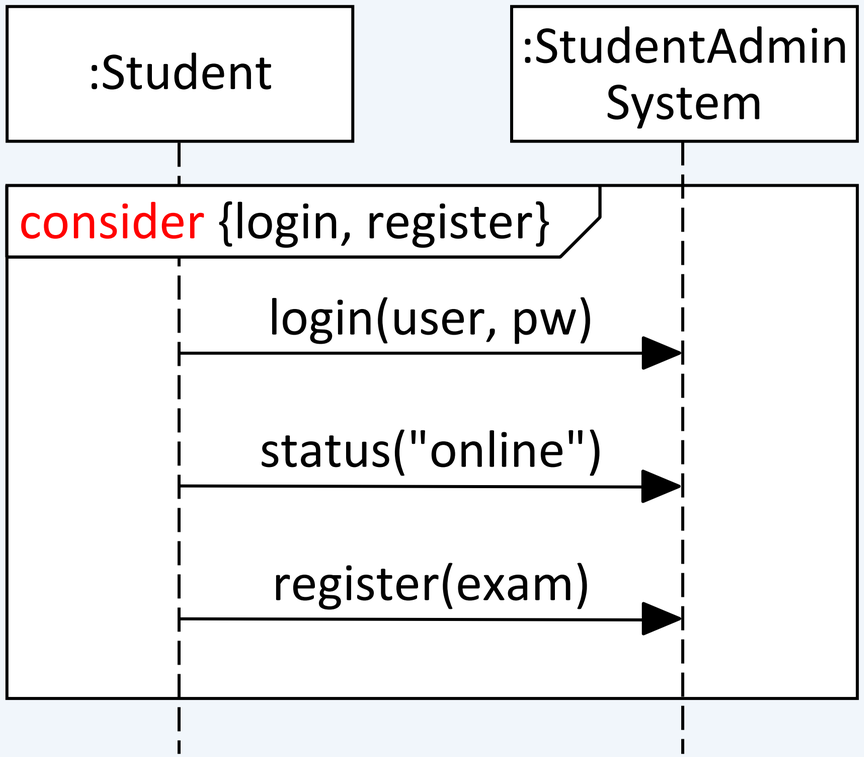
consider{...}

- Inverse of **`ignore`**
- Specification of particularly relevant messages
- Other messages in the operand are automatically classified as not relevant

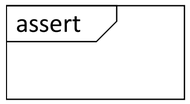


`ignore` VS. `consider`



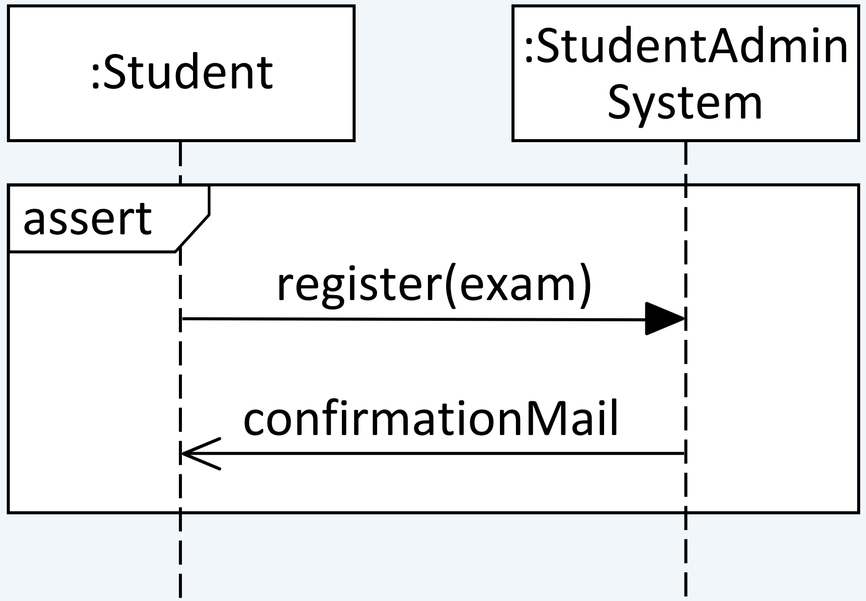


Concurrency and Order: `assert` Operator

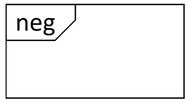


- Indicates that the interaction is mandatory
- Any deviations that are not taken into account in the diagram, but occur in reality, are not permitted

  ⇒ Requirement of true mapping in the implementation
- 1 Operand



Concurrency and Order: `neg` Operator



- Illustration of an invalid interaction process
- Situations in this form must not occur
- Exactly 1 Operand
- Purpose
  - Explicitly point out common errors
  - Map relevant but incorrect processes