

CSSE2003

Software Engineering Studio

Semester 2, 2009

9: UML Sequence Diagrams

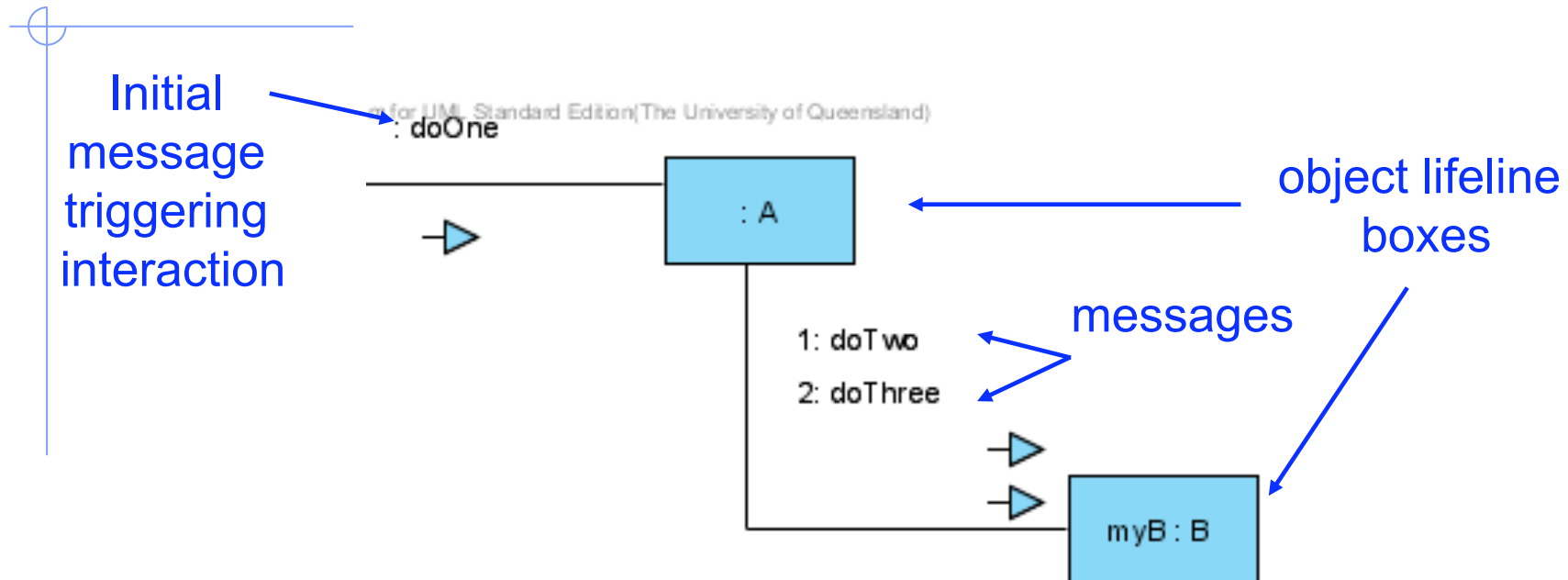
Lecture Summary

- ◆ Two kinds of interaction diagram in UML
 - Sequence diagrams
 - Communication diagrams
- ◆ Focus in this lecture is on sequence diagrams and notation

UML Interaction Diagrams

- ◆ Interaction diagrams are an important artifact of object-oriented design because they address system dynamics.
- ◆ Before an object can send another a message, there must be a relationship between them.
 - A communication diagram describes both static and dynamic relationships between interacting objects. It shows message exchanges as well as structural relationships.
 - A sequence diagram only describes dynamic relationships between interacting objects. It shows message exchanges arranged in a time sequence.

Example of a Communication Diagram



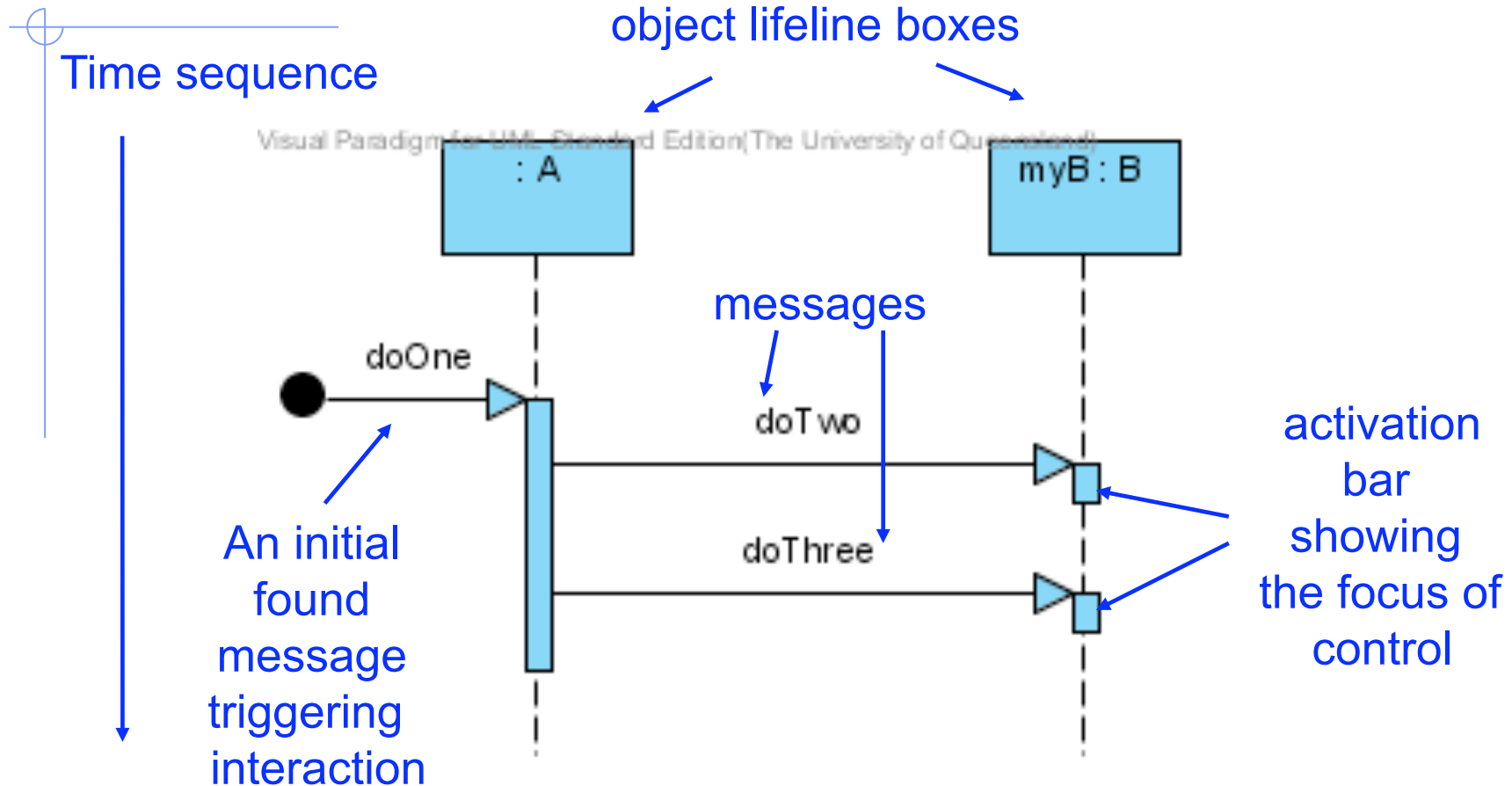
- ◆ *doOne* is sent to an object of *Class A* then
- ◆ the object of *Class A* sends *doTwo* to object *myB* of *Class B* then
- ◆ the object of *Class A* sends *doThree* to object *myB* of *Class B*

Possible code of the interaction

```
Public class A
{
    Private B myB = new B();

    Public void doOne()
    {
        myB.doTwo();
        myB.doThree();
    }
    // ...
}
```

Example of a Sequence Diagram

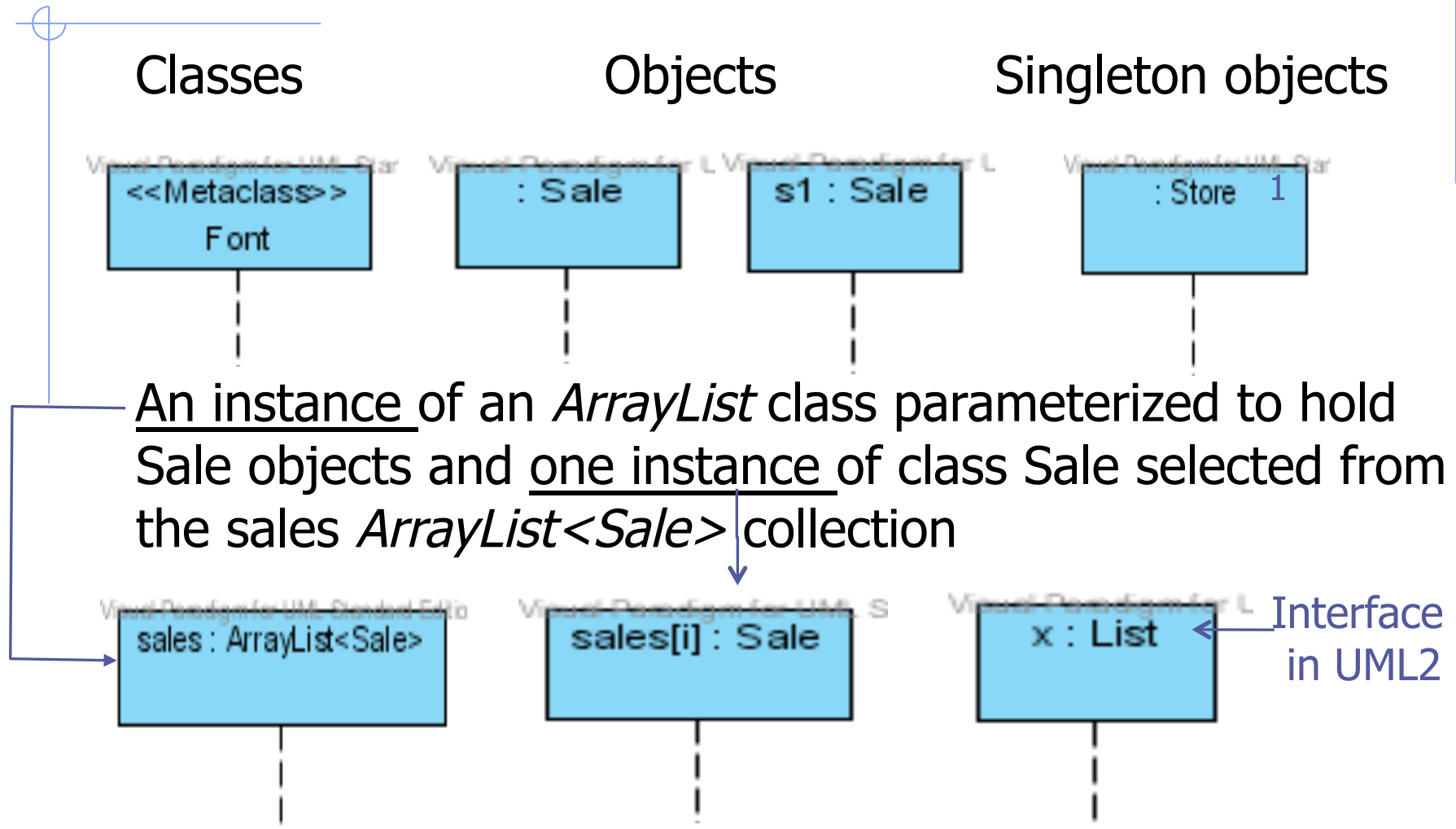


- ◆ What can you show with a sequence diagram that you can't show with a communication diagram ?

Strengths and Weaknesses

Diagram	Strengths	Weaknesses
Communication	Economical in space, new objects can be added in two dimensions. Includes structural aspect.	More difficult to read time order of messages. Fewer notation options.
Sequence	Tracking behaviour of several objects in a given scenario (use case). Clearly shows time order of messages.	Can quickly consume horizontal space by extending rightward when adding new objects.

UML Notation - Participant with Lifeline Boxes



UML Notation - Message syntax

return := message (parameter : parameterType,
...) : returnType

Examples:

Initialize(code)



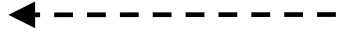
Initialize()

spec := getProductSpec (id)

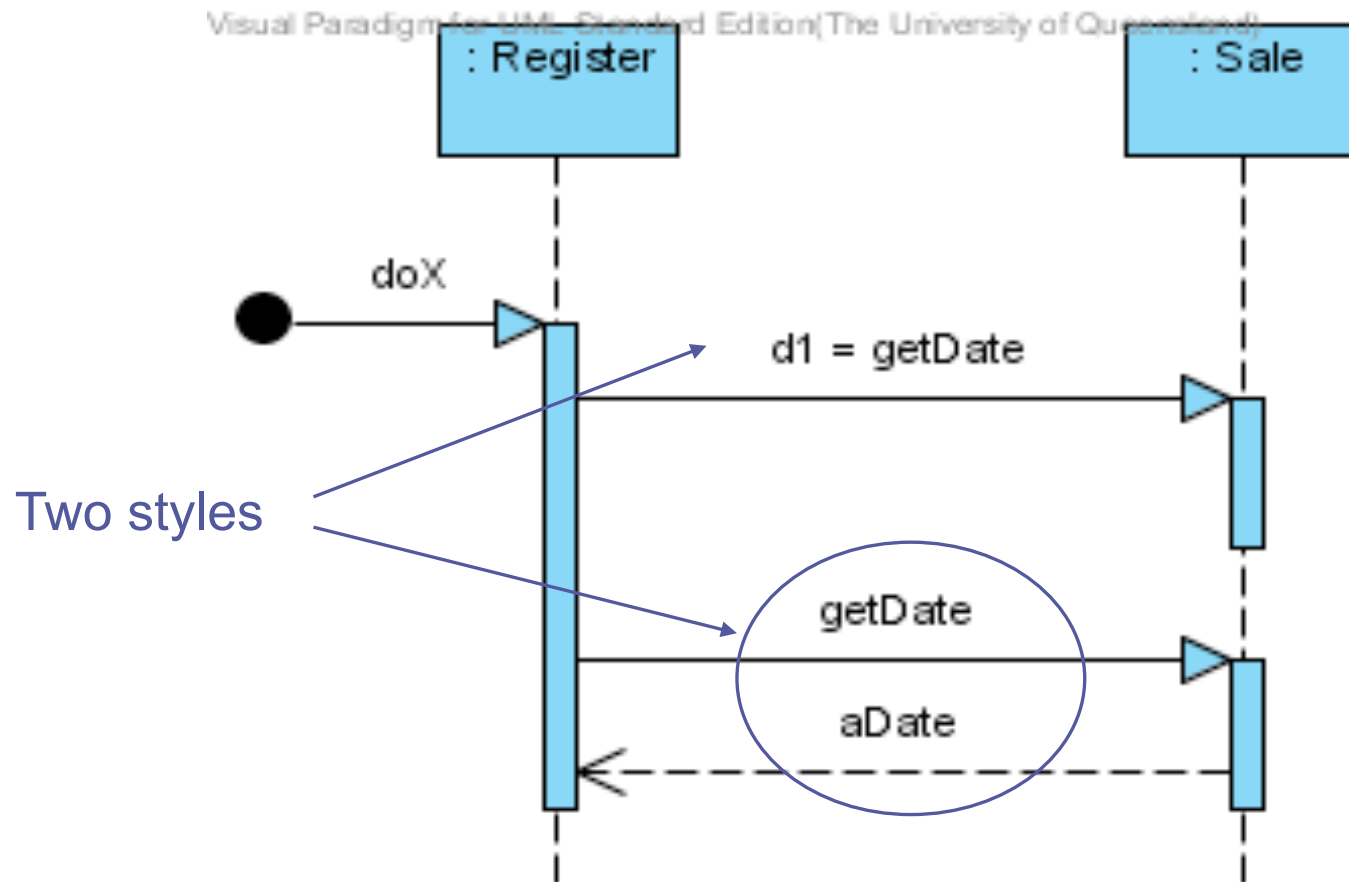
spec := getProductSpec (id: ItemID)

spec := getProductSpec (id: ItemID): ProductSpec

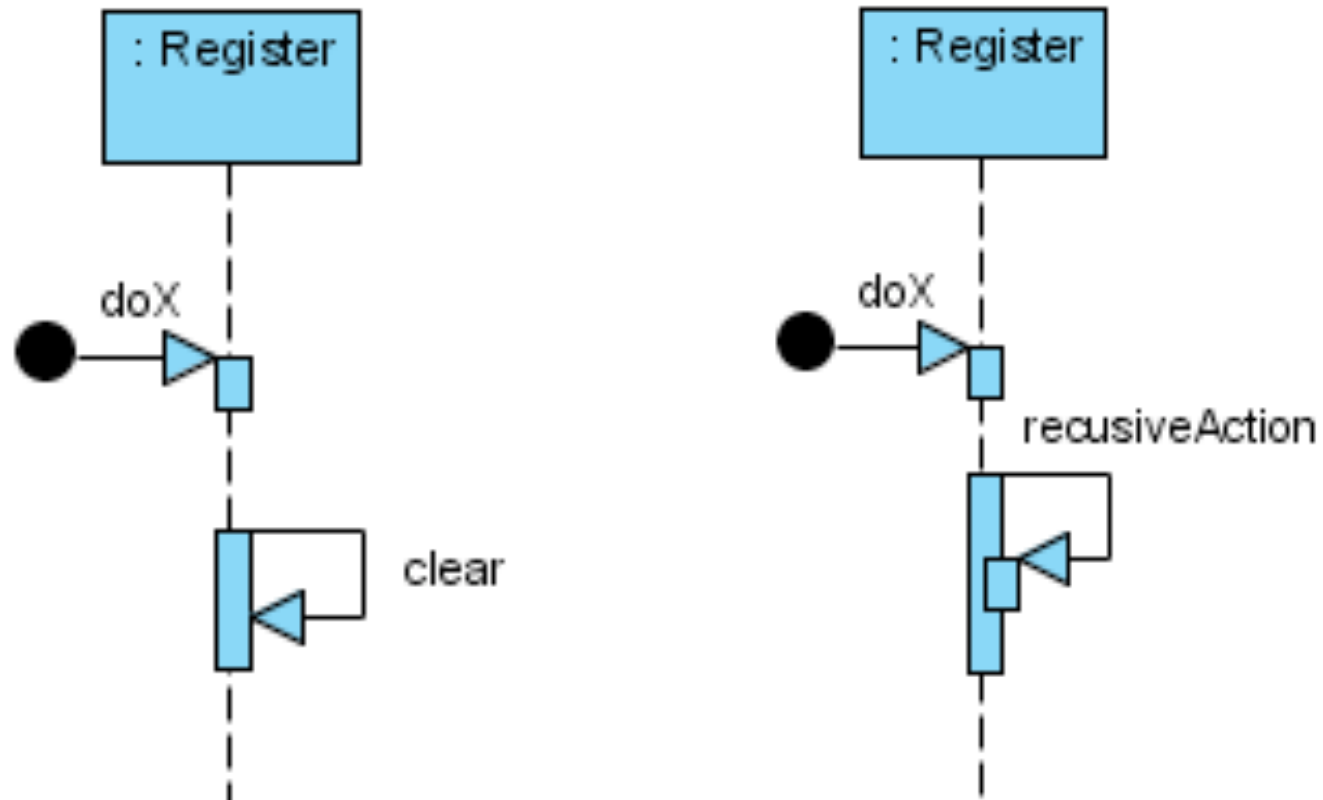
UML Notation – Message arrows

	Semantics
	Synchronous method call (most common): caller waits until method terminates
	Asynchronous call: caller does not wait for the called method to terminate, e.g. when programming with multiple threads
	Return from a method call: this is usually left implicit

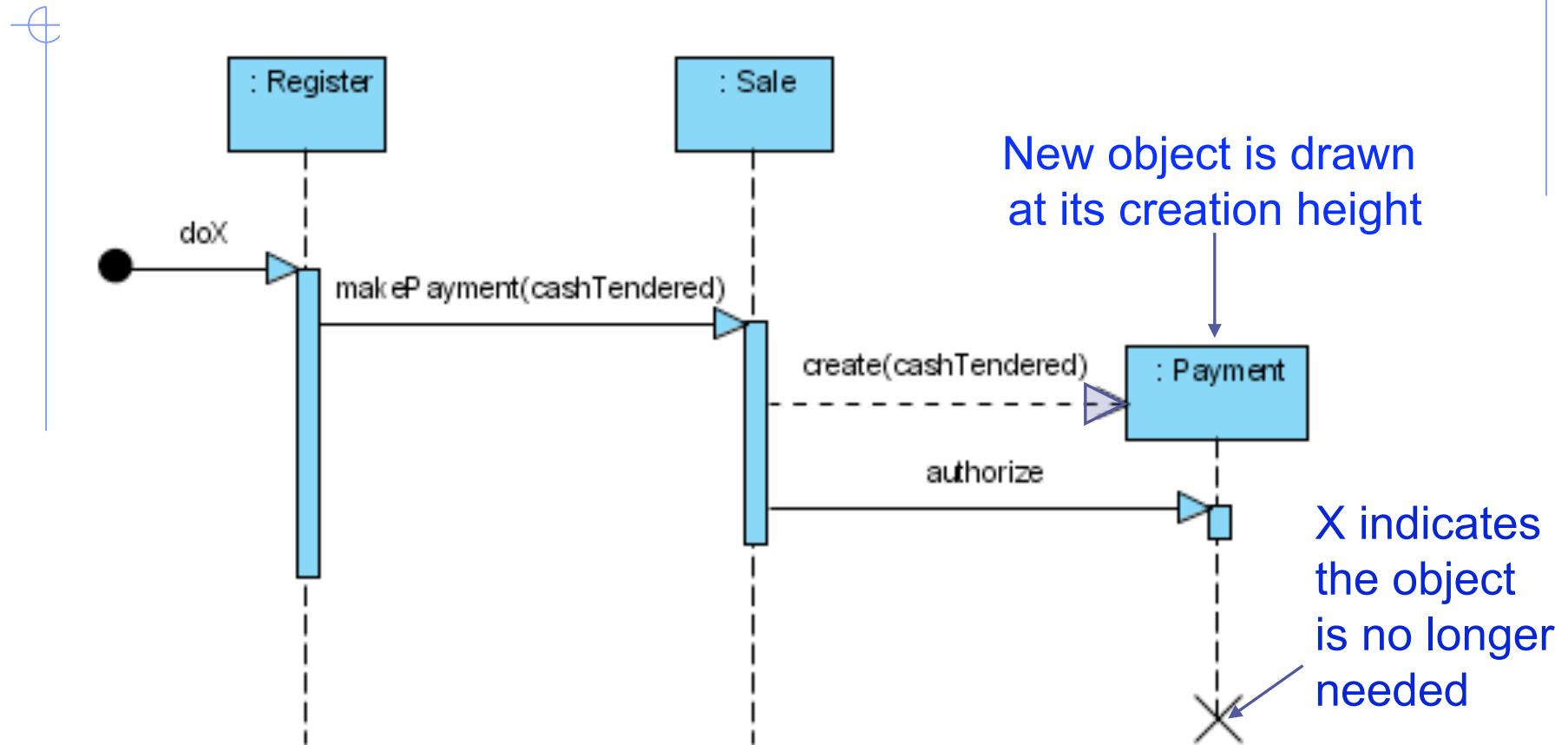
Showing Method Return



Messages to Self

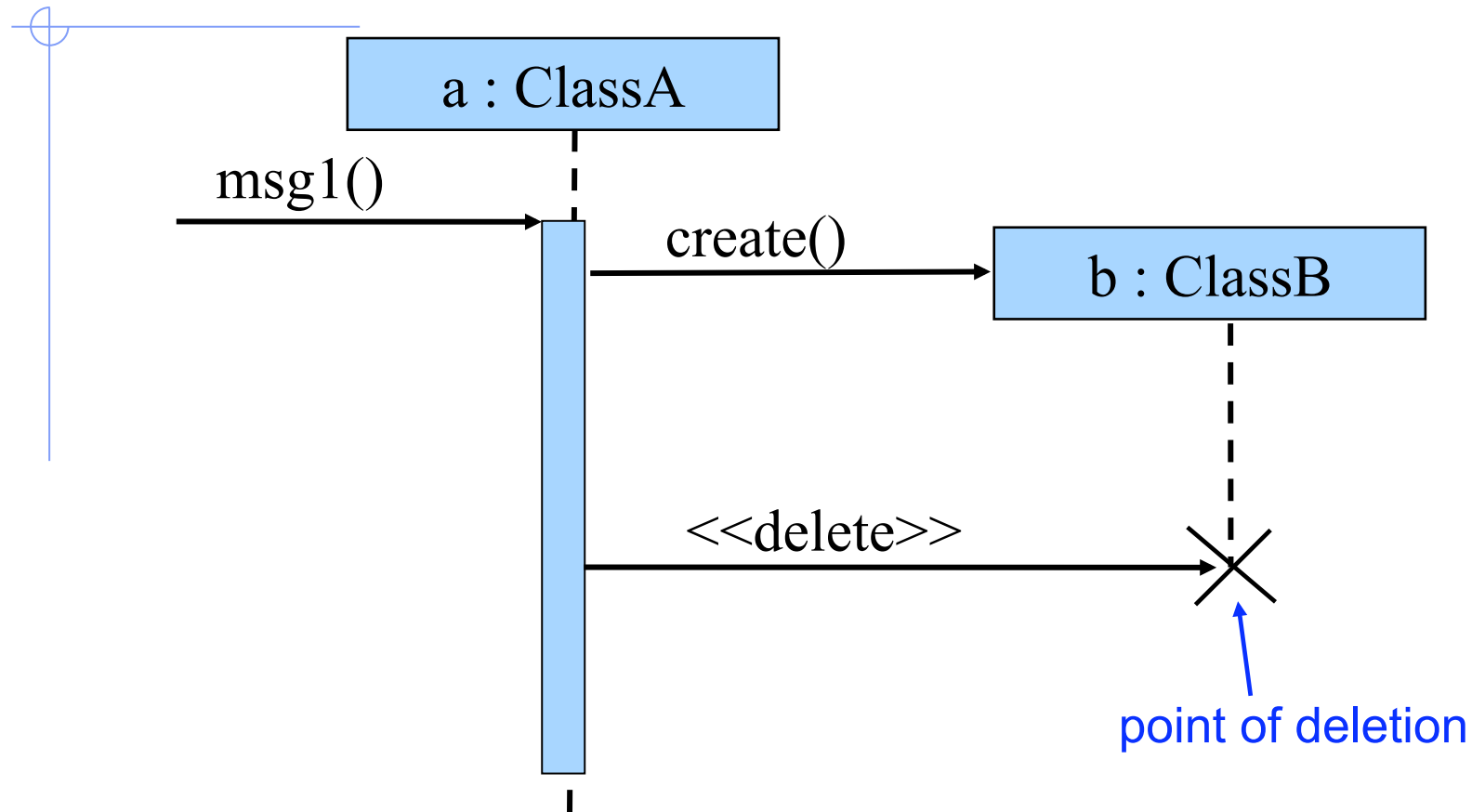


Messages That Create Objects



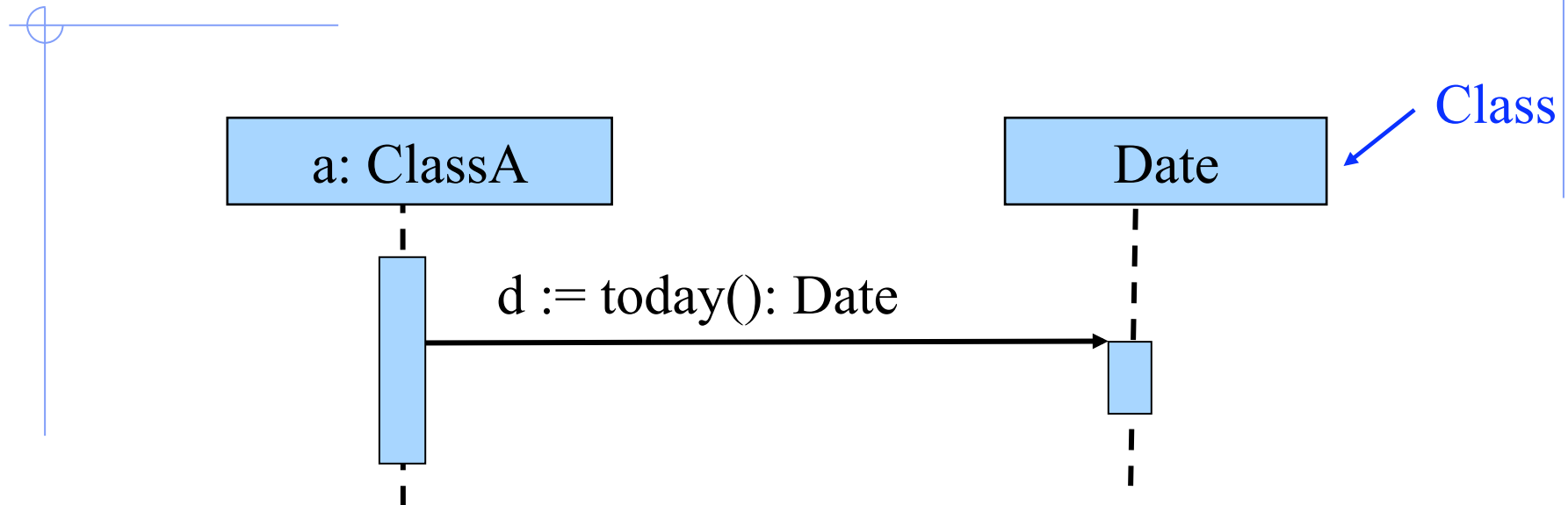
- ◆ The create message arrow head should be filled but Visual Paradigm does not support it (not compliant to the UML standard).

Deleting an Object



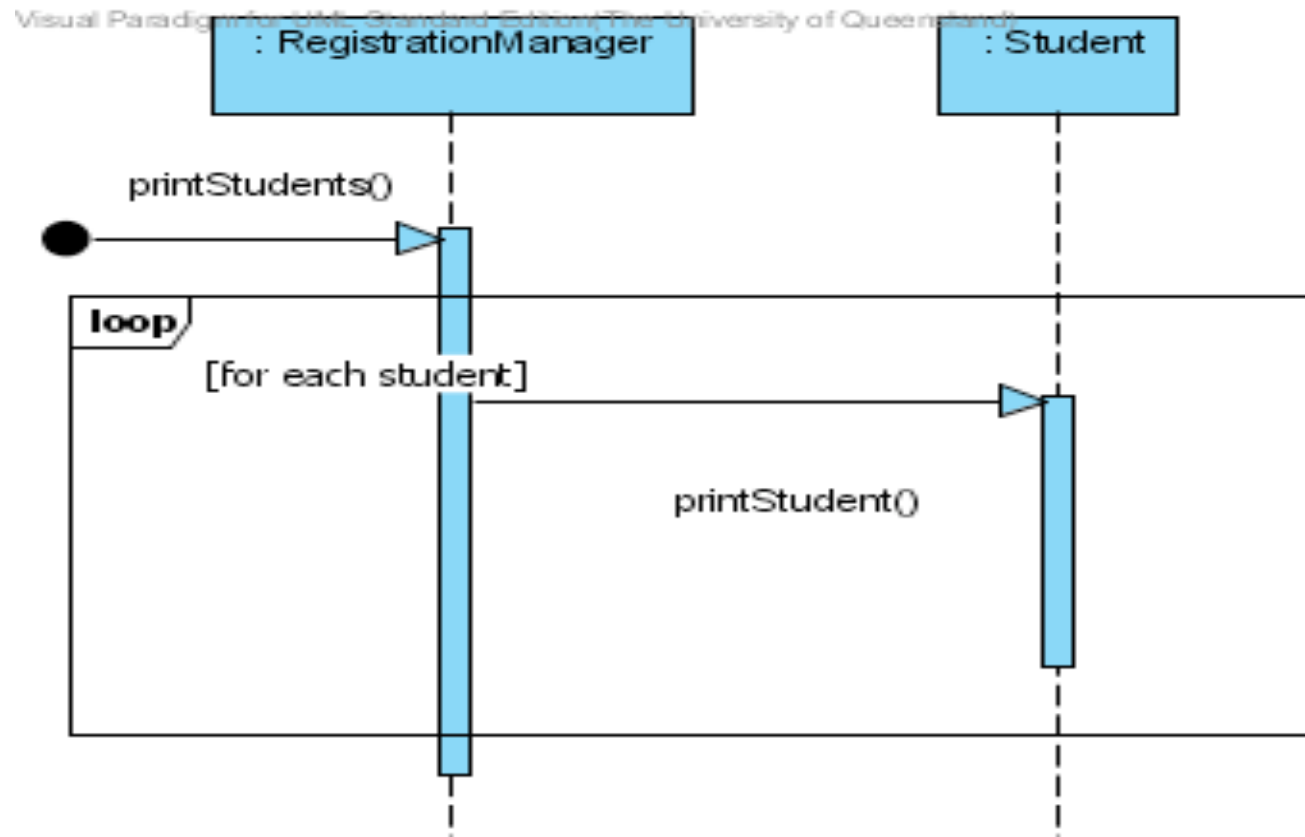
- ◆ Message arrow indicates that *a* explicitly deletes *b*

Messages to Classes to Invoke Static Methods



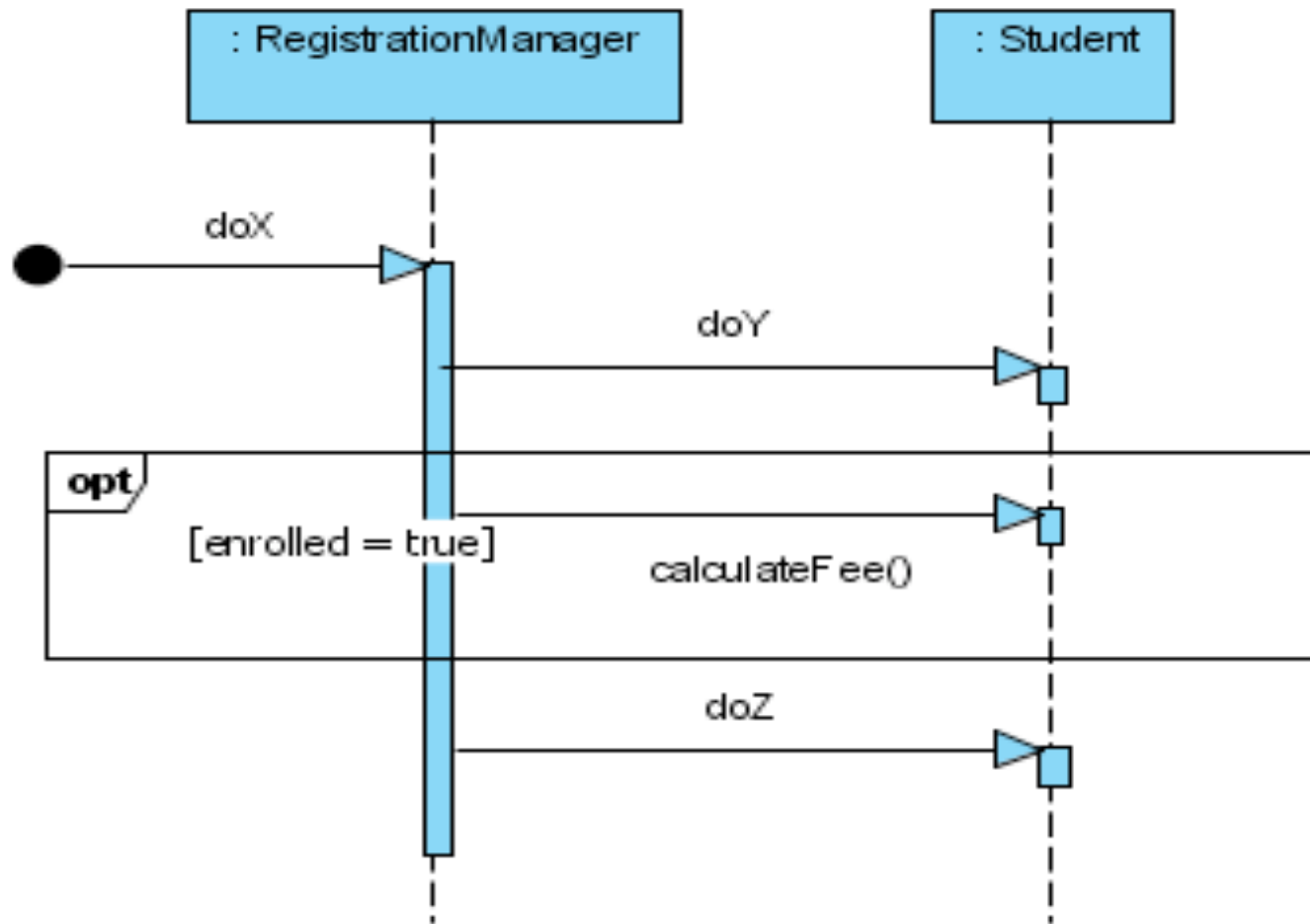
- ◆ The ':' (and underlining) designates an object participant.
- ◆ A participant that is not prefixed by ':' designates a class.
- ◆ Message `today()` is sent to class `Date`, rather than to an instance of `Date` (e.g. `Date d = Date.today();`).

Expressing Repetition (loop)

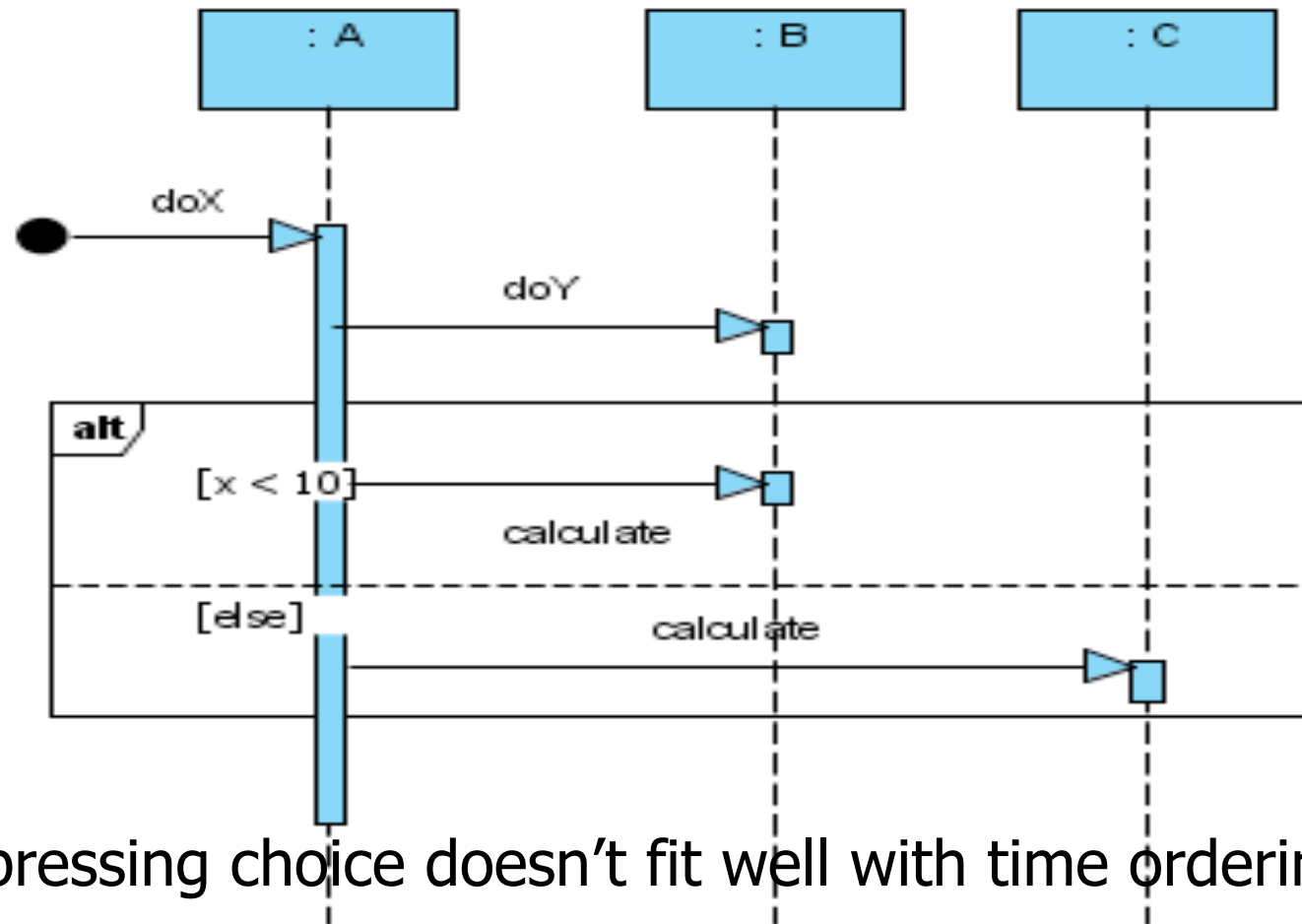


- ◆ Extension in UML2.x

Conditional Messages (opt)

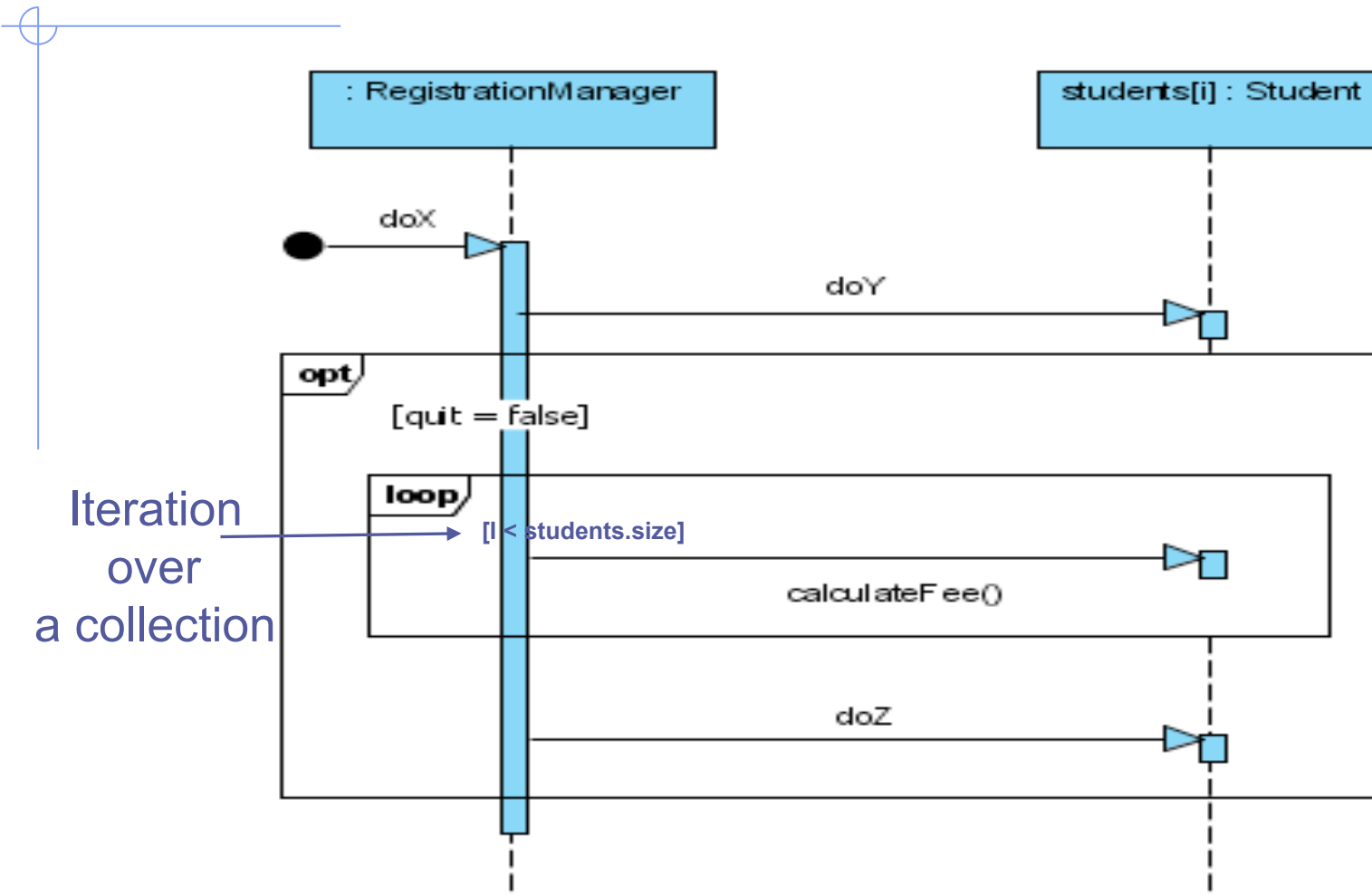


Mutually Exclusive Conditional Messages (alt)



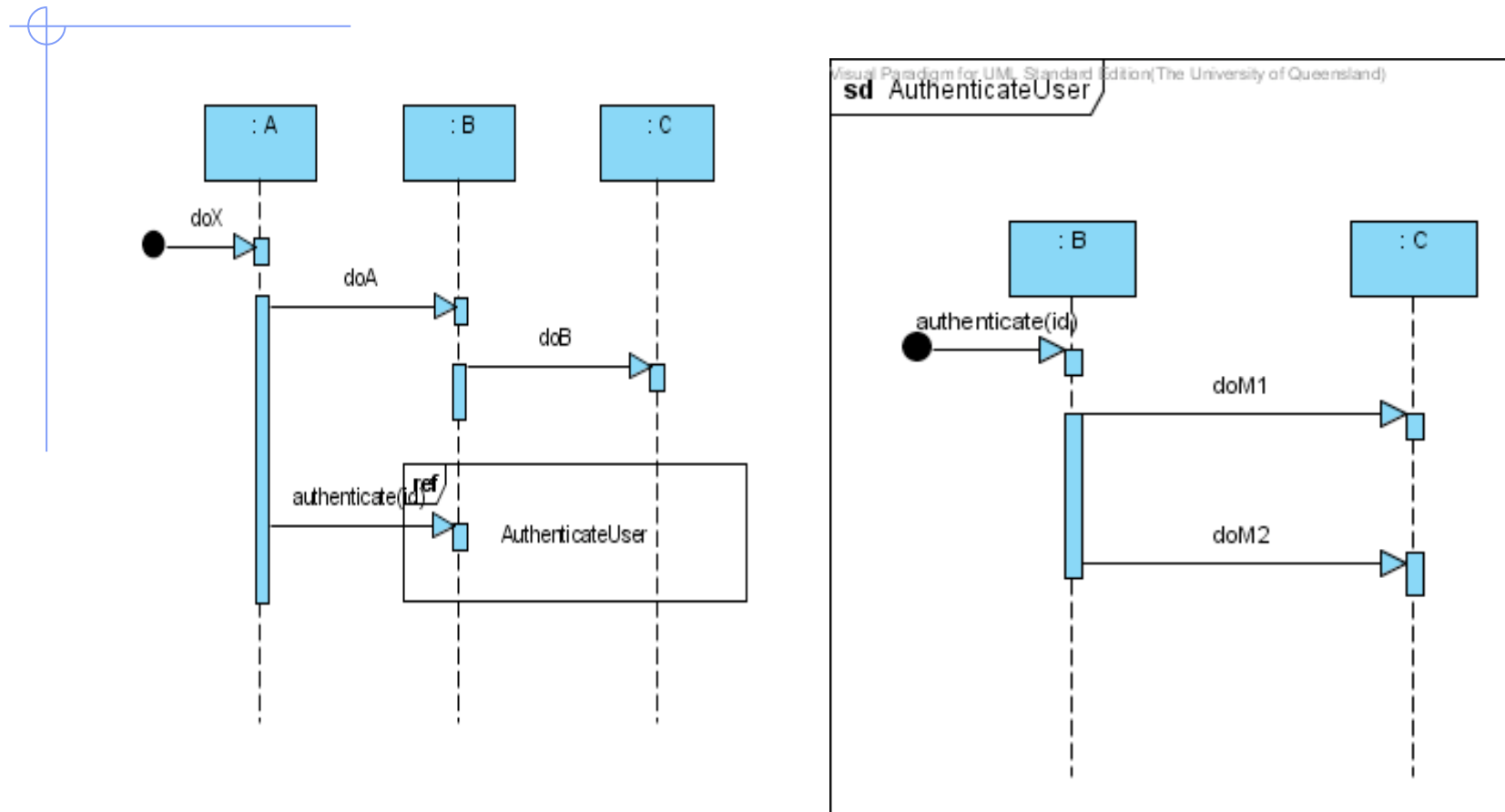
- ◆ Expressing choice doesn't fit well with time ordering
- ◆ This is UML 2 notation

Nesting of Frames



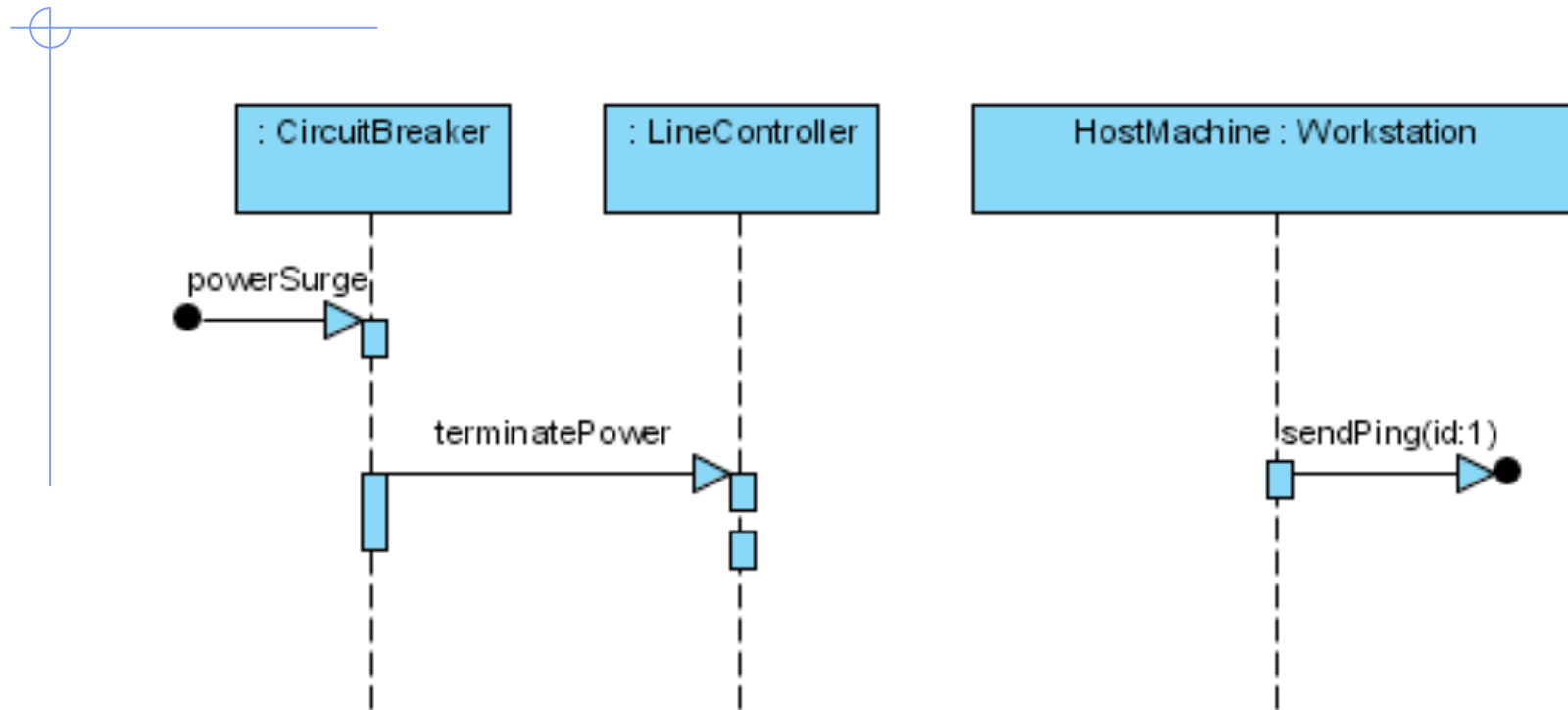
- ◆ Use these notations very sparingly

Reference Interactions



- ◆ Referencing other interactions (SDs)

Found and Lost Messages



- ◆ Found message from unknown sources with respect to the current SD (e.g. used to model exceptions)
- ◆ Lost message: does not reach its destination (e.g. used to model a network failure resulting an undelivered message)

Follow-up Reading

- ◆ Larman: Chapter 15 (Interaction diagrams)