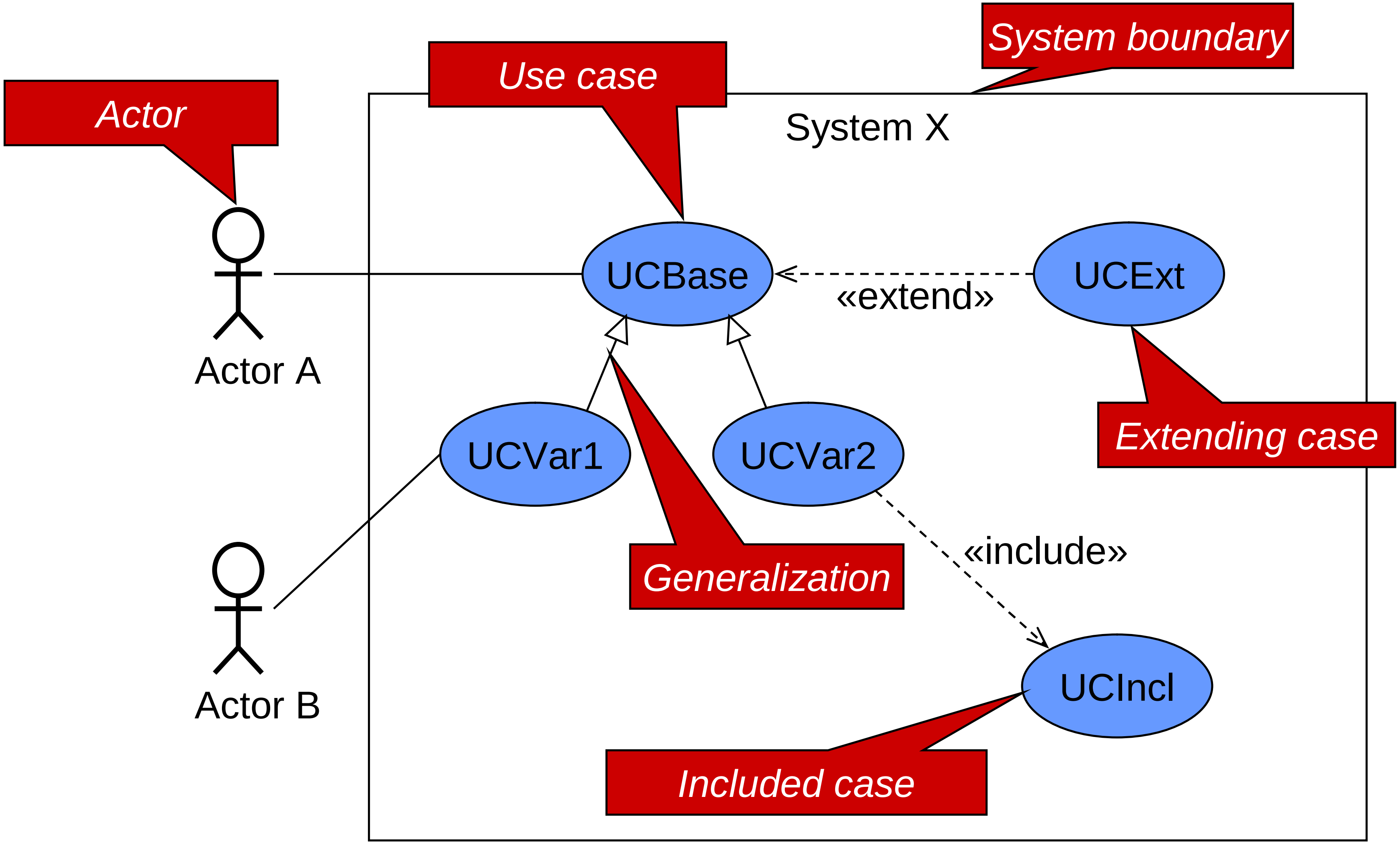
From Analysis to Design

Use Cases: Notation Overview



Paths, Scenarios, Processes

- Use case definitions are mostly an overlay of several conditionally closely related paths.

- A certain path through a use case is called a scenario.

  A scenario shows a particular set and combination of conditions within the same use case, e.g., ordering some goods under

  - scenario 1: All goes well.
  - scenario 2: There are not enough goods.
  - scenario 3: Credits are insufficient.

  ⇒ Interaction Diagrams

- Processes involving *different* use cases are shown in workflows, e.g., from ordering to delivery and payment.

  ⇒ Activity Diagrams

Use Cases Example: The University Library System

- Problem statement :

You have been contracted to develop a software system for a university library. The library currently uses a 1960's program, written in an obsolete language, for some simple bookkeeping tasks, and a card index, for user browsing. You are asked to build an interactive system which handles both of these aspects on line.

- Vague, but typical of an initial request

University Library – requirements facts after some investigation

- **Books and journals** The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals. New books and journals arrive regularly and are sometimes disposed of. The current year's journals are sent away to be bound into volumes at the end of each year. There may in future be be a requirement for users to be able to extend the loan of a book if it is not reserved.
- **Borrowing** The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.
- **Browsing** The system will allow users to search for a book by topic, author, etc., to check whether a copy of a book is available for loan, and if not, to reserve the book. Anybody can browse in the library

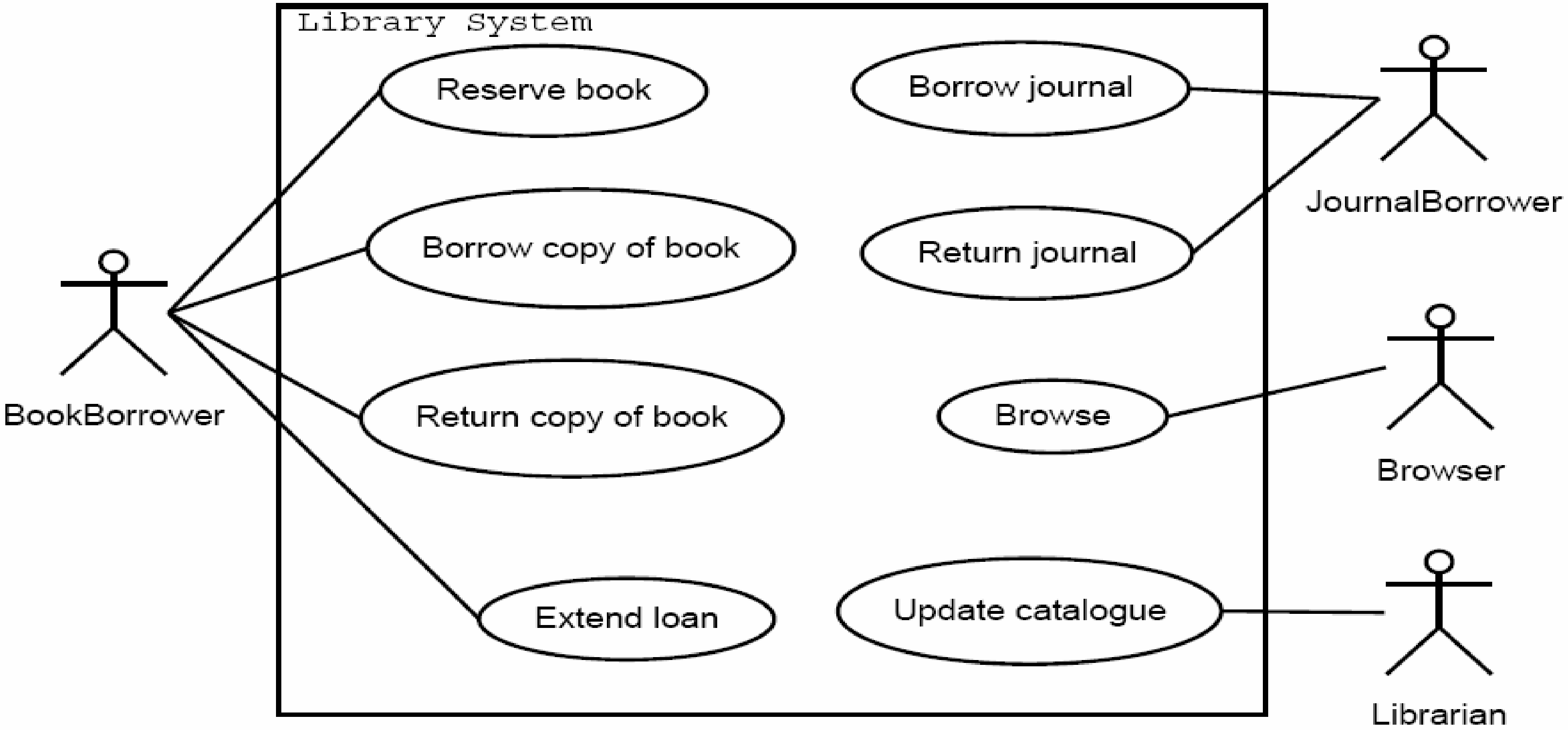
Actors in the Library System

- Users:
  - Librarian
  - Library member
  - Non-library member
- Actors (roles):
  - Librarian
  - BookBorrower
  - JournalBorrower
  - Browser

Use Cases in the Library System

- BookBorrower
  - Reserve book, Borrow copy of book, Return copy of book, Extend loan
- JournalBorrower
  - Borrow journal, Return journal
- Browser
  - Browse
- Librarian
  - Update catalogue

Library System Use Case Diagram



Documenting Use Cases

- Detail of each use case should be documented
  - Use third-person, active voice English
  - Use terms from the problem domain (i.e. Those that the user would understand)
  - Say ***what*** the system does, not ***how*** or implementation detail
- The **basic course of action** is the main start-to-finish path the user will follow under normal circumstances
- An **alternative course of action** can represent an infrequently used path, an exception, or error

Borrow copy of book use case

Actor: BookBorrower

Trigger: A potential BookBorrower presents a book to the Librarian

Preconditon(s):

The Librarian has successfully identified himself or herself to the library system by entering a valid library staff identification number and password.

Basic course:

A BookBorrower presents a book. The system checks that the potential borrower is a member of the library and that he/she does not already have the maximum permitted number of books on loan. The system records that this library member has this copy of the book on loan.

Alternative course(s):

1) If potential borrower is not a member of the library the system refuses the loan.

2) If the library member has the maximum permitted number of books (12 for staff members, otherwise 6) the system refuses the loan

Postcondition(s):

If the loan is successful then the system records the loan against the library member's record. Otherwise, nothing in the system has changed when this use case ends.

Documenting Use Cases

- Granularity? How big or small should a use case be?
  - Guideline: a use case should accomplish something of value for the actor involved
- E.g. ATM system: is “enter pin number using keypad,, a use case?
  - No, it doesn’t achieve something of value for the actor
  - Probably one step in a “withdraw funds,, use case — here the value for the user is the funds received from the ATM

Interaction with External Systems

- Interaction with external systems can be represented in four ways:
  1. Show each interaction with external systems in the diagram.
  2. Only show use cases for external interaction, if the other system initiates the contact.
  3. Only show system actors if they need the already identified use cases.
  4. Disallow systems as actors, concentrate on users.

Use **external events** to identify use cases not covered by actors. Find every possible event of the "world outside" needing a reaction.

Modeling System Structure

Class Diagrams in Analysis

- Notation required for:
  - Classes
  - Relations:
    - Multiplicity
    - Roles
    - Associations
    - Aggregation
    - Composition
  - Generalization
  - Objects

?

Three Views on OO-Modeling

- Conceptual view (analysis)
  - classes represent the application concepts
  - language and system independent
  - few classes, few diagrams
- Design view (late analysis, early design)
  - classes represent SW-interfaces
  - describes the solution of difficult problems of the implementation
  - structures the system in layers, subsystems and packages
- Implementation view (late design and programming)
  - classes represent code in a programming language
  - is directly mapped on the implementation
  - example: mapping of associations

Views are part of Unified Process, not of UML itself.

Analysis: How to find Classes?

- First cut: Identify classes by nouns
  - Categorize the nouns
  - Remove nouns and concepts which do not represent independent conceptual classes
  - Choose meaningful and substantial class names
  - Document each class in short (~1 defining sentence)

Example: University Library System

- **Books and journals** The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals. New books and journals arrive regularly. The current year's journals are sent away to be bound into volumes at the end of each year.
- **Borrowing** The system must keep track of when books and journals are borrowed and returned, enforcing the rules described above.

Nouns in the library system

- Discard:
  - library, because it is outside of the scope of our system
  - short term loan, because a loan is really an action, the lending of a book to a user
  - member of the library, which is redundant: same as library member
  - week, because it's a measure of time, not a thing
  - time, because it's outside the scope of the system
  - system, because it's part of the meta-language of requirements description
  - rule, for the same reason

Nouns in the library system

- Left with:
  - book
  - journal
  - copy (of book)
  - library member
  - member of staff
  - volume
- Note, library member and member of staff are also *users* of the system
  - represented in the system because data on these users will be maintained

Analysis: Find Attributes

- Identify attributes following the noun method
- Check the attribute name
  - Each attribute name should be
    - a noun,
    - chosen as concrete as possible,
    - no homonym.
- Define the type of the attributes
  - In the conceptual class diagram the type can remain unspecified.

Analysis: Find Associations

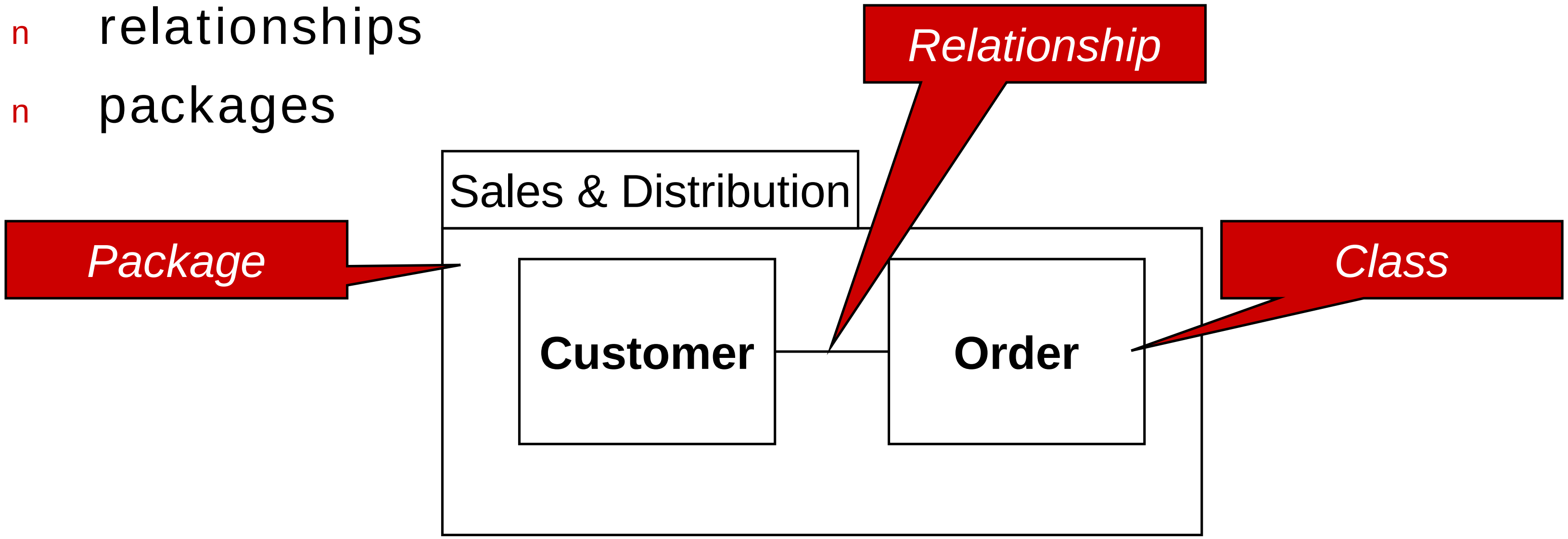
- Identify possible associations between objects
  - In the relevant documents, find verbs and nouns identifying actions or processes.
  - Identify the concerned classes for each association.
- Categorize these associations
  - actions: e.g., *drives car, books flight*
  - properties: e.g., *has age*
  - general relations: e.g., *depends on, is married to*
- Delete non-conceptual associations
- Define association and role names if necessary
- Determine the multiplicity of each role of each association

Conceptual Modeling by Classes

- Definitional goals:
  - conceptual perspective
    - defining the application domain by
      - capturing its relevant concepts (concrete or abstract ones) and
      - representing them by classes
    - provides language independence
- Representational means:
  - class diagrams
    - classes
    - relationships
  - object diagrams
    - objects (as an instance of a class)
    - references

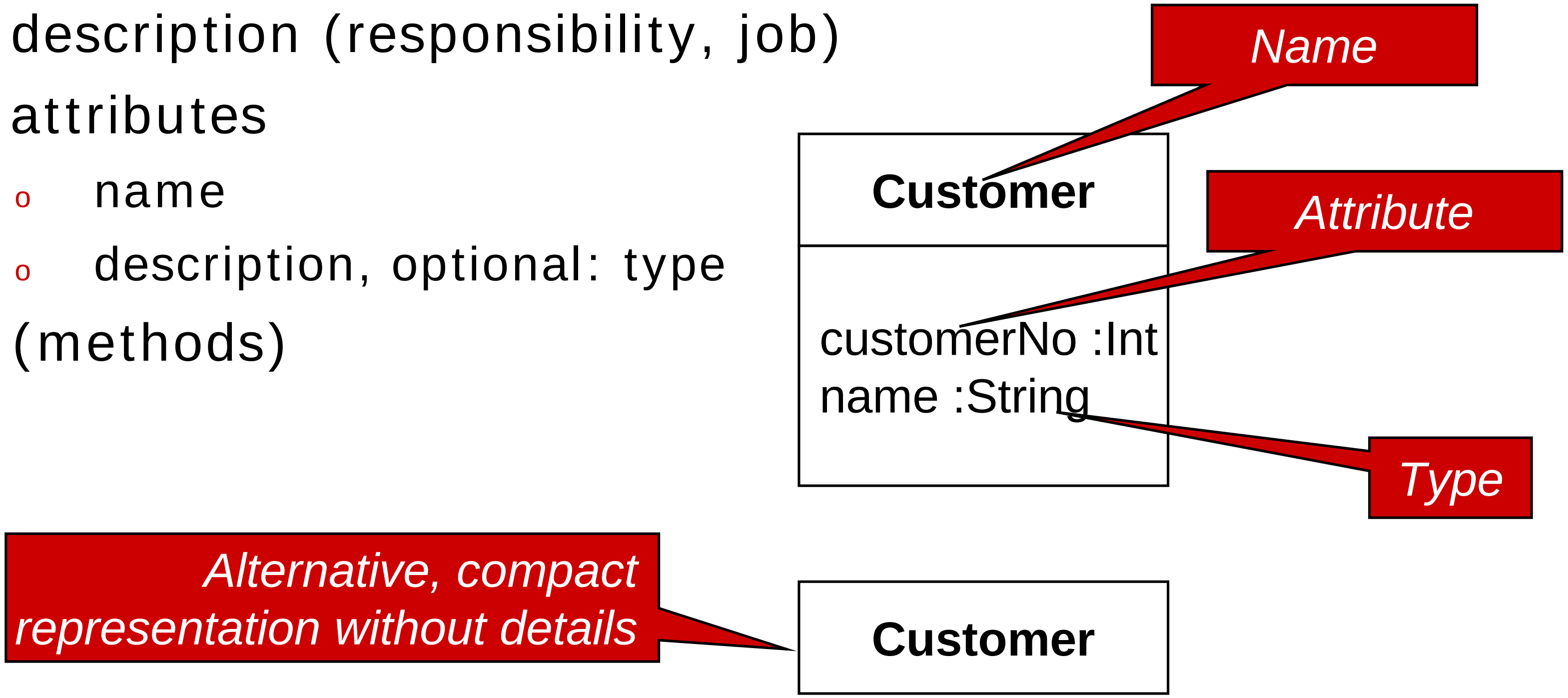
Class Diagram

- A class diagram is a graphical representation of a static view on declarative, static elements.
- Class diagrams contain:
  - classes
  - relationships
  - packages



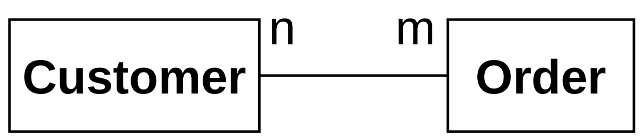
Class Diagram on Conceptual Level

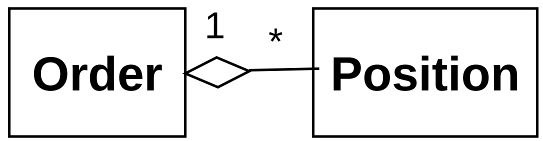
- A class has
  - name
  - description (responsibility, job)
  - attributes
    - name
    - description, optional: type
  - (methods)

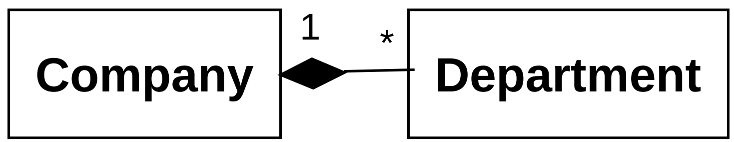


Three Cases of Relations

- A Relation can take the form of:
  - Association
    - Most general (n:m-relationship)
  - Aggregation
    - Stronger relationship, one is part of the other
  - Composition
    - Even stronger than aggregation, also ties lifecylces together



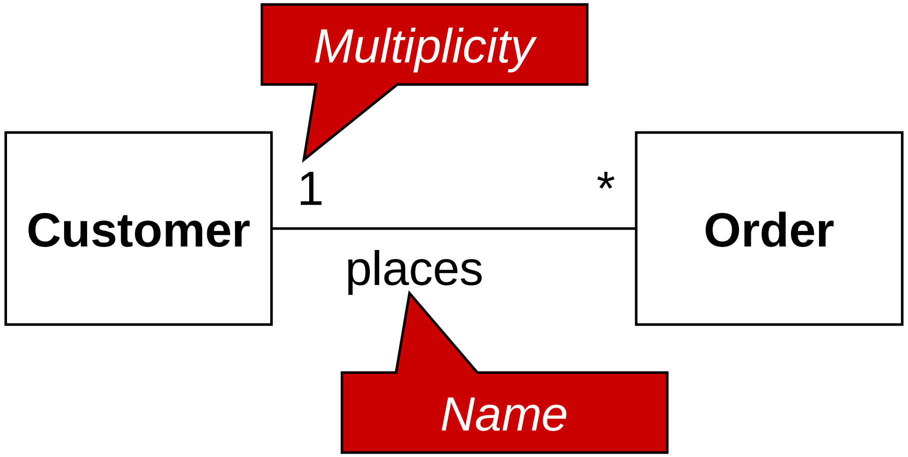




More details on the following slides.

Association

- An association is a semantic relationship between classes which concerns the connection (e.g., references) between its instances.
- Notation:
  - name
  - multiplicity
  - description (extracted from use cases)

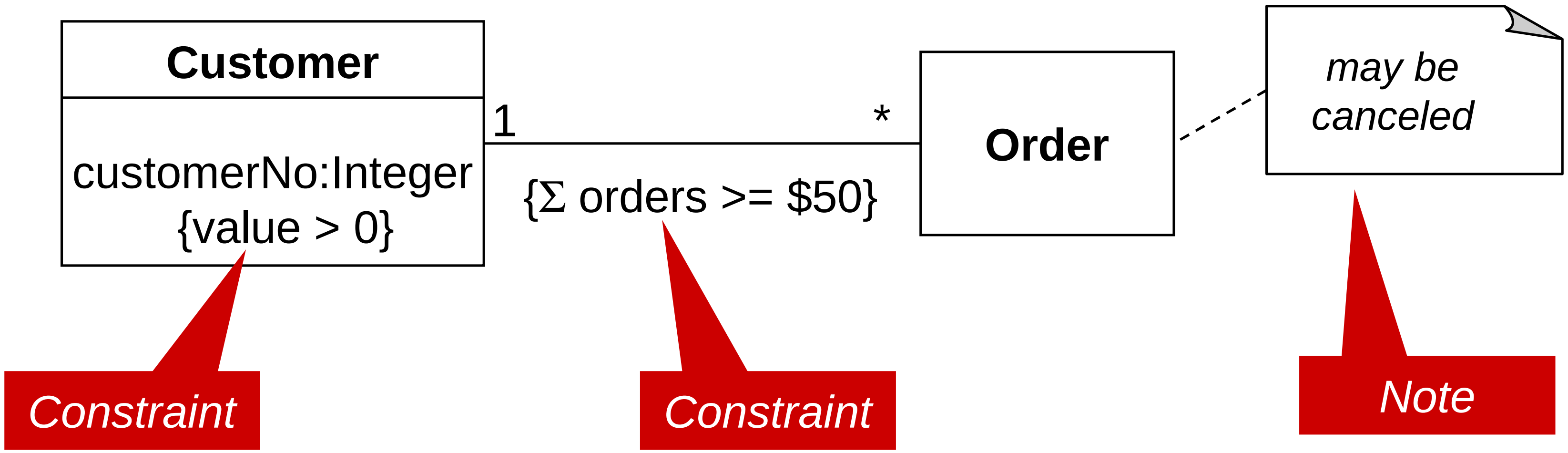


An order is done by exactly one customer.

Each customer can place none or several orders.

More Semantics on Classes and Relationships

- There is a need for *non-structural* semantic elements on associations, attributes, classes etc.
- Constraints are restrictions denoted by expressions.
- Notes are in natural language. They may also contain constraints.
- Constraints and notes can be used for annotating any element in UML diagrams.

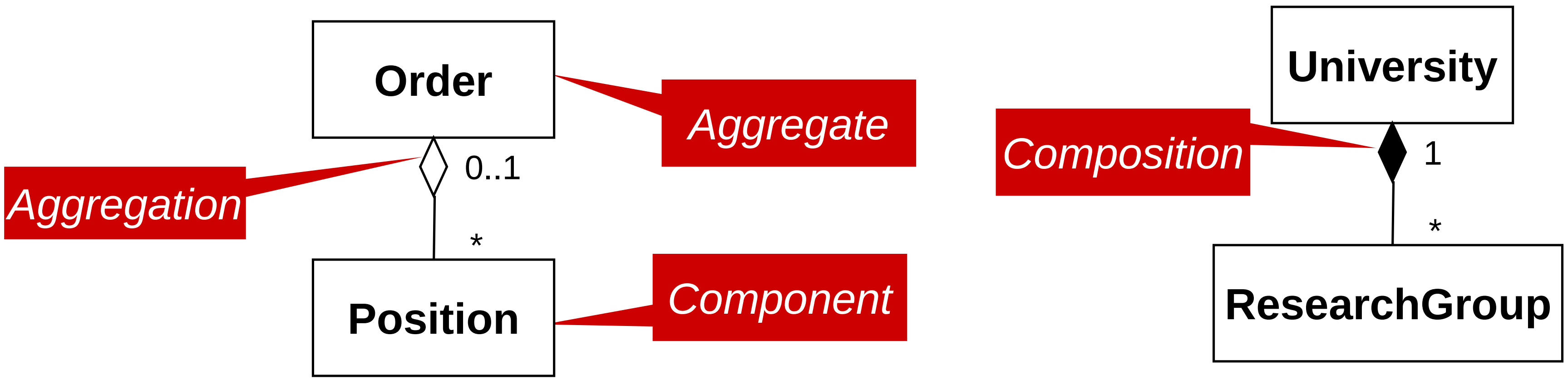


Association vs. Aggregation

- The following rules can give a *hint* that an association is an aggregation:
  - Can the relationship be described by „consists of“ or „is part of“? (Collection, container, whole & parts, group & members, ...)
  - Is the multiplicity on one side of the association 1 or 0..1 (only a vague indicator)?
  - Is the association transitive and asymmetric?
  - Are the part objects accessed exclusively by the aggregate object?
  - Is the lifetime of the component restricted by the lifetime of the aggregate?

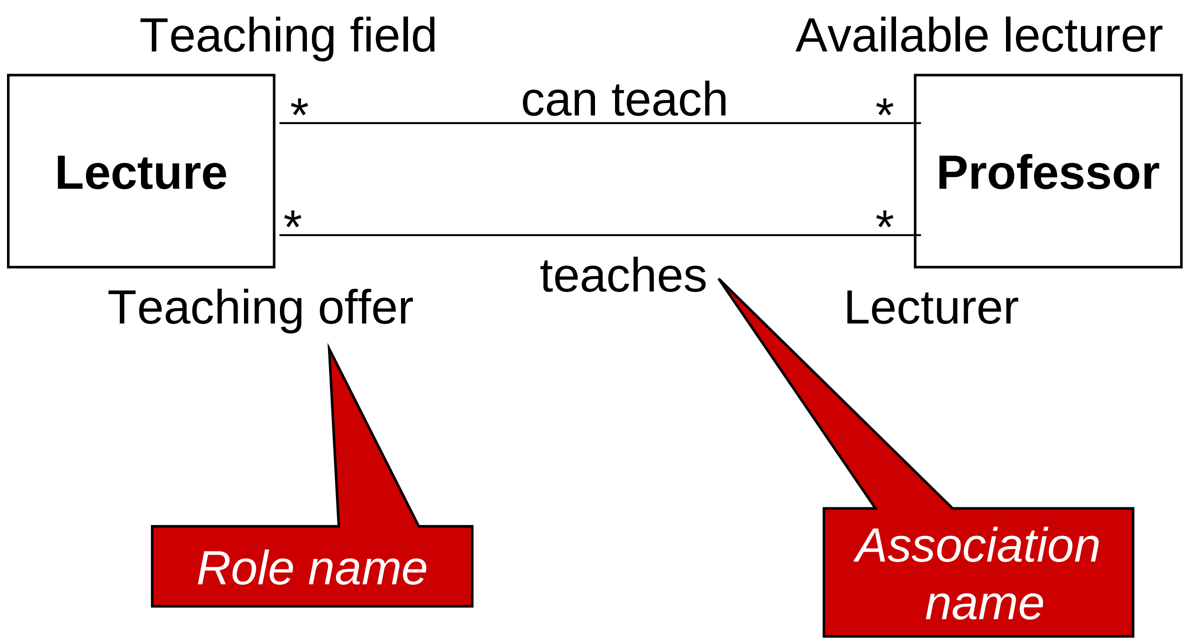
Aggregation vs. Composition

- Properties of aggregation:
  - an aggregation is a specific semantic relationship between aggregate and component B. “B is a part of A.”
- Additional properties of composition:
  - a composition is an aggregation with the additional property of *dependent* existence of the component
  - exclusive aggregation (component can only be component of a single aggregate)



Role

- A role has a name and describes the meaning of the classes participating in an association more precisely.



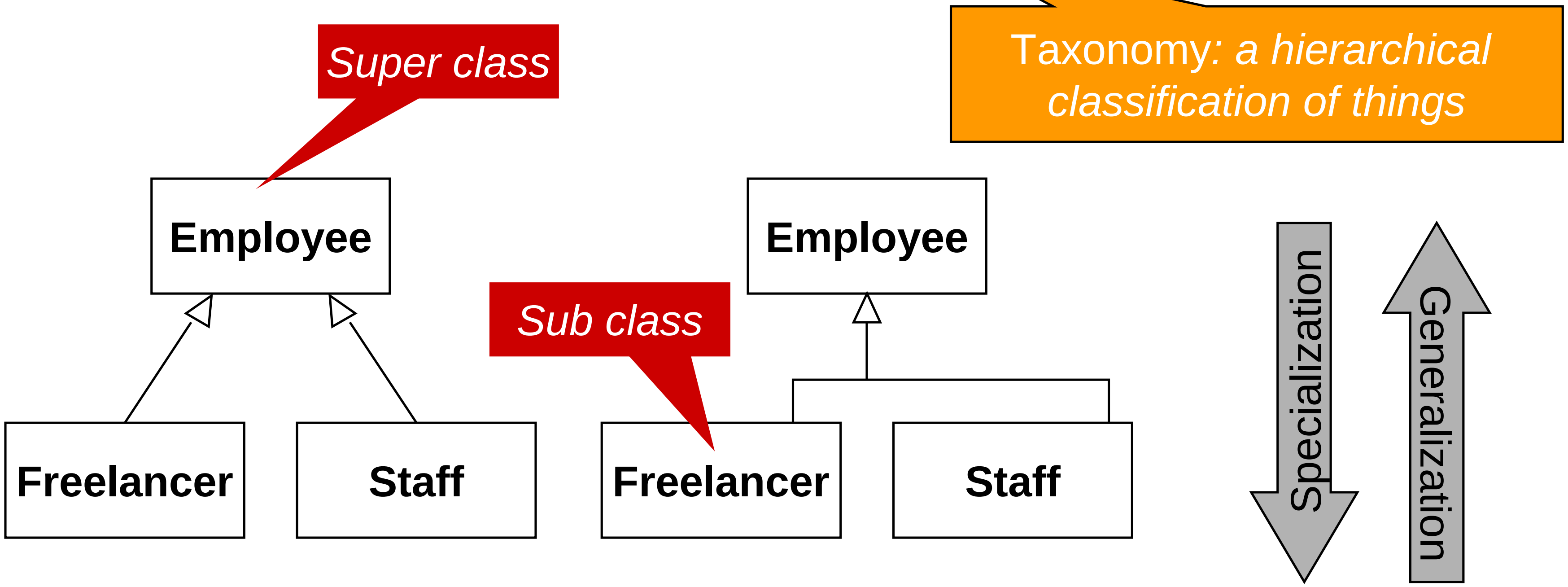
Association Multiplicities: Summary

- The multiplicity of an association defines the valid range of values for the number of objects taking part in the association (cardinalities).

Notation	*Examples*	*Meaning*
number	1, 4	exactly this many
number1..number2	1..5, 2..10	[number1, number2]
number..*	0..*, 4..*	[number, infinity]
*	*	0..*

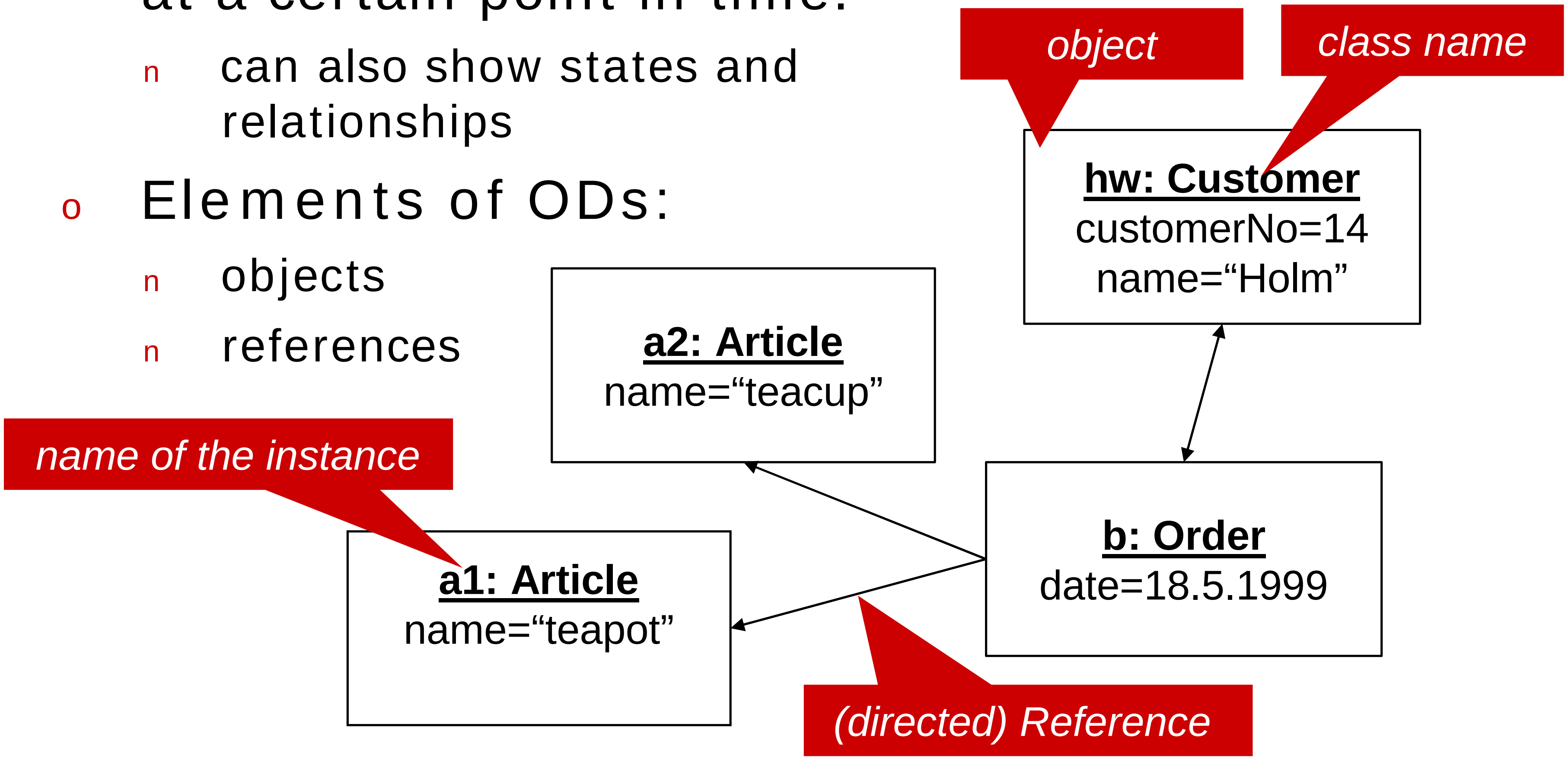
Generalization

- Generalization is a semantic relationship between a more general concept A (super class) and a more special concept B (sub class).
- For classes: Inheritance builds a taxonomy.

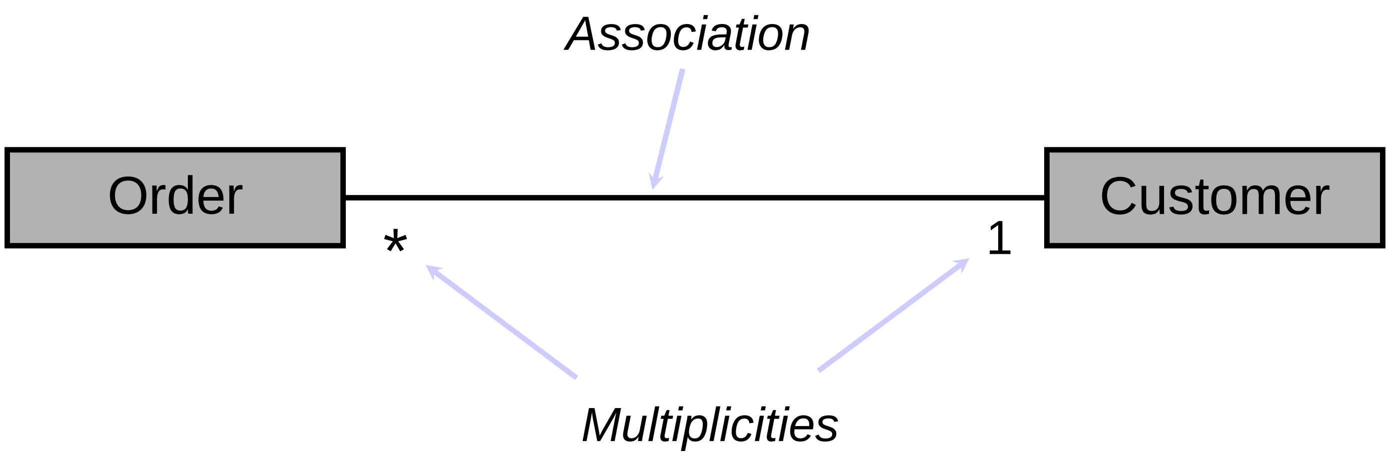


Object Diagrams (vs. Class Diagrams)

- An object diagram shows objects (instances) at a certain point in time.
  - can also show states and relationships
- Elements of ODs:
  - objects
  - references

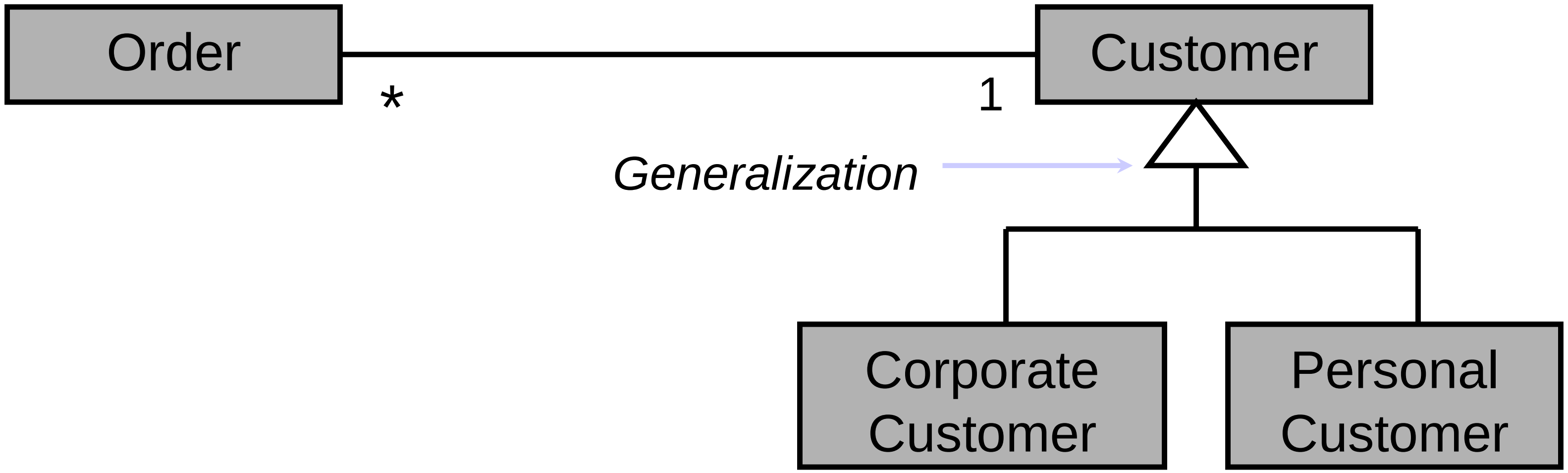


Example: Associations



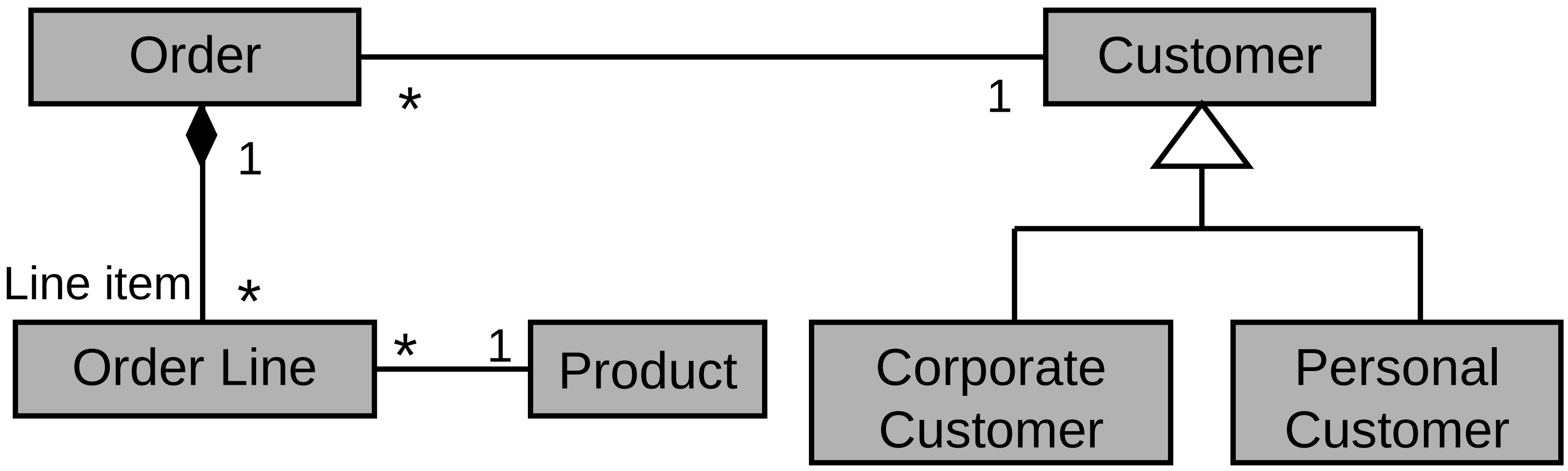
Order: Customers may issue several orders.
Each order belongs to a customer.
An order is never related to more than one customer.

Example: Generalization



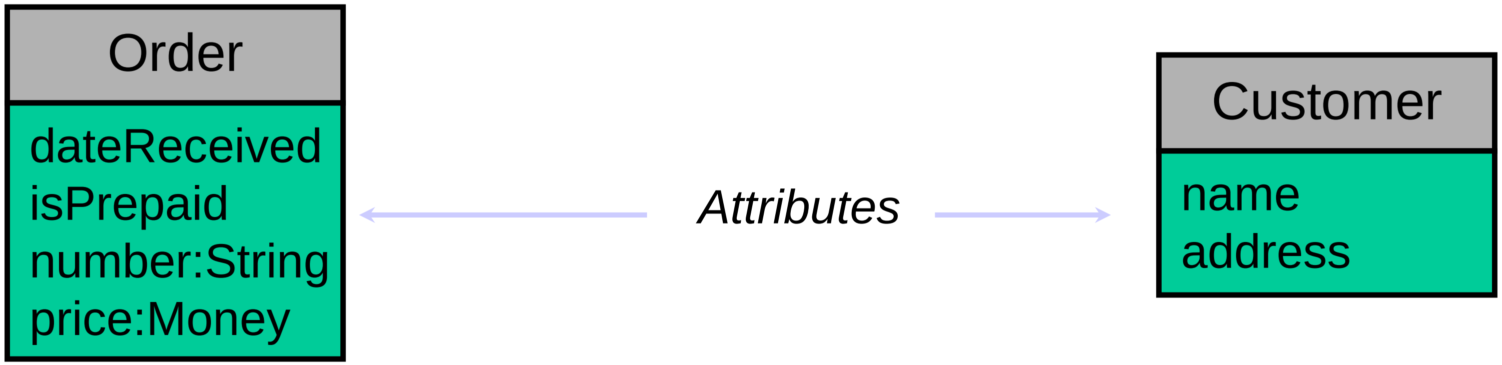
We distinguish corporate customers from personal customers, since corporate customers are billed monthly whereas personal customers need to prepay their orders with a credit card.

Example: More Associations



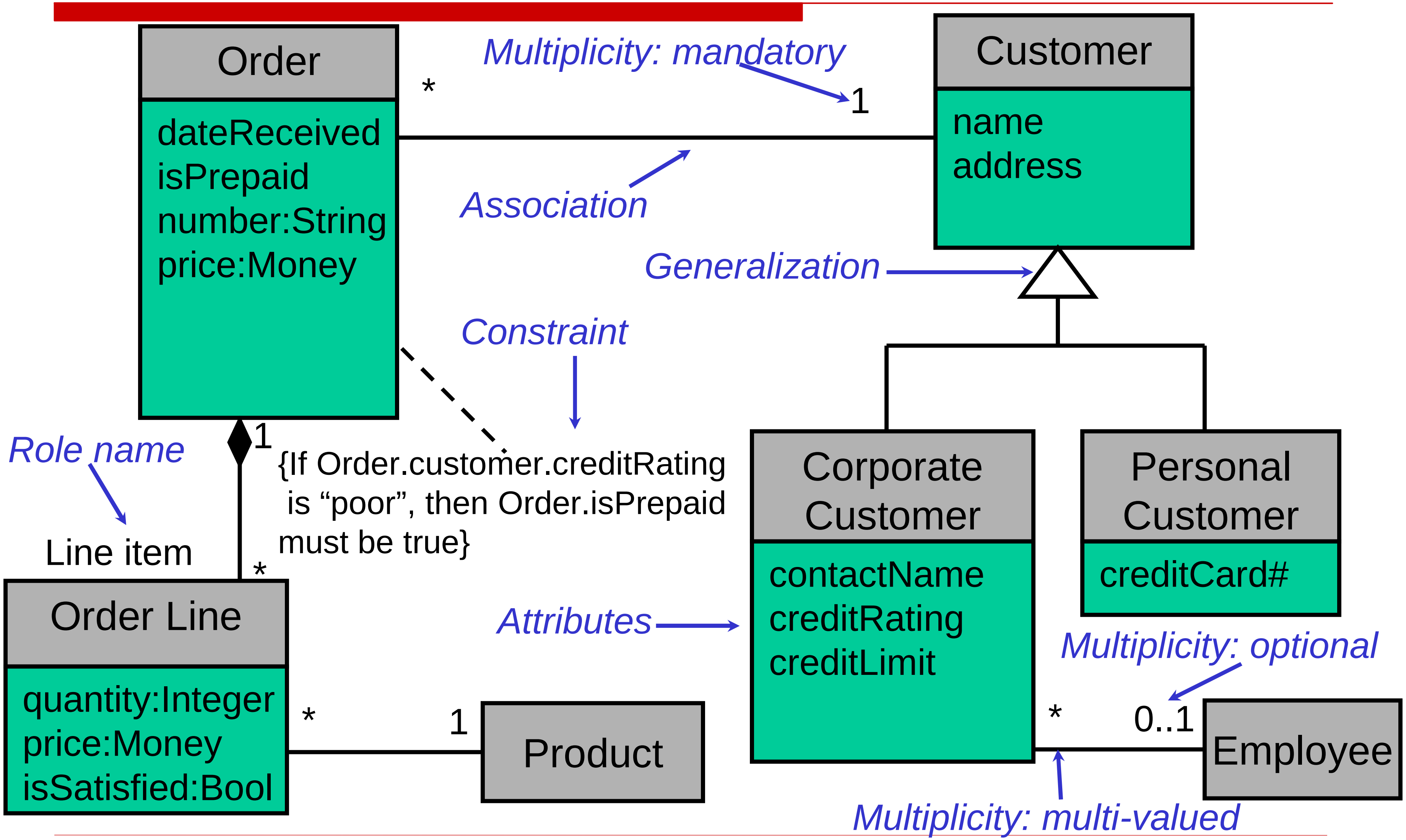
Order:
We want our orders to be lined up product by product.
Each line should contain the amount and the price of each product.

Example: Attributes & Operations



We have customers who order our products.
We distinguish corporate customers from personal customers, since corporate customers are billed monthly whereas personal customers need to prepay their orders with a credit card.
We want our orders to be lined up product by product.
Each line should contain the amount and the price of each product.

Example: Order - Full Class Diagram



Modeling System Behaviour

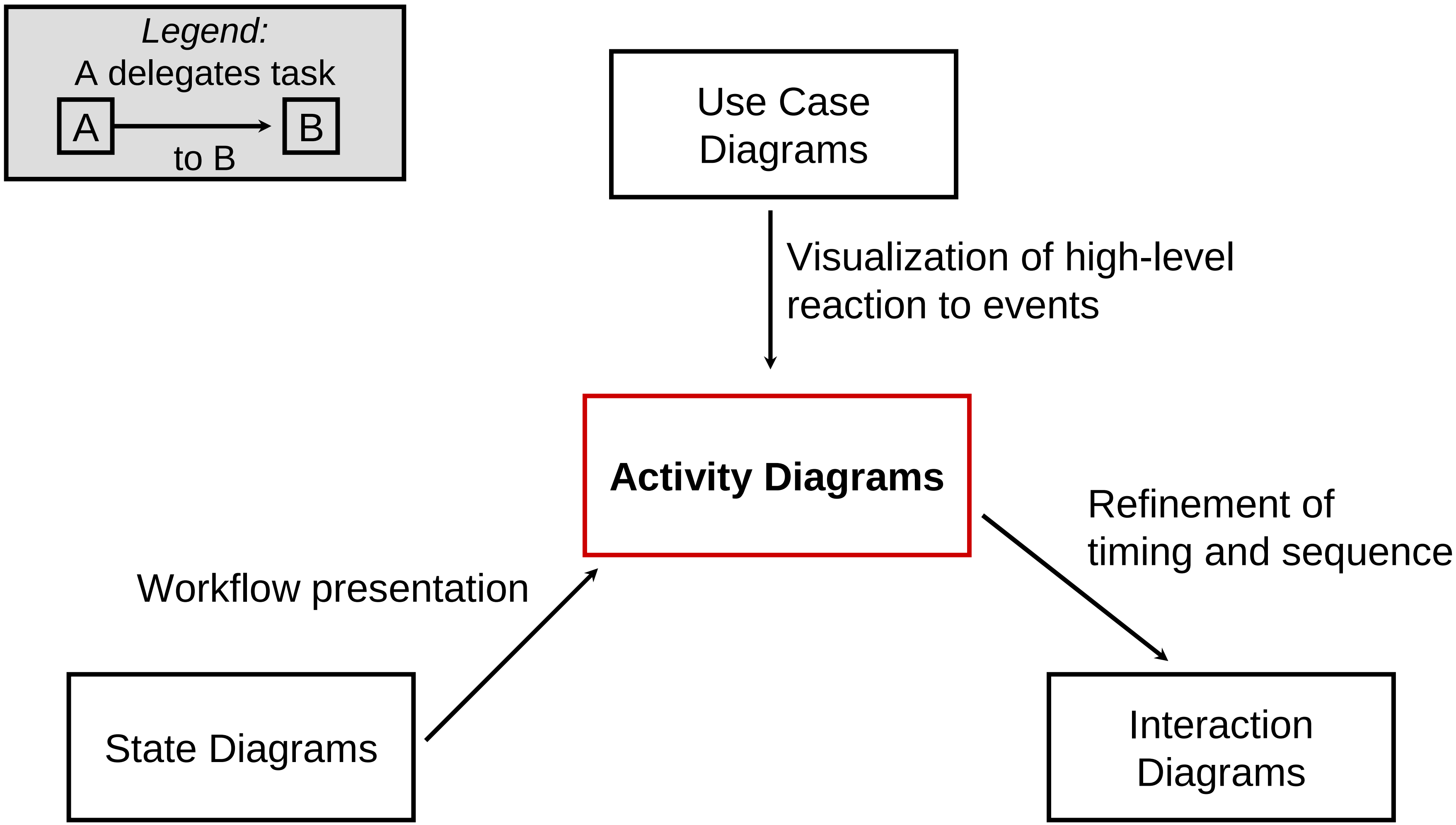
Modeling of System Behavior

- How do systems behave?
- Modeling system behavior is so much harder than modeling system structure
- Use cases ⇒
  - scenarios/processes → interaction diagrams
  - workflows → activity diagrams

Activity Diagrams

- Elements
  - States
  - Actions, Activities
  - Transitions
  - Branches, Merge
  - Concurrency, Synchronization
  - Swimlanes, Object Flow
  - Signals

Role of Activity Diagrams in UML



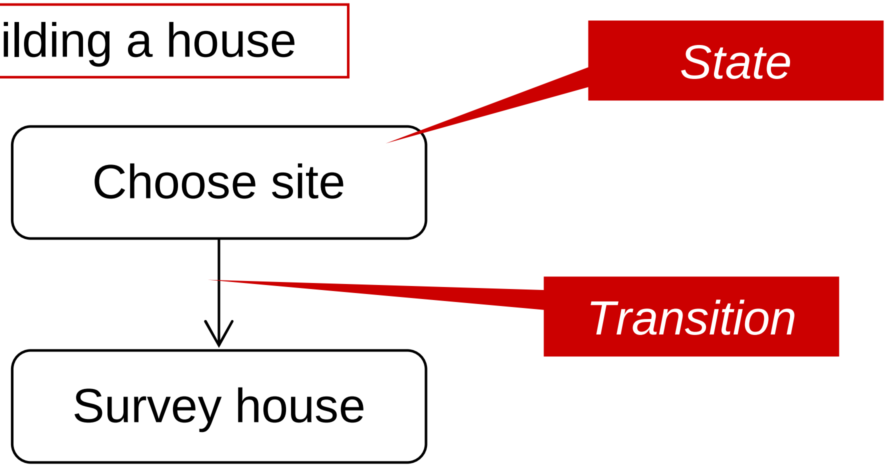
Specification of Behavior

- Computer Science developed several models for behavior specification
  - logic-oriented models: predicate transformers on pre- and post-conditions
  - graph-oriented models:
    - Petri nets
    - state machines
    - activity diagrams
- Goals
  - Specification of state changes of an object (or an interaction) triggered by an external event or a received signal.
  - Definition of protocols, i. e., legal sequences of operations of a class or an interface.

Activity Diagrams

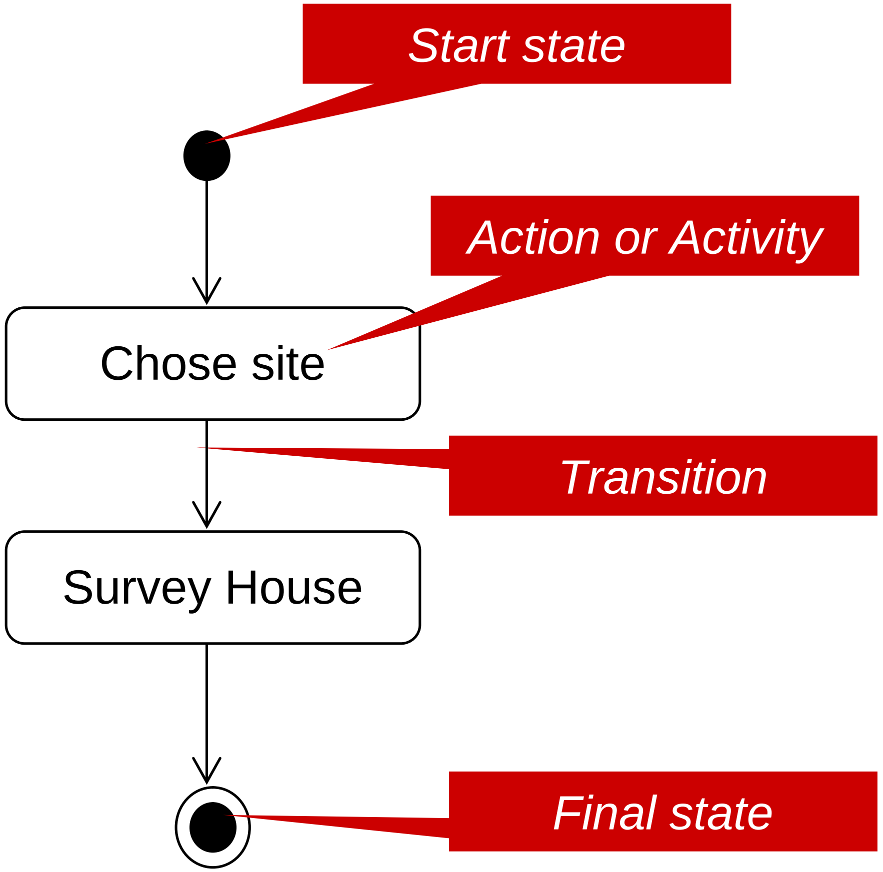
- The states are *action states* or *activity states* and the transitions are fired (triggered) by the termination of activities.
- In activity diagrams the concept of *state* does not refer to a static situation, but to named clusters of *acts*.

Example: Building a house



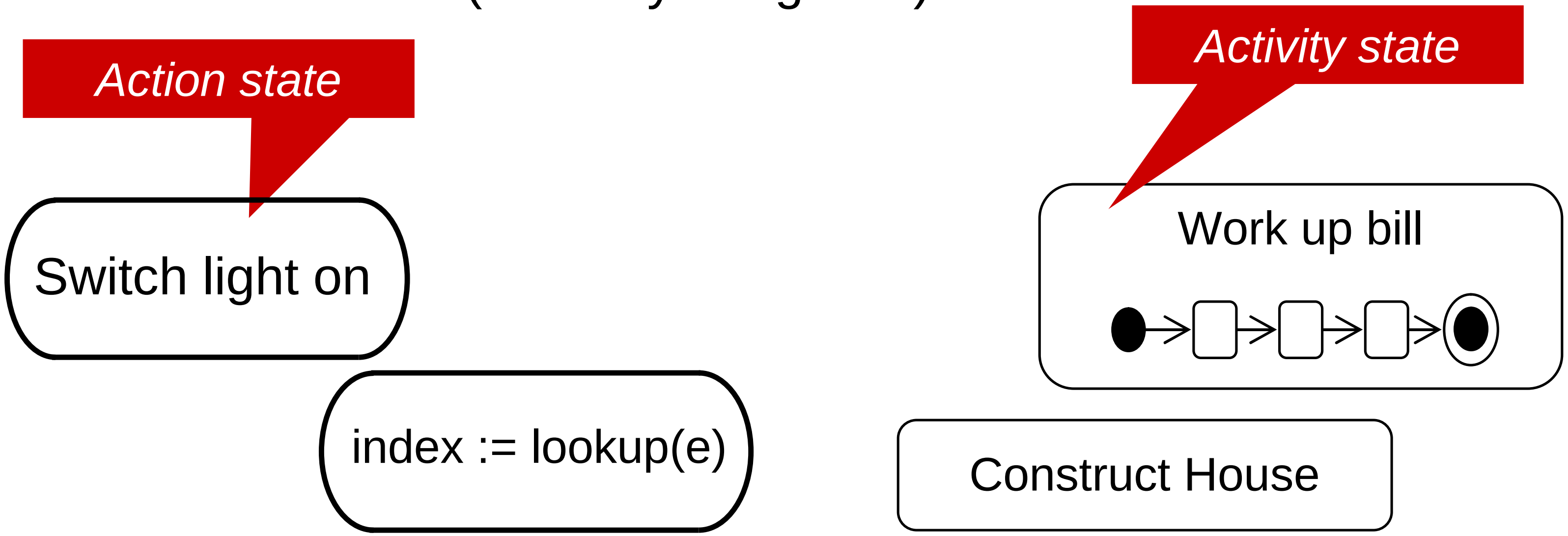
Transitions

- Transitions describe how to get from one state into another.
- A transition is executed when the previous action or activity is terminated.



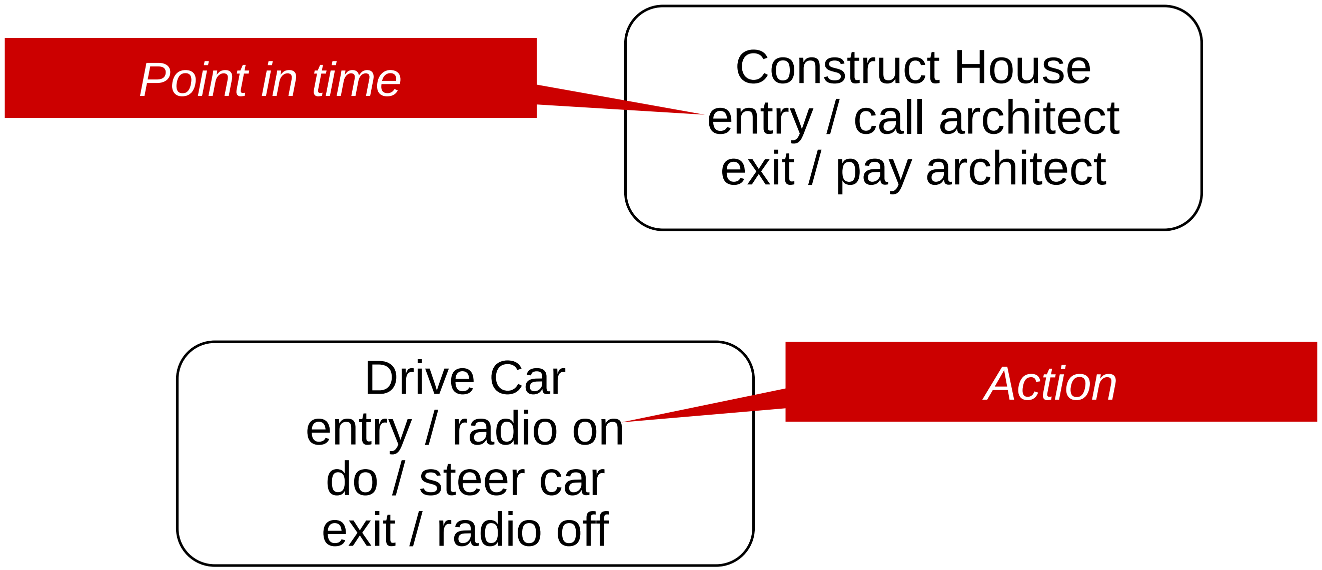
Action States and Activity States

- An action state in an activity diagram describes an atomic change of a system's state without temporary fine structure, e. g., an operation call or the calculation of a value. Action states cannot be decomposed.
- An activity state describes an enduring activity which can be interrupted and typically is described by a submachine (activity diagram).



Activity States: Actions

- In activity states actions can be executed at the beginning (entry point) or end (exit point) of an activity.



Additional Reference

- Book (from the “I want to model X, how do I do that with UML?” perspective)
  - G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley 1999