
6. Design Phase: Modeling of System Behavior

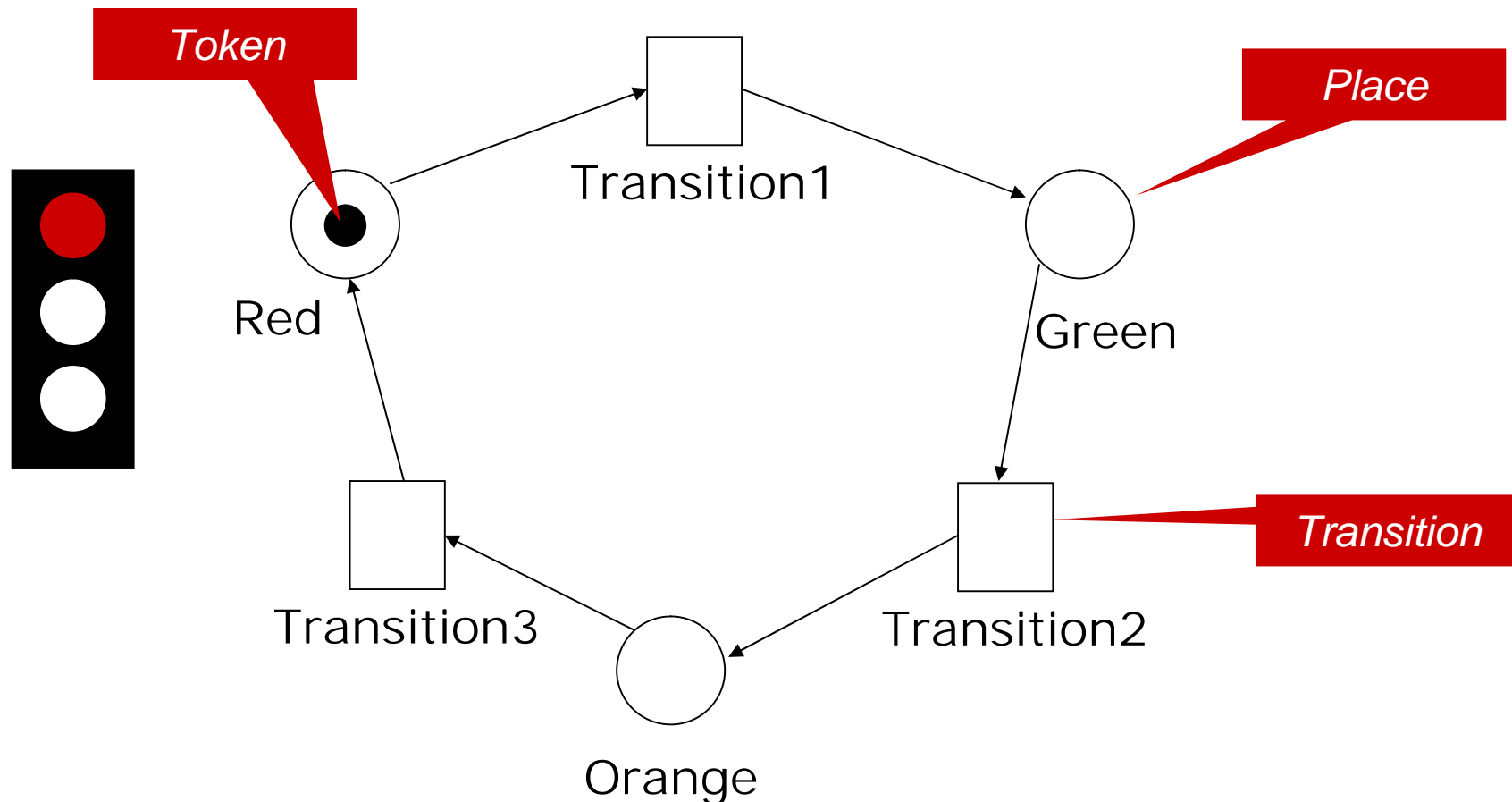
Modeling of System Behavior

- o How do systems behave?
- o Modeling system behavior is so much harder than modeling system structure
- o Use cases \Rightarrow
 - n scenarios/processes \rightarrow interaction diagrams
 - n workflows \rightarrow activity diagrams

Specification of Behavior

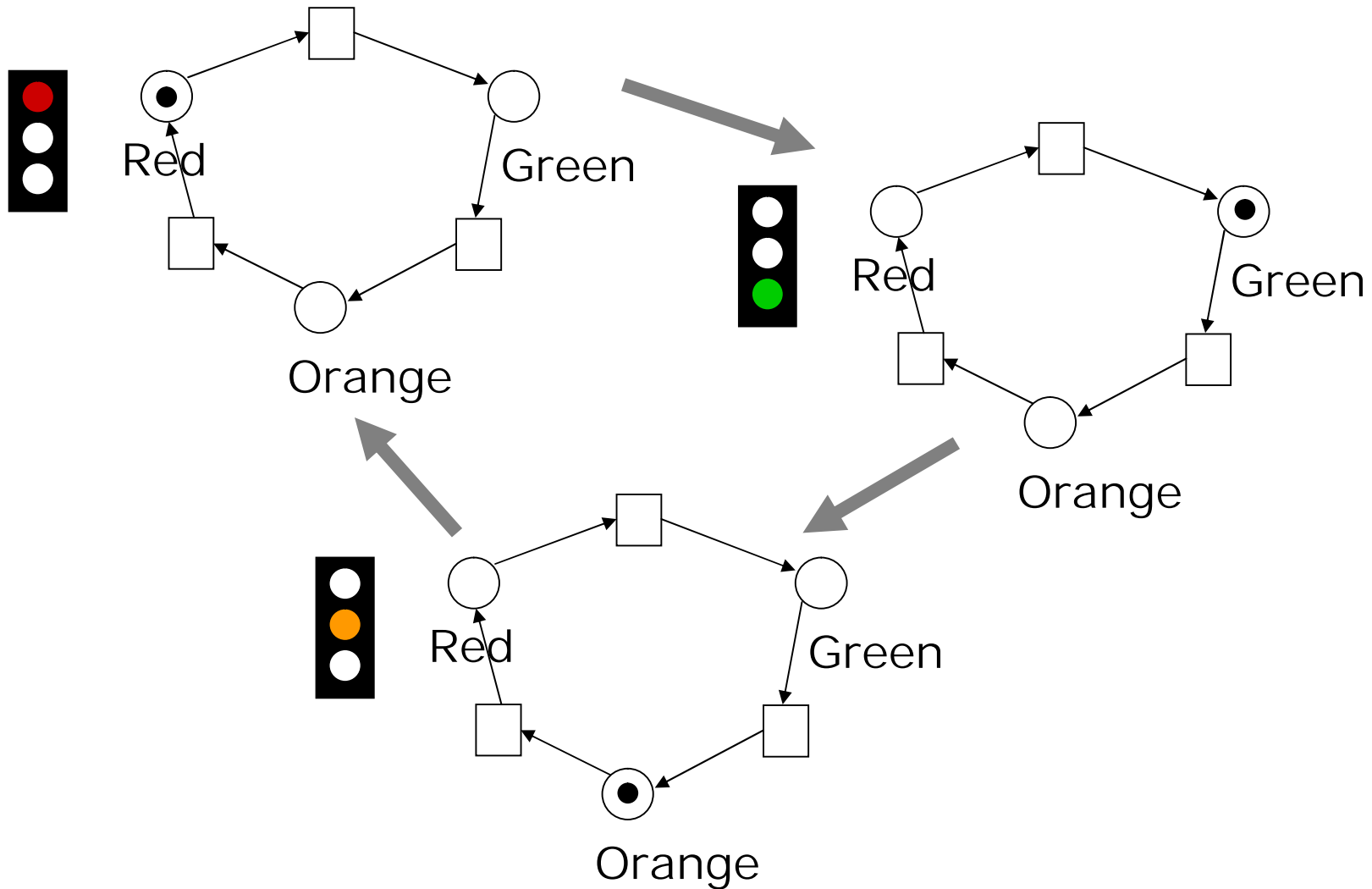
- o Computer Science developed several models for behavior specification
 - n logic-oriented models: predicate transformers on pre- and post-conditions
 - n graph-oriented models:
 - o Petri nets
 - o state machines
 - o activity diagrams
- o Goals
 - n Specification of state changes of an object (or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

Petri Net Example: Traffic Light



example from [1]

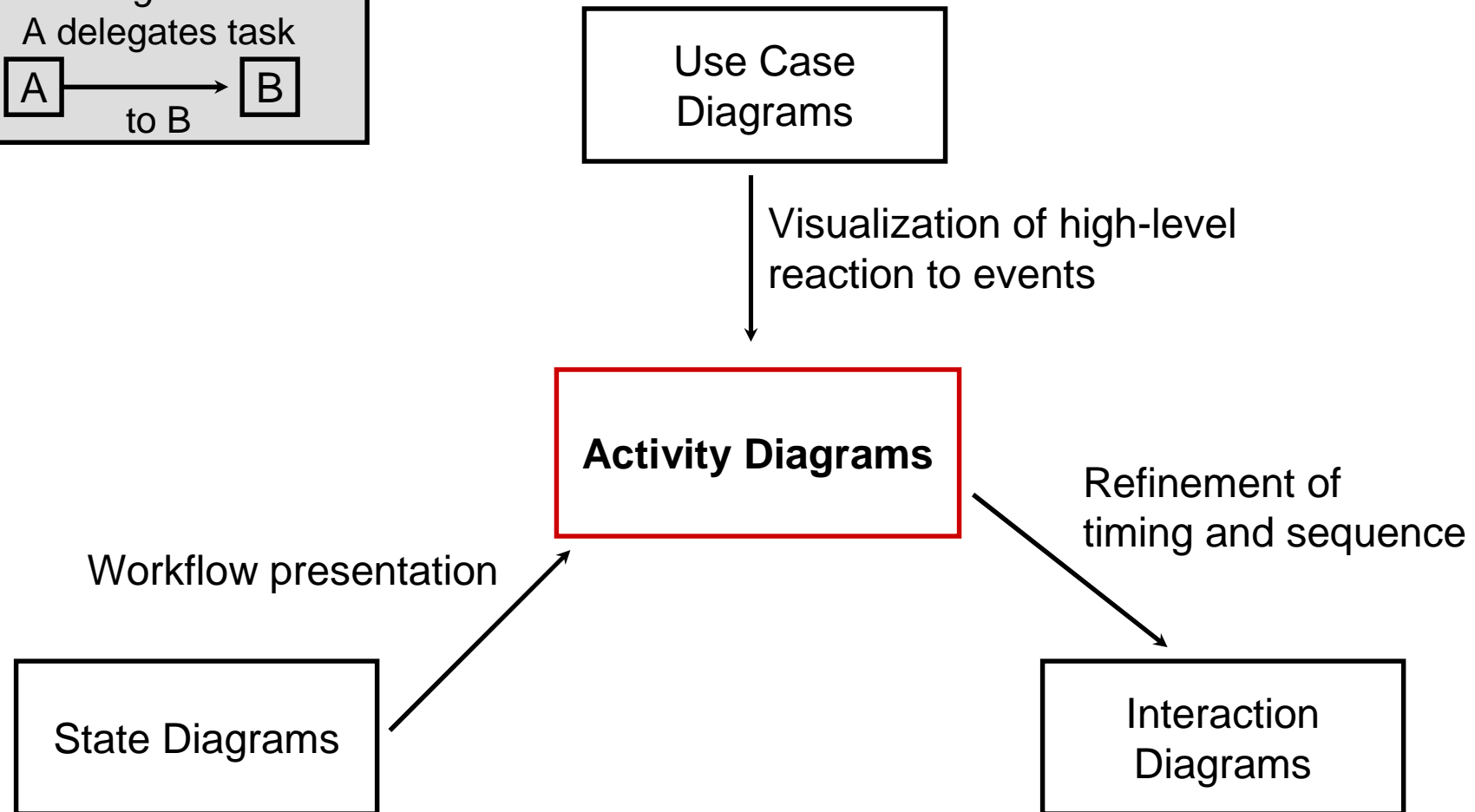
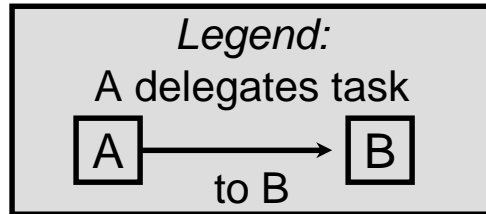
Petri Net Example in Action



Activity Diagrams

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

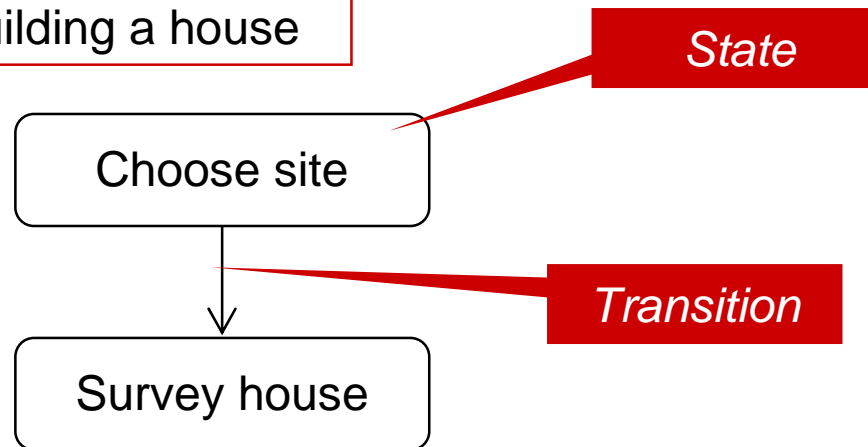
Role of Activity Diagrams in UML



Activity Diagrams

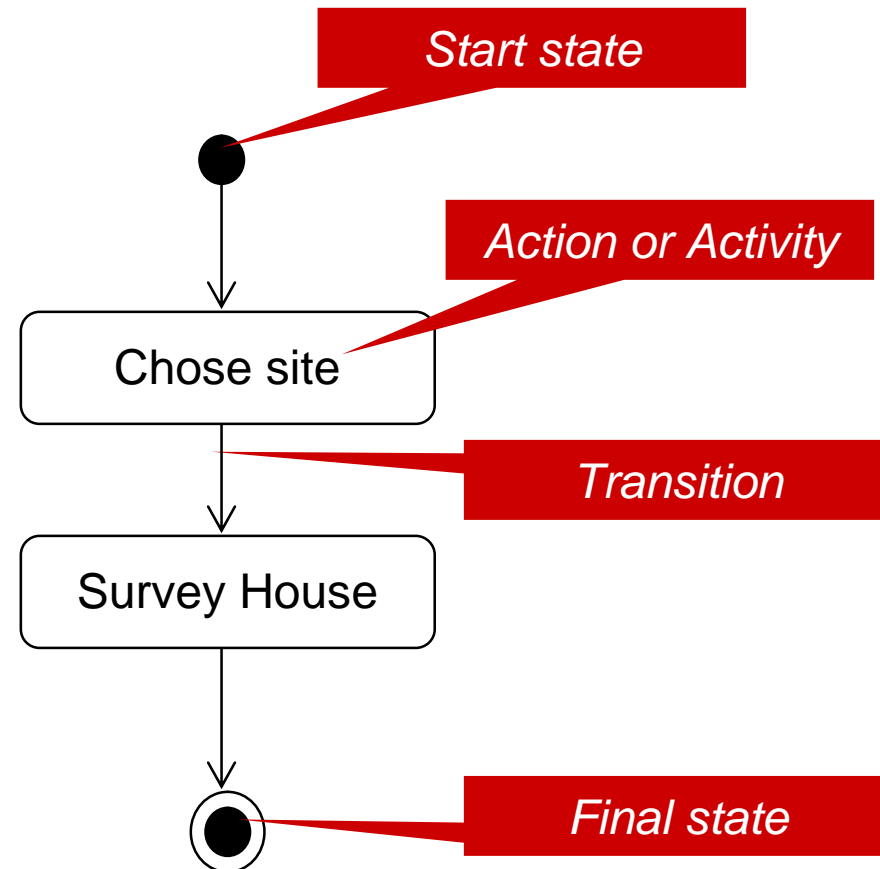
- The states are action states or activity states and the transitions are fired (triggered) by the termination of activities.
- In activity diagrams the concept of state does not refer to a static situation, but to named clusters of acts.

Example: Building a house



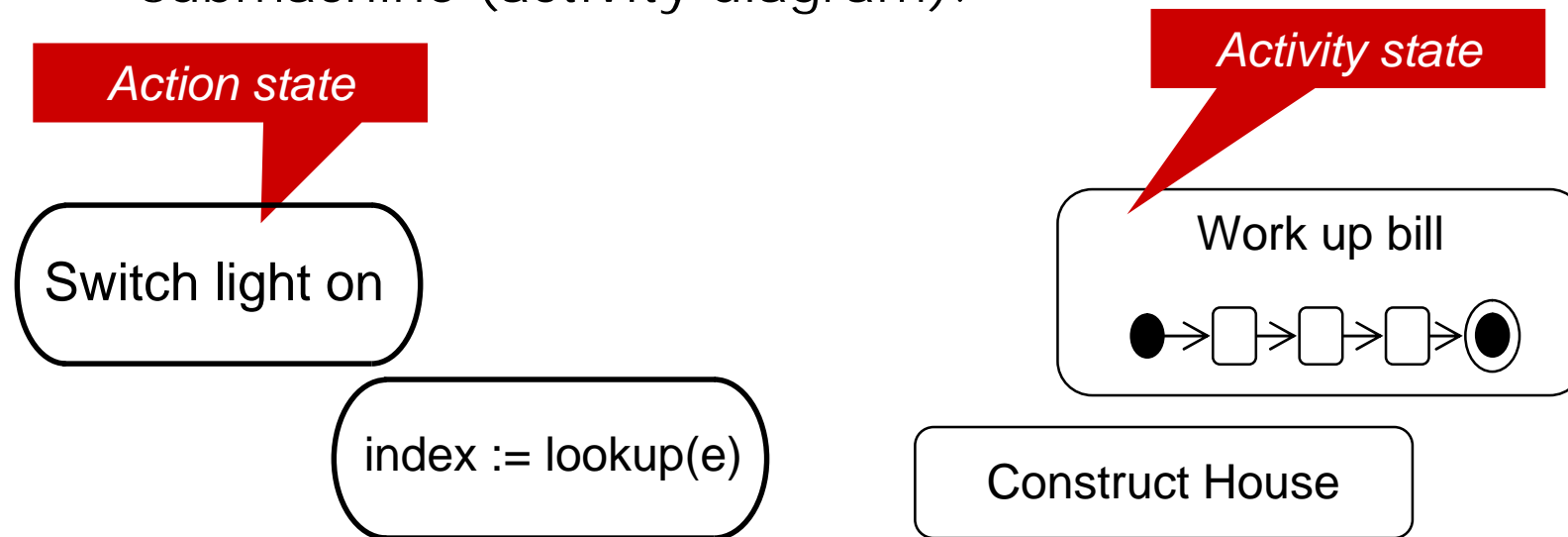
Transitions

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



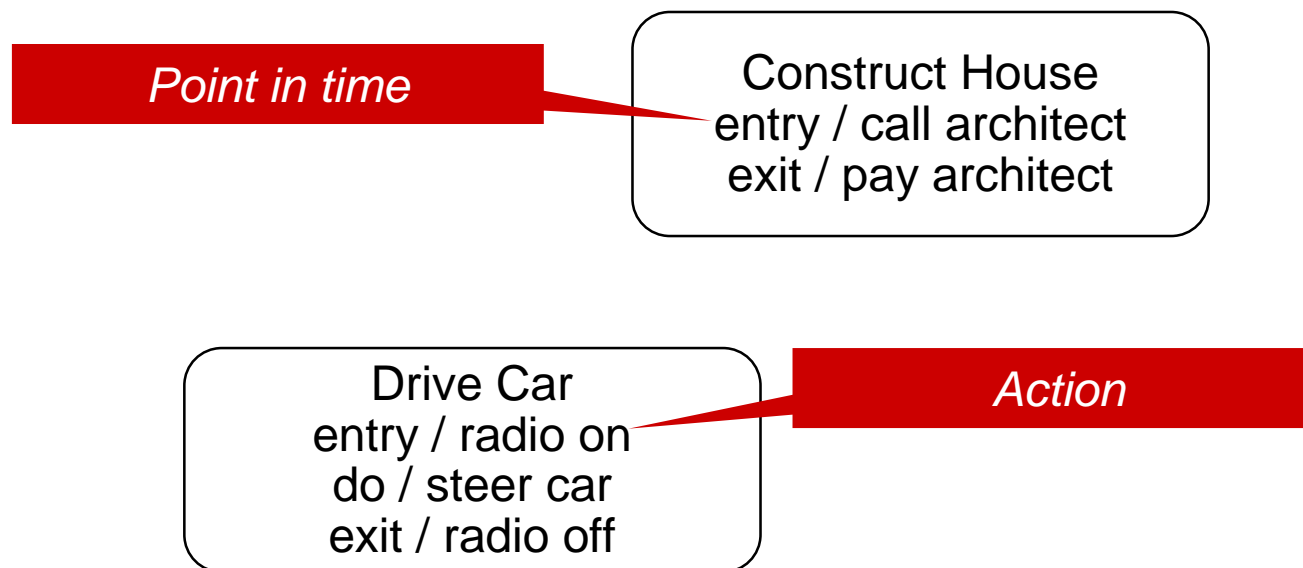
Action States and Activity States

- 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).



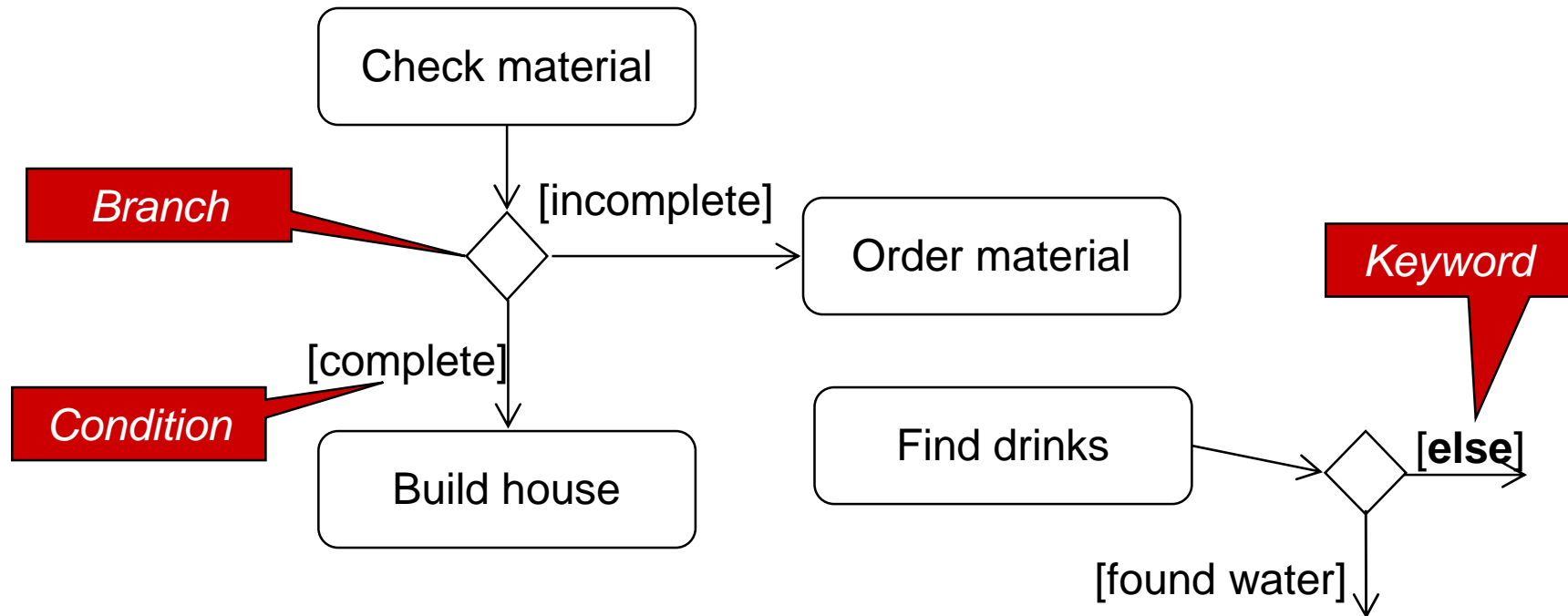
Activity States: Actions

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



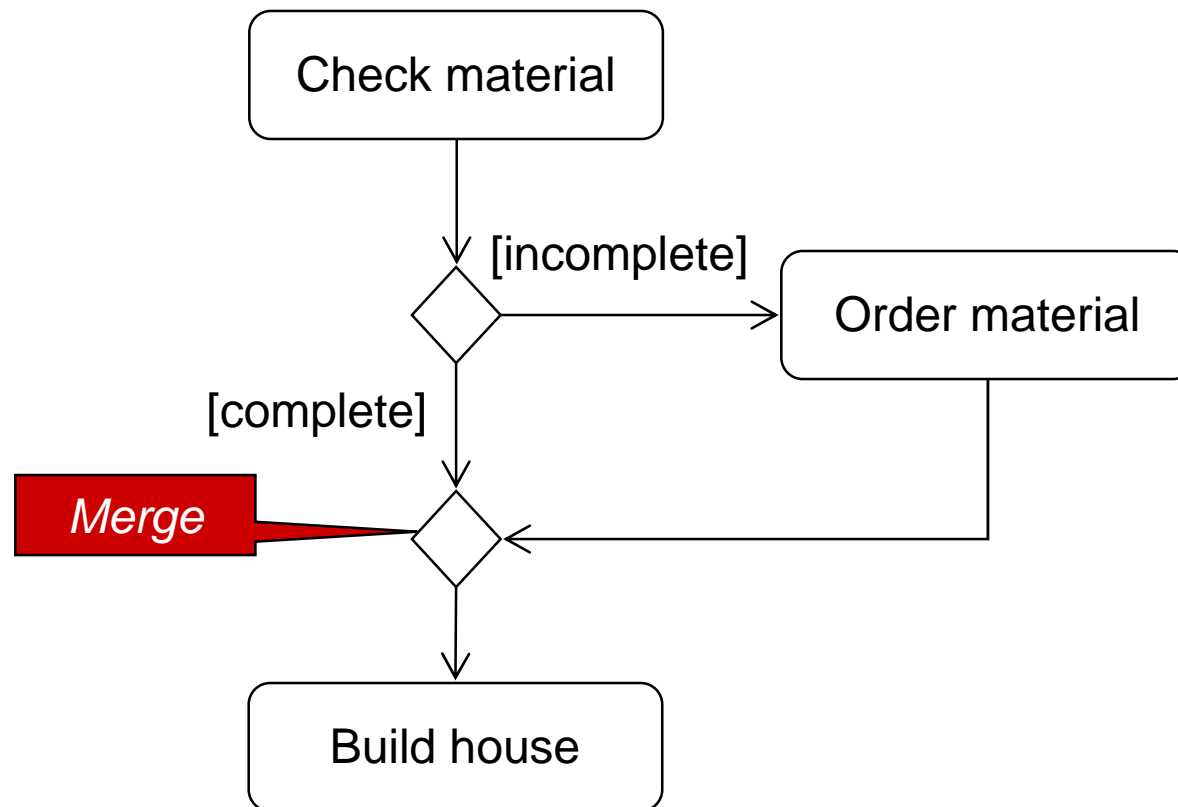
Branches

- o A branch appears, if a decision is to be made and depending on a condition several following actions/activities are to be initiated.



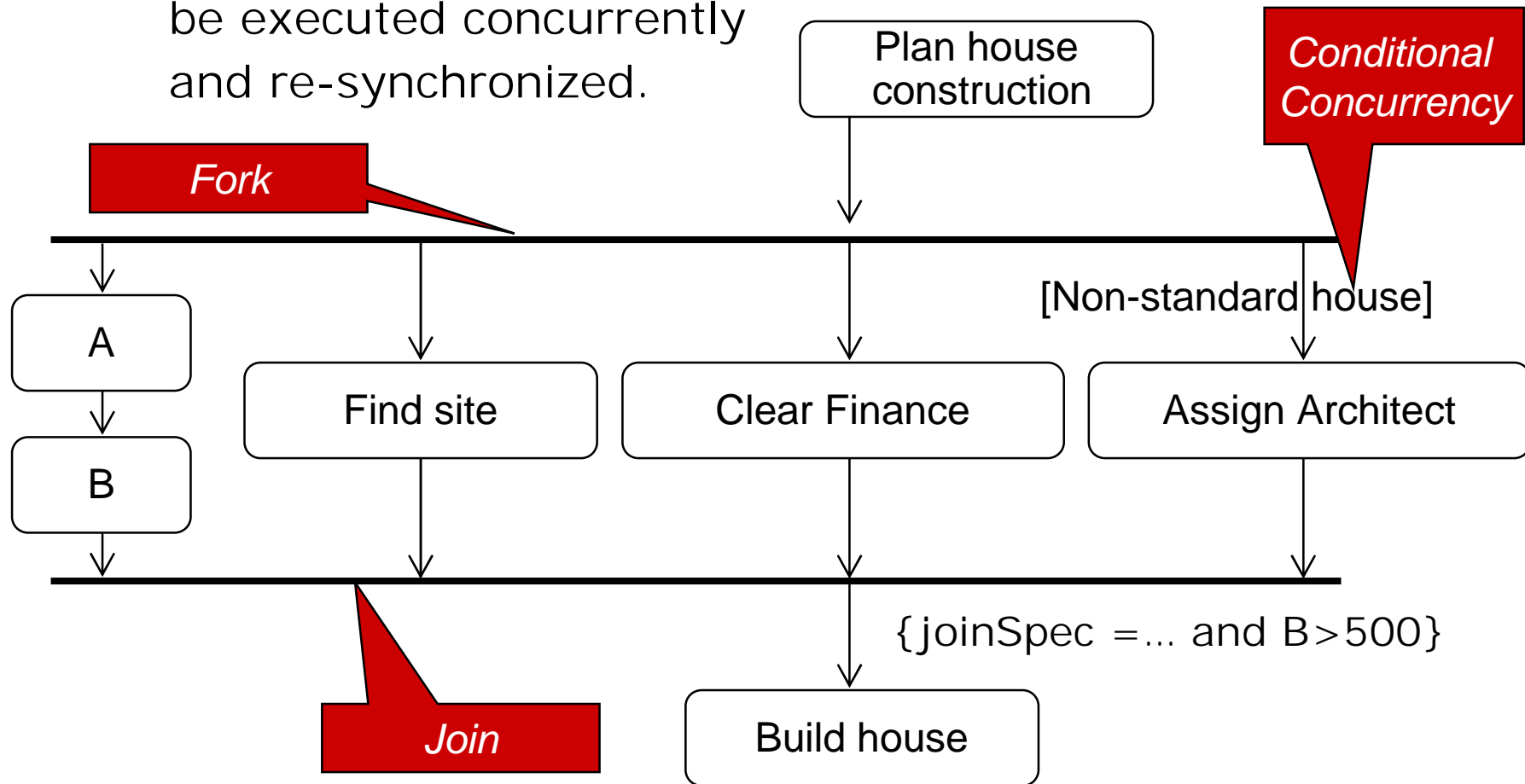
Merge

- A merge merges optional branches in activity diagrams.



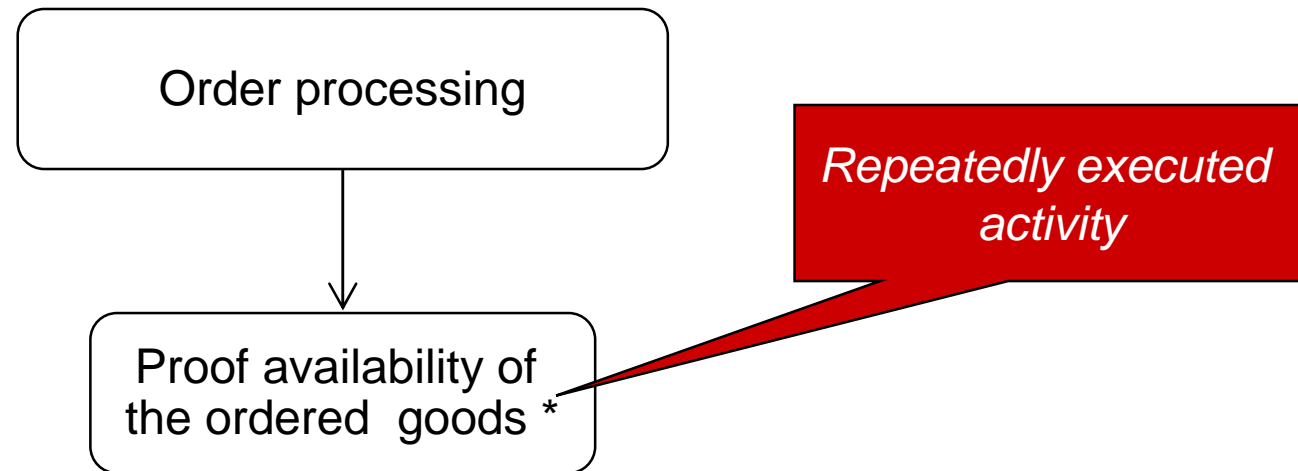
Concurrency

- Actions and Activities can be executed concurrently and re-synchronized.



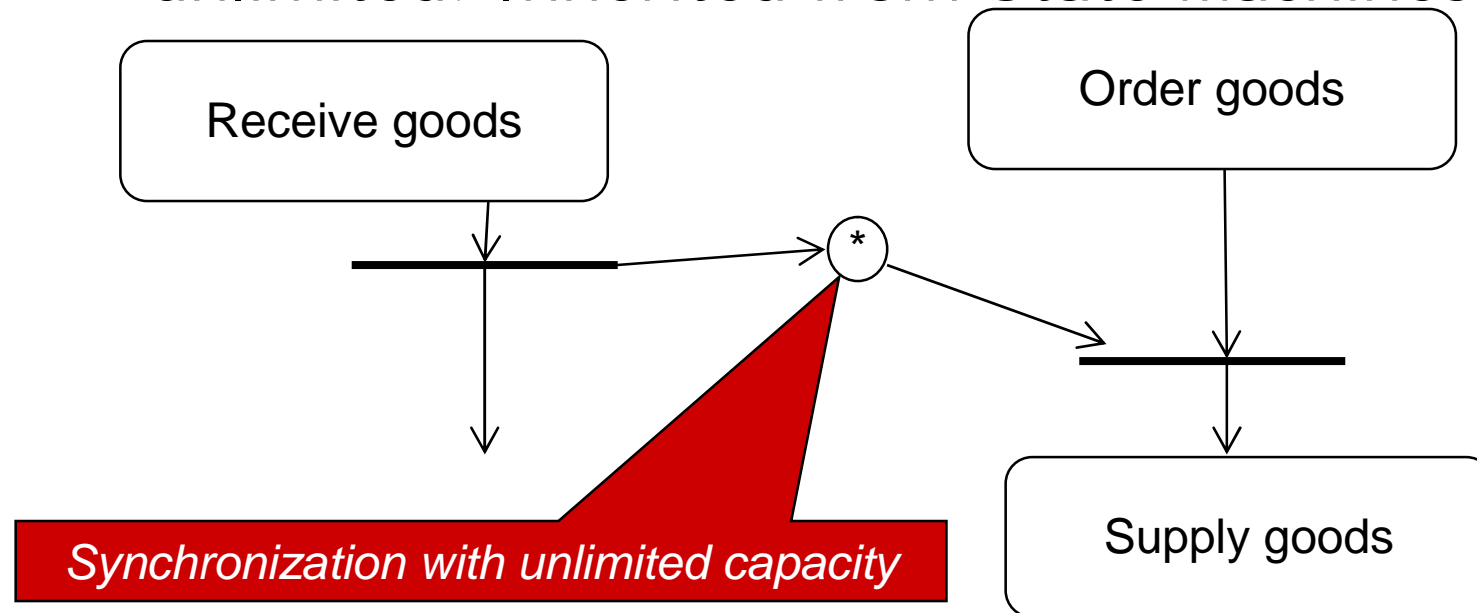
Dynamic Concurrency

- o An activity is executed concurrently multiple times where the number of concurrent activities depends on a list of arguments.



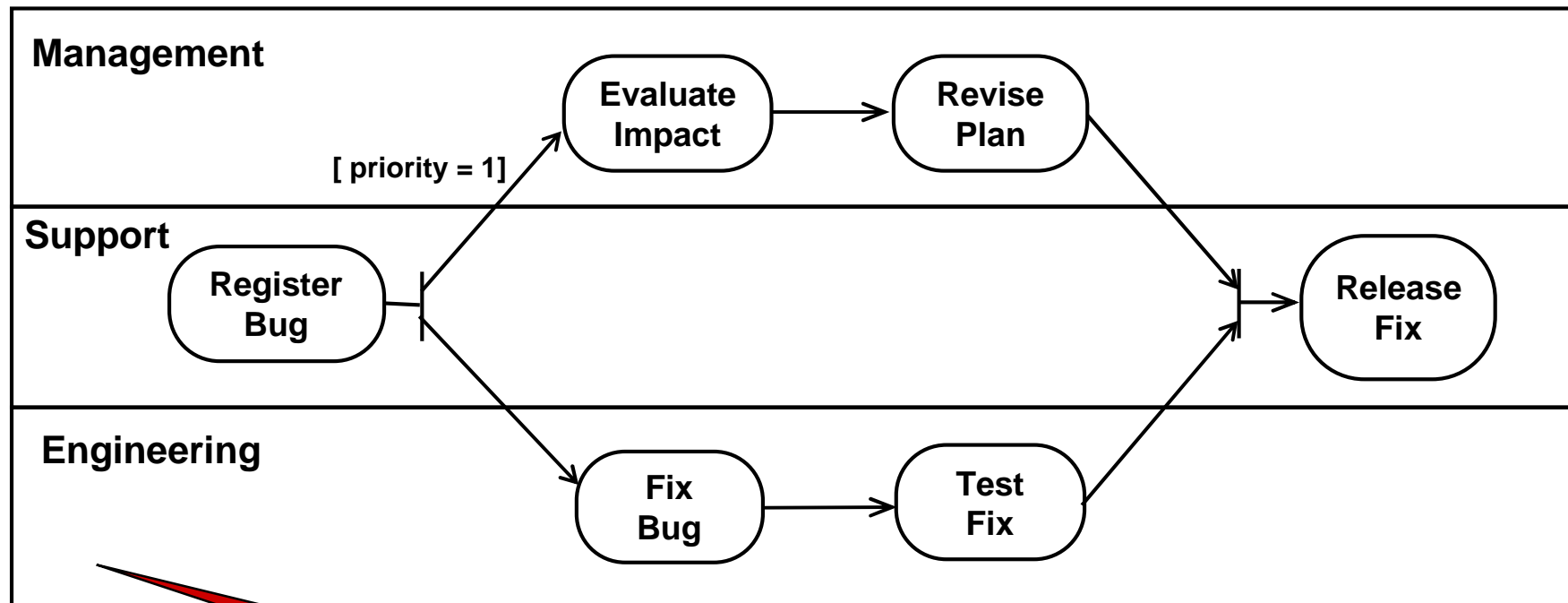
Synchronization of Concurrent Activities

- Concurrent activities can be synchronized by a synchronization state. The state contains implicit a counter representing the number of waiting activities. The counter can be limited or unlimited. Inherited from State Machines



Swimlanes

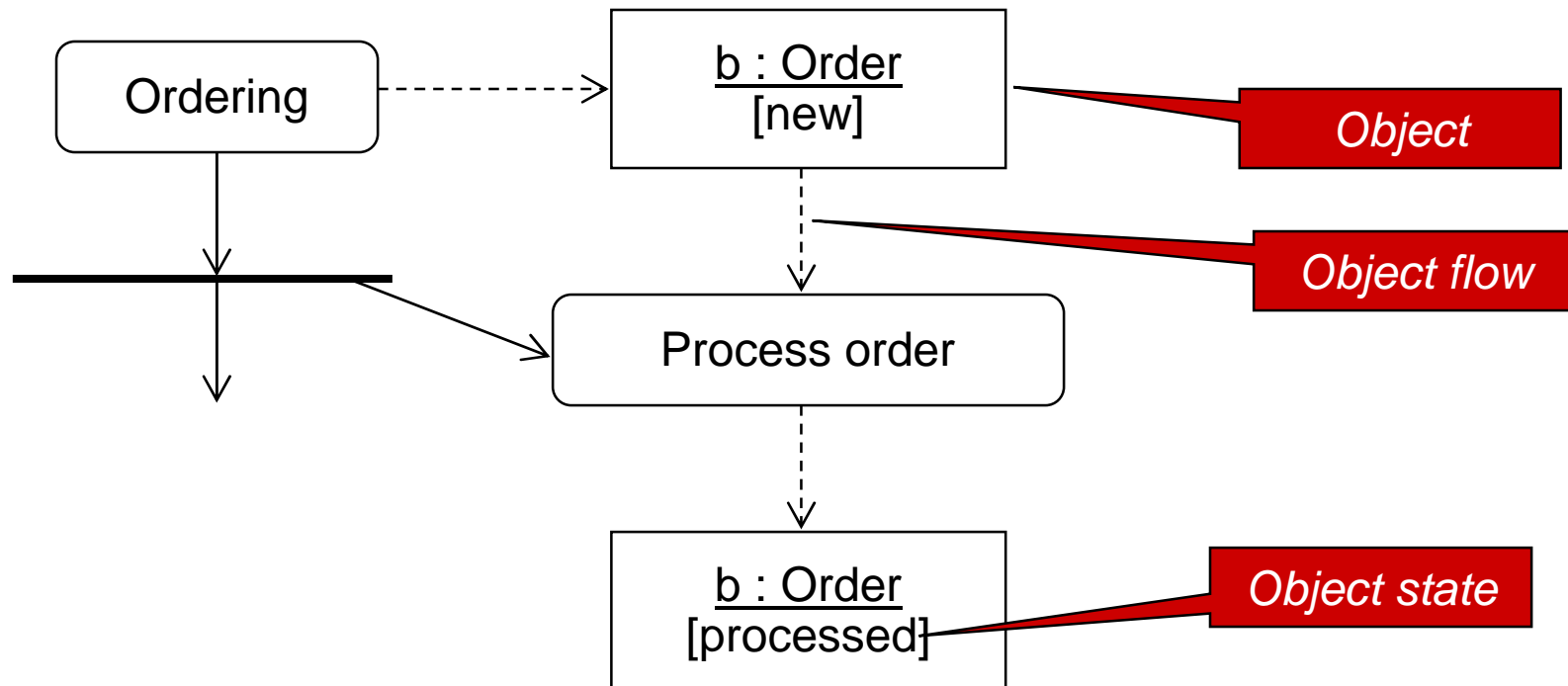
- Swimlanes structure actions and activities and show them clustered by execution unit (object or a class, mostly concurrent to other actions/activities).



Swimlane

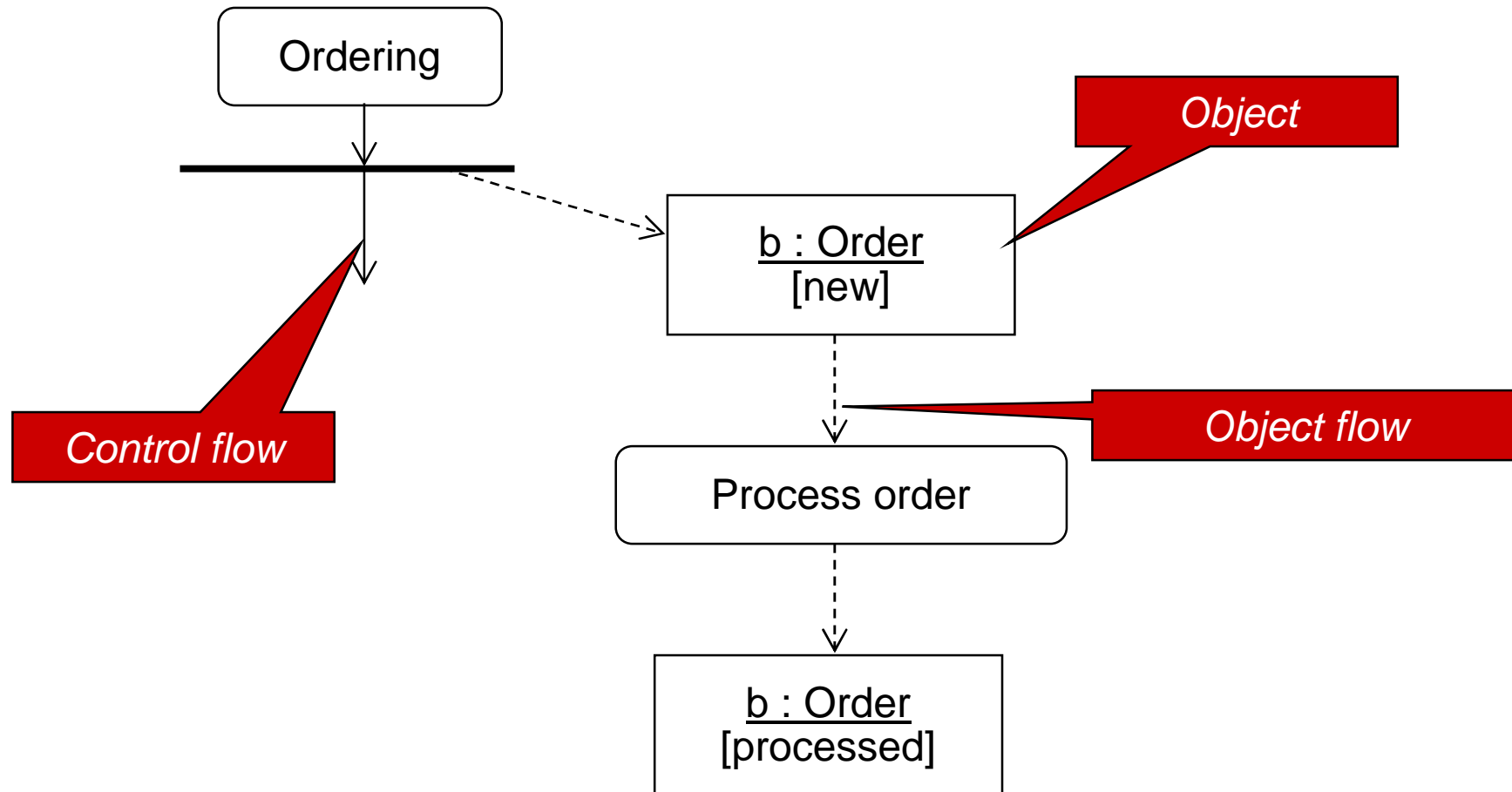
Object Flow

- o An object flow shows objects with their states being generated or used by actions or activities in activity diagrams.



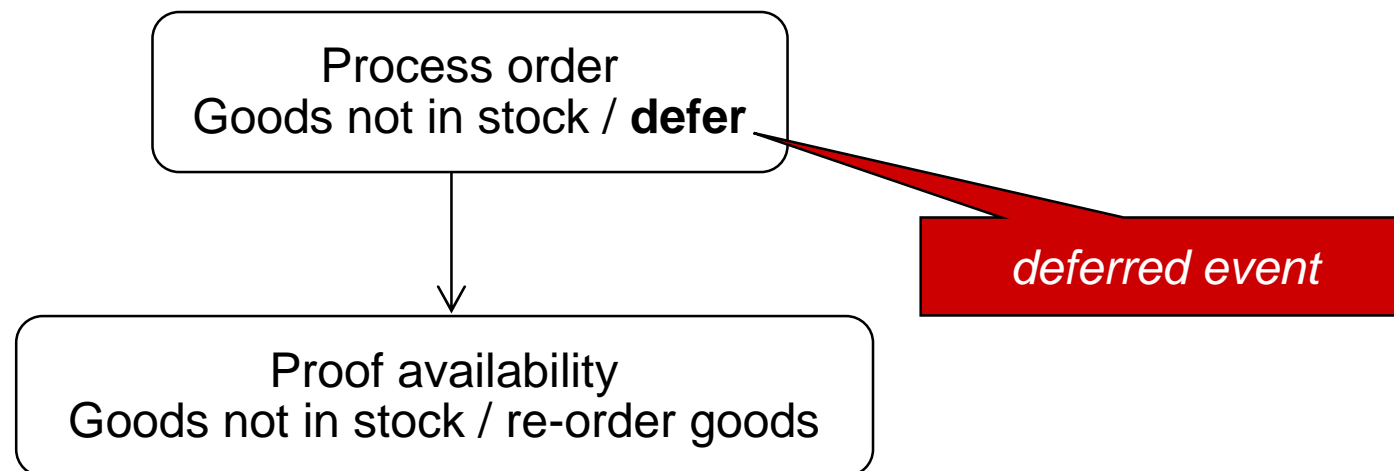
Object and Control Flow

- Object flow is combined with control flow.



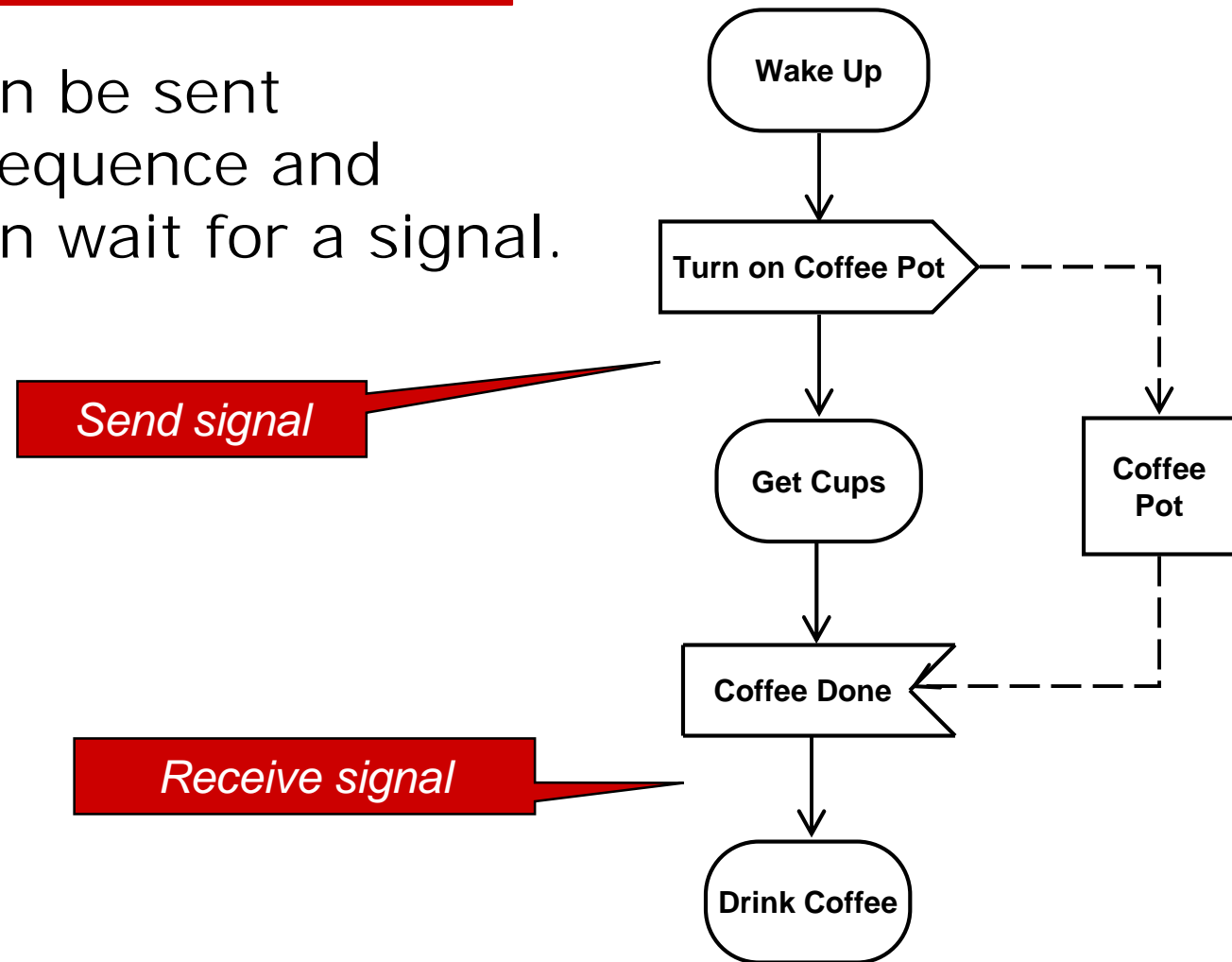
Deferred Events

- Events which cannot be reacted upon in the current action, can be deferred (queued).
- Deferred events can later lead to a reaction. Other events expire if they are not consumed immediately.



Sending and Receiving Signals

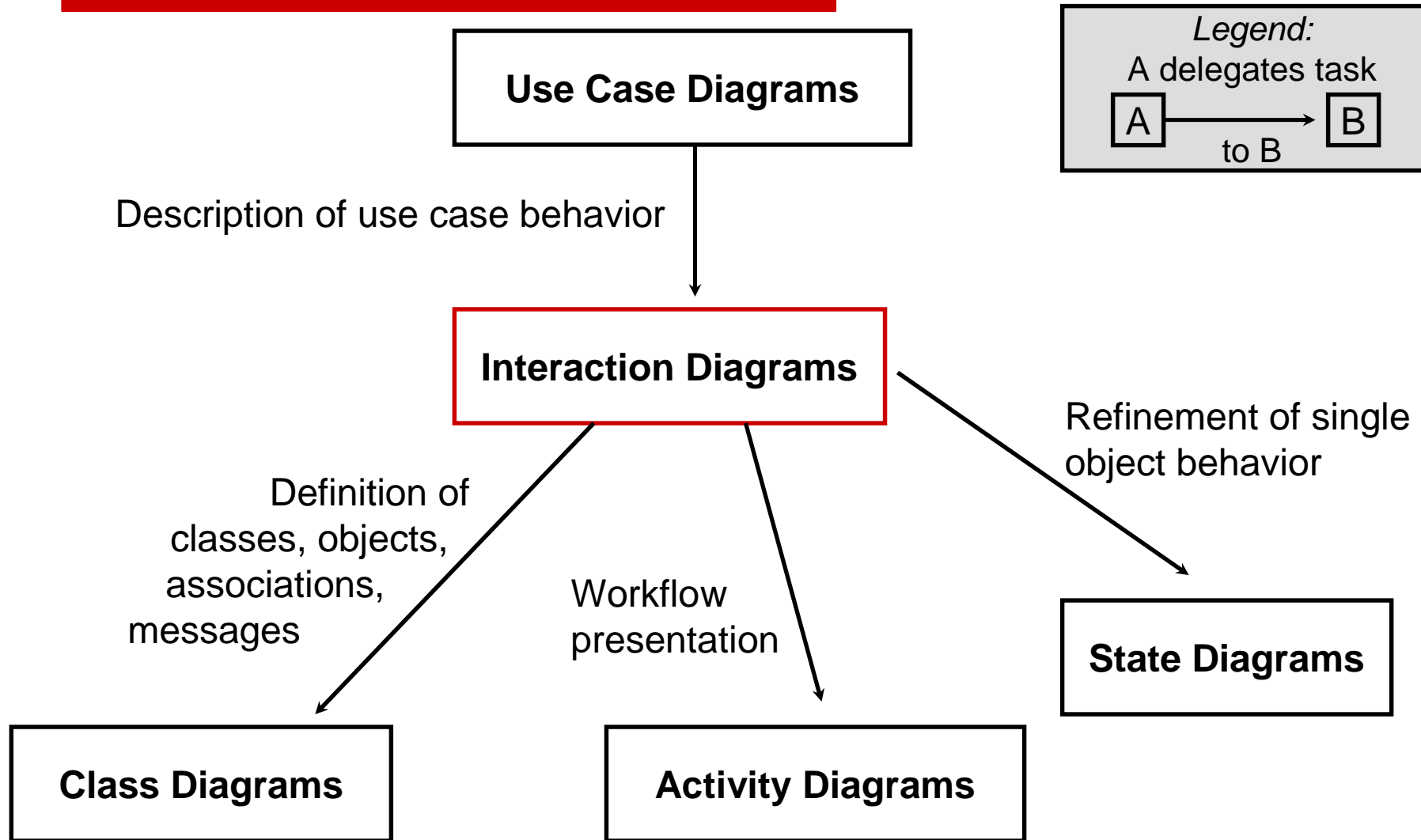
- Signals can be sent during a sequence and objects can wait for a signal.



On Interaction Modeling

- o Need to model dynamic aspects of interaction between objects:
 - n specifies details of system functionality;
 - n functionality is realized as interaction (message passing) between objects;
 - n uses the elements specified in structure diagrams (e.g. instances of classes).
- o Individual interaction diagrams usually show a much smaller part of the system than static diagrams.

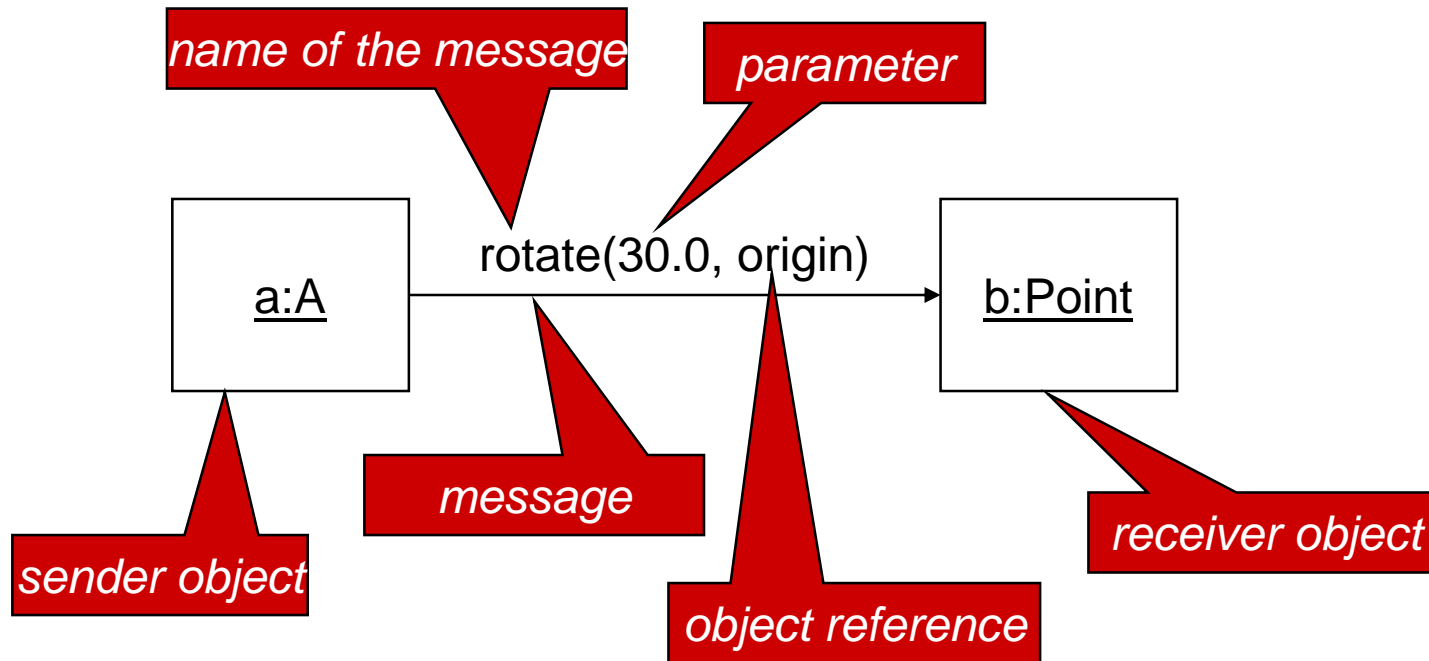
Role of Interaction Diagrams in UML



Interaction Diagrams in UML

- o Interaction diagrams are a series of diagrams describing the behavior of an object-oriented system with object interaction.
 - n A sequence diagram stresses the temporal sequence of object interactions.
 - n A collaboration diagram stresses the objects taking part in the object interaction.
- o For the description the following terms are used:
 - n message (asynchronous signal and synchronous method call)
 - n activation of an object
 - n active object

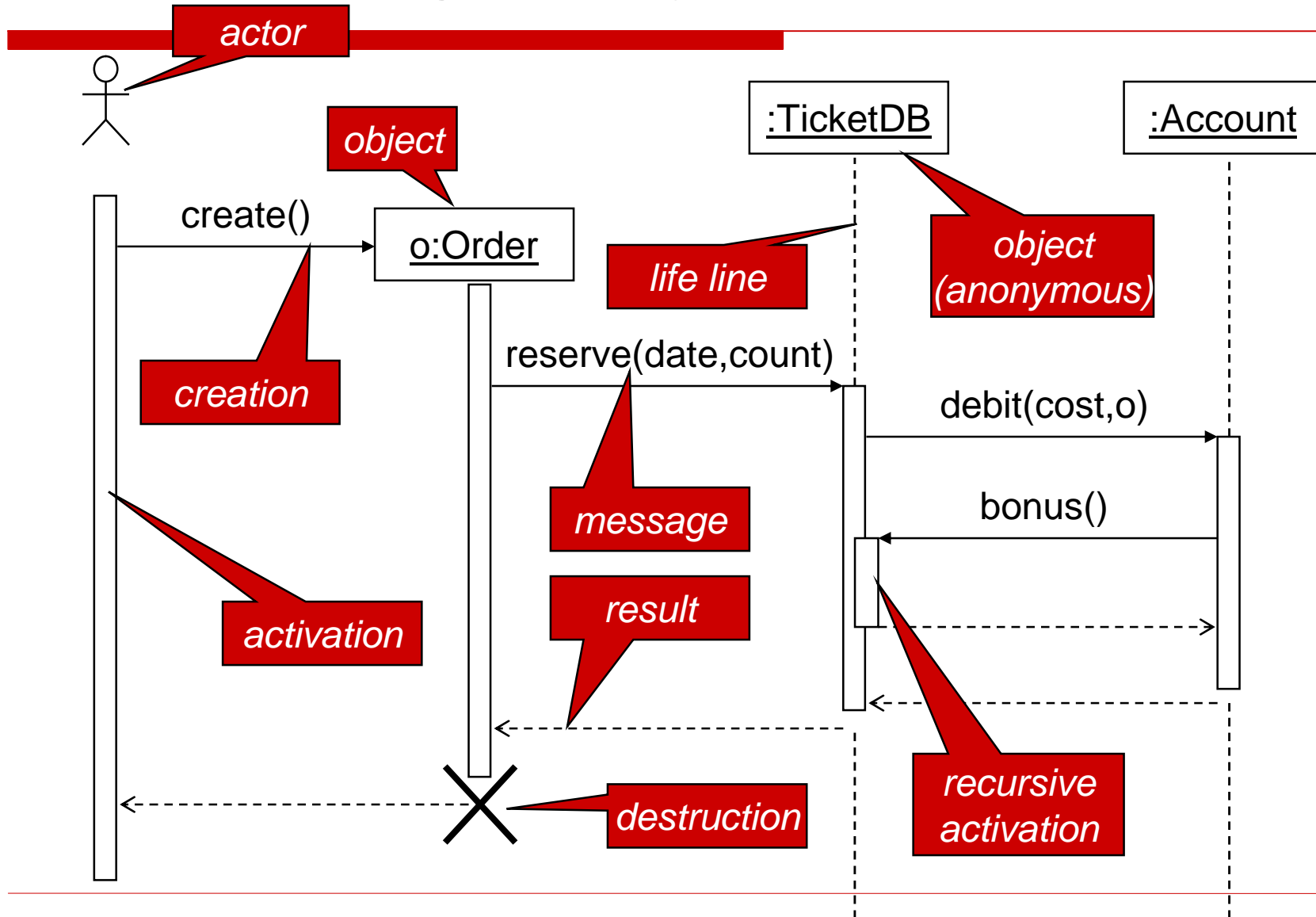
Conceptual Modeling of Interactions



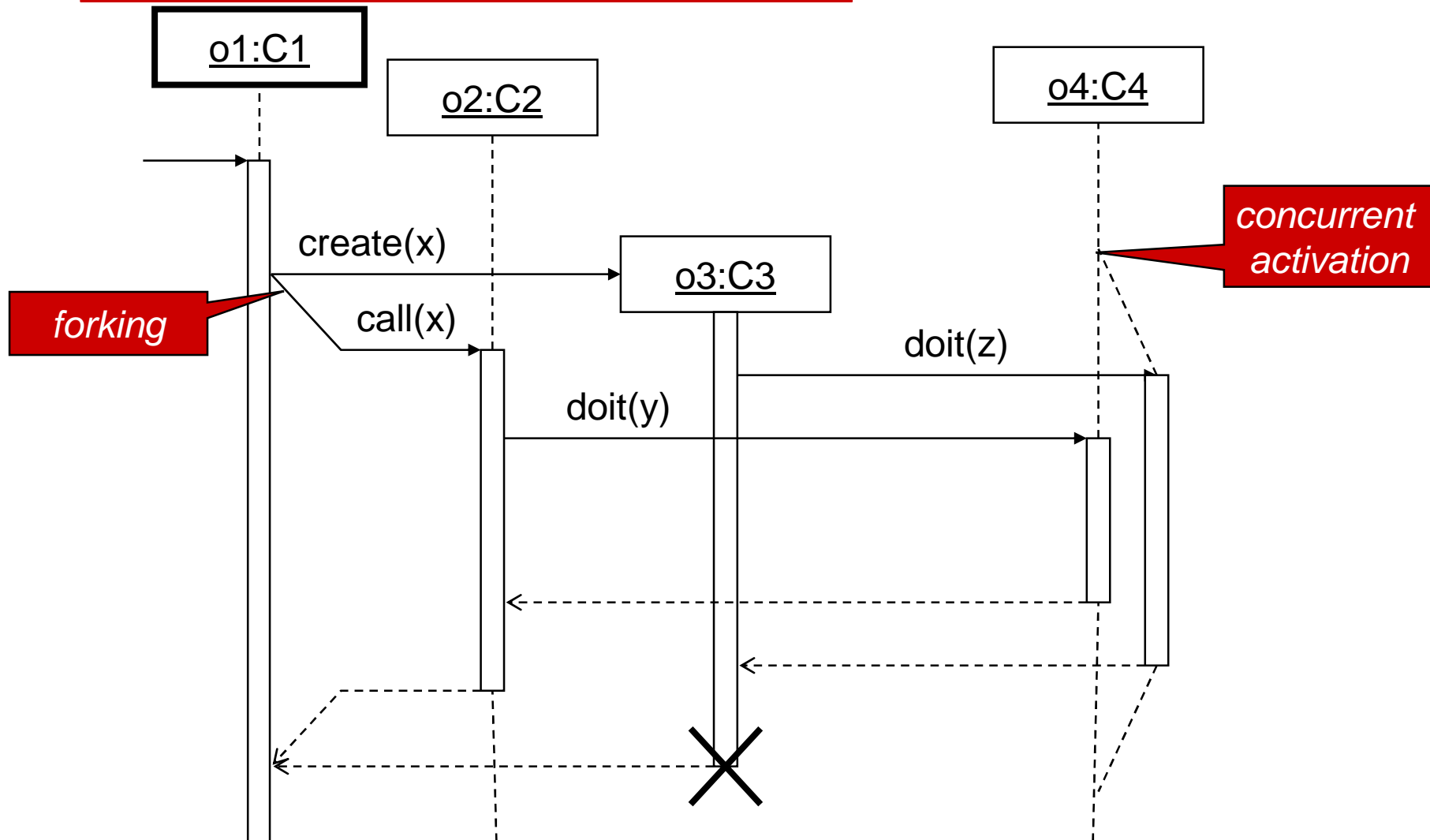
Synchronous (Procedural) Interaction

- o Synchronous interaction
 - n message sent between objects
 - n sending object must be activated and activates the receiving object. The sender object blocks (i.e., waits) until control returns.
- o Asynchronous interaction
 - n objects send signals
 - n sender stays active as well but does not block (i.e., continues)
 - n receiver consumes signal and is activated in parallel

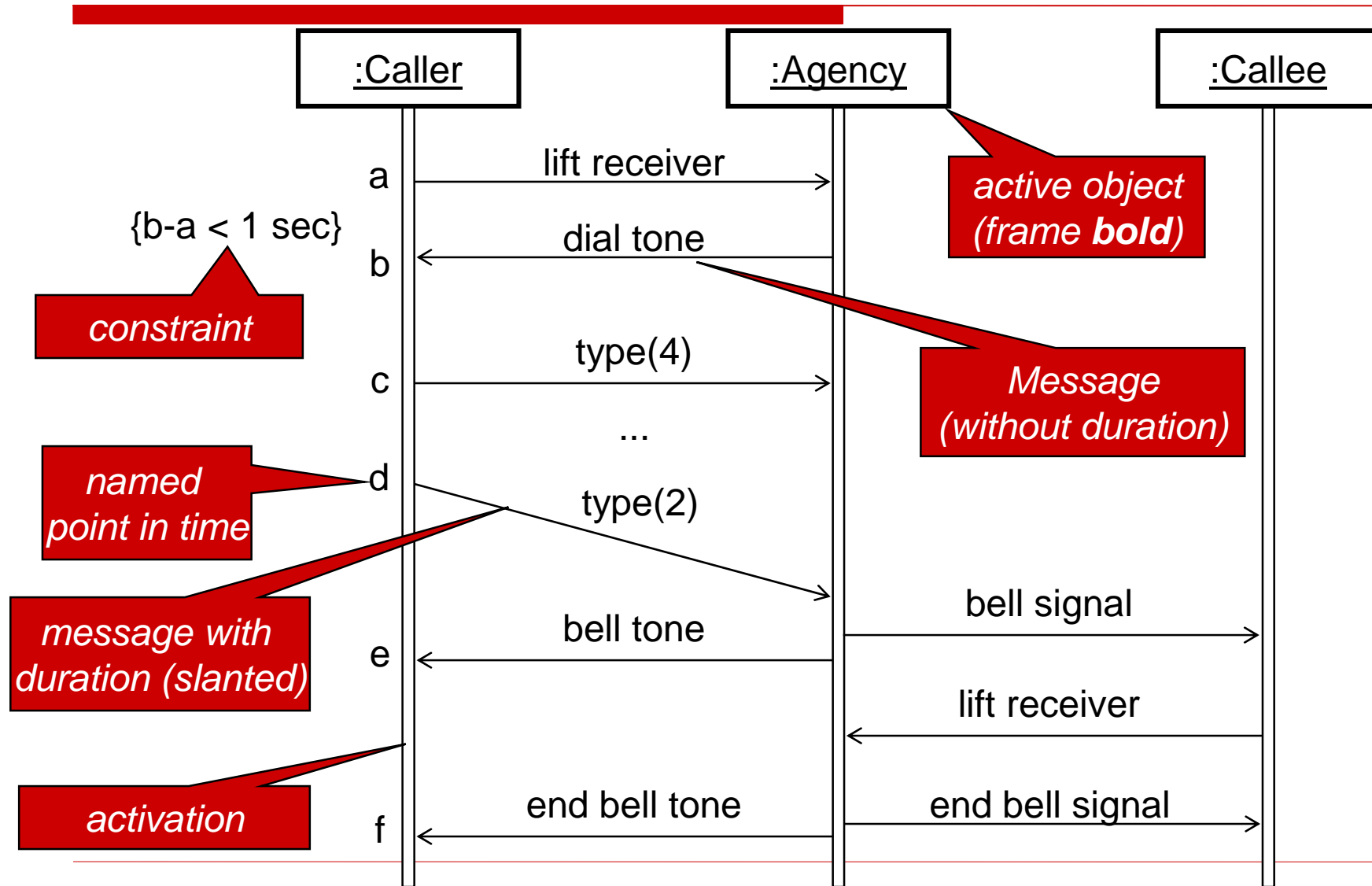
Sequence Diagram: Synchronous Interaction



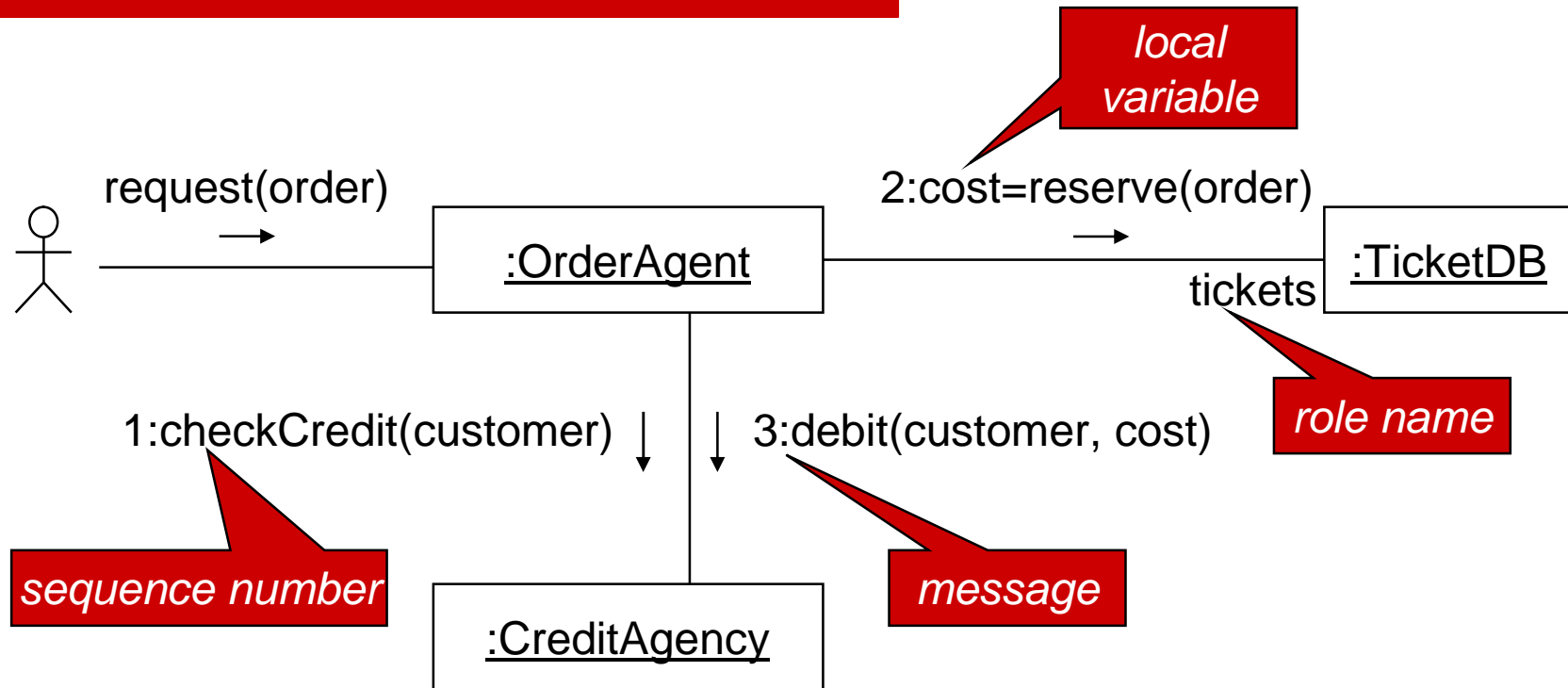
Sequence Diagram: Details



Sequence Diagram: Asynchronous Interaction



Collaboration Diagram: Example



Sequence numbers denote chronology of events.

- Simple scheme: just number messages
- Decimal scheme: nested numbering of messages where call-stack depth is visible

Message Labels

Message	Semantics
2: display(x, y)	Simple message with sequence number
1.2.1: p=find(name)	nested call with decimal sequence number
[x < 4] 2: invert(rect)	conditional message (x less than 4) with simple sequence number
3.1*: update	iteration
3.2* [i=1..8]: draw(v[i])	iteration

State Diagrams

- o Elements:
 - n States
 - n Transitions
 - n Events
 - n Composition
 - n Concurrency
 - n Branch
 - n History

Intra-Class Behaviour: State Machines

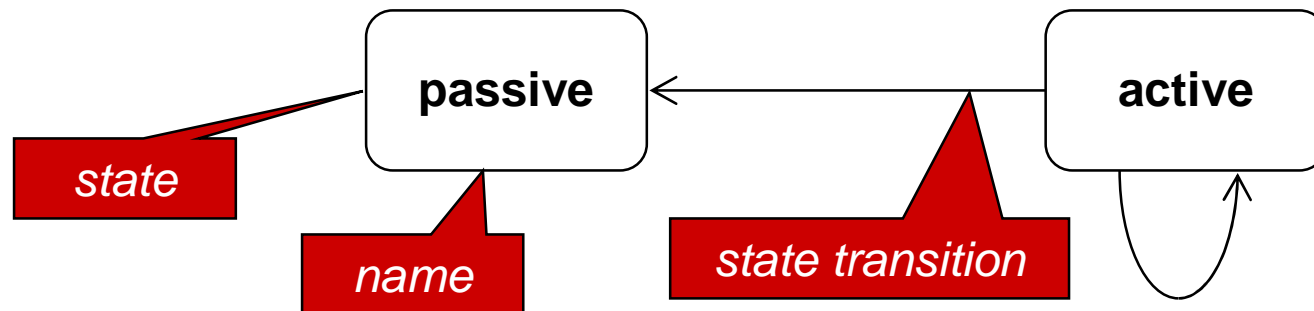
- o Most objects keep an internal state.
 - n It usually cannot change in arbitrary ways.
 - n There needs to be a precise model that makes explicit which state changes are legal.
 - n Well-known concept for this: state machines.
- o A state machine ...
 - n specifies states and state transitions of an object
 - n specifies possible actions in each states
 - n is attached to exactly one class
 - n is inherited by subclasses (but a concept of the refinement is not defined precisely)
- o A state machine thus defines a protocol (i.e., legal sequences of operations) for its class.

Elements of State Machines

- o State:
 - n in a specific state only certain actions are possible
 - n a state is a phase in the life cycle of an object
- o Transition:
 - n concrete actions are associated with state transitions
- o Event:
 - n transitions and their actions are triggered by events;
- o Guards:
 - n guards may prevent transitions from taking place
 - n guards are modeled by boolean conditions.

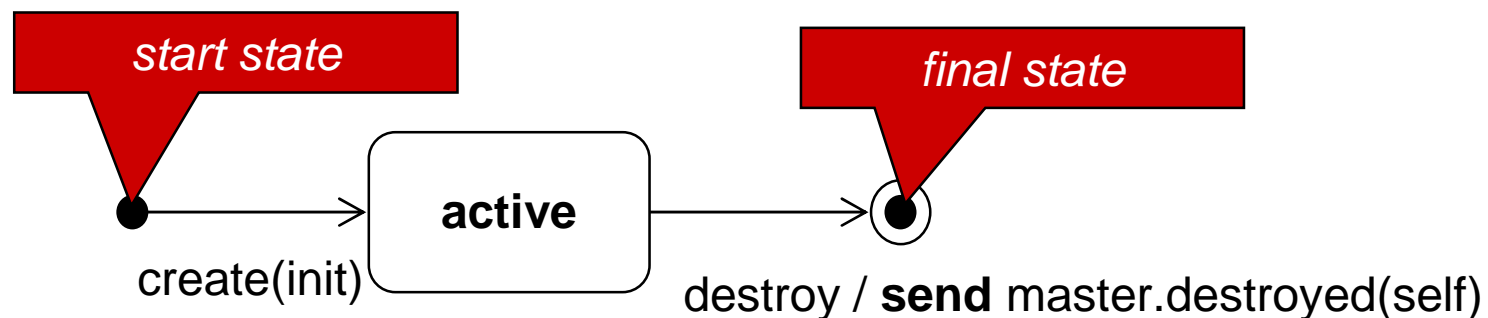
States and State Diagrams

- o A state diagram is a graph consisting of
 - n states
 - o simple states
 - o composite states (states refined by nested state diagrams)
 - n state transitions connecting the states.



Start and Final States

- Need well defined beginning and end of life-cycle.
- Start state: State transition is executed immediately during the creation of the object.
- Only possible event: create(parameter)



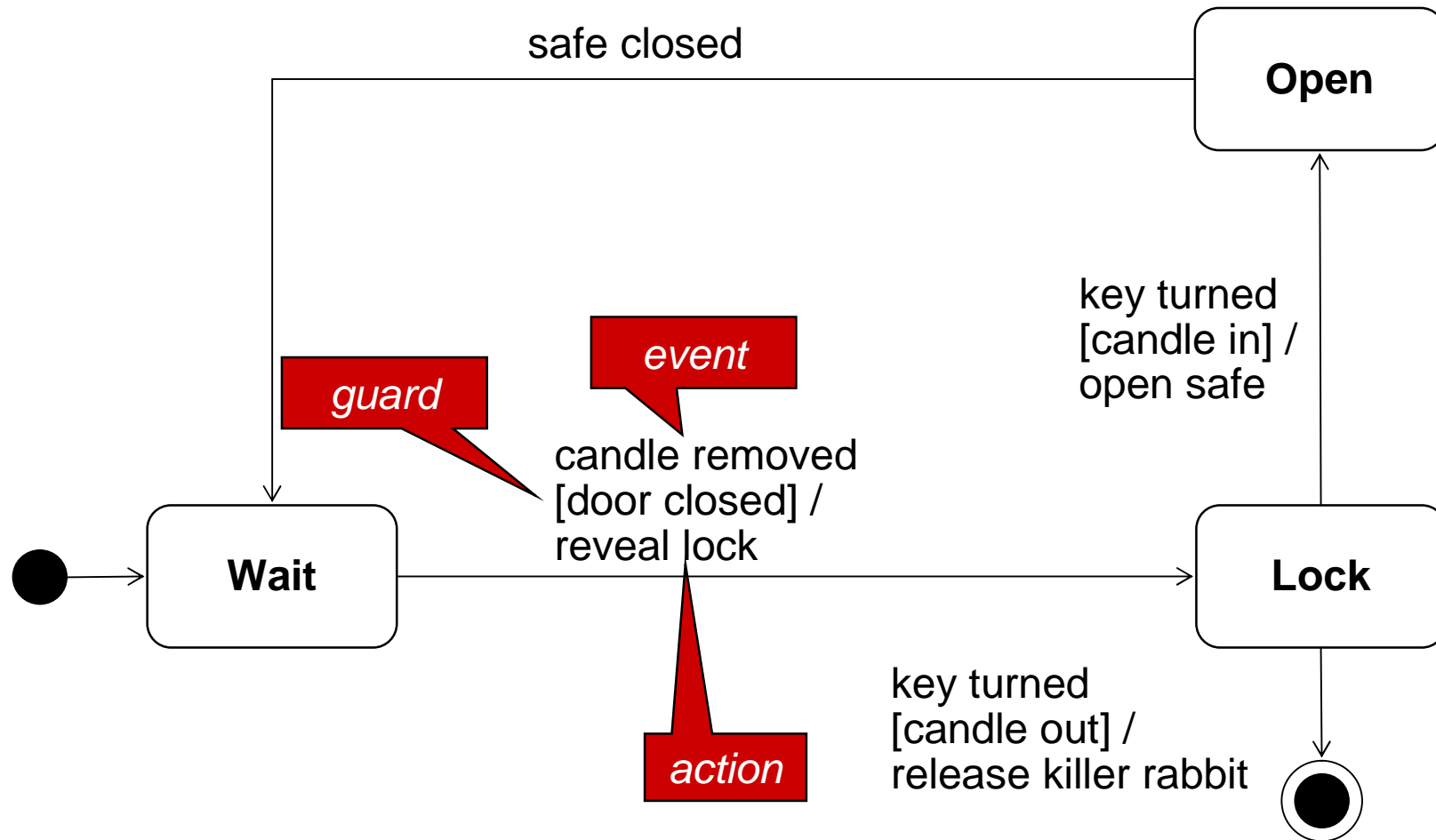
Transitions

- Transitions connect two states
- Transition can include a triggering event, a guard and actions to be executed.
- Transitions without event and guard are executed immediately (possibly after passing through all sub states).

... I decided to use a controller for a secret panel in a Gothic castle. In this castle, I want to keep my valuables in a safe that's hard to find. So to reveal the lock to the safe, I have to remove a strategic candle from its holder, but this will reveal the lock only while the door is closed.

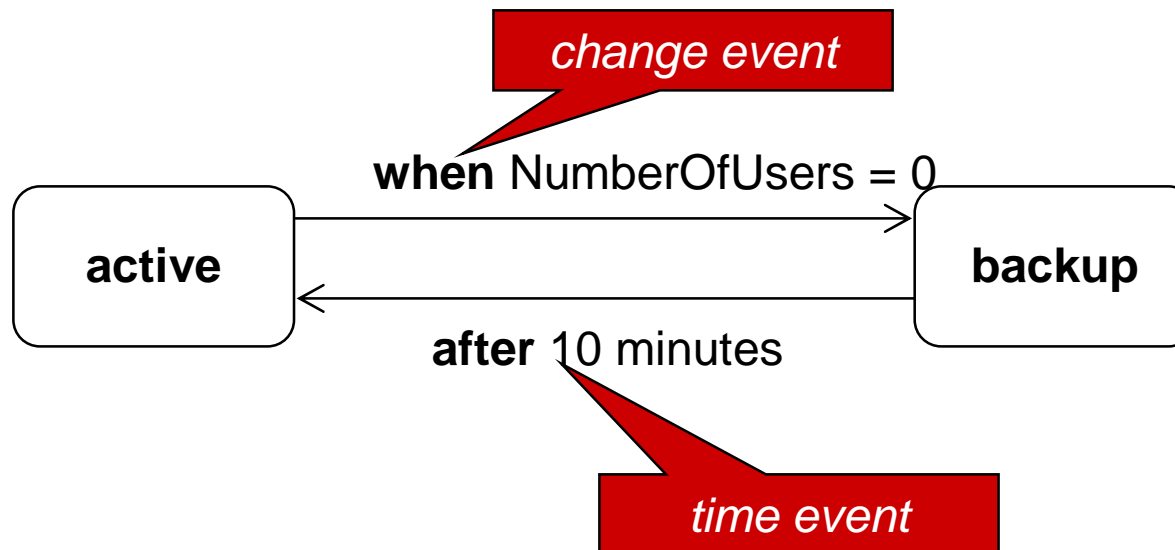
Once I can see the lock, I can insert my key to open the safe. For extra safety, I make sure that I can open the safe only if I replace the candle first. If a thief neglects this precaution, I'll unleash a nasty monster to devour him

Example



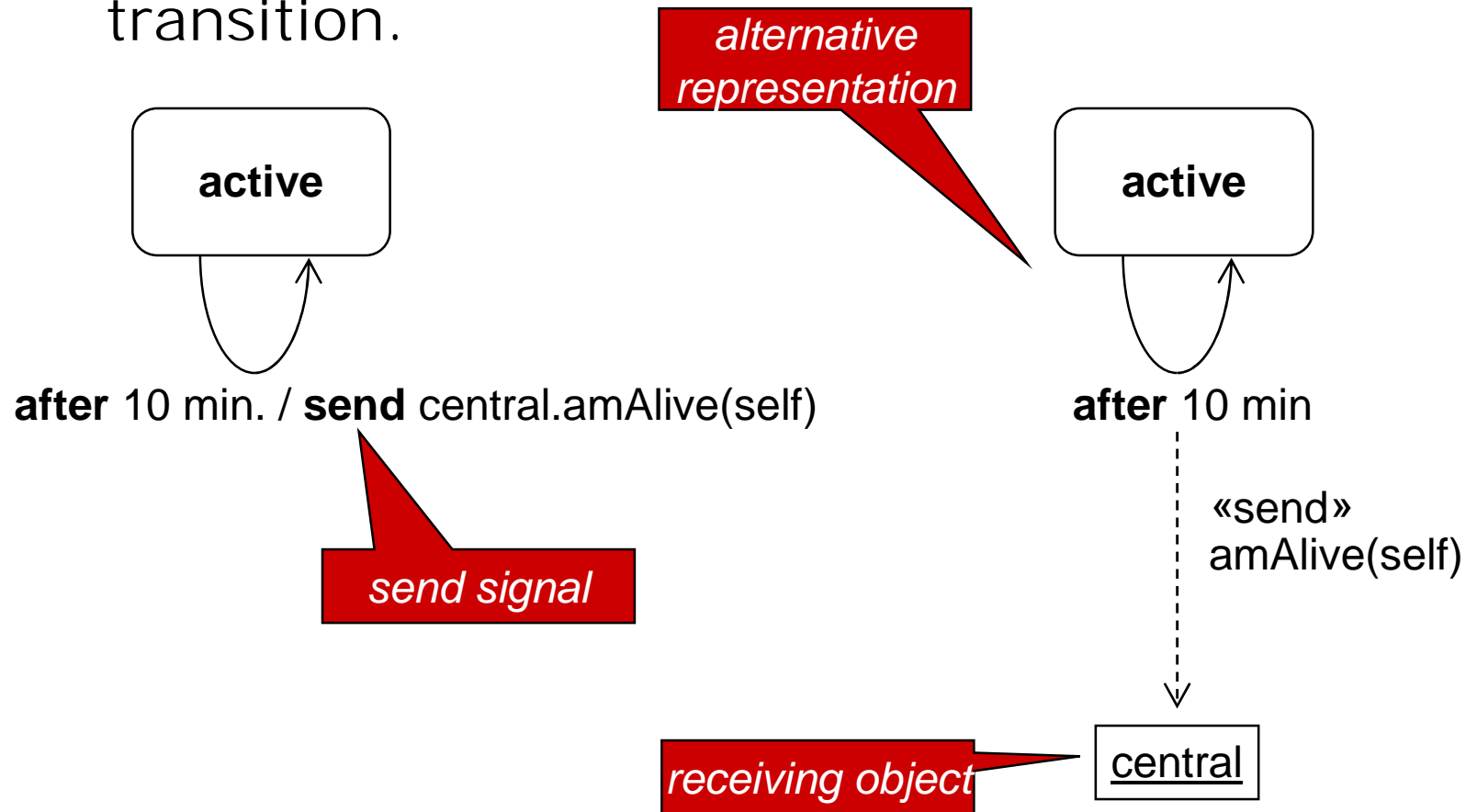
Time and Change Events

- A change event occurs if a specific constraint is fulfilled. The transition is made as soon as the constraint expression evaluates to true.
- A time event appears after the expiration of a time period.

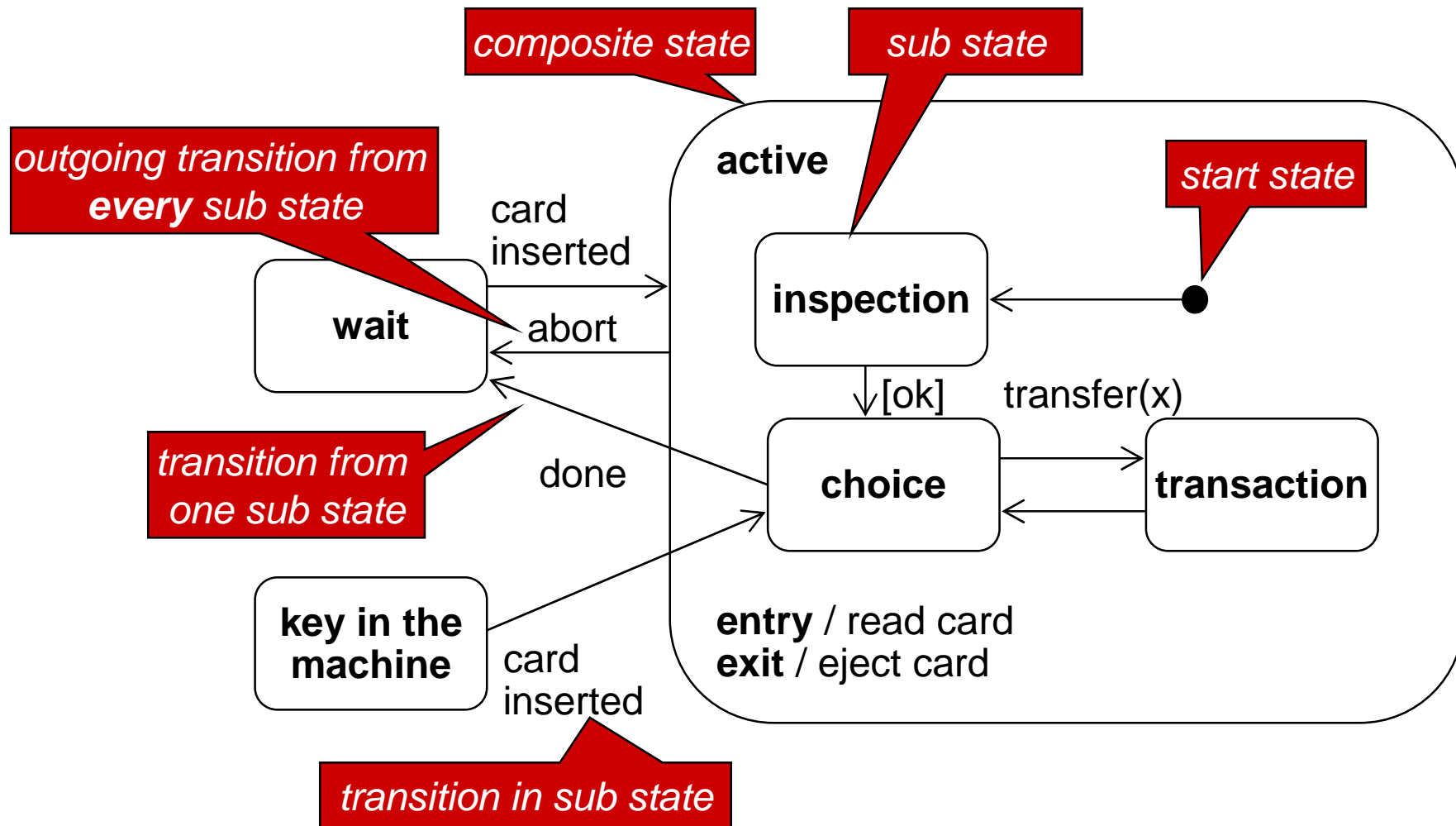


Sending Signals

- Signals can be sent to other objects during a transition.

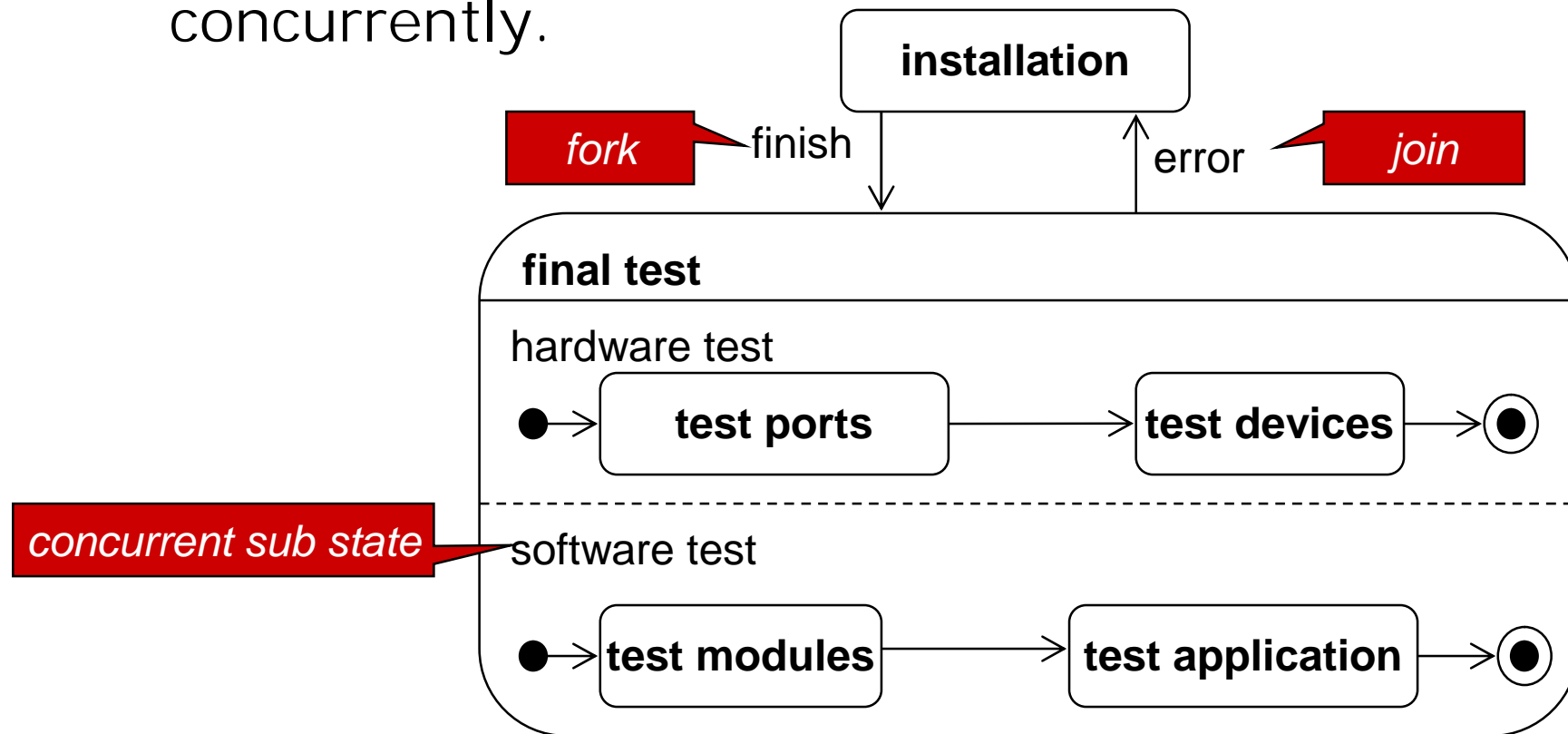


Composite States

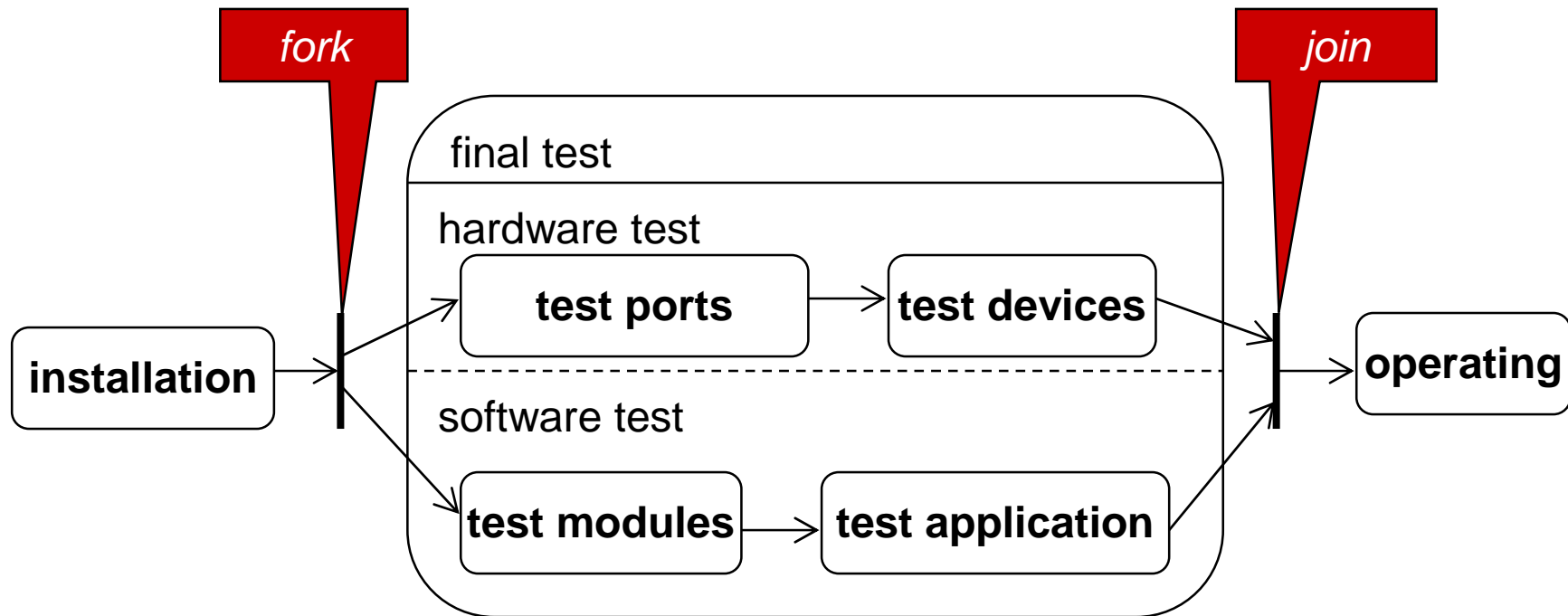


Concurrent Sub States

- o In a state several sequences of sub states described by state machines can be performed concurrently.



Concurrent Sub States: Alternative

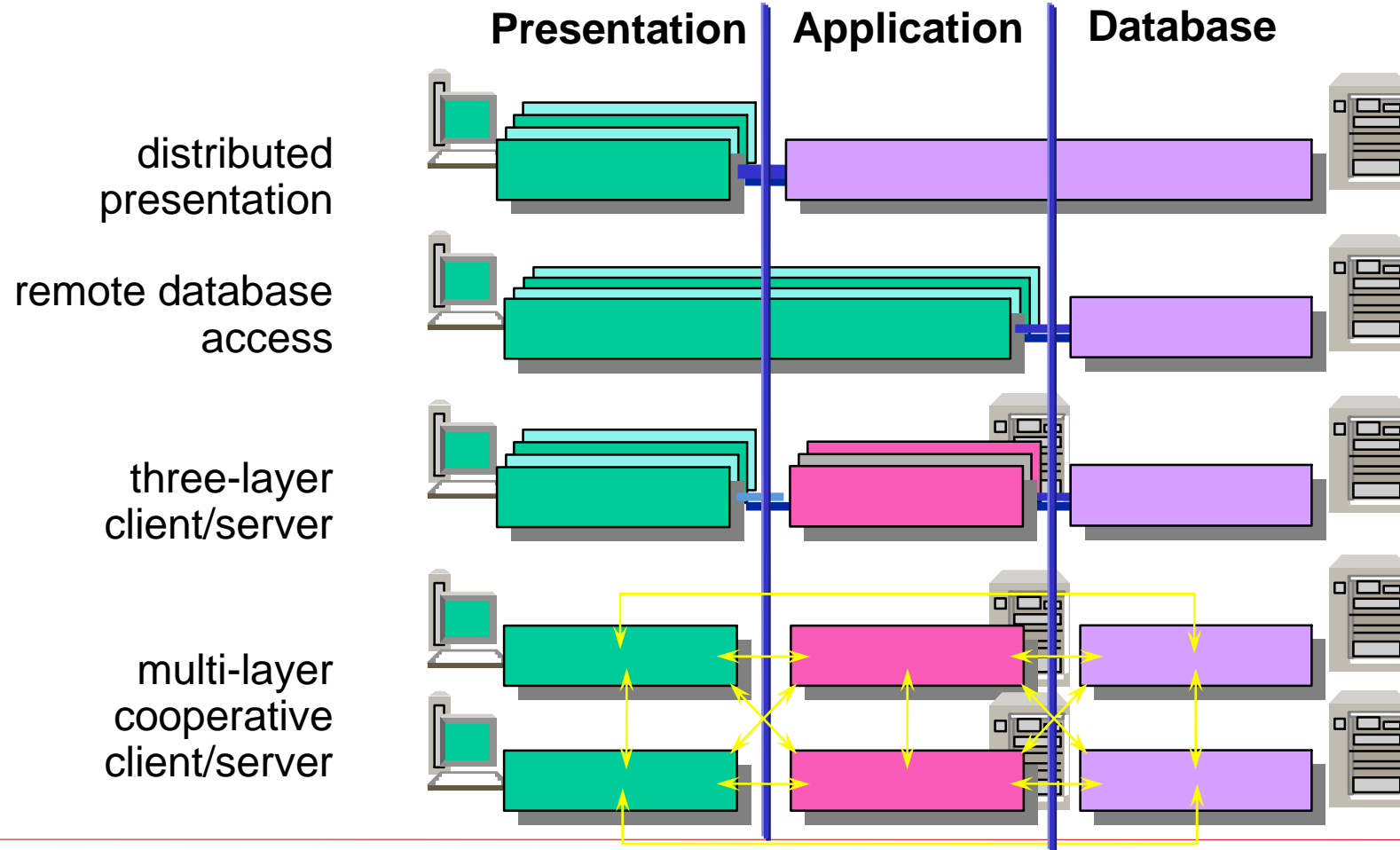


Architecture Design Models

- o An architecture model (structure model) is a model of a data processing system describing the static structure of the components of a system.
- o For large systems you should model this explicitly.
- o Examples:
 - n network topology (hardware)
 - n function tree (software)
 - n deployment diagram, package diagram (software)
 - n module diagram (hard/software)
 - n organization chart in a company model

Example: SAP R/3

- o Flexible three-tier client/server-architecture

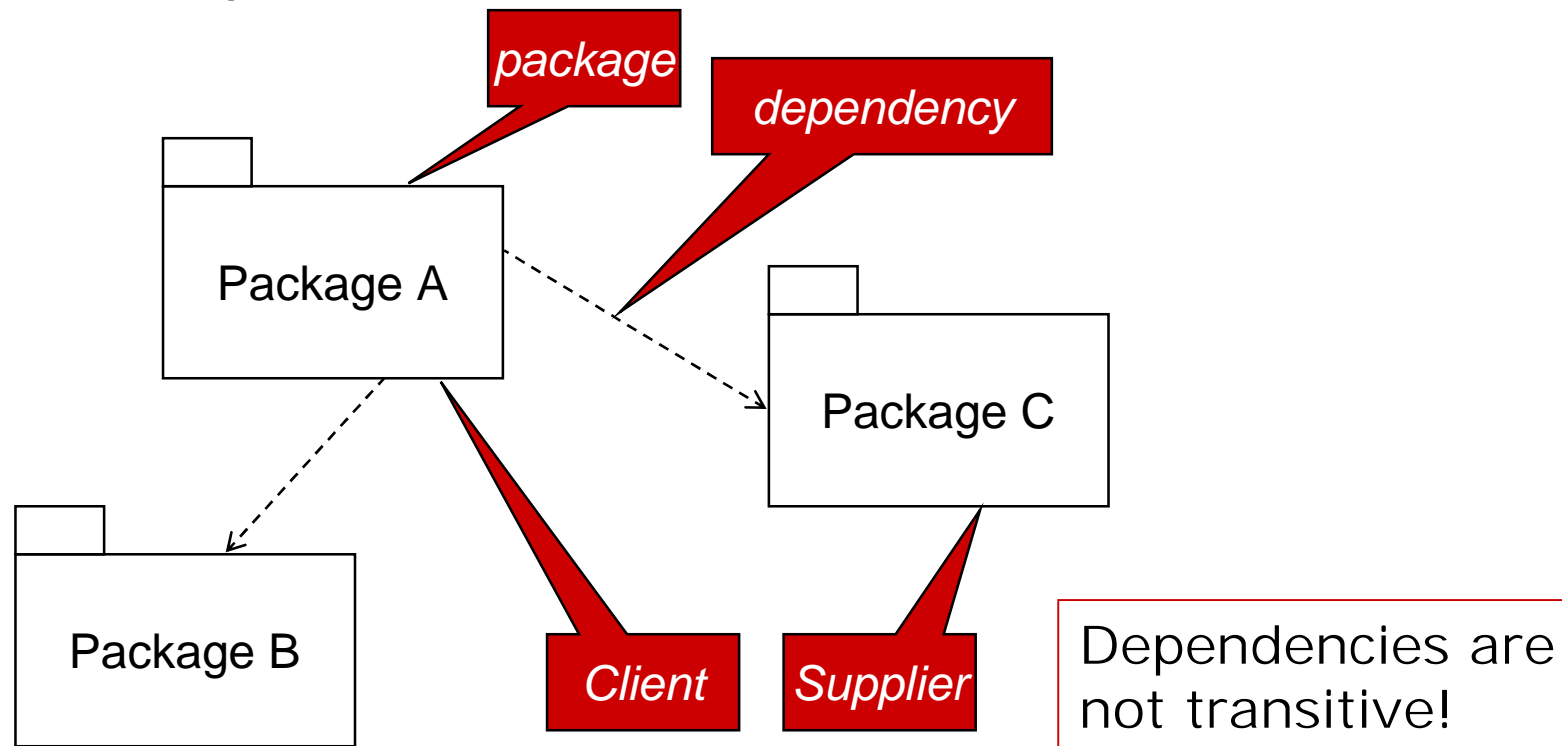


Specification Models in UML

- o Architecture diagrams have these elements:
 - n Package Diagrams:
 - o Classes, Interfaces & Packages
 - o Nested Packages
 - o Dependencies
 - n Deployment Diagrams:
 - o Nodes & Components
- o Specification models for architecture design :
 - n logical structures: packages with package diagrams; subsystems; interfaces
 - n physical structures: components in component diagrams; nodes; deployment diagrams

Package Diagram

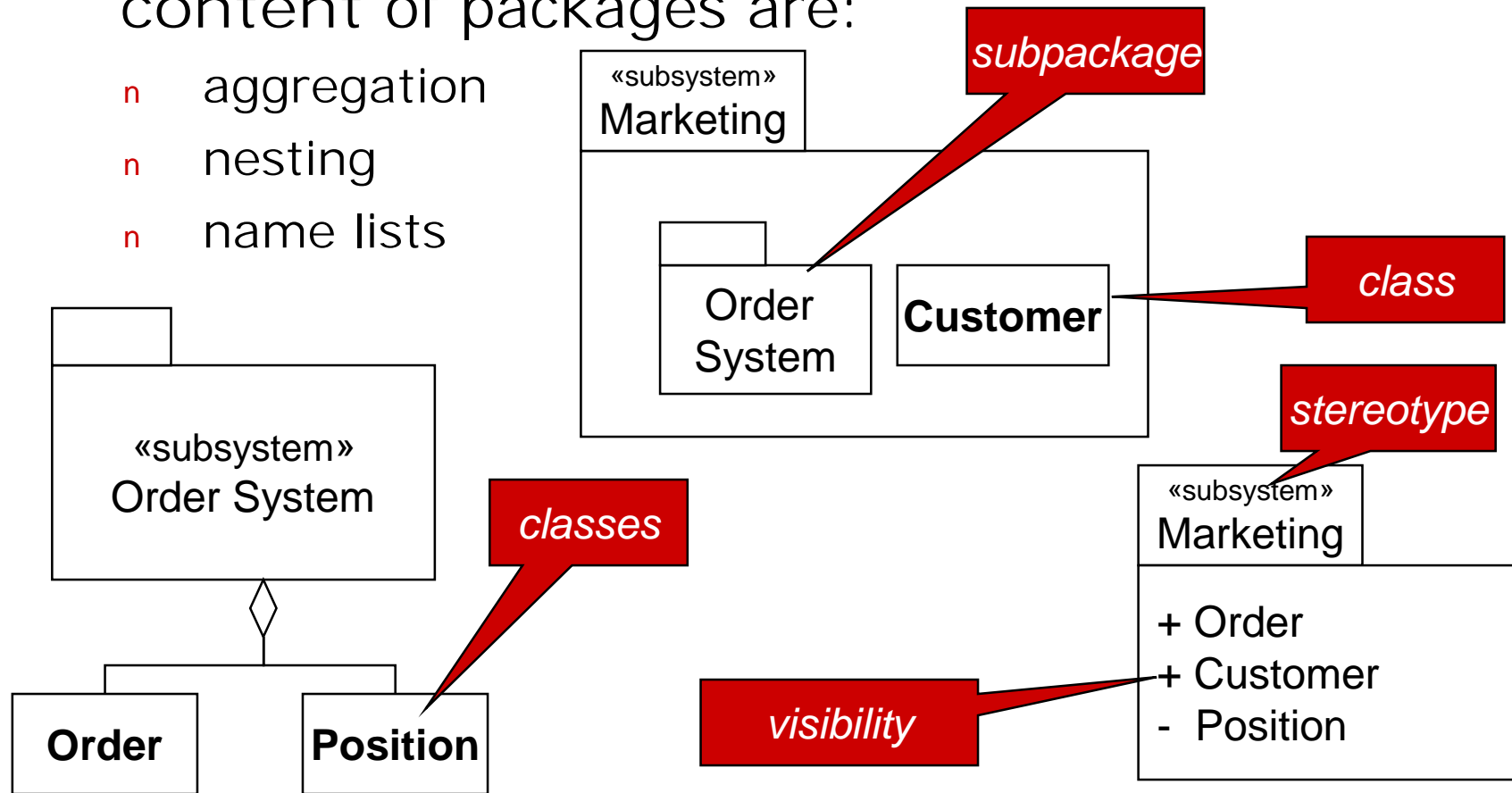
- o A package diagram shows the coherence (dependencies) between different packages of the system.



Package Contents

- o Alternatives for the representation of the content of packages are:

- n aggregation
- n nesting
- n name lists



Subsystems

- o Problem: How do I break down a large system into smaller systems?

Functional Decomposition

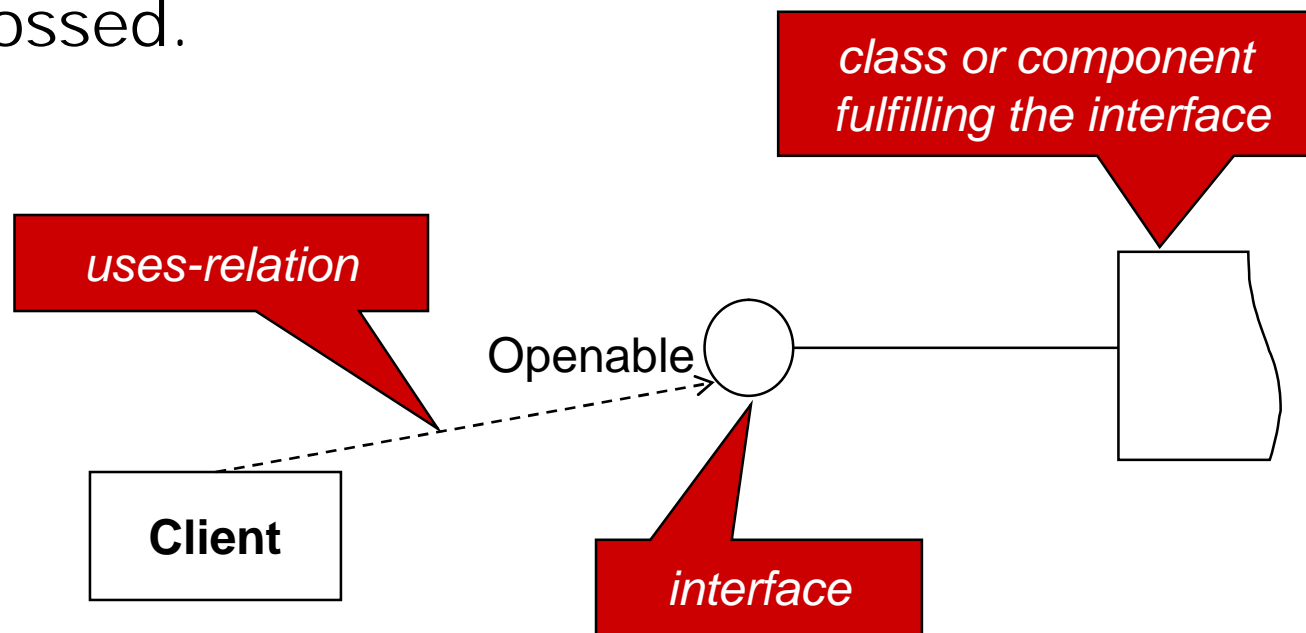
Map the total system on functions and subfunctions, starting with the use case.

OO Packages

- n Collect classes into subsystems.
- n Build layers of subsystems.
- n **Abstraction:** Concentrate on essentials.
- n **Locality:** Group together related components (data and algorithm).
- n **Hiding:** Restrict the visibility of details, so that only those parts of a system that need to know the details have access to them.

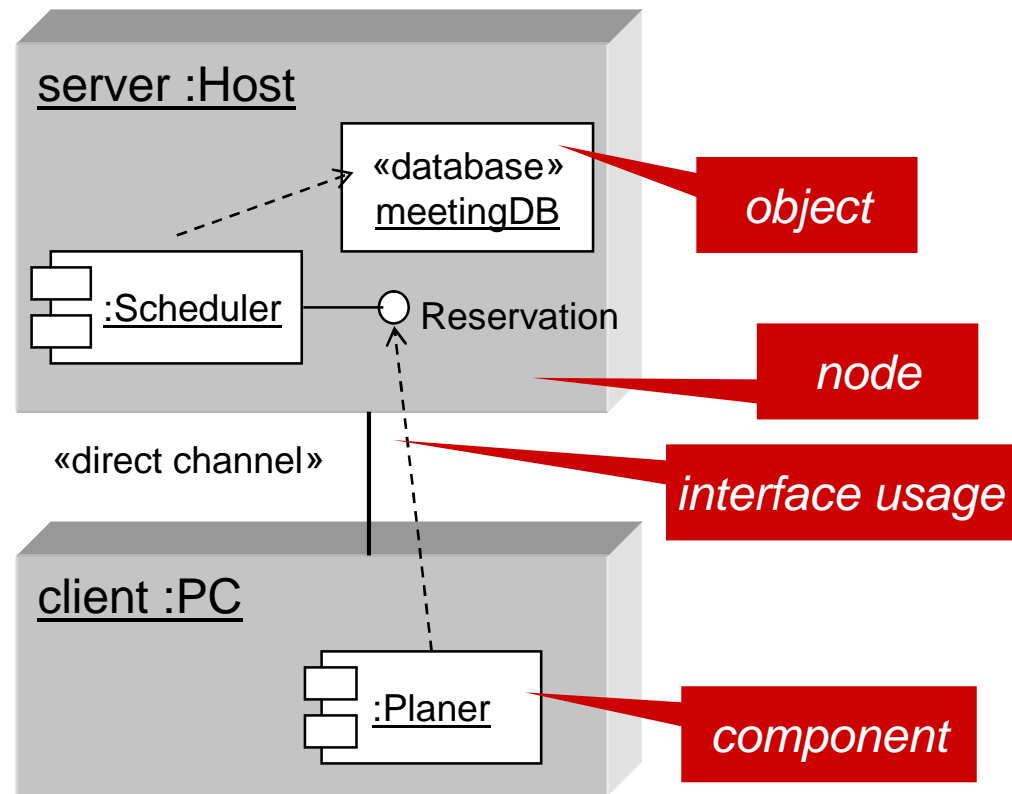
Interfaces in Architecture

- Interfaces let you specify the outward appearance.
- They are important in architecture diagrams, because they show how system boundaries are crossed.



Deployment Diagram

- A deployment diagram shows the configuration of a node at runtime as well as the components (instances) and objects residing on it.
- Components not existing as runtime object (instance) should appear in component diagrams only .



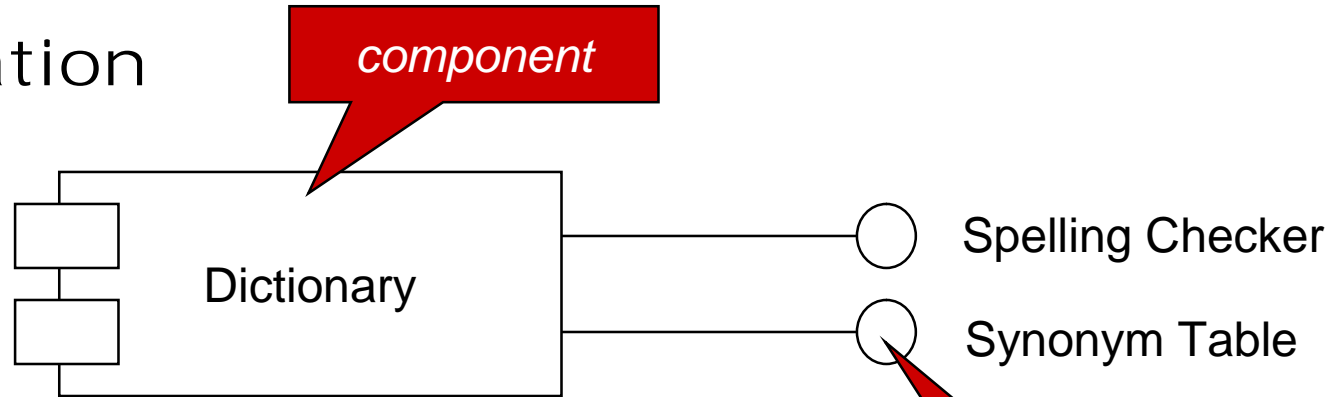
Components (1)

- o A component is a physical, replaceable part of a system containing an implementation which realizes a set of interfaces
- o Components have two aspects:
 - n Code: A component consists of code, components can contain or use components.
 - n Identity: A component can have identity and state represented by objects. An object which wants to use services of the component must specify the instance. (e.g., Bean reference, DLL handle, process, CORBA-IOR,...)

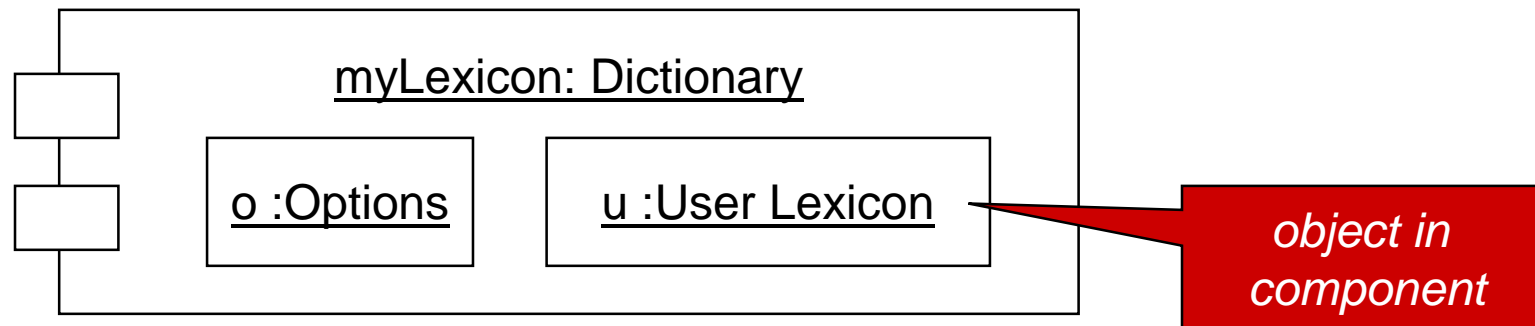
Example: Spelling checker as component:
Identity and state by user dictionary,
different versions and languages.

Components (2)

- o Notation

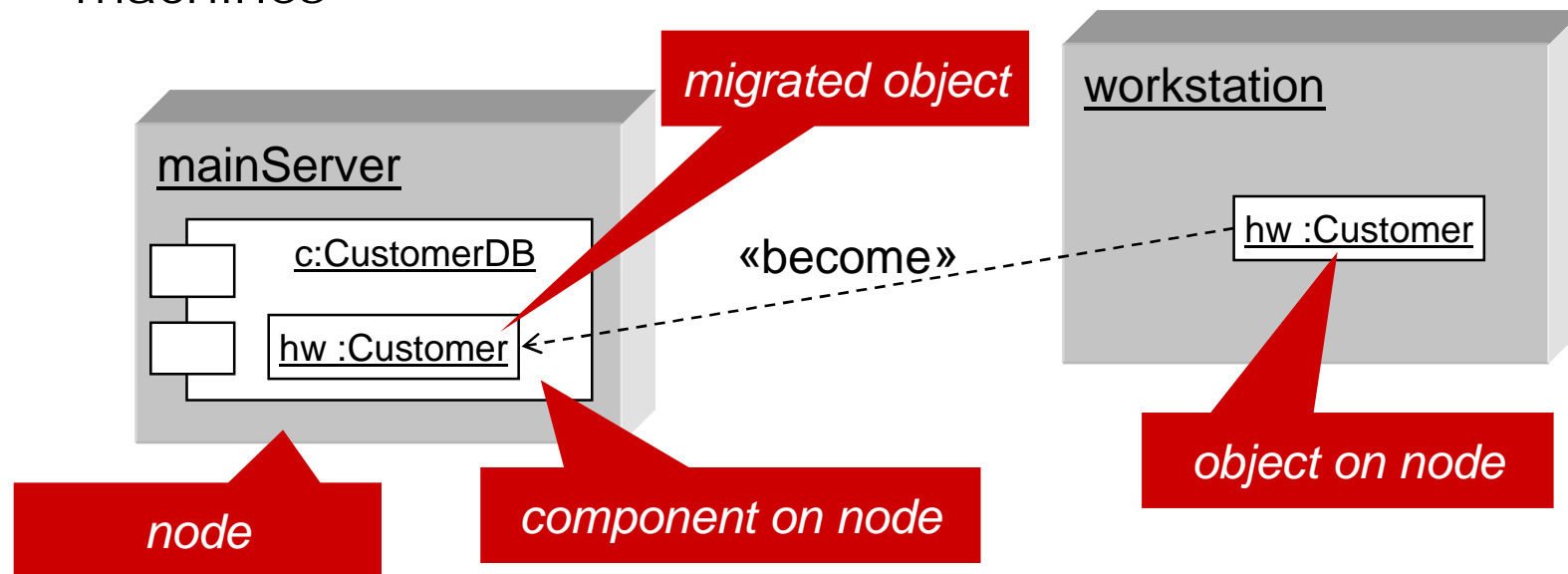


- o component with identity



Node

- o A node is a physical entity existing at runtime
 - n represents processing resources
 - n has storage and computation capacity
 - n On a node objects and components can be live
- o Nodes can be computers, humans, or other devices or machines



Discussion

- o Components ...
 - n are conceptually and functionally larger than a class.
 - n unite behavior and collaboration of a group of classes (see collaboration diagrams)
 - n are independent of other components but usually collaborate with them
 - n are replaceable with other realizations of the same interface
 - n are fundamental building blocks of component-based architectures
 - n can be built recursively. A system can be a component on the next higher inspection level.

References

- o Petri Nets
 - n Introduction:
http://www.daimi.au.dk/PetriNets/introductions/pn2000_introtut.pdf
 - n Examples:
<http://www.daimi.au.dk/PetriNets/introductions/aalst>