
Development of Component- and Service-Oriented Software Architectures with UML and UML-RT — Sequence Diagrams and statecharts —

Ingolf H. Krueger
ikrueger@ucsd.edu

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0404, USA



California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA

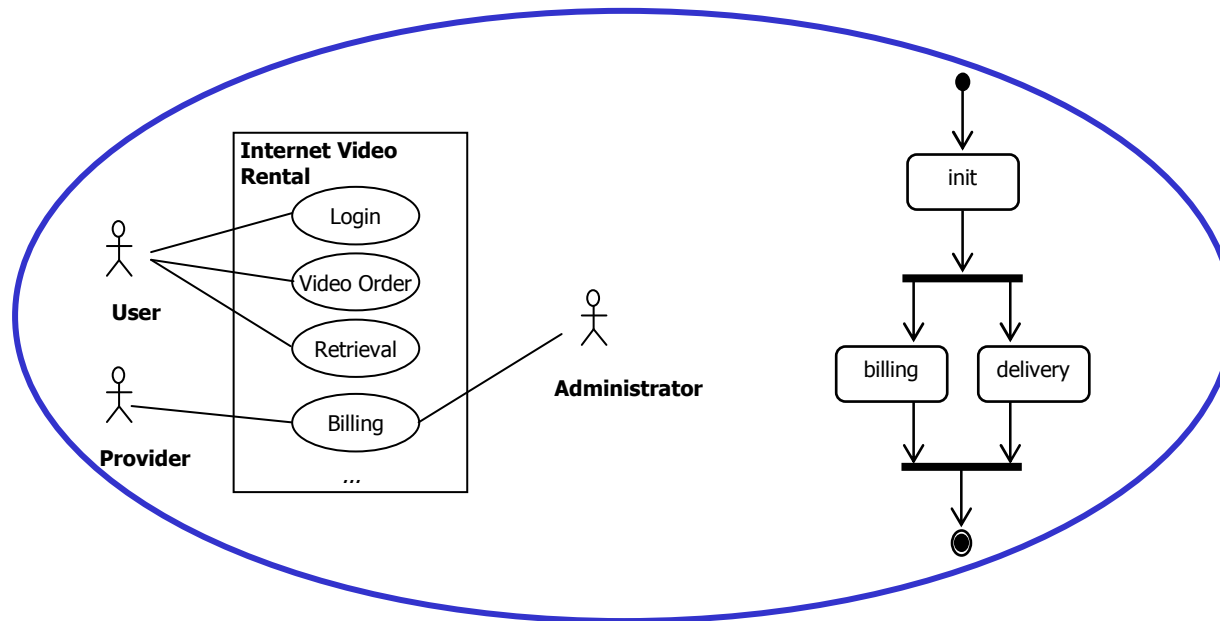


Overview

- Components and their Roles
- Interaction Diagrams
- Statecharts

Components and their Roles

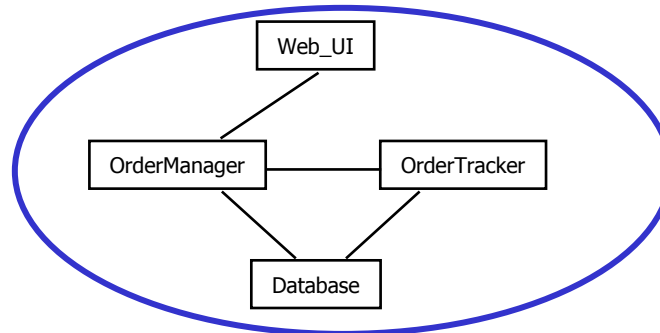
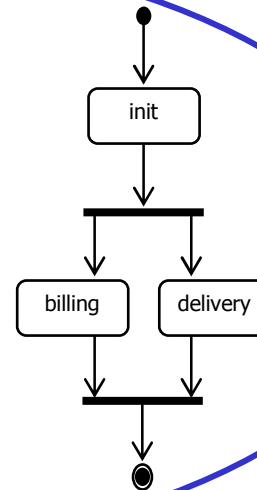
What happened so far...



Elicitation of

- Use Cases
- Activities

No/little assignment of responsibilities

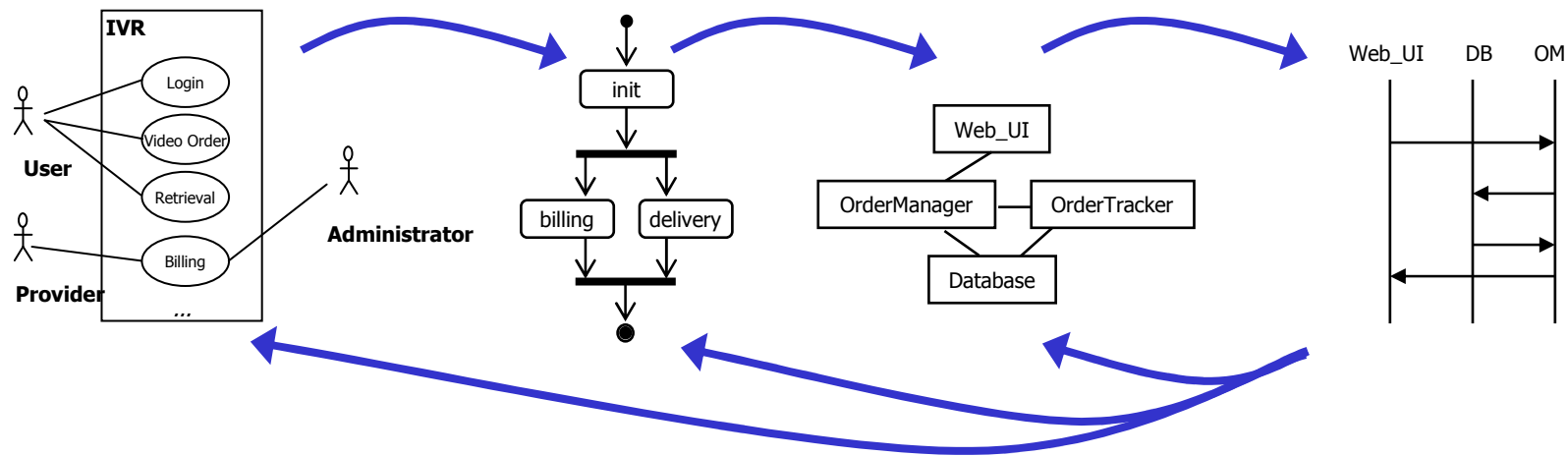


Domain model

Components and their Roles

Key Points:

- Mapping between components and use cases/activities
 - Assignment of responsibilities
- Component interaction required to establish use case/activity
 - Interaction Model
- Validation of use cases, activities, domain model
 - Iterative Process



Overview

- Components and their Roles
- Interaction Diagrams
- Statecharts

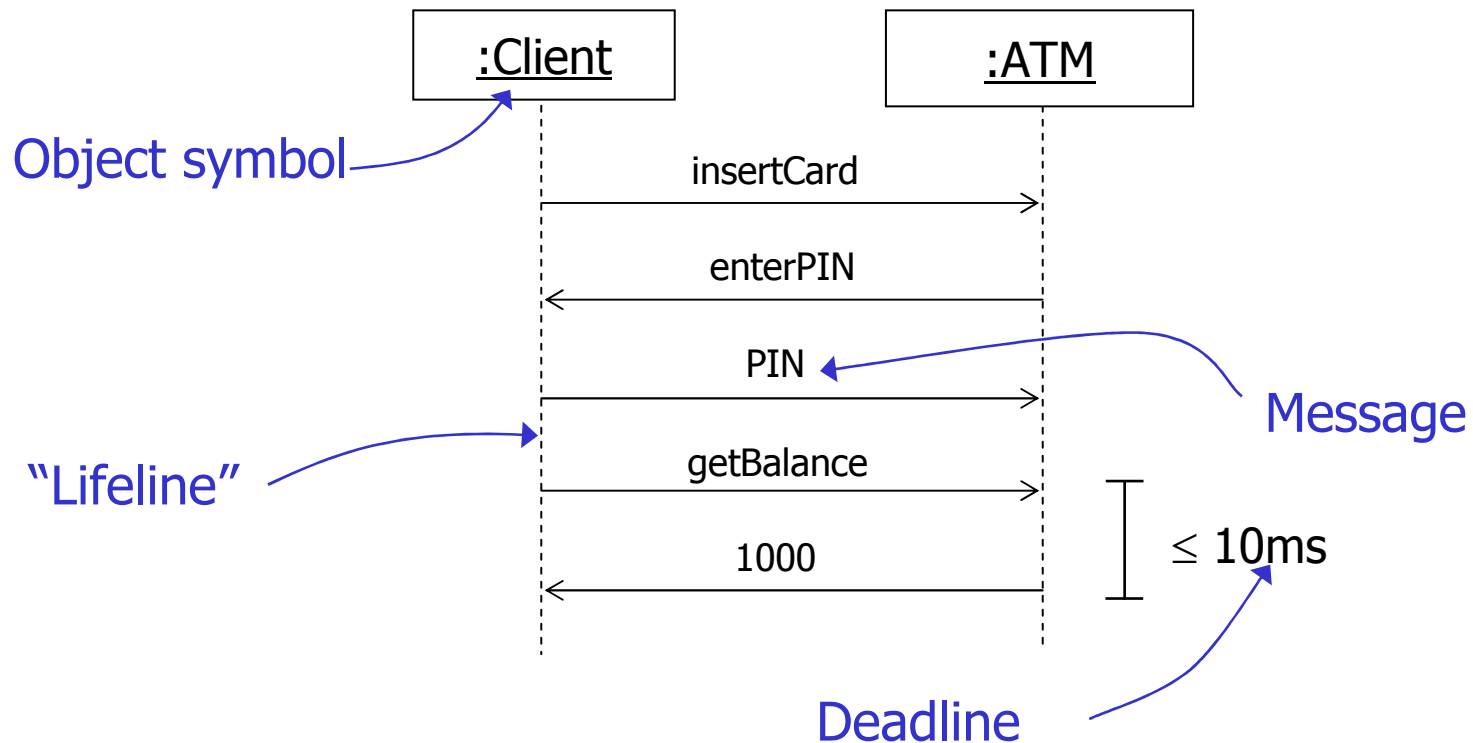
Interaction Diagrams

- Interaction is key ingredient of distributed system's SW architecture:
 - synchronous method-/operation calls
 - asynchronous communication via message exchange
- Coordination/Synchronization via interaction to establish use case
- Communication infrastructure:
 - Object/Component identifiers and interfaces
 - Communication paths ($\Rightarrow$ domain model)
 - Method call / message exchange mechanisms
- Communication defines:
 - Assumptions at environment
 - Commitment towards environment

Interaction Diagrams

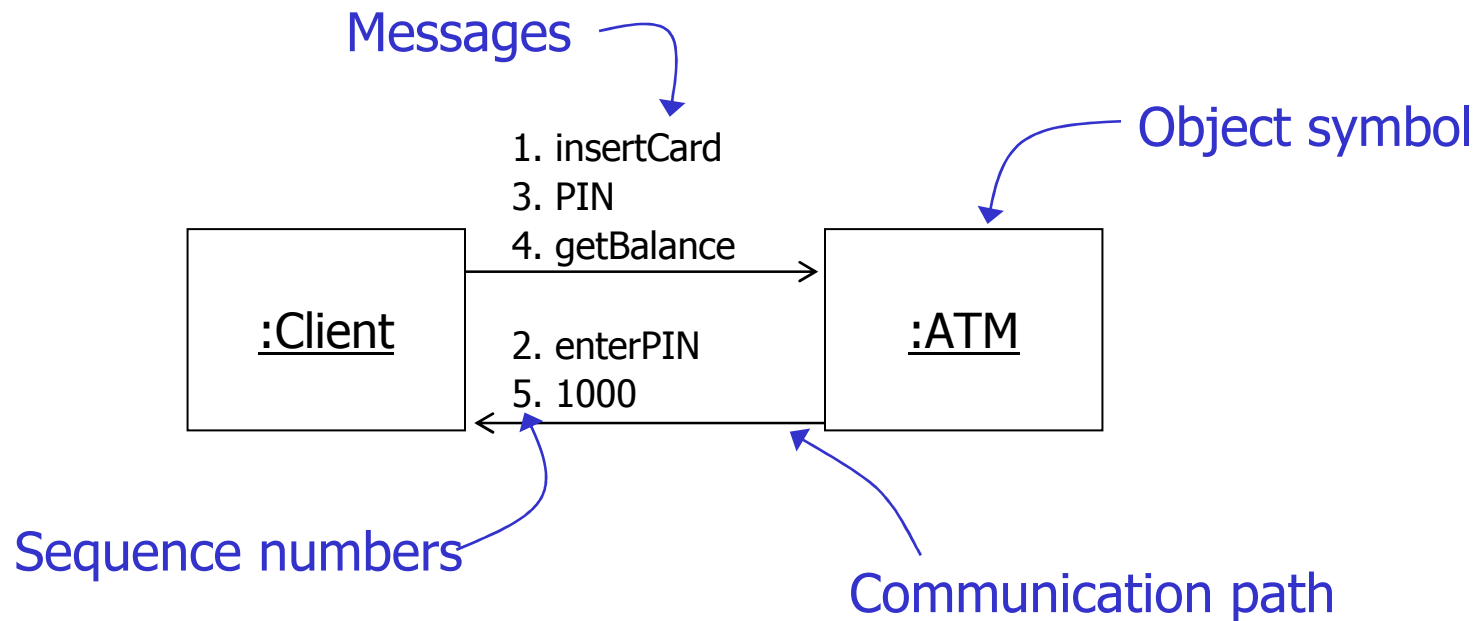
UML: Sequence and collaboration diagrams

Sequence Diagrams (SDs):



Interaction Diagrams

Collaboration diagrams:

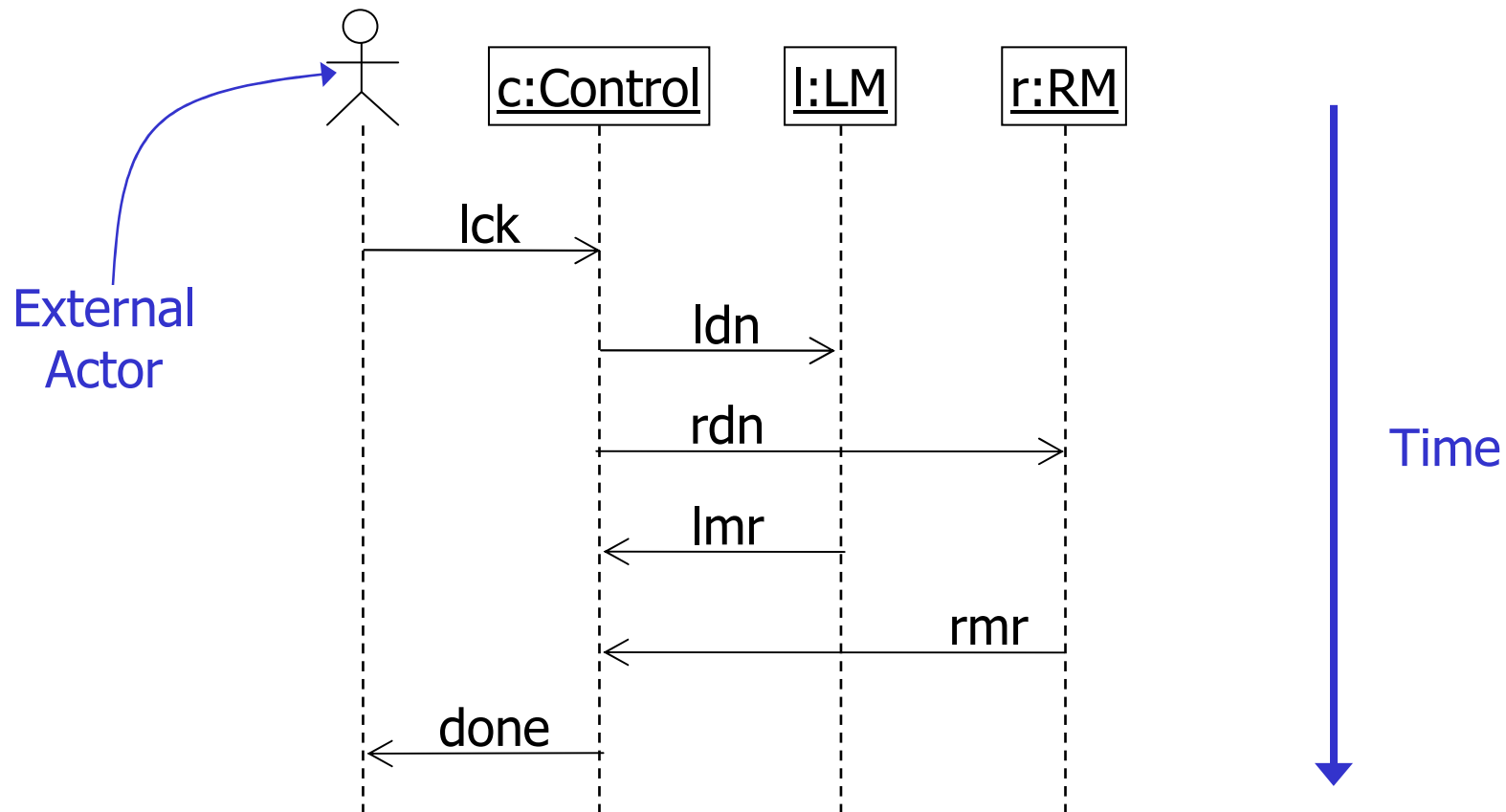


Sequence Diagrams

Roots of SDs:

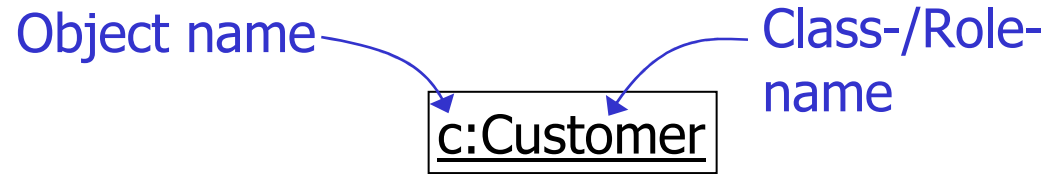
- MSC-96 (Message Sequence Charts, ITU-T Standard Z.120)
 - ⇒ Specification and documentation of asynchronous telecommunications protocols
- OMSCs (Object Message Sequence Charts, [POSA96])
 - ⇒ Specification and documentation of control-flow-oriented systems

Sequence Diagrams: Basic Notation

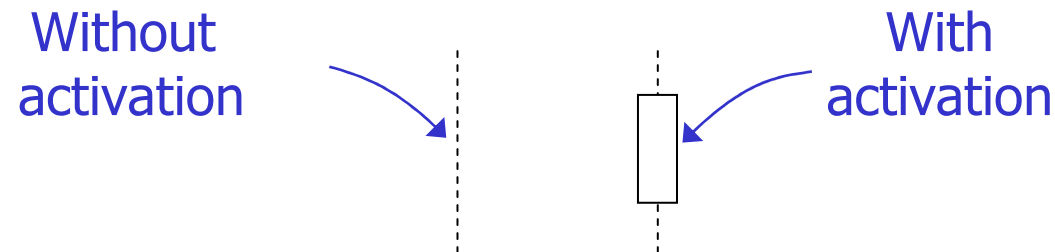


Sequence Diagrams: Basic Notation

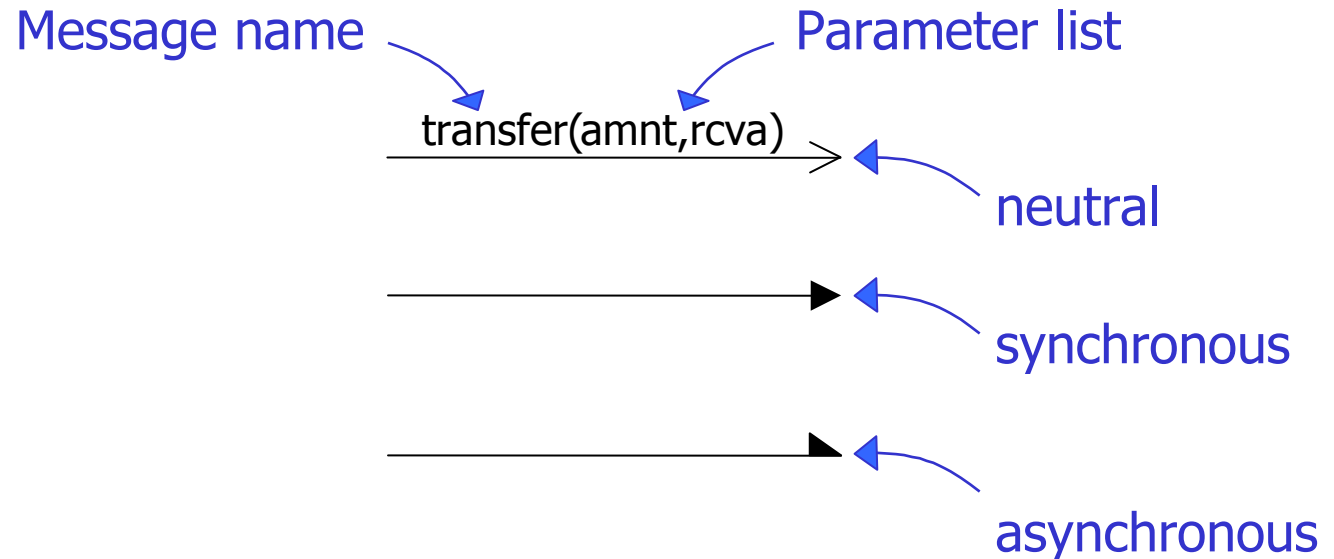
- object symbol:



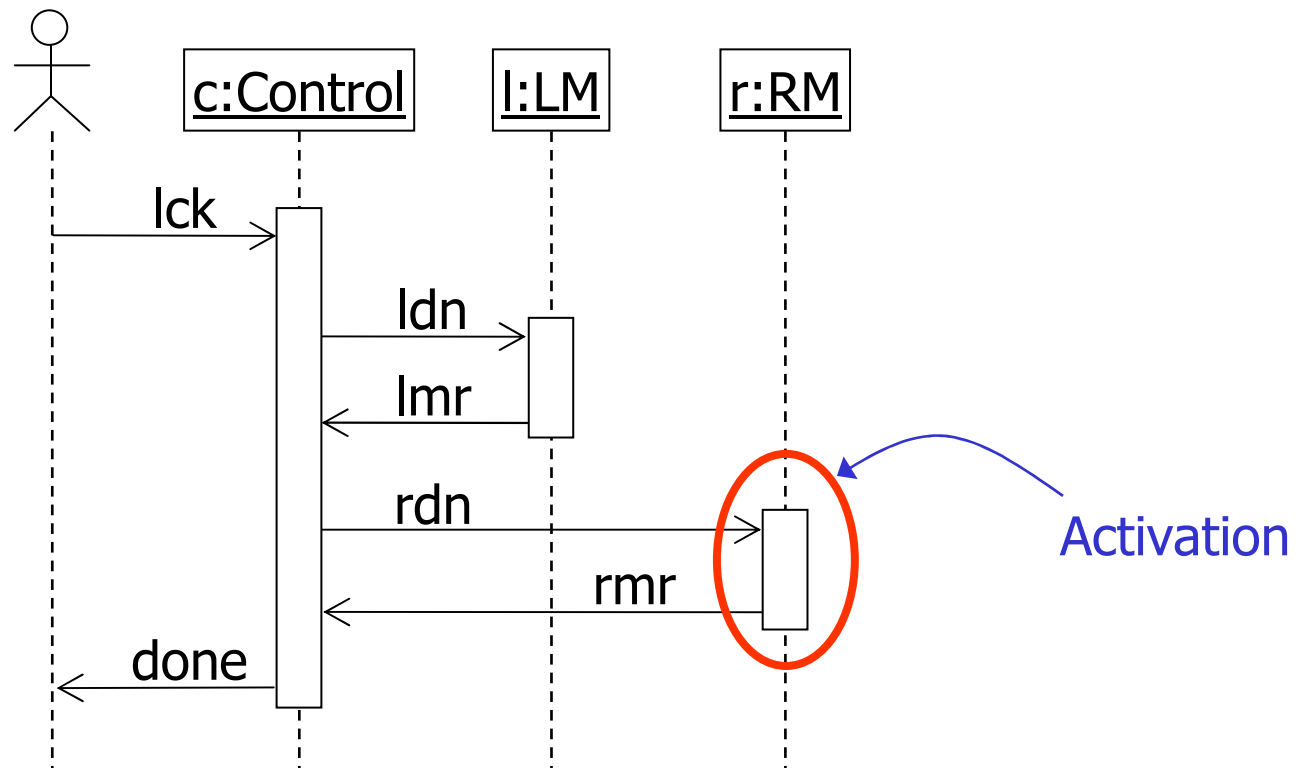
- lifeline:



- arrows:

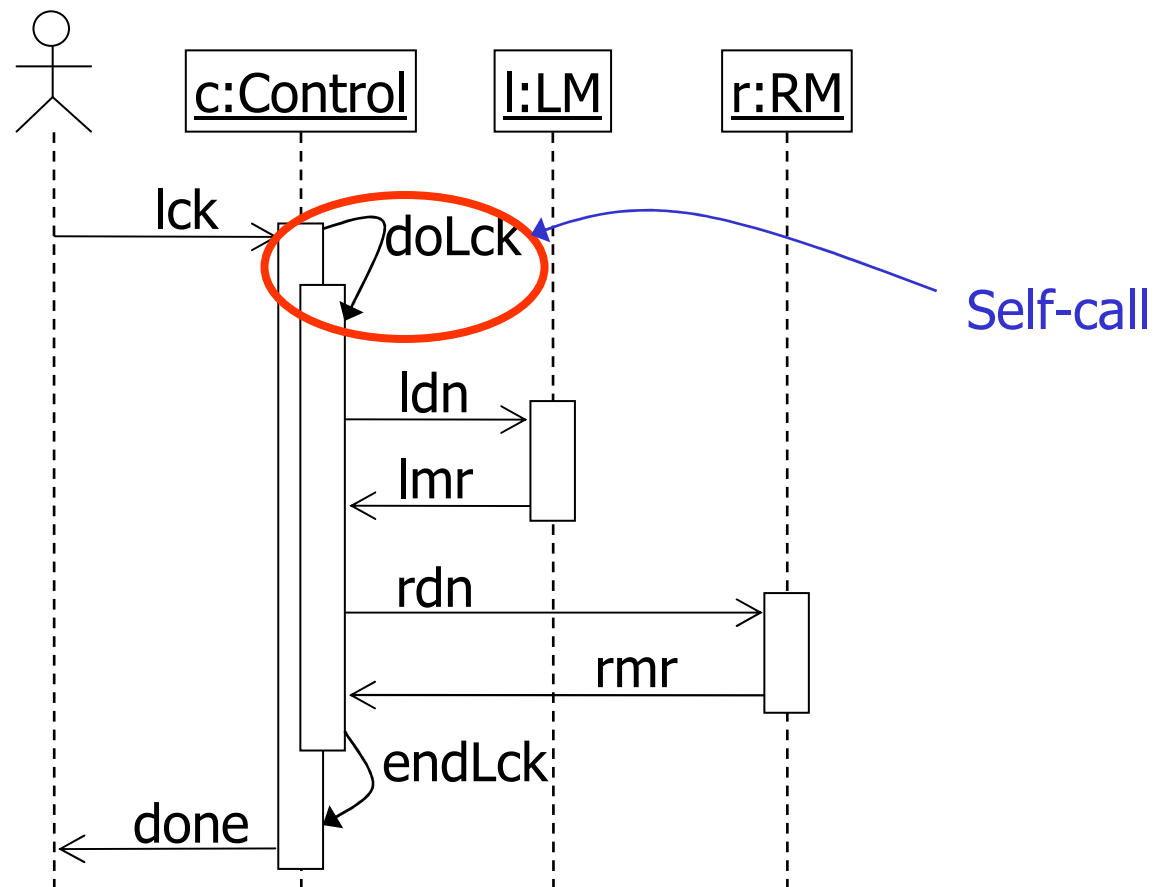


Sequence Diagrams: Activations



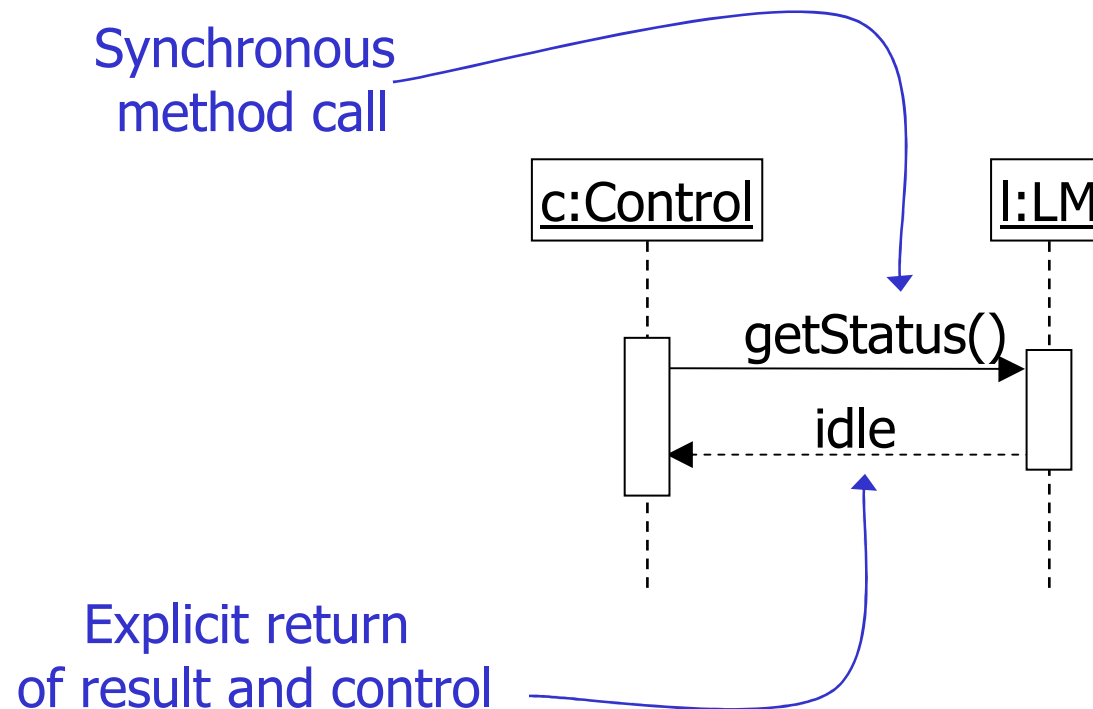
Sequence Diagrams: Activations

Internal Control Flow



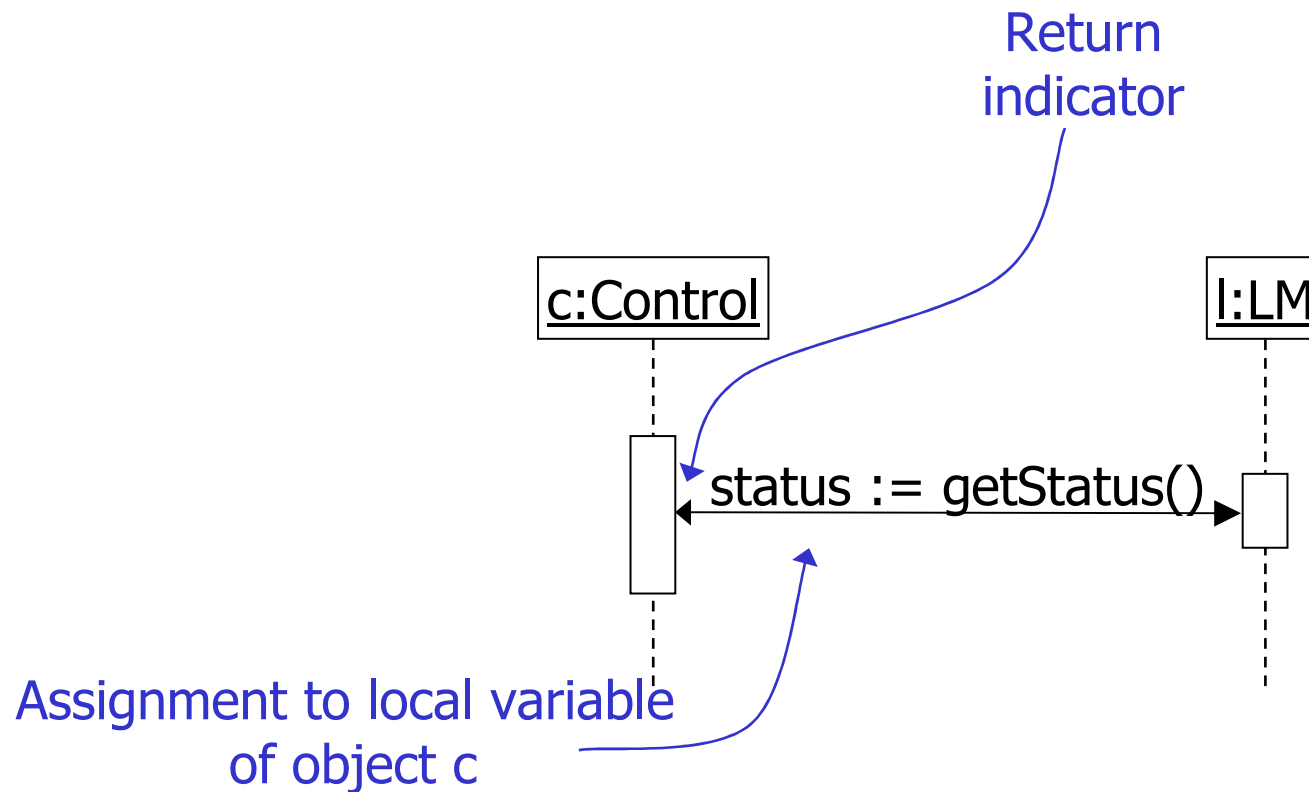
Sequence Diagrams: Returns

Explicit return arrows indicate return of result and control to caller



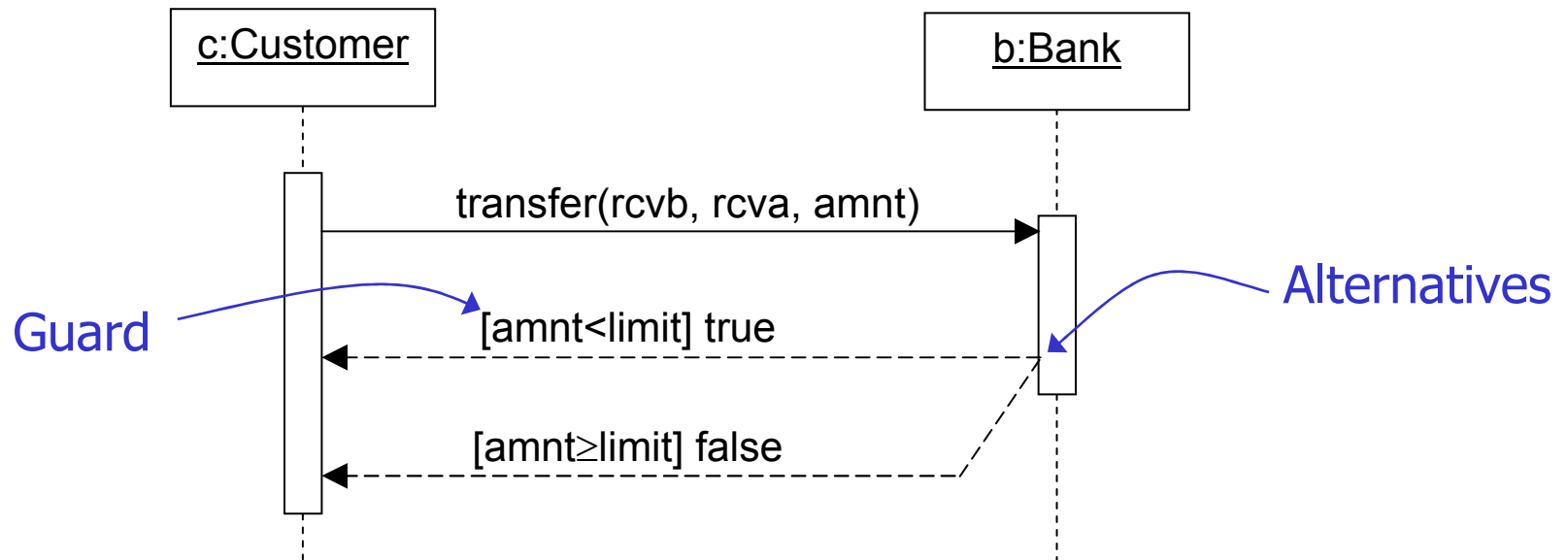
Sequence Diagrams: Returns

Implicit return of result and control



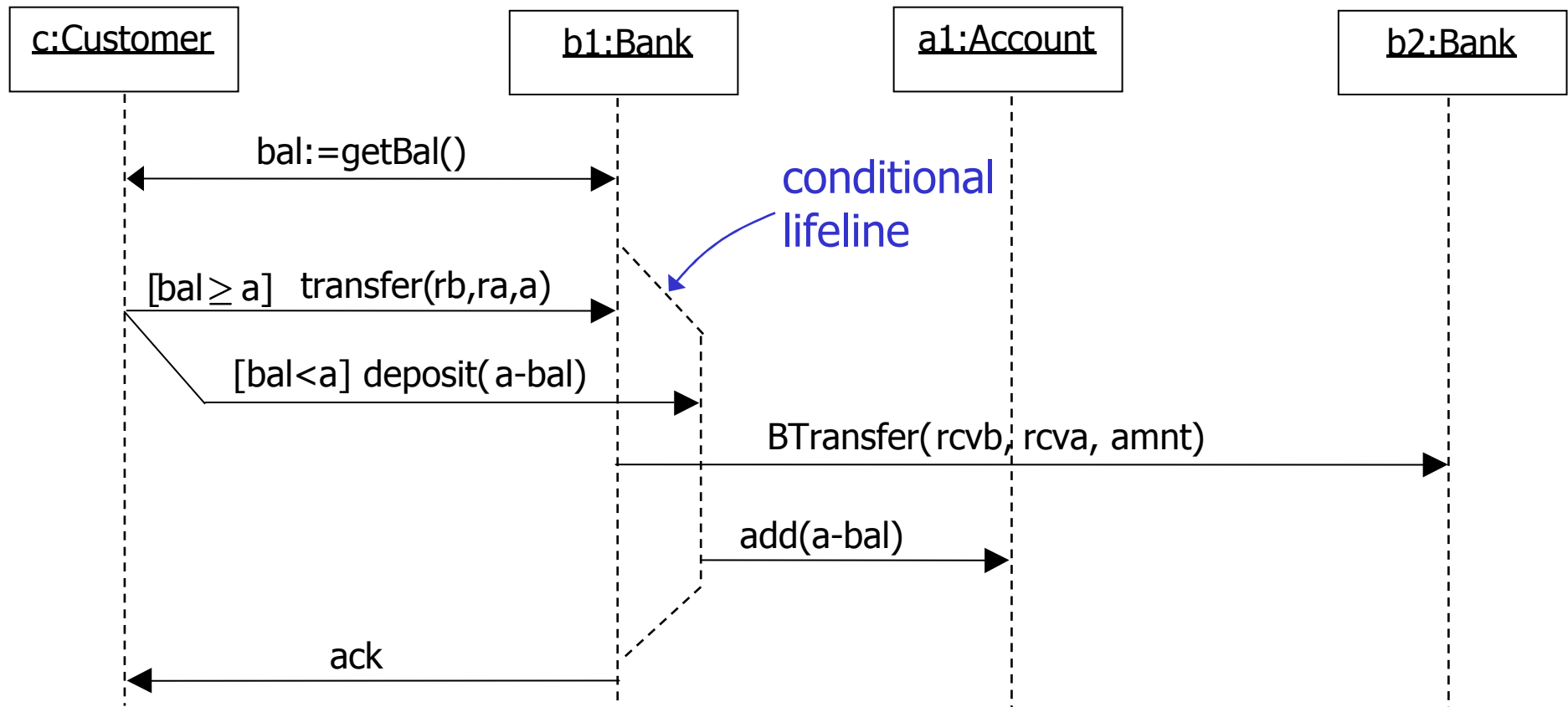
Sequence Diagrams: Alternatives

Boolean guard: occurrence condition for message



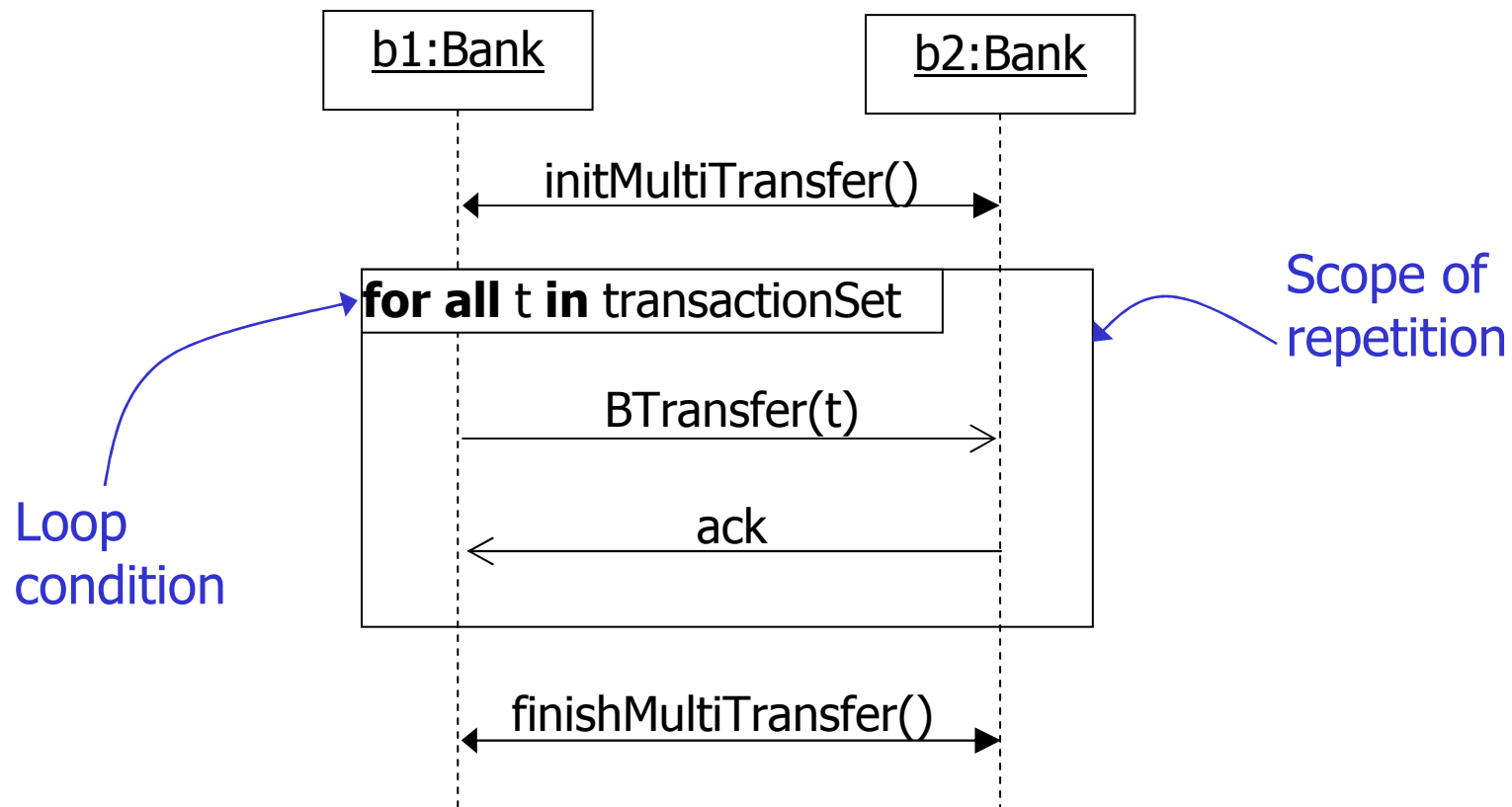
Sequence Diagrams: Alternatives

Conditional lifelines

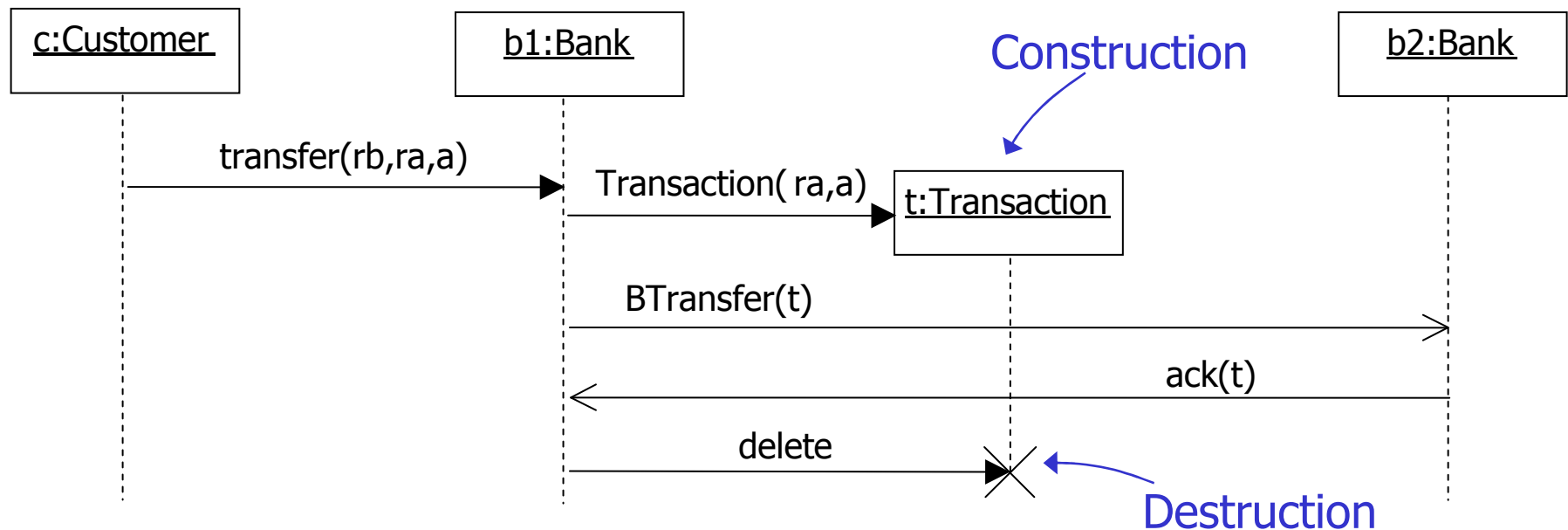


Sequence Diagrams: Repetition

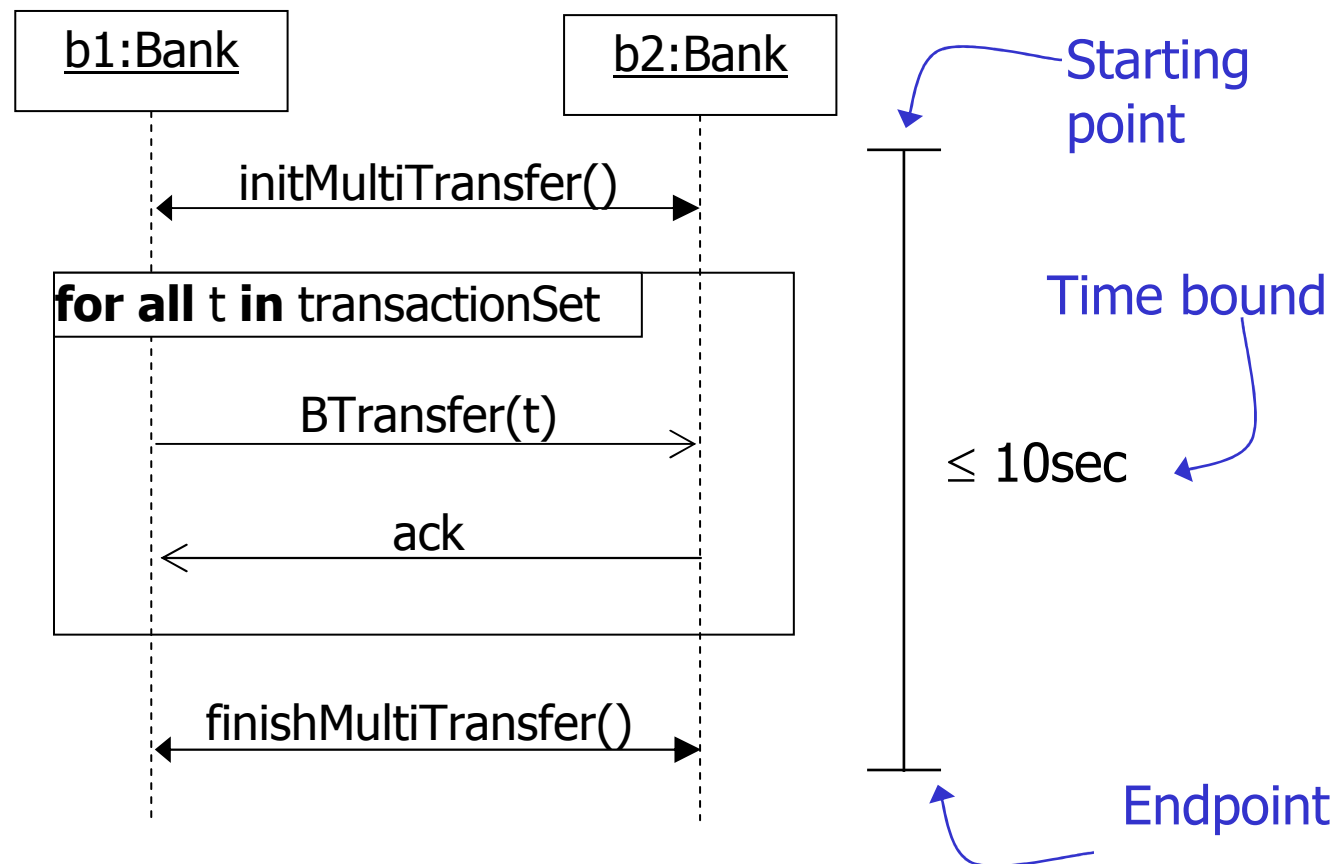
Enclosure with loop condition



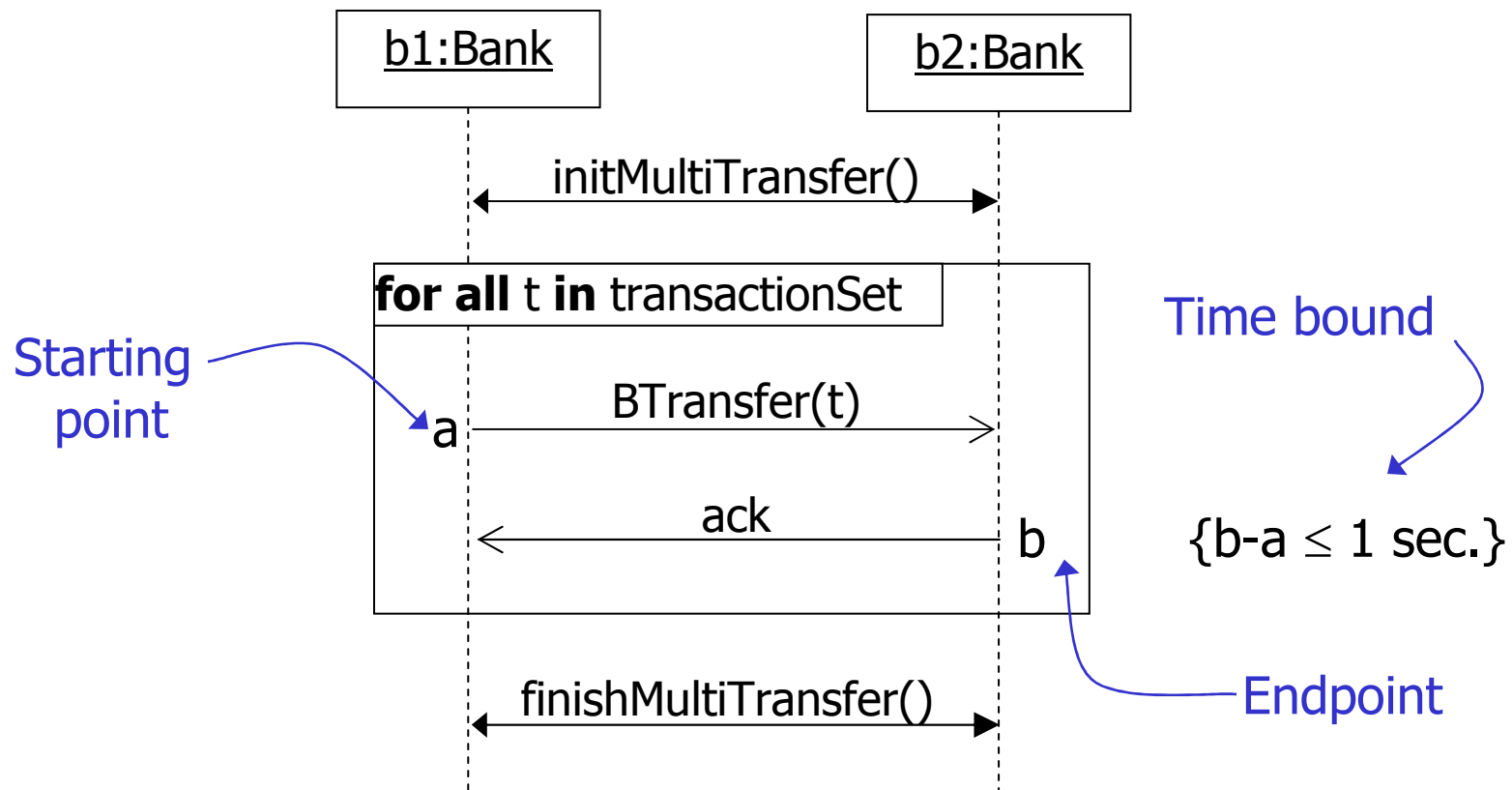
Sequence Diagrams: Construction and Destruction



Sequence Diagrams: Time Bounds

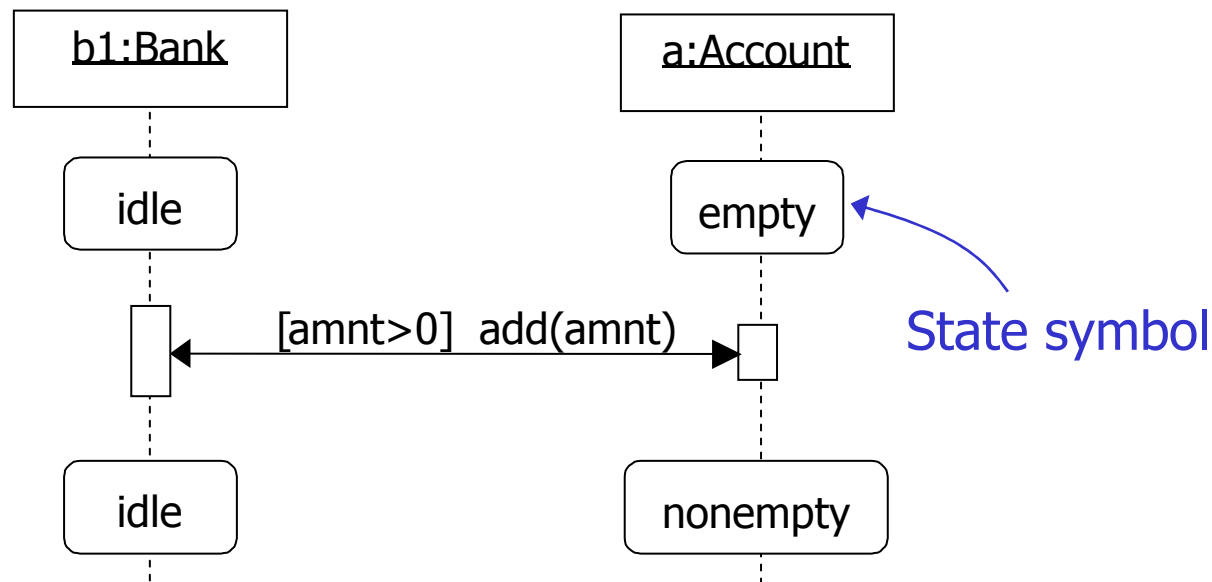


Sequence Diagrams: Time Bounds



Sequence Diagrams: State Markers

- Main use for SDs: “global” communication patterns
- Every interaction can change component state!



Summary Interaction Diagrams

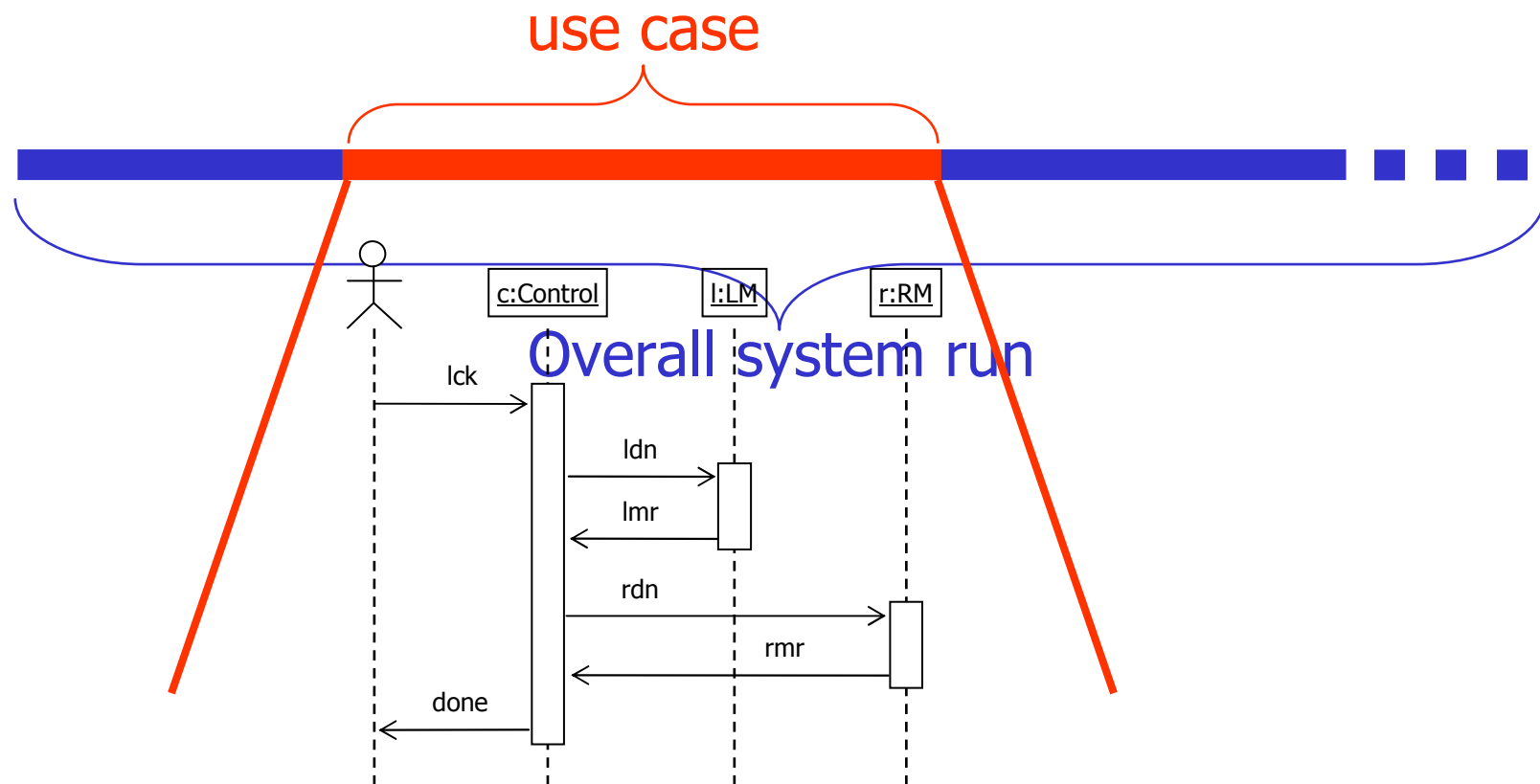
- UML offers
 - Sequence diagrams
 - Collaboration diagramsfor modeling **communication patterns**
- SDs support
 - neutral, asynchronous, synchronous communication
 - alternatives, repetition, object creation/deletion, time bounds, state symbols
- High potentials for interface specification (structure & behavior)

Overview

- Components and their Roles
- Interaction Diagrams
- Statecharts

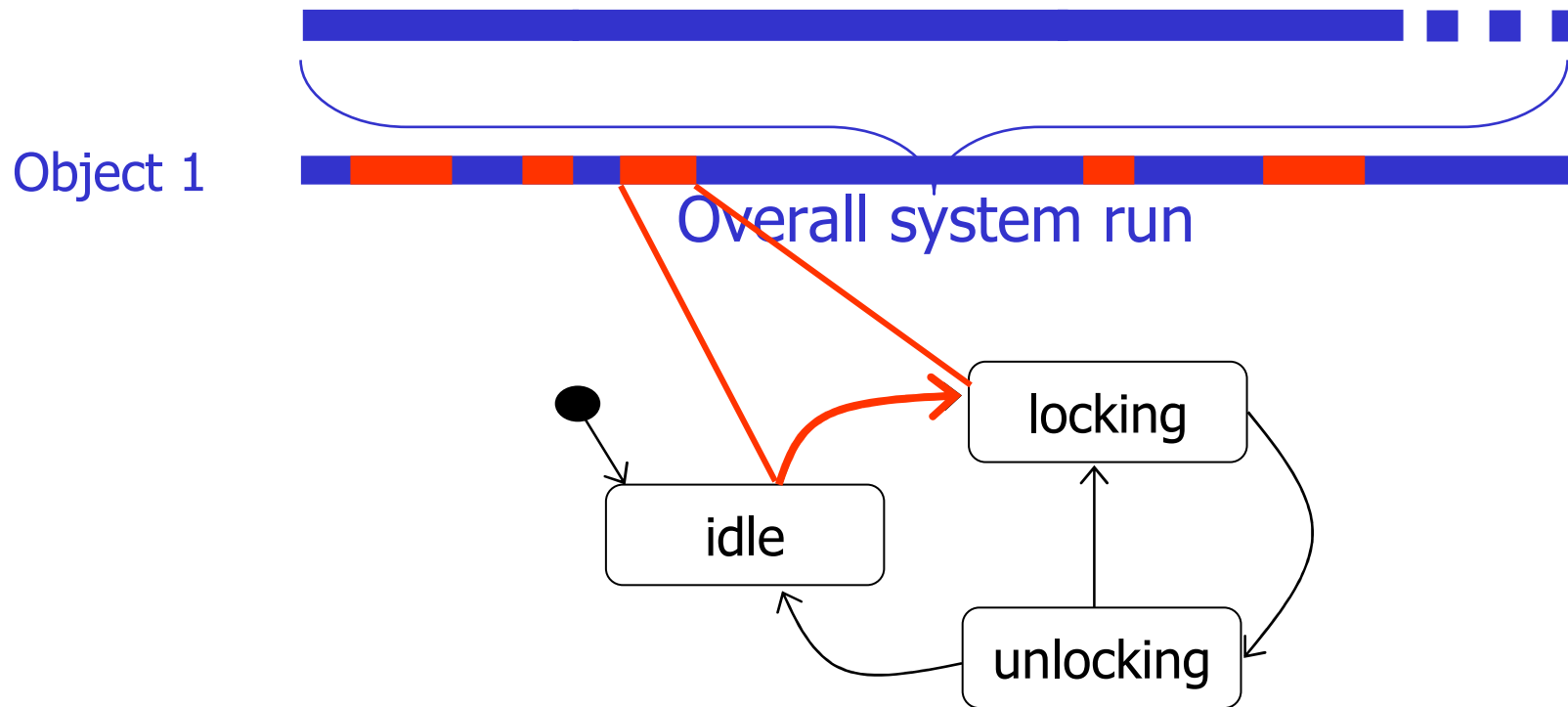
From Interaction Models to State Models

- Interaction model:
⇒ Projection of system behavior onto services/use cases



From Interaction Models to State Models

- State model:
⇒ Projection of system behavior onto individual components



From Interaction Models to State Models

"A state machine is a localized view of an object, a view that separates it from the rest of the world and examines its behavior in isolation. It is a reductionist view of a system. This is a good way to specify behavior precisely, but often it is not a good way to understand the overall operation of a system." [RJB99]

From Interaction Models to State Models

State:

Qualitative “snapshot” of an object, defined by

- Assignments to attributes
- Control state (program counter A implicit attribute)
- Defined reactions to incoming messages

Transition:

- State change
- Triggered by event (such as message receipt)
- Guarded by constraint (such as certain data state)
- Defines reactions to incoming messages

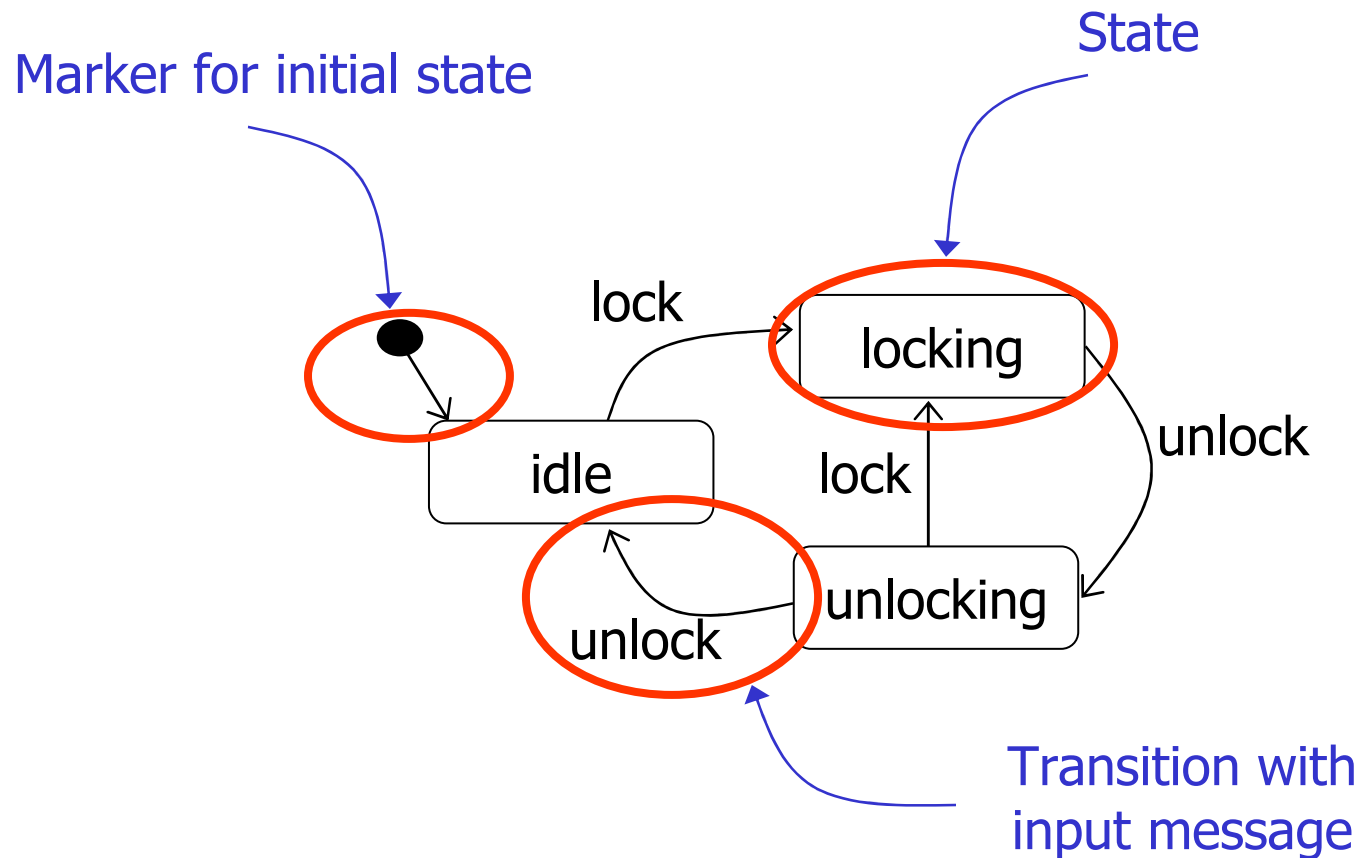
From Interaction Models to State Models

History of statecharts

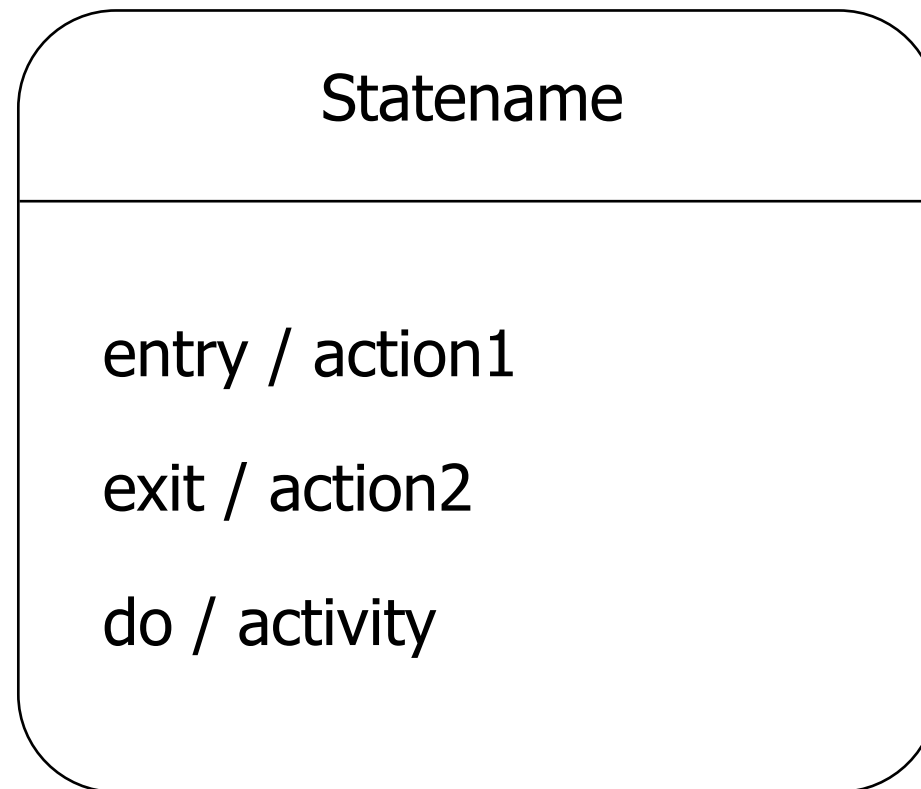
- Background: Theory of finite automata ($\sim$ 1950-60)
- Origin (David Harel, 1987):
 - Mealy-/Moore-Automata
 - Hierarchy
 - Concurrency
- Tool support for statecharts available
- Many semantics definitions for statecharts

statecharts – Basics

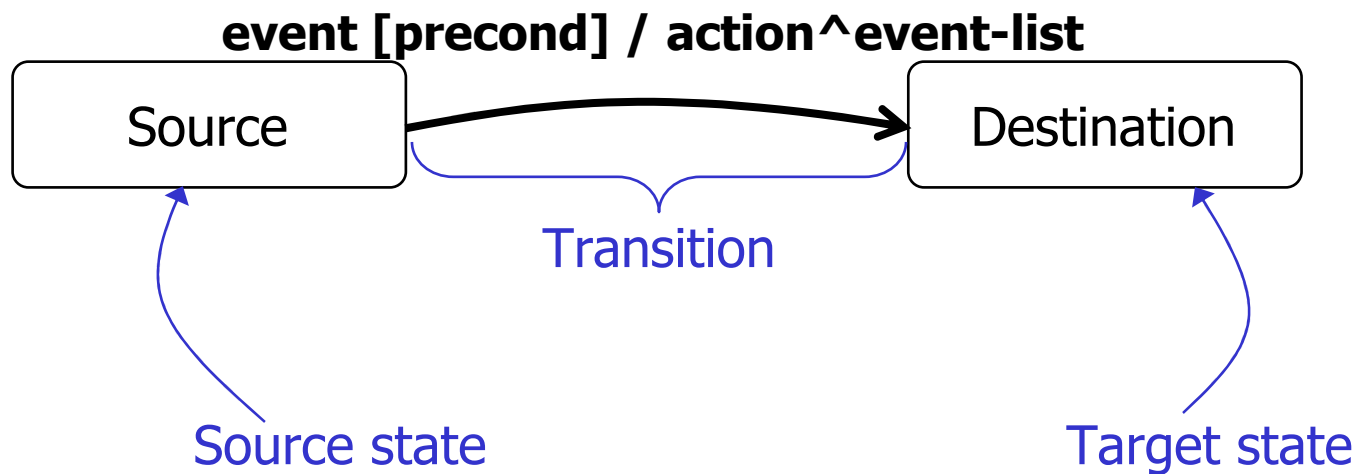
Example: Controller for Central Locking System



statecharts – Basics

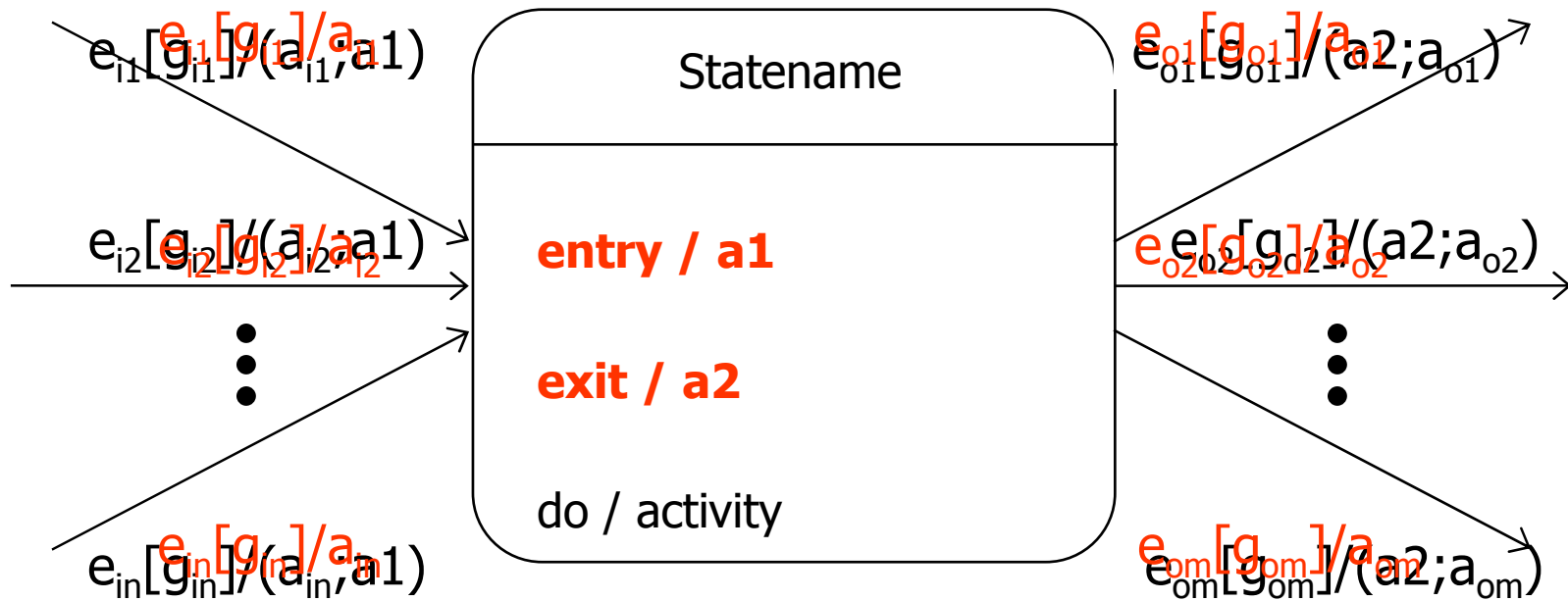


statecharts – Basics



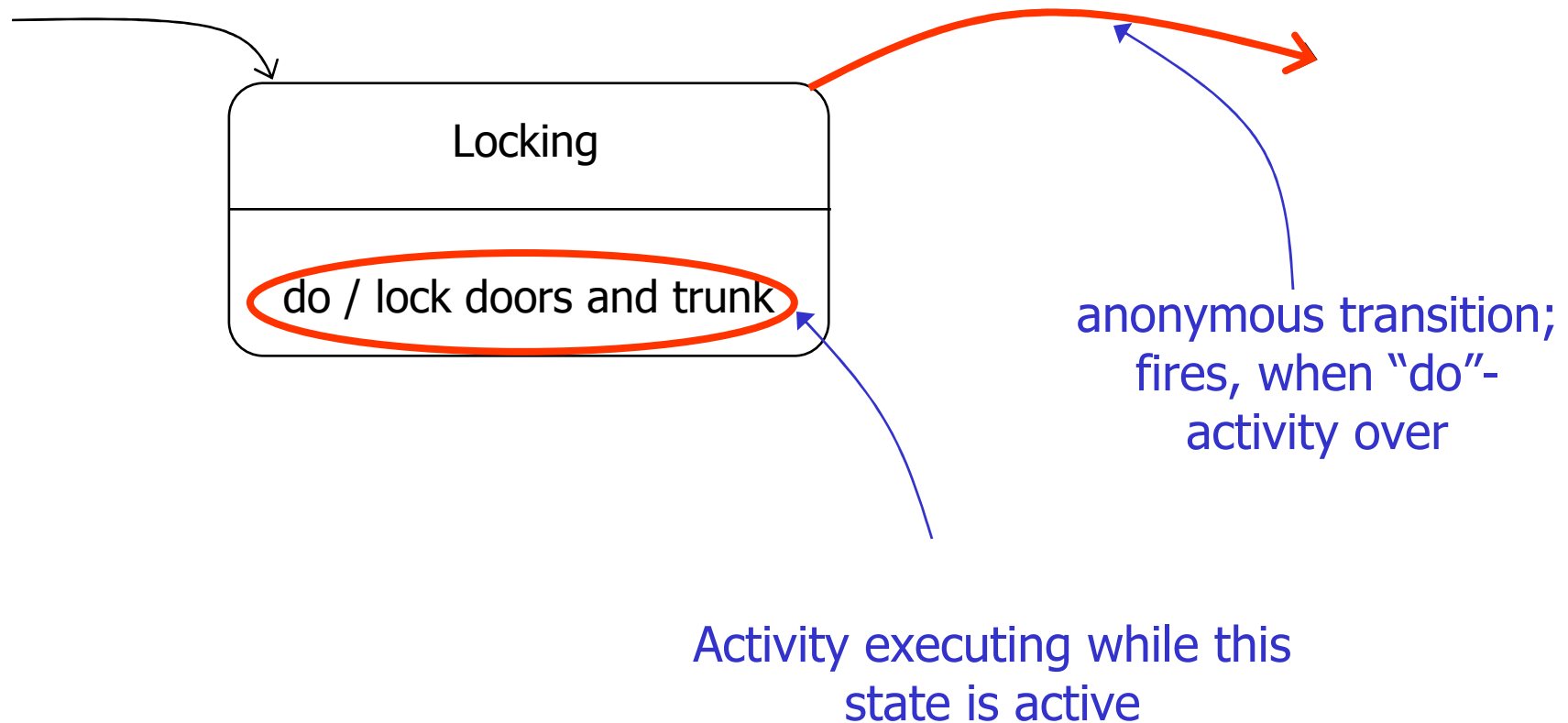
statecharts – Basics

entry- and exit-Actions



statecharts – Basics

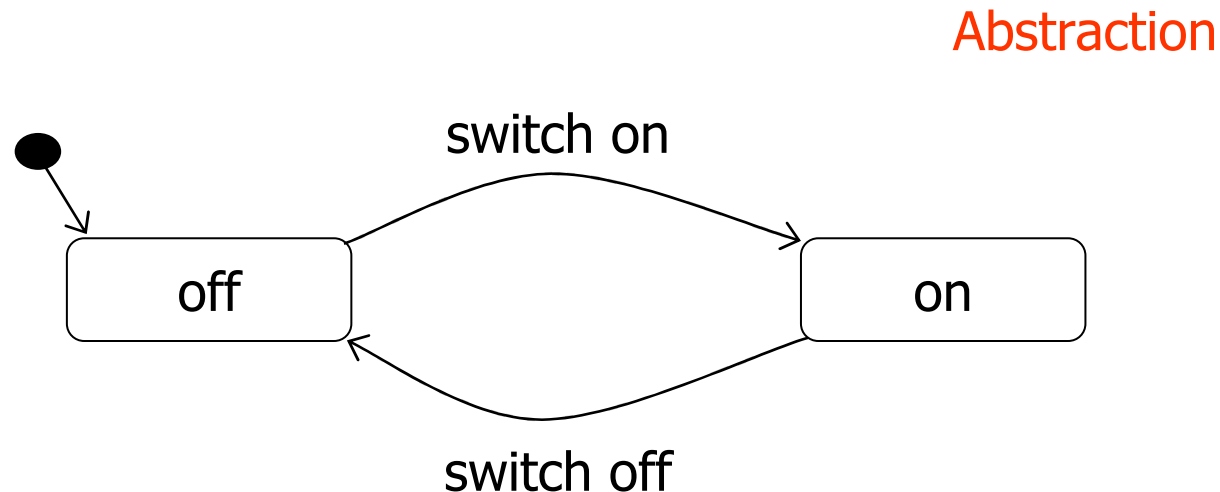
State activities



Hierarchy: OR-states/AND-states

- Problem of “flat” automata
 - ⇒ No handle at abstraction
 - ⇒ “Explosion” of graphical notation

- Examples:



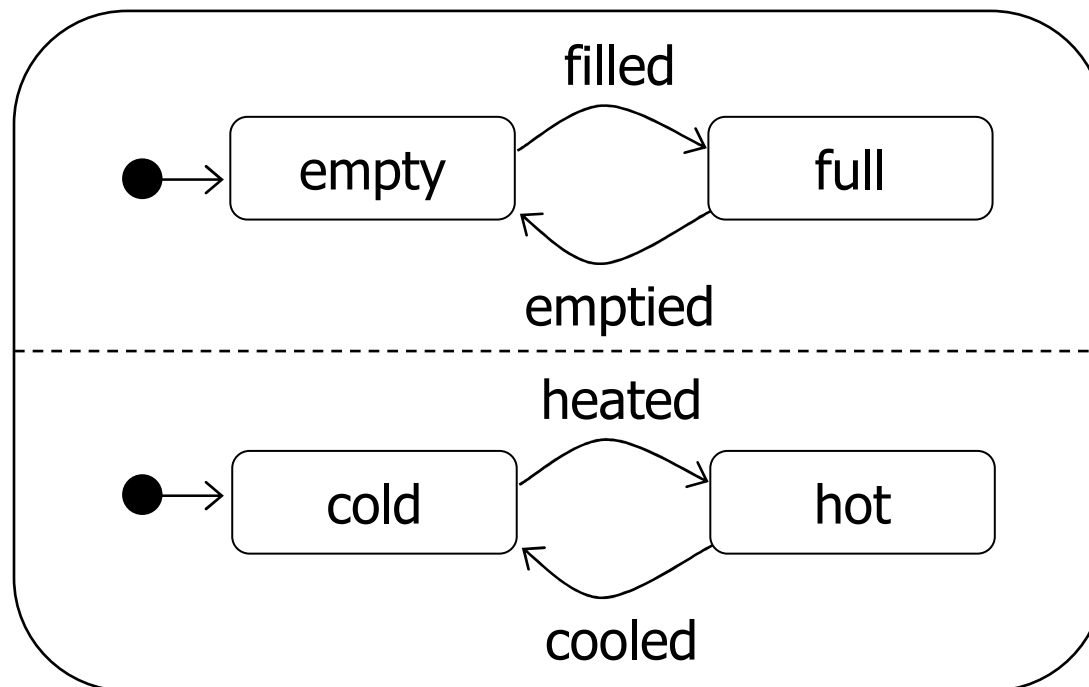
Hierarchy: OR-states/AND-states

- Problem of “flat” automata
 - ⇒ No handle at abstraction
 - ⇒ “Explosion” of graphical notation

- Examples:

Combined properties

“ideal picture”

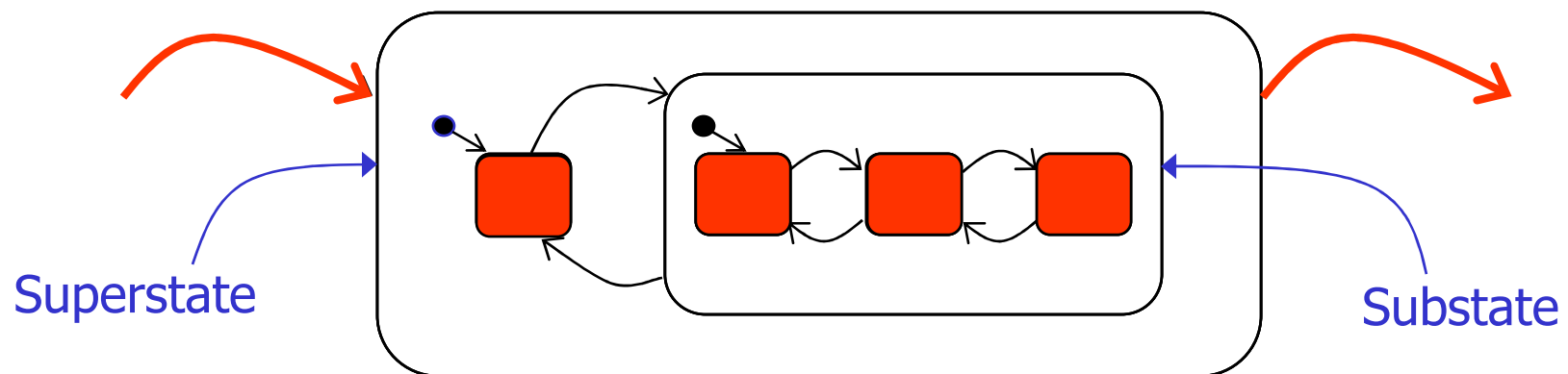


Hierarchy: OR-states/AND-states

Solution in UML statecharts:

OR-states:

- Hierarchical
- Every state can contain an entire statechart
- At any time only one “substate” of a “superstate” is active

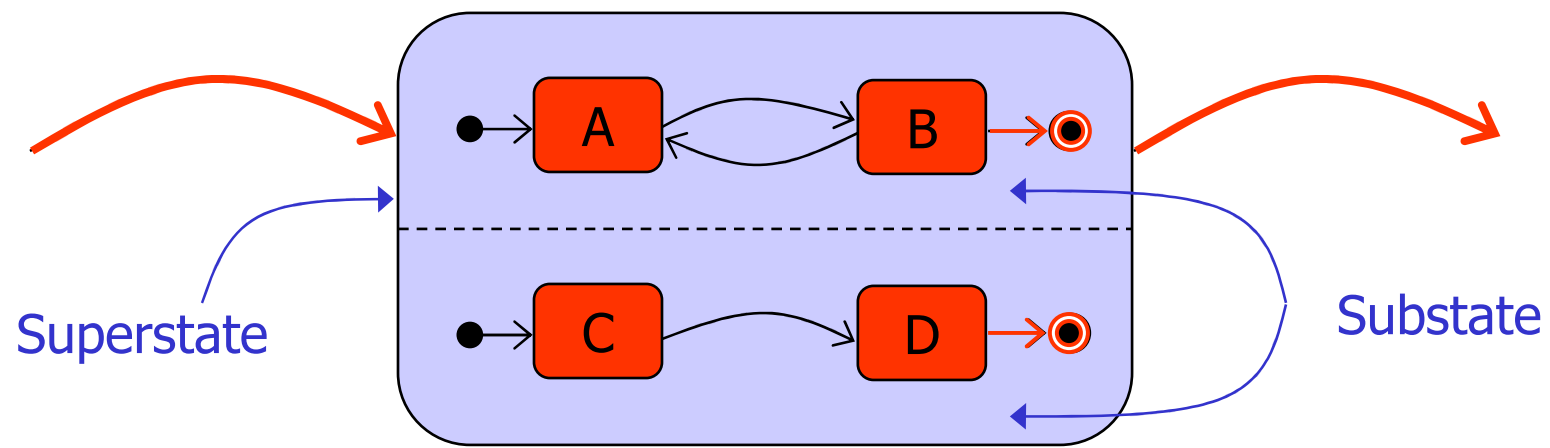


Hierarchy: OR-states/AND-states

Solution in UML statecharts:

AND-states:

- Hierarchical, concurrent states (Replacement for “product automata”)
- At any time every substate of a superstate is active
- Substates can communicate via synchronous signal exchange



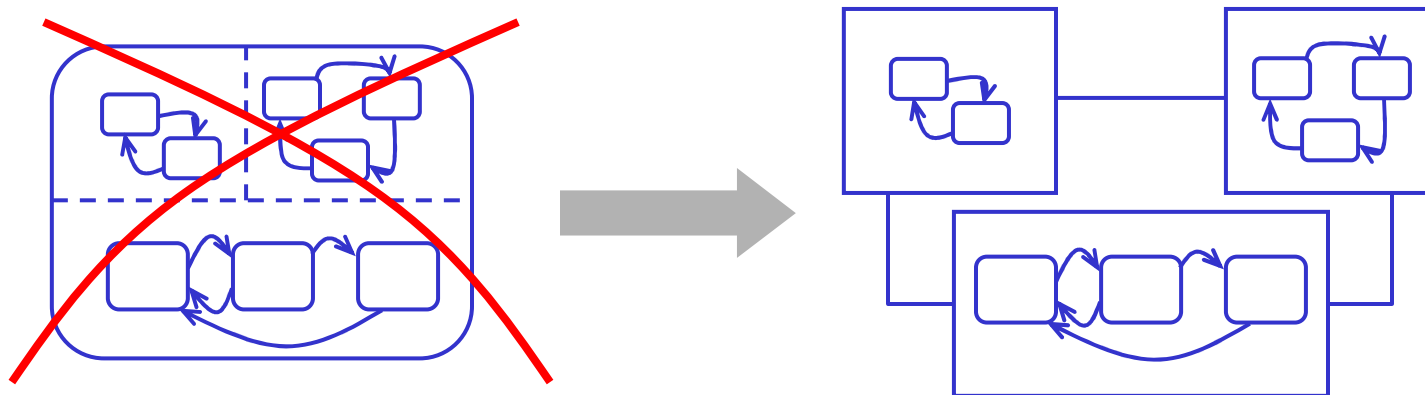
Hierarchy: OR-states/AND-states

Beware: Broadcasting in AND-states

Transition rules for AND-states with broadcast communication can lead to incomprehensible specs!

Better:

explicit message exchange between separate objects via corresponding interfaces



Summary statecharts

- UML offers statecharts for modeling **state-based** behavior of **individual** components
- statecharts support
 - Hierarchy: OR states
 - Concurrency: AND states
 - Broadcasting communication model
- Rule of thumb: prefer use of concurrent components over concurrent states

