



Foto: Thomas Josek

Software Engineering

System Modeling

Software & Systems Engineering | Prof. Dr. Andreas Vogelsang | 23.10.2023



@andivogelsang



vogelsang@cs.uni-koeln.de

Learning Goals for Today

- Understand what modeling is and what it is good for
- Understand what the Unified Modeling Language (UML) is
- Understand what activity diagrams are and what they are used for
- Create activity diagrams to model flows of actions
- Understand what state machine diagrams are and what they are used for
- Create state machine diagrams to model states and their transitions



System Modeling: Why?

Motivation for Modeling

UML User Guide

“A successful software organization is one that consistently deploys **quality software** that meets the needs of its users. An organization that can develop such software in a **timely and predictable** fashion, with an **efficient and effective use of resources**, both human and material, is one that has a sustainable business. [...]

Modeling is a central part of all the activities that lead up to the deployment of good software. We build models to **communicate** the desired structure and behavior of our system. We build models to **visualize and control** the system's architecture. We build models to better **understand** the system we are building, often exposing opportunities for **simplification and reuse**. And we build models to **manage risk**.”

UML User Guide

“We build models of complex systems because we cannot comprehend such a system in its entirety”

Recap: SE vs. Programming



What is System Modeling?

System Modeling

“**System modeling** is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. [...] Models are used during the requirements engineering process to help derive the **detailed requirements** for a system, during the design process to **describe the system to engineers** implementing the system, and after implementation to **document the system's** structure and operation.”

[Sommerville]



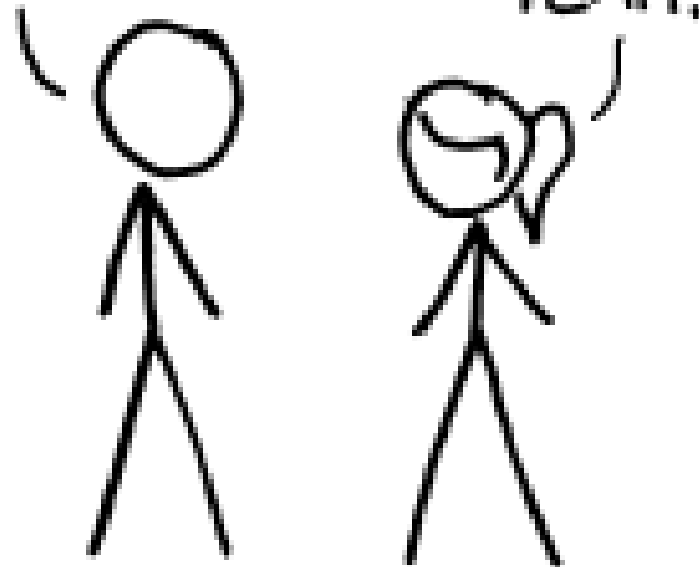
The Unified Modeling Language (UML)

HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

The Unified Modeling Language

UML

“The Unified Modeling Language (UML) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system.”
[UML Reference Manual]

UML User Guide

“Modeling yields an understanding of a system. No one model is ever sufficient. Rather, you often need multiple models that are connected to one another [...].”

Different Kinds of UML Diagrams

Structure Diagrams (Strukturdiagramme)

“**Structure diagrams** show the static structure of the objects in a system. That is, they depict those elements in a specification that are irrespective of time. The elements in a structure diagram represent the meaningful concepts of an application, and may include abstract, real-world and implementation concepts.”

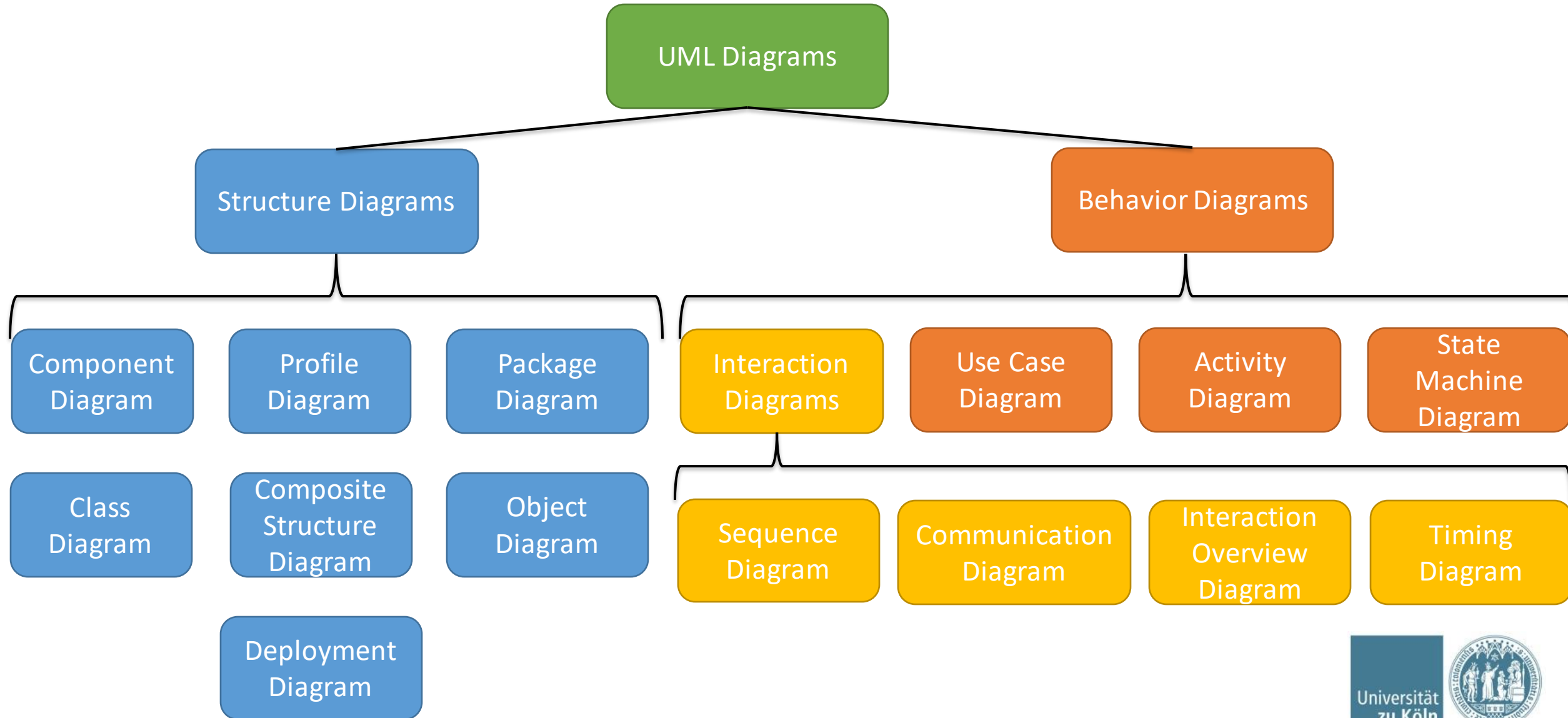
[UML 2.5.1]

Behavior Diagrams (Verhaltensdiagramme)

“**Behavior diagrams** show the dynamic behavior of the objects in a system, including their methods, collaborations, activities, and state histories. The dynamic behavior of a system can be described as a series of changes to the system over time.”

[UML 2.5.1]

14 Types of UML Diagrams





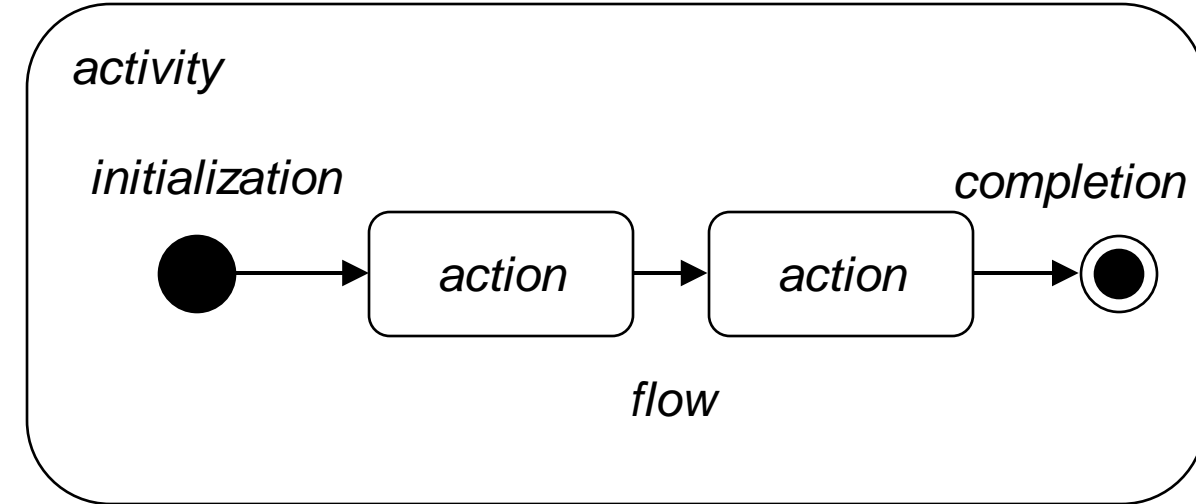
Activity Diagrams

Activity Diagrams

Activity Diagram (Aktivitätsdiagramm)

An **activity diagram** is a diagram visualizing **activities** and their order of execution. An activity contains **actions** (rounded box) that are connected by means of **flows** (solid arrows). The execution begins at the **initialization** (filled circle) and ends with the **completion** node (bull's eye).

(Aktivität, Fluss, Startzustand, Endzustand)

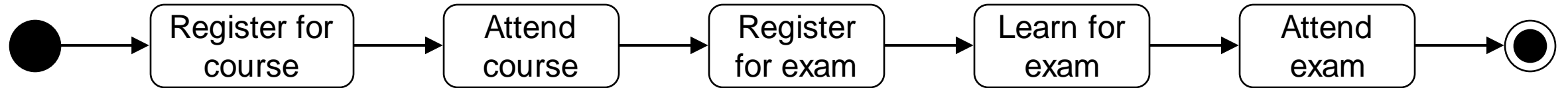


Rules for Activity Diagrams

- exactly one initialization/completion node
- at least one action
- every action has exactly one incoming and one outgoing flow
- every action is reachable from initialization
- completion is reachable from every action

Example of Sequential Activities

Attend a course



Branching and Merging in Activity Diagrams

Branching and Merging [UML User Guide]

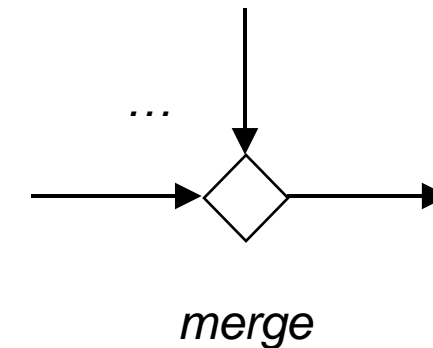
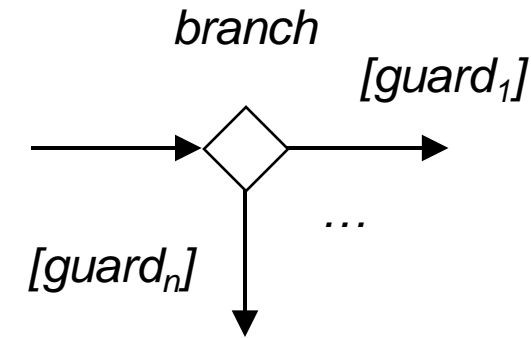
Motivation: model control flow that depends on certain conditions (i.e., actions that may happen)

Branching: A **branch** has exactly one incoming and two or more outgoing flows. Each outgoing flow has a Boolean expression called **guard**, which is evaluated on entering the branch. (*Verzweigung*)

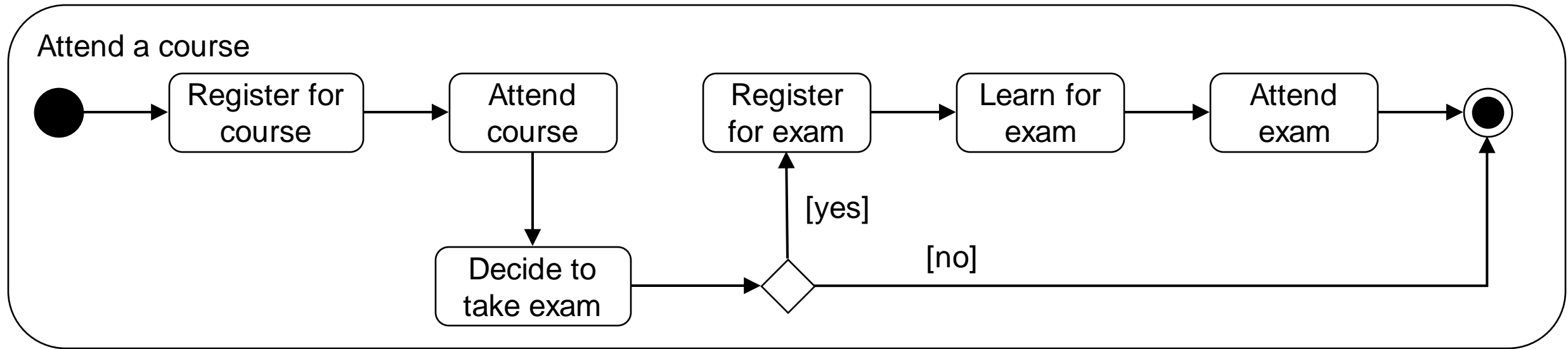
Merging: A **merge** has two or more incoming and exactly one outgoing flow. (*Zusammenführung*)

Further Rules for Activity Diagrams

- guards on outgoing flows should not overlap (flow of control is unambiguous)
- guards should cover all possibilities (flow of control does not freeze)
- keyword **else** possible for one guard (*sonst*)



Example of Conditional Activities



Forking and Joining in Activity Diagrams

Forking and Joining [UML User Guide]

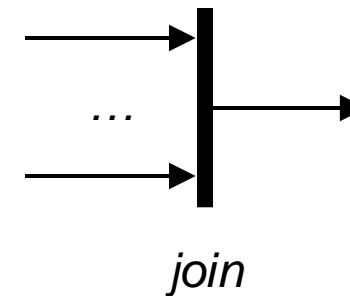
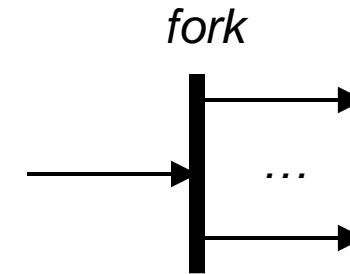
Motivation: model concurrent control flows (i.e., activities that run in parallel)

Forking: A **fork** (thick horizontal or vertical line) has exactly one incoming and two or more outgoing flows. ([Gabelung](#))

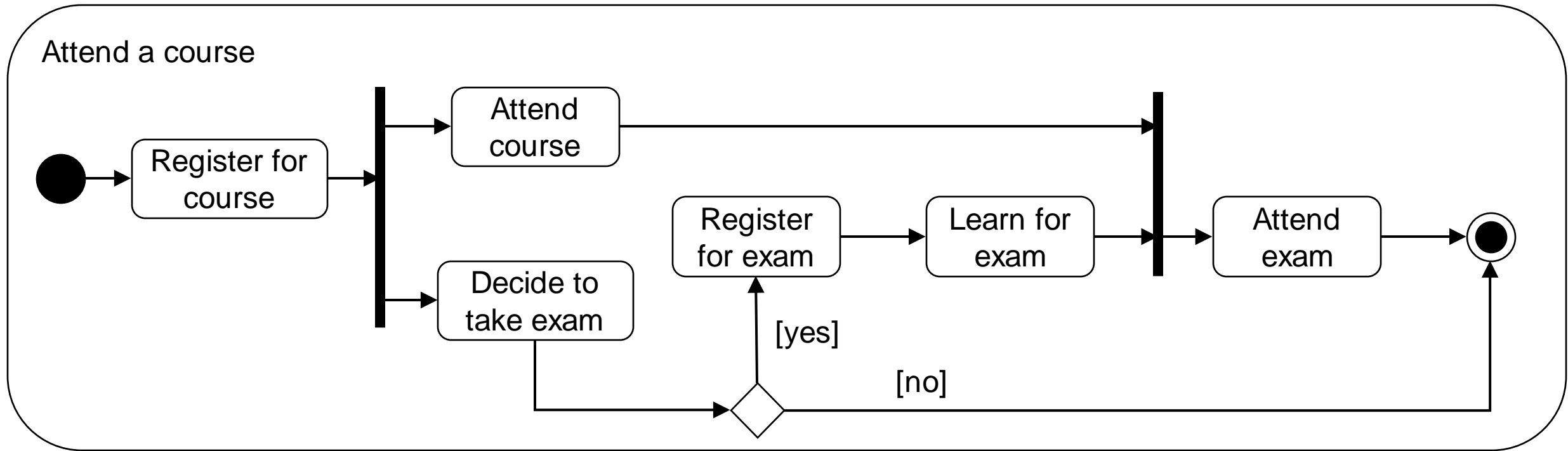
Joining: A **join** (thick horizontal or vertical line) has two or more incoming and exactly one outgoing flow. ([Vereinigung](#))

Further Rules for Activity Diagrams

- branched paths must be merged eventually ([letztendlich](#))
- forked paths must be joined eventually
- only outgoing edges of branch nodes have guards



Example of Concurrent Activities



Swimlanes in Activity Diagrams

Swimlanes [UML User Guide]

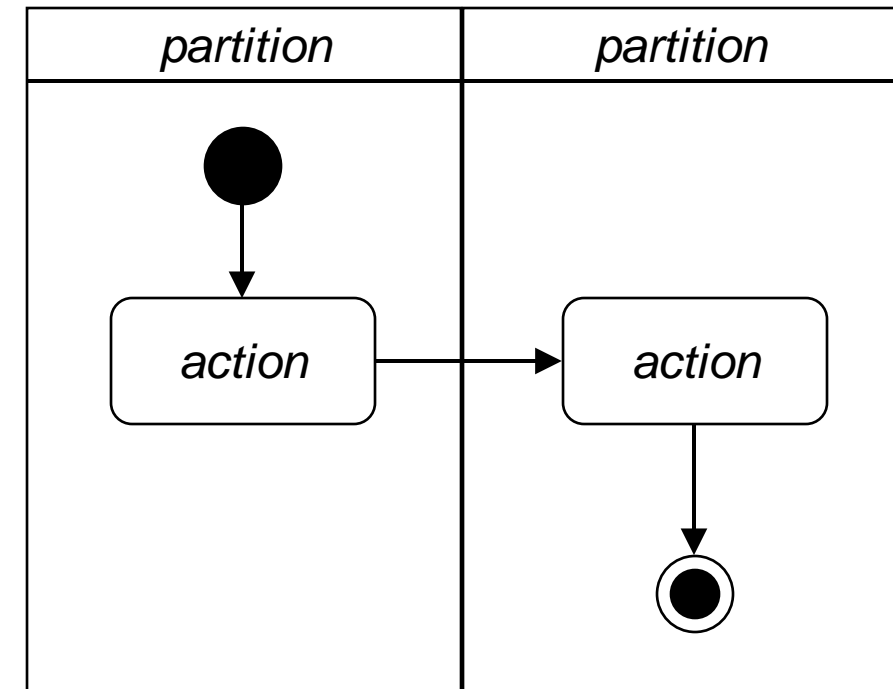
Motivation: group activities according to responsibilities

Swimlane: An activity diagram may have no or at least two swimlanes. A **swimlane** (rectangle) represents a high-level responsibility activity within an activity diagram. ([Verantwortlichkeitsbereiche](#))

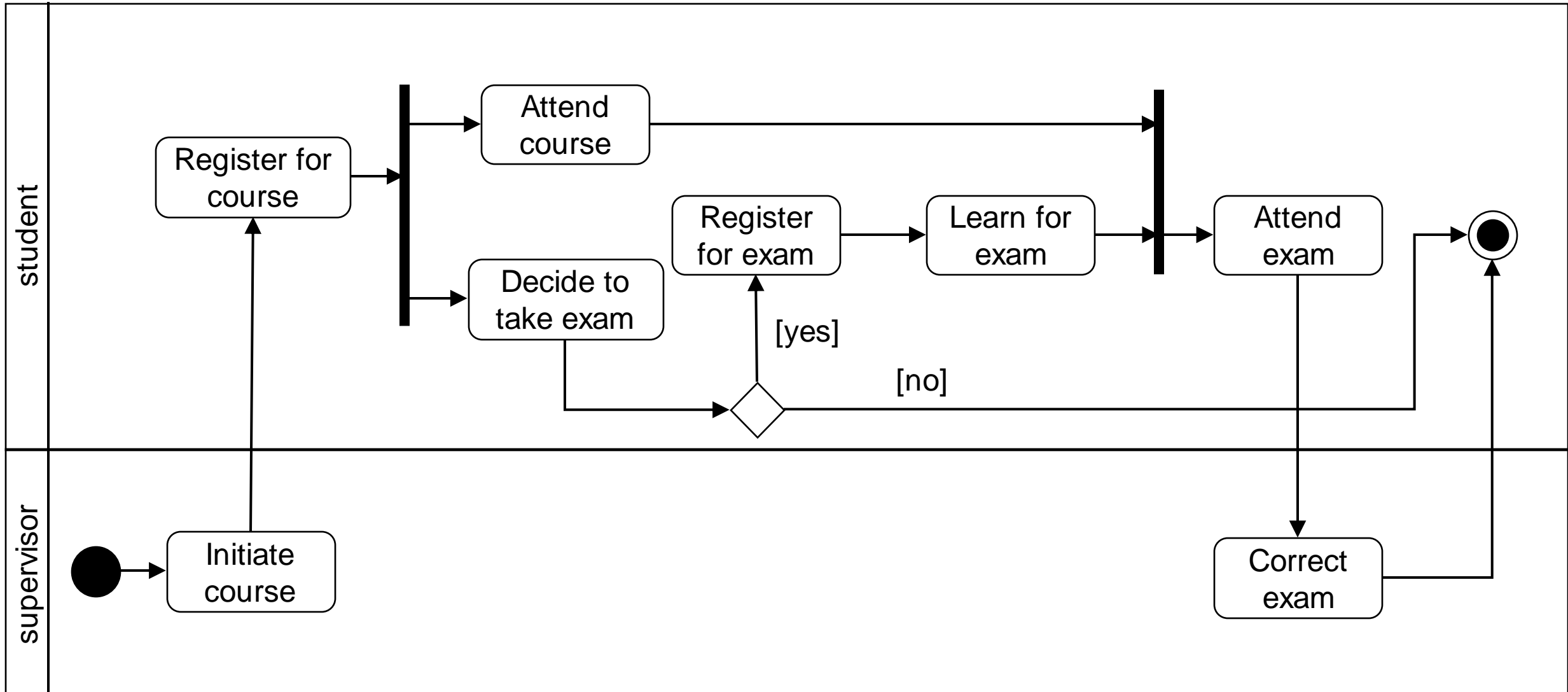
Further Rules for Activity Diagrams

- each swimlane has a name unique within its diagram
- every activity belongs to exactly one swimlane
- only flows may cross swimlanes

swimlanes



Example of Activities with Swimlanes



Activity Diagram Semantics

AD Semantics

Motivation: which behaviors are defined by an activity diagram?

Token-based Semantics: The possible flows in an activity diagram are defined by the possible token-flows. Elements of an activity diagram process tokens in different ways.

Token propagation rules

Init node: Produces exactly one token

Action: If the action receives a token, it also produces one token

Completion node: Consumes all tokens it gets (“sink”)

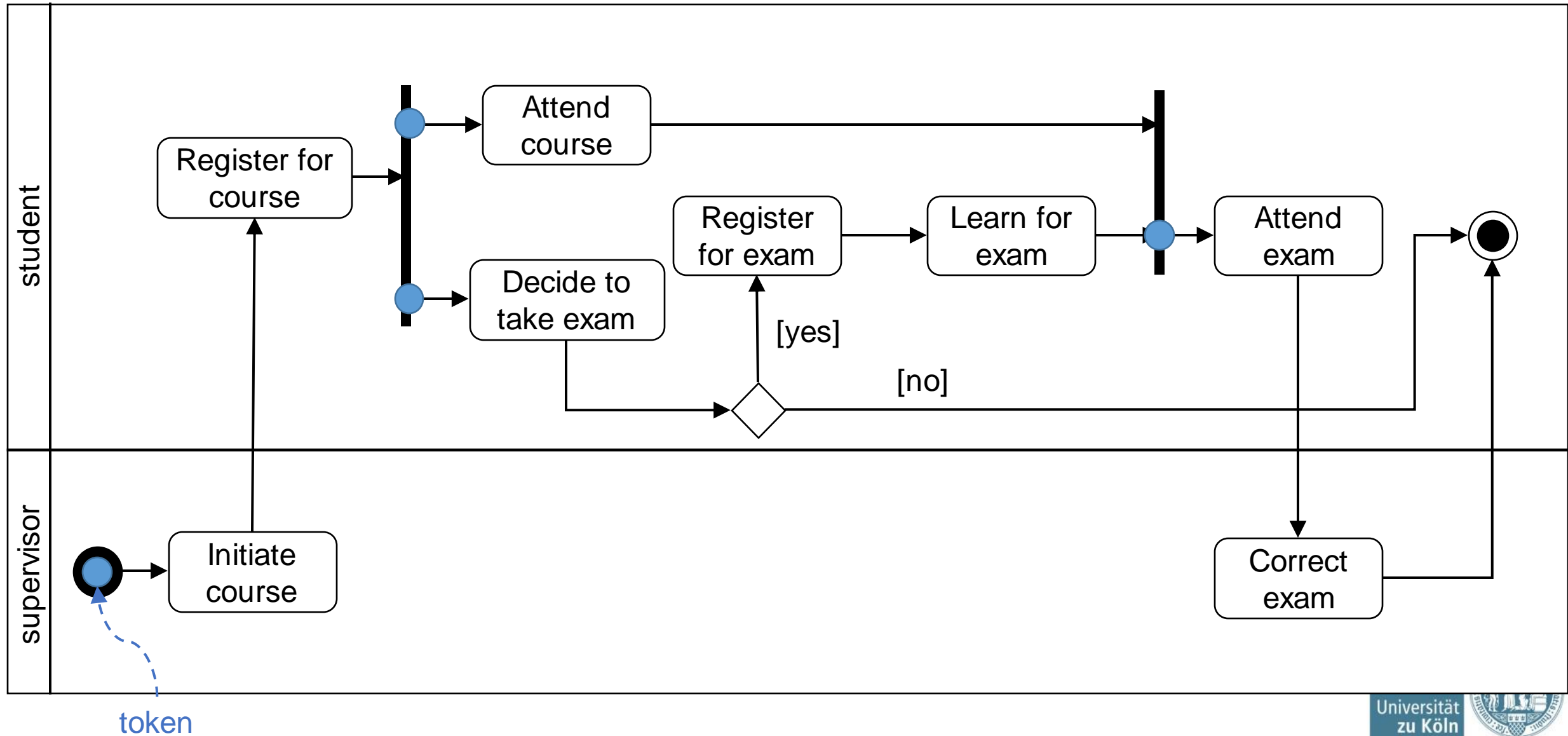
Branch node: If the branch node receives a token, it produces exactly one token on the flow where the guard evaluates to true.

Merge node: If the merge node receives a token on any of its incoming flows, it produces one token on the outgoing flow.

Fork node: If the fork node **receives** a token, it produces one token on all its outgoing flows.

Join node: If the join node has received a token on all its incoming flows, it produces one token on the outgoing flow.

Example of Token-based Semantics

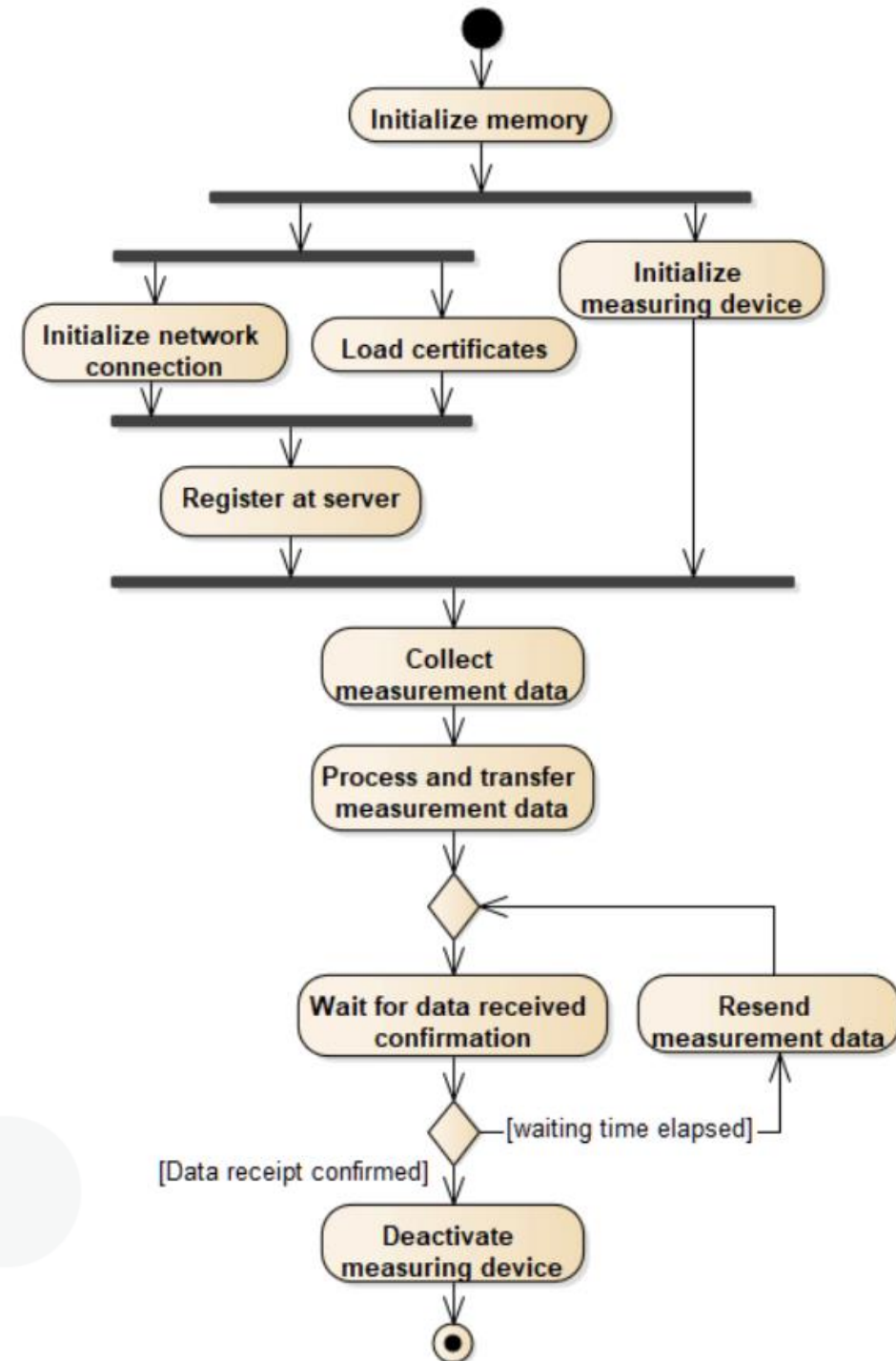


Quick check

Which of the following statements match the diagram?

- *Initialize measuring device* must happen prior to *Register at server*.
- *Register at server* happens as soon as *Load certificates* is ready.
- *Initialize network connection* and *Load certificates* must finish at the same time.
- *Deactivate measuring device* is executed as soon as *Data receipt confirmed* is true.

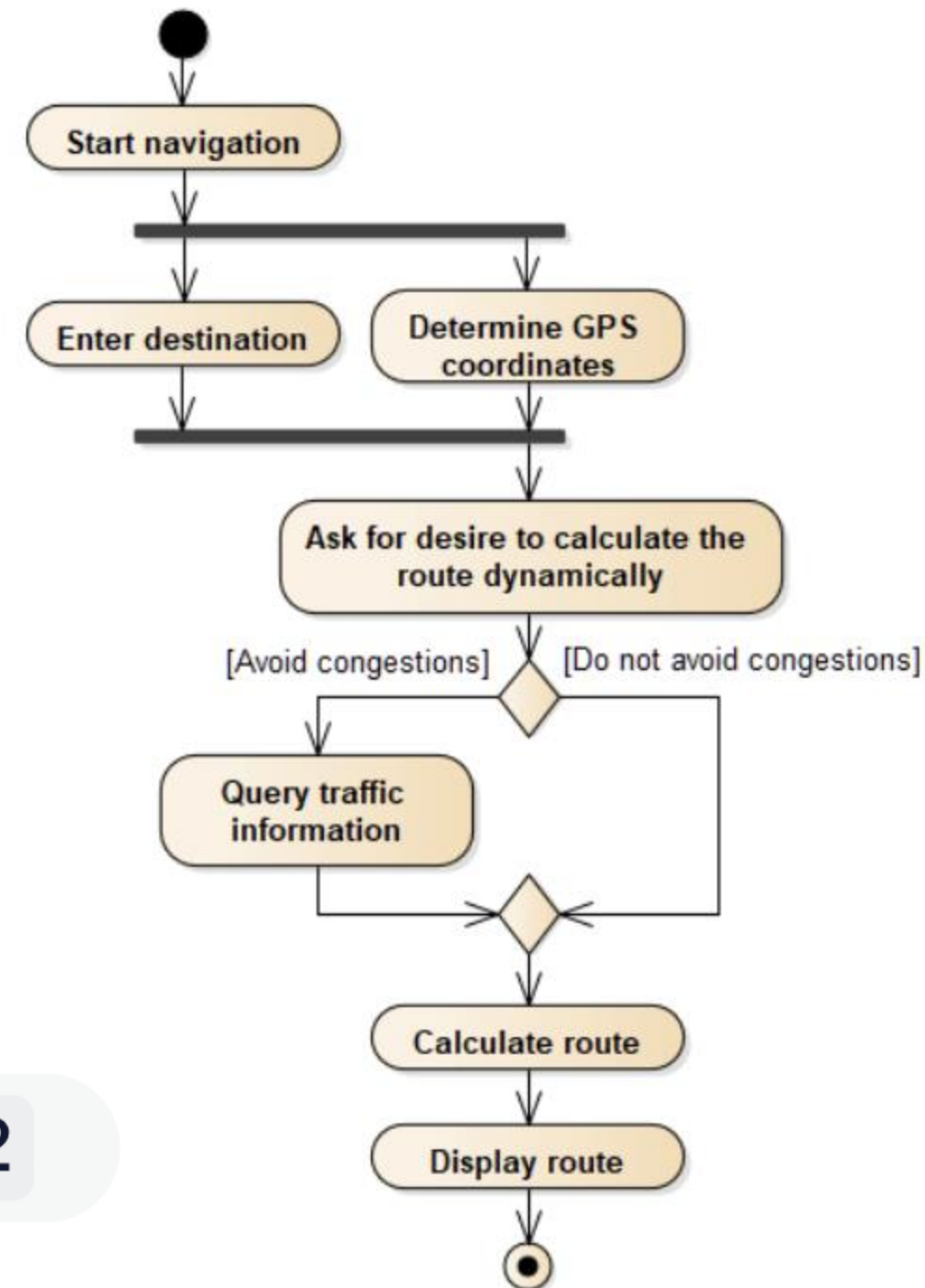
Join at **menti.com** use code **4388 7642**



Quick check

Which of the following statements match the diagram?

- A route can be calculated without querying traffic information.
- A route can be calculated after querying traffic information.
- The system can ask for the desire to calculate the route dynamically without having to determine the GPS coordinates first.
- The order of *Enter destination* and *Determine GPS coordinates* is arbitrary.



Join at **menti.com** use code **4388 7642**

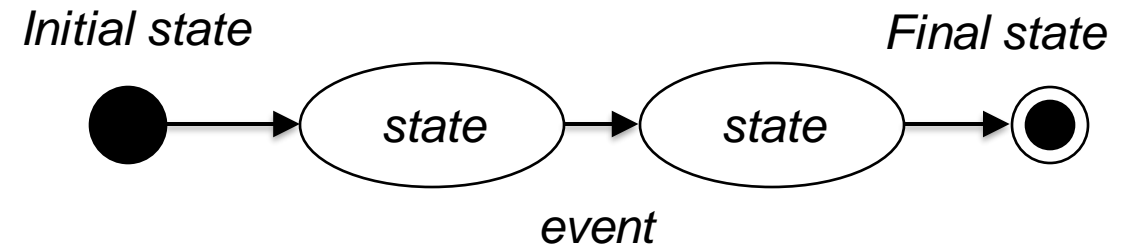


State Machine Diagrams

State Machine Diagrams

State Machine Diagram (Zustandsdiagramm)

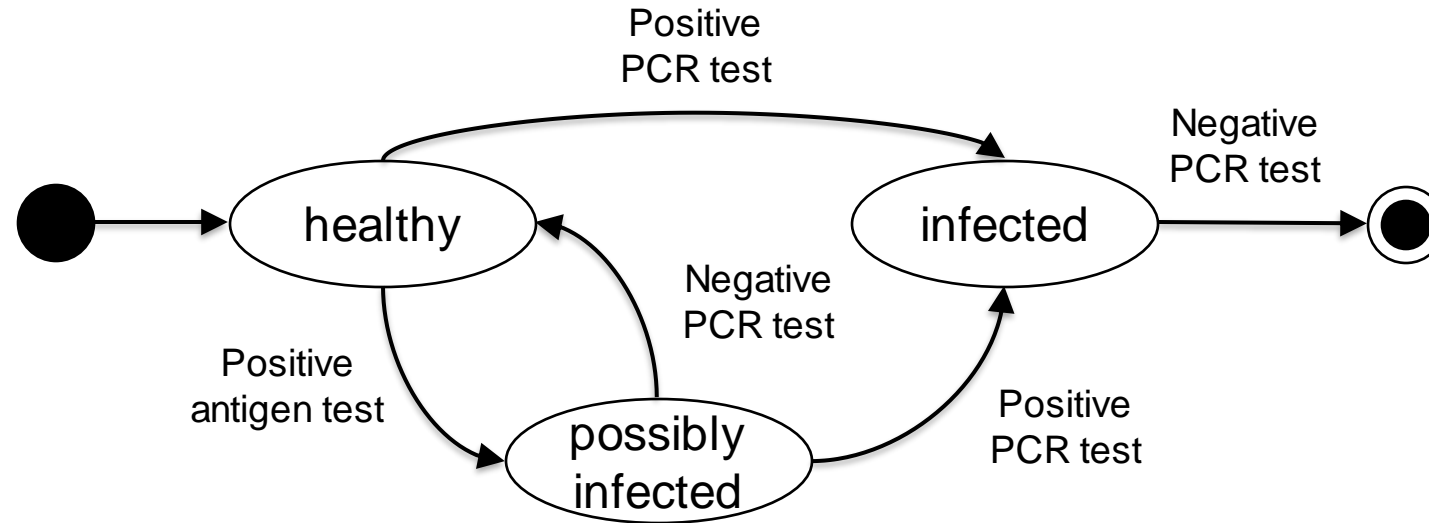
A **state machine diagram** specifies the sequences of states the (a part of) the system goes through during its lifetime in response to events, together with its responses to those events. Every **state** (oval) is characterized by a condition or situation. An **event** is an occurrence of a stimulus that can trigger a state transition. A **transition** (solid arrow) is a relationship between two states. (Zustand, Ereignis, Zustandsübergang)



Rules for State Machine Diagrams

there is a single **initial state** (filled circle) and a single **final state** (bull's eye) (Start- und Zielzustand) — see exception below

Example of a State Machine Diagram



Hierarchical State Machine Diagrams

Single and Composite State [UML User Guide]

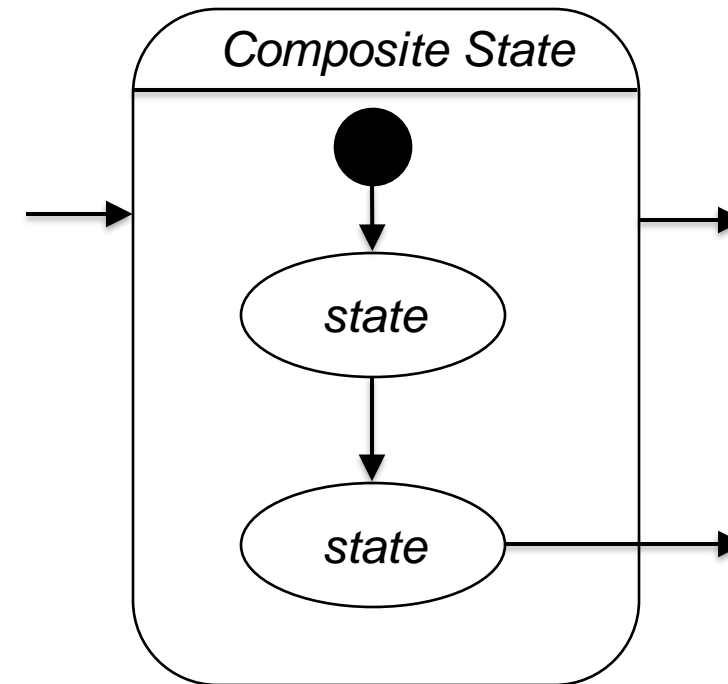
Motivation: avoid duplicated transitions, improve overview in complex state machine diagrams

Simple State: A simple state is a state that has no substructure. (*einfacher Zustand*)

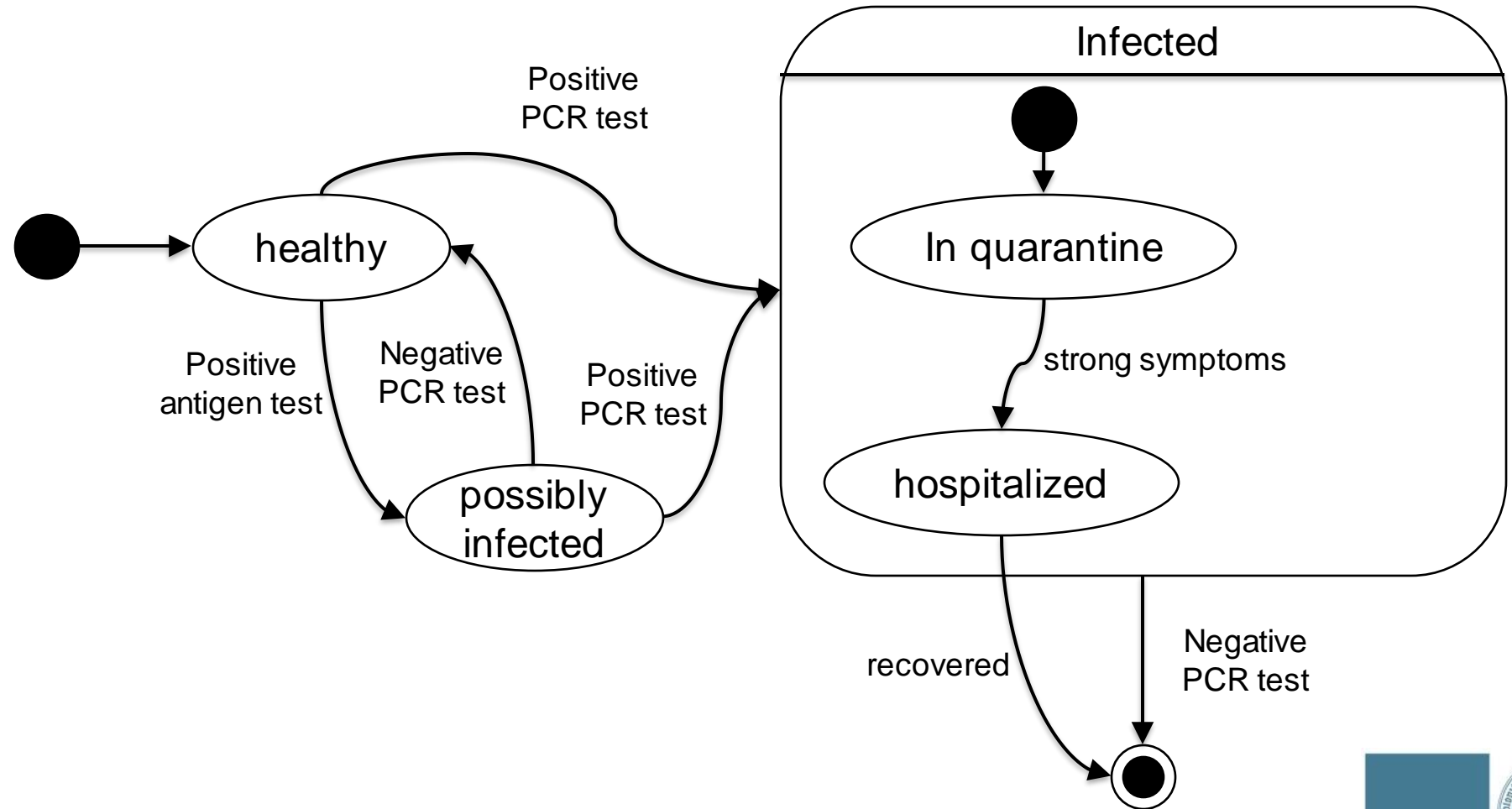
Composite State: A state that has substates (i.e., nested states) is called a composite state. (*komplexer Zustand*)

Rules for State Machine Diagrams

- every composite state has its own single initial state (*Startzustand*)
- substates may be nested to any level



Example of a State Machine Diagram



State Machine Diagram Semantics

SM Semantics

Motivation: which behaviors are defined by a state machine diagram?

Finite State Machine Semantics: The possible behavior induced by a state machine diagram is the set of event sequences that are accepted by a corresponding finite state machine

Accepting an event sequence

State: The outgoing state transitions of the current state define the possible transitions that can be taken

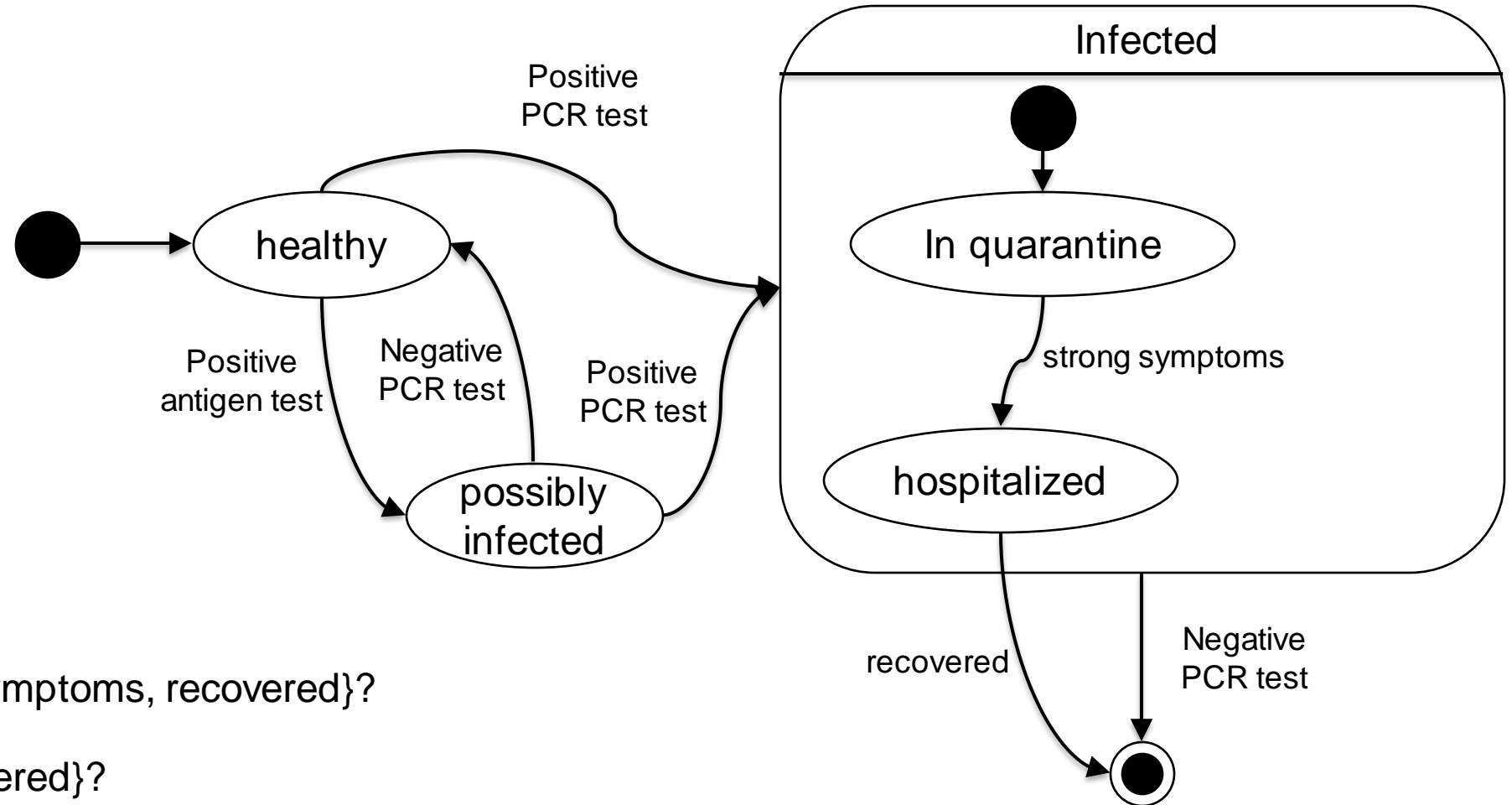
State transitions: Take the first event of the sequence and check if it matches any of the possible transitions. If a transition matches, take the transition, make the next state the current state, and remove the element from the sequence.

Final state: If the final state has been reached and the sequence is empty, the event sequence is accepted.

Non-acceptance: Event sequences are not accepted if

- (1) the sequence is empty, but the final state has not been reached
- (2) The final state has been reached but the event sequence is not yet empty
- (3) No transition matches the current event in the sequence

Example of a FSM Semantics



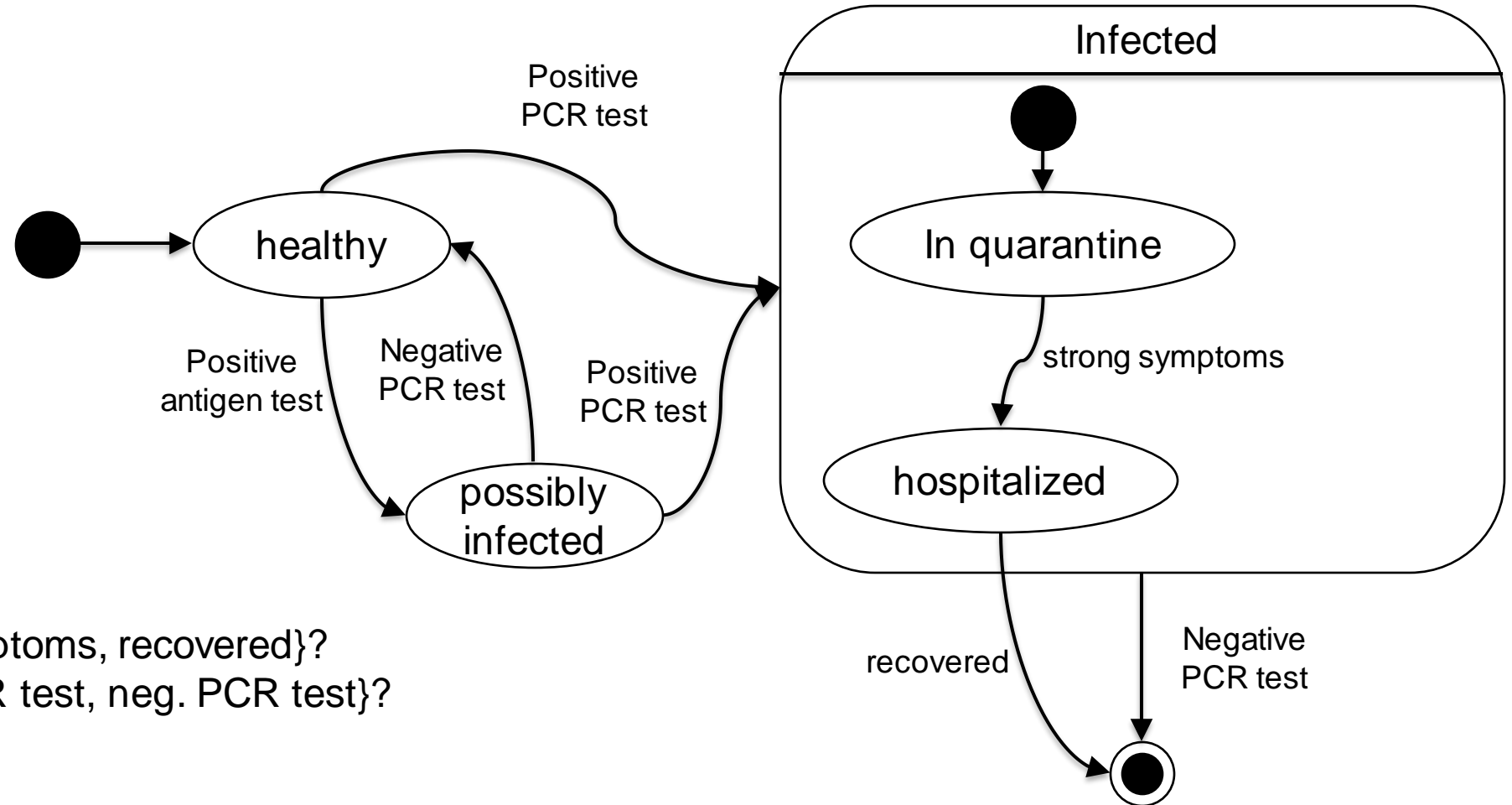
{pos. PCR test, strong symptoms, recovered}?

{strong symptoms, recovered}?

{recovered}?

{}

Example of a FSM Semantics



{pos. PCR test, strong symptoms, recovered}?
 {pos. antigen test, pos. PCR test, neg. PCR test}?

...

(pos. antigen test, neg. PCR test)* (pos. PCR test | (pos. antigen test, pos. PCR test))
 ((strong symptoms, recovered | (strong symptoms, neg. PCR test) | neg. PCR test)

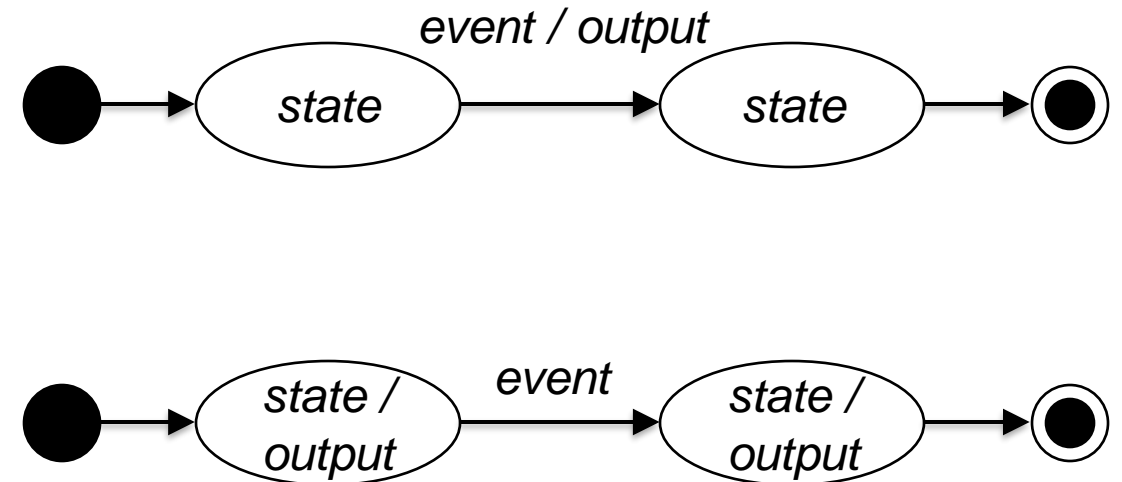
State Machine Diagrams with Output

Input/Output State Machines

Motivation: allow to attach output behavior to state machine diagrams

Output in transitions (Mealy Machines): While processing an input, the transition produces an output.

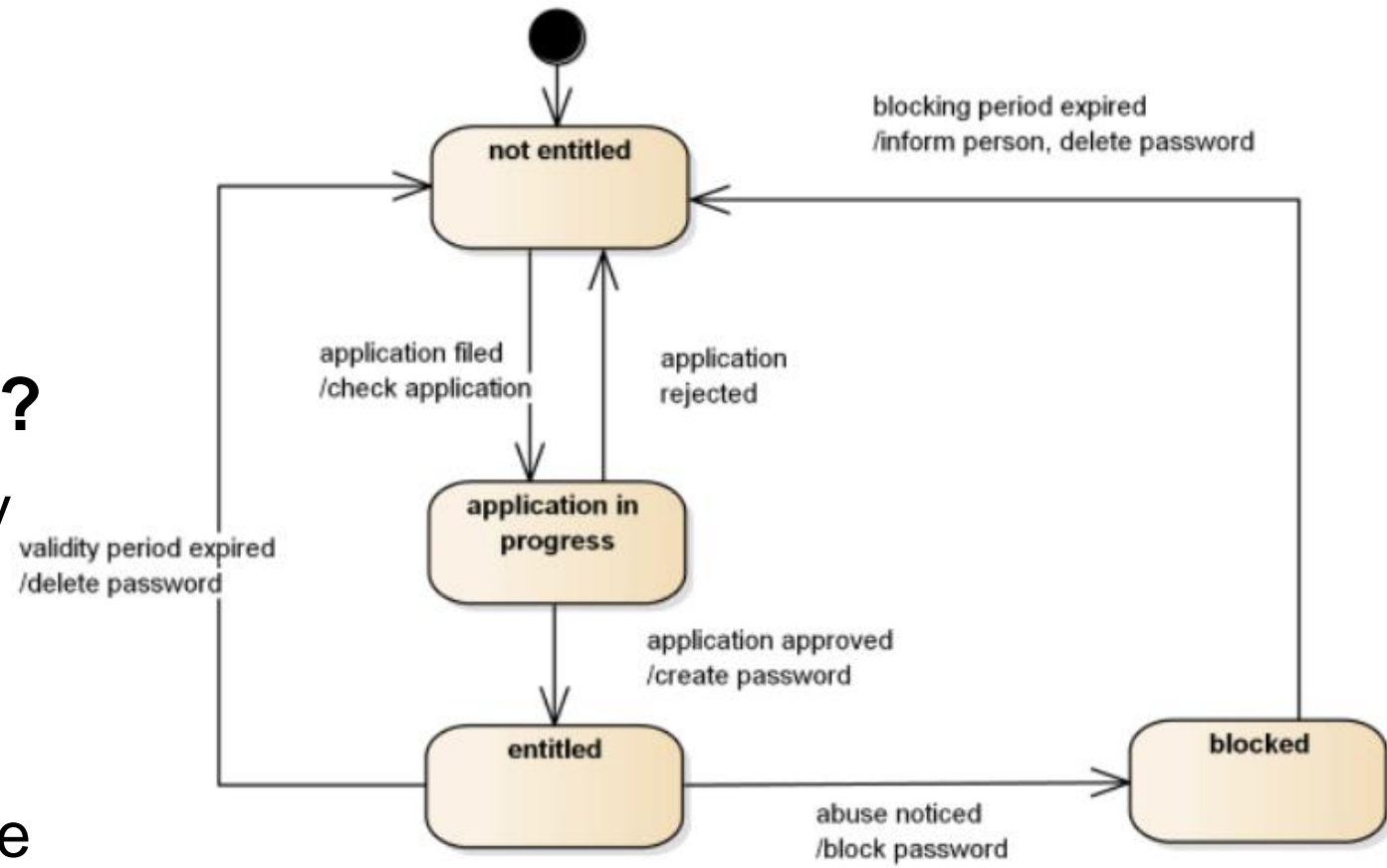
Output in states (Moore Machines): While entering/leaving a state, an output is produced.



Quick check

Which of the following requirements are modeled correctly in the state diagram?

- Blocked users can be unblocked by resetting the user's password.
- If abuse has been noticed, the password has to be blocked.
- If the validity period has expired, the password has to be deleted.
- If an application is approved, no new application can be created.



Join at menti.com use code **4388 7642**

Summary

Activity and State Machine Diagrams

We can visualize the dynamics of execution in two ways: by emphasizing the flow of control from activity to activity (**activity diagrams**) or by emphasizing the potential states and transitions among those states (**state machine diagrams**).