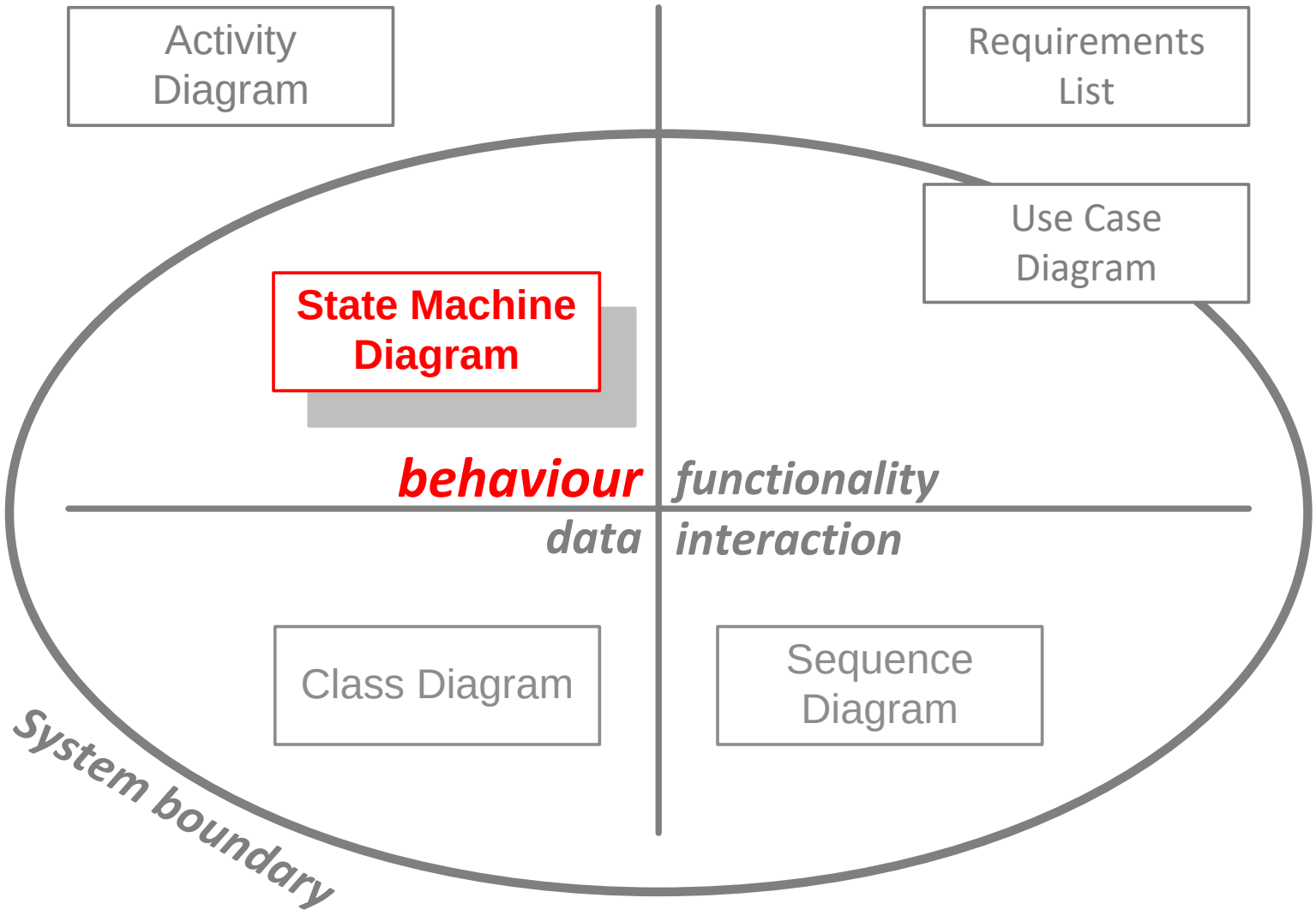
Software Systems

Design – 3A
State Machine Diagrams

Joke van Staalduinen

Credits to Klaas Sikkel

Today's topic

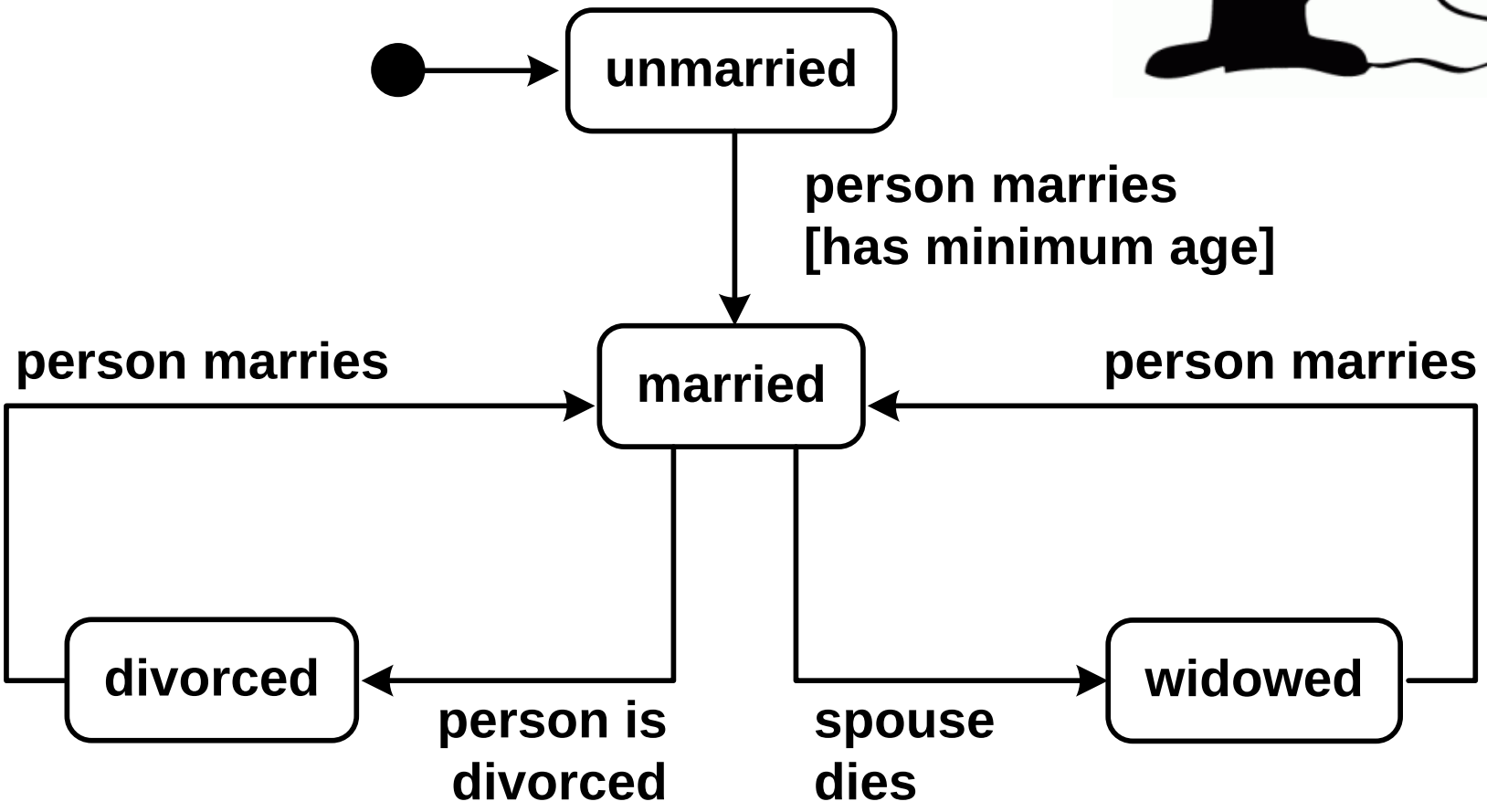


Contents

1. State Machine basics
2. From Use Cases to State Machines
3. Extension 1: composite states
4. Extension 2: parallelism
5. Comparison AD – SM
6. Exercise

Winding up UML

Example of a state machine



SM basics

- An object (in the system) can switch between different states.
  - This object typically is the representation inside the system of a real object in the outside world
- A change from one state to another state is called a transition
- We draw a state machine for a class (as a blueprint). All objects of this class have their own individual state
- Formally called “State Machine Diagram”, sometimes abbreviated to “State Machine”

SM basics

Example: Marital status

- A (real) person can be unmarried, married, divorced, or widowed

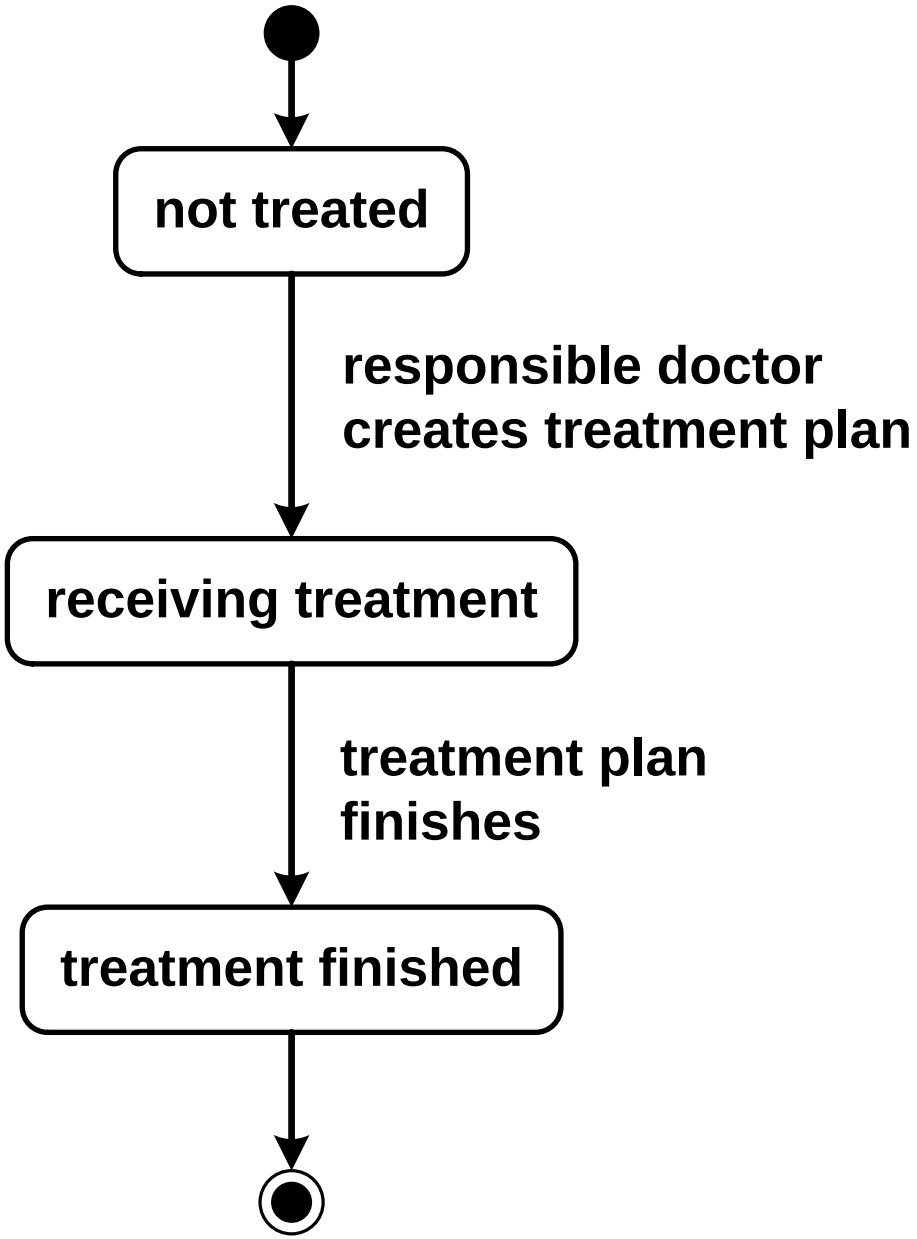
- In the government's Register of Residents the person is represented by an object which has marital status as one of its attributes

(Side note)

- If the administration is perfect,
  there is 100 % correspondence between
  - the states of entities in the real world
  - the states of objects in the system that represent these entities in de real world

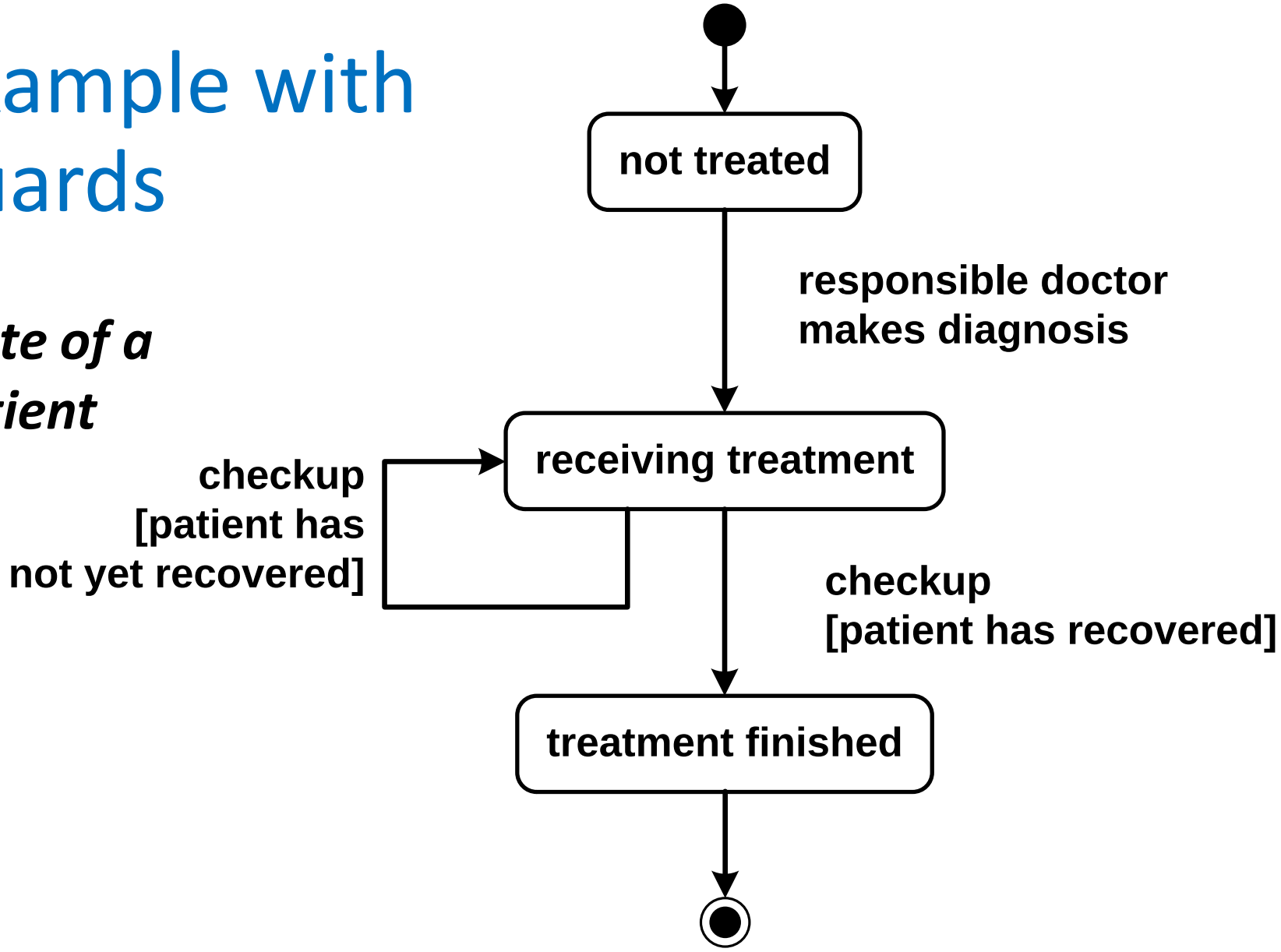
Example

State of a patient



Example with guards

State of a patient



Guards

- A guard imposes a specific condition.
  A transition only can take place if the guard is true.
  - We write guards between brackets ("[…]")

- If there are guards on transitions leaving a state, make sure that there is always a possibility to leave this state

Elements of a SM (1)

- A SM denotes possible states and transitions between states for an object of a particular class.
- A *state* is labelled with a name describing the meaning of this state.
- A *transition* is triggered by an event which causes the object to move from one state to another.
  - A transition can be constrained by a *guard*.
  - A transition is instantaneous, an object is always in some state, never in between states.

* An event is instantaneous (executing a use case takes some amount of time) Transitions are also instantaneous: An object cannot be “in between” states. At some moment during the use case execution the transition takes place)

Elements of a SM(2)

- An SM always has one *start symbol* (●)
- An SM may have a *stop symbol* (◉)

More precisely: ● and ◉ are **pseudo-states**,
preceding / following the real initial / final state .

- "not treated" is the initial state,
- "treatment finished" is the final state

Why has it been defined this way? The final state is usually a permanent state, while the object remains in the system.

(This is different in Activity Diagrams, where the final activity is always completed)

Elements of a SM

- Official syntax for the label of a transition:
  <event> [“[” <guard>“]”] [“/” <action>]

2. From use cases to state machines

- Step 1: select relevant events (use cases/user stories)
  - Only those events that concern the state of the object
- Step 2: compile a state transition list
  - For each transition: state from, state to, guard
- Step 3: construct State Machine
  - Iterate until the SM is complete
- Step 4: validation
  - No unreachable states? Every state can be left? Proper final state?

Example

In the Register of Residents essential data about resident persons are stored.

Concerning marital status we distinguish the states

- unmaried
- married
- divorced
- widowed

Make a state machine for a person

Step 1: Relevant use cases

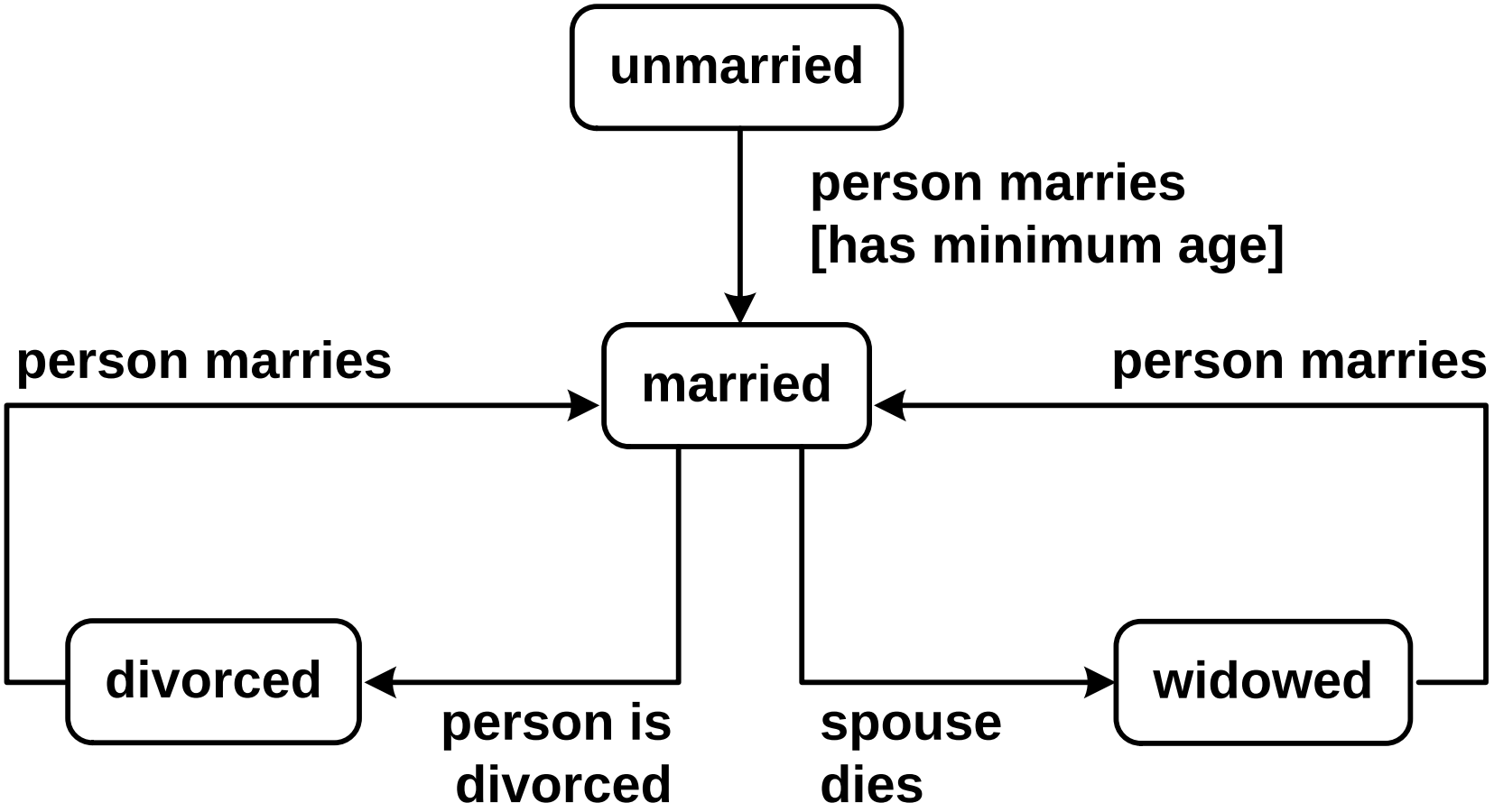
Actor	Use case
Registrar	Register marriage
Registrar	Register divorce
Registrar	Register death

N.B.: Use cases related to moving address or getting a new passport are not relevant

Step 2: State Transition List

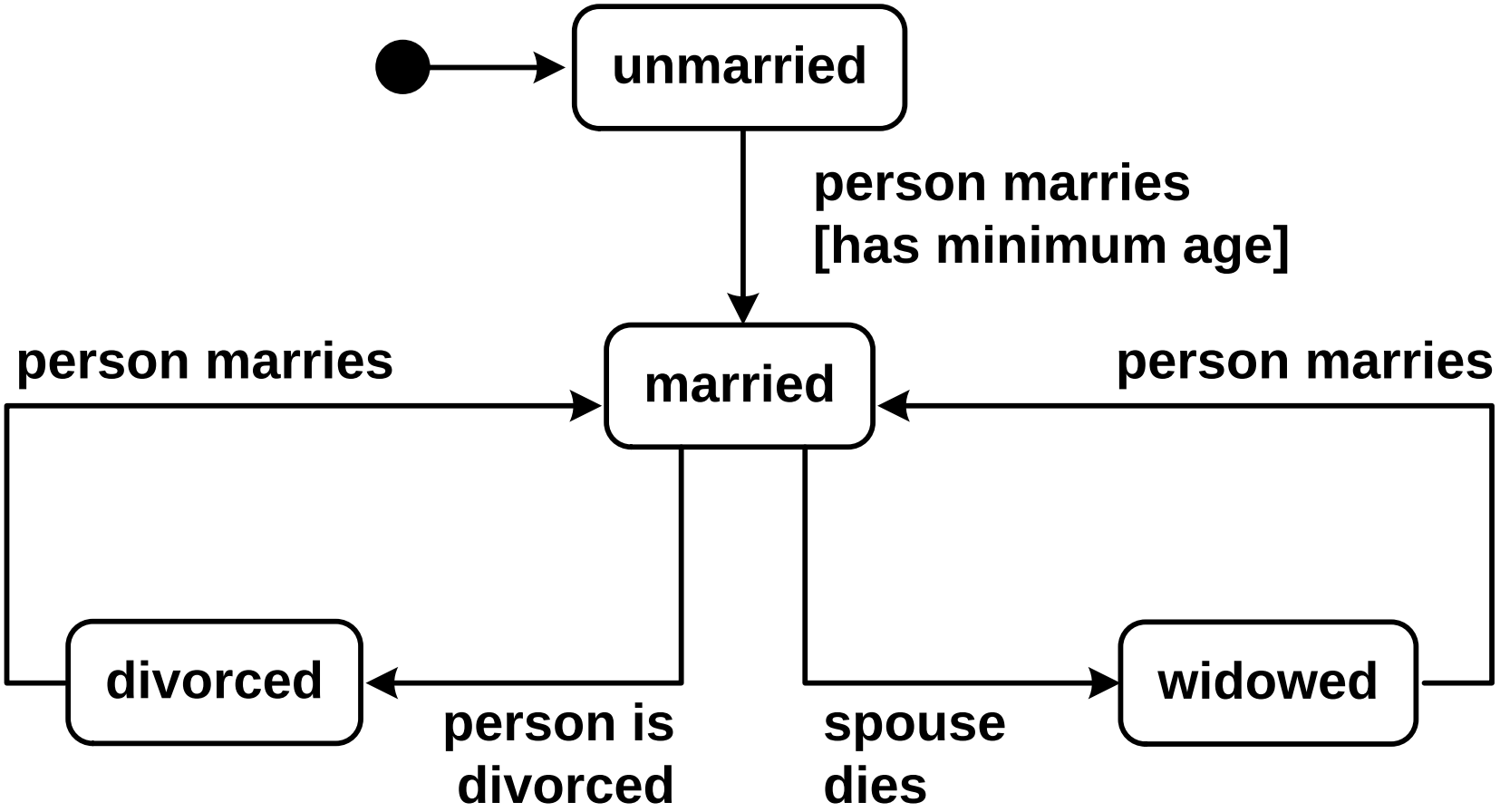
Event	Guard	from State	to State
Person marries	Minimum age	unmarried	married
Person marries		divorced	married
Person marries		widowed	married
Person divorces		married	divorced
Spouse dies		married	widowed

Step 3: state machine



Is this state machine complete?

Final version (no unreachable states etc.)



2. From use cases to state machines – Afterword

In principle there are two options

- First identify the transitions (events, use cases)
- then define appropriate states
- then check for forgotten transitions

Usually works better

or

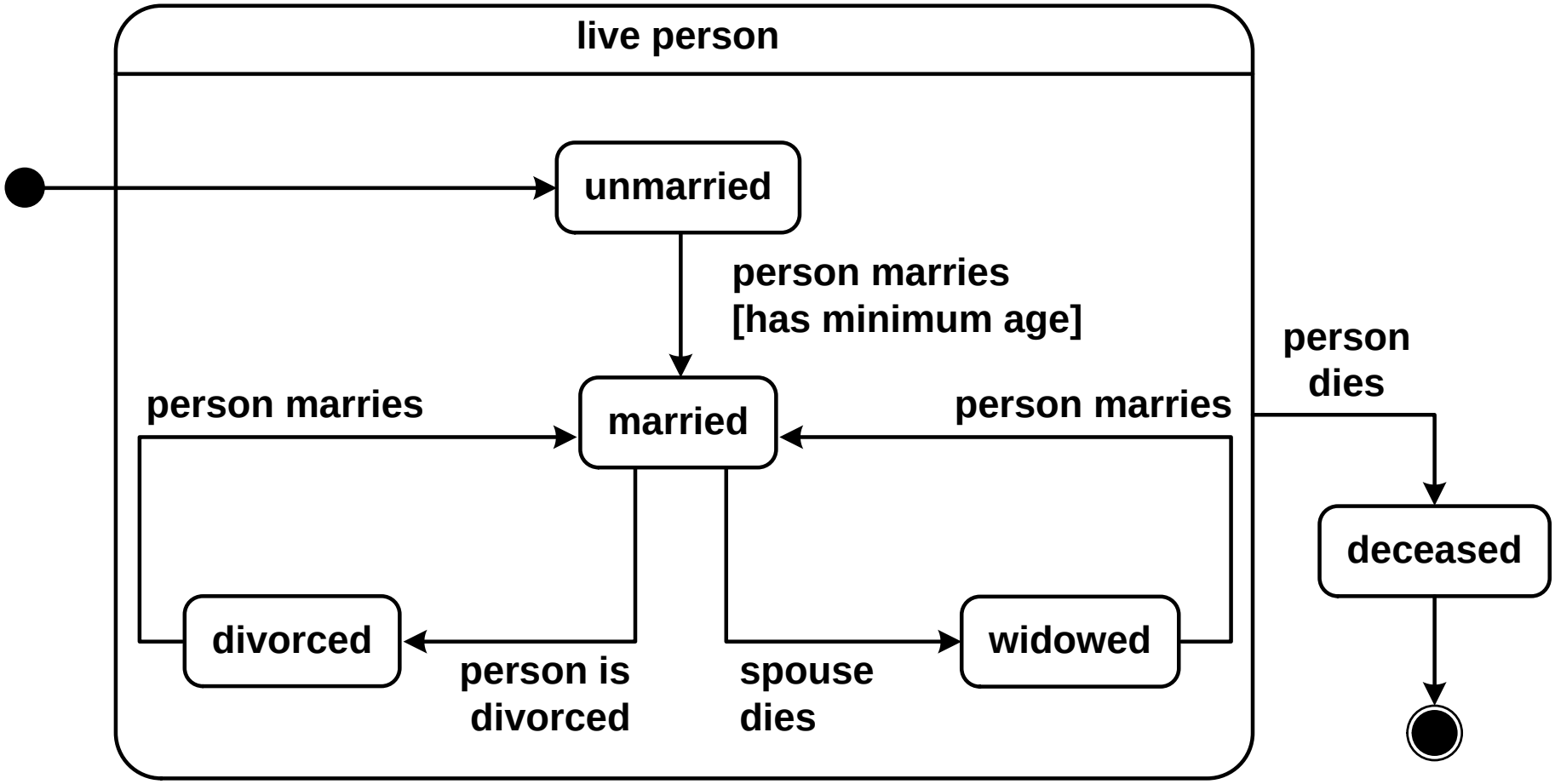
- First identify the states,
- then define the required transitions
-

You need to know the Process in order to Identify the states

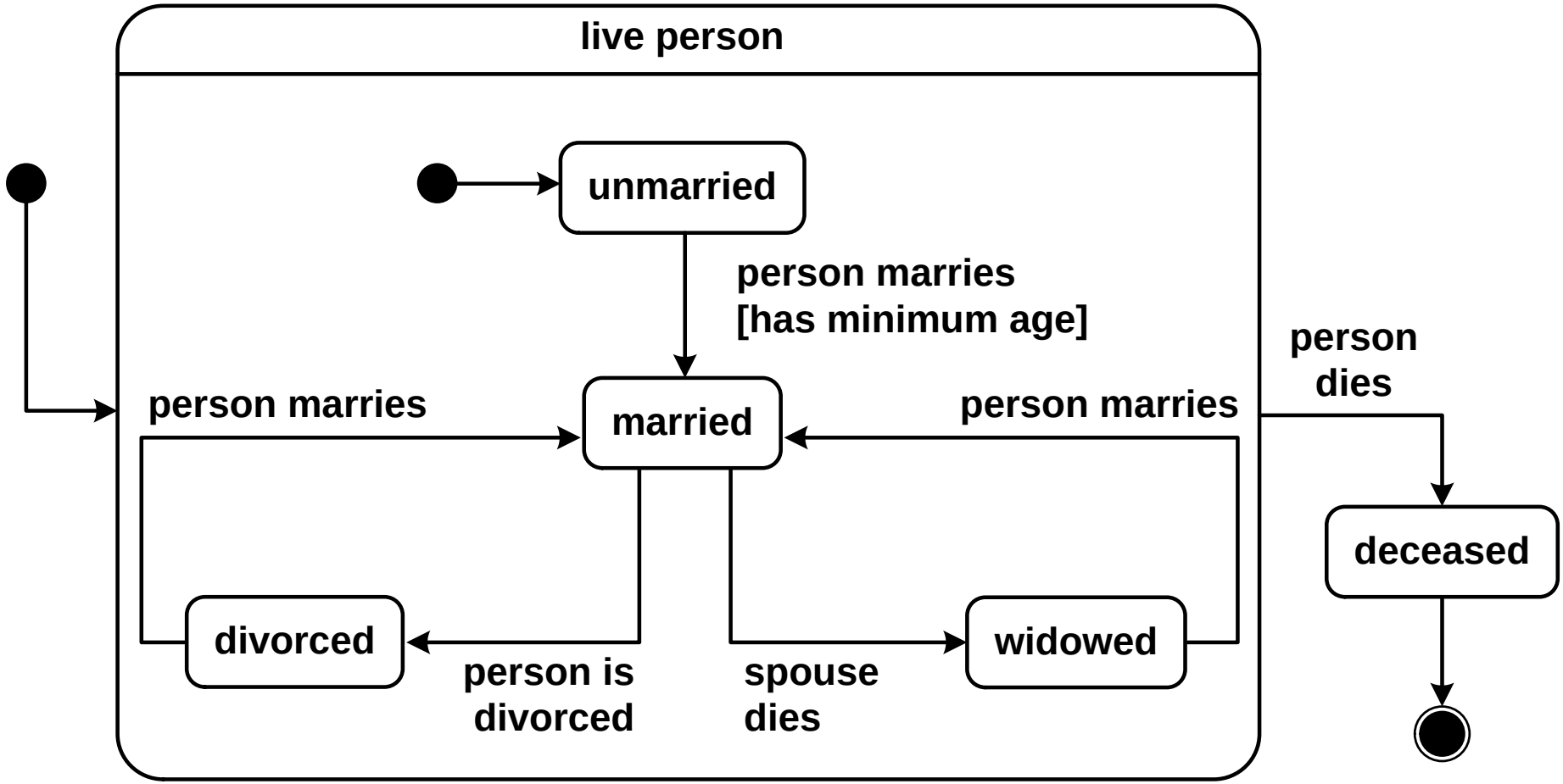
3. Composite states

- A composite state consists of a set of substates (with transitions between them)
- *What can we use this for?*
  - To abstract from substates if the diagram becomes complex
  - A single transition suffices to leave the composite state

Example


live person
unmarried
person marries
[has minimum age]
person marries
married
person marries
person
dies
divorced
person is
divorced
spouse
dies
widowed
deceased

Equivalent notation



Elements of a SM (3)

- Composite states can have a name (not mandatory)
- Entry transition:
  - Usually to a specific substate within the composite state *(alternatively: include a start symbol in the composite state)*
- Exit transition:
  - From the composite state as a whole (irrespective of the current substate)
  - Or from a specific substate*

  * *An example will follow for composite state with parallellism*

4. Parallelism

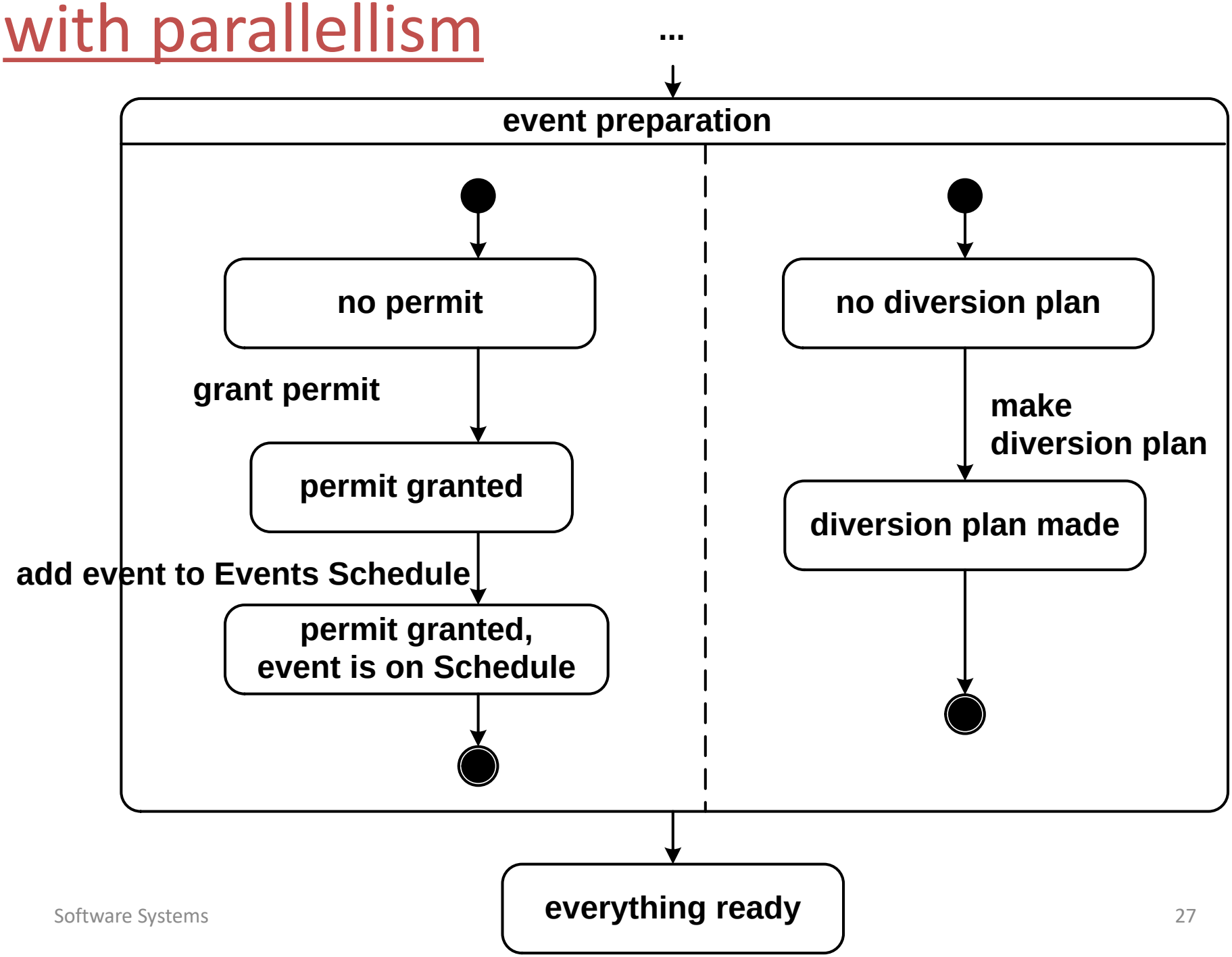
- Different aspects of an object can relate to different states

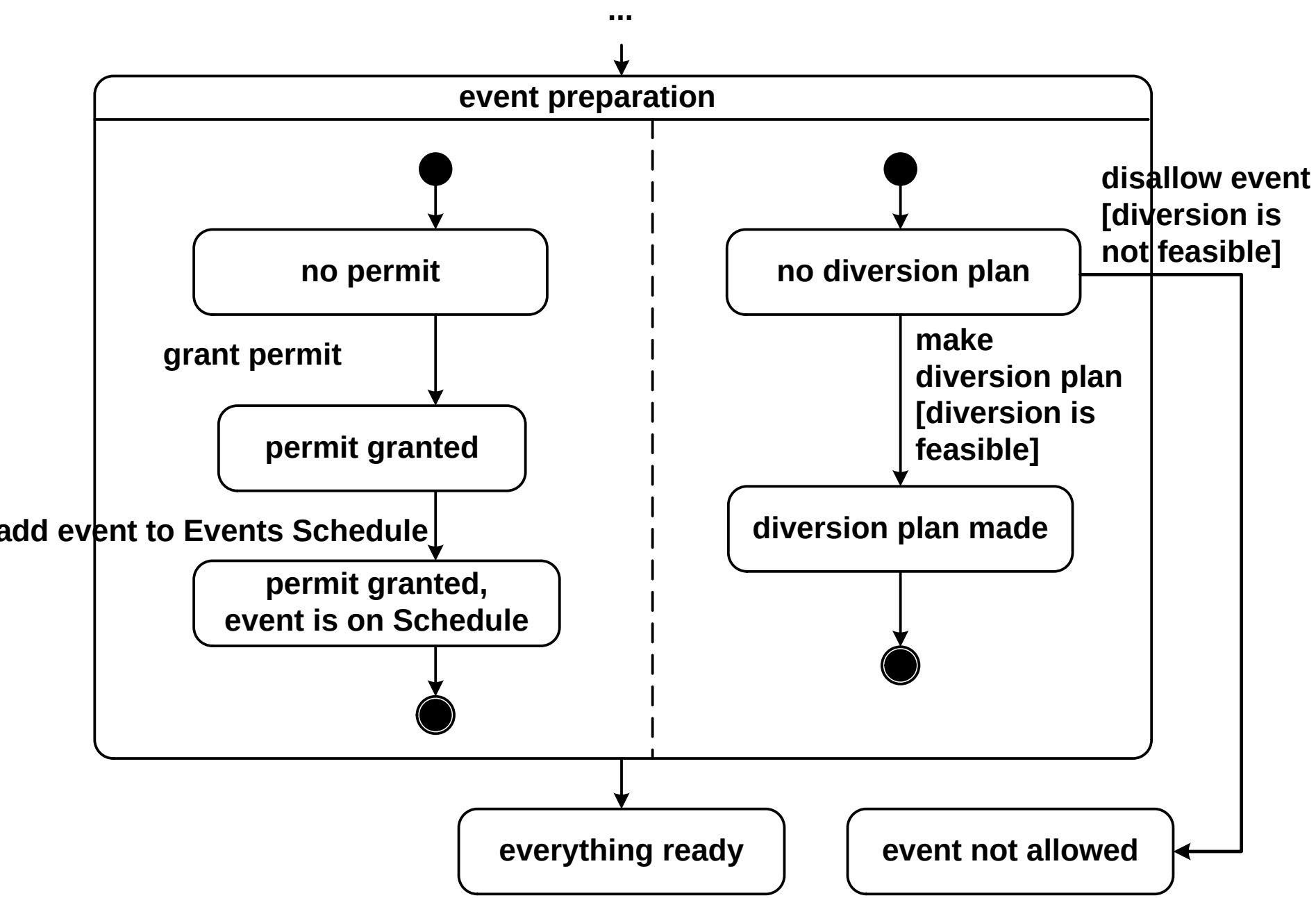
- Introduced by the following example ./.

Example: Event planning for a city

- When a request for an event has been approved, the following steps need to be taken
  - The City Council
    - grants a permit for the event
    - adds the event to the Events Schedule
  - The traffic department
    - makes a diversion plan

with parallellism




...
event preparation
no permit
grant permit
permit granted
add event to Events Schedule
permit granted,
event is on Schedule
no diversion plan
disallow event
[diversion is
not feasible]
make
diversion plan
[diversion is
feasible]
diversion plan made
everything ready
event not allowed

Elements of a SM (4)

- Composite state with parallelism
  - Every parallel part has a start and stop symbol
- Entry transition: always to the composite state as a whole
- Exit transition:
  - Normal case: From the composite state (precondition: all parallel parts have finished)
  - Abnormal case (e.g. when the process is aborted): From within one specific substate

Elements of a SM (5)

- “Normal” transitions are labelled with an event, possibly in combination with a guard
- Transition from the start symbol and to the stop symbol are not labelled

- For a normal exit transition from a composite state with parallellism a label is allowed but not mandatory

5. Activity diagram versus State Machine

AD models are steps in a process (activities)

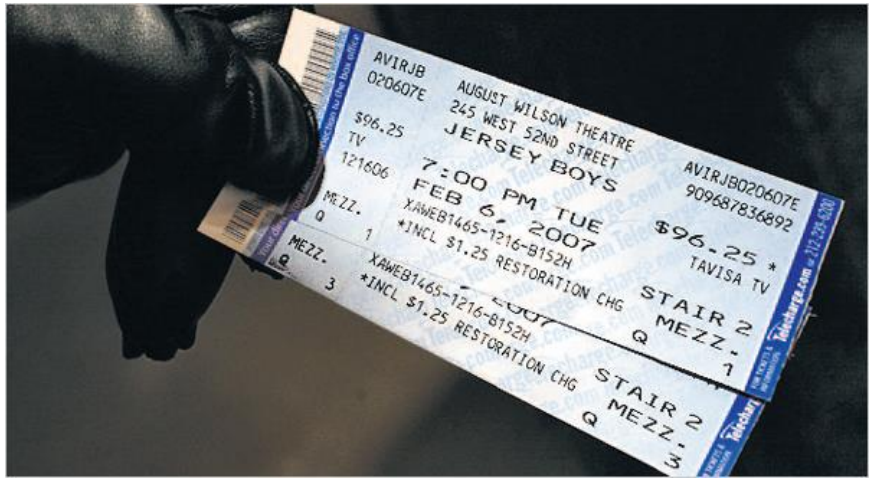
- in the real world

SM models states of an object (caused by events)

- inside the system

6. Exercise

- Consider the system for theatre tickets (Section 1.3.4, UCD D1-10, D1-13)



Make a state machine diagram for the state of a seat: a place (seat) for a particular performance

State machine diagram for a theatre ticket