State Machine Diagram

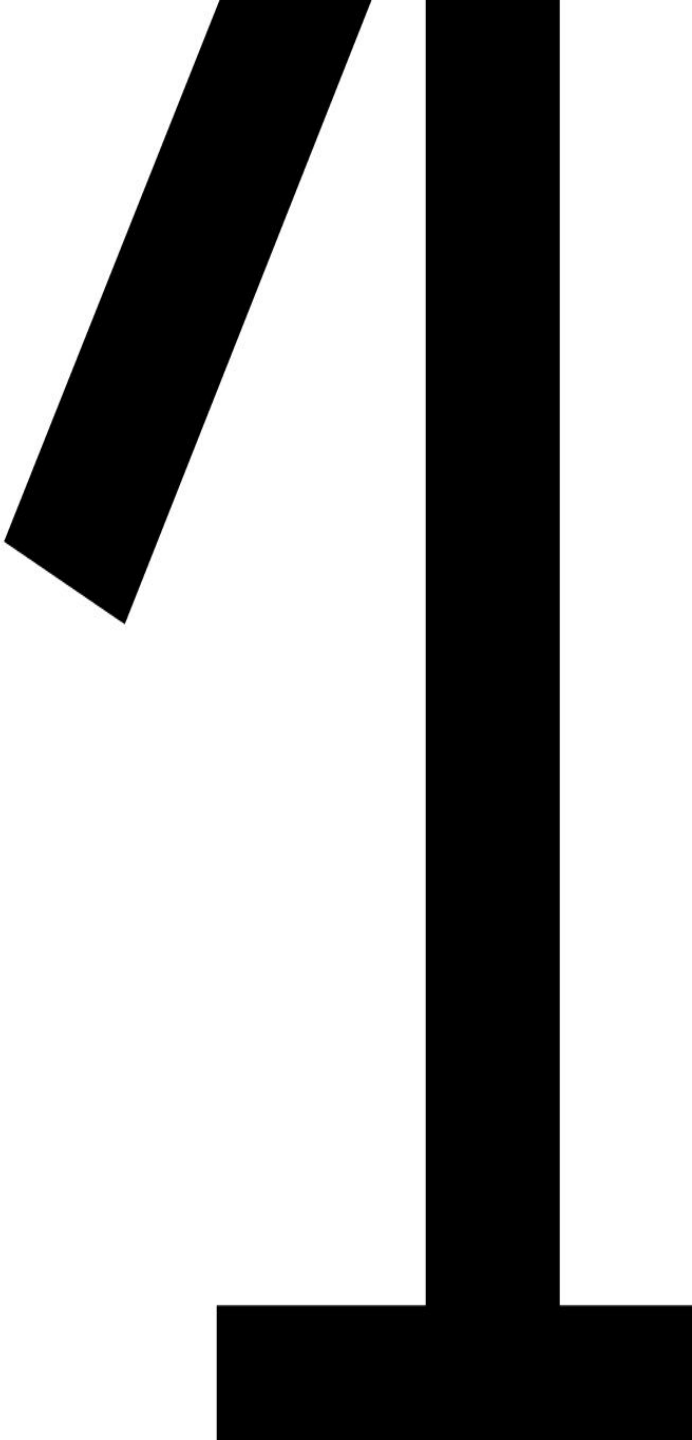
Programming 4

KdG Karel de Grote Hogeschool

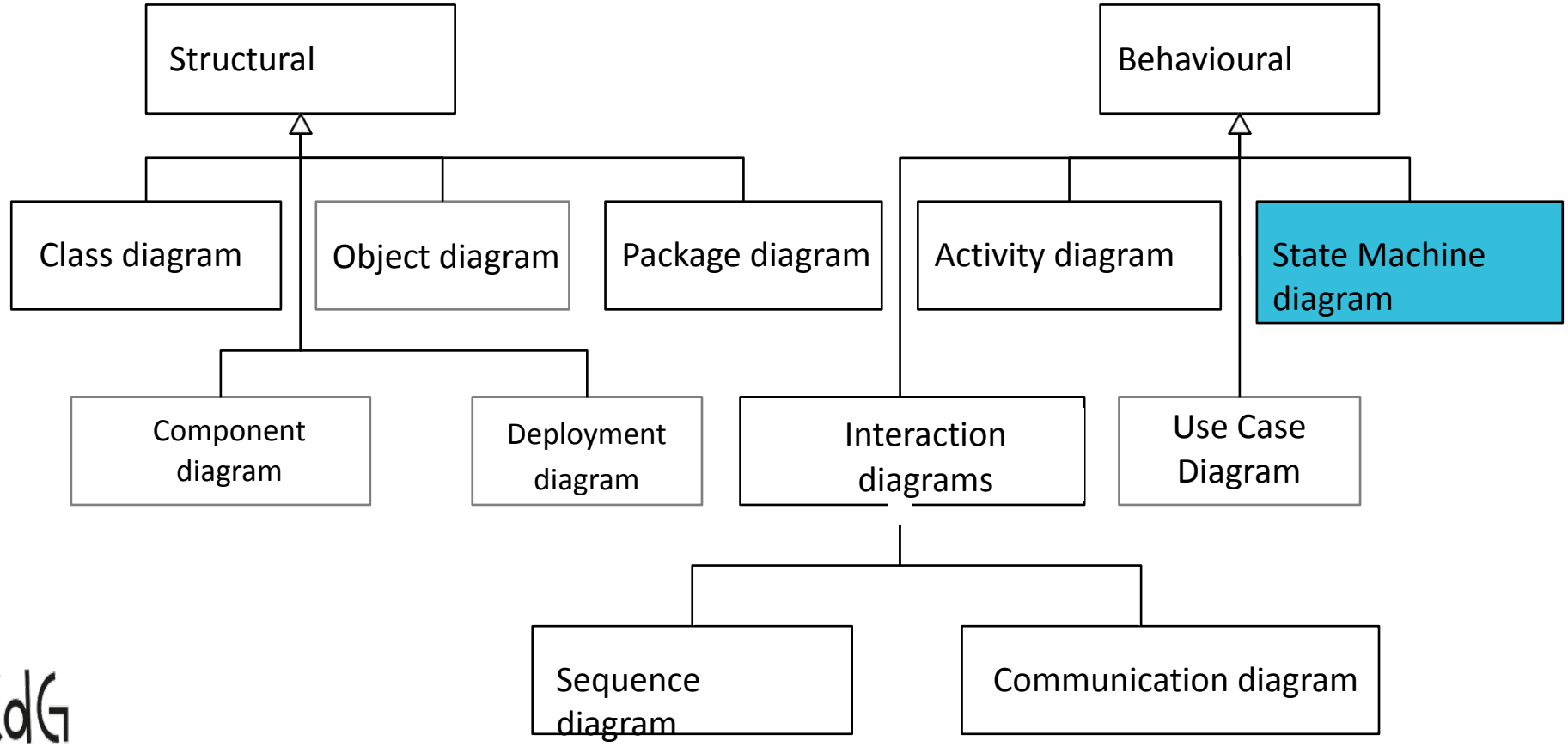
Agenda

1. Introduction
2. Transition
3. State
4. Superstate and Substate

Introduction



UML diagrams



State and behaviour

- An object has state and behaviour
  - state: instance values
  - behaviour: methods (defined in class)
- In a state-dependent object the behaviour changes depending on the state
  - When the value of an attribute changes, the methods will behave differently
  - State machine diagrams model state transitions in state dependent objects

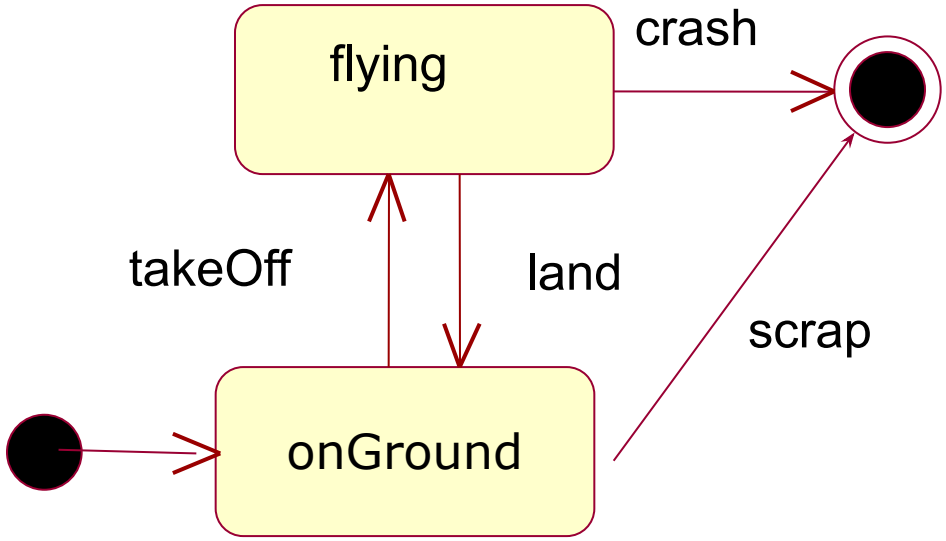
State and behaviour

State Diagram

useful for objects with a lot of states (which moves are possible in which states)

- Example: airplane
  - Can only land in state "flying"
  - break() works differently in state onGround (wheel breaks) and in state flying (spoilers)

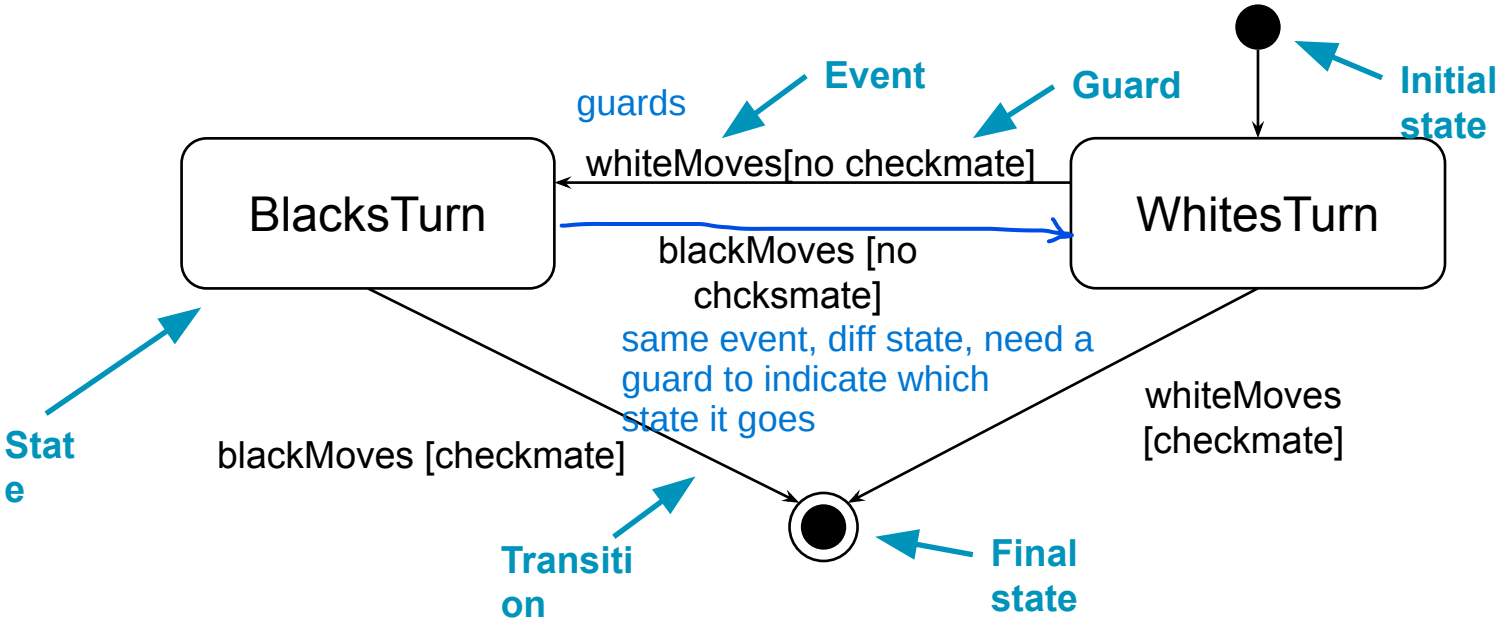
opposite of an activity diagram (where actions are in the boxes)



What is a State Machine Diagram?

- *Describes states an object goes through and the events triggering state transitions.*
- *Shows lifecycle of objects*
  - *The lifecycle of objects can span multiple user stories*

State Machine diagram example: chess



Chess code

java diagram using a java ENUM

```
public enum Chess {
 START,
 WHITES_TURN,
 BLACKS_TURN,
 END;

 public Chess whiteMoves(boolean checkMate) {
  if (this == START ||this == WHITES_TURN)
   return checkMate ? END : BLACKS_TURN;
  throw new IllegalStateException("whiteMoves not allowed in
state " + this);
 }

 public Chess blackMoves(boolean checkMate) {
  if (this == BLACKS_TURN)
   return checkMate ? END : WHITES_TURN;
  throw new IllegalStateException("blackMoves not allowed in
state " + this);
 }
}
```


https://gitlab.com/kdg-ti/acs-programming-2.2/chess-state

Chess code

```
public static void main(String[] args) {
 Chess game = Chess.START; System.out.println(game);
 game = game.whiteMoves(false);System.out.println(game);
 game = game.blackMoves(false);System.out.println(game);
 game = game.whiteMoves(true);System.out.println(game);
 game = game.blackMoves(false);
}
```

```
WHITES_TURN
BLACKS_TURN
WHITES_TURN
END
Exception in thread "main" java.lang.IllegalStateException:
blackMoves not allowed in state END
```

Notation

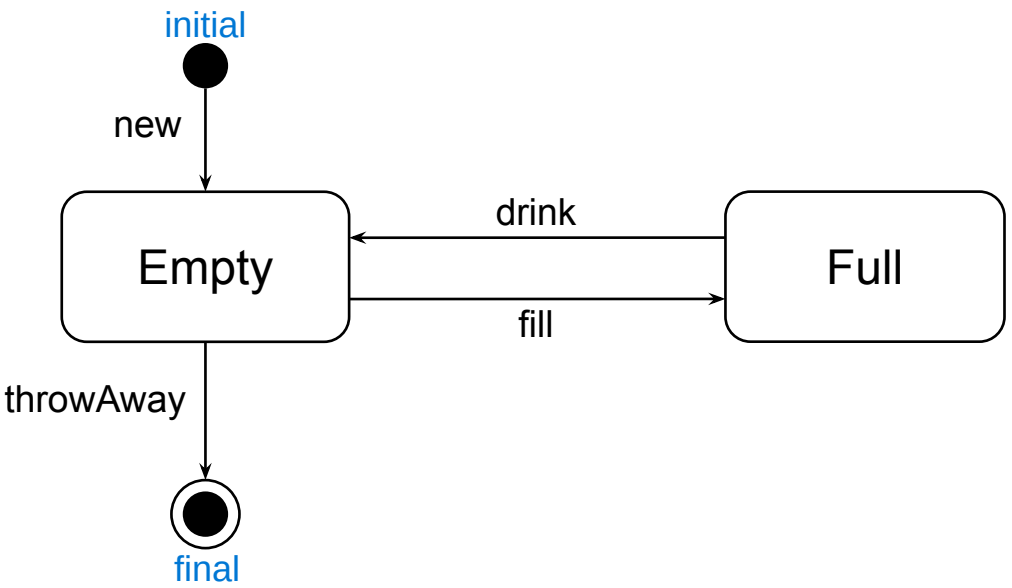
- **State:** "*Stable condition in which an object is a while.*"
  - Determined by attributes
  - Result of events
  - Notation: rounded rectangle

- **Transition:**
  - Possible change of one state to another
  - Notation: solid line with open arrowhead

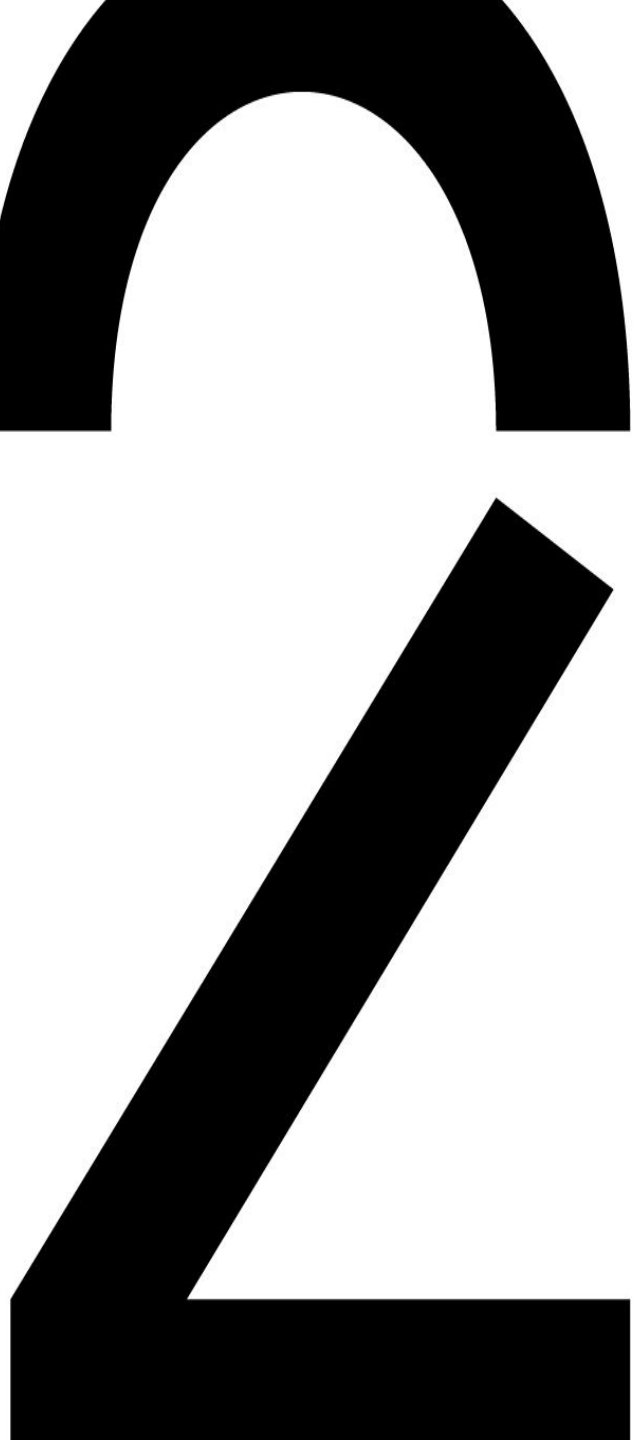
Notation

- **Initial state:**
  - State of the object when it is created
  - Notation: bullet
- **Final state:**
  - end state of the diagram
  - Notation: bull's eye

- Example: cup



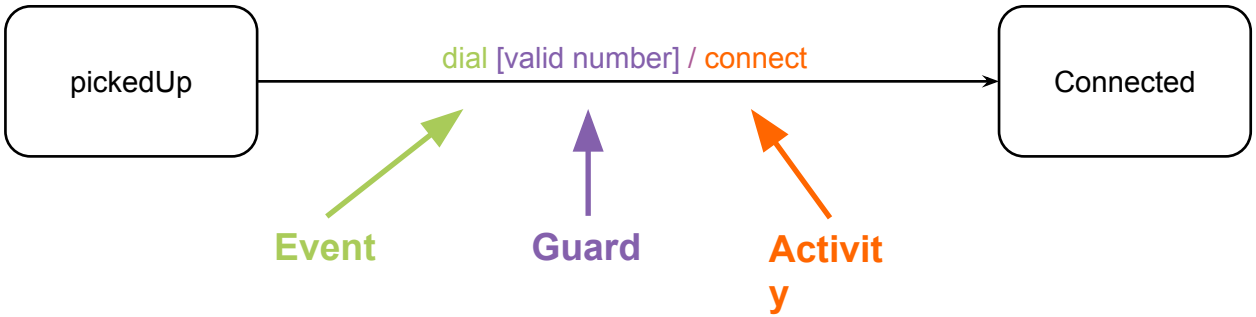
Transition



KdG
Karel de Grote
Hogeschool

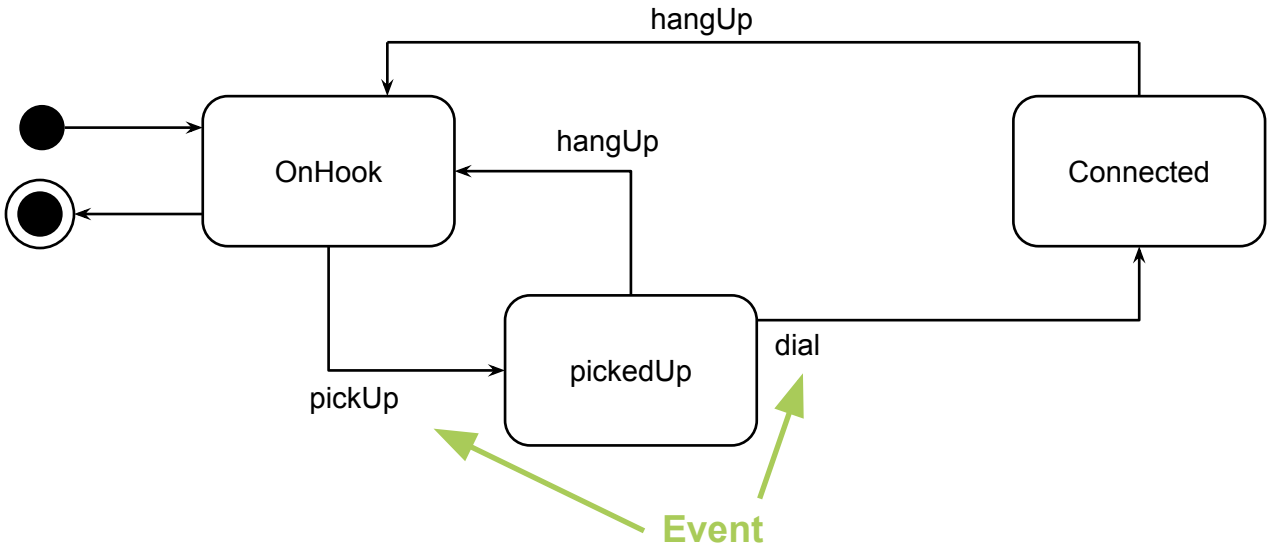
Transition syntax

- A transition can consist of:
  - **Event** (trigger)
  - **Guard** (condition)
  - **Activity**
- All these elements are optional
- **multiple** transitions can start or end in a state



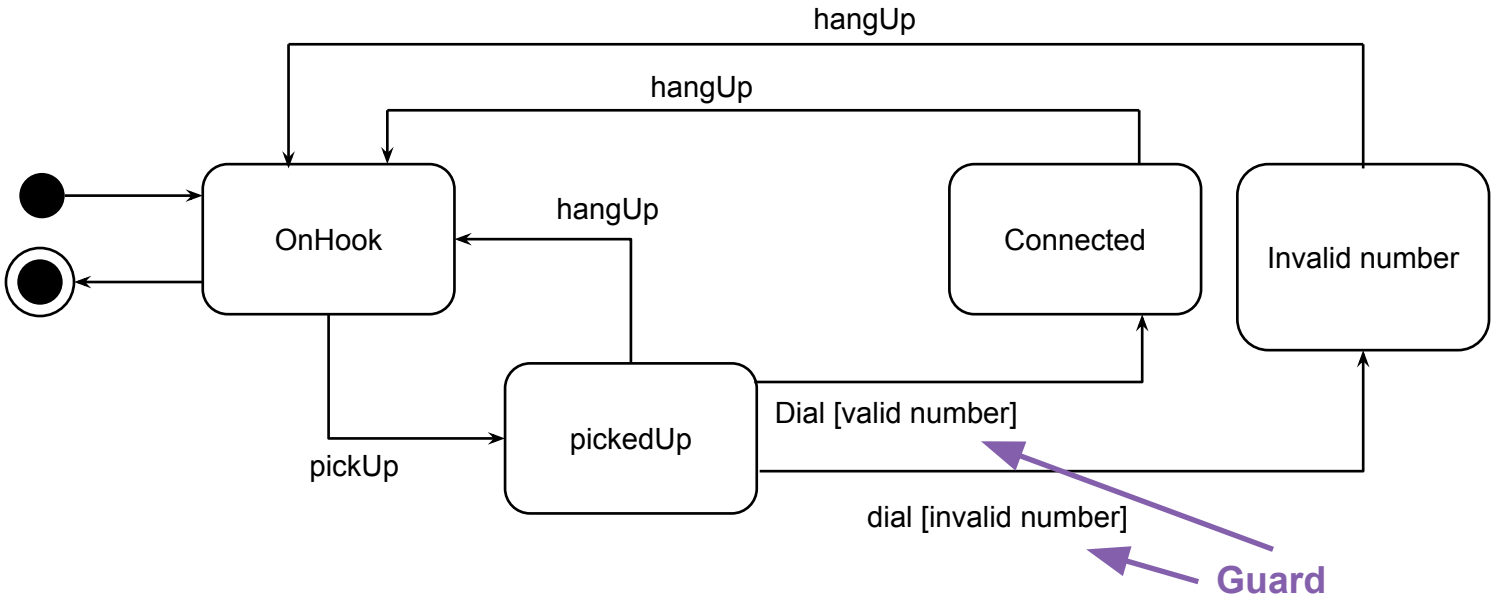
Event

- A trigger that initiates a state transition
  - notation: plain text on a transition is an event
  - Can be a method that is called on an object
- Event types:
  - **External event:** Enters the system
  - **Internal event:** From within the system
  - **Time event:** Generated at a specific time or after an interval



Guard

- Condition to do a transition
- Mandatory if two transitions from a state are triggered by the same event
  - Notation: logical expression between "[" and "]"

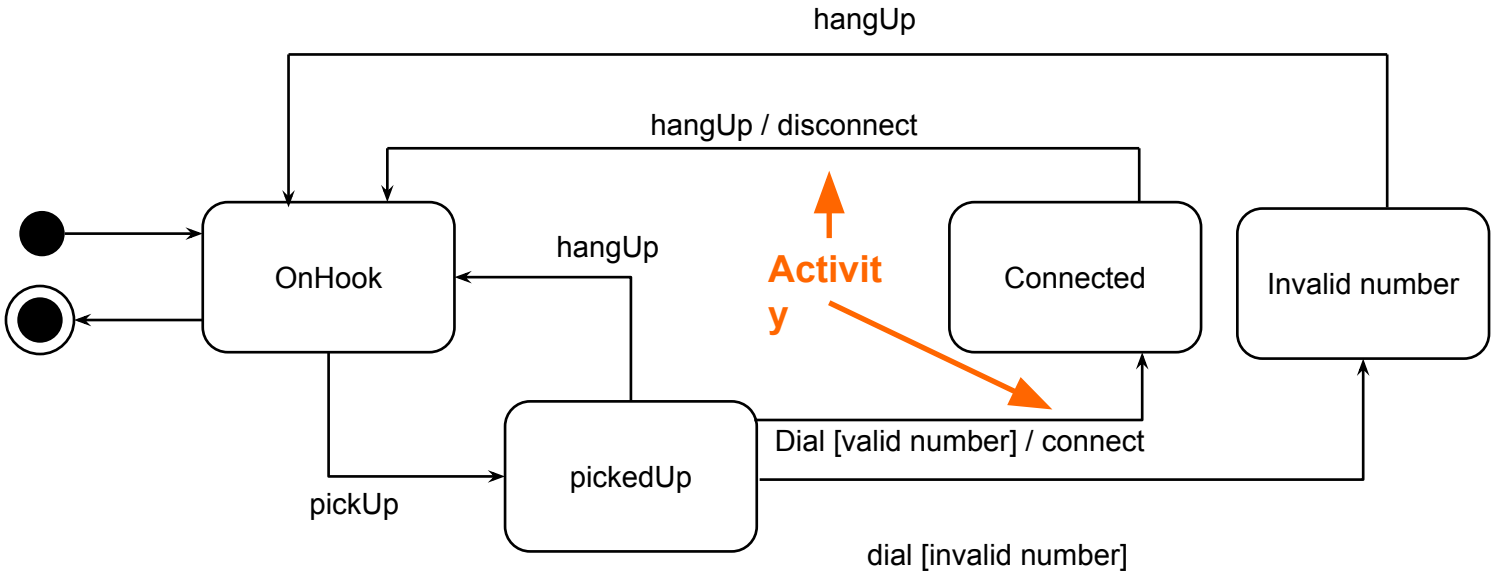


Activity

- Actions that are executed during a transition

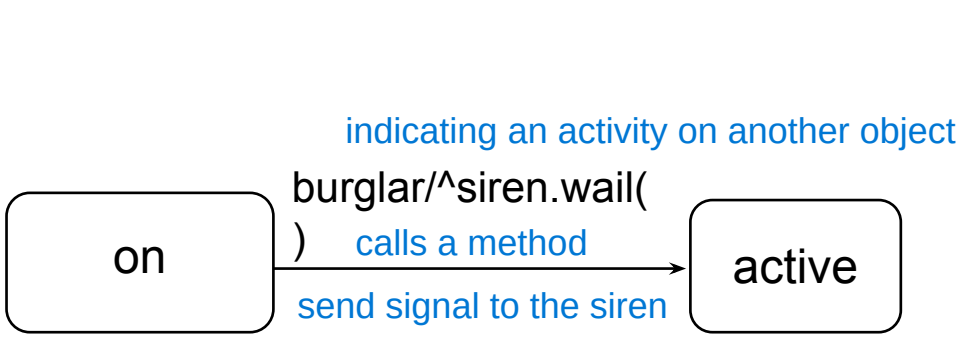
Atomic: Can not be interrupted

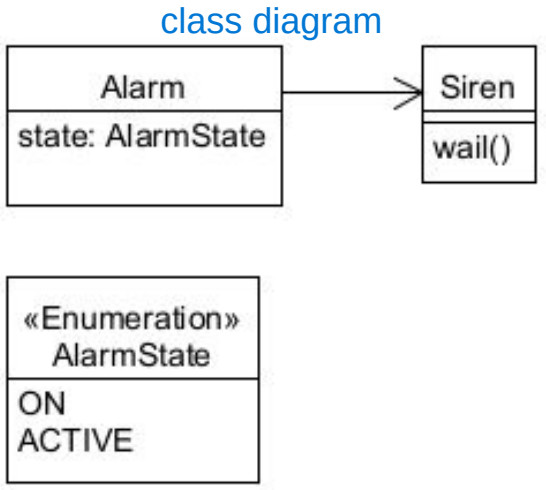
- Notation: preceded by "/"



Activity

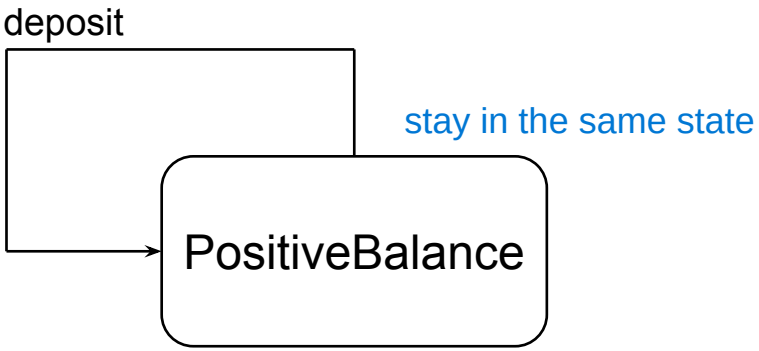
- When modelling an object an activity can be used to stress an important action during the execution of a method
- A message (event) **sent** during a transition, is indicated by prepending the ^-symbol to an activity
- in the diagram below a wail message is sent to the siren object





Reflexive transition

- An event can transition to the same state again



St ate

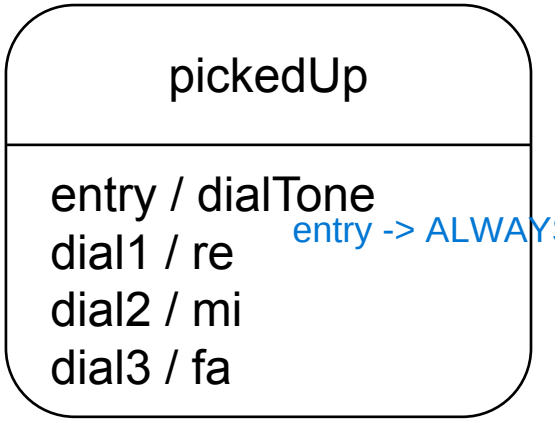
State Syntax

- A state has at least a name
- **internal transitions** can be used for events that do not change state
  - In compartment below the name
  - A line for each internal activities:

    event [guard] / activity

in a state you can add detail
same state as in a transition (internal transition)

only one event -> put it on the event



entry -> ALWAYS dial

what happens inside the state

Event

- Standard internal events:

 entry

For activities that are executed when entering the state

 do

stay in an event -> as long as there is no state to make it end

For activities that are executed while this state is active

 exit

For activities that are executed when leaving the state

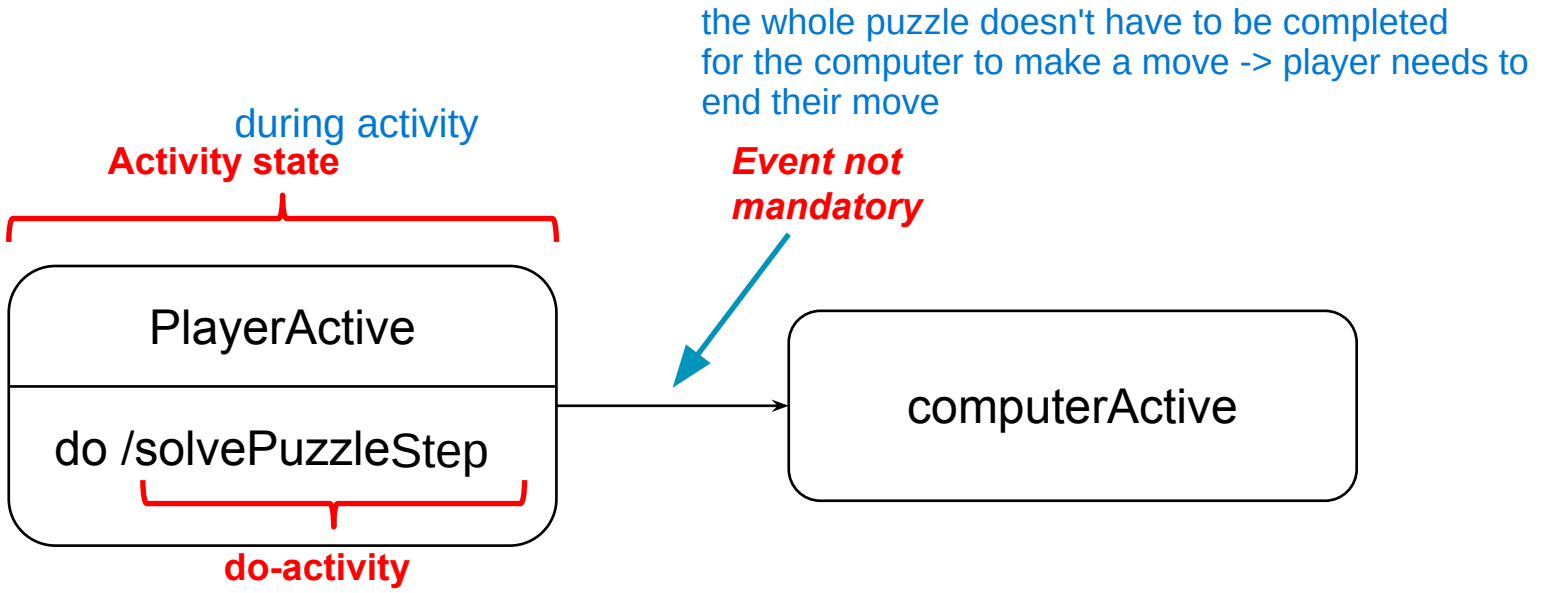
if condition fullfilled - > exit state

Do activity

CAN be interrupted by an external transition

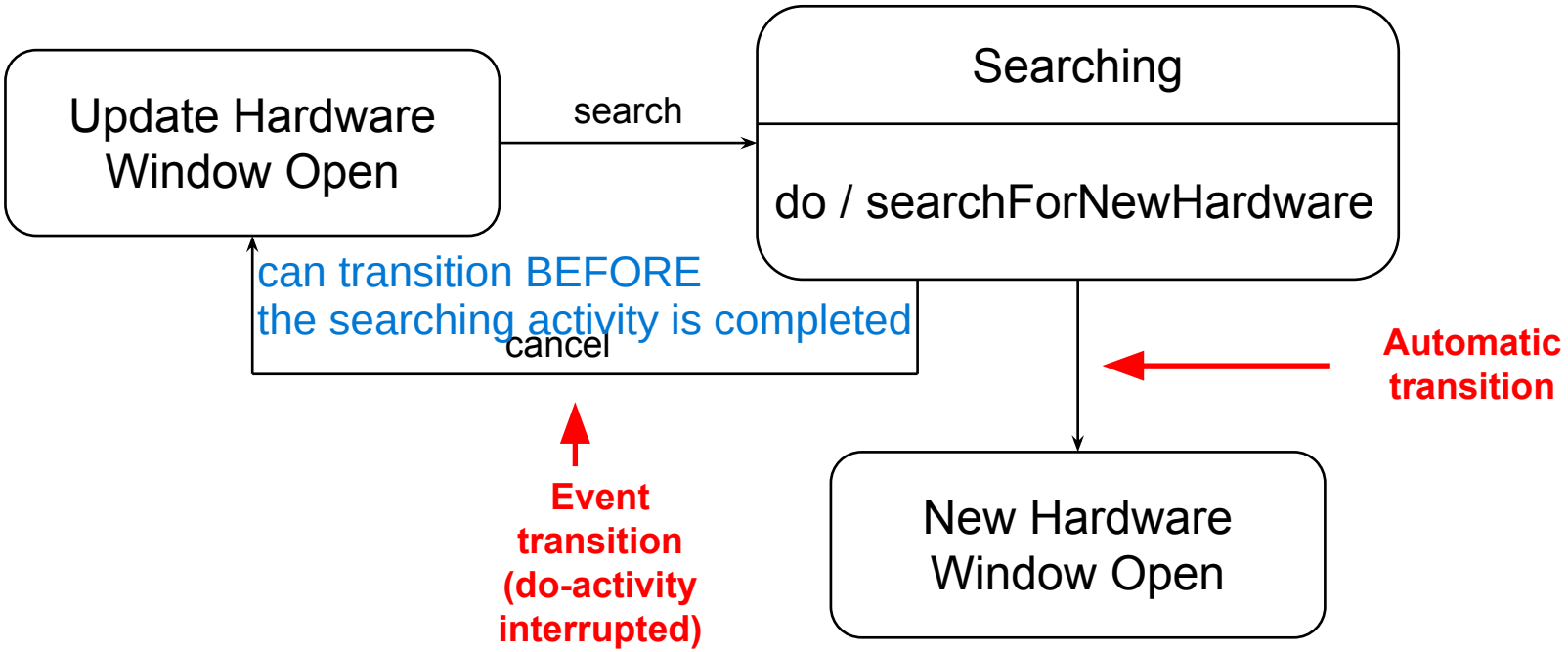
- **Automatic transition:**
  - When an activity associated with a do event is finished the state ends
    - An event is not madatory for such a transition
  - A state with a do activity is called an **activity state**

activity and state diagram CANNOT be interupted



Do activity

- Can be interrupted (in contrast with a transition activity)



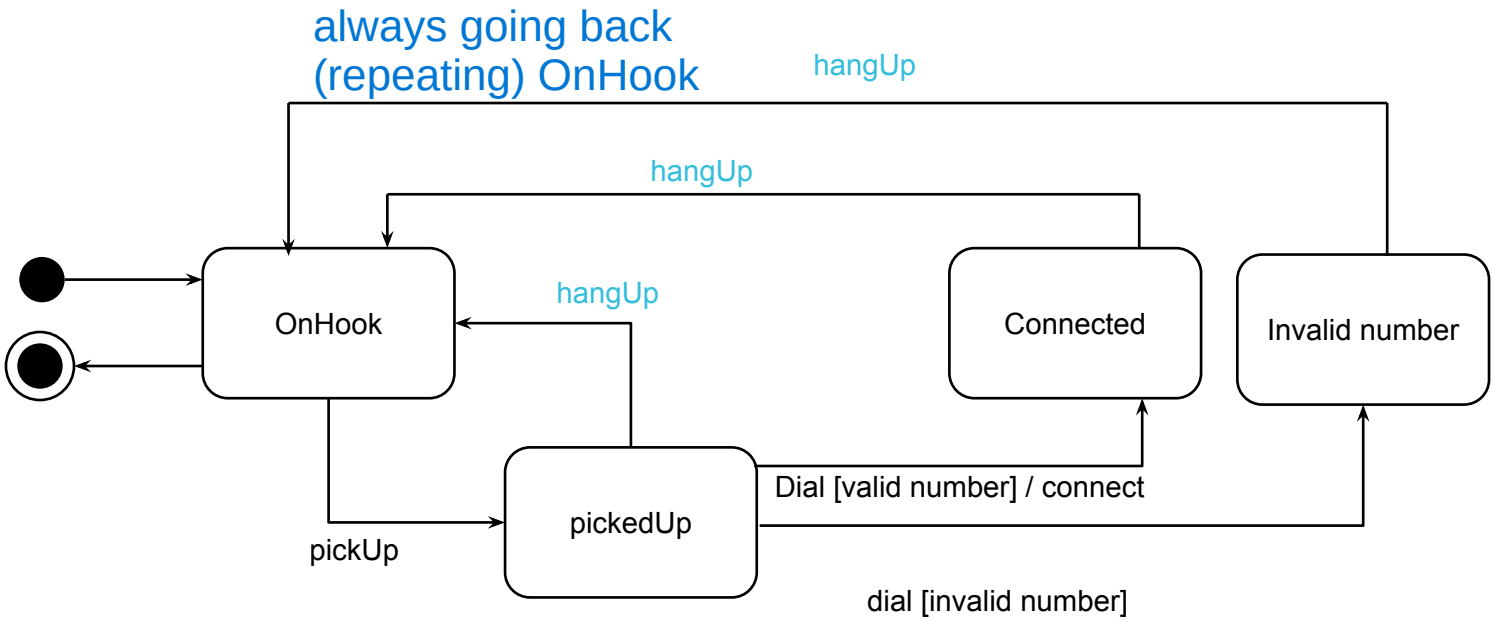
Superstate and Substate

4

KdG
Karel de Grote
Hogeschool

Problem: duplicate transitions

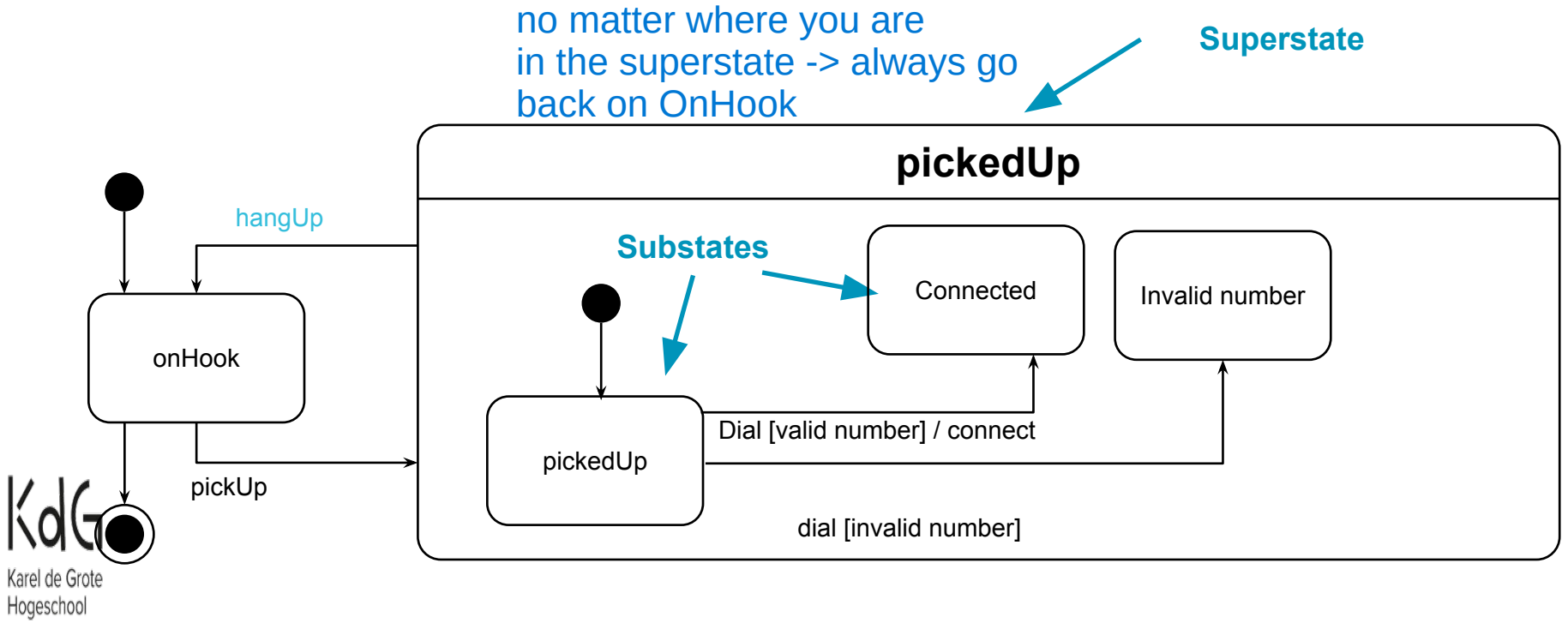
- States in a diagram can have shared transitions, complicating the diagram



Solution: Superstate and Substate

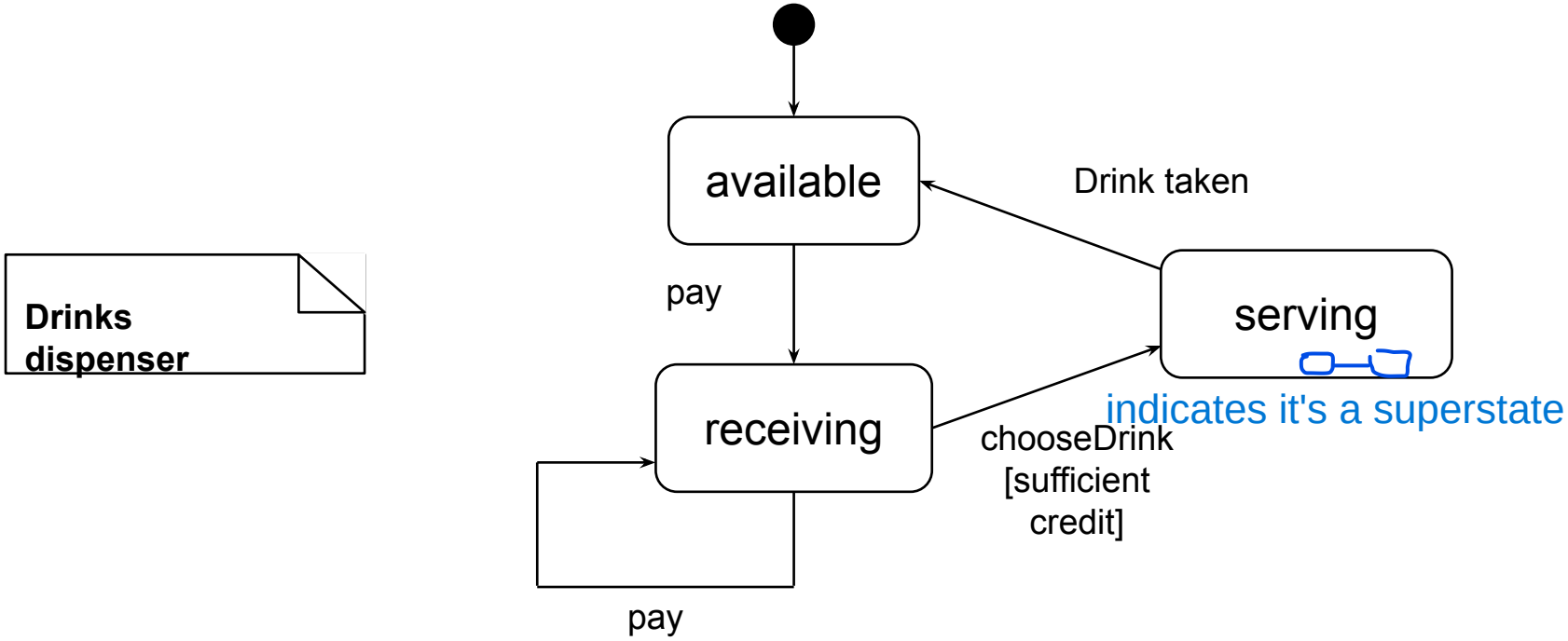
Solution: move to a superstate

- **Superstate:** contains nested states
- **Substate:** state within state
  - The initial substate indicates where you enter the superstate
  - You can also put the event that exits the superstate on a transition to a final substate

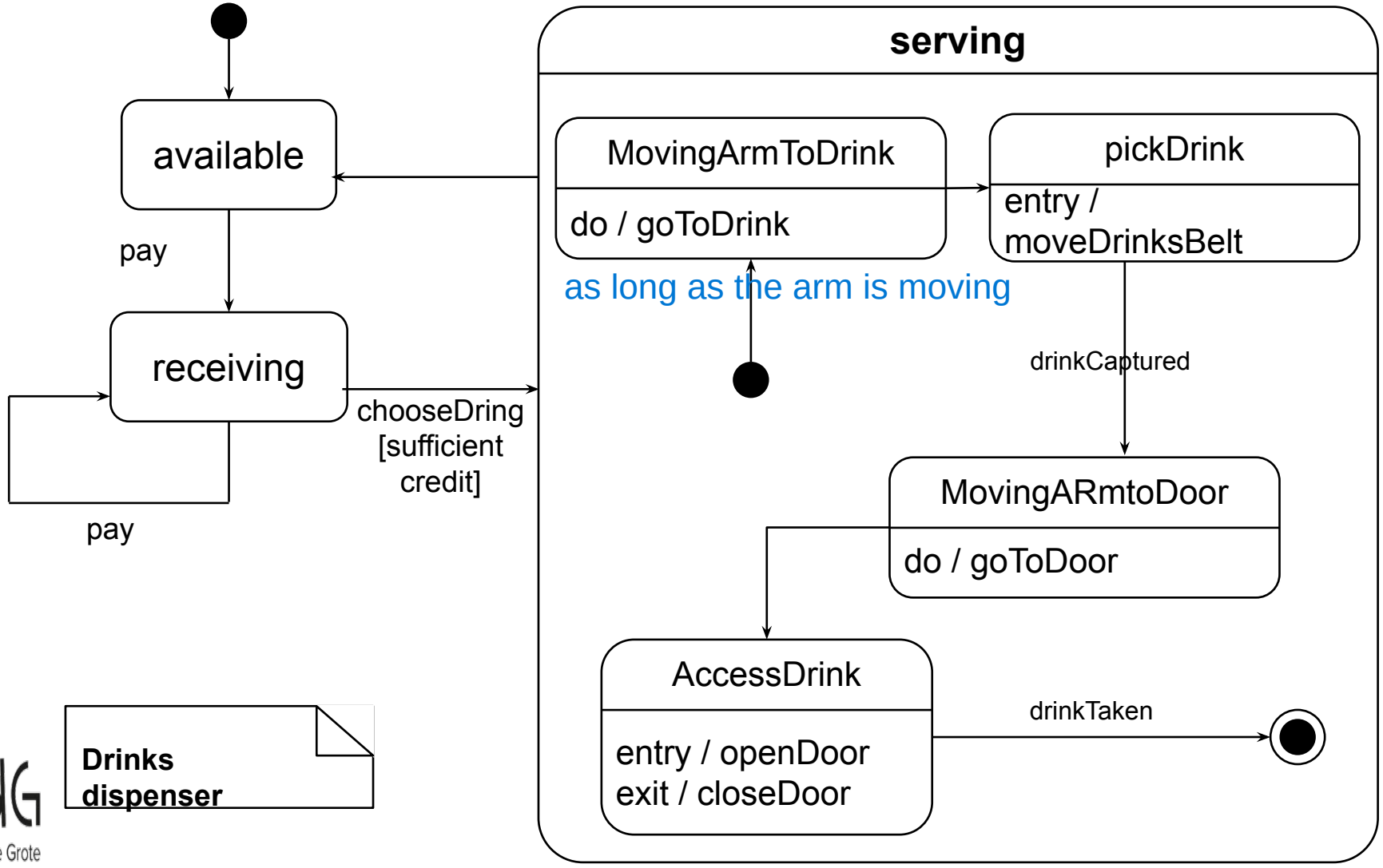


Example: drinks dispenser

- Elaboration serving as a superstate
  - You can use this technique to split complex diagrams



Voorbeeld drankautomaat



Summary

1. Introduction
2. Transition
3. State
4. Superstate and Substate