

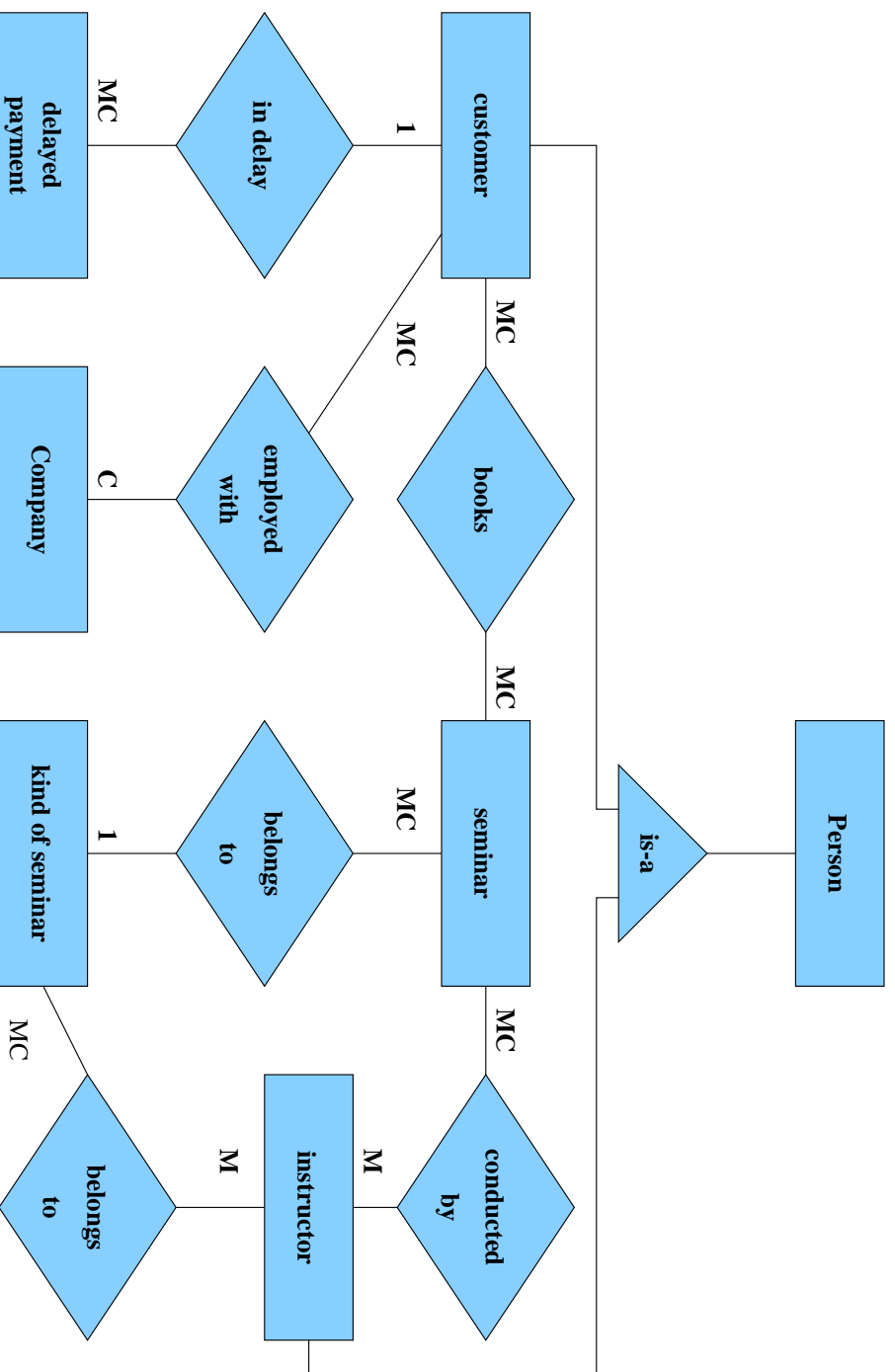
3.2.11 Entity-Relationship Model

- *information modeling* (*Gegenstands-Beziehungs-Modell*)
- graphical description of persistent **data** and **relations** amongst it
- **entity**: individual object with attributes
further specification e.g. in DD
- **entity set**: set of entities with the same attributes
- **key**: minimal identifying attribute set

ER-Diagram

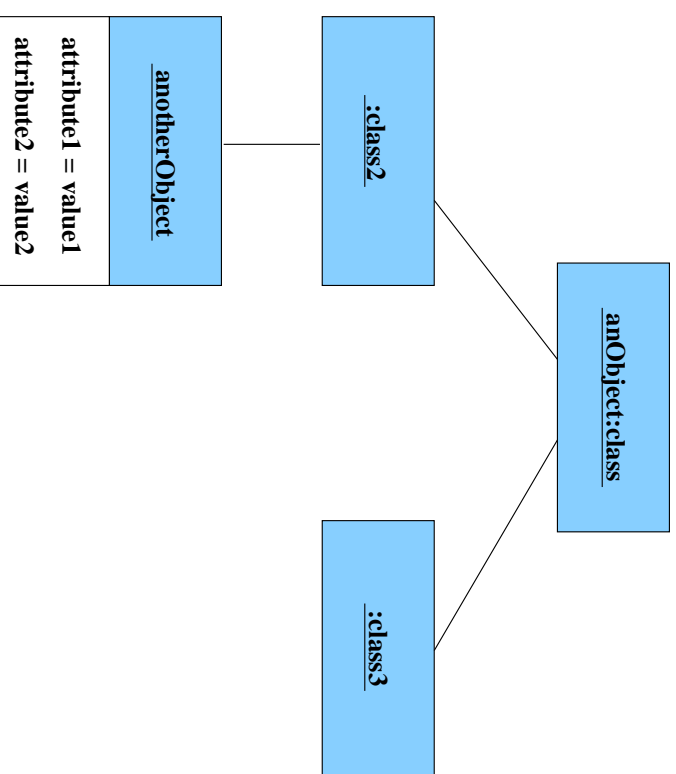
- undirected graph
- **nodes**
 - entity set (**rectangle**)
 - relations between entity sets (**rhombus**)
 - attributes (**rounded rectangle**)
- **edges**
 - entity sets \leftrightarrow relations
 - entity sets or relations \leftrightarrow attributes
- edges labeled with cardinalities (opt.)
- special relations:
 - **is-a** generalization, attributes inherited from parent (**triangle**)
 - **part-of** aggregation

Example: ER-Diagram



3.2.12 Object Diagrams

- graphical representation of **objects** and their **associations**
- UML notation:
 - **nodes**: objects (**rectangles**), labeled with object name:type
 - **edges**: links between objects
“objects that know each other”



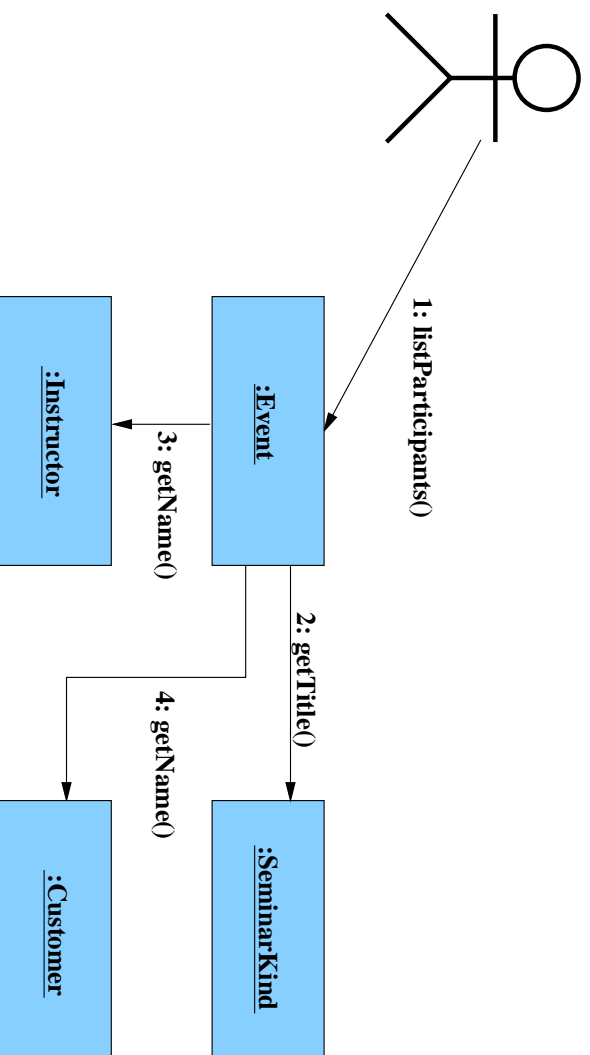
Properties of object diagrams

- snapshot of a system state
- static representation of the configuration of a particular group of objects

dynamic properties → collaboration diagrams

- labeling objects with stereotypes
 - {new}
 - {transient}
 - {destroyed}
- object stands for “any object of that class”
- labeling links with numbered operations
- numbering implies sequence of execution

Example: Collaboration Diagram



3.2.13 Class Diagrams

- representation of **classes** and their static **associations**
 - UML notation is graph with
 - **nodes**: classes (**rectangles**)
 - **edges**:
 - * open arrows from subclass to superclass
 - * associations between classes
- classes whose objects exchange messages
different kinds of association possible

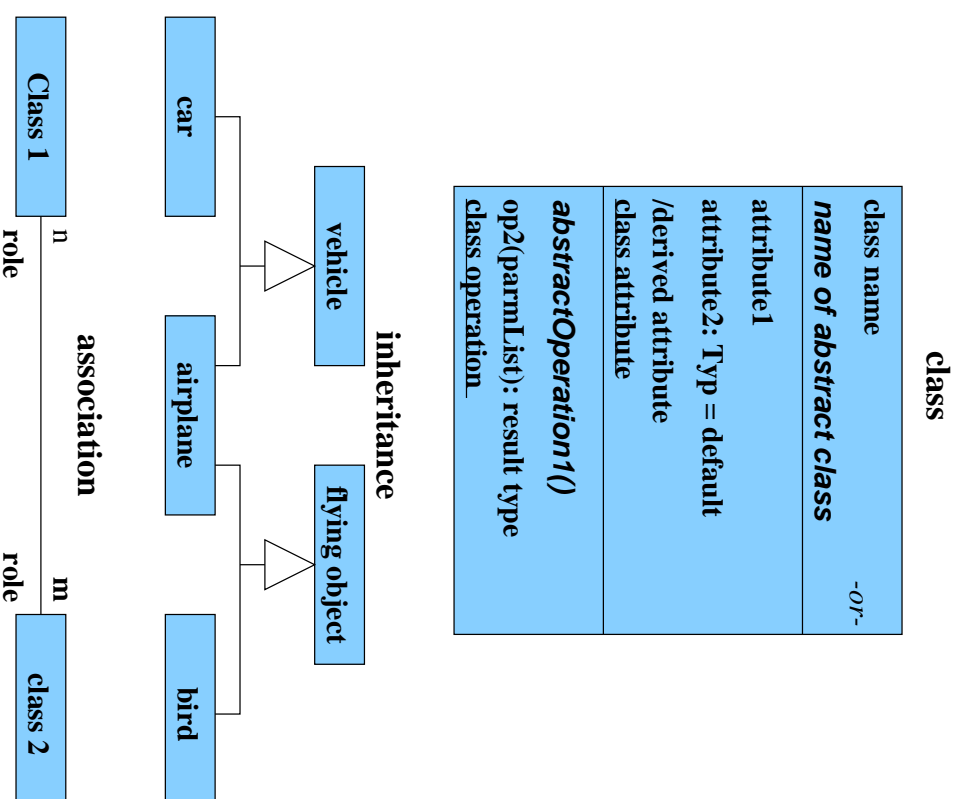
Example: Class Student

Student
registration number
name
grades
<u>count</u>
issue certificate ()
enter grade ()
<u>list degrees ()</u>

Guidelines

- do not model trivial operations like
 - **new**: object creation
 - **delete**: object deletion
 - **set**(Attribute): update an attribute
 - **get**(Attribute): read an attribute
- class attributes and class operations are underlined
- for simplicity: each class “knows” all of its instances in OOA
- administration must be implemented

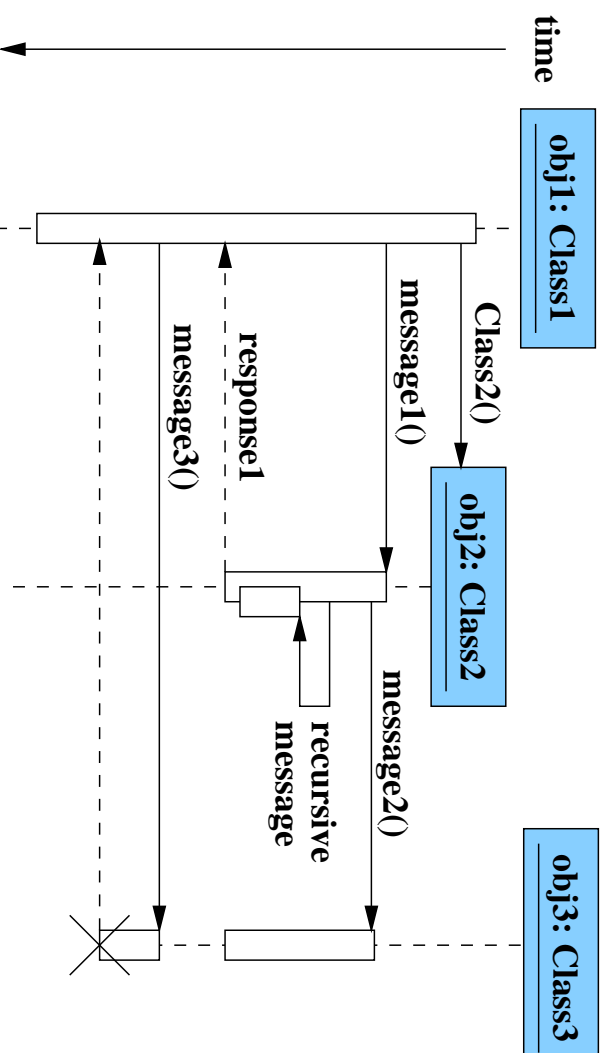
Example: Class Diagram



3.2.14 Sequence Diagram

- description of the sequence of messages
- → communications protocols

Example:



3.2.15 Finite State Machines (FSM)

- modeling the evolving state of an object
e.g., Harel-automata in UML
- \rightarrow real-time systems
- automaton $A = (Q, \Sigma, \delta, q_0, F)$ where
 - Q : finite set of states
 - Σ : finite input alphabet
 - $\delta: Q \times \Sigma \rightarrow Q$ transition function
 - $q_0 \in Q$ initial state
 - $F \subset Q$ set of final states

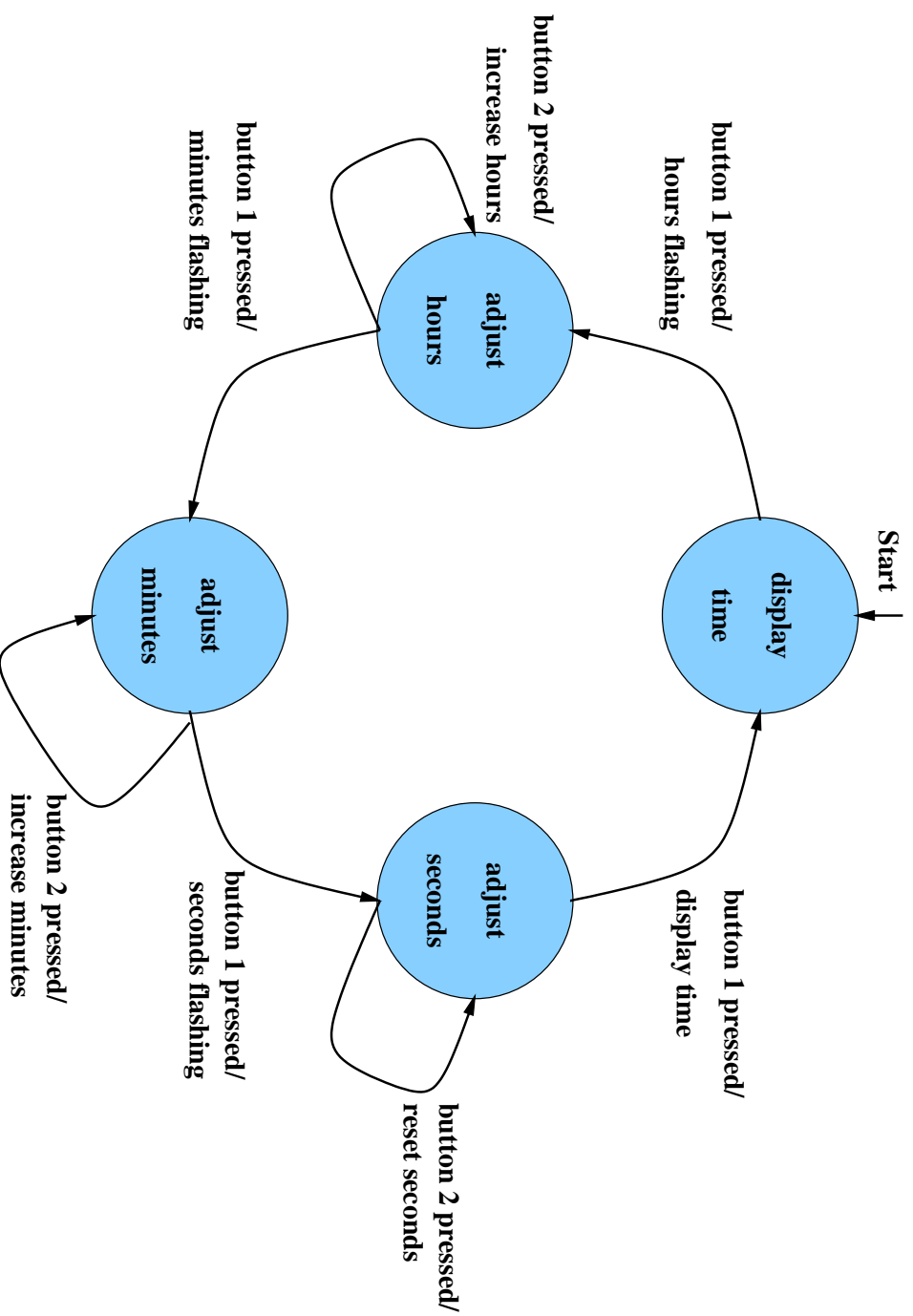
Graphical Representation of FSM

- **nodes:** states of the automaton (circles or rectangles)
- arrow pointing to q_0
- final states indicated by double circle
- **edges:** if $\delta(q, a) = q'$ then directed edge labeled a from q to q'

Often: FSM with output

- $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
- replace final states F by output alphabet Δ and output function λ
- **Mealy-automaton:** $\lambda : Q \times \Sigma \longrightarrow \Delta$
edge from q to $\delta(q, a)$ additionally carries $\lambda(q, a)$
- **Moore-automaton:** $\lambda : Q \longrightarrow \Delta$
state q labeled with $\lambda(q)$

Example: digital clock as a Mealy-automaton

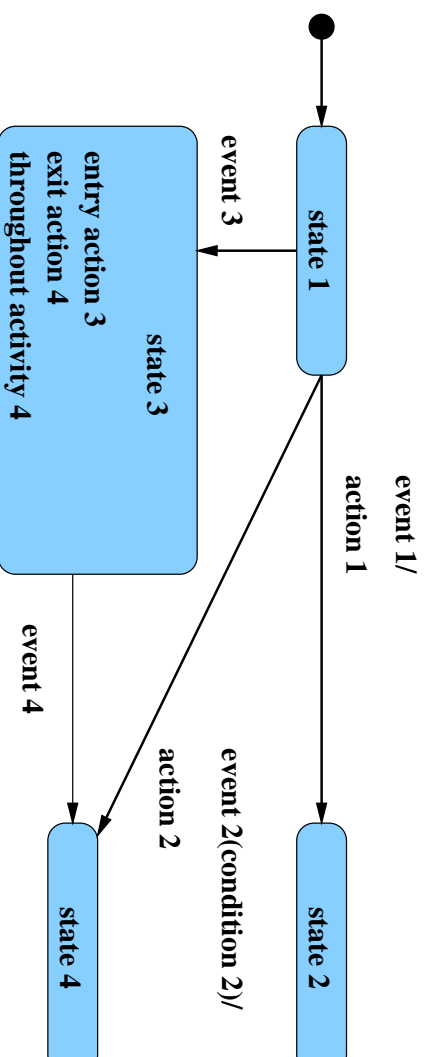


Drawback: FSMs get big too quickly → structuring required

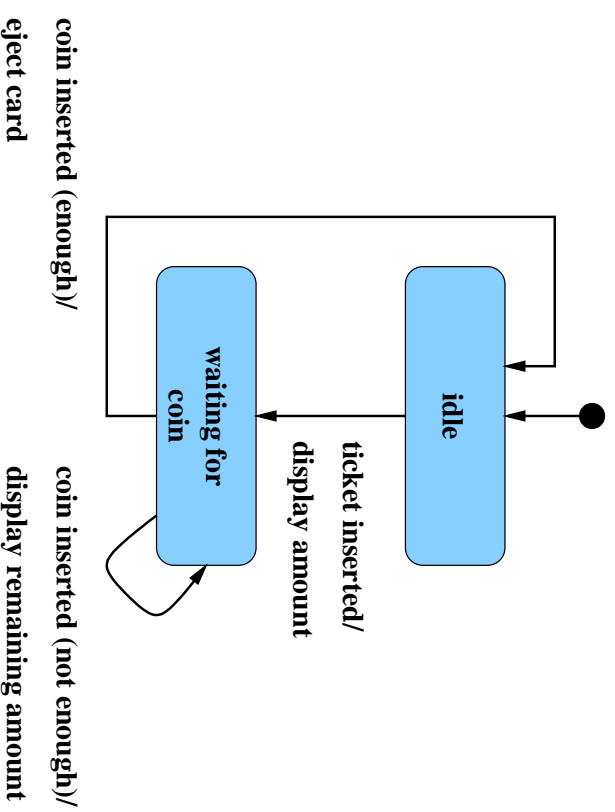
Statecharts (Harel, UML)

- hierarchical state automata
- states with history
- hybrid automata (“Moore + Mealy”)
 - optionally: each state may have
 - **entry action**: executed on entry to state
 - \cong labeling all incoming edges
 - **exit action**: executed on exit of state
 - \cong labeling all outgoing edges
 - **throughout activity**:
executed while in state
- concurrent states
- optional: conditional state transitions

Example: Harel-automaton

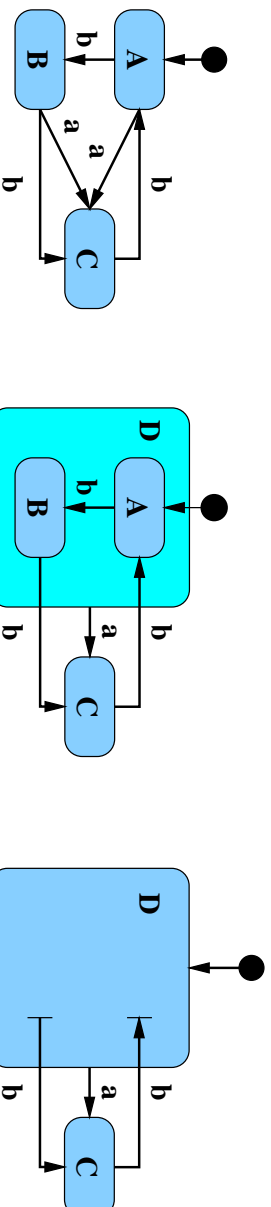


Example: parking lot

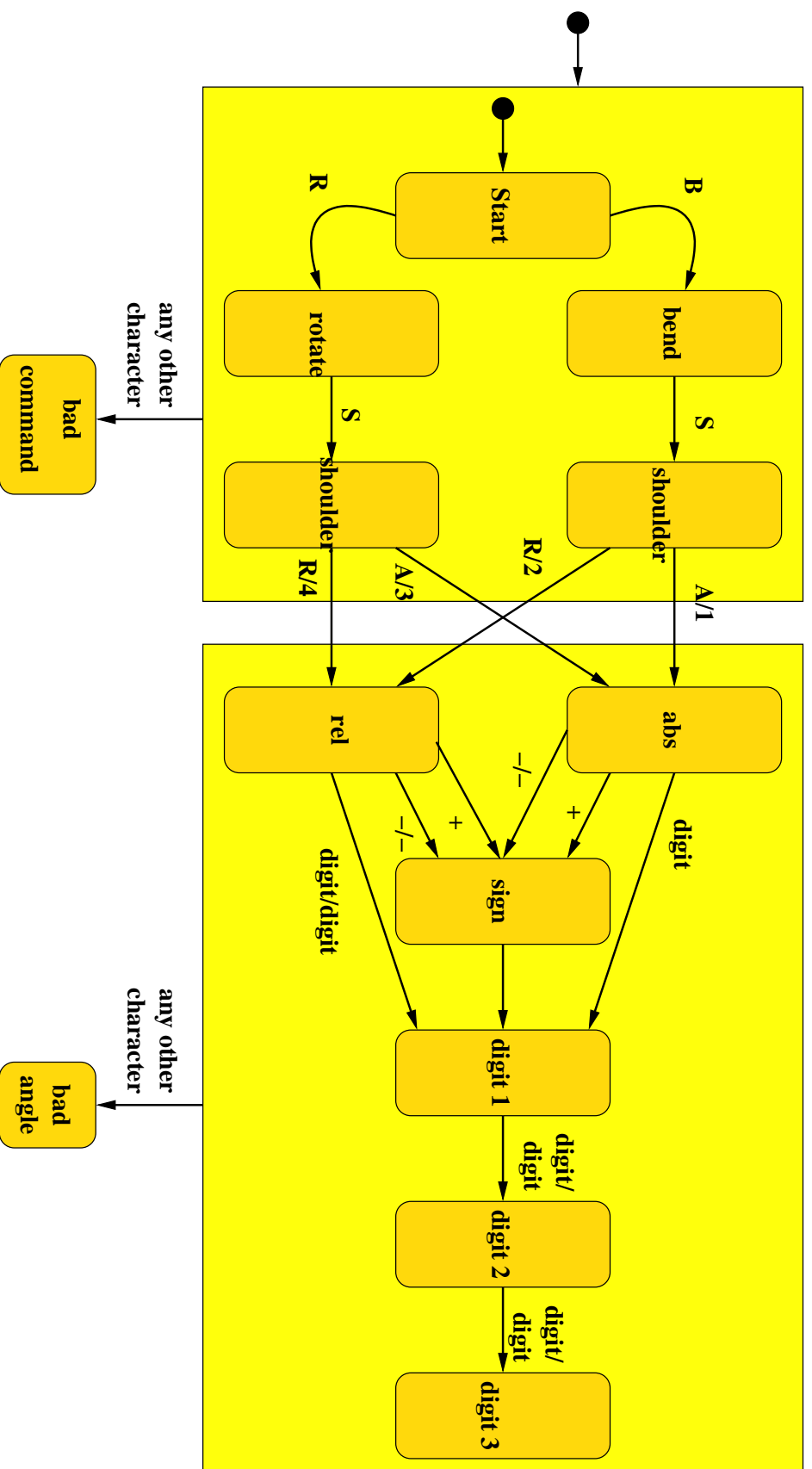


Hierarchical Harel-automata

- states can be grouped into a node (\rightarrow hierarchy)
- edges may start in any level
- labeled edge from/to a state group \cong set of edges with identical labels from/to all members of the group

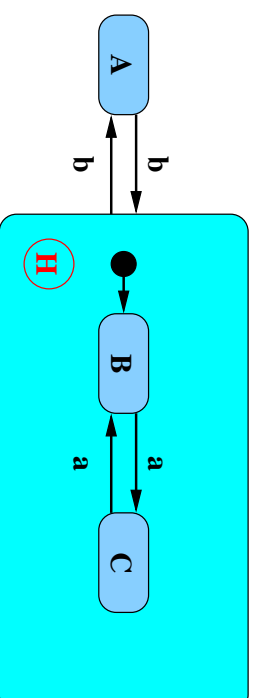


Example: Hierarchical Harel-automaton



States with History

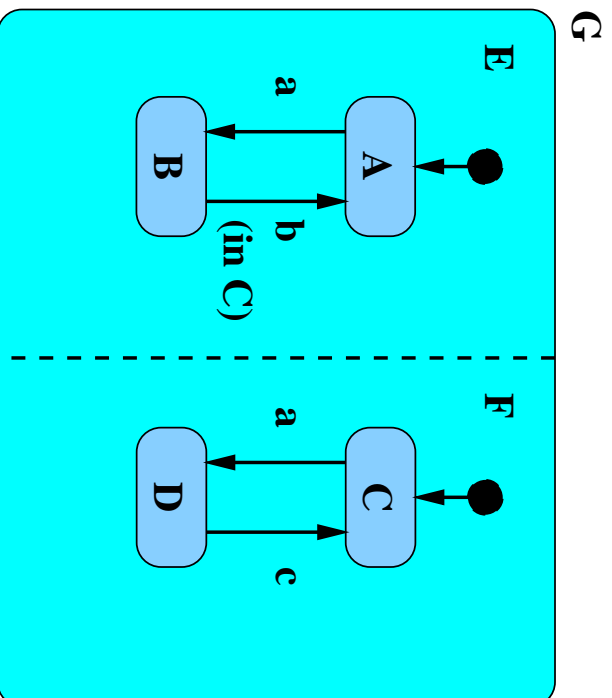
- a state group with history — marked **(H)** — remembers the internal state on exit and resumes in that internal state on the next entry



Concurrent States

- state can be split into **concurrent components** (separated by dashed lines)
- automaton executes all components concurrently
- transitions may depend on state of another component (synchronisation)

Example:



sequence of states on input abcb:

$(A, C), (B, D), (B, D), (B, C), (A, C)$

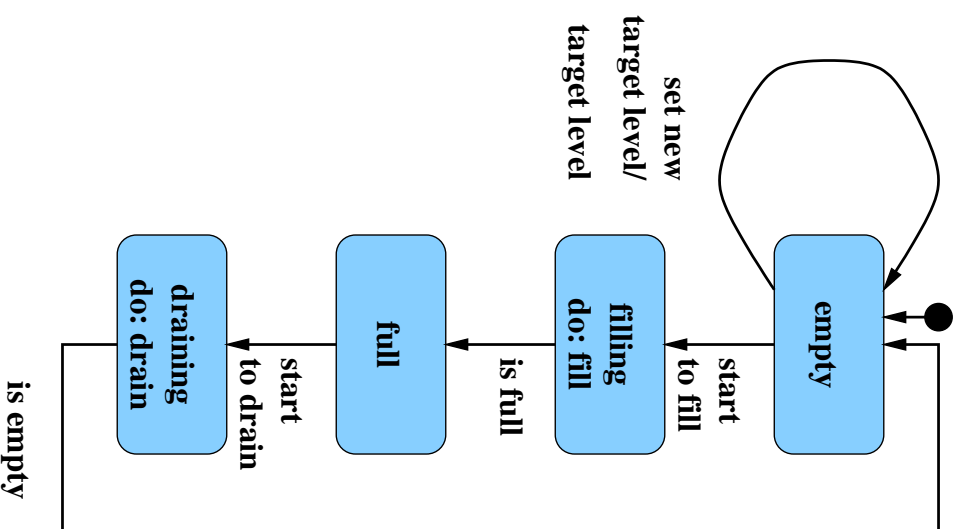
Harel-automata and class diagrams

- operations can only be executed in particular state
- idea: incoming message (in class diagram) $\hat{=}$ event (in HA) that triggers the operation
- trivial event names may be dropped

Alternative use

- class has operation that determines reception of an event

Example: Tank



Tank	
max level	
target level	
current level	
fill	
drain	
set target level	

- can fill only if empty
- can drain only if full

3.2.16 Petri Nets

- scope: modeling of distributed systems (parallel and non-deterministic)

- Petri net: directed bipartite Graph $N = (P, T; F)$ mit

P : set of **places (states)**

$p \in P$ represented by circle

T : set of **transitions**

$t \in T$ represented by rectangle

$F \subseteq (P \times T) \cup (T \times P)$ set of directed edges

- **labeling** $\sigma : P \rightarrow \mathbf{N}$ (**tokens**)

- for $v \in P \cup T$ define $\bullet v := \{v' \mid (v', v) \in F\}$ **domain of v**

- for $v \in P \cup T$ define $v \bullet := \{v' \mid (v, v') \in F\}$ **range of v**

Variants of Petri Nets

- condition/event nets (*Bedingungs/Ereignis-Netze*):
 ≤ 1 token/place (condition)
- place/transition nets (*Stellen/Transitions-Netze*):
 $0 \leq m \leq \kappa(p)$ tokens on place p ($\kappa \hat{=}$ capacity)
- predicate/transition nets:
colored tokens

Condition/Event Nets

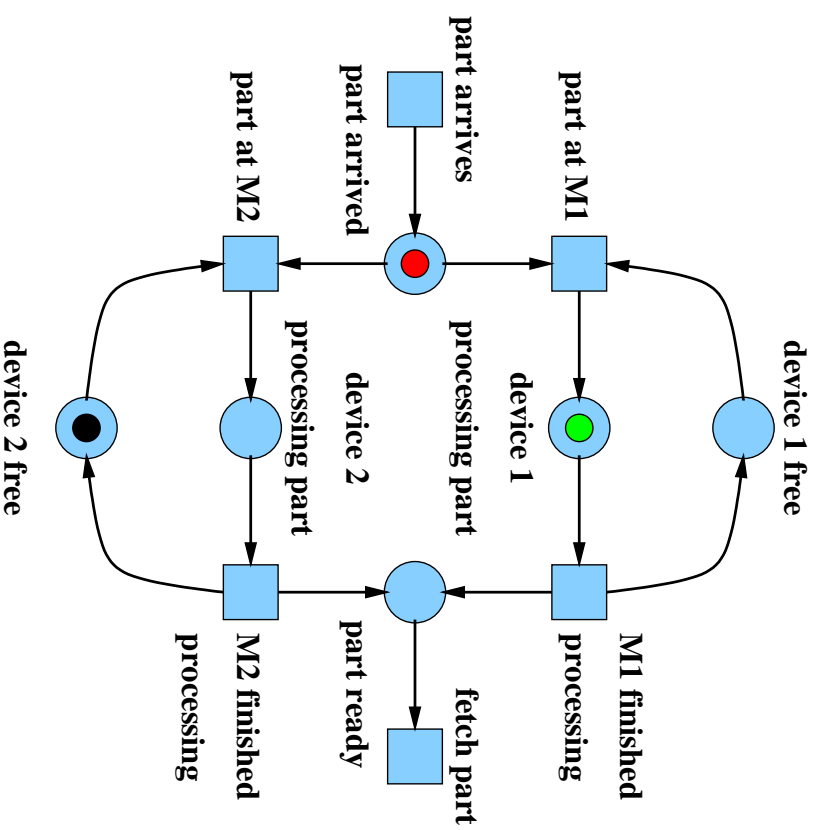
- $\sigma : P \longrightarrow \{0, 1\}$

Switching rule

- a transition t can fire if each place in the domain of t contains a token and each place in the range of t is empty:
 $(\forall p \in \bullet t) \sigma(p) = 1$ and $(\forall p \in t \bullet) \sigma(p) = 0$
- if t fires, then one token is removed from each place in the domain of t and one token is added to each place in the range of t :

$$\sigma'(p) := \begin{cases} 0, & \text{if } p \in \bullet t \\ 1, & \text{if } p \in t \bullet \\ \sigma(p), & \text{otherwise} \end{cases}$$

Example: C/E Net



conflict → non-determinism

Place/Transition Nets

- each place p has **maximum capacity** $\kappa(p) \in \mathbf{N}$
- each edge t has a **weight** $\beta(t) \in \mathbf{N}$
 $\hat{=}$ # tokens transferred
- generalization of condition/event nets:
a c/e net is a p/t net where
 - $(\forall p \in P) \kappa(p) = 1$
 - $(\forall t \in T) \beta(t) = 1$

Switching Rule

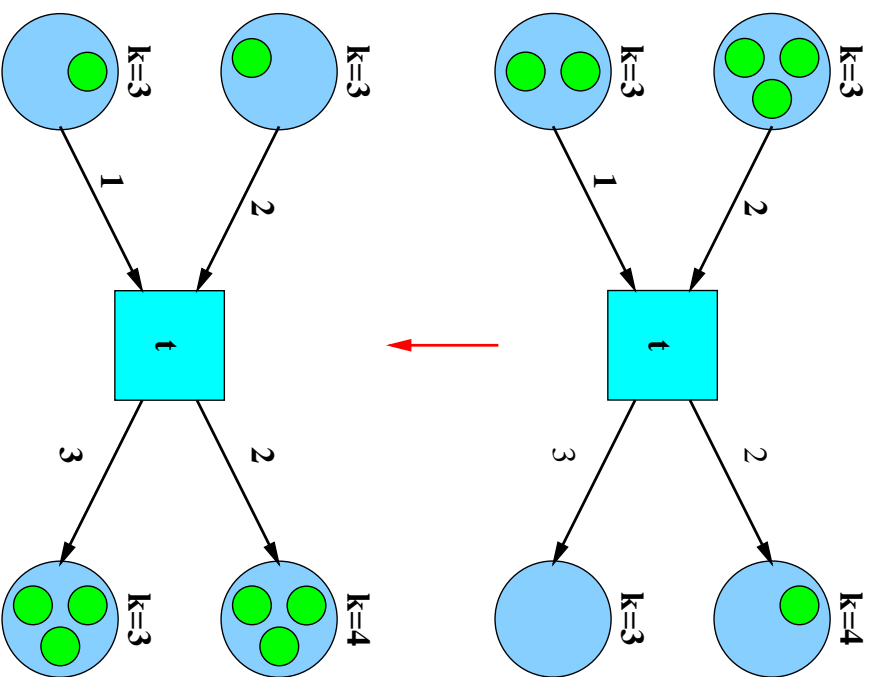
A transition t fires if each place in the domain contains a sufficient number of tokens and if each place in the range of t can take on a sufficient number of tokens:

- $(\forall p \in \bullet t) \sigma(p) \geq \beta((p, t))$ and
- $(\forall p \in t \bullet \setminus \bullet t) \kappa(p) \geq \beta((t, p)) + \sigma(p)$ and
- $(\forall p \in t \bullet \cap \bullet t) \kappa(p) \geq \beta((t, p)) - \beta((p, t)) + \sigma(p)$

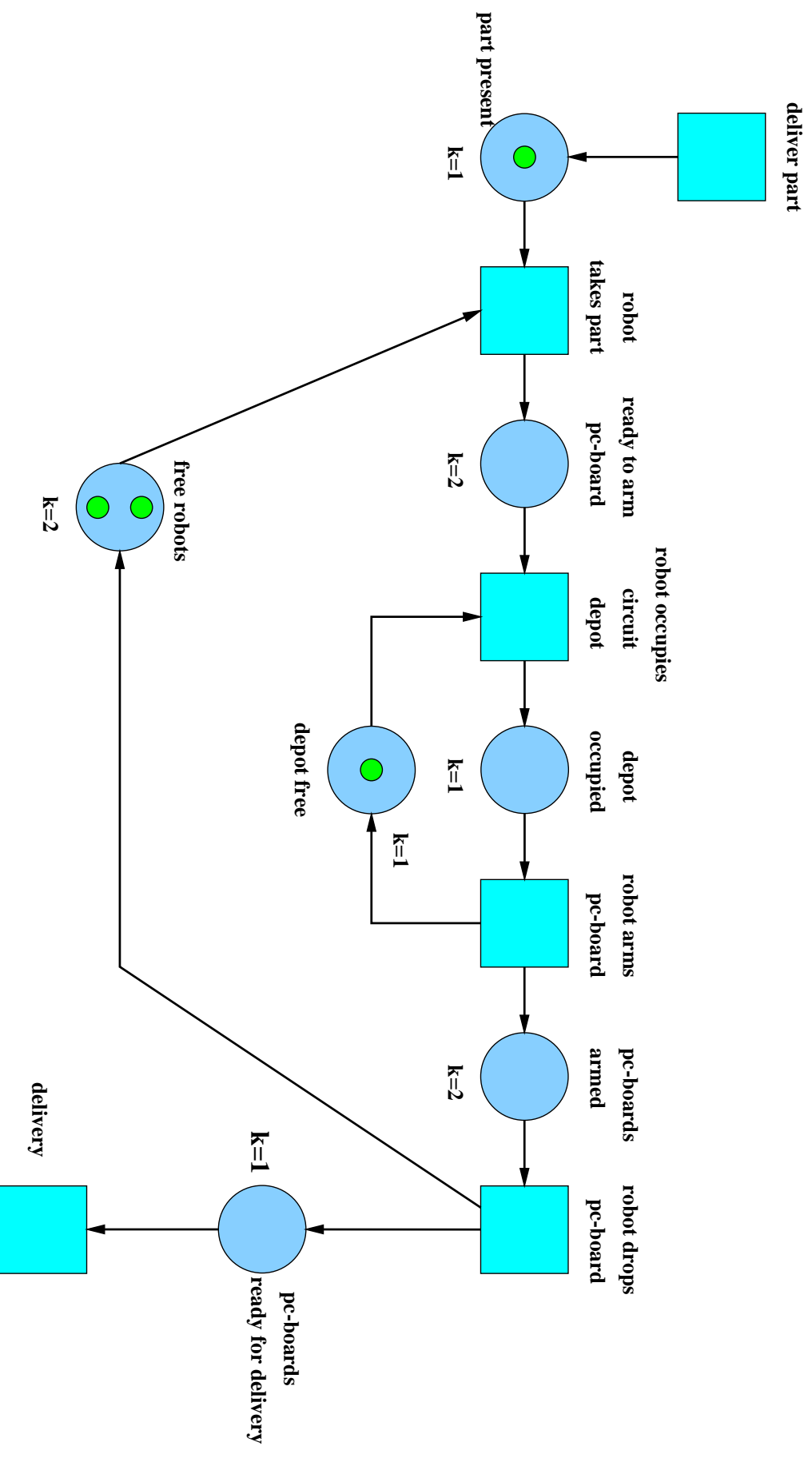
If t fires, then — according to β — tokens are removed from each place in the domain of t and token are added to each place in the range of t :

$$\sigma'(p) := \begin{cases} \sigma(p) - \beta((p, t)), & \text{if } p \in \bullet t \setminus t \bullet \\ \sigma(p) + \beta((t, p)), & \text{if } p \in t \bullet \setminus \bullet t \\ \sigma(p) + \beta((t, p)) - \beta((p, t)), & \text{if } p \in t \bullet \cap \bullet t \\ \sigma(p), & \text{otherwise} \end{cases}$$

Example: switching rule

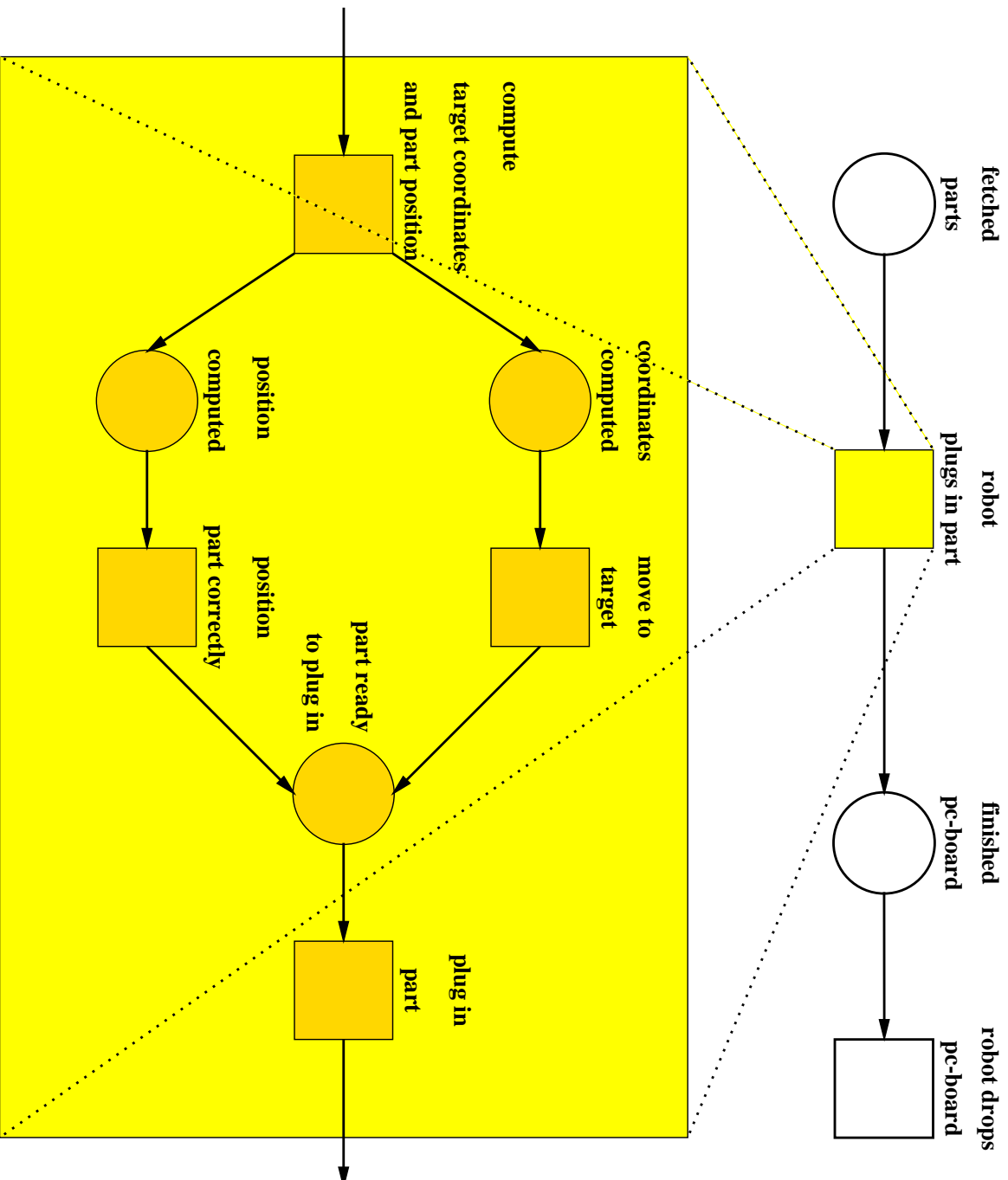


Example: s/t net

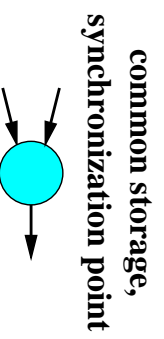
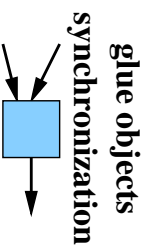
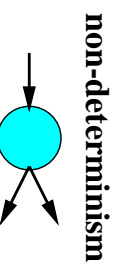
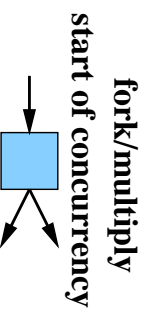
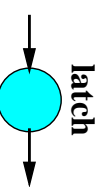
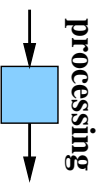
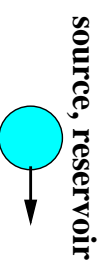
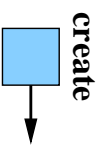
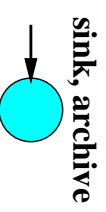
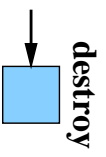


Hierarchical Petri Nets

- improved structuring, more tidy
- refinement of places and transitions by sub-nets
- refinement must respect edges and tokens
- recommended: top-down-approach



Typical Structures



Advantages and Disadvantages of Petri Nets

- Properties can be checked by tools
termination, liveness, deadlock freedom, . . .
(→ theoretical foundation; simulation)
- not combined with other concepts
- only static structure
- no methodology for developing Petri nets