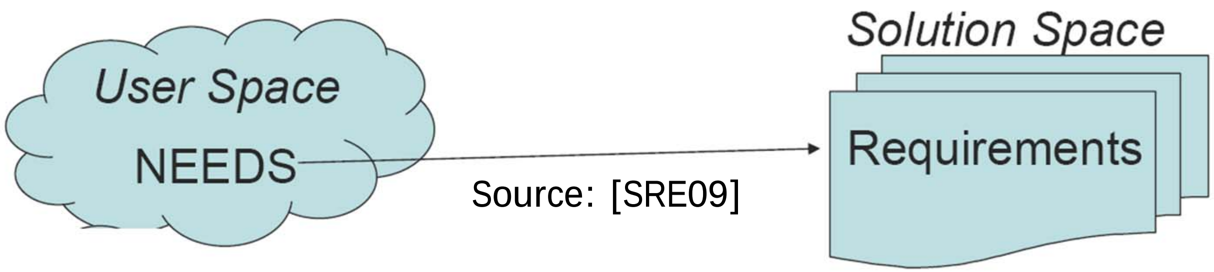
Embedded Systems 4

REVIEW

What is a requirement ?

- Describes **what** the system should do but not how to implement it
- IEEE 1012 standard
  (IEEE= Institute of Electrical and Electronics Engineers)
  - A condition or capability of the system needed by a user to solve a problem or achieve an objective
  - A condition or capability that must be met or possessed by a system… to satisfy a contract standard, specification, or other formally imposed document



- Ranges from a high-level abstract statement of a service (function, feature) or of a system constraint to a detailed mathematical functional specification

Examples of functional requirements REVIEW

- When the Memory receives a READ request it shall transmit the data at the given address to the controller.

- When the Memory receives a WRITE request it shall store the data to the given address.

- When the Memory receives a READ request it shall transmit the data at the given address to the controller within 55,70ns.

REVIEW

Non-functional requirements

- Product requirements
  - Requirements which specify that the delivered product must have certain qualities e.g. execution speed, reliability, etc.
    - *8.1 The memory shall have an equal cycle time of 55,70ns. (functional)*
    - *8.2 The operational voltage of the memory shall be between 4,5V and 5,5V.*

- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc
    - *9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.*

- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. safety, interoperability requirements, legislative requirements, etc

Mealy automaton

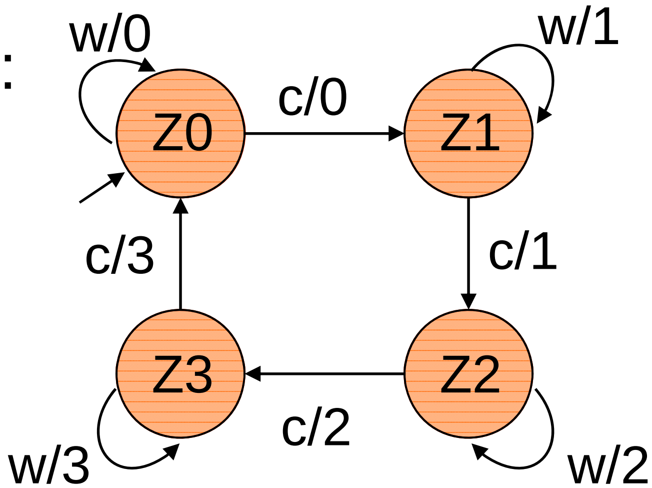
REVIEW

- **Definition:**

  $M=(I, O, S, s_0, \delta, \lambda)$ is a Mealy automaton iff

  - I is a finite, non-empty set (input symbols),
  - O is a finite, non-empty set (output symbols),
  - S is a finite, non-empty set (states),
  - s_0 … initial state,
  - $\delta : S \times I \rightarrow S$ (transition function),
  - λ : *S* £ *I* ! O (output function).

- Example for representation:



StateCharts

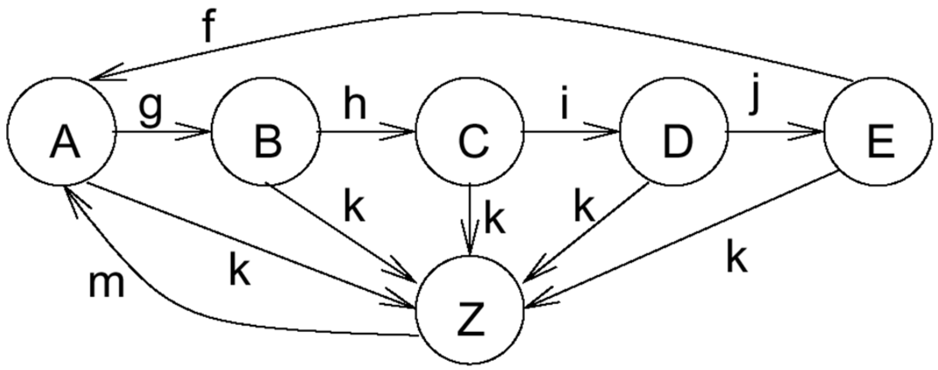
REVIEW

- StateCharts = the only unused combination of
  „flow“ or „state“ with „diagram“ or „charts“
- Based on classical automata (FSM):
  StateCharts = FSMs + Hierarchy + Orthogonality + Broadcast communication
- Industry standard for modelling automotive applications
- Appear in UML (Unified Modeling Language), Stateflow, Statemate, …
- *Warning: Syntax and Semantics may vary.*

- **Start with brief review on Finite State Machines.**

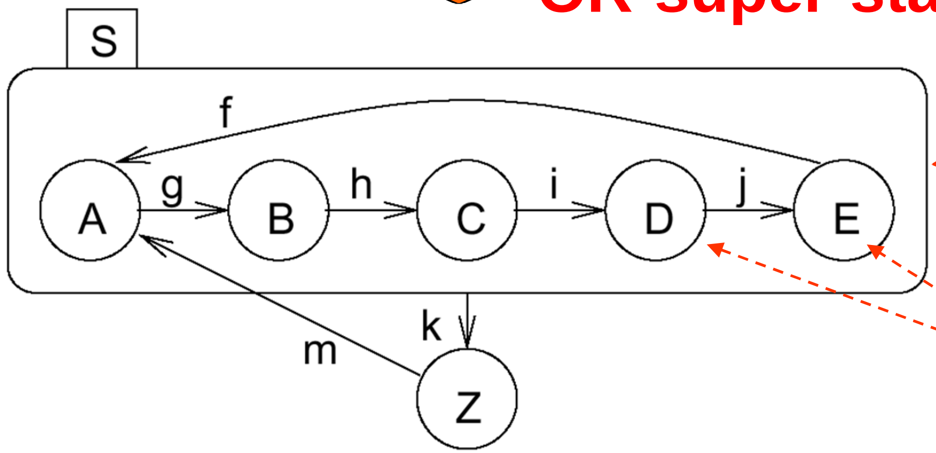
Introducing hierarchy

REVIEW



OR-super-states

FSM will be **in** exactly one of the substates of S if S is **active**
(either in A or in B or ..)



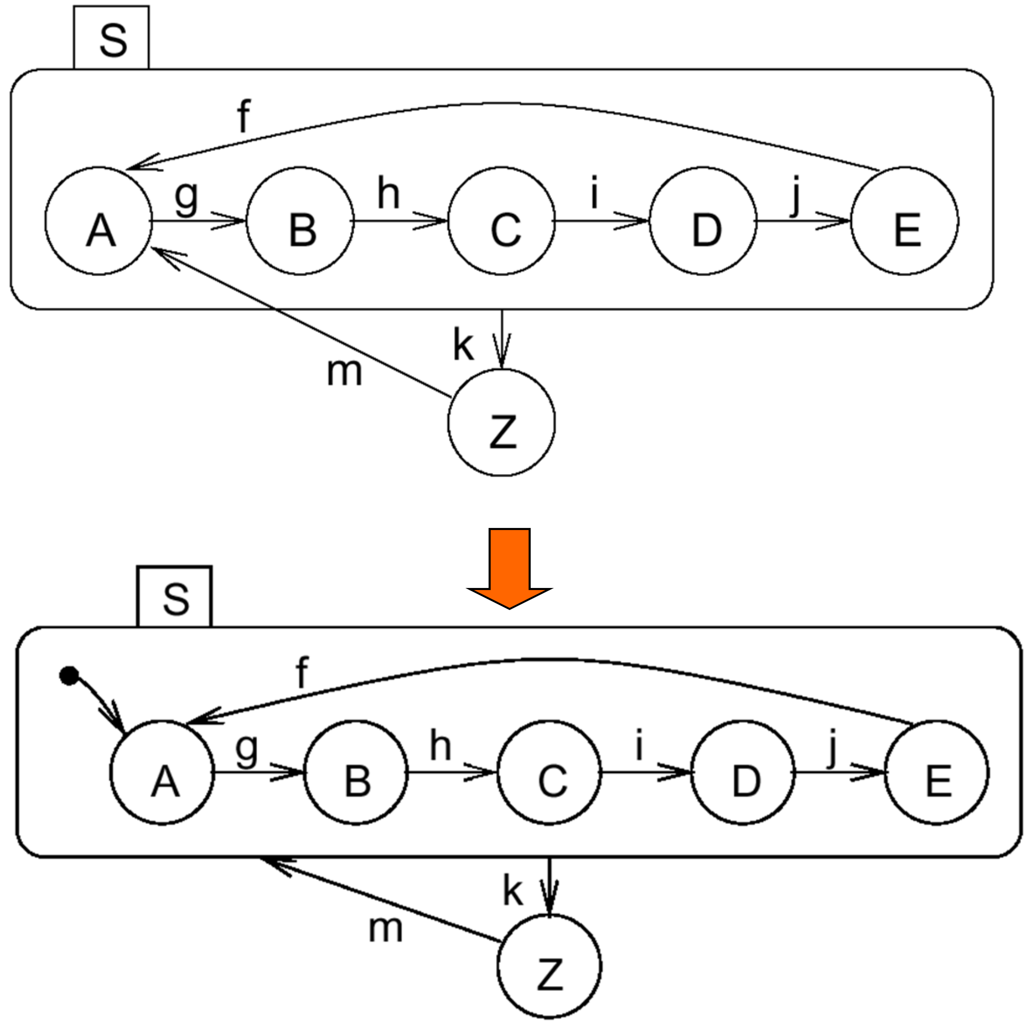
superstate

substates

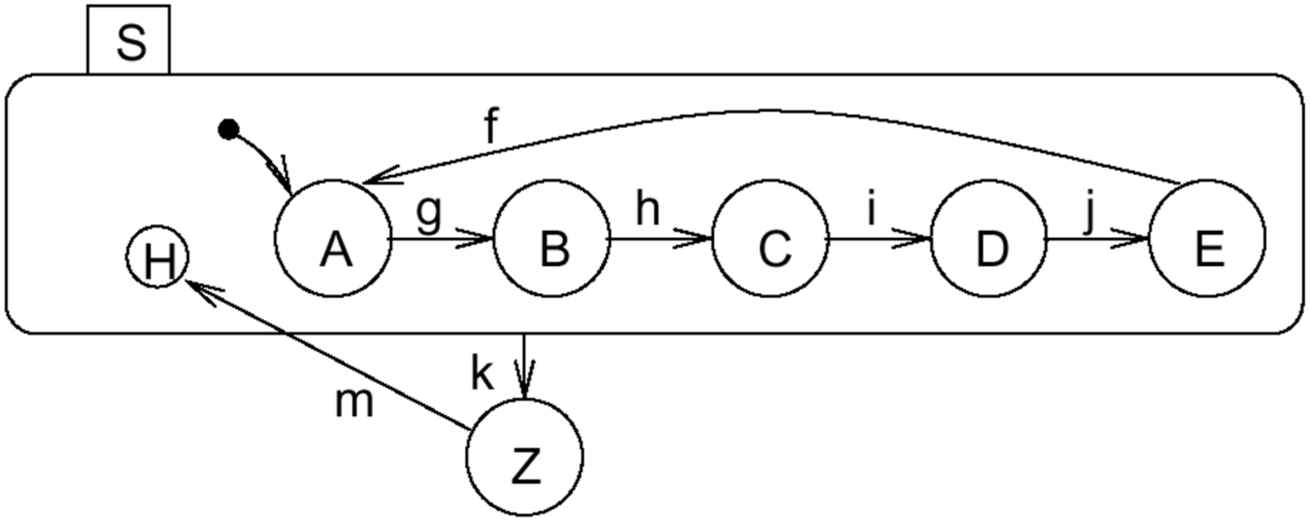
Default state mechanism

REVIEW

- Filled circle indicates sub-state entered whenever super-state is entered.
- Not a state by itself!
- Allows internal structure to be hidden for outside world



History mechanism



- For event m, S enters the state it was in before S was left (can be A, B, C, D, or E). If S is entered for the very first time, the default mechanism applies.

General form of edge labels

REVIEW

Meaning:

- Transition may be taken, if event occurred in last step and condition is true
- If transition is taken, then reaction is carried out.

Conditions:

- Refer to values of variables

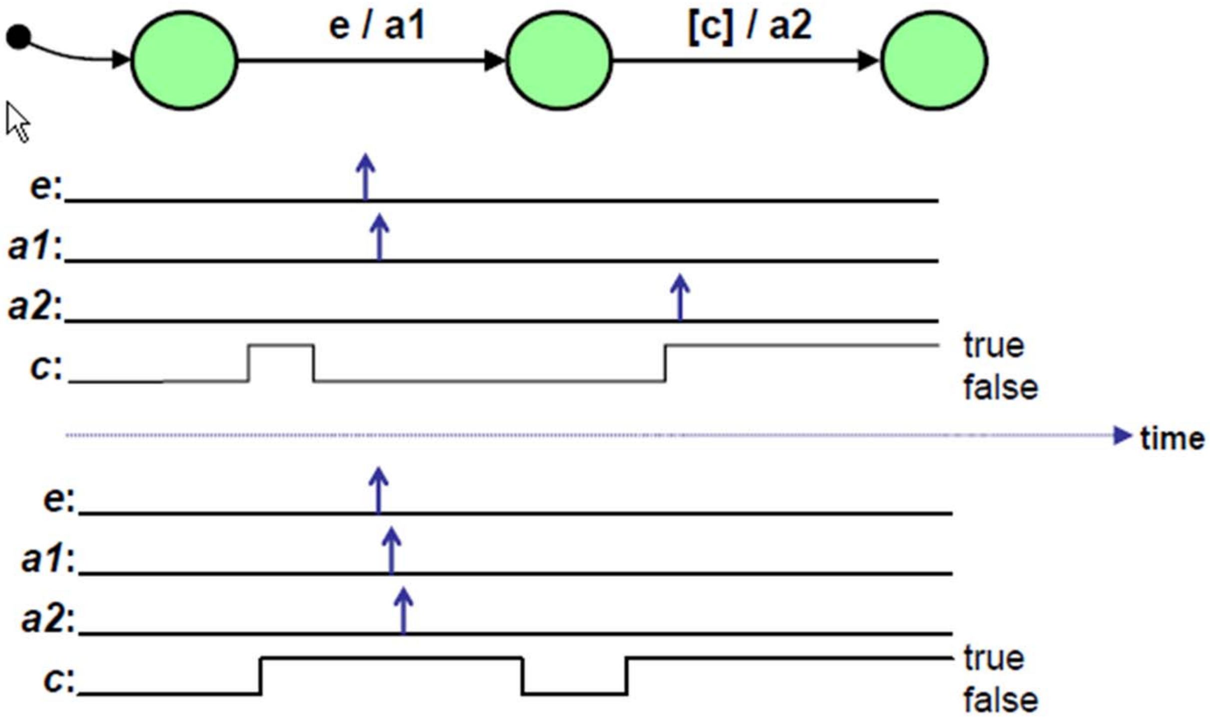
Actions:

- Can either be assignments for variables or creation of events

Example:

- a & [x = 1023] / overflow; x:=0

Example

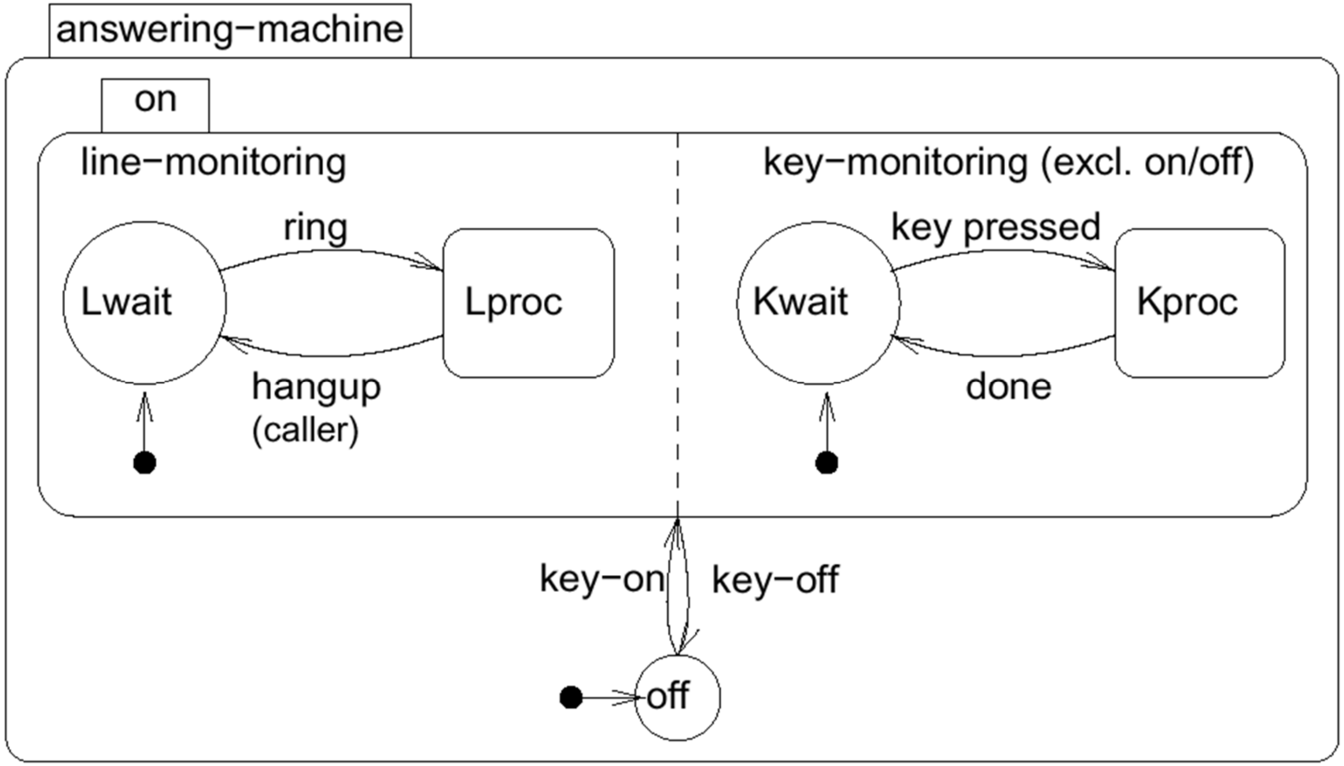


Wattenhofer, ETHZ

Concurrency

REVIEW

- Convenient ways of describing concurrency are required.
- **AND-super-states**: FSM is in **all** (immediate) sub-states of a AND-super-state; Example:



Types of states

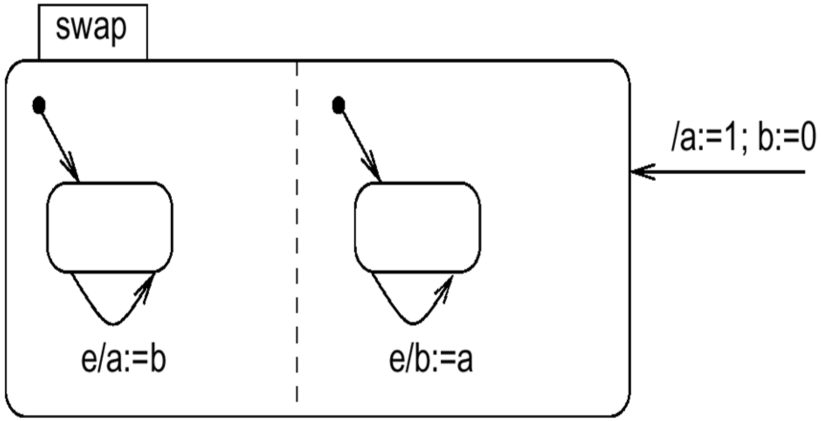
REVIEW

In StateCharts, states are either

- **basic states,** or
- **AND-super-states,** or
- **OR-super-states.**

Edge labels evaluation

REVIEW



Three phases:

1.Effect of external changes on events and conditions is evaluated

2.The set of transitions to be made in the current step and right hand sides of assignments are computed

3.Transitions become effective, variables obtain new values

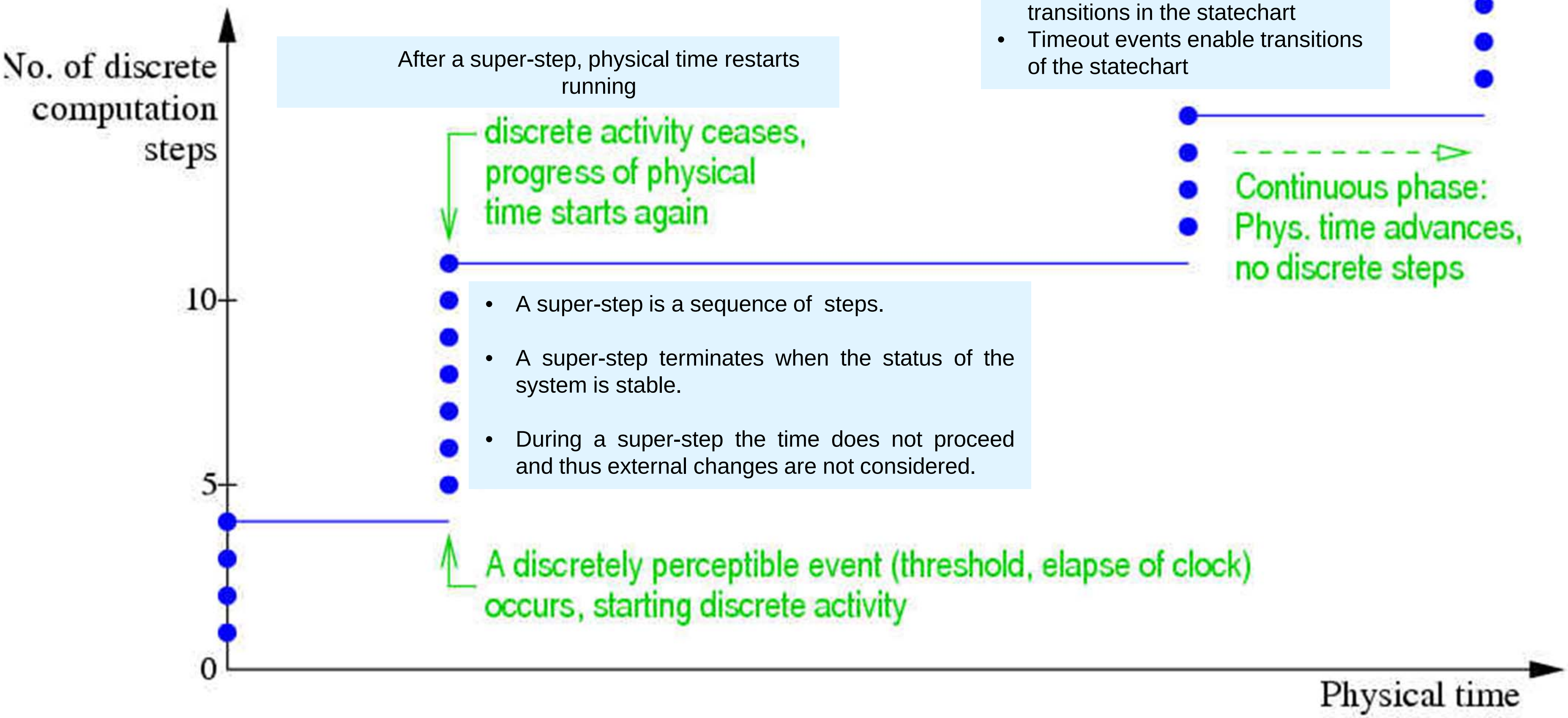
- In phase 2, variables a and b are assigned to temporary variables. In phase 3, these are assigned to a and b. As a result, variables a and b are swapped.
- In a single phase environment, executing the left state first would assign the old value of b (=0) to a and b. Executing the right state first would assign the old value of a (=1) to a and b. The result would depend on the execution order.

The super-step time model (2)

REVIEW

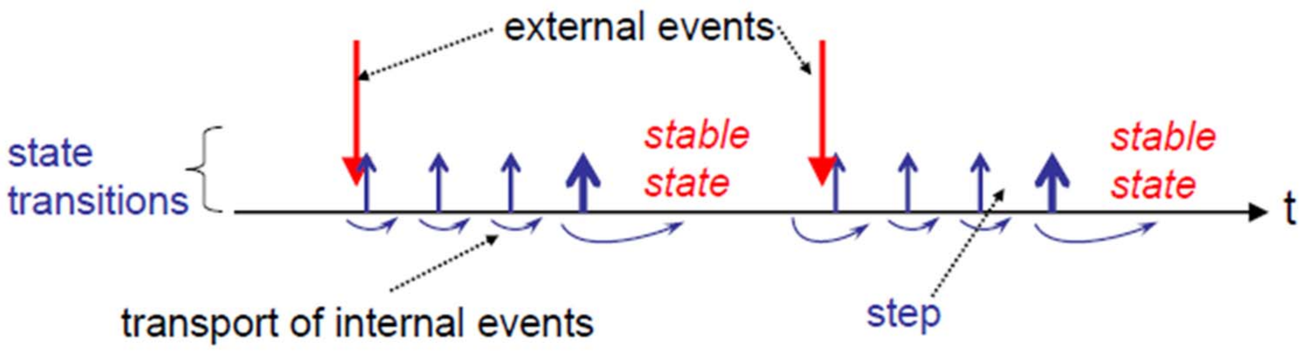
- Two-dimensional time:

computation of the statechart is resumed when
- external changes enable transitions in the statechart
- Timeout events enable transitions of the statechart



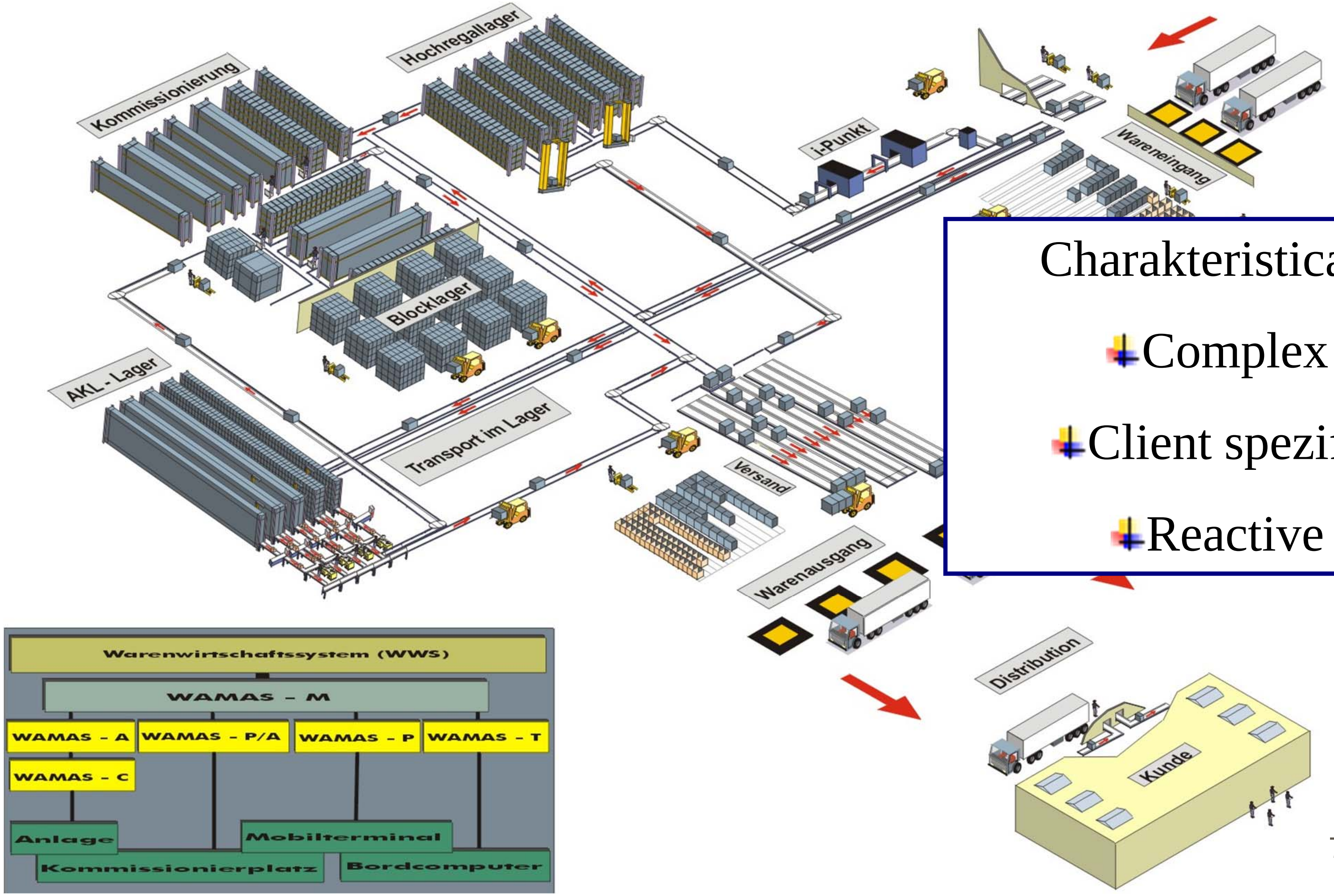
- Assumption: Computation time is neglegible compared to dynamics of the environment.

Another point of view



Wattenhofer, ETHZ

Use case – logistic system



Charakteristica

- Complex
- Client spezific
- Reactive

Different approaches to earn money with software

- **Write software. Burn CD. Box it. Sell to everyone.**

- ~~**Leave customer alone**~~
  - ~~**(he has to adapt his problem!).**~~

- **Customers need solutions to their problems**

- **individual, working solutions ! flexibility !**

  - **Software REUSE**

IDEA

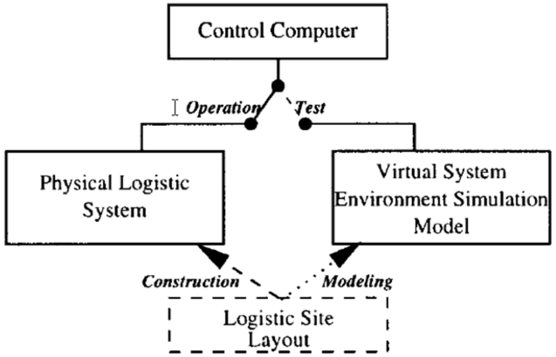


Figure 1. Environment simulation for software testing

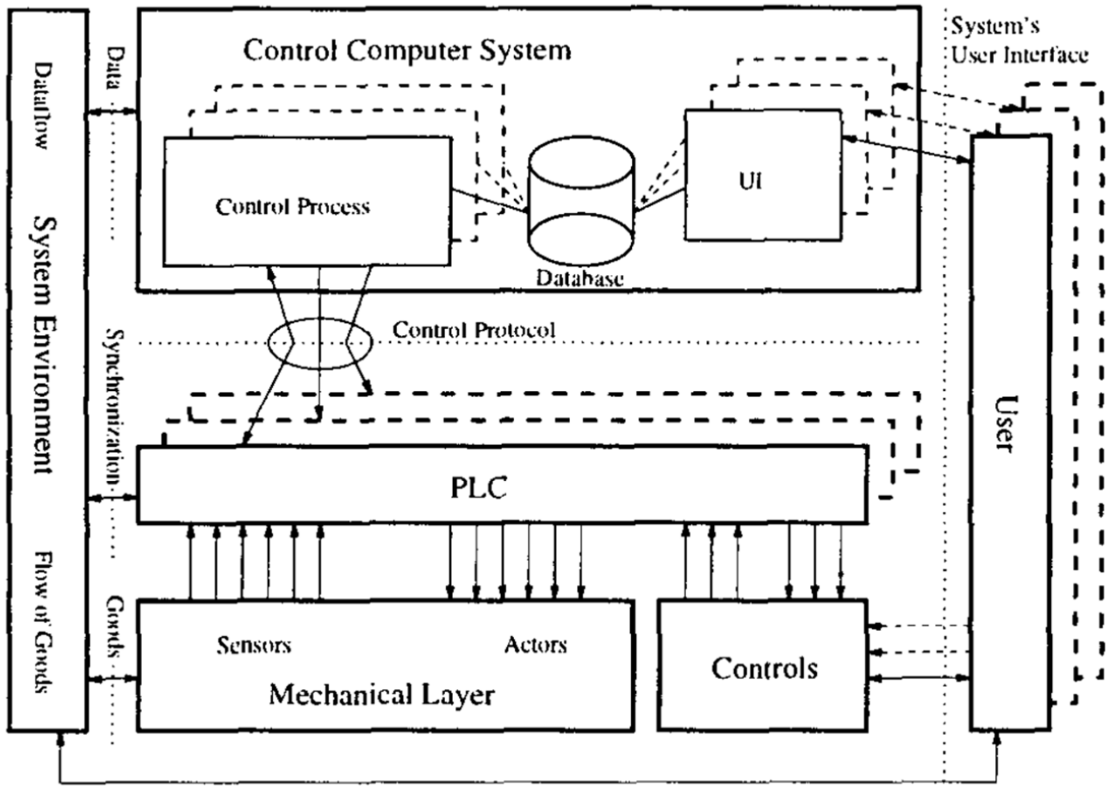


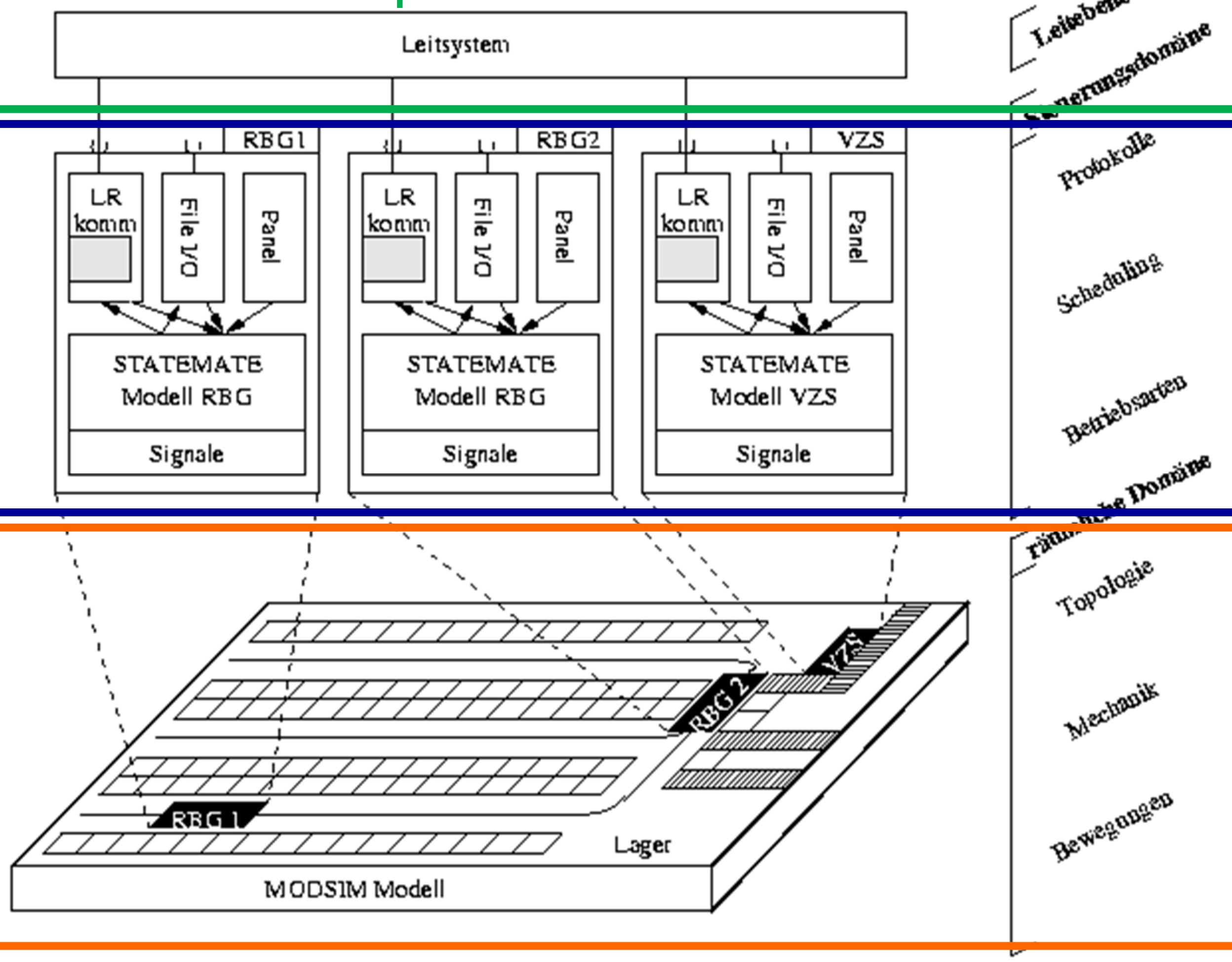
Figure 2. Control architecture of automatic logistic systems

Modeling domains

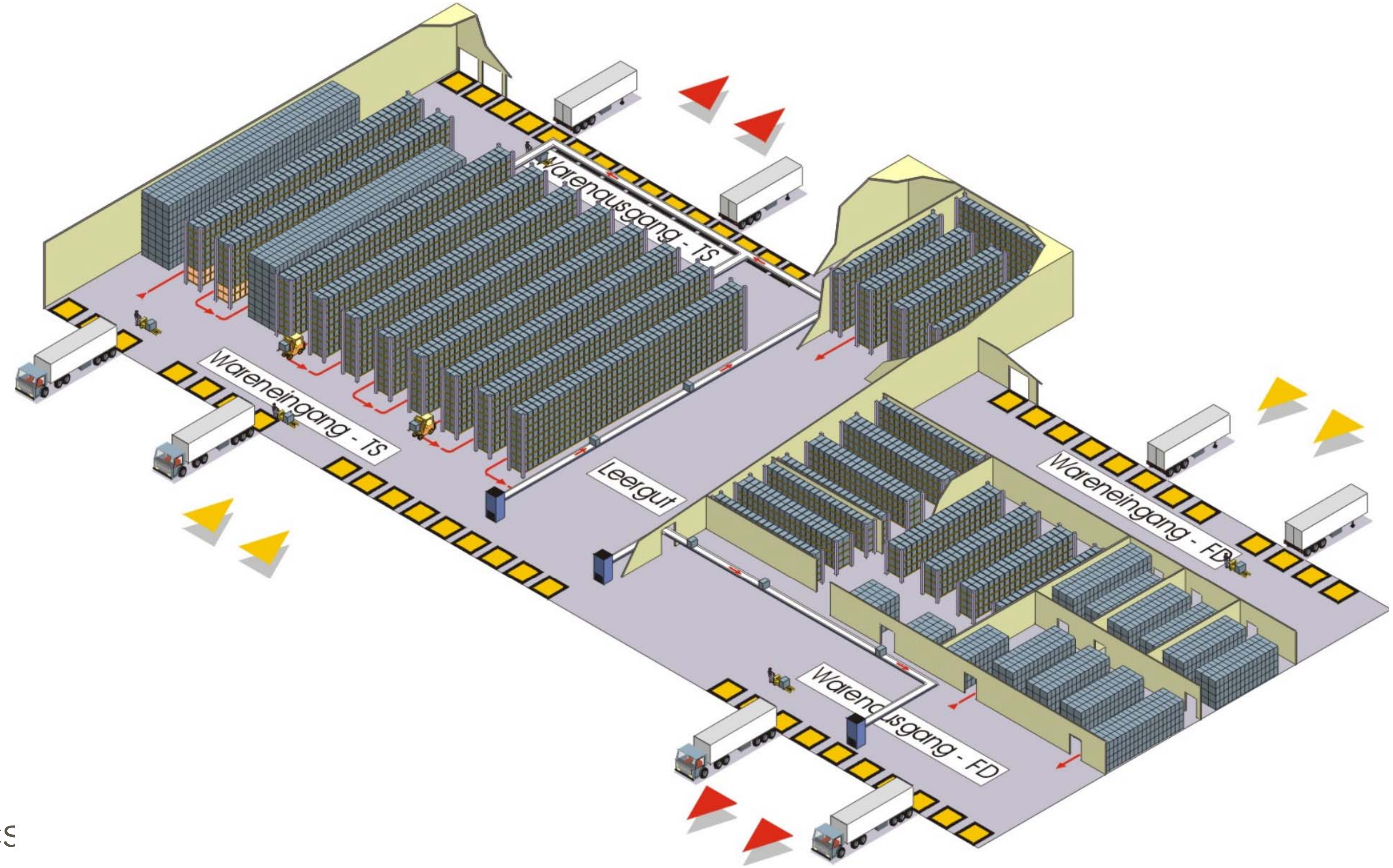
- Geometry domain(⇨MODSIM)
  - Topology (elements, parametrisation)
  - Movements (animation, Interfcaces)
- Control domain (⇨STATEMATE)
  - Communication protocols (to central computer)
  - Scheduling of the transport orders
- Different tools⇨Cosimulation

Example- High rack warehouse

Software under development

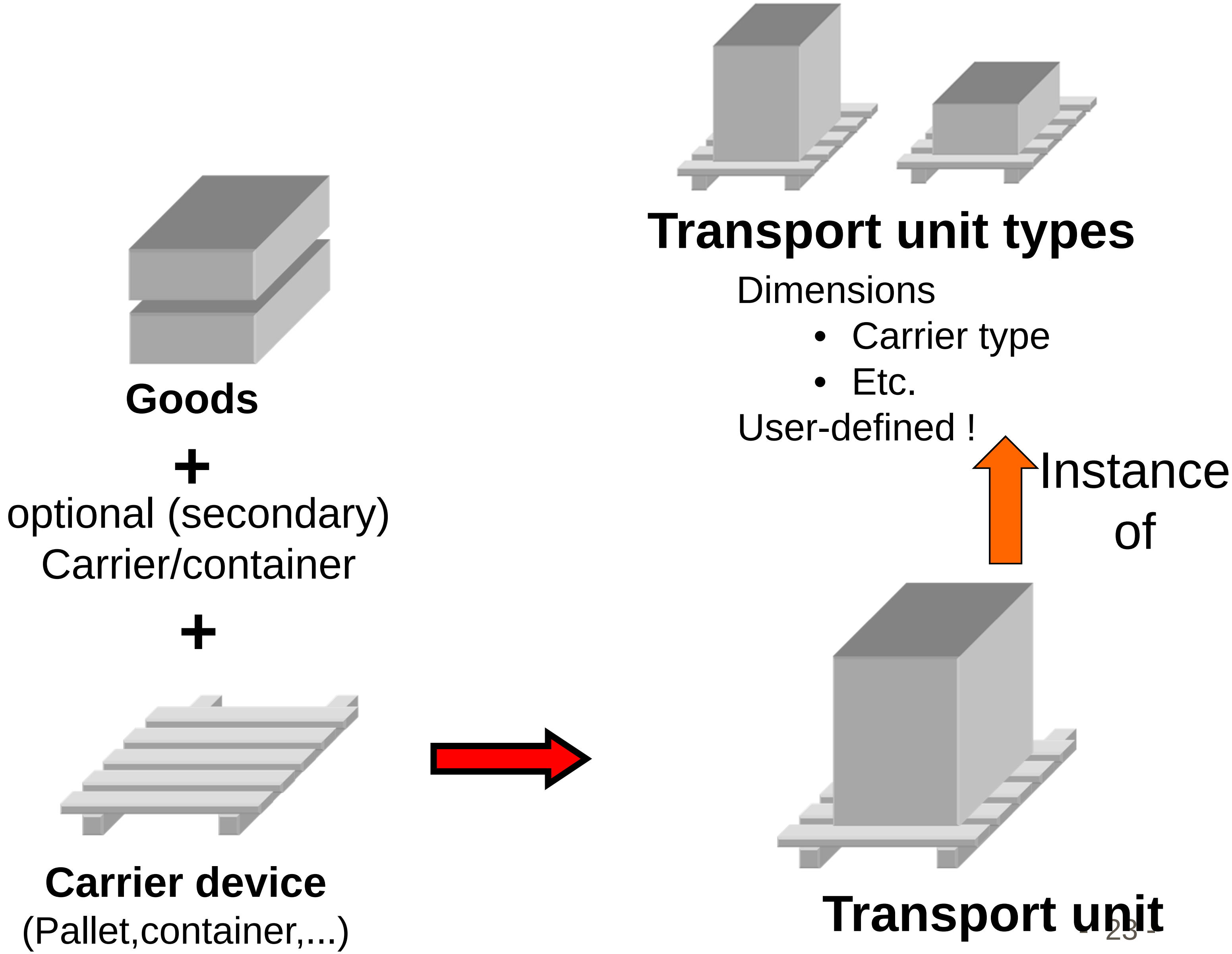


What do we find ?

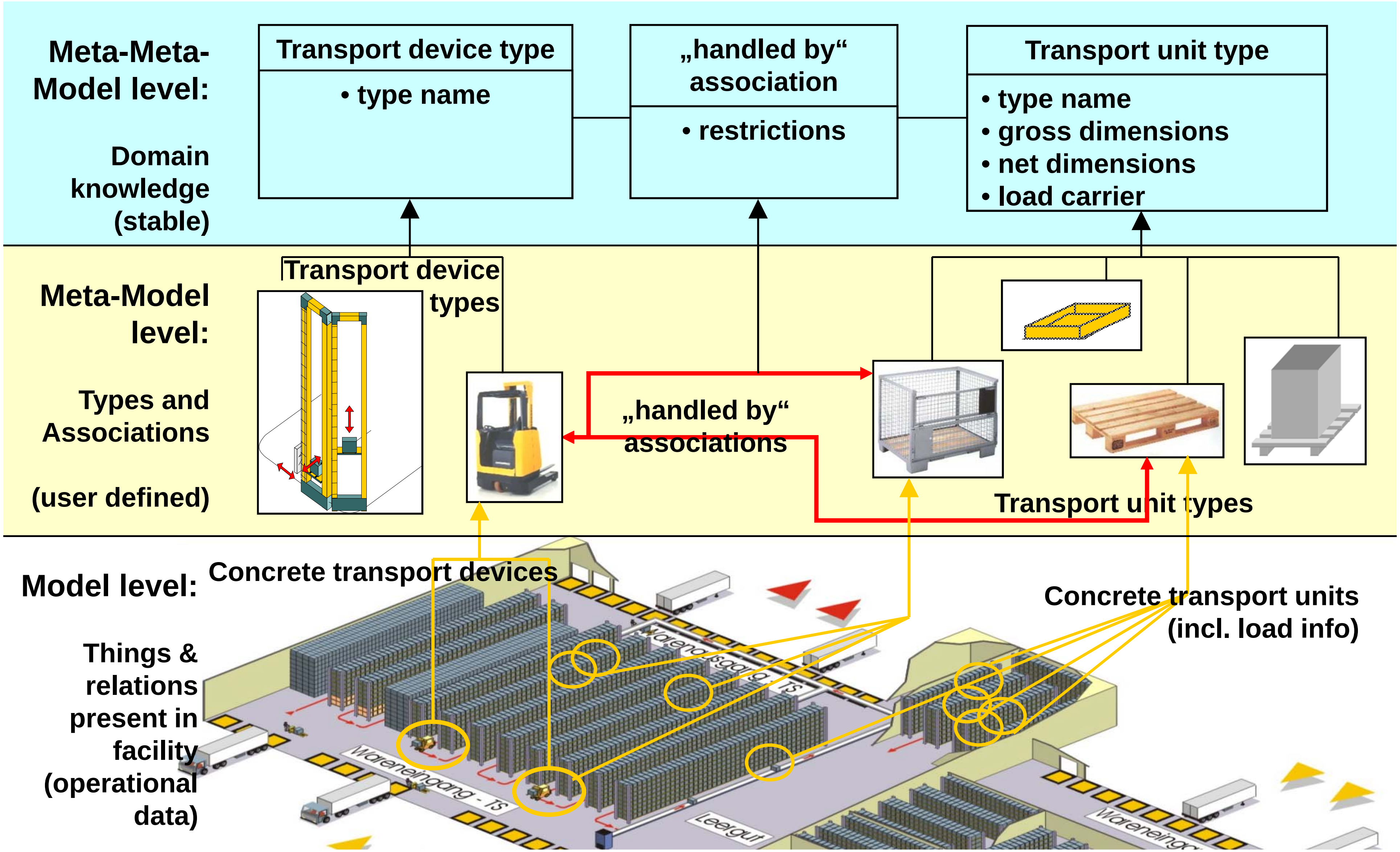


CS

... Something to store + handle

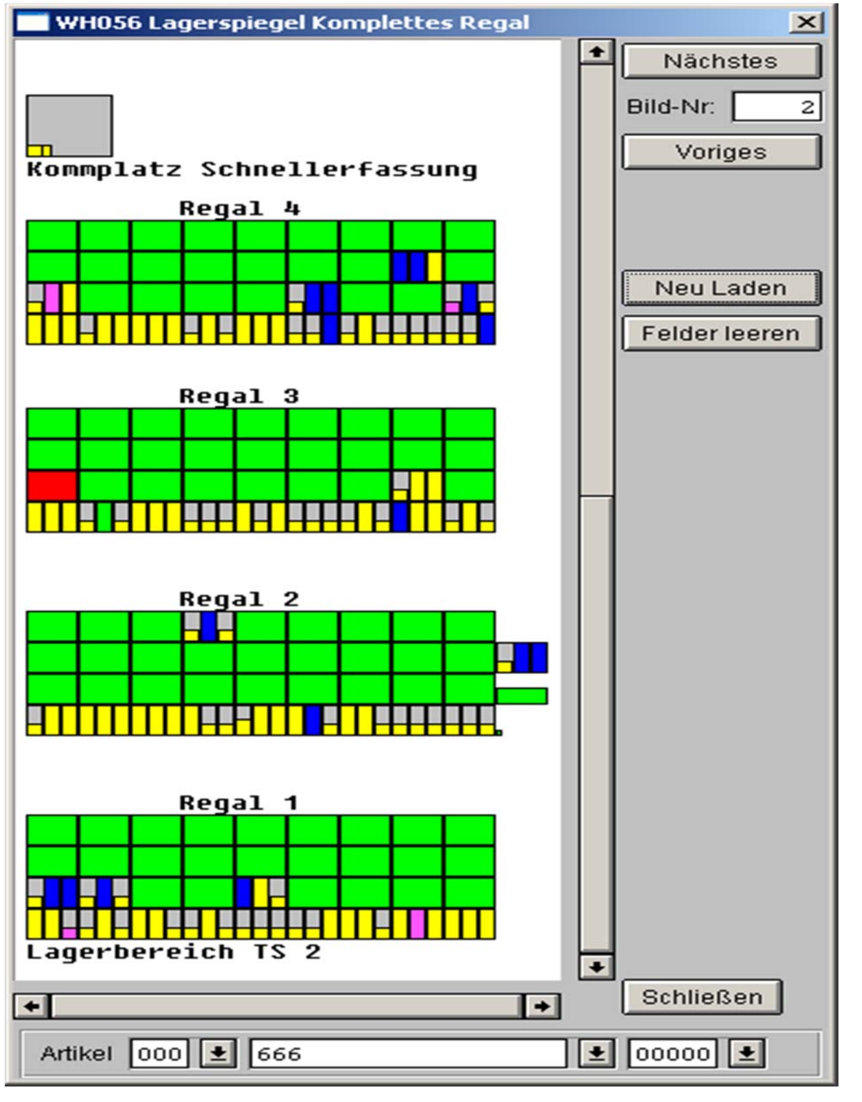


Models and meta-models



Storage visualization

- Generic software module (=reuse!)
- Visualization of storage
  - Different planes
  - Location status
- Operations on locations
  - Lock/unlock
  - Data maintenance
  - Content
  - Transport job



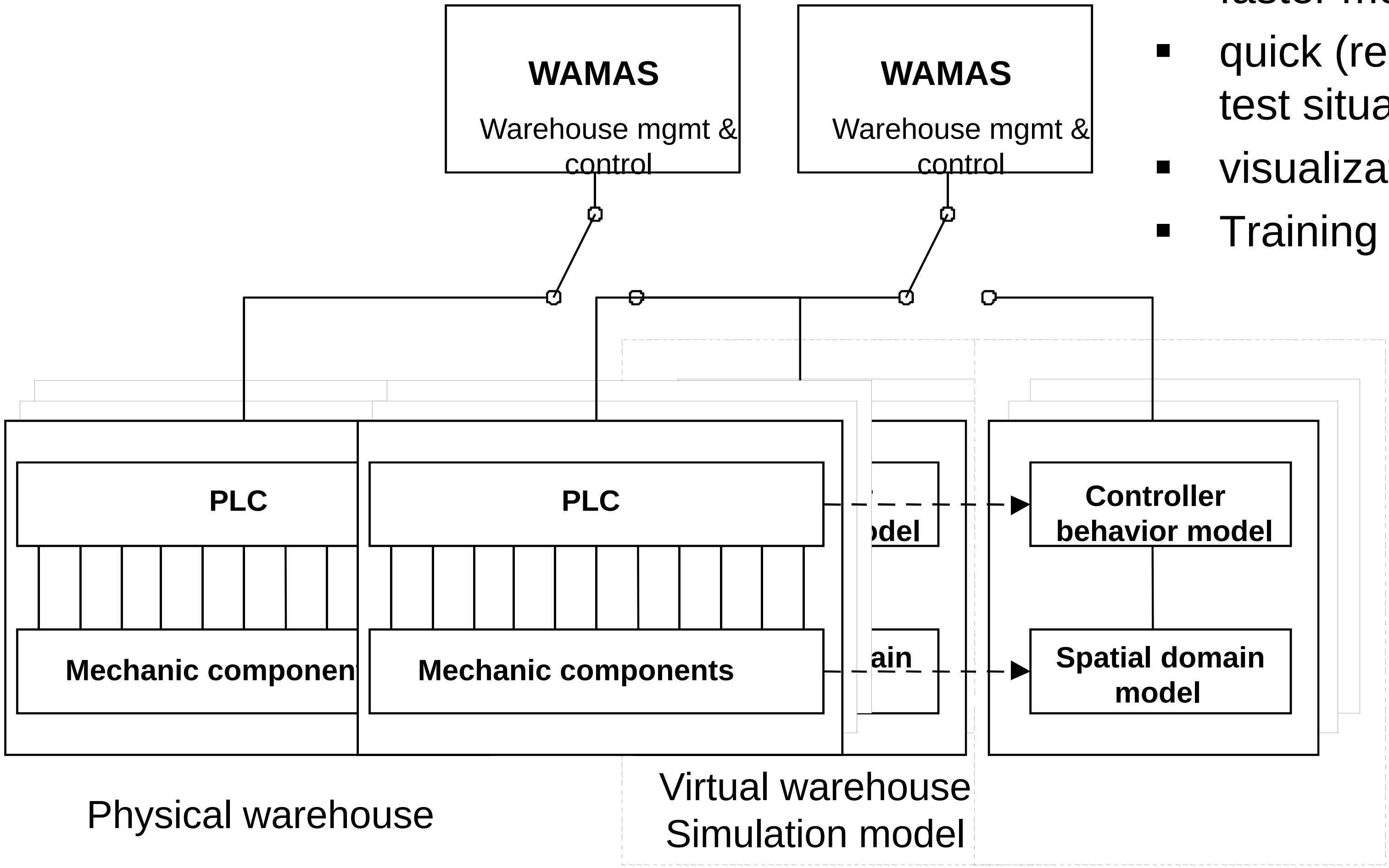
Benefits

- Customer
  - Change topology model, etc. (himself)
    - Add/remove locations, routes, types, transport facilities
- Analyst/project planning
  - Initial goods flow and storage modeling
  - Can use the language he knows
- Programmer
  - Generic storage allocation algorithm
  - Generic storage visualization
  - Generic transport routing & optimization
  - Generic goods flow & picking planning
  - „plugins“ for different warehouse types

“generic” = software reuse

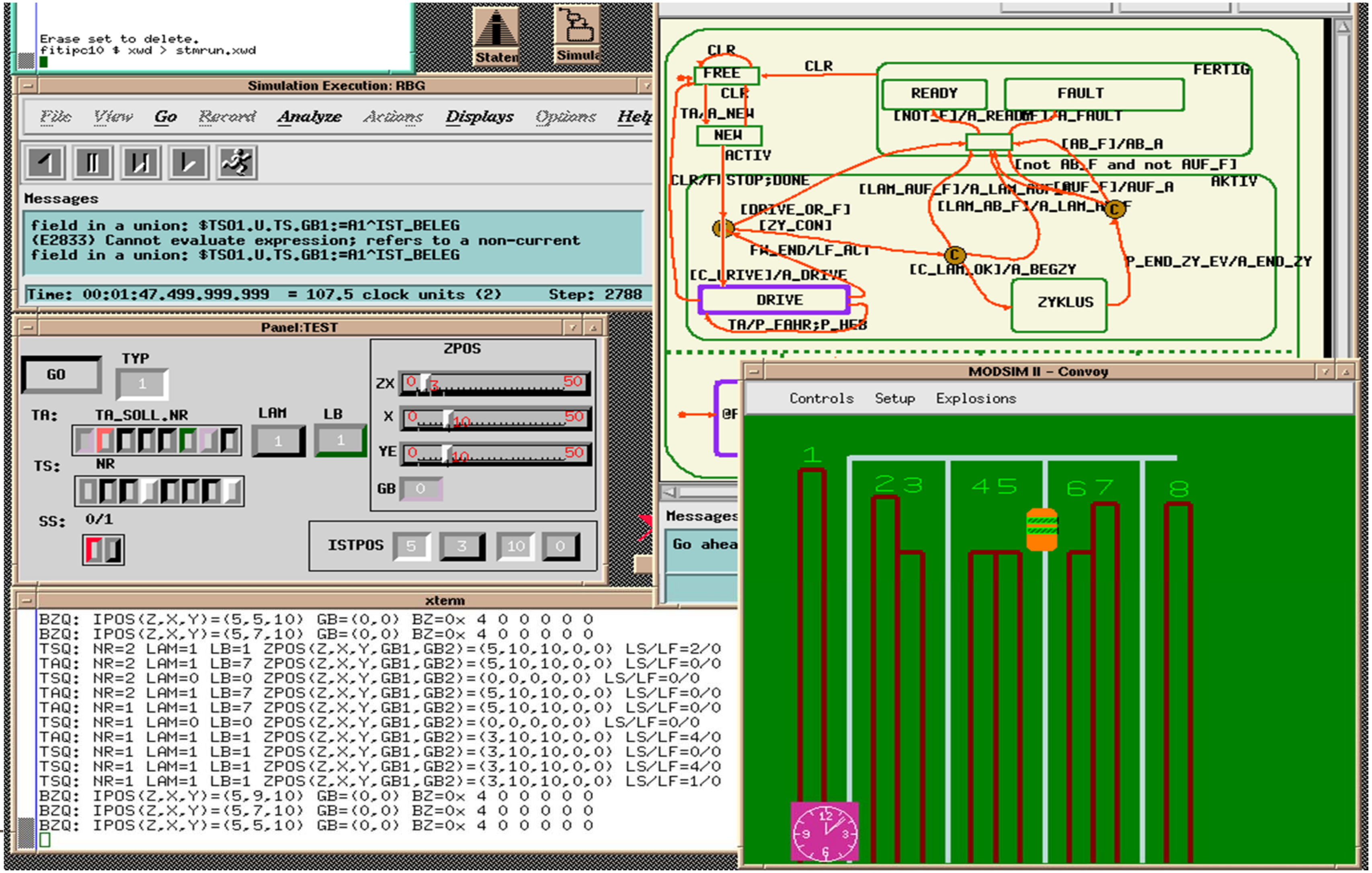
Environment simulation
Virtual warehouse

- early, in-house tests
  - cuts costs (on-site stays
  - earlier, longer tests possible
- faster movements
- quick (re-)arrangement for test situations
- visualization: better overloo
- Training



WATIS²

Environment simulation



Results

	Application Model				Components		Reuse
Group	Plant	Conveyors	Racks/Cranes	Protocol	reused	extensions	
Type	Configuration manual		Configuration generated	C-code	MODSIM+ STATEMATE		estimated %
Project							
A	120	500	13000	150	10000	100	91
B	80	0	12500	150	10000	0	97
C	50	0	60000	150	10000	80	97
D	50	400	6500	150	10000	150	93
E	20	50	0	150	10000	20	97

Table 1. Model construction efforts and estimated reuse factor (approximate number of code resp. configuration lines)

Table 1 then shows an estimated reuse factor greater than 90 % in each case.
Experience has shown that, with the use of the existing *WATIS* component library, a complete warehouse model can be configured in 1-3 days, depending on size and complexity.

Conveyors take approximately twice as long to model as racks and cranes.

Results

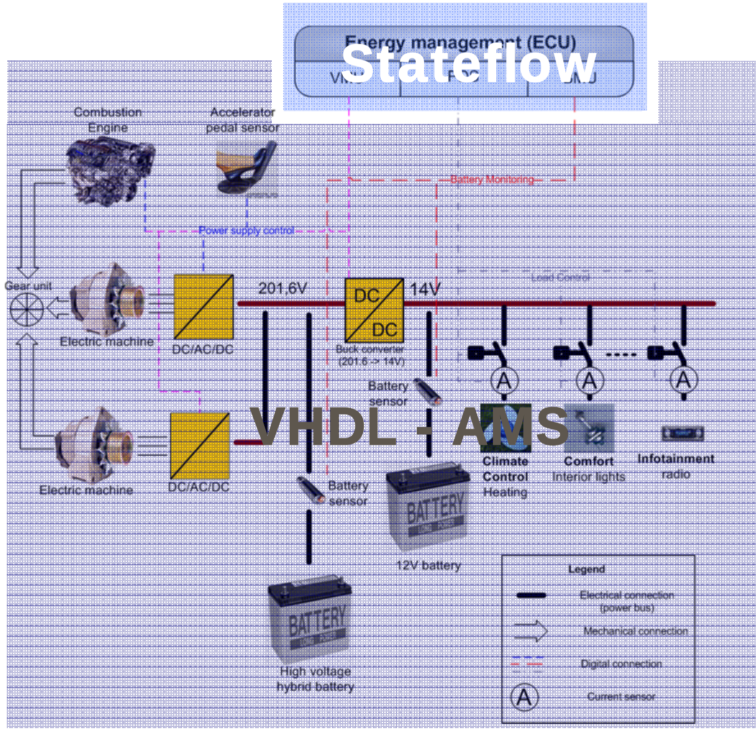
Project	Phase Duration (days) In-House Requirements + Design	Coding	Test	On-Site Installation	Support	Total	On-Site %
U	11	76	4	35	34	160	43
V	32	35	63	18	404	552	76
W	94	59	32	42	175	402	54
X	142	47	56	53	212	510	52
Y	35	86	76	143	118	458	57
Z	133	21	35	42	11	242	22
B	210	44	103	10	69	436	18
C	173	94	39	36	69	411	26
D	31	102	11	42	12	198	27

Table 2. Phase durations and on-site time of logistic software development projects

Table 2 shows a collected phase duration metrics for nine automatic logistic software projects. Tests of projects B, C, D have been supported by WATIS models, projects U to Z had no environment simulation support. Whereas most of the latter typically show on-site time around 50 % of the total project duration, all of the *WATIS* supported projects are well below 30 %, apparently due to better software maturity in the on-site phases.

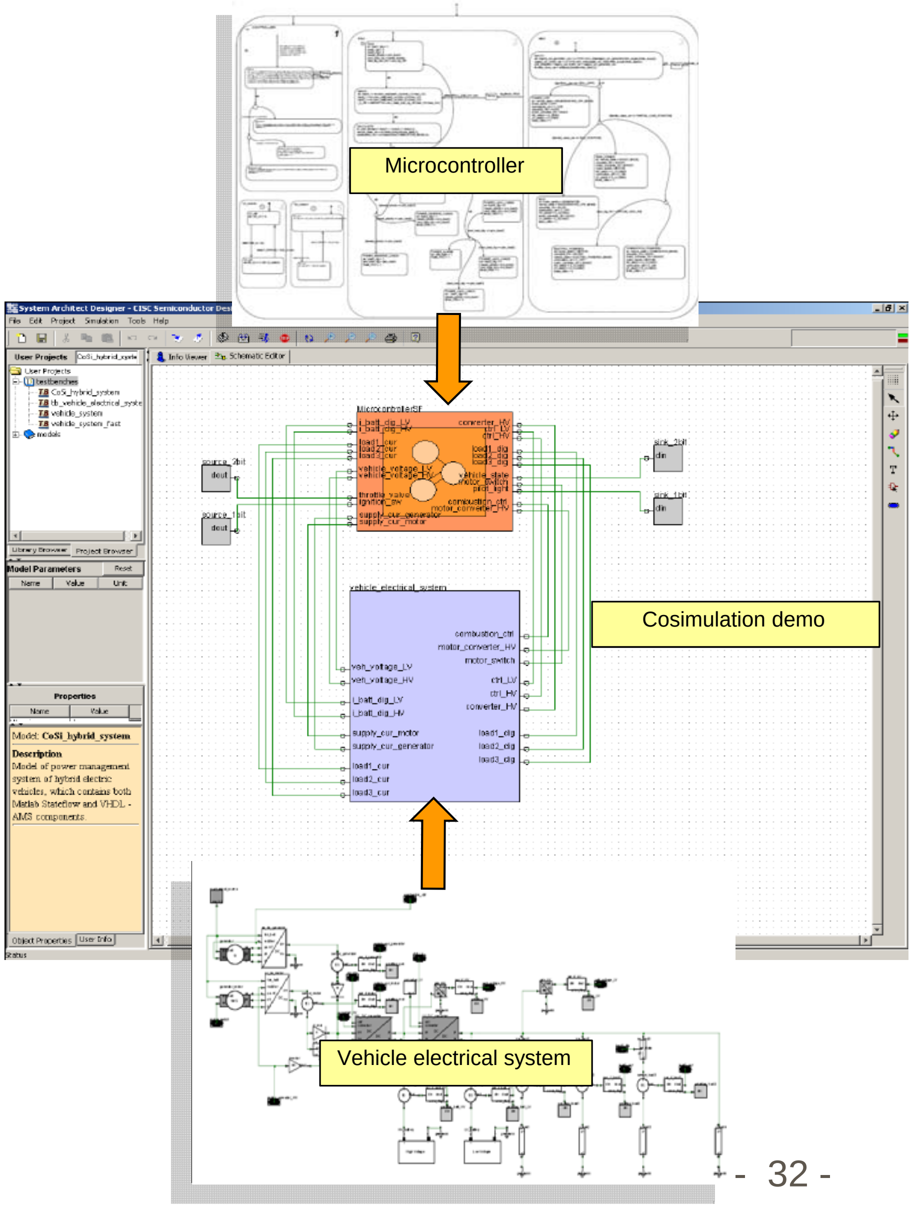
Use Case 2 - Hybrid Power Management System

- Heterogeneous closed loop consisting of two different domain specific models
  - Electronic control unit (ECU)
    - Modelled with Matlab Stateflow
    - High abstraction level
    - Stateflow eases readability
  - Vehicle electrical system
    - Modelled with VHDL-AMS
    - Detailed design with high accuracy
- Cosimulation demo of automotive domain
- Demand for a Co-simulation framework



Demo Presentation

- The Hybrid Power Management demo CoSi_hybrid_system can be found in the “Project Browser” tab by selecting “User Projects” and expanding the “testbenches” folder.
  - Microcontroller
    - Electronic control unit
    - Matlab Stateflow
  - Vehicle electrical system
    - Hierarchical component containing battery, generator,…
    - VHDL-AMS



Specifying timing in spec. languages

4 types of timing specs required [Burns, 1990]:

- **Measure elapsed time**
  Check, how much time has elapsed since last call
- **Means for delaying processes (e.g., wait in VHDL)**
- **Possibility to specify timeouts**
  We would like to be in a certain state only a certain maximum amount of time.
- **Methods for specifying deadlines**
  With current languages not available or specified in separate control file.

☞ **StateCharts comprises a mechanism for specifying timeouts. Other types of timing specs are not supported.**

Concurrency vs. Parallism

- Concurrency is central to embedded systems. A computer program is said to be concurrent if different parts of the program conceptually execute simultaneously.

- A program is said to be parallel if different parts of the program physically execute simultaneously on distinct hardware (multi-core, multi-processor or distributed systems)

Petri Nets

Petri nets

Introduced in 1962 by Carl Adam Petri in his PhD thesis.

Different “Types” of Petri nets known

- Condition/event nets
- Place/transition nets
- Predicate/transition nets
- Hierachical Petri nets,
- Timed Petri nets
- …

theoretical

Mathematics

study interesting,
consistent structures

theory ? practice

Computer Science
??

Engineering

build practicable,
useful structures

experimental

Physics

predict & measure
"real world" structures

idealised ***pragmatic***

Simple models of complex worlds

in models of existing systems, *automata imply an approximation*.

(simpler, but applicable only if their assumptions hold)

in designs of new systems, *automata involve over-specification*!

(engineers have to implement the assumptions!)

Models can be unrealistic if they are too simple, and simplifying designs are harder to realise!

Petri's nets - complex foundations for simple models

For his nets, Carl Adam Petri has made an attempt to combine automata from theoretical CS, insights from physics, and pragmatic expertise from engineers:

- *state is distributed, transitions are localised (space is relevant)*
- *local causality replaces global time (time as a derived concept)*
- *subsystems interact by explicit communication*
  (information transport is as relevant as information processing)

engineers can often ignore the background - Petri nets just work!

(but the background explains why things work, why concepts from other disciplines, such as logic, have been integrated into Petri nets so easily, and why foundational research has to continue)

Application areas

- modelling, analysis, verification of distributed systems
- automation engineering
- business processes
- modeling of resources
- modeling of synchronization

Key Elements

- **Conditions**
  Either met or not met. Conditions represent “local states”. Set of conditions describes the potential state space.
- **Events**
  May take place if certain conditions are met. Event represents a state transition.
- **Flow relation**
  Relates conditions and events, describes how an event changes the local and global state.
- **Tokens**
  Assignments of tokens to conditions specifies a global state.

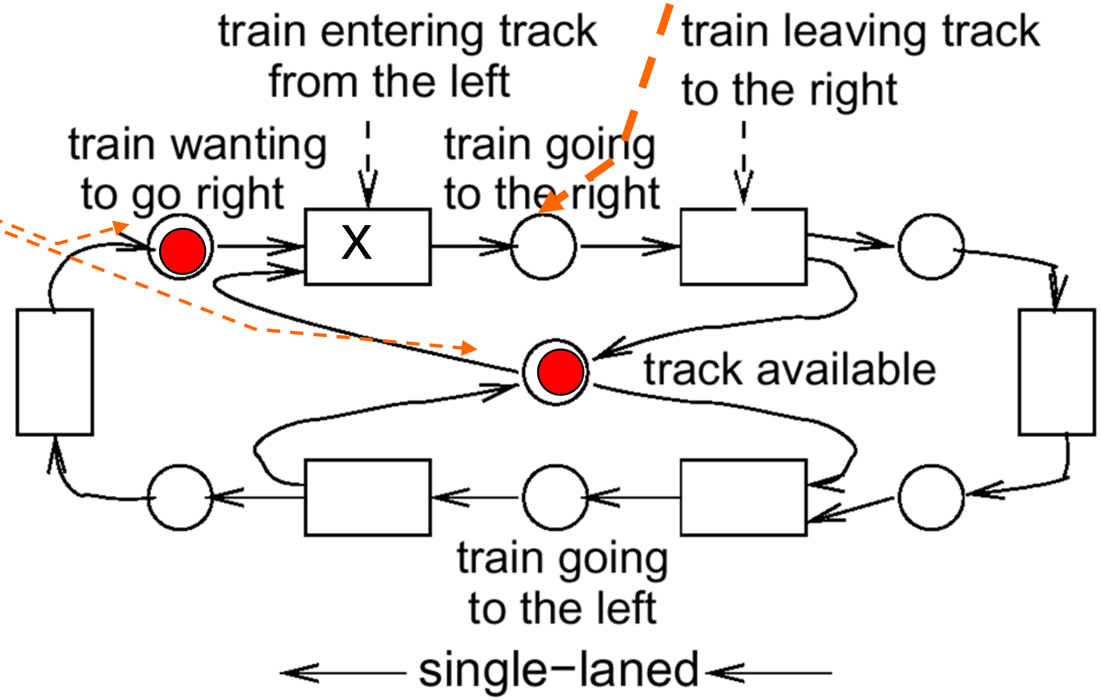
Conditions, events and the flow relation form
a **bipartite graph** (graph with two kinds of nodes).

Example 2:
Synchronization at single track rail segment

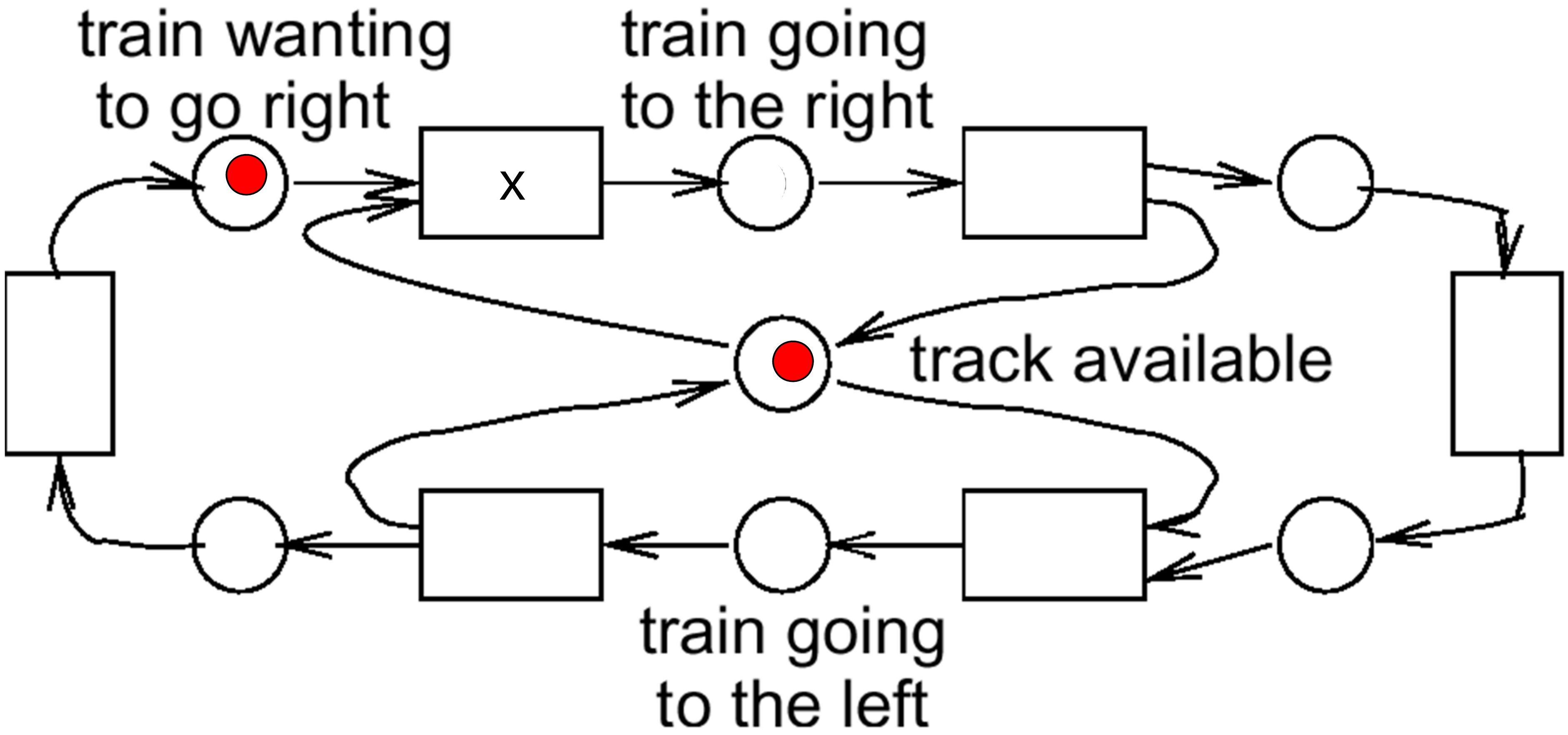
- **mutual exclusion:**
  there is at most one train using the track rail

„Postcondition“
of x fulfilled

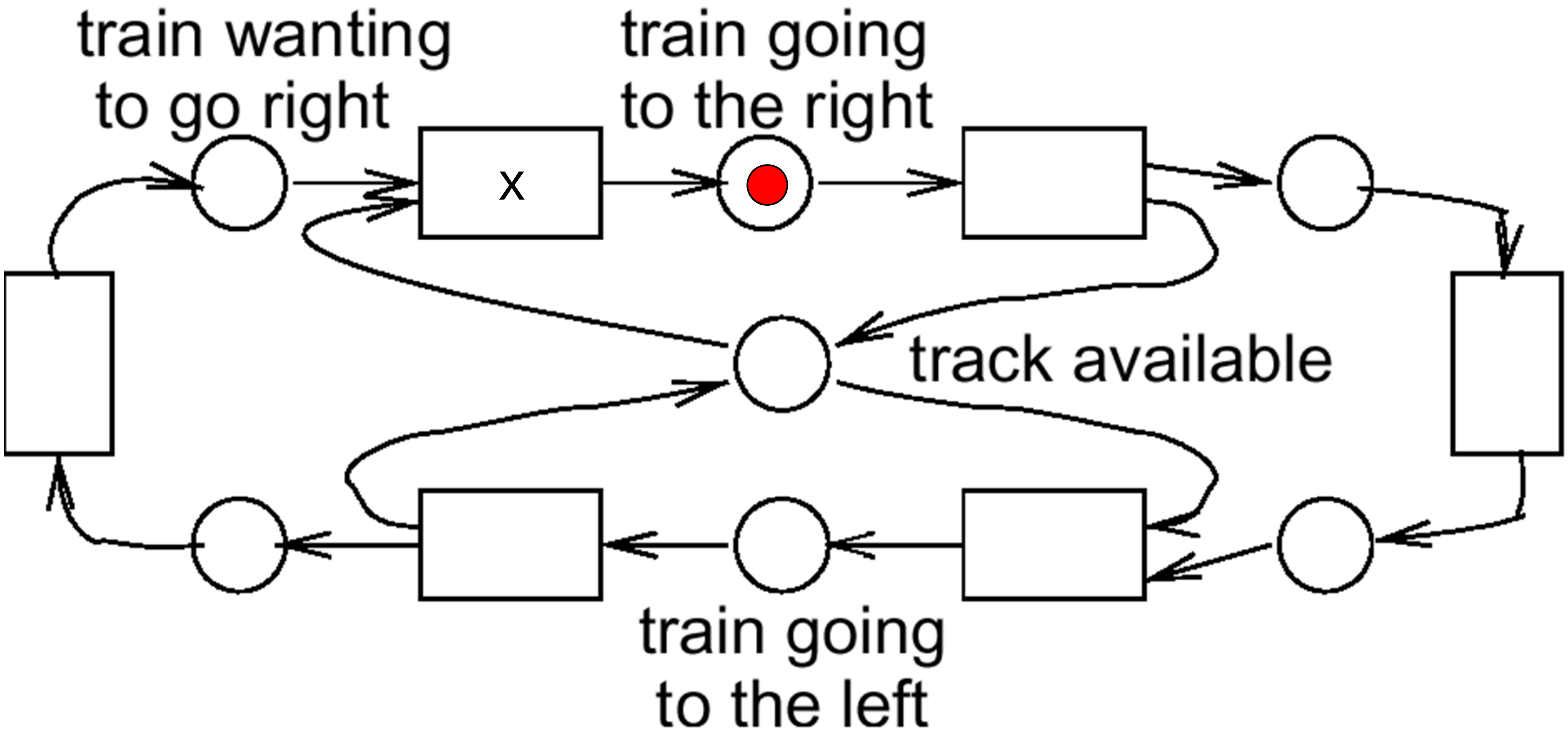
„Preconditions“
of x fulfilled



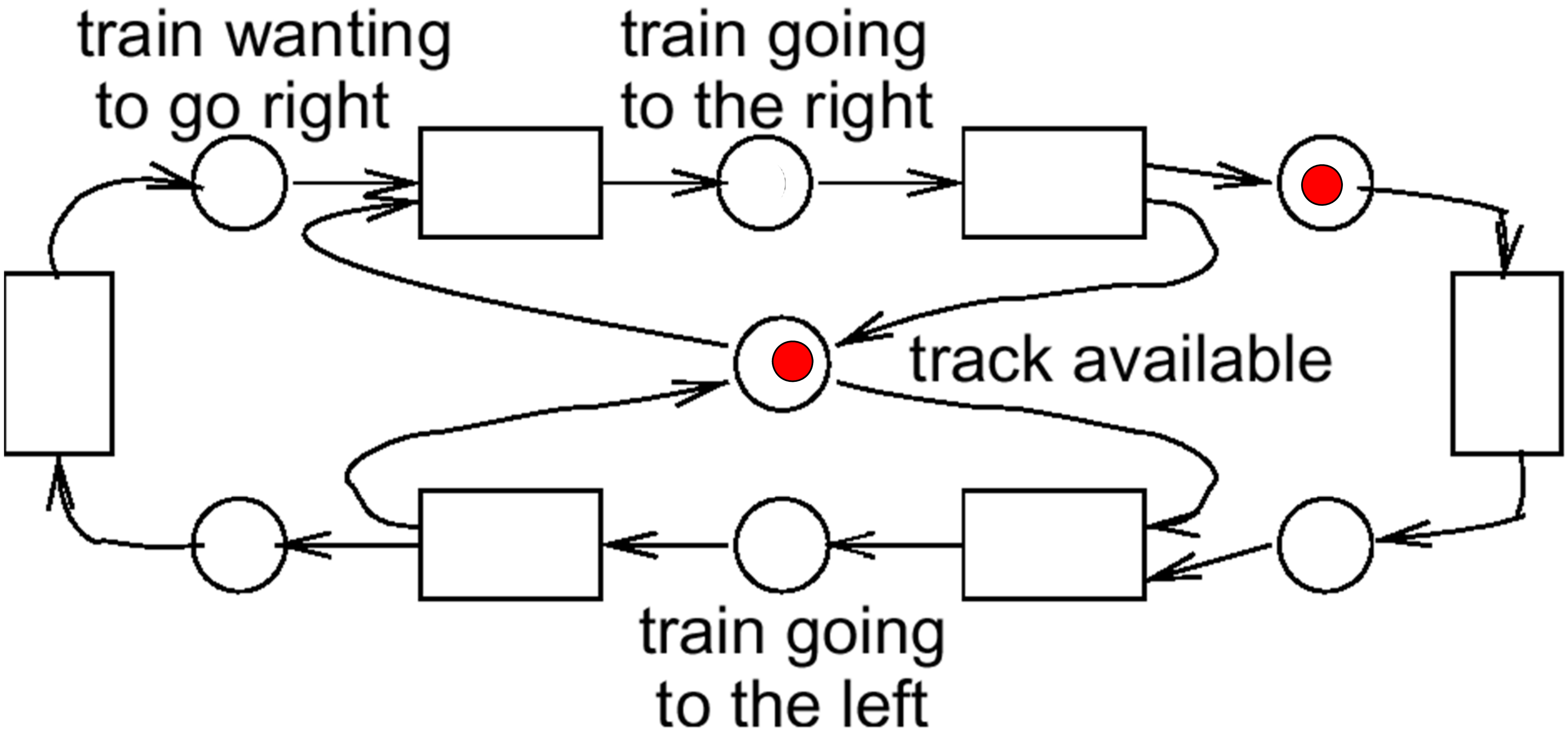
Playing the „token game": dynamic behavior



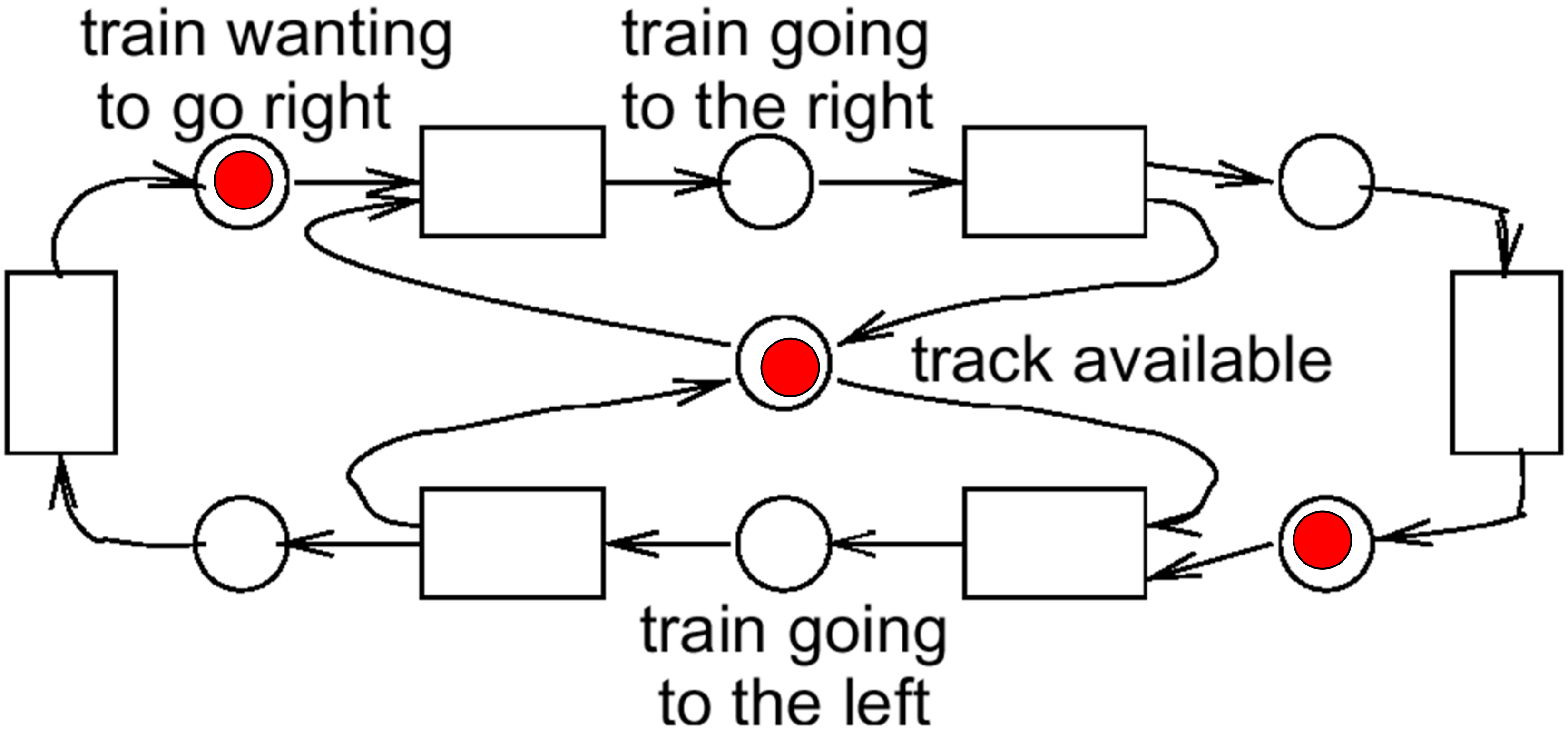
Playing the „token game“: dynamic behavior



Playing the „token game“: dynamic behavior



Conflict for resource „track“: two trains competing



Condition/event Petri nets

single token per place

Def.: $N=(C,E,F)$ is called a **net**, iff the following holds

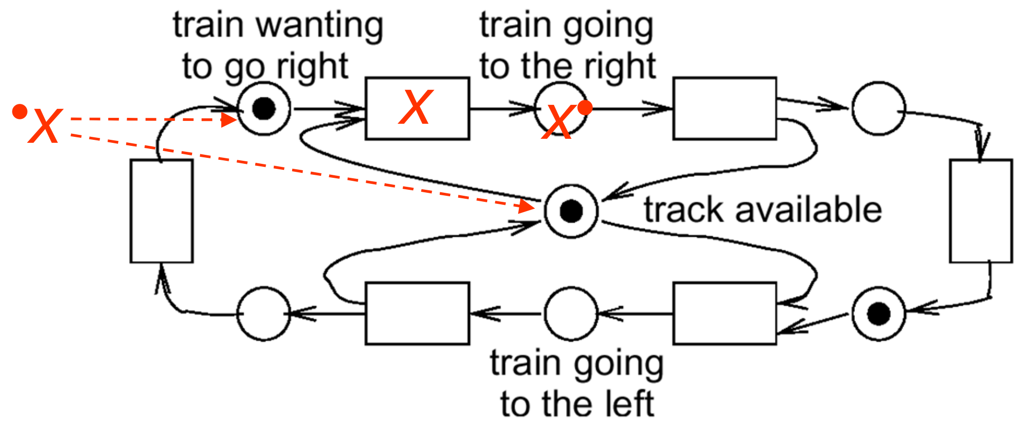
1. C and E are disjoint sets
2. $F \subseteq (C \times E) \cup (E \times C)$; is binary relation, („**flow relation**“)

Def.: Let N be a net and let $x \in (C \cup E)$.

$^\bullet x := \{y \mid y\ F\ x\}$ is called the set of **preconditions.**

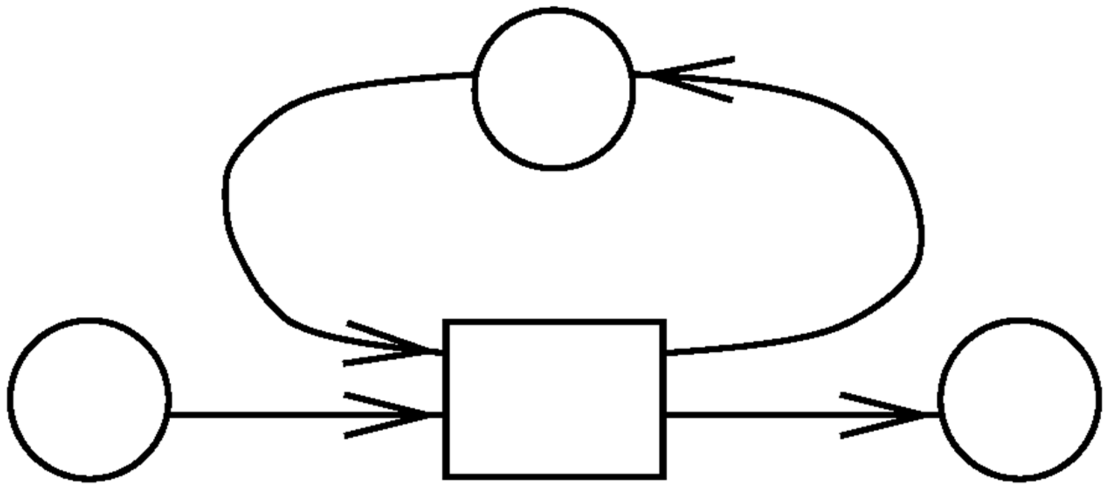
$x^\bullet := \{y \mid x\ F\ y\}$ is called the set of **postconditions.**

Example:



Basic structural properties: Loops and pure nets

Def.: Let $(c,e) \in C \times E$. (c,e) is called a **loop** iff $cFe \wedge eFc$.

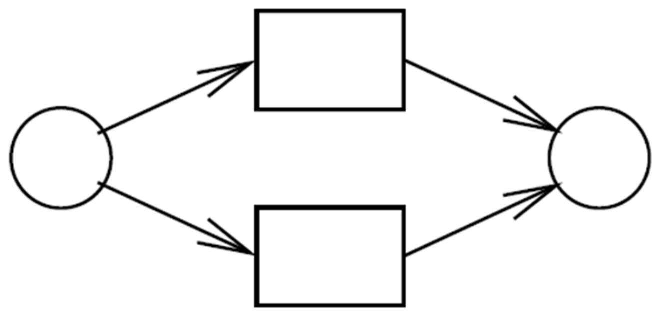


Def.: Net $N=(C,E,F)$ is called **pure**, if F does not contain any loops.

Simple nets

- **Def.:** A net is called **simple** if no two nodes $n1$ and $n2$ have the same pre-set and post-set.

- Example (not simple):

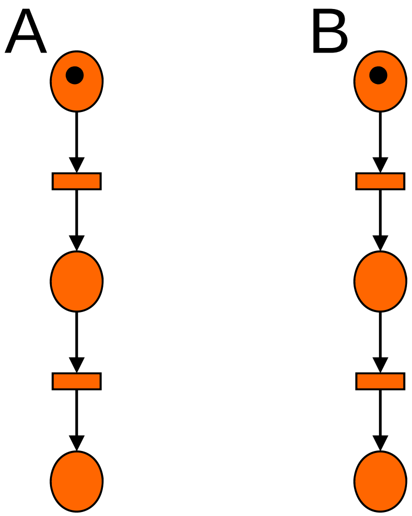


Properties of C/E

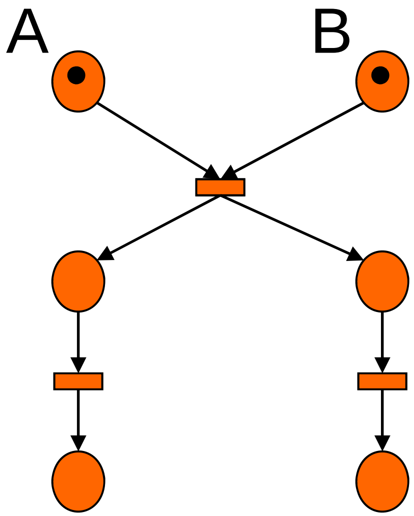
Def.:

- Marking M’ is **reachable** from marking M, iff there exists sequence of firing steps transforming M into M’ (Not.: M➲M’)
- A C/E net is **cyclic**, iff any two markings are reachable from each other.
- A C/E net fulfills **liveness,** iff for each marking M and for each event e there exists a reachable marking M’ that activates e for firing

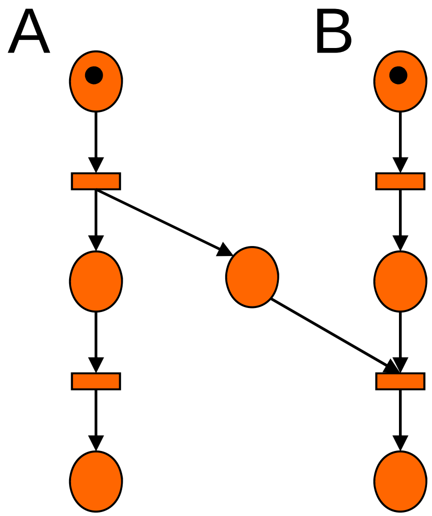
Basic examples



concurrency



synchronisation



communication

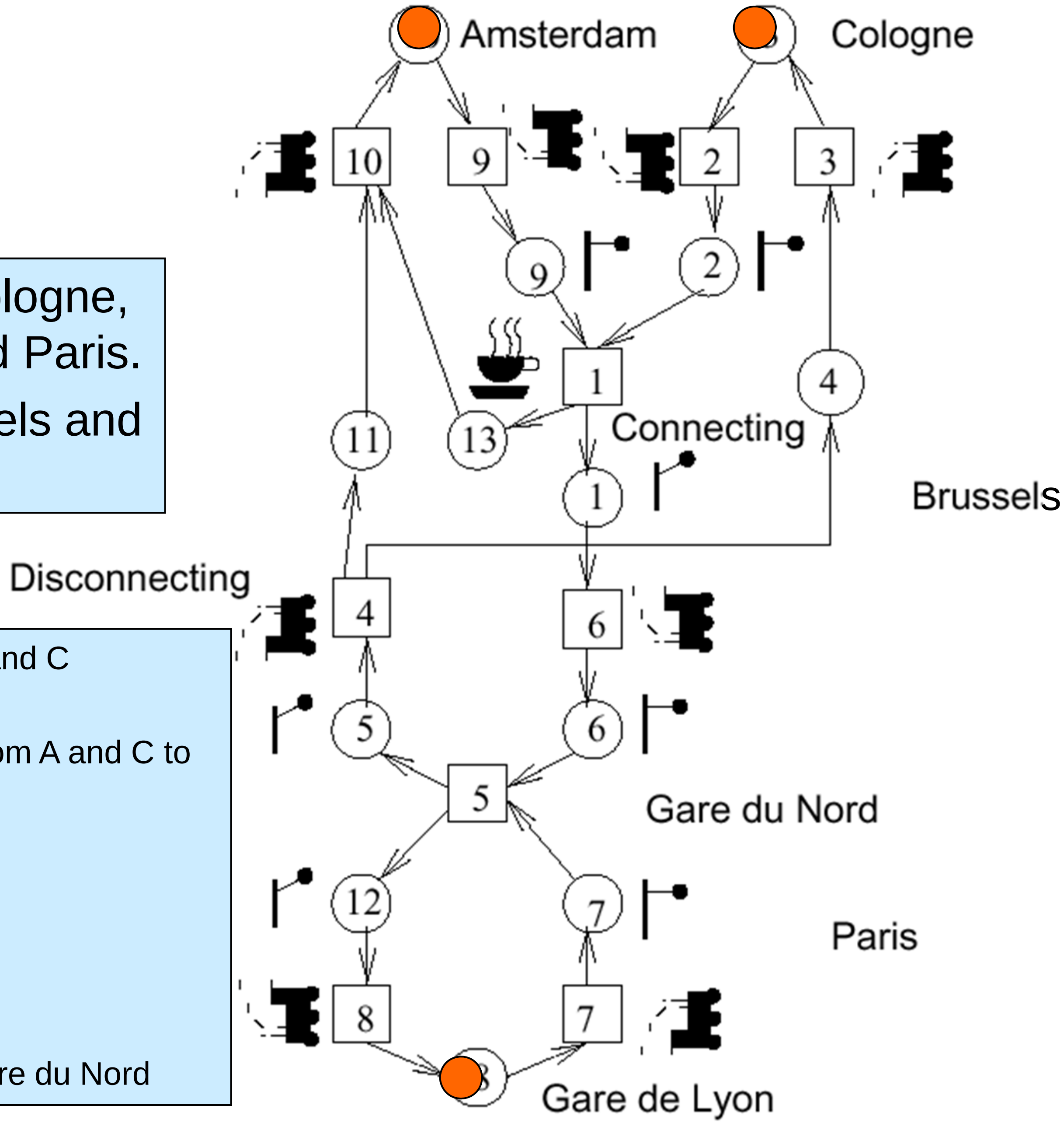
More complex example (1)

Thalys trains between Cologne, Amsterdam, Brussels and Paris.

[http://www.thalys.com/be/en]

Example Thalys trains: more complex

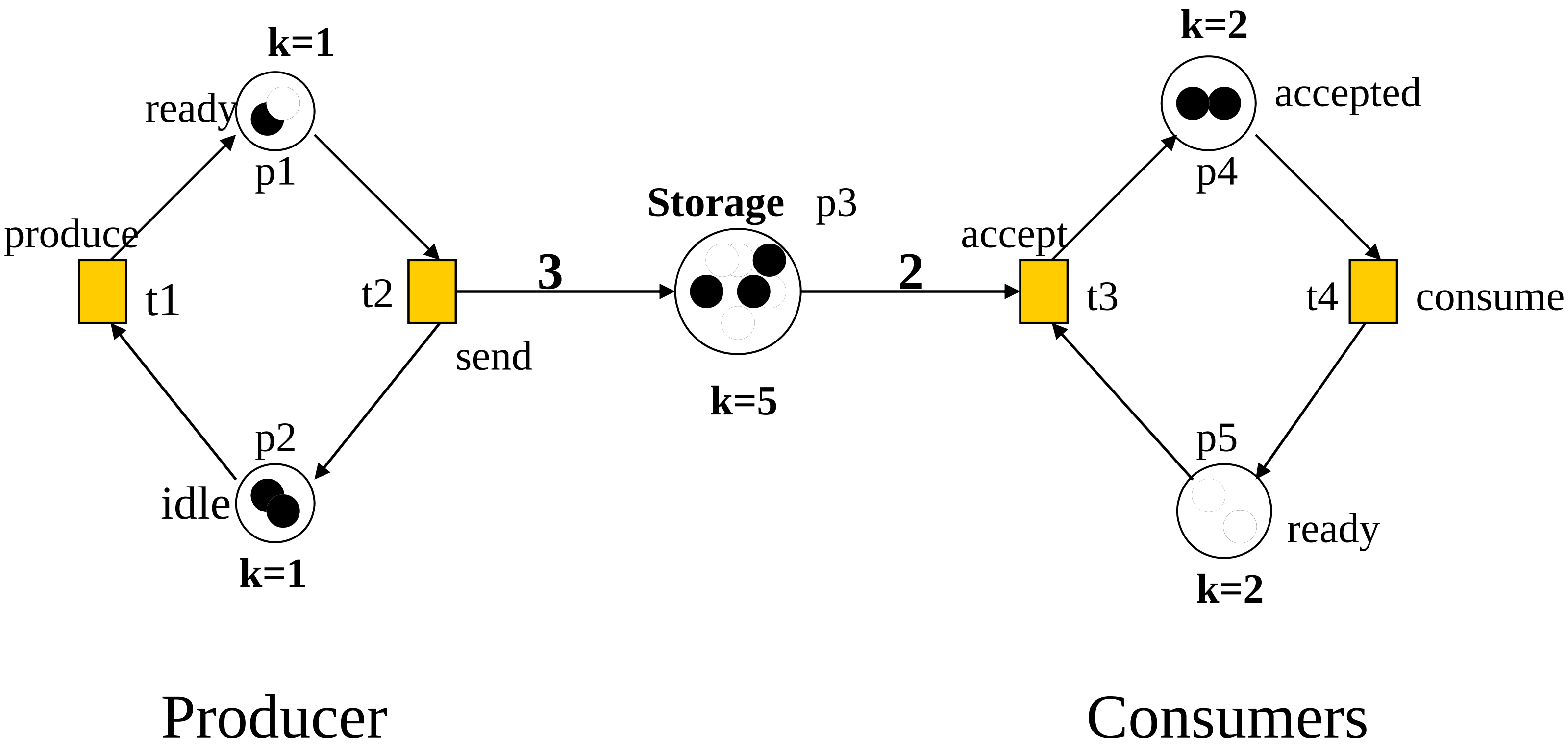
- Thalys trains between Cologne, Amsterdam, Brussels and Paris.
- Synchronization at Brussels and Paris



- Places 3 and10: trains waiting in A and C
- Transitions 9 and 2: trains driving from A and C to Brussels
- T1: connecting the two trains
- Break for driver P13
- T5 synchronization with trians at Gare du Nord

Realistic scenarios need more general definitions

- More than one token per condition, capacities of places
- weights of edges
- state space of Petri nets may become infinite!

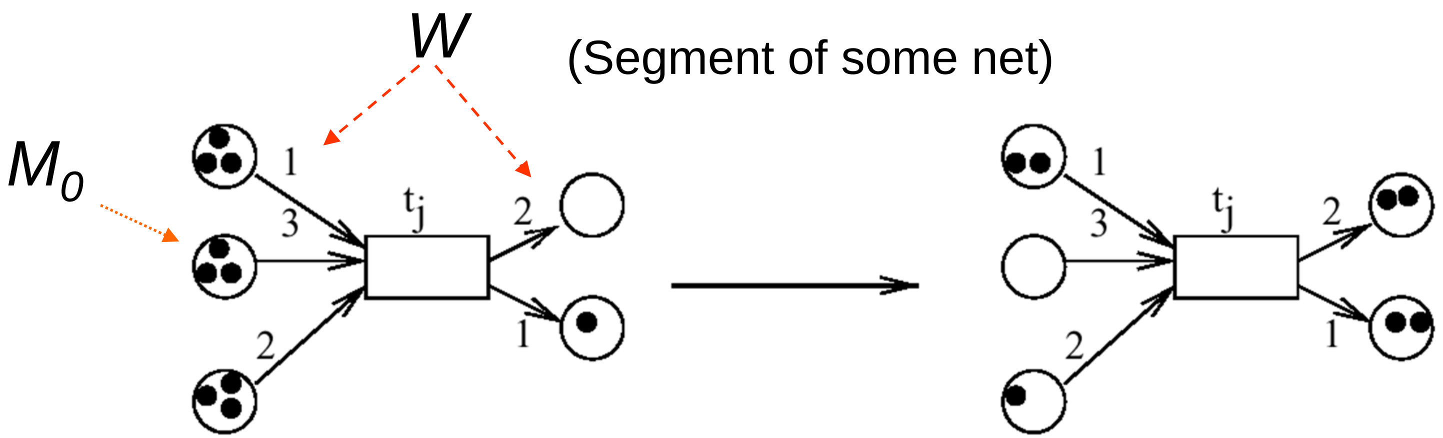


Place/transition nets

multiple tokens per place

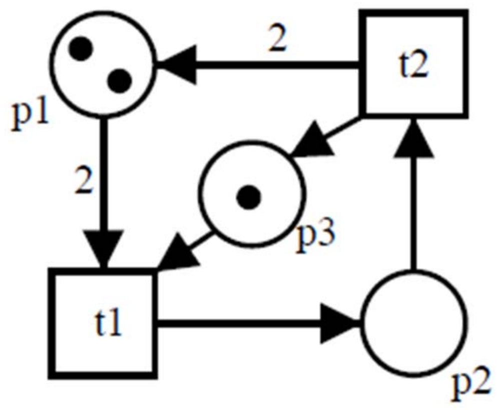
Def.: (P, T, F, K, W, M_0) is called a **place/transition net (P/T net)** iff

1. $N=(P,T,F)$ is a **net** with places $p \in P$ and transitions $t \in T$
2. $K: P \rightarrow (\mathbf{N_0} \cup \{\omega\}) \setminus \{0\}$ denotes the **capacity** of places
   (ω symbolizes infinite capacity)
3. $W: F \rightarrow (\mathbf{N_0} \setminus \{0\})$ denotes the **weight of graph edges**
4. $M_0: P \rightarrow \mathbf{N_0} \cup \{\omega\}$ represents the **initial marking** of places



defaults:
$K = \omega$
$W = 1$

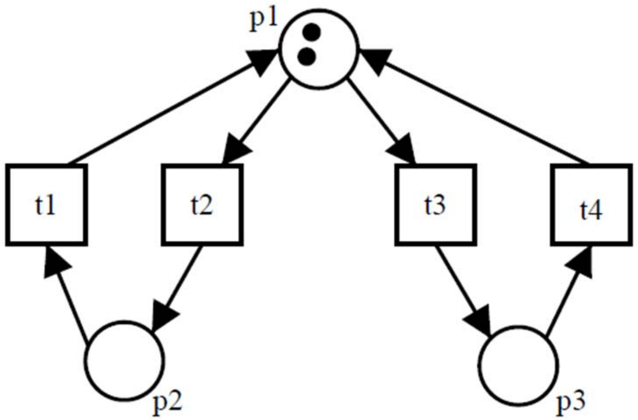
Example



- P = {p1, p2, p3}
- T = {t1, t2}
- F = {(p1, t1), (p2, t2), (p3, t1), (t1, p2), (t2, p1), (t2, p3)}
- W = {(p1, t1) → 2, (p2, t2) → 1, (p3, t1) → 1, (t1, p2) → 1,
  (t2, p1) → 2, (t2, p3) → 1}
- m0 = (2, 0, 1)

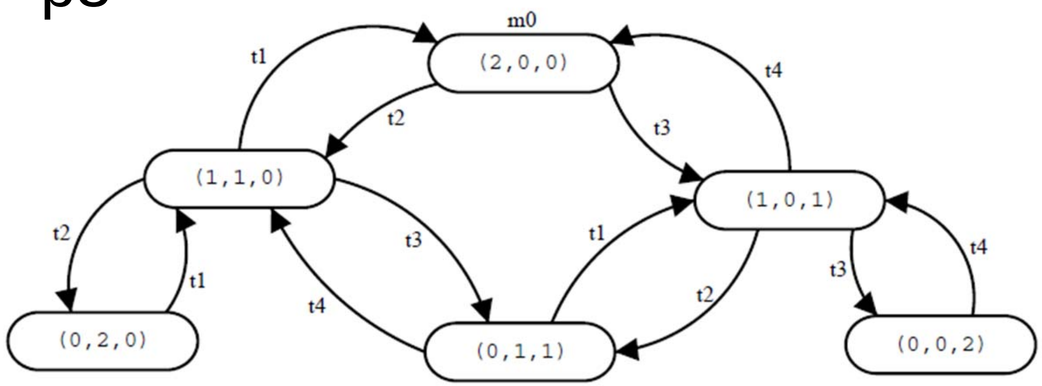
  p1 p2 p3

Reachability



m0 = (2, 0, 0)

p1 p2 p3



reachability graph

From conditions to resources (1)

- c/e-systems model the flow of information, at a fundamental level (true/false)

- there are natural application areas for which the flow/transport of resources and the number of available resources is important (data flow, document-/workflow, production lines, communication networks, www, ..)

- place/transition-nets are a suitable generalisation of c/e-systems:
  - state elements represent places where resources (tokens) can be stored
  - transition elements represent local transitions or transport of resources

From conditions to resources (2)

- a transition is enabled if and only if
  - sufficient resources are available on all its input places
  - sufficient capacities are available on all its output places

- a transition occurrence
  - consumes one token from each input place and
  - produces one token on each output place

Specifications