# StateCharts

Peter Marwedel
Informatik 12
TU Dortmund
Germany

# StateCharts

Used here as a (prominent) example of a model of computation based on shared memory communication.

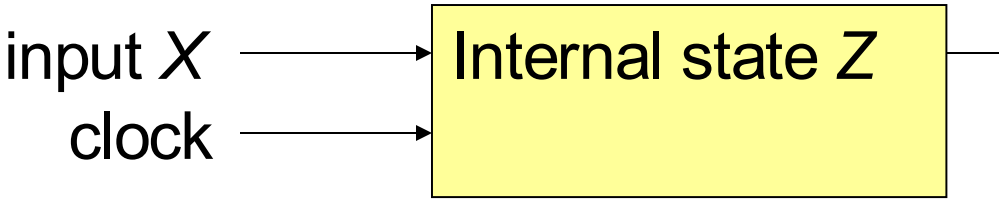
☞ appropriate only for local (non-distributed) systems

# Models considered in this course

| Communication/ local computations | Shared memory | Message passing Synchronous \| Asynchronous | |
|---|---|---|---|
| Communicating finite state machines | StateCharts | | SDL |
| Data flow model | | | Kahn process networks |
| Von Neumann model | C, C++, Java | C, C++, Java with libraries CSP, ADA \| | |
| Discrete event (DE) model | VHDL, Verilog, SystemC | Only experimental systems, e.g. distributed DE in Ptolemy | |

# StateCharts: recap of classical automata

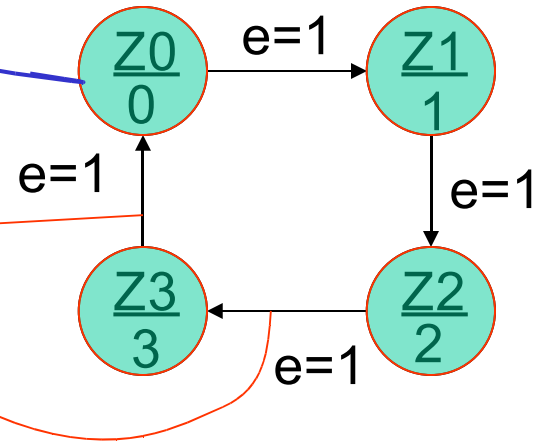
Classical automata:




Next state $Z^+$ computed by function $\delta$
Output computed by function $\lambda$

Moore- + Mealy automata=finite state machines (FSMs)

- Moore-automata:
  $Y = \lambda(Z)$; $Z^+ = \delta(X, Z)$
- Mealy-automata
  $Y = \lambda(X, Z)$; $Z^+ = \delta(X, Z)$



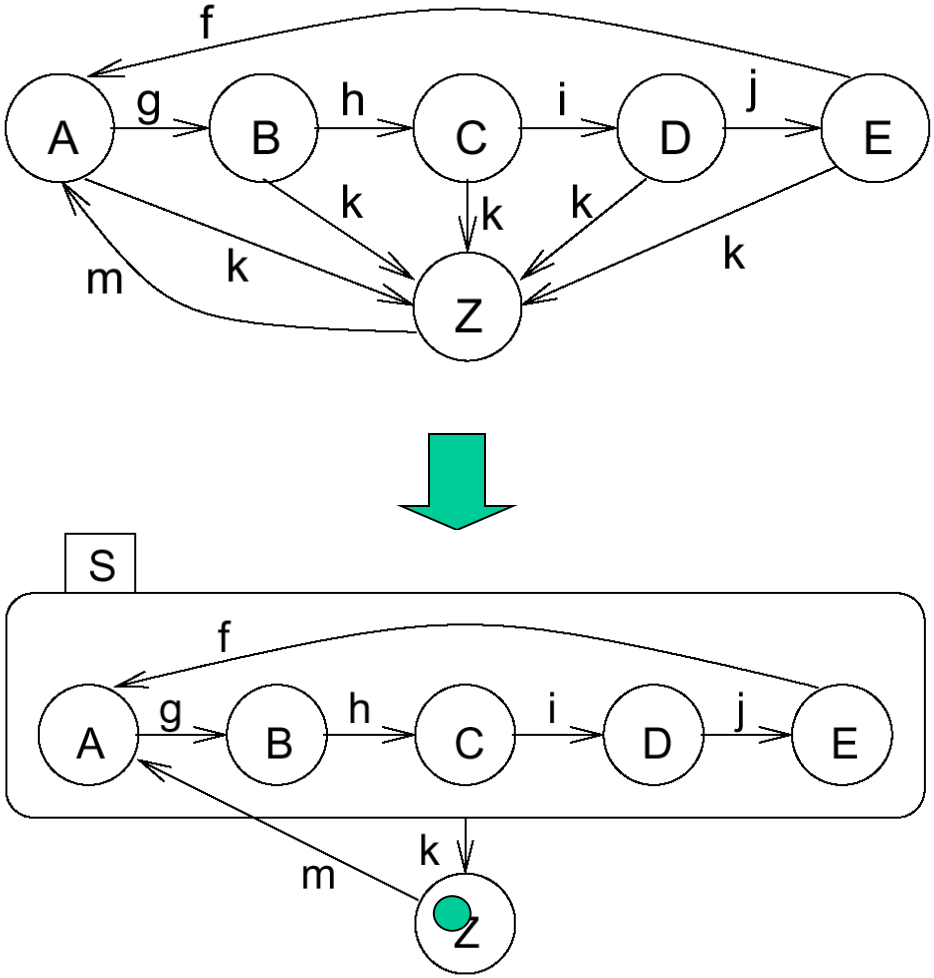
# StateCharts

Classical automata not useful for complex systems (complex graphs cannot be understood by humans).

☞ Introduction of hierarchy ☞ StateCharts [Harel, 1987]

StateChart = *the only unused combination of „flow“ or „state“ with „diagram“ or „chart“*

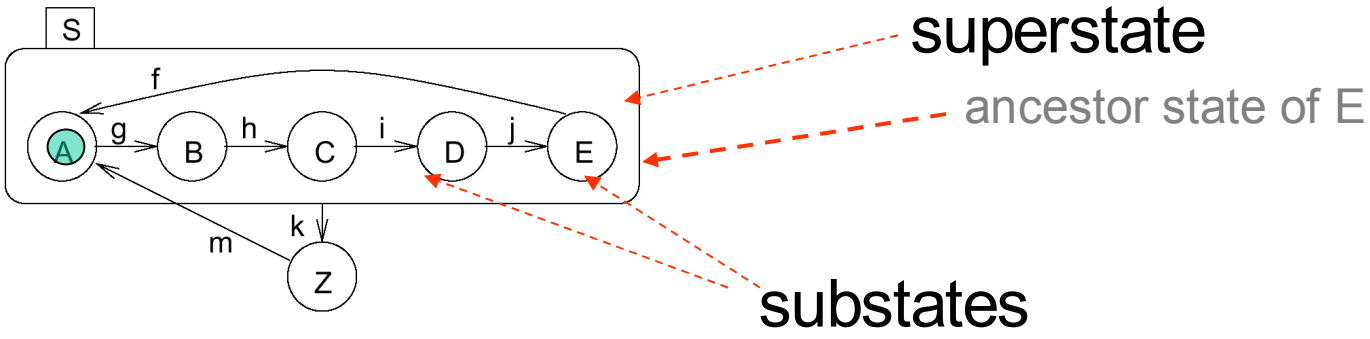
# Introducing hierarchy




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

# Definitions

- Current states of FSMs are also called **active** states.
- States which are not composed of other states are called **basic states.**
- States containing other states are called **super-states**.
- For each basic state s, the super-states containing s are called **ancestor states**.
- Super-states S are called **OR-super-states**, if exactly one of the sub-states of S is active whenever S is active.



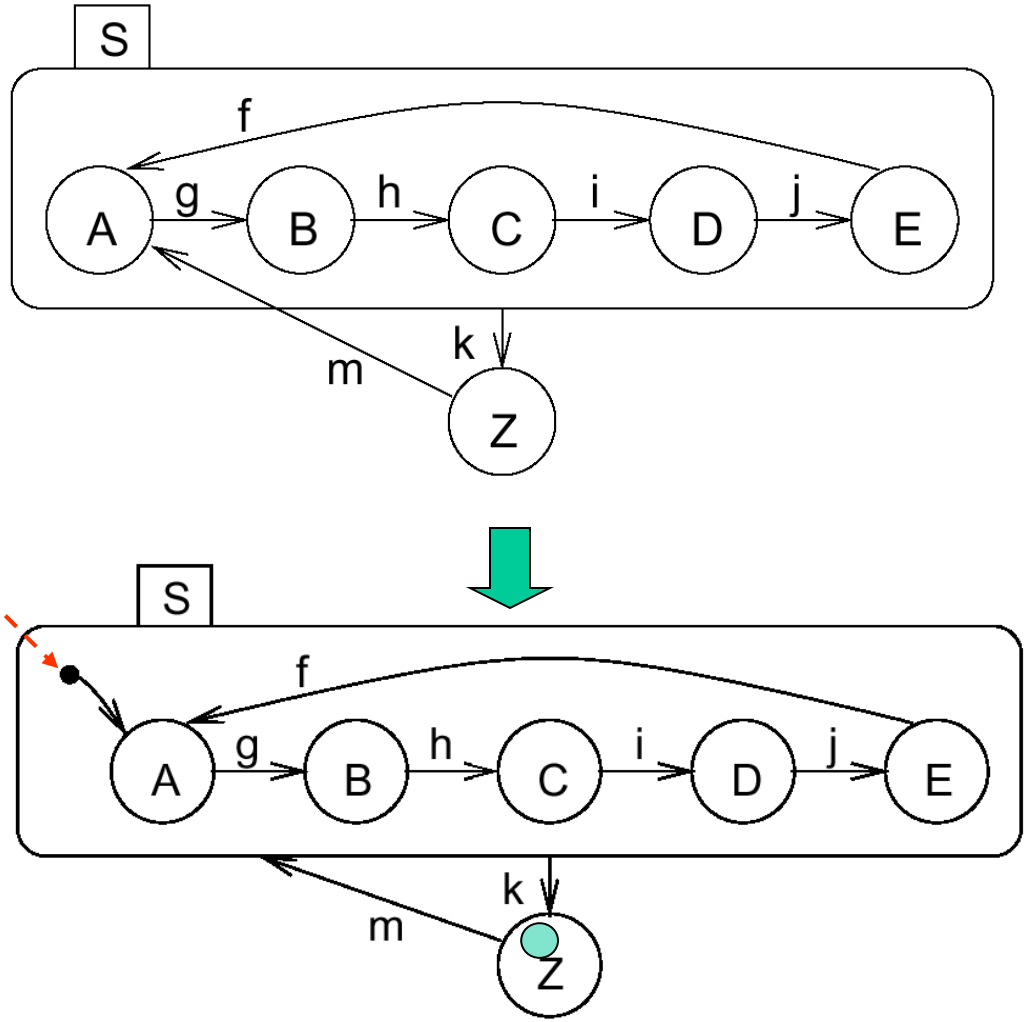
# Default state mechanism

Try to hide internal structure from outside world!

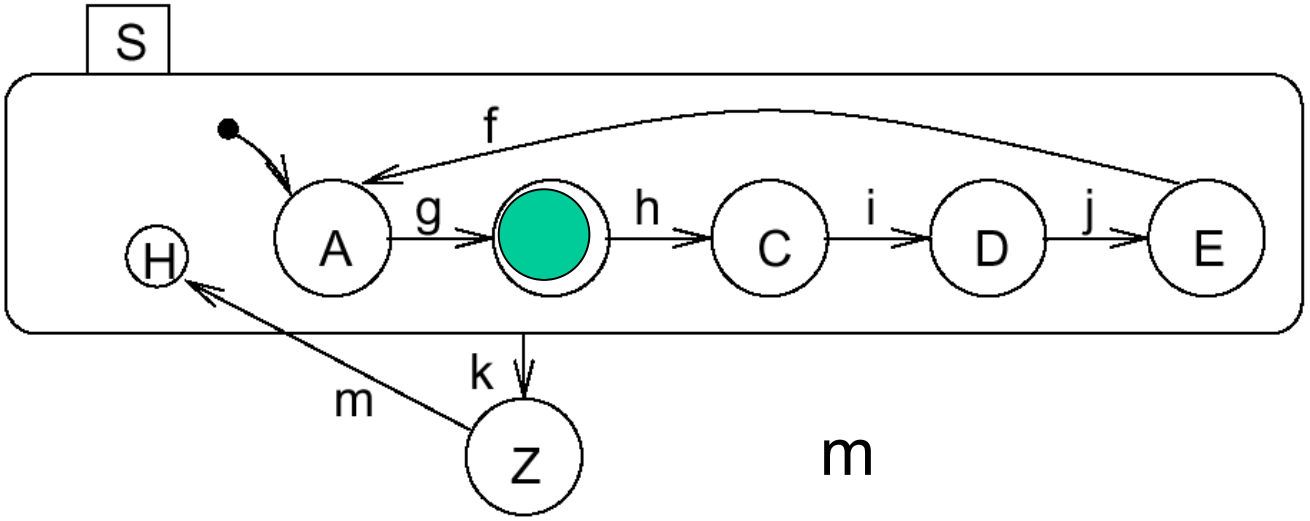
☞ Default state

Filled circle indicates sub-state entered whenever super-state is entered.

Not a state by itself!



# History mechanism



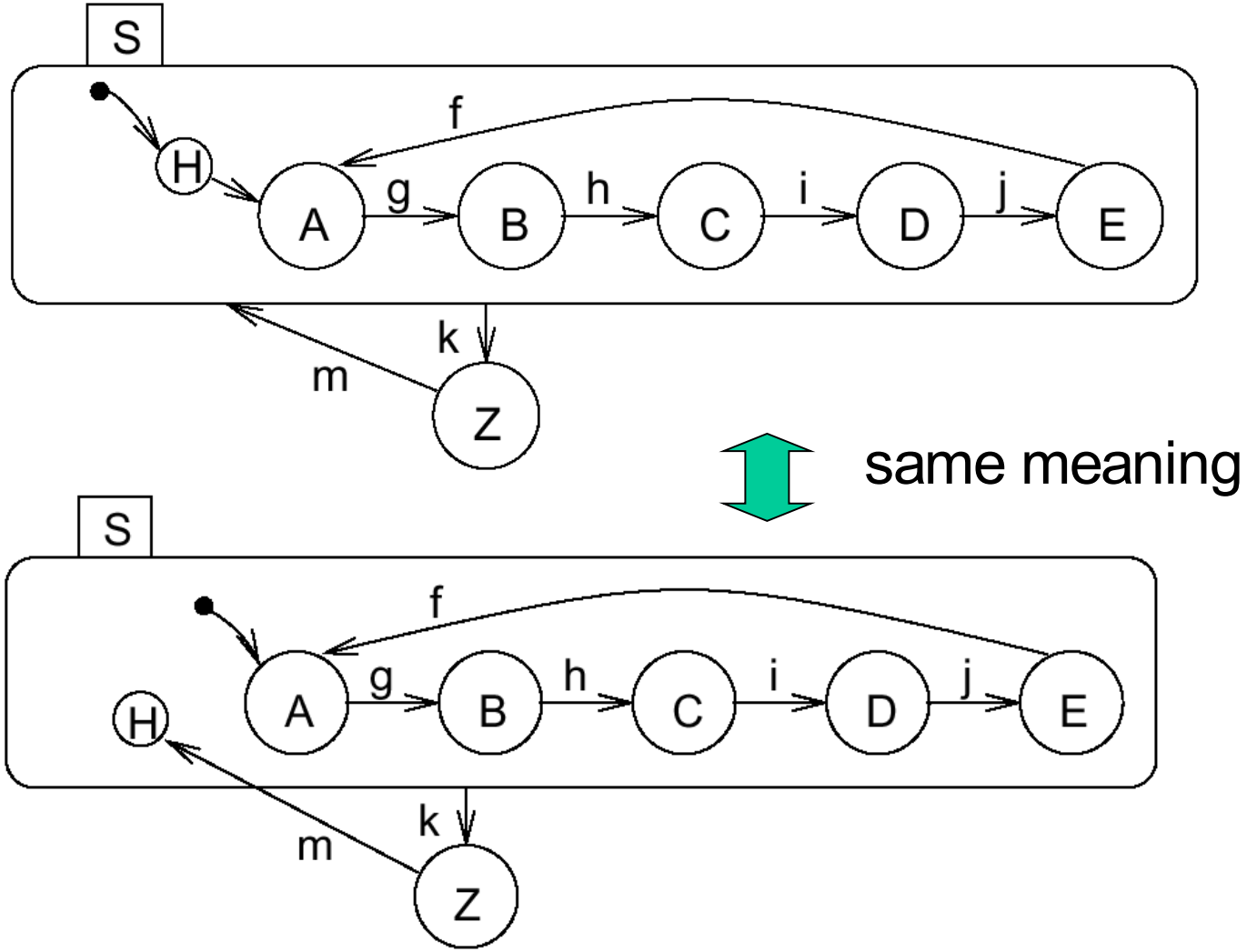

(behavior different from last slide)

For input 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 first time, the default mechanism applies.

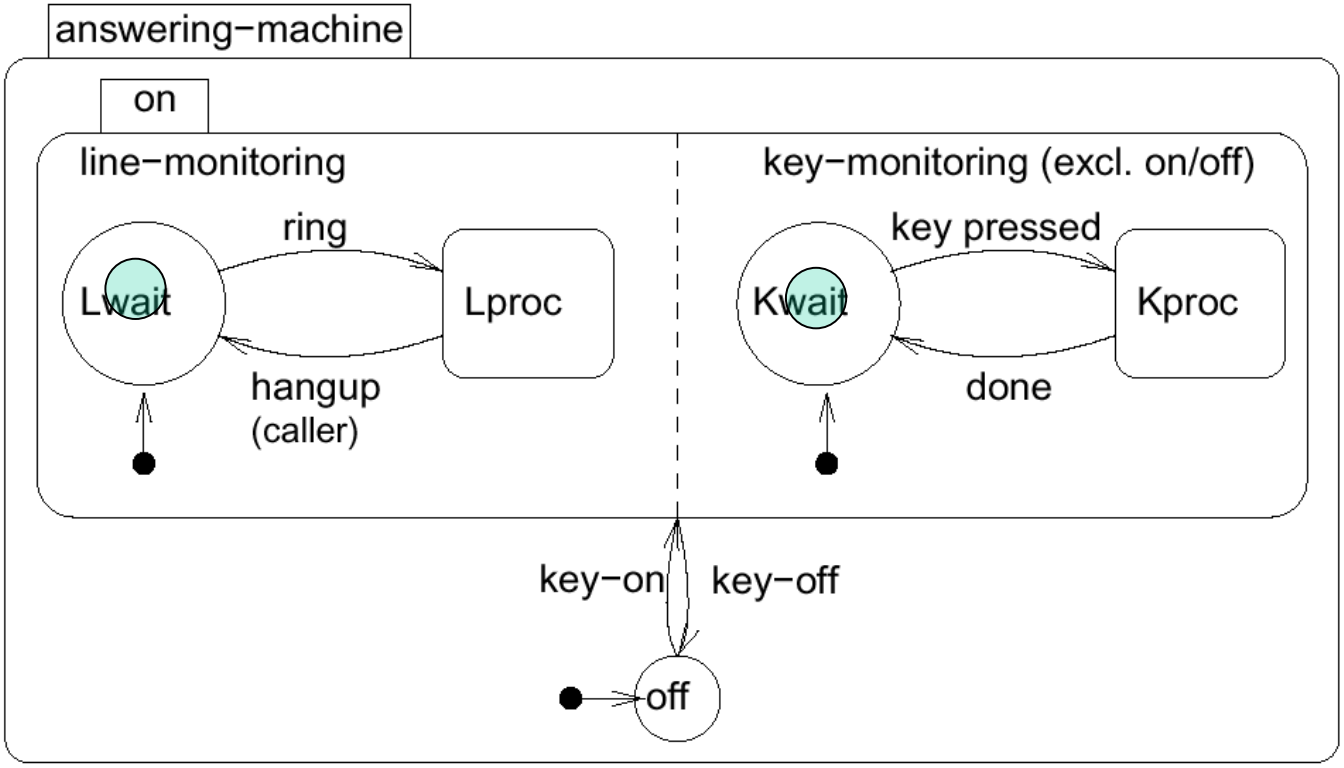
History and default mechanisms can be used hierarchically.

# Combining history and default state mechanism

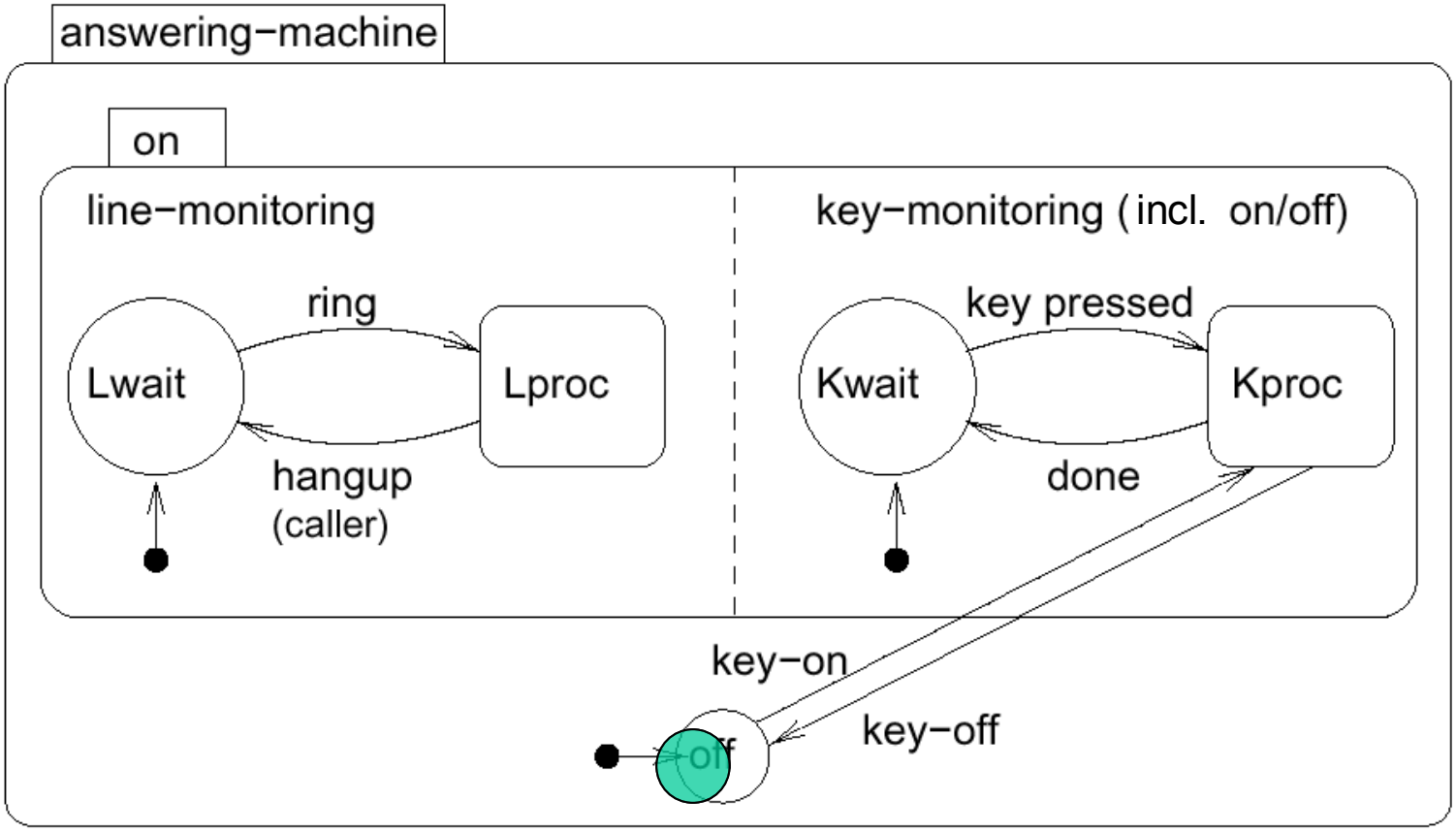


# Concurrency

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



# Entering and leaving AND-super-states




Line-monitoring and key-monitoring are entered and left, when service switch is operated.

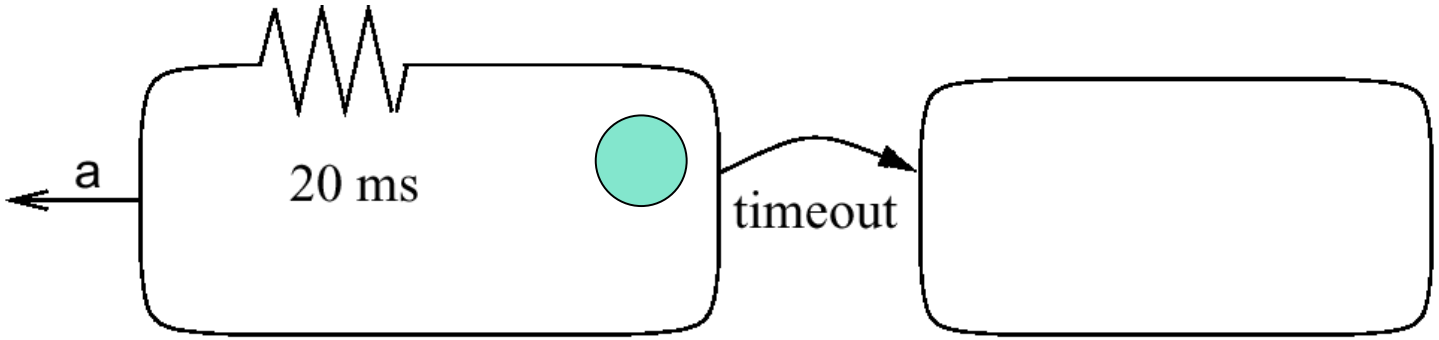
# Types of states

In StateCharts, states are either

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

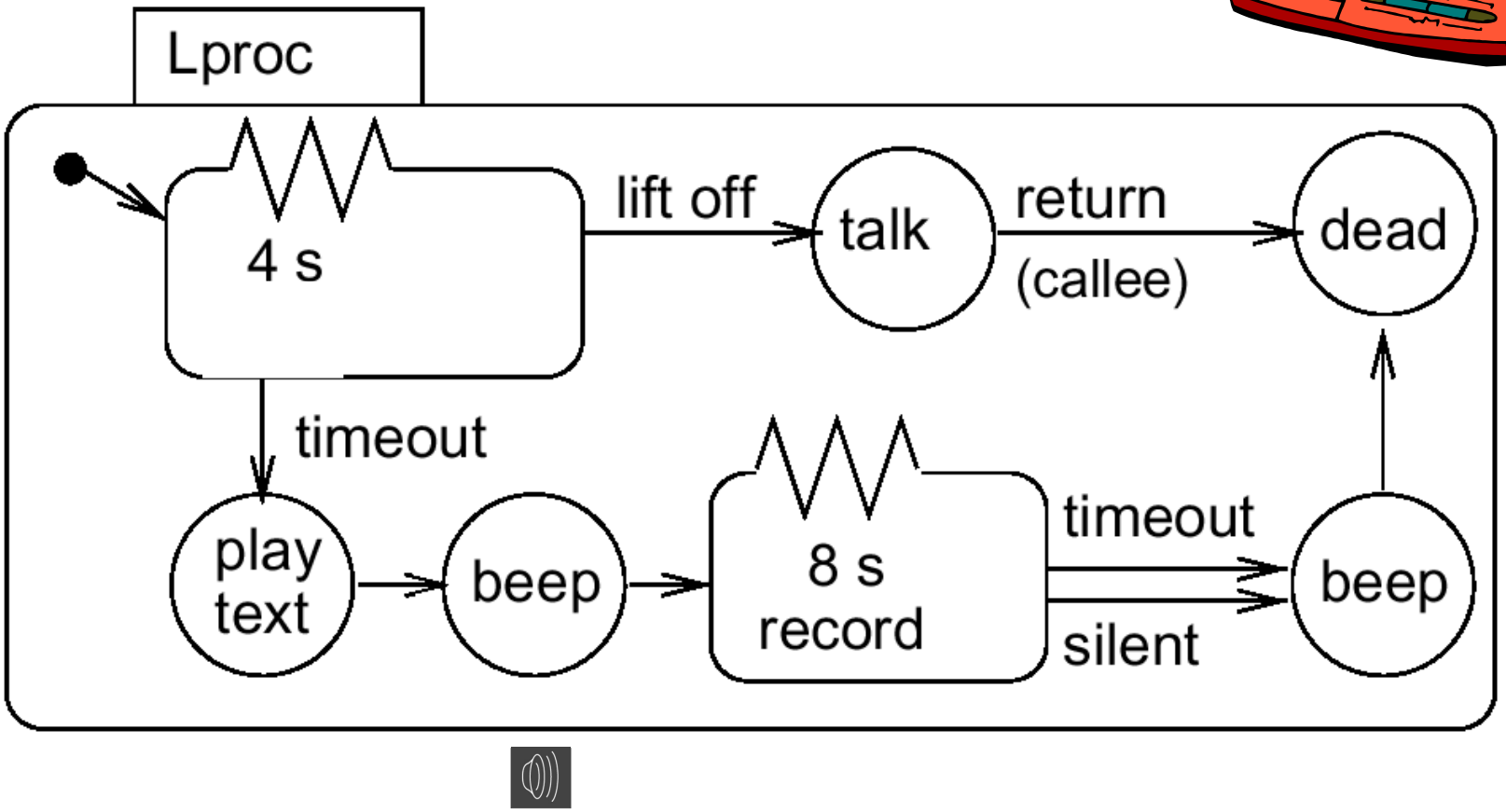
# Timers

Since time needs to be modeled in embedded systems, timers need to be modeled.
In StateCharts, special edges can be used for timeouts.




If event a does not happen while the system is in the left state for 20 ms, a timeout will take place.

# Using timers in an answering machine



# General form of edge labels


**Events:**

- Exist only until the next evaluation of the model
- Can be either internally or externally generated

**Conditions:**

- Refer to values of variables that keep their value until **they are reassigned**

**Reactions:**

- Can either be assignments for variables or creation of events

**Example:**

- service-off [not in Lproc] / service:=0

# The StateCharts simulation phases (StateMate Semantics)

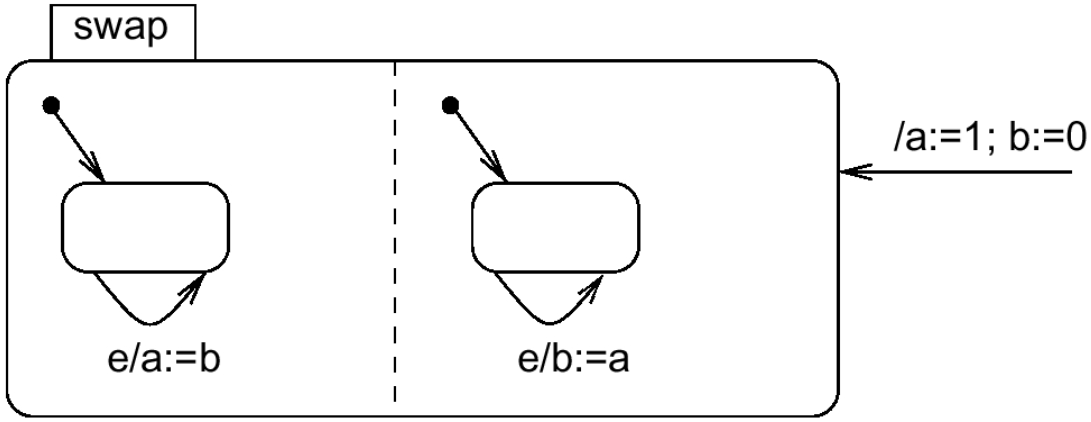
How are edge labels evaluated?

Three phases:

3. Effect of external changes on events and conditions is evaluated,
4. The set of transitions to be made in the current step and right hand sides of assignments are computed,
5. Transitions become effective, variables obtain new values.

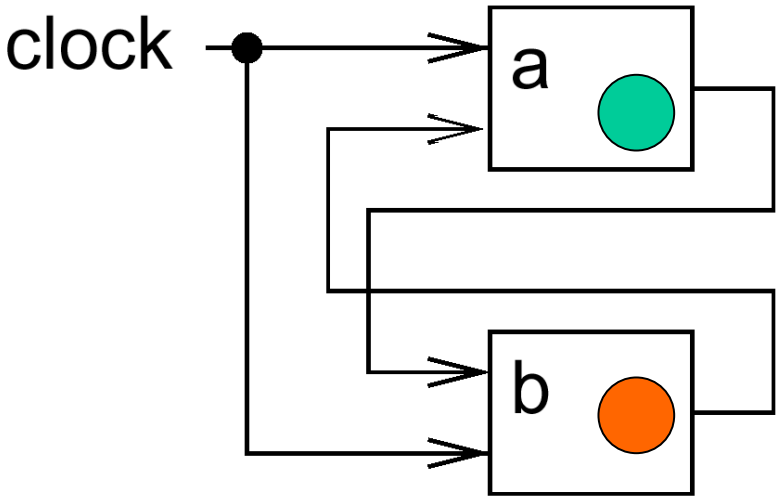
Separation into phases 2 and 3 guarantees deterministic and reproducible behavior.

# Example




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 execution would be non-deterministic.

# Reflects model of clocked hardware

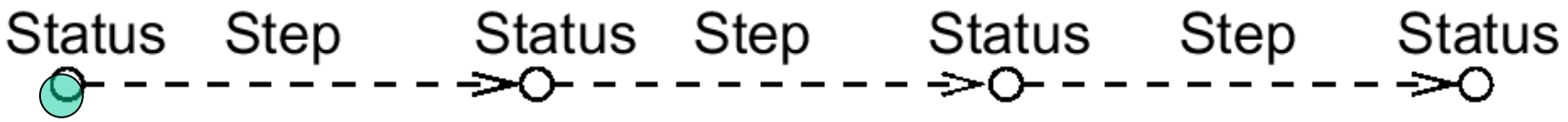



In an actual clocked (synchronous) hardware system, both registers would be swapped as well.

Same separation into phases found in other languages as well, especially those that are intended to model hardware.

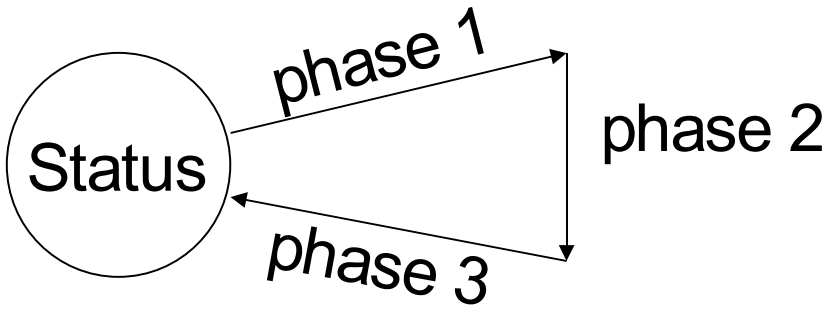
# Steps

Execution of a StateMate model consists of a sequence of (status, step) pairs




Status= values of all variables + set of events + current time

Step = execution of the three phases (StateMate semantics)




Other implementations of StateCharts do not have these 3 phases (and hence are nondeterministic)!

# Other semantics

Several other specification languages for hierarchical state machines (UML, dave, …) do not include the three simulation phases.

These correspond more to a SW point of view with no synchronous clocks.

LabView seems to allow turning the multi-phased simulation on and off.

# Broadcast mechanism

Values of variables are visible to all parts of the StateChart model
New values become effective in phase 3 of the current step and are obtained by all parts of the model in the following step.

!

- StateCharts implicitly assumes a **broadcast** mechanism for variables
  (→ implicit ***shared memory communication***
  –other implementations would be very inefficient -).
- StateCharts is appropriate for local control systems (☺), but not for distributed applications for which updating variables might take some time (☹).

# Lifetime of events

**Events live until the step following the one in which they are generated** („one shot-events“).

# StateCharts deterministic or not?

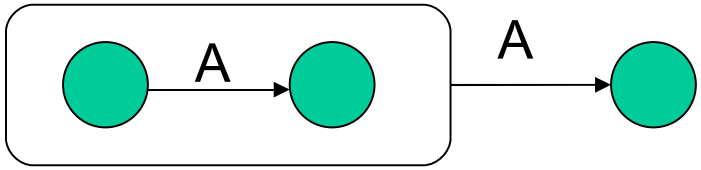
Deterministic (in this context) means:

Must all simulators return the same result for a given input?

- Separation into 2 phases a required condition
- Semantics ≠ StateMate semantics may be non-deterministic

Potential other sources of non-deterministic behavior:

- Choice between conflicting transitions resolved arbitrarily




Tools typically issue a warning if such non-determinism could exist

→ **Deterministic behavior for StateMate semantics if transition conflicts are resolved deterministically and no other sources of non-determinism exist**

# Evaluation of StateCharts (1)

**Pros:**

- Hierarchy allows arbitrary nesting of AND- and OR-super states.
- (StateMate-) Semantics defined in a follow-up paper to original paper.
- Large number of commercial simulation tools available (StateMate, StateFlow, BetterState, ...)
- Available „back-ends“ translate StateCharts into C or VHDL, thus enabling software or hardware implementations.

# Evaluation of StateCharts (2)

**Cons:**

- Generated C programs frequently inefficient,
- Not useful for distributed applications,
- No program constructs,
- No description of non-functional behavior,
- No object-orientation,
- No description of structural hierarchy.

**Extensions:**

- Module charts for description of structural hierarchy.

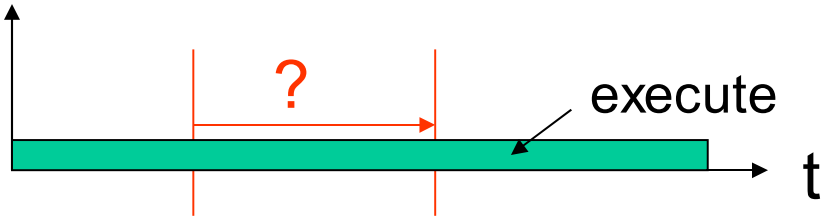
# Some general properties of languages

Peter Marwedel
Informatik 12
Univ. Dortmund
Germany

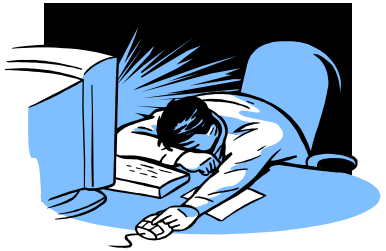
# 1. Specifying timing (1)

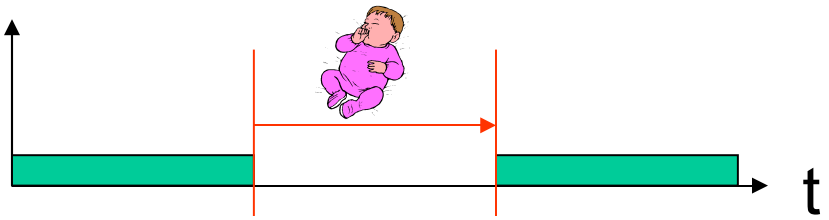
4 types of timing specs required [Burns, 1990]:

- **Measure elapsed time**
  Check, how much time has elapsed since last call



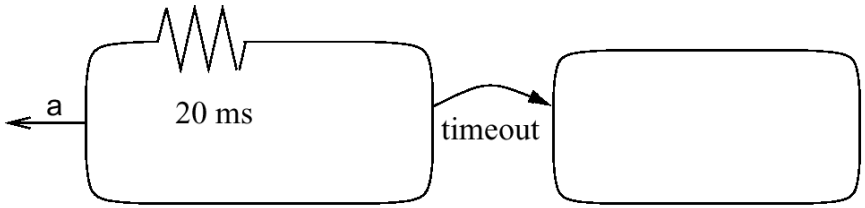

- **Means for delaying processes**



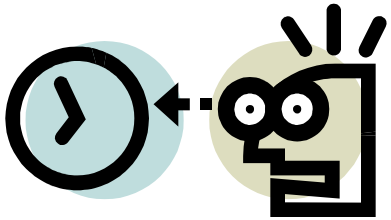


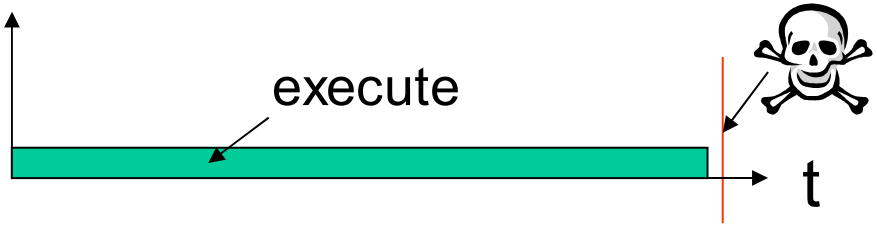
# 2. Specifying timing (2)

- **Possibility to specify timeouts**
  Stay in a certain state a maximum time.




- **Methods for specifying deadlines**
  Not available or in separate control file.






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

# 2. Properties of processes (1)

- **Number of processes**
  static;
  dynamic (dynamically changed hardware architecture?)

- **Nesting:**
  - Nested declaration of processes
    **process** {
      **process** {
        **process** {
    }}}
  - or all declared at the same level
    **process** { … }
    **process** { … }
    **process** { … }

# 2. Properties of processes (2)

- Different techniques for **process creation**
  - **Elaboration in the source (c.f. ADA, below)**

    ```
    declare
       process P1 …
    ```
  - **explicit fork and join (c.f. Unix)**

    ```
    id = fork();
    ```
  - **process creation calls**

    ```
    id = create_process(P1);
    ```

☞ **StateCharts comprises a static number of processes, nested declaration of processes, and process creation through elaboration in the source.**

# 3. Using non-standard I/O devices -

Direct access to switches, displays etc;
No protection required; OS can be much faster than for operating system with protection.

- **No support in standard StateCharts.**
- **No particular OS support anyhow.**

# 4. Synchronous vs. asynchronous languages (1)

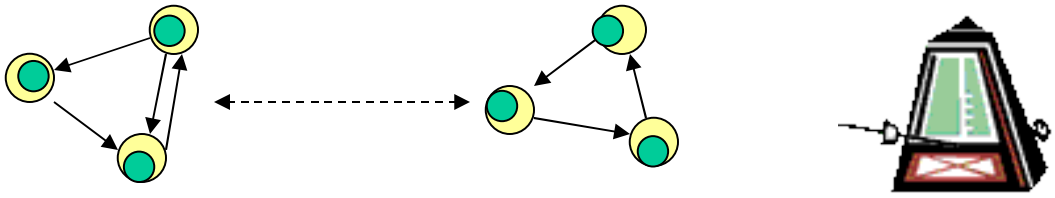
Description of several processes in many languages non-deterministic:
The order in which executable tasks are executed is not specified (may affect result).

Synchronous languages: based on automata models.

“Synchronous languages aim at providing high level, modular constructs, to make the design of such an automaton easier [Halbwachs].

Synchronous languages describe concurrently operating automata. “.. *when automata are composed in parallel, a transition of the product is made of the "simultaneous" transitions of all of them“.*

# 4. Synchronous vs. asynchronous languages (2)



Synchronous languages implicitly assume the presence of a (global) clock. Each clock tick, all inputs are considered, new outputs and states are calculated and then the transitions are made.

Requires a broadcast mechanism for all parts of the model.

Idealistic view of concurrency.

Has the advantage of guaranteeing deterministic behavior.

☞ **StateCharts using StateMate semantics is a synchronous language.**

# Summary

**StateCharts as an example of shared memory MoCs**

- AND-states
- OR-states
- Timer
- Broadcast
- Semantics
  - multi-phase models
  - single-phase models

**Some general language properties**

- Process creation techniques,
- asynchronous/synchronous languages