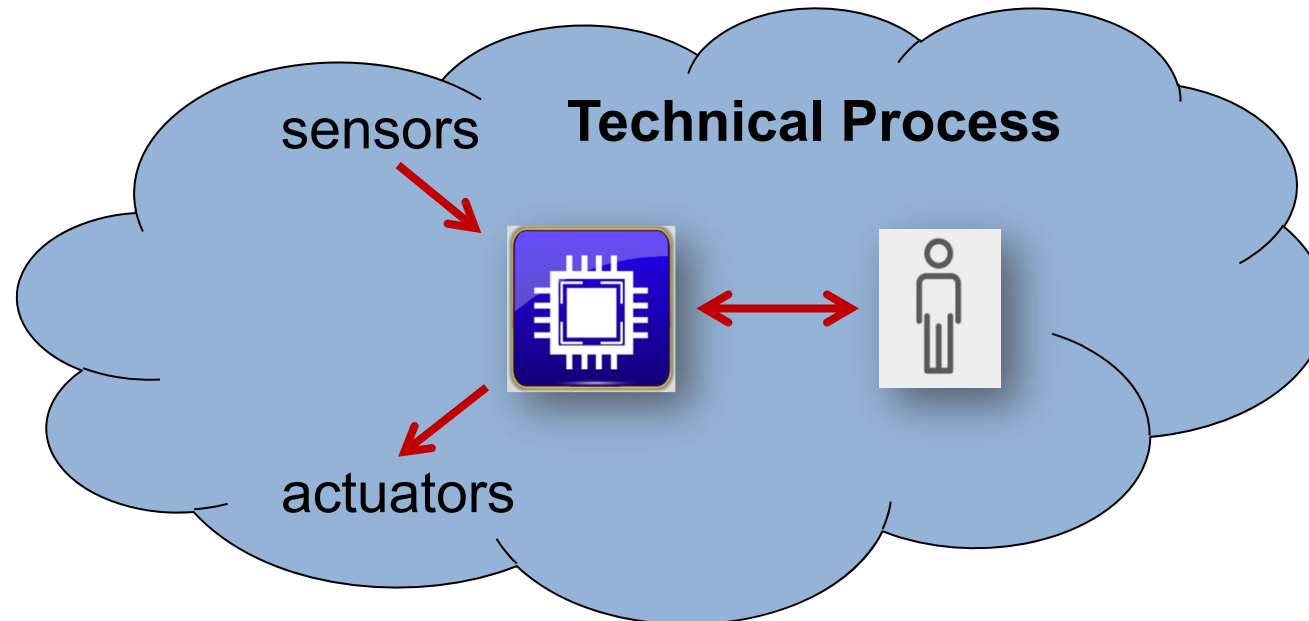
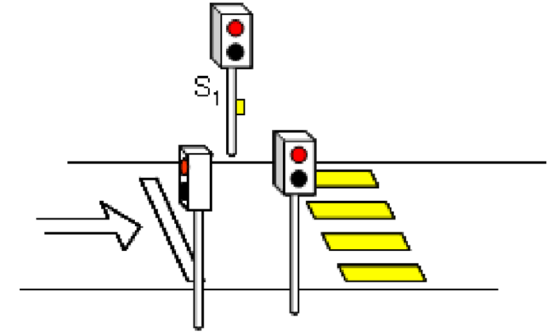


Software State Machines

Computer Engineering 2

- **Embedded Systems**
 - Embedded in technical context / process
 - Application specific
- **How is a Finite State Machine (FSM) modeled in software?**



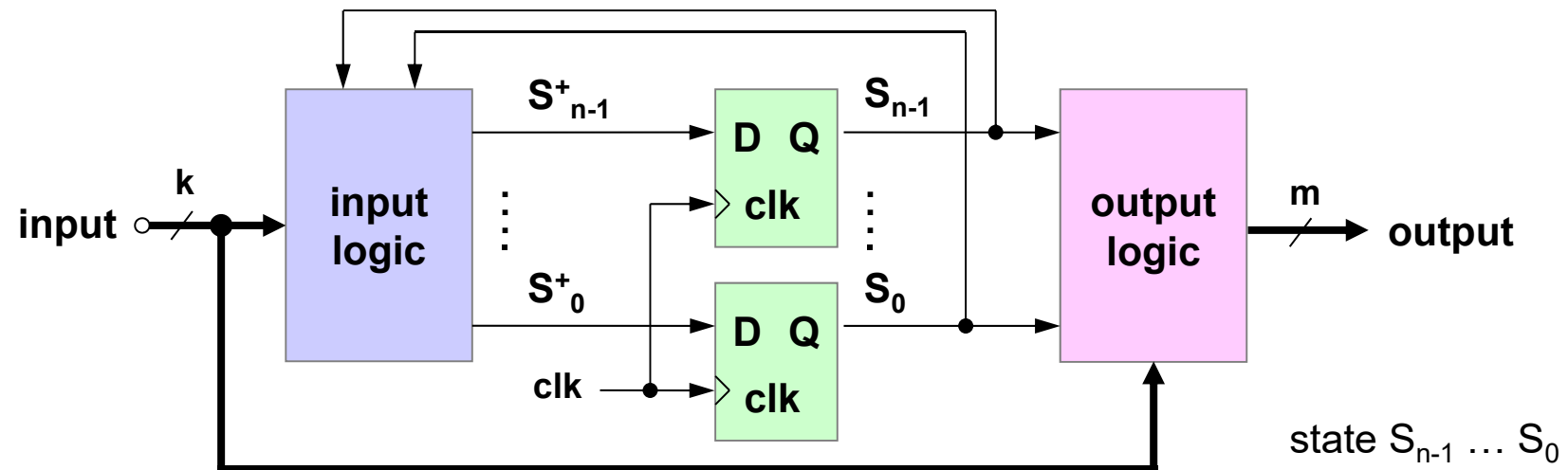
- **Repetition: FSM in Hardware**
- **FSM in Software**
- **Modeling State Machines in UML**
- **Implementation in C**
- **Interaction of FSMs**

At the end of this lesson you will be able

- to explain the term «Finite State Machine» (FSM)
- to outline why FSMs in software are often modeled in a different way than in hardware
- to explain the concept of a reactive system (state-event model)
- to correctly use the related terms and to interpret a UML state diagram with the basic elements
- to correctly model/describe an FSM using the basic elements of a UML state diagram
- to name and describe the semantics of an UML FSM
- to translate a simple FSM modeled in UML into C-Code

■ Finite State Machine (FSM) in hardware

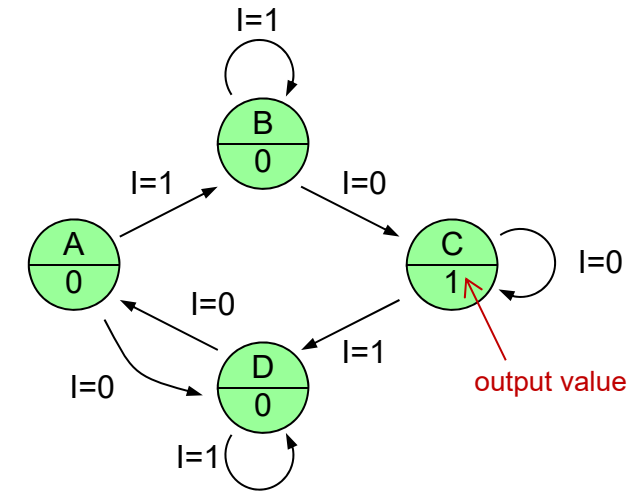
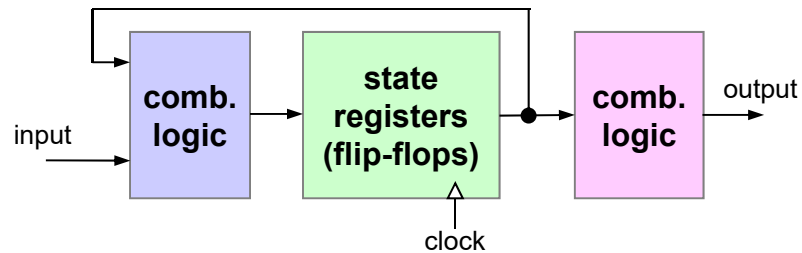
- Flip-flops store internal state
- Clock-driven
 - Inputs are evaluated¹⁾ at each clock edge
 - State can only change on a clock edge



¹⁾ Evaluation of inputs → interpretation of signal levels: '0' vs. '1'

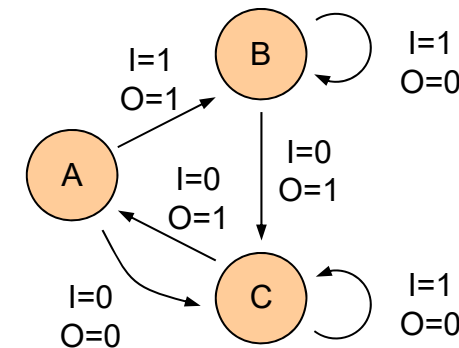
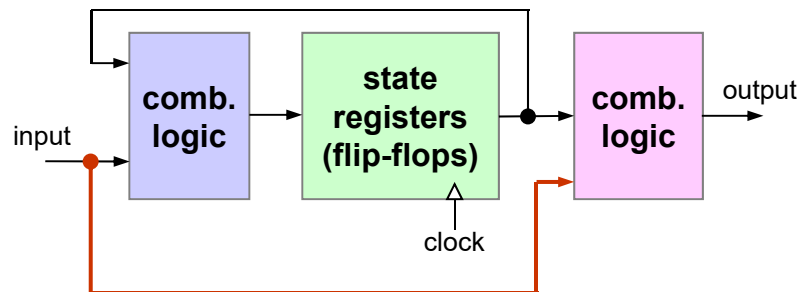
■ Moore

- Input signals influence state only



■ Mealy

- Input signals influence state AND output signals



■ Why software is different

- **Hardware** is intrinsically **parallel**
 - Several FSMs can be processed in parallel using the same clock – Large number of flip-flops and gates evaluate simultaneously
 - Use of common clock as the only event
 - Evaluate signal levels of inputs at clock edges
 - Unchanged signal levels of inputs do not create overhead
- **Software** is intrinsically **sequential**
 - CPU has to process one FSM after the other
 - "HW approach" would require a function call on each clock edge
 - All FSM inputs would have to be evaluated on each function call even if they have not changed → creates a large processing load for CPU
 - Cooperating FSMs: Using a "synchronous clock approach" as in hardware creates a lot of synchronization issues in sequential system



→ use a different approach for software

■ Reactive system → State-event model

- Responds to external events (input)
 - Event-driven
 - Only evaluate the FSM if an input changes
- Internal state
 - Memory of what happened before
- Actions
 - Influence the outside world
- Each event may or may not
 - Change the internal state
 - Trigger actions

Depending on the current state an event may have a different effect.



source www.thediligentadvisor.com

■ State-event applications

Process control

- Washing machines
- Vending machines
- Heating systems



Communication protocols

- Opening and closing connection



Human-machine interface

- Recognition and validation of user inputs



Parsing

- Programming languages



image sources: colourbox, itflow.biz

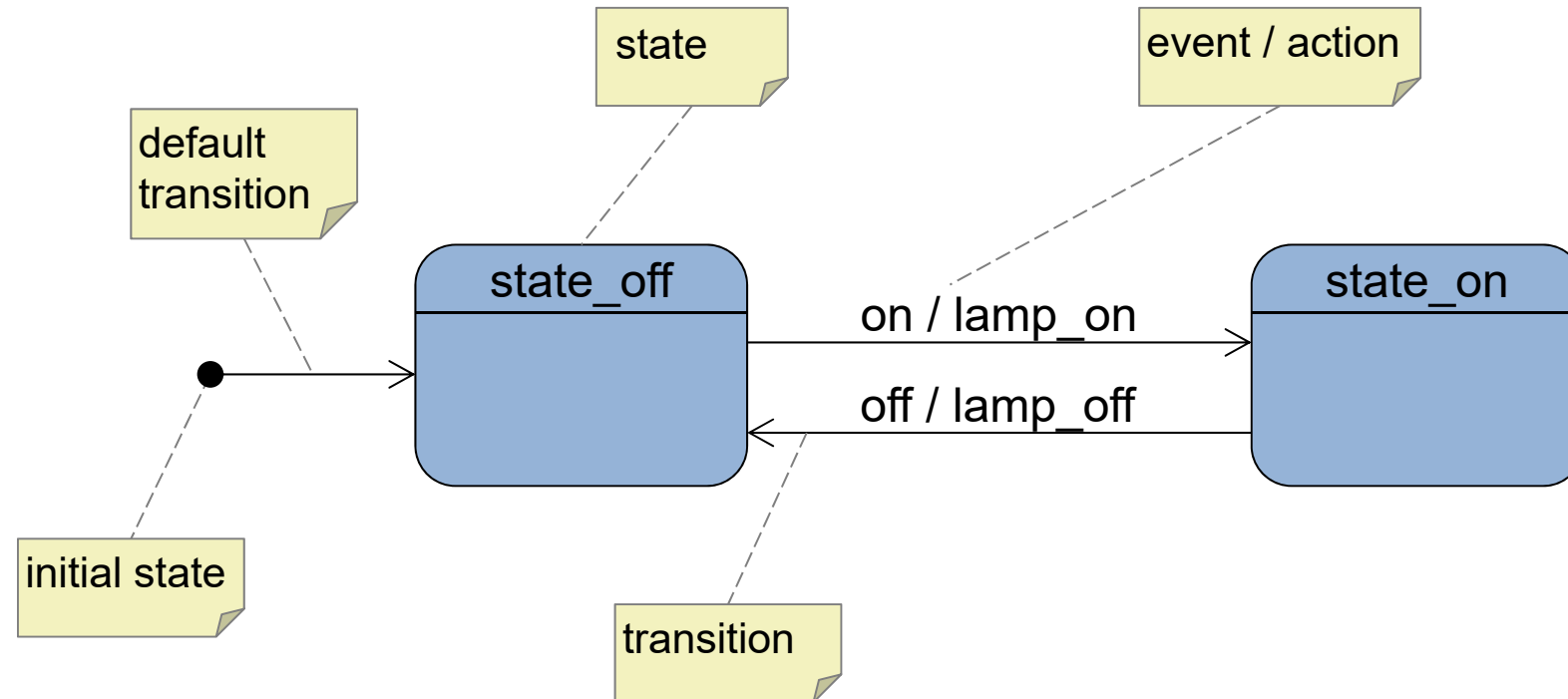
■ UML – Unified Modeling Language

- Graphical modeling of software systems
- Object oriented concepts
- Introduced in the 1990s
 - Since 1997 maintained by Object Management Group (OMG)
 - Since 2000 certified by ISO
- State diagrams
 - Only a part of UML
 - Describe the reactive behavior of classes
 - Based on notation of Prof. David Harel



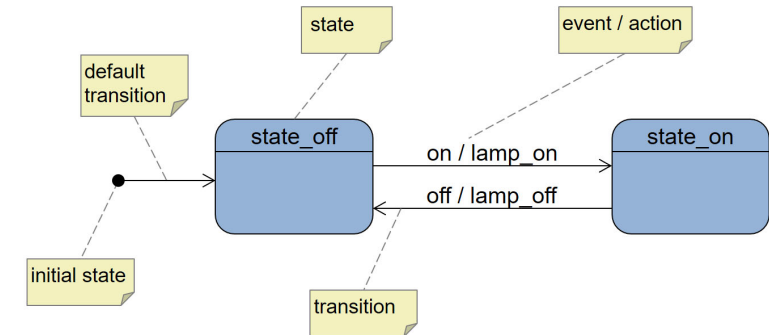
We only cover a small part of the available state diagram notations

■ UML state diagram example



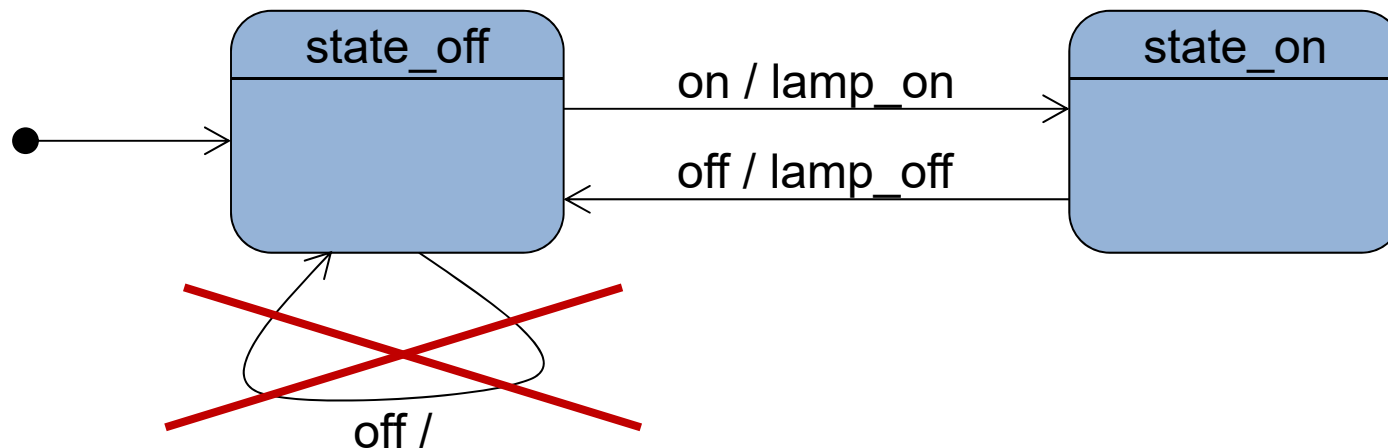
■ Terminology

- State
 - Internal state of the system in which it is awaiting the next event
- Event
 - Asynchronous input that may cause a transition
- Transition
 - Reaction to an event: May change the state and/or trigger an action
- Action
 - Output associated with a transition
 - ▶ Either a directly carried out operation or a
 - ▶ Message to another FSM (seen as an event by the receiving FSM)
- Finite State Machine (FSM)
 - Machine with a finite number of states and transitions

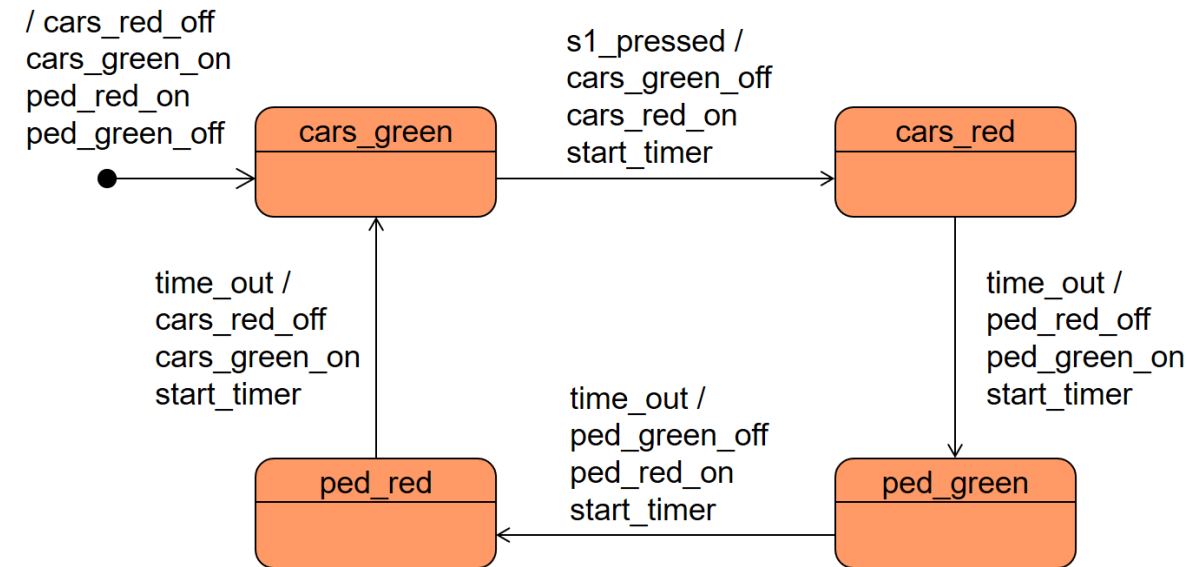
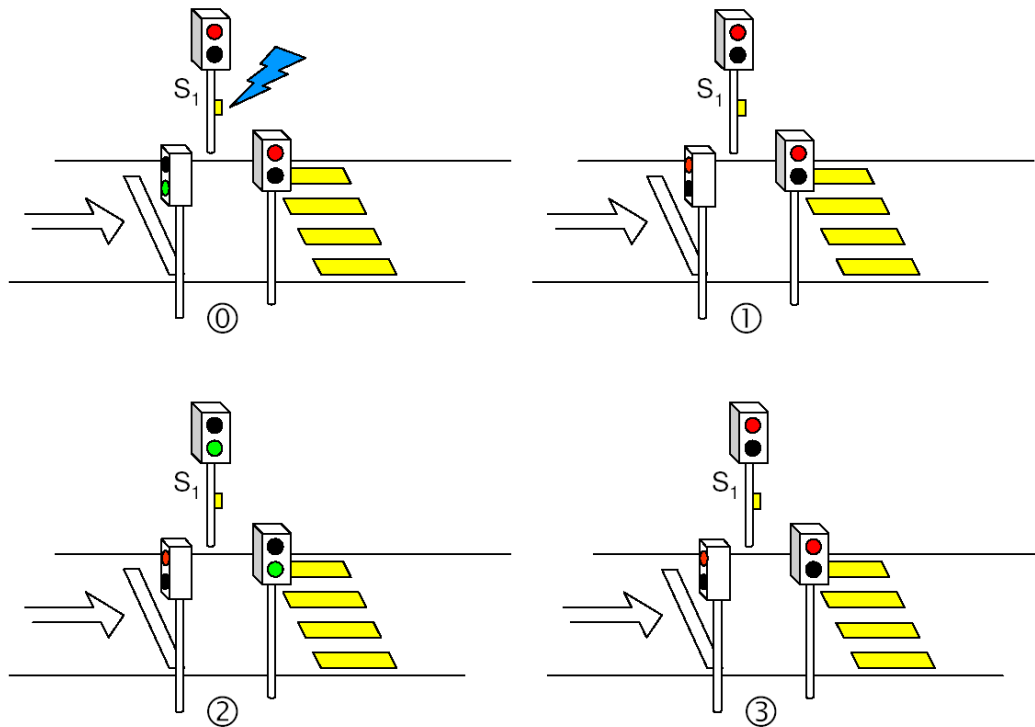


■ In contrast to Mealy notation

- Input asynchronous event
- Output actions → interaction with outside world
- Inputs without effect are omitted
 - If an event triggers neither a state transition nor an action it will not be drawn in the UML state diagram
 - Increases clarity of diagram
- UML contains Mealy and Moore notations as a subset



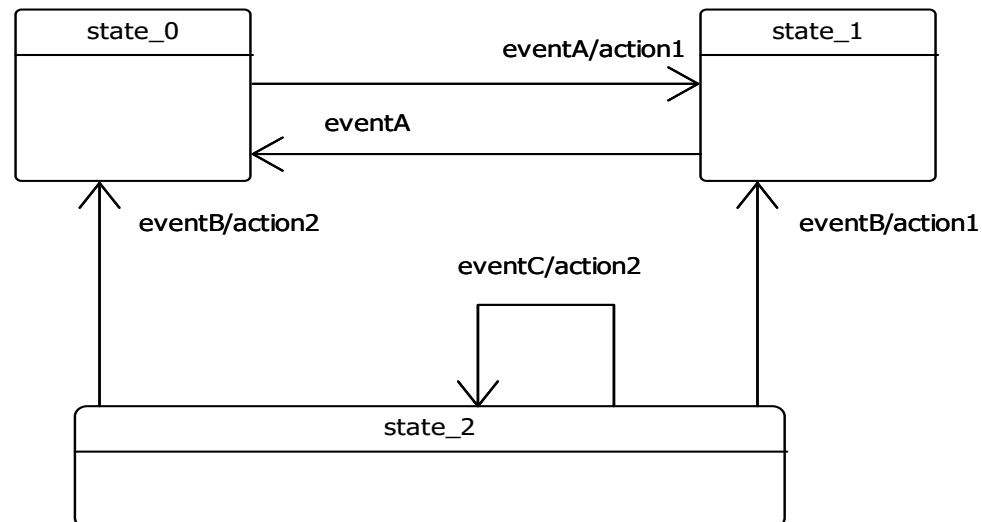
■ Example: Traffic light



■ Rules for UML state machines

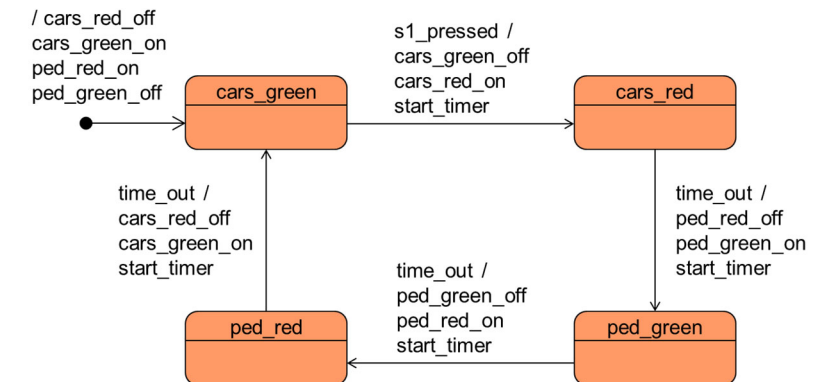
- Every state-diagram must have an initial state
- Each state has to be reachable through a transition
- The state-diagram has to be deterministic
 - I.e. for each event it has to be defined which transition is triggered

What's wrong?



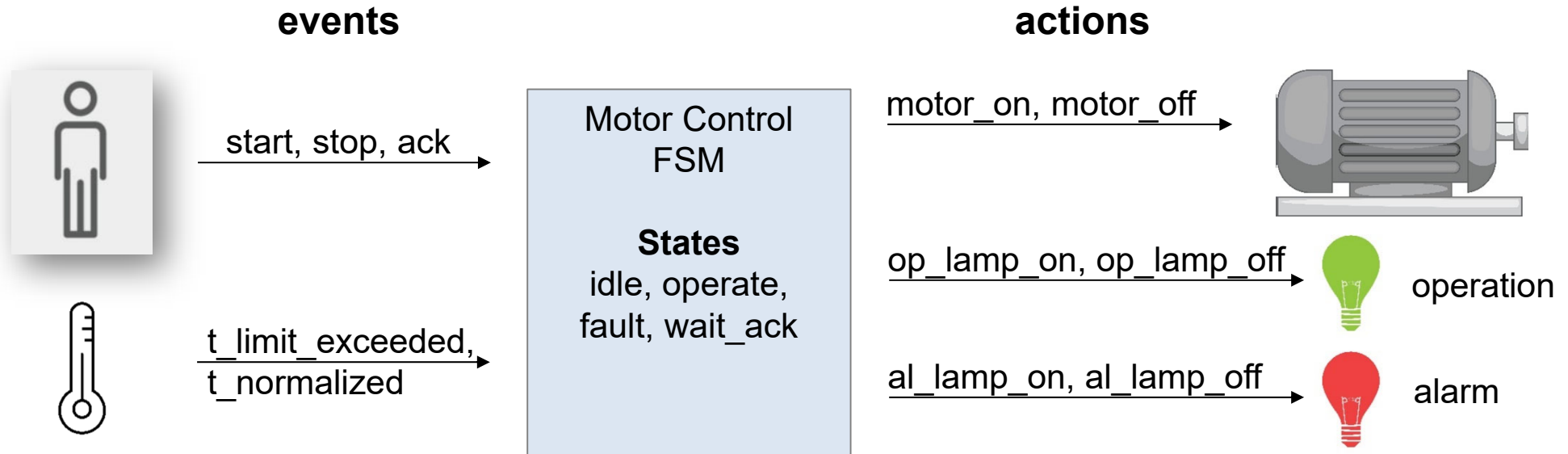
■ Semantics of an UML FSM → understanding the model

- FSM is passive
 - It only reacts to events from the outside
 - Does not initiate any actions on its own
- Always has a defined state
- Reaction to an event depends on the current state
 - State defines the corresponding transition for an event
 - Event is discarded (lost) if no transition for it exists in current state
- Run-to-completion
 - Once started a transition cannot be interrupted
- Strives to avoid querying additional input
 - The executed transition only depends on the event and not on additional inputs

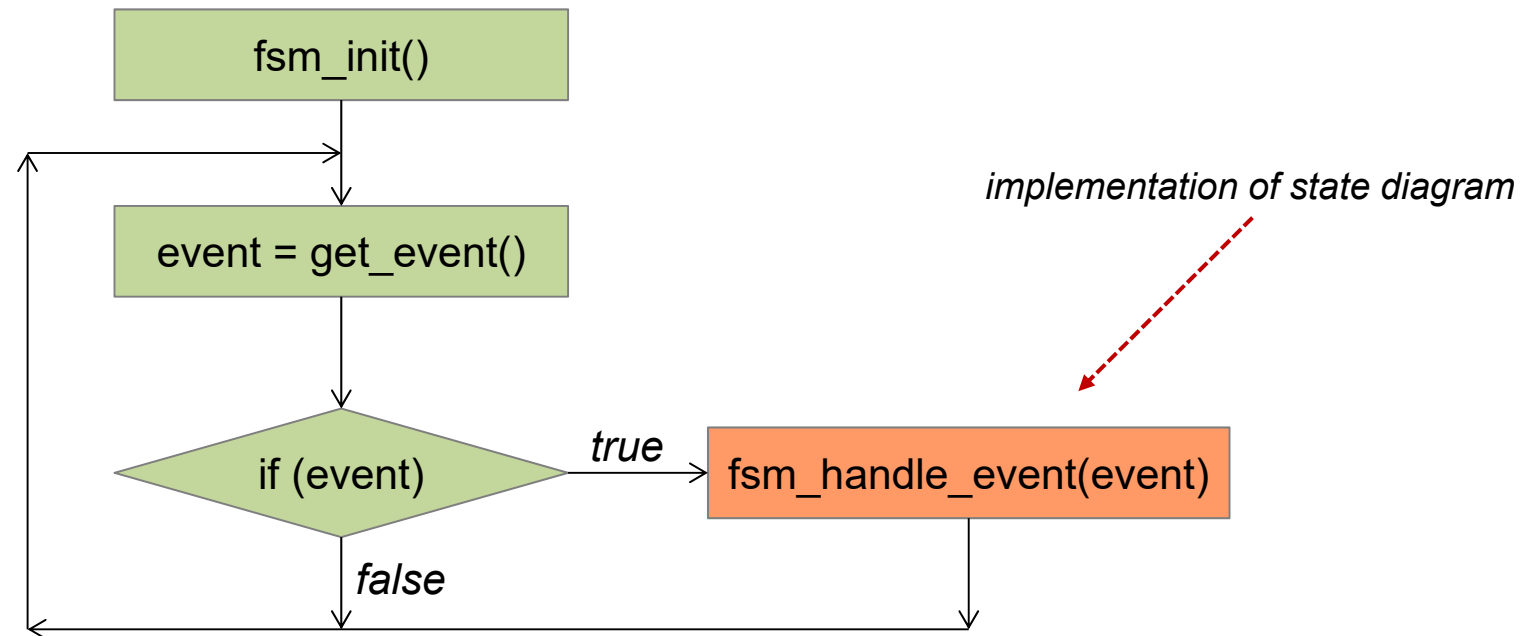


■ Exercise: Draw the UML-State-Diagram for “Motor Control”

- The operator shall be able to turn the motor on and off by pressing the start button and the stop button respectively.
- Motor control shall switch the motor off if the temperature limit is exceeded.
- Restarting the motor after a temperature shutdown shall require pressing the acknowledge button before pressing the start button.
- Normal operation shall be indicated with an operation lamp whereas an alarm condition shall be indicated with an alarm lamp.

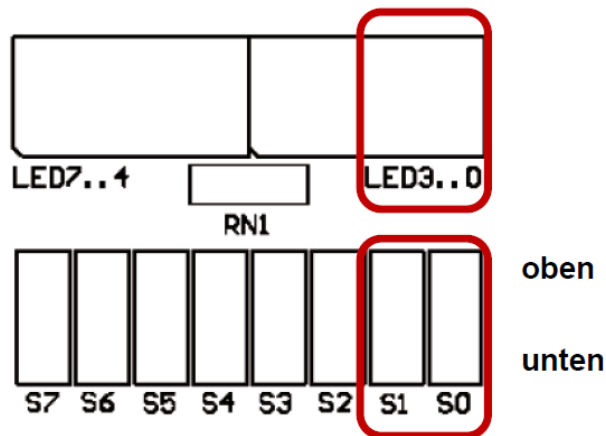


■ Simple FSM System

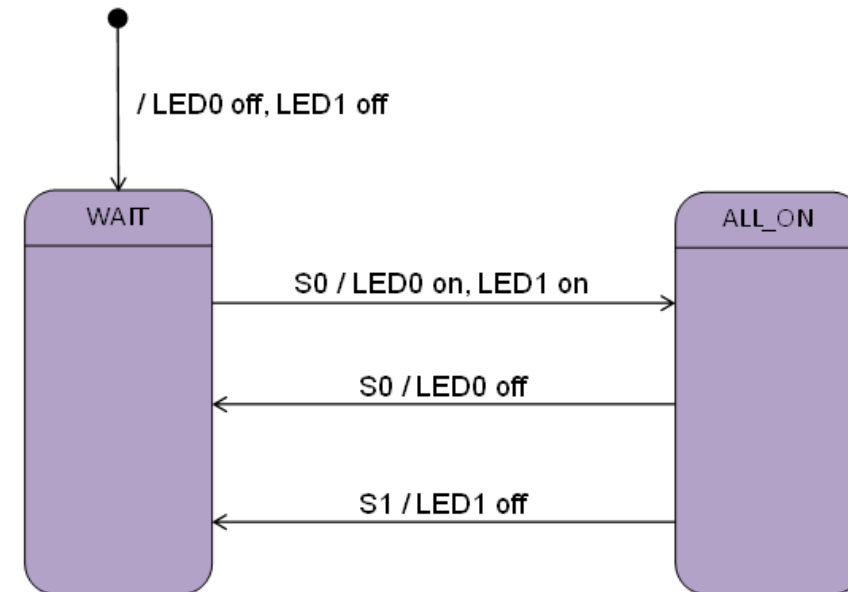


■ Example: LED control on CT-board

- Shifting switch S_x from 'unten' to 'oben' → event S_x
- $x \in \{0, 1\}$



→ event S_x

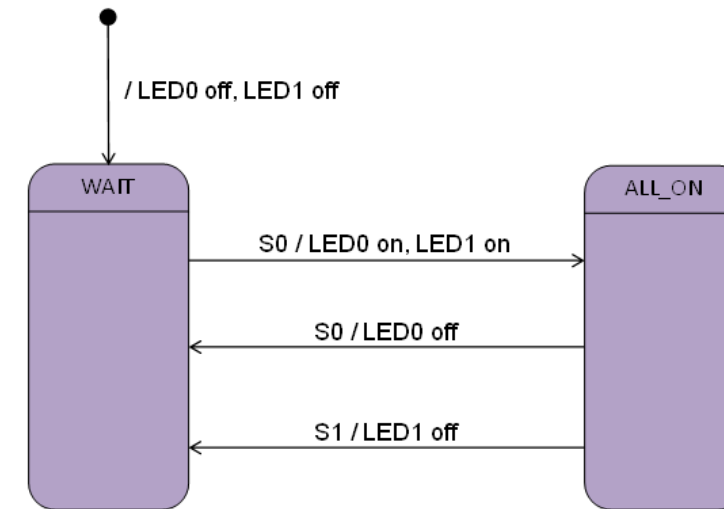


■ Example: LED control

```
int main(void)
{
    event_t event;

    fsm_init();
    while (1) {
        event = get_event();
        if (event != NO_SWITCH) {
            fsm_handle_event(event);
        }
    }
}
```

→ see code example



WAIT		ALL_ON			default
event = ?		event = ?			state = WAIT
S0	default	S0	S1	default	
LED0 on LED1 on	-	LED0 off	LED1 off	-	
state = ALL_ON		state = WAIT	state = WAIT		

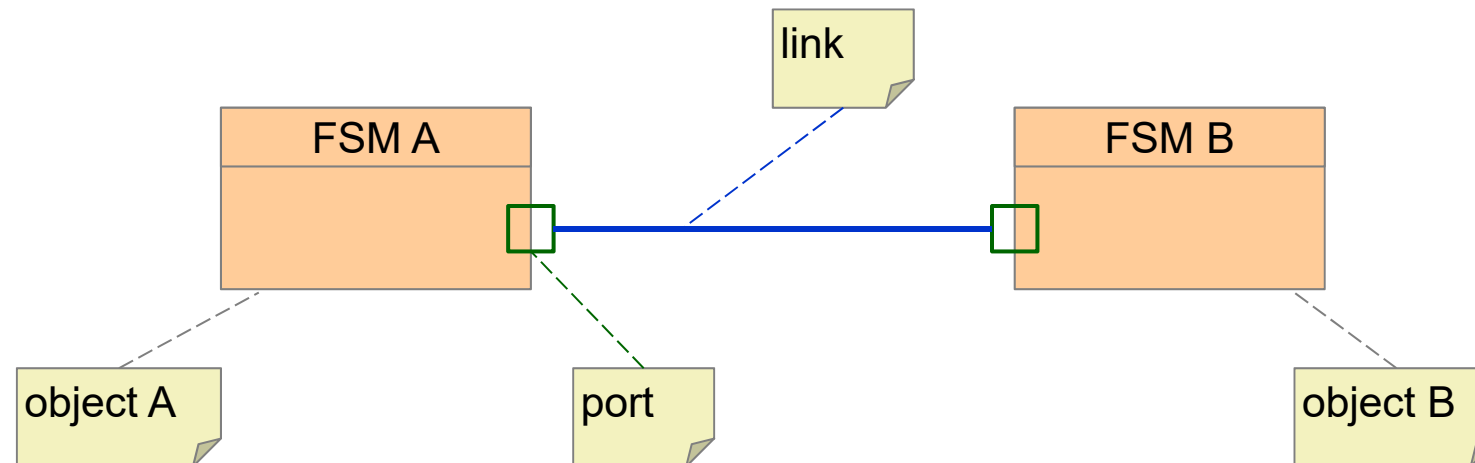
Interaction of FSMs

■ Port

- Defines the messages that can be sent and received by an FSM
 - Output message → action of the FSM
 - Input message → event of the FSM

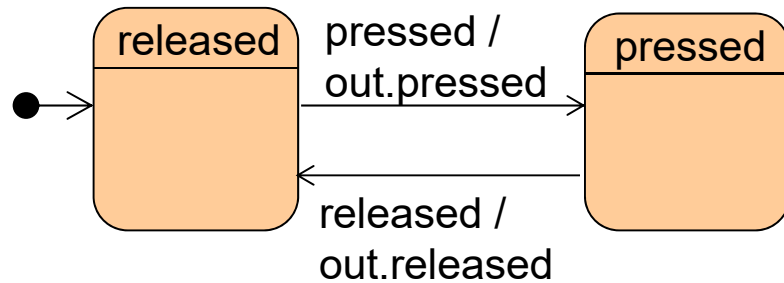
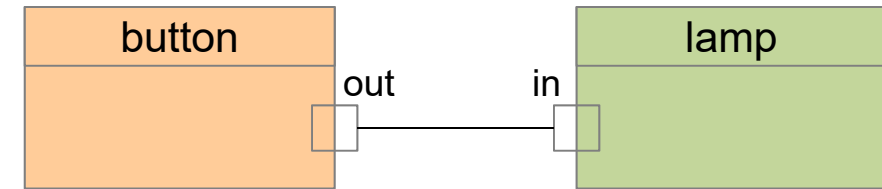
■ Link

- Defines a connection for sending messages

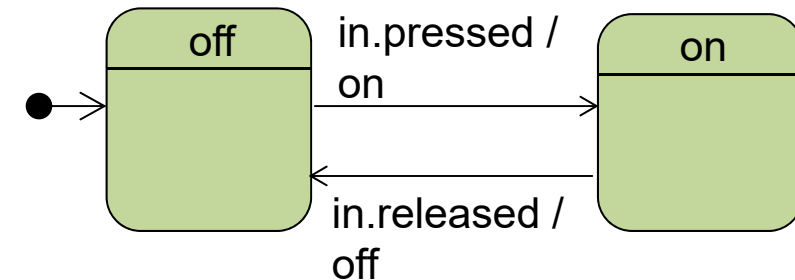


■ Example

- Reactive system partitioned into two FSMs
- Interaction of FSMs happens through event-messages



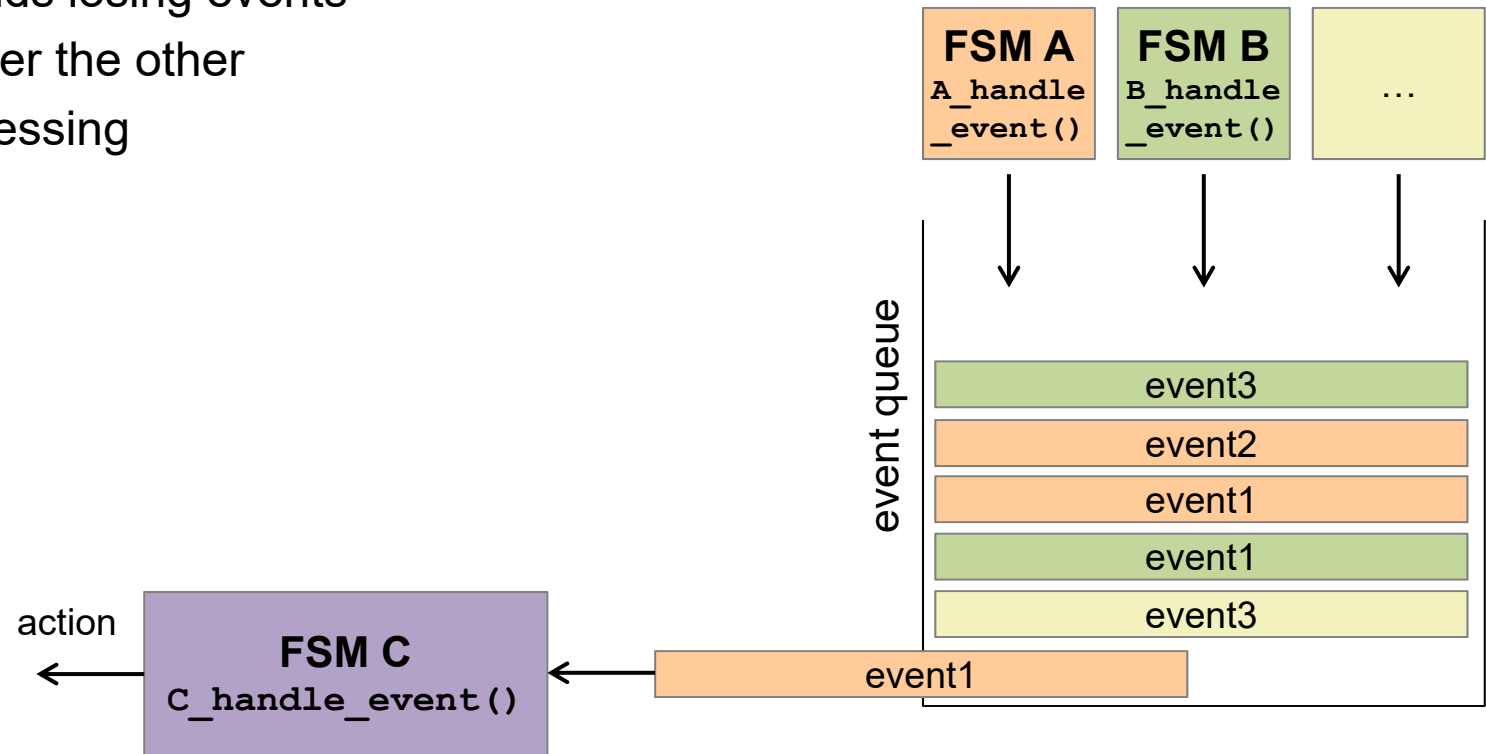
button produces messages on port out



lamp reacts to events on port in

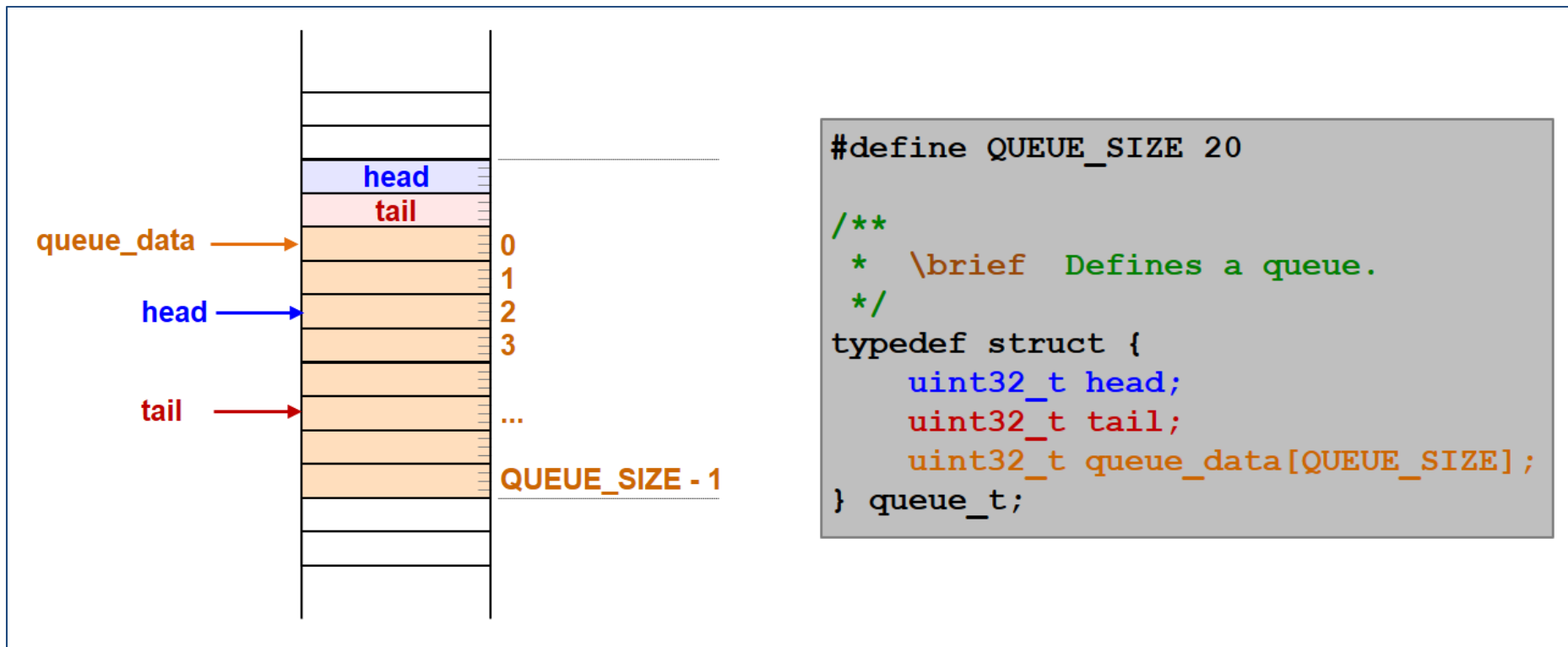
■ Event queue

- Collect events generated by different objects
- Buffered in event queue: Avoids losing events
- FSM processes one event after the other
- Events are deleted after processing



■ Event queues

- Covered in Microcomputer Systems 1 (MC1)



■ Finite State Machine (FSM)

- Allows modeling of reactive systems
- Hardware vs. software
 - Digital logic → clock driven
 - Software → event driven
- Modeling in UML
 - State / Event / Transition / Action
- Implementation in C
 - Switch case
- Interaction of FSMs
 - (Output-) actions of one FSM become (input-) events for other FSM
 - Event queue

```
switch (state) {  
    case STATE_A:  
        switch (event) {  
            case S0:  
                action();  
                state = NEW_STATE;  
                break;  
            default:  
                state = WAIT;  
        }  
        break;  
  
    case STATE_B:  
        switch (event)
```