Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

12.24196

Introduction to Embedded Systems

Prof. Dr.-Ing. Stefan Kowalewski | Julius Kahle, M. Sc.
Summer Semester 2025

Part 3

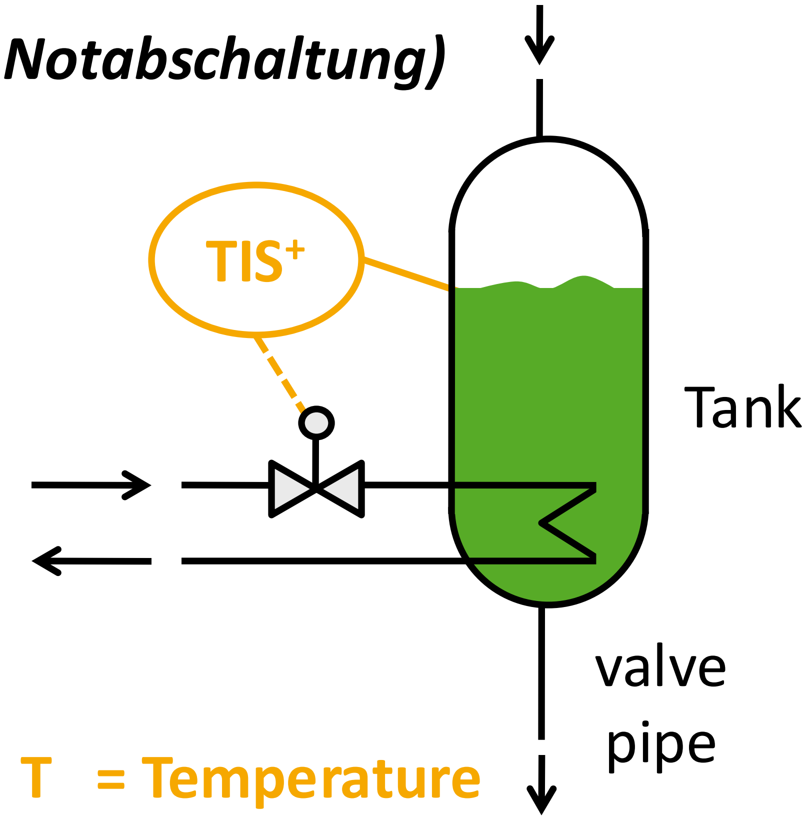
Programmable Logic Controllers

Content

1. Logic Control
2. PLC Technology
3. Programming languages
   - Function Block Diagram (FBD)
   - Ladder Diagram (LD)
   - Instruction List
4. Model-based Design
5. Sequential Controllers
   - Sequential Function Charts

Logic Control vs. Continuous Control (1)

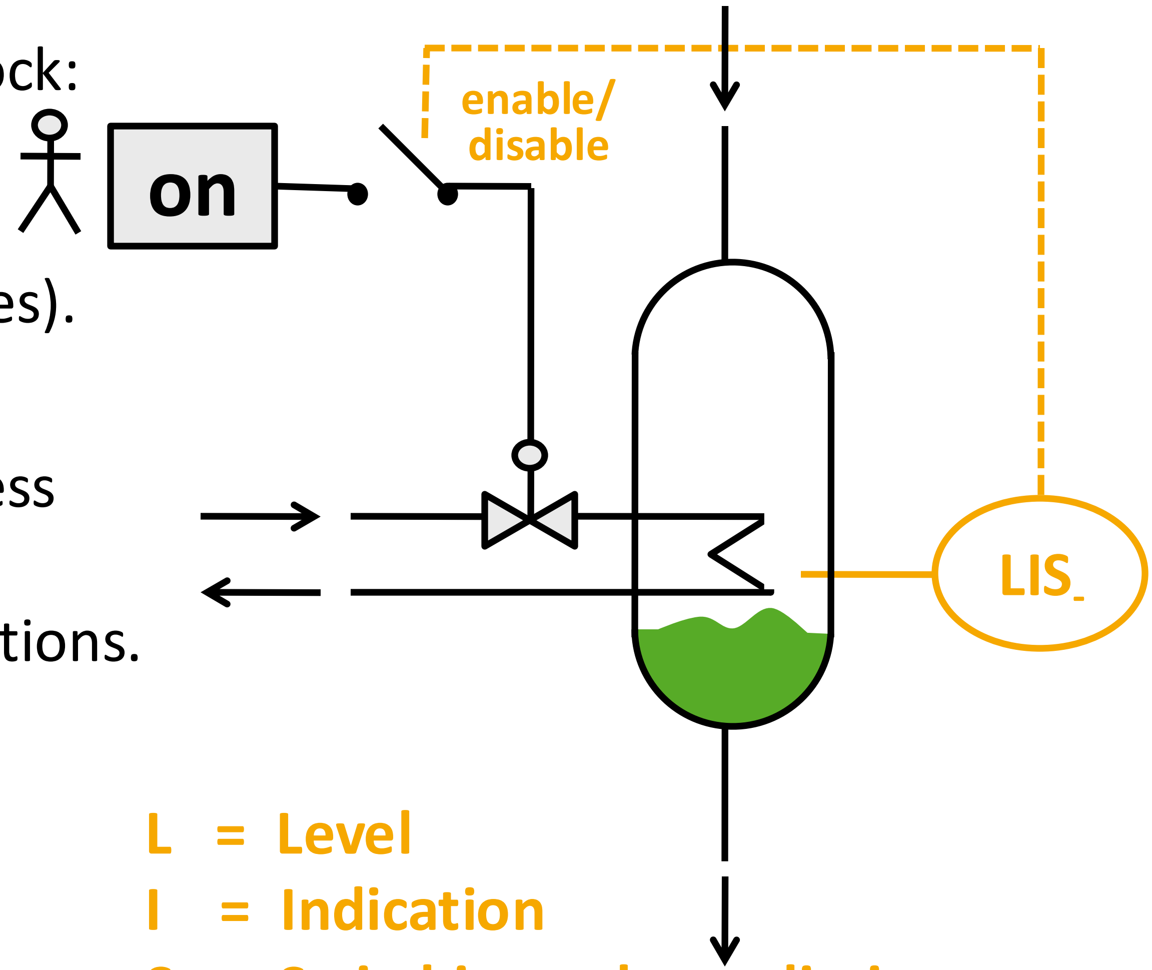
- Introduction by examples
- Example 1: A ***Trip (Abschaltung, Notabschaltung)***
- Representation:
  Piping and Instrumentation Diagram (P&ID, ISO 3511)
- A P&ID shows all major plant equipment like pipes, tanks, valves, and the main control functions
- Function of the trip:
  Switch off heating, when temp has reached a certain value.
- Purpose: Avoid undesired (e.g. dangerous) process states



T = Temperature
I = Indication
S^+ = Switching at upper limit

Logic Control vs. Continuous Control (2)

- Example 2: An ***Interlock (Verriegelung)***
- Function of the interlock:
  Enable only desired operator action (disable undesired ones).
- Purpose:
  Avoid undesired process states caused by undesired operator actions.

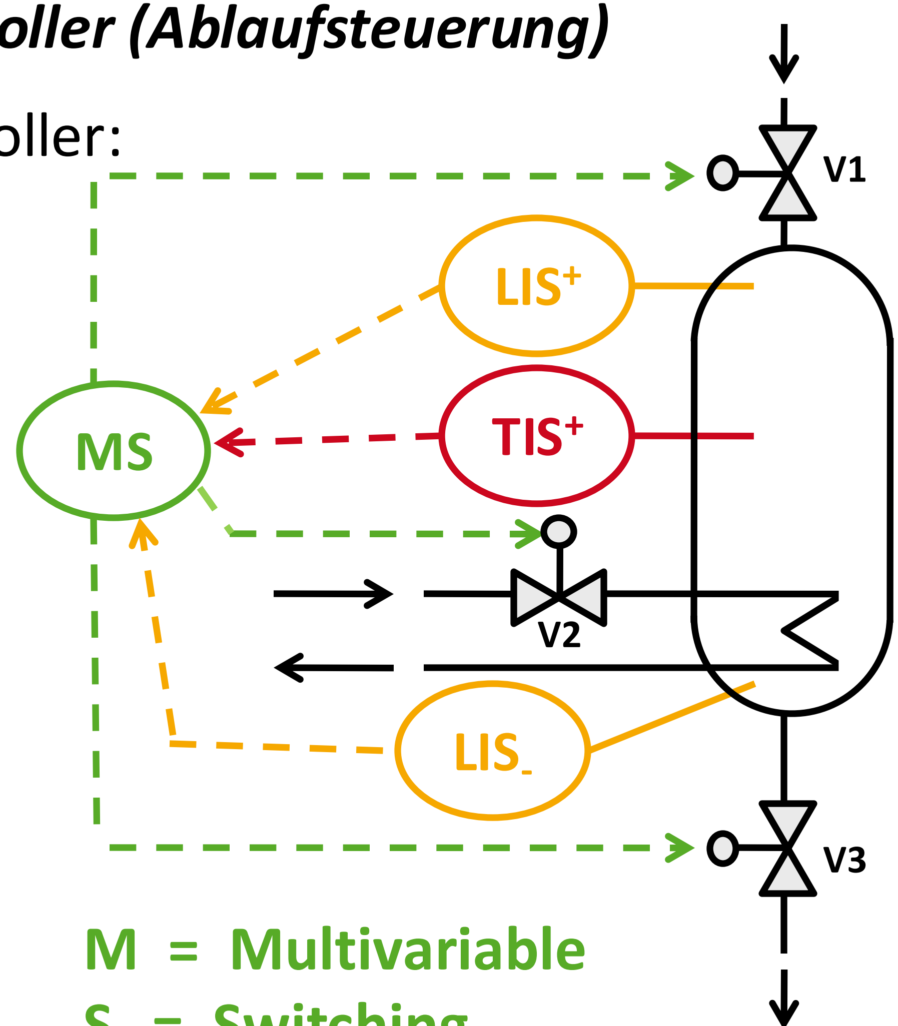


L = Level
I = Indication
S- = Switching at lower limit

Informatik 11 Embedded Software
RWTH AACHEN UNIVERSITY

Logic Control vs. Continuous Control (3)

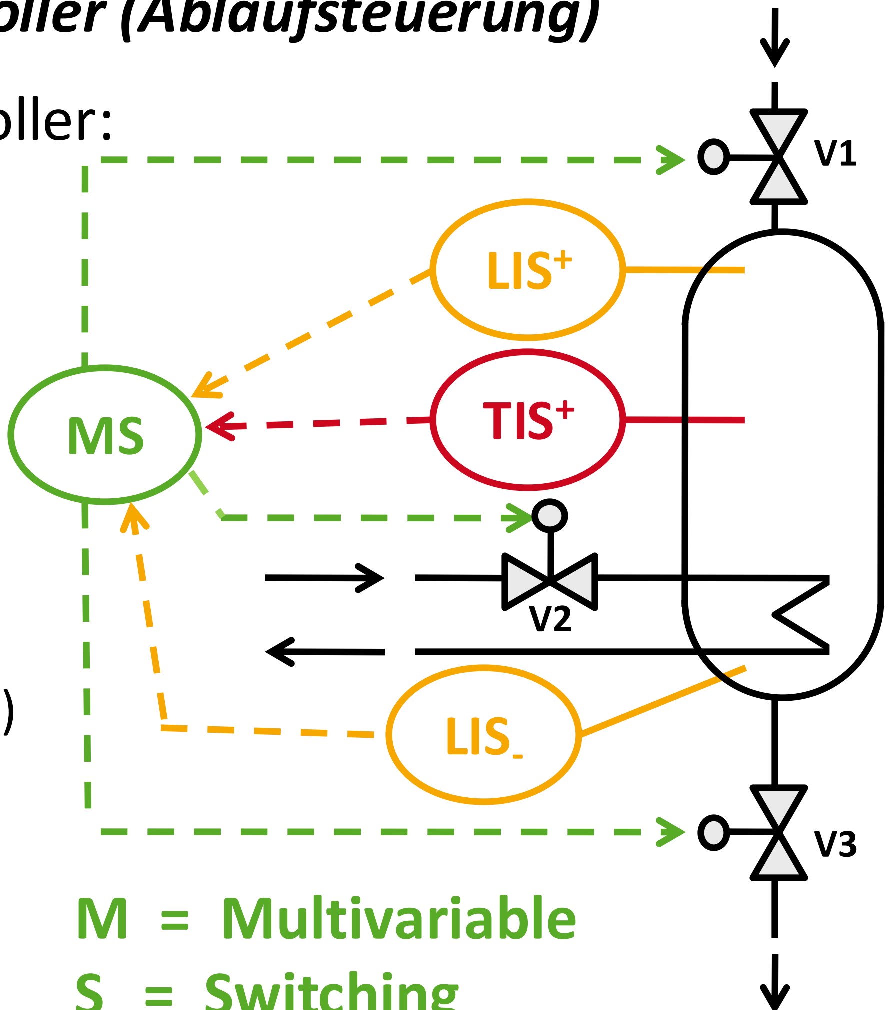
- Example 3: A ***Sequence Controller (Ablaufsteuerung)***
- Function of a sequence controller: Realize a desired sequence of process steps.
- Here:
  - 1. Fill tank (open V1 until LIS⁺)
  - 2. Heat up (open V2 until TIS⁺)
  - 3. Drain tank (open V3 until LIS₋)



M = Multivariable
S = Switching

Logic Control vs. Continuous Control (3)

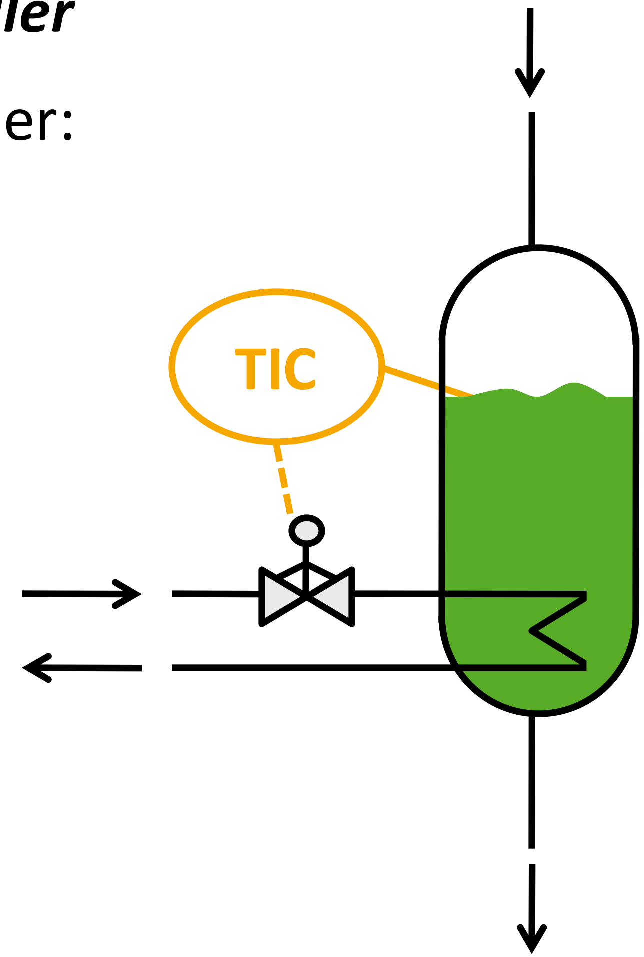
- Example 3: A ***Sequence Controller (Ablaufsteuerung)***
- Function of a sequence controller: Realize a desired sequence of process steps.
- Here:
  - 1. Fill tank (open V1 until LIS⁺)
  - 2. Heat up (open for m minutes)
  - 3. Drain tank (open for n minutes)



M = Multivariable
S = Switching

Logic Control vs. Continuous Control (4)

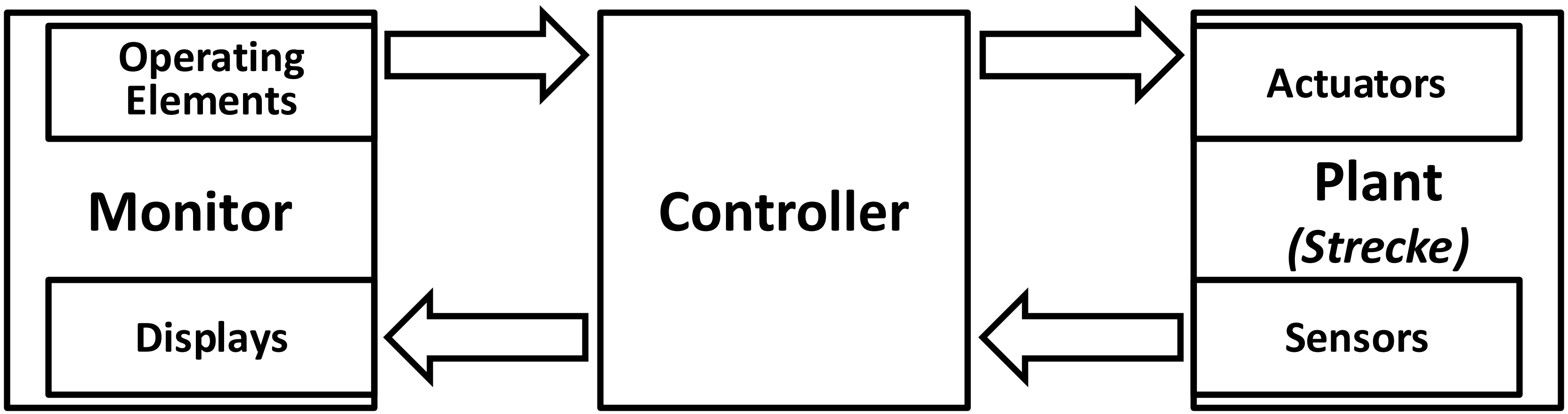
- Example 4: A ***Continuous Controller***
- Function of a continuous controller: Keep process variable at a desired value (despite disturbances or in case of changes of the settings).



Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

Logic Control vs. Continuous Control (5)

- General Control Structure (for all three examples)



Logic Control vs. Continuous Control (6)

- Main difference between logic (trip, interlock, sequence controller) and continuous control?
- The main purpose of logic controllers is the processing of discrete variables
- Example: Trip

Logic Control:

$$\textbf{temp(t)} \in \{ \textbf{below\_limit, above\_limit} \}$$

Continuous Control:

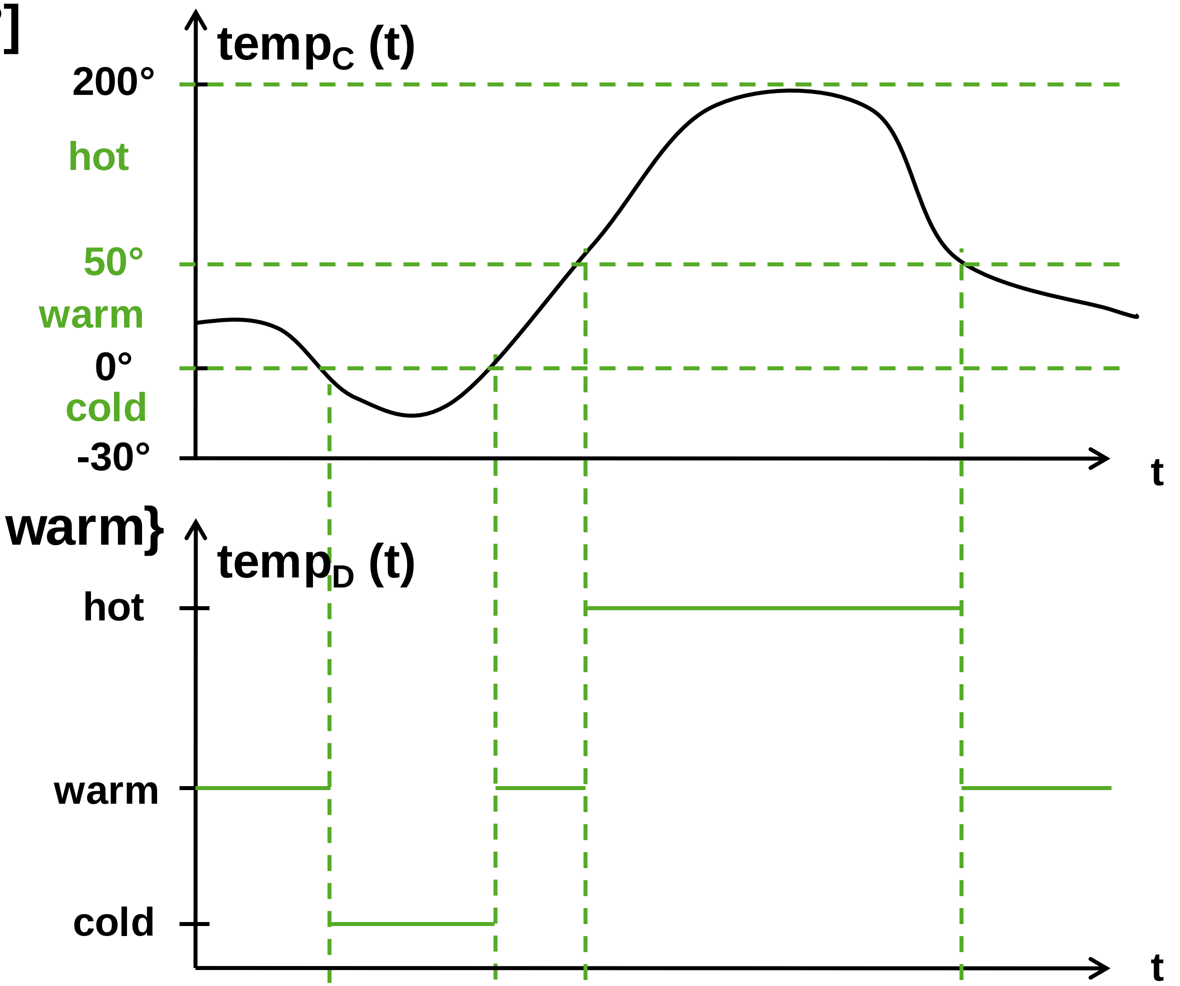
$$\textbf{temp(t)} \subset \mathbb{R} \times \{ {}^{\circ}\textbf{C} \}$$

Informatik 11 Embedded Software
RWTH AACHEN UNIVERSITY

Logic Control vs. Continuous Control (6)

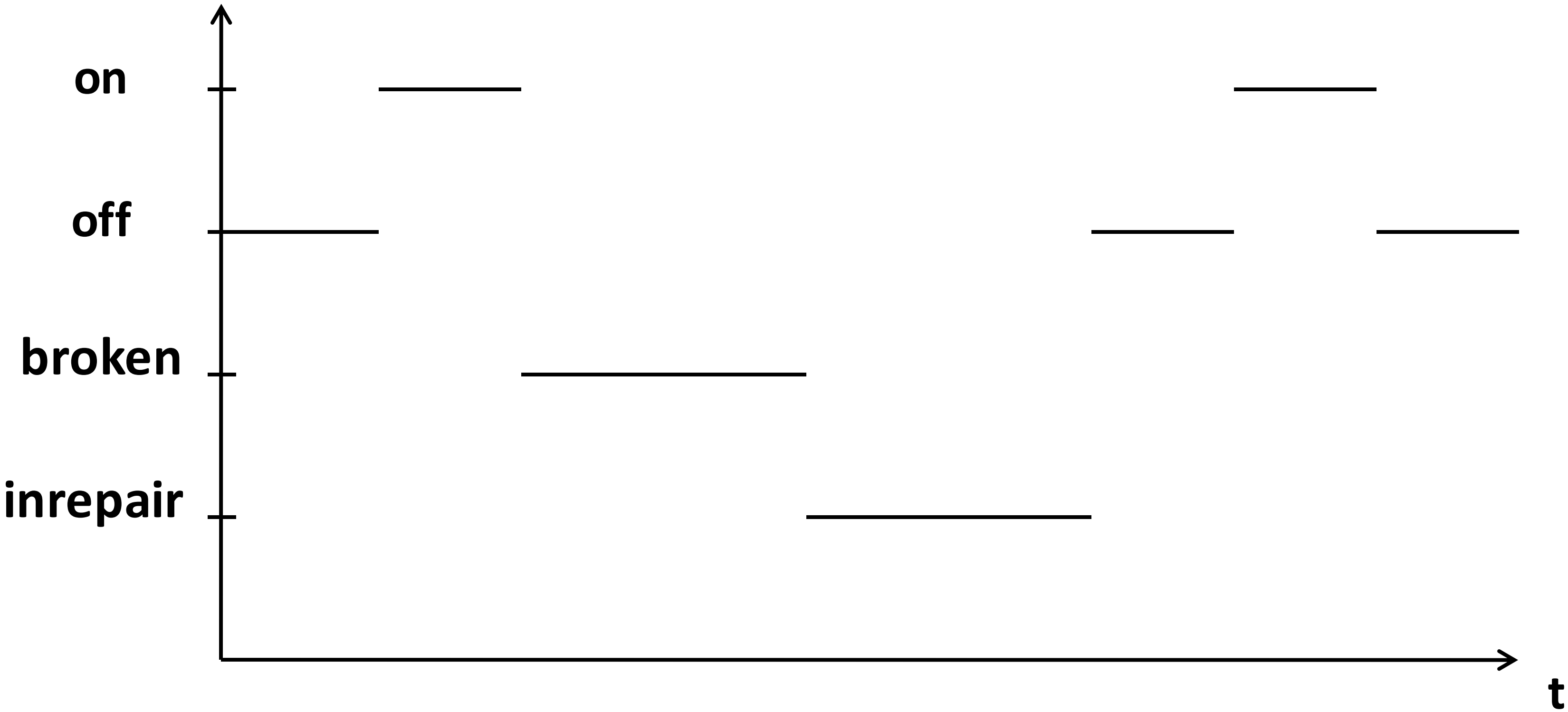
$temp_C\ (t) \in [\ -30° \ldots 200°]$

$temp_D\ (t) \in \{\ cold,\ hot,\ warm \}$



Logic Control vs. Continuous Control (7)

heating(t) $\in$ { on, off, broken, inrepair}



- Difference?

Content

1. Logic Control
2. PLC Technology
3. Programming languages
   - Function Block Diagram (FBD)
   - Ladder Diagram (LD)
   - Instruction List
4. Model-based Design
5. Sequential Controllers
   - Sequential Function Charts

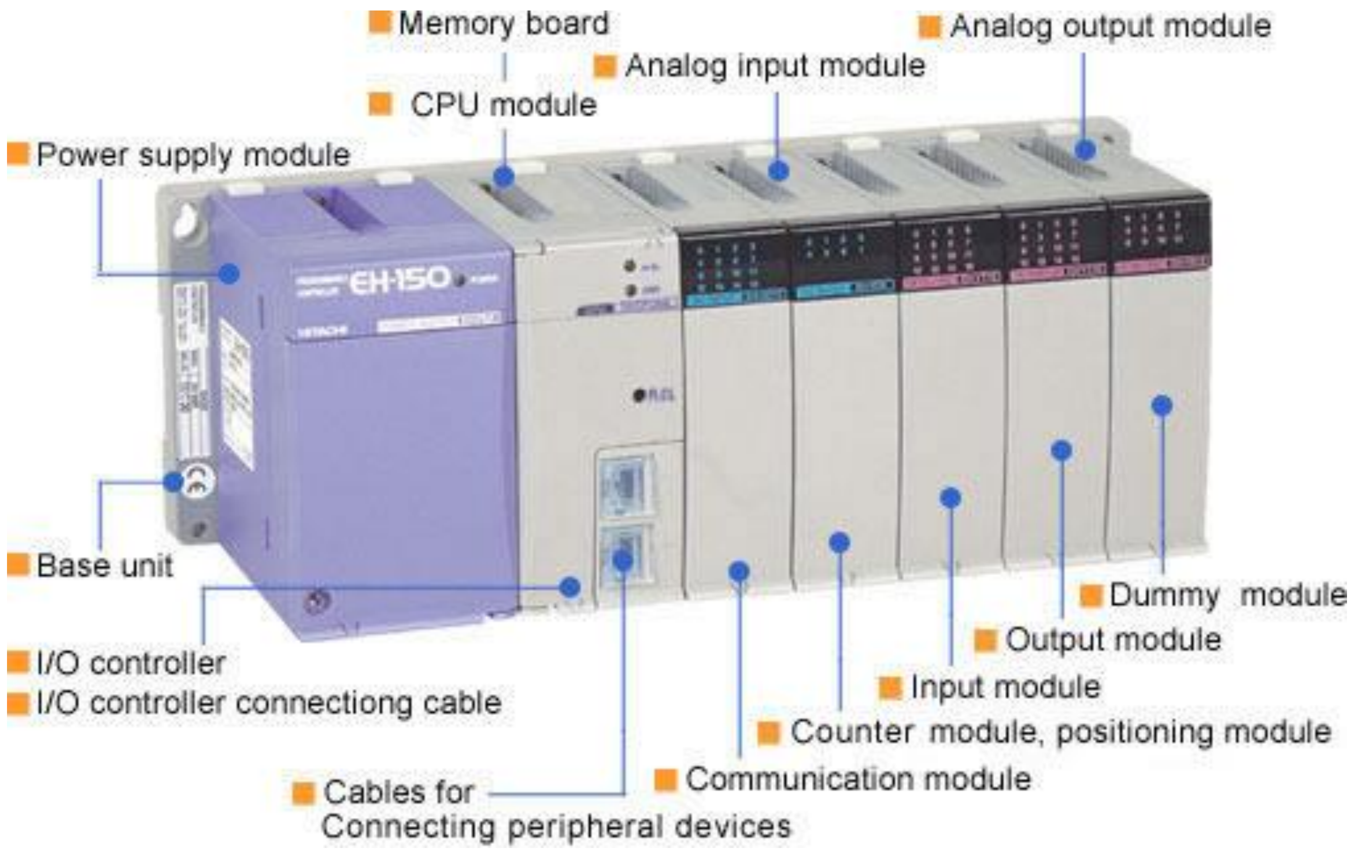
Programmable Logic Controllers (1)

Device platforms for implementing logic controllers:

- Dedicated hardware („hardwired logic“)
  - German: Verbindungsprogrammierte Steuerung (VPS)
- Programmable Logic Controllers (**PLC**)
  - German: Speicherprogrammierbare Steuerung (**SPS**)
- Distributed Control Systems (DCS)
  - Prozessleitsysteme (PLS)
- Industrial PCs (IPCs)
- Soft-PLCs

Programmable Logic Controllers (2)

Examples



© Hitachi, http://www.hitachi-ies.co.jp/english/products/plc/eh_150/product_range.htm

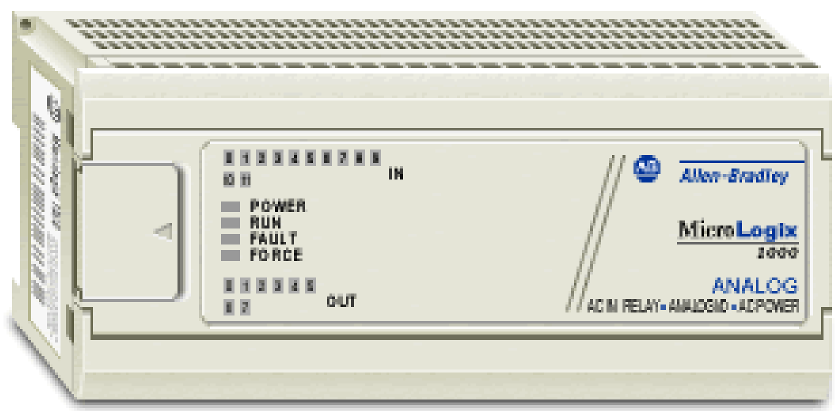
Programmable Logic Controllers (3)

Examples

© plcs.net, http://www.plcs.net/chapters/whatis1.htm, http://www.geometrixar.com/plc_programming

Programmable Logic Controllers (4)

Examples



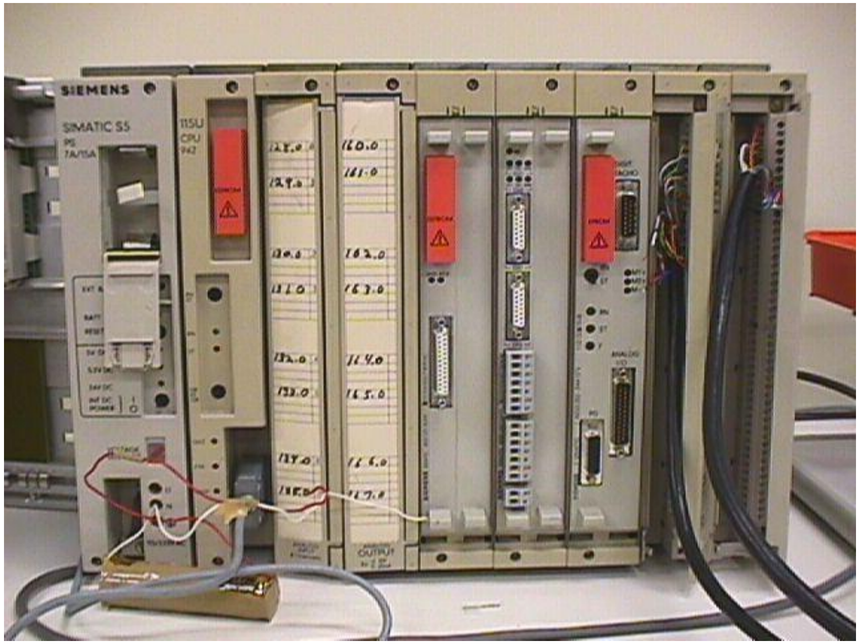
© Degensha, http://www.dengensha.com/site.cfm/nut-feeders.cfm

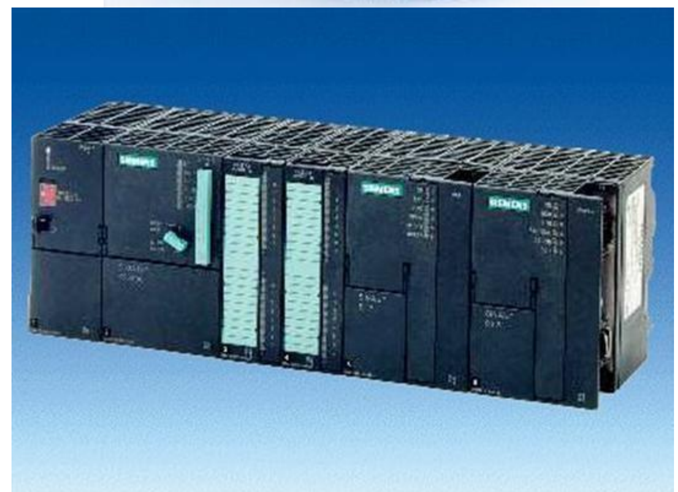
Programmable Logic Controllers (5)

Examples

Programmable Logic Controllers (6)

Examples





© https://engineering.purdue.edu/ManLab/plc.html, http://www.pacontrol.com/siemens-plc-training.html,

Programmable Logic Controllers (7)

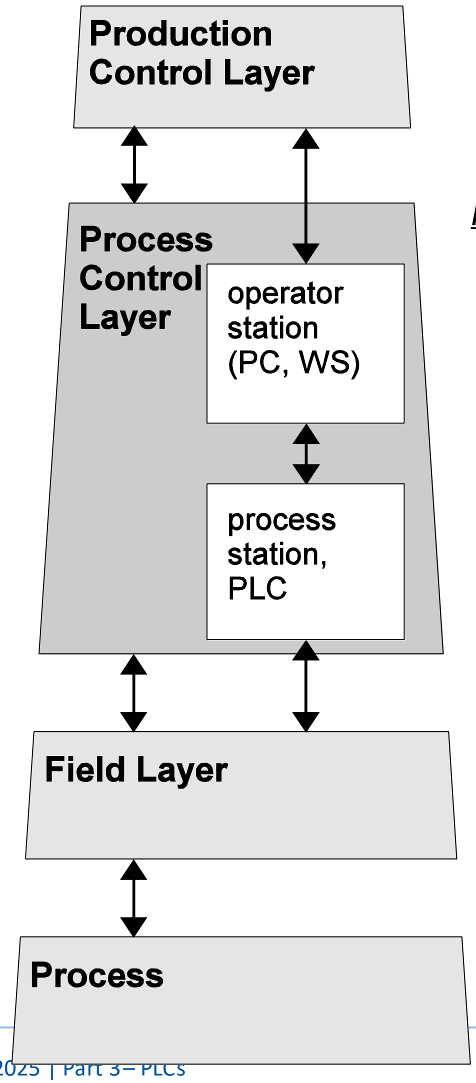
Examples

© http://www.hinrichsen-haustechnik.de/grafik/sps2.jpg

Programmable Logic Controllers (8)

Pyramid model of industrial automation

Layers:



Functions:

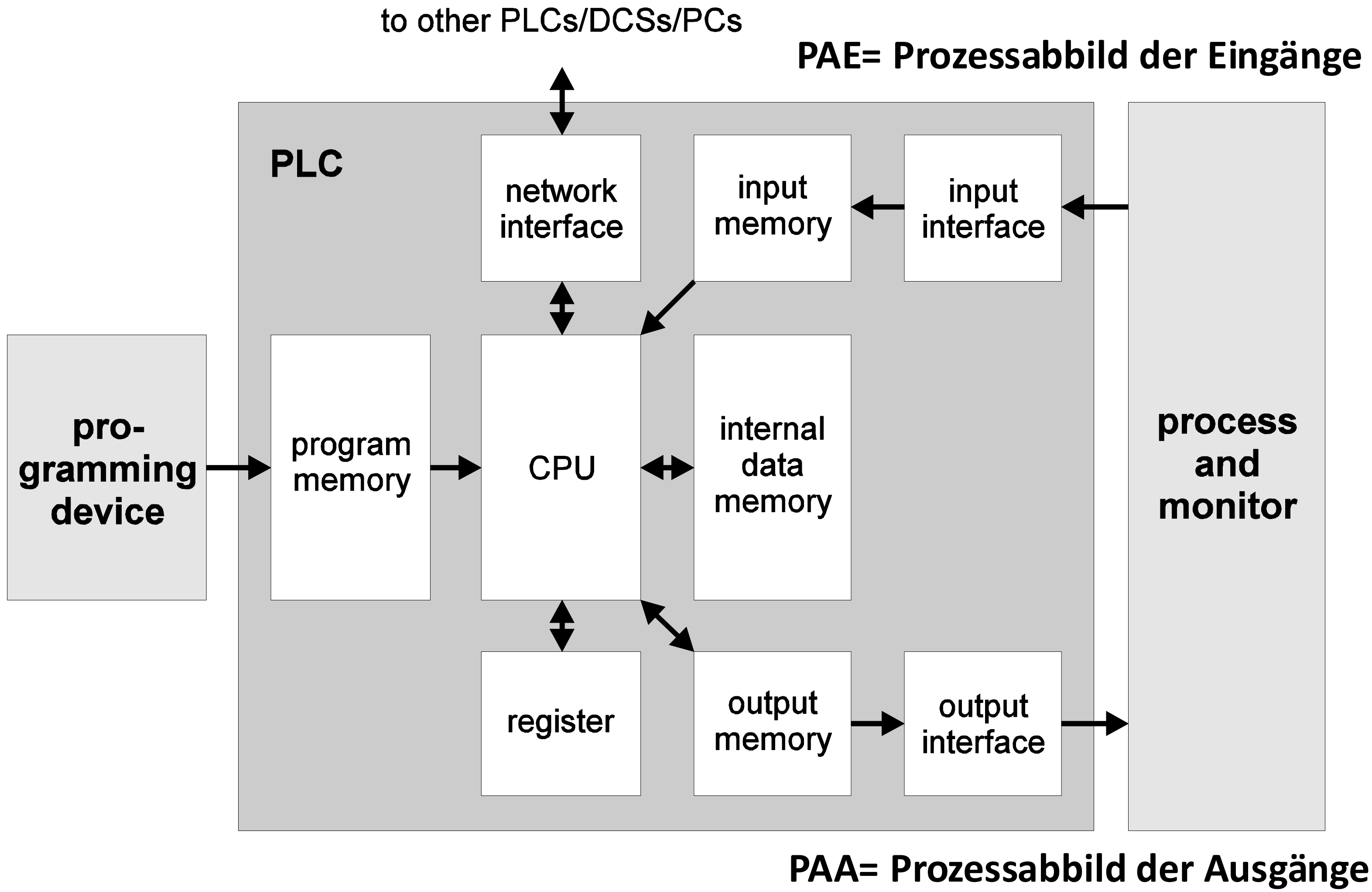
Higher Functions:
- advanced continuous control
- recipe control
- alarm handling/registration
- visualization
- displaying/operating

Basic functions:
- basic continuous control
- **sequence control**
- **threshold supervision**
- **safety trips**
- **interlocks**

- local displays
- local manual operating
- sensing
- actuating

Programmable Logic Controllers (9)

PLC architecture

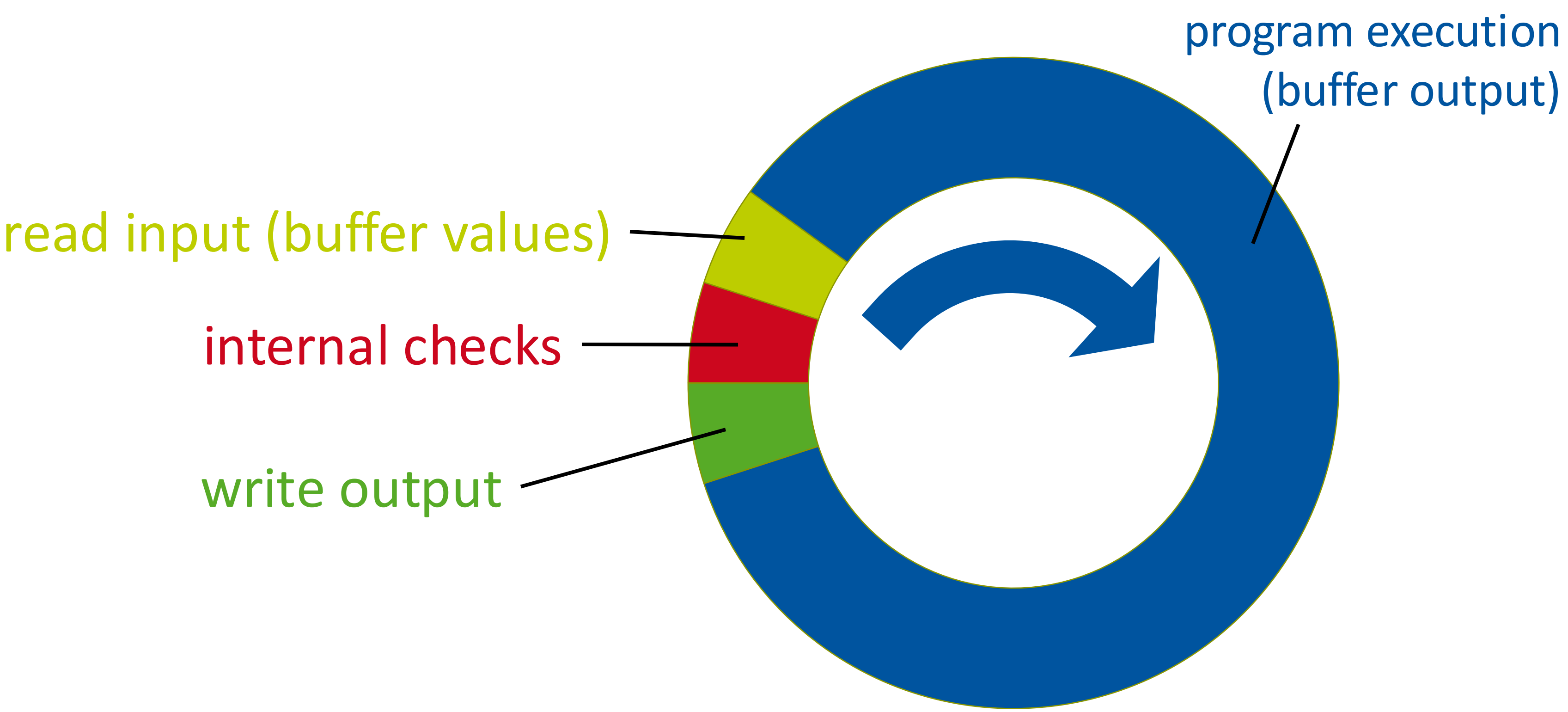


Programmable Logic Controllers (10)

PLC operating system: **Cyclic Scanning Mode**

Programmable Logic Controllers (10)

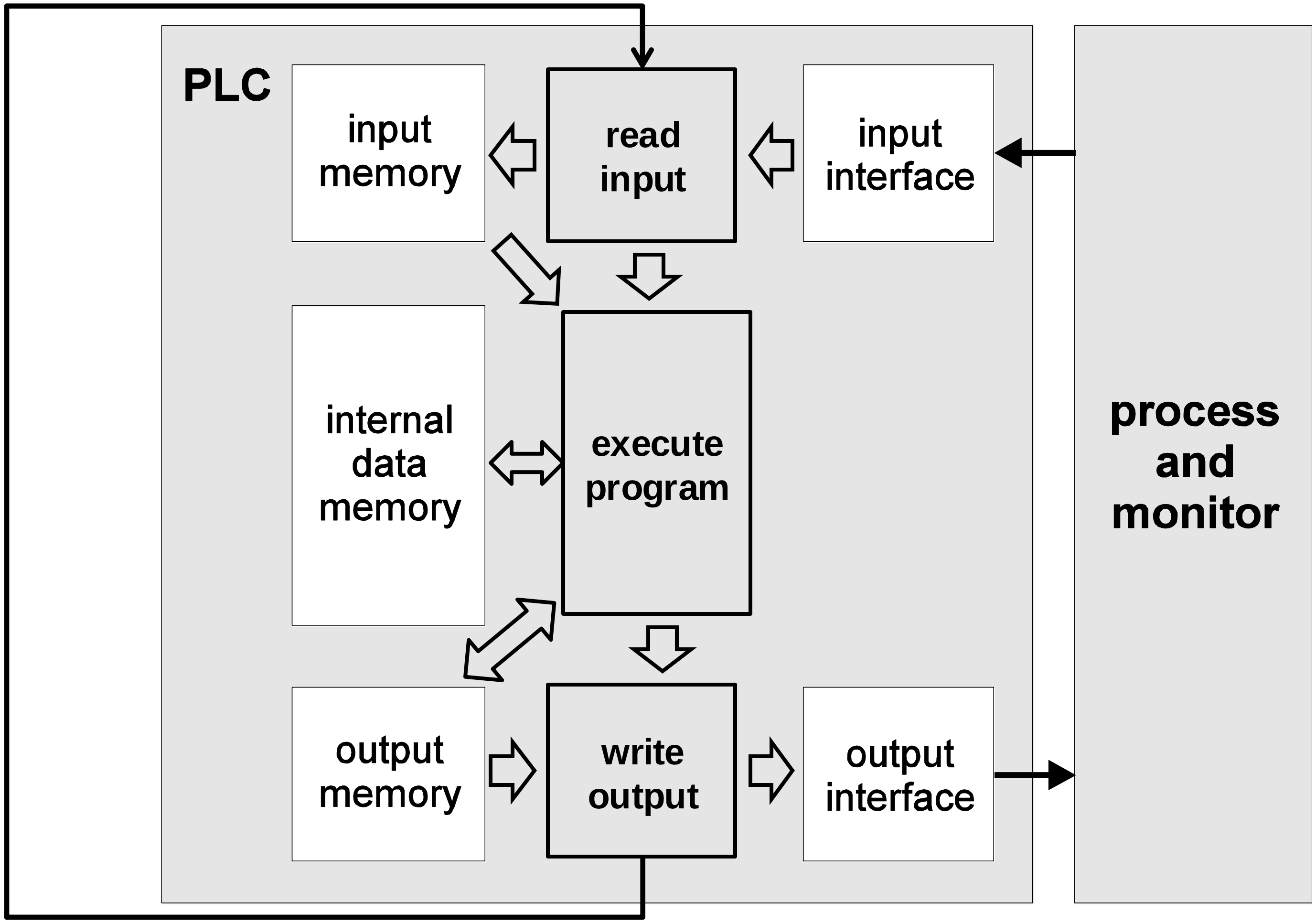
PLC operating system: **Cyclic Scanning Mode**



One turn around = cycle (≈ ms)

Programmable Logic Controllers (11)

Execution of cyclic scans in the PLC architecture:



Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

Programmable Logic Controllers (12)

The cycle time can vary depending on the program execution.

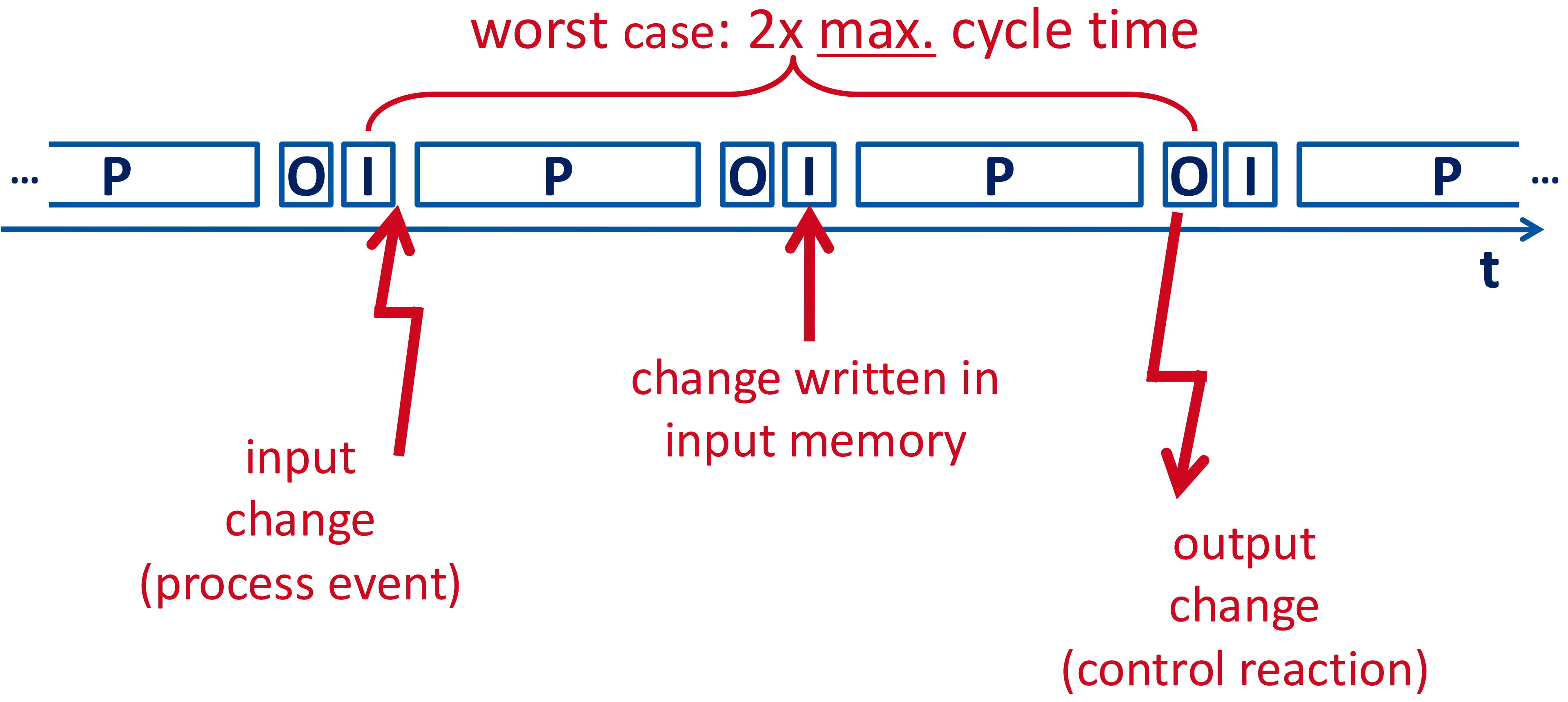
- Example:

```
                IF input=TRUE THEN GOTO END
                ELSE
              .
              . (many instructions)
              .
END:            END_IF
```

- Example: Program with one integer variable **i**, initially **i:= 0**.
- **i:=i+1**:
  - “normal” computer after execution i=1.
  - PLC: after first cycle i=1.
  - after second cycle i=2 …

Programmable Logic Controllers (13)

Maximal reaction time (delay between process event and corresponding control action)?



Programmable Logic Controllers (14)

- Maximal reaction time = 2 * maximal cycle time
- Cyclic scanning mode supports estimation of real-time behavior:
  - Determine longest program execution path.
  - Measure or estimate cycle time for this execution path.
  - Apply formula above.
- What to do, if reaction time is too long?
  - Optimize Program
  - Choose more performant PLC
  - Use interrupts

Content

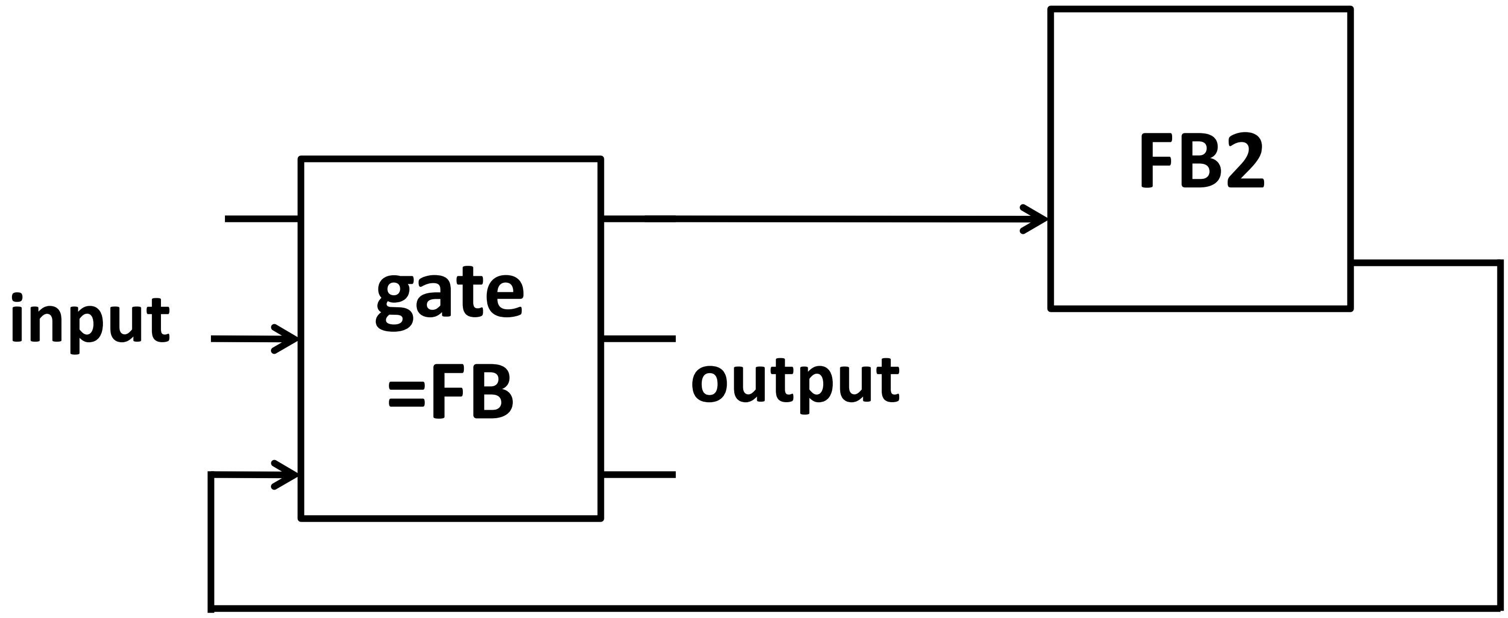
PLC programming languages

- Before 1992:
  - „Similar“ PLC languages
  - either resembling electrical design or assembly
  - Each PLC vendor used its own dialect
  - Disadvantages?
- 1992: IEC 61131 – International standardization of five PLC languages

Abbreviation	Name	German Name
FB	Function Block	Funktionsblocksprache
LD	Ladder Diagram Relay Ladder Logic (RLL)	Kontaktplan
IL	Instruction List	Anweisungsliste
ST	Structured Text	Strukturierter Text
SFC	Sequential Function Charts	Ablaufsprache

IEC 61131 languages: Function block language (FB)

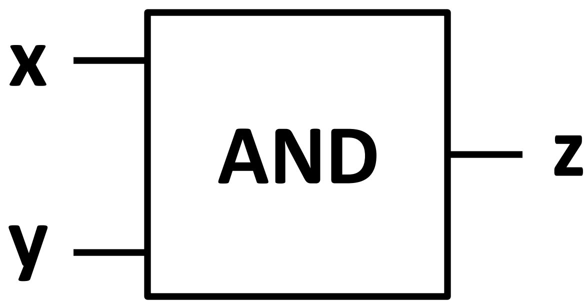
- Idea:
  - Resembling switching circuit design by ***gates***
  - blocks with input and output

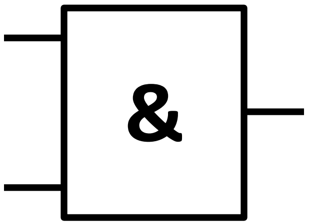


IEC 61131 languages: Function block language (FB)

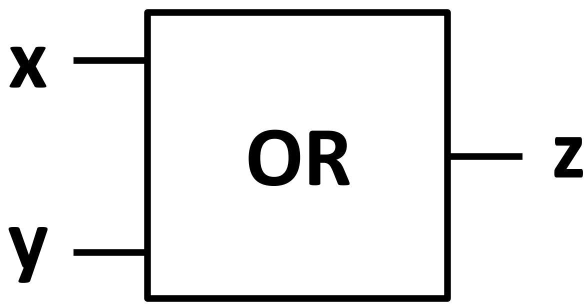
- Basic logic operators:

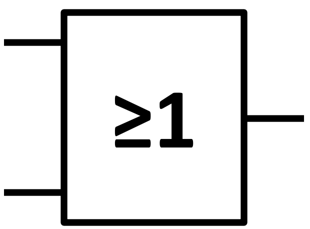
$$z = x \land y$$





$$z = x \lor y$$

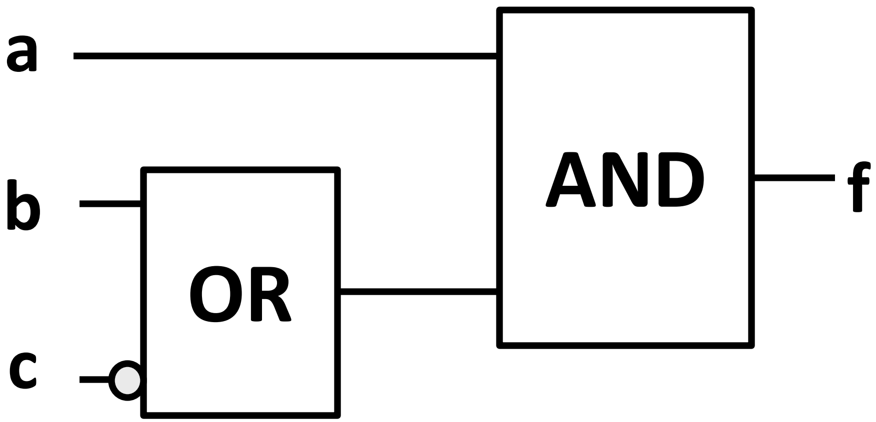




IEC 61131 languages: Function block language (FB)

- Example:

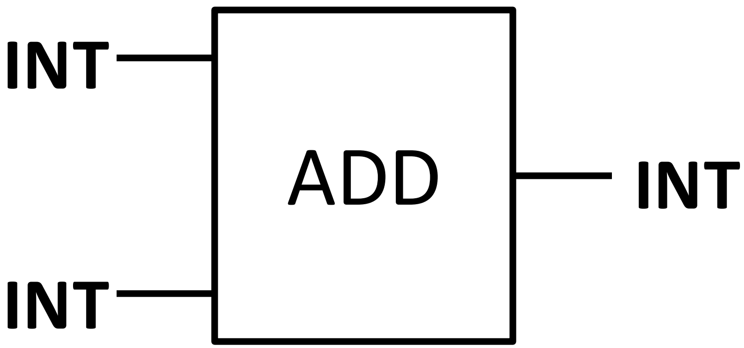
$$f := a \wedge (b \vee \overline{c})$$

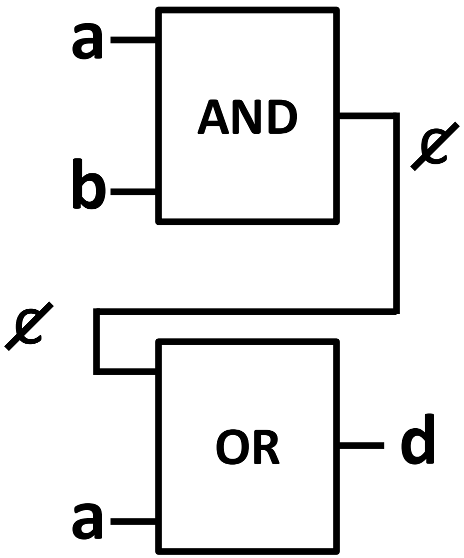


IEC 61131 languages: Function block language (FB)

- Blocks are not restricted to logic operators.

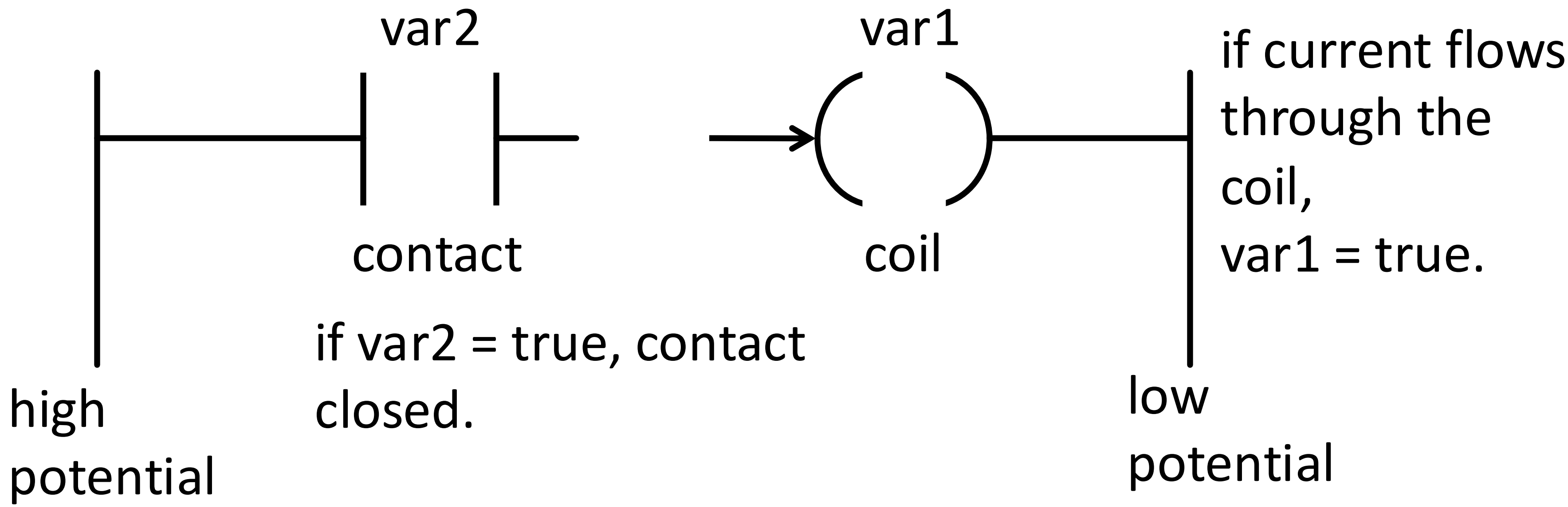
NAND, ADD, MULT, TIMER





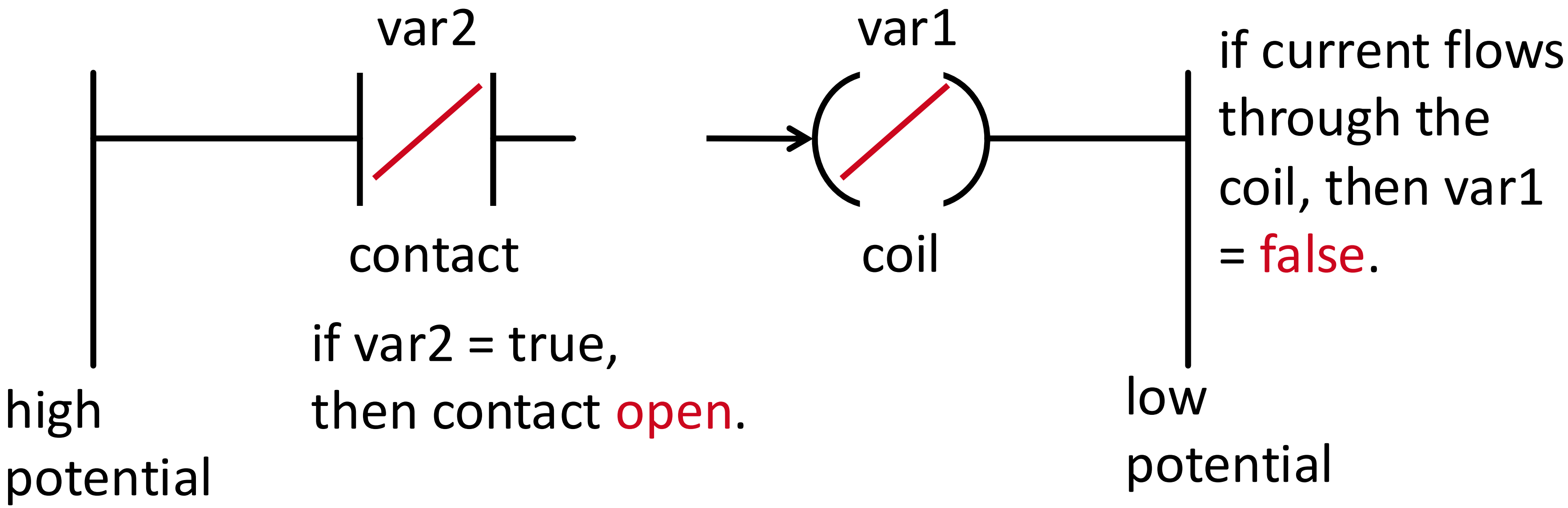
IEC 61131 languages: Ladder Diagram (LD)

- Idea:
  - Resembling switching circuit design by ***relays („schematics“)***
  - „contacts“ and „coils“



IEC 61131 languages: Ladder Diagram (LD)

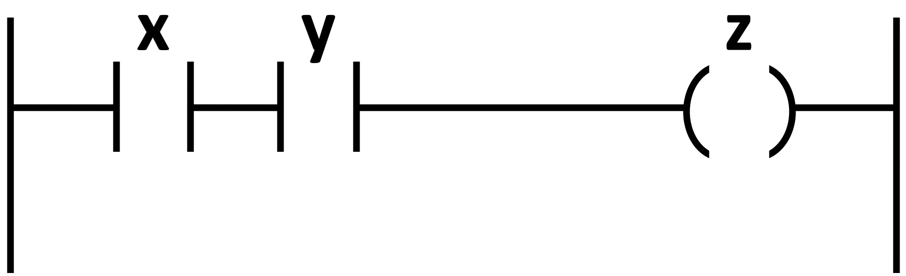
- Idea:
  - Resembling switching circuit design by ***relays („schematics“)***
  - „contacts“ and „coils“



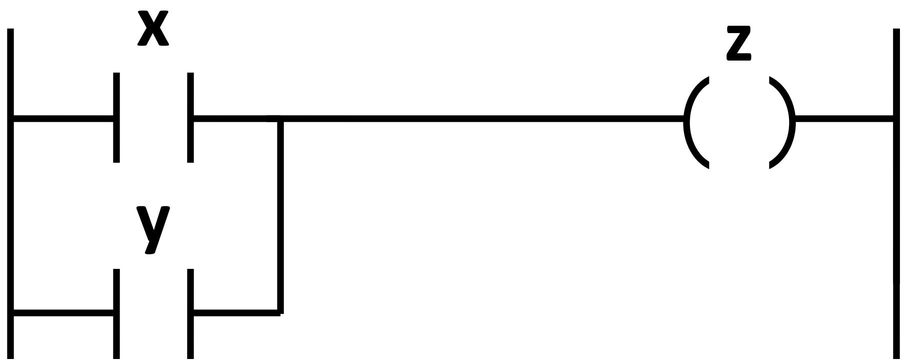
IEC 61131 languages: Ladder Diagram (LD)

- Basic logic operators:

$z = x \wedge y$



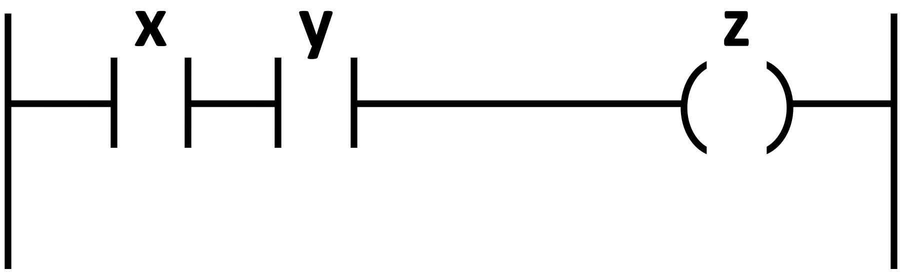
$z = x \vee y$



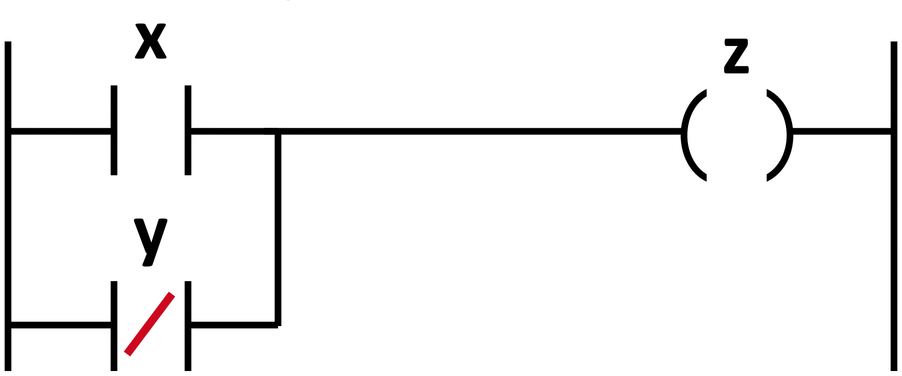
IEC 61131 languages: Ladder Diagram (LD)

- Basic logic operators:

$$z = x \wedge y$$



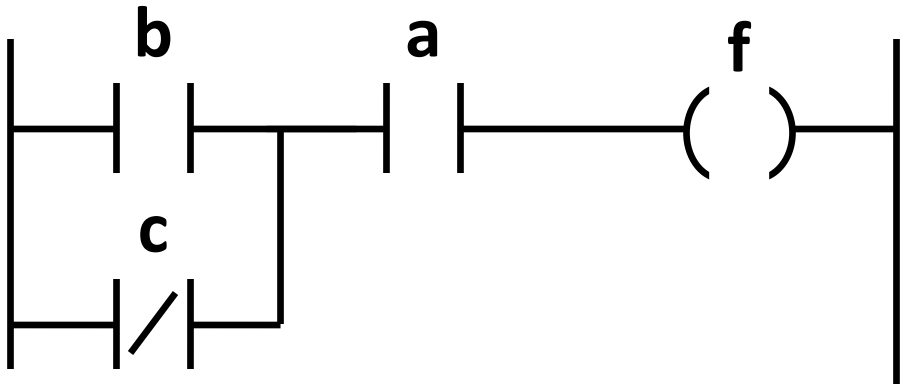
$$z = x \vee \overline{y}$$



IEC 61131 languages: Ladder Diagram (LD)

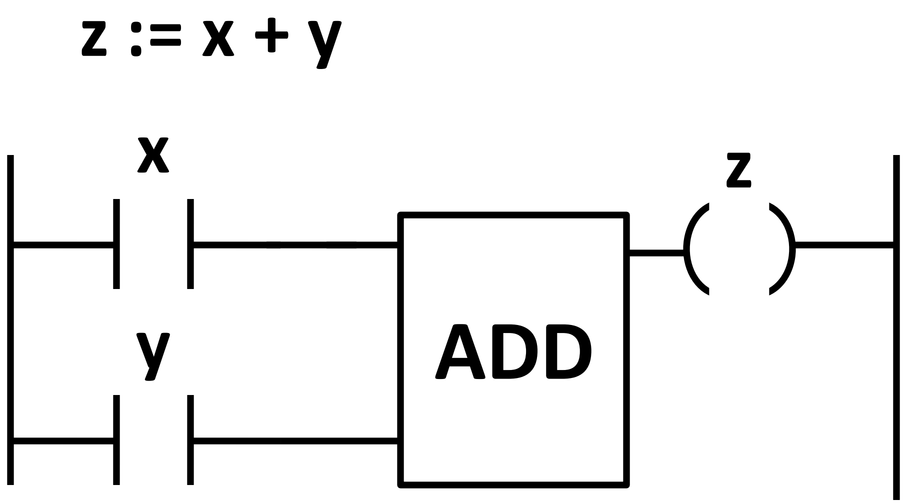
- Example:

$$f = a \wedge (b \vee \overline{c})$$



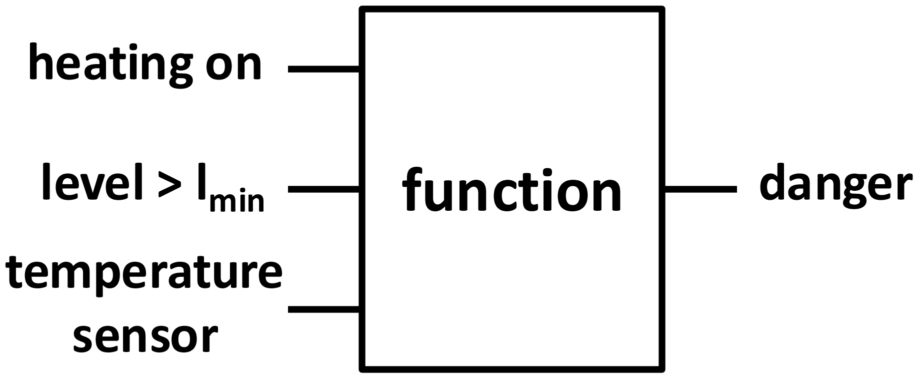
IEC 61131 languages: Ladder Diagram (LD)

- Function blocks can be integrated into LD programs:



IEC 61131: Functions vs. Function Blocks

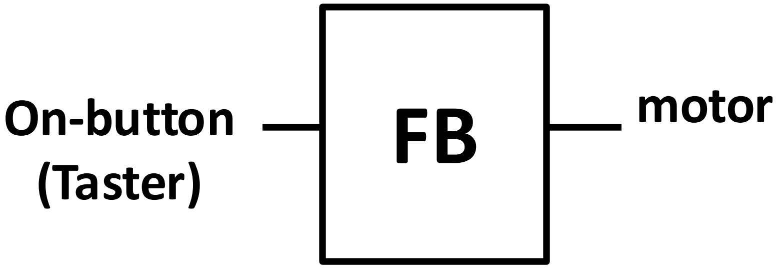
- Example of a „Function“:



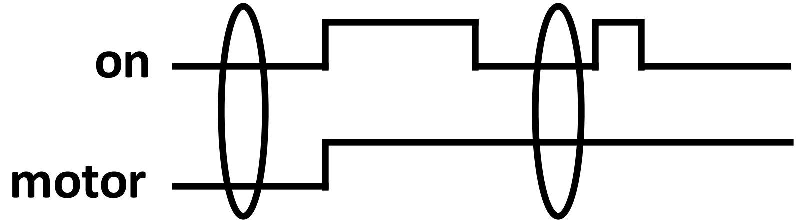
static

danger = f(input)

- Example of a „Function Block“:

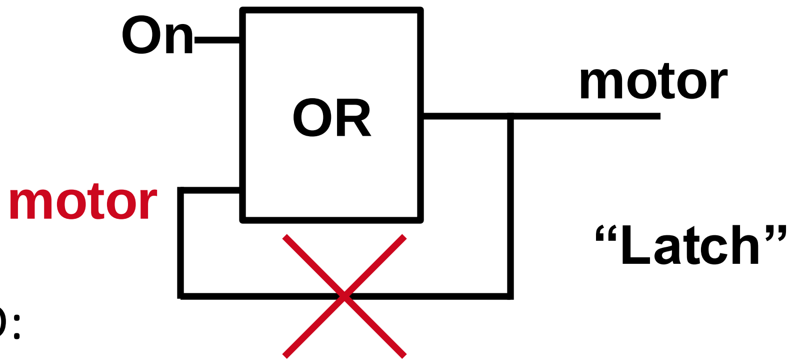


dynamic, internal state
(“Memory”)

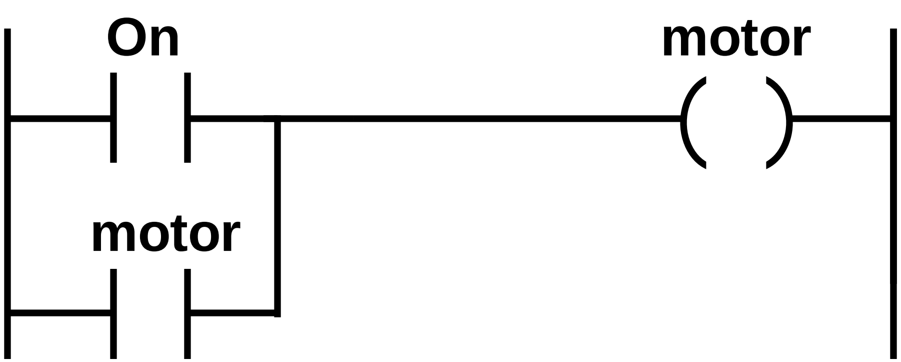


IEC 61131 Standard Function Blocks: Latches (1)

- How can the desired functionality be programmed using only the already known basic operators?
  - FB:



  - LD:

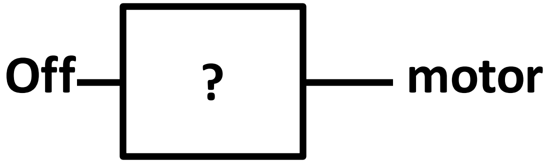


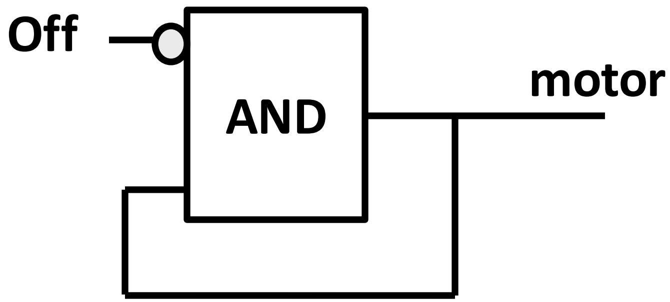
Abbreviation:

S:= Set

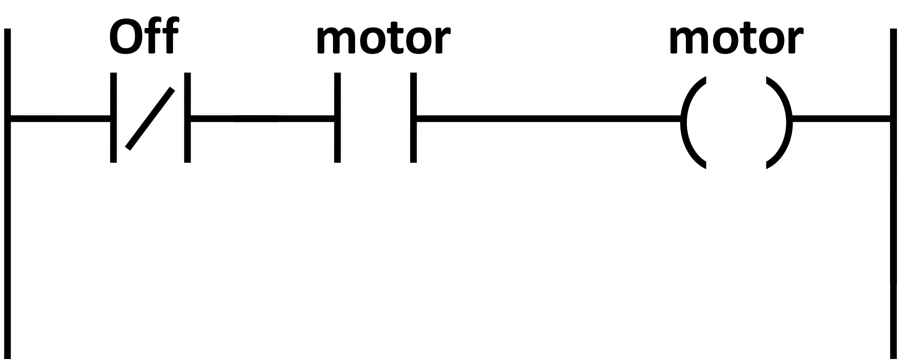
IEC 61131 Standard Function Blocks: Latches (2)

- Switching off the motor?
  - FB:



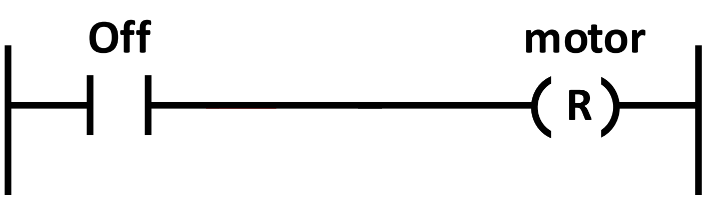


  - LD:



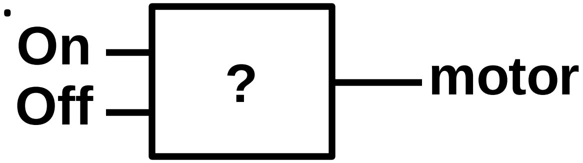
Abbreviation:

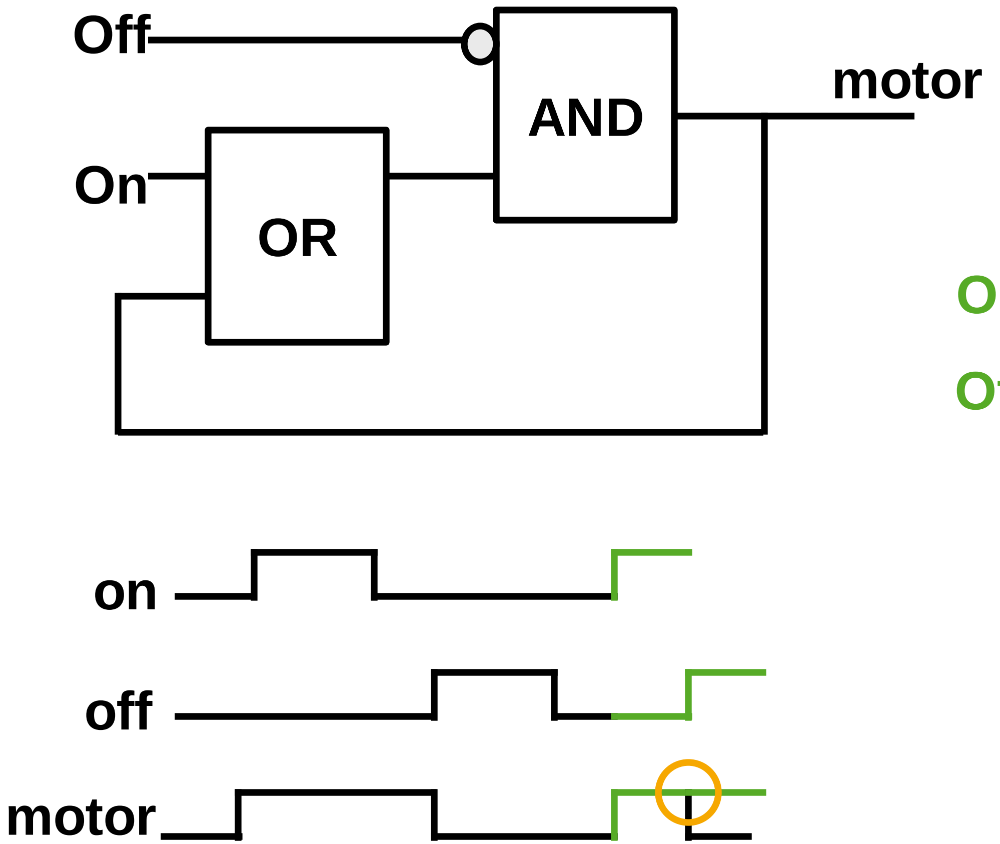
R:= Reset

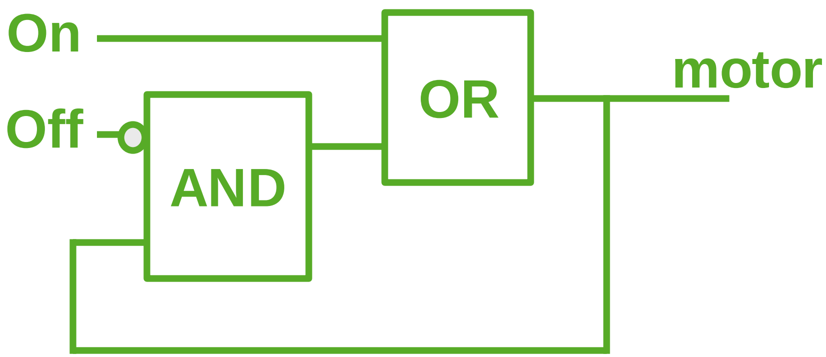


IEC 61131 Standard Function Blocks: Bistables (1)

- Combination of both latches:

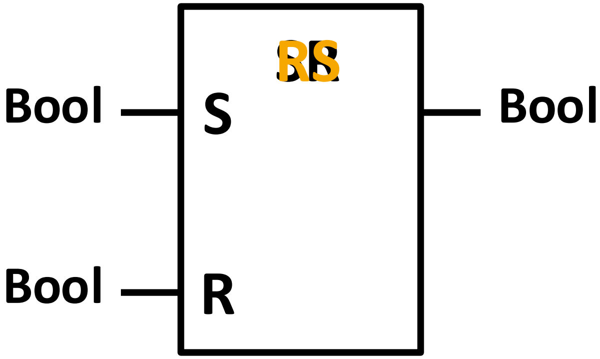






IEC 61131 Standard Function Blocks: Bistables ctd.

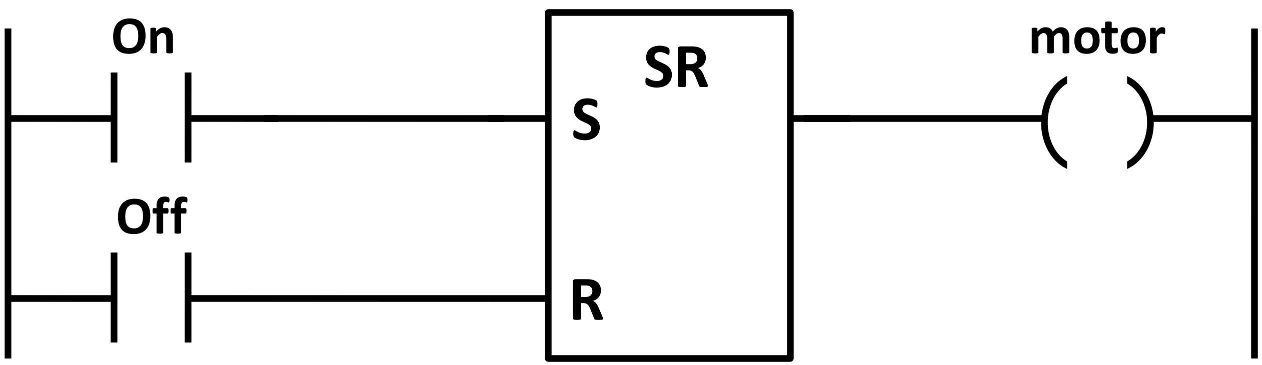
- Bistable as standard Function Block:



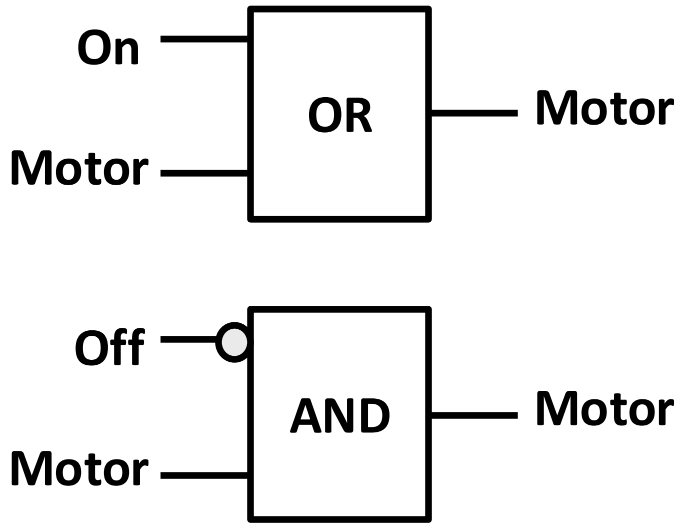
Set dominant bistable.

Reset dominant bistable.

- LD:

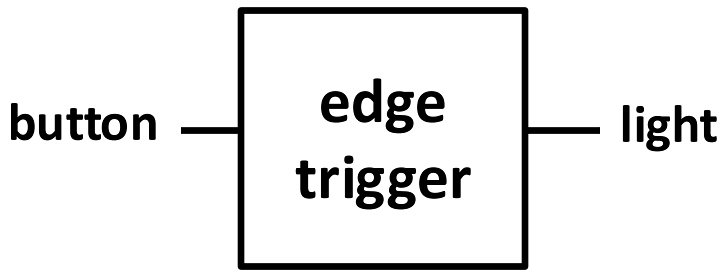


IEC 61131 Standard Function Blocks: Bistables ctd.

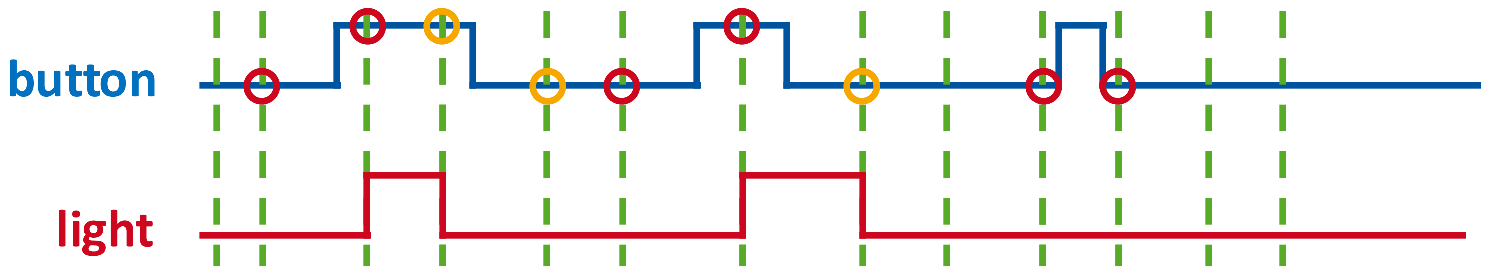


IEC 61131 Standard Function Blocks: Edge Triggers

- Problem: A change of a variable shall be detected or indicated.
- Example:



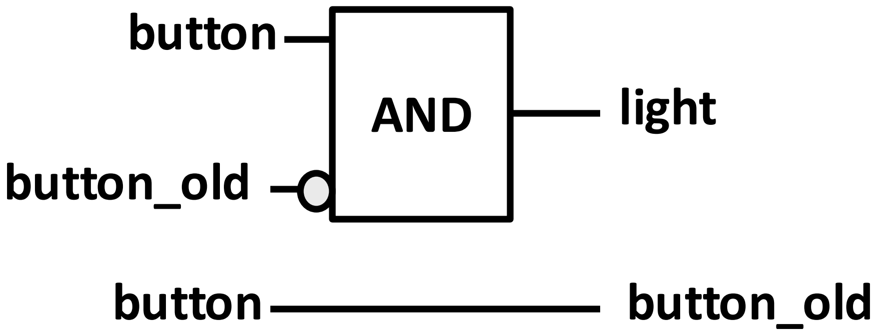
- Desired function:



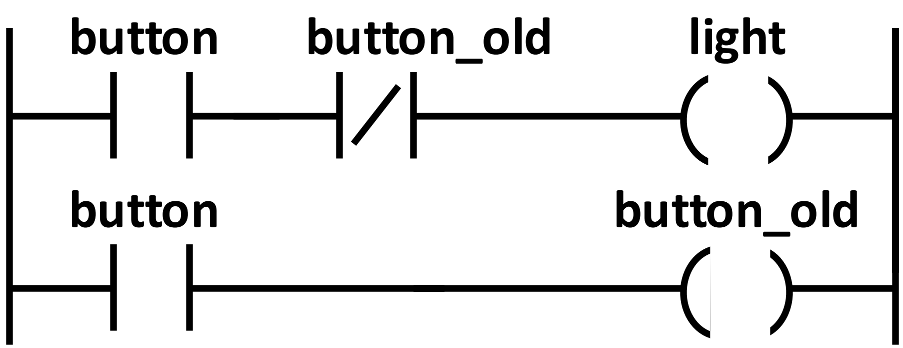
IEC 61131 Standard Function Blocks: Edge Triggers (ctd.)

Realization:

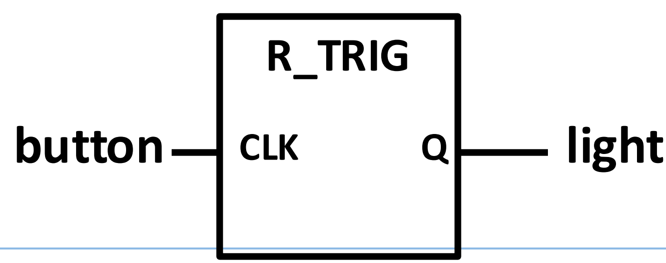
- FB:



- LD:

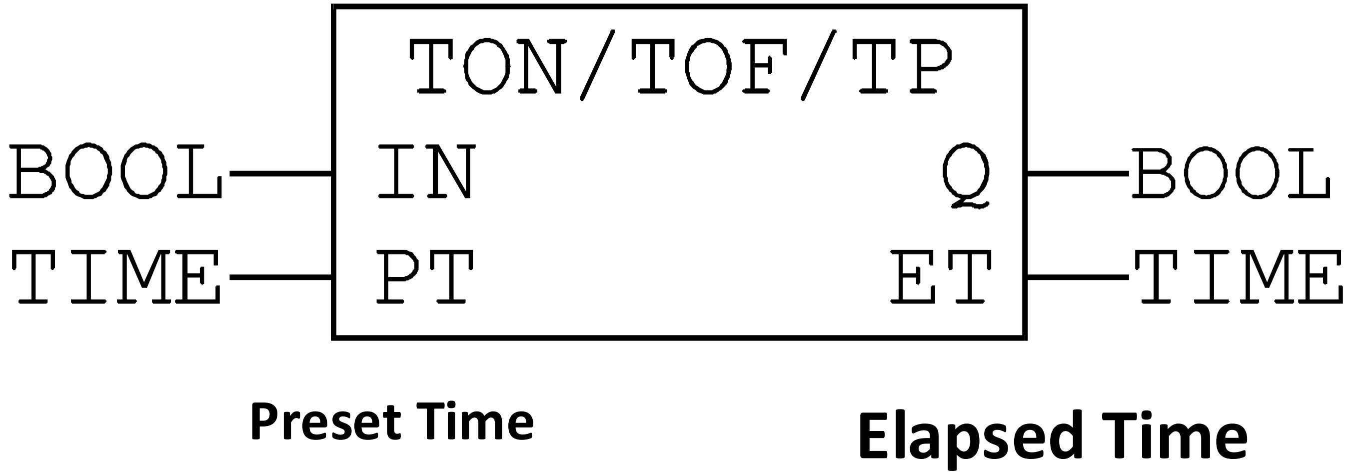


- Block symbol:



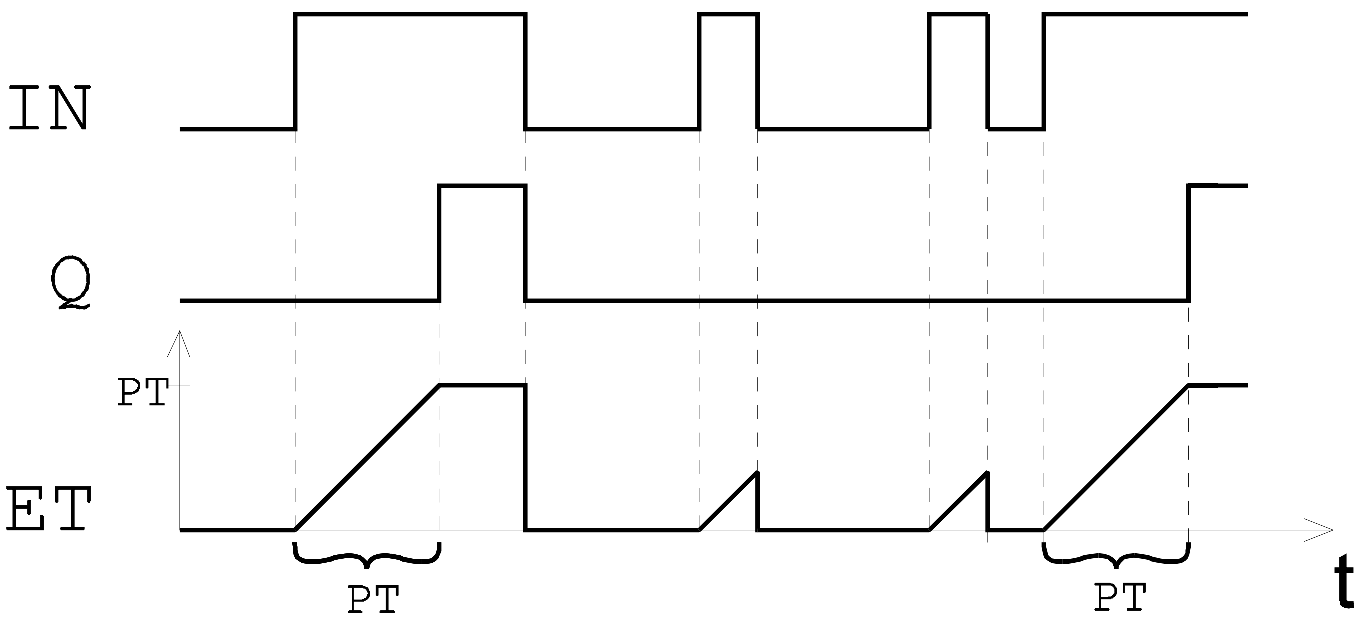
IEC 61131 Standard Function Blocks: Timers

- IEC 61131 defines three different timer function blocks:
  - TON Timer On Delay
  - TOF Timer Off Delay
  - TP Timer Pulse



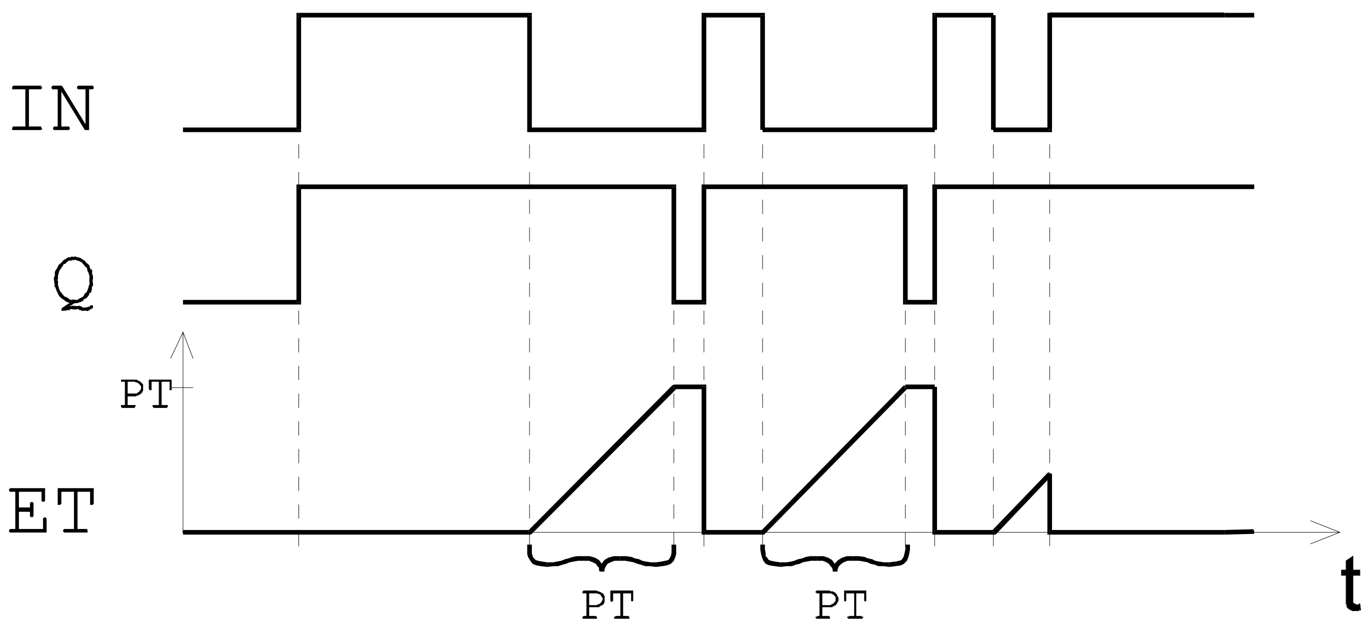
IEC 61131 Standard Function Blocks: Timers (ctd.)

- Timing diagram for TON:



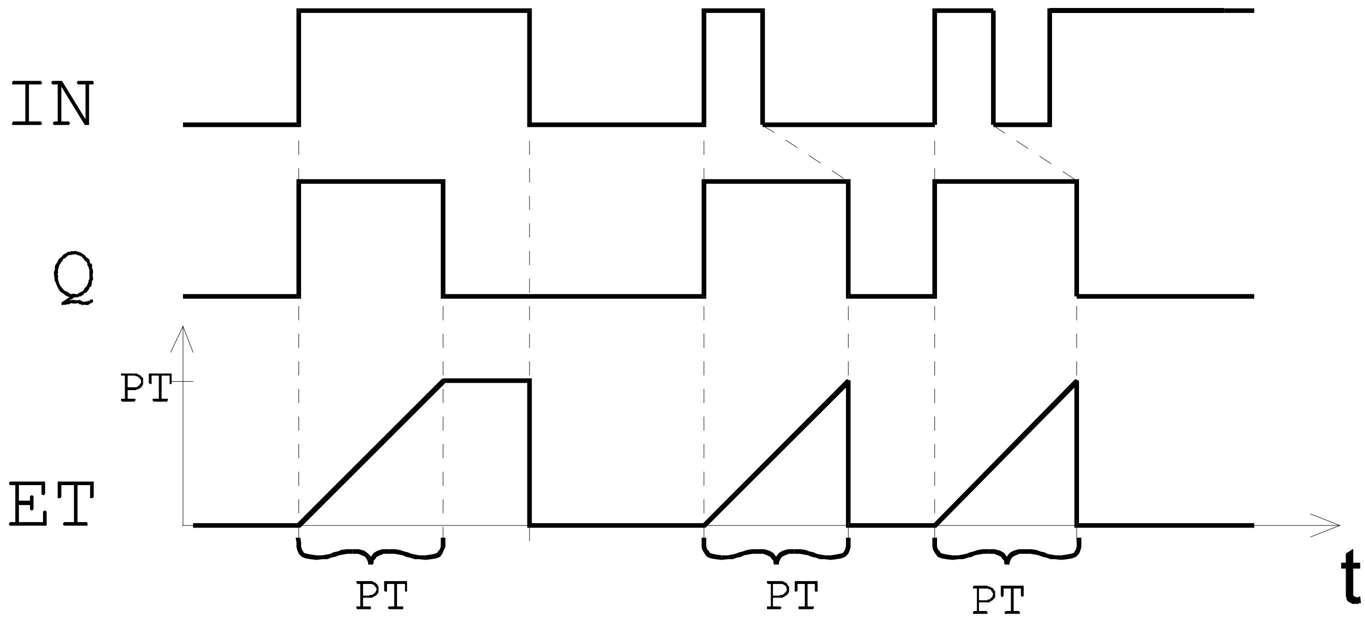
IEC 61131 Standard Function Blocks: Timers (ctd.)

- Timing diagram for TOF:



IEC 61131 Standard Function Blocks: Timers (ctd.)

- Timing diagram for TP:



IEC 61131 languages: Instruction List

Anweisungsliste (AWL)

- Textual programming language („list of instructions“)
- ~ Assembly language
- Instruction syntax:

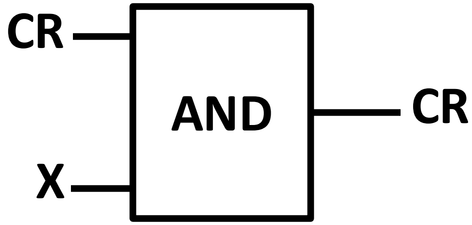
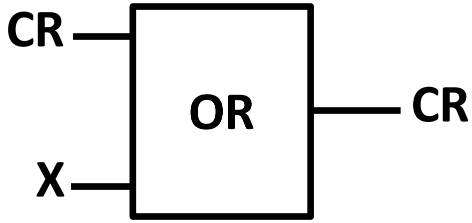
(label:)	operator	operand
e.g.	AND	sensor1

- One-register or accumulator architecture
- Accumulator is called CR („current result“)

Aktuelles Ergebnis (AE)

IEC 61131 languages: Instruction List (ctd.)

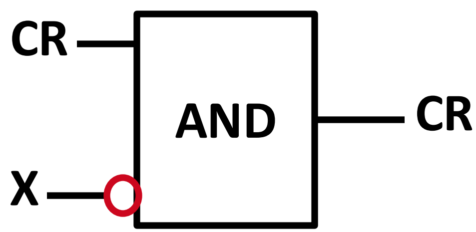
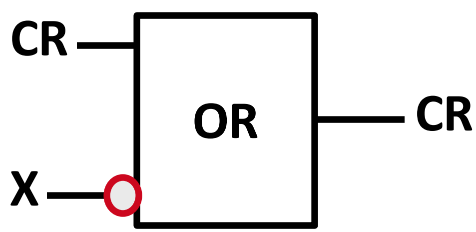
- Basic operators:

LD	X	"Load"	CR:=X
AND	X		$CR:=CR \wedge X$
OR	X		$CR:=CR \vee X$

Also: XOR (exclusive OR)

IEC 61131 languages: Instruction List (ctd.) (2)

- Basic operators with negation:

LDN	X	“Load”	$CR{:=}\overline{X}$
ANDN	X		$CR{:=}CR \wedge \overline{X}$
ORN	X		$CR{:=}CR \vee \overline{X}$

IEC 61131 languages: Instruction List (ctd.)

- Basic operators: Math instructions

GT	X	"Greater-than"	$CR := \begin{cases} \mathbf{1 \text{ if } CR > X} \\ \mathbf{0 \text{ else}} \end{cases}$
EQ	X	"Equal"	$CR := \begin{cases} \mathbf{1 \text{ if } CR = X} \\ \mathbf{0 \text{ else}} \end{cases}$
MOD	X	"Modulo"	CR:= CR mod X

Also: LT, GE & LE (Less or Equal) , NE (Not Equal), ADD, SUB, MUL, DIV

IEC 61131 languages: Instruction List (ctd.)

- Basic operators (ctd.):

ST	X	"Store"	X:=CR
S	X	"Set"	$X := \begin{cases} \mathbf{1 \text{ if } CR=1} \\ \mathbf{X \text{ if } CR=0} \end{cases}$
R	X	"Reset"	$X := \begin{cases} \mathbf{0 \text{ if } CR=1} \\ \mathbf{X \text{ if } CR=0} \end{cases}$

IEC 61131 languages: Instruction List (ctd.)

- Basic operators (ctd.): GOTO instructions

JMP	label	"jump"
JMPC	label	"jump conditionally" Go to label if CR=1
JMPCN	label	"jump conditionally negated" Go to label if CR=0

Also: RET/RETC/RETCN (Return, i.e. jump to the end of the program)

IEC 61131 languages: Instruction List (ctd.)

- Basic logic operations in IL:

$z := x \land y$	$z := x \lor \neg y$	$z := x + y$
LD X	LD X	LD X
AND Y	ORN Y	ADD Y
ST Z	ST Z	ST Z

- declaration of variables

VAR_INPUT	VAR_OUTPUT	VAR
X: Bool;	Y: Bool:=TRUE;	Internal: Bool:=TRUE;
END_VAR	END_VAR	END_VAR

IEC 61131 languages: Instruction List (ctd.)

- Example: $f := a \wedge (b \vee \neg c)$

```
LD     b          LD     a
ORN    c          AND(   b
AND    a          ORN    c
ST     f          )
                  ST     f
```

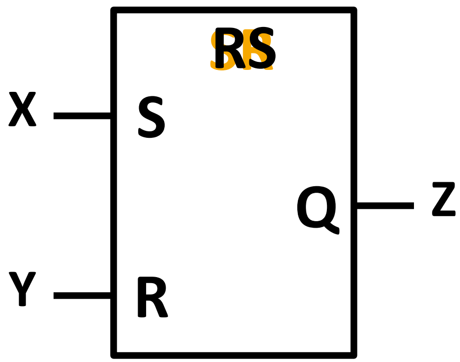
IEC 61131 languages: Instruction List – Example

(* Methode 1 *)	(* Methode 2 *)	(* Methode 3 *)
(* Parameterversorgung: *)	LD t#500ms ST Zeit1.PT LD Frei ST Zeit1.IN	LD t#500ms PT Zeit1 LD Frei
(* Aufruf: *) CAL Zeit1(IN:=Frei, PT:=t#500ms, Q=>Aus, (*Ausgangsp*) ET=>Wert(*Ausgangsp*))	CAL Zeit1	IN Zeit1
	(* Auswertung der Ausgangsparam. *) LD Zeit1.Q ST Aus LD Zeit1.ET ST Wert	

[Source: John & Tiegelkamp: SPS-Programmierung mit IEC 61131-3, Springer-Verlag, 4. Aufl., S. 115]

IEC 61131 languages: Instruction List (ctd.)

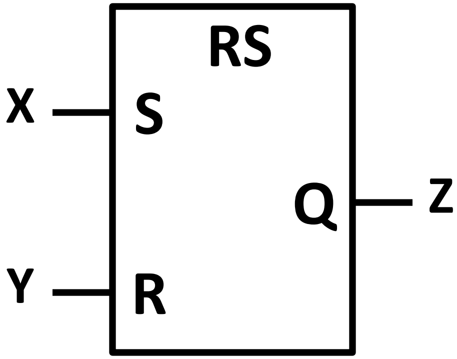
- Bistables?



RS:	SR:
LD X	LD Y
S Z	R Z
LD Y	LD X
R Z	S Z

IEC 61131 languages: Instruction List (ctd.)

- Function block calls in IL: Example RS




```
VAR
myBistable : RS;
x,y,z : BOOL;
END_VAR
```

Variant 1:

```
LD      x
ST      myBistable.S
LD      y
ST      myBistable.R
...
CAL     myBistable
LD      myBistable.Q
ST      z
```

Variant 2:

```
CAL     myBistable(S:=x,R:=y)
LD      myBistable.Q
ST      Z
```


Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

IEC 61131 languages: Instruction List (ctd.)

- Function block calls in IL: Example TON

```
VAR
myTimer:TON;
start,motor:BOOL;
END_VAR


LD  start
ST  myTimer.IN
CAL myTimer(PT:= T#1min30s)
...
LD  myTimer.Q
S   motor
```

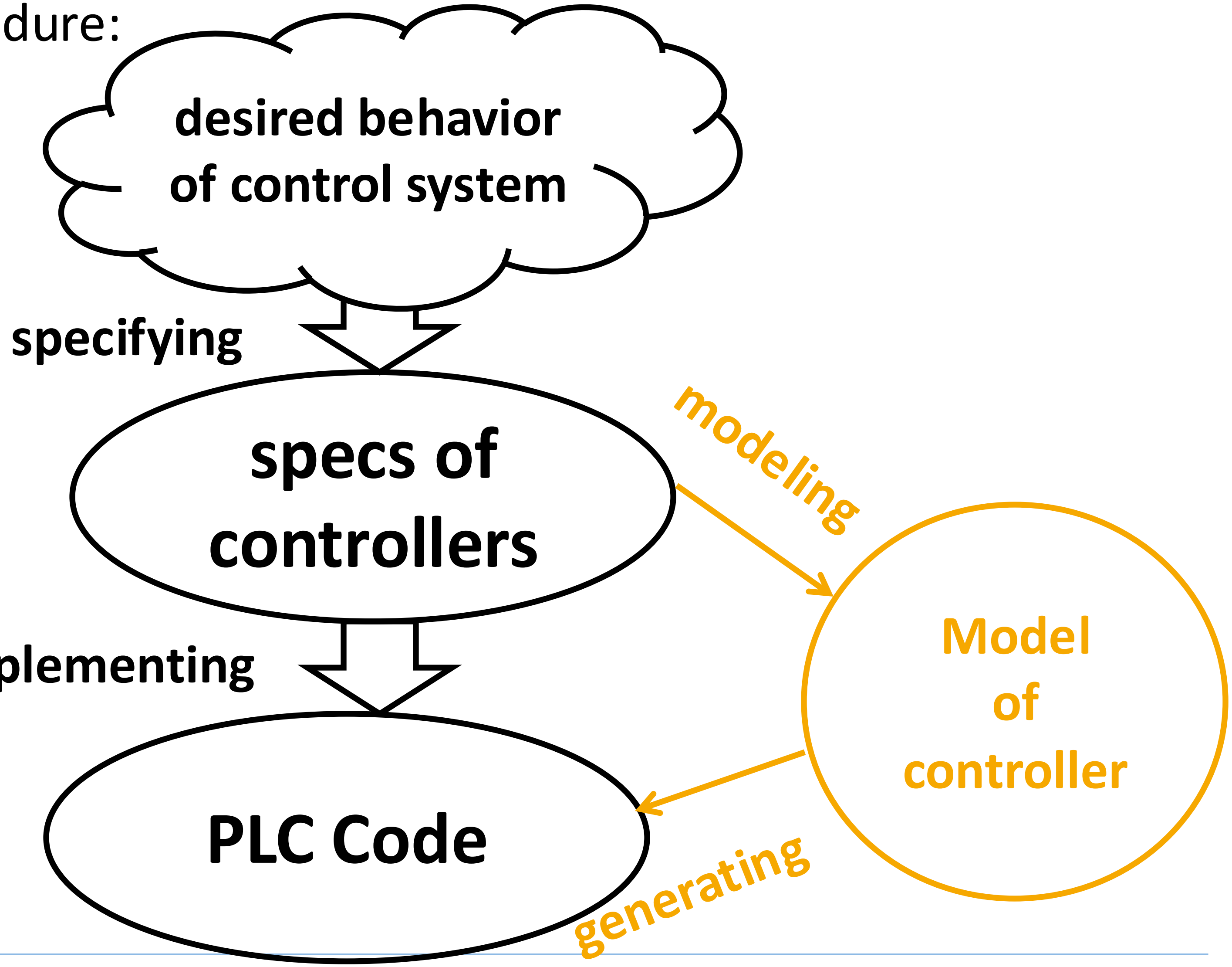

Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

Content

1. Logic Control
2. PLC Technology
3. Programming languages
   - Function Block Diagram (FBD)
   - Ladder Diagram (LD)
   - Instruction List
4. Model-based Design
5. Sequential Controllers
   - Sequential Function Charts

Model-based development of logic control programs

- Usual procedure:



Model-based development of logic control programs (ctd.)

- Problems with this approach:

1. Does the specification represent the originally desired behavior?
2. Is the specification complete and consistent
3. How to implement the specification?

- Instead: Build a model of the controller first.

1. It helps discussing/clarifying the specification.
2. It can be analyzed algorithmically.
3. Automatic code generation becomes possible.

Model-based development of logic control programs (ctd.)

- Our model: Moore and Mealy automata.

$$A = (X, U, Y, f, g, x_0)$$

X: Set of discrete states

U: input alphabet

Y: output alphabet

f: $X \times U \rightarrow X$

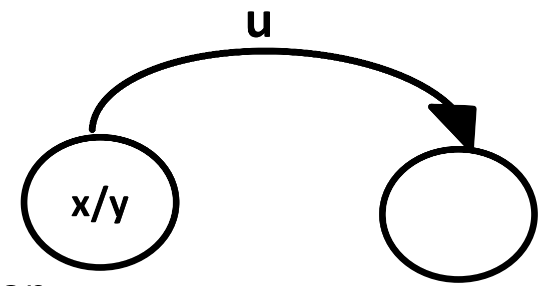
g: output function

x_0: initial state

state trans. fct.:

$x' = f(x,u)$

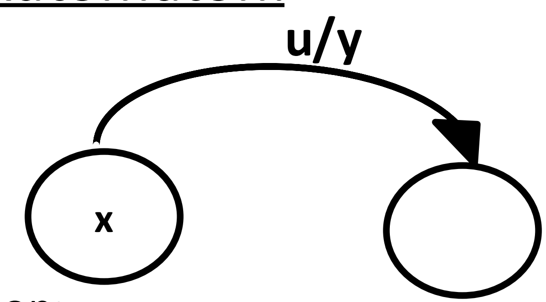
<u>Moore Automaton:</u>



Function:

$X \rightarrow Y$ $\quad$ $y=g(x)$

<u>Mealy Automaton:</u>

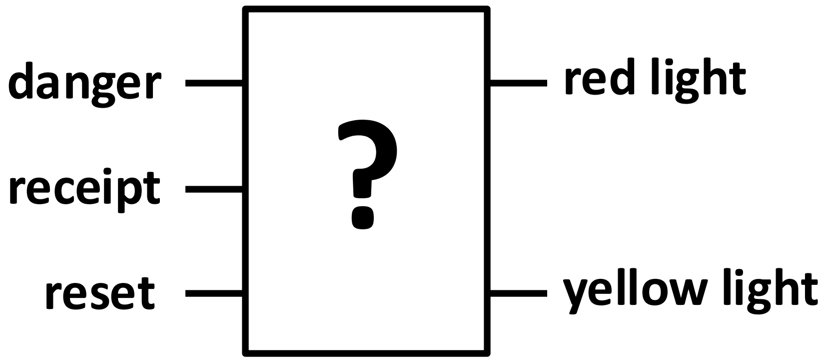


Function:

$X \times U \rightarrow Y$ $\quad$ $y=g(x,u)$

Model-based development of logic control programs (ctd.)

Our example for model-based design: A function block for alarm handling



Specification:

1. The RED light must be switched on as soon as DANGER becomes TRUE.
2. When the operator presses the RECEIPT button, the RED light is switched off and the YELLOW light is switched on.
3. When the danger is over, the YELLOW light can be switched off by pressing the RESET button.

Model-based development of logic control programs (ctd.)

Modeling the example

1. Variables, type of automata

$$x = (\text{red}, \text{yellow}) \in \{\underbrace{(0,0)}_{x_0}, \underbrace{(1,0)}_{x_1}, \underbrace{(0,1)}_{x_2}\}$$

$$u = (\text{danger}, \text{receipt}, \text{reset}) \in \mathbb{B}^3$$

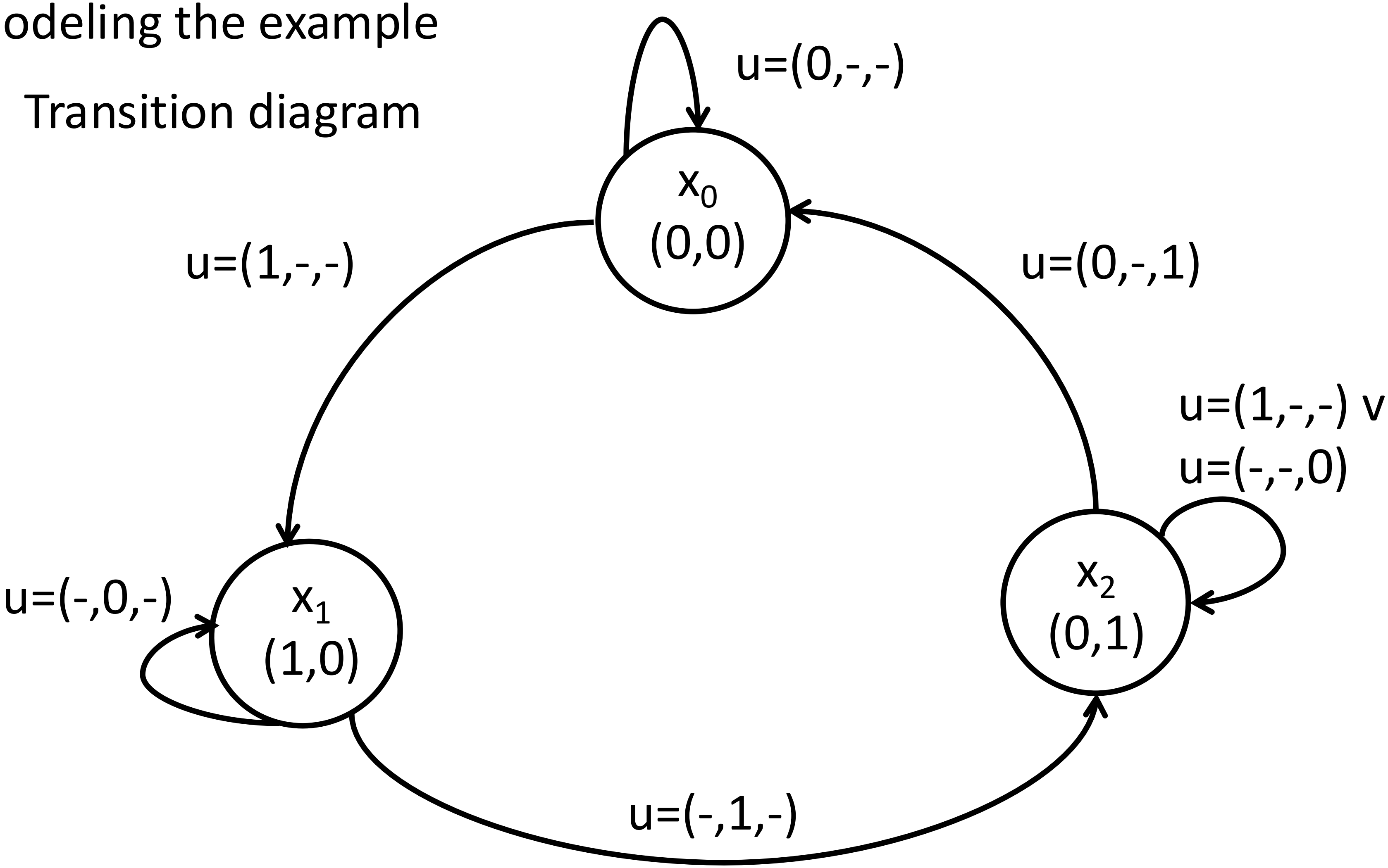
$$y = x$$

Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

Model-based development of logic control programs (ctd.)

Modeling the example

2. Transition diagram



Model-based development of logic control programs (ctd.)

Code generation: Your suggestions?

Model-based development of logic control programs (ctd.)

Code generation

Systematic approach: Divide program into two parts:

1. Check and perform transitions
2. Set or reset output

Model-based development of logic control programs (ctd.)

Code Generation, Solution: **Variable declarations**

```
VAR_INPUT
  danger,receipt,reset:BOOL;
END_VAR

VAR_OUTPUT
  red,yellow:BOOL;
END_VAR

VAR
  X0:BOOL:=TRUE;
  X1,X2:BOOL:=FALSE;
END_VAR
```

Model-based development of logic control programs (ctd.)

Code Generation, Solution: **Check and perform transitions**

```
LD   x0
  AND  danger
  R    x0
  S    x1
  JPMC output

  LD   x1
  AND  receipt
  R    x1
  S    x2
  JPMC output

  LD   x2
  ANDN danger
  AND  reset
  R    x2
  S    x0
```

Model-based development of logic control programs (ctd.)

Code Generation, Solution: **Set/reset outputs**

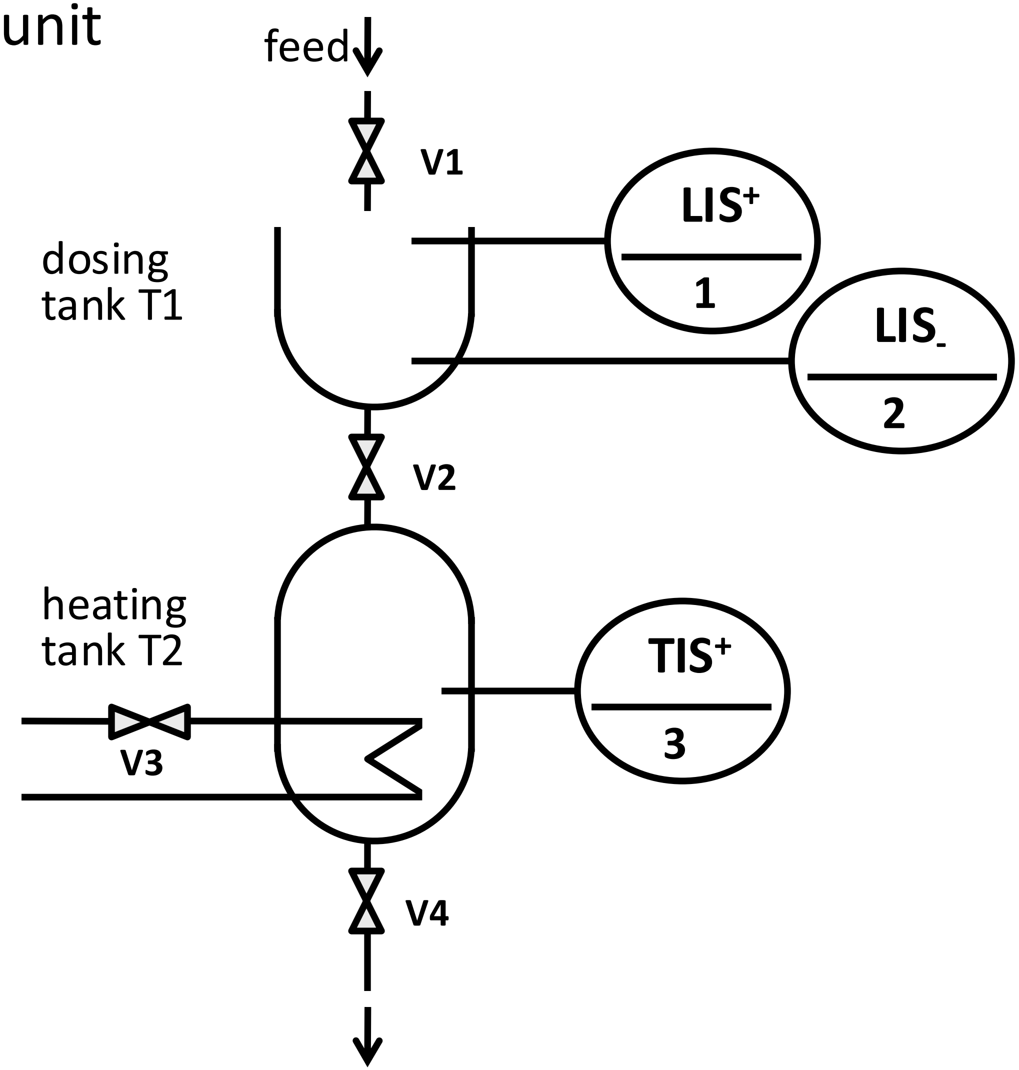
```
output: LD x0
        R  yellow
        R  red
        LD x1
        S  red
        R  yellow

        LD x2
        R  red
        S  yellow
```


Informatik 11 Embedded Software
RWTH AACHEN UNIVERSITY

Content

Specification and Implementation of Sequence Controllers

Example: A dosing and heating unit



Specification and Implementation of Sequence Controllers

Process specification for the example

sequence

Steps	Control Actions	Transition Conditions
1.) Fill T1	by opening V1	Until LIS1 activated
2.) Fill T2	by opening V2	Until LIS2 activated
3.) Heat T2	by opening V3	Until TIS3 activated
4.) Empty T2	by opening V4	Until T2 is empty
5.) go back to step 1		

Specification and Implementation of Sequence Controllers

Consequence:

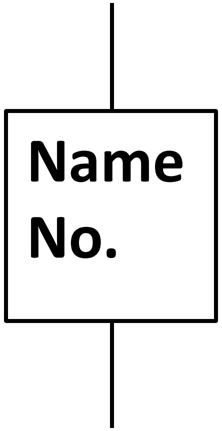
- A specification and programming language for sequence controllers must be able to represent all these elements (steps, actions, transitions, conditions, sequences, parallelism).

→ Sequential Function Chart (SFC)

Informatik 11
Embedded Software
RWTH AACHEN UNIVERSITY

Elements of SFCs

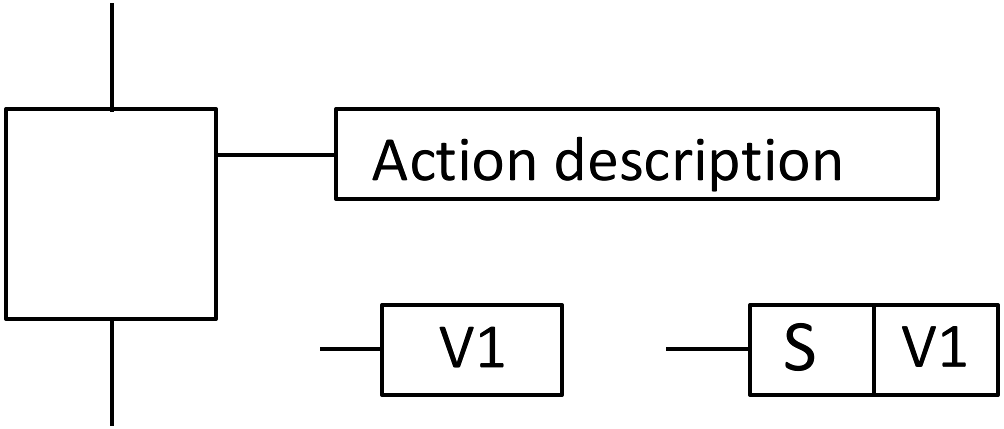
1. Steps:



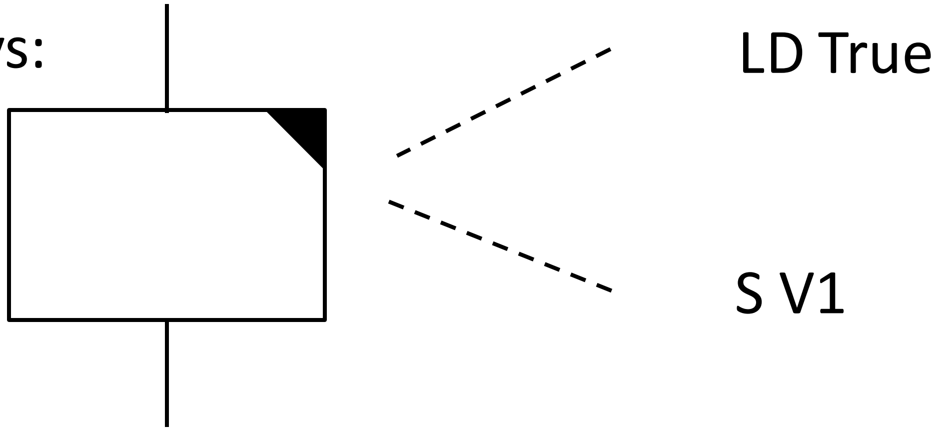
Can be active or inactive

Elements of SFCs

2. Actions:

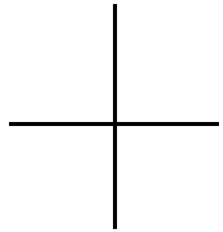


in CoDeSys:

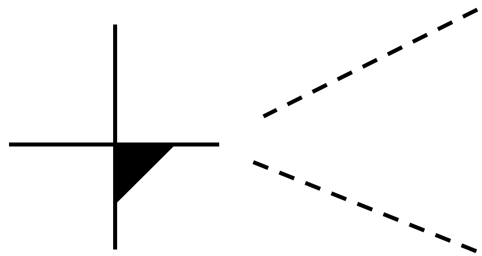


Elements of SFCs

3. Transitions:

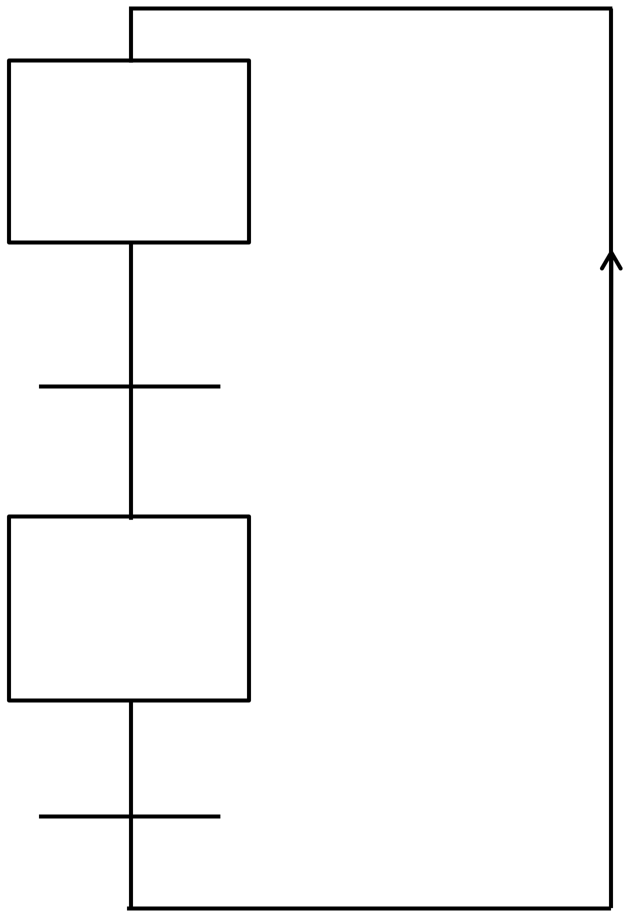


in CoDeSys



Elements of SFCs

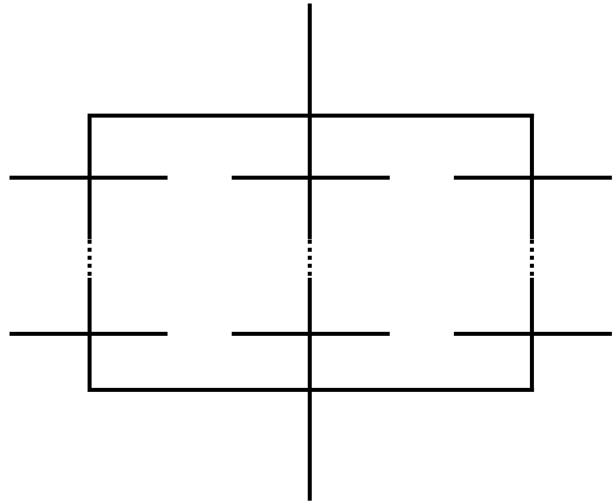
4. Connections:

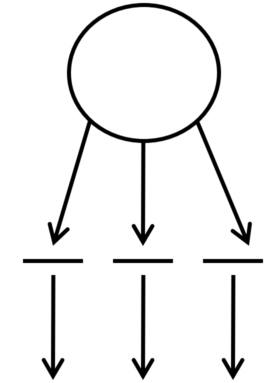


Elements of SFCs

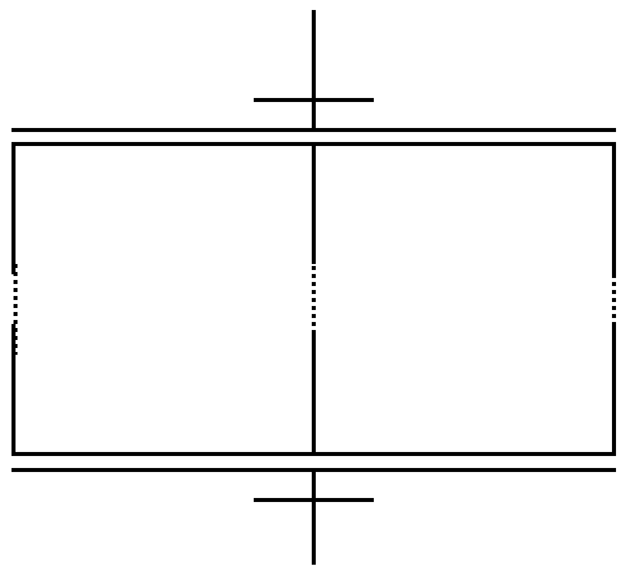
5. Branching and joining:

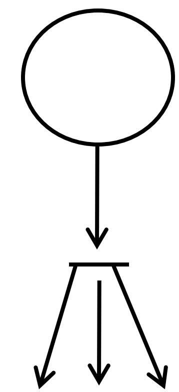
- Alternative
  - Start & End branch with Transitions





- Parallel
  - Start & End branch with Steps





SFCs

Firing rule:

A transition of a SFC fires, if

- all preceding steps are active, and
- all succeeding steps are inactive, and
- the transition condition is true.

Strong firing rule

Weak firing rule

When a transition fires, all preceding steps become inactive and all succeeding steps become active.

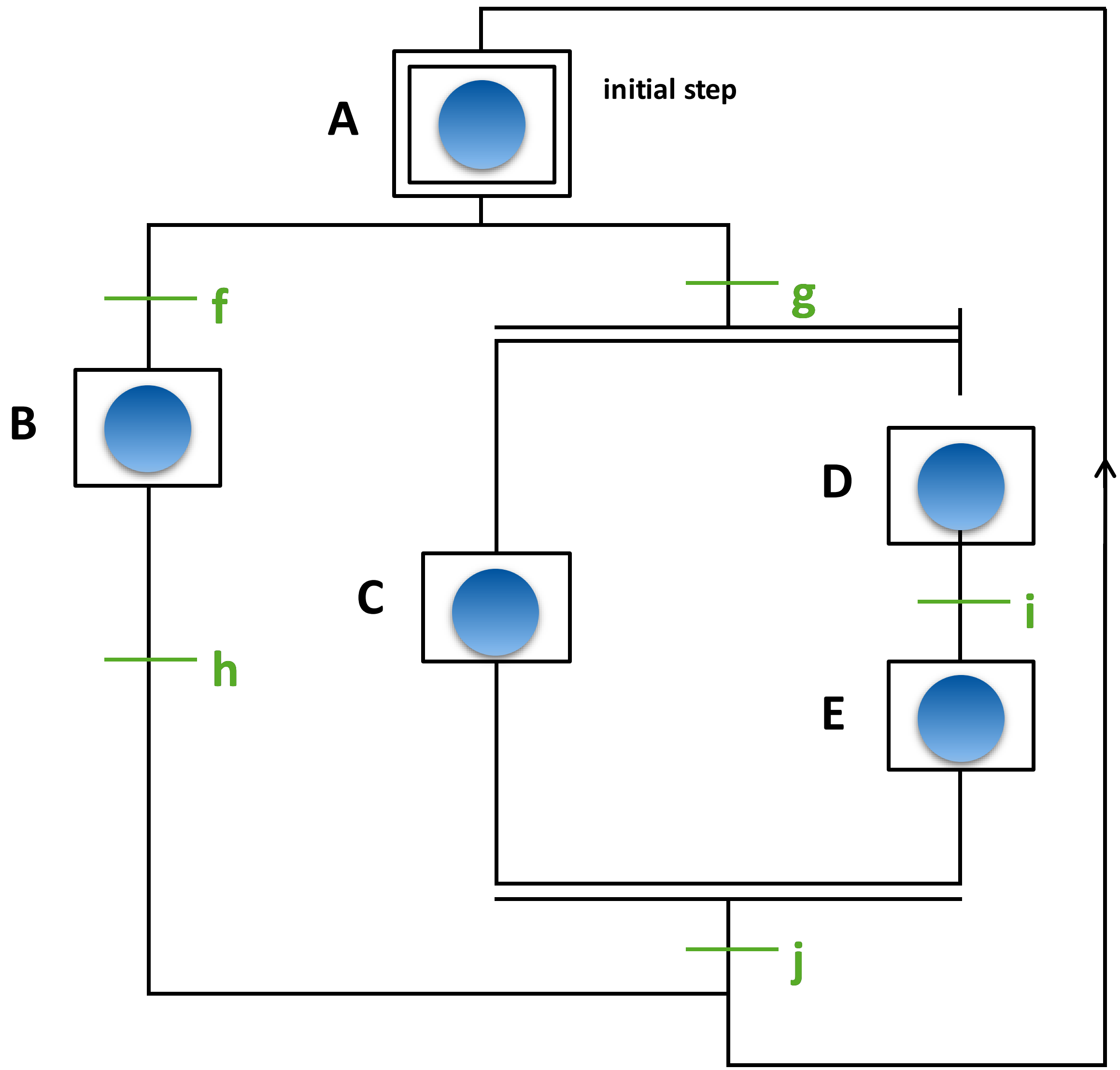
SFCs

Firing rule (ctd.)

If more than one transition can fire, priority rules apply.

Example:

SFCs

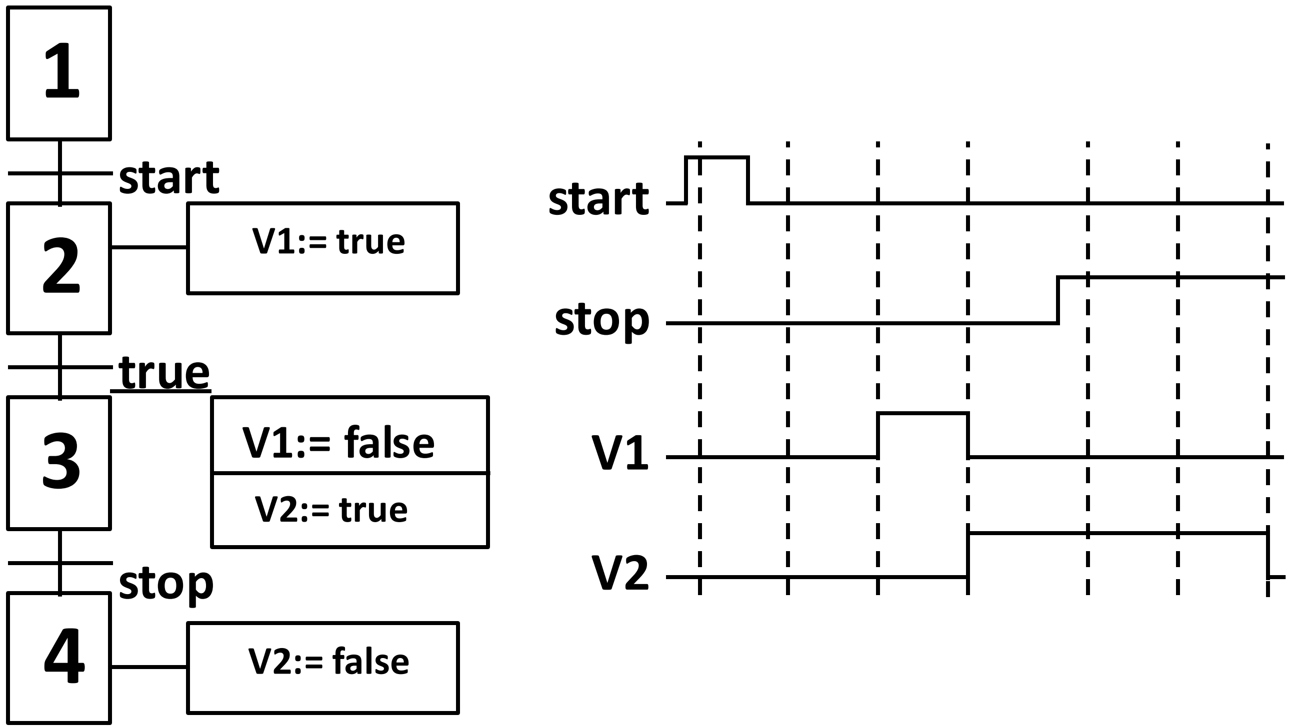


SFCs

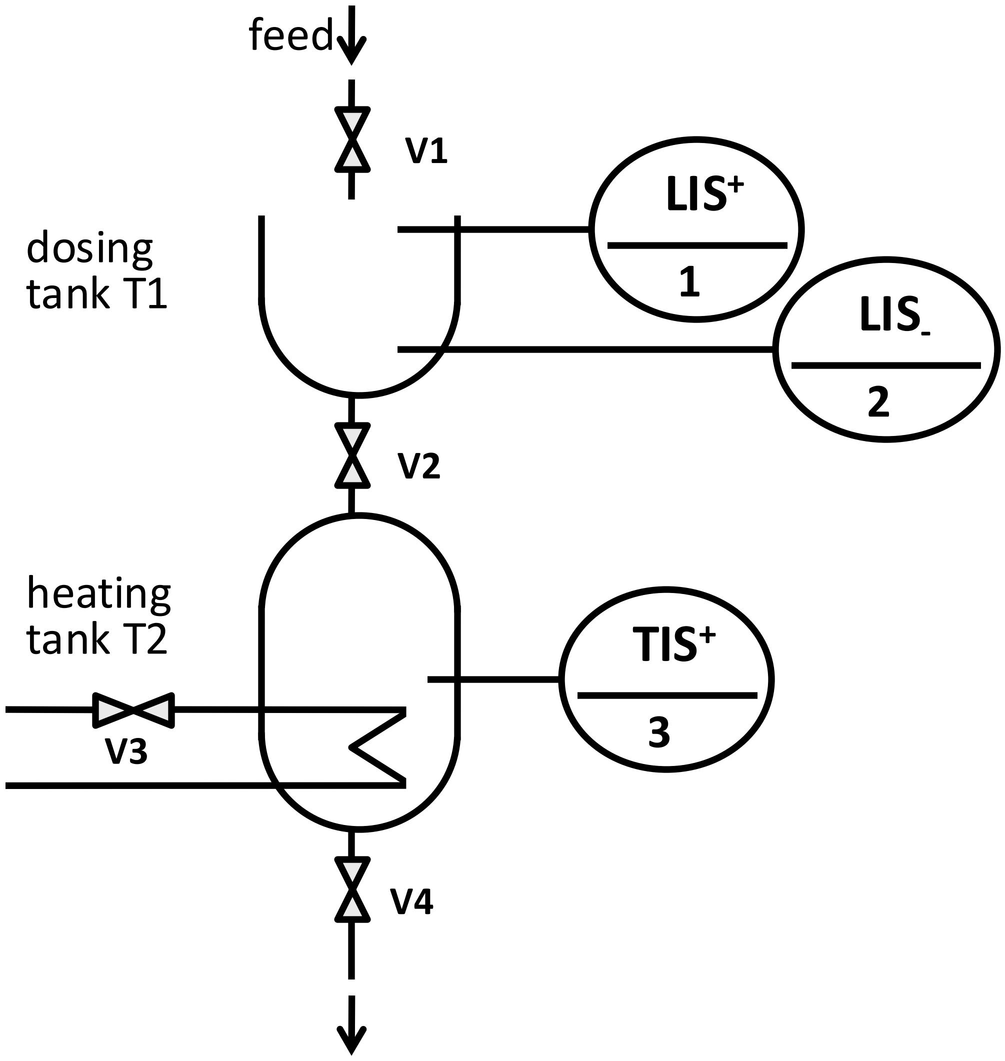
Embedding of the SFC execution into the PLC scanning cycle:

1. (Read inputs.)
2. Execute actions or underlying programs for the active steps.
3. Determine transitions which can fire.
4. Fire transitions.
5. (Write outputs.)

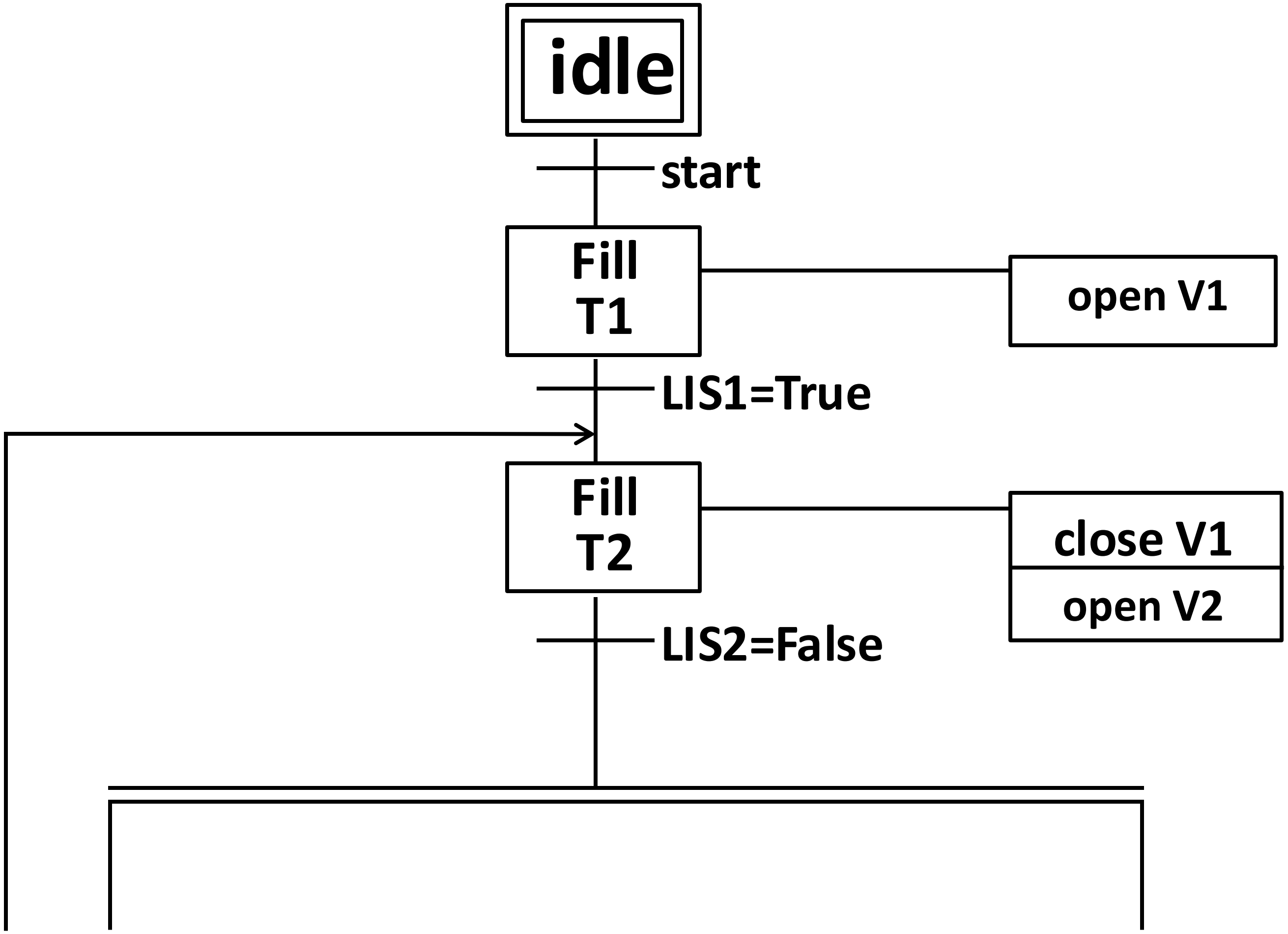
Example:



Dosing and heating tank



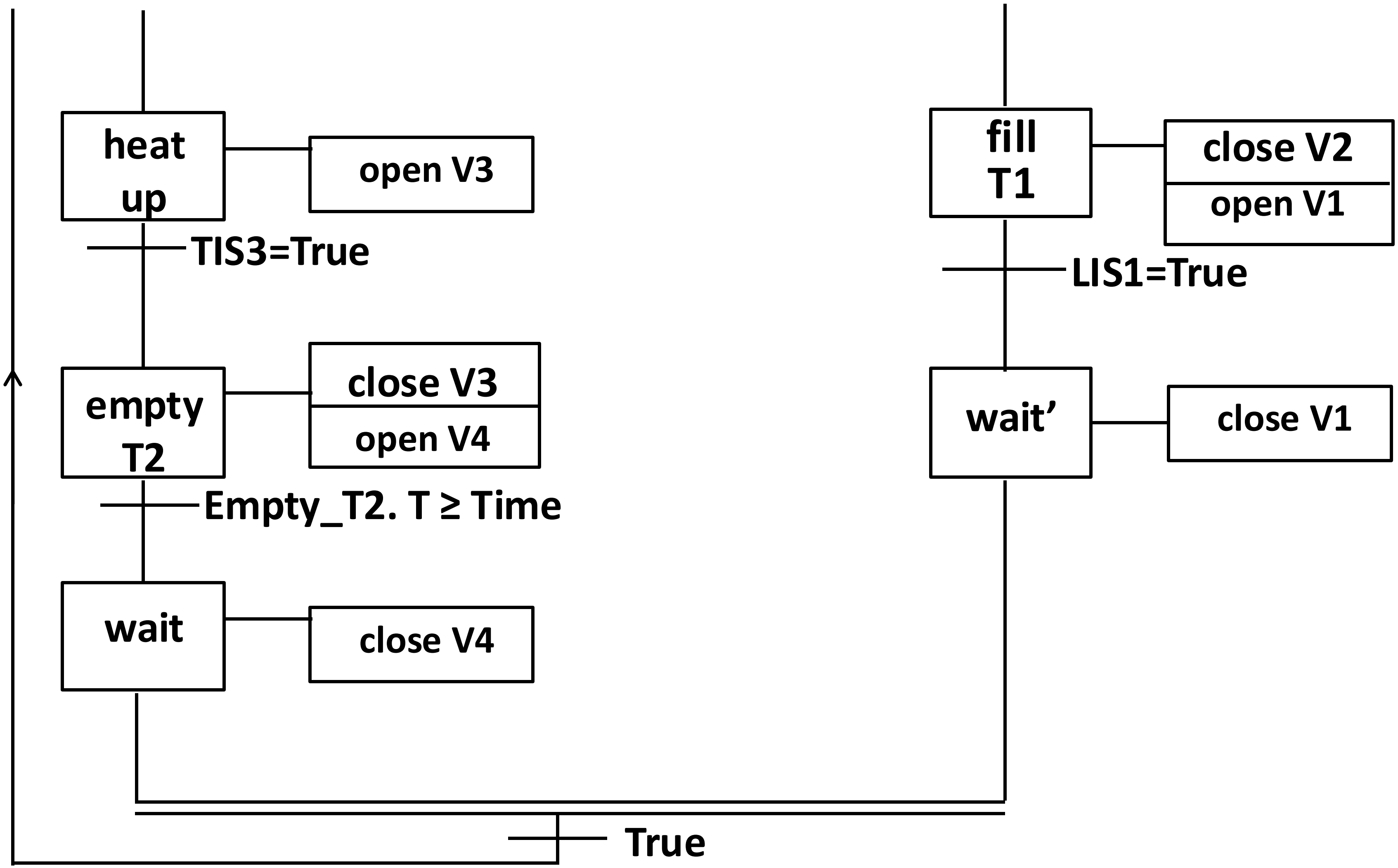
SFC for the dosing and heating tank



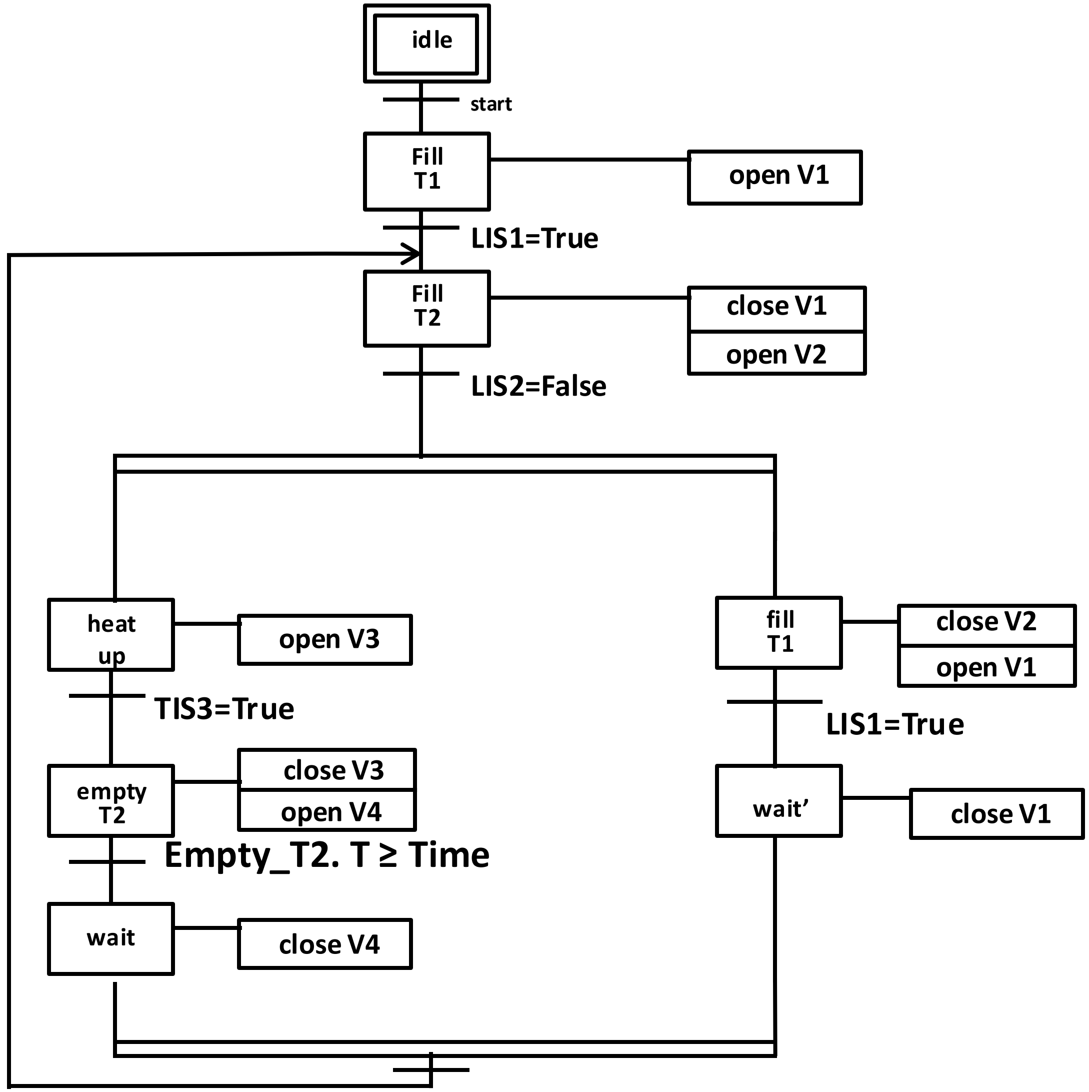
Informatik 11
Embedded Software

RWTH AACHEN UNIVERSITY

SFC for the dosing and heating tank



SFC for the dosing and heating tank (complete view)



Informatik 11 Embedded Software
RWTH AACHEN UNIVERSITY