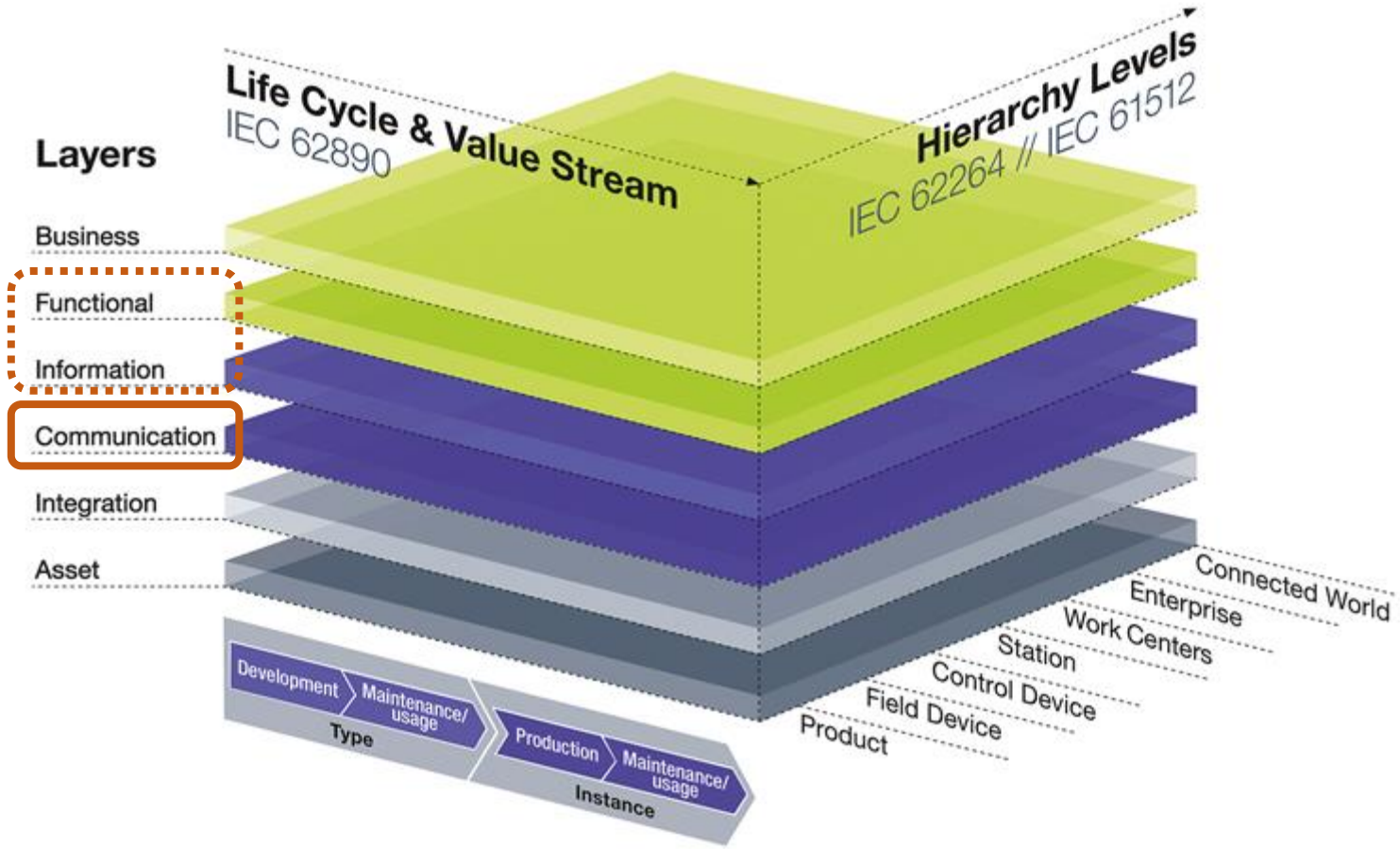
Exercises
Industrial Data Transport Technologies SS23

Introduction

Goals of this course

- Supporting the theoretical part of the lecture with practice
- Get a deeper understanding of the technologies
- Get familiar with implementation tools
- Use „real“ examples

Position of this course in RAMI4.0



Goals for today

- Getting to know each other
- What is planned for this semester?
- How will be the workflow?
- Checking the necessary requirements for the course
- Clarifying organizational issues
- Short introduction to our asset: the Real-time factory
- Getting a brief idea how to integrate the factory

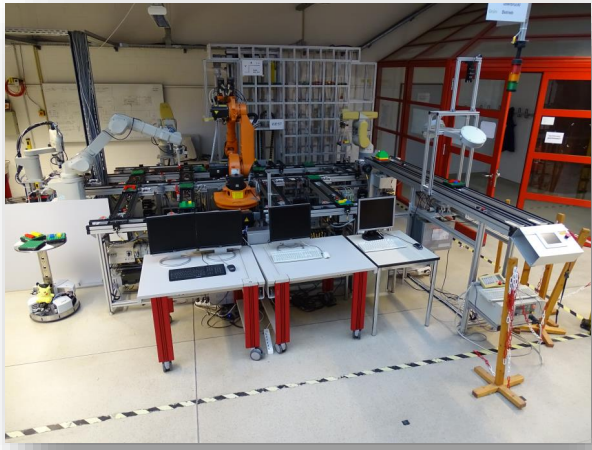
About Me

- Research assistant
- Responsible for the **practical course** since 2017
- Responsible for the research focus „**Industrial Informatics**“ of the University
- Responsible for the **„Digital Factory“** and other research facilities
- Knowledge in C, C++, Java, Python
- Experienced with SOAP, REST, OPC UA, MQTT

Jeffrey Wermann
M.Eng. Industrial Informatics

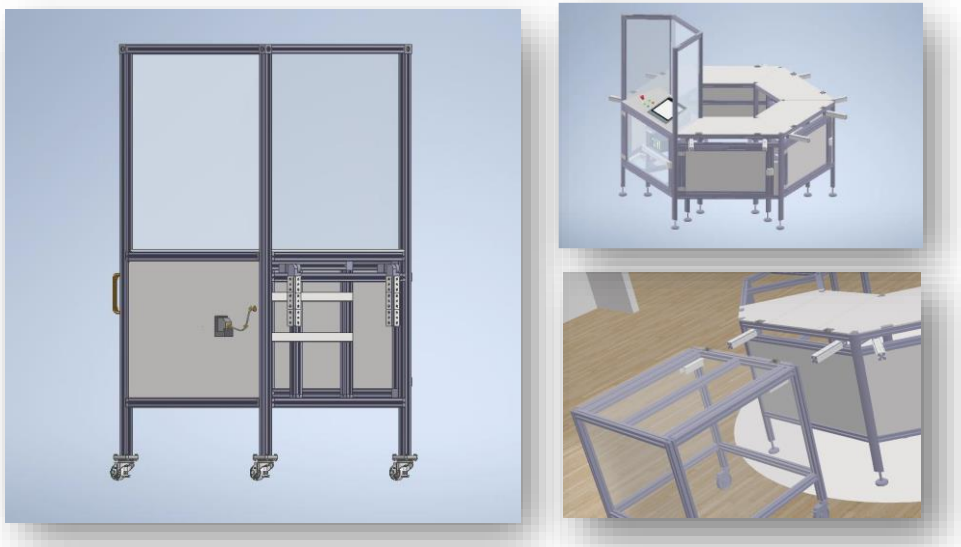
Research Assistant in „Industrial Informatics“

Phone: 04921/807-1948
E-Mail: jeffrey.wermann@hs-emden-leer.de



Digital Factory

„Old“ Digital Factory (2 years ago)



Plans for Digital Factory 2.0 (or 4.0?)

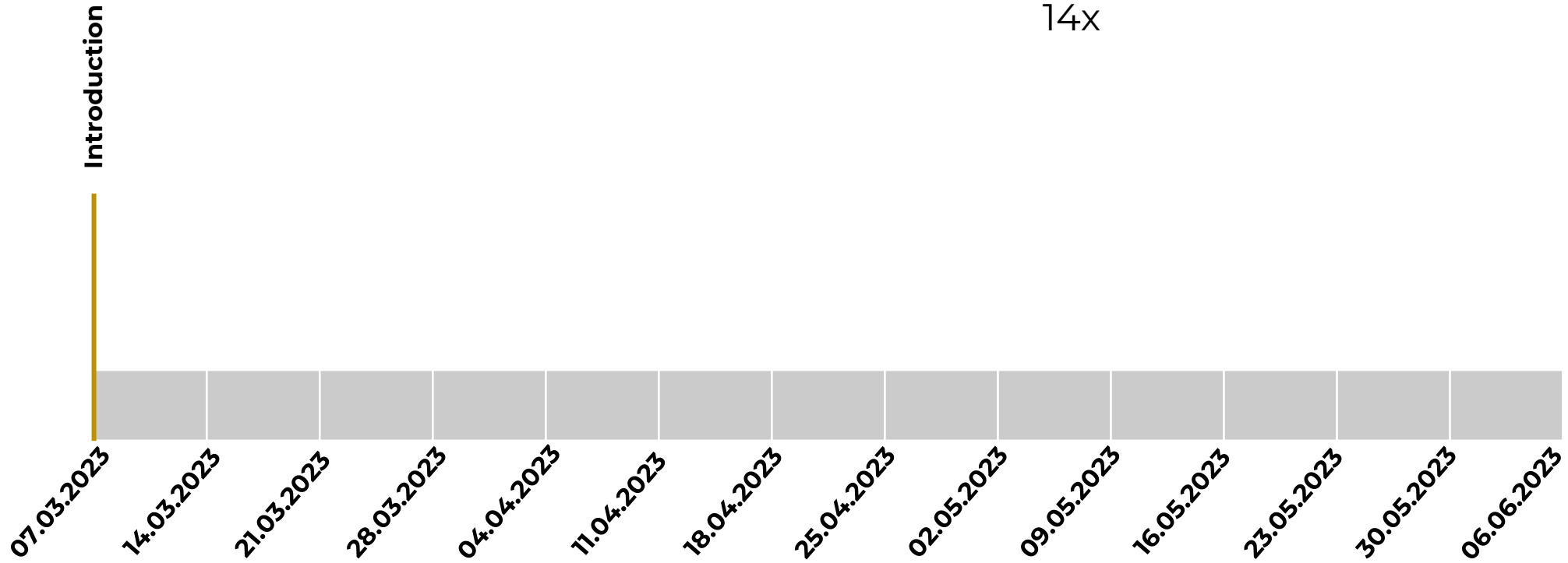
Digital Factory currently

About You

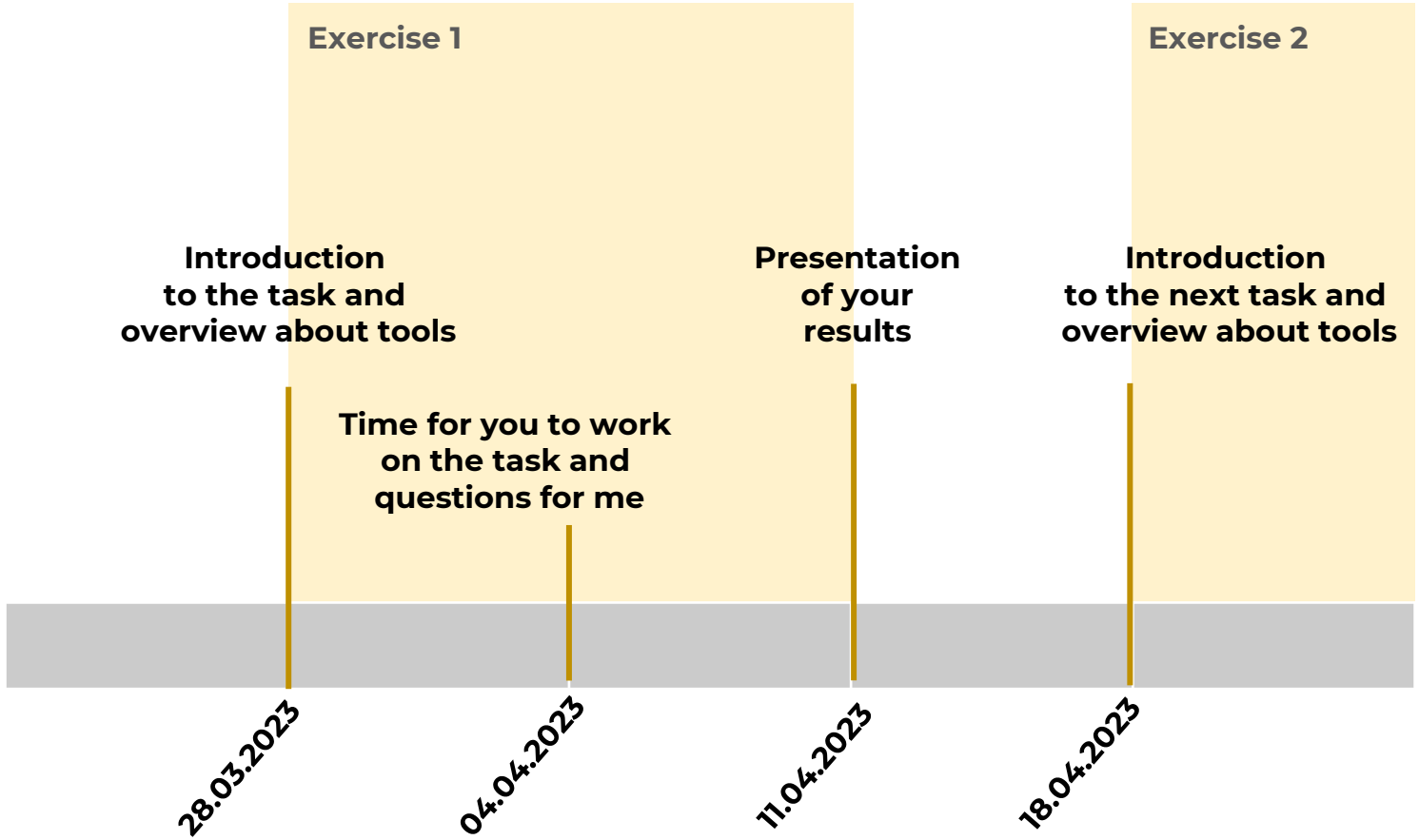
- What is your knowledge background (programming languages, protocols, etc.?)

Schedule

Each **Tuesday, 15:45 – 17:15**
14x

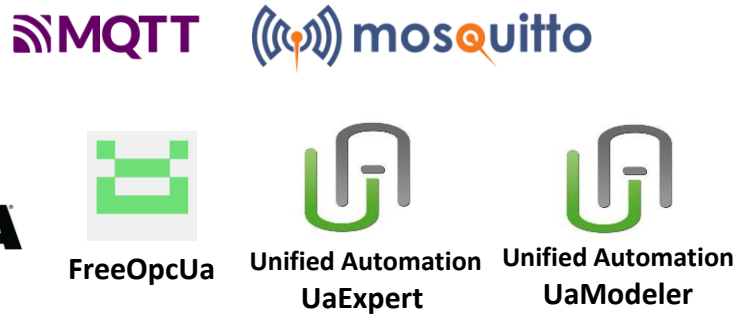


Workflow


Exercise 1
Exercise 2
Introduction to the task and overview about tools
Time for you to work on the task and questions for me
Presentation of your results
Introduction to the next task and overview about tools
28.03.2023
04.04.2023
11.04.2023
18.04.2023

Technologies covered

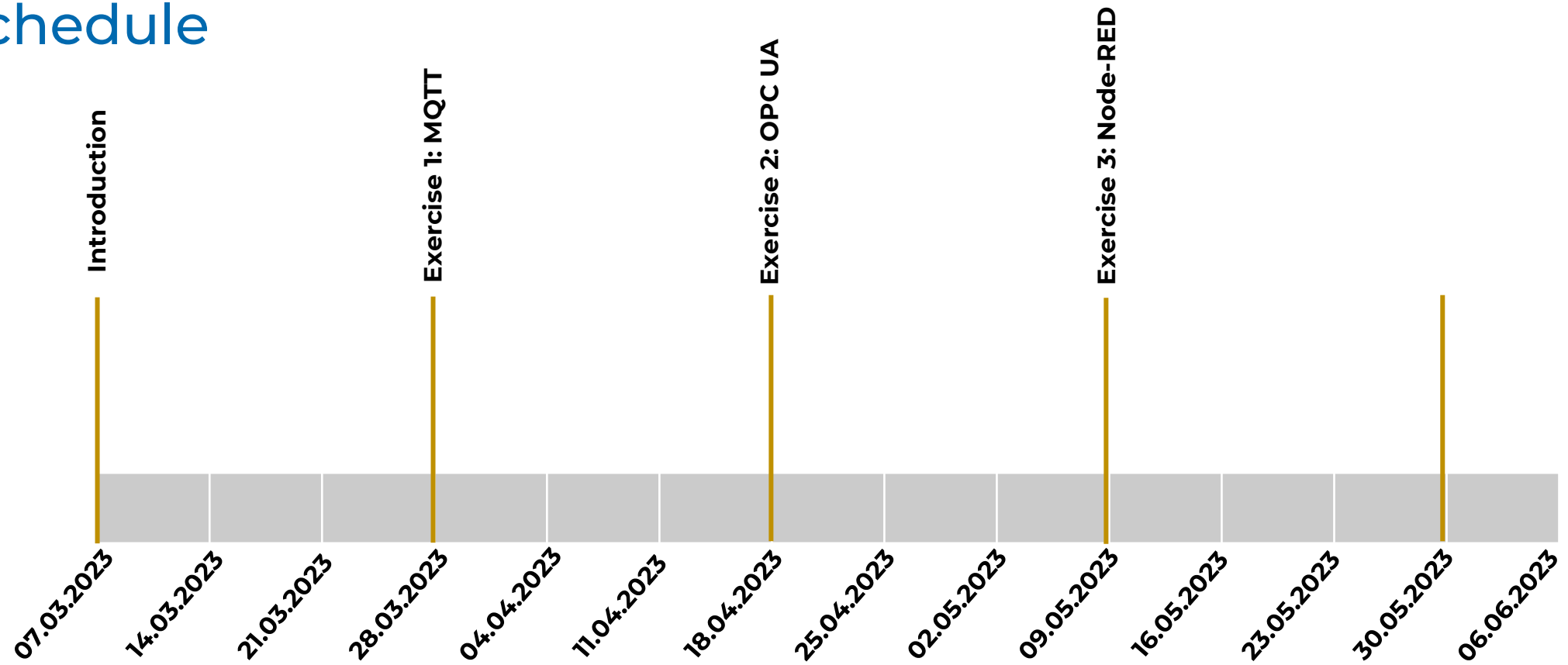
- Message Broker (MQTT)
- Industrial SoA (OPC UA)
- Tools for Information integration + processing (Node-RED)
- Tools for Information Monitoring (Grafana)
- Others (CoAP, ...?O)



Exercises (Might change)

- **Exercise 0**: Familiarizing with the asset (the Real-time factory), module selection and basic integration in Python
- **Exercise 1:** Implementation of an MQTT client which publishes information to a message broker
- **Excercise 2:** Implementation of a small OPC UA Server application using a custom modeled OPC UA node set
- **Exercise 3:** Processing of information acquired from MQTT clients and OPC UA servers in Node-RED and tracking/monitoring it in Grafana dashboards

Schedule


Introduction
Exercise 1: MQTT
Exercise 2: OPC UA
Exercise 3: Node-RED
07.03.2023
14.03.2023
21.03.2023
28.03.2023
04.04.2023
11.04.2023
18.04.2023
25.04.2023
02.05.2023
09.05.2023
16.05.2023
23.05.2023
30.05.2023
06.06.2023

Groups

- We will work in groups of 2-3
- Try to arrange yourself accordingly
- Group work means: **All group members** are actually working!
- If you have trouble finding a partner, let me know
- I will write down the groups at the beginning of next class

General Requirements

- Knowledge
  - Basic skills in Python programming
  - Basic understanding of Digitalization concepts

- Installed Libraries/Tools
  - Python v3.11.2 (or at least v3.7)
  - Python SDK of your choice (if you need)

Python Tutorials/Examples

- Python tutorials:
  - https://docs.python.org/3/tutorial/
  - https://www.w3schools.com/python/default.asp

Important Platforms we will use

- Moodle
  - Course overview, important links, exercise descriptions, announcements, ...
  - https://moodle.hs-emden-leer.de/moodle/course/view.php?id=8741

- Git?

Room Access

- Try to get a transponder dongle (also useful for other courses)
- Request access to room E13 from me (J. Wermann)

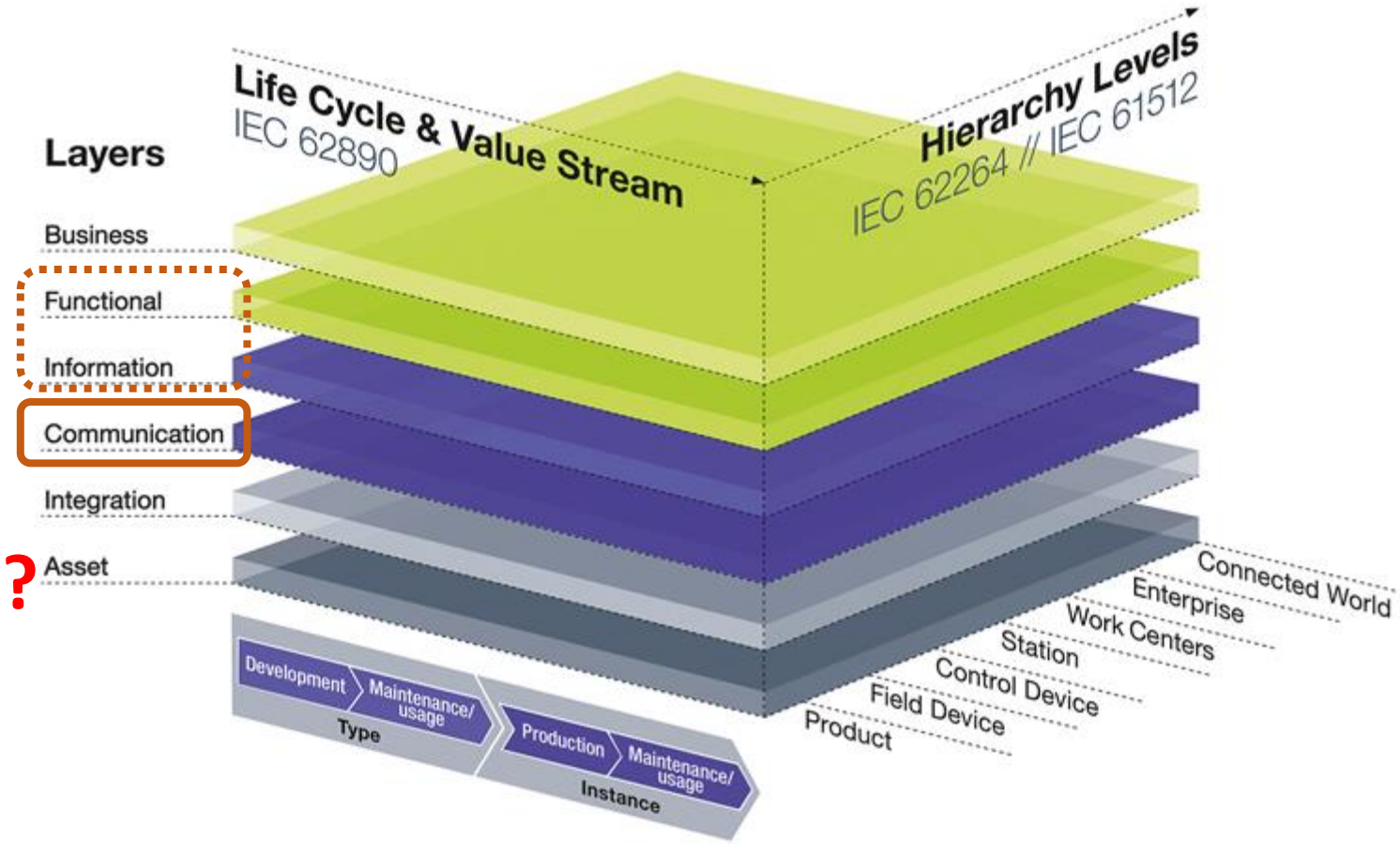
„Exercise 0“

Introducing our Asset: The Real-time Factory

Requirements for Exercise 0

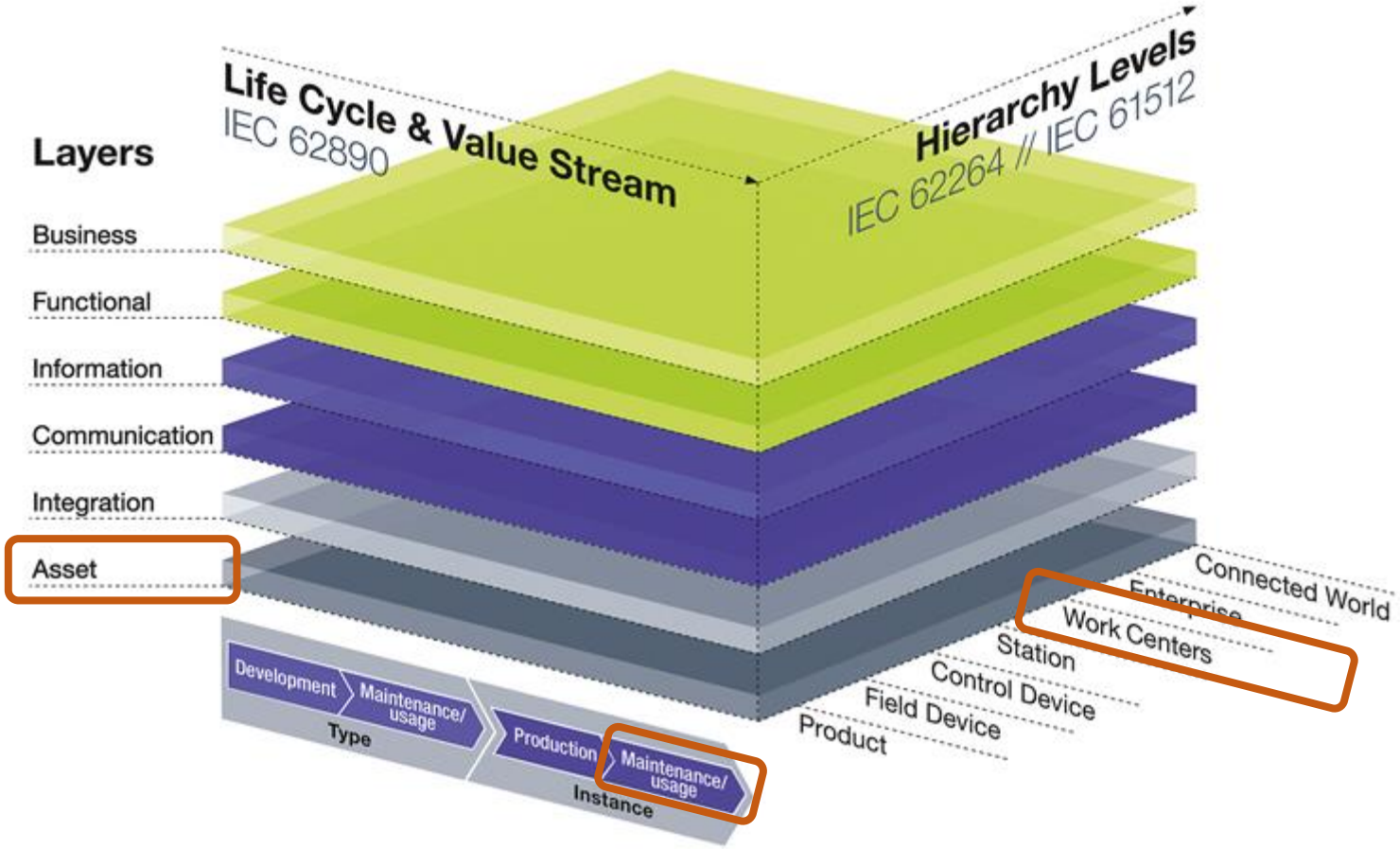
- Knowledge
  - Basic Python programming skills
- Installed Libraries/Tools
  - Python
  - pyModbusTCP
  - Python SDK (VS Code, Eclipse, etc.)

What is our asset?



The Manufacturing Model for Real-Time Data Processing (Room E13)

Position of the factory in RAMI4.0



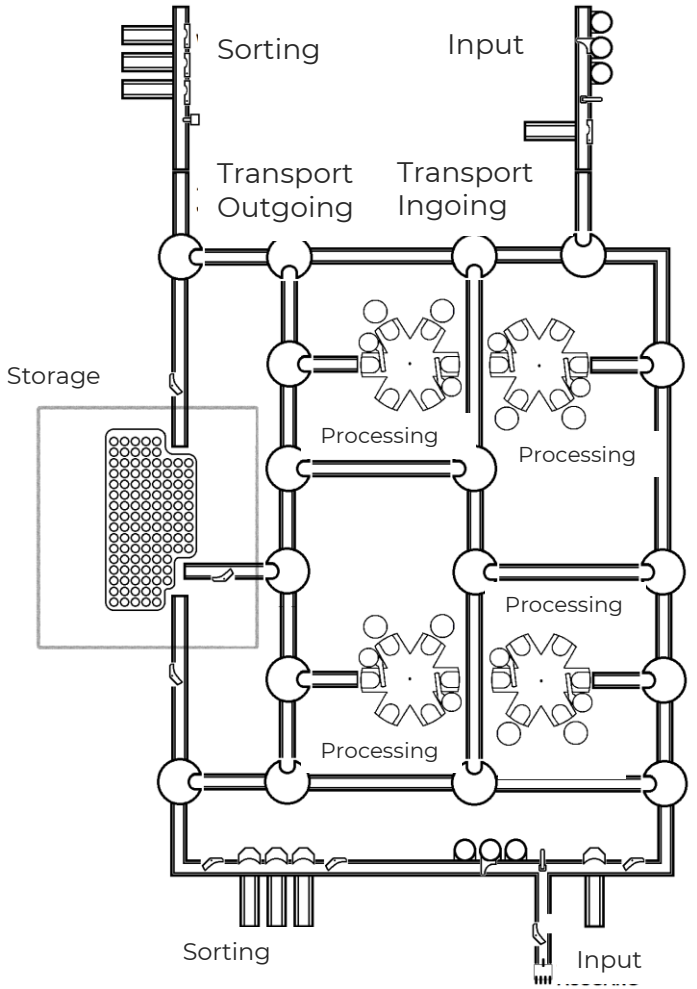
Exercise 0

Schematic Overview

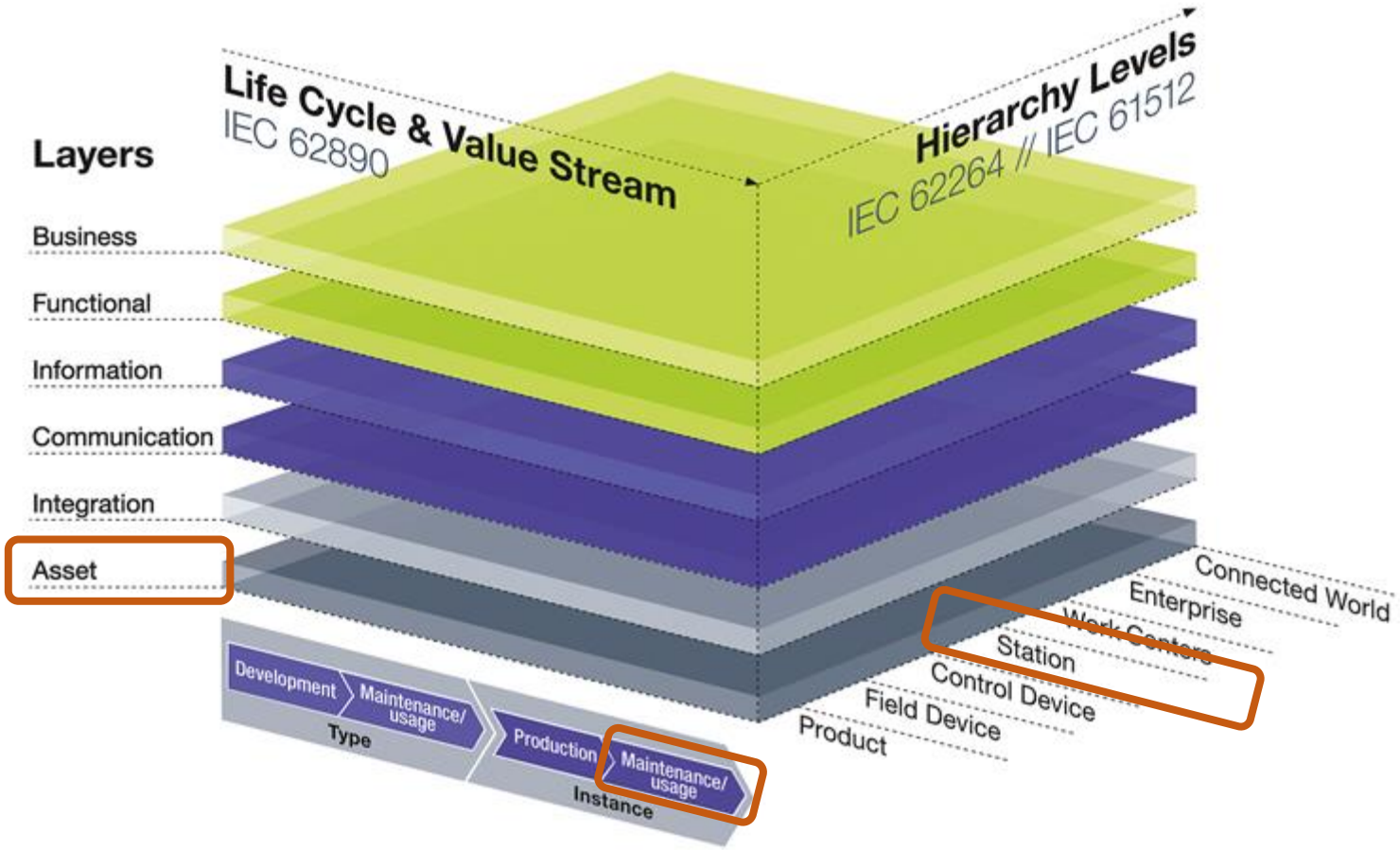
Manufacturer: Festo Didactic

Modules:

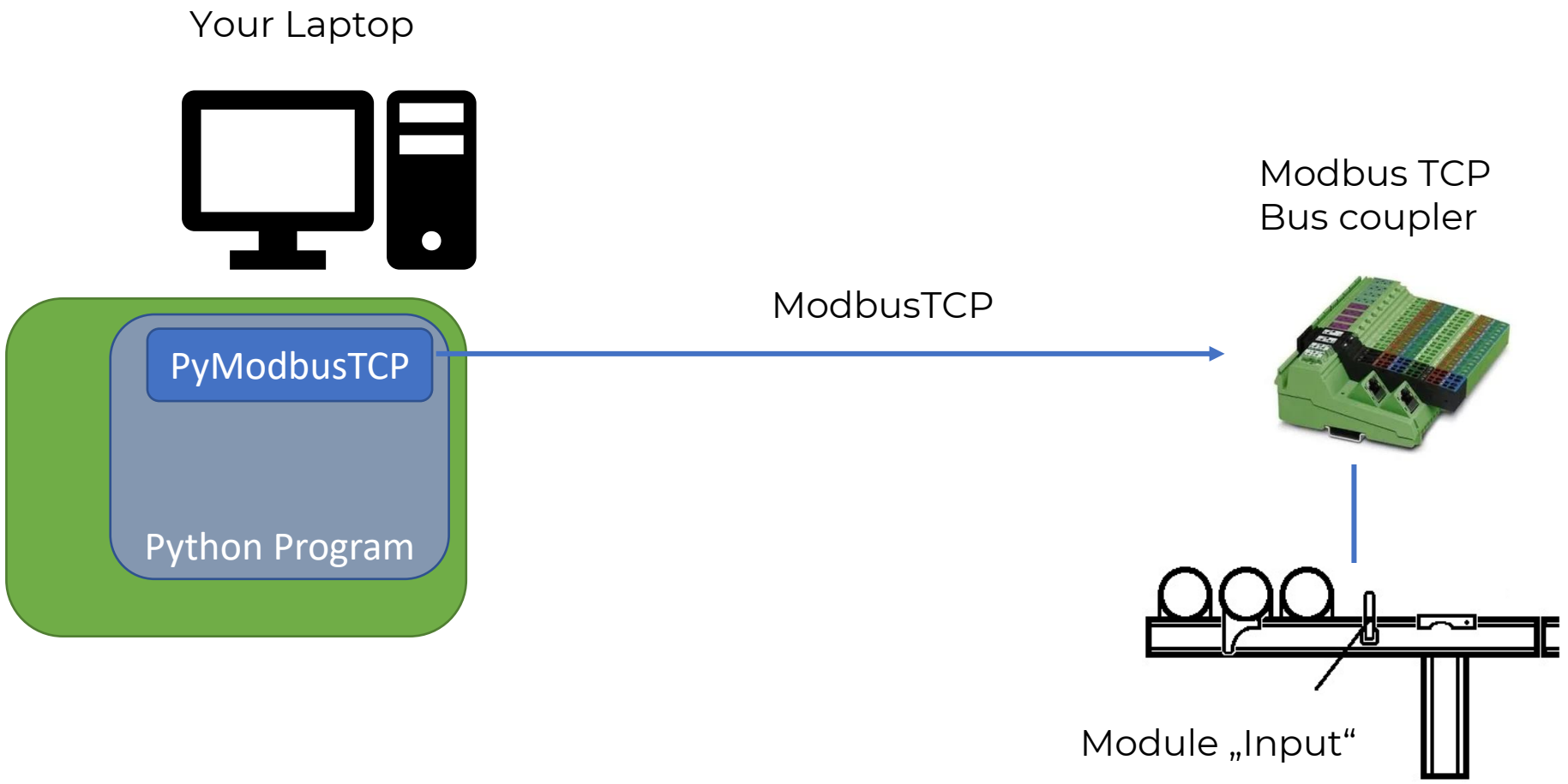
- 1x Input
- 1x Output (Sorting)
- 2x Transportation
- 4x Work Stations
- Storage



Position of the modules of the factory in RAMI4.0



Exercise Setup

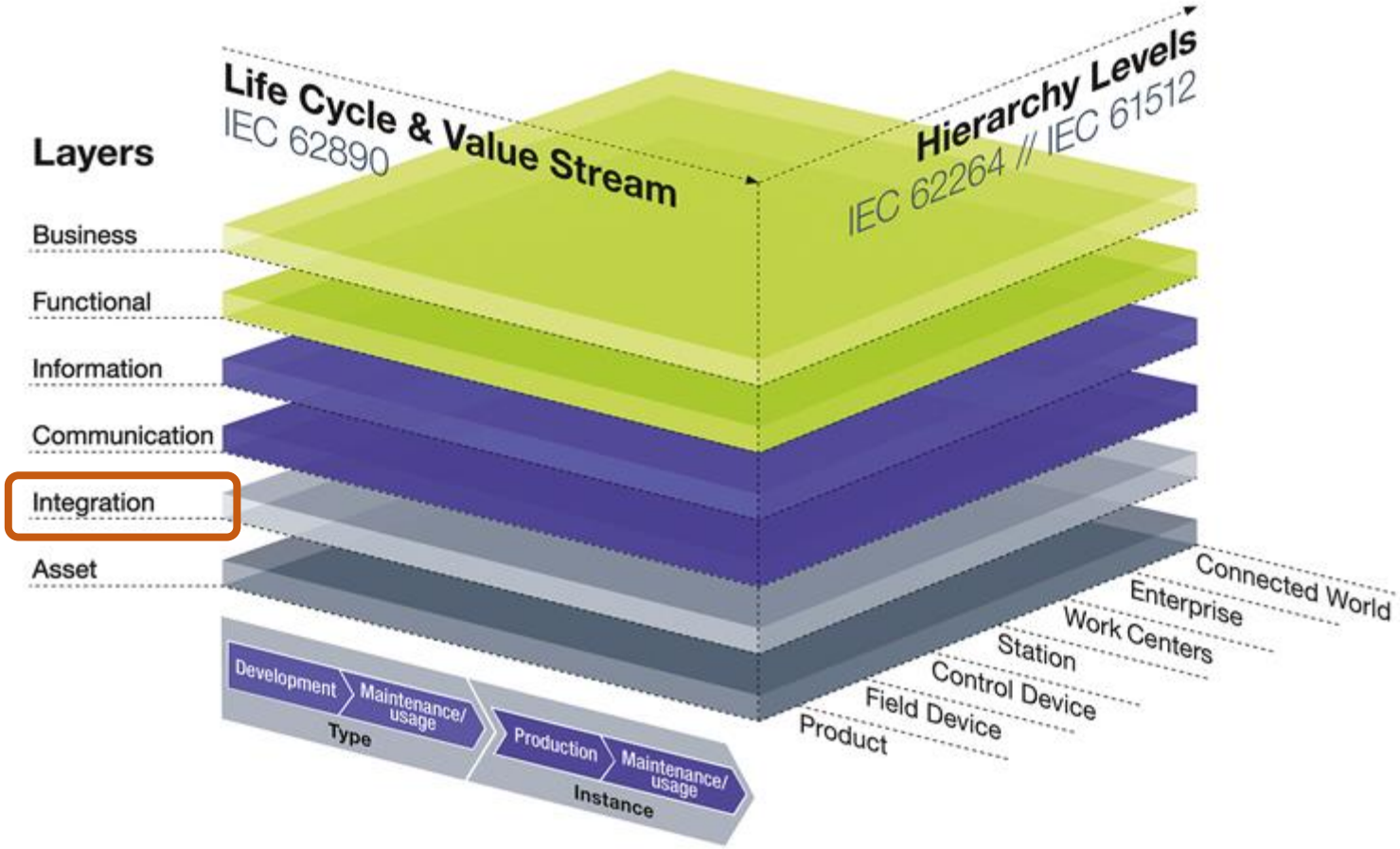


ModbusTCP

- Standard for Data Exchange
- **Modbus**: Developed for communication between PLCs and automation devices (Modicon, Schneider) -> **field bus**
- **ModbusTCP**: Ethernet-based variation

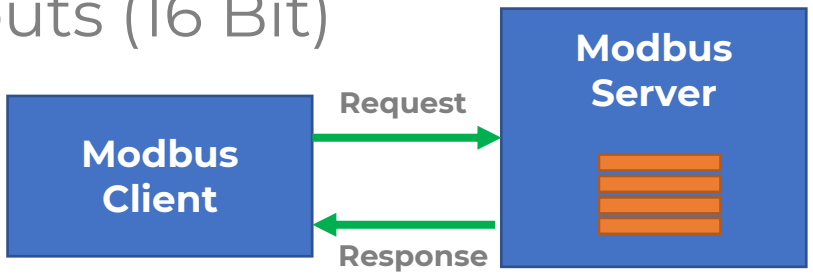
The role of ModbusTCP in our setup

In the context of Digitalization field busses are **not** used for communication between digitalized assets. But they help integrating assets into the Digital Twin.

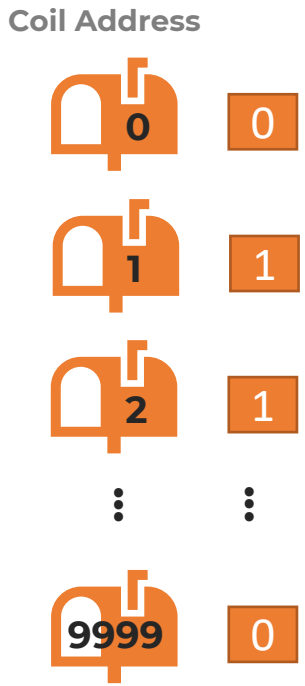


ModbusTCP – Basics

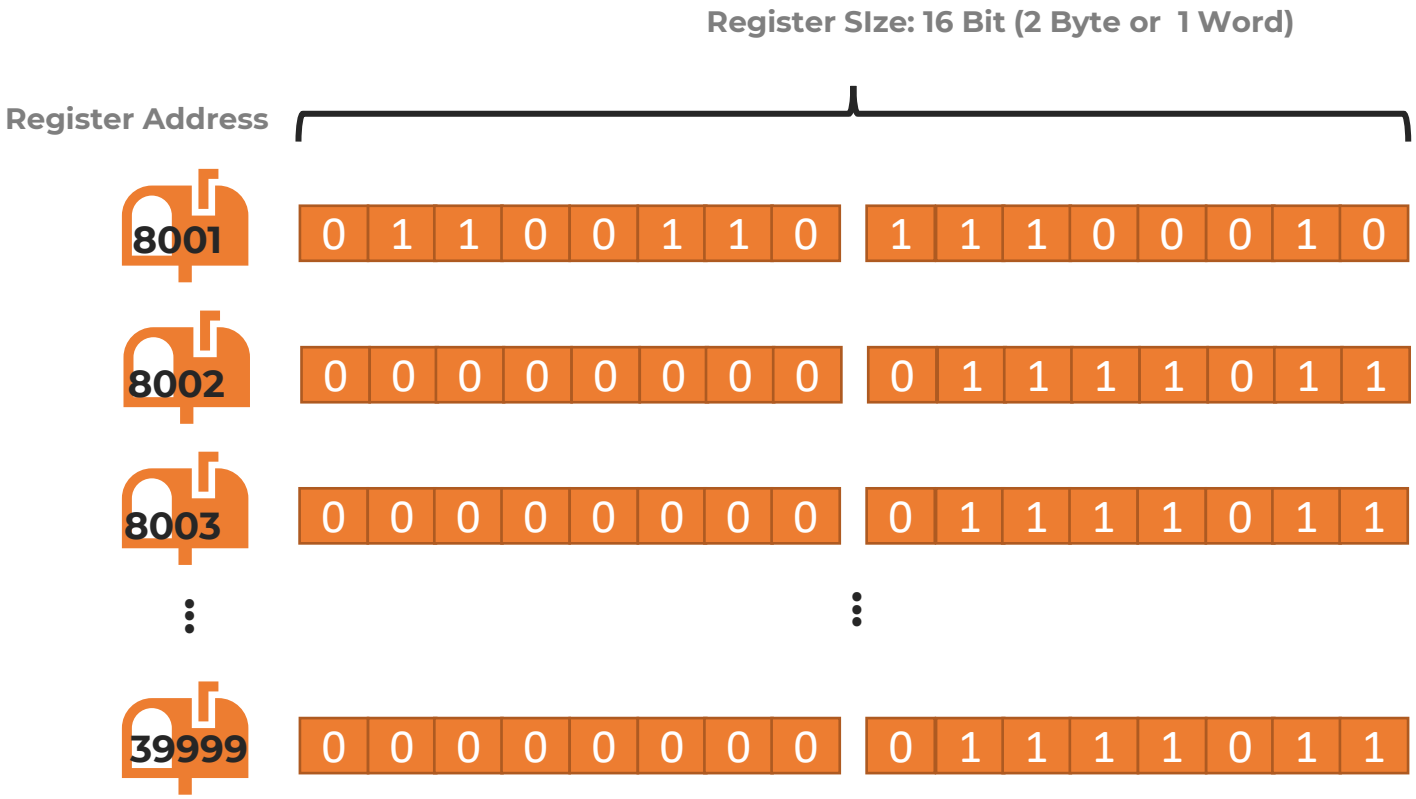
- **Server**: Provides data (passive)
- **Client**: requests data (active)
- **Request**: message sent from client to server to request data
- **Response:** server response message following a request
- **Coil:** Single In-/Output (1 Bit)
- **Register:** register with multiple in- and outputs (16 Bit)



ModbusTCP – Coils



ModbusTCP – Register



ModbusTCP – Readable Objects

Object	Access	Size
Coil	R/W	1 Bit
Discrete Input	R	1 Bit
Holding Registers	**R/W**	**16 Bits**
Input Registers	R	16 Bits

ModbusTCP – Function Codes

- **Function Codes (FC)** are transmitted with every request to the server, providing him information about what to do (e.g. read, write, access to coils, access to registers, etc.)
- Most important FCs for us:

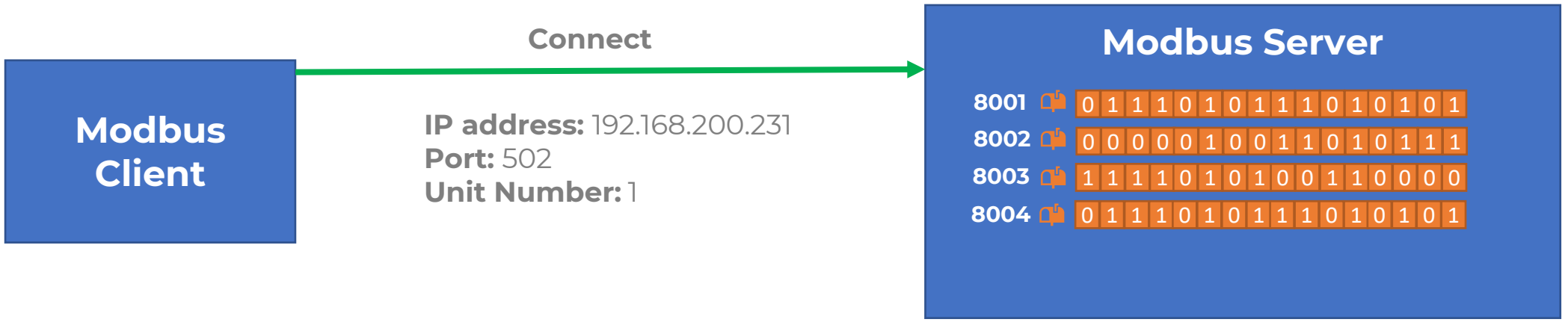
FC	Name	Description
01	Read Coils	Read *n* Coils starting at address *x*
02	Read Discrete Inputs	Read *n* discrete Inputs starting at address *x*
03	**Read Multiple Holding Registers**	**Read *n* Holding Registers starting at address *x***
04	Read Input Registers	Read *n* Input registers starting at address *x*

Read

FC	Name	Description
05	Write Single Coil	Write 0 or 1 into Coil *x*
06	Write Single Holding Register	Write Word (16 Bit) into Holding Register *x*
15	Write Multiple Coils	Write *n* Bits in Coils starting at address *x*
16	**Write Multiple Holding Register**	**Write *n* Words (16 Bit each) in Holding Register starting at address *x***

Write

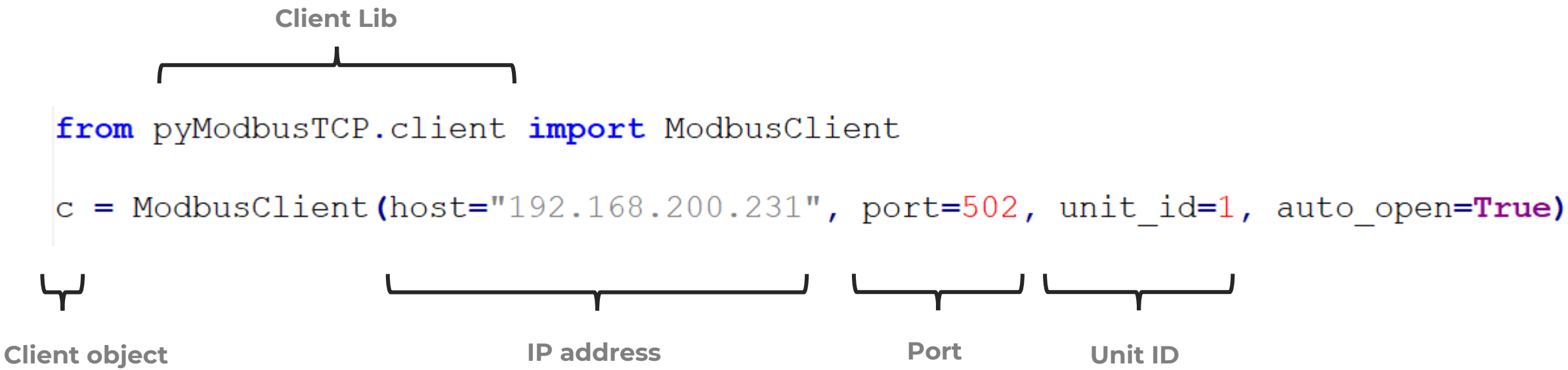
ModbusTCP – Step 1: Connecting



pyModbusTCP

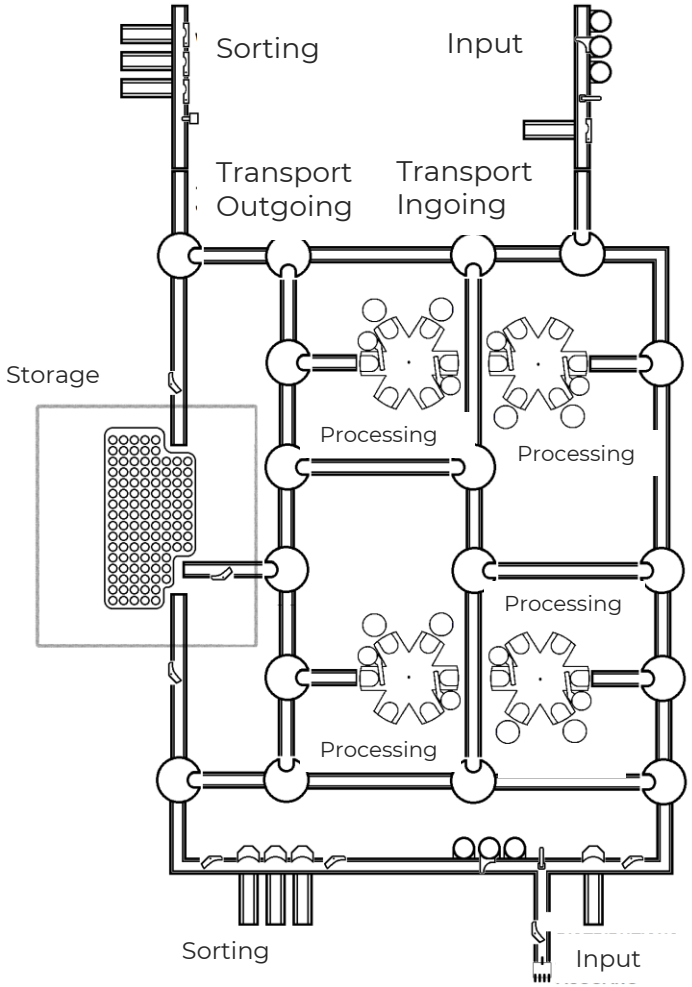
- Python library for ModbusTCP communication
- https://pymodbustcp.readthedocs.io/en/latest/quickstart/index.html

Connecting using pyModbusTCP



Exercise 0

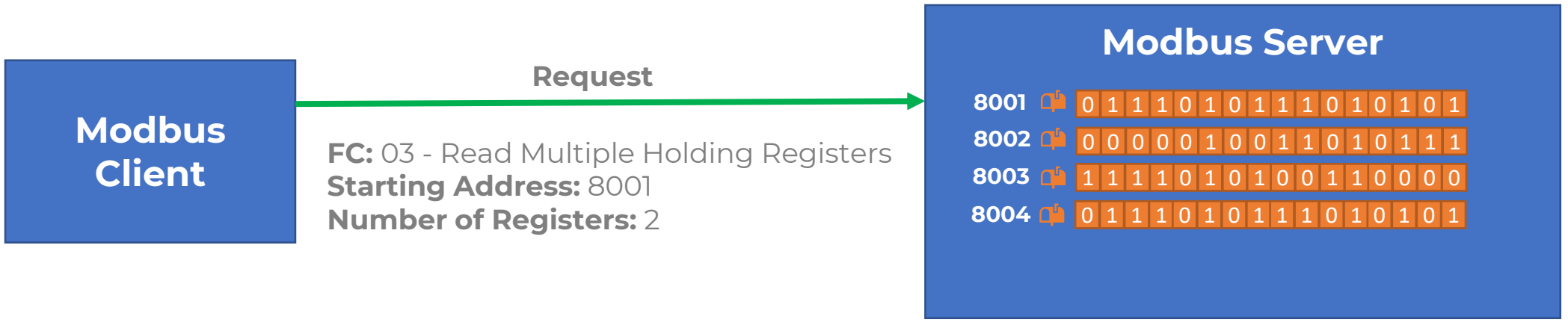
Modbus addresses



	IP
Distribution Center Input (DZE)	192.168.200.226
Distribution Center Output (DZA)	192.168.200.228
Input (WE)	192.168.200.230
Workstation 1 (B1)	192.168.200.231
Workstation 2 (B2)	192.168.200.232
Workstation 3 (B3)	192.168.200.233
Workstation 4 (B4)	192.168.200.234
Transport In (TWE)	192.168.200.235
Transport Out (TWA)	192.168.200.236
Storage (LAG)	192.168.200.237
Output/Sorting (WA)	192.168.200.238

Port: 502
Unit Id: 1

ModbusTCP – Step 2: Request (Read)



ModbusTCP – Step 3: Response (Read)

Modbus Client

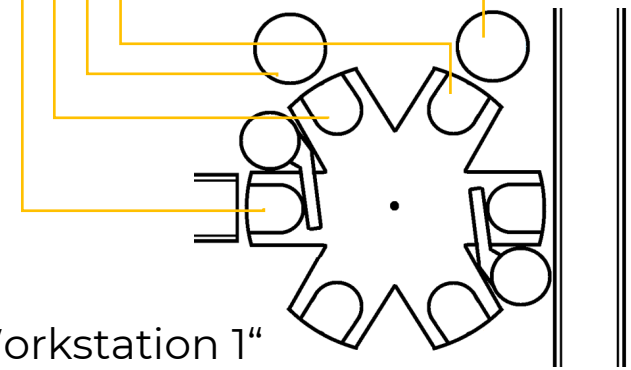
Response

Register values =
[0]: 0111010111010101
[1]: 0000010011010111

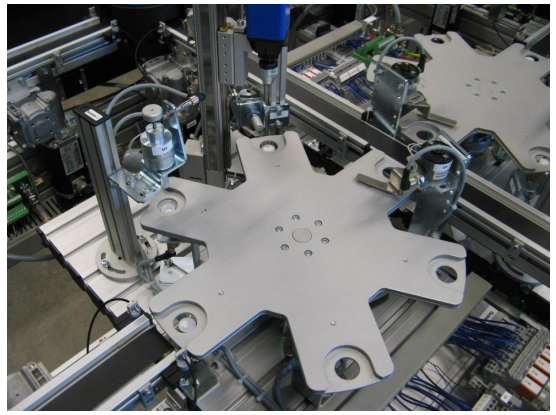
Modbus Server

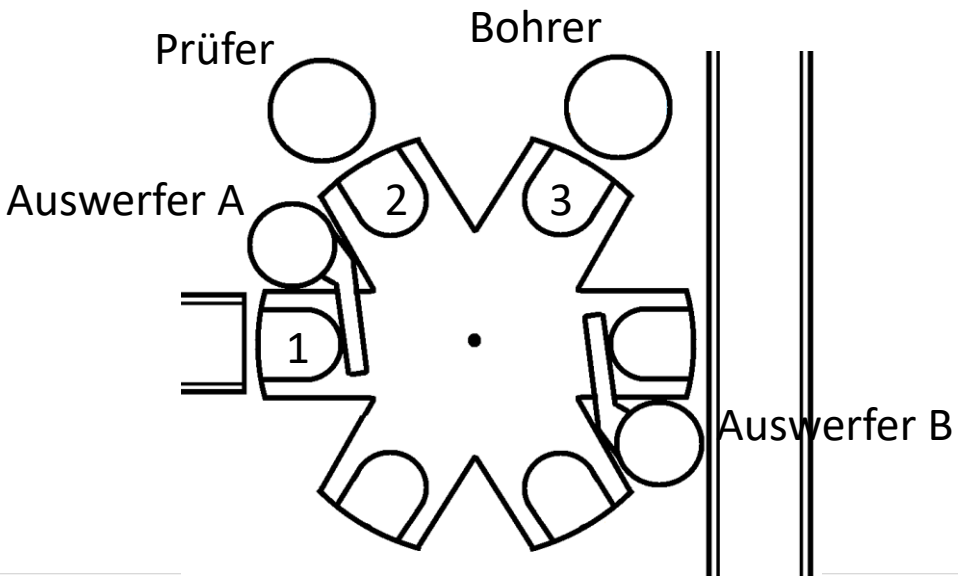
8001	0	1	1	1	0	1	0	1	1	1	0	1	0	1	0	1
8002	0	0	0	0	0	1	0	0	1	1	0	1	0	1	1	1
8003	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0
8004	0	1	1	1	0	1	0	1	1	1	0	1	0	1	0	1

Phoenix Contact ModbusTCP Bus coupler



Module/Asset „Workstation 1“

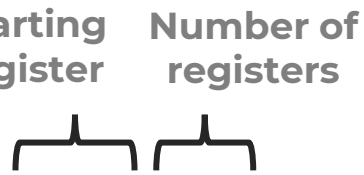




Inputs

Bit	Sensor
0	Piece in Postion 1
1	Piece in Position 3 (Drill)
2	Piece in Position 2 (Checker)
3	Drill up
4	Drill down
5	Turntable in position
6	Checker fully extended (Piece OK)

Read request using FC03 with pyModbusTCP

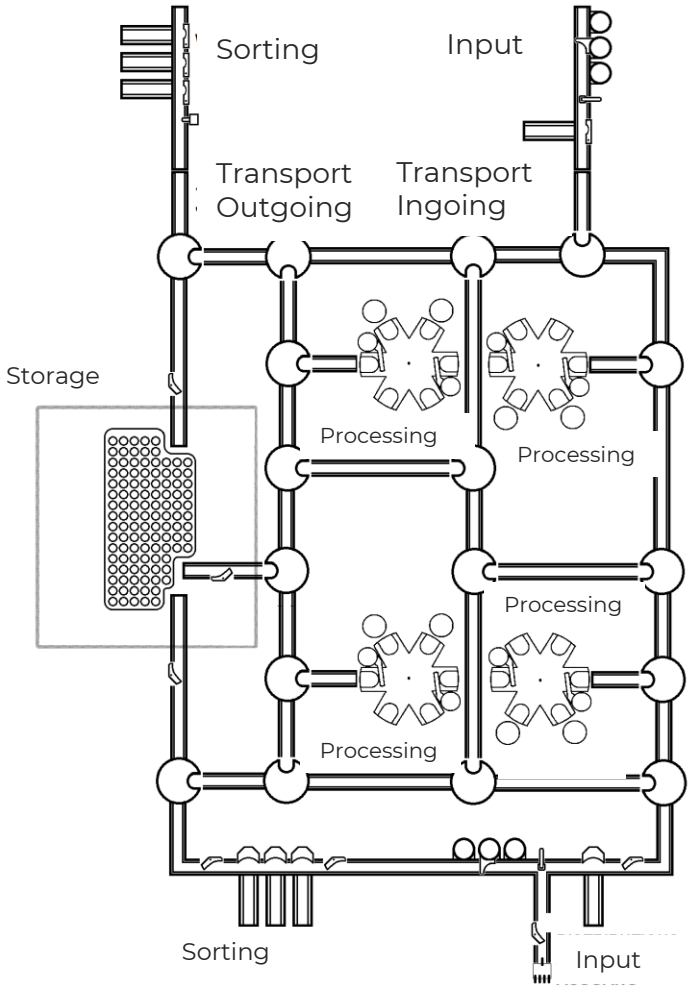



```
read_reg = c.read_holding_registers(8001, 2)
print(read_reg)
```

Return value is a list! Each element of the list contains a register value

Exercise 0

Modbus-Adressen



	Starting Address of Input Registers
Distribution Center Input (DZE)	8001
Distribution Center Output (DZA)	8001
Input (WE)	0
Workstation 1 (B1)	8001
Workstation 2 (B2)	8001
Workstation 3 (B3)	8001
Workstation 4 (B4)	8001
Transport In (TWE)	8001
Transport Out (TWA)	8001
Storage (LAG)	0
Output/Sorting (WA)	0

Helper function „test_bit“

Facilitates testing if a specific bit in a register is set

```
from pyModbusTCP.utils import test_bit
```

Variable to be checked

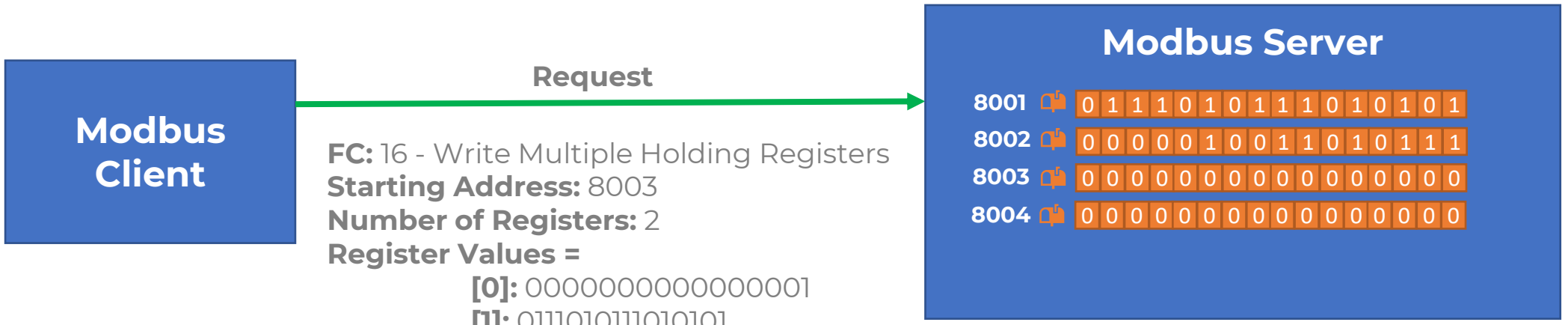
Bit to be Checked (0-15)

```
if test_bit(read_reg[0], 0) == True:
    print("Sensor Bit0 aktiv")
```

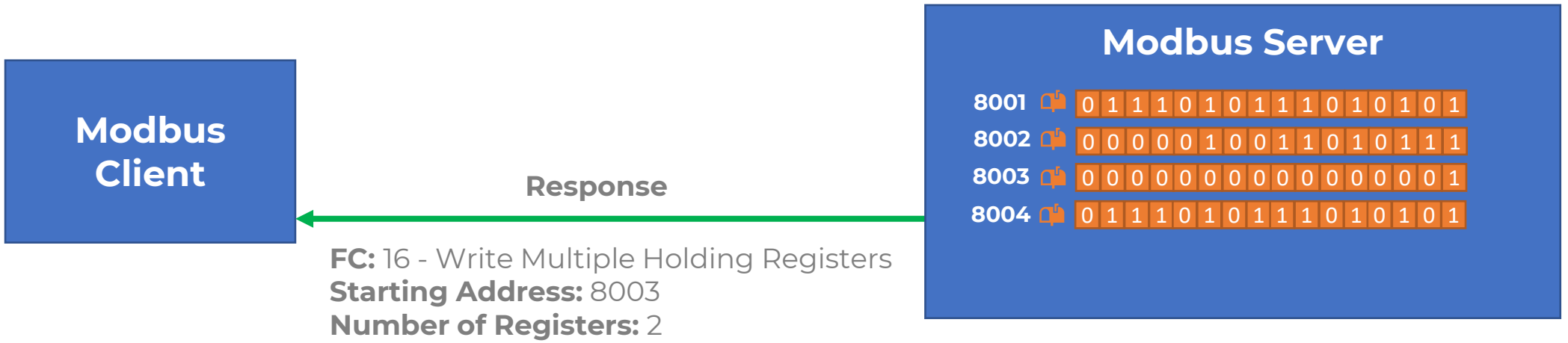
Return Value:
True: Set (1)
False: Not Set (0)

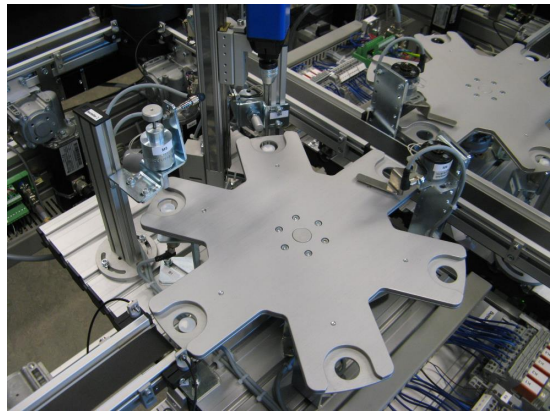
```
from pyModbusTCP.client import ModbusClient
from pyModbusTCP.utils import test_bit

from time import sleep

c = ModbusClient(host="192.168.200.231", port=502, unit_id=1, auto_open=True)

while True:
    read_reg = c.read_holding_registers(8001, 2)
    if test_bit(read_reg[0], 0) == True:
        print("Sensor Bit0 aktiv")

    sleep(1)
```

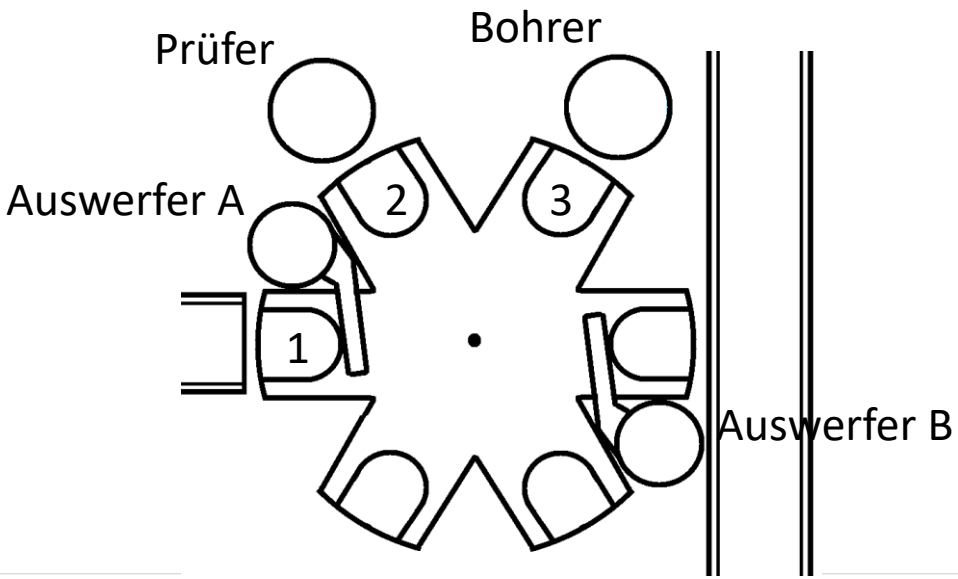
ModbusTCP – Step 2: Request (Write)



ModbusTCP – Step 3: Response (Write)







Outputs

Bit	Actuator
0	Turn drill on
1	Turn turntable*
2	Move drill up
3	Move drill down
4	Lock piece in drill position
5	Extend checker
6	Extend Ejector B
7	Extend Ejector A

*turntable only needs impulse to rotate exactly one position

Write Request with FC16 and pyModbusTCP

```
write_reg[0] = 0b0000000000000001
write_reg[1] = 0b1111010111010101
```

Starting register

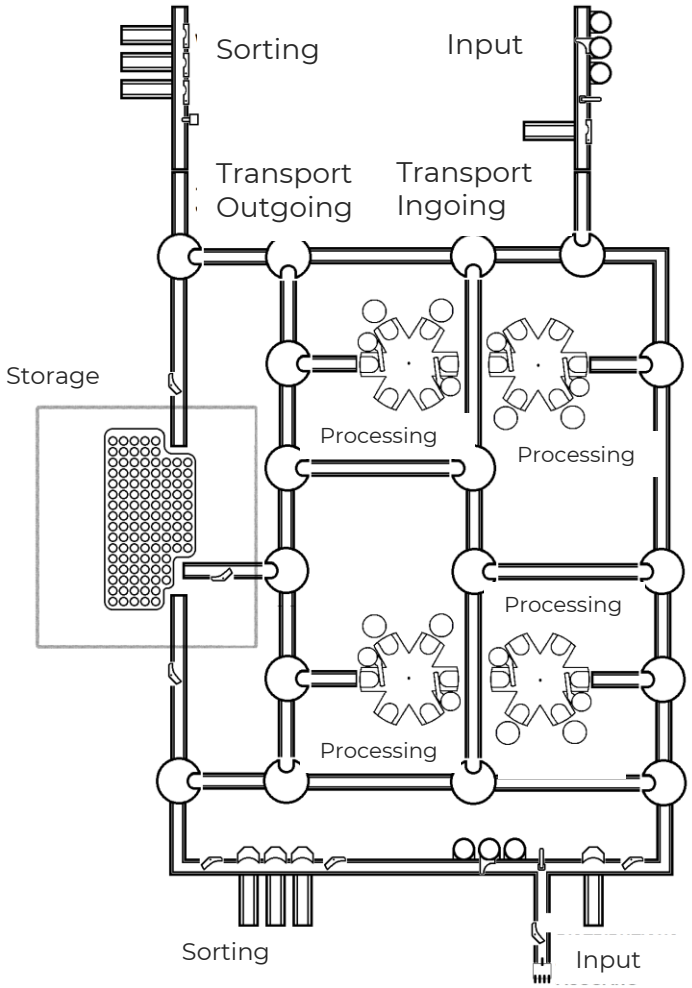
List of register values to be written

```
c.write_multiple_registers(8003, [write_reg])
```

Writing with FC16 expects a list as parameter!

Exercise 0

Modbus Addresses



	Starting Address of Output Registers
Distribution Center Input (DZE)	8011
Distribution Center Output (DZA)	8009
Input (WE)	8003
Workstation 1 (B1)	8003
Workstation 2 (B2)	8003
Workstation 3 (B3)	8003
Workstation 4 (B4)	8003
Transport In (TWE)	8018
Transport Out (TWA)	8018
Storage (LAG)	384
Output/Sorting (WA)	384

Helper function „set_bit“

Facilitates setting a specific bit at position x in a register

reset_bit( ) does the opposite

```
from pyModbusTCP.utils import set_bit
```

Variable where Bit should be set

Bit to be set (0-15?)

```
write_reg = set_bit(write_reg, 0)
```

Return Value:
Register with bit Set at desired position

Exercise 0

Modbus Address Overview

	IP	Starting Address of Input Registers	Starting Address of Output Registers
Distribution Center Input (DZE)	192.168.200.226	8001	8011
Distribution Center Output (DZA)	192.168.200.228	8001	8009
Input (WE)	192.168.200.230	0	8003
Workstation 1 (B1)	192.168.200.231	8001	8003
Workstation 2 (B2)	192.168.200.232	8001	8003
Workstation 3 (B3)	192.168.200.233	8001	8003
Workstation 4 (B4)	192.168.200.234	8001	8003
Transport In (TWE)	192.168.200.235	8001	8018
Transport Out (TWA)	192.168.200.236	8001	8018
Storage (LAG)	192.168.200.237	0	384
Output/Sorting (WA)	192.168.200.238	0	384

Port: 502
Unit Id: 1

Read: FC 03 (Read Holding Registers)

Write: FC 16 (Write Multiple Holding Registers)

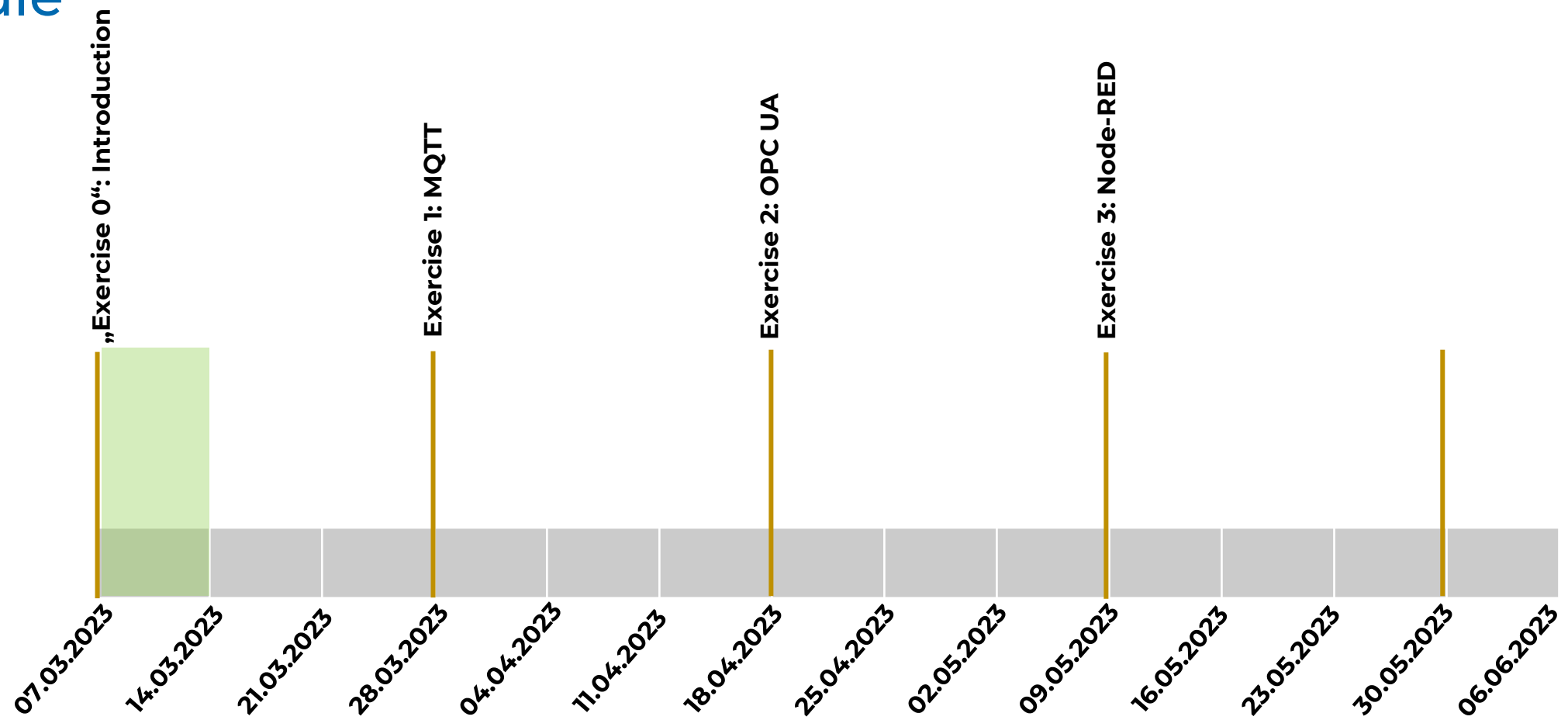
```python
from pyModbusTCP.client import ModbusClient

from pyModbusTCP.utils import set_bit
from pyModbusTCP.utils import reset_bit

from time import sleep


c = ModbusClient(host="192.168.200.231", port=502, unit_id=1, auto_open=True)

write_reg = 0

while True:
    write_reg = set_bit(write_reg, 0)
    c.write_multiple_registers(8003, [write_reg])

    sleep(2)

    write_reg = reset_bit(write_reg, 0)
    c.write_multiple_registers(8003, [write_reg])

    sleep(2)
```

Questions?

Exercise 0

Module selection and implementation of basic functionality in Python

Schedule


„Exercise 0“: Introduction
Exercise 1: MQTT
Exercise 2: OPC UA
Exercise 3: Node-RED
07.03.2023
14.03.2023
21.03.2023
28.03.2023
04.04.2023
11.04.2023
18.04.2023
25.04.2023
02.05.2023
09.05.2023
16.05.2023
23.05.2023
30.05.2023
06.06.2023

Task Description

1. Select a Module of the Manufacturing System per Group
2. Think about possible data to be extracted from the module
3. Try to run the Python library with Control Logic
4. Try to implement a simple workflow with the library
5. Adjust the code to be able to collect interesting data

Agenda for Today

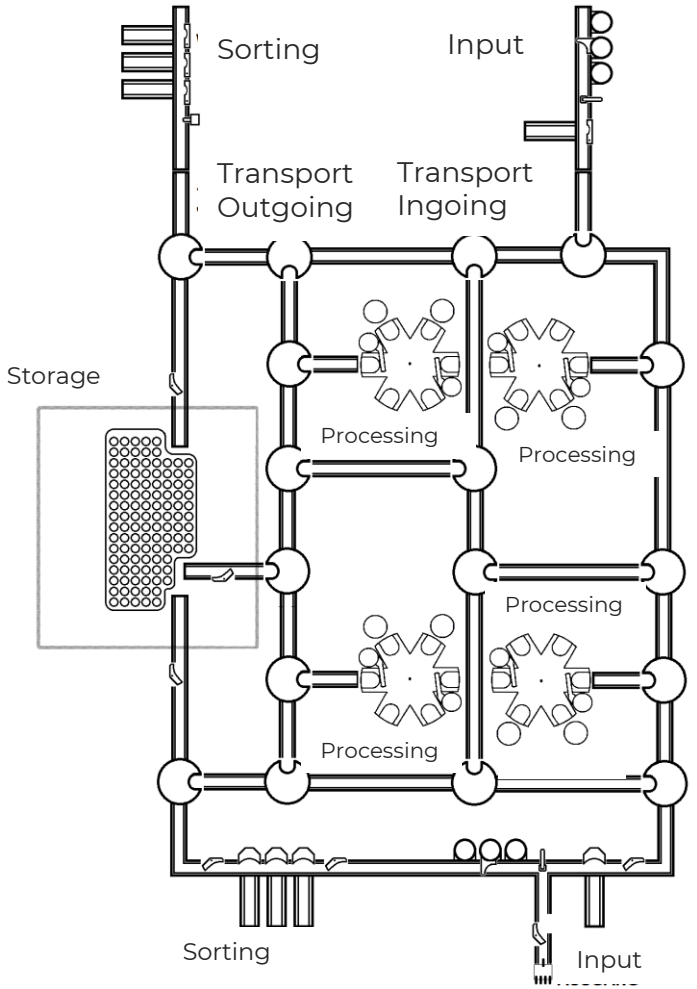
1. Group selection
2. Module selection
3. Familiarization with your module
4. Testing Modbus interface with Python
5. Accessing some I/Os using Modbus
6. If we have time: Small Introduction into the Python library for the Digital Factory

Exercise 0

Schematic Overview

Modules:

- 1x Input
- 1x Output (Sorting)
- 2x Transportation
- 4x Work Stations
- Storage



Lab Wifi

SSID: RT_Lab
PW: a*bKzZF98c

Groups

Group A – Module: Transportation Output
Shreya
Niharika

Group B – Module: Input 1
Heiko
Eduardo

Group C – Module: Transportation Input
LJ
Romin
Mobeen

Group D – Module: Storing
Niklas
Jannik

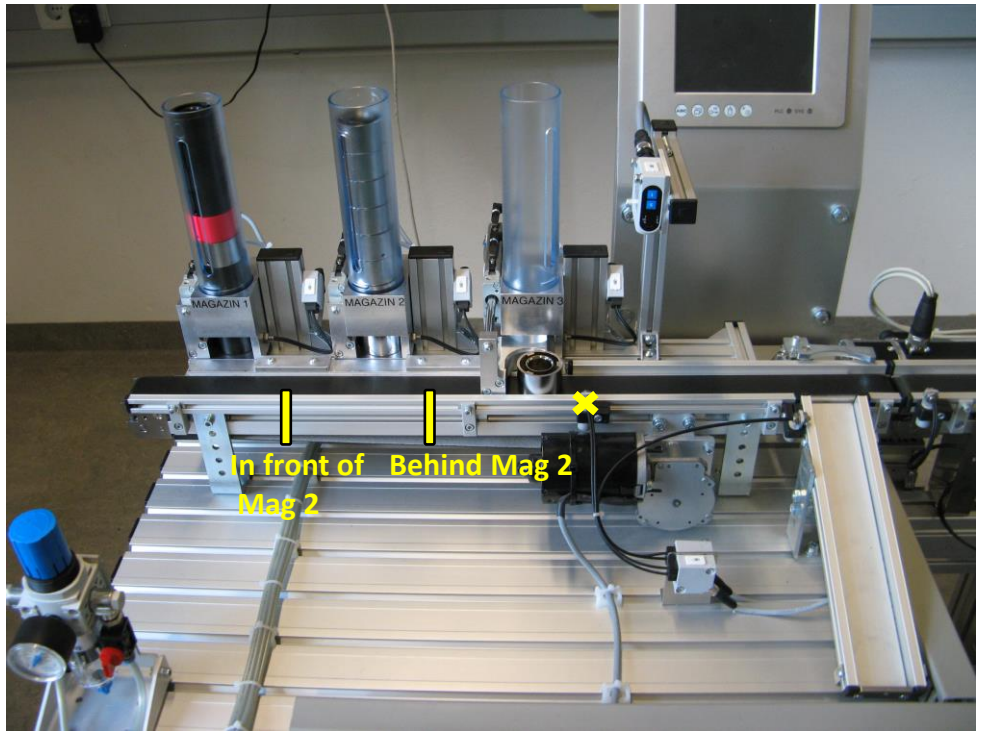
Group E – Module: Input 2
Paco
Agustin

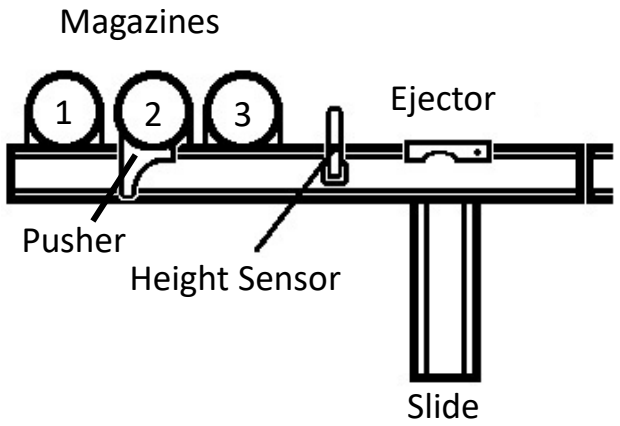
Group F – Module: Workstation 4
Shubham
Hemanth

Group G – Module: Output
Swetha
Gaston

Group H – Module: Workstation 2
Punith
Priyanka

Group I – Module:


MAGAZIN 1
MAGAZIN 2
MAGAZIN 3
In front of Mag 2
Behind Mag 2


Magazines
1
2
3
Ejector
Pusher
Height Sensor
Slide

Inputs

Bit	Sensor
0	Magazine 1 Pusher retracted
1	Magazine 1 Pusher extended
2	Magazine 1 empty
3	Magazine 2 Pusher retracted
4	Magazine 2 Pusher extended
5	Magazine 2 empty
6	Piece behind Magazine 2
7	Magazine 3 Pusher extended
8	Magazine 3 Pusher extended
9	Magazine 3 empty
10	Ejector retracted
11	Ejector extended
12	Height sensor registered piece
13	Height sensor measured: Piece is ok
14	Slide full
15	Piece in front of Magazine 2

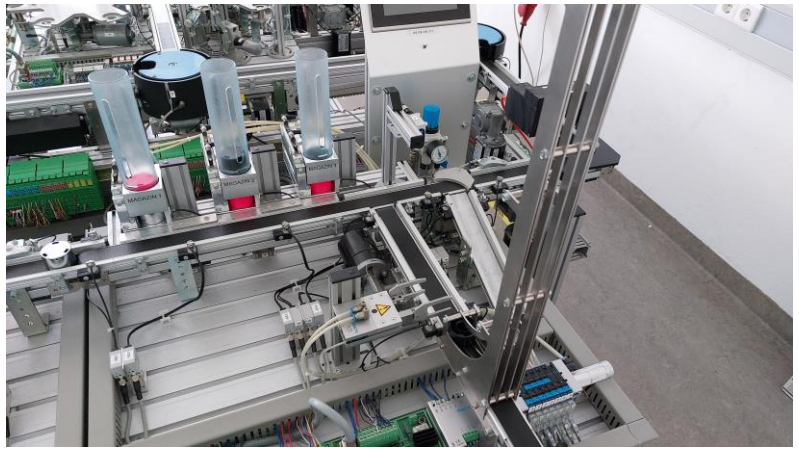
Outputs

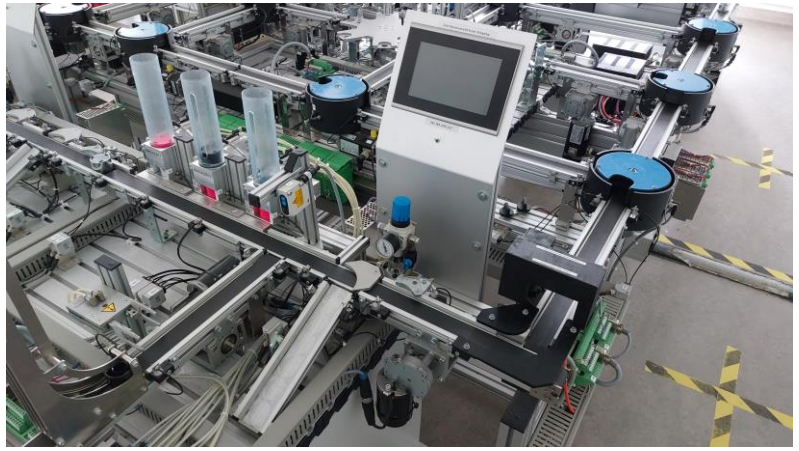
Bit	Digital Actuators
0	Retract Magazine 1 Pusher
1	Extend Magazine 1 Pusher
2	Retract Magazine 2 Pusher
3	Extend Magazine 2 Pusher
4	Extend Ejector
5	Retract Magazine 3 Pusher
6	Extend Magazine 3 Pusher
7	Conveyor on

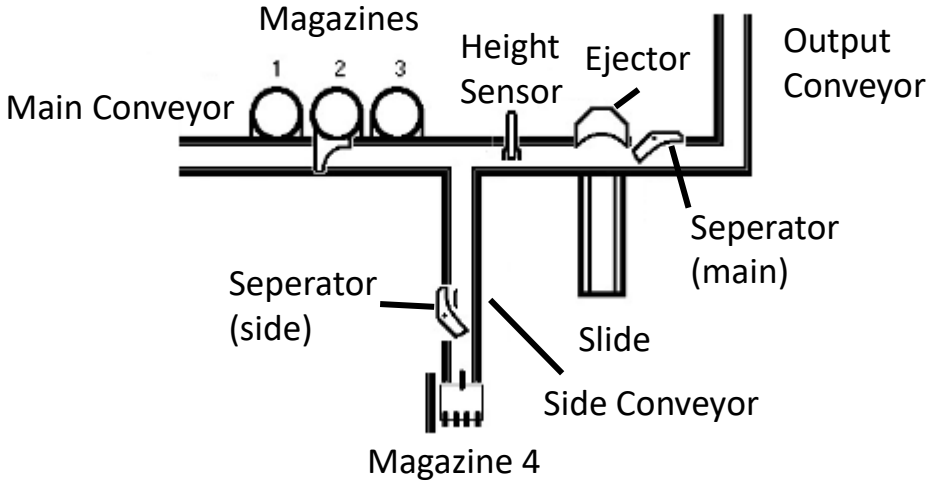
Address	Analog Actuator
0	Height value

Exercise 0

Module: Distribution Center Input (Input 2)







Exercise 0

I/Os Distribution Center Input (Input 2)

Inputs

Bit	Address	Digital Sensor
0	8002	Conveyor (Side) Piece at end
1	8002	Conveyor (Output) Piece at end
2	8002	Conveyor (Main) Piece at begin
3	8002	Conveyor (Main) Piece between Mag 1+2
4	8002	Conveyor (Main) Piece between Mag 2+3
5	8002	Conveyor (Main) Piece in front of Ejector
6	8002	Seperator (Main) is set
7	8002	Seperator (Main) Piece in front
8	8002	Seperator (Side) is set
9	8002	Seperator (Side) Piece in front
10	8002	Magazine 1 is retracted
11	8002	Magazine 1 is ejected
12	8002	Piece in Magazine 1
13	8002	Magazine 2 is retracted
14	8002	Magazine 2 is ejected
15	8002	Piece in Magazine 2

Bit	Address	Digital Sensor
16	8001	Magazine 3 is retracted
17	8001	Magazine 3 is ejected
18	8001	Piece in Magazine 3
19	8001	Ejector in right position
20	8001	Ejector in left position
21	8001	Ejector in middle position
22	8001	Ejector lock is set
23	8001	Slide full
24	8001	Height sensor: Piece not ok
25	8001	Height sensor: Measurement correct

Address	Analog Sensor
0	Height value

Exercise 0

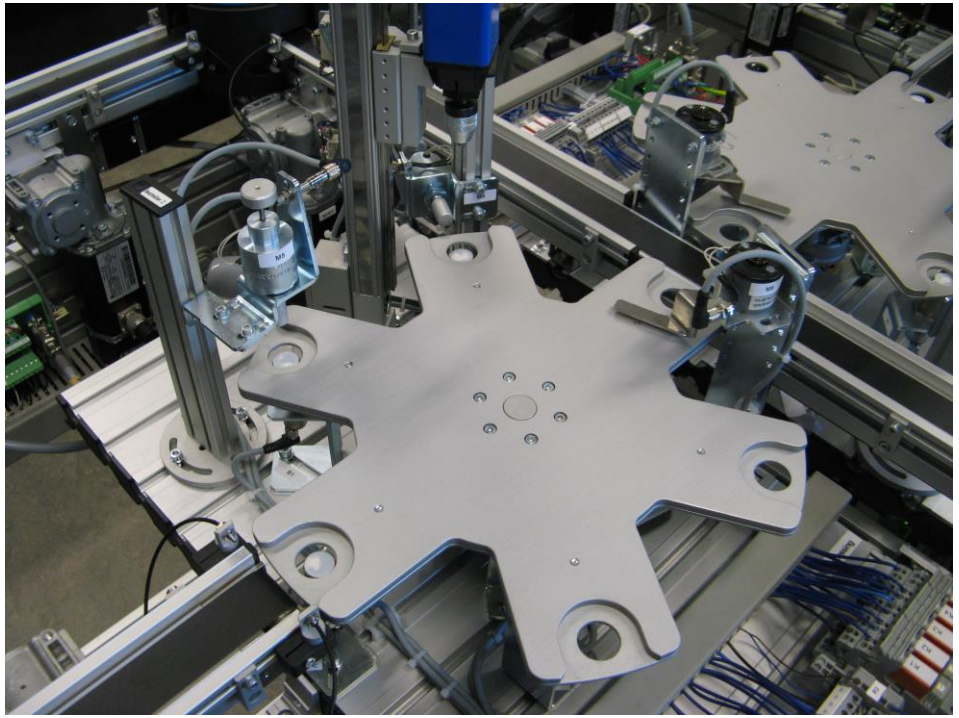
I/Os Distribution Center Input (Input 2)

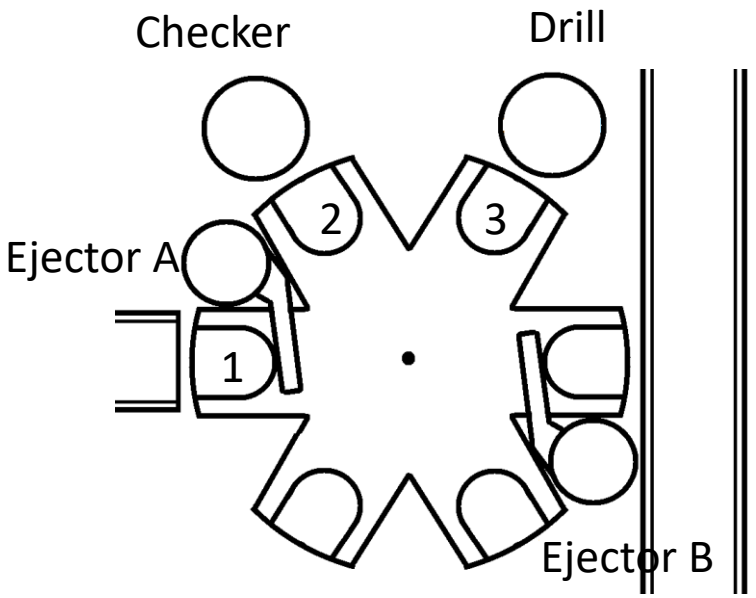
Outputs

Bit	Address	Digital Actuator
0	8012	Stop Conveyor (Main)
1	8012	Slow down Conveyor (Main)
2	8012	Move Conveyor (Main) left
3	8012	Move Conveor (Main) right
4	8012	Turn Conveyor (Side) on
5	8012	Conveyor (Output) backward
6	8012	Conveyor (Output) forward
7	8012	Set Seperator (Main)
8	8012	Set Seperator (Side)
9	8012	Eject Magazine 1
10	8012	Eject Magazine 2
11	8012	Eject Magazine 3
12	8012	Retract Magazine 1
13	8012	Retract Magazine 2
14	8012	Retract Magazine 3
15	8012	Move Ejector right

Bit	Address	Digital Actuator
16	8011	Move Ejector left
17	8011	Activte lock in Ejector (Middle pos)

Address	Analog Actuator
0	Conveyor (Output) Speed




Checker
Drill
Ejector A
1
2
3
Ejector B

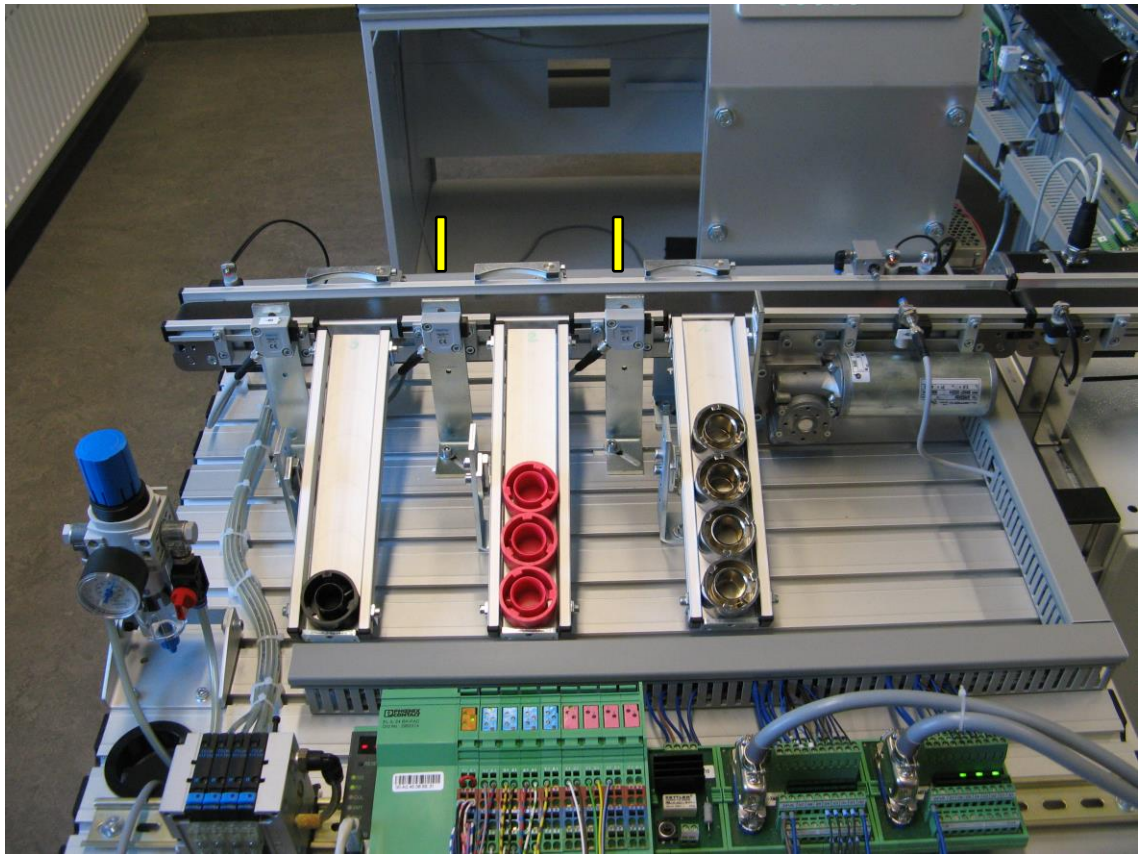
Inputs

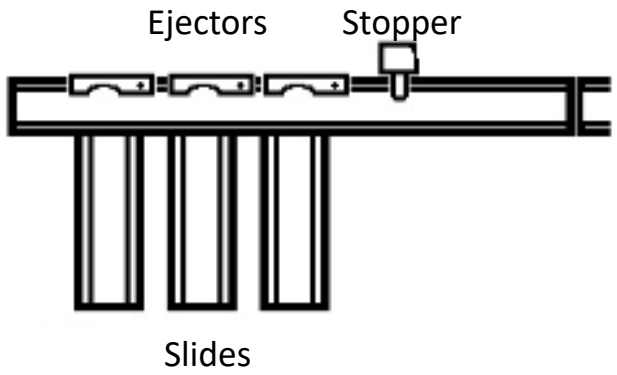
Bit	Sensor
0	Piece in Postion 1
1	Piece in Position 3 (Drill)
2	Piece in Position 2 (Checker)
3	Drill up
4	Drill down
5	Turntable in position
6	Checker fully extended (Piece OK)

Outputs

Bit	Actuator
0	Turn drill on
1	Turn turntable*
2	Move drill up
3	Move drill down
4	Lock piece in drill position
5	Extend checker
6	Extend Ejector B
7	Extend Ejector A

*turntable only needs impulse to rotate exactly one position



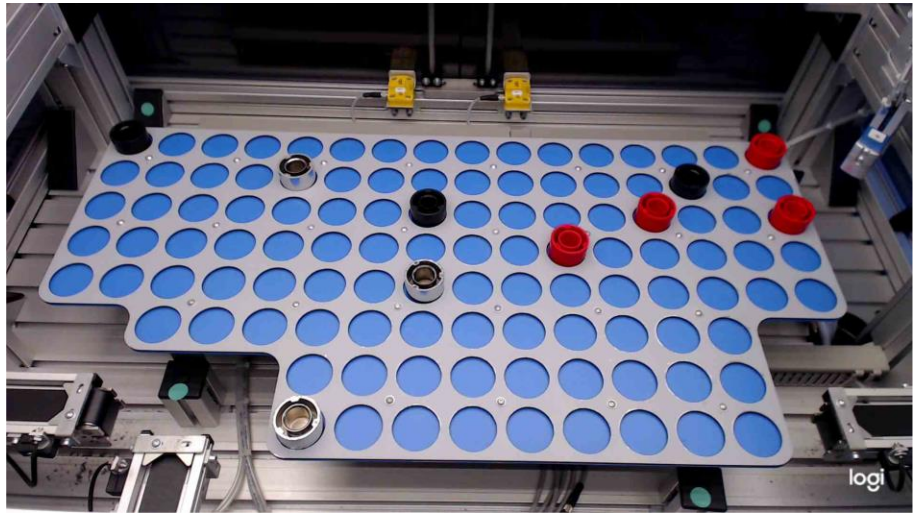

Ejectors
Stopper
Slides

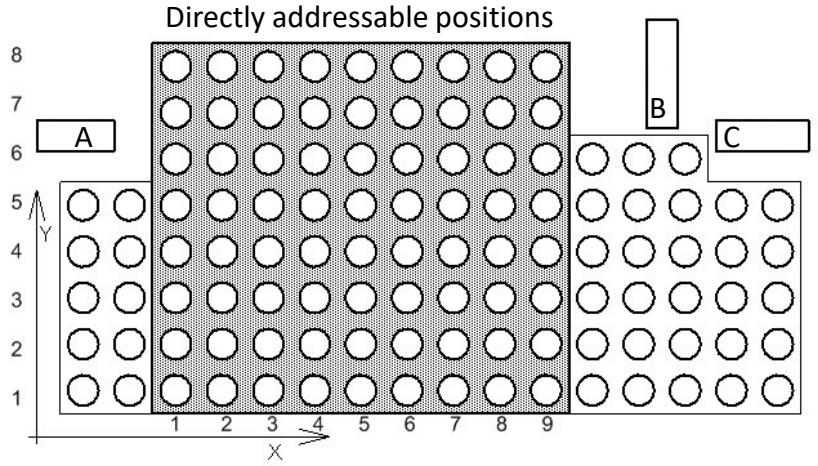
Inputs

Bit	Sensor
0	Piece arrived
1	Piece is metal
2	Piece is not black
3	Slide 1 full
4	Slide 2 full
5	Slide 3 full
6	Piece reached end of conveyor
7	Ejector 1 retracted
8	Ejector 1 extended
9	Ejector 2 retracted
10	Ejector 2 extended
11	Ejector 3 retracted
12	Ejector 3 extended
13	Piece passed Ejector 1
14	Piece passed Ejector 2

Outputs

Bit	Actuator
0	Switch Conveyor on
1	Extend Ejector 1
2	Extend Ejector 2
3	Extend Ejector 3
4	Open Stopper


logi


Directly addressable positions
A
B
C
8
7
6
5
4
3
2
1
Y
1 2 3 4 5 6 7 8 9
X

Inputs

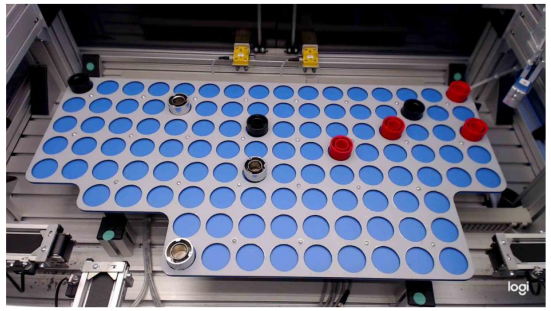
Bit	Sensor
0	Movement in x position allowed
1	Movement in x position done
2	Movement in y position allowed
3	Movement in y position done
4	Gripper up
5	Gripper down
6	Gripper open
7	Gripper closed
8	Piece under gripper*
9	Safety door closed

*always senses piece one position right of the gripper position

Outputs

Bit	Actuator
0-3	X coordinate**
4	Start movement in x direction
5-8	Y coordinate**
9	Start movement in y direction
10	Move gripper up
11	Move gripper down
12	Open gripper
13	close gripper
14	Allow movement
15	Turn light on

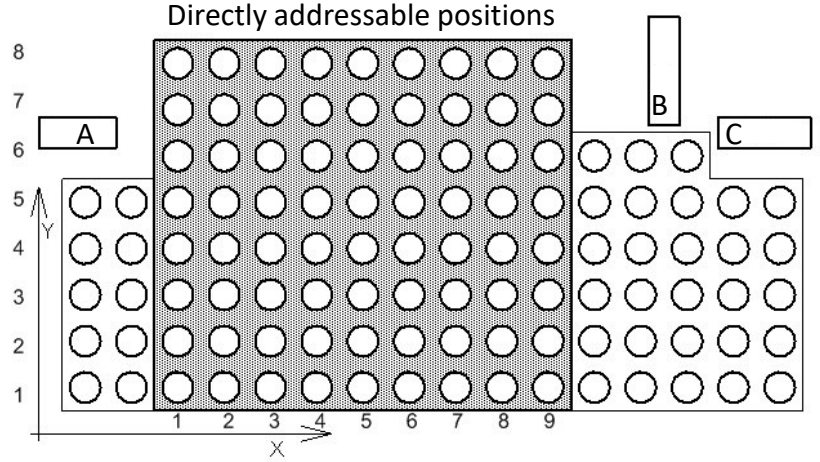
**the coordinates are mapped in 4 bits each, allowing values between 0 and 15



Dec	Function
0	Reference Position of the linear axis
1-9	x/y coordinate
10	Conveyor A
11	Conveyor B
12	Conveyor C
13	-
14	Single Step in positive direction
15	Single Step in negative direction

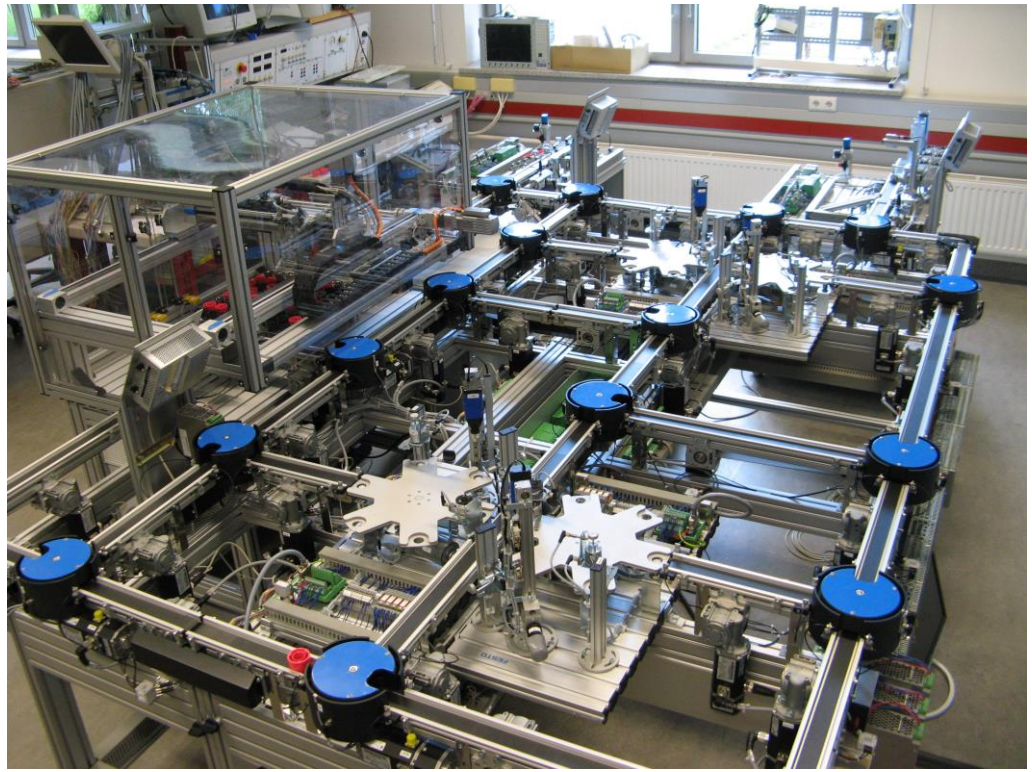
Bit	Actuator
0-3	X coordinate**
4	Start movement in x direction
5-8	Y coordinate**
9	Start movement in y direction
10	Move gripper up
11	Move gripper down
12	Open gripper
13	close gripper
14	Allow movement
15	Turn light on

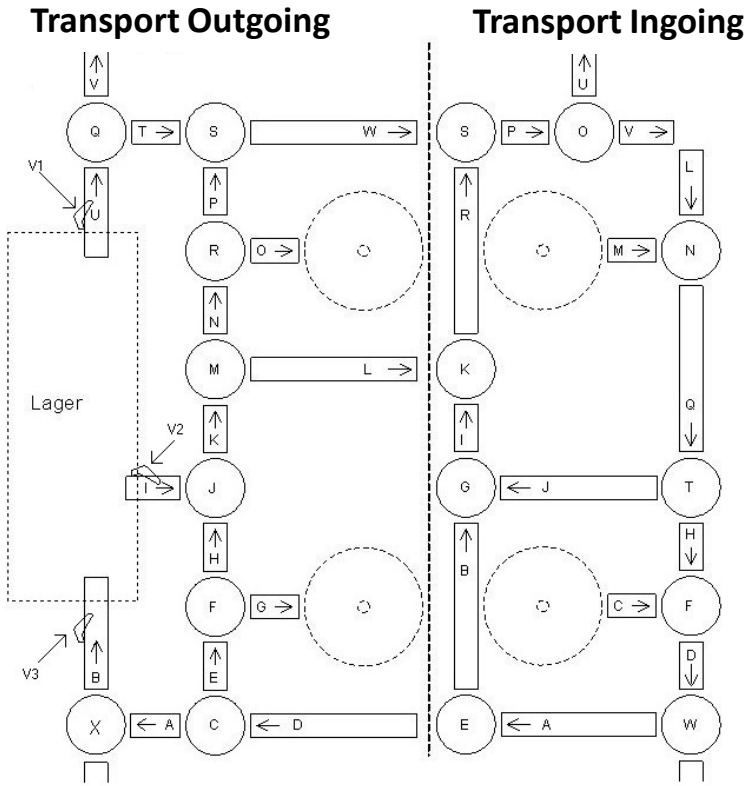
Directly addressable positions

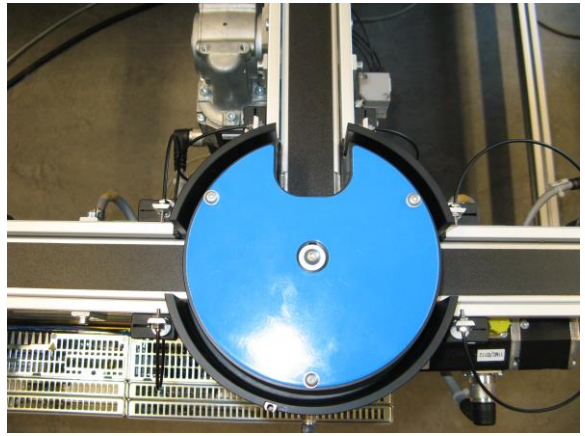


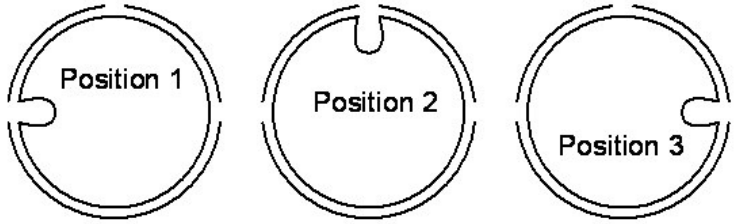
Exercise 0

Modules: Transportation



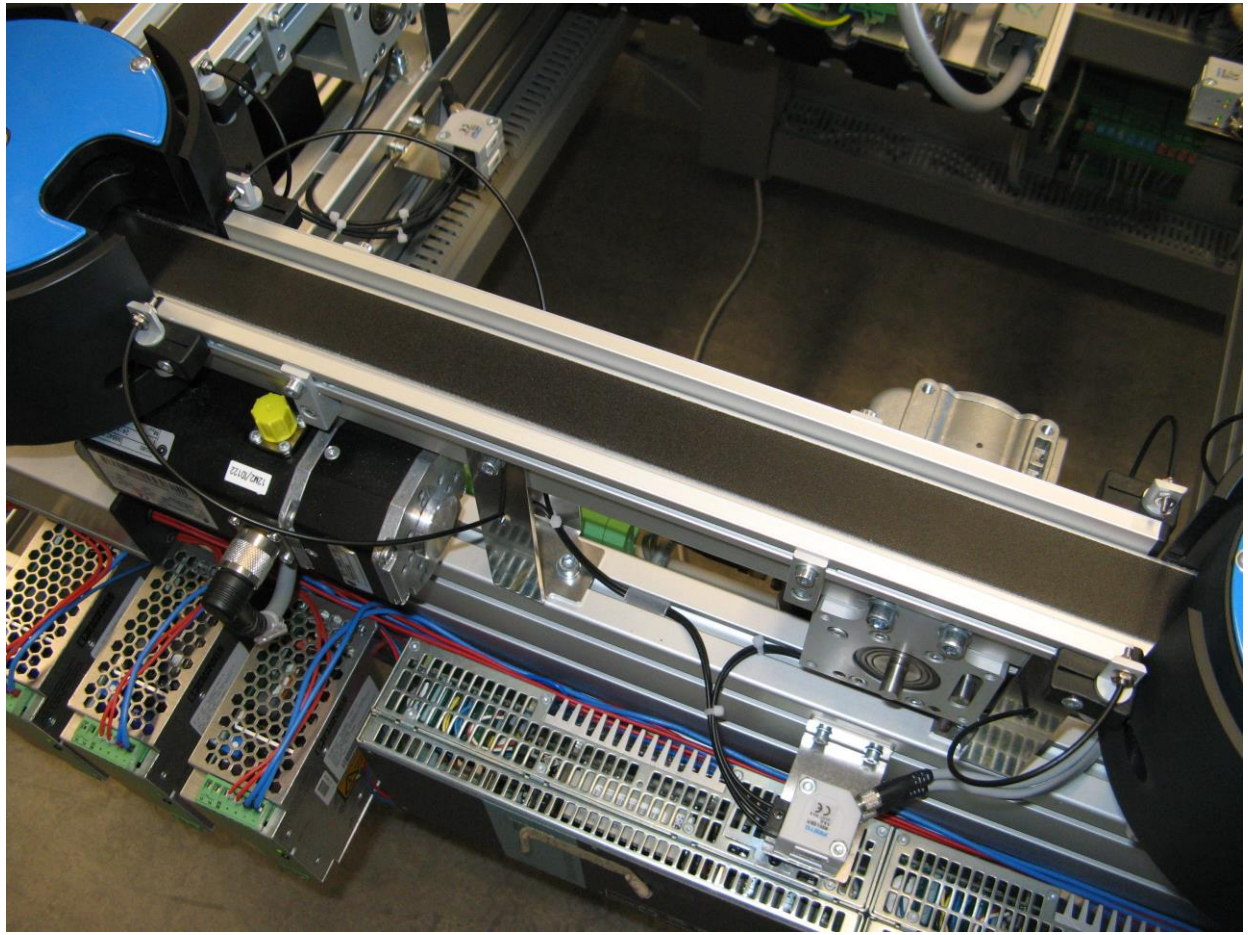


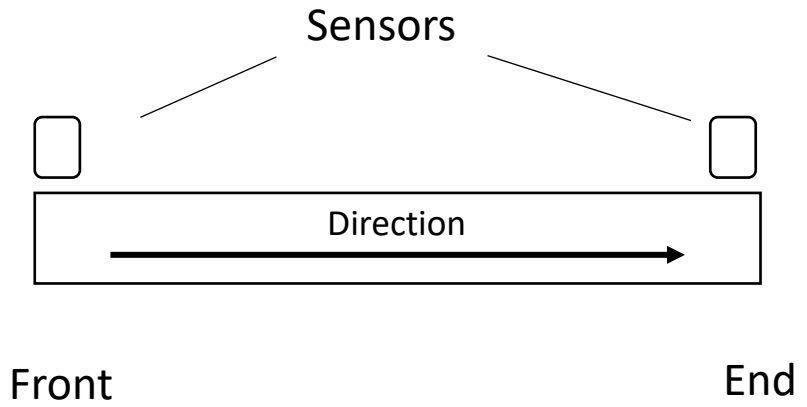



Position 1
Position 2
Position 3

Exercise 0

Conveyor belts in the Transportation Modules





Transport Input - Conveyors

Actuators

Conveyor	Register	Forward	Backward
A	8019	0	1
B	8019	2	3
C	8019	4	5
D	8019	6	7
H	8018	4	5
I	8018	6	7
J	8018	8	9
L	8018	14	15
M	8021	0	1
P	8021	10	11
Q	8021	12	13
R	8021	14	15
U	8020	8	9
V	8020	10	11

Sensors

Conveyor	Register	Piece at front	Piece at end
A	8002	0	1
B	8002	2	3
C	8002	4	5
D	8002	6	7
H	8001	4	5
I	8001	6	7
J	8001	8	9
L	8001	14	15
M	8004	0	1
P	8004	10	11
Q	8004	12	13
R	8004	14	15
U	8003	8	9
V	8003	10	11

Transport Input - Switches

Switch	Register	Ref.	Pos 1	Pos 2	Pos 3
E	8019	8	9	10	11
F	8019	12	13	14	15
G	8018	0	1	2	3
K	8018	10	11	12	13
N	8021	2	3	4	5
O	8021	6	7	8	9
S	8020	0	1	2	3
T	8020	4	5	6	7
W	8020	12	13	14	15

Actuators

Switch	Register	Pos reached	In movement	Piece in switch
E	8002	8	9	10
F	8002	12	13	14
G	8001	0	1	2
K	8001	10	11	12
N	8004	2	3	4
O	8004	6	7	8
S	8003	0	1	2
T	8003	4	5	6
W	8003	12	13	14

Sensors

Exercise 0

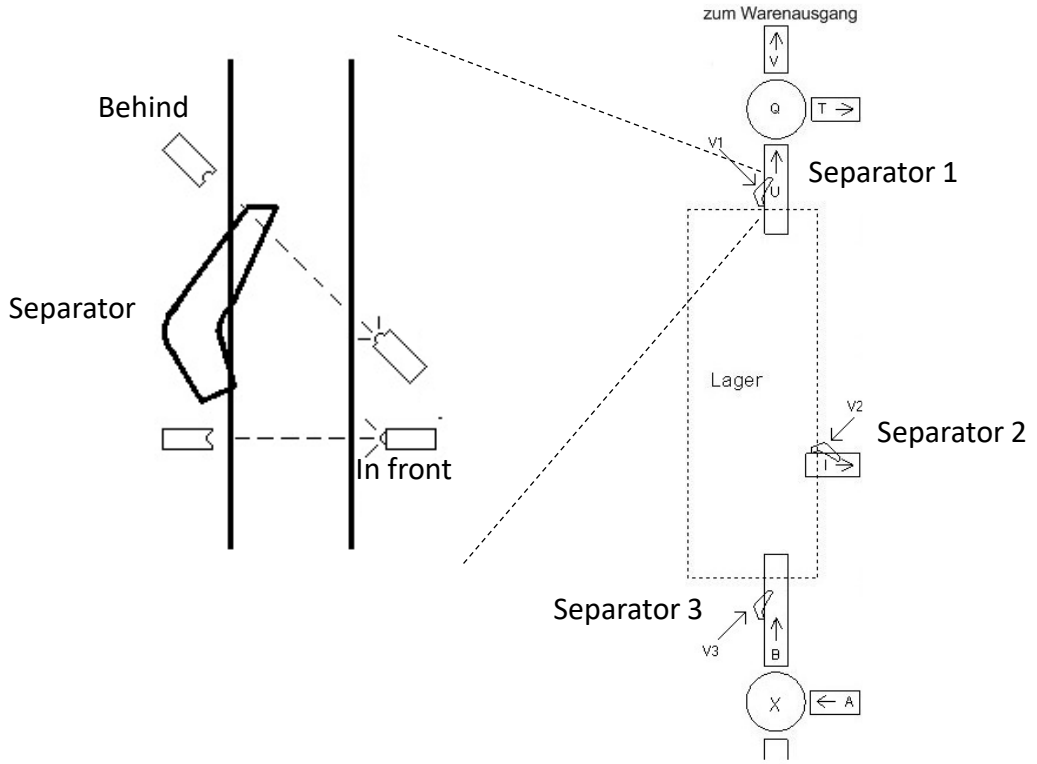
Separators in the outgoing Transportation module (inside storage)

Actuators

Separator	Set
1	64
2	67
3	70

Sensors

Separator	Piece behind	Piece in front
1	65	66
2	68	69
3	71	72



Transport Output - Conveyors

Actuators

Conveyor	Register	Forward	Backward
A	8019	0	1
B	8019	2	3
D	8019	8	9
E	8019	10	11
G	8018	0	1
H	8018	2	3
I	8018	4	5
K	8018	10	11
L	8018	12	13
N	8021	2	3
O	8021	4	5
P	8021	6	7
T	8020	4	5
U	8020	6	7
V	8020	8	9
W	8020	10	11

Sensors

Conveyor	Register	Piece at front	Piece at end
A	8002	0	1
B	8002	2	3
D	8002	8	9
E	8002	10	11
G	8001	0	1
H	8001	2	3
I	8001	4	5
K	8001	10	11
L	8001	12	13
N	8004	2	3
O	8004	4	5
P	8004	6	7
T	8003	4	5
U	8003	6	7
V	8003	8	9
W	8003	10	11

Transport Output - Switches

Switch	Register	Ref.	Pos 1	Pos 2	Pos 3
C	8019	4	5	6	7
F	8019	12	13	14	15
J	8018	6	7	8	9
M	8018/8021	14	15	0	1
Q	8021	8	9	10	11
R	8021	12	13	14	15
S	8020	0	1	2	3
X	8020	12	13	14	15

Actuators

Switch	Register	Pos reached	In movement	Piece in switch
C	8002	4	5	6
F	8002	12	13	14
J	8001	6	7	8
M	8001/8004	14	15	1
Q	8004	8	9	10
R	8004	12	13	14
S	8003	0	1	2
X	8003	12	13	14

Sensors