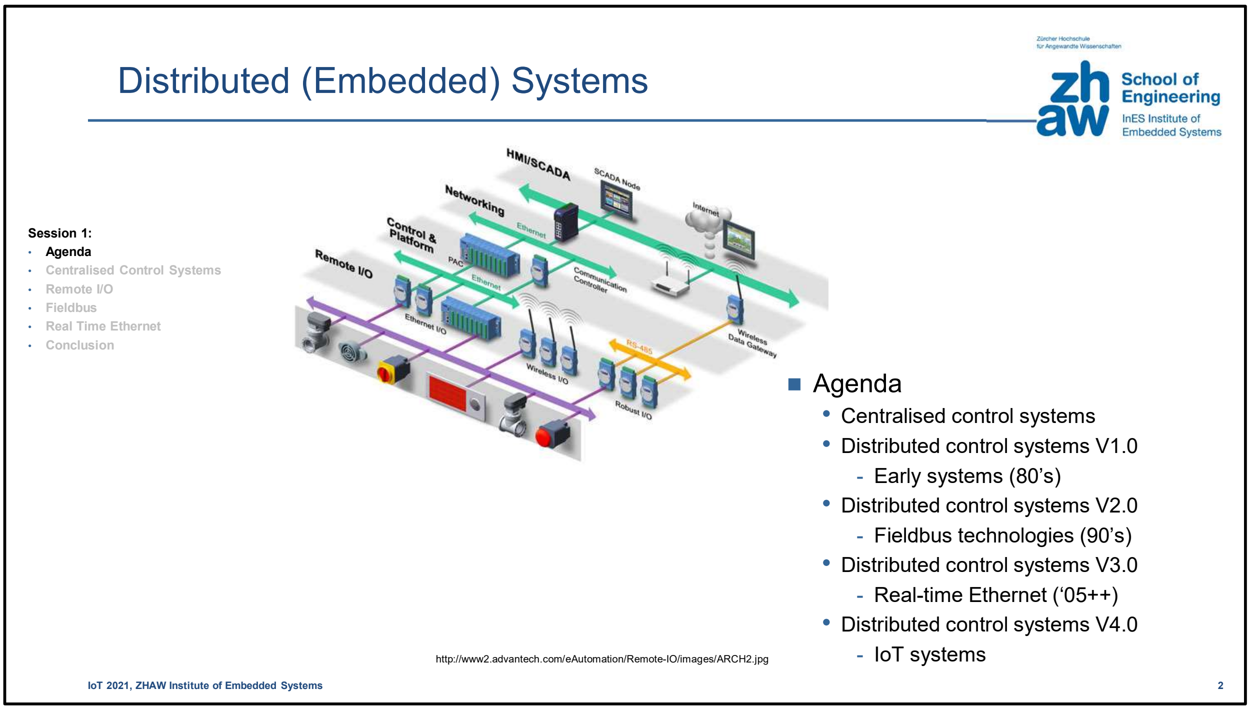
Distributed Embedded Systems

Zürcher Hochschule für Angewandte Wissenschaften

School of Engineering

InES Institute of Embedded Systems



HMI/SCADA
SCADA Node
Networking
Internet
Ethernet
Control & Platform
PAC
Ethernet
Communication Controller
Remote I/O
Ethernet I/O
Wireless I/O
RS-485
Wireless Data Gateway
Robust I/O

Distributed (Embedded) Systems

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

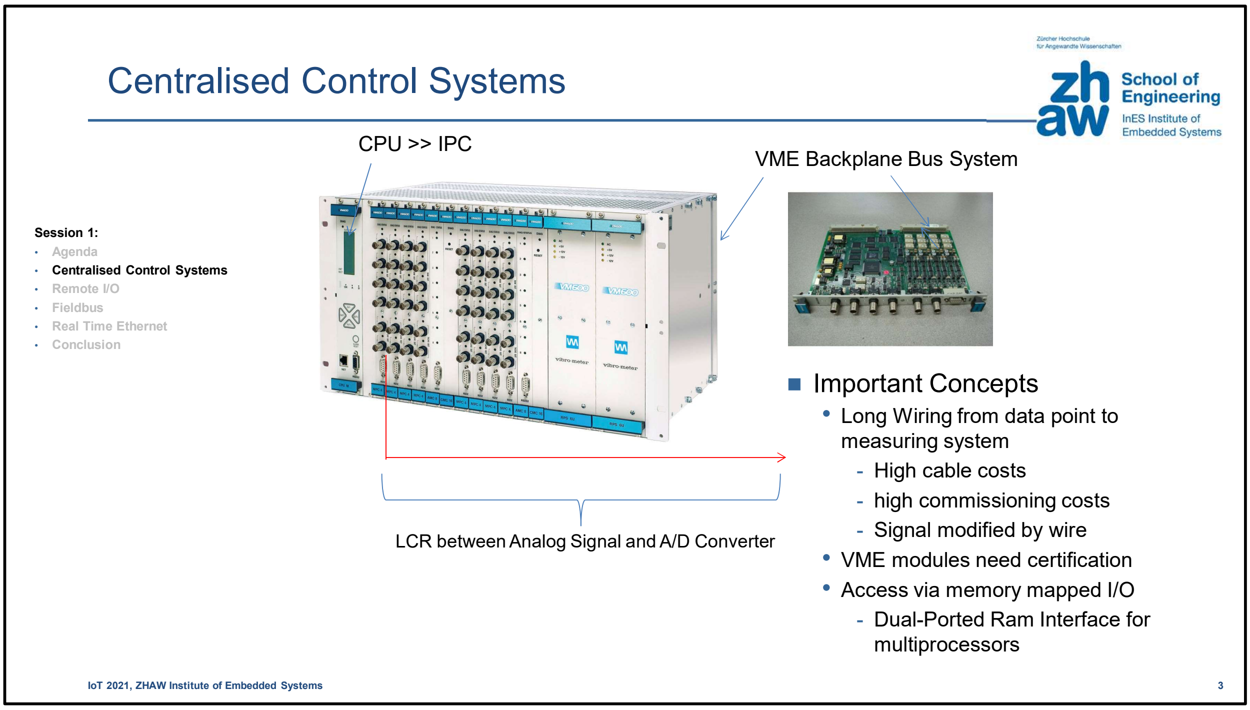
Session 1:
- **Agenda**
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- Agenda
  - Centralised control systems
  - Distributed control systems V1.0
    - Early systems (80’s)
  - Distributed control systems V2.0
    - Fieldbus technologies (90’s)
  - Distributed control systems V3.0
    - Real-time Ethernet (‘05++)
  - Distributed control systems V4.0
    - IoT systems

http://www2.advantech.com/eAutomation/Remote-IO/images/ARCH2.jpg

So we want to get to the point where we link up “professional” technology, be it for factory or building automation to IoT and Industrial Internet of Things (IIoT.) The advance towards IIoT can be clearly demonstrated by the introduction of various concepts over the last 50 or so years. Far from being a lesson in technical archaeology, the systems we describe here are still widely used in industry and so are worth knowing about for their own sake.

We start with centralised control, where the I/O of a computer system was generally speaking kept close to the central processing unit. It became obvious that some form of distribution was required, not least in connectivity to “intelligent” systems (systems with more computing power which we term distributed systems V1.0 – where remote units were abstracted as the equivalent of a user terminal. For reasons we shall consider later, this kind of solution became unsatisfactory and a shift to the fieldbus became necessary. This overdue development brought up some important concepts helping to drag distributed systems into the 20th century. A small advancement was possible by the advent of Real-Time-Ethernet which also resulted in APL, a technology we are likely to see deployed in the IoT as well as the IIoT world. From these developments, the likely further development of IoT can be anticipated.



Centralised Control Systems

Zürcher Hochschule für Angewandte Wissenschaften
zh aw School of Engineering
InES Institute of Embedded Systems

Session 1:

- Agenda
- **Centralised Control Systems**
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

CPU >> IPC

VME Backplane Bus System

LCR between Analog Signal and A/D Converter

- Important Concepts
  - Long Wiring from data point to measuring system
    - High cable costs
    - high commissioning costs
    - Signal modified by wire
  - VME modules need certification
  - Access via memory mapped I/O
    - Dual-Ported Ram Interface for multiprocessors

IoT 2021, ZHAW Institute of Embedded Systems

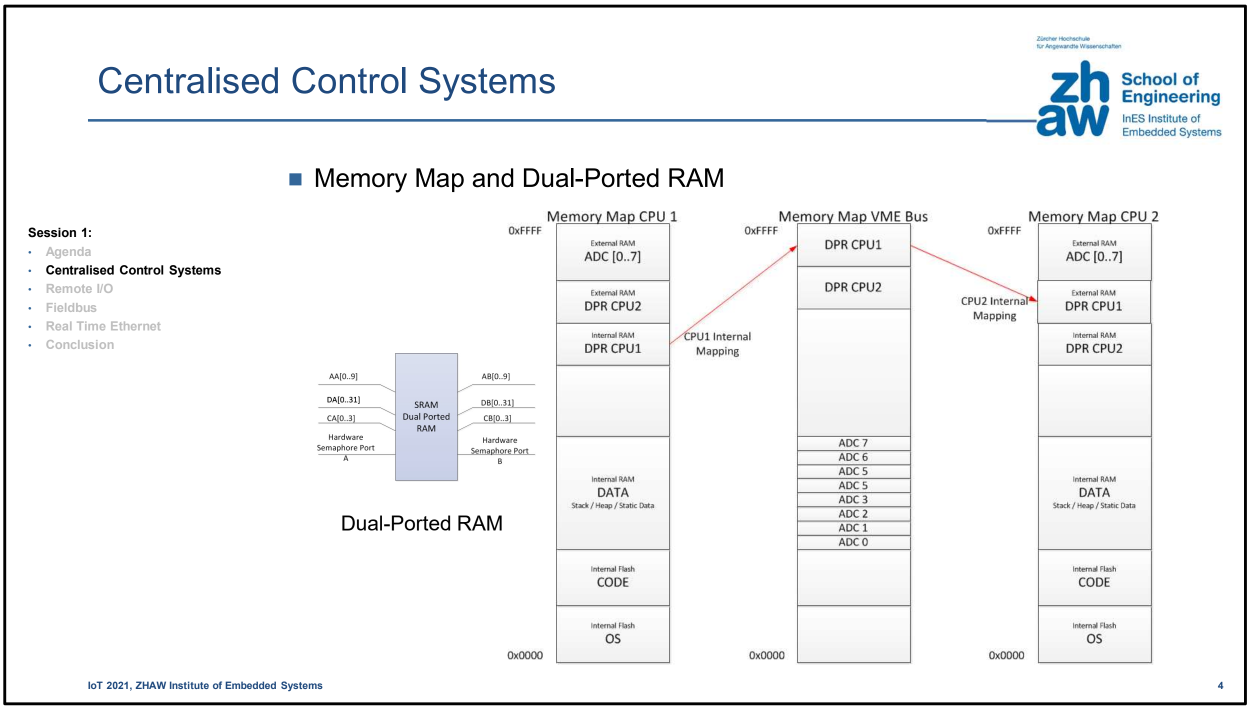
3

This is a vibration monitoring system from Vibrometer/Meggitt. Vibration detectors are placed on equipment, for instance turbine shafts. The vibration is measured and if the frequency of vibration exceeds a limit, an emergency shutdown is initiated so that the risk of expensive-to-repair damage is minimised. A turbine shaft - the rotating bit between the turbine and the energy converter (physical to electrical) - can be 100 meters long. This is technology for insurance relevant problems. If a turbine flies off the shaft or the shaft breaks, repairs have to be paid by somebody so we generally see a logging system attached where the measured values are stored for later reference. This general architecture is also seen in Industrial IoT systems (IIoT) where terabytes of data/hour can be generated but where the data is distributed across the intranet into cloud-based technology and storage.

Versa Modular Eurocard (VME) technology is one example of a technology used to put together high-performance measurement and control units. When the author was developing this kind of system (1992) a rack like this could easily cost 30kCHF – I don’t think they have gotten any cheaper. The rack consists of, on the very left a CPU and 8 A/D converter cards, with 2 status cards (bedside groups of four converter cards and a redundant power supply (right). The cards are connected to each other by a backplane bus – in this case 2 * 96 pol connectors. The VME bus system as such could accommodate 21 cards and is multi processor capable. The high cost of this and other technologies saw the introduction of Industry PC’s (IPC) but the advantages over IPC’s of systems like this were the high quality operating systems (traditional VME -> OS9 -> Unix-like), relatively easy multi-processing capabilities and the ability to make high reliability and functional safe systems.

This is an example of a centralised system where all the active components are in one physical position in the network and the signal sources are in another position. We need cables to connect the analogue conversion cards to the analogue signal source. These cables are sources of noise and signal skew due to inductive and capacitive effects.

VME had the additional advantage that it was multi-CPU capable, about 6 CPU’s could easily either co-exist or cooperate on the bus.



Centralised Control Systems

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Session 1:

- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- Memory Map and Dual-Ported RAM

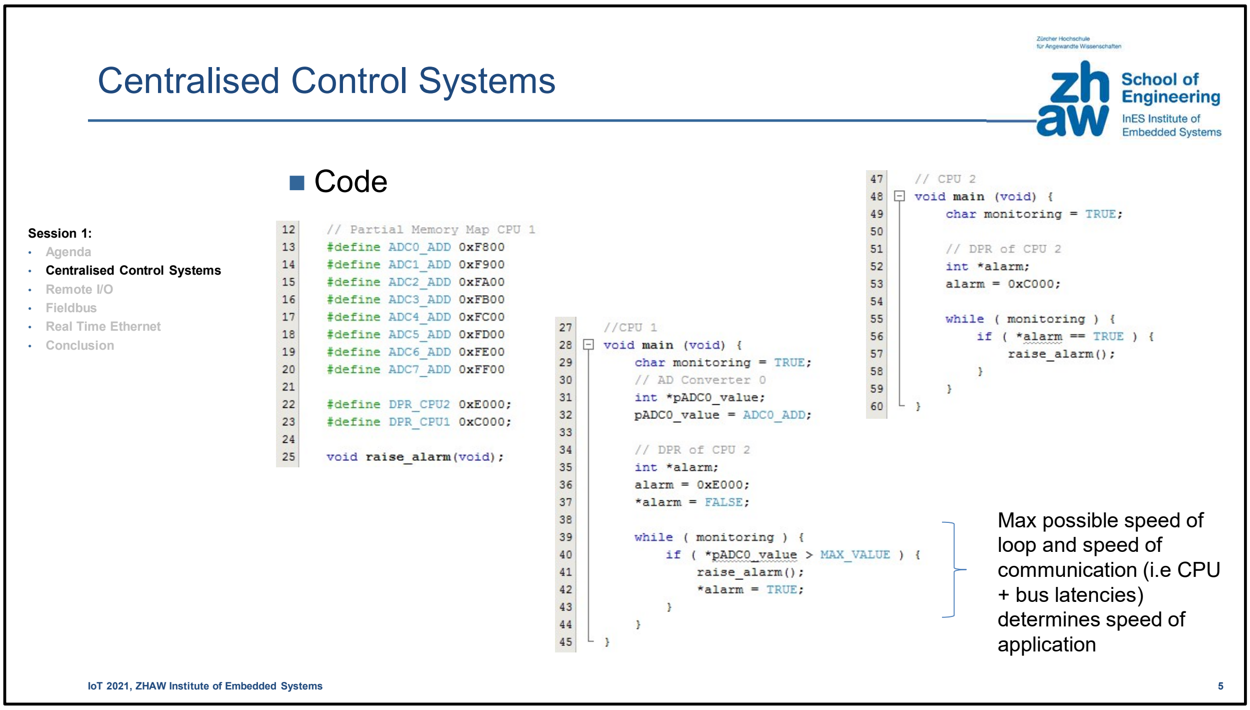
Dual-Ported RAM

In a VME-Bus like environment there is a memory map of a processor board and a memory map of the bus. There is no designated I/O address space so I/O is memory mapped.

Imagine two processors on the same bus. If the addressing of one processor is visible to that of a second processor, then the second processor must wait (stall) before being able to access that bus. Since each processor has other things to do this is clearly inefficient. The two processor boards will sometimes wish to access the bus and then contention occurs, but otherwise the processors are free to run. Therefore the bus system will support a different memory mapping to that of the processor boards.

Nevertheless, the two processors may want to communicate with each other. One way is via a common memory. The most elegant is a small memory on each processor board which can be accessed by bother processors. The only way to achieve this is a dual-ported arrangement, and memory components exists for just that purpose.

These components feature two separate address and data busses. These can access the bus simultaneously so a protection is needed if both access the same location simultaneously, this usually takes the form of a hardware implementation of a semaphore.



Centralised Control Systems

Session 1:
- Agenda
- **Centralised Control Systems**
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- Code

```
12    // Partial Memory Map CPU 1
13    #define ADC0_ADD 0xF800
14    #define ADC1_ADD 0xF900
15    #define ADC2_ADD 0xFA00
16    #define ADC3_ADD 0xFB00
17    #define ADC4_ADD 0xFC00
18    #define ADC5_ADD 0xFD00
19    #define ADC6_ADD 0xFE00
20    #define ADC7_ADD 0xFF00
21
22    #define DPR_CPU2 0xE000;
23    #define DPR_CPU1 0xC000;
24
25    void raise_alarm(void);
```

```
27    //CPU 1
28    void main (void) {
29        char monitoring = TRUE;
30        // AD Converter 0
31        int *pADC0_value;
32        pADC0_value = ADC0_ADD;
33
34        // DPR of CPU 2
35        int *alarm;
36        alarm = 0xE000;
37        *alarm = FALSE;
38
39        while ( monitoring ) {
40            if ( *pADC0_value > MAX_VALUE ) {
41                raise_alarm();
42                *alarm = TRUE;
43            }
44        }
45    }
```

```
47    // CPU 2
48    void main (void) {
49        char monitoring = TRUE;
50
51        // DPR of CPU 2
52        int *alarm;
53        alarm = 0xC000;
54
55        while ( monitoring ) {
56            if ( *alarm == TRUE ) {
57                raise_alarm();
58            }
59        }
60    }
```

Max possible speed of loop and speed of communication (i.e CPU + bus latencies) determines speed of application

To communicate effectively the locations of the variables in the DPR need to be known by both processors – and of course they need the same endianness and types. In this slide on the left we define the memory map as seen by processor 1. Lines 22 and 23 define the start addresses of the DPR of CPU2 and CPU1 respectively.

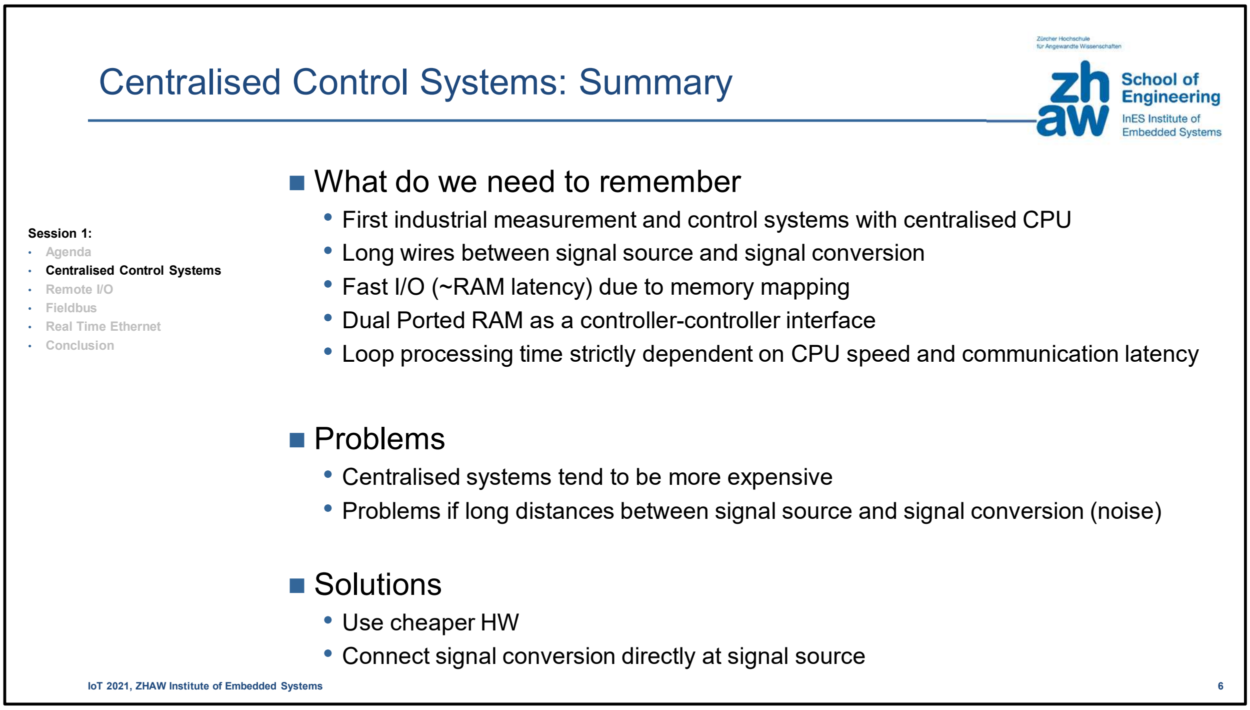
In the middle we have an application that defines a pointer to a variable alarm and locates this variable to the beginning address of the DPR of CPU2. It also sets this variable to FALSE. In other words, CPU 1 puts the value 0x0000 into the first memory location of the DPR of CPU2. The application then, in an endless loop, polls the value of ADC0 and if it is greater than some threshold, sets the value of the first memory location of the CPU2 DPR to 0x0001.

On the right we have the application of CPU 2 which, also in an endless loop, polls the value of the first location of its DPR and if this is set to 0x0001, calls some local routine.

The important points this slide makes is that

1.) The location of variables in the DPR must be known to both CPU1 and CPU2 at compile-time

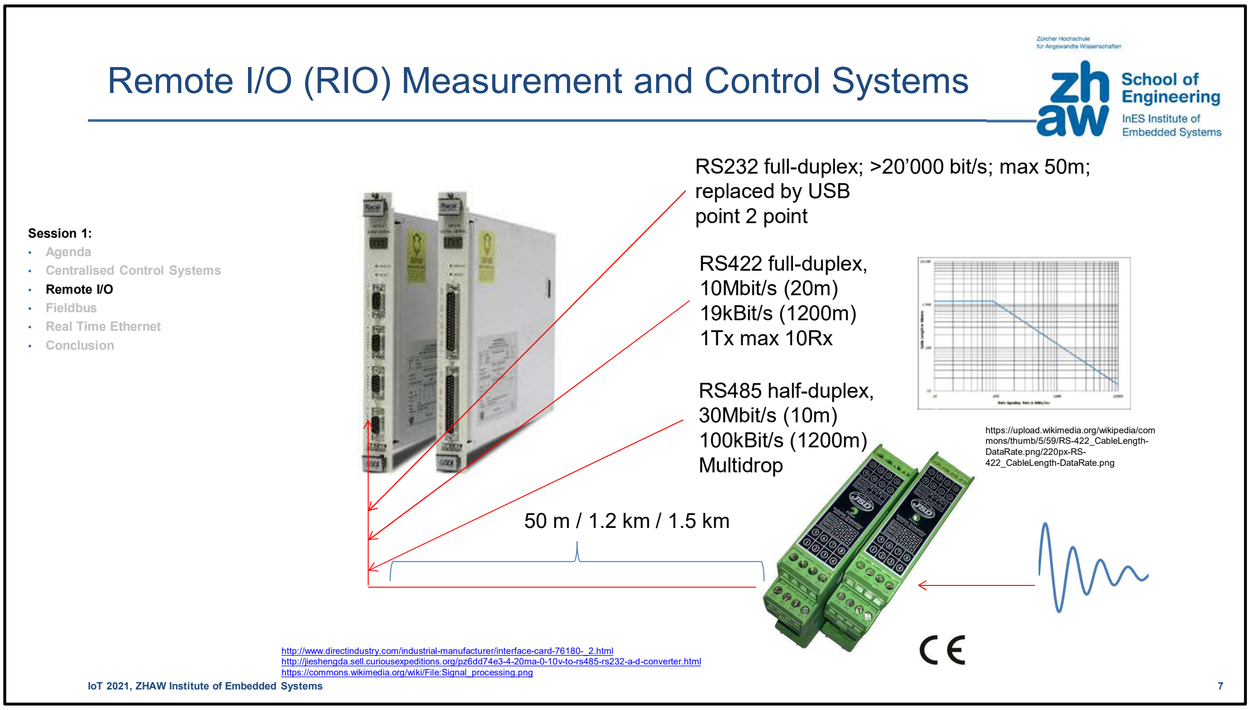
2.) The polling rate is dependent on the speed of the processors.



Centralised Control Systems: Summary

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- What do we need to remember
  - First industrial measurement and control systems with centralised CPU
  - Long wires between signal source and signal conversion
  - Fast I/O (~RAM latency) due to memory mapping
  - Dual Ported RAM as a controller-controller interface
  - Loop processing time strictly dependent on CPU speed and communication latency
- Problems
  - Centralised systems tend to be more expensive
  - Problems if long distances between signal source and signal conversion (noise)
- Solutions
  - Use cheaper HW
  - Connect signal conversion directly at signal source



Remote I/O (RIO) Measurement and Control Systems

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Session 1:

- Agenda
- Centralised Control Systems
- **Remote I/O**
- Fieldbus
- Real Time Ethernet
- Conclusion

RS232 full-duplex; >20’000 bit/s; max 50m;
replaced by USB
point 2 point

RS422 full-duplex,
10Mbit/s (20m)
19kBit/s (1200m)
1Tx max 10Rx

RS485 half-duplex,
30Mbit/s (10m)
100kBit/s (1200m)
Multidrop

https://upload.wikimedia.org/wikipedia/commons/thumb/5/59/RS-422_CableLength-DataRate.png/220px-RS-422_CableLength-DataRate.png

http://www.directindustry.com/industrial-manufacturer/interface-card-76180-_2.html
http://jieshengda.sell.curiousexpeditions.org/pz6dd74e3-4-20ma-0-10v-to-rs485-rs232-a-d-converter.html
https://commons.wikimedia.org/wiki/File:Signal_processing.png

Remote I/O soon became desirable. The obvious thing was to use known technology, that is the signalling systems (physical/datalink) available at the time and used to connect user terminals to a central (mainframe) computer. Certain physical layers could be driven over considerable distances for instance RS422 and RS485, albeit at decreasing baud rates. The advantages are that now I can use cheap hardware to do signal conditioning and conversion at source and digital transport mechanisms to communicate (almost) lossless to a calculation unit. At the computing unit the incoming data could be treated as a stream of bytes or as character-based communication – mostly one way. By treating this RIO as a terminal we can also cheaply benefit from data formatting and low level protocols associated with terminal operation such as; software control flow XON-XOFF; Framing STX, ETX, Data formats (ASCII …) and the operating system interface, namely a COM port.

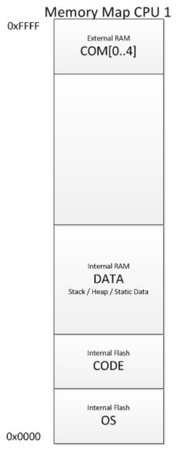
Nevertheless the electrical interfacing problems are non-negligible. Different connectors, different cables, some cable/connectors crossed, some not. In practice it got very messy very quickly.

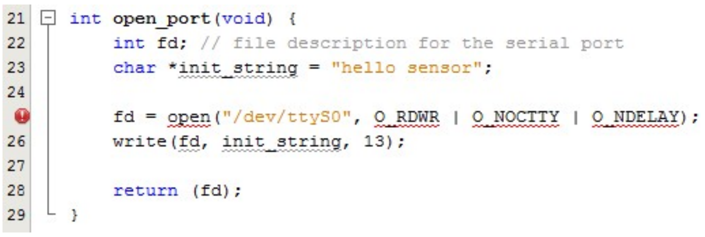
What of course happened then is that RIOs needed to have some kind of configuration capability which eventually resulted in proprietary protocols being implemented. The problem with such protocols is that every manufacturer has his or her own and they become both complicated and act as a disincentive or more specifically a lock-in effect occurs where it becomes very difficult for customers to integrate products from different manufacturers.

Remote I/O (RIO) Measurement and Control Systems

Session 1:
- Agenda
- Centralised Control Systems
- **Remote I/O**
- Fieldbus
- Real Time Ethernet
- Conclusion

■ Memory Map and interface code



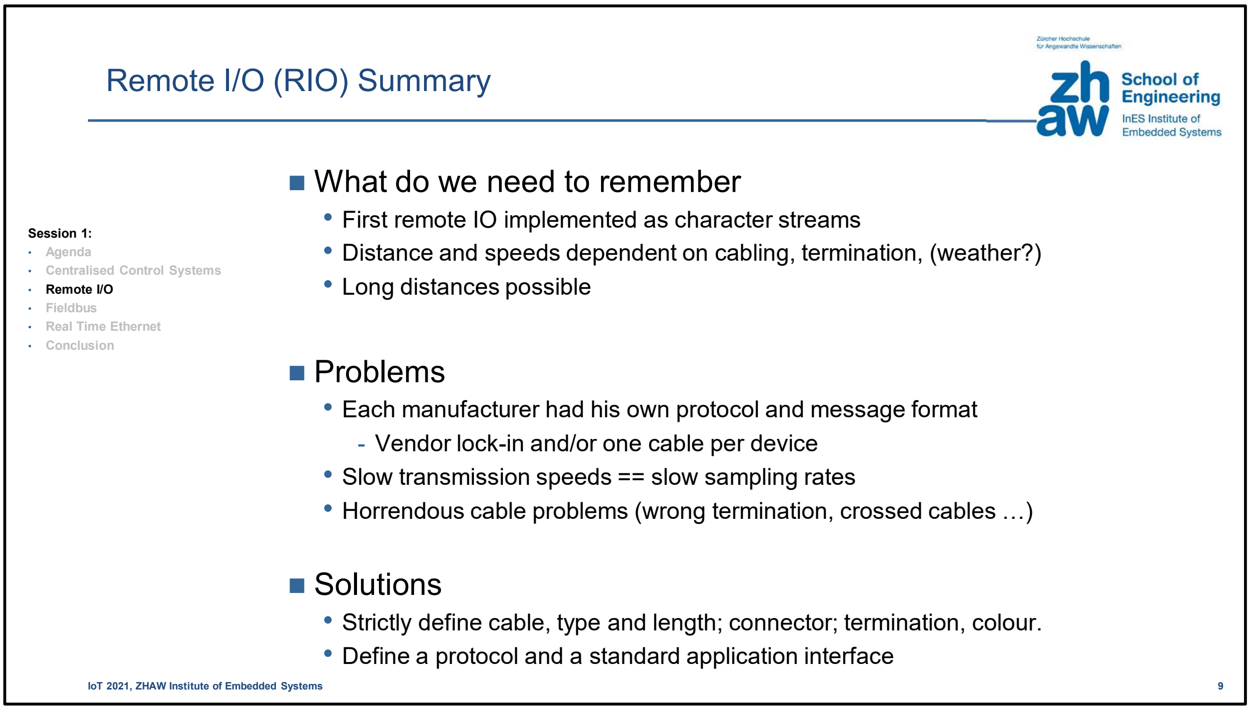


```
int open_port(void) {
    int fd; // file description for the serial port
    char *init_string = "hello sensor";

    fd = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_NDELAY);
    write(fd, init_string, 13);

    return (fd);
}
```

Interfacing to the RIO became as simple as opening a COM port with specific parameters.



Remote I/O (RIO) Summary

Session 1:
- Agenda
- Centralised Control Systems
- **Remote I/O**
- Fieldbus
- Real Time Ethernet
- Conclusion

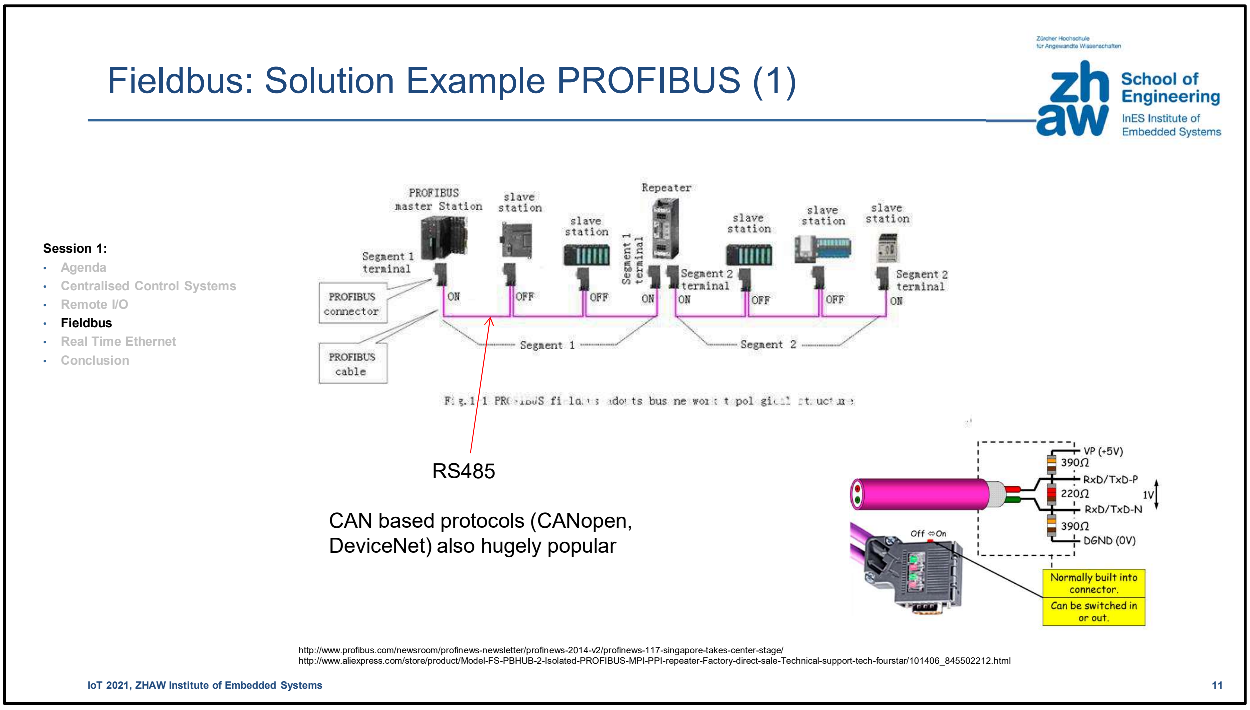
- What do we need to remember
  - First remote IO implemented as character streams
  - Distance and speeds dependent on cabling, termination, (weather?)
  - Long distances possible
- Problems
  - Each manufacturer had his own protocol and message format
    - Vendor lock-in and/or one cable per device
  - Slow transmission speeds == slow sampling rates
  - Horrendous cable problems (wrong termination, crossed cables …)
- Solutions
  - Strictly define cable, type and length; connector; termination, colour.
  - Define a protocol and a standard application interface

Fieldbus

Session 1:

- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

- Why
  - Problems with RIO cause a substantial overhaul
- What
  - Applications
  - Architectures
  - Traffic types
  - Profiles
  - System Start-up



Fieldbus: Solution Example PROFIBUS (1)

Zürcher Hochschule für Angewandte Wissenschaften
zh aw School of Engineering
InES Institute of Embedded Systems

Session 1:

- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

Fig.1.1 PROFIBUS fieldbus adopts bus network topological structure

RS485

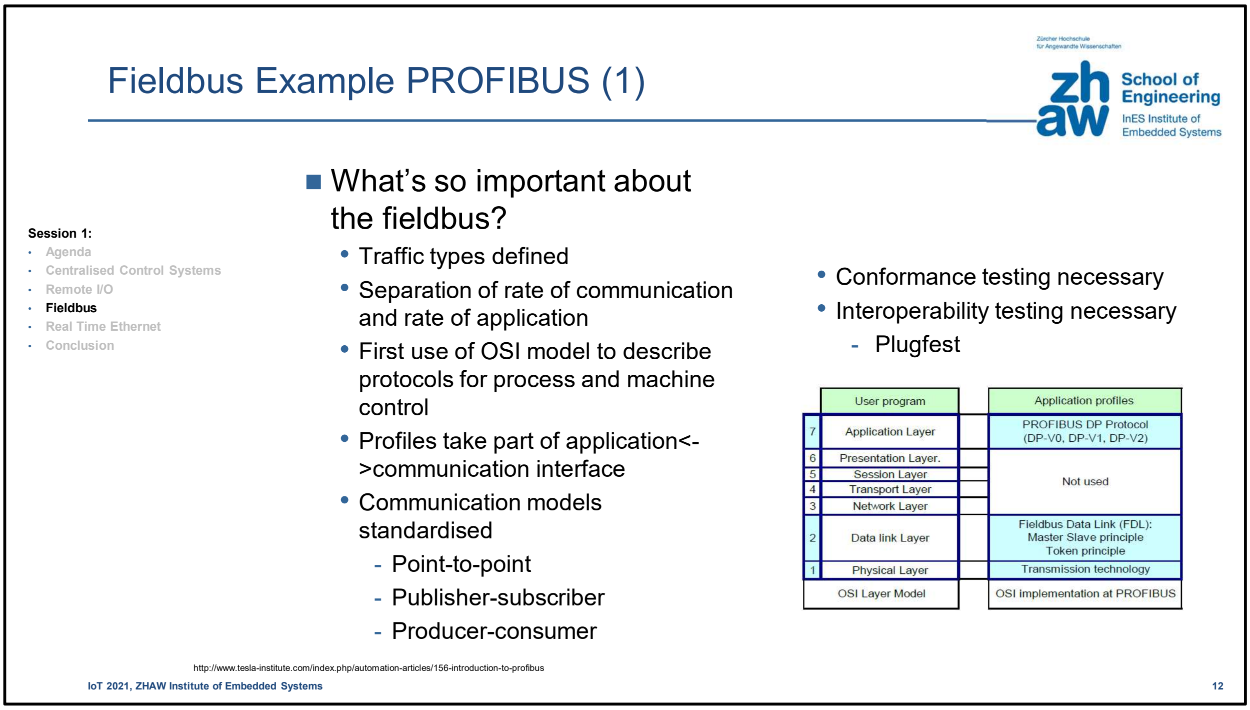
CAN based protocols (CANopen, DeviceNet) also hugely popular

http://www.profibus.com/newsroom/profinews-newsletter/profinews-2014-v2/profinews-117-singapore-takes-center-stage/
http://www.aliexpress.com/store/product/Model-FS-PBHUB-2-Isolated-PROFIBUS-MPI-PPI-repeater-Factory-direct-sale-Technical-support-tech-fourstar/101406_845502212.html

A primary driver of improvements was customers requiring greater flexibility to integrate components into their machines. One way of bringing this about was to define fieldbus protocols by special interest groups out of industry. In practice this meant that several companies developed their own protocol, supported by a special interest group. {Siemens, PROFIBUS}, {Phoenix Contact, InterBus-S}, {Schneider, Modbus} or the CiA – CAN in Automation which is an independent group with lots of small companies and where the big beasts (Siemens, ABB, Schneider) don't have as much say. These special interest groups then define systems in their entirety, generally producing IEC or ISO accepted international standards.

The first question to be answered when defining a protocol is the physical layer. Developing physical layers is expensive and interfacing to specialist physical layers is expensive as well (cost of integrated circuits.) Most fieldbus systems therefore use available physical layers – PROFIBUS uses RS485, CANopen uses the Controller Area Network (CAN) bus made popular and very cheap from the automotive industry.

The commissioning of systems in the field is expensive, so cabling mistakes, hard-to-detect transmission failures ought to be avoided. This is the reason why some fieldbus specifications specify the colour of the cable, the type of connector, the earthing and the termination as well as baudrate, maximum lengths etc.



Fieldbus Example PROFIBUS (1)

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

- What's so important about the fieldbus?
  - Traffic types defined
  - Separation of rate of communication and rate of application
  - First use of OSI model to describe protocols for process and machine control
  - Profiles take part of application<->communication interface
  - Communication models standardised
    - Point-to-point
    - Publisher-subscriber
    - Producer-consumer
  - Conformance testing necessary
  - Interoperability testing necessary
    - Plugfest

	User program	Application profiles
7	Application Layer	PROFIBUS DP Protocol (DP-V0, DP-V1, DP-V2)
6	Presentation Layer.	Not used
5	Session Layer	
4	Transport Layer	
3	Network Layer	
2	Data link Layer	Fieldbus Data Link (FDL): Master Slave principle Token principle
1	Physical Layer	Transmission technology
	OSI Layer Model	OSI implementation at PROFIBUS

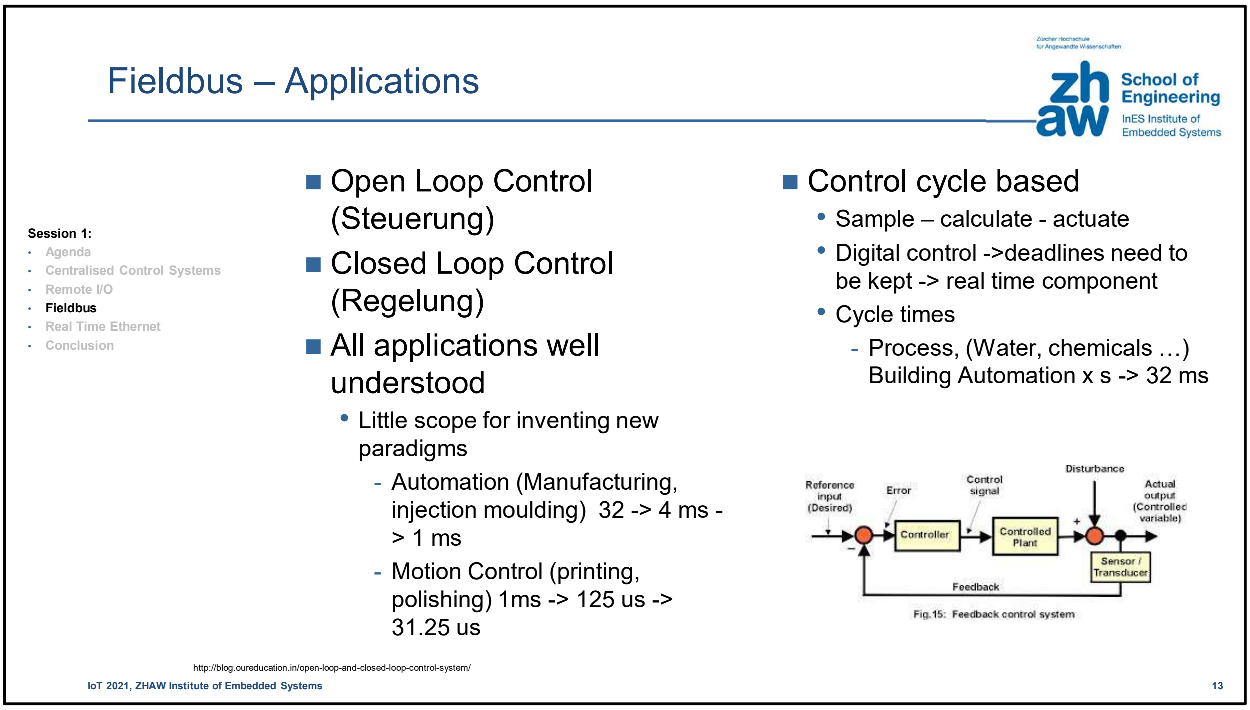
http://www.tesla-institute.com/index.php/automation-articles/156-introduction-to-profibus

The fieldbus revolution, in our context, formally established some important principles namely the idea of traffic types and profiles. The traffic types arise out of the timing and quality of information demands of the application whereas the profiles are a standardisation of the application interface.

As the profile serves as a dual ported RAM and DPR specifically, but memory in general, separate clock domains, the rate of communication can be separated from the rate of application. In other words the communication can process more messages than are handed to the application.

We see the communication models standardised (= generally understood and agreed to be best practice for a particular application type)

The VME bus world was predominately an electrical interface so conformance to electrical standards needed to be tested to ensure that the systems would function over the specified climatic and other ranges. In the serial communication world we also find that interoperability testing is required, that is to ensure that on an application layer, the devices really do understand each other.



Fieldbus – Applications

Zürcher Hochschule für Angewandte Wissenschaften
zh aw School of Engineering
InES Institute of Embedded Systems

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- Open Loop Control (Steuerung)
- Closed Loop Control (Regelung)
- All applications well understood
  - Little scope for inventing new paradigms
    - Automation (Manufacturing, injection moulding) 32 -> 4 ms -> 1 ms
    - Motion Control (printing, polishing) 1ms -> 125 us -> 31.25 us

- Control cycle based
  - Sample – calculate - actuate
  - Digital control ->deadlines need to be kept -> real time component
  - Cycle times
    - Process, (Water, chemicals …) Building Automation x s -> 32 ms

Fig.15: Feedback control system

http://blog.oureducation.in/open-loop-and-closed-loop-control-system/

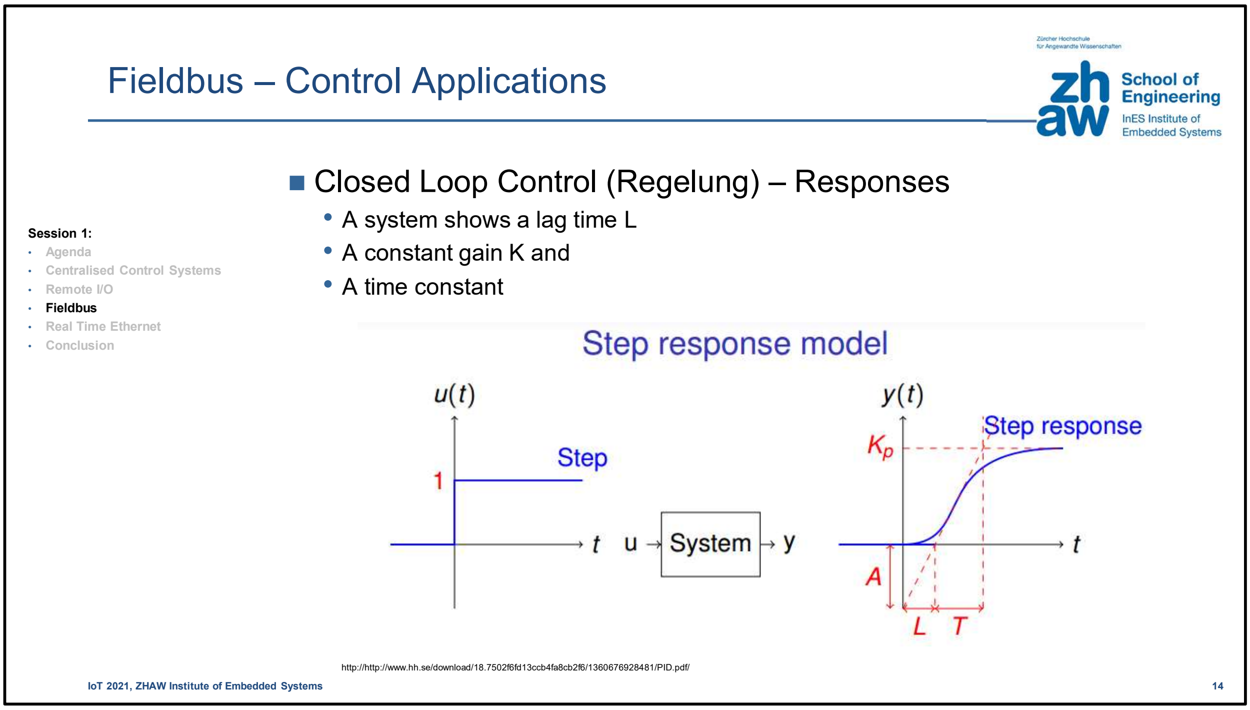
In terms of control applications there are two types, **open** and **closed loop control**. These are well understood paradigms.

Open loop control is for instance opening a garage door. The door opener can be turned on and the garage door opens. The opening mechanism can be made to stop under two conditions, either an end-switch is closed or we measure the “on-time” of the opening mechanism. The latter is not preferred because in practice motors slip over time and after a couple of years operation the garage door will only half open. The former is preferred for this reason, as long as the position of the switch doesn’t move, the door will stop opening at the same physical position every time. The door will however only go in one direction during the opening process and in one direction in the closing process.

In closed loop control – to carry the analogy further – the precise position of the door will be controlled, this might involve the door going backward and forward in an opening process until the controller is satisfied with the final position.

Another example of the two is a water kettle. In an open loop system the water will boil until a thermostat turns the heater off. In closed loop system the water will boil and be kept at a constant temperature regardless of what happens outside of the kettle. In the diagram above the HW/SW manipulating the heater is the controller and the water/kettle is the controlled plant. The sensor, obviously, being the temperature sensor. So close-loop control consists of taking a expected value and subtracting the actual value from it and using this difference as an input value for a controller to control the plant (not the biological plant, the process plant.)

Another way of looking at it is that open loop control, unlike closed loop control, does not follow the **sample-calculate-actuate** paradigm. There is no sampling or calculating, only actuating or not. In closed-loop control, a controller samples a signal, calculates an output and actuates that output on the controlled plant. For reasons we shall see in the next slide, this involves keeping time.



Zürcher Hochschule für Angewandte Wissenschaften

zh aw School of Engineering

InES Institute of Embedded Systems

Fieldbus – Control Applications

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

- Closed Loop Control (Regelung) – Responses
  - A system shows a lag time L
  - A constant gain K and
  - A time constant

Step response model

http://http://www.hh.se/download/18.7502f6fd13ccb4fa8cb2f6/1360676928481/PID.pdf/

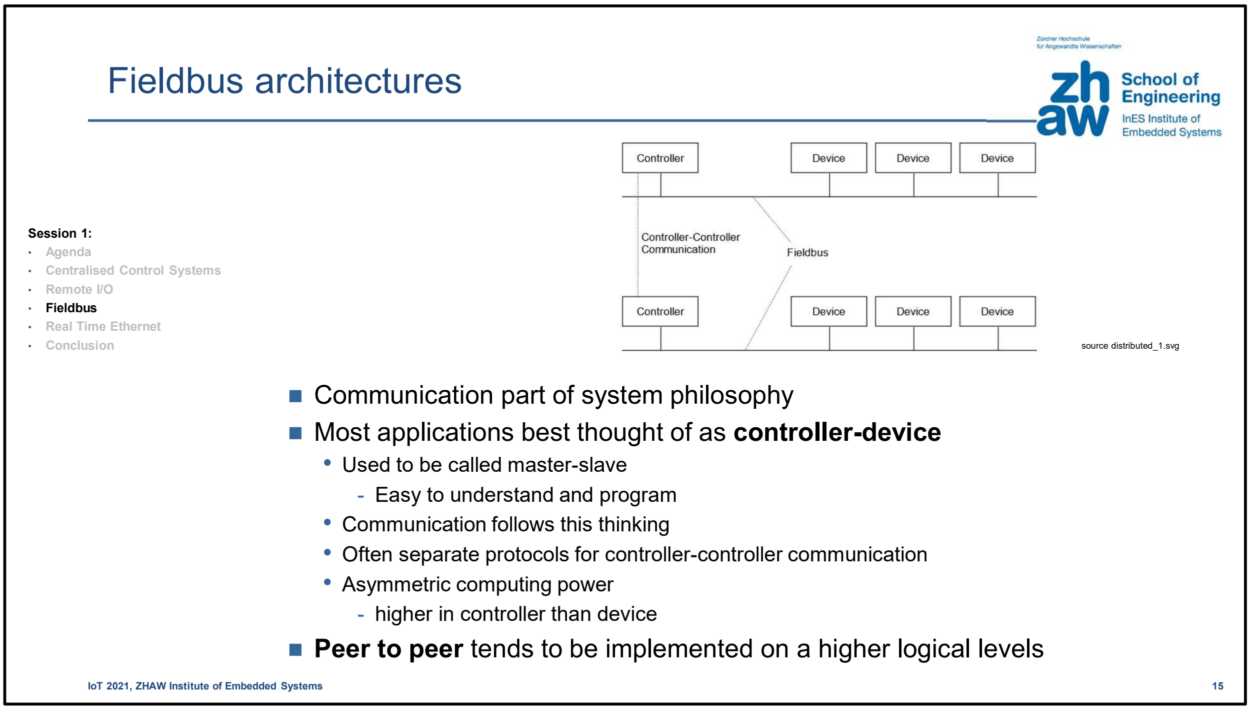
IoT 2021, ZHAW Institute of Embedded Systems

14

If turn the heater on, it will take time for the room to heat up and it will take time for the room to cool down having turned the heater off. These times need not be symmetrical. Turning the heater on is equivalent to a **step-input** (left.) The temperature sensor will show a response to this heater similar to the diagram on the right. That is it will take time (time lag L) for the heated air around the radiator (presuming convection) to reach the sensor and it will take time (characteristic time constant T) before the ambient temperature will stabilise to a temperature. The further away from the heater the longer the lag (L) and probably also the T – depending on the thermal masses in the room.

Sampling the response to an action (turning on the heater) must be done after a time L+T, when the output is stabilised. Otherwise the controller thinks that its actuation wasn’t enough and will actuate even more. This will cause overshoot, which in excess is bad. This time therefore dominates the choice of sample time, which must be within small percentages of L+T.

In a digital system, this sample time ought/must be constant and synchronised to L+T. after which the calculate and the application of the actuation, the step, is expected to take a small fraction of time and then the system can wait for another L+T time before taking the next sample. In a distributed system the sensor delivers values to the controller every L+T time. The controller calculates and, over an (remote) output device, actuates. This sample->sample time is therefore visible in communication over the communication bus and we refer to it as the **cycle time**.



Fieldbus architectures

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

source distributed_1.svg

- Communication part of system philosophy
- Most applications best thought of as **controller-device**
  - Used to be called master-slave
    - Easy to understand and program
  - Communication follows this thinking
  - Often separate protocols for controller-controller communication
  - Asymmetric computing power
    - higher in controller than device
- **Peer to peer** tends to be implemented on a higher logical levels

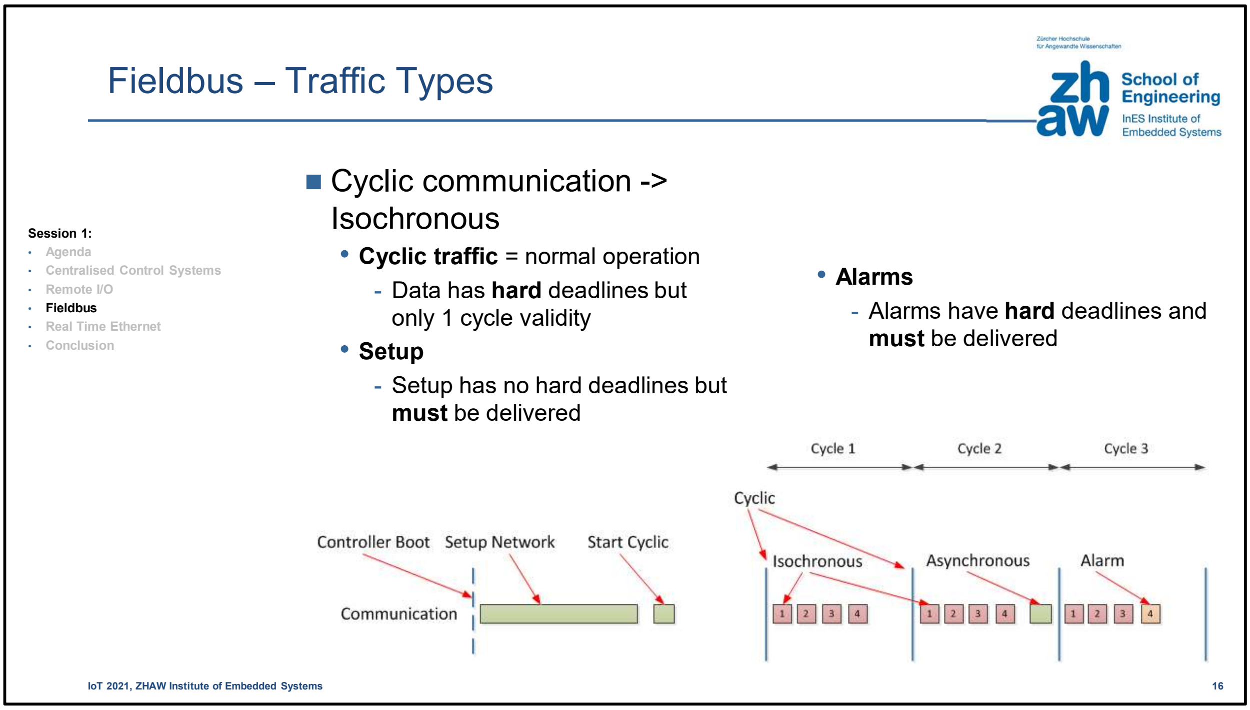
IoT 2021, ZHAW Institute of Embedded Systems

15

The application philosophy still plays an important part in how communication is built up. In the internet arena we talk about clients and servers which have an underlying peer-to-peer philosophy. We don’t really make assumptions about the “power” of a server. Control systems are largely what used to be known as master-slave, or in PC-English, **controller-device**/responder. The terminology may be unfortunate but describes the situation much better than alternatives as the master determines precisely what the remote I/O does.

Peer-to-peer applications exist, often in the context of modular systems where controller-device philosophy. Programming in this context is much more difficult as we move into “services” oriented programming which is not necessarily helpful in classic control applications.

We shall talk about controller-device applications in later slides.



Fieldbus – Traffic Types

Zürcher Hochschule für Angewandte Wissenschaften
zh aw School of Engineering
InES Institute of Embedded Systems

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

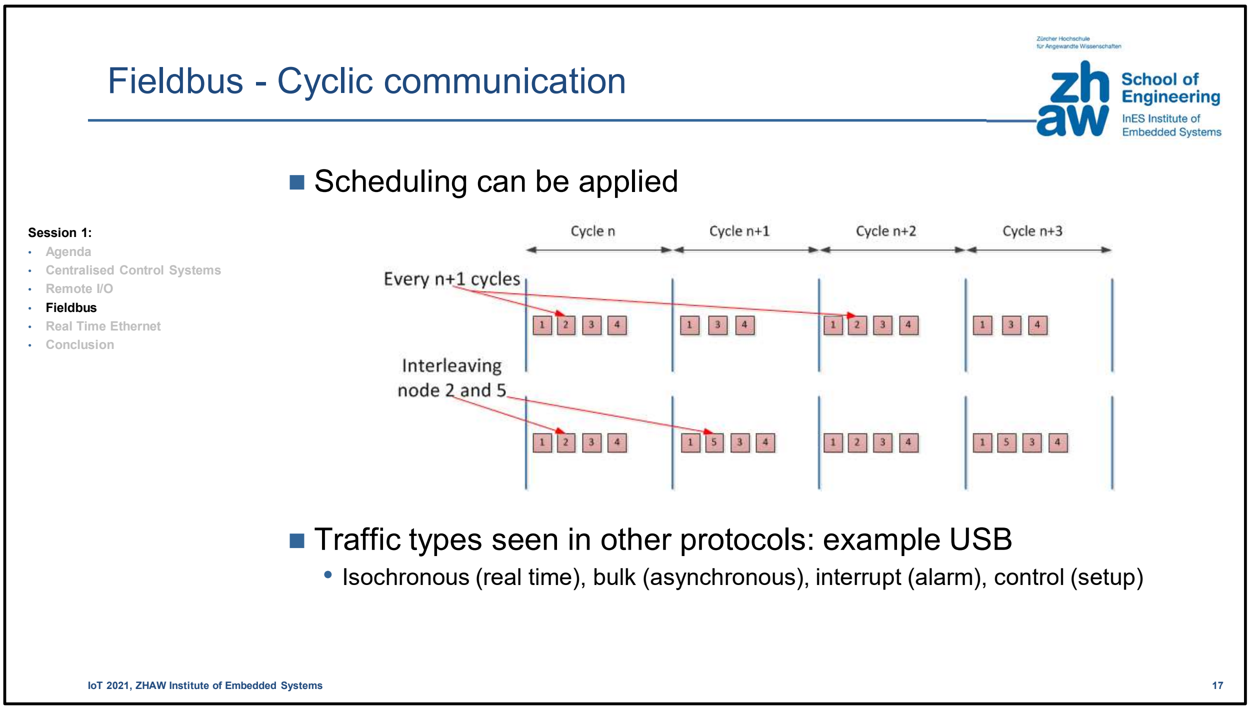
- Cyclic communication -> Isochronous
  - **Cyclic traffic** = normal operation
    - Data has **hard** deadlines but only 1 cycle validity
  - **Setup**
    - Setup has no hard deadlines but **must** be delivered
- **Alarms**
  - Alarms have **hard** deadlines and **must** be delivered

IoT 2021, ZHAW Institute of Embedded Systems 16

The cycle-time is the real-time aspect of communication. If the frame carrying the sensor value gets lost then there is no reason to ask for it again as this data only has one cycle validity (because in a controlled situation the measured values change, otherwise there wouldn’t be a need for control.) Remote devices have to be set up before a control cycle can start. The setup must be correct, so missing frames need to be detected and re-sent.

Things go wrong – sometimes we need to send an alarm, a higher level exception. Alarms are generally handled in a two-stage process. In the first stage the alarm is transmitted and the receiver acknowledges reception. When the receiver has actually processed the alarm, it then will generally send an acknowledgement that this has happened. This two stage process ensures that 1.) after confirmed reception, the transmitter stops sending the same alarm and 2.) after confirmed handling, the transmitter can clear his buffer of the alarm and continue with what he was doing.

We now have two phases of operation, the **setup phase** where the controller sets up the network, initialising itself and searching for devices and then the **cyclic** (**isochronous**) **operation** where devices are regularly polled or deliver data to the controller. In our cycle time we need to leave space for asynchronous communication or (re-)setup communication.



Fieldbus - Cyclic communication

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

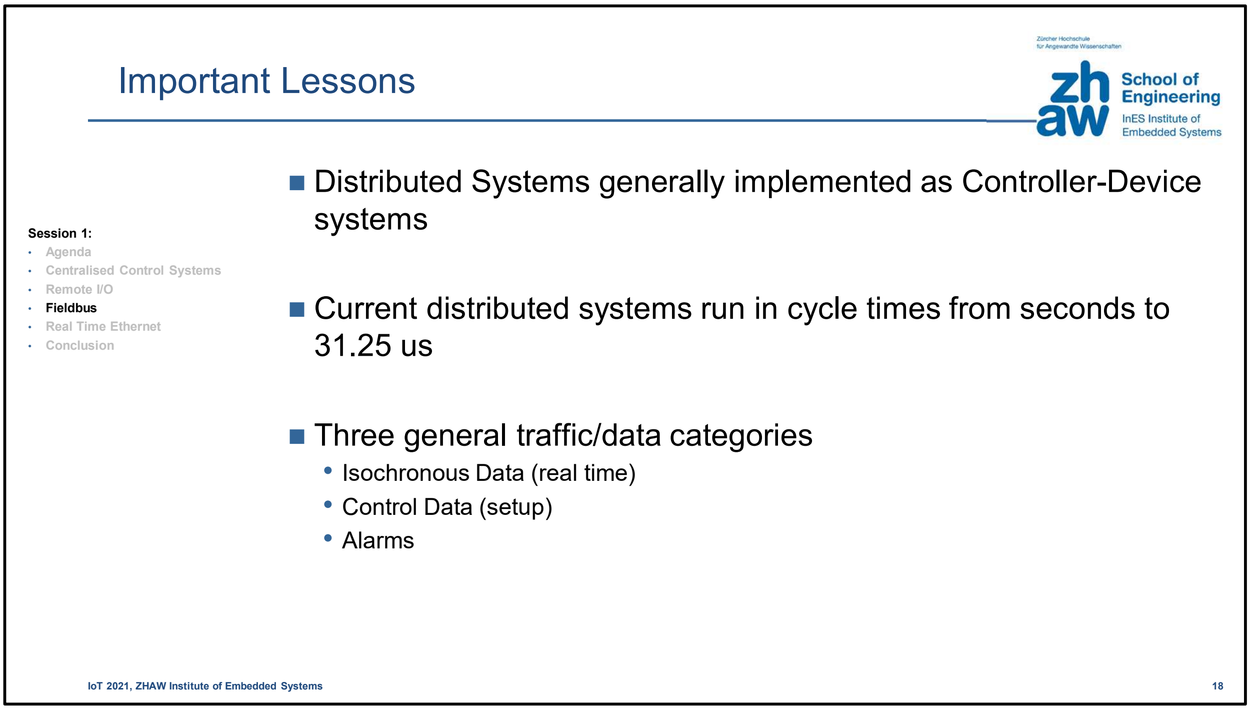
- Scheduling can be applied

- Traffic types seen in other protocols: example USB
  - Isochronous (real time), bulk (asynchronous), interrupt (alarm), control (setup)

Not all sensors in a network need to be polled or deliver values according to the cycle time. Because we divide up our time into regular intervals, the polling/delivery time needs to be a multiple of the cycle time. The system can accommodate more sensors by applying scheduling which generally either lengthens or shortens the isochronous bandwidth required within a cycle or use interleaving where the isochronous bandwidth remains more or less constant but sensors only deliver data at pre-agreed multiples of the cycle time.

So now the Data Link layer includes a bandwidth management system.

We see precisely this – and scheduling – in well-known consumer communication systems like USB.



Important Lessons

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

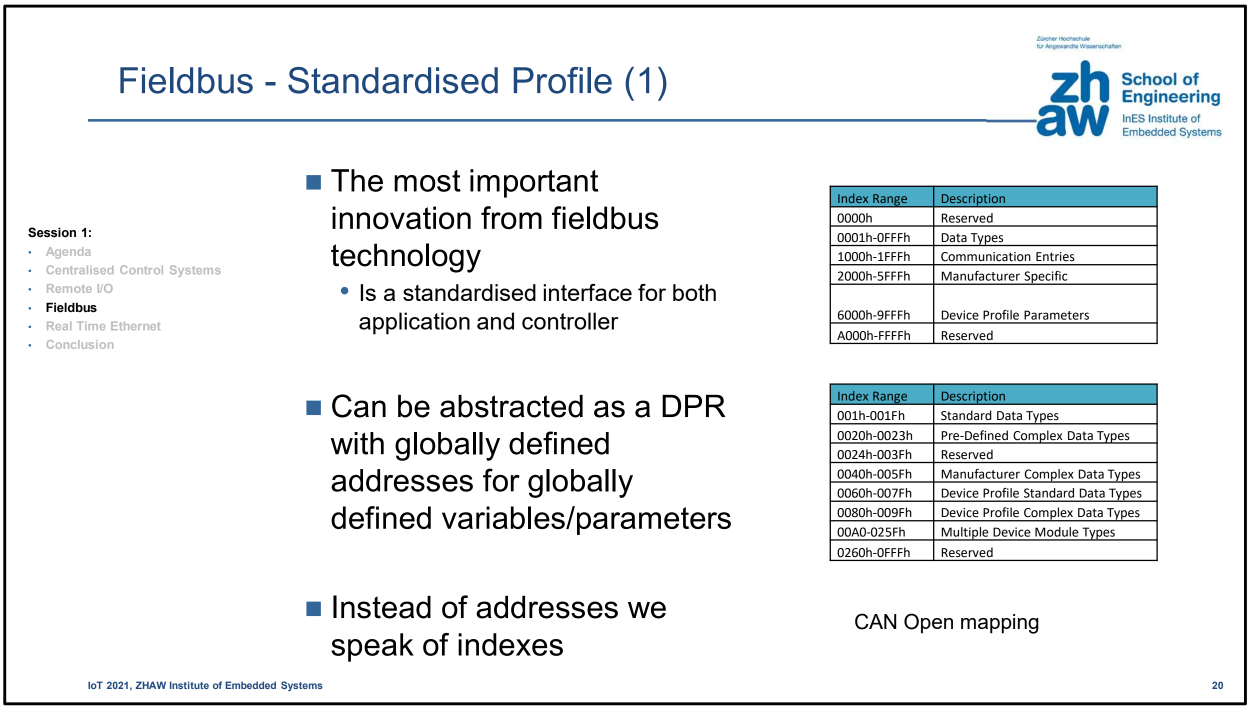
- Distributed Systems generally implemented as Controller-Device systems
- Current distributed systems run in cycle times from seconds to 31.25 us
- Three general traffic/data categories
  - Isochronous Data (real time)
  - Control Data (setup)
  - Alarms

Exercise

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

Application: Sleep control (monitoring ?). Wireless sensors placed across the mattress sensing small motions, like breathing and heart rate and large motions caused by tossing and turning during sleep, providing data available through an app on the smart phone.

- Lets presume this thing is in a hospital and lets make the following assumptions:
  - If the heart rate and breathing sensors don’t detect a heartbeat after 5 seconds and the patient isn’t tossing and turning then a warning is sent to the app. After 10 seconds and an alarm is sent
  - The doctor wants to see the relationship between tossing and turning and heartbeat every morning
  - As an option the nightwatch can elect to project the heartbeat onto a screen in his/her office
- Determine the transfer types for each case



Fieldbus - Standardised Profile (1)

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- The most important innovation from fieldbus technology
  - Is a standardised interface for both application and controller
- Can be abstracted as a DPR with globally defined addresses for globally defined variables/parameters
- Instead of addresses we speak of indexes

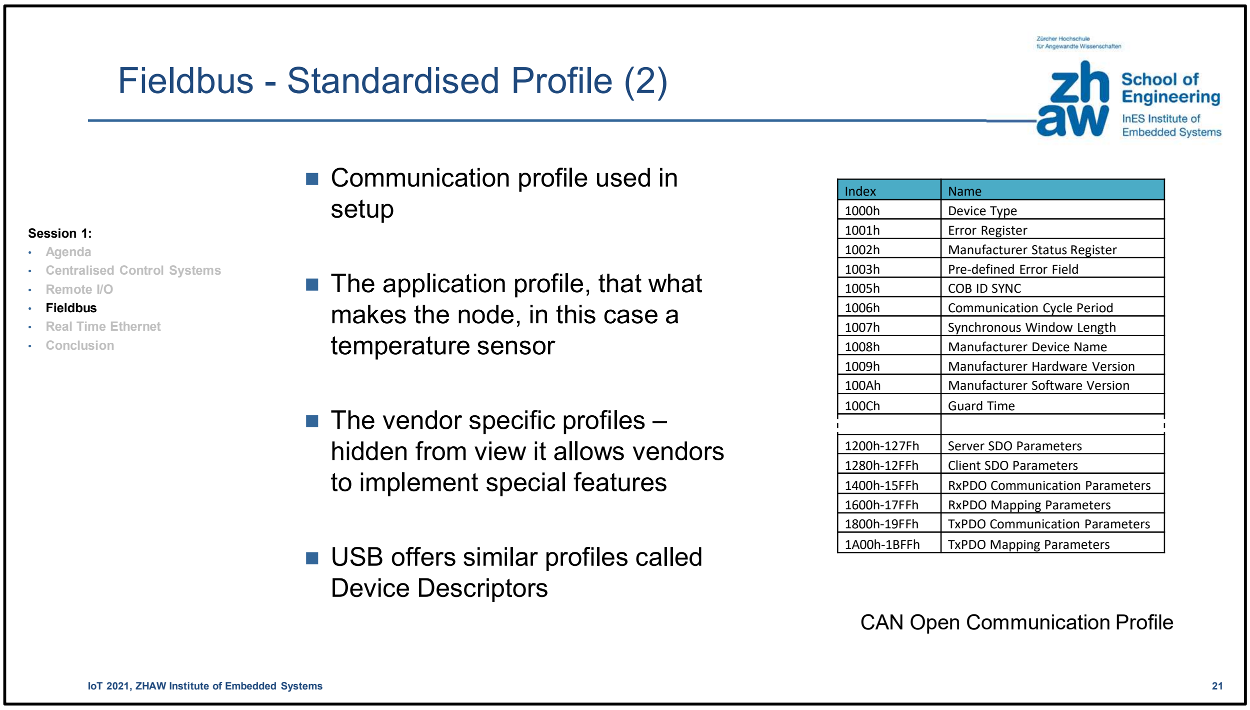
Index Range	Description
0000h	Reserved
0001h-0FFFh	Data Types
1000h-1FFFh	Communication Entries
2000h-5FFFh	Manufacturer Specific
6000h-9FFFh	Device Profile Parameters
A000h-FFFFh	Reserved

Index Range	Description
001h-001Fh	Standard Data Types
0020h-0023h	Pre-Defined Complex Data Types
0024h-003Fh	Reserved
0040h-005Fh	Manufacturer Complex Data Types
0060h-007Fh	Device Profile Standard Data Types
0080h-009Fh	Device Profile Complex Data Types
00A0-025Fh	Multiple Device Module Types
0260h-0FFFh	Reserved

CAN Open mapping

In my opinion the most important innovation to come out from the fieldbus era was the standardised **profile**. Remember we spoke about the dual ported RAM. We had an agreed, typically ad-hoc, memory map between two devices, largely on a per-variable basis. Some early communication systems also transferred memory blocks between devices, essentially mapping a memory space into that of the controller. The fundamental difference here is that the memory space is divided up into standardised **objects** (records would be a better term,) In order for this to work we need to standardise data types, we need to standardise the communication parameters, we need some area for manufacturer specific fields and then we have standardised device profiles, for instance temperature sensor, drive (motor controller) parameters, lift control and the like. This distribution we see for the CANopen (automation) protocol on the upper right. On the lower right we see the memory map for the definition of the data types. In CANopen we speak of **indexes** and **sub-indexes**. In PROFIBUS slots and sub-slots … In CANopen the collection of objects in a indexes/sub-index space is known as an **object dictionary.**

This idea of a standardised profile is also known in other protocols, for instance USB.



Fieldbus - Standardised Profile (2)

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

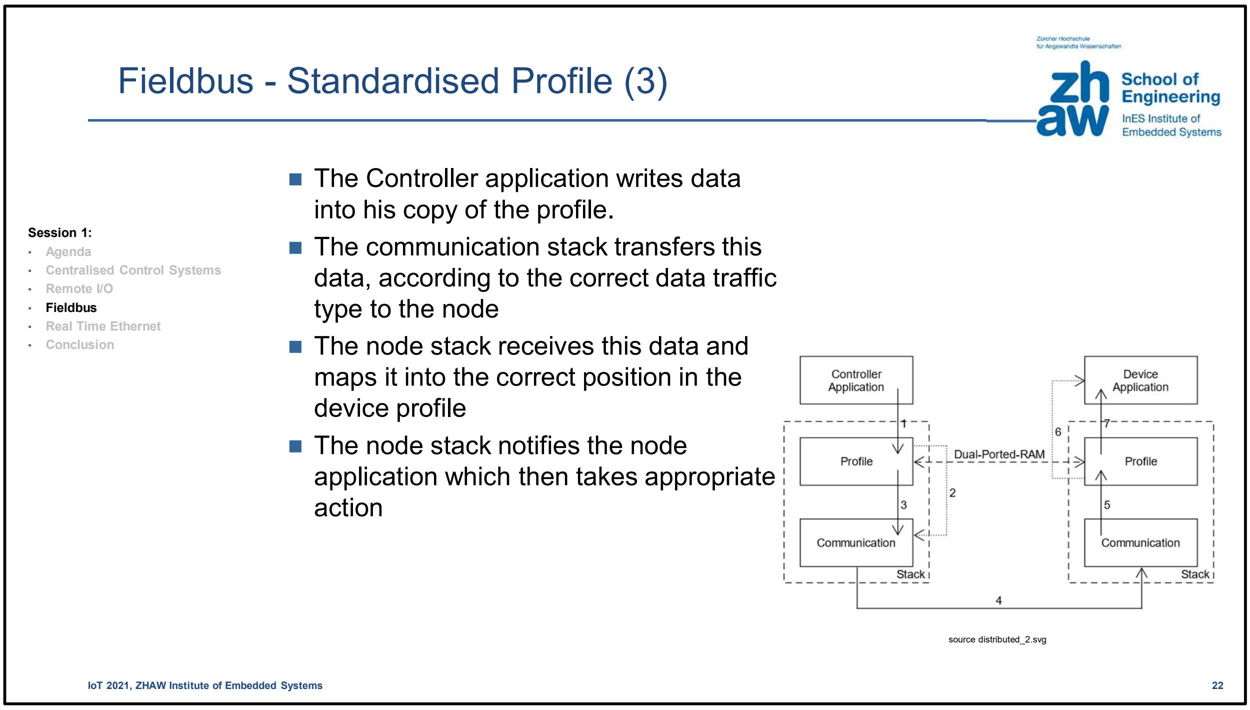
- Communication profile used in setup
- The application profile, that what makes the node, in this case a temperature sensor
- The vendor specific profiles – hidden from view it allows vendors to implement special features
- USB offers similar profiles called Device Descriptors

Index	Name
1000h	Device Type
1001h	Error Register
1002h	Manufacturer Status Register
1003h	Pre-defined Error Field
1005h	COB ID SYNC
1006h	Communication Cycle Period
1007h	Synchronous Window Length
1008h	Manufacturer Device Name
1009h	Manufacturer Hardware Version
100Ah	Manufacturer Software Version
100Ch	Guard Time
1200h-127Fh	Server SDO Parameters
1280h-12FFh	Client SDO Parameters
1400h-15FFh	RxPDO Communication Parameters
1600h-17FFh	RxPDO Mapping Parameters
1800h-19FFh	TxPDO Communication Parameters
1A00h-1BFFh	TxPDO Mapping Parameters

CAN Open Communication Profile

The communication parameters are set in this index space.

In CANopen we speak of Service Data Objects (SDO – setup data) and Process Data Objects (PDO – isochronous data.) For the isochronous (real-time) data we need fast data transfer from the ingress port to the application layer and the content of the isochronous frame is application specific, i.e. usually only known at runtime. We therefore have a mapping object which tells us which profile object (by index) is inserted in which part of the PDO frame. Obviously Rx and TX differ here.



Fieldbus - Standardised Profile (3)

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

- The Controller application writes data into his copy of the profile.
- The communication stack transfers this data, according to the correct data traffic type to the node
- The node stack receives this data and maps it into the correct position in the device profile
- The node stack notifies the node application which then takes appropriate action

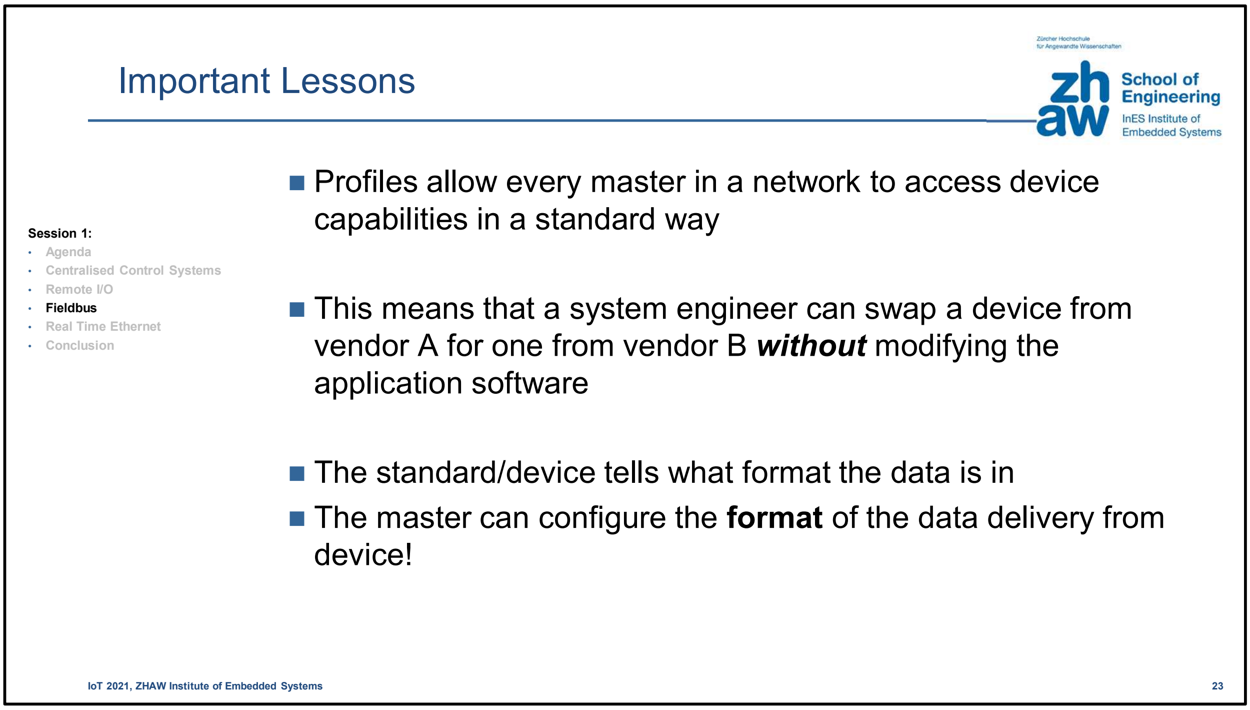
source distributed_2.svg

The mechanism functions as follows: The application writes, for instance, an object into a profile index/subindex in the scope of its local stack (1). This action inspires a call-back to the communication stack (2) which now decides to transfer a SDO (service data object, i.e. asynchronous.) It assembles the frame whose payload comprises this object (3). The stack transfers the object via the physical layer (4) to the stack on the receiver, in its own sweet time. The stack unpacks the frame and writes the object into the local object-dictionary (5) and informs the application (6).

In isochronous communication the stack is (generally) aware of when precisely the message is to be sent and therefore at a specific point in time assembles the PDO from the values in the object dictionary and transmits them (3,4). This allows the stack and the application to exists in a synchronous or asynchronous state. In an asynchronous state the application will write into the object in its own time (1) and this object will be transferred in either the current frame scheduled slot or the next slot.

In a modern stack on start-up, the application will determine the number and size of objects required and will allocate memory and initalise these values. For instance the application will only allocate and initialise the object “Manufacturer Device Name” if indeed there is one. A typical communication protocol will specify mandatory objects and optional objects.

Note that effectively we have described a two DPR system. The object director acts in both cases as a DPR with the application on one side and the respective stack on the other. The stacks communicate with each other according to the bandwidth management scheme of the communication protocol.

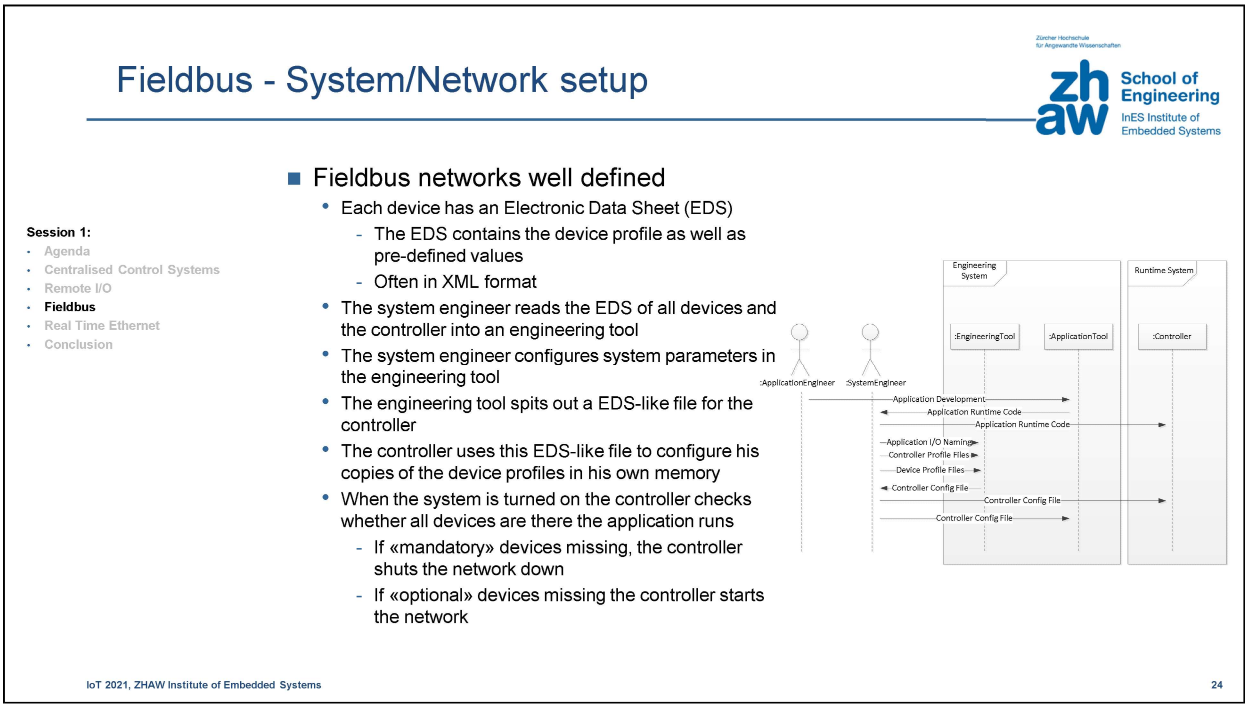


Important Lessons

Session 1:

- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- Profiles allow every master in a network to access device capabilities in a standard way
- This means that a system engineer can swap a device from vendor A for one from vendor B ***without*** modifying the application software
- The standard/device tells what format the data is in
- The master can configure the **format** of the data delivery from device!



Fieldbus - System/Network setup

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

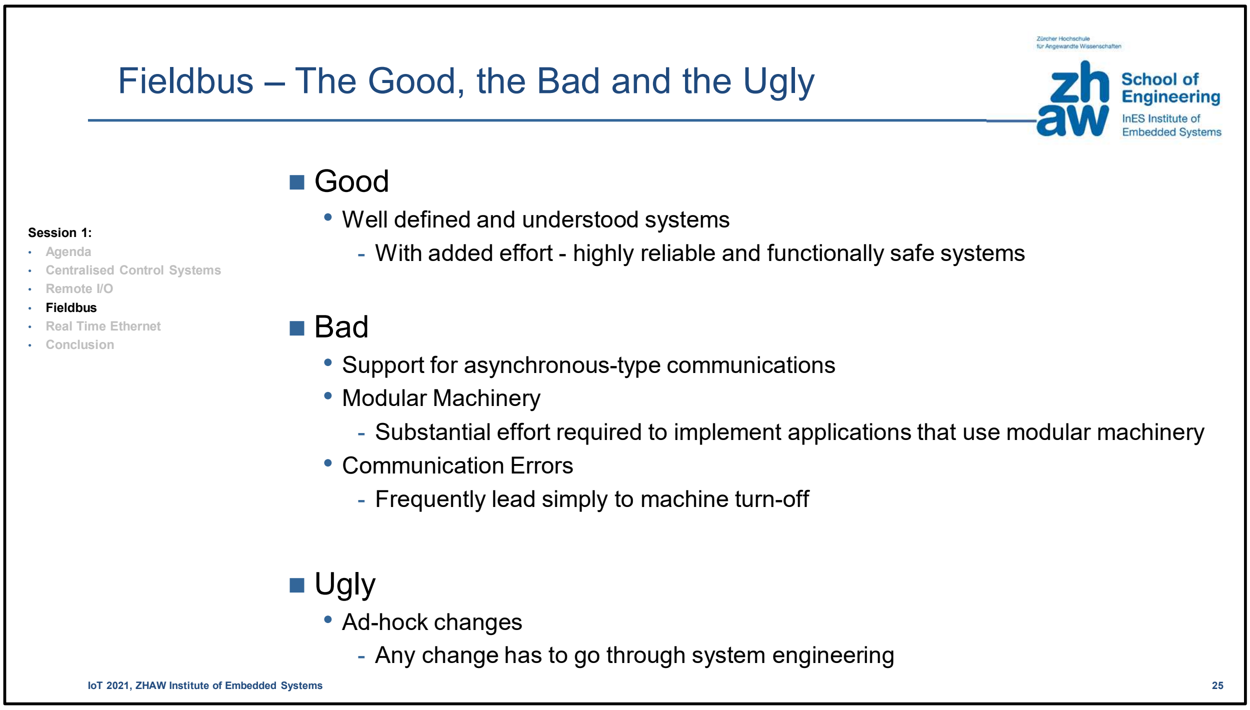
- Fieldbus networks well defined
  - Each device has an Electronic Data Sheet (EDS)
    - The EDS contains the device profile as well as pre-defined values
    - Often in XML format
  - The system engineer reads the EDS of all devices and the controller into an engineering tool
  - The system engineer configures system parameters in the engineering tool
  - The engineering tool spits out a EDS-like file for the controller
  - The controller uses this EDS-like file to configure his copies of the device profiles in his own memory
  - When the system is turned on the controller checks whether all devices are there the application runs
    - If «mandatory» devices missing, the controller shuts the network down
    - If «optional» devices missing the controller starts the network

The application of a distributed system is determined (usually) by the controller. On system-boot, the controller must first of all ensure that all **mandatory** devices are available and ascertain which **optional** devices are available, then the controller must initialise all the devices with the correct parameters and then the controller can start the application. If the mandatory devices are not available then the application may not start. So the controller must know what devices are supposed to be available, either as mandatory or optional devices. Then the controller must know what the parameters are for each device.

This systematic can be best understood By thinking of an application engineer programming an application using some namespace, not necessarily associated with networking components.

A System Engineer specifies the components. Each device supplies an **Electronic Data Sheet**. This sheet contains all necessary data defining the capabilities of the device. The system engineer uses an engineering tool to connect the devices together, schedule the bandwidth and work out IP addresses and the like. The engineering tool will read in the data sheet of each device and use it to produce some of the network parameters and/or check that the devices are suited for the required use. Once the network has been set up the tool will produce a controller-specific file which tells the controller what devices should be in the network, how to identify them and how to parametrize them. The tool will also match the application namespace to the network namespace. This file will generally be packed with the application on an SD card from which the controller (PLC or industry PC) can read what it has to do to boot the system.

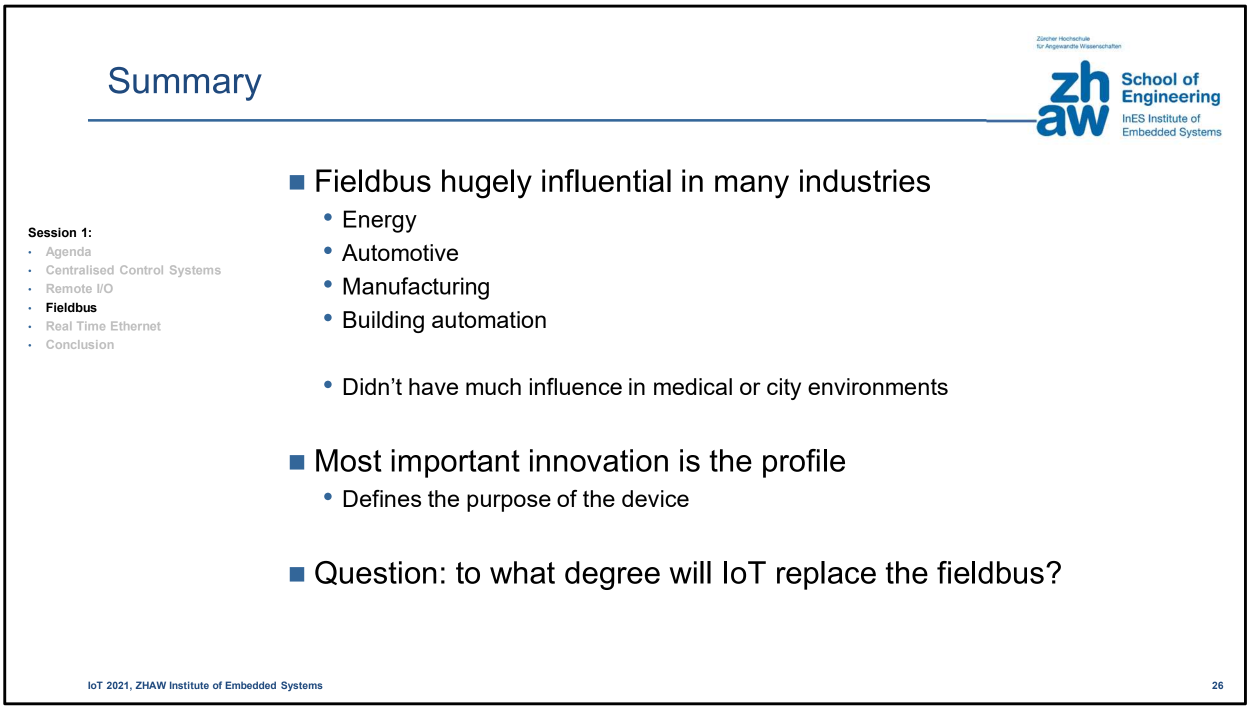
A similar method is used in the building automation industry



Fieldbus – The Good, the Bad and the Ugly

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

- Good
  - Well defined and understood systems
    - With added effort - highly reliable and functionally safe systems
- Bad
  - Support for asynchronous-type communications
  - Modular Machinery
    - Substantial effort required to implement applications that use modular machinery
  - Communication Errors
    - Frequently lead simply to machine turn-off
- Ugly
  - Ad-hock changes
    - Any change has to go through system engineering



Summary

Session 1:

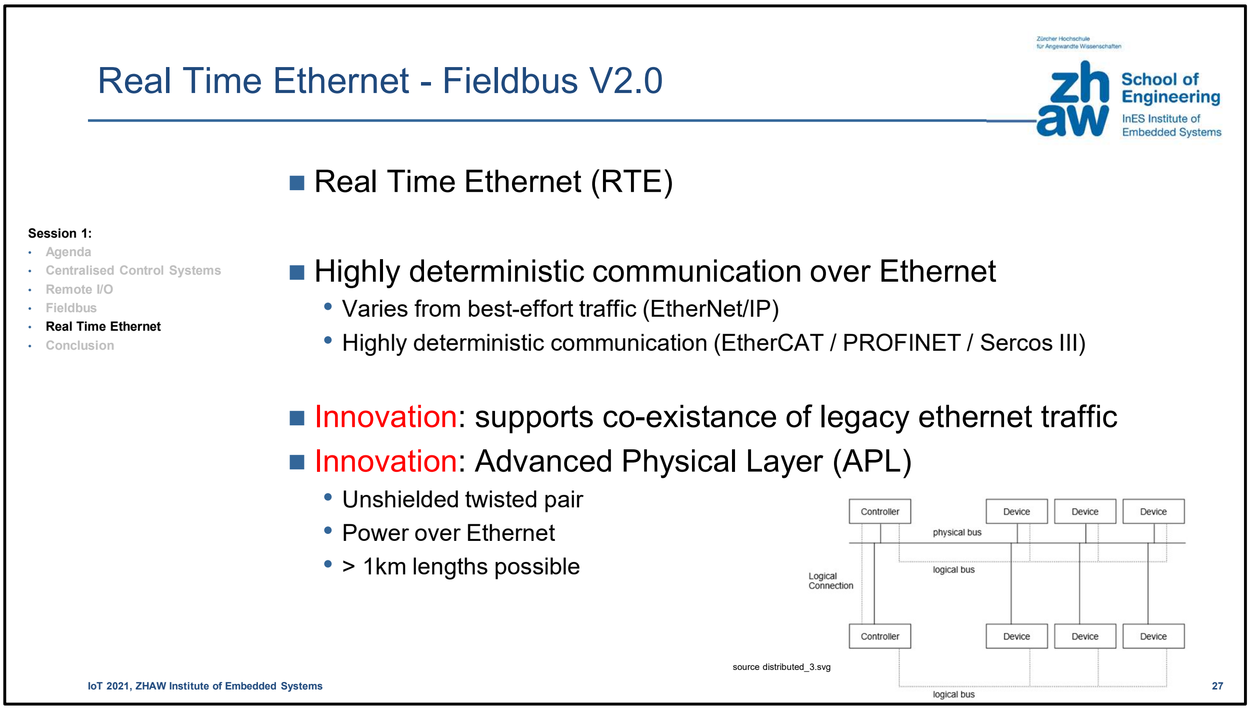
- Agenda
- Centralised Control Systems
- Remote I/O
- **Fieldbus**
- Real Time Ethernet
- Conclusion

- Fieldbus hugely influential in many industries
  - Energy
  - Automotive
  - Manufacturing
  - Building automation

  - Didn’t have much influence in medical or city environments

- Most important innovation is the profile
  - Defines the purpose of the device

- Question: to what degree will IoT replace the fieldbus?



Real Time Ethernet - Fieldbus V2.0

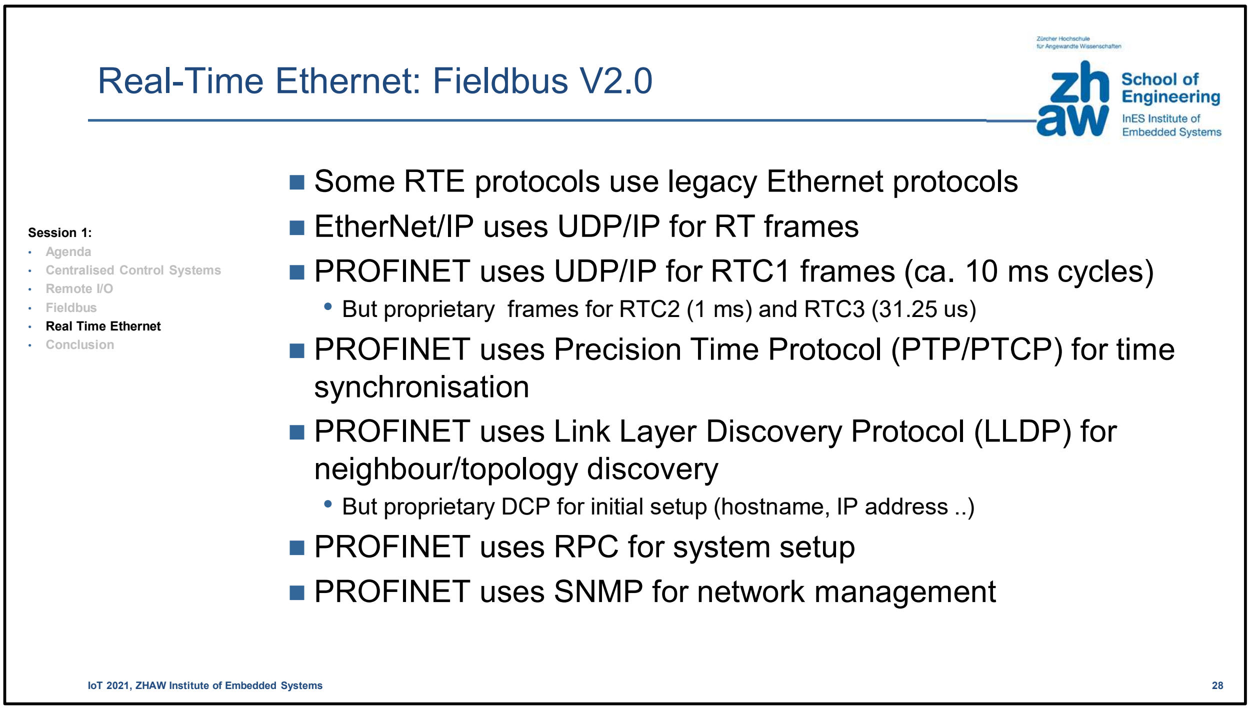
Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- Conclusion

- Real Time Ethernet (RTE)
- Highly deterministic communication over Ethernet
  - Varies from best-effort traffic (EtherNet/IP)
  - Highly deterministic communication (EtherCAT / PROFINET / Sercos III)
- Innovation: supports co-existance of legacy ethernet traffic
- Innovation: Advanced Physical Layer (APL)
  - Unshielded twisted pair
  - Power over Ethernet
  - > 1km lengths possible

source distributed_3.svg

The real time Ethernet field was notable for a number of things. In essence the protocol profiles were kept constant and the physical layer was changed and bandwidth management made more performant. With the advent of Time Sensitive Networking (TSN) the question of bandwidth management became (almost) redundant.

One important development of the RTE era, almost an afterthought, was the development of the **Advanced Physical Layer** (APL) out of the automotive industry towards the process industry where distances of over 1km and (low) power over the signalling lines are required.



Real-Time Ethernet: Fieldbus V2.0

Session 1:
- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- **Real Time Ethernet**
- Conclusion

- Some RTE protocols use legacy Ethernet protocols
- EtherNet/IP uses UDP/IP for RT frames
- PROFINET uses UDP/IP for RTC1 frames (ca. 10 ms cycles)
  - But proprietary frames for RTC2 (1 ms) and RTC3 (31.25 us)
- PROFINET uses Precision Time Protocol (PTP/PTCP) for time synchronisation
- PROFINET uses Link Layer Discovery Protocol (LLDP) for neighbour/topology discovery
  - But proprietary DCP for initial setup (hostname, IP address ..)
- PROFINET uses RPC for system setup
- PROFINET uses SNMP for network management

Conclusion

Session 1:

- Agenda
- Centralised Control Systems
- Remote I/O
- Fieldbus
- Real Time Ethernet
- **Conclusion**

- Dual Ported RAM as Communication Interface
  - Known position of variables
- Cost, technical and handling reasons for serial communication/distributed control
- Manufacturer independent communication protocols required
  - Define standardised physical layers
  - Define different traffic types
  - Define application profiles residing in DPR-like structures
  - Define a process for setting up distributed systems
- Ethernet in Distributed Embedded Systems
  - “Legacy” tools for embedded systems
  - Long distances due to APL physical layer.