Looking for People

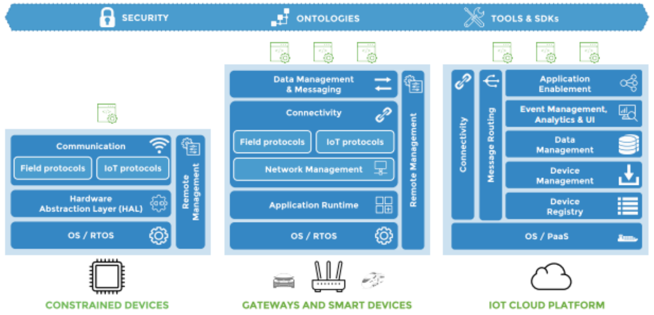
Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

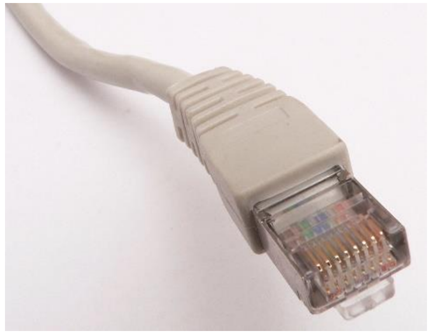
- Safe and Dependable Machine Learning
  - FPGA and SW
- Safe Vision
  - FPGA and SW
- Drone/Autonomous Vehicles
  - SW / HW

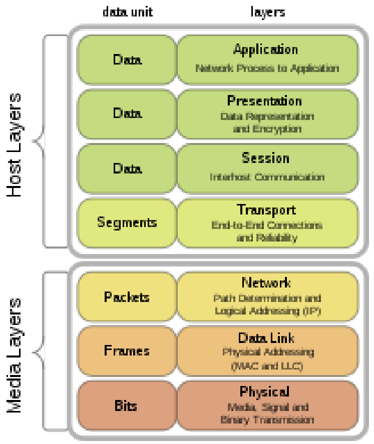
- Looking for
  - 1-2 full time HW/SW and/or 1-2 MSE SW

http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

IoT Lecture 7







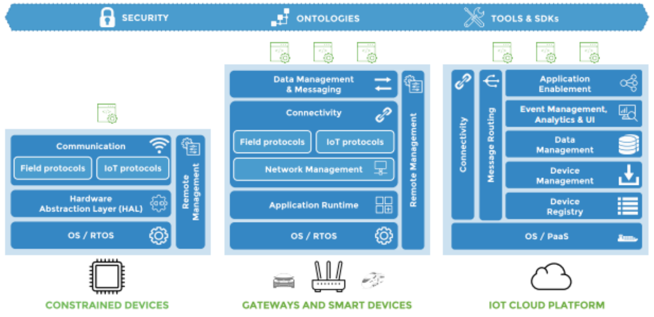
https://www.cisoplatform.com/profiles/blogs/classification-of-iot-devices
https://fr.wikipedia.org/wiki/Ethernet
https://en.wikipedia.org/wiki/Protocol_stack

Constrained Devices – Agenda

- RFC definition of constrained device
  - Benchmarking performance on constrained devices
- Ethernet Communication
  - Physical layers – Fast Ethernet and Advanced Physical Layer
  - HW footprint
  - Serial to parallel conversion
  - Data transfer to memory
  - Data buffering schemes
- Communication stacks
  - Zero copy stack
  - Linux IP stack
  - Socket interfaces

IoT Lecture 7 – Constrained Devices

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



https://www.cisoplatform.com/profiles/blogs/classification-of-iot-devices

Constrained Device Definitions (1)

Session 1: Constrained Devices
- Definitions
- Benchmarking
- Computational Power

Session 2: Ethernet Communication
- Physical Layer

Session 3: Communication Stacks
- Communication Stack

- RFC 7228 defines terminology of constrained devices
  - Constrained -> resource constrained
- Cost of devices silicon cost based on
  - Size of wafer, area of die, yield after wafer production
  - Number of pins, yield after packaging

	Name	data size (e.g., RAM)	code size (e.g., Flash)
Not IP Capable (?)	Class 0, C0	<< 10 KiB	<< 100 KiB
CoAP Capable	Class 1, C1	~ 10 KiB	~ 100 KiB
	Class 2, C2	~ 50 KiB	~ 250 KiB

Table 1: Classes of Constrained Devices (KiB = 1024 bytes)

Constrained Device Definitions (2)

- Also categorized devices by energy/power requirements

Name	Definition	SI Unit
Ps	Sustainable average power available for the device over the time it is functioning	W (Watt)
Et	Total electrical energy available before the energy source is exhausted	J (Joule)

Table 2: Quantities Relevant to Power and Energy

Name	Type of energy limitation	Example Power Source
E0	Event energy-limited	Event-based harvesting
E1	Period energy-limited	Battery that is periodically recharged or replaced
E2	Lifetime energy-limited	Non-replaceable primary battery
E9	No direct quantitative limitations to available energy	Mains-powered

Table 3: Classes of Energy Limitation

Name	Strategy	Ability to communicate
P0	Normally-off	Reattach when required
P1	Low-power	Appears connected, perhaps with high latency
P9	Always-on	Always connected

Table 4: Strategies of Using Power for Communication

Constrained Device Experience (1)

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering
InES Institute of Embedded Systems

Bare Metal

- Bit-Bang serial UART, 8085, 4MHz., bare metal, assembler,
  - 32k code EPROM (ca. 2k used), 128 byte RAM used
- CANopen devices, 20+ MHz., bare metal
  - 4-5k code -> 25 k -50 k code.
  - ~100-200 Bytes data -> 0-5k – 1k data
- Ethernet Powerlink, 75MHz. ARM, bare metal (no IP stack)
  - ~250k code, ~100k data
- General expectation that IP stack requires OS (multi-tasking)

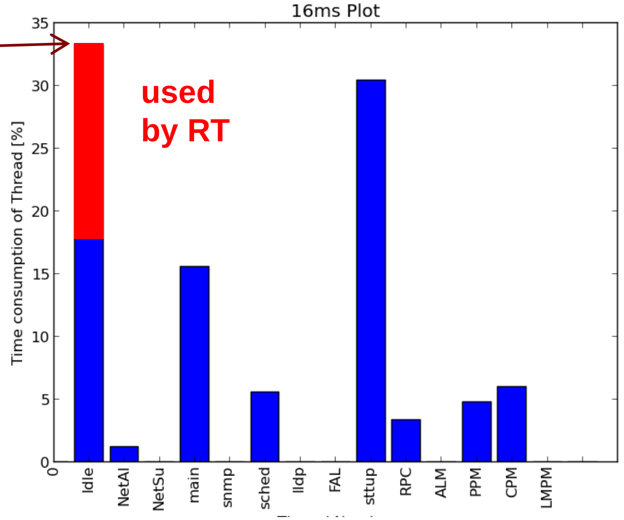
Constrained Device Experience (1)

- Typical embedded OS
  - eCOS – 80k kernel
  - BSD2000 TCP/IP stack – 350 kByte
- TCP/IP stack is fundamentally large – problem for constrained devices
  - Several lite IP-Stacks in the embedded market
  - (Coldfire 45 kBytes code as a webserver and 16 kBytes stack+heap)
  - Also HW IP, UDP stacks available
- Provisioning
  - What HW capabilities are required to
    - Provide required performance
    - “future-proof” the application

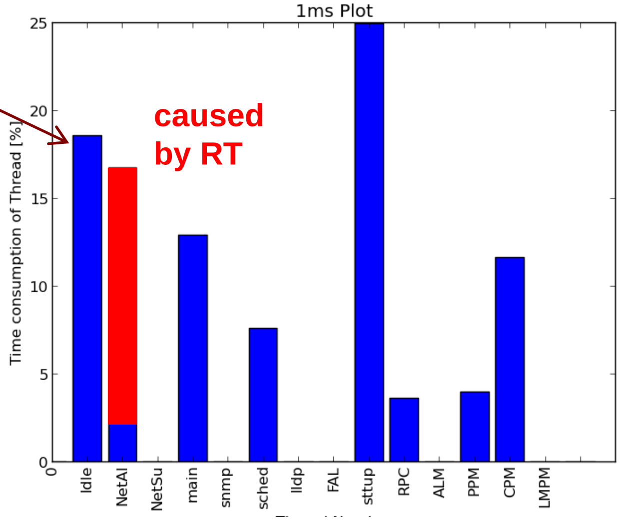
Provisioning and Performance (1)

- **Provisioning**
- ARM9 666MHz. eCOS - Altera
- 18% idle time @666MHz for 1ms?
  - WTH

@16 ms cycle time 34% idle



@1 ms cycle time 18% idle



Constrained Devices – Performance (2)

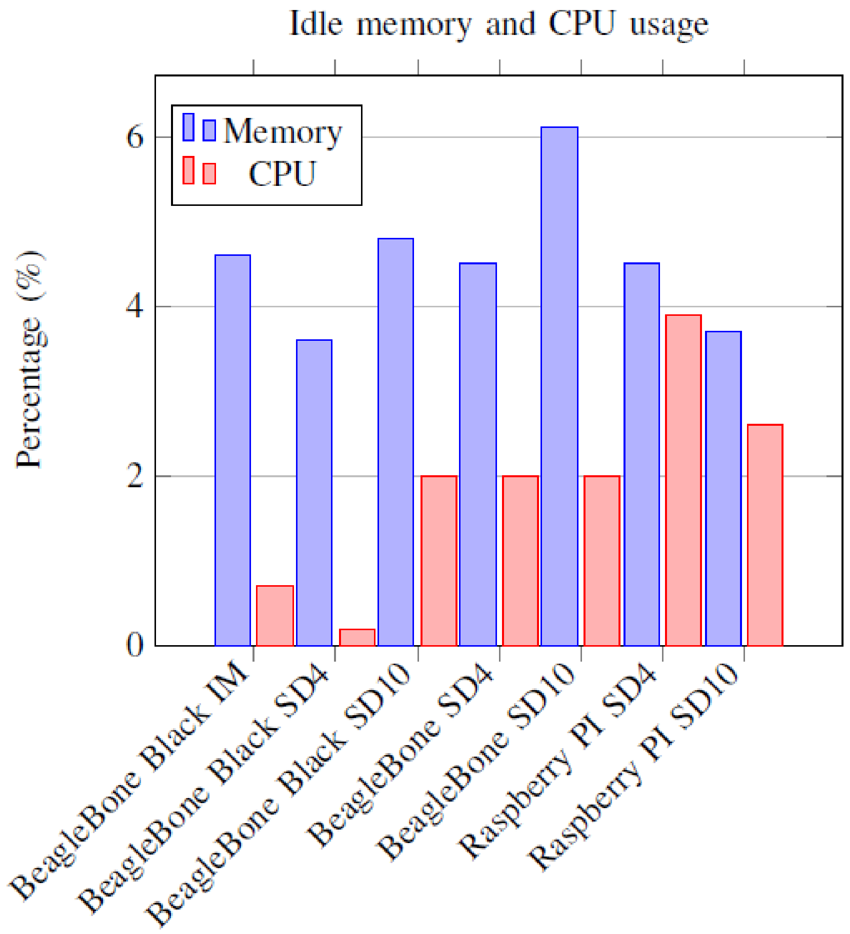
- Macro and Micro-Benchmarking
  - Determine suitability of platform for a task

- Micro-Benchmarks
  - Used to determine general platform capabilities
  - Lmbench can be used to test (Unix-like) platforms
  - Processor and Processes
  - Integer / floating point operations
  - Local communication latencies and bandwdiths
  - File and VM latencies
  - Memory Latencies

lmbench: Portable Tools for Performance Analysis Larry McVoy, Carl Staelin https://www.usenix.org/legacy/publications/library/proceedings/sd96/full_papers/mcvoy.pdf

Constrained Devices – Performance (3)

- For instance
  - Raspberry PI versus BeagleBone
  - Different system variations in idle mode, memory usage and CPU usage
  - SD4 and SD10 -> secure SD card for bootloading and filesystem



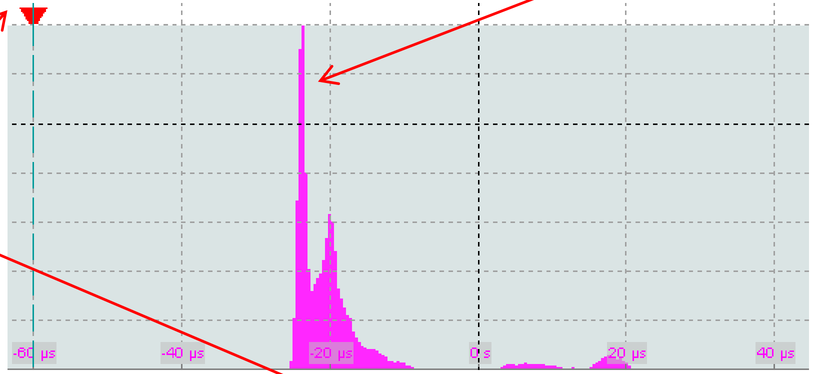
Benchmarking Internet of Things devices , C.P. Kruger y and G.P. Hancke

Constrained Devices – Performance (4)

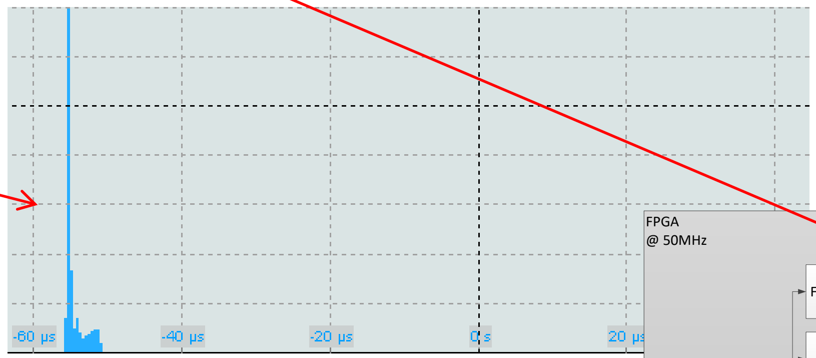
Macro-Benchmarking
- Application orientated benchmarking

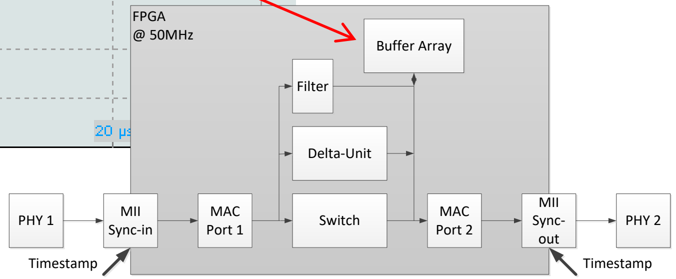
Trigger on HW signal generated by write to output – first line of DSR (Driver under eCOS)

Trigger on Interrupt – HW signal generated by input buffer in FPGA. 30’000 Frames



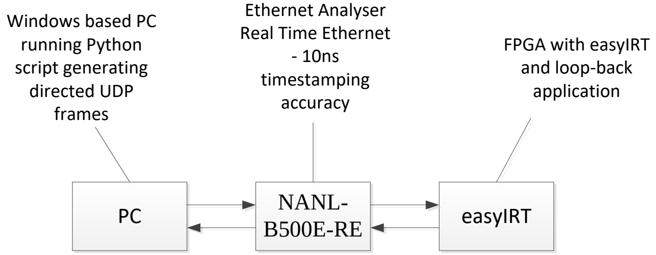
Trigger on HW signal generated by write to output – first line of ISR





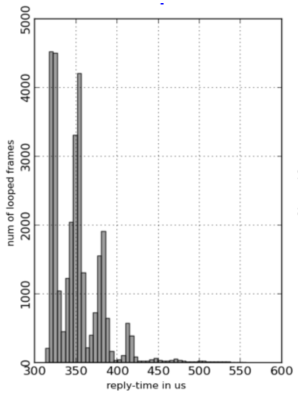
Constrained Devices – Performance (5)

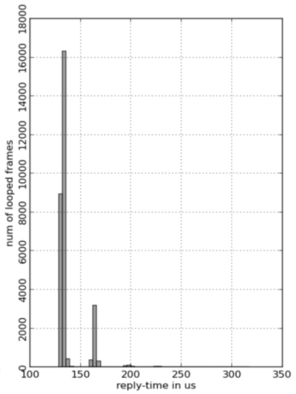
- Loop- Back Times, UDP frames, two platforms
  - Loopback done by driver (DSR)
  - Similar results for socket-level loopback

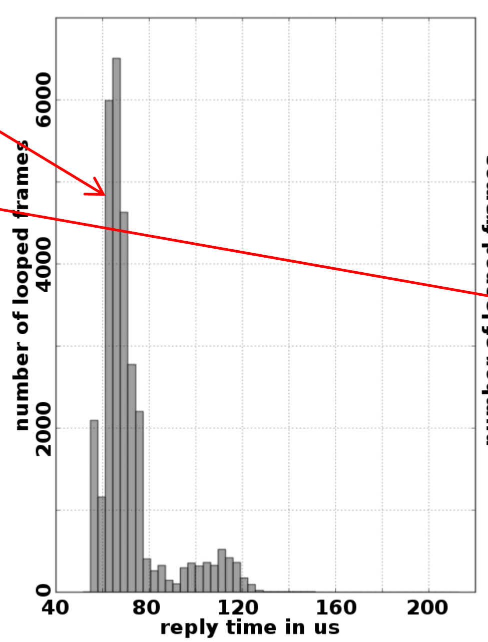


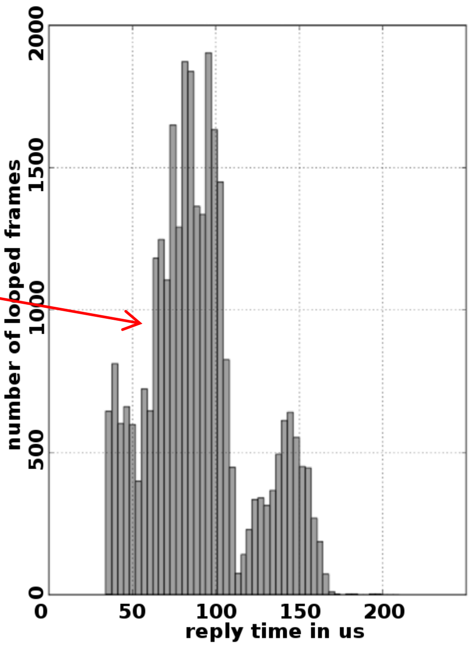
Xilinx A9 666 MHz. eCOS
30’000 frames

NIOS II 100MHz. eCOS.
30’000 frames







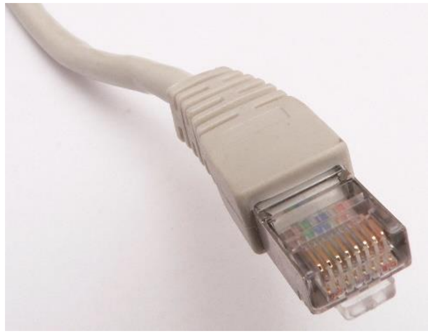


L. Itin and H. D Doran, "Optimising non-real time frame handling in real time ethernet nodes," *2015 IEEE World Conference on Factory Communication Systems (WFCS)*, 2015, pp. 1-8, doi: 10.1109/WFCS.2015.7160551.

IP benchmarking

- RFC 2544 – IP independent benchmarking
- RFC 5180 – IPv6 benchmarking specifica

IoT Lecture 7 – Ethernet Communication



https://fr.wikipedia.org/wiki/Ethernet

Physical layers (1)

- Already met CAN and RSxxx
  - Fieldbus physical layers
- Ethernet
  - IEEE 802.3 – 10 MBit/s – largely obsolete
  - IEEE 802.3u – Fast Ethernet – 100 MBit/s – in industry
  - IEEE 802.3ab – Gigabit Ethernet – 1 Gbit/s – in office
  - IEEE 802.3cg – Advanced Physical Layer – in industry
- In terms of constrained devices, Ethernet and IoT
  - Physical footprint
    - Physical size AND power requirements of the integrated circuit (IC) and support circuitry
  - Maximum rate of data consumption by microcontroller
    - Ability to shift frames into memory
    - Ability to process frame headers
    - Ability to process frame data

Learning Aim: The student will know the four main Ethernet types in IoT/industry use and their IEEE standards numbers

http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

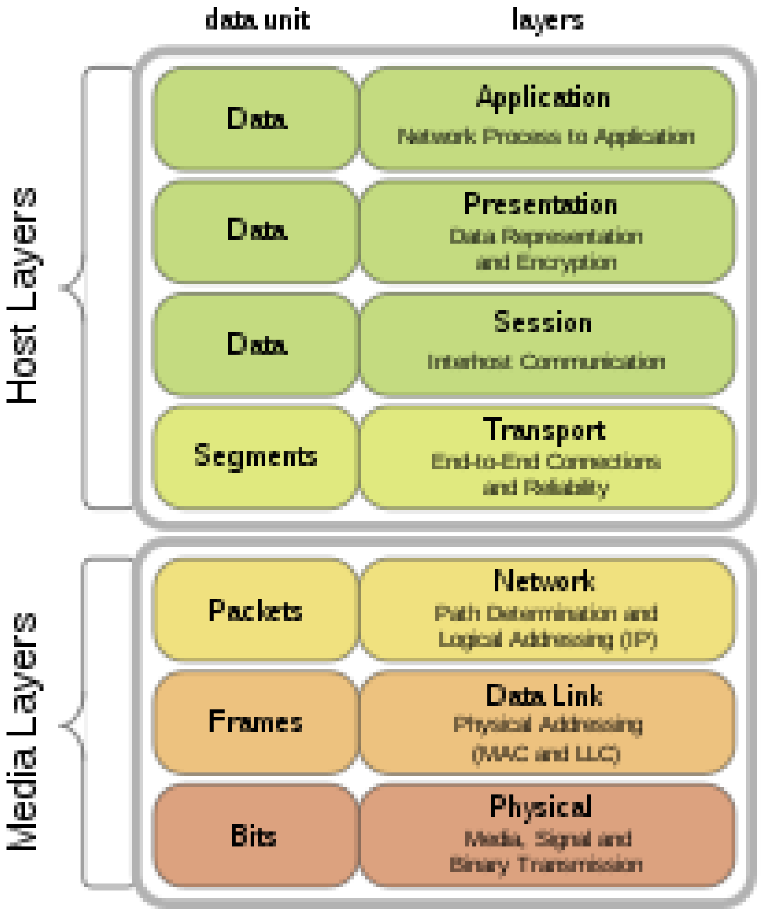
The Layer System

OSI layer model

- Used to understand communication systems
- Difficult to program in purity

Ethernet Relevance

- Application Layer -> Application hands over data that needs to be sent
- Presentation -> how the data is to be formatted for transfer to application
- Session -> the context of the data transfer
- Transport -> the connection
- Network -> the logical reachability
- Data Link -> the physical connection
- Physical layer -> assemble bits to bytes



Learning Aim: The student will understand the layering system

http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

Footprint - Connector

Session 1: Constrained Devices
- Definitions
- Benchmarking
- Computational Power

Session 2: Ethernet Communication
- **Physical Layer**

Session 3: Communication Stacks
- Communication Stack

Connector

M12 4-pin D-Coded
(M12 8-pin A-Coded)

Filter/Magnetics — PHY — MAC — DMA / uP / Memory

10GBit/s

Automotive connector

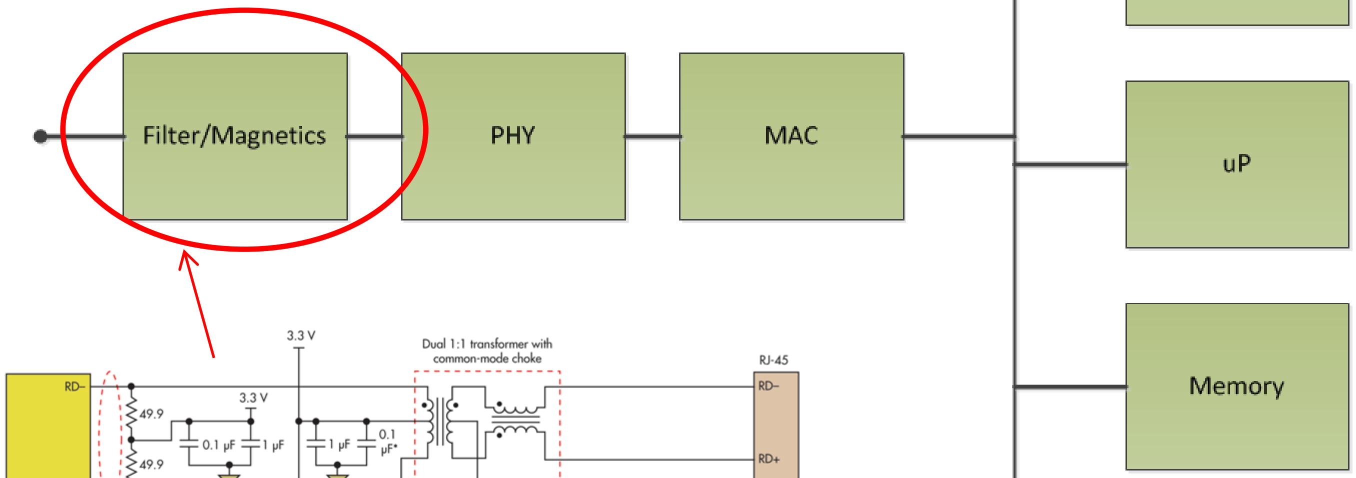
Learning Aim: The students will be able to describe the practical issues around Ethernet connector technologies

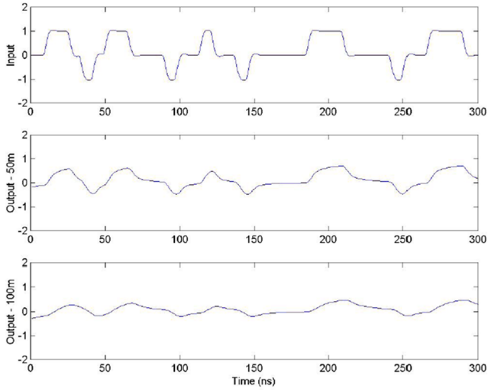
http://www.iebmedia.com/index.php?id=5873&parentid=63&themeid=255&showdetail=true
http://www.steinerpylenational.com/images/Pulse-Net-Industrial-Ethernet_large.jpg
https://encrypted-tbn1.gstatic.com/images?q=tbn:ANd9GcTiGTUf7MiR4D3CM7UYeeundXHwVpoXWGL6T4IYB0HdD-bd5nrEDQ

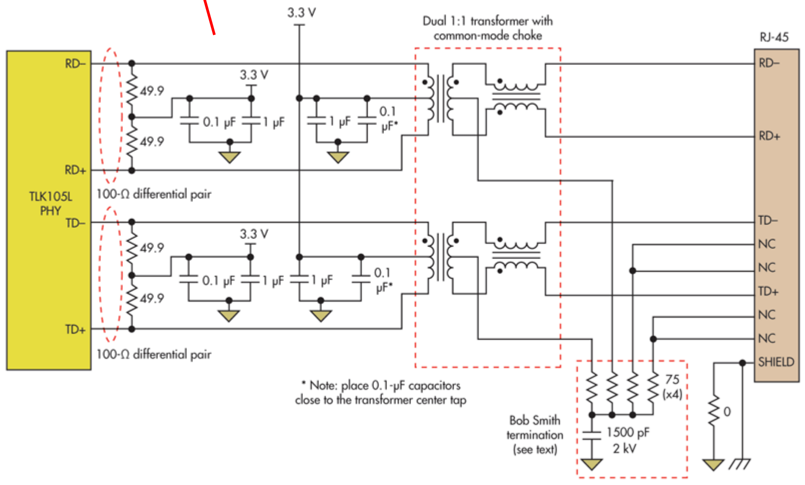
Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InES Institute of Embedded Systems

Footprint - Filter / Magnetics

- Differential signal input (no DC element)
- RC filters
- Transformers for
  - Galvanic isolation (no common ground)
  - Common mode rejection (robustness to noise)





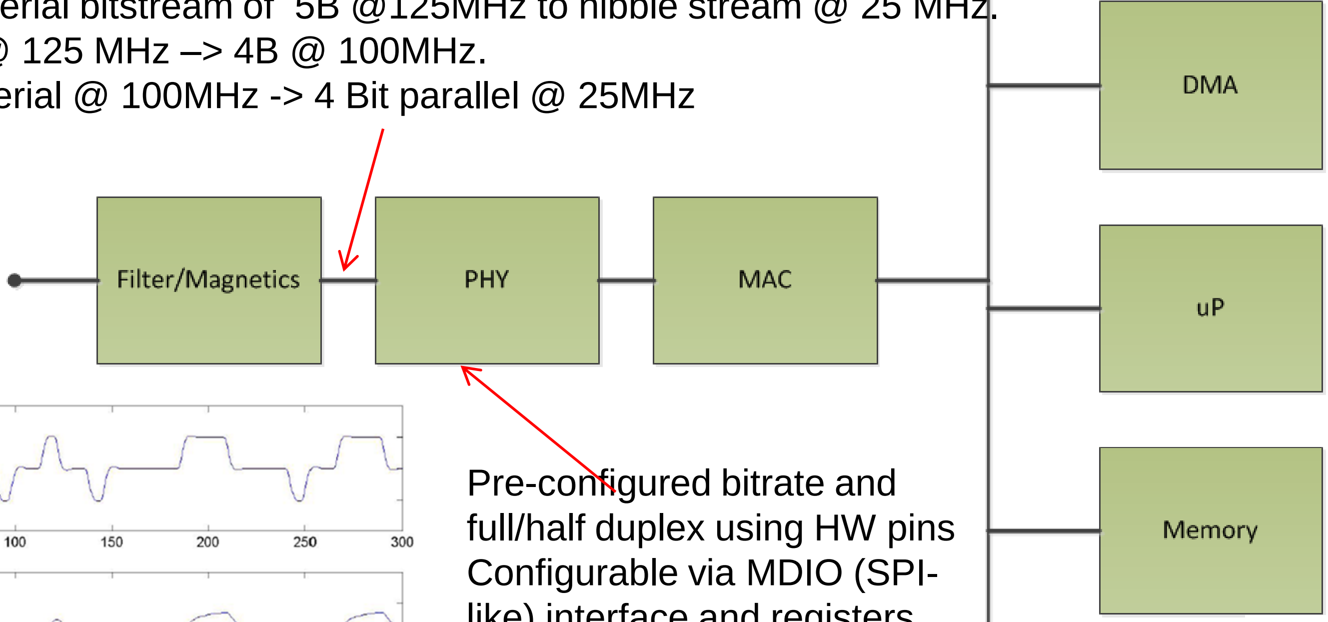


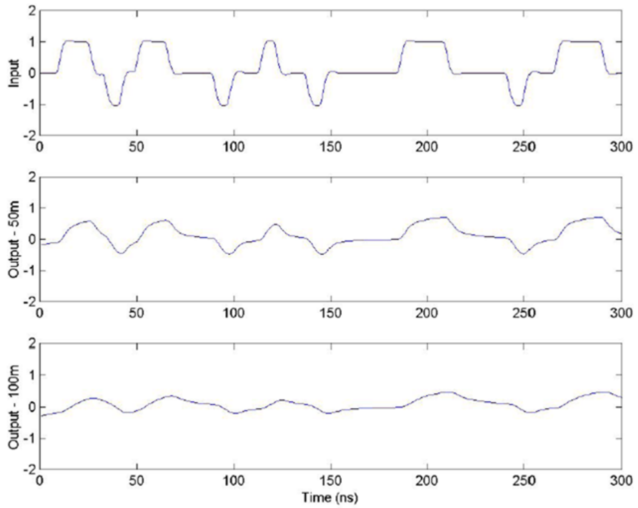
Learning Aim: The students will know the purpose of the filter/magnetics circuits

http://electronicdesign.com/site-files/electronicdesign.com/files/uploads/2014/04/0614_EnergyZarr_FO-big.gif

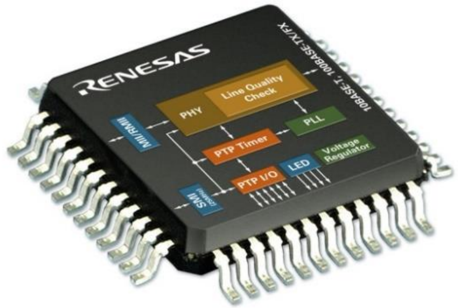
The PHY (1)

- PHYsical Layer Device
- Mixed signal device
- Converts serial bitstream of 5B @125MHz to nibble stream @ 25 MHz.
  - 5B @ 125 MHz –> 4B @ 100MHz.
  - 4B serial @ 100MHz -> 4 Bit parallel @ 25MHz





Pre-configured bitrate and full/half duplex using HW pins
Configurable via MDIO (SPI-like) interface and registers



Learning Aim: The student will know the purpose of the PHY device
Learning Aim: The student will be able to explain the frequencies from wire to MAC

http://donar.messe.de/exhibitor/hannovermesse/2015/E894782/gallery-868x0-301096.jpg

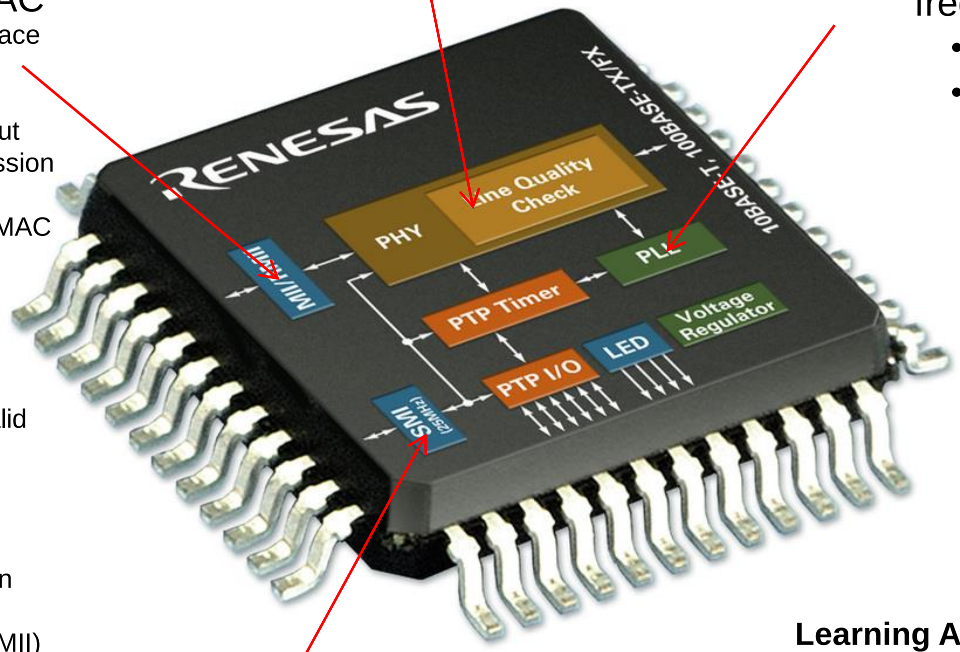
The PHY (2)

- Serialiser, scrambler, NRZI converter and MTL3 encoder
- Usually Digital Signal Processor (DSP) as sampling -> decoding element
- Due to DSP, modern PHY's can handle over 100m (f.i. 125 meters) cable lengths

Data interface to MAC
- Media Independent Interface (MII)
- TX_CLK (from PHY)
- TX_D [0 .. 3] -> 25MHz out
- TX_EN – enable transmission (MAC to PHY)
- TX_ER – transmit error (MAC to PHY – kill packet)

- RX-CLK (from PHY)
- RX_D [0 .. 3]
- RX_DV – receive data valid
- RX_ER – Receiver error
- CRS – Carrier sense
- COL – Collision Detect

- **Problem:** MII has high pin count -> Reduced Media Independent Interface (RMII)



- Phase-Locked Loop generates sampling frequency
  - Also generates heat
  - PHY usually as separate component

Control interface between PHY and MAC/uP

Learning Aim: The student will know the general components of the PHY device and its interfaces

http://donar.messe.de/exhibitor/hannovermesse/2015/E894782/gallery-868x0-301096.jpg

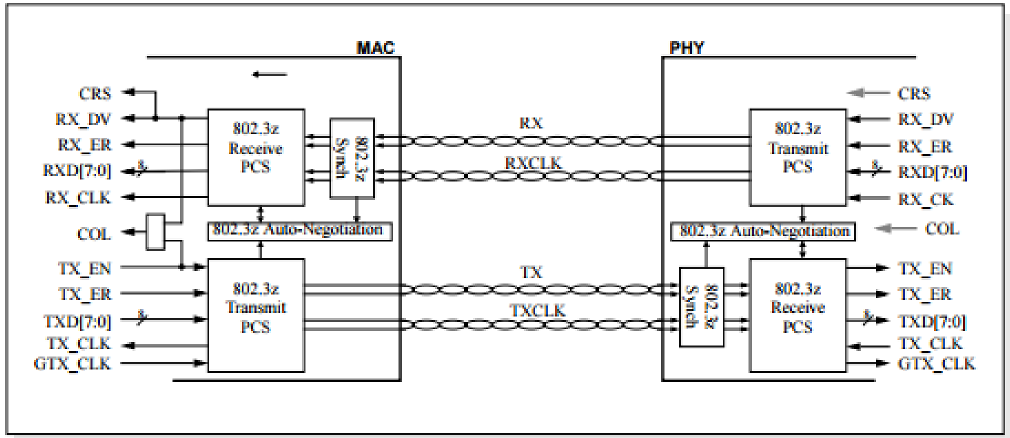
The PHY (5)

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

GMII

- GTX-CLK (from PHY)
- TX-CLK
- TX_D [0 .. 7] -> **25MHz out**
- TX_EN – enable transmission (MAC to PHY)
- TX_ER – transmit error (MAC to PHY)

- RX-CLK (from PHY)
- RX_D [0 .. 7]
- RX_DV – receive data valid
- RX_ER – Receiver error
- CRS – Carrier sense
- COL – Collision Detect



Takeaway: The physical footprint of the interface gets larger with increased transmission speed/bit-rate.

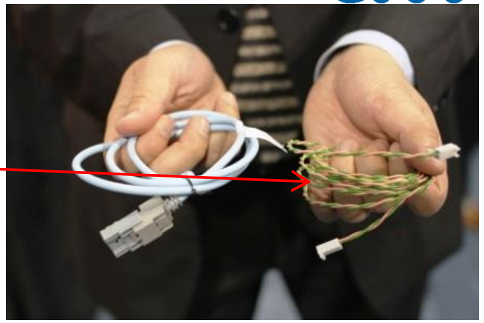
SGMII

- High speed serial interface differential DDR
- 625 MHz.
- GMII signals are encoded , serialised and transmitted with the data

http://www.angelfire.com/electronic2/sharads/protocols/MII_Protocols/sgmii.pdf

Advanced Physical Layer

- **Beginnings: Automotive BroadReach**
  - 2-wire unshielded twisted pair
  - 100Mbit/s full duplex
  - Max 15 meters
  - 4B/3B coding – aggressive filtering
- **Takeover: Process industry**
  - 10BASE-T1L IEEE 802.3cg-2019
  - 2-wire unshielded pair
  - Power-over-Ethernet (-like)
    - For EX-environments
  - 10 Mbit/s full duplex
  - Trunk and Spur architecture
    - Unpowered trunk segment max 1000 meters
      - (PHY – 1700 meters)
    - Spur segment max 200 m
  - PHY and (SPI) MAC-PHY components available
- **Future: Building Automation?**



Currently under IEEE 802.3 standardisation

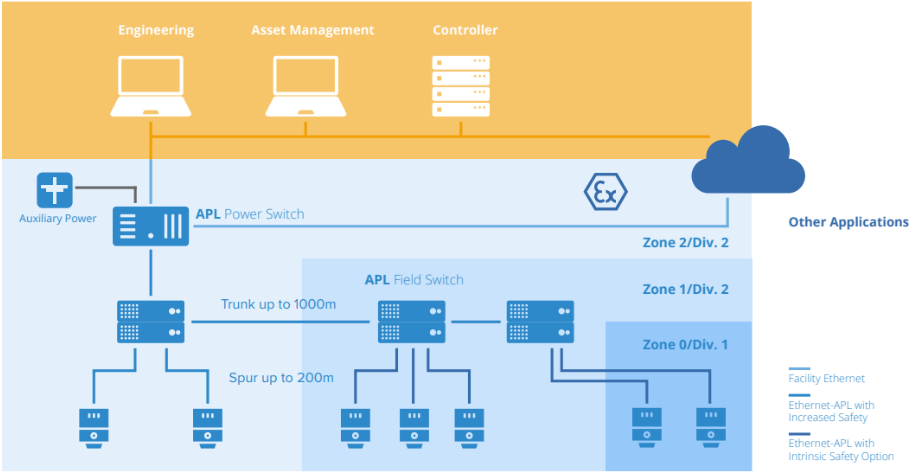


Figure 5: Example topology for long cable reach with up to 1000 m between switches on the trunk

https://encrypted-tbn1.gstatic.com/images?q=tbn:ANd9GcTiGTUf7MiR4D3CM7UYeeundXHwVpoXWGL6T4IYB0HdD-bd5nrEDQ
https://static1.squarespace.com/static/5c33ae96f407b4bfc5884978/t/60c7ef3a4807d14ca771825d/1623715647354/Ethernet-APL_Ethernet-To-The-Field_EN_FINAL_June-2021.pdf
Reference: https://www.ieee802.org/3/1TPCESG/public/BroadR_Reach_Automotive_Spec_V3.2.pdf

The MAC (2)

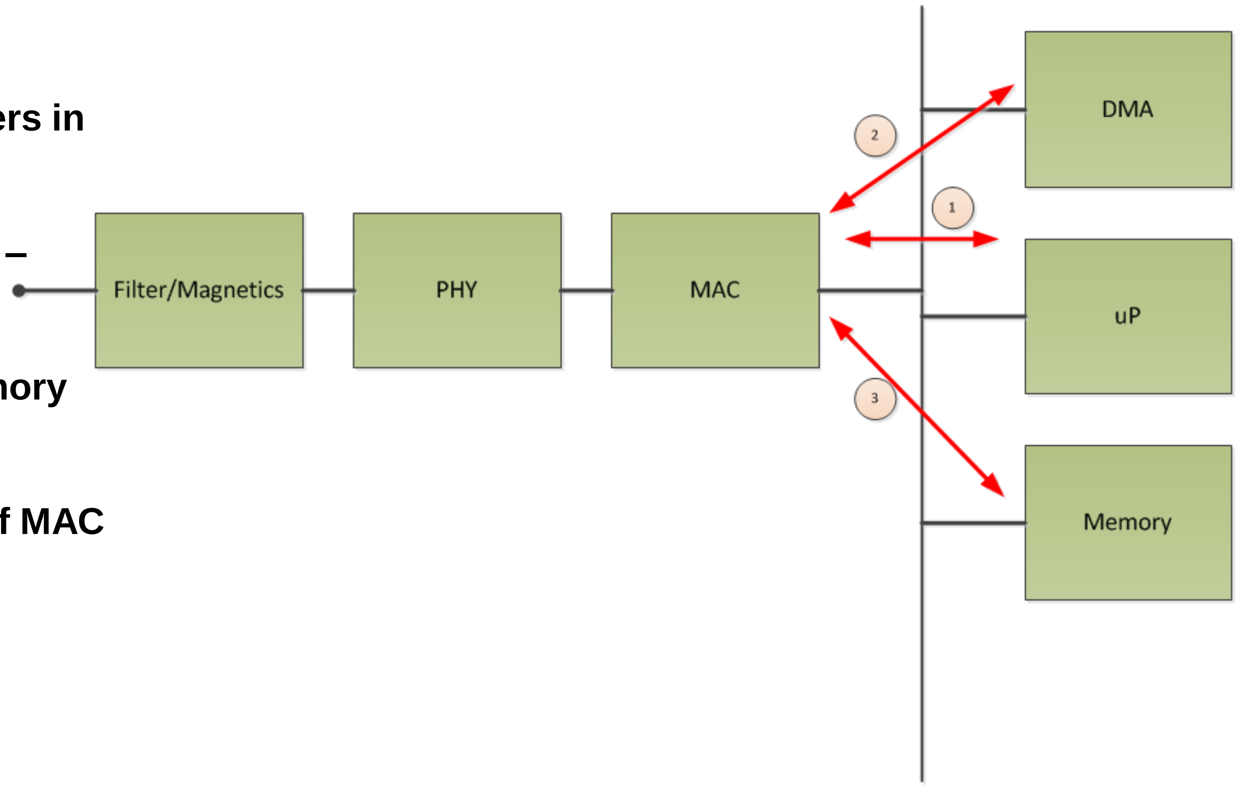
Three ways of getting frame to memory

1.) direct collect by CPU (if buffers in MAC)

2.) MAC initialises DMA transfer – both types of MAC

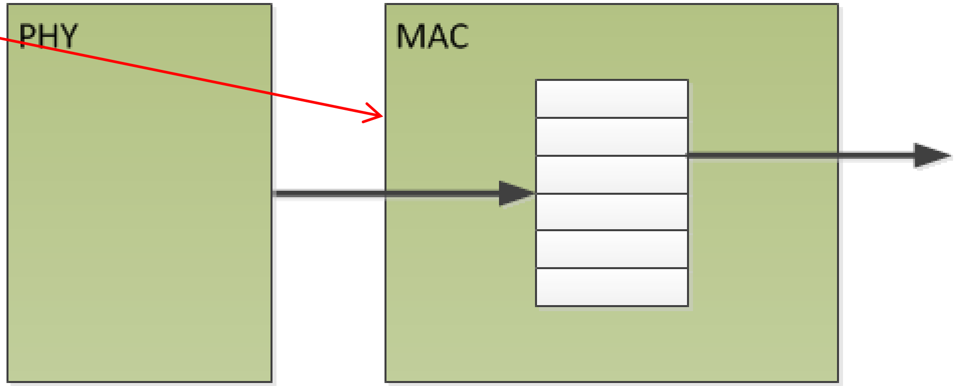
3.) MAC writes directly into memory

The entire system is affected in different ways by the decision of MAC architecture



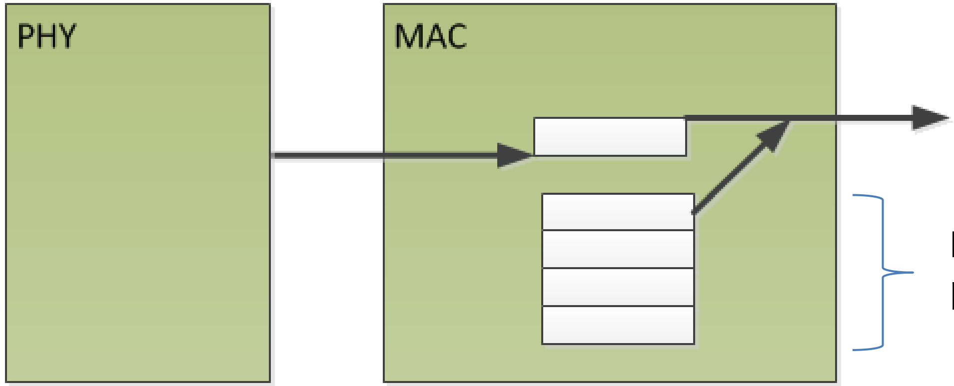
The MAC (3)

- **Media Access Controller – two general architectures**
  - **Integrated buffer**
  - **External buffer**



Single frame or multiple frame buffer
Disadvantage is that there is a delay and effort in getting the data from MAC to main memory
Need to collect data fast enough to avoid overflow

Word-sized buffer need sufficient system bus bandwidth to ensure no underflow



Buffer descriptors provide interface

Learning Aim: The student will be able to explain the difference between buffering architectures and their effect on data transfer in the MAC

The MAC (4)

Session 1: Constrained Devices
- Definitions
- Benchmarking
- Computational Power

Session 2: Ethernet Communication
- **Physical Layer**

Session 3: Communication Stacks
- Communication Stack

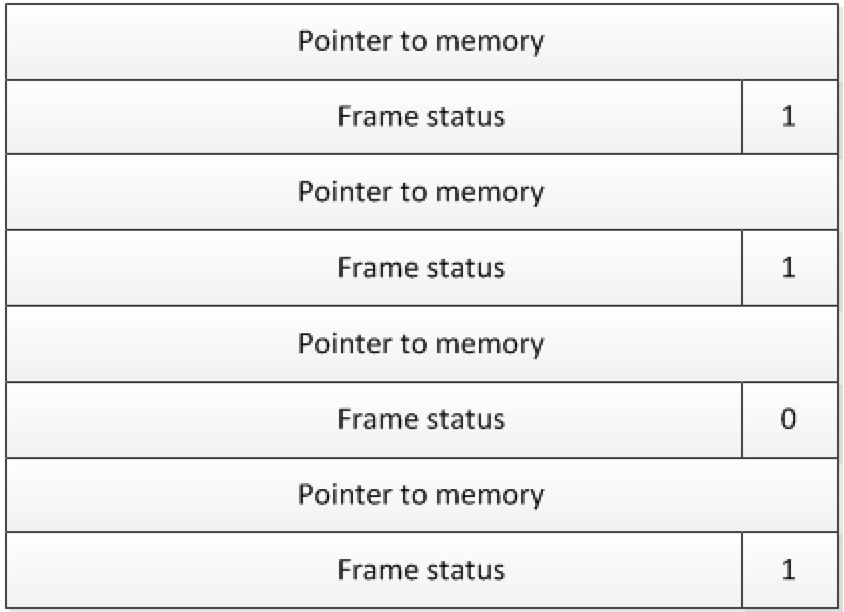
Buffer Descriptors

An array of descriptors with pointer to memory and a status word

Status word includes any errors, frame size and typically a «continue» bit

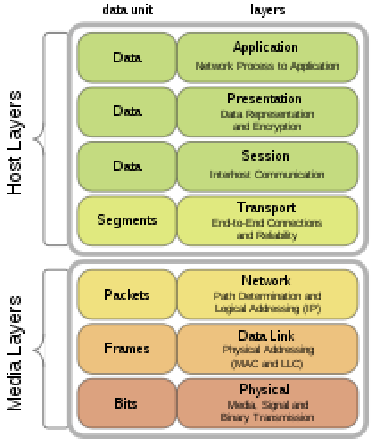
If, f.i. bit set then after frame tx or rx use the next buffer descriptor / next buffer descriptor is valid

Data can be transferred word for word f.i. using DMA



Learning Aim: The student will be able to program a buffer descriptior

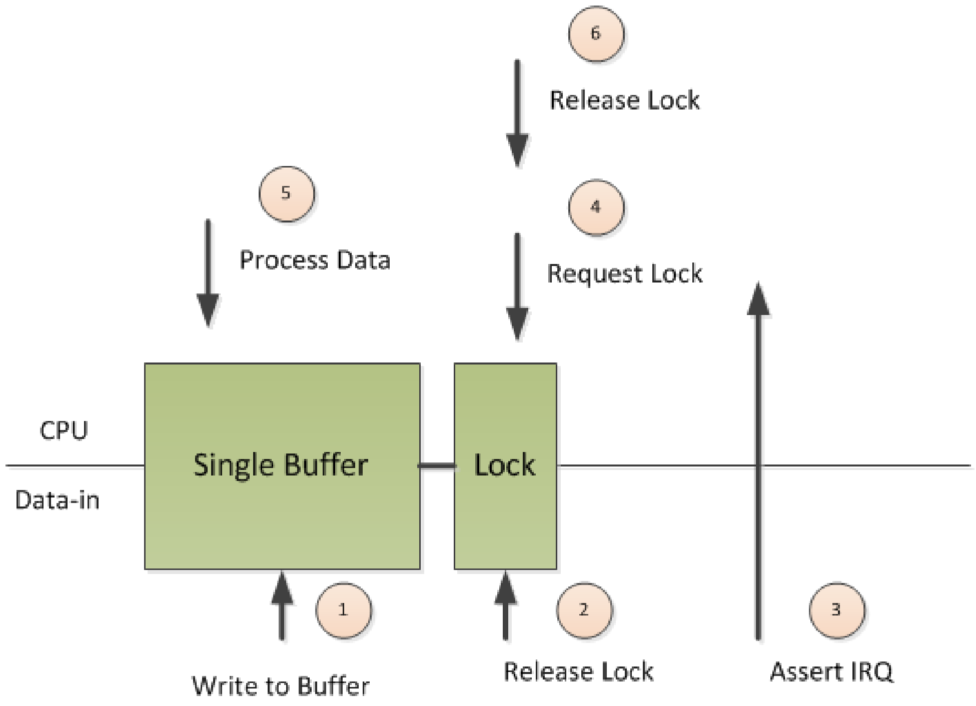
IoT Lecture 7 – Communication Stacks



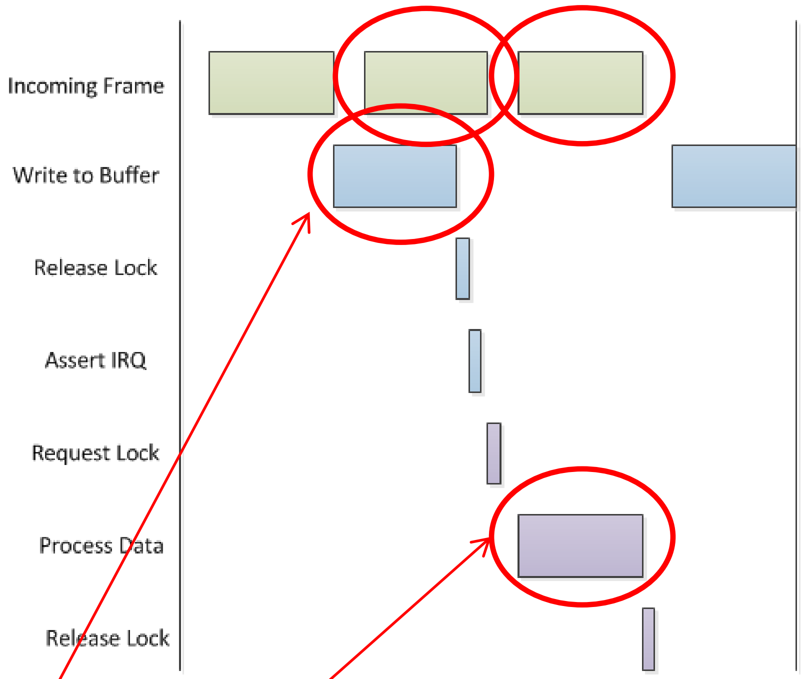
https://en.wikipedia.org/wiki/Protocol_stack

Data Buffering Memory (1)

Simple HW and Software Systems Buffering: Single Buffer



One of these gets dropped



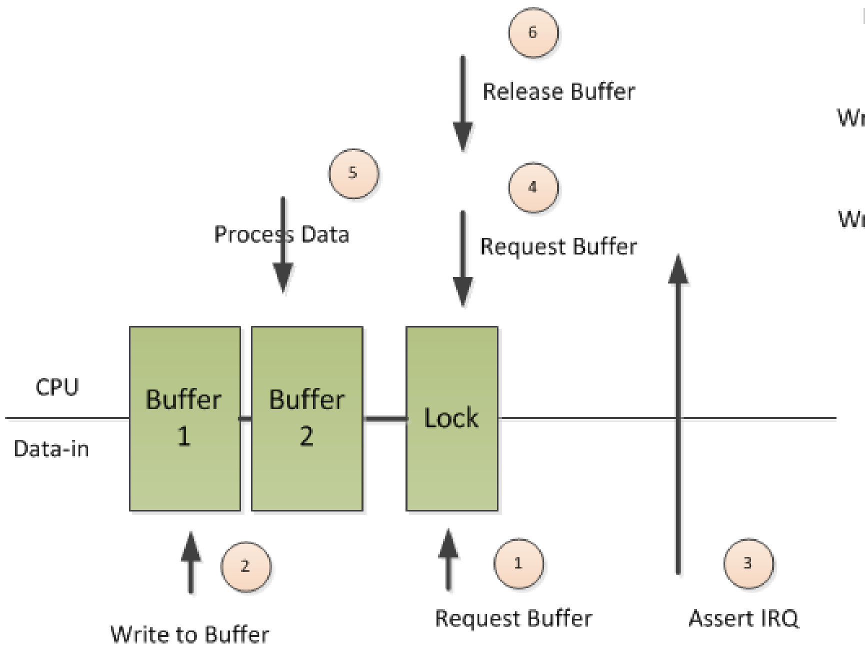
Example Schedule Single Buffer

Processor synchronous with communication -> Critical Activities

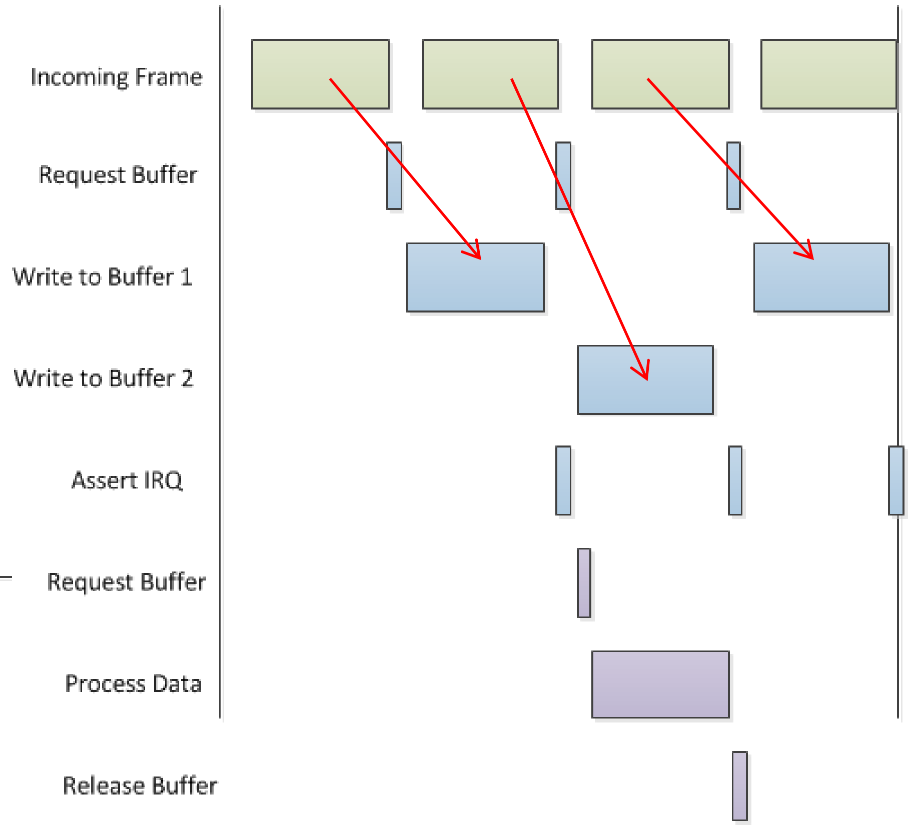
Learning Aim: The student will be able to explain the three data buffering methods, single, double and triple

Data Buffering Memory (2)

Simple HW and Software Systems
Buffering: Double Buffer



Will get into trouble later than sooner



Example Schedule Double Buffer

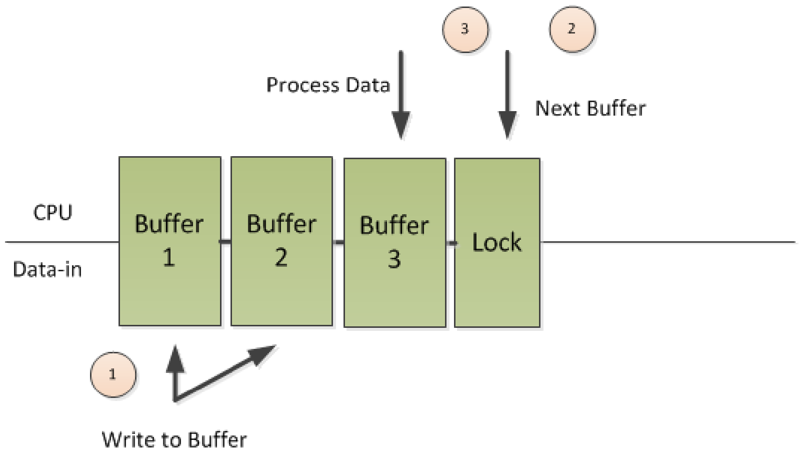
Data Buffering Memory (3)

Session 1: Constrained Devices
- Definitions
- Benchmarking
- Computational Power

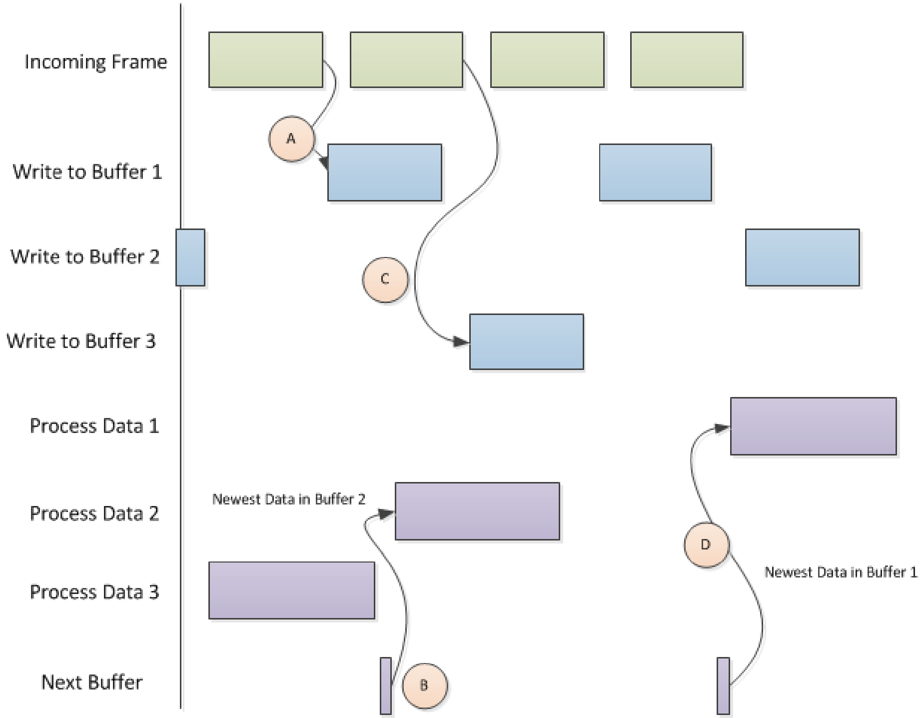
Session 2: Ethernet Communication
- Physical Layer

Session 3: Communication Stacks
- **Communication Stack**

Simple HW and Software Systems Buffering: Triple Buffer



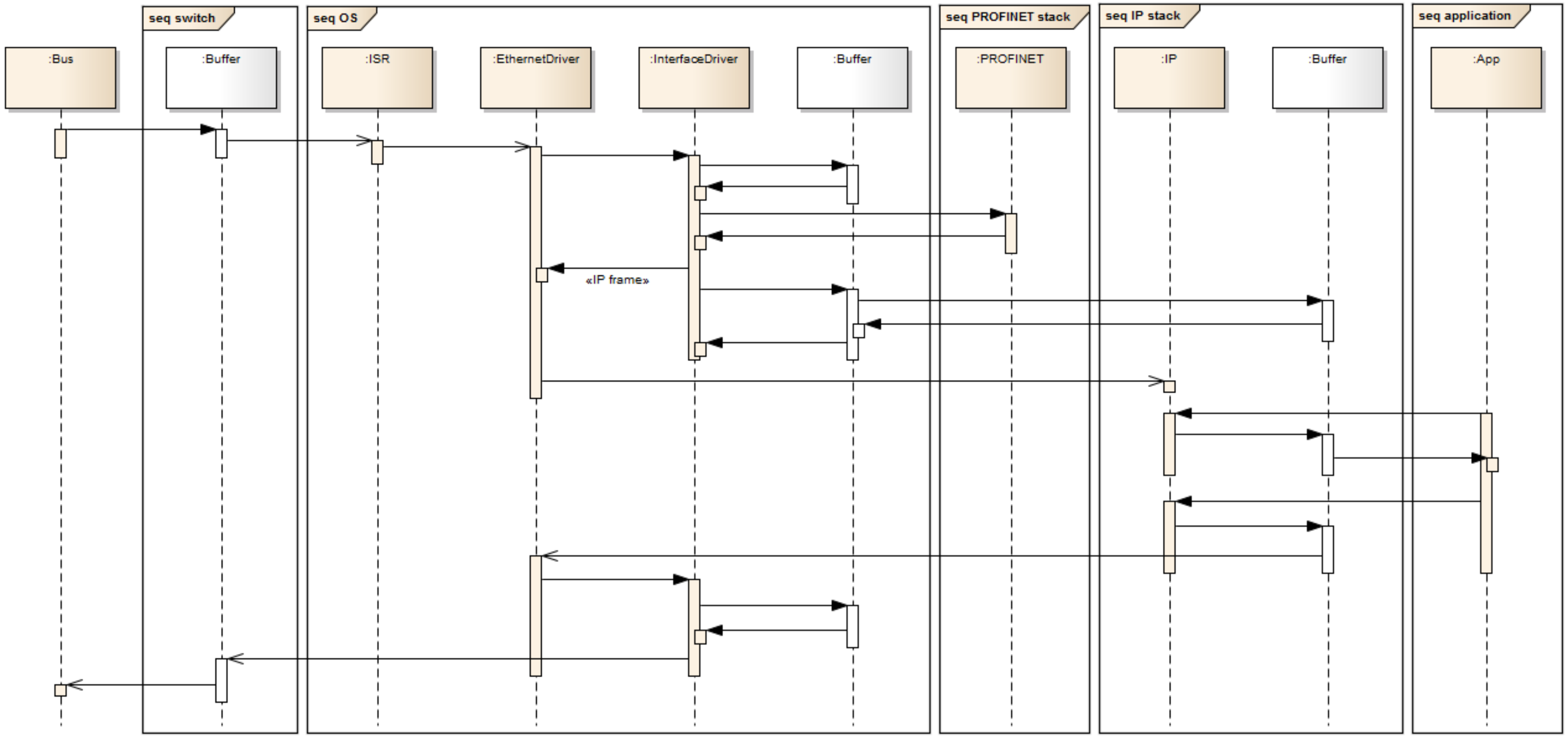
Triple buffer completely decouples CPU from communication



Example Schedule Triple Buffer

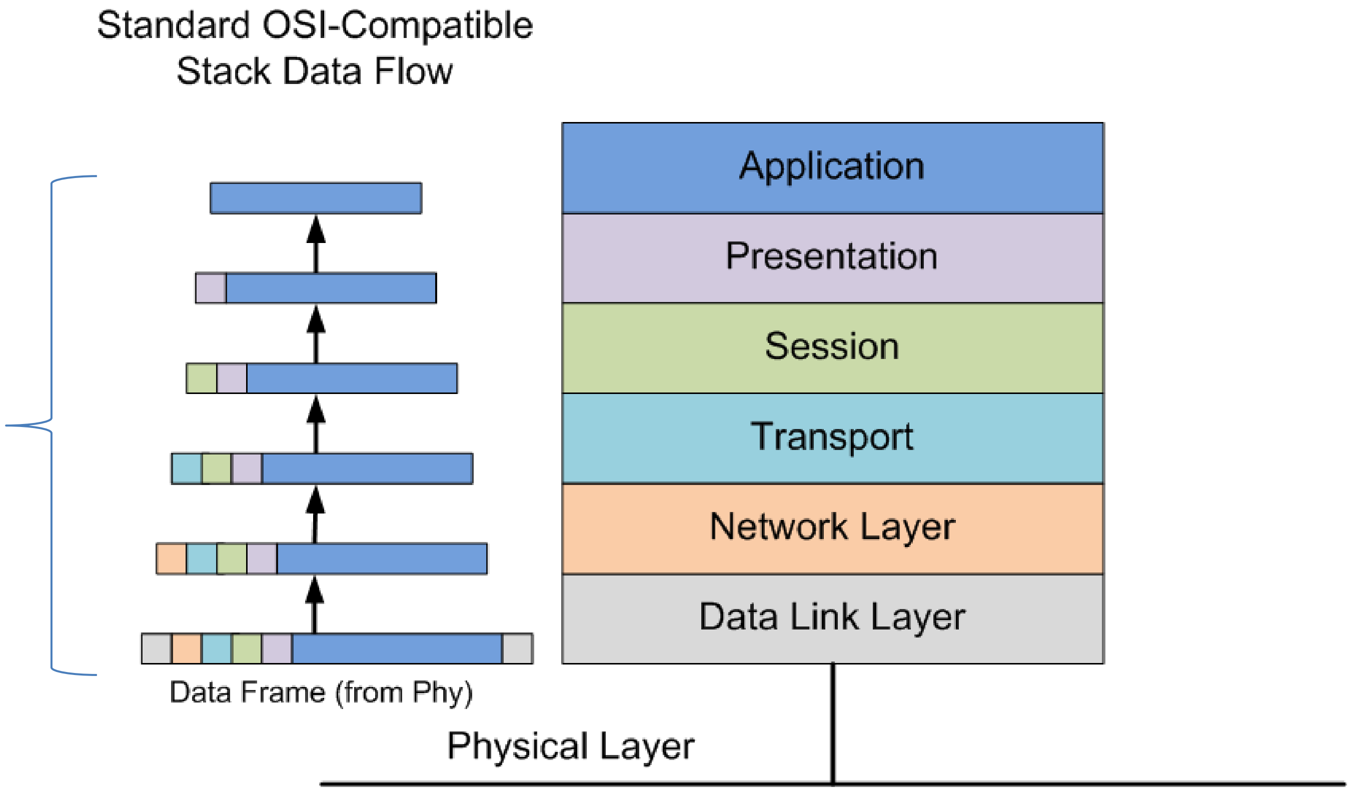
The Software (1)

Sequence diagram describes the passage of an IP frame through a PROFINET node under eCOS using BSD2000 stack. Application is a simple UDP server



Naïve Stack Architecture

Each layer maintains its own buffers



Learning Aim: The student will be able to explain the two presented stack architectures

The Copy Problem

Zürcher Hochschule für Angewandte Wissenschaften

School of Engineering

InES Institute of Embedded Systems

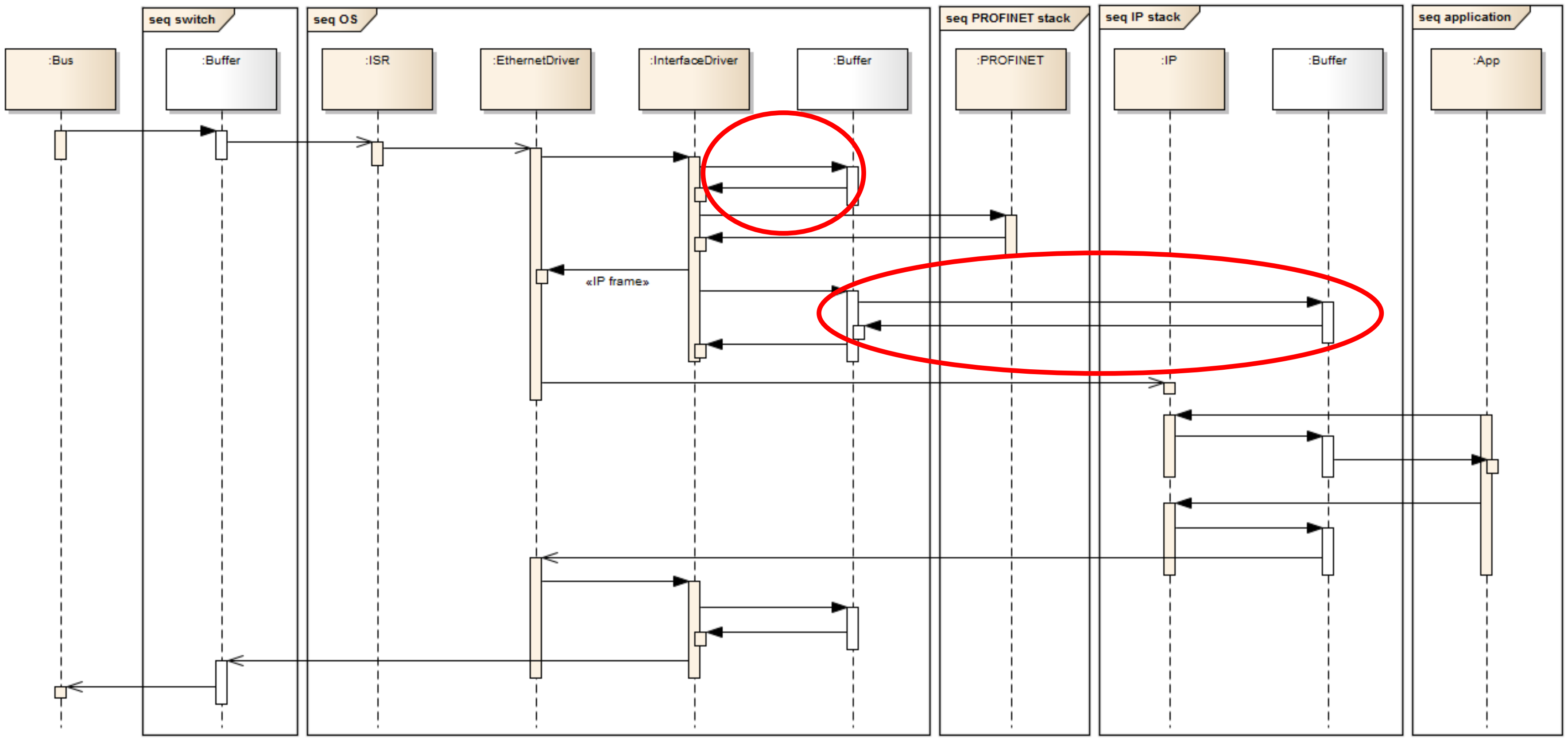
Sequence diagram describes the passage of an IP frame through a PROFINET node

memcpy() !!!

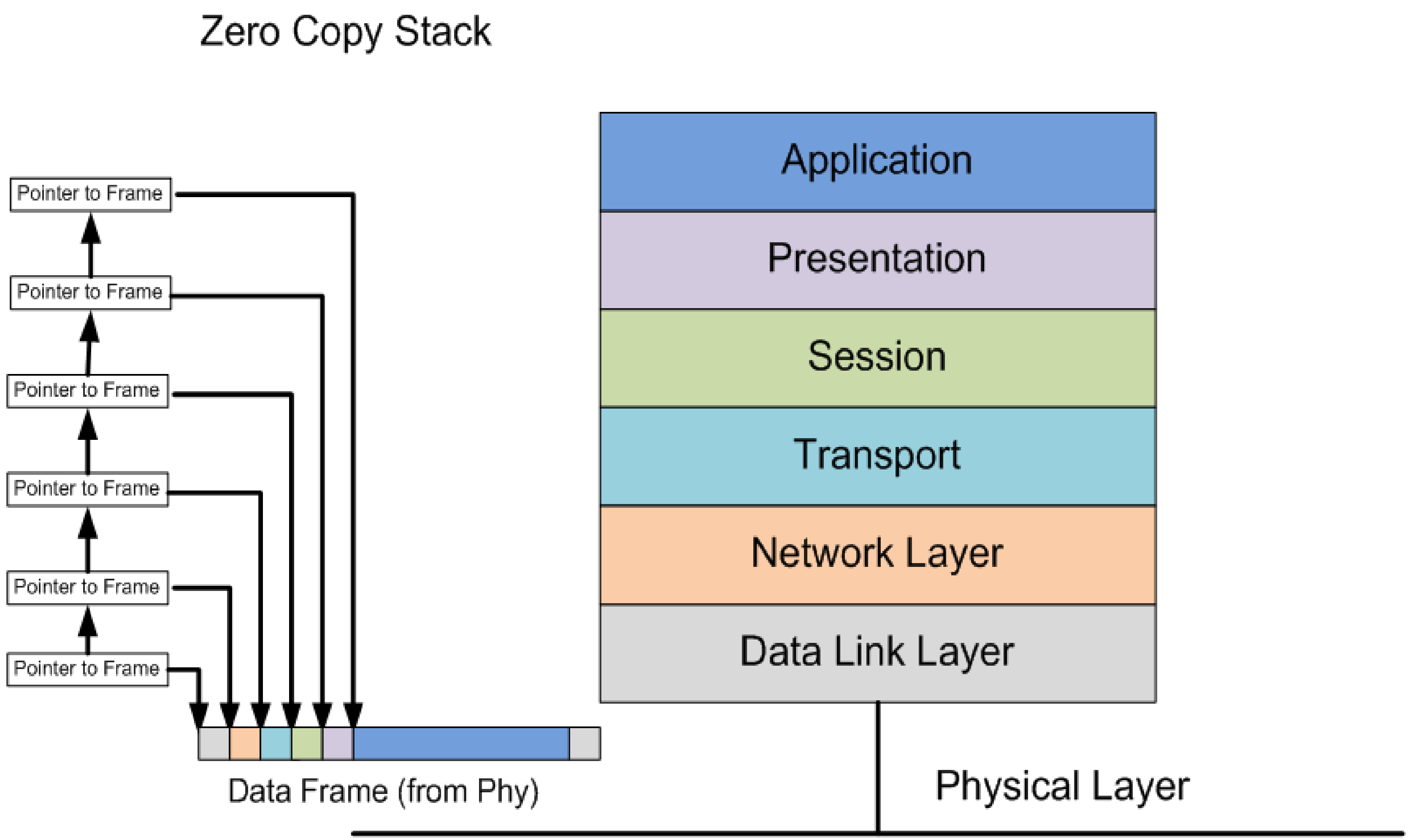
Session 1: Constrained Devices
- Definitions
- Benchmarking
- Computational Power

Session 2: Ethernet Communica
- Physical Layer

Session 3: Communication Stacl
- **Communication Stack**



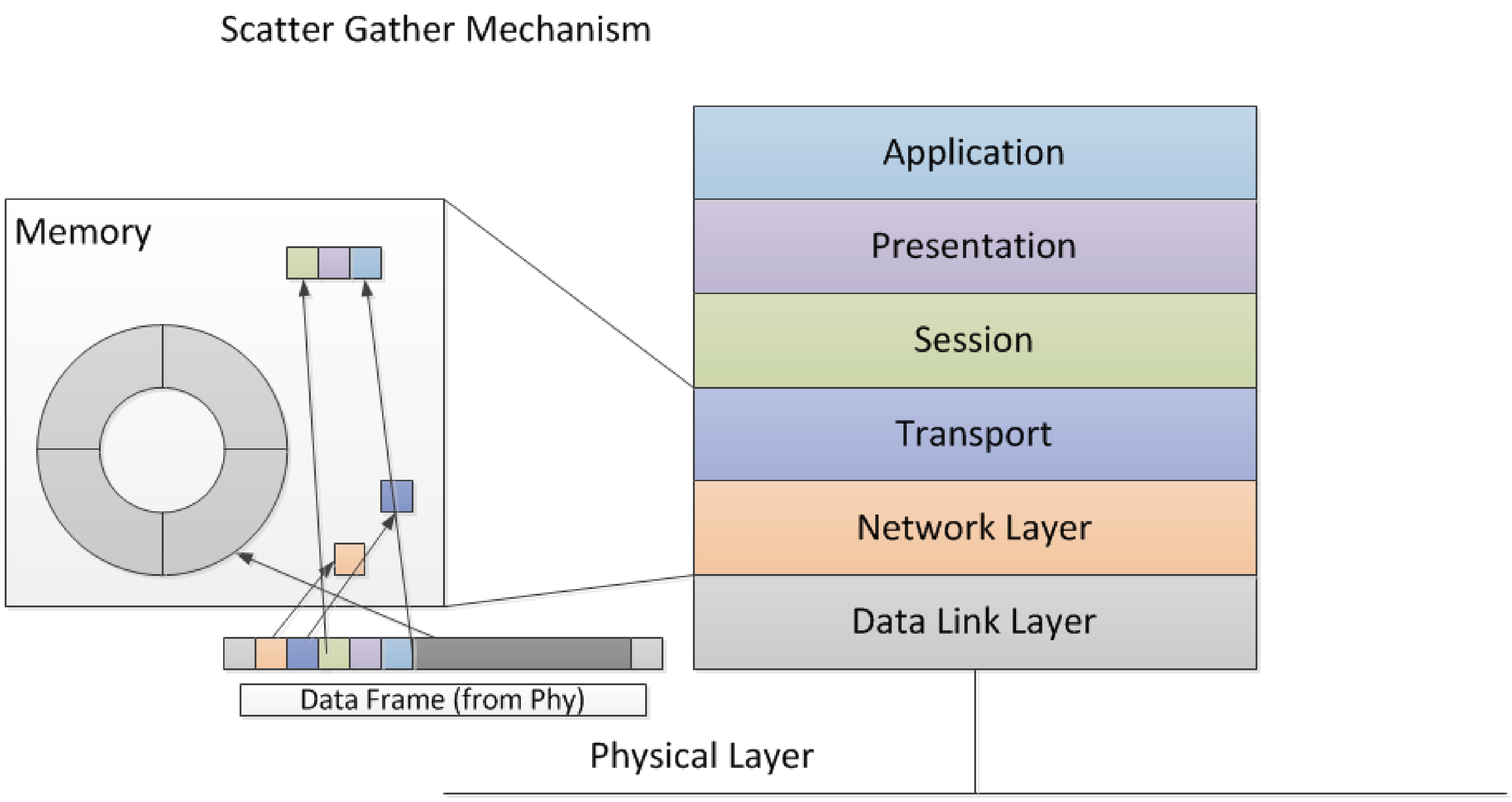
Zero Copy Stack



Learning Aim: The student will be able to explain zero-copy stack

Scatter-gather

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



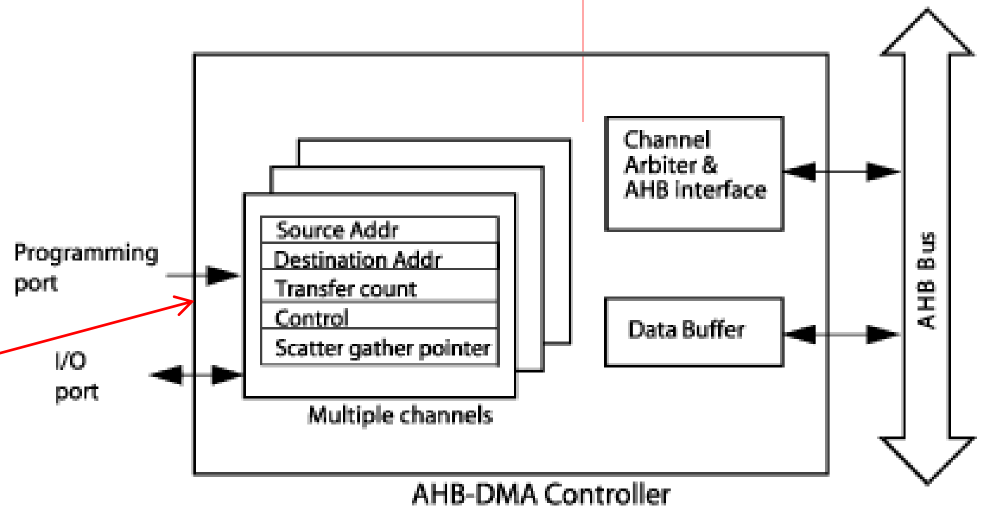
Learning Aim: The student will be able to explain scatter-gather DMA
Learning Aim: The student will understand that an optimised stack architecture utilises scatter-gather and zero-copy features

Scatter-gather – example code

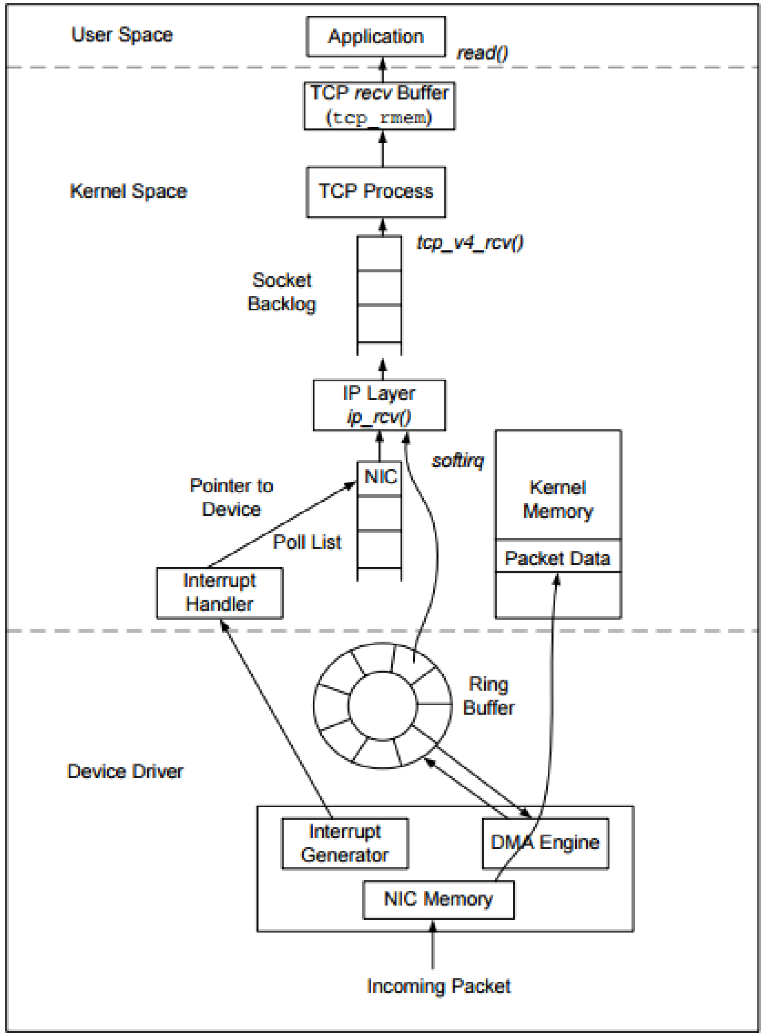

```
static void sopc_bulk_aio_2p_recv( struct eth_drv_sc *sc, struct eth_drv_sg *sg_list, int sg_len) {
    cyg_uint32  pos;
    cyg_uint8   i;

    sopc_bulk_aio_2p_priv_data *cpd = (sopc_bulk_aio_2p_priv_data *)sc->driver_private;
    debug1_printf("recv packet with elem %d len %d\n", sg_len, cpd->recv_len);

    pos = 0;

    for (i=0; i<sg_len; i++) {
        debug1_printf("r: sg_list elem %d, len %d\n", i, sg_list[i].len);
        memcpy((void*)sg_list[i].buf, &(cpd->recv_buffer[pos]), sg_list[i].len);
        pos += sg_list[i].len;
    }
}
```

There's memcpy again !!

lucky we have HW guys ….

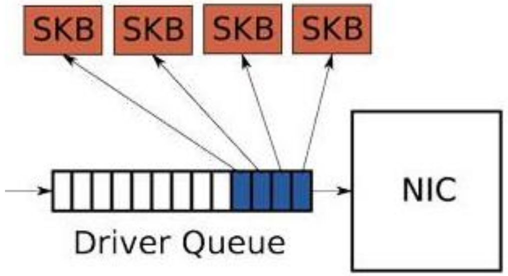


Linux



- The NIC maintains a ring buffer of descriptors to sk_buffs. These are initalised by the kernel and point to kernel locations where frames and their meta data can be stored. The packet data is part of this structure and has pointers to the data in the structure.

- When a packet is passed from IP to TCP then the sk_buff is cloned. The metadata is copied but the packet data isn't. The clone is then passed from IP to TCP - the packet itself remains in kernel memory



The student will be able to explain how Linux Implements zero-copy

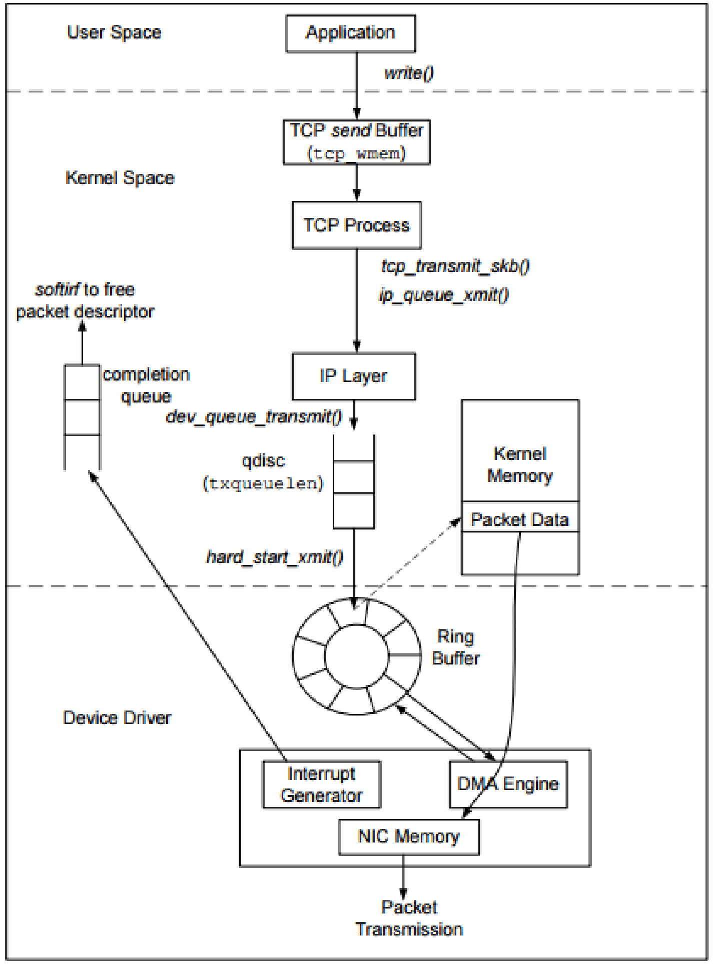
Learning Aim: The student will be able to explain how Linux implements zero-copy

http://www.ece.virginia.edu/cheetah/documents/papers/TCPlinux.pdf

Linux – sk_buff

The process for transmit is broadly similar below the first sk_buff entries

```
struct sk_buff {
    /* These two members must be first. */
    struct sk_buff          *next;
    struct sk_buff          *prev;

    struct sock             *sk;      /* owner socket */
    struct skb_timeval      tstamp;   /* arrival time */
    struct net_device       *dev;     /* output dev   */
    struct net_device       *input_dev;  /* input dev */
        union {
                struct tcphdr   *th;
                struct udphdr   *uh;
                struct icmphdr  *icmph;
                struct igmphdr  *igmph;
                struct iphdr    *ipiph;
                struct ipv6hdr  *ipv6h;
                unsigned char   *raw;
        } h;      // <--- Transport header address

        union {
                struct iphdr    *iph;
                struct ipv6hdr  *ipv6h;
                struct arphdr   *arph;
                unsigned char   *raw;
        } nh;     // <--- Network header address

        union {
                unsigned char   *raw;
        } mac;    // <--- MAC header address
```



http://www.ece.virginia.edu/cheetah/documents/papers/TCPlinux.pdf

Linux – User Interface Sockets (1)

The socket is an interface methodology developed at UC Berkeley

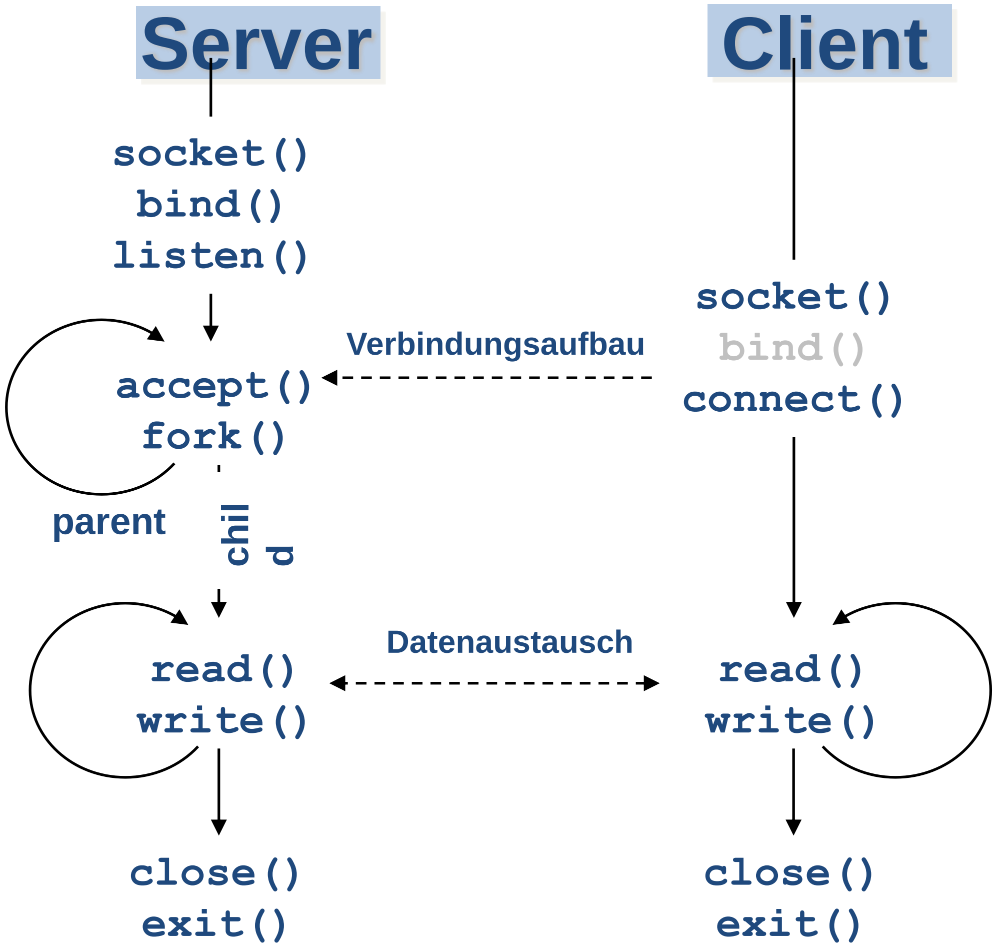
It is optimised for streaming interfaces

It has become the de-facto standard networking interface

In our world we usually identify it with TCP and UDP protocols however it also supports other protocols such as AppleTalk and X.25

It also supports raw sockets – Layer 2/3 packets

Like most Linux I/O sockets are realised as file operations using SOCKFS and socket/close/read/write operations



Learning Aim: The student will be able to explain how Linux implements sockets
Learning Aim: The student will be able to explain how Linux implements client-server communications

Linux – Sockets (2)

- Create a socket

```
/* Create socket */
socket_desc = socket( AF_INET , SOCK_STREAM , 0 );
if( socket_desc == -1 )
{
  fprintf( stderr, "could not create socket. exiting ...\n" );
  exit( EXIT_FAILURE );
}
```

IPV4 **TCP**

```
server.sin_addr.s_addr = inet_addr( "195.176.255.236" );
server.sin_family = AF_INET;
server.sin_port = htons( 80 );

/* Connect to remote server */
if( connect( socket_desc, ( struct sockaddr * )&server , sizeof( server ) ) < 0 )
{
  fprintf( stderr, "connect error. exiting ...\n" );
  exit( EXIT_FAILURE );
}
```

Connect to this address/port

Could be write()

```
/* Send some data (in this case a http request) */
message = "GET / HTTP/1.1\r\n\r\n";

if( send( socket_desc, message, strlen( message ), 0 ) < 0 )
{
  fprintf( stderr, "failed to send data. exiting ...\n" );
  exit( EXIT_FAILURE );
}
```

Send HTTP GET

Linux – Sockets (3)

Session 1: Constrained Devices
- Definitions
- Benchmarking
- Computational Power

Session 2: Ethernet Communication
- Physical Layer

Session 3: Communication Stacks
- **Communication Stack**

Wait for reply

```
/* receive a reply from the server */
if( recv( socket_desc, server_reply , 2048 , 0 ) < 0 )
{
  fprintf( stderr, "failed to receive data from server. exiting ...\n" );
  exit( EXIT_FAILURE );
}
```

Could be read()

Close socket – terminate TCP connection

```
errsv = close( socket_desc );
simple_err_check( "failed to close socket file descriptor.", errsv );
```

Linux – Raw Sockets

Session 1: Constrained Devices
- Definitions
- Benchmarking
- Computational Power

Session 2: Ethernet Communication
- Physical Layer

Session 3: Communication Stacks
- **Communication Stack**

Use IPV4 infrastructure

```
/* Submit request for a socket descriptor to look up interface. */
if( ( sd = socket( AF_INET, SOCK_RAW, IPPROTO_RAW ) ) < 0 )
{
  fprintf( stderr, "socket() failed to get socket descriptor for usir
  print_usage();
  exit( EXIT_FAILURE );
}
```

raw socket **No IP header**

Raw Sockets allow you build your own protocols on user level

```
/* Sender hardware address (48 bits): MAC address */
memcpy( &arphdr.sender_mac, src_mac, 6 * sizeof( uint8_t ) );
```