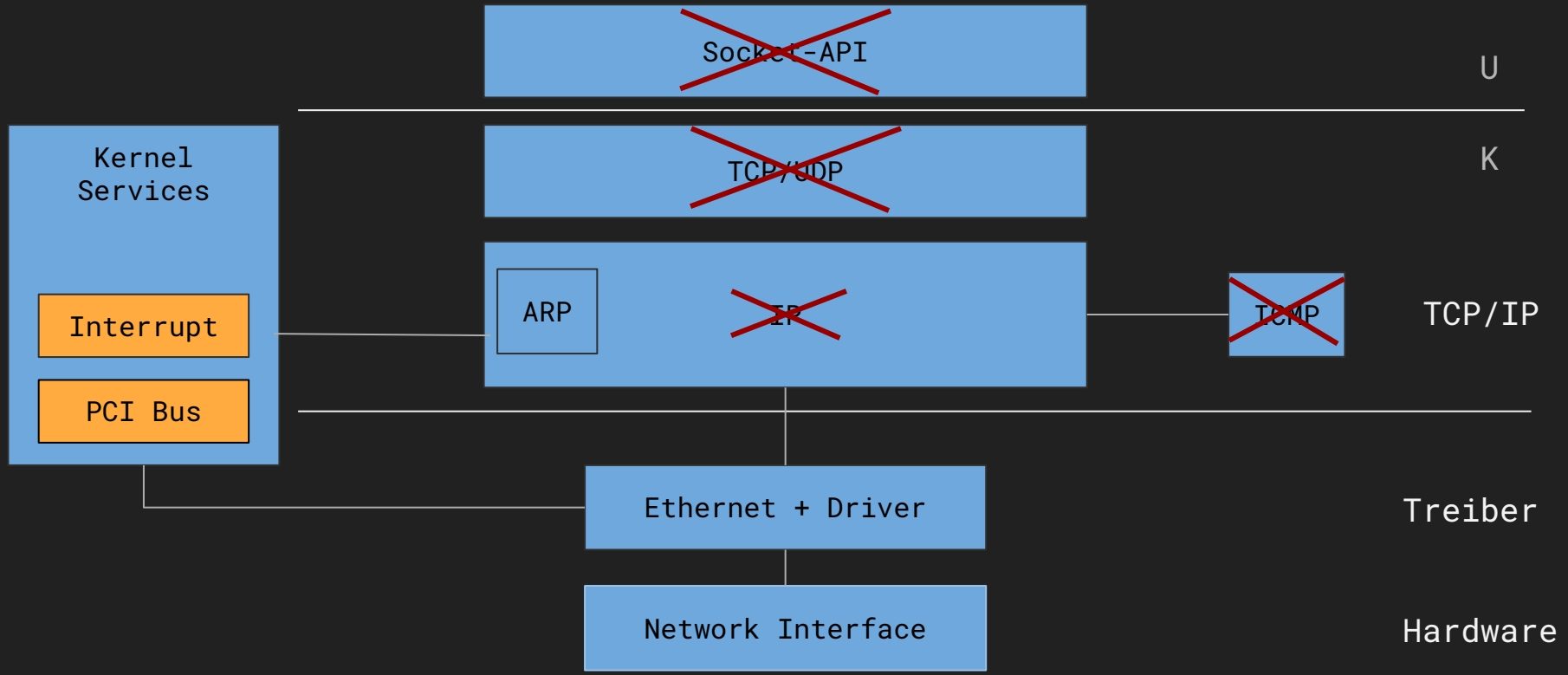
Project Kraken

A Network Stack for Ninjastorms OS


DEMO

Was war unser Scope?

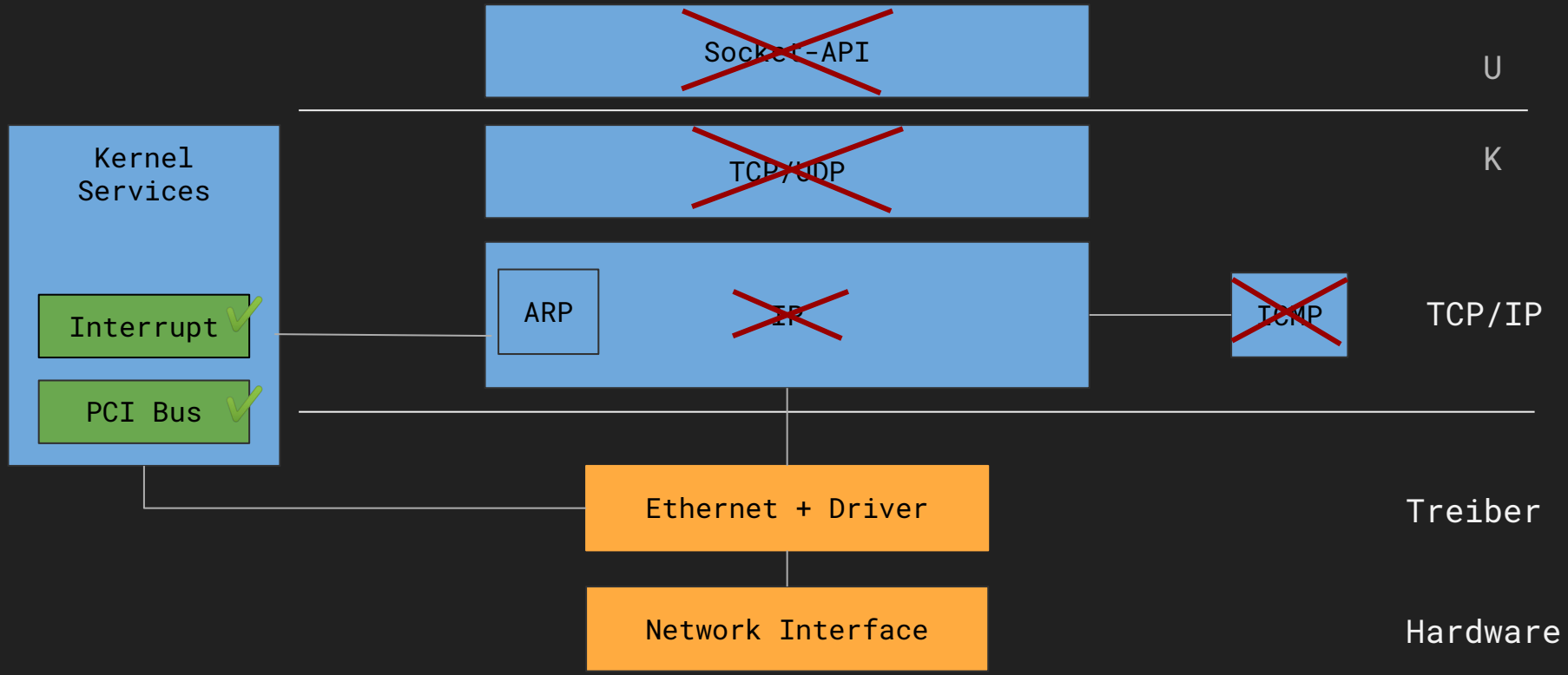


Kommunikation mit der NIC - PCI-seitig

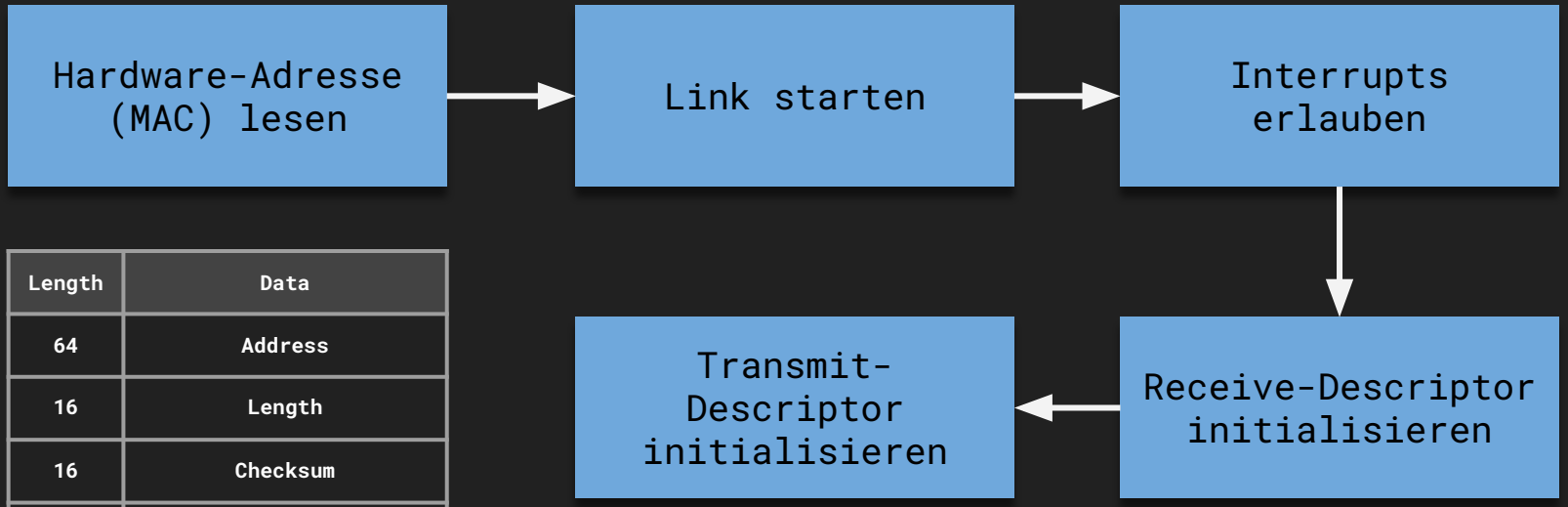
Notwendige Schritte:

1. PCI Controller konfigurieren, um mit ihm zu reden
2. PCI Enumeration zum Auslesen aller Devices
    a. Simples Iterieren über Adressbereich der PCI-Devices
    b. Suchen nach unserem Device anhand der Vendor-ID der NIC

Was war unser Scope?



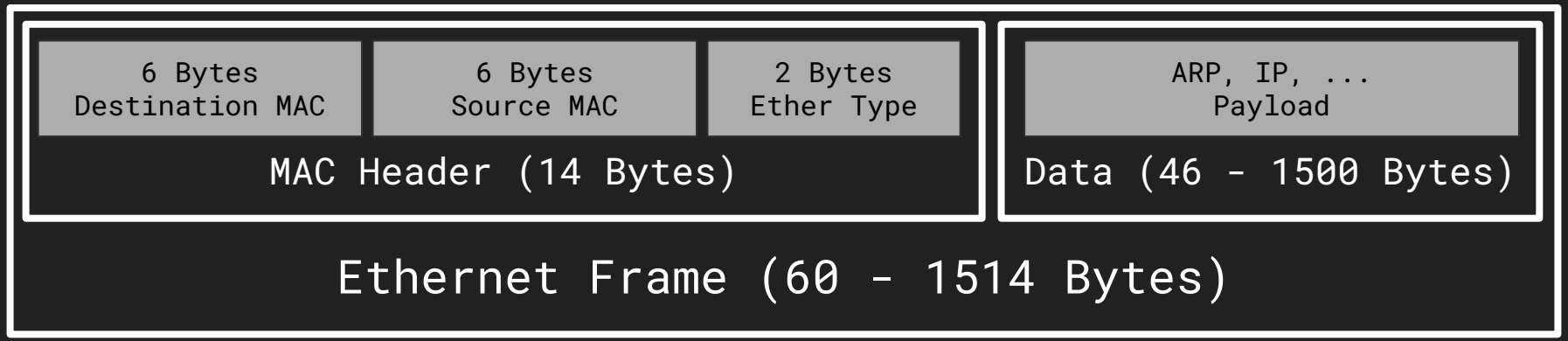
Kommunikation mit der NIC selbst



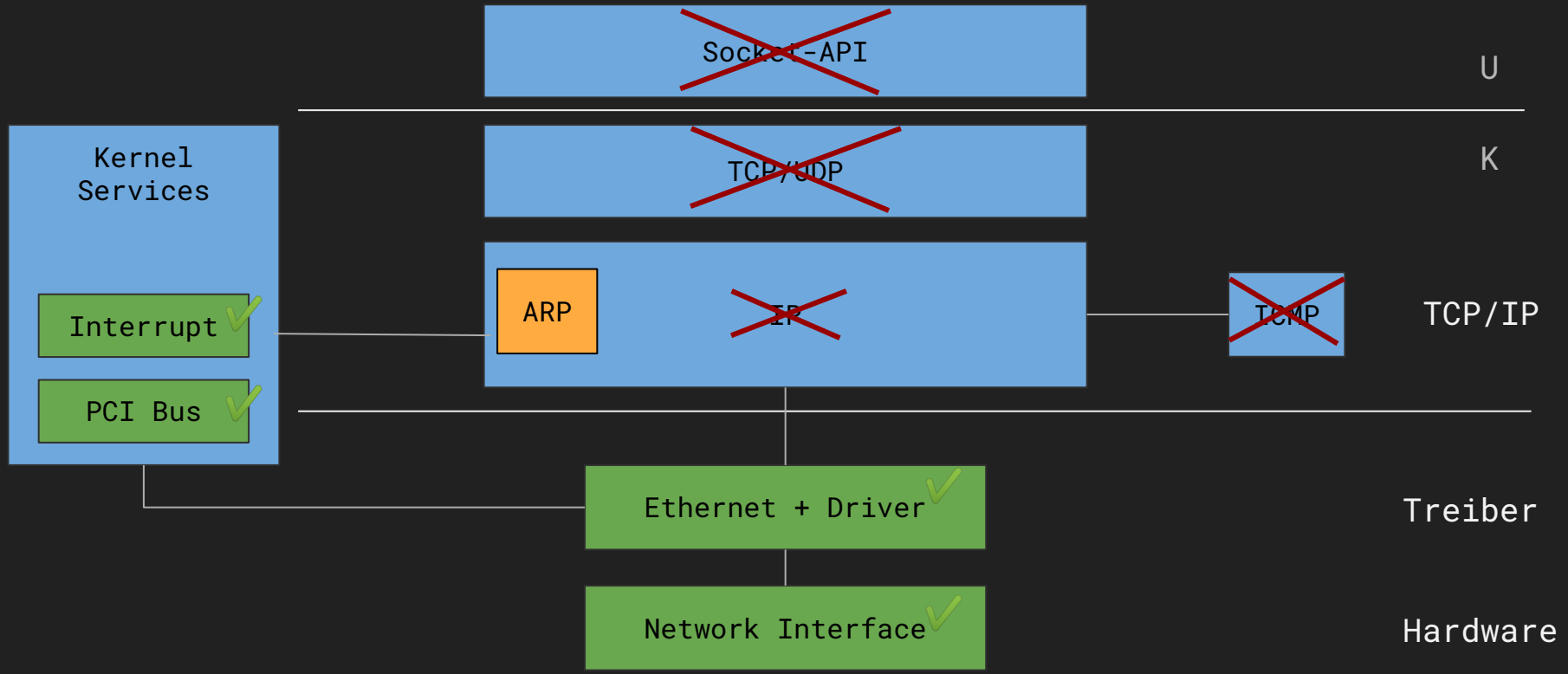
Length	Data
64	Address
16	Length
16	Checksum
8	Status
8	Errors
16	Special

Receive Descriptor im E1000

Aufbau eines Ethernet Frames



Was war unser Scope?



ARP-Frames

- **Hardware-Type**:
  Ethernet (Länge 6)
- **Protocol-Type**:
  IPv4 (Länge 4)
- **Operation**:
  Reply / Request

Octet Offset	0	1
0	Hardware Type	
2	Protocol Type	
4	Hardware Address Length	Protocol Address Length
6	Operation	
8	Sender Hardware Address	
10		
12		
14	Sender Protocol Address	
16		
18	Target Hardware Address	
20		
22		
24	Target Protocol Address	
26		

Wie erkennen wir ARP-Frames?

```
void
start_pdu_encapsulation(raw_packet_t *buf)
{
  ethernet_frame_t* frame = (ethernet_frame_t*) &(buf->data);

  uint16_t ether_type = ntohs(frame->ether_type);

  switch (ether_type)
    {
      case TYPE_ARP:
        arp_receive(frame);
        break;
      case TYPE_IPv4:
      case TYPE_IPv6: // we don't wanna support ipv6 yet ;)
      default:
        break;
    }
}
```

Und wie behandeln wir sie dann? - RFC 826

- Absender in unserer ARP Table?
  - Update ARP Eintrag
- Unsere IP?
  - ARP Eintrag vorher nicht geupdatet?
    - Neuen ARP Eintrag anlegen
  - OPCode = **REQUEST**?
    - ARP Reply an Anfragenden senden

ARP Tabelle

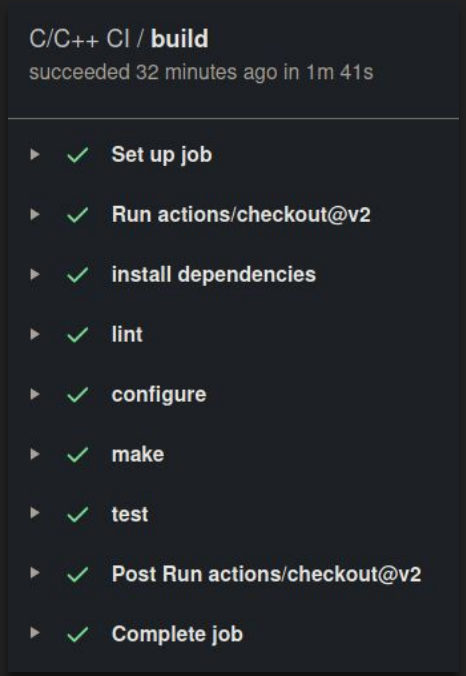
```
[DEBUG] [00:05:921] [kernel/network/routing.c]: Arp Table:
 Time    |        MAC          |      IP
--------+---------------------+-----------------
 5       | f6:09:3d:ac:44:46 | 10.0.2.15
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
 0       | 00:00:00:00:00:00 | 0.0.0.0
```

Was wir noch so gemacht haben

```
[DEBUG] [00:00:783] [kernel/main.c]: DEBUG
[INFO]  [00:00:783] [kernel/main.c]: INFO
[WARN]  [00:00:783] [kernel/main.c]: WARN
[ERROR] [00:00:783] [kernel/main.c]: ERROR
[FATAL] [00:00:783] [kernel/main.c]: FATAL
  task a: 8
[DEBUG] [00:00:896] [kernel/main.c]: DEBUG
[INFO]  [00:00:896] [kernel/main.c]: INFO
[WARN]  [00:00:896] [kernel/main.c]: WARN
[ERROR] [00:00:896] [kernel/main.c]: ERROR
[FATAL] [00:00:896] [kernel/main.c]: FATAL
```

Zeit seit Systemstart

Logging mit Level



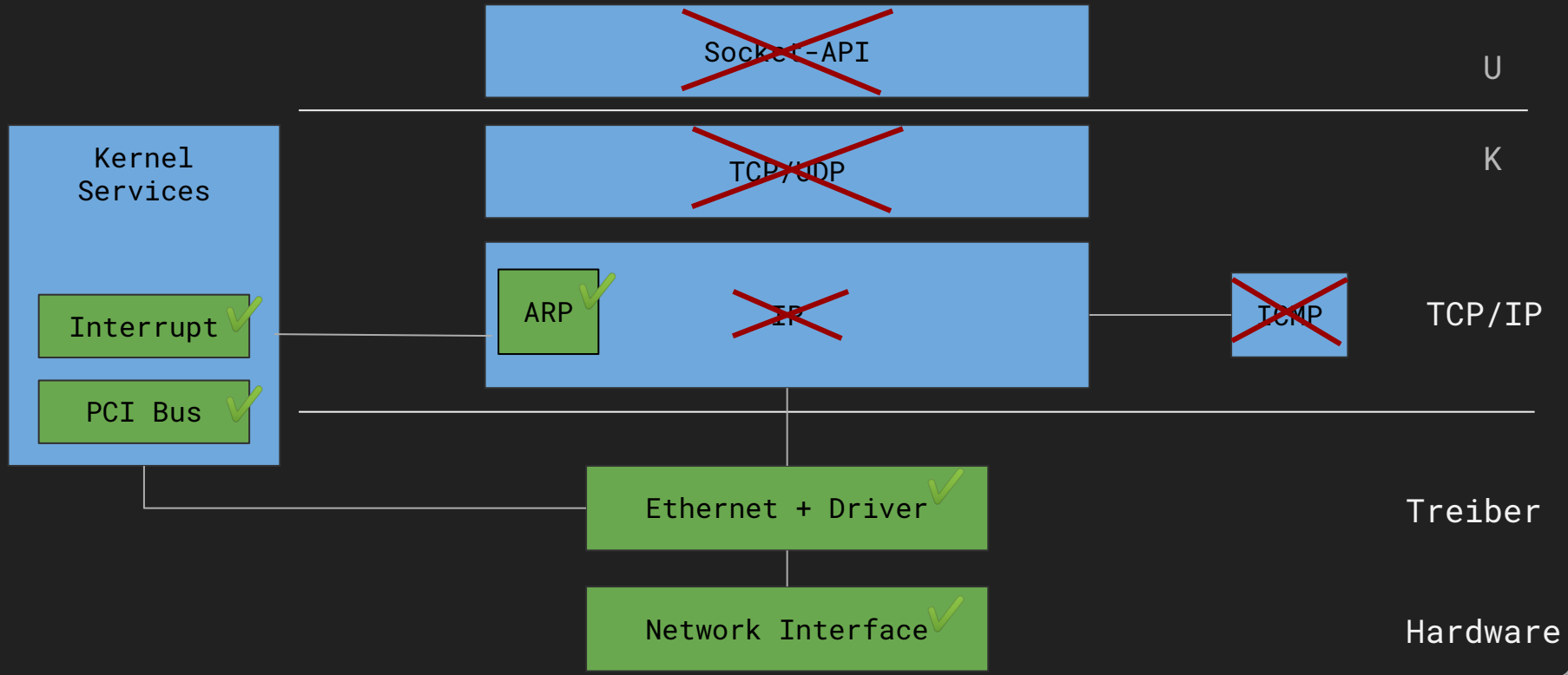
CI

Was wir noch so gemacht haben

```
/*
 * A very basic malloc that uses a sbrk with global memory. Use with care!
 * It always returns 16-byte aligned memory addresses.
 * It is also NOT thread safe!
 *
 * Space layout is as follows:
 *
 * Offset              Memory
 *  0x00   +--------------------------------+
 *         | 16-byte aligned header block_t |
 *  0x10   +--------------------------------+
 *         |   previously allocated space   |
 *         |         (e.g. 4 byte)          |
 *  0x14   +--------------------------------+
 *            unused space to align header
 *  0x20   +--------------------------------+
 *         |  16-byte aligned next header   |
 *  0x30   +--------------------------------+
 *         |       .................        |
 *         |     requested program space    |
 *         |       .................        |
 *         +--------------------------------+
 * In this example the return value of malloc is a pointer to 0x30 so header
 * at 0x20 won't be overwritten by the requester.
 * free can then calculate back from 0x30 to 0x20 to get original header.
 */
void *
malloc(size_t size)
```

malloc

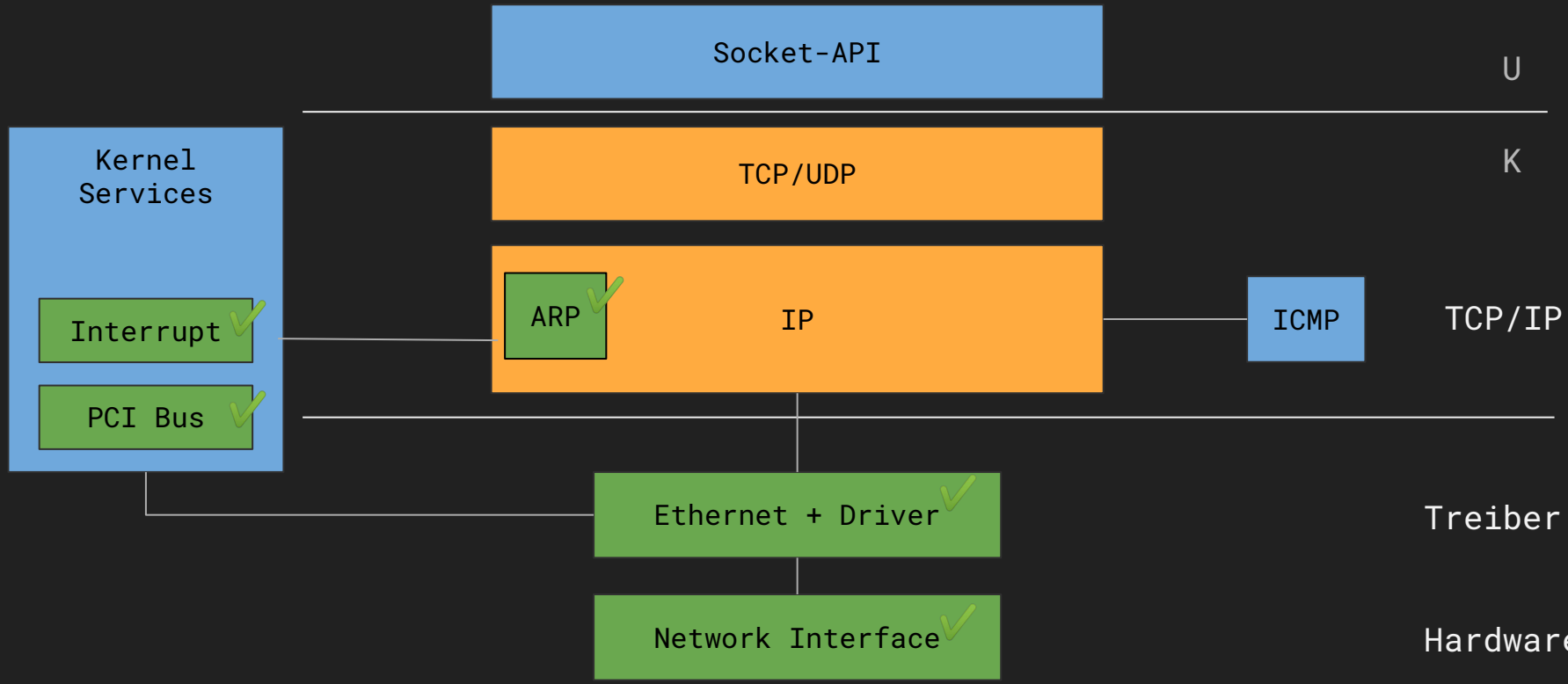
Was war unser Scope?



Probleme

- Verständnis der Dokumentation (“set appropriately”) und die Verweise auf die beiliegende CD
- Chip nutzt Little Endian, in Netzwerken ist Big Endian die Konvention
- vprintf und die Arbeit mit longs

Ausblick


Socket-API
U
Kernel Services
TCP/UDP
K
Interrupt
ARP
IP
ICMP
TCP/IP
PCI Bus
Ethernet + Driver
Treiber
Network Interface
Hardware