PROJEKTARBEIT: RASPBERREY PI

Modul: Grundlagen der technischen Informatik

LUKAS BAI

KENNETH JOSS

INHALT

AUFGABENSTELLUNG

VORGEHEN

REFLEXION

DISKUSSION

DEMO

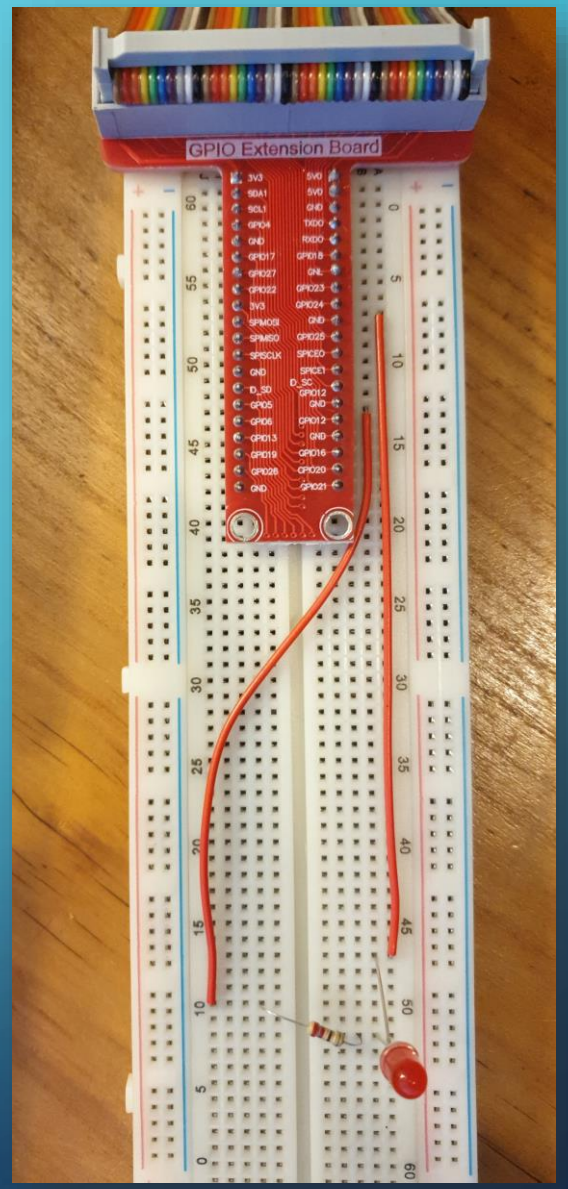
AUFGABENSTELLUNG

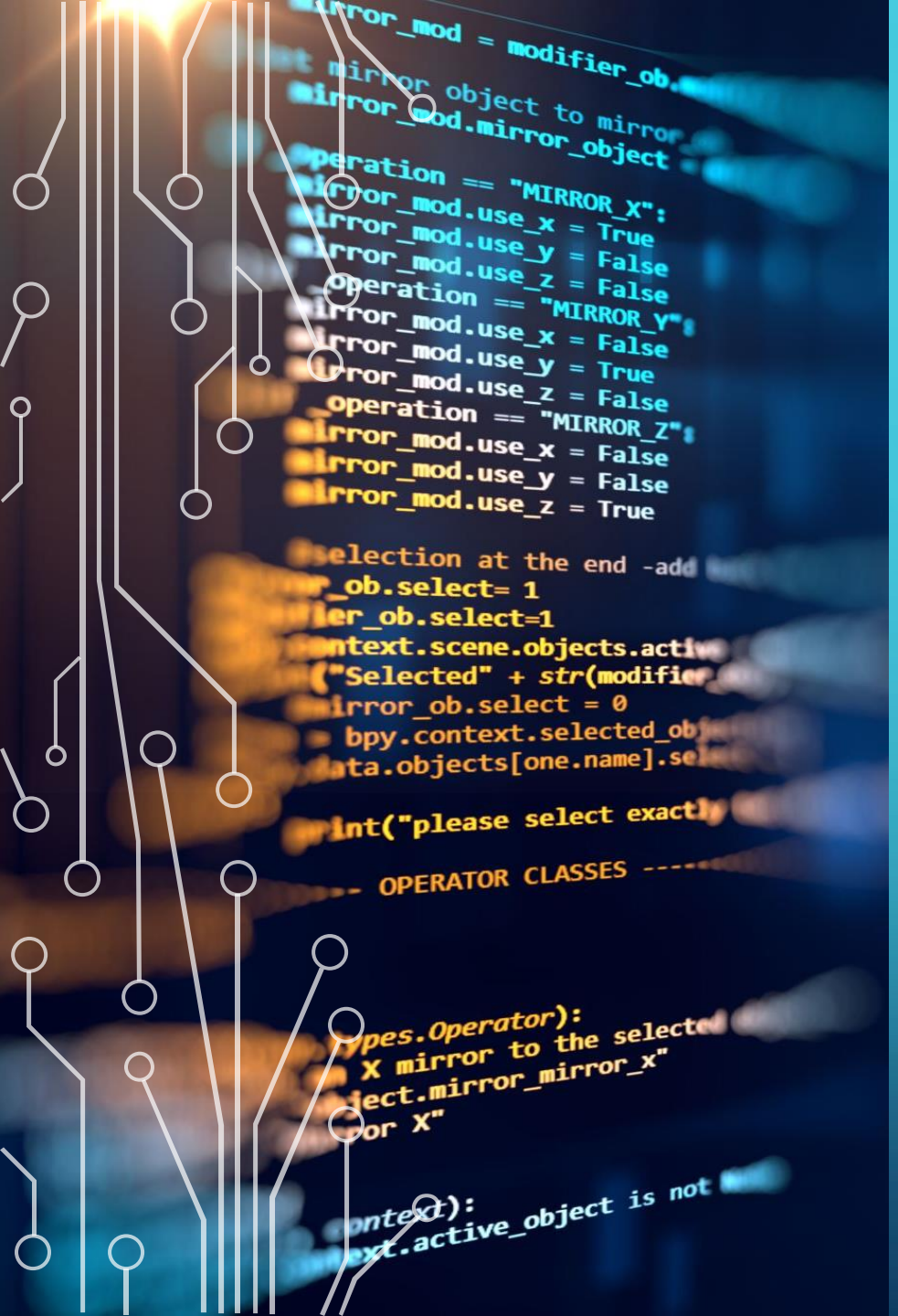
- Minimale Anforderungen
  - Sub-Routinen
  - LED on/off mittels Timer
  - Andere Signale als SOS möglich

- Zusätzliche Anforderung
  - Ausgabe eines beliebigen Text in Morsecode
    - Morsesequenz im RAM abgespeichert

UMSETZUNG

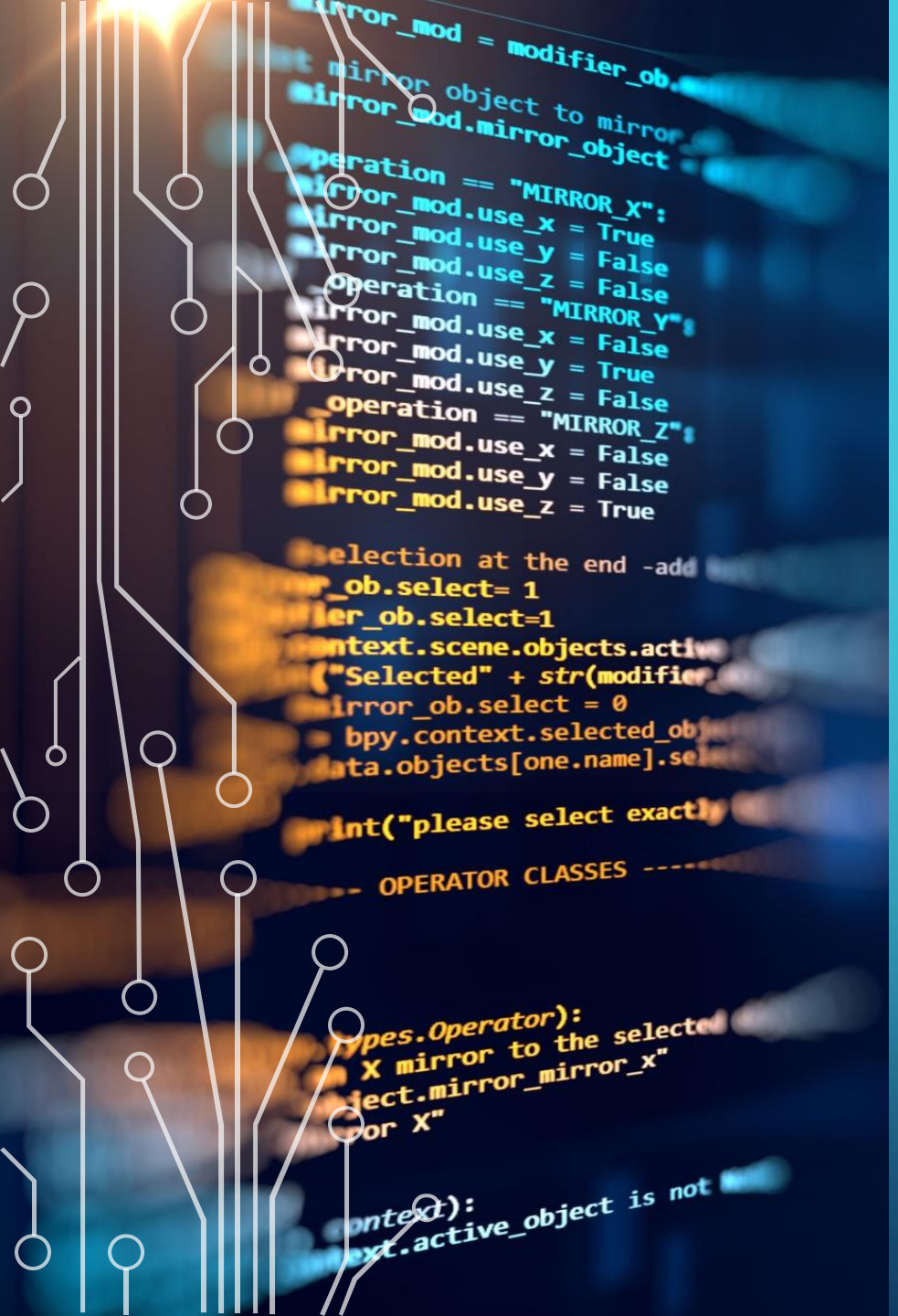
- Arbeitsumgebung: Windows & Linux
- Toolchain: GNU
- Hardware Setup





VORGEHEN

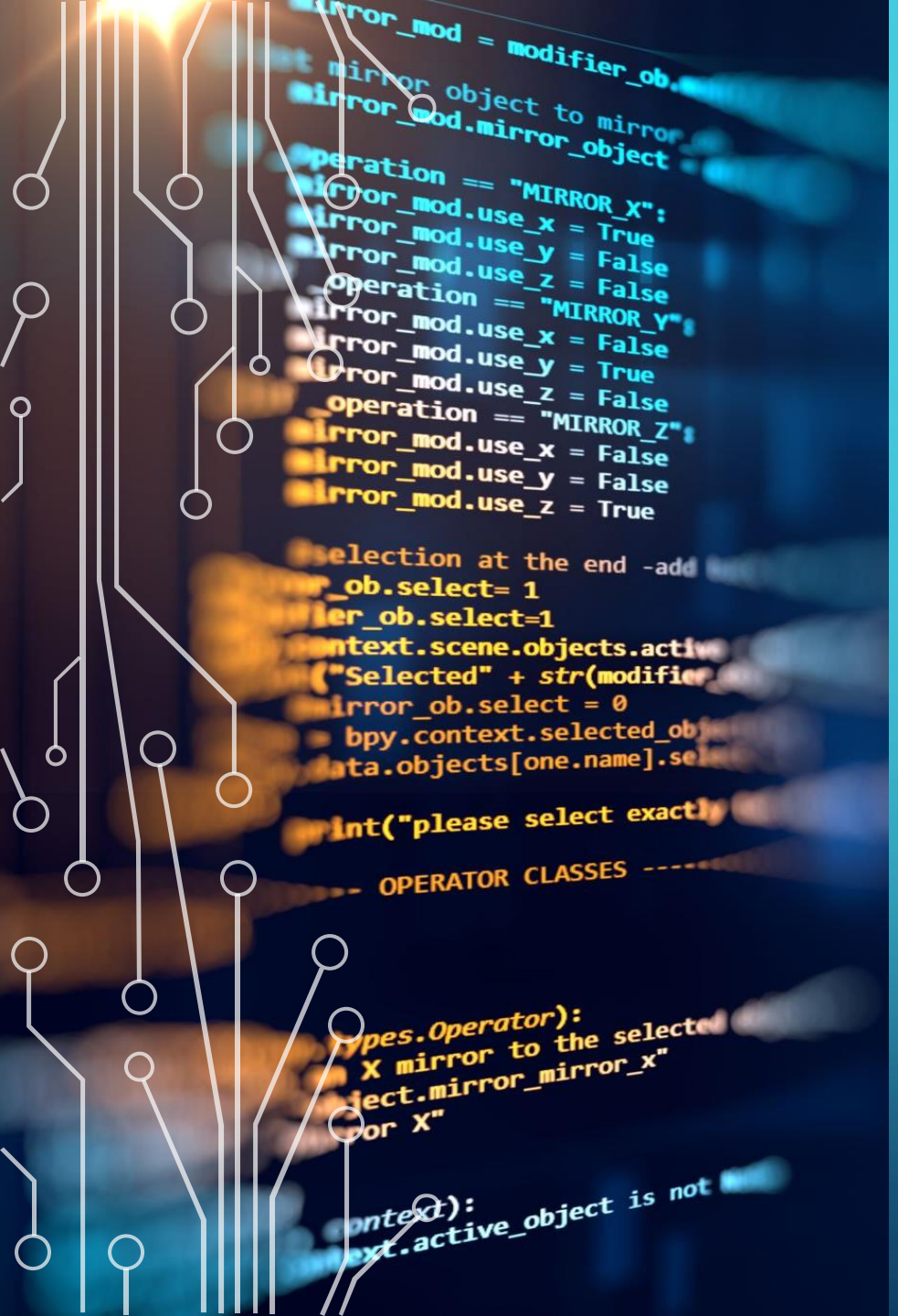
- Vorbereitung
  - Verkabelung GPIO Extension Board
  - Funktionstest mit Python Script
  - GPIO Basis Adresse ausfindig machen (Userspace)
- Vorgehen
  - LED leuchten mit Assambler
  - LED Blinken mit HW Timer
  - Stackpointer, Function-Call, variablen Text implementeiren
  - Code optimieren



- RAM Adresse definieren
  - RAM Adresse definieren
  - Start an Adresse 0x80000
  - Wächst gegen 0x0

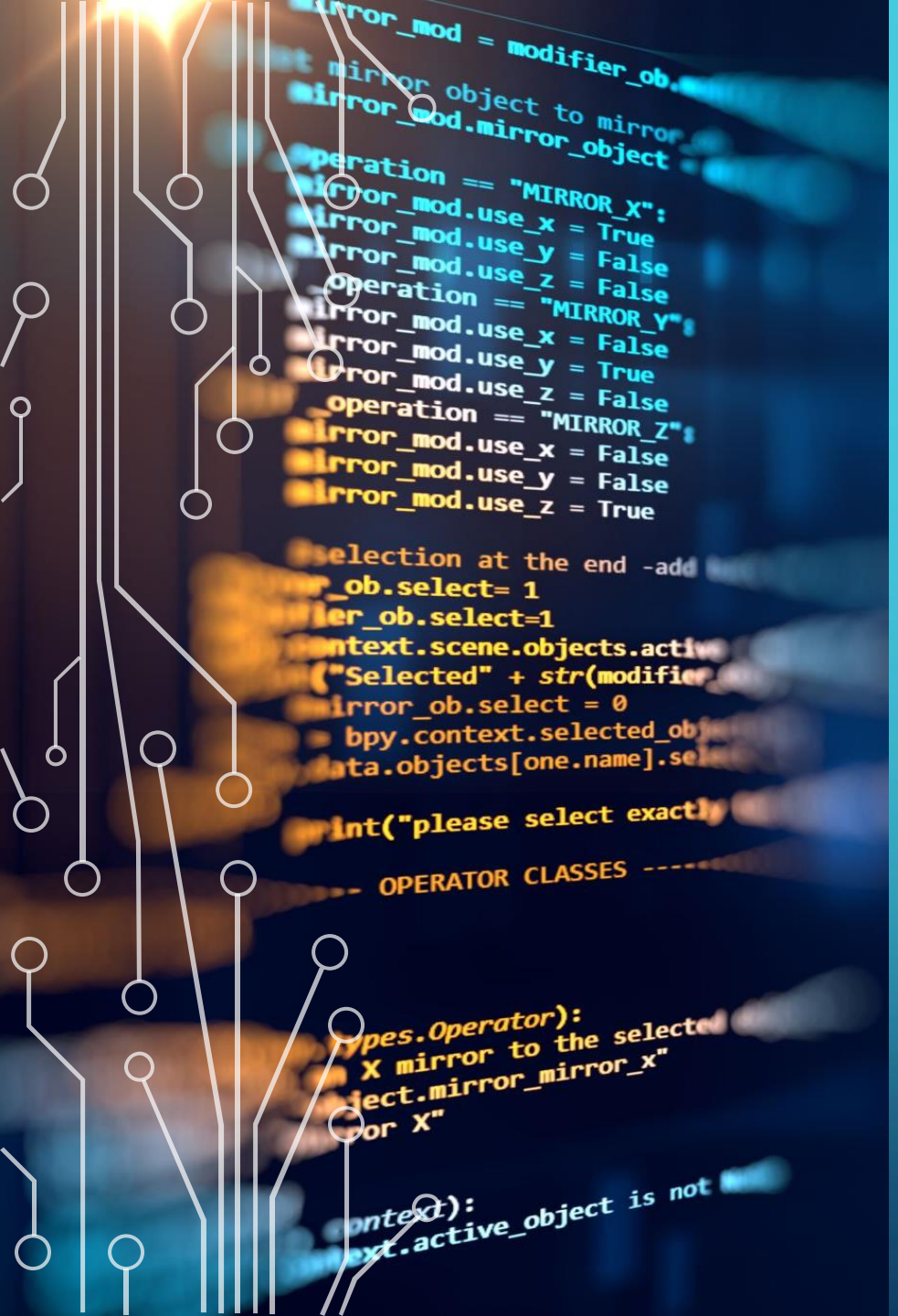
src > sys.ld

```
1   /* https://www.math.utah.edu/docs/info/ld_3.html */
2
3   MEMORY
4   {
5       ram : ORIGIN = 0x80000, LENGTH = 0x10000
6   }
7
8   SECTIONS
9   {
10      .text   : {*(.text*)} > ram
11      .bss    : {*(.bss*)} > ram
12  }
13
```



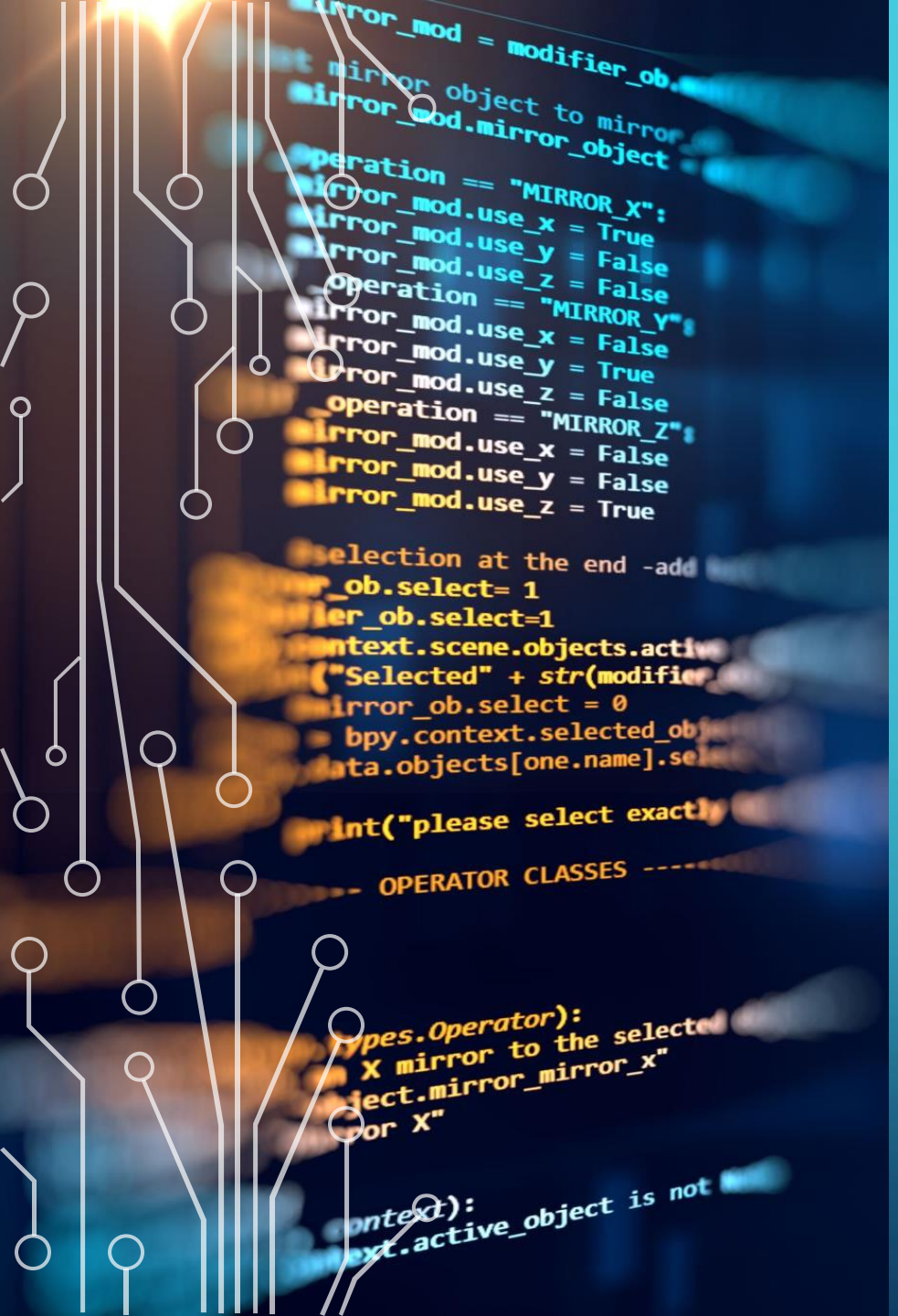
- Stackpointer initialisieren
  - SP init mit 0x80000
  - SP wächst gegen 0x0, PC zwischen 0x80000 / 0x90000
  - SP: Spezialregister
    - R29: Framepointer
    - R30: Procedure link
    - XZR/WZR: Zero register
    - PC: Programcounter

```
26
27  .L_START:  /* We're on the main core! */
28      mov    sp, #0x80000          /* init stack pointer */
```



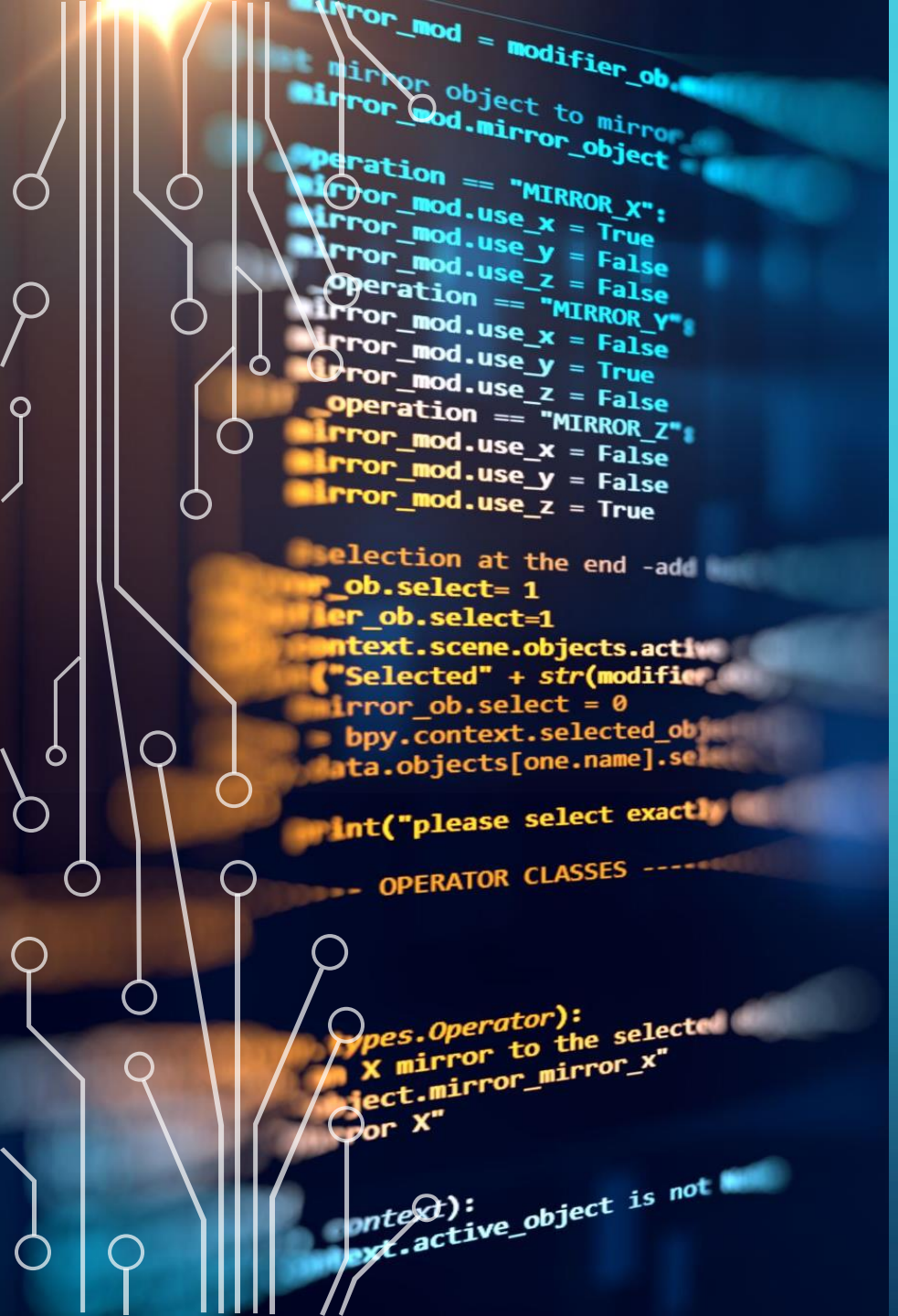
- Funktionsaufruf
  - 419: Funktionsargument in R0 speichern
  - 420: Aufruf von “blink” mit “bl” (branch and link)
  - Stackpointer nach Aufruf wieder gleich wie vorher
  - 421: Register aus dem Stack laden (Context wiederherstellen)

```
417  .L43: /* continue loop */
418      ldr     x0, [sp, 24]          /* load into register: load back sp + 24bit into x0 (text pointer)*/
419      ldrb    w0, [x0]              /* load into register (byte): get char from x0 address (w0 will be argument for blink(char c) */
420      bl      blink                 /* branch and link: call blink(w0) */
421      ldr     x0, [sp, 24]          /* and again load back text address pointer */
422      add     x0, x0, 1             /* increment text address pointer by one */
```



- Funktion
  - 149: x29 (Framepointer) und x30 (Procedure link) im Stack speichern
  - 154: x29 (Framepointer) bekommt neue Stackpointer Adresse

```
147  blink:
148  .LFB7:
149      stp     x29, x30, [sp, -32]!  /* store pair (x29 und x30) into sp -32, sp = sp -32
150                                       https://stackoverflow.com/questions/64638627/expla
151                                       1. suptract 32 from address stored at sp (stack po
152                                       2. store x29 and x30 into memory at address stored
153                                    */
154      mov     x29, sp               /* store stack pointer into register 29 */
155      strb    w0, [sp, 31]          /* store argument (char) into stack at 31 */
156      ldrb    w0, [sp, 31]          /* .. load back into register w0 (why?) */
157      cmp     w0, 97                /* is it 97 ('a') ? */
158      bne     .L11                  /* else try next at .L11 */
```



- Return
  - 403: Vorheriger Stackframe und Procedure link laden
  - 404: Springern auf vorherige Adresse

```
400 .L12: /* end of function (everithing meets here) */
401     mov     w0, 2
402     bl      space
403     ldp     x29, x30, [sp], 32      /* free stack and restore registers... */
404     ret                             /* ... and return to old pc */
405
```

REFLEXION

- Einfaches Setup der Toolchain unter Linux
- Assembler verstehen eher schwierig
- Headless Setup und schwieriges debugging
- Zeitdruck neben dem Job und anderen Studien-Arbeiten

DISKUSSION

FRAGEN UND ANREGUNGEN?

DEMO

DEMONSTRATION DER LÖSUNG