1 Programmierung

Ein erstes Programm

```
 0:  df 93
 2:  cf 93
 4:  00 d0
 6:  00 d0
 8:  cd b7
 a:  de b7
 c:  9a 83
 e:  89 83
10:  7C 83
12:  6b 83
14:  29 81
16:  3a 81
```

```
18:  8b 81
1a:  9c 81
1c:  82 0f
1e:  93 1f
20:  0f 90
22:  0f 90
24:  0f 90
26:  0f 90
28:  cf 91
2a:  df 91
2c:  08 95
```

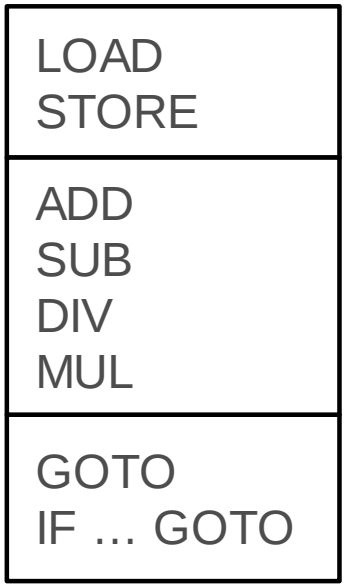
1.1 Arbeitsweise eines Computers

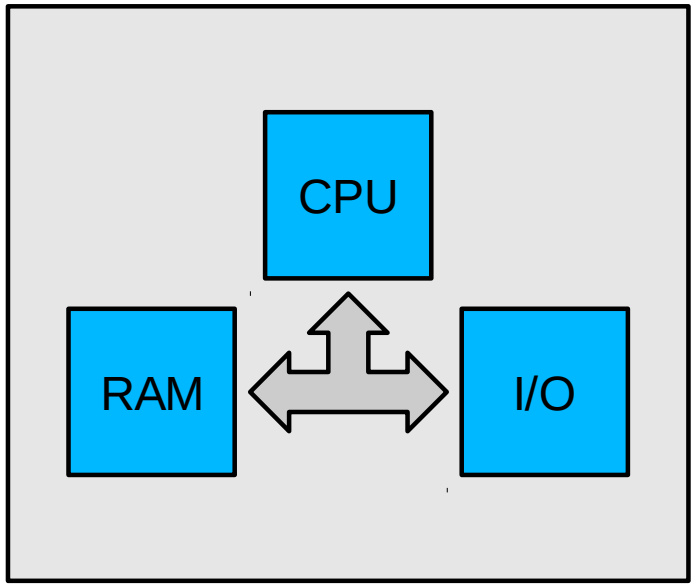
Befehlssatz

- einfache Befehle
  - RISC
  - CISC

Von-Neumann-Zyklus

1. FETCH
2. DECODE
3. FETCH OPERANDS
4. EXECUTE





1.2 Motivation

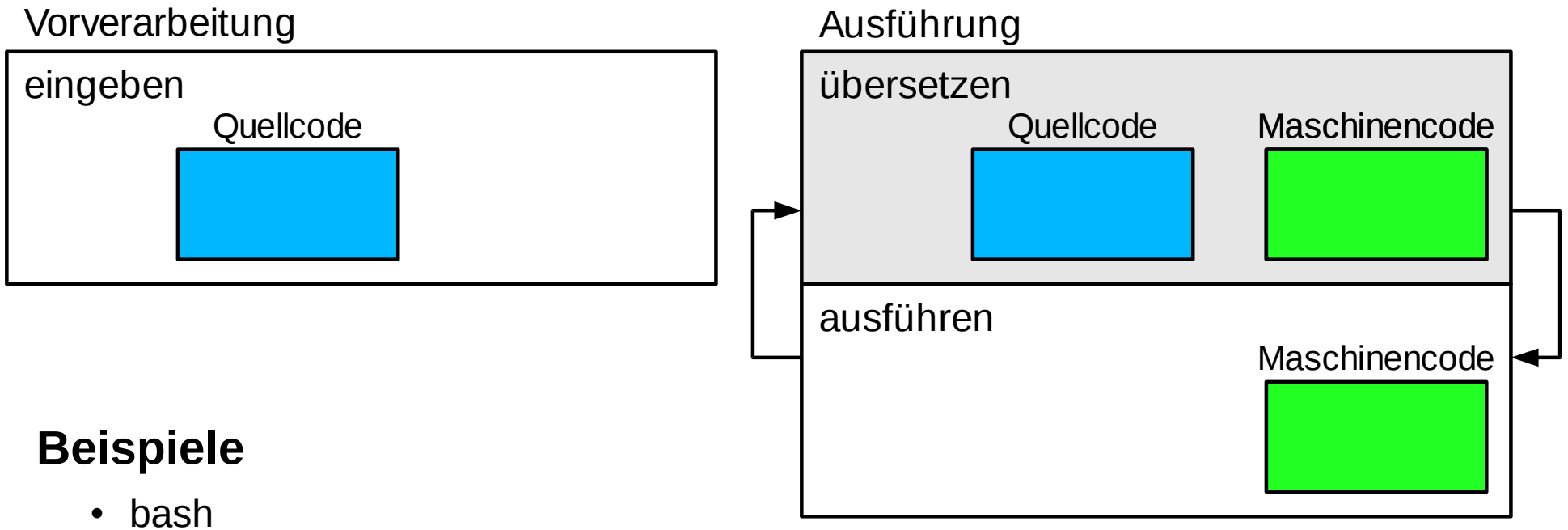
Problem

- Programm in Maschinencode
  - Codierung
  - unterschiedliche Befehlssätze

Lösungsansätze

- Assemblersprachen
- höhere Programmiersprachen
  - Interpreter
  - Compiler

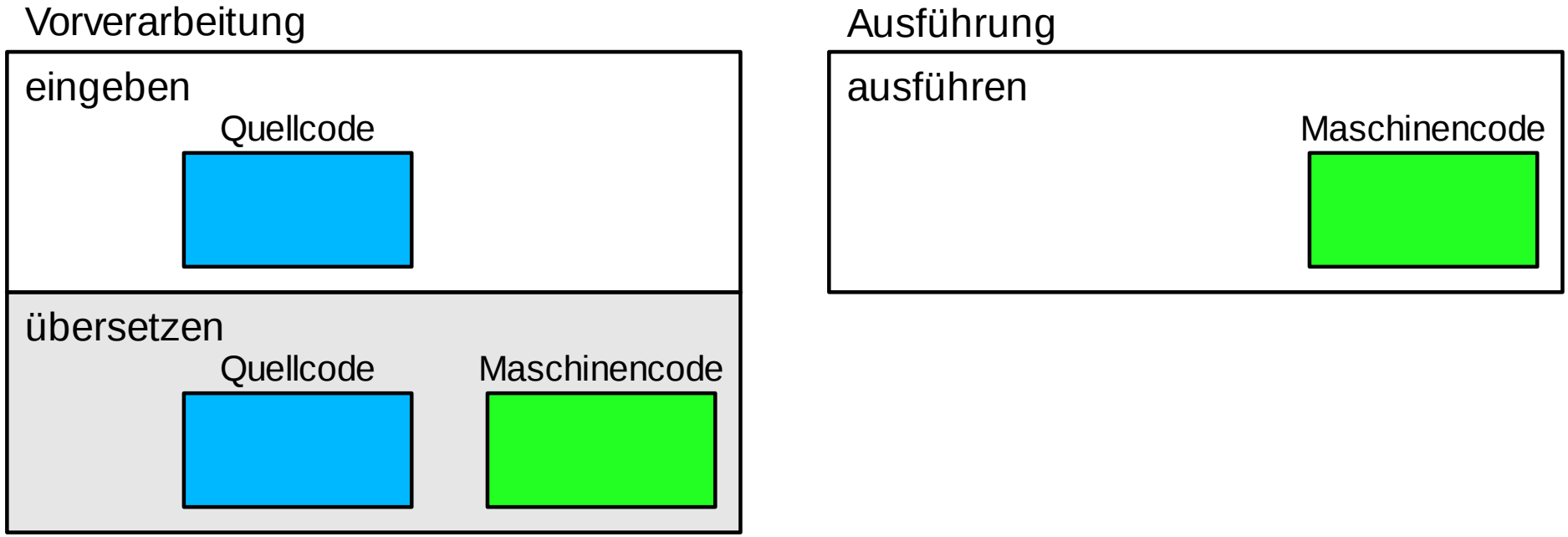
1.3 Interpreter



Beispiele

- bash

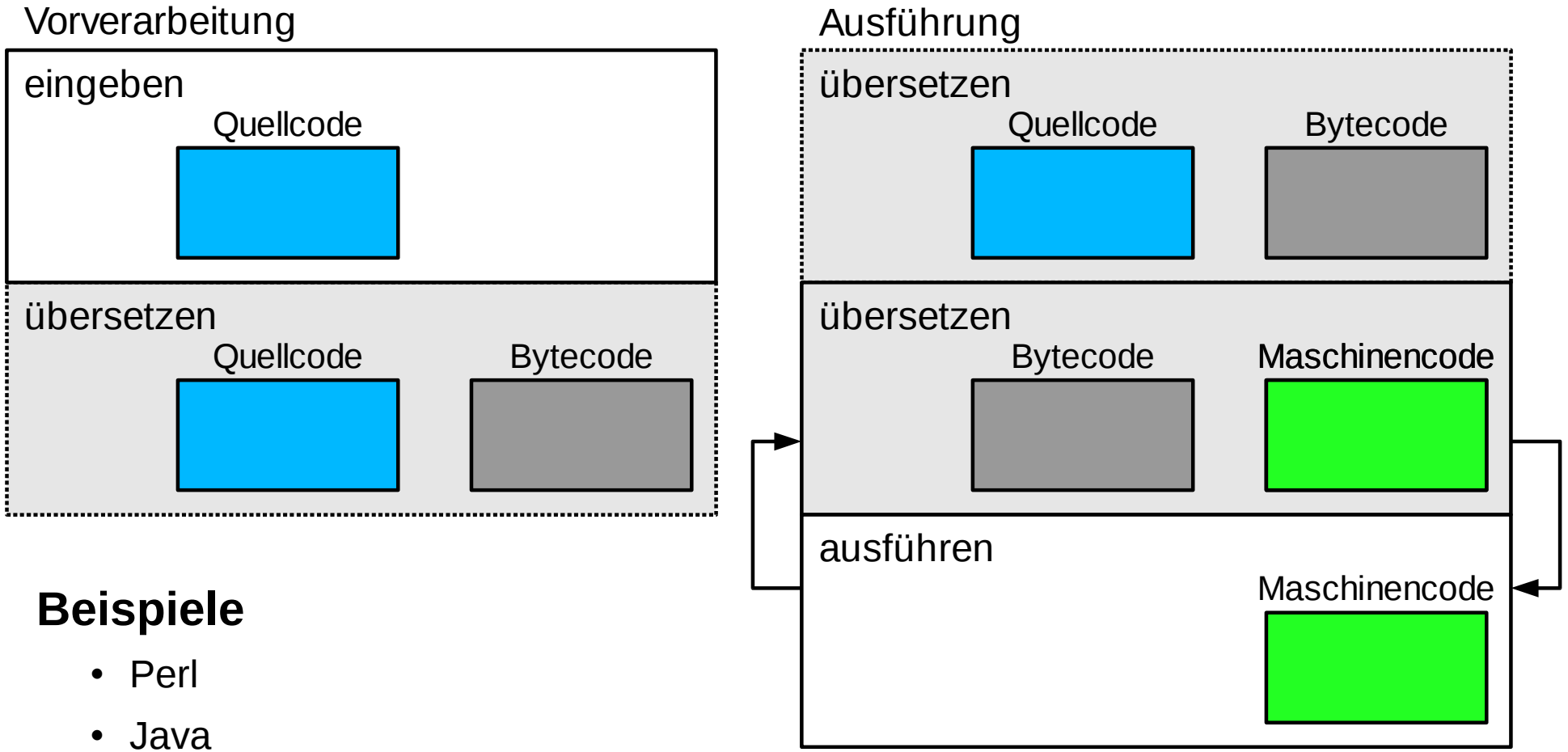
1.4 Compiler



Beispiele

- GNU Compiler Collection
  - C / C++
  - Fortran

1.5 Hybridcompiler



Beispiele

- Perl
- Java
- Python

2 Compiler / Linker

Bearbeitungsschritte

- Quellcode

Präprozessor

- modifizierter Quellcode

Compiler

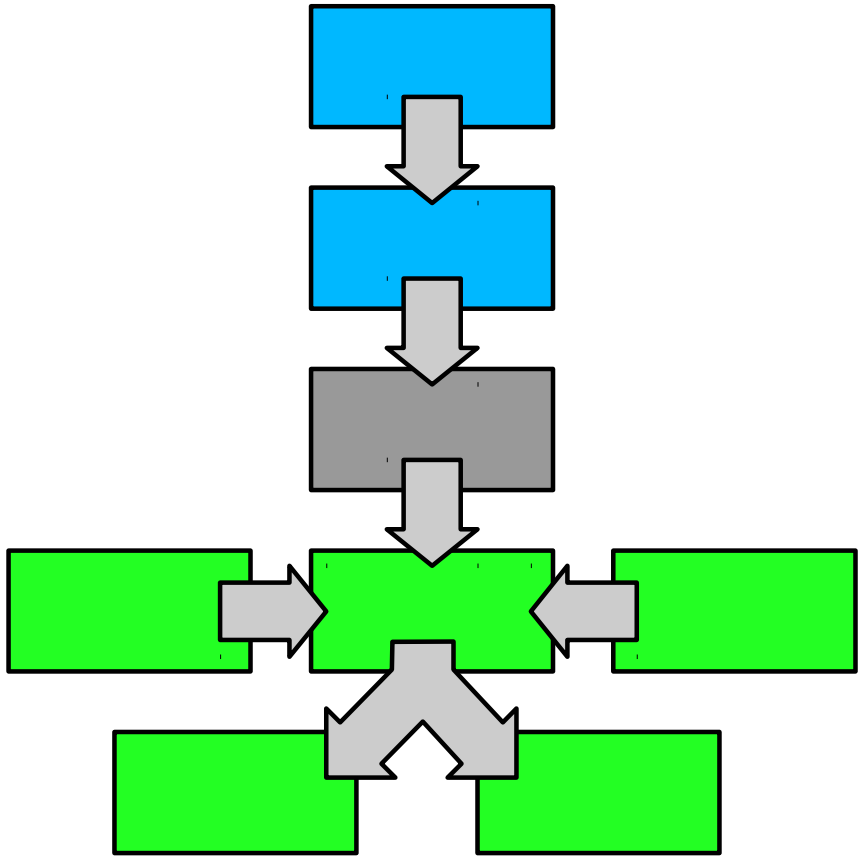
- Assemblercode

Assembler

- verschiebbarer Maschinencode

Linker

- Prozessabbild
- Shared Object



2.1 von der Quelle zur Objektdatei

Beispiel

main.c

```
#include <stdio.h>

int main (int argc, char* argv[])
{
  printf ("hello world.\n");
  return 0;
}
```

gcc -E main.c

```
...

extern int printf (__const char *__restrict __format, ...);
...

int main (int argc, char* argv[])
{
  printf ("hello world.\n");
  return 0;
}
```

gcc -S main.c

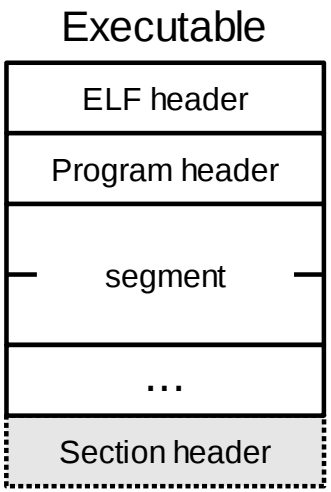
```
      .file "main.c"
      .section   .rodata
.LC0:
      .string    "hello world.\n"
      .text
.globl main
      .type main, @function
main:
      pushl %ebp
      movl  %esp, %ebp
      andl  $-16, %esp
      subl  $16, %esp
      movl  $.LC0, %eax
      movl  %eax, (%esp)
      call  printf
      movl  $0, %eax
      leave
      ret
      .size main, .-main
...
```

gcc -c main.c → main.o

2.2 Objektdatei

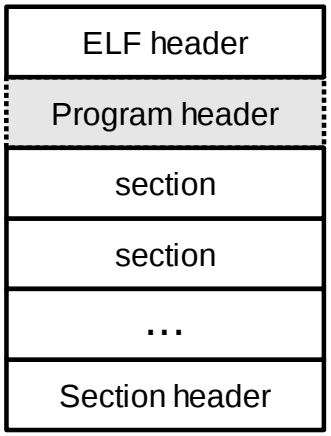
ELF – Executable and Linking Format

- executable
- relocatable object
- shared object



readelf -S main.o

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 00001d 00  AX  0   0  4
  [ 2] .rel.text         REL             00000000 000344 000010 08      9   1  4
  [ 3] .data             PROGBITS        00000000 000054 000000 00  WA  0   0  4
  [ 4] .bss              NOBITS          00000000 000054 000000 00  WA  0   0  4
  [ 5] .rodata           PROGBITS        00000000 000054 00000e 00   A  0   0  1
  [ 6] .comment          PROGBITS        00000000 000062 000024 01  MS  0   0  1
  [ 7] .note.GNU-stack   PROGBITS        00000000 000086 000000 00      0   0  1
  [ 8] .shstrtab         STRTAB          00000000 000086 000051 00      0   0  1
  [ 9] .symtab           SYMTAB          00000000 000290 0000a0 10     10   8  4
  [10] .strtab           STRTAB          00000000 000330 000014 00      0   0  1
```



2.2.1 Sektionen

Maschinencode

- .text
- ~~symbolische Namen~~
- Adressen

Relokation

- .rel.text

Symboltabelle

- .symtab
- definierte Symbole
- undefinierte Symbole

objdump -d main.o

```
Disassembly of section .text:
00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 10                sub    $0x10,%esp
   9:   b8 00 00 00 00          mov    $0x0,%eax
   e:   89 04 24                mov    %eax,(%esp)
  11:   e8 fc ff ff ff          call   12 <main+0x12>
  16:   b8 00 00 00 00          mov    $0x0,%eax
  1b:   c9                      leave
  1c:   c3                      ret
```

readelf -r main.o

```
Relocation section '.rel.text' at offset 0x344 contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0000000a  00000501 R_386_32          00000000   .rodata
00000012  00000902 R_386_PC32        00000000   printf
```

readelf -s main.o

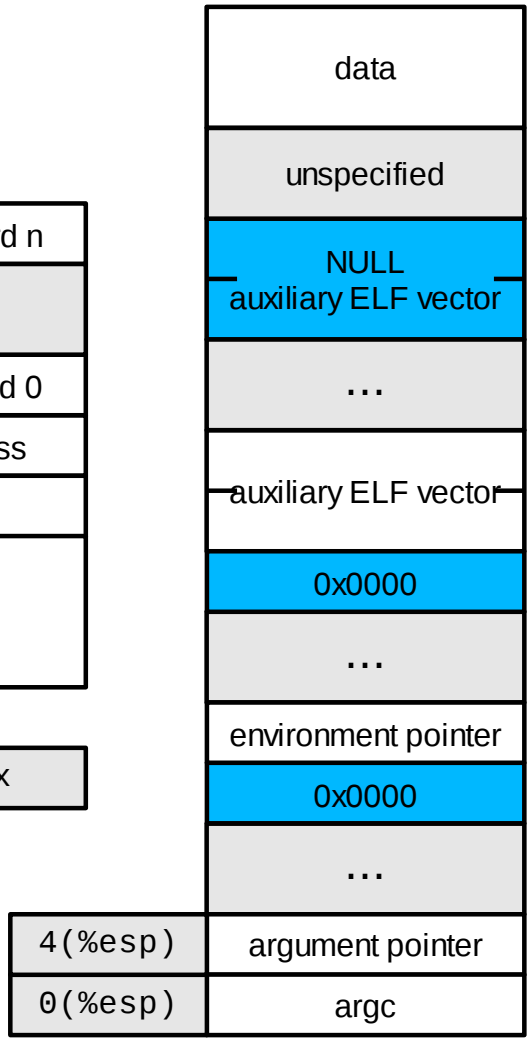
```
Symbol table '.symtab' contains 10 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    6
     8: 00000000    29 FUNC    GLOBAL DEFAULT    1 main
     9: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

2.3 Programmstart und Funktionsaufrufe

LSB / SysV ABI

n4+8(%ebp)	argument word n
...	...
8(%ebp)	argument word 0
4(%ebp)	return address
0(%ebp)	old %ebp
...	
0(%ebp)	

return value	%eax



start.s

```
     .text
.globl _start
     .type     _start, @function
_start:
        movl   %esp, %eax
        leal   4(%eax), %edx
        pushl  %edx
        pushl  %eax
        call   main
        pushl  %eax
        call  _exit
     .size     _start, .-_start
```

2.4 Systemaufrufe

%eax	Kennzahl
%ebx	Parameter 0
%ecx	Parameter 1
%edx	...
%esi	
%edi	

int 0x80
linux-gate.so.1

Rückgabewert	%eax

/usr/include/asm/unistd_32.h

```
 ...
#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
#define __NR_open                 5
#define __NR_close                6
#define __NR_waitpid              7
#define __NR_creat                8
 ...
```

end.s

```
      .text
.globl _exit
      .type exit, @function
_exit:
      pushl %ebp
      movl  %esp, %ebp
      movl  8(%ebp), %ebx
         movl  $0x01, %eax
         int   $0x80
      .size exit, .-exit
```

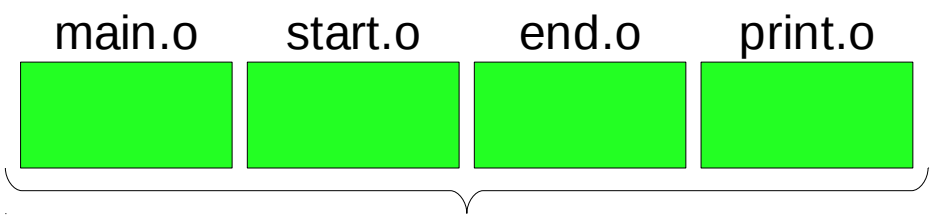
print.c

```
void put (char c)
{
  __asm__ ("movl $0x04, %%eax\n"
           "movl $0x01, %%ebx\n"
           "movl $01, %%edx\n"
           "int $0x80"
           :: "c" (&c)
           : "eax", "ebx", "edx");
}

void print (char* c)
{
  while (*(c) != '\0')
    put (*(c++));
}
```

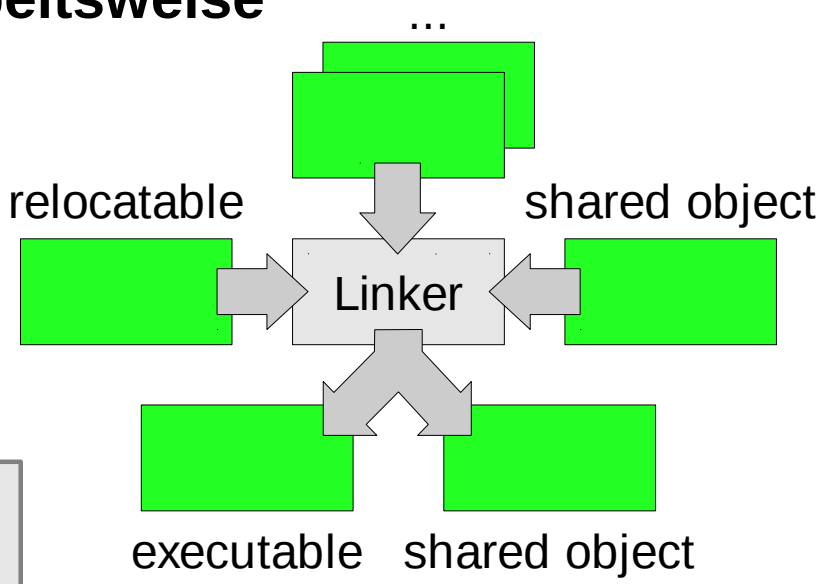
2.5 Linker

Beispiel




```
gcc -nostdlib main.o start.o end.o print.o -o hello
```

Arbeitsweise




```
readelf -l hello
```

```
Elf file type is EXEC (Executable file)
Entry point 0x80480d4
There are 3 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x08048000 0x08048000 0x00156 0x00156 R E 0x1000
  NOTE           0x000094 0x08048094 0x08048094 0x00024 0x00024 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4

 Section to Segment mapping:
  Segment Sections...
   00     .note.gnu.build-id .text .rodata
   01     .note.gnu.build-id
   02
```

2.6 statisch linken

Speicherlayout

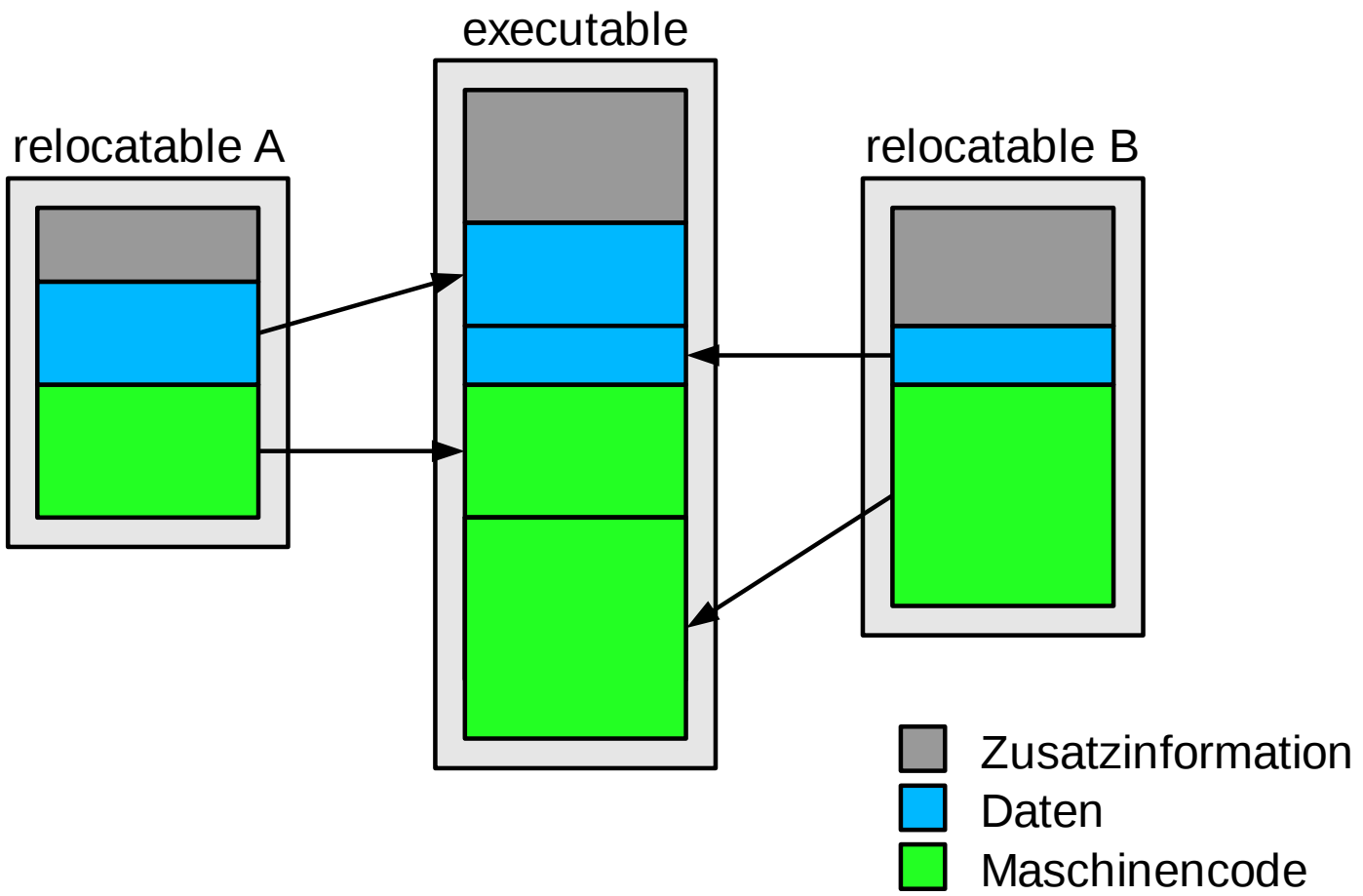
Segmente

- R/W .data ...
- R/E .text .rodata ...

Relokation

Addressen

- Sprünge
- Daten



2.7 in der Praxis

... ist es zum Glück viel einfacher :)

main.c

```
#include <stdio.h>

int main (int argc, char* argv[])
{
  printf ("hello world.\n");
  return 0;
}
```

gcc main.c

a.out

```
$ ./a.out
hello world.
$
```

2.8 GNU/Linux – Speicherverwaltung

physikalischer Speicher

- physikalische Adresse

virtueller Speicher

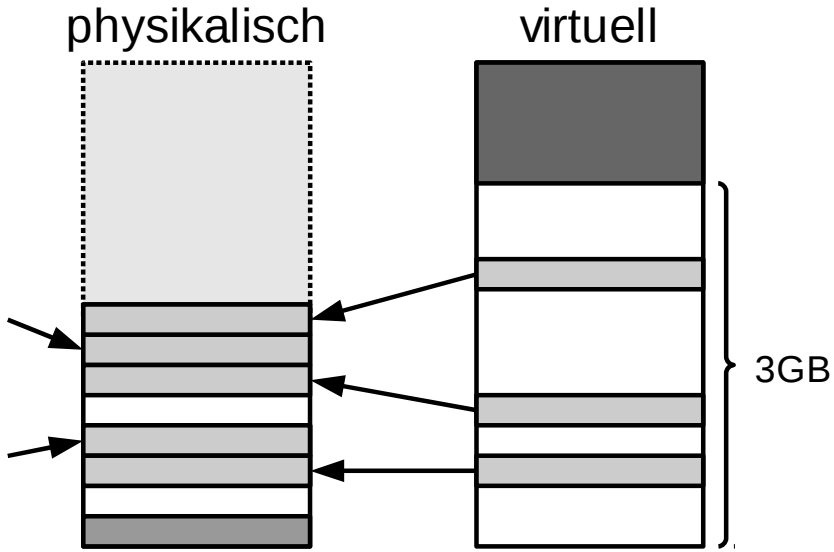
- virtuelle Adresse

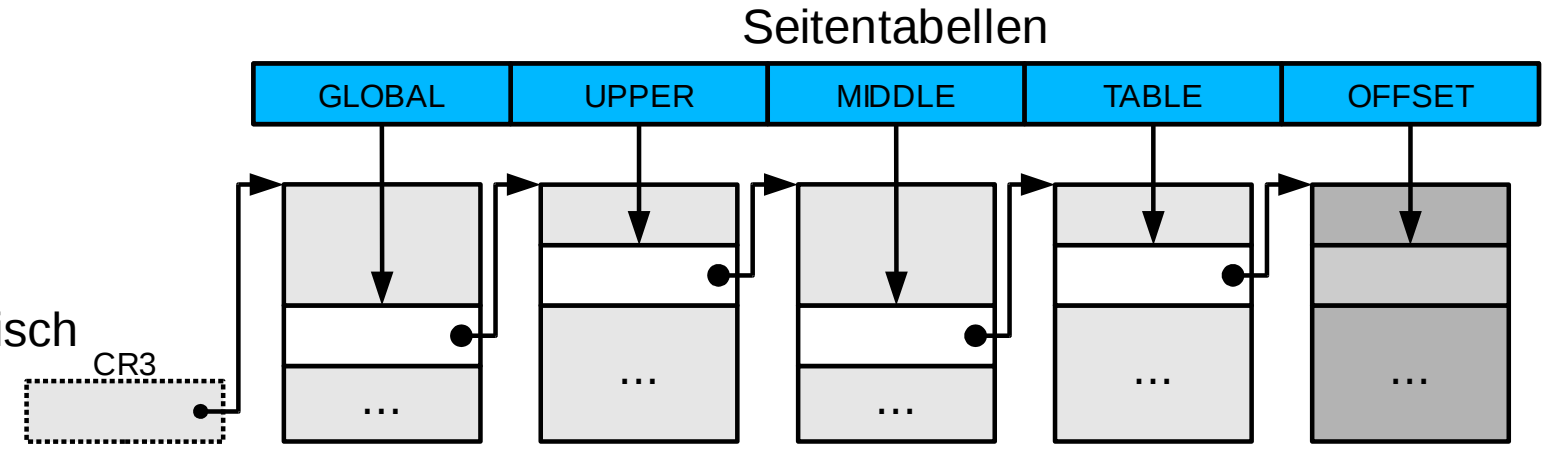
Memory Management Unit

- ~~Segmentierung~~
- Paging

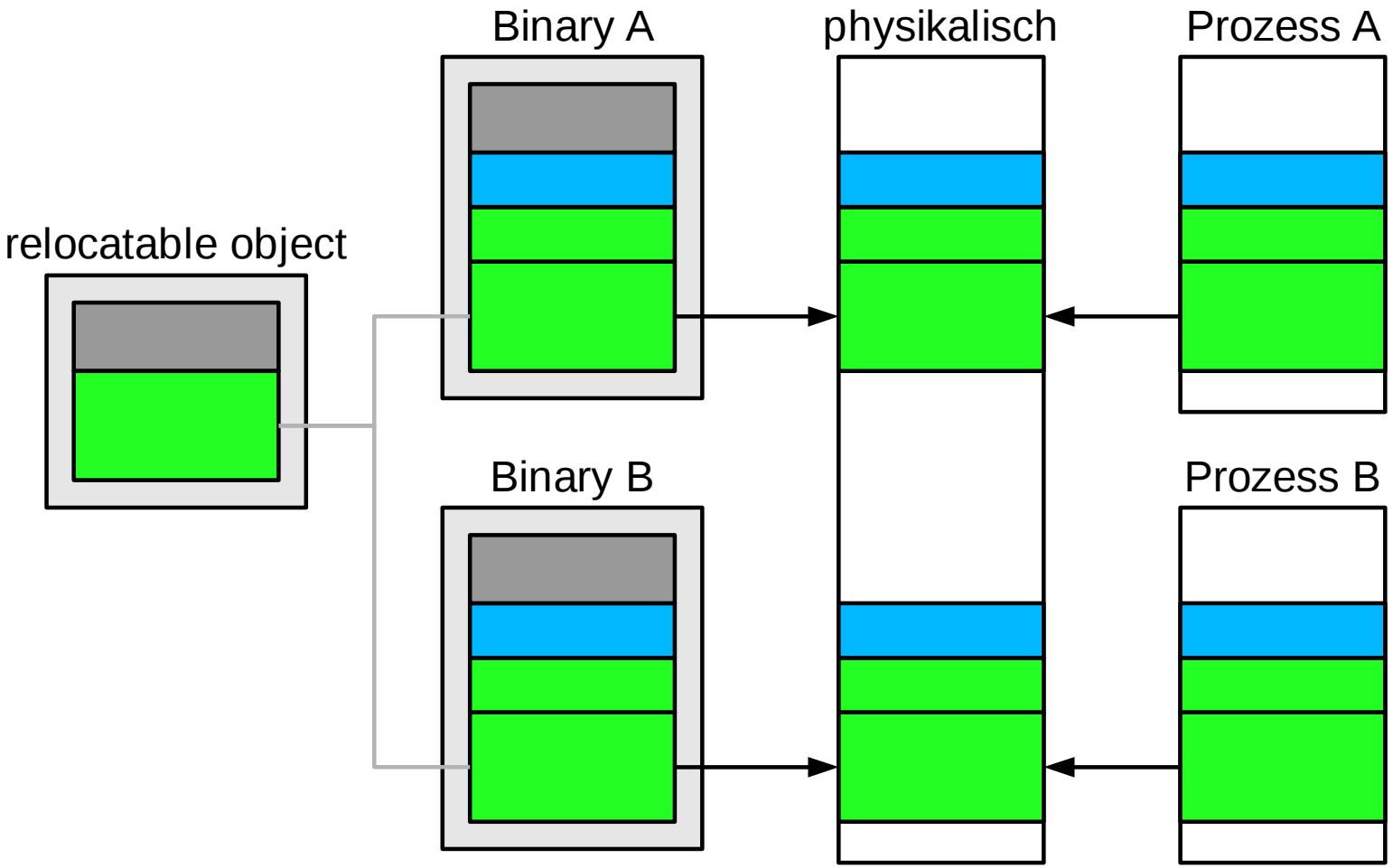
Seitentabelle

- Transformation
- virtuell - physikalisch

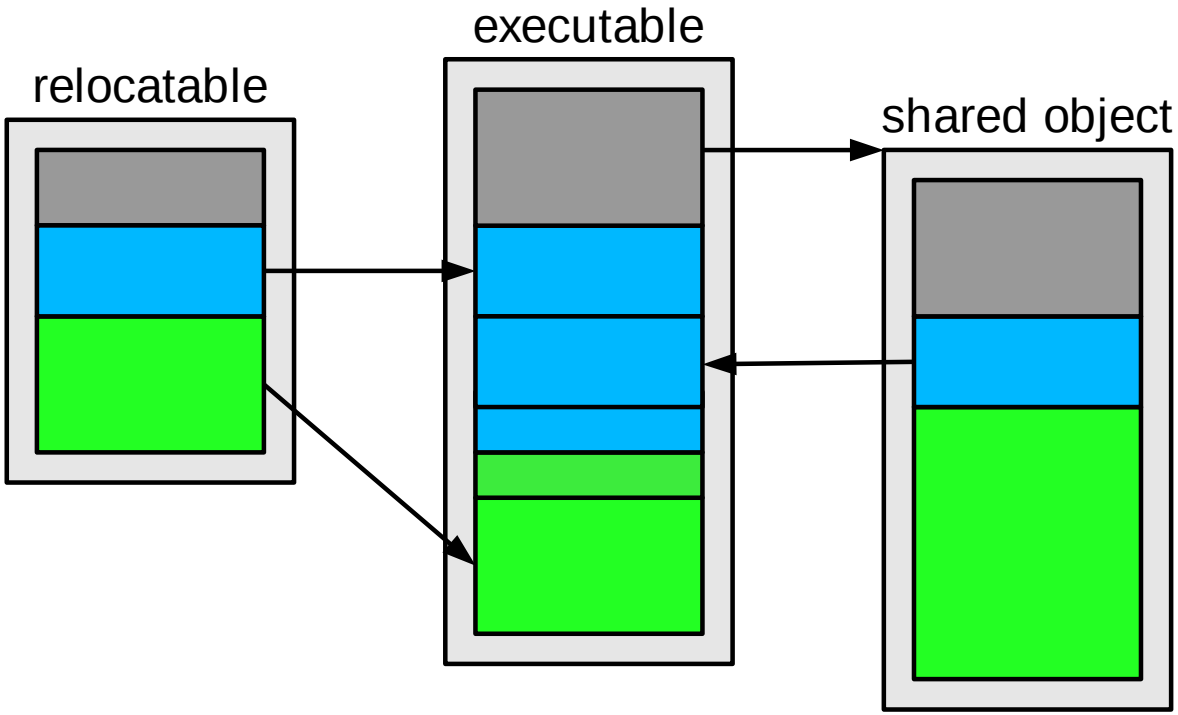




2.8.1 statisch - zur Ausführungszeit



2.8.2 dynamisch linken



2.8.3 Shared Objects

hello.c

```
#include <stdio.h>

void hello_world (void)
{
  printf ("hello world.\n");
}
```

```
gcc -c -fPIC hello.c

gcc -shared -Wl,-soname,libhello.so -o libhello.so
```

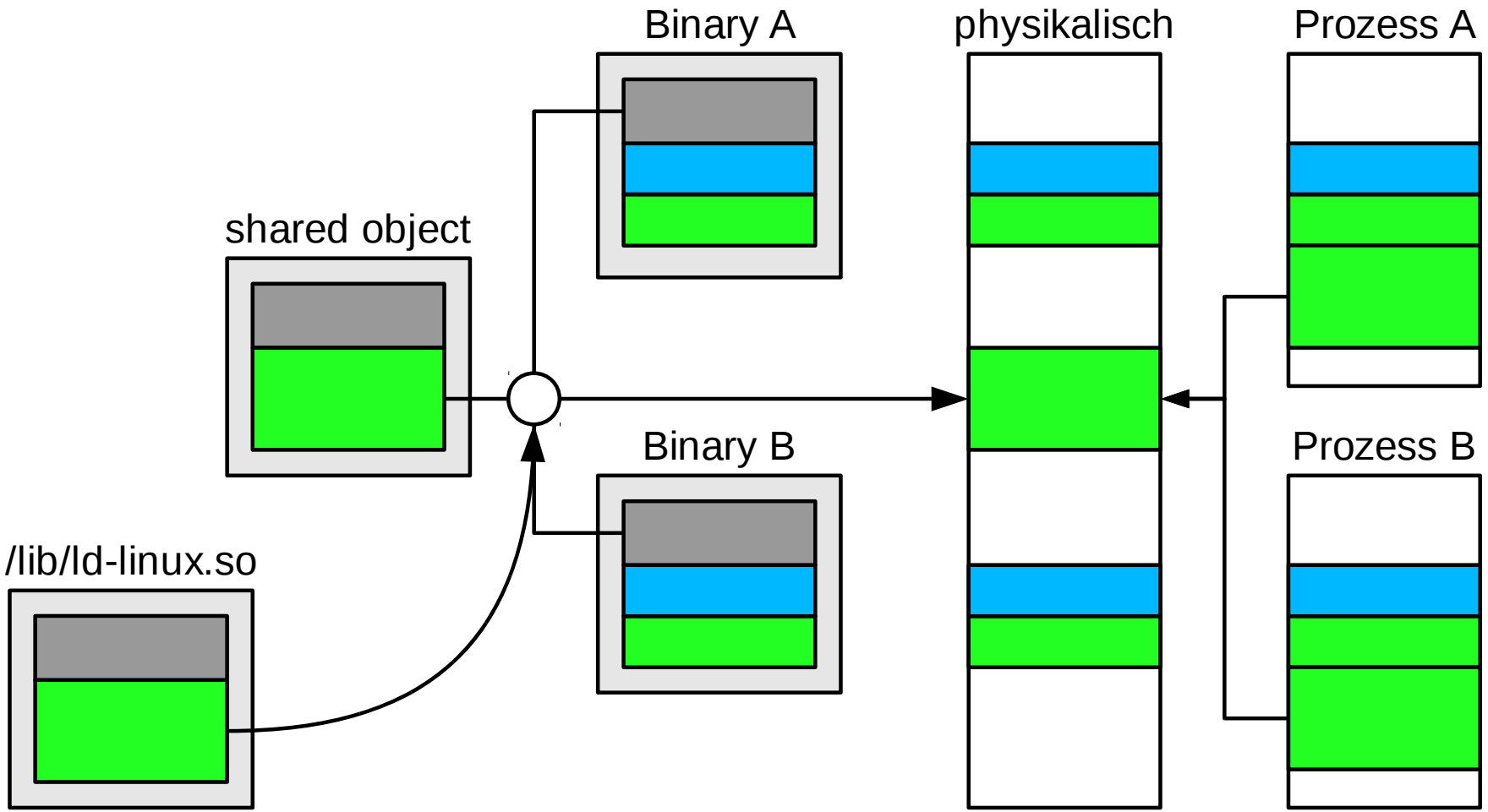
main.c

```
int main (int argc, char* argv[])
{
  hello_world ();
  return 0;
}
```

```
gcc -o hello -L. -lhello main.c
```

```
$ export LD_LIBRARY_PATH=$(pwd)
$ ./hello
hello world.
$
```

2.8.4 dynamisch – zur Ausführungszeit



2.9 dynamischer Linker

Cache

- /etc/ld.so.cache
- ldconfig

Environment

- LD_LIBRARY_PATH
- LD_BIND_NOW
- LD_DEBUG
- LD_PRELOAD

Details

- man ld.so
- man ldconfig

2.9.1 LD_PRELOAD

hello.c

```
#include <stdio.h>

void hello_world (void)
{
  printf ("hello world.\n");
}
```

main.c

```
int main (int argc, char* argv[])
{
  hello_world ();
  return 0;
}
```

myhello.c

```
#include <stdio.h>

void hello_world (void)
{
  printf ("my hello world.\n");
}
```

```
gcc -o myhello -shared myhello.c
```

```
$ LD_PRELOAD=myhello ./hello
my hello world.
$
```

2.9.2 dlopen

POSIX API

- öffnen / schließen

```
void *dlopen (char *filename, int flag)
int dlclose (void *handle)
```

- Symboladresse

```
void *dlsym (void *handle, char *symbol)
```

- Fehlermeldung

```
char *dlerror (void)
```

GNU Erweiterungen

```
#define _GNU_SOURCE
```

- Information

```
int dladdr (void *addr, Dl_info *info)
```

- Version

```
void *dlvsym (void *handle, char *symbol, char *version)
```

main.c

```
#include <dlfcn.h>

int main(int argc, char *argv[])
{
  void *handle;
  void (*fp) (void);

  handle = dlopen ("libhello.so", RTLD_LAZY);

  fp = dlsym (handle, "hello_world");
  fp ();

  dlclose  (handle);

  return 0;
}
```

3 Werkzeuge

verschiedene Komponenten bedienen...

Komponenten

- Präprozessor
- Compiler
- Assembler
- Linker

...und zwar möglichst nicht von Hand

Front-Ends

- GNU Compiler Collection
- GNU Make

3.1 GNU Compiler Collection

Compile/Link Manager

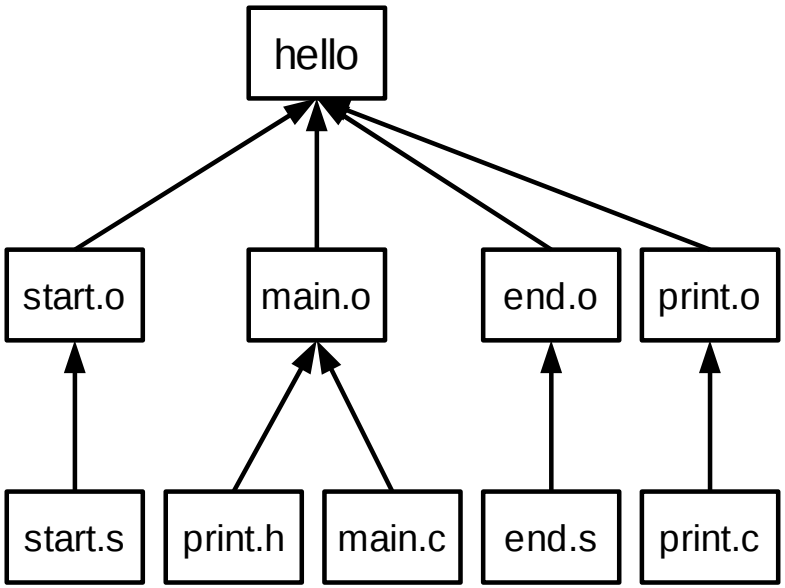
- Präprozessor
- Compiler
- Assembler
- Linker

specs

- gcc -dumpspecs

3.2 GNU make

Abhängigkeitsgraph



Makefile

```
hello: main.o start.o end.o print.o
     gcc -nostdlib main.o start.o end.o print.o -o hello

main.o: main.c print.h
     gcc -c main.c

print.o: print.c
     gcc -c print.c

start.o: start.s
     gcc -c start.s

end.o: end.s
     gcc -c end.s

clean:
     rm -rf main.o start.o end.o print.o
```

3.2.1 Makefile

Makros

- Variablen
- Funktionen

Regeln

- Explizite Regeln
- Musterregeln
- eingebaute Regeln

- Pseudoziele

```
C_SOURCES = main.c print.c
ASM_SOURCES = start.s end.s

OBJS = $(subst .c,.o, $(C_SOURCES))
OBJS += $(subst .s,.o, $(ASM_SOURCES))

PROGRAM = hello

$(PROGRAM): $(OBJS)
     gcc -nostdlib $(OBJS) -o $(PROGRAM)

main.o: main.c print.h

clean:
     rm -rf main.o start.o end.o print.o
```

```
%.o: %.c
     gcc -c $<

%.o: %.s
     gcc -c $<
```

4 C und Varianten

Erscheinungsjahr 1972!

- seitdem zahlreiche Dialekte

einige Varianten

- K&R C
- ANSI C / C89
- (C95)
- C99

- GNU C

```
auto        default     float       long        static      unsigned
break       do          for         register    struct      while
case        double      goto        return      switch
char        else        if          short       typedef
continue    extern      int         sizeof      union
```

```
const       enum        signed      void        volatile
```

```
inline      long long   restrict    _Bool       _Complex    _Imaginary
```

4.1 K&R C

Deklaration von Funktionen

- ohne Parameter

```
power ();
```

Blockkommentar

```
/* don't use with n negative!
   ...
*/
```

Deklaration von Funktionsparametern

- separat vor der öffnenden Klammer
- nicht deklarierte Paramter (= int)

```
power (b, n)
int b;
```

Deklaration von Variablen

- nur am Anfang eines Blocks

```
{
  int result;
  int i;

  result = 1;
  for (i=0; i<n; ++i)
    result *= b;
  return result;
}
```

4.2 ANSI C

Deklaration von Funktionen

- mit Typangabe

```
int power (int, int);
```

Blockkommentar

```
/* don't use with n negative!
   ...
*/
```

Deklaration von Funktionsparametern

- direkt vor Ort

```
int power (int b, int n)
```

Deklaration von Variablen

- nur am Anfang eines Blocks

```
{
  int result;
  int i;

  result = 1;
  for (i=0; i<n; ++i)
    result *= b;
  return result;
}
```

4.3 C99

Deklaration von Funktionen

- unverändert

```
int power (int, int);
```

Kommentarzeilen

- zusätzlich erlaubt

```
// don't use with n negative!
// ...
```

Deklaration von Funktionsparametern

- unverändert

```
int power (int b, int n)
```

Deklaration von Variablen

- muss nicht am Anfang eines Blocks erfolgen

```
{
  int result;
  result = 1;

  for (int i=0; i<n; ++i)
    result *= b;
  return result;
}
```

5 Bibliotheken und Header

Typinformation

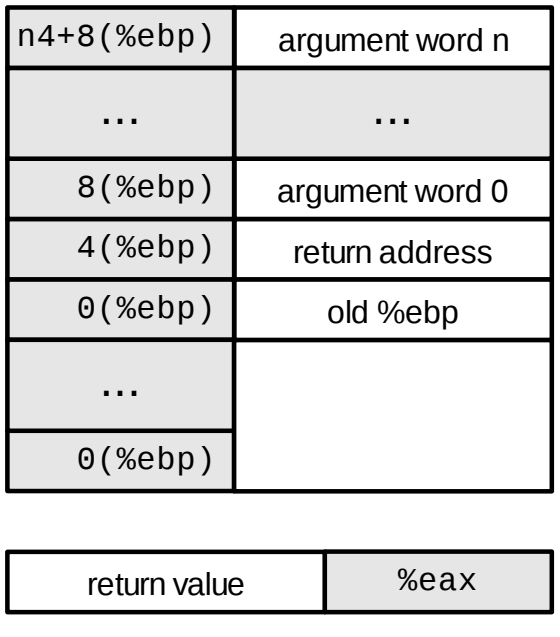
- zur Übersetzungszeit

```
float add (float a, float b)
{
  return a + b;
}
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  printf ("%d + %d = %d\n", 1, 2, add (1, 2));
  return 0;
}
```

```
$ ./a.out
1 + 2 = -1081551932
$
```



5.1 …wie machen das die anderen?

Typinformation

- zur Laufzeit

Beispiel: Python

```
...
typedef struct {
    PyObject_HEAD
    long ob_ival;
} PyIntObject;
```

```
...
#define PyObject_HEAD                   \
     _PyObject_HEAD_EXTRA               \
     Py_ssize_t ob_refcnt;              \
     struct _typeobject *ob_type;
```

```
...
#define _PyObject_HEAD_EXTRA
```

5.2 C Standard Library

von unverzichtbar ...

- <stdio.h>
  - Ein-/Ausgabe, öffnen / schließen von Dateien
- <errno.h>
  - Fehlerbehandlung

<stdlib.h>
- Speicherverwaltung
- <string.h>
  - Zeichenketten aber auch! memmove

... bis exotisch

- <stdarg.h>
  - variable Argumentenlisten
- <setjmp.h>
  - nicht lokale Sprünge

5.3 POSIX

I/O

Prozessverwaltung

IPC

Regular Expressions

6 Speicherverwaltung

statischer Speicher

- statische / globale variablen
- feste Zuweisung im Adressraum

automatischer Speicher

- automatische Variablen
- auf dem Stack

dynamischer Speicher

- malloc () / realloc ()
- free ()
- alloca ()

6.1.1 Vorsicht, Falle!

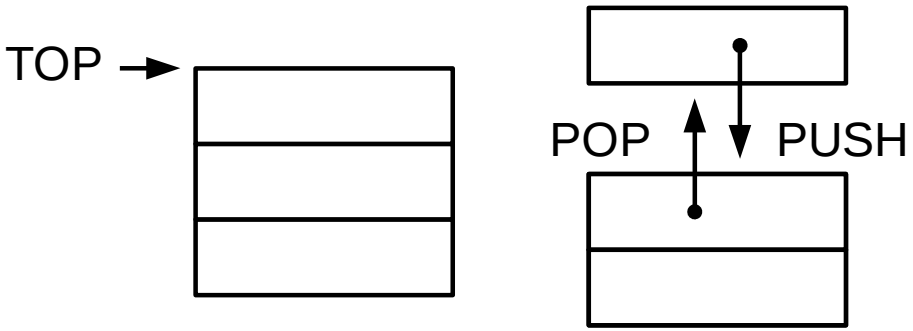
```
typedef struct _stack_item {
  struct _stack_item *prev;
  void *data;
} StackItem;

typedef struct _stack {
  StackItem* top;
} Stack;

void push (Stack *stack, void *data)
{
  StackItem item;
  item.prev = stack->top;
  item.data = data;
  stack->top = &item;
}

void* pop (Stack *stack)
{
  void *data = NULL;
  StackItem *top = stack->top;

  if (stack->top)
    {
      data = stack->top->data;
      stack->top = stack->top->prev;
      free (top);
    }
  return data;
}
```




```
#include <stdio.h>
#include <stdlib.h>

...

int main(int argc, char *argv[])
{
  Stack S = { NULL };
  char *string;

  push (&S, "first item");
  push (&S, "second item");
  push (&S, "third item");

    while ((string = pop (&S)) != NULL)
      printf ("%s\n", string);

  return 0;
}
```

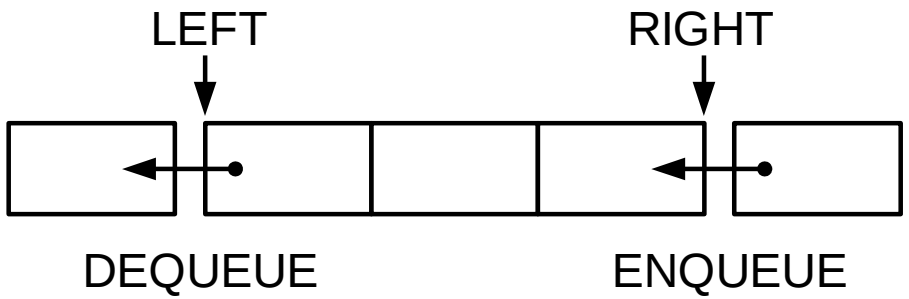
6.2 …schlimmer …geht's immer!

```
typedef struct _queue_item{
  struct _queue_item *prev;
  void *data;
} QueueItem;

typedef struct {
  QueueItem *left;
  QueueItem *right;
} Queue;

void enqueue (Queue *queue, void* data)
{
  QueueItem *item = malloc (sizeof (QueueItem));
  item->data = data;
  item->prev = NULL;
  if (queue->right)
    queue->right->prev = item;
  else
    queue->left = item;
  queue->right = item;
}

void* dequeue (Queue *queue)
{
  void *data = NULL;
  if (queue->left)
    {
      data = queue->left->data;
      queue->left = queue->left->prev;
      if (!queue->left)
        queue->right = NULL;
    }
  return data;
}
```




```
#include <stdio.h>
#include <stdlib.h>


...

int main(int argc, char *argv[])
{
  Queue Q = { NULL, NULL };
  char *string;

  while (1)
    {
      enqueue (&Q, "first item");
      string = dequeue (&Q);
    }

  return 0;
}
```

7 POSIX – Regular Expressions

POSIX

- BRE (= Basic Regular Expressions)
  - `\( \)  \{m,n\}`
- ERE (= Enhanced Regular Expressions)
  - `( )  {m,n} + ? |`
- gemeinsam
  - `.  *  ^ $  []`

nicht verwechseln mit

- Wildcard-Matching
  - `fnmatch()`
- Globbing
  - `glob(), globfree()`

7.1 Reguläre Ausdrücke

Datentyp

```
regex_t
        .re_nsub     # Sub-Expressions  (von regcomp() ausgefüllt)
```

Funktionen

- Erzeugen (= „Übersetzen“)
  - `regcomp (regex_t *preg, const char *regex, int cflags)`
- Freigeben
  - `regfree (regex_t *preg)`

Flags

- `REG_EXTENDED, REG_NOSUB, …`
- Details
  - man regcomp
  - info libc

7.2 Treffer

Datentyp

`regmatch_t`

`.rm_so`	Start-Offset (-1 für keinen Treffer)
`.rm_eo`	End-Offset

Funtionen

- „Match“
  - `regexec (regex_t *preg, const char *string, size_t nmatch, regmatch_t pmatch[], int eflags)`
- Treffer 0 – kompletter Ausdruck
- Treffer i – i-te sub-expression

Flags

- siehe manual-pages / libc info Seiten

7.3 Beispiel

```
int main(int argc, char *argv[])
{
  int line = 0;
  char *string = NULL;
  int length = 0;
  int n;
  regex_t regex;
  regmatch_t *match;

  if (argc != 2)
    exit (EXIT_FAILURE);

  regcomp (&regex, argv[1], REG_EXTENDED);
  n = regex.re_nsub + 1;

 match = malloc (n * sizeof (regmatch_t));

  while (getline (&string, &length, stdin) != -1)
    {
      char *pos = string;
      while (!regexec (&regex, pos, n, match, 0))
        {
          int i;
          printf ("Treffer in Zeile %d:", line);
          for (i=0; i < n; ++i)
            print_match (pos, match[i]);
          printf ("\n");
          pos += match[0].rm_eo;
        }
      free (string); string = NULL; length = 0;
      ++line;
    }
  free (match); regfree (&regex); return 0;
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <regex.h>
#include <stdlib.h>

void print_match (char *string, regmatch_t match)
{
  printf (" \"%.*s\"", match.rm_eo - match.rm_so,
          string + match.rm_so);
}
```

```
$ ./match "(\w*a\w*)(n\w*)" <<EOF
> Bei Banane und Ananas
> kann man Treffer finden.
> EOF
Treffer in Zeile 0: "Banane" "Bana" "ne"
Treffer in Zeile 0: "Ananas" "Ana" "nas"
Treffer in Zeile 1: "kann" "kan" "n"
Treffer in Zeile 1: "man" "ma" "n"
$
```

8 Fragen …