Softwareentwicklung für eingebettete Systeme

Bachelorstudiengang Technische Informatik
Hochschule Pforzheim

Vorlesung 1
Sommersemester 2014

Dipl.-Ing.(FH) Marc Jüttner

Inhalt der Vorlesung

- Toolchains, Objektdateien, Buildkonzepte
- Systemkonzepte
- Grafische Benutzerschnittstellen
- Anwendung der Systemarchitektur
  - DMA, IPC
- Socketprogrammierung
- Deploymentmechanismen

Grundsätzliches

- Interaktive Vorlesung bevorzugt
- Fragen sofort stellen!
- Verständnis des Inhalts steht im Vordergrund

Termine

- Mittwochs 8.00 – 11.15
- Termine können sich kurzfristig verschieben
  - Bitte Aushang beachten!
- Sechs bis sieben Vorlesungseinheiten

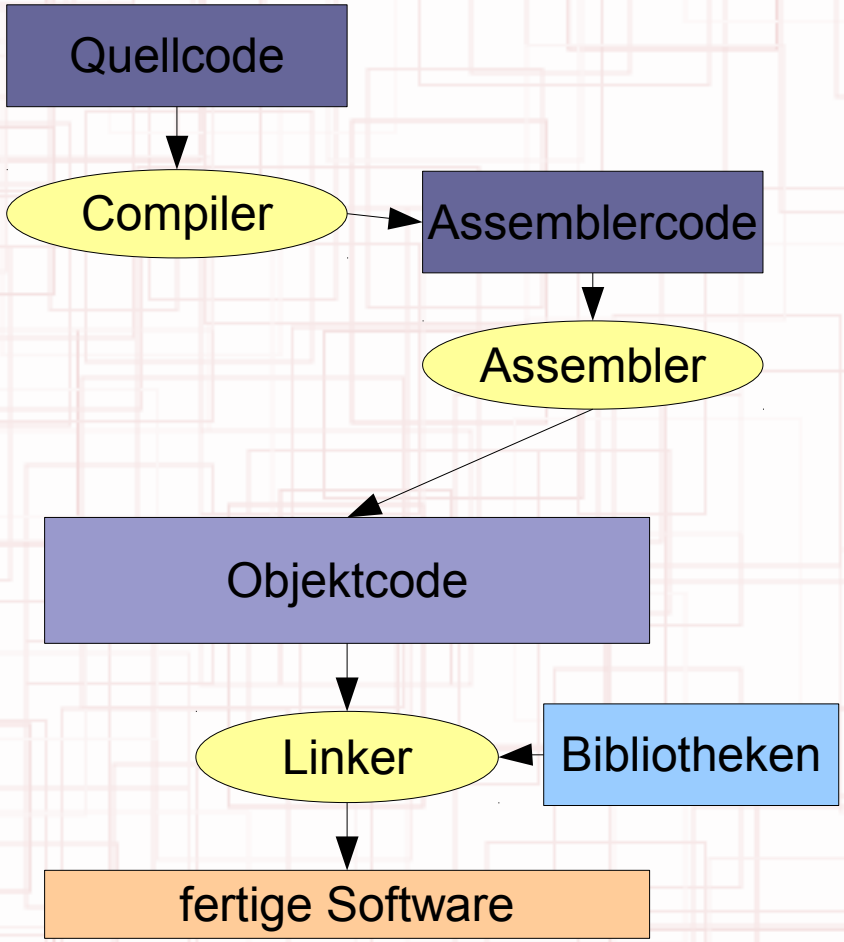
Skript

- Es gibt kein Skript
- Vorlesungsfolien werden im Internet bereitgestellt
  - marcjuettner.de/cms
  - Registrierung erforderlich
  - Links, Informationen und Literaturliste

Literatur

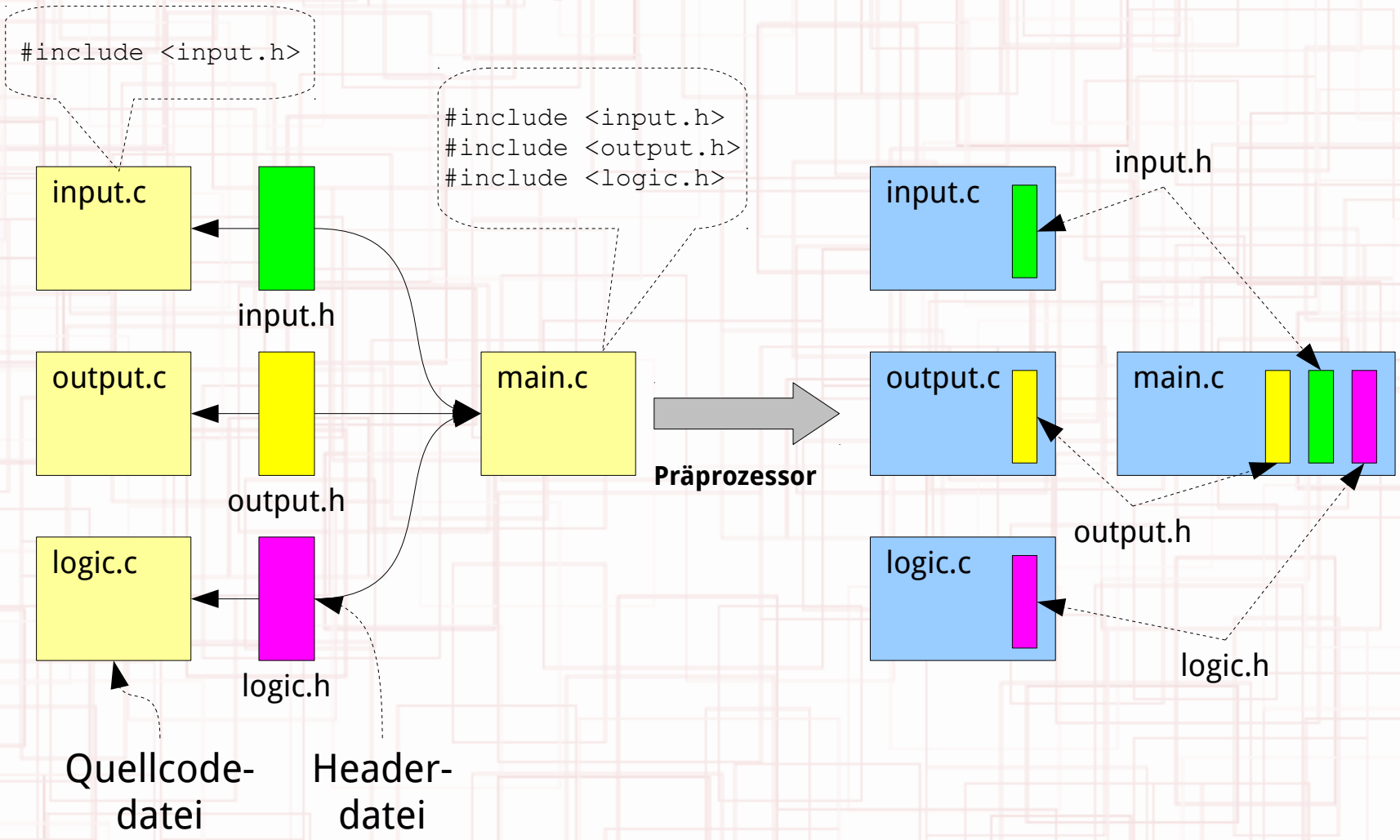
- C und C++ für Embedded Systems, Friedrich Bollow/Matthias Homann/Klaus-Peter Köhn, mitp 2009
- Embedded Systems, Jack Ganssle (Ed.), Newnes/Elsevier 2008
- Embedded Software, Jean Labrosse, Newnes/Elsevier 2008
- Internet...

Softwareübersetzung

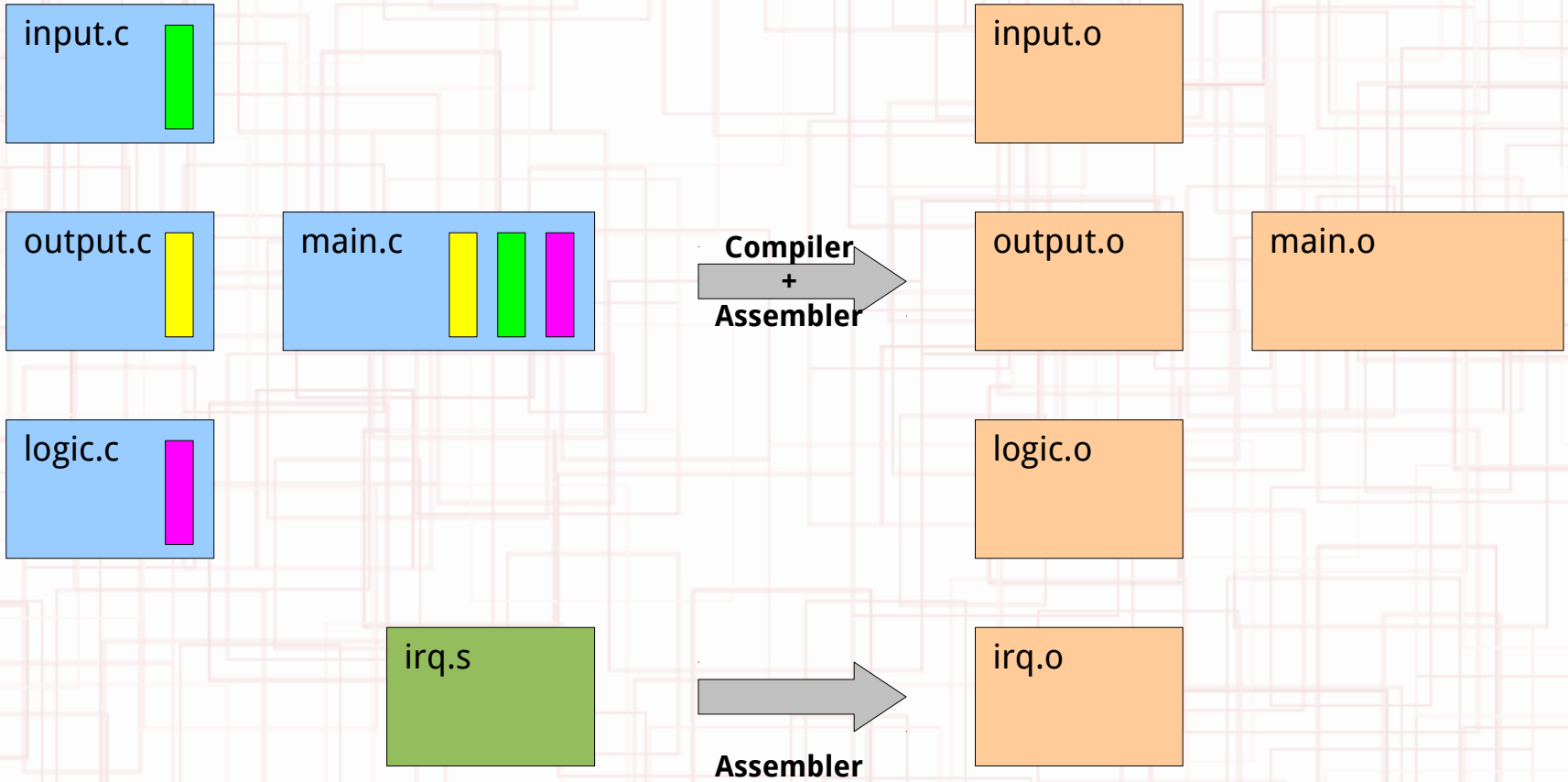


- Compiler erzeugt Assemblercode aus Quellcode
- Assembler erzeugt Objektcode aus Assemblercode
- Linker fügt Objekte zusammen

Präprozessor



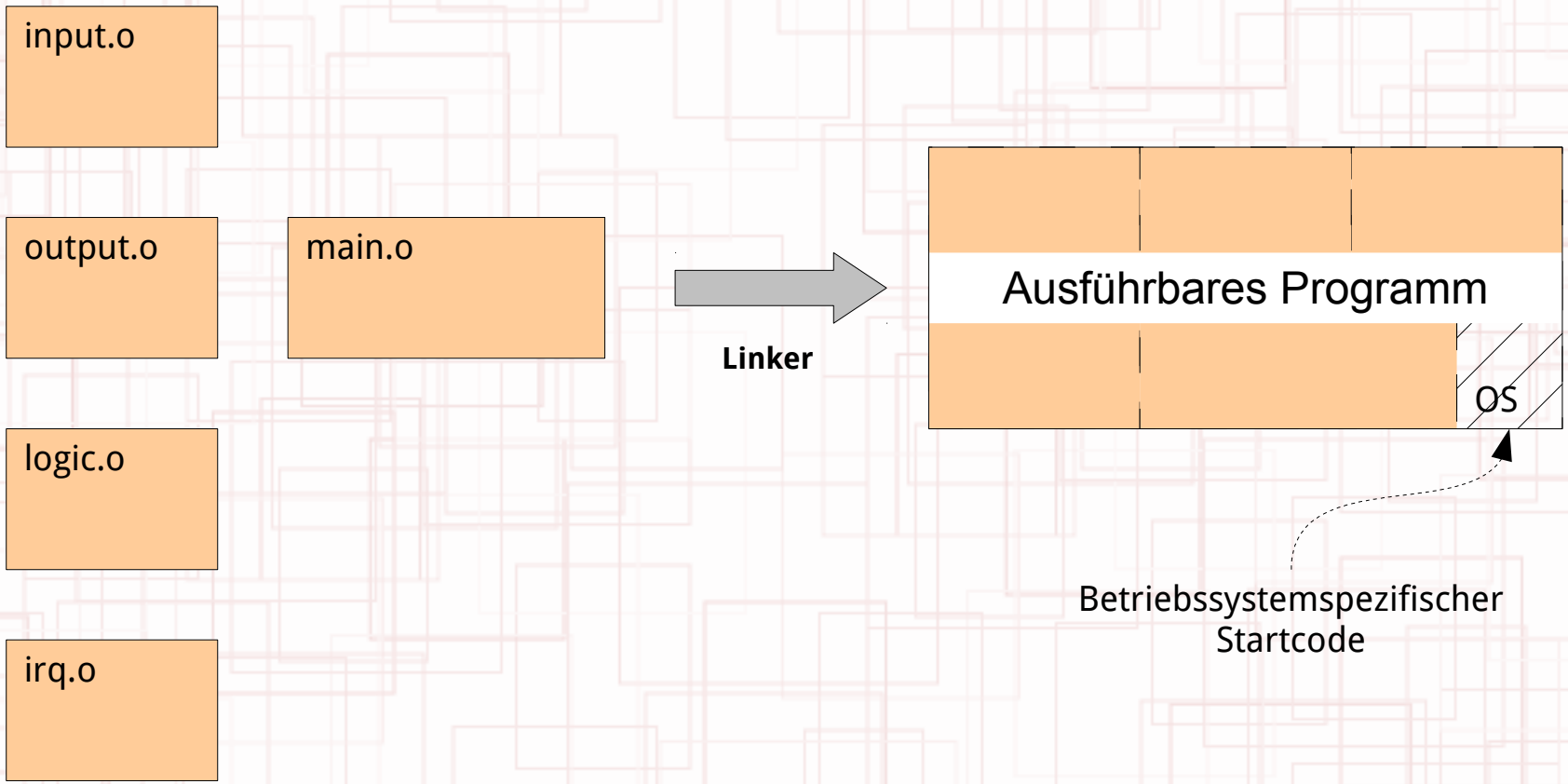
Compiler und Assembler



Objektdateien

- Vorwiegend Maschinencode
- Linkerinformationen
  - Vorinitialisierte Variablen
  - Konstanten (Strings)
  - Debuginformationen
  - Symbole
    - Import: Referenzierte Symbole
    - Export: Exportierte Symbole

Linker


input.o
output.o
main.o
logic.o
irq.o
Linker
Ausführbares Programm
OS
Betriebssystemspezifischer
Startcode

Objektsegmente

- Header: Beschreibung und Steuerung
- Textsegment: Ausführbarer Code
- Datensegment: Statische Daten
- BSS-Segment: uninitialisierte statische Daten
- Externe Definitionen und Referenzen

Beispiel

```c
#include "printstring.h"

const int Const_Int = 1;
static int Static_Int_Uninitialized;
static int Static_Int = 2;
static const int Static_Const_Int = 3;

const char string[] = "Teststring";

void main(void)
{
    printstring(string);
}
```

sample.c

```c
#if !defined _PRINTSTRING_H
#define _PRINTSTRING_H

void printstring(const char *s);

#endif
```

printstring.h

```c
#include <stdio.h>
#include "printstring.h"

void printstring(const char *s)
{
    printf("%s\n", s);
}
```

printstring.c

```
$ gcc sample.c printstring.c -o sample
```

printstring.o

```
$ objdump -t printstring.o

printstring.o:     Dateiformat elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*              0000000000000000 printstring.c
0000000000000000 l    d  .text              0000000000000000 .text
0000000000000000 l    d  .data              0000000000000000 .data
0000000000000000 l    d  .bss               0000000000000000 .bss
0000000000000000 l    d  .note.GNU-stack    0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame          0000000000000000 .eh_frame
0000000000000000 l    d  .comment           0000000000000000 .comment
0000000000000000 g     F .text              000000000000001a printstring
0000000000000000               *UND*         0000000000000000 puts
```

Symboltabelle 'sample'

```
$ objdump -t sample

sample:     Dateiformat elf64-x86-64

DYNAMIC SYMBOL TABLE:
0000000000000000      DF *UND*  0000000000000000  GLIBC_2.2.5 __libc_start_main
0000000000000000      DF *UND*  0000000000000000  GLIBC_2.2.5 puts

SYMBOL TABLE:
0000000000000000       F *UND*  0000000000000000 __libc_start_main@@GLIBC_2.2.5
0000000000000000       F *UND*  0000000000000000 puts@@GLIBC_2.2.5
...
0000000000000000 l    df *ABS*  0000000000000000 printstring.c
0000000000000000 l    df *ABS*  0000000000000000 sample.c
...
0000000000400440 g     F .text  0000000000000000 _start
0000000000400440 l    d  .text  0000000000000000 .text
...
000000000040052d g     F .text  0000000000000010 main
000000000040053d g     F .text  000000000000001a printstring
...
00000000004005f4 g     F .fini  0000000000000000 _fini
00000000004005f4 l    d  .fini  0000000000000000 .fini
```

Symboltabelle 'sample'

```
0000000000400600 g     O .rodata 0000000000000004 _IO_stdin_used
0000000000400600 l    d  .rodata 0000000000000000 .rodata
0000000000400604 g     O .rodata 0000000000000004 Const_Int
0000000000400608 l     O .rodata 0000000000000004 Static_Const_Int
000000000040060c g     O .rodata 000000000000000b string
...
0000000000601030 g       .data   0000000000000000 __data_start
0000000000601030 l    d  .data   0000000000000000 .data
0000000000601030  w      .data   0000000000000000 data_start
0000000000601038 g     O .data   0000000000000000 .hidden __dso_handle
0000000000601040 l     O .data   0000000000000004 Static_Int
0000000000601044 g       .data   0000000000000000 _edata
0000000000601044 g       .bss    0000000000000000 __bss_start
0000000000601044 l    d  .bss    0000000000000000 .bss
0000000000601044 l     O .bss    0000000000000001 completed.6992
0000000000601048 l     O .bss    0000000000000004 Static_Int_Uninitialized
0000000000601050 g       .bss    0000000000000000 _end
```

Bedeutung der Flags

```
The flag characters are divided into 7 groups as follows:

l, g, u, !
     The symbol is a local (l), global (g), unique global (u), neither global
     nor local (a space) or both global and local (!).
w
     The symbol is weak (w) or strong (a space).
C
     The symbol denotes a constructor (C) or an ordinary symbol (a space).
W
     The symbol is a warning (W) or a normal symbol (a space). A warning
     symbol's name is a message to be displayed if the symbol following the
     warning symbol is ever referenced.
I, i
     The symbol is an indirect reference to another symbol (I), a function to be
     evaluated during reloc processing (i) or a normal symbol (a space).
d, D
     The symbol is a debugging symbol (d) or a dynamic symbol (D) or a normal
     symbol (a space).
F, f, O
     The symbol is the name of a function (F) or a file (f) or an object (O) or
     just a normal symbol (a space).
```

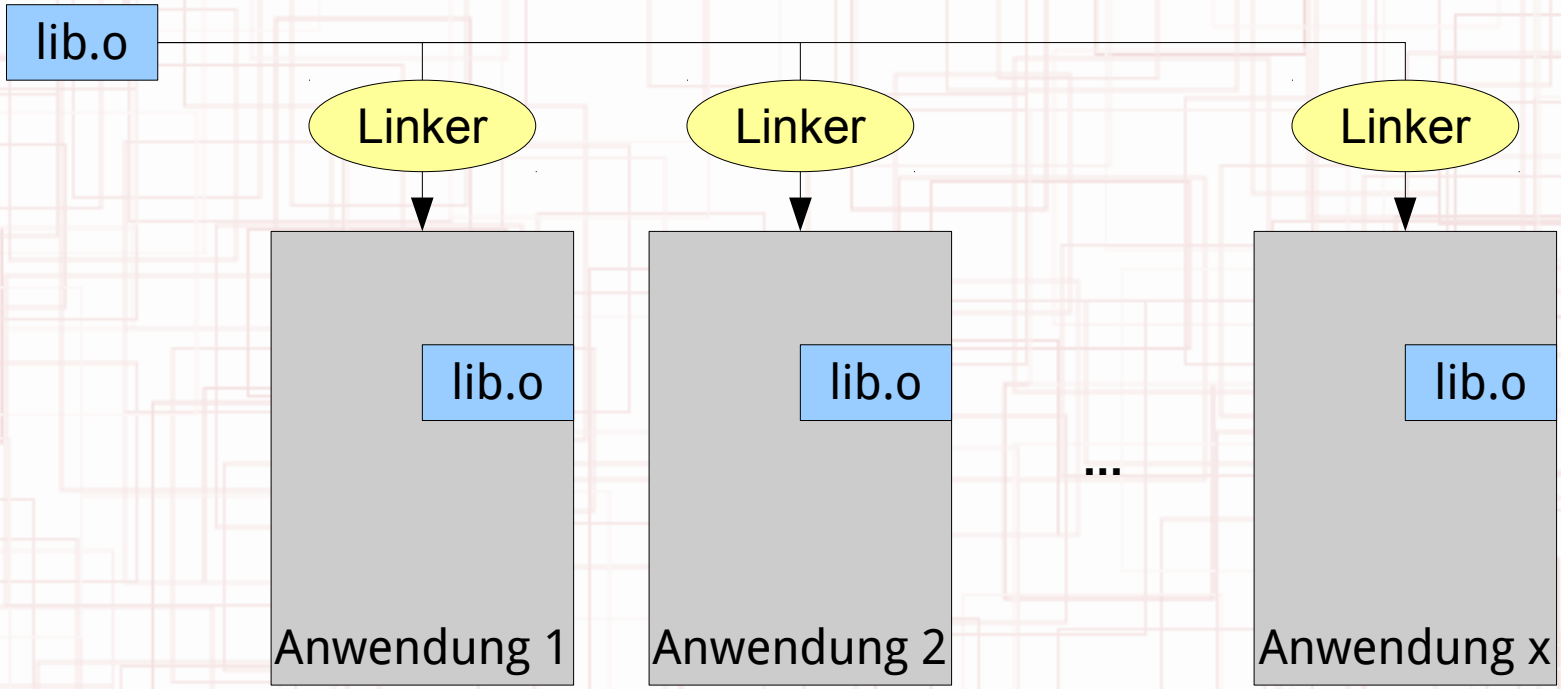
Arten von Programmteilen

- Ausführbare Datei
  - eigentliche Anwendung
  - kann direkt ausgeführt werden
  - kann Kommandozeile/Umgebung auswerten
- in Hochsprache C
  - enthält main(int argc, char **argv)
  - Kern eines Prozesses

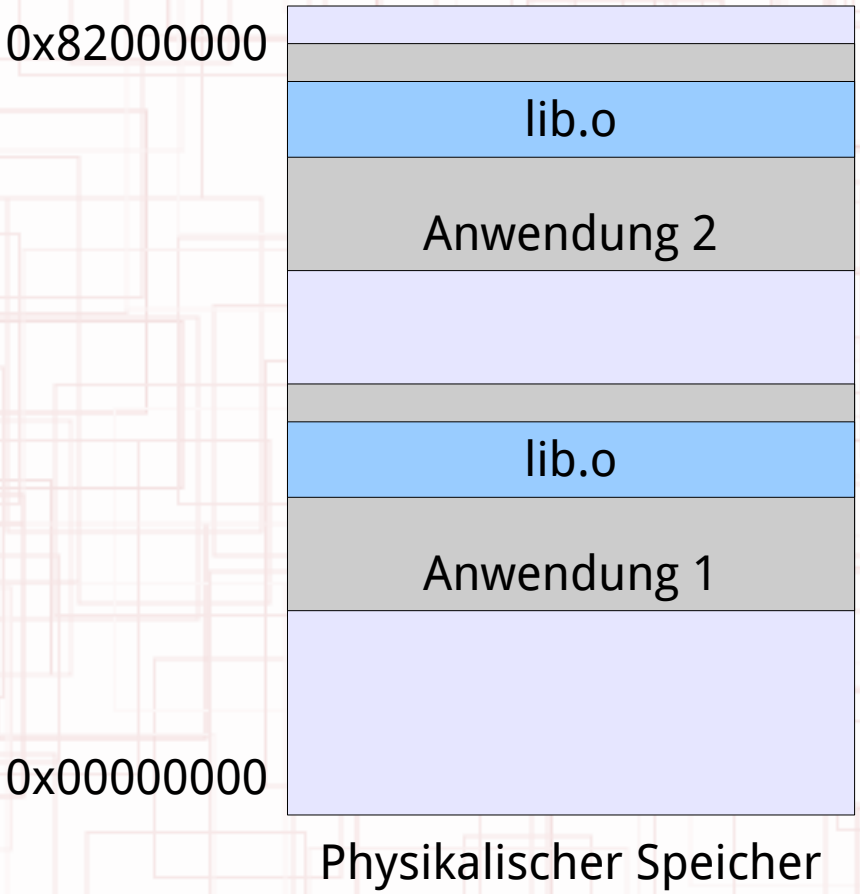
Arten von Programmteilen

- Objektdatei
  - statische Bibliothek, kann „hinzugelinkt“ werden
  - je eine Kopie pro Anwendung!
  - muss für jede Anwendung in den Speicher geladen werden -> Speicherbedarf!
  - Immer verfügbar -> Performance
  - Einbindung durch Linker bei Anwendungserstellung

Statische Bibliothek



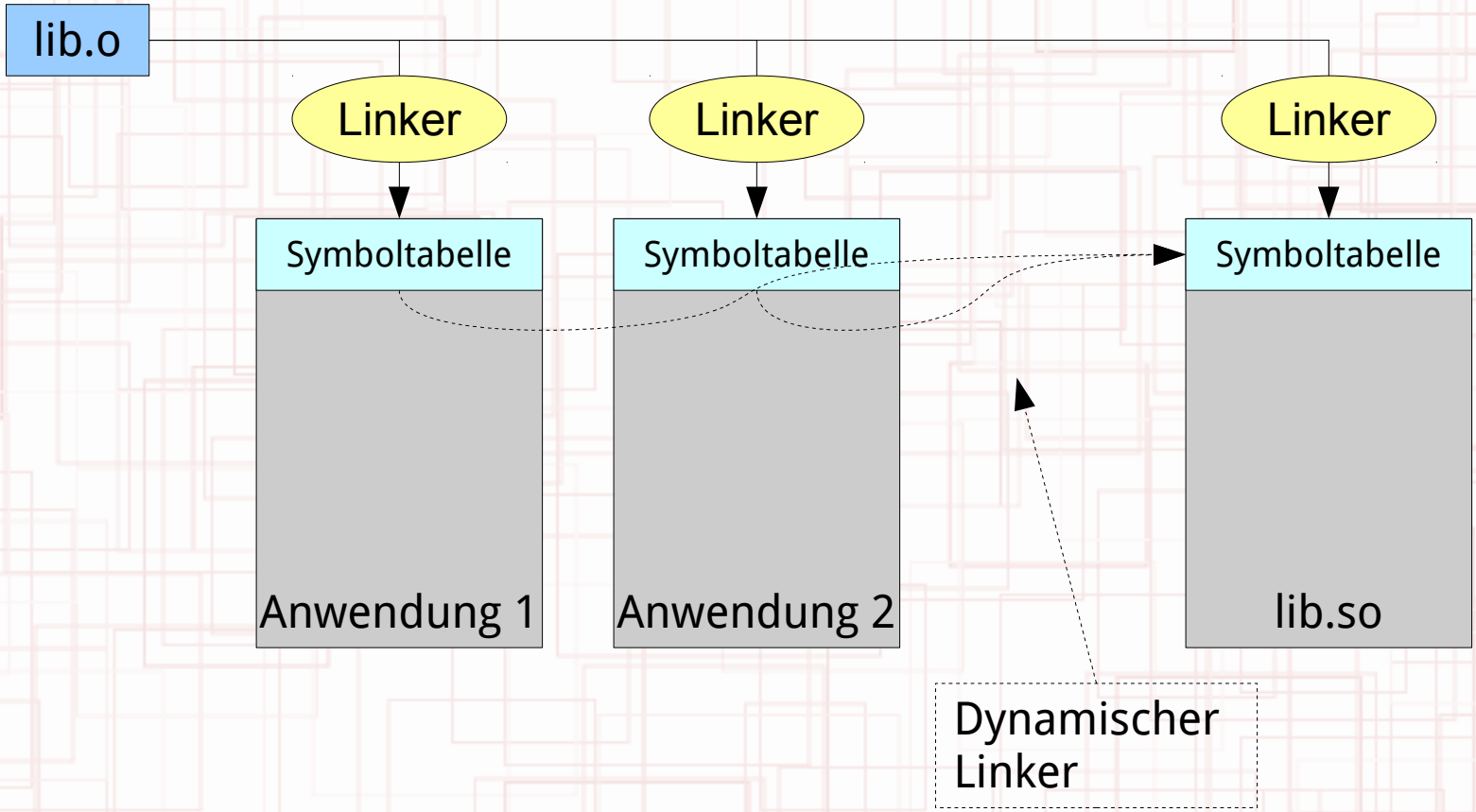
Speicheransicht



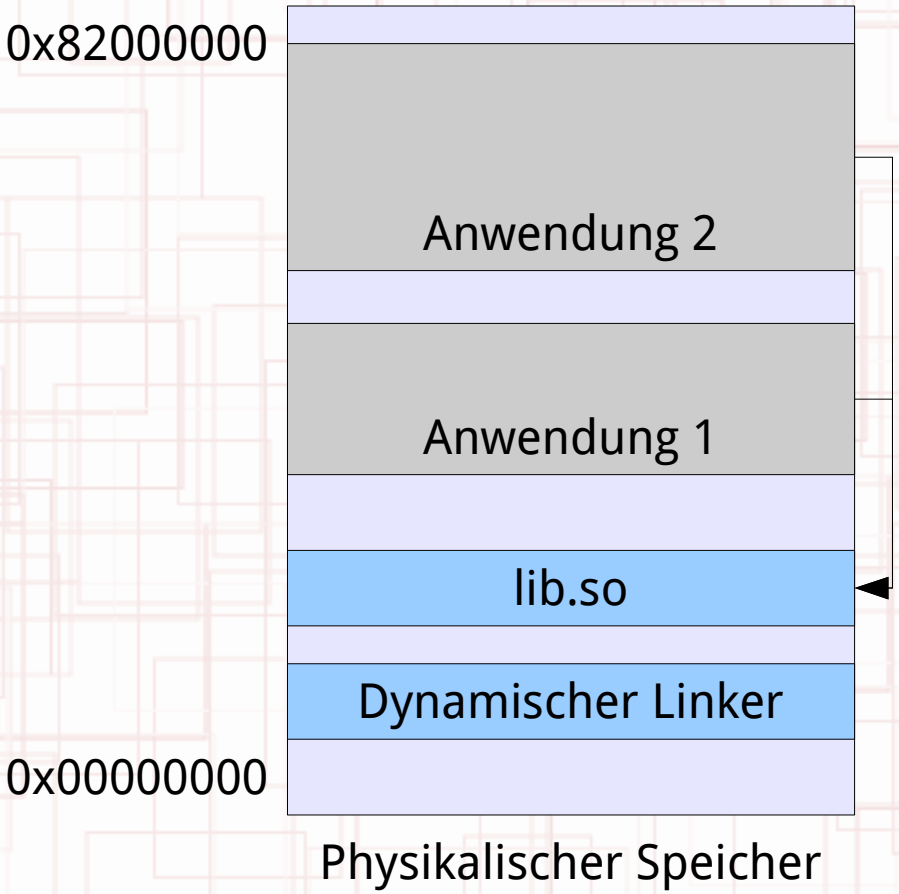
Arten von Programmteilen

- Dynamisches Objekt
  - dynamisch ladbare Bibliothek
  - DLL, shared object, ...
  - eine Kopie für alle Anwendungen, dadurch niedrigerer Speicherbedarf
  - erfordert Lademechanismen -> Performance
  - Einbindung zur Laufzeit -> dynamischer Linker!

Dynamische Bibliothek



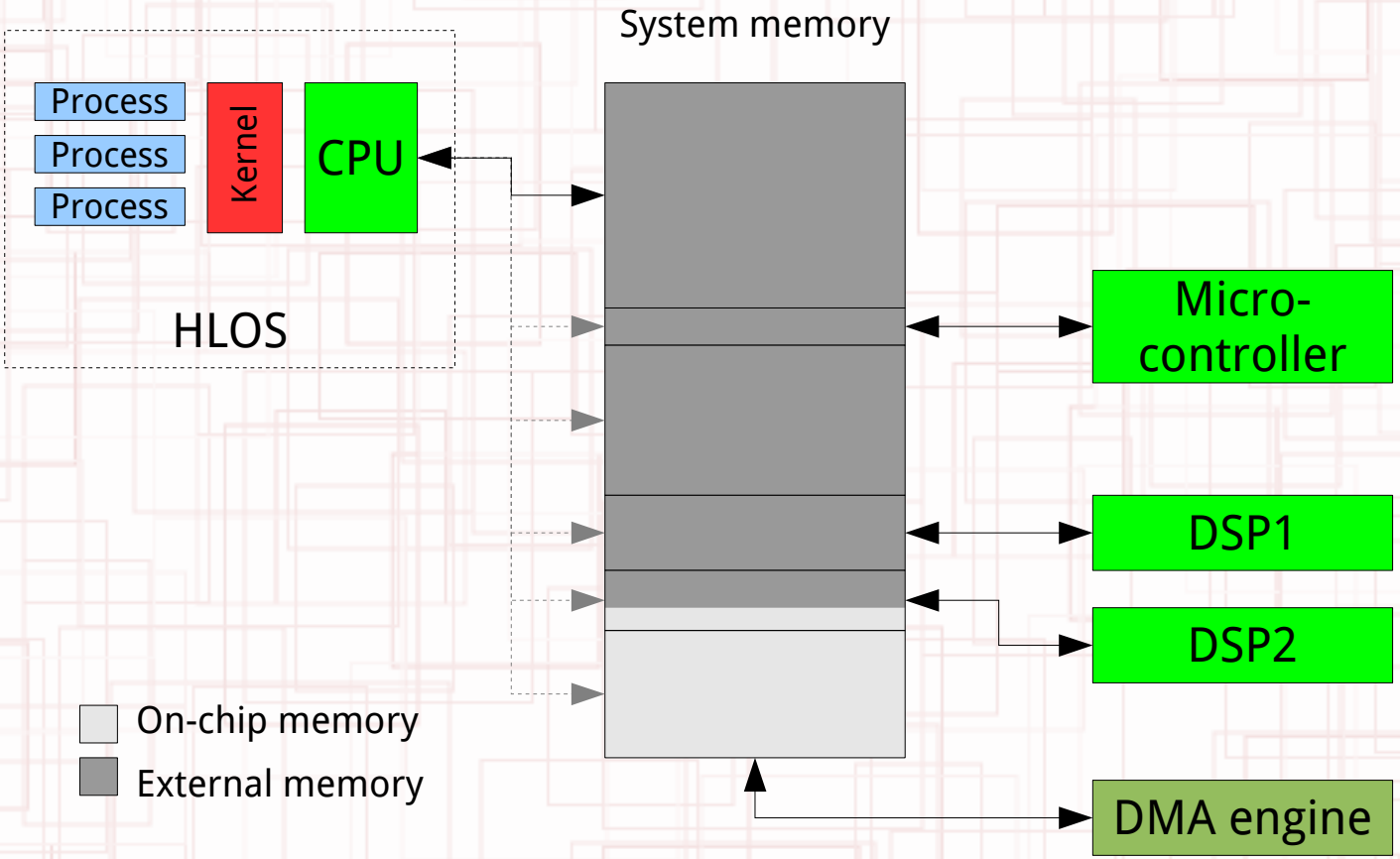
Speicheransicht



Adresskonfiguration

- Bei HLOS: Linker bestimmt die Lage der Objektsegmente selbst
  - Virtueller Speicher pro Prozess!
  - Automatisches Speicherlayout
  - Konsistenter Speicherzugriff innerhalb der Anwendung
- Bei heterogenen SoCs ist Eingriff möglich
  - Speicherstruktur, *memory map*

Speicherlayout



Speicherlayout

Kern	Speicheradresse	Größe	Zweck	Konfiguration via
MPU/Cortex A15	0x80000000	0x10000000	VM	Kernelkonfiguration
MCU/Cortex M4	0x90000000	0x00100000	alles	Linkerparameter
DSP1	0x90100000	0x01000000	alles	Linkerparameter
DSP2	0x92000000 0x02000000	0x01000000 0x00080000	.text, .data, .bss .data	Linkerparameter

- HLOS stellt virtuellen Speicher bereit
- DSP2 verarbeitet schnell anfallende Daten
  - Interner Speicher ist schneller

Einfaches Beispiel

- Internes RAM für Programmcode
- Externes RAM für Variablen und Daten

```
SECTIONS
{
   . = 0x10000;
   .text : { *(.text) }
   . = 0x8000000;
   .data : { *(.data) }
   .bss : { *(.bss) }
}
```

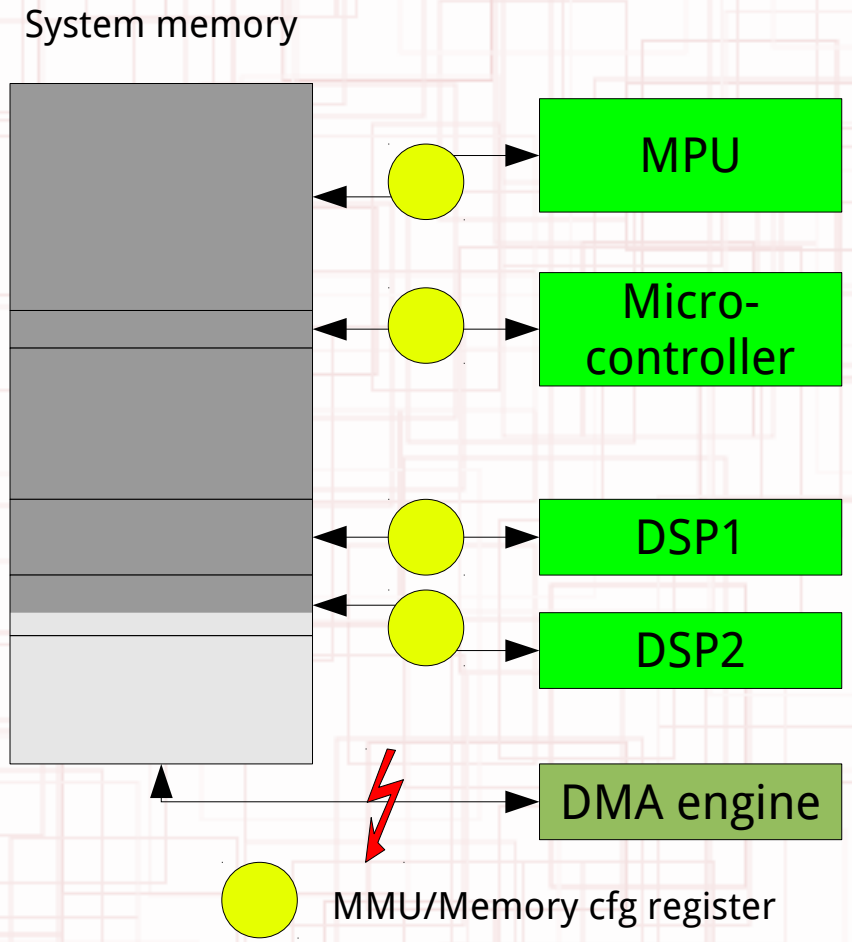
Beispiel: tftp.ld

```
OUTPUT_ARCH(arm)
ENTRY(stext)
SECTIONS
{
 .text 0x00380000 : {  /* Real text segment  */
  _text = .;  /* Text and read-only data */
  *(.text)
  *(.rodata)
  . = ALIGN(4);
  _etext = .;  /* End of text section  */
 }

 .data : {
  __data_start = .;
  *(.data)
  . = ALIGN(4);
  _edata = .;
 }

 .bss : {
  __bss_start = .; /* BSS    */
  *(.bss)
  . = ALIGN(4);
  _end = .;
 }
}
```

Quelle: BiosLT, tftp-Submodul

SoC-Ebene

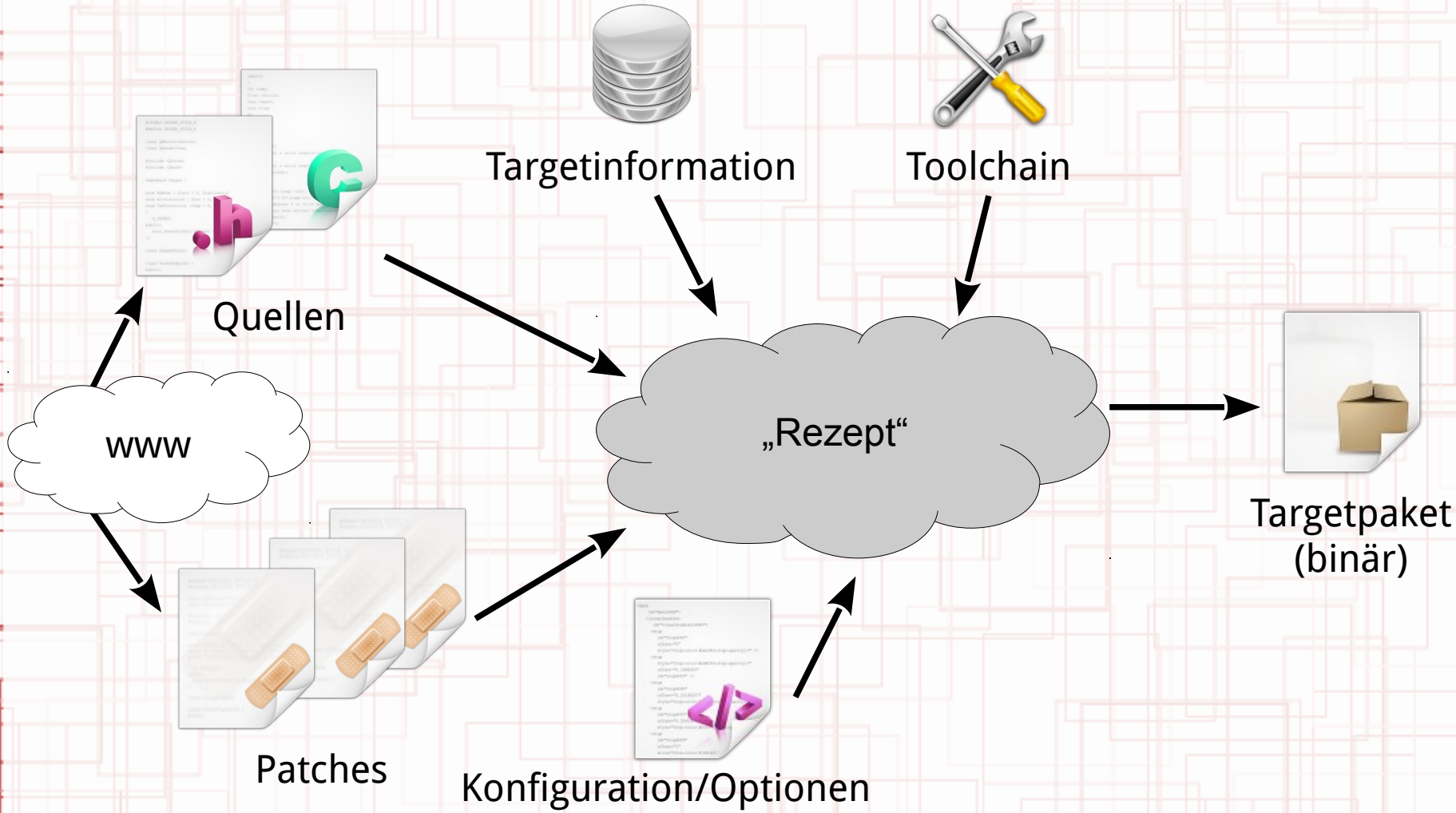
- Prozessoren haben MMU oder Memory-register
- Speicherlayout ist systemweit gültig
  - Konfiguration erforderlich



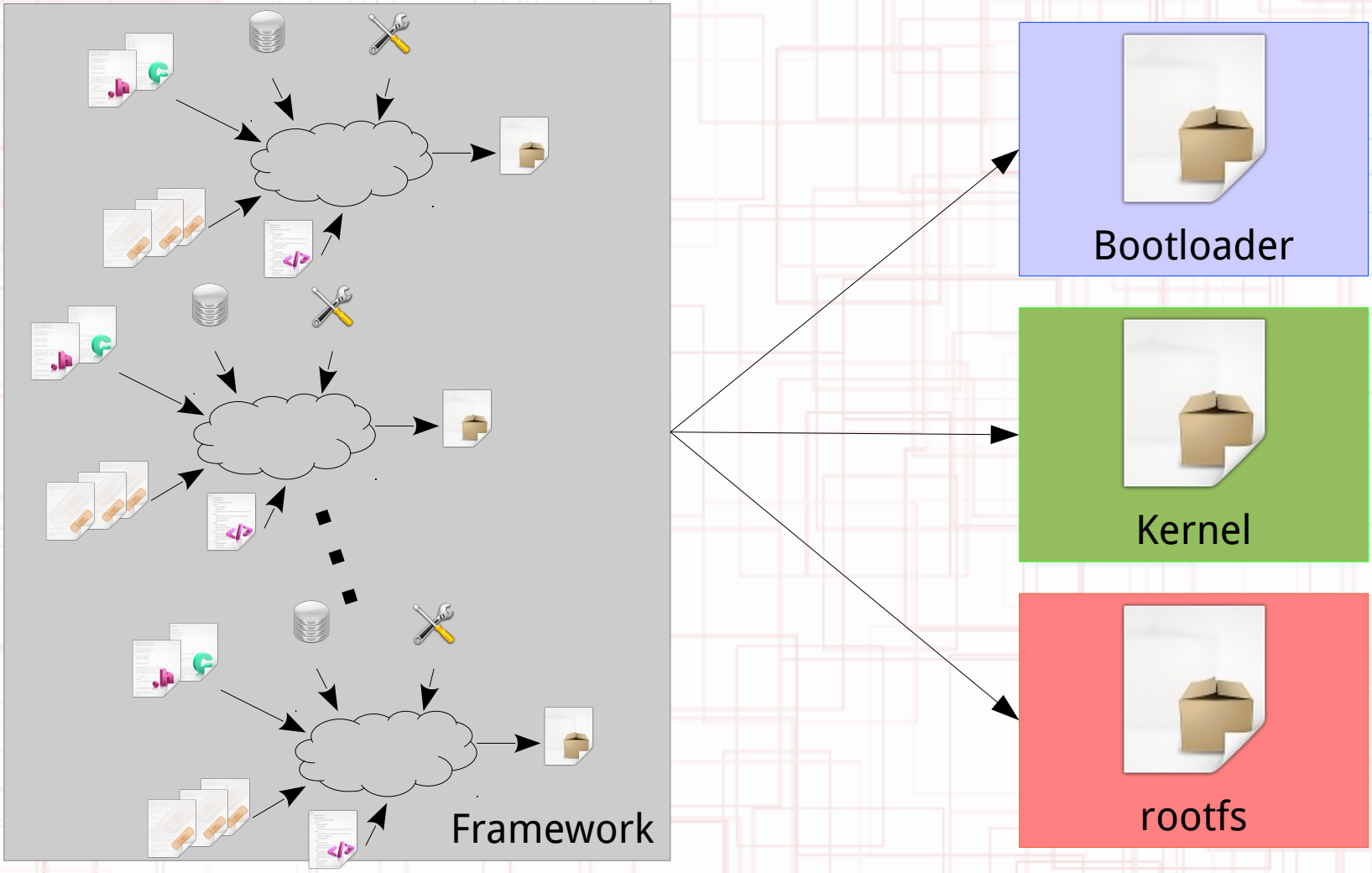
Softwarebuild

- Software liegt oft nur in Quellform vor
- Build für Hostsysteme meist einfach
  - Wie gelangt Software auf das target?
- Verschiedene Schritte erforderlich
  - Konfiguration für das Zielsystem
  - Einspielen von Patches
    - Sonderfunktionen, Erweiterungen
  - Übersetzen mit Crosscompiler

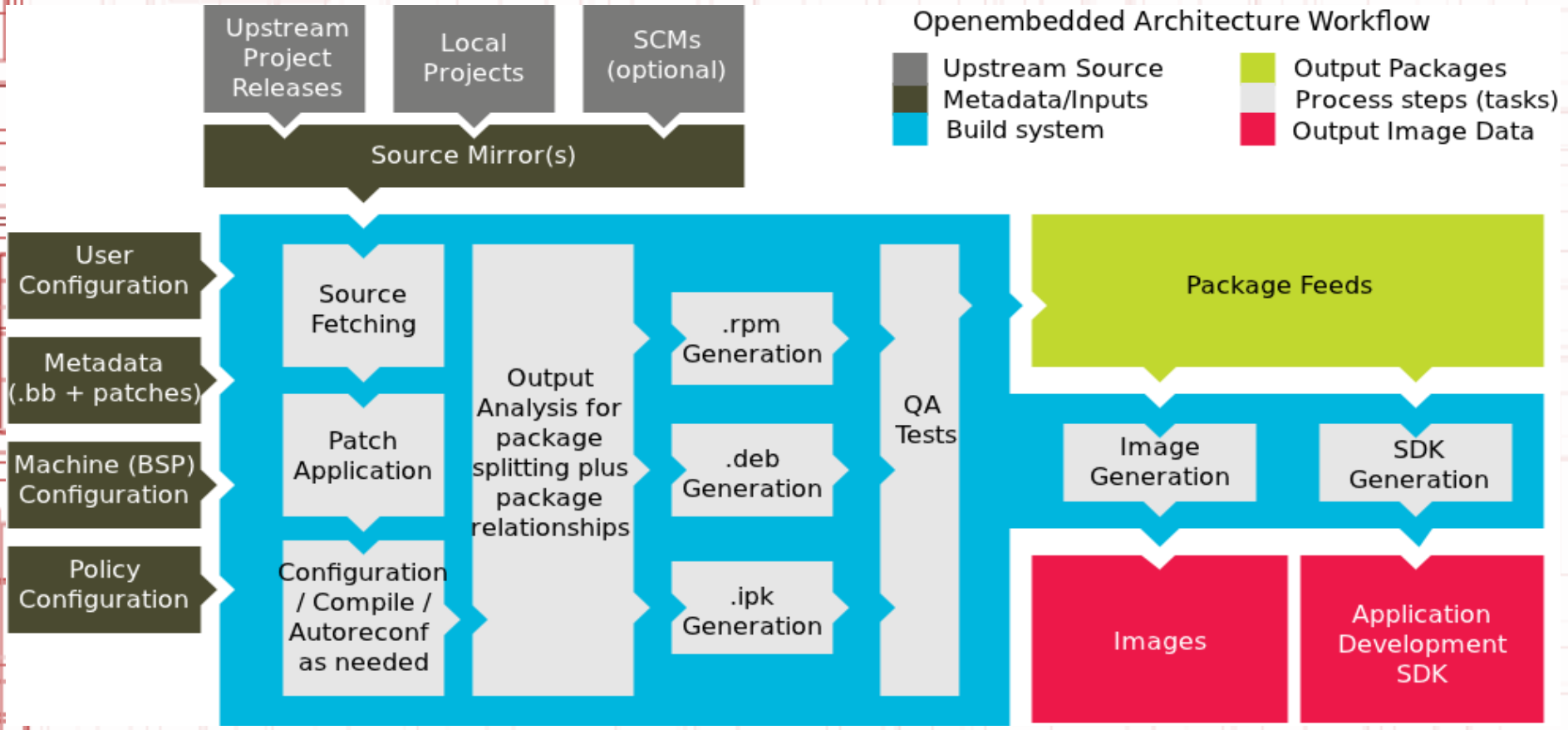
Paketbehandlung



Systemintegration



Yocto



Quelle: http://www.yoctoproject.org/docs/current/mega-manual/mega-manual.html