

Verlässliche Echtzeitsysteme - Übungen

Triple Modular Redundancy

Wintersemester 2024

Eva Dengler, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg

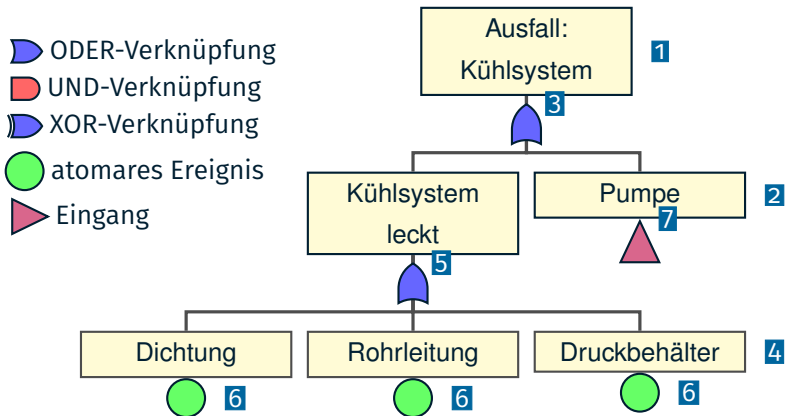
Lehrstuhl Informatik 4 (Systemsoftware)

<https://sys.cs.fau.de>

- 1 Wiederholung: Grundlagen Fehlerbäume
- 2 Wiederholung: Triple Modular Redundancy
- 3 Ausblick: Rechnerarchitektur, Replikation und Redundanz
- 4 Replikation von Code

- 1 Wiederholung: Grundlagen Fehlerbäume
- 2 Wiederholung: Triple Modular Redundancy
- 3 Ausblick: Rechnerarchitektur, Replikation und Redundanz
- 4 Replikation von Code

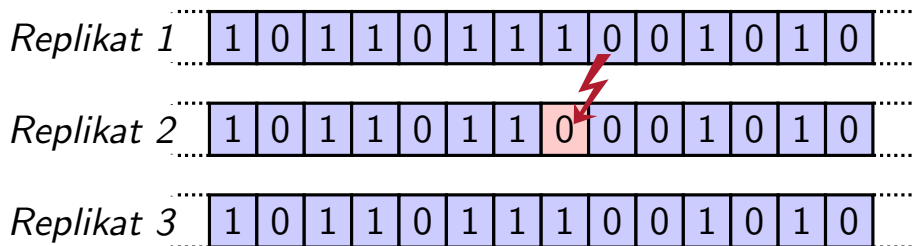
Fehlerbäume – Wiederholung



1. Schadensereignis
2. Ereignisse auf Ebene 2
3. Logische Verknüpfung
4. Ereignisse auf Ebene 3
5. Logische Verknüpfung
6. Atomare Ereignisse
7. Eingänge zerlegen den Fehlerbaum → Neuer Teilbaum

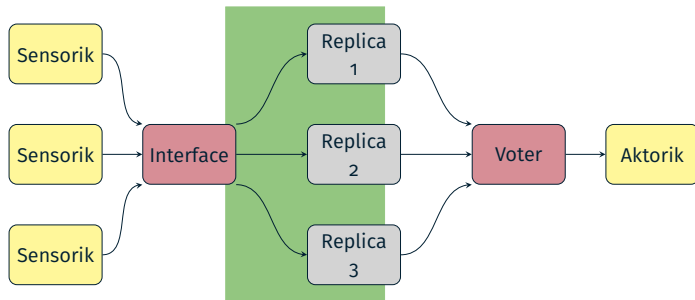
- 1 Wiederholung: Grundlagen Fehlerbäume
- 2 Wiederholung: Triple Modular Redundancy**
- 3 Ausblick: Rechnerarchitektur, Replikation und Redundanz
- 4 Replikation von Code

Fehlerhypothese



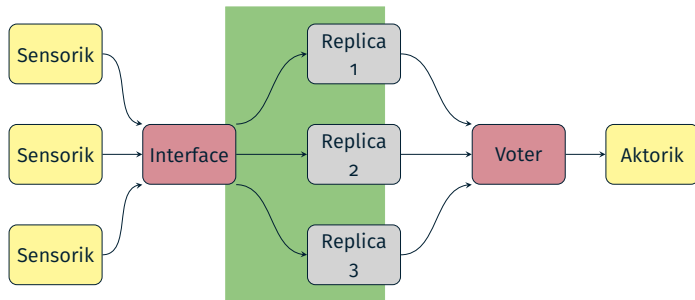
- **Wie viele Replikate** benötigt man zur Fehlermaskierung?
- Arten des Fehlverhaltens (von n Replikaten sind f fehlerhaft)
 1. fail-silent \rightarrow # Replikate: $n = f + 1$
 2. fail-consistent \rightarrow # Replikate: $n = 2f + 1$
 3. malicious \rightarrow # Replikate: $n = 3f + 1$, bösartige verteilte Systeme

Triple Modular Redundancy



- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Insbesondere: Mögl. Mehrheitsentscheid für redundante Sensordaten
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

Triple Modular Redundancy



Redundanzbereich

Ausschließlich Replikatausführung

- Mehrheitsentscheid über Berechnungsergebnisse
- Erweiterung der Ausgangsseite mit Informationsredundanz

Replikdeterminismus

Replik 1

```
void repl_1(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replik 2

```
void repl_2(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

Replik 3

```
void repl_3(void *p){  
    ticks_t time =  
        ezs_get_time();  
  
    ...  
}
```

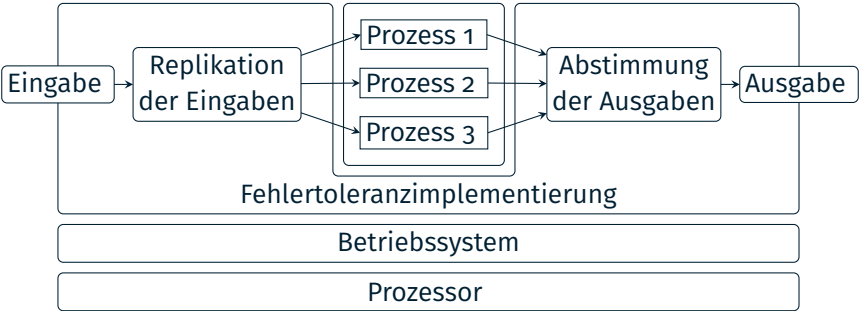
Sicherstellung Replikdeterminismus

- Globale diskrete Zeitbasis
- Einigung über Eingabewerte
- Statische Kontrollstruktur der Replikate
- Deterministische Algorithmen

Sicherstellung Systemverhalten

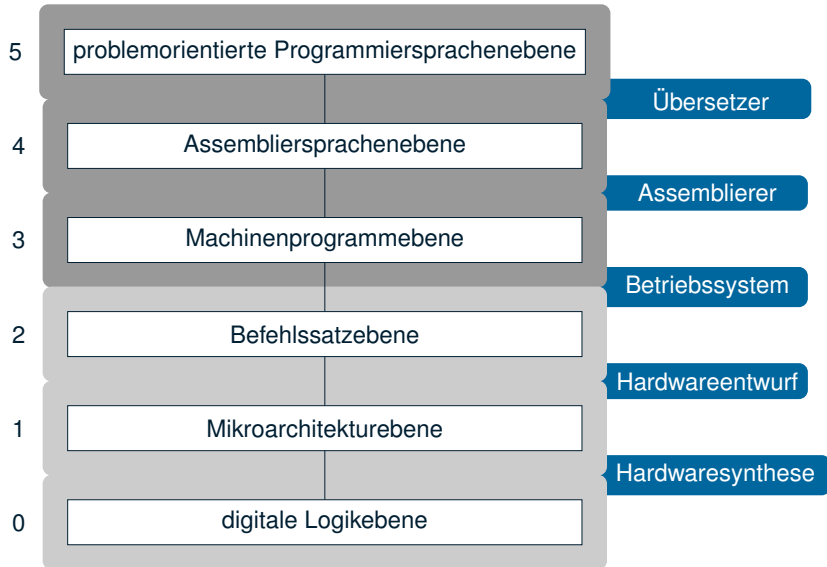
☞ Replikate müssen *innerhalb bestimmter Zeitspanne* terminieren

Process-Level Redundancy



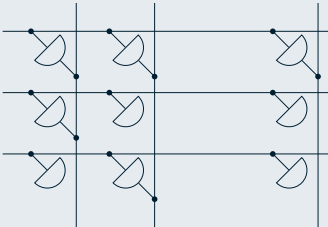
- 1 Wiederholung: Grundlagen Fehlerbäume
- 2 Wiederholung: Triple Modular Redundancy
- 3 Ausblick: Rechnerarchitektur, Replikation und Redundanz**
- 4 Replikation von Code

Ebenen

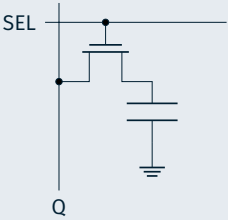


Digitale Logikebene

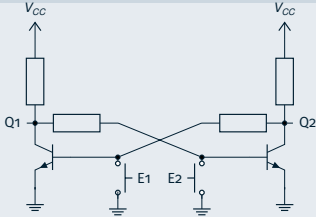
ROM



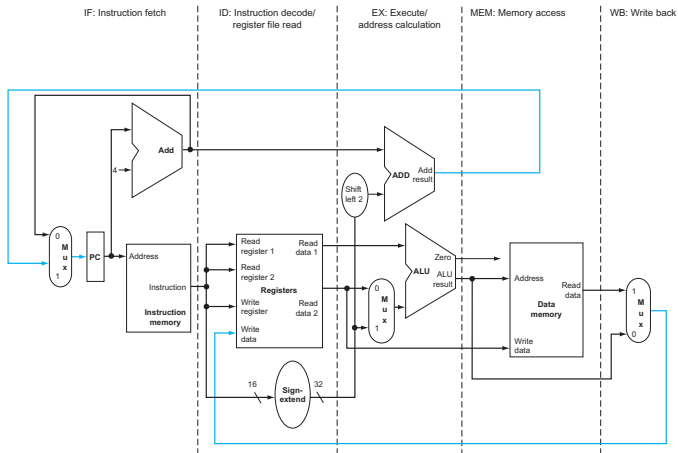
DRAM



RS - FlipFlop

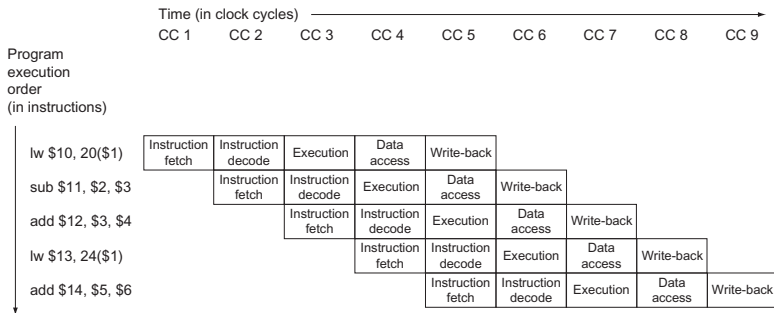


MIPS: Single-Cycle



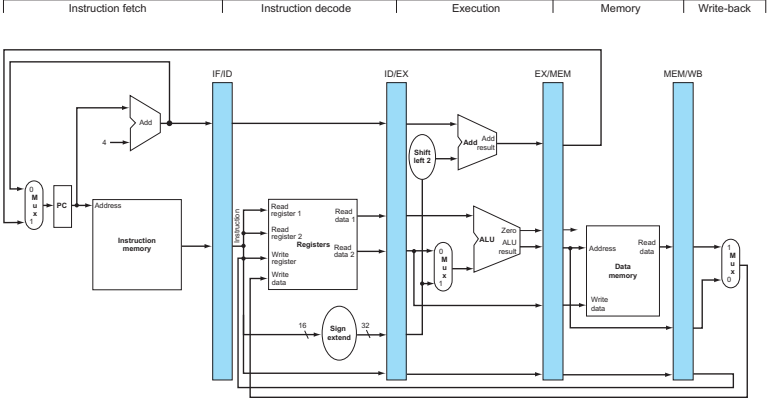
Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining



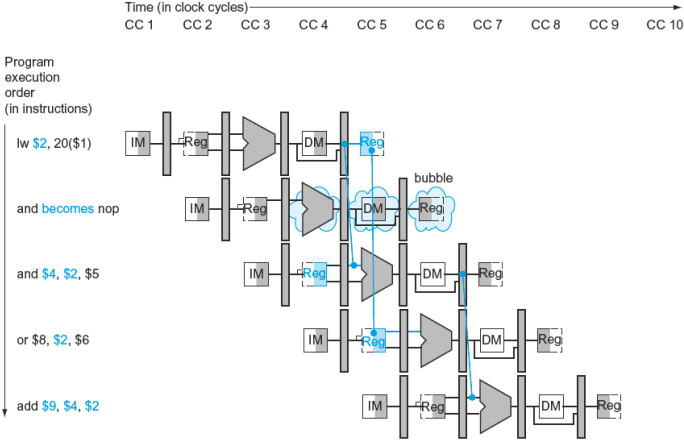
Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining



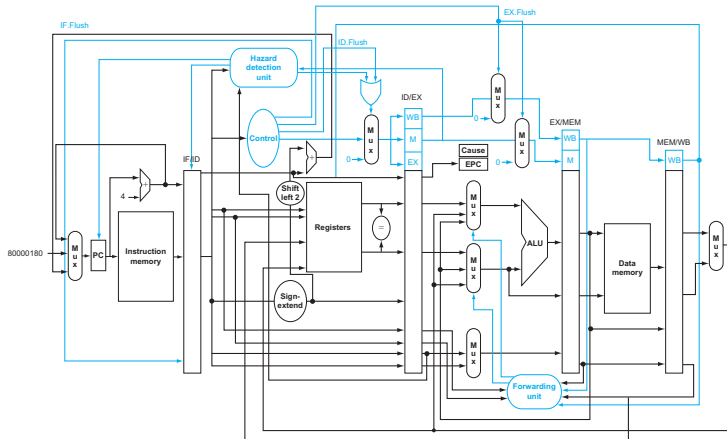
Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining



Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

MIPS: Pipelining

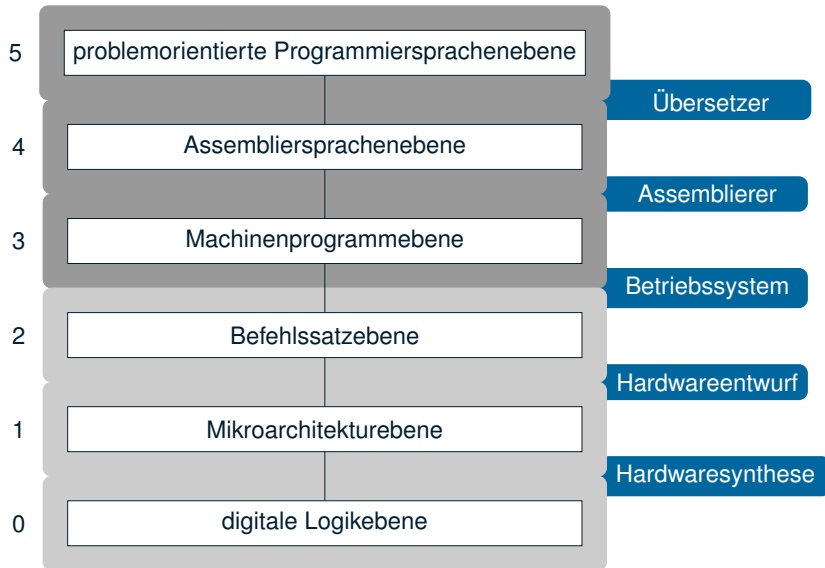


Source: D. A. Patterson und J. L. Hennessy, Computer organization and design: the hardware/software interface, 4th ed., 2012

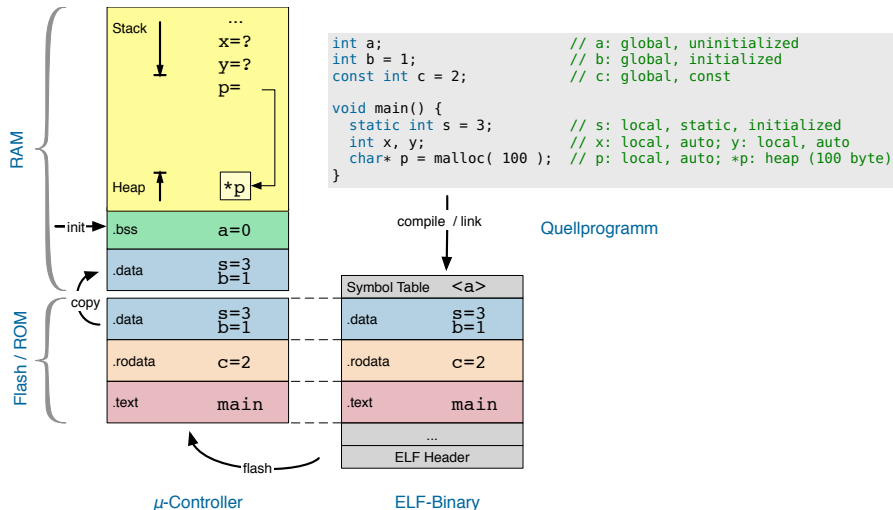
Eigenschaften von CPU-Architekturen

- Mikroprogrammierbar vs. Fixed-Function
 - Out-of-Order-Prozessoren
 - Sprungvorhersage
 - Transaktionaler Speicher
 - Superskalarität
 - Mehrkernarchitekturen
 - Hyperthreading
 - ...
- ☞ All diese zusätzlichen Fehlerpunkte müssen im Fehlermodell berücksichtigt werden
- ☞ Ein Ein-Bit-Fehler in einer dieser Komponenten kann zu komplexen Mehrbitfehlern auf ISA-Ebene führen

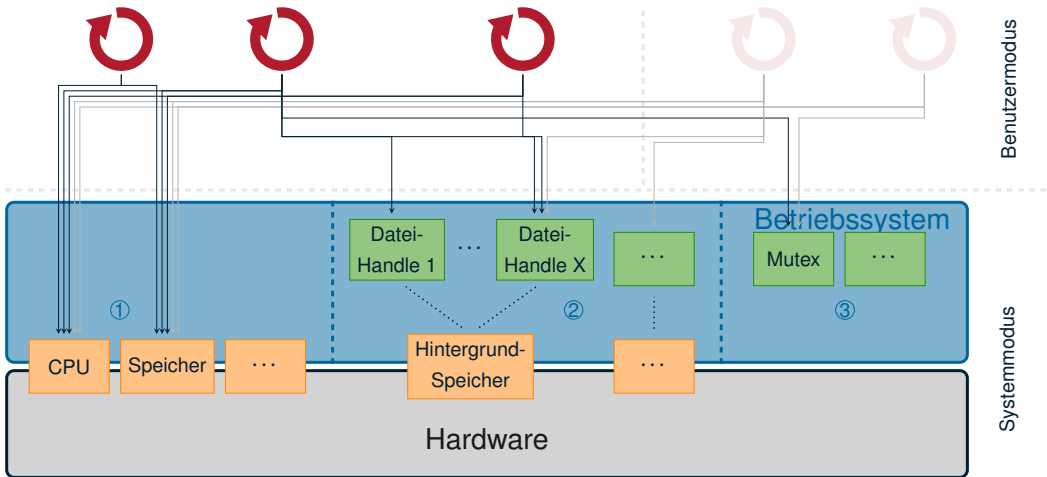
Ebenen



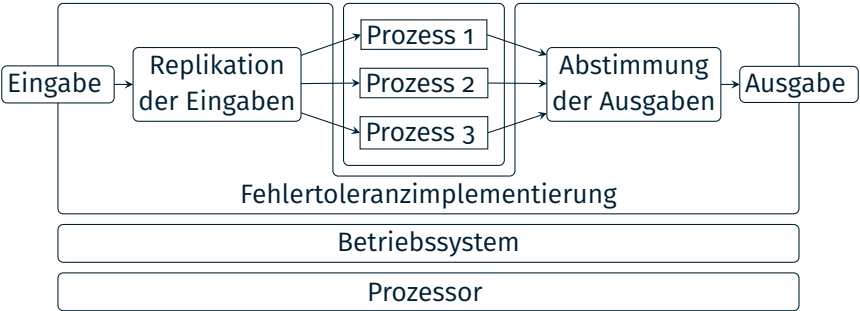
Speicherorganisation auf einem Mikrocontroller



Betriebssystem



Process-Level Redundancy



C-Code vs. Assembler-Code

C-Code

```
int a;
int b = 1;
const int c = 2;
void main() {
    static int s = 3;
    int x, y;
    char* p =
        malloc(100);
}
```

Assembler-Code

```
4004f0 <main>:
4004f0: push    %rbp
4004f1: mov    %rsp,%rbp
4004f4: sub    $0x10,%rsp
4004f8: movabs $0x64,%rdi
400502: callq  4003e0 <malloc@plt>
400507: mov    %rax,-0x10(%rbp)
40050b: add    $0x10,%rsp
40050f: pop    %rbp
400510: retq
```

Wo können Datenfehler auftreten?

1. RAM: `-0 x10 (% rbp)`
 ↪ Stack, globale Daten, Heap, (Programmcode)
2. Allgemeine CPU-Register: `% rsp`
3. Sonstige CPU-Register: `% rip , % rflags`

- 1 Wiederholung: Grundlagen Fehlerbäume
- 2 Wiederholung: Triple Modular Redundancy
- 3 Ausblick: Rechnerarchitektur, Replikation und Redundanz
- 4 Replikation von Code**

Code-Replikation: Stringification

Stringification von CPP

```
#define CMP_FUNC(pre, repl, type, op) \  
    type pre##repl(type a, type b) { \  
        return a op b ? a : b; \  
    }  
  
CMP_FUNC(max, 1, int, >); // Funktion ?max1  
CMP_FUNC(max, 2, int, >); // Funktion ?max2  
...  
CMP_FUNC(min, 1, int, <); // Funktion ?min1
```

- Verwendung des C-Präprozessors (CPP)
- #: „Token Pasting Operator“
- Konkatenieren zweier Token zu einem
- Aufruf & Deklaration müssen erstellt werden
 - ☞ Es geht eleganter ...

C++ Template

```
template <typename T>
T max(T x, T y) {
    T value;
    if (x < y) value = y;
    else value = x;
    return value;
}
double md = max<double>(2.3, 4.2);
auto mi = max<int>(23U, 42);
```

- Templates ermöglichen generische Programmierung
- Wiederverwendung durch **Parametrisierung**
- Unterscheidung von Funktions- & Klassen-Templates
- Expansion zur Compilezeit \leadsto Quelltext muss verfügbar sein (Header)
- „Code Bloat“ beim Compilieren \rightarrow **nutzbar für Replikation von Code**

- Explizite Typen als Templateparameter möglich
- Nutzbar zum „Zählen“ von Templates

Indizierte Template-Spezialisierung

```
template <typename T, unsigned INDEX>
T max(T x, T y) {
    T value;
    if (x < y)
        value = y;
    else
        value = x;
    return value;
}
...
auto m0 = max<int, 0>(23, 42);
auto m1 = max<int, 1>(23, 42);
```

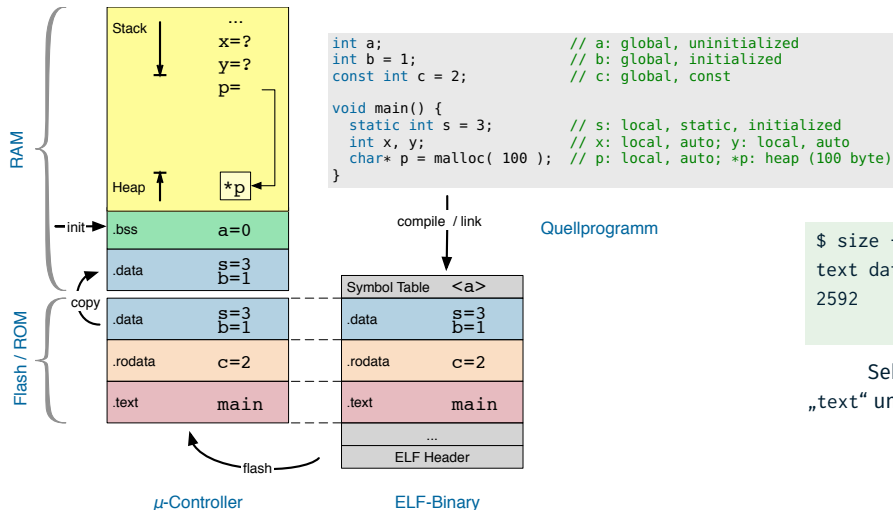
Einbinden von C++-Code in C

C Linkage

```
// C++ code
extern "C" void f(int); // one way
extern "C" {           // another way
    int g(double);
    double h();
};
void code(int i, double d)
{
    f(i);
    int ii = g(d);
    double dd = h();
    // ...
}
```

- C++ betreibt name mangling
⇒ `int test(int) ↦ _Z4testi`
- Name mangling verhindern
⇒ C++-Code aus C-Code aufrufbar

Speicherorganisation und Sektionsgrößen



```

$ size --format=berkeley a.out
text data bss  dec  hex  filename
2592   8   4 2604 a2c   a.out
    
```

Sektionsgrößen in Byte,
„text“ umfasst hier: .text + .rodata

Umgang mit Assembly-Code I

nm: Ausgabe der Symboltabelle

```
00000000000004028 D var_initialized
...
00000000000201028 B __bss_start
00000000000201028 b var_uninitialized
...
0000000000000061a T main
00000000000000580 t register_tm_clones
00000000000000510 T _start
...
0000000000000066d t int max<int,0u>(int,int)
0000000000000067c t int max<int,1u>(int,int)
```

■ Nützliche Optionen

- -C, --demangle: Dekodieren der C++-Namensmangelung:
_Z3maxIiLj0EET_S0_S0_ ⇒ `int max<int, 0u>(int, int)`
- -S, --print-size: Ausgabe der Symbolgrößen
0000000000000061 a 00000028 T main

Umgang mit Assembly-Code II

objdump: Ausgabe von Informationen über Objektdateien

```
int main(){
4007cd: 55                push   %rbp
4007ce: 48 89 e5         mov    %rsp,%rbp
4007d1: 48 83 ec 10      sub    $0x10,%rsp
int a = max<int>(23U, 42);
4007d5: be 2a 00 00 00   mov    $0x2a,%esi
4007da: bf 17 00 00 00   mov    $0x17,%edi
4007df: e8 04 01 00 00   callq 4008e8 <_Z3maxIiET_S0_S0_>
4007e4: 89 45 fc         mov    %eax,-0x4(%rbp)
std::cout << a << "\n";
4007e7: 8b 45 fc         mov    -0x4(%rbp),%eax
...
```

■ Nützliche Optionen

- -S: Ausgabe von Quell-Code im Assembly-Code (Debug-Symbole notwendig)
- -D: alle Sektionen disassemblieren