Hochschule Bochum
Bochum University
of Applied Sciences

Grundlagen Rechnertechnik

Prof. Dr. Peter Gerwinski

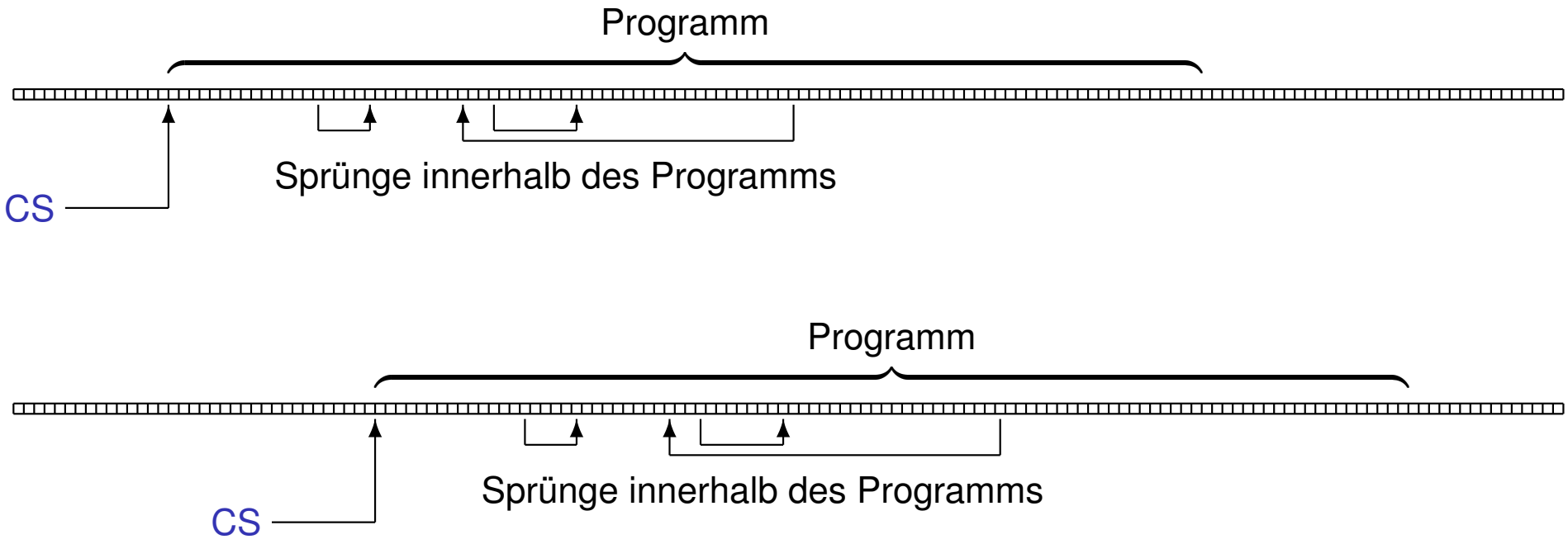
22. Januar 2013

Grundlagen Rechnertechnik

6 Anwender-Software

6.1 Relokation und Linken

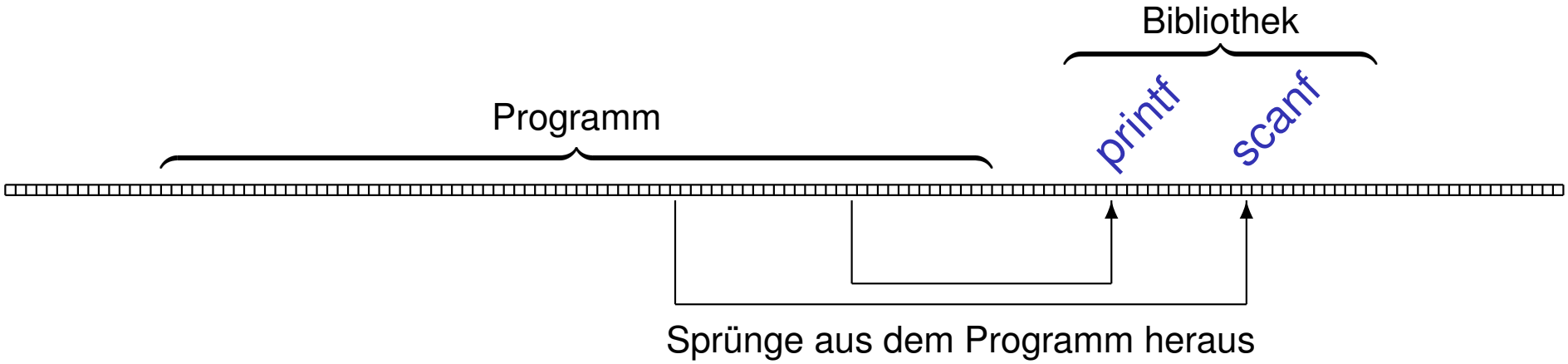
Software im Speicher

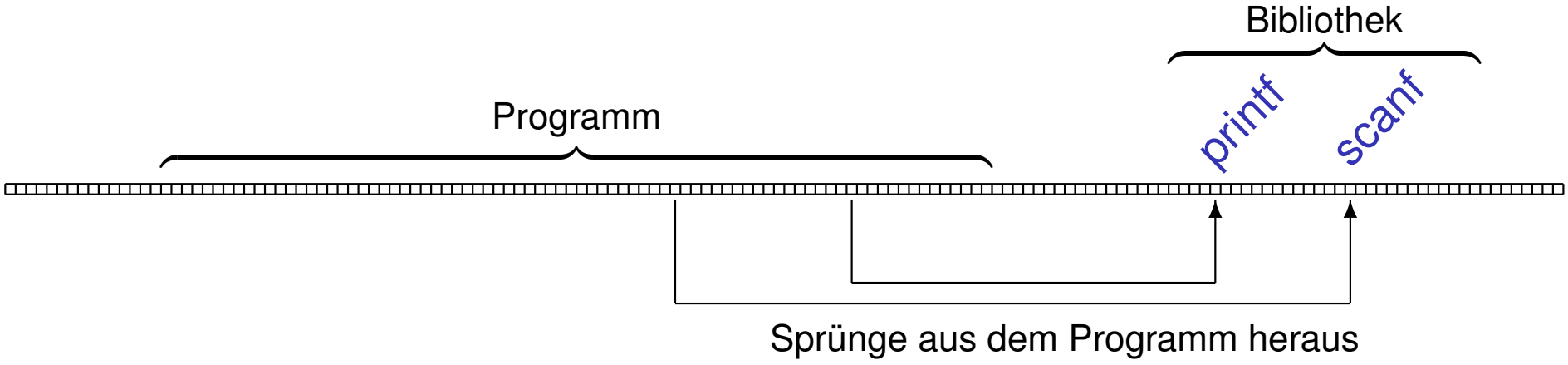


Sprünge anpassen: Relokation

Hardware-Unterstützung (z. B. Intel): Speichersegmentierung
CS = Code-Segment: Segment-Register oder Selektor

6.1 Relokation und Linken





Sprünge anpassen: Linken

Beim Erzeugen der Datei: statisches Linken
Beim Laden: dynamisches Linken

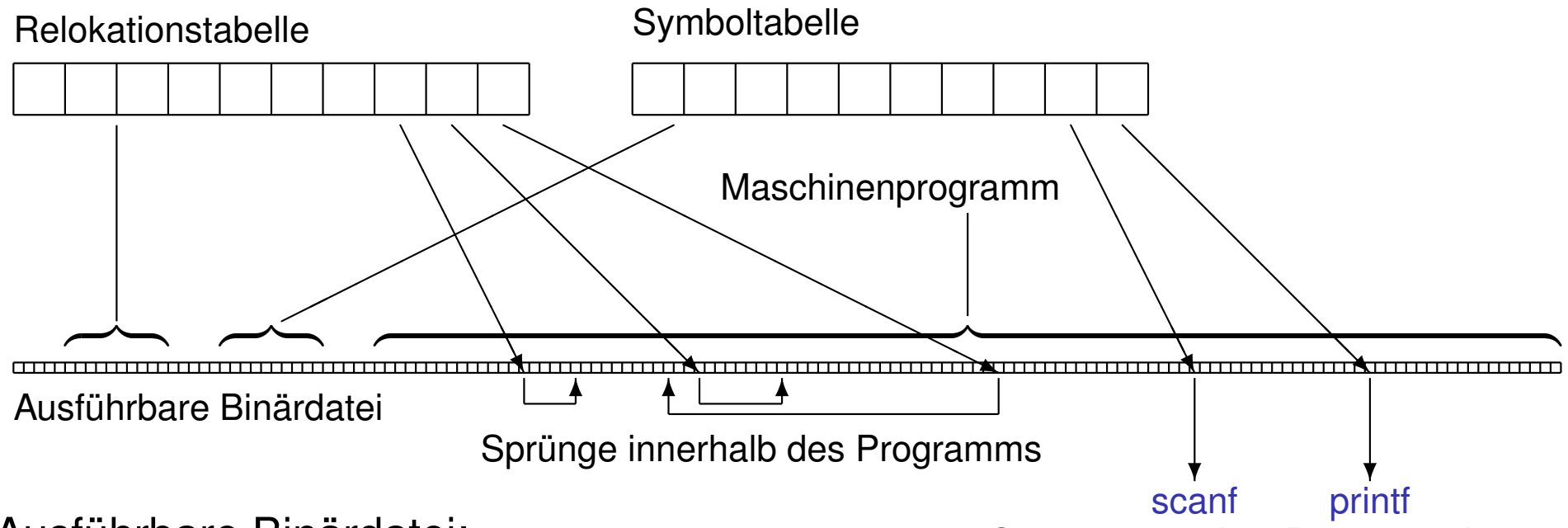
6.2 Dateiformate

Man kann Maschinenprogramme nicht „einfach so“ in den Speicher laden.

Sprünge anpassen

- Relokation: Relokationstabelle
- Linken: Symboltabelle

6.2 Dateiformate

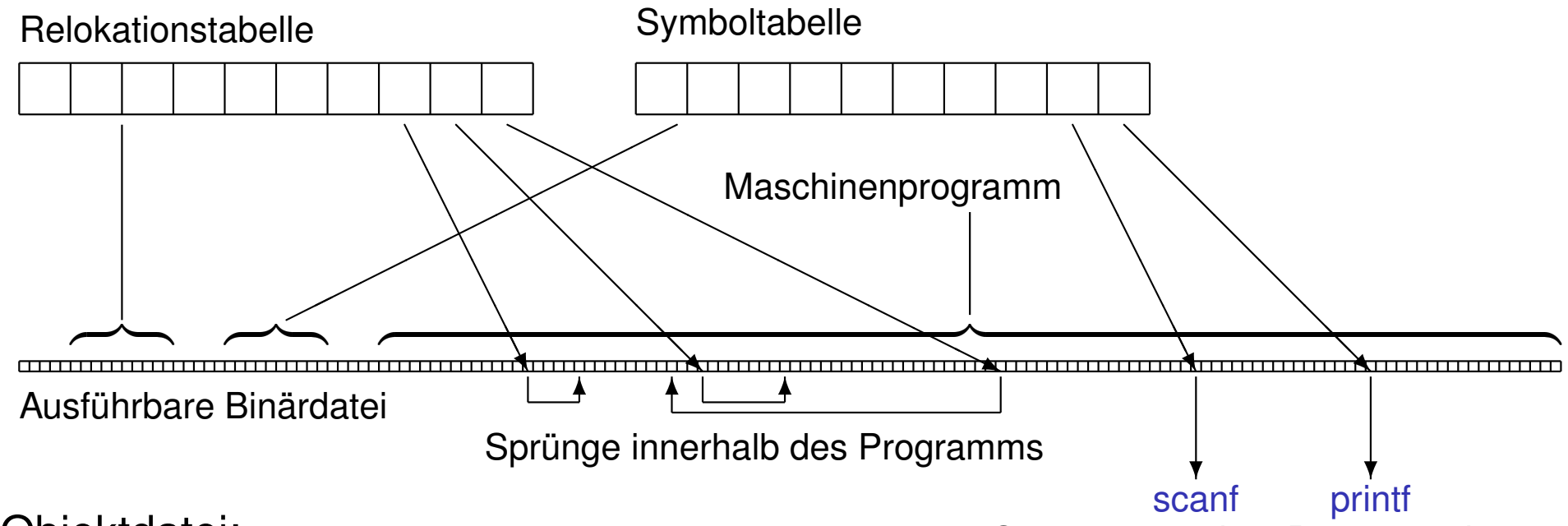


Ausführbare Binärdatei:

Relokationstabelle,
Symboltabelle für dynamischen Linker

Formate: a.out, COFF, ELF, . . .
Dateiendungen: (keine), .elf, .com, .exe, .scr

6.2 Dateiformate

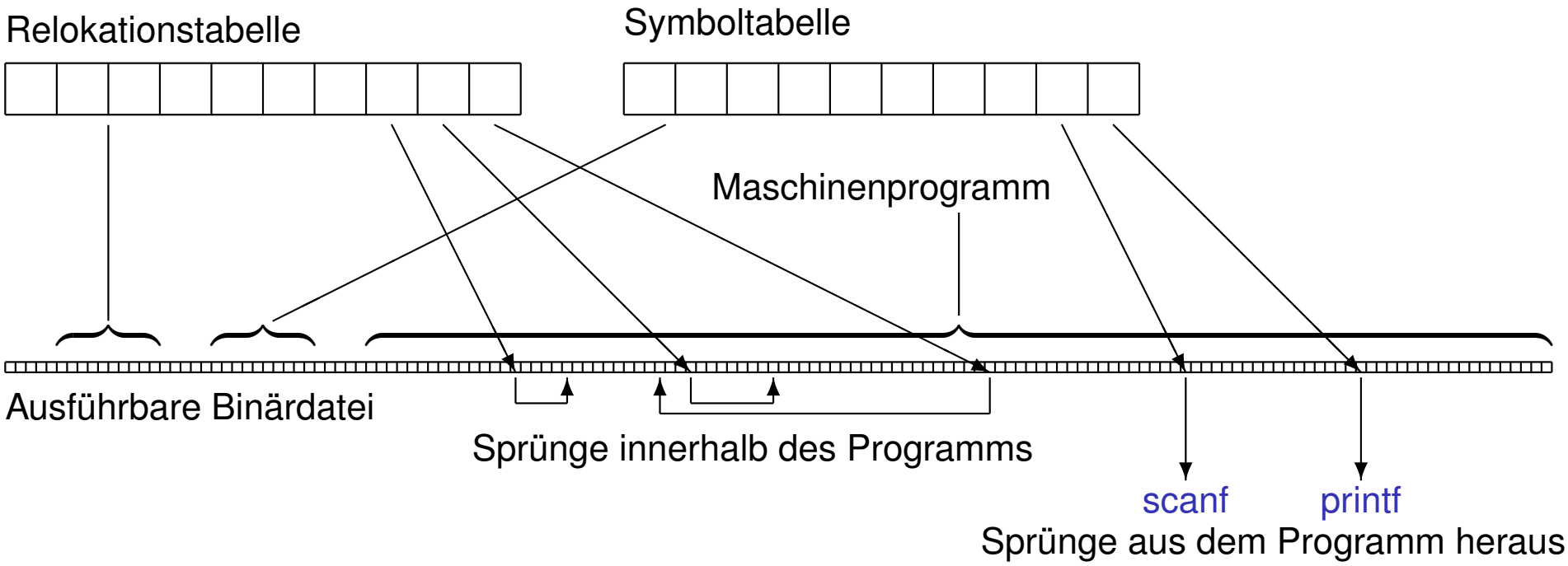


Objektdatei:

Relokationstabelle,
Symboltabellen für statischen und dynamischen Linker

Formate: a.out, COFF, ELF, . . .
Dateiendungen: .o, .obj

6.2 Dateiformate

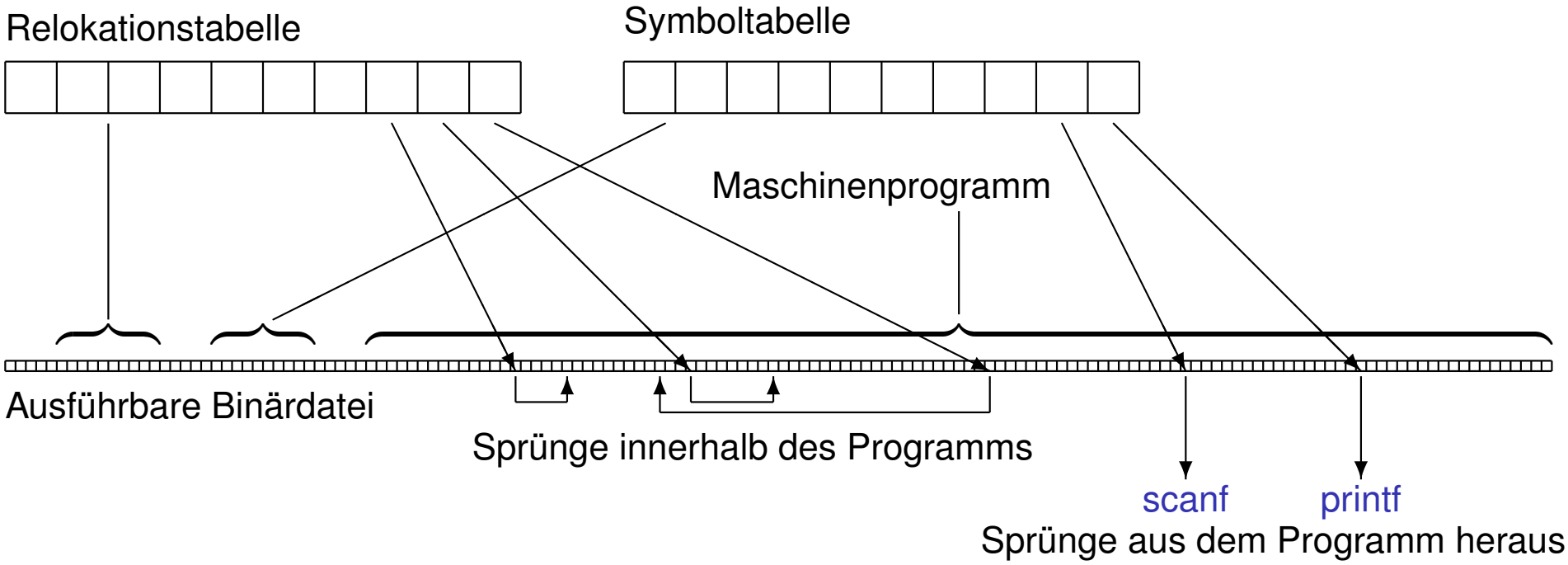


Bibliothek:

Zusammenfassung mehrerer Objekt-Dateien

Statische Bibliotheken: .a, .lib
Dynamische Bibliotheken: .so, .dll

6.2 Dateiformate



Bibliothek:

Zusammenfassung mehrerer Objekt-Dateien

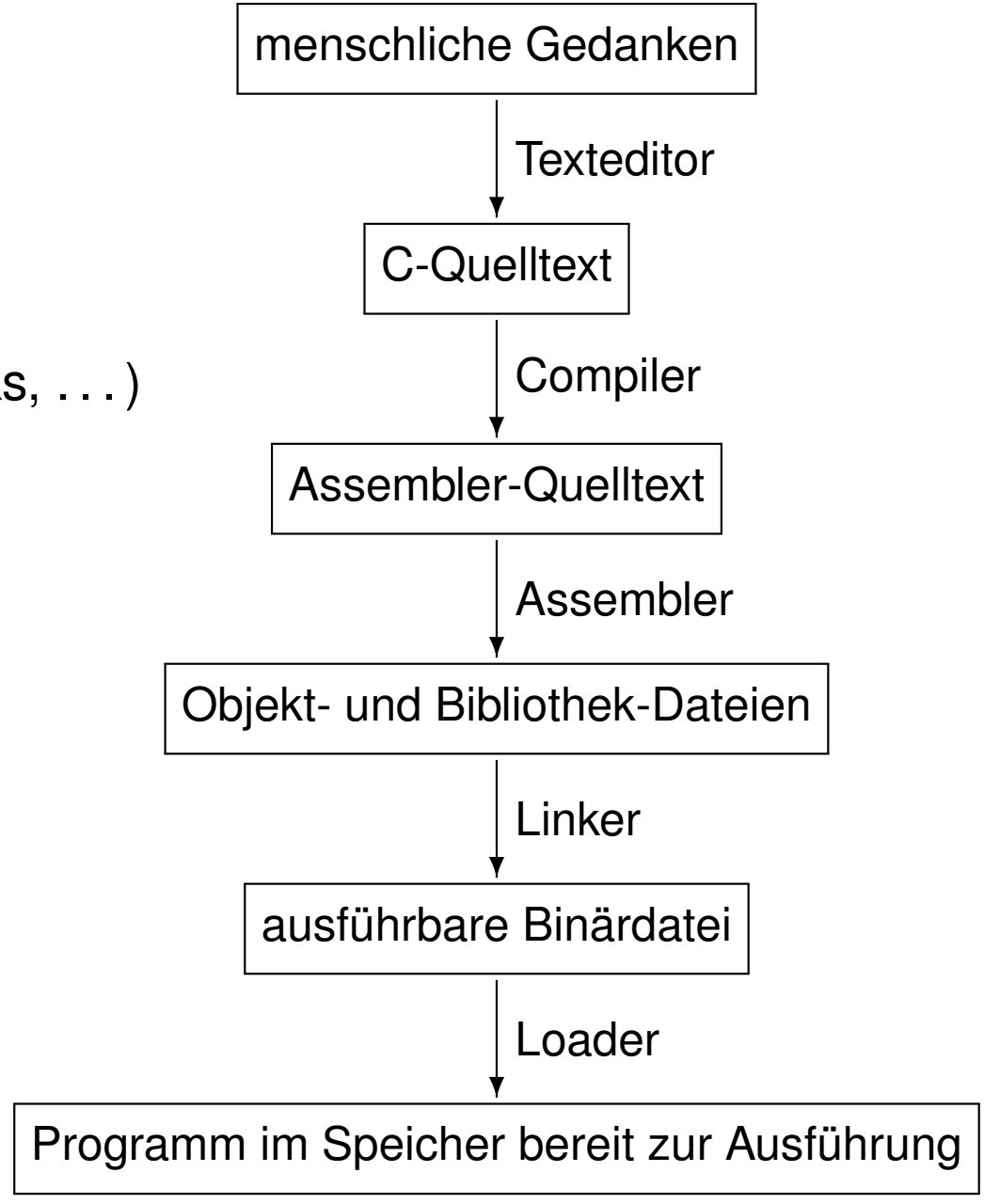
Statische Bibliotheken: .a, .lib
Dynamische Bibliotheken: .so, .dll

Statisches Linken: „alles an Bord“
Dynamisches Linken: kleinere Dateien, Speicher besser nutzen

6.3 Die Toolchain

Automatischer Aufruf:

- Entwicklungsumgebungen
  (z. B. Eclipse, Code::Blocks, . . .)
- gcc = Compiler
  + Assembler
  + Linker
  + . . .
- make kann *alles* aufrufen



6.4 Besonderheiten von Mikro-Controllern

Kein Betriebssystem
$\longrightarrow$ kein Relocator, kein dynamischer Linker
$\longrightarrow$ Wir müssen dem Mikro-Controller alles „mundgerecht“ servieren.

- fertiges ROM: Hersteller
- Flash-Speicher und In-System Programmer (ISP)
- Flash-Speicher und Boot-Loader

In jedem Fall: statisch linken, Relokation vorher
$\longrightarrow$ ELF-Datei in HEX-Datei umwandeln

Format: Intel-Hex-Format
Dateiendung: .hex

Grundlagen Rechnertechnik

7 Bus-Systeme

7.1 Konzept

Datenübertragung

- von Schaltkreis zu Schaltkreis im Prozessor
- zwischen Prozessor und Speicher
- zwischen Prozessor und Gerät
- zwischen Prozessor und Controller
- zwischen Controller und Gerät
- zwischen Netzwerkkarte und Netzwerkkarte
- . . .

7 Bus-Systeme

7.1 Konzept

Datenübertragung „ohne Umsteigen“

- von Schaltkreis zu Schaltkreis im Prozessor
- zwischen Prozessor und Speicher
- zwischen Prozessor und Gerät
- zwischen Prozessor und Controller
- zwischen Controller und Gerät
- zwischen Netzwerkkarte und Netzwerkkarte
- . . .

7 Bus-Systeme

7.2 Zu berücksichtigen

- Elektromagnetische Störungen von/nach außen
  - Abschirmung
  - symmetrische Signalübertragung
  - unterschiedliche Verdrillung („Twisted Pair“)
  - ...

7 Bus-Systeme

7.2 Zu berücksichtigen

- Elektromagnetische Störungen von/nach außen
- Reflexion am Kabelende
  - Abschlußwiderstand („Terminator“)
  - ignorieren

7 Bus-Systeme

7.2 Zu berücksichtigen

- Elektromagnetische Störungen von/nach außen
- Reflexion am Kabelende
  - Abschlußwiderstand („Terminator“)
  - ignorieren
- Kabelwege
  - Wenige Leitungen (seriell) vs. viele (parallel)
  - Punkt-zu-Punkt-Verbindung vs. Adressierung
  - Topologie: linear, ring-, sternförmig, kaskadierbar

7 Bus-Systeme

7.2 Zu berücksichtigen

- Elektromagnetische Störungen von/nach außen
- Reflexion am Kabelende
  - Abschlußwiderstand („Terminator“)
  - ignorieren
- Kabelwege
  - Wenige Leitungen (seriell) vs. viele (parallel)
  - Punkt-zu-Punkt-Verbindung vs. Adressierung
  - Topologie: linear, ring-, sternförmig, kaskadierbar
- Kollisionsvermeidung
- Sicherheitsaspekte
- . . .

7 Bus-Systeme

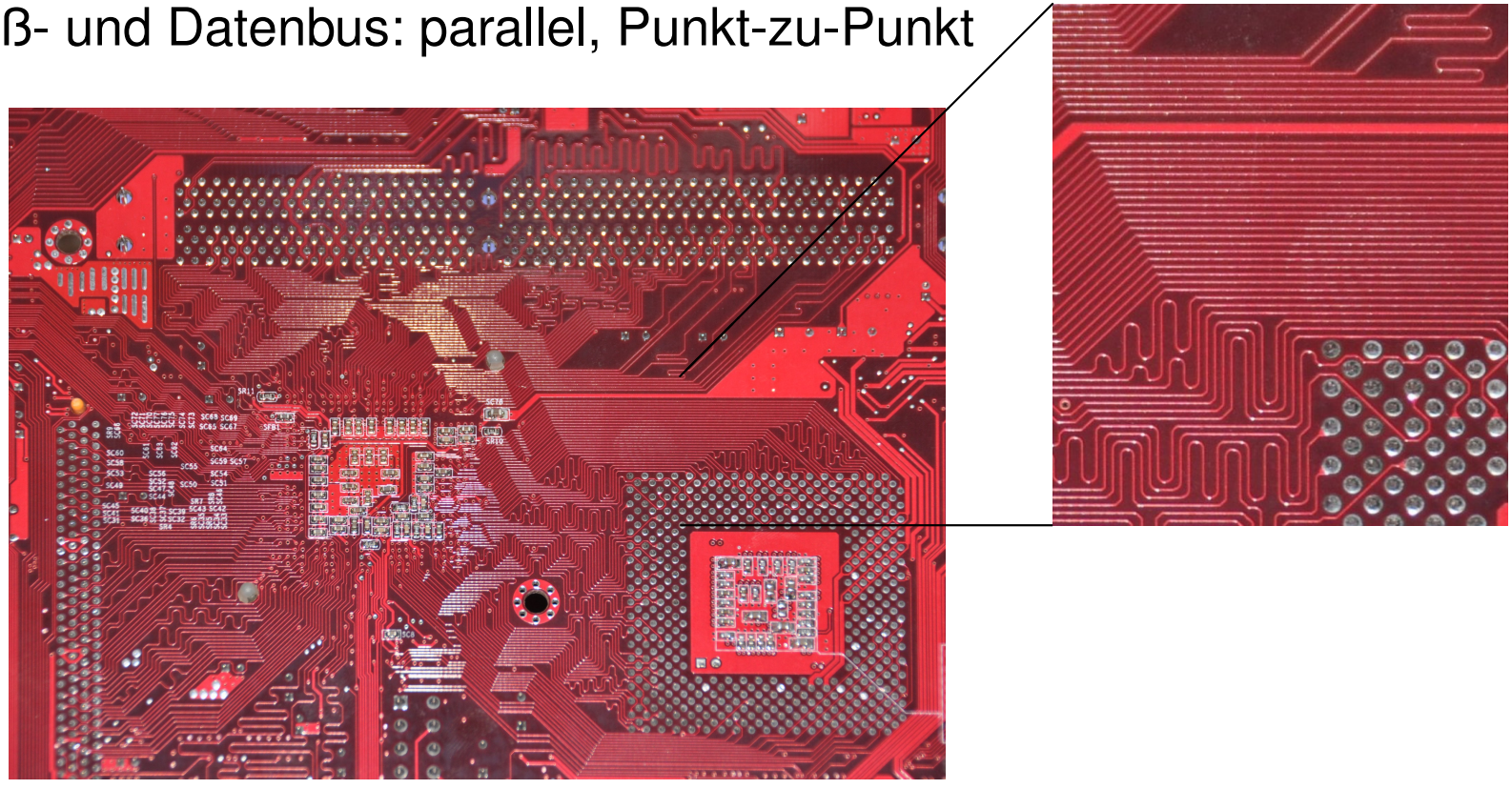
7.3 Beispiele

- von Schaltkreis zu Schaltkreis im Prozessor:
  parallel, Punkt-zu-Punkt

7 Bus-Systeme

7.3 Beispiele

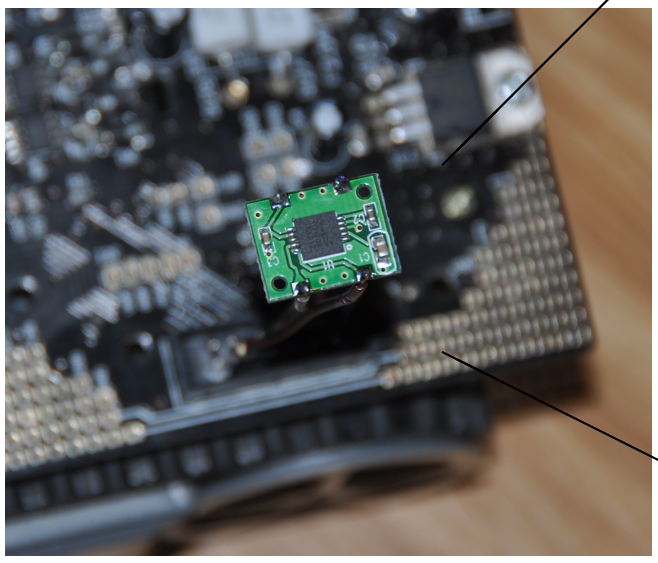
- von Schaltkreis zu Schaltkreis im Prozessor:
  parallel, Punkt-zu-Punkt
- zwischen Prozessor und Speicher:
  Adreß- und Datenbus: parallel, Punkt-zu-Punkt

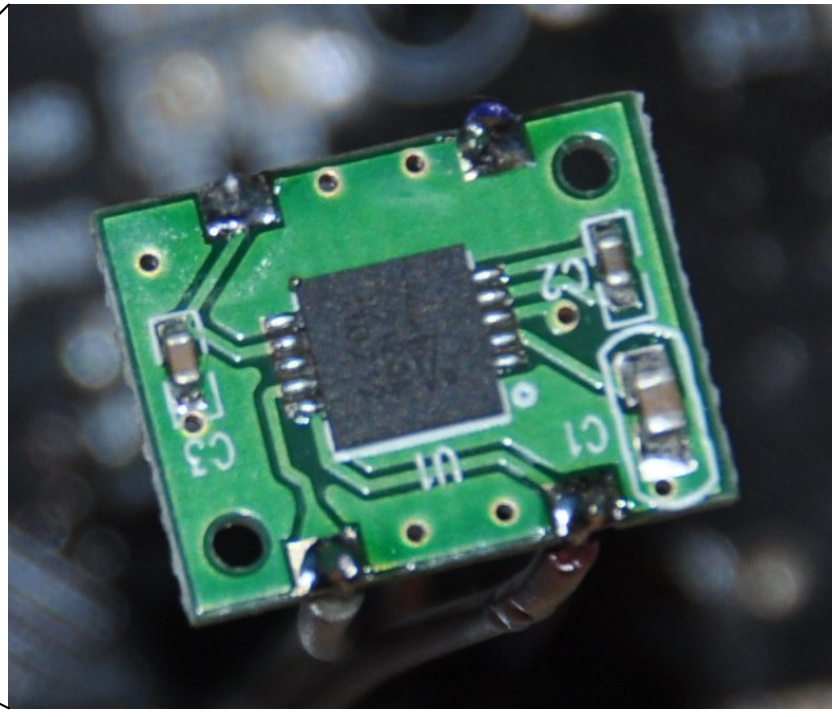


7 Bus-Systeme

7.3 Beispiele

- von Schaltkreis zu Schaltkreis im Prozessor:
  parallel, Punkt-zu-Punkt
- zwischen Prozessor und Speicher:
  Adreß- und Datenbus: parallel, Punkt-zu-Punkt
- zwischen Prozessor und Gerät:
  I^2C: seriell, mit Adressierung

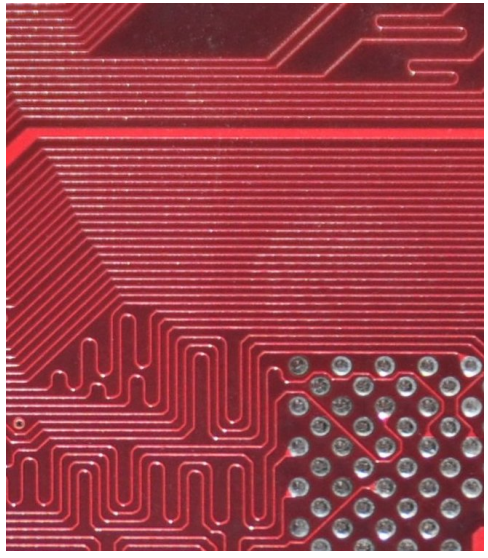




7 Bus-Systeme

7.3 Beispiele

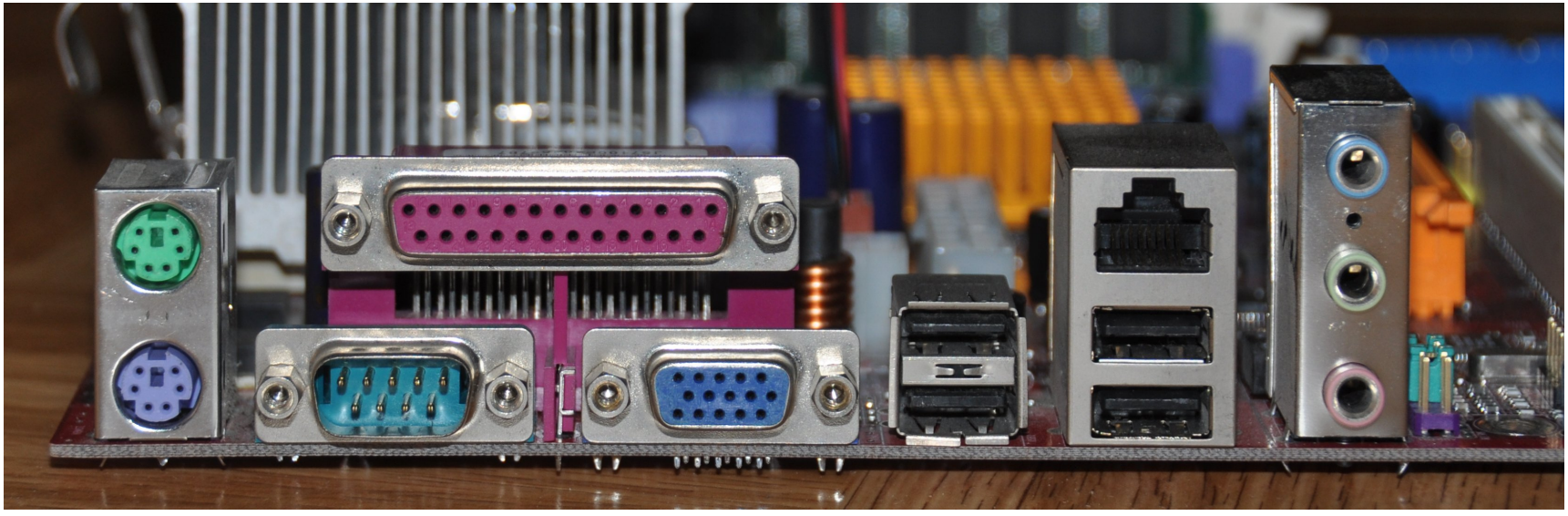
- von Schaltkreis zu Schaltkreis im Prozessor:
  parallel, Punkt-zu-Punkt
- zwischen Prozessor und Speicher:
  Adreß- und Datenbus: parallel, Punkt-zu-Punkt
- zwischen Prozessor und Gerät:
  I^2C: seriell, mit Adressierung
- zwischen Prozessor und Controller:
  parallel, mit Adressierung



7 Bus-Systeme

7.3 Beispiele

- zwischen Controller und Gerät:
  - IEEE 1284 (Centronics): parallel, Punkt-zu-Punkt
  - RS-232: seriell, Punkt-zu-Punkt
  - RS-485, USB, CAN: seriell, mit Adressierung



7 Bus-Systeme

7.3 Beispiele

- zwischen Controller und Festplatte:
  - SCSI: parallel, terminiert,
    mit Adressierung, linear

7 Bus-Systeme

7.3 Beispiele

- zwischen Controller und Festplatte:
  - SCSI: parallel, terminiert,
    mit Adressierung, linear
  - IDE/ATAPI/PATA: parallel, nicht terminiert,
    mit Adressierung (Master/Slave), linear

7 Bus-Systeme

7.3 Beispiele

- zwischen Controller und Festplatte:
  - SCSI: parallel, terminiert,
    mit Adressierung, linear
  - IDE/ATAPI/PATA: parallel, nicht terminiert,
    mit Adressierung (Master/Slave), linear
  - SATA: seriell, Punkt-zu-Punkt

7 Bus-Systeme

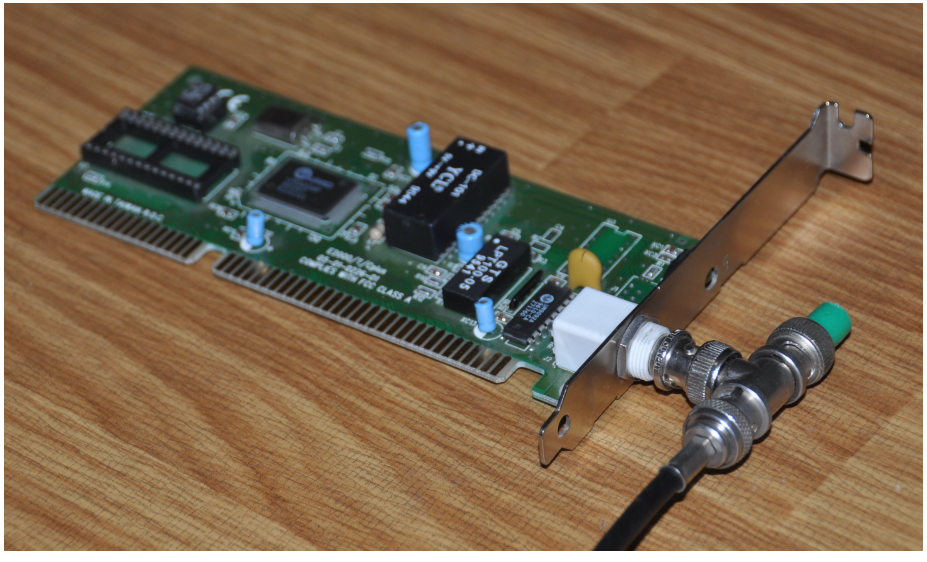
7.3 Beispiele

- zwischen Netzwerkkarte und Netzwerkkarte:
  seriell, mit Adressierung
  - Token Ring: ringförmig

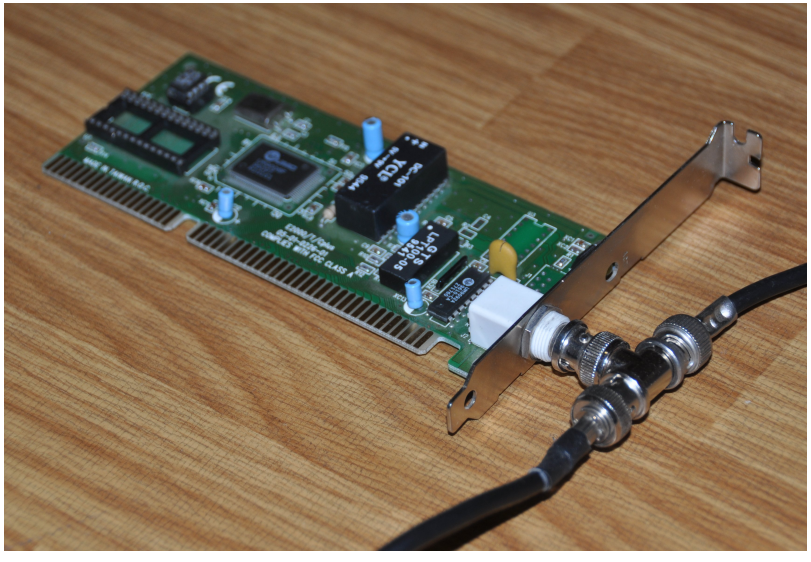
7 Bus-Systeme

7.3 Beispiele

- zwischen Netzwerkkarte und Netzwerkkarte:
  seriell, mit Adressierung
  - Token Ring: ringförmig
  - BNC-Ethernet: linear, terminiert



Endpunkt: terminiert

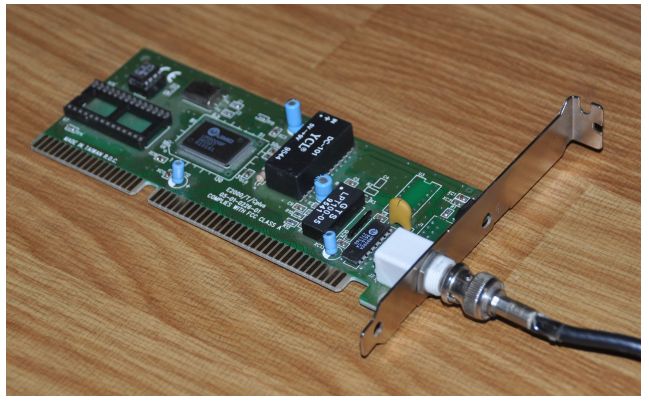


Mitte: vorbeiführen

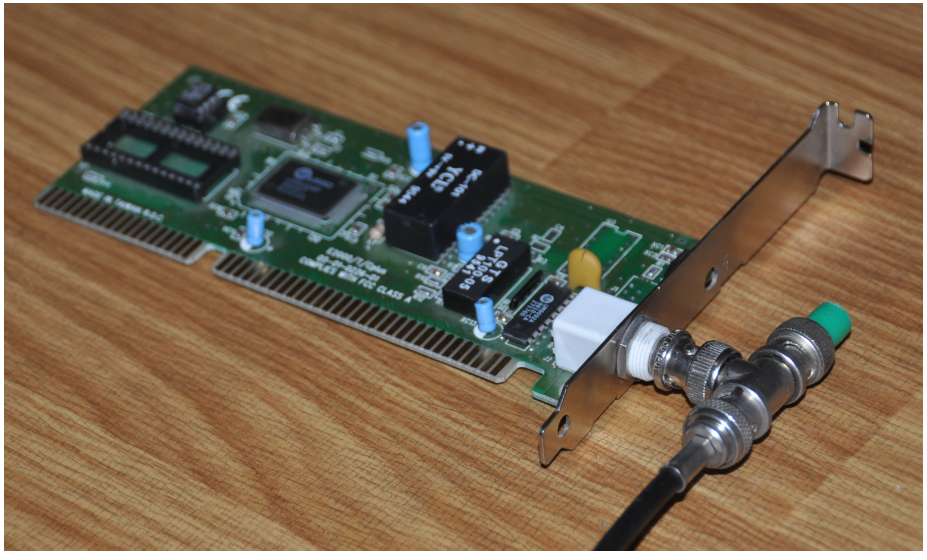
7 Bus-Systeme

7.3 Beispiele

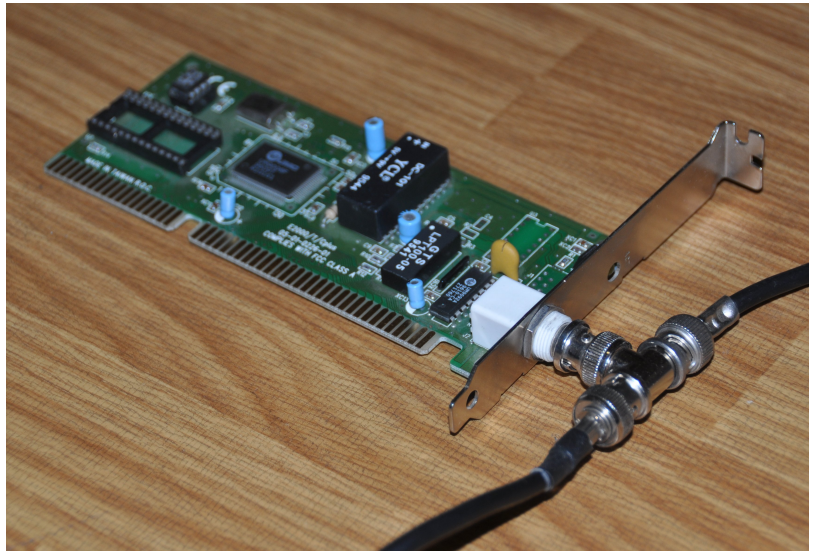
- zwischen Netzwerkkarte und Netzwerkkarte: seriell, mit Adressierung
  - Token Ring: ringförmig
  - BNC-Ethernet: linear, terminiert



falsch



Endpunkt: terminiert

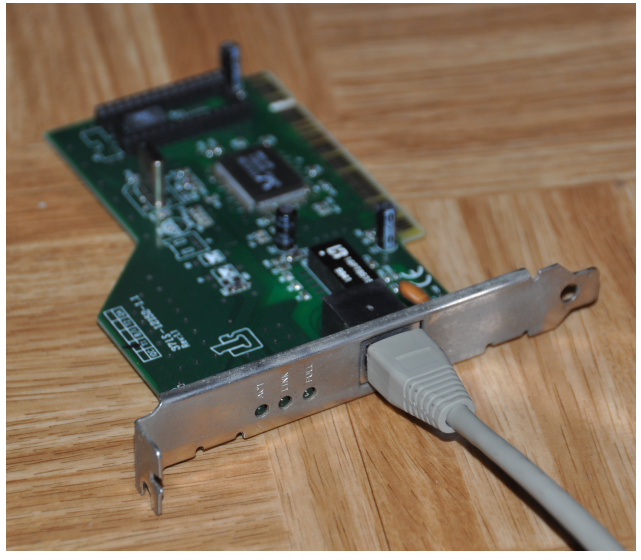


Mitte: vorbeiführen

7 Bus-Systeme

7.3 Beispiele

- zwischen Netzwerkkarte und Netzwerkkarte: seriell, mit Adressierung
  - Token Ring: ringförmig
  - BNC-Ethernet: linear, terminiert
  - Twisted-Pair-Ethernet: Punkt-zu-Punkt, mit Hubs/Switches: sternförmig



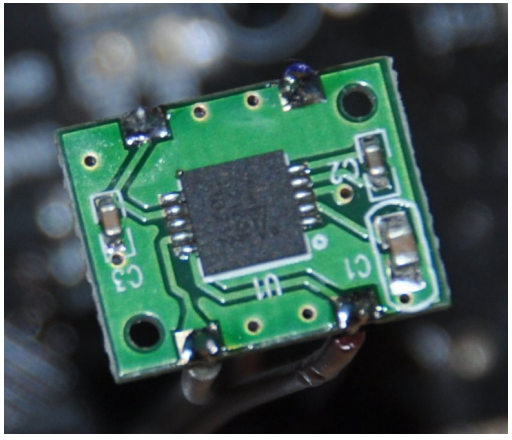
7 Bus-Systeme

7.3 Beispiele

- zwischen Netzwerkkarte und Netzwerkkarte: seriell, mit Adressierung
  - Token Ring: ringförmig
  - BNC-Ethernet: linear, terminiert
  - Twisted-Pair-Ethernet: Punkt-zu-Punkt, mit Hubs/Switches: sternförmig
  - WLAN: sternförmig (Access-Point), im Ad-Hoc-Modus: beliebige Topologie

7 Bus-Systeme

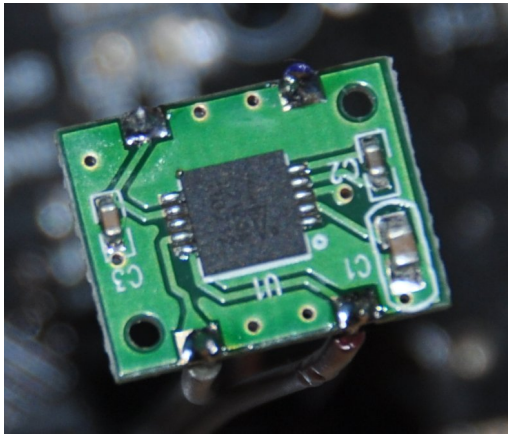
7.4 Beispiel: Benutzung des I²C-Busses



```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses

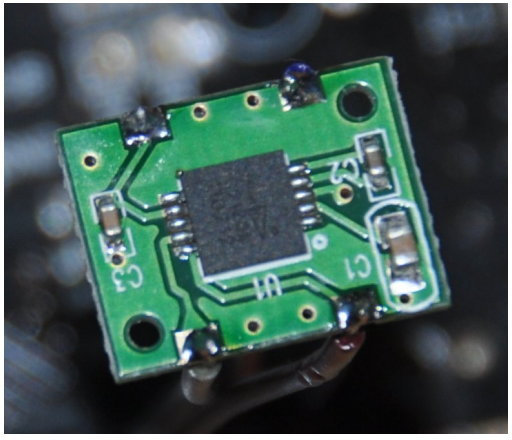


```
void read_compass (uint16_t *x, uint16_t *y)
{ ... }

int main (void)
{
   uint16_t compass_x, compass_y;
   read_compass (&compass_x, &compass_y);
   writeInteger (compass_x, DEC);
   writeChar (' ');
   writeInteger (compass_y, DEC);
   writeChar ('\n');
   return 0;
}
```

7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses

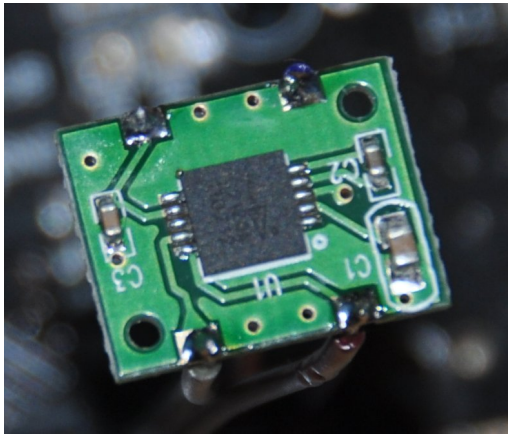


```
void read_compass (uint16_t *x, uint16_t *y)
{ ... }

int main (void)
{
   uint16_t compass_x, compass_y;
   read_compass (&compass_x, &compass_y);
   writeInteger (compass_x, DEC);
   writeChar (' ');
   writeInteger (compass_y, DEC);
   writeChar ('\n');
   return 0;
}
```

7 Bus-Systeme

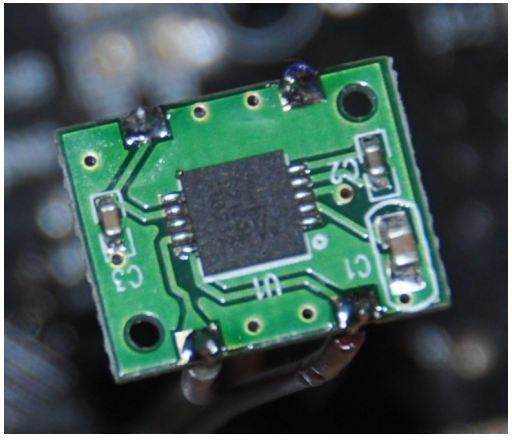
7.4 Beispiel: Benutzung des I²C-Busses



```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses

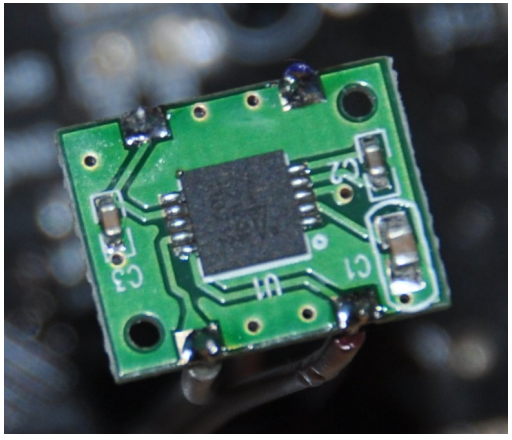


```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

Gerät adressieren:
Schreib-Adresse

7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



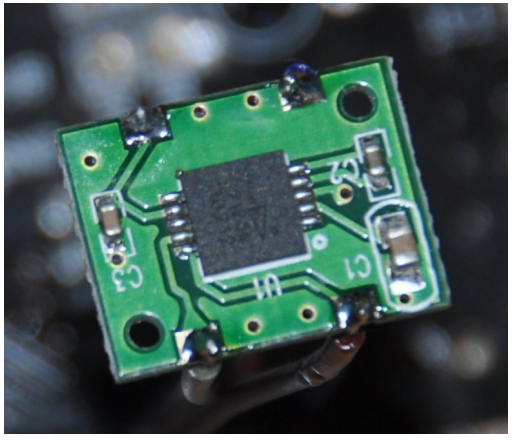
```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

Gerät adressieren:
Schreib-Adresse

Register-Nr.
innerhalb des Geräts

7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

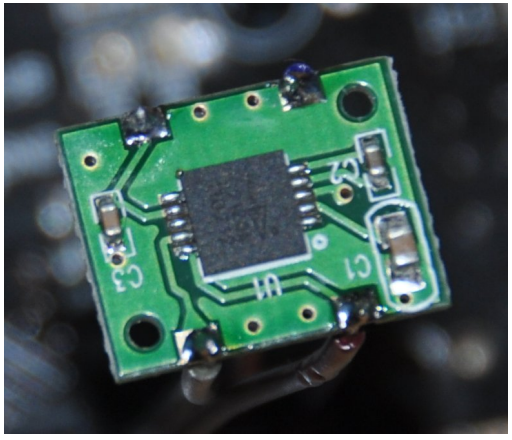
Gerät adressieren: Schreib-Adresse

Register-Nr. innerhalb des Geräts

Befehl: Spule an

7 Bus-Systeme

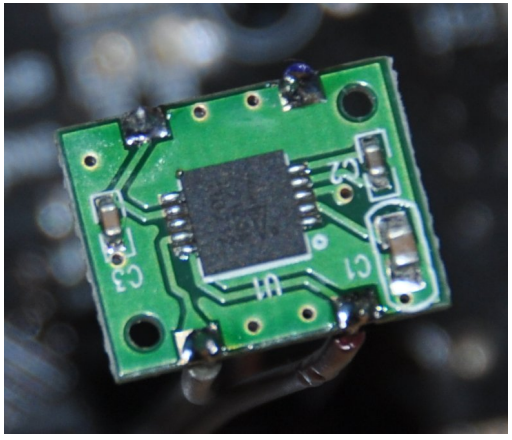
7.4 Beispiel: Benutzung des I²C-Busses



```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

7 Bus-Systeme

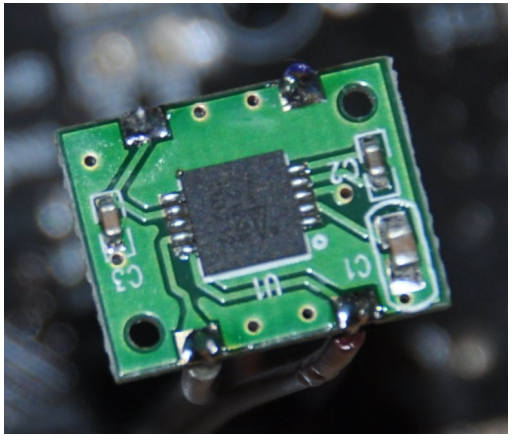
7.4 Beispiel: Benutzung des I²C-Busses



```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

7 Bus-Systeme

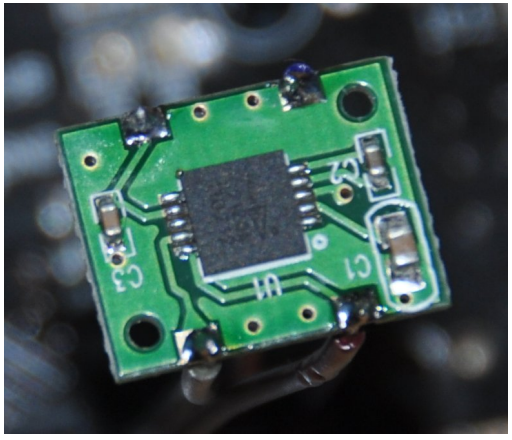
7.4 Beispiel: Benutzung des I²C-Busses



```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

7 Bus-Systeme

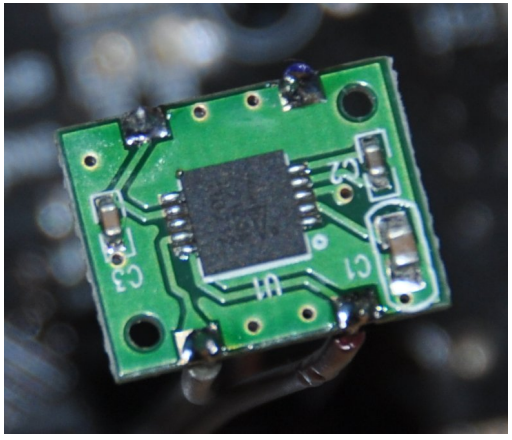
7.4 Beispiel: Benutzung des I²C-Busses



```
void read_compass (uint16_t *x, uint16_t *y)
{
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x02); // set coil
  mSleep (1);
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x04); // reset coil
  mSleep (5);
  uint8_t result[5];
  I2CTWI_transmit2Bytes (0x60, 0x00, 0x01); // Messung starten
  mSleep (5); // 5ms warten, bis Sensor fertig gemessen hat
  I2CTWI_transmitByte (0x60, 0x01); // Leseindex setzen
  I2CTWI_readBytes (0x61, result, 4); // lesen: msb x, lsb x, msb y, lsb y
  result[0] &= 0b00001111; // Unwichtge Bits vom msb abschneiden
  result[2] &= 0b00001111;
  *x = (result[0] << 8) + result[1]; // Wert berechnen aus msb und lsb
  *y = (result[2] << 8) + result[3];
}
```

7 Bus-Systeme

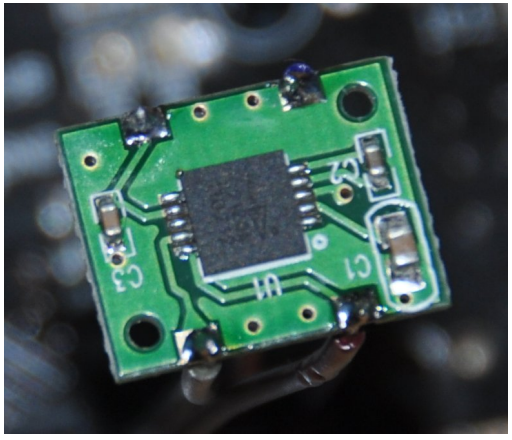
7.4 Beispiel: Benutzung des I²C-Busses



Messung:
Roboter im Kreis drehen lassen,
x- und y-Werte aufzeichnen
und gegeneinander auftragen

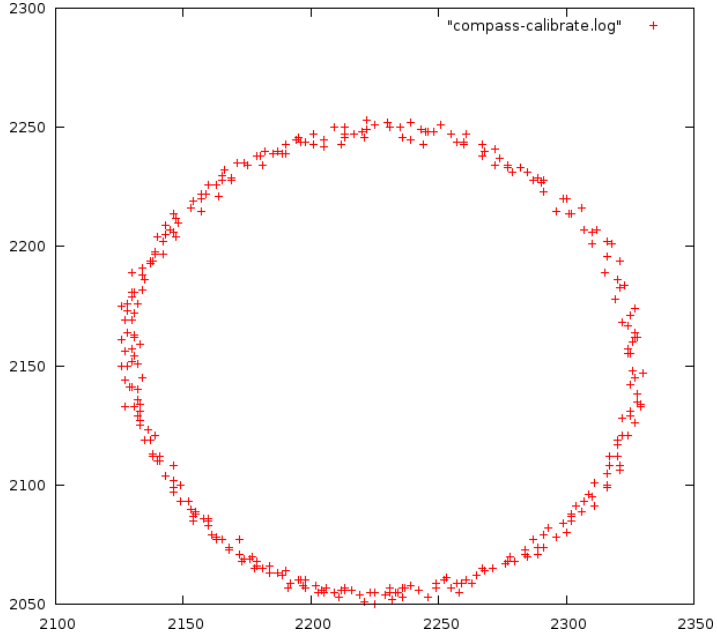
7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



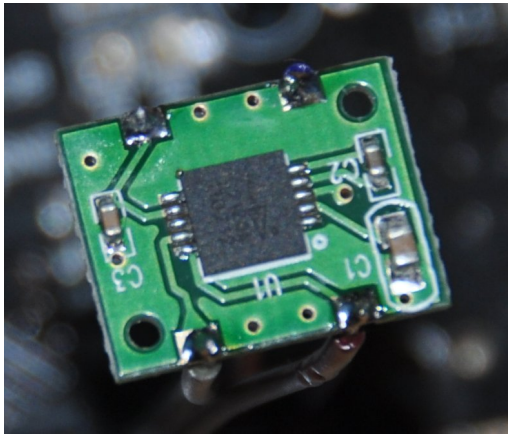
Messung:
Roboter im Kreis drehen lassen,
x- und y-Werte aufzeichnen
und gegeneinander auftragen

Ergebnis: Ellipse



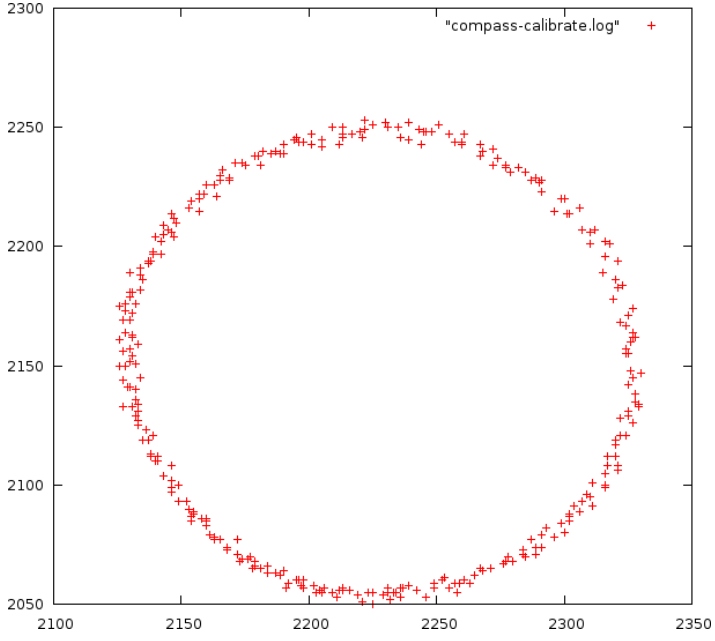
7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



Messung:
Roboter im Kreis drehen lassen,
x- und y-Werte aufzeichnen
und gegeneinander auftragen

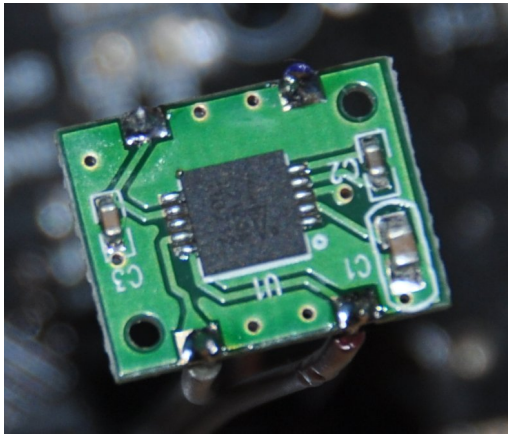
Ergebnis: Ellipse



Anwendung:
Roboter in eine Himmelsrichtung ausrichten

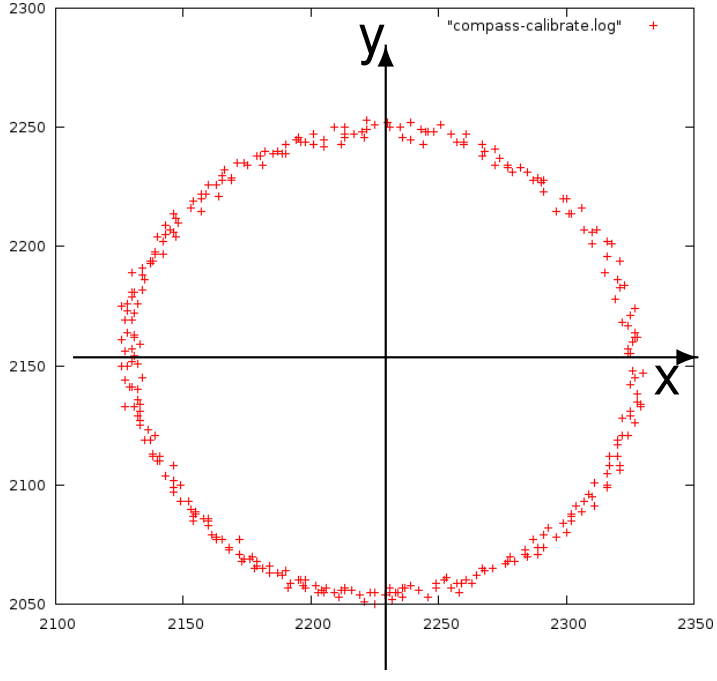
7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



Messung:
Roboter im Kreis drehen lassen,
x- und y-Werte aufzeichnen
und gegeneinander auftragen

Ergebnis: Ellipse

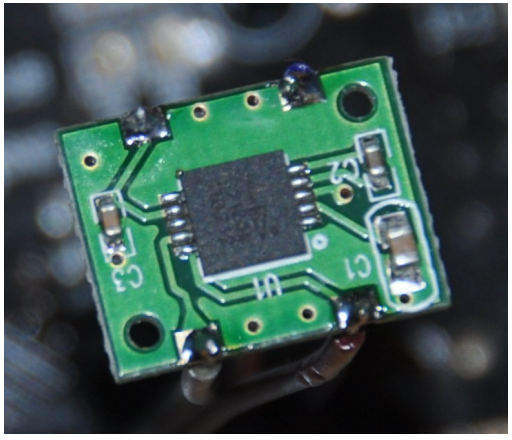


Anwendung:
Roboter in eine Himmelsrichtung ausrichten

- Achsen normieren:
  Mittelwert subtrahieren

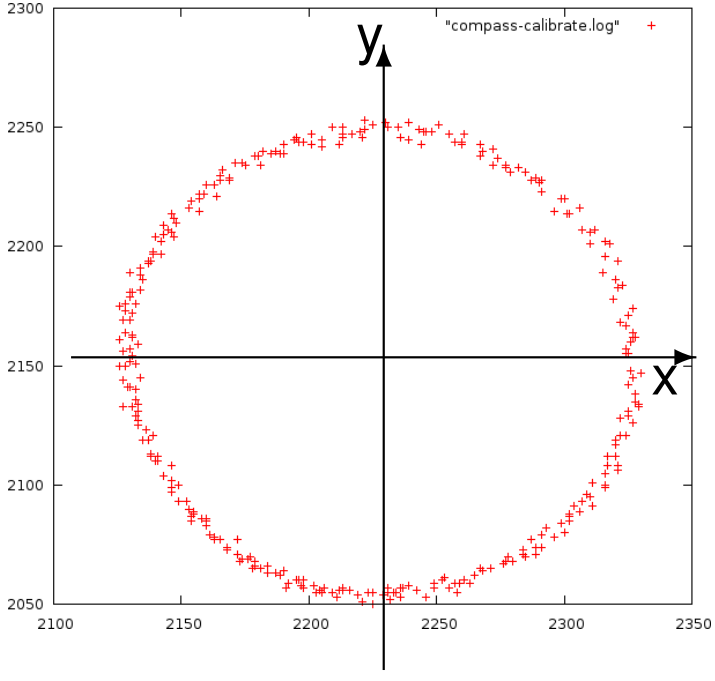
7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



Messung:
Roboter im Kreis drehen lassen,
x- und y-Werte aufzeichnen
und gegeneinander auftragen

Ergebnis: Ellipse

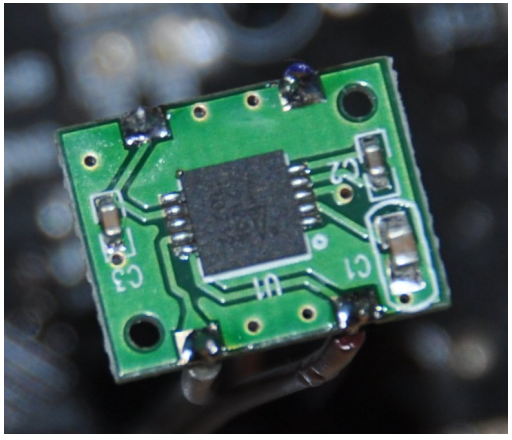


Anwendung:
Roboter in eine Himmelsrichtung ausrichten

- Achsen normieren:
  Mittelwert subtrahieren
- Nulldurchgang einer Achse
  markiert eine Himmelsrichtung

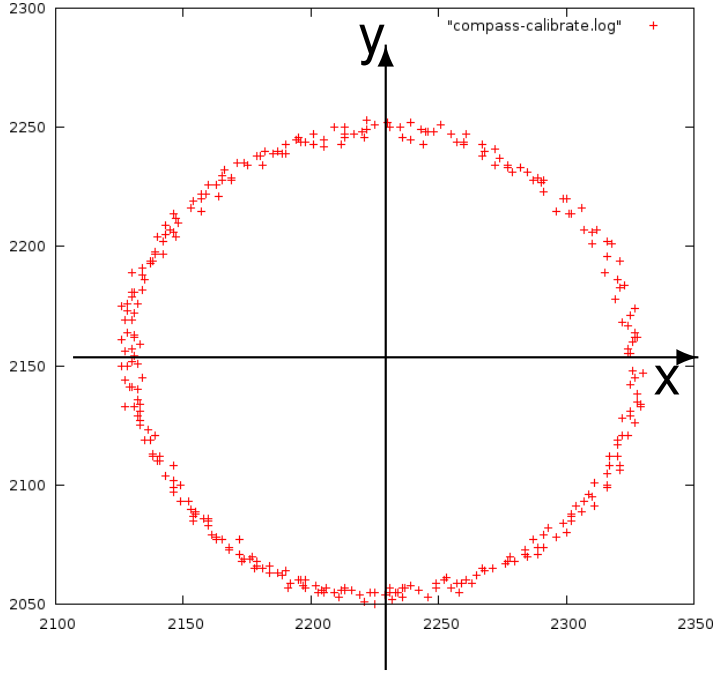
7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



Messung:
Roboter im Kreis drehen lassen,
x- und y-Werte aufzeichnen
und gegeneinander auftragen

Ergebnis: Ellipse

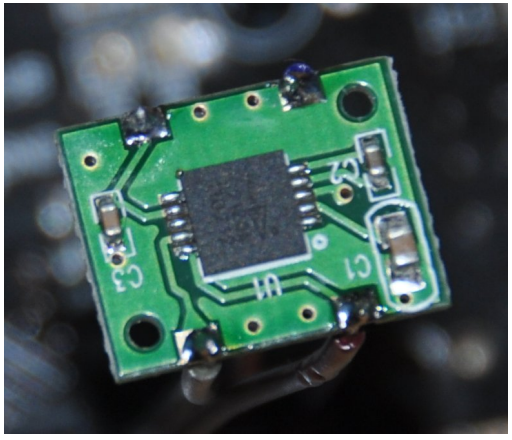


Anwendung:
Roboter in eine Himmelsrichtung ausrichten

- Achsen normieren:
  Mittelwert subtrahieren
- Nulldurchgang einer Achse
  markiert eine Himmelsrichtung
- Vorzeichen der anderen Achse
  sagt aus, welche Himmelsrichtung

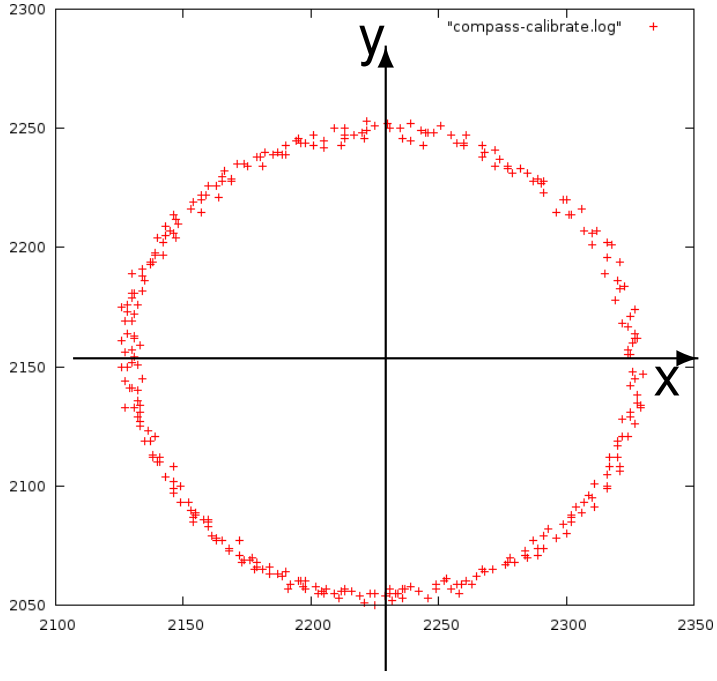
7 Bus-Systeme

7.4 Beispiel: Benutzung des I²C-Busses



Messung:
Roboter im Kreis drehen lassen,
x- und y-Werte aufzeichnen
und gegeneinander auftragen

Ergebnis: Ellipse



Anwendung:
Roboter in eine Himmelsrichtung ausrichten

- Achsen normieren:
  Mittelwert subtrahieren
- Nulldurchgang einer Achse
  markiert eine Himmelsrichtung
- Vorzeichen der anderen Achse
  sagt aus, welche Himmelsrichtung

Beispiel: nach Norden ausrichten
Grob drehen, bis y positiv ist, fein drehen, bis $x = 0$ ist

Grundlagen Rechnertechnik

8 Pipelining

8 Pipelining

8.1 Konzept

- Aufgabe in Teilaufgaben zerlegen
- Teilaufgaben parallel ausführen

8 Pipelining

8.1 Konzept

- Aufgabe in Teilaufgaben zerlegen
- Teilaufgaben parallel ausführen

Beispiel: Wäsche waschen

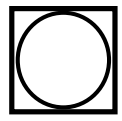
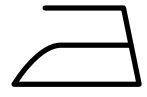
8 Pipelining

8.1 Konzept

- Aufgabe in Teilaufgaben zerlegen
- Teilaufgaben parallel ausführen

Beispiel: Wäsche waschen

- Teilaufgaben: 

,  , 

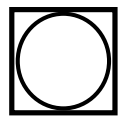
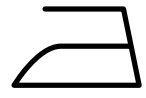
8 Pipelining

8.1 Konzept

- Aufgabe in Teilaufgaben zerlegen
- Teilaufgaben parallel ausführen

Beispiel: Wäsche waschen

- Teilaufgaben: 

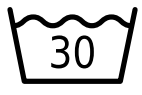
,  , 
- müssen nacheinander ausgeführt werden

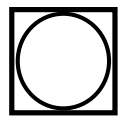
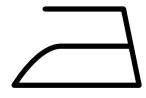
8 Pipelining

8.1 Konzept

- Aufgabe in Teilaufgaben zerlegen
- Teilaufgaben parallel ausführen

Beispiel: Wäsche waschen

- Teilaufgaben: 

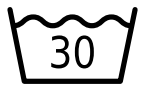
,  , 
- müssen nacheinander ausgeführt werden: Datenfluß

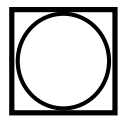
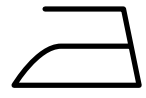
8 Pipelining

8.1 Konzept

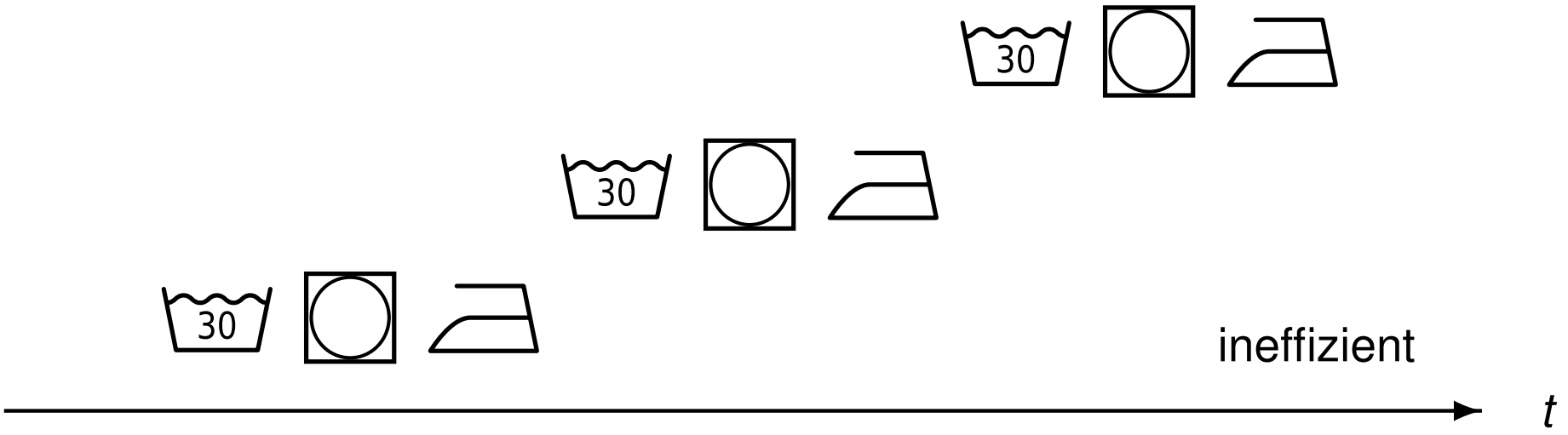
- Aufgabe in Teilaufgaben zerlegen
- Teilaufgaben parallel ausführen

Beispiel: Wäsche waschen

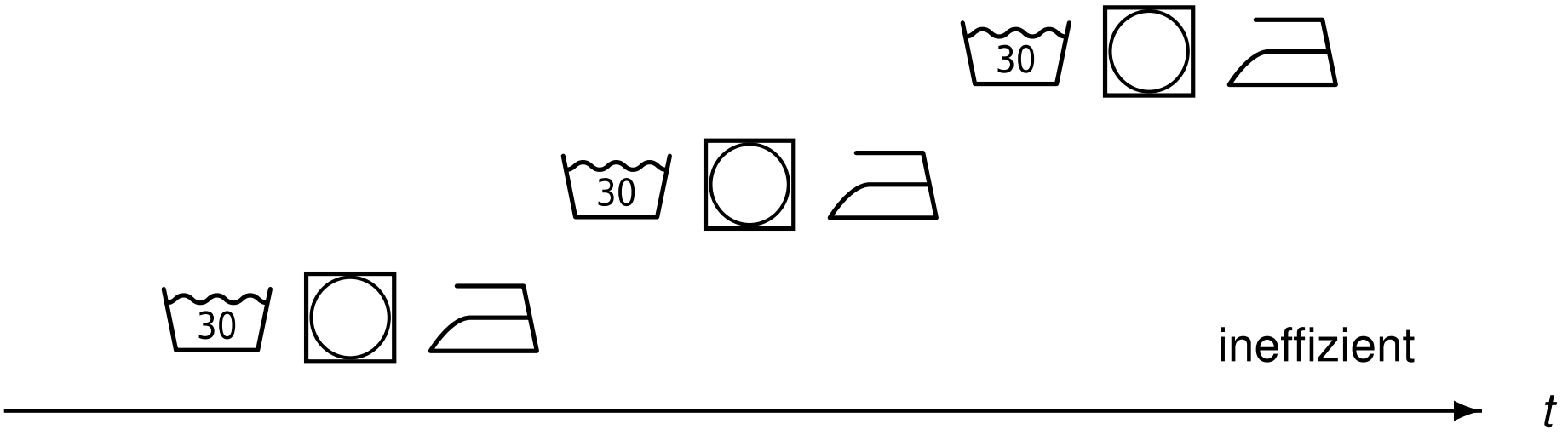
- Teilaufgaben: 

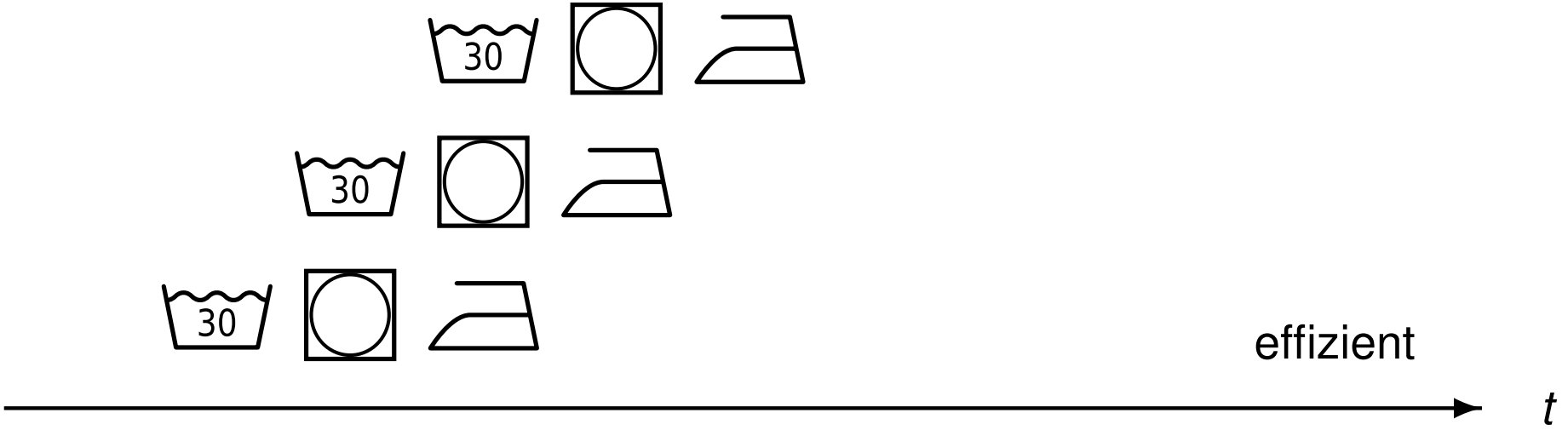
,  , 
- müssen nacheinander ausgeführt werden: Datenfluß
- belegen jeweils 1 Ressource

3 Ladungen Wäsche

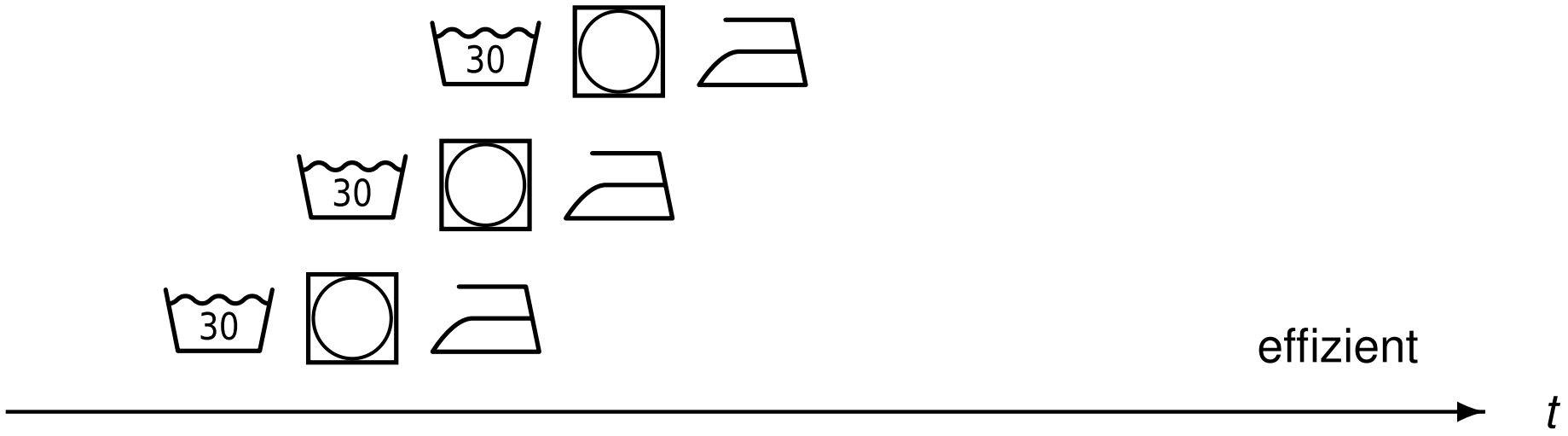


3 Ladungen Wäsche

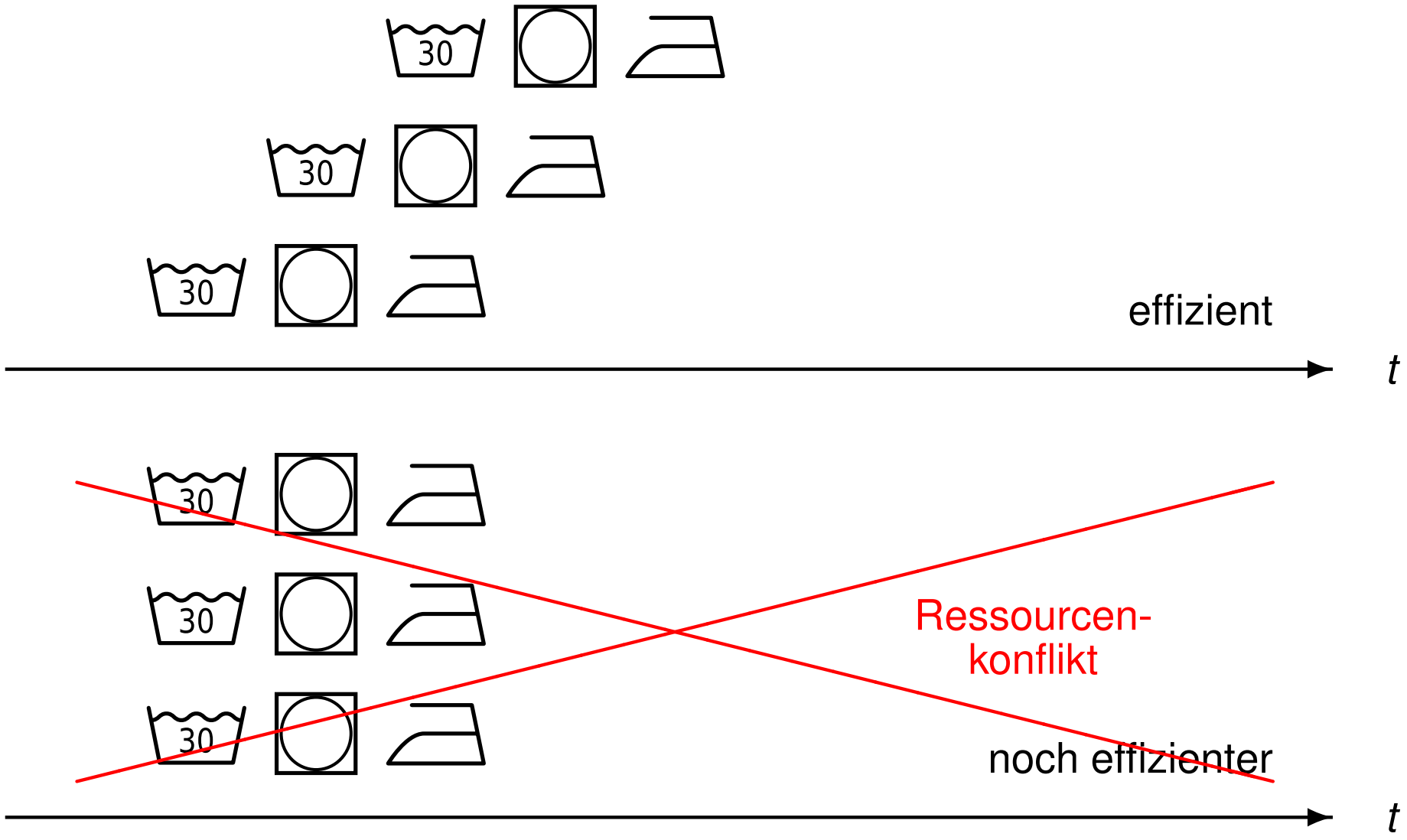




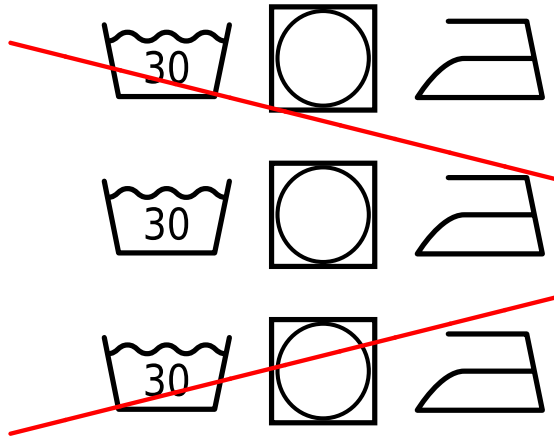
3 Ladungen Wäsche



3 Ladungen Wäsche



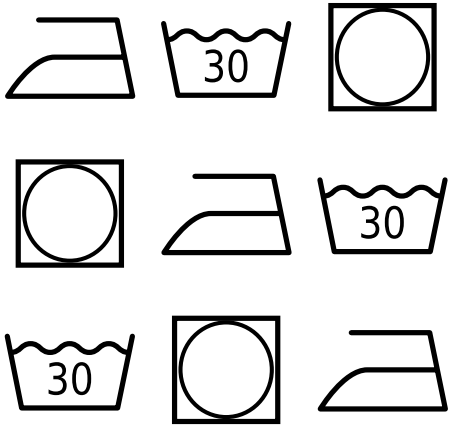
3 Ladungen Wäsche



Ressourcen-
konflikt

noch effizienter

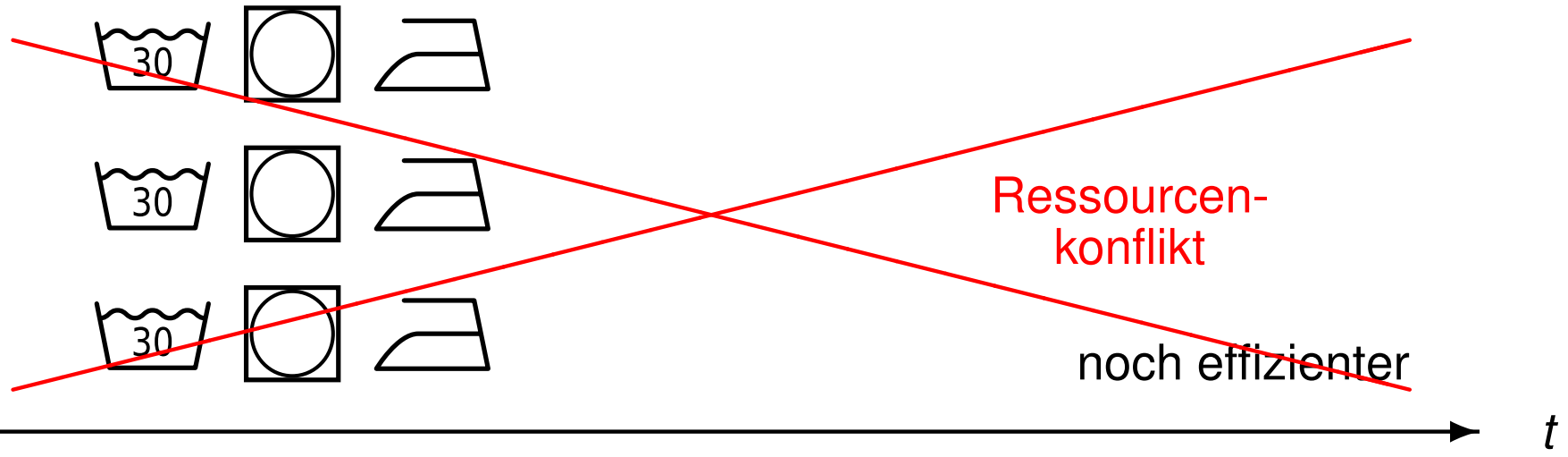
t

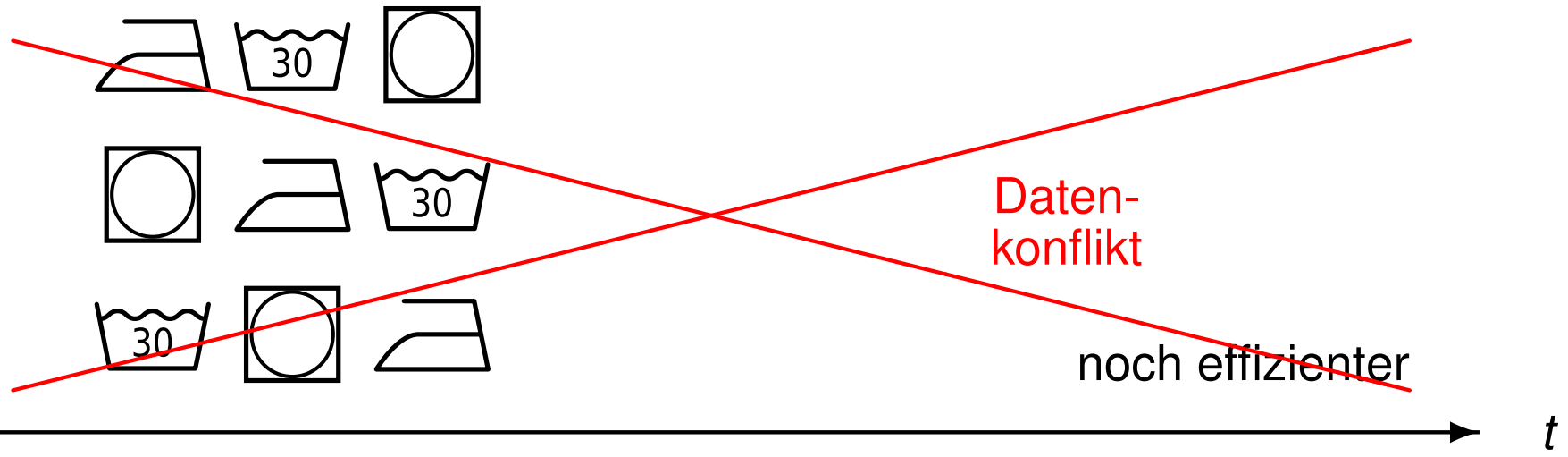


noch effizienter

t

3 Ladungen Wäsche





8.2 Arithmetik-Pipelines

„Register-FIFO“

8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$

8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$

8.2 Arithmetik-Pipelines

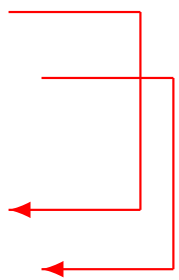
„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$
push $a_2 \cdot b_2$



8.2 Arithmetik-Pipelines

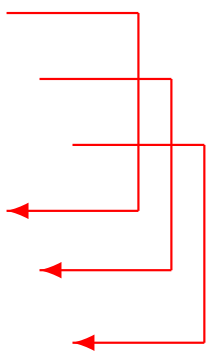
„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$
push $a_2 \cdot b_2$
push $a_3 \cdot b_3$



8.2 Arithmetik-Pipelines

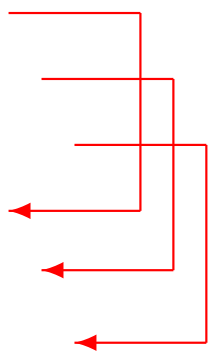
„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$
push $a_2 \cdot b_2$
push $a_3 \cdot b_3$
s_1 = pop



8.2 Arithmetik-Pipelines

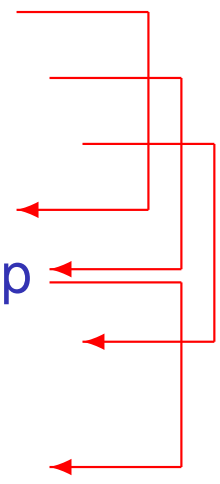
„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$
push $a_2 \cdot b_2$
push $a_3 \cdot b_3$
s_1 = pop
push $s_1 +$ pop



8.2 Arithmetik-Pipelines

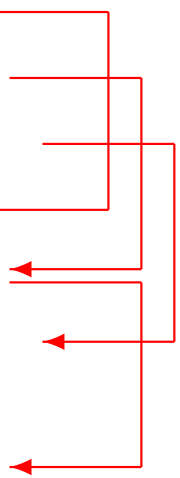
„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3

push $a_1 \cdot b_1$
push $a_2 \cdot b_2$
push $a_3 \cdot b_3$
s_1 = pop
push $s_1 +$ pop
s_3 = pop



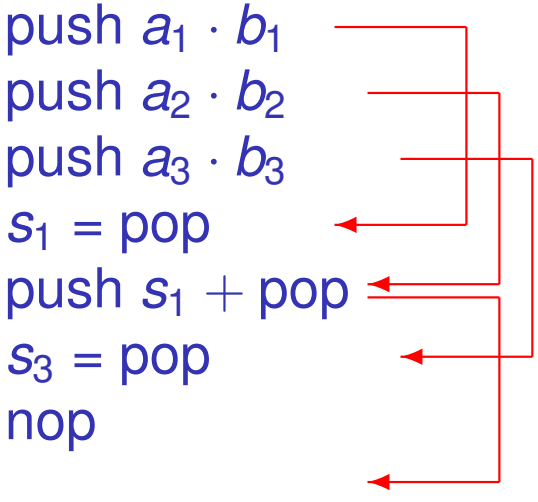
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



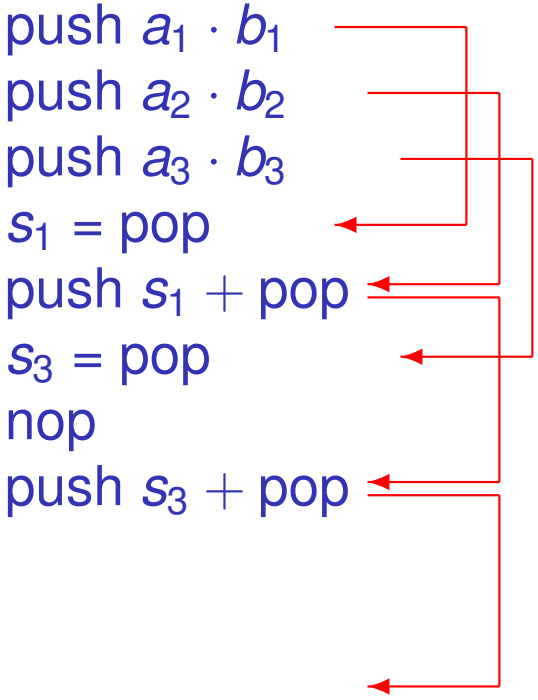
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



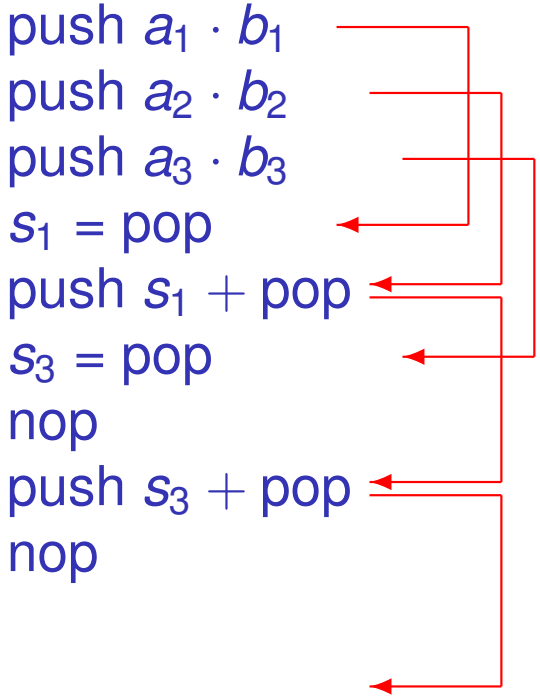
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



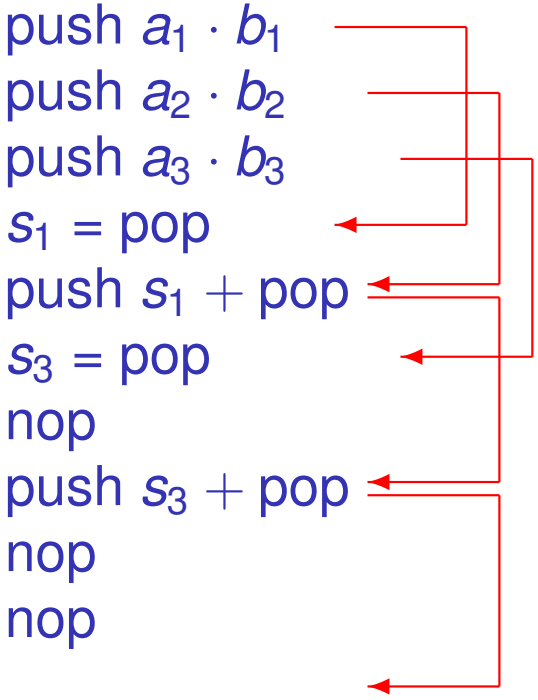
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



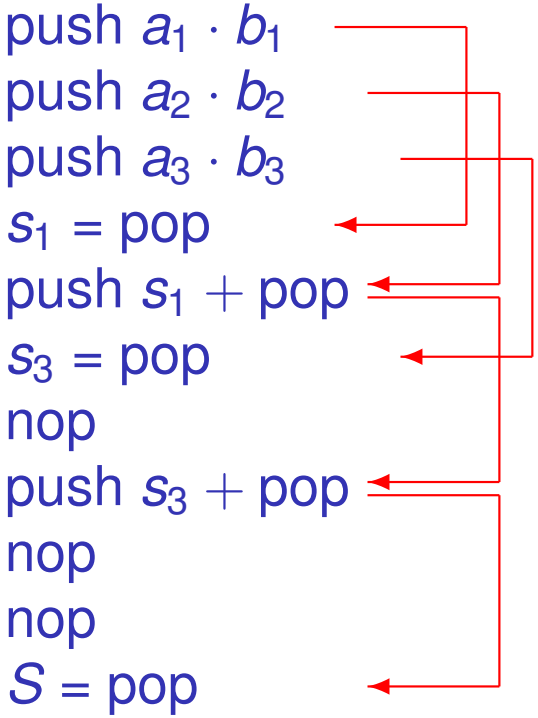
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



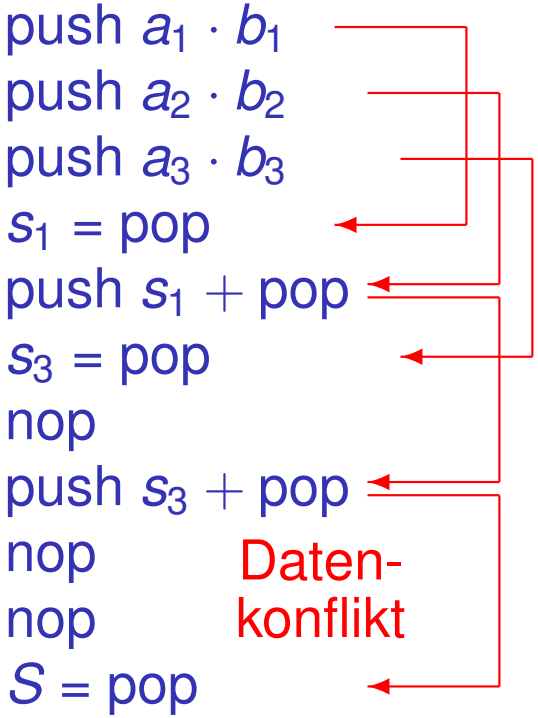
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



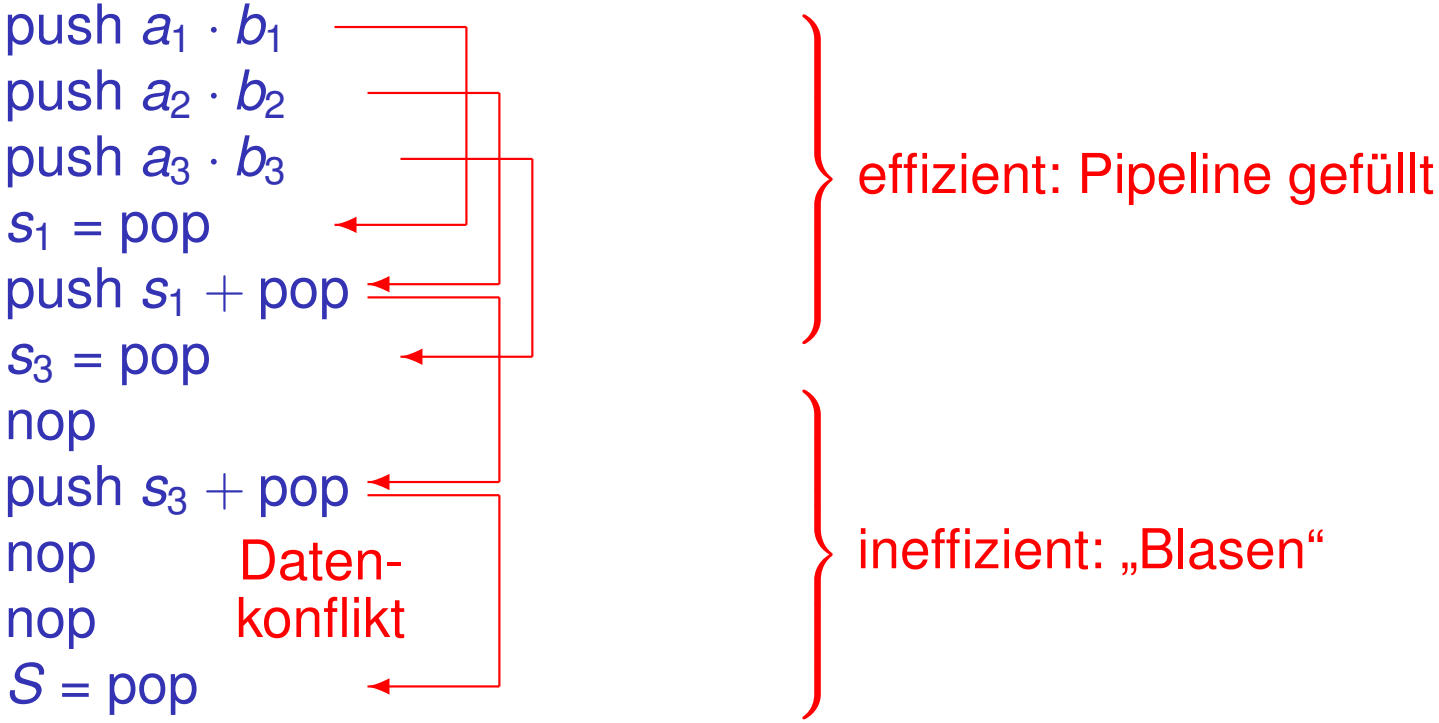
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



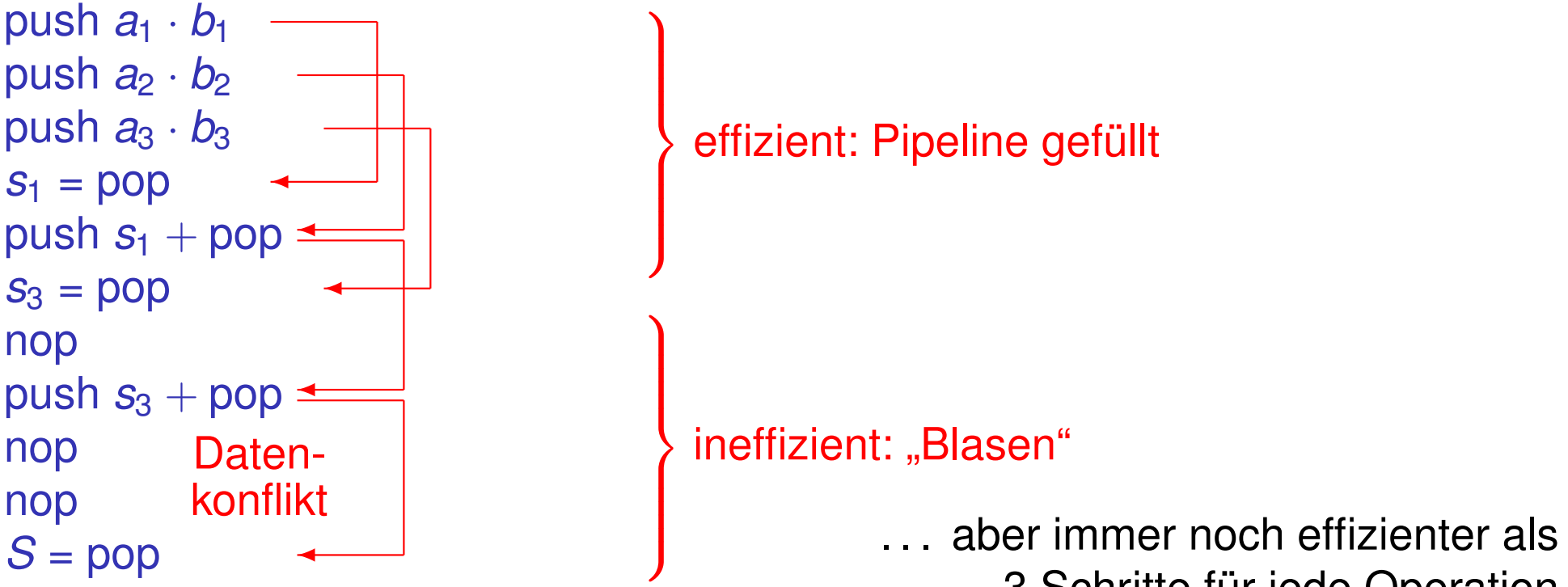
8.2 Arithmetik-Pipelines

„Register-FIFO“

Pseudo-Code: Berechnung von

$$S = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 + a_3 \cdot b_3$$

mit einer Pipeline der Länge 3



Reales Beispiel: Vektor-Addition auf i860

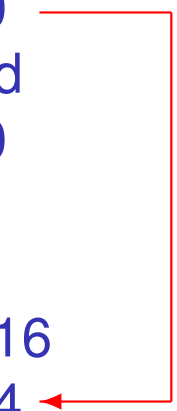
```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```

Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```

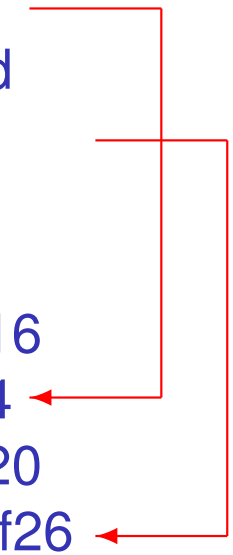
Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



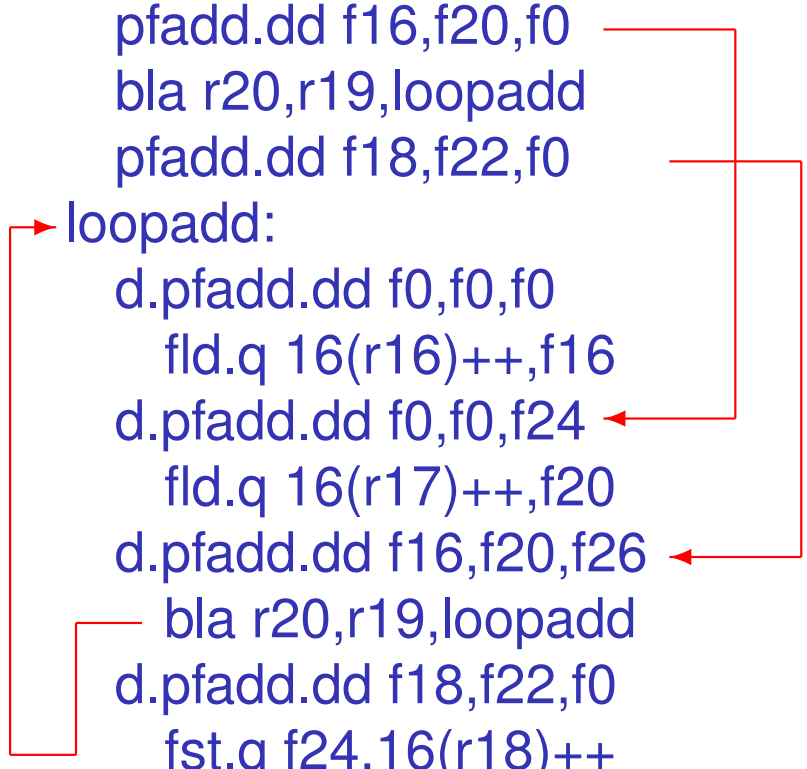
Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



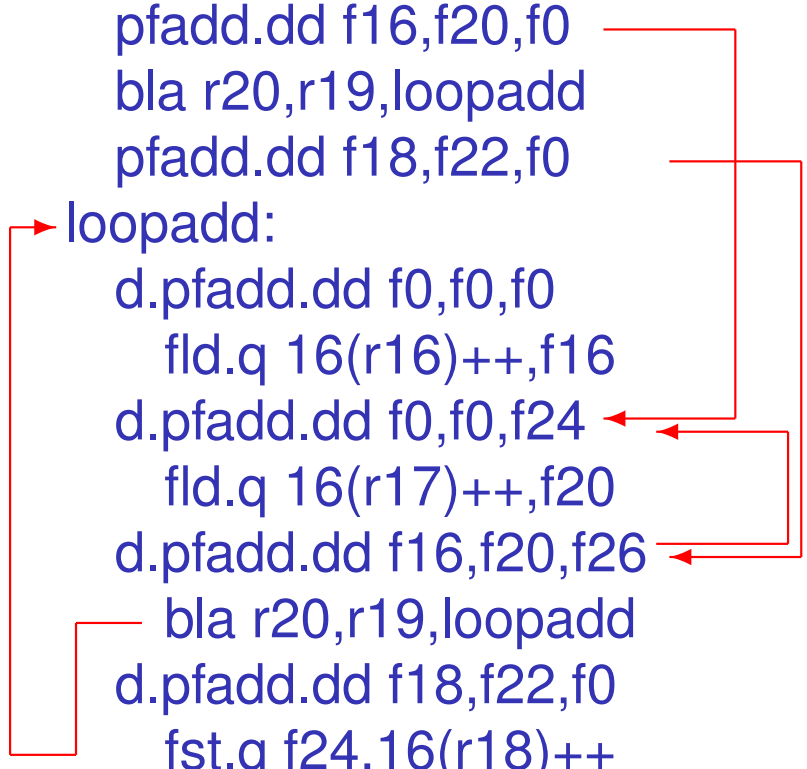
Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



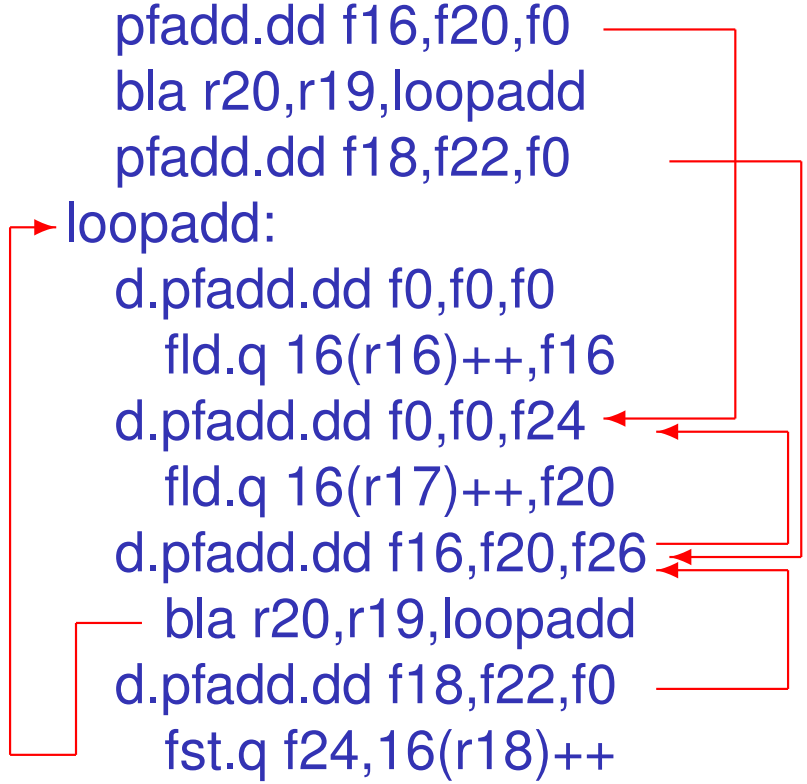
Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



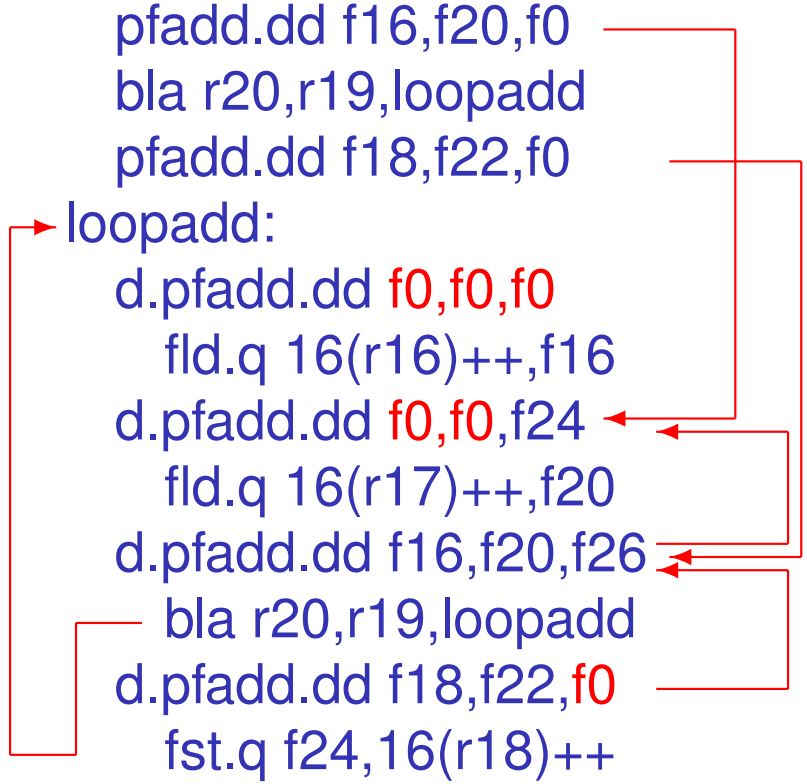
Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



6mal f0 = 2 Blasen

Reales Beispiel: Vektor-Addition auf i860

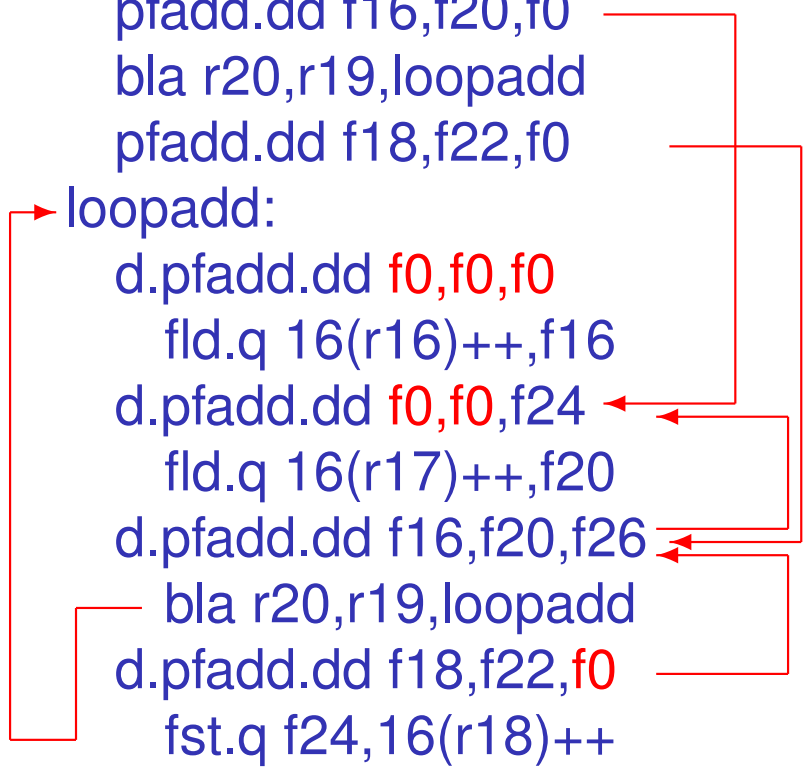
```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```

6mal f0 = 2 Blasen

Immerhin: 2 Additionen in 4 Taktzyklen

Reales Beispiel: Vektor-Addition auf i860

```
    .align 8
.globl _vadd
    nop
_vadd:
    shr 1,r19,r19
    bte r19,r0,exitadd
    addu 0x000F,r16,r16
    andnot 0x000F,r16,r16
    adds -16,r16,r16
    addu 0x000F,r17,r17
    andnot 0x000F,r17,r17
    adds -16,r17,r17
    addu 0x000F,r18,r18
    andnot 0x000F,r18,r18
    adds -16,r18,r18
    mov -1,r20
    fld.q 16(r16)++,f16
    fld.q 16(r17)++,f20
    pfadd.dd f16,f20,f0
    bla r20,r19,loopadd
    pfadd.dd f18,f22,f0
loopadd:
    d.pfadd.dd f0,f0,f0
      fld.q 16(r16)++,f16
    d.pfadd.dd f0,f0,f24
      fld.q 16(r17)++,f20
    d.pfadd.dd f16,f20,f26
      bla r20,r19,loopadd
    d.pfadd.dd f18,f22,f0
      fst.q f24,16(r18)++
    nop
    nop
    nop
exitadd:
    bri r1
    nop
```



6mal f0 = 2 Blasen

Immerhin: 2 Additionen in 4 Taktzyklen

Dies ist ein *einfaches* Beispiel.

8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

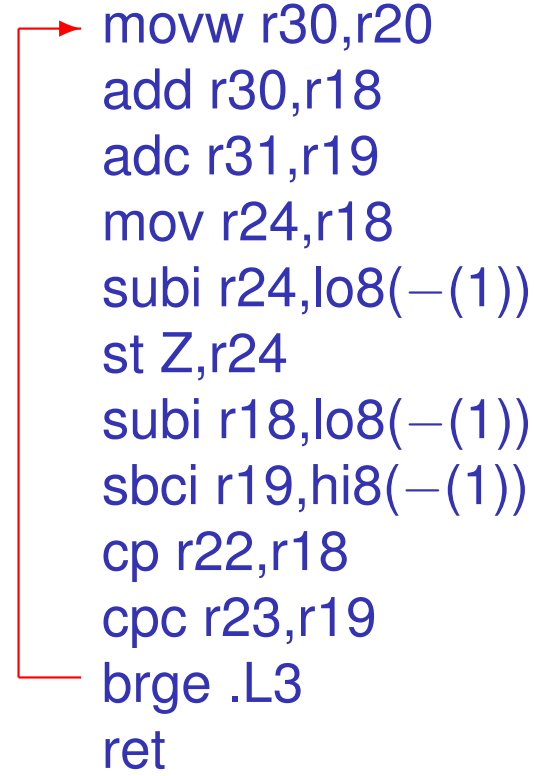
8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

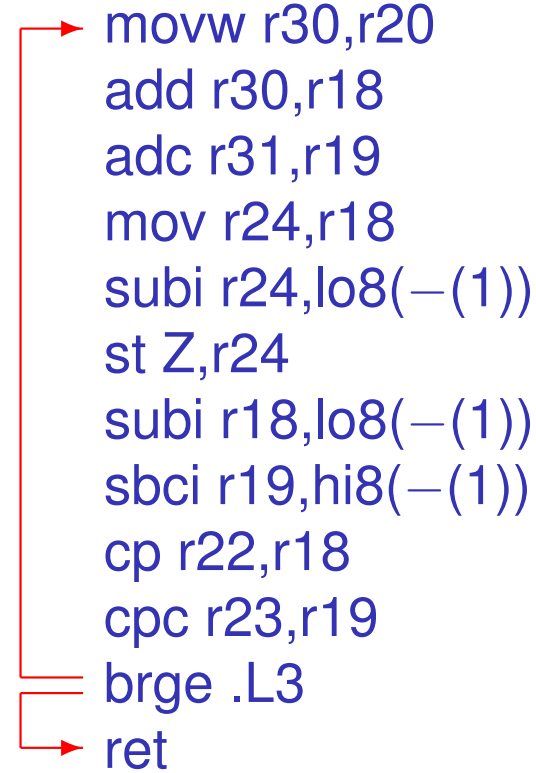
8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen



8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen



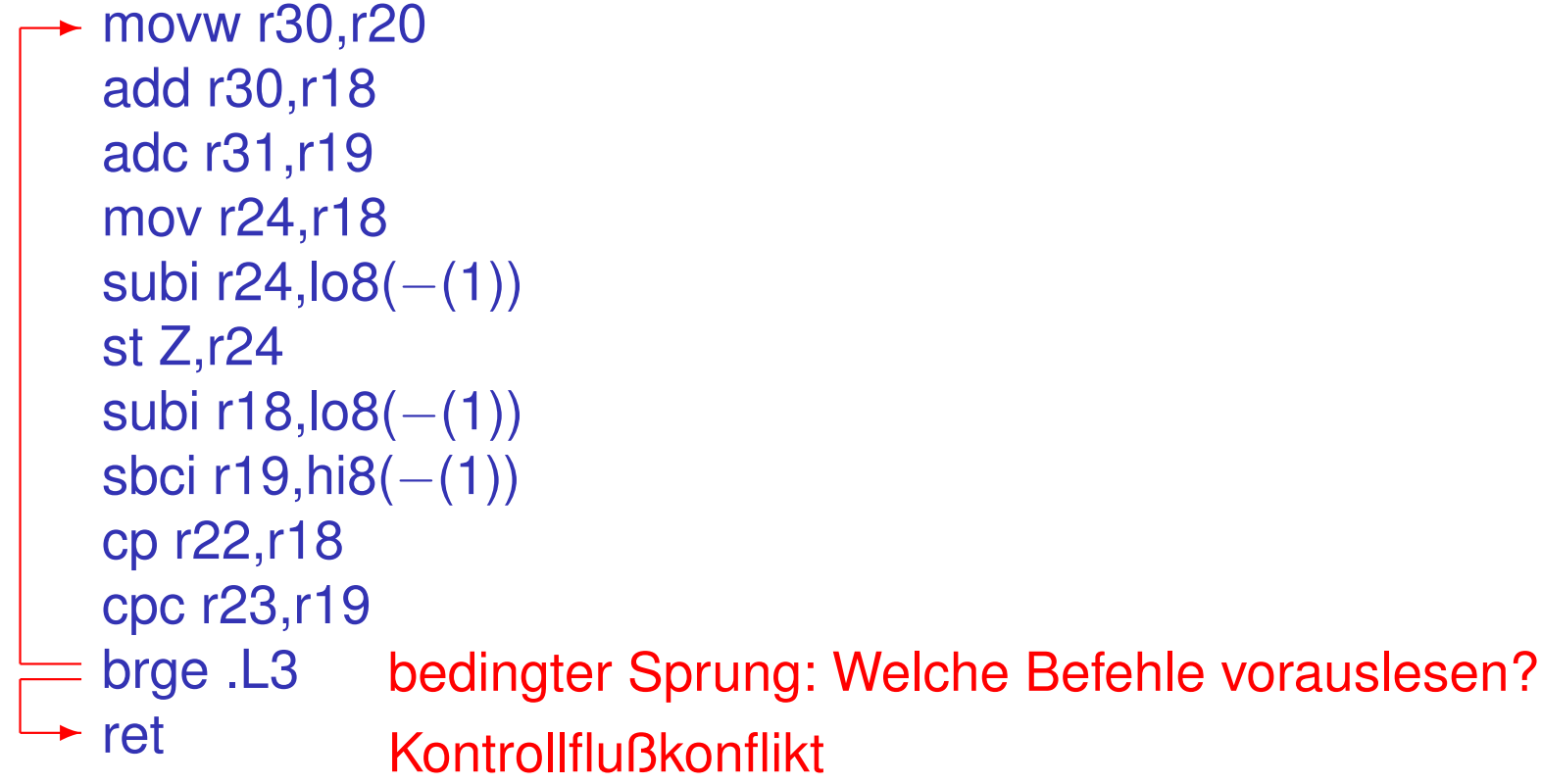
8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3      bedingter Sprung: Welche Befehle vorauslesen?
    ret
```

8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen



8.3 Instruktions-Pipelines

Ein Prozessor benötigt Zeit, um einen Befehl zu verstehen.
$\longrightarrow$ Während Befehlsausführung nächste Befehle vorauslesen

```
.L3:
    movw r30,r20
    add r30,r18
    adc r31,r19
    mov r24,r18
    subi r24,lo8(-(1))
    st Z,r24
    subi r18,lo8(-(1))
    sbci r19,hi8(-(1))
    cp r22,r18
    cpc r23,r19
    brge .L3
    ret
```

bedingter Sprung: Welche Befehle vorauslesen?

Kontrollflußkonflikt

Lösungsansatz: Zweigvorhersage

Zweigvorhersage – Branch Prediction

Zweigvorhersage – Branch Prediction

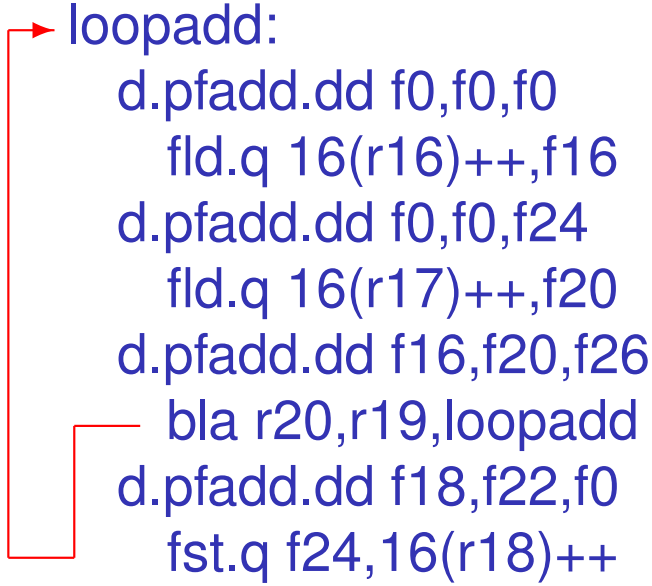
- Sprünge nach oben sind Schleifen: „Ja“
  Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“

Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
  Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
  $\longrightarrow$ Optimierung manuell oder durch Compiler

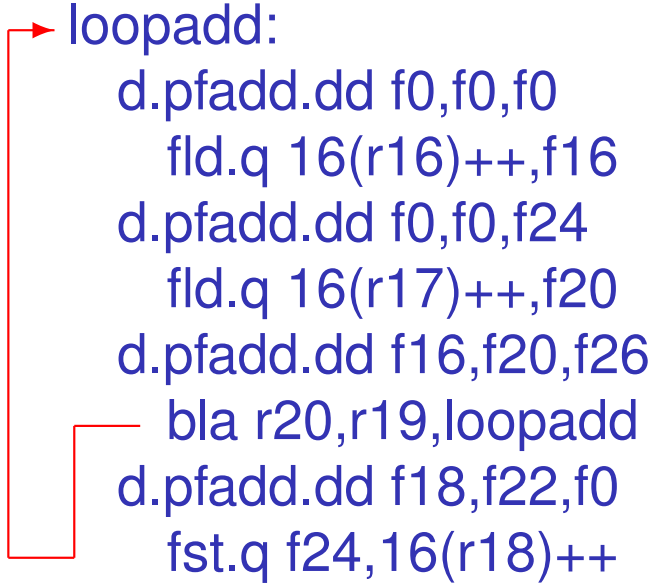
Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
  Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
  $\longrightarrow$ Optimierung manuell oder durch Compiler



Zweigvorhersage – Branch Prediction

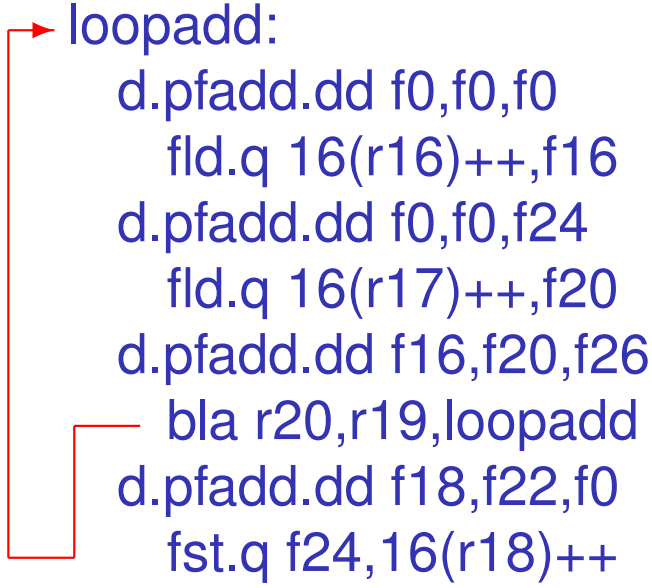
- Sprünge nach oben sind Schleifen: „Ja“
  Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
  $\longrightarrow$ Optimierung manuell oder durch Compiler



- Branch History Table: Sprünge merken

Zweigvorhersage – Branch Prediction

- Sprünge nach oben sind Schleifen: „Ja“
  Sprünge nach unten sind Auswahl-Verzweigungen: „Nein“
- Delayed Branches: Sprungbefehl verspätet ausführen
  $\longrightarrow$ Optimierung manuell oder durch Compiler



- Branch History Table: Sprünge merken
- ...

8 Pipelining

Zusammenfassung

- Teilaufgaben parallel ausführen
- Arithmetik-Pipelines führen Berechnungen parallel aus, Instruktions-Pipelines lesen Befehle voraus
- Ressourcen-, Daten- und Kontrollflußkonflikte führen zu „Blasen“
- Zweigvorhersage reduziert Kontrollflußkonflikte in Instruktions-Pipelines
  - nach oben / nach unten
  - Delayed Branches: manuell optimieren
  - Branch History Table: Sprünge merken

Grundlagen Rechnertechnik

Grundlagen Rechnertechnik

9 Ausblick

- Explicitly Parallel Instruction Computing (EPIC)
- Field Programmable Gate Arrays (FPGAs)
- C-to-Hardware-Synthese