Einführung

Adrian Schüpbach

adrian_laurent.schuepbach@alumni.ethz.ch

Algorithmus, Programmierkonzept, Programmiersprache, Programm

- Algorithmus: Lösungsstrategie
  - Unabhängig von der Programmiersprache
- Konzept: Modell, um Code und Daten zu organisieren
  - Imperativ
  - Objektorientiert
- Sprache: Werkzeug, um den Algorithmus in ausführbare Maschinenbefehle zu übersetzen
- Programm: Sammlung von Befehlen und Daten
  - Programm ist Produkt, das auf PC ausgeführt wird

Programmieren

Zyklus

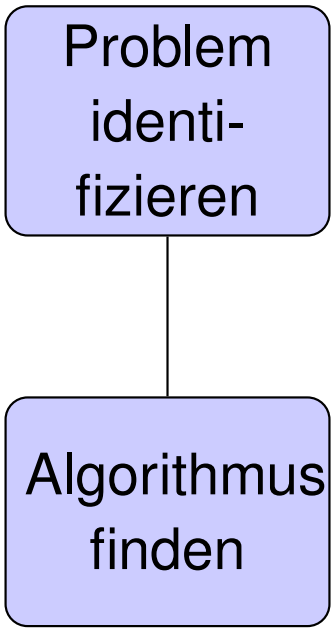
Programmieren

Zyklus

Problem identifizieren

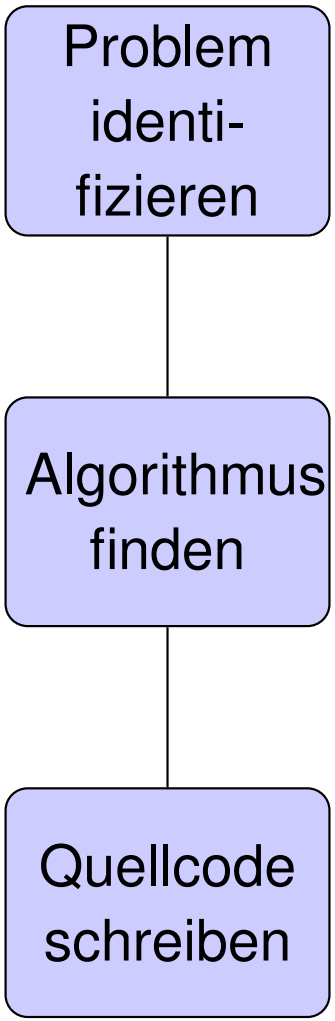
Programmieren

Zyklus



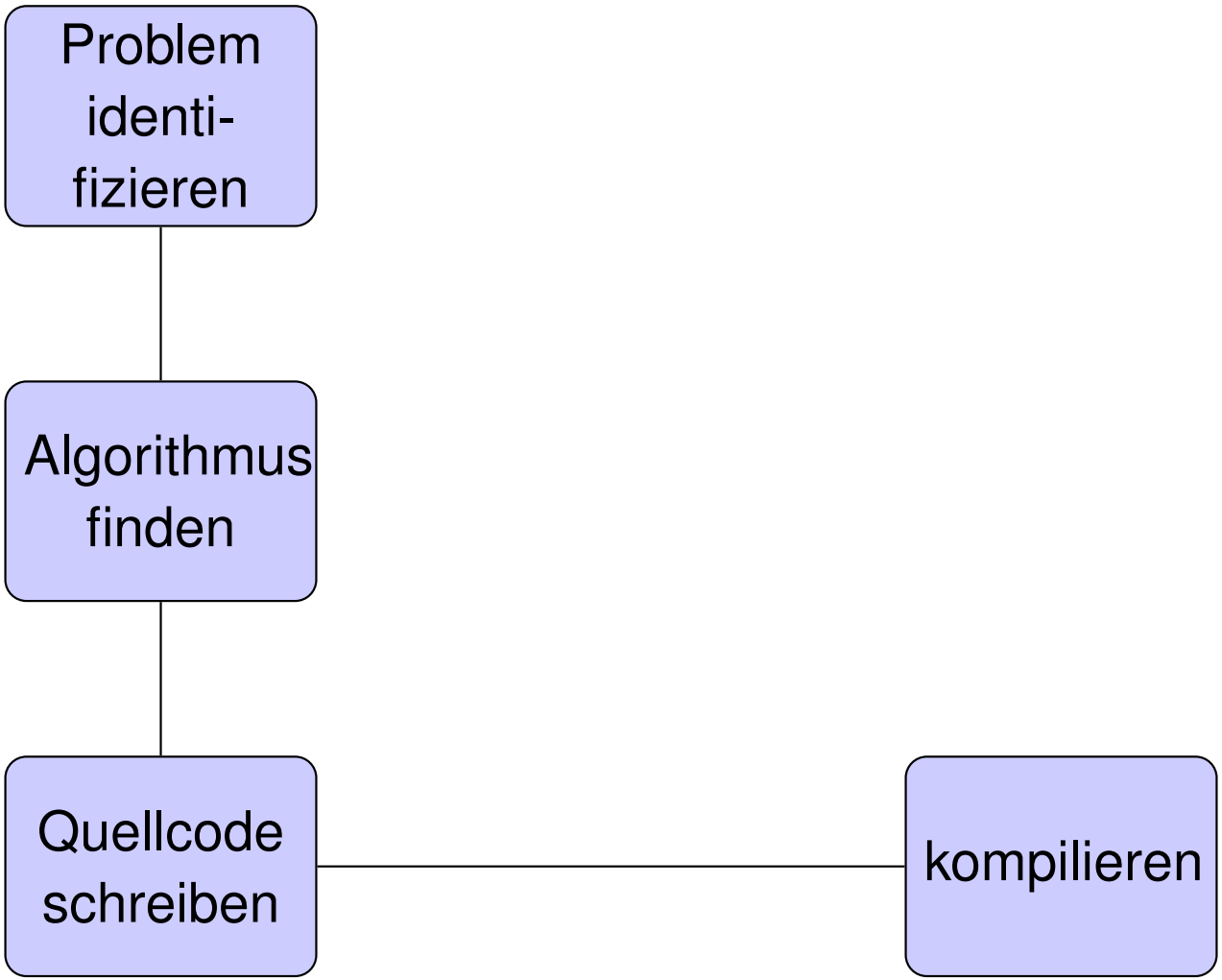
Programmieren

Zyklus



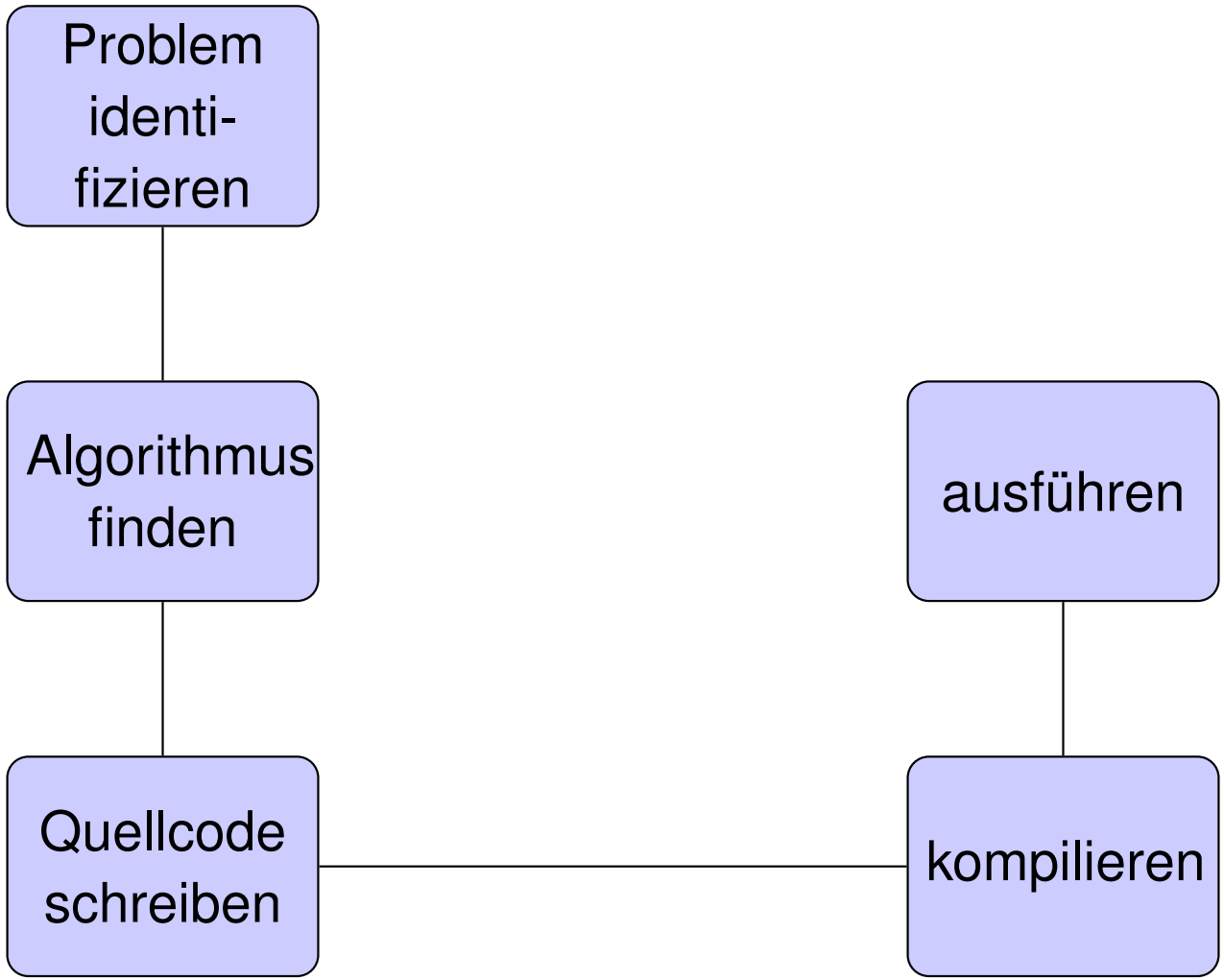
Programmieren

Zyklus



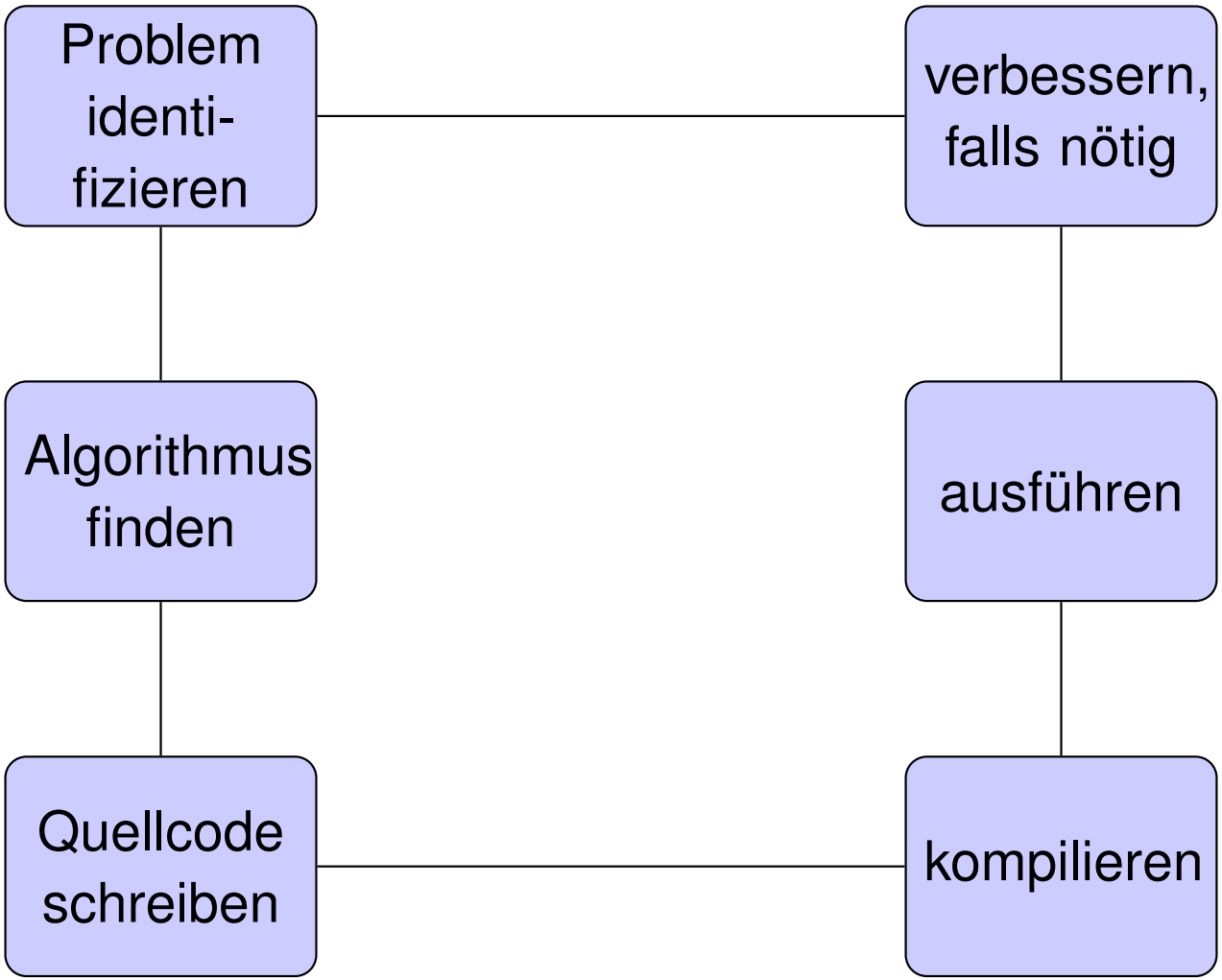
Programmieren

Zyklus



Programmieren

Zyklus



Tools

- Für das Programmieren braucht man Werkzeuge (Tools)
  - Entwicklungsumgebung oder Texteditor (Eclipse-IDE)
  - Compiler, Linker (java-SDK)
  - Erstellungswerkzeug/Build-Tool (make, ant)
  - Verionskontrollensystem/Repository (Subversion)

Tools

- Für das Programmieren braucht man Werkzeuge (Tools)
  - Entwicklungsumgebung oder Texteditor (Eclipse-IDE)
  - Compiler, Linker (java-SDK)
  - Erstellungswerkzeug/Build-Tool (make, ant)
  - Verionskontrollensystem/Repository (Subversion)
- Um ein Programm auszuführen, braucht man auch Werkzeuge
  - Betriebssystem
  - Laufzeitsystem

Tools

- Für das Programmieren braucht man Werkzeuge (Tools)
  - Entwicklungsumgebung oder Texteditor (Eclipse-IDE)
  - Compiler, Linker (java-SDK)
  - Erstellungswerkzeug/Build-Tool (make, ant)
  - Verionskontrollensystem/Repository (Subversion)
- Um ein Programm auszuführen, braucht man auch Werkzeuge
  - Betriebssystem
  - Laufzeitsystem

- Man sollte sich die Programmierumgebung so angenehm wie möglich einrichten

Programmstruktur

- Miniprogramm braucht:
  - Klasse
    `public class Programm1 { }`
    (mehr dazu später)
  - Startpunkt des Programms
    `public static void main(String[] args) { }`

Programmstruktur eines Miniprogramms

```
public class Programm1 {
    public static void main(String[] args) {
        // ... Anweisungen ...
        System.out.println("Hallo!");
    }
}
```

Daten

- Daten
  - Aneinanderkettung mehrerer binärer Werte
  - Bsp. 8 bits: 01001001 = hex 0x49 = dez 73 = Buchstabe I
  - Daten ohne Typ: Unklar, was Daten bedeuten (73 oder I?)
- Variablen speichern Daten
  - Brauchen Datentyp, damit man Variable richtig interpretiert
  - Muss Variable mit Datentyp deklarieren: `int i;`
  - → Programm2

Datentypen

Wertebereiche der Zahlen

Ganzzahlen: Wertebereich: -2^{Bits-1} bis $2^{Bits-1} - 1$		
Typ	Bits	Wertebereich
`byte`	8	-128 – 127
`short`	16	-32768 – 32767
`int`	32	-2147483648 – 2147483647
`long`	64	-9223372036854775808 – 9223372036854775807
`char`	16	00000 – 65535

Datentypen

Wertebereiche der Zahlen

Ganzzahlen: Wertebereich: -2^{Bits-1} bis $2^{Bits-1} - 1$		
Typ	Bits	Wertebereich
`byte`	8	-128 – 127
`short`	16	-32768 – 32767
`int`	32	-2147483648 – 2147483647
`long`	64	-9223372036854775808 – 9223372036854775807
`char`	16	00000 – 65535

Gleitkommazahlen:			
Typ	Bits	Kleinste Zahl > 0	Grösste Zahl
`float`	32	1.40e-45f	3.4028235e38f
`double`	64	4.9e-324	1.79769313486231570e308

Datentypen

Wertebereiche der Zahlen

Wahrheitswert:

Typ	Werte
`boolean`	`true` oder `false`

Werte vergleichen

- Manchmal muss ein Programm Werte vergleichen
- Resultat eines Vergleichs ist ein `boolean`
  - `true`, wenn der Vergleich stimmt
  - `false`, wenn der Vergleich **nicht** stimmt

Vergleichsoperatoren: `int z1 = 3; int z2 = 7;`

Operator	Bsp	Resultat	Bedeutung
`==`	`z1 == z2`	`false`	Ist `z1` gleich `z2` ?
`!=`	`z1 != z2`	`true`	Ist `z1` ungleich `z2` ?
`<`	`z1 < z2`	`true`	Ist `z1` kleiner als `z2` ?
`<=`	`z1 <= z2`	`true`	Ist `z1` kleiner oder gleich `z2` ?
`>`	`z1 > z2`	`false`	Ist `z1` grösser `z2` ?
`>=`	`z1 >= z2`	`false`	Ist `z1` grösser oder gleich `z2` ?

Werte vergleichen

Beispiel

- Problem
  - Speichere grössere der Zahlen `z1` und `z2` in `max`

Werte vergleichen

Beispiel

- Problem
  - Speichere grössere der Zahlen `z1` und `z2` in `max`

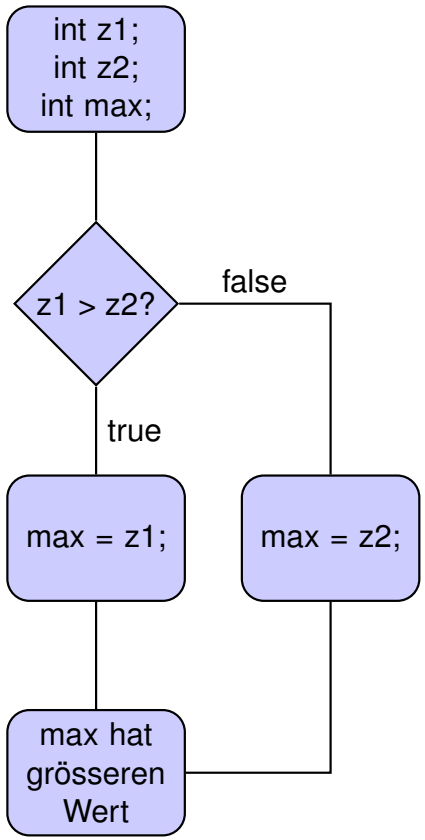
- Algorithmus (Kochbuchrezept)
  1. Zahlen mit $>$ vergleichen
  2. Falls **`true`**, linke Zahl in `max` speichern
  3. Falls **`false`**, rechte Zahl in `max` speichern

Werte vergleichen

Beispiel

- Problem
  - Speichere grössere der Zahlen `z1` und `z2` in `max`
- Algorithmus (Kochbuchrezept)
  1. Zahlen mit > vergleichen
  2. Falls **`true`**, linke Zahl in `max` speichern
  3. Falls **`false`**, rechte Zahl in `max` speichern

Algorithmus (Flussdiagramm)



Werte vergleichen

- Zahlen verglichen können → Konzept
- Spezielle Schlüsselwörter → Programmiersprache

Verzweigungen

- Zwei wichtige Arten der Verzweigung
  - `if(BEDINGUNG) { } else { }`
  - `switch(VARIABLE) case FALL 1: ... FALL N:`
- `BEDINGUNG` ist ein Test mit einem `boolean` als Resultat
- → Programm3

Bedingungen verknüpfen

- Manchmal mehr als einen Wert vergleichen
  - Etwas muss gelten UND etwas anderes auch
  - Eine ODER die andere ODER beide Bedingungen müssen gelten
  - Bedingung soll NICHT gelten

Verknüpfungen

Verknüpfung	Schreibweise	Beispiel
UND	&&	`((z1 < 5) && (z2 > 3))`
ODER	||	`((z1 > 0) || (z2 > 0))`
NICHT	!	`(!(z1 == z2))`

- Klammerung optional, Operatoren *binden* verschieden stark
  - Bindet stark `!`, `&&`, `||` bindet schwach

Selektionen

Variable auf verschiedene Werte testen

- Manchmal je nach Variablenwert andere Aktion ausführen
- Fallunterscheidung
  - Übersichtlich
  - Einfach zu schreiben
  - Sprachkonstrukt: `switch-case`
  - `default` optional
  - `break` optional
- Übersichtlicher als `if-else`

Schleifen

- Manchmal müssen Anweisungen mehrfach ausgeführt werden
- Schleife, um Anweisungen mehrfach auszuführen
- Drei Typen
  - `while`-Schleife (Abweisschleife)
  - `do-while`-Schleife (Durchlaufschleife)
  - `for`-Schleife (Zählschleife)
- → Programm5, Programm6, Programm7

Kurzschreibweisen

Kurzschreibweisen

Anweisung	Kurzschreibweise
`i = i + 1`	`i++;`
`i = i + 5`	`i+=5;`
`i = i - 1`	`i--;`
`i = i - 5`	`i-=5;`

`if (z1 > z2) {`
`max = z1;`
`} else {`
`max = z2;`
`}`
	`// (Bedingung ? falls ja : falls nein)`
	`max = (z1 > z2 ? z1 : z2);`