Einführung in C++ und die Objektorientierte Programierung

Florian Adamsky, B. Sc. (PhD cand.)

florian.adamsky@iem.thm.de
http://florian.adamsky.it/

Softwareentwicklung im WS 2014/15

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Outline

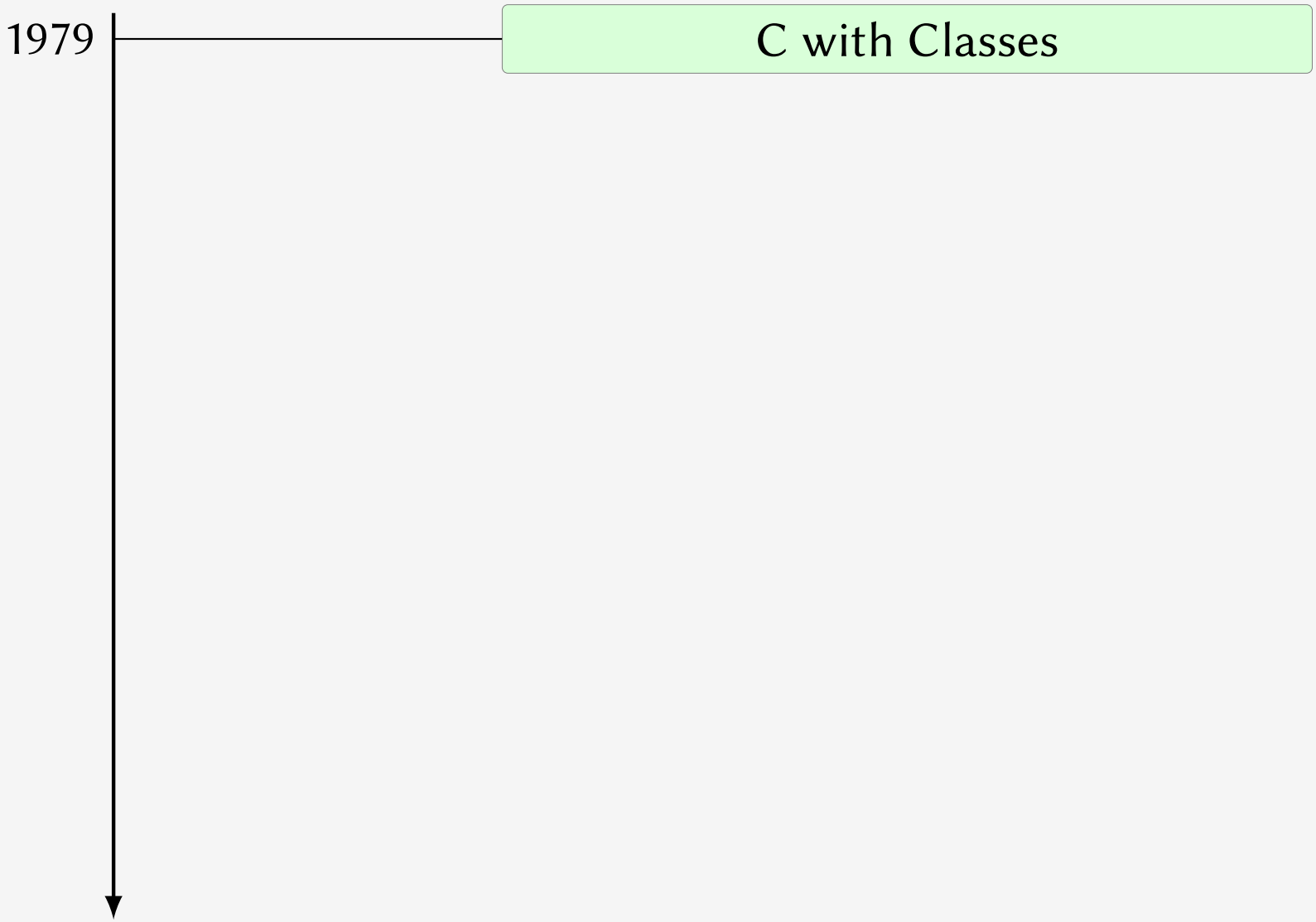
Inhaltsverzeichnis

Der Anfang

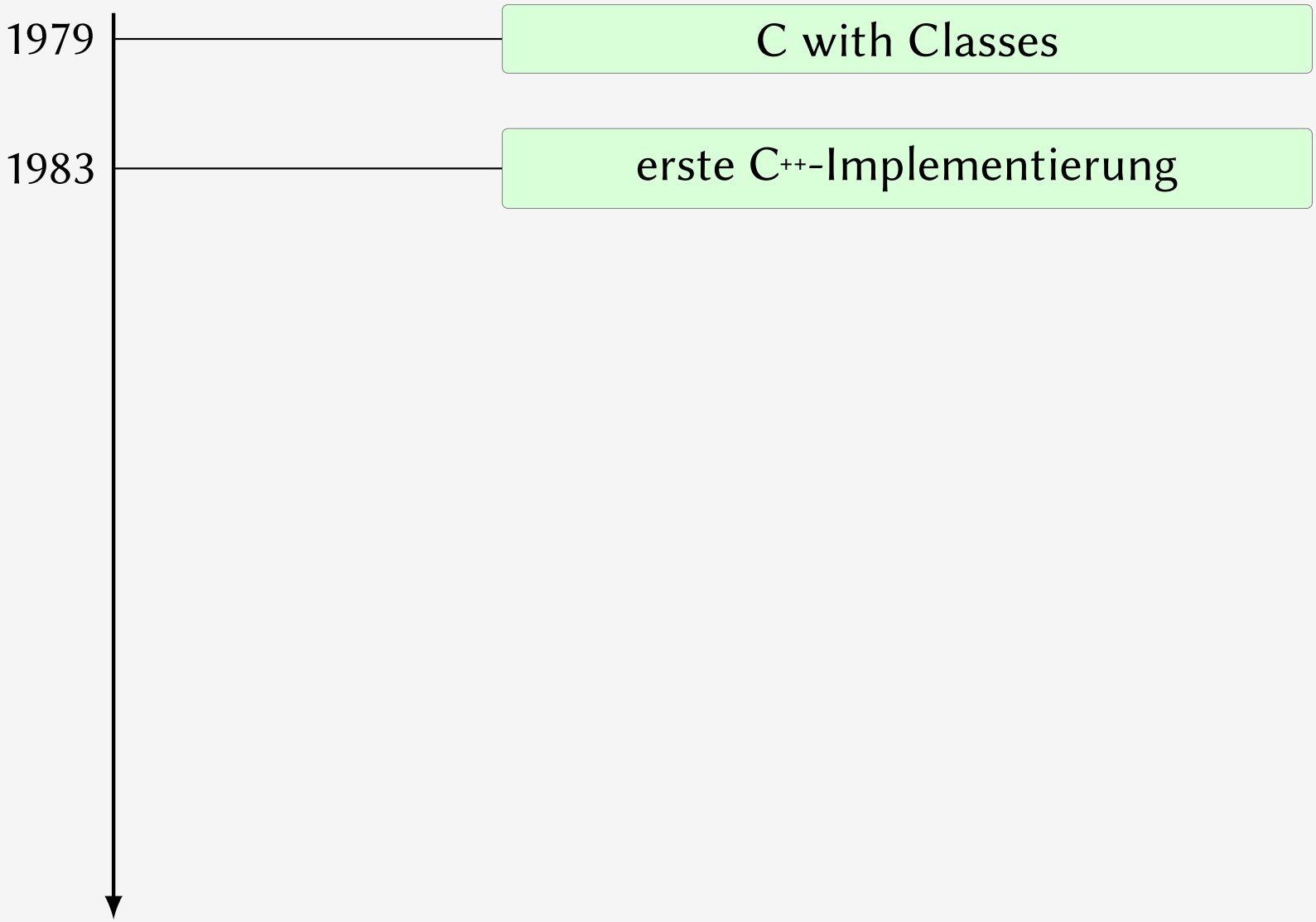
- Bjarne Stroustrup programmierte während seiner Dokotorarbeit in der Sprache Simula 67
- 04/1979 arbeitete Stroustrup an der Frage inwieweit man den Unix Kernel über das Netzwerk verteilen kann
- in welcher Programmiersprache schreibt man den Simulator?
  - Simula war objektorientiert, aber zu langsam
  - C war schnell, aber nicht objektorientiert
- $\Rightarrow$ 10/1979 began Stroustrup an *C with Classes*

Abbildung: Bjarne Stroustrup

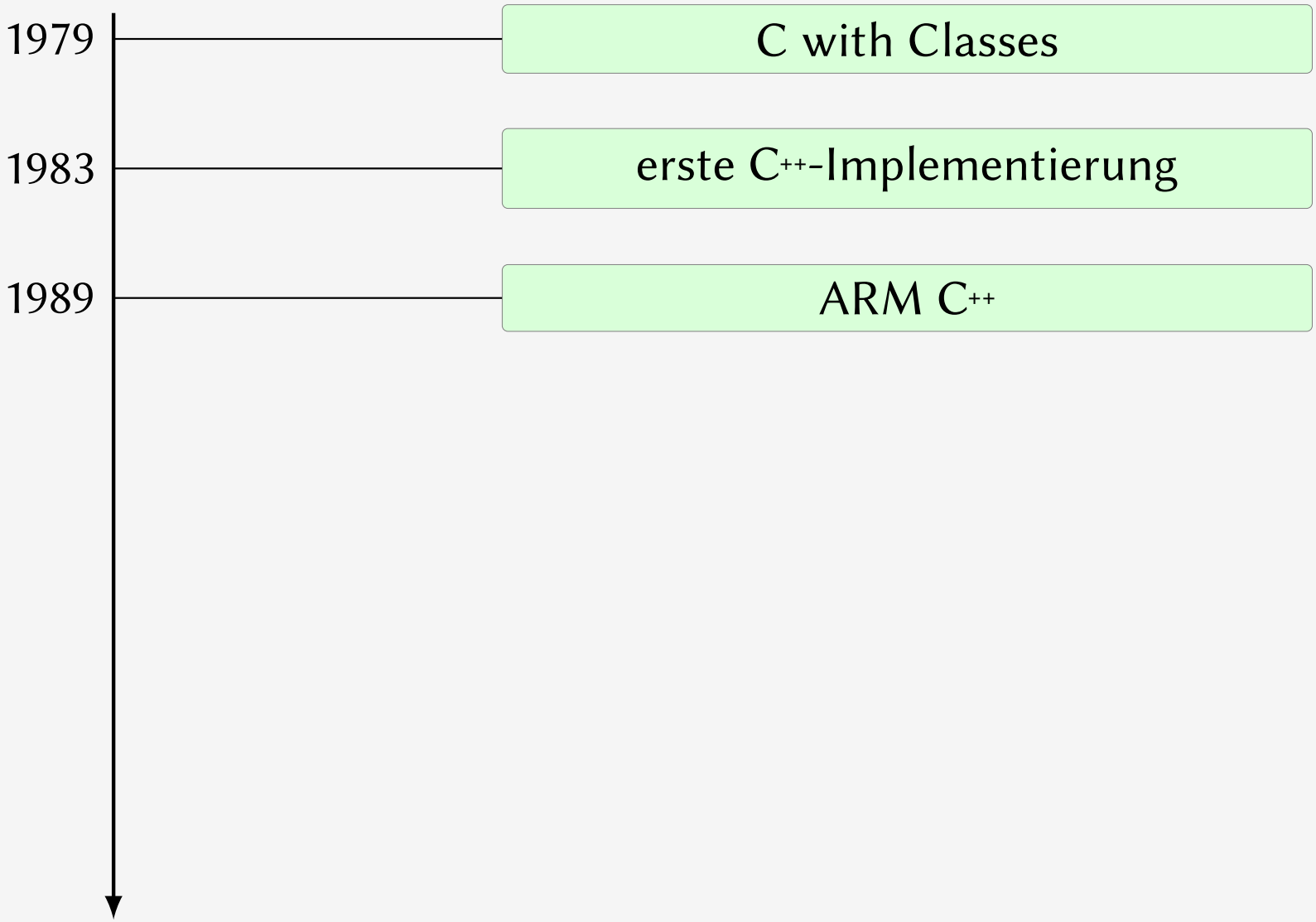
Zeitstrahl



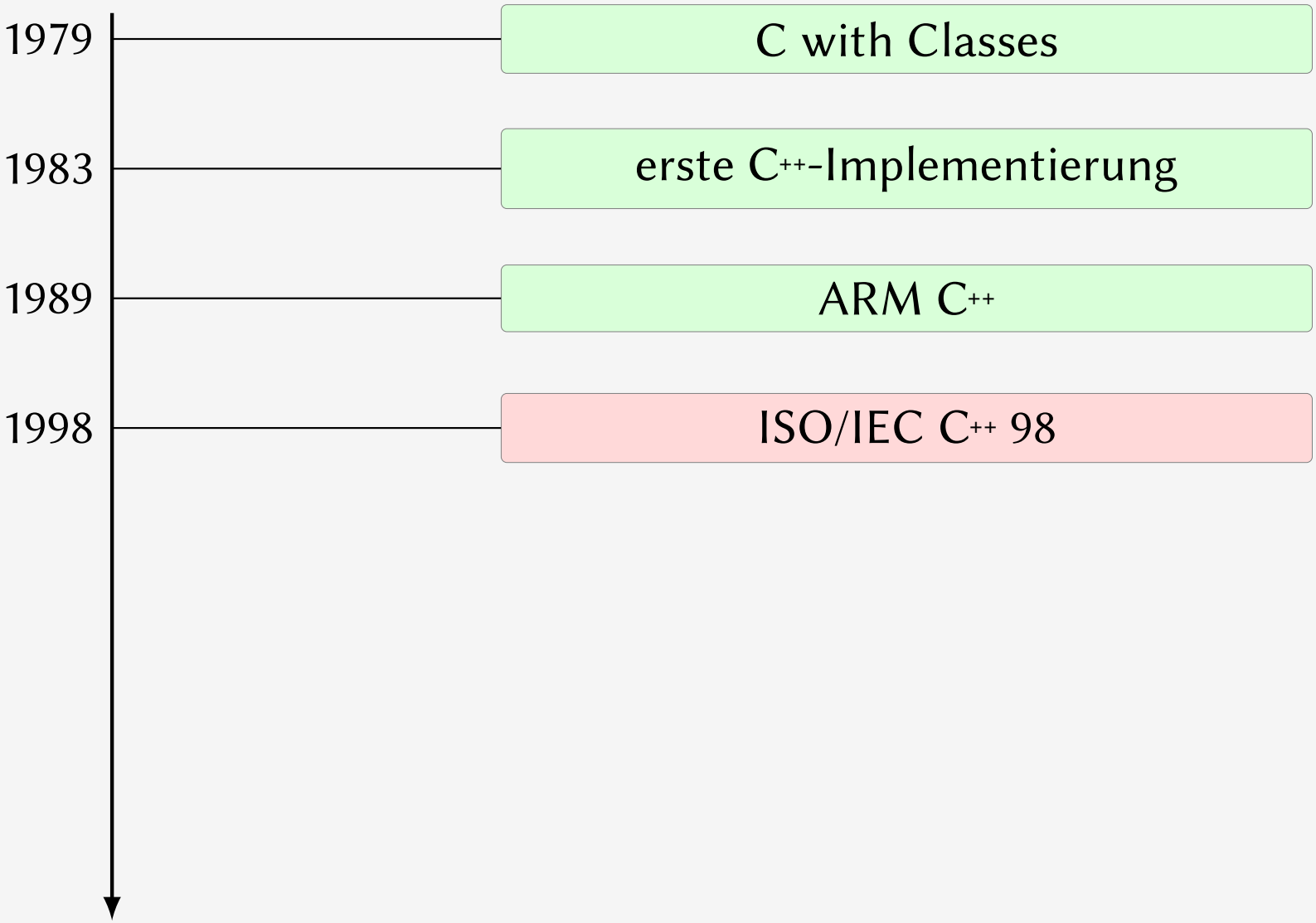
Zeitstrahl



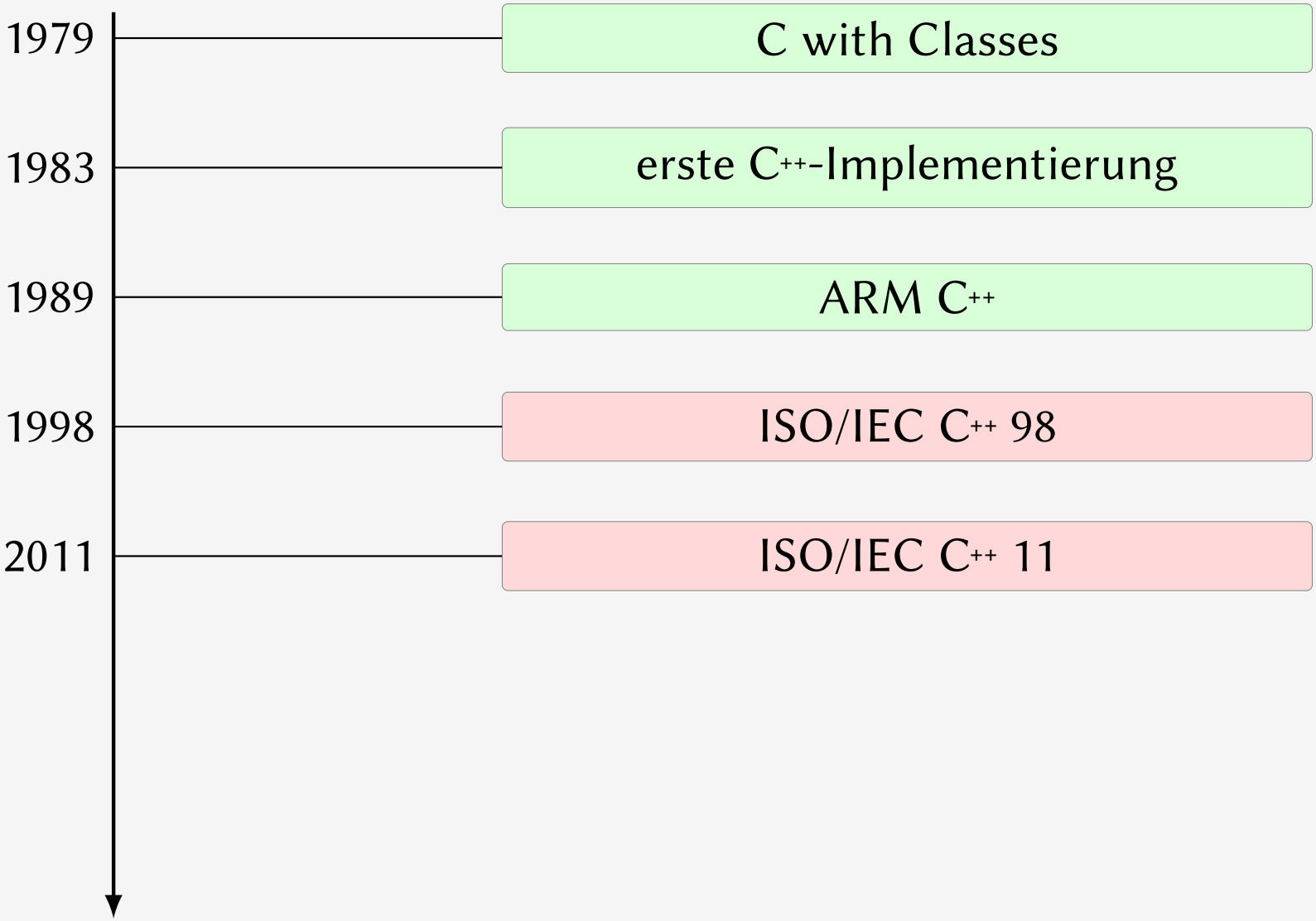
Zeitstrahl



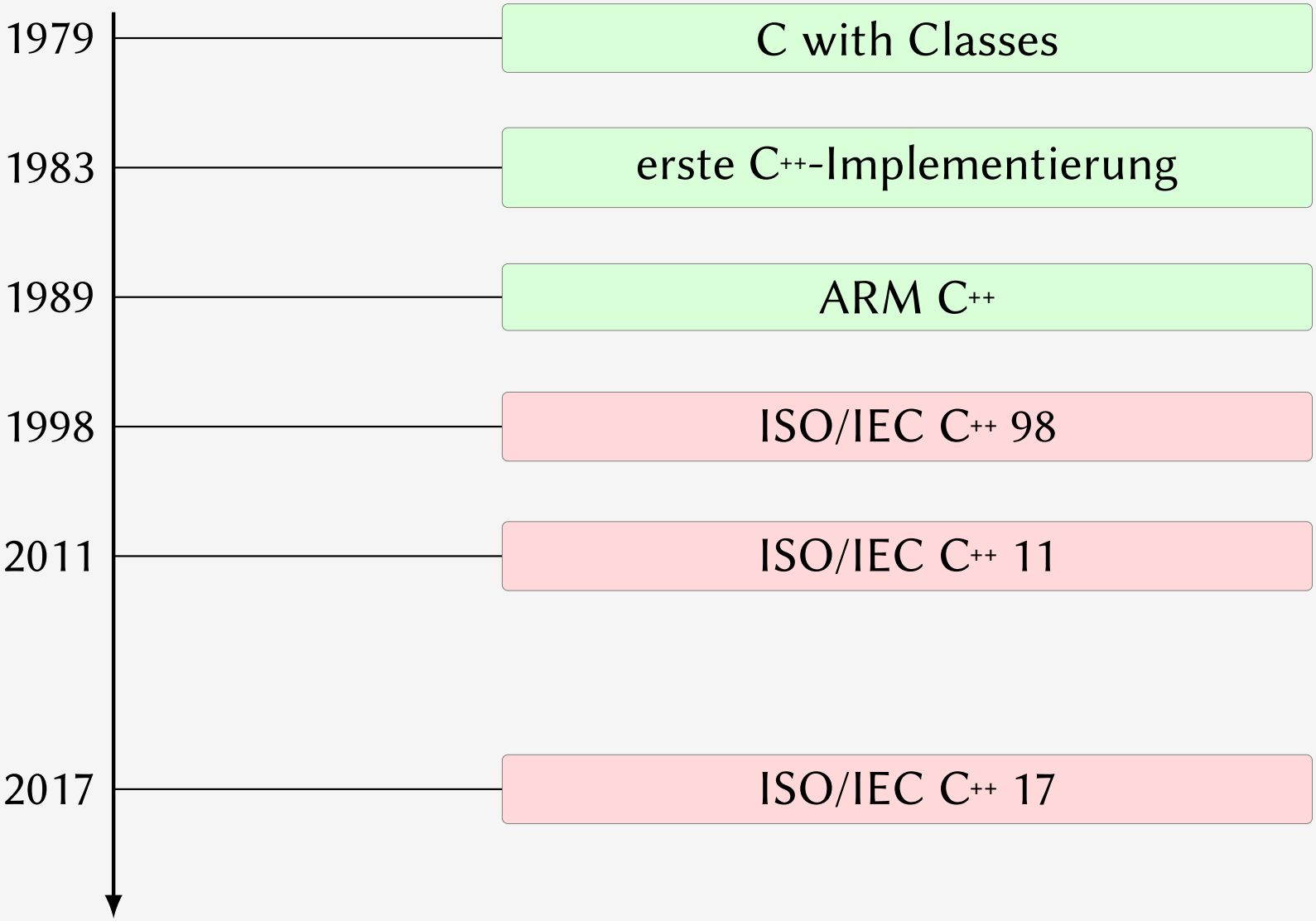
Zeitstrahl


1979
C with Classes
1983
erste C++-Implementierung
1989
ARM C++
1998
ISO/IEC C++ 98

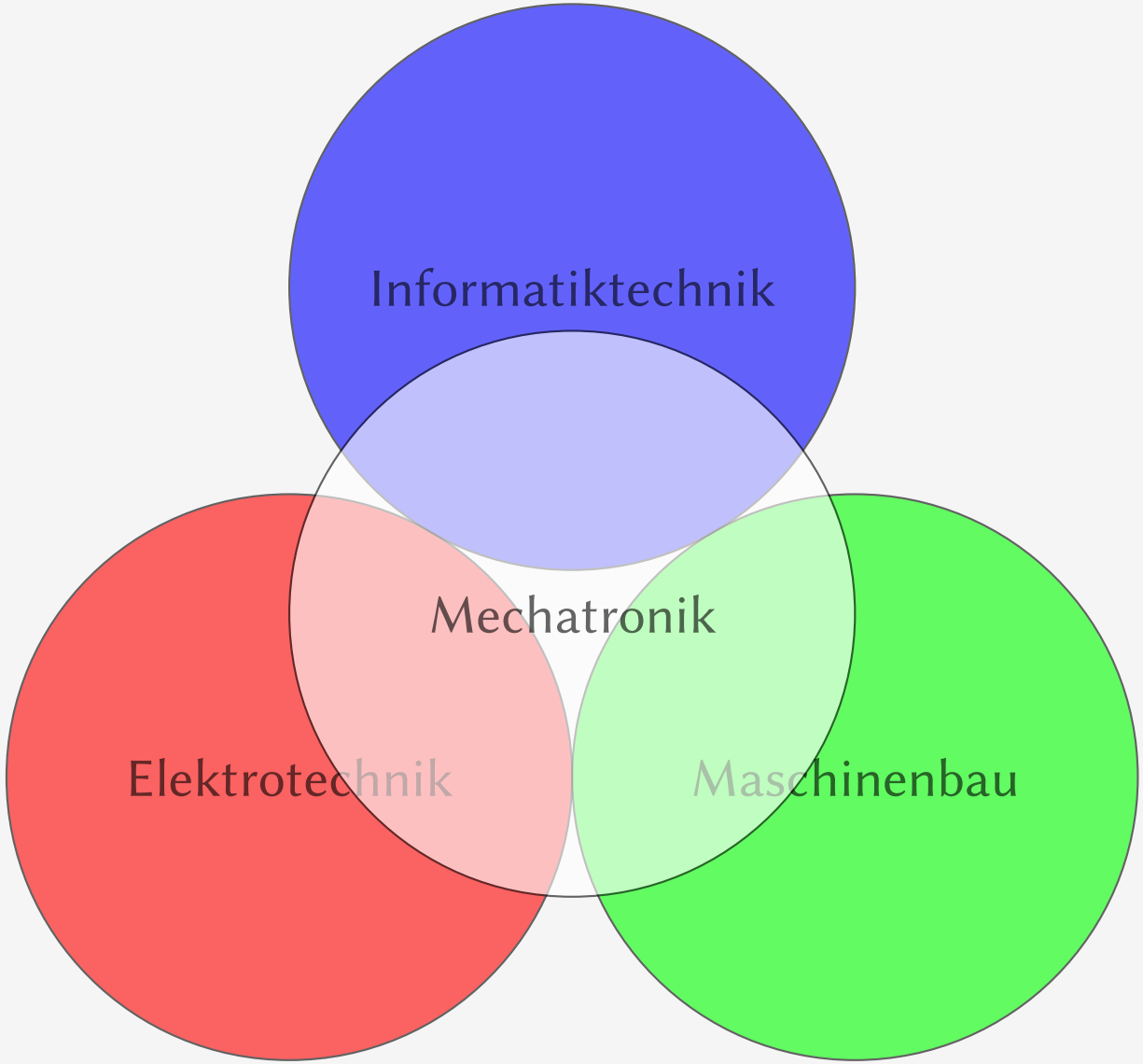
Zeitstrahl


1979
C with Classes
1983
erste C++-Implementierung
1989
ARM C++
1998
ISO/IEC C++ 98
2011
ISO/IEC C++ 11

Zeitstrahl



Mechatronik



Warum C++?

- unterstützt mehrere Programmierparadigmen (Multiparadigma)
  - imperativ
  - objektorientiert
  - teilweise funktional (mit C++ 14 gibt es Lambda Expressions)
- Maschinennah, performant und trotzdem eine hohe Abstraktion
- mächtige Bibliotheken erleichtern das Programmieren (STL, Boost)

Warum C++?

- unterstützt mehrere Programmierparadigmen (Multiparadigma)
  - imperativ
  - objektorientiert
  - teilweise funktional (mit C++ 14 gibt es Lambda Expressions)
- Maschinennah, performant und trotzdem eine hohe Abstraktion
- mächtige Bibliotheken erleichtern das Programmieren (STL, Boost)

> *"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off."*
>
> *Bjarne Stroustrup in 1986*

Inhaltsverzeichnis

Grundgerüst

- C++ achtet auf Groß- und Kleinschreibung
- Alle Anweisungen (eng. statements) verlangen ein Semikolon am Ende

Example (Hello World)

```
#include <iostream>

using namespace std;

int main() {
    // Ausgabe
   cout << "Hello World!";
}
```

Präprozessor

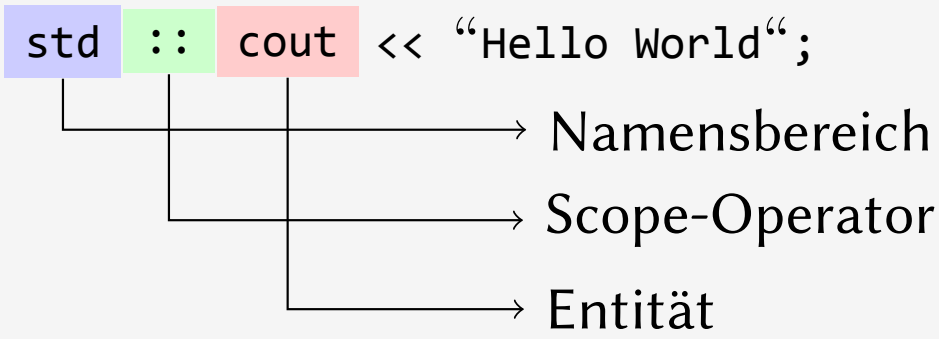
- `#include` ist kein Sprachelement von C++, sondern ein Befehl des Präprozessors
- Präprozessor ist Teil des Compilers
- Alle Befehle fangen mit eimem # an
  - müssen immer in der ersten Spalte
- Spitzenklammern `< ... >` weisen den Präprozessor an im Standardverzeichnis nach Bibliotheken zu suchen
- C++ Standard-Bibliotheken sind ohne `*.h` am Ende

Example (Hello World)

```
#include <iostream>

using namespace std;

int main() {
    // Ausgabe
   cout << "Hello World!";
}
```

Namespace/Namensbereiche

- Vermeidung von Doppelbenennungen
- Ohne `using namespace` müsste die Ausgabe wie folgt aussehen:



- Analogie: Vorwahlen bei Telefonummern

Example (Hello World)

```
#include <iostream>

using namespace std;

int main() {
    // Ausgabe
    cout << "Hello World!";
}
```

Funktionen

- Hauptfunktion heißt `main()`
- Jedes Programm muss eine `main()`-Funktion haben
- Laut Ansi-Norm hat die `main()`-Funktion eine implizierte `return 0` Anweisung

Example (Hello World)

```
#include <iostream>

using namespace std;

int main() {
    // Ausgabe
    cout << "Hello World!";
}
```

Kommentare

- Kommentare sind wie in C
- Mehrzeilige Kommentare mit

```
/*
  Eine Zeile
  Noch eine Zeile
*/
```

- Einzeilige Kommentaren mit

```
// Kein Kommentar
```

Example (Hello World)

```
#include <iostream>

using namespace std;

int main() {
    // Ausgabe
    cout << "Hello World!";
}
```

Ausgabe

- `cout` steht für *Consolen Output* und heißt soviel wie Konsolen Ausgabe
- Ausgabeoperator << schiebt die Daten in den Ausgabestrom
- dient auch zur Verbindung von mehreren Daten

```
int x = 42;
cout << "Hello " <<  x << endl;
```

Example (Hello World)

```
#include <iostream>

using namespace std;

int main() {
    // Ausgabe
    cout << "Hello World!";
}
```

Operatoren

Abkürzung	Beispiel	Bedeutung
`++`	`i++`	`i = i + 1`
`--`	`i--`	`i = i - 1`
`+=`	`x += 5`	`x = x + 5`
`-=`	`x -= 5`	`x = x - 5`
`*=`	`x *= 5`	`x = x * 5`
`/=`	`x /= 5`	`x = x / 5`

Ganzzahlen

Datentyp	Größe	Wertebereich
`int`	2 Bytes	-32768 bis +32767
`unsigned int`	2 Bytes	0 bis +65535
`long`	4 Bytes	-2147483648 bis +2147483647
`unsigned long`	4 Bytes	0 bis +4294967295
`short`	1 Byte	-128 bis +127
`unsigned short`	1 Byte	0 bis 255

Boolesche Werte

- Boolesche Variablen können nur 1 oder 0 annehmen

```
bool wahr   = 1;
bool falsch = 0;
```

- C++ definiert auch die folgenden Schlüsselwörter zur besseren Lesbarkeit

```
bool wahr   = true;
bool falsch = false;
```

Fließkomma

Datentyp	Größe	Nachkommastellen
`float`	4 Bytes	6
`double`	8 Bytes	10
`long double`	10 Bytes	10

Strings

- C++ STL unterstützt sowohl `cstring` als auch `string`
- die Bibliothek `string` wird empfohlen

Example (Strings)

```
#include <iostream>
#include <string>

int main()  {
    std::string s1 = "Hallo ";
    std::string s2 = "Welt";
    std::string ergebnis = s1 + s2;

    if (s1 == s2) {
        std::cout << "string ist gleich" << std::endl;
    }
}
```

Definition/Initialisierung

Definition (Definition)

Die *Definition* weist den Compiler an eine Variable zu erstellen.

```
int x;
```

Definition (Initialisierung)

Bei der *Initialisierung* initialisiert der Compiler die Variable mit einem Startwert.

```
x = 23;
```

Definition und Initialisierung

```
int x = 23;
```

Beispiel für uninitialisierte Variablen

Example (Was passiert?)

```
#include <iostream>

void funktion() {
    int x;
    std::cout << "x has the following value: " << x << std::endl;
}

int main() {
    int x;
    std::cout << "x has the following value: " << x << std::endl;

    funktion();
}
```

Beispiel für uninitialisierte Variablen

Example (Was passiert?)

```
#include <iostream>

void fu
    int
    st
}

int main() {
    int x;
    std::cout << "x has the following value: " << x << std::endl;

    funktion();
}
```

Ausgabe

```
$ ./programm
x has the following value: 0
x has the following value: 32592
```

Uninitialisierte Variablen

- statische Variablen werden mit 0 automatisch initialisiert

```
int x;                  // Wert ist 0

void funktion() { static int x; }  // Wert ist 0
```

- Nicht-statische Variablen (lexikalische Variablen) sind nicht initialisiert

```
void funktion () {
    int x;
    cout << x << endl;
}
```

- der Compiler ist hier frei abzustürzen oder irgendwelche Werte zu definieren

Uninitialisierte Variablen

- statische Variablen werden mit 0 automatisch initialisiert

```
int x;                    // Wert ist 0

voi
```

- Nic alisiert

```
voi

}
```

Tipp

Initialisieren Sie immer alle Variablen. Dadurch vermeiden Sie Fehler die sehr schwer zu finden sind.

- der Compiler ist hier frei abzustürzen oder irgendwelche Werte zu definieren

Inhaltsverzeichnis

Warum Objektorientiertes Programmieren?

- Probleme in der realen Welt sind schwer prozedural zu lösen
  - Trennung zwischen Funktionen und Daten
- Dadurch schlechte Wiederverwendbarkeit
- Objektorientierte Paradigma ähnelt der menschlichen Abstraktion

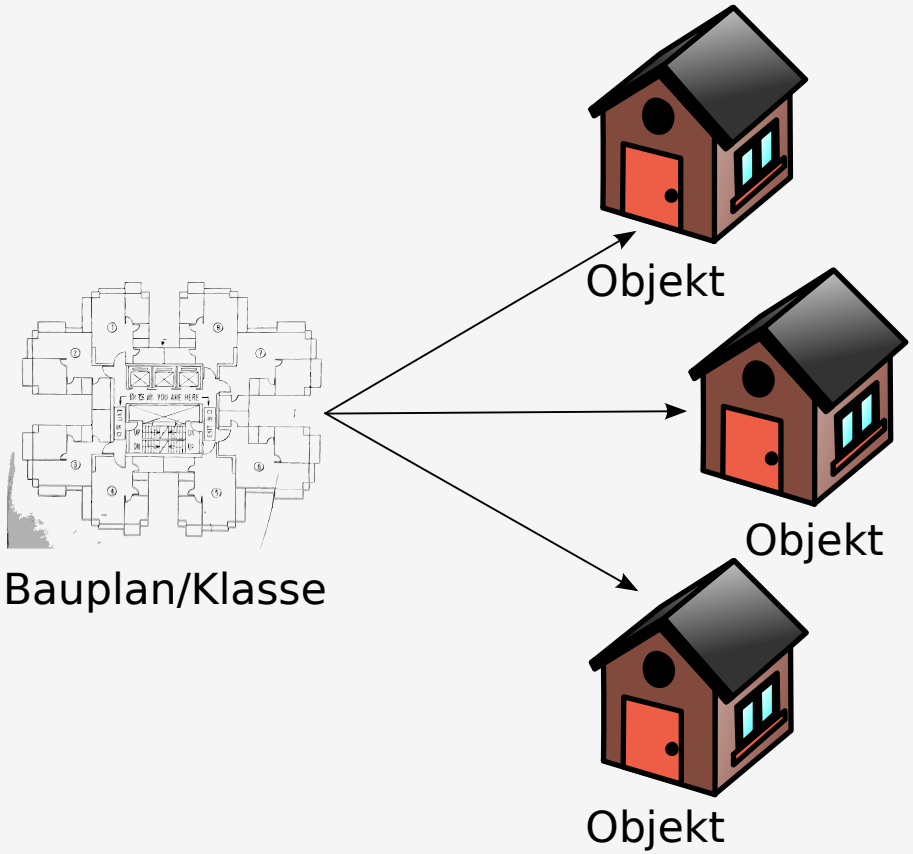
Geschichte

- 1965: Simula-1 Programmiersprache
  - erste objektorientierte Sprache
- 1967: Simula-67 Programmiersprache
  - Klassen
  - Objekte
  - Vererbung
  - virtuelle Methoden
  - Garbage Collection

Was ist eine Klasse?

Definition (Klasse)

Eine Klasse ist ein Bauplan nachdem der Compiler Objekte erzeugt



Was ist ein Objekt?

Definition

Ein Objekt ist eine konkrete Instanz einer Klasse im Speicher.

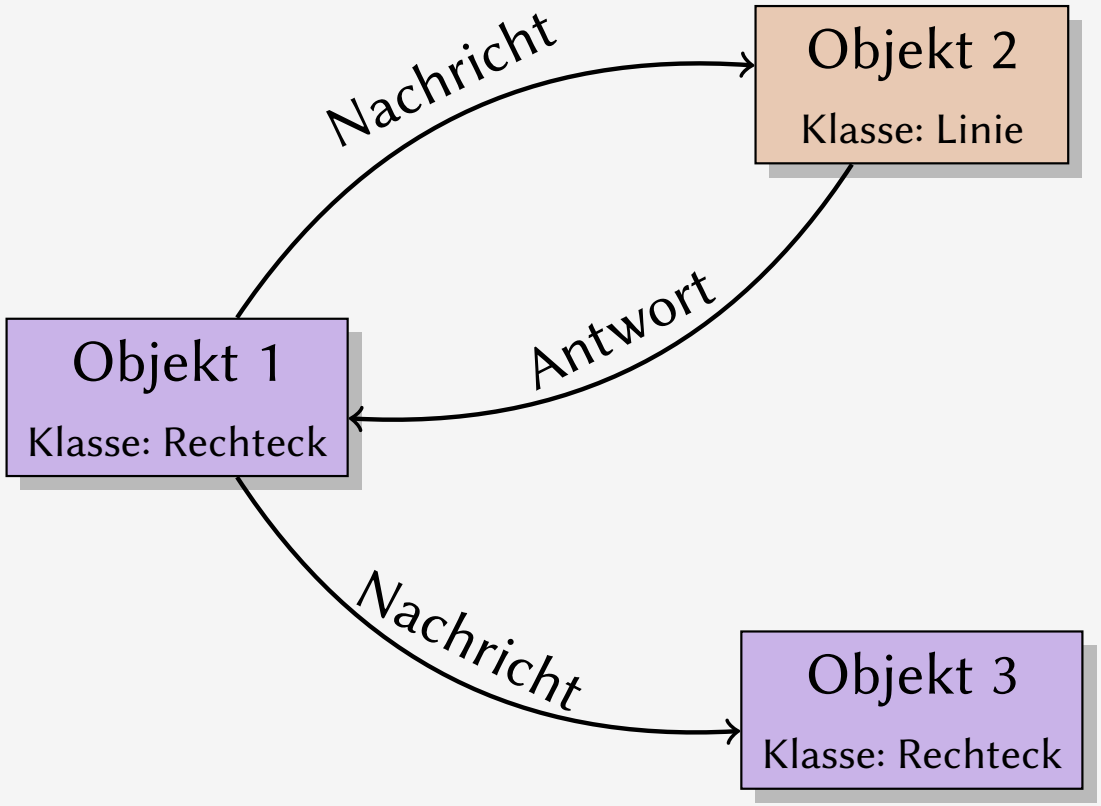
- vereint Daten und Funktionen, die auf diesen Daten operieren
- sind verwandt mit `struct` aus C

Example (Struct aus C)

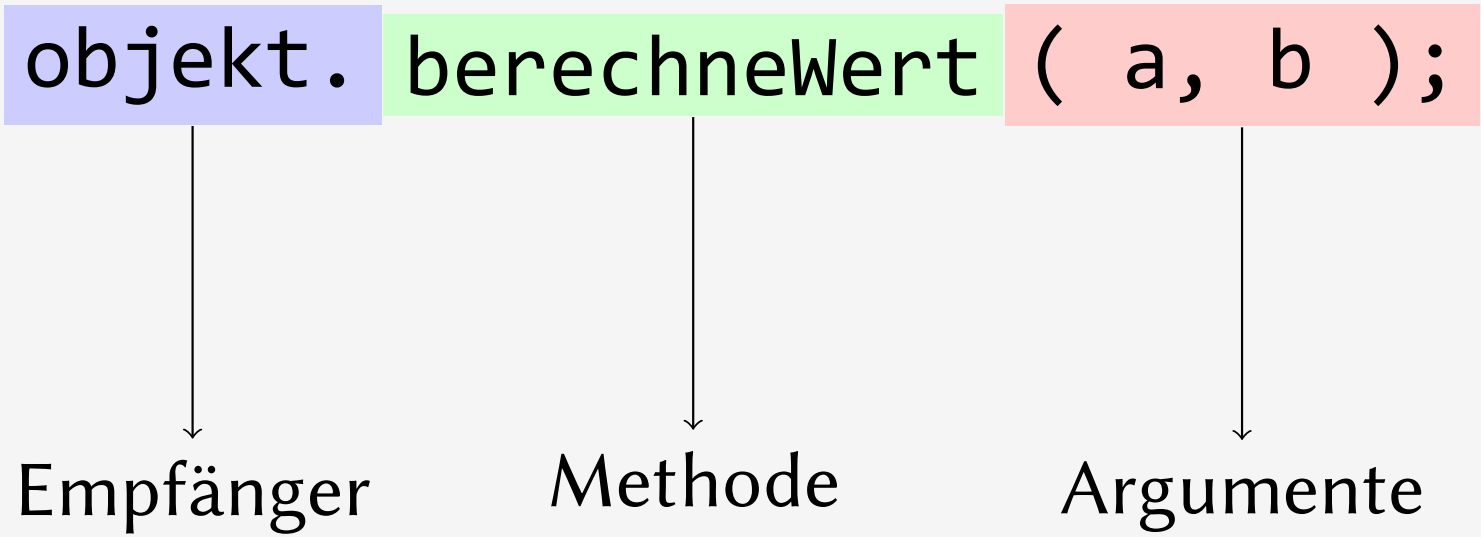
```c
struct punkt {
    int x;
    int y;
};

struct punkt p1 = { .x = 23, .y = 42 };
p1.x = 42; // Punktoperator
```

Nachrichten

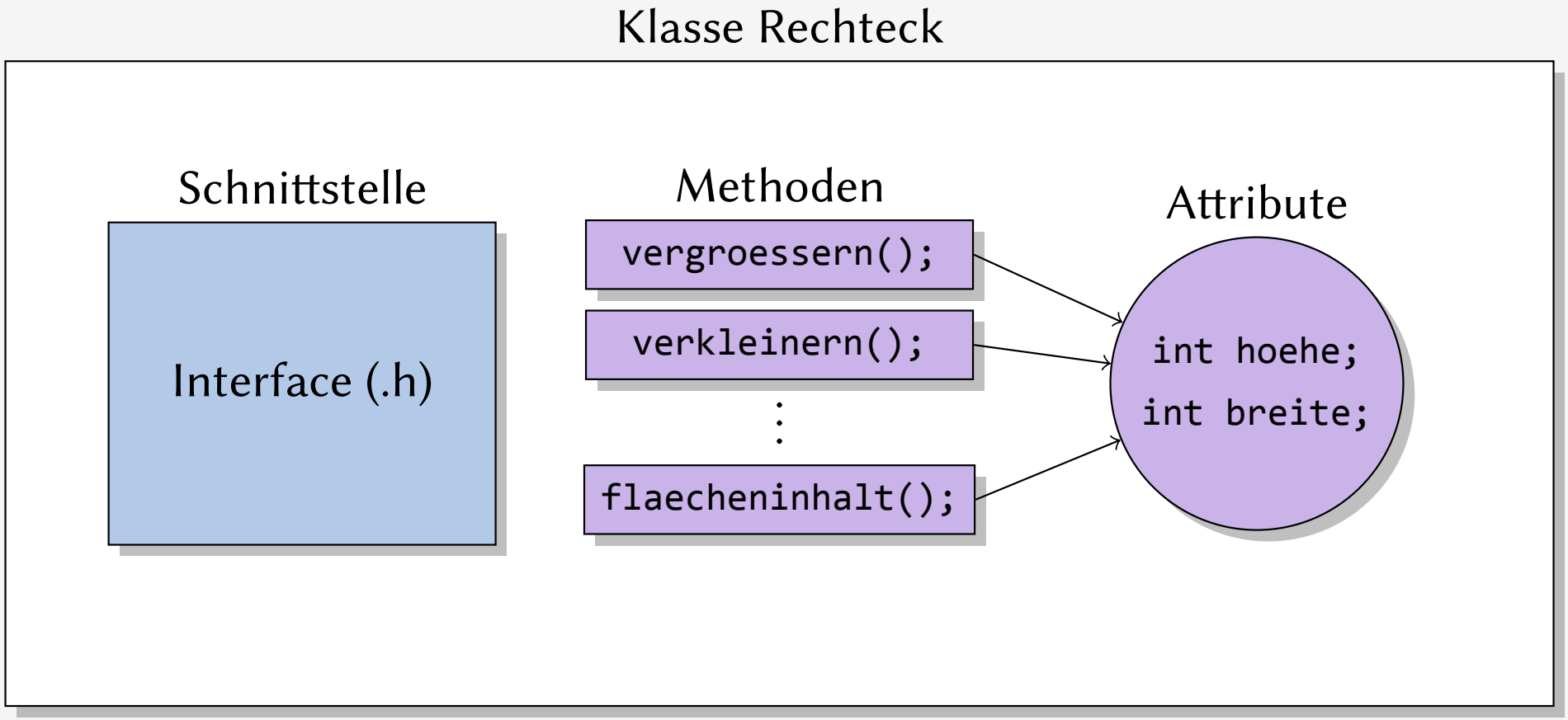
- Objekte können sich untereinander Nachrichten schicken



Nachrichten 2



Klassenstruktur (Datenkapselung)



Attribute und Methoden

Attribute Eigenschaften die ein Objekt besitzen

Methoden Ein Verhalten oder eine Tätigkeit die ein Objekt ausführen kann

Example (Attribute)

- Name
- Farbe

Example (Methoden)

- berechnen
- essen

Beispiele

Auto

Attribute

- `Motor`
- `Gangschaltung`

Methoden

- `fahren()`
- `bremsen()`

Beispiele

Auto

Attribute

- `Motor`
- `Gangschaltung`

Methoden

- `fahren()`
- `bremsen()`

Hund

Attribute

- `Halsband`
- `Leine`

Methoden

- `essen()`
- `bellen()`

Beispiele

Auto

Attribute

- `Motor`
- `Gangschaltung`

Methoden

- `fahren()`
- `bremsen()`

Hund

Attribute

- `Halsband`
- `Leine`

Methoden

- `essen()`
- `bellen()`

Student

Attribute

- `Auto`
- `Hund`

Methoden

- `UniFahren()`
- `sparzieren()`

Example (Klasse Rechteck.h)

```
class Rechteck {

public:
    int hoehe;
    int breite;

    void vergroessern(int d) {
        breite += d;
        hoehe  += d;
    }
    void verkleinern(int d) {
        breite -= d;
        hoehe  -= d;
    }

    int flaecheninhalt() {
        return breite * hoehe;
    }
};
```

Example (Klasse Rechteck.h)

`class Rechteck {` ← `class` Schlüsselwort definiert in C++ eine Klasse

```
public:
    int hoehe;
    int breite;

    void vergroessern(int d) {
        breite += d;
        hoehe  += d;
    }
    void verkleinern(int d) {
        breite -= d;
        hoehe  -= d;
    }

    int flaecheninhalt() {
        return breite * hoehe;
    }
};
```

Example (Klasse Rechteck.h)

```
class Rechteck {

public:
    int hoehe;
    int breite;

    void vergroessern(int d) {
        breite += d;
        hoehe  += d;
    }
    void verkleinern(int d) {
        breite -= d;
        hoehe  -= d;
    }

    int flaecheninhalt() {
        return breite * hoehe;
    }
};
```

`class` Schlüsselwort definiert in C++ eine Klasse

Obacht: Semikolon nicht vergessen!

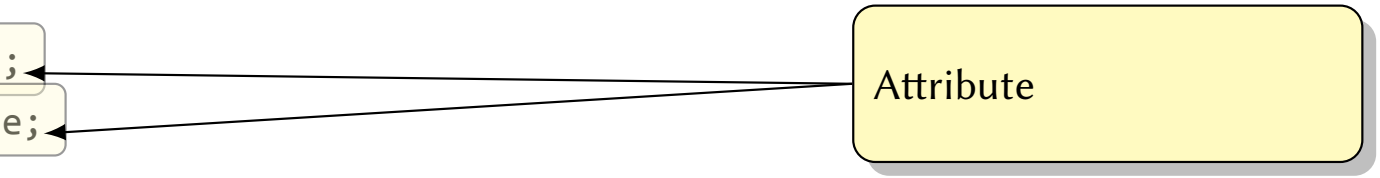
Example (Klasse Rechteck.h)

```
class Rechteck {

public:
    int hoehe;
    int breite;

    void vergroessern(int d) {
        breite += d;
        hoehe  += d;
    }
    void verkleinern(int d) {
        breite -= d;
        hoehe  -= d;
    }

    int flaecheninhalt() {
        return breite * hoehe;
    }
};
```

Zugriffsschutz

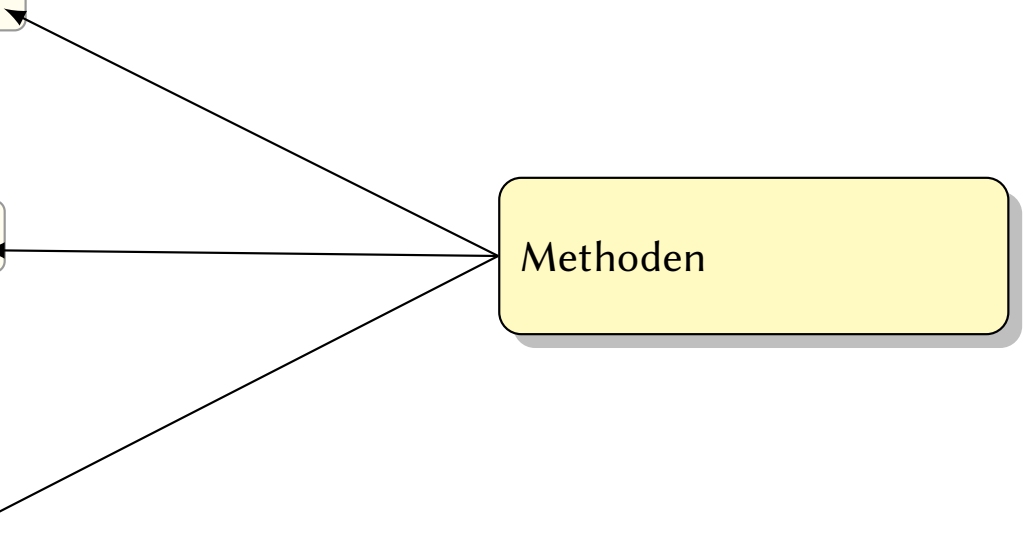
Example (Klasse Rechteck.h)

```
class Rechteck {

public:
    int hoehe;
    int breite;

    void vergroessern(int d) {
        breite += d;
        hoehe  += d;
    }
    void verkleinern(int d) {
        breite -= d;
        hoehe  -= d;
    }

    int flaecheninhalt() {
        return breite * hoehe;
    }
};
```



Example (Klasse Rechteck.h)

```
class Rechteck {

public:
    int hoehe;
    int breite;

    void vergroessern(int d) {
        breite += d;
        hoehe  += d;
    }
    void verkleinern(int d) {
        breite -= d;
        hoehe  -= d;
    }

    int flaecheninhalt() {
        return breite * hoehe;
    }
};
```



Example (Objekt)

```
#include <iostream>
#include "Rechteck.h"

using namespace std;

int main(void) {
    Rechteck quadrat;

    quadrat.hoehe  = 23;
    quadrat.breite = 23;

    cout << quadrat.hoehe  << endl;
    cout << quadrat.breite << endl;

    quadrat.vergroessern(10);

    cout << quadrat.flaecheninhalt() << endl;
}
```

Example (Objekt)

```
#include <iostream>
#include "Rechteck.h"

using namespace std;

int main(void) {
    Rechteck quadrat;

    quadrat.hoehe  = 23;
    quadrat.breite = 23;

    cout << quadrat.hoehe  << endl;
    cout << quadrat.breite << endl;

    quadrat.vergroessern(10);

    cout << quadrat.flaecheninhalt() << endl;
}
```

Einbindung der Klasse `Rechteck.h`

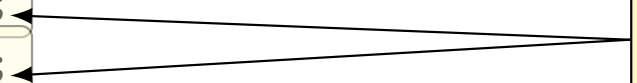
Example (Objekt)

```
#include <iostream>
#include "Rechteck.h"

using namespace std;

int main(void) {
    Rechteck quadrat;

    quadrat.hoehe  = 23;
    quadrat.breite = 23;

    cout << quadrat.hoehe  << endl;
    cout << quadrat.breite << endl;

    quadrat.vergroessern(10);

    cout << quadrat.flaecheninhalt() << endl;
}
```

Erstellung eines Objekts `quadrat` nach der Bauvorlage `Rechteck`

Example (Objekt)

```
#include <iostream>
#include "Rechteck.h"

using namespace std;

int main(void) {
    Rechteck quadrat;

    quadrat.hoehe  = 23;
    quadrat.breite = 23;

    cout << quadrat.hoehe  << endl;
    cout << quadrat.breite << endl;

    quadrat.vergroessern(10);

    cout << quadrat.flaecheninhalt() << endl;
}
```



Zugriff auf die Attribute des Objekts von außen

Example (Objekt)

```
#include <iostream>
#include "Rechteck.h"

using namespace std;

int main(void) {
    Rechteck quadrat;

    quadrat.hoehe  = 23;
    quadrat.breite = 23;

    cout << quadrat.hoehe  << endl;
    cout << quadrat.breite << endl;

    quadrat.vergroessern(10);

    cout << quadrat.flaecheninhalt() << endl;
}
```

Methodenaufruf von außen

Zugriffsschlüsselwörter (access specifier)

public	jede Klasse darf darauf zugreifen
private	nur die eigene Klasse darf darauf zugreifen

Zugriffsschutz

Example (Klasse)

```
class Rechteck {

private:
    int hoehe;

public:
    int breite;

    // [...]
};
```

Example (Main)

```
int main(void) {
    Rechteck quadrat;

    quadrat.breite = 23;
    quadrat.hoehe  = 23;
}
```

Zugriffsschutz

Example (Klasse)

```
class Rechteck {

private:
    int hoehe;

public:
    int breite;

    // [...]
};
```

Example (Main)

```
int main(void) {
    Rechteck quadrat;

    quadrat.breite = 23;
    quadrat.hoehe  = 23;
}
```

Fehler: int Rechteck::hoehe is private

Warum Zugriffsschutz?

Definition (Datenkapselung)

(auch Geheimnisprinzip) ist ein zentrales Konzept der objektorientierten Programmierung. Nach außen soll nur bekannt sein, was eine Klasse kann, aber nicht wie. Eine Klasse wird zur Black-Box

Zusammenfassung

- C++ ist maschinennah, schnell und unterstützt das objektorientierte Paradigma
- eine Klasse ist ein Bauplan und kann in C++ über das Schlüsselwort `class` erzeugt werden
- eine Klasse besitzt Attribute und Methoden
- von einer Klassen können viele Objekte erzeugt werden
- der Zugriff auf die Attribute und Methoden eines Objekt erfolgt über den Punktoperator `.`
- Objekte können über Methoden Nachrichten austauschen
- Datenkapselung ist kann über die Zugriffsschlüsselwörter `private` und `public` in C++