

Objektorientiert mit C –

Typische UML-Diagramme und ihre Umsetzung

Frank Listing  
f.listing@microconsult.com

Wie sieht die heutige embedded Entwicklung aus?

Welche UML-Elemente können in Code umgesetzt werden?

Wie können objektorientierte Elemente in C umgesetzt werden?

Ist der Schutz von Daten einer Struktur gegen direkten Zugriff möglich?

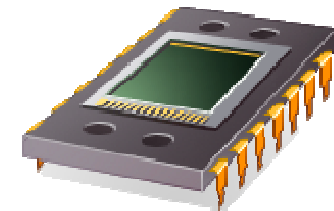
Aktuelle Situation

## Embedded Entwicklung heute:

- Es wird immer mehr objektorientiert entwickelt
- UML setzt sich als einheitliche Notation durch
- Es gibt aber auch Berührungsängste neue Techniken einzusetzen
- Größere und komplexere Projekte erfordern neue SW-Techniken

## Einsatz der UML für:

- Beschreibung der Anforderungen
- Beschreibung der Architektur
- Beschreibung des Designs der Implementierung



## Programmiersprache C

- Standardprogrammiersprache im embedded Umfeld
- Funktionale Sprache
- Nicht objektorientiert



Das zentrale Element der OOP – die Klasse – gibt es nicht in C.

Aber:

- Objektorientierung findet im Kopf statt und nicht in der Programmiersprache
- C bietet mehr Möglichkeiten als geahnt
- Regeln und Programmervorschriften können genutzt werden

## Implementierung von UML-Elementen

Viele Elemente der UML sind völlig unabhängig von der Programmiersprache.

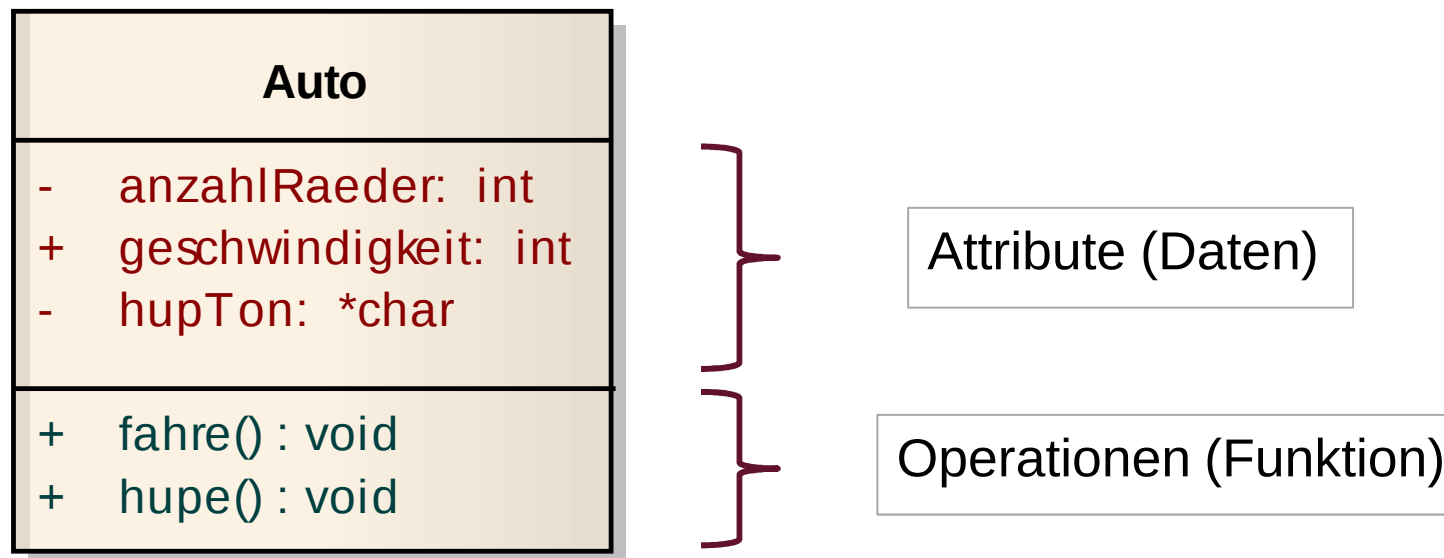
- Anforderungen
- Pakete
- Zeitliche Abläufe

Folgende Elemente aus dem UML-Klassendiagramm werden am häufigsten im Code umgesetzt:

- Klassen
- Beziehungen
  - Assoziation
  - Aggregation
  - Generalisierung (Vererbung)

Eines der wichtigsten Elemente in der UML ist die Klasse. Sie steht im Mittelpunkt der objektorientierten Programmierung.

In der Programmiersprache C gibt es kein Syntaxelement für die Klasse. Allerdings gibt es das `struct`-Element als Mittel, komplexe Daten strukturiert abzulegen.



Es wird eine Struktur mit zugeordneten Funktionen definiert.  
Der Zugriff auf die Struktur erfolgt nur über spezielle Funktionen.

```
typedef struct
{
    int anzahlRaeder;
    char* hupTon;
    int geschwindigkeit;
}Auto;

void Auto_init(Auto *this,
               int anzahlRaeder, char* hupTon);

void Auto_fahre(Auto *this);
void Auto_hupe(Auto *this, int anzahl);
```

Attribute (Daten)

Operationen (Funktion)

Die Zuordnung von Funktion und Struktur erfolgt über den ersten Parameter der Funktion – das ist immer ein Zeiger auf die Instanz der aktuellen Struktur. Dieser Parameter entspricht dem this-Pointer in C++.

Objekte können auf dem Stack oder dynamisch auf dem Heap angelegt werden.

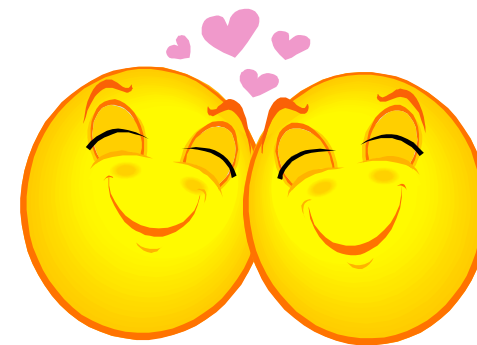
```
// Objekt auf dem Stack  
Auto a;  
  
Auto_init(&a, 4, "tut");  
Auto_fahre(&a);  
Auto_hupe(&a, 3);
```

Dank malloc( ) ist das dynamische Anlegen von Objekten auf dem Heap kein Problem.

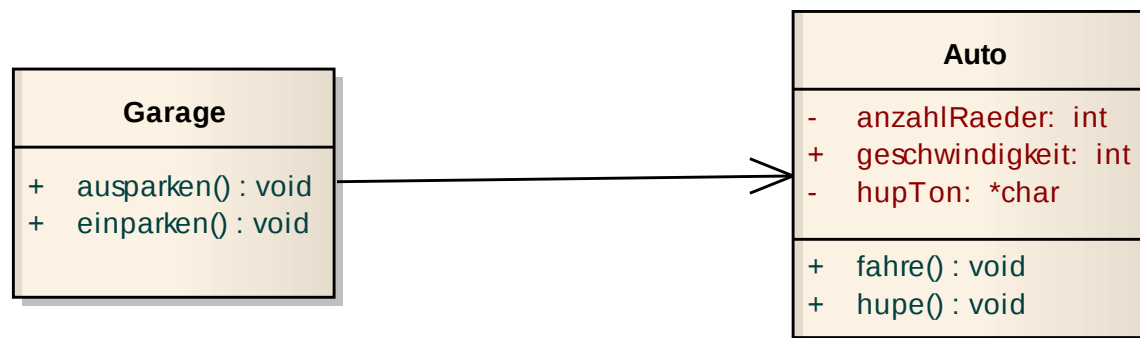
```
// Objekt auf dem Heap  
Auto *pa= (Auto*)malloc(sizeof(Auto));  
  
Auto_init(pa, 3, "maep");  
Auto_fahre(pa);  
Auto_hupe(pa, 2);  
  
free(pa);
```

Beziehungen beschreiben die Zusammenarbeit zwischen einzelnen Klassen.

- Klassen können sich kennen.
- Eine Klasse kann andere Klassen beinhalten.
- Eine Klasse kann Eigenschaften an eine andere Klasse weitergeben.



Die Assoziation ist eine einfache Beziehung zwischen zwei gleichrangigen Klassen.



Eine Klasse enthält einen Zeiger auf die andere Klasse.

```
typedef struct
{
    Auto* pAuto;
}Garage;

void Garage_init(Garage *this);
void Garage_init2(Garage *this, Auto *pAuto);
void Garage_einparken(Garage *this, Auto *pAuto);
void Garage_ausparken(Garage *this);
```

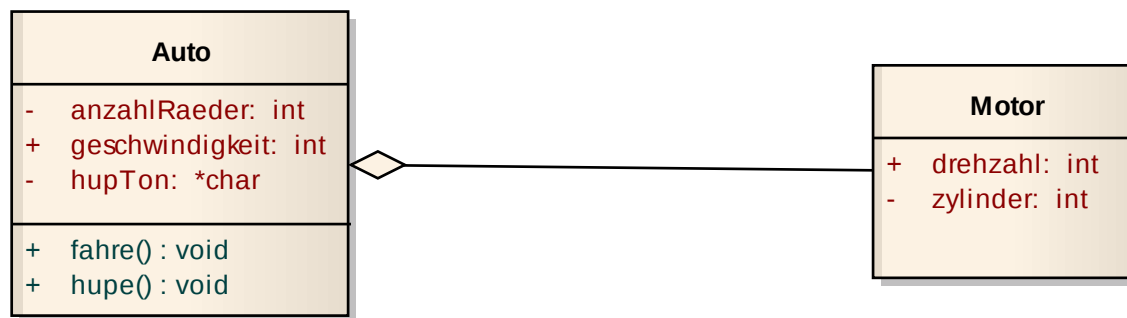
```
typedef struct
{
    int anzahlRaeder;
    char* hupTon;
    int geschwindigkeit;
}Auto;
```

Die Verbindung zwischen den Objekten wird erst zur Laufzeit des Programmes hergestellt.

```
Auto a;
Garage g;

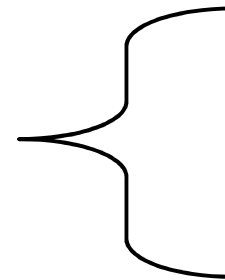
Auto_init(&a, 4, "tut");
Garage_init(&g);
Garage_einparken(&g, &a);
```

Die Aggregation ist eine „besteht aus“-Beziehung. Ein Objekt wird in ein anderes eingebettet.



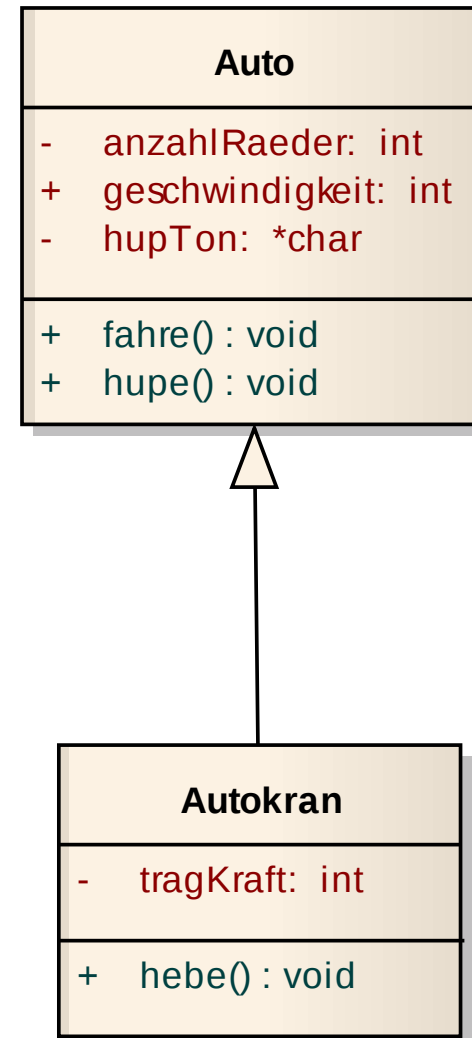
Eine Aggregation wird realisiert, indem das einzubettende Objekt als Member in die Struktur des äußeren Objektes aufgenommen wird.

```
typedef struct
{
    int anzahlRaeder;
    char* hupTon;
    int geschwindigkeit;
    Motor motor;
}Auto;
```



```
typedef struct
{
    int zylinder;
    int drehzahl;
}Motor;
```

Daten und Funktionen einer Basisklasse werden in einer abgeleiteten Klasse wiederverwendet und erweitert.



```
typedef struct
{
    int anzahlRaeder;
    char* hupTon;
    int geschwindigkeit;
    Motor motor;
}Auto;

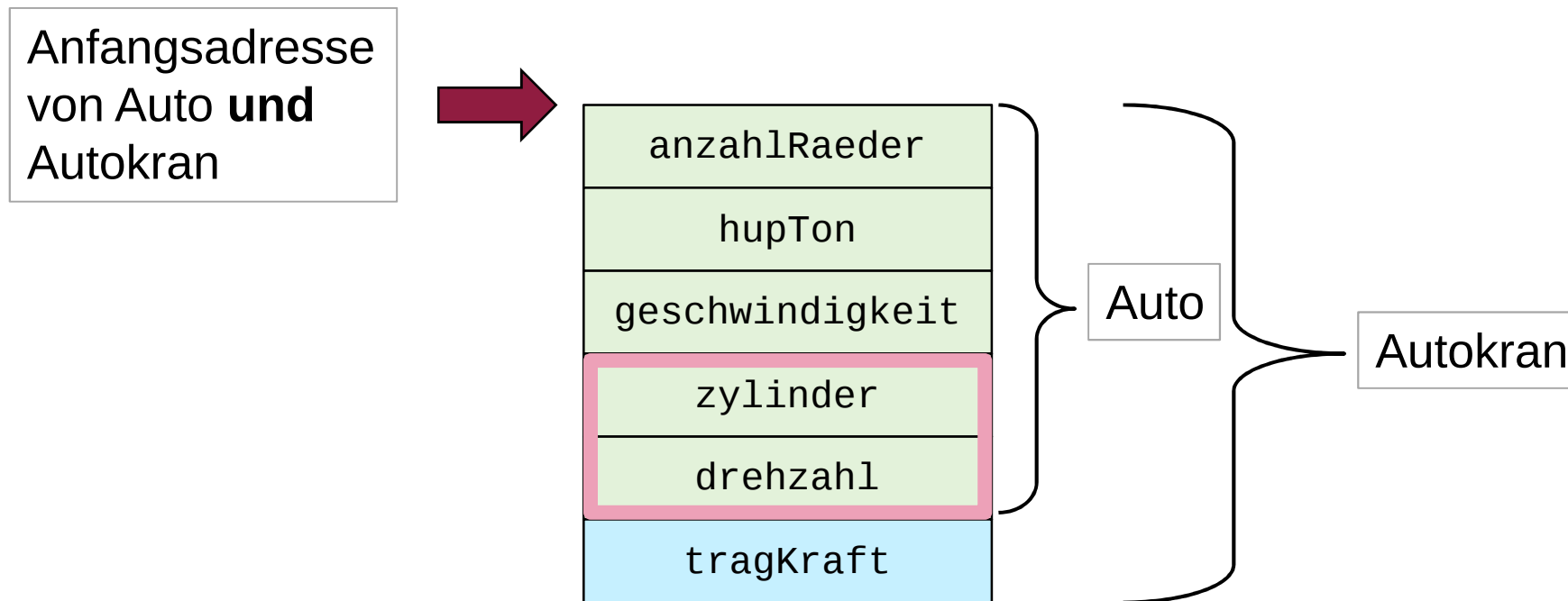
void Auto_init(Auto *this, int anzahlRaeder, char* hupTon);
void Auto_fahre(Auto *this);
void Auto_hupe(Auto *this, int anzahl);
void Auto_setGeschwindigkeit(Auto *this, int geschwindigkeit);
```

Die Vererbung in C ist ein Sonderfall der Aggregation (die Basisklasse ist das erste Element der Struktur).

```
typedef struct
{
    Auto;
    int tragKraft;
}Autokran;

void Autokran_init(Autokran *this, int tragKraft);
void Autokran_hebe(Autokran *this, int gewicht);
```

Die Struktur Autokran im Speicher:



Durch dieselbe Anfangsadresse können die Funktionen der Klasse Auto auch für Autokran-Objekte verwendet werden.

```
void test(void)
{
    Autokran kran;
    Autokran_init(&kran, 50);

    Auto_setGeschwindigkeit((Auto*)&kran, 20);
    Auto_hupe((Auto*)&kran, 3);
    Auto_setGeschwindigkeit((Auto*)&kran, 0);

    Autokran_hebe(&kran, 20);
    Autokran_hebe(&kran, 60);
}
```

Ruft intern Auto\_init() auf

Funktionen der Basisklasse können problemlos benutzt werden.

Schutz der Daten gegen direkten Zugriff

Direkter Zugriff → FALSCH



a.geschwindigkeit= 10;

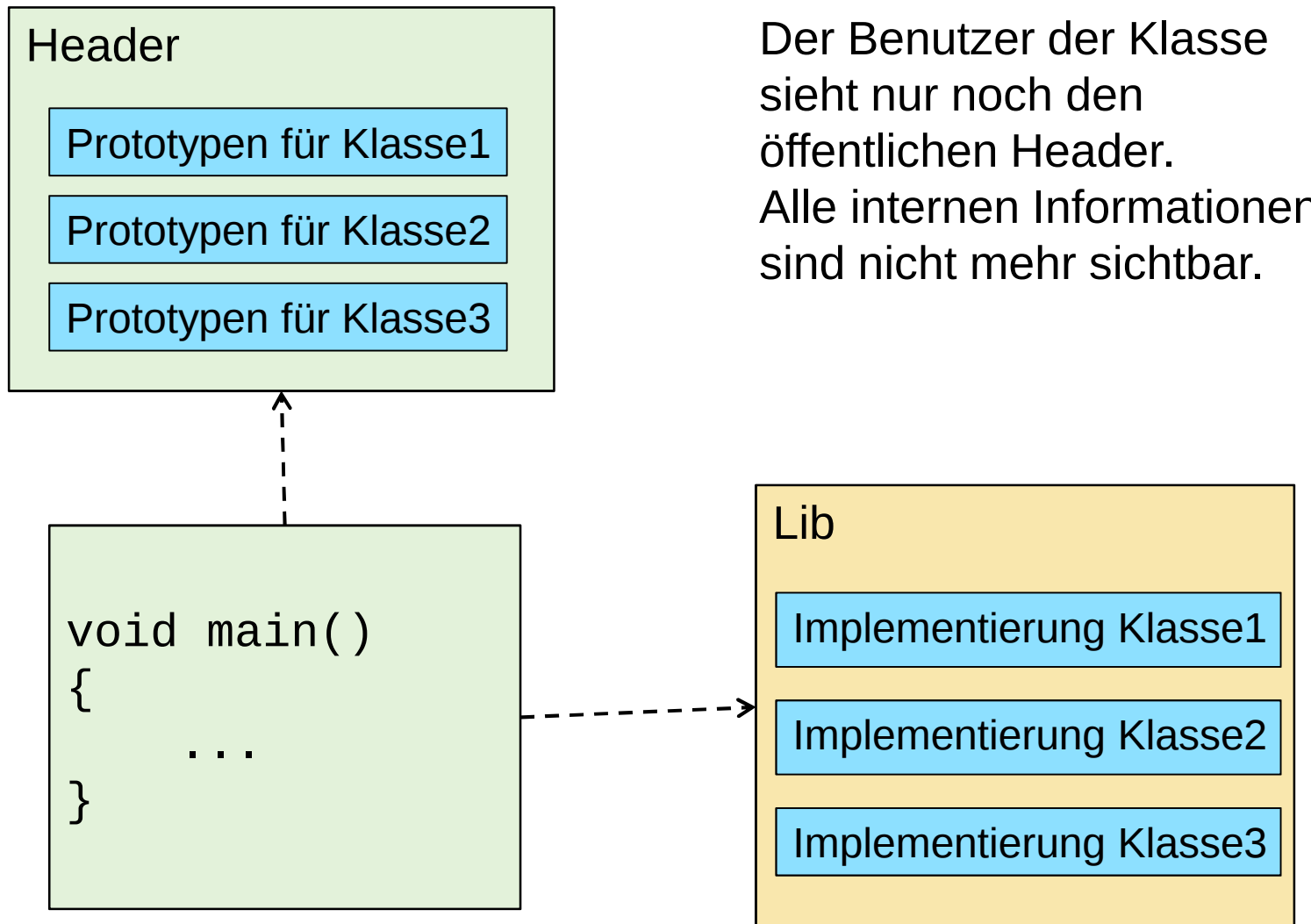
```
typedef struct
{
    int anzahlRaeder;
    char* hupTon;
    int geschwindigkeit;
    Motor motor;
}Auto;
```

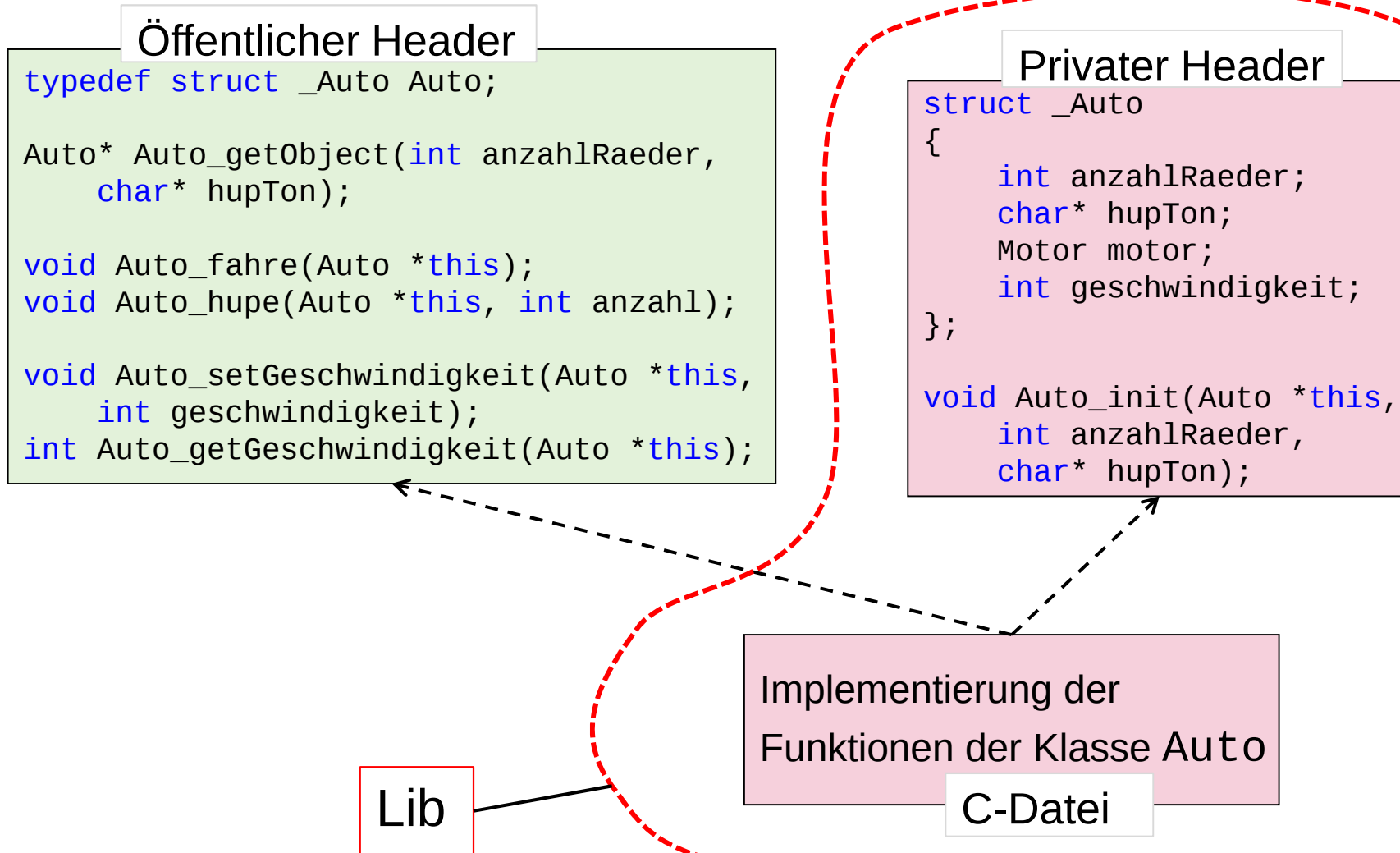
Zugriff über Funktion → RICHTIG

Auto\_setGeschwindigkeit(&a, 10);

Direkter Zugriff kann nur über Programmierrichtlinien und regelmäßige Reviews verhindert werden.

ODER: Die Struktur wird versteckt.

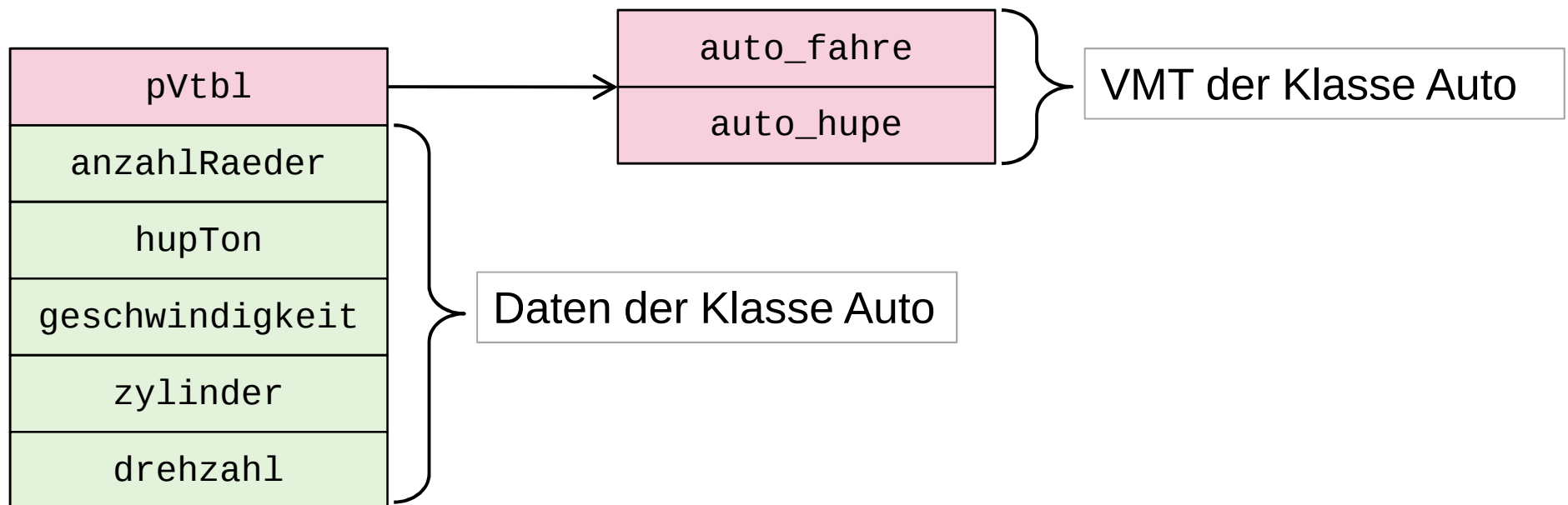




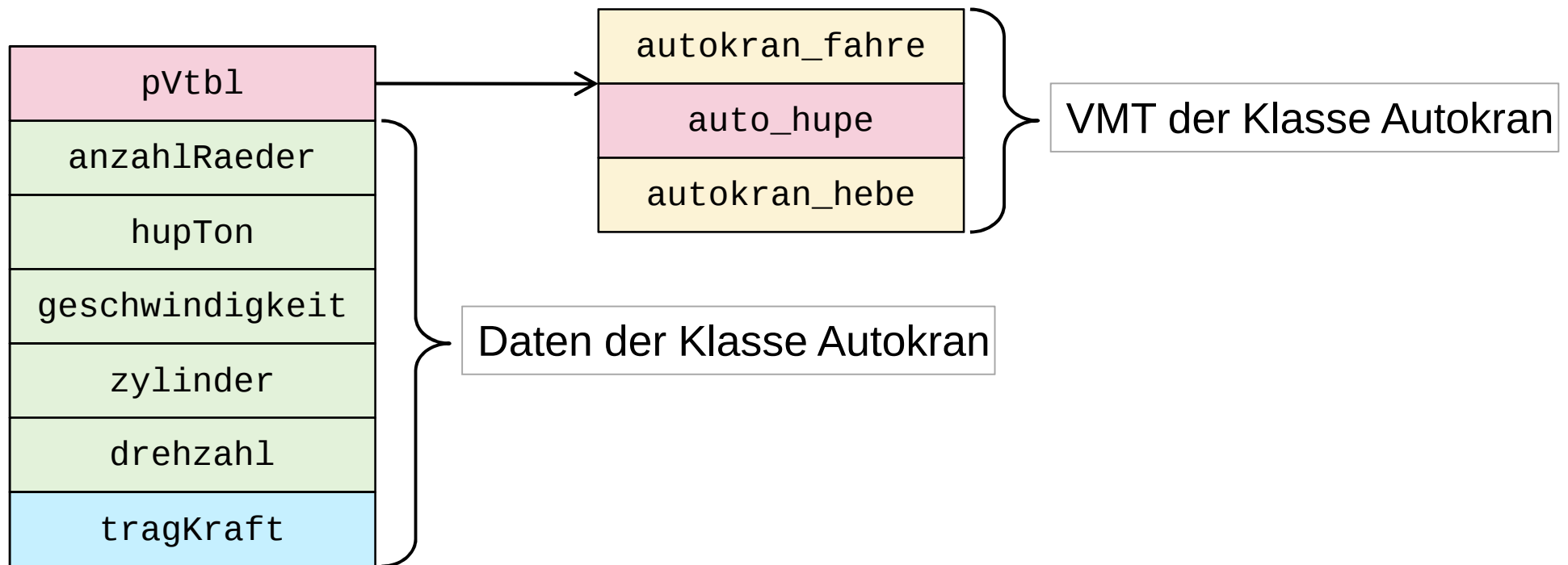
Der Aufbau der Struktur ist nur innerhalb der Lib bekannt.  
Nur die Funktionen im öffentlichen Header sind den Nutzern bekannt.

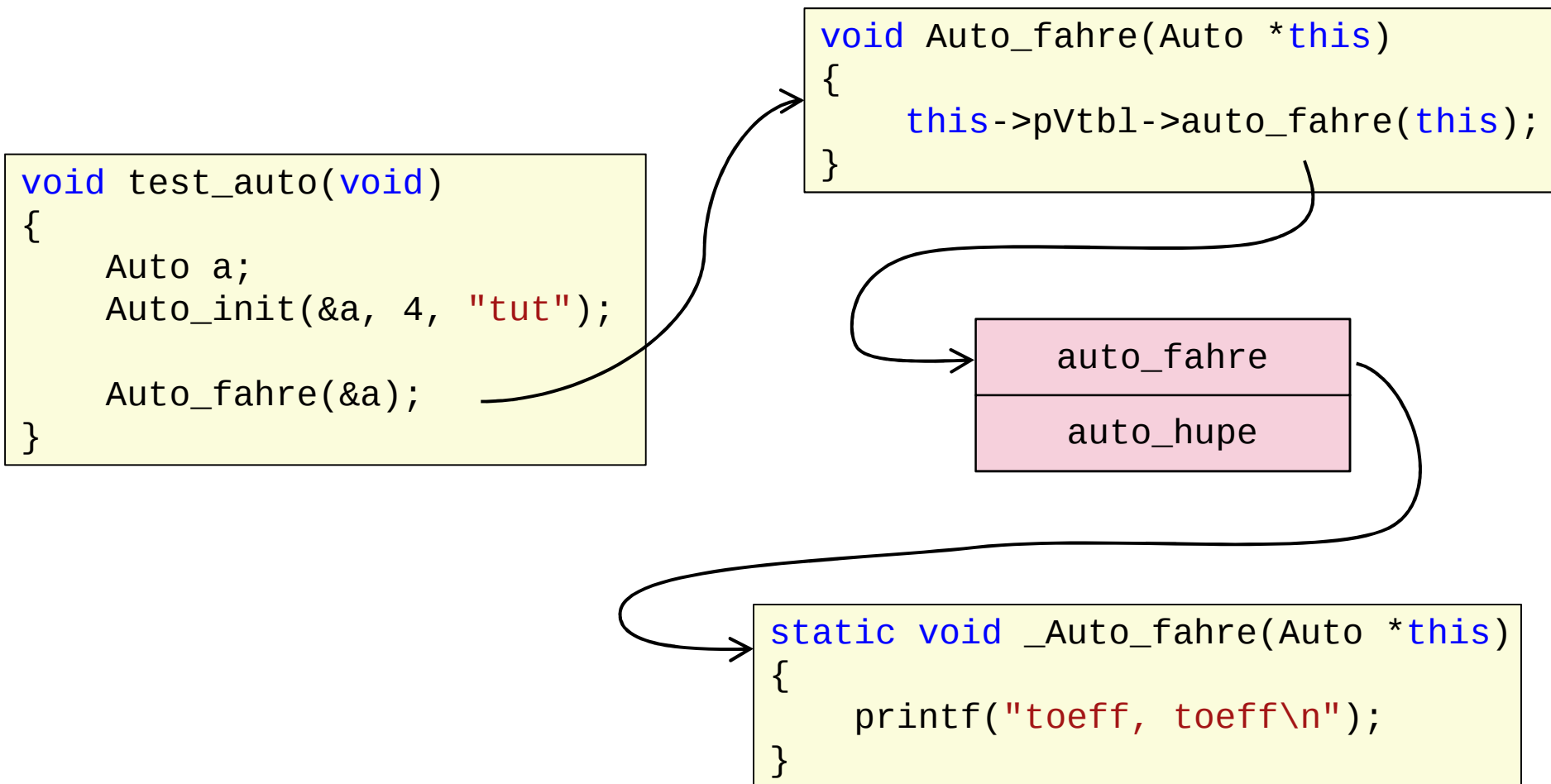
## Virtuelle Methoden

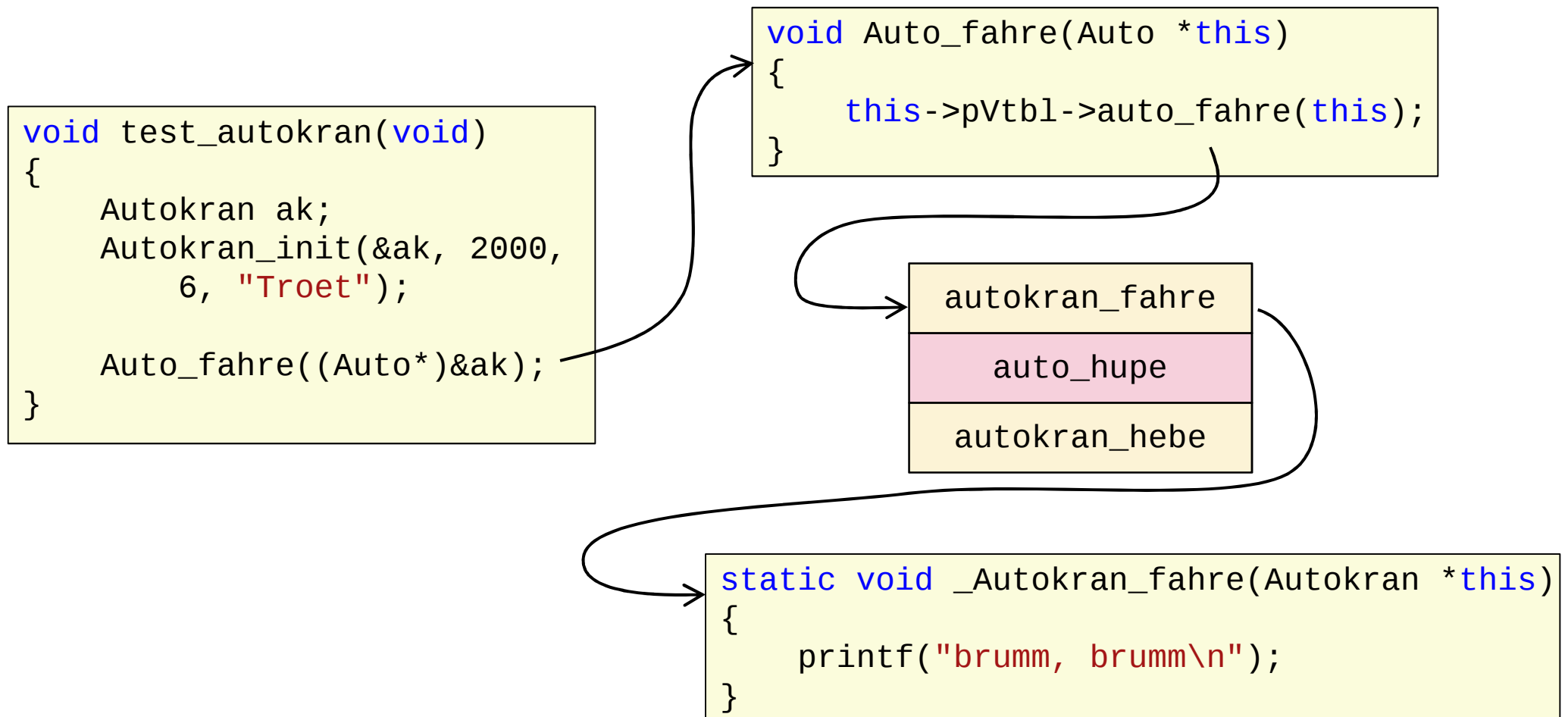
Um in einer abgeleiteten Klasse Methoden überschreiben zu können, muß die Basisklasse mit einer Virtuellen-Methoden-Tabelle (VMT) ausgestattet werden. In diese Tabelle werden die Funktionszeiger aller überschreibbaren Methoden aufgenommen.



Die abgeleitete Klasse hat ihre eigene VMT. Hier werden je nach Bedarf, die Methoden der Basisklasse durch eigene ersetzt.

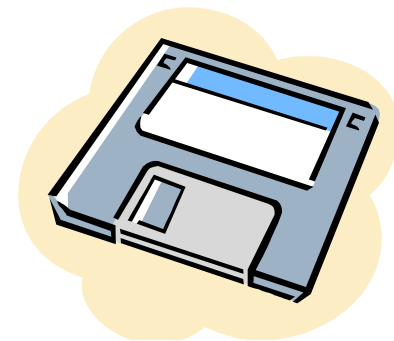






Setzen Sie moderne Methoden der SW-Entwicklung auch unter C ein.

Die Programmiersprache C kann mehr als auf den ersten Blick ersichtlich.



Im Internet sind Beispiele für den Vortrag als Download unter folgender Adresse verfügbar:

[http://download.microconsult.net/ESE2009/UML\\_C.zip](http://download.microconsult.net/ESE2009/UML_C.zip)

Bitte beachten Sie, daß der komplette Pfad inklusive Dateinamen angegeben wird.