

Kapitel 5

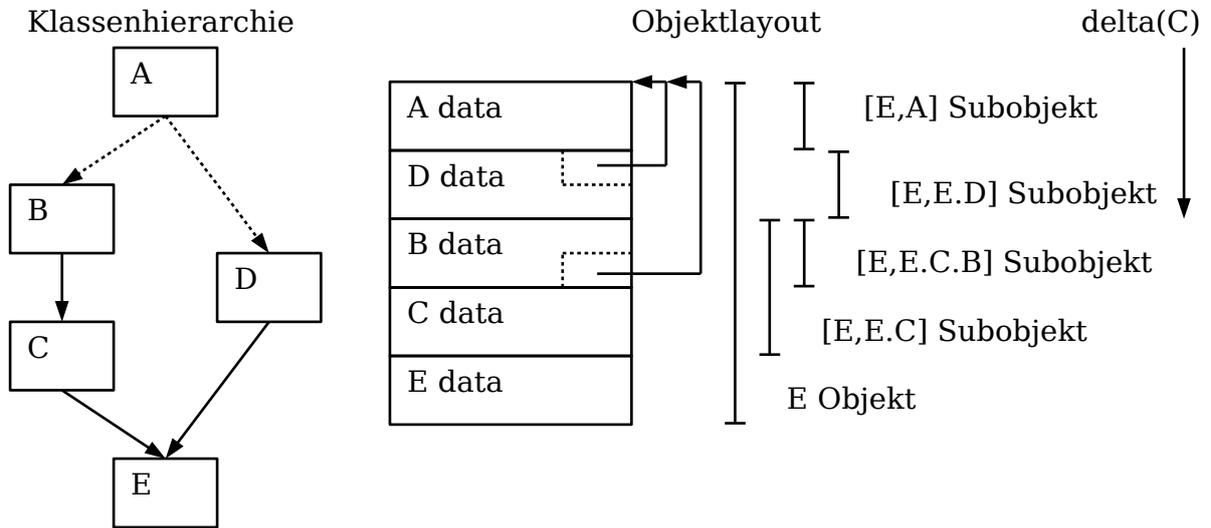
Der vtable-Mechanismus

5.1 C++: Objektlayout

klassisch (Erinnerung):

- nur Data Members im Objekt, in der Reihenfolge ihrer Deklaration; feste Offsets
- Oberklassen-Subobjekte liegen physikalisch *vor* den eigenen Members
- nichtvirtuelle Mehrfachvererbung kann deshalb Subobjektkopien einführen
- virtuelle Mehrfachvererbung verwendet Pointer, um Mehrfachkopien zu eliminieren

Beispiel:



5.2 C++: Type Casts

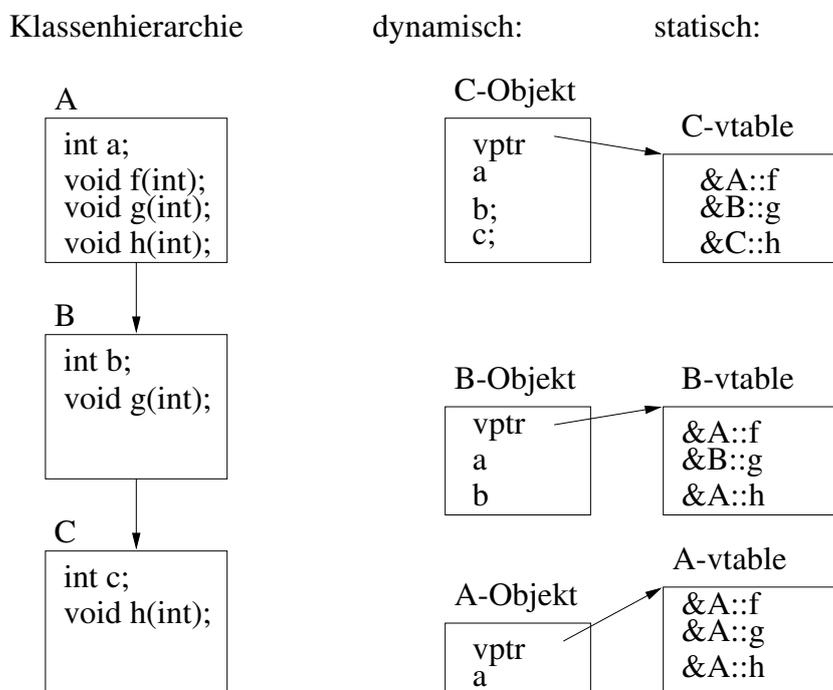
- bei nichtvirtueller Einfachvererbung: Nullcode!
- bei nichtvirtueller Mehrfachvererbung
 $C : A, B$ mit $B^* pb; C^* pc$:
 $pb = pc; \text{ bzw } pb = (B^*)pc$; wird zu
 $pb = (B^*)((\text{char}^*)pc) + \text{delta}(B)$;
 mit $\text{delta}(B)$ der Offset des B -Subobjekts in einem C -Objekt
- $\text{delta}(B)$ ist zur Compilezeit bekannt
- bei virtueller Vererbung: Verfolgen des Subobjekt-Pointers
 $pb = (B^*)(*((\text{char}^*)pc) + \text{delta}(B))$;
- 0-Pointer werden nicht gecastet
- Auch in expressions zB `if (pc == pb)`: impliziter Type Cast
- Auch bei Aufruf einer Oberklassenmethode: impliziter Typecast für den `this`-Pointer!

5.3 C++: vtables

Standardimplementierung für virtuelle Methoden

- für jede Klasse gibt es eine (statische) “vtbl” (virtual table)
- enthält Einsprungadressen f. zugreifbare Methoden
- jedes Objekt enthält Pointer auf vtbl seiner Klasse

Beispiel:

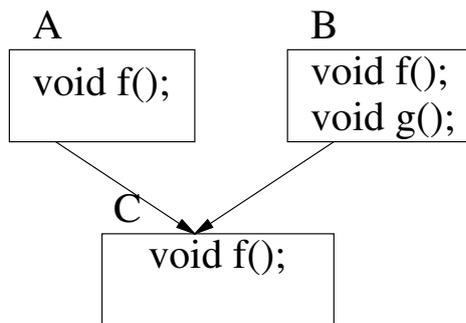


- Methoden-Indizes sind für jeden Methoden-Namen global eindeutig! (werden vom Compiler vergeben)
- beim Aufruf zusätzlicher Indirektionsschritt:
`C* pc = new C; pc->h(42);` wird realisiert als
`(*(pc->vptr[2]))(pc, 42)`

5.4 Mehrfachvererbung

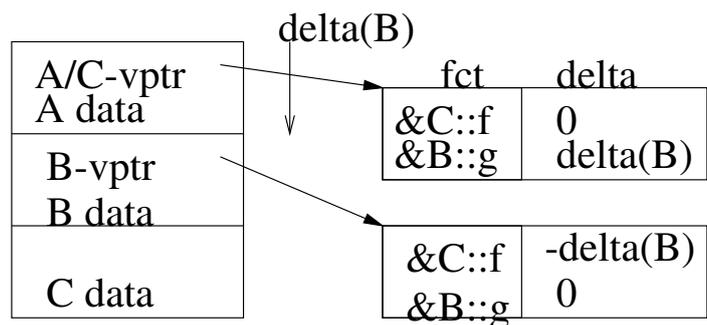
Beispiel:

Klassenhierarchie



Laufzeitstrukturen

C-Objekt



B-Objekt



Beispielaufruf:

```
B* pb = new C; pb->f()
```

ruft C::f()

⇒ this-Pointer muß auf C-Objekt zeigen.

pb zeigt aber auf B-Subobjekt!

⇒ this-Pointer muss gecastet werden!

jedoch: Subobjekt-Deltas sind nicht mehr zur Compilzeit bekannt (s.u.)!

⇒ speichere Subobjekt-Deltas zur Laufzeit i. d. vtbl!

Aufruf

```
B* pb; C* pc;  
pc = new C; pb = pc; pb->f(42);
```

wird zu

```
register vt = &(pb->vptr[0]);  
(*vt->fct)(pb+(vt->delta), 42)
```

Übung: Ergänzen Sie notwendige Casts!

- **Achtung:** B-Subobjekt in C-Objekt hat andere vtbl als gewöhnliches B-Objekt! (denn Upcasts schalten nicht die dynamische Bindung ab)

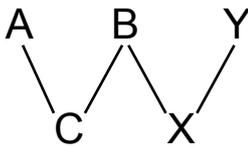
Allgemein haben alle Subobjekte eigenen vptr/vtbl, außer denen, die zur „linken Außenkante“ der Hierarchie gehören

Dies ist konsistent mit dem Einfachvererbungs-Fall

- **Achtung:** i.a. ist der Kontext von Subobjekt-Zeigern nicht bekannt: pb könnte auch auf B-Subobjekt eines X-Objektes zeigen. Wie kann der this-Pointer richtig gecastet werden?
- **Unterschied zu Upcasts:** bei Upcasts ist Ausgangsklasse bekannt, so dass Compiler delta statisch bestimmen kann

dazu Beispiel:

- angenommen, außer $C : A, B$ es gibt weitere Klasse $X : Y, B$, die f redefiniert



⇒ auch X -Objekt hat B -Subobjekt, das jedoch andere Relativadresse hat als das B -Subobjekt im C -Objekt:

$$\text{delta}_C(B) \neq \text{delta}_X(B)$$

- Nun betrachte

```

if (...)
  pb = new C();
else
  pb = new X();
pb->f();
  
```

pb zeigt auf $[C, C \cdot B]$ Subobjekt oder $[X, X \cdot B]$ Subobjekt
Dynamische Bindung ruft entweder $C::f$ oder $X::f$

⇒ pb muss entweder nach C oder X gecastet werden, damit es korrekter $this$ -Pointer im Methodenrumpf ist

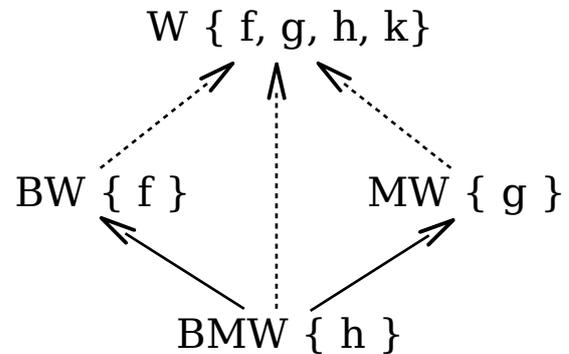
- Da zur Compilezeit nicht bekannt ist, ob X oder C , und $\text{delta}_C(B) \neq \text{delta}_X(B)$, müssen die Delta-Werte zur Laufzeit gespeichert werden!

Beispiel 2: verteilte Implementierung (→ Kapitel 2):

```

class W {
    virtual f();
    virtual g();
    virtual h();
    virtual k();
};
class MW : virtual W {
    g();
};
class BW : virtual W {
    f();
};
class BMW : BW, MW,
    virtual W {
    h();
}

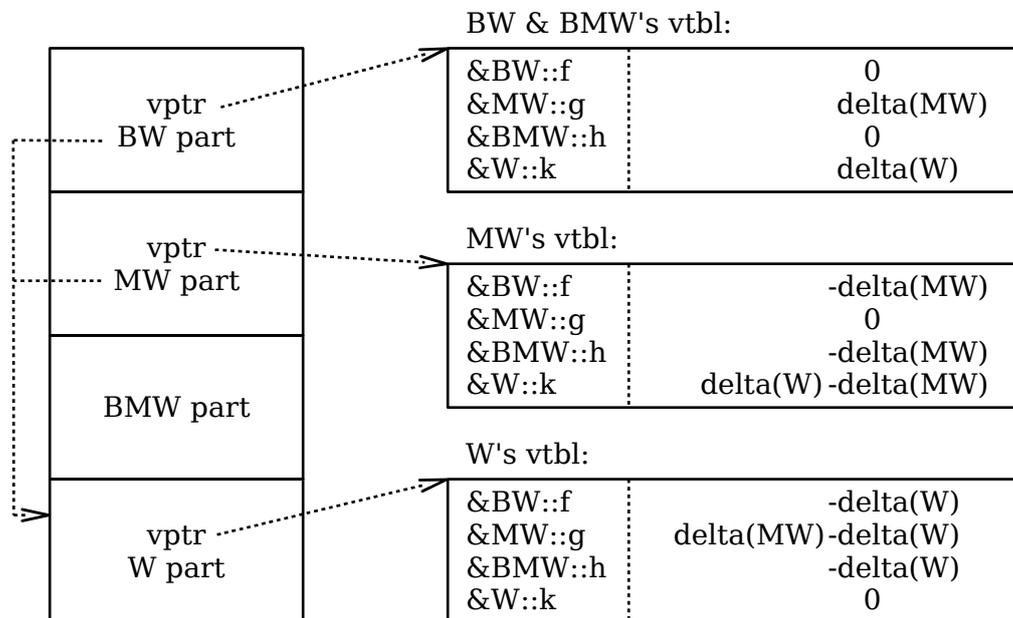
```



Aufruf `BMW* pbmw; MW* pmw = pbmw; pmw->f();` ruft `BW::f()` !
 (→ Static Lookup)

Dieses Verhalten ist sinnvoll.

Objekt-Layout und vtbls:



Code für Aufruf `pmw->f(42)`; incl. this-Pointer :

```
register vt = &(pmw->vptr[0]);
// f hat index 0
(*vt->fct)(pmw+(vt->delta), 42)
```

Kostenberechnung: `fct`, `delta` sind konstante Offsets \Rightarrow Autoincrement-Maschinenops

ergo 3 Dereferenzierungen, 1 Indexzugriff (Addition), 1 Addition, plus regulärer Methodenaufruf

`delta(W)` ist physikalischer Offset, da man Pointer auf W-Subobjekt für Downcasts nicht verwenden kann. Zugriffe auf Instanzvariablen von W hingegen über diesen Pointer wg Effizienz