

Der Aufbruch in die OO-Welt

Erfahrungsbericht



Mielert, Kurt
DV-Organisation

Köln,
11. März 1999

wer bin ich?

- Kurt Mielert - LVM-Versicherungen
- seit über 25 Jahren in der DV-Welt tätig
 - davon 20 Jahre beim LVM in der AE

... über Lochkarte

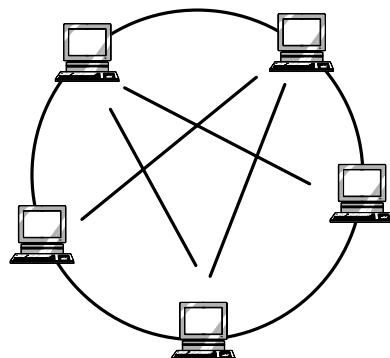
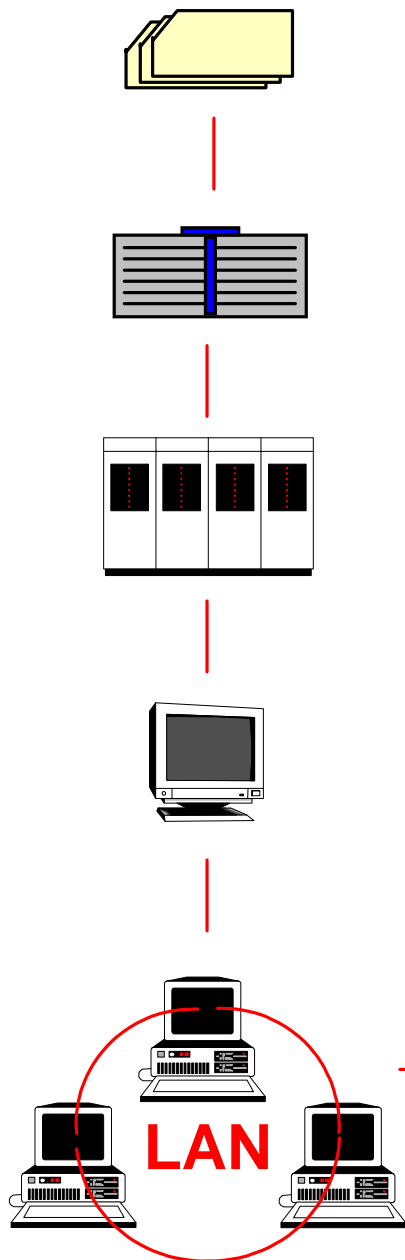
... die ersten Magnetplatten

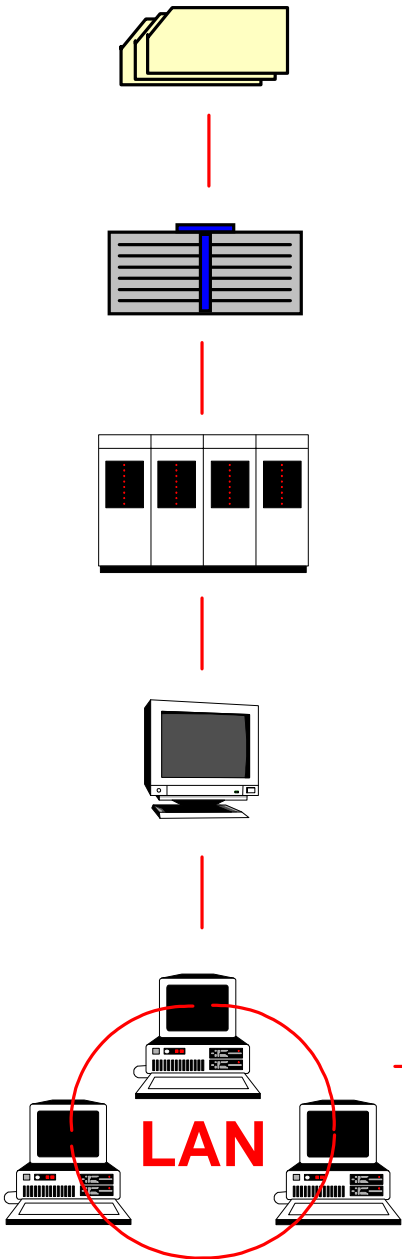
... die ersten zentralen Mainframes

... die ersten Onlineanwendungen (Host)

... die ersten vernetzten PC-Systeme

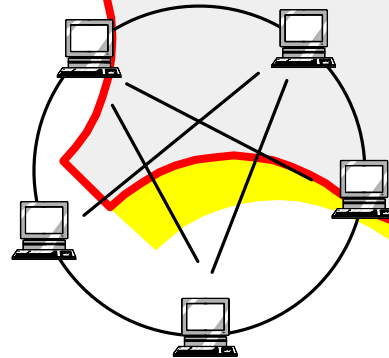
... zu globalisierten Netzwerken





**Die Vergangenheit
war schon spannend**

**Die Zukunft wird
noch spannender**



Themenübersicht

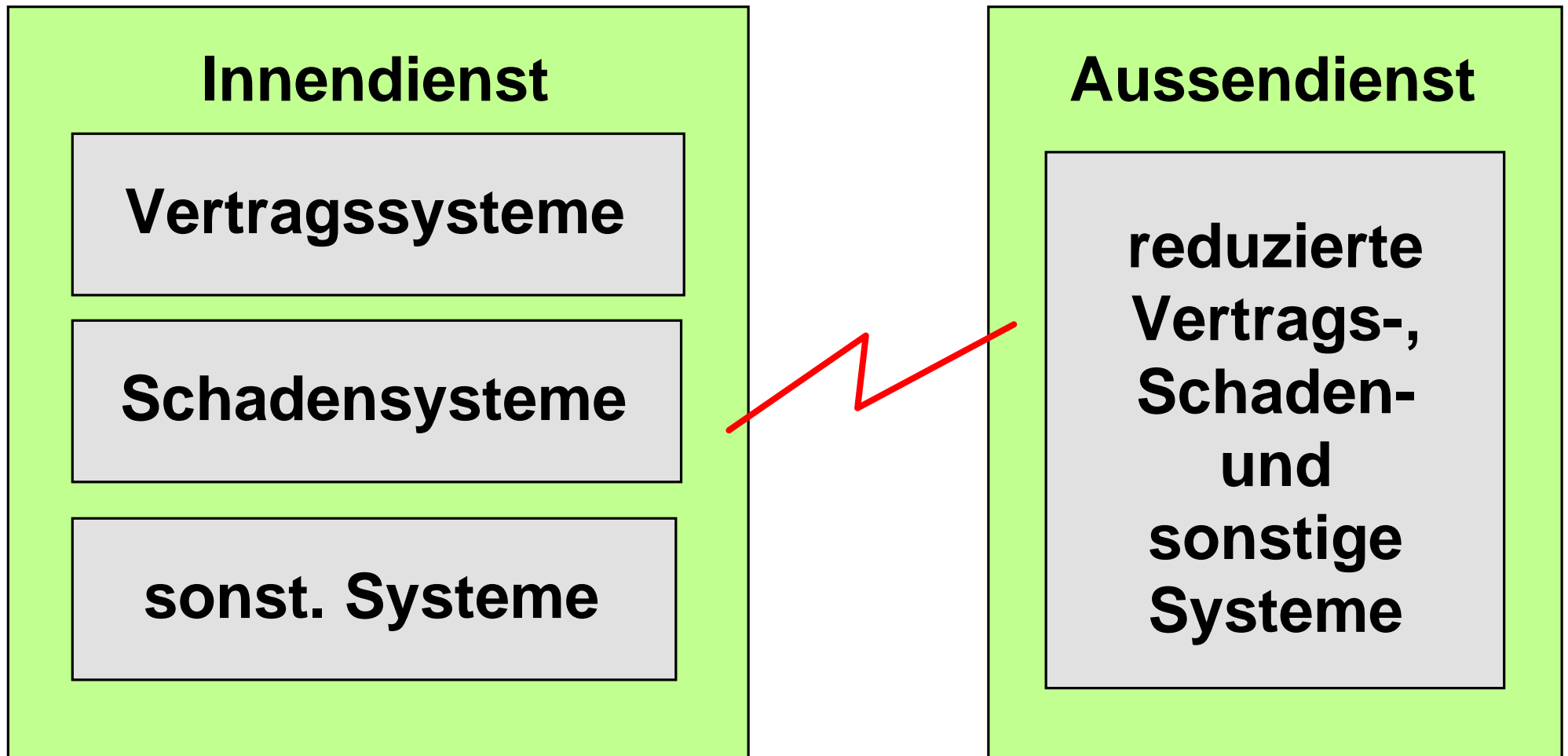
- Ausgangssituation beim LVM
 - warum neue OO-Anwendungen?
- Ein 'paar' Sätze zu unserem Projekt
 - Anforderungen an ein neues Schadensystem
- OO-Welt - was ist das eigentlich?



VA for JAVA
Klassen
objektorientierte AE
grafische Benutzeroberfläche
Use-Cases
verteilte Anwendungen
Objektmodell
plattformübergreifend

- 'der Sprung ins kalte Wasser'
- Erfahrungen eines Jahres in der OO-Welt

Ausgangssituation



können nur über Datenträgeraustausch
miteinander kommunizieren

Ausgangssituation

- Neuentwicklungen und Änderungsanforderungen,
- die für den Innen-, und Außendienst relevant sind,
- werden sowohl im Host-,
- als auch im Agentur-System realisiert.



d.h.: doppelter Aufwand für Analyse,
Programmierung und Test

Ausgangssituation

- **Operative Datenbestände werden nur in bestimmten Zyklen dem Außendienst übermittelt.**
- **Inhalte aus Beitrags-, Schlüssel-, und Beziehungstabelle(n) (-Dateien) werden über aufwendige Mechanismen dem Agentur-System übermittelt.**



d.h.: gleicher Informationsstand ist nur unmittelbar nach der DFÜ gewährleistet

Ausgangssituation

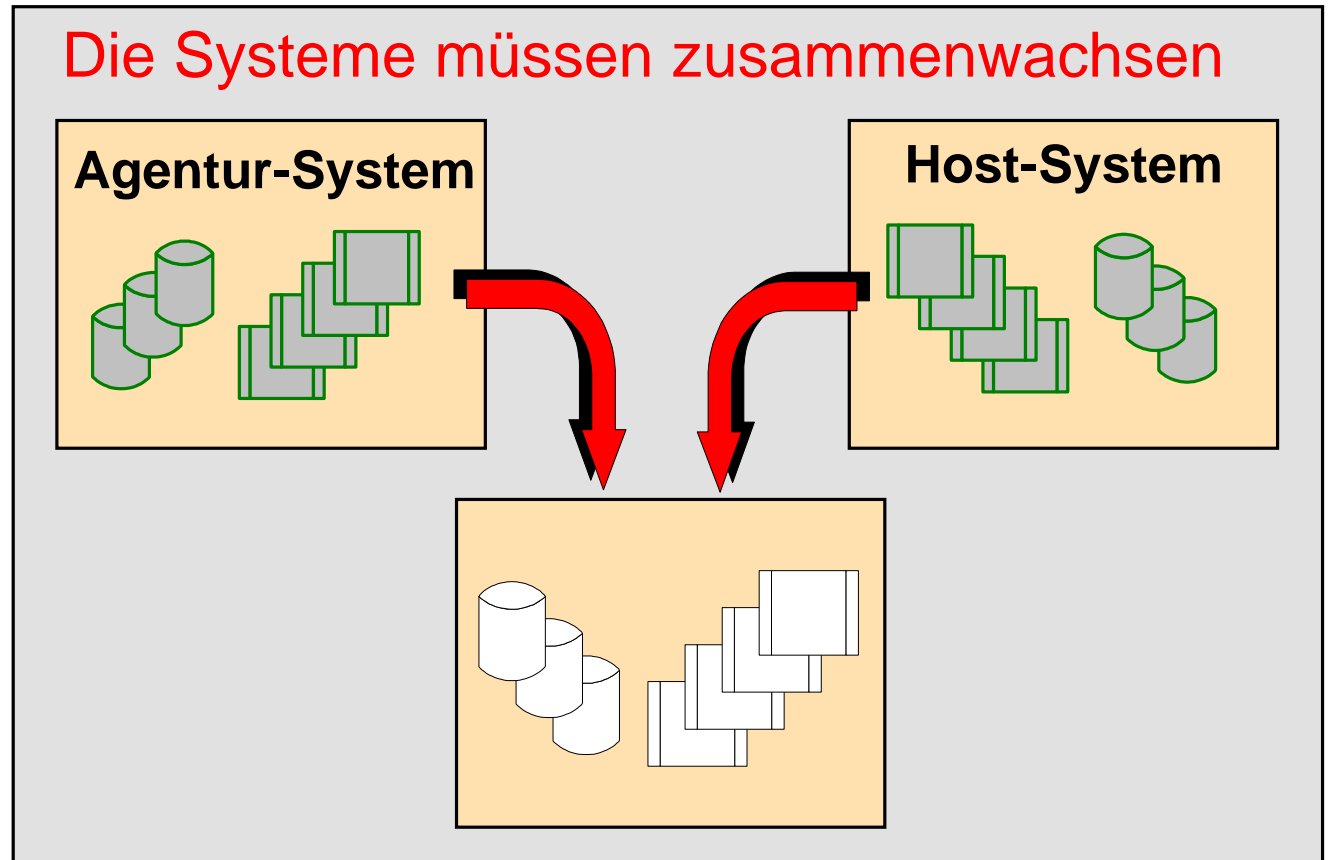
- Es gibt Agentur-Anwendungen, die dem Innendienst nicht zur Verfügung stehen, obwohl dieses sinnvoll wäre.
- und umgekehrt: Host-Anwendungen, die dem Außendienst nicht zur Verfügung stehen, obwohl ...



d.h.: diese Anwendungen müßten für die jeweils 'andere Seite' zusätzlich entwickelt werden

Zieldefinition

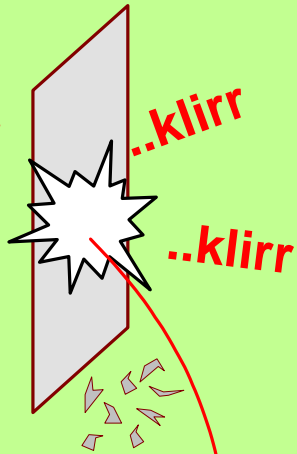
**Strategisches Ziel
ist,
Softwareprodukte
einzusetzen,
die für
a l l e Anwender
nutzbar sind.**



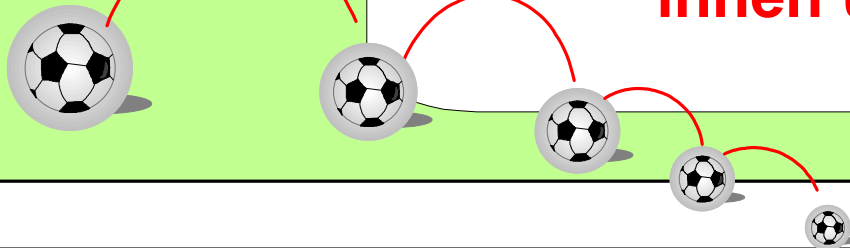
**d.h.: Die Trennung zwischen Agentur- und Hostsystemen
wird es zukünftig (langfristig) nicht mehr geben**

unser Projektauftrag

Entwicklung eines Schadenssystemes für den **Außen- und Innendienst,**
das die Regulierung möglichst vieler Schäden
fallabschließend und elektronisch unterstützt.

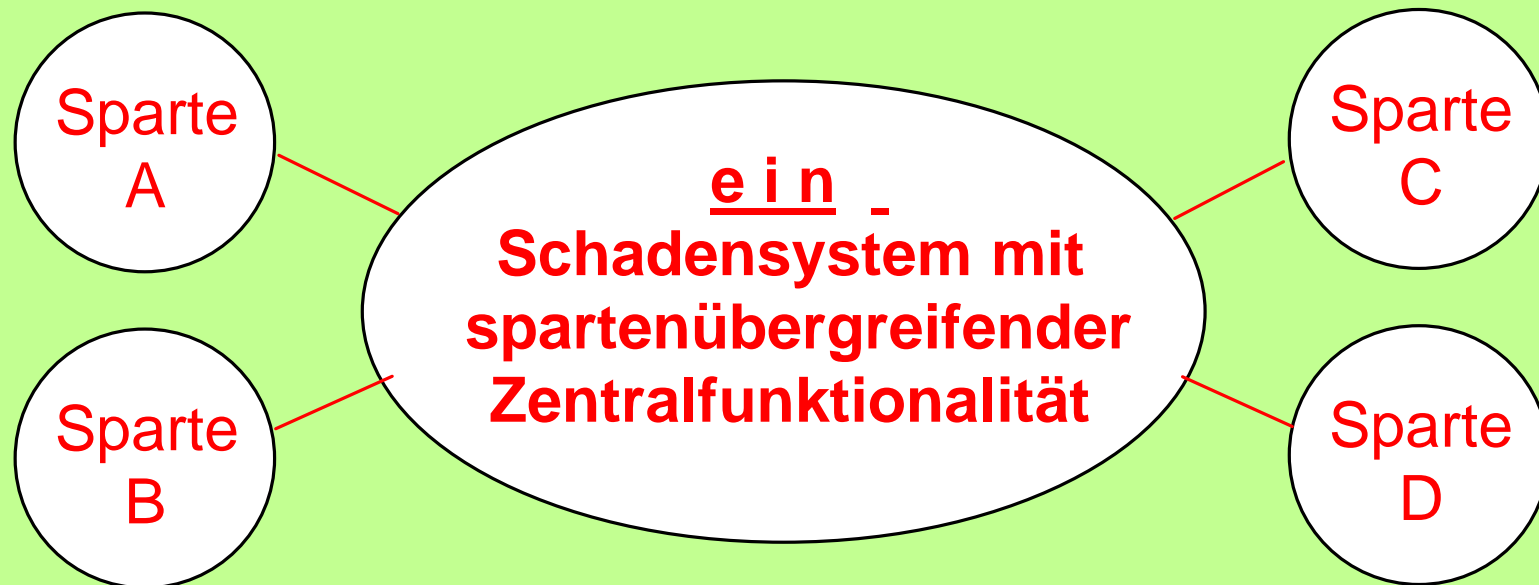


**gemeinsame Programme mit
gemeinsamer Datenbasis,
aber dem Bedarf entsprechender
Benutzeroberfläche für
'innen und aussen'**



unser Projektauftrag

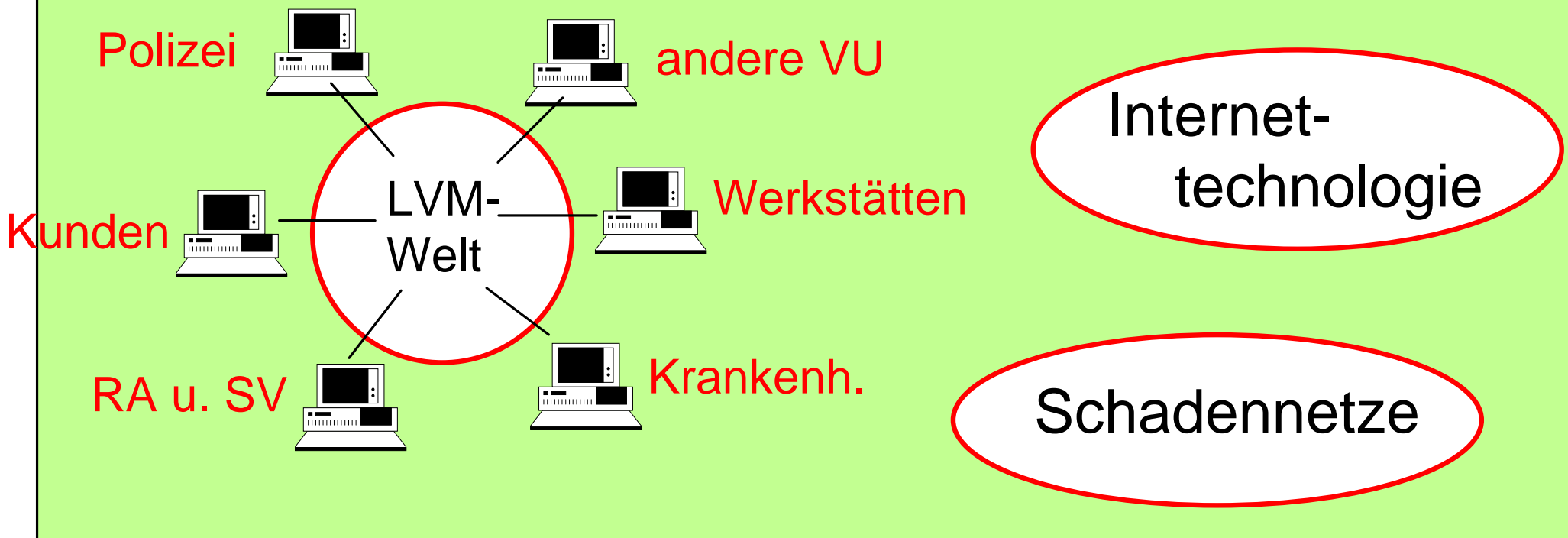
Die Systemstruktur wird ein spartenunabhängiges Schadenssystem enthalten, an das sukzessive die einzelnen Sparten angegliedert werden können.



ein weiteres wichtiges Ziel

Ausbau der Servicefunktionalität

- einbeziehen aller am Schadenprozeß beteiligten Personen und Gruppen.
- Informationsaustausch über 'LVM-Grenzen' hinaus



wenn wir Programme für INNEN und AUSSEN
nur einmal entwickeln wollen,
und zukünftig die technischen Möglichkeiten nutzen wollen,
dann geht das n u r:

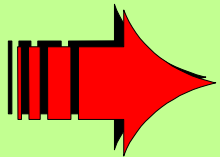
➤ **plattformunabhängig**

➤ losgelöst von verschiedenen Betriebssystemen

➤ **objektorientiert**

und mit **JAVA**

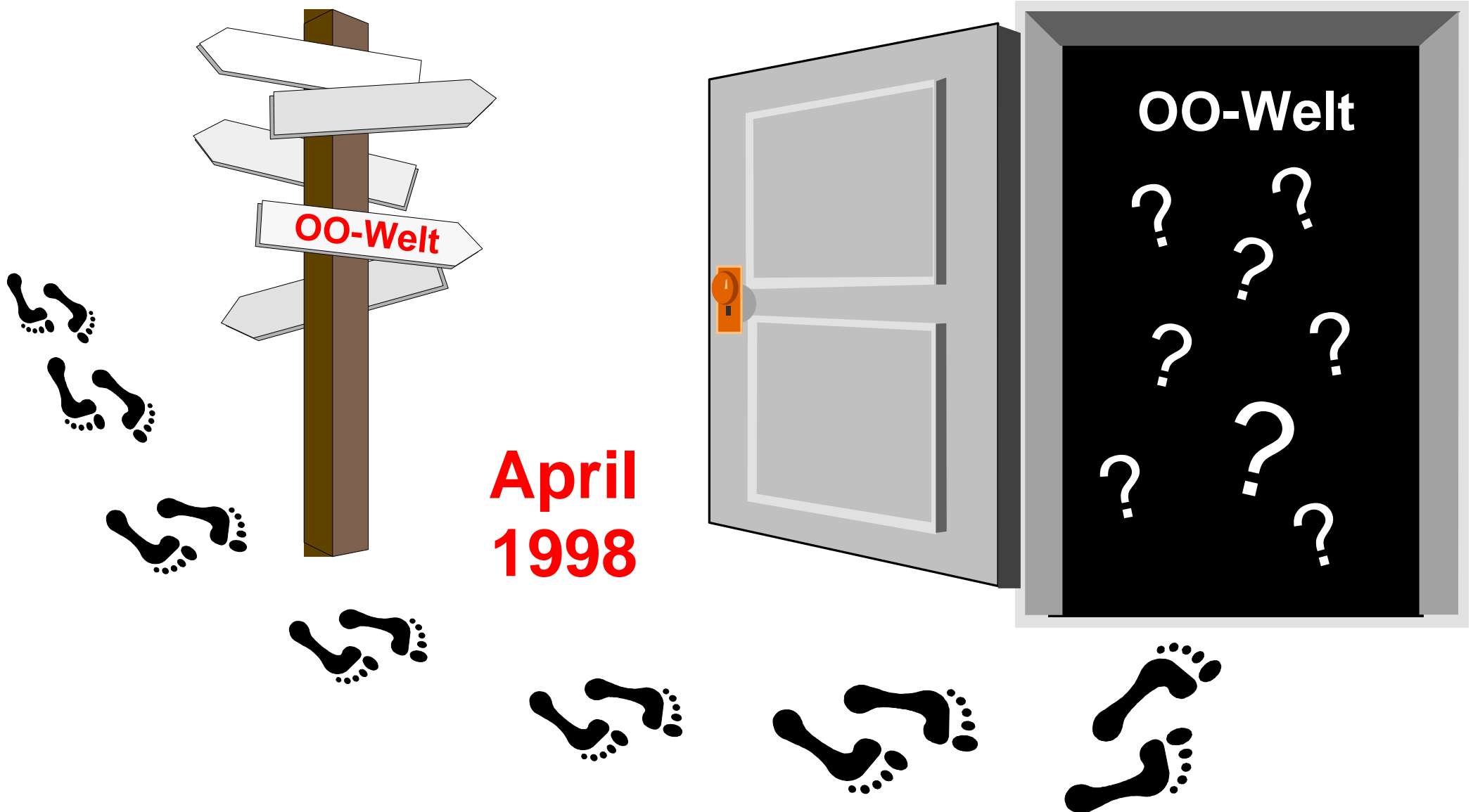
Java bietet die besten Voraussetzungen
für Plattformunabhängigkeit und ist
eine objektorientierte Sprache



das erfordert den Einstieg in die OO-Welt



Der Aufbruch in die OO-Welt



... anno April 1998

Objekte + Klassen ?

Visual Age for Java ?

3-Tier-Architektur ?

plattformübergreifend ?

San Francisco ?

Pattern ?

Persistenz ?

verteilte Applikationen ?

Use Cases ?

GUI ?

Framework ?

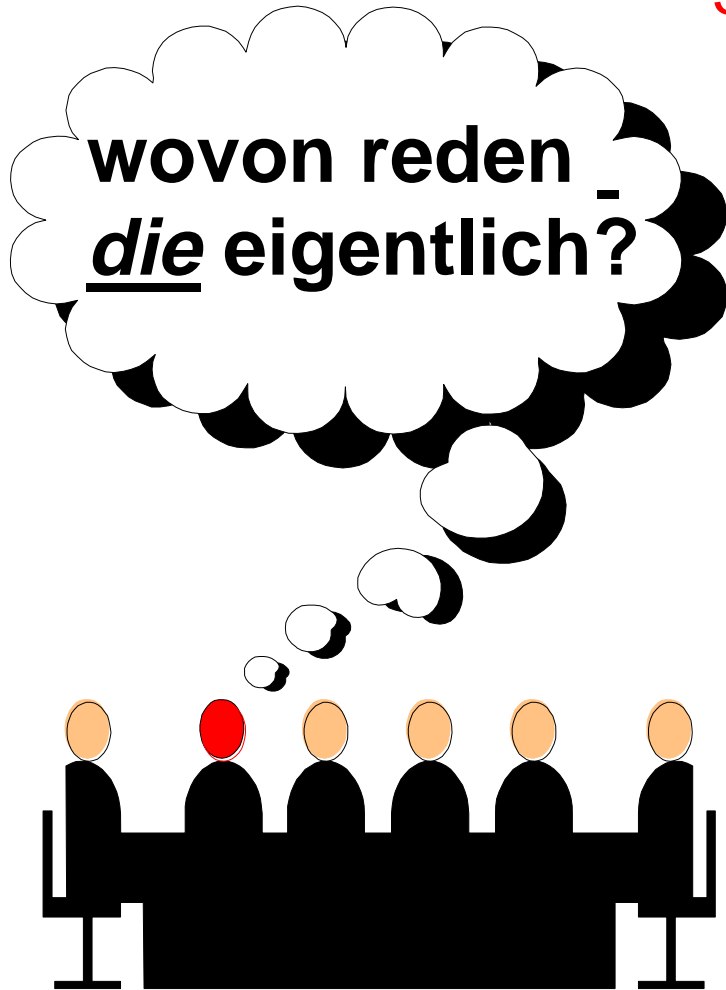
Präsentation ?

Komponenten ?

thin client ?

PGO, AGO, EGO, ... ?

... usw.



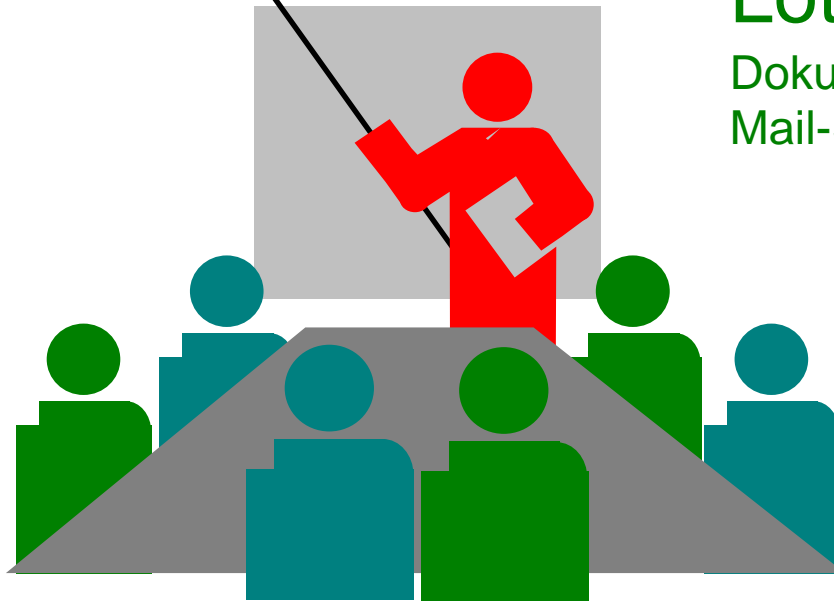
Ausbildung

Objektorientierung

Grundlagen / Analyse u Design

WSDDM

Worldwide Solution Design
and Delivery Methods
Vorgehensmodell



Innovator

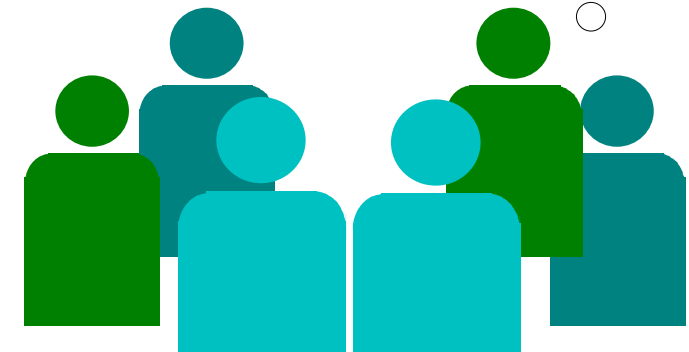
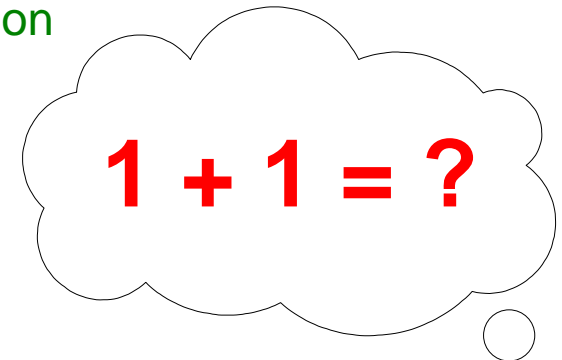
Modellierungswerkzeug
für Objektmodelle und
Use-Case-Dokumentation

Lotus Notes

Dokumentation und
Mail-Software

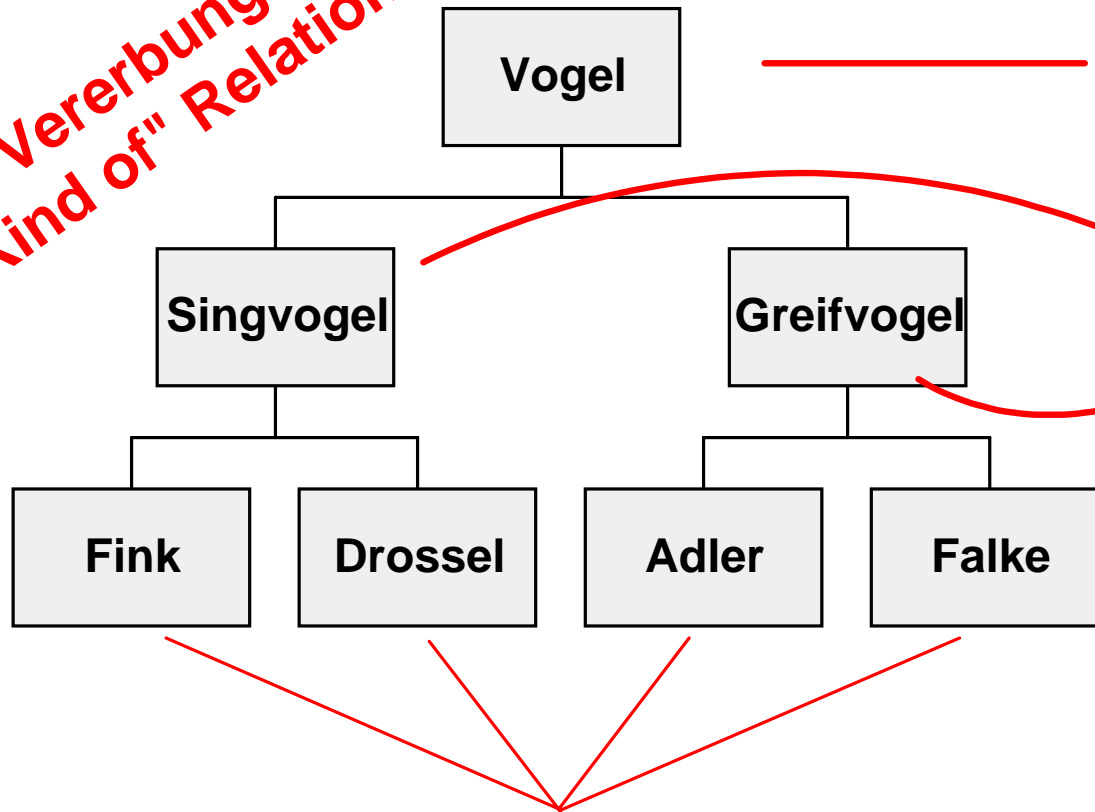
VA for Java

objektorientierte
Programmiersprache



Ein 'kleines' OO-Beispiel

Vererbung
"Kind of" Relation



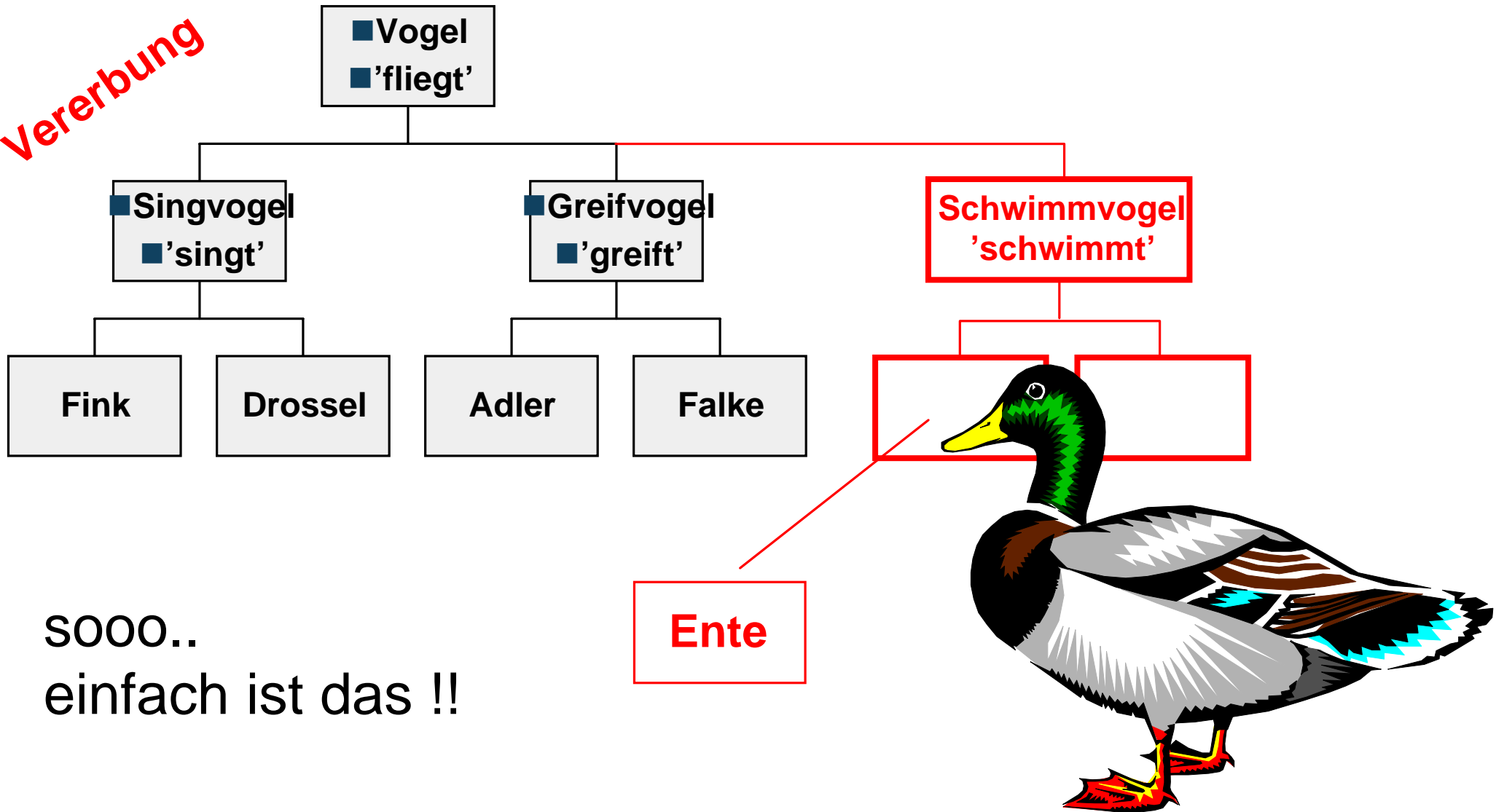
fliegen können sie alle

die einen **singen**,
die anderen **greifen**

Jedes Tier hat noch
besondere Eigenschaften

Ein 'kleines' OO-Beispiel

Vererbung

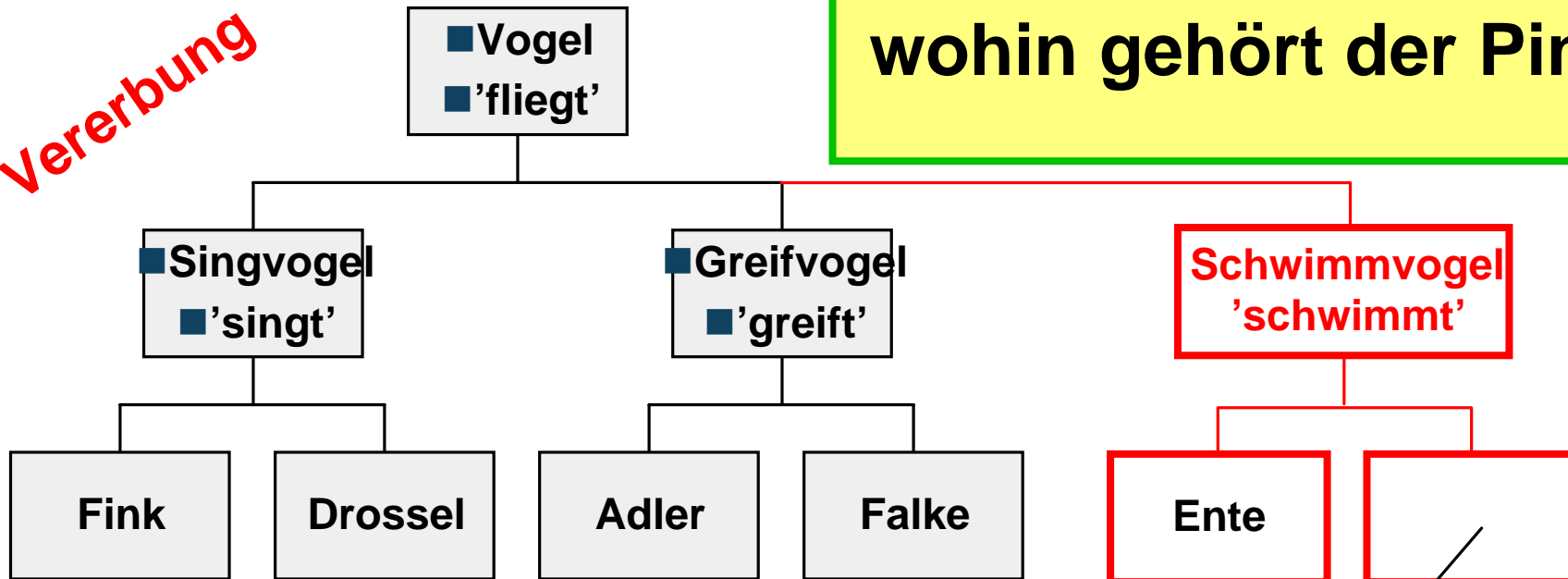


sooo..
einfach ist das !!

Ein 'kleines' OO-Beispiel

versuchen Sie es einmal
wohin gehört der Pinguin ??

Vererbung



... aber,
er kann
garnicht fliegen
und laufen können die anderen auch.



OO was ist das eigentlich?

- OO entspricht der natürlichen Wahrnehmung des Menschen
- Objekte entsprechen den real vorkommenden Gegenständen
- - Fahrrad, Auto, Mensch
- Diese Objekte können wiederum aus anderen Objekten bestehen
- - Schrauben, Räder, Hand, Fuß
- **Objekte haben Eigenschaften und ein Verhalten**
- - Eigenschaften werden als Attribute definiert (z.B.: Farbe)
- - Verhalten werden als Methoden definiert (z.B.: bremsen/fliegen)

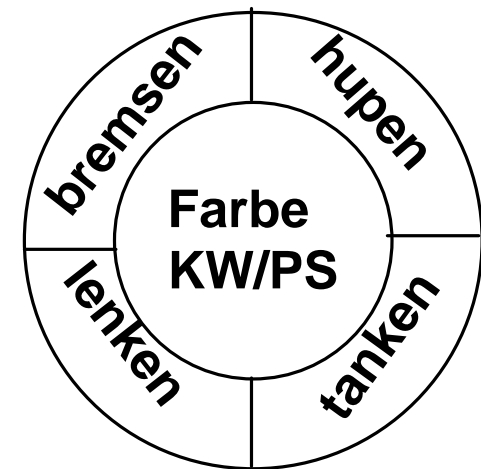


Daten und Funktionen werden in den Objekten zusammengehalten.

Objekt: Auto

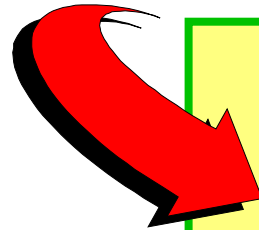


=



Theorie und Praxis

- Wir haben es n i c h t mit singenden Vögeln
- und hupenden Autos zu tun
- unsere Objekte bzw. Klassen sind:
 - Verträge
 - Schäden
 - Zahlungen
 - Ereignisse
 - Partner
 - Aktennotizen
 - Verletzungen
 - Grundstück
 - . . . etc.




Die Abstraktion
unserer 'Objektwelt'
stellt eine besondere
Herausforderung dar

Theorie und Praxis

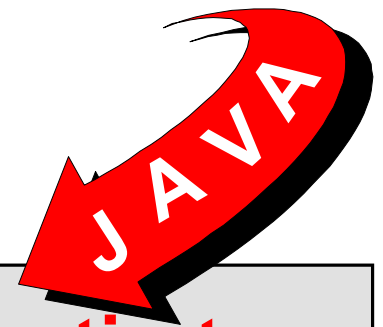
- Wir haben es n i c h t mit singenden Vögeln
- und hupenden Autos zu tun
- unsere Objekte bzw. Klassen sind:

- Verträ
- Schäd
- Zahlun
- Ereign
- Partne
- Aktenn
- Verletz
- Grund
- . . . etc.

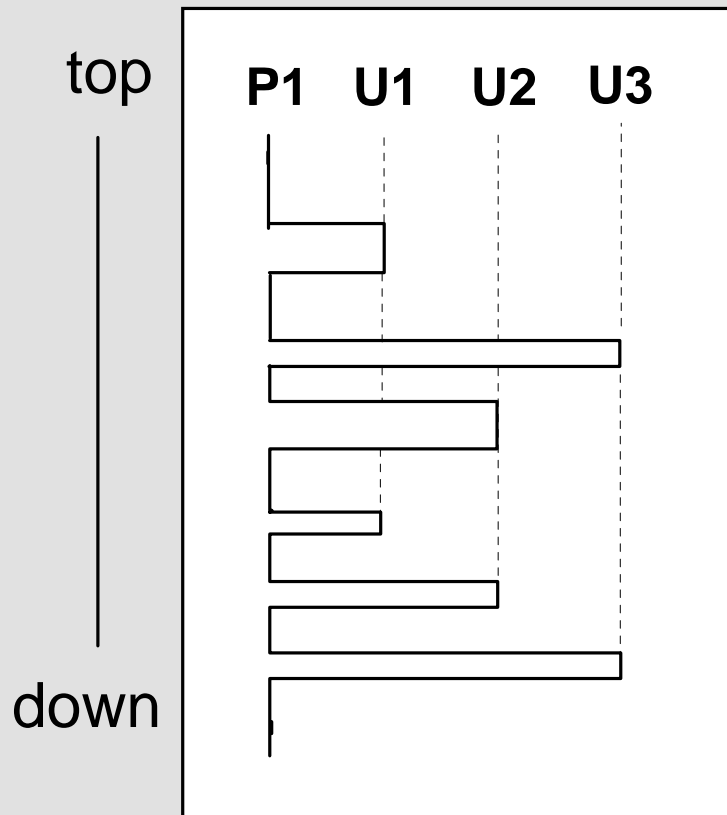


| | <u>real</u> | / | <u>abstrakt</u> |
|---------------|-------------|---|-----------------|
| Klassen | = Auto | / | Vertrag |
| Subklassen | = Räder | / | Tarif |
| Eigenschaften | = Farbe | / | Beitrag |
| Verhalten | = fahren | / | berechnen |

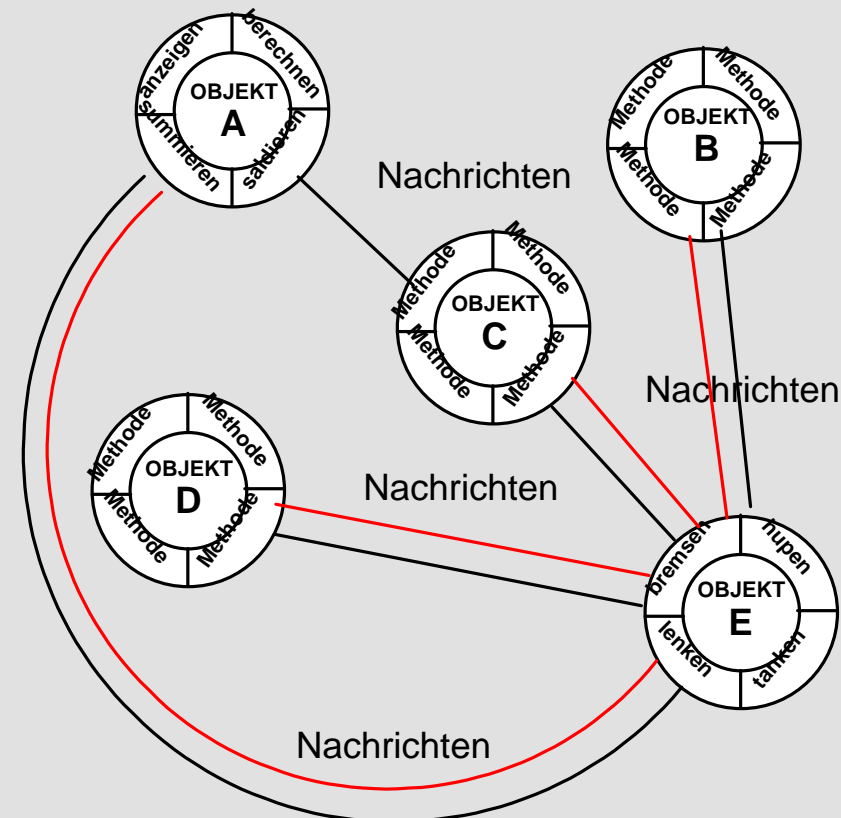
Anwendungsentwicklung



Programm-heute

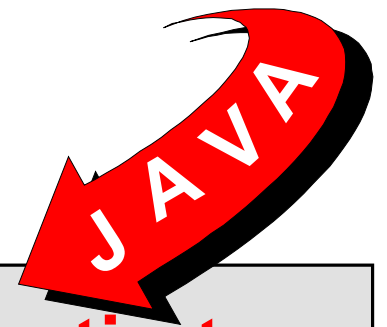


PG-objektorientiert



Darstellungsform der Objekte
'Doughnut-Darstellung'

Anwendungsentwicklung



Programm-heute

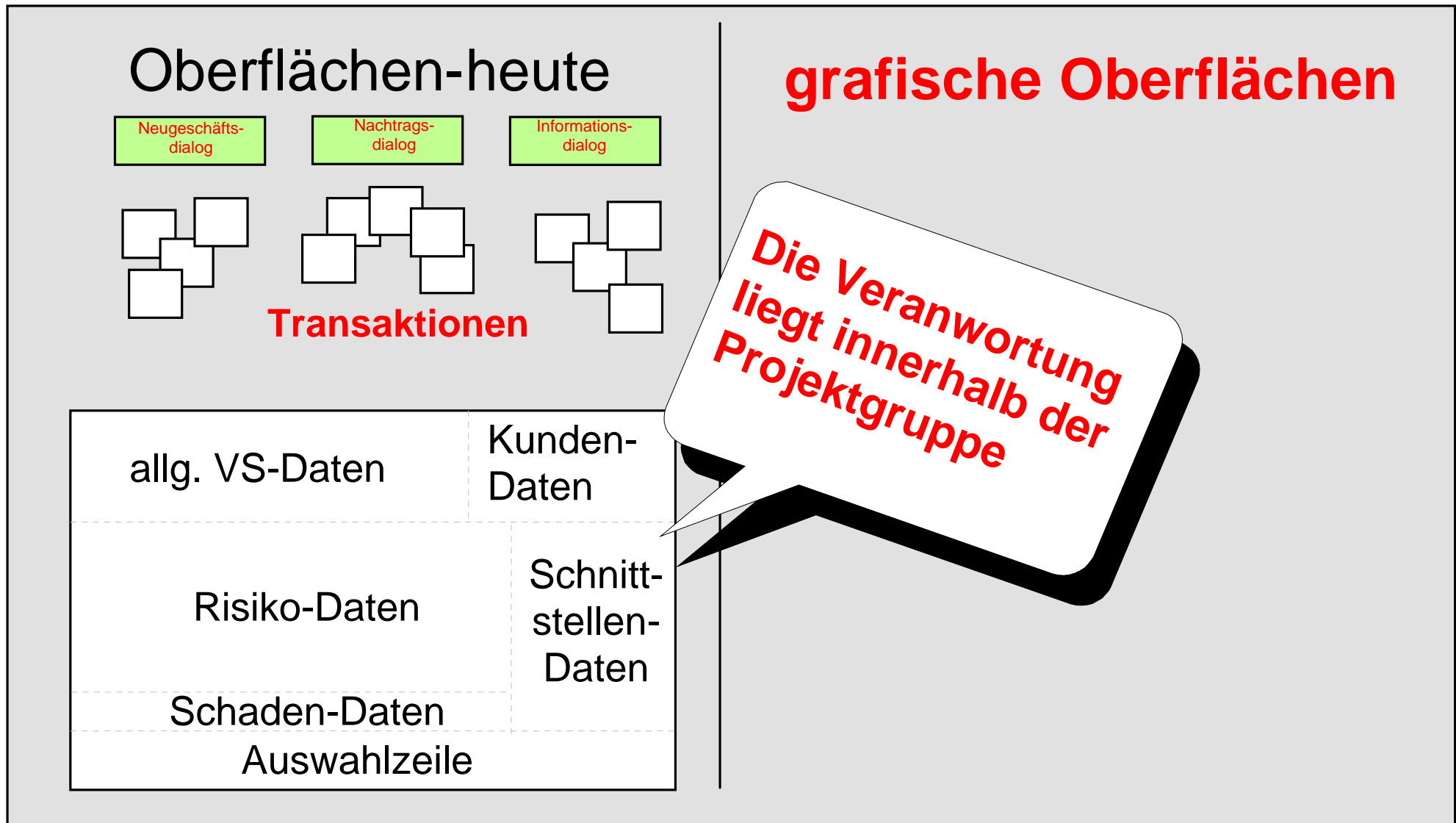
i.d.R: 'dicke Wälzer', nur von dem(r)jenigen interpretierbar, der(die) die Programme programmiert hat.

- > leicht verteilbar
- > Fehlerlokalisierung überschaubar
- > kaum wiederverwendbar
- > Daten/Funktionen nicht geschützt (gekapselt)

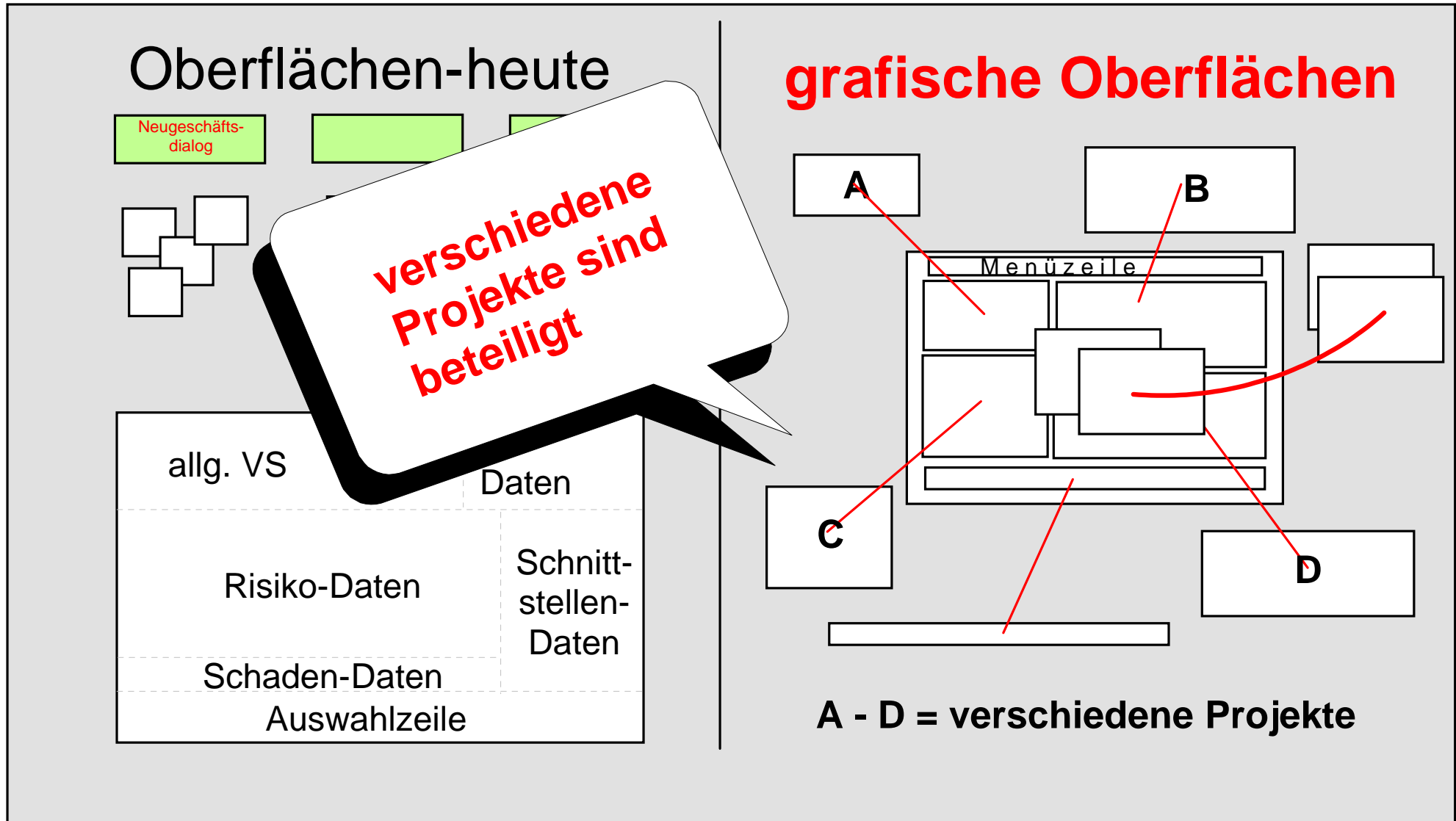
PG-objektorientiert

- > kleine Komponenten, aber dafür 'unendlich viele'
- > Daten/Funktionen sind geschützt (gekapselt)
- > höherer organisatorischer Aufwand (innerhalb und außerhalb des Projektes)
- > Wiederverwendbarkeit **wird die Zukunft zeigen**
- > Wartbarkeit / Fehlerlokalg. **wird die Zukunft zeigen**

grafische Benutzeroberfläche

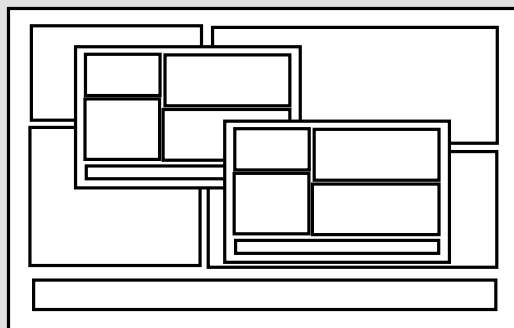
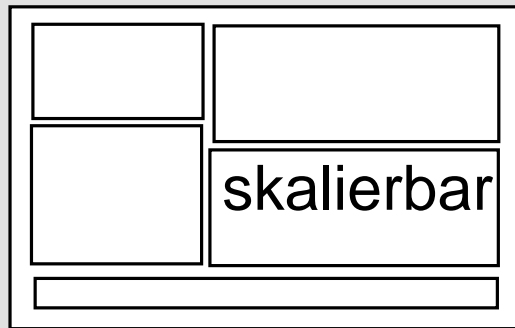


grafische Benutzeroberfläche

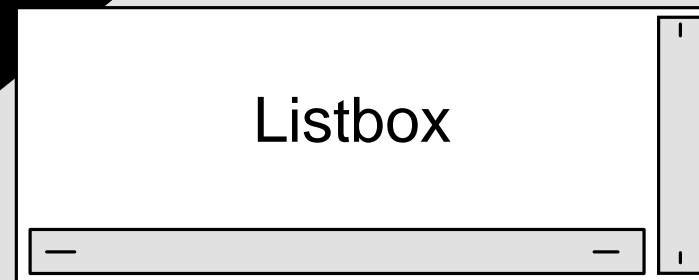


grafische Benutzeroberfläche

- mehr Möglichkeiten der Gestaltung
- parallele Fenster möglich
- Fenster wiederverwendbar
- Wiedererkennungswert für den Anw.
- deutlich höhere Komplexität
- erfordert mehr Abstimmungen
- höchste Anforderungen an Ergonomie
- Standard hat höheren Stellenwert



Fenstertechnik



Aktivitäten seit Projektstart

- Start wie in jedem anderen Projekt
 - Projektinitiierung
 - Istanalyse
 - . . . etc.

- fachliche Beschreibung des zukünftigen Schadenssystemes

Use-Cases



Aktivitäten seit Projektstart

Use-Case ist die Beschreibung eines Geschäftsvorfalles in Form von Interaktionen zwischen dem Anwender und dem System.

- **was beinhalten Use-Cases?**
- **wie beschreibt man Use-Cases?**
- **wie findet man Use-Cases?**
- **wie wird es weitergehen?**
 - Objektmodell ?
 - Klassenfindung ?
 - Vererbung ?
 - Java Programmierung
- **usw.**



Aktivitäten seit Projektstart

Use-Case ist die Beschreibung von Interaktionen zwischen

- was beinhaltet
- wie beschreibt
- wie findet man
- wie wird es verwendet
 - Objektmodell ?
 - Klassenfindung ?
 - Vererbung ?
 - Java Programm.
- usw.

Beratung !

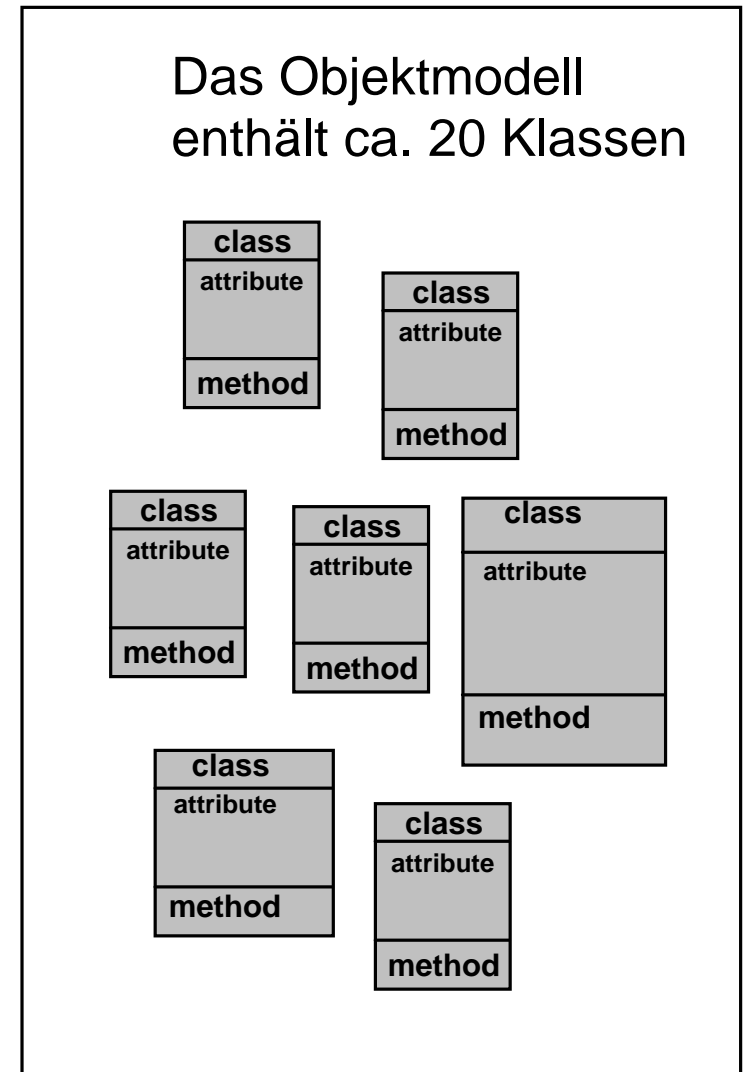
jemand, der etwas von Objekten und Klassen versteht



die ersten Schritte in die OO-Welt

Implementierung eines 'kleinen' Prototypen

- **Use-Case-Beschreibungen**
- **Objektmodell**
- **(Klassenstruktur eines Aufgabenbereiches)**
 - **identifizieren und klassifizieren**
 - **der Objekte**
 - **modellieren der hierarchischen**
 - **Beziehungen**
 - **modellieren der Beziehungen**
 - **zwischen den Objekten**
 - **festlegen der Verantwortlichkeit**
 - **der Klassen**
 - **(Klassen müssen etwas tun)**
 - **festlegen der Attribute der Klassen**
 - **(Kapselung der Daten)**

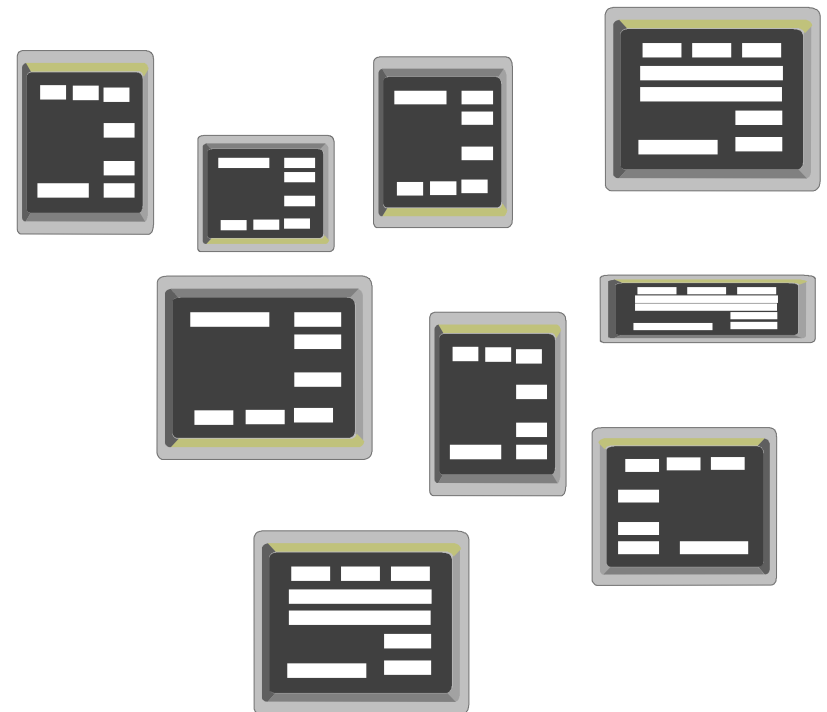


die ersten Schritte in die OO-Welt

Implementierung eines 'kleinen' Prototypen

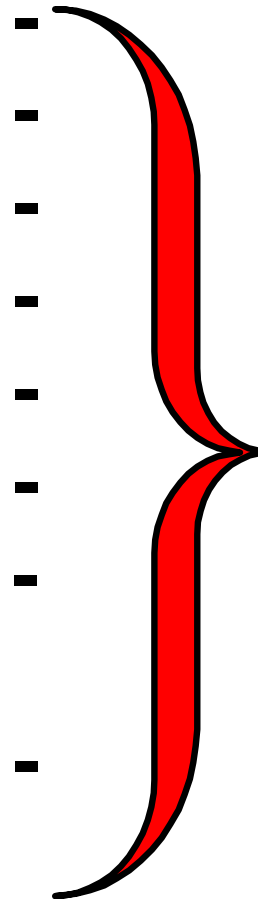
- **Oberflächengestaltung**
- **Implementierung der**
- **Benutzeroberflächen**
- **(Viewklassen)**
- **Implementierung des**
- **Objektmodells**
- **(Modell-Klassen)**
- **Zuordnung der Modell-Klassen**
- **zu den Oberflächen**
- **Implementierung der**
- **Steuerungslogik in den**
- **GUI-Klassen**

Die Benutzeroberfläche
enthält ca. 50 'Fenster'



Was nicht berücksichtigt wurde:

- San Francisco
- Swing-Klassen
- Style-Guide
- VA f Java 2.0
- Rational Rose
- Thin Client
- Transaktionskonzept
- Persistenz
- Schnittstellen
- ...



**bisher blieben
wir von den
'großen'
Heraus-
forderungen
'verschont'**

was hat sich tatsächlich geändert ?

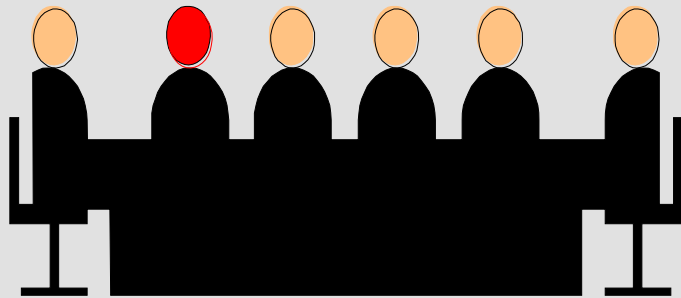


- objektorientierte Denkweise
- visuelles programmieren
- keine Programmlisten mehr
- iteratives entwickeln (frühzeitig echte Prototypen)
- neue Tools für Entwicklung und Dokumentation
- neue 'Welt' der Begriffe und Abkürzungen
- Englisch wird zum Standard
- Informationen über das Internet

auch für die 'nächste Zeit' kommen wir
nicht ohne Beratung aus

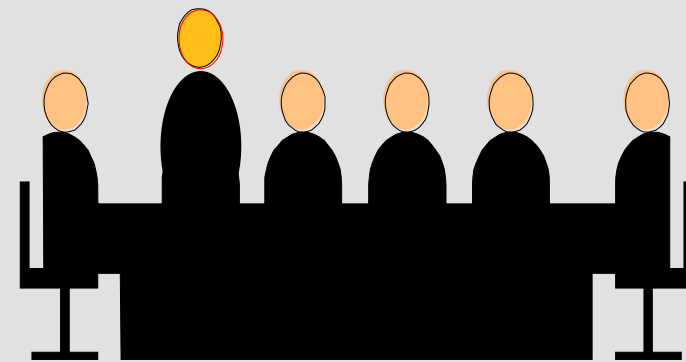
... anno April 1998

wovon reden
die eigentlich?



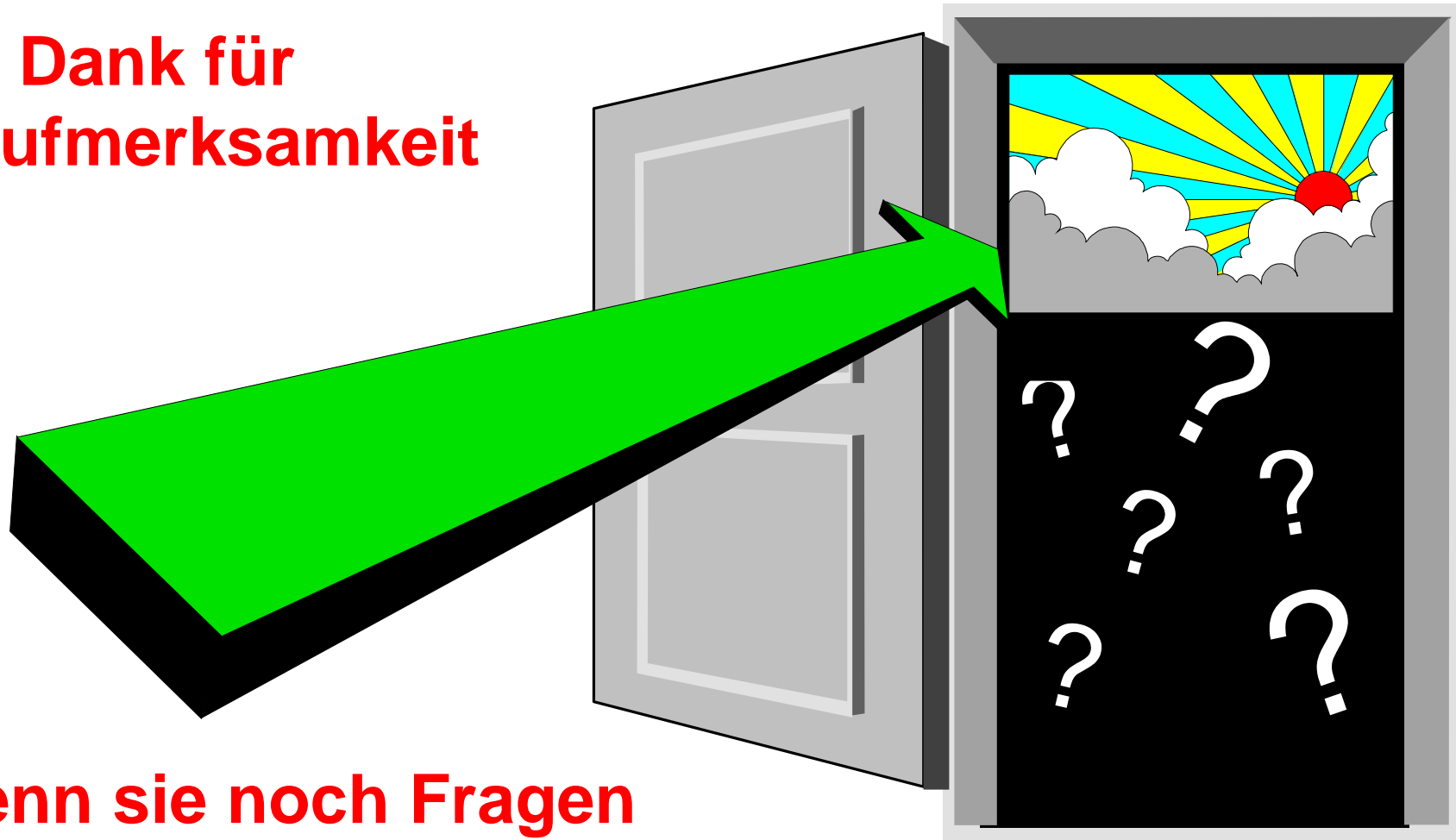
... heute März 1999

OO-Welt ist ...
packen wir es
an ...



OO-Welt - quo vadis ??

**vielen Dank für
Ihre Aufmerksamkeit**



**wenn sie noch Fragen
haben ?**