Kooperierende Objektsysteme: Betrieb und Entwicklung

Gerald Schröder

Technische Universität Hamburg-Harburg

8. April 1999

Gliederung

1. Aufgabenstellung
2. Objektsysteme
3. Kooperierende Objektsysteme
4. Zusammenfassung

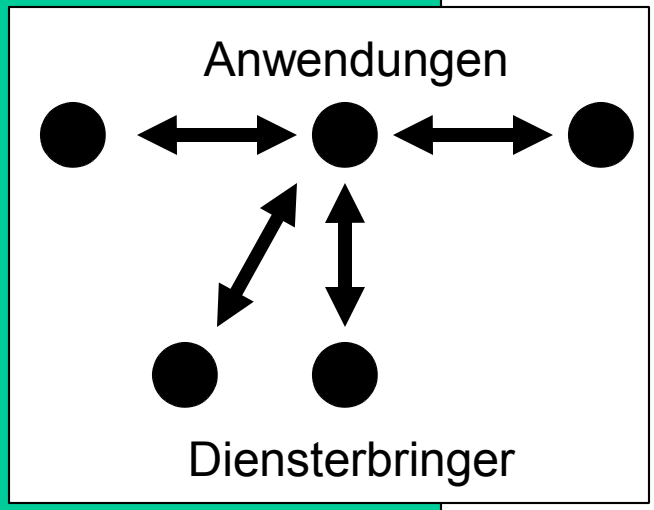
1. Aufgabenstellung: Übergreifende Nutzung autonomer Softwaresysteme

Beispiele:

- Eigenständige Anwendungen: Word und Excel, ...
- Anwendungen und Diensterbringer: SAP R/3 und Oracle, ...
- Datenverteilung: WWW Browser und WWW Server, ...
- Lastverteilung: Workstations und Compute-Server, ...

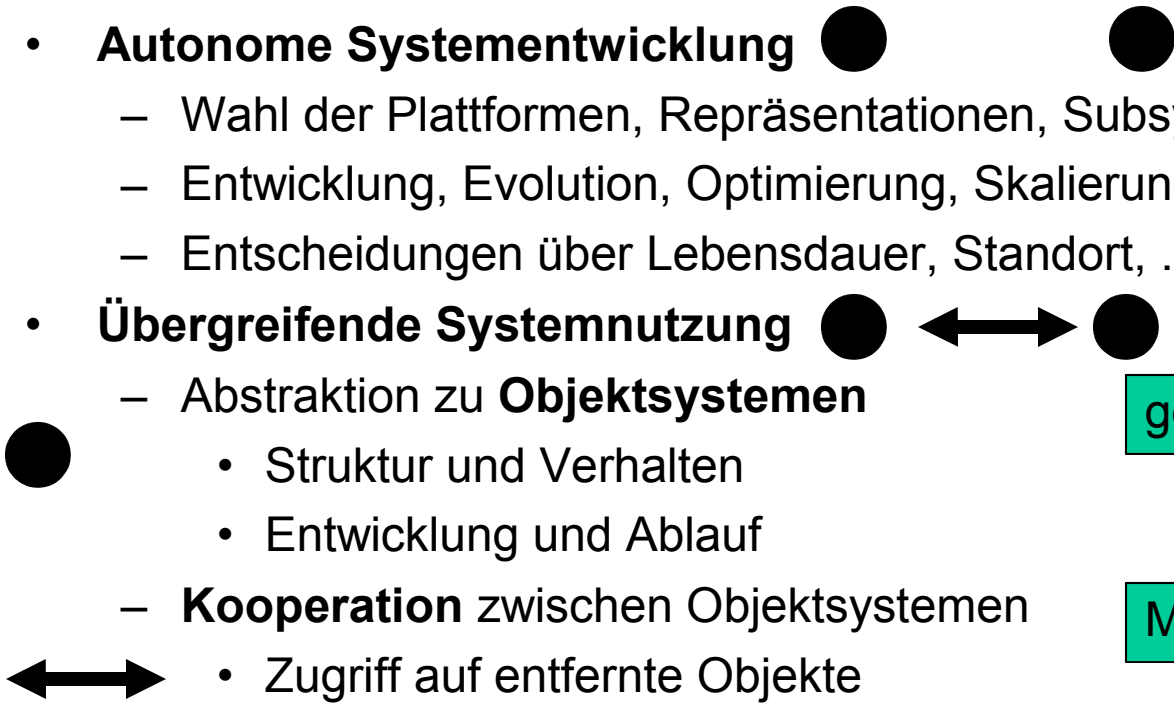
Eigenschaften

- eigenständige Anwendungen oder generische Diensterbringer
  - unabhängig voneinander entwickelt
  - getrennt weiterentwickelt
  - aufgabenabhängig spezialisiert
- miteinander kooperierend
  - zeitlich begrenzt
  - wechselnd
  - lokal oder entfernt



Nutzung und Entwicklung

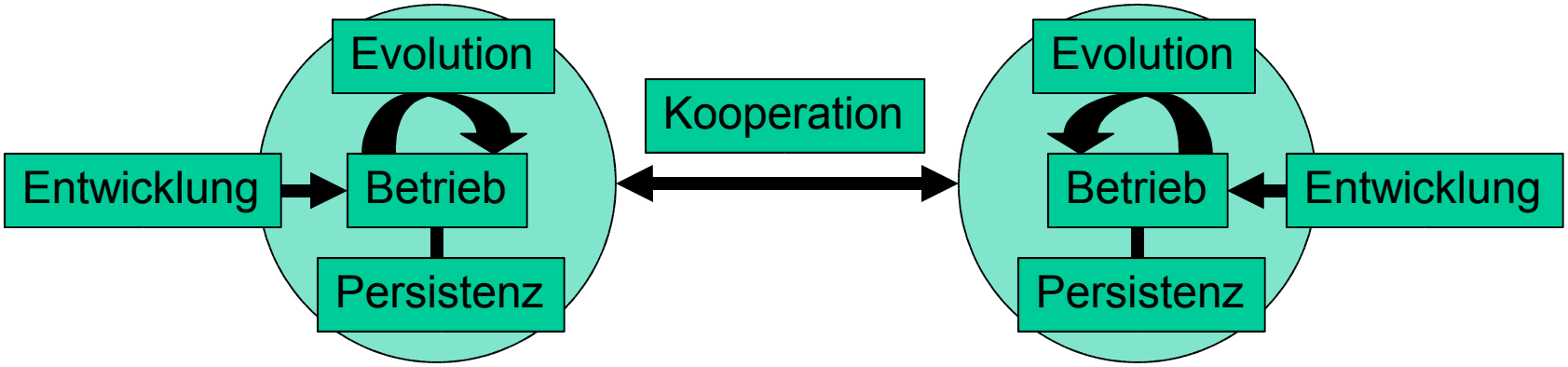
- **Autonome Systementwicklung**
  - Wahl der Plattformen, Repräsentationen, Subsysteme, ...
  - Entwicklung, Evolution, Optimierung, Skalierung, ...
  - Entscheidungen über Lebensdauer, Standort, ...
- **Übergreifende Systemnutzung**
  - Abstraktion zu **Objektsystemen**
    - Struktur und Verhalten
    - Entwicklung und Ablauf
  - **Kooperation** zwischen Objektsystemen
    - Zugriff auf entfernte Objekte
    - Migration von Objekten



Der **Nutzen** wird nur realisiert, wenn die **Ablauffähigkeit** der kooperierenden Objektsysteme erhalten bleibt.

Zielsetzung

Konsolidierung der Anforderungen aus **Autonomie** und **Kooperation** von Objektsystemen



Aufgabenstellung:

- **Konzept**entwicklung für autonome, kooperierende Objektsysteme
- Entwurf eines übergreifenden **Systemmodells** für Betrieb, Entwicklung, Kooperation, Evolution, Persistenz, ...
- **Realisierung** mit innovativer Softwaresystemtechnik

Schwerpunkte:

- Sicherung der **Ablauffähigkeit**
- **Phasenübergreifende** Nutzung von Entwicklungsinformationen

Kontext

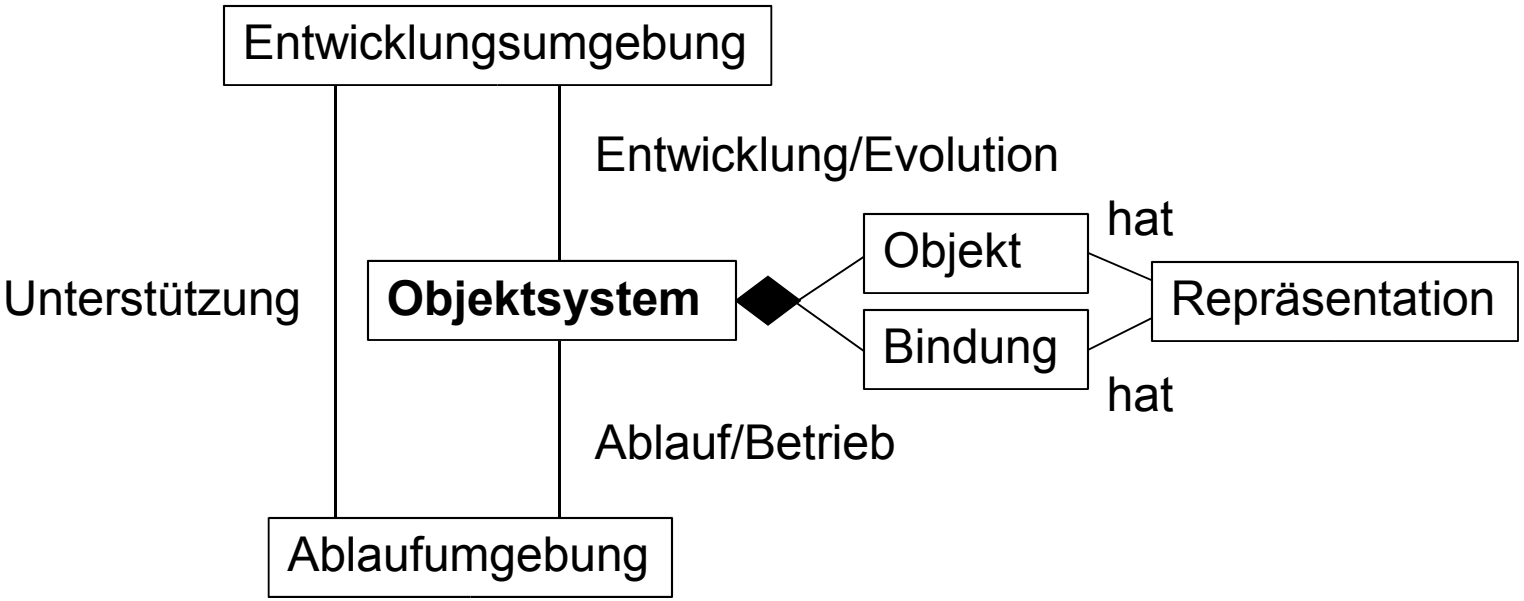
Beiträge des **Tycoon***-Projekts (Dissertationen am AB Softwaresysteme)

- Integration heterogener Systeme mit polymorpher **Typisierung**
- **Generierung** von Objektsystemen
- Orthogonale **Persistenz** von Daten, Funktionen und Prozessen
- Orthogonale **Mobilität** von Daten, Funktionen und Prozessen
- **Nutzungssicherheit** in offenen Umgebungen

Beitrag dieser Arbeit:
Verbesserung der **Ablauffähigkeit** von kooperierenden Objektsystemen durch Verzahnung von **Betrieb** und **Entwicklung**

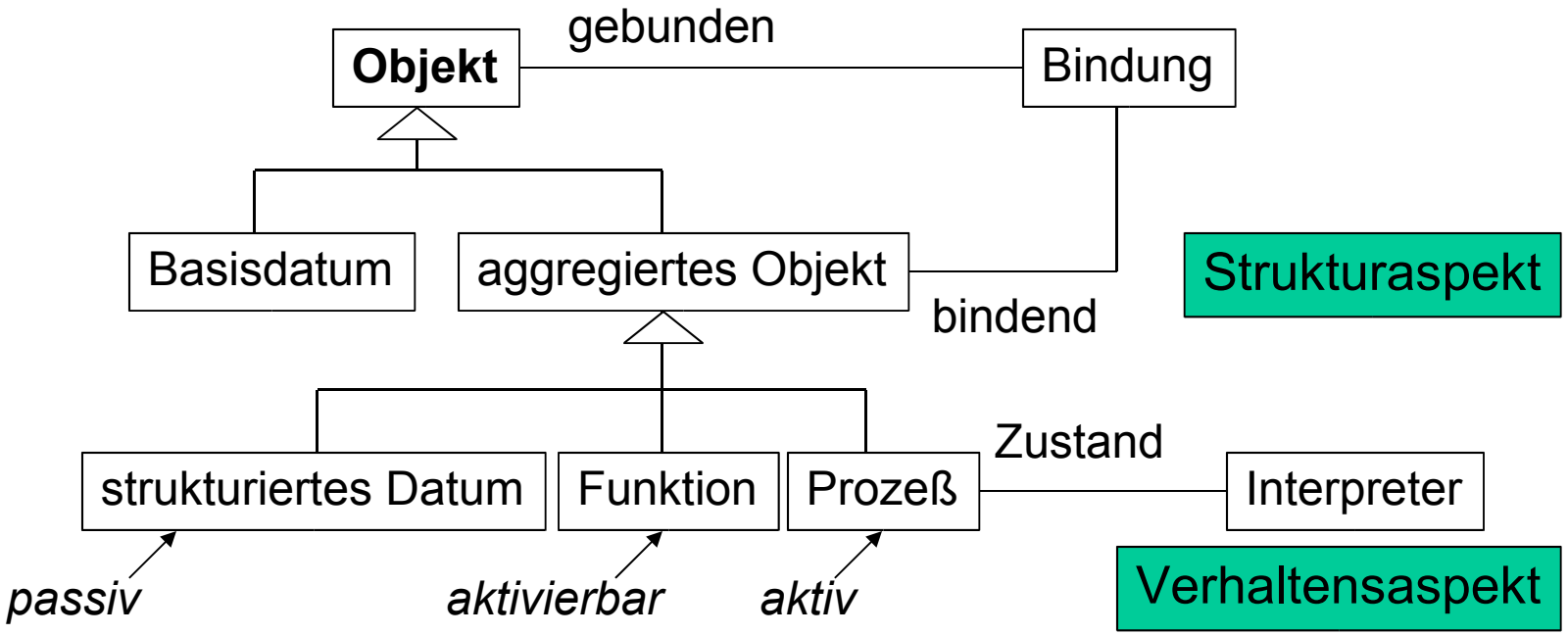
* Tycoon = Typed Communicating Objects in Open Environments

2. Abstraktion: Objektsysteme



- Objekte und Bindungen
- Repräsentationen
- Entwicklungs- und Ablaufumgebung

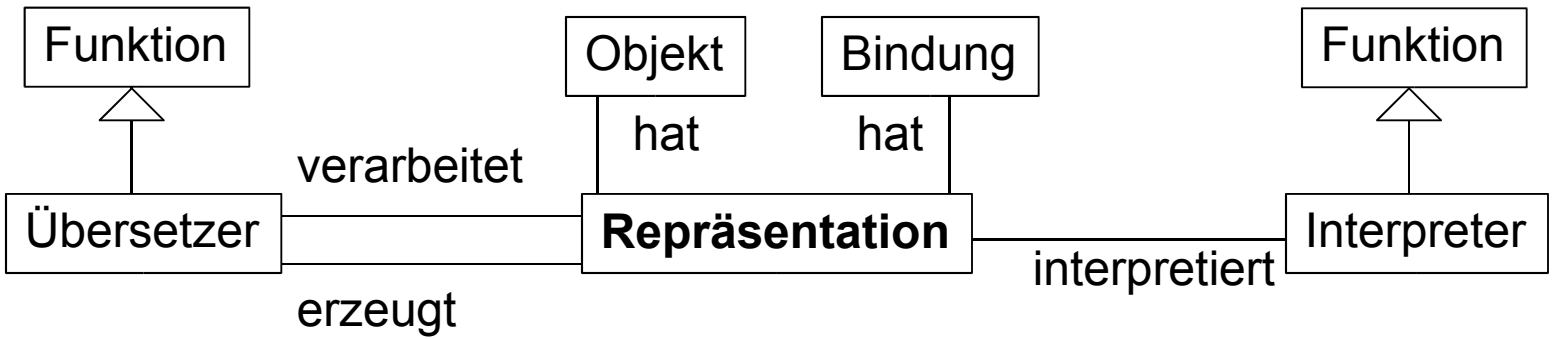
Objektsysteme: Objekte & Bindungen



Bindungen

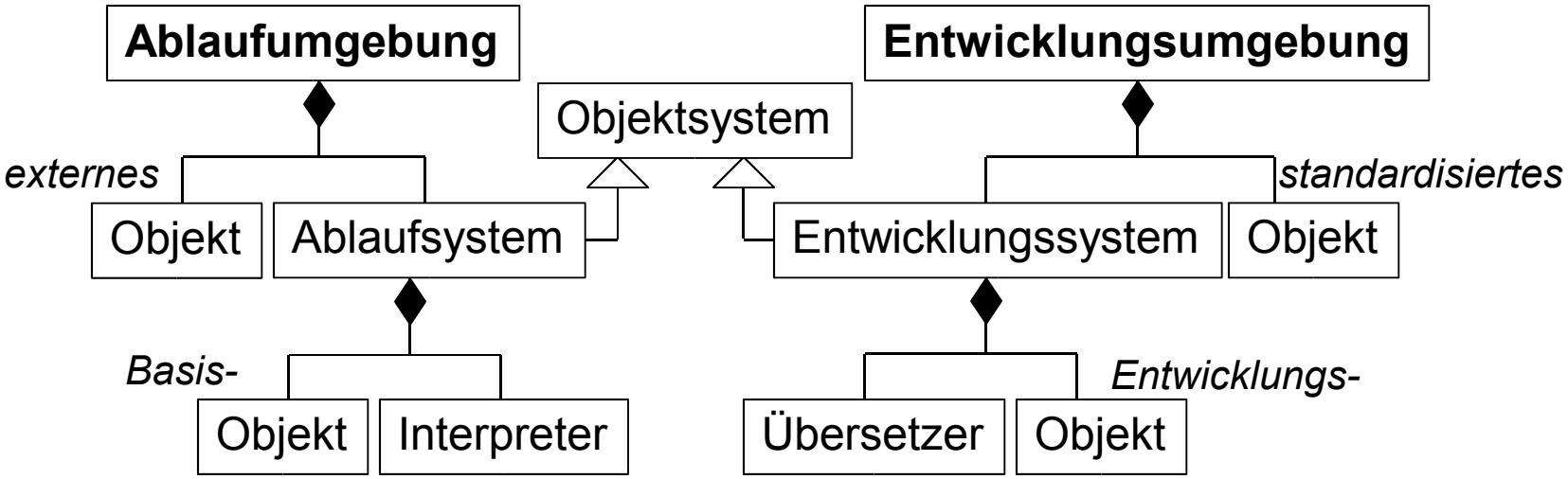
- bestehen **zwischen** Objekten,
  u.U. über verschiedene Objektsysteme hinweg
- weisen eine **Lebensdauer** auf

Objektsysteme: Repräsentation



Entwicklung:	anwendungsnahe Rep.	Compiler	maschinennahe Rep.
Persistenz:	transiente Rep.	Datenbank	persistente Rep.
Kooperation:	proprietäre Rep.	Konverter	portable Rep.

Objektsysteme: Ablauf- und Entwicklungsumgebung

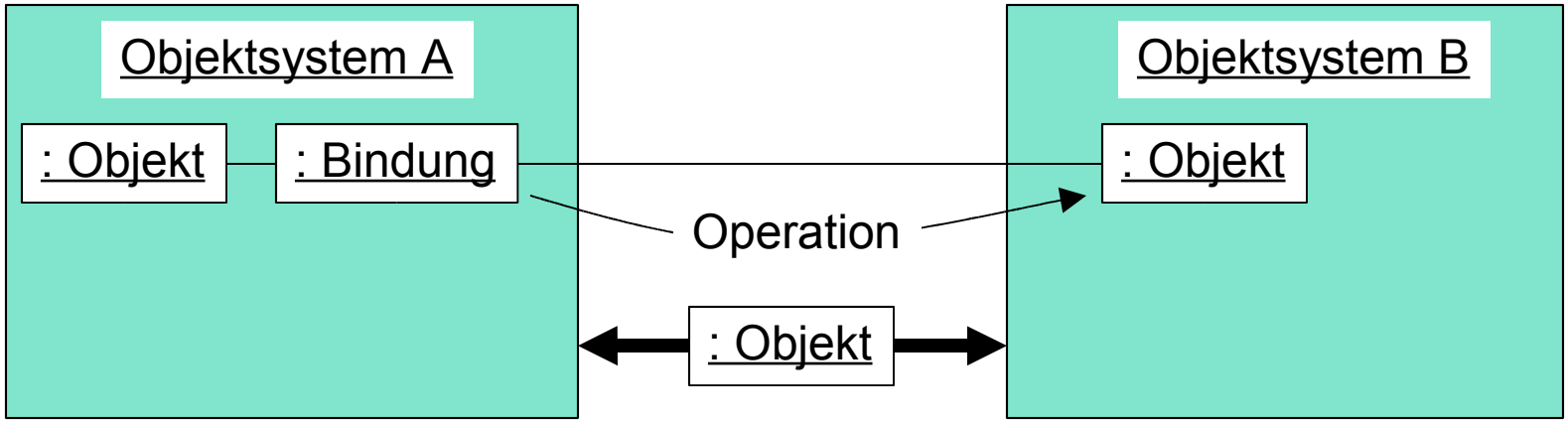


Kooperation **durch** Objekte der Ablauf- und Entwicklungsumgebungen

Verhaltensreflektion

linguistische Reflektion

3. Kooperierende Objektsysteme



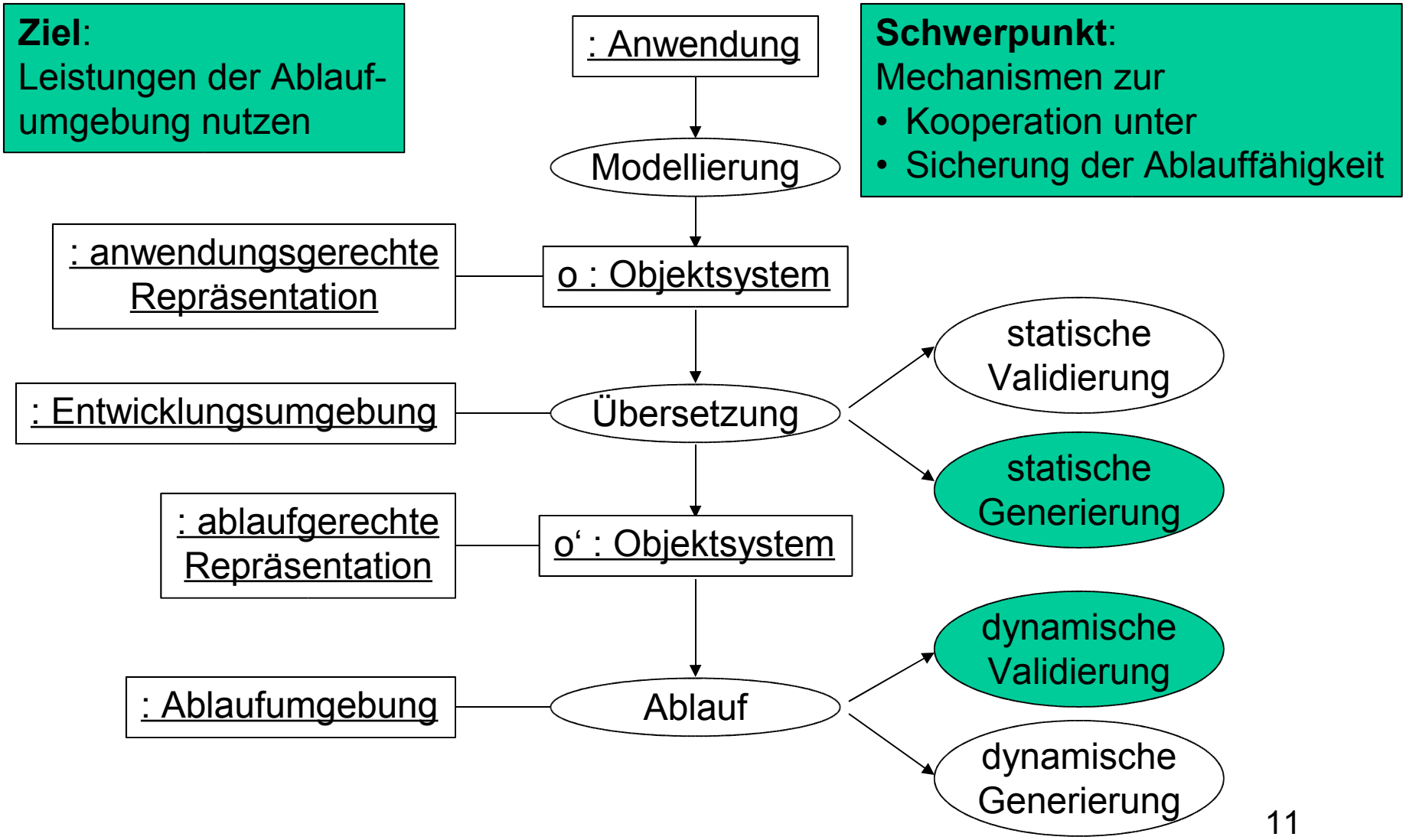
Autonome Objektsysteme gehen auf Zeit Nutzungsbeziehungen ein durch

- **Bindungen** an **entfernte** Objekte
- Auslösen von **Operationen** auf **entfernten** Objekten
- **Migration** von Objekten und Bindungen
  - **Repräsentationsumwandlung**
  - **Neubindung**

Lebensdauerkontrolle

dynamisch durch Funktionen

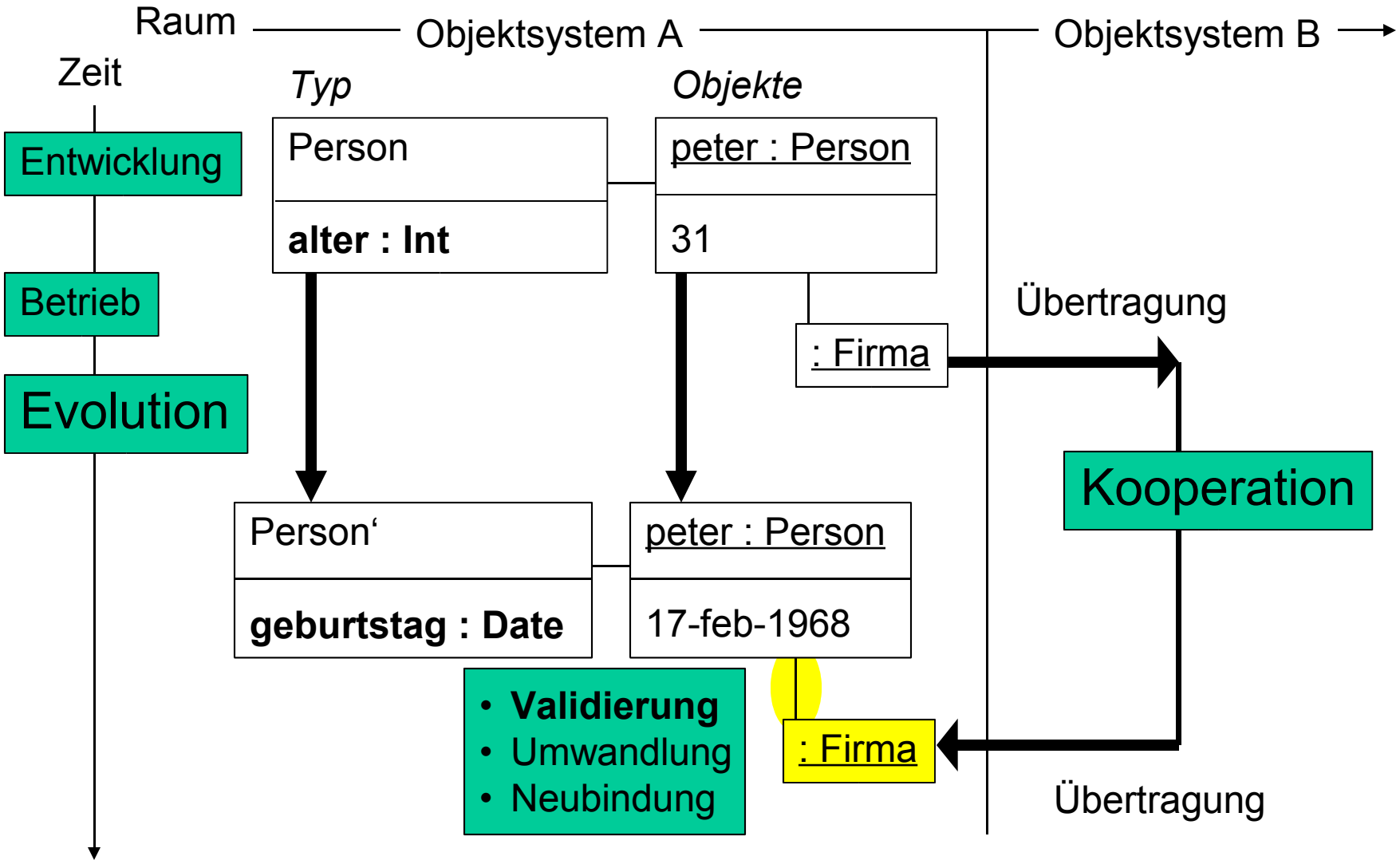
Entwicklung und Betrieb



Generierung und Validierung

	Validierung	**Generierung**
statisch (Entwicklung)	Typsystem & Typprüfer	**Entwurfsmuster** • Definition • reflektive Einbindung in Entwicklungssysteme
dynamisch (Ablauf)	**Dynamische Typen** • Einbindung von Typen & Typprüfer in das Objekt-system (Reflektion) • Übertragung von Typen in heterogenen Systemen • Auswirkungen von Evolution	• Funktionen höherer Ordnung • Übersetzung zur Laufzeit (Reflektion)

Dynamische Validierung



Statische Generierung von Entwurfsmustern

```
migrate to gruppenleiter with remote budgetDB : BudgetDB do
  ueberpruefe(budgetDB abrechnung.summe)
end
```

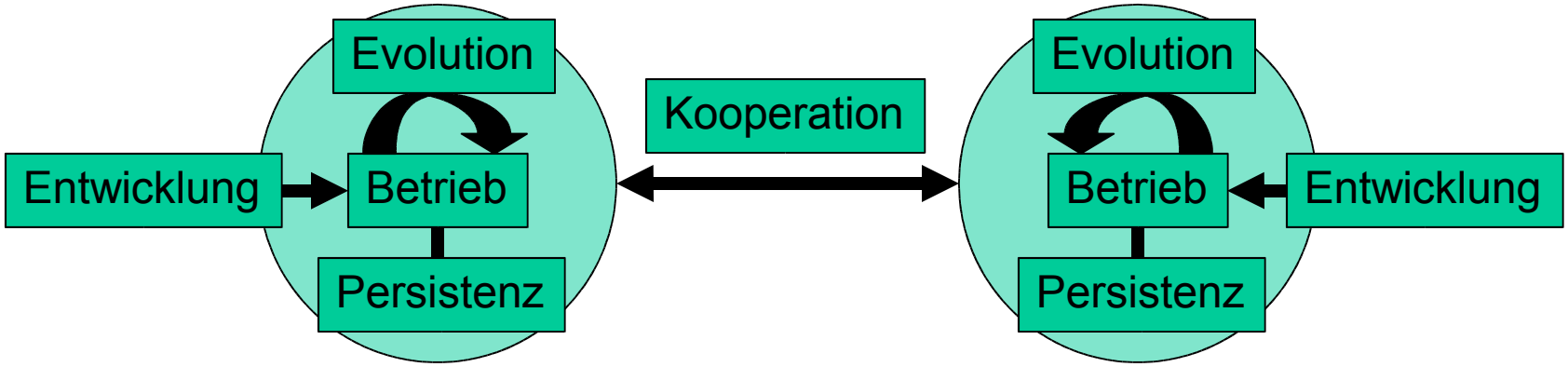
gewünschte anwendungsgerechte Repräsentation

Generierungsregel für Entwurfsmuster „Migration“ mit **Lebensdauerkontrolle** für entfernte Bindungen

```
migrateG :== „migrate“ „to“ destination = valueG
  „with“ „remote“ localId = ideG „:“ type = typeG
  „do“ sequence = valueG „end“
=>
let f(localID : type) = begin sequence end
 mark(f)
gate.migrateTo(:type destination f)
unmark(f)
```

```
let f(budgetDB : BudgetDB) =  begin ueberpruefe(budgetDB abrechnung.summe) end
mark(f)
gate.migrateTo(:BudgetDB gruppenleiter f)
unmark(f)
```

4. Zusammenfassung



Beiträge:

- **Modell** für kooperierende Objektsysteme mit **dynamischen Bindungen**, **heterogenen Repräsentationen**, **integrierten Ablauf**- und **Entwicklungsumgebungen**
- Sicherung der **Ablauffähigkeit** kooperierender Objektsysteme durch **phasenübergreifende** Nutzung von Entwicklungsinformationen
  - **Lebensdauerkontrolle entfernter Bindungen**
  - **Validierung, Umwandlung** und **Neubindung** bei **Migration.**