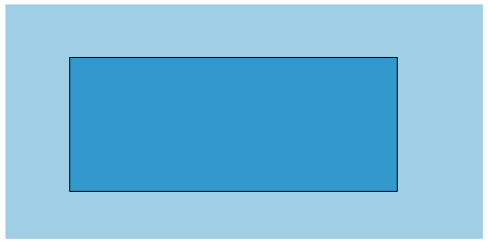
Agenten als Strukturen verteilter Systeme

Entwicklung der Agenten in den VS

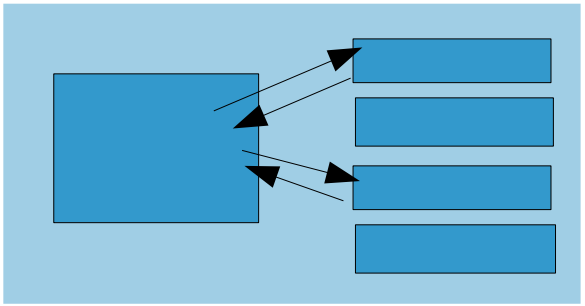
- Agenten in den verteilten Systemen
  - geringer Fokus auf Intelligenz
  - Effizienz, Betriebssystem-Komponenten
  - neue Betriebssystem-Strukturen?
- Agenten als erwartungsgemäße Weiterentwicklung

Entwicklung der Systeme

- Monolithische, lokale Systeme
  - älteste Form, einfache Unterprogramme
  - globale Variablen

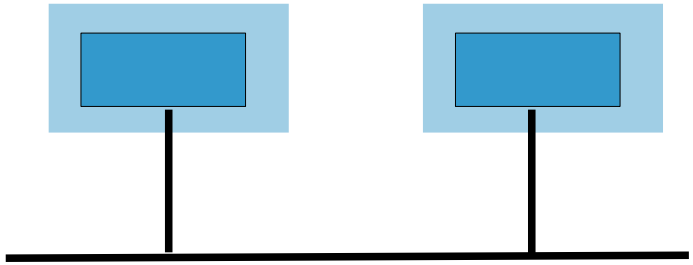


- Lokale prozedurale Programmierung
  - Programmelemente, die von verschiedenen Stellen aus aufgerufen werden können
  - Konzept der Parameterübergabe



Entwicklung der Systeme

- Verbindung über Netze
  - Kommunikation von Programmen über Netzwerke
  - Programmierschnittstelle *Sockets*
  - Zunächst keine spezifische Struktur, teilweise einfache Byte-orientierte Kommunikation



Beispiel Java

- Socketverwendung in Java

```
Socket sockConn = null;
try {
        sockConn = new Socket(ipServer, 1024);
        OutputStream os = sockConn.getOutputStream();
        InputStream is = sockConn.getInputStream();
}
...
```

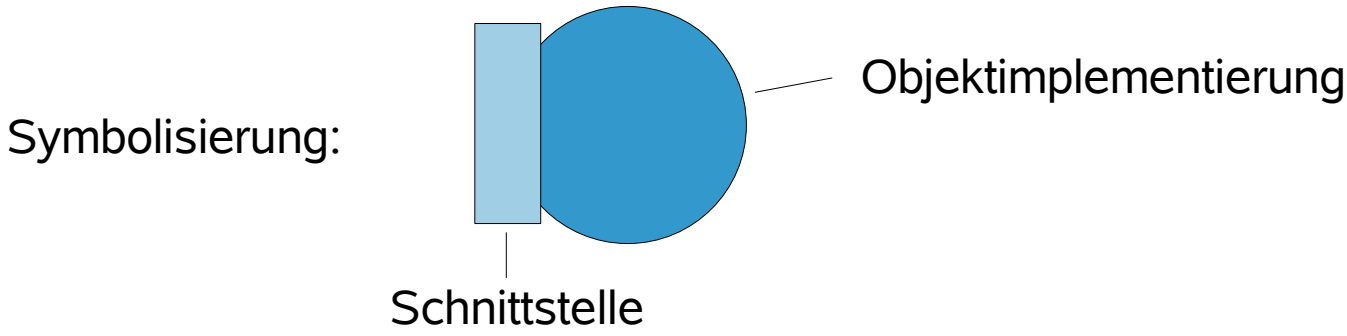
```
ServerSocket ssock = new ServerSocket(1024);
Socket sockClientConn = null;
try {
        sockClientConn = ssock.accept();
}
...
```

- Von Bytekommunikation zur Datenkommunikation
  - Java bietet spezielle Datenströme an

```
os.write((byte)0x40);
DataOutputStream dos = new DataOutputStream(os);
dos.writeInt(1000);
```

Objekte

- Charakterisierung nach G. Booch (1991)
  - Zustand
    - Gesamtheit der diesem Objekt zugeordneten Speicherzellen
    - Beeinflusst das Verhalten
  - Identität
    - Ein Objekt besitzt eine eindeutige, unverwechselbare Identität, die es von anderen unterscheidbar macht.
  - Verhalten
    - Reaktionen auf Veränderungen in der Umgebung



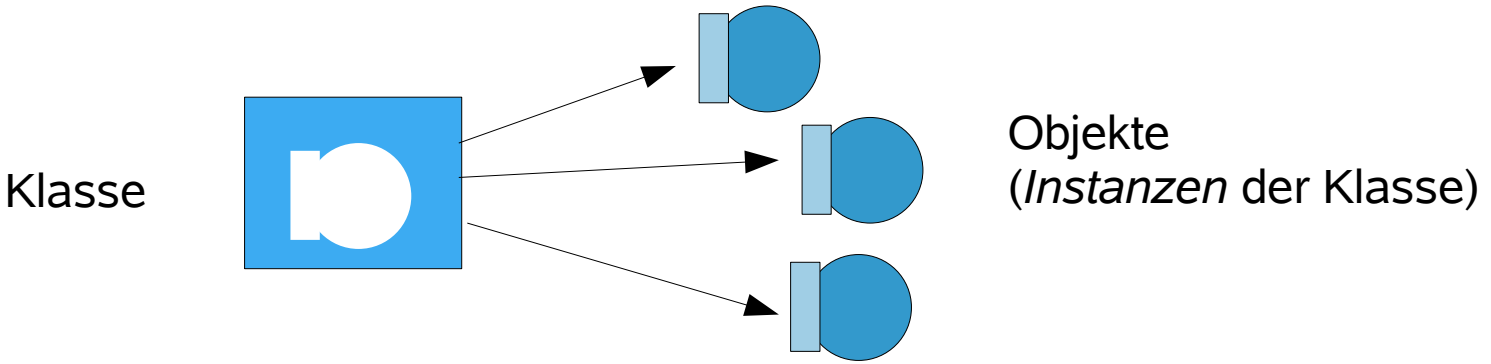
Besonderheiten

Vereinfachung der Wartung!

- Verkapselung
  - Objektverhalten und Zustand ist von außen nicht sichtbar
  - Zugriff auf Objekt nur über Schnittstelle
- Vererbung (inheritance = „Beerbung“)
  - Verhalten/Schnittstelle definiert eine Gruppe von Objekten (Klasse)
  - Spezialisierung/Erweiterung definiert Untergruppe davon
  - Unterklassen definieren abgeleitete Objekte, die stets an Stelle der ursprünglich erwarteten Objekte treten dürfen

Klassen

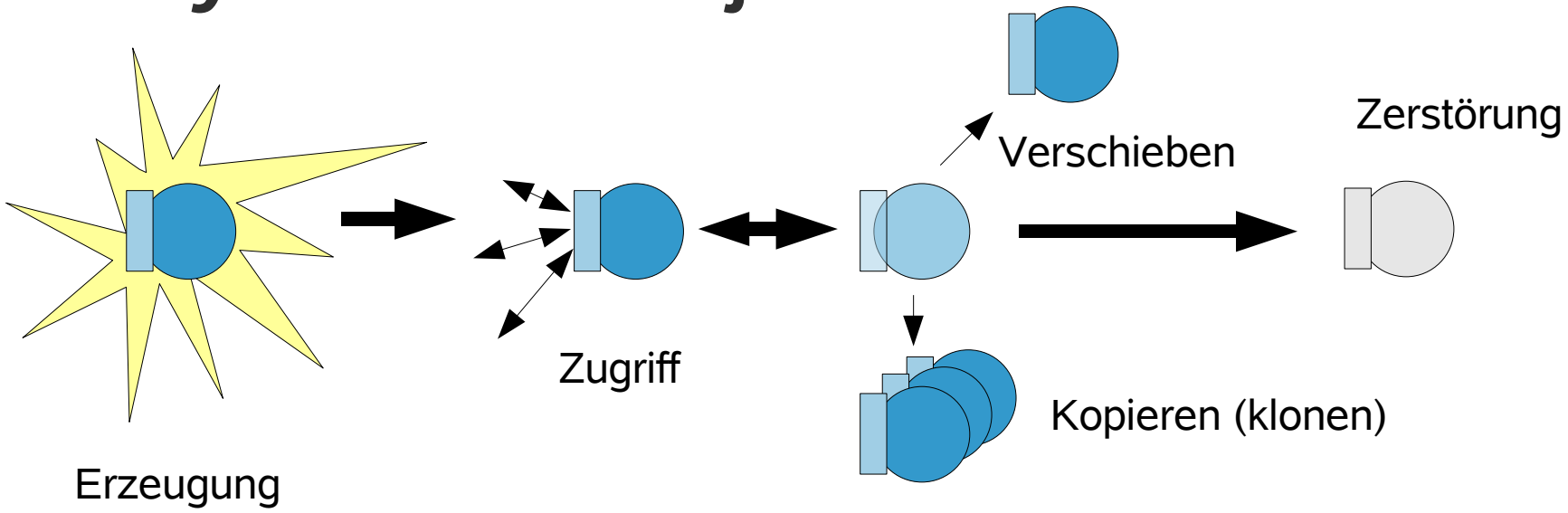
- Schablonen, um Objekte zu erzeugen
- Definieren das Verhalten und die Schnittstelle
- Definieren weder Zustand noch Identität des Objekts
- Klassenobjekte
  - Modellierung von Klassen als spezielle Objekte (alles sind Objekte)
  - als Factorys (Objekterzeuger)
  - siehe Java:
    ```
    public class java.lang.Class extends java.lang.Object
    ```



Zugriff auf Objekte

- Schnittstelle als einziger Zugriffsweg
  - definiert Methoden (Prozeduren des Objekts)
  - definiert Felder (Objektvariablen, Instanzvariablen, „Member variables“)
  - strikter: Felder müssen durch „Getter“- und „Setter“-Methoden angesprochen werden
    - mithin nur Methoden in der Schnittstelle
    - bessere Kontrolle des Zugriffs (z.B. nur lesend)
- Referenz (Objektreferenz)
  - „Abbild“ des Objekts
  - ermöglicht den Zugriff auf das Objekt z.B. durch Methodenaufruf

Lebenszyklus eines Objekts



Erzeugung: *Instanziierung* von Klassen

Zugriff: Aufruf von Methoden, Übergeben von Parametern

Verschieben: Verlagerung des Objekts zwischen Lokationen

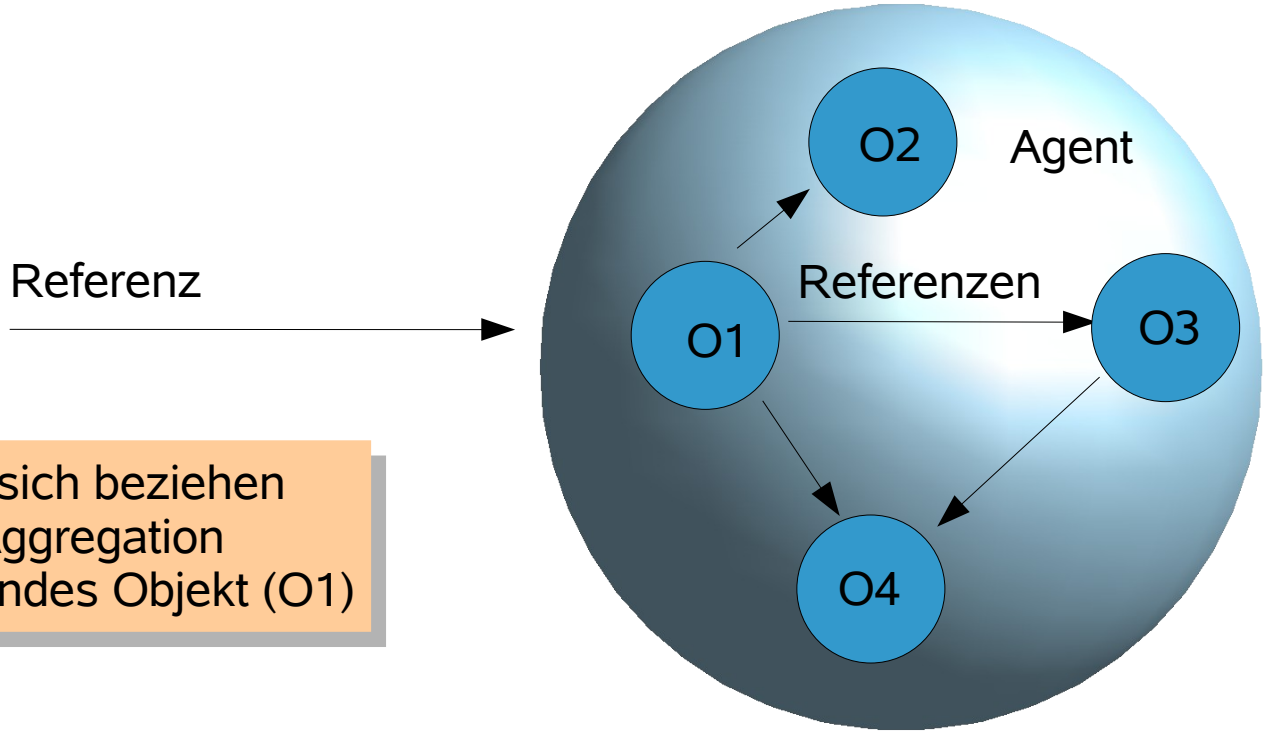
Zerstören: Freigabe der zugeordneten Ressourcen

Umgang mit Referenzen

- Unterscheidung
  - Referenz im Programmiersprachensinne: erlaubt Prüfung durch Compiler; direkte Repräsentation in Programmiersprache
  - allgemeine Referenz: Behandlung nur auf Programmebene
- Erzeugung liefert Referenz auf Objekt
  - Erzeuger hat immer Zugriff auf das Objekt (Sicherheit?)
- Klonen des Objekts
  - Identität nur immer einem Objekt zugeordnet, also neue Identitäten
  - Zugriff durch klonendes Objekt
- Zugriff über Referenzen
  - Zugriff ist aber abhängig von der Verteilung (lokal / entfernt)!
  - Verschieben macht Referenzen ungültig

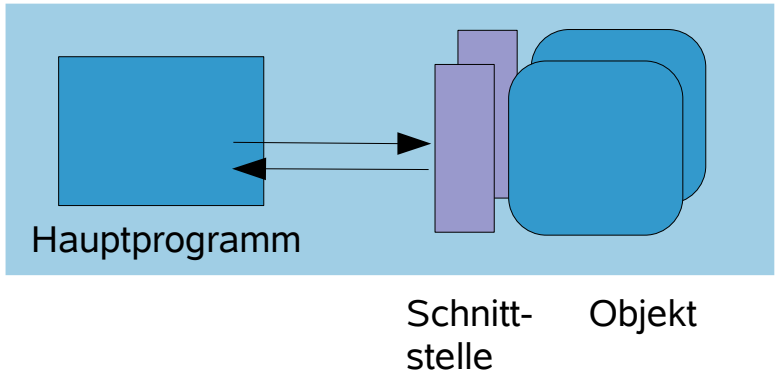
Aggregationen

- Identität betrifft Objekt
- Agent kann aus mehreren Objekten bestehen
- Agent bildet eine Einheit dieser Objekte

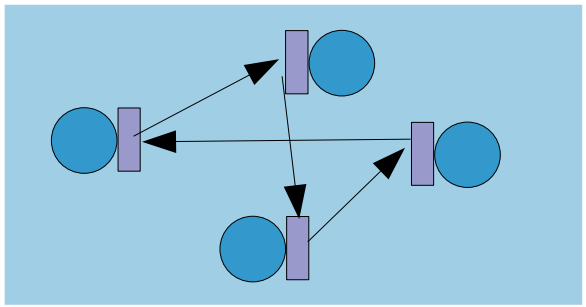


Die Referenz kann sich beziehen
- auf die gesamte Aggregation
- auf ein aufspannendes Objekt (O1)

Entwicklung der Systeme



- Objektbasierung
  - Modellierung von Datenstrukturen als eingekapselte Strukturen
  - Schnittstellen als *Funktionszusicherung*



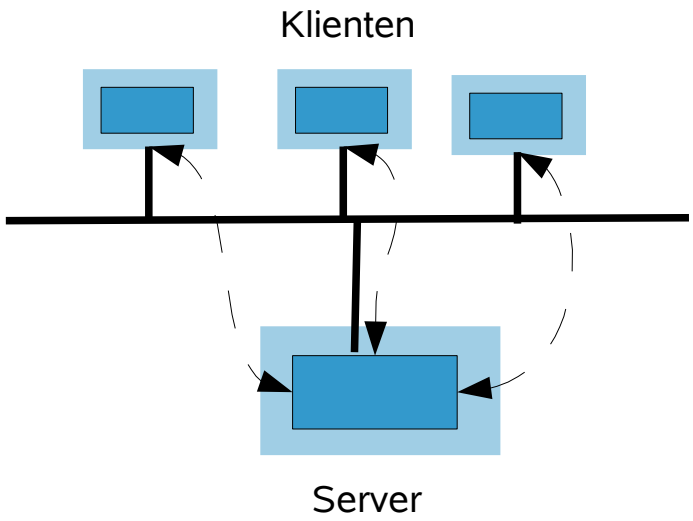
- Objektorientierung
  - Modellierung von Programmen als Menge miteinander kommunizierender Strukturen
  - Verwendung von Vererbung

Objektorientierte Sprachen

- Smalltalk
  - Konsequente Objektorientierung, geringe Verbreitung
- C++
  - Erweiterung von C, jedoch schwere Erbschaft („halb“ objektorientiert)
  - Objective C als „konsequentere“ Umsetzung
- Modula-2
- Eiffel
  - Neuentwurf; kein Aufbau auf früherer Sprache
  - „Design by Contract“: Vor- und Nachbedingungen
- [incr Tcl]
  - Erweiterung der Skriptsprache Tcl (*Tool command language*)
- Java
- C#

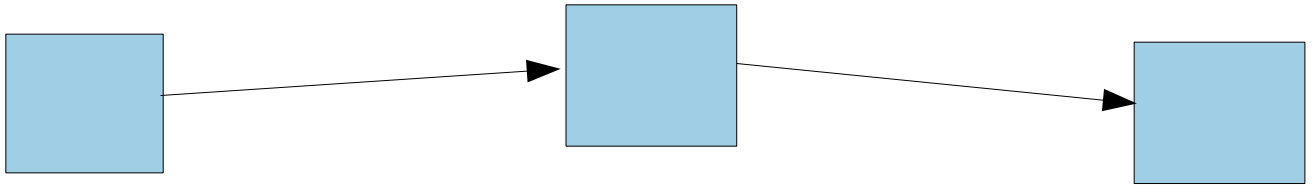
Entwicklung der Systeme

- Klient-Server-System
  - Kommunikationsrollen
    - Klient = Dienstnachfrager
    - Server = Dienstanbieter
  - lokal
  - verteilt

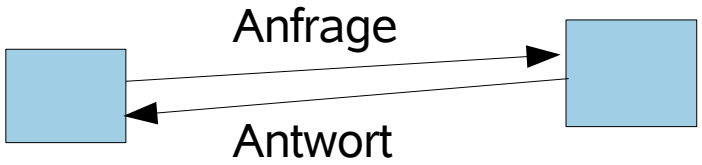


Entwicklung der Systeme

- Klient und Server
  - nur Rollen: Server können auch andere Server ansprechen (und sind dadurch Klienten)



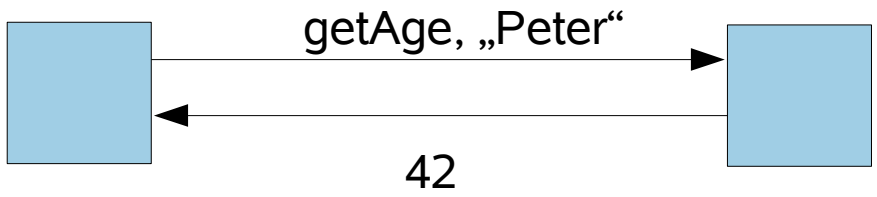
- Kommunikationsschema
  - in der Regel Anfrage – Antwort, daher meist starre Rollenzuordnung



Entwicklung der Systeme

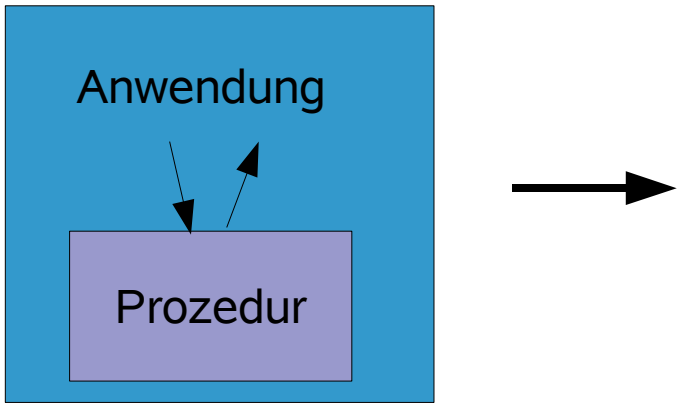
- Anfrage-Antwortschema
  - einfachere Kommunikationsbeschreibung
  - Schnittstellen in Form von Methodensignaturen
  - Anfrage durch Methodenaufruf mit Argumenten, Antwort über Rückgabewert

```
public interface MyObject {
    public String getLocation();
    public int getAge(String sName);
}
```

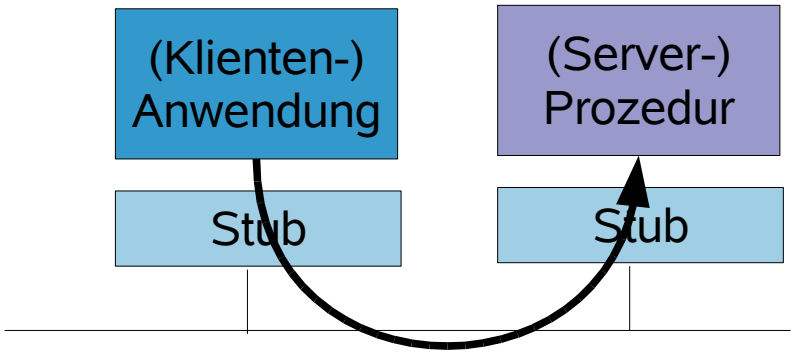


Entwicklung der Systeme

- Entfernte Prozeduraufrufe
  - Kombination von
    - Klient-Server-Technik
    - Prozedurale Programmierung
  - Beispiel: RPC



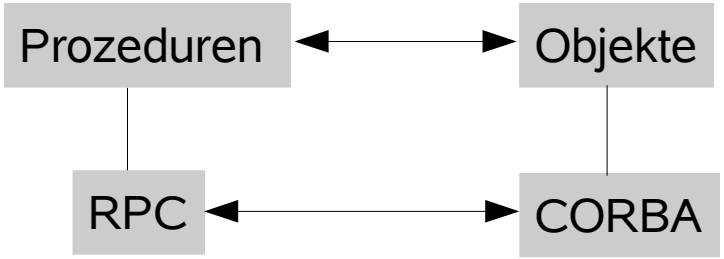
Stubs vermitteln auf Klientenseite den Anschein, die Prozedur sei lokal



Stubs vermitteln den Anschein auf Serverseite, der Aufrufer der Prozedur sei lokal

Verteilte Objekte

- CORBA
  - OMG: Common Object Request Broker Architecture
  - Erlaubt die Verteilung von Objekten im Netz



- Java / RMI
  - Verteiltes Objektmodell, spezifisch für Java

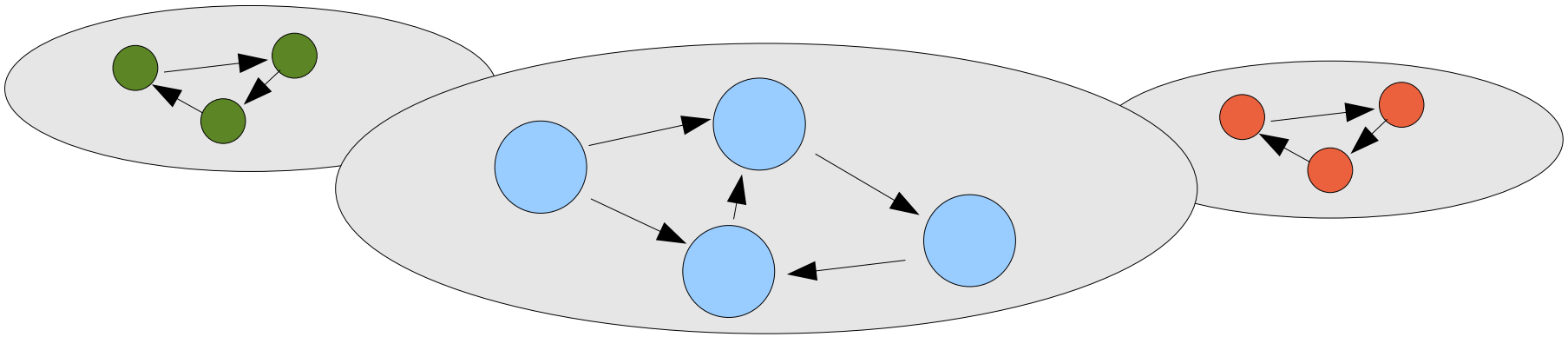
Verteilte Objekte

- Idee
  - Entfernter Zugriff auf Objekt, als wäre es lokal
  - Objekt als Diensterbringer
- MAF
  - Mobile Agent Facility
  - Agenten als Erweiterung der Object Management Architecture
  - Mobile Objekte

Sind Agenten schlichtweg mobile Objekte?

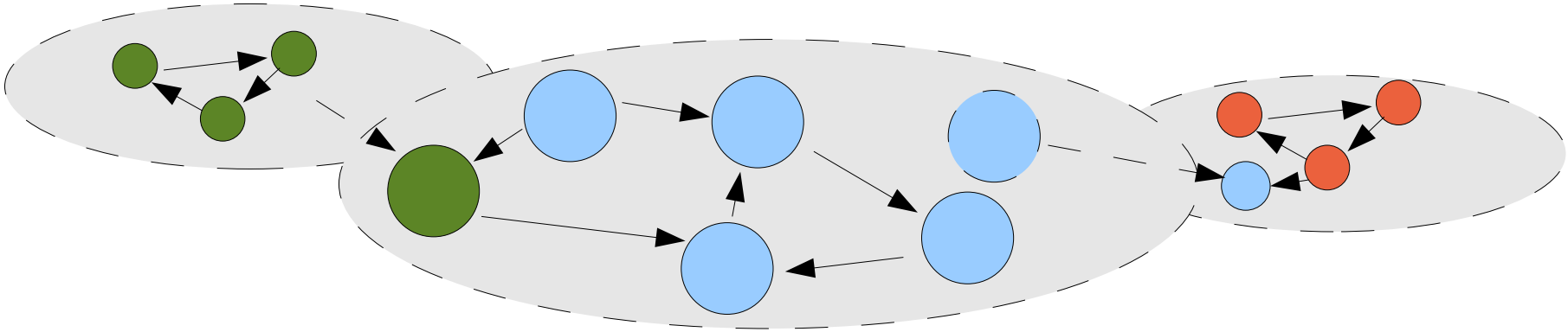
Welten

- Geschlossene Welt
  - Feste Menge an Anwendungen
  - Feste Menge an verwendeten Komponenten und Agenten
  - Dadurch auch maximale Menge an Kommunikationsformen und Interaktionsformen
  - Äußert sich in Objektsystemen durch eine klar definierte Menge an Objekten und Klassen



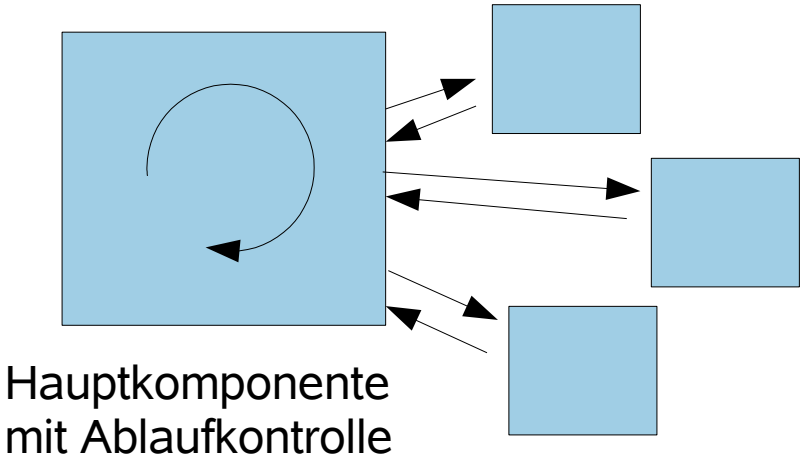
Welten

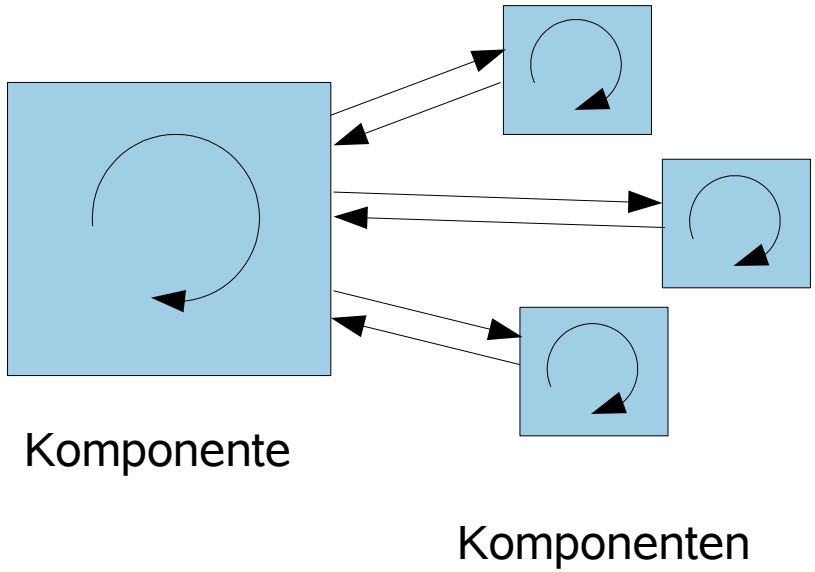
- Offene Welt
  - entsprechend, eben die andere Seite
  - beliebige Zahl von Komponenten und beliebige Zahl von Kommunikationsformen
  - Äußert sich dadurch, dass zur Laufzeit noch immer neue Komponenten auftauchen können



Entwicklung der Systeme

- Nebenläufige Programme
  - Multitasking / Multithreading
  - Dämonen, Hintergrundtasks





Entwicklung der Systeme

- Agenten
  - eigenständige Prozesse (eigener Thread)
  - Objekte (Identität, Zustand, Verhalten)
  - Ggf. Mobilität (sofern von der Infrastruktur angeboten)
  - **Autonomie**

Erste Definition

Agenten sind Objekte, die während ihres Lebenszyklus einen Plan verfolgen, für den sie die Rolle eines Klienten nie aufgeben. Sie erfordern eine Infrastruktur, welche ihnen in dieser Rolle die geeignete Dienste anbietet.

M. Zapf, DAIS-Konferenz 1997

Hintergrund: Alle Initiative geht vom Agenten aus; Serverrolle ist mit Autonomie nicht vereinbar.

Agenten

- Zusammenfassung aller weiteren Merkmale zu einer neuen Definition / Ablösung von strenger Objektnomenklatur

Zweite Definition

Unter einem Softwareagenten versteht man eine Software-Entität, welche als abgeschlossene Einheit verstanden wird und während ihres Lebenszyklus eine eindeutige Identität besitzt. Sie ist in der Lage, mit äußeren Entitäten (ihrer Umgebung) zu interagieren, indem sie Informationen von diesen bezieht oder an diese weitergibt. Sie ist so gestaltet, dass sie in Folge der Interaktionen mit ihrer Umgebung an dieser Umgebung selbständig Modifikationen durchführt, welche als Ergebnis eines in ihrem Verhalten definierten Plans betrachtet werden können.

M. Zapf, Dissertation

Agenten und Objekte

- Sind Agenten Objekte?
  - Nach boochschem Modell müssen sie Zustand, Identität, Verhalten tragen
  - wird in den gebräuchlichen Definitionen explizit oder implizit angenommen
  - weiter gehende Eigenschaften (besitzt Schnittstelle mit Methodensignaturen) sind jedoch nicht unbedingt gegeben
  - Agenten sind Komplexe aus Objekten (Aggregationen)
- Sind Objekte Agenten?
  - Objekte sind „simple“ Agenten, die sich auf Klient-Server-Interaktion beschränken

Klassifikation mobilen Codes

Klassifikation mobilen Codes

- Code on Demand (COD)
- Remove Evaluation (REV)
- Mobile Agent (MA)

Code on Demand

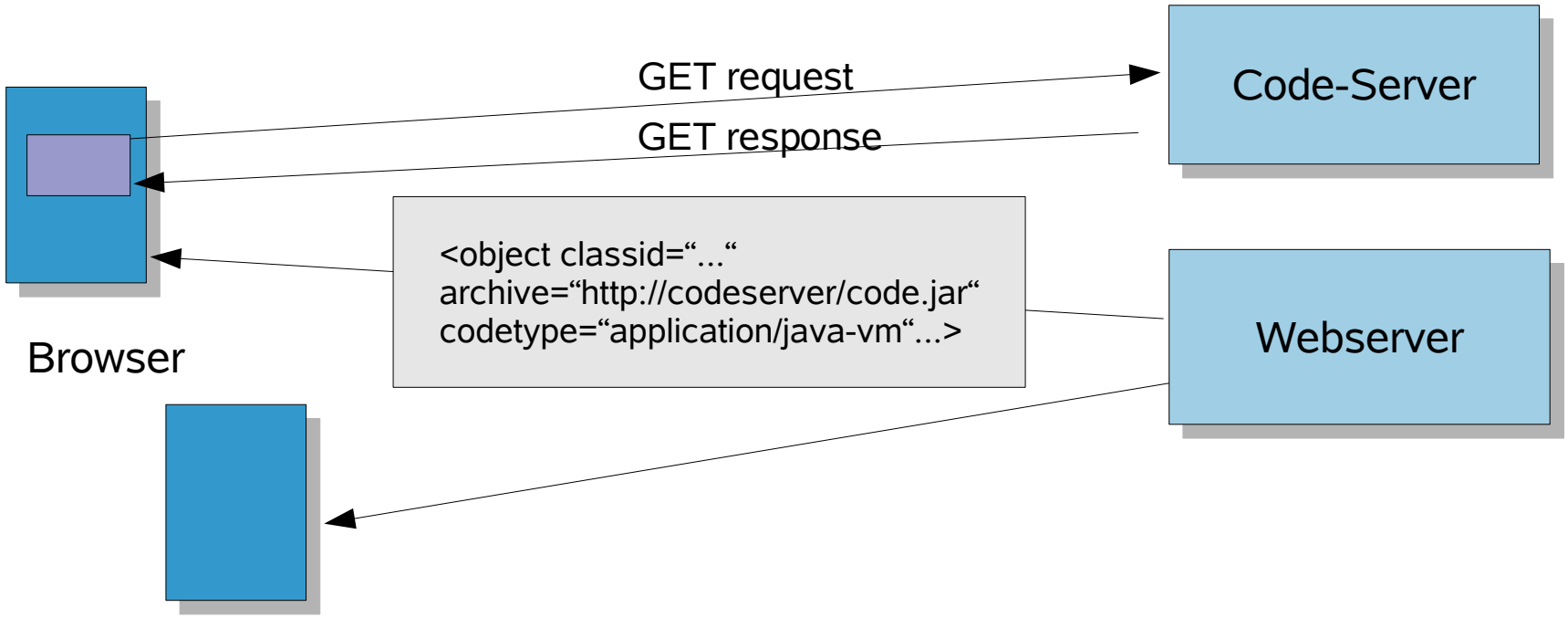
- Idee
  - Code (Programmcode) nicht lokal vorhanden, sondern auf Server
  - Code wird auf Bedarf zum lokalen Rechner kopiert (heruntergeladen)
- Beispiel
  - Java-Applets
  - Skriptsprachen in Webseiten (insbes. Javascript)
  - Shockwave Flash
  - jedoch keine Servlets

Lokale Ausführungsressourcen
Zentrale Speicherungsressourcen
„Code pulling“

CoD am Beispiel Applet

- Code ist unveränderlich für alle Anfragen
  - z.B. gleiches Applet bei gleicher Anfrageadresse
  - Keine persistente Auswirkung der Ausführung

Zustand geht verloren



Remote Evaluation

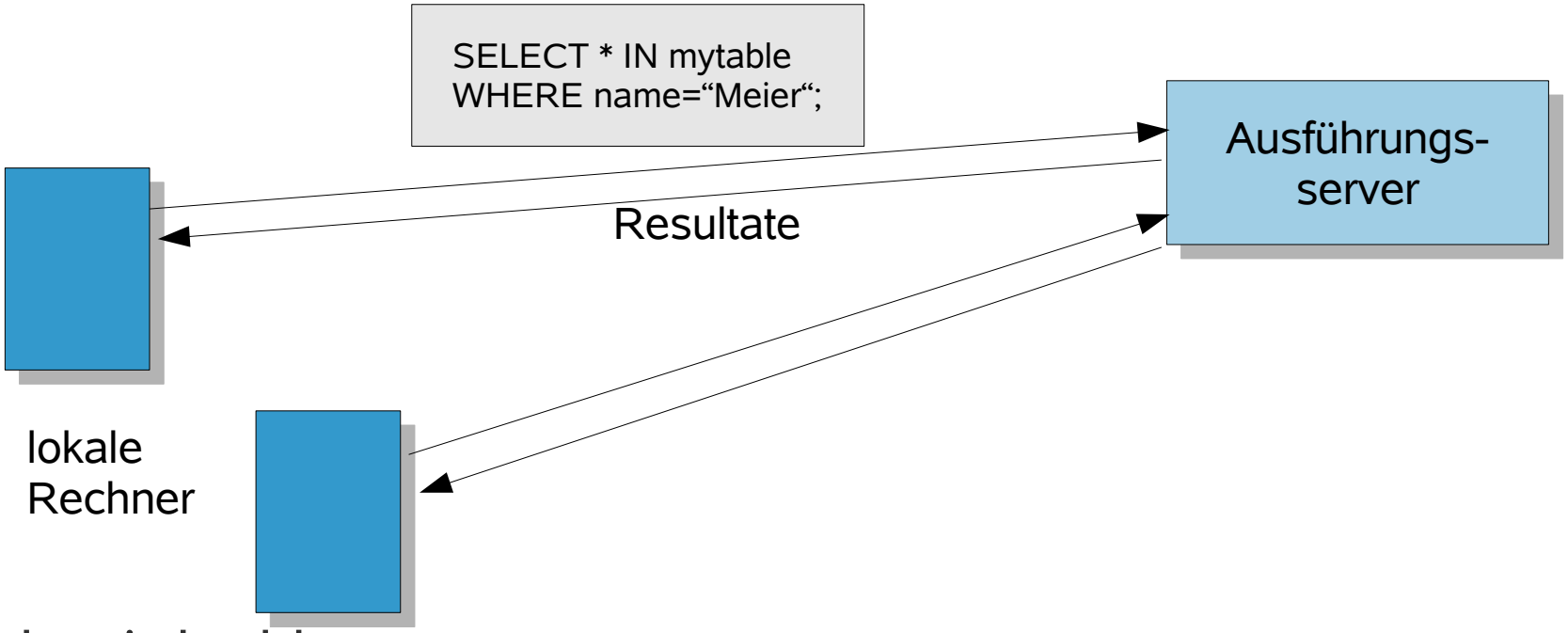
- Idee der „entfernten Auswertung“:
  - Code (Programmcode) lokal vorhanden
  - Code wird zu einem Server hochgeladen und dort ausgeführt
- Beispiel
  - Datenbankabfragen (z.B. in SQL) bei entfernter Datenbankanbindung
  - Postscript (Drucker führt Anweisungen aus)

Lokale Speicherungsressourcen
Zentrale Ausführungsressourcen
„Code pushing“

Remote Evaluation

- Vergleich mit RPC (entfernte Prozeduraufrufe)
  - RPC-Server bietet starre Menge von Funktionen
  - REV-Server stellen sich als „programmierbare Prozessoren“ dar
- Vorteil
  - Wesentlich geringeres Kommunikationsaufkommen!
  - Also mögliches, ähnliches Einsatzgebiet wie MA

REV am Beispiel SQL



- Code wird ad-hoc erzeugt
  - Abfragen sind abhängig von der jeweiligen Situation des Klienten
  - Ausführung ändert Zustand des Servers

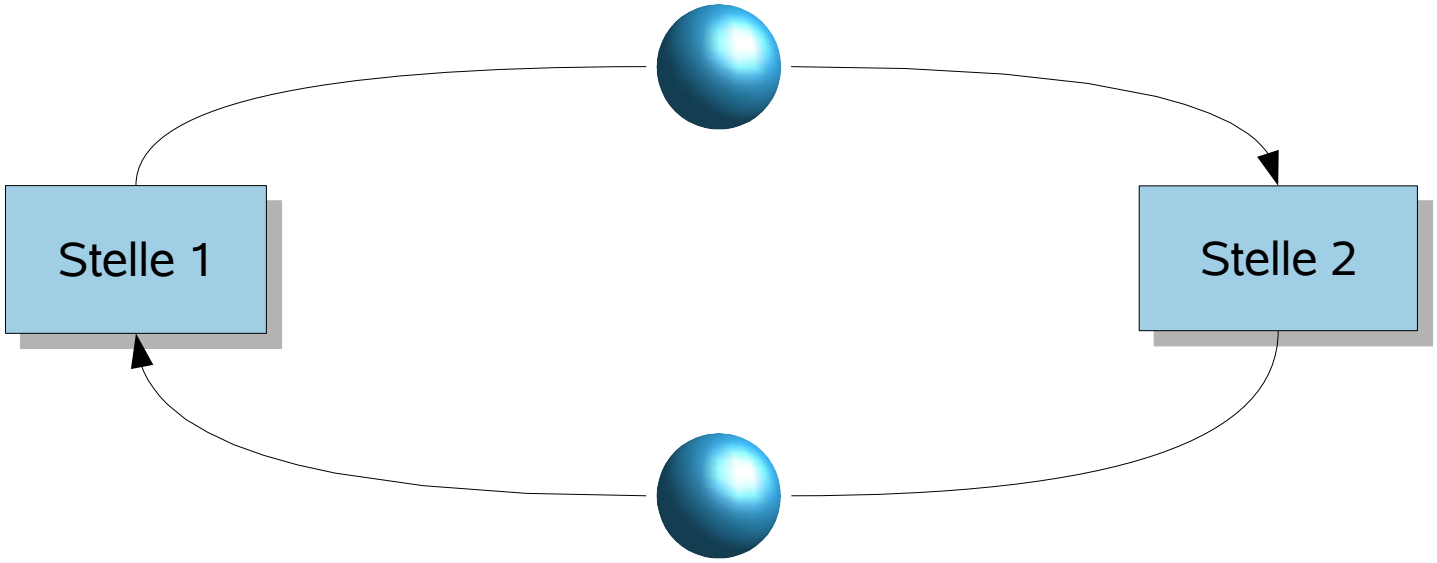
Keine definierten, abgeschlossenen Codeeinheiten (Aggregationen); keine Identität

Mobile Agenten

- Idee
  - Zusammengehörige Einheit von Code, Zustand, Identität
  - Code bestimmt Verhalten (entspricht also boochschem Objektbild)

- Sind MA „mobile Objekte“?
  - andauernde Diskussion zwischen Befürwortern und Skeptikern
  - Agenten haben Autonomiecharakter
  - „Intelligenz“ lange Zeit von geringem Interesse

Typische MA-Anwendung



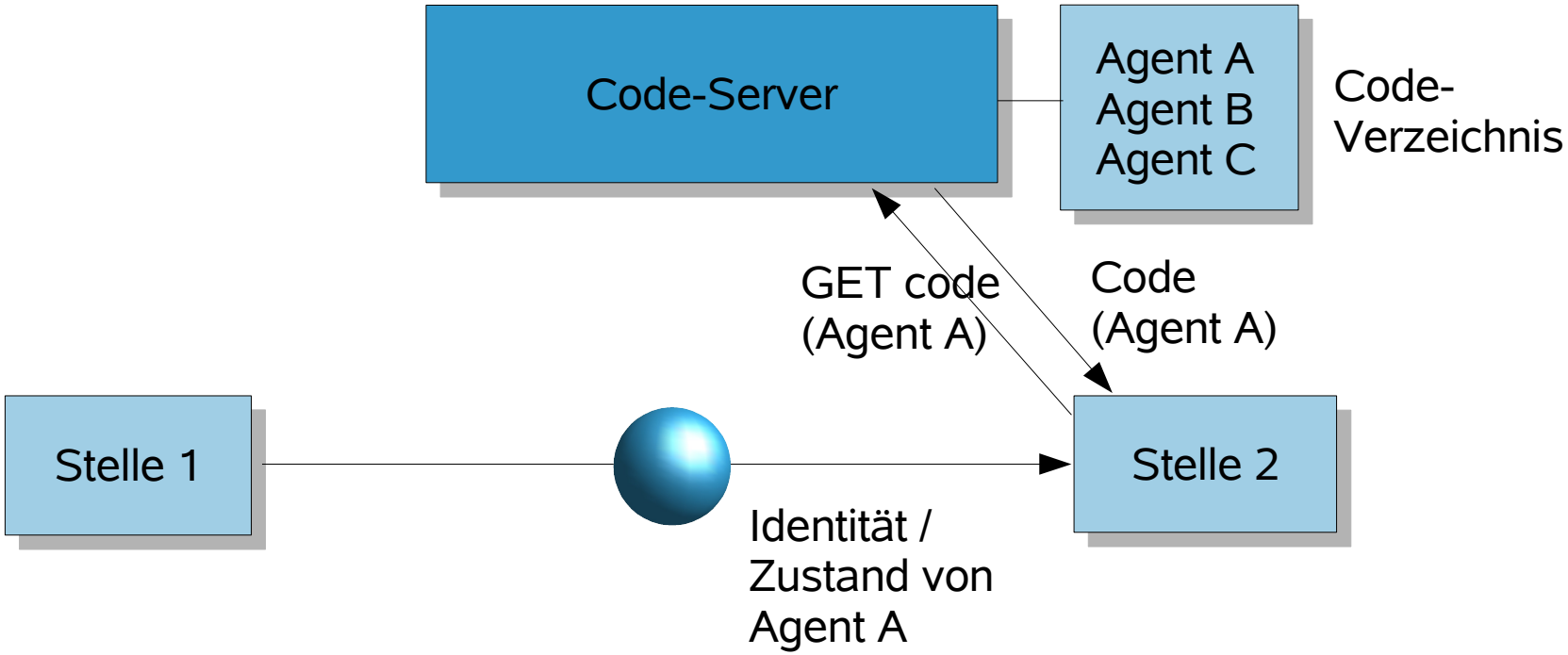
- MA wandert zu einer anderen Lokation („Stelle“)
- MA führt seinen Plan aus
  - Zustand des MA und seiner Umgebung ändert sich
- MA kehrt mit Ergebnissen zurück

Vergleich mit anderen Systemen

- Vergleich mit CoD
  - Agent bewahrt seinen Zustand und kann zum Absender zurückkehren
- Vergleich mit REV
  - Verhalten des Agenten ist vorgegeben; Pläne können Verhalten beeinflussen
  - Agent ist eine wohldefinierte, Zustand und Identät tragende Einheit
- Agenten bringen ihren Code mit
  - Keine Notwendigkeit einer „Installation“ an allen Stellen
  - kann dadurch zu erhöhter Netzbelastung führen
  - Sicherheit?

MA und CoD

- Mischform
  - Mobile Agenten bestehen nur aus der Identität und dem Zustand
  - Code wird zum Herunterladen von einem zentralen Server angeboten



MA und CoD

- Vorteil
  - Keine Versionsprobleme (nur einmal ändern!)
  - Code kann besser gesichert werden (ein Server lässt sich leichter überwachen; Signierung)
- Nachteil
  - Single Point of Failure (wie immer bei zentralistischen Komponenten)
  - Idee des abgeschlossenen Agenten abgeschwächt
  - nur für geschlossene Systeme geeignet