Verteilte Systeme

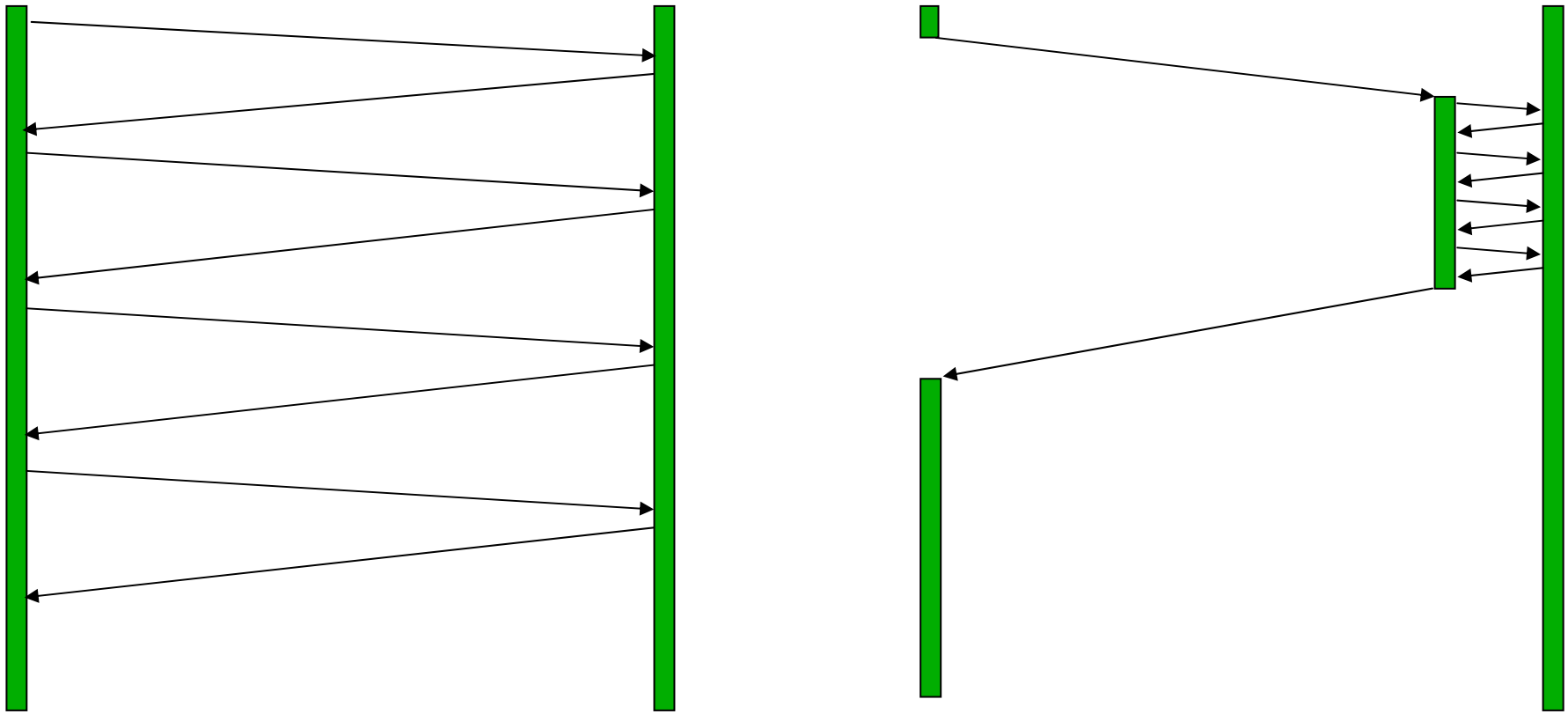
2. Die Client-Server-Beziehung und daraus resultierende Techniken

2.4 Objektmigration

Sebastian Iwanowski
FH Wedel

Objektmigration: Prinzip

Entfernter Zugriff **versus** **Migration**



Objektmigration: Anwendungsziele

- **Verringerung des Zeitaufwands für die Kommunikation**
- **Balancierung der Ressourcenverteilung**
- **Besitz- oder Verantwortungswechsel der SW im verteilten System**
- **Einsatz auf mobilen Geräten**
- **Ermöglichung des off-line-Betriebs von Clients und Servern für lange Zeiträume**

Klassifikation nach Eigenschaften der migrierten Objekte

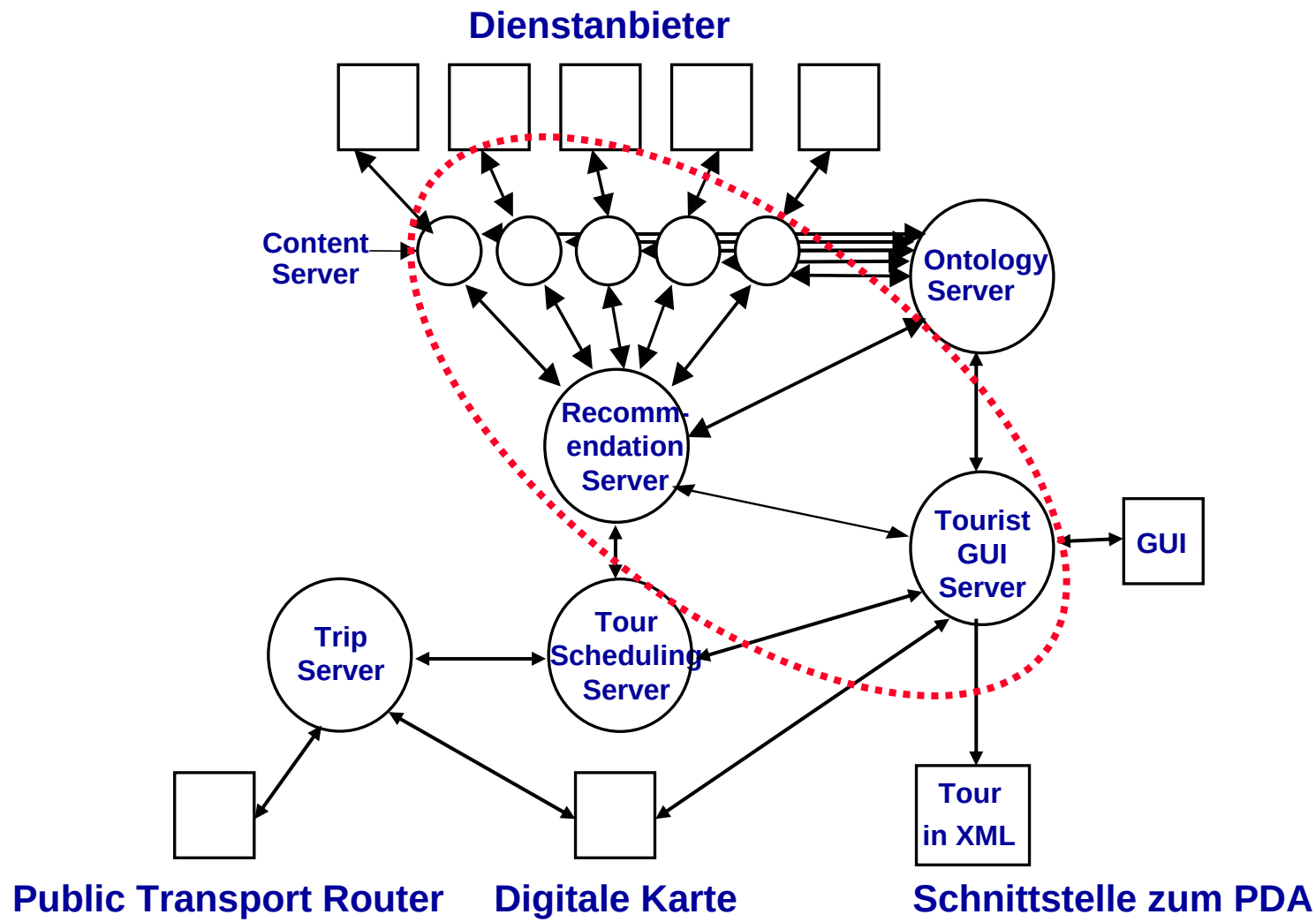
- **Objekte als Datenbehälter** — Klassische Datenübertragung
- **Objekte als Methodenträger** — Klassische Objektmigration
- **Objekte als autonome Softwareeinheiten mit eigenen Zielen** — Mobile Agenten

Anforderungen zur Ermöglichung von Objektmigration

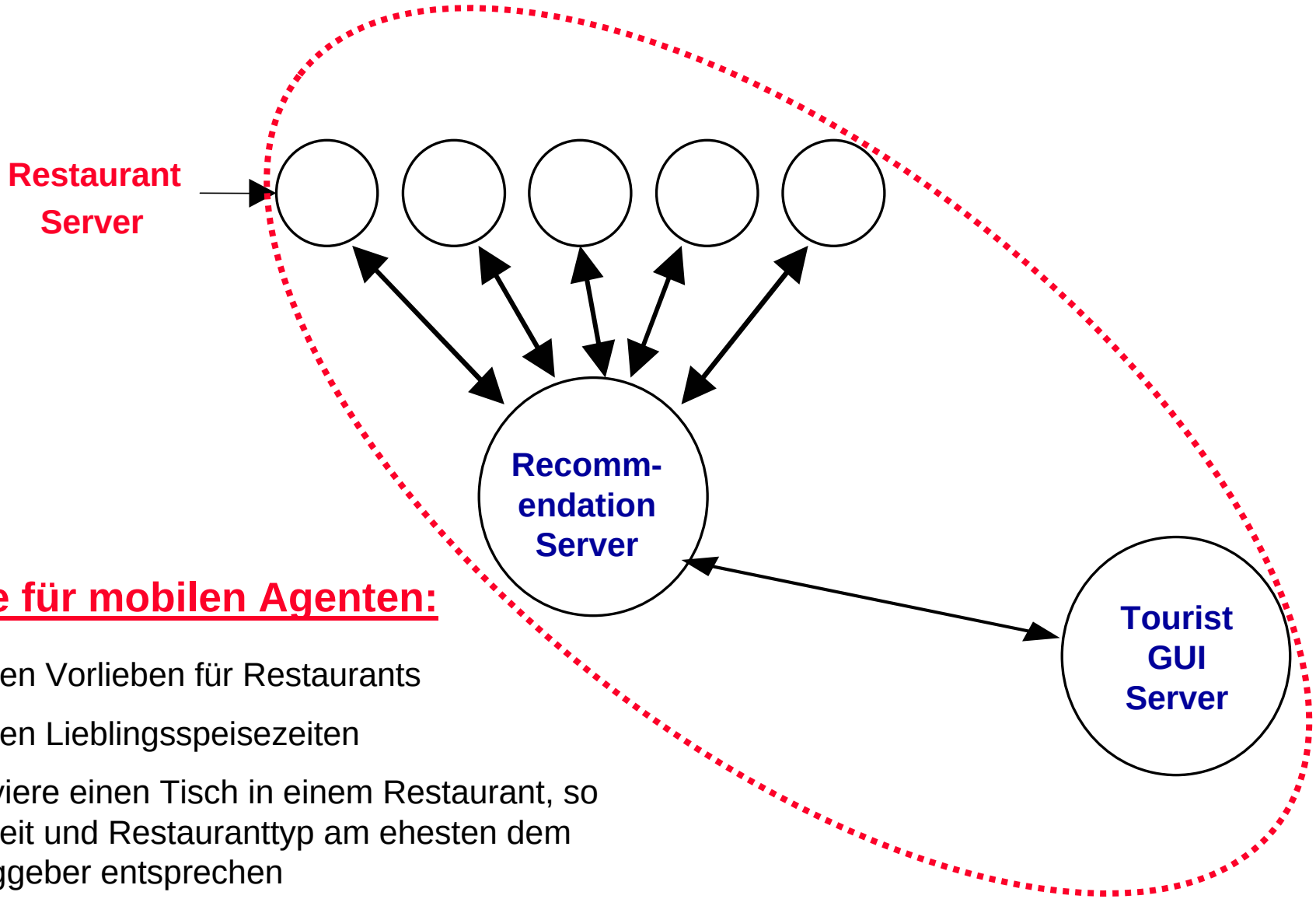
gelten gleichermaßen für klassische Objektmigration und mobile Agenten:

- **Laufzeitumgebung an allen Hosts, die besucht werden sollen**
- **Registratur zum Auffinden von mobilen Objekten**
- **Stubs (Proxies), die zur Laufzeit verändert werden können**

Praxisbeispiel: Touristeninformationssystem



Praxisbeispiel: Touristeninformationssystem



Aufgabe für mobilen Agenten:

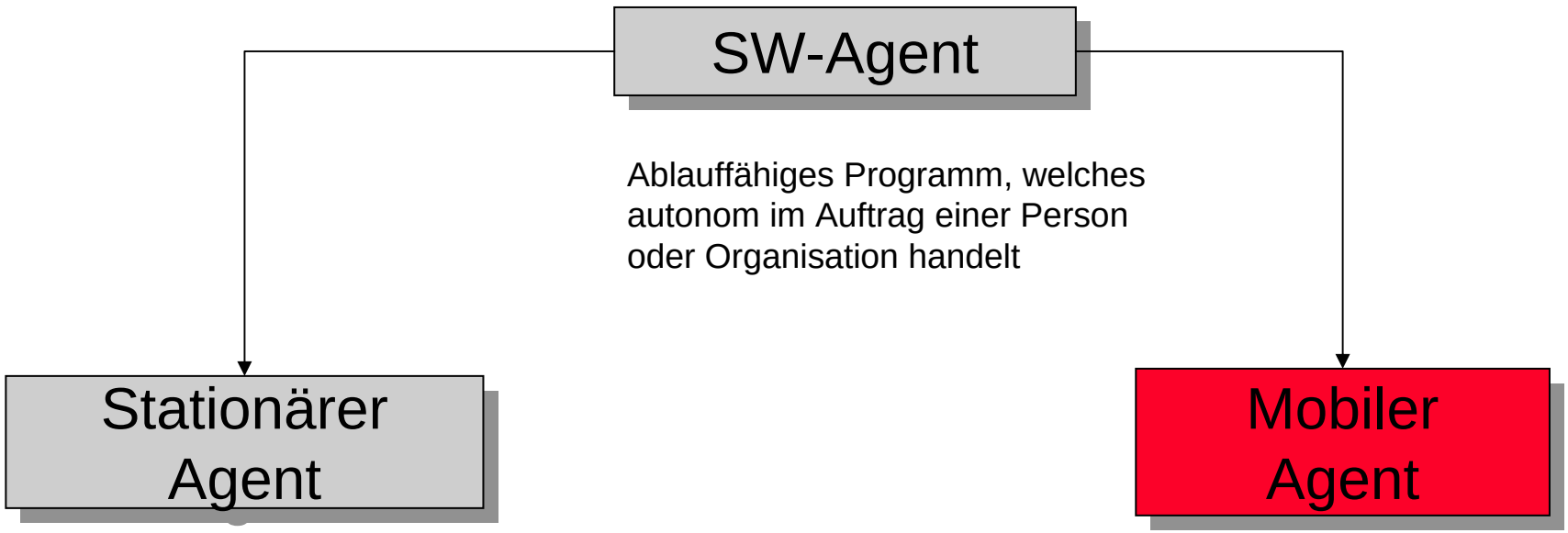
- Gegeben Vorlieben für Restaurants
- Gegeben Lieblingsspeisezeiten
- Reserviere einen Tisch in einem Restaurant, so dass Zeit und Restauranttyp am ehesten dem Auftraggeber entsprechen

Auszüge aus dem Seminarvortrag Nr. 4 des Informatik-Seminars SS 2004 von Matthias Rohr zum Thema:

Mobile Agenten

Die gesamte Präsentation sowie eine Ausarbeitung befinden sich im Internet unter ~si/seminare/ss04/Termine/Themen.html, erreichbar über ~iw/Archiv

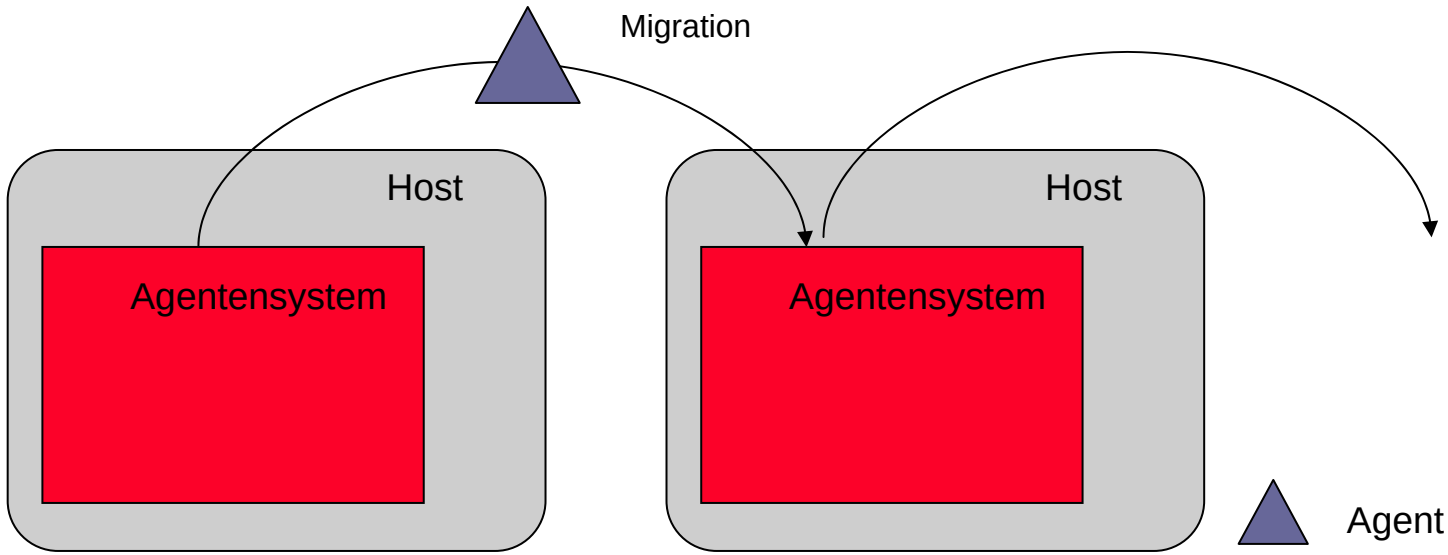
Seminarvortrag: Agententheoretische Grundlagen



Quelle: MASIF-Spezifikation

Seminarvortrag: Agententheoretische Grundlagen

SW-Eigenschaft: Mobilität



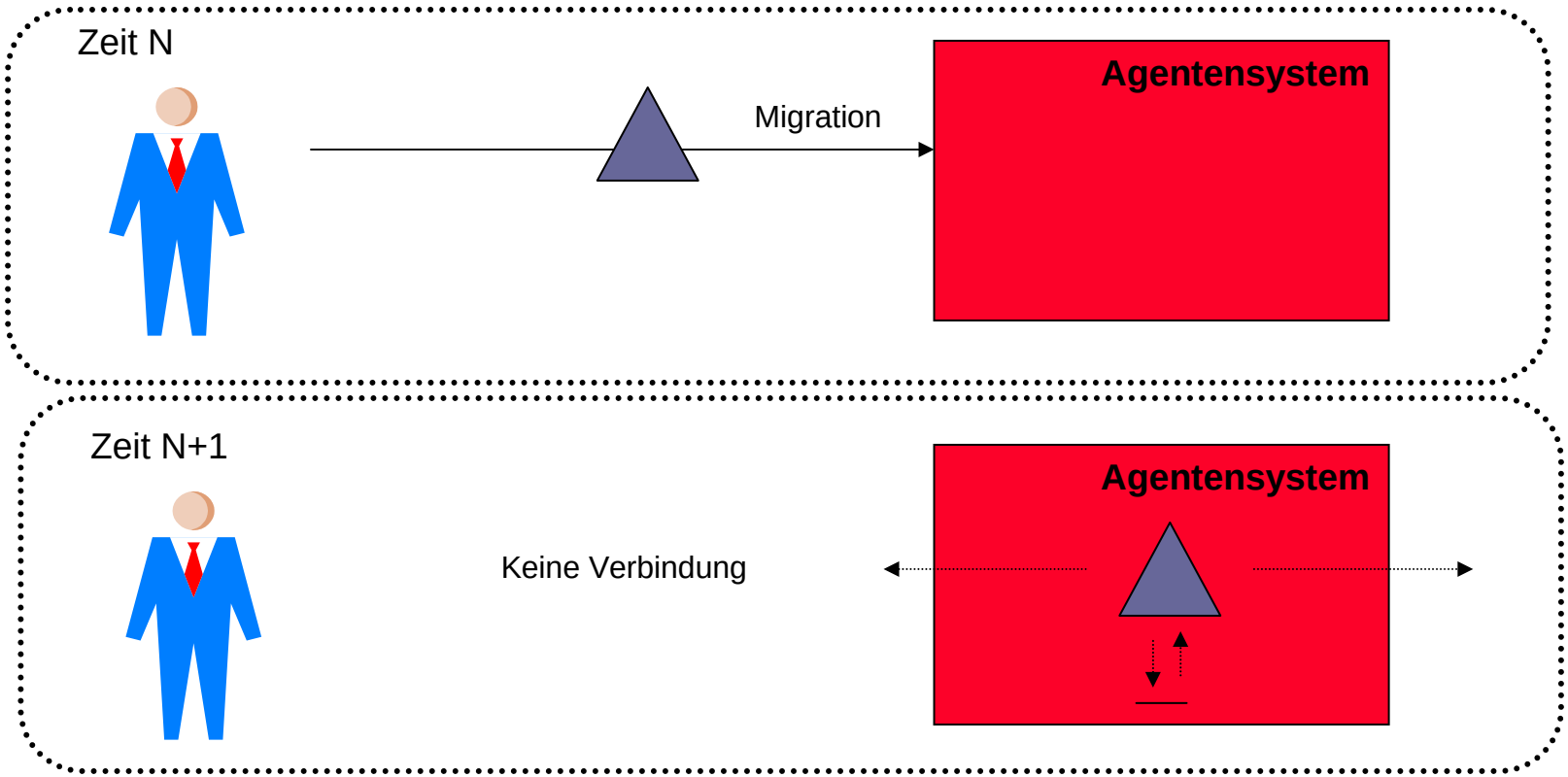
Agent kann zwischen Agentensystemen wechseln (migrieren)

Voraussetzungen:

- Agenteninstanz kann in Datenfluss umgewandelt werden (Serialisierbarkeit)
- Die Zielsystem kann den Code des Agenten empfangen und interpretieren

Seminarvortrag: Agententheoretische Grundlagen

Agenteneigenschaft: Autonomie



Der Agent entscheidet selbstständig anhand bestimmter Kriterien über seine nächste Aktion (Problem: Kontrollmöglichkeiten).

Seminarvortrag: Agententheoretische Grundlagen

Agenteneigenschaft: Soziale Kompetenz

Agent kann mit anderen

- Agenten
- (Agenten-)Systemen
- Menschen

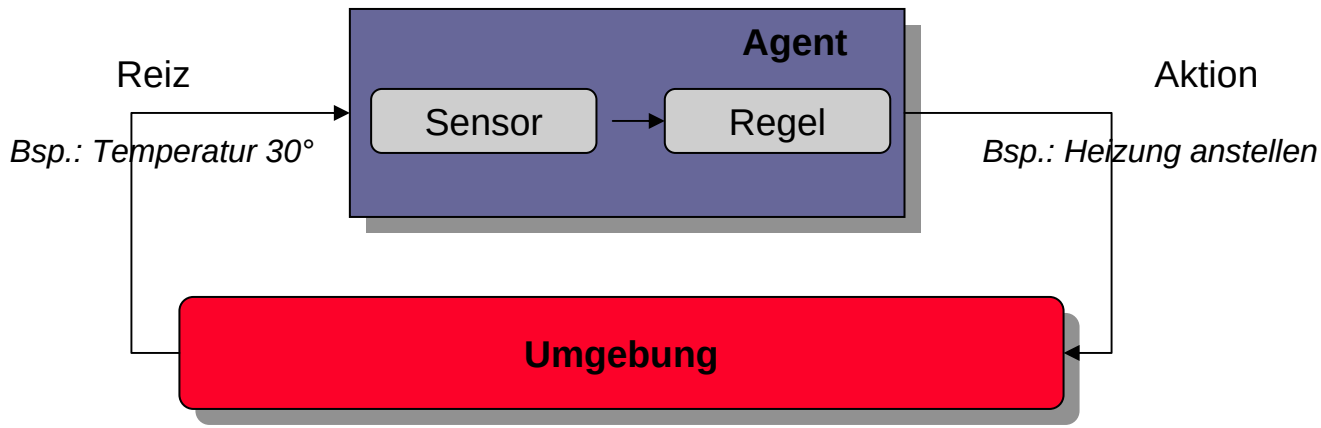
kommunizieren / kooperieren

Vorraussetzungen:

- Gemeinsame Protokolle
- Gemeinsame Sprache / Schnittstellen
- Gemeinsame Ontologien

Seminarvortrag: Agententheoretische Grundlagen

Agenteneigenschaft: Reaktivität

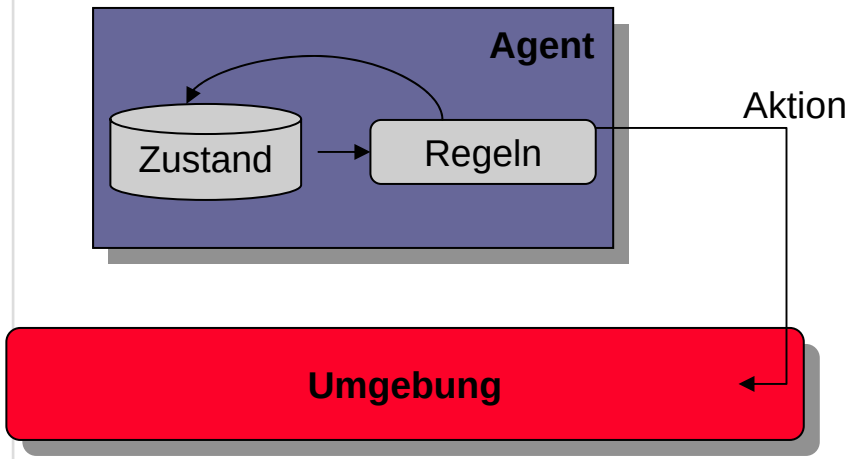


Der Agent reagiert auf Änderungen seiner Umgebung (engl. Environment)

Vorraussetzung: Existenz von Sensoren und Regeln

Seminarvortrag: Agententheoretische Grundlagen

Agenteigenschaft: Proaktivität (Zielgerichtetheit)



Agenten reagieren nicht nur auf Reize der Umgebung, sondern besitzen internen Zustand und sind zu zielgerichtetem Planung und Handeln fähig.

=> Sie ergreifen die **Initiative**

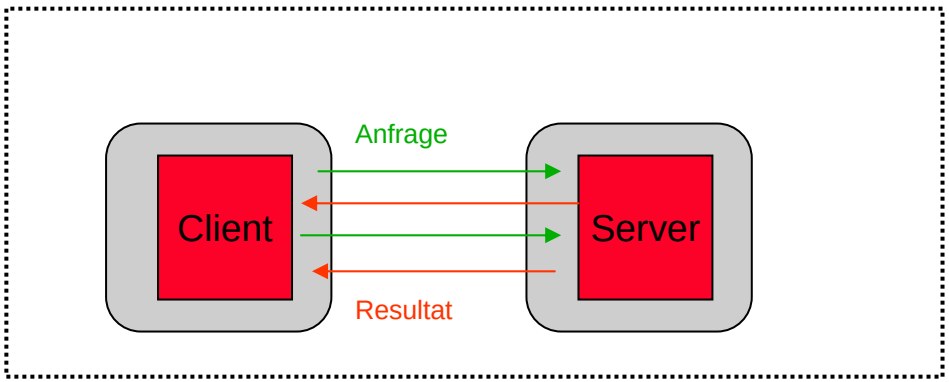
*„The difference between an automation and an agent is a somewhat like the difference between a dog and a butler. If you send your dog to buy a copy of the New York Times every morning, it will come back with its mouth empty if the news stand happens to have run out one day. In contrast, the butler will probably take the **initiative** to buy you a copy of the Washington Post, since he knows, that sometimes you read it instead."*

Le Du

Seminarvortrag: Agententheoretische Grundlagen

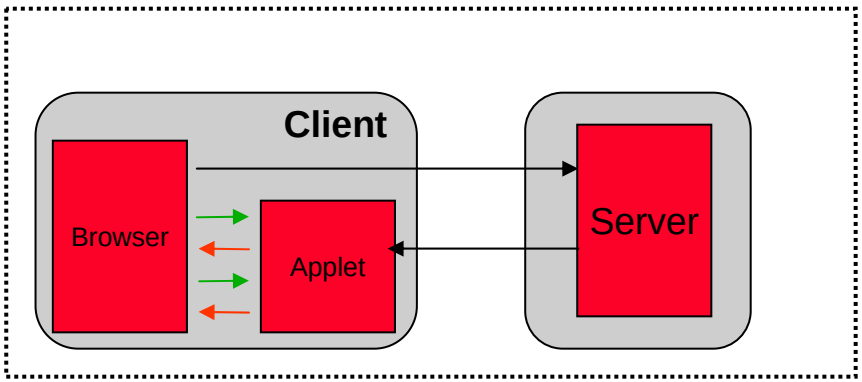
Mobile Agenten als Softwareparadigma: Abgrenzung

Client Server (CS)



- Funktionsweise wird vom Server festgelegt
- Ressourcenverbrauch beim Server

Code on Demand (CoD),
z.B.Java Applets



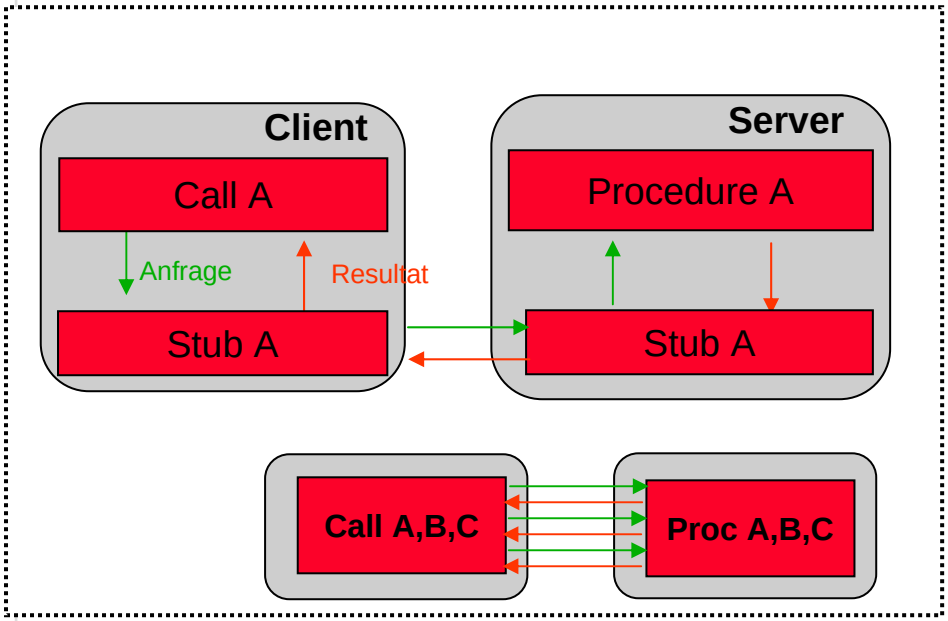
- Funktionsweise wird vom Server festgelegt
- Ressourcenverbrauch beim Client

Seminarvortrag: Agententheoretische Grundlagen

Mobile Agenten als Softwareparadigma: Abgrenzung

Remote Evaluation (REV)

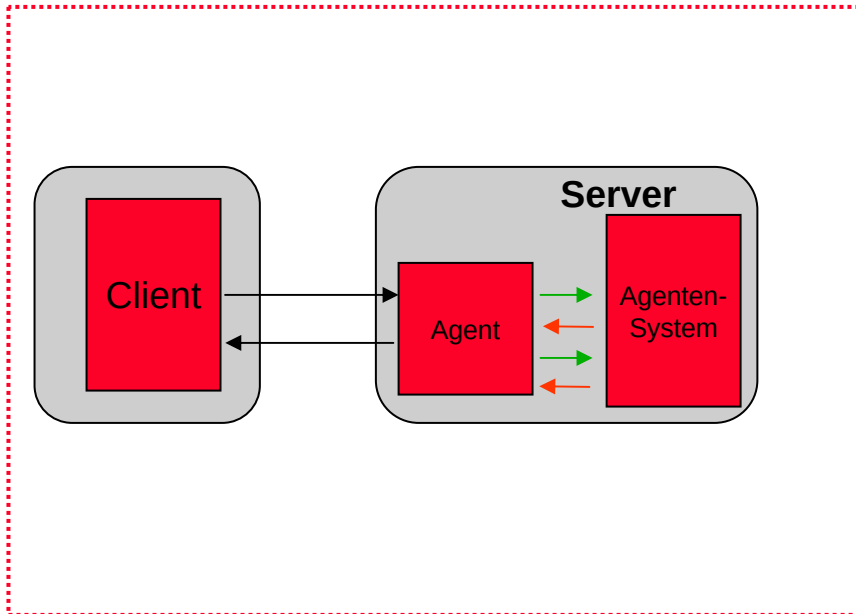
z.B. RPC, RMI, Corba



- Funktionsweise wird vom Server festgelegt
- Client behandelt Funktion wie eine eigene
- Ressourcenverbrauch hauptsächlich Server

Mobile Agent (MA)

z.B. Aglets, Voyager, Grasshopper

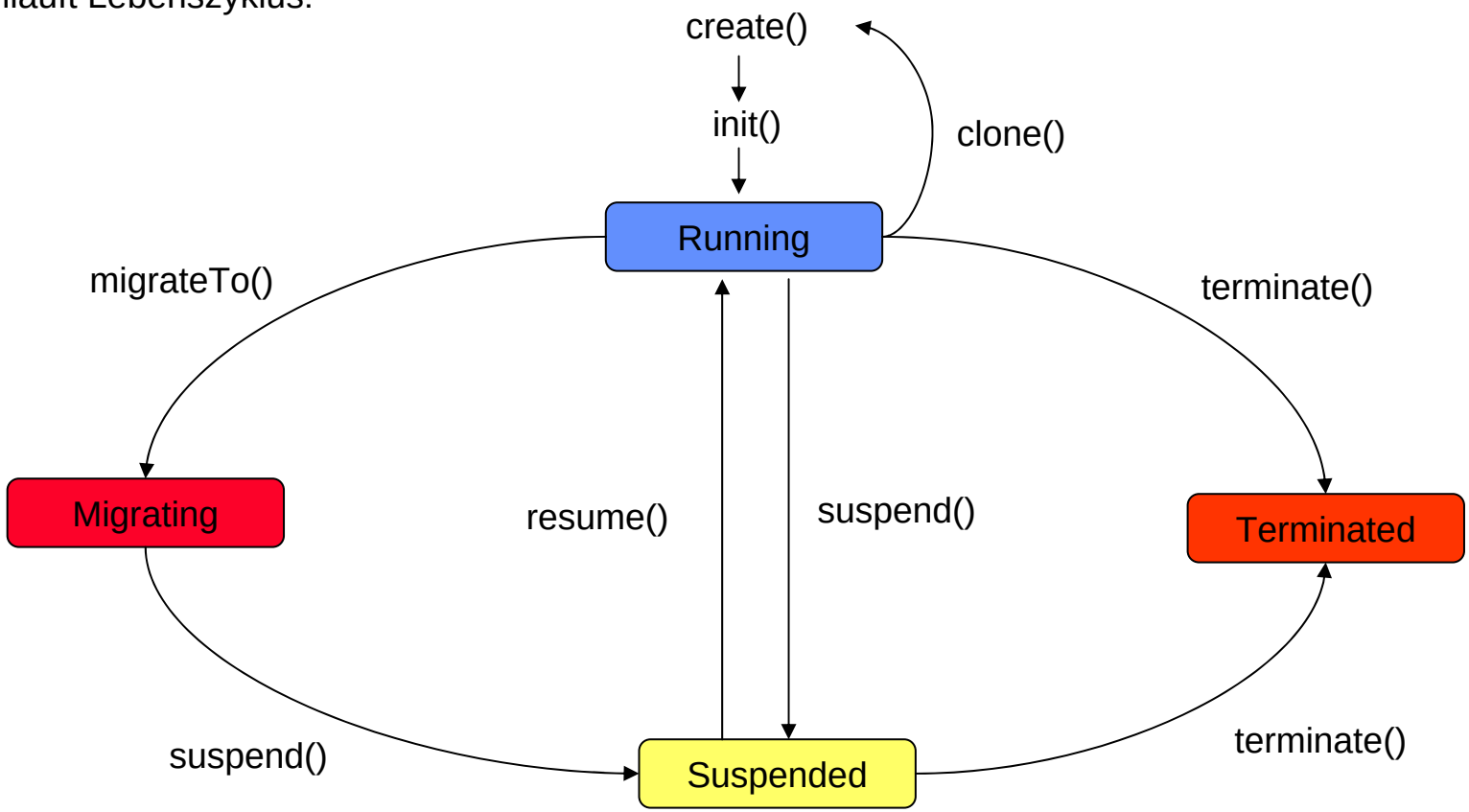


- Funktionsweise wird vom Client festgelegt
- Ressourcenverbrauch beim Server

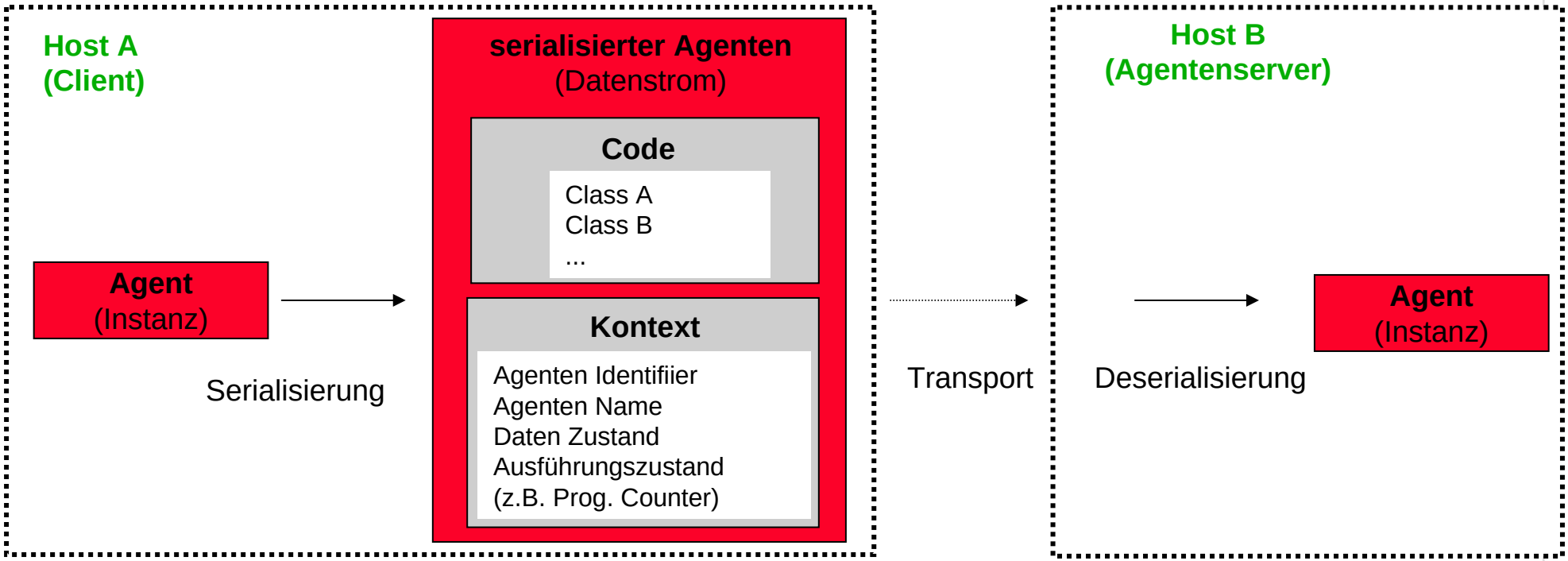
Seminarvortrag: Agententheoretische Grundlagen

Mobile Agenten als Softwareparadigma: Grundfunktionalität

- Besitzt eindeutige Identifikation / Namen
- Exklusivität ? (Wie oft darf ein Agent existieren ?)
- Durchläuft Lebenszyklus:



Seminarvortrag: Migration von Agenten



Unterschiedliche Migrationsarten (was migriert ?)

- **schwache Migration:** ohne Ausführungszustand
- **starke Migration:** mit Ausführungszustand

Seminarvortrag: Migration von Agenten

Schwache Migration

mit festem Einstiegspunkt (z.B. Aglets)

```
public class Display
          implements java.io.Serializable
   Boolean isRemote;
   public class Example1 extends Aglet {
       public void onCreation (Object init) {
       addMobilityListener( new MobilityAdapter() {
          public void onArrival( MobilityEvent e) {
                System.out.println(„i am remote“);
                isRemote = true;
          }
       }
   };

public void run() {
   if (! isRemote) {
          moveTo(„Host1“);
   }
}
```

mit beliebigem Einstiegspunkt (z.B. Voyager)

```
public class Display
          implements java.io.Serializable
{
   public void display(String message )
   {      System.out.println( message);    }
}

public class AgentClient {
    public static void main( String[] args )
   {
      Display agent1 = Proxy.of( new
                                  Display() );
      Agent.of( agent1 ).moveTo(
         "//host:8000", "display",
         new Object[]{
            „show on remote";
}
```

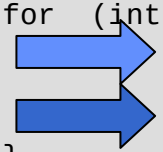
Ausstiegspunkt Host A

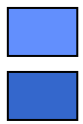
Einstiegspunkt Host B

Seminarvortrag: Migration von Agenten

Starke Migration (= Transparente Migration)

```
public class AgentClient extends StrongAgent {
    public static void main( String[] args )
   {
           String Hosts[] = { „Host1“, ... „Hostn“ };
           for  (int i=0; i < Hosts.length(); i++) {
                  //lokal
                  moveTo(Hosts[i] );
                  // remote
           }
   }
};
```





Ausstiegspunkt Host A

Einstiegspunkt Host B

- Einstiegspunkt = Ausstiegspunkt
- Möglich durch Serialisierung des Ausführungsstacks (z.B. Program Counter)
- Selten umgesetzt da sehr schwierig zu realisieren; z.B. in D‘Agents (AgentTCL)

=> je schwächer Migrationsart, desto mehr Aufwand hat der Programmierer

=> je stärker die Migrationsart, desto komplizierter für das Agentensystem

Seminarvortrag: Migration von Agenten

Migration in Java

- Serialisierung erfolgt durch *Java Objekt Serialisierung*

```
class Agent implements
          java.io.Serializable
```

- Datenzustand kann separat angegeben werden

serialisiert:

```
class {
          int a;
          static String b;
```

nicht serialisiert:

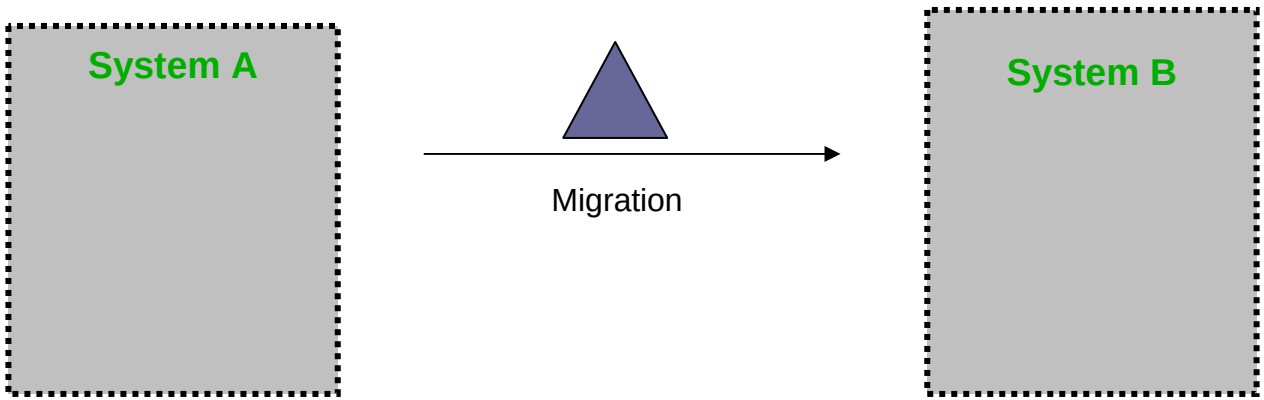
```
class {
           transient int c;
```

- Starke Migration wird nicht direkt unterstützt. Möglichkeiten:

1. Durch Anpassung der JVM
   - keine Kompatibilität zu anderen Systemen
2. Durch Anpassung des Sourcecodes
   - sehr aufwendig zu Programmieren (siehe Fünfröcken ‘98)

Seminarvortrag: Migration von Agenten

Migrationsablauf



1. Ausführung des Agenten unterbrechen
2. Zu übertragenen Teile des Agenten identifizieren
3. Agent serialisieren
4. Kodieren entsprechend dem verwendeten Transport Protokoll
5. Authentifizierung bei System B
6. Agent mit Code und Zustand verschicken

1. Authentifizierung von System A
2. Decodierung
3. Deserialisierung
4. Instantiierung
5. Agentenzustand wiederherstellen
6. Fortsetzen der Ausführung

nach MAFO-Spezifikation

Seminarvortrag: Migration von Agenten

Push vs. Pull Migration (wann migriert was ?)

1. **Push Migration**
   - Migriert werden Datenzustand + Klassen des Agenten
   - Eingesetzt z.B. von Aglets, Concordia

2. **Pull Migration**
   - Migirert wird zunächst nur der Datenzustand
   - Benötigte Klassen werden nach Bedarf einzeln nachgeladen
   - Vorteil: u.U. Einsparung von Bandbreite
   - Nachteil: Bei kleinen Agenten zusätzliche Bandbreite (siehe RFC vs. MA)
   - Eingesetzt z.B. von Voyager, Grasshopper

Seminarvortrag: Agentenkommunikation

Agent <-> Agent	Kooperation
Agent <-> System	Aufgabenerfüllung
System <-> System	Steuerung, Agentensuche

Seminarvortrag: Agentenkommunikation

Agent <-> Agent

1. **Direkt**
   - Agent sendet Nachricht direkt an Adresse des anderen Agenten
   - Adressierung: Agent -> AgentABC@AgentHost ...

- **Indirekt**
  1. Benannt (durch Mailboxen)
     - Agent schickt Mitteilung an Mailbox des anderen Agenten
     - Adressierung: Agent -> „MailboxABC“
  2. Anonym (durch Blackboards)
     - Agent schickt Mitteilung an ein Blackboard (Schwarzes Brett) ohne direktem Agentenbezug
     - Adressierung: Agent -> „Queue123“

Seminarvortrag: Sicherheit bei mobilen Agenten

1. **bei Transport / Migration**
   - Man-In-The-Middle
2. **vor böswilligen Agenten**
   - spähen Systeminformationen aus
3. **vor böswilligen Agentensystemen**
   - senden Agent an anderen / falschen Ort weiter
   - spähen Agenteninformationen aus
   - verändern Code
   - manipulieren Abfragen

Lösung (Ansatz):

- Verschlüsselung + Signierung des Agenten (z.B. SSL, PGP)
- Migration nur zu vertrauenswürdigen /geschlossenen Systemen (z.B. durch Zertifizierungsstellen)
- keine sensiblen Daten im Agenten ablegen (Was wird serialisiert ?!)
- Vollständige Absicherung (insbesondere für Fall 3) nur schwer möglich

Seminarvortrag: Plattformen für mobile Agenten

- Verschiedene Mobile-Agenten-Systeme (Stand 2000). Darunter:
  - AgentTCL alias D‘Agents (TCL)
  - IBM Aglets (Java)
  - Odysee (Java)
  - Voyager (Java)
  - SeMoA (Java)
  - Grasshopper (Java)
  - ...
- Teilweise sehr unterschiedliche Konzepte (Migration / Kommunikation / Sicherheit)
- Kaum Interoperabilität / Agenten-Portabilität zwischen diesen Systemen

Fazit im Seminarvortrag zum Thema Mobile Agenten

- Viele mögliche Anwendungsgebiete (v.a. durch steigende KI)
- Aber keine „Killerapplikation“ in Sicht
- Viele unterschiedliche / inkompatible Konzepte von Agentensystemen
- Standards lassen viele Punkte offen:
  - Agenten
  - Sicherheit
  - Anwendungen
- Kaum Umsetzung der vorhandenen Standards
- Konkrete Ansätze zur Portabilität in vorhandenen Systemen kaum erkennbar
- Mobile Agenten bisher noch hauptsächlich theoretische Disziplin

Mobile Agenten: Mögliche Einsatzgebiete

- **Suchmaschinen**
- **Börsenbeobachter**
- **Elektronischer Preisvergleich**
- **Mehrwertleistungen**
- **Just-in-Time-Produktion**
- **Mobile Computing**

Wichtige Richtlinie:

Keine mobilen Agenten in sicherheitsrelevanten Anwendungen !

Mobile Agenten: Bewertung

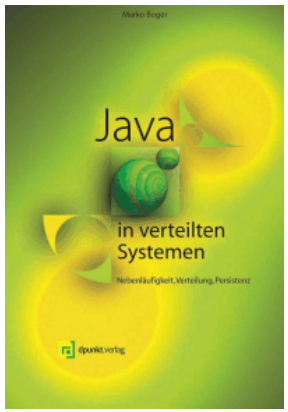
Vorteile:

- **Reduktion der Netzwerkkommunikationslast**
- **Möglichkeiten zum Offline-Gehen des Auftraggebers, während der Agent aktiv ist**
- **Flexiblere Reaktion auf Umgebung möglich als z.B. reines RMI**

Nachteile:

- **Komplexität der Programmierung/Erstellung**
- **Gewisse Infrastruktur erforderlich**
- **Sicherheitsprobleme**
  (sowohl für den Agenten und als auch für die jeweilige Wirtsumgebung)

Weitere Java-Beispiele



Einfache Beispiele mit vollständigem Programmiercode in:

Marko Boger: *Java in verteilten Systemen*, Nebenläufigkeit, Verteilung, Persistenz, dpunkt.Verlag 1999, ISBN 3-932588-32-0

Wir haben bisher behandelt: Kap. 1 bis 4

Kap. 6 und 7: Objektmigration mit dem System Voyager

Programmiercode zu Buchbeispielen elektronisch verfügbar unter:

http://www.dpunkt.de/leseproben/3-932588-32-0/jivs_code.zip