Verteilte Systeme
(Betriebssysteme II)

Kapitel 5.2: klassische Middleware

Prof. Dr. Wolfgang Küchlin

Dipl.-Inform., Dr. sc. techn. (ETH)

Arbeitsbereich Symbolisches Rechnen
Wilhelm-Schickard-Institut für Informatik
Fakultät für Informations- und Kognitionswissenschaften

Universität Tübingen

Steinbeis Transferzentrum
Objekt- und Internet-Technologien (OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

Überblick

- Virtual Shared Memory
- Linda
- DCE
- Mobile Agenten

Virtual Shared Memory

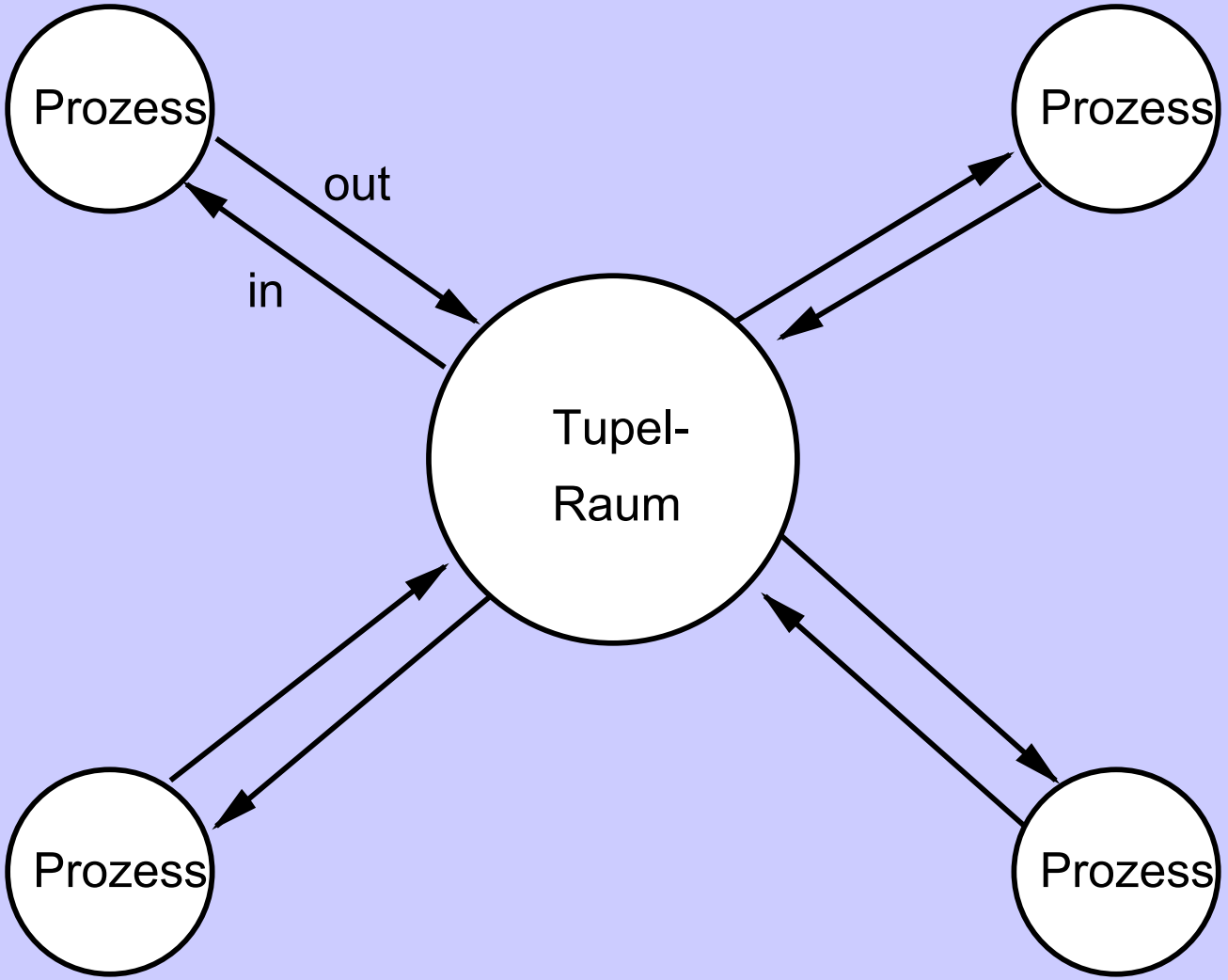
- Simulation eines gemeinsamen Hauptspeichers auf dem Netz
- Grundgedanke:
  - jeweils einem Schreiber wird Exklusivzugriff auf ein Speichersegment gegeben
  - Segment wird per Nachrichten an die Leser verteilt.

  → verteilter wechselseitiger Ausschluss gebraucht.
- zentrale oder dezentrale Realisierungsmöglichkeit (vgl. PVM)
- Verwaltungsaufwand hoch
- Beispiele
  - verteilte Process Run Queue im VSM: wenig Kommunikations-Overhead
  - globale Variablen, die selten geschrieben und oft gelesen werden

Linda

- entwickelt von Ahuja, Carriero und Gelernter (Bells Labs und Yale University)
- paralleles Programmieren auf abstrakter und portabler Ebene
- Linda entkoppelt Kommunikationspartner durch **gemeinsamen verteilten Nachrichtenspeicher**
  → Linda **Tupel-Raum** (*tuple-space*).
  - Tupel-Raum enthält Nachrichten-Tupel, die von Prozessen dort abgelegt und von dort gelesen werden
- ungeordneten Menge von Prozessen, die nur über den Tupel-Raum kommunizieren.
- keine Punkt-zu-Punkt Verbindung und keine weitere Prozess-Synchronisation

Linda



Linda – Die Tupel-Operationen

- `out(<tupel>);`
  - Schreibe mein Tupel in den Tupel-Raum
- `in(<pattern>);`
  - Hole ein Tupel, das dem Muster <pattern> gehorcht aus dem Tupel-Raum
- `read(<pattern>);`
  - Lese ein Tupel, das dem Muster <pattern> gehorcht aus dem Tupel-Raum. Tupel verbleibt im Tupel-Raum.
- `eval(<tupel>);`
  - Schreibe einen **un**-evaluierten Funktionsaufruf in den Tupel-Raum. Er wird später vom Empfänger ausgeführt (evaluiert).

Linda Tupel

- Tupel ist Sequenz von getypten Werten
  - <"p",5,false>.
- Muster (*pattern*) ist Tupel mit ”formalen Parametern“
  - bei Match werden formale Parameter durch Komponenten aus Tupel ersetzt
  - Tupel <"p",5,false> passt zu Muster <"p",int i, bool b>
    - Match hat Seiteneffekt i=5, b=false.
- Tupel-Raum ist universeller Kommunikationskanal
  - Unterteilung in Subkanäle, die jeweils nur Tupel mit gleichem ersten Parameter enthalten → erhöhte Effizienz der Linda Operationen
- Realisierung des Tupel-Raums
  - Implementierung als verteilte Hash-Tabelle
    - Auswahl des Hosts für Speicherung gemäß Hash-Wert des Tupels
  - Optimierungen: Wird z.B. Tupel nur an einer Stelle erzeugt und konsumiert → direkte Umsetzung in Nachricht

Linda – Matrixmultiplikation

- Initialisierung
- Matrix A als Zeilen und Matrix B als Spalten in Tupeln ablegen

```
out ("A",1,<A's erste Zeile>);
out ("A",2,<A's zweite Zeile>);
:
out ("B",1,<B's erste Spalte>);
out ("B",2,<B's zweite Spalte>);
:
// Synchronisationstupel
out ("Next",1);
```

Linda – Matrixmultiplikation

- Berechnung
  - *worker* nimmt Synchronisationstupel
  - Zähler gibt an, welches Element berechnet werden soll
  - Zähler hochzählen, neues Synchronisationstupel ablegen
  - worker berechnet Ergebnis und legt es als Tupel ab.

```
in ("Next",formal Index);
if (Index < dim * dim) out ("Next",Index + 1);
i = (Index - 1) / dim + 1; // Zeile dekodieren
j = (Index - 1) % dim + 1; // Spalte dekodieren
read ("A",i, formal Row);
read ("B",j, formal Column);
out ("res",i,j,DotProduct(Row,Column));
```

- Tupel abholen und ausgeben

Linda – Bewertung

- bestechend klares Konzept
- effizienter und portabler Code möglich
- Synchronisations- und Kommunikationsmittel müssen in Tupel-Sprache übersetzt werden
  - Sowohl Synchronisation als auch Kommunikation über Tupel
  - Code dadurch schwer verständlich
  - verborgener Aufwand
- Linda liegt/lag in einer kommerziellen Version vor
- Re-Implementierung in Java: Java Spaces

DCE

- RPC middleware von Digital Equipment Corp.
  - → Lehrbuch von Mühlhäuser / Schill.
- besonders mächtig
  - Komfortabler RPC
    - z.B. IDL zur Verbindung verschiedener Programmiersprachen
  - Threads
  - Global directory service
  - Sicherheitssystem mit Verschlüsselung / Authentifizierung
- besonders umfangreich
- besonders komplex und unhandlich
- → Wenig gebräuchlich

DCE – Zellen und Threads

- System in **Zellen** aus mehreren Maschinen organisiert
  - getrennte Administration
- Zugriffe innerhalb von Zellen konventionell
- Zugriffe über Zellengrenzen über definierte Eingangspunkte
  - verstecken der inneren Struktur
  - Sicherheitsprüfungen an den Eingangspunkten
- DCE bietet ein Threads-System
  - Abbildung auf Threads des lokalen BS
  - oder als User-level Threads

DCE – RPC

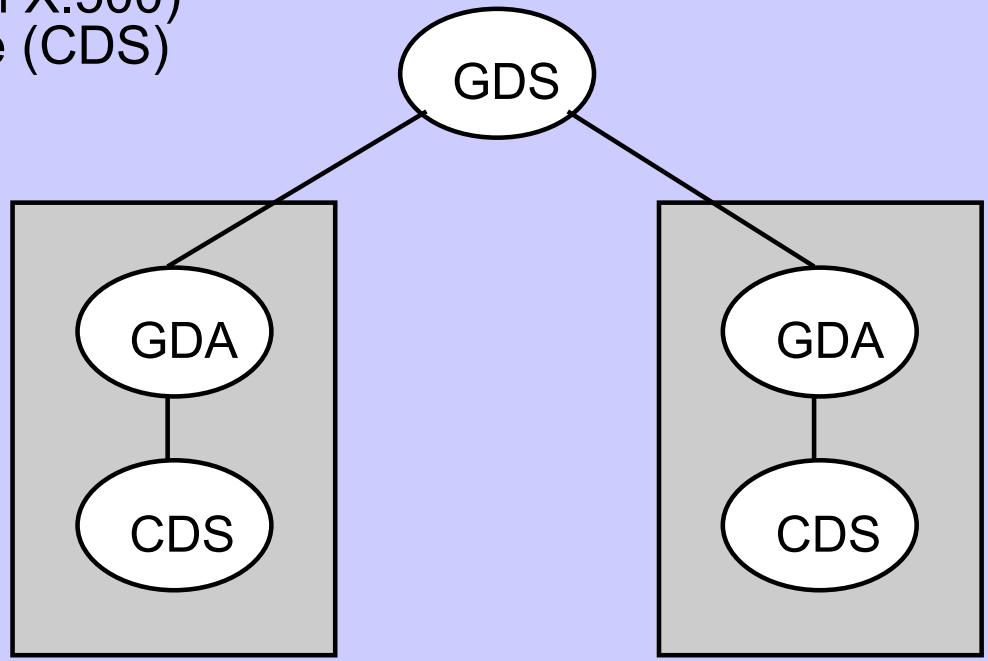
- zusätzliche Dienste
  - automatisches Lokalisieren und Binden an einen Server
  - Nachrichtentransport mit Fragmentierung
  - Datentypkonversionen zwischen heterogenen Maschinen
  - Verbindung zwischen Programmen in unterschiedlichen Programmiersprachen
- Schnittstellendefinition mit IDL (Interface definition language)
  - Schnittstelle bekommt eindeutige ID (UUID)
    - 128-bit Nummer mit Ort und Uhrzeit der Erzeugung
- Registrierung der Schnittstelle bei einem Cell Directory Server möglich
  - Anfragen an Directory Server
  - Bekanntgabe der Adresse der Maschine X, die Interface bereitstellt
  - RPC-Dämon auf Maschine X liefert den Port

DCE – Time Service

- Synchronisierung der Uhren durch Berechnen des Durchschnitts aller Zeitintervalle
  - momentan angezeigte Zeit innerhalb der Unsicherheitsgrenzen (Drift der Uhr)

DCE –Directory Service

- Directory Service vergibt Namen, die Ressourcen global innerhalb DCE lokalisieren
- Auffindung von Ressourcen
  - Global Directory Service GDS (nach X.500) kooperiert mit Cell Directory Service (CDS)
  - CDS lokalisiert Ressourcen innerhalb einer Zelle
  - Global Directory Agent GDA verbindet CDS mit GDS
- X.500 Name besteht aus hierarchisch angeordneten Komponenten
  - Land (C), Organisation (O), Organisationeinheit (OU), Nachname (SURNAME), Titel (TITLE), …
  - `/C=US/O=OSU/OU=CIS/home/kuechlin/myfile`



DCE – Sicherheitsdienst

- Sicherheitskonzept basiert auf **Kerberos** System von MIT
  - alle Netzwerkzugriffe (RPC) durch verschlüsselte Berechtigungen validiert
  - Zugriff auf individuelle Ressourcen durch Access Control Lists (ACL) geregelt
  - ACL = Liste von Einträgen von Zugriffsberechtigungen
- Teilnehmer
  - Einheit (Benutzer oder Prozess), die sicher kommunizieren will
  - Jeder Teilnehmer hat eindeutige ID (UUID - unique user ID)
- Authentifizierung
  - Bestimmen der Identität eines Teilnehmers
  - z.B. Verifizieren des Passworts
- Autorisierung
  - Gewähren (oder Verweigern) von Zugriffen auf Ressourcen
  - z.B. Nachschlagen in einer ACL

DCE - Sicherheitsdienst

- Jede DCE Zelle hat einen Sicherheitsdienst mit einem Sicherheitsserver
  - Sicherheitsserver hat sichere Datenbank mit Schlüsseln, Passwörtern, …
- Sicherheitsdienst benutzt geheime Schlüssel, um Ausweise zu verschlüsseln
  - Ausweise werden an Nachrichten angehängt, um Authentifizierung und Autorisierung zu ermöglichen
- Verschlüsselung von Nachrichten durch Funktion

```
C = E(Nachricht,Schlüssel)
```

- Entschlüsselung durch Funktion

```
Nachricht = D(C,Schlüssel)
```

- kurz

```
Ciphertext = {Plaintext}key
```

DCE - Sicherheitsdienst

- mögliche Angriffe
  - Berechnung des Schlüssels
    - Lange Schlüssel besser als kurze
  - gefälschte verschlüsselte Nachrichten
    - z.B. mit Aufforderung, ein Passwort einzugeben
  - Wiedereinspielen alter abgehörter Nachrichten
- DCE benutzt geheime Schlüssel
- DCE stempelt Ausweise mit einer Verfallszeit
  - alte Ausweise werden schnell ungültig

DCE – Sicherheitsdienst

- Ticket
  - um sich gegenüber einem anderen Server auszuweisen (*authentication*)

```
ticket =    Server-Name,
            {session-key,client,exp-time,message-id}
            Server-Key
```

- Privilege Attribute Certificate (PAC)
  - zeigt zusätzlich Gruppen- und Organisationmitgliedschaft an
    → Gewährung der Rechte aus ACLs
  - Sessionkey bezeichnet die mit dem Ticket eröffnete Sitzung
  - Ticket gilt nur für diese Sitzung und bis es verfällt
- Authenticator ist ein Ausweis

```
a = {sender, MD5-checksum, timestamp} Key
```

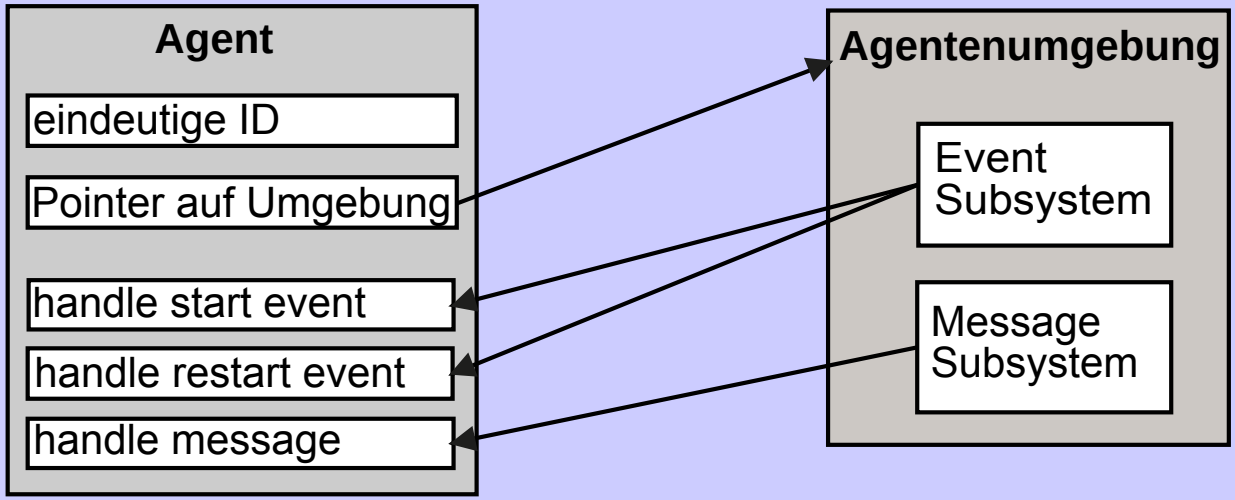
  - Klartextnachricht N kann angefügt werden
  - MD5(N) ist eine 128 bit Prüfsumme
  - N kann dann nicht verfälscht werden, ohne Veränderung von MD5(N)

Systeme mobiler Agenten

- Software-Agenten: aktive Software mit folgenden Eigenschaften:
  - **Autonomie**
    - Kapselung eines Zustands
    - Treffen von Entscheidungen aufgrund dieses Zustands ohne Eingriff durch Menschen oder andere Agenten
  - **Zielorientierung**
    - Agenten ergreifen die Initiative zur Aufgabenerfüllung
  - **Kommunikation**
    - Agent kann über ein Protokoll mit anderen Agenten kommunizieren
  - **Beständigkeit**
    - Agent darf durch äußere Einflüsse nicht endgültig verschwinden
- Mobile Agenten
  - zusätzliche Eigenschaft: Mobilität
- Bsp: IBM Aglets

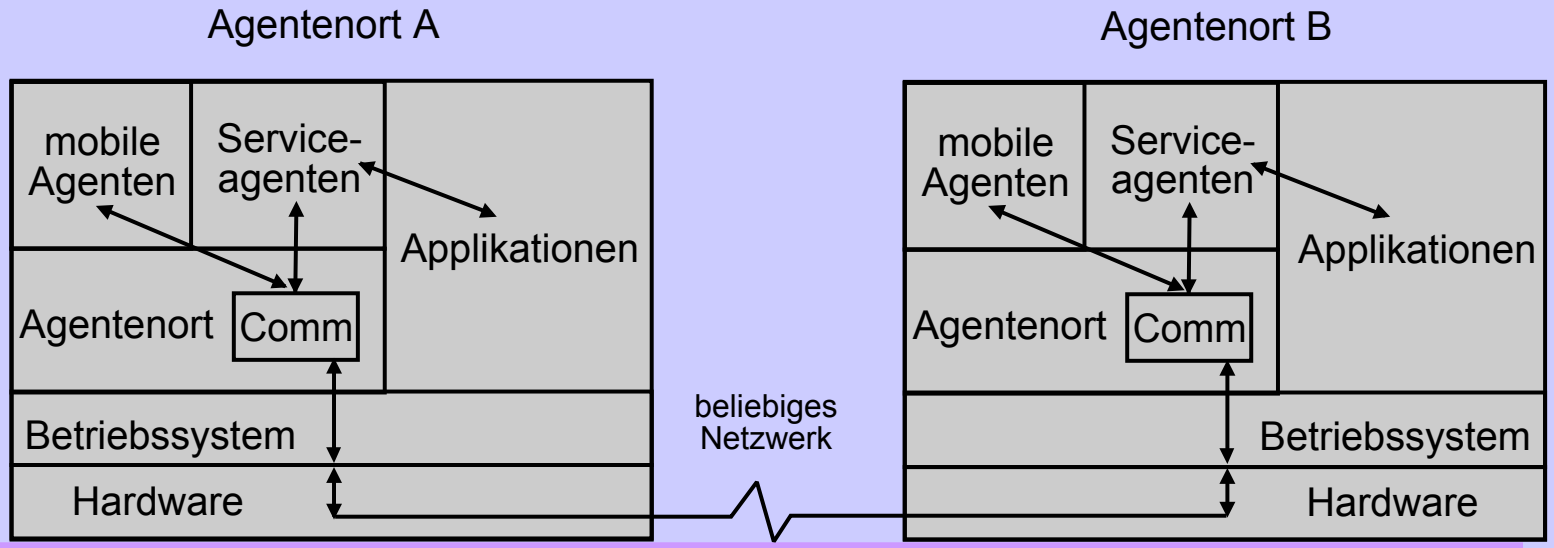
Systeme mobiler Agenten – Aufbau

- Agentensystem = Agenten + Infrastruktur (Agentenort)
- Agent
  - systemweit eindeutige Kennzeichnung (Agentenname)
  - Schnittstelle mit Funktionen
    - bei Erzeugung: *Start*-Routine
    - nach Migration: *Restart* mit lebenslanger Schleife
    - Eingang von Nachrichten: Messagehandler



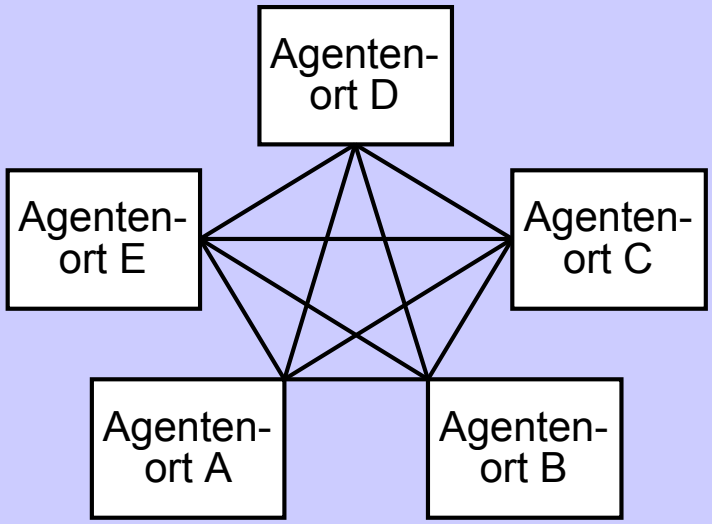
Systeme mobiler Agenten – Agentenort

- Umgebung für Agenten nennt man *Agentenort* oder *Platz*
  - Anbieten von Rechenzeit und Speicher
- Mindestanforderung an Funktionen
  - zur Generierung eines neuen Agenten
  - zur Kommunikation zwischen Agenten
  - zum Versand von Agenten an andere Orte
- Einordnung im ISO/OSI-Schichtenmodell
  - Agentenort auf ISO-Schicht 6
  - Agenten auf Applikationschicht (Schicht 7)

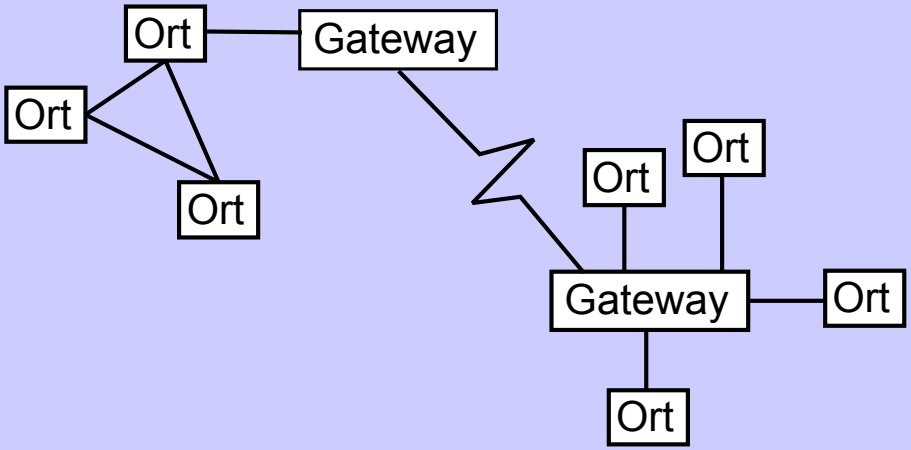


Systeme mobiler Agenten – Agentenort

- Verschiedene Verbindungsstrukturen von Agentenorten.
  - (**A**) Kleine Systeme: Vollständige Verknüpfung
  - (**B**) Verbindungsstrukturen sind nur logischer Natur, die physikalische Netzwerkverbindung kann ganz anders aussehen



A



B

Systeme mobiler Agenten – Agentenkommunikation

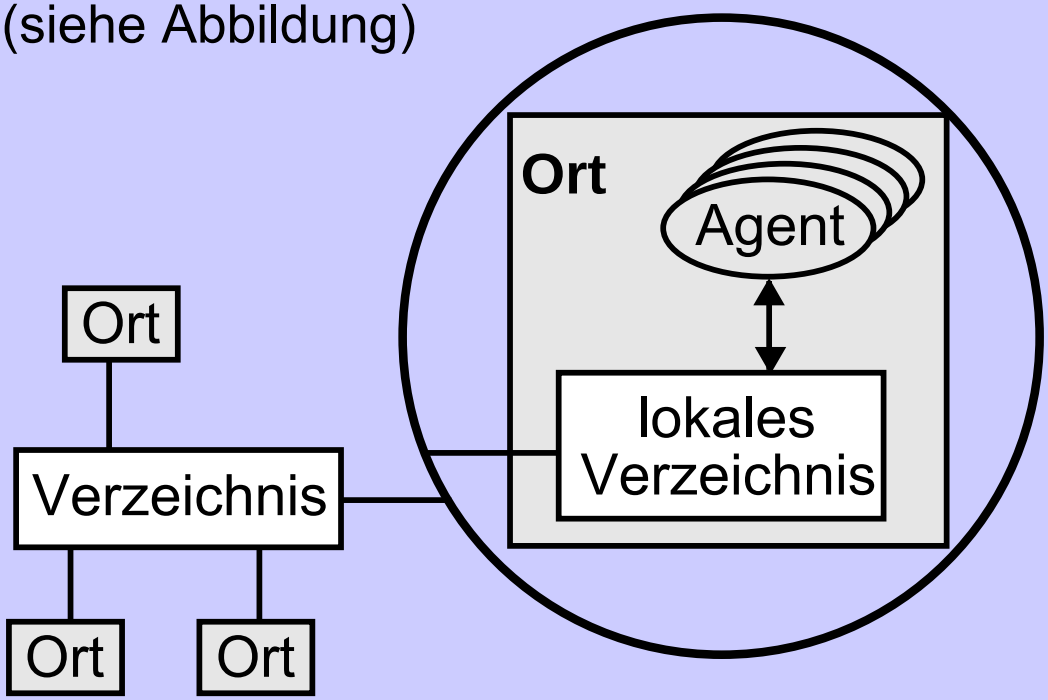
- Autonomie → Agenten kommunizieren nie direkt miteinander
  - Interaktion nur mittels des Agentenorts
- Agentensystem legt Format der Nachrichten fest, mit denen sich Agenten unterhalten
  - Gemeinsamer Standard: Agenten lassen sich auch in fremden Agentensystemen verwenden.

Systeme mobiler Agenten – Agentenmigration

- Anforderung: transferierter Agent muss am Zielort seinen Ablauf (sein Leben) wieder aufnehmen können
- starke Migration
  - Agent wird inklusive seines Zustands eingefroren und übertragen
  - Übertragung von Programmtext, Heap, Programcounter und allen Stacks
  - Problem
    - Offene I/O-Kanäle
    - Übertragung teuer
- schwache Migration
  - beim Start am Zielort wird eine bestimmte Routine aufgerufen
  - für Programmierer aufwändiger
  - zu übertragende Datenmenge geringer
  - Synchronisation nach Transfer implizit festgelegt → Verhalten des Agenten leichter durchschaubar

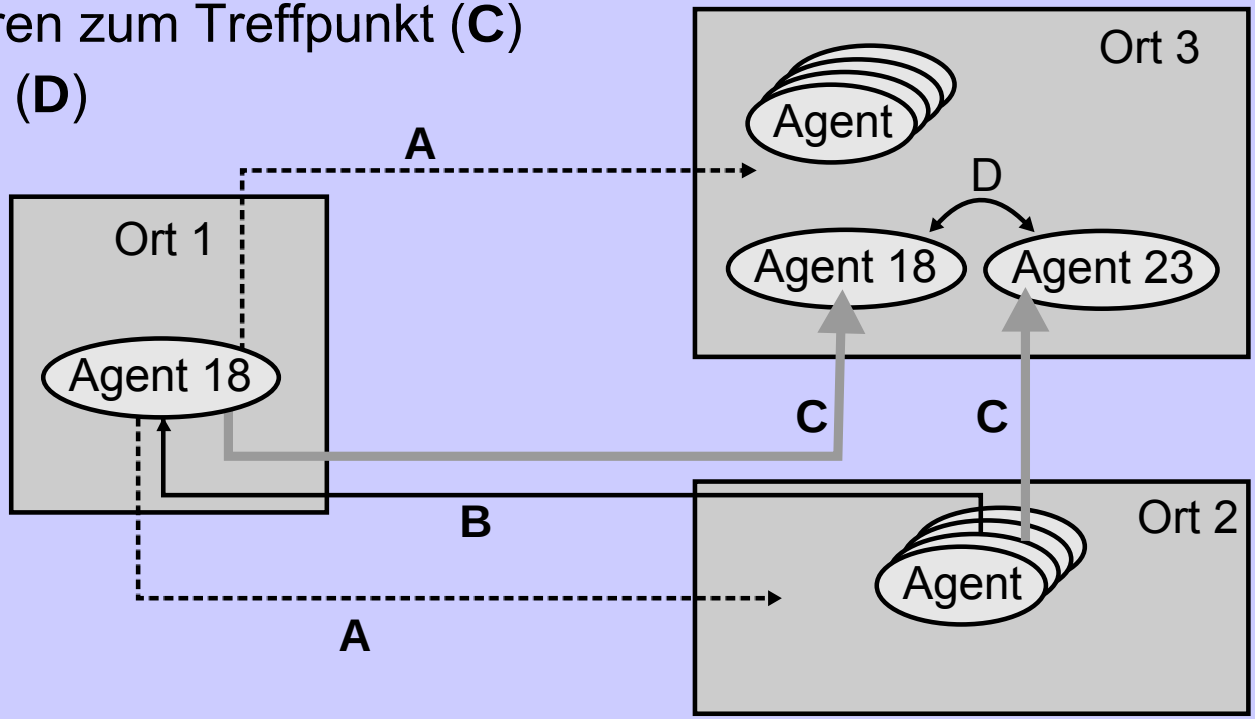
Systeme mobiler Agenten – Verzeichnisdienst

- Verzeichnisdienst: Ermittlung des Ziels eines Transfers
  - nicht unbedingt als Dienst des Agentenortes
  - Alternativ
    - zentraler Verzeichnisdienst (siehe Abbildung)
    - System mit Broadcasts (siehe nächste Folie)



Systeme mobiler Agenten – Verzeichnisdienst

- Broadcast Lösung
  - Anfrage von Agent 18 wird an alle Agentenorte gesendet (**A**)
  - Passende Agenten (Agent 23), antworten mit Namen und Agentenort als Treffpunkt (**B**)
  - Beide Agenten migrieren zum Treffpunkt (**C**)
  - lokale Kommunikation (**D**)



Systeme mobiler Agenten – Agentenprogrammiersprachen

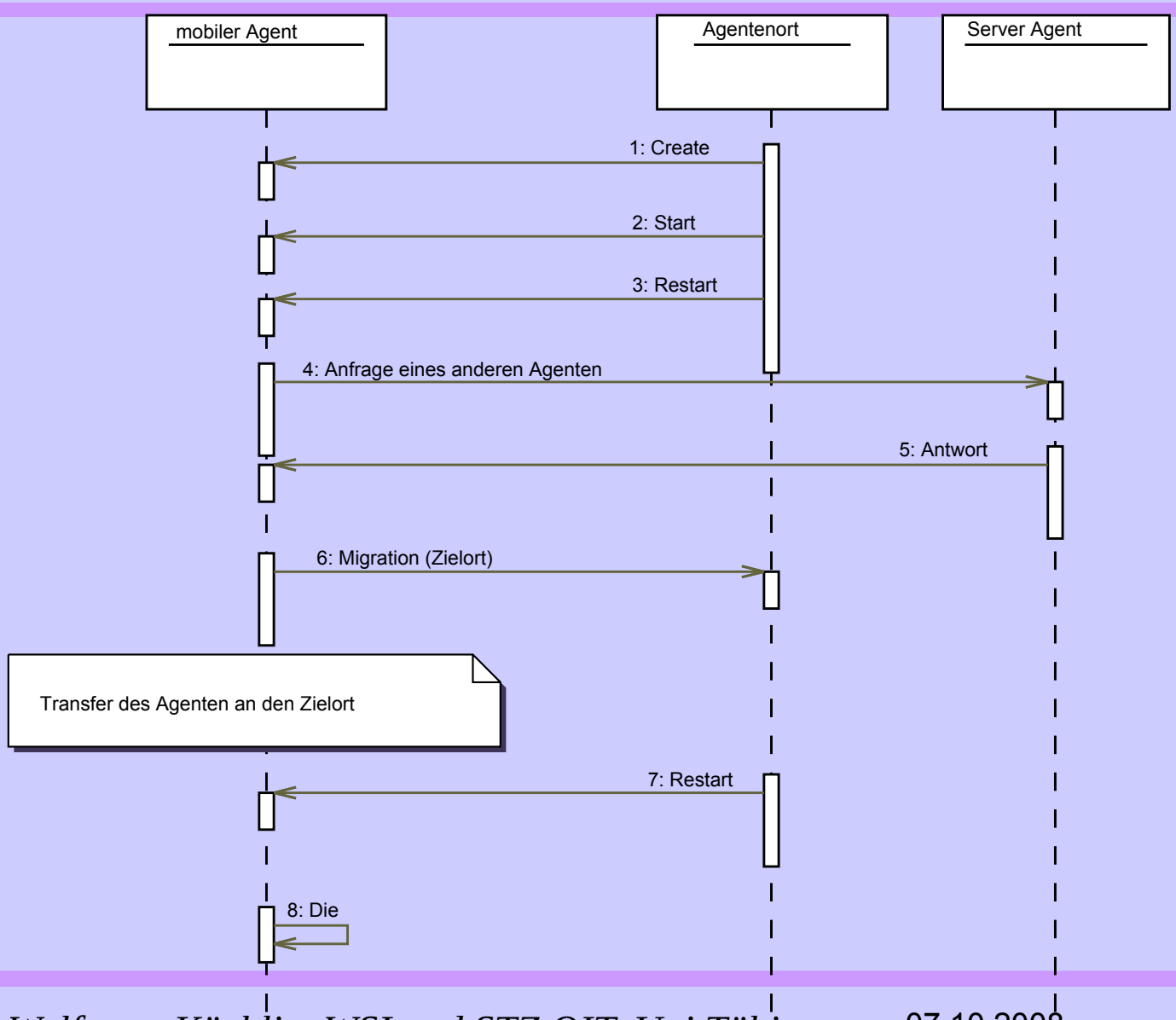
- jede Programmiersprache möglich
- Einsatz in heterogenen Umgebungen
  - plattformunabhängige Sprachen → Java
- wichtig: *late binding*
  - Code wird erst zu dem Zeitpunkt in die Laufzeitumgebung geladen, wenn er tatsächlich benötigt wird
  - bei Java objektklassenweise
  - Agenten können auf Funktionen zurückgreifen, die am Ort ihres Starts gar nicht vorhanden sind

Systeme mobiler Agenten – Lebenslauf eines Agenten

- Beispiel einer Aufgabe
  - alle Orte im lokalen Netzwerk besuchen und dort Fehlermeldungen der Applikationen einsammeln

```
Neustart:
  hole Liste aller Agentenorte im lokalen Netzwerk
Start nach Migration:
  Falls am Heimatort
       liefere Fehlermeldungen ab
       stirb
  sonst
       hole Fehlermeldungen
       füge sie zur Liste der Fehlermeldungen dazu
       Ziel = Eintrag aus Liste
       lösche Ziel aus Liste
       gehe zu Ziel
```

Systeme mobiler Agenten – Lebenslauf eines Agenten



Systeme mobiler Agenten – Bewertung

- Transaktionszeit
  - Beispiel: Flugbuchung
  - bei Agenten Netzwerkübertragung nicht mehr im kritischen Pfad
- Skalierbarkeit
  - Agenten unterstützen Verteilung von Informationssystemen in heterogenen Umgebungen
  - Administration von verteilten Systemen mit Agenten
- Personalisiertes Serververhalten
  - Beispiel: Preisvergleich von verschiedenen Anbietern mit Agenten
  - Umsetzung auch ohne Agenten möglich aber bei höherer Netzlast
- Netzwerkbelastung
  - Nachteil: Es werden nicht nur Daten sondern auch Programme übertragen
- Unterstützung mobiler Clients
  - möglichst kurze Verbindungsdauer bei PDAs oder Mobiltelefonen

Systeme mobiler Agenten – Verwendungsmöglichkeiten

- Workflow Modellierung
  - Teilschritte einer *Business Logic* in Agenten
  - Objekte der Business Logic als Agenten
- Middleware
  - flexiblere Aufteilung der Aufgaben zwischen Client und Server.
  - Beispiel: Agent als Client auf PDA/Handy kann veranlassen Teile der Verarbeitung auf einen leistungsfähigeren Server auszulagern
- Administration von verteilten Anwendungen
  - Verteilung von Softwareupdates
  - Fehlermeldungen lokal am Ort des Auftretens filtern und auswerten
- Mobile Applikationen
  - Gruppe kooperierender Agenten bildet eine mobile Applikation
  - Beispiel: Textverarbeitungsprogramm
    - Agenten für die Benutzeroberfläche
    - Agent für Dokumentenverwaltung
    - frei verfügbare Agenten für Rechtschreibprüfung, Trennhilfe oder Drucken