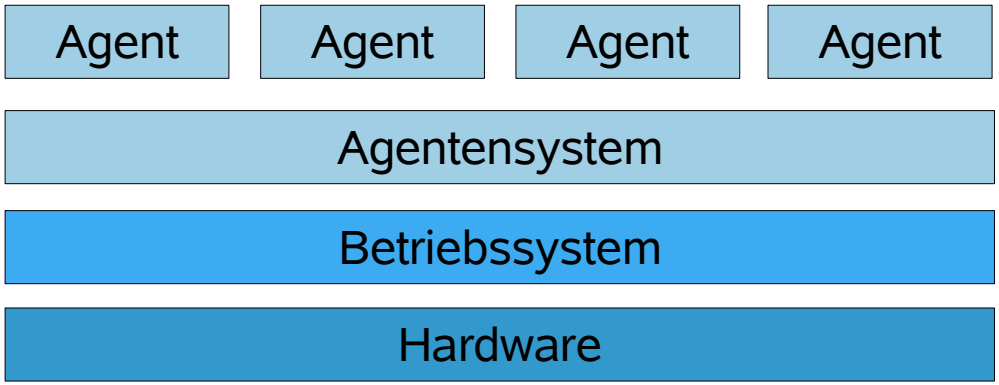
Agentensysteme und Implementierungen

Agentensystem als Anwendung

- Agentensystem als Anwendung
  - bedient sich einer (üblichen) Programmiersprache
  - nutzt Eigenschaften der Programmiersprache und der Umgebung
  - bietet eine neue Umgebung auf Basis der bestehenden



  - Erzeugung einer Bibliothek

Agentensystem als Anwendung

- Vorteil
  - Gewohnte Programmierumgebung
  - Leicht zu implementieren
  - bessere Interoperabilität mit existierenden Anwendungen
- Nachteil
  - Höherer Aufwand: Speicherverbrauch, Performanz
  - Keine originär neuen Eigenschaften
    - Beispiel Migrationsarten: Java kann Ausführungsposition nicht wiederherstellen
    - Schwache Migration ist eine Kombination aus existierenden Mechanismen
    - Starke Migration nur unvollständig oder durch Modifikation der Umgebung

Programmierung von Agenten

- Nutzung einer Standardsprache (z.B. Java)
  - also einer solchen, die von selbst keine spezifische Unterstützung für Agenten bietet
  - Eigenschaften von Agenten über Bibliotheksfunktionen
    - Beispiel: Migration ist ein Methodenaufruf auf einem Bibliotheksobjekt, etwa `m_Driver.go(sDestination)`
  - Agenten sind „gewöhnliche“ Anwendungen, welche das Agentensystem als Laufzeitbibliothek nutzen

Agentenprogrammiersprache

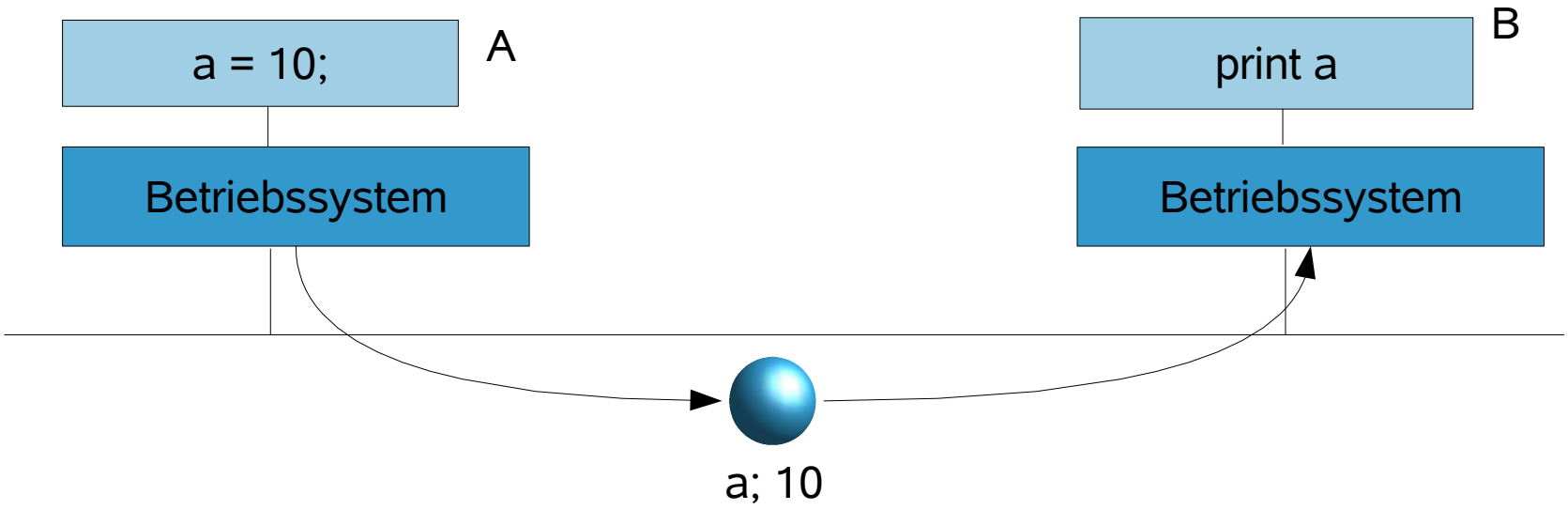
- Einführung einer speziellen Programmiersprache
  - Realisierung eines Agentensystems über die Programmiersprache
  - Konzepte der Agenten sind in der Programmiersprache manifestiert
- Vorteil
  - „Natürliche“ Programmierung eines Agenten
  - kein umständliches Aufrufen von Bibliotheksfunktionen
  - Realisierung spezieller Eigenschaften (z.B. starke Migration)
- Nachteil
  - Geringe Verbreitung
  - In der Regel geringe Chance der Interoperabilität

Agentenprogrammiersprache

- Fehlende Verbreitung der Programmiersprache bewirkt mangelnden Erfahrungsaustausch
  - und zudem kein Nutzen aus bereits entwickelten Konzepten
- Gefahr
  - Proprietäre Sprachen hängen von der Unterstützung des Herstellers ab, die ggf. eingestellt werden könnte

Betriebssystem-Agenten

- Agenten als Teil des Betriebssystems
  - Einsatz der Agenten „unterhalb“ der Anwendungsebene
  - Eigentliche Aufgaben sehen nicht unbedingt „agentenorientiert“ aus
  - Untersystem setzt Agenten ein, um Aufträge entfernt zu erledigen

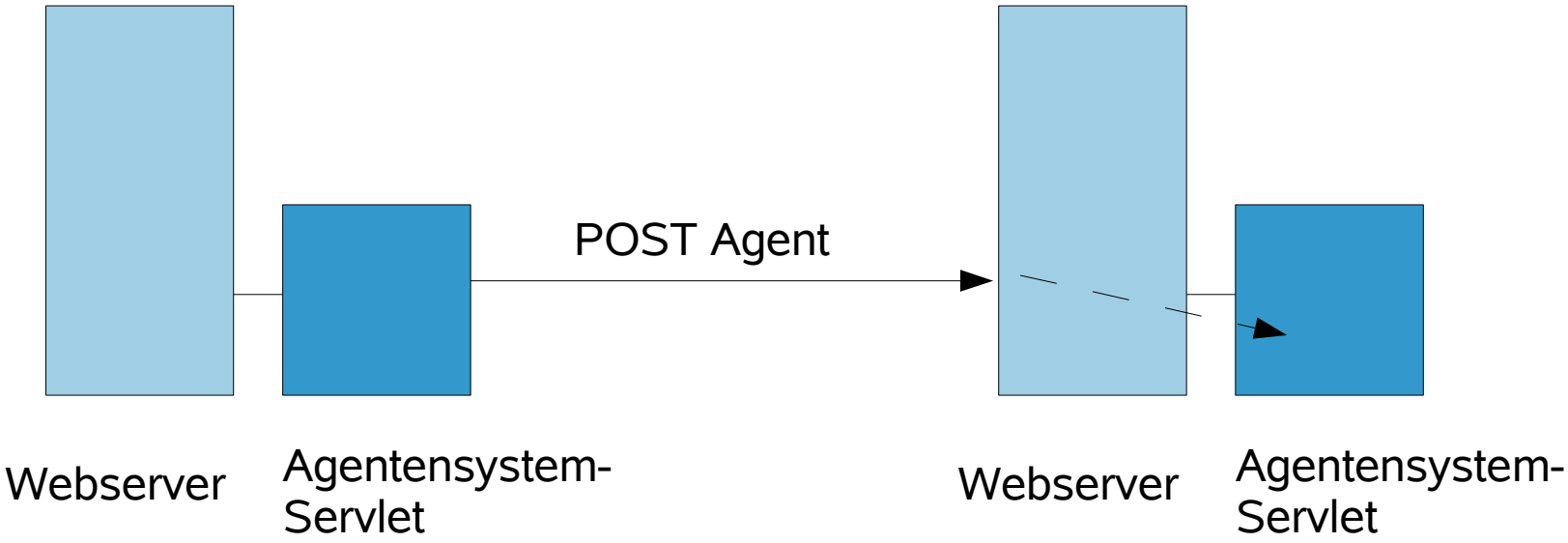


Betriebssystem-Agenten

- Eigenschaften
  - sehr einfache Agenten
  - ähnlich wie „Smart packets“
  - Microkernel mit ansteckbaren Agenten (wie Module)
- Nachteil
  - Anwendungen sind nicht unbedingt „agentenorientiert“; Betriebssystem verbirgt Agenteneigenschaft
  - Komplexe Programmierung auf Betriebssystemebene
  - Übliche Betriebssysteme sehen solche Mechanismen einfach nicht vor
  - Fragen der Sicherheit ausgeblendet

Nutzung existierender Netzdienste

- Nutzung existierender Dienste wie Webserver



  - Agenten werden über **http** übertragen
    - Spezielles Servlet verarbeitet Agenten (bietet Ausführungsumgebung)

Nutzung existierender Netzdienste

- Vorteil
  - Verwendung existierender Infrastrukturen
  - Webserver sind weit verbreitet bzw. einfach zu installieren
  - http-Protokoll wird durch Firewalls gelassen
- Nachteil
  - Zusätzlicher Aufwand durch Webserver
  - Kommunikationsaufwand höher
  - Agentensystem kann so implementiert werden, dass es selbst http spricht, also wozu dient der Webserver?

Java

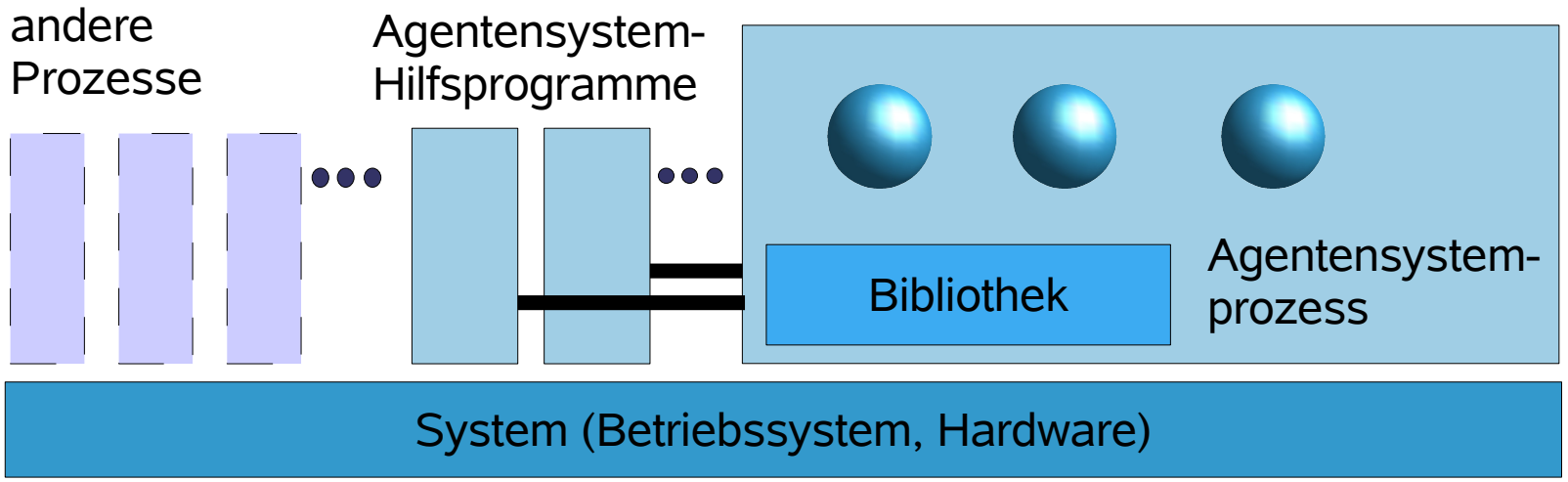
- Erschien Mitte der 1990er-Jahre
- Java beinhaltet keine unmittelbare Agentenunterstützung
  - bietet jedoch ausreichende Mechanismen an, um entsprechende Bibliothek zu schaffen
- Eigenschaften
  - objektorientiert (Agenten als spezielle Objekte)
  - Serialisierung
  - netzwerktauglich
  - Threads

Agenten in Java

- Agenten in Java
  - Implementierung kann zur Laufzeit nachgeladen werden
  - Anwendung von Threads ermöglicht die Sichtweise als eigenständig laufende Komponenten
  - Ggf. Nutzung objektorientierter Eigenschaften: Vererbung, Codewiederverwendung usw.
  - Zugriff auf Netzwerkressourcen in der Sprache abgebildet
    - leider nicht die Migration

Agenten in Java

- Java-Laufzeitumgebung ist **ein** Prozess
  - verwaltet Teilprozesse (Threads)
  - kann Klassen nachladen
  - daraus die prototypische Java-Implementierung eines Agentensystems:



Prozesse

- Agentensystem-Prozess
  - betreibt alle Agenten und jene Komponenten, zu denen die Agenten unmittelbar Zugriff benötigen (z.B. für die Kommunikation)
- In Java als einzelne Anwendung realisierbar
  - Subsysteme und Agenten als Threads (Verwendung mehrerer Prozesse erhöht den Aufwand)
- Generell auch als einzelne Prozesse realisierbar
  - dann aber wird Interprozesskommunikation benötigt
  - Gegenseitiger Schutz durch Betriebssystem gesichert
- Hilfsprogramme
  - Externe Dienste, z.B. Namensdienst, Dienstverzeichnis usw.

Namensdienst

- Modellierung von Agentenunterstützungen als Prozesse
  - Kann man mehrere Agentenumgebungen auf demselben Rechner starten? (parallel laufend)
  - Wie adressiert man eine solche Umgebung?
- Nutzung existierender Namensdienste
  - DNS
    - keine Auskunft zu Portnummern (evtl. über spezielle Resource Records)
    - Oder festlegen, dass nur ein System pro Rechner laufen darf
    - Administrativer Zugang zu einem DNS erforderlich
  - Eigener Namensdienst
    - Auf Situation perfekt anzupassen
    - aber eine weitere, mögliche Fehlerquelle
    - Zentralistischer Dienst widerspricht der Verteilungsidee von Agenten!

Mobile Agenten

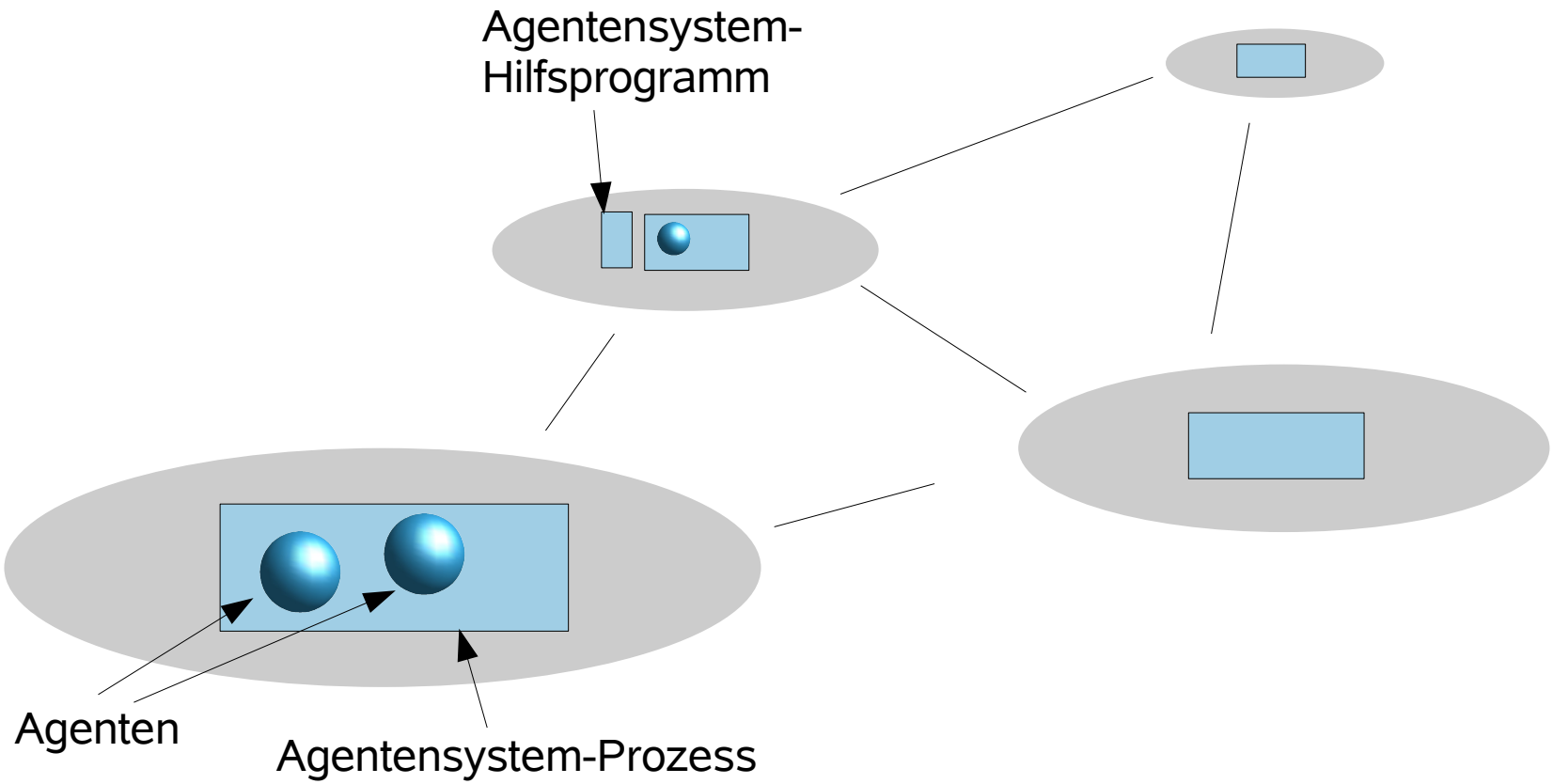
- Mobile Agenten
  - implizieren den Betrieb einen „passenden“ Agentensystems auf verschiedenen Netzknoten
  - Agent kann zwischen den Knoten wechseln
  - Agent muss überall korrekt ausgeführt werden

Das System muss gleich oder *kompatibel sein*

kann als Kriterium für die Einführung
von Agentensystem-**Typen** dienen

Mobile-Agenten-System

- Unterstützung auf verschiedenen Knoten



Agentensystem

- Geschlossenes Agentensystem
  - Begrenzte Menge an Lokationen und/oder begrenzte Menge an Agenten-„Typen“
- Offenes Agentensystem
  - Unbegrenzte Menge an Lokationen und/oder Agententypen

vergleichbar: Geschlossene/offene Agenten**anwendung**

(auf Anwendungsebene, nicht auf Infrastrukturebene)

Agentensysteme

Auswahl von Agentensystemen

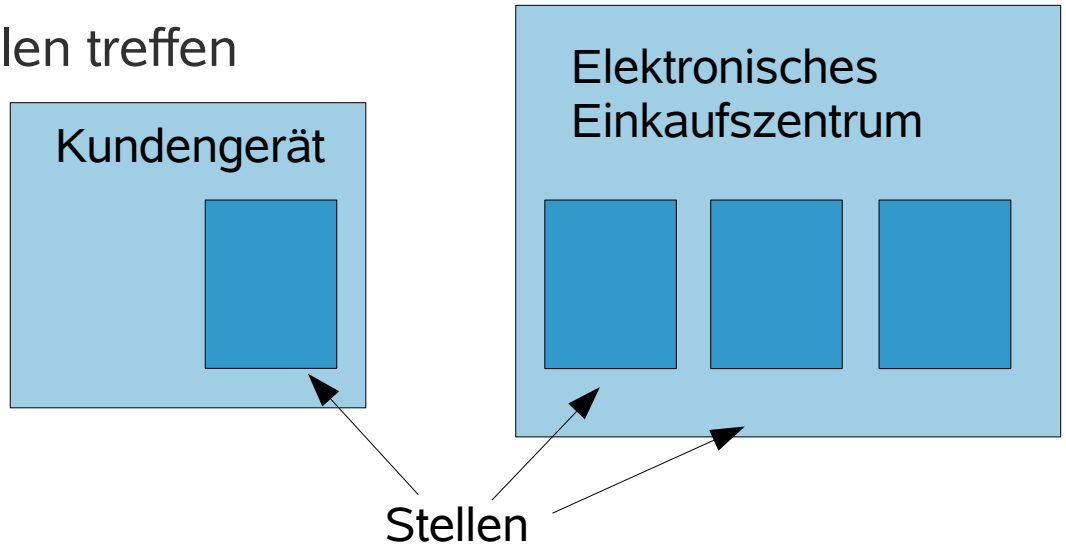
- Intelligente Agenten
  - Java Agent Template
  - Jack
- Theoretische Ansätze
  - Mobile Ambients
- Mobile Agenten
  - Telescript (General Magic)
  - Messengers (Tschudin)
  - D'Agents (Dartmouth College)
  - Mole (Hohl)
  - Aglets (IBM)
  - Ara (Peine)
  - Grasshopper (IKV++)
  - ffMain (Lingnau)
  - AMETAS (Zapf)

Telescript

- „Urahn“ der Mobile-Agenten-Systeme
- Hersteller: General Magic
- Prägte die Nomenklatur zahlreicher Agentensysteme
- Patenteinreichung im Jahre 1993 unter der Nummer 5,603,031 beim US-Patentamt (http://www.uspto.gov/patft/index.html)
  - *„System and method for distributed computation based upon the movement, execution, and interaction of processes in a network“*
  - jedoch ist General Magic nicht mehr im Geschäft...

Telescript

- Places
  - Telescript-Welt ist ein „elektronischer Marktplatz“
  - Heimat-Place auf eigenem Gerät, Places der anderen Marktteilnehmer bei einem Dienstanbieter
  - im Deutschen ist der Begriff „Stelle“ gebräuchlich
- Einsatz von Places
  - Agenten können sich an Stellen treffen
  - Stellen haben Semantik
    - Ticket-Büro
    - Blumenladen
    - Information

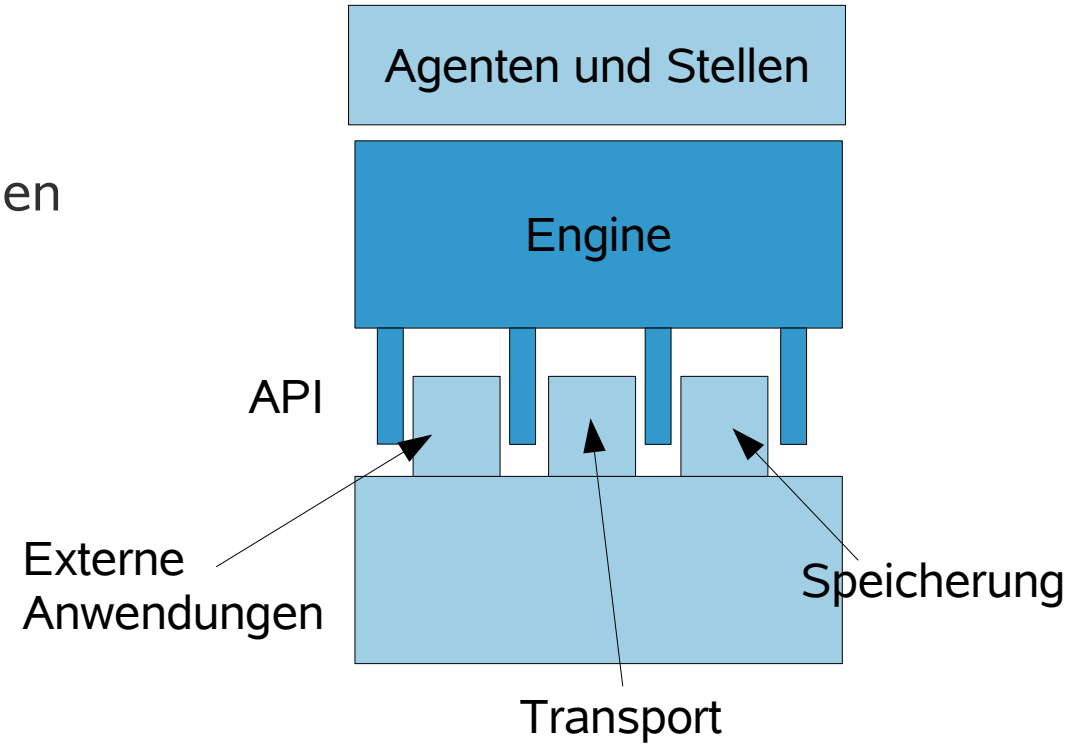


Spezialitäten von Telescript

- Spezielle Programmiersprache / - umgebung
- Stellen beinhalten einen Agenten, der die Funktion dieser Stelle ausmacht
  - wird jedoch als stellenintern behandelt, also kein eigentlicher, eigenständiger Agent
- Stellen und Agenten sind in der Telescript-Sprache geschrieben
- Telescript-Engine
  - integriert in „MagicCap personal intelligent communicator“
  - für übliche Rechner (PC, Mainframes) in C++ geschrieben

Telescript-Engine

- Speicherung
  - Persistenzdienst für die Stelle und die Agenten
- Transport
  - Zugang zu den Kommunikationskanälen zum Versenden und Empfangen von Agenten
- Externe Anwendungen
  - Schnittstelle zu nativen Anwendungen



Telescript-Sprache

- Objektorientiert
  - Klassen mit Hierarchie
- Dynamisch
  - Klassenladen während der Ausführung (wie in Java)
  - für offene Anwendung nützlich, sogar erforderlich
- Persistent
  - Daten werden von der Engine transparent gesichert, **auch der Programmschrittzähler!**
  - Nach Absturz kann man wieder an der Stelle beginnen, an der die Ausführung endete

Telescript-Sprache

- Portabel
  - Interpretierung statt direkter Ausführung
- Sicherheit
  - Telescript-Engine interpretiert Telescript-Programm; keine Befehle zum direkten Zugriff auf Speicher
  - Credentials zur Kontrolle des Zutritts zu Stellen
  - Permits zur Kontrolle der möglichen Aktionen und des Zugriffs auf Ressourcen
- Kommunikationszentriert
  - Spezielle Befehle für Navigation, Transport, Authentifikation, Zugriffskontrolle usw.

Telescript-Agenten

- Eigenständige Prozesse, können andere Agenten nicht beeinflussen
- **go** zum Wandern von einer Stelle zu einer anderen
  - Ticket erforderlich: beinhaltet Ziel, Reisedauer, Reiseart (z.B. drahtlos)
  - „Als Konsequenz des go wird die nächste Anweisung im Agentenprogramm an der Zielstelle ausgeführt, nicht mehr an der Ursprungsstelle [...] Es ist wie Magie.“ Telescript White Paper

Starke Migration!

Kommunikation zwischen Agenten

- Lokale Kommunikation über **meet**
  - Agenten treffen sich an einer Stelle
  - Agent A bittet B um eine Kommunikation („meet“)
  - B stimmt zu oder lehnt ab
  - Bei Zustimmung: Austausch von Referenzen, gewöhnliche objektorientierte Kommunikation
- Entfernte Kommunikation über Verbindungen (connections)
  - Ähnlicher Vorgang wie im lokalen Falle
  - Austausch von Telescript-Objekten

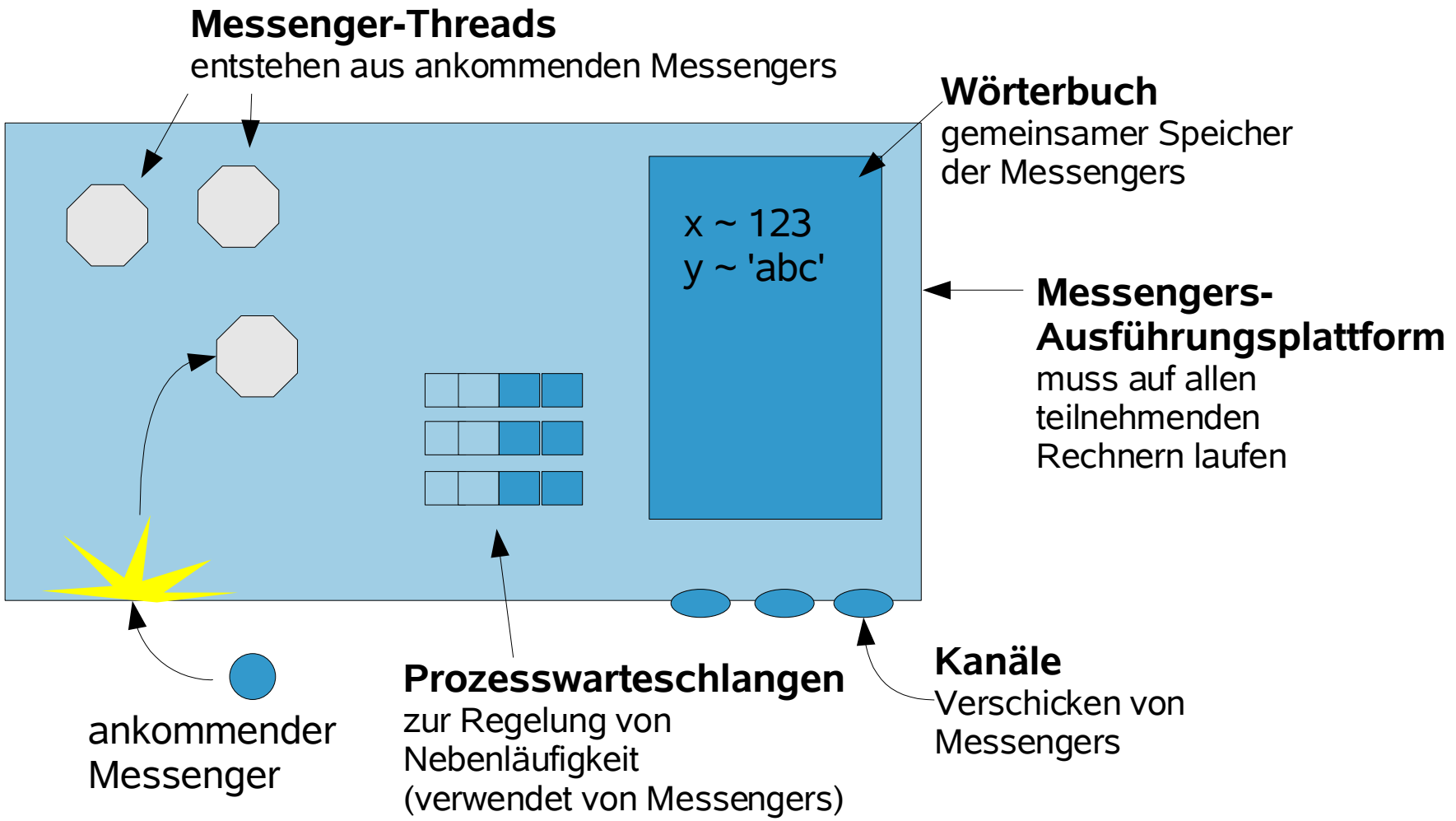
Schicksal von Telescript

- Pläne: Telescript im Einsatz in Szenarien für elektronische Marktplätze
  - Hinweis auf kommerziellen Einsatz durch Patentanmeldung
- Jedoch keine Unterstützung aus der Agentengemeinschaft
  - zu früh?
- Proprietäres System
  - kostenlose Alternativen, Java
- Voyager als Nachfolger, Java-basiert
- Keinerlei Verweise mehr auf den Webseiten
  - GM hatte sein Engagement sehr bald eingestellt
  - Jedoch noch immer gerne zitiert in wissenschaftlichen Artikeln

Messengers

- Eingebracht als Idee von Christian Tschudin, 1993
  - „MØ“ (M-null)
  - Telescript ein Jahr später
- Damals bezeichnet als
  - „Wurm-ähnliche Programme,die zwischen Netzwerkknoten ausgetauscht werden“
- Messengers als Bestandteile verteilter Betriebssysteme
  - neues Kommunikationsparadigma

Messengers-Architektur



Messengers-Programmierung

- Ansprechen von Prozesswarteschlangen und Kanälen sowie globalen Variablen (im Wörterbuch) durch **Schlüssel**.
- Abstrakte Definition erforderlicher Programmierelemente
  - u.a. arithmetisch/logische Befehle
  - ggf. auch native CPU-Befehle
  - außerdem folgende spezielle **Primitive**

Primitive	**Beschreibung**
key()	Erzeugung von Schlüsseln
get(k:key), set(k:key, v:value)	Lesen und Schreiben globaler Variabler
submit(k:key, m:msgr_descr)	Senden von Messenger über Kanal
chain(m:msgr_descr)	Neues Verhalten setzen
enter(q:key)	Warteschlange betreten (am Ende)
leave	Warteschlange verlassen (am Kopf)
stop(q:key)	Schlange stoppen
start(q:key)	Schlange wieder starten

Messenger-Implementierung

- MØ-System
  - ähnelt sehr stark Postscript, jedoch ohne Grafikbefehle
  - Kurz- und Langform der Befehle verfügbar
    - Kurzform sehr kompakt; einbuchstabige Befehle (Lesbarkeit?)
  - kann einfach über UDP-Datagramme transportiert werden
- Beispiel anhand des *Alternating Bit Protocol**
  - Stationärer Sende-Messenger
  - Mobiler Transfer-Messenger: Inklusive ACK-Msgr. nur 73 Byte!
- MSGR-S: auf Basis von Scheme (->VKI)

*Einfaches, dennoch nicht triviales Protokoll, Standardbeispiel

Beispiel in MØ

```
null                  # specify local submit
[ ;                   # Q: random start queue
  "_dat _log"         # C: the code
  "hello world"       # D: the data
] _ctm                # make this a messenger packet
$                     # submit it
```

MØ ist stackorientiert
Operatoren erwarten
Werte auf dem Stack

_ctm: (convert-to-messenger)
Wandelt ein Feld mit drei Elementen in einen Messenger um, der in einem String geliefert wird.
Erstes Element: Index auf die Empfangswarteschlange
Zweites Element: Messenger-Code
Drittes Element: Messenger-Daten

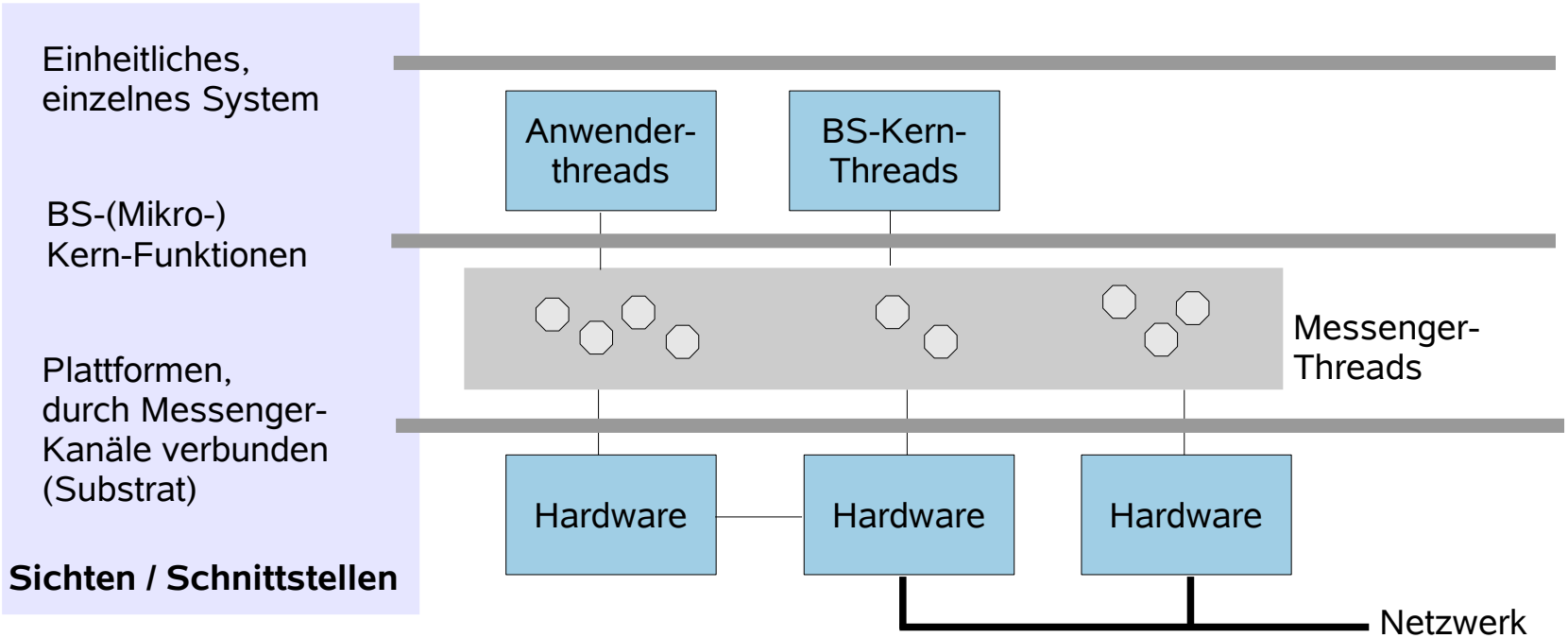
_dat:
Jeder Prozess hat ein _dat-Eintrag im lokalen Wörterbuch (Dictionary-System). _dat ist assoziiert mit dem Datenstring des Messengers

_log:
Optionale Debugging-Hilfe; gibt eine Nachricht mit Zeitstempel in der Logdatei der Plattform aus

$ (submit):
Eine der zentralen Messenger-Operationen; sorgt dafür, dass ein neuer Messenger lokal oder entfernt gestartet wird.
Erstes Argument = Index des Kanals, über den der Messenger zu schicken ist (null=lokal)
zweites Argument: Zu sendende Daten

Einsatz von Messengers

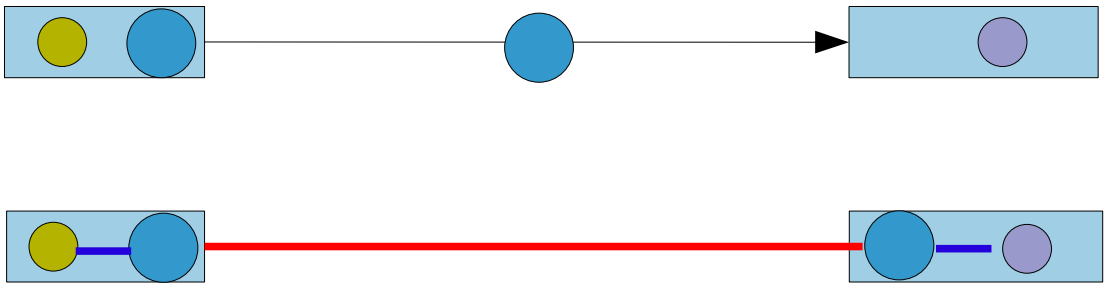
- Messengers als Zwischenschicht in einem verteilten System



Messengers an der Nahtstelle zwischen User Mode und Kernel Mode;
Zugriffe auf die CPU nur über Messenger

Messenger-basierte BS

- Vorteil
  - verschiedene BS möglich, kompatibel durch Messengers
    - Ist das so? Das hängt doch nicht nur von der Kommunikation ab!
  - Messengers können Speichermanagement, Prozessmanagement implementieren
  - Kommunikationsprotokolle wesentlich vereinfacht
    - „Protokoll-freier Mikrokern“



Messengers implementieren das Kommunikationsprotokoll

D'Agents

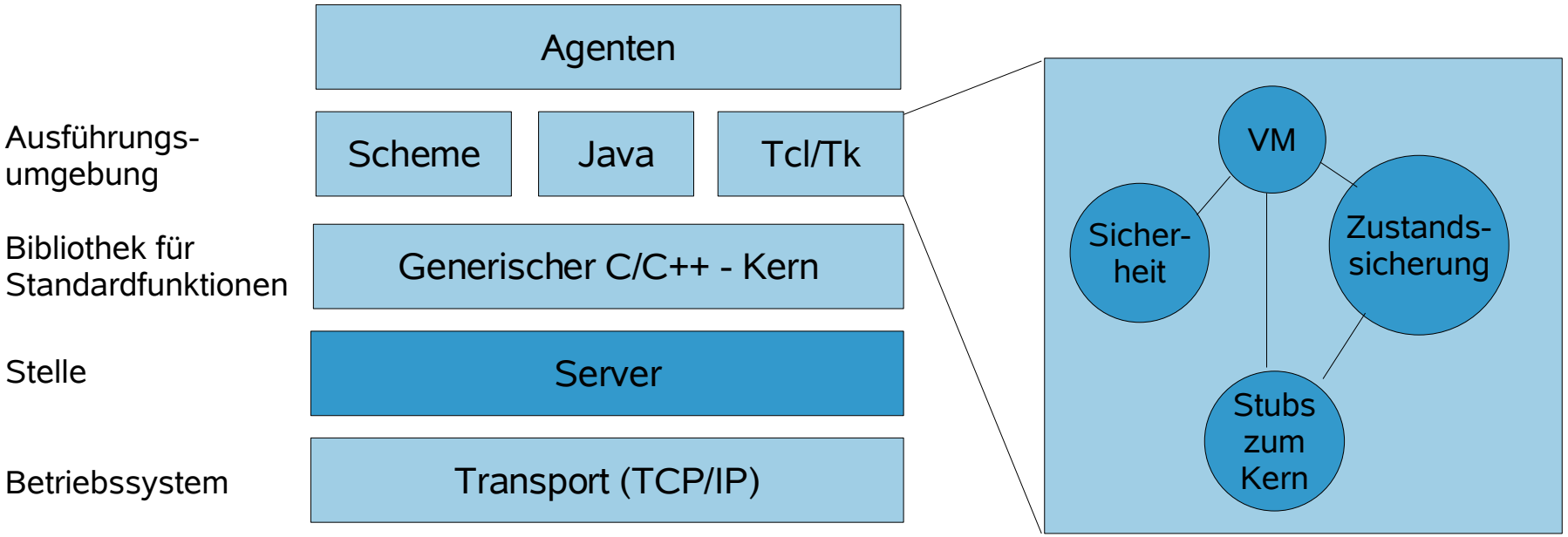
- „Dartmouth Agents“
- D'Agents 1.1 war bekannt als „Agent Tcl“
- Erweiterung von Tcl (Tool control language)
- Unterstützt drei Sprachen
  - Tcl
  - Java
  - Scheme
- **Starke Migration** in Java und Tcl

D'Agents

- Hauptmerkmale
  - Unterstützung verschiedener Programmiersprachen
    - Spezielle Ausführungsumgebungen für jede Programmiersprache
    - Ausführungsumgebungen warten in einem Pool auf eintreffende Agenten
    - Beispiel Java: Server startet mehrere Java-VMs und lässt darin jeweils einige Agenten als Threads laufen
    - Beispiel Tcl/Scheme: Server startet pro Agent eigenständigen Prozess
  - Grundfunktionen für Agenten auf niedriger Ebene
    - Adressierung eines Agenten über Rechneradresse + Agenten-ID
    - Ortsunabhängige Adressierung über Verzeichnisagenten
    - Vorteil: Hohe Performanz; Funktionalität ausreichend für die meisten Anwendungen

D'Agents-Architektur

- Server: Kernbestandteil der Agentenarchitektur
  - empfängt und versendet Agenten
  - authentifiziert Benutzer
  - regelt und kontrolliert den Ressourcenzugriff



Mobilität in D'Agents

- Starke Migration
  - leichter für den Anwender („for undergraduate programmers“)
  - jedoch:

> The system must capture enough state information to restore the agent in exactly the same control state on its new machine, and in fact, we had to modify off-the-shelf Tcl and Java interpreters to include the necessary state-capture routines.*

  - damit Interoperabilität mit anderen Plattformen bzw. mit Ressourcen anderer VMs fraglich
- Threadmigration
  - nur der ausführende Thread migriert, der Rest bleibt zurück

*D'Agents: Applications and Performance of a Mobile-Agent System; http://agent.cs.dartmouth.edu/papers/gray:spe.pdf

Daten- und Codemigration in D'Agents

- Datenmigration
  - Transparente Migration aller relevanten lokalen Variablen und des Heaps
  - Außerdem Instanzvariablen (wie in Java üblich)
- Codemigration
  - Push-Mechanismus: Die gesamte Codebasis des Agenten wird von der Ursprungslokation zur Ziellokation transferiert
  - kein On-demand-Abholen: Vermeidung von Abhängigkeiten (besonders interessant bei drahtlosen und Ad-hoc-Netzen)

Kommunikation in D'Agents

- Kommunikation zwischen Agenten
  - Adressierung auf Basis eindeutiger Bezeichner in einem servergebundenen Namensraum
- Konmmunikationsmechanismen
  - **Nachrichten**: Standard-send- und -receive-Primitive
  - **Ströme**: direkte Verbindung mit einem anderen Agenten
  - **Ereignisse**: Agent sendet Ereignisse (Events) an einen anderen Agenten, welche dieser behandelt

Zusammenfassung

- Zahlreiche Gemeinsamkeiten bei Mobile-Agenten-Systemen
  - Unterstützung mehrerer Agenten → Multiprocessing/-threading
  - Funktionsbibliothek für Standardaufgaben
  - meist interpretierte Sprachen oder plattformunabhängige Sprachen
  - selten spezifische Sprachen, sondern eher bekannte
- Migration
  - wenige Systeme mit starker Migration, meist nur schwache Migration
- Unterschiedliche Ansätze für die Kommunikation
  - Message passing; reine Strings oder komplexe Objekte
  - Methodenaufruf
  - Publish/subscribe-Modell