NET.OBJECTDAYS 2000

Tutorial 2

(09. Okt. 2000, nachmittags)

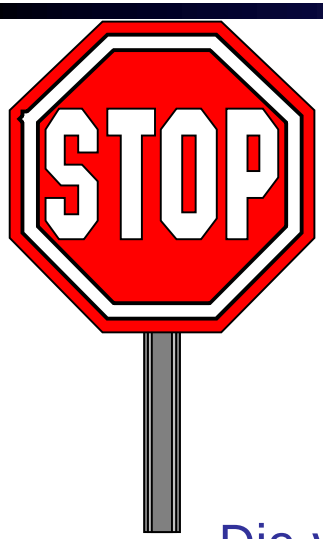
Von verteilten Objekten zu mobilen Agenten



Peter Braun, Christian Erfurth, Wilhelm Rossak
Friedrich-Schiller-Universität Jena
Institut für Informatik
Lehrstuhl für Softwaretechnik
Ernst-Abbe-Platz 1-4
07743 Jena
Tel. 0 36 41 / 94 63 01, Fax 0 36 41 / 94 63 02
E-Mail: braunpet, nce, rossak@informatik.uni-jena.de
http://swt.informatik.uni-jena.de

LEHRSTUHL SOFTWARETECHNIK

The Mobile Agent System Tracy



Die vollständigen Unterlagen finden Sie auf unserer Homepage:

http://tracy.informatik.uni-jena.de/tutorials



Übersicht zu diesem Tutorial

- Teil 1: Motivation und Einführung in mobile Agenten
- Teil 2: Funktionsweise von mobilen Agenten 
- Teil 3: Das Mobile Agentensystem Tracy 
- Teil 4: Programmierung eines minimalen Agentensystems 
- Teil 5: Mobile Agenten als neues Design Paradigma 

- Weitere Informationen 
- Zusammenfassung 



Motivation und Einführung in mobile Agenten

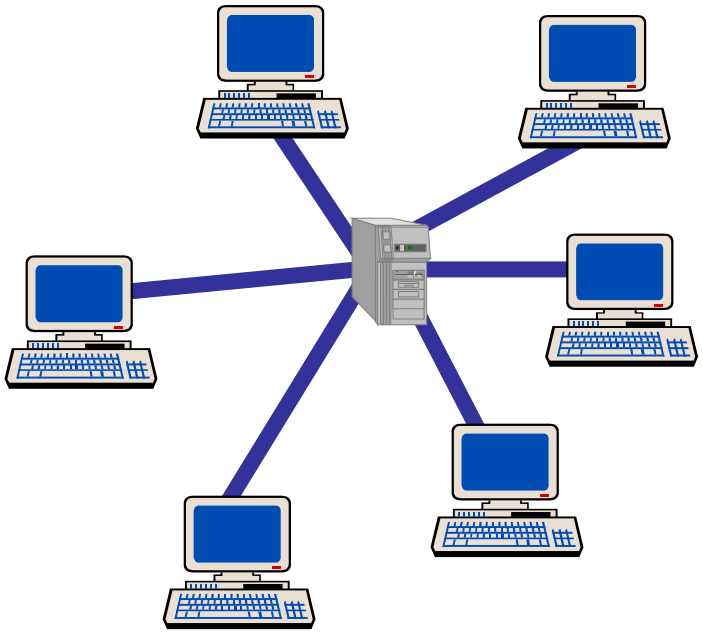


Überblick

- Motivation
- Was sind mobile Agenten?
- Programmierung verteilter Systeme
- Vorteile von mobilen Agenten
- Anwendungsmöglichkeiten



Wo stehen wir heute?



- Lokale Netzwerke
- Netzwerkqualität
  - hohe Bandbreite
  - geringe Latenz
  - hohe Verfügbarkeit
  - gleichbleibende Qualität
- Knoten
  - geringe Anzahl
  - homogene Computersysteme
  - geringe Ausfallwahrscheinlichkeit
- Gegenseitiges Vertrauen
- Programmierung
  - Client-Server Architektur
  - Ortstransparenz
  - Remote Procedure Calls
  - Remote Method Invocation



Herausforderungen der Zukunft

- Weltweites Informationssystem
- Anzahl verfügbarer Services ist um ein vielfaches größer als in traditionellen verteilten Systemen
- Unterschiedliche Rechnerarchitekturen
- Qualität der Verbindung höchst unterschiedlich
  - Bandbreite, Latenz
  - Verfügbarkeit
  - Zuverlässigkeit
- Mobilität der Benutzer
  - die Netzverbindung wird von unterschiedlichen Orten zu verschiedenen Zeiten aufgebaut; größtenteils jedoch off-line
- Knoten in verteilten Systemen sind nicht nur Server
  - sondern auch: Laptops, Palmtops, Mobile Phones, E-Books
  - und auch Kühlschränke, Autos, Wearable Computers
    - mit geringem Speicher, kleinen Prozessoren

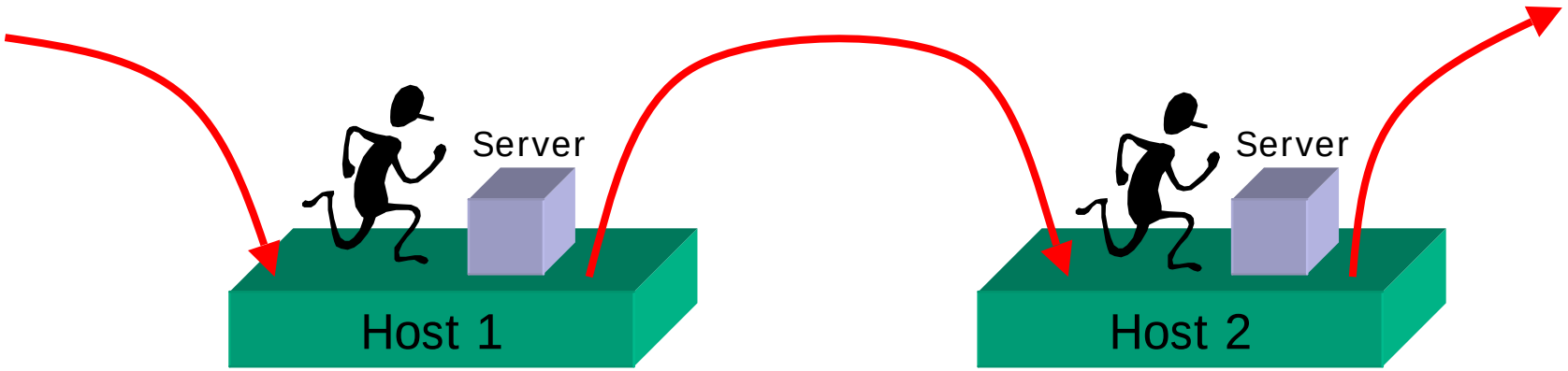


Was sind mobile Agenten?

Agent: Eine Software-Entität, die autonom im Auftrag eines Nutzers agiert und dabei mit anderen Agenten kommuniziert.

Mobiler Agent: Ein Agent, der

- von Rechner zu Rechner migriert
- in einem heterogenen Netzwerk agiert
- selbst entscheidet, wann und wohin er migrieren will





Was sind mobile Agenten?

- Mobile Agenten sind eine Form von Software Agenten
  - autonom, pro-aktiv, langlebig
  - ziel-orientiert, anpassungsfähig
- Mobile Agenten können während der Laufzeit
  - eingefroren und eingepackt …
  - … an einen anderen Rechner geschickt …
  - … und dort ausgepackt und wieder gestartet werden.
- dabei werden folgende Elemente verschickt:
  - sämtliche Daten des Agenten (lokale und referenzierte)
  - der Code (alle Klassendateien)
  - Zustandsinformationen (IP, Call-Stack, Operand-Stack)



Agenten in der KI

Definition des Begriffs Agent mit Hilfe von Eigenschaften:

- Autonomie
- Intelligenz
- Adaption
- Mobilität
- Kommunikation
- Kooperation
- Persistenz
- Zielorientierung

Es genügt, wenn einige dieser Eigenschaften zutreffen!

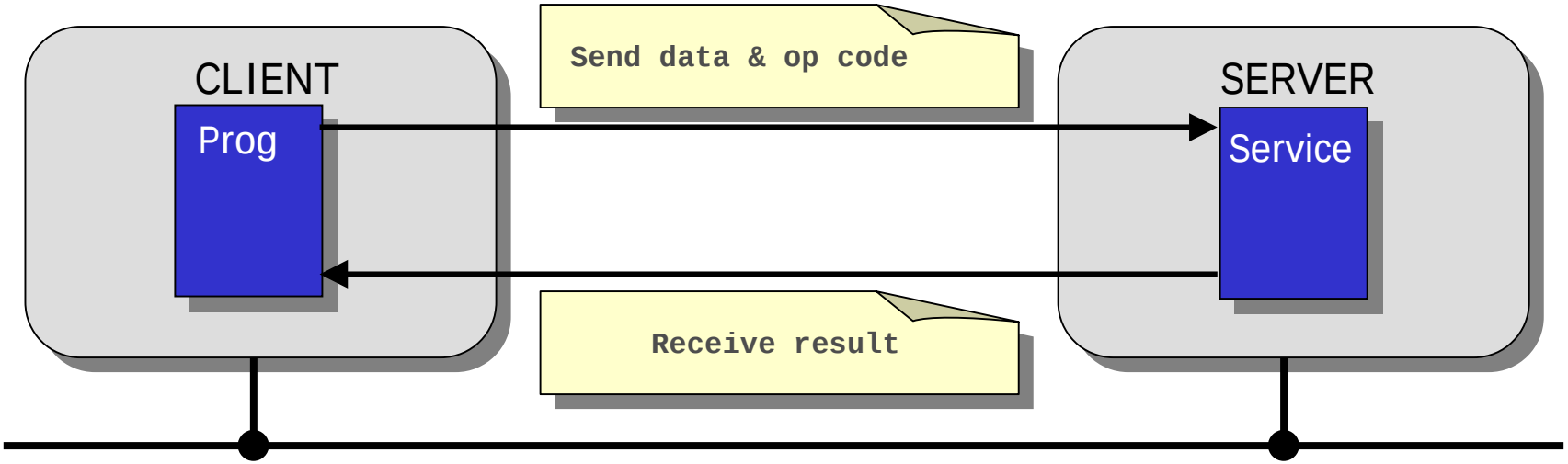


Das sind keine Mobile Agenten!

- Java Scripts / WML Script
  - Code, der zusammen mit einer HTML Seite von einem Server geladen und auf dem Client ausgeführt wird
- Java Applets
  - Applets werden von einem Server geladen (passive, pull)
  - dabei wird der Code, aber (meist) keine Daten übertragen
  - eine Netzverbindung muss ständig vorhanden sein
  - Applets unterliegen strengen Sicherheitsüberprüfungen
  - Applets „migrieren“ stets nur einmal
  - Applets sterben, wenn der Browser beendet wird
- MIDlets (Mobile Information Device Applications)
- Suchmaschinen
  - WebCrawler, WebRobots, Spiders, Worms usw.
- Jini
  - spontane Vernetzung und Veröffentlichung von Services



Programmierung verteilter Systeme



- Client Server Mechanismus
- bei einem „send“ werden nur Daten verschickt
  - ein Operationscode gibt an, wie die Anfrage zu verstehen ist
- mit „receive“ werden Daten auf dem Client empfangen
- kein Verbindungsaufbau

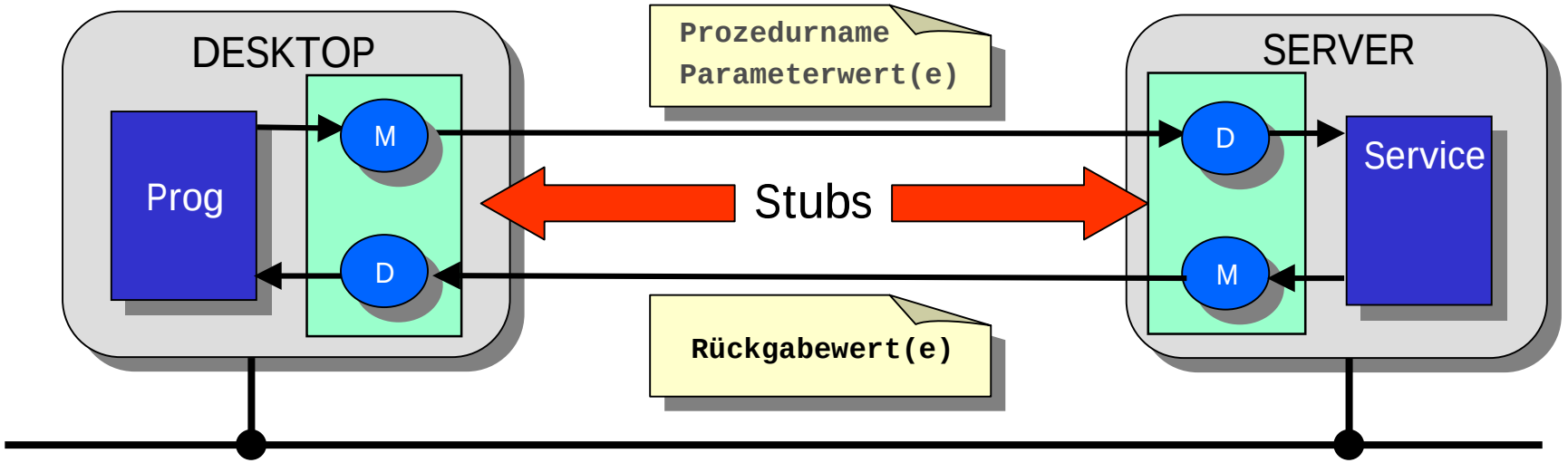


Programmierung verteilter Systeme

- Send und Receive machen die Verteilung der Services für den Programmierer sichtbar
- Ziel: Erreichen einer möglichst hohen und vielfältigen Transparenz der Verteilung für den Benutzer:
  - Ortstransparenz
    - Die Benutzer wissen nicht, wo sich die Betriebsmittel befinden
  - Migrationstransparenz
    - Die Betriebsmittel können migrieren, ohne daß sich ihr Name ändert
  - Replikationstransparenz
    - Die Benutzer wissen nicht, wie viele Kopien existieren
  - Nebenläufigkeitstransparenz
    - Mehrere Benutzer können automatisch Betriebsmittel gemeinsam nutzen
  - Parallitätstransparenz
    - Aktivitäten können ohne das Wissen des Benutzers parallel stattfinden



Remote Procedure Call (Birrell, Nelson 1984)



- Aufruf einer entfernten Prozedur erfolgt genau wie der Aufruf einer lokalen Prozedur; synchron, blockierend
- Dynamisches Binden: der Ort des Services ist transparent
- Fehlerbehandlung nur in lokalen Netzen (effizient) zu realisieren
  - Verlust von Anfragen; Lösung: Stoppuhr
  - Verlust von Antworten; Lösung: Flags schützen vor Kopien
  - Serverabsturz; exactly-once Semantik nicht allg. zu erreichen
  - Clientabsturz; Lösung: Reincarnation



Zusammenfassung - Programmierung

- Bekannte Techniken zur Programmierung lokaler verteilter Systeme:
  - Client/Server
  - Remote Procedure Call, Remote Method Invocation, CORBA
- Ziel für den Programmierer
  - Transparenz der Verteilung der Ressourcen
- Gute Anwendbarkeit im Bereich lokaler Netze
  - mit homogenen Architekturen
  - mit guten Netzwerkqualitäten
- Ausreichende Fehlerbehandlung
  - da Fehler in diesen Netzen selten sind
- Aber: diese Techniken sind für die Programmierung weit verteilter Systeme nicht geeignet:
  - schlechte Performance
  - Fehlerbehandlung unzureichend

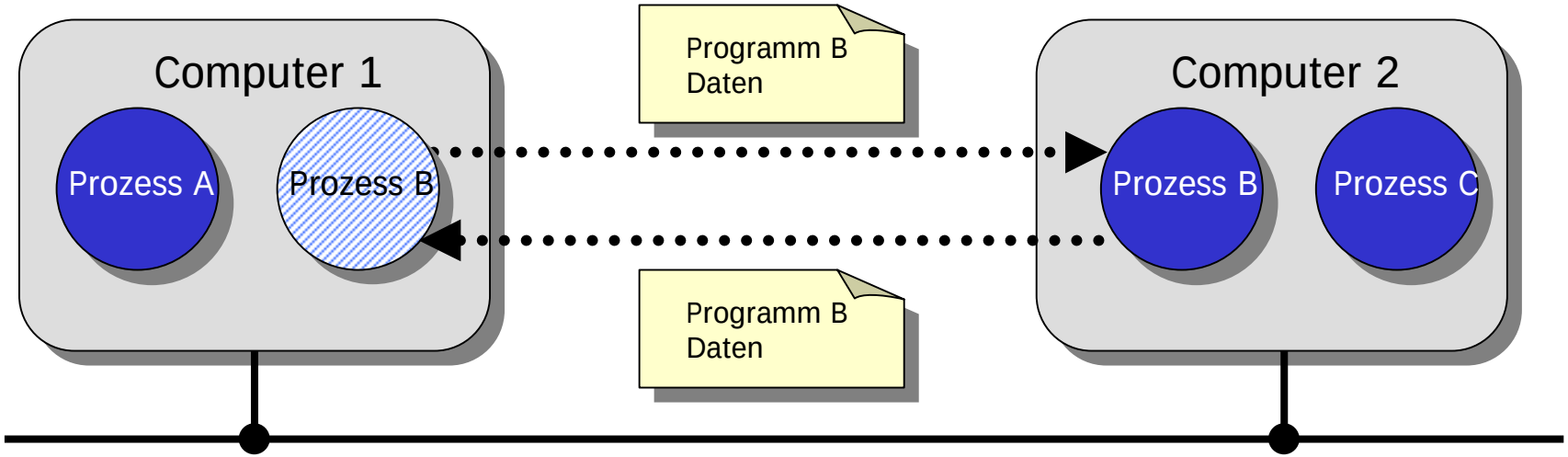


Überblick Code-Mobilität

- Remote-Batch-Jobs (Anfang 70er Jahre)
- PostScript (80er Jahre)
- Prozessmigration in verteilten Betriebssystemen
  - CONDOR (Mitte 80er Jahre)
  - SPRITE (Anfang 90er Jahre)
- Objektmigration
  - Emerald (Ende 80er Jahre)
  - JavaParty (ca. 1997)
- Mobiler Code
  - Java-Applets (ca. 1995)
- Mobile Agenten
  - Telescript, AgentTCL (ca. 1996)
  - Aglets, D'Agents, Mole, Ara, Tracy u.v.a.m. (ab ca. 1997)



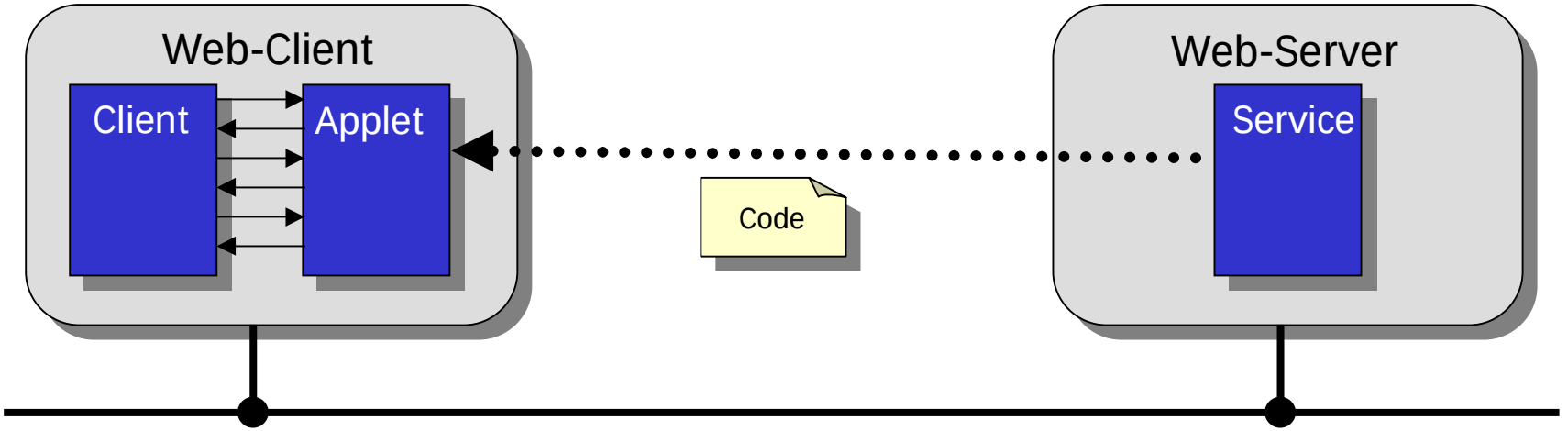
Prozessmigration



- Ziel: Lastverteilung
- Migration wird durch das Betriebssystem initiiert
- Checkpointing in festgelegten Zeitintervallen
- Transparenz für den Benutzer
- Homogene Umgebung



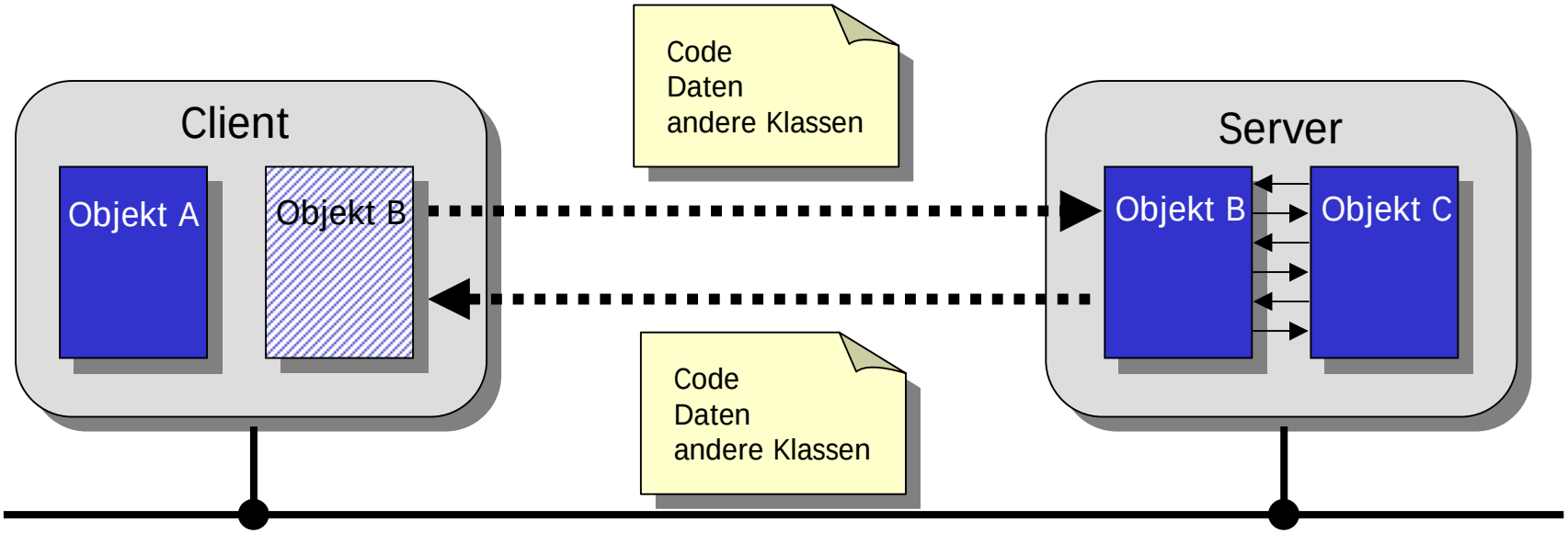
Code on Demand: Java Applets



- Ziel: dynamisches Erweitern der Funktionalität des Browsers
- Es werden meist nur Klassendateien übertragen
  - mit serialisierten Applets können auch Daten übertragen werden
- Die Anforderung wird vom Client gesteuert
- Verbindung zum Netz notwendig, da ständiges Nachladen von fehlenden Klassen oder Ressourcen (z.B. Grafiken) möglich



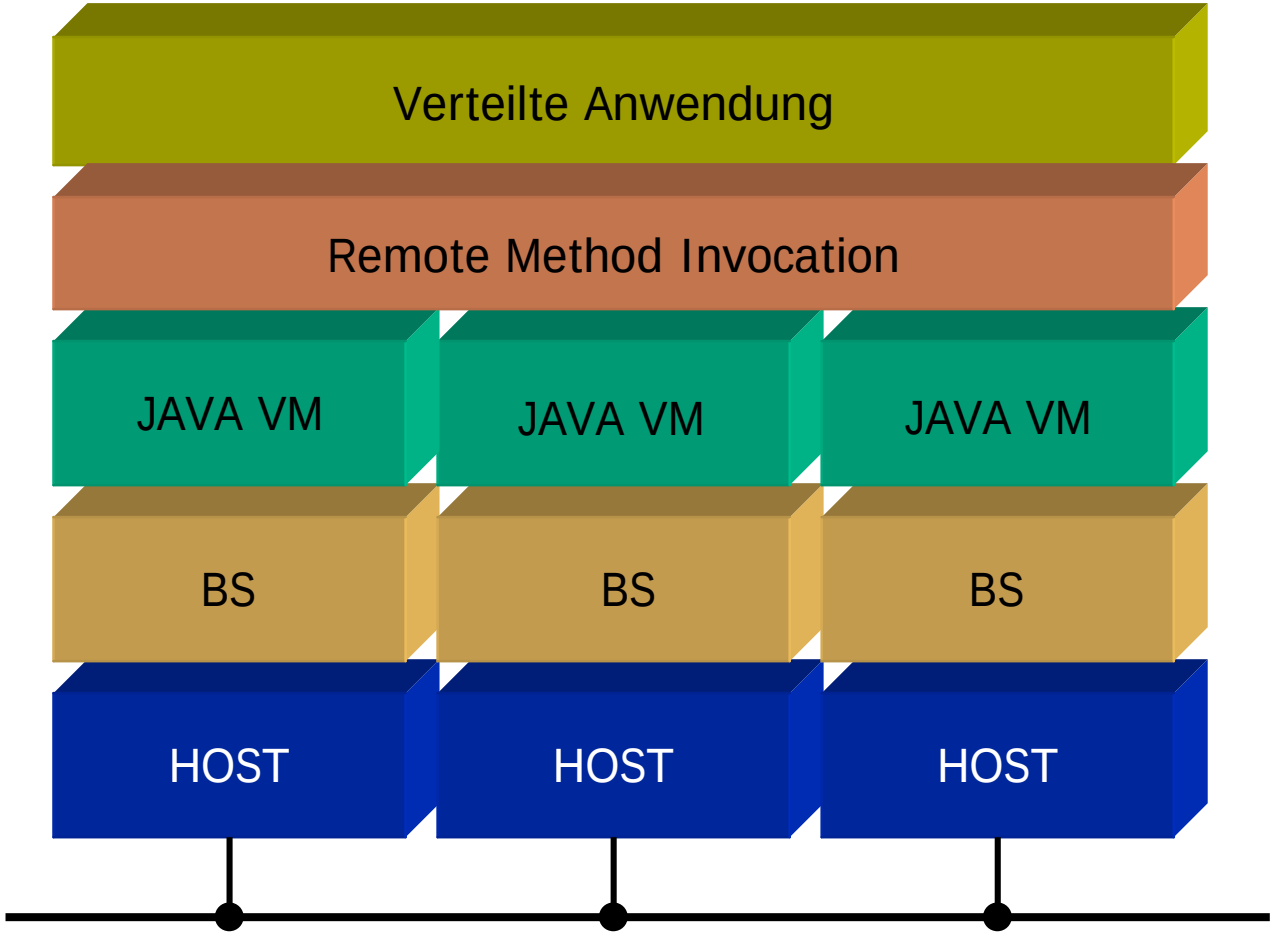
Verteilte Objekte: RMI



- Im Zuge eines entfernten Methodenaufrufes wird ein serialisiertes Objekt übertragen, dessen Code auf dem Server nicht vorhanden ist
- Der Code von Objekt B migriert auf Anforderung auf den Server
- Transparenz für den Benutzer
- Problem: ein Objekt ist weniger abgeschlossen als ein Prozess

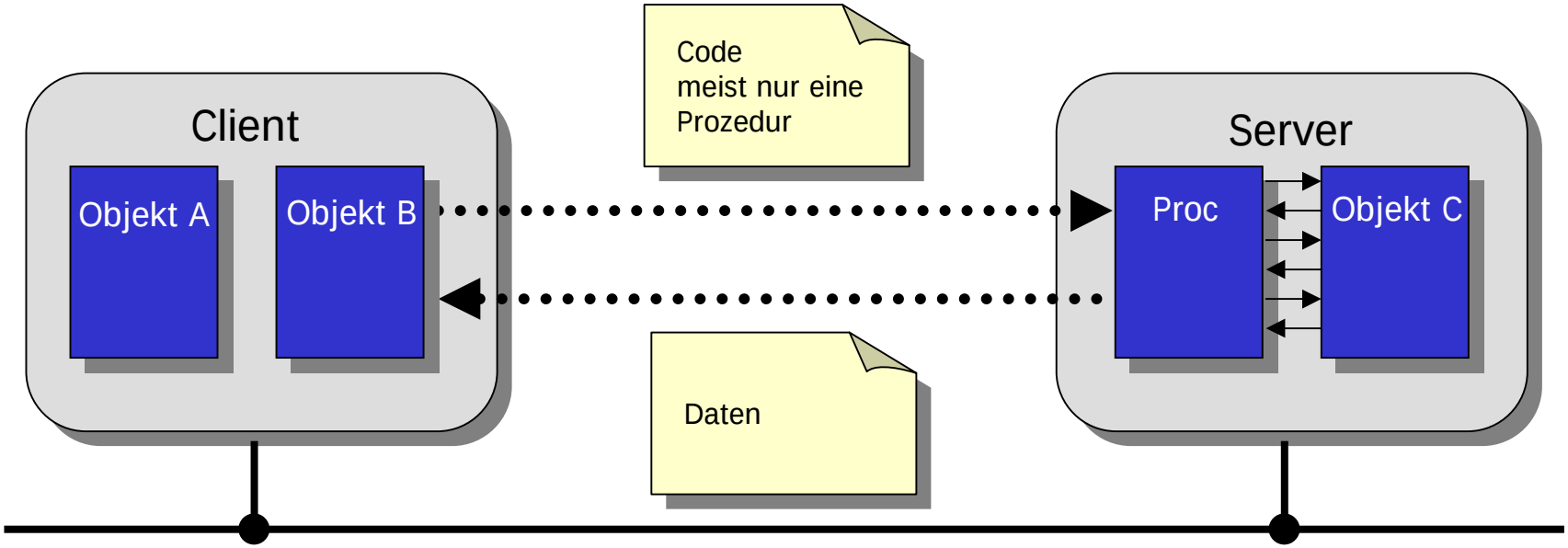


Architektursicht: Verteilte Objekte





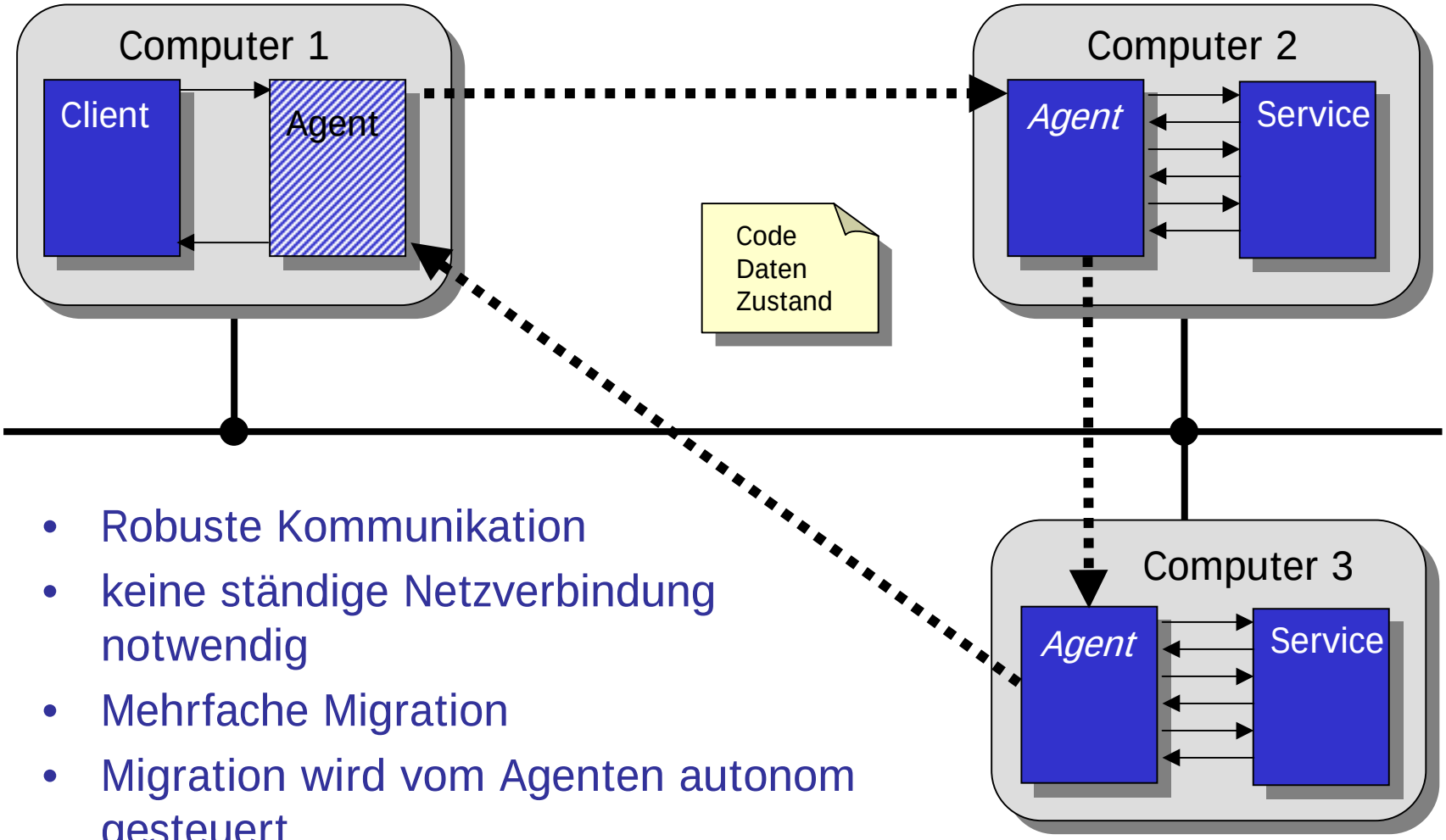
Remote Evaluation



- Der Code einer Prozedur wird mit Parametern verschickt.
- Gut für einen Hop
- Prozeduren können nicht kommunizieren, keine anderen Funktionen aufrufen
- Prozeduren können nicht wieder Prozeduren verschicken



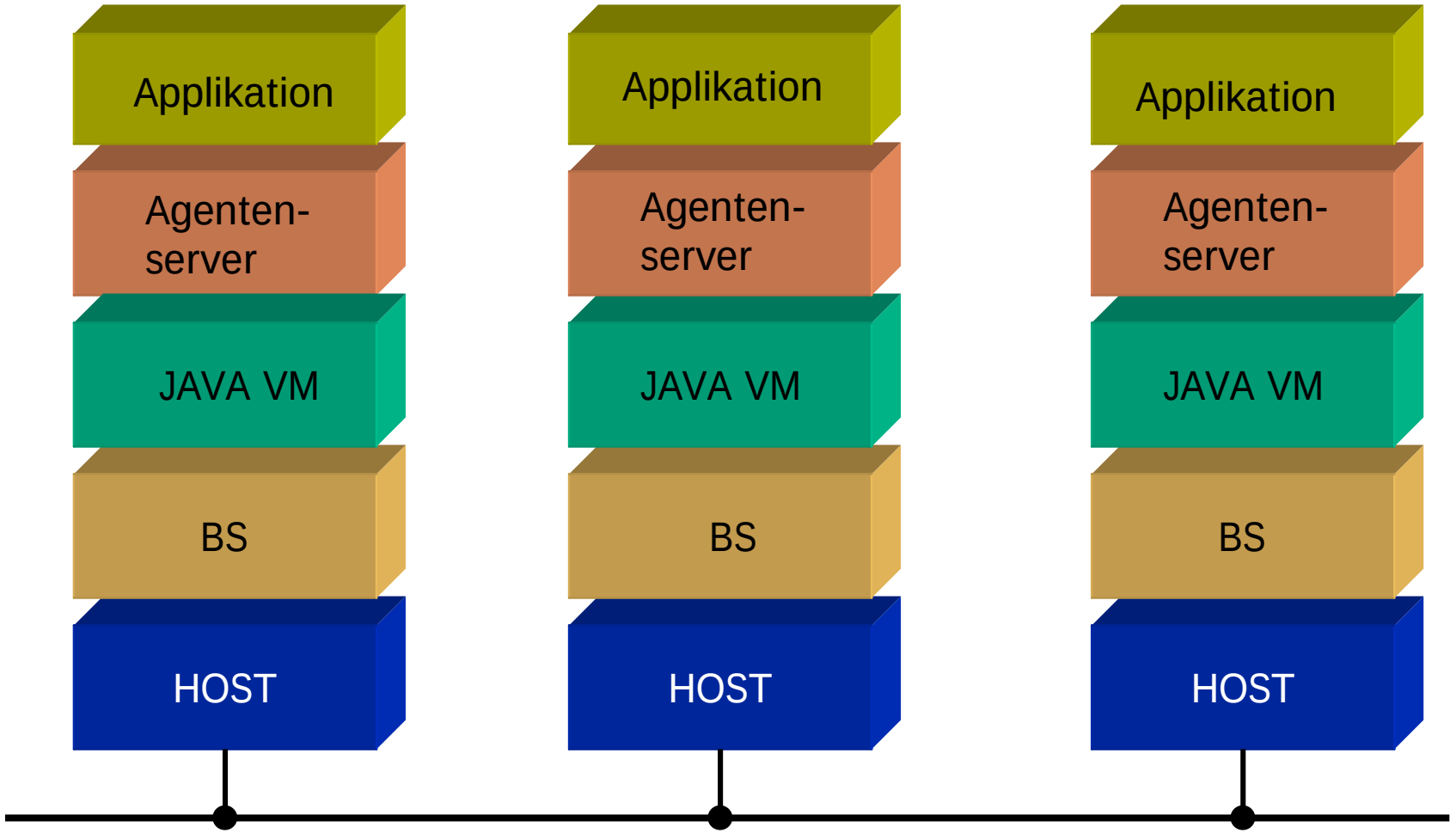
Mobile Agenten



- Robuste Kommunikation
- keine ständige Netzverbindung notwendig
- Mehrfache Migration
- Migration wird vom Agenten autonom gesteuert
  - Wann? Wohin?



Architektursicht: Mobile Agenten





Mobile Objekte vs. Mobile Agenten

	Mobile Objekte	Mobile Agenten
Netzwerk	klein, homogen: LAN	groß, heterogen: WAN
Migrationsgrund	Lastverteilung	Nutzung verteilter Ressourcen
Migration	transparent	sichtbar
Auslöser	Betriebssystem/ Middleware	Agent/ Programmierer
Einheit	Objekt als Teil einer Applikation	Agent als eigene Applikation
Kommunikation	Methodenaufruf	Nachrichten



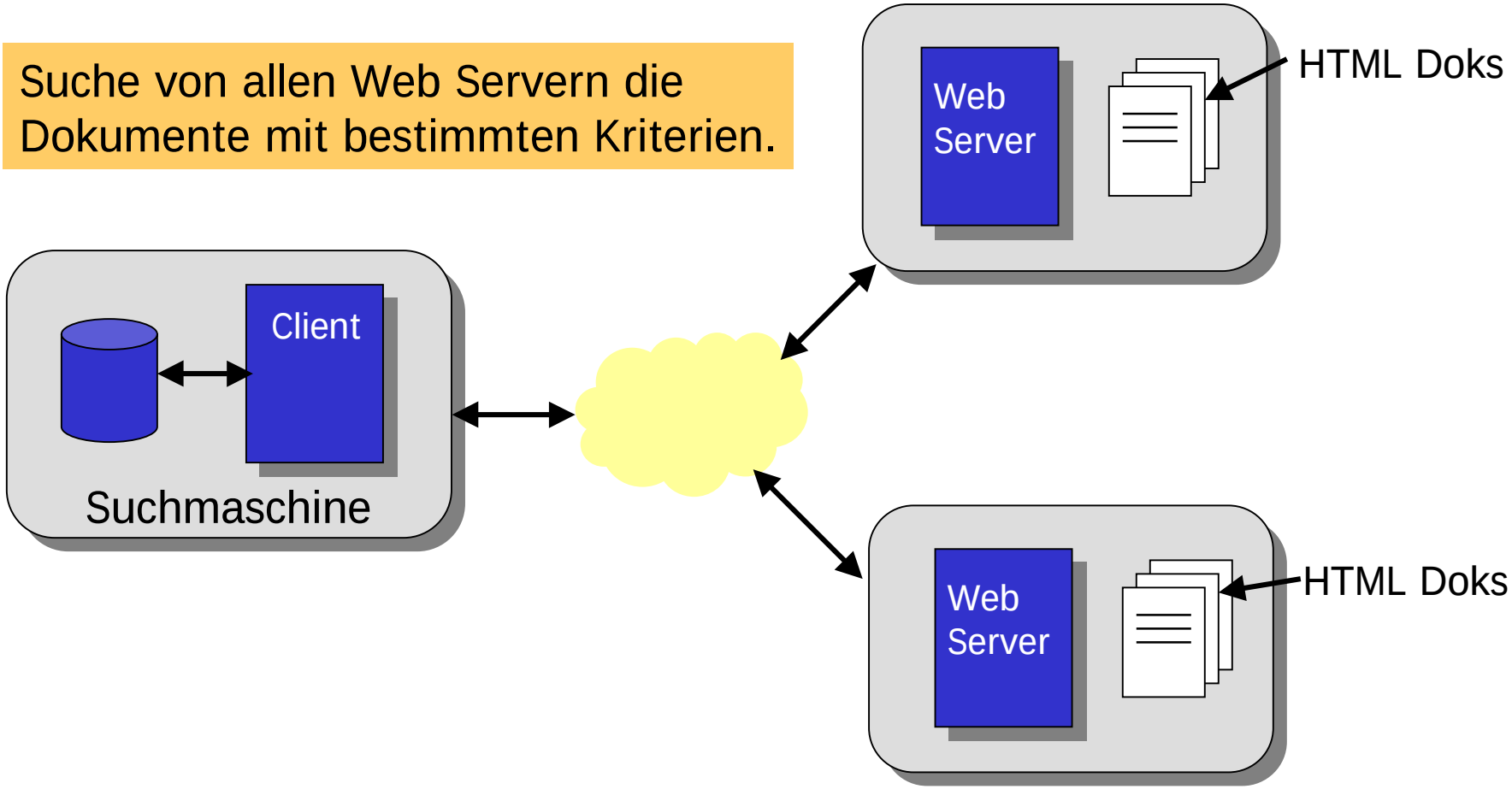
Vorteile von mobilen Agenten (1)

- Sie reduzieren die Netzbelastung im Vergleich zu CS
  - durch Selektierung der Daten und Reduzierung der Ergebnismenge
  - durch Vermeidung von Zwischenkommunikation
  - durch Peer-to-Peer Migration
- Wenn die Übertragungszeit die Ausführungszeit dominiert, ist damit auch die gesamte Ausführungszeit geringer!



Beispiel 1: HTML Dokumente

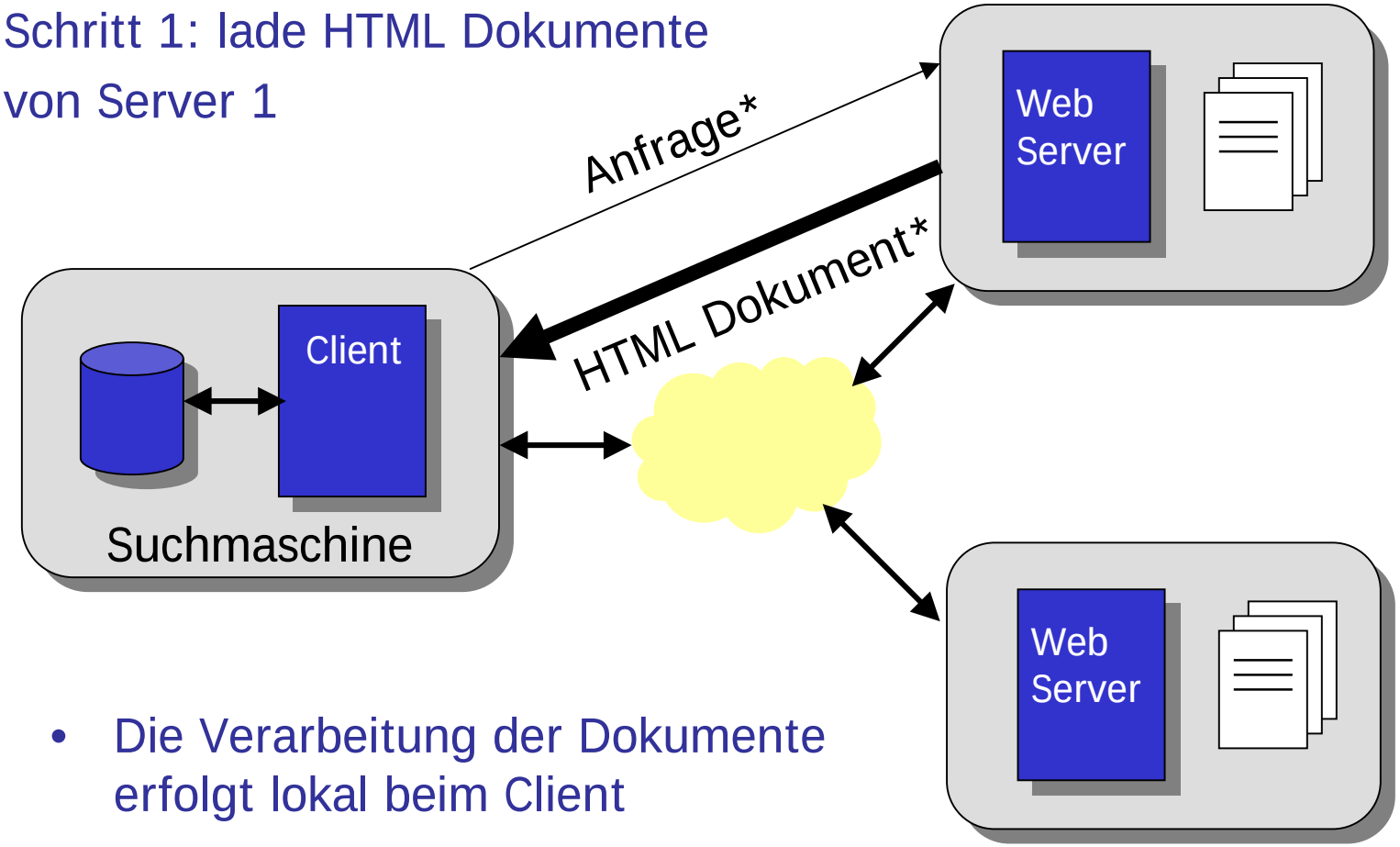
Suche von allen Web Servern die Dokumente mit bestimmten Kriterien.





Client/Server Ansatz

Schritt 1: lade HTML Dokumente von Server 1

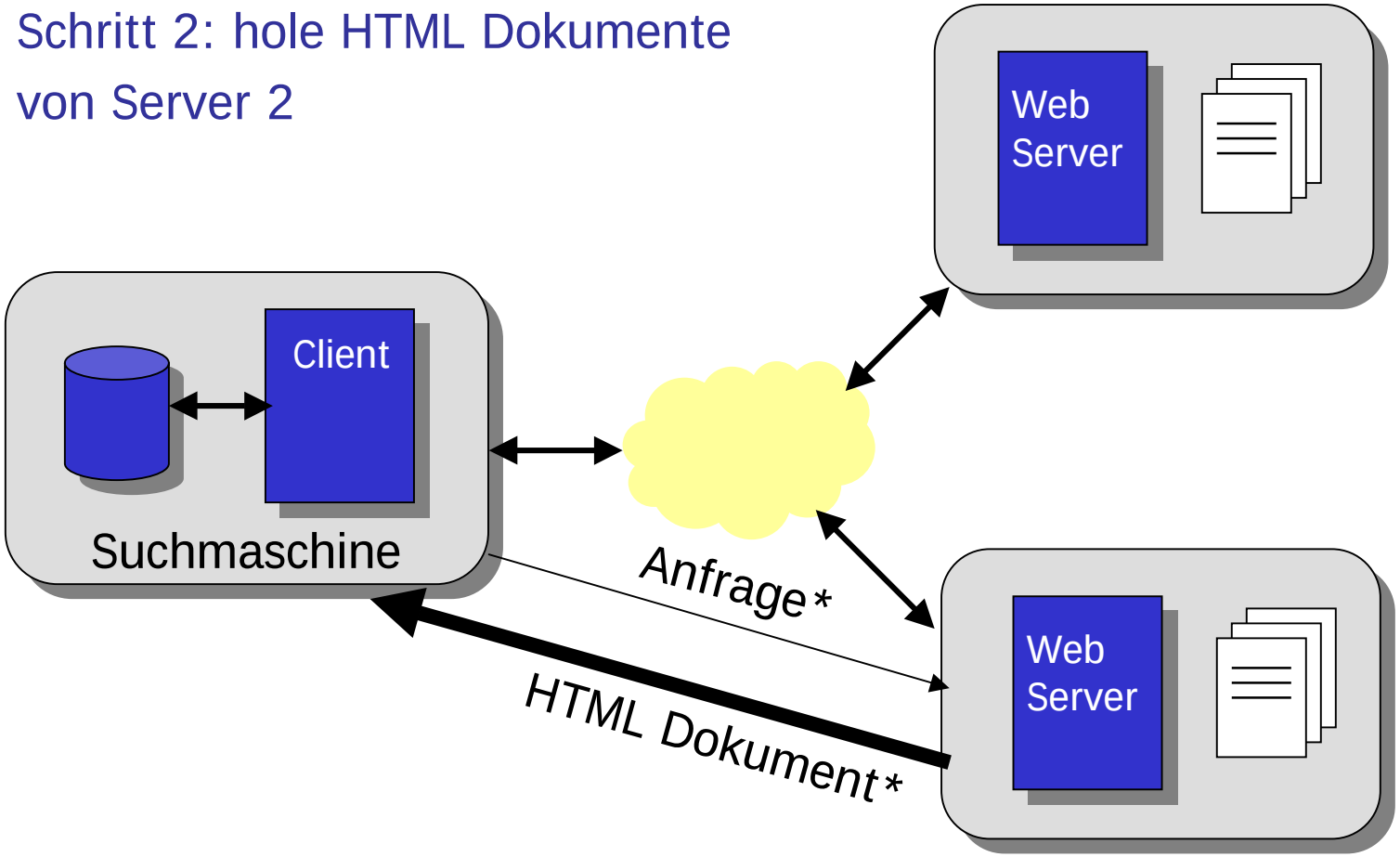


- Die Verarbeitung der Dokumente erfolgt lokal beim Client



Client/Server Ansatz

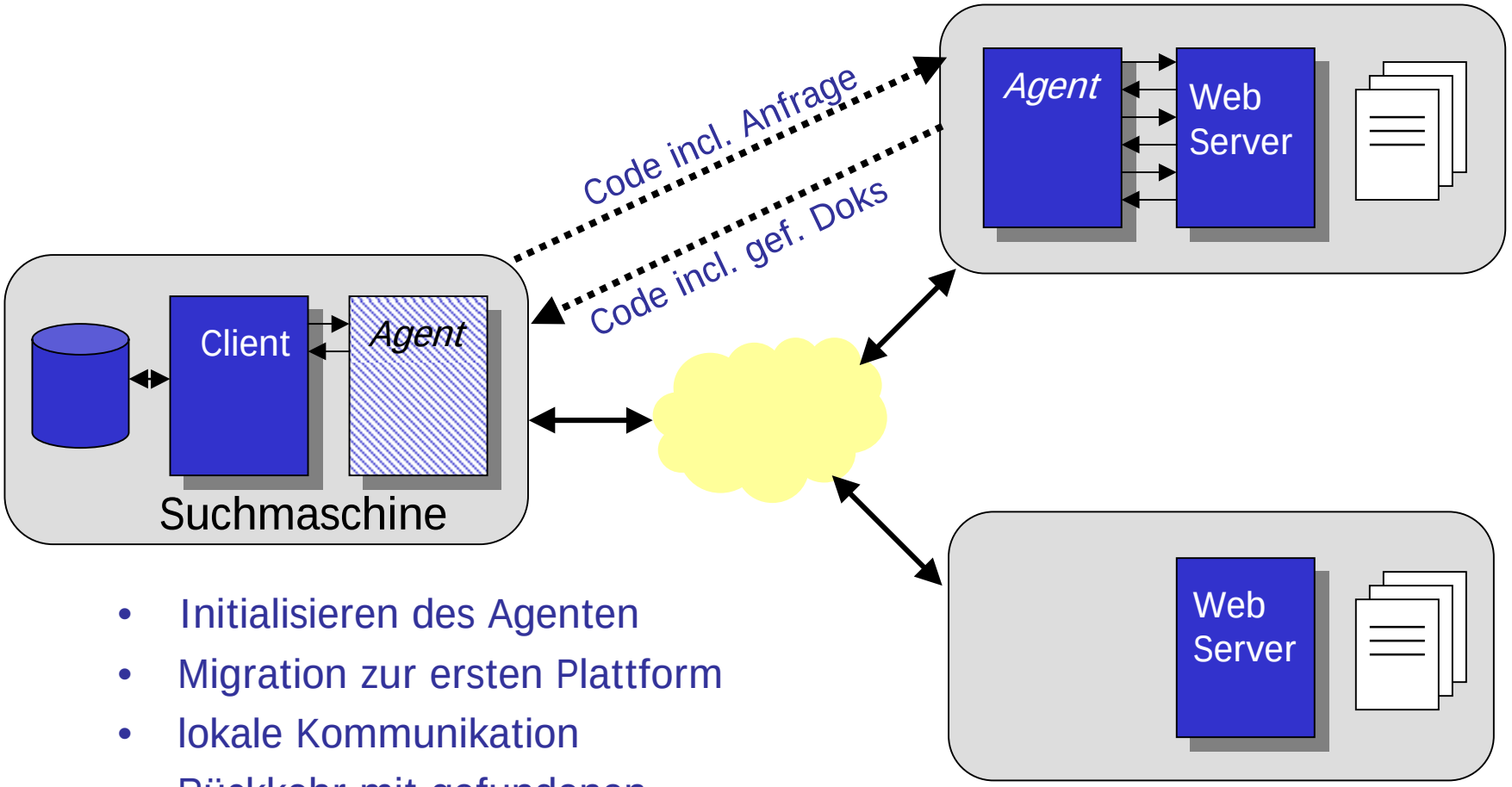
Schritt 2: hole HTML Dokumente von Server 2



- hohe Netzbelastung, da stets alle Dokumente geladen werden müssen
- skaliert schlecht in der Anzahl der „relevanten“ Dokumente



Mobile Agenten Ansatz

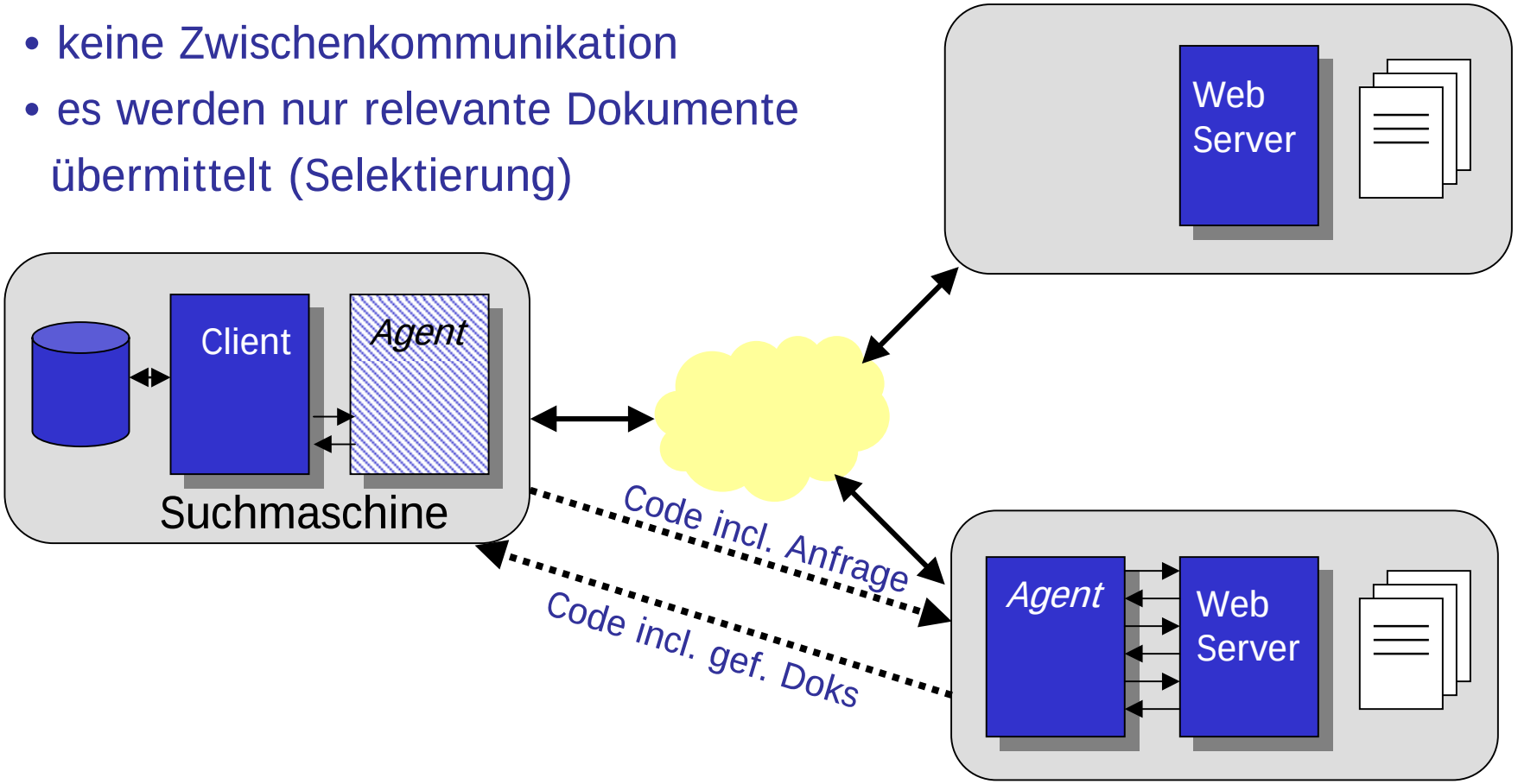


- Initialisieren des Agenten
- Migration zur ersten Plattform
- lokale Kommunikation
- Rückkehr mit gefundenen Dokumenten



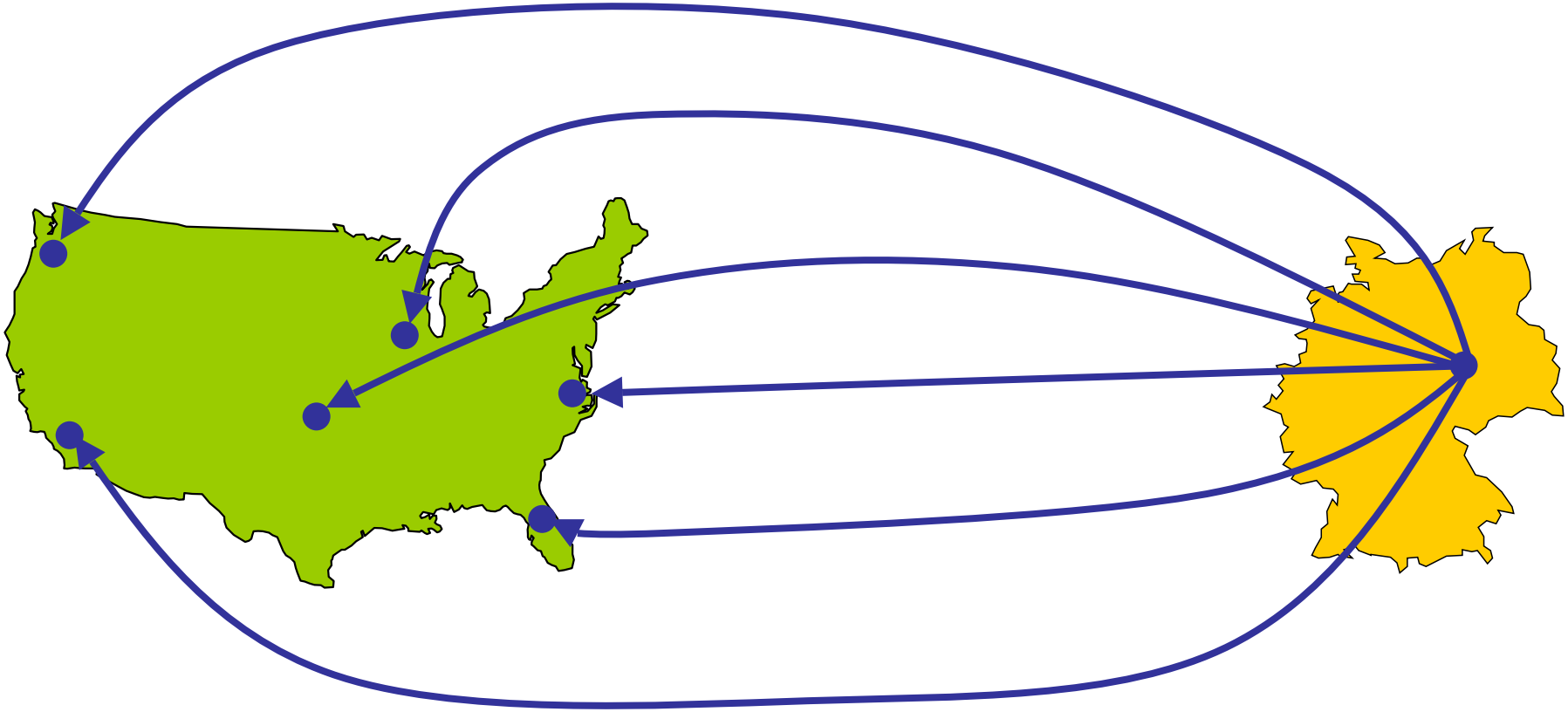
Mobile Agenten Ansatz

- keine Zwischenkommunikation
- es werden nur relevante Dokumente übermittelt (Selektierung)





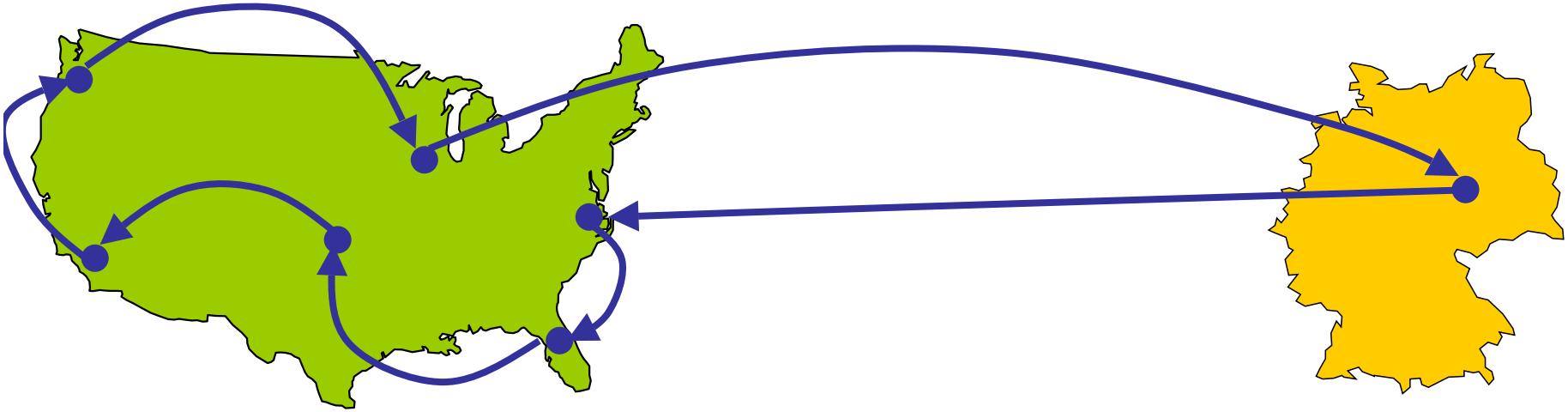
Beispiel 2: Verteilte Anfragen



- Mehrere entfernte Ziele sollen angesprochen werden.
- Im Client/Server Fall müssen zahlreiche einzelne, kostenintensive Verbindungen aufgebaut werden.



Beispiel 2: Verteilte Anfragen

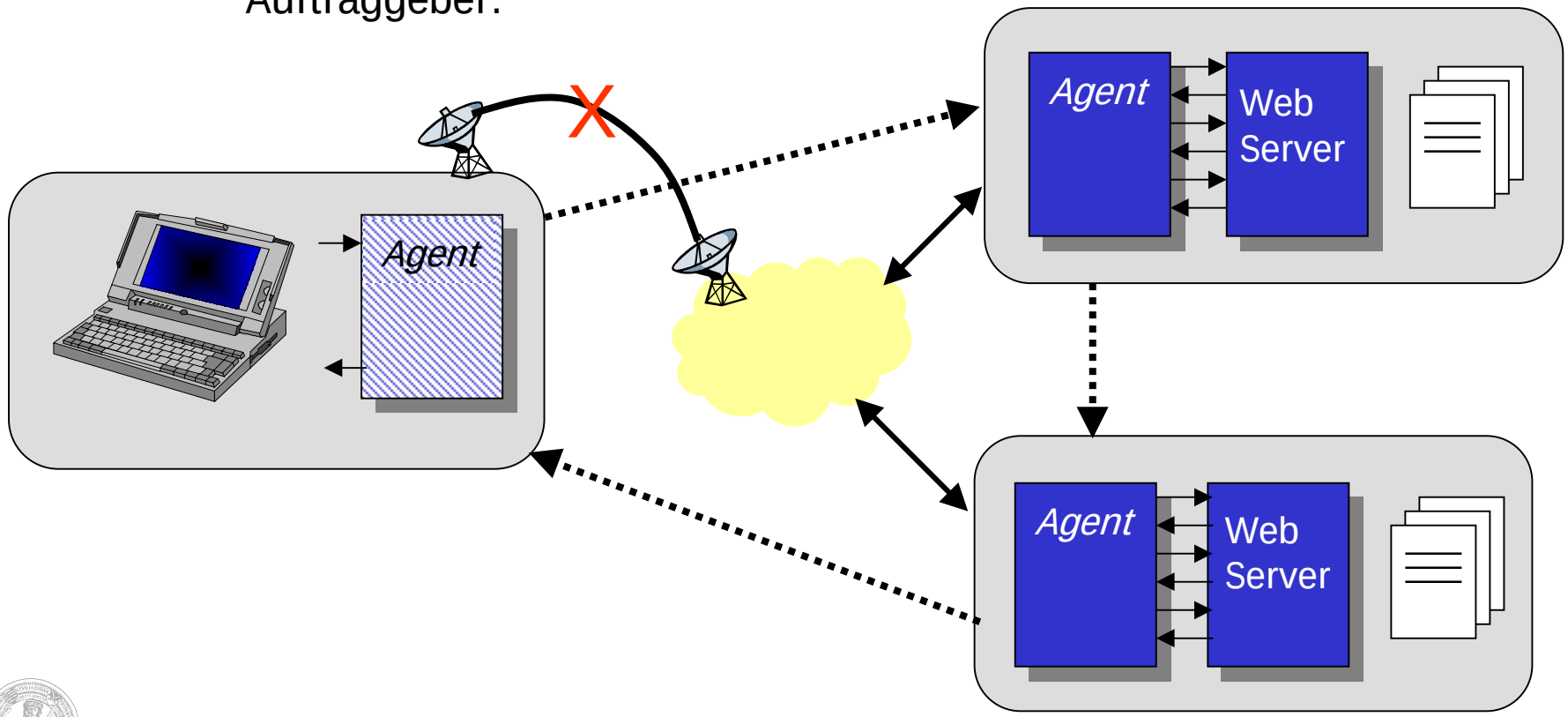


- Mehrere entfernte Ziele sollen angesprochen werden.
- Ein mobiler Agenten nutzt die geringere Entfernung zwischen den Zielen aus.
- Der Agent ist in der Lage, die günstigste Route zu den Zielen selbst zu bestimmen.



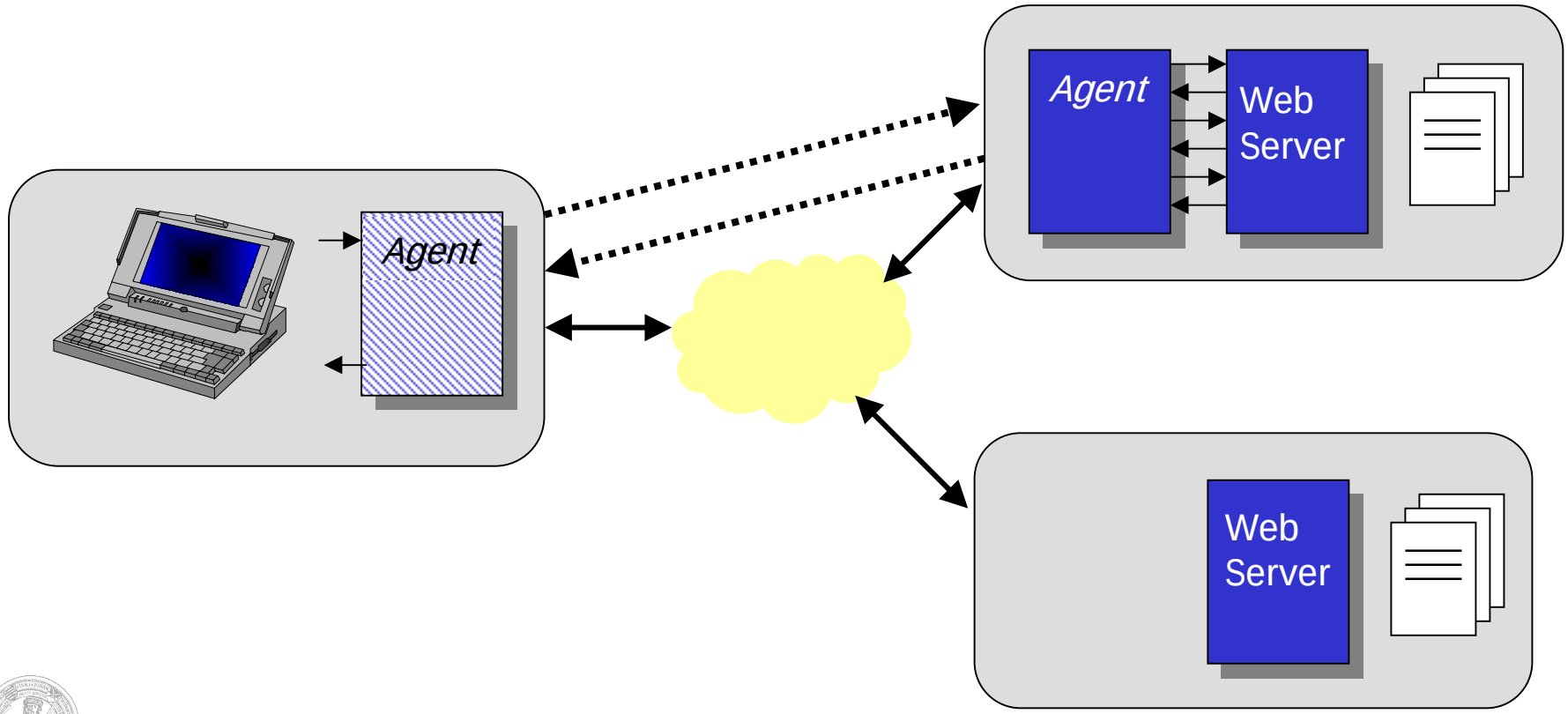
Vorteile von mobilen Agenten (2)

- Sie arbeiten asynchron und autonom.
  - Berechnungen können auf einen leistungsfähigen Rechner ausgelagert werden.
  - Der Agent arbeitet ohne ständige Netzverbindung zum Auftraggeber.



Vorteile von mobilen Agenten (3)

- Sie vermeiden Netzwerk-Latenzen.
  - Ein mobiler Agent kann als „Statthalter“ vor Ort auf Änderungen der Umgebung in Echt-Zeit reagieren.
  - Vermeidung der bei Netzübertragung anfallenden Latenzzeit.





Vorteile im Fehlerfall

- Verlust des Agenten während der Migration
  - wird durch das TCP Protokoll und Transaktionsmechanismus im Agentensystem verhindert
- Verlust von Anfragen
  - kann nicht auftreten, da die Kommunikation lokal ist
- Verlust von Antworten
  - kann nicht auftreten, da die Kommunikation lokal ist
- Serverabsturz
  - wenn „absehbar“, dann kann der Agenten eingefroren werden oder migrieren
  - andersfalls, offenes Problem
- Clientabsturz
  - unkritisch, der Agent wartet auf neue Verfügbarkeit (store-and-forward)

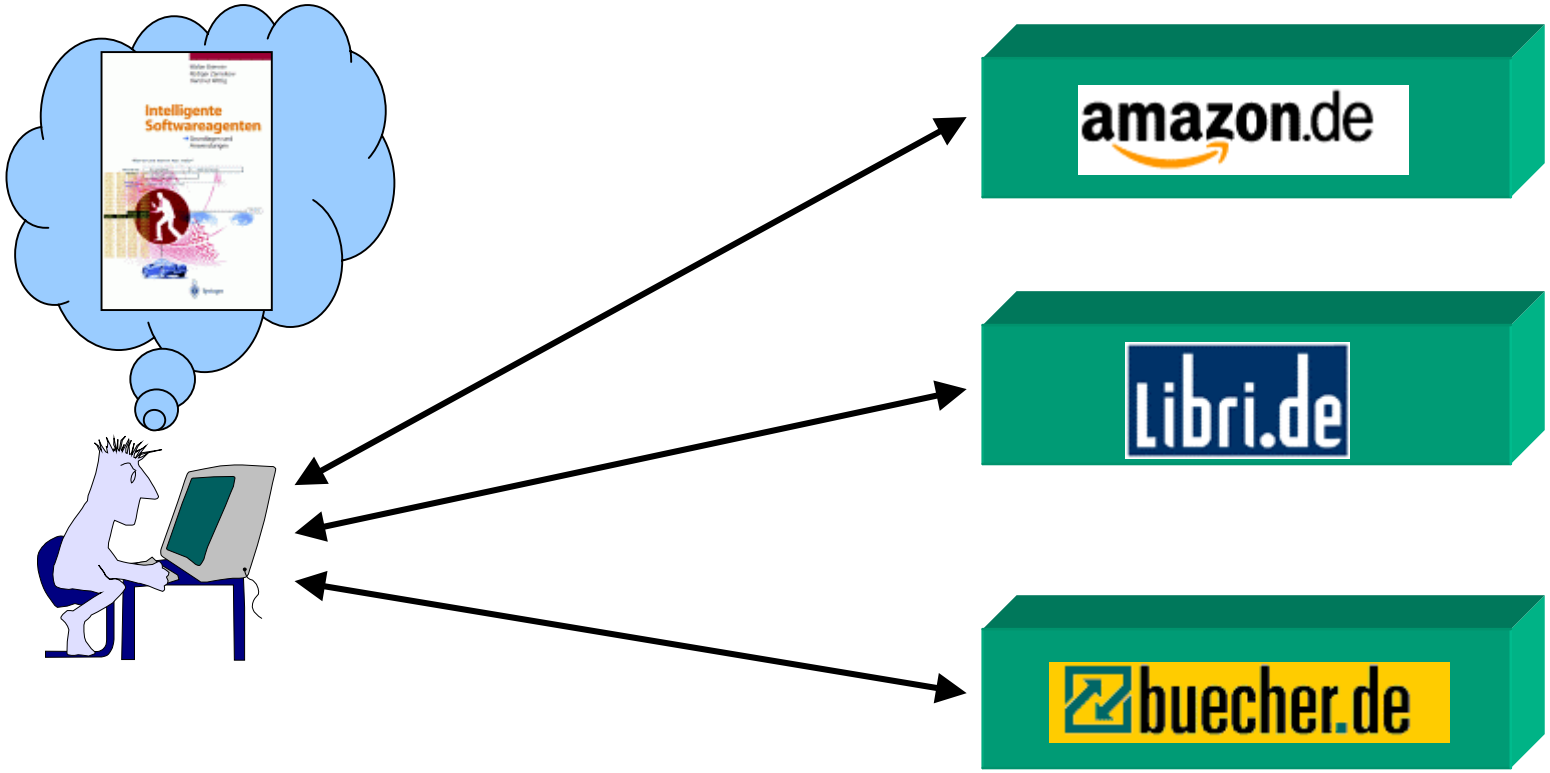


Anwendungsgebiete

- Intelligentes Abfragen verteilter Datenbanken
  - bei mobilen Computern
  - dynamische parallele Suche durch Clonen von Agenten
- Beobachten von entfernten Ereignissen
  - ein Agent migriert zu einem Server, beobachtet die Daten und meldet sich nur bei „interessanten“ Veränderungen (z.B. Börsendaten, Katalogen)
  - Vermeidung von wiederholten Datenzugriffen (Pulls)
- Unterstützung von verteilten Workflow-Anwendungen
  - statt Daten fließen Agenten mit Daten und Code
- Nutzung verteilter Rechenleistung
- Software deployment, reconfiguration,
- Network management, active networks
- E-Commerce
  - Marktplätze von handelnden Agenten
  - Shopping Agenten



Anwendung: Online-Shop



- Der Nutzer möchte ein Buch kaufen und erkundigt sich bei einigen Shops nach dem günstigsten Preis.
- Dazu sucht er manuell bei diesen Shops Preis und Lieferbedingungen heraus und entscheidet sich schließlich.



Mobile Einkaufsagenten





















- Ein Agent migriert selbständig zu vielen Shops und erfragt den Preis.
- Anschließend kauft er das Buch beim günstigsten Anbieter.
- Andere Auswahlkriterien sind denkbar.



Bedenken

- Sicherheit!
  - ein mobiler Agent ist ja wie ein Virus!
  - Aber
    - viele Sicherheitsprobleme sind mit bekannten Methoden zu lösen
    - Anforderungen an die Sicherheit sind abhängig von der Anwendung
- Keine Kontrolle möglich (Folge der gewünschten Autonomie)
  - Wie finde ich meinen Agenten wieder?
- Netzwerke werden immer schneller!
  - daher ist eine Optimierung der Netzbelastung überflüssig
  - Aber:
    - Schere zwischen Backbone und „letzter Meile“ wird größer
    - Netzbelastung wird in den nächsten Jahren noch dramatisch zunehmen
    - Datenbanken werden immer größer
- Keine Killer-Application!
  - Aber:
    - mobile Agenten sind eine gute Ergänzung, keine Alternative!



Zusammenfassung Teil I

- Mobile Agenten sind ein neues Paradigma für den Aufbau und die Programmierung von verteilten Systemen.
- Mobile Agenten ergänzen bisherigen Techniken durch mobilen Code und Multi-Hop Möglichkeit.
- Mobile Agenten haben zahlreiche Potenziale im Bereich der
  - Performance
  - Programmierbarkeit
  - Skalierbarkeit
- Fehlerbehandlung ist einfacher als bei RPC o.ä.
- Viele Probleme sind noch offen (mehr dazu in Teil II).

- Mobile Agenten sind auf jeden Fall sinnvoll, wenn
  - wenige Operationen auf großen entfernten Datenmenge ausgeführt werden soll
  - mehrere entfernte Datenbanken abgefragt werden sollen



Funktionsweise von mobilen Agenten



Überblick zu diesem Teil

- Begriffe
- Arbeitsweise mobiler Agenten
- Migrationsmodelle
- Kommunikationsaspekte
- Kontrollprobleme und -mechanismen bei mobilen Agenten
- Sicherheitsaspekte
- Produkte und Projekte



Java als Sprache für mobile Agenten

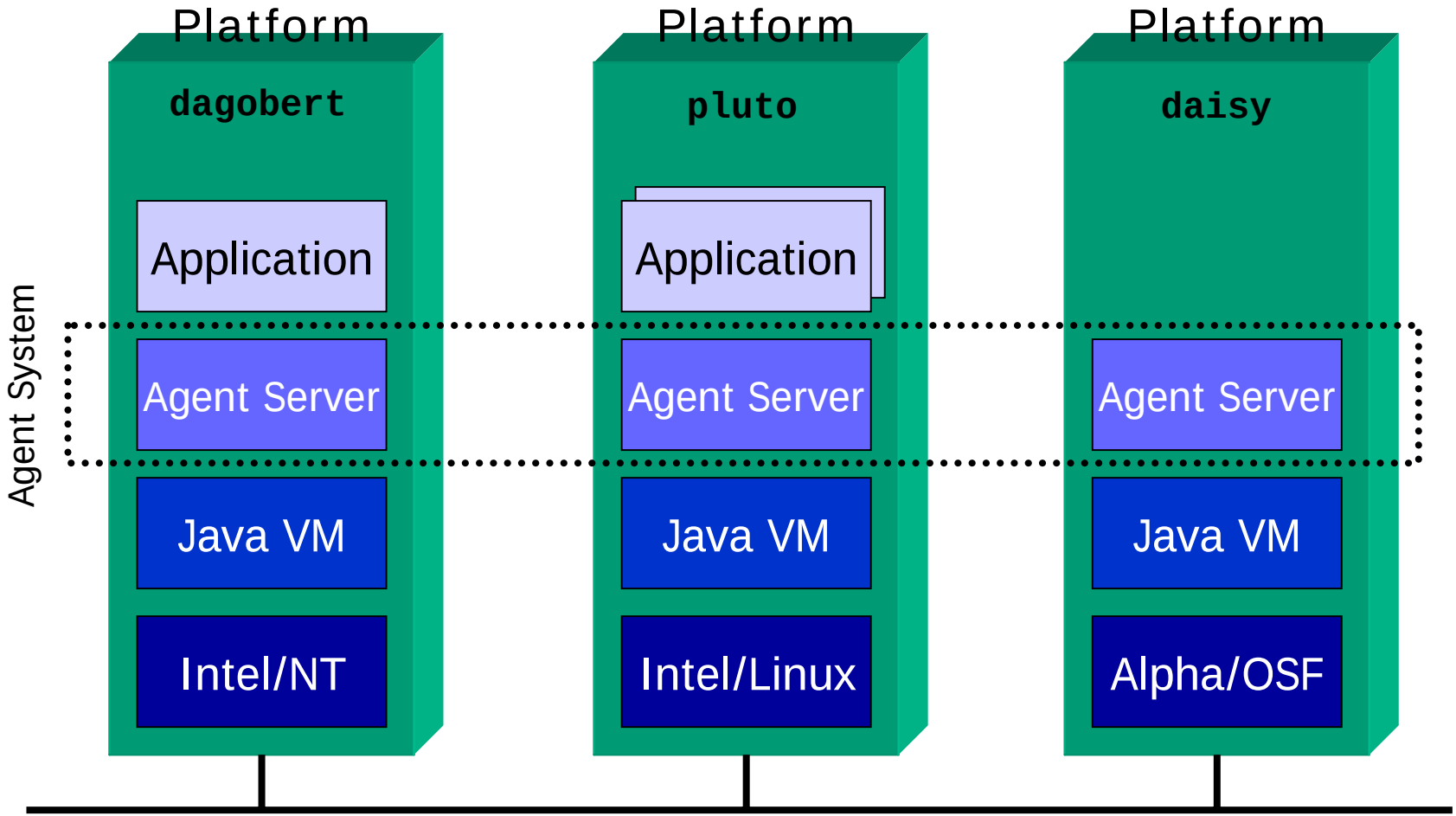
- Java ist seit ca. zwei Jahren die Programmiersprache für mobile Agenten und Agentensysteme
- durch Java wurden Agentensysteme einfach zu programmieren

- Vorteile von Java
  - Portabel
    - Java Programme werden in architekturneutrales Bytecode-Format übersetzt
  - Sicherheit
    - Überprüfung des Codes (Code Verifier)
    - Java 2 Sicherheitsmodell mit der Möglichkeit der individuellen Vergabe von Rechten
  - Effizienz
    - durch Just-In-Time Compilation und HotSpot-Technik
    - im Vergleich zu „reinen“ Interpretersprachen
  - außerdem: dynamisches Nachladen, Reflection, Objektserialisierung





Infrastruktur





Arbeitsweise mobiler Agenten

System Agent

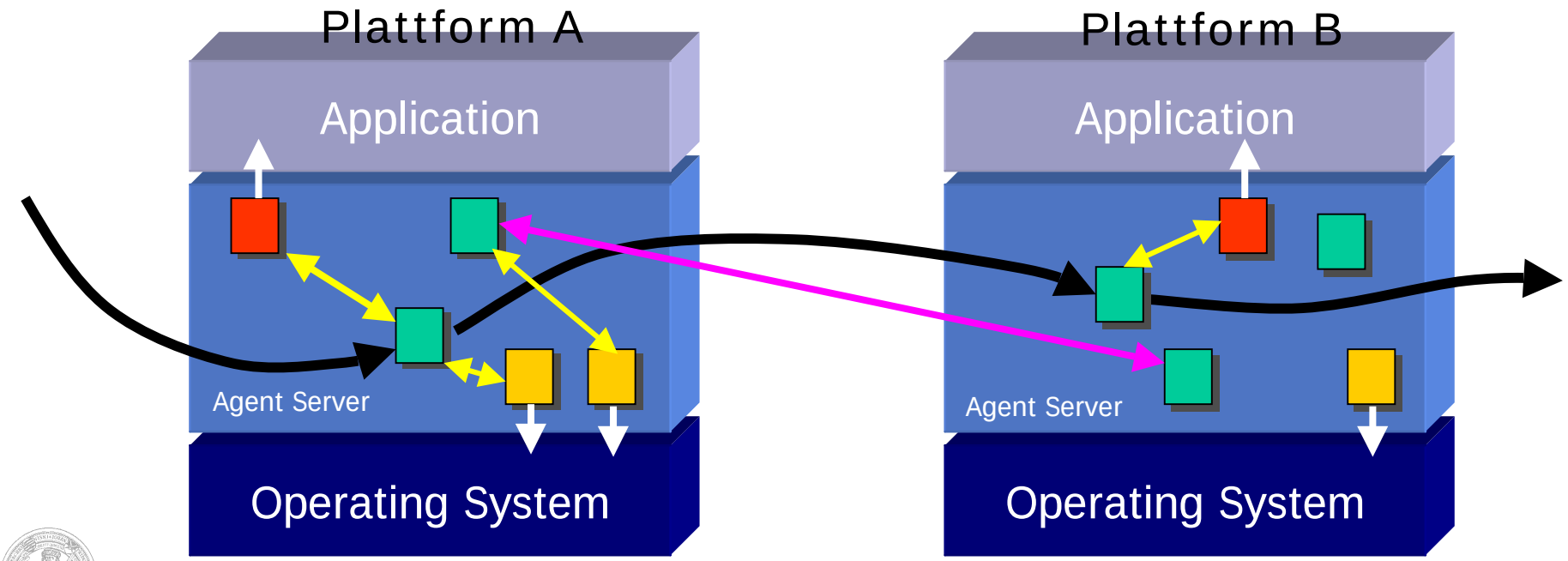
Gateway Agent

Mobiler Agent

Migration eines Agenten

Lokale Kommunikation

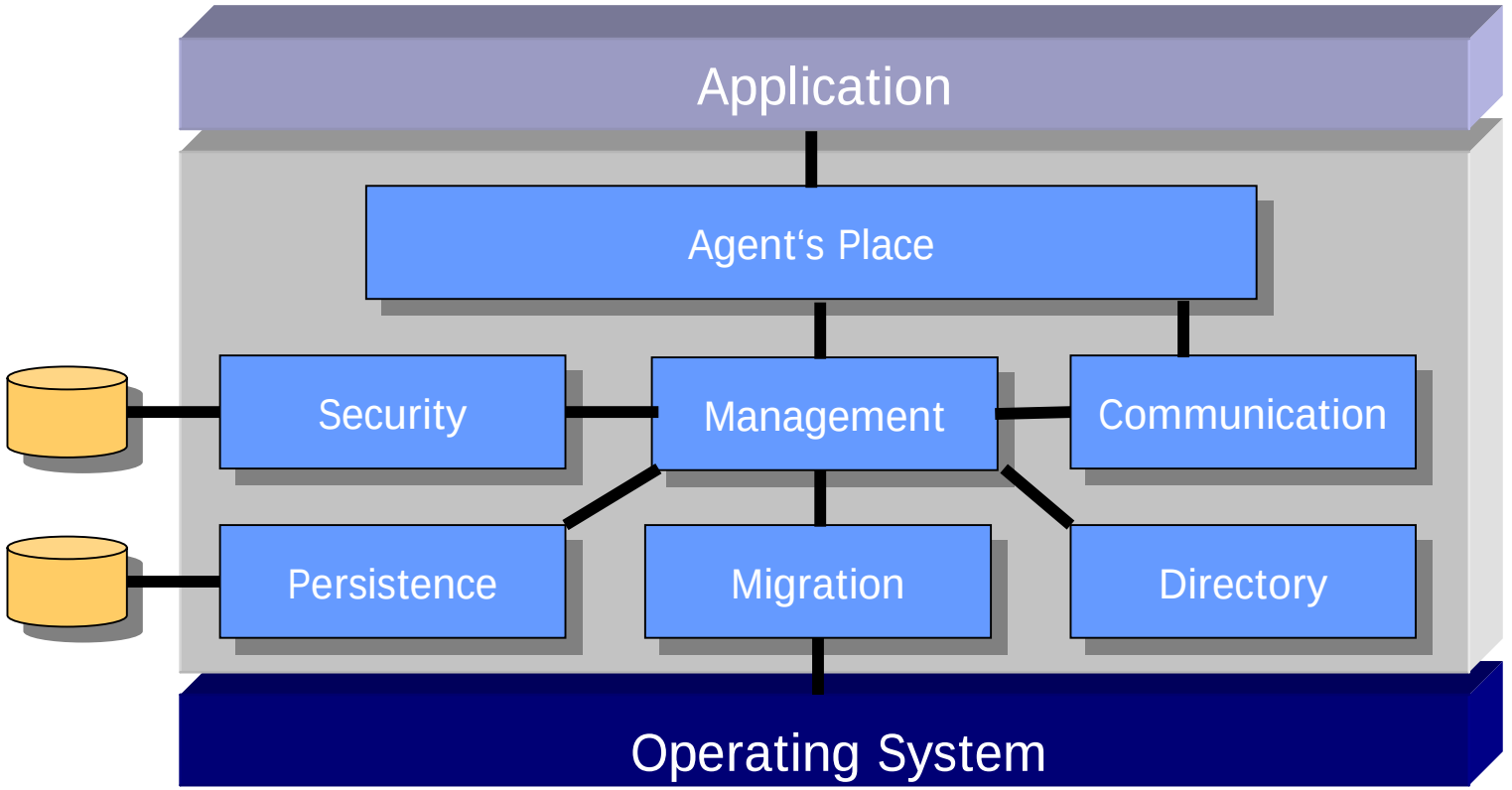
Entfernte Kommunikation





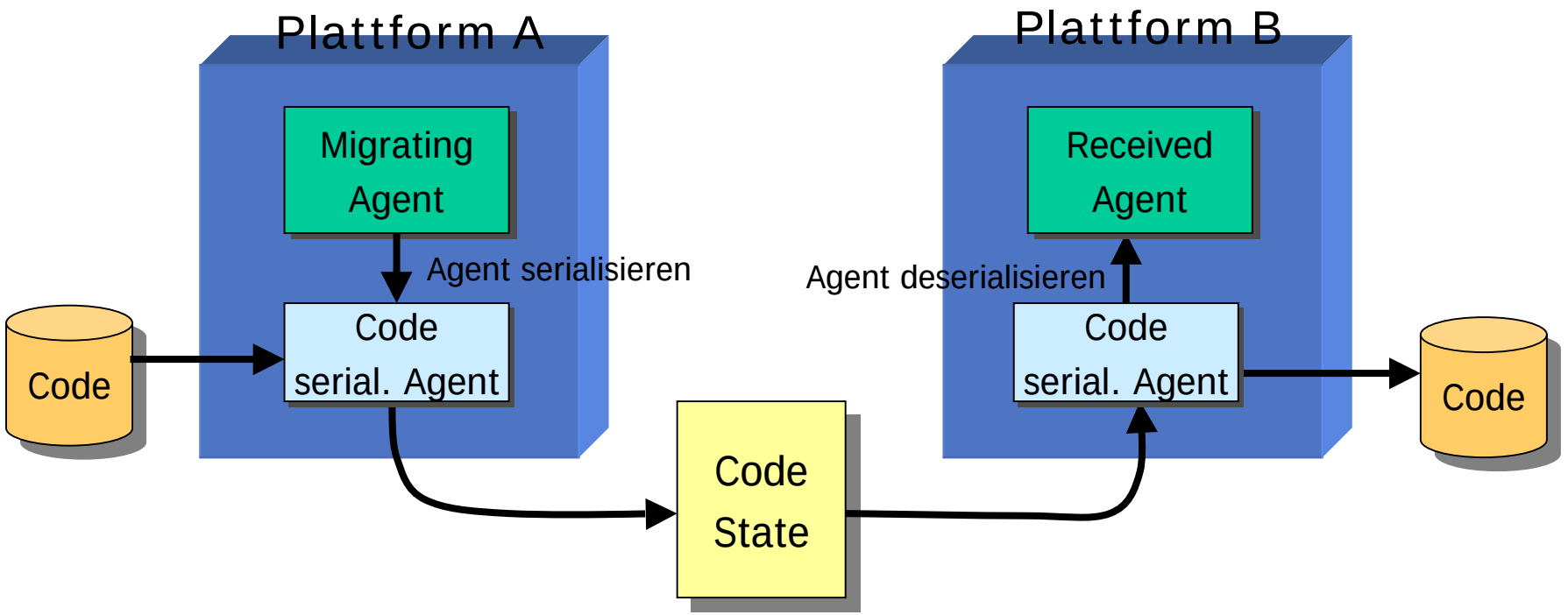
Agent Server

- Der Agent Server nimmt Agenten auf und führt sie aus.
- Er bietet grundlegende Dienste an:
  - Migration, Kommunikation, Verzeichnisse





Ablauf einer Migration



1. Ausführung unterbrechen
2. Agent serialisieren
3. Code einlesen
4. Agent mit Code und State verschicken
8. Agent ausführen
7. Agent deserialisieren
6. Code speichern
5. Agent empfangen



Migrationsmodell (1)

- Sicht des Programmierers
  - Migrationsart: stark/schwach
  - Ziel der Migration wird vorgegeben/kann selbst definiert werden
  - Kommandos zur Migration
- Sicht auf die Migrationsstrategie
  - *was* migriert *wohin*?
    - migrieren alle oder nur bestimmte Daten
    - migrieren alle Klassen, nur einige oder nur bestimmte einzelne Methoden/Befehle
  - von wo können benötigte Klassen nachgeladen werden
- Sicht auf die Übertragungsstrategie
  - welches Protokoll wird verwendet?
  - Datenkomprimierung? Verschlüsselung?
  - Transaktionsmanagement?



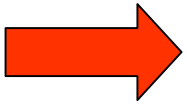
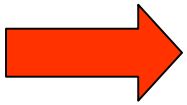
Sicht des Programmierers

- Agentensysteme können nach der Art der angebotenen Migration unterschieden werden
  - schwache Migration mit festem Einsprung
  - schwache Migration mit beliebigem Einsprung
  - starke Migration
- je schwächer die Migrationsart, desto mehr Aufwand hat der Programmierer
- je stärker die Migrationsart, desto komplizierter wird die Migration für das Agentensystem
- Beispiele:
  - Schwache Migration mit festem Einsprung: Aglets (Java)
  - Schwache Migration mit beliebigem Einsprung: Jumping Beans (Java)
  - Starke Migration: AgentTCL (Tcl), D‘Agents (Java)



Migration mit festem Einsprung

```
public class Dispatching Example extends Aglet {
        boolean isRemote = false;
        public void onCreation( Object init ) {
        addMobilityListener( new MobilityAdapter() {
                        pubic void onDispatching( MobilityEvent e ) {}
                        public void onArrival( MobilityEvent e ) {
                                isRemote = true;
                        }
                }
        );
        public void run() {
                if( isRemote == true ) { ... }
        }
}
```

1. 
2. 

- Hier am Beispiel Aglets von IBM
- vor der Migration wird „onDispatching“ aufgerufen
- nach der Migration wird „onArrival“ und dann „run“ aufgerufen



Migration mit beliebigem Einsprung

```
public class TestAgent extends MobileAgent {
        int _state_of_value = 0;
        public void work() {
                int value = f();
                _state_of_value = value;          // save variable
                go( „iaxp12.inf.uni-jena.de“, „runRemote“);
        }
        public void runRemote() {
                // restore variable
                int value = _state_of_value;
                ...
        }
}
```

- einfach zu implementieren (mit Java Reflection)
- schwer zu benutzen, da lokale Variablen gerettet werden müssen



Starke Migration

```
public class TestAgent extends MobileAgent {
        String[] route = new String[] { ... };
        public void run() {

                for( int i=0; i<10; i++ ) {
                        System.out.println(„leave platform“);
                        go( route[i] );
                        System.out.println(„arrived on platform“);
                        // do something
                }
        }
}
```

- höherer Übertragungsaufwand durch Ausführungszustand
- komfortabelste Lösung für den Programmierer
- sehr schwer im Agentensystem zu implementieren



Implementierung starker Migration

- Änderung der virtuellen Maschine von Java
  - die Java Virtual Machine wird modifiziert, so dass der auf den Call-Stack zugegriffen werden kann
  - Migration ist schnell
  - keine Vergrößerung des Quelltextes
  - schwierig zu implementieren
  - schlecht zu warten, bei neuer Version der JVM muss angepasst werden
  - ermöglicht auch „checkpointing“
- Source Code Transformation
  - keine Veränderung der Java Virtual Machine
  - leicht zu warten und an neue Versionen anzupassen
  - Quelltext wird größer (Faktor 2 bis 3)
  - Ausführungszeit wird länger (Faktor 1,5 bis 2)
  - ermöglicht kein „checkpointing“

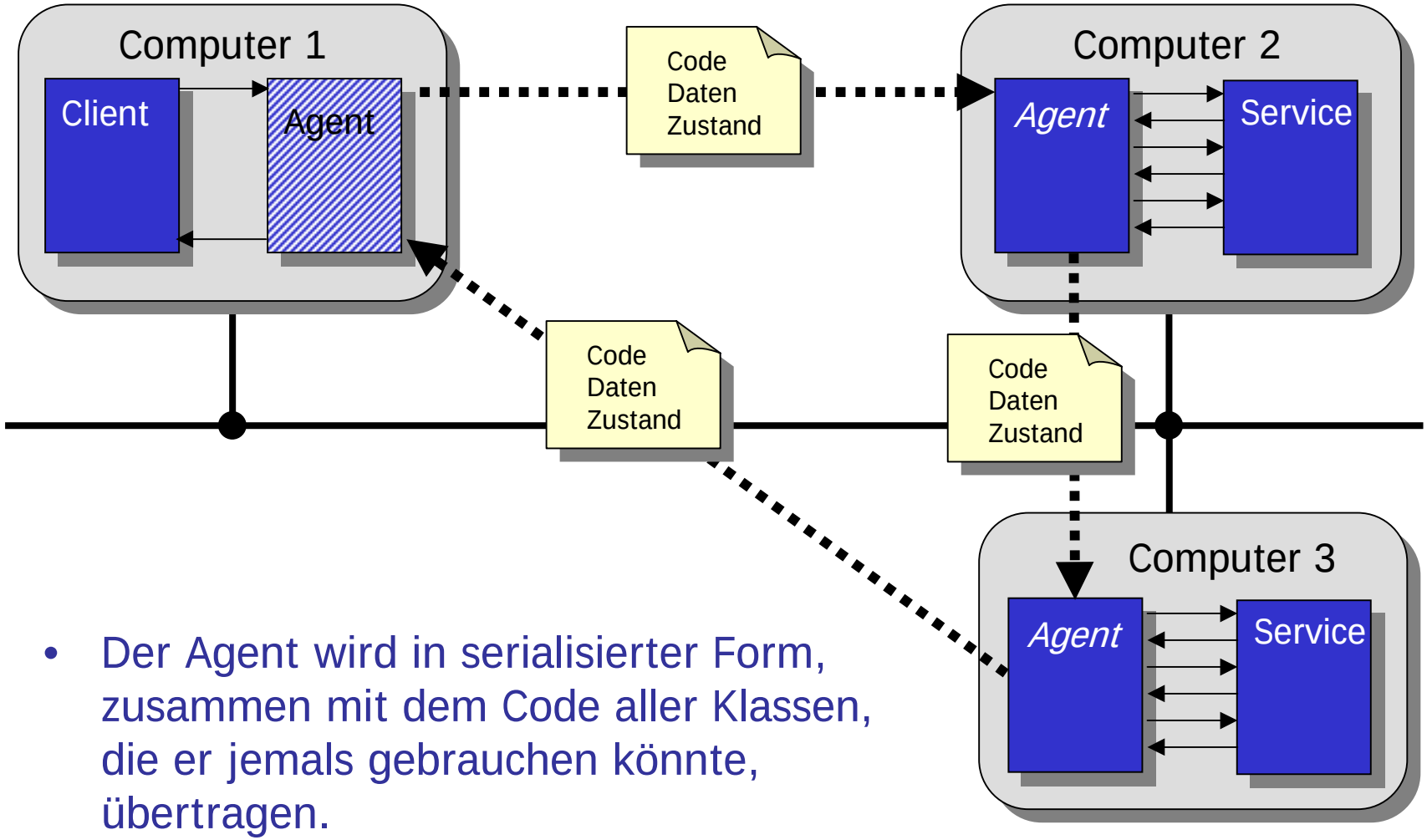


Migrationsmodell (2)

- Sicht des Programmierers
- Migrationsart: stark/schwach
  - Ziel der Migration wird vorgegeben/kann selbst definiert werden
  - Kommandos zur Migration: go, retract, meet
- Sicht auf die Migrationsstrategie
  - *was* migriert *wohin*?
    - migrieren alle oder nur bestimmte Daten
    - migrieren alle Klassen, nur einige oder nur bestimmte einzelne Methoden/Befehle
  - von wo können benötigte Klassen nachgeladen werden
- Sicht auf die Übertragungsstrategie
  - welches Protokoll wird verwendet?
  - Datenkomprimierung? Verschlüsselung?
  - Transaktionsmanagement?



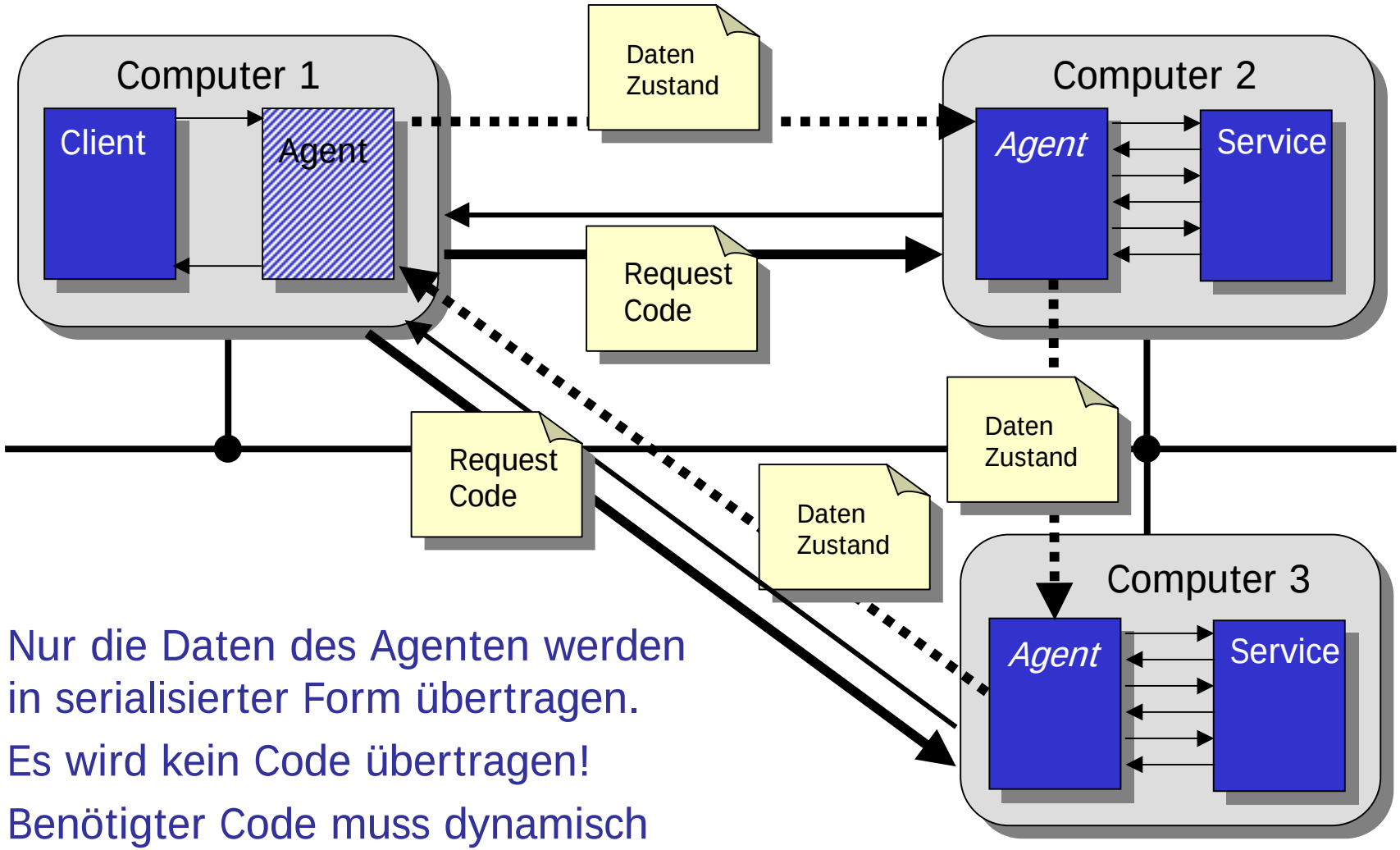
Push-Technik



- Der Agent wird in serialisierter Form, zusammen mit dem Code aller Klassen, die er jemals gebrauchen könnte, übertragen.



Pull-Technik



- Nur die Daten des Agenten werden in serialisierter Form übertragen.
- Es wird kein Code übertragen!
- Benötigter Code muss dynamisch nachgeladen werden.



Migrationsstrategien von Code

- Übertragung der Klassen mit den Daten (Push-Technik)
  - Voyager 2.0, Ara, Extended Facile, AgentTCL, Telescript
  - *ubiquitous classes* werden nicht übertragen (z.B. java.*)
  - schnell (nur eine große Übertragung)
  - Netzverbindung wird danach garantiert nicht mehr gebraucht
  - Klassen werden evtl. doppelt übertragen
  - es werden Klassen übertragen, die evtl. gar nicht gebraucht werden
- Übertragung der Klassen auf Anfrage (Pull-Techniken)
  - Java RMI, Voyager 1.0, MOLE, Sumatra
  - nur die Daten werden übertragen, Klassen werden nachgeladen
  - es werden nur die tatsächlich benötigten Klassen übertragen
  - bereits auf der Zielplattform vorhandene Klassen werden nicht geladen
  - Netzverbindung muß offen bzw. verfügbar sein
  - langsam (mehrere kleine Übertragungen)



Migrationsmodell (3)

- Sicht des Programmierers
  - Migrationsart: stark/schwach
  - Ziel der Migration wird vorgegeben/kann selbst definiert werden
  - Kommandos zur Migration
- Sicht auf die Migrationsstrategie
  - *was* migriert *wohin*?
    - migrieren alle oder nur bestimmte Daten
    - migrieren alle Klassen, nur einige oder nur bestimmte einzelne Methoden/Befehle
  - von wo können benötigte Klassen nachgeladen werden
- Sicht auf die Übertragungsstrategie
  - welches Protokoll wird verwendet?
  - Datenkomprimierung? Verschlüsselung?
  - Transaktionsmanagement?



Übertragungsstrategien

- Übertragungsstrategien legen fest, wie der Agent (Zustand + Code + Variablen) verschickt wird.
- Adresse der Zielplattform
  - bei den meisten TCP/IP basierten Systemen: IP + Portnummer
  - über den Namen des Agentenservers
    - macht einen Name Service notwendig
      - bei RMI der „ServerName“, z.B. „daisy.uni-jena.de/Webster“
- Viele existierende Systeme arbeiten mit Java RMI
- Andere Protokolle denkbar
  - UDP
  - TCP/IP
  - Mobile IP
  - SSL (Secure Socket Layer)
  - RMI/HTTP Tunneling
  - SMTP (Mail-Versand)



Migrationsmodell - Zusammenfassung

- Es existiert zur Zeit kein einheitliches Migrationsmodell.
- Migrationsart:
  - schwache Migration mit beliebigem Einsprung quasi Standard
    - guter Kompromiss zwischen Performance und Convenience
    - brauchen wir starke Migration überhaupt?
  - Unterschiedliche Programmiermodelle, Migrationsbefehle, Fehlerbehandlung
- Migrationsstrategie:
  - existierende Systeme benutzen Pull- oder Push-Techniken
  - Aglets benutzt eine Mischung
    - „push“ alle lebenden Klassen, „pull“ alle anderen
  - welche Migrationsstrategie ist die beste? schnellste?
  - Sollte der Agent die Strategie adaptieren können?
- Übertragungsstrategie:
  - meist wird nur eine Übertragungsstrategie angeboten, meist RMI
  - Flexibilität wünschenswert (Beispiel Grasshopper, Tracy)



Kommunikation zwischen Agenten

- Kommunikation zwischen Agenten ist eine grundlegende Aufgabe des Agentensystems

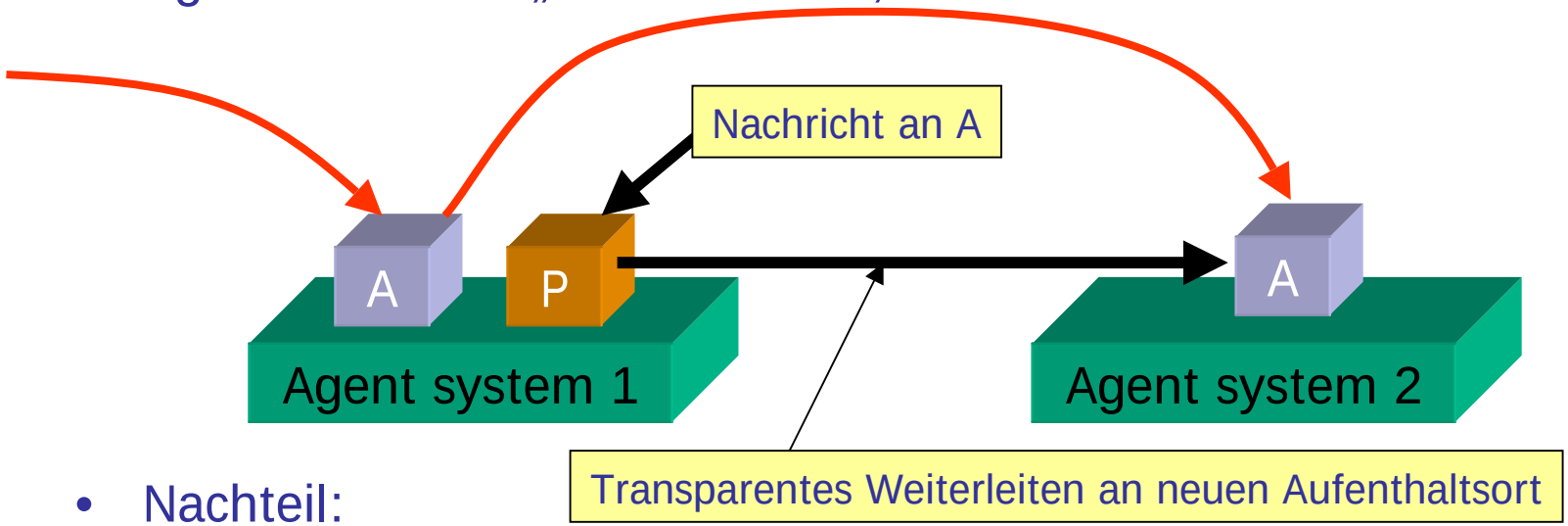
- Techniken:
  - anonyme Kommunikation über TupleSpaces, Blackboards o.ä.
  - direkter Methodenaufruf (Autonomie, Sicherheit)
  - synchrone oder asynchrone Nachrichten
  - peer-to-peer oder multicast
  - lokaler Nachrichtenaustausch / globaler Nachrichtenaustausch

- Probleme
  - Wie finde ich meinen Partner?
  - Der Partner migriert während der Migration.



Proxies

- Lösung für beide Probleme
  - vom Partner muss nur der Name und ein (alter) Aufenthaltsort bekannt sein
  - funktioniert auch, wenn Partner zwischenzeitlich migriert
- Agent hinterläßt „Stellvertreter“, der Nachrichten weiterleitet



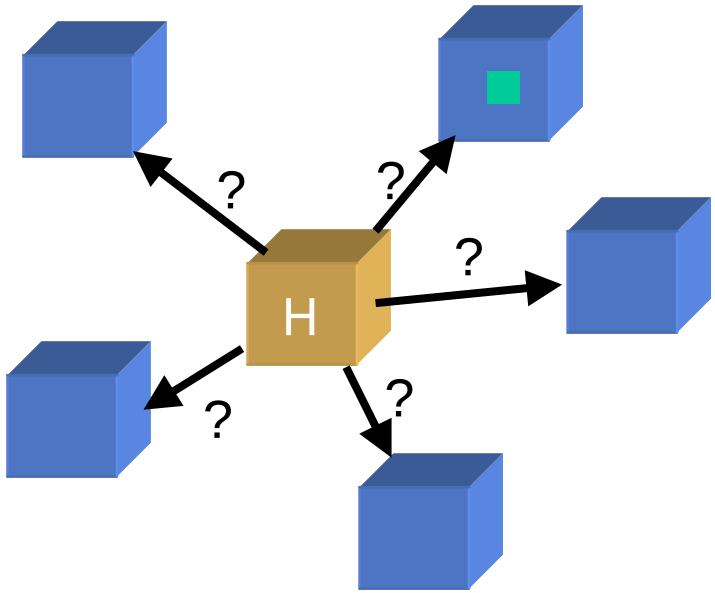
- Nachteil:
  - Proxy explosion
  - wann kann ein Proxy gelöscht werden?



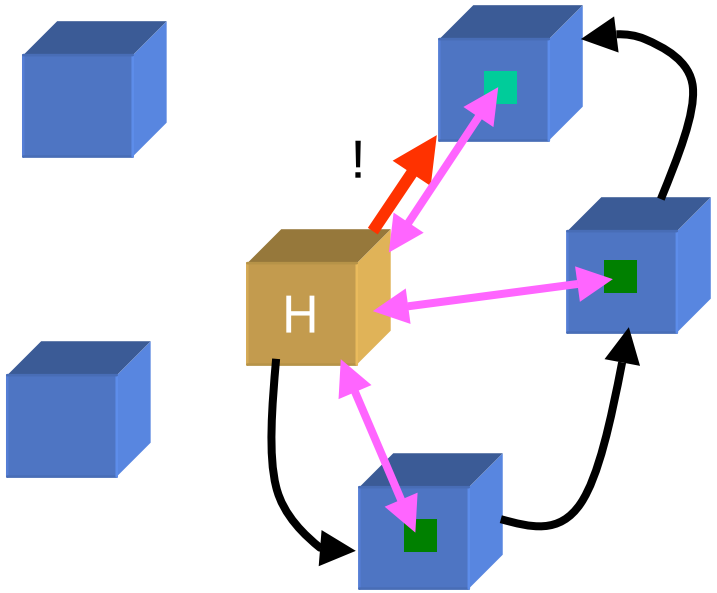
Kontrolle von mobilen Agenten

- Wie finde ich einen mobilen Agenten?
  - Ich möchte ihm eine Nachricht schicken.
  - Ich möchte ihn zurückholen.
  - Lebt er überhaupt noch?

Broadcast / Search



Central Registration





Weitere Kontrollprobleme

- Bestimmung von Waisen
  - ein Waise ist ein mobiler Agent, dessen Auftraggeber nicht mehr existiert
  - ein Waise verbraucht Rechenzeit, Bandbreite usw. ohne Nutzen und sollte daher beendet werden
  - Lösungsidee:
    - zentrale Registrierung
    - in regelmäßigen Abständen prüft der Agent, ob sein Auftraggeber noch lebt
- Terminierung
  - ein Agent wurde fehlerhaft programmiert oder definiert
  - Lösungsidee:
    - Agent lokalisieren und durch einen zweiten Agenten beenden
    - Energiekonzept
      - der Agent muss sich nach einer gewissen Zeit (time-to-live, TTL) wieder bei seiner Heimatplattform melden



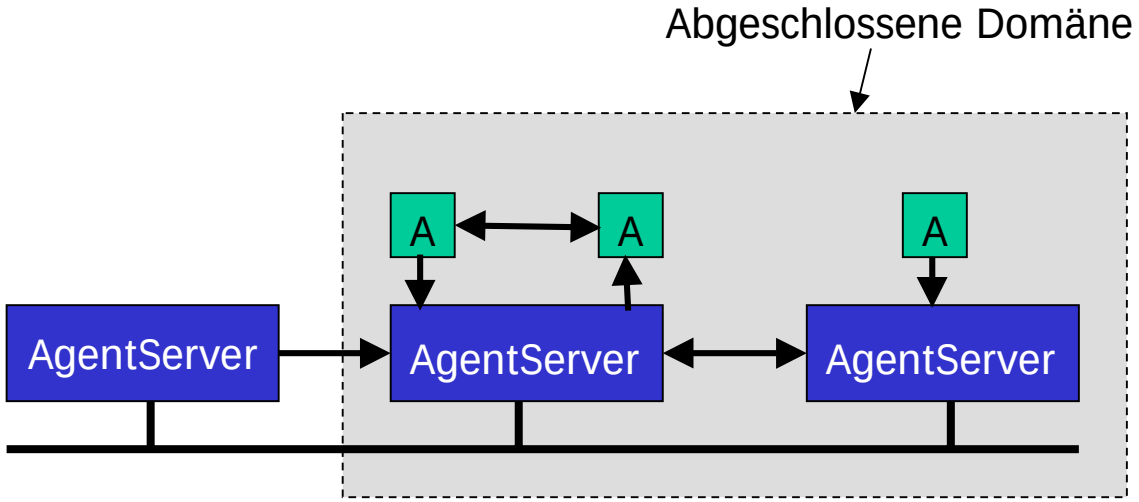
Sicherheit bei mobilen Agenten

- Warum ist Sicherheit wichtig?
  - Offene Systeme, bei dem jeder Benutzer
    - Agenten programmieren kann
    - neue Agenten in das Agentensystem einfügen kann
    - einen Agentenserver betreiben kann
- Agenten machen kritische Operationen
  - Zugriff auf lokale Ressourcen der Agentenserver
- Agenten besitzen sensible Daten
  - elektronisches Geld
  - Produktinformationen
- Agenten müssen sich vor Manipulationen schützen
- Agentenserver müssen sich vor bösartigen Agenten schützen



Angriffsmöglichkeiten

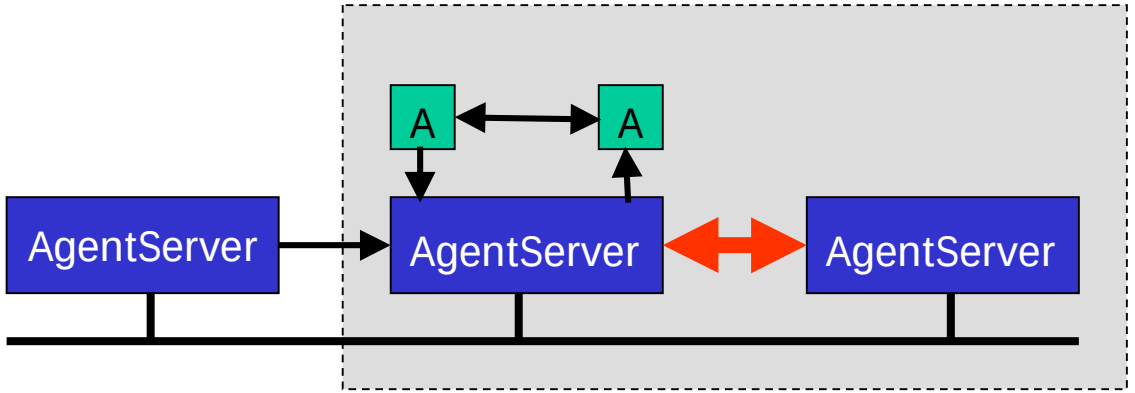
- Ausspähen und Verändern von Daten während der Migration eines Agenten.
- Bösartige Agenten greifen das Agentensystem, den Host oder andere Agenten an.
- Agenten geben eine andere Identität vor.
- Bösartige Agentensysteme spionieren Agenten aus.
- Agenten werden von nicht-autorisierten Servern geschickt .





Übertragung der Agenten

- Ziele
  - Confidentiality
    - Daten können nicht gelesen werden
    - Verschlüsselung
  - Integrity
    - Datenmanipulationen werden erkannt
  - Authentification
    - Überprüfung der Identität
    - Austausch von Zertifikaten
    - Digitale Signaturen
- Lösung
  - Verschlüsselung
    - Secret Key Encryption
      - schnell
      - schwierig zu verwalten
    - Public Key Encryption
      - langsam
      - einfach zu verwalten
  - Secure Socket Layer
    - legt sich transparent über TCP
    - löst alle drei Ziele





Bösartige Agenten

- Wollen das Agentensystem attackieren, ausspionieren
- Lösung: Java Sandbox
  - nur unbedenkliche Befehle werden ausgeführt
  - bei sicherheitskritischen Befehlen wird anhand von „grants“ entschieden, ob dieser Code ausgeführt werden darf
- kann als gelöst angesehen werden

- Wollen andere Agenten ausspähen
- Lösung
  - Agenten gegeneinander schützen
  - keine Referenzen auf andere Agenten
  - innerhalb des Agentenservers zu realisieren

- Geben vor, ein anderer zu sein
- Lösung
  - Agent digital signieren
    - mit dem privaten Schlüssel des Benutzers
    - mit dem privaten Schlüssel des Agentenservers
  - Was wird signiert?
    - Code
    - Daten



Bösartige Agentensysteme

- Bösartige Agentensysteme attackieren Agenten, in dem
  - der Agent nicht ausgeführt wird, sondern getötet wird
  - der Agent wird nicht korrekt ausgeführt wird
  - der Agent nicht migriert oder an einen falschen Ort geschickt wird
  - ...

- Beispiel: ein Agentensystem manipuliert die gefundene Preisliste des Einkaufsagenten, so dass die eigene Plattform den geringsten Preis fordert

- Heutige Ansätze
  - Schutz vor solchen Attacken durch vertrauenswürdige Hosts
    - keine Lösung
  - Manipulationen entdecken
  - Manipulationen verhindern
    - kryptographische Hardware



Schutz von „geschlossenen“ Systemen

- In geschlossenen Systemen garantieren sich die Teilnehmer gegenseitig gutes Verhalten.
  - Agenten vertrauen den Agentenservern
  - Agentenservern vertrauen Agenten
- Wie kann man gewährleisten, dass keine Spione eingeschleust werden?
  - Ansatz 1: Übertragung wird über Schlüssel geschützt, den nur Agentensysteme in der Gruppe kennen.
  - Ansatz 2: Agenten werden nur von bestimmten Plattformen akzeptiert.
    - Einfach zu implementieren über Java Security Manager.



Sicherheit - Zusammenfassung

- Anforderungen an Sicherheit sind von der Applikation abhängig
- in „geschlossenen“ Systemen reicht eine sichere Übertragung und Schutz vor „Eindringlingen“ durch Authentifizierung
- in „offenen“ Systemen mit unkritischen Diensten reicht der Schutz der Plattform durch Sandboxes aus
- wenn elektronische Bezahlmethoden usw. hinzukommen, muss auch der Agent geschützt werden können

- Literaturhinweis:
  - Giovanni Vigna: Mobile Agents and Security. Vol. 1419 of Lecture Notes in Computer Science. Berlin: Springer-Verlag, 1998.



Übersicht Produkte und Projekte

- Mobile Agenten sind ein hochaktuelles Forschungsthema
- zahlreiche Forschungsgruppen arbeiten an Prototypen von Agentensystemen
- nur vereinzelt werden Standards berücksichtigt, daher so gut wie keine Kommunikation zwischen diesen Systemen möglich
- zur Zeit (je nach Zählweise und Aktualität) zwischen 40 und 70 Systeme weltweit
- nur wenige können als „Produkt“ bezeichnet werden
  - Aglets von IBM (Japan)
  - Grasshopper von IKV++ GmbH (Berlin)
  - Jumping Beans von AdAstra (USA)
  - (Voyager von ObjectSpace)
- Forschungsprojekte (Auswahl):
  - D‘Agents (Dartmouth, Hanover, USA)
  - Mole (Universität Stuttgart)
  - Tracy (Friedrich-Schiller-Universität Jena)



Vergleich Produkte/Projekte

- Existierende Systeme unterscheiden sich durch
  - die verwendete Sprache
    - Java, C, C++, Tcl, Facile, Python, Scheme,
    - das verwendete Migrationsmodell
      - schwache oder starke Migration aus Programmierersicht
      - Push oder Pull code
      - TCP/IP, RMI oder eigenes Protokoll als Übertragungsstrategie
  - Kommunikationstechniken
    - synchrone / asynchrone Nachrichten
    - anonyme Nachrichten
  - Sicherheitsmodell
    - Schutz der Plattform
    - Schutz des Agenten
  - Unterstützung von Standards
    - MASIF
    - FIPA



Aglets, IBM (Japan)

- Homepage: http://www.trl.ibm.co.jp/aglets/
- Wurde kürzlich auf ein Open Source Projekt umgestellt

- Basiert auf Java 1.1.x
  - eine Portierung auf Java 1.2 ist nicht geplant

- Schwache Migration mit festem Einsprung
- Pull Techniken (entweder Klasse oder JAR Archiv)
- Delegation-based Event Model (Clone, Mobility, Persistence)
- Synchrone und asynchrone Kommunikation, ortstransparent!
- Schutz der Plattform vor böswilligen Agenten durch Java Sandbox
- unterstützt MASIF Standard





Grasshopper

- Homepage http://www.grasshopper.de
- Entwickelt von GMD FOKUS und IKV++ GmbH, Berlin

- lauffähig unter Java 2

- schwache Migration
- verschiedene Übertragungstechniken
  - Java RMI, TCP, SSL, CORBA IIOP
- ortstransparente Kommunikation
- Verwaltung logischer Netzwerke durch „region registry“

- Unterstützt sowohl den MASIF als auch den FIPA Standard

GRASSHOPPER
THE AGENT PLATFORM

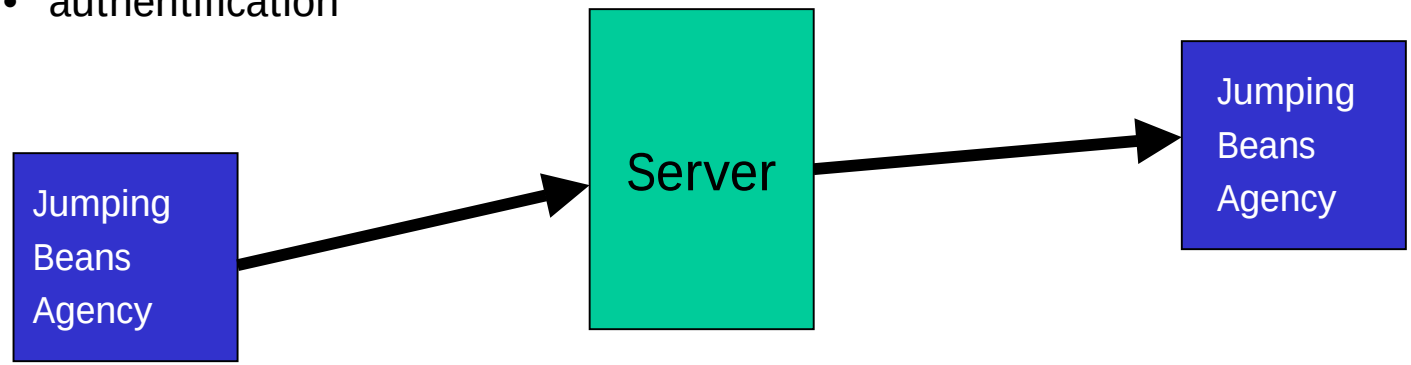


Jumping Beans

- Homepage: http://www.jumpingbeans.com
- Entwickelt von AdAstra Inc. (USA)
- basiert auf Java 1.1.x
- schwache Migration



- Architektur: Client/Server
  - Migration und Kommunikation immer über den Server
  - zentrale Verwaltungskomponente
    - agent tracking
    - authentification





Einführung in das Agentensystem Tracy





Übersicht Tracy (1)

- Entwickelt am Lehrstuhl Softwaretechnik des Institutes für Informatik der Friedrich-Schiller-Universität Jena
- Homepage: http://tracy.informatik.uni-jena.de

- Java 2 ohne Veränderung der Java Virtual Machine

- Tracy ist Applikation und Komponente
  - eigenständiges Programm
    - über eine GUI zu administrieren
    - über RMI und CORBA zu programmieren
  - eingebettete Komponente in Java Programmen



Übersicht Tracy (2)

- Kommunikation
  - asynchroner Nachrichtenaustausch
  - Blackboard als zentrales Medium zum Austausch von Infos
    - Agent-Agent, Agent-Nutzer, Agent-Applikation
    - hierarchisch angeordnet
  - keine Kommunikation über Rechnergrenzen
    - die Philosophie von Tracy besteht darin, stets zu einem Ort zu migrieren, wenn dort kommuniziert werden soll
  - kein direkter Methodenaufruf zwischen Agenten möglich
    - ein Agent wird stets über seinen Namen angesprochen



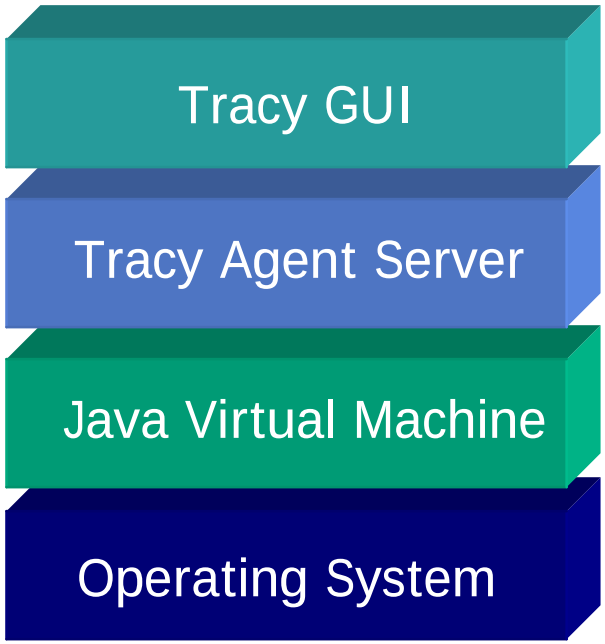
Übersicht Tracy (3)

- Migration
  - schwache Migration mit beliebigem Einsprung
  - Migrationsstrategie programmierbar
    - existierende Strategien können ausgewählt werden
    - neue können dynamisch vom Agenten programmiert werden
  - Übertragungsstrategie auswählbar
    - TCP, RMI, UDP, SSL, SMTP, RMI/HTTP
    - neue Übertragungsstrategien können programmiert werden

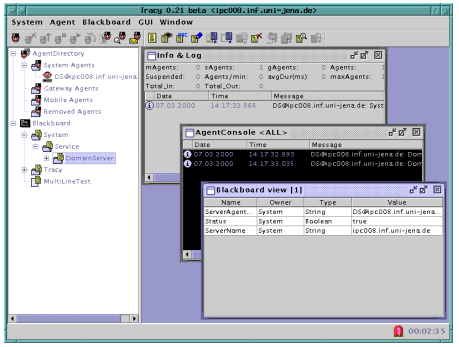
- Sicherheit
  - drei Typen von Agenten
    - GatewayAgent zur Kommunikation zu externen Anwendungen
    - SystemAgent zur Kommunikation mit dem Host
    - MobileAgent zur Migration
  - Schutz des Agentenservers durch Sandbox
  - Schutz des Agenten bei der Übertragung durch SSL
  - kein Schutz des Agenten während der Ausführung



Architektur von Tracy (1)

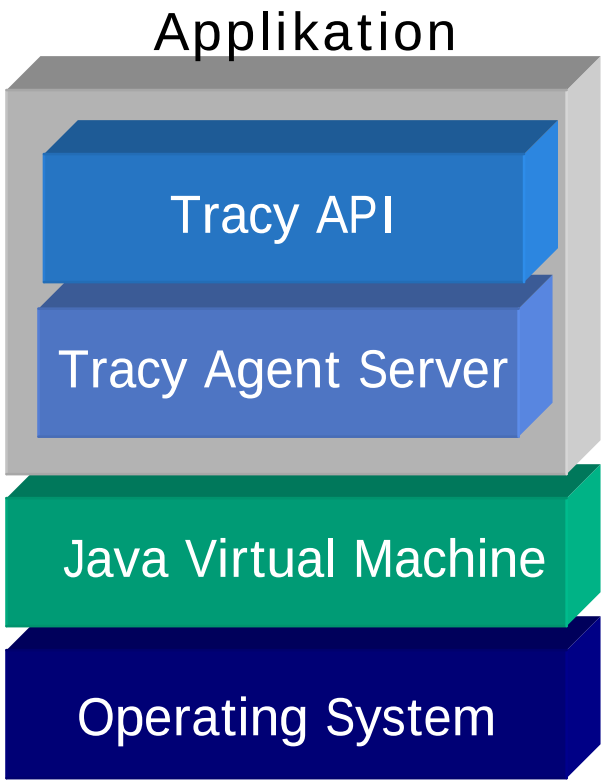


- Der Tracy Agent Server stellt die Laufzeitumgebung für alle Arten von Agenten bereit.
- Der Tracy Agent Server ist als eigenständige Anwendung lauffähig.
- Die Tracy GUI ist eine graphische Benutzeroberfläche zur vollständigen Administration eines Tracy Agent Servers.





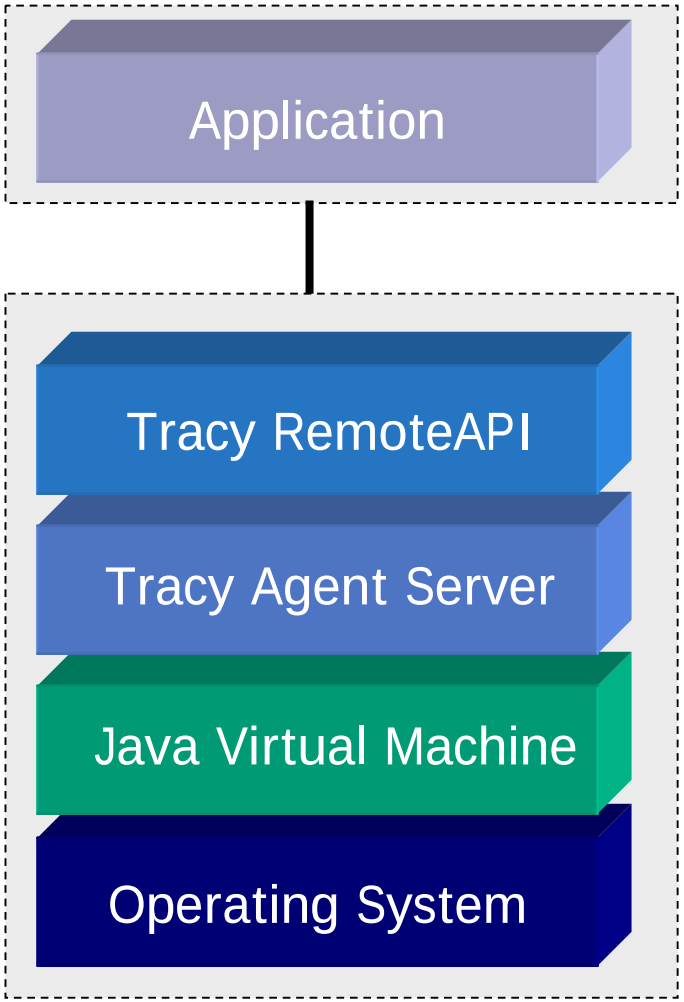
Architektur von Tracy (2)



- Ein Tracy Agent Server kann in eine Java-basierte Applikation als Software-Komponente eingebettet sein.
- Die Kommunikation von Applikation zum Agent Server erfolgt über die Tracy API.
- Die Tracy API erlaubt die vollständige Steuerung eines Agent Servers.
  - Agent starten, stoppen
  - Nachrichten an Agenten schicken
  - Blackboard schreiben, lesen, beobachten



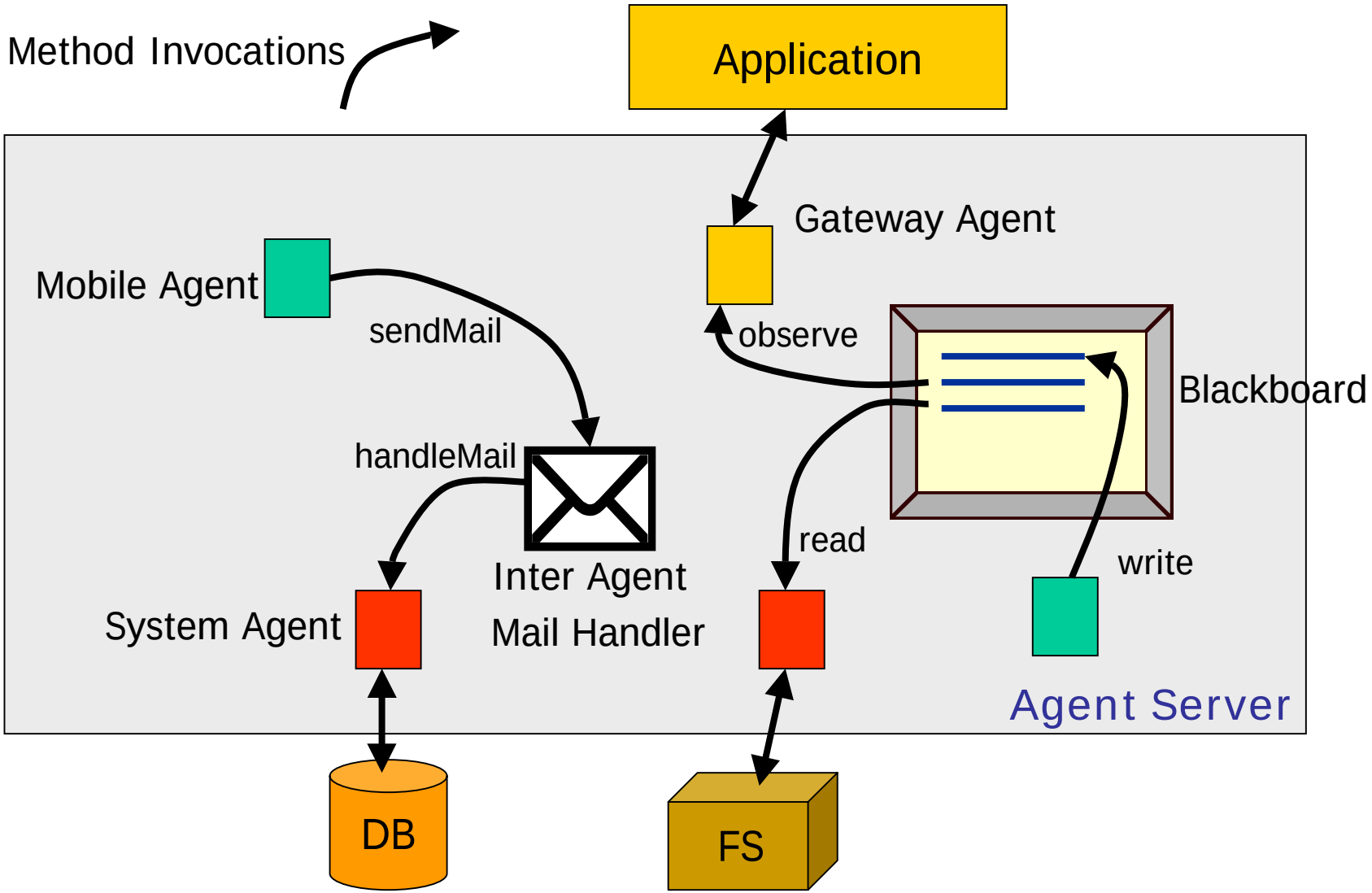
Architektur von Tracy (3)



- Die Tracy RemoteAPI öffnet die TracyAPI, so dass über Remote Method Calls auf einen Tracy Agent Server zugegriffen werden kann.
- Damit können Applikationen auch auf anderen Hosts liegen.
- Auch die Tracy GUI kann über die Tracy RemoteAPI einen Tracy Agent Server administrieren.

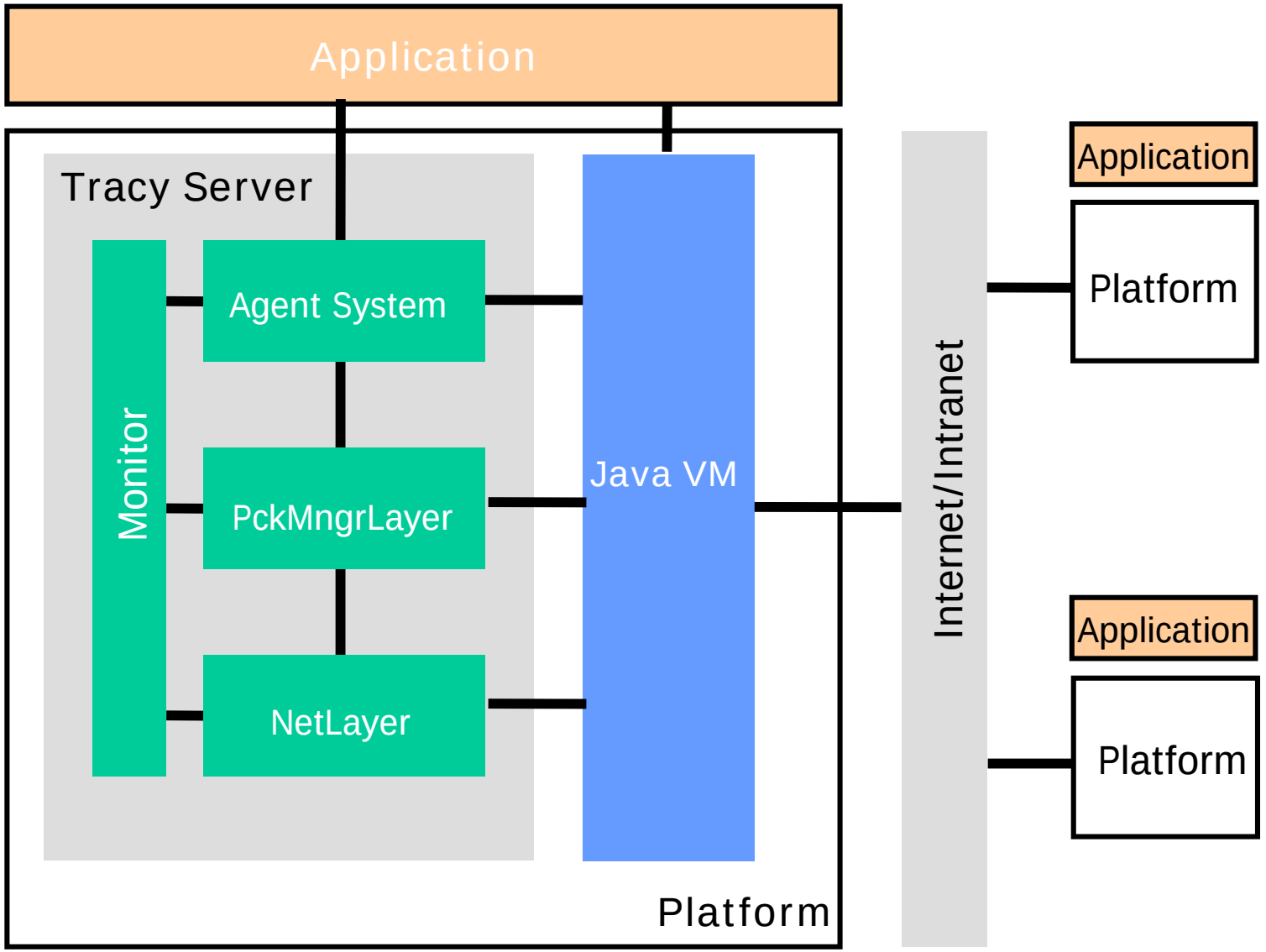


Aufbau eines Agent Servers





Architektur eines Agent Servers

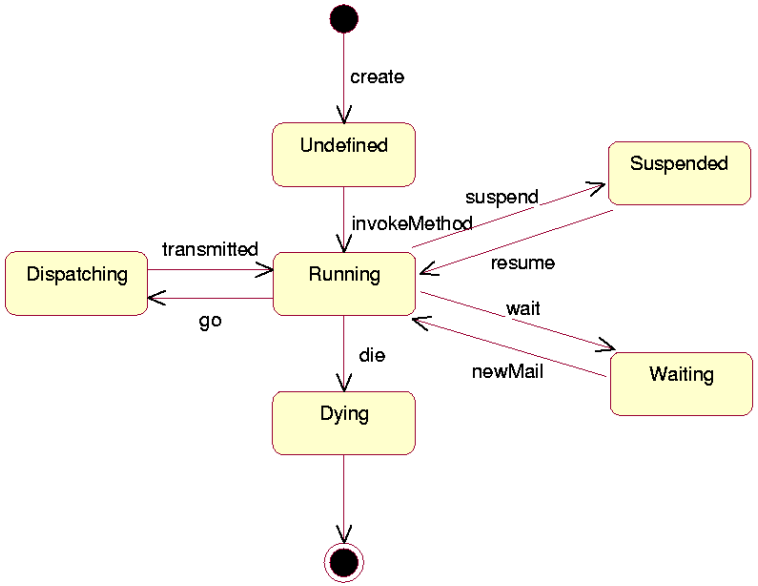




Eigenschaften eines Agenten

- Jeder Agent besitzt einen global eindeutigen Namen
  - „Agent @ Hostname“
  - z.B. „Bond@daisy.uni-jena.de“
- Hostname ist hierbei der Name der *Heimatplattform*

- Jeder Agent durchläuft einen Lebenszyklus





Klasse Agent

```
public abstract class Agent {

        public Agent( String name ) { ... }

        public startAgent();
        protected void waiting() { ... }
        protected void die() { ... }

        // console output
        protected void printMessage( String msg ) { ... }
        ...
        // methods for messaging
        protected void sendMessage( String agent, Message msg ) {}
        ...
        // method for blackboard access
        protected void putEntity( String agent,
                                  String entity,
                                  BlackboardContent content );
        ...
}
```



Beispiel: MyFirstAgent

- Für jeden der drei Agententypen gibt es eine Ableitung der Klasse „Agent“.
- Beispiel für einen Systemagenten:

```
public class MyFirstAgent extends SystemAgent {

        public MyFirstAgent( String name ) {
                super( name );
        }

        public void startAgent() {

                printMessage( „Agent started“ );
                waiting();
        }

}
```



Nachrichtenversand

- Asynchroner Nachrichtenaustausch zwischen Agenten
- Agenten werden über ihren Namen angesprochen
- Nachricht besteht aus Typ und Inhalt

```
public void func() {

   Message msg = new Message( „go“, „pluto.uni-jena.de“);
   sendMessage( „Bond@daisy.uni-jena.de“, msg );
}
```

- den Namen eines Agenten erhält man
  - vom Agent Directory
  - über das Blackboard, auf dem sich Agenten für Dienste eingetragen haben



Nachrichtenempfang

- Agent wird über neue Nachrichten informiert
- es bleibt dem Agenten überlassen, ob und wie er auf Nachrichten reagiert

```
public class MyFirstAgent extends SystemAgent {
  ...

  protected void handleLastMail( Message inMsg ) {
        if (inMsg.getTyp().compareTo(“read") == 0) {
          // process read command
        } else
          printError("Unknown message type!");
   }

   ...
}
```

- Message enthält den Absender, so dass einfach geantwortet werden kann



Blackboard

- Das Blackboard bietet eine Möglichkeit zur anonymen Kommunikation zwischen Agentengruppen.
- Daten werden auf dem Blackboard unter einen Namen abgelegt.
  - Namen sind hierarchisch aufgebaut
    - Beispiel: System/OS/Version
  - Bei den Daten sind verschiedene Typen möglich:
    - String, XML, Grafik, ByteArrays
- Daten können unter ihrem Namen wiedergefunden werden.
- Ein Agent kann sich als Abonnent bei einem Blackboard-Eintrag anmelden.
  - Der Agent wird über jede Änderung dieses Eintrages und der darunterliegenden Einträge mit einer Message informiert.



Daten auf Blackboard ablegen

```
public class MyFirstAgent extends SystemAgent {

   ...
   public void runRemote() throws AgentExecutionException {
        try{
               putEntity(     name,
                              “/System/TestEntry”,
                              new BBStringContent(“My Content”));

        }
        catch( BlackboardException be ) {
               printError(be.getMessage());
        }

   }

}
```



Daten vom Blackboard lesen

```
public class MyFirstAgent extends SystemAgent {

   public void runRemote() throws AgentExecutionException {

        try{
                BlackboardContent bbContent
                        = AS.getEntity(  name, “/System/TestEntry”);

                if (bbContent.getContentType().compareTo(“String")==0)
                        content = (String)bbContent.getContent();

        }
        catch( BlackboardException be ) {
                printError(be.getMessage());
        }
   }

   private String content = null;
}
```



Blackboard-Einträge beobachten

```
public class MyFirstAgent extends SystemAgent {

   public void runRemote() throws AgentExecutionException {
        try{
                getEntityWhenChanged( name, „/System/TestEntity“ );
        }
        catch( BlackboardException be ) {
                printError(be.getMessage());
        }
   }

   public void handleMail( Message msg ) {

        if( msg.getTyp().compareTo( Blackboard.LEAF_CHANGED ) == 0 ) {
                // do something
        } else if( msg.getTyp().compareTo( Blackboard.LEAF_DELETED ) == 0 ) {
                // do something else
        }

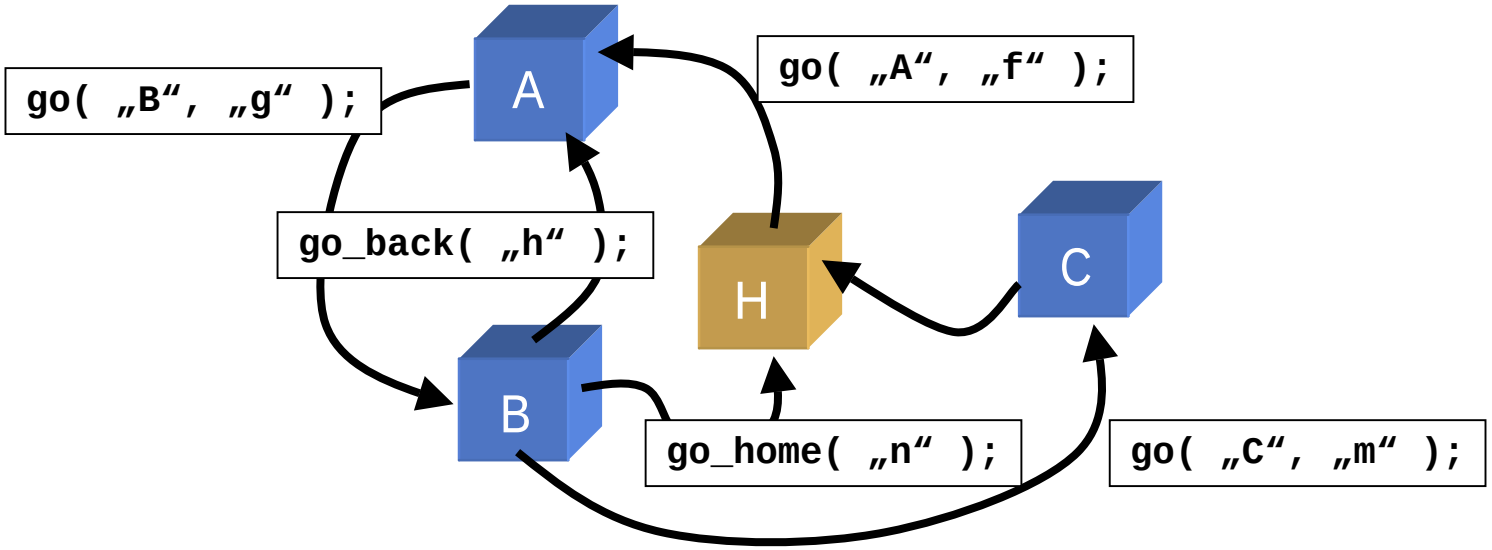
   }

}
```



Migration

- Tracy bietet eine schwache Form der Migration mit beliebigem Einsprung.
- Eine Migration wird durch den go-Befehl ausgelöst.
  - Dabei wird stets das Ziel und der Name der dort aufzurufenden Methode mitgegeben.
  - Wenn die Migration nicht erfolgreich ist, wird die Methode „migrationFailed“ auf der Ausgangsplattform aufgerufen.





Beispiel: Durchlaufen einer Route

```
public class Wanderer extends MobileAgent {

private String[] route = new String[] { „daisy.uni-jena.de“,
                                        „pluto.uni-jena.de“,
                                        „goofy.uni-jena.de“ };
private index = 0;

public void startAgent() throws AgentExecutionException {
        go( route[ index ], „startOnPlatform“ );
}

public void startOnPlatform() throws AgentExecutionException {
        // do something
        if( route.length > index )
                go( route[ index++ ], „startOnPlatform“ );
        else
                go_home( „finish“ );
}

public void finish() throws AgentExecutionException {
          die();
}
}
```



Programmierung der Migration

- Tracy bietet die Möglichkeit zur Steuerung der Migration
  - Migrationsstrategie
  - Übertragungsstrategie
- Mittel: MigrationProperties
  - fasst die Eigenschaften der Migration zusammen
    - Ziel, Methode, Migrationsstrategie, Übertragungsstrategie usw.

```
public class Wanderer extends MobileAgent {

private MigrationProperties migProps = null;

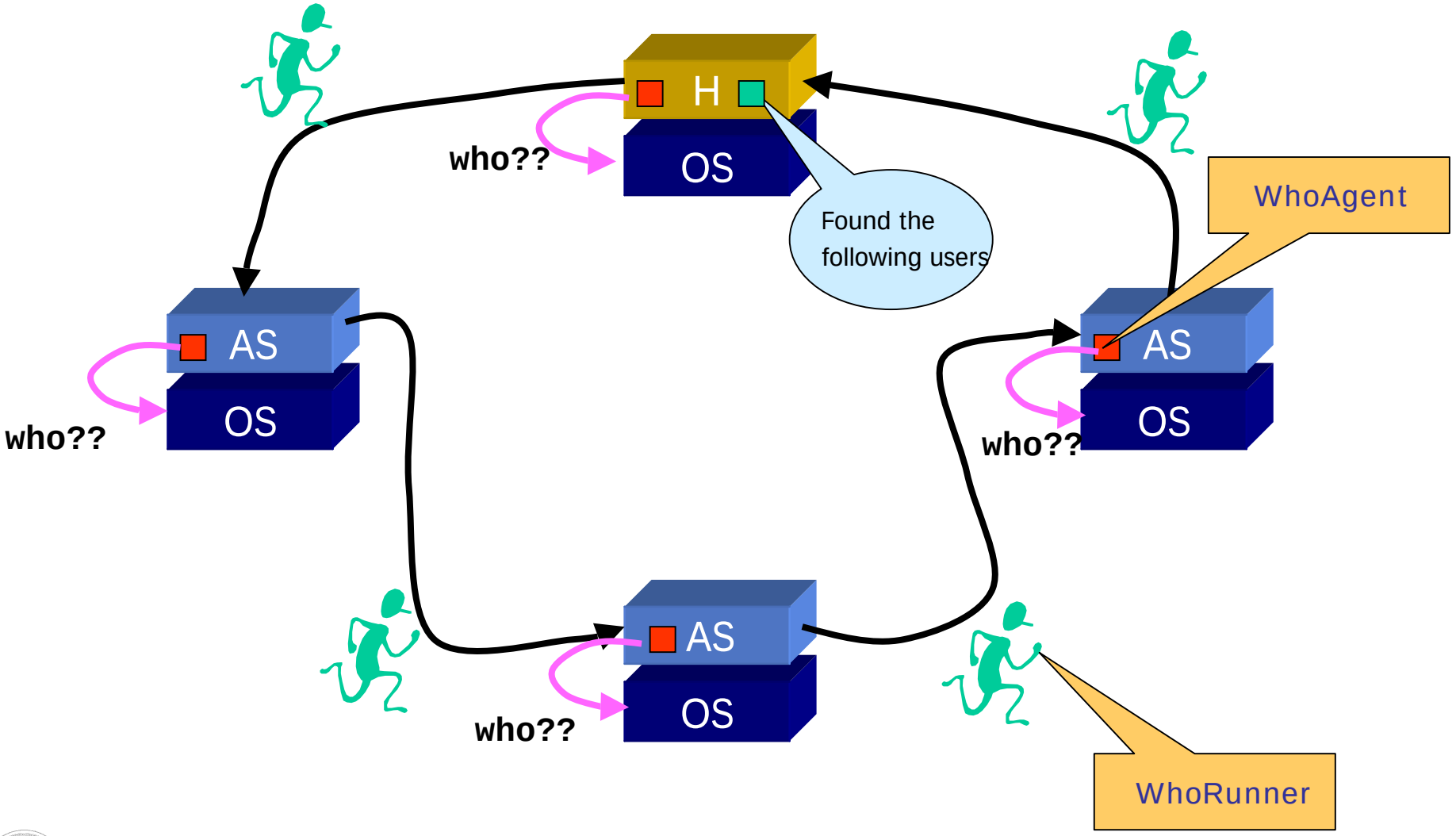
public void startAgent() throws AgentExecutionException {

        migProps.setMigrationStrategy( „PASSIVE“ );
        migProps.setTransmissionProperty( „strategy“, „TCP/SSL“ );
        migProps.setTransmissionProperty( „compress“, „yes“ );

        setDefaulMigrationStrategy( migProps );
        go( route[ index ], „startOnPlatform“ );
}
```



Beispielanwendung: Finde alle Nutzer



System Agent: WhoAgent

```
public class WhoAgent extends SystemAgent {

  protected void handleLastMail(Message inMsg) {

    if (inMsg.getTyp().equals("who")) {
      try {
        Process whoProcess = java.lang.Runtime.getRuntime().exec("who");
        java.io.InputStream is = whoProcess.getInputStream();
        java.io.InputStreamReader isr = new java.io.InputStreamReader( is );
        java.io.BufferedReader br = new java.io.BufferedReader( isr );

        // parse the input stream and generate one string with all logins

        try{
          Message reply = new Message( name, "reply-who", allLogins );
          sendMessage(inMsg.getSender(), reply );
        } catch (MessageQueueException e) {
          System.out.println(inMsg.getSender() + e.getMessage());
        }
      }
      catch( java.io.IOException e ) { ... }
   ...
```



Starten eines Agenten

- Tracy wird als Stand-alone Server gestartet
  - ein Agent kann beim Hochfahren des Servers automatisch gestartet werden
  - **`$ tracy agent.Who.class=examples.WhoAgent`**

- Tracy wird über eine grafische Oberfläche gesteuert
  - Agenten werden über einen Dialog gestartet



Starten eines Agenten über die API

- Tracy ist eingebettete Softwarekomponente
  - Agenten werden über die Tracy API gestartet

```
public class TestTracy {
        public static void main( String[] args ) {

        try {
                Tracy myTracyServer = new Tracy(„./tracy.config“);
                myTracyServer.startAgentFromFile(
                        „examples.WhoAgent“,
                        „Who“ );

        } catch( TracyException e ) {
                System.err.println( e.getMessage() );
        }
        }
}
```



Mobiler Agent: WhoRunner (1)

```
public class WhoRunner extends MobileAgent {

    public void startAgent() throws AgentExecutionException {
        startOnPlatform();
    }

    public void startOnPlatform() throws AgentExecutionException {
      String hostname = getHostName();
      try {
         sendMessage( "Who@"+hostname, "who", "" );
      } catch( MessageQueueException e ) {
         System.out.println( e.getMessage() );
      }
      waiting();
    }
```



Mobiler Agent: WhoRunner (2)

```
   protected void handleLastMail( Message inMsg )
                                    throws AgentExecutionException {
    if( inMsg.getTyp().equals("reply-who") ) {

      // merge two lists of logins:
      // allLogins and inMsg.getContent and store the result
      // in allLogins

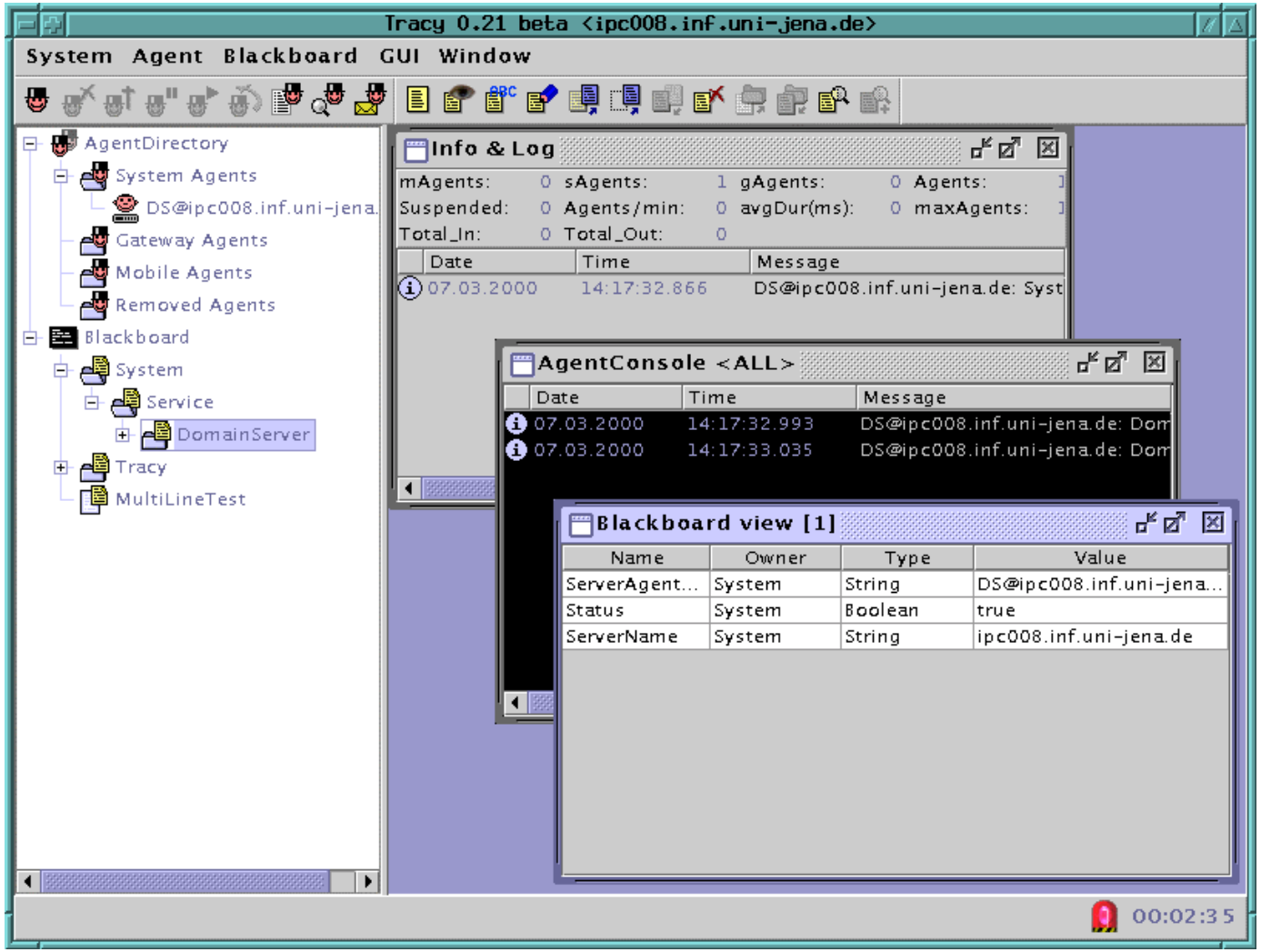
      if( route.length > index )
         go( route[ index++ ], "startOnPlatform" );
      else
         go_home( "finish" );
    }
  }

  public void finish() throws AgentExecutionException {
    printMessage( "Found the following users: " + allLogins );
    die();
  }

  protected String allLogins = new String();
  protected int index = 0;
  protected String[] route = new String[] { ... };
}
```



TracyGUI


Tracy 0.21 beta <ipc008.inf.uni-jena.de>
System Agent Blackboard GUI Window
AgentDirectory
System Agents
DS@ipc008.inf.uni-jena.
Gateway Agents
Mobile Agents
Removed Agents
Blackboard
System
Service
DomainServer
Tracy
MultiLineTest
Info & Log
mAgents: 0 sAgents: 1 gAgents: 0 Agents:
Suspended: 0 Agents/min: 0 avgDur(ms): 0 maxAgents:
Total_In: 0 Total_Out: 0
Date Time Message
07.03.2000 14:17:32.866 DS@ipc008.inf.uni-jena.de: Syst
AgentConsole <ALL>
Date Time Message
07.03.2000 14:17:32.993 DS@ipc008.inf.uni-jena.de: Dom
07.03.2000 14:17:33.035 DS@ipc008.inf.uni-jena.de: Dom
Blackboard view [1]
Name Owner Type Value
ServerAgent... System String DS@ipc008.inf.uni-jena...
Status System Boolean true
ServerName System String ipc008.inf.uni-jena.de
00:02:35



Mehr Informationen zu Tracy

- Technischer Bericht
  - P. Braun, Ch. Erfurth, W. Rossak: An Introduction to the Tracy Mobile Agent System. Technical Report No. 2000/24, Friedrich Schiller University Jena, Computer Science Department, 2000.
- Web-Seite
  - http://tracy.informatik.uni-jena.de
- Kontakt
  - Peter Braun, E-Mail: Peter.Braun@informatik.uni-jena.de



Weitere Informationen

- Dieses Tutorial
  - http://tracy.informatik.uni-jena.de/tutorials
- Web Seiten
  - http://tracy.informatik.uni-jena.de
  - http://www.agentlink.org
- Liste existierender Systeme
  - http://tracy.informatik.uni-jena.de/maslist.html
  - http://mole.informatik.uni-stuttgart.de/
- Bibliographien
  - http://tracy.informatik.uni-jena.de/links.html
- Bücher
  - G. Vigna: Mobile Code Security: LNCS 1419, Springer, 1998.
- Konferenzen
  - ASA/MA 2000 (Zürick, September 2000)

