Parallele und Verteilte Systeme - Jini

Jini - Schlüsselkonzepte

- *Ziele:*
  - Einfache Kommunikationsprotokolle
  - spontane Vernetzung
  - automatische Konfiguration
  - Services On Demand
  - Plug & Work

Leitlinien

- Keep it Small and Simple
- Potential der Java Technologie ausschöpfen
- echte Objektorientierung
- Daten und Prozesse verschieben, falls nötig
- Netzwerkfehler erkennen und beheben

Was ist Jini Technologie ?

- Einfaches Set verteilter Protokolle
- Föderation von Java Virtual Machines
- Modell für die verteilte Programmierung

Verteilte Realitäten

- **Verteilte Systeme sind *intrinsisch unterschiedlich* von lokalen Systemen:**
  - **Latenz der Netzwerkanfragen**
  - **Speicherverwaltungsfragen**
  - **Nebenläufigkeitsfragen**
  - **partielle Ausfälle und Fehlertoleranz**

Geschichte von RMI und Jini Technologie

- Sun Laboratories Research (pre-1994)
- „A Note on Distributed Computing“ (1994) [Waldo ..]
- Java Technologie & Web (1995)
- Java RMI Building Blocks (1996)
- Jini Technologie (1998)
- Weiterentwicklung: RPC zu ORBs zu Jini Technologie

RMI Building Blocks

- gleiche Datentypen netzwerkweit
- sicheres Übermitteln von Objekten (Serialisierung)
- Erhalt der Objekthierarchie
- verteilter Polymorphismus
- verteilter Garbage Collector
- remote Objektaktivierung

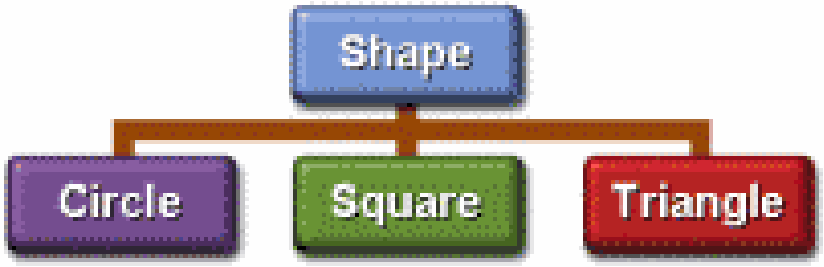
RMI ist *Object-orientiert?*

- virtuelle Maschinen machen das Netzwerk homogen
- es wird ein einziges Java *Objekt* Modell eingesetzt
- die Kapselung von Objekten bleibt erhalten
- keine Verluste durch unterschiedliche Objekt-Subtypen
- Objekte können sicher über das Netzwerk übertragen werden

Verteilter Polymorphismus

- Objekte können zum Aufrufer verschoben werden
- alle Java Objekttypen können eingesetzt werden
- ermöglicht den Einsatz von OO Entwurfsmuster
- Applikationen werden dynamisch erweiterbar
- die aktuelle Ausführung kann verteilt werden

Klassischer Polymorphismus



- class Shape {

...

Shape sh = new Shape();

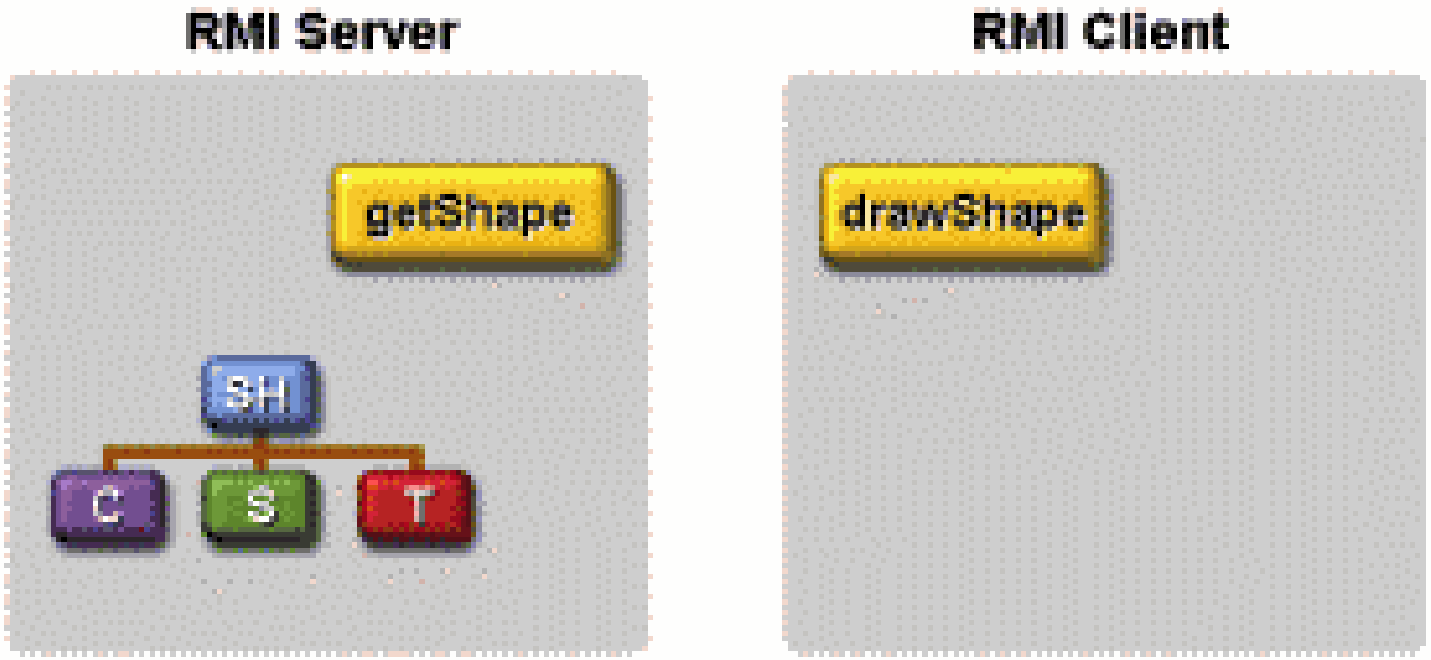
...

sh.drawShape();

...

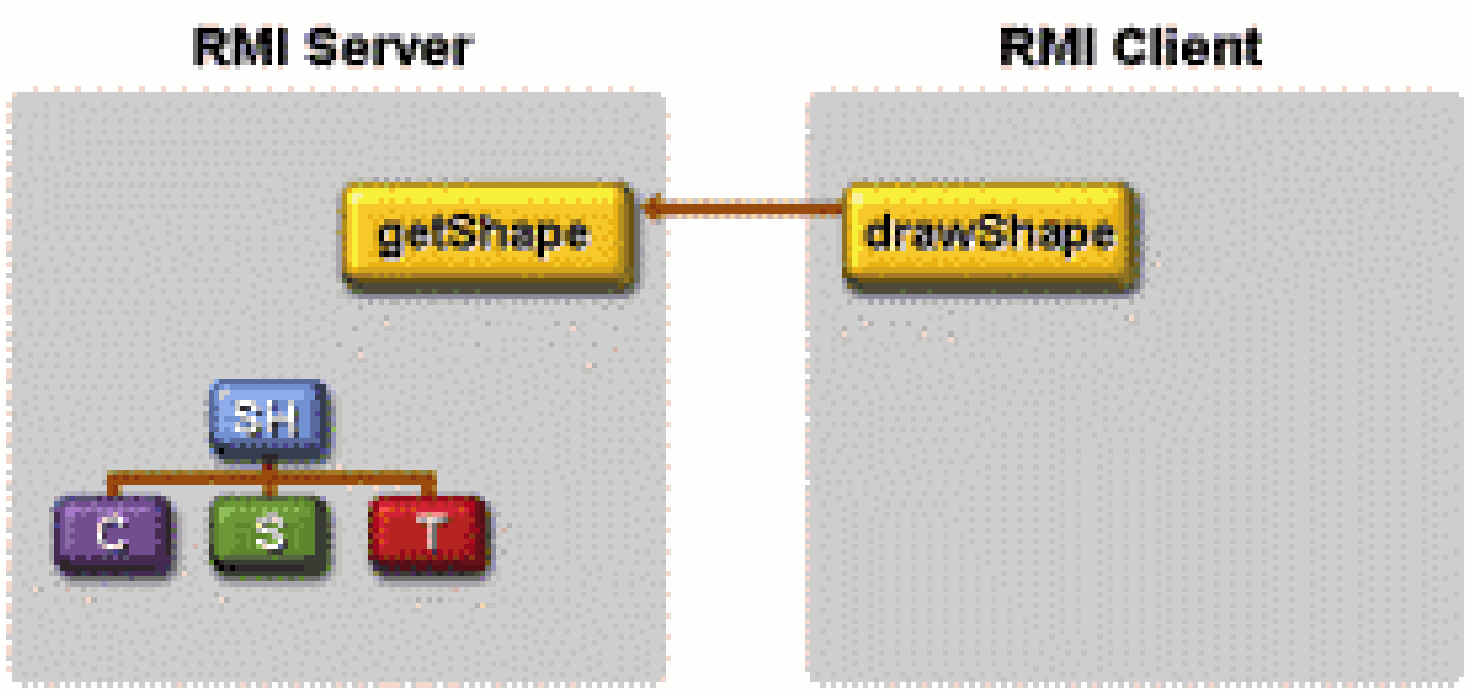
}

Verteilter Polymorphismus



- schauen wir uns das selbe Polymorphismus- Beispiel an, aber diesmal über Maschinengrenzen
  - der Client verlangt, dass ein Objekt gezeichnet wird `drawShape()`

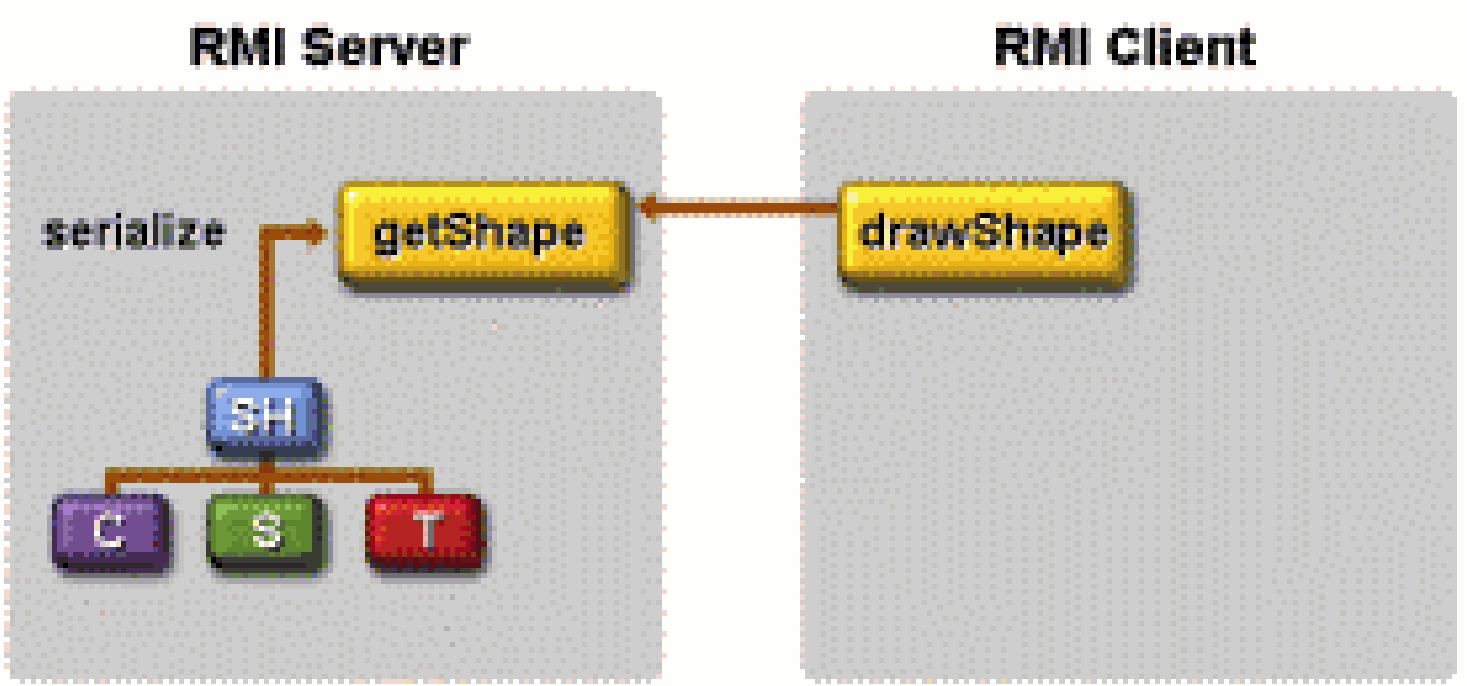
Verteilter Polymorphismus



Client:

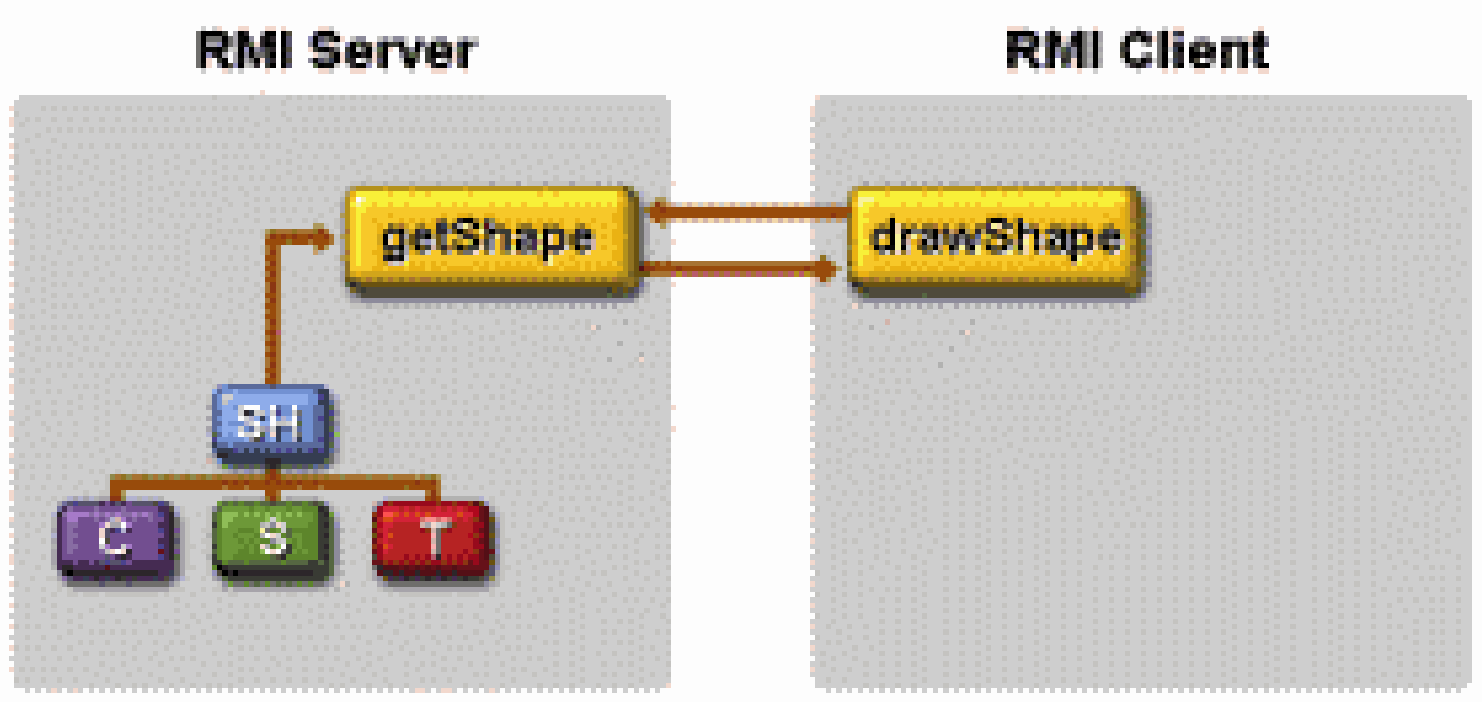
```
...
Shape shr = remote.getShape();
```

Verteilter Polymorphismus



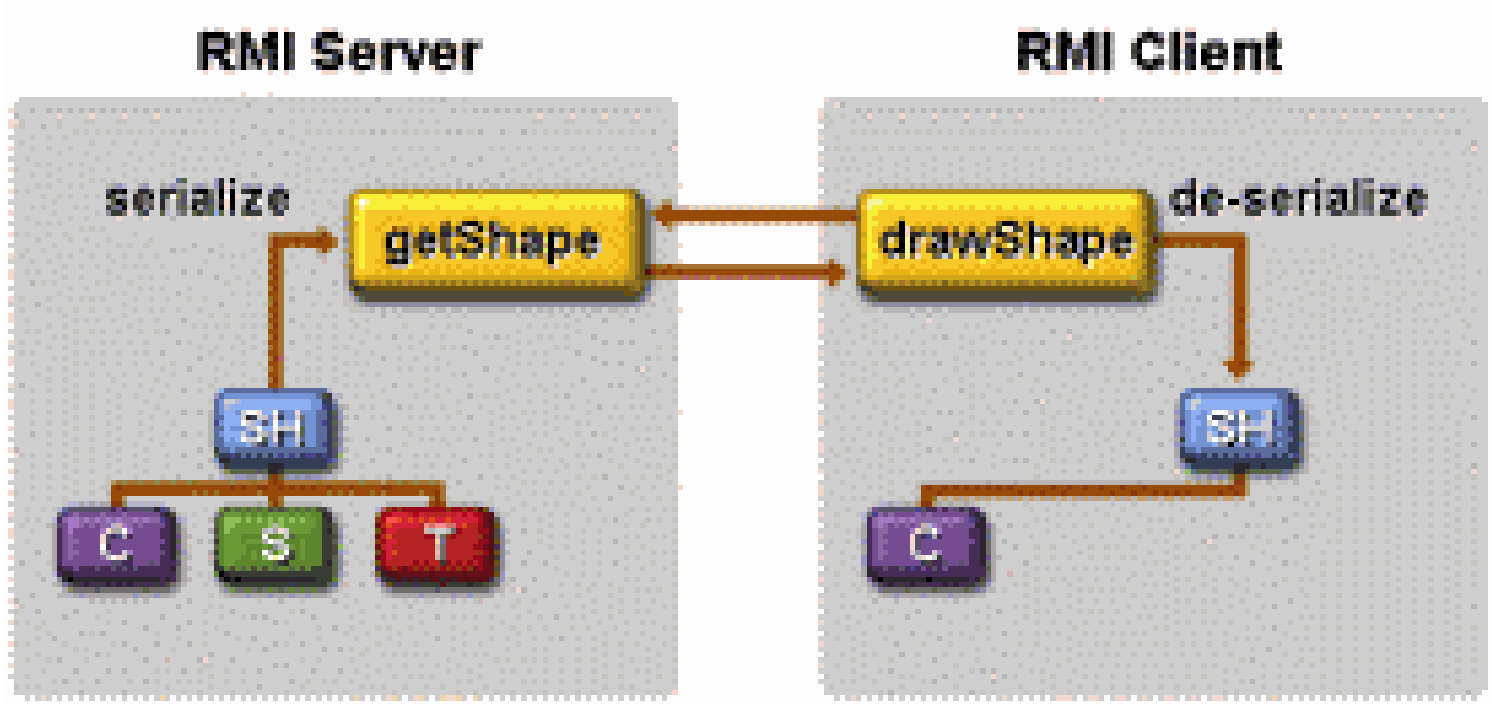
- Das Shape-Objekt wird in einen Bytestrom serialsiert

Verteilter Polymorphismus



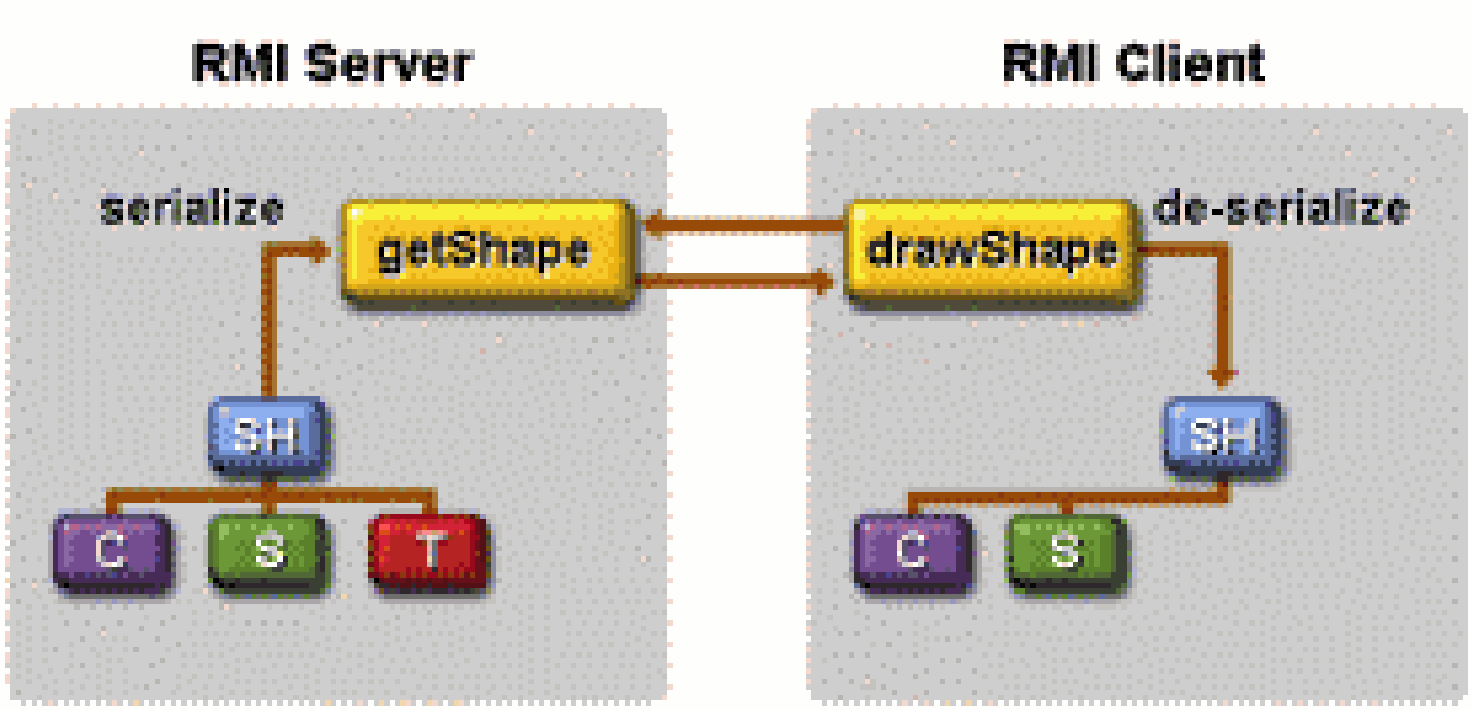
- Das Objekt wird über das Netzwerk transportiert (als Rückgabewert *by value*, also als Objekt serialisiert)

Verteilter Polymorphismus



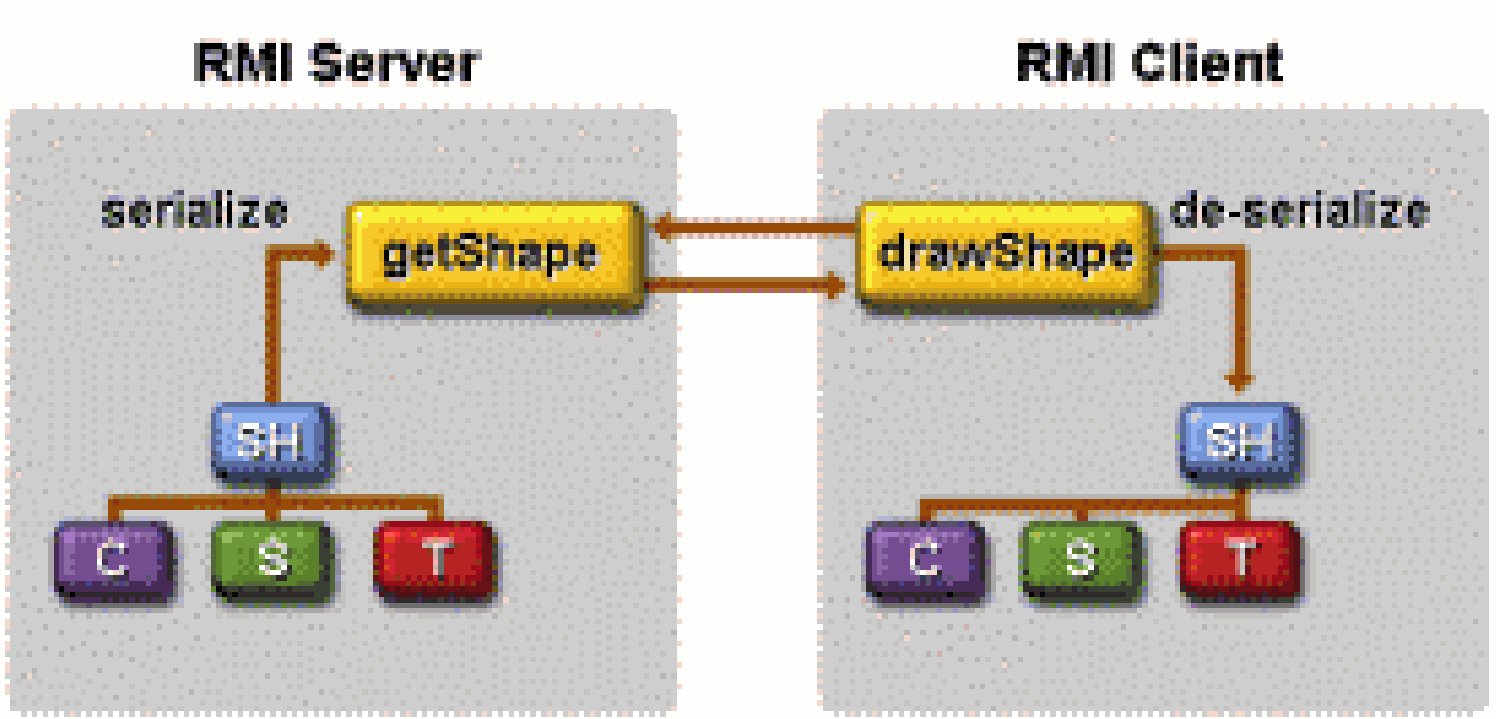
- Das Objekt wird auf der Client Seite deserialisiert;
  - Shape Untertyp (in diesem Fall „C“ [Circle]) wird aufgebaut und angezeigt

Verteilter Polymorphismus



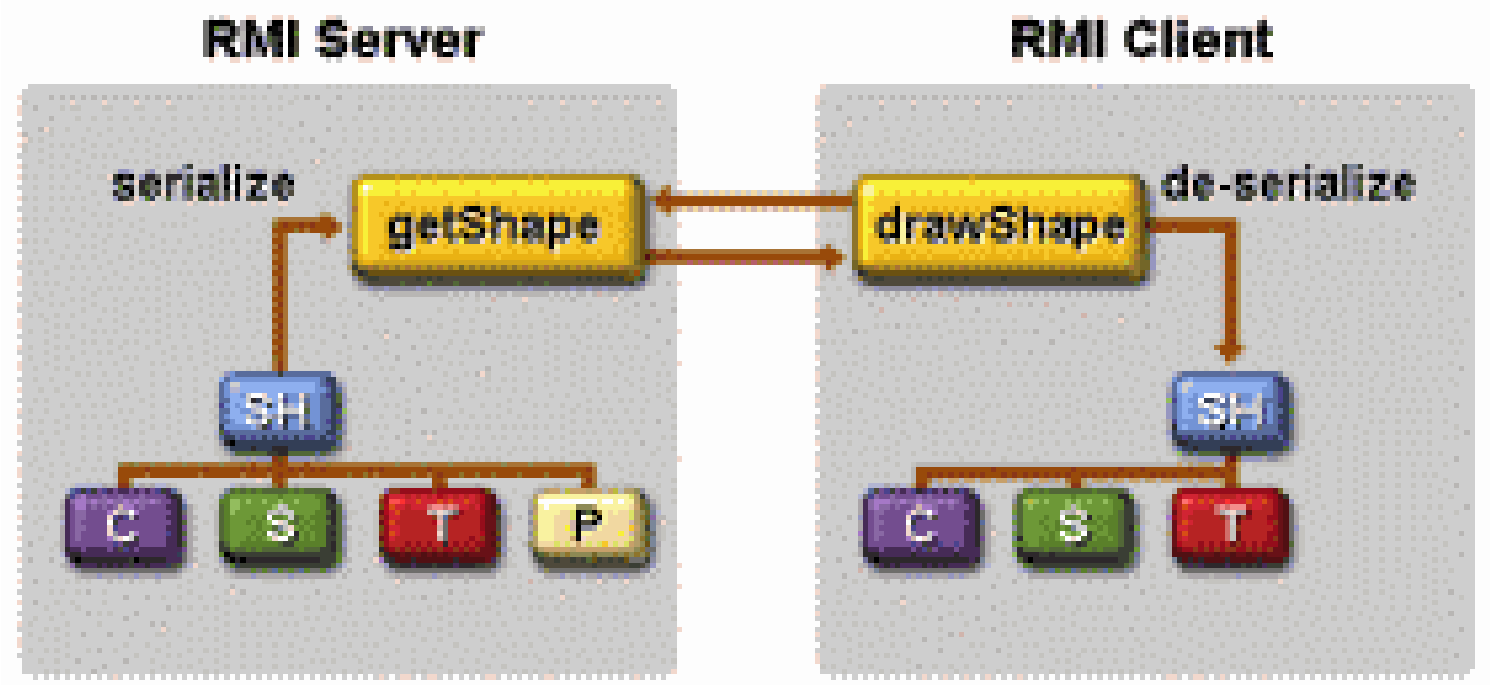
- Die Shape Subtypen werden verarbeitet
  - als nächstes Konstrukt wird das Quadrat („S“ wie Square) aufgebaut

Verteilter Polymorphismus



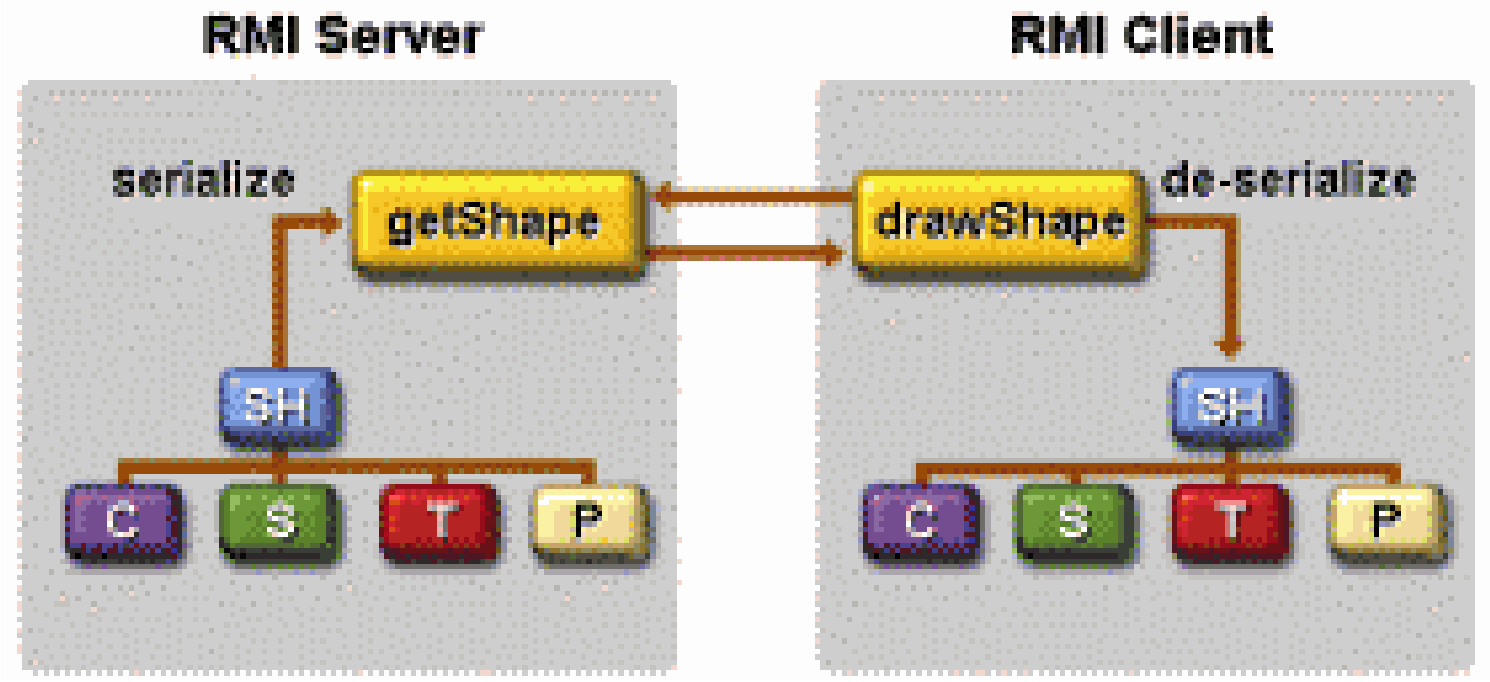
- Die Shape Subtypen werden verarbeitet
  - als nächstes Konstrukt wird das Dreieck („T“ wie „Triangle“) aufgebaut

Verteilter Polymorphismus



- der Shape Typ wird erweitert
  - auf dem Server wird die Klasse „Shape“ erweitert durch die Unterklasse „P“ wie „Polygon“

Verteilter Polymorphismus

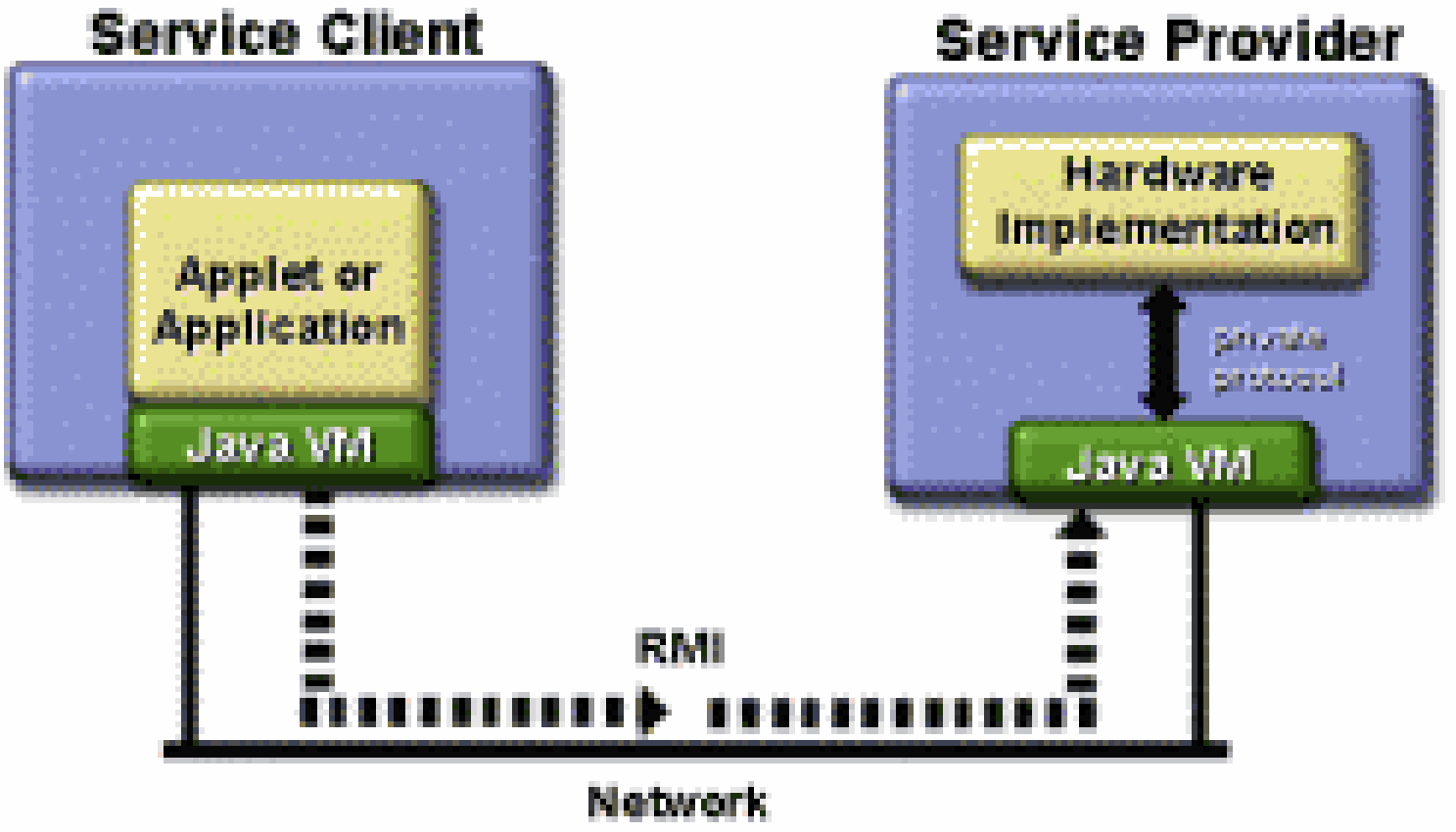


- Der *neue* Subtyp (Unterklasse „P“) wird *polymorph* an den Client geliefert
  - serialisiert- übermittelt- deserialisiert

Das Magische an der Jini Technologie

- Föderation von Java virtuellen Maschinen
- Protokolle auf der Basis von RMI
- verteilte Rechnerinfrastrukturen
- spontane Vernetzung
- von Grund auf einfaches Design
- Programmierschnittstellen

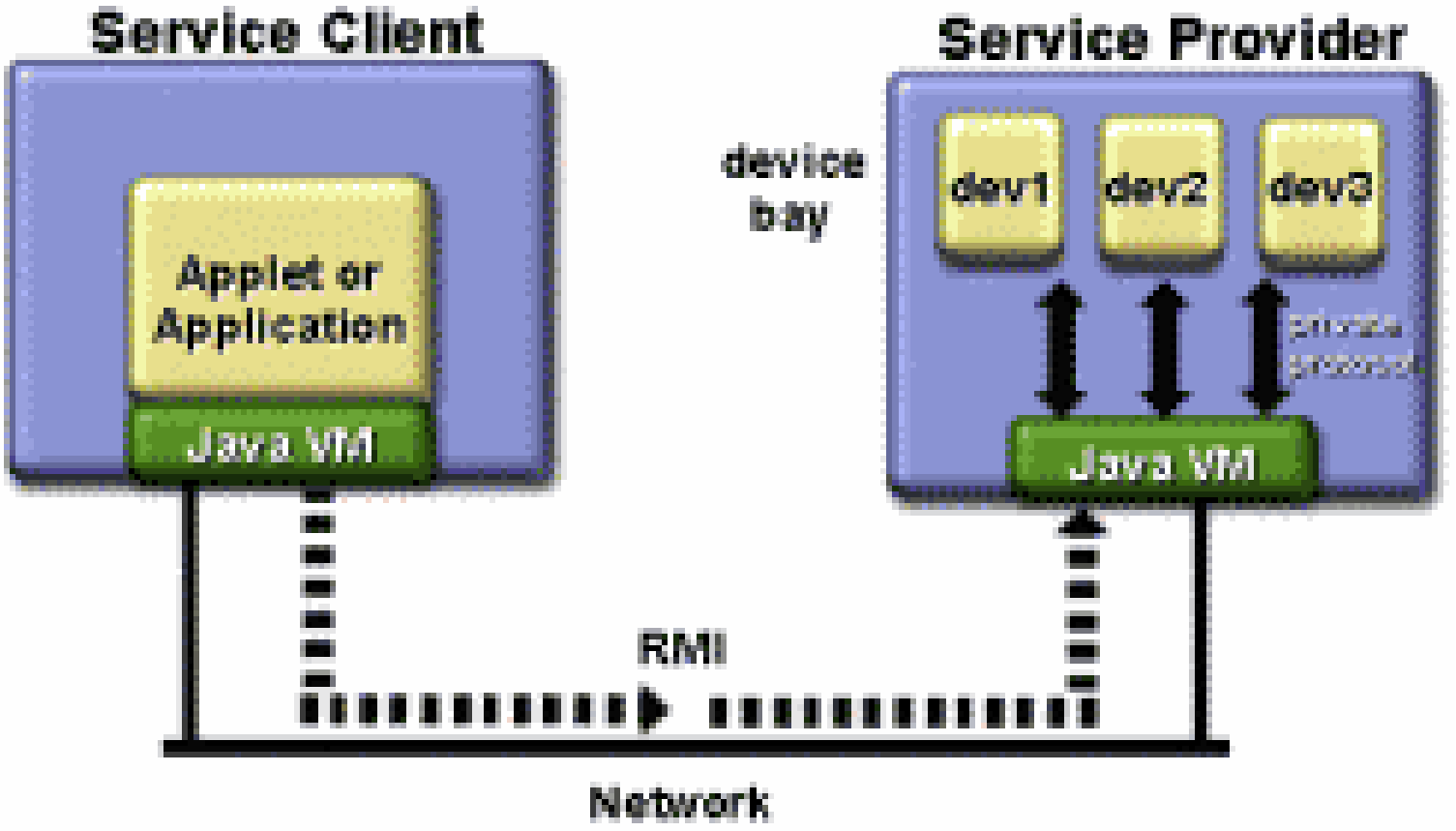
Das Jini Service *Modell 1*



Private

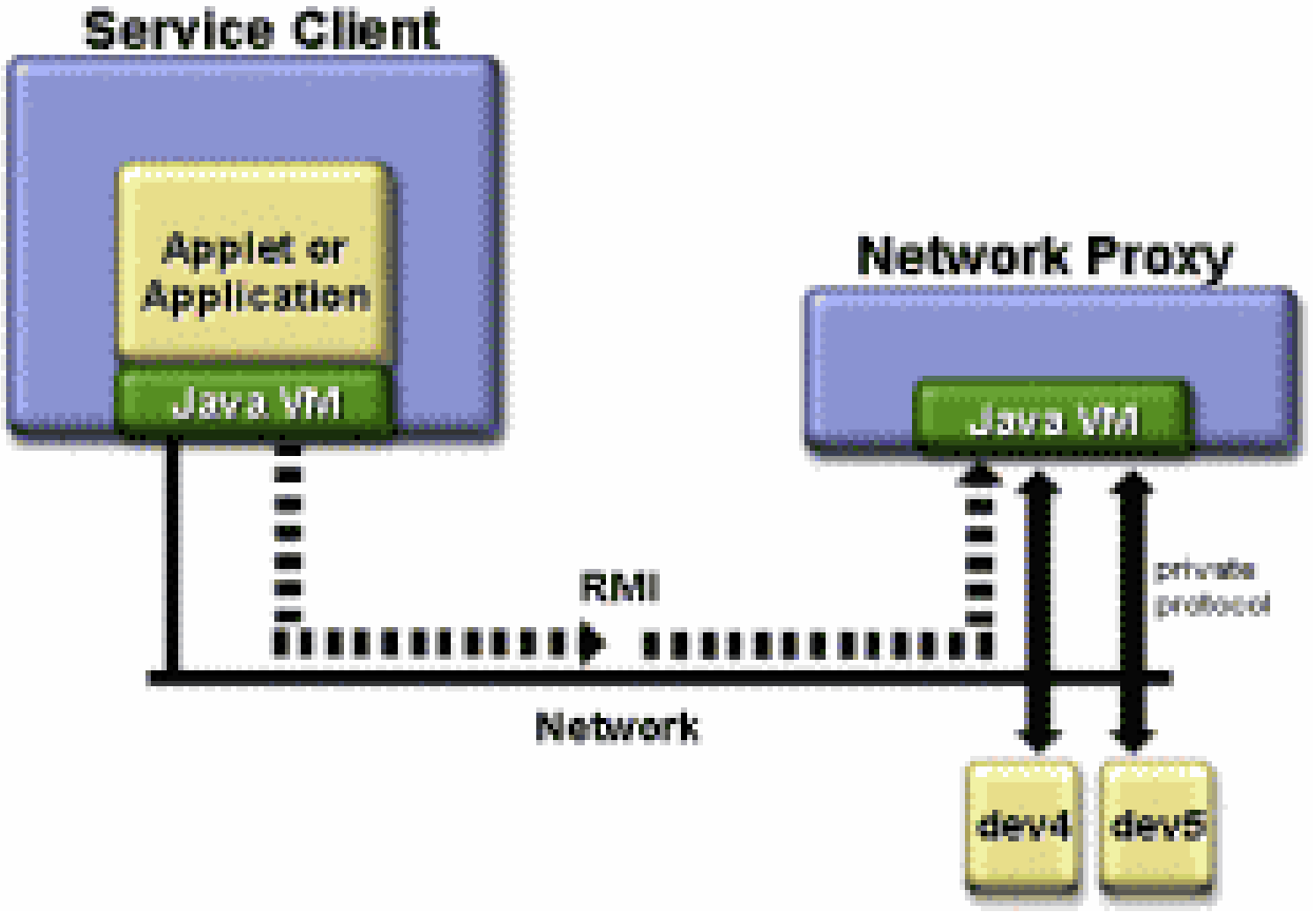
Protokolle

Das Jini Service *Modell 2*



Anbindung kleinster Geräte über einen Service Provider

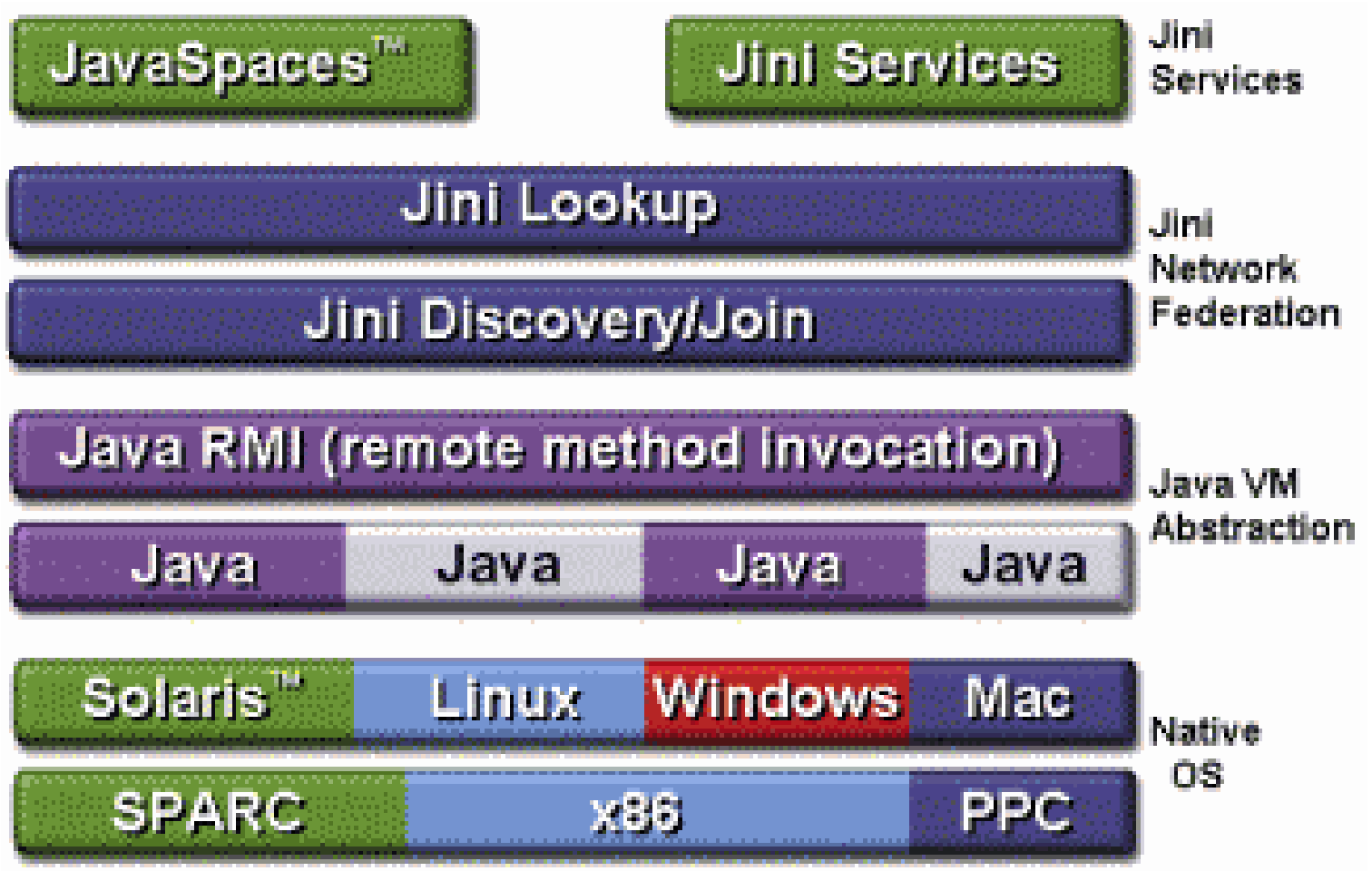
Das Jini Service *Modell 3*



Jini Architektur

- Java *RMI* stellt die Verbindung her
- Jini stellt die *Infrastruktur* zur Verfügung, um sich an Netzwerken zu beteiligen
  - Discovery / Join (1)
  - Lookup Service (2)
- APIs für Jini-fähige *Technologie*
  - Leasing (3)
  - Events (4)
  - Transaktionen (5)

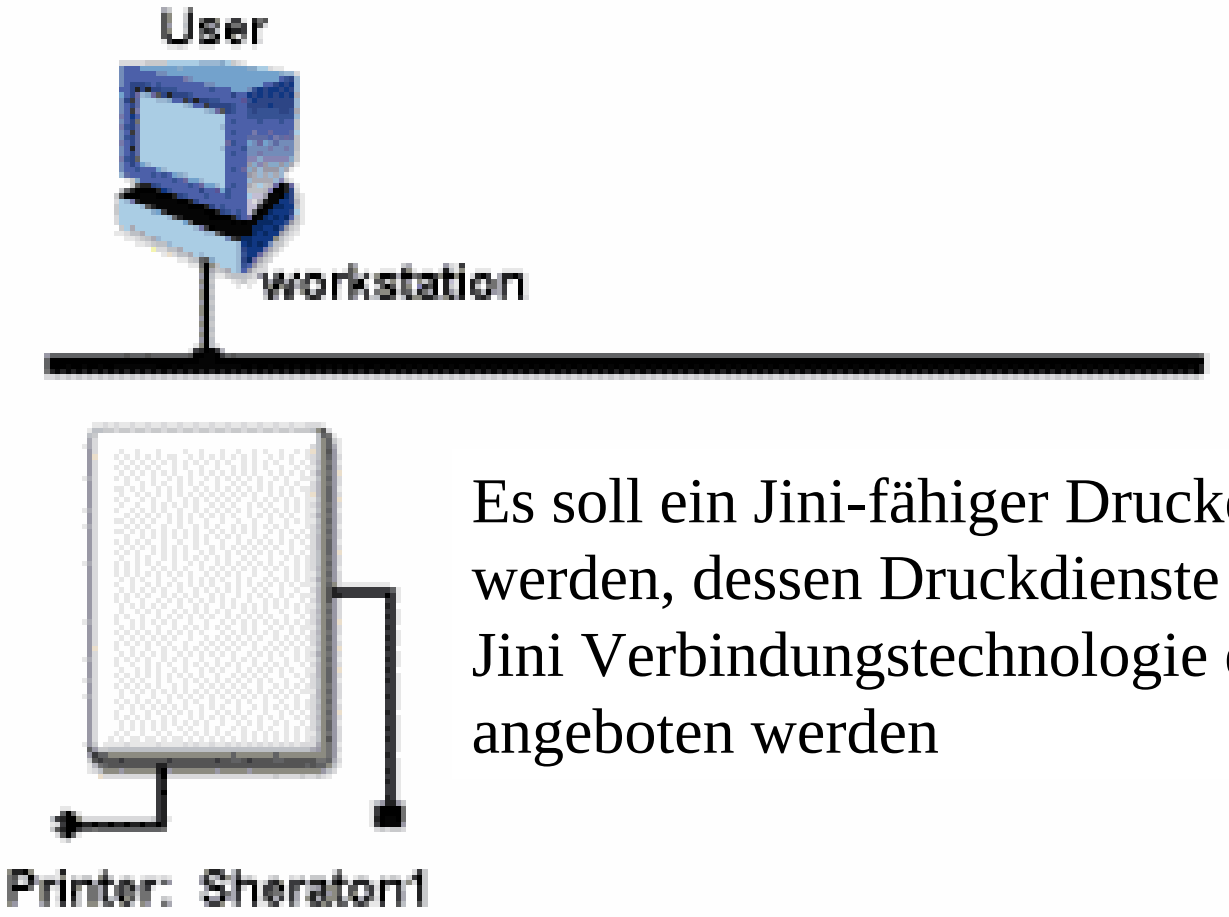
Jini Architektur



Jini Übersicht

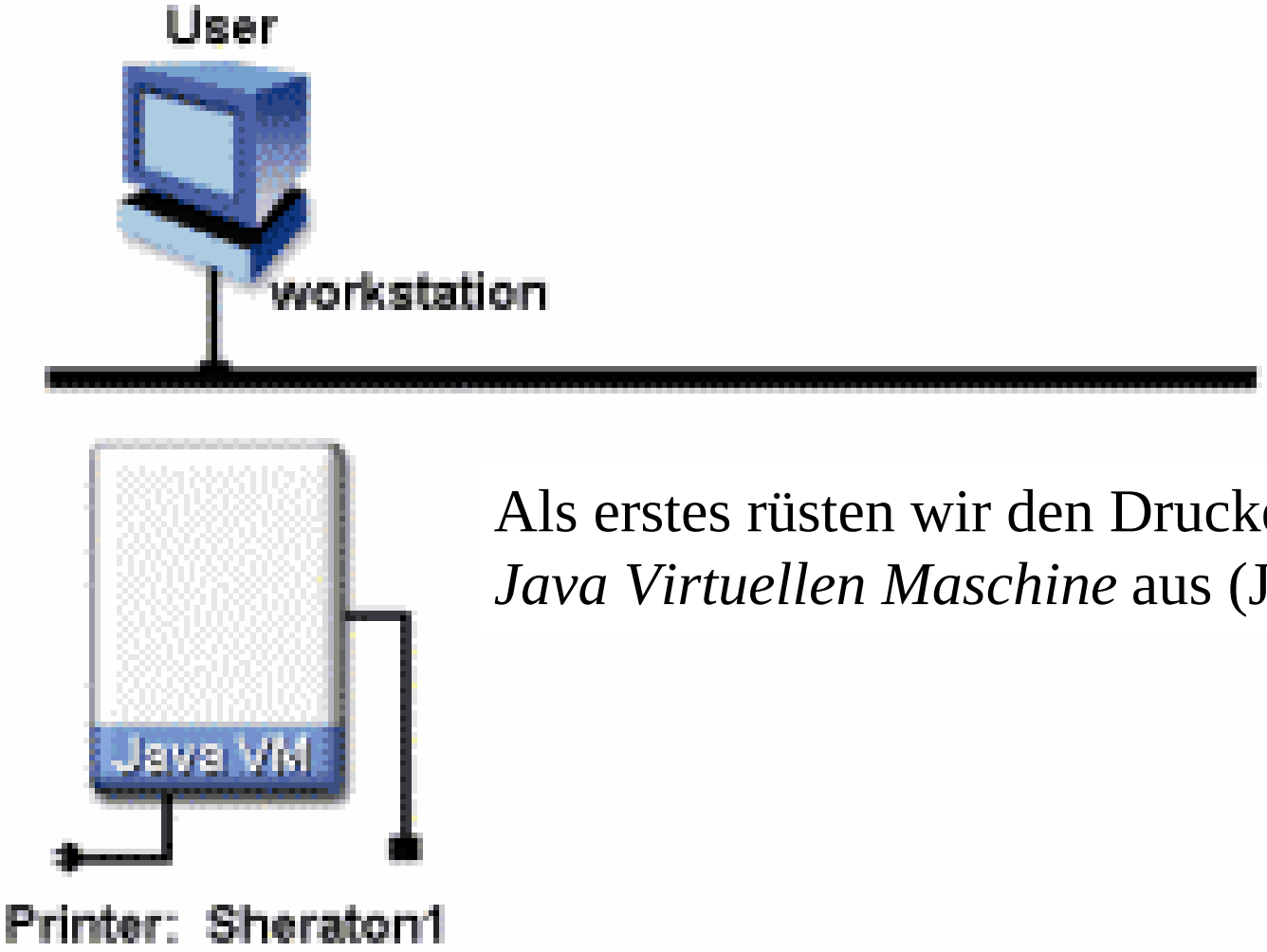
	Infrastruktur	*Programmier Modell*	*Services*
Jini	-Discovery -Lookup -erweiterte Security	-Lease -Ereignisse -Transaktionen	-JavaSpaces -TX Manager
Programmierung	-Java VM -RMI -Security	-Java Technologie-basierte APIs - Beans ...	-Enterprise JavaBeans -Java Naming & Directory Interfaces -JTB

Drucker in einem Jini Netzwerk



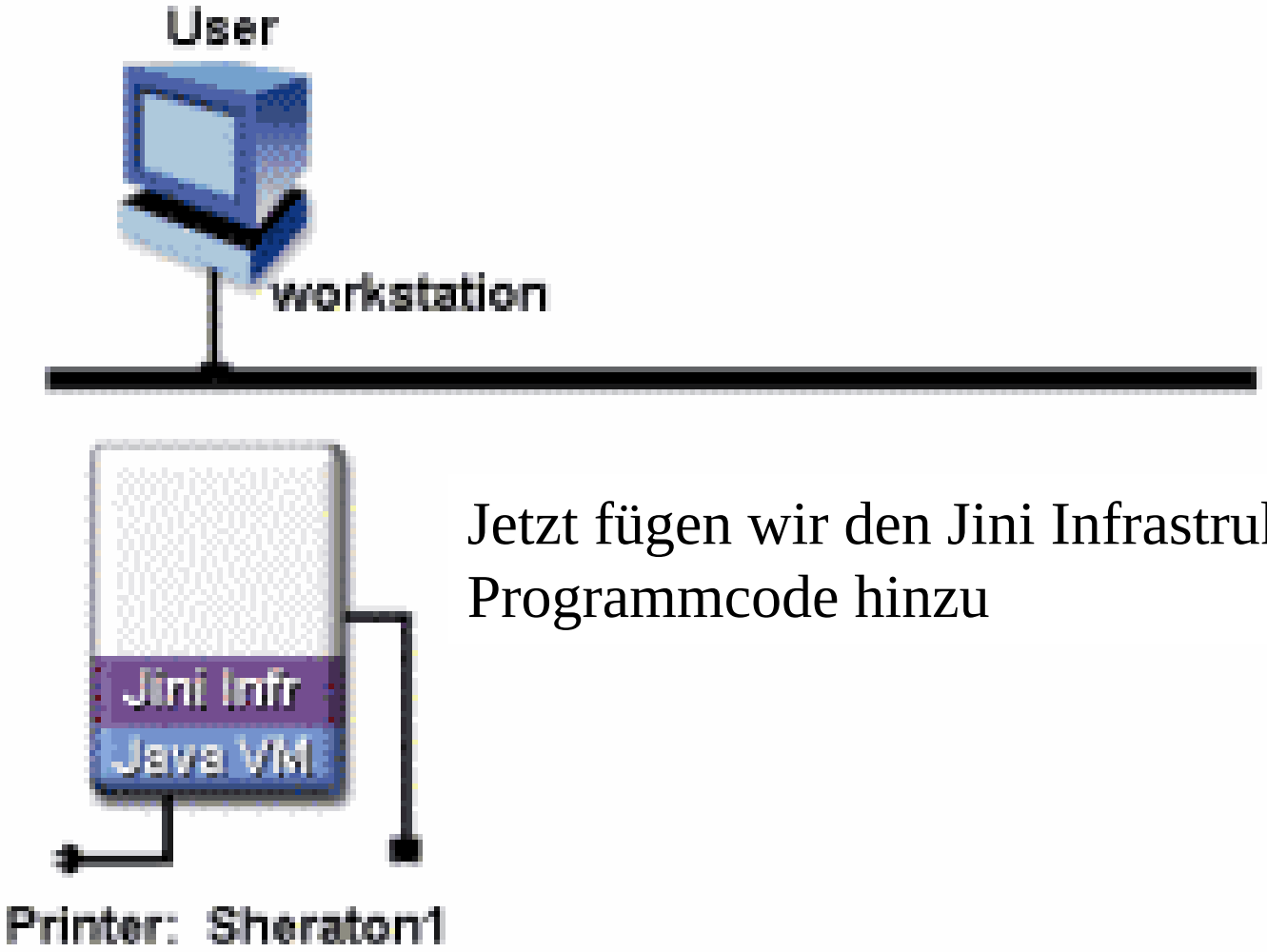
Es soll ein Jini-fähiger Drucker gebaut werden, dessen Druckdienste mit Hilfe von Jini Verbindungstechnologie ermöglicht und angeboten werden

Drucker in einem Jini Netzwerk



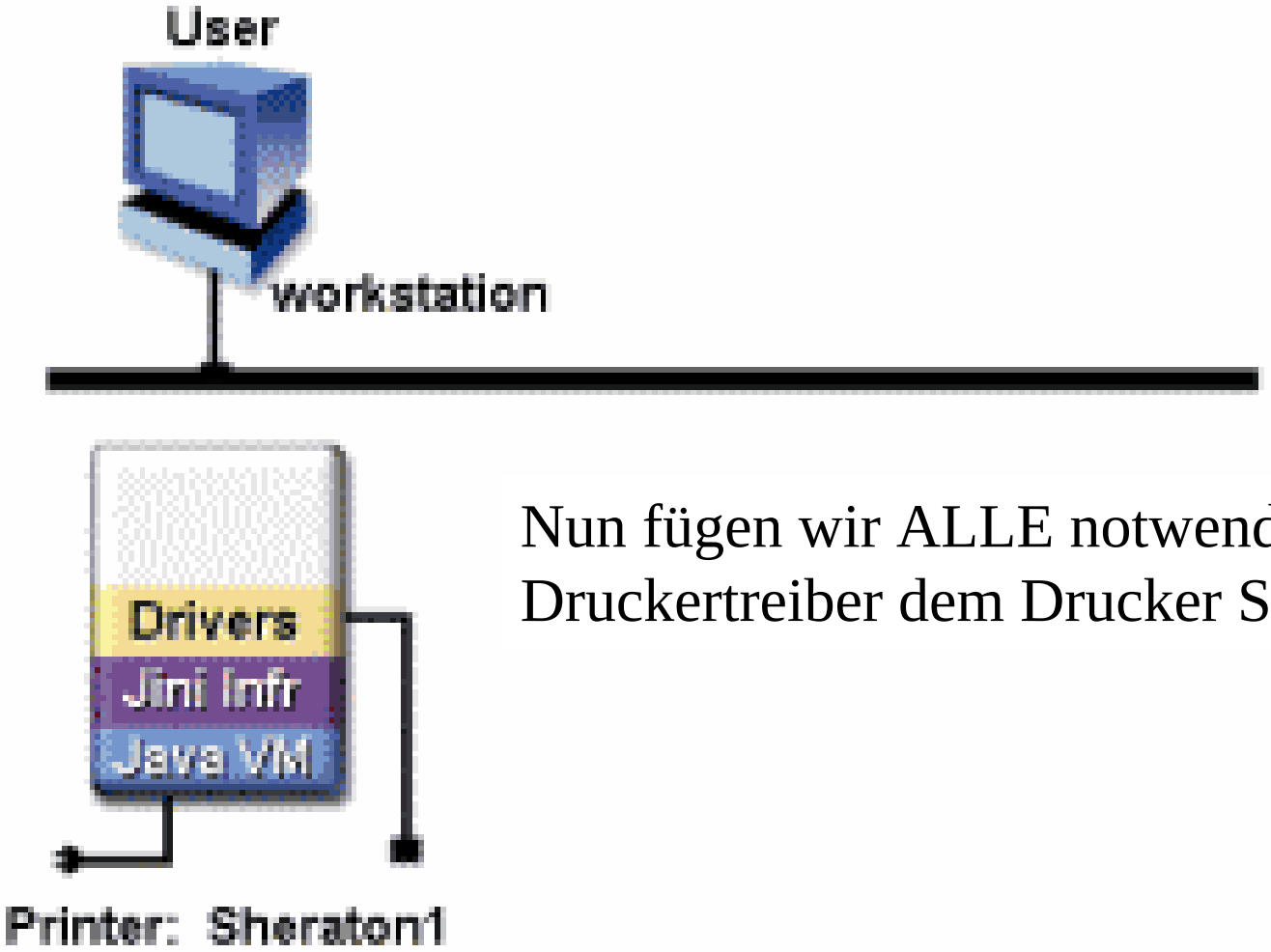
Als erstes rüsten wir den Drucker mit einer *Java Virtuellen Maschine* aus (Jini Modell 1)

Drucker in einem Jini Netzwerk



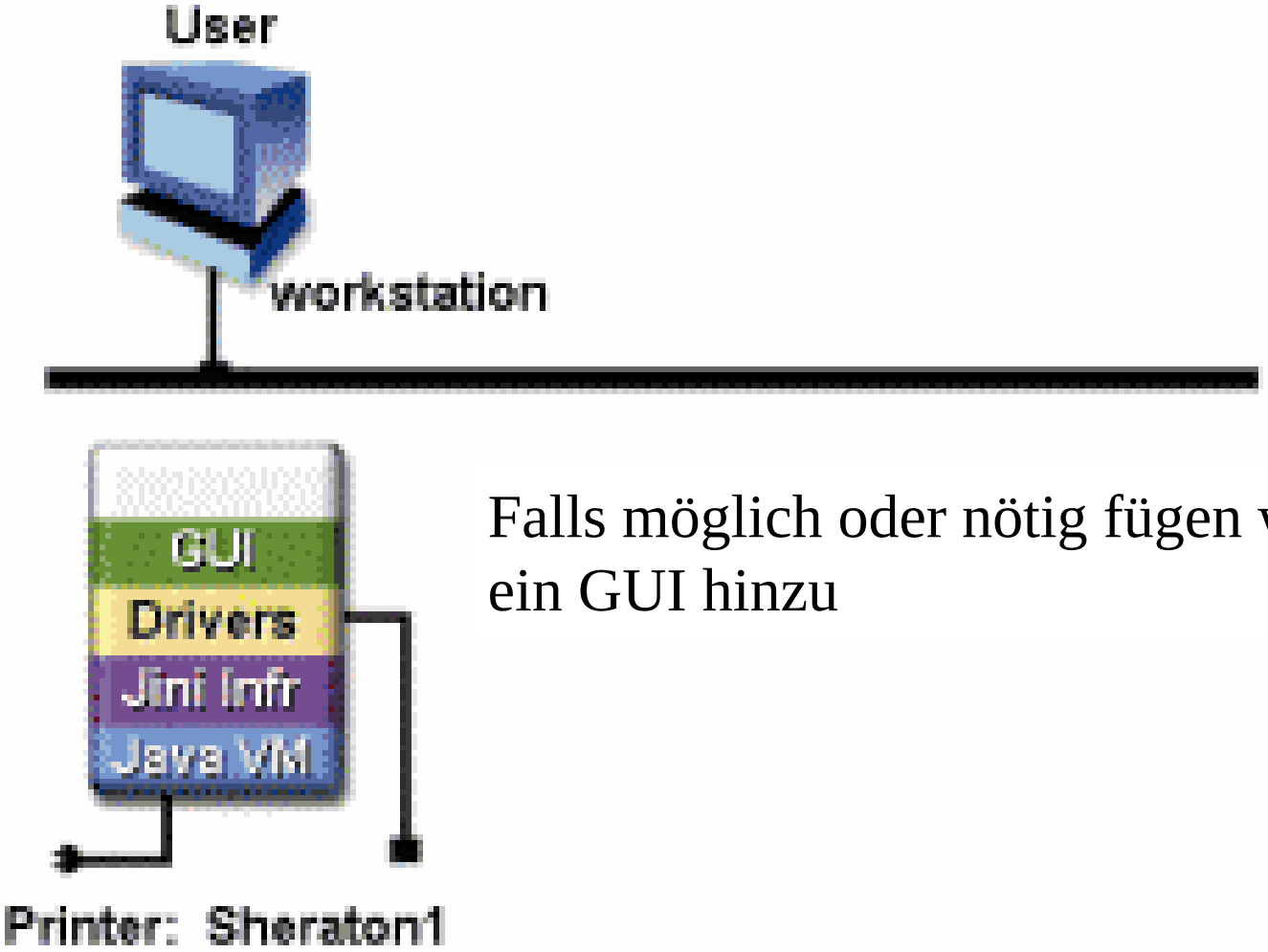
Jetzt fügen wir den Jini Infrastruktur Programmcode hinzu

Drucker in einem Jini Netzwerk



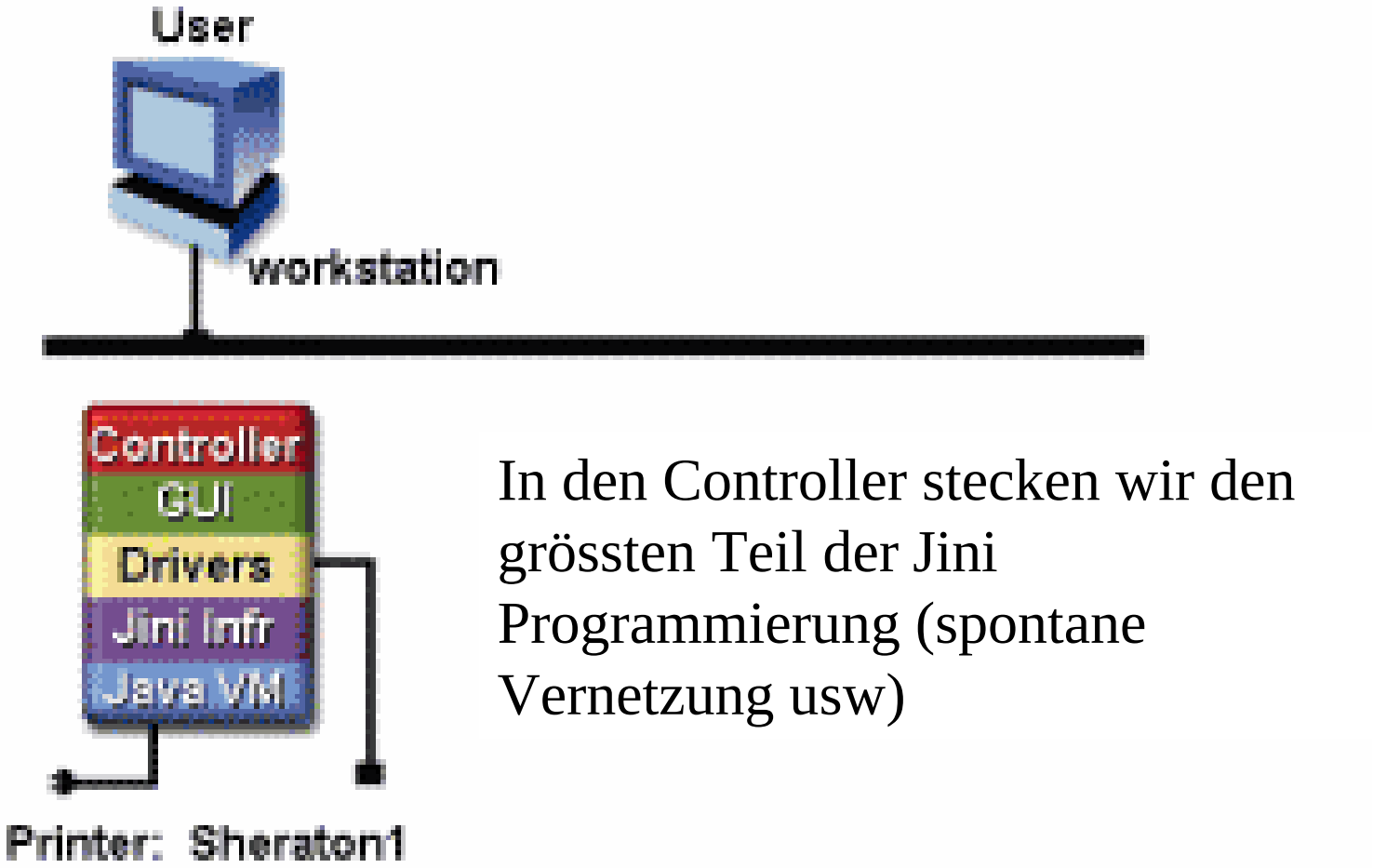
Nun fügen wir ALLE notwendigen Druckertreiber dem Drucker Service hinzu

Drucker in einem Jini Netzwerk



Falls möglich oder nötig fügen wir auch noch ein GUI hinzu

Drucker in einem Jini Netzwerk

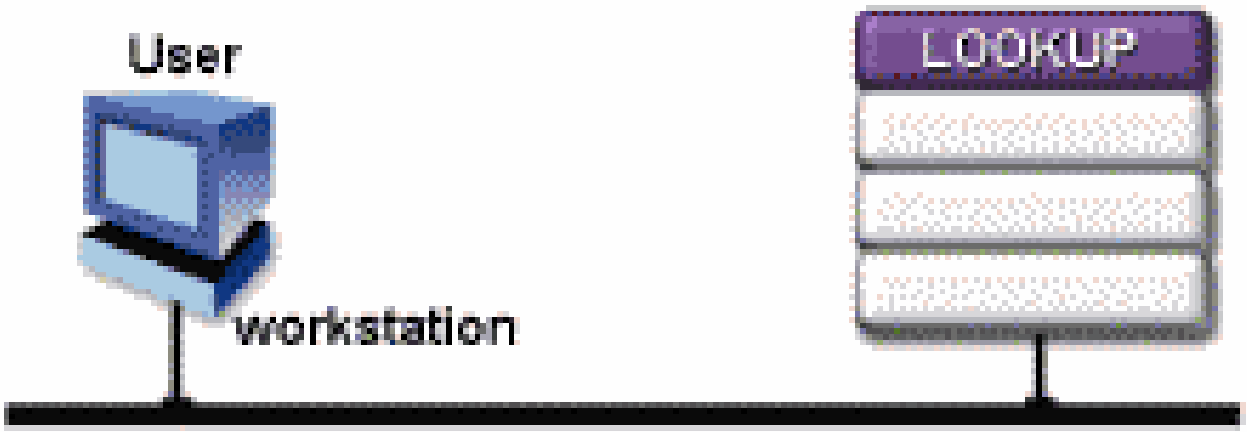


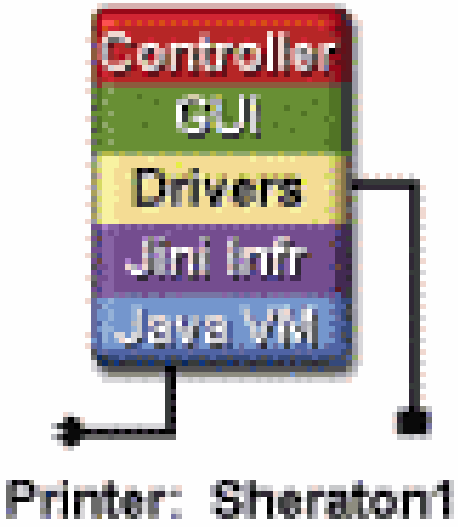
In den Controller stecken wir den grössten Teil der Jini Programmierung (spontane Vernetzung usw)

Discovery / Join

- Auffinden und beitreten zu einer Gruppe von Diensten
- Senden von Multicast Broadcasting
- Ankündigen der Dienste / Fähigkeiten
- *benötigte Software und Treiber zur Verfügung stellen*
- Verbindung zum Lookup Server herstellen

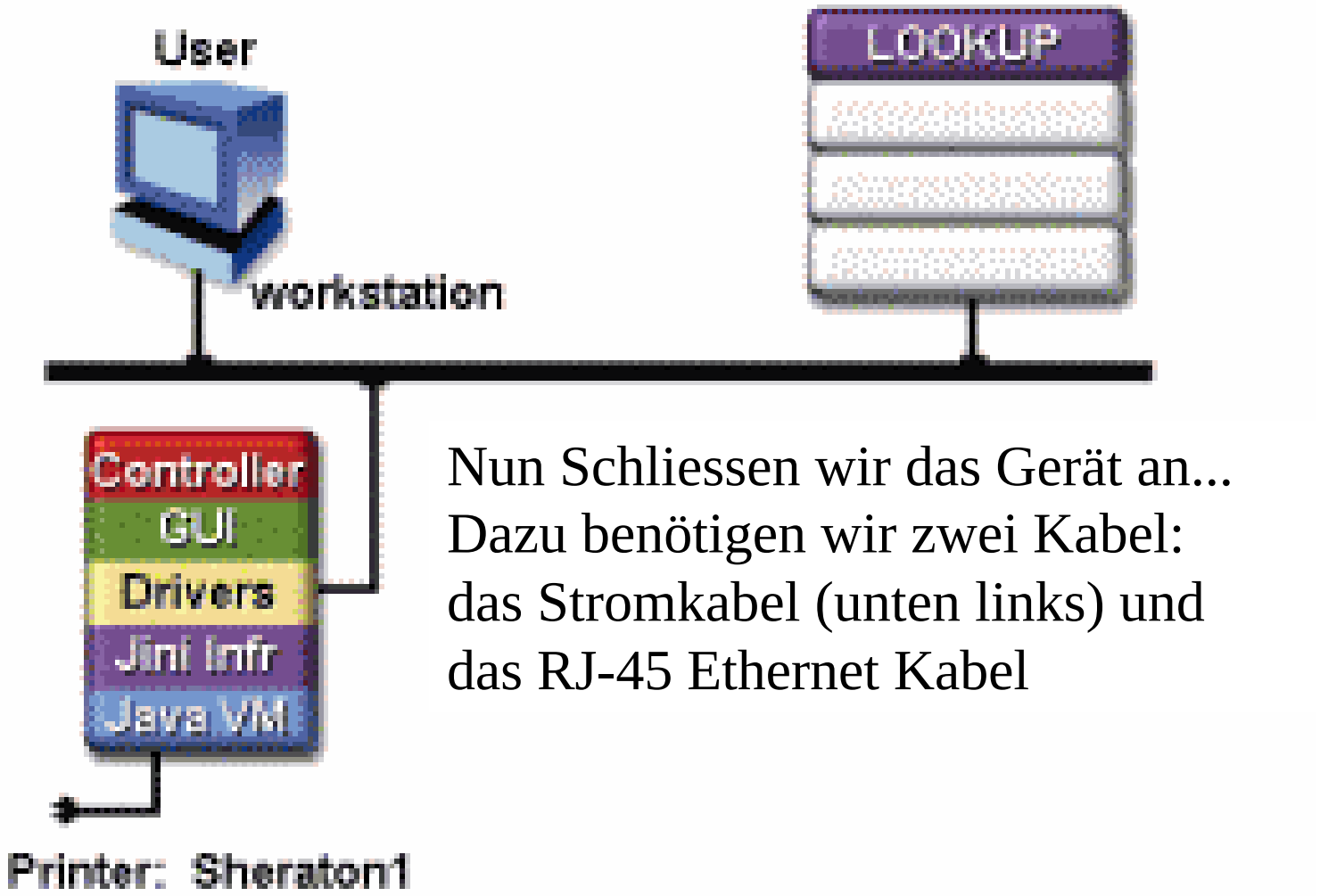
Discovery / Join





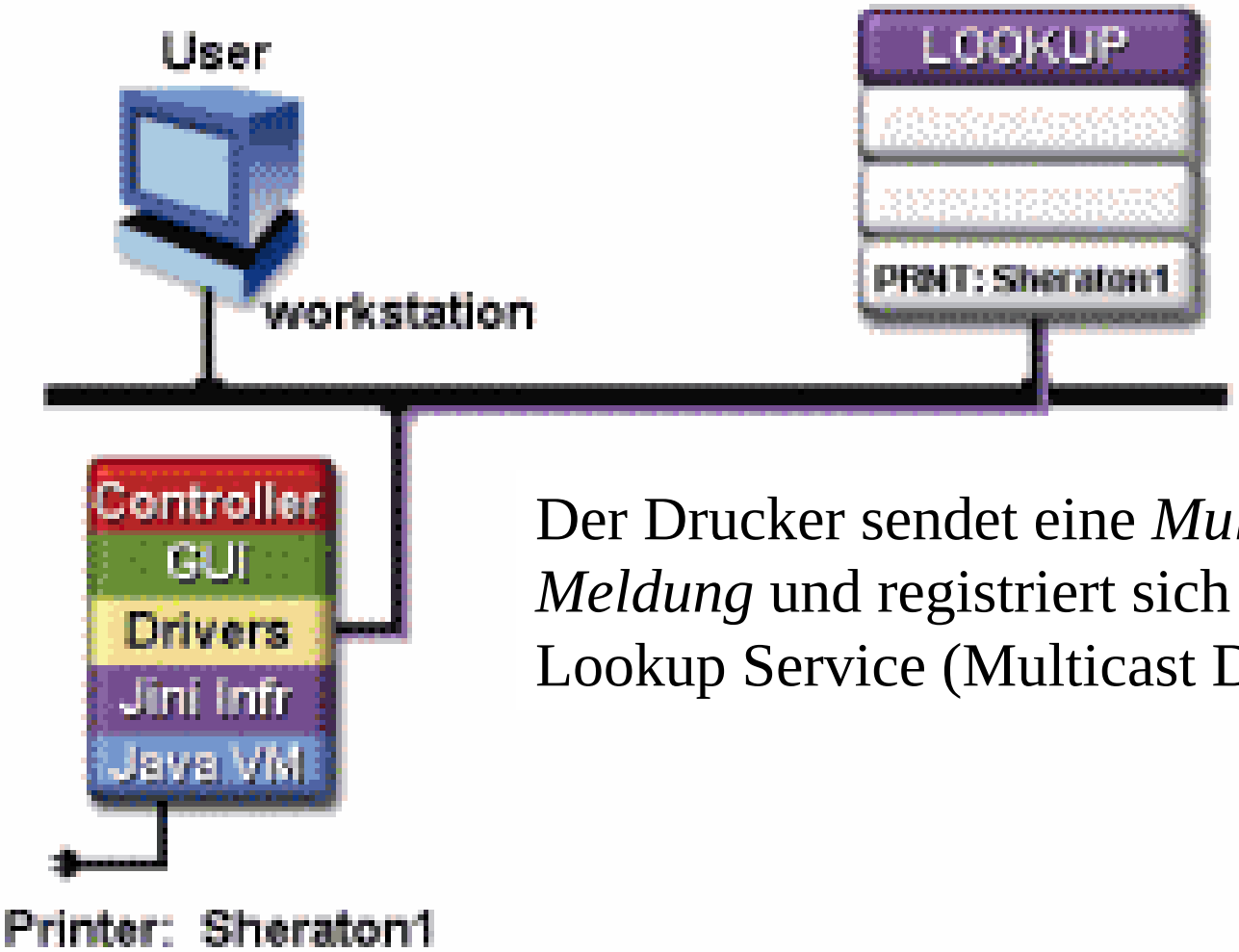
Ein Jini Lookup Service muss *irgendwo* im Netzwerk laufen

Discovery / Join



Nun Schliessen wir das Gerät an...
Dazu benötigen wir zwei Kabel:
das Stromkabel (unten links) und
das RJ-45 Ethernet Kabel

Discovery / Join

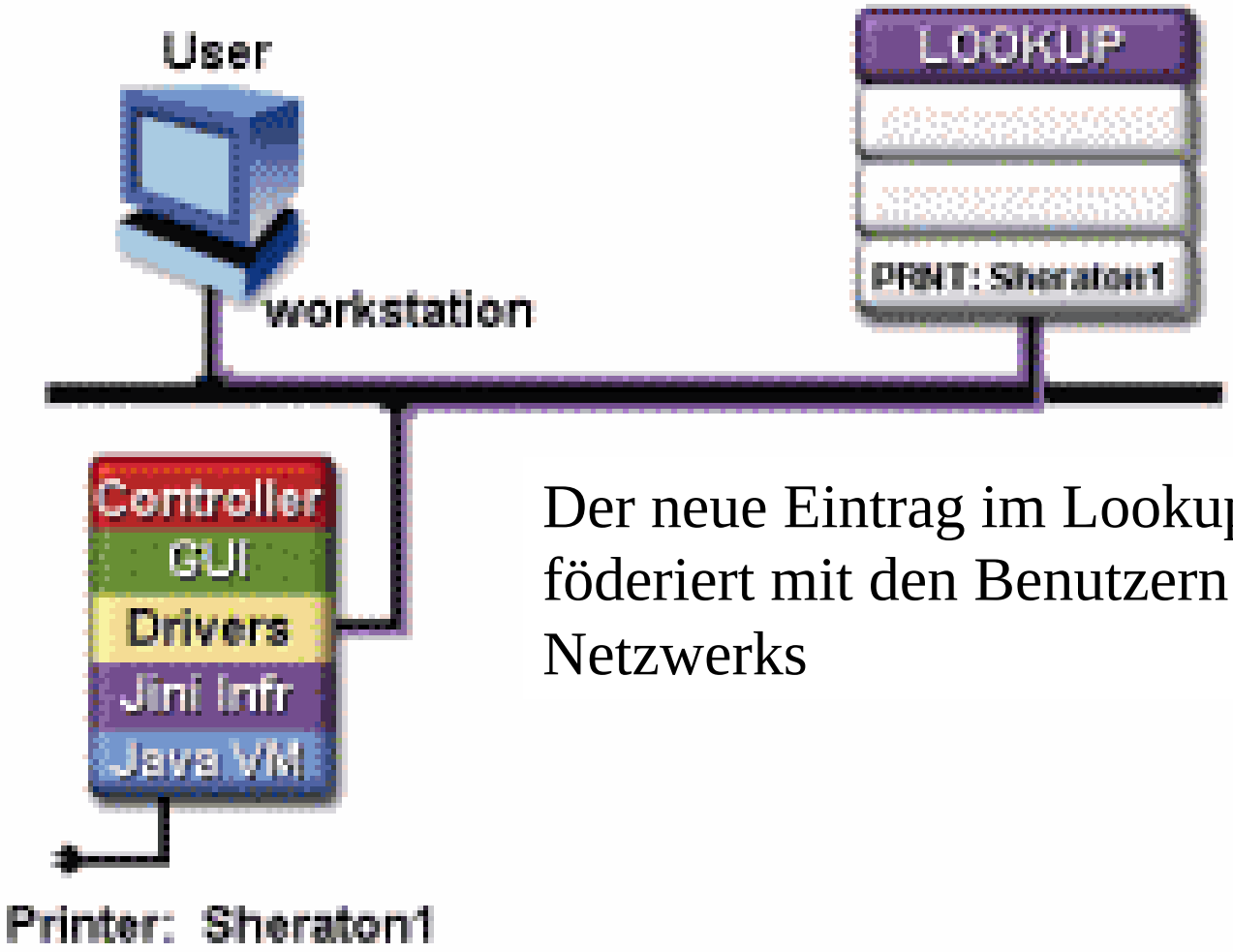


Der Drucker sendet eine *Multicast Meldung* und registriert sich beim Jini Lookup Service (Multicast Discovery)

Lookup Service

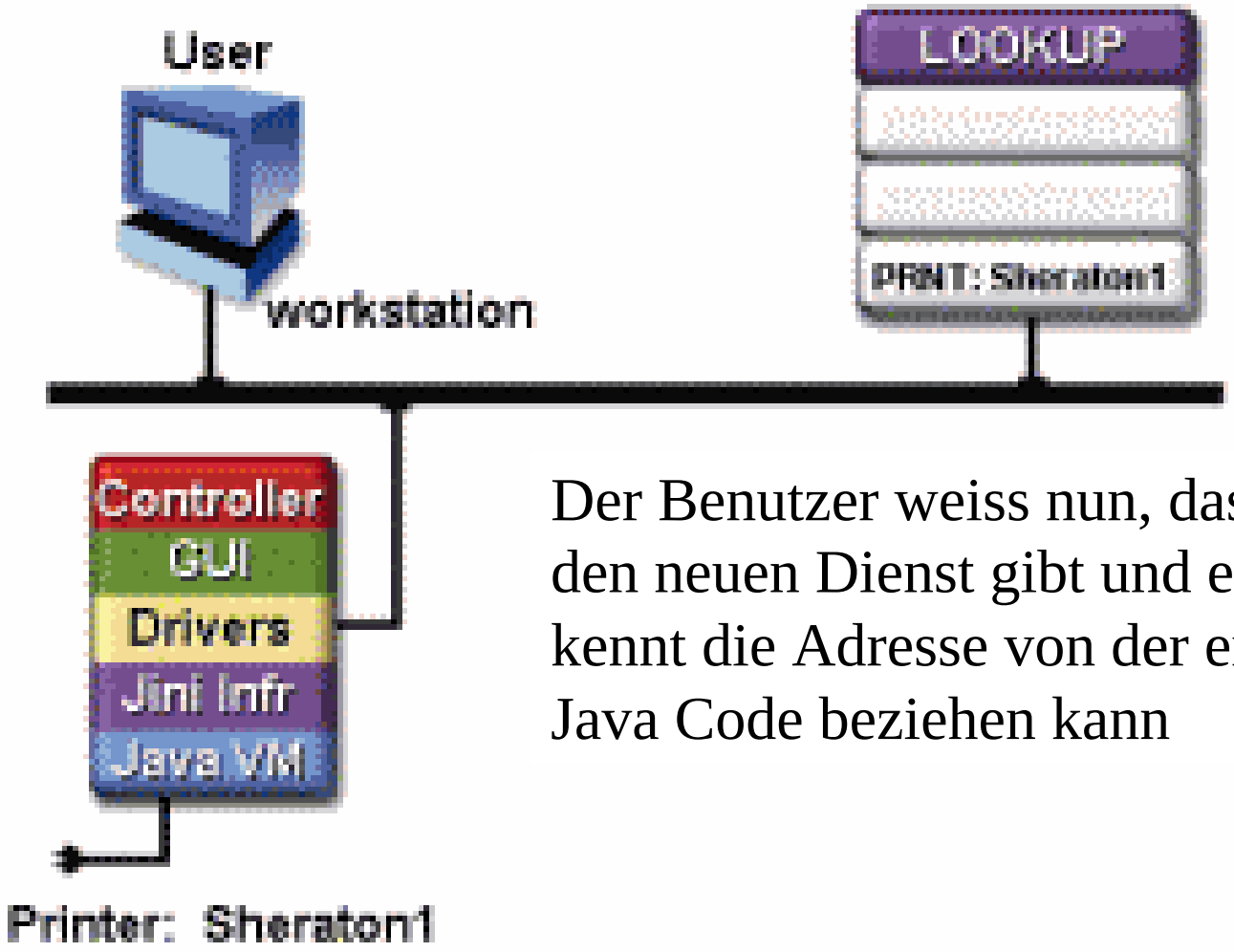
- Repository der verfügbaren Dienste
- speichert *Dienste* als erweiterbare Java Objekte
- der Benutzer kann Dienste nach Bedarf herunterladen
- der Lookup Service kann mit andern Lookup Services föderieren

Lookup Service



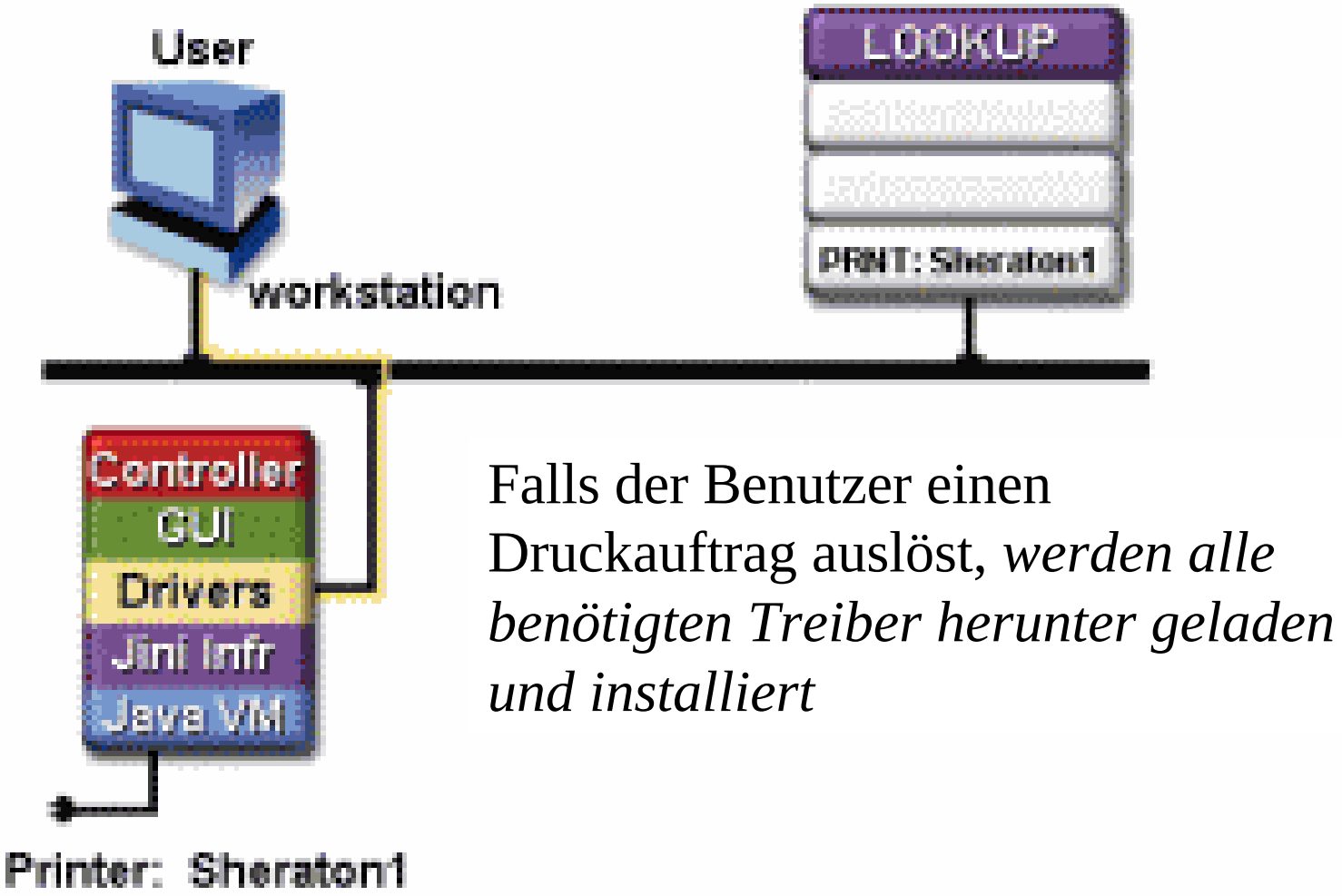
Der neue Eintrag im Lookup Service föderiert mit den Benutzern des Netzwerks

Bedarfsgesteuerte Dienste / Services on Demand



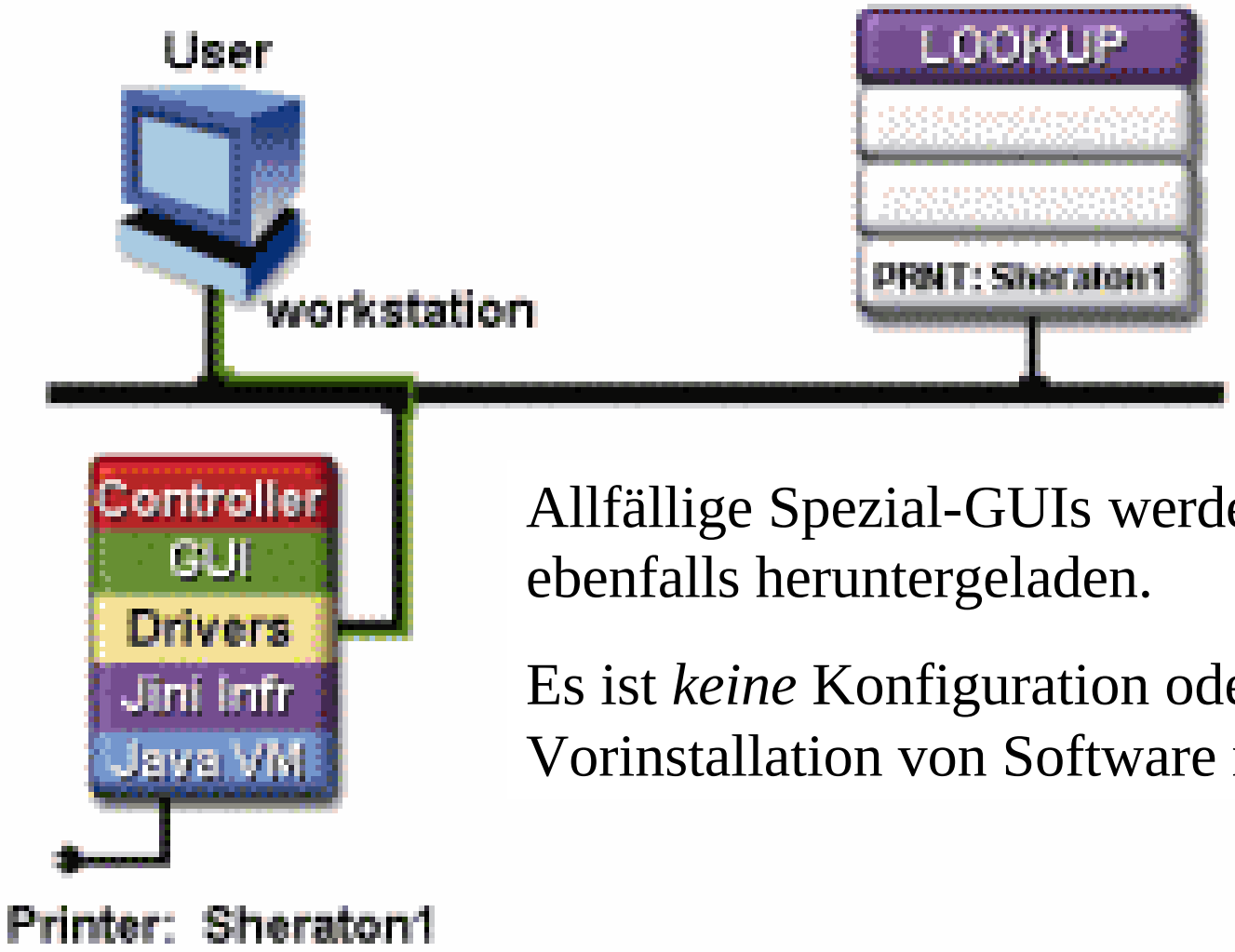
Der Benutzer weiss nun, dass es den neuen Dienst gibt und er kennt die Adresse von der er den Java Code beziehen kann

Services on Demand



Falls der Benutzer einen Druckauftrag auslöst, *werden alle benötigten Treiber herunter geladen und installiert*

Services on Demand



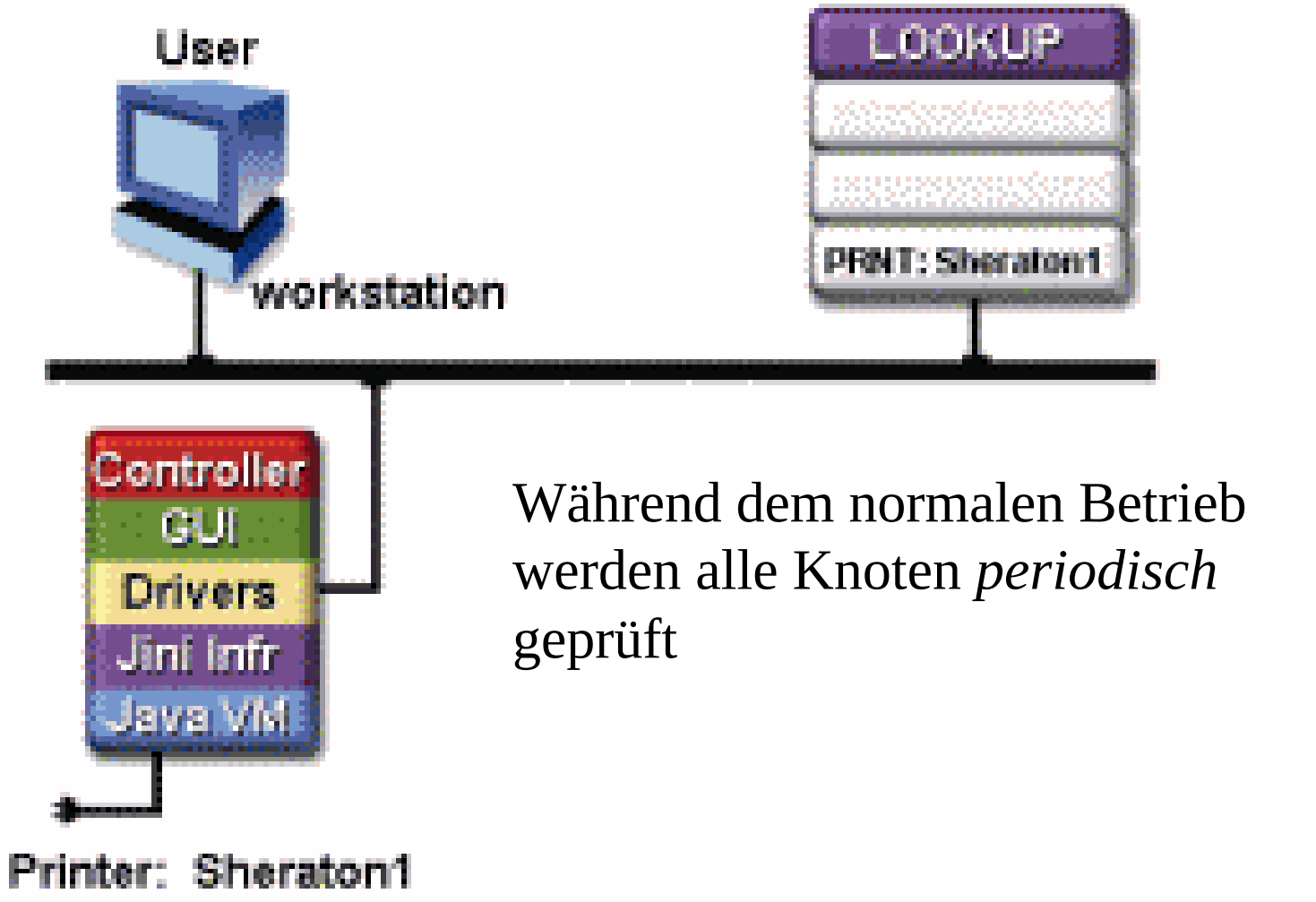
Allfällige Spezial-GUIs werden ebenfalls heruntergeladen.

Es ist *keine* Konfiguration oder Vorinstallation von Software nötig.

Distributed Leasing

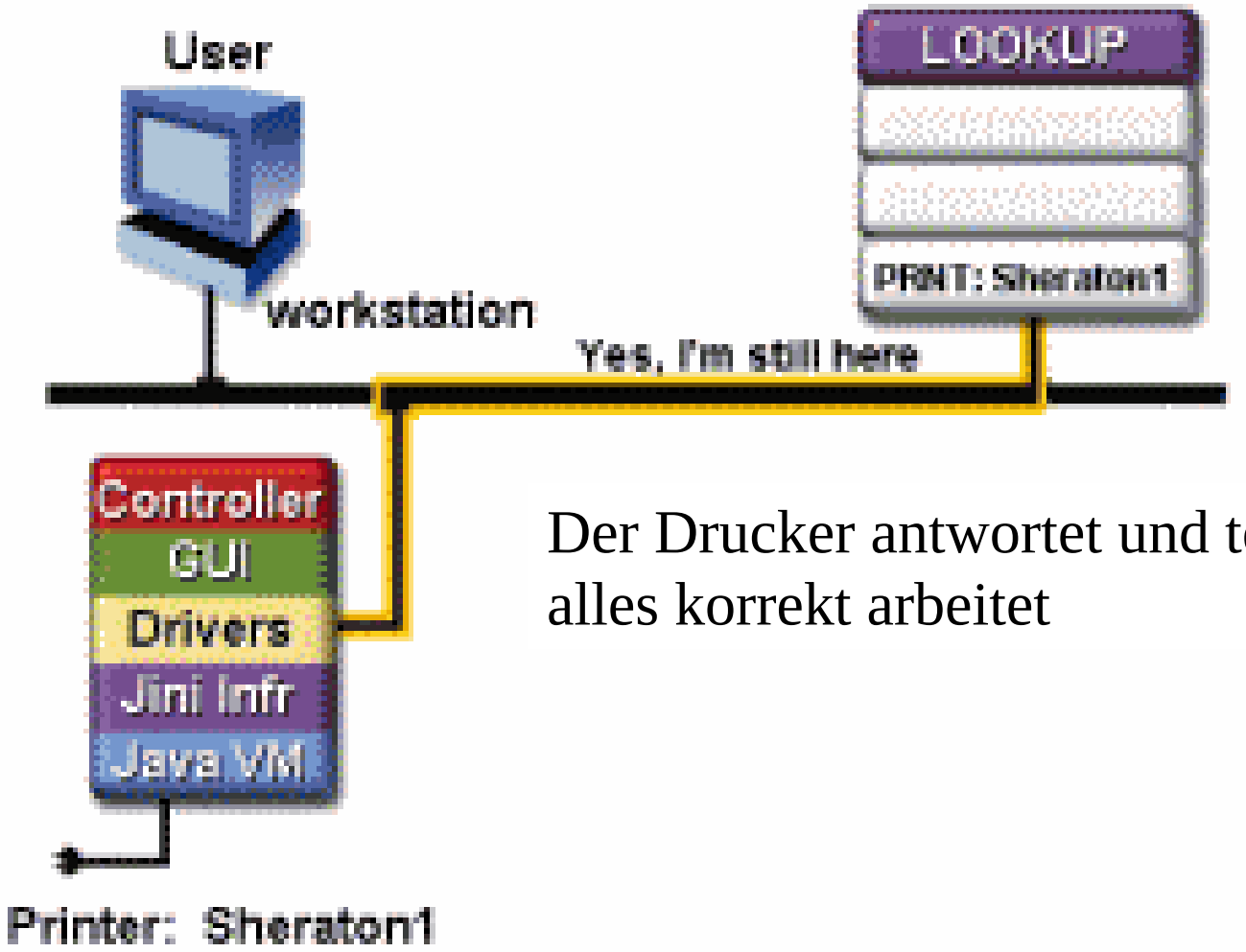
- Protokoll zum Verwalten der Ressourcen
- verwendet ein erneuerbares, zeitbasiertes Protokoll
- stellt Kontrakte her zwischen Objekten
- Ressourcen können shared oder auch nicht-shared sein

Distributed Leasing



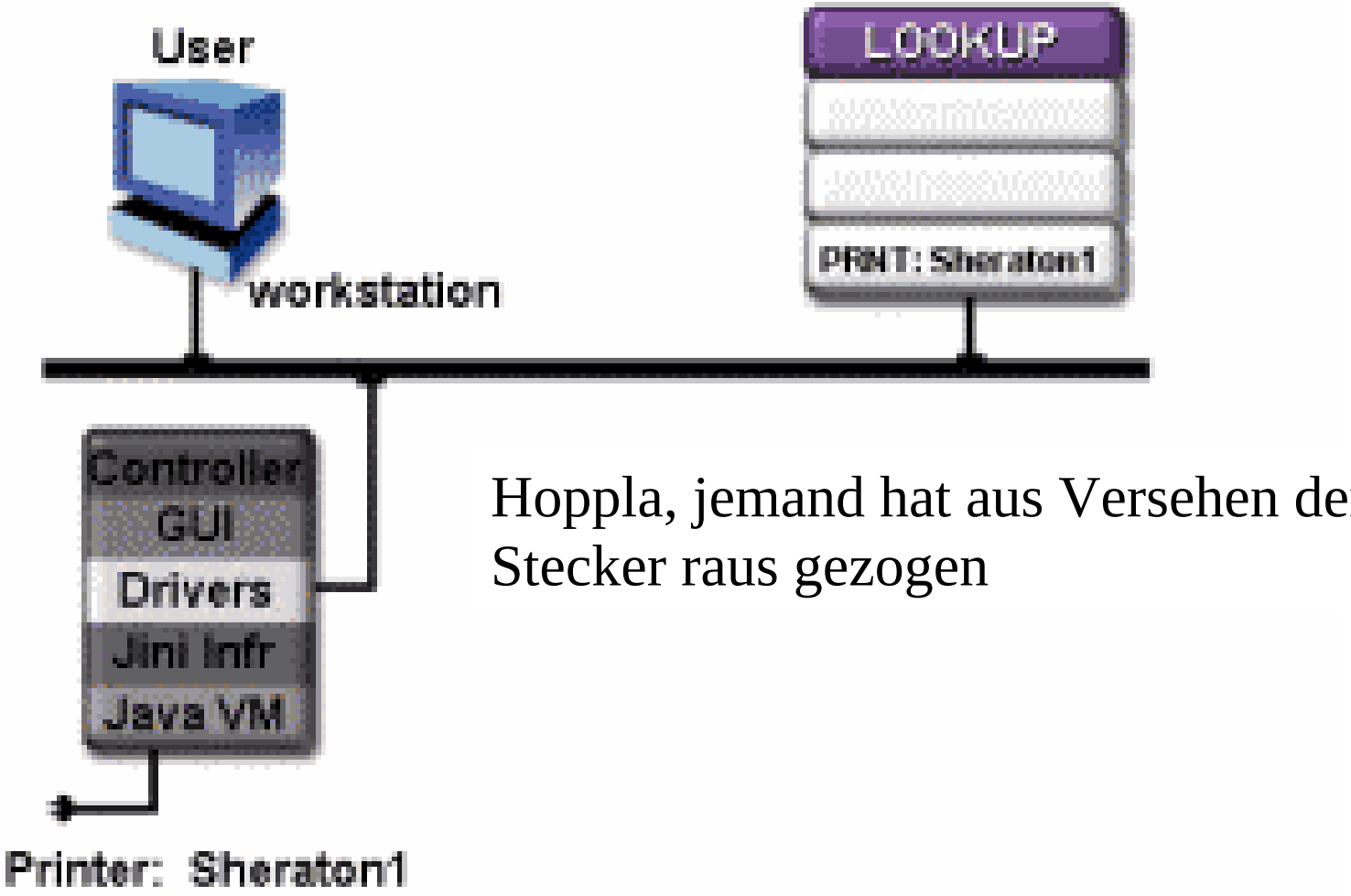
Während dem normalen Betrieb werden alle Knoten *periodisch* geprüft

Distributed Leasing



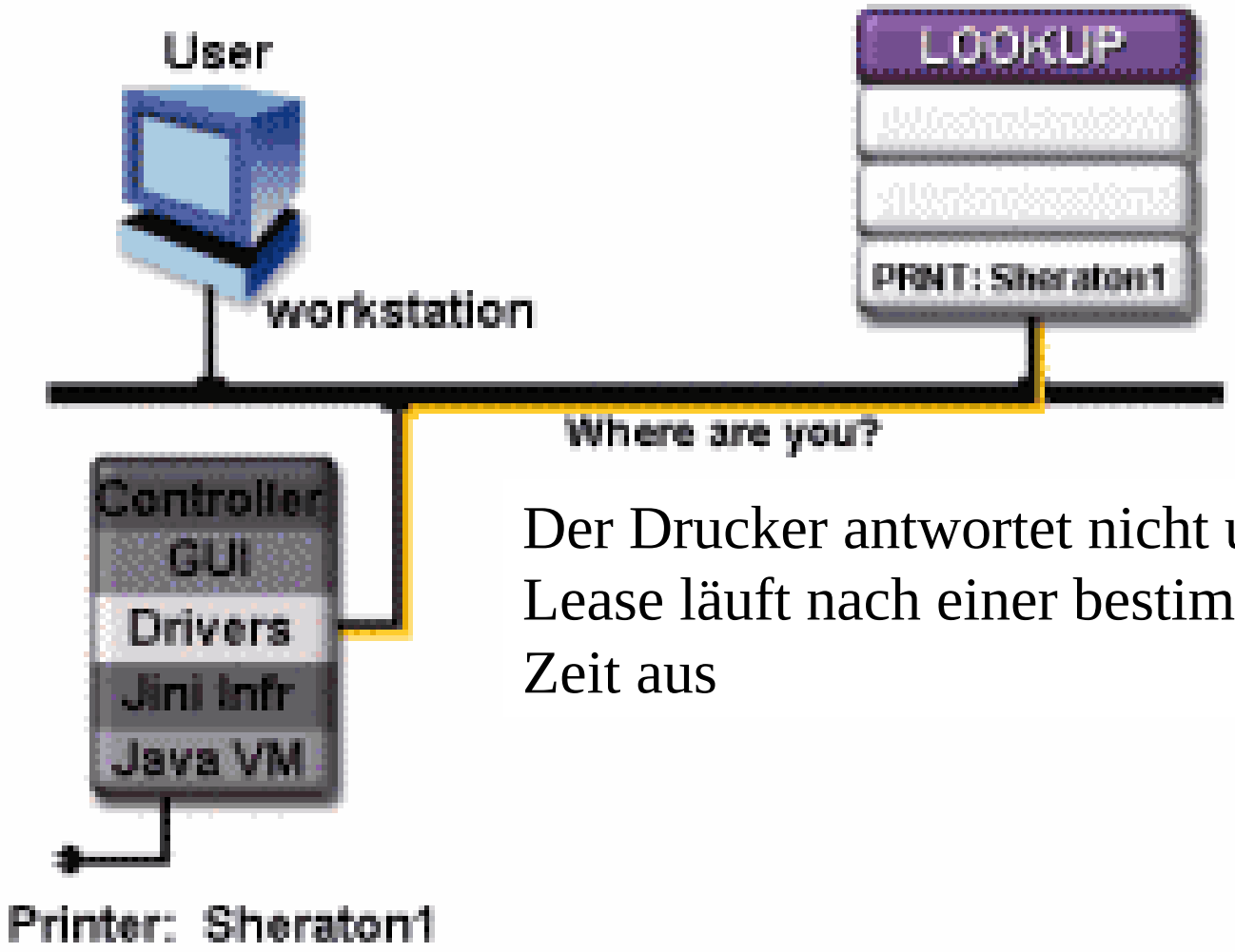
Der Drucker antwortet und teilt mit, dass alles korrekt arbeitet

Distributed Leasing



Hoppla, jemand hat aus Versehen den Stecker raus gezogen

Distributed Leasing

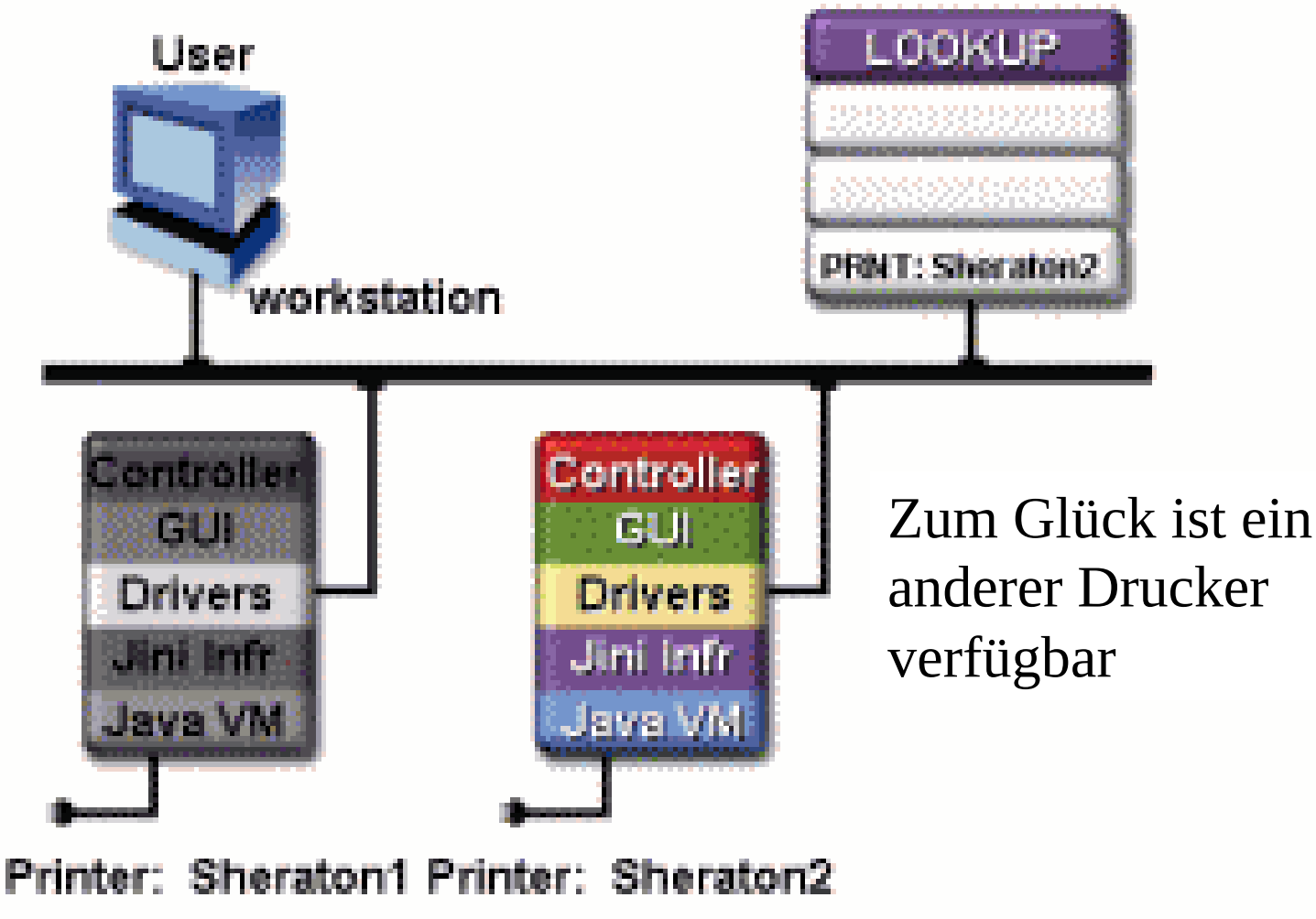


Der Drucker antwortet nicht und das Lease läuft nach einer bestimmten Zeit aus

Distributed Events

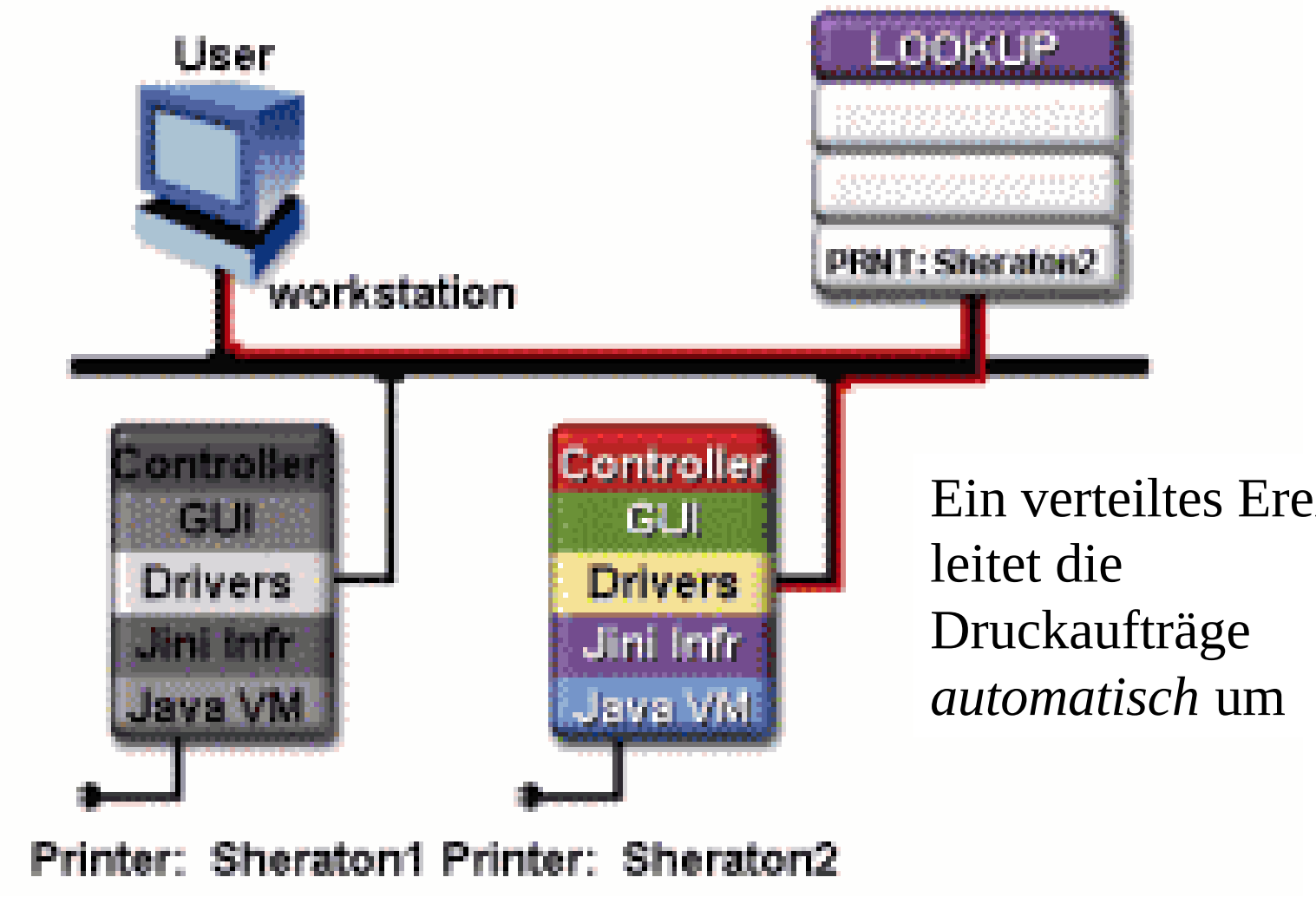
- Erweitert das Java Ereignis Modell
- Registration der Interessen,
  Empfangen von Benachrichtigungen
- gestattet den Einsatz einfacher „Ereignis Manager“
- unterstützt mehrere Liefermodelle
- verwendet verteiltes Lease Protokoll

Distributed Events



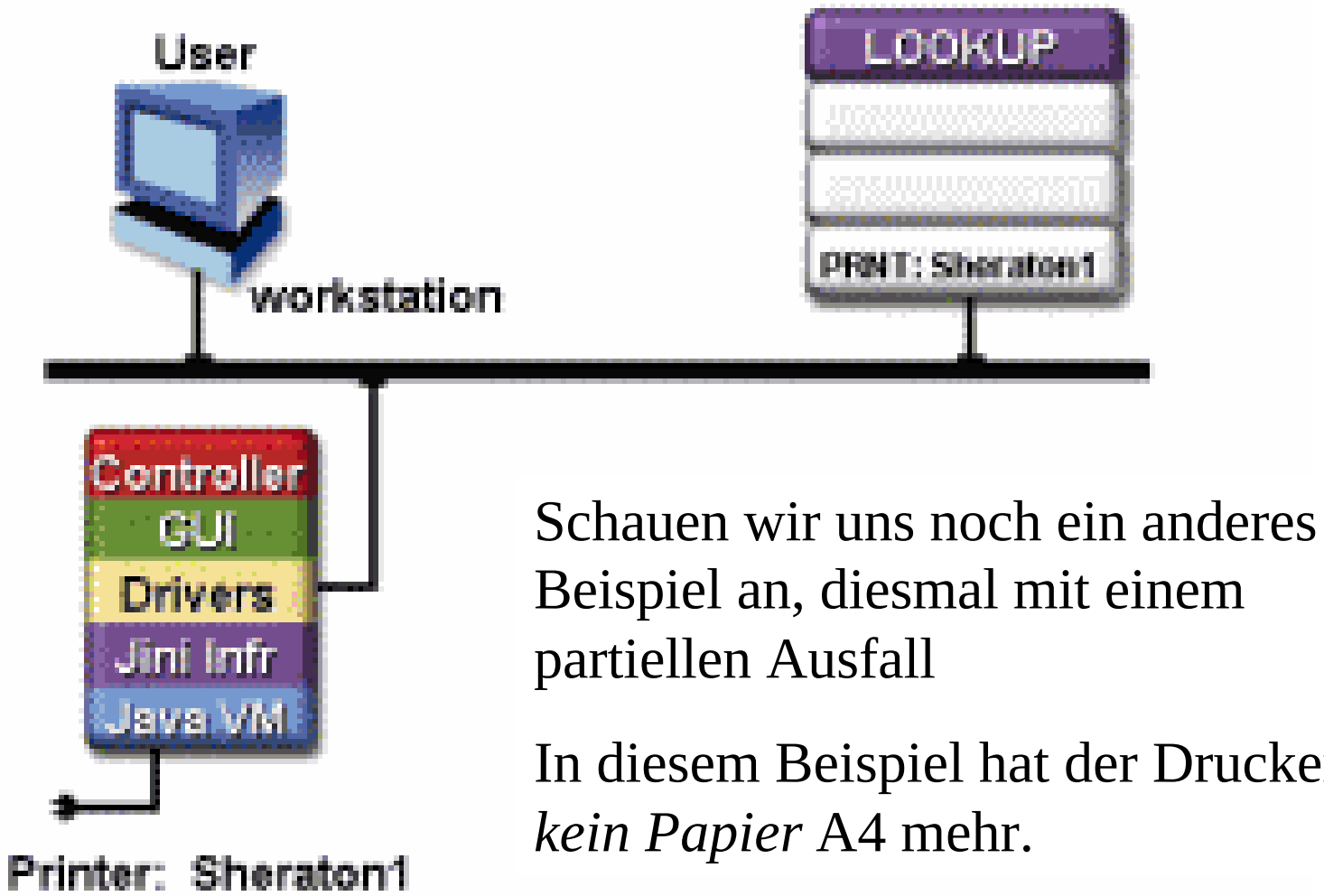
Zum Glück ist ein anderer Drucker verfügbar

Distributed Events



Ein verteiltes Ereignis leitet die Druckaufträge *automatisch* um

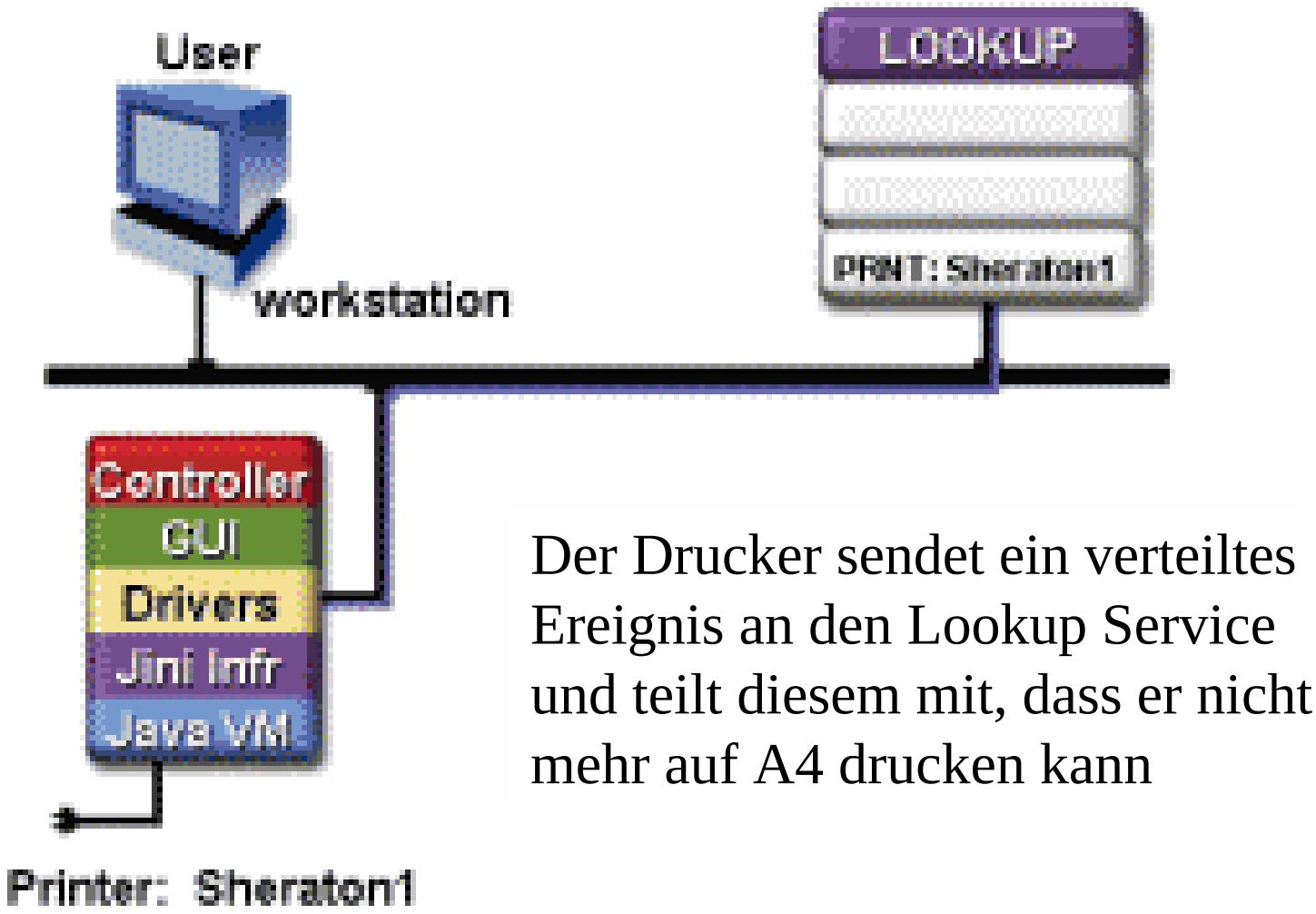
Distributed Events



Schauen wir uns noch ein anderes Beispiel an, diesmal mit einem partiellen Ausfall

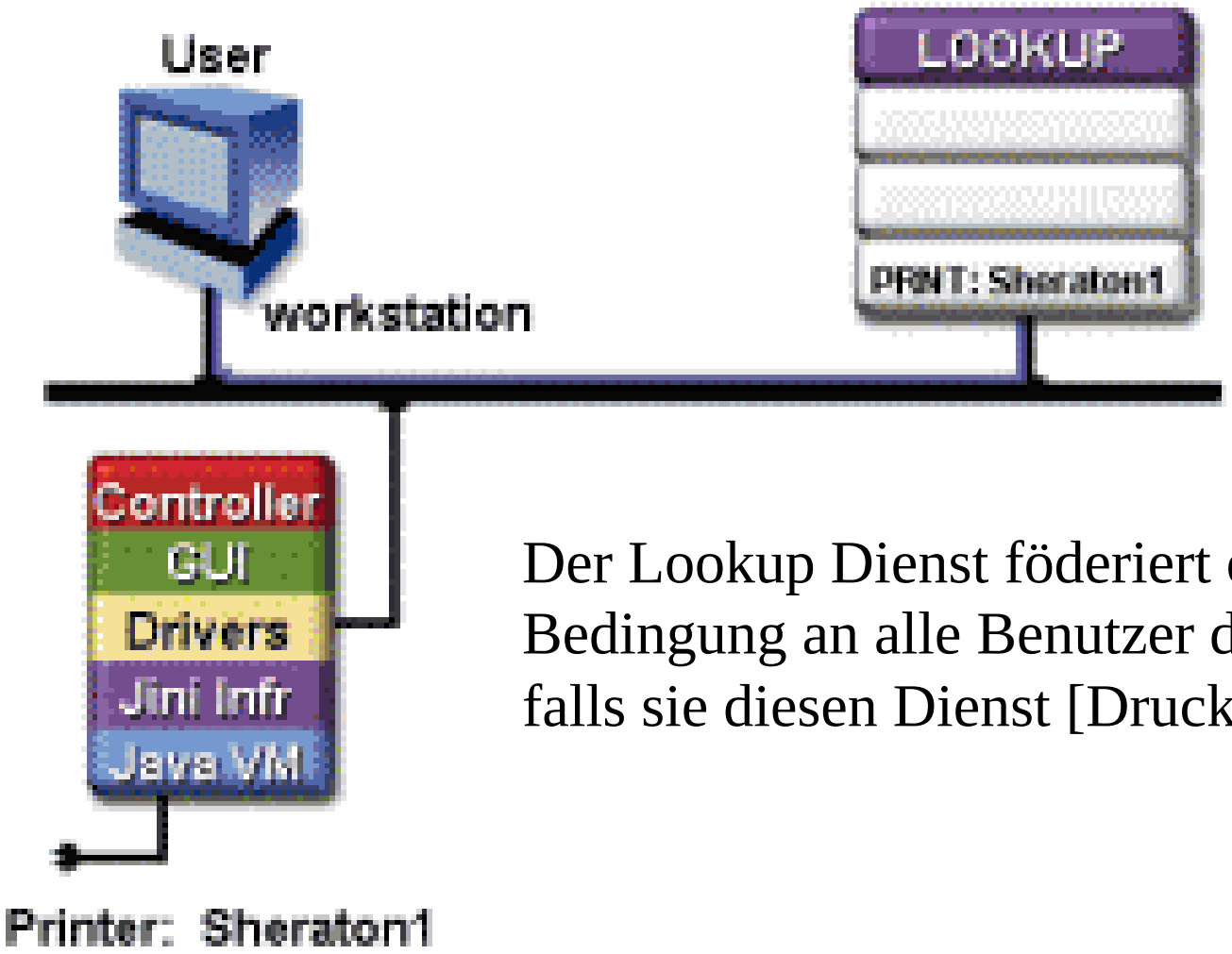
In diesem Beispiel hat der Drucker *kein Papier* A4 mehr.

Distributed Events



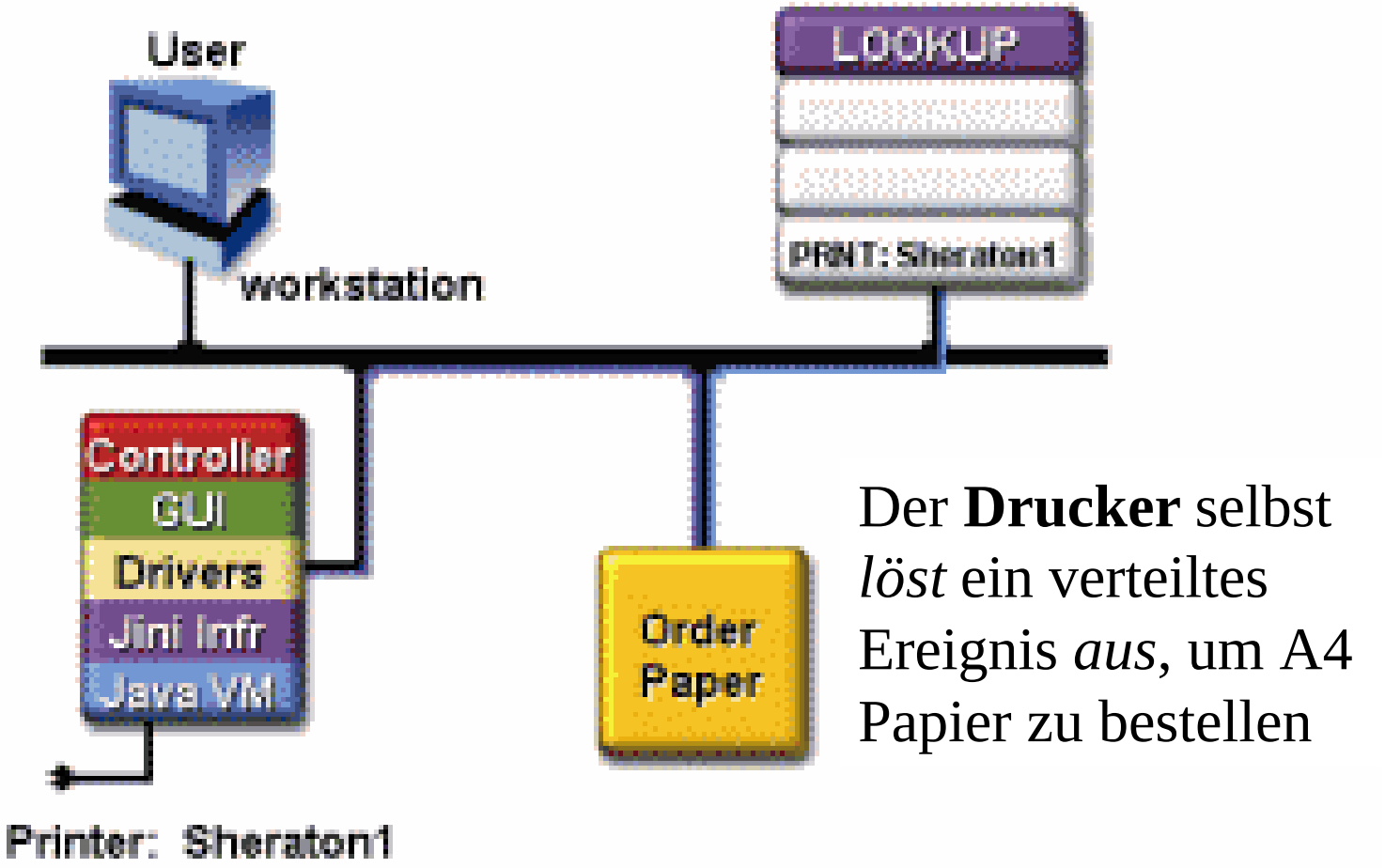
Der Drucker sendet ein verteiltes Ereignis an den Lookup Service und teilt diesem mit, dass er nicht mehr auf A4 drucken kann

Distributed Events



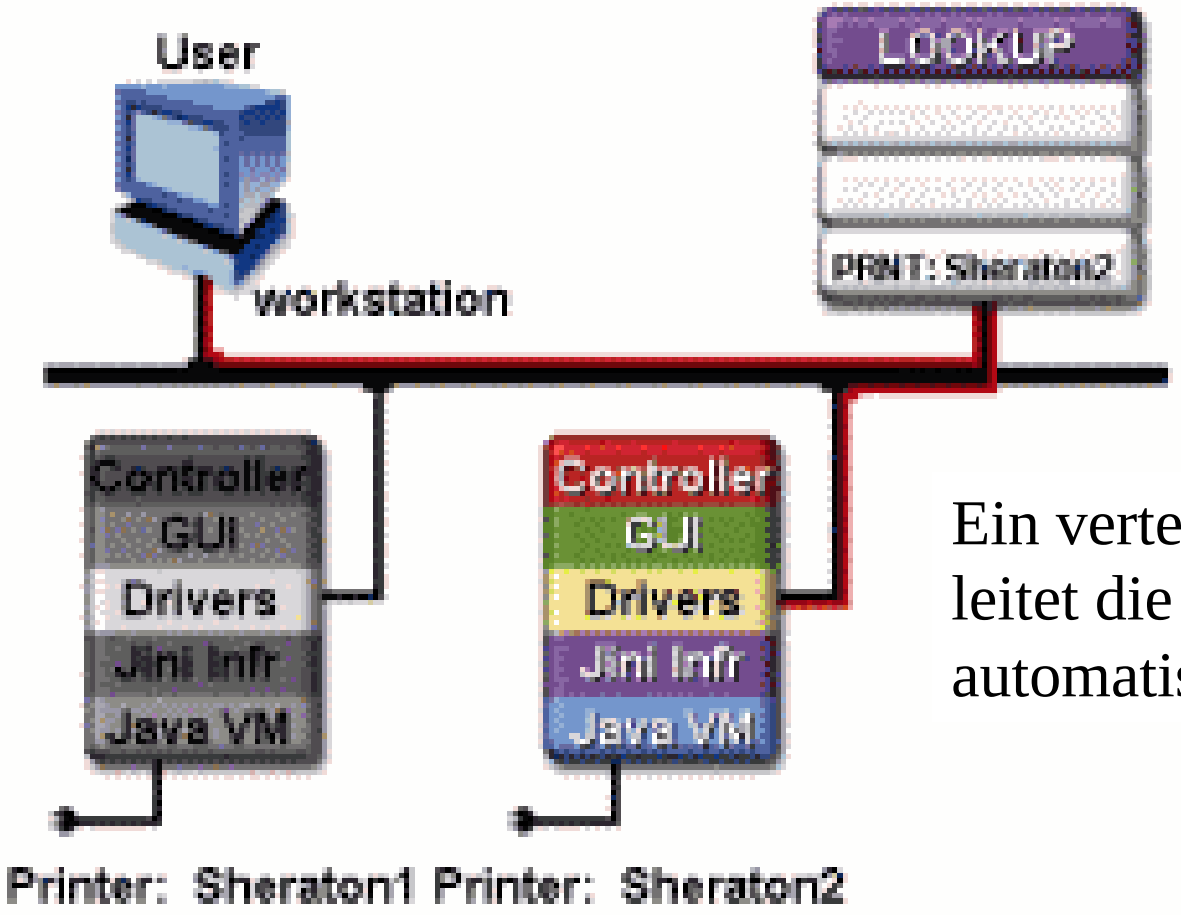
Der Lookup Dienst föderiert diese Bedingung an alle Benutzer des Netzwerks, falls sie diesen Dienst [Drucker] benötigen

Distributed Events



Der **Drucker** selbst *löst* ein verteiltes Ereignis *aus*, um A4 Papier zu bestellen

Distributed Events



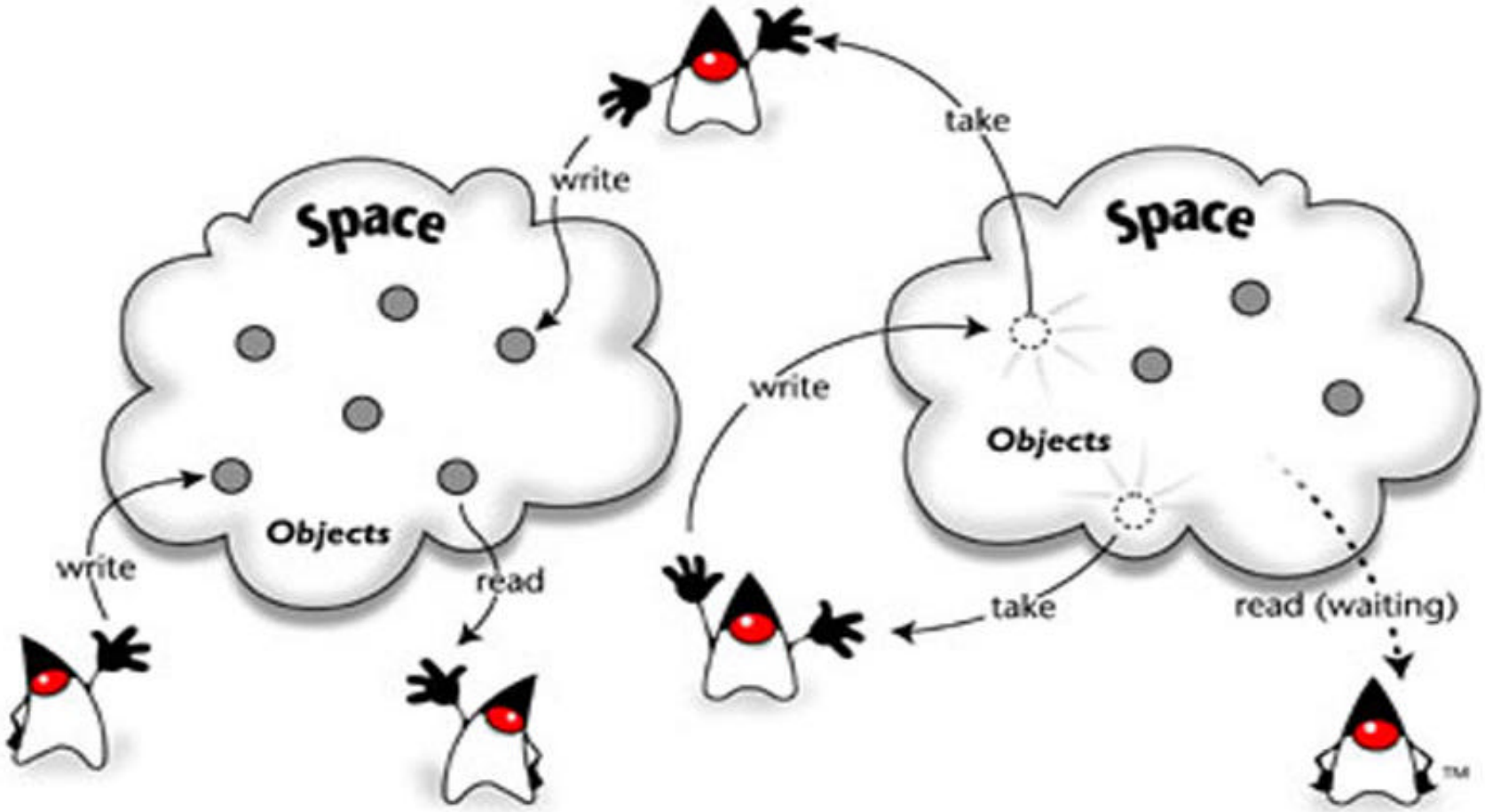
Ein verteiltes Ereignis leitet die Druckaufträge automatisch um

Distributed Transaction - verteilte Transaktion

- Entworfen für verteilte Koordination
- handlich, Objekt-orientiert
- unterstützt verkettete Transaktionen
- benutzt das verteilte Lease Protokoll

Soweit zu den Jini Grunddiensten.
Doch Jini ist mehr ...

JavaSpaces - damit schliesst sich der Kreis



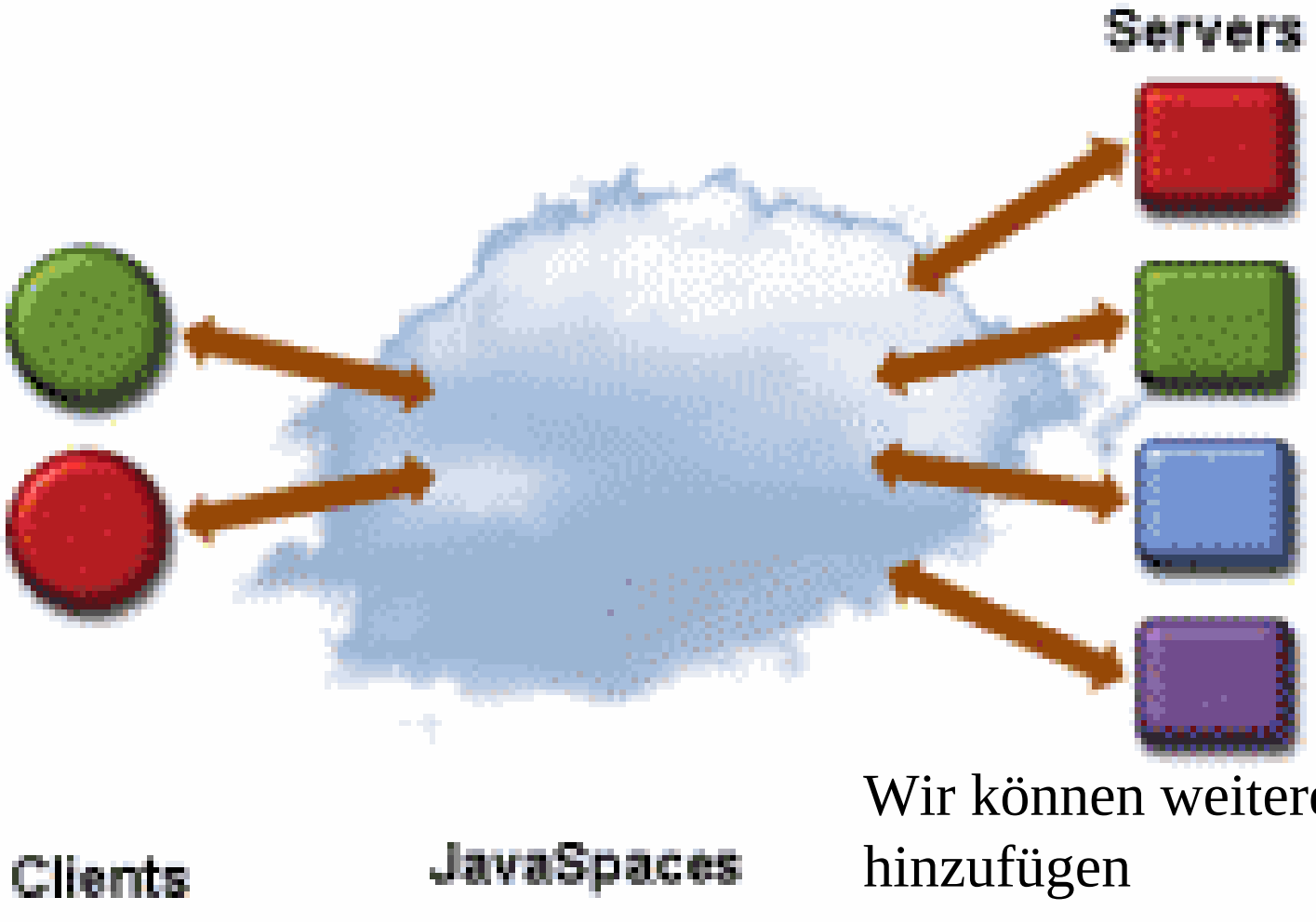
JavaSpaces Technologie Dienste

- Verteilter, gemeinsam genutzter „dynamischer Speicher“
- hilft ein Netzwerk von (J)VMs zu föderieren
- erlaubt eine einfache dynamische Persistenz / Speicherung
- basiert auf dem verteilten Transaktionsprotokoll
- benutzt verteilte Lease und Ereignis-Protokolle

Aufbau

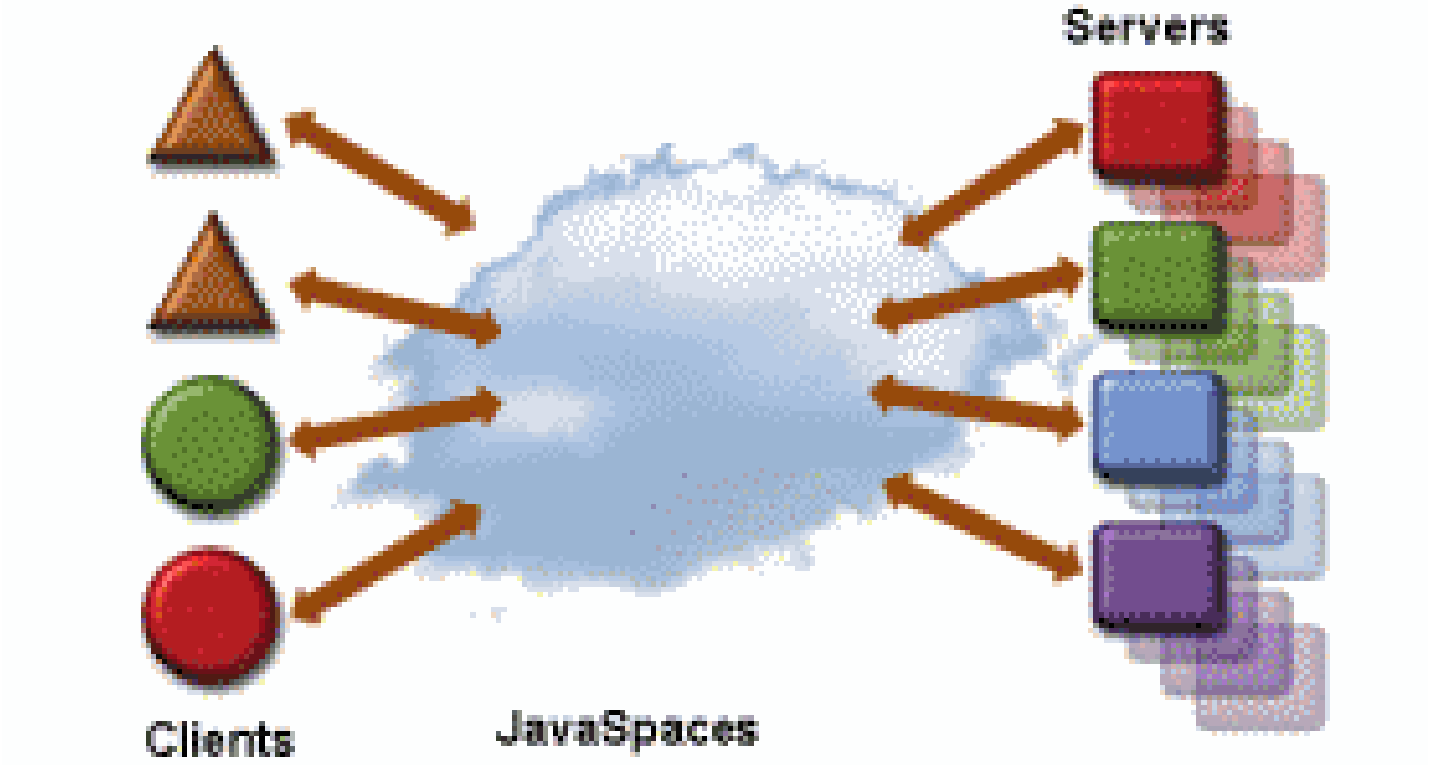
- *Das JavaSpaces Modell*
- *Geschichtlliches*
- *Entries*
- *Beispielszenarien*
- *Das JavaSpaces Interface*
  - *Operationen*
  - *Entries*
  - *Beispielcode*
- *Transaktionen*
- *JavaSpaces und Datenbanken*
- *Bewertung*

JavaSpaces Technologie ist skalierbar



Wir können weitere Server hinzufügen

JavaSpaces Technologie ist skalierbar



... Und noch mehr Server und andersartige Clients ...

JavaSpaces - das Modell

- Programmierung Verteilter Systeme:
  - Austausch von Nachrichten
  - direkte Kommunikation von Prozessen
  - Kontrolle konkurrierender Zugriffe von Clients

  - verlässliche Speicherstrategie
  - verteiltes Transaktionsprinzip (2PC=two phase commit)
  - Aufruf entfernter Methoden (RMI)
  - komplexe Programmierung

Java Spaces - das Modell

- Service stellt einen Speicherbereich (Spaces) für Objekte bereit
- Prozesse kommunizieren nicht direkt miteinander
- Objekte werden zwischen den Services ausgetauscht
- Prozess kann Objekt in den Space schreiben
- im Speicherbereich sind nur *passive* Objekte

das Modell

- Eigenschaften von Spaces
  - mehrere entfernte Prozesse können mit *einem* Space kommunizieren
  - Spaces sind persistent
  - Spaces sind assoziativ
  - Transaktionen sind safe
  - Spaces erlauben den Austausch ausführbarer Inhalte
- Elementare Operationen von Spaces
  - `write()`
  - `read()`
  - `take()`
  - `notify()`

Geschichtliches

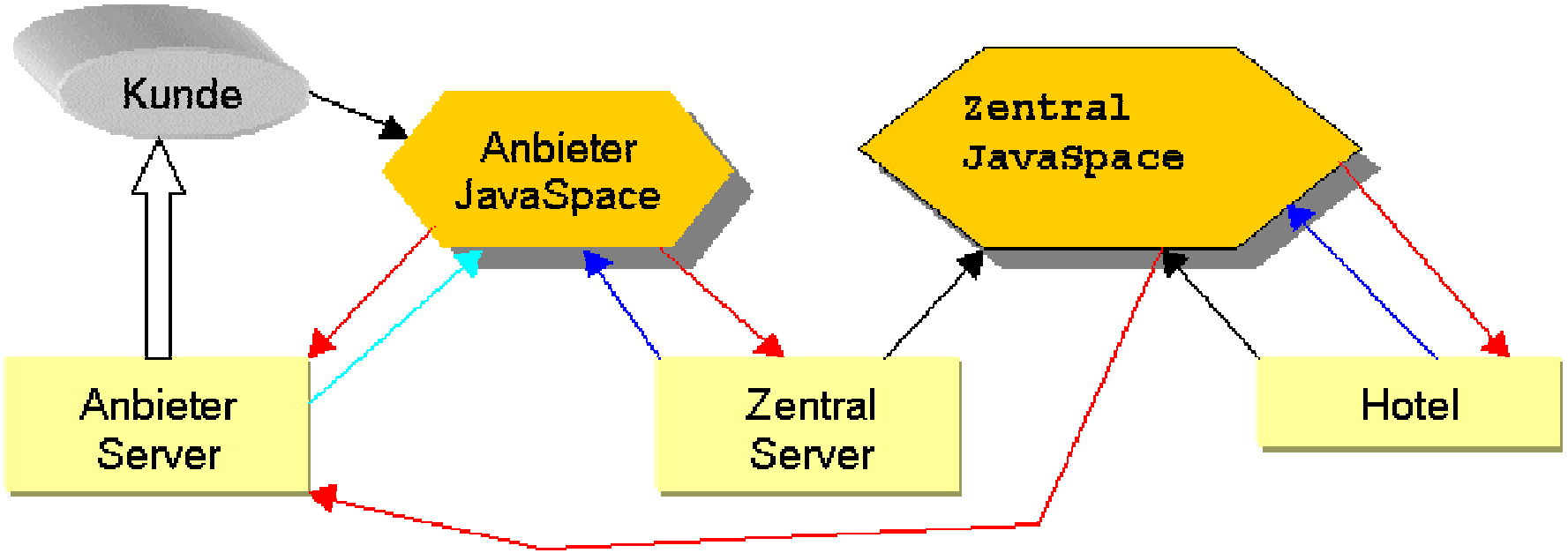
- Linda System : Dr Gelernter, Yale University, 1980
  - kleine Anzahl von Operationen, die auf einen verteilten Speicher (Tuple Space; Tspace) angewendet werden
  - geeignet für eine Vielzahl von verteilter Anwendungen
  - besonders effizient und schnell zu implementieren

- JavaSpaces : Waldo & al, Sun Microsystems, 1997
  - JavaSpaces behält die Ideen vom Linda System bei
  - beschränkt sich auf Java
  - erweitert die Mächtigkeit bisheriger Java Netzwerk Technologien (Sockets, RMI, Jini)

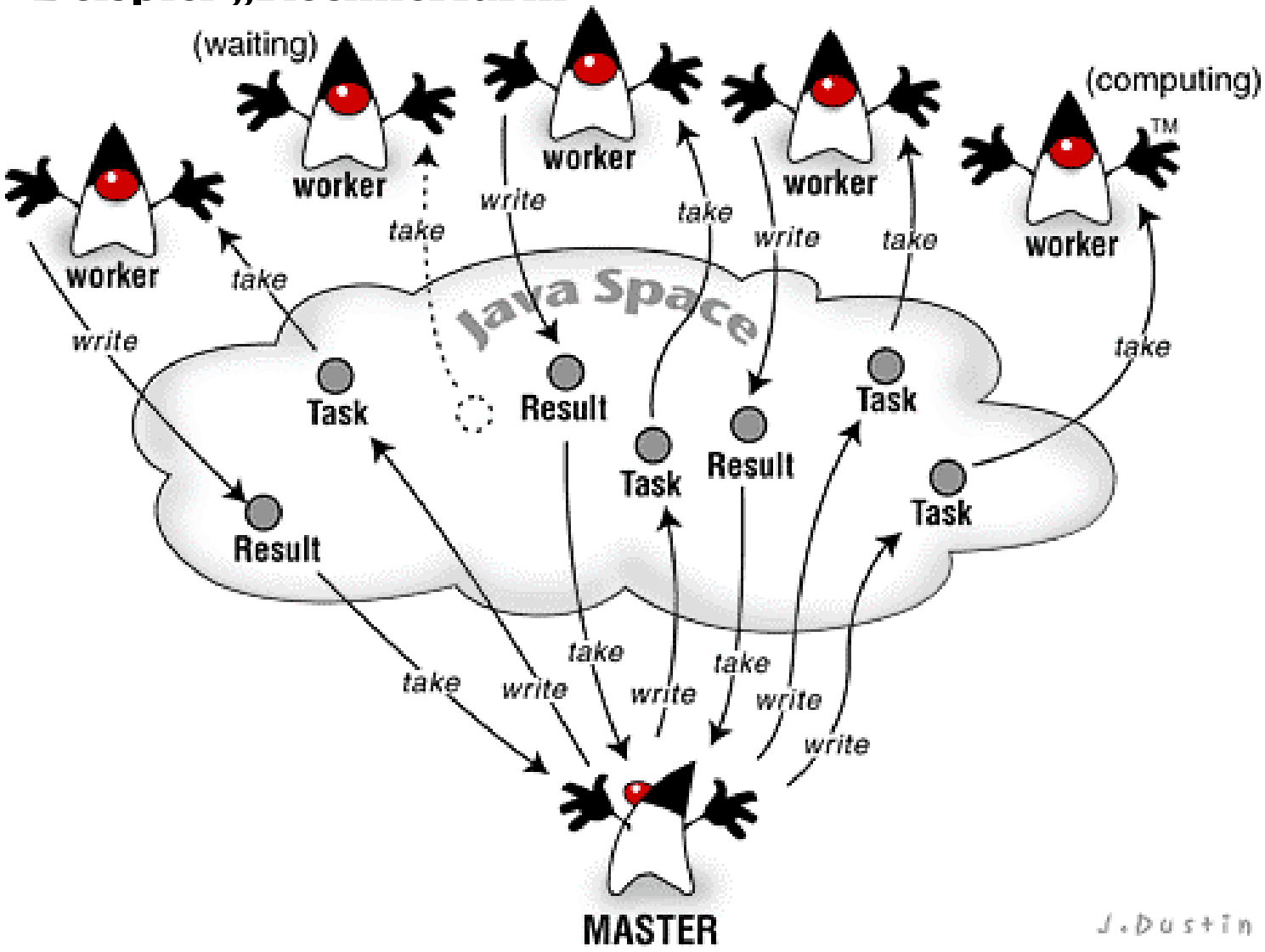
grundlegende Definitionen

- Spaces
  - sind Speicherbereiche für Objekte (rein passive Daten)
- Entries
  - sind Objekte, die mit einem Space in Kontakt treten können
- Templates
  - sind spezielle Entry Objekte, die zur assoziativen Suche in Spaces eingesetzt werden können.

Anwendungsbeispiel „Hotel“



Beispiel „Rechnerfarm“



Beispielszenario „Broker“

Broker System

- der Kunde erstellt ein Kaufgesuch für ein Auto
- registrierte Händler entnehmen die Spezifikation
- der Händler erstellt ein Angebot und plaziert es im Space
- der Kunde holt sich die Angebote ab (oder setzt dafür JMS ein)
- der Kunde entscheidet sich und setzt sich mit dem Händler in Verbindung

API

- Voraussetzungen
  - Java Virtual Machine
  - Jini Technologie (Starter Kit)
  - Java Spaces Technologie (Java Spaces Technology Kit)

- JavaSpaces ist ein Jini Service (`net.jini.space`)

- Spaces erlauben nur folgende Operationen
  - `read, readIfExists`
  - `take, takeIfExists`
  - `notify, snapshot, write`

API

- Basiert auf
  - Java Remote Methode Invocation (RMI)
  - Java Objekt Serialisierung
  - Jini Entry Spezifikation, Java Entries Utilities Spezifikation
  - Jini Distributed Event Spezifikation
  - Jini Distributed Leasing Spezifikation
  - Jini Transaction Spezifikation

API


```
package net.jini.space;
    import net.jini.coreevent.*;
    import net.jini.coretransaction.*;
    import net.jini.corelease.*;
    public interface JavaSpace {
            Lease write(Entry e, Transaction tx, long lease);
            public final long NO_WAIT=0;
            Entry read(Entry e, Transaction tx, long timeout);
            Entry readIfExist(Entry e, Transaction tx, long timeout);
            Entry take(Entry e, Transaction tx, long timeout);
            Entry takeIfExist(Entry e, Transaction tx, long timeout);
            EventRegistration notify(Entry  tmp, Transaction tx, ...)
            Entry snapshot(Entry e) throws RemoteException;
    }
```

Entry

- Entries sind Objekte der Java Programmiersprache, die in einem Space gespeichert werden.
- Klassen, die das Interface `net.jini.core.entry.Entry` implementieren
- das Interface enthält keine Konstanten oder Methoden
- Besonderheiten der Entry Klassen:
  - argumentfreier Konstruktor
  - Felder müssen `public` sein
  - Variablen sind Referenzen auf Objekte

Entry Beispiel

- Message Entry

```
import net.jini.core.entry.Entry.*;
public class Message implements Entry {

        public String content;

        public Message() { // leerer Konstruktor }
}
```

Das Hello World Programm

- Quelle : JavaSpaces Principles, Patterns and Practice (Sun : Freeman et al)

```
package javaspace.helloWorld;

import net.jini.core.entry.Entry;

public class Message implements Entry {
    public String content;

    public Message() {
    }
}
```

Das Hello World Programm

- Quelle : JavaSpaces Principles, Patterns and Practice (Sun : Freeman et al)

```
package javaspace.helloWorld;

import javaspace.SpaceAccessor;

import net.jini.core.lease.Lease;
import net.jini.space.JavaSpace;
```

Das Hello World Programm


```
public class HelloWorld {
    public static void main(String[] args) {
        try {
            Message msg = new Message();
            msg.content = "Hello World";
            JavaSpace space = SpaceAccessor.getSpace();
            space.write(msg, null, Lease.FOREVER);
            Message template = new Message();
            Message result =
                (Message)space.read(template, null, Long.MAX_VALUE);
            System.out.println(result.content);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Das Hello World Programm

- Quelle : JavaSpaces Principles, Patterns and Practice (Sun : Freeman et al)

```
package javaspace.helloWorld;

import net.jini.core.entry.Entry;

public class Message implements Entry {
    public String content;

    public Message() {
    }
}
```

Transaktionen

- JavaSpaces verwendet den Jini Transaktionsmanager
  - two phase commit (2PC) Protokoll
  - ACID
    - atomar
    - consistent
    - isolation
    - durability
  - Space koordiniert nebenläufige Zugriffe
    - Synchronisation wird von Space erledigt

JavaSpaces & Datenbanken

- JavaSpaces
  - JavaSpaces speichert (passiv) Java Objekte
  - die Suche geschieht assoziativ
    - inhaltsbasierte Suche
  - soll Netzwerkprogrammierung vereinfachen

- Datenbanken
  - Daten stehen in Beziehung zueinander
  - relationales Modell soll Datenbanksysteme vereinfachen

Zusammenfassung

- JavaSpaces speichern Objekte
  - zur Nutzung der Objekte müssen diese
    - gelesen werden : eine Kopie wird erstellt (`read()`)
    - aus dem Space entfernt werden (`take()`)
  - ein Nutzer kann den Eventdienst von Jini nutzen, um benachrichtigt zu werde, sobald ein Objekt eingelagert wird, welches den „Interessen“ des Nutzers entsprechen

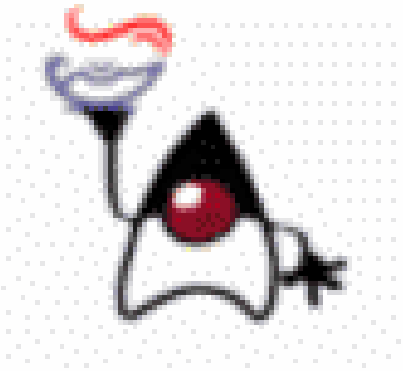
Jini Technologie - Wertschöpfung

- keine Konfiguration für den Anwender
- einfachere Auslieferung für den Hersteller
- neuer Markt für Produkte und Dienste
- vereinfachte Entwicklung für Programmierer

Jini Technologie - Möglichkeiten

- In der Fabrik
- im Büro
- zu Hause
  - intelligente Haushaltgeräte
- in Spezialgeräten
  - Spielkonsolen

Jini Technologie - reale Einsätze

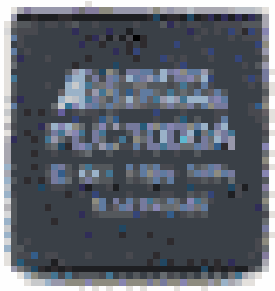


Wird die Technologie wirklich eingesetzt?

Wer arbeitet heute an Jini Projekten?

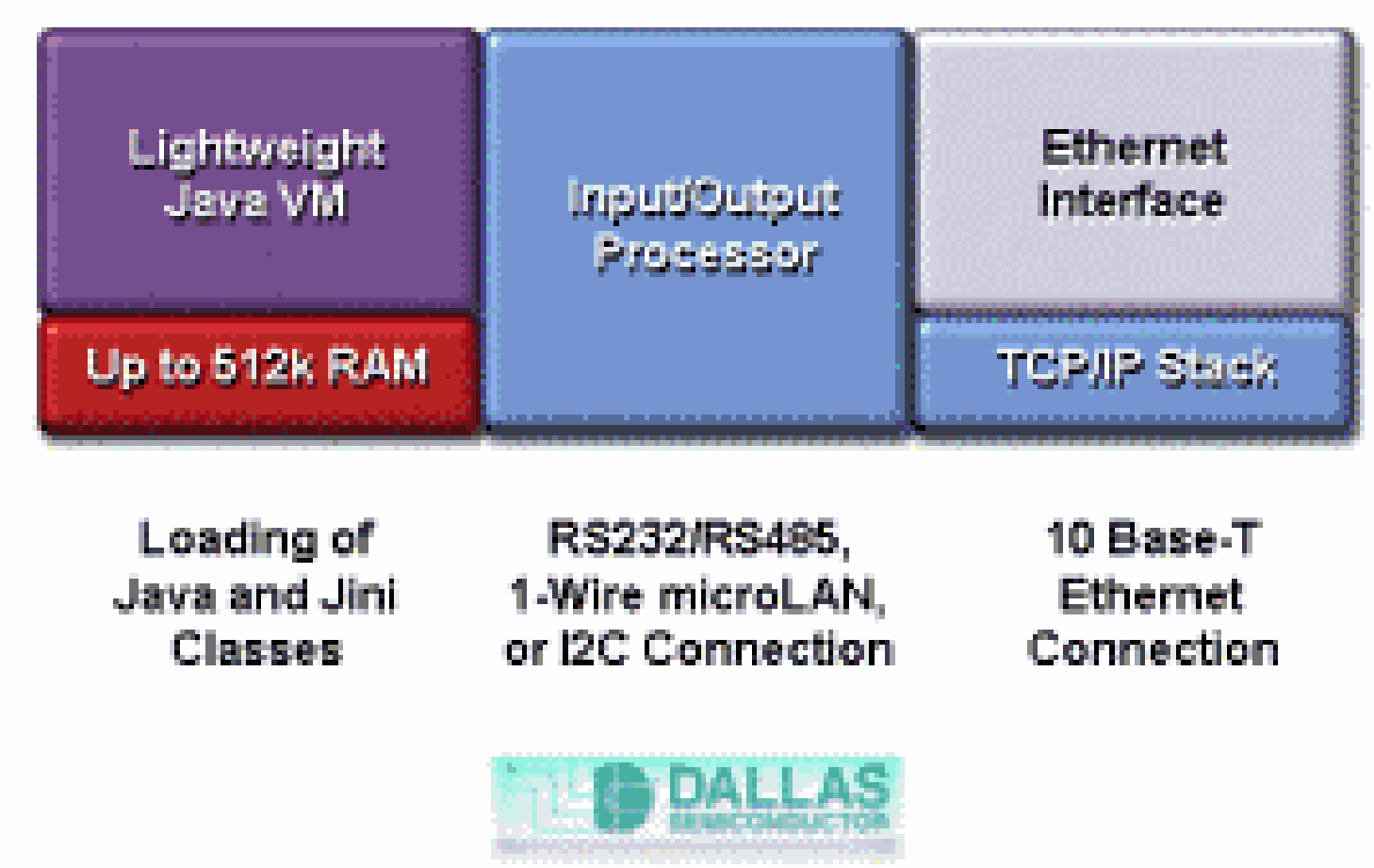
Adaptive Networks

- Jini Technologie kommuniziert mit Hilfe des überall verlegten Stromkabels



ADAPTIVE NETWORKS

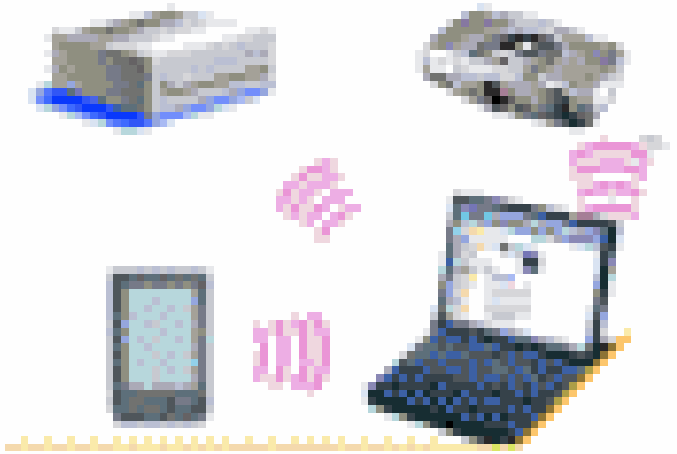
Dallas Semiconductors



Anwendungsbeispiel : der iButton oder der JavaRing

Ericsson

- Jini Technologie fähiges Proximity Netzwerk bei Ericsson
  - Kombination von Jini mit Bluetooth Technologie
  - mobile Kommunikation

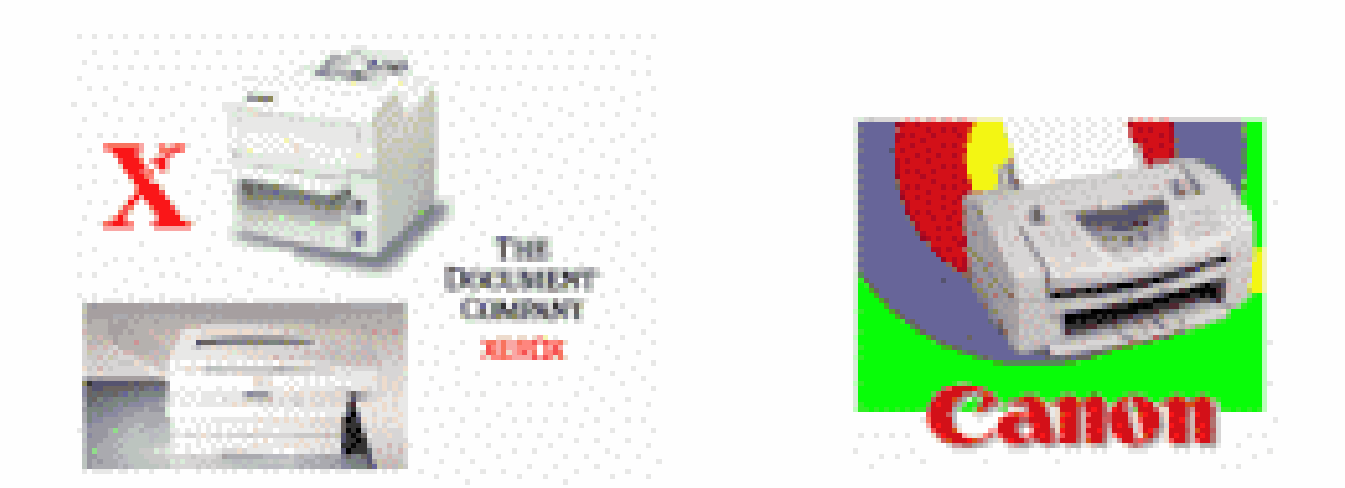


Quantum und Seagate

- Jini-vorbereitete Festplatten : Quantum, Sun, Seagate

Xerox und Canon

- Drucker mit eingebauter Jini Technologie

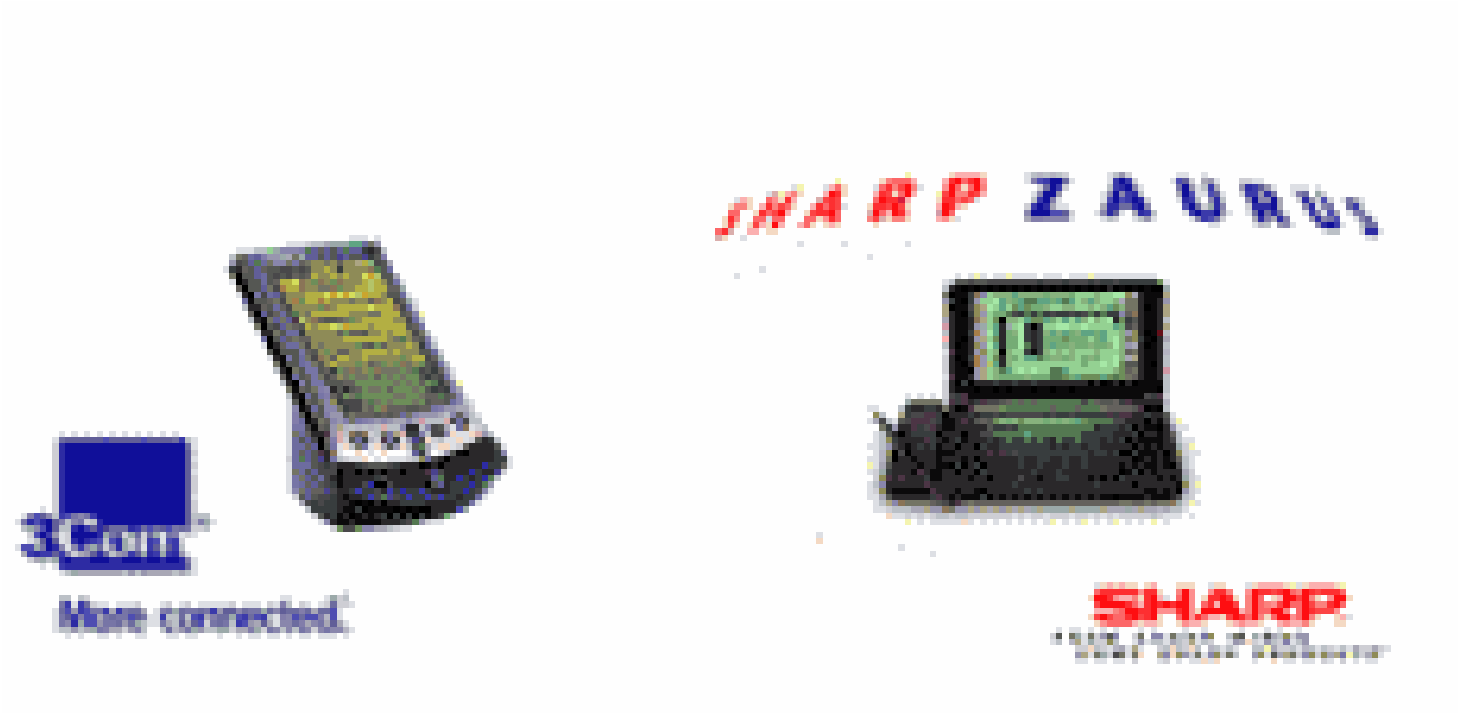


Epson

- Jini-vorbereitete Drucker und Projektoren

3Com und Sharp

- Jini vorbereitete PDAs



Nokia

- Mobile Telefonie

Kodak

- Jini vorbereitete Kameras

Sony

- Mini-Discs und Unterhaltungselektronik

Philips

- Jini in Fernsehern und DVDs

Siemens

- Jini Technologie in der Geschirrspühlmaschine

CISCO

- Jini im Netzwerkbereich

Computer Associates

- Jini ermöglicht intelligentes Netzwerkmanagement

Metroworks

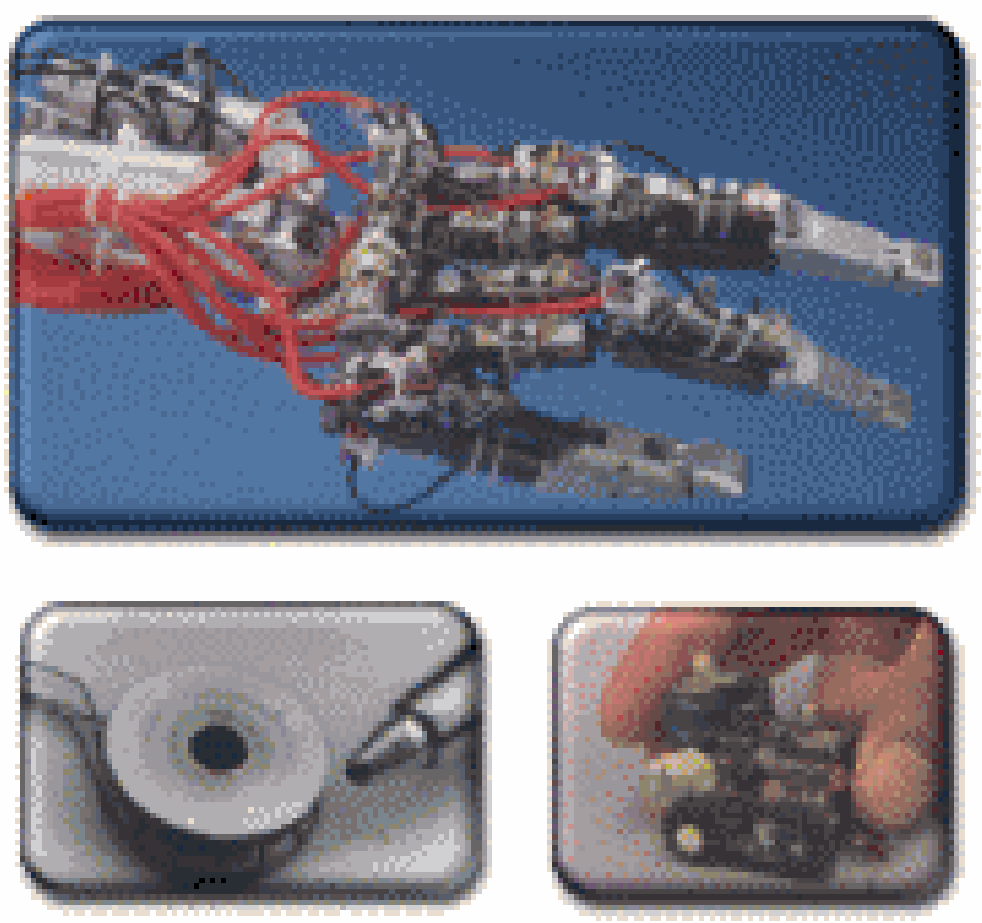
- Jini im Software Entwicklungsprozess

Symbian

- Jini für eingebettete Systeme

... und noch weitere ...

Und die Zukunft?



Ausserhalb traditioneller Kommunikation

- Kabellose Verbindungsgeräte
- SmartCards
- vernetztem eingebettete Microcontroller
- intelligente kollaborative Geräte
- stark verteilte Systeme