

7 Fernaufrufe

7.1 Fernaufrufe und Verteilungsabstraktion	3
7.2 Implementierung von Fernaufrufen	11
7.3 Mängel bei der Verteilungsabstraktion	20
Zusammenfassung	35

Motivation:

- Bei einer Vielzahl von Diensten im Netz legt der Klient auf die asynchrone Ausführung gar keinen Wert.
- Aufrufbares Modul/Objekt/Klasse ... würde genügen.

Ziel:

- Programmiert wird aufrufbasiert.
- Aufrufer und Aufgerufener können auf verschiedene Rechner plaziert werden.
- Aufruf und Rücksprung werden nicht durch Sprungbefehle, sondern durch Nachrichten-Versendung/Empfang realisiert.

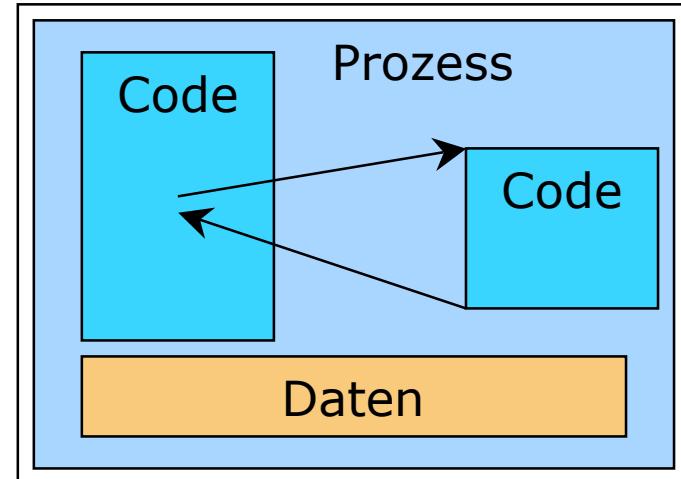
7.1 Fernaufrufe und Verteilungsabstraktion

Def.: Ein Aufruf ist als **Fernaufruf** (*remote invocation*) implementiert, wenn der Aufgerufene von einem anderen Prozess in einem anderen Adressraum - und eventuell in einem anderem Rechner - als dem des Aufrufers ausgeführt wird.

Achtung: Dies hat zunächst nichts mit dem Dualitätsprinzip (5.1) zu tun. Bei der Programmierung sowohl des Aufrufers als auch des Aufgerufenen spielt Nichtsequentialität keine Rolle!

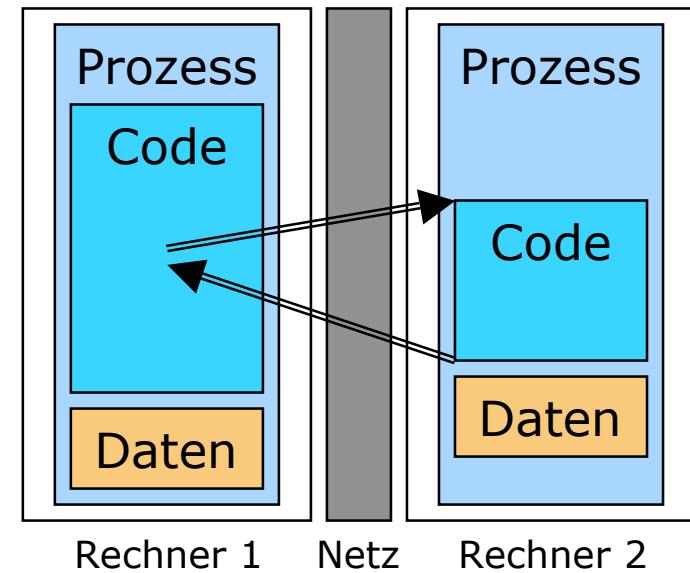
➤ *Lokaler Aufruf:*

- Argumente bereitstellen (Keller)
- Sprung zu aufgerufenem Code
- Ergebnisse bereitstellen (Keller und/oder Halde)
- Rücksprung zum Aufrufer



➤ *Fernaufruf:*

- Argumente in Nachricht verpacken
- Nachricht vom Klienten an Server
- Server stellt Ergebnisse bereit
- Ergebnisse in Antwort verpacken
- Antwort von Server an Klienten



- **Prozedurfernaufruf** (*remote procedure call, RPC*)
 - bei prozeduralen Sprachen (Modula, C, ...)
 - aufgerufen wird Prozedur eines Moduls (oder Klassenmethode)
 - historisch die *erste Fernaufruf-Variante* [Birrell/Nelson 1984]
 - auch für heterogene Sprachwelten: *DCE, DCOM, ...*
- **Objektfernaufruf** (*remote object invocation, ROI*)
 - bei objektorientierten Sprachen (Java, C++, Eiffel, C#, ...)
 - aufgerufen wird eine Operation/Methode eines Objekts
 - Java: „*Remote Method Invocation*“, *RMI*
 - C#: „*.NET Remoting*“
 - auch für heterogene Sprachwelten: *CORBA*

Ziel eines guten Fernaufruf-Systems
ist die Erreichung eines möglichst hohen Grades von
Verteilungsabstraktion (*distribution transparency*).

Verteilungsabstraktion hat mehrere Facetten:

- Zugriffs- Abstraktion (*access transparency*)
 - Lage/Orts- Abstraktion (*location transparency*)
 - Migrations- Abstraktion (*migration transparency*)
 - Replikations- Abstraktion (*replication transparency*)
- und weitere . . .

(vgl. 1.2)

Zugriffsabstraktion (objektorientiert):

Zugriff auf entferntes Objekte unterscheidet sich weder syntaktisch noch semantisch (!) von einem lokalen Zugriff:

```
Set<String> words = .....; words.add(word);
```

Dieser Aufruf kann zur Laufzeit als *lokaler Aufruf*, möglicherweise aber auch als *Fernaufruf* stattfinden - abhängig davon, wo das bezeichnete Objekt liegt.

Ortsabstraktion :

Wo ein nichtlokales Objekt tatsächlich liegt, ist irrelevant und spiegelt sich *nicht* im Code wieder. Im Beispiel

`x.op(arg)`

folgt das aus der Zugriffsabstraktion.

Bei der Erzeugung von Objekten dagegen,

`x = new X(arg);`

wäre die Ortsabstraktion verletzt, wenn eines der Argumente einen bestimmten Rechner identifizierte.
(Hier lassen wir zunächst offen, wo das Objekt erzeugt würde.)

Allerdings: häufig ist Ortsabstraktion *nicht* erwünscht.

Migrationsabstraktion :

Womöglich bleibt das Objekt nicht dort liegen, wo es erzeugt wurde, sondern es wird woandershin verlagert (es „wandert“, „migriert“), z.B. aus Effizienzgründen.

Im Code ist davon nichts zu sehen.

Achtung: häufig ist Migrationsabstraktion *nicht* erwünscht.

Replikationsabstraktion :

Womöglich werden im Netz mehrere Kopien des Objekts gehalten, z.B. aus Effizienz- und/oder Sicherheitsgründen.

Im Code ist davon nichts zu sehen.

Aber nie vergessen:

Ein Fernaufruf ist um mehrere Größenordnungen
aufwendiger als ein lokaler Aufruf!

7.2 Implementierung von Fernaufrufen (objektorientiert)

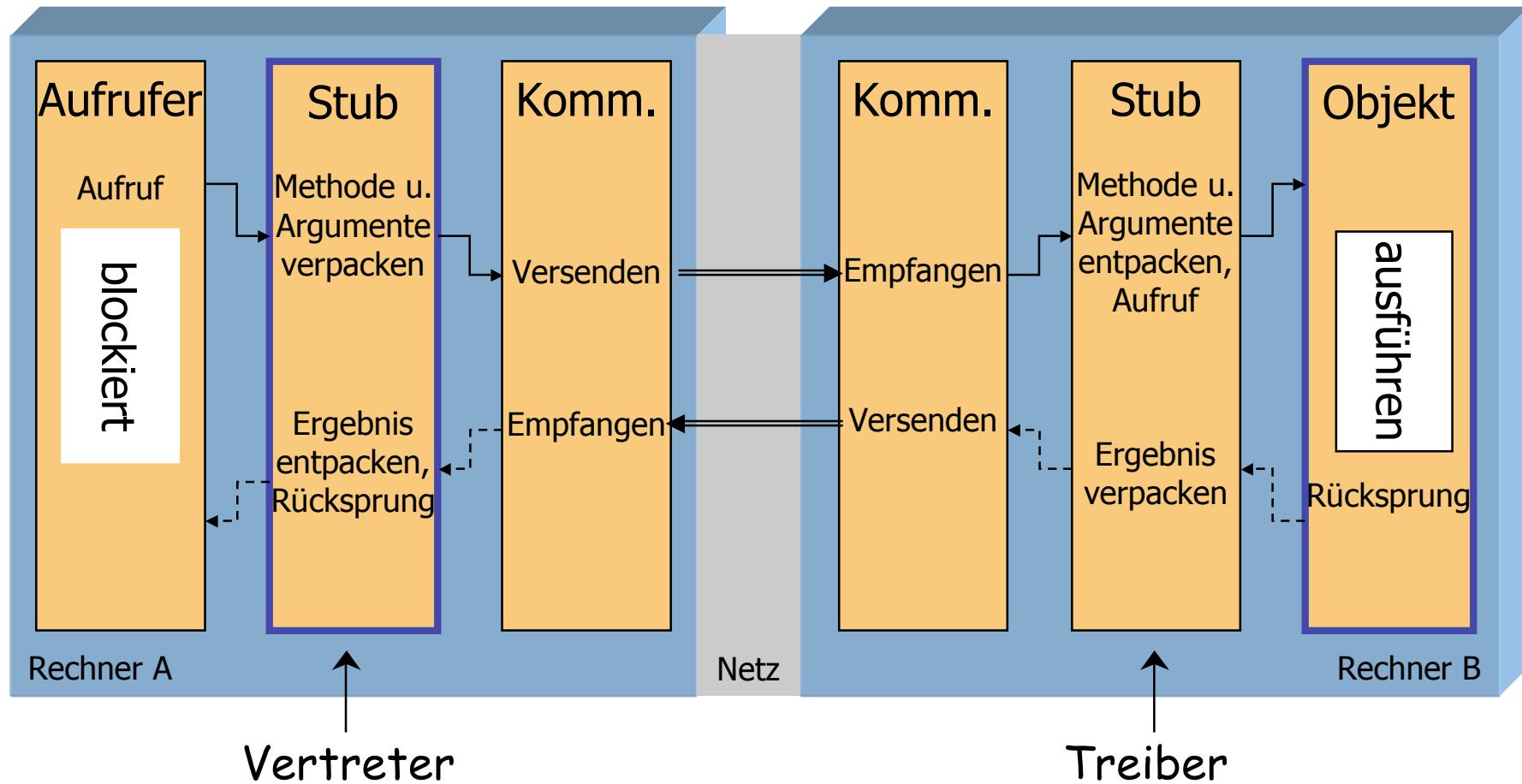
Dem Aufrufer wird ein **Stellvertreter** (*proxy, client stub*) des aufgerufenen Objekts untergeschoben, der die gleiche Schnittstelle wie das Objekt hat.

In einem Prozess des entfernten Rechners:

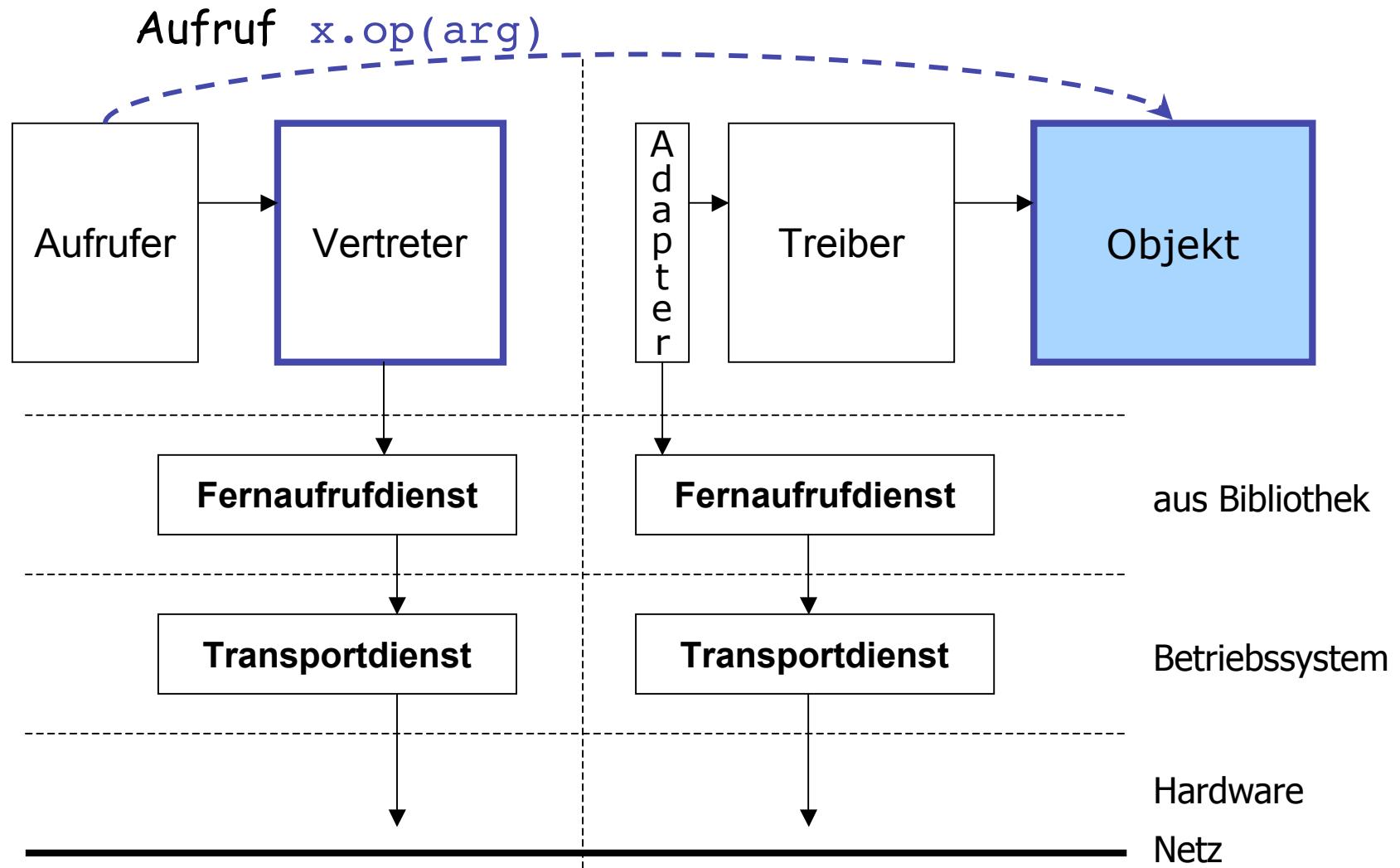
Das Objekt wird von einem **Treiber** (*skeleton, server stub*) aufgerufen, der den Aufrufer vertritt.

stub = Stumpf, Stummel

Aufrufe und Datenfluss



Funktionale Hierarchie



Packen und Entpacken

Die Hauptaufgabe der Stubs ist das *Packen* und *Entpacken* von Argumenten und Ergebnissen (auch *De/Serialisierung* oder *De/Linearisierung*, engl. *marshaling/unmarshaling*):

Zum *Versenden* müssen die Daten aus dem Adressraum des Senders zusammengesammelt und in eine Nachricht verpackt werden - gemäß einem **Fernaufruf-Protokoll**.

Nach dem *Empfangen* der Nachricht müssen die Daten im Adressraum des Empfängers so arrangiert werden, wie sie beim Sender arrangiert waren.

Client Stub für Objekt o mit Methoden m1, m2, ...:

Methode `float m1(int i, char c, out double d)`:

Nachricht (m1, i, c) verschicken an (host, port);

Nachricht (x, result) entgegennehmen;

`d = x;`

`return result;`

Ergebnisparameter

Methode `void m2.....`

Server Stub für Objekt o:

Nachricht (m, a1, a2, a3, ...) entgegennehmen;

Fall m==m1 : Aufruf `f = o.m1(a1, a2, y);` ;

Nachricht (y, f) verschicken.

Fall m==m2:

(alternativ: Server Stub für alle Objekte mit Schnittstelle s)

Fernafrufdienst

ist zuständig für die anwendungsunabhängigen Teile:

- Identifizierung der Partnersysteme
- Identifizierung der entfernten Objekte bzw. Aufrufer
- Einrichtung der benötigten Sockets/Verbindungen
- Aufruf des Transportdienstes
- Sicherung bei Einsatz von UDP

Adapter

ist ein eigener Thread, der wiederholt folgendes tut:

- Entgegennahme einer Nachricht vom Transportdienst
- Übergabe der Nachricht an einen eigenen Thread:
 - dieser ermittelt das Zielobjekt
 - und ruft den entsprechenden Server Stub auf

Stub-Erzeugung

Der Code kann aus einer *formalen* Beschreibung der Schnittstelle *automatisch* erzeugt werden (z.B. bei Java aus einem [interface](#)). Dies leistet ein **Vertreter/Treiber-Generator** (*stub generator*).

Frühe Fernaufrufsysteme:

Expliziter Einsatz des *Stub Generator* und manuelle Bereitstellung der in den jeweiligen Systemen benötigten Stubs.

Heutzutage (Java RMI, .NET Remoting, ...):

Vollautomatische Erzeugung der Stubs

Fernbinden (*remote linking*)

Wie erlangt der Aufrufer Kenntnis von Zielobjekt/Klasse?

„Private Objekte“: `x x = new X(...);`

Klassenname und/oder Konstruktorargumente
können zum Zielrechner und dortiger `.class`-Datei
in Bezug gebracht werden.

→ Fernerzeugung des Objekts, `x` ist Stub!

„Öffentliche Objekte“: `x x = Remote.lookup("coolServer");`
mit entsprechender Infrastruktur hinter `Remote`.
Auch hier wird ein Stub `x` geliefert.

7.3 Mängel bei der Verteilungsabstraktion

... können an verschiedenen Stellen sichtbar werden:

- Parametermechanismen
- Geflechte als Parameter
- Objektidentität
- Verteilte Speicherbereinigung
- Zusammenbruch von Netz und/oder Rechnern
- unelegante Umsetzung des Fernaufruf-Prinzips

7.3.1 Parameterübergabe

Parametermechanismen bei Werten ohne Verweise:

- **Wertparameter** (*call by value*):
werden in Anfragenachricht übertragen, unproblematisch
- **Ergebnisparameter** (*call by result*) und Ergebniswert:
werden in Antwortnachricht übertragen, unproblematisch
- **Wert/Ergebnisparameter** (*call by value/result*):
entsprechend wie oben, unproblematisch

... und wenn Adressen im Spiel sind:

- Variablenparameter (*call by reference*):
problematisch, weil es keinen Sinn macht, die Adresse des aktuellen Parameters in den Adressraum des Aufgerufenen zu übergeben. Was tun?
- *Ähnliche Problematik bei Parametern mit Verweistypen oder Typen, die Verweise enthalten !*

2 Ansätze zum Umgang mit solchen Adressen:

- ① Anstelle einer lokalen Adressen wird ein **Fernverweis** übergeben, der vom Empfänger für einen **Fernzugriff** benutzt werden kann. Diese Technik bleibt gewöhnlich auf Objektverweise beschränkt: der Empfänger kann das Objekt mit einem **Fernaufruf** („callback“) ansprechen !
- ② Beim Packen werden die hinter der Adresse stehenden Werte mit verpackt, und die Struktur des Geflechts wird in der Nachricht nachgebildet. Das bedeutet, dass der Empfänger auf einer **tiefen Kopie** des Geflechts statt auf dem Original arbeitet. Bei Variablenparametern bedeutet es, dass sie wie Wert/Ergebnisparameter behandelt werden !

② Kopie statt Original bearbeiten

bedeutet tückisch veränderte Semantik von
Fernaufufen gegenüber lokalen Aufrufen !
→ **eingeschränkte** Verteilungsabstraktion !

Beispiele (Pseudo-Java):

- Utilities.removeDuplicates(myArray);
der gewünschte Effekt **bleibt aus!**
- printer.write("hello!");
kein Problem - Argument ist Konstante („immutable object“)!
- object.method(node1, node2);
fatal, wenn node1, node2 im gleichen Graphen liegen

Beispiele mit Variablenparametern:

Wert/Ergebnisübergabe ist hier kein Problem.

- ```
➤ void incr(bool cond, ref int x, ref int y) {
 if(cond) x++; else y++; } (C#)
```

Aufruf `incr(cond, ref a[i], ref a[k]);`?

Jedenfalls sollte eines der Elemente des Feldes  $a$  erhöht sein, auch im Fall  $i == k$ .

Aber: bei Fernaufruf **kein Effekt** falls cond && i==k !

Bemerkung zu Ada:

Ada hat drei Parametermechanismen:

**in** Wertparameter

**out** Ergebnisparameter

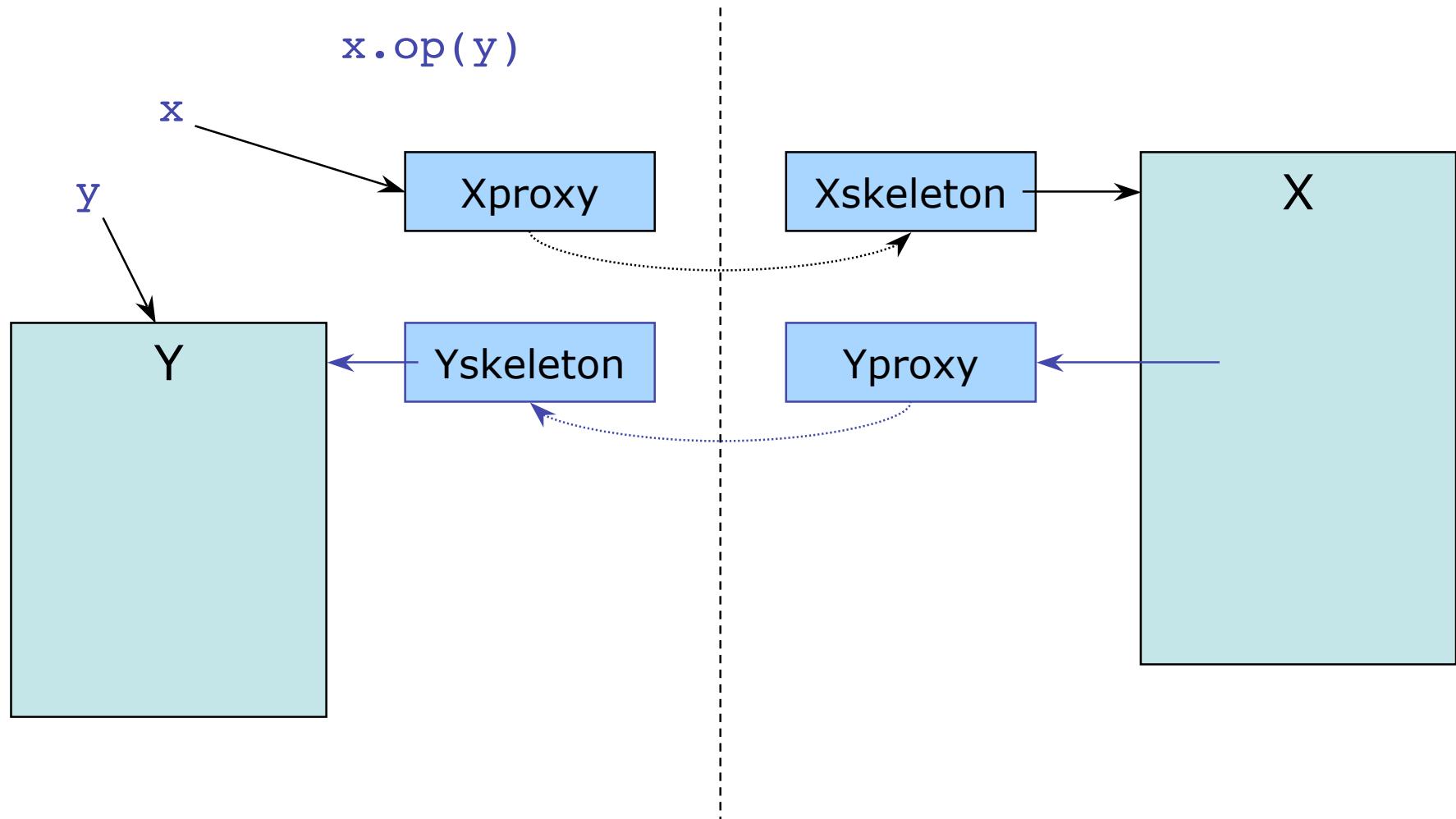
**in out** Wert/Ergebnisparameter *oder* Variablenparameter.  
Die Sprachdefinition lässt offen, wie **in out** implementiert wird! „*The execution of a program is erroneous if its effect depends on which mechanism is selected by the implementation.*“

## Fernverweise (*remote references*) anstelle von Kopien:

**Fernverweis** = Initialisierungsdaten eines Vertreters  
(oder direkt: Fernverweis = Vertreter)

**Prinzip:** Ein Verweis in einem Parameter wird vom Sender „im Flug“ in einen Fernverweis und beim Empfänger in einen Vertreter umgewandelt; der Vertreter ist auf einen ebenfalls vom Sender erzeugten Treiber bezogen.

Über den Vertreter kann das Objekt seinerseits per Fernaufruf angesprochen werden („callback“).  
(Voraussetzung: Code der Stubs ist vor Ort verfügbar)



## Achtung:

Diese Parameterbehandlung kommt natürlich *nicht in Frage* für Objekte, die wegen fehlender Methoden gar nicht aufrufbar sind.

## Beispiele:

- `class Complex { double re; double im; }`  
`void method(Complex c) { ... c.re ... }`  
(„Fernzugriff“ auf Variable wäre prohibitiv teuer!)
  
  - `int max(int[] myArray) { ... myArray[i] ... }`  
... entsprechend !
- hier keine Alternative zur Übergabe einer Kopie !

### 7.3.2 Verteilungsbedingte Fehler

Beteiligte Prozesse oder Systeme können abstürzen,  
Netze können ausfallen.

- Durchführung eines Fernauftrufs kann scheitern.
- Somit müssen **verteilungsspezifische Ausnahmen** berücksichtigt werden.
- wiederum **eingeschränkte** Verteilungsabstraktion

Im zentralisierten Fall läuft das Programm entweder - oder es stürzt ab. Im verteilten Fall womöglich **partieller Defekt!**

Wenn die Fernausführung einer Methode nicht ordnungsgemäß abgeschlossen wird, kann das im wesentlichen 3 Gründe haben:

- Der Fernaufruf ist nicht zum Aufgerufenen durchgedrungen.
- Der Aufgerufene ist während der Ausführung zusammengebrochen.
- Die Antwort ist nicht zum Aufrufer durchgedrungen.  
(● Die Ausführung dauert noch an ... )
  - Verschiedene Fernaufrufsysteme gehen unterschiedlich mit diesen Situationen um - durchaus auch anwendungsspezifisch:

## ① Höchstens-einmal-Semantik (*at-most-once semantics*):

Die Ausführung erfolgt vollständig oder gar nicht  
(mit transaktionaler Sicherung).

Neustart des Aufgerufenen nach Ausfall.

Nach Zeitüberschreitung wird der Fernaufruf wiederholt.

Buchführung über Aufrufe verhindert mehrfache Ausführung.

Eventuell genügt die Neuübertragung des schon von einer  
vorigen Ausführung vorliegenden Ergebnisses.

## ② Mindestens-einmal-Semantik (*at-least-once semantics*):

Ausführung erfolgt mindestens einmal, eventuell mehrmals.

Nach Zeitüberschreitung wird der Fernaufruf und die Ausführung wiederholt. Keine Buchführung nötig, daher einfach und zuverlässig zu implementieren.

Neustart des Aufgerufenen nach Ausfall.

Dies ist nur vertretbar bei *idempotenten Operationen* - bei denen wiederholte Ausführungen den gleichen Effekt habe wie eine einfache Ausführung, z.B. Schreiben in ein Objekt oder Lesen (Beispiel: File Server des Sun NFS).

### ③ Genau-einmal-Semantik (*exactly-once semantics*):

... hätte man gern!

Wird erreicht durch Kombination der Techniken für mindestens-einmal und höchstens-einmal.

Ist natürlich nur dann erreichbar, wenn der Aufgerufene *nicht dauerhaft* ausfällt.

Und auch noch zu beachten: *Ausfall des Aufrufers* hinterlässt womöglich verwaiste Ausführungen (→ *orphan detection*).

## Zusammenfassung

- Fernaufruf:  
*bottom-up-Sicht:*  
prozedurale Einkleidung synchroner Dienst-Kommunikation  
*top-down-Sicht:*  
Aufruf in anderen Adressraum, mit Nachrichten realisiert
- Implementierung mittels *client/server stubs*
- Formale Schnittstellenbeschreibung ermöglicht automatische Code-Erzeugung für Stubs
- Gute Verteilungsabstraktion ist schwer zu realisieren  
(Parametermechanismen, Verteilungsfehler, ....)

## Quellen

A.D. Birrell, B. J. Nelson: *Implementing remote procedure calls.*  
ACM TCS 2.1, February 1984  
<http://dl.acm.org/citation.cfm?id=357392>

F. Panzieri, S.K. Shrivastava: *Rajdoot: a remote procedure call mechanism supporting orphan detection and killing.*  
IEEE 14.1, January 1988

Sun ONC RPC: [www.ietf.org/rfc/rfc1057.txt](http://www.ietf.org/rfc/rfc1057.txt)

DCE: [www.opengroup.org/dce](http://www.opengroup.org/dce)

... und weiterführende Literatur auf  
[www.inf.fu-berlin.de/lehre/WS12/alp5/literatur.html](http://www.inf.fu-berlin.de/lehre/WS12/alp5/literatur.html)