AI-B.41: Verteilte Systeme

- Lecture Notes [SL] -

VIII. Interprozesskommunikation [II]

C. Schmidt | SG AI | FB 4 | HTW Berlin

Stand: WiSe 18
Urheberin: Prof. Dr. Christin Schmidt

Roadmap (Stand: 9 II 2018)

	SL
1	Organisatorisches, Einführung
2	Systemmodelle & Architekturstile
3	Systemarchitekturen I: C/S
4	Systemarchitekturen II: P2P
5	Cloud Computing
6	Netzwerke
7	Interprozesskommunikation I: Socket-Primitives
 8	Interprozesskommunikation II: Middleware (RPC/RMI)
9	Synchronisation und Koordination I: Concurrency
10	Synchronisation und Koordination II: Time
11	Sicherheit
12	Globale Zustände & verteilte Algorithmen
13	Ausgewählte Themen: Dienste
14	Ausgewählte Themen: tbd. (z.B.: Slot f. Gastvortrag / Exkursion)
15	Prüfungsvorbereitung/Klausurtipps

Rückblick letzte Vorlesung

Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- Aspekte und Eigenschaften der Interprozesskommunikation
- Bedeutung, Eigenschaften und Charakteristika von Sockets als logische Assoziation (Endpunkt für die Kommunikation) zwischen Prozessen
- Ports als Prozessen zugeordnete und an Sockets gebundene Nachrichtenziele innerhalb eines Hosts/Computers sowie damit verbundene Adressierungsaspekte
- Socket-Primitive bei:
  - » Verbindungsorientierter Kommunikation über TCP sowie die Besonderheit asymmetrischer Initialisierung
  - » Verbindungsloser Kommunikation über UDP
- Abstrakte Kommunikationsmodelle bei der Interprozesskommunikation und die Kriterien Adressierung, Blockierung, Pufferung, Kommunikationsform
- Ausgewählte Beispiele

✓ Studierende kennen grundlegende Aspekte der Interprozesskommunikation auf Transportebene, damit verbundene Technologien als Basis zur praktischen Implementierung einfacher verteilter Systeme und Grundlage sowie Bestandteil weiterführender Middleware-Konzepte (z.B. RPC, vgl. Lecture Notes „IPC II“)

Ausgangspunkt

vgl. Weber [1998:76]

- Kommunikation zwischen verteilten Prozessen musste ursprünglich über Ein-/Ausgabe-Primitive wie send und receive umständlich und aufwändig realisiert werden.
- Elementare Mechanismen und Programmierabstraktionen wie Sockets sind limitiert, Erweiterungen sind nötig:
  - » Kodierung/Dekodierung komplexer Datenstrukturen (Bäume, Listen)
  - » Konvertierung von Datenformaten, Compiler-Repräsentationen (Marshalling/Unmarshalling)
  - » Syntax und Semantik unterscheiden sich zur Verarbeitung im lokalen Fall (z.B. bei der Interaktion über Prozedur-/Methodenaufrufe)

- ✓ Wunsch / Notwendigkeit nach Möglichkeit zum Aufruf von entfernten Prozeduren / Funktionen mit Ähnlichkeit zum lokalen Aufruf mit höherer Transparenz (zur Ermöglichung von auftragsorientierter Kommunikation,)
- ✓ Lösungsansatz: Middleware-Kommunikationsformen und entfernte Aufrufe

Lernziele heute

Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

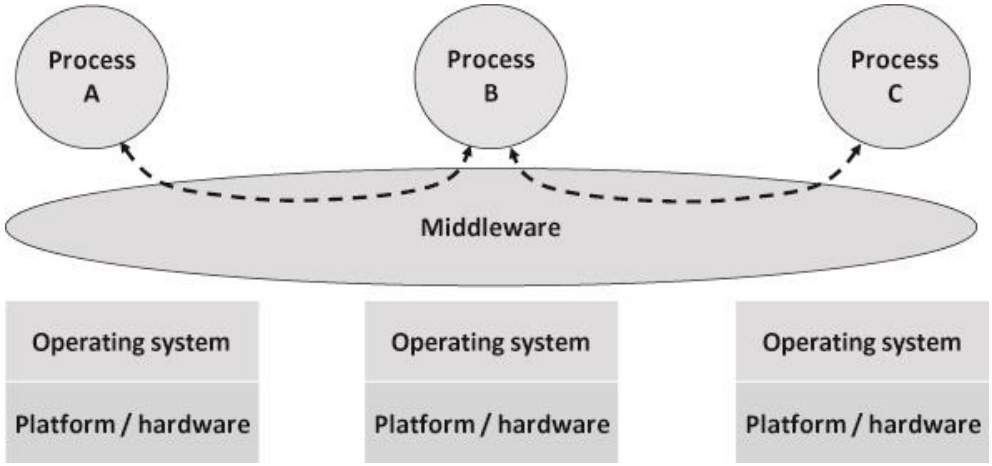
- Erweiterungen elementarer Kommunikationsmechanismen der Interprozesskommunikation auf Basis von Sockets (vgl. Lecture Notes IPC Teil I)
- Middleware als Softwareschicht zwischen Transport- und Anwendungsschicht zur Erhöhung von Transparenz und Überwindung von Interoperabilitätsproblemen aufgrund von Heterogenität
- Beispiele für Middleware-Kommunikationsformen sowie Prinzipien verteilter Aufrufe
- Die Bedeutung von Schnittstellen für Anwendungsentwickler_innen
- Ausgewählte Beispiele für höhere Konzepte bei der Interprozesskommunikation:
  - Remote Procedure Call (RPC: entfernte Prozeduraufrufe)
  - Remote Method Invocation (RMI: entfernter Methodenaufruf)
- Entwicklungsschritte verteilter Aufrufe am praktischen Beispiels einer einfachen RMI-Anwendung
- Serverzustände

✓ Studierende kennen grundlegende Aspekte der Interprozesskommunikation auf Middleware-Ebene, Eigenschaften damit verbundener ausgewählter Konzepte und Technologien sowie deren grundlegende Funktionsweise im Zusammenspiel beteiligter Komponenten. Zudem lernen Studierende praktische Beispiele für die Implementierung einfacher verteilter Systeme mittels entfernter Aufrufe und (entfernten) Schnittstellen kennen, welche als Grundlage sowie Bestandteil weiterführender Standards und Middleware-Kommunikationsformen dienen.

„Middleware“: Begriffseingrenzung

Beschreibungen / Definitionen aus der Fachliteratur (Auszug):

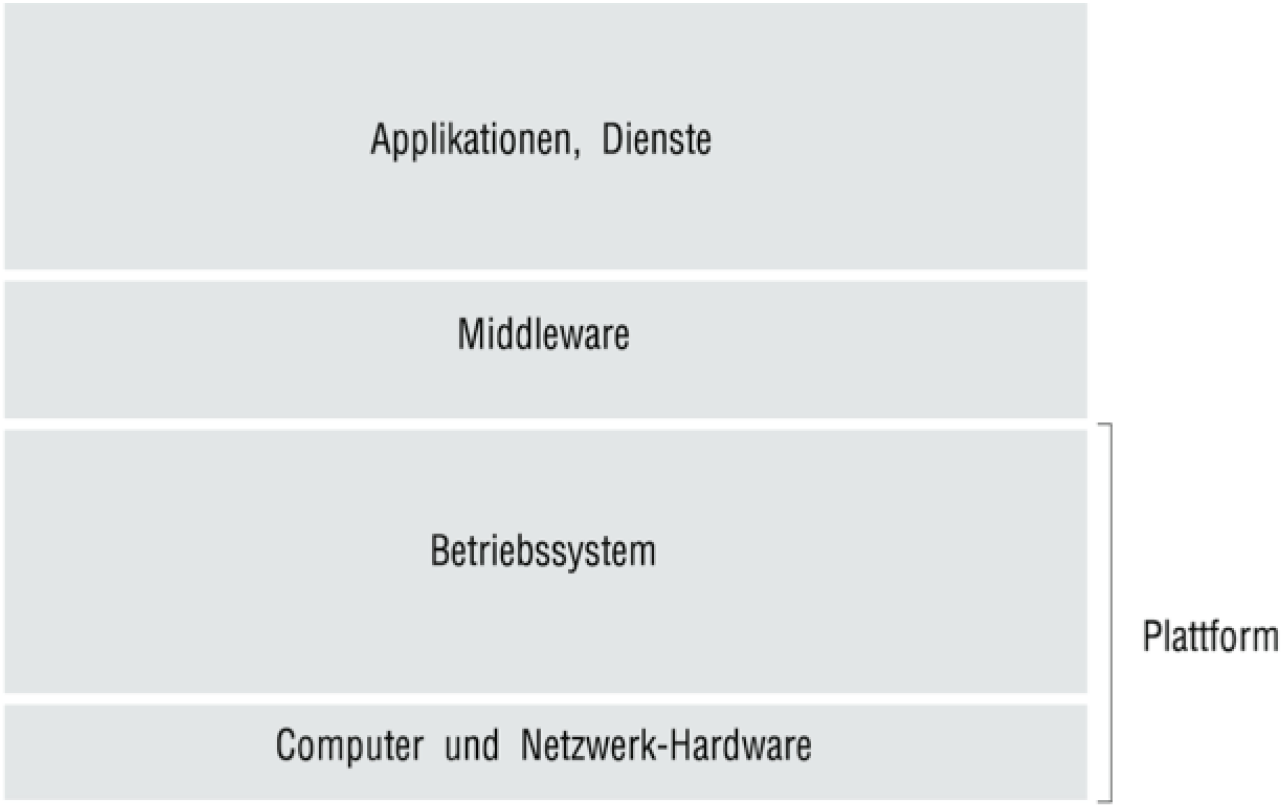
Definition	Quelle(n)
„Als Middleware bezeichnet man eine Anwendung, die logisch (hauptsächlich) in der Anwendungsschicht angesiedelt ist, aber viele Protokolle für allgemeine Zwecke enthält, die unabhängig von anderen, spezielleren Anwendungen ihre eigenen Schichten rechtfertigen.“	Tanenbaum & van Steen [2008:147]
“Software, die ein Programmermodell auf den grundlegenden Bausteinen der Prozesse und der Nachrichtenübergabe aufsetzt, wird als Middleware bezeichnet. Die Middleware-Schicht verwendet Protokolle, die auf Nachrichten zwischen Prozessen basieren, um ihre höher angesiedelten Abstraktionen zu realisieren wie beispielsweise entfernte Aufrufe und Ereignisse. […] Wichtige Aspekte der Middleware sind die Bereitstellung lokaler Transparenz sowie die Unabhängigkeit von den Details der Kommunikationsprotokolle, Betriebssysteme und der Computer-Hardware. Einige Formen der Middleware ermöglichen, dass einzelne Komponenten in unterschiedlichen Programmiersprachen geschrieben werden.”	Coulouris et al. [2002:204f.]
„Middleware is a collection of services, which sit between software components and the platforms they run on, to provide various forms of transparency to applications. Middleware usually provides location transparency by means of an in-built name service or subscription to an external service such as DNS.“	Anthony [2016:173]



Quelle: Anthony [2016:332]

Middleware [I]

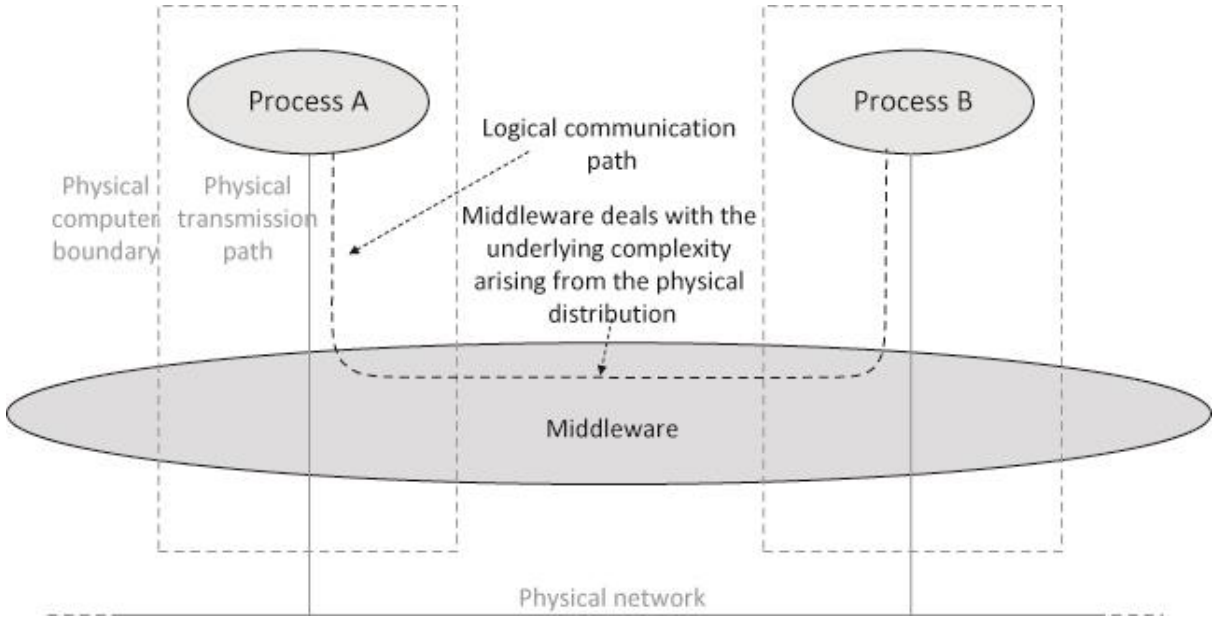
- Verteilungsplattform /Infrastruktur zur Programmierung und Nutzung verteilter Dienste
- (Standard-)Software, die zwischen Anwendungssoftware und Betriebssystem angesiedelt ist



Quelle: Coulouris et al. [2002:50]

Middleware [II]

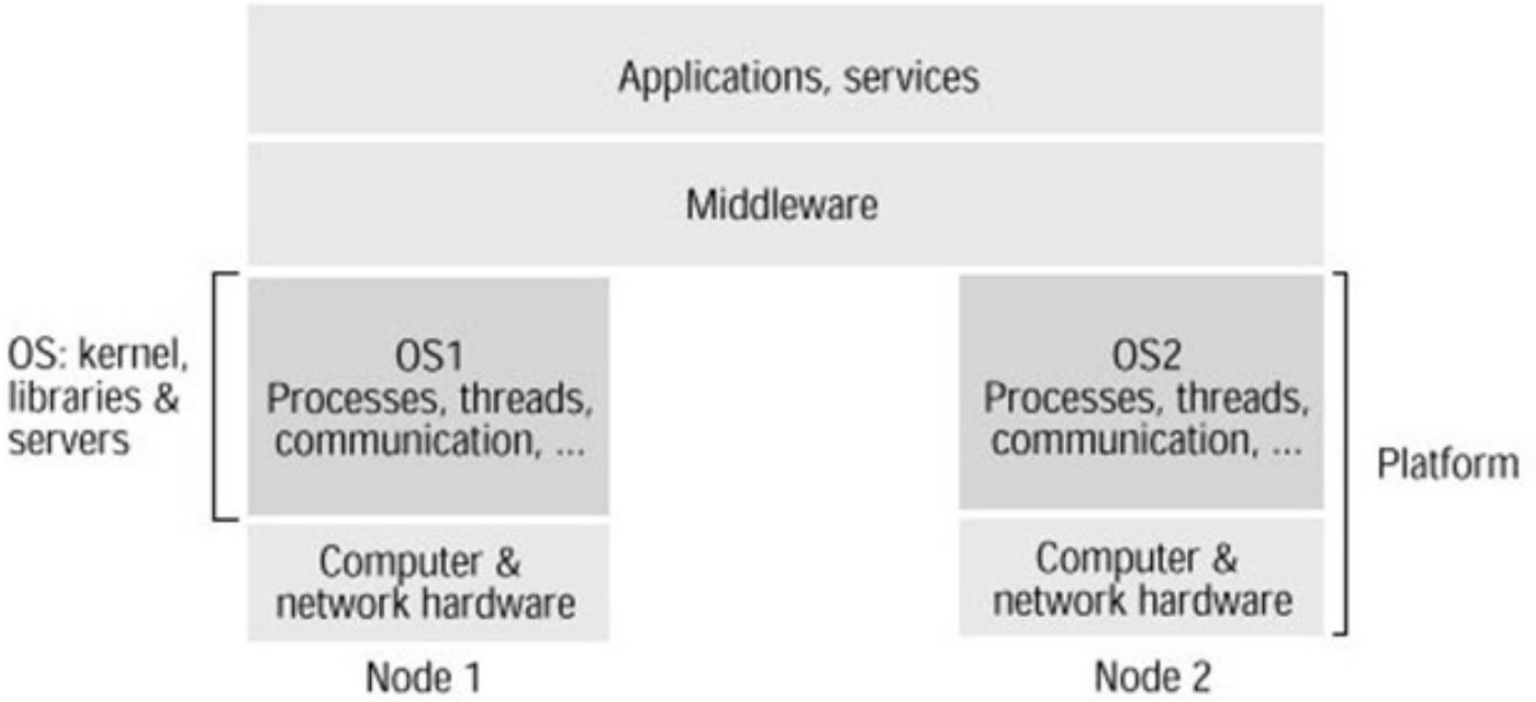
- Funktionen:
  - » Übertragung von Dienstaufrufen/Dienstantworten, d.h. Ermöglichung von Kommunikation/Nachrichtenaustausch zwischen: Anwendungen auf verschiedenen Computern, Komponenten einer verteilten Anwendung
  - » Führung von Verzeichnissen (Welcher Knoten implementiert einen spezifischen Dienst? Wie ist dieser aufzurufen?)



Quelle: Anthony [2016:452]

Middleware [III]

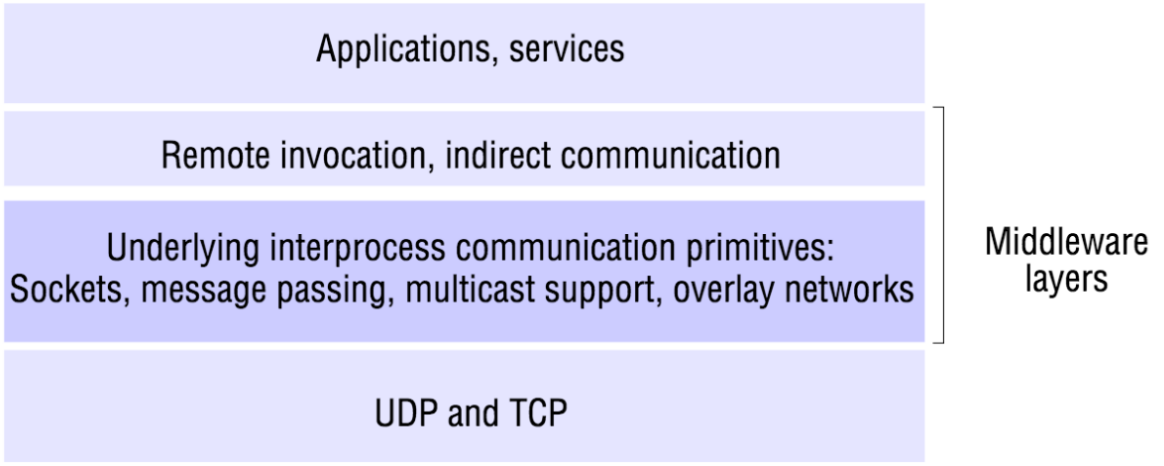
Ziel (I): Schaffung von Transparenz: Verbergung der zu Grunde liegenden heterogenen Netzwerke, Hardware, OS und Programmiersprachen



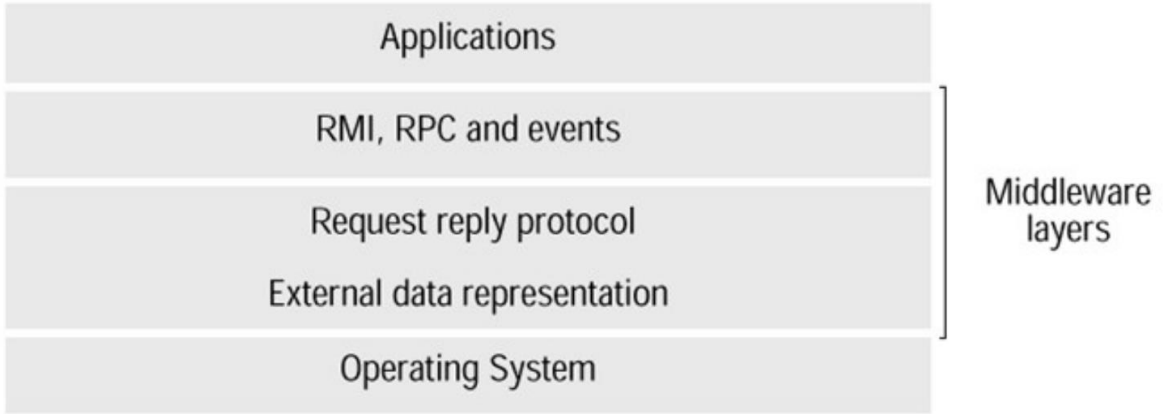
Quelle: Coulouris et al. [2002:252]

Middleware [IV]

Ziel (II): Ermöglichung von Offenheit: einheitliche (Programmier-)Schnittstellen für Komponenten/Entwickler



Quelle: Coulouris et al. [2012:186]

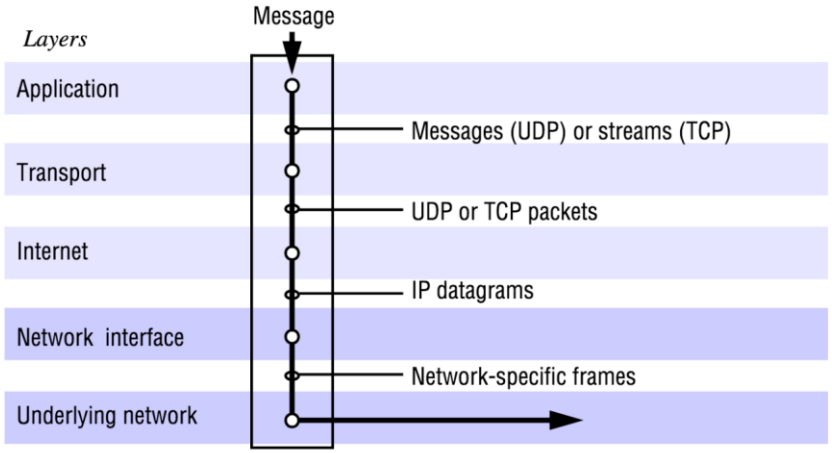


Quelle: Anthony [2016:332]

Middleware [V]

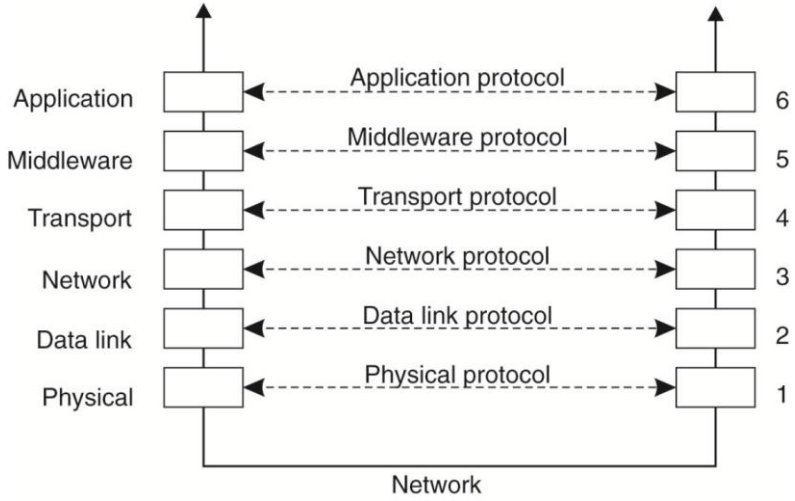
- Implementierung meist über IP (-> Überwindung der Netzwerkheterogenität)

Basis: TCP/IP-Stack als Schichtenkonzept (vgl. Lecture Notes Netzwerke, IPC I):



Quelle: Coulouris et al. [2012: 106]

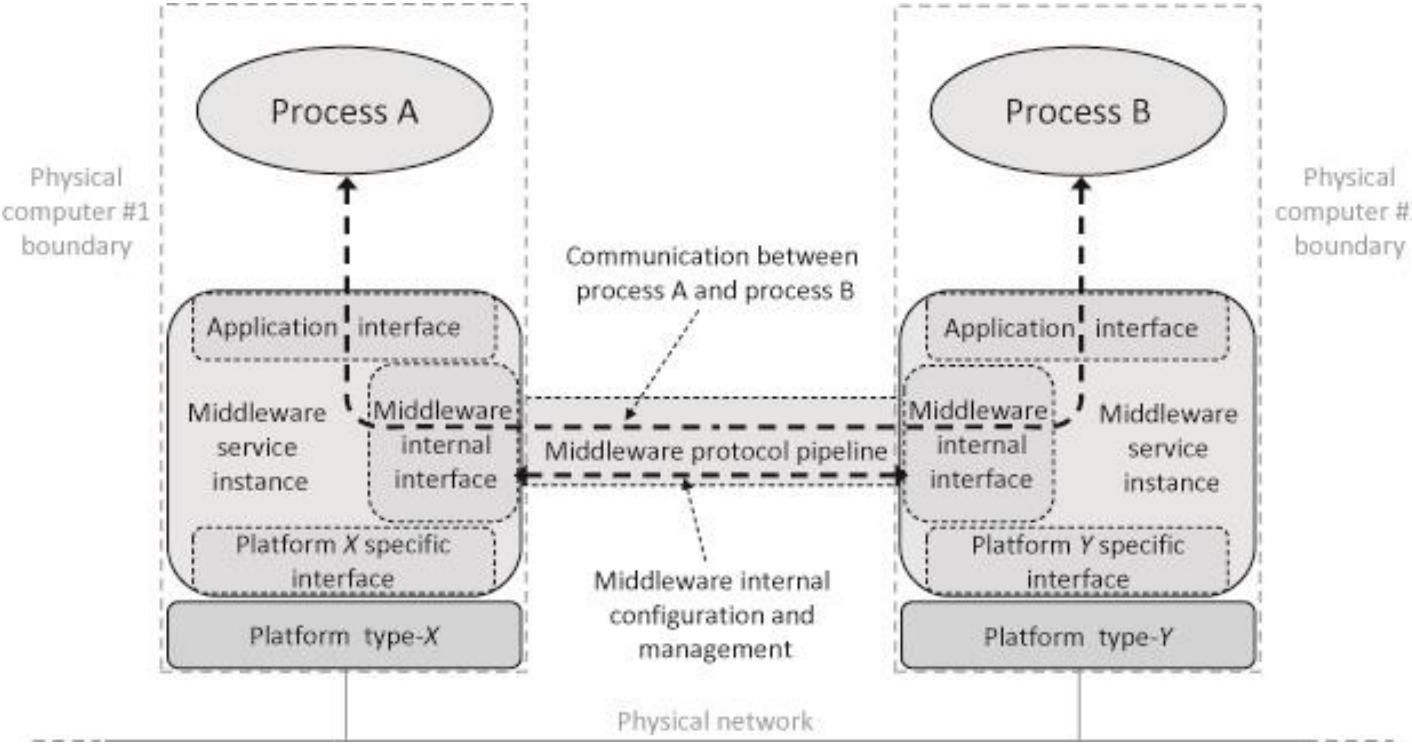
Middleware bildet eine neue virtuelle Schicht zwischen Transport- und Anwendungsebene:



Quelle: Tanenbaum & van Steen [2008:148]

Middleware [VI]

- Bedeutung von (entfernten) Schnittstellen (Interfaces): Middleware
  - » Setzt auf Programmierschnittstellen der Betriebssysteme einzelner Rechnerknoten auf
  - » Bietet Anwendungsprogrammen einheitliche Schnittstellen

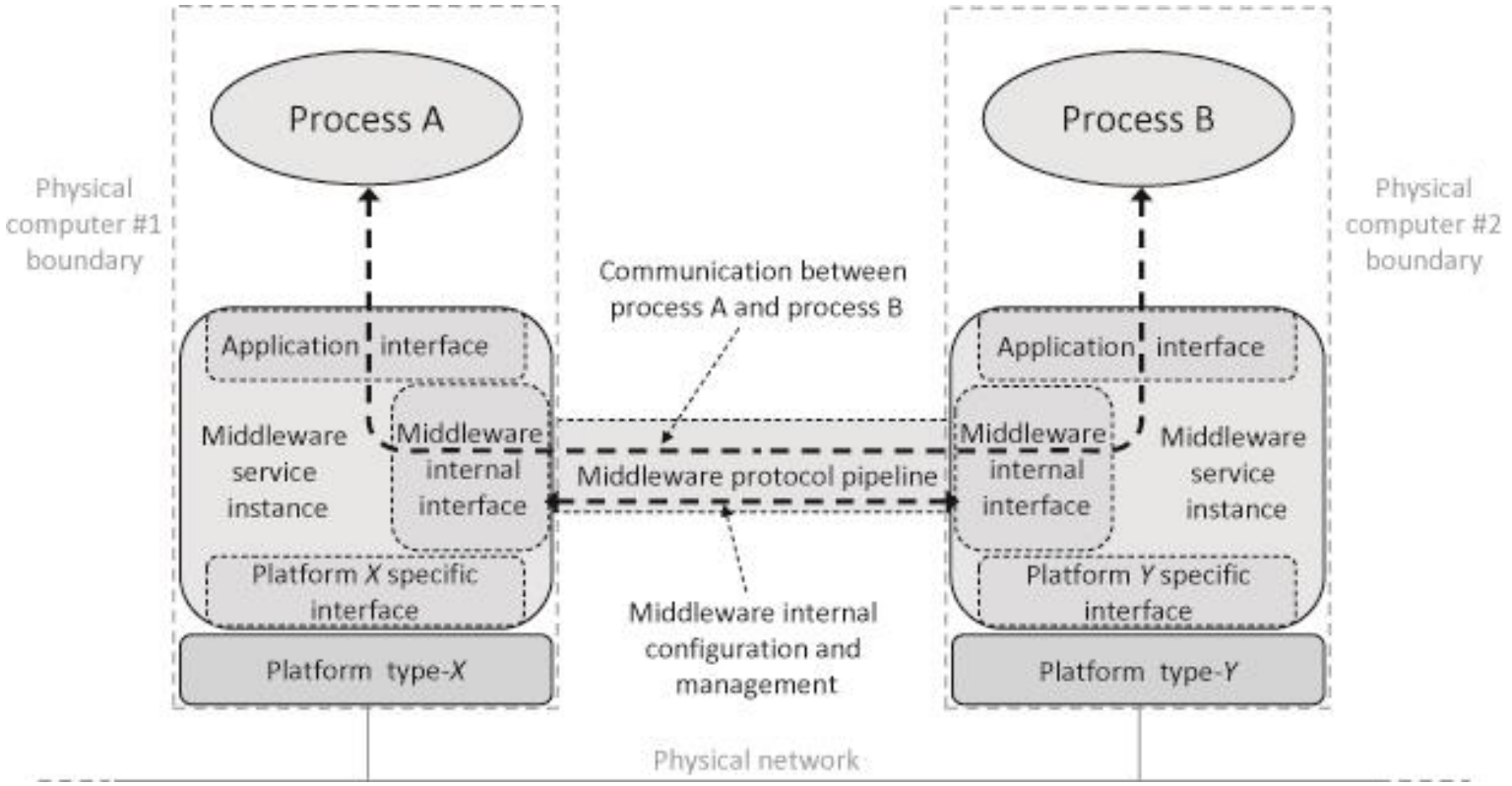


Quelle: Anthony [2016:453]

Schnittstelle (Interface) [I]

vgl. Coulouris et al. [2002:205]

- Gibt diejenigen (entfernten) Prozeduren und Variablen an, auf die andere Module zugreifen können.
- Enthält Definition der Typen der Argumente, Rückgabewerte und Ausnahmen
- Enthält nicht die Implementierung der Schnittstelle



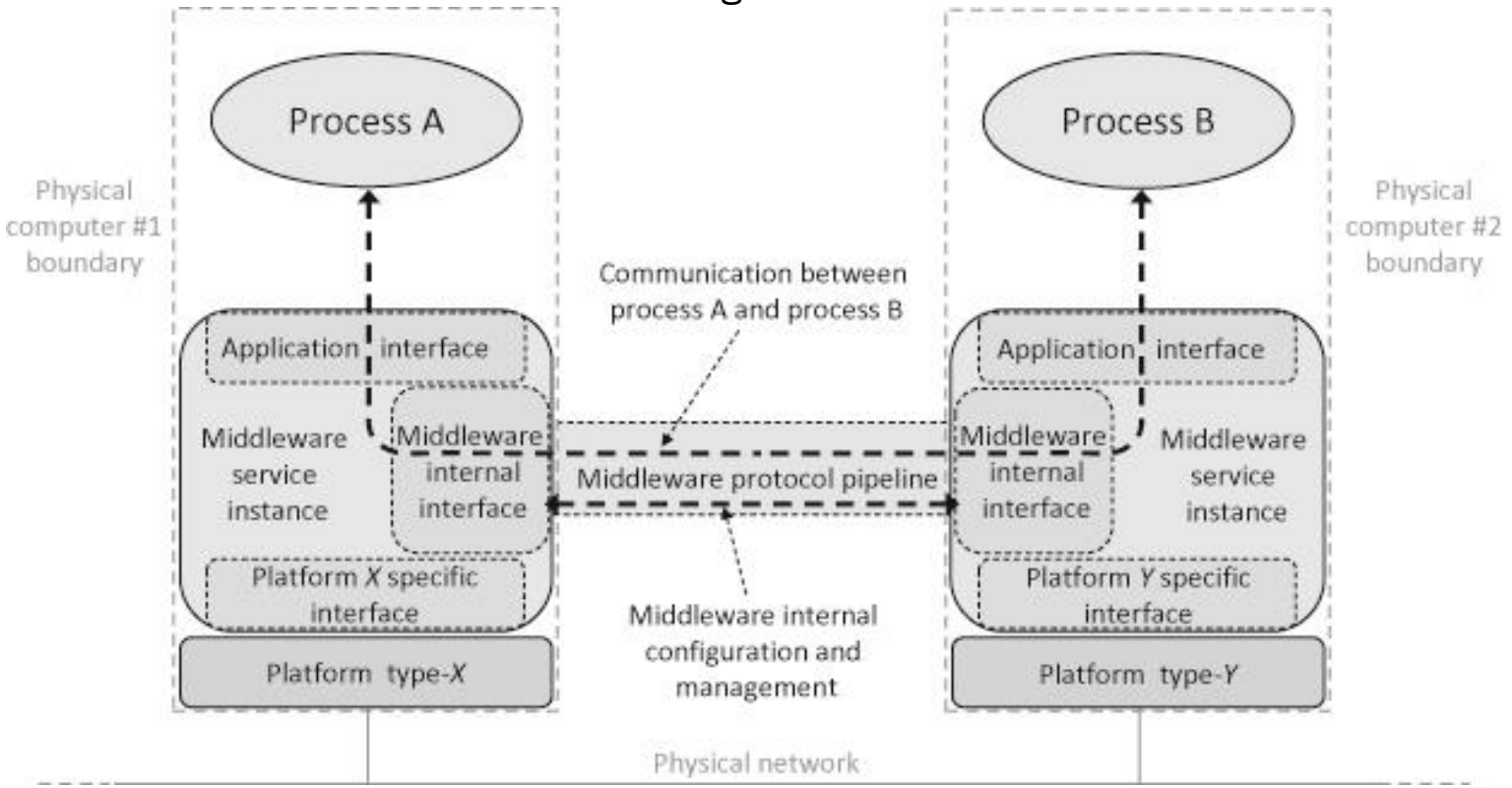
Quelle: Anthony [2016:453]

Schnittstelle (Interface) [II]

vgl. Coulouris et al. [2002:205]

Implementierungsaspekte für Anwendungsentwickler_innen:

- ✓ Module werden so implementiert, dass sie alle Informationen über sich verbergen, außer denen, die über ihre Schnittstelle offen gelegt werden.
- ✓ Solange die Schnittstelle gleich bleibt, kann die Implementierung geändert werden, ohne die Benutzer des Moduls zu beeinträchtigen



Quelle: Anthony [2016:453]

Schnittstelle (Interface) [III]

vgl. Coulouris et al. [2002:205]

- Schnittstellenarten:
  - » Dienstschnittstelle: RPC
  - » Entfernte Schnittstelle: RMI
- Implementierung von Modulen erfolgt mit Verbergung jeglicher Modulinformation (außer der in der Schnittstelle offengelegten Information)
- Bei gleicher Schnittstelle kann die Implementierung eines Moduls geändert werden
- Realisierung mittels Interface Definition Language (IDL)
- Da nicht alle VS in der gleichen Sprache geschrieben sind, werden Interface Definition Languages (IDLs) benötigt, um den entfernten Zugriff auf Programme/Objekte zu erlauben, die in unterschiedlichen Sprachen geschrieben/implementiert sind.
- IDLs stellen Notation für die Definition von Schnittstellen bereit (Beispiele vgl. Appendix)
  - » Jeder Parameter einer Methode kann für die Ein- und Ausgabe vorgesehen werden
  - » Jeder Parameter einer Methode enthält eine Typzuordnung

Middleware [VII]

- Ausprägungen von Programmierschnittstellen / Beispiele für Middleware-Kommunikationsdienste:
  - » Entfernter Prozeduraufruf (Remote Procedure Call (RPC), vgl. ff.)
    - Funktionen werden aufgerufen
    - Resultat wird erhalten
  - » Entfernter Methodenaufruf (Remote Method Invocation (RMI), vgl. ff.)
    - Methoden (Implementierungen von Operationen) werden aufgerufen
    - Methoden führen sich aus und verändern als Resultat den Objektzustand
  - » Warteschlangendienste
  - » Unterstützung für kontinuierliche Medien durch Streams
  - » Gruppenkommunikation / Multicasting
  - » Entfernte Ereignisbenachrichtigung,
  - » Entfernte SQL-Zugriff
- Beispiele für Middleware-Produkte und Standards:
  - » Distributed Computing Environment
  - » Common Object Request Broker Architecture (CORBA): mittels CORBA können Applikationen (infrastrukturelle) Dienste bereit gestellt werden: Namensdienste, Sicherheitsdienste, Transaktionsdienste, Persistente Speicherdienste, Ereignisbenachrichtigungsdienste

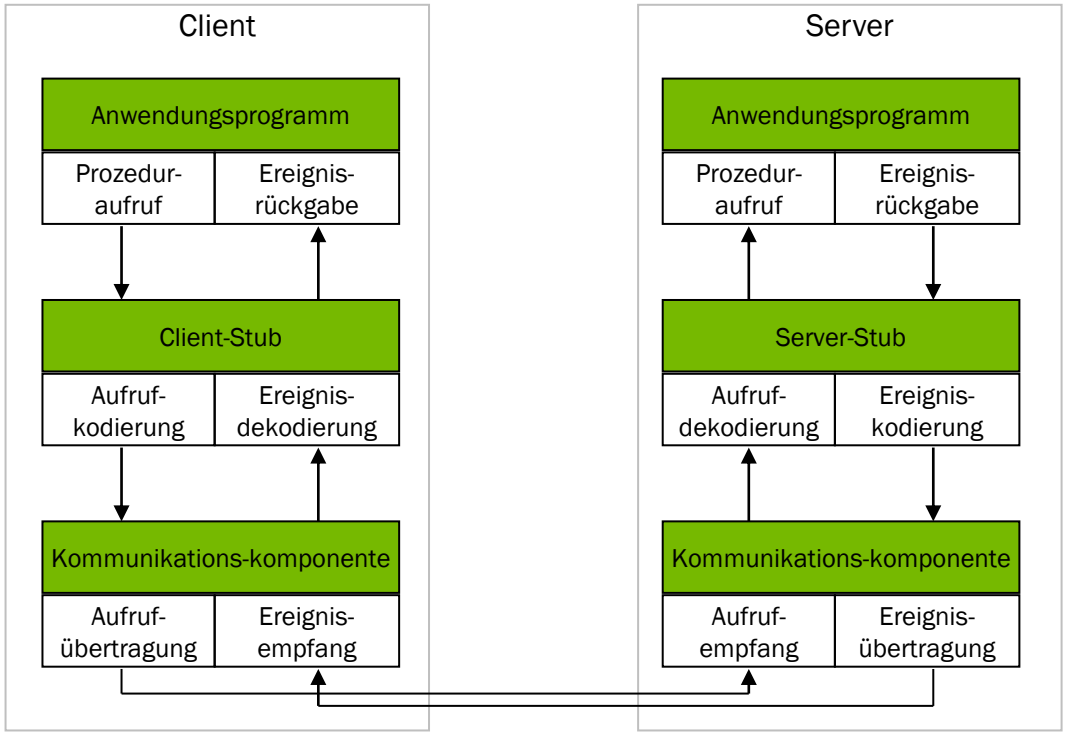
Remote Procedure Call (RPC) : Begriffseingrenzung

Beschreibungen / Definitionen aus der Fachliteratur (Auszug):

Definition	Quelle(n)
„Der Mechanismus des entfernten Prozeduraufrufs (engl. remote procedure call, RPC) wurde von Birrell and Nelson [BiNe84] erstmals veröffentlicht. Ein RPC-System erlaubt den Aufruf von Prozeduren auf anderen Rechnern, inklusive der Übergabe von Parametern, basierend auf den Mechanismen des synchronen entfernten Dienstaufrufs. Wie bei einer lokalen Prozedur blockiert der Sender bis der Aufruf komplett abgeschlossen ist. Ein Programmierer kann also nach dem gleichen Paradigma programmieren wie er es in einem lokalen Programm verwendet.“	Weber [1998:76]
„Ruft ein Prozess auf einer Maschine A eine Prozedur auf einer Maschine B auf, so wird der aufrufende Prozess suspendiert und die Abarbeitung der aufgerufenen Prozedur findet auf der Maschine B statt. Information vom Aufrufer zum Aufgerufenen kann über die Parameter transportiert werden und Information kann über das Ergebnis der Prozedur zurücktransportiert werden. Durch dieses Vorgehen zeigt ein Remote Procedure Call das vertraute Verhalten von lokalen Prozeduraufrufen.“	Bengel [2004:137]
„Zusammengefasst schlugen Birrell und Nelson vor, dass Programme die Möglichkeit erhalten, Prozeduren auf anderen Rechnern aufzurufen. Wenn ein Prozess auf Rechner A eine Prozedur auf Rechner B aufruft, wird der Prozess auf A angehalten und die Ausführung der Prozedur findet auf B statt. Die Parameter können Informationen vom Aufrufer zum Aufgerufenen übertragen und im Ergebnis der Prozedur zurückkommen. Der Programmierer sieht überhaupt keine Nachrichtenübergabe. Dieses Verfahren wird als entfernter Prozeduraufruf (Remote Procedure Call) oder oftmals einfach nur als RPC bezeichnet.“	Tanenbaum & van Steen [2008:147]
„[..] RPC-Modell [..] ermöglicht, dass Client-Programme Prozeduren in Server-Programmen aufrufen, die in separaten Prozessen ausgeführt werden und im Allgemeinen auch auf anderen Computern als dem Client sind.“	Coulouris et al. [2002:203]
„Remote Procedure Call (RPC) is an example of a higher-level mechanism built on top of the TCP. RPC involves making a call to a procedure that is in a different process space to that of the calling procedure. [...] Remote Procedure Call is an extension of local procedure call in which the called procedure is part of a different programm to the calling procedure, and thus, at run time, the two procedures are in two different process spaces.“	Anthony [2016:124]
“In particular, in RPC, procedures on remote machines can be called as if they are procedures in the local address space. The underlying RPC system then hides important aspects of distribution, including the encoding and decoding of parameters and results, the passing of messages and the preserving of the required semantics for the procedure call. This concept was first introduced by Birrell and Nelson [1984] and paved the way for many of the developments in distributed systems programming used today.”	Coulouris et al. [2012:195]

Remote Procedure Call (RPC) [I]

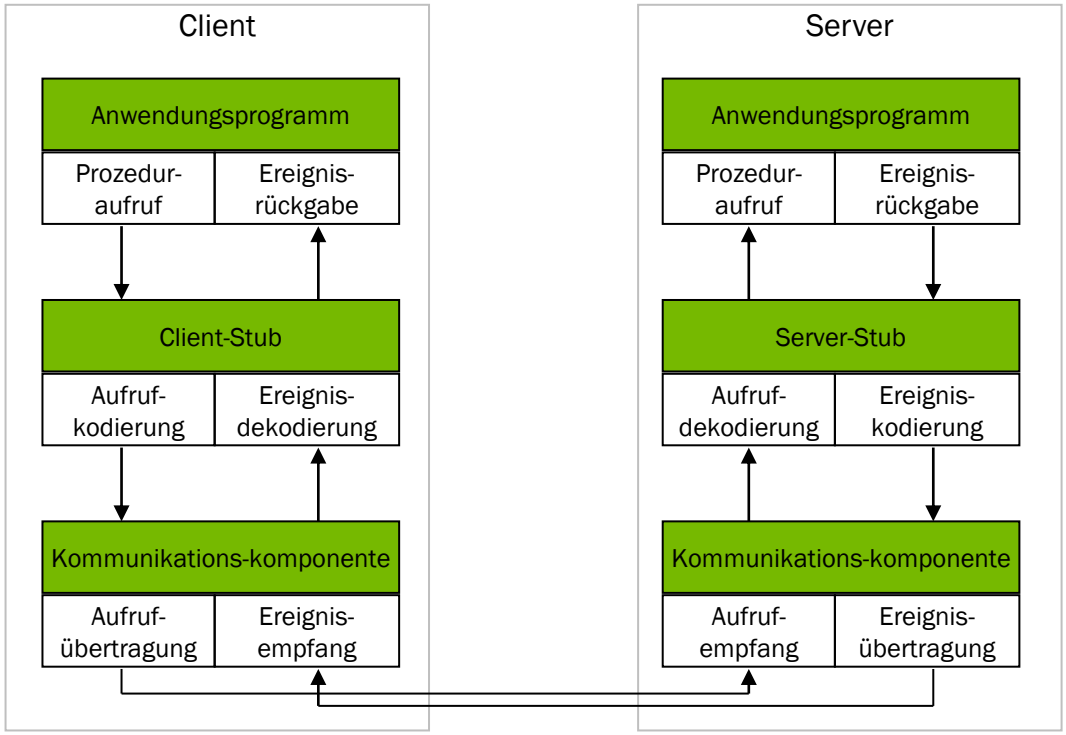
- Mechanismus, bei dem Anwendungsentwickler_innen für die Kommunikation zwischen Prozessen auf unterschiedlichen Rechnern dasselbe Paradigma benutzen wie beim lokalen Prozeduraufruf
- Programmieraufwand bezieht sich auf Spezifikation der Prozedurschnittstelle sowie Programmierung der Anwendung(en)



Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC) [II]

- ✓ Programmcode zur Kommunikation der Prozesse wird automatisch generiert
- ✓ **Synchrone Kommunikation:** Client blockiert / wartet nach Aufruf (request) auf den Erhalt eines Ergebnis‘ durch den Server (response)

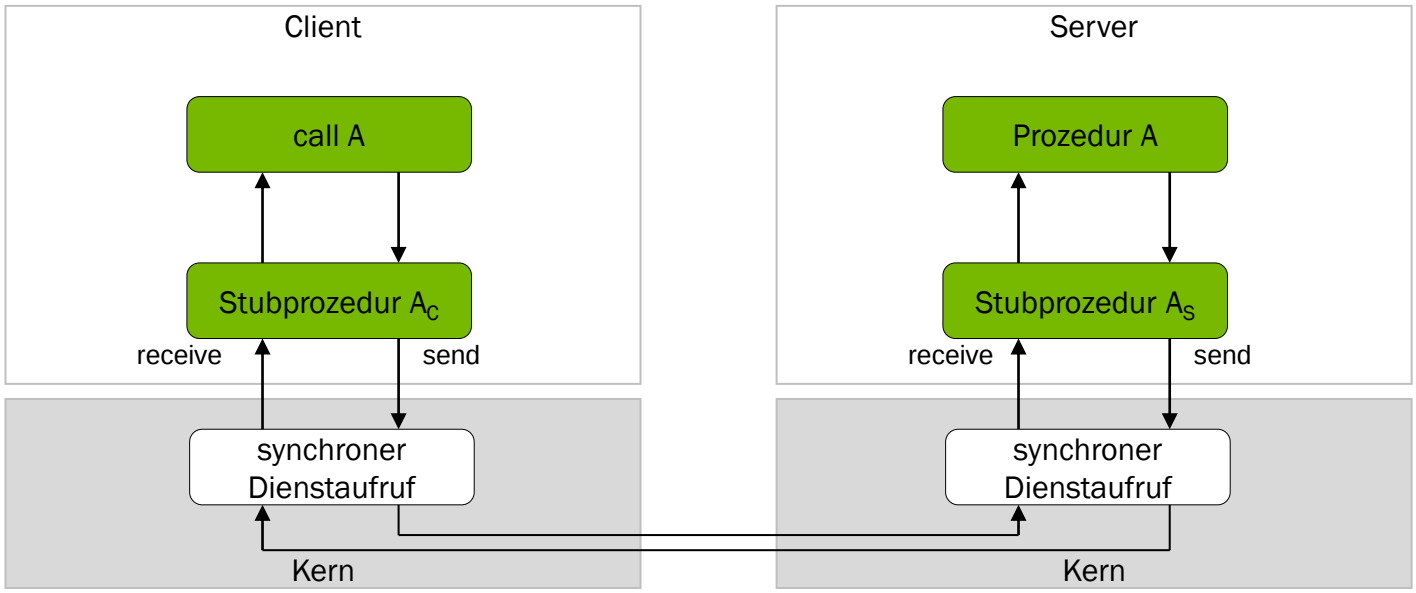


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Komponenten [I]

- Client: Beschreibung/Spezifikation der Schnittstelle (oft mittels Interface Definition Language (IDL))

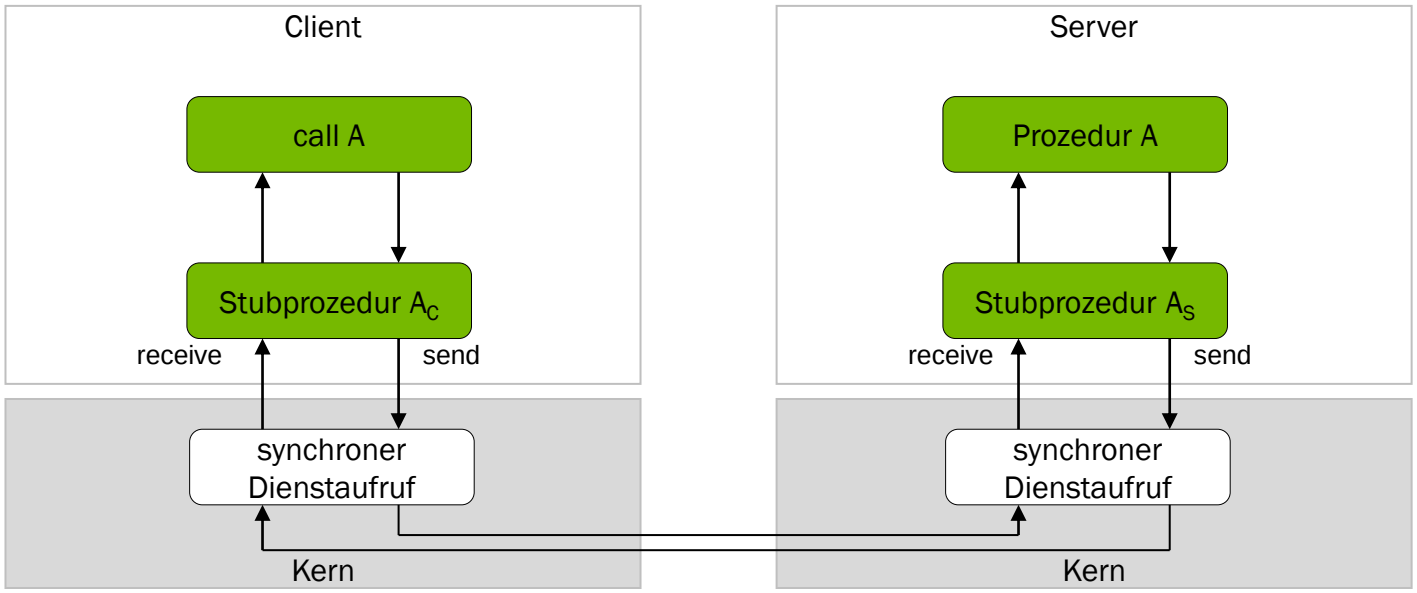
- Server:
  - » Beschreibung/Spezifikation der Schnittstelle (oft mittels Interface Definition Language (IDL))
  - » Implementierungen der aufzurufenden Prozeduren



Quelle: i.A. an Weber [1998:77]

Remote Procedure Call (RPC): Komponenten [II]

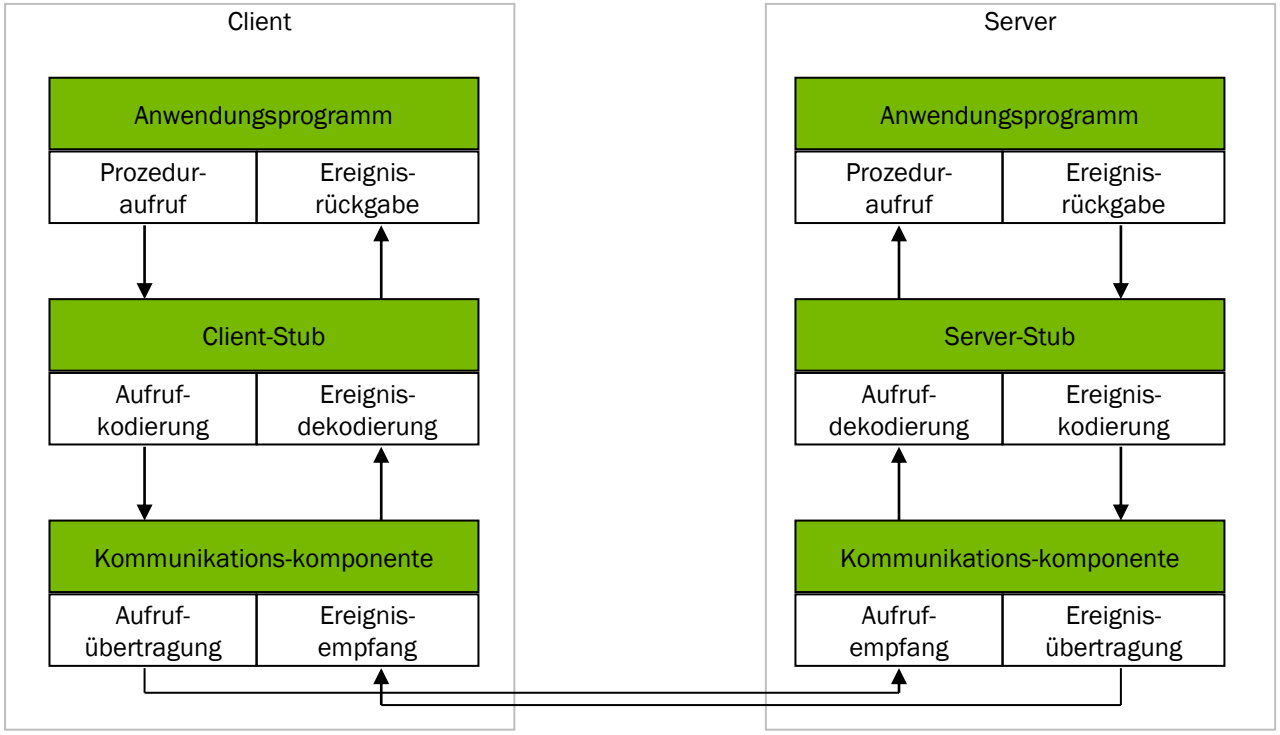
- „Stubs“: Proxies für Client und Server
  - Codemodule, welche alle notwendigen Funktionen zur Übermittlung lokaler Aufrufe an entfernten Prozess kapseln
  - Enthalten Konvertierungsroutinen (z.B. Marshalling / Unmarshalling)
  - Compiler generiert „Stubs“



Quelle: i.A. an Weber [1998:77]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [I]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

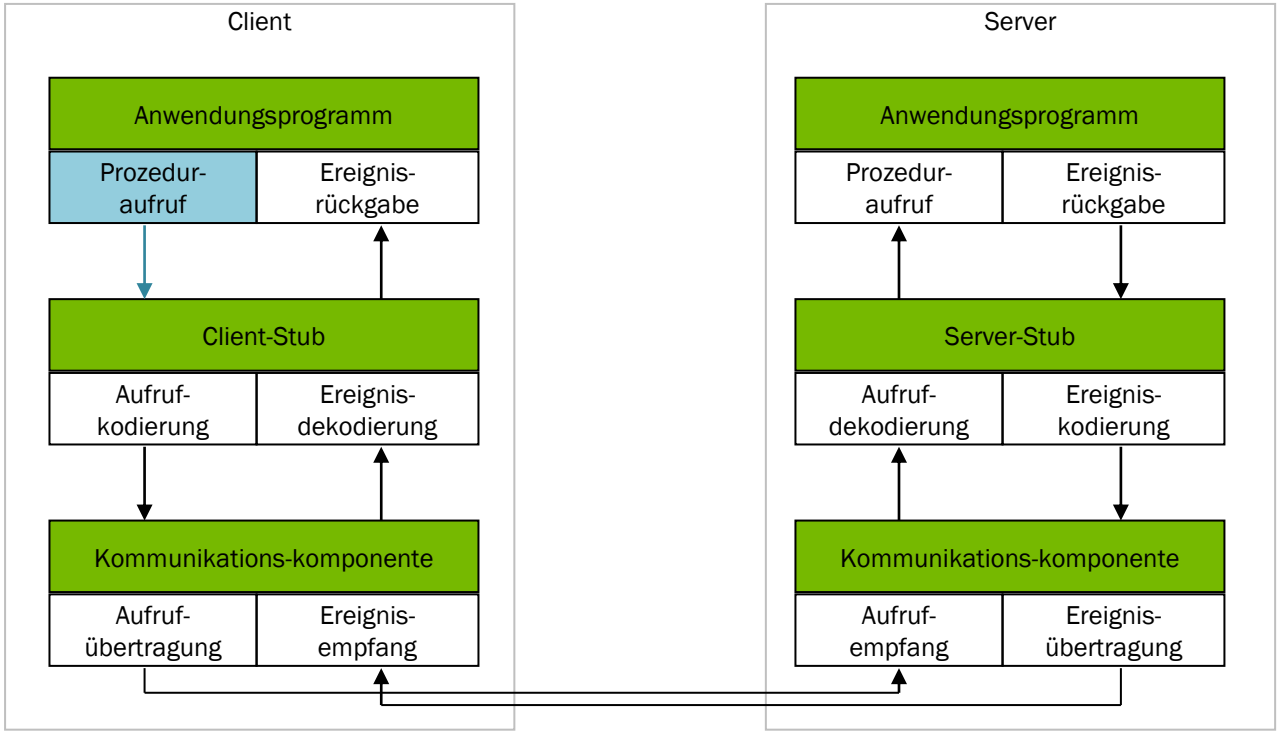


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [II]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

1. *Client* ruft eine spezifizierte Prozedur auf
- Implementierung der Prozedur steht nicht lokal, sondern auf dem Server zur Verfügung
- Gleichnamige Prozedur des Client-Stubs wird aufgerufen / aktiviert, welche lokal auf dem selben Rechner wie der Client läuft



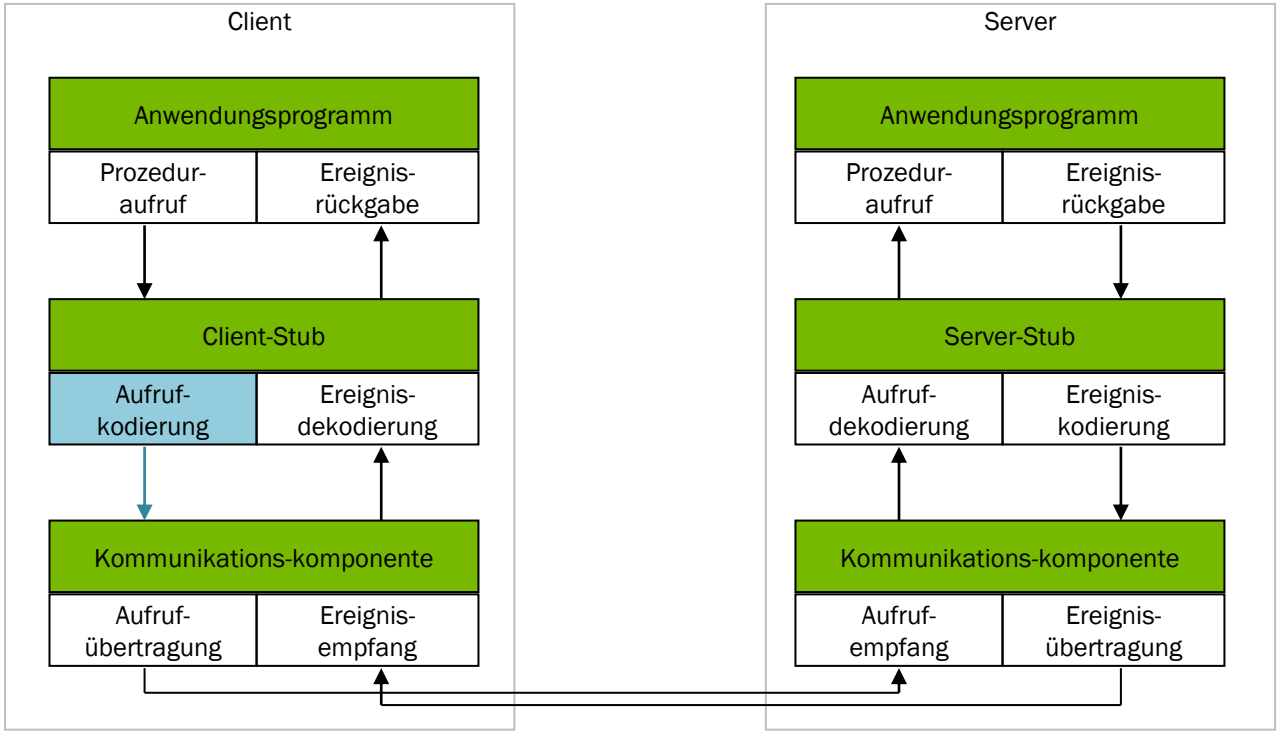
Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [III]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

2. *Client-Stub* stellt eine Nachricht zusammen und ruft das lokale Betriebssystem auf:

- Konvertiert Eingabeparameter („Marshalling“) in ein plattformunabhängiges Datenformat (z.B. eXternal Data Representation (XDR))
- Verschickt konvertierte Parameter mit weiteren Angaben über das Netz, z.B. Kennung, welche Prozedur aufgerufen wird, Information über Server, der Prozedur anbietet (liegt vor oder muss beschafft werden)
- Blockiert nach dem Senden, bis Nachricht zurück kommt

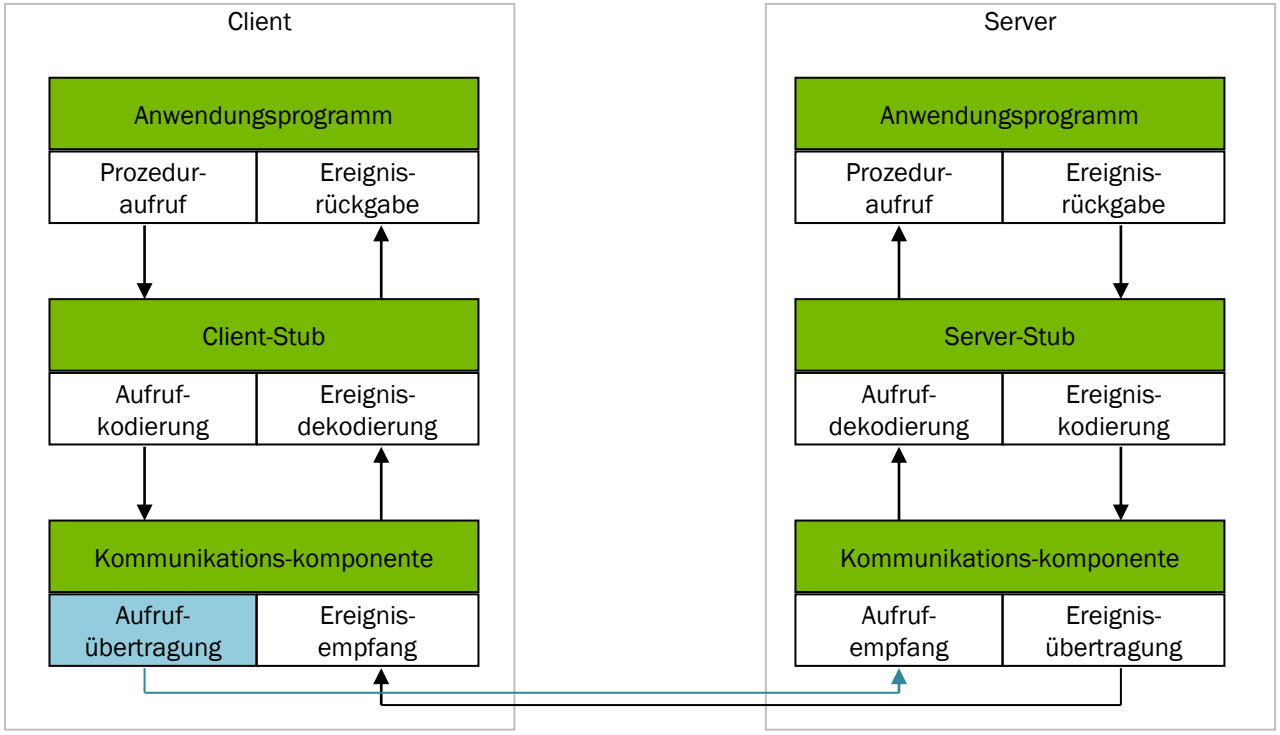


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [IV]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

3. Betriebssystem des Clients sendet die Nachricht an das entfernte Betriebssystem

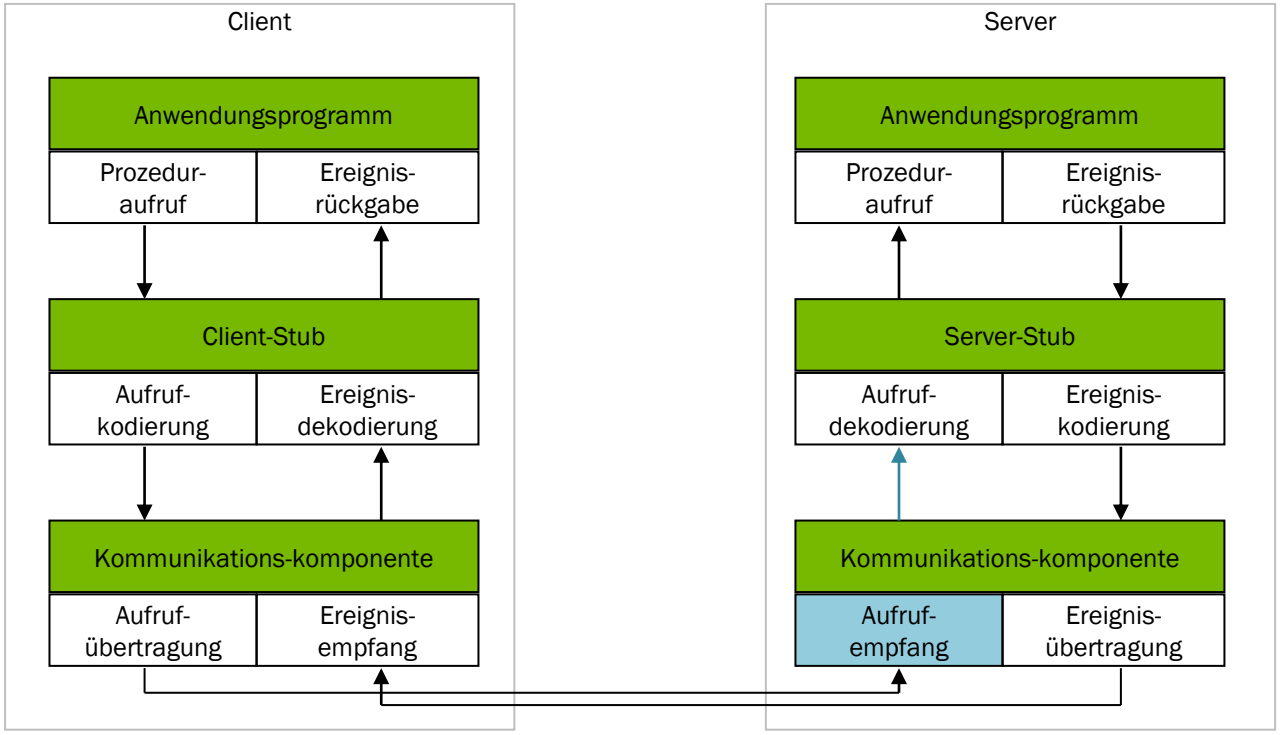


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [V]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

4. Das entfernte Betriebssystem übergibt die Nachricht an den Server-Stub

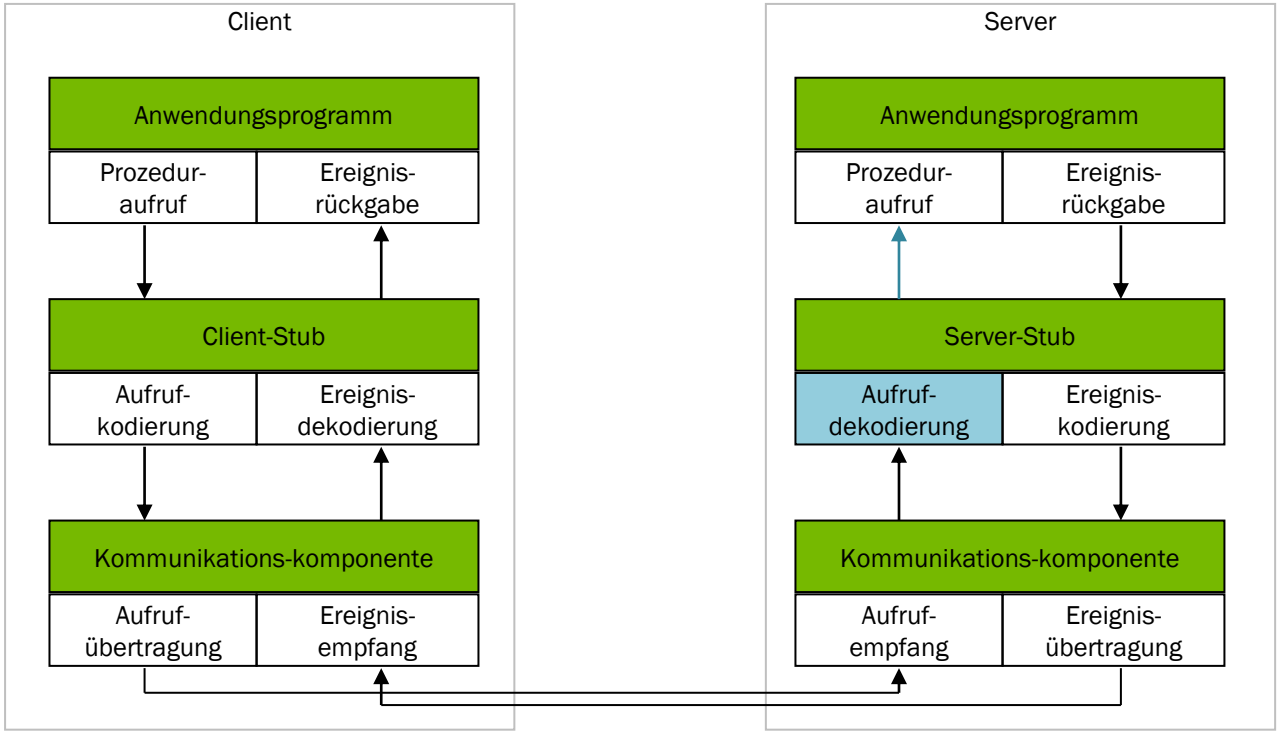


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [VI]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

5. *Server-Stub* entpackt die Parameter und ruft den Server auf
   - Nimmt Nachricht des Client-Stubs entgegen (typisch: Endlosschleife zum Warten auf Client-Anfragen)
   - Eingabeparameter werden in lokales Datenformat konvertiert („Unmarshalling“)
   - Aufruf der gewünschten Prozedur

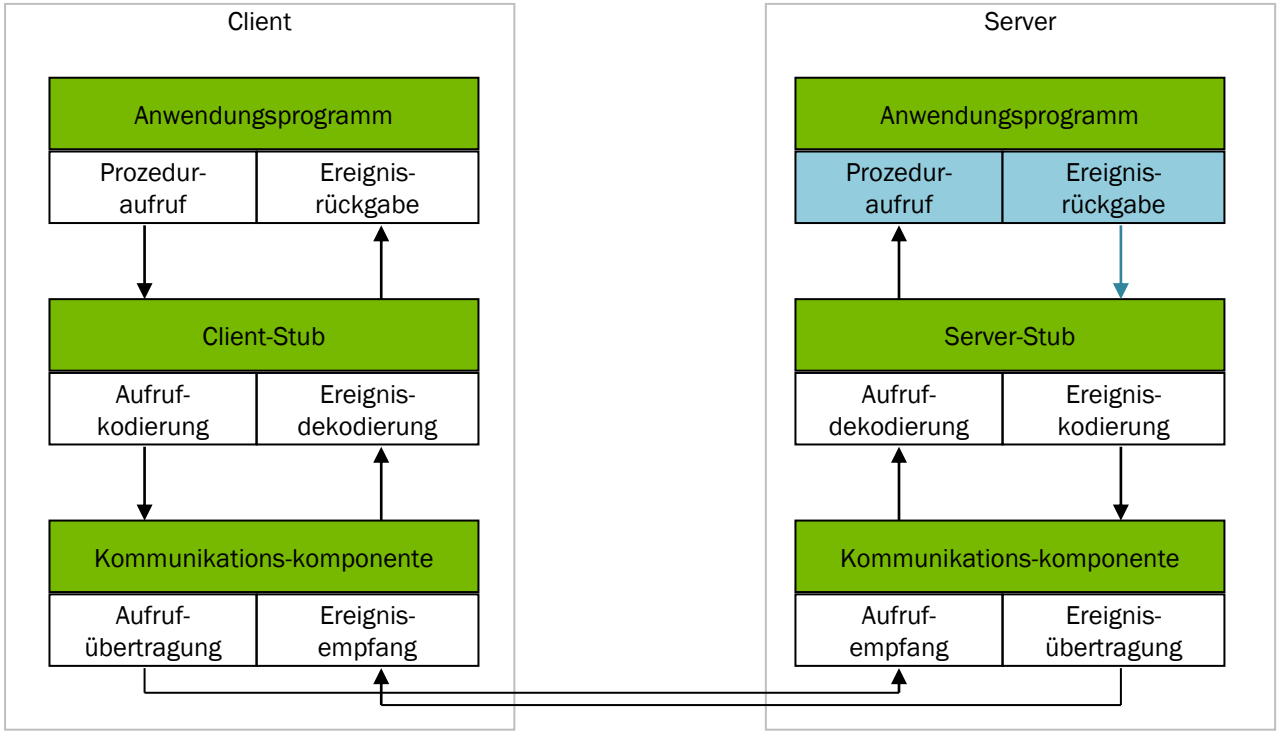


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [VII]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

6. *Server* verrichtet die Arbeit und gibt dem Stub das Ergebnis zurück
   - Führt Prozedur aus
   - Übergibt Ergebnis (Ausgabeparameter/Rückgabewert) an Server-Stub

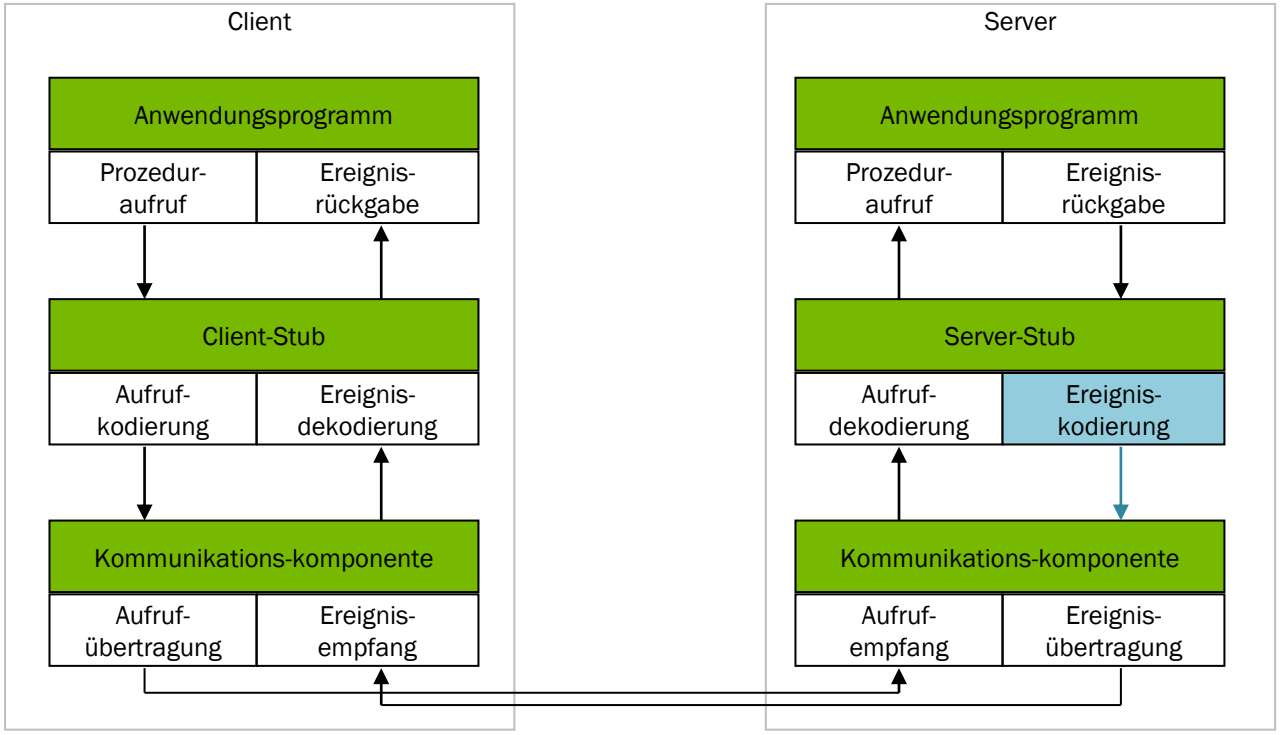


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [VIII]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

7. *Server-Stub* verpackt Ergebnis in eine Nachricht und ruft sein lokales Betriebssystem auf
   - Wandelt Ausgabeparameter/Rückgabewert in plattformunabhängiges Datenformat
   - Schickt Ergebnis über OS zurück an Client-Stub

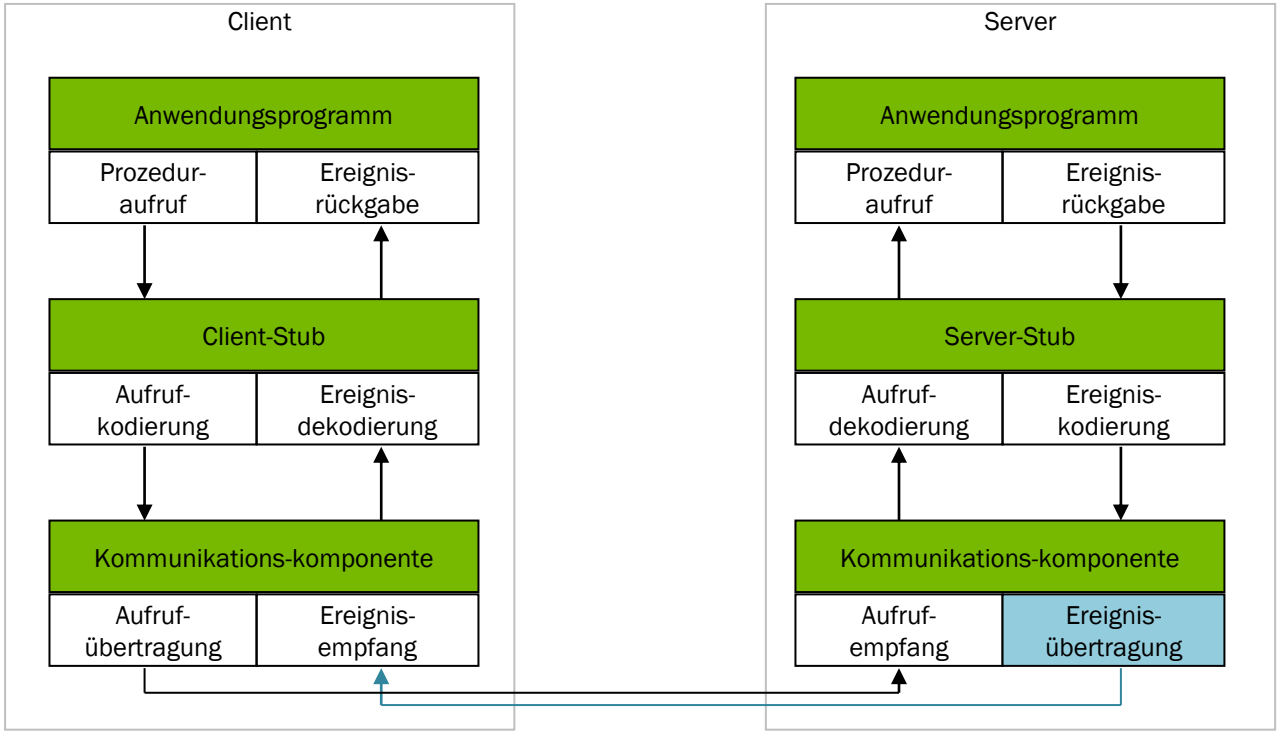


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [IX]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

8. Das Server- Betriebssystem sendet die Nachricht an das Betriebssystem des Clients

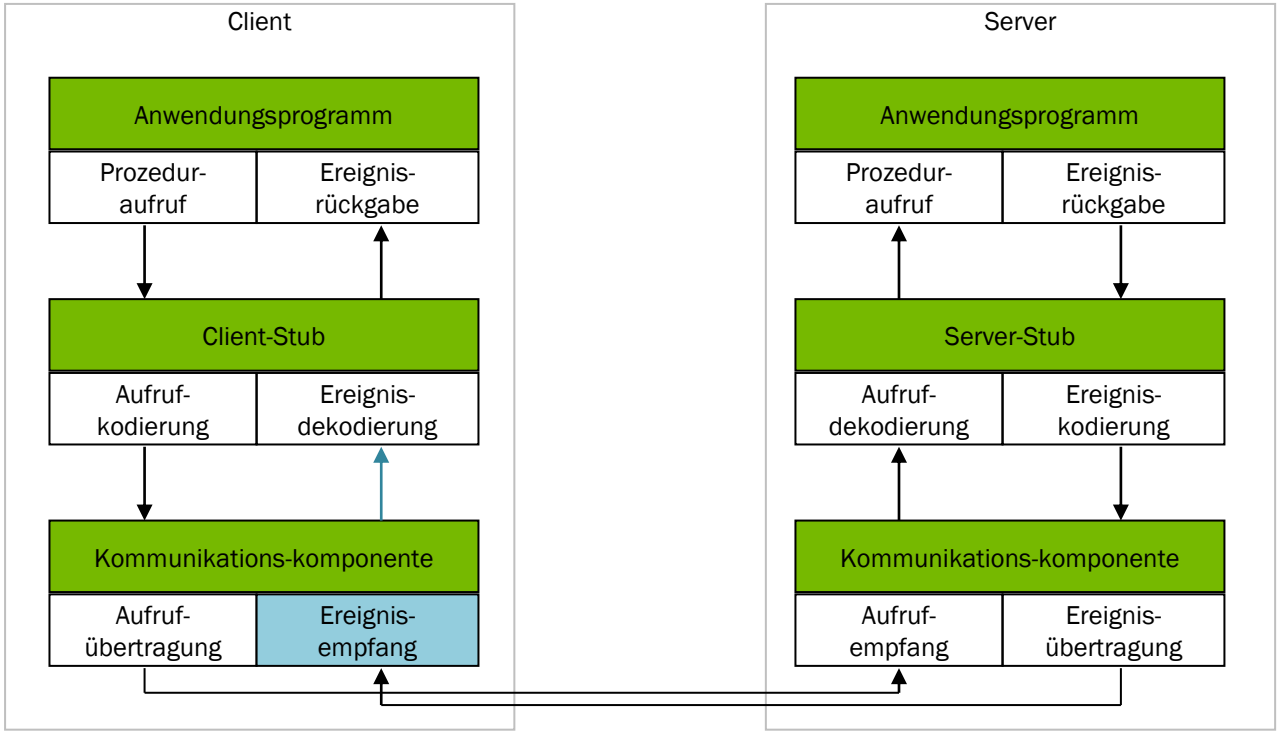


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [X]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

9. Das Client-Betriebssystem übergibt die Nachricht an den Client-Stub

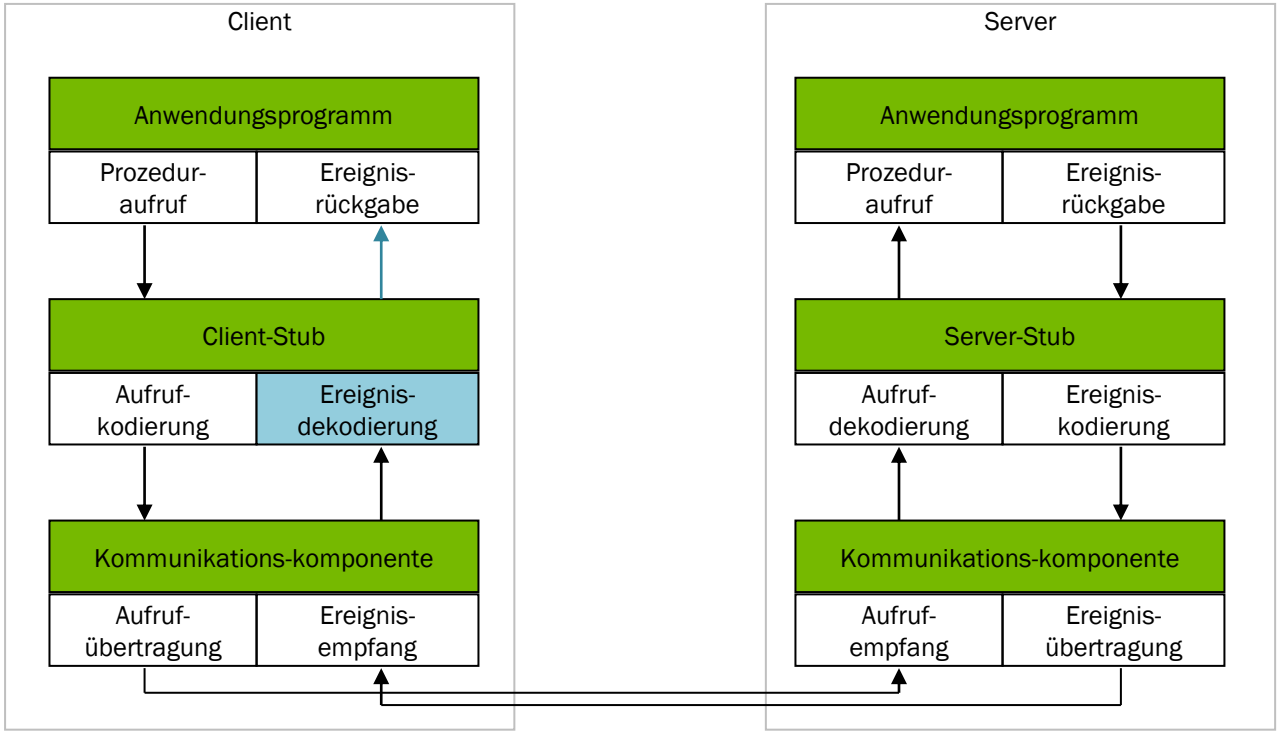


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [XI]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

10. *Client-Stub* entpackt das Ergebnis und gibt es an den Client zurück
- Entgegennahme der Nachricht des Server-Stubs
- Unmarshalling der erhaltenen Datenformate
- Übergabe von Ausgabeparametern/Rückgabewert an Client
- Deblockierung sowie Fortsetzen der lokalen Programmabarbeitung des Anwendungsprogramms

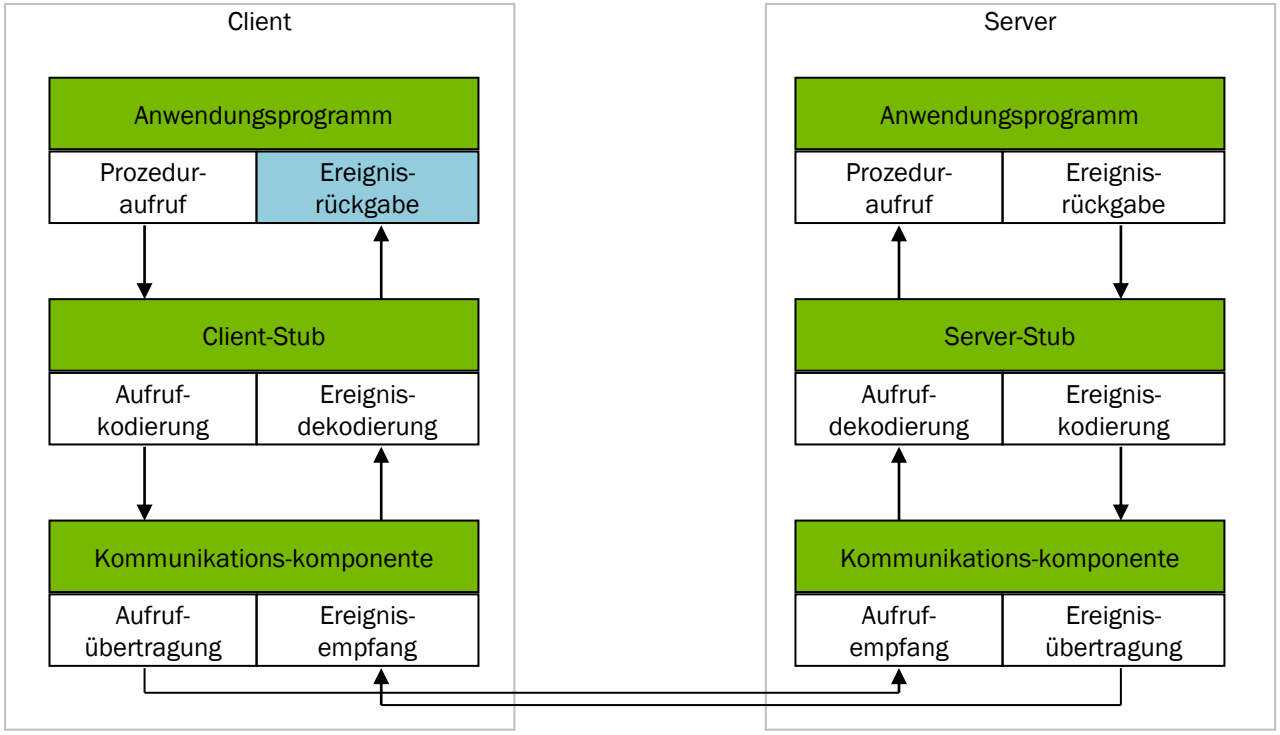


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [XII]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

- ✓ Nach Erhalt des Ergebnisses: Deblockierung sowie Fortsetzen der lokalen Programmabarbeitung des Anwendungsprogramms

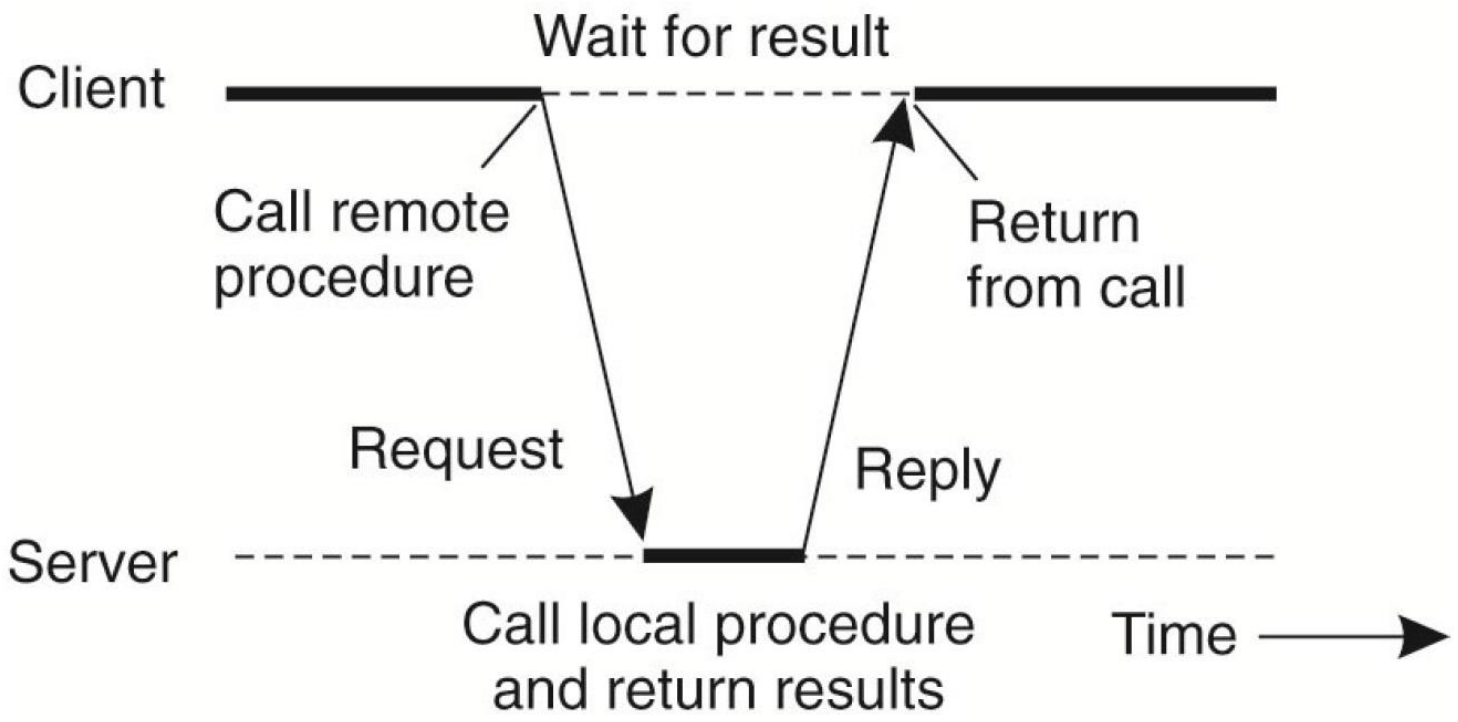


Quelle: i.A. an Bengel et al. [2008:271]

Remote Procedure Call (RPC): Funktionsweise und Ablaufstruktur [XIII]

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

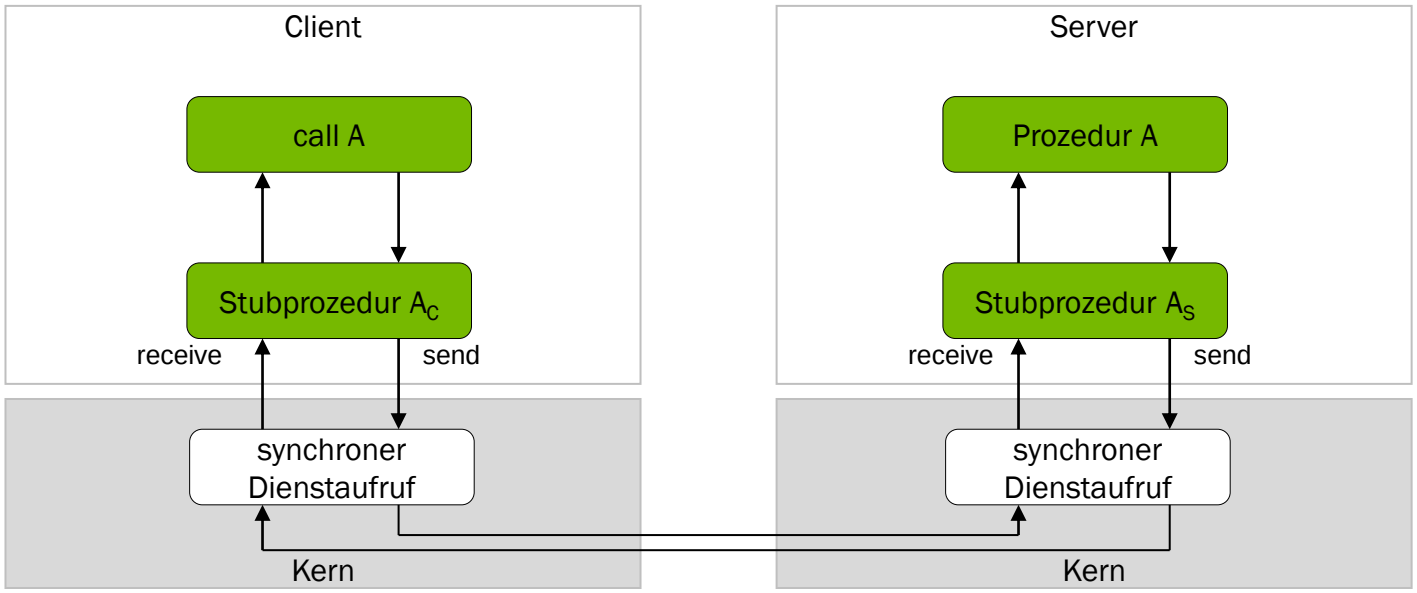
- ✓ Synchrones (blockierendes) Kommunikationsmodell



Quelle: Tanenbaum & van Steen. [2002:128]

Remote Procedure Call (RPC): Vorteile

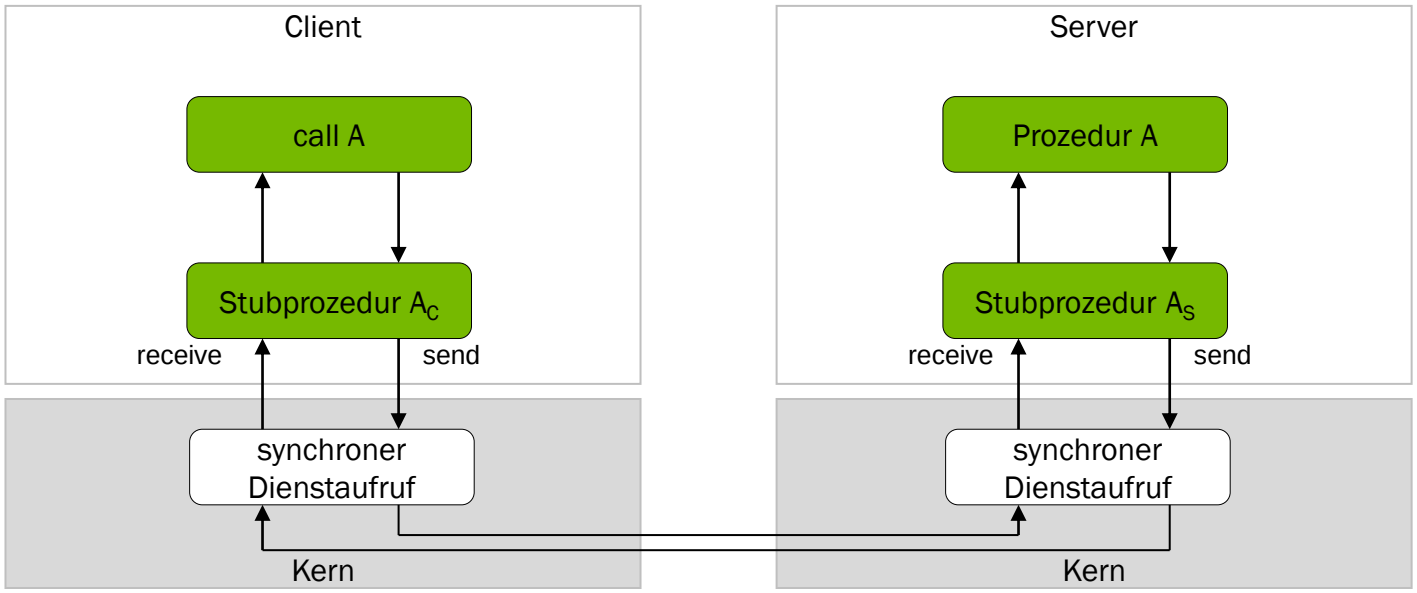
- ✓ Kommunikation mit Stubs ähnelt lokalen Aufrufen
- ✓ Ermöglichung der Nutzung elementarer Kommunikationsmechanismen (Sockets) mit Erweiterungen
- ✓ Erhöhtes Maß an Transparenz für Anwendungsschicht
- ✓ Programmierung ähnelt lokalen Aufrufen (Unterschied: Trennung zwischen Schnittstellendefinition und Implementierung der Schnittstelle, vgl. Entwicklungsschritte ff.)



Quelle: i.A. an Weber [1998:77]

Remote Procedure Call (RPC): Besonderheiten / Einschränkungen

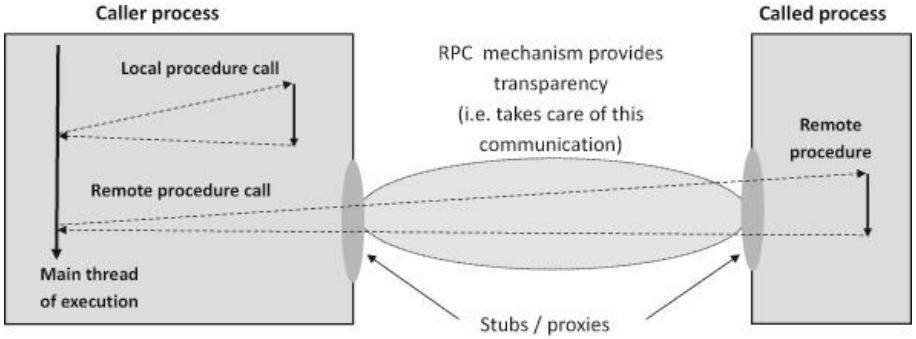
- ✓ Da ein Kommunikationssystem involviert ist, ergeben sich zusätzliche Fehlerquellen
- ✓ Da auf zwei verschiedenen Rechnern gearbeitet wird, gibt es keinen gemeinsamen Adressraum. Dies hat großen Einfluss auf die Parameterübergabe.
- ✓ Die beteiligten Prozesse haben separate Lebenszyklen. Es ist nicht sicher, ob die entfernte Prozedur immer ansprechbar ist.
- ✓ Durch den Kommunikationsoverhead ist die Aufrufdauer um Größenordnungen länger als bei lokalen Aufrufen.



Quelle: i.A. an Weber [1998:77]

Remote Procedure Call (RPC): Systeme

- In vielen OS-Distributionen und weiteren Anwendungen enthalten
- Sun RPC (Open Network Computing RPC):
  - IDL: Remote Procedure Call Language (RPCL; sprachunabhängig)
  - Maschinenunabhängige Datenrepräsentation: eXternal Data Representation (XDR)
  - Basis vieler UNIX-Dienste (z.B. Network File System (NFS), Network Information Service (NIS))
- Weiterentwicklungen: XML-RPC (-> vgl. Lecture Notes „Cloud Computing“: SOAP, WSDL, Web-Services), JSON-RPC
- ISO-RPC
- Remote Function Call (RFC, Aufruf von Funktionsbausteinen innerhalb von SAP R/3)
- Distributed Computing Environment DCE-RPC
- Weiterentwicklung: objektorientierte RPC-Systeme
  - RPC für Java: Java Remote Method Invocation (Java RMI, vgl. ff.) erlaubt entfernten Methoden-Aufruf zwischen Java-Objekten, die in verschiedenen virtuellen Maschinen residieren
  - Nachrichtenorientierte Middleware: DCOM (Distributed Component Object Model), Common Object Request Broker Architecture (CORBA))

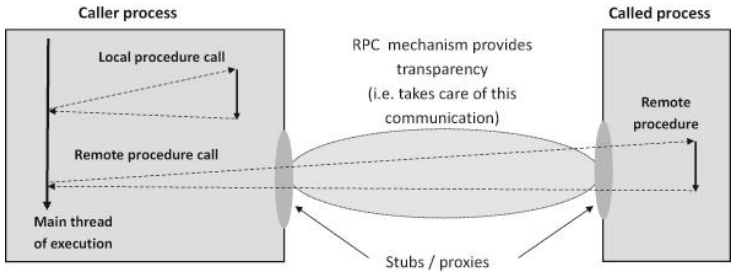


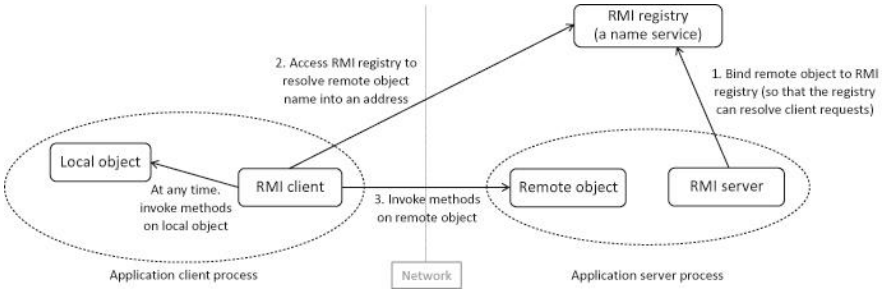
Quelle: Anthony [2016:126]

Remote Method Invocation (RMI): Begriffseingrenzung

Beschreibungen / Definitionen aus der Fachliteratur (Auszug):

Definition	Quelle(n)
„Remote Method Invocation (RMI) ermöglicht es, Methoden für Java-Objekte aufzurufen, die von einer anderen Java Virtual Machine (JVM) erzeugt und verwaltet werden. […] Ein solches Objekt einer anderen JVM nennt man dementsprechend entferntes Objekt (remote object).[…] RMI ist eine rein Java-basierte Lösung, deshalb muss nicht auf eine eigene Sprache zur Definition der Schnittstelle zurückgegriffen werden. Der Compiler [..] generiert den Stub für den Client und den Server direkt aus den existierenden Klassen für das Programm. Der Server Stub heißt bei Java Skeleton und stellt nur ein Skelett für den Aufruf dar. [..] Die Ansprache eines entfernten Objekts von der Client-Seite aus geschieht über einen handle-driven Broker. Alle entfernten Objekte sind beim Broker oder der Registry registriert. Die Adresse der Registry ist standardmäßig die Portnummer 1099. [..] Möchte nun ein Client eine Methode eines entfernten Objekts aufrufen, so muss er zunächst in der Registry nachschauen, ob das Objekt registriert ist. Ist es registriert, kann der Client anschließend den Stub für die entfernte Methode anfordern. Mit dem erhaltenen Stub kann dann die entfernte Methode aufgerufen werden.“	Bengel et al. [2008:285]
“Remote Method Invocation (RMI) is the object-oriented equivalent of RPC. Instead of remotely calling a procedure, a remote method is invoked. RMI was first introduced in the Java language, and this specific implementation is sometimes referred to as Java RMI. […] As with RPC, RMI performs its communication over a TCP connection, which is set up and managed silently from the programmer’s perspective.”	Anthony [2016:126f.]
“In particular, in RPC, procedures on remote machines can be called as if they are procedures in the local address space. The underlying RPC system then hides important aspects of distribution, including the encoding and decoding of parameters and results, the passing of messages and the preserving of the required semantics for the procedure call. This concept was first introduced by Birrell and Nelson [1984] and paved the way for many of the developments in distributed systems programming used today.”	Coulouris et al. [2012:195]

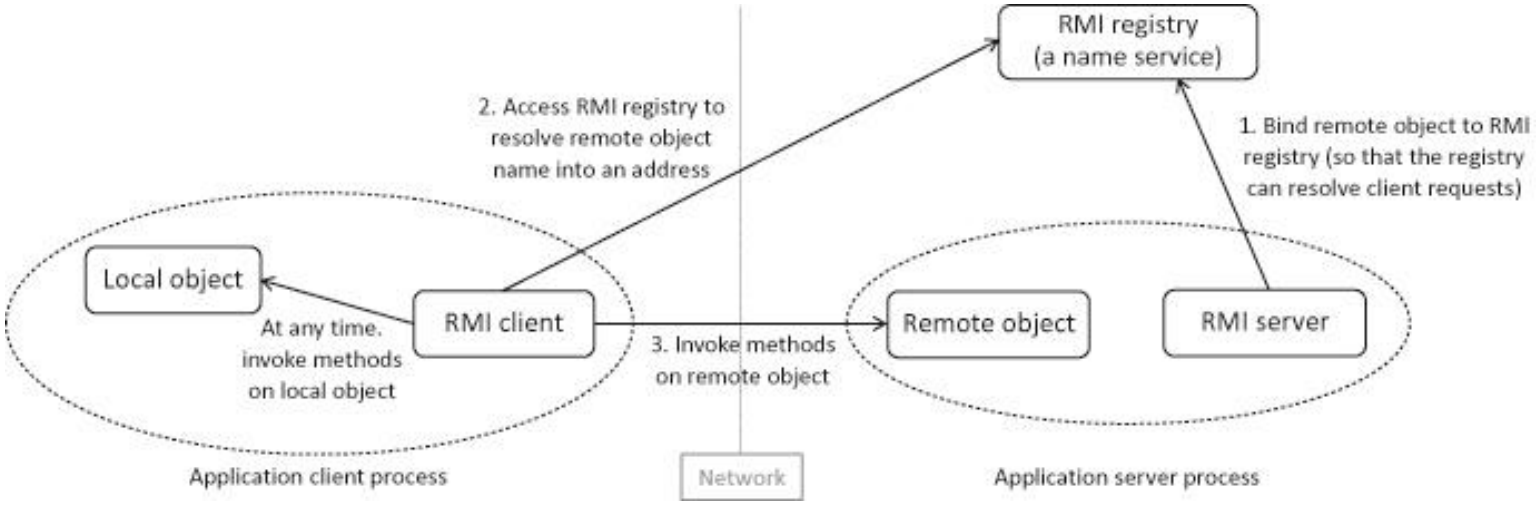




Quelle: Anthony [2016:126]

Remote Method Invocation (RMI): Komponenten

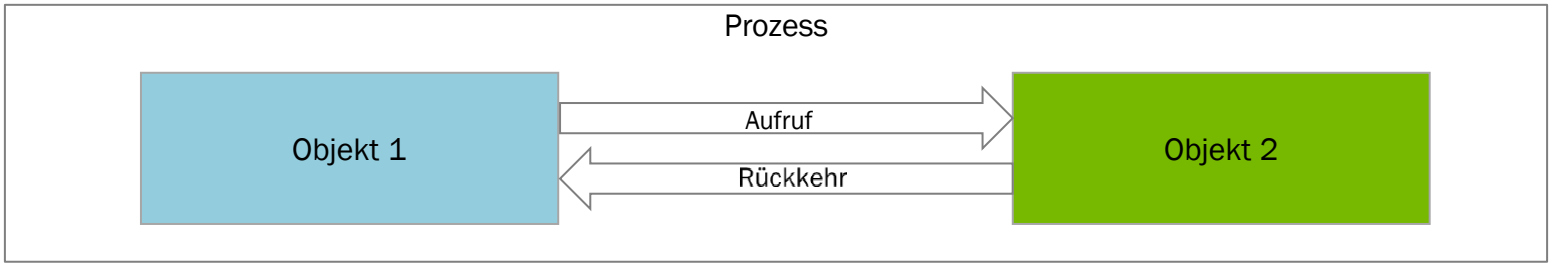
- ✓ Kommunikation mit Stubs ähnelt lokalen Aufrufen
- ✓ Ermöglichung der Nutzung elementarer Kommunikationsmechanismen (Sockets) mit Erweiterungen
- ✓ Erhöhtes Maß an Transparenz für Anwendungsschicht
- ✓ Programmierung ähnelt lokalen Aufrufen (Unterschied: Trennung zwischen Schnittstellendefinition und Implementierung der Schnittstelle, vgl. Entwicklungsschritte ff.)



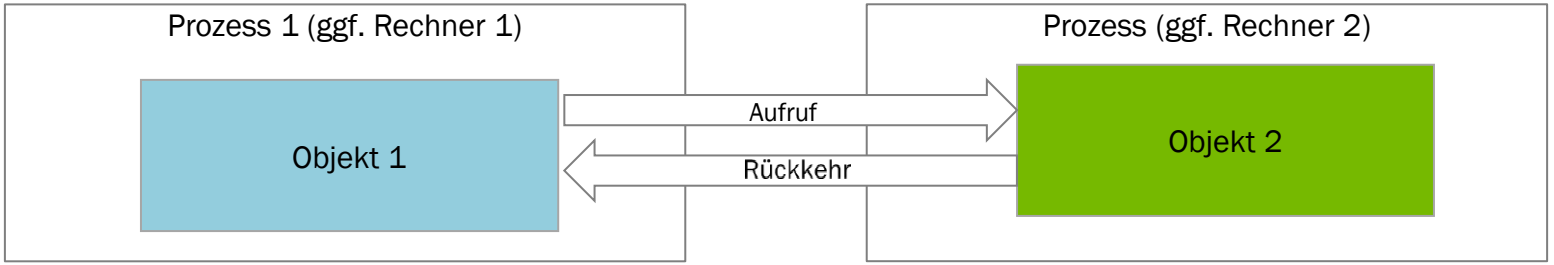
Quelle: Anthony [2016:126]

Remote Method Invocation (RMI): Funktionsweise und Ablaufstruktur [I]

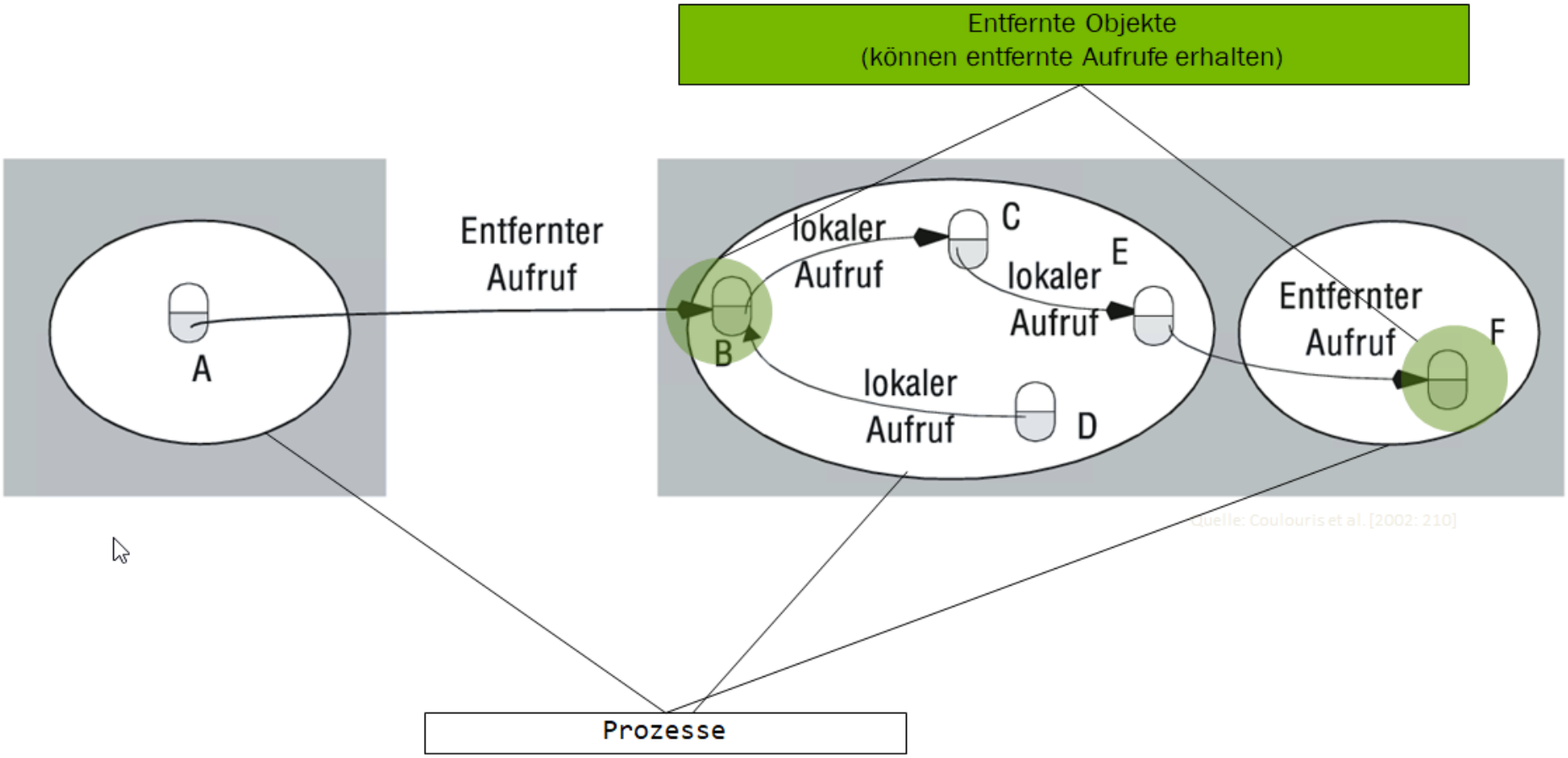
- ✓ Erweiterung des lokalen Methodenaufrufs, bei dem sich beide Objekte auf demselben Rechner, sogar im selben Adressraum (d.h. in demselben Prozess und derselben JVM) befinden.



- ✓ Fern-Methodenaufruf: beide Objekte befinden sich in unterschiedlichen Prozessen (und damit in unterschiedlichen Adressräumen / JVMs), i. d. R. (nicht zwingend) auch auf unterschiedlichen Rechnern



Verteiltes Objektmodell



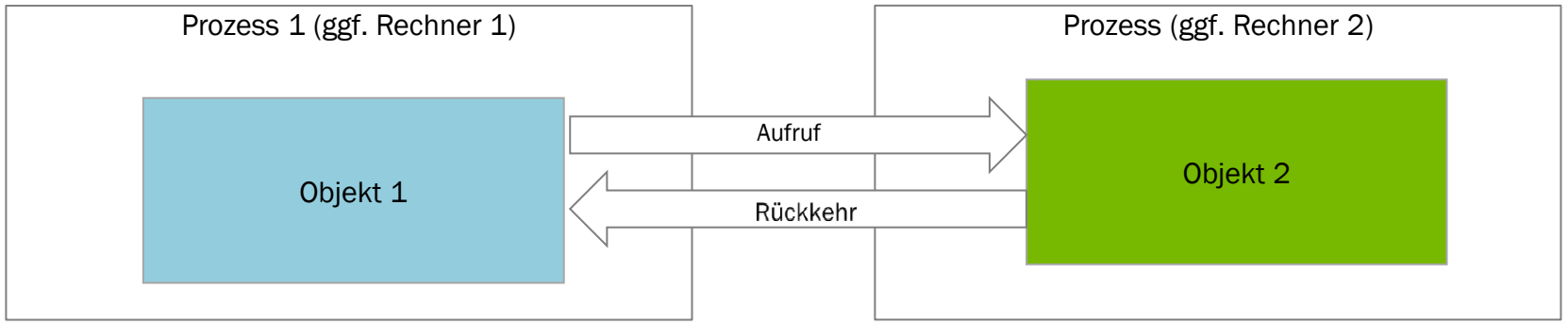
Quelle: Coulouris et al. [2002: 210]

Quelle: Coulouris et al. [2002: 210]

Entfernte Objekte und entfernte Schnittstellen [I]

(1) Objekte in anderen Prozessen können nur diejenigen Methoden aufrufen, die zu ihrer *entfernten Schnittstelle* gehören

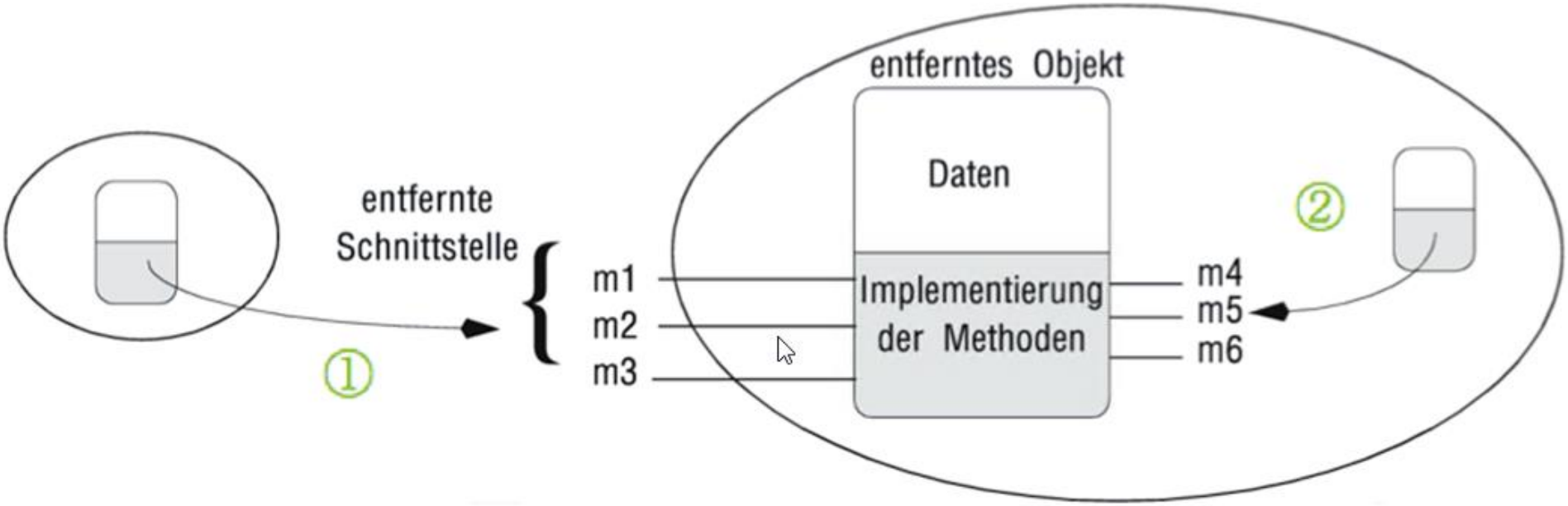
(2) Lokale Objekte können alle Methoden eines entfernten Objekts aufrufen (lokale Methoden + Methoden der entfernten Schnittstelle)



Entfernte Objekte und entfernte Schnittstellen [II]

(1) Objekte in anderen Prozessen können nur diejenigen Methoden aufrufen, die zu ihrer *entfernten Schnittstelle* gehören

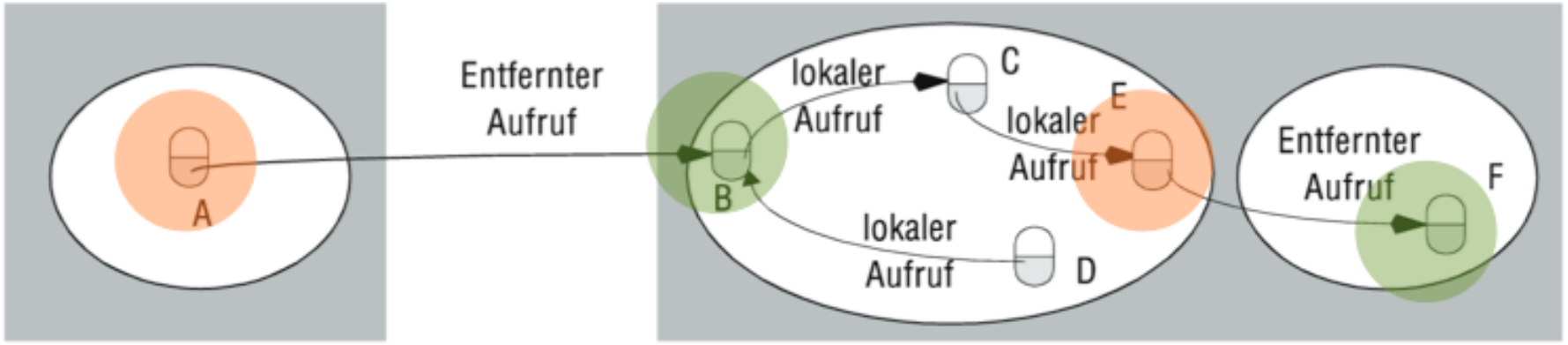
(2) Lokale Objekte können alle Methoden eines entfernten Objekts aufrufen (lokale Methoden + Methoden der entfernten Schnittstelle)



Quelle: Coulouris et al. [2002:211]

Voraussetzungen für die Verteilung [I]: Entfernte Objektreferenz

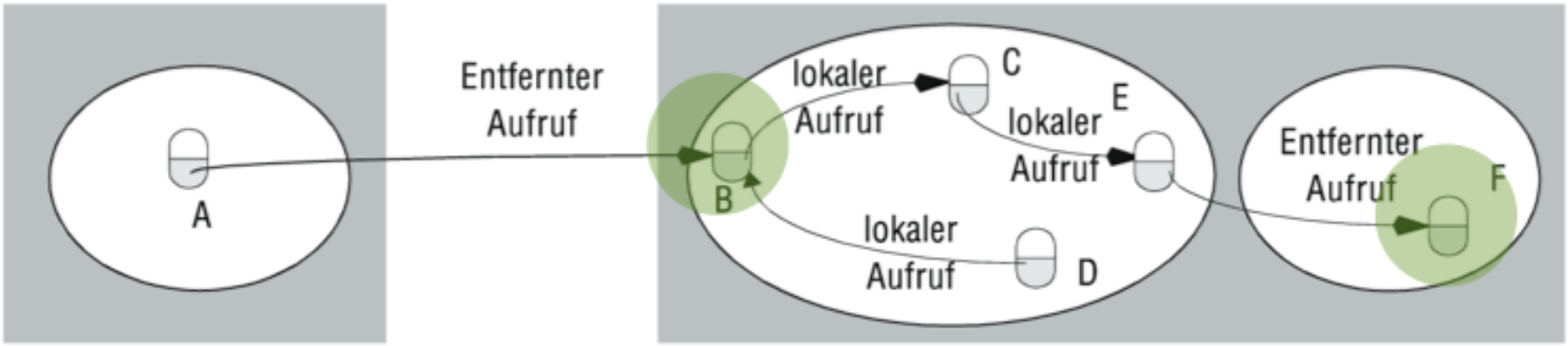
- Andere Objekte können die Methoden eines entfernten Objekts aufrufen, wenn sie Zugriff auf dessen Objektreferenz haben
- „entfernte Objektreferenz“ = „RemoteObjectId“ als Bestandteil einer Nachricht im Kontext des genutzten Protokolls
- Beispiel:
  - A benötigt Objektreferenz von B
  - E benötigt Objektreferenz von F



Quelle: Coulouris et al. [2002:210]

Voraussetzungen für die Verteilung [II]: Entfernte Schnittstelle

- Jedes entfernte Objekt hat eine entfernte Schnittstelle (diese gibt an, welche seiner Methoden entfernt aufgerufen werden können)
- Die Klasse des entfernten Objekts implementiert dessen definierte Schnittstelle
- Schnittstelle
  - » enthält Definition der Typen der Argumente, Rückgabewerte und Ausnahmen
  - » enthält nicht die Implementierung der Schnittstelle
- Beispiel: B und F müssen entfernte Schnittstelle bereitstellen



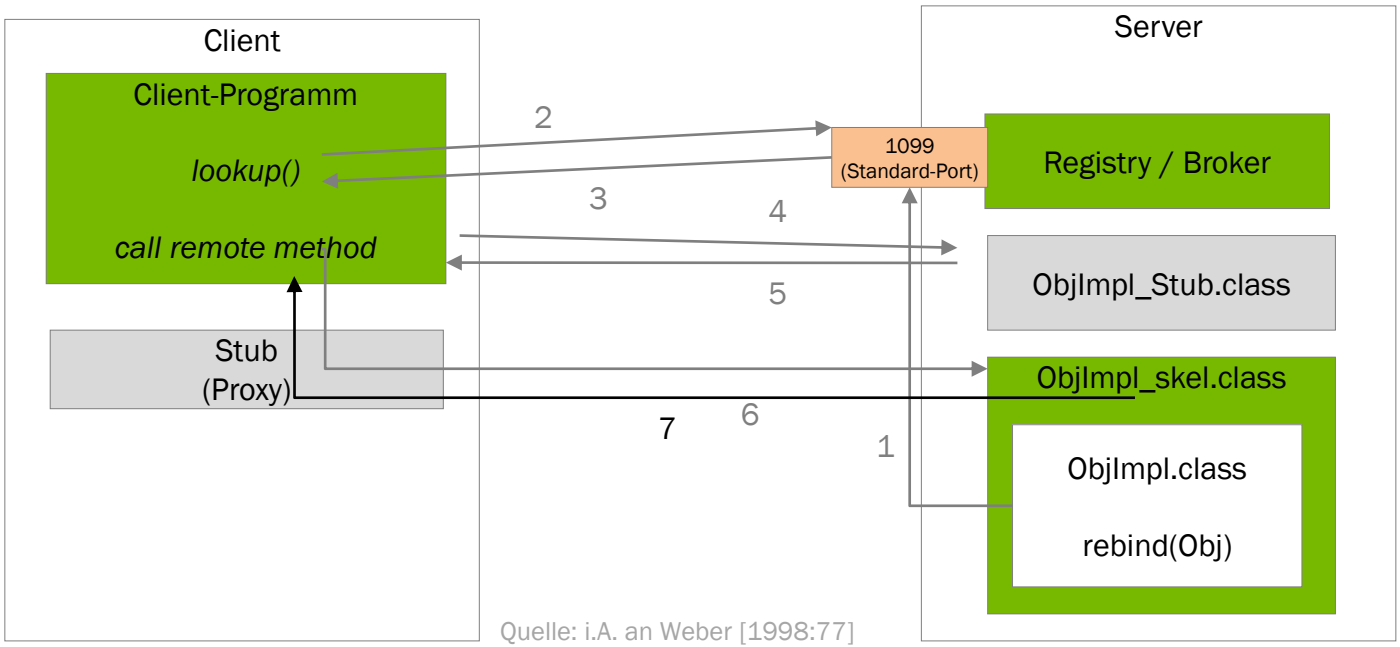
Quelle: Coulouris et al. [2002: 210]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [I]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

- ✓ Verteilungstransparenz: Programmierer_innen müssen Schnittstelle definieren, Client und Server, aber nicht:
  - ✓ Stub: wird automatisch erzeugt
  - ✓ Skeleton: in RMI implementierter Programmteil

Erzeugung durch RMI-Compiler

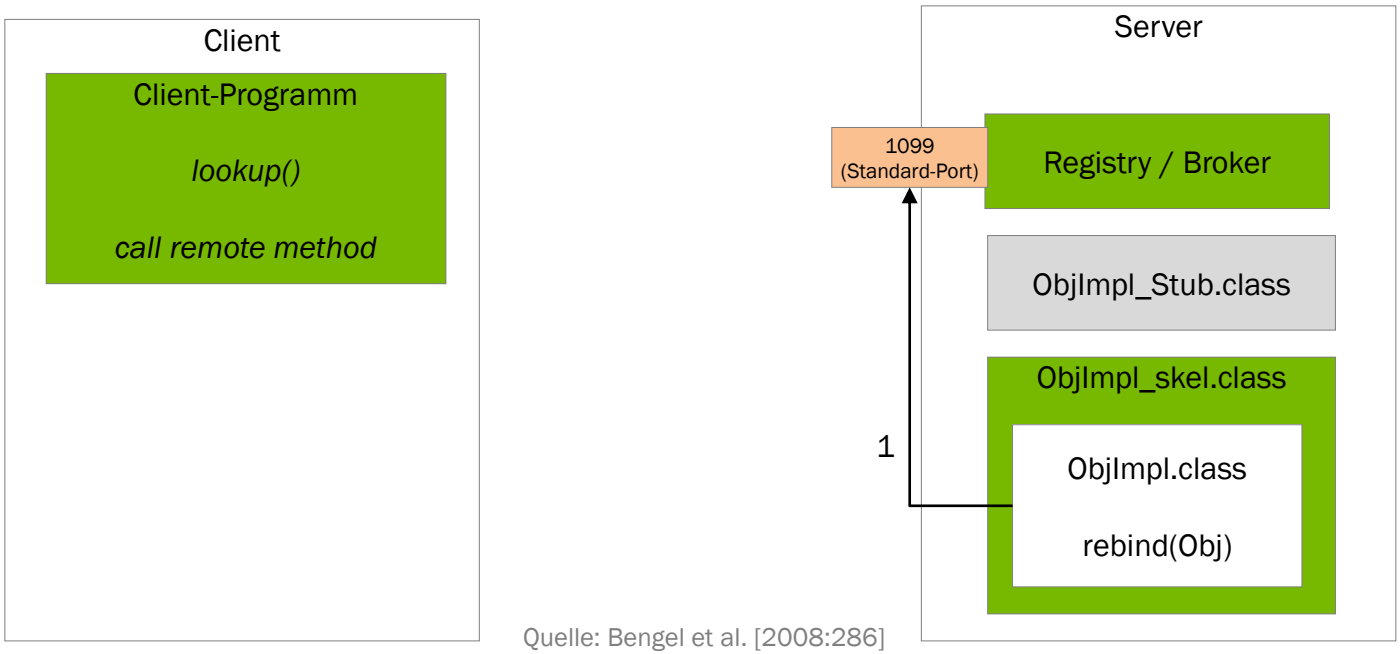


Quelle: i.A. an Weber [1998:77]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [II]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

1) Registriere entferntes Objekt in der Registry

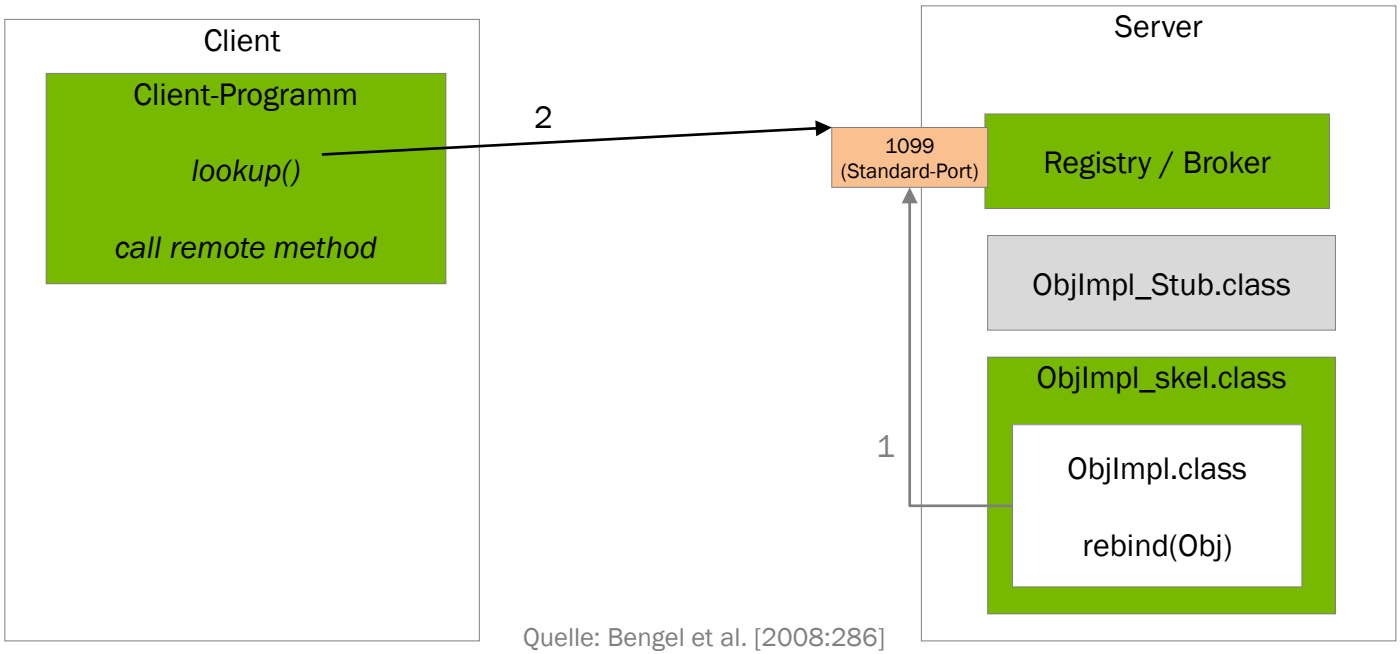


Quelle: Bengel et al. [2008:286]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [III]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

1) Registriere entferntes Objekt in der Registry
2) Lookup(): wo ist das entfernte Objekt?

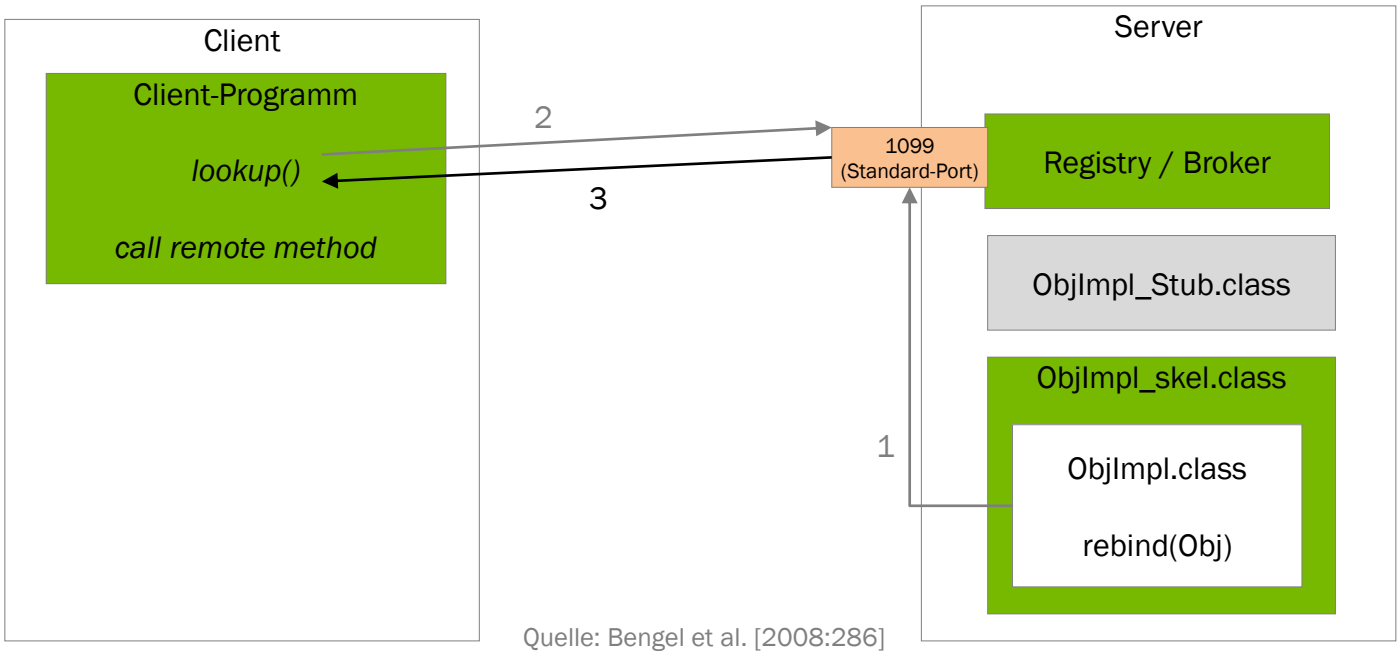


Quelle: Bengel et al. [2008:286]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [IV]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

1) Registriere entferntes Objekt in der Registry
2) Lookup(): wo ist das entfernte Objekt?
3) Entferntes Objekt gefunden

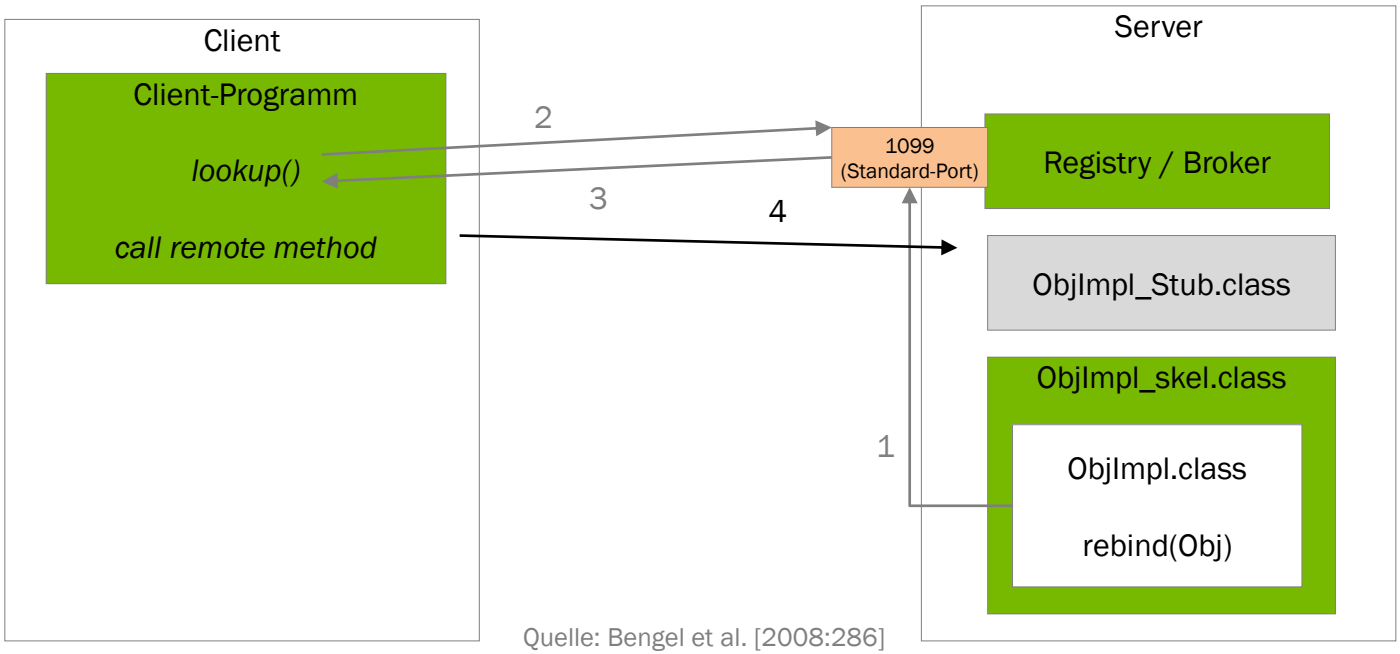


Quelle: Bengel et al. [2008:286]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [V]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

1) Registriere entferntes Objekt in der Registry
2) Lookup(): wo ist das entfernte Objekt?
3) Entferntes Objekt gefunden
4) Gib mir den Stub (Proxy)

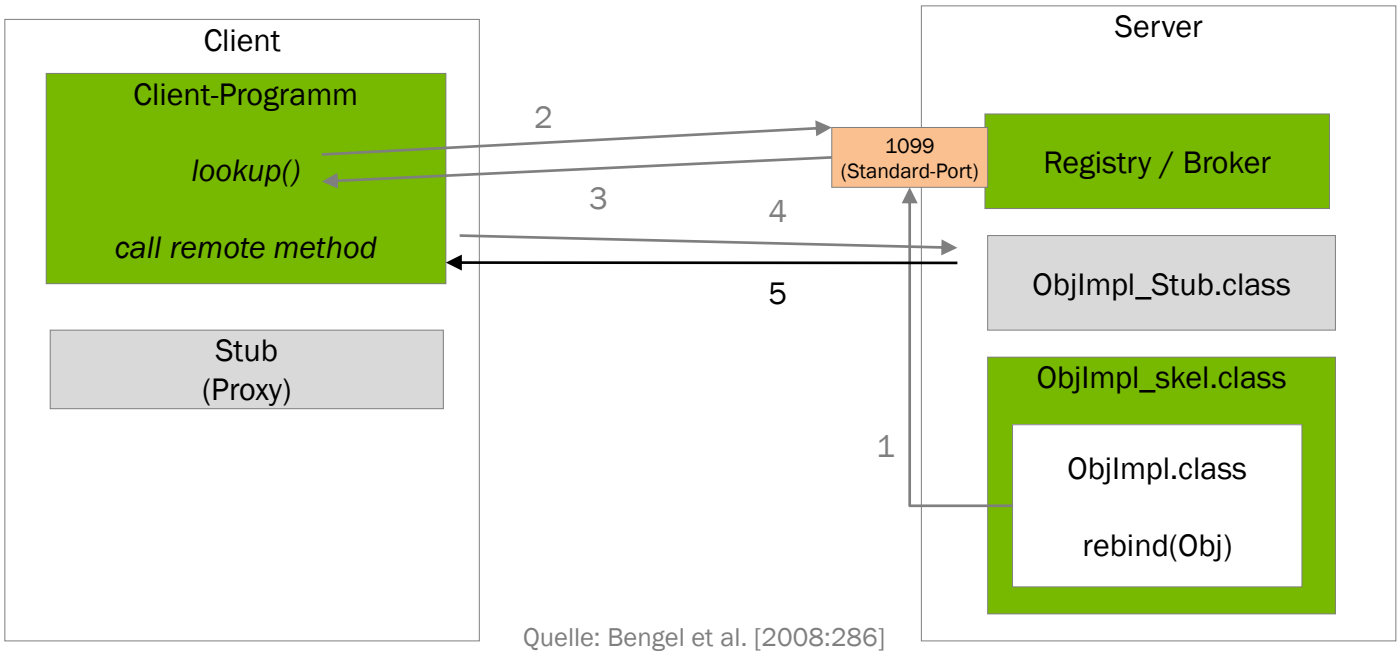


Quelle: Bengel et al. [2008:286]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [VI]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

1) Registriere entferntes Objekt in der Registry
2) Lookup(): wo ist das entfernte Objekt?
3) Entferntes Objekt gefunden
4) Gib mir den Stub (Proxy)
5) Hier ist der Stub (Proxy)

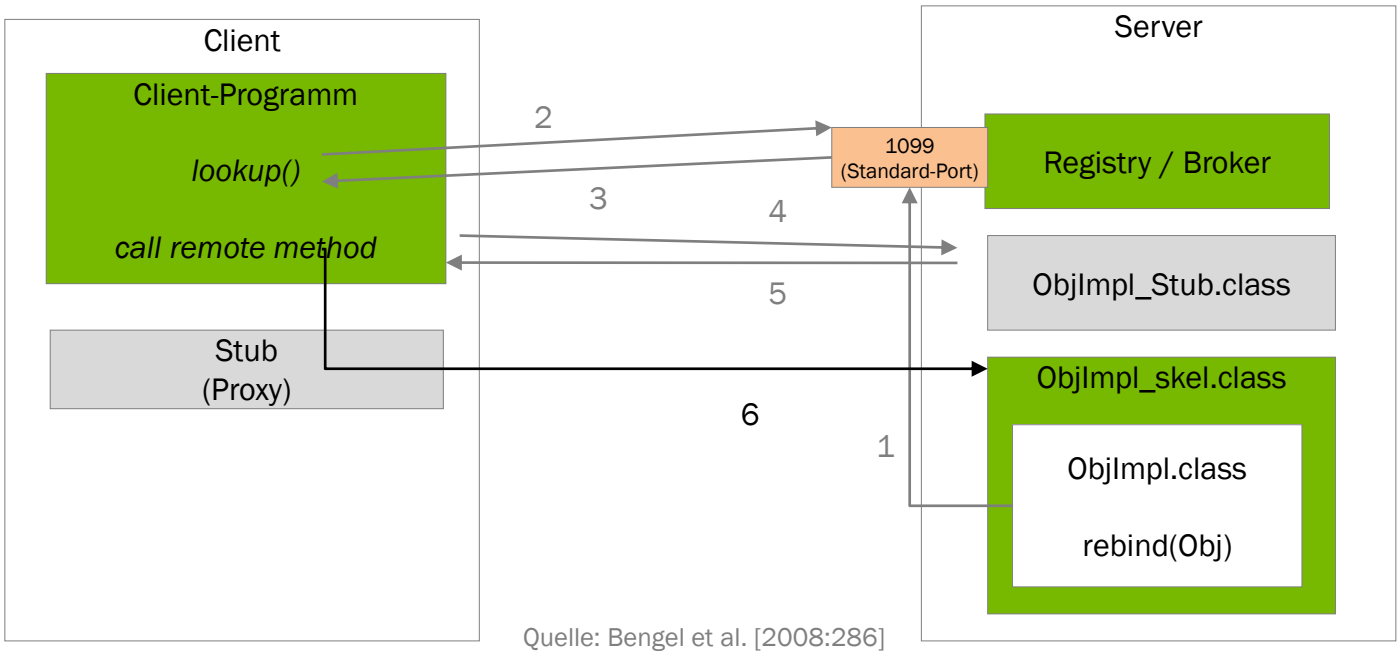


Quelle: Bengel et al. [2008:286]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [VII]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

1) Registriere entferntes Objekt in der Registry
2) Lookup(): wo ist das entfernte Objekt?
3) Entferntes Objekt gefunden
4) Gib mir den Stub (Proxy)
5) Hier ist der Stub (Proxy)
6) Rufe entfernte Methode auf
   - » Stub: verpacke Parameter und sende diese zum Server
   - » Skeleton: Entpacke Parameter und rufe Methode auf

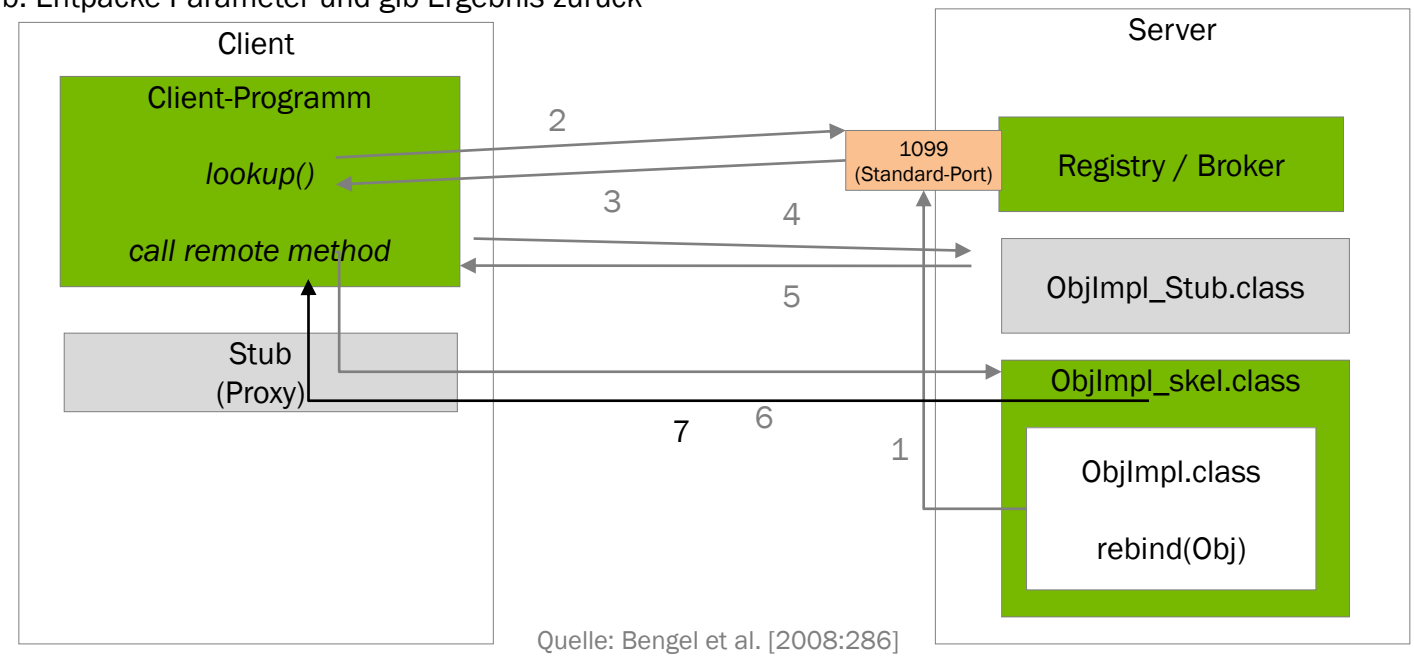


Quelle: Bengel et al. [2008:286]

Remote Method Invocation (RMI): Komponenten und Ablaufstruktur [VIII]

vgl. Bengel [2004:170ff.], Bengel et al. [2008:285ff.]

1) Registriere entferntes Objekt in der Registry
2) Lookup(): wo ist das entfernte Objekt?
3) Entferntes Objekt gefunden
4) Gib mir den Stub (Proxy)
5) Hier ist der Stub (Proxy)
6) Rufe entfernte Methode auf
   - » Stub: verpacke Parameter und sende diese zum Server
   - » Skeleton: Entpacke Parameter und rufe Methode auf
7) Gib Ergebnis zurück
   - » Skeleton: verpacke Parameter und sende diese zum Client
   - » Stub: Entpacke Parameter und gib Ergebnis zurück



Quelle: Bengel et al. [2008:286]

Remote Method Invocation (RMI): Entwurfsaspekte

- Ausgangspunkt: Lokale Aufrufe werden einmal ausgeführt (dies kann und soll für entfernte Aufrufe nicht immer der Fall sein)
- Gestaltung der programmatischen Aufrufsemantik (vgl. Appendix: Aufrufsemantik) von Methodenaufrufen unter Berücksichtigung der Kriterien:
  - Wiederholung der Übertragung einer Anforderungsnachricht
  - Duplikatfilterung redundant übertragener Anforderungen
  - Wiederholung der Übertragung einer Ergebnisnachricht
  - Verschiedene Kombinationen sind möglich
- Fehler können auftreten (Netzwerk, Prozess, vgl. Appendix)
- Objekte, die entfernte Aufrufe vornehmen, sollten Risiken minimieren
  - So wenig Interaktion wie möglich
  - Wiederherstellung von Zuständen vor einem Aufruf bei Fehlern von nicht idempotenten Methoden auf Server-Seite (ACID-Prinzip: Atomarität, Konsistenz, Isolation, Dauerhaftigkeit)

Remote Method Invocation (RMI): Entwicklungsschritte

Schritt 1: Definition der Schnittstelle

Schritt 2: Implementierung der Schnittstelle

Schritt 3: Programmierung des Servers

Schritt 4: Programmierung des Clients

Schritt 5: Übersetzung/Ausführung der Anwendung

Beispiel: „Hello World“ (distributed)

vgl. Dunkel et al. 2008: 35ff.;
Oracle (online: https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html; letzter Zugriff: 6 I 18)

- Hello.java ist die gemeinsame Remote-Schnittstelle (remote interface) als Kontrakt für Client und Server
- Server.java ist das Server-Hauptprogramm: implementiert ein entferntes Objekt, welches wiederum das remote interface implementiert
- Client.java ist ein Java RMI-Client als zum Hallo-Server passendes Beispiel; ruft eine entfernte Methode der Remote-Schnittstelle auf

Schritt 1: Definition der Schnittstelle [I]

Wir erinnern uns…Schnittstelle (Interface):

- Ermöglicht Interaktion zwischen Modulen eines Programms
- Gibt Prozeduren und Variablen an, auf die andere Module zugreifen können

Beispiel für einen in eine Programmiersprache integrierten RMI-Mechanismus: Java RMI

- Unterstützt geeignete Notation für die Schnittstellendefinition
- Erlaubt Zuordnung von Ein- und Ausgabeparametern
- Entwicklungsaspekte:
  - » Ableitung der Schnittstelle Remote aus dem Package java.rmi
  - » Erforderlich: Fehler-/Ausnahmebehandlung - Kennzeichnung der Methoden der Schnittstelle mit „throws RemoteException“
  - » Vorteil: Entwickler kann lokale und entfernte Aufrufe mit der gleichen Sprache realisieren

```
import java.rmi.*;

public interface BEZEICHNERDERSCHNITTSTELLE extends Remote
{
        public int METHODE1() throws RemoteException;
        public int METHODE2() throws RemoteException;
}
```

Schritt 1: Beispiel „Hello World“ (distributed)“ - entfernte Schnittstelle (remote interface)

vgl. Dunkel et al. 2008: 35ff.; Oracle

- Hello.java ist die gemeinsame Remote-Schnittstelle (remote interface) als Kontrakt für Client und Server

```
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4  public interface Hello extends Remote {
5      String sayHello() throws RemoteException;
6  }
```

Schritt 2: Implementierung der Schnittstelle

- Erforderlich: Klasse, die die Definition der Schnittstelle implementiert
- Beachten:
  - » Einem Objekt muss Nutzung von außen über RMI eingeräumt werden („Export“)
  - » Möglichkeit: Ableitung der Klasse aus *UnicastRemoteObject*

```
import java.rmi.*;
Import java.rmi.server.UnicastRemoteObject;

public class BEZEICHNERDERSCHNITTSTELLENIMPLEMENTIERUNG extends UnicastRemoteObject implements
BEZEICHNERDERSCHNITTSTELLE
{
        ...;
}
```

Schritt 3: Programmierung des Servers

- Ausgangspunkt: Schritt 2 erzeugte eine Klasse, welche die Schnittstelle (definiert in Schritt 1) implementiert
- Klassen können / sollen Objekte enthalten
- Schritt 3 erzeugt eines oder mehrere Objekte der Klasse aus Schritt 2
- Die Objekte können bei Auskunftsdienst („Binder“, RMIregistry) unter einem frei wählbaren Namen angemeldet werden

Schritt 2 & 3: Beispiel „Hello World“ (distributed)“ - Server [I]

w/out comments

vgl. Dunkel et al. 2008: 35ff.; Oracle

```
1  import java.rmi.registry.Registry;
2  import java.rmi.registry.LocateRegistry;
3  import java.rmi.RemoteException;
4  import java.rmi.server.UnicastRemoteObject;
5
6  public class Server implements Hello {
7
8      public Server() {}
9      public String sayHello() {
10         return "Hello, world!";
11     }
12     public static void main(String args[]) {
13
14         try {
15             Server obj = new Server();
16             Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);
17             Registry registry = LocateRegistry.getRegistry();
18             registry.bind("Hello", stub);
19             System.err.println("Server ready");
20         } catch (Exception e) {
21             System.err.println("Server exception: " + e.toString());
22             e.printStackTrace();
23         }
24     }
25 }
```

*/

Schritt 2 & 3: Beispiel „Hello World“ (distributed)“ - Server [II]

w/ comments

vgl. Dunkel et al. 2008: 35ff.; Oracle

```java
/* Quellen: Dunkel et al. 2008: 35ff. (Kap. 3.3.2 RPC mit Java RMI);
Oracle (online: https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html; 6 I 18
Server.java ist das Server-Hauptprogramm: implementiert die Remote-Schnittstelle und stellt Implementierung
der remote method im remote interface */

import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

/* The implementation class Server implements the remote interface Hello, providing an implementation for the
remote method sayHello. Note: A class can define methods not specified in the remote interface, but those methods
can only be invoked within the virtual machine running the service and cannot be invoked remotely. */

public class Server implements Hello {

    public Server() {}
        /* The method sayHello does not need to declare that it throws any exception because the method implementation
    itself does not throw RemoteException nor does it throw any other checked exceptions. */
    public String sayHello() {
        return "Hello, world!";
    }
    /* The Server's main method creates and exports a remote object and registers the remote object with a Java RMI registry */
    public static void main(String args[]) {

        try {
                // Create an instance of the remote object implementation
            Server obj = new Server();
            /* Export the remote object, for a caller (client, peer, or applet) to be able to invoke a method on a remote object,
            that caller must first obtain a stub for the remote object. */
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);
            /* Bind the remote object's stub in the registry; a Java RMI registry is a simplified name service that
            allows clients to get a reference (a stub) to a remote object. */
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);

            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

*/

Schritt 4: Programmierung des Clients

- Programm, welches sich über die Auskunft einen Stub für das beim Server angemeldete RMI-Objekt (Schritt 3) beschafft
  - » Client muss Rechner kennen, auf dem Server läuft
  - » Client muss Namen des Objekts kennen, welches vom Server in Schritt 3 angemeldet wurde
- Stub kann so verwendet werden, als wäre er das RMI-Objekt selbst, d.h.:
  - » Client nutzt Methoden der Schnittstelle aus Schritt 1 (Schnittstellendefinition)
  - » Durch Nutzung der Methoden aus Schritt 1 werden deren in Schritt 2 implementierte Methoden auf dem RMI-Objekt des Servers ausgeführt

Schritt 4: Beispiel „Hello World“ (distributed)“ – Client

w/out comments

vgl. Dunkel et al. 2008: 35ff.; Oracle

```
1  import java.rmi.registry.LocateRegistry;
2  import java.rmi.registry.Registry;
3
4  public class Client {
5
6      private Client() {}
7
8      public static void main(String[] args) {
9
10         String host = (args.length < 1) ? null : args[0];
11         try {
12             Registry registry = LocateRegistry.getRegistry(host);
13             Hello stub = (Hello) registry.lookup("Hello");
14             String response = stub.sayHello();
15             System.out.println("response: " + response);
16         } catch (Exception e) {
17             System.err.println("Client exception: " + e.toString());
18             e.printStackTrace();
19         }
20     }
21 }
```

Schritt 4: Beispiel „Hello World“ (distributed)“ – Client

w/ comments

vgl. Dunkel et al. 2008: 35ff.; Oracle

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

// The client program obtains a stub for the registry on the server's host,
// looks up the remote object's stub by name in the registry,
// and then invokes the sayHello method on the remote object using the stub.
public class Client {

    private Client() {}

    public static void main(String[] args) {

        String host = (args.length < 1) ? null : args[0];
        try {
            // This client first obtains the stub for the registry by
            // invoking the static LocateRegistry.getRegistry method with the hostname specified on the command line.
            // If no hostname is specified, then null is used as the hostname indicating
            // that the local host address should be used.
            Registry registry = LocateRegistry.getRegistry(host);
            // Next, the client invokes the remote method lookup on the registry stub
            // to obtain the stub for the remote object from the server's registry.
            Hello stub = (Hello) registry.lookup("Hello");
            // Finally, the client invokes the sayHello method on
            // the remote object's stub, which causes the following actions to happen:
            // The client-side runtime opens a connection to the server using the
            // host and port information in the remote object's stub and then serializes the call data.
            // The server-side runtime accepts the incoming call, dispatches the call to the remote object,
            // and serializes the result (the reply string "Hello, world!") to the client.
            // The client-side runtime receives, deserializes, and returns the result to the caller.
            String response = stub.sayHello();
            // The response message returned from the
            // remote invocation on the remote object is then printed to System.out
            System.out.println("response: " + response);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Schritt 5: Übersetzung/Ausführung der Anwendung

- Übersetzung aller JAVA-Dateien
  - » Hello.java ist die gemeinsame Remote-Schnittstelle (remote interface) als Kontrakt für Client und Server
  - » Server.java ist das Server-Hauptprogramm: implementiert ein entferntes Objekt, welches wiederum das remote interface implementiert
  - » Client.java ist ein Java RMI-Client als zum Hallo-Server passendes Beispiel; ruft eine entfernte Methode der Remote-Schnittstelle auf
- Ausführung der Programme: Vor dem Ausführen des Server-programms muss die RMI-Registry gestartet werden (Server wird Objekte bei diesem Auskunftsdienst anmelden)

Wir erinnern uns: Schritt 1: Definition der Schnittstelle

- Da nicht alle VS in der gleichen Sprache geschrieben sind, werden Interface Definition Languages (IDLs) benötigt, um entfernten Zugriff auf Programme/Objekte zu erlauben, die in unterschiedlichen Sprachen geschrieben/implementiert sind.

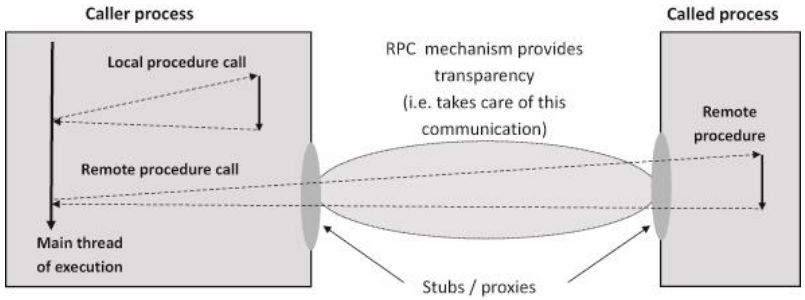
- IDLs stellen Notation für die Definition von Schnittstellen bereit
  - » Jeder Parameter einer Methode kann für die Ein- und Ausgabe vorgesehen werden
  - » Jeder Parameter einer Methode enthält eine Typzuordnung

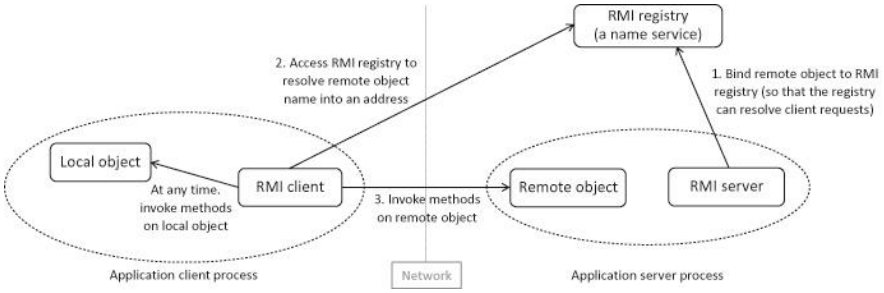
✓ Beispiele: vgl. Appendix (z.B. Nutzung konstruierter Typen in CORBA (Common Object Request Broker Architecture))

Remote Procedure Call (RPC) vs. Remote Method Invocation (RMI)

RPC gleicht einem RMI

- Statt Methoden werden Prozeduren aufgerufen
- Entfernter Aufruf kann erfolgen, wenn Server eine Dienstschnittstelle bereit stellt (≠ entfernte Schnittstelle bei RMI)
- Fehlersemantik/Aufrufsemantik (wie bei RMI, vgl. Appendix)
- Implementierung über Anforderung-/Antwort-Protokoll (RPC: Objektreferenzen werden weggelassen, vgl. Appendix)





Quelle: Anthony [2016:126]

Serverzustände: Klassifizierung

vgl. Bengel et al. [2008:115ff.]

Ausgangspunkt: Eine Anforderung, die ein Client an einen Server sendet, kann evtl. Änderungen in vom Server verwalteten Daten oder Objekten hervorrufen.

Zustandsinvariante Server:

- Anforderungen von Clients führen nicht zu neuen Zuständen des Servers
- Reihenfolge der Client-Anfragen beeinflusst Server nicht
- Server liefert (variable) Informationsmenge und Parameterausprägung unabhängig von Client-Anfrage nach Information und Parameter
- Beispiele: Web-Server, ftp-Server, Auskunfts-Server (z.B. Dienste: Name, Directory, Vermittlung, Broker, Zeit,...)

Zustandsändernde Server:

- Anforderungen von Clients führen ggf. zu neuen Zuständen des Servers
- Reihenfolge der Client-Anfragen beeinflusst Server (Abhängig vom neuen Zustand können gewisse weitere Anforderungen von Clients nicht mehr befriedigt werden (Beispiel: Fehlermeldungen bei File-Server im Falle einer Anforderung read bei voriger Löschung))
- Zustandsspeichernd (stateful): Server speichert seinen neuen Zustand
  - » Vorteil: vergleichsweise höhere Performanz (Zustandsinformation im Hauptspeicher)
  - » Nachteil: Mögliche Inkonsistenzen nach Absturz, da Zustandsbeschreibungen alle verloren
- Zustandslos (stateless): Server speichert seinen neuen Zustand nicht (Client muss Zustandsinformation vorhalten
  - » Vorteil: erhöhte Fehlertoleranz (z.B. keine Inkonsistenzen bei Absturz, da Zustandsbeschreibungen bei Clients)
  - » Nachteil: vergleichsweise geringere Performanz (keine Zustandsinformation im Hauptspeicher eines Servers, sondern bei jeweiligem Client)

Danke. Lernziele erreicht?

Nach dieser Lehrveranstaltung kennen Studierende idealerweise:

- Erweiterungen elementarer Kommunikationsmechanismen der Interprozesskommunikation auf Basis von Sockets (vgl. Lecture Notes IPC Teil I)
- Middleware als Softwareschicht zwischen Transport- und Anwendungsschicht zur Erhöhung von Transparenz und Überwindung von Interoperabilitätsproblemen aufgrund von Heterogenität
- Beispiele für Middleware-Kommunikationsformen sowie Prinzipien verteilter Aufrufe
- Die Bedeutung von Schnittstellen für Anwendungsentwickler_innen
- Ausgewählte Beispiele für höhere Konzepte bei der Interprozesskommunikation:
  - Remote Procedure Call (RPC: entfernte Prozeduraufrufe)
  - Remote Method Invocation (RMI: entfernter Methodenaufruf)
- Entwicklungsschritte verteilter Aufrufe am praktischen Beispiels einer einfachen RMI-Anwendung
- Serverzustände

✓ Studierende kennen grundlegende Aspekte der Interprozesskommunikation auf Middleware-Ebene, Eigenschaften damit verbundener ausgewählter Konzepte und Technologien sowie deren grundlegende Funktionsweise im Zusammenspiel beteiligter Komponenten. Zudem lernen Studierende praktische Beispiele für die Implementierung einfacher verteilter Systeme mittels entfernter Aufrufe und (entfernten) Schnittstellen kennen, welche als Grundlage sowie Bestandteil weiterführender Standards und Middleware-Kommunikationsformen dienen.

Quellen

Anthony, R. (2016) *Systems Programming - Designing and Developing Distributed Applications*; Amsterdam et al.: Morgan-Kaufman / Elsevier.

Bengel, G. (2004) *Grundkurs Verteilte Systeme*; Wiesbaden: Vieweg.

Bengel, G.; Baun, C.; Kunze, M.; Stucky, K.-U. (2008) *Masterkurs Parallele und Verteilte Systeme*; Wiesbaden: Vieweg & Teubner.

Coulouris, G.; Dollimore, J.; Kindberg, T. (2002) *Verteilte Systeme - Konzepte und Design*; 3., überarbeitete Auflage; München: Pearson Studium.

Coulouris, G.; Dollimore, J.; Kindberg, T., Blair. G. (2012) *Distributed Systems - Concepts and Design*; 5., überarbeitete Auflage; Boston et al.: Addison-Wesley, .

Dunkel, J; Eberhart, A.; Fischer, S.; Kleiner, C. ; Koschel, A. (2008) *Systemarchitekturen für Verteilte Anwendungen*; München: Hanser.

Oechsle, R. (2011) *Parallele und verteilte Anwendungen in JAVA*; 3., erweiterte Auflage; München: Carl Hanser.

Oracle (n.n.) *Getting Started Using Java™ RMI*; online: https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html; [letzter Zugriff: 6 I 18]

Schill, A.; Springer, T. (2012) *Verteilte Systeme*; 2. Auflage; Berlin, Heidelberg: Springer Vieweg.

Tanenbaum, A.; van Steen, M. (2002) *Distributed Systems – Principles and Paradigms*, 2nd edition. Upper Saddle River: Pearson/ Prentice Hall, online: http://barbie.uta.edu/~jli/Resources/MapReduce&Hadoop/Distributed%20Systems%20Principles%20and%20Paradigms.pdf [letzter Zugriff: 9 II 18]]

Tanenbaum, A.; van Steen, M. (2008) *Verteilte Systeme – Prinzipien und Paradigmen*; 2., überarbeitete Auflage; München: Pearson Studium.

Weber, M. (1998) *Verteilte Systeme*; Heidelberg, Berlin: Spektrum.

Appendix

Ergänzende Informationen (nicht klausurrelevant):

- ✓ Protokolle für den RPC-Austausch
- ✓ Anforderungs-/Antwortprotokoll: Operationen/Methoden
- ✓ Anforderungs-/Antwortprotokoll: Nachrichtenstruktur
- ✓ Fehler: Bedeutung und Ursachen
- ✓ Fehler: Aufrufsemantik (Kombinationsmöglichkeiten)
- ✓ Marshalling / Unmarshalling
- ✓ Beispiele: Interface Definition Language (IDL)
  - ✓ SUN
  - ✓ CORBA
- ✓ RPC: Asynchrones (nicht-blockierendes) Kommunikationsmodell („deferred synchronous“)

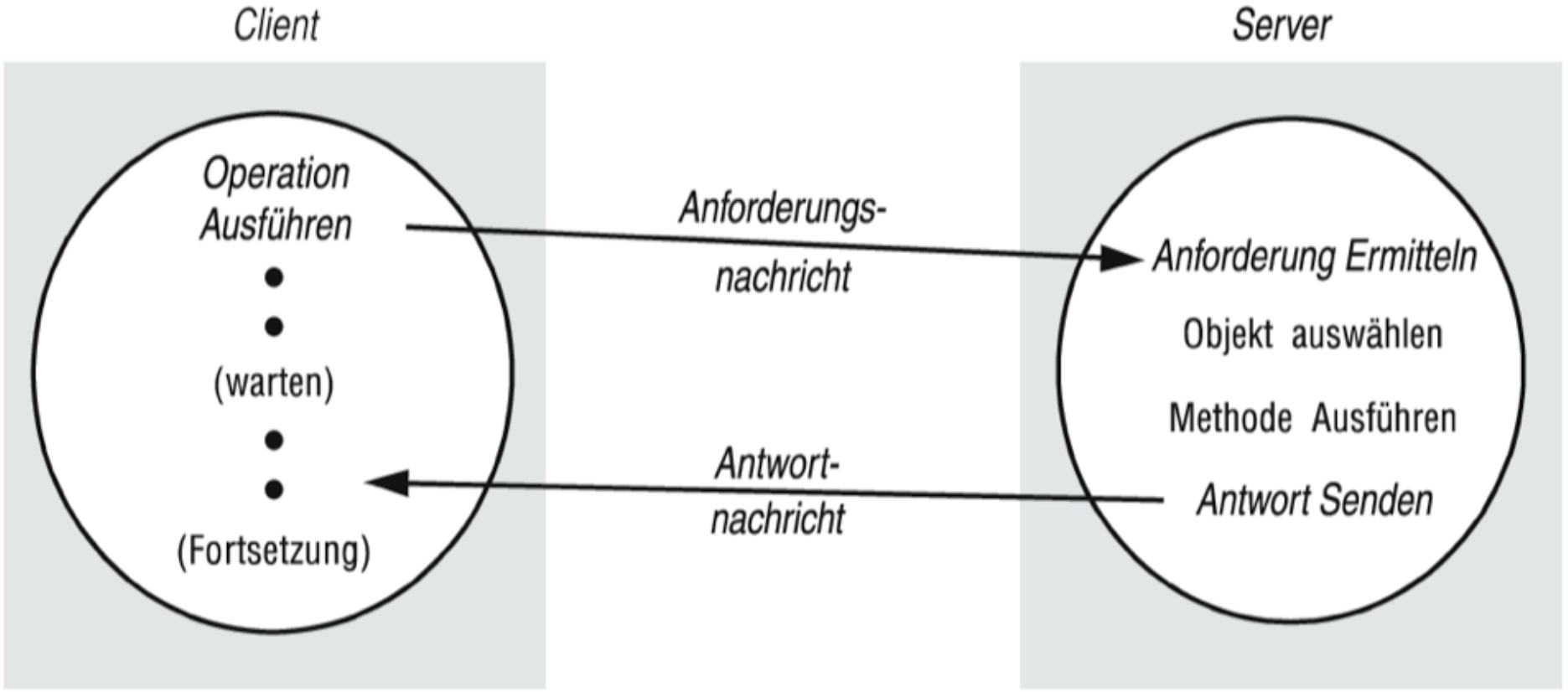
Protokolle für den RPC-Austausch

- R-Protokoll (Request):
  - » Von der Prozedur muss kein Wert zurück gegeben werden
  - » Client benötigt keine Bestätigung über die Ausführung einer Prozedur
- RR-Protokoll (Request-Reply)
- RRA-Protokoll (Request-Reply-Acknowledge Reply)

Name	Nachrichten gesendet von		
	Client	*Server*	*Client*
R	*Anforderung*		
RR	*Anforderung*	*Antwort*	
RRA	*Anforderung*	*Antwort*	*Bestätigungsantwort*

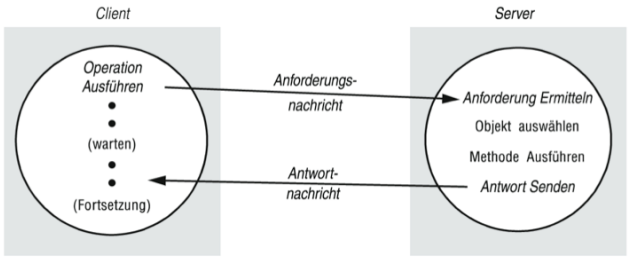
Quelle: Coulouris et al. [2002:183]

Anforderungs-/Antwortprotokoll: Operationen/Methoden [I]



Quelle: Coulouris et al. [2002:179]

Anforderungs-/Antwortprotokoll: Operationen/Methoden [II]



Client

public byte[] doOperation (RemoteObjectRef o, int methodId, byte[] arguments)
sends a request message to the remote object and returns the reply
The arguments specify the remote object, the method to be invoked and the arguments of that method.

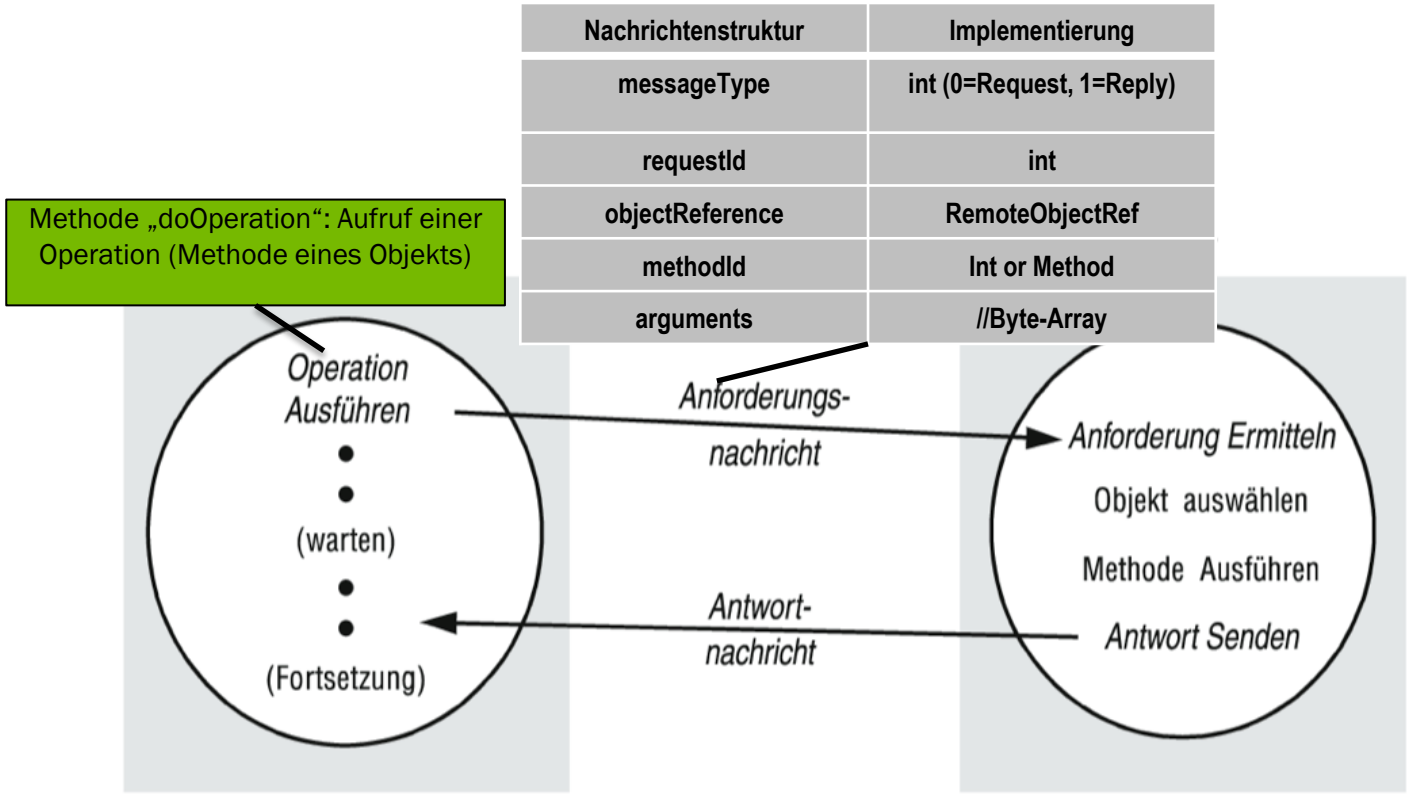
Server

public byte[] getRequest ();
acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);
sends the reply message *reply* to the client at its Internet address and port.

Quelle: Coulouris et al. [2002:181]

Anforderungs-/Antwortprotokoll: Operationen/Methoden [III]

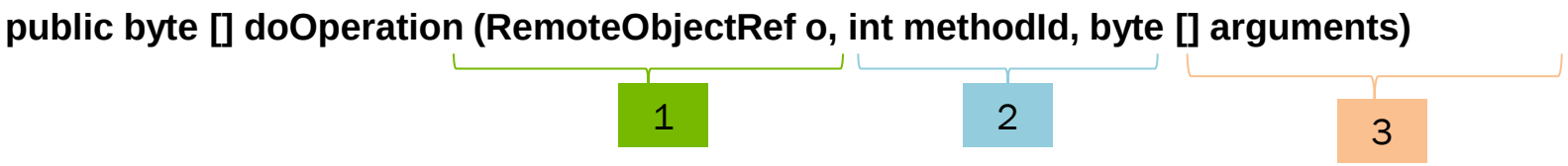


Quelle: Coulouris et al. [2002:179]

Beispiel:

public byte [] doOperation (RemoteObjectRef o, int methodId, byte [] arguments)

Anforderungs-/Antwortprotokoll: Operationen/Methoden [IV]

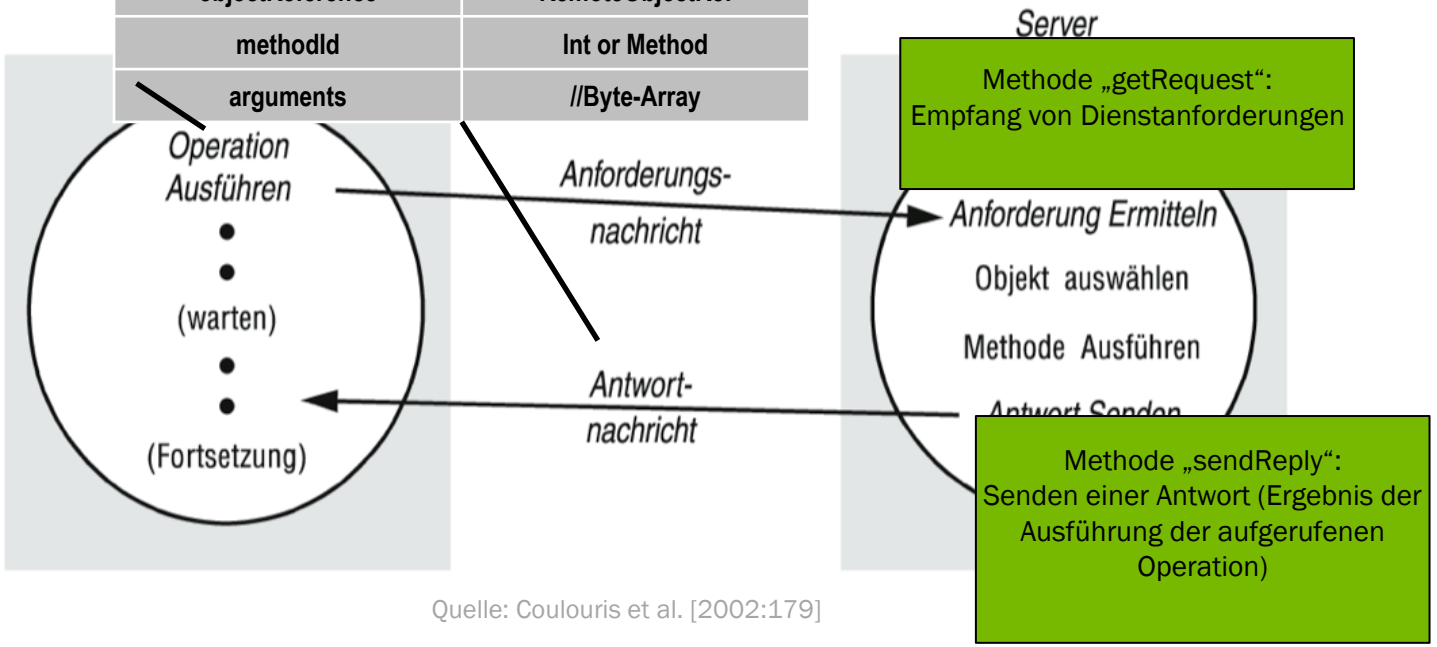


- Wird von Clients verwendet, um entfernte Operationen aufzurufen
- Argumente
  - 1 Angabe des entfernten Objekts (Instanz der Klasse „RemoteObjectRef“)
    - RemoteObjectRef stellt Methoden zur Ermittlung der Internetadresse und des Ports des Servers, auf dem das entfernte Objekt verfügbar ist, bereit
  - 2 Angabe, welche Methode aufgerufen werden soll („methodId“)
  - 3 Wenn nötig: Argumente, die die aufzurufende Methode benötigen könnte (als Byte-Array)

- Ergebnis: Aufruf von doOperation receive zum Erhalt einer Antwortnachricht des Servers
  - » Extraktion des Ergebnisses
  - » Übergabe des Ergebnisses an den Aufrufer (Client-Prozess)
- Der Aufrufer von doOperation wird bis zum Erhalt einer Antwortnachricht blockiert

Anforderungs-/Antwortprotokoll: Operationen/Methoden [V]

Nachrichtenstruktur	Implementierung
messageType	int (0=Request, 1=Reply)
requestId	int
objectReference	RemoteObjectRef
methodId	Int or Method
arguments	//Byte-Array



Quelle: Coulouris et al. [2002:179]

Beispiel:

public byte [] doOperation (RemoteObjectRef o, int methodId, byte [] arguments)

Anforderungs-/Antwortprotokoll: Nachrichtenstruktur [I]

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
objectReference	*RemoteObjectRef*
methodId	*int or Method*
arguments	*// Byte-Array*

Quelle: Coulouris et al. [2002:181]

Anforderungs-/Antwortprotokoll: Nachrichtenstruktur [II]

Nachrichtenstruktur	Implementierung
messageType	int
requestId	int
objectReference	RemoteObjectRef
methodId	Int or Method
arguments	//Byte-Array

„messageType“

- ✓ 0 (Anforderung, Request)

  oder

- ✓ 1 (Antwort, Reply)

Anforderungs-/Antwortprotokoll: Nachrichtenstruktur [III]

Nachrichtenstruktur	Implementierung
messageType	int
requestId	int
objectReference	RemoteObjectRef
methodId	Int or Method
arguments	//Byte-Array

„RequestId“

- Wird vom Client bei Aufruf der Methode „doOperation“ erzeugt
- Server kopiert diese in seine Antwortnachricht mit messageType = 1 (Reply)
- Client überprüft die requestId der Antwortnachricht durch doOperation (handelt es sich um das Ergebnis des Operationsaufrufs oder um das Ergebnis eines verzögerten, früheren Aufrufs)

Anforderungs-/Antwortprotokoll: Nachrichtenstruktur [IV]

Nachrichtenstruktur	Implementierung
messageType	int
requestId	int
objectReference	RemoteObjectRef

„objectReference“

- Entfernter Objektverweis
- Marshalling muss vorher erfolgt sein, um folgendes Format zu erhalten:

Abbildung 4.10 Darstellung eines entfernten Objektverweises

32 Bit	*32 Bit*	*32 Bit*	*32 Bit*	
Internetadresse	Port-Nummer	Zeit	Objektnummer	Schnittstelle des entfernten Objekts

Quelle: Coulouris et al. [2002:178]

Anforderungs-/Antwortprotokoll: Nachrichtenstruktur [V]

Nachrichtenstruktur	Implementierung
messageType	int
requestId	int
objectReference	RemoteObjectRef
methodId	Int or Method
arguments	//Byte-Array

„methodId“

- Nummerierung der Methoden einer Schnittstelle (1,2,3,...)

Fehler: Bedeutung und Ursachen

vgl. Weber [1998:71f.]; Bengel et al. [2008:111]

- Bei auftragsorientierter Kommunikation sind zwei Partner-Prozesse involviert, d.h. ein fehlerfreier Ablauf eines Prozesses hängt nicht nur von seiner eigenen Umgebung ab, sondern auch vom Verhalten des anderen Prozesses und der zum Nachrichtenaustausch verwendeten Kommunikationseinheiten.
- Verlässliche Semantiken erfordern eine Untersuchung von Fehlerursachen und Maßnahmen, wie diesen potenziellen Fehlern begegnet werden kann.
- Ursachen: Interaktionsfehler
  - Anforderung geht verloren
  - Anforderung erfährt Verzögerung
  - Rückantwort geht verloren
  - Rückantwort erfährt Verzögerung
  - Server oder Client sind zwischenzeitlich abgestürzt

Fehler: Aufrufsemantik (Kombinationsmöglichkeiten)

Fehlertoleranzmaßnahmen			Aufrufsemantik
Anforderungsnachricht wiederholt übertragen	*Duplikatfilterung*	*Wiederholte Ausführung der Prozedur oder wiederholte Übertragung der Antwort*	
Nein	Trifft nicht zu	Trifft nicht zu	Vielleicht ...maybe
Ja	Nein	Prozedur wiederholt ausführen	Mindestens einmal ...at-least-once Für idempotente Operationen Bsp.: SUN RPC
Ja	Ja	Antwort erneut übertragen	Höchstens einmal ...at-most-once (JAVA RMI, CORBA)

Quelle: Coulouris et al. [2002:214]

Marshalling/Unmarshalling

- Bezeichnet die Vorgehensweise des Umwandelns von Daten in ein für die Übertragung geeignetes Format, z.B.:
  - » External Data Representation (XDR)
  - » XDR ist spezifiziert in: RFC 1014, RFC 1832, RFC 4506
- Übersetzung strukturierter Datenelemente für die externe Datendarstellung
- Un-Marshalling: Umwandeln der externen Datendarstellung in elementare Werte und Datenstrukturen

✓ Marshalling/Unmarshalling werden normalerweise durch die spezifizierte / gewählte **Middleware** automatisch durchgeführt (z.b. CORBA CDR (Common Data Representation))

Beispiele: Interface Definition Language (IDL)

SUN XDR/RPC

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
      int length;
      char buffer[MAX];
};
struct writeargs {
      FileIdentifier f;
      FilePointer position;
      Data data;
};
struct readargs {
      FileIdentifier f;
      FilePointer position;
      Length length;
};

program FILEREADWRITE {
 version VERSION {
      void WRITE(writeargs)=1;
      Data READ(readargs)=2;
      }=2;
} = 9999;
```

CORBA IDL

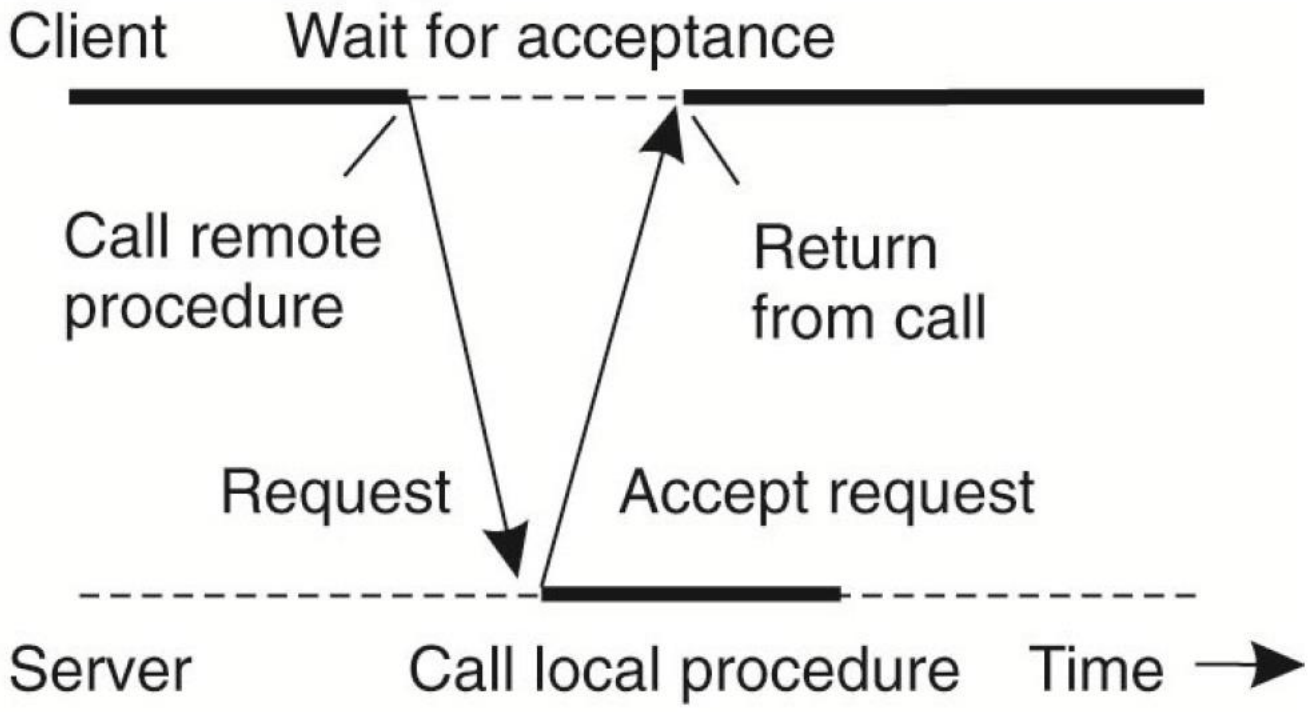
```
struct Person{
      string name;
      string place;
      long year;
};
interface PersonList{
      readonly attribute string listname;
      void addPerson(in Person p);
      void getPerson(in string name, out Person
      p);
      long number();
};
```

Quelle: Coulouris et al. [2002: 225]

Remote Procedure Call (RPC)

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

- ✓ Asynchrones (nicht-blockierendes) Kommunikationsmodell

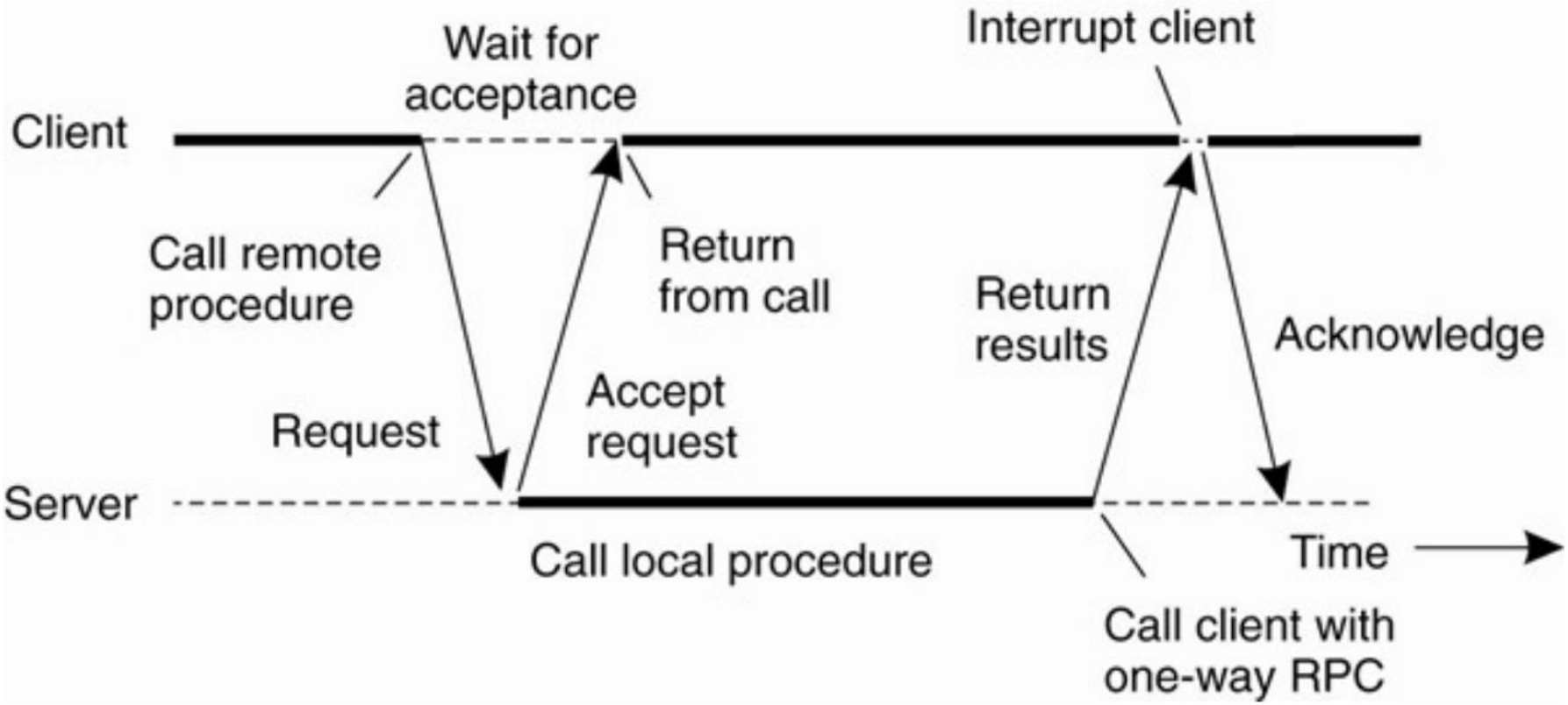


Quelle: Tanenbaum & van Steen. [2002:135]

Remote Procedure Call (RPC)

vgl. Bengel et al. [2008:276ff.], Tanenbaum & van Steen [2008:152ff.]

- ✓ Asynchrones (nicht-blockierendes) Kommunikationsmodell mit zwei RPC-Aufrufen



Quelle: Tanenbaum & van Steen. [2002:135]