## Verteilter gemeinsamer Speicher



- Multiprozessor
  - eng gekoppelte Prozessoren
  - gemeinsamer und kohährenter Speicher
- Multicomputer und Verteilte Systeme
  - lose gekoppelte Prozessoren
  - nur privater Speicher verfügbar
- verteilter gemeinsamer Speicher
  - Illusion eines gemeinsamen Speichers



## Verteilter gemeinsamer Speicher (2)



- Mögliche Realisierung
  - seitenbasierte Speicherverwaltung
  - Seitenadressierung, unterstützt durch MMU
  - entspricht logischem/virtuellem Speicher
  - ein virtueller Adressraum über alle beteiligten Rechner
  - eine Seite residiert zu einem Zeitpunkt immer nur auf genau einem Rechner
- Lokaler Zugriff
  - Seite ist in der lokalen Seitentabelle eingetragen, Präsenzbit ist gesetzt
    $\Rightarrow$ Zugriff ist möglich
  - Zugriff erfolgt auf lokalen Speicher



## Verteilter gemeinsamer Speicher (3)



- Entfernter Zugriff
  - Seite ist in lokaler Seitentabelle nicht eingetragen, Präsenzbit ist nicht gesetzt
  - lokaler Zugriff löste einen Seitenfehler (page fault) aus
    $\Rightarrow$ Unterbrechungsbehandlung durch das Betriebssystem
  - Betriebssystem holt die Seite von dem entfernten Rechner (hierbei wird die Seite dort ausgetragen!)
  - Seite wird in lokale Seitentabelle eingetragen, Präsenzbit wird gesetzt
  - Speicherzugriff wird wiederholt
- Kohärenter Speicher
  (Lese-Operation liefert immer den zuletzt geschriebenen Wert)
  - falls Hole- und Weitergabe-Operationen für Seiten atomar
  - falls keine Ausfällte auftreten



## Verteilter gemeinsamer Speicher (4)



- Nebenläufige Zugriffe auf eine Seite sind sehr ineffizient
  - Seitenflattern zwischen Rechnern
- False-Sharing
  - zwei Datenobjekte liegen in der selben Seite, werden aber von unterschiedlichen Programmbereichen benutzt
  - die unterschiedlichen Programmbereiche werden parallel auf verschiedenen Rechnern ausgeführt
  - Ergebnis: gegenseitiges „Stehlen“ der Seite bei Datenzugriffen
- Maßnahmen zur Effizienzsteigerung erforderlich
  - ausgefeiltere Kohärenzprotokolle
  - paralleles Lesen erlauben
  - Abschwächung der Speicherkonsistenz-Anforderungen

# Verteilter gemeinsamer Speicher (5)



- Resumee
  - Verteilter gemeinsamer Speicher nur bei sehr enger Rechnerkopplung sinnvoll
    - spezielle Hardware-Unterstützung
    - kurze Latenzen
    - schnelle Datenübertragung

    ⇒ Höchstleistungsrechner
  - In lose gekoppelten Systemen ist reine Nachrichtenübertragung vorzuziehen
    - Kommunikation nicht über gemeinsamen Speicher, sondern über Prozeduraufrufe



# Nachrichten zusammenstellen / auspacken

- *marshalling*

  *to arrange (troops, things, ideas, etc.) in order; array; dispose*
  - die einzelnen Datenelemente in einen Nachrichtenpuffer gepackt anordnen:
    1. für die **Serialisierung** verstreut vorliegender Daten sorgen
    2. die vereinbarte **Repräsentation** (Typ) der Daten gewährleisten
  - je nach Herangehensweise sind Referenzen aufzulösen und umzuwandeln
    - wenn die referenzierten Strukturen *call-by-value* zu übertragen sind
  - die so zusammengestellte Nachricht geht per IPC an den Empfangsprozess
- *unmarshalling*
  - die *inverse Funktion* auf Empfangsseite
  - Nachricht entpacken und die ursprünglichen Datenelemente wieder herstellen
  - hierbei ggf. eine andersartige Repräsentation auf der Zielseite berücksichtigen



# Serialisierung/Deserialisierung

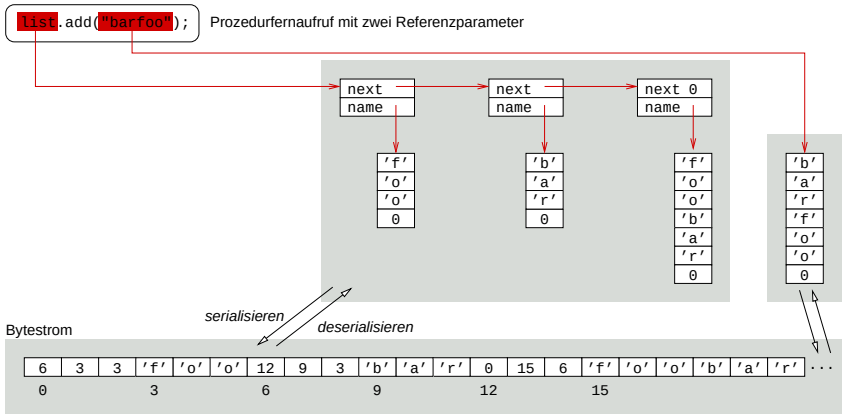
- geläufig ist, **verzeigerte Datenstrukturen** *call-by-value* (*-result*) zu übergeben
  - zum Marshalling wird eine Beschreibung des logischen Aufbaus strukturierter/dynamischer Daten benötigt (= Typbeschreibung)
  - alle Referenzen werden aufgelöst und die referenzierten Objekte kopiert
  - nach dem Empfang werden die Datenstrukturen wieder rekonstruiert
- zwei Sorten von Zeigern sind dabei zu unterscheiden:

  innere Referenzen: die Verkettungszeiger rekursiver Datenstrukturen
  - sind vergleichsweise unproblematisch: „logische Zeiger“ vergeben

  äußere Referenzen: Zeiger von außen hinein auf einzelne Verbundelemente
  - die relative Position der Verbundelemente kann sich ändern: Relokation
- der Aufwand kann je nach Art/Aufbau der Datenstrukturen beträchtlich sein



# Packen/Entpacken strukturierter Daten




## Objektorientierung „erschwert“ Automatisierung

- Referenzen vom Typ einer Oberklasse zeigen ggf. auf Instanzen von Unterklassen
  - eine *abstrakte Oberklasse* sagt wenig/nichts aus über die Objektstruktur
    ⇒ in der Fernaufrufschnittstelle ist die konkrete Ausprägung des zu übertragenden Objekts unbekannt
  - erst die spezialisierende Unterklasse enthält die erforderlichen Strukturinformationen
- der *tatsächliche Typ* des Parameters ist erst zur Laufzeit bekannt
  - automatische Generierung des Stubs mit der Marshalling-Funktion ist zur Übersetzungszeit nicht möglich
  - statt dessen „spezialisierte Zusammenstellung“ der Fernaufrufnachrichten
    - erfordert eine vollständige Analyse des (zu verteilenden) Quellprogramms
  - Alternative: Analyse der Parametertypen zum Zeitpunkt des Aufrufs — *Reflection*



## Datenrepräsentation

Heterogenität – die einzelnen in Nachrichten übertragenen Elemente können Werte unterschiedlichster (elementarer) Datentypen repräsentieren.

- *Speicherung* wie auch *Darstellung* von Instanzen dieser Typen ist nicht in allen Rechnern identisch:
  - natürliche/ganze Zahlen: vorzeichenbehaftet, {1,2}er-Komplement
  - Fließkommazahlen: Basis, Mantisse, Exponent
  - Zeichensätze: ISO-8859-Familie (ASCII), BCD, EBCDIC, Unicode
  - Speicherreihenfolge: *big endian* vs. *little endian*

⟹ Daten sind ggf. zu konvertieren, damit kooperierende Prozesse funktionieren!



## Konvertierungsoptionen

beidseitig *external data representation* (XDR)
- zu sendende Daten in eine **kanonische Darstellung** umkodieren *und*
- empfangene („kanonische“) Daten in die lokale Darstellung umkodieren
- Problem: nutzloser Mehraufwand im Falle „gleichartiger“ Rechner

sendeseitig „*sender makes it right*“
- zu sendende Daten in die empfangsseitige Darstellung ggf. umkodieren
- Problem: Mehrteilnehmerkommunikation, Weiterleiten von Nachrichten

empfangsseitig „*receiver makes it right*“
- empfangene Daten in die lokale Darstellung ggf. umkodieren → *endian tag*



## *External Data Representation* — XDR

entwickelt von Sun Microsystems, *RFC 1014*, 1987

- sprachbasierter Standard zur Beschreibung und Kodierung von Daten
  - der ISO/OSI **Präsentationsschicht** (*presentation layer*, 6) zugeordnet
  - **implizite Typung**, nicht explizit wie bei ASN.1
    - die Typen der einzelnen Daten ind er nachricht ergeben sich implizit aus dem Typ der aufgerufenen Schnittstelle
    - bei ASN.1 werden sie explizit in der Nachricht mitgeschickt
  - Annahme: Bytes bzw. Oktets (d.h. Einheiten von 8 Bits) sind portabel
- Daten werden als „Vielfaches von vier Bytes“ (32 Bits) repräsentiert (*Tradeoff* „Vier“ — Groß genug als effiziente Lösung für die meisten Maschinen, mit Ausnahme von 64-Bit Architekturen, und klein genug als vertretbarer Mehraufwand zur Repräsentation der kodierten Daten.)
  - Füllbytes (0 – 3) ergänzen den Datenstrom immer zum Vielfachen von 4
- die Reihenfolge (der „Byte-Sex“) ist *big endian*

## XDR „*Considered Harmful*“?

- Virtualisierung:
  Jede Maschine, deren physikalische Darstellung mit der durch XDR vorgegebenen virtuellen übereinstimmt, wird effizienter arbeiten als jene, bei der die physikalische von der virtuellen Darstellungsform stark abweicht:
  - Speicherreihenfolge („Byte-Sex“)

| | endian | | endian | | |
|---|---|---|---|---|---|
| Sun | *big* | ↔ | *big* | m68K | „optimal“ |
| IBM 370 | *big* | ↔ | *little* | Z80 | |
| Alpha | *little* | ↔ | *big* | R10000 | |
| VAX | *little* | ↔ | *little* | x86 | „suboptimal“ |

  - Verschnitt („interne Fragmentierung“)



## Sprachintegration von Fernaufrufen

- Fernaufrufsysteme lassen sich in zwei Hauptkategorien einteilen:
  1. in einer Programmiersprache **integriertes Konzept** . . . . . Argus
     - internes Wissen über Datentyp- und Laufzeitmodell ist verfügbar
     - der Übersetzer agiert gleichzeitig als *Stub-Generator* (*stub generator*)
     - umfassende Analysen, Optimierungen und Automatismen werden möglich
  2. von einer Programmiersprache **separiertes Konzept** . . . . . CORBA
     - das Schnittstellenverhalten entfernter Prozeduren wird in einer eigenen „*Interface Definition Language*“ explizit beschrieben (IDL)
     - fehlendes internes Wissen schränkt Analysen, Optimierungen etc. ein
  3. **teilweise integriertes Konzept** . . . . . Java RMI
     - der Compiler kennt Konzepte des Fernaufrufs (Remote-Schnittstellen, etc.) nicht
     - in den Java-Standards sind die Konzepte definiert
     - Unterstützung für RMI integraler Bestandteil der Laufzeitumgebung
- nur das integrierte Konzept ($\rightarrow$ ) definiert eine einheitliche Semantik



## Entlastung von Routineaufgaben

- die Stubs sorgen für eine **lose Kopplung** zweier Ausführungsumgebungen:

  *client stub*: den Ort des Dienstabieters (beim Namensdienst) erfragen
  1. *Marshalling* und versenden der Anforderungsnachricht

  ⋮ die Durchführung der angeforderten Operation abwarten
  2. empfangen und *Unmarshalling* der Antwortnachricht

  *server stub*: den Ort des Dienstabieters (dem Namensdienst) angeben
  1. empfangen und *Unmarshalling* der Anforderungsnachricht
  2. durchführen der angeforderten Operation (lokaler Aufruf)
  3. *Marshalling* und versenden der Antwortnachricht
- **Stub-Generatoren** erzeugen die dafür notwendigen Programmsequenzen



## Transparenz von Fernaufrufen

- Verteilungstransparenz ist nur bedingt erreichbar, trotz integriertem Konzept:
  - syntaktische Unterschiede (in der Schnittstelle) werden vermieden
  - Parameterübergabe, *Marshalling* und *Unmarshalling* wird verborgen
  - Nachrichtenpuffer und Fäden werden erzeugt, verwaltet und entsorgt
  - Stubs werden automatisch erzeugt — was ist denn sonst noch nötig?
- Fernaufrufe sind fehleranfälliger als konventionelle lokale Prozeduraufrufe

  entfernt ein Netzwerk, ein anderer Rechner und ein anderer Prozess
  ¬ entfernt kein Netzwerk, kein anderer Rechner und kein anderer Prozess
- verteilte Systeme unterliegen einem radikal anderem **Fehlermodell**

## Zustellungsgarantien

- *request-reply*-Protokolle eröffnen Optionen für Fehlertoleranzmaßnahmen:
  1. Wiederholung der Anforderungsnachricht . . . . . *request retry*
     - bis die Antwort eintrifft oder ein Anbieterausfall angenommen wird
  2. Filterung der Anforderungsduplikate . . . . . *duplicate suppression*
     - wenn die Anforderung empfangen wurde und noch in Arbeit ist
  3. Wiederholung der Antwortnachricht . . . . . *reply retry*
     - bis die nächste Anforderung oder eine Bestätigung eintrifft
- Kombinationen dieser Optionen begründen verschiedene Aufrufsemantiken



## Fehlertoleranzmaßnahme vs. Aufrufsemantik

| **Option** | | | **Semantik** | |
|---|---|---|---|---|
| *request retry* | *dupl. supr.* | *re-execute reply retry* | | |
| Nein | — | — | „kann sein“ | *maybe* |
| Ja | Nein | *re-execute* | „wenigstens einmal“ | *at-least-once* |
| Ja | Ja | *reply retry* | „höchstens einmal“ | *at-most-once* |



## Aufrufsemantiken (1)

- *maybe*: kann sein, der Aufruf wurde ausgeführt oder auch nicht
  - der Klient hat im Fehlerfall keine Gewissheit über die korrekte Ausführung
- *at-least-once*: mindestens einmal, die mehrfache Ausführung ist möglich
  - der Klient erwartet die Antwort innerhalb einer vorgegebenen Zeitspanne
    - jeder Fernaufruf wird mit einem *Timeout* versehen
    - nach der dadurch definierten Pause erfolgt die Aufrufwiederholung
    - die Anzahl der Wiederholungen ist (üblicherweise) begrenzt
    - mit Erreichen der max. Anzahl tritt eine *Ausnahmesitutation* ein
  - der Anbieter muss **idempotente Operationen** exportieren



## Aufrufsemantiken (2)

*at-most-once*: höchstens einmal, mehrfache Ausführung ist ausgeschlossen

- vergleichsweise aufwändiges, speicherintensives Verfahren:
  - jedem neuen Aufruf wird eine eindeutige *Aufrufkennung* gegeben
  - Wiederholungen kommen mit derselben Kennung und werden verworfen
    - Möglichkeit der Ausnahmesituation entsprechend *at-least-once*
  - Antworten werden gespeichert und bei Wiederholungen zurückgeliefert
  - ein neuer Aufruf ist Anlass, gespeicherte Antworten zu entsorgen
    (Ansonsten muss der Anbieter hin und wieder beim Klienten nachfragen, ob dieser noch „lebt“, um so ein Kriterium für die Entsorgung gespeicherter Antwortnachrichten zu gewinnen.)
- die Ausführung erfolgt nur, wenn keine Rechnerausfälle vorliegen

## Aufrufsemantiken (3)

- *last-of-many*: akzeptiert nur die Antwort zum jüngst zurückliegenden Aufruf
  - jeder Aufruf, auch der wiederholte, erhält eine eindeutige Kennung
  - jede Antwort trägt die Kennung des zugehörigen Aufrufs
    - der Klient verwirft Antworten mit nicht mehr aktueller Aufrufkennung
    - eine Variante von *at-least-once*, falls keine Idempotenz vorliegt
  - die wiederholte Ausführung der Operationen ist ggf. weiterhin problematisch
  - zumindest können die Klienten aber mit „aktuellen“ Daten weiterarbeiten

  ( Im Gegensatz dazu liefert *at-most-once* ggf. „veraltete“ Daten, die zur ersten Ausführung wiederholt abgesetzter Aufrufe korrespondieren. Dieser Fall kann vorliegen, wenn die vom Anbieter verwalteten Daten von mehreren Klienten be- und verarbeitet werden (*information sharing*).)



## Aufrufsemantiken (4)

- *exactly-once*: genau einmal, entspricht der Semantik lokaler Aufrufe
  - erfordert Transaktionskonzepte mit Wiederanlauf von Komponenten
    - gibt damit (eine gewisse) Garantie bei Systemfehlern bzw. -ausfällen
  - nach Wiederanlauf sind Aussagen zur Operationsdurchführung „unscharf“:

    *Imagine, for example, a chocolate factory, in which vats of liquid chocolate are filled by having a computer set a bit in some device register to open a valve. After recovering from a crash, there is no way for the chocolate server to see if the crash happened one microsecond before or after the bit was set. (Tanenbaum, Computer Networks, 1989)*

  - wünschenswerte, ideale Semantik, die in „Reinform“ jedoch unerreichbar ist



## Idempotente Operationen

- Idempotenz „idem“ (*lat.*) dasselbe; wenn die wiederholte Ausführung derselben Operation (mit denselben Parameterwerten) durch den Dienstanbieter immer den Effekt einer einmaligen Ausführung besitzt
  - wiederholte Ausführungen sind möglich im Falle ungefilterter Duplikate
    - kritisch sind z.B. Schreiboperationen auf Dateien (`write(2)`)
    - ebenso Operationen, die eine Folge (Liste) um Elemente erweitern[1]
  - ein **zustandsfreier Dienstanbieter** (*stateless server*) ist gefordert
    - bekannter Vertreter ist das Sun *Network File System* (NFS)
  - bei zustandsbehafteten Dienstanbietern ist Duplikatelimination notwendig



## Fehlersemantiken

| **Aufrufsemantik** | **Fehlerart** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | fehlerfreier Ablauf | | Verlust von Nachrichten | | zusätzlicher Ausfall von Anbieter | | Klient | |
| | Aus. | Erg. | Aus. | Erg. | Aus. | Erg. | Aus. | Erg. |
| *maybe* | 1 | 1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 |
| *at-least-once* | 1 | 1 | $\geq 1$ | $\geq 1$ | $\geq 0$ | $\geq 0$ | $\geq 0$ | 0 |
| *at-most-once* | 1 | 1 | 1 | 1 | 0/1 | 0/1 | 0/1 | 0 |
| *exactly-once* | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Aus.= Ausführung, Erg.= Ergebnis; die jeweilige Anzahl ist angegeben

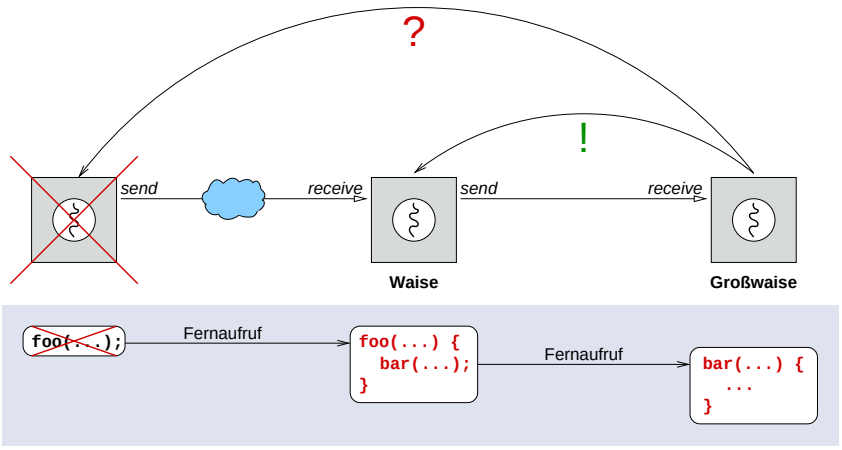
# Verwaister Aufruf (1)

- *Orphan* ein Klient, der den Aufruf (z.B. wegen eines ggf. zu kurzen *Timeout*) abgebrochen hat und nicht mehr am Ergebnis interessiert ist oder der mittlerweile überhaupt nicht mehr zur Verfügung steht

- Maßnahmen, um unnötige Arbeit des Dienstanbieters zu vermeiden:
  1. zusätzliche Zeitüberwachung des Klienten durch den Dienstanbieter
     - die *Timeout*-Wahl ist klientenabhängig ($T_S \gg T_C$)
  2. pro Klienten einen Ausfallzähler beim Dienstanbieter verwalten
     - Abbruch aller alten Aufrufe nach Ausfall und Wiederanlauf des Klienten
  3. zusätzliche direkte Statusanfragen an den Klienten senden
     - schnelle Waisenerkennung, sofern der Ausfall diagnostizierbar ist
- besondere Schwierigkeiten bereiten „Fernaufrufketten“: **Transitivität**



# Verwaister Aufruf (2)






# Waisenbehandlung (1)

- *extermination* (Vertilgung, Ausrottung, Wegschaffung)
  - nach Wiederanlauf wird (beim Klienten) auf ausstehende Fernaufrufe geprüft
    - den betreffenden Anbietern gehen sodann Abbruchanforderungen zu
    - rekursives Vorgehen, wegen der Möglichkeit von „Großwaisen“
  - Klientenstümpfe müssen „Buch“ führen über alle Fernaufrufe
    - *Log* anlegen vor der Anforderung und zerstören nach Empfang der Antwort
- *expiration* (Verscheiden, Ablauf)
  - der Anbieter gibt dem (Fern-) Aufruf ein *Zeitquantum* zur Ausführung
    - mit Ablauf erbittet der Anbieter ein weiteres Zeitquantum vom Klienten
    - im Fehlerfall bricht der Anbieter den Aufruf ab bzw. terminiert er
  - nach Wiederanlauf ist der erste Fernaufruf um ein Zeitquantum zu verzögern
  - *Leasing*: zeitlich begrenzt gültige Referenzen - erstmalig realisiert in *Jini*



# Waisenbehandlung (2)

- *reincarnation* (Wiederverkörperung, Wiedergeburt)
  - bei Wiederanlauf eines Klients wird eine neue *Epoche* eröffnet
    (Epochen entstehen dadurch, dass die Zeit in sequentiell nummerierte Abschnitte aufgeteilt wird.)
    - der Beginn einer Epoche wird allen Maschinen mitgeteilt (*broadcast*)
    - auf den Maschinen werden daraufhin alle Anbieter (Prozesse) terminiert
  - jede Anforderungs- und Antwortnachricht enthält eine Epochenkennung
    - damit können unerwartete Antworten von Waisen herausgefiltert werden
- *gentle reincarnation* (sanfte Wiedergeburt)
  - zum Epochenbeginn versucht der Anbieter „seinen“ Klienten zu lokalisieren
  - ist der Klient nicht lokalisierbar, werden die Prozesse terminiert

## Ausnahmebehandlung — *Exception Handling*

- volle Verteilungstransparenz erreichen zu können, ist reine Illusion
  - auch *exactly-once* kann nur die erkennbaren Fehler maskieren
  - in den anderen Fällen ist die Rückkehr vom Aufruf keinesfalls garantiert ( Es ist der Fernaufrufmechanismus selbst, der einen Klienten z.B. bleibend verklemmen kann. Sicherlich kann auch die Implementierung des per Fernaufruf in Anspruch genommenen Dienstes Verklemmungsursache sein, was aber hier nicht zur Debatte steht.)
- die Stubs (auf beiden Seiten) sollten Ausnahmesituationen anzeigen
  - Ausnahmen der Anbieterseite werden als Rückruf zum Klienten propagiert
  - Ausnahmen der Klientenseite werden konventionellerweise „hochgereicht“
- die Anwendungen sollten Maßnahmen zur Fehlertoleranz beinhalten



## Zusammenfassung

- Fernaufrufe sind konventionellen Prozeduraufrufen nur ähnlich, nicht gleich:
  - Parameter{arten, Übergabe}, Gültigkeitsbereiche und Speicheradressen
  - Fehlermodell, Zustellungsgarantien, Aufrufsemantiken, verwaiste Aufrufe
  - die Latenz entfernter Aufrufe ist um Größenordnungen höher: *Promise*
- Verteilungstransparenz kann nicht wirklich (durchgängig) erzielt werden
  - *exactly-once* ist nicht (bzw. nur mit Abstrichen) zu verwirklichen
  - Ausnahmen sind zu behandeln, die nur in verteilten Systemen auftreten
  - Fernaufrufe machen fehlertolerante Anwendungssoftware keinesfalls obsolet
- der Unterschied zu konventionellen Aufrufen ist (doch) explizit zu machen