

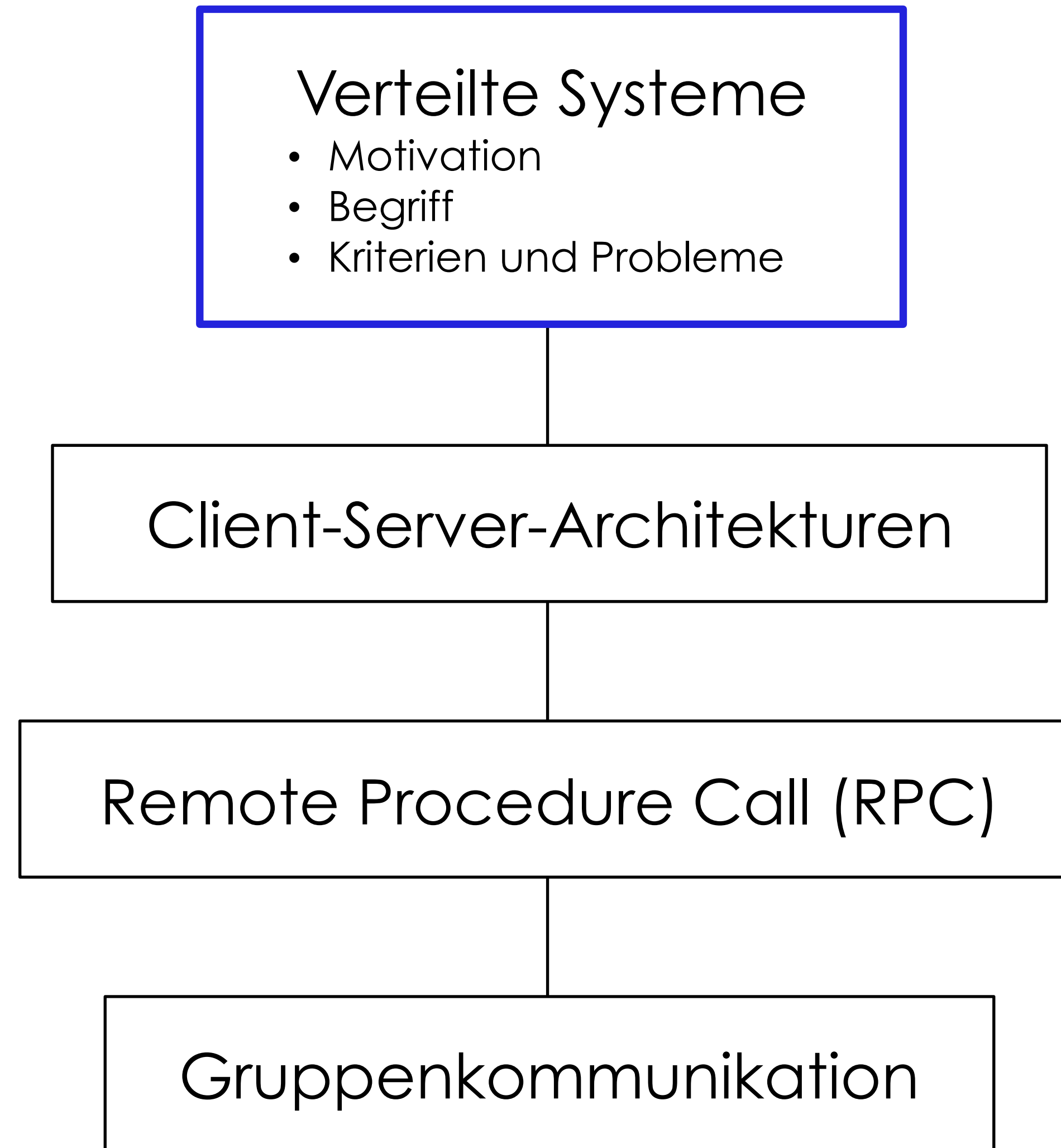


**TECHNISCHE  
UNIVERSITÄT  
DRESDEN**

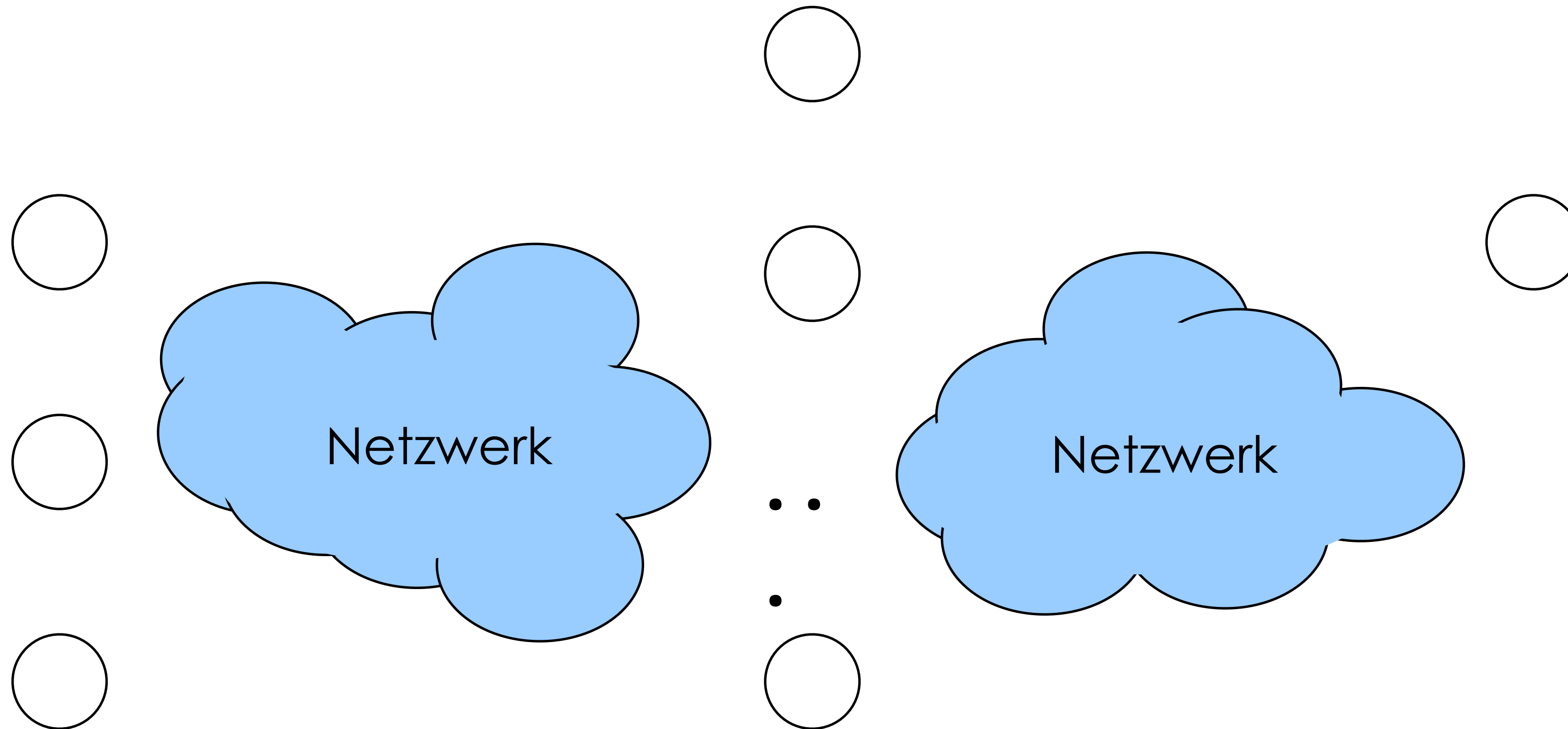
Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

# **CLIENT-SERVER- ARCHITEKTUREN**

**MICHAEL ROITZSCH**

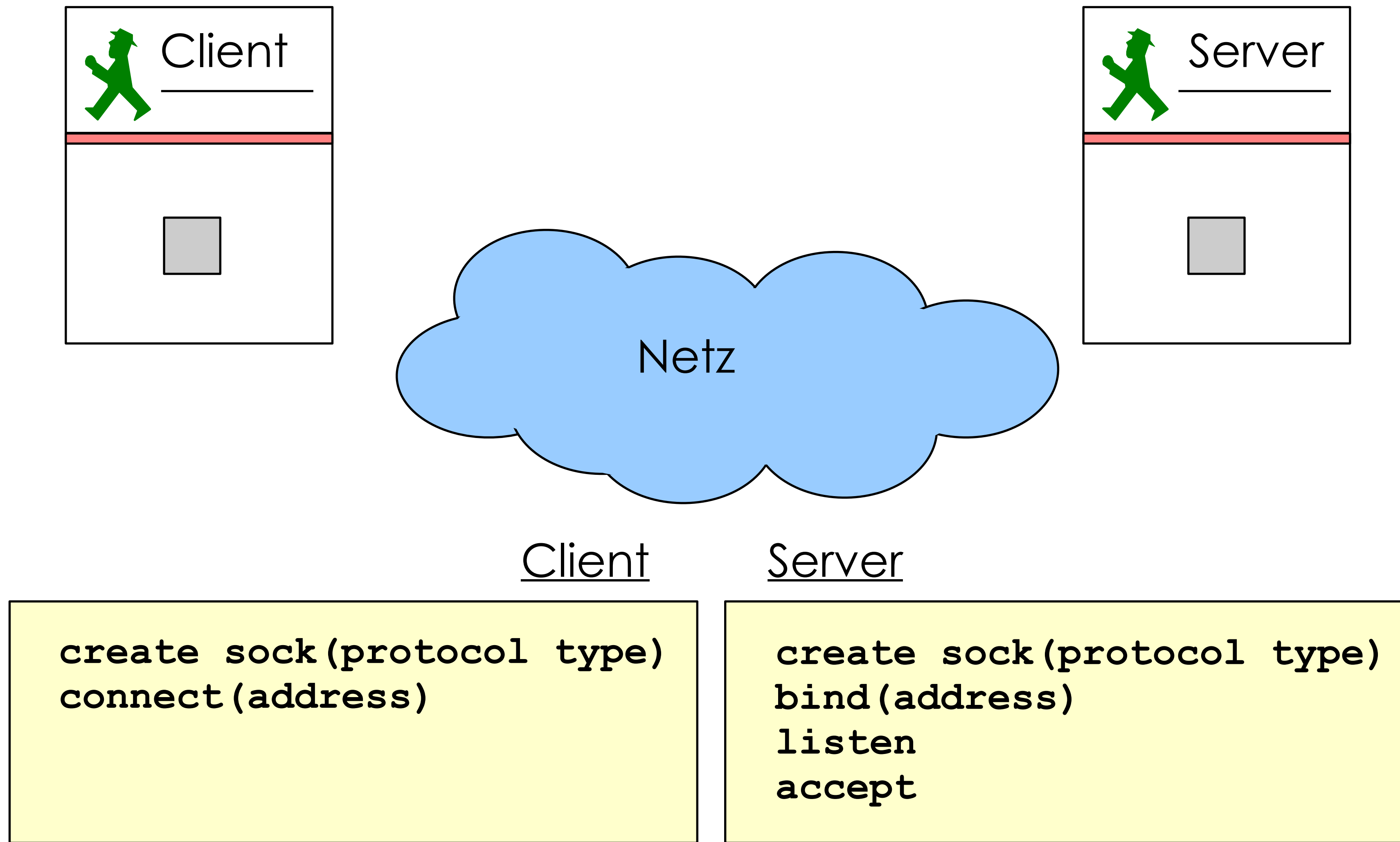


# Ausgangspunkt: Netzwerke

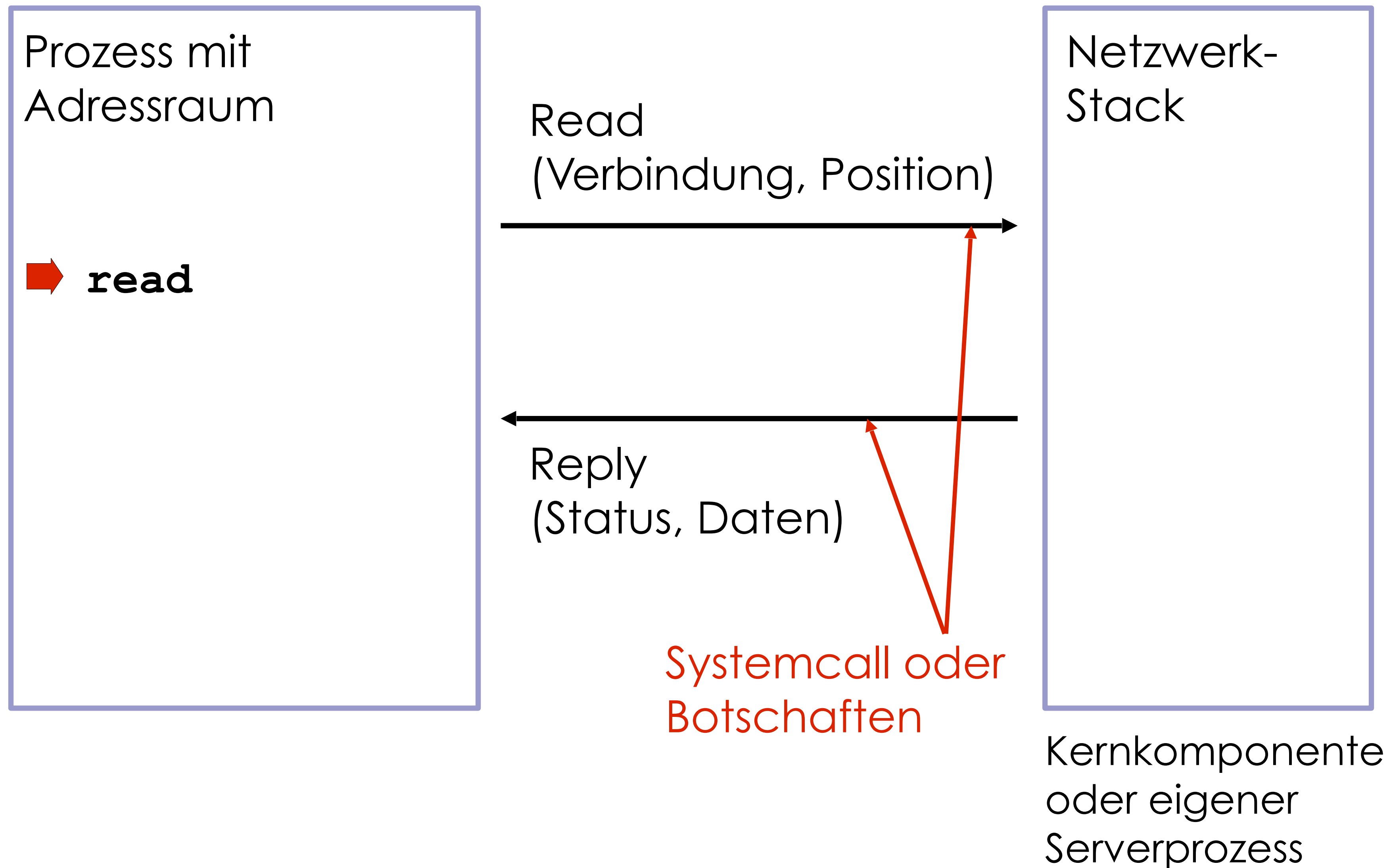


**Zustellen von  
Netzwerkpaketen**

# Sockets



# Zugriff mittels Kopieren (read/write)



# Verteilte Systeme

---

Ansammlung autonomer Rechner, die dem Benutzer als ein System präsentiert werden.

— *Tanenbaum*

Ansammlung autonomer Rechner, die über ein Netzwerk verbunden sind und mit verteilter Systemsoftware ausgestattet sind.

— *Colouris*

You know you have one when the crash of computer you have never heard of stops you from getting any work done

— *Leslie Lamport*

# Verteilte Systeme: Begriff

---

Ein verteiltes System liegt vor, wenn mehrere Rechner eine gemeinsame Aufgabe übernehmen.

## **Vier Problembereiche (nach Michael D. Schroeder)**

- voneinander unabhängige Ausfälle
- unzuverlässige Kommunikation
- unsichere Kommunikation: Mithören, Manipulieren
- teure Kommunikation: Bandbreite, Latenz, Kosten

# Verteilte Systeme: Kriterien

---

## **Beurteilungskriterien**

- **Transparenz:** Orts-, Zugriffs-, Namens-Transparenz, Replikations-, Migrations-Transparenz
- **Skalierbarkeit**
- **Leistung**
- **Administrierbarkeit** (Betriebsmittelnutzung)
- **Zuverlässigkeit** (Umgang mit Fehlern, Ausfällen)
- **Sicherheit** (gegen Angriffe)
- **Offenheit**



# Beispiel für neue Fehlersituation

---

am Beispiel eines Benutzerprogrammes, das entfernte Funktion (API) benutzt:

- lokal: Programm allein fällt aus oder  
Programme und Funktion (Bibliothek)
- verteilt: die entfernte Funktion fällt (allein) aus

## **Denkbare Varianten**

- Programm fällt auch aus
- Programmierer muss neue Fehlersituation berücksichtigen
- Besseres?

# Neue, durch Verteilung verursachte Probleme

---

## **Neue Fehlertypen**

- Botschaften können verloren gehen oder verfälscht werden
- Ausfall von Rechnern oder Netzwerk

## **Neue Angriffsmöglichkeiten**

- Botschaften werden unterdrückt, mitgehört, manipuliert, wiederholt
- gefälschte Absenderangaben
- Rechner werden mit Botschaften bombardiert

## **Neues Betriebsmittel: Netzwerk**

- Netz als Flaschenhals; Bandbreite, Latenz, Kosten
- Adressierung: Finden des richtigen Rechners bzw. Ports

## **Heterogenität**

- Hardware
- Betriebssystem

# Eine grundsätzliche Überlegung

---

Gegeben folgendes Problem

- p, q Prozesse,  
fallen nicht aus,  
kommunizieren via **send/receive**  
Botschaften können verloren gehen  
keine Annahme über Laufzeit der Botschaften möglich
- a, b mögliche Operationen von p, q
- p, q sollen entweder:
  - (i) dieselbe (nicht-triviale) Operation einmal ausführen
  - (ii) keine Operation ausführen

Gibt es eine Botschaftenfolge, die das leistet?

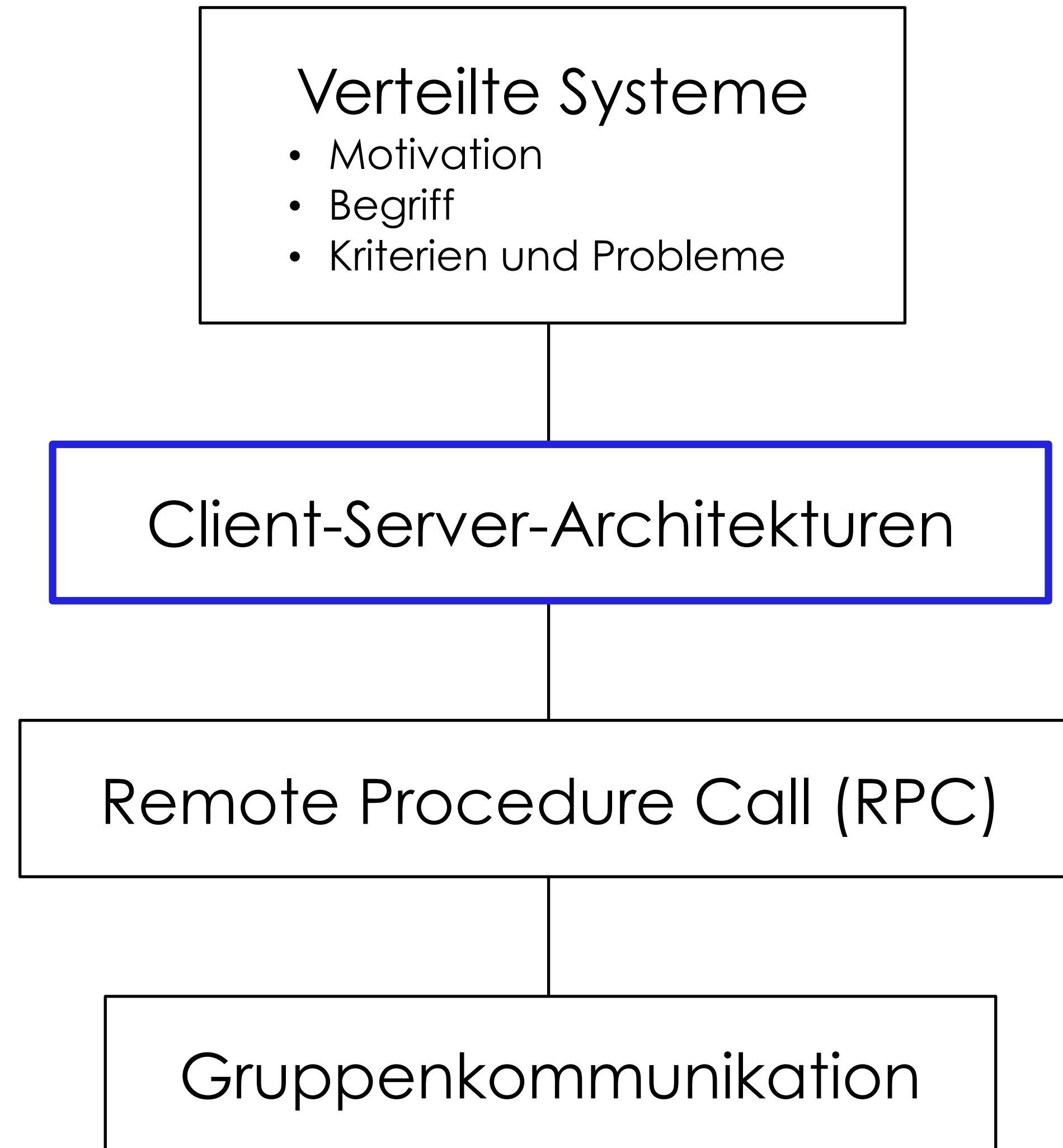
# Eine grundsätzliche Überlegung

## Behauptung

Es gibt kein solches Protokoll, das mit endlich vielen Botschaften auskommt!

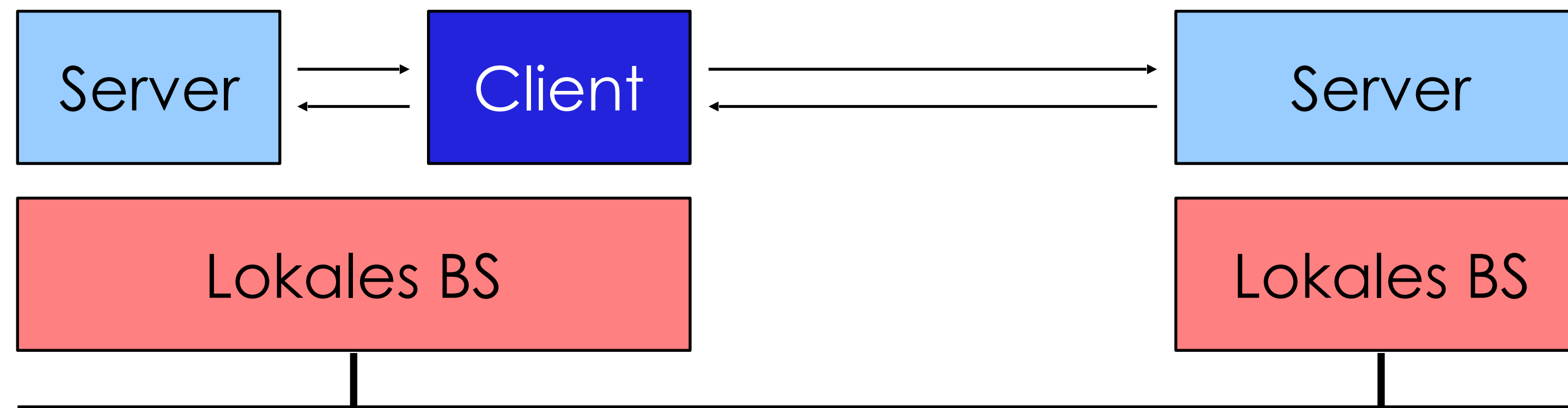
## Beweis

- Angenommen, es gebe solche Protokolle, bestehend aus Folgen von Botschaften:  $(m_{p \rightarrow q}, m_{q \rightarrow p})^*$ .
- Wähle: Protokoll mit geringster Zahl von Botschaften.
- $\underline{m}_{p \rightarrow q}$  sei letzte Botschaft dieses Protokolls, dann kann  $p$  eine solche Entscheidung nicht auf der Annahme aufbauen, dass  $\underline{m}_{p \rightarrow q}$  ankommt (analog für  $q$ ).
- Also kann  $\underline{m}_{p \rightarrow q}$  auch wegfallen.
- Widerspruch, da offensichtlich nicht kleinstes Protokoll!



# Client-Server-Architekturen

- Strukturierung von (Betriebs-) Systemen als eine Menge kooperierender Prozesse (Server), die Dienste anbieten
- Benutzer-Prozesse (Clients) nehmen Dienste durch IPC in Anspruch



**mit oder ohne Netzwerk  
(Mikrokern-System)**

# Konstruktionsprinzip

---

## **Client-Server-Architekturen**

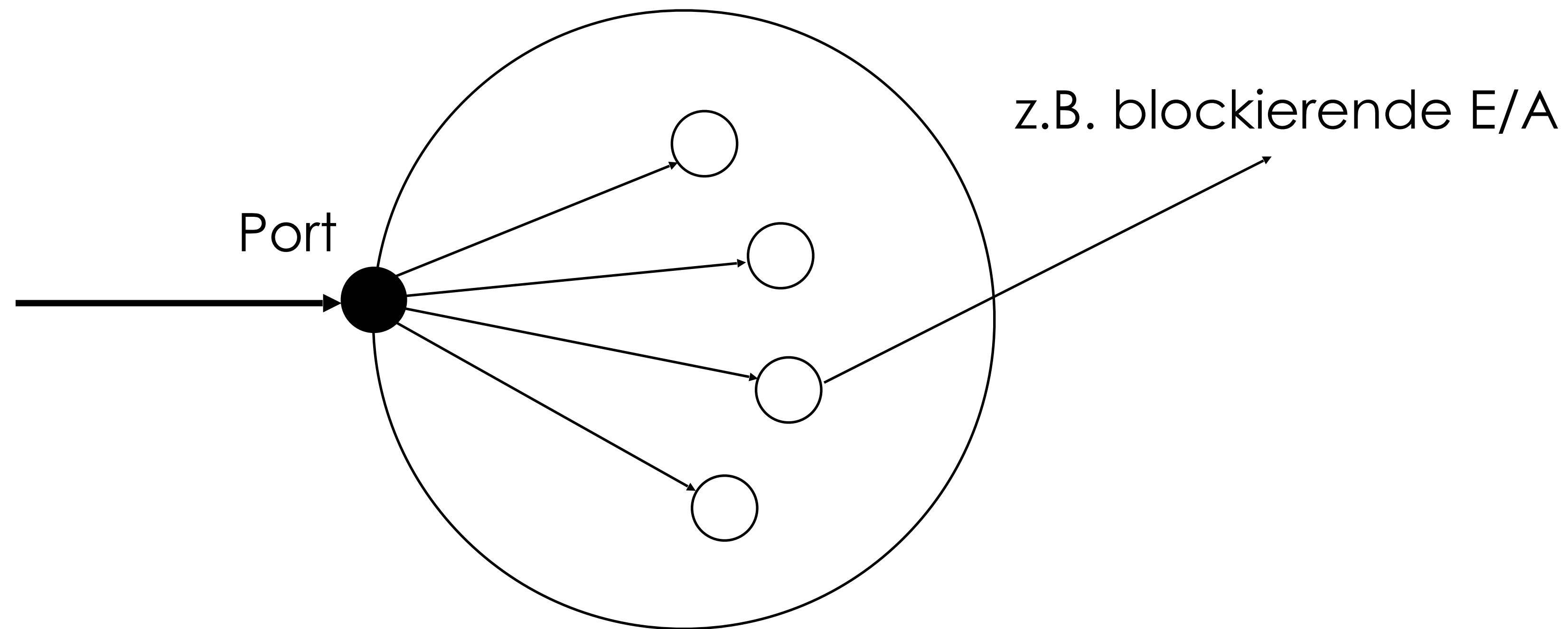
- Dienste werden durch Serverprozesse erbracht
- Klienten nehmen diese durch IPC in Anspruch
- Zustellung von Botschaften (z.B. via Sockets)
- zwischen Prozessen auf demselben Rechner oder verteilt über Netzwerk

## **Kommunikation basierend auf (Funktions-) Aufrufen**

- Simulation durch Botschaften
- Verpackung der In-Parameter in Botschaft, Versenden
- Durchführen der Operation auf der anderen Seite
- Verpackung der Out-Parameter und Zurücksenden

# Einsatz vieler Threads

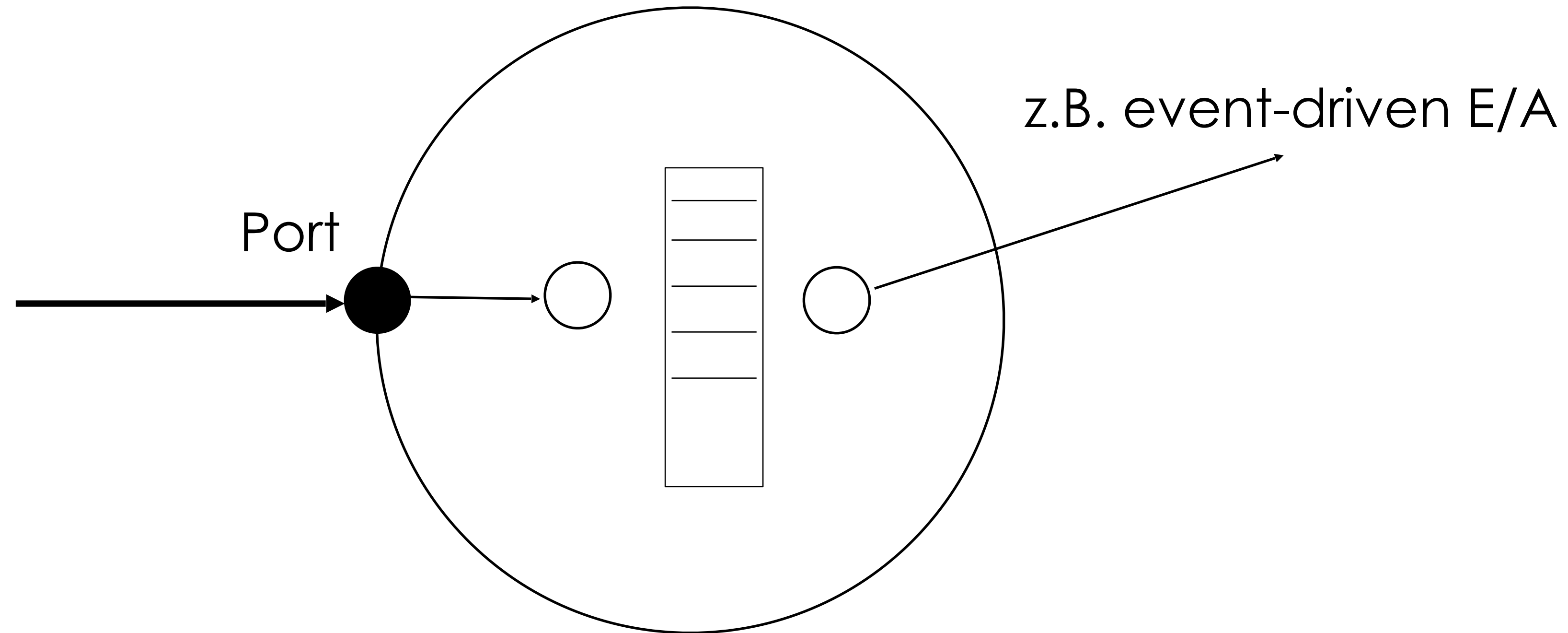
- Server müssen in der Lage sein, viele Aufrufe überlappend auszuführen
- alle Betriebsmittel eines Servers in einem Adressraum
- Grundidee: ein Thread pro Anfrage





# Abarbeitung mit Ereignissen / Events

- Server müssen in der Lage sein, viele Aufrufe überlappend auszuführen
- alle Betriebsmittel eines Servers in einem Adressraum
- Grundidee: Events werden in Stufen behandelt, ein Thread pro Stufe oder pro Core



# Peer-To-Peer Systeme

---

## Ziel und Charakteristika

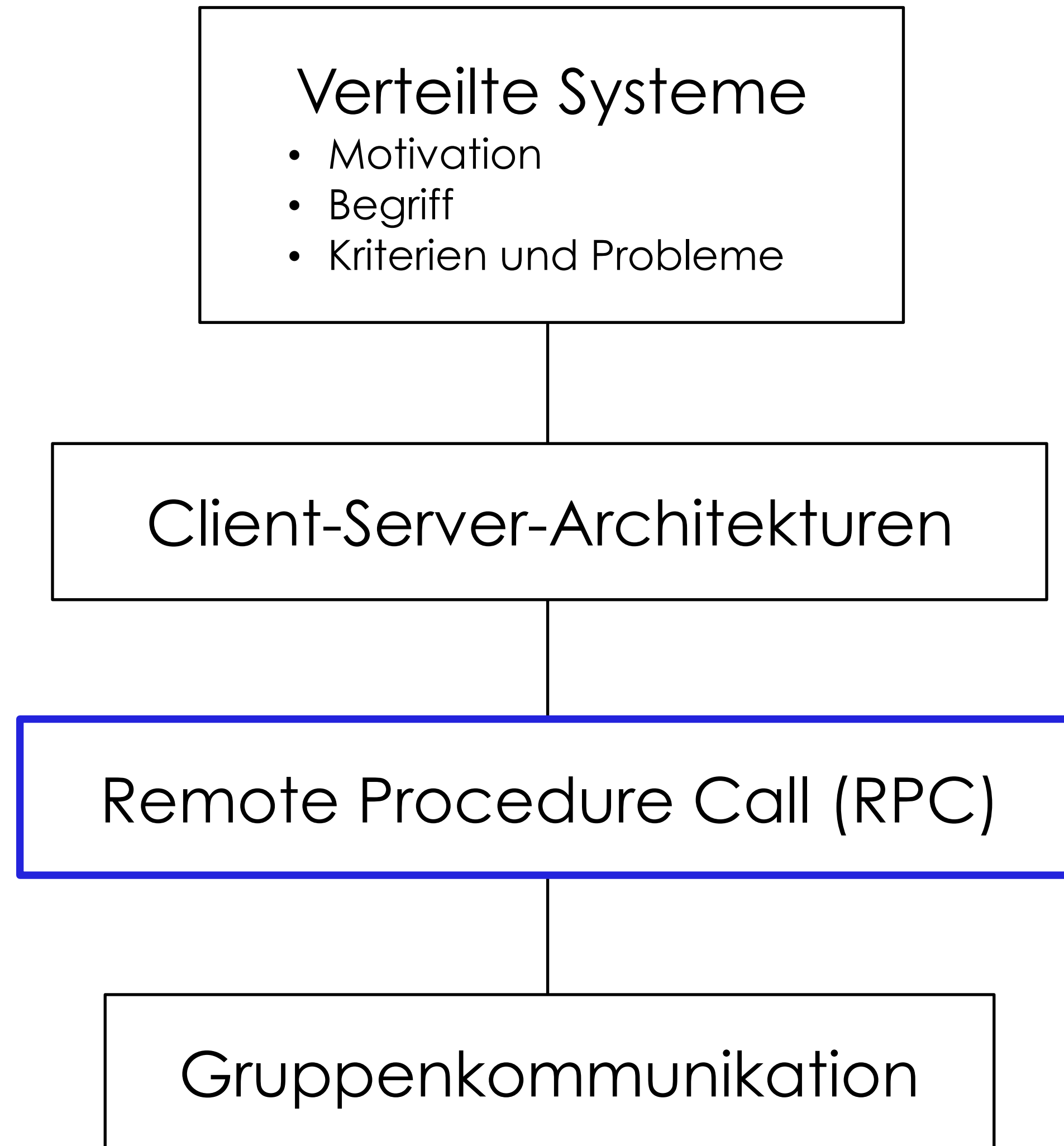
- Nutzung / Kooperation zahlreicher Rechner im Internet
- keine ausgewiesenen und (wohl-)administrierten Server
- zuverlässige, gemeinsame Nutzung (sharing) verteilter und potentiell unzuverlässiger Rechner

## Anwendungen

- gemeinsame Nutzung von Dateien
- Web-Caches

## Problembereiche

- sehr große Zahlen beteiligter Rechner
- Lastverteilung
- große Dynamik in der Verfügbarkeit der beteiligten Rechner
- Sicherheit



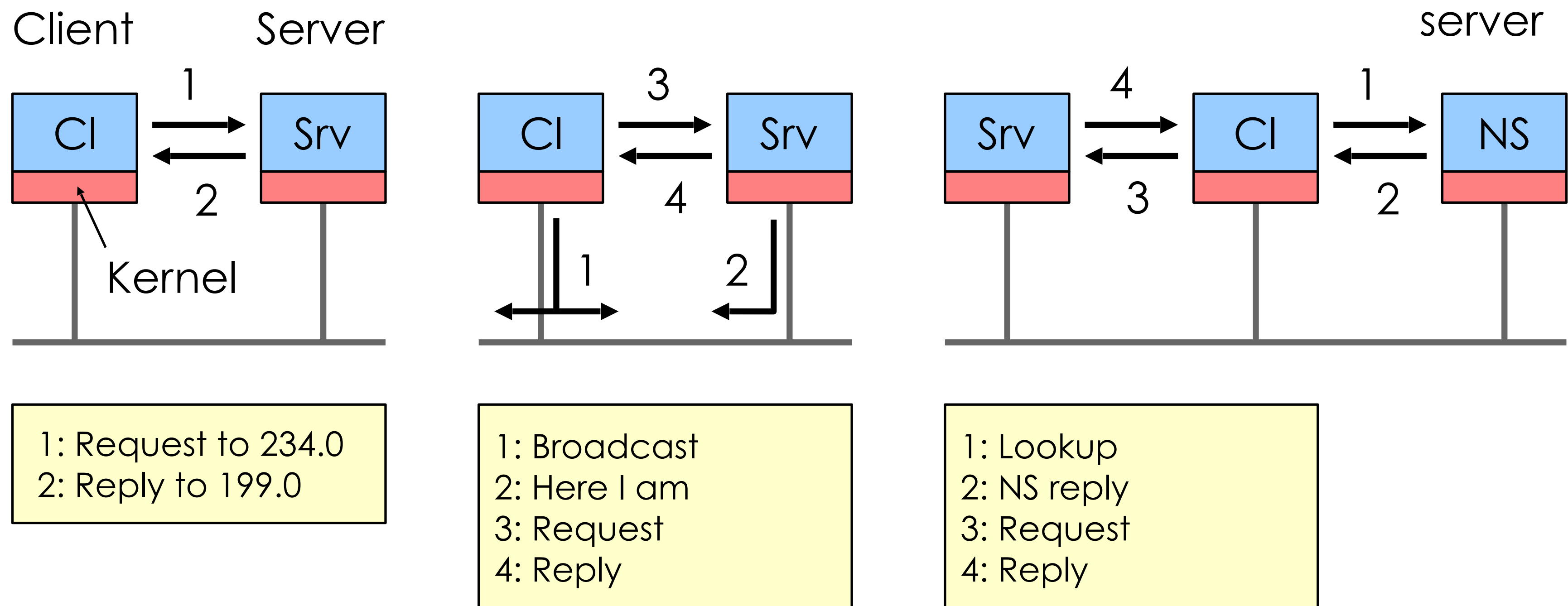
# Einige Begriffe und Abkürzungen

---

- IPC                    Inter Process Communication
- RPC                    Remote Procedure Call
  
- lokal                    auf einem Rechner
- entfernt (remote)    über Rechnergrenze
  
- unicast                ein Partner
- broadcast             an alle (z. B. an alle in einem Teilnetz)
- multicast             an einige (z. B. Mitglieder einer Gruppe)

# Adressierung in Client-Server-Systemen

- Maschine.Prozess (IP-Adresse.Port)
- Broadcast, z. B. große Zahlen als Adressen
- Nameserver
- Spezialfälle: inhaltsbasierte Adresse



# Fernaufruf / Remote Procedure Call

- Ziel: Client-Server-Architekturen handhabbar machen
- Simulation des lokalen Prozedur-Aufrufes mittels Botschaften

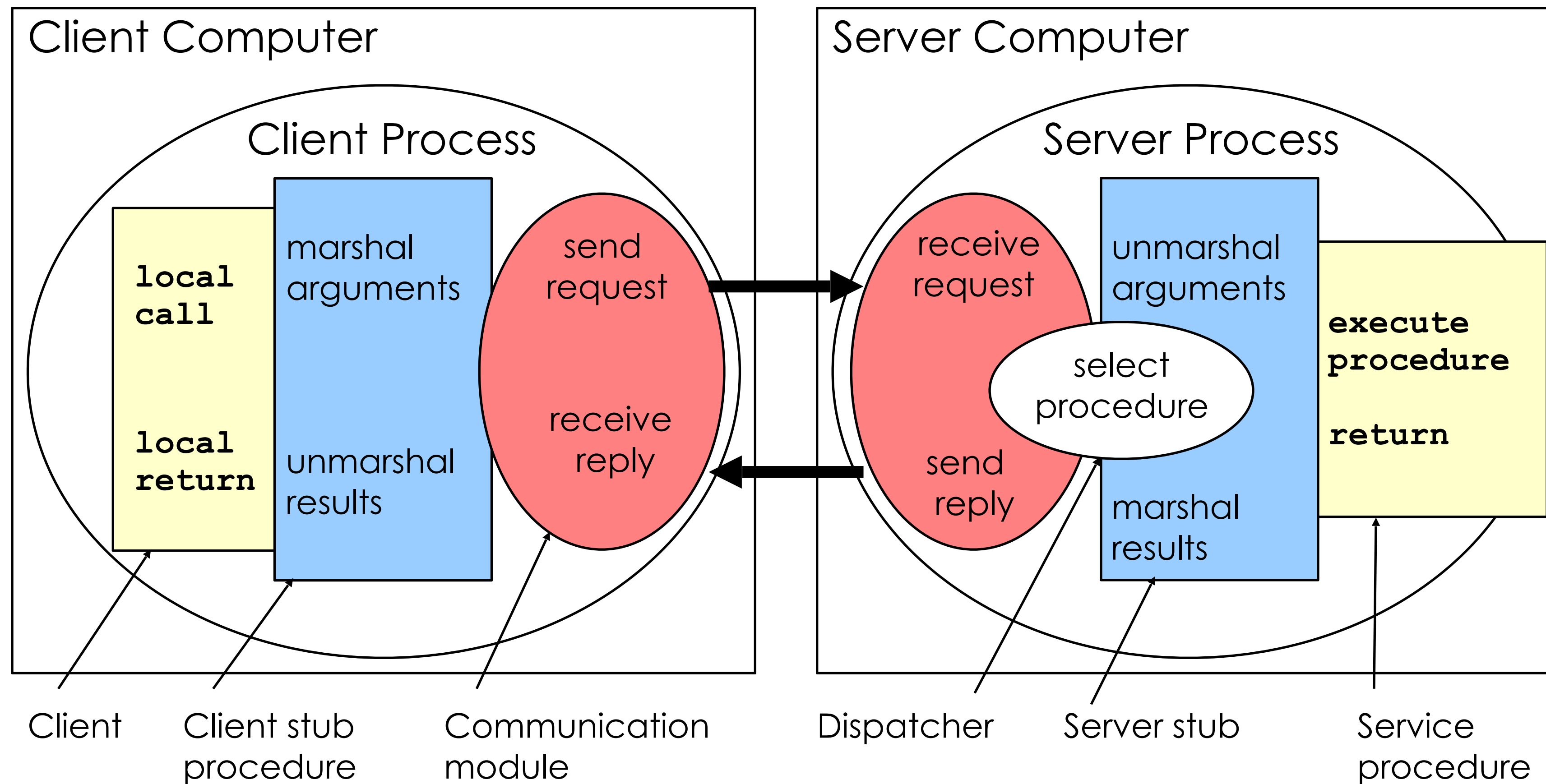
## Client

```
send(server, message);  
receive(reply);
```

## Server

```
while (1) {  
    wait(message);  
    switch (message.op) {  
    case procl:  
        ...  
        send(client, reply);  
        break;  
    }  
}
```

# RPC-Implementierung



Aufgaben: Parameterrückgabe (Marshalling), Binden (Auffinden des zuständigen Servers), Kommunikation

# Ablauf RPC

---

- Client ruft Client-Stub-Funktion auf
- Client steckt Parameter in Nachricht, verzweigt in Kern
- Kern sendet Nachricht an Server
- Client-Stub: receive, wird blockiert
- entfernter Kern gibt Nachricht an Server-Stub
- Server-Stub packt Parameter aus, ruft Prozedur auf
- Server-Stub packt Resultat in Nachricht, verzweigt in Kern
- Server-Stub: send, receive wird blockiert
- Client-Kern gibt Nachricht an Client-Stub
- Client-Stub packt Resultat aus, übergibt an Client



# Unterschiede RPC / Funktionsaufruf

---

## **Parameter**

- Call by Reference (unüblich, nicht unmöglich)
- statt dessen: Kopieren

## **Fragestellungen**

- komplexe Datenstrukturen als Parameter: deep copy
- Fehlerisolation
- Sicherheit

## **Effizienz**

- Funktionsaufruf: 0–100 Takte
- Lokales RPCs: 1000–10000 Takte

# Fehlerbehandlungen

---

## **Fehlersemantik Funktionsaufruf**

- exakt ein Aufruf / exactly once

## **Verfälschte Botschaften**

- erkennbar und korrigierbar durch redundante Kodierung

## **Verlorene Request-Botschaft**

- Timeout und Wiederholung

## **Verlorene Reply-Botschaft**

- Timeout und Wiederholung des Requests
- dann: „at least once“-Semantik / mögliche Duplikation
- sonst: „at most once“-Semantik

# RPC-Fehlersemantik

---

## **May Be**

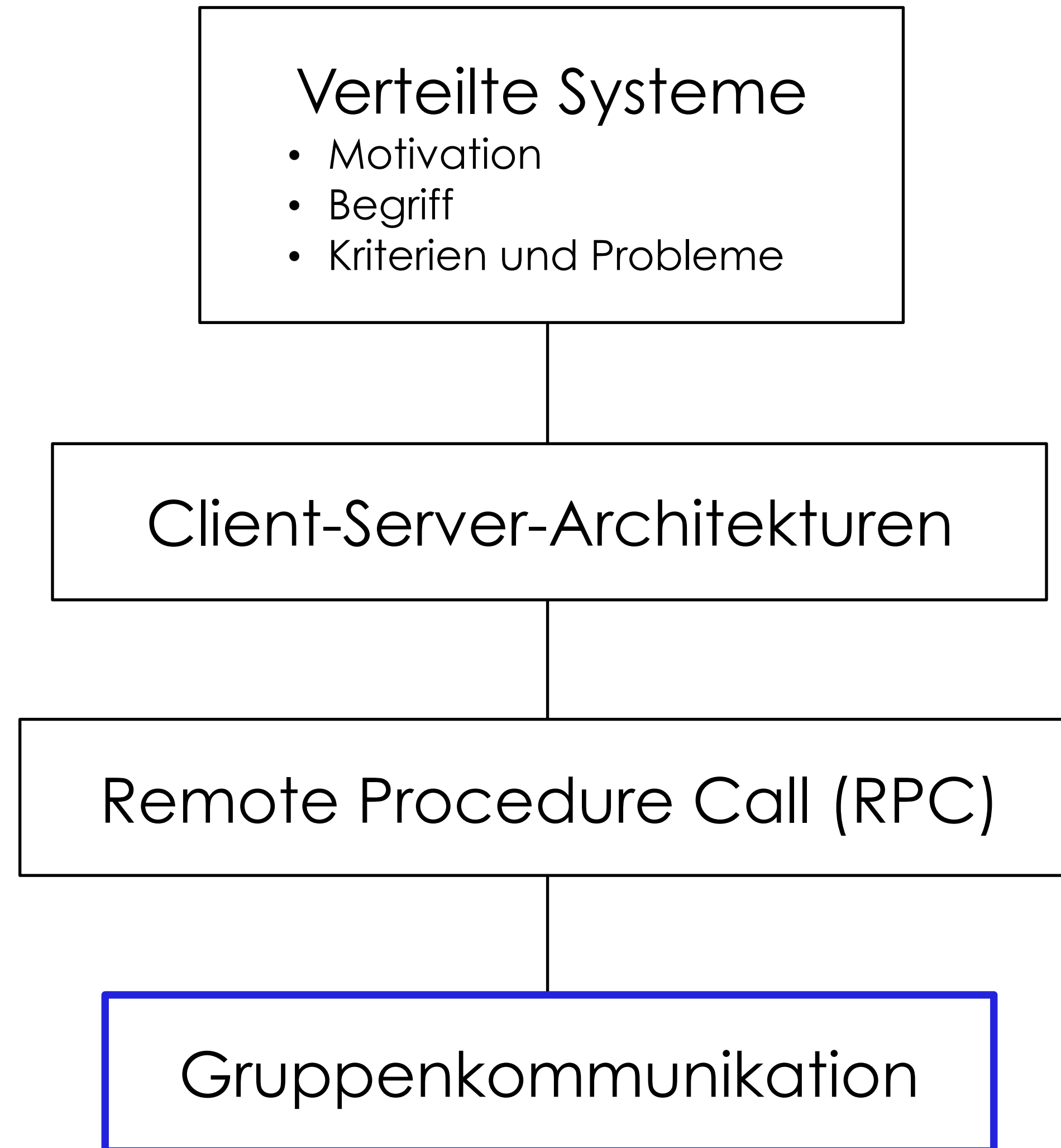
- Nachricht genau einmal versenden, keine Kontrolle

## **At Least Once (originales NFS, SUN RPC)**

- wiederholtes Senden der Request-Botschaft bis zum Reply
- nur für Anwendungen mit idempotenter Semantik

## **At Most Once**

- Senden einer Fehlernachricht:  
Duplikat-Erkennung / Reply-Wiederholung
- wenn Server nicht ausfällt und ein Reply kommt, dann „exactly once“-Semantik, sonst unklar



# Gruppen-IPC: Multicast

---

## **Gruppe**

Menge von Prozessen, die auf eine vom System oder von Nutzern festgelegte Art zusammenarbeiten

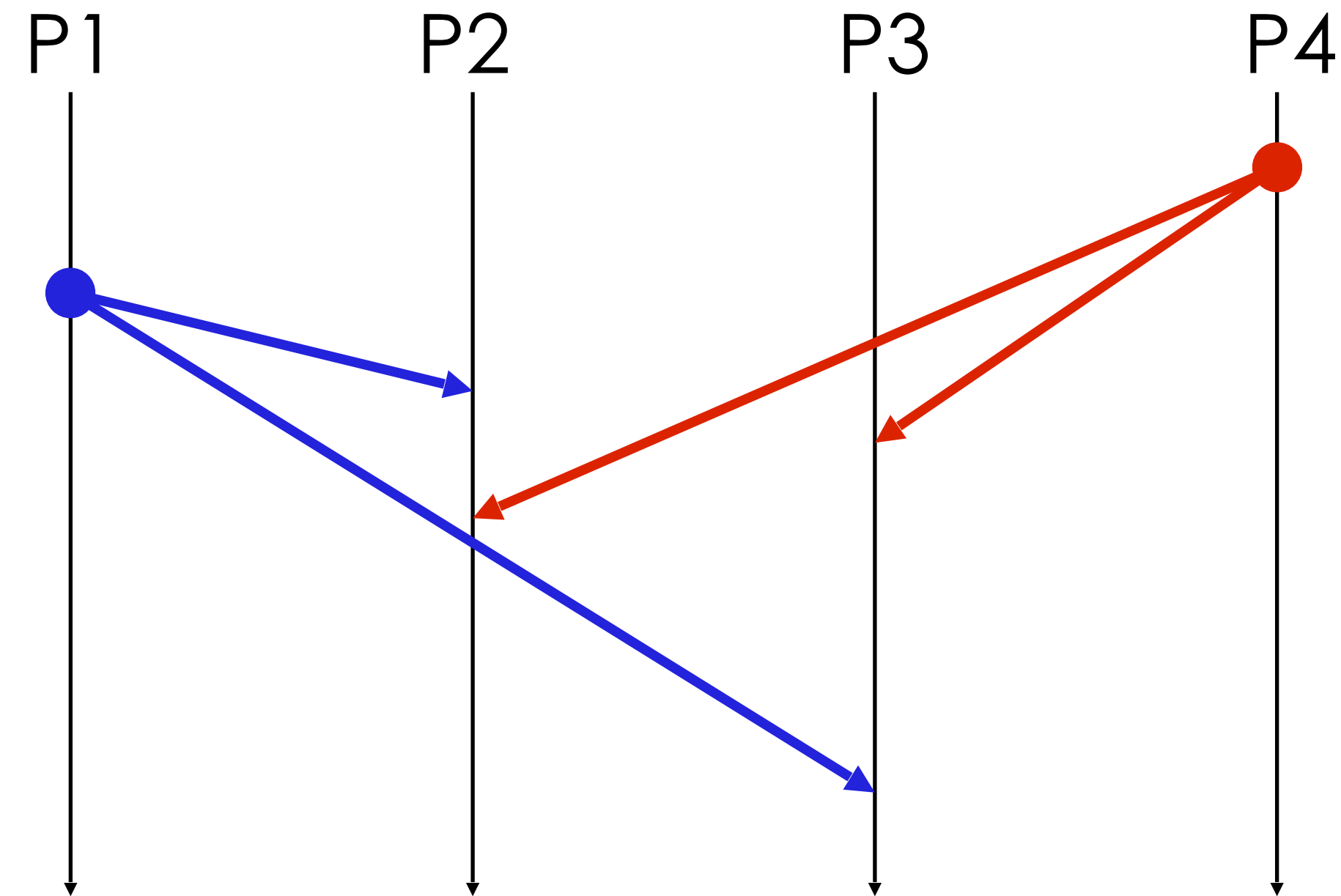
## **Multicast**

1 Botschaft eines Prozesses an alle Prozesse einer Gruppe

## **Anwendungen**

- Fehlertoleranz und Skalierbarkeit durch replizierter Server
- Konsistenz von Kopien (multiple update)
- Auffinden von Objekten in verteilten Anwendungen
- Bandbreitenreduzierung (1 mal übertragen statt n mal)

# Botschaften-Reihenfolge



P1, P4

Klienten

P2, P3

replizierte Server

Botschaften Schreib-Operationen führen zu Konsistenz-Problem

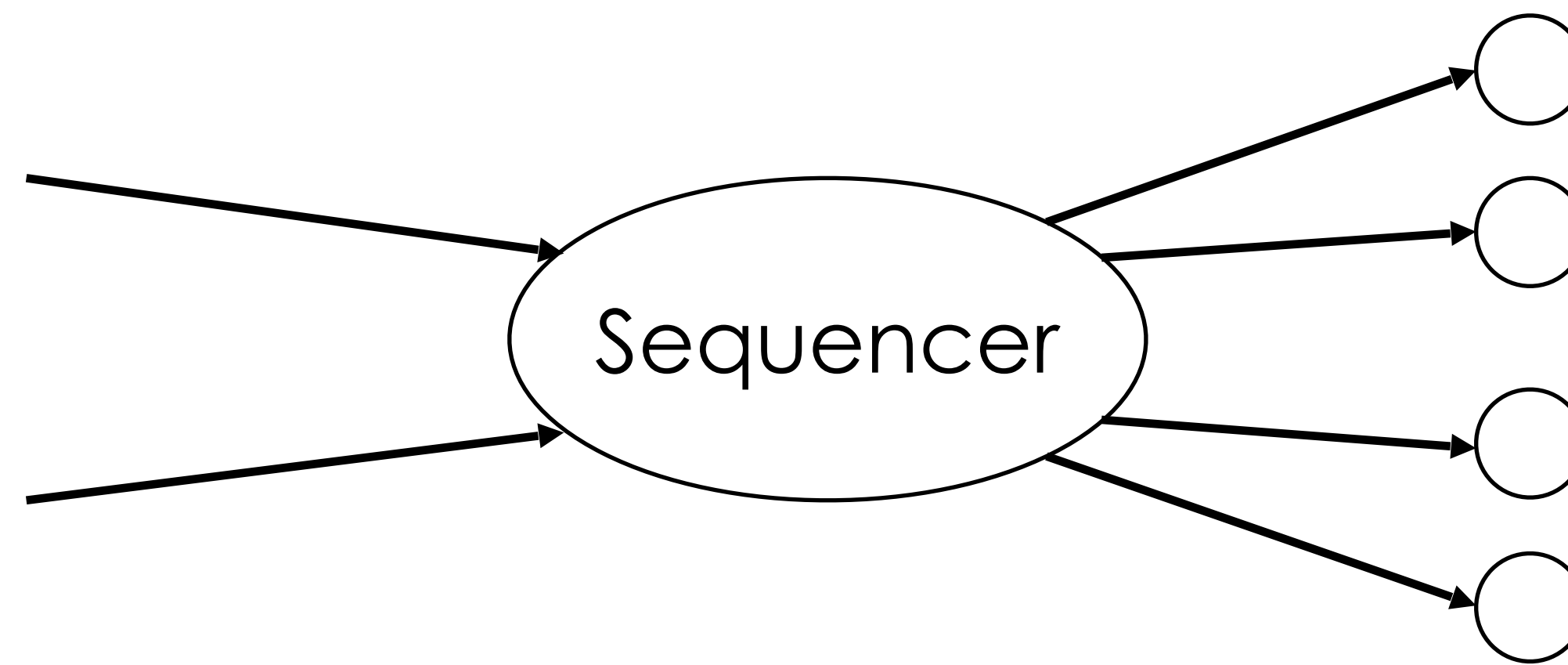
# Botschaften-Reihenfolge

## **Totally Ordered Multicast**

Werden mehrere Botschaften an eine Gruppe gesendet, so werden sie von allen Mitgliedern der Gruppe in gleicher Reihenfolge empfangen

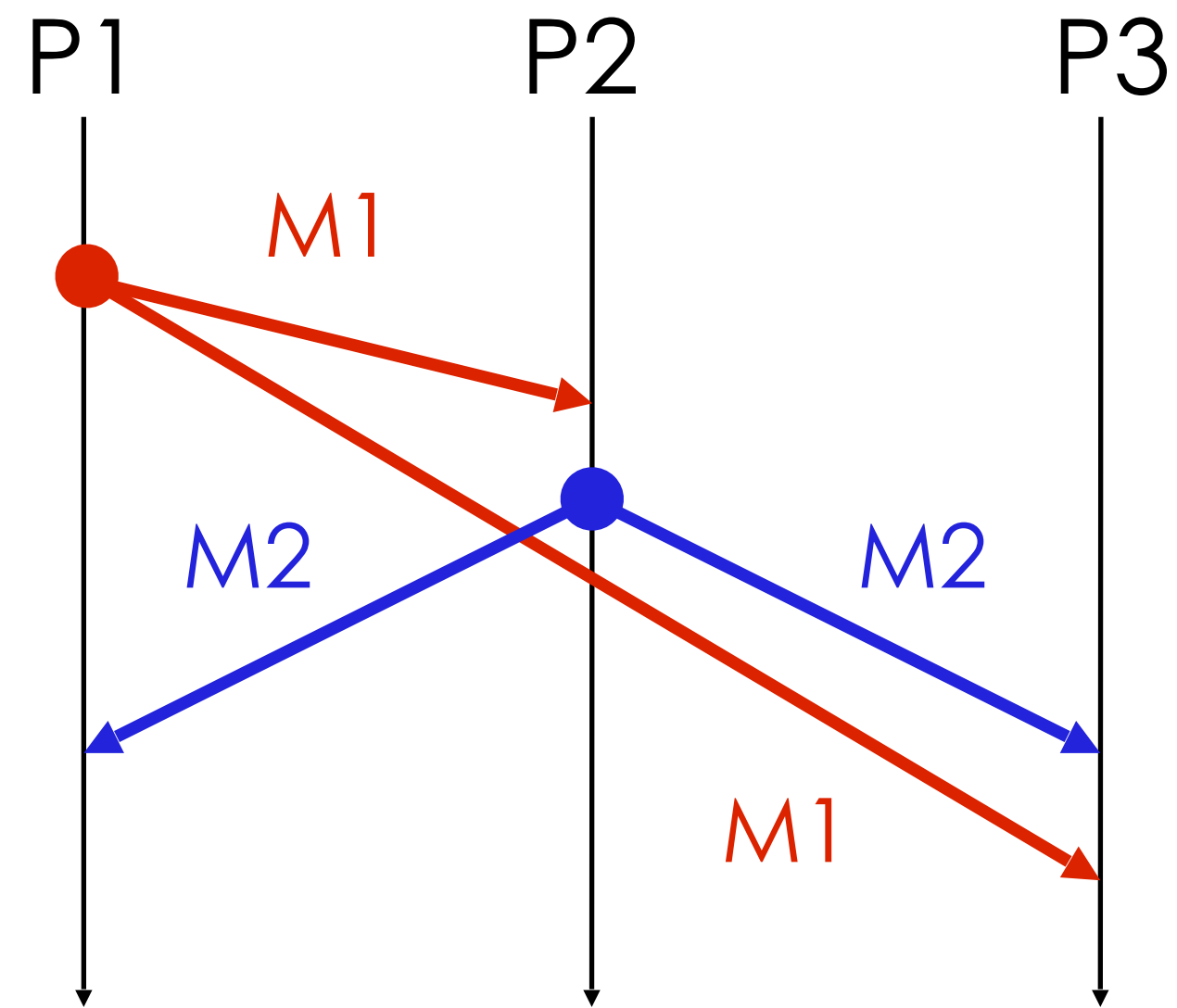
## **Mögliche Implementierung**

ein Prozess (Sequencer) ist für Zustellung zuständig



# Botschaften-Reihenfolge

Beispiel für nicht  
kausal geordnet:



P1, P2, P3 Teilnehmer an verteiltem Datenspeicher

M1 Anfrage

M2 Antwort auf Anfrage

## Causally Ordered Multicast

Aufrechterhaltung einer potentiellen kausalen Abhängigkeit zwischen Ereignissen



# Zusammenfassung

---

- Inter Process Communication
- Grundlagen verteilter Systeme
- Client-Server-Architekturen
- Fehlersemantik

## **Werkzeuge**

- original: SunRPC & rpcgen
- moderner: Google Protocol Buffers & gRPC
- oder JSON, REST, ...