Verteilte Systeme

Disjunktives Warten

Select (BSD)

```
fd_set fds; // bitmap of fds awaiting/having data
int maxfd = max(fd1,fd2)+1; // length of the bitmask
struct timeval timeout;
for (;;) {
    FD_ZERO(&fds);
    FD_SET(fd1, &fds); FD_SET(fd2, &fds);
    timeout.tv_sec = 5; timeout.tv_usec = 0;

                                read-fds    write-fds    error-fds

    int ready = select(maxfd, &fds, NULL, NULL, &timeout);

    if (ready == 0) { /* handle timeout */ continue; }
    if (FD_ISSET(fd1, &fds)) { /* read from df1 */ }
    if (FD_ISSET(fd2, &fds)) { /* read from fd2 */ }
}
```

Poll (SysV)

```
int fdcount = 2
struct pollfd *fds;
fds = calloc( fdcount, sizeof(struct pollfd) );

fds[0].fd = fd1; fds[0].events |= POLLIN;
fds[1].fd = fd2; fds[1].events |= POLLIN;
fds[1].fd = fd2; fds[1].events |= POLLOUT;

for (;;) {
   int ready = poll(fds, fdcount, 5*1000);
   if (ready == 0) { /* handle timeout */ continue; }
   if (fds[0].fd.revents & POLLIN) { /* read from fd1 */ }
   if (fds[1].fd.revents & POLLIN) { /* read from fd2 */ }
   if (fds[1].fd.revents & POLLOUT) { /* write to fd2 */ }
}
```

Disjunktives Warten (Alternativen)

epoll (in Linux)

API ähnlich zu poll, benutzt aber gekapselt durch eine Userspace-Library einen speziellen Filedeskriptor zur Abfrage der anstehenden I/O-Operationen im Kernel.

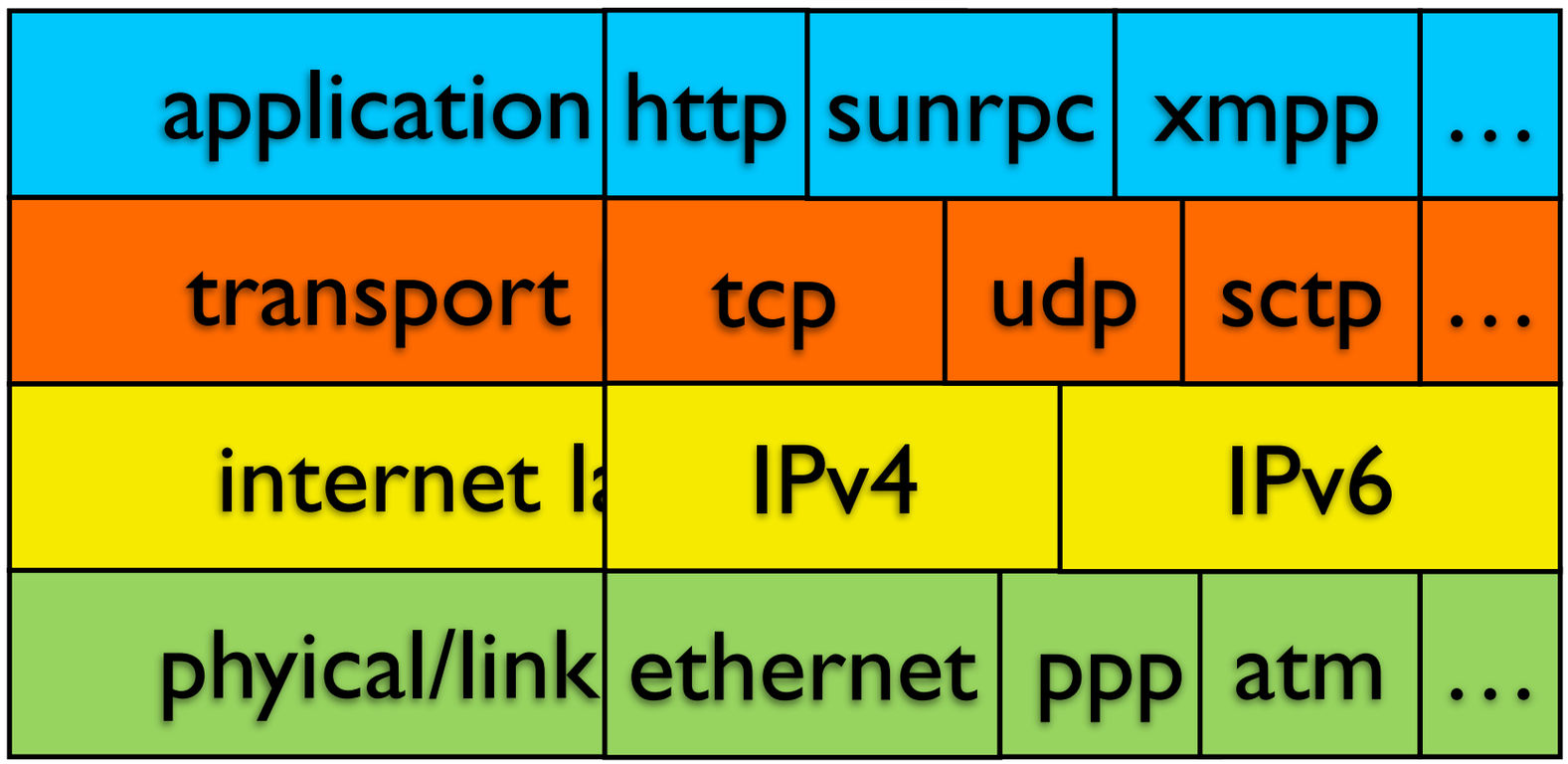
kqueue (in FreeBSD / MacOS X)

Ermöglicht es auf Mengen verschiedener Ereignisse zu warten. Flexibler als poll/select/epoll, aber etwas umständlicher zu benutzen.
kqueues werden nicht an Kindprozesse vererbt (im Gegensatz zu Filedeskriptoren).

Verteilte Systeme

Kommunikationsdienste im Internet

Kommunikationsdienste im Internet



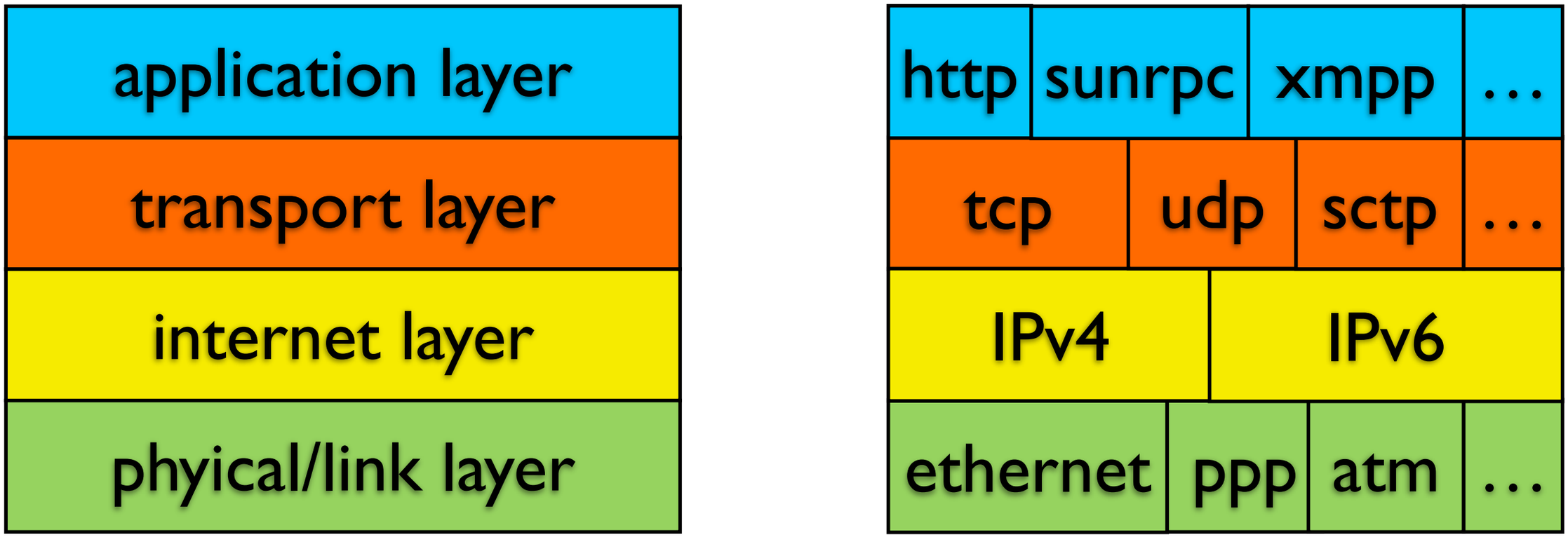
IPv4 (veraltet)

- Adressierung von Rechner im Netz (32bit Adresse)
- Prüfsumme gegen Headerkorruption
- Erweiterung für QoS und ECN
- Adressierung von Protokollen der höhergelegen Schicht
- Fragmentierung von Paketen, falls diese für darunterliegende Schicht zu groß sind.

IPv6

- Adressierung von Rechner im Netz (128bit Adresse)
- QoS und ECN
- Adressierung von Protokollen der höhergelegen Schicht
- Prüfsumme gegen Headerkorruption und Fragmetierung optional über Extensions

Kommunikationsdienste im Internet



UDP

- Unzuverlässige Nachrichtenübertragung
- Prüfsumme gegen Datenkorruption
- Nicht Reihenfolgetreu
- Keine Fluß- und Staukontrolle

DCCP

- Unzuverlässige Nachrichtenübertragung
- Prüfsumme gegen Datenkorruption
- optional Reihenfolgetreu
- Integrierte Fluß- und Staukontrolle

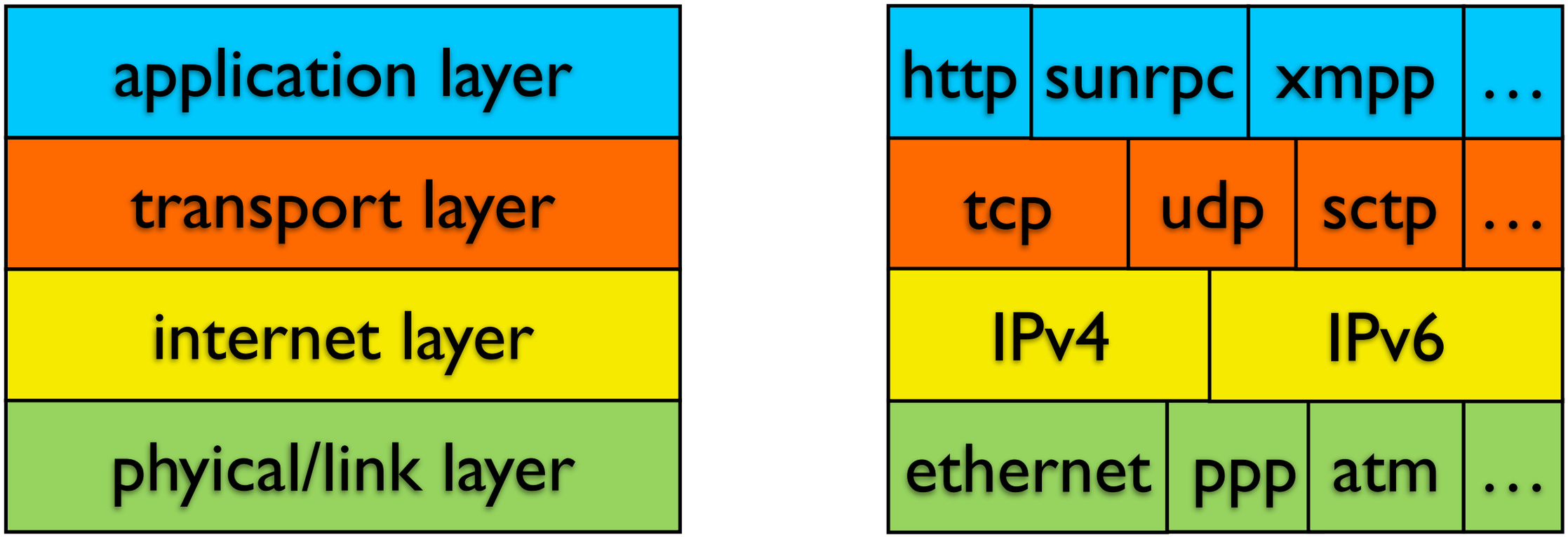
TCP

- Zuverlässige Streamübertraung
- Reihenfolgetreu
- Integrierte Fluß- und Staukontrolle

SCTP

- Zuverlässige Nachrichtenübertragung
- optional Reihenfolgetreu
- Mehrere Paketstreams in einer Verbinung (Association)
- Integrierte Fluß- und Staukontrolle pro Paketstream
- Kann mehrere Übertargungswege gelichzeitig verwenden

Kommunikationsdienste im Internet



HTTP

- Einfaches Request/Response Protokoll
- basiert auf TCP
- Mehrere Requests pro TCP Verbindung (optional)
- Aushandlung welche Datentypen übertragen werden
- Viele Erweiterungen (z.B. Authentication)

XMPP

- Zuverlässiges Nachrichtenübertragungsprotokoll
- Basiert auf TCP
- Nachrichten sind in XML kodiert
- Eigene Adressierung
- Übertragung über Server als Mittler
- Pufferung durch Server, falls Empfänger nicht Erreichbar

RTP

- Unzuverlässiges Streamübertragungsprotokoll für Audio und Videoübertragung
- Kann wahlweise auf UDP, TCP, SCTP oder DCCP basieren
- Handelt mehrere unzuverlässige Streams aus, die ggf. über eine oder mehrere Verbindungen realisiert werden
- Kontrollprotokoll (RTCP), dass QoS und Zeitsyncronisation zwischen mehreren Verbindungen ermöglicht

TCP Client

```
int sockfd, servlen,n;
struct sockaddr_in6  serv_addr;
struct hostent *hp;

hp =  gethostbyname2(“ipv6.google.com”, AF_INET6);
if (hp == NULL) …
if ((sockfd = socket(AF_INET6, SOCK_STREAM,0)) < 0) ...
```

```
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin6_len = sizeof(serv_addr);
serv_addr.sin6_family = AF_INET6;
memcpy((char *)&serv_addr.sin6_addr, hp->h_addr, hp->h_length);
serv_addr.sin6_port = 80;
if (connect(sockfd, (struct sockaddr *) &serv_addr, servlen) < 0)
    perror("Connecting");
write(sockfd,”HTTP/1.0\nGET /\n”,18);
```

TCP Server

```
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin6_len = sizeof(serv_addr);
serv_addr.sin6_family = AF_INET6;
serv_addr.sin6_port = 23;
serv_addr.sin6_addr = in6addr_any;

if ((sockfd = socket(AF_INET6, SOCK_STREAM,0)) < 0)
    perror("Creating socket");
if(bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)<0)
    error("binding socket");
listen(sockfd,5);

clilen = sizeof(cli_addr);
clifd = accept(sockfd,(struct sockaddr *)&cli_addr,&clilen);
```

Verteilte Systeme

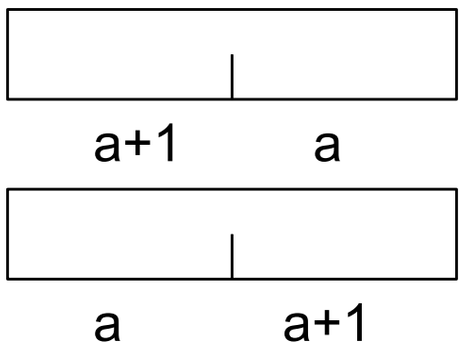
Verteilungsabstraktion

Verteilte Programmierung ist mühsam und fehleranfällig, da viele Probleme zu Berücksichtigen sind:

- Eigenschaften des Kommunikationssystems
- Datenrepräsentation der beteiligten Plattformen
- Packen/Entpacken der Daten für den Transport
- Verbindungsaufbau zwischen den Prozessen

Datenrepräsentation

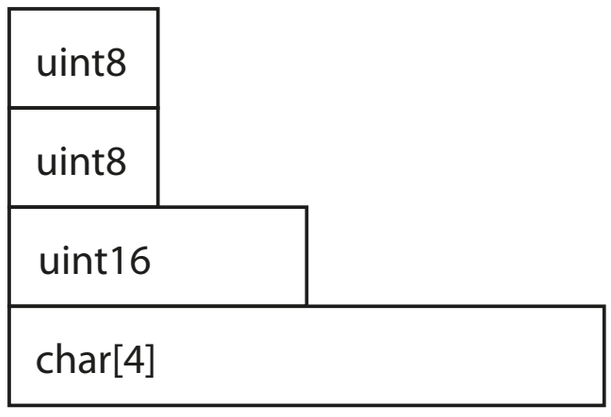
- Byte-Reihenfolge

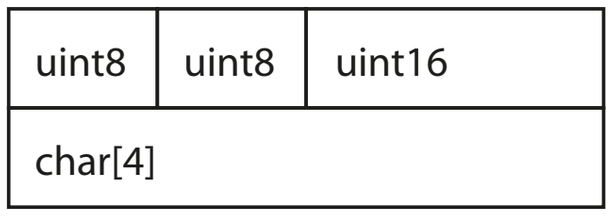


big-endian (z.B. PPC)

little-endian (z.B. Intel)

- Zeichencodierung (ASCII, UTF-8, UTF-16, EBCDIC)
- Gleitkommazahlen
- Strukturausrichtung





Datenrepräsentation

Es gibt zwei Alternativen, das Problem zu lösen:

- „receiver makes it right" – Umwandlung bei Bedarf

- Verwendung einer kanonischen Darstellung im Netz
  z.B. gemäß OSI-Schichtmodell durch Presentation Layer (6)

Network Byte Order

host byte order ↔ network byte order

```
u_long htonl(u_long number);   “host to network long“
u_long ntohl(u_long number);   “network to host long“

u_short htons(u_short number); “host to network short“
u_short ntohs(u_short number); “network to host short“
```

Eine Abstraktionsschicht für die Verteilung, die “Middelware”, kann darüber hinaus weitere Dienste bieten:

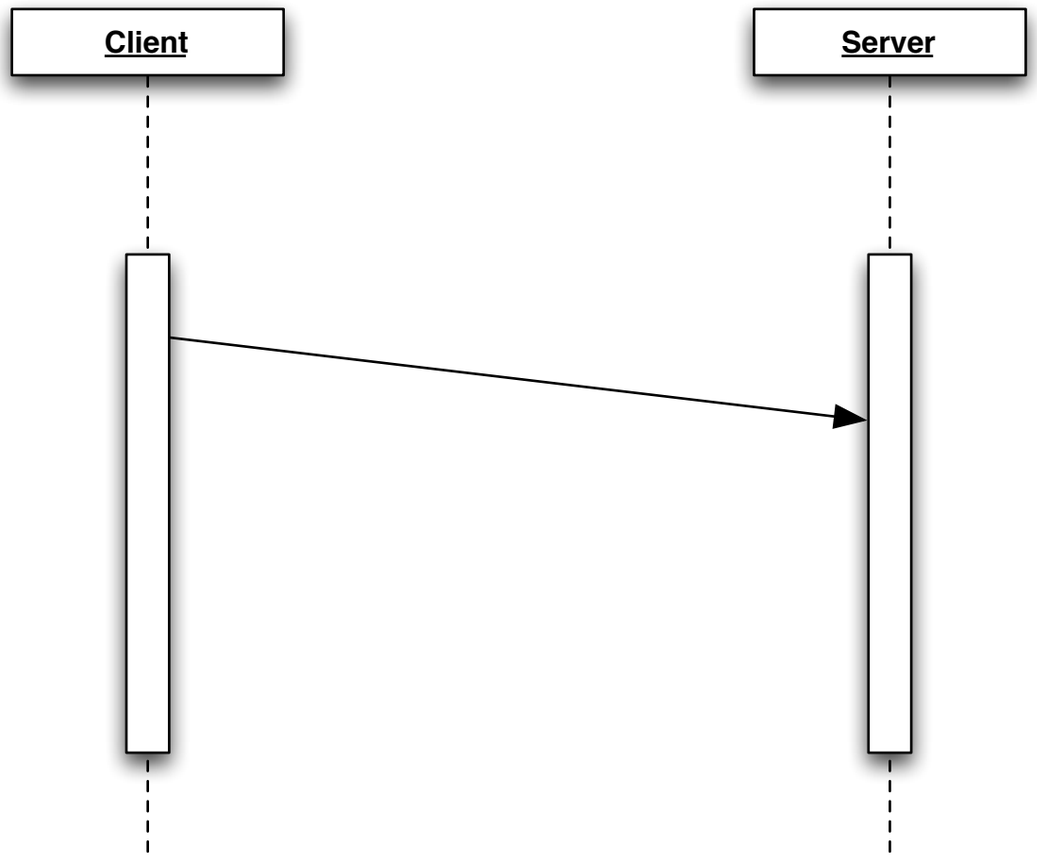
- Fernaufrufe
- Aushandlung der Datenrepräsentation
- Sprach- und Plattformunabhängige Serialisierung von Strukturen oder Objekten
- Lokalisierung von Diensten, Lastverteilung, …

- Lokale Funktionsaufrufe verhalten sich deterministisch: die aufgerufene Funktion wird genau einmal aufgerufen.

- Will man Funktionsaufrufe durch Anfragen/Antworten in einem Kommunikationssystem nachbilden, können Probleme auftreten:

  - Anfrage kann verlohren gehen
  - Antwort kann verlohren gehen
  - Anfrage kann mehrfach ausgeführt werden

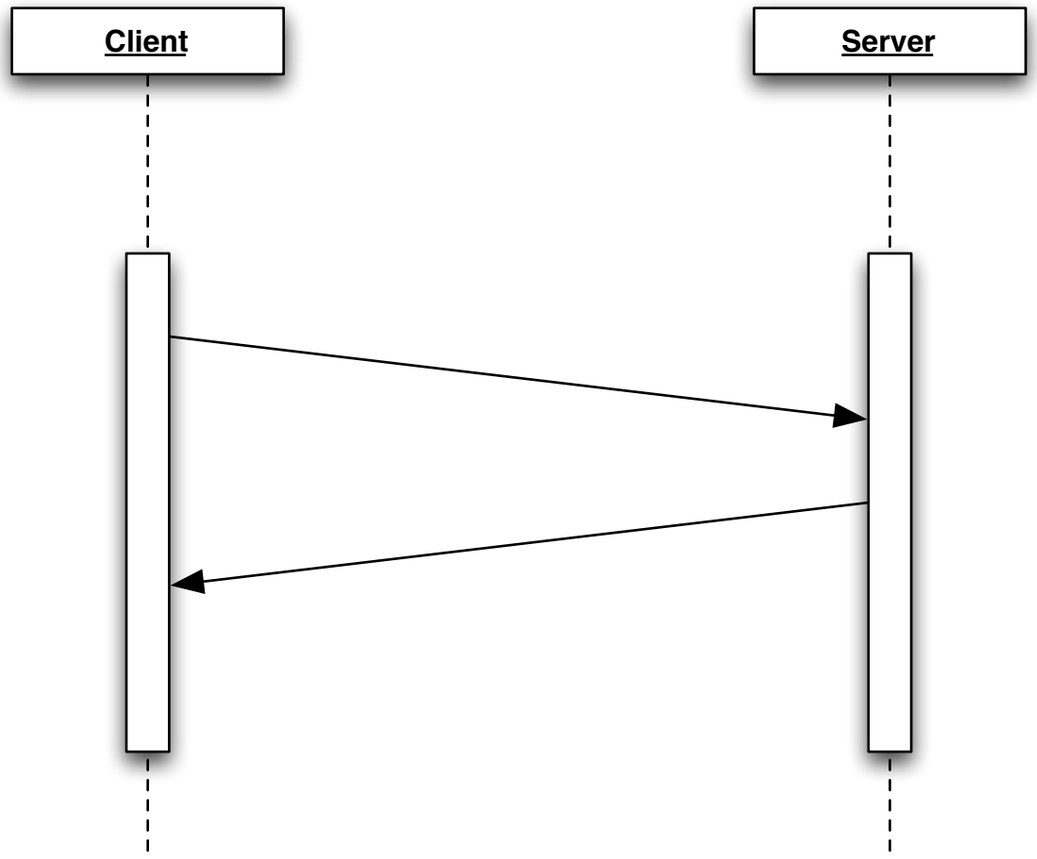
Request



Semantik: Maximal einmal

Problem: Es ist für den Client nicht entscheidbar, ob die Anfrage ausgeführt wurde

Request, Replay



Semantik: Mindestens einmal

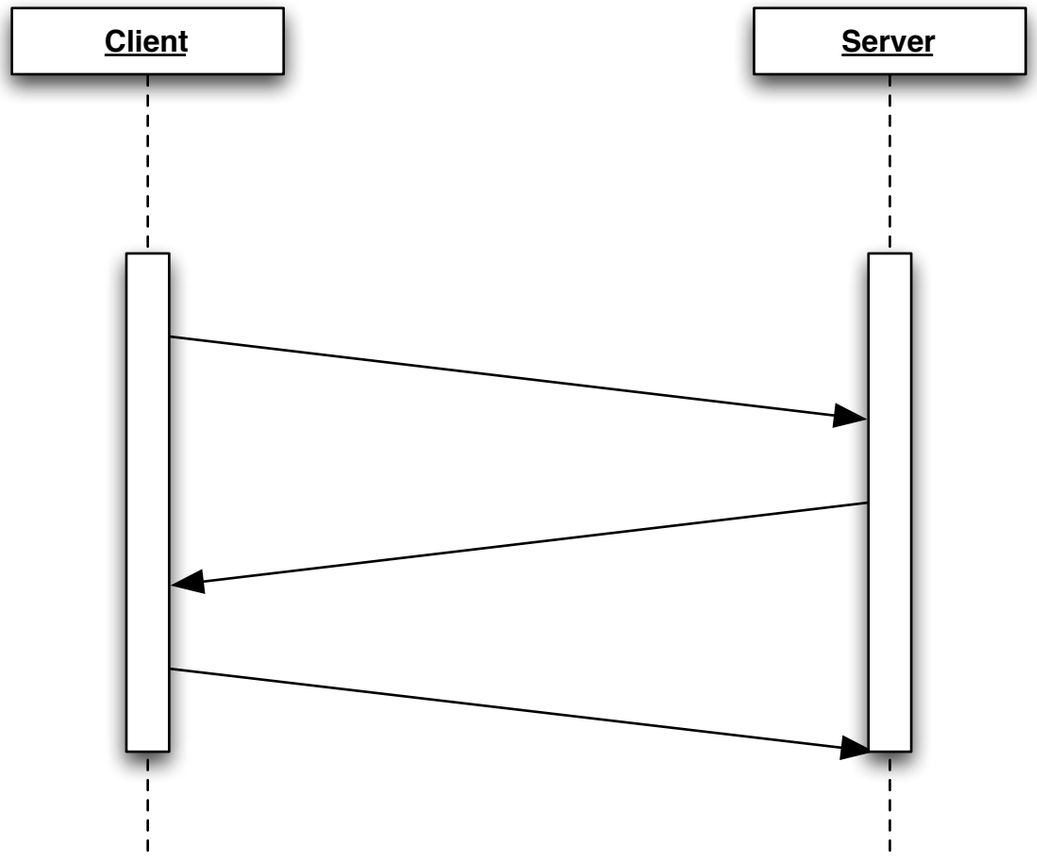
Antwort bleibt aus ⇒ Anfrage nach Timeout wiederholen.

Problem: Erfüllung der Anfrage ggf. mehrfach nötig.

Lösung: ID der Anfrage merken, ggf. Antwort wiederholt senden.

neues Problem: wann kann der Zustand der Anfrage verworfen werden?

Request, Replay, Acknowledge



Semantik: Genau einmal

Antwort bleibt aus ⇒ Anfrage nach Timeout wiederholen.

Bestätigung bleibt aus ⇒ Antwort nach Timeout wiederholen.

Beispiel: SUN-RPC

- Fehlersemantik:
  at-most-once für TCP,
  at-least-once für UDP

- Datendarstellung: XDR (RFC 1014)

- Zugriffsschutz: entweder keiner oder Unix-Autorisierung oder Kerberos.

SunRPC XDR

Quelle: W.R.Stevens: UNIX Network Programming

```
/* date.x Specification of the remote date and time server */
/* Define two procedures
 *  bin_date_1() returns the binary date and time (no args)
 *  str_date_1() takes a binary */
program DATE_PROGRAM {
   version DATE_VERSION {
      long BIN_DATE(void) = 1; /* get binary time code */
                               /* procedure number = 1 */
     string STR_DATE(long) = 2; /* convert code to string */
                               /* procedure number = 2 */
   } = 1;                      /* version number = 1 */
} = 0x31234567;                /* program number */
```

SunRPC XDR

Stub-Generator rpcgen generiert daraus:

date.h	Gemeinsame Modul-Definition (header)
date_clnt.c	Stub für den Klienten
date_svc.c	Stub für den Anbieter, inklusive main()

und optional mit Schalter -Sc bzw. -Ss :

date_client.c	Beispiel für Zugriff durch Klienten
date_server.c	Beispiel für Implementierung des Anbieters

SunRPC Verbindungsaufbau

1. Auf dem Server existiert (hoffentlich) ein Namensdienst: Portmapper, stets über Port 111 ansprechbar.

2. Fernaufruf (!) an Portmapper übermittelt den „Namen“ (program, version, protocol) und liefert als Antwort die Portnummer des entsprechenden Anbieters.

3. Internetadresse von Host sowie gelieferte Portnummer werden im Client Handle eingetragen (und dieses wird beim Fernaufruf als zusätzliches Argument (!) dem Vertreter übergeben)

Übungsaufgabe zum 3.5.2011

- Lesen Sie Sich in die Java Socket API (javax.nio) ein
- Implementieren Sie das Channel-Interface für UDP.
- Beschreiben Sie die Limitierungen Ihrer Implementierung z.B. im Bezug auf Paketgrößen und skizzieren Sie Lösungsansätze dafür

Benutzen Sie die “UdpChannelFactory”-Klasse als Vorlage.