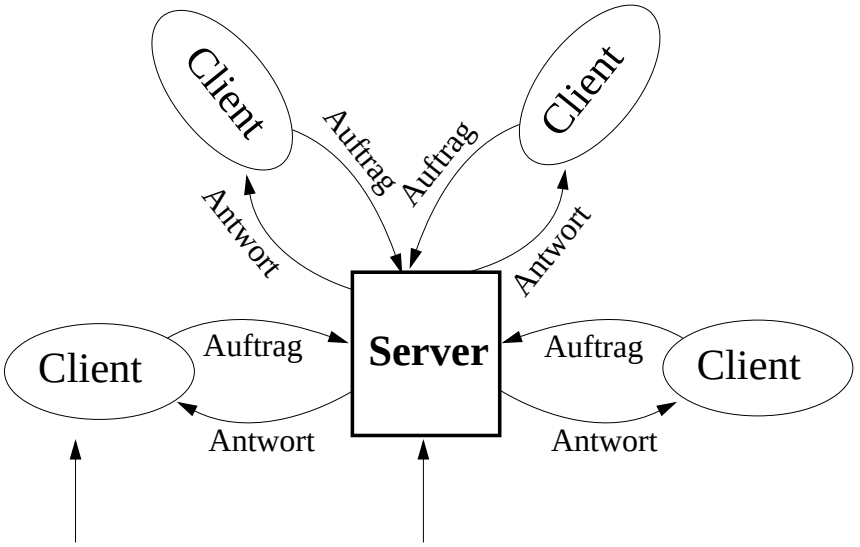
Client/Server-Modell

Das Client/Server-Modell



- Aufgabenteilung und asymmetrische Struktur
  - *Clients*: typischerweise Anwendungsprogramme und graphische Benutzungsschnittstelle (“front end”)
  - *Server*: zuständig für Dienstleistungen
- Typisches Kommunikationsparadigma: RPC

?? The customer sends a request to the waiter who executes him ??

http://perso.wanadoo.fr/lexiqueinformatique/

Sur mon site, vous trouverez tout le vocabulaire informatique que vous cherchez avec une définitions pour bien comprendre ce vocabulaire en Français, en Allemand et en Anglais.

Auf meiner Gegend werden Sie der ganze informations-Wortbestand finden, dass Sie mit ein Definitionen suchen, um diesen Wortbestand gut zu verstehen (einzuschließen).

Client-serveur: Une telle architecture décrit comment deux entités (le client et le serveur) peuvent communiquer sur un réseau. Le client envoie une requête au serveur, qui l'exécute et lui renvoie le résultat

Client-Kellner: Eine solche Architektur beschreibt wie zwei Wesenheiten (der Kunde und der Kellner) können in einem Netz kommunizieren. Der Kunde sendet eine Bitte (Antrag) dem Kellner, der ihn ausführt und ihm das Ergebnis zurücksendet.

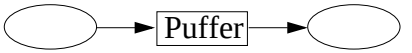
Eignung des Client/Server-Paradigmas

- Populär wegen des eingängigen Modells
  - entspricht Geschäftsvorgängen in unserer Dienstleistungsgesellschaft
  - gewohntes Muster --> intuitive Struktur, gute Überschaubarkeit
- Effizienz durch spezialisierte „Dienstleister“
  - grosszügige Ausstattung (CPU-Leistung, Speicherkapazität usw.)
  - bestückt mit spezieller Software (Datenbank etc.)
- Kosteneffektivität durch bessere Auslastung wertvoller Ressourcen bei “Compute Server”
  - Clients brauchen oft kurzfristig Spitzenleistung
  - einzelner Client kann Ressourcen aber nicht dauerhaft auslasten
- Passend für viele Kooperationsbeziehungen, z.B.
  - Client erbittet Auskunft von einem spezielisierten Service
  - gefährdete Clients geben wertvolle Daten in Obhut des (gegen Missbrauch, Verlust, Diebstahl usw.) hoch gesicherten Servers

---

- Modell ist für viele Zwecke geeignet, jedoch nicht für alle (z.B. Pipelines, “peer-to-peer”, asyn. Mitteilung)!

Beispiel Puffer als Pipeline:

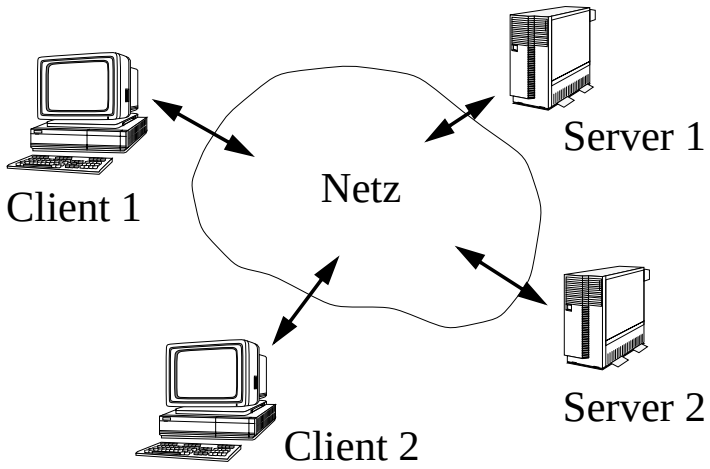


Produzent Konsument

- Puffer ist weder Client noch Server, sondern hat beide Rollen! (passiv gegenüber Produzent; aktiv gegenüber Konsument)
- Inversion der Kommunikationsbeziehung bei einem Puffer-Server!

Client- und Server-Maschinen

- Übertragung des Client/Server-Modells auf *Rechner*



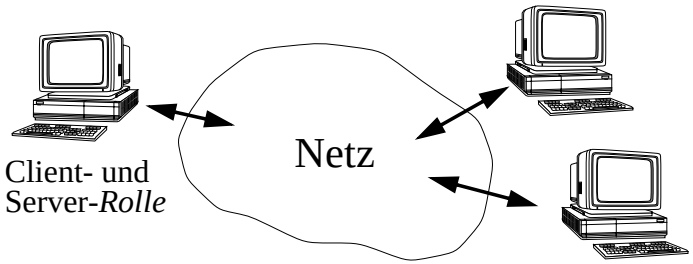
- Typischerweise PCs oder Workstations als Clients
  - u.a. mit graphischem Benutzungsinterface
- Andere Rechner als Server
  - "zentrale" Dienste (z.B. Speicherserver)
  - gemeinsam benutzte Betriebsmittel

- Im allgemeinen müssen sich aber Server- und Client-*Prozesse* nicht auf dedizierten Rechnern befinden!
- "Client/Server-Computing" wurde früher oft missbräuchlich als synonym zu "Distributed Computing" gebraucht (letzteres bezeichnet heute eher Peer-to-Peer-Strukturen!)

Peer-to-Peer-Strukturen

"Gleichrangiger"

- Im "Gegensatz" zum Client/Server-Modell

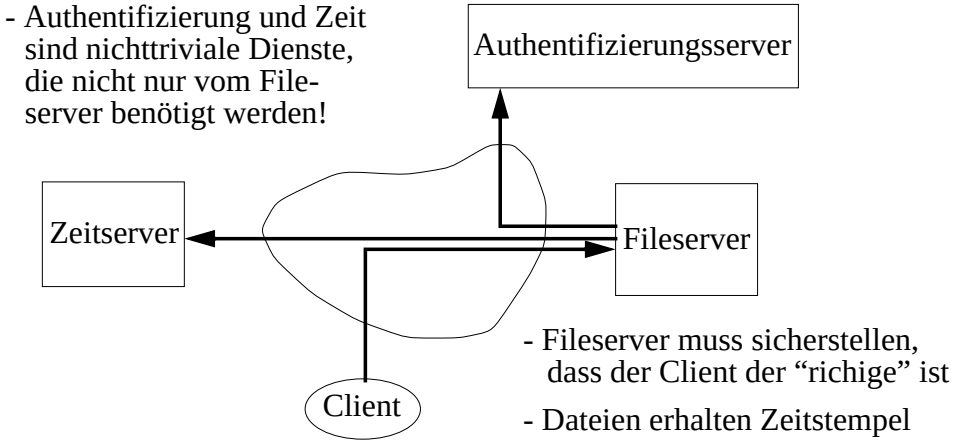


- Jeder Client fungiert zugleich als Server für seine Partner
  --> keine (teuren) dedizierten Server notwendig
  --> oft als Billiglösung von "echtem" Client/Server-Computing angesehen
- Ggf. können zusätzliche dedizierte Server existieren

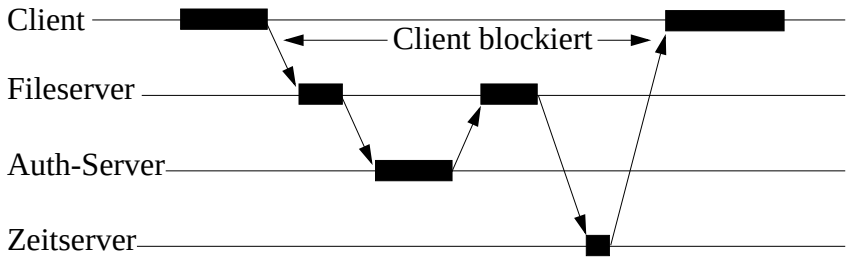
- *Nachteile*:
  - "Anarchischer" als *maschinenbezogene* Client/Server-Architektur
  - Rechner müssen leistungsfähig genug sein (cpu-Leistung, Speicherausbau), um für den "Besitzer" leistungstransparent zu sein
  - geringere Stabilität (Besitzer kann seine Maschine ausschalten...)
  - Datensicherung muss ggf. dezentral durchgeführt werden
  - Sicherheit und Schutz kritisch: Lizenzen, Viren, Integrität...

Client/Server-Rollen

- Server müssen ggf. zur Durchführung eines Dienstes die Dienstleistungen anderer Server in Anspruch nehmen



- Fileserver hat prinzipiell die Rolle eines Servers, zwischenzeitlich jedoch die Rolle eines Clients



Zustandsändernde /-invariante Dienste

(Vgl. frühere Diskussion bzgl. RPC-Fehlersemantik!)

- Verändern Aufträge den Zustand des Servers?
  - nur zwischenzeitlich (--> Atomizität) oder generell?
- Typische *zustandsinvariante* Dienste:
  - Auskunftsdienste (Name-Service; aktuelle Lastsituation;...)
  - Zeitservice
- Typische *zustandsändernde* Dienste:
  - Datei-Server

Idempotente Dienste / Aufträge

- Wiederholung eines Auftrags liefert gleiches Ergebnis

nicht zustandsinvariant!

- Beispiel: “Schreibe in Position 317 von Datei XYZ den Wert W”
- Gegenbeispiel: “Schreibe ans Ende der Datei XYZ den Wert W”
- Gegenbeispiel: “Wie spät ist es?” — aber zustandsinvariant!

Wiederholbarkeit von Aufträgen

- Bei Idempotenz oder Zustandsinvarianz kann bei Verlust des Auftrags (timeout beim Client) dieser erneut abgesetzt werden (--> einfache Fehlertoleranz)

Zustandslose / -behaftete Server

stateless — statefull

- Hält der Server Zustandsinformation über Aufträge hinweg? ("session")
  - z.B. (Protokoll)zustand des Clients
  - z.B. Information über frühere damit zusammenhängende (Teil)aufträge
- Aufträge an zustandslose Server müssen autonom sein

- *Beispiel*: Datei-Server

```
open("XYZ");
read;
read;
close;
```

In klassischen Systemen hält sich das Betriebssystem Zustandsinformation über die Position des Dateizeigers geöffneter Dateien (in UNIX ausserdem i-node-Nummer etc.)

- bei zustandslosen Servern entfällt open/close; jeder Auftrag muss vollständig beschrieben sein (Position des Dateizeigers etc.)
- zustandsbehaftete Server daher i.a. effizienter (notw. Zustandsinformation ist beim Client)
- Dateisperren sind bei echten zustandslosen Servern nicht (einfach) möglich
- zustandsbehaftete Server können wiederholte Aufträge erkennen (z.B. durch Speichern von Sequenznummern) --> Idempotenz

- *Crash* eines Servers: Weniger Probleme im zustandslosen Fall (--> Fehlertoleranz)! (entscheidender Vorteil!)

- NFS (Network File System von Sun) ist zustandslos
- RFS (Remote File System von UNIX System V) ist zustandsbehaftet

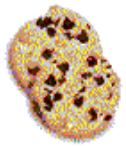
Sind WWW-Server zustandslos?

- Beim HTTP-Zugriffsprotokoll wird über den Auftrag hinweg keine Zustandsinformation gehalten
  - jeder Hyperlink, den man anklickt, löst eine neue "Transaktion" aus
- Stellt ein Problem beim Online-Commerce dar
  - gewünscht sind Transaktionen über mehrere clicks hinweg und
  - Wiedererkennen von Kunden (beim nächsten Klick oder Tage später)
  - erforderlich z.B. für Realisierung von "Einkaufskörben" von Kunden
  - gewünscht vom Marketing (Verhaltensanalyse von Kunden)

Lösungsmöglichkeiten (für Einkaufskörbe im WWW)

- IP-Adresse des Kunden an Auftrag anheften
  - Problem: Proxy-Server --> viele Kunden haben gleiche IP-Adresse
  - Problem: dynamische IP-Adressen --> keine Langzeitwiedererkennung
- "tag propagation"
  - Einstiegsseite eine eindeutige Nummer anheften, wenn der Kunde diese erstmalig aufruft
  - diese Nummer jedem link anheften und mit zurückübertragen
- Cookies
  - erstmalig im Netscape Navigator 1.0.1 verwendet
  - Textdatei < 4k, die ein Server einem Browser (= Client) schickt und die im Browser gespeichert wird
  - nur der Sender des Cookies darf dieses später wieder lesen

Cookies



Auszug aus
http://home.netscape.com/newsref/std/cookie_spec.html:

Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications.

A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store... Any future HTTP requests made by the client... will include a transmittal of the current value of the state object from the client back to the server. The state object is called a cookie, for no compelling reason.

This simple mechanism provides a powerful new tool which enables a host of new types of applications to be written for web-based environments. Shopping applications can now store information about the currently selected items, for fee services can send back registration information and free the client from retyping a user-id on next connection, sites can store per-user preferences on the client, and have the client supply those preferences every time that site is connected to.

A cookie is introduced to the client by including a Set-Cookie header as part of an HTTP response... The expires attribute specifies a date string that defines the valid life time of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out... A client may also delete a cookie before it's expiration date arrives if the number of cookies exceeds its internal limits.

- Denkübung: Müssen Proxy-Server geeignete Massnahmen vorsehen?
- Übung: Man finde heraus, was doubleclick.net macht (und wie)

Cookies (2)

- Anwendung von cookies war und ist umstritten (Ausspionieren des Verhaltens); dazu kam eine gewisse Paranoia:

  http://www.cookiecentral.com/creport.htm

  http://www.ciac.org/ciac/bulletins/i-034.shtml

  The Energy Department's Computer Incident Advisory Capability (CIAC) recently issued a report on cookie technology and its use on the web....

  The report stressed that there's a sense of paranoia involved with cookies, cookies cannot harm your computer or pass on private information such as an email address without the user's intervention in the first place. Paranoia has recently been sparked by one rumour involving AOL's new software, it claimed that AOL were planning to use cookies to obtain private information from users hard drives.

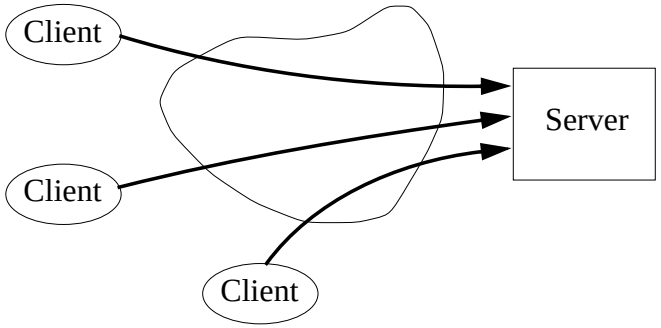
- Problemlos ist das allerding nicht:

  http://www.cookiecentral.com/cookie5.htm

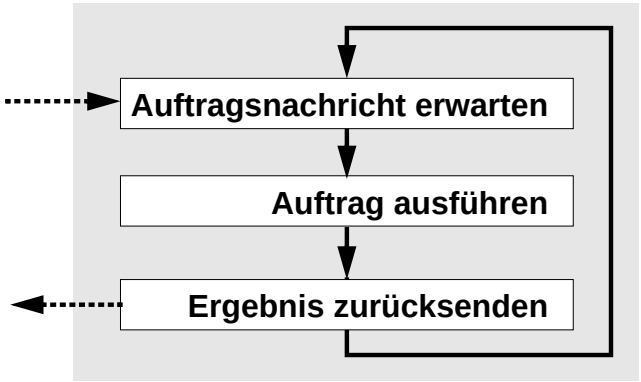
  Unfortunately, the original intent of the cookie has been subverted by some unscrupulous entities who have found a way to use this process to actually track your movements across the Web. They do this by surreptitiously planting their cookies and then retrieving them in such a way that allows them to build detailed profiles of your interests, spending habits, and lifestyle... it is rather scary to contemplate how such an intimate knowledge of our personal preferences and private activities might eventually be used to brand each of us as members of a particular group.

Iterative Server

- Problem: Viele “gleichzeitige” Aufträge



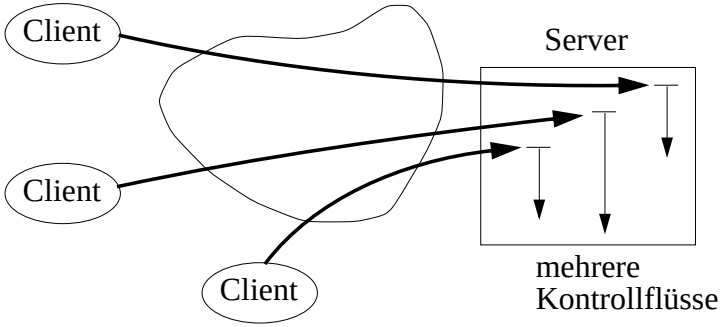
- *Iterative Server* bearbeiten nur einen Auftrag pro Zeit



- häufige Bezeichnung: “single threaded”
- eintreffende Anfragen während Auftragsbearbeitung: abweisen, puffern oder einfach ignorieren
- einfach zu realisieren
- bei “trivialen” Diensten sinnvoll (mit kurzer Bearbeitungszeit)

Konkurrente (“nebenläufige”) Server

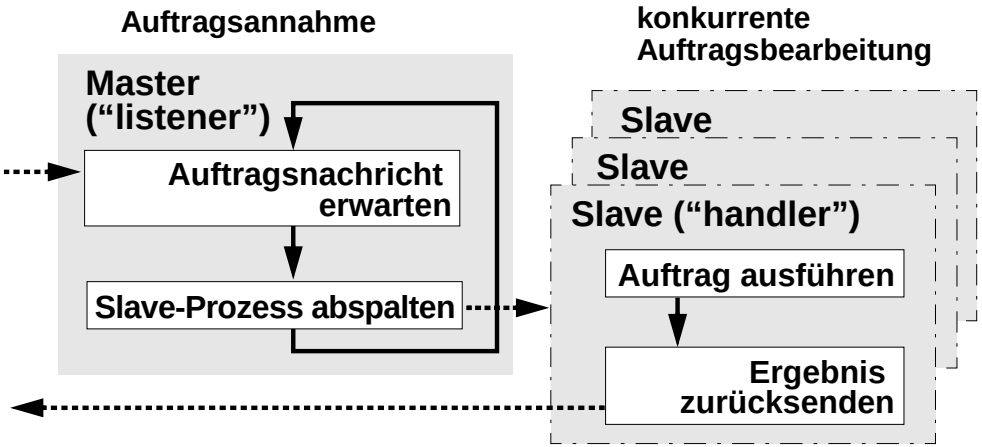
- Gleichzeitige Bearbeitung mehrerer Aufträge
  - sinnvoll (d.h. effizienter für Clients) bei langen Aufträgen (z.B. in Verbindung mit E/A)
  - Beispiel: Web-Server oder Suchmaschinen



- Ideal bei Mehrprozessormaschinen (physische Parallelität)
  - aber auch bei Monoprozessor-Systemen (vgl. Argumente bei Timesharing-Systemen): Nutzung erzwungener Wartezeiten eines Auftrags für andere Jobs; kürzere mittlere Antwortzeiten bei Jobmix aus langen und kurzen Aufträgen
- Interne Synchronisation bei konkurrenten Aktivitäten sowie ggf. Lastbalancierung beachten
- Verschiedene denkbare Realisierungen, z.B.
  - mehrere Prozessoren
  - Verbund verschiedener Server-Maschinen (z.B. LAN-Cluster)
  - dynamische Prozesse (bei Monoprozessor-Systemen)
  - dynamische threads
  - feste Anzahl vorgegründeter Prozesse
  - internes Scheduling und Multiprogramming

Konkurrente Server mit dynamischen Handler-Prozessen

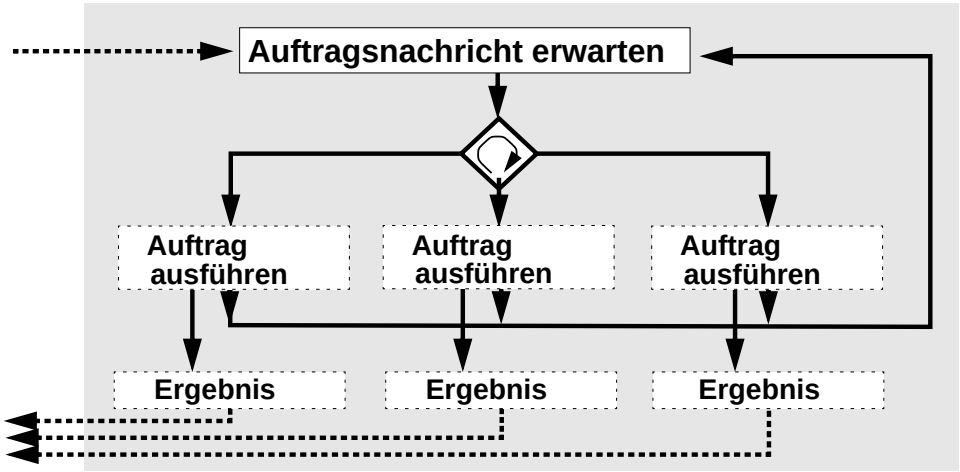
- Für jeden Auftrag gründet der *Master* einen neuen *Slave*-Prozess und wartet dann auf einen neuen Auftrag
  - neu gegründeter Slave (“handler”) übernimmt den Auftrag
  - Client kommuniziert dann ggf. direkt mit dem Slave (z.B. über dynamisch eingerichteten Kanal bzw. Port)
  - Slaves sind ggf. Leichtgewichtsprozesse (“thread”)
  - Slaves terminieren i.a. nach Beendigung des Auftrags
  - die Anzahl gleichzeitiger Slaves sollte begrenzt werden



- Alternative: “Process preallocation”: Feste Anzahl statischer Slave-Prozesse
  - ggf. effizienter (u.a. Wegfall der Erzeugungskosten)
- Übungsaufgaben:
  - herausfinden, wie es bei Web-Servern gemacht wird (z.B. Apache)
  - wie sollte man bei grossen WWW-Suchmaschinen vorgehen?

Quasi-konkurrente Server

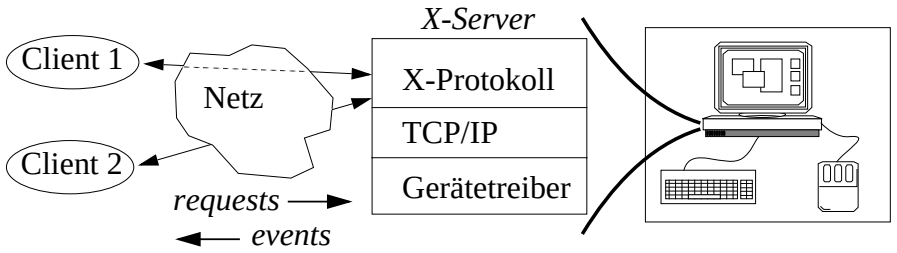
- Server besteht aus einem *einzigen Prozess*, der im Multiplexmodus mehrere Aufträge verschränkt abarbeitet
  - ggf. sinnvoll, wenn z.B. Clients grosse Datenmengen “stückweise” senden und die Wartezeiten dazwischen für die Bearbeitung der anderen Aufträge verwendet werden kann



- Keine Neugründung von Slave-Prozessen
- Keine Adressraumgrenzen zwischen Auftragsdaten
  - keine kostspieliger Kontextwechsel
  - auftragsübergreifende gemeinsame Datenhaltung effizienter (vgl. X-Server: Alle Clients (z.B. xclock) schreiben Display-Daten in einen gemeinsamen “Display-Puffer”)
- Potentielle Nachteile: kein Adressraumschutz zwischen verschiedenen Aufträgen; ggf. unnötige Wartezeiten z.B. bei blockierenden Betriebssystemaufrufen

“X-Window” als Client/Server-Modell

- Erstes netzwerkunabhängiges Graphik- und Fenstersystem für seinerzeit neue pixelorienterte Bildschirme
  - entwickelt Mitte der 80er Jahre am MIT, zusammen mit der Firma DEC



- i.a. bedient ein Server mehrere Client-Prozesse (“Applikationen”), die ihre Ausgabe auf dem gleichen Bildschirm erzeugen
- *Window-Manager*: Spezieller Client, der Grösse und Lage der Fenster und Icons steuert (Beispiele: twm, mwm, fvwm)

→ X windows system protocol (über TCP)

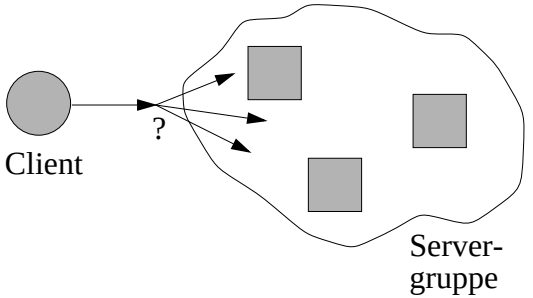
- *Requests*: Service-Anforderung an den X-Server (z.B. Linie in einer bestimmten Farbe zwischen zwei Koordinatenpunkten zeichnen); zugehörige Routinen stehen in einer Bibliothek (*Xlib*)
- *X-Library* (Xlib) ist die Programmierschnittstelle zum X-Protokoll; damit manipuliert ein Client vom Server verwaltete Ressourcen (Window, font...); höhere Funktionen (z.B. Dialogboxen) in einem (von mehreren) X-Toolkit
- *Events*: Tastatur- und Mauseingaben (bzw. -bewegungen) werden vom X-Server asynchron an den Client des “aktiven Fensters” gesendet (keine klassische Server-Rolle --> schwierig mit RPCs zu realisieren!)
- X ist ein *verteiltes System*: Client-Prozesse können sich auf verschiedenen Rechnern befinden (“Fenster verschiedener Rechner”)
- *X-Terminal* hat Server-Software im ROM bzw. lädt sie beim Booten (heute gegenüber PC preislich kaum ein Vorteil, vgl. auch “Web-Terminal”)
- vielfältige Standard-*Utilities* und *Tools* (xterm, xclock, xload...)

Servergruppen und verteilte Server

- Idee: Ein Dienst wird nicht von einem einzigen Server, sondern von einer Gruppe von Servern erbracht

a) *Multiple Server*

- Jeder einzelne Server kann den Dienst erbringen
- Zweck:
  - *Leistungssteigerung* (Verteilung der Arbeitslast auf mehrere Server) ← “Lastverbund”
  - *Fehlertoleranz* durch Replikation (Verfügbarkeit auch bei vereinzelten Server-Crashs) ← “Überlebensverbund”

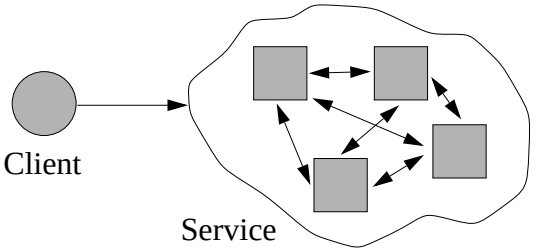


- z.B. Multicast an eine Server*gruppe*
- oder Auswahl eines *bestimmten* Servers (z.B. nahegelegenen oder wenig belasteten)

b) *Kooperative Server*

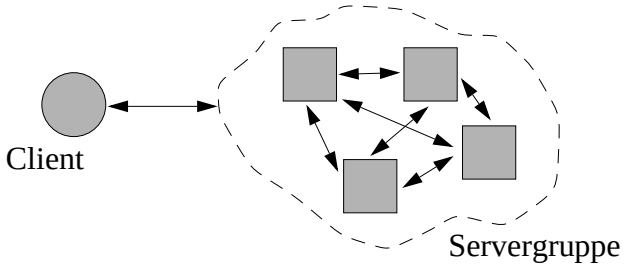
← “Know-how-Verbund”

- ein Server allein kann den Dienst nicht erbringen

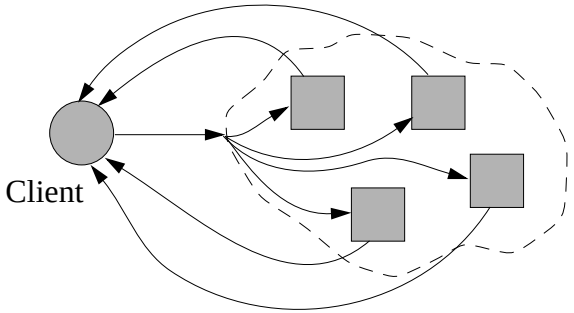


Strukturen kooperativer Server

1) Echte Kooperation: Server liefern gemeinschaftlich ein Gesamtergebnis

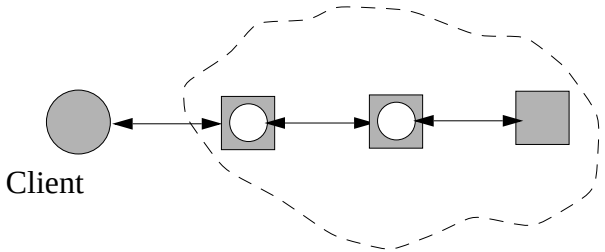


2) Paarweise Kooperation mit dem Client: Client akkumuliert Teilergebnisse



Z.B.: Broadcast stösst einzelne Rück-meldungen an und Client "akkumuliert" die Information

3) Kaskadierung: Dienst als Menge von Teildiensten realisiert, z.B.:



- Server während der Auftragsbearbeitung als Client bzgl. Teilaufträgen

Beispiel: ruptime

- "remote uptime" (vgl. UNIX-Kommando "uptime")
- UNIX-Kommando, das einen verteilten Dienst im LAN implementiert
- heute jedoch nicht mehr aktuell (durch andere Dienste ersetzt)

```
NAME
     ruptime - show host status of local machines

     ruptime gives a status line like uptime for each machine  on
     the  local  network; these are formed from packets broadcast
     by each host on the network once a minute.

     Machines for which no status report has been received for  5
     minutes are shown as being down.

BUGS
     Broadcasting does not work through gateways.
```

Router etc.

```
sol[52] [~] ruptime
cadsun        up 34+12:39,     0 users,  load 1.28, 1.28, 1.06
hssun2      down     1:21
martine       up  5+10:55,     0 users,  load 0.10, 0.05, 0.04
nuriel        up  5+11:04,     0 users,  load 0.11, 0.11, 0.11
octopus       up  5+10:43,     0 users,  load 0.02, 0.04, 0.03
paloma        up  5+07:10,     0 users,  load 0.00, 0.08, 0.06
quantas       up  5+10:52,     0 users,  load 0.00, 0.02, 0.02
rtracer       up 39+13:18,     4 users,  load 2.05, 1.21, 0.52
salamander    up  2+05:18,     0 users,  load 0.00, 0.06, 0.06
sol           up  1+06:27,    11 users,  load 5.12, 5.12, 5.12
```

verschiedene Maschinen

Der rwhod-Dämon

- “Dämon”: Service, der auf das Auftreten von Ereignissen wartet, und dann darauf reagiert; wird i.a. bei Systemstart gegründet.

```
NAME
     rwhod - system status server

DESCRIPTION
     rwhod is the server which maintains the database used by the
     rwho(1C) and ruptime(1C) programs.

     rwhod operates as both a producer and consumer of status
     information.  As a producer of information it periodically
     queries the state of the system and constructs status mes-
     sages which are broadcast on a network.  As a consumer of
     information, it listens for other rwhod servers' status mes-
     sages,...

     Status messages are generated approximately once every 60
     seconds.

BUGS
     This service takes up progressively more network bandwidth
     as the number of hosts on the local net increases. For
     large networks, the cost becomes prohibitive.
```

- kein eigentliches Client/Server-Modell!
- aus Performance-Gründen oft deaktiviert
- Neuimplementierung (‘rup’ statt ‘ruptime’): kein default-Broadcast, sondern nur Broadcast bei Aufruf des Kommandos; rstatd-Dämonen anderer Rechner antworten dann

kernel statistics server

Das rup-Kommando

```
DESCRIPTION
     rup gives a status similar to uptime for remote machines.
     It broadcasts on the local network, and displays the
     responses it receives.

     Normally, the listing is in the order that responses are
     received.

BUGS  Broadcasting does not work through gateways.
```

```
sun10    up          11:56,    load average: 2.01, 2.01, 1.96
sun33    up 10 days,  2:51,    load average: 0.98, 1.00, 1.01
sun72    up           9:26,    load average: 0.21, 0.25, 0.30
sun13    up  1 day,  10:29,    load average: 0.02, 0.04, 0.04
sun14    up          15:24,    load average: 0.10, 0.05, 0.04
sun45    up  1 day,  11:07,    load average: 0.00, 0.02, 0.04
sun16    up 22 days,  9:36,    load average: 0.07, 0.02, 0.03
sun17    up          15:29,    load average: 0.02, 0.05, 0.05
sun18    up  2 days, 15:15,    load average: 0.01, 0.01, 0.01
sun19    up  2 days, 15:31,    load average: 0.84, 0.37, 0.21
sun20    up 10 days, 15:17,    load average: 0.00, 0.02, 0.05
sun27    up  9 days, 15:21,    load average: 1.00, 1.05, 1.07
sun18    up 14 days, 13:37,    load average: 0.09, 0.08, 0.07
sun31    up 65 days, 12:42,    load average: 0.04, 0.03, 0.05
sun34    up 23 days,  3:15,    load average: 0.02, 0.02, 0.02
sun56    up  2 days, 15:06,    load average: 0.00, 0.02, 0.04
sun57    up 22 days,  9:03,    load average: 0.02, 0.04, 0.04
sun58    up  3 days,  8:34,    load average: 0.00, 0.01, 0.03
sun59    up  3 days, 15:22,    load average: 0.05, 0.05, 0.04
sun60    up          15:23,    load average: 0.00, 0.02, 0.03
sun61    up 31 days,  7:10,    load average: 0.01, 0.03, 0.04
```