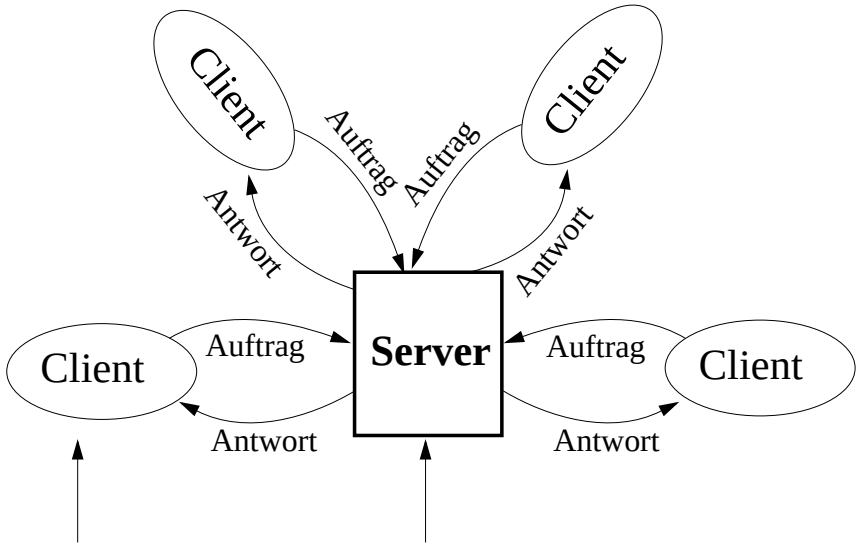
Das Client/Server-Modell



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

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“
  - verfügen ggf. über Spezialhardware
  - grosszügige Ausstattung (CPU-Leistung, Speicherkapazität usw.)
  - bestückt mit Spezialsoftware (Datenbank etc.)
- Kosteneffektivität durch bessere Auslastung wertvoller Ressourcen
  - Clients brauchen oft kurzfristig Spitzenleistung
  - einzelner Client kann Ressourcen aber nicht dauerhaft auslasten
- Passend für viele Kooperationsbeziehungen, z.B.
  - schwacher Client erbittet Hilfe vom „grossen Bruder“
  - gefährdete Clients geben wertvolle Daten in Obhut des (gegen Missbrauch, Brand, Diebstahl usw.) hoch gesicherten Servers

---

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

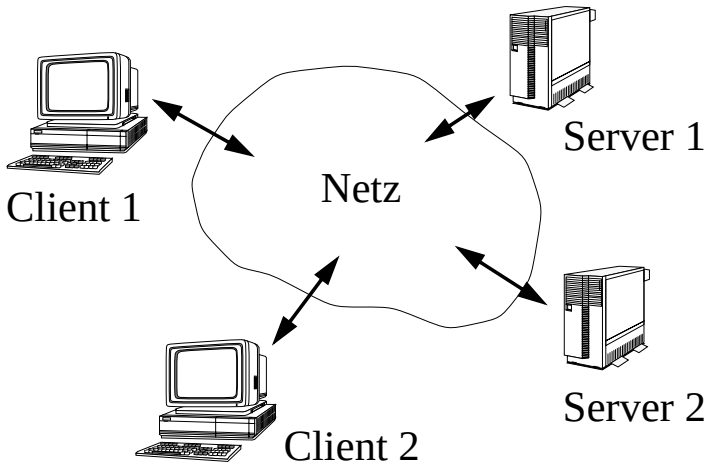
Beispiel Puffer als Pipeline:

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*



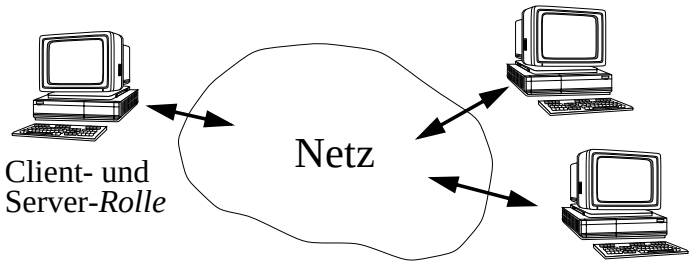
- Typischerweise PCs oder Workstations als Clients
  - u.a. mit graphischem Benutzungsinterface
- Andere Rechner als Server
  - "zentrale" Dienste (z.B. Plattenserver)
  - gemeinsam benutzte Betriebsmittel

- Im allgemeinen müssen sich aber Server- und Client-*Prozesse* nicht auf dedizierten Rechnern befinden!
- "Client/Server-Computing" wird heute in der Praxis oft missbräuchlich als synonym zu "Distributed Computing" gebraucht
  - beachte: Kern der Rechenleistung wird *zentral* auf dem Server erbracht!

Peer-to-peer-Strukturen

"Gleichrangiger"

- Im "Gegensatz" zum Client/Server-Modell

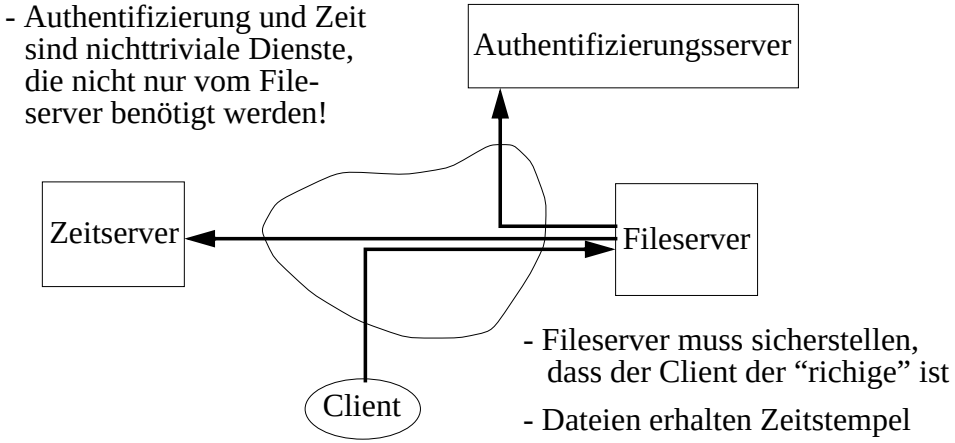


- Jeder Client fungiert zugleich als Server für seine Partner
  --> keine (teuren) dedizierten Server notwendig
  --> oft als Billiglösung von "echtem" Client/Server-Computing angesehen
- Ggf. können zusätzliche dedizierte Server existieren

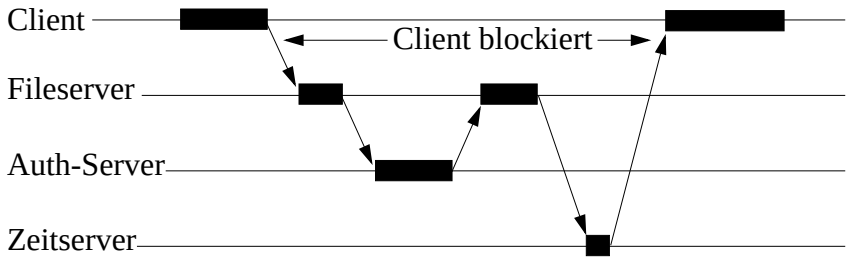
- *Nachteile*:
  - "Anarchischer" als *maschinenbezogene* Client/Server-Architektur
  - Rechner müssen leistungsfähig genug sein (cpu-Leistung, Speicherausbau), um für den "Besitzer" leistungstransparent zu sein
  - geringere Stabilität (Besitzer kann seine Maschine ausschalten...)
  - Datensicherung 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

"session"

- Hält der Server Zustandsinformation über Aufträge?
  - 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.)

notw. Zustandsinformation ist beim Client

- zustandsbehaftete Server daher i.a. effizienter
- 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