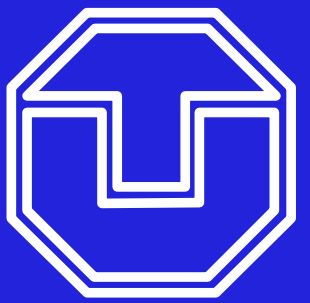
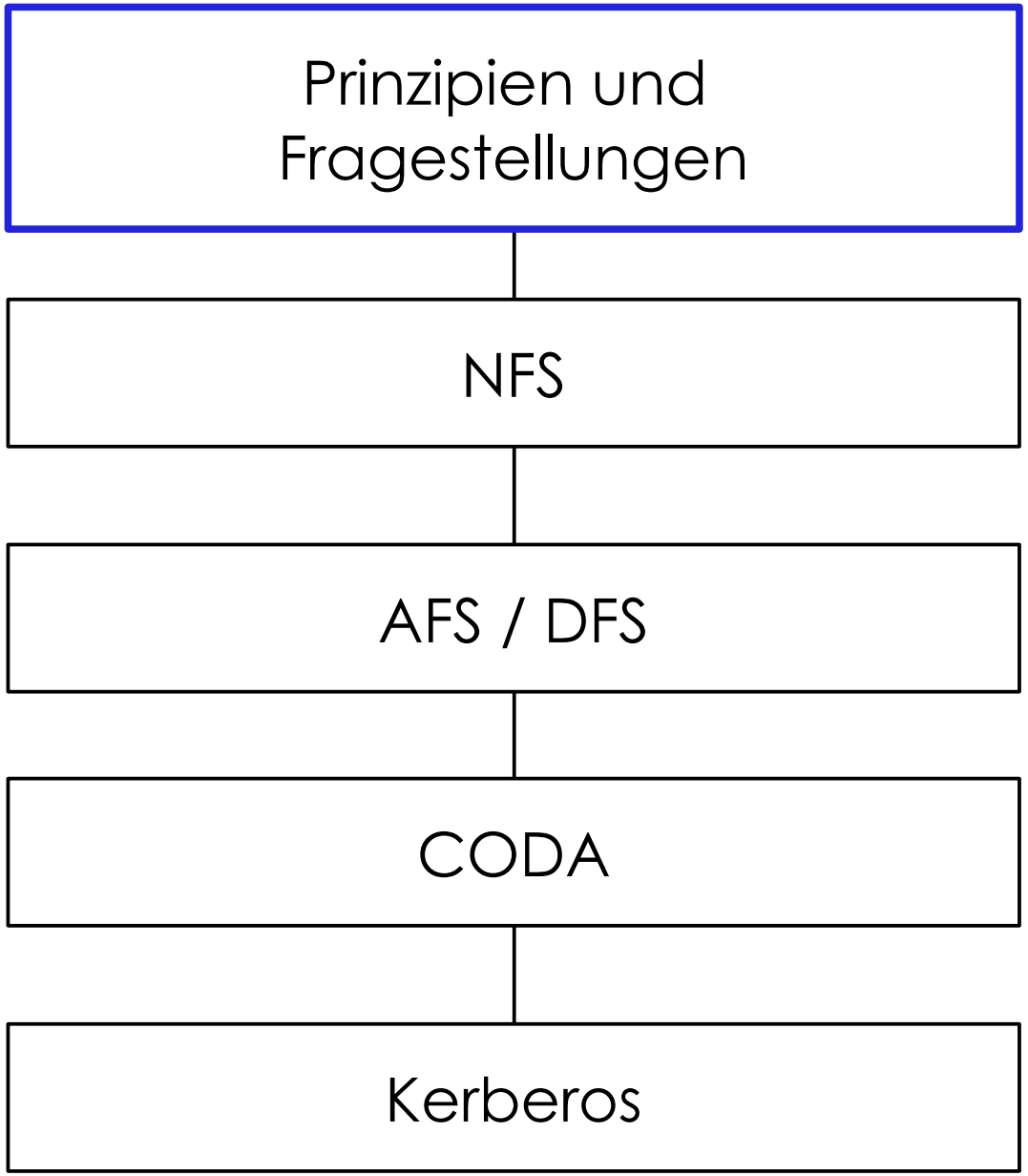
Verteilte Dateisysteme Und Kerberos

Eng nach:

Hermann Härtig
TU Dresden

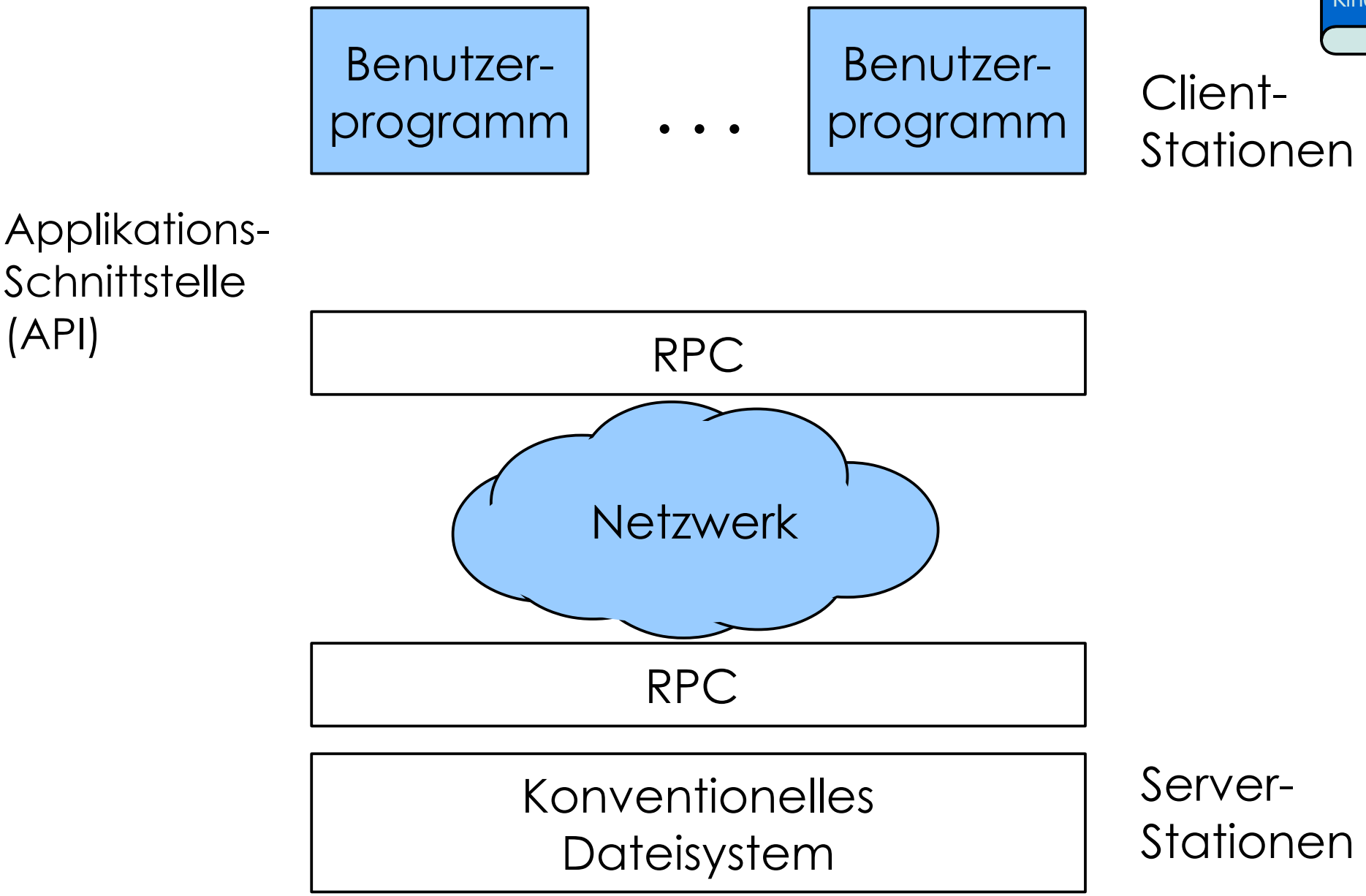


Wegweiser



Naheliegend: einfach Prozedur durch RPC ersetze

Coulouris Dollimore Kindberg



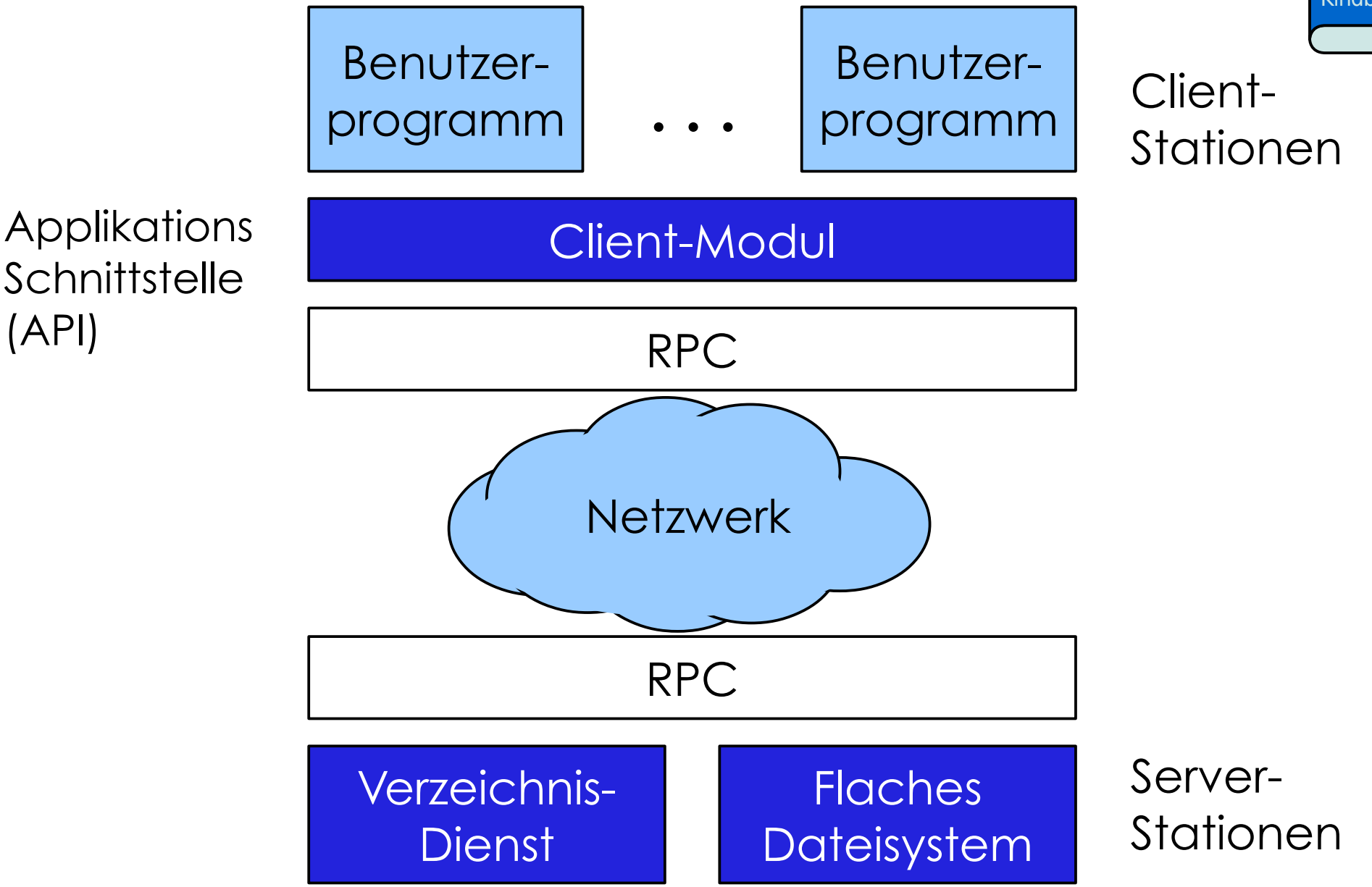
Nachteile der einfachen Variante

Coulouris
Dollimore
Kindberg

- Leistungsfähigkeit mangelhaft:
  - zu viele Botschaften
  - Overhead bei kleinen Lese-/Schreib-Einheiten zu groß
- Fehlersemantik nicht berücksichtigt
- nicht offen:
  - unterschiedliche Client-Betriebssysteme
  - unterschiedliche Namensformen für Dateisysteme

Architektur verteilter Dateisysteme

Coulouris
Dollimore
Kindberg



Aufgaben

Coulouris
Dollimore
Kindberg

Client-Modul:

- Anpassung an Client-Betriebssystem
- Organisation des Zusammenspiels von Verzeichnisdienst und flachem Dateisystem
- Fehlerbehandlung
- Caching

Flaches Dateisystem:

- Operationen zum Lesen, Schreiben, Erzeugen, Löschen ... persistenter Dateien auf einer (flachen) Menge von Dateien
- Zugriffssteuerung (Access Control)

Namensdienste (Directory Services):

- Abbildung symbolischer Namen auf „Unique File Identifiers“

Fragestellungen und Ziele bei Entwurf (1)

Coulouris
Dollimore
Kindberg

- Zugriff
  gleichartiger Zugriff auf nahe und entfernte Dateien
- Ort
  Grad der Unabhängigkeit von Netztopologie
- Fehler
  Grad der Sichtbarkeit für Benutzer
- Replikation
- Skalierbarkeit
  Anpassbarkeit an sehr große Zahlen von Clients und Servern
- Heterogenität
  unterschiedliche Basis-Hardware und Betriebssystem

Fragestellungen und Ziele bei Entwurf (2)

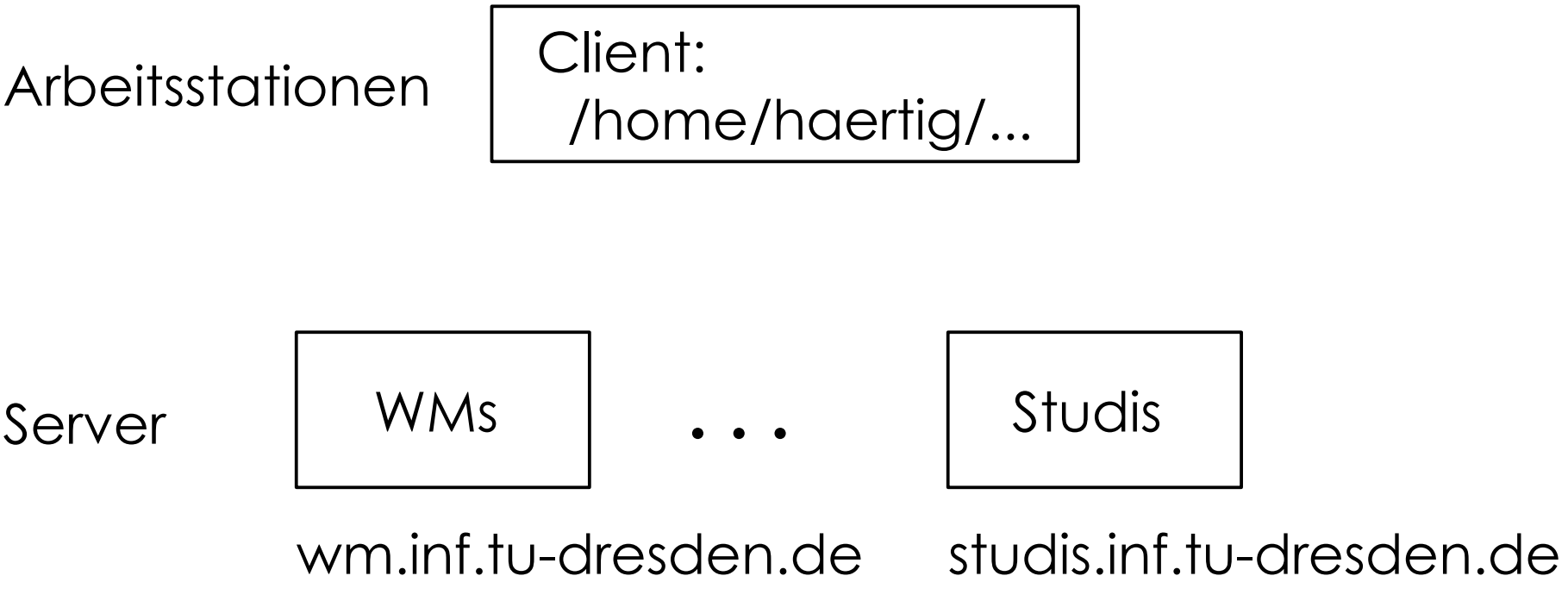
Coulouris
Dollimore
Kindberg

- Schreib-Konsistenz
  Sichtbar-Werden bei parallelen Schreiboperationen
- Effizienz
  Bandbreite, Latenz
- Administration
  zentral / dezentral
- Echtzeitfähigkeit
  Zusagen für Bandbreite („Quality of Service“)
- Schutz
  Angreifermodell, Schutzziele (→ eigenes Kapitel)

Namen in verteilten Dateisystem (1)

Coulouris
Dollimore
Kindberg

- Grundsätzlicher Ausgangspunkt:
  hierarchische Namensräume (Pfadnamen)



Namen in verteilten Dateisystem (2)

Coulouris
Dollimore
Kindberg

Variante 1: Globale Namen

- Servernamen (und Benutzernamen) werden Bestandteil des Dateinamens
- eingeschränkte Ortstransparenz
  (man muss Servernamen kennen)
- skaliert weltweit
- Anker für andere Namensschemata

Beispiele

```
scp  wm.inf.tu-dresden.de:/home/otto/tmp/bla   .
scp  Heinrich@Stud.inf.tu-dresden.de:tmp/bla   .
cp   /../WM/home/ss10    .
```

Namen in verteilten Dateisystem (3)

Coulouris
Dollimore
Kindberg

Variante 2: Client-Station oder -Prozess baut Namensraum

- Namen danach ortsunabhängig
- *Clients* können Dateien (transparent) migrieren
- keine Zusammenfassung verschiedener DS so, dass sie zusammen wie ein Verzeichnis aussehen („unionizing“)
- Bsp.: entferntes „Montieren“
  mount WM.inf.tu-dresden.de:/home /home/haertig/wm/

Beispiele

/home/haertig/wm/Otto, /home/haertig/Stud/Heinrich

→ aber nicht:

/home/haertig/Otto, /home/haertig/Heinrich

Namen in verteilten Dateisystem (4)

Coulouris
Dollimore
Kindberg

Variante 3: Darstellung eines einheitlichen Bildes

- für alle Clients
- ortsunabhängige Namen
- *Administratoren* der Server können Dateien migrieren

Beispiel

- Serverstruktur:
  {WM1, WM2, ... , Wmn}.inf.tu-dresden.de
- Namen:
  /TUD/home/Heinrich, /TUD/home/Otto

Namen in verteilten Dateisystem (5)

Serv1.inf.tu-dresden.de:

/user/anton/.... /user/eva/....
/user/evi/... /user/zeppelin/...

Serv2.inf.tu-dresden.de:

/user/martin/....
/user/henriette/...

Variante1: Serv1.inf.tu-dresden.de:/user/anton/...

Variante 2: /user/Serv1/anton, /user/Serv2/henriette/...

Variante 3: /user/anton/..., /user/henriette/...

Flache Benennung („Unique File Identifier“)

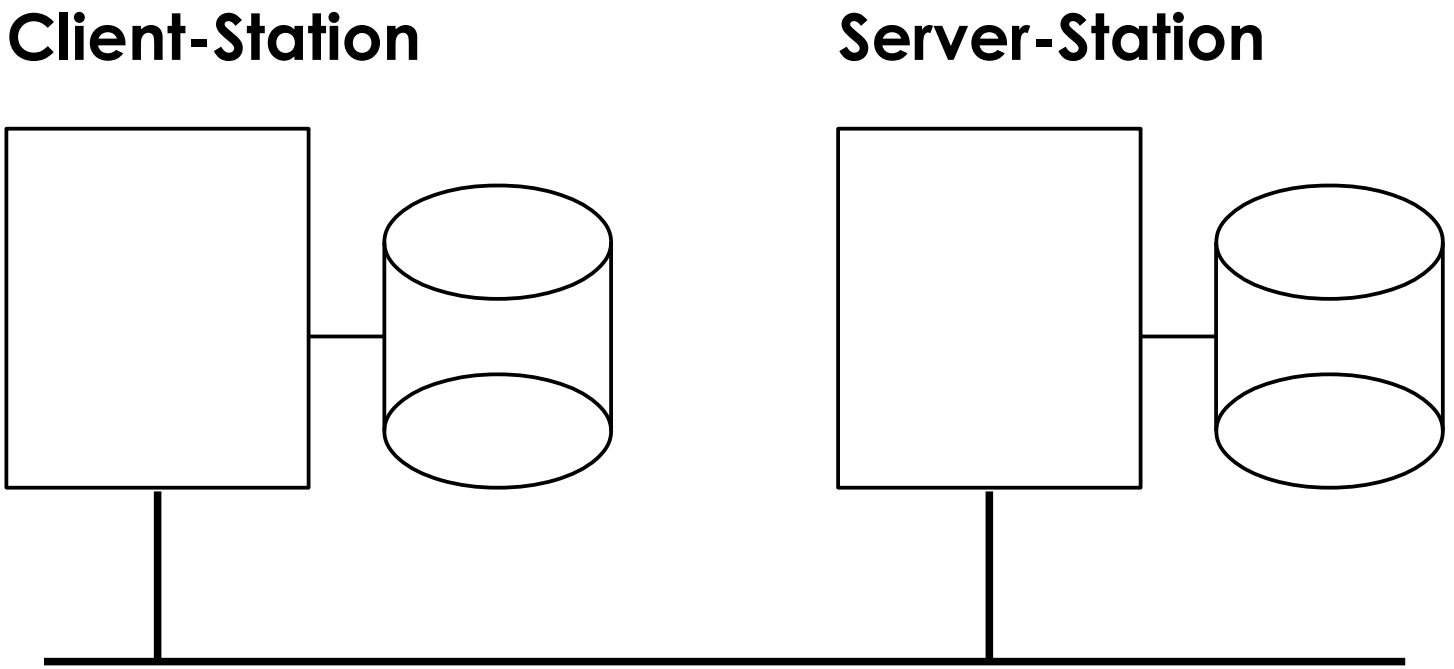
Coulouris
Dollimore
Kindberg

Aufgaben/Kriterien

- Finden: UFId → Adresse
  - für jede Datei
  - für Dateigruppen (Menge von Dateien, die nur als Ganzes migriert)
- Eindeutigkeit:
  - zentrale Autorität
  - große Zufallszahl
- Fälschungssicherheit
- erlaubte Operationen

Caching: Wo?

Coulouris
Dollimore
Kindberg



Fragestellungen

- Persistenz des Caches nach Abstürzen
- Zugreifbarkeit der Dateien bei Netzpartitionierungen
- Effizienz

Caching: Was?

Coulouris
Dollimore
Kindberg

In welchen Einheiten?

- Variante 1: kein Cache
  jede Operation durch RPC
- Variante 2: ganze Dateien
  - vollständige Übertragung bei erstem Zugriff
  - Speicherung auf Client-Platte
  - Zurückschreiben bei close
- Variante 3: Blöcke
  asynchrones Zurückschreiben

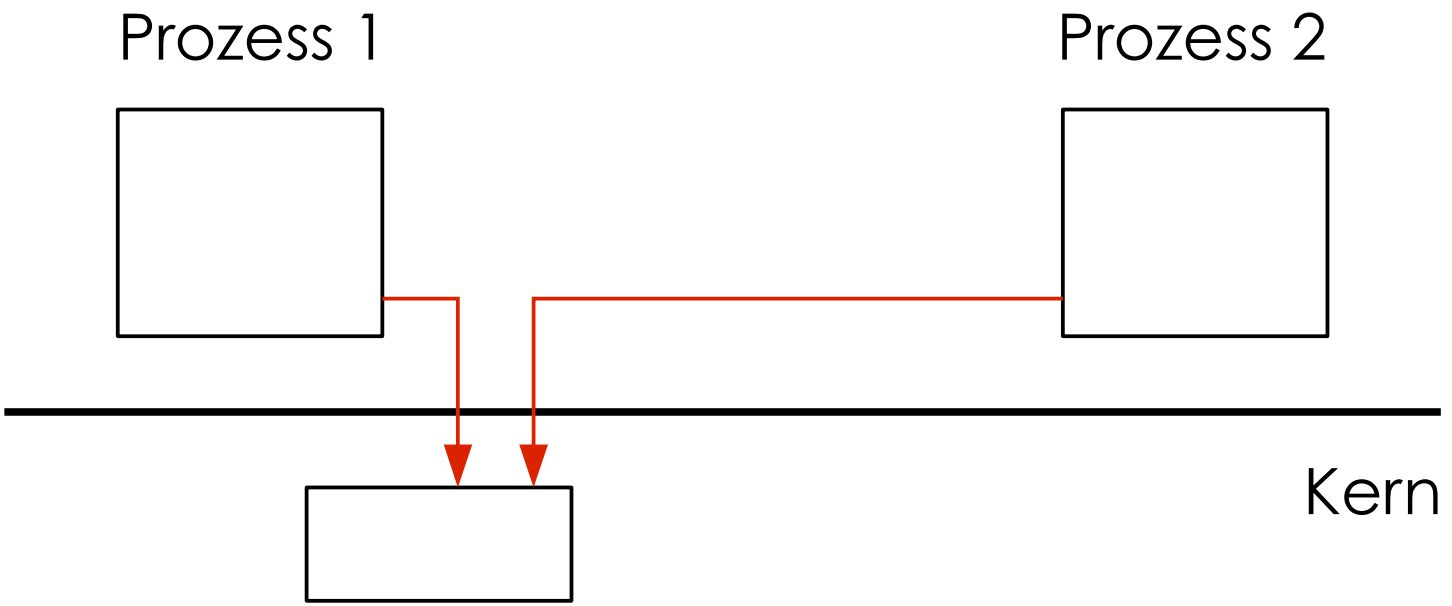
Schreib-Konsistenz (1)

Coulouris
Dollimore
Kindberg

Variante 1: „one copy semantics“

- Jeder schreibende Zugriff wird in allen Prozessen in derselben Reihenfolge gesehen

➔ z. B. Unix-Dateisystem lokal



Schreib-Konsistenz (2)

Coulouris
Dollimore
Kindberg

Variante 2: Sitzungssemantik („session semantics“)

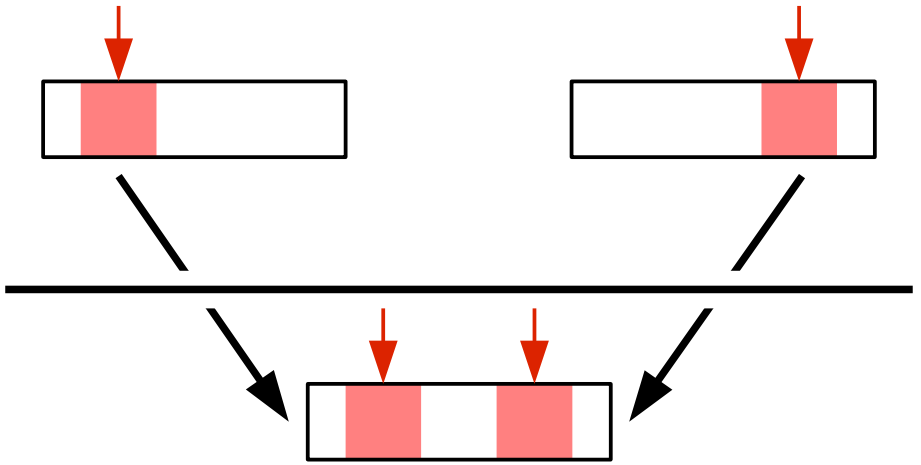
- Sitzung: open (read/write)* close
- jeder Prozess sieht:
  - die von ihm verursachten Änderungen innerhalb einer Sitzung
  - die von anderen Prozessen verursachten Änderungen
    - bei Beginn der eigenen Sitzung (beim Öffnen)
    - nach dem Ende der Sitzung anderer Prozesse

Schreib-Konsistenz (3)

Coulouris
Dollimore
Kindberg

Variante 3: keine Konsistenz

auch: Überschreibung benachbarter Bereiche
„False Sharing“



➔ bei Zurückschreiben des ganzen Blocks wird eine Schreiboperation ungültig

Variante 4: Bindung an Synchronisationsoperationen

- Sperren von Bereichen
- „Commit“-Operationen (explizites Schreiben)

„Stateful“ vs. „Stateless“ (1)

- Zustandsinformationen über Klienten bei Servern?
  - ja: stateful
  - nein: stateless
- Beispiel klientenspezifischer Zustand:
  - Dateiposition
  - Identifier

stateful: read (fd, buf, count)

stateless: read (UFId, position, buf, count)

„Stateful“ vs. „Stateless“ (2)

Vorteile (nach Tanenbaum)

stateless	stateful
Fehlertoleranz	kürzere request-Botschaften
keine OPEN/CLOSE – Operation nötig	effizienter
keine aufwendigen Tabellen auf Server	Vorauslesen möglich
keine Limitierung der Zahl offener Dateien	Nicht-idempotente Operation möglich
keine Probleme bei Klienten-Abstürzen	File-locking möglich

Replikation

Coulouris
Dollimore
Kindberg

Mehrere Kopien von Dateien

- Lesen: von einer Kopie
- Schreiben: alle Kopien

Motivation

- Effizienz: Zugriff auf weniger belastete Server
- Zuverlässigkeit/Verfügbarkeit: Tolerierung von
  - Serverausfällen
  - Netzpartitionierung

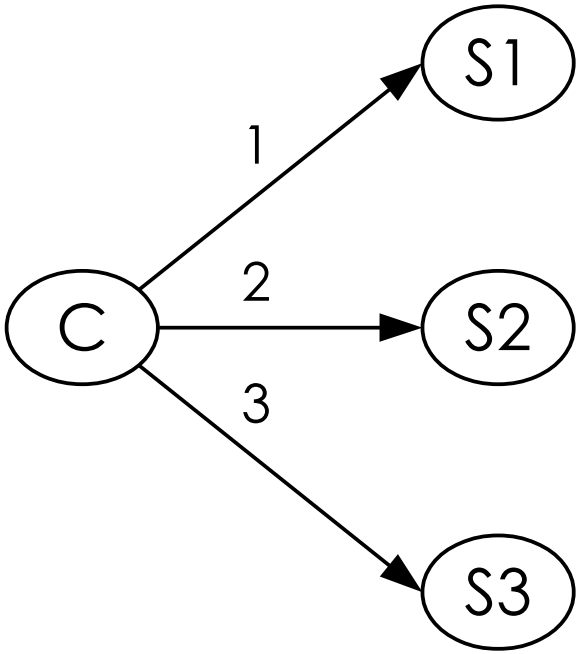
Replikationsmodelle (1)

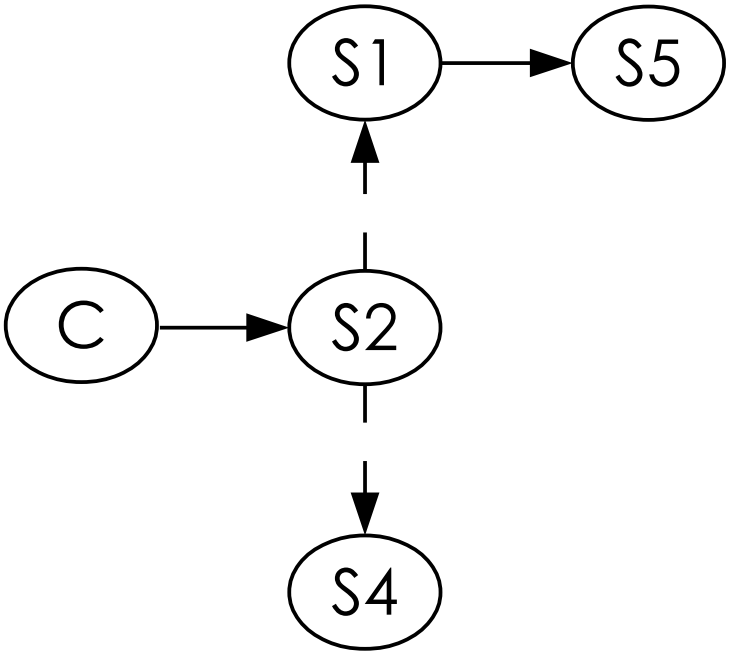
Coulouris
Dollimore
Kindberg

Clientorganisierte Replikation

Serverorganisierte Replikatio träge („lazy replication“)

- grundsätzlich vs. nur im Fehlerfall



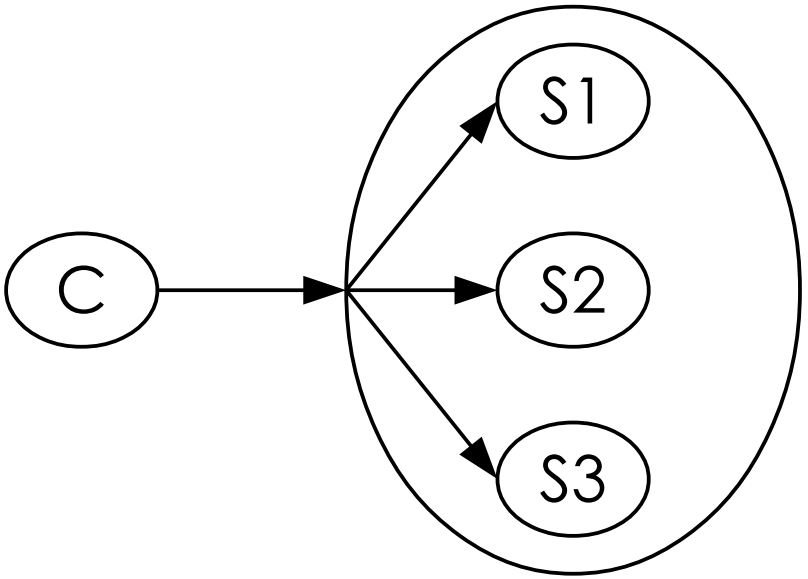


→ sofort – – ► später

Replikationsmodelle (2)

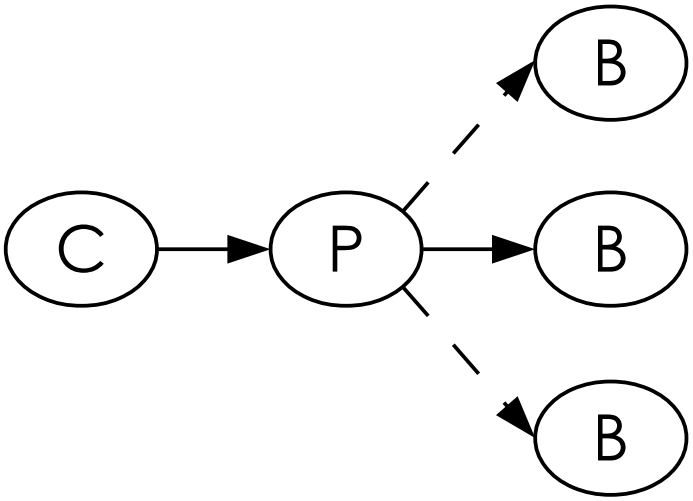
Coulouris
Dollimore
Kindberg

gleichberechtigte Gruppenbildung



- Server organisiert Replikation sofort auf der Basis von Multicast-Kommunikation

„Primary-Backup“-Verfahren



Fehlerfälle:

- Gruppenmitglied fällt aus
- Primary fällt aus
  (Bully Algorithmus → LV „Verteilte Systeme“)

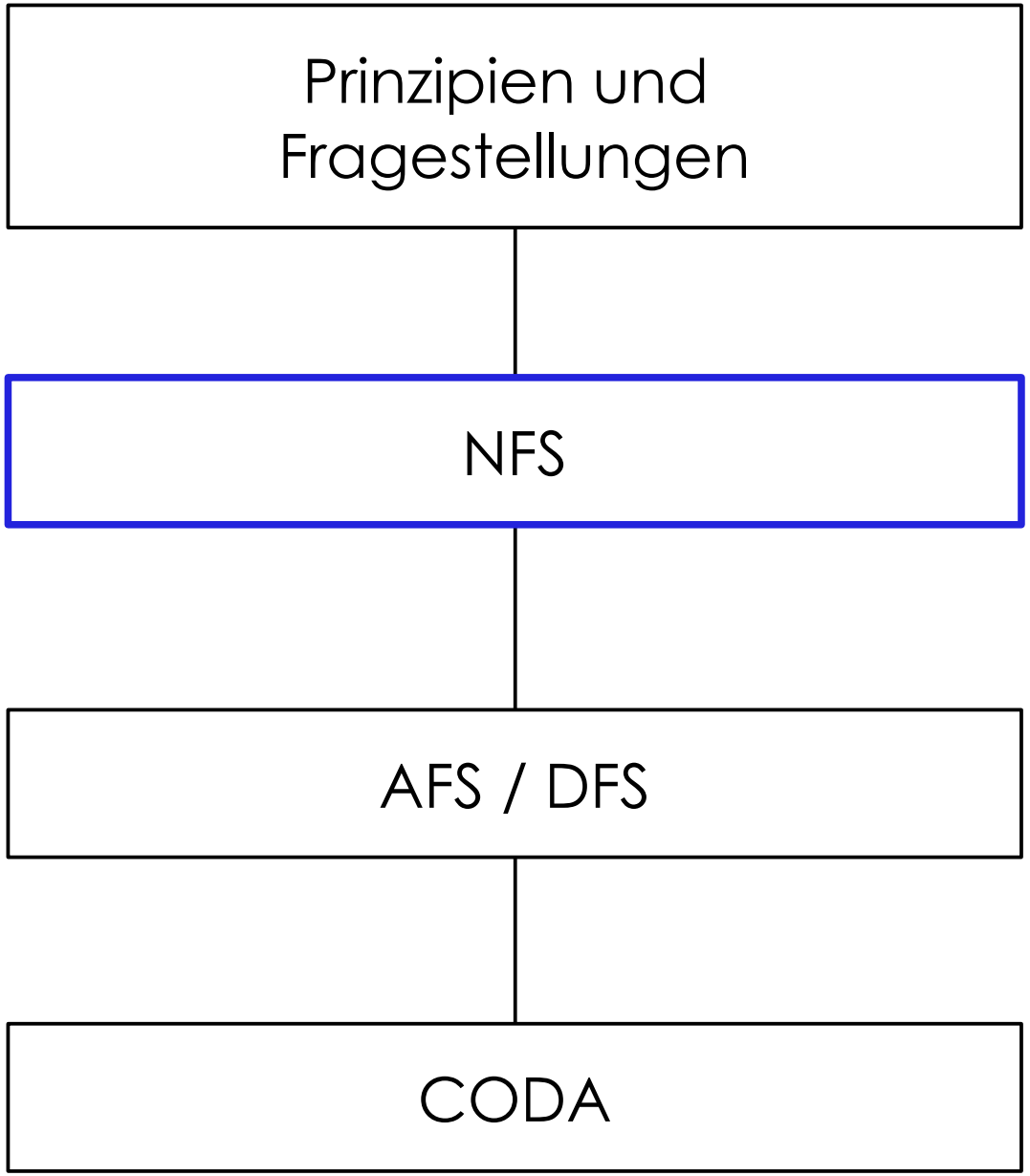
Replikationsmodelle

- Replikation: pessimistisch vs. optimistisch
  - pessimistisch:
    nur dann schreiben, wenn alle Replikate erreichbar sind
  - optimistisch:
    schreiben:
    auch wenn eine Teilmenge nicht erreichbar ist
    lesen:
    Konfliktauflösung (z.B.: Voting, kausal jüngstes, …)

→ Werkzeuge/Systeme zur Unterstützung von Replikation

Isis, Horus, Ensemble ... → LV „Verteilte Systeme“
Coda → nächstes Kapitel

Wegweiser



Fallbeispiel 1: NFS

Coulouris
Dollimore
Kindberg

- NFS (Network File System),
  UC Berkeley / Sun Microsystems, 1985

Entwurfsziele

- transparenter Zugriff auf entfernte Dateien
  - keine besonderen Bibliotheken (Unix-System-Calls)
- Ortstransparenz
- Heterogenität hinsichtlich Hardware und Betriebssystem

Protokolle

- NFS-Protokoll (Program 100003, Version 2/3)
- Mount-Protokoll (100005, 2/3)
- Network Lock Manager (100021,4)

Anwendersicht (API)

Coulouris Dollimore Kindberg

- Server: exportiert Dateisysteme (DS)
  - /etc/exports: enthält Liste von DS
  - Mount-Rechte der Clients: ACL
- Clients: "mounten" exportiertes DS im eigenen Namensraum
  - manche Implementierungen auch beliebigen Teilbaum davon
  - üblicherweise: System-Start-Sequenz
- Zugriff transparent über Pfadnamen
  - API (Schnittstelle des Client-Moduls): betriebssystemeabhängig (nicht standardisiert)

Beispiel

Coulouris
Dollimore
Kindberg

- /etc/exports auf irz301:

  /var/spool/mail irz101(rw) irz102 (rw) irz103 (rw)

- /etc/rc auf client:

  mount -t nfs irz301:/var/spool/mail /var/mail

- ls /var/mail

Unix-Schnittstellen

Coulouris
Dollimore
Kindberg

- Open (API)
  - ➔ parsiert Pfadnamen (lokal)
  - bei jedem Teilnamen:
    prüfen ob Mount-Punkt erreicht
  - bei Erreichen eines NFS-Mount-Punktes für jeden weiteren Teilnamen:
    NFS-Protokoll: lookup Dirfh, Name → fh
  - am Ende des Pfadnamens:
    Unix erzeugt Unix-FileDescriptor (fd) und merkt sich: Unix-fd → NFS-fh
- Read/Write (fd)
  - falls NFS-Datei: fd → fh
  - NFS-Protokoll: read(fh, offset, ...)
  - Kopieren ...

Automounting

Coulouris
Dollimore
Kindberg

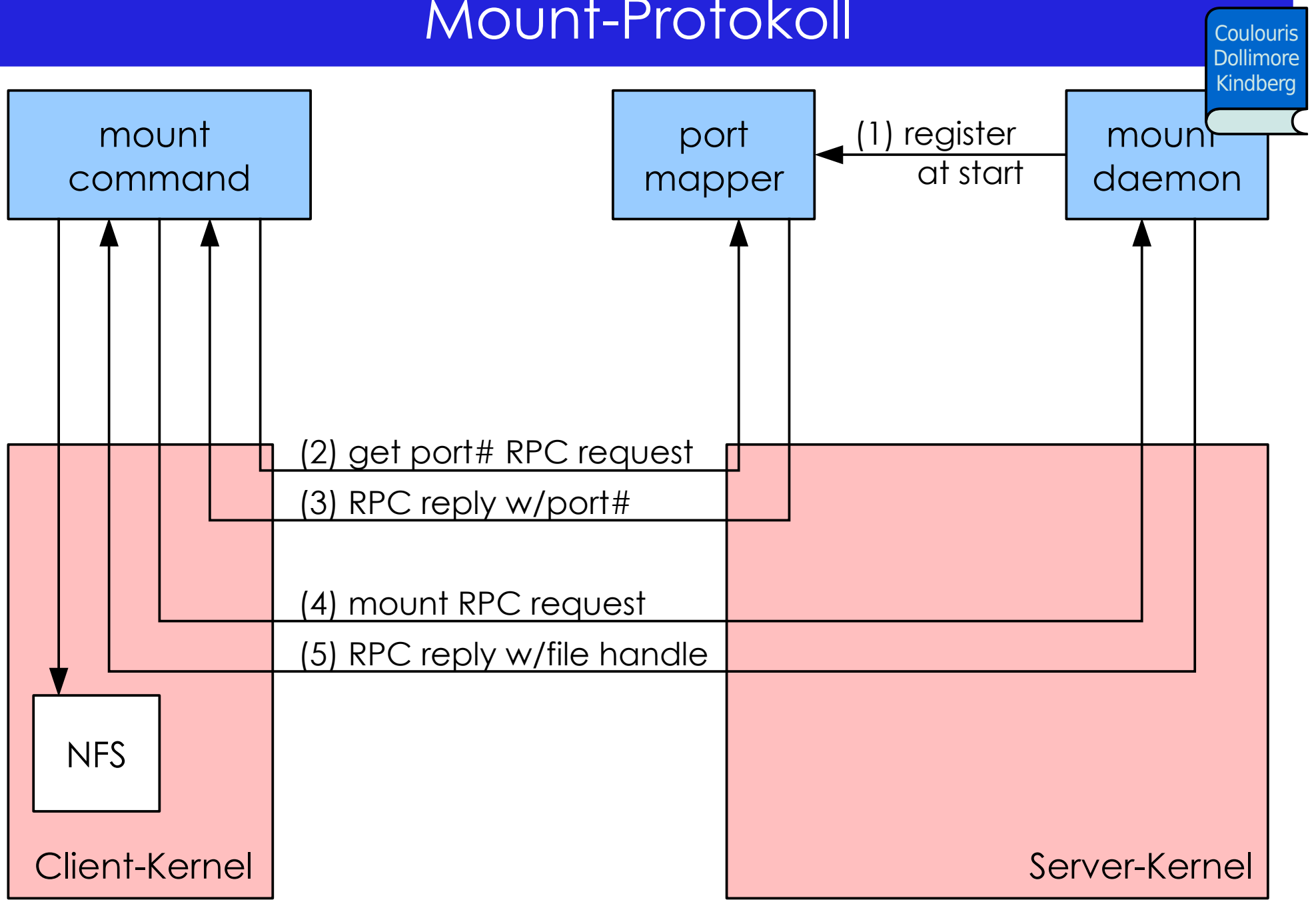
- verzögertes Montieren bei erstem Zugriff
- mehrere Server können angegeben werden

Vorteile

- Klient kann hochgefahren werden, wenn einer der zu montierenden Server nicht verfügbar ist

- einer von mehreren Servern kann genutzt werden
  schwache Art von Replikation

Mount-Protokoll



NFS-Protokoll

Coulouris
Dollimore
Kindberg

Dateiattribute

- Type
- Mode
- Uld, Gld
- Number of Links
- File System Id
- File Id

- Atime - letzter Zugriff
- Ctime - letzte Modifikation eines Dateiattributes (I-Node)
- Mtime - letzte Modifikation der Datei

NFS-Protokoll

Coulouris
Dollimore
Kindberg

File Handle (fh)

- ➔ Privater Typ der Server-Implementierung
- Unix:
  - File-System-Id
  - I-Knoten-Nummer
  - I-Knoten-Generationsnummer

NFS-Protokoll

Coulouris
Dollimore
Kindberg

Auszug (insgesamt 21 Operationen)

- lookup(dirfh, name) → (fh, attr)

  suche nach Einzelnamen (nicht Pfadnamen) in angegebenem Verzeichnis liefert File Handle und Dateiattribute

- read (fh, offset, count) → (attr, data, eof, count)

  offset expliziter Parameter

- idempotente Operationen

NFS-Protokoll

Coulouris
Dollimore
Kindberg

- write (fh, offset, count, stable) → (result, pre-op-attr, post-op-attr, committed)
  - stable: file-sync oder unstable
  - committed: Daten auf Platte

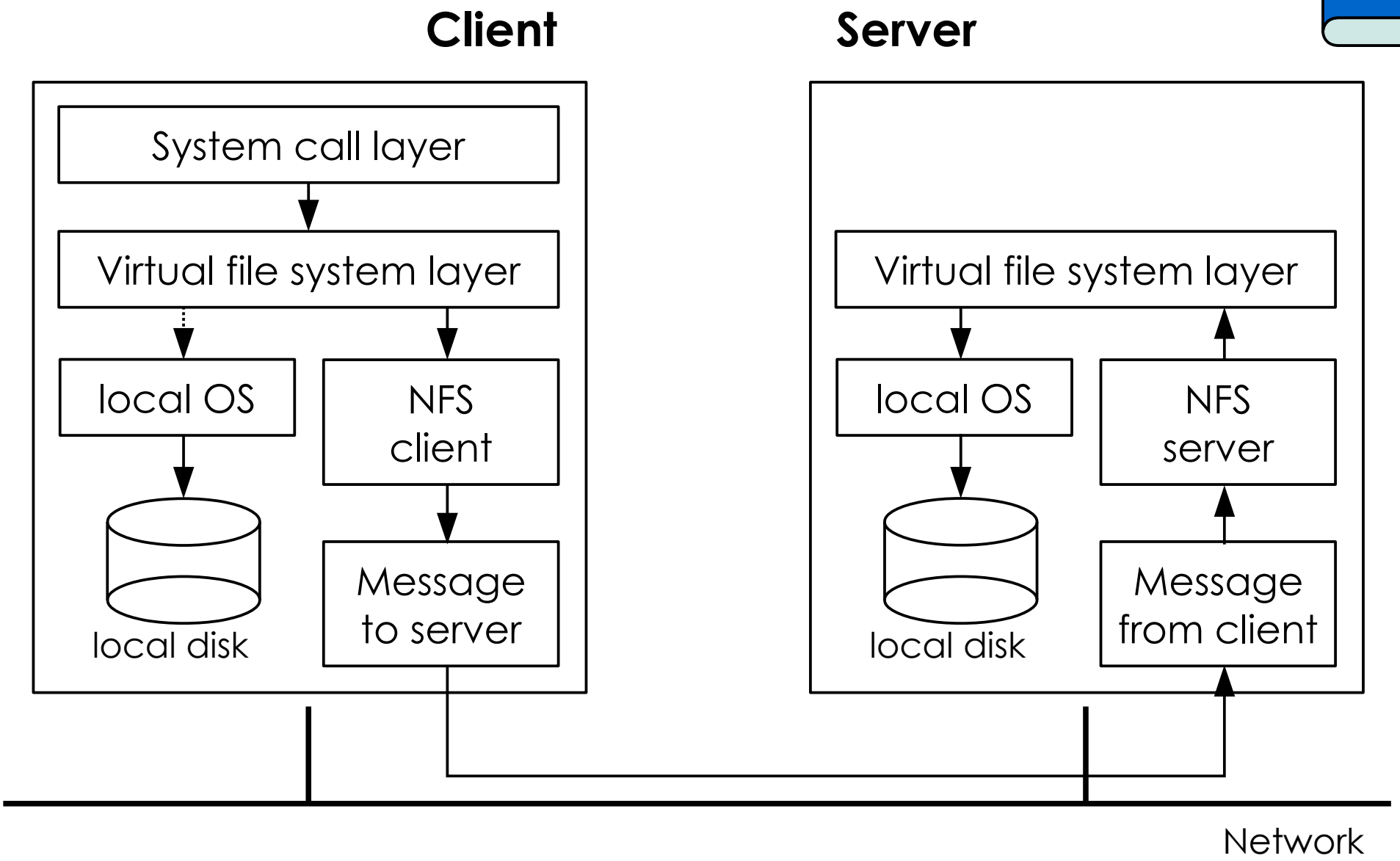
- commit (fh, offset, count) → (pre-op-attr, post-op-attr, verf)
  - verf: Neustart nach letztem write

Bemerkungen

- Operationen idempotent
- keine Zustandsinformation über "offene" Dateien (stateless)
- keine Synchronisationsoperationen (lock)

Unix-Implementierung (1)

Coulouris
Dollimore
Kindberg



Unix-Implementierung (2)

Coulouris
Dollimore
Kindberg

- Client-Modul (und Server-Programm) sind in Kern integrie
  - ➔ daher keine Recompilation von Anwendungsprogrammen notwendig
- v-node für jede offene Datei auf Klientenstation
- r-node auf Serverstation
- mount
  - parsiert Pfadnamen
  - beschafft
    - Filehandle fh
    - filesystem Id
    - i-node number
    - i-node generation number
  - erzeugt v-node und r-node

Unix-Implementierung (3)

Coulouris Dollimore Kindberg

- open
  - parsiert Pfadnamen (lokal)

    bei jedem Teilnamen:

    prüfen ob Mount-Punkt erreicht

  - bei Erreichen eines NFS.mount-Punktes für einen Teilnamen:

    lookup an Server → fh

  - Ende des Pfadnamens:

    Unix erzeugt Unix-FileDescriptor (fd) und erzeugt

    v-node sowie r-node für entfernte

    i-node für lokale Datei

Unix-Implementierung (4)

Coulouris Dollimore Kindberg

- read
  - prüft, ob Block in lokalem Speicher
  - falls nicht, read an Server (ganzen Block)
  - NFS-Server nutzt normales Dateisystem auf Server
  - „read ahead“

- write
  - schreibt zunächst in Client-Puffer
  - sammelt mehrere writes
  - asynchrones Auslösen der Schreib-Operationen an Server
  - ➔ Version 2: Schreib-Operationen erst dann abgeschlossen, wenn Daten auf Platte („write through cache“)

Fehlermodell

Coulouris
Dollimore
Kindberg

- Toleriert werden sollen
  - transiente Fehler
  - Server-Abstürze
  - ➔ nicht aber:
    permanente Fehler auf Speichermedium

- NFS ist „stateless“:
  - ➔ Server braucht keinen Client-spezifischen Zustand für korrektes Funktionieren
  - ➔ aber: aus Effizienzgründen kann Zustand gehalten werden

Umgang mit Fehlern

Coulouris
Dollimore
Kindberg

- Anwendung blockiert, bis Server wieder verfügbar
  - Operationen werden solange wiederholt, bis gelungen
  - nutzt „at least once“ RPC-Semantik des SUN RPC
  - am häufigsten auftretende Operationen sind idempotent

„hard“ vs. „soft“ mounting

Bei Serverausfall oder Netz-Partitionierung wird aufrufender Prozess

- suspendiert, falls „hard mounted“
- abgebrochen, falls „soft mounted“

Caches in NFS

Coulouris
Dollimore
Kindberg

Client-Seite

- Datei-Attribute (i-node) (auch über close-Operation hinaus)
- Ergebnisse von lookup-Operation
- Daten-Blöcke
  asynchrones, verzögertes Schreiben an Server

  ➔ Inkonsistenz möglich bei mehreren schreibenden Clients

Daten-Blöcke auf Server-Dateisystem

- Puffer write through (NFS2)
- explizites commitment

Konsistenz

Coulouris Dollimore Kindberg

- Schwache Konsistenz (weak consistency)

Vorgehen

- Zeitattribute (atime, ctime, mtime)
  pre-op-attr, post-op-attr für schreibende Operationen

- Client prüft Cache-Gültigkeit in „freshness“-Intervallen:
  - bei Datenblöcken alle 3 Sekunden
  - bei Verzeichnis-Blöcken alle 30 Sekunden
  - beim Öffnen einer Datei

- alle Pufferinhalte werden regelmäßig alle 30 Sekunden zum Server geschrieben

Cache validity

Coulouris Dollimore Kindberg

T current time

Tc time of last validation (from client at server)

Tm time of last modification (client/server)

t freshness interval (e.g. 3 s)

Cache is considered valid if:

(T – Tc < t) OR (Tmclient = Tmserver)

Zugriffsschutz und Authentfikation

Coulouris
Dollimore
Kindberg

- Basiert wie Unix auf Zugriffssteuerlisten (ACL)

- Drei Varianten:
  - Uld und Gld Teil jeder Botschaft im Klartext
  - Uld und Gld per DES verschlüsselt
  - Kerberos

- Erste Versionen:

  jeder mit physischem Netz-Zugriff oder Root-Rechten auf Arbeitsstation kann Pakete mit „geeigneter“ Uld/Gld zusammenbauen

- Daten werden grundsätzlich nicht verschlüsselt

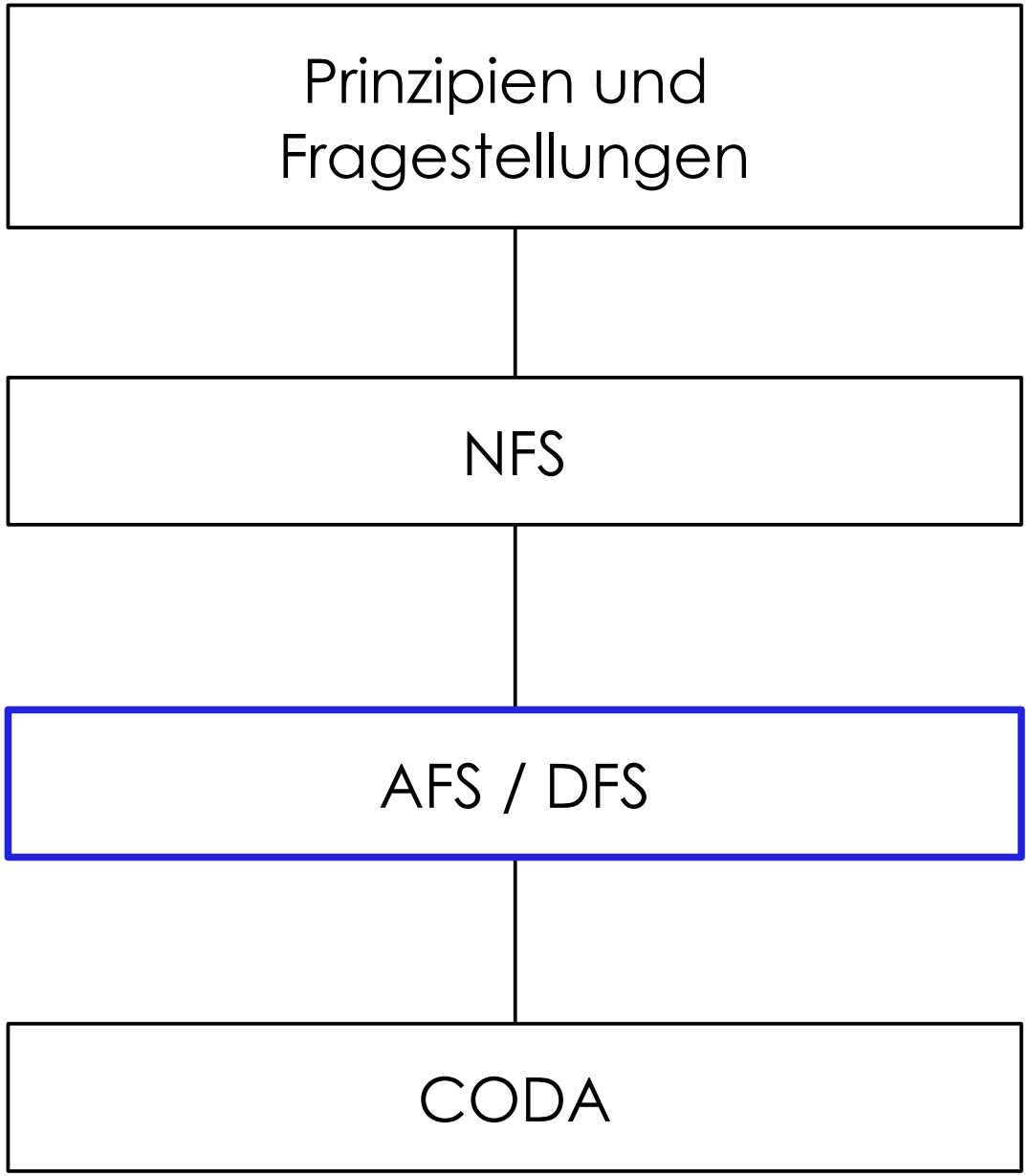
Bewertung NFS

Coulouris
Dollimore
Kindberg

- Leistungsfähigkeit: www.spec.org
- Lehrbuchangaben:
  AIX, 5 x 1 Gbps Ethernet, 24 x 450 MHz RS64,
  289 Platten, 4 Controller, 4 - 8 ms Latenz
  ca. 29.000 Serveroperationen
- Skalierbarkeit !?
- einfache, klare Schnittstelle
- sehr erfolgreich!
- Protokoll ist Internet-Standard (RFC 1094)
- auf vielen Betriebssystemen verfügbar
  z. B. Unix …, VMS, Mainframes, DOS, Mac OS, Windows
- freie Implementierungen verfügbar

Wegweiser

Coulouris
Dollimore
Kindberg



Fallbeispiel 2: AFS/DFS

Coulouris
Dollimore
Kindberg

- Andrew File System, Carnegie Mellon University
- Hauptautor: Mahadev Satyanarayanan, ca. 1989
- Weiterentwicklung: DFS

- Ziele:
  - Skalierbarkeit
    5.000 – 10.000 Arbeitsstationen
  - einfache Administration

Annahmen / Beobachtungen (Unix)

Coulouris
Dollimore
Kindberg

Beobachtungen

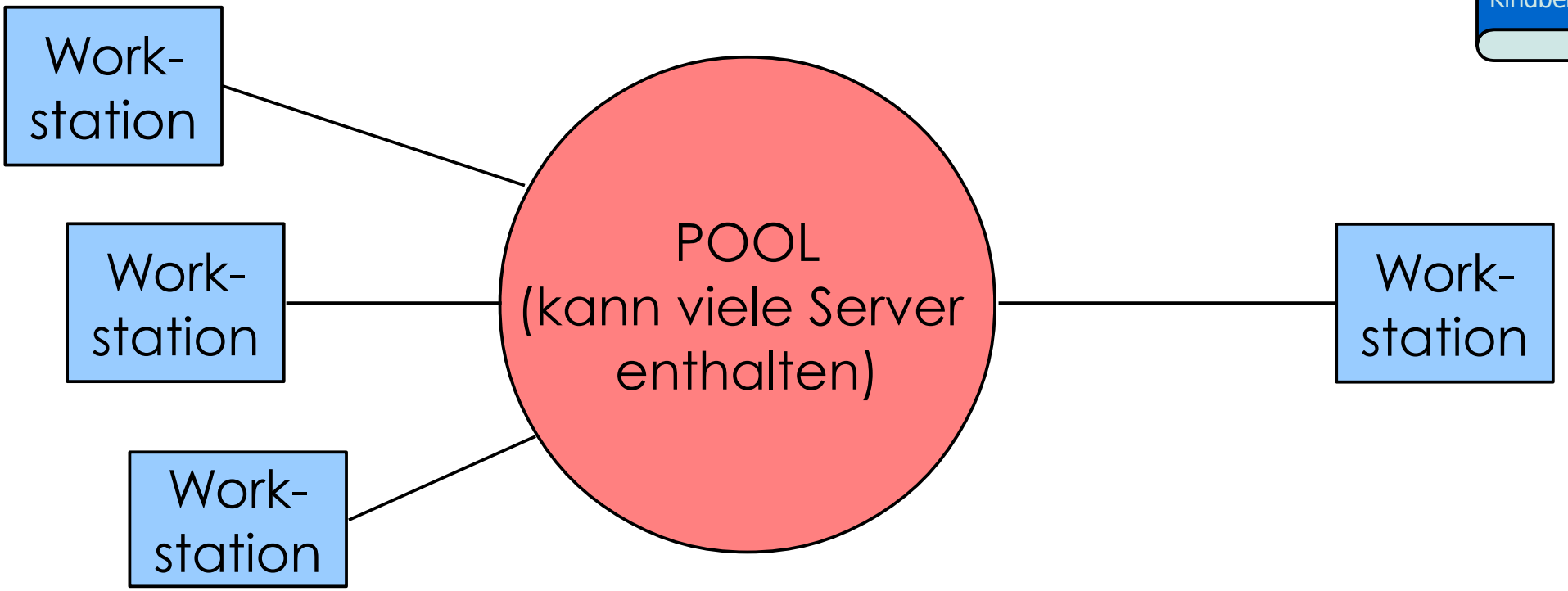
- Dateien sind in der Regel klein (zur Zeit des Entwurfs)
- Leseoperationen sind häufiger als Schreiboperationen.
- Sequentieller Zugriff ist häufiger als direkter.
- Die meisten Dateien haben nur einen Benutzer.
- Bei mehreren Nutzern schreibt meist nur einer.
- Dateizugriffe erfolgen burstartig.

➔ heute signifikanter Anteil sehr großer Dateien

(neue Messungen Ende 1999,
Werner Vogels für Windows NT)

Architektur von AFS

Coulouris
Dollimore
Kindberg

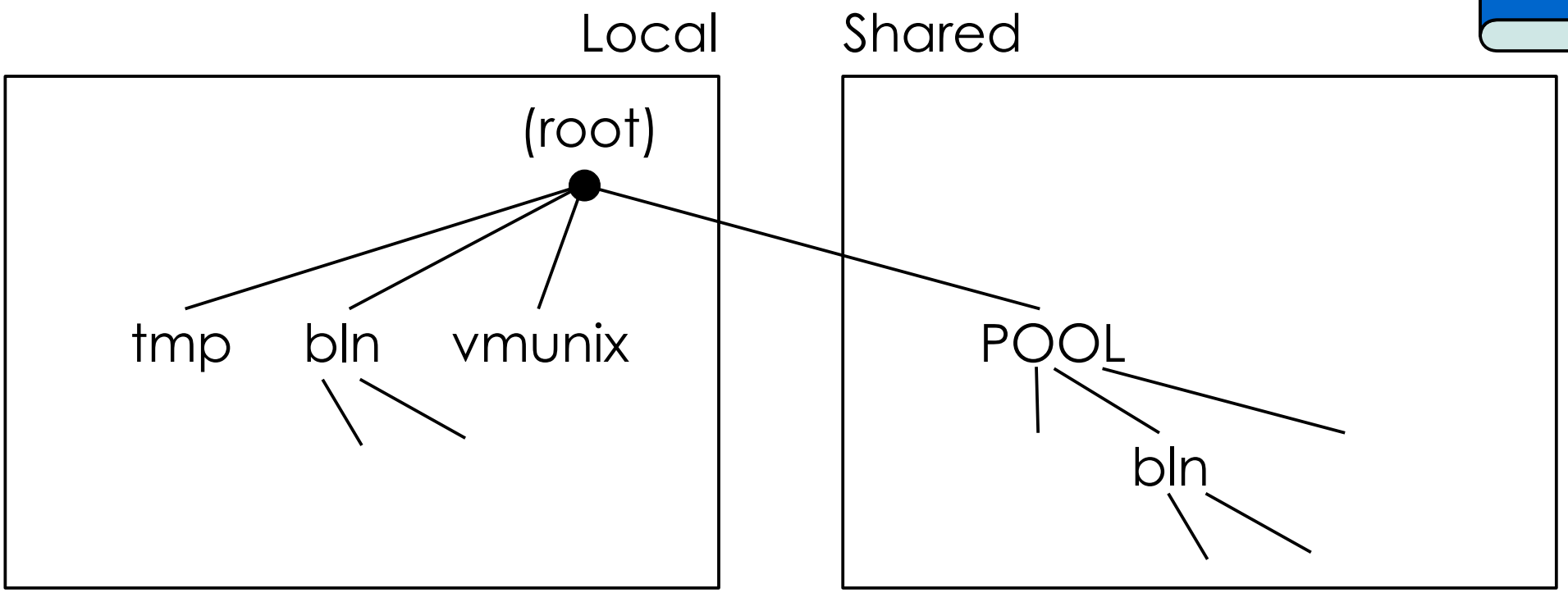


Prinzipien

- ganze Dateien werden in Arbeitsstationen-Caches gehalten
- Arbeitsstationen-Caches sind permanent
- sehr große Dateien werden nicht berücksichtigt

Namen

Coulouris
Dollimore
Kindberg



- alle Dateien auf den Servern sind über einen Teilbaum benannt
- Anpassung an Klienten-Bedarf durch "symbolic links"

  z. B. ln -s /usr/spool/mail /POOL/usr/haertig/spool/mail

Namen

Coulouris
Dollimore
Kindberg

- Verzeichnisse der obersten Ebene enthalten Dateinamen aller Server

- Replikation:
  - auf allen Servern
  - Lesen von beliebigen Replikaten
  - Schreiben über Master

Namen

Coulouris
Dollimore
Kindberg

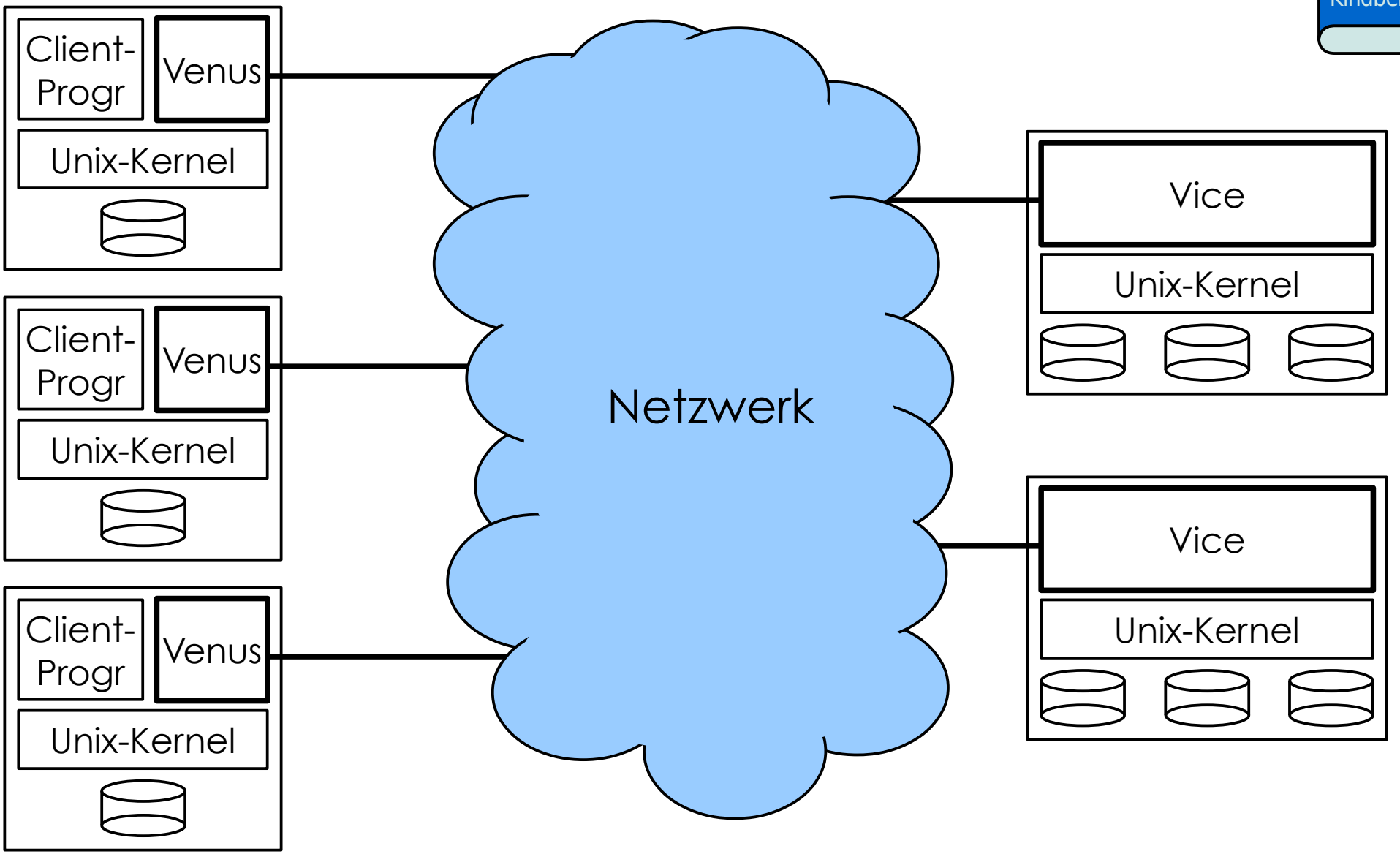
- Dateien sind in "Volumes" gruppiert
- Aufbau UFId:

Volume-Id	Id innerhalb Volume	Uniquifier

- Abbildung: Pfadname → UFId auf Klienten
  Verzeichnisse werden in Klienten-Cache gehalten
- jeder Server hat eine “Location-Database” zur Ermittlung einer Serveradresse
- Volumes können transferiert werden ohne Beteiligung von Arbeitsstationen

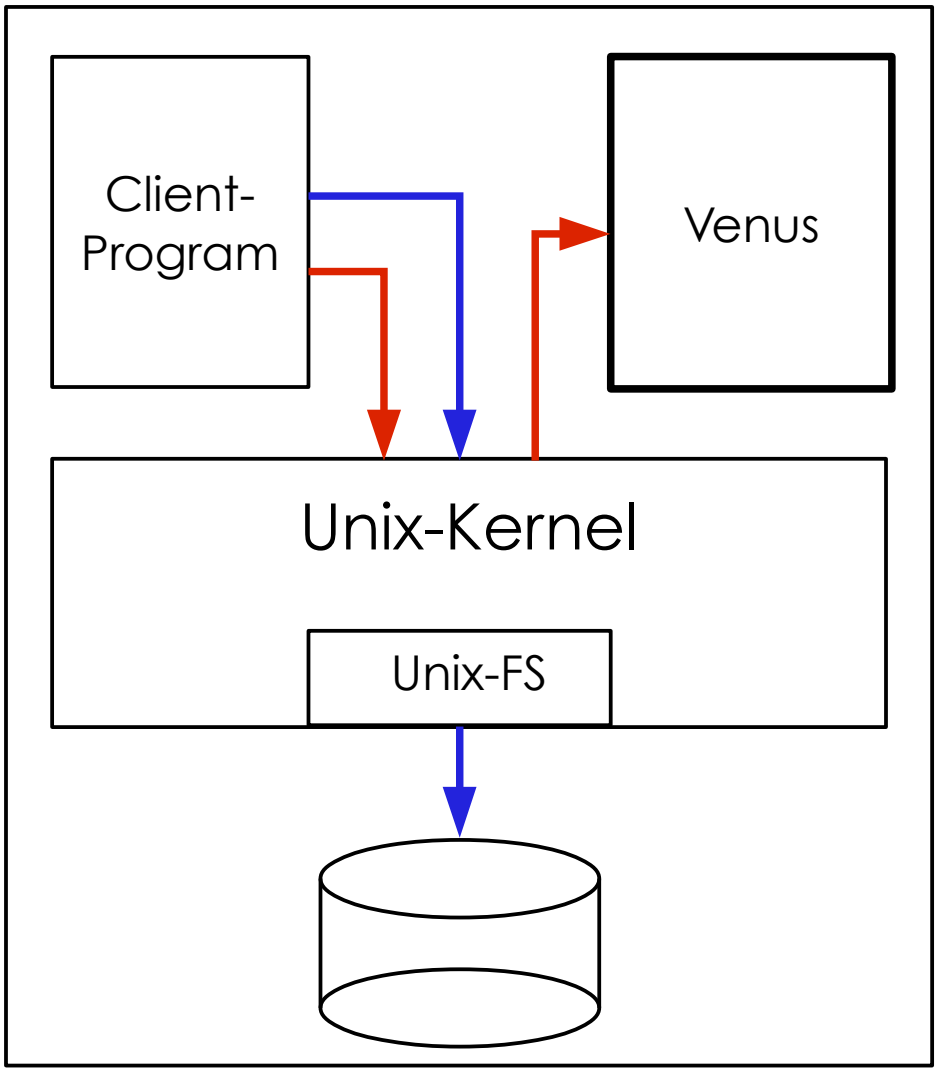
Struktur (1)

Coulouris
Dollimore
Kindberg



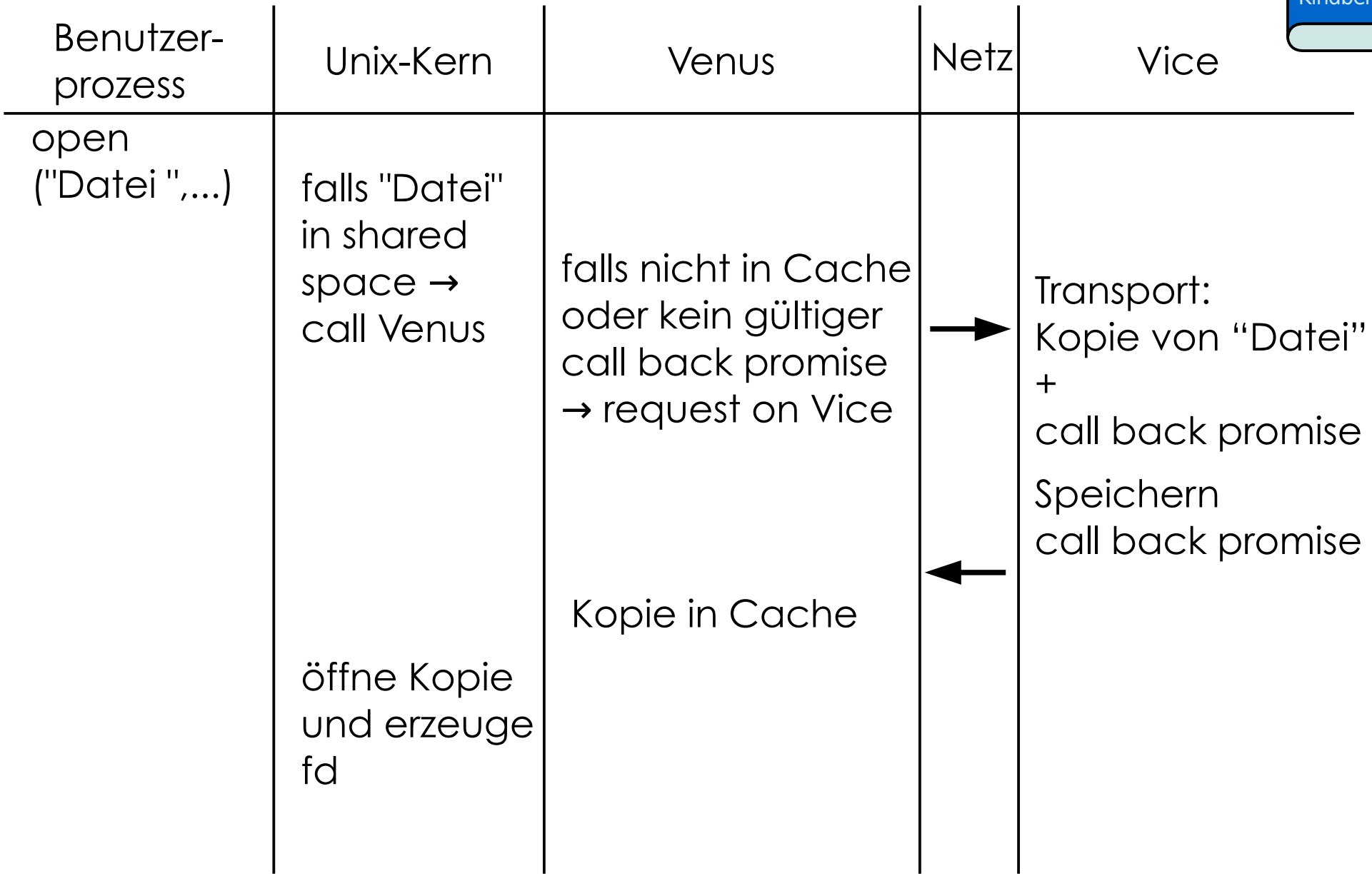
Struktur (2)

Coulouris
Dollimore
Kindberg



Protokoll (1)

Coulouris Dollimore Kindberg



Protokoll (2)

Coulouris Dollimore Kindberg

Benutzer-prozess	Unix-Kern	Venus	Netz	Vice
close (fd)	schließen lokale Kopie call Venus	falls lokale Kopie geändert → Kopie transportieren an Vice	→	ersetze "Datei" sende call back an alle Klienten, die call back promise haben
read/write	normaler Unix-Syscall auf Kopie			

Cache

Coulouris
Dollimore
Kindberg

- ganze Dateien
- Hauptspeicher und Platte des Klienten als Cache
- enthalten “working set” von Dateien
- Tausch nach LRU

Konsistenz

Coulouris Dollimore Kindberg

- bei Übertragung einer Datei von Vice nach Venus wird "call back promise" mitgegeben
  mögliche Werte: valid/cancelled
- bei Änderung an einer Datei auf Vice:
  - callback-RPC an alle Klienten, die ein call back promise haben
  - Wirkung: valid → cancelled
- nach Systemstart auf Klienten für jede Datei mit valid:
  cache validation request an Vice mit Modifikations-Zeitstempel
  - falls noch gültig: valid
  - falls ungültig: neue Kopie
- Callback wird erneuert bei open, falls seit der Zeit T keine Kommunikation mit Server stattfand
  - ➔ T = einige Minuten

Vice-Schnittstelle

Coulouris
Dollimore
Kindberg

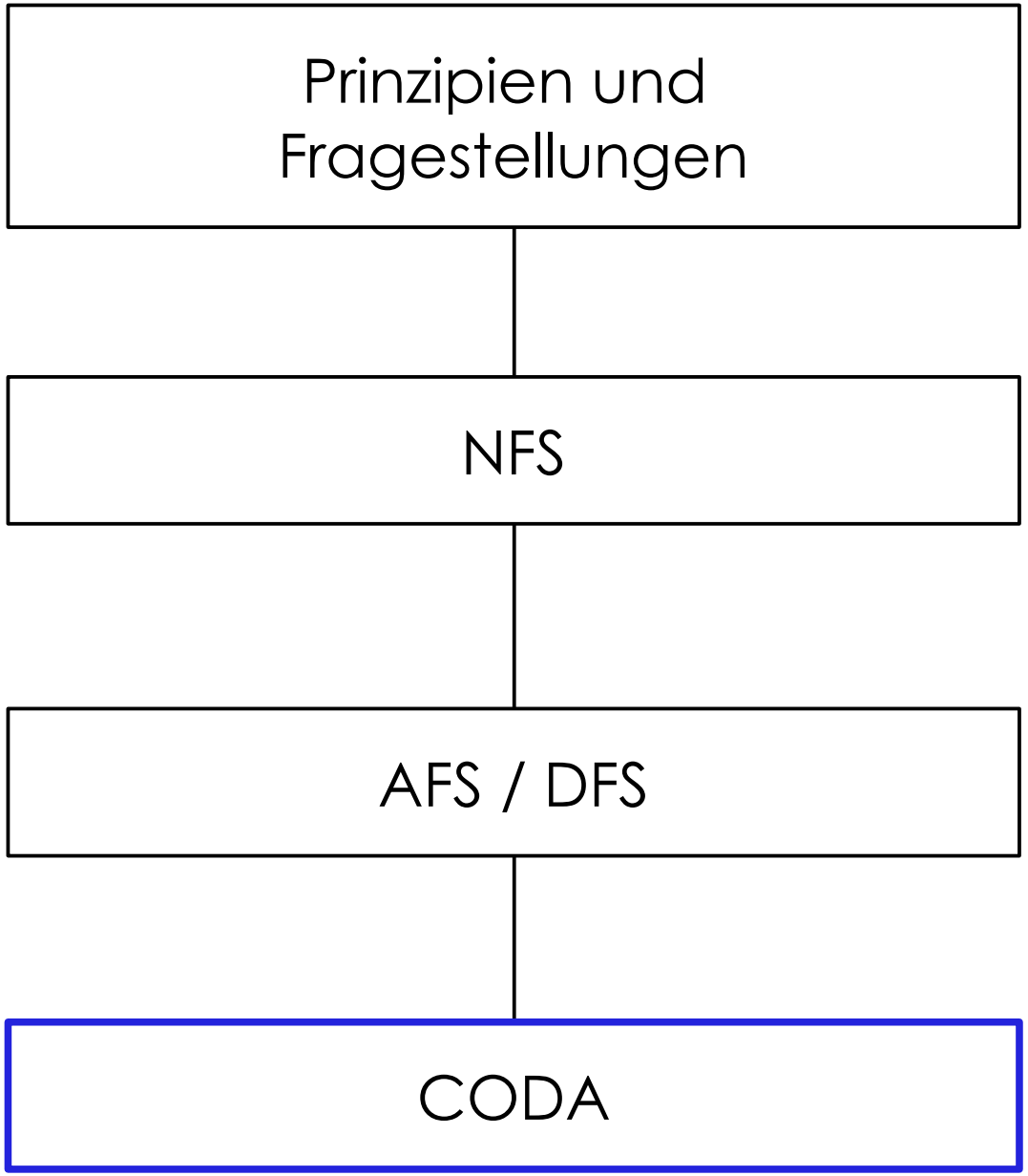
- Fetch (fid) -> attr, data
- Store (fid, attr, data)
- Create (), Remove (fid)
- Set Lock (fid, mode), Release Lock (fid)
- Remove Call back (fid)

  Klient sagt Server, dass Datei aus Cache entfernt
- Break Callback (fid)

  Vice → Venus: cancel callback promise

Wegweiser

Coulouris
Dollimore
Kindberg



Fallbeispiel 3: CODA

Coulouris
Dollimore
Kindberg

- Carnegie Mellon University
- Satyanarayanan, Kistler 1992
- Weiterentwicklung von AFS

Ziele

- Erhöhung der Verfügbarkeit auch bei Netz-Partitionierung
- Mobile Computing als Spezialfall von Netz-Partition

Techniken

- Replikation
- abgekoppelter Betrieb (Disconnected Operation)

Replikation (1)

Coulouris
Dollimore
Kindberg

- Volumes werden repliziert

  VSG: Volume Storage Group
  Menge der Server, die Replikate enthalten

  AVSG : available VSG
  AVSG⊆ VSG
  AVSG = ∅: disconnected operation

- open-Operationen:
  - ein Server des AVSG liefert Datei-Inhalt
  - alle Server werden wg. Status-Info angefragt

- close-Operationen:
  - multicast geänderter Dateien an alle Server aus AVSG

Replikation (2)

Coulouris
Dollimore
Kindberg

- optimistisches Replikationsverfahren:
  - Modifikation werden in jedem Fall bei den Servern der AVSG ausgeführt
  - Beseitigung von Inkonsistenzen bei Feststellung durch Klienten nach Aufhebung der Partition
  - wenn möglich, automatisch, sonst manuell

Replikationen (3)

Coulouris
Dollimore
Kindberg

Technik: CVV (Coda Version Vector)

- für jedes Replikat einer Datei
- $CVV_i := (S_{i1}, \ldots, S_{in})$

  S_{ik} : Anzahl der Modifikationen von Replikat i auf Server k
- CVV_i "dominiert " CVV_j <=>

  $\forall$ k = 1... n gilt $S_{ik} \geq S_{jk}$ (ist aktueller als)
- bei jeder Modifikation (close-Operation) durch Server k:

  $S_{ik} := S_{ik} + 1$ für alle $i \in$ AVSG
- dann kann bei Lesen (lazy) eine Inkonsistenz automatisch repariert werden, wenn:

  $\exists S_i \in$ AVSG so, dass gilt: CVV_i dominiert CVV_j, $\forall j \neq i$

  (S_i ist aktueller als alle anderen)

Beispiel

- Datei F
- $VSG_F = \{ S_1, S_2, S_3 \}$
- Klienten: C_1, C_2

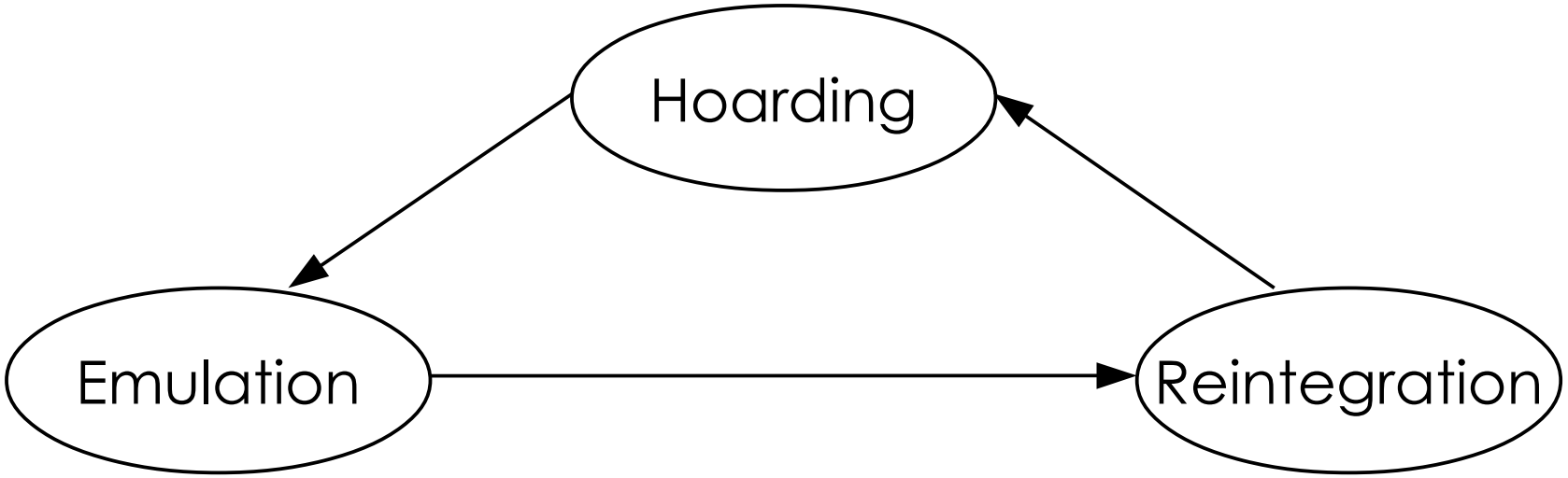
Beispiel

Ereignis	AVSG	CVV_1	CVV_2	CVV_3
start	{S1, S2, S3}	1, 1, 1	1, 1, 1	1, 1, 1
C1 : öffnet, modifiziert schließt		2, 2, 2	2, 2, 2	2, 2, 2
S3 : fällt aus	{S1, S2}			
C1 : modifiziert		3, 3, 2	3, 3, 2	2, 2, 2
S3 : Neustart	{S1, S2, S3}			
C1 : liest		3, 3, 2	3, 3, 2	3, 3, 2
S3 : fällt aus	{S1, S2}			
C1 : modifiziert		4, 4, 2	4, 4, 2	3, 3, 2
S3 : Neustart	{S1, S2, S3}			
S1, S2 : fallen aus	{ S3 }			
C2 : modifiziert		4, 4, 2	4, 4, 2	3, 3, 3
S1 : Neustart	{S1, S3}			
C1 : liest				

Abgekoppelter Betrieb (1)

Coulouris
Dollimore
Kindberg

- AVSG = 0, z. B. Mobile Computing
- Problem: LRU-Verfahren für Datei-Caching funktioniert nicht gut bei längeren Abkoppelungen
- Benutzer können Dateien angeben (hoard database)
- Werkzeug: ermittelt Dateien für eine Anwendung



Abgekoppelter Betrieb (2)

Coulouris
Dollimore
Kindberg

Phasen des Betriebes

- generiert temporäre fid
- Zugriffsschutz
- mod. Dateien aufheben bis zur Reintegrationsphase
- hat “replay-log”

➔ Reintegration: führt Aktion aus replay log aus

Bewertung (Erfahrungen)

Coulouris
Dollimore
Kindberg

- Wenig genutzt
- sehr seltenes Auftreten nicht automatisch behebbarer Konflikte

Zusammenfassung

Coulouris
Dollimore
Kindberg

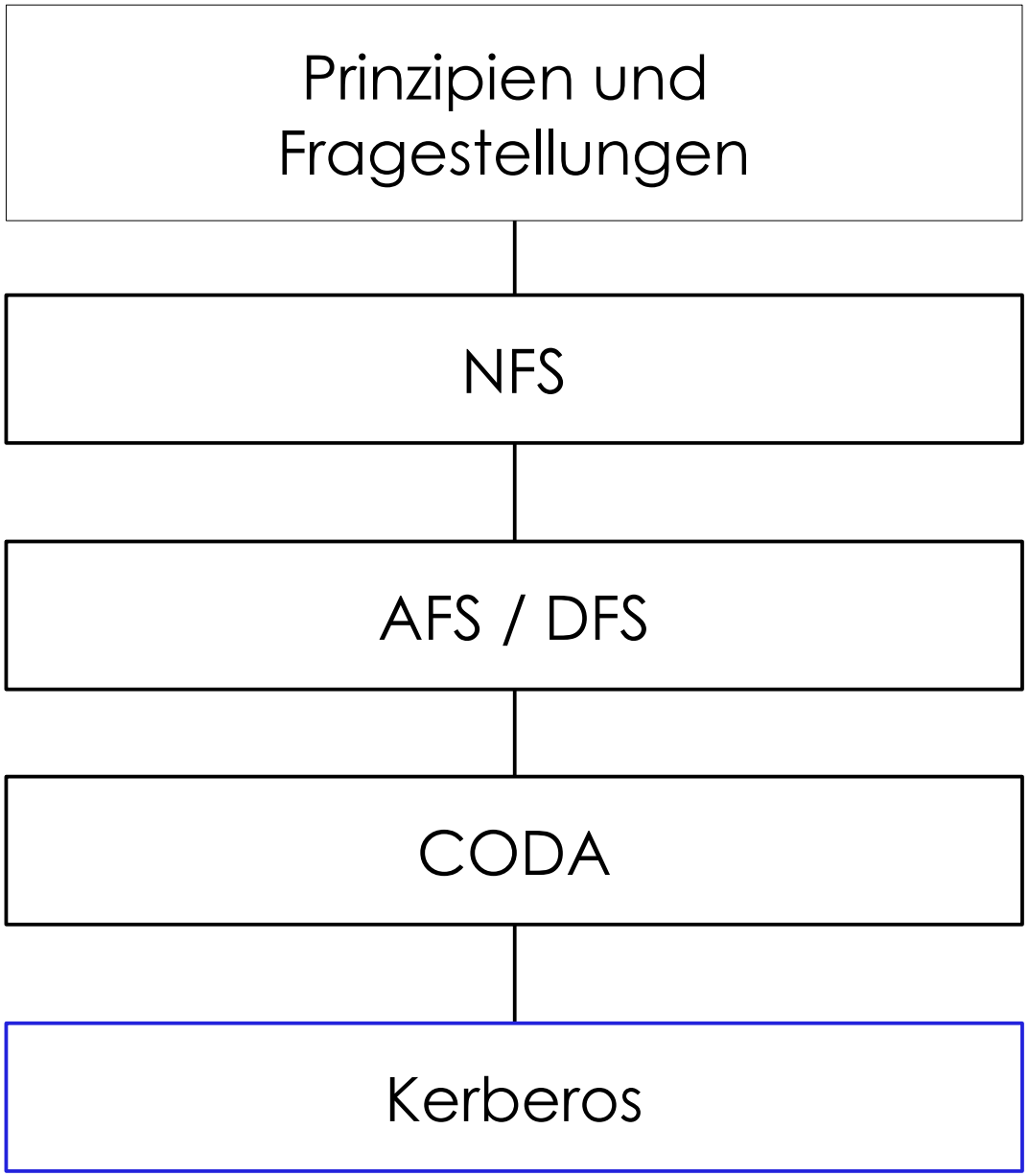
Beispiel für Anwendung des RPC

Entwicklung: NFS - AFS - Coda …

Mehr (Distributed Operating Systems):

- Skalierbare Dateisysteme (Google FS)

Wegweiser



Bisher

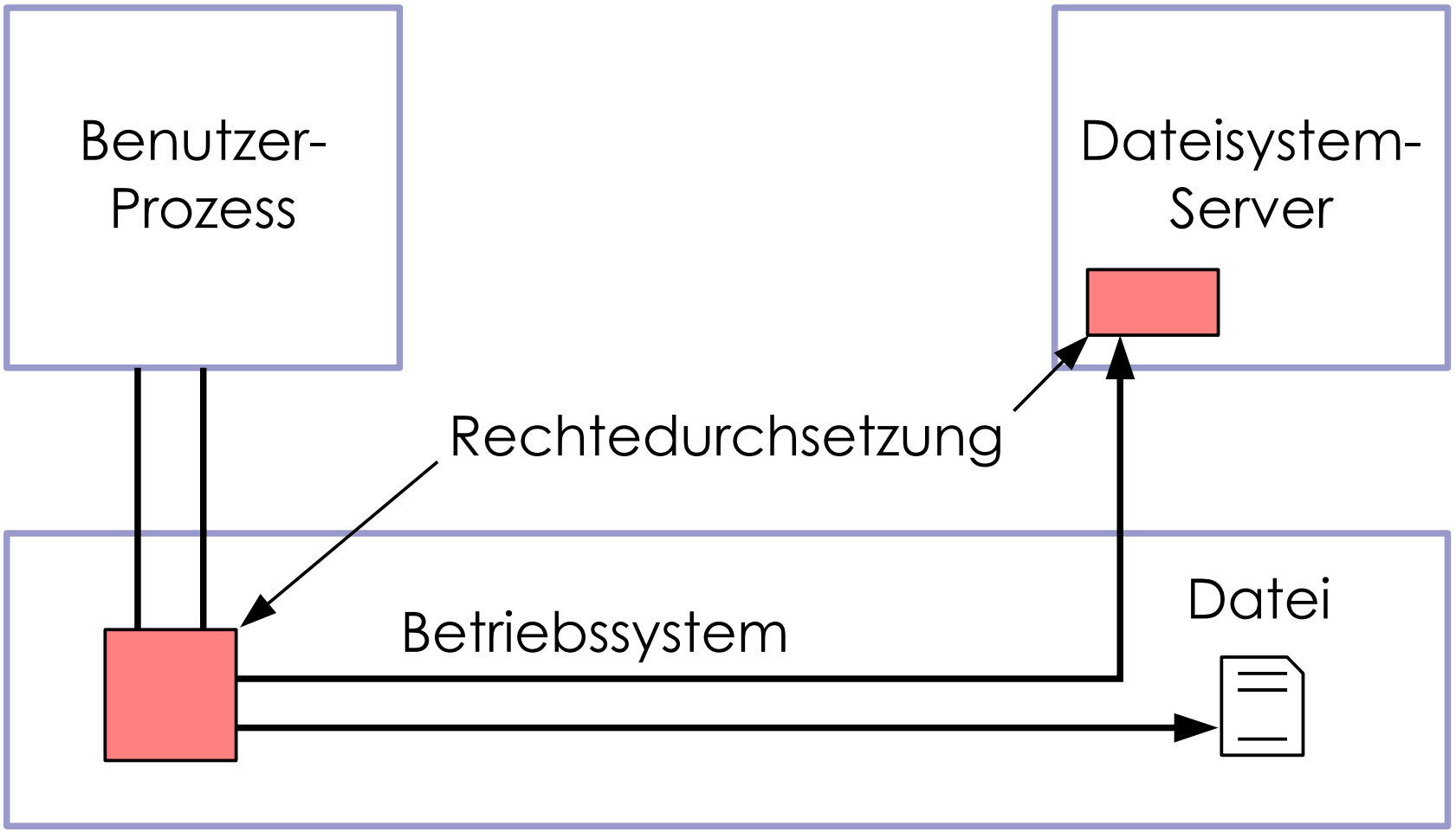
Schutzziele

- Vertraulichkeit (Confidentiality)
- Integrität (Integrity)
- Verfügbarkeit (Availability)
- Wiederherstellbarkeit (Recoverability)

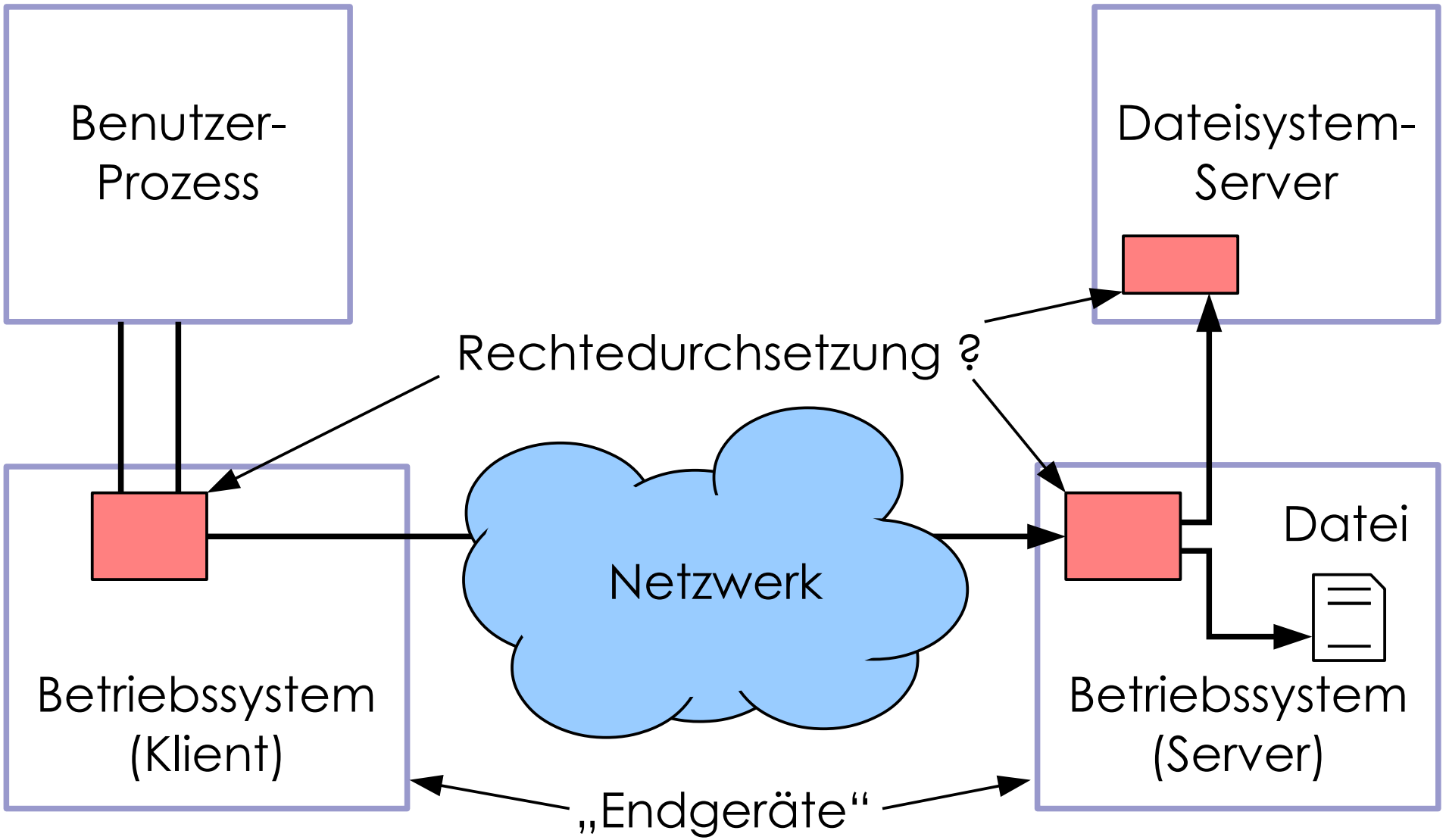
Grundlagen und Mechanismen (lokal)

- Prozesse als Repräsentanten von Benutzern
- Isolation (Adressräume)
- Authentifikation von Benutzern
- Schutzmechanismen (ACL, Capabilities, Sandboxing)

Lokal


Benutzer-
Prozess
Dateisystem-
Server
Rechtedurchsetzung
Betriebssystem
Datei

Verteilt


Benutzer-
Prozess
Dateisystem-
Server
Rechtedurchsetzung ?
Netzwerk
Datei
Betriebssystem
(Klient)
Betriebssystem
(Server)
„Endgeräte“

Bedrohungen

Ein Angreifer kann:

- in Kommunikationspfade eindringen
- Nachricht
  abhören – ändern – hinzufügen – löschen
- abgehörte Nachricht später wiederholen
- mehrere Nachrichten vertauschen
- Quell- oder Zieladresse einer Nachricht verändern
- seinen Rechner so modifizieren, dass er sich für einen anderen Rechner mit einer anderen Adresse ausgibt

Forderungen

- Konzelation als Grundlage von Vertraulichkeit

- Authentifikation als Grundlage der Integrität
  - Absender
  - Inhalt

→ Grundlage für Botschaften: kryptographische Systeme
  - Konzelationssysteme
  - Authentifikationssysteme

Einfache Anwendungen von Kryptoverfahren

Feststellung einer Identität A

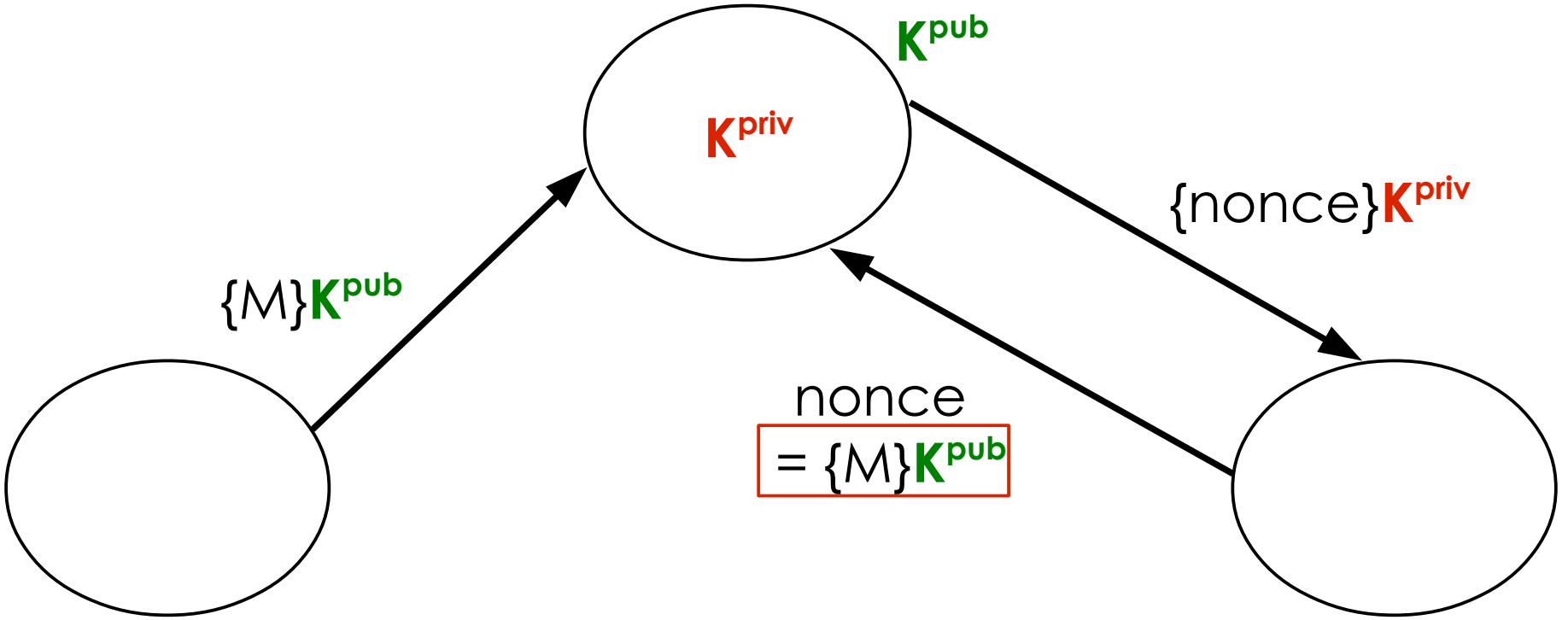
Sender:	„nonce“
Antwort von A:	{nonce, t} K^{priv}
Sender:	{{nonce, t} K^{priv} }K^{pub} enthält „nonce“

- Voraussetzung:
  Sender muss wissen, dass K^{pub} zu A gehört

Vertraulichkeit

Sender:	{Botschaft} K^{pub}
Empfänger:	{{Botschaft} K^{pub} }K^{priv}

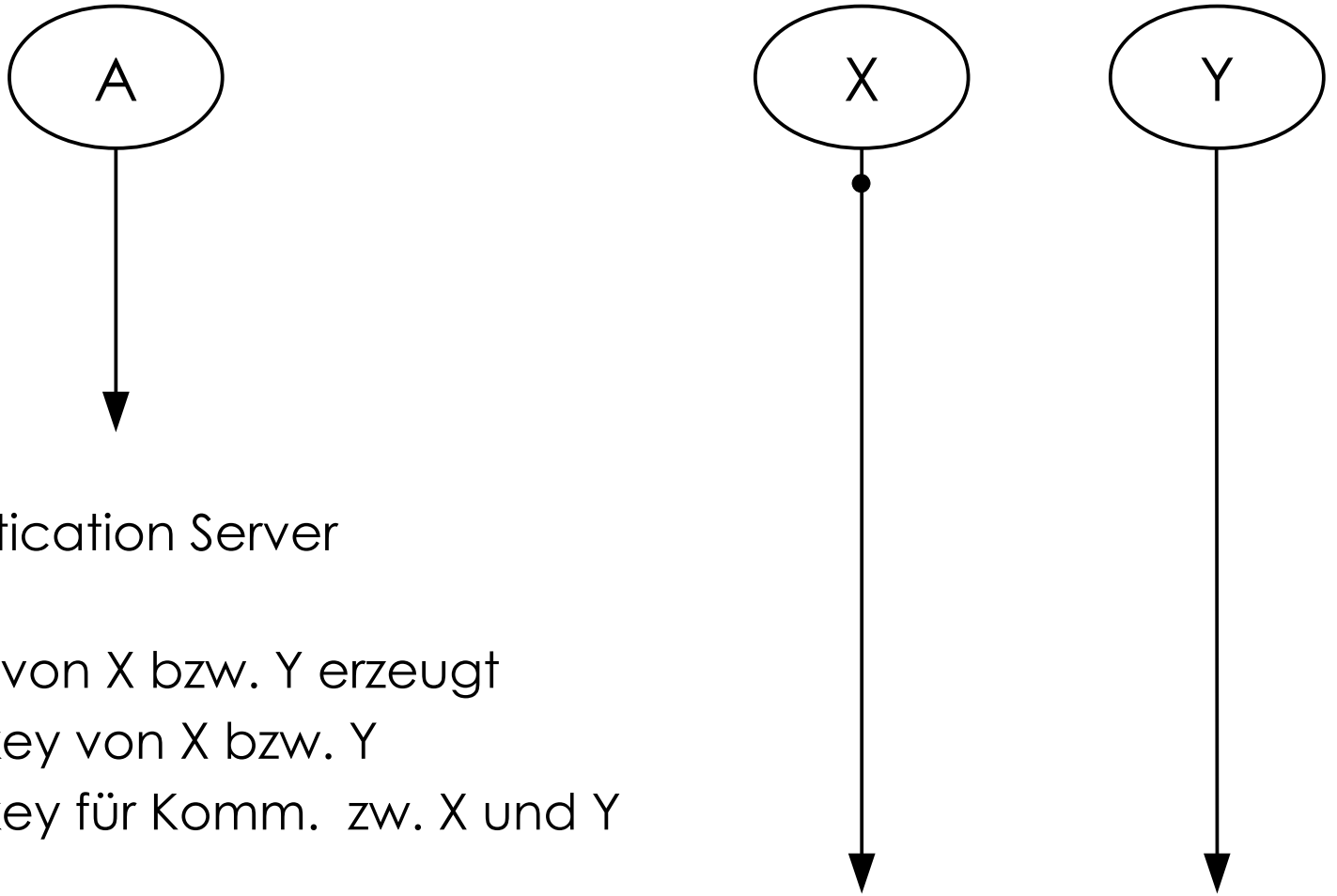
Beispiel



Falsch!

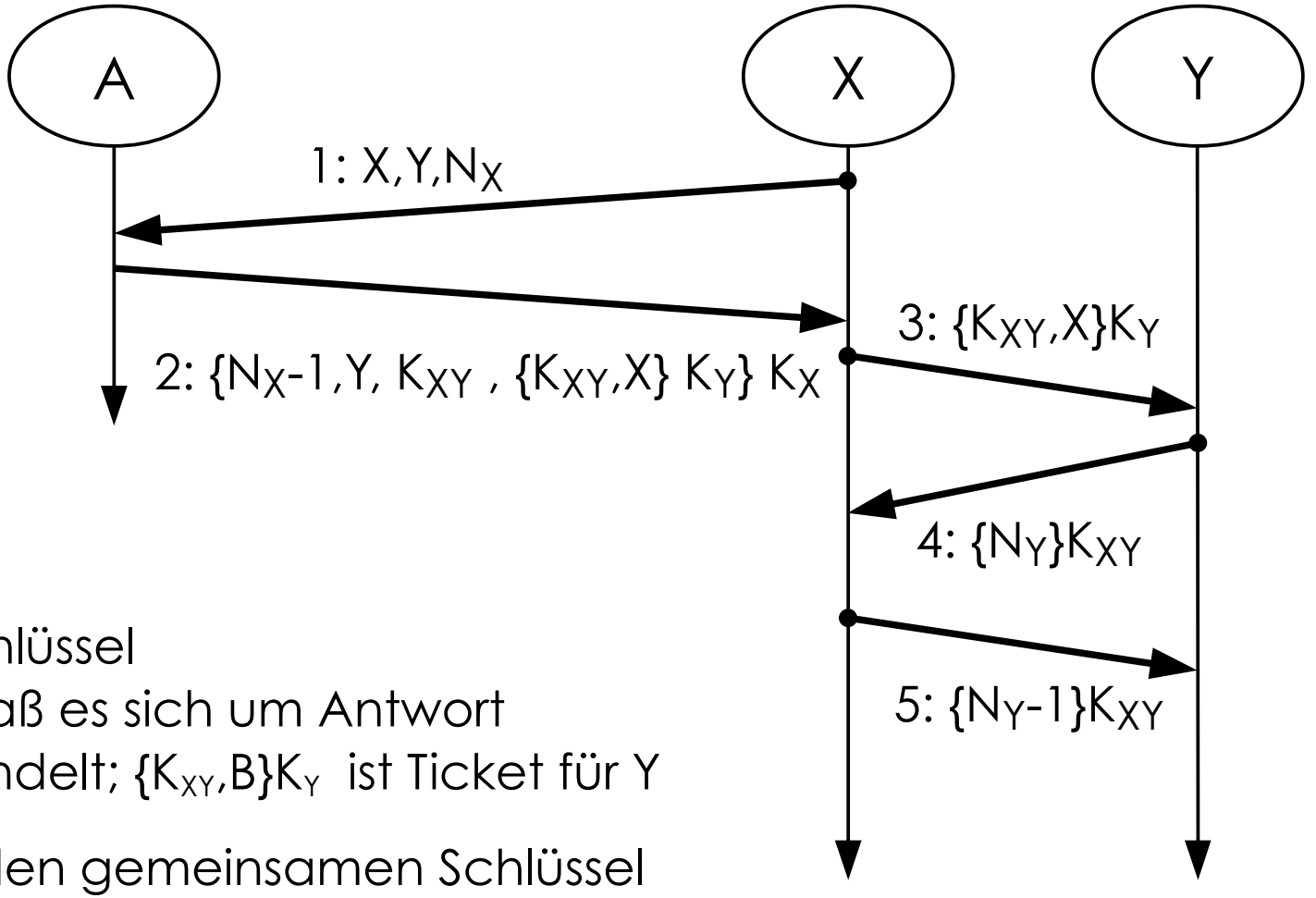
- Auf den folgenden Folien ist ungeschickt, dass A einmal für Client steht und später dann für Authenticationsservice → ändern

Protokoll (symmetric key NS)



A: Authentication Server
X,Y: Clients
N_X, N_Y : nonce, von X bzw. Y erzeugt
K_x, K_y : secret key von X bzw. Y
K_{XY} : secret key für Komm. zw. X und Y

Protokoll (symmetric key NS)



1: key request

2: K_{XY} neuer Schlüssel
N_X-1 zeigt, daß es sich um Antwort auf 1 handelt; {K_{XY},B}K_Y ist Ticket für Y

3: X gibt an Y den gemeinsamen Schlüssel

4/5: durch {N_Y-1} zeigt Y, daß Y tatsächlich der Absender von 3 war

NS-Verfahren: Public-key based

➔ public key basiertes Verfahren

- Principal X, Y

  Benutzer, einige Dienste

  "Einheit", der Rechte zugestanden werden

- Authentication Server A

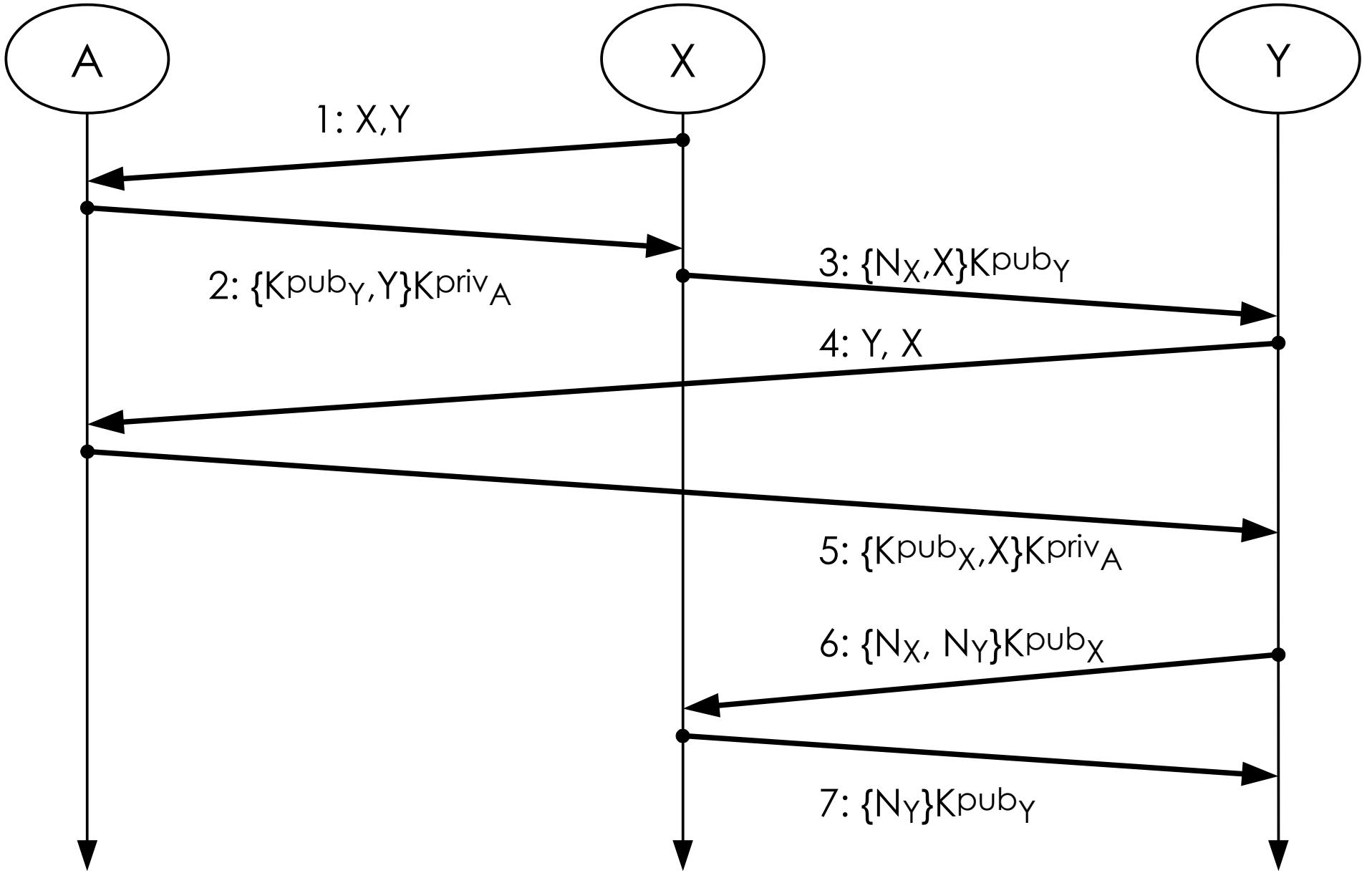
  kennt <u>öffentliche</u> Schlüssel der Principale

K^{priv}_{E} private key von E

K^{pub}_{E} analog ...

N_E nonce, von E erzeugt

Protokoll (public key NS) (1)



Protokoll (public key NS) (2)

1: X will public key von Y

2: jeder kann K^{pub}_{Y} erfahren mittels K^{pub}_{A}

3: nur Y kann "X" herausfinden

4,5: Y besorgt X's public key von A

6,7: durch Bestätigung der "nonces" wird Aktualität gezeigt

Kerberos

- Netzwerk-Authentifizierungsservice
- ➔ Integration des Needham/Schroeder-Verfahrens in CS-Architektur

Wesentliche Elemente

- Authenticator
  - beweist Identität des Absenders; enthält Zeitstempel jedoch kein Passwort
  - einmalig gültig

- Ticket
  - für ein Client-Server-Paar; beweist Authentifikation
  - beschränkte Gültigkeit
  - enthält Sitzungsschlüssel

- Sitzungsschlüssel
  - secret key für Kommunikation mit Servern

- ➔ Mehr Information: http://web.mit.edu/kerberos

Kerberos

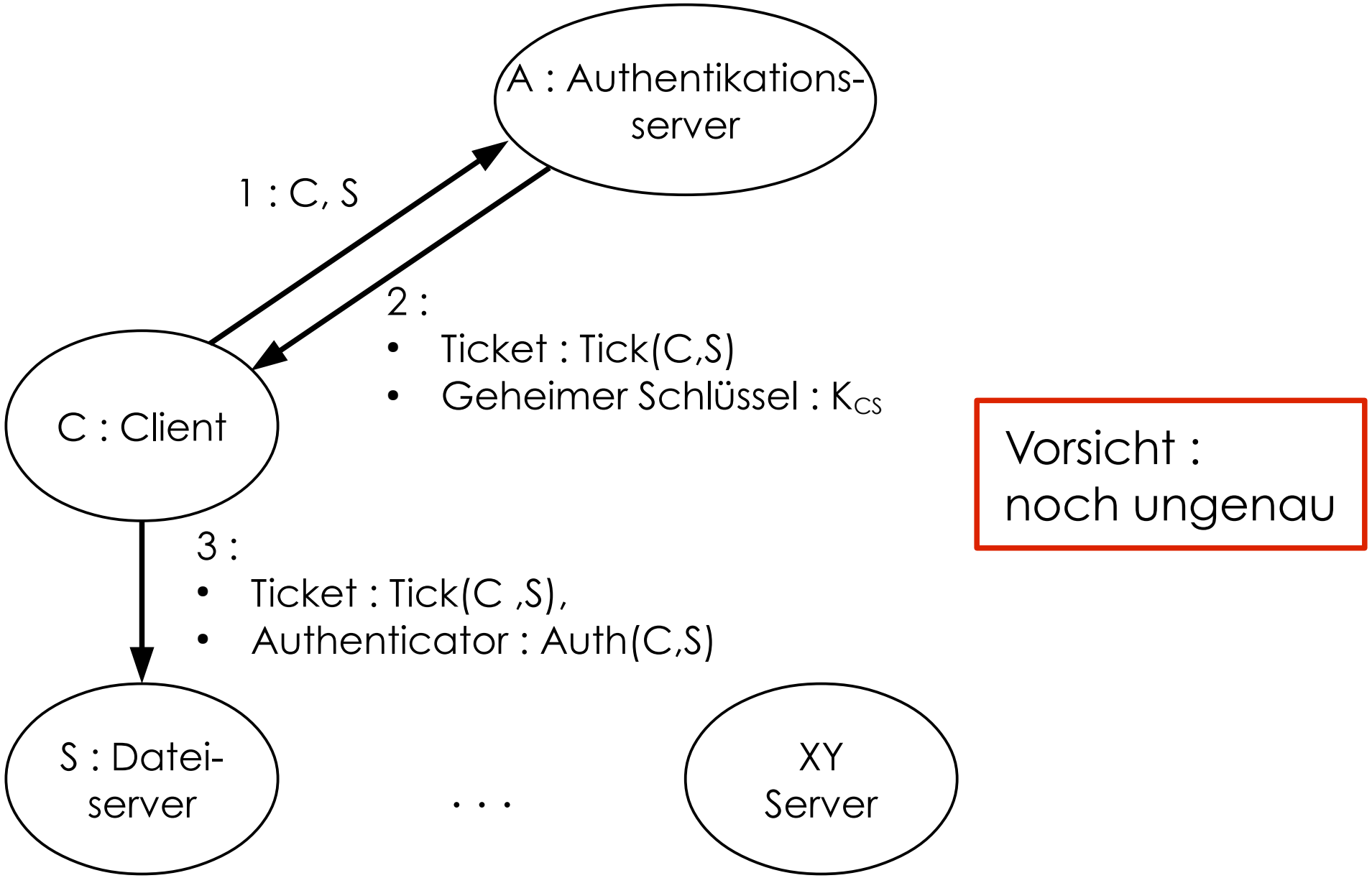
Annahmen

- Offenes (unsicheres) Netz
- Abhör- und Manipulierbarkeit
- Kein Vertrauen in Rechnernamen (Hostnames)
- Arbeitsstationen
- wechselnde Nutzer
- Passwort : sicher eingebbar, langfristig nicht sicher speicherbar
- Uhren synchronisiert

Konsequenzen für Passwörter

- Nicht im Klartext auf Netzwerk
- Auf Arbeitsstationen nur sehr kurz

Struktur



Vorsicht :
noch ungenau

Login ohne Passwort auf Netzwerk

Voraussetzung

- A kennt Passwörter aller Prinzipale
- Funktion f : Passwort → geheimer Schlüssel

Protokoll

C → A : C, S, time, nonce
A holt K_C und
erzeugt Session-Key K_{CS}

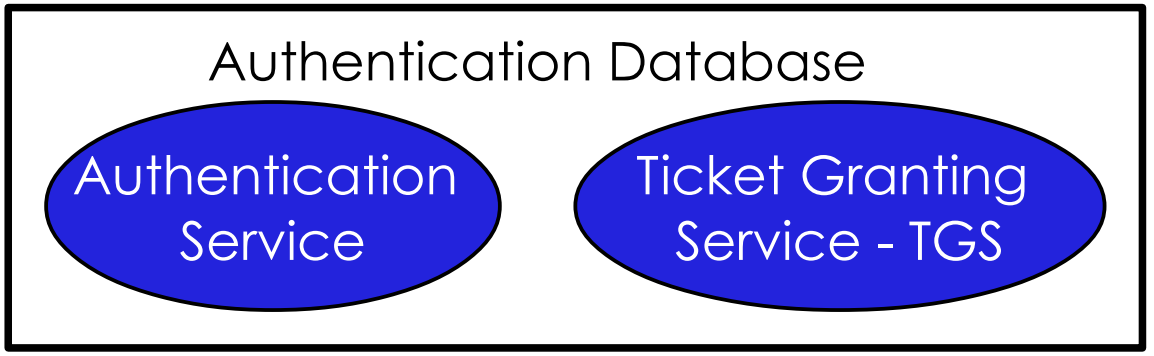
A → C : $\{ K_{CS}$, nonce, ... $\}K_C$ + Ticket
C fragt Benutzer nach Passwort,
berechnet K_C und packt aus
und löscht Passwort + K_C sofort wieder

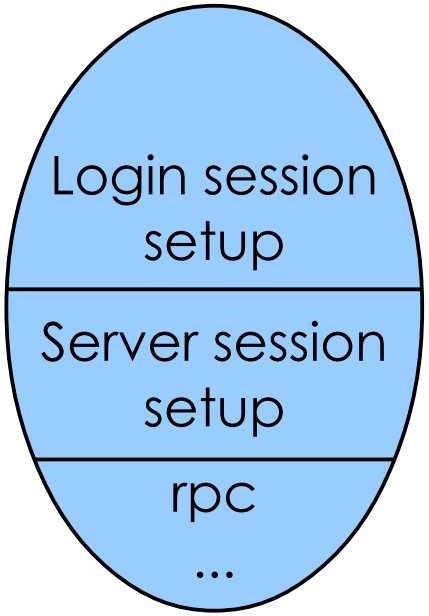
Eigenschaften / Konsequenzen

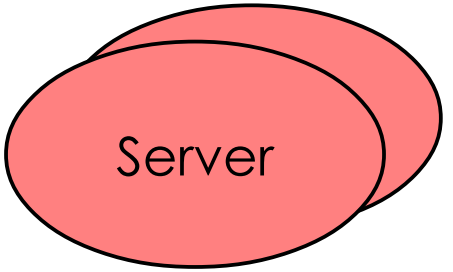
- Passwort bzw Schlüssel :
  muss für jeden Server neu angegeben werden
- ➔ Zur Vermeidung :
  Einführung eines Ticket Granting Service

- Authentikationsserver (Admin)
  benötigt Passwörter (oder Schlüssel) im Klartext !!!

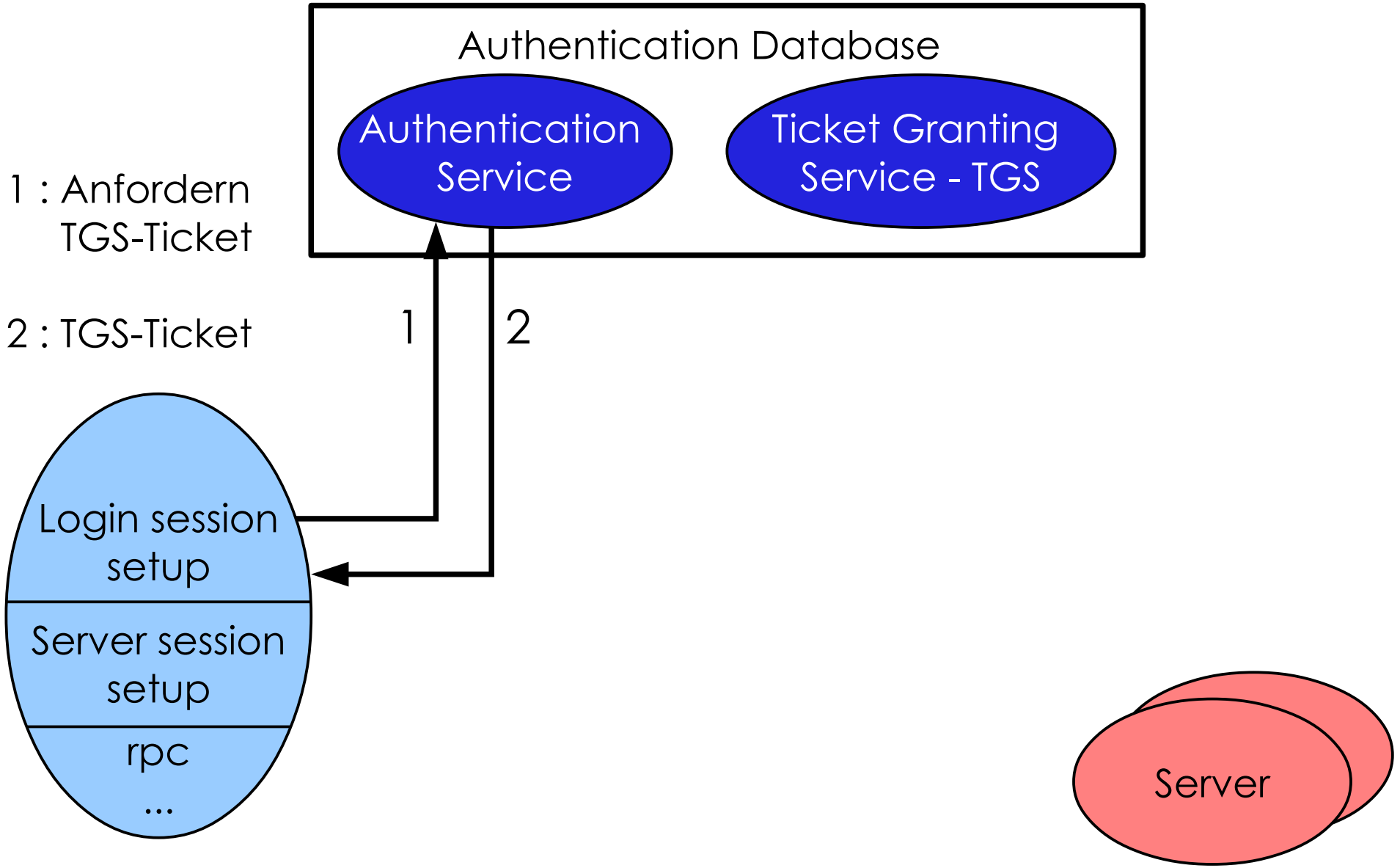
Kerberos Key Distribution Center



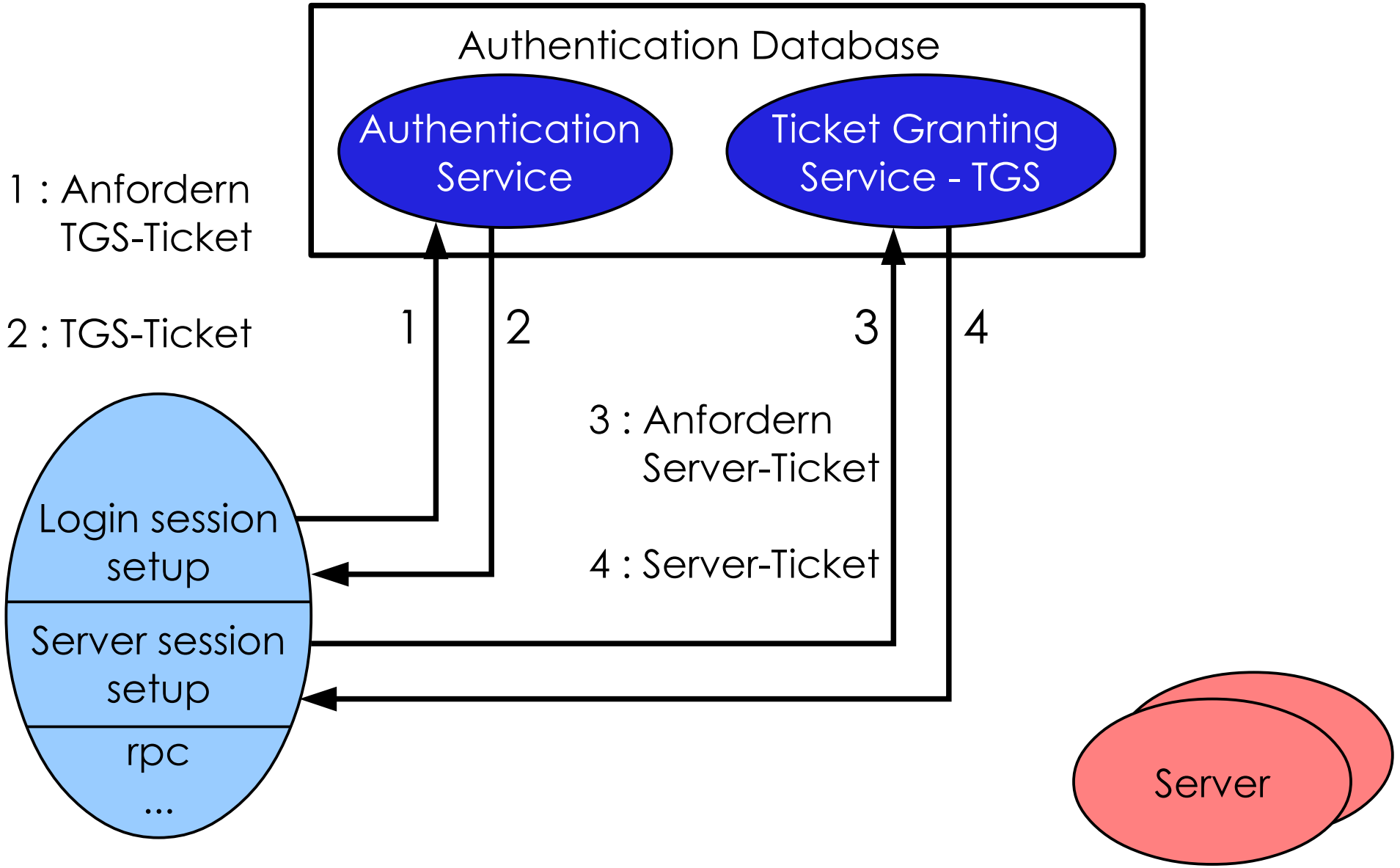




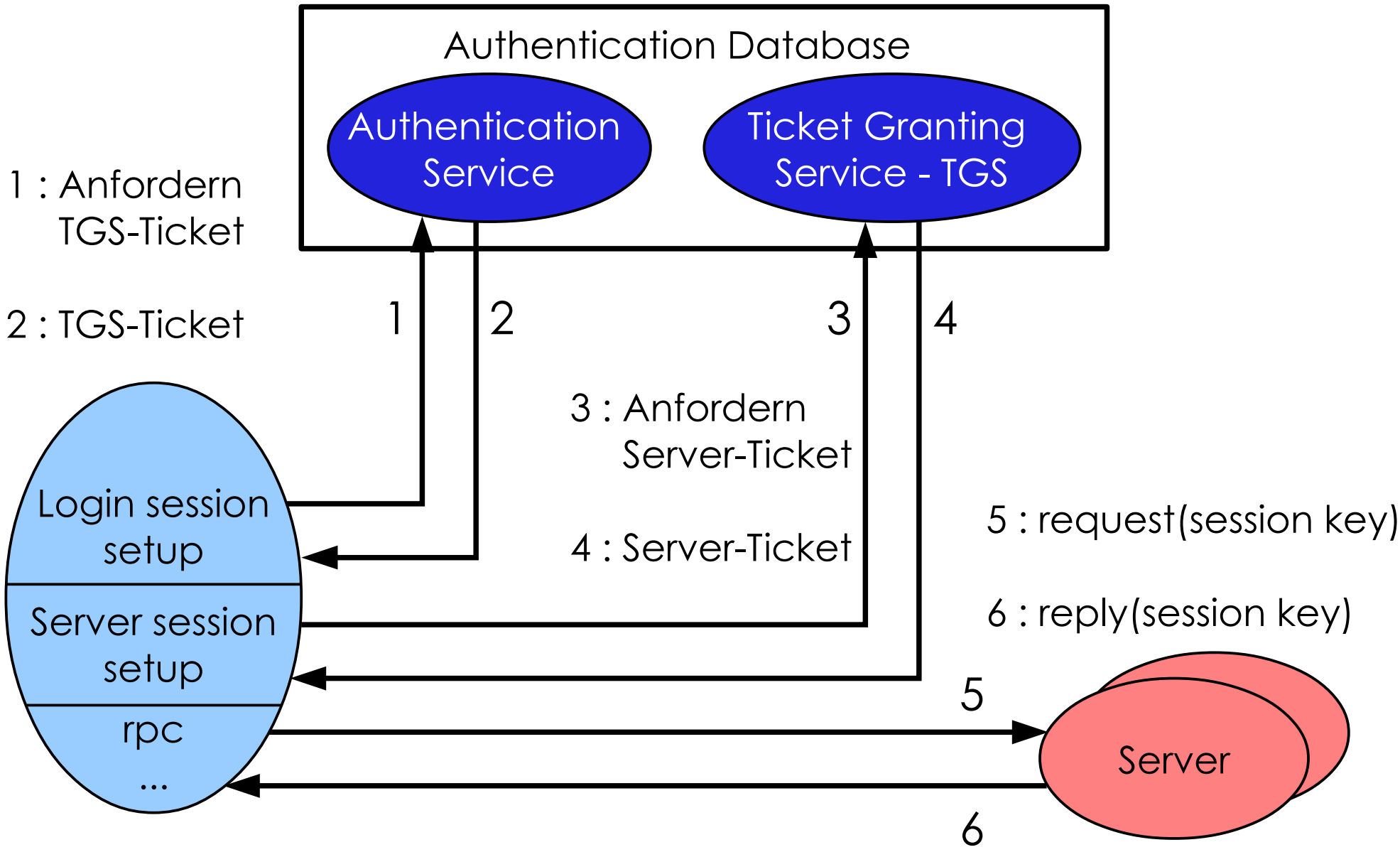
Kerberos Key Distribution Center



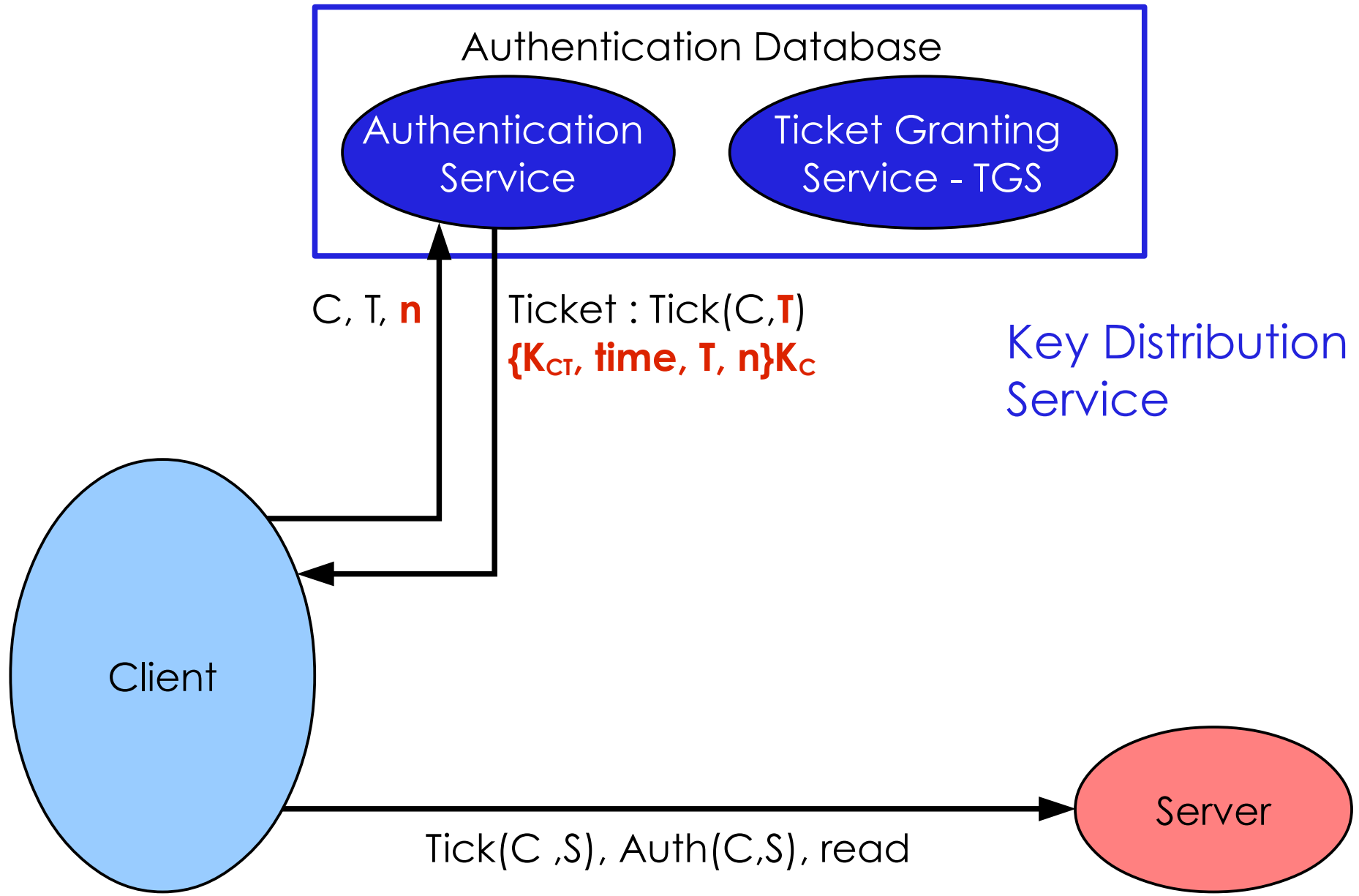
Kerberos Key Distribution Center



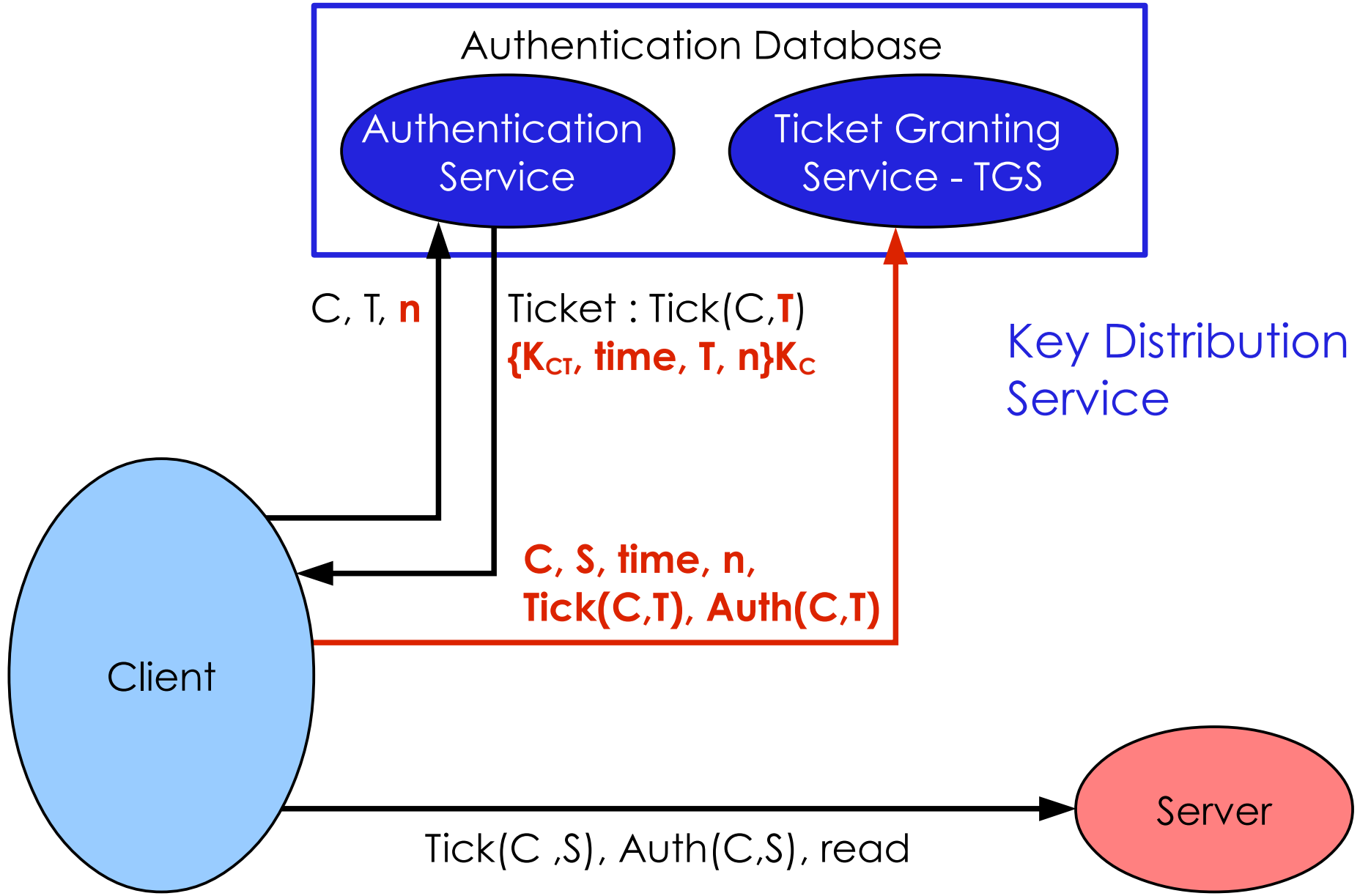
Kerberos Key Distribution Center



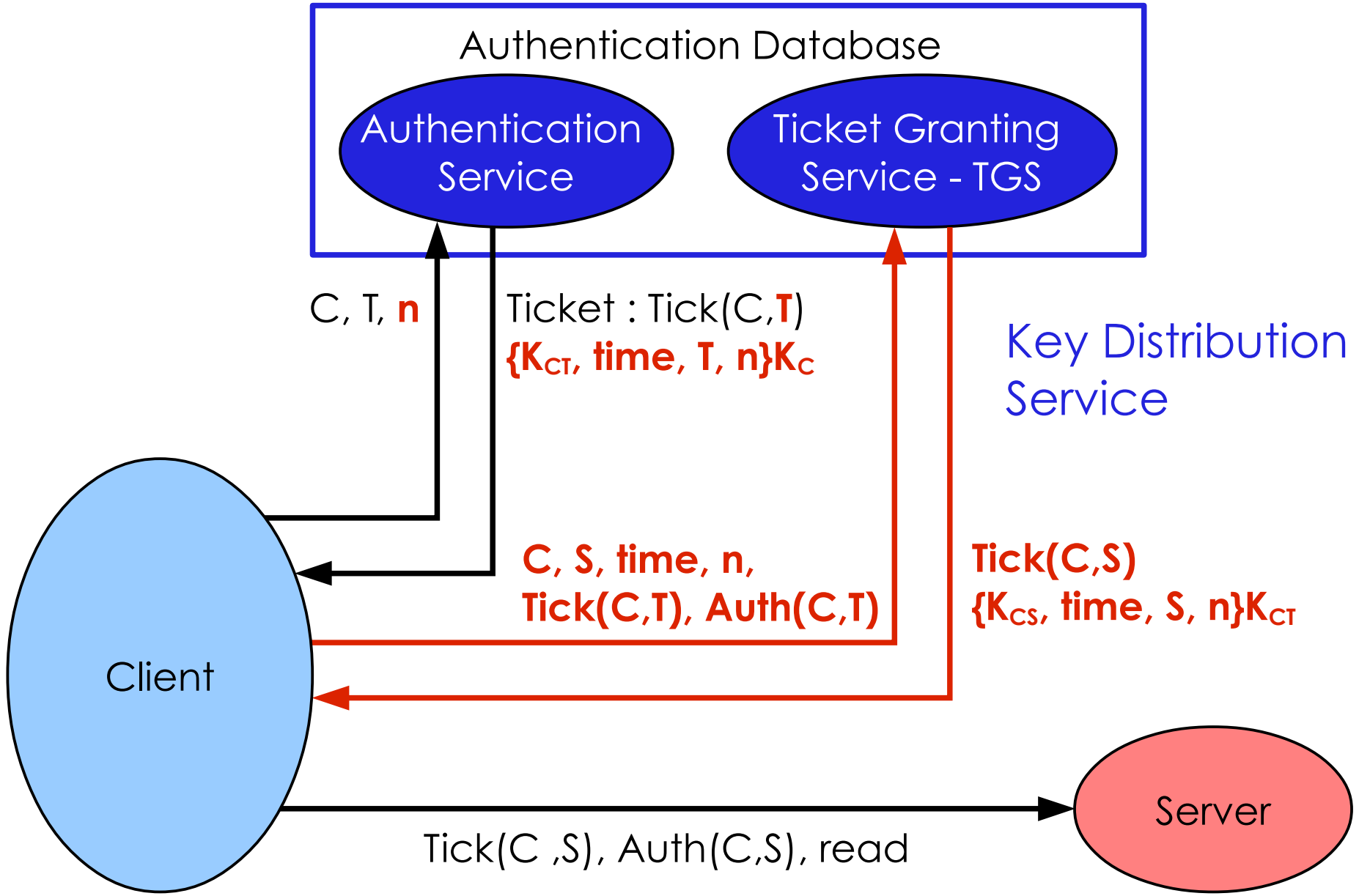
Kerberos : Key Distribution Service



Kerberos : Key Distribution Service



Kerberos : Key Distribution Service



A und T

Authentication-Server

- kennt geheime Schlüssel
  - der Prinzipale, der aus deren Passwort erzeugt wird
  - und eines speziellen Servers : "Ticket-Granting-Server"

Ticket-Granting-Server

- kennt geheime Schlüssel der Server

Tickets

Tick(C,S) : $\{C,S,t1,t2,K_{CS}\}K_S$

- Lebensdauer : t1 bis t2
- gelten für ein Client-Server-Paar : CS
- nur S kann Ticket interpretieren/manipulieren
  da verschlüsselt durch K_S ...
- enthalten session key : K_{CS}

➔ Aufgabe: zeigen, daß "kürzlich" Authentication erfolgte

Authenticator

Auth(C,S) : $\{C,t\}K_{CS}$

- enthalten timestamp : t
- gelten für ein Client-Server-Paar : CS
- verschlüsselt durch session key

- Aufgabe : Idendität des Absenders beweisen

- Werden neu berechnet.

Eigenschaften / Limitationen

- Annahmen :
  - Auf Arbeits-, Serverstation läuft tatsächlich Kerberos !!! (z.B. Passworteingabe)
  - ➔ Frage : Wie sicherstellen ???
  - Physischer Schutz von Servern (Wegsperren)

- Passwort erraten ...
- Anpassung der Service-Implementierung („Kerberizing“)

Ergänzendes zu Kerberos

Realms

- administrative Einheiten mit AS und TGS
- Server zu einer realm
- geheime Zwischen-Realm-Schlüssel

Kerberos und NFS

- Ticket pro Request : "zu teuer"
- mehrere Sicherheitsstufen

Weiter

- Journaling Dateisystem
  Carsten Weinhold in dieser Vorlesung
- Neuere Verteilte Dateisysteme
  Google FS, …. → Distributed OS(DOS)
- Sicheres Urladen, DRM etc → DOS
- Capabilities genauer → DOS