

Universität Karlsruhe (TH)

Forschungsuniversität · gegründet 1825

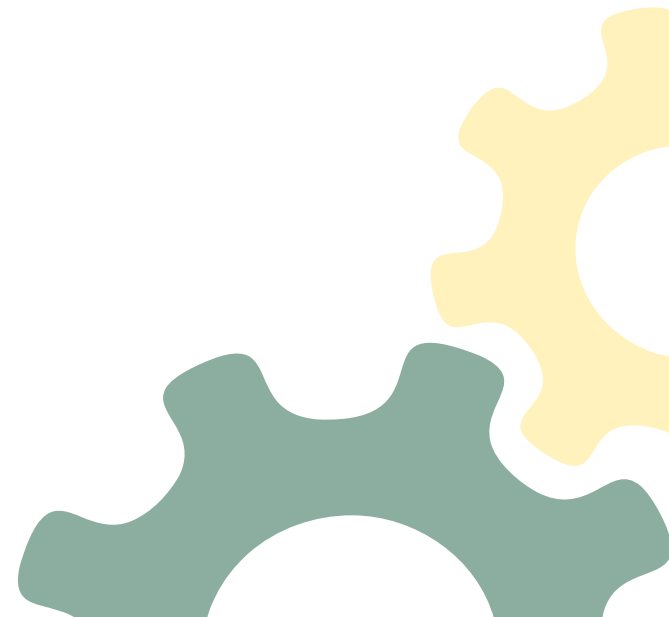
Parallele und verteilte Dateisysteme

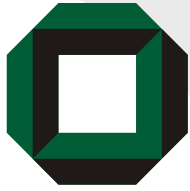
Prof. Dr. Walter F. Tichy
Thomas Moschny
Ali Jannesari



Fakultät für **Informatik**

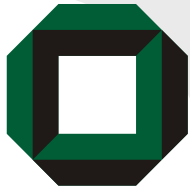
Lehrstuhl für Programmiersysteme





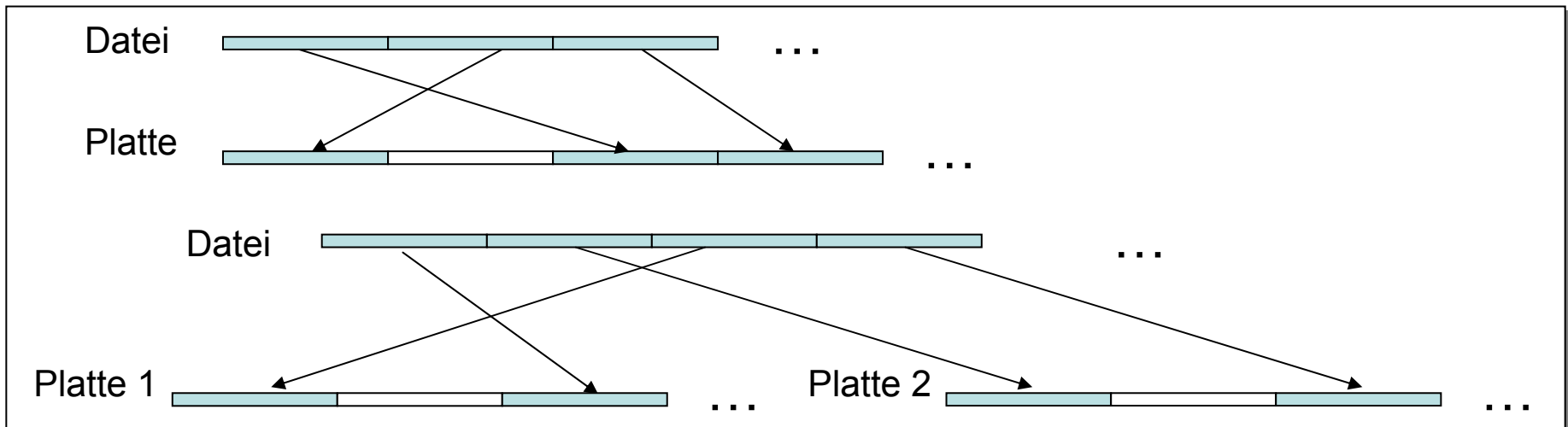
Inhalt

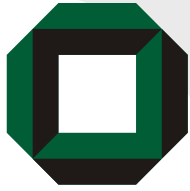
- Einführung
 - Lokale Dateisysteme
- Verteilte Dateisysteme
 - Architekturen
 - Semantik der gemeinsamen Nutzung von Dateien
 - Caching
 - Skalierbarkeit
- Parallele Dateisysteme
 - Architektur
 - Anwendungsbereich
 - Logische und physikalische Partitionen
- Zusammenfassung



Dateisystem

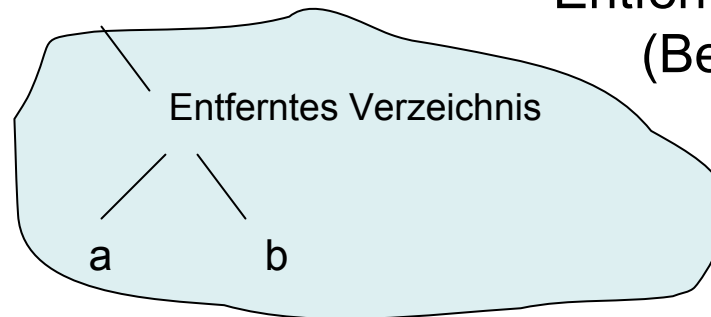
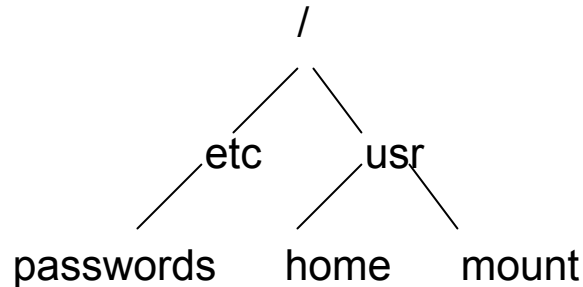
- Das Dateisystem ist der Teil des Betriebssystems, der die Plattenverwaltung implementiert.
- Plattentreiber bieten eine einfache Schnittstelle:
 - Funktion: Lesen/Schreiben von Plattenblöcken
- Das Dateisystem ordnet Plattenblöcke Dateien zu:
 - Datei \leftrightarrow virtuelle Platte



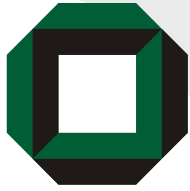


Dateisystem

- Unix bietet einen hierarchischer Namensraum.
 - Die Dateisystemschnittstelle ist (im Wesentlichen) auf allen Teilen des Baumes identisch.
 - Das Dateisystem wird zum **verteilten Dateisystem** durch (transparentes) Einhängen von entfernten Verzeichnissen.



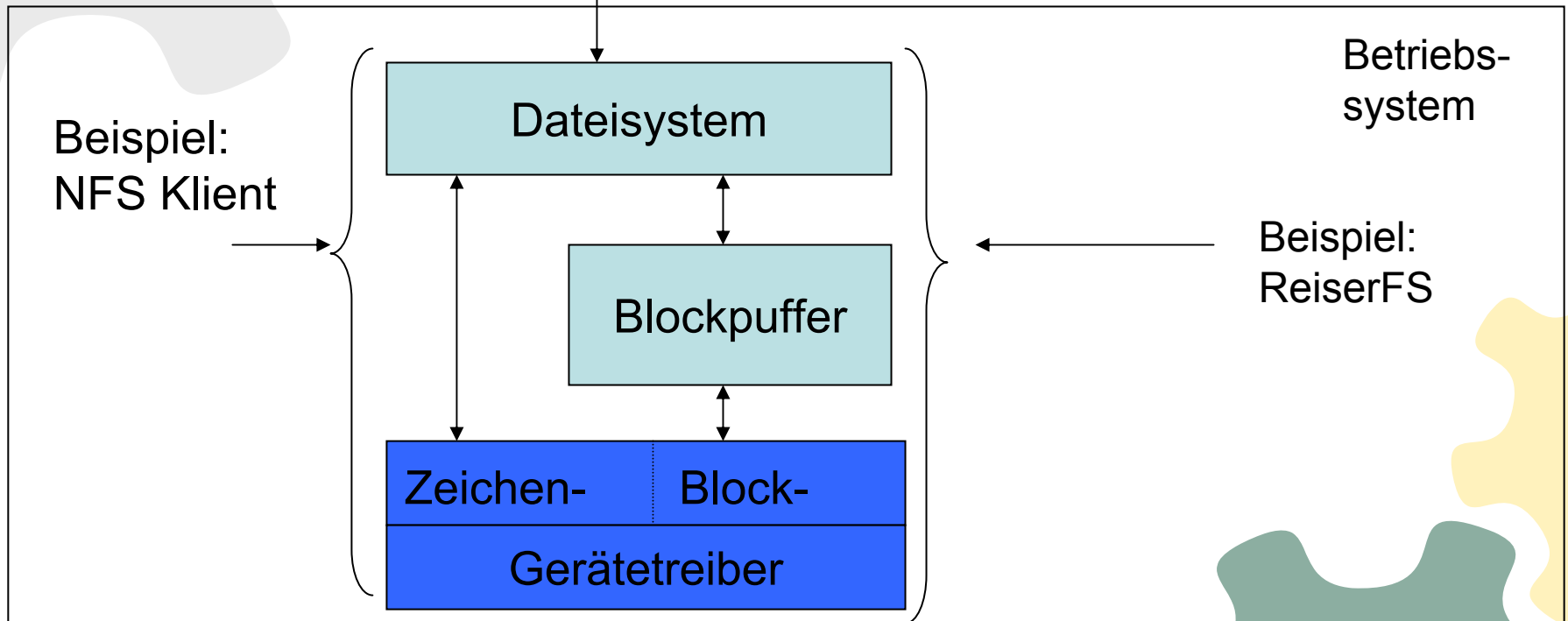
Entferntes Verzeichnis
(Beispiel: NFS)

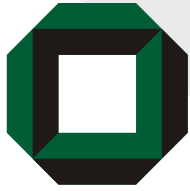


UNIX Dateisystem

Systemaufrufe: mount/open/close/read/write/seek etc.
(UNIX:Virtual File System - Schnittstelle)

Benutzer-
ebene

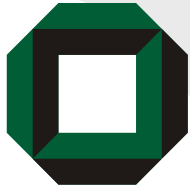




UNIX Dateisystem

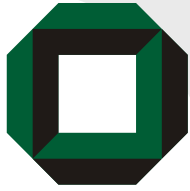
- Der *Bootblock* enthält Code für die Betriebssysteminitialisierung
- Der *Superblock* beschreibt allgemeine Dateisystemattribute (Größe, freien Platz)
- Ein *Inode* enthält die Attribute einer einzelnen Datei wie Typ (Verzeichnis, regulär), Änderungszeiten, Länge, zugeh. Datenblöcke
- Jeder *Datenblock* kann nur zu einer einzigen Datei gehören





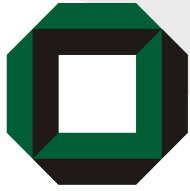
Verteilte Dateisysteme

- Die Komponenten eines verteilten Dateisystems sind auf mehrere Rechnerknoten verteilt.
- Wichtige Eigenschaften eines verteilten Dateisystems:
 - Semantik der gemeinsamen Nutzung von Dateien?
 - Architekturen?
 - Caching?
 - Skalierbarkeit?



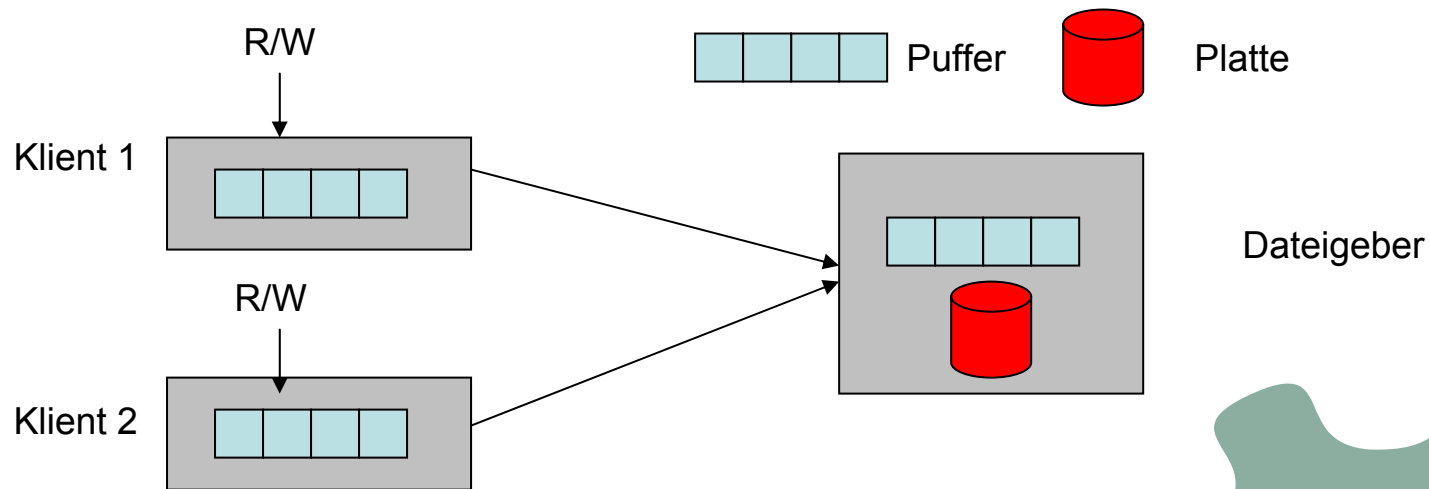
Vert. Dateisystem: Architekturen

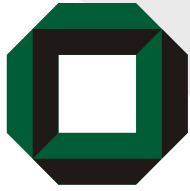
- File-Service: Beschreibung der Dienstleistungen, die ein Dateisystem anbietet (Syntax und Semantik der Aufrufe).
- Mögliche Architekturen
 - Klient/Dienstgeber
 - Achtung: Begrifflichkeit "Klient":
 - Die **Anwendung** ist **Klient** des verteilten Dateisystems.
 - Bei einer Klient/Dienstgeber-**Implementierung** des verteilten Dateisystems ist der Klient jene Komponente des Dateisystems, die mit dem Server kommuniziert.
 - Ohne Dienstgeber („serverless“ oder „peer-to-peer“)



Klient-Dienstgeber-Architektur

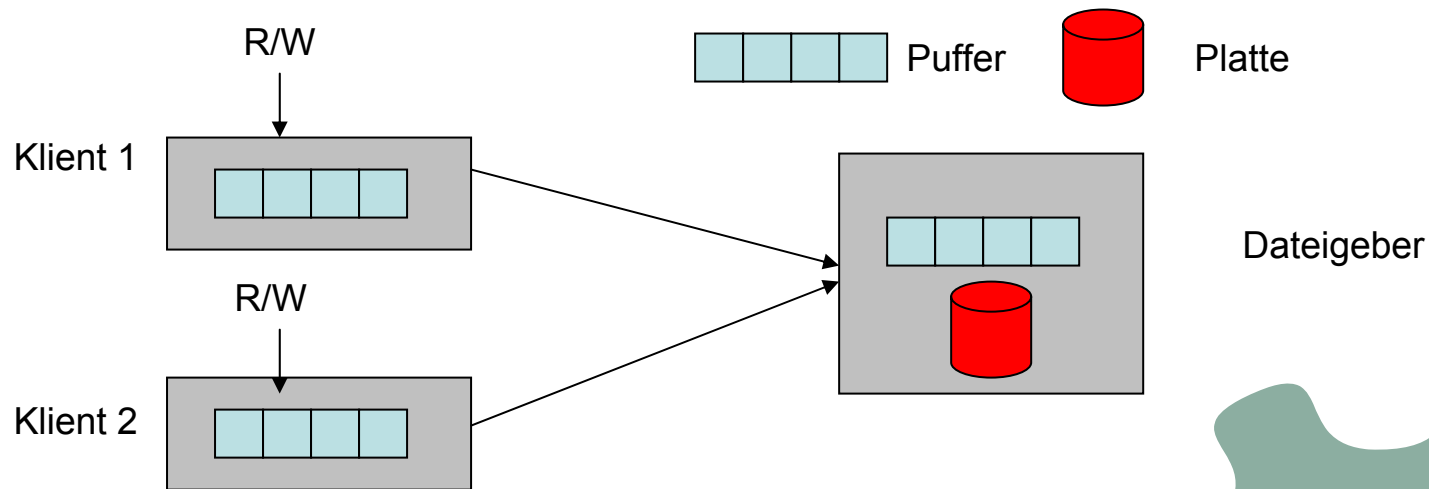
- Dateigeber (file server): Prozess, der auf einem Rechnerknoten läuft, und einen Dateidienst (file service) implementiert.
- **Klient**: Prozess, der die Anforderungen transparent für den Benutzer an einen Dateigeber über das Netzwerk verschickt.
 - Behält die am meisten benutzten Blöcke in einem lokalen Cache.
- Beispiel: Sun NFS

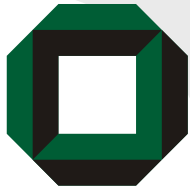




Architektur von NFS

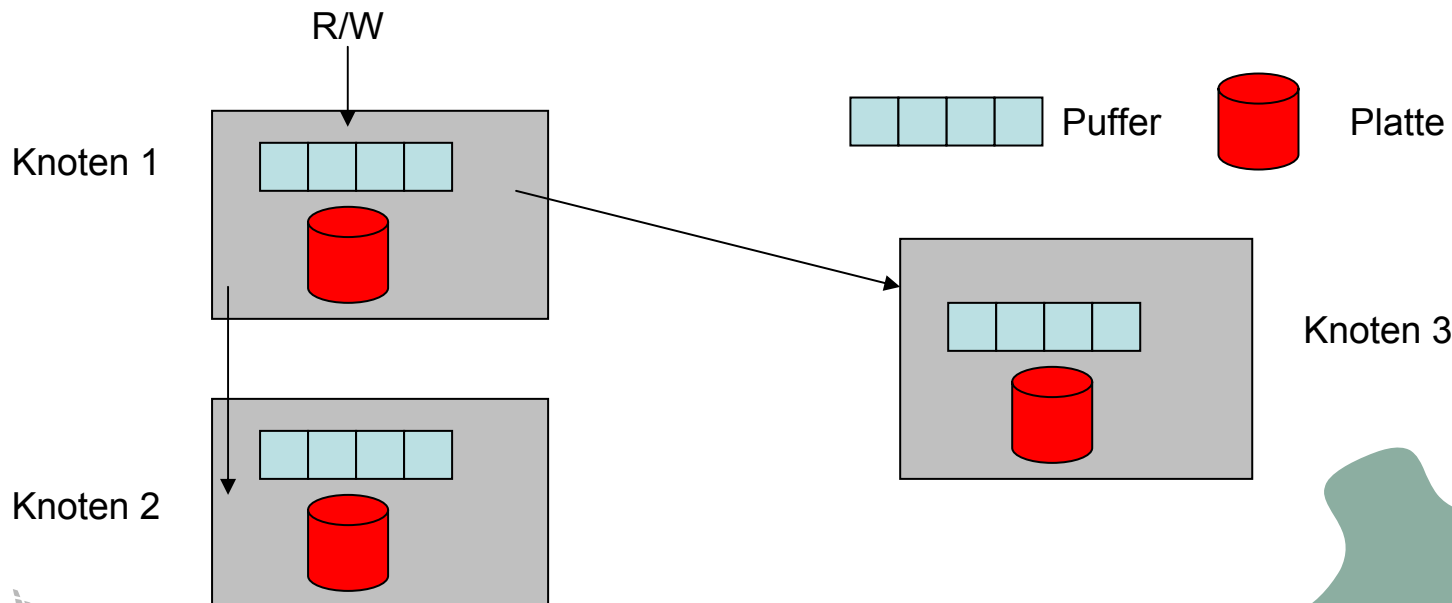
- Dateigeber: exportiert lokales Dateisystem oder nur einzelne Verzeichnisse.
- Der Klient kann die exportierten Verzeichnisse einhängen (montieren).
- Der Benutzer (die Anwendung) sieht die entfernten Verzeichnisse, als ob sie lokal wären.
- Der Dienstgeber stellt einen Flaschenhals dar.

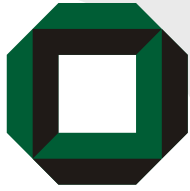




Architektur ohne Dienstgeber

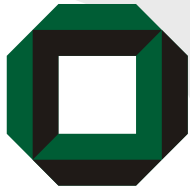
- Die Architektur besteht aus mehreren Prozessen, die gleichzeitig Klienten und Dateigeber sind.
- Eine Datei kann auf mehreren Knoten verteilt werden.
- Die Prozesse kooperieren, um einen Datei-Dienst zu implementieren.
 - z.B. kooperative Pufferung: wenn die Daten sich nicht im lokalen Puffer befinden, werden die Puffer der anderen Knoten zuerst durchsucht.
- Beispiel: Berkeleys xFS





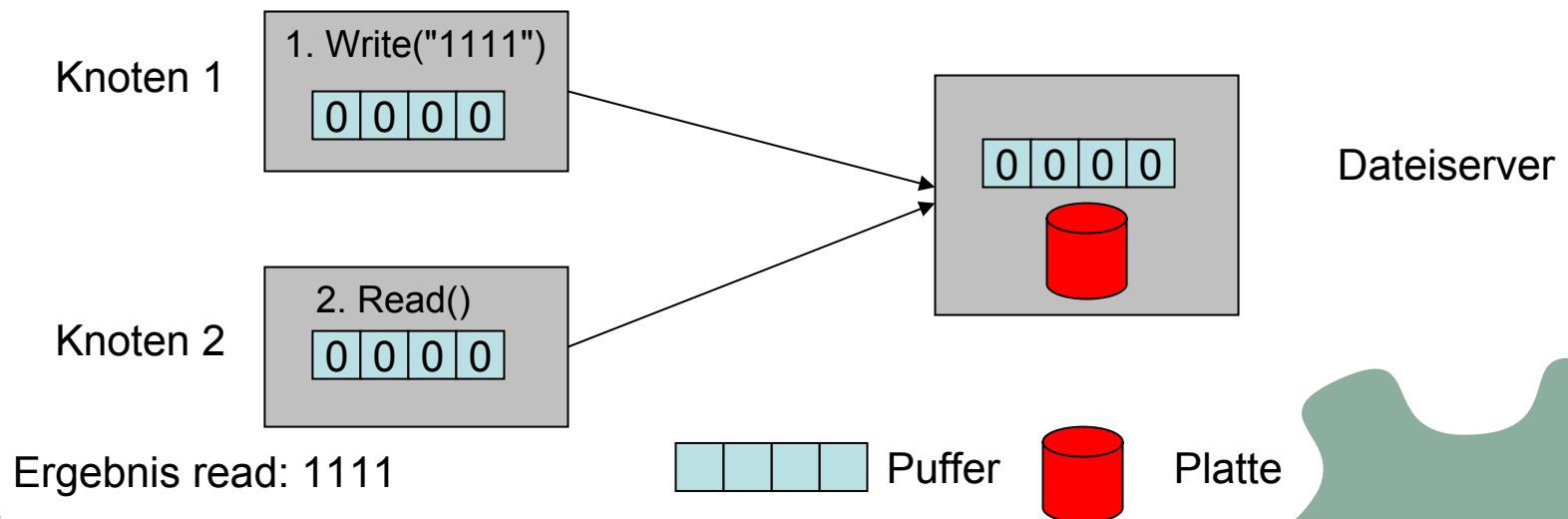
Semantik der gemeinsamen Nutzung von Dateien (1)

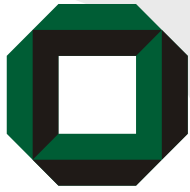
- Was garantiert das verteilte Dateisystem, wenn **mehrere** Prozesse auf **dieselbe** Datei zugreifen?
- Probleme treten auf, wenn mindestens ein Schreibzugriff dabei ist (Konflikt).
- Fragen:
 - Sind überhaupt Änderungen des Dateiinhalts erlaubt?
 - Sehen andere Prozesse diese Änderungen,
 - und wenn ja, wann?
- Wir unterscheiden:
 - Unix-Semantik
 - Sitzungs-Semantik
 - Unveränderliche Dateien
 - Transaktionen



Semantik der gemeinsamen Nutzung von Dateien (2)

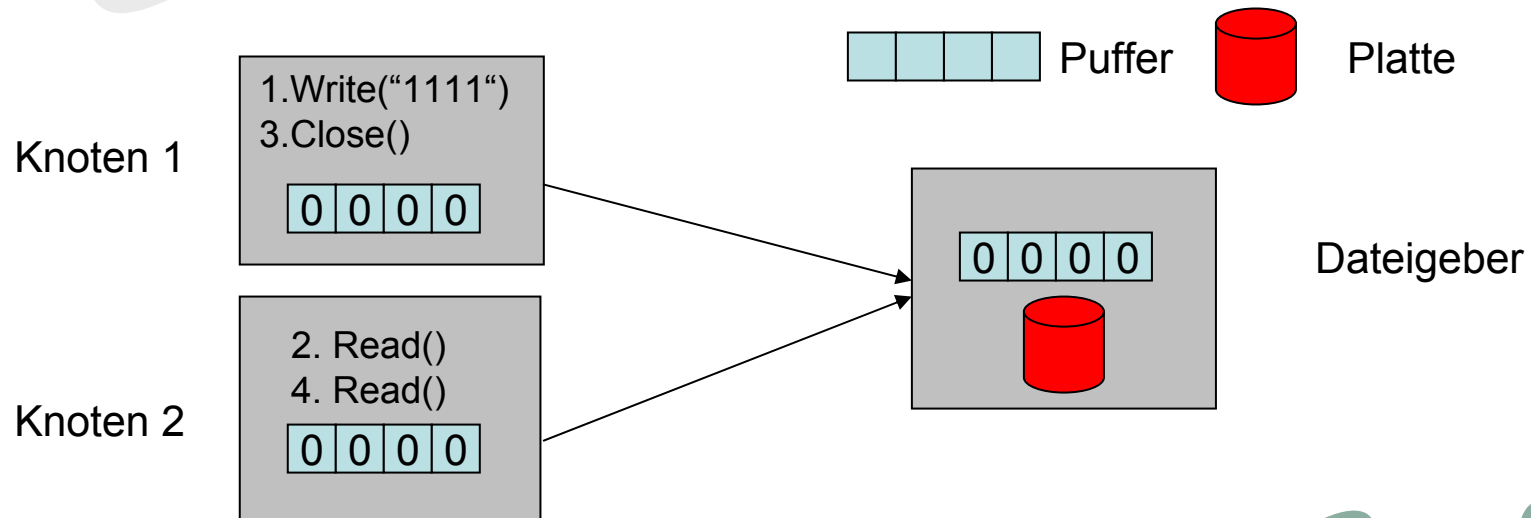
- Unix-Semantik:
 - Jede Dateioperation wird von allen anderen Prozessen **sofort** gesehen.
 - Einfach zu implementieren in einem **lokalen** Dateisystem (alle Operationen benutzen einen **gemeinsamen** Puffer).
 - **Aufwendig** in einem verteilten Dateisystem, insbesondere, falls die Knoten lokale Puffer für die entfernten Dateien benutzen.
 - Definition von „Gleichzeitigkeit“ bei verteilten System schwierig.





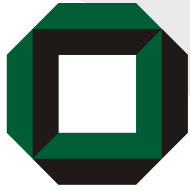
Semantik der gemeinsamen Nutzung von Dateien (3)

- Sitzungs-Semantik
 - Abgeschwächte Semantik.
 - Die Änderungen eines Knotens an einer Datei werden auf den anderen Knoten erst sichtbar, wenn die Datei geschlossen wird.



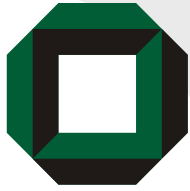
Ergebnis beim ersten Lesen: 0000

Ergebnis beim zweiten Lesen: 1111



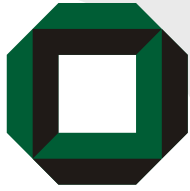
Semantik der gemeinsamen Nutzung von Dateien (4)

- Unveränderliche Dateien:
 - Erlaube keine Änderungen der Dateien.
 - Änderungen sind nur möglich durch Herstellung einer modifizierten neuen Datei und Löschung der alten.
 - Dieses Schema erlaubt effiziente gemeinsame Nutzung und Replikation der Dateien.
 - Schema ist allerdings uneffizient für Änderungen an großen Dateien.
 - „Abwälzen“ der Konsistenz-Problematik auf die Anwendung.
 - sie muss z.B. selbst die beteiligten Prozesse darüber informieren, wenn eine neue, geänderte Datei angelegt wurde.

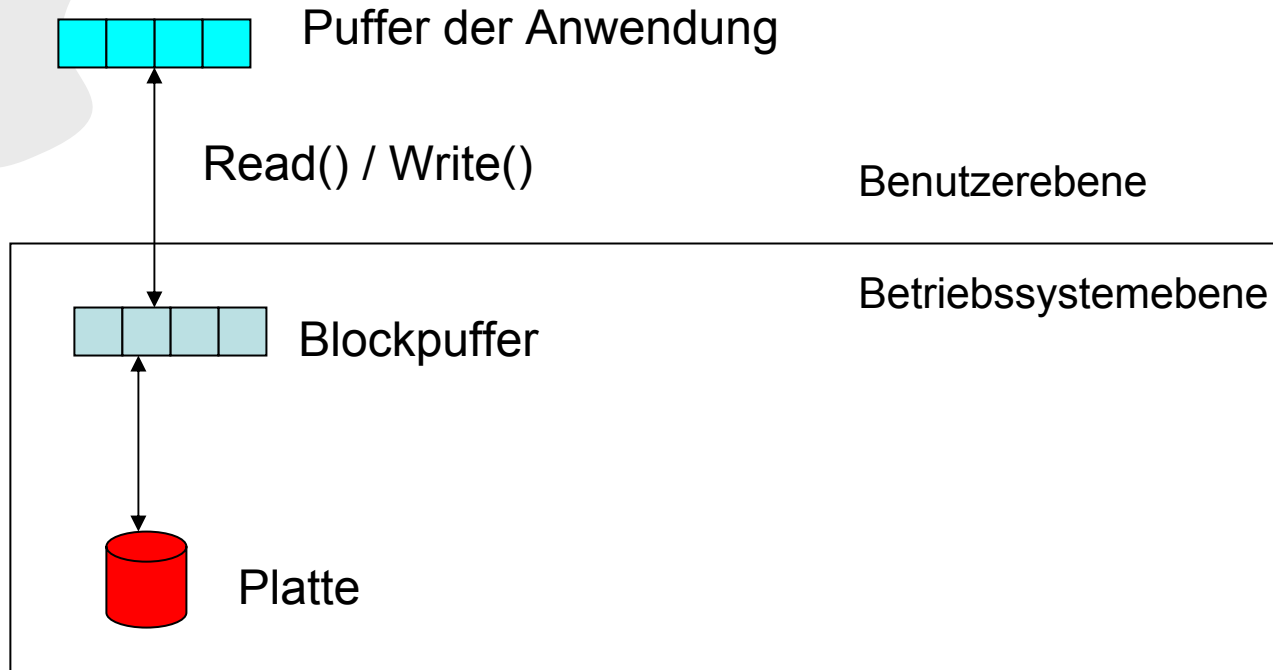


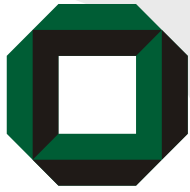
Semantik der gemeinsamen Nutzung von Dateien (5)

- Atomare Transaktionen
 - Jede Änderung wird vollständig ausgeführt oder scheitert.
 - Das ist die Datenbanksemantik.
 - Erfordert eine Erweiterung der Standard-Dateisystem-Schnittstelle, die diese Semantik nicht unterstützt.
 - Hier nicht weiter behandelt.



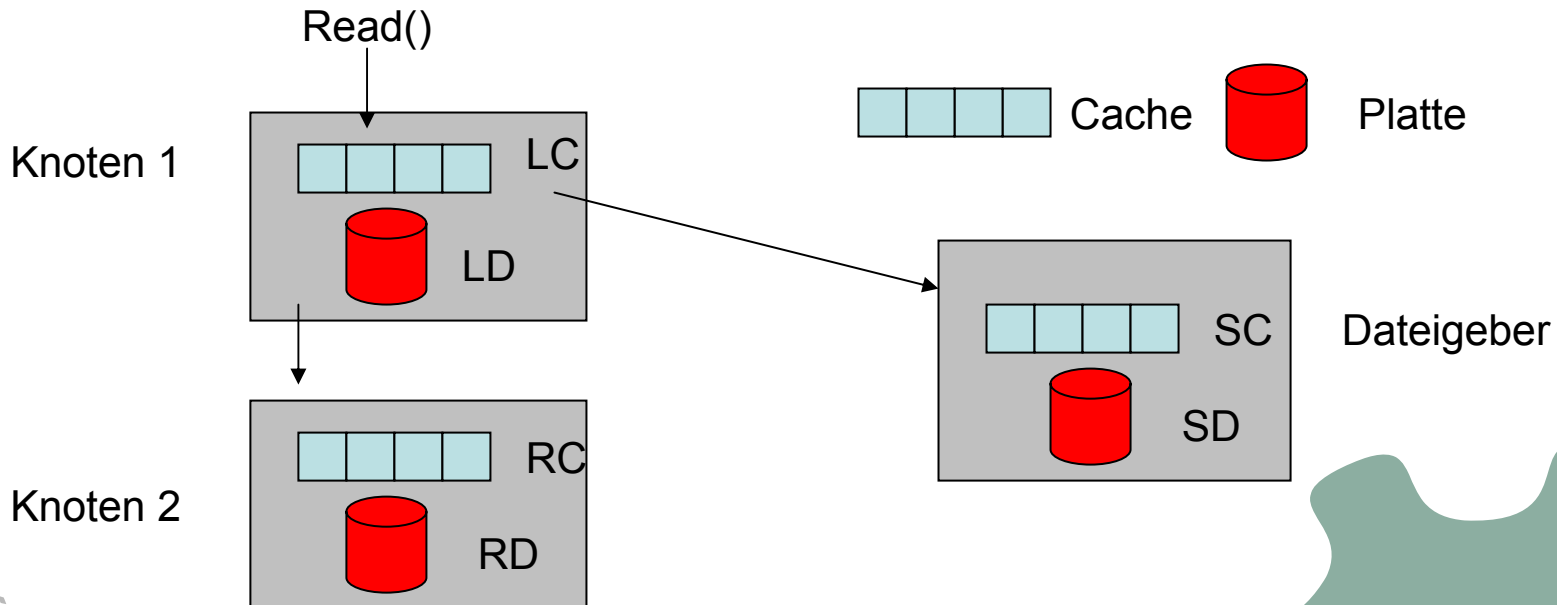
Pufferung: einfachster Fall

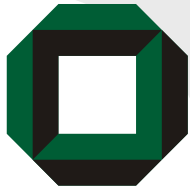




Pufferung

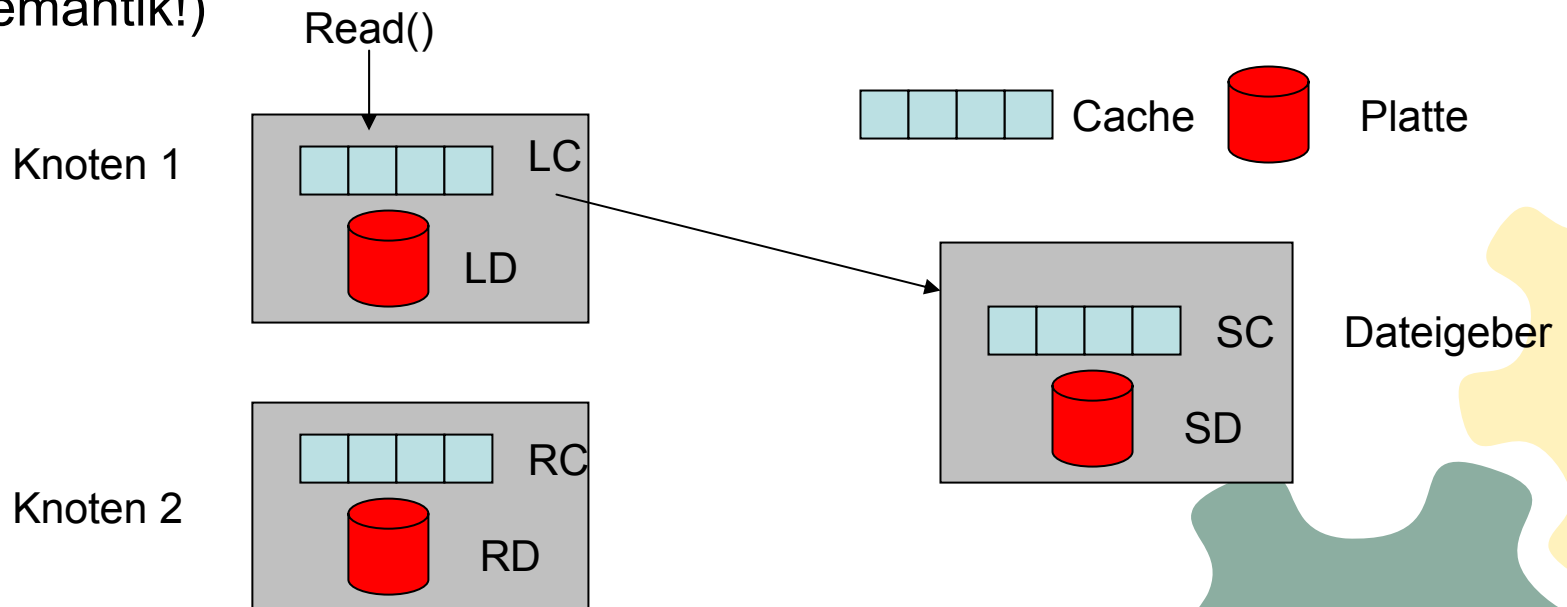
- Wo können sich die Dateiblöcke aus der Sicht eines Knotens befinden?
 - Platte/Puffer des Dateigebers (Server Disk (SD) und Server Cache (SC)), wenn es einen solchen gibt
 - Lokale Platte/lokaler Puffer des Knotens (LD und LC)
 - Platte/Puffer eines entfernten Knotens (RD und RC)

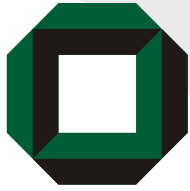




NFS Pufferung

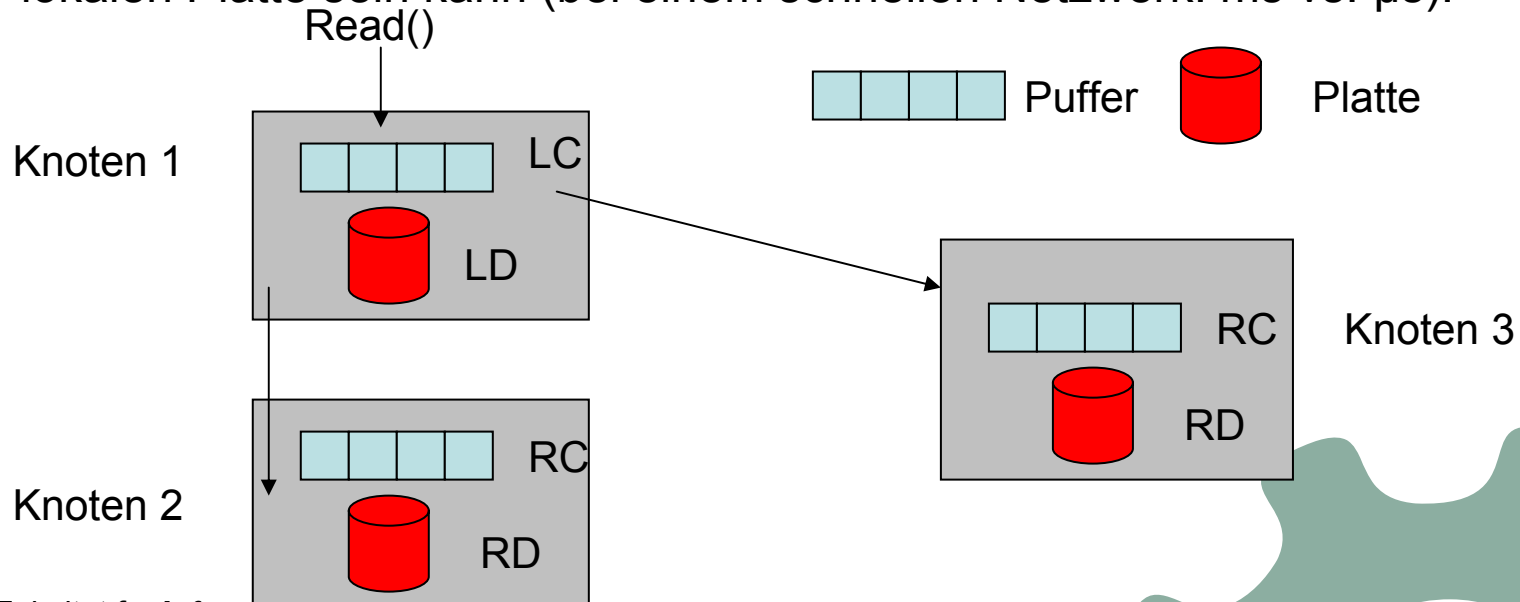
- Suchreihenfolge des Klienten:
 - lokaler Puffer (LC),
 - Puffer des Dateigebers (SC),
 - Platte des Dateigebers (SD)
- Änderungen des Klienten werden alle 3s (für den Dateiinhalt) bzw. alle 30s (Dateiattribute) an den Server geschickt. (keine UNIX Semantik!)





XFS Pufferung

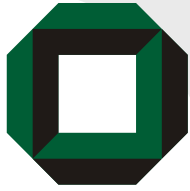
- Suchreihenfolge des Klienten:
 - lokaler Puffer (LC),
 - Puffer der anderen Knoten (RC),
 - lokale Platte (LD),
 - Platten der anderen Knoten (RD)
- Puffer werden zuerst durchsucht, weil das Lesen aus dem Puffer eines entfernten Knoten 2 bis 3 Größenordnungen schneller als das Lesen von der lokalen Platte sein kann (bei einem schnellen Netzwerk: ms vs. μ s).





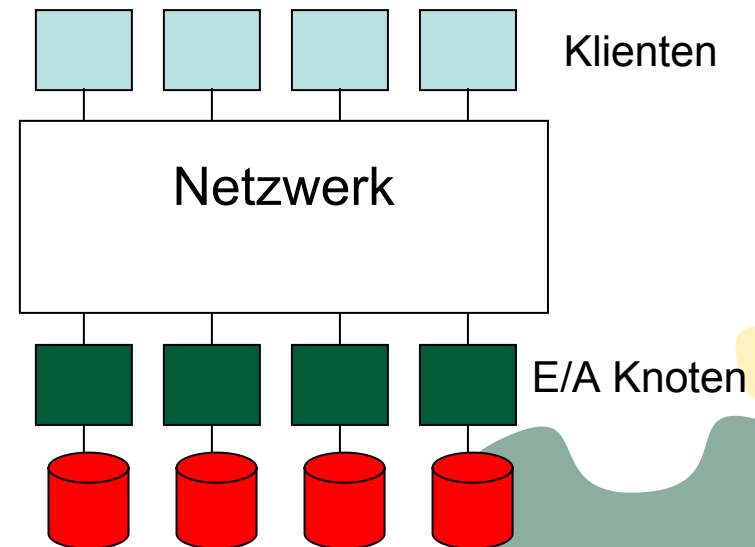
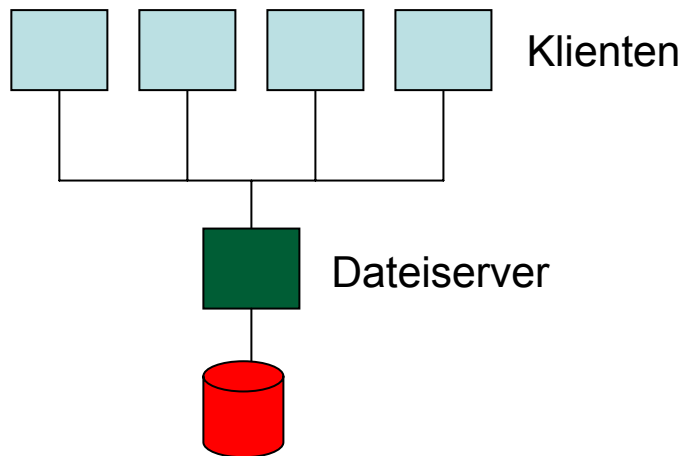
Skalierbarkeit: Datei reproduzieren oder Datei verteilen?

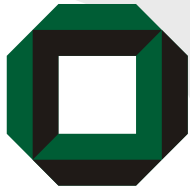
- Wenn eine Datei sich auf einem einzigen Dienstgeber befindet, dann stellt dieser einen Flaschenhals dar.
- Lösung 1: Datei-Replikation auf mehrere Dienstgeber
 - Gut, wenn die Dateien vollständig gelesen werden (z.B. ausführbare Dateien).
 - Nachteil: Erhaltung der Konsistenz beim Schreiben ist sehr aufwendig.
- Lösung 2: Aufspaltung einer Datei auf mehreren Dienstgeber
 - Gut, wenn unterschiedliche Knoten unterschiedliche Teile einer Datei lesen/schreiben (d.h. eine Partitionierung vorliegt).
 - Nachteil: Wenn Dateien vollständig gelesen werden, kontaktieren alle Knoten alle Server (skaliert nicht).



Parallele Dateisysteme

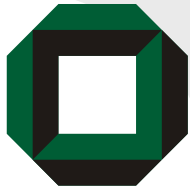
- In einem verteilten Dateisystem kann man den parallelen Zugriff auf eine gegebene Datei nicht steuern.
- NFS: keine Parallelität innerhalb einer Datei; die parallelen Zugriffe werden vom Dateigeber serialisiert.





Warum Parallelität beim Dateizugriff? (1)

- Mehrere Prozesse einer einzigen Anwendungen (typischerweise: SPMD, MPI) laufen auf verschiedenen Prozessoren/Rechenknoten,
 - Die dadurch induzierte Parallelität nennt man „internal parallelism“.
- Die Prozesse kooperieren
 - durch direkte Kommunikation (i. A. Nachrichtentausch),
 - über verteilten gemeinsamen Speicher,
 - über ein gemeinsam genutztes Dateisystem.
- Verschiedene Anwendungen, die gleichzeitig auf demselben Rechnerbündel ablaufen („external parallelism“):
 - Dateigeber, Datenbanken, Webserver, interaktive Programme.

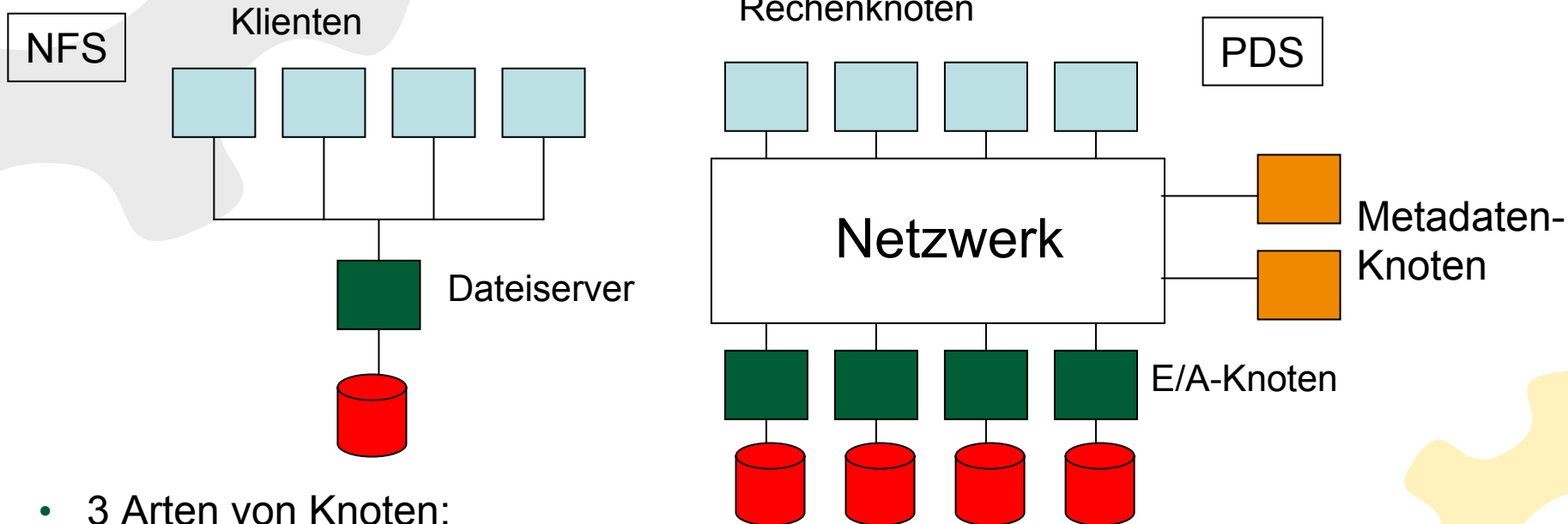


Warum Parallelität beim Dateizugriff? (2)

- Parallelverarbeitung sollte nicht beim Dateizugriff Halt machen:
 - Serielle Stellen in einem parallelen Programm verschlechtern die Performanz wesentlich (Amdahls Gesetz).
 - Es ist wichtig, die im Rechnerbündel vorhandene Parallelität auszunutzen!
 - Genauso wie die DSM-Abstraktion für den Speicherzugriff erlaubt der gemeinsame, parallele Zugriff auf eine (möglicherweise große, und möglicherweise auf verschiedene E-/A-Knoten verteilte) Datei eine elegante Formulierung für einen parallelen Algorithmus.



Architektur eines Parallelen Dateisystems (PDS) im Vergleich zu NFS

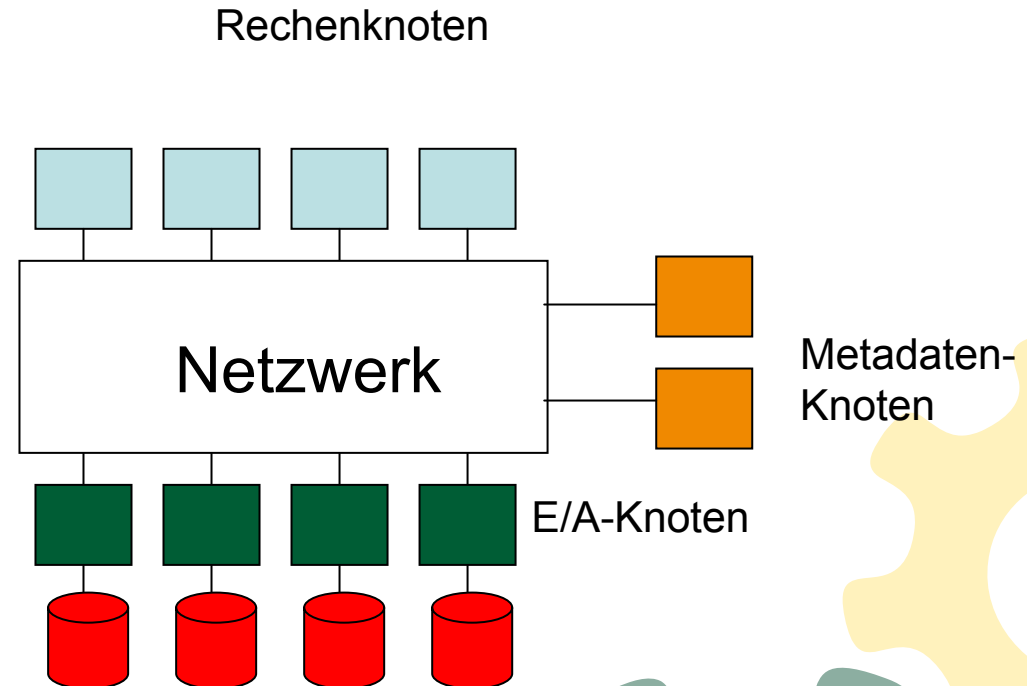


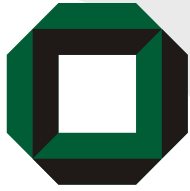
- 3 Arten von Knoten:
 - E/A-Knoten speichern die Dateien auf Ihren Platten.
 - Die parallele Anwendungen laufen auf Rechenknoten.
 - Die Metadaten-Knoten verwalten die Dateiattribute (Metadaten).
- **Wichtig:** Jeder Knoten in einem Rechnerbündel kann mehrere Rollen spielen.



Paralleles Dateisystem: Operationen

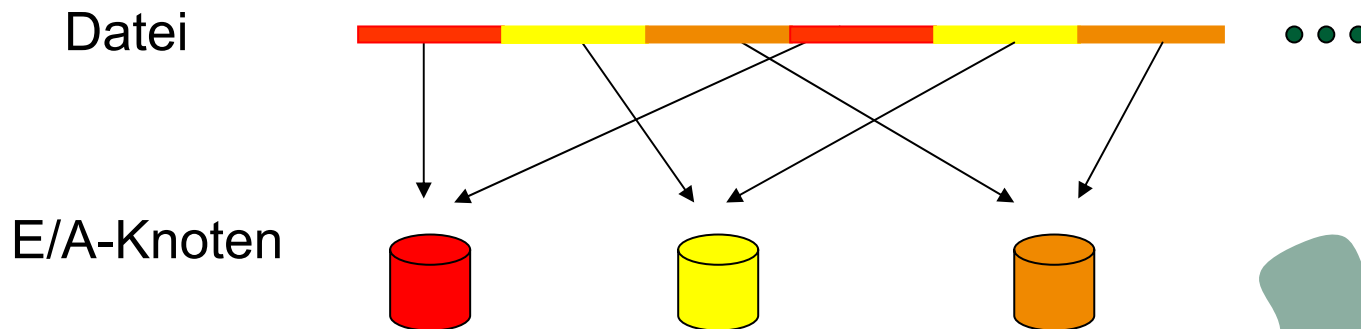
- Jeder Rechenknoten kann mit jedem E/A-Knoten direkt kommunizieren.
- Die physikalische Verteilungs-Information wird von Metadaten-Knoten verwaltet.
- Rechenknoten können selbst die physikalische Position der Dateiblöcke (E/A-Knoten + Plattenposition) berechnen.
- Daten können entweder auf Rechenknoten oder E/A-Knoten zwischengespeichert werden.

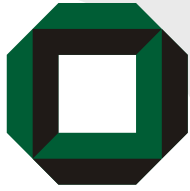




Physikalische Partitionierung einer Datei

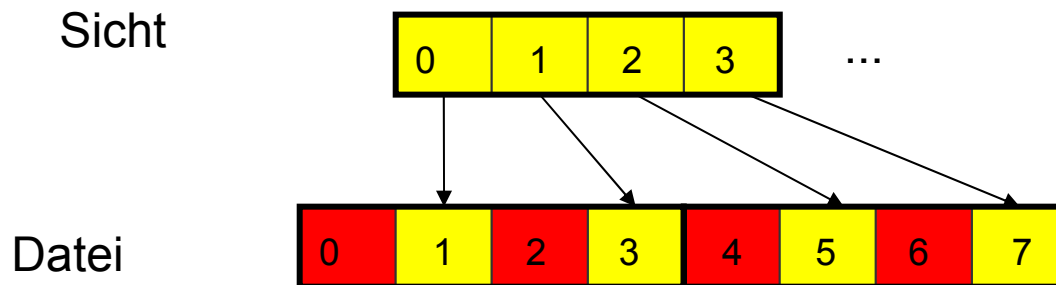
- Eine Datei wird auf mehrere E/A Knoten verteilt.
- Die häufigste Verteilung ist die Umlauf-Verteilung von Dateiblöcken über alle E/A Knoten im Stil von RAID 0.
(Beispiele: PVFS der Universität Clemson, GPFS von IBM)
- Andere physikalische Verteilungen:
 - Zweidimensionale, rechteckige Blöcke (Vesta von IBM).
 - N-dimensionale Blöcke von Kantenlänge 2^N (nCube von E/A Software).
 - Beliebig (ClusterFile, Universität Karlsruhe).

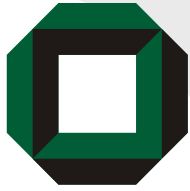




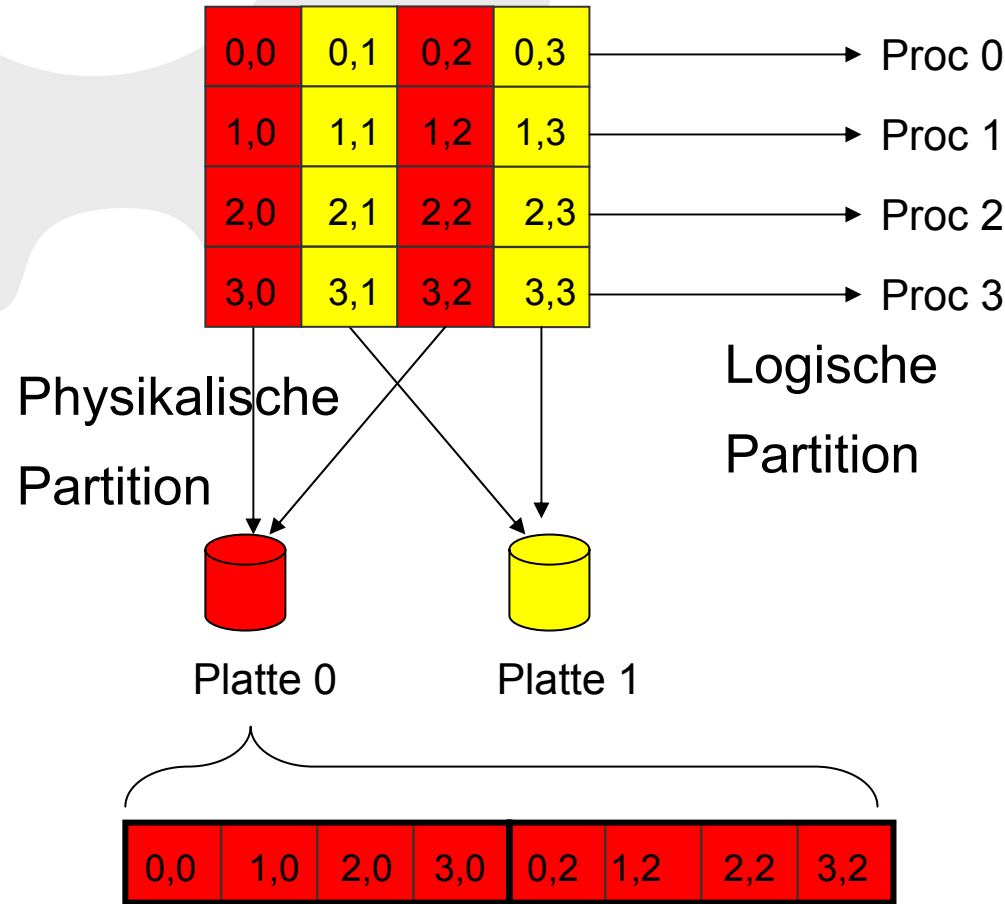
Logische Partitionierung einer Datei

- Wie wird eine Datei zwischen den Prozessen einer parallelen Anwendung partitioniert?
- Sicht: eine Untermenge einer Datei mit linearen Adressen.
 - Eine Sicht wird von einem Rechenknoten gesetzt.
 - Alle Dateioperationen können auf eine Sicht angewandt werden.
 - Vorteil: komplexe Zugriffsmuster können in einem einzigen Aufruf ausgeführt werden.

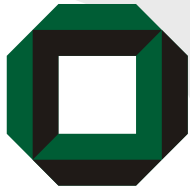




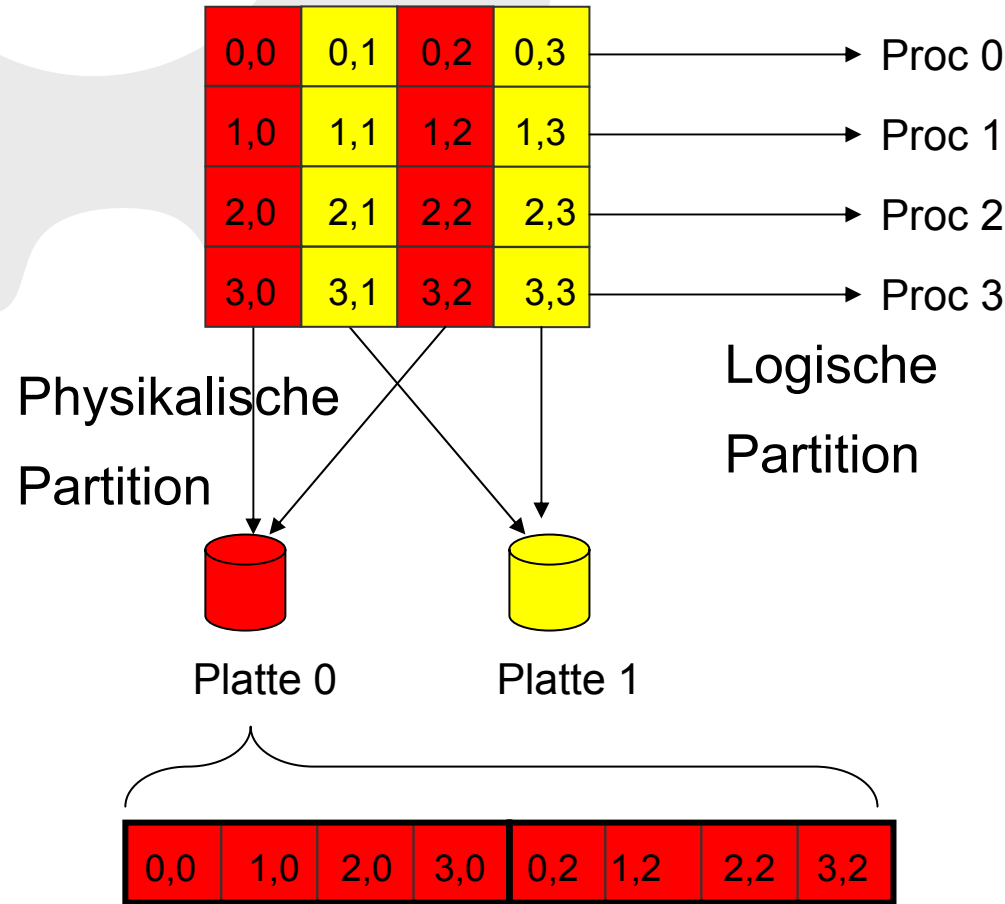
Parallele Anwendungen: E/A Analyse (1)



- Parallele wissenschaftliche Anwendungen: Studien an der Universität von Illinois und am Dartmouth College
- Eine Datei ist physikalisch auf mehreren Platten verteilt.
- Eine Datei wird gemeinsam von mehreren Knoten benutzt.
- Die Dateizugriffe sind oft verschachtelt.
- Die Zugriffe überlappen sich selten.
- Multidimensionale Felder sind die meistbenutzten Datenstrukturen.



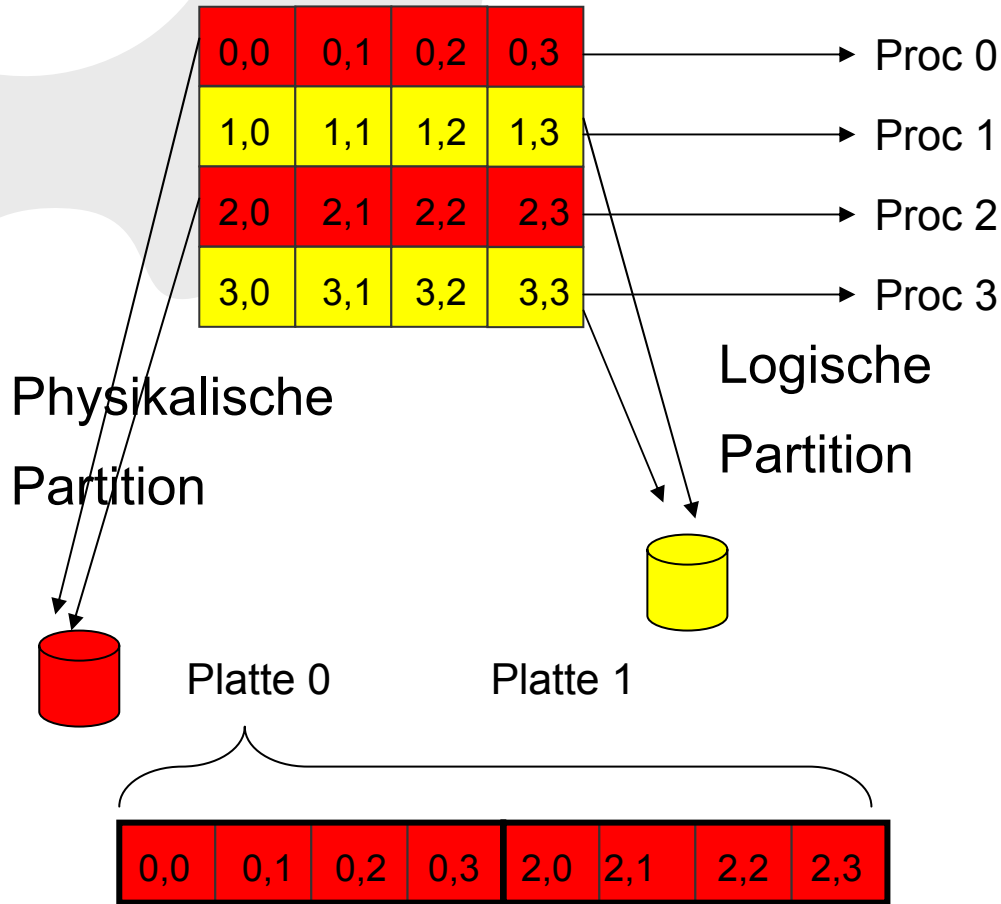
Parallele Anwendungen: E/A Analyse (2)



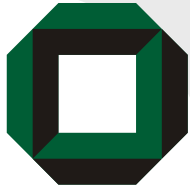
- Die parallelen E/A Zugriffe ergeben viele kleine Nachrichten.
- Überlastung von Platten (Extremfall: alle Prozessoren schicken Anforderungen an alle Platten).
- Die Plattennutzung kann unbalanciert sein: zu einem bestimmten Zeitpunkt kann eine Platte Aufforderungen von mehreren Prozessoren empfangen, während andere Platten inaktiv sind.
- Aus der Sicht eines Prozessors liegen die Daten fragmentiert auf den Platten .
- Hauptgrund: schlechte Anpassung zwischen physikalischer und logischer Partitionierung .



Parallele Anwendungen: E/A Analyse (3)



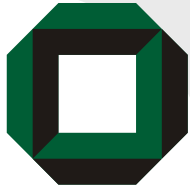
- In diesem Fall passen physikalische und logische Partitionierung genau aufeinander.
- Hohe Performanz möglich.
- Man kann es als Aufgabe eines parallelen, verteilten Dateisystems ansehen, eine physikalische Partitionierung zu finden, die gut zu den verwendeten logischen Partitionierungen passt
- Aber: Es können gleichzeitig, oder direkt hintereinander unterschiedliche logische Part. verwendet werden, deshalb keine leichte Aufgabe.



Überblick Partitionierung

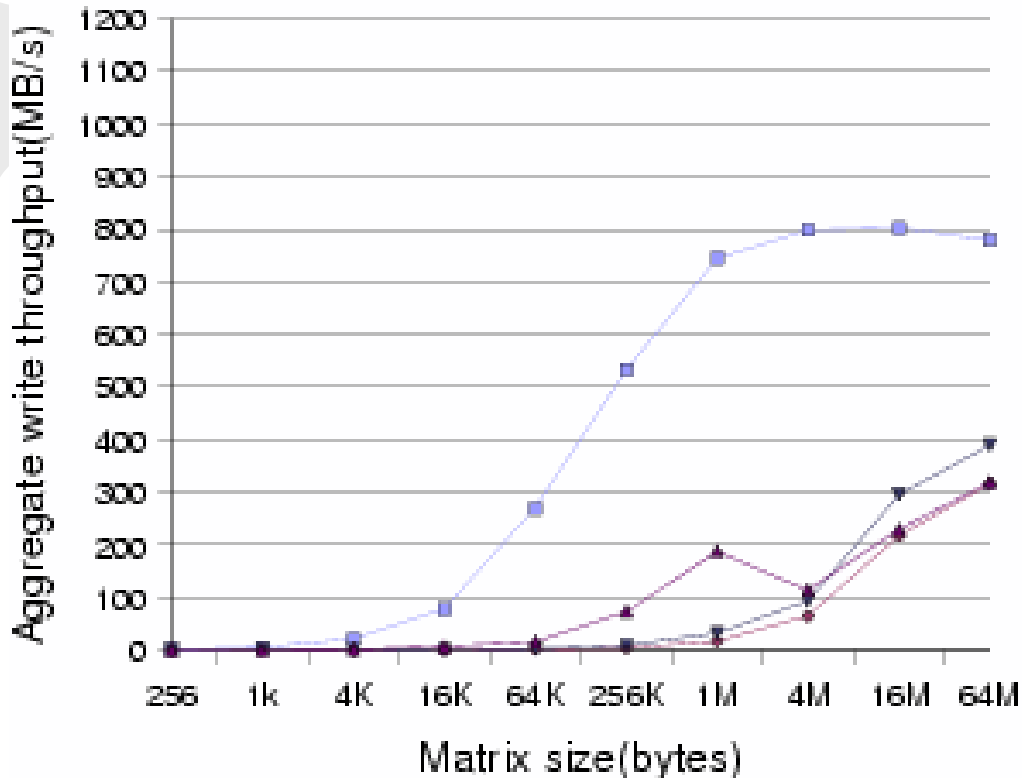
	Physikalische Partitionierung	Logische Partitionierung
Vesta	2 dim., rechteckige Blöcke	2-dim., rechteckige Blöcke
PVFS	Im Umlauf verteilte Blöcke	N-dim., rechteckige Blöcke
MPIIO	abhängig vom gewählten Dateisystem (z.B. Clusterfile)	beliebig
nCUBE I/O software	N-dimensionale Blöcke der Kantenlänge 2^N	N-dimensionale Blöcke der Kantenlänge 2^N
Clusterfile IPD, F. Isaila u.a.	beliebig	beliebig

vergl. spätere Folien



Gesamtdurchsatz „Schreiben“ in Dateipuffer

Aggregate write throughput

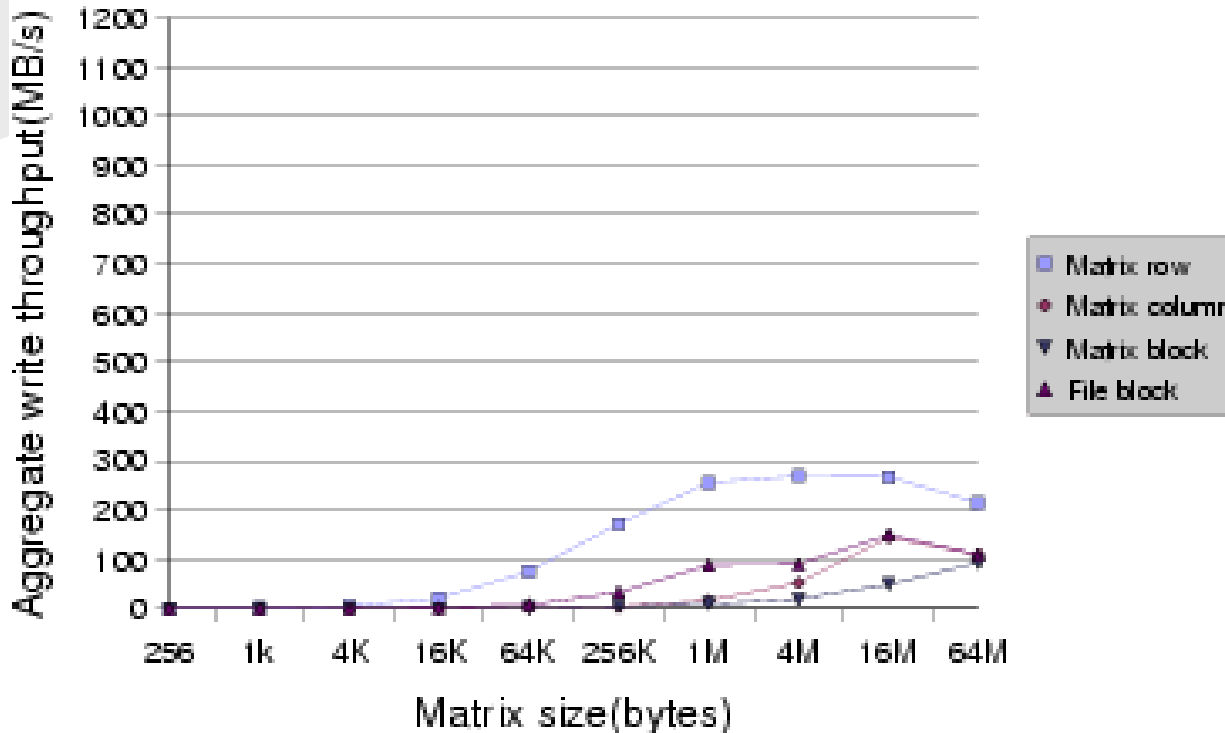


- 16 Rechenknoten
- 4 E/A Knoten
- 1 logische Partitionierung
 - Matrixreihenblöcke
- 4 physikalische Partitionierungen
 - Matrixreihenblöcke
 - Matrixspaltenblöcke
 - Matrixquadratblöcke
 - Umlauf-verteilt
- Pentium III 1GHz
- Verbunden mit Myrinet



Gesamtdurchsatz „Schreiben auf Platte“

Aggregate write throughput



- 16 Rechenknoten
4 E/A Knoten

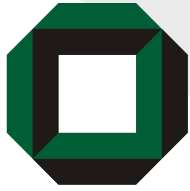
1 logische Partitionierung

- Matrixreihenblöcke

4 physikalische Partitionierungen

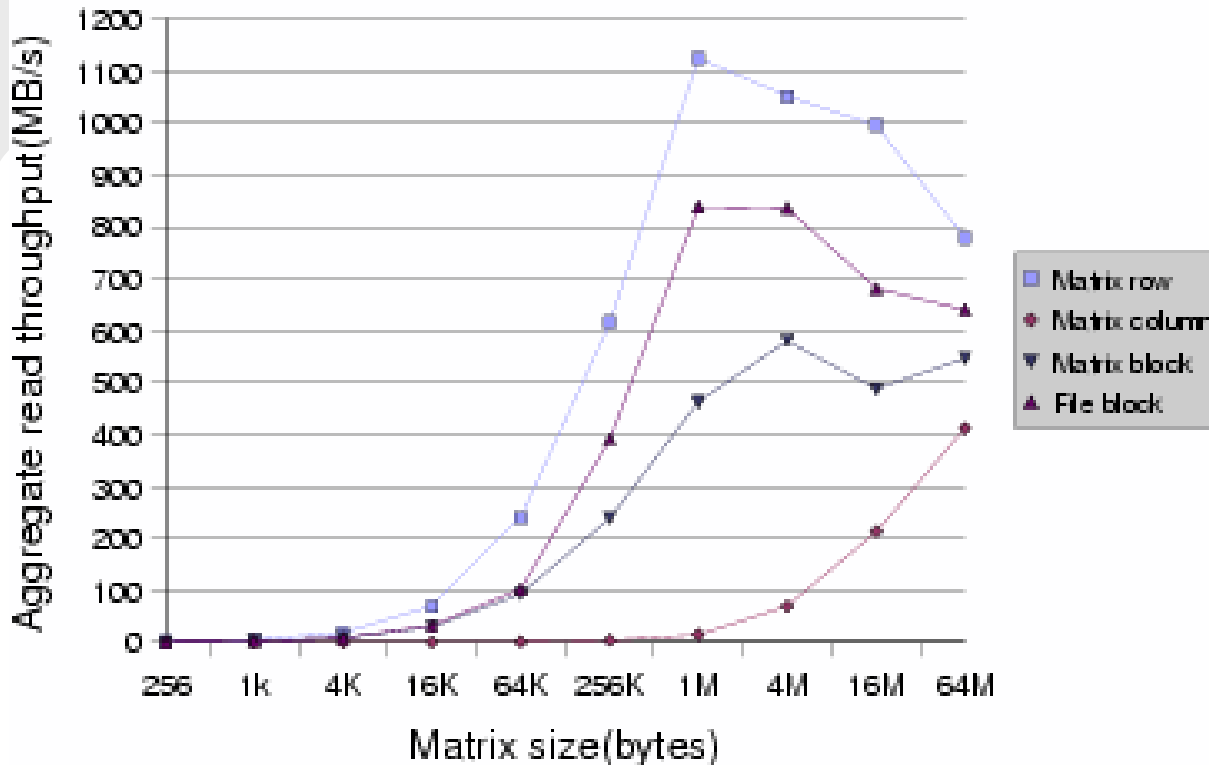
- Matrixreihenblöcke
- Matrixspaltenblöcke
- Matrixquadratblöcke
- Umlauf-verteilt

Pentium III 1GHz
Verbunden mit Myrinet



Gesamtdurchsatz „Lesen“

Aggregate read throughput

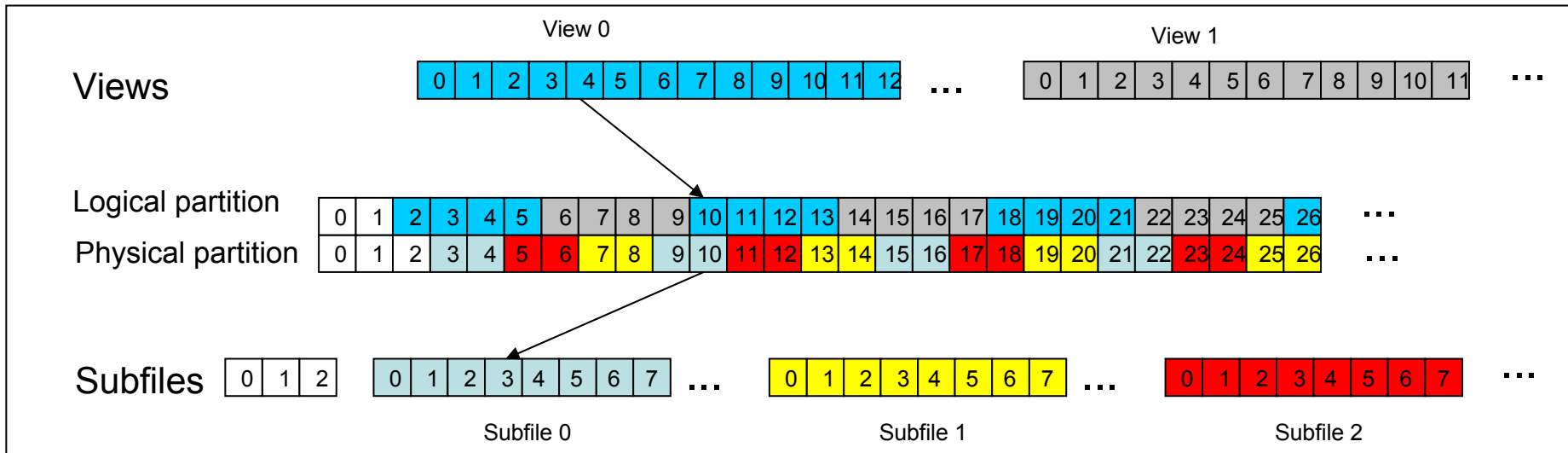


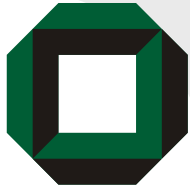
- 16 Rechenknoten
- 4 E/A Knoten
- 1 logische Partitionierung
 - Matrixreihenblöcke
- 4 physikalische Partitionierungen
 - Matrixreihenblöcke
 - Matrixspaltenblöcke
 - Matrixquadratblöcke
 - Umlauf-verteilt
- Pentium III 1GHz
- Verbunden mit Myrinet



ClusterFile: Dateimodell

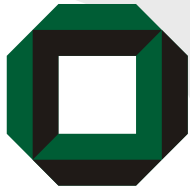
- Eine Datei ist eine linear adressierbare Folge von Bytes.
- Physikalische Partitionierung:
 - beschreibt, wie sich die Datei aus den auf E-/A-Knoten gespeicherten "Teildateien" zusammensetzt.
- Logische Partitionierung:
 - beschreibt die Aufteilung der Datei in "Sichten". Jeder Prozess kann seine eigene Sicht auf einen Teil der Datei definieren.
 - Eine Sicht ist eine linear adressierbare Folge von Datenbytes, die eine Untermenge der Datei bilden.





ClusterFile: Beschreibung der Partitionierung

- ClusterFile verwendet zur Beschreibung der logischen und der physikalischen Partitionierungen jeweils sog. nested PITFALLS
 - steht für „**p**rocessor **i**ndexed **t**agged **f**amily of **l**ine **s**egments“, hier nicht weiter vertieft.
 - geschachtelte Tupel von Ganzzahlen, die komprimiert eine reguläre Partitionierung einer Datei beschreiben.
 - besonders geeignet für die Beschreibung von regulären Partitionen von mehrdimensionalen Feldern (zyklisch, blockzyklisch, ... → spätere Vorlesung)
 - Durch die Schachtelung sind im Prinzip beliebig komplizierte Partitionierungen beschreibbar, analog zu den benutzerdefinierten Datentypen von MPI.



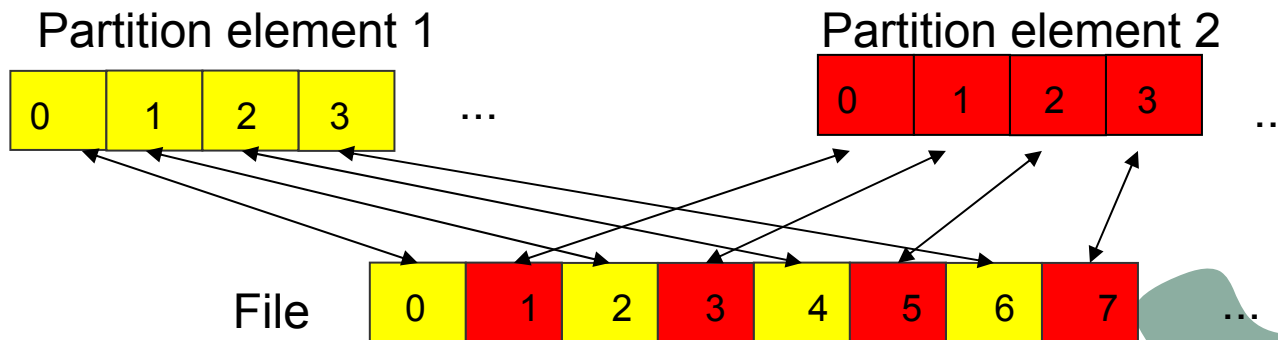
ClusterFile

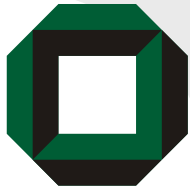
Abbildungsfunktionen (1)

- Für jede Partitionierung P bildet die Abbildungsfunktion MAP die Adresse innerhalb der Datei auf ein Tupel aus Nummer der Partition und Offset innerhalb der Partition ab:

$$MAP_P(\text{offset})=(i,x) ; MAP_P^{-1}(i,x)=\text{offset}$$

- Beispiel: $MAP_P(7)=(2,3)$





ClusterFile

Abbildungsfunktionen (2)

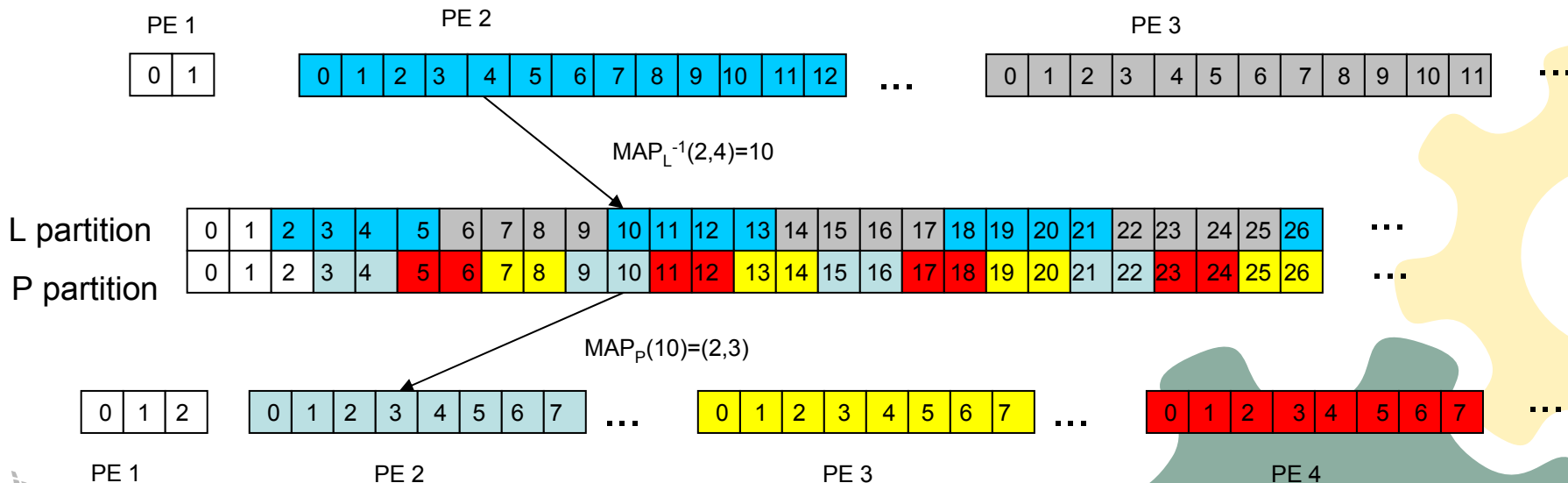
- Gegeben seien zwei Partitionierungen P und L derselben Datei und zwei Elemente $l \in L$ und $p \in P$, dann gilt:

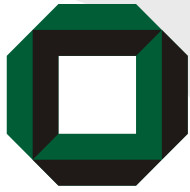
$$(l, \text{off}_l) = \text{MAP}_L(\text{MAP}_P^{-1}(p, \text{off}_p))$$

- Um den physikalischen Speicherort eines Bytes einer Sicht herauszufinden, muss man also die inverse Abbildungsfunktion von L und die Abbildungsfunktion von P komponieren.

Eine effiziente Implementierung sollte

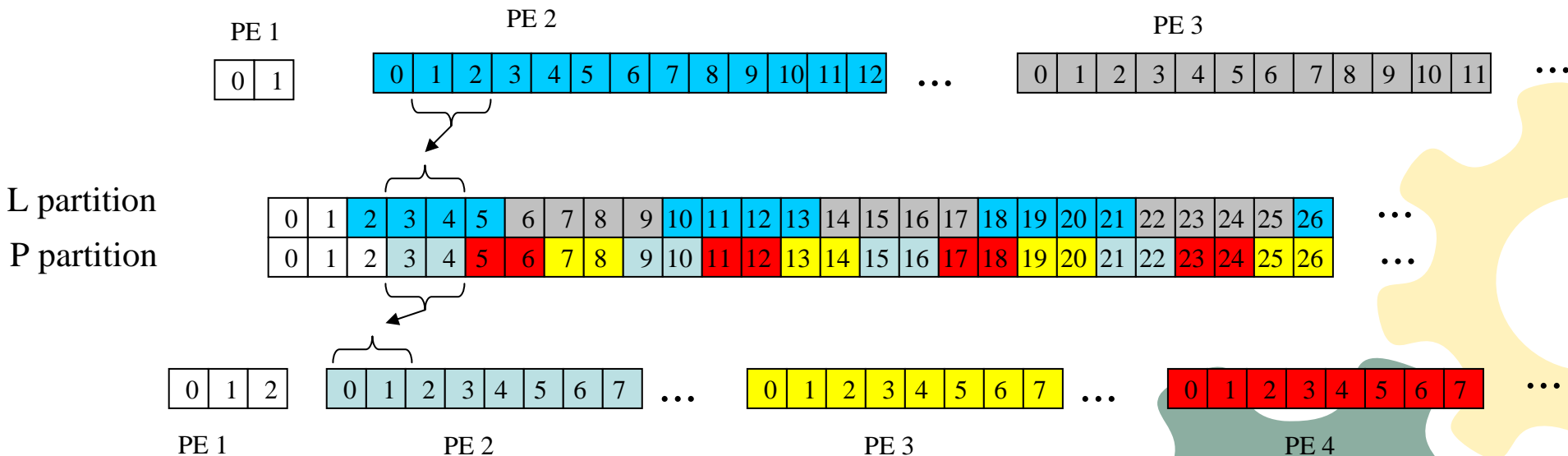
- nicht für jedes einzelne Byte diese Berechnung neu ausführen, sondern
- die Regularität der Partitionierung ausnutzen.

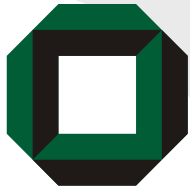




ClusterFile Abbildungsfunktionen (3)

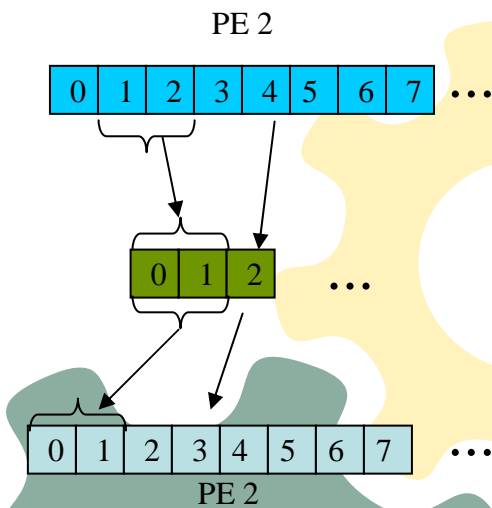
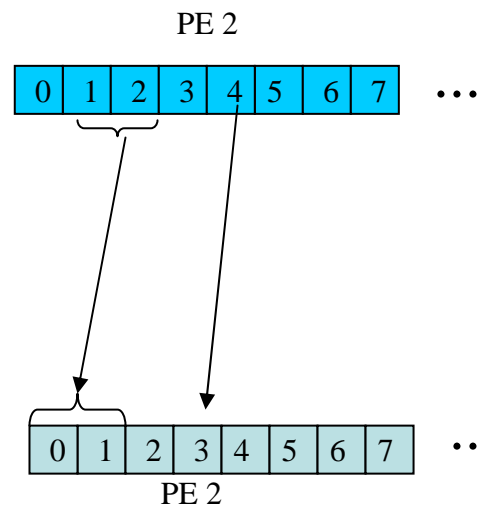
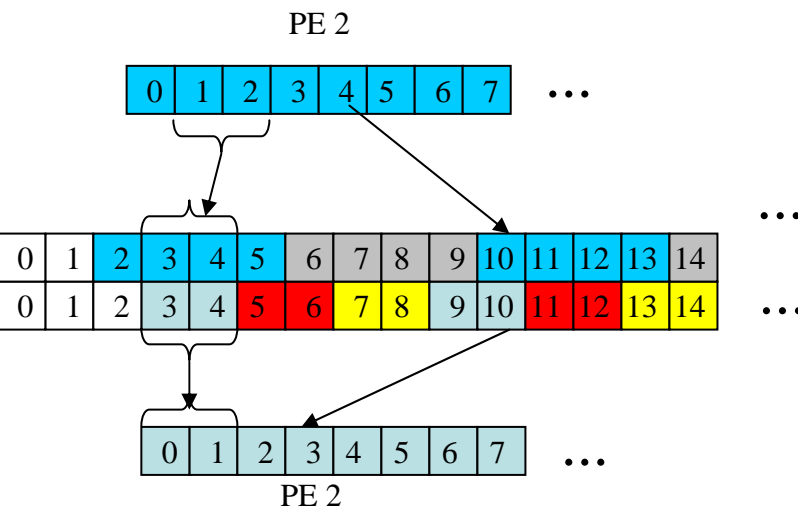
- Umverteilungsalgorithmus
 - bildet zusammenhängende Segmente in der einen auf zusammenhängende Segmente in der anderen Partition ab,
 - unter Ausnutzung der Regularität der Partitionierung
 - Problem: Wenn für den Zugriff Kommunikation notwendig ist, werden unter Umständen viele kleine Nachrichten durchs Netz geschickt.

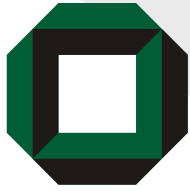




ClusterFile Abbildungsfunktionen (4)

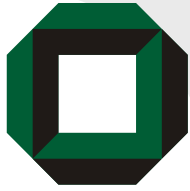
- Abhilfe: Verwendung einer zwischengeschalteten Abbildung in einen linearen Puffer
 - vermeidet das Versenden vieler kleiner Nachrichten
 - braucht Scatter beim Sender, Gather beim Empfänger
 - Aufwand: eine zusätzliche Kopieroperation.





ClusterFile Ausblick

- Collective-I/O
 - Zusammenfassen von E/A-Anfragen verschiedener Rechenknoten.
 - dadurch weniger Festplattenzugriffe nötig.
 - Zusammenfassen kann geschehen
 - beim Klienten, d.h. den Rechenknoten,
 - beim Server.
- Cooperative Caching
 - Beantworte Anfragen aus dem Puffer anderer Rechenknoten.
 - entlastet die E/A-Knoten.
 - Fernzugriff auf Speicher 2-3 Größenordnungen schneller als Festplattenzugriff.



Zusammenfassung

- Verteilte Dateisysteme
 - Der logische Dateibaum (Dateien, Verzeichnisse) wird über mehrere Rechner verteilt.
- Semantik für gemeinsamen Dateizugriff
 - UNIX-Semantik ist für verteilte Dateisysteme aufwändig.
 - Verteilte Dateisysteme bieten keinen paralleler Zugriff auf eine einzelne Datei.
- Parallele Dateisysteme
 - Es gibt Rechenknoten und E/A-Knoten mit Kommunikation untereinander.
 - Der parallele Zugriff auf eine einzelne Datei durch mehrere Rechenknoten ist hier erlaubt.
 - Die Blöcke einer gegebenen Datei können auf mehrere E/A-Knoten verteilt werden (die Datei kann als logische Datei gesehen werden).
 - Die Anpassung zwischen physikalischer und logischer Partitionierungen einer Datei ist wichtig für Leistung.