

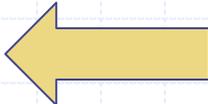
Betriebssysteme Vorlesung 9

File Management

Dauerhafte Datenhaltung in Dateien

Literatur: Stallings Kapitel 12

Stand der Vorlesung

- ◆ Einführung: Ein Betriebssystem - Was ist das ?
- ◆ Prozeßmanagement
 - Prozesse, Threads
 - Wechselseitiger Ausschluß, Deadlocks & Starvation
- ◆ Speicher Management
 - einfache Speicherverwaltung, virtueller Speicher, Paging
- ◆ Prozessor Scheduling
 - Time-sharing, Realtime, Scheduling bei 1 CPU, mehreren CPUs
- ◆ E/A Management, Festplattenscheduling
 - Schichtenmodell, SCAN Verfahren für Platten, RAID
- ◆ Datei Management  HEUTE !
- ◆ Netzwerke
- ◆ Sicherheit

Übersicht

- ◆ Grundbegriffe, Aufgaben im Dateimanagement
- ◆ Dateiararten
 - Pile, sequentielle Datei, index-sequentielle Datei, Datei mit Index, Datei mit Hashing
 - Directories
- ◆ Gemeinsamer Zugriff auf Dateien
- ◆ Zusammenfassen von Records in Blöcken
- ◆ Verwaltung von Dateien im Sekundärspeicher
- ◆ Beispiele
 - Dateimanagement in UNIX
 - Dateimanagement in Windows 2000

Nutzen

- ◆ Sie lernen Grundkonzepte zur dauerhaften Speicherung von Daten kennen.
- ◆ Sie können Grenzen und Aufwand in der Dateiverwaltung realer Betriebssysteme nachvollziehen.
- ◆ Sie erkennen Querbezüge zu Problemen aus der Speicherverwaltung.
 - Auswahl von Speicherbereichen zur Belegung mit Dateien
 - Verfahren zur Verringerung des Suchaufwandes in Daten

Dateimanagement

◆ Dateimanagement: Systemsoftware,

- die Anwendern und Prozessen die Nutzung von Dateien ermöglicht, die dauerhaft im Sekundärspeicher gespeichert werden.
- die nicht unmittelbar zum Betriebssystem gehört, dessen Funktionalität jedoch zum Teil durch das Betriebssystem erbracht wird.

◆ Teilaufgaben

- Zugriff auf Dateien: öffnen, lesen, schreiben, schliessen
- Directorymanagement: Verwaltung von Dateien in hierarchischen Strukturen, erzeugen, umbenennen, verschieben, löschen von Dateien und Directories
- Zugriffskontrolle, Rechtevergabe und Überwachung

◆ Intern:

- Datenstrukturen zur Unterstützung der Dateifunktionen
- Verwaltung des belegten und freien Festplattenplatzes

◆ Anforderungen: Effizienz (Platz & Zeit), Zuverlässigkeit

Grundbegriffe

◆ Field, Feld

- Feld nimmt einzelnen Wert auf, z.B. Zahl, Datum, Zeichenkette
- Feld durch Länge und Datentyp charakterisiert
 - ◆ Feld kann feste Länge haben
 - ◆ Feld kann variable Länge haben => Teilfelder mit Einträgen und Länge oder spezielles Ende-Zeichen z.B. `'\0'`

◆ Record

- Zusammenstellung von Feldern, die für ein Anwendungsprogramm eine Einheit bilden,
- z.B. Anschrift: Name, Str, PLZ, Ortsbezeichnung
- Länge kann durch Felder mit variabler Länge oder durch variable Anzahl an Feldern variabel sein

Grundbegriffe

◆ Field, Feld

◆ Record

◆ File, Datei

- Zusammenstellung gleichartiger Records
- durch Anwendungen als eine Einheit betrachtet
- anhand des (eindeutigen) Namens referenziert
- Dateien können erzeugt, gelöscht werden, haben Zugriffsrechte

◆ Database, Datenbank

- Zusammenstellung verwandter Daten
- Beziehungen der Daten sind explizit bekannt
- Datenbank wird von mehreren Anwendungen genutzt
- eigenes Datenbankmanagementsystem
- legt Daten in Dateien ab

Elementare Operationen für Records

Lesen

- Retrieve-All: sequentieller Zugriff auf alle Records
- Retrieve-One: Zugriff auf einen speziellen Record
- Retrieve-Next / - Previous: liefert Vorgänger, Nachfolger bzgl einer Reihenfolge
- Retrieve-Few: Zugriff auf Auswahl von Records mit bestimmten Eigenschaften, kann Suche erfordern

Einfügen

- Insert-One: Einfügen, ggfs an spezieller Position

Löschen

- Delete-One: Löschen eines einzelnen Records

Ändern

- Update-One: kann Auswirkung bzgl Reihenfolgen haben, bei variabler Länge nicht trivial

je nachdem welche Operationen durch eine Anwendung im wesentlichen genutzt wird, wird der Dateityp gewählt.

Anforderungen

- Basisfunktionalität, elementare Operationen auf Dateien
- Validität der Daten in Dateien möglichst weitgehend sicherstellen
- Performance
 - ◆ aus Systemsicht: hoher Gesamtdurchsatz
 - ◆ aus Anwendersicht: kurze Antwortzeiten
- Unterstützung für viele Speichermedien
- Zuverlässigkeit, minimales Risiko für Datenverluste
- Standardisierte Menge von E/A Routinen
- Support für Mehrbenutzerbetrieb

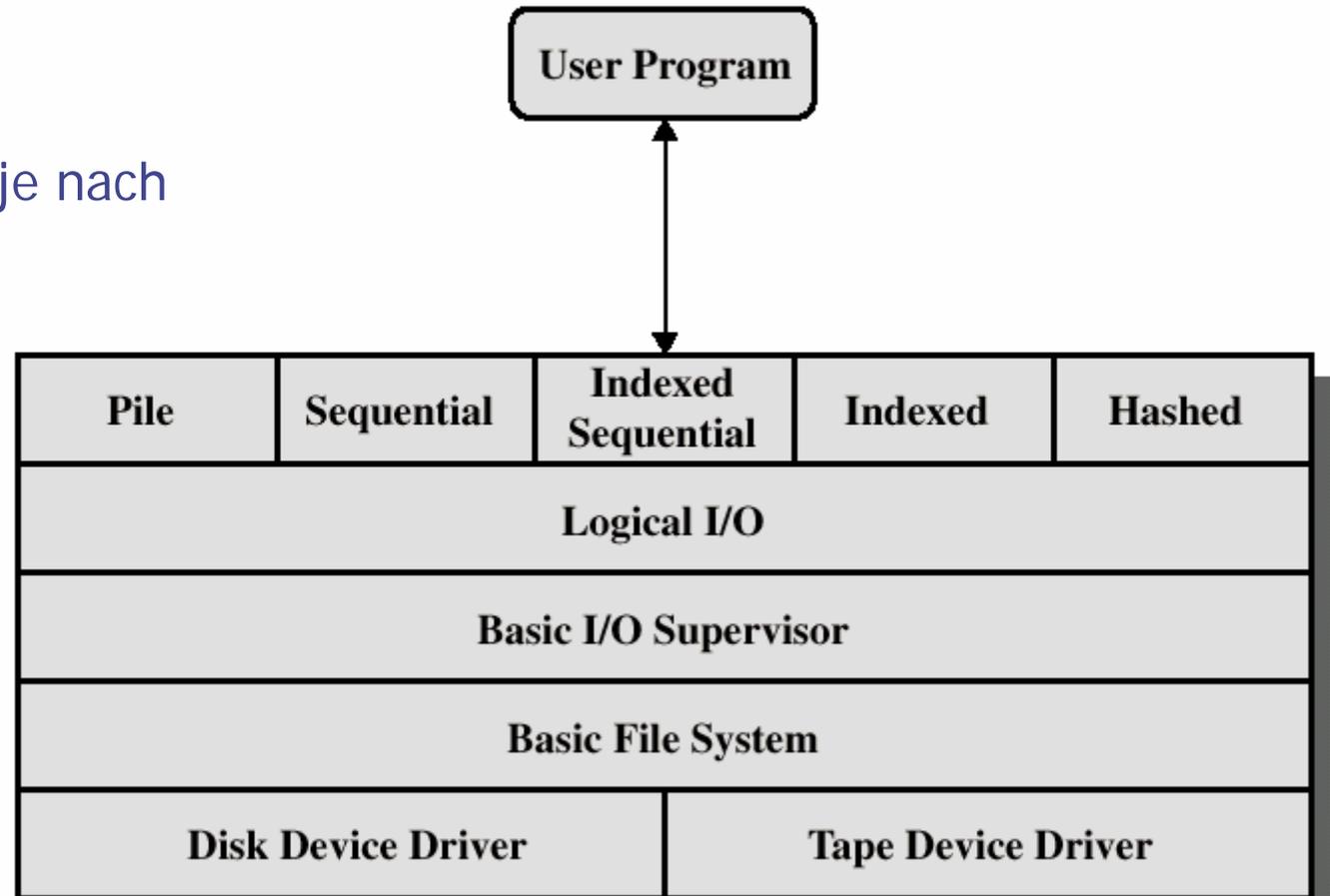
Minimum für interaktives Mehrbenutzersystem, je User:

- Erzeugen, Löschen, Lesen, Ändern von Dateien
- kontrollierter Zugriff auf fremde Dateien
- Kontrolle über Zugriffsbeschränkungen bei eigenen Dateien
- Restrukturieren von Dateien, (Directories)
- Verschieben von Daten zwischen Dateien
- Backup und Recovery im Falle der Zerstörung von Dateien
- Zugriff auf Dateien mittels symbolischer Namen

Softwarearchitektur eines Dateisystems

Zugriffsmethoden je nach Dateitypen

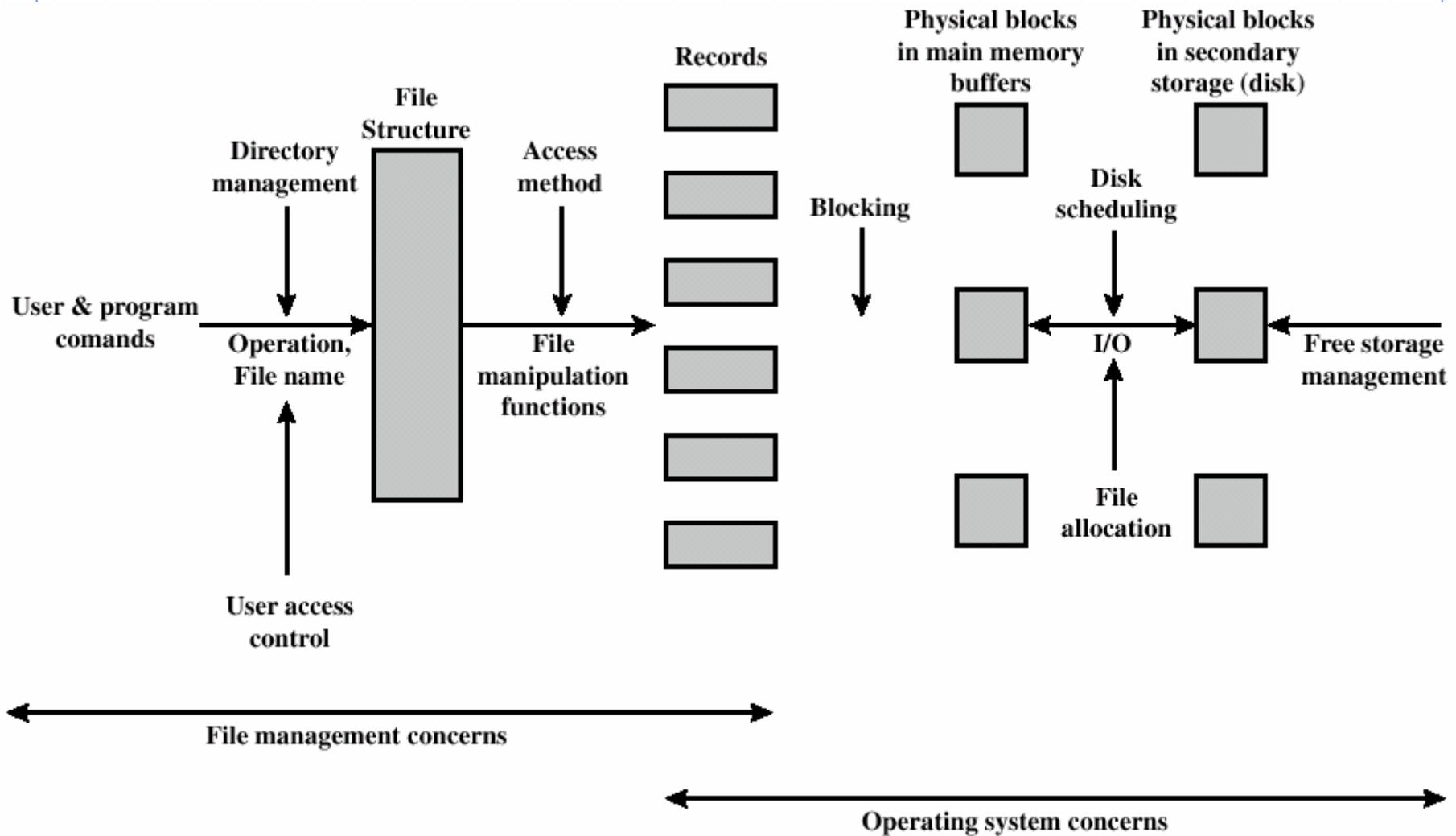
Logische E/A:
ermöglicht
Zugriff auf
Records,
verwaltet
Dateien als
Menge von
Records



E/A Supervisor: regelt E/A Zugriff, Status von Dateien, Geräteauswahl und Scheduling

Basic File System, physikalische E/A Ebene: behandelt Daten auf Blockebene, z.B. Platzierung von Blöcken im Sekundärspeicher, Puffern von Blöcken beim Transfer, Struktur von Dateien hier nicht erkennbar

Dateimangement, einige Aspekte



Dateiorganisation und Zugriffsmethoden

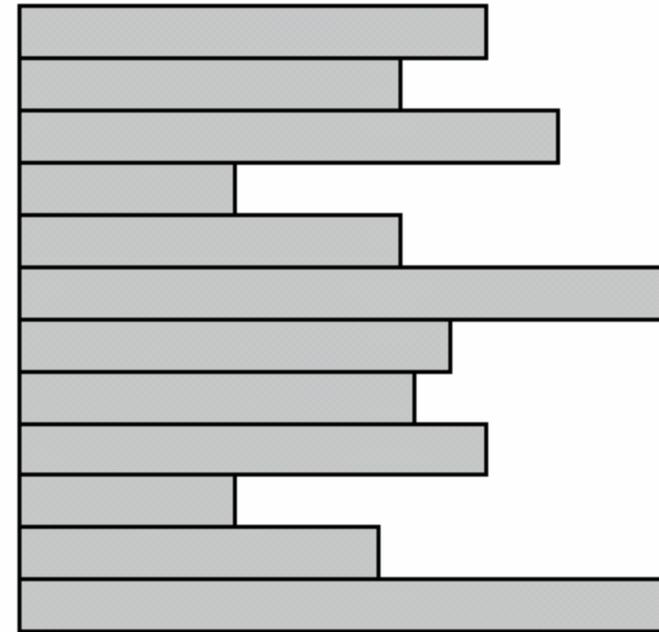
- ◆ Eine Datenstruktur erzeugt die logische Struktur einer Datei aus Records und Operationen für Records, die physikalische Struktur besteht jedoch aus Blöcken, die im Sekundärspeicher verwaltet werden müssen.
- ◆ Auswahlkriterien für Datenstrukturen
 - Zugriff erfolgt schnell, Änderungen sind einfach
 - Speicherplatz ökonomisch genutzt
 - Wartung einfach, Zuverlässigkeit
- ◆ es existieren viele Varianten von Dateistrukturen
 - einfachste Variante: Byte Sequenz, unstrukturierte Folge von Bytes, erfordert interne Struktur, die von Anwendungsprogrammen selbstverantwortlich organisiert wird, z.B. Executables, Archive
 - Pile
 - Sequentielle Datei
 - Index-sequentielle Datei
 - Datei mit Index
 - Datei mit direktem Zugriff, Hashing

Pile

einfache Datenstruktur,
neue Records werden einfach angehängt,
Anwendung:
einfache Akkumulation von Daten
Records können

- mehrere Felder haben
- von variabler Länge sein
- je Feld selbsterklärend, d.h. Bezeichner, Länge muss erkennbar sein (explizit, mittels Endezeichen oder typbedingt)

Zugriff auf speziellen Record erfordert erschöpfende Suche
Platzeffizient, falls Daten in Größe und Struktur variieren.



Variable-length records
Variable set of fields
Chronological order

Wegen des Aufwands für eine erschöpfende Suche ist dieser Dateityp nur bei speziellen Anwendungen sinnvoll.

Index-sequentielle Datei

gleiche Charakteristika wie sequentielle Datei,
2 zusätzliche Features:

- Index unterstützt Suche nach einzelnen Records
- Overflow Datei analog zum Log File, nur sind Einträge mittels Zeigern bereits in die Datei integriert.

Einfachste Variante: nur eine Ebene Indexebene,

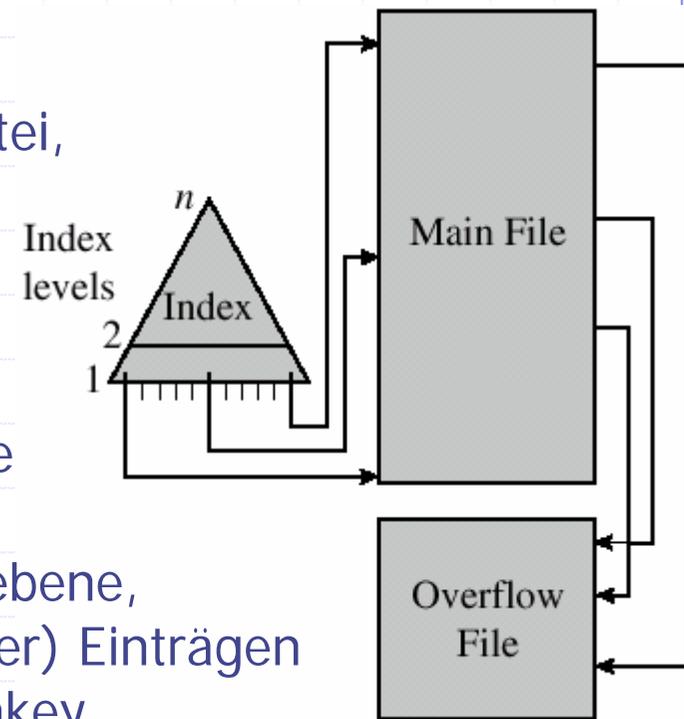
- Index ist sequentielle Datei mit (Key, Zeiger) Einträgen
- Suche erfolgt nach größtem Key \leq Suchkey
- Anzahl Einträge im Indexfile \ll Anzahl Einträge in der Datei selbst.

z.B. Datei mit $N=1.000.000$ Records, Index mit $n=1.000$ Einträgen
sequentielle Suche ohne Index, im Mittel $N/2 = 500.000$ Einträge,
sequentielle Suche mit Index, im Mittel $n/2 + N/(1.000 \cdot 2) = 1.000$

Fortgeschrittene Variante: mehrstufige Indexdateien

Einfügeoperation: in der Datei beginnt mit einem Zeiger eine lineare Liste, die in der Overflow Datei abgelegt wird.

Overflow Datei wird gelegentlich mit der eigentlichen Datei verschmolzen.



Indexierte Datei

Sequentielle und index-sequentielle Datei sind nur nach 1 Schlüssel sortiert.

Suche anhand anderer Felder sehr aufwendig.

Abhilfe:

Indexierte Datei

Records werden ausschließlich über indexierte Felder erreicht, daher keine Restriktion bzgl der Reihenfolge der Speicherung von Records.

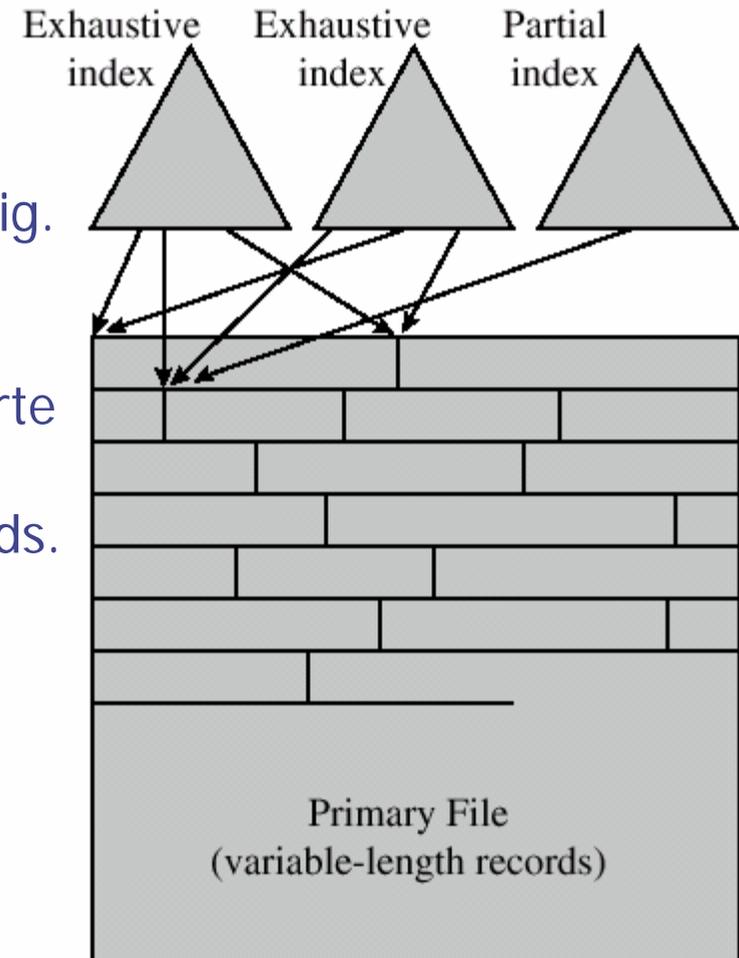
2 Arten von Index:

- vollständig
- partiell, d.h. einige Records haben keine passenden Werte für das Feld

Indexierte Datei ist sinnvoll

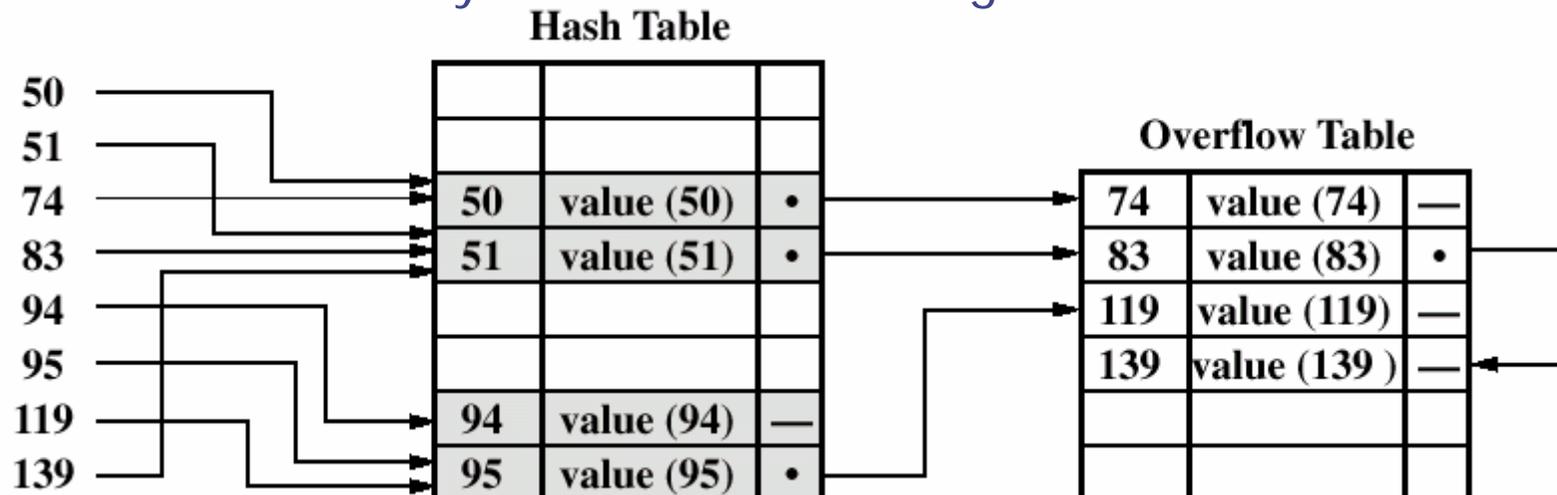
- wenn Suchzugriffe schnell sein müssen
- selten alle Records bearbeitet werden müssen

Beispiele: Reservierungssysteme, Verwaltung von Lagerbeständen



Dateien mit direktem Zugriff, mit Hashing

- ◆ bisheriges Problem ist eine schnelle Suche eines einzelnen Eintrags
- ◆ warum nicht mit Hashing ?
- ◆ sinnvoll, falls Zugriffszeiten auf Records im Speichermedium überall gleich schnell sind, z.B. bei Festplatten, (im Unterschied zu Bandlaufwerken) und keine sequentielle Bearbeitung auftritt
- ◆ Grundidee: Hashfunktion bildet Key auf Position des Records in der Datei ab, für Kollisionen wird eine lineare Liste in der Hashtabelle begonnen, deren Elemente dann in der Overflow Tabelle abgelegt werden.
- ◆ z.B. Hashfunktion h mit $h(51)=h(83)=h(139)$, d.h. in der Abbildung wird Record mit $\text{key}=139$ über die Einträge 51 und 83 erreicht.



Bewertung

File Method	Space		Update		Retrieval		
	Attributes		Record Size		Single record	Subset	Exhaustive
	Variable	Fixed	Equal	Greater			
Pile	A	B	A	E	E	D	B
Sequential	F	A	D	F	F	D	A
Indexed sequential	F	B	B	D	B	D	B
Indexed	B	C	C	C	A	B	D
Hashed	F	B	B	F	B	F	E

- A = Excellent, well suited to this purpose $\approx O(r)$
 B = Good $\approx O(o \times r)$
 C = Adequate $\approx O(r \log n)$
 D = Requires some extra effort $\approx O(n)$
 E = Possible with extreme effort $\approx O(r \times n)$
 F = Not reasonable for this purpose $\approx O(n^{>1})$

where

- r = size of the result
 o = number of records that overflow
 n = number of records in file

Verwaltung von Dateien mit Directories

- ◆ Directories sind selbst wiederum Dateien (natürlich)
- ◆ je Datei sind folgende Inhalte wichtig
 - Basisinformation
 - ◆ **Name**: Bezeichner, innerhalb des Directories eindeutig (8 Zeichen vs beliebige Zeichenketten, Unterscheidung Groß/Kleinschreibung, Unicode)
 - ◆ **Dateityp**: z.B. text, binär, ... (häufig auch als Suffix, z.B. .txt, .doc, .o)
 - ◆ **Dateiorganisation**, falls die unterschiedliche Varianten unterstützt
 - Adreßinformation
 - ◆ **Volume**: auf welchem Gerät ist die Datei gespeichert
 - ◆ **Anfangsadresse**: physikalische Adresse, z.B. Cylinder,Track,Sektor
 - ◆ **aktuelle und maximale Größe**: in Bytes, Worten, Blocks
 - Information für die Zugriffskontrolle
 - ◆ **Eigentümer, Zugriffsrechte, erlaubte Aktionen**
 - Information zur Nutzung
 - ◆ **Datum** der Erzeugung, des letzten Lese/Schreibzugriffs, letztes Backup
 - ◆ **Identität** des Erzeugers, Lesers, Schreibers
 - ◆ **aktuelle Nutzung**, z.B. durch wen geöffnet, im Hauptspeicher verändert, aber noch nicht gesichert etc.

Operationen auf Directories

- ◆ natürlich: erzeugen, lesen, ändern, löschen, d.h.
- ◆ Datei suchen
 - zur Referenzierung einer Datei muss der entsprechende Eintrag im Directory ermittelt werden
- ◆ Datei erzeugen
 - impliziert neuen Eintrag im Directory
- ◆ Datei löschen
 - Directoryeinträge für gelöschte Dateien müssen entfernt werden
- ◆ Directoryinhalt auflisten
 - typische Anforderung durch Anwender, alle Dateien oder alle Dateien eines Typs auflisten
- ◆ Directory aktualisieren
 - bei Änderung von Attributen einer Datei muß Directory aktualisiert werden

Directories werden ausgehend von einem Master- oder Wurzeldirectory hierarchisch organisiert, d.h. ein Directory hat nicht nur Dateien sondern auch andere Directories als Einträge.

Directories bilden baumartige Strukturen aus

eindeutige Namensgebung:

Der Pfadname von der Wurzel (Directory) über Zwischenebenen (Directories) zum Blatt (Datei) liefert einen eindeutigen Bezeichner.

z.B. UserB/Word/UnitA/ABC

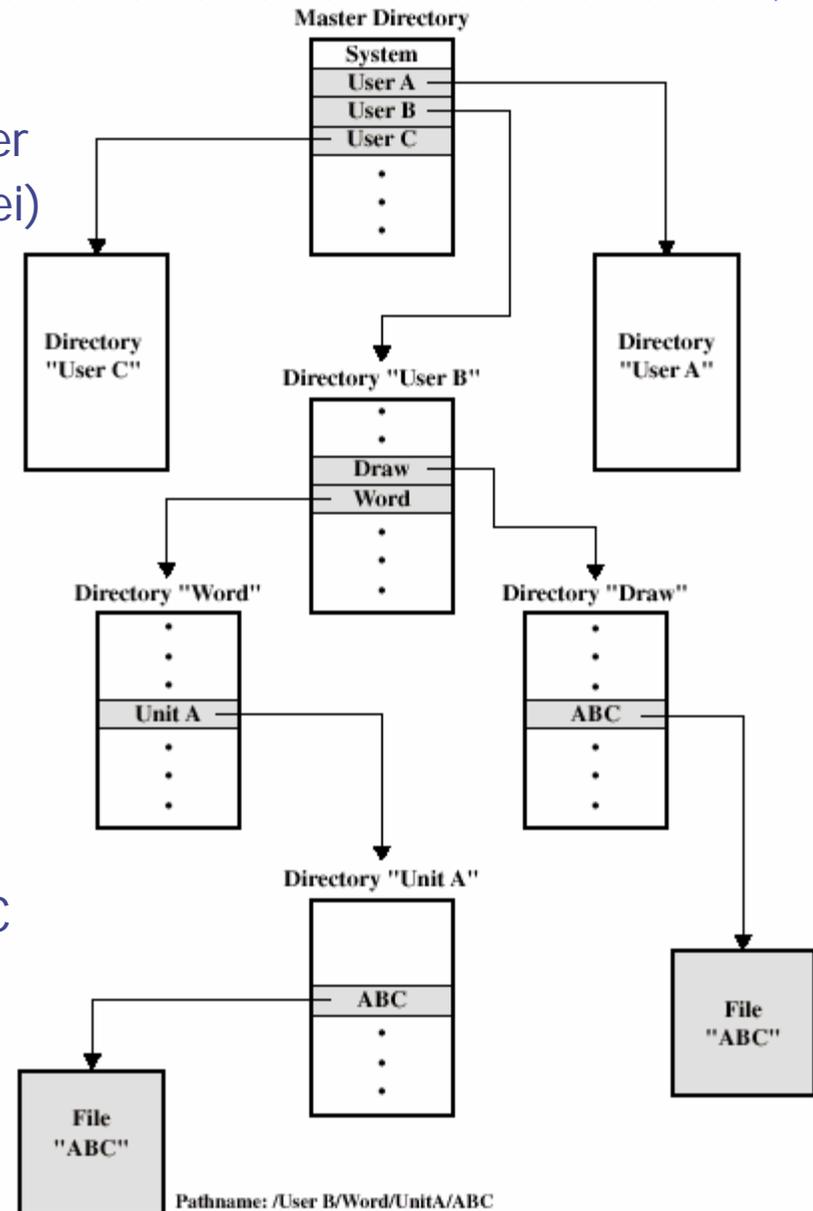
Directory selbst kann dann mit einer sequentiellen Datei (oder anderer Dateart) realisiert sein.

Nutzungskomfort:

Navigation innerhalb des Baums,
Working Directory = aktuelles Directory,
relative Pfadnamen ausgehend von der aktuellen Position, z.B. in Unit A

../../Draw/ABC verweist auf /UserB/Draw/ABC

Jeder Nutzer hat „eigenen“ Dateibaum,
Rechtevergabe für ganze Teilbäume
und für die Navigation im Directory leicht realisierbar.



Geteilte Dateien: Zugriffsrechte

Dateien können von mehreren Anwendern genutzt werden.

Welche Zugriffsrechte werden unterschieden?

- keine Rechte: der Anwender erfährt nichts von der Existenz einer Datei
-> keine Leserechte für das Directory mit der Datei
- Wissen um die Existenz: Anwender erfährt die Existenz und den Eigentümer, um ggfs um mehr Rechte fragen zu können
- Ausführungsrechte: Anwender kann die Datei als Programm laden und ausführen lassen, jedoch nicht kopieren. Typischerweise für proprietäre, lizenzierte Software.
- Leserechte: Datei kann für beliebige Zwecke gelesen werden, aber nicht modifiziert werden. Impliziert für die meisten Systeme auch die Möglichkeit des Kopierens.
- Ergänzung: Anwender darf Daten an einem Ende anfügen
- Schreibrechte: Anwender darf Daten ändern, löschen, ergänzen
- Zugriffsrechte ändern: liegt typischerweise beim Eigentümer
- Löschen: Zerstören der Datei, Löschen des Eintrags im Directory

Rechte bilden Hierarchie aus, z.B. Schreibrechte implizieren Leserechte, muß in Systemen aber nicht gelten, z.B. Schreibrechte für ein Directory hebeln Begrenzung auf Leserechte bei einer Datei im Directory aus.

Geteilte Dateien: Rechte und Simultaner Zugriff

- ◆ Unterscheidung der Anwender bei der Rechtevergabe
 - spezielle Anwender gemäß ID
 - Anwendergruppen, für das System jeweils die Mitgliedschaft nachprüfen kann
 - Alle Anwender, d.h. jeder Anwender, der Zugang zum Rechensystem hat
- ◆ Simultane Zugriffe von mehreren Anwendern auf 1 Datei impliziert wie bei Prozessen Sequentialisierung der Zugriffe:
 - Leser/Schreiberproblem: Nutzung im wechselseitigen Ausschluß mit entsprechender Zugangsregelung
 - Deadlockproblem
 - Starvationproblem
 - Granularität für Locking: ganze Dateien oder einzelne Records in Dateien
- ◆ Geteilte Dateien, falls an mehreren Stellen im Directory präsent
 - Directory: Baum -> gerichteter azyklischer Graph
 - „Querverbindung“: Link, (in Unix: Hard-Link, Soft-Link)
 - Hard-Link: Daten bzgl der Plattenblöcke sollte nicht im Directory sondern in eigener DS liegen, z.B. Directory enthält Verweise auf Inodes, Inodes enthalten Blöcke

Layout eines Dateisystems

- ◆ Dateisystem wird in Partition einer Festplatte gespeichert
- ◆ Festplatte:
 - Master Boot Record: Sektor 0, mit Partitionstabelle, Kennzeichnung aktiver Partition (mit BS)
 - Partition 1:
 - ◆ Bootblock, nur bei BS Partition gefüllt,
 - ◆ Superblock, Schlüsselparameter des Dateisystems, Magic Number zur Identifizierung, Anzahl Blöcke,
 - ◆ Freispeicherverwaltung, mit Bitmap oder Liste von Zeigern
 - ◆ I-Nodes, FAT o. ä. Strukturen zur Dateiverwaltung
 - ◆ Wurzelverzeichnis
 - ◆ Dateien und Verzeichnisse
 - Partition 2
 - ...

Records und Blöcke

Bei Byte Sequenzen ist Blockbildung trivial.

Records: logische Einheit beim Dateizugriff

Blöcke: physikalische Einheit der E/A mit Sekundärspeicher

- typischerweise feste Blockgröße
 - ◆ wg sequentieller Zugriffe eher große Blöcke
 - ◆ wg E/A Puffergrößen und Pufferverwaltung nicht zu groß
- 3 Arten Blöcke aus Records herzustellen
 - ◆ feste Blockbildung: bei Records fester Länge wird eine ganzzahlige Anzahl Records in einem Block untergebracht, interne Fragmentierung, für sequentielle Dateien und Records fester Länge typischerweise genutzt
 - ◆ Blockbildung bei variabler Länge und Überlappung: Records variabler Länge werden ohne Fragmentierung in Blöcken fester Länge angeordnet. Falls der Rest eines Records in einem anderen Block untergebracht wird, wird die Überlappung durch Zeiger hergestellt. Schwierig zu realisieren.
 - ◆ Blockbildung bei variabler Länge aber ohne Überlappung: mit Records variabler Länge, wobei interne Fragmentierung auftritt, falls im verbleibenden Platz eines Blocks der Folgerecord nicht komplett untergebracht werden kann. Limitiert die maximale Länge eines Records auf die Blocklänge.

Verwaltung von Dateien im Sekundärspeicher (Festplatte)

◆ Aufgabe: Zuordnung von Blöcken zu Dateien

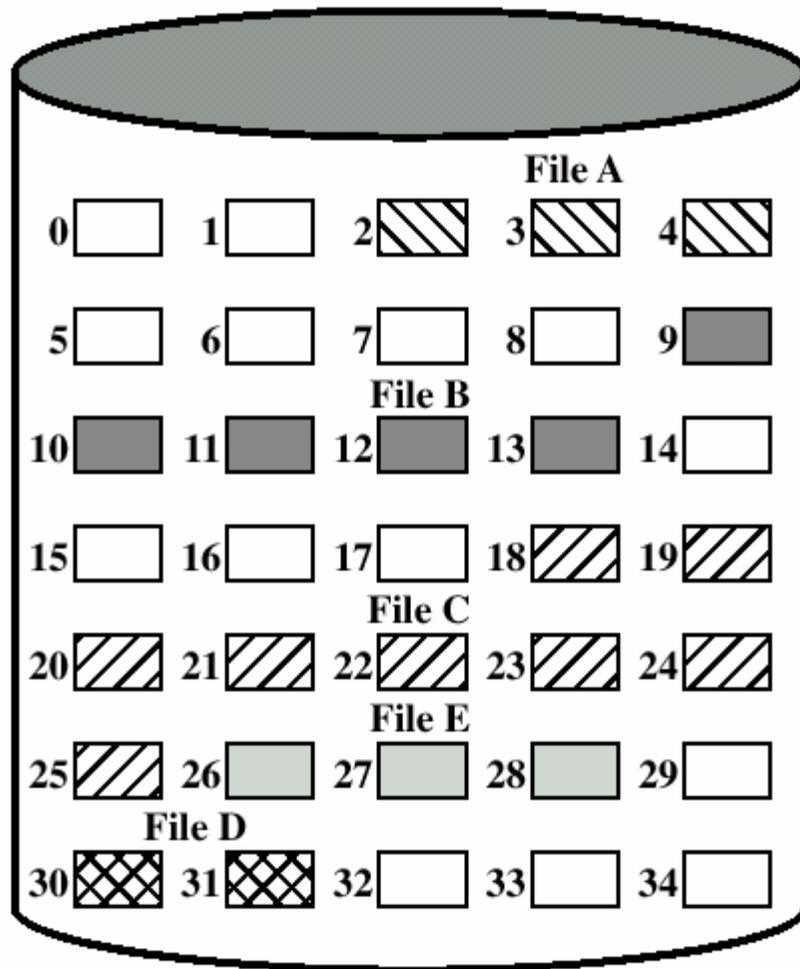
- Anmerkung: hat Ähnlichkeiten zur Hauptspeicherverwaltung
- Aufbau einer Verwaltungsstruktur: File Allocation Table (FAT)
- Preallocation des maximalen Platzes oder dynamische Belegung von Blöcken durch eine Datei nur bei Bedarf
- sinnvolle Größe von Teilen einer Datei: 1 Block <-> ganze Datei
 - ◆ 1. Variante: große kontinuierliche Teile variabler Länge, keine interne Fragmentierung, kleine FAT, gute Performance bei sequentiellen Zugriffen, externe Fragmentierung wie bei Segmentierungsverfahren -> First fit, Best fit, Next fit Selektionsverfahren, kein klarer Kandidat
 - ◆ 2. Variante: kleine Teile fester Länge, größere Flexibilität, größere oder komplexere FAT, keine kontinuierliche Belegung von Speicherzellen

◆ bekannte Verfahren

- Kontinuierliche Belegung,
- Verkettete Belegung,
- Belegung mit Index

z.B. bei CD-ROMs naheliegend

Kontinuierliche Belegung von Blöcken



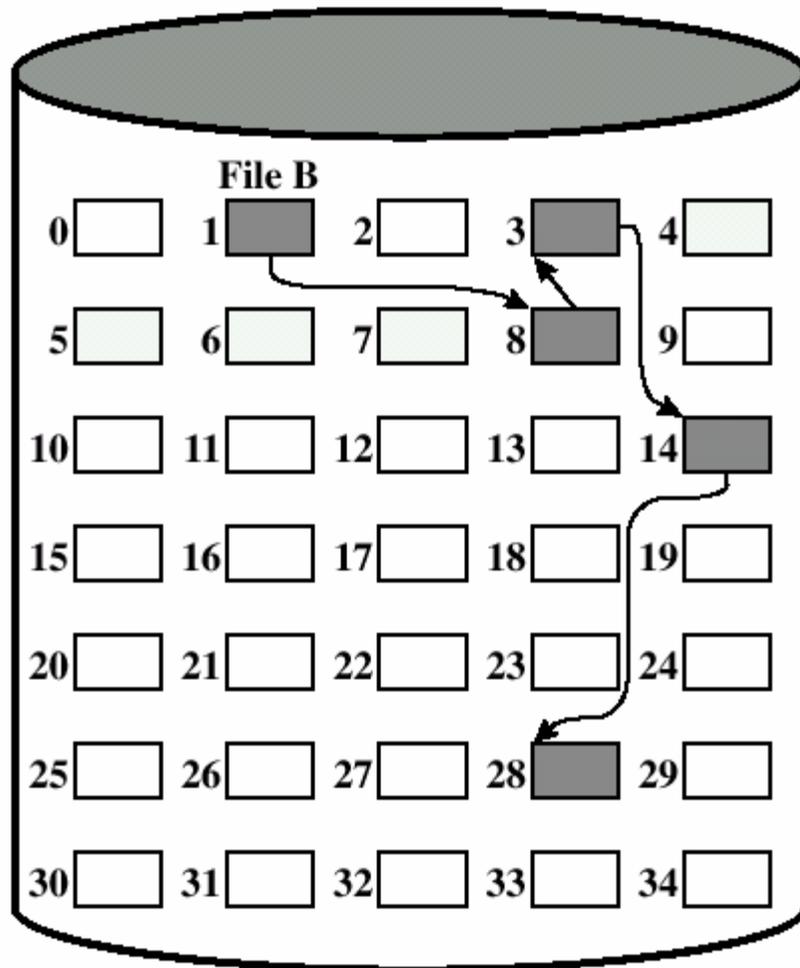
File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Preallocation Strategie mit Teilen variabler Länge

- gute Performance wg sequentieller Zugriffe auf die Festplatte
- kleine FAT, nur Anfang & Länge
- einfache Lokalisierung von Blöcken
- externe Fragmentierung -> defragmentieren durch Verschieben von Blöcken (gelegentlich)
- Problem: Bestimmung der Größe

Verkettete Belegung von Blöcken



Nutzdaten je Block ist wg des Zeigers keine Potenz von 2, ungünstig!

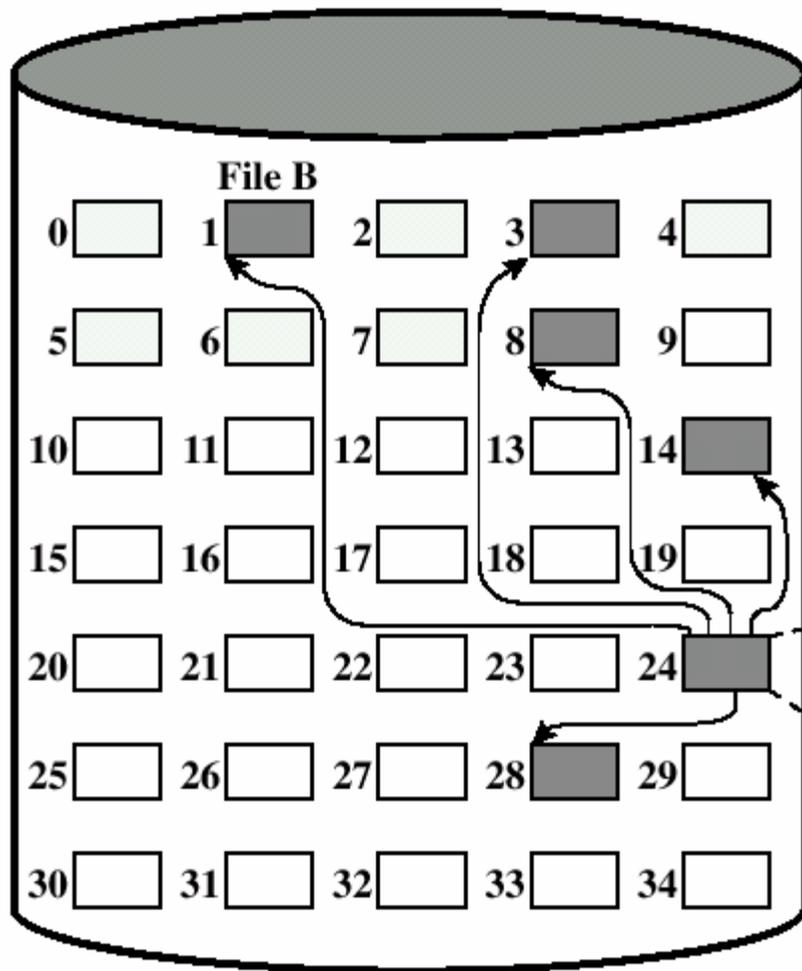
File Allocation Table

File Name	Start Block	Length
...
File B	1	5
...

Strategie mit dynamischer Belegung von Blöcken, je Block Zeiger auf den Nachfolgeblock

- kleine FAT
- einfache Auswahl & Zuordnung freier Blöcke
- keine externe Fragmentierung
- gut bei sequentiellen Dateien mit sequentieller Bearbeitung
- Probleme: keine Lokalität, z.B. beim Laden mehrerer Blöcke, beim Suchen 1 Blocks

Belegung von Blöcken mit Index (übliches Schema)



File Allocation Table

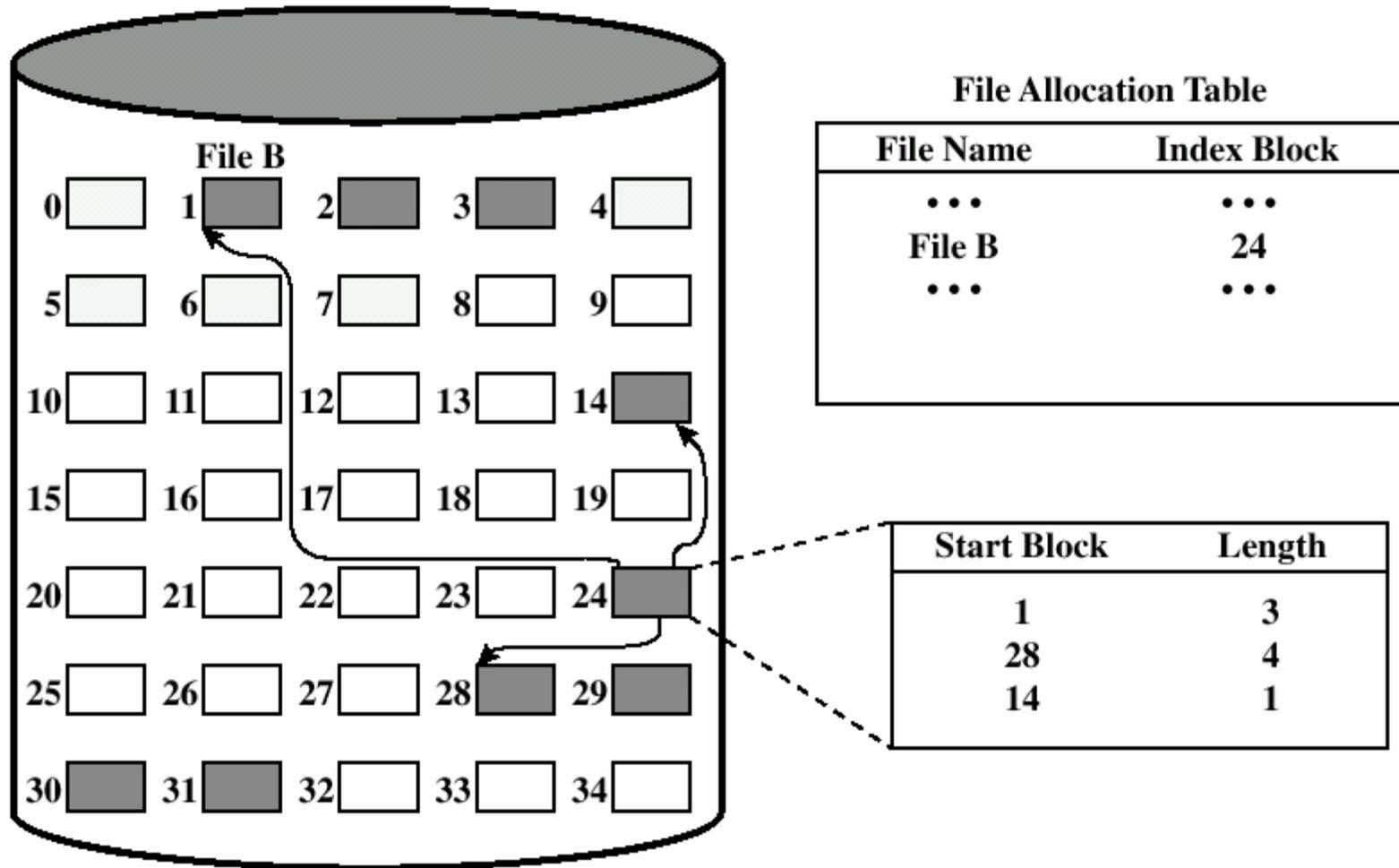
File Name	Index Block
...	...
File B	24
...	...

Eine Ebene der Indexierung,
1 Block enthält Verweise auf
Sequenz der Blöcke einer
Datei

- für feste und variable Teile
(im Bsp. hier fest, 1 Block)
- keine externe Fragmentierung
- variable Länge erhöht
Lokalität

Konsolidierung erhöht jeweils Lokalität, bei variabler Länge verringert sich
der Platzbedarf für den Index

Belegung von Blöcken mit Index und variablen Teilen



Platzbedarf des Indexes je Eintrag ist größer, lohnt erst bei Längen > 2 . Konsolidierung bedeutet Verschieben von Blöcken, so daß längere Teile entstehen. Kann in Phasen geringer Plattenauslastung durchgeführt werden.

Übersicht zu Belegungsverfahren

	Contiguous	Chained	Indexed	
Pre-Allocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium

Teile variabler Länge führen zu geringem Platzbedarf für FAT.
Lokalität ist günstig für eine sequentielle Bearbeitung
ganzer Dateien, weil Plattenzugriffszeiten dann besser sind.

Freispeicherverwaltung

◆ Disk Allocation Table zur Verwaltung freier Blöcke

◆ Techniken

- Bitvektoren: Vektor mit 1 Bit je Block signalisiert, ob ein Block belegt ist oder nicht, Länge: Plattengröße / 8 x Blockgröße, z.B. 2^{34} Bytes / $2^3 \cdot 2^9 = 2^{22}$, also 4 MB im Hauptspeicher, Suche nach größeren freien Bereichen ggfs teuer, daher weitere Hilfstabellen
- Verkettung freier Teile: da der Speicherplatz frei ist, kann er für den Aufbau einer linearen Liste genutzt werden, freie Blöcke verweisen auf Nachfolger, Allokieren und Freigeben kann teuer sein.
- Indexierung: betrachtet freien Speicher als Datei und verwendet Indextabelle zur Darstellung mit variabler Länge von Teilen, Indexierung unterstützt alle Belegungsverfahren recht gut
- Liste freier Blöcke: Blöcke werden durchnummeriert, Liste freier Blöcke wird auf der Platte abgelegt. Platzbedarf hoch aber relativ gering (< 1%), nur relevanter Teil liegt im Hauptspeicher, bei Stack nur der obere Teil, bei Queue Anfang- und Endteil. Hilfreich bei Teilen fester Länge.

Zuverlässigkeit

Durch einen Systemabsturz gehen Daten im Hauptspeicher verloren, nur Daten auf der Festplatte liegen noch vor.

Zuverlässigkeit

- die Datenstrukturen für die Dateiverwaltung sind in sich stimmig
- es gehen keine Daten von Anwendern verloren

Hier nur 1. Aspekt betrachtet, Problem entsteht durch Nutzung des Hauptspeichers als „Cache“ und nur gelegentliche Aktualisierung der Platteninhalte.

Variante 1: FAT und Disk Allocation Table werden jeweils sofort aktualisiert -> reduziert Performance zu stark

Variante 2: Batch Allocation, d.h. eine Menge von Seiten wird für die Belegung genutzt und auf der Platte als „in Gebrauch“ gekennzeichnet. Ist der Batch verbraucht, wird der Platteninhalt aktualisiert. Bei einem Absturz muß lediglich der letzte Batch kontrolliert werden, z.B. bei einem Journaling System.

Beispiel: Dateimanagement in UNIX

Alle Dateien werden als Strom von Bytes betrachtet

4 Typen werden unterschieden

- gewöhnliche Dateien
- Directories: enthalten eine Liste der Dateien / Directories und Zeiger of zugehörige Inodes (Information nodes). Besonderheit sind Schreib- und Leserechte. Schreiben darf nur das Dateisystem.
- Spezialdateien: für externe E/A Geräte wie Terminals, Drucker.
- Benannte Dateien: Pipes zur Interprozeßkommunikation

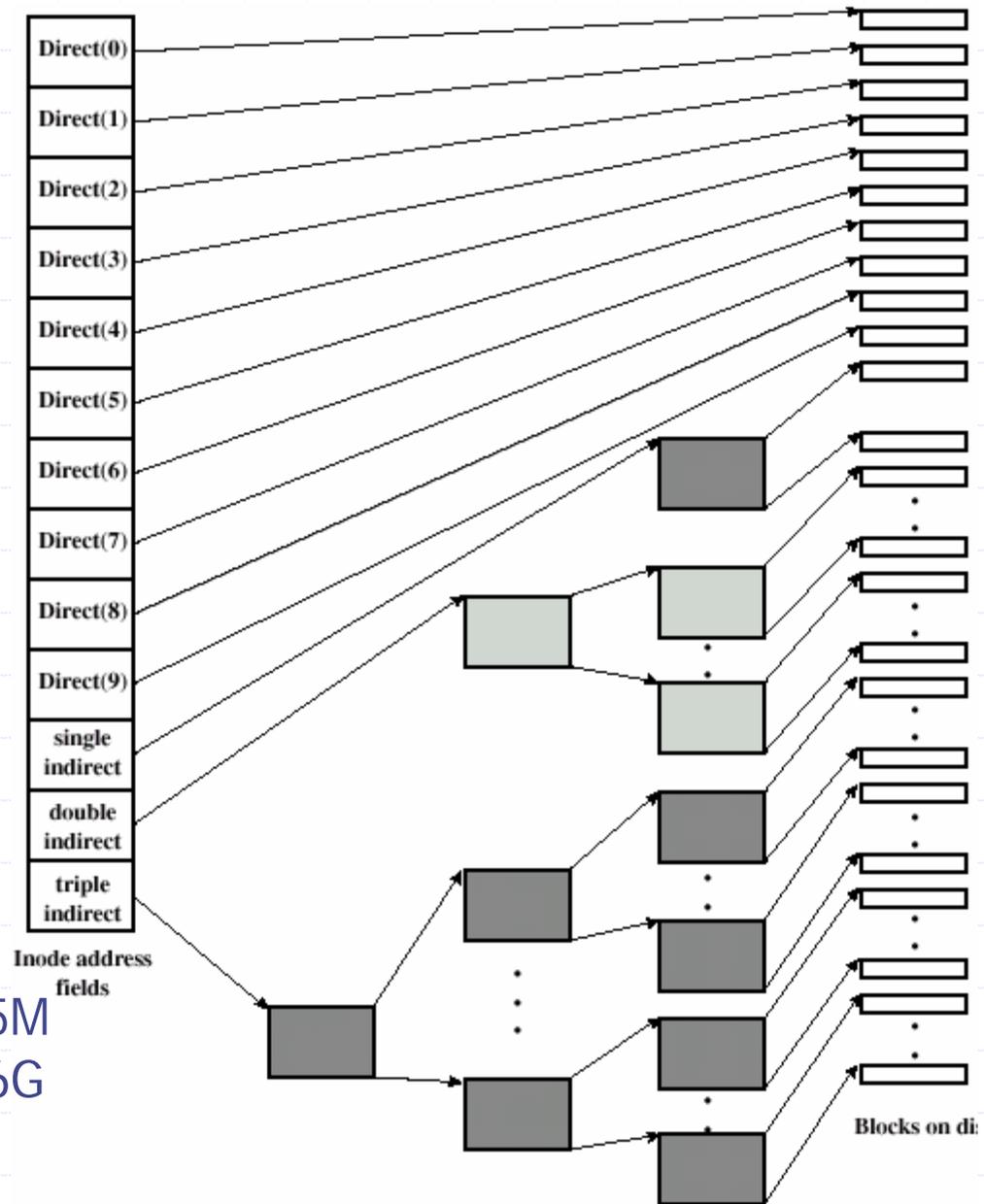
Information Node (Inode) je Datei enthält

- File Mode: Zugriffs- und Ausführungsrechte (Owner, Group, World)
- Anzahl Verweise (Links) aus Directories auf die Datei
- Gruppen-Id und User-Id des Eigentümers
- Anzahl Bytes in der Datei
- Daten bzgl letzter Änderung, letztem Zugriff, letzter Inode Änderung
- 39 Bytes Adressinformation, 13 Adressen a 3 Byte
 - ◆ 10 Adressen für die ersten 10 Blöcke (direkte Adressierung)
 - ◆ 11. Adresse zeigt auf Block mit Adressen (einfache indirekte Adressierung)
 - ◆ 12. Adresse zeigt auf doppelt indirekten Block
 - ◆ 13. Adresse zeigt auf dreifach indirekten Block

Block Adressierung mit Inodes in UNIX

Vorteile des Schemas

- Inodes haben feste Größe, relativ klein, kann daher im Hauptspeicher relativ lange verbleiben
- kleine Dateien kommen mit direkter Adressierung oder mit einfach indirekter Adressierung aus
- das theoretische Maximum der Dateigröße reicht aus
 - z.B. 1 Kbyte Block,
 - => 256 Adressen je Block
 - => Adressen 1-10 10 K
 - 11. Adresse 256 K
 - 12. Adresse 256² K, 65M
 - 13. Adresse 256³ K, 16G



Beispiel: Dateimanagement in Windows 2000, NTFS

◆ unterstützt mehrere Systeme

- FAT-16, FAT-32 aus Windows 95, MS-DOS, OS2
- aber auch NTFS für Workstations und Server

◆ NTFS Features

- Recoverability: nach Systemabstürzen und Festplattenausfällen kann NTFS Diskvolumen rekonstruieren. Hilfsmittel: Transaktionskonzept, Logfiles, redundante Speicherung von kritischen Systemdateien
- Sicherheit, mittels W2K Object Model gibt es je Dateiojekt auch einen Sicherheitsdeskriptor.
- Unterstützung für große Platten und große Dateien
- Mehrere Datenströme, jede Datei ist ein Strom aus Bytes, mehrere Ströme pro Datei sind möglich. Maximale Stromlänge: 2^{64} Byte
- Allg. Indexunterstützung, Dateibeschreibungen werden mittels einer relationalen Datenbank verwaltet, so dass alle Attribute als Index genutzt werden können.
- Kompression: jeweils 16 Blöcke werden komprimiert abgelegt
- Verschlüsselung: EFA (Encrypting File System) als Treiber zwischen NTFS und Anwenderprozess, symmetrischer 128-bit Schlüssel je Datei, Schlüssel werden mit Public Key selbst verschlüsselt abgelegt

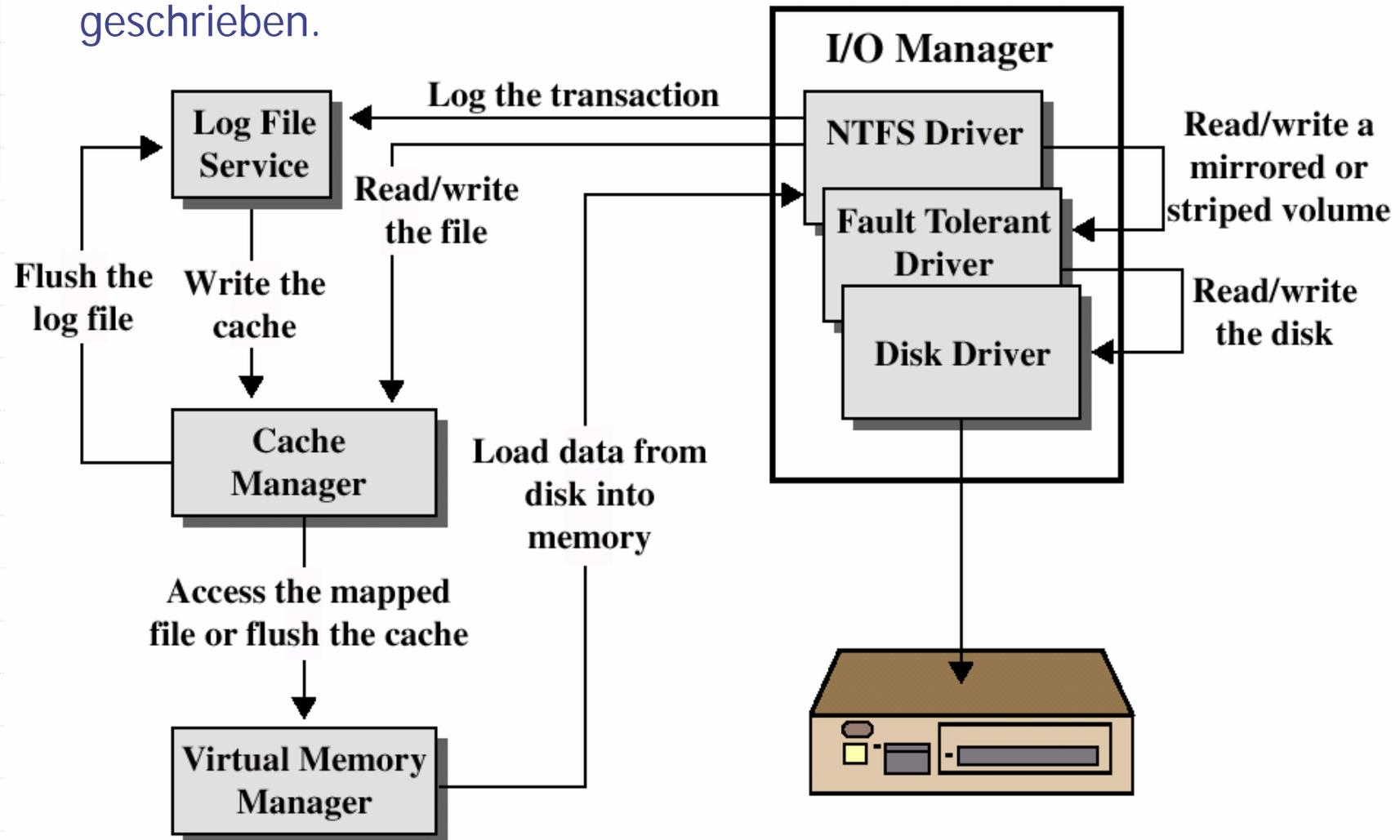
Beispiel: Dateimanagement in Windows 2000

NTFS Volume und Dateistruktur

- ◆ Sector: kleinste physikalische Einheit der Platte, typ. 512 B
- ◆ Cluster: eine/mehrere aufeinanderfolgende Sektoren, jeweils 2^n , Cluster bilden kleinste belegbare Speichereinheit, typ. 1K. Maximale Dateigröße 2^{32} Cluster, 2^{48} Bytes. Große Dateien lassen sich durch Verwendung großer Cluster verwalten, weil dann weniger Objekte verwaltet werden müssen. Clustergrößen lassen sich bei Formatierung eines Volumes festlegen.
- ◆ Volume: logische Partition einer Platte aus einem/mehreren Clustern. Besteht aus Dateisysteminformation, Dateien und freiem Speicher, der noch belegt werden kann. Volumes dürfen über Platten verteilt sein, maximale Größe 2^{64} Bytes.
- ◆ Volume Layout:
 - Partition Boot Sector (≤ 16 Sektoren): Layout Information, Dateisystemstruktur, Boot Startup Informationen und Code
 - Master File Table: Informationen bzgl Dateien, Directories, Freispeicher im Format einer Tabelle einer relationalen Datenbank
 - System Files (1MB): Kopie des MFT Anfangs (MFT2), Logfile für Transaktionen und Recoverability, Clusterbitmap, Attributdefinitionen
 - File Area: eigentliche Dateien

Beispiel: Dateimanagement in Windows 2000

Recoverability nur in Bezug auf Systemdateien, i.w. werden log Dateien mit Transaktionen geführt, Änderungen wirken nur auf den Cache, log Dateien werden vor den eigentlichen Dateien auf die Platte geschrieben.



Zusammenfassung

- ◆ Grundbegriffe, Aufgaben im Dateimanagement
- ◆ Dateiformate
 - Pile, sequentielle Datei, index-sequentielle Datei, Datei mit Index, Datei mit Hashing
 - Directories
- ◆ Gemeinsamer Zugriff auf Dateien
- ◆ Zusammenfassen von Records in Blöcken
- ◆ Verwaltung von Dateien im Sekundärspeicher
 - kontinuierlich, verkettet, mit Index (feste oder variable Längen)
- ◆ Recoverability in Bezug auf Systemdateien:
 - Logfiles, Transaktionskonzept
- ◆ Beispiele
 - Dateimanagement in UNIX: Inode
 - Dateimanagement in Windows 2000: Cluster, Volume,