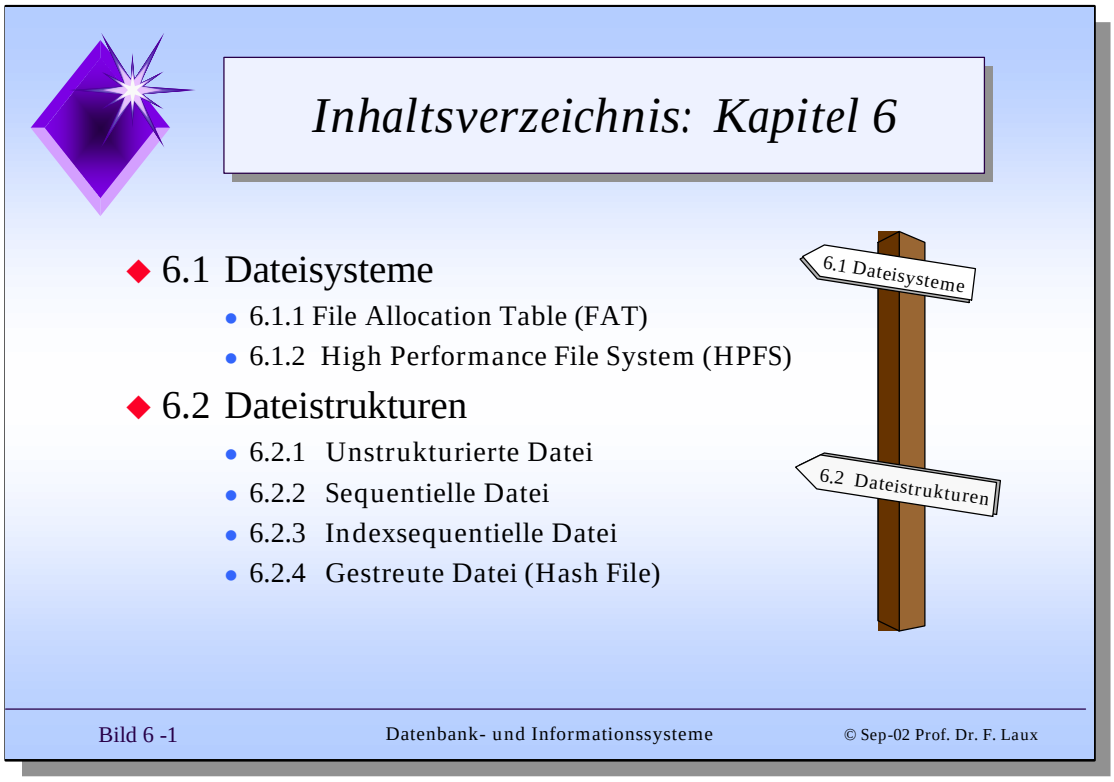
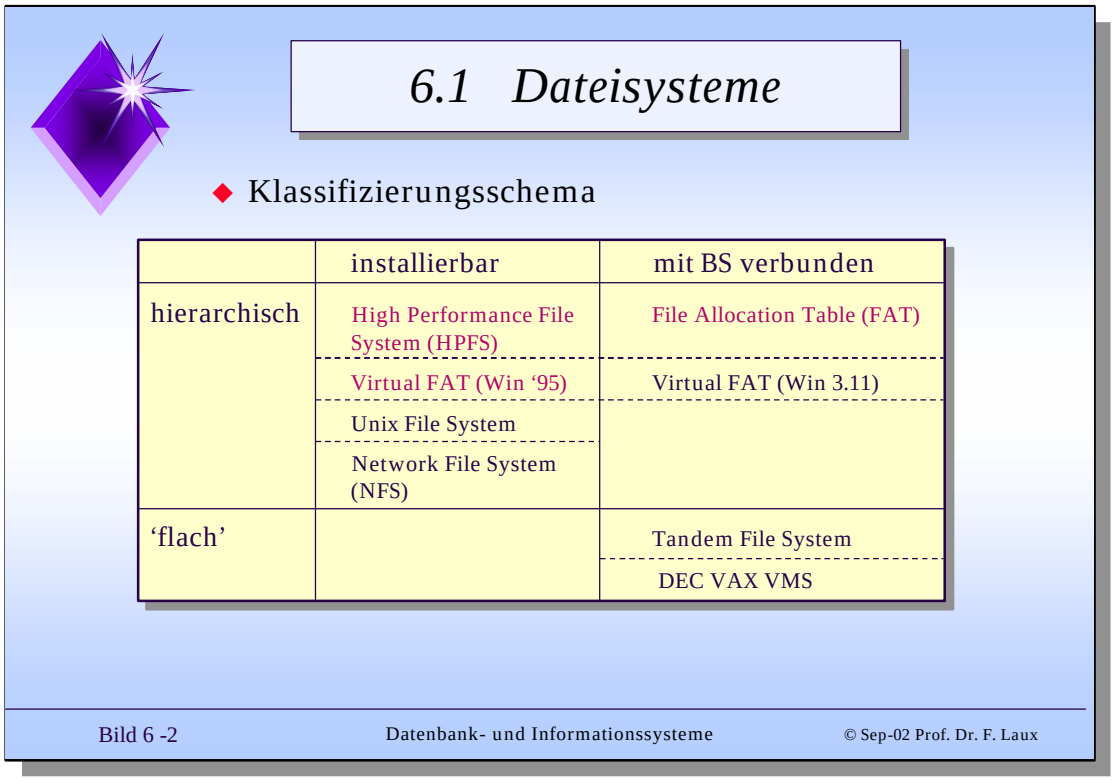


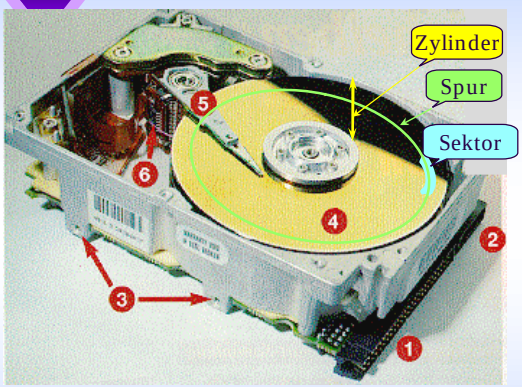
Das letzte Kapitel ist der Dateiorganisation gewidmet. Wir werden zwei Dateisysteme kennenlernen und einige wichtige Dateistrukturen besprechen. Vom Datenbankstandpunkt beziehen sich diese Betrachtungen auf die interne Ebene der ANSI/SPARC Architektur.



Anhand der oben gezeigten Klassifizierung können wir uns leicht orientieren. Wir unterscheiden hierarchische und flache Dateisysteme. Bei letzteren sind die Verzeichnisse nicht oder nur gering geschachtelt (z.B. 2 Stufen). Dadurch sind sie inflexibel und gelten als veraltet. Moderne Dateisysteme sind installierbar, d.h. sie sind unabhängig vom Betriebssystem und benötigen einen Treiber, der beim Systemstart installiert wird. Das NFS, NTFS (New Technology File System) und das HPFS sind Beispiele für installierbare Systeme. Manche Betriebssysteme erlauben die gleichzeitige Verwendung mehrerer Dateisysteme (z.B. OS/2, Unix, Windows NT).

Unabhängig von Dateisystem können Dateien hierarchisch oder nichthierarchisch dem Benutzer präsentiert werden. Verweise von der Arbeitsoberfläche direkt auf Dateien sind 'flache' Sichten auf die Dateien. Die Abbildung der hierarchischen Struktur auf eine Ebene wird oft bei der Installation von Software vorgenommen, kann aber auch manuell durchgeführt werden.

6.1 Aufbau eines Plattenspeichers



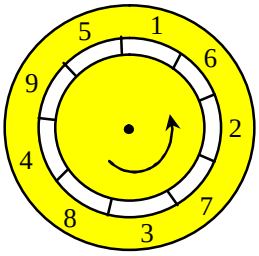
- **Begriffe**
  - Sektor := kleinste physische Datenübertragungseinheit
  - Spur := alle Sektoren einer Plattenoberfläche bei konstanter Kopfposition
  - Zylinder := alle Spuren einer Kopfposition
- **Beispiel**
  - Sektor := 512 Bytes
  - Spur := 63 Sektoren (= 32 kB)
  - Zylinder := 2633 à 16 Spuren (1,358 GB)

(1) Datenanschluß (SCSI, IDE)
(2) Stromversorgung
(3) Befestigung
(4) Datenträger
(5) Positionierarm mit Schreib/Lese-Köpfen
(6) Schrittmotor zur Positionierung

Bild 6 -3 Datenbank- und Informationssysteme

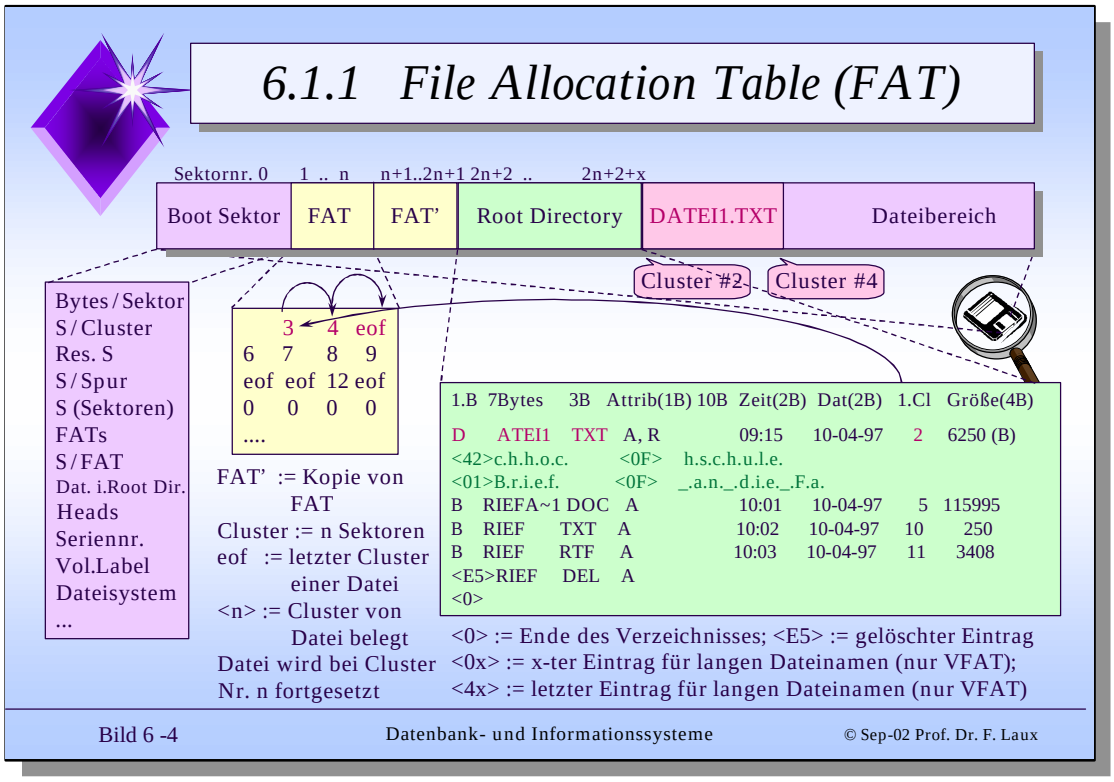
Der übliche Massenspeicher für direkten (*wahlfreien*) Datenzugriff ist die Festplatte. Moderne Speicher können in einem Gehäuse von 6 x 10 x 2 cm (etwa Zigarettenschachtelgröße) über 15 Gigabyte speichern (Stand 1999). Die *Zugriffszeit* **z** für eine beliebige Dateneinheit beträgt im Durchschnitt ca. 12 ms. Hochleistungsspeicher können durch höhere Rotationsgeschwindigkeit und große Datenpuffer (cache) auf mittlere Zugriffszeiten von unter 8 ms kommen. Die Daten selbst werden als magnetisierte Bereiche (*Sektoren*) in konzentrischen Ringen angelegt. Daher muss der Schreib- und Lesekopf sich zuerst auf einen solchen Ring (*Spur*) positionieren, bevor Daten gelesen werden können. Wenn mehrere Platten an einer Spindel montiert sind, muss der Positionierarm auch für jede Spur einen Schreib/Lesekopf haben. Also können in einer Position sämtliche Daten eines *Zylinder*s gelesen oder geschrieben werden. Der Lese- oder Schreibvorgang erfolgt, wenn die gewünschte Dateneinheit (Sektor) unter dem Schreib/Lesekopf vorbei kommt. Je schneller eine Platte rotiert, desto geringer ist die Latenz- oder Wartezeit **w** (max. eine Umdrehung) für einen gewünschten Datenbereich und desto schneller können die Daten gelesen werden (Transferzeit **t**). Bei langsamen Prozessoren kommen die Daten schneller am Lesekopf vorbei, als sie dann zum Arbeitsspeicher (Memory) weitertransportiert werden können. In diesem Fall muss die Platte einmal leer drehen, bis der nächste Sektor gelesen werden kann. In diesem Fall ist es günstiger, die Sektoren nicht aufeinanderfolgend zu numerieren, sondern einen oder zwei Sektoren Zwischenraum zu lassen (interleave factor), so daß aufeinanderfolgende Daten ohne Leerdrehung gelesen werden können.

Beispiel: Interleave Factor = 2



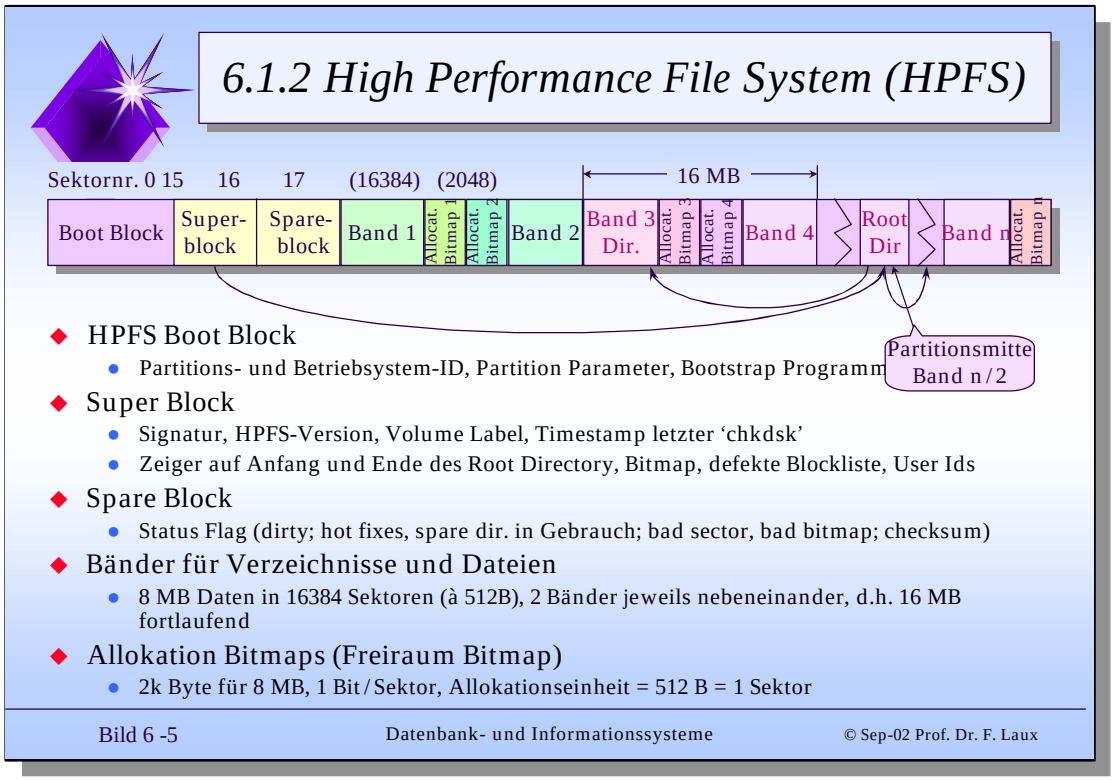
Zugriffszeit z = p + w + t
Positionierzeit p, Wartezeit w, Transferzeit t

mittlere Zugriffszeit
Æz = Æp + w/2 + Æt



Die File Allocation Table (FAT) wurde als ein in DOS integriertes Dateisystem konzipiert, das vor allem bei kleineren Speichergrößen (Disketten) sehr effizient und problemlos arbeitet. Mit zunehmender Speicherkapazität wächst allerdings die Clustergröße wegen der DOS-Limitierung auf maximal 2^{16} = 65536 Cluster pro logischer Platte. Bei einer Plattengröße zwischen 512 und 1024 MB sind es bereits 16 kB/Cluster. Für die zur Zeit üblichen Plattengrößen (Stand 1999) reicht auch die maximale Clustergöße von 64kB nicht mehr aus, um die gesamte Kapazität zu adressieren. Bei der FAT32 Variante können 2^{32} = 4 Mrd. Cluster eingesetzt. Nachdem die Speicherzuweisungseinheit ein Cluster beträgt, belegt eine Datei immer Vielfache einer Clustergröße. Dies bedeutet, daß durchschnittlich eine halbe Clustergröße bei einer Datei unbenutzt bleibt. Bei vielen kleinen Dateien ist dies eine beachtliche Verschwendung.

Sektor 0 der 1. Spur wird als Bootsektor bezeichnet, da diese Information beim Systemstart zuerst gelesen wird. Danach folgen die Zuweisungstabellen (FAT) in zweifacher Ausführung. Dadurch ist in Fehlerfällen häufig eine leichte Reparatur des Dateisystems möglich. Die FAT enthält für jeden Cluster einen Eintrag (die Zählung beginnt bei Cluster 2) An jeder Clusterposition steht entweder die Nummer des nächsten von der Datei belegten Clusters, EOF (wenn die Datei in diesem Cluster endet) oder 0 wenn der Cluster frei ist. Der dritte Bereich ist das Stammverzeichnis (Root Directory). Dieses Verzeichnis enthält die 1. Verzeichnisebene. Für jede Datei oder für jedes Unterverzeichnis (ebenfalls eine Datei, deren Inhalt ein Dateiverzeichnis ist) ist ein Eintrag vorhanden. In diesem wird der 1. Cluster der Datei angegeben. Weitere Cluster, die zu der Datei gehören werden über die FAT ermittelt. Windows 95 erlaubt für eine Datei oder ein Verzeichnis mehr als 8 Zeichen (VFAT). In diesem Fall wird ein normaler Eintrag mit einem verkürzten Namen, der das Tilde-Zeichen (~) enthält, erzeugt. Damit wird sichergestellt, daß auch DOS und ältere Windows-Versionen die Datei ansprechen können. Für den langen Namen werden weitere Einträge (Ergänzungseinträge) erzeugt, die im 1. Byte des Verzeichniseintrags eine Binärzahl zwischen 1 und 42 hex enthält. Der letzte dieser Einträge, die zu einer Datei gehören enthält den Wert 42 hex. Die Einträge folgen lückenlos aufeinander, dadurch können zusammengehörige Haupt- und Ergänzungseinträge festgestellt werden. DOS betrachtet die Ergänzungseinträge als fehlerhaft und überliest sie deshalb.



Bei FAT, VFAT und FAT32 tendieren die Dateien dazu, daß ihre Daten (Cluster) auf der gesamten Platte verstreut sind. Dies bedeutet Zeitverzögerungen durch mehrfaches Positionieren. Neben dem bereits erwähnten 'Verschnitt' bei großen Cluster ist eine effiziente Pufferung von Daten nicht möglich, da das Dateisystem keinen Indikator hat, ob ein Datenbereich verändert wurde oder nicht.

Um diese Einschränkungen zu überwinden, entwickelten IBM und Microsoft das HPSF Dateisystem. Es teilt den Plattenspeicher in 8 MB große Bänder (bands) ein, die jeweils auch ein Verzeichnis besitzen, das im gleichen Band gespeichert wird. Dadurch verringert sich der Positionieraufwand zwischen Verzeichnis und Dateilokation. Durch diese Einteilung in Bänder wird die Zugriffszeit geringer und einer Fragmentierung entgegengewirkt. Die Zuweisungseinheit beträgt 512 Bytes (Block). Es gibt einen Reserveblock (spare block) in dem defekte Bereiche markiert und Ersatzblöcke zugewiesen werden. Außerdem wird darin festgehalten, ob ein Block verändert, aber noch nicht wieder zurückgeschrieben wurde (dirty flag). Damit können Schreibvorgänge optimiert werden indem logisches (WRITE-Befehl in einem Programm) und physisches Schreiben (eines oder mehrerer Sektoren) entkoppelt wird. Das Stammverzeichnis befindet sich zur Optimierung der Zugriffszeit in der Mitte der Platte. Im Superblock befinden sich alle Einstiegsadressen für Stammverzeichnis, Spare Block, User Ids, etc und wann die Platte zuletzt überprüft wurde.

6.1.2 HPFS Kenndaten

- Verzeichnis
  - 254 Zeichen für Dateinamen, Datum der Erstellung, letzte Änderung und Zugriff
- Dateien
  - für keine und große Dateien geeignet (16 MB am Stück, max. 64 GB (2 TB), Verschnitt ~ 1/2 Sektor (256 B))
  - erweiterte Attribute (Datentyp, Icons, Zugriffsrechte) integriert
- Zugriffsgeschwindigkeit und Optimierung
  - Verzeichnis, Alloc. Bitmaps, erw. Attrib. und Dateien in einem 8 MB großen Band
  - Dateinamen sortiert in B-Baum
  - Fragmentierung wird durch Überhang am Dateiende vermieden
  - Caching entsprechend des Dateityps, verzögertes Schreiben
- Sicherheit
  - Fehlertoleranz durch redundante Speicherung von Bitmaps, Reserveblock und Hot Fix
  - Datenrecovery von HPFS unterstützt (dirty bit, Doppelverkettung, Signatur)

Bild 6 -6 Datenbank- und Informationssysteme

Die Folie gibt einen Überblick über die Vorzüge des HPFS gegenüber FAT. Besonders eindrucksvoll kommen die Vorteile bei Platten mit großer Speicherkapazität und sehr vielen kleinen Dateien zu Geltung. Beispielsweise ist der Aufwand, einen Dateieintrag zu suchen nur logarithmisch (O(log(n)) bezogen auf die Gesamtzahl (n) der Einträge; bei der FAT hingegen ist er linear (O(n)). Das Dateisystem NTFS für Windows NT ist ähnlich aufgebaut; bis zur Version 3.5 konnte OS/2 auch NTFS Platten lesen.

Da die Schreibvorgänge verzögert erfolgen (gepuffert werden), darf das Betriebssystem nicht wie bei DOS einfach ausgeschaltet werden, da sonst Datenverluste auftreten können. Deshalb ist unbedingt darauf zu achten, daß Betriebssysteme mit installiertem HPFS, NFS, NTFS oder Unix Filesystem immer richtig 'herunter gefahren' werden. Ansonsten sind HPFS und NTFS wesentlich robuster als FAT. Verlorene Blöcke (Cluster), wie sie bei FAT vorkommen, sind durch die zusätzlichen Informationen im Superblock und in der Allocation Bitmap äußerst selten.

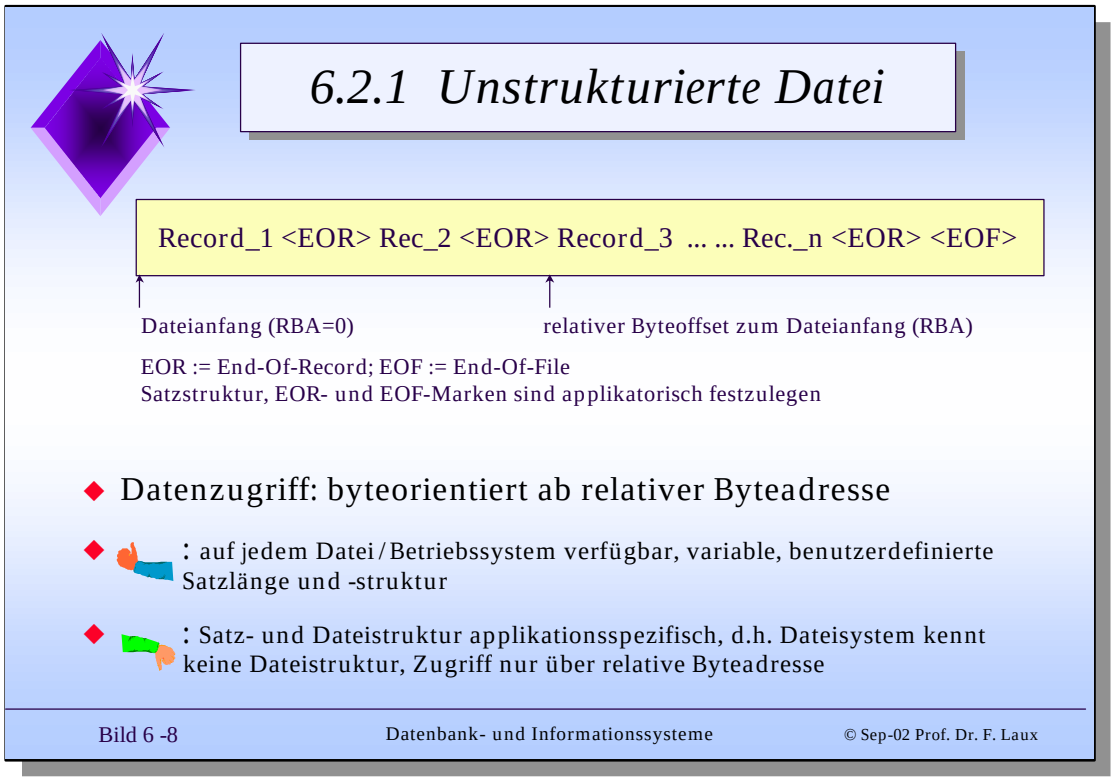
6.2 Dateistrukturen

- Unstrukturierte Datei (Unix, DOS, Windows)
  - Bytefolge (Stream), relative Byte-Adressierung
- sequentielle Datei (COBOL, HSAM)
  - variable Satzlänge, Reihenfolge zeitlich geordnet, Einfügen nur am Dateiende, sequentieller Zugriff über die Satznummer
- relative Datei (COBOL)
  - feste Satzlänge, direkter Zugriff über die Satznummer
- index-sequentielle Datei (ISAM, COBOL, DBS)
  - variable Satzlänge, sequentieller und indizierter Zugriff über Primärschlüssel
- gestreute (hash) Datei (DBS)
  - feste Satzlänge, direkter Zugriff über Hash-Schlüssel

Bild 6 -7 Datenbank- und Informationssysteme

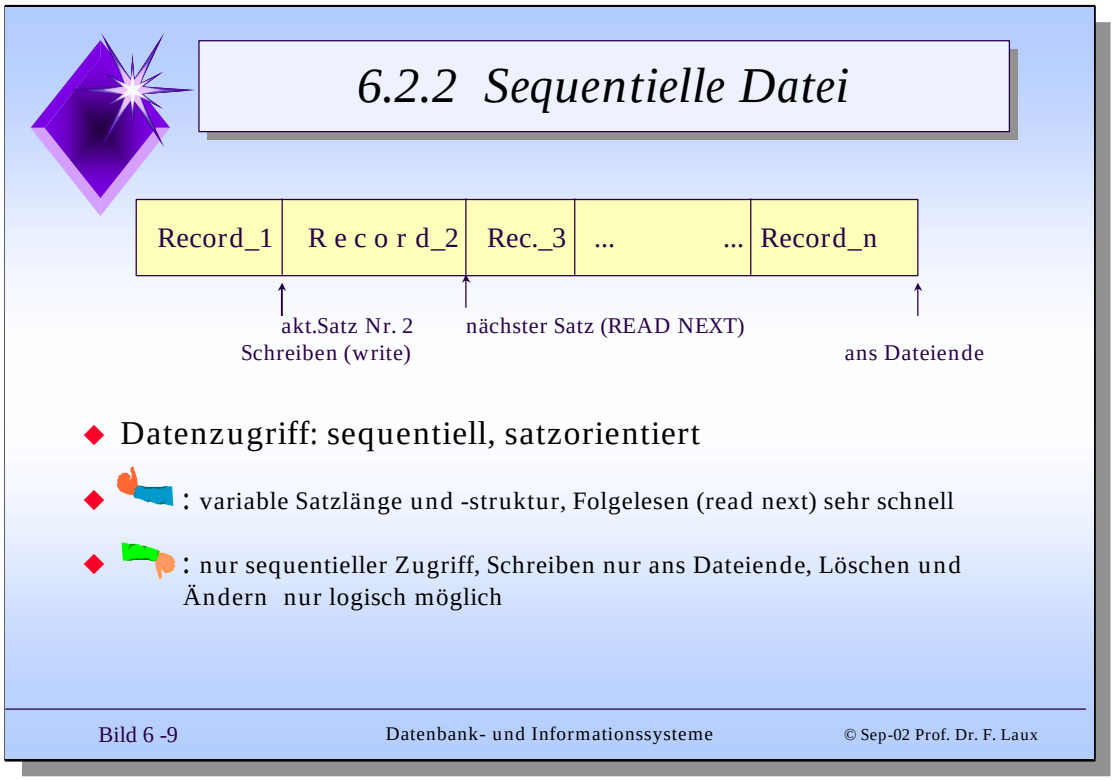
Wir haben bisher die Einteilung von Massenspeichern mit direktem Datenzugriff (Festplatten, Disketten, Wechselplatten, etc.) vorgestellt und gezeigt, wo und wie Dateien abgelegt werden. Im folgenden soll nun der interne Dateiaufbau besprochen werden.

Es werden sechs Dateistrukturen untersucht, von denen die erste von PC Systemen her bekannt sein dürfte. Sie bietet die flexibelste Datenablage aber gleichzeitig auch den wenigsten Komfort. Die restlichen Strukturen mit Ausnahme der letzten stehen unter COBOL zur Verfügung. Als COBOL entwickelt wurde, gab es noch keine Datenbanken. Deshalb wurde auf eine gute Unterstützung effizienter Dateistrukturen geachtet. Heute haben diese Dateiorganisationen an Bedeutung verloren, da moderne Datenbanksysteme komfortablere Speicher- und Suchmöglichkeiten bieten. Insbesondere ist der Satzaufbau (Datenfelder, Attribute) bei keiner der zu besprechenden Dateistrukturen dem Dateisystem bekannt. Im Gegensatz hierzu können Datenbanken bekanntlich einzelne Attribute adressieren. Wir besprechen die genannten Dateistrukturen trotzdem, da sie auch heute vor allem bei der Organisation von Daten im Arbeitsspeicher von Bedeutung sind. Gestreut organisierte Dateistrukturen kommen wegen ihrer festen Satzlänge vor allem in relationalen Datenbanksystemen zum Einsatz.



Bei *unstrukturierten Dateien* hat der Programmierer bzw. der Benutzer für die Einteilung (Strukturierung) selbst zu sorgen. Das Dateisystem kennt nur die Datei selbst und kann nur durch eine *relative Byteadresse* (Bytes werden ab Dateianfang durchnumeriert) positioniert werden. Satz- und Feldmarken werden bei der Adressierung mitgezählt, da das System diese nicht als solche erkennt. Der Benutzer muß diese Marken beim Schreiben selbst hinzufügen und beim Lesen wieder herausfiltern.

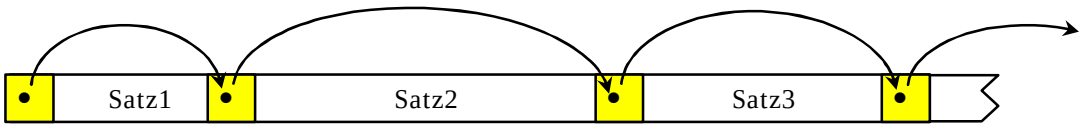
Damit ist es möglich, jede beliebige Struktur zu verwenden, sogar die Struktur innerhalb der Datei zu ändern (z.B. ein Headerrecord, dann die Datensätze), solange dies in allen Anwendungsprogrammen gleich realisiert wird. Werden variable Satzlängen verwendet, können die Sätze nicht direkt adressiert werden, sondern müssen sequentiell vom Anwendungsprogramm gesucht werden.

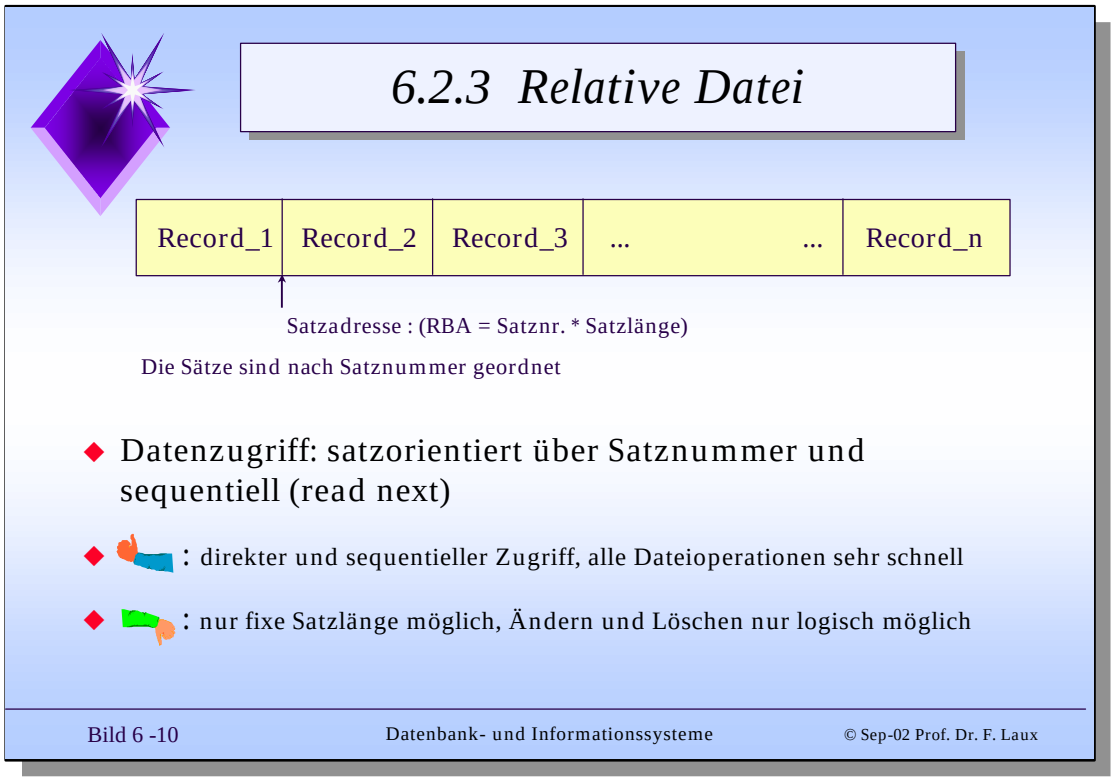


Die sequentielle Dateistruktur bietet die gleiche Flexibilität für die Gestaltung der Satzstruktur wie eine unstrukturierte Datei. Der Unterschied besteht darin, daß das Dateisystem die Satzstruktur kennt. Das Filesystem kann deshalb satzorientiert die Datei bearbeiten. Vom Anwendungsprogramm aus kann auf einen beliebigen Satz positioniert werden, wegen der variablen Satzlänge ist aber das Dateisystem gezwungen, sequentiell zu positionieren (sich von Satz zu Satz zu 'hangeln'). Neue Datensätze werden jeweils ans Ende der Datei geschrieben, d. h. sie sind entsprechend dem Erfassungszeitpunkt geordnet (chronologische Ordnung).

Es gibt mehrere Möglichkeiten für das Dateisystem, die Sätze zu markieren:

- für Textdateien kann ein nichtdruckbares Zeichen als Satzende verwendet werden
- für Dateien mit beliebigem Inhalt werden Zeiger auf das Satzende verwendet.

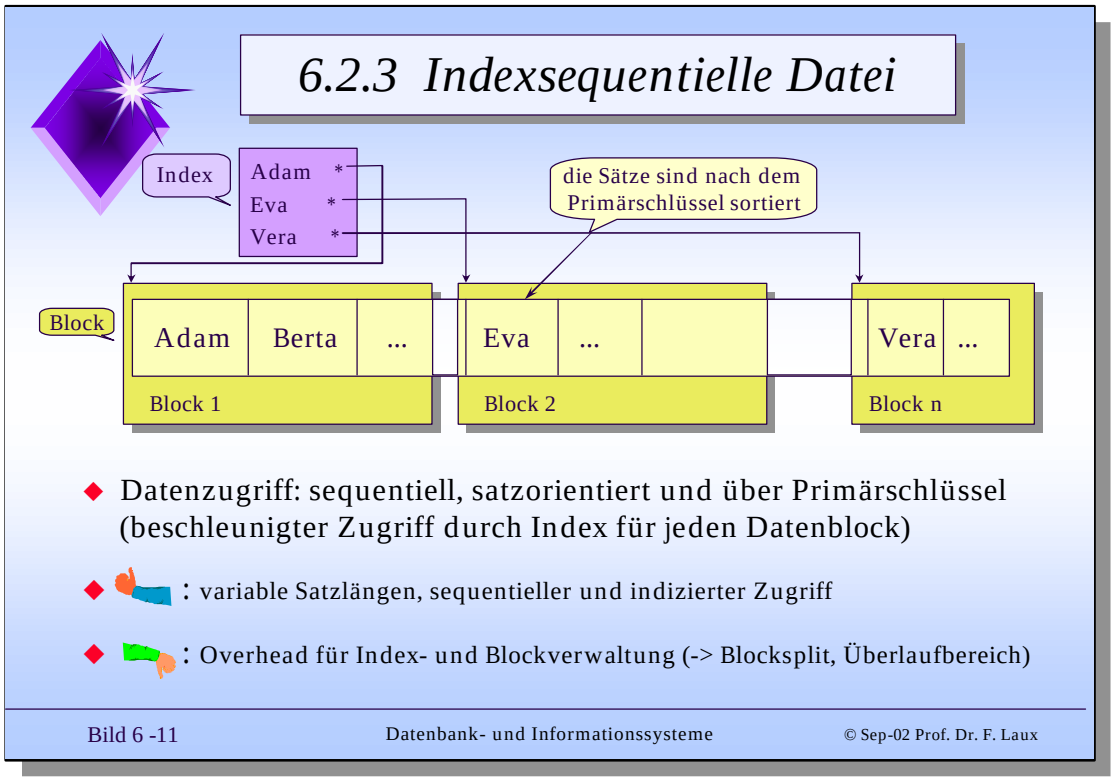




Die relative Datei ist aus Sätzen fester Länge aufgebaut. Dadurch kann das Filesystem die Position eines Satzes berechnen:

Satzposition (RBA) := Satznr. * Satzlänge (inkl. Satzmarken)

Dies ermöglicht eine direkte Positionierung auf einen bestimmten Satz. Allerdings können - wie auch bei den andern bisher besprochenen Strukturen - die Sätze nicht physisch gelöscht oder in ihrer Länge geändert werden, ohne die Datei zu reorganisieren.

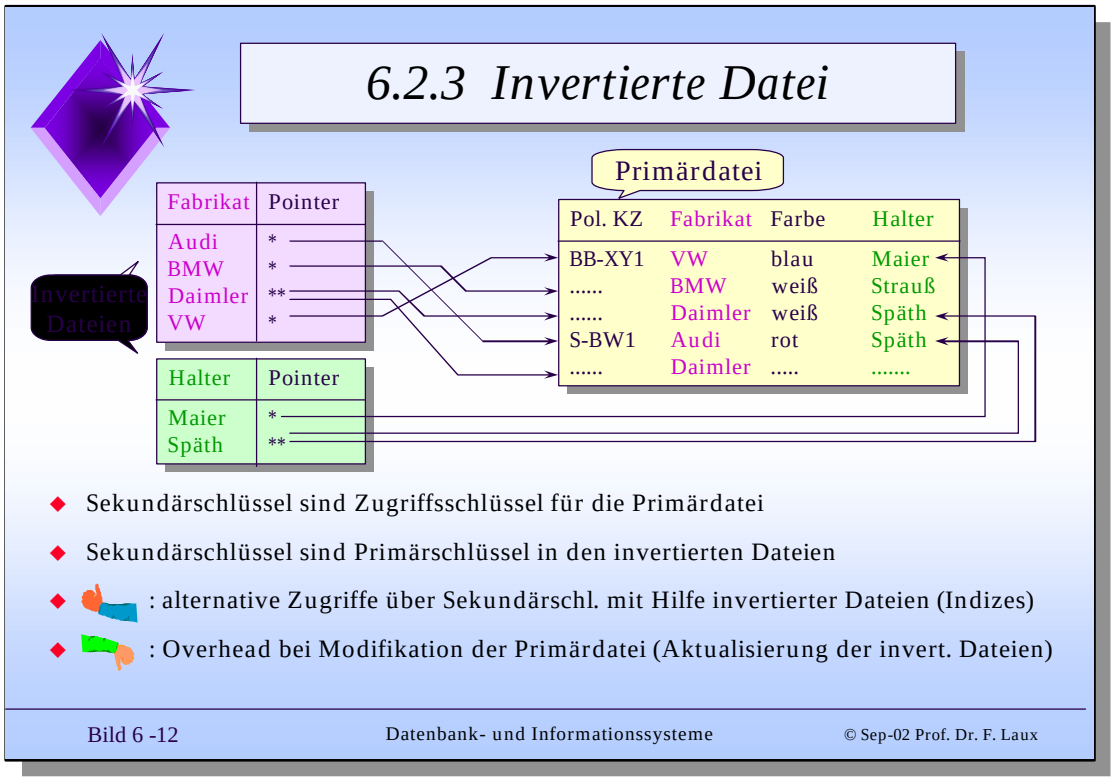


Die bisherigen Strukturen hatten entweder einen feste Satzlänge und konnten deshalb einen Satz direkt adressieren oder Sie hatten ein variable Satzlänge, konnten aber die Sätze nur sequentiell suchen oder lesen.

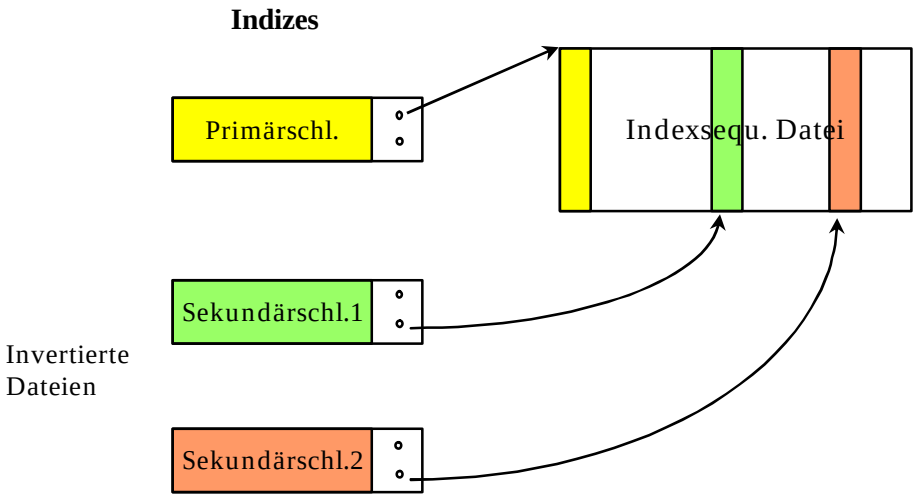
Die indexsequentielle Datei versucht die Vorteile beider Strukturen miteinander zu verbinden. Deshalb wird eine variable Satzlänge wie bei einer sequentiellen Datei verwendet. Die Sätze werden durch ein eindeutiges Kriterium (Primärschlüssel) identifiziert und danach sortiert in die sequentielle Datei eingefügt. Um einen schnelleren Zugriff auf die Sätze zu haben, werden diese zu Blöcken gruppiert und die Blöcke mit einem Index versehen. Dadurch kann bei bekanntem Primärschlüssel die Blockadresse für den Satz aus dem Index entnommen werden. Danach ist nur noch der Block nach dem gewünschten Satz zu durchsuchen.

Die technische Realisierung dieser Dateiorgansisation erfolgte durch IBM mit dem System /360. Die ISAM (Index Sequential Access Method) benützt physische Blöcke (Zylinder) und löste das Einfügeproblem durch einen Überlaufbereich. Dort werden Sätze, die in einem Block keinen Platz mehr finden, abgelegt. Der letzte Satz eines Blocks enthält einen Pointer auf den folgenden Satz im Überlaufbereich. Wenn sehr viele Sätze eingefügt werden, wird der Überlaufbereich sehr groß und damit der Zugriff zunehmend sequentiell. Deshalb ist bei Bedarf die Datei zu reorganisieren. Dabei wird die Datei komplett neu aufgebaut, wobei ein wählbaren Prozentsatz eines Blockes für zukünftige Sätze freigehalten wird (*Reservebereich* pro Block).

Eine elegantere Methode, das Insert-Problem zu lösen, sind sogenannte *Blocksplits*. Wenn ein Block einen neuen Satz nicht mehr aufnehmen kann, wird ein neuer Block angelegt und die Sätze auf beide Blöcke gleichmäßig verteilt. Dann muß ein neuer Index hinzugefügt werden, damit der neue Block auch adressiert werden kann. Dieses Vorgehen bedeutet einen größeren Aufwand beim Einfügen, macht aber eine Reorganisation überflüssig und bietet kalkulierbare Zugriffszeiten.

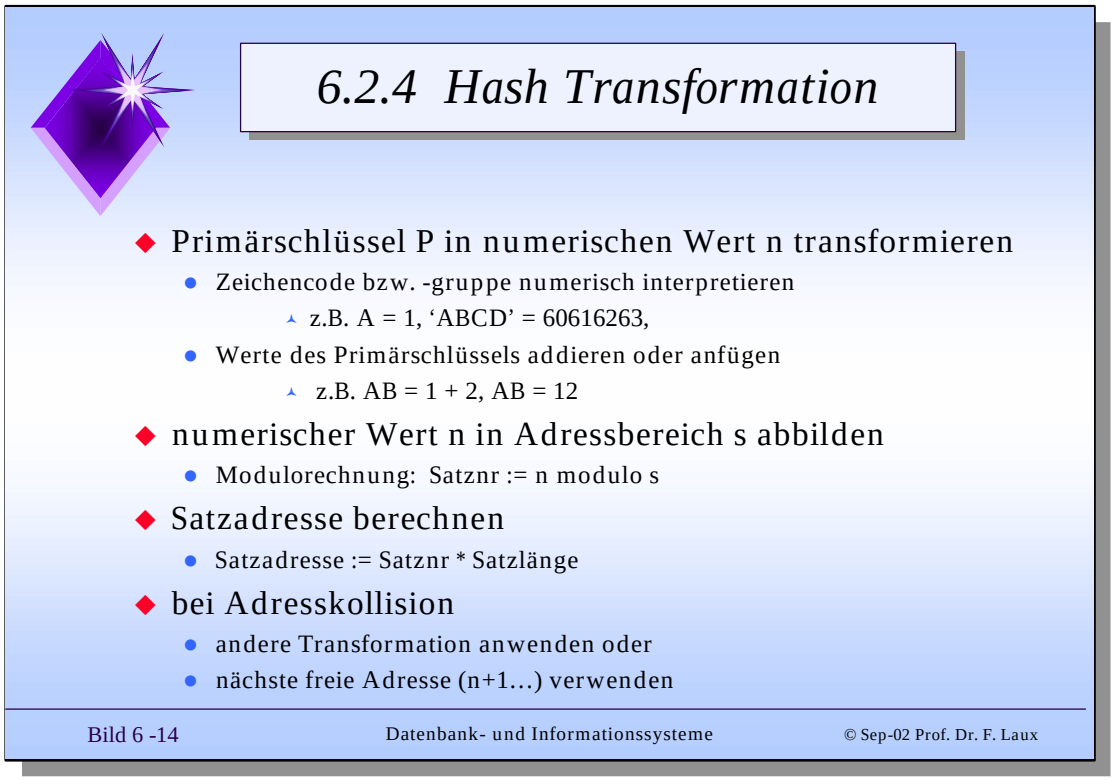


Benötigt man schnellen Zugriff nach verschiedenen Kriterien (Datenfeldern), so können weitere Indizes angelegt werden. Diese werden in sogenannten invertierten Dateien (inverted files) gespeichert. Der Name kommt daher, daß ein Feld, das nicht Primärschlüssel ist, also üblicherweise hinter diesem positioniert ist, in der invertierten Datei als Index und damit als Primärschlüssel (d.h. an der ersten Position) verwendet wird.



Gestreut organisierte Dateien verwenden eine Transformation zur Bestimmung der Satzposition. Der Wert des Primärschlüssels wird in eine Satzadresse umgerechnet. Dadurch sind direkte Satzzugriffe ohne Suche auf der Platte möglich. Werden Sätze in ihrer physischen Anordnung in der Datei gelesen, so erscheinen sie völlig ungeordnet. Außerdem muß die Datei komplett gelesen werden, da Datensätze über die ganze Datei 'verstreut' sind. Dies bedeutet bei großen Dateien einen beachtlichen Aufwand, auch wenn nur wenige Sätze in der Datei vorhanden sind. Sucht man hingegen nach einem bestimmten Satz und ist der Primärschlüssel bekannt, so kann die Adresse sofort berechnet werden.

Bei typischen betriebswirtschaftlichen Aufgaben liegt dem Primärschlüssel (Personalnr, Artikelnr, etc.) eine bestimmte Systematik zugrunde. Dies bedeutet, daß Primärschlüssel nicht gleichmäßig über den gesamten Wertebereich verteilt sind. Deshalb ist es eine Herausforderung, eine Transformation zu finden, welche die Sätze möglichst gleichmäßig auf die gesamte Datei verteilt (vgl. nächste Seite). In der Praxis lassen sich allerdings Adresskollisionen (mehrere Primärschlüssel erhalten die gleiche Adresse zugewiesen) kaum vermeiden. Für diese Fälle müssen Ersatzadressen zugeteilt werden. Dies erhöht den Adressierungsaufwand. Statistisch gesehen tritt dieser Zustand erst bei einem Füllgrad von über 95% merkbar in Erscheinung.



Der gezeigte Algorithmus ermittelt aus einem Schlüsselwert eine Zahl, die durch die Dateigröße dividiert wird. Der dabei entstehende Teilerrest (Modulo) liegt in einem durch die Dateigröße vorgegebenen Bereich. Die relative Byteadresse des Satzes ergibt sich durch Multiplikation mit der Satzlänge.

Wenn sich eine Adresskollision ergibt, kann eine andere Transformation angewandt werden, um eine frei Adresse zu finden. Die alternativen Transformationen sind natürlich vorher festzulegen. Eine weitere Möglichkeit ist, von der zuerst berechneten Adresse ausgehend sequentiell weiterzusuchen, bis eine freie Adresse gefunden wird. Bei beiden Methoden ist ein Abbruchkriterium erforderlich, um 'ewiges' Suchen zu vermeiden.