

Dateisysteme

Betriebssysteme WS 2007/2008



Jörg Kaiser
IVS – EOS

Otto-von-Guericke-Universität Magdeburg

Dateisysteme: Motivation

Wozu wird eine zusätzliche Art von Speicher benötigt?

Persistenz ?

Gemeinsame Nutzung ?

Zugriffsschutz ?

Größe ?



Themen zu Dateisystemen



Allgemeine Struktur eines Dateisystems

- Organisation der Dateien
- Organisation der Verzeichnisse
- Zugriff zu Dateien und Verzeichnissen



Organisation der Platte

- Blockstruktur der Platte
- Abbildung von Dateien und Verzeichnissen
- gemeinsame Nutzung von Dateien



Verwaltung der Platte auf Blockebene



Verbessern der Leistung von Dateisystemen



Zuverlässigkeit und Konsistenz von Dateisystemen



Dateisysteme: Die systemorientierte Sicht

Dateien als allgemeine Abstraktion für langlebige Einheiten:

- ➔ **Benutzerdokumente: reguläre Dateien**
- ➔ **Programme: ausführbare Dateien**
- ➔ **Strukturen der Dateiorganisation: Verzeichnisse**
- ➔ **Abstraktionen für Speichergeräte: Block-Dateien**
- ➔ **Abstraktionen zur Modellierung v. Geräten: Spezial-Dateien**

Unterschiede werden im Dateityp festgehalten.



Dateinamen

Beispiele

name.extension	Bedeutung
name.txt	Textdatei
name.c	C Quelldatei
name.o	Objektdatei (Maschinencode nicht gelinked)
name.bak	Backup-Datei
name.jpg	Datei codiert im JPEG Standard
name.mp3	Datei codiert im MPEG 3 Standard
name.pdf	pdf Datei (portbale document format)

gif, tiff, as, ps, zip, tex, hlp, html, doc, exe, xls.....

Variationen: Characters: upper/lower case, unicode,. .

Erweiterungen: Konventionen vs. interpretiert durch das BS



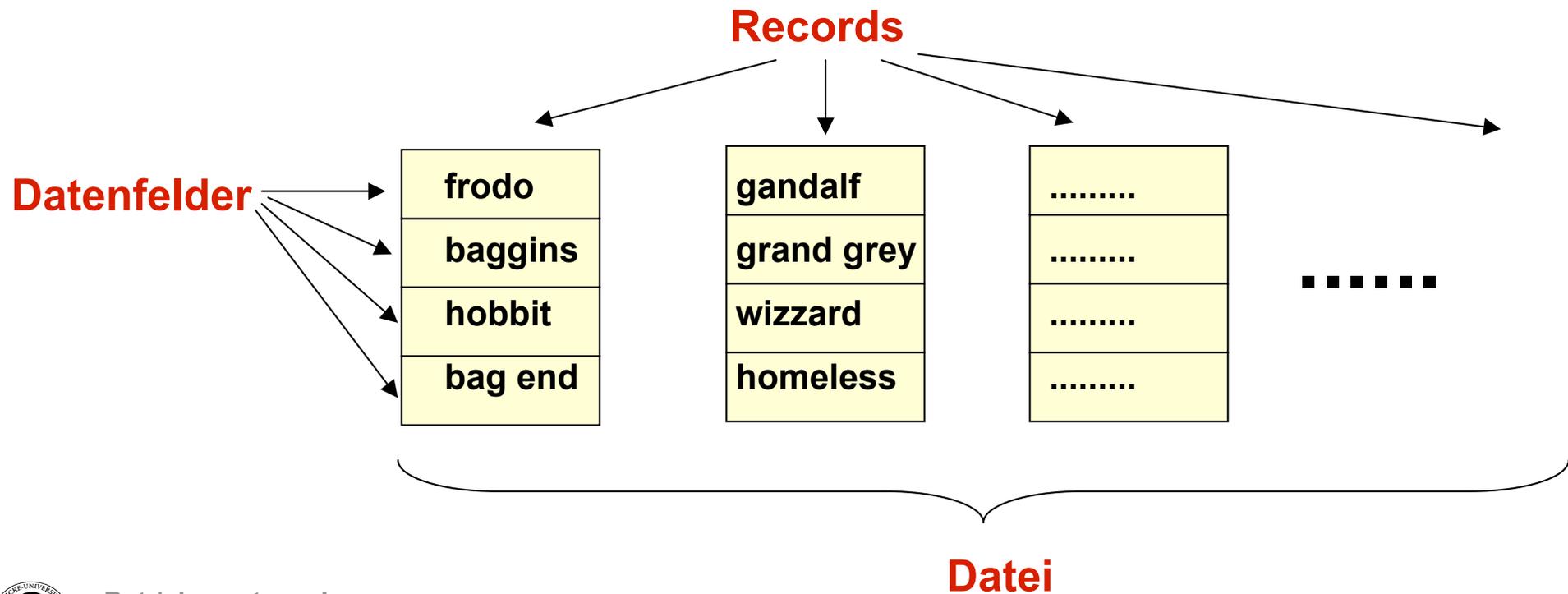
Informationsstruktur

Feld: grundlegendes Datenelement

Record: Menge zusammengehörender Felder

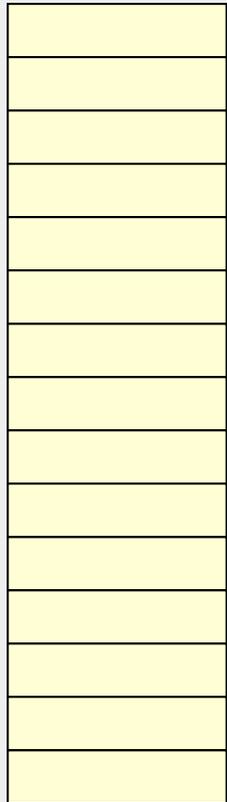
Datei: Menge zusammengehörender Records

Beispiel für die Informationsstruktur: <first name>, <family name>, <origin>, <home address>

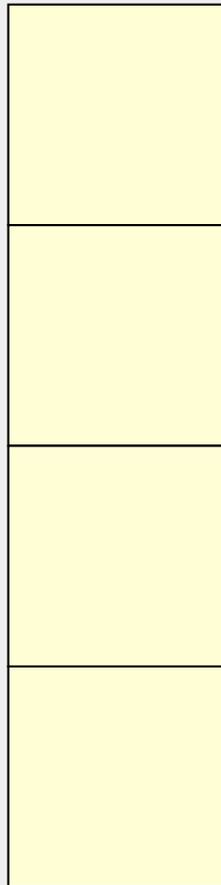


Dateiorganisation

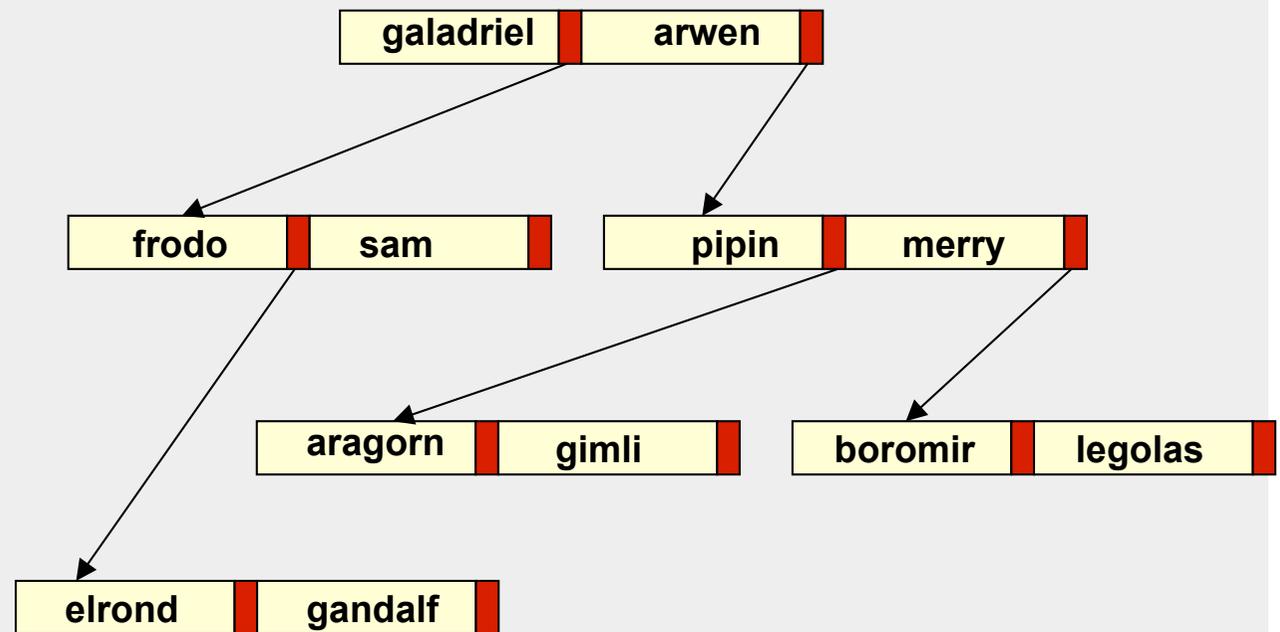
Folge von Bytes



Folge von Records



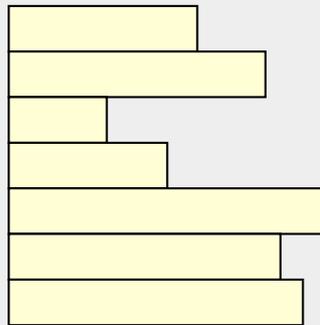
verkettete
(Baum-)
Struktur



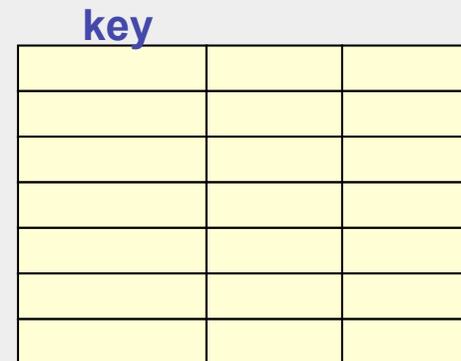
Dateiorganisation und Zugriff

Wie findet man einen Record? Alternativen in der Dateiorganisation:

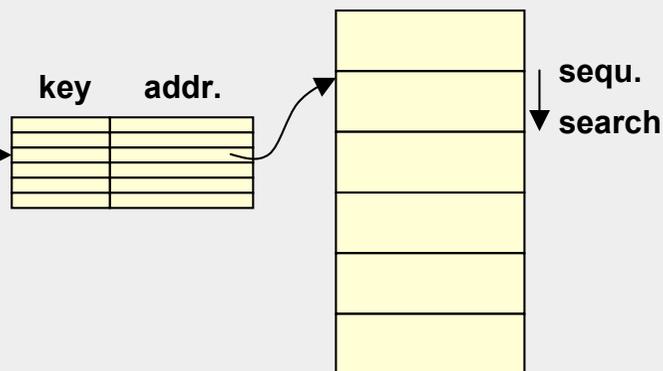
Sammlung (collection):
Records variabler Länge
in chronologischer
Reihenfolge.



Sequentiell:
Records fester Länge
in sequentieller
Ordnung gemäß
eines
Schlüsselfeldes



Index sequentiell:
Sequentielle Suche
im entsprechenden
Bereich



vollständig indiziert:
Index für mehrere Felder
Index für jeden Record
keine sequentielle Suche



Dateiattribute

- Basisinformation:** Dateiname, Dateityp, (Dateiorganisation)
- Addressinformation:** Gerät, phys. Startadresse, akt. Größe, max. Größe
- Zugriffskontrollinfo.:** Besitzer, Zugriffsautorisation, Zugriffsrechte
- Dateiinformatio:** Erzeugungsdaten, Erzeuger ID, letzter Lesezugriff, ID des letzten Lesers, Datum der letzten Modifikation, ID des letzten Schreibers, Datum des Backup, aktuelle Benutzungsinformation.



Dateiattribute

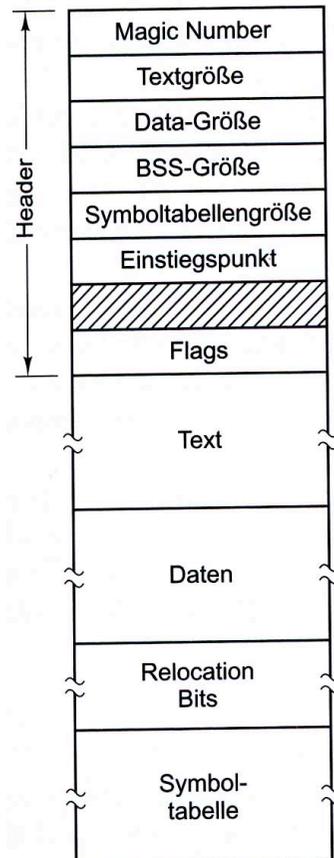
Beispiele von Dateiattributen

access control	who is allowed to do what with the file
passwd	passwd for the file access
creator	ID of the file creator
owner	current owner
read-only-flag	0: R/W, 1:read only
hidden flag	0: default, 1: invisible
system flag	0: normal file, 1: system file
archive flag	0: changes saved, 1: not yet saved
ASCII/binary flag	0: ASCII file, 1: binary file
random access flag	0: sequential file, 1: random access
temporary flag	0: normal, 1: delete file on process termination
lock flags	0: not locked, ≠ 0: file locked
record length	number of bytes in a record
key position	offset to the key in the record
key length	number of bytes in the key
time of last access	date and time of last access to this file
time of last modification	date and time of last change
actual (max) size	number of (max) bytes in file

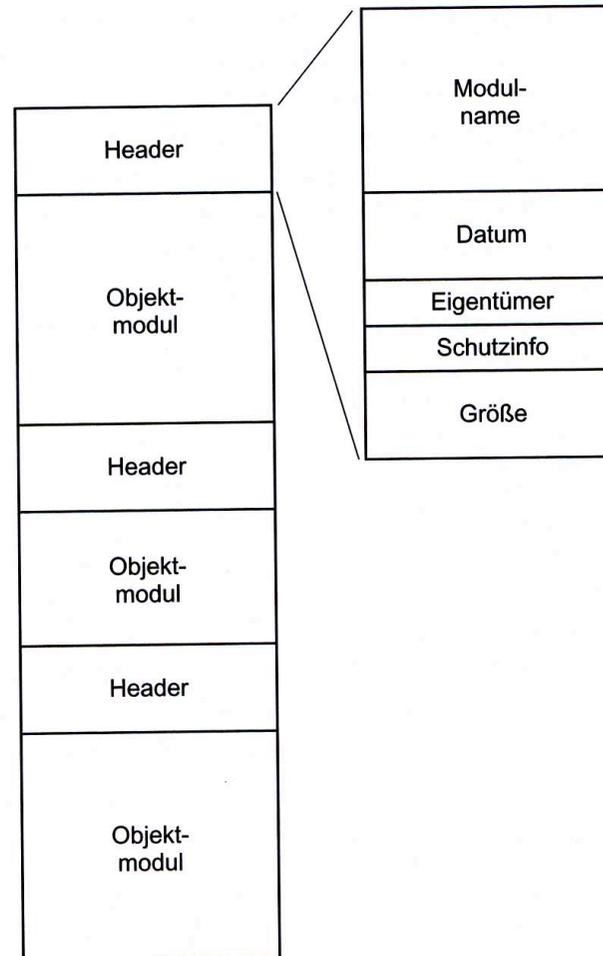


Dateitypen

Ausführbare Datei



Archiv



Typische Zugriffsoperationen für Dateien

Vorbereitung zum
Dateizugriff

create	—————	delete
open	—————	close

normaler
Dateizugriff

read	—————	write
append		

Suchen &
Verwalten

seek		
set attributes	—————	get attributes
rename		



"Memory Mapped" Dateien

Idee: Abbildung von Dateien in den virtuellen Speicher. Ausnutzung des Seiten-Mechanismus, um Dateien zwischen Platte und Hauptspeicher ein- und auszulagern.

Vorteil: Zugriff auf eine Datei kann über normale Lese- und Schreiboperationen auf den Speicher realisiert werden.

Systemaufrufe:

map (virtual address): bildet die Datei in den virtuellen Adressraum ab.

Startadresse: virtual address

unmap: entferne Datei aus dem virtuellen Adressraum.

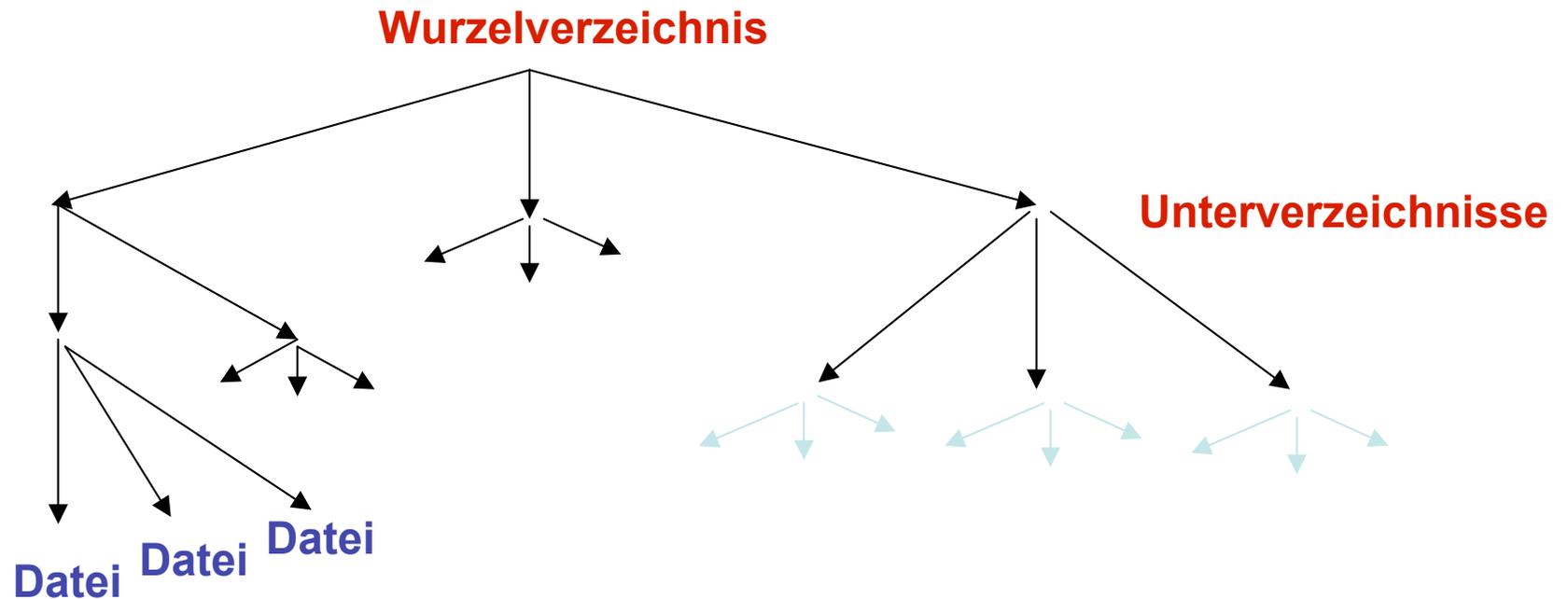
Probleme: exakte Größe der Ausgabedatei, gemeinsame Nutzung von "Memory Mapped" Dateien, Datei ist größer als der virtuelle Adressraum.



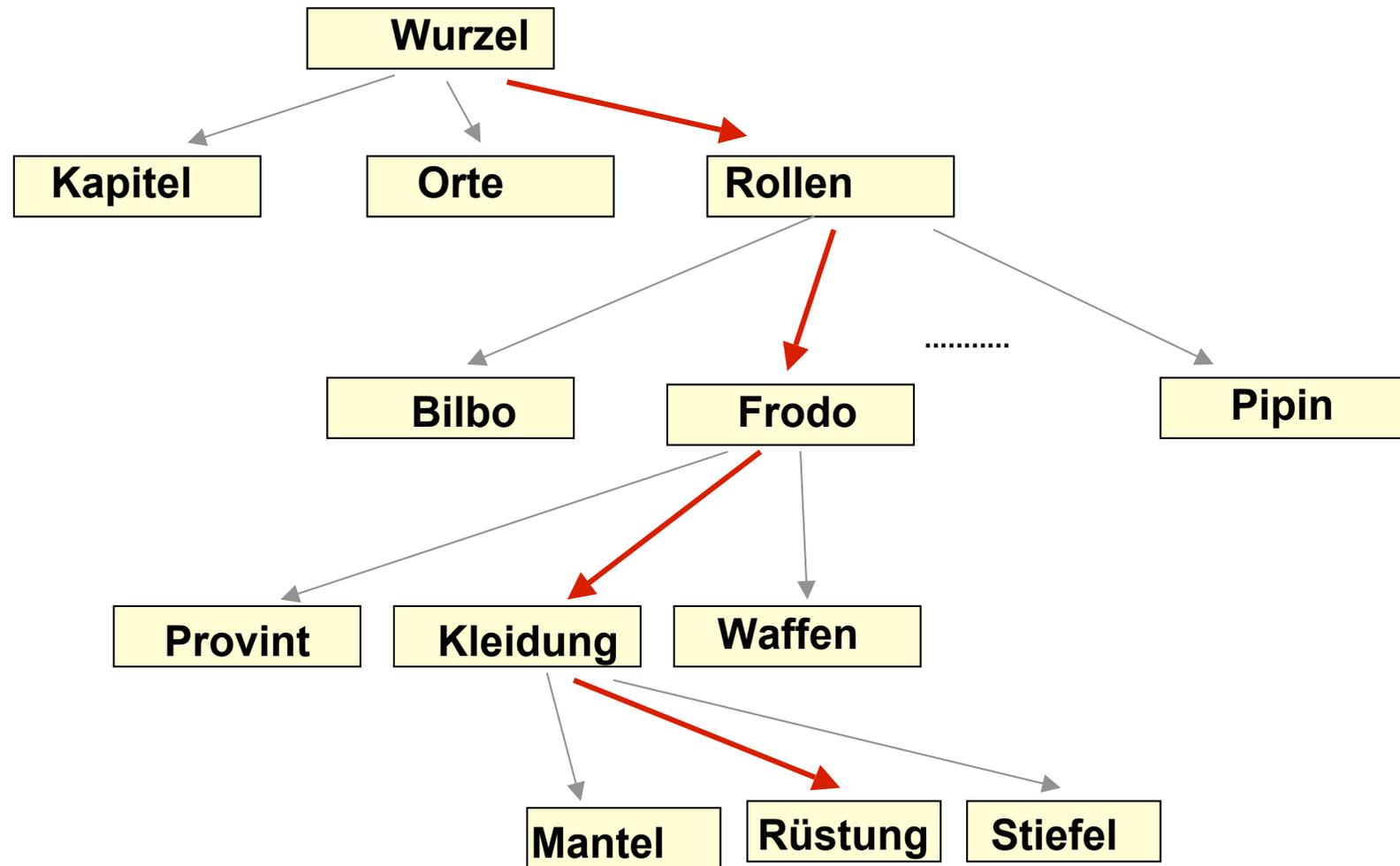
Verzeichnisse und Ordner

Flache Organisationsstruktur: Dateien werden als Sammlung einem Benutzer zugeordnet. Einfache Realisierung.

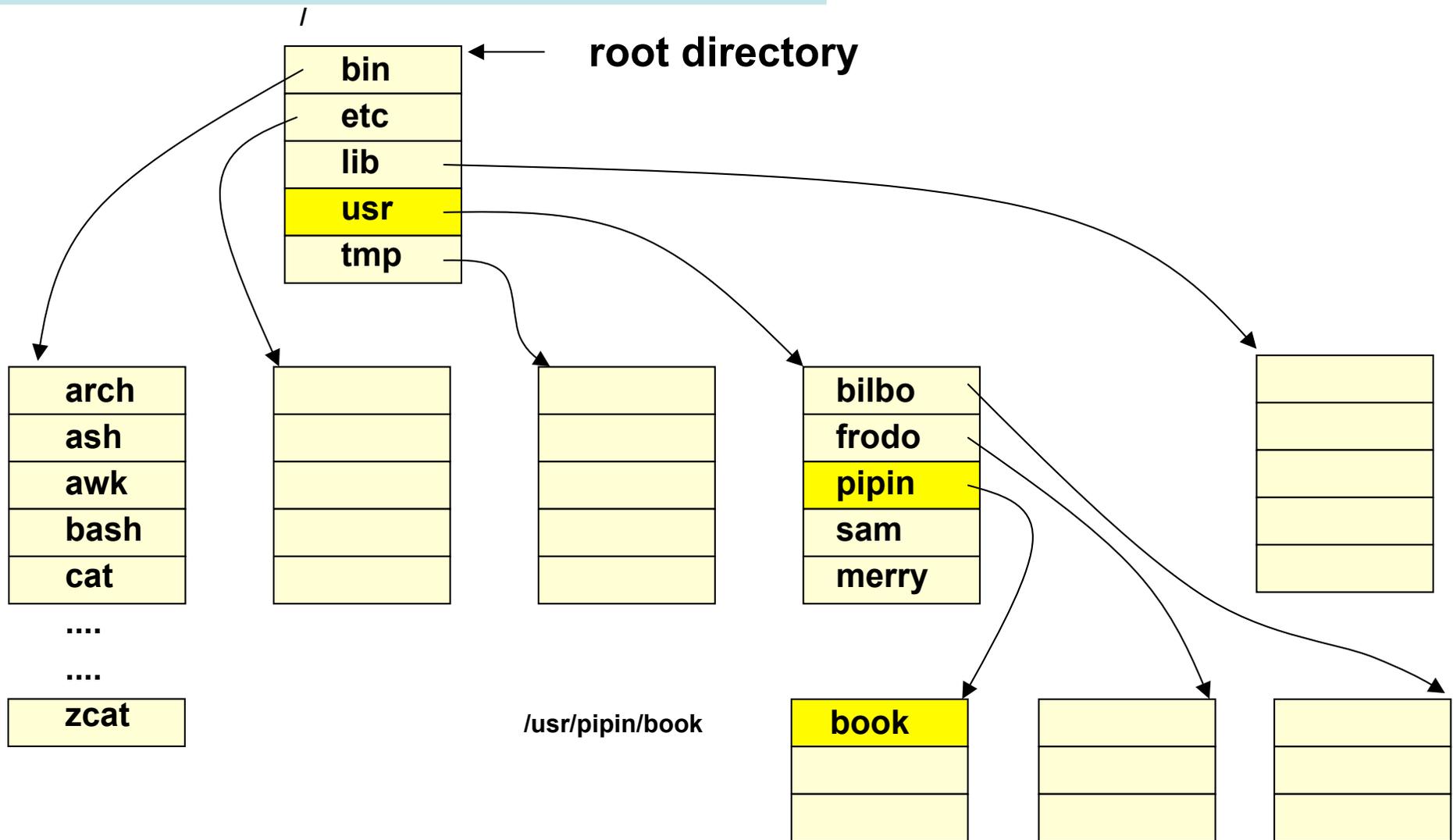
Hierarchische Verzeichnisstruktur:



Hierarchische Verzeichnisstruktur und Pfadnamen



UNIX Dateibaum



Verzeichnisse und Pfadnamen

Beispieldialog in Unix: **eingeegebene Kommandos**, **Reaktion**

```
cd /
pwd
/
ls
bin boot dev etc home lib lost+found tmp usr var
cd
pwd
/usr/kaiser
ls -all
drwxr-xr-x 14 kaiser root    4096 March 22 18:17  .
drwxr-xr-x  3 root    root    4096 Dec   11 2003  ..
-rw-----  1 kaiser  usr     742068 Nov   13 2004  pubsub-12112003.tar.gz
....
....
cd ..
pwd
/usr
```



Typische Operationen auf Verzeichnissen

- **creat(e)**
- **delete**
- **opendir**
- **closedir**
- **readdir**
- **rename**
- **link**
- **unlink**



Implementierung von Dateisystemen

Punkte:

Wie werden Dateien auf Plattenblöcke abgebildet?

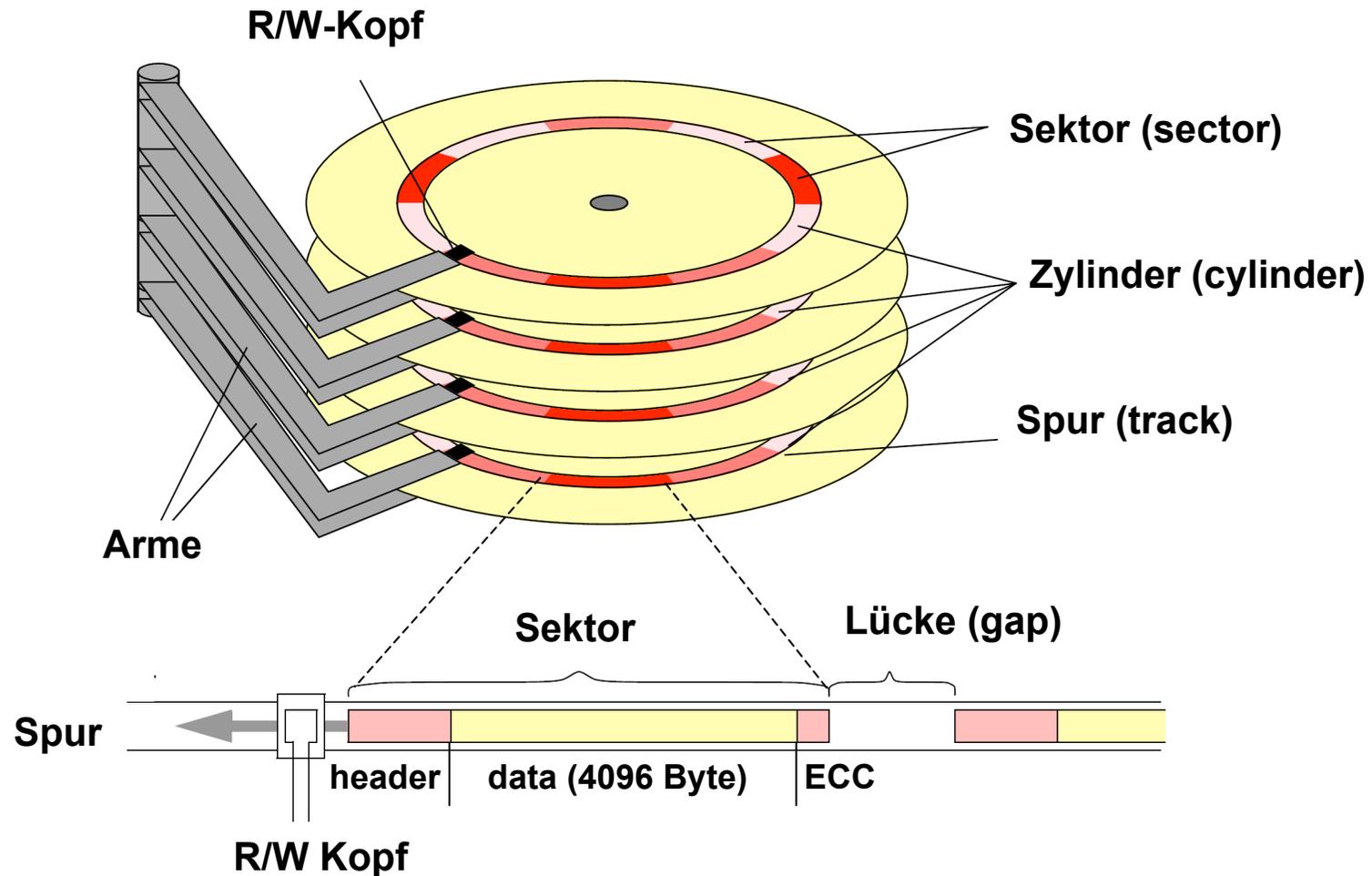
Wie werden die entsprechenden Plattenblöcke gefunden?

Wie werden Verzeichnisse realisiert?

Wie werden Dateien gemeinsam genutzt?



(Physische) Organisation einer Platte



(Physische) Eigenschaften einer Platte

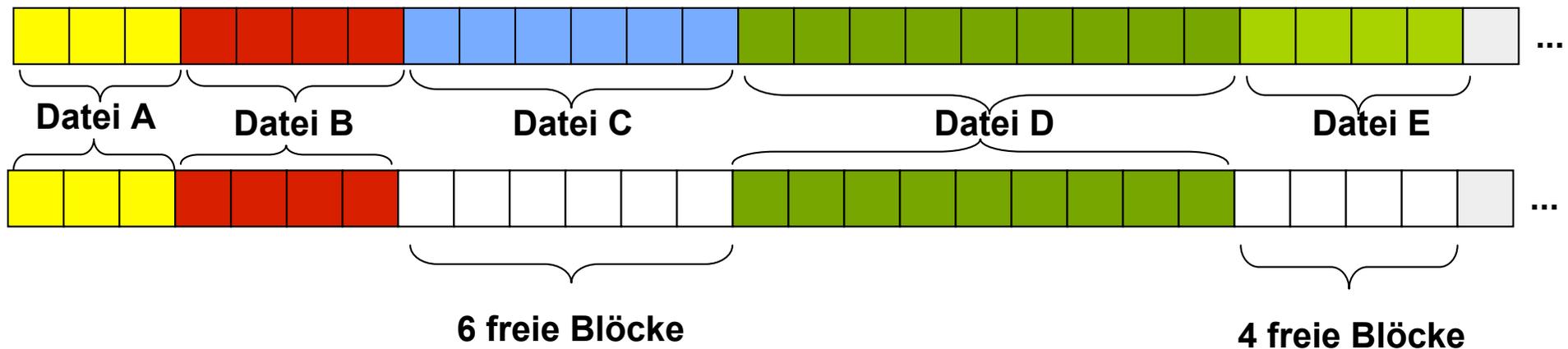
Plattentyp		Seagate ATA-U100	Seagate Cheetah
Kapazität		30 GB	73,4 GB
Platten/Köpfe		1/2	4/8
Zylinderzahl		CHS 16383/16/63	29.594
Cache		2 MB	4 MB
Positionierzeiten	Spur zu Spur		0,4/0,6 ms
	mittlere	8,9 ms	5,1/5,5 ms
	maximale		10/11 ms
Transferrate		8,5 MB/s	38–64 MB/s
Rotationsgeschw.		5.400 U/min	10.000 U/min
eine Plattenumdrehung		11 ms	6 ms
max. Stromaufnahme		7,5 W	11 W



Organisationsvariationen



Fortlaufende Belegung von Plattenblöcken



Pro: einfache Implementierung

gute Lese-Performanz

Con: Dateigröße muss a priori bekannt sein

Auffinden und Wiederverwendung von freien Blöcken ist schwierig



Organisationsvariationen



verkettete Liste realisiert durch: **File Allocation Table (FAT)** im Speicher

phys. Block Nr.	
0	
1	
2	-1
3	7
4	
5	3
6	
7	11
8	
9	-1
10	
11	9
12	
13	2
14	13
15	

-1: Dateiende symbol

← Datei A beginnt in phys. Block 5

← file B beginnt in phys, Block 14

Pro: mildert die Probleme des wahlfreien Zugriffs auf verkettete Listen

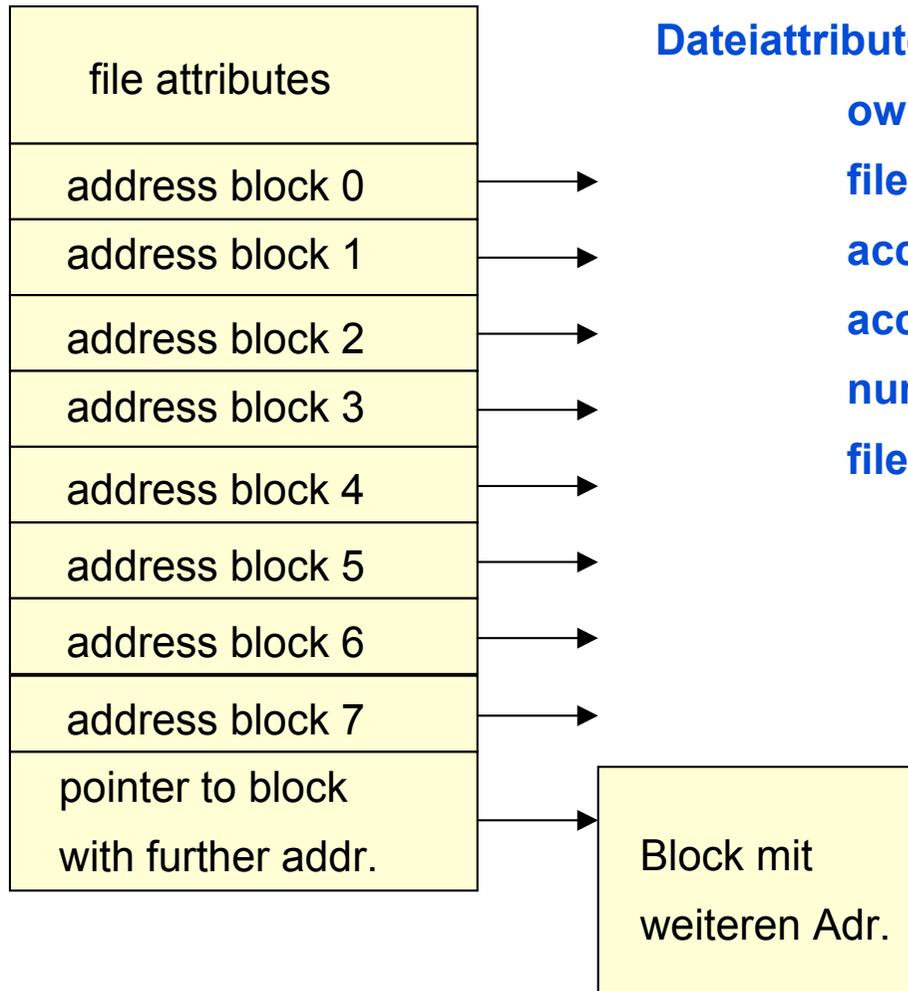
Con: die Größe ist proportional zur Plattengröße.



Organisationsvariationen



i-node, inode, index node



Dateiattribute sind z.B.:

owner

file type

access permissions

access time

number of links to the file

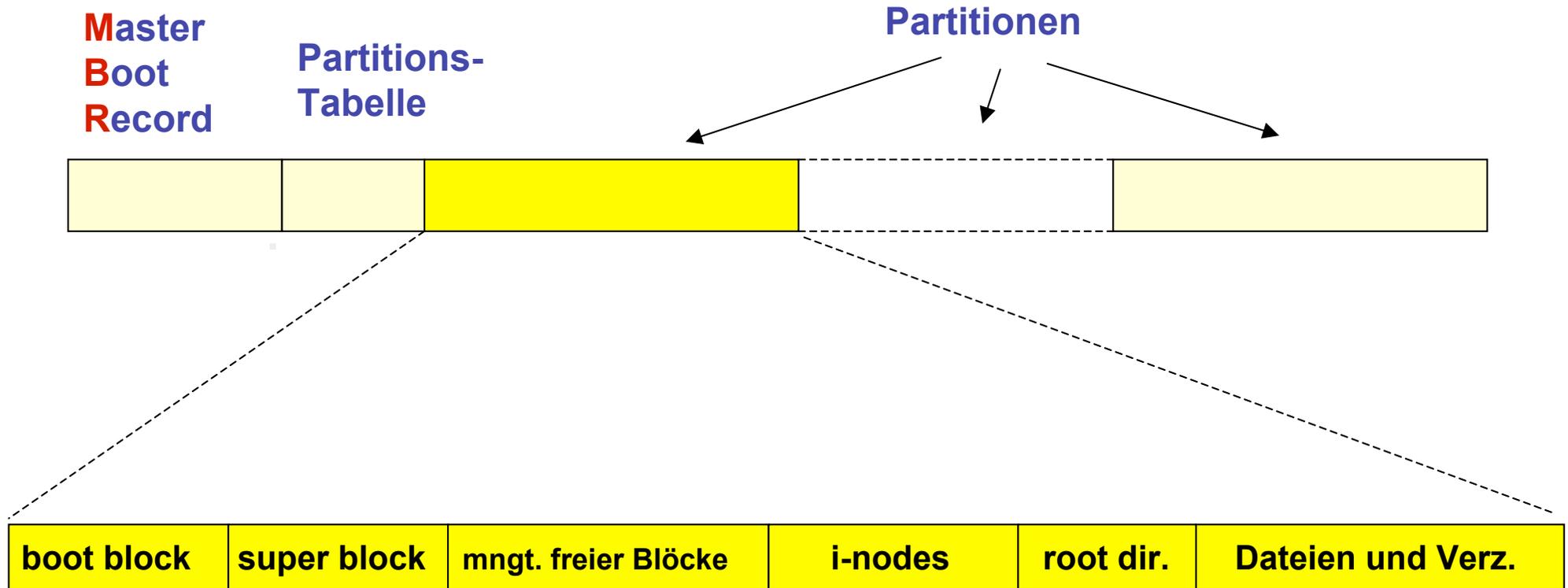
file size

pro: Nur die i-nodes offener Dateien benötigt Platz im Hauptspeicher
nicht mit Plattengröße korreliert

Dateiattribute und Dateinhalt können getrennt gespeichert werden.

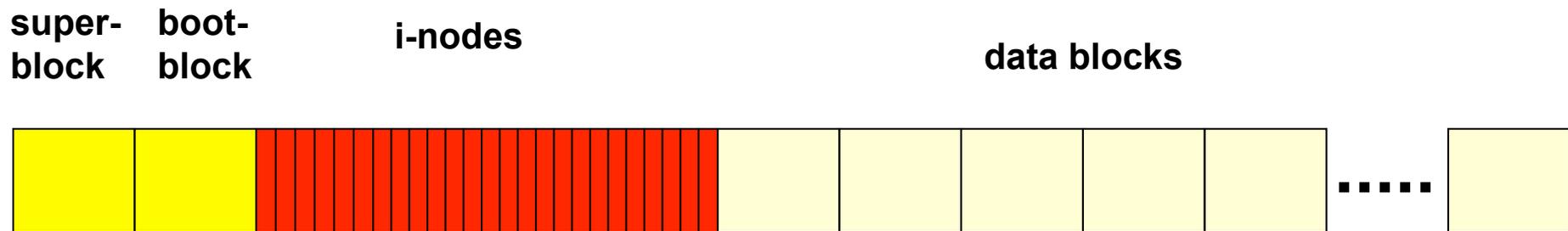


Layout eines Dateisystems



Unix Dateisystem Verwaltung

★ Klassisches Unix System



★ Berkeley Fast File System:

- lange Dateinamen (255 Zeichen)
- Platte wird in Zylindergruppen mit eigenem Super Block, i-nodes und Datenblöcken strukturiert.
- 2 Blockgrößen zur effizienten Verwaltung

★ Linux File System: sehr ähnlich zum Berkeley Fast File system.

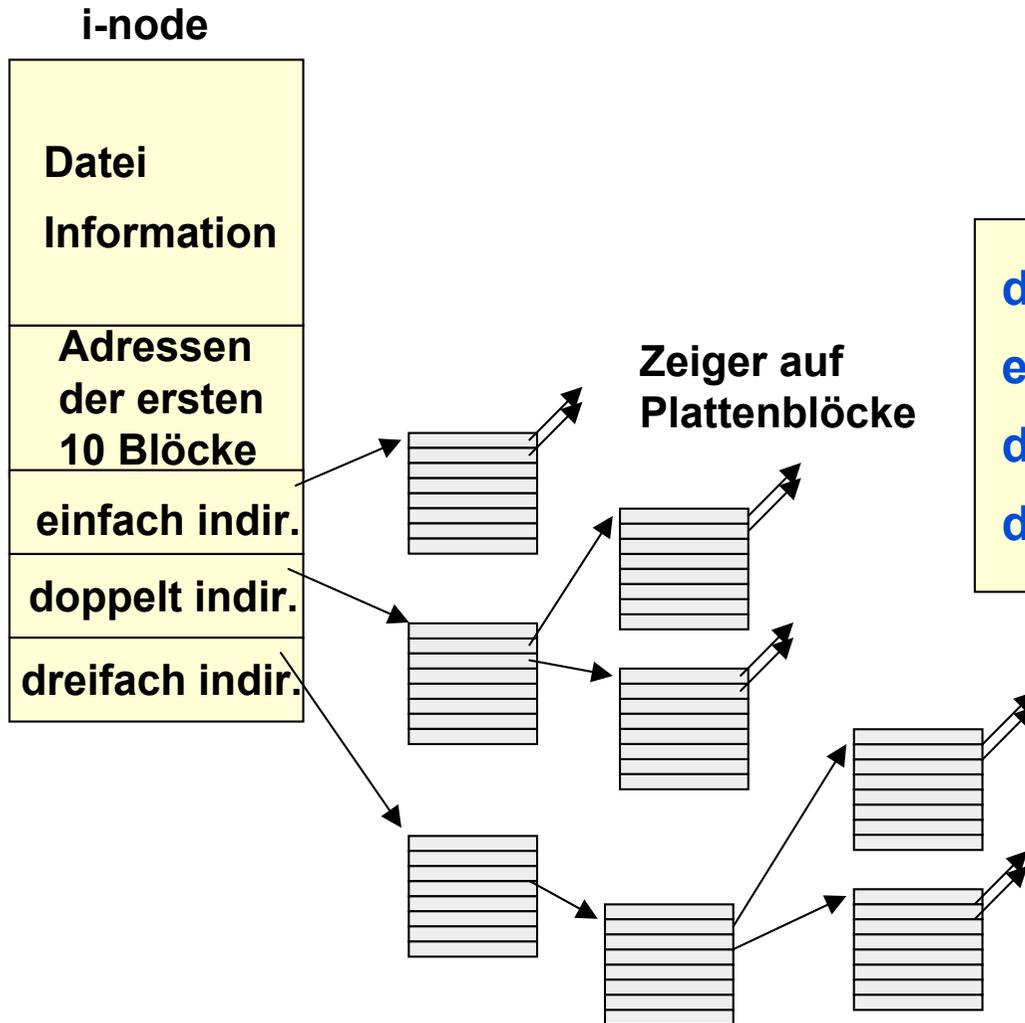


i-nodes in UNIX

File Mode:	16-Bit Flag which stores access rights 0 .. 2 rights for "all" users <read, write, exec> 3 .. 5 rights for the "group" <read, write, exec> 6 .. 8 rights for "owner" <read, write, exec> 9 ..11 execution flag 12..14 file type (regular, char./block-oriented, FIFO pipe)
Link Counter	number of directory references to this i-node
UID	Owner ID
GID	Group ID
Size	in Bytes
File address	39 byte file address information
Last access	date/time
Change of i-node	date/time
Address info for blocks	direct, single ind., double ind., triple ind.



Dateiallokation



Kapazität des UNIX Dateisystems

direkt	10 Blöcke	10 K
einfach indir.	256 Blöcke	256 K
doppelt ind.	64K Blöcke	64 M
dreifach ind.	256x64K Blöcke	16 G



Implementierung von Verzeichnissen

welche Information wird in einem Verzeichniseintrag benötigt?

Informationen über den Dateityp.

Wie wird die Datei auf der Platte gefunden?

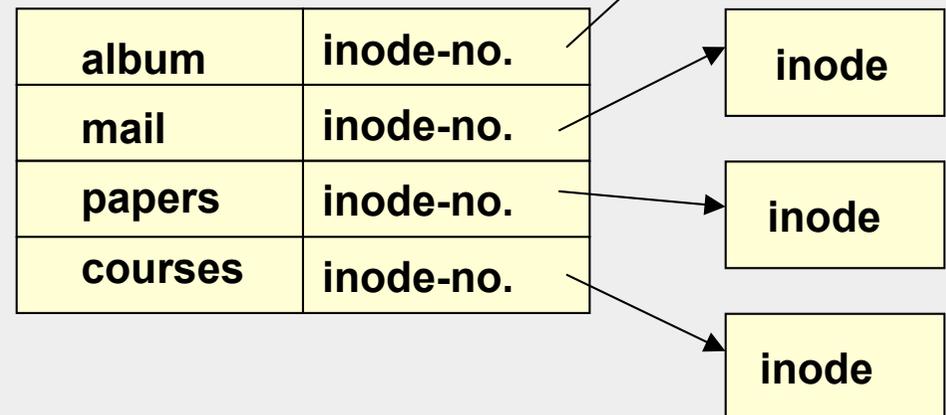
Zusätzliche information.

Dateiattribute

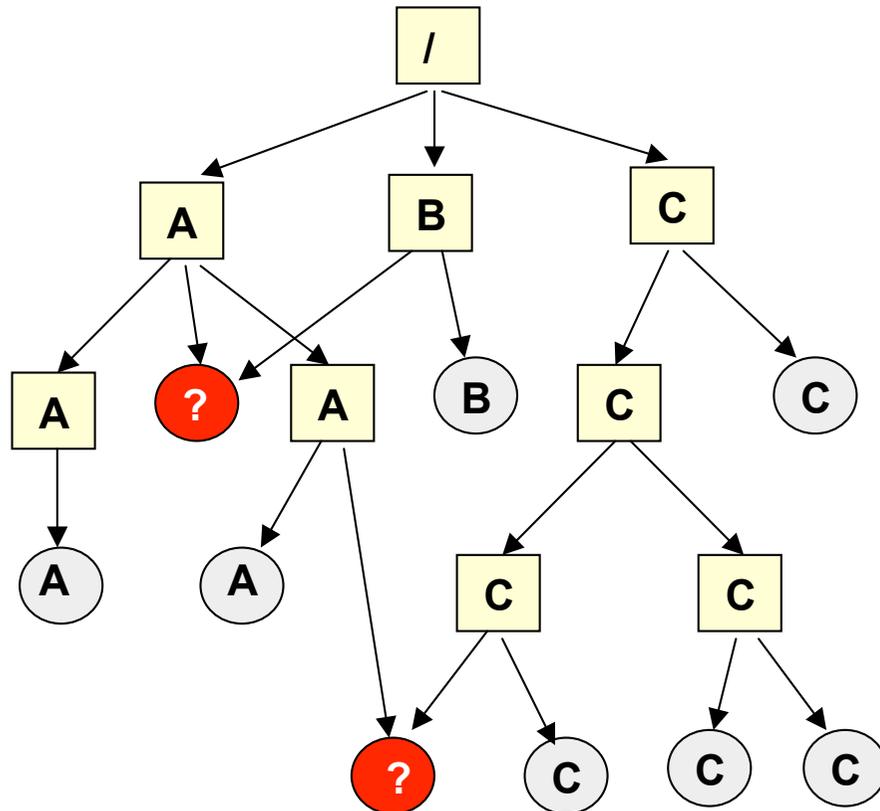
Einfache Verzeichnisstruktur mit Einträgen fester Länge

album	attributes
mail	attributes
papers	attributes
courses	attributes

Verzeichnisstruktur, die i-nodes ausnutzt.



Gemeinsame Nutzung von Dateien



Verzeichnis



Datei



gemeinsam benutzte Datei (shared file)

A,B,C : owner

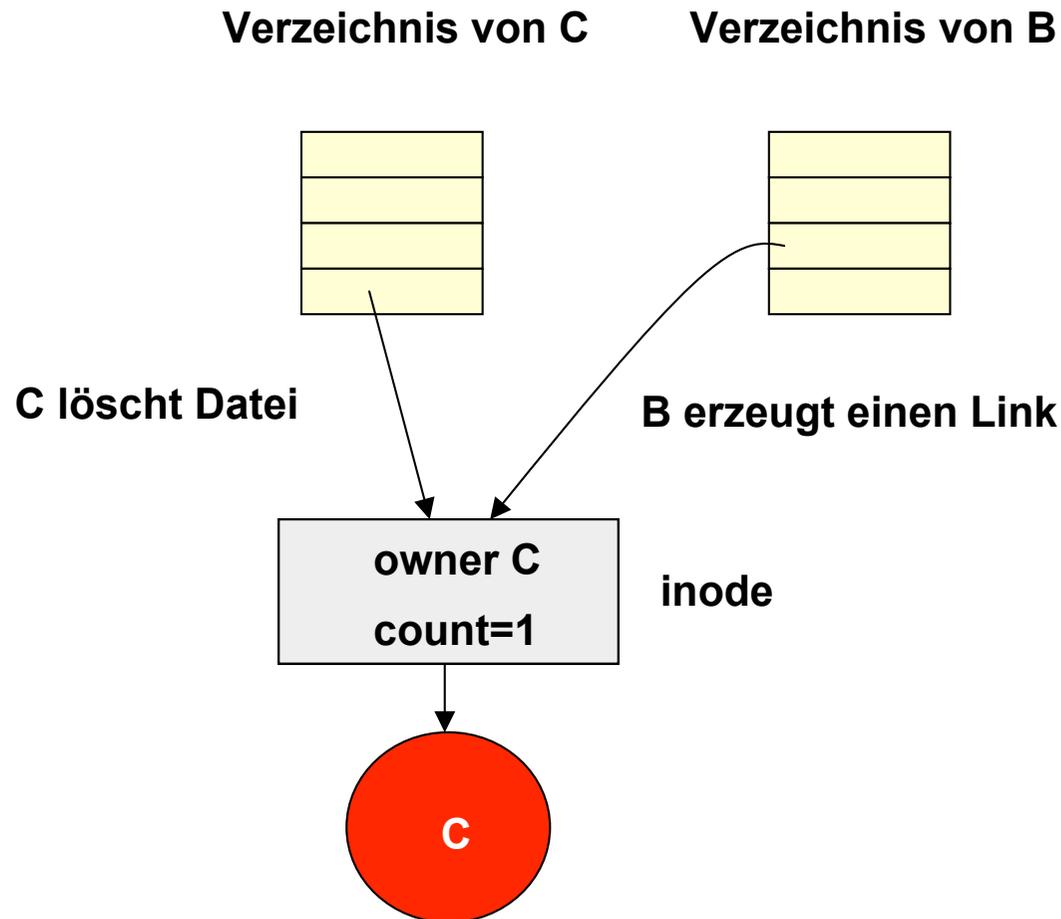
gerichteter azyklischer Graph

Probleme:

- Wer ist der Besitzer einer "shared" Datei?
- Wie wird die Sichtbarkeit von Änderungen durchgesetzt?



Gemeinsame Nutzung von Dateien

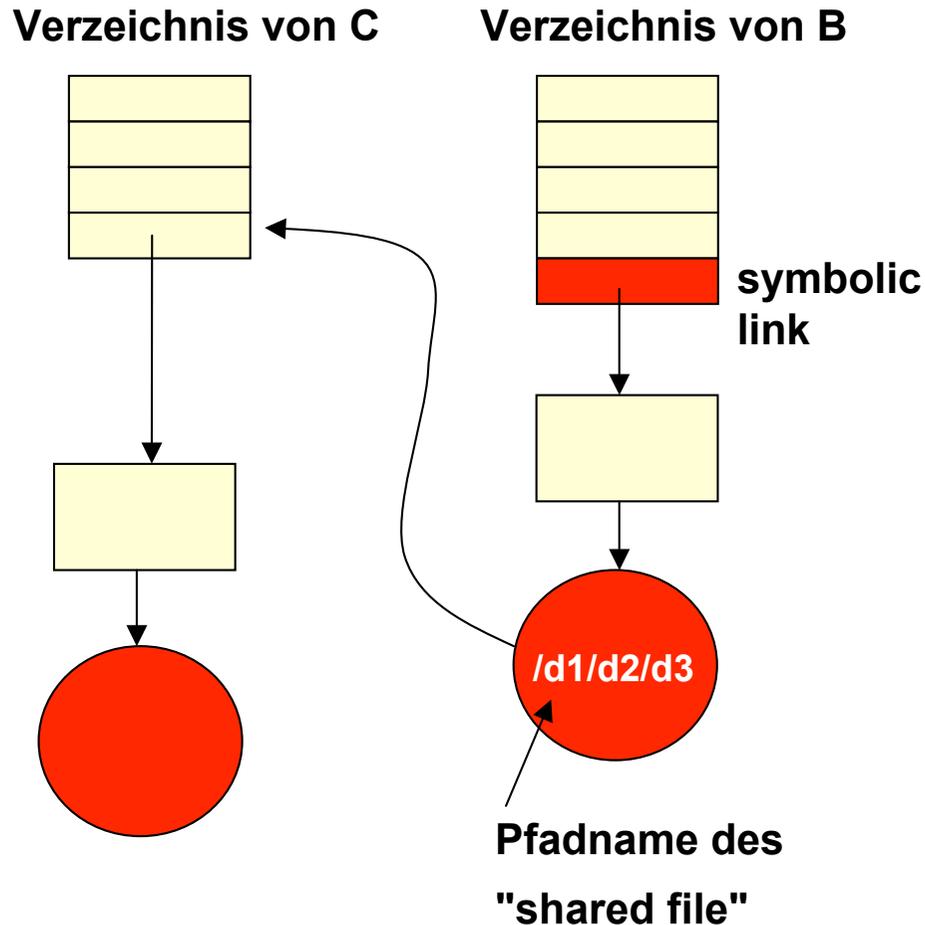


Problem:

B ist der einzige Nutzer
einer Datei deren Besitzer
C ist.



Datei-Sharing mit symbolischen Links



Besitzer hat volle Kontrolle über die Datei.

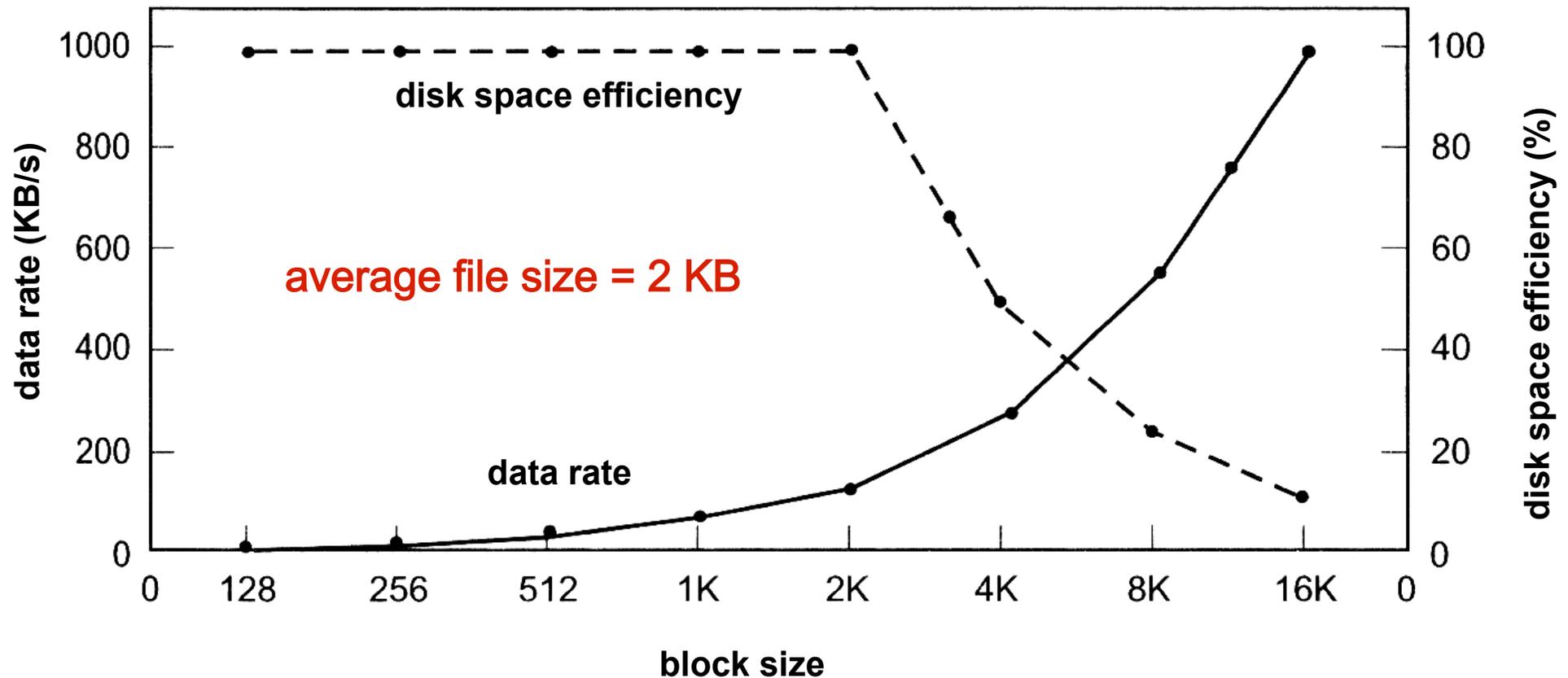
Problem: Aufwand

- Analyse und Verfolgung des Pfads benötigt zusätzlich Plattenzugriffe
- zusätzlicher i-node für jeden Link.



Verwaltung des Plattenspeichers

Einfluß der Blockgröße auf Datenrate und Speichereffizienz



(Tanenbaum 2003)



Verwaltung des Plattenspeichers

1. Verkettete Liste freier Blocks

Größe der Liste und max. Speicherplatzanforderungen:

16 GB Platte, Blockgröße 1k:

--> 16M Einträge a 32 bit

--> 1 Block 255 (+1 zur Verkettung der Blocks) Einträge --> ~ 40 K Blöcke

ändert sich mit der Zeit, wenn
mehr Blöcke benötigt werden

2. Bit-Matrix freier Blöcke

Größe der Liste und max. Speicherplatzanforderungen:

16 GB Platte, Blockgröße 1k:

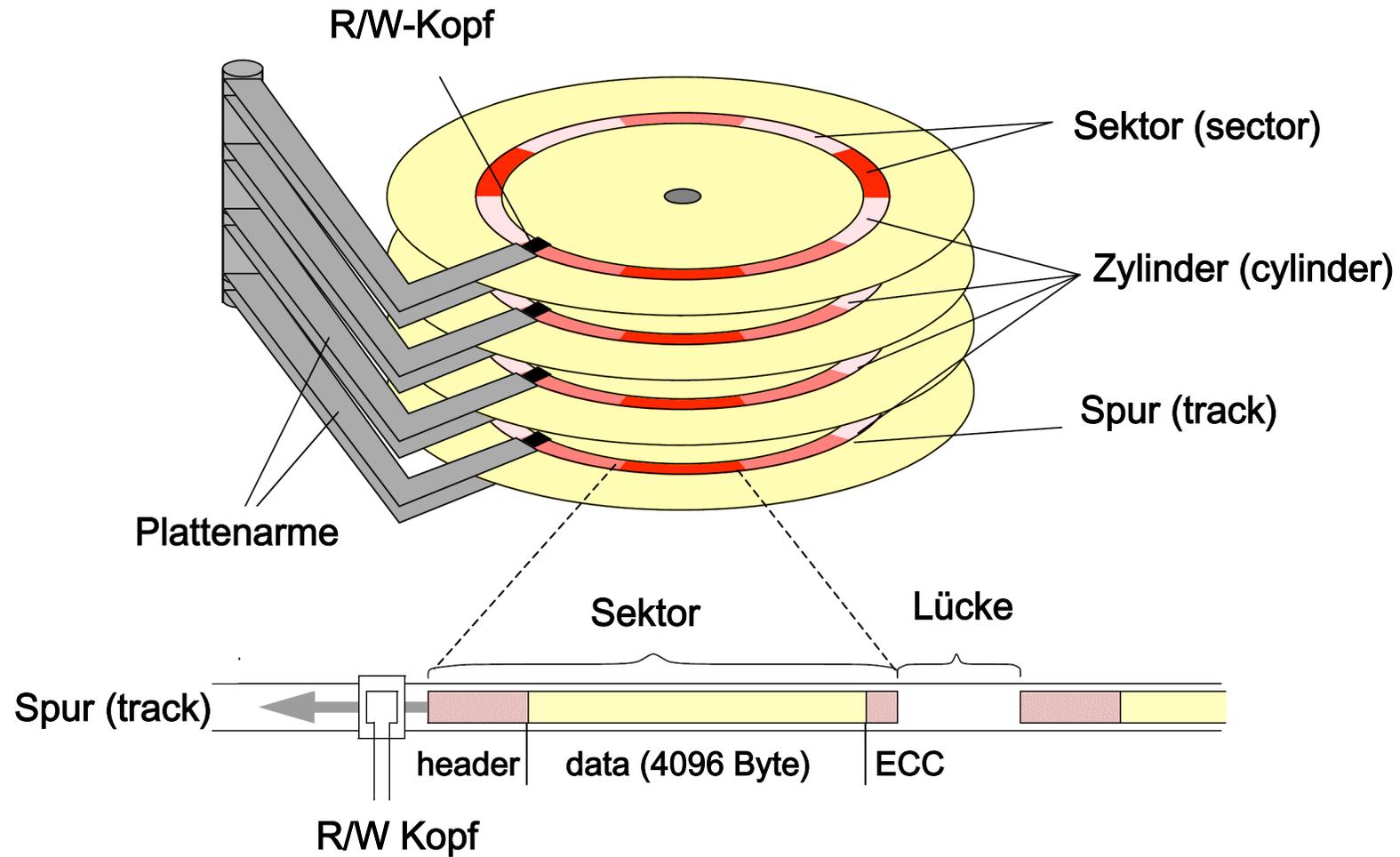
--> 16M Einträge a 1 bit

--> 1 Block 1k x 8 bBits --> ~ 2K Blöcke

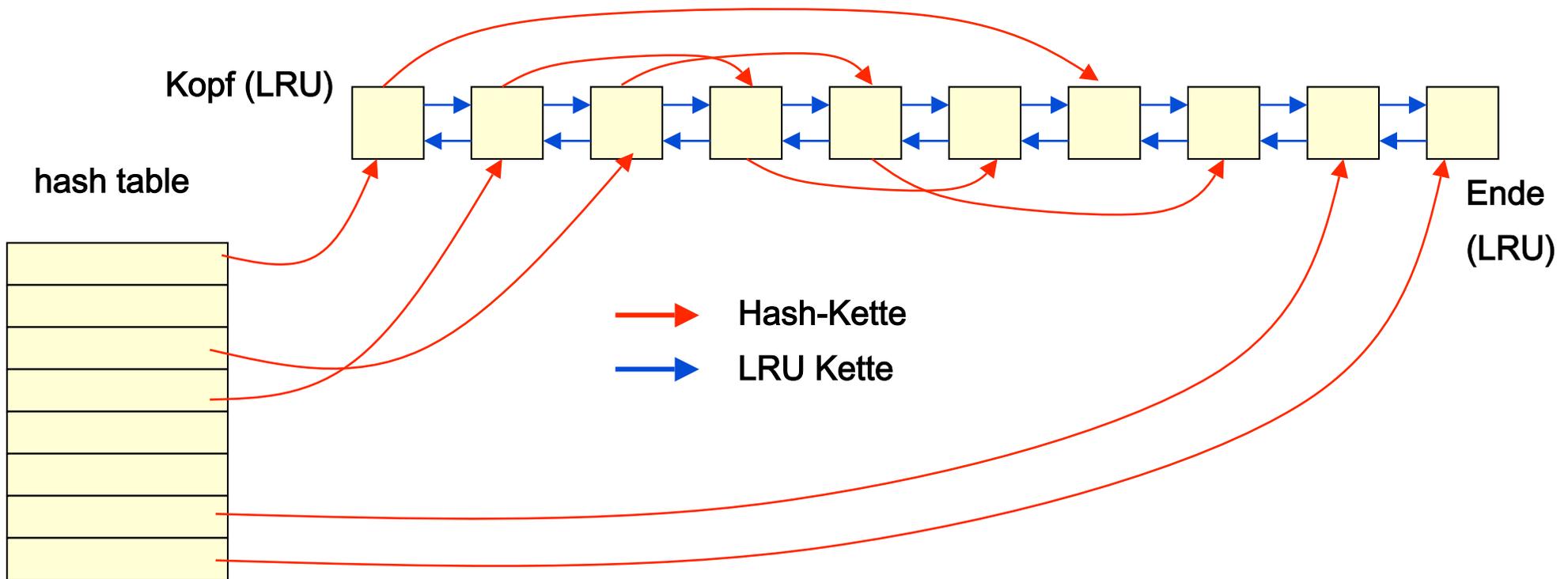
← feste Größe



Effizienzsteigerungen



Der Puffer-Cache

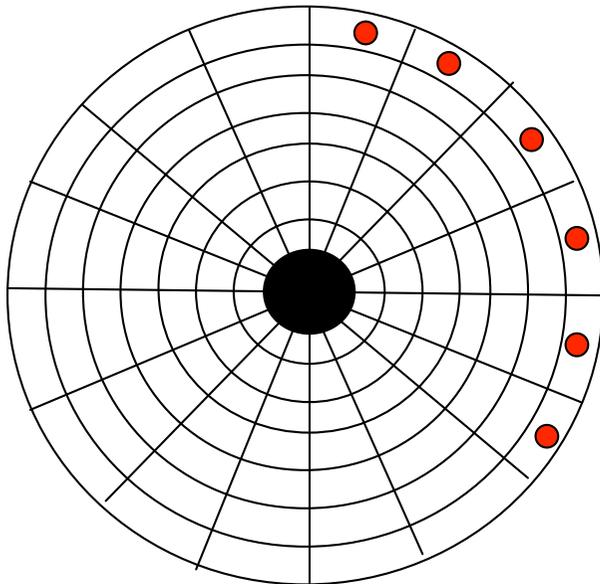


Problem: Inhalt von Plattenblöcken und Cache-Inhalt sind nicht identisch.

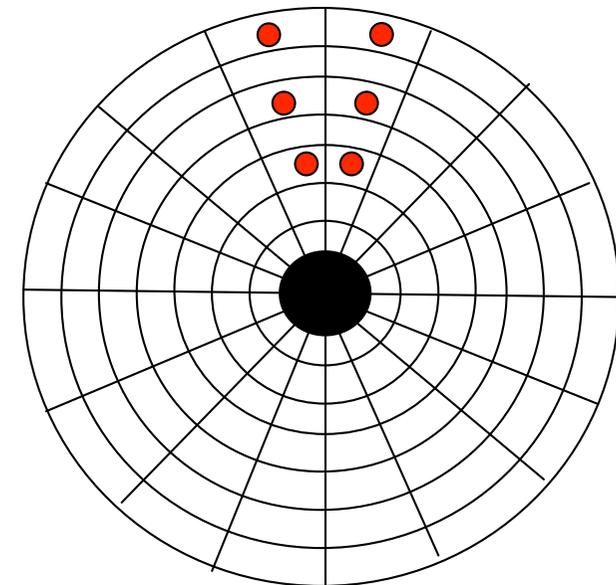
- ➔ **Inkonsistenzen bei Abstürzen.**
- ➔ **Zielkonflikt zwischen häufiger Aktualisierung und Datenverlust.**
- ➔ **Explizite Synchronisation (sync).**



Optimierung des Plattenzugriffs



i-nodes stehen alle am Anfang der Platte.
Abstand der i-nodes und der damit
assoziierten Plattenblöcke: **Anzahl der Zylinder/2**



i-nodes und assoziierte Plattenblöcke werden
in Zylindergruppen organisiert.

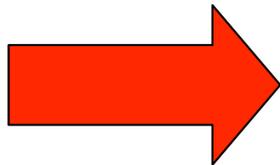


Zuverlässigkeit eines Dateisystems

Datenverlust ist der "Super GAU" in einem Computersystem!

(Hardware-) Kosten eines neuen Computers 5.000 €

Kosten von Datenverlusten: oft mehrere Größenordnungen höher !



Das Dateisystem muss geschützt werden gegen:

- **Plattencrashes**
- **fehlerhafte Software**
- **bösartige (nicht autorisierte) Zugriffe**



Zuverlässigkeit eines Dateisystems

Impairment	Countermeasures
defective blocks from manufacturing	directory of bad blocks on medium
transient reading and writing errors	code redundancy
physical destruction of disk	backup on redundant medium, mirrored disk (e.g. RAID 2), data replication
software faults	user related access rights, least privilege
system crashes	fsck, scandisk, journaled file systems
malicious accesses	access protection, encryption, fragmentation
erroneous deletion of files	no physical deletion, backups



Sicherungskopien

Physische Sicherung: kopiert alle Blöcke der Platte auf ein Sicherungsmedium

pro: einfach

con: speichert auch freie Blöcke und hat Probleme mit "bad blocks", nur vollständige Sicherung möglich.

Logische Sicherung: basiert auf der logischen Dateisystemstruktur.
Rekursiv werden Verzeichnisse und Dateien abgespeichert.

pro: Incrementeller Algorithmus speichert nur Änderungen seit der letzten Sicherung.

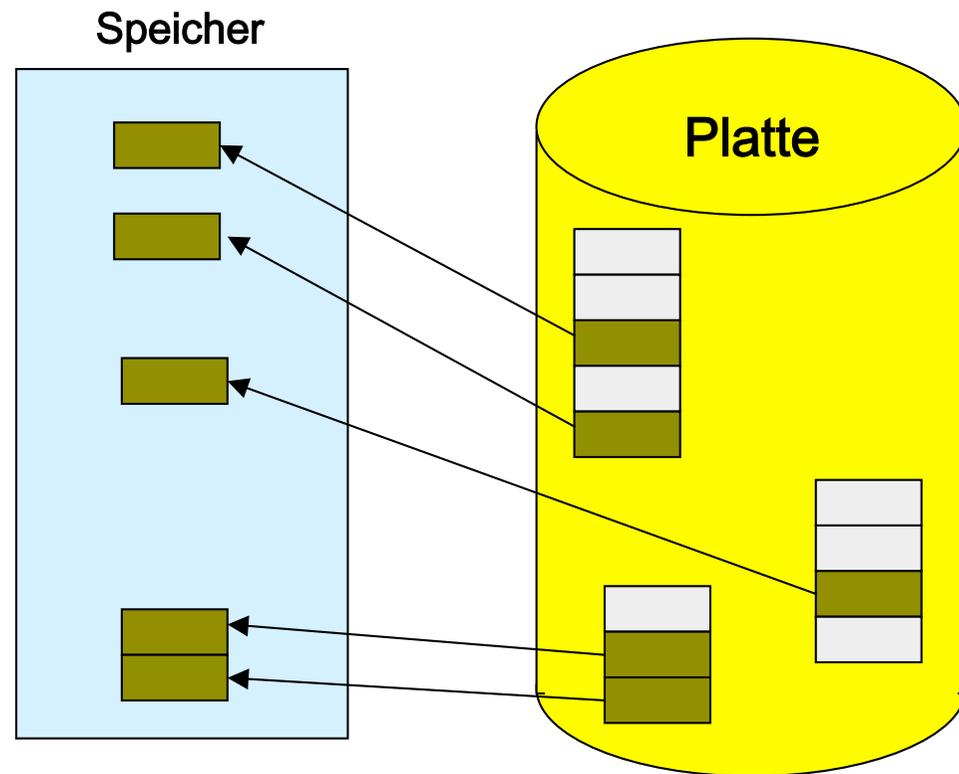
con: kompliziertere Implementierung.



Konsistenz des Dateisystems

Problem: Änderungen werden im schnellen Hauptspeicher gemacht und erst später persistent auf Platte gespeichert.

- Datei Abbildungen
(einige Dateiblöcke)
- Verzeichnis Abbildungen
(einige Verzeichnisblöcken)
- i-node Abbildungen
(einige Blöcke der i-node Tabelle)
- Abbildung der Freiliste
(einige Blöcke der Freiliste)



Konsistenz des Dateisystems

nach einem Crash...

Erstes Ziel: Erhaltung der Konsistenz von **Meta-Daten**,
d.h. alle Datenstrukturen die bei die Verwaltung
des Dateisystems benötigt werden.

Z.b. i-nodes, Verzeichnisse, Freilisten.



Ausnutzen der Redundanz bei der Organisation von Dateisystemen.

Normalerweise nicht berücksichtigt: Modifikation von Daten.

Sie sind verloren.

Berücksichtigt werden:



1. vermisste oder duplizierte Blöcke

2. Verzeichnisstrukturen



Konsistenz des Dateisystems

Vermisste oder duplizierte Blöcke: fsck

1. geht durch alle i-nodes, generiert die Liste der **benutzten** Blöcke
2. gleicht sie mit der Liste oder der Bit-Matrix freier Blöcke ab

block number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
used	1	1	1	0	0	1	1	0	0	0	1	1	2	0	0	1
free	0	0	0	0	1	0	0	1	2	1	0	0	0	1	1	0

every field counts hits

missed block: is not present in either list

duplicated block in the free list

duplicated data block



Konsistenz des Dateisystems

1. Fall: Vermisster Block

block number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
used	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	1
free	0	0	0	1	1	0	0	1	2	1	0	0	1	1	1	0

Problem: verminderte Plattenkapazität

Lösung: Vermisster Block wird der Freiliste zugeordnet



Konsistenz des Dateisystems

Fall 2: Duplizierte Blöcke in der Freiliste

block number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
used	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	1
free	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0

Lösung: Erneuter Aufbau der Freiliste und Löschen des Blocks



Konsistenz des Dateisystems

Fall 3: Duplizierter Block, d.h. der Block erscheint in mehreren Dateien

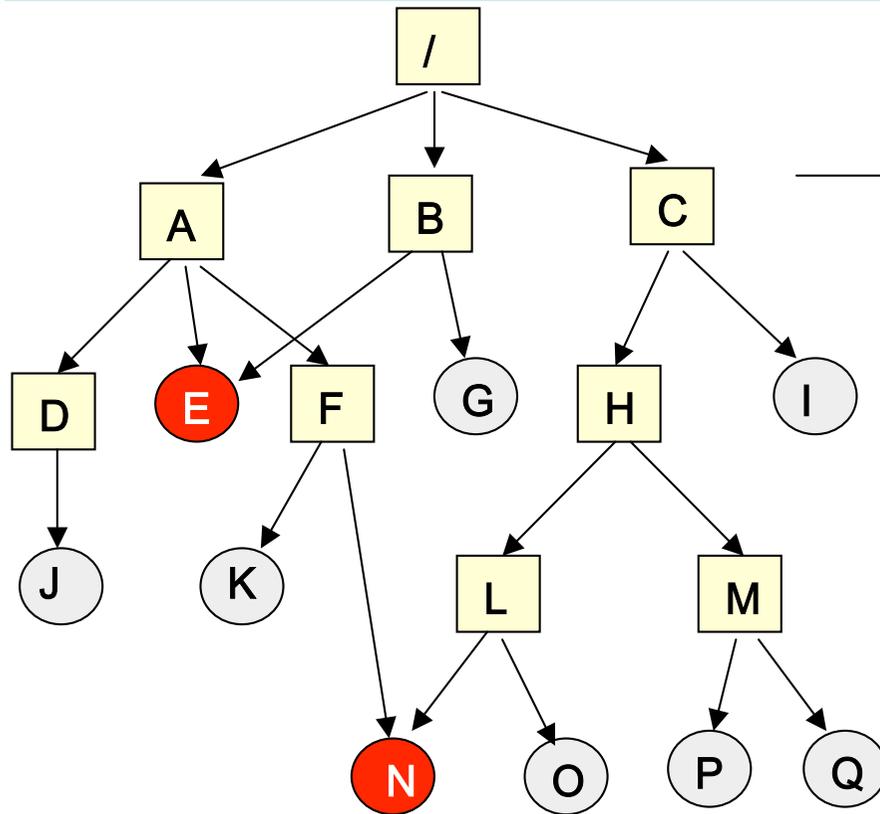
block number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
used	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	1
free	0	0	0	0	1	0	0	1	1	1	0	0	0	1	1	0

Problem: Einfaches Löschen resultiert in weiteren Inkonsistenzen.

Lösung: Kopieren eines Blocks in einen freien Block und Aktualisierung der Listen



Überprüfung des Verzechnissystems



i-node # A	count=1
i-node # B	count=1
⋮	
i-node # C	count=2
⋮	
i-node # N	count=3
⋮	
i-node # E	count=1
⋮	
i-node # Q	count=1

Zähler zu hoch

Zähler zu niedrig

1. Schritt: Durchwandern des Dateisystems. Dabei Aufbau einer Liste die durch i-node Nummern indiziert wird. Zählen des Auftretens einer Datei in jedem Verzeichnis.
2. Schritt: Vergleichen der Liste mit den Zählerständen in den i-node-Einträgen für die Dateien.



Überprüfung des Verzeichnissystems

i-node # A	count=1
i-node # B	count=1
⋮	
i-node # C	count=2
⋮	
i-node # N	count=3
⋮	
i-node # E	count=1
⋮	
i-node # Q	count=1

link Zähler im i-node
ist höher als der aktuelle Zähler in der aufgebauten Liste.

link Zähler im i-node
ist niedriger als der aktuelle Zähler in der aufgebauten Liste.

unkritisch: i-node bleibt bestehen, sogar wenn
alle Dateien des Verzeichnisses
gelöscht sind.
→ ein Effizienzproblem !

kritisch: i-node wird gelöscht, auch wenn noch ein
Verweis zu einer Datei in einem Verzeichnis besteht.
Wenn der Zähler auf "0" geht, würde der i-node als frei
gekennzeichnet und die assoziierten Blöcke werden
freigegeben.



Consistency of a file system

A consistent state of the file system has the following properties:

- The number of directory entries that point to an i-node exactly equals a link count in the i-node.
- Each disk block belongs to, at most, one file (one pointer in an i-node or in an indirect block).
- Each block is contained exactly once in either the list of free blocks or the list of used blocks.



Problems with recovery in large file systems

The system must scan all of the meta-data structures of the entire file system on disk to restore a consistent state. Thus, recovery time is related to **file system size**.

File systems grow dramatically and hence recovery time reaches the order of hours (or even days).

Idea: Relate the recovery effort to the last few **operations before the crash**



which may have caused an inconsistent state.

Consequence: We have to know which operations occurred before a crash.
Need a **logging** facility.



Log structured file systems

Motivation:

CPU performance
disk capacity
main memory capacity

} grow rapidly

Problem: disk access time doesn't improve much (seek ~10ms, wait ~4ms, write 50 μ s).

- ➔ read accesses can be optimized through caching.
- ➔ write accesses will be the most frequent operation (to disk!).
- ➔ write acces to disk becomes a substantial bottleneck.



idea: collect all changes to disk blocks and write them in a single segment to disk.
The resulting data structure is called a "log".



Journaling (logging) file system

Journaling file systems use data base techniques to secure sequences of operations:

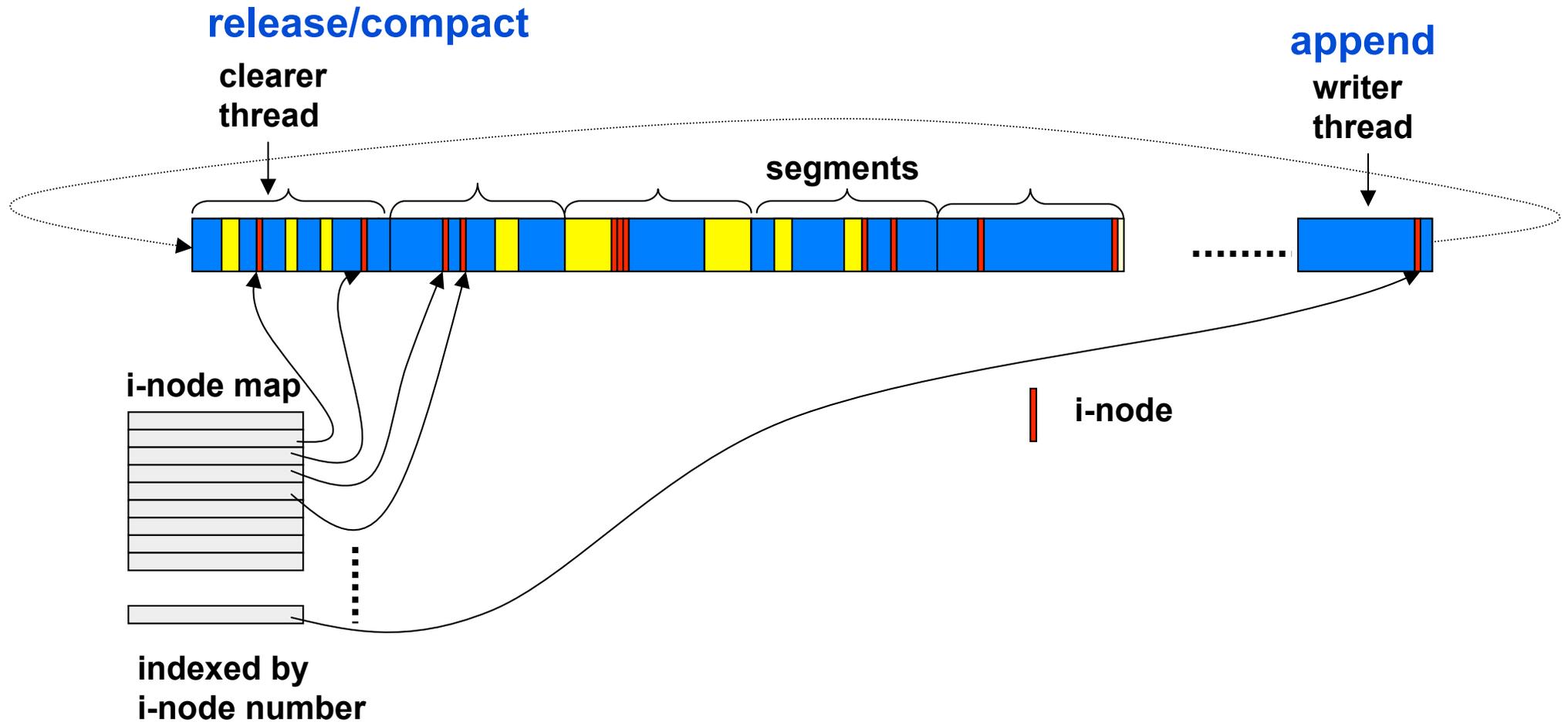
Motivation: Long recovery times (log operations on meta-data)
 Data loss (log operations on all data)

- all changes on metadata are written to a serial log,
- a serial log is a persistent data structure which survives crashes,
- efficiency can be traded against data loss,
- usually only meta-data are written to the log,
- recovery effort is related to the amount of log data rather than to total file system size.

Examples: IBM JFS, Veritas, Sprite LFS, MAC OS X, XFS (Open Source, developed by Silicon Graphics)



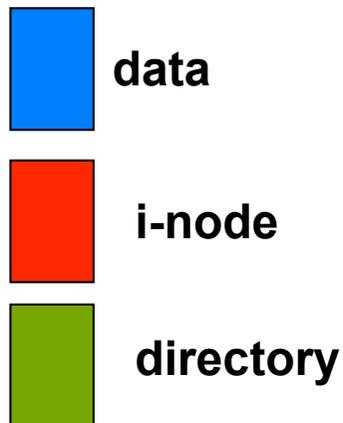
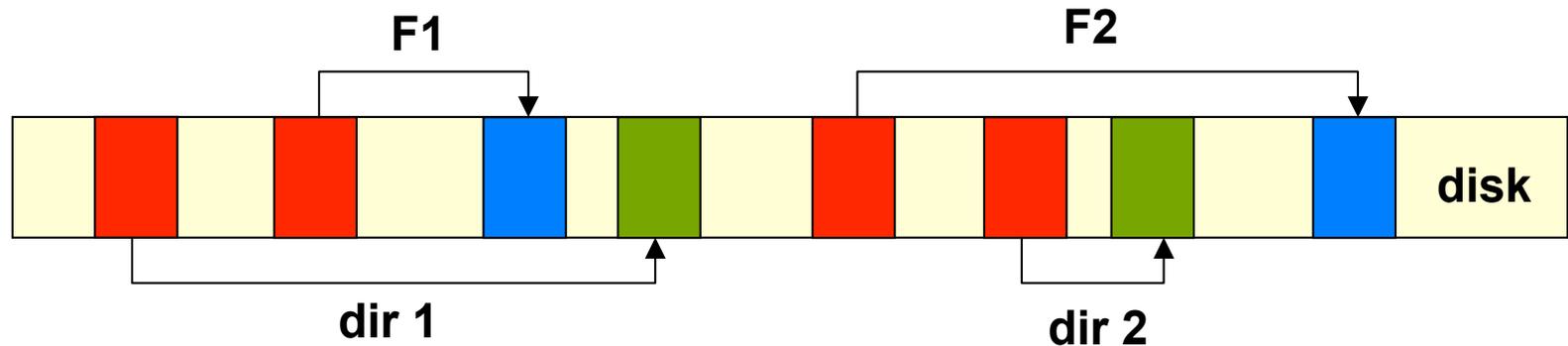
Log-structured File System (LFS)



Log structured file systems

Creating files in a conventional file system (FFS):

creating:
/dir/F1 and
/dir/F2

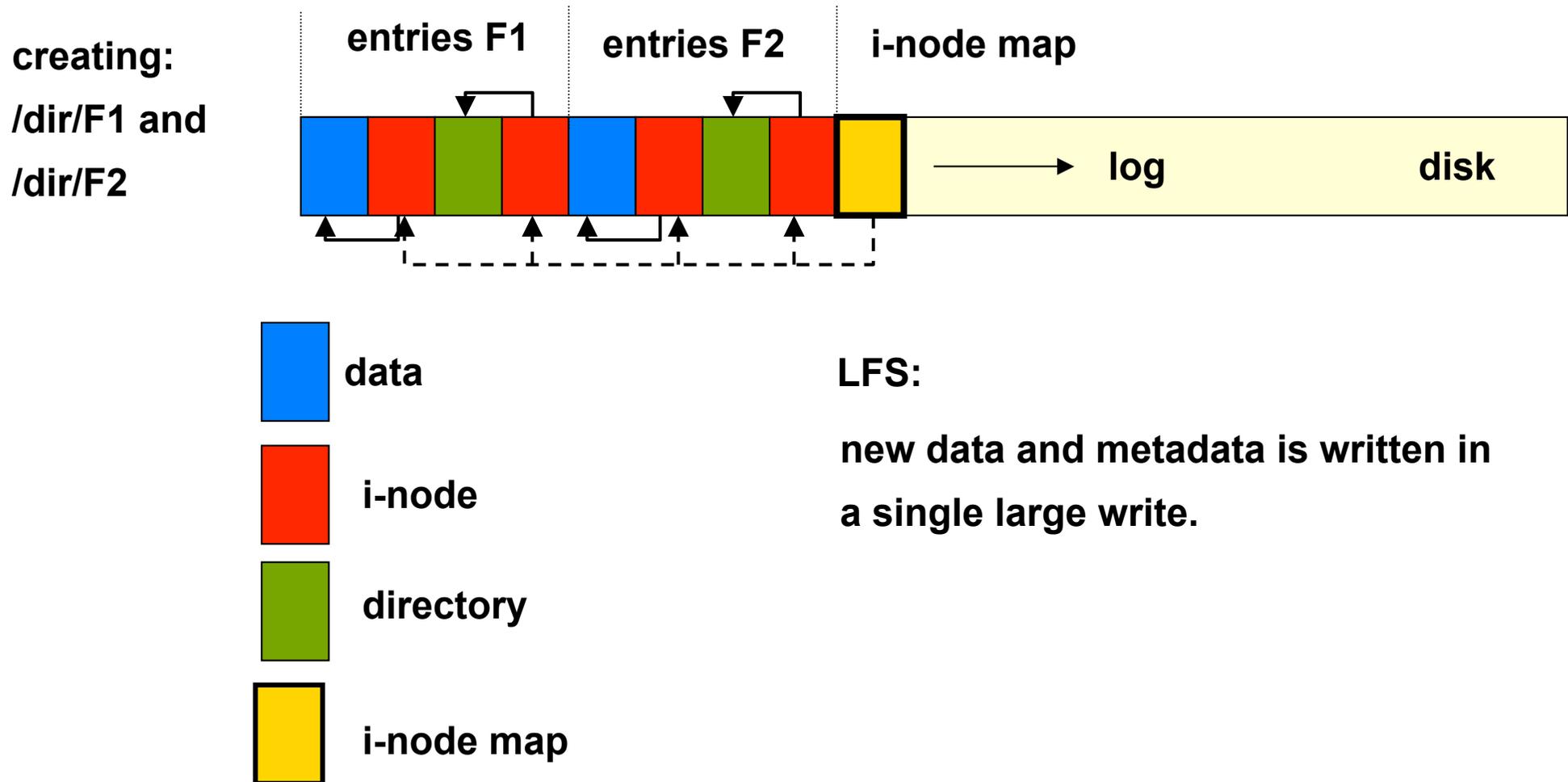


FFS:

**FFS requires 10 non-sequential writes
preceded by a seek. (I-nodes for new
files are written twice to ease recovery)**



Log structured file systems



Log structured file systems

Traditional file systems:

achieve **LOGICAL LOCALITY** assuming certain access patterns and organize information optimally for certain read patterns. **LOGICAL LOCALITY** pays in the writing process to organize information appropriately.

Log-structured file systems:

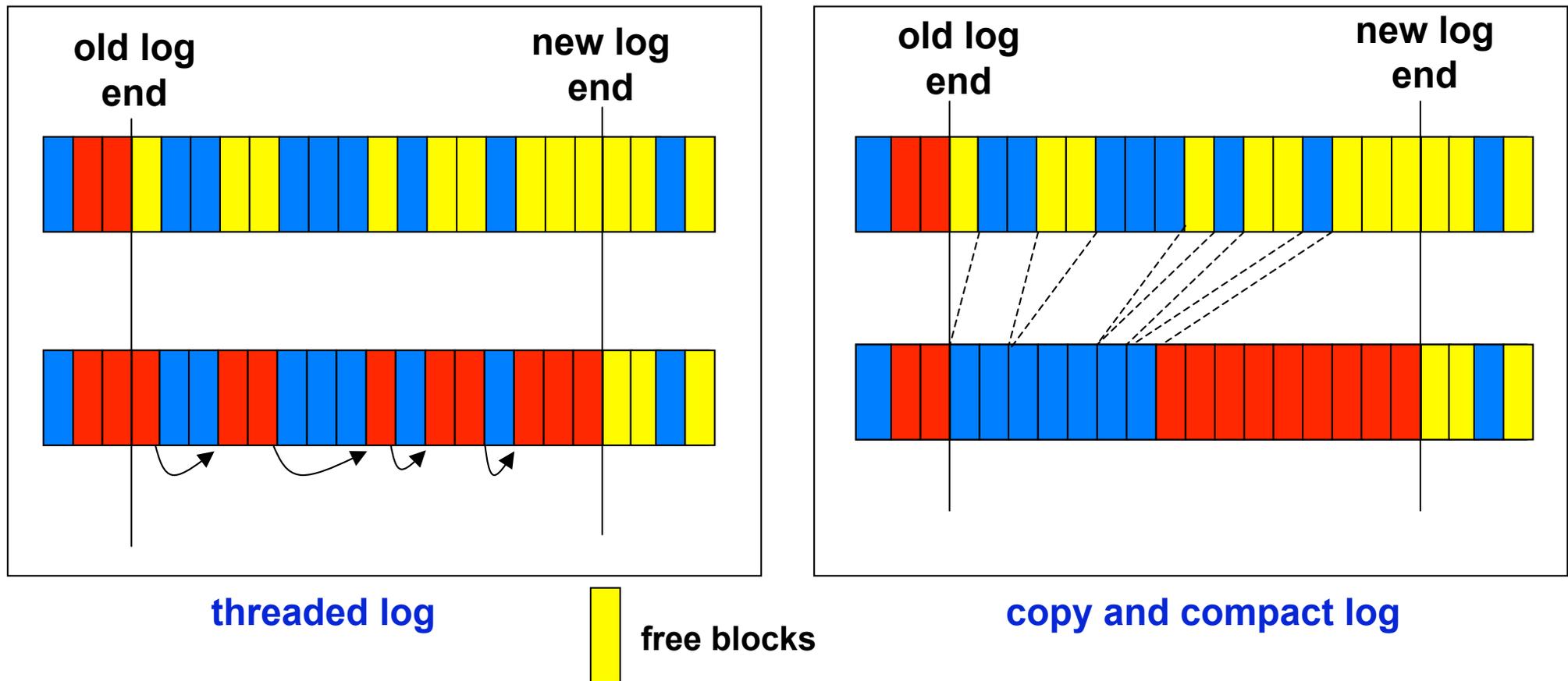
achieve **TEMPORAL LOCALITY** info which is created or modified at the same time is grouped closely on disk. **TEMPORAL LOCALITY** pays in the reading process. This however can be moderated by caching.



Log structured file systems

Problem: No infinite disk! → Free space management is needed.

Free space management alternatives for log-structured file systems:



Log structured file systems

Problems with "Threading":

Over time the log becomes fragmented and the benefits are lost

Problems with "Copy and Compact" in a circular log:

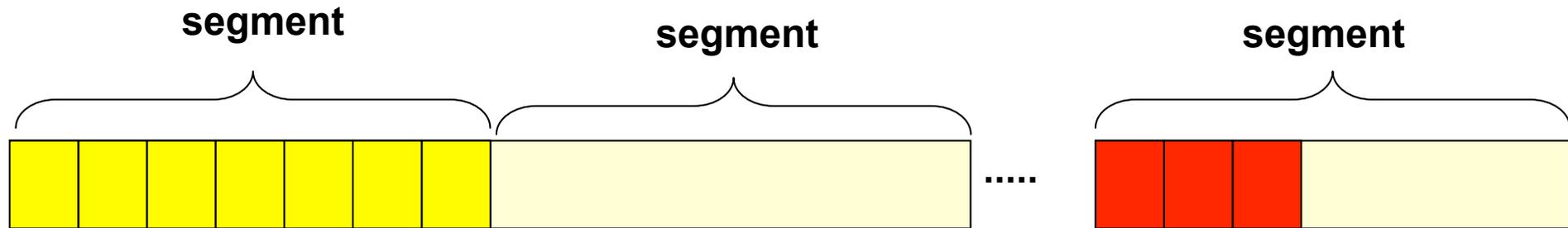
Long-lived files have to be copied in every pass of the log across the disk.



Combine threading and copying.



Log-structured File System (LFS)



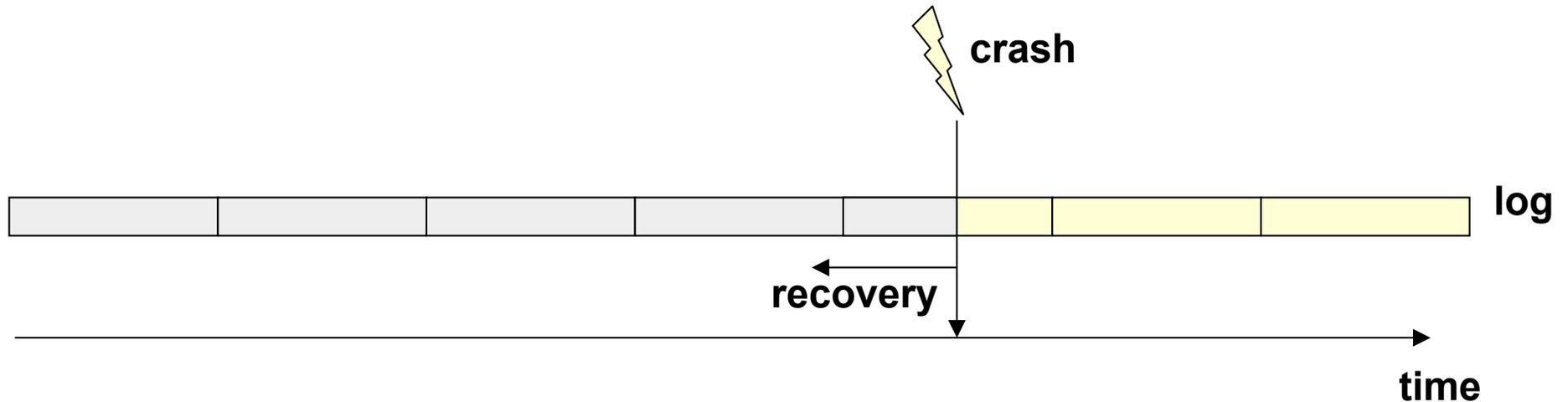
- segments:**
- large number of fixed size contiguous blocks
 - transfer time for read and write of the whole segment is large compared to the cost of a seek to the beginning of the segment. (LFS segment size 512k or 1M)

Segmented structure allows a combination of threading and copying.

- ➔ All segments are written sequentially from the beginning to the end.
- ➔ Before a segment can be rewritten all "live" data must be copied out
- ➔ Long-lived data is collected together in segments which are skipped over.



Recovery in LFS

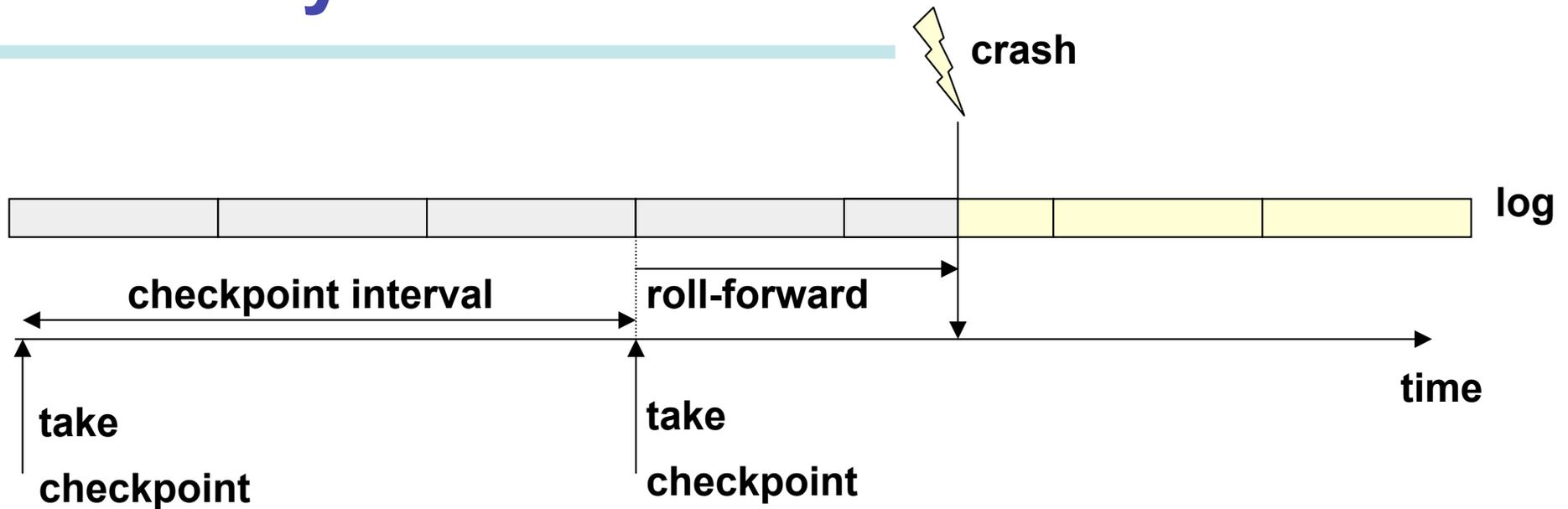


**find most recent operations
which may have left the file
system in an inconsistent state**

Problem: How far to go back?



Recovery in LFS



- ➔ checkpoint regions are kept in special fixed positions on disk
- ➔ checkpoint region contains:
 - ➔ addr. of all blocks in the i-node map
 - ➔ segment usage tables
 - ➔ current time and pointer to last segment written



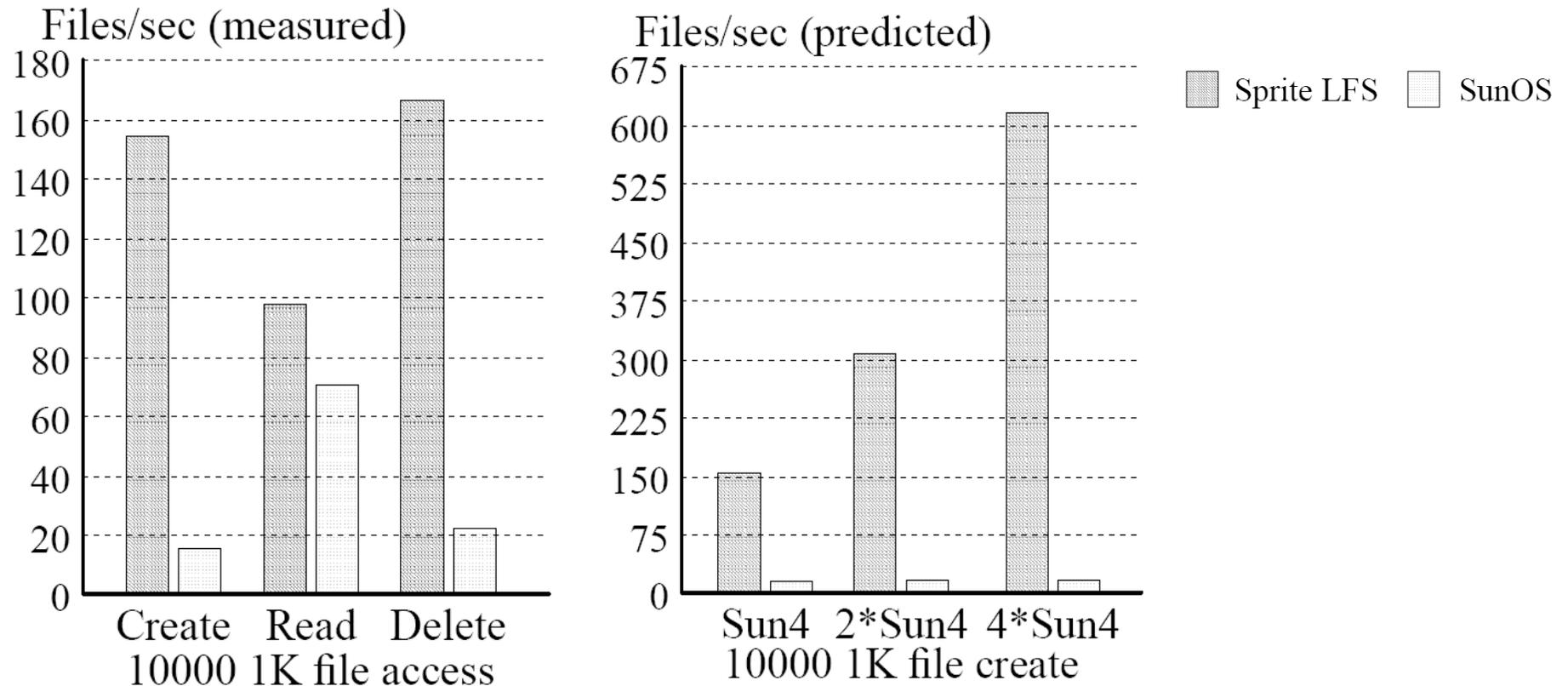
Log-structured File System (LFS)

- ➔ **segments are written periodically or on demand**
- ➔ **more overhead for finding information**
- ➔ **much better performance than regular UNIX file system on writing small amounts of data**
- ➔ **better or similar as ordinary UNIX file system for reads and writing large portions of data**



Log-structured File System (LFS)

performance comparison: small file performance

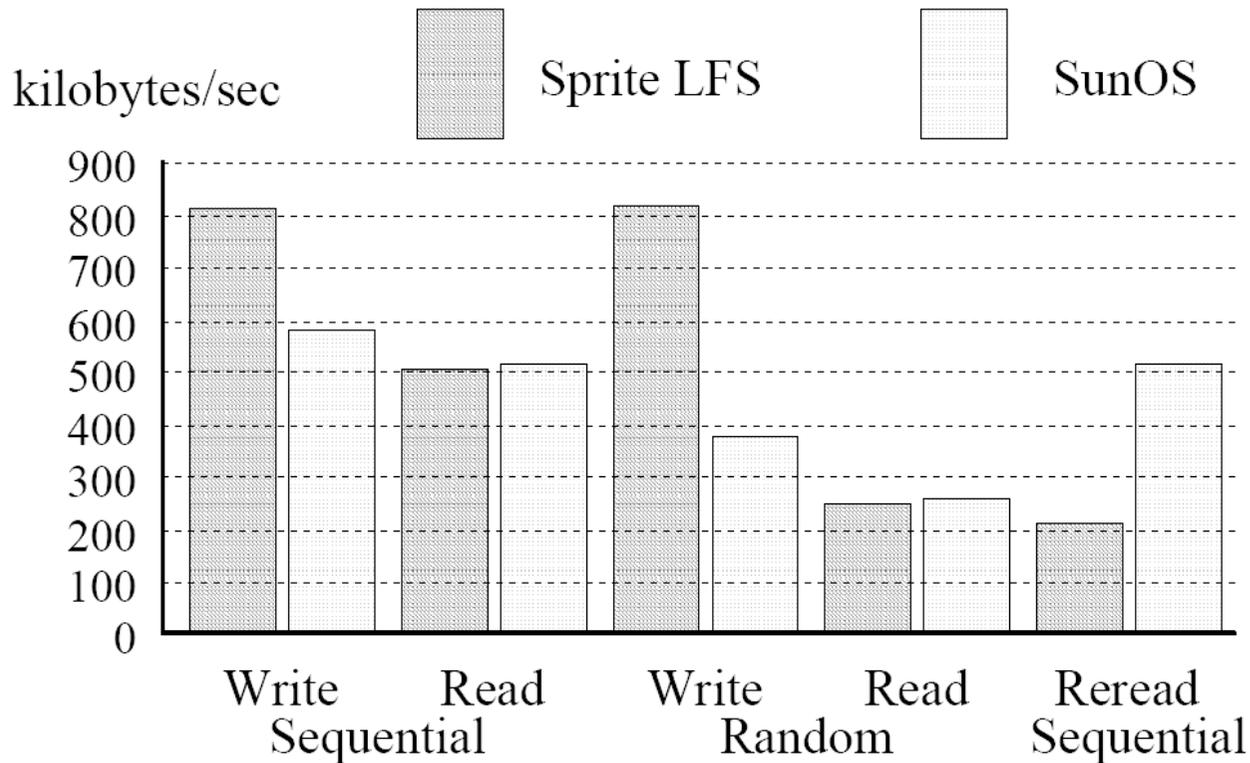


Mendel Rosenblum, John K. Ousterhout: The Design and Implementation of a Log-Structured File System, ACM Transactions on Computer Systems, 1991



Log-structured File System (LFS)

performance comparison: large file performance



Mendel Rosenblum, John K. Ousterhout: The Design and Implementation of a Log-Structured File System, ACM Transactions on Computer Systems, 1991



Characteristics of Journaling File Systems

	Ext3	ReiserFS	XFS	JFS	OSX
Largest block size supported	4 Kb	4 Kb	4 Kb	4 Kb	32 Kb
File size maximum	2 Tb	1 Eb	9 EB	4 Pb	16 Tb
Growing the file system size	Patch	Yes	Yes	Yes	No
Access Control Lists	Patch	No	Yes	Yes*	No
Dynamic disk inode allocation	No	Yes	Yes	Yes	Yes
Data logging	Yes	No	No	No	No
Place log on an external device	Yes	Yes	Yes	Yes	No

Tb = Terabyte, or 1024 Gigabytes = 10^{12} bytes

Pb = Petabyte, or 10^{15} bytes,

Eb = Exabyte or 10^{18} bytes

From: http://www.backupbook.com/03Freezes_and_Crashes/02Journaling.html



Lernziele



Allgemeine Struktur eines Dateisystems

- Organisation der Dateien
- Organisation der Verzeichnisse
- Zugriff zu Dateien und Verzeichnissen



Organisation der Platte

- Blockstruktur der Platte
- Abbildung von Dateien und Verzeichnissen
- gemeinsame Nutzung von Dateien



Verwaltung der Platte auf Blockebene



Verbessern der Leistung von Dateisystemen



Zuverlässigkeit und Konsistenz von Dateisystemen

