Operating Systems II

File Systems

File Systems: Motivation

Why do we need another sort of memory

Persistence ?

Sharing ?

Protection ?

Size ?

File System Issues

- General structure of a file system
  - organization of files
  - organization of directories
  - accessing files and directories

- Organization of the disk
  - block structure of the disk
  - mapping files and directories on blocks
  - sharing files

Operating Systems I

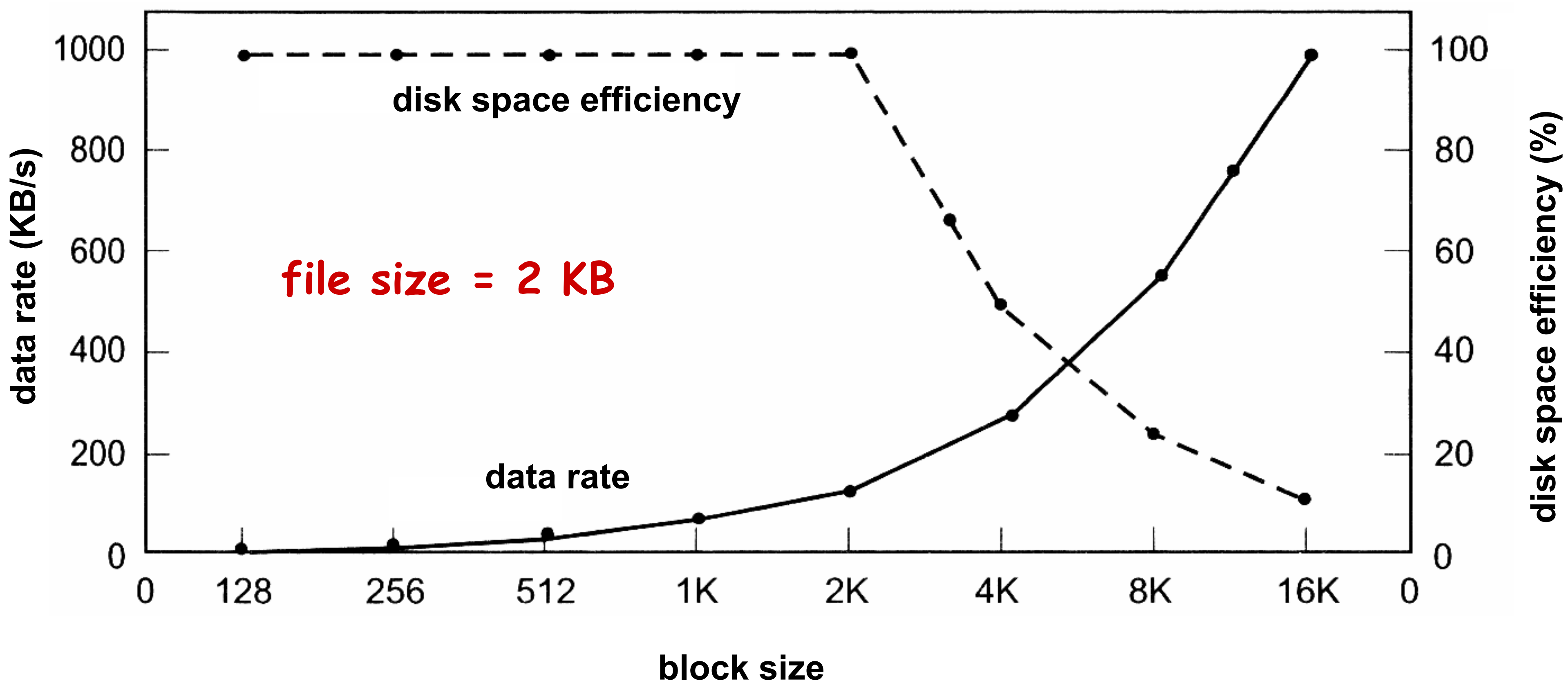
File System Issues

- Managing the disk
    - block size
    - allocation of free blocks
- Improving the performance of the file system
    - caching
    - block read ahead
- File system robustness and reliability
    - backups and recovery
    - consistency
- Journaling and log-based File Systems
- RAID (improving the disk properties)
- Examples of File Systems

Operating Systems II

managing the disk

impact of block size on space efficiency and data rate



(Tanenbaum 2003)

managing the disk

1. Linked list of free blocks

size of list and max. space requirements:

16 GB disk, block size 1k:

--> 16M entries by 32 bit

--> 1 block 255 (+1 to link the blocks) entries --> ~ 40 K blocks

changes over time when
more disk space is allocated

2. bit map of free blocks

size of list and max. space requirements:

16 GB disk, block size 1k:

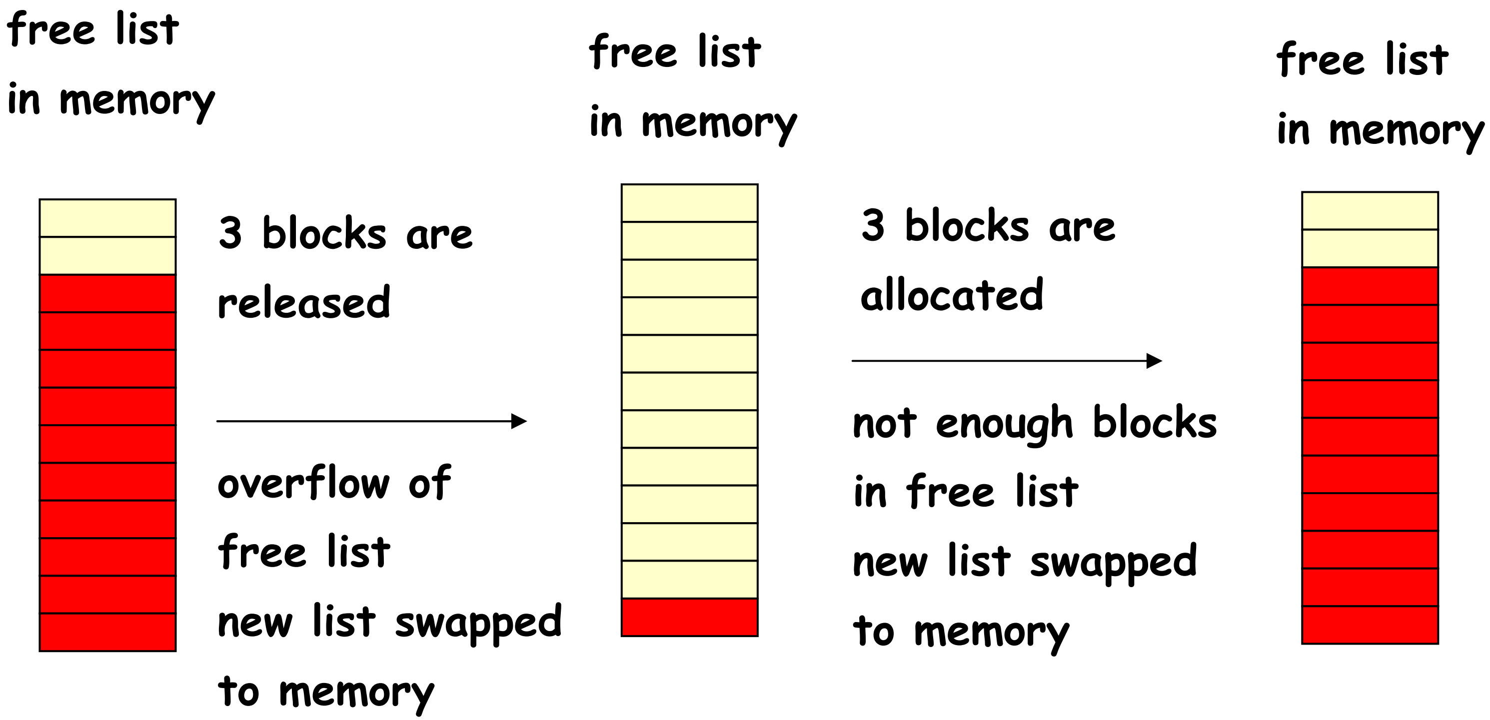
--> 16M entries by 1 bit

--> 1 block 1k x 8 bits --> ~ 2K blocks

fixed over time

managing the disk

problem with caching of free entries in main memory



free list
in memory

3 blocks are
released

overflow of
free list
new list swapped
to memory

free list
in memory

3 blocks are
allocated

not enough blocks
in free list
new list swapped
to memory

free list
in memory

pointer to free blocks 🟥

managing the disk

disk quotas restrict disk space on a per user base.

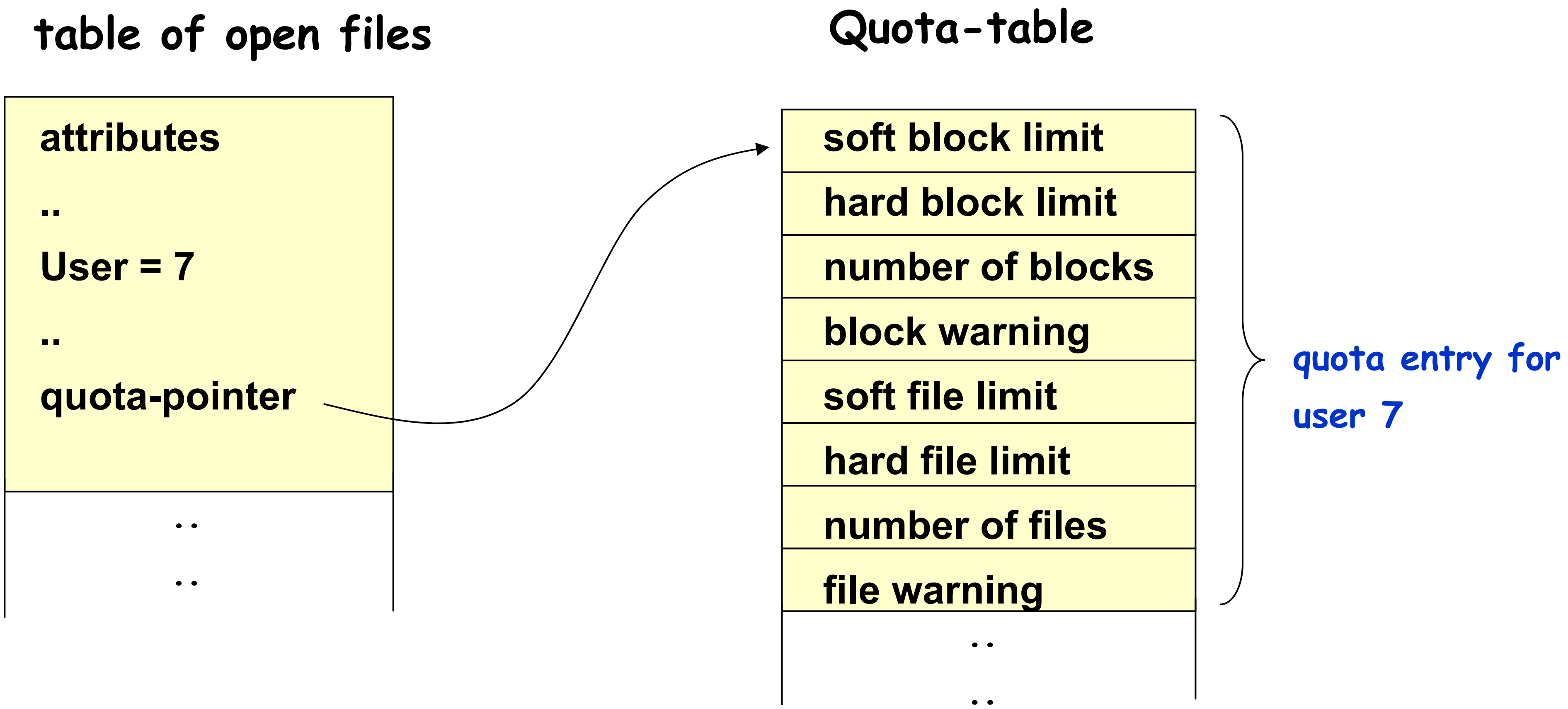


table of open files

attributes
..
User = 7
..
quota-pointer

..
..

Quota-table

soft block limit
hard block limit
number of blocks
block warning
soft file limit
hard file limit
number of files
file warning

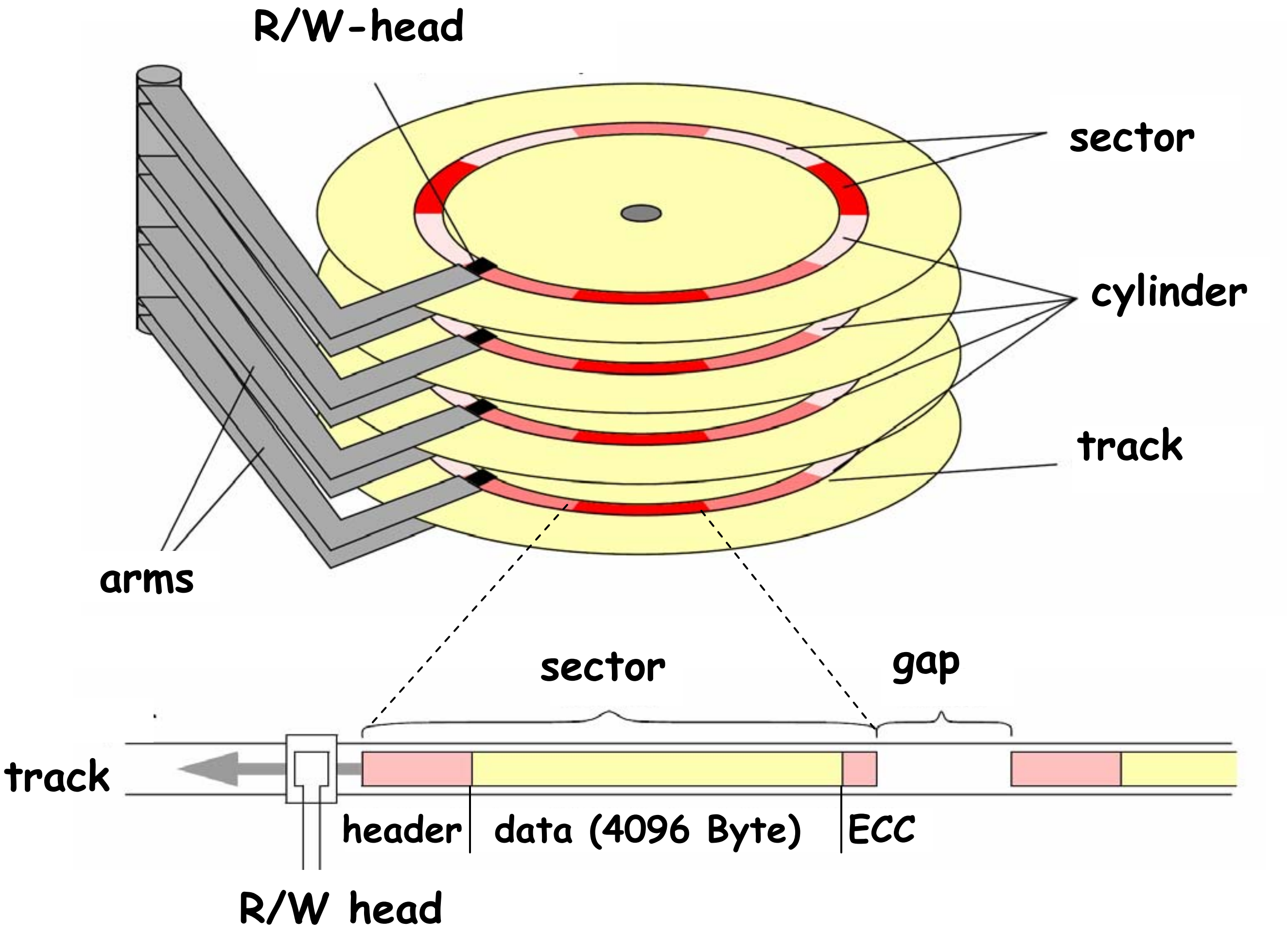
..
..

quota entry for user 7

improving file system performance

- caching
- block read ahead
- optimizing disk head movements
- log-based file systems

recall: the physical organization of a disk

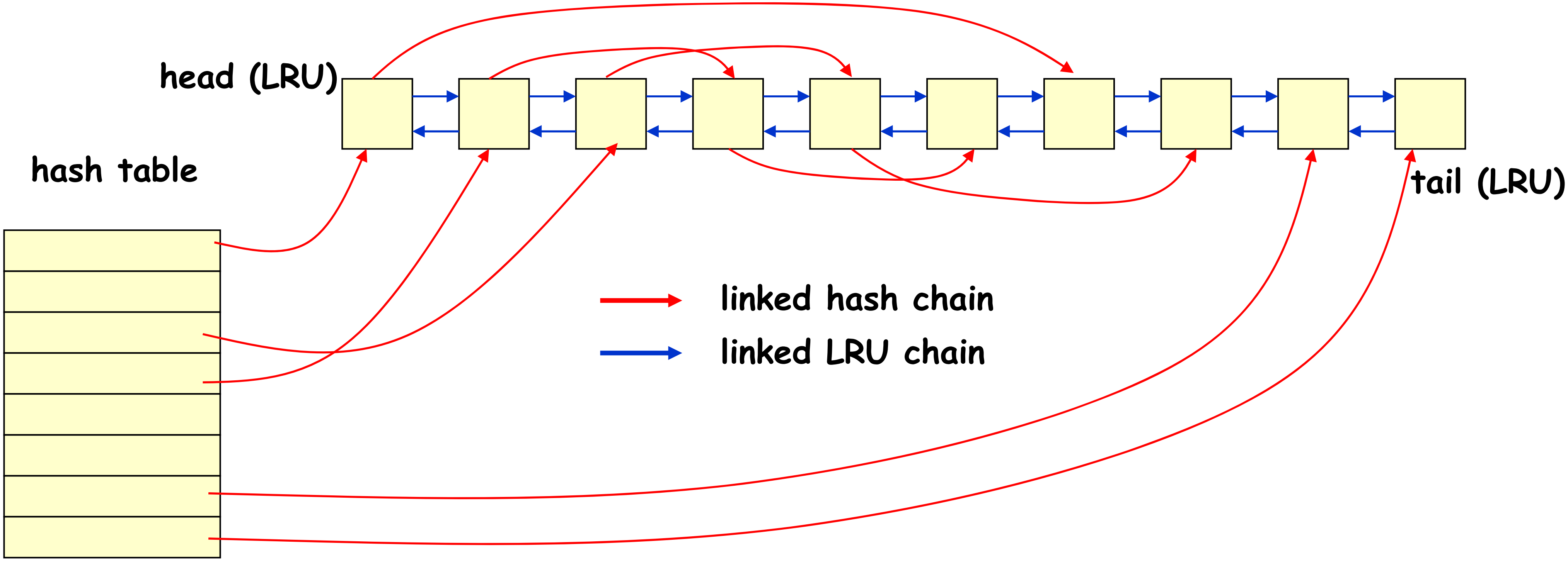


the buffer cache

Problem: access to main memory is up to 6 orders of magnitute faster than a disk access

- ➡ map files to virtual memory.
  - ➡ under explicit progr. control
- ➡ treat main memory as a cache for the disk.
  - ➡ transparent
  - ➡ similarities to virtual memory management.

the buffer cache



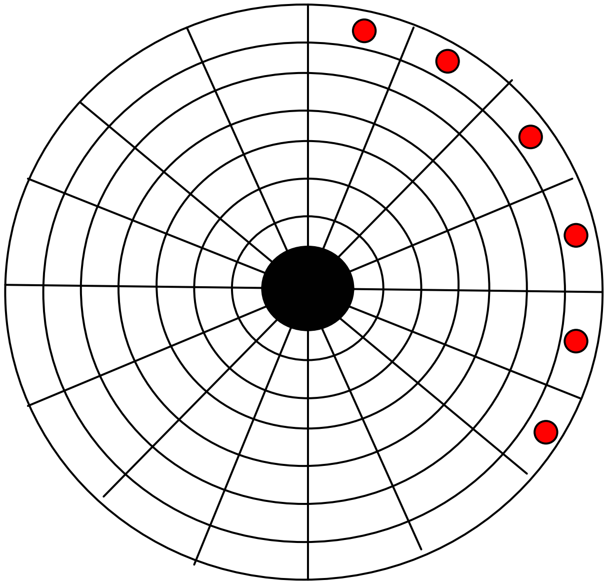
Problem: block contents in memory and block contents on disk are not identical.

- inconsistencies in case of crashes.
- trade-off between frequent disk updates and loss of data.
- explicit synchronization (sync).

Disk Properties

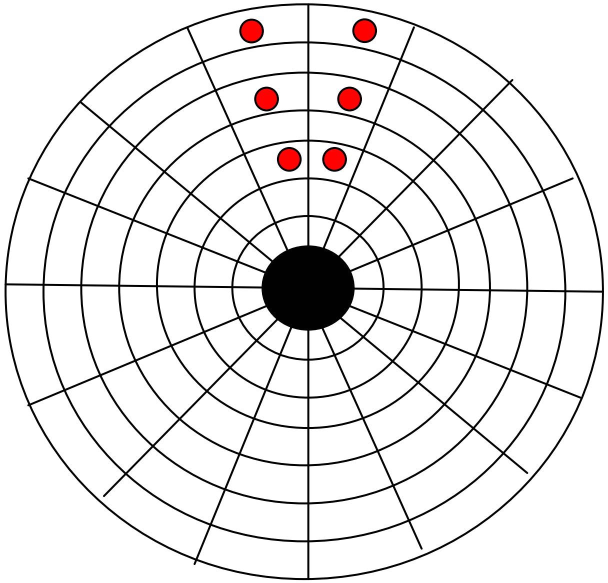
Plattentyp			Seagate Cheetah
Kapazität			73,4 GB
Platten/Köpfe			4/8
Zylinderzahl			29.594
Cache			4 MB
Posititionier-zeiten	Spur zu Spur		0,4/0,6 ms
	mittlere		5,1/5,5 ms
	maximale		10/11 ms
Transferrate			38–64 MB/s
Rotationsgeschw.			10.000 U/min
eine Plattenumdrehung			6 ms
max. Stromaufnahme			11 W

optimizing disk access



i-nodes at the beginning of the disk.

distance between i-node and asociated blocks: number of cylinders/2



i-nodes and asociated blocks are organized in cylinder groups.

Robustness and Dependability of a File System

Loss of Data is the "Super GAU" in a computer system!

While the cost of a new computer is in the order of 5.000 €
the cost of lost date may easily be higher many orders of magnitudes !

File system must be protected against:

- disk crashes
- erroneous software
- malicious accesses

Robustness and Dependability of a File System

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 priviledge
system crashes	fsck, scandisk, journaled file systems
malicious accesses	access protection, encryption, fragmentation
erroneous deletion of files	no physical deletion, backups

Backup copies

physical backup: copies all blocks of the disk to the backup medium.

pro: simple

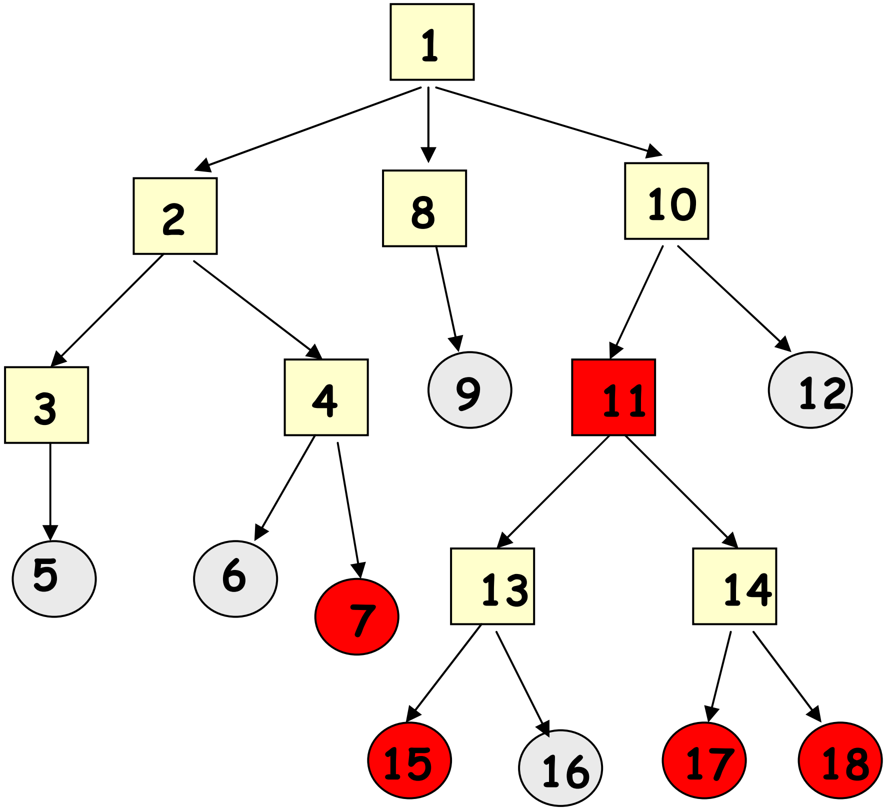
con: saves free blocks, problems with bad blocks, complete backup only.

logical backup: based on the file system structure. Recursively saves directories and files starting at user selected dir's.

pro: incremental algorithm only saves changes since last backup.

con: more complicated implementation.

incremental backup



🟨 unmodified directory

○ file: unmodified since last backup

🔴 🟥 modified since last backup

Incremental backup:

- exploits time and date to save modifications since last backup
- saves the entire path to the modified files including directories even when they didn'd change.

incremental backup

phase 1,2 : mark

phase 3,4 : save

1. mark modified files and all dir's
2. unmark dir's to unmodified files
3. store marked directories
4. store marked files

- the scheme stores all needed directories on the backup record first.
- during recovery they will occur first on the sequential medium and restored first.

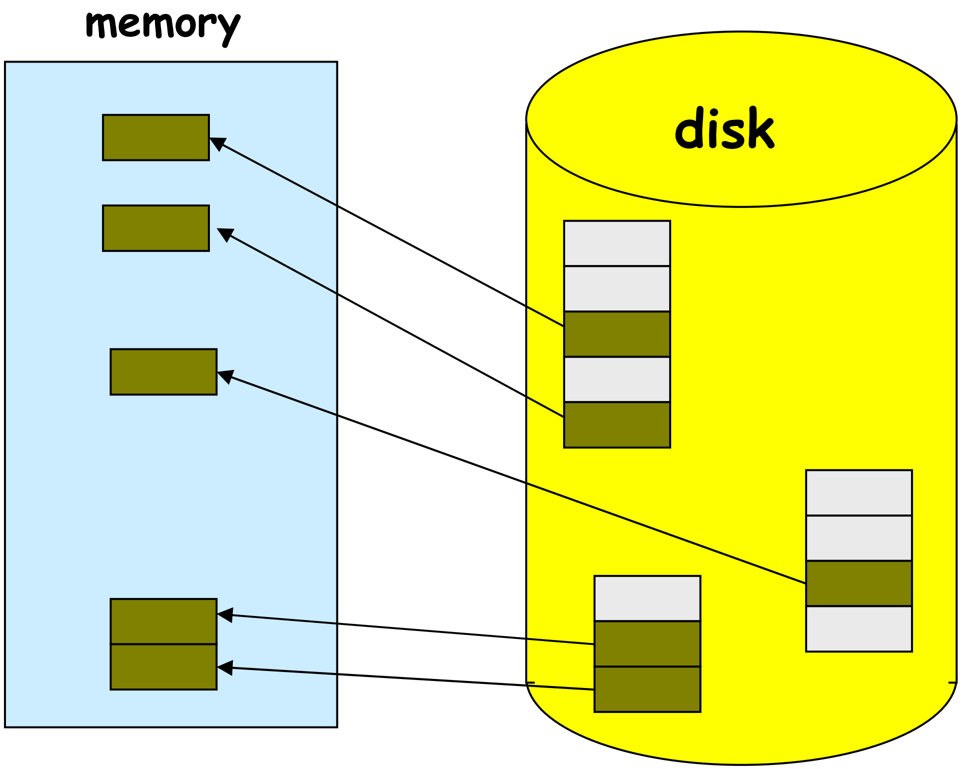
file backup

Issues to be considered:

1. List of free blocks is a data structure in volatile memory and has to be rebuilt.
2. Multiple links to a file. This file has to be restored only once but the link has to be re-established in all directories.
3. Sparsely used files with holes.
4. Special files as pipes and device specific files should not be backed up.

file system consistency

Changes on files are made in volatile fast memory and are not immediately stored on disk persistently.



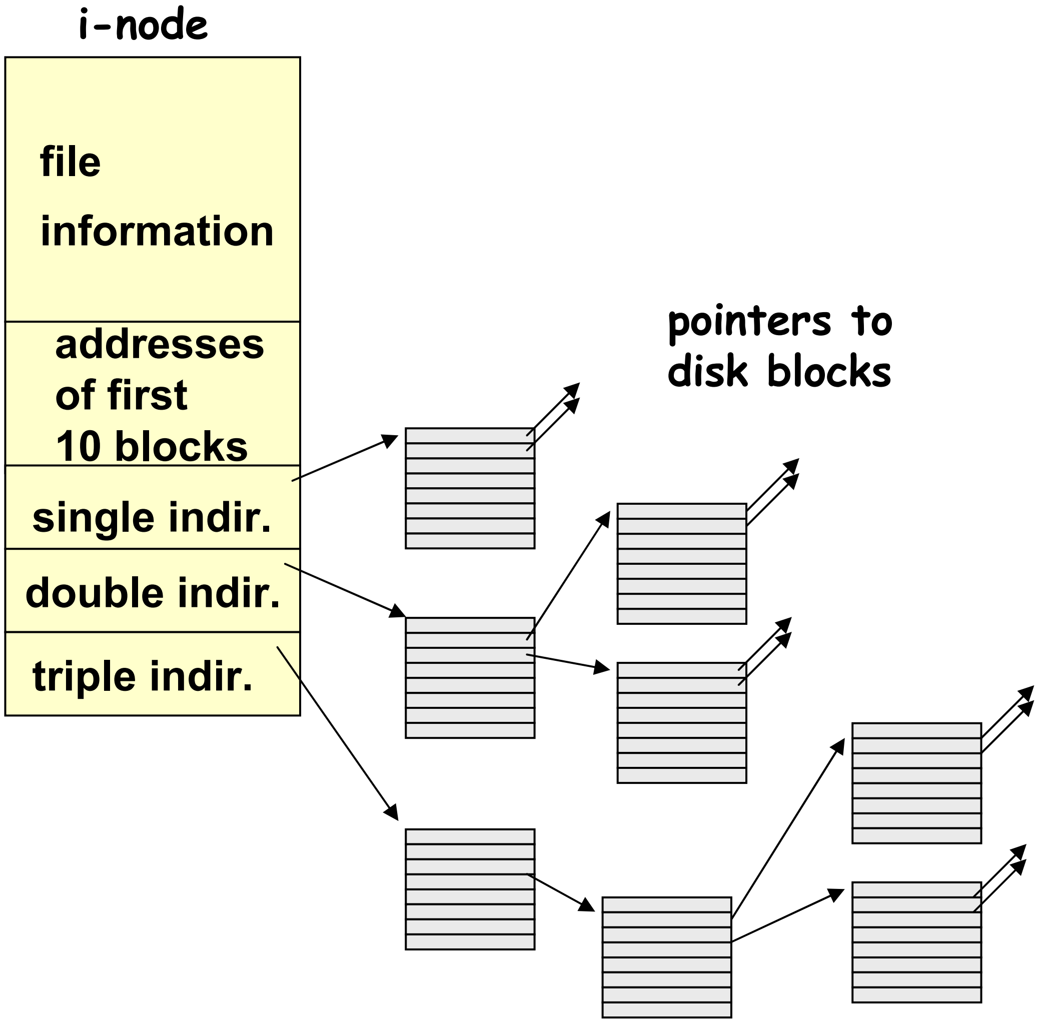
file images
(some blocks of a file)

directory images
(some blocks of a directory)

i-node images
(some blocks of the inode table)

free list images
(some blocks of the free list)

i-nodes in UNIX



i-node

file information

addresses of first 10 blocks

single indir.

double indir.

triple indir.

pointers to disk blocks

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.

file system consistency

after a crash...

First goal: maintain the consistency of the **meta-data**, i.e. all data structures which are involved in the management of the file system. E.g. i-nodes, directories, free-lists.

→ Exploit redundancy in the file system organization.

Normally not considered: modifications on file data. **They are lost.**

→ Journaled File Systems, Data Bases

file system consistency

fsck: file system check

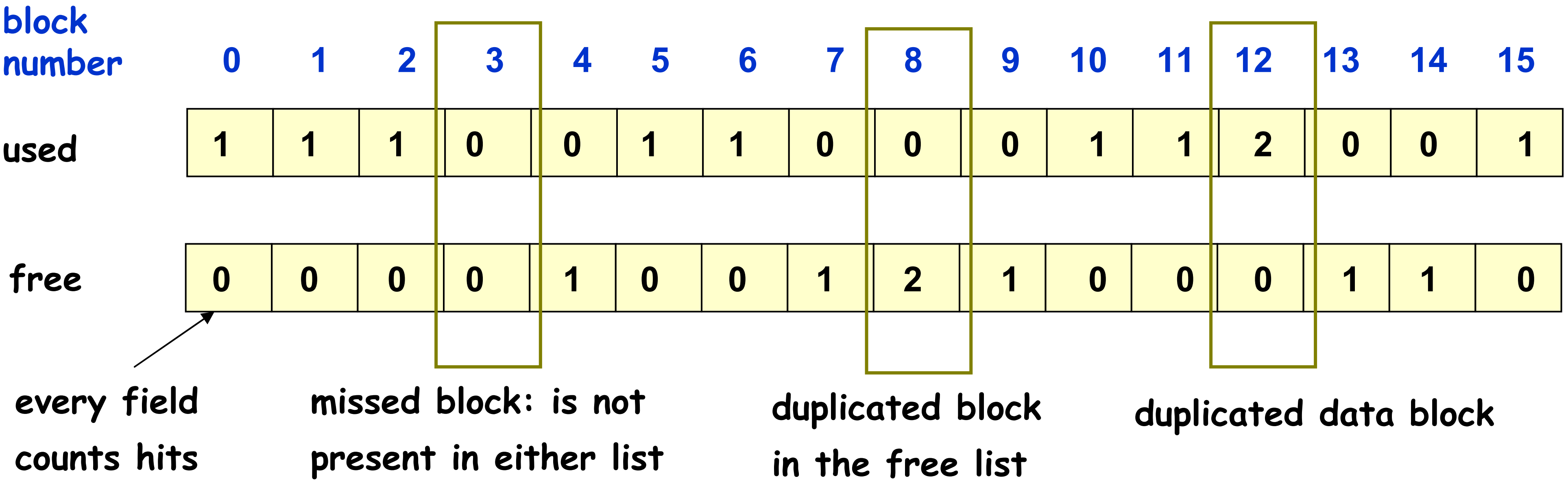
checks file system meta data on consistency.

1. missed or duplicated blocks
2. directory structure

file system consistency

Missed or duplicated blocks: fsck

1. scans all inodes to build the list of used blocks
2. scans the free list or bit map to find the free blocks



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

file system consistency

Case 1: Missed 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: reduced disk capacity

Solution: Assign missed blocks to free list

file system consistency

Case 2: Duplicated block in the free list

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

Solution: Rebuild free list and delete duplicated entry

file system consistency

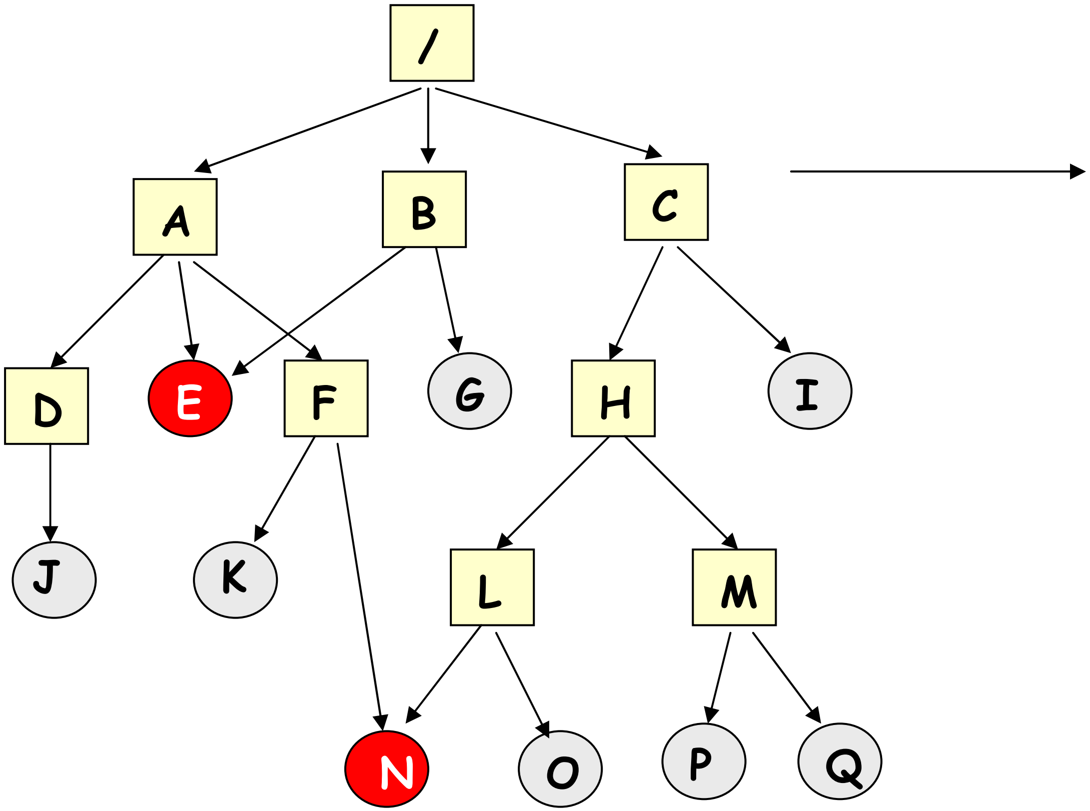
Case 3: Duplicated data block, i.e. block occurs in two files.

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: simple deletion results in further inconsistencies.

Solution: copy one block to a free block and update the lists.

checking the directory system



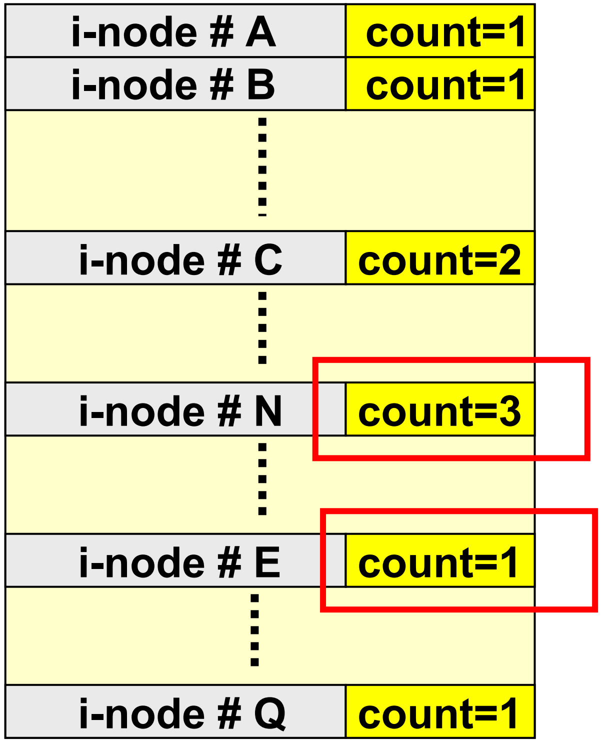
i-node # A	count=1
i-node # B	count=1
⋮	
i-node # C	count=2
⋮	
i-node # N	count=3
⋮	
i-node # E	count=1
⋮	
i-node # Q	count=1

count too high

count too low

1. step: build a list indexed by i-node numbers and count the occurence of every file in every directory.
2. step: compare the list count with the link counter in the i-node entries of files.

checking the directory system



non-critical: i-node remains existent even when all links to a file in the directories are removed.

--> a space/efficiency problem

link count in i-node
is higher than act. count in list

link count in i-node
is lower than act. count in list

critical: i-node will be deleted even if there exists a link to the file in some directory. When link counter goes to "0" the file system marks i-node as free and releases associated blocks.

File system consistency

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 occured before a crash. Need a logging facility.

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)

References:

A. Tanenbaum: Moderne Betriebssysteme, Chapter 6.3.8

M. Rosenblum, J.K. Ousterhout: The Design and Implementation of a Log-structured File System, in:Proc. 13th Symposium on Operating System Principles, ACM, 1991

A. Chang, M.F. Mergen, R.K. Rader, J.A. Roberts, S.L. Porter: Evolution of Storage Facilities in AIX Version 3 for RISC System/6000 Processors, IBM Journal on Research and Development, Vol.34, No. 1, January 1990

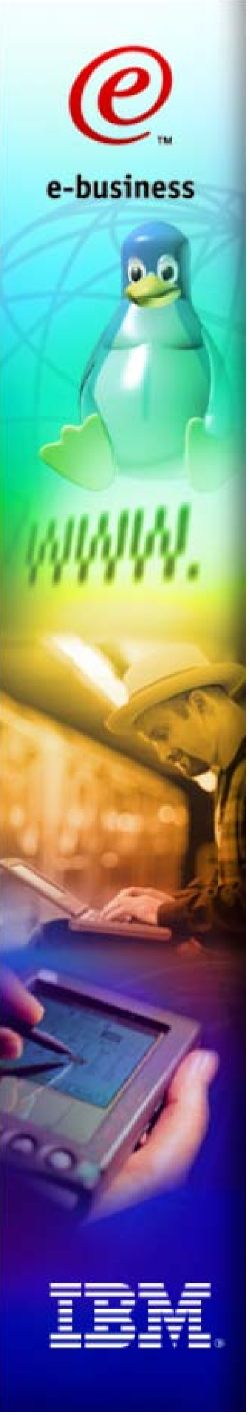
http://www.backupbook.com/03Freezes_and_Crashes/02Journaling.html

Why use JFS ?

Steve Best
Linux Technology Center -
JFS for Linux
IBM Austin

- **Highly Scalable 64 bit file system:**
  - **scalable from very small to huge (up to 4 PB)**
  - **algorithms designed for performance of very large systems**
- **Performance tuned for Linux**
- **Designed around Transaction/Log**
  - **(not an add-on)**
- **Restarts after a system failure < 1 sec**

http://www.perl.org/tpc/2002/sessions/best_steve.pdf



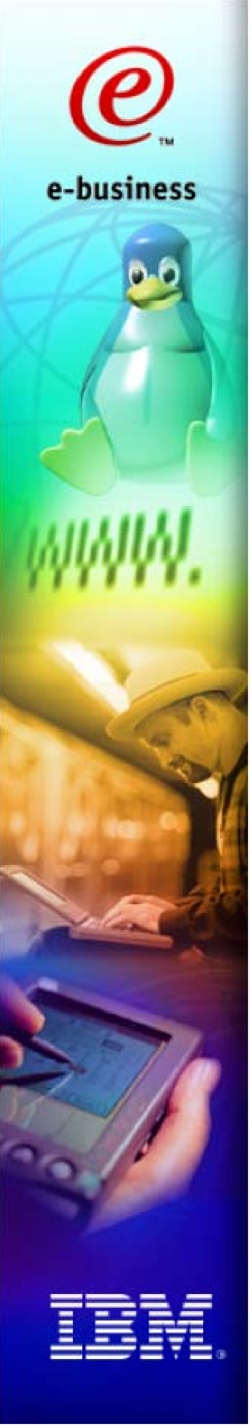
What operations are logged

Steve Best
Linux Technology Center -
JFS for Linux
IBM Austin

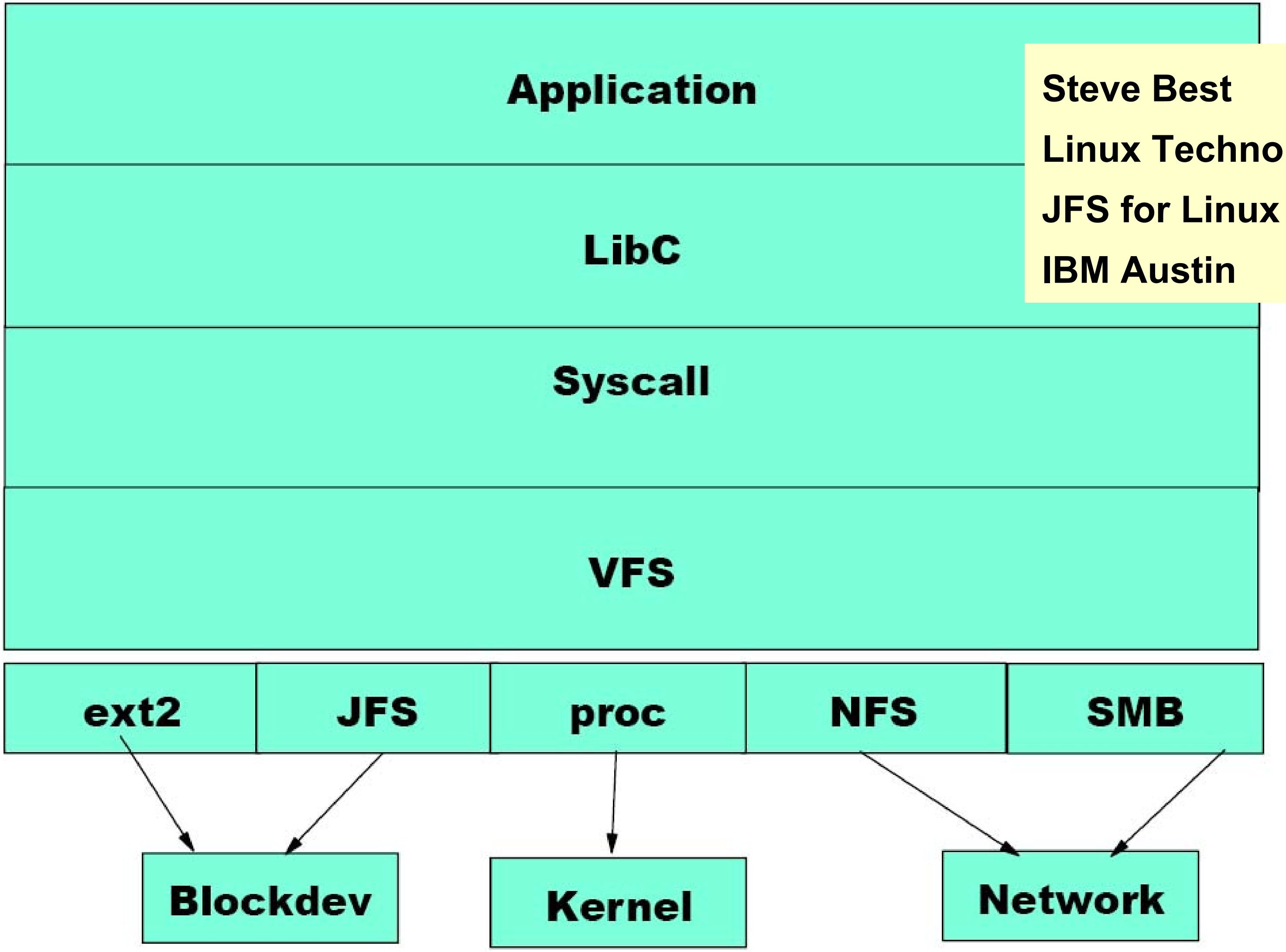
Only meta-data changes:

- File creation (create)
- Linking (link)
- Making directory (mkdir)
- Making node (mknod)
- Removing file (unlink)
- Symbolic link (symlink)
- Set ACL (setacl)
- Truncate regular file

http://www.perl.org/tpc/2002/sessions/best_steve.pdf



Virtual and Filesystem



Application

LibC

Syscall

VFS

ext2

JFS

proc

NFS

SMB

Blockdev

Kernel

Network

Steve Best

Linux Technology Center -

JFS for Linux

IBM Austin

http://www.perl.org/tpc/2002/sessions/best_steve.pdf

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 50μs).

➡ read acceses 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".

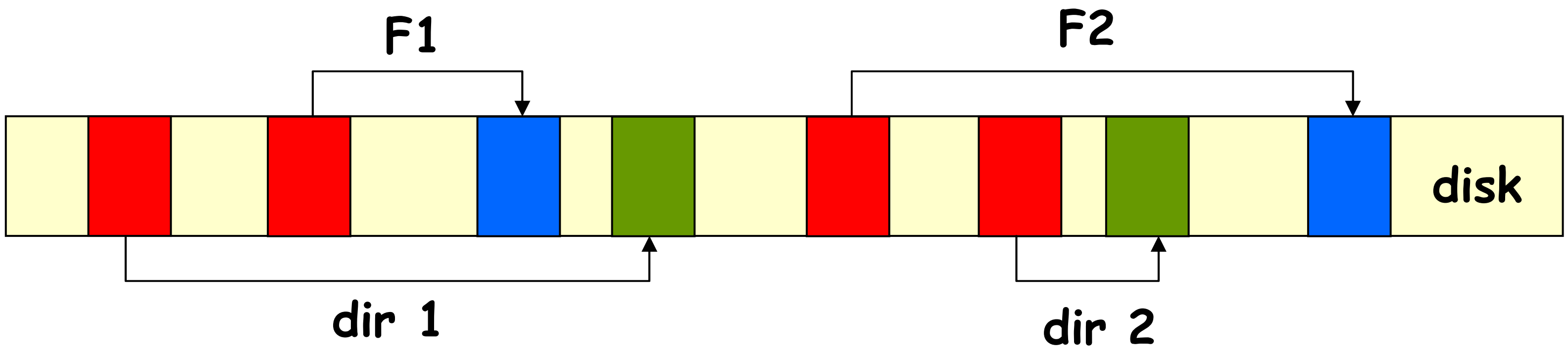
Log structured file systems

Creating files in a conventional file system (FFS):

creating:

/dir/F1 and

/dir/F2



🟦 data

🟥 i-node

🟩 directory

FFS:

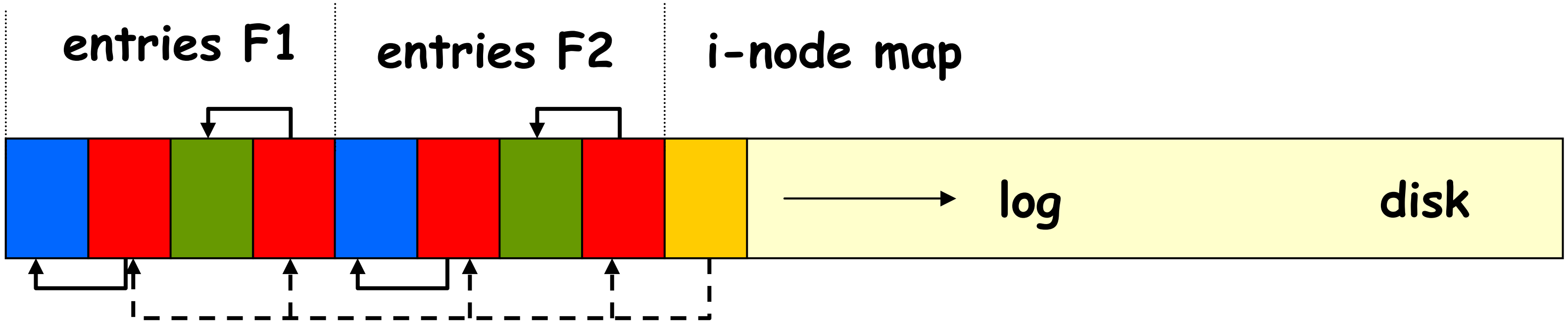
FFS requires 10 non-sequential writes preceeded by a seek. (I-nodes for new files are written twice to ease recovery)

Log structured file systems

creating:

/dir/F1 and

/dir/F2



🟦 data

🟥 i-node

🟩 directory

🟨 i-node map

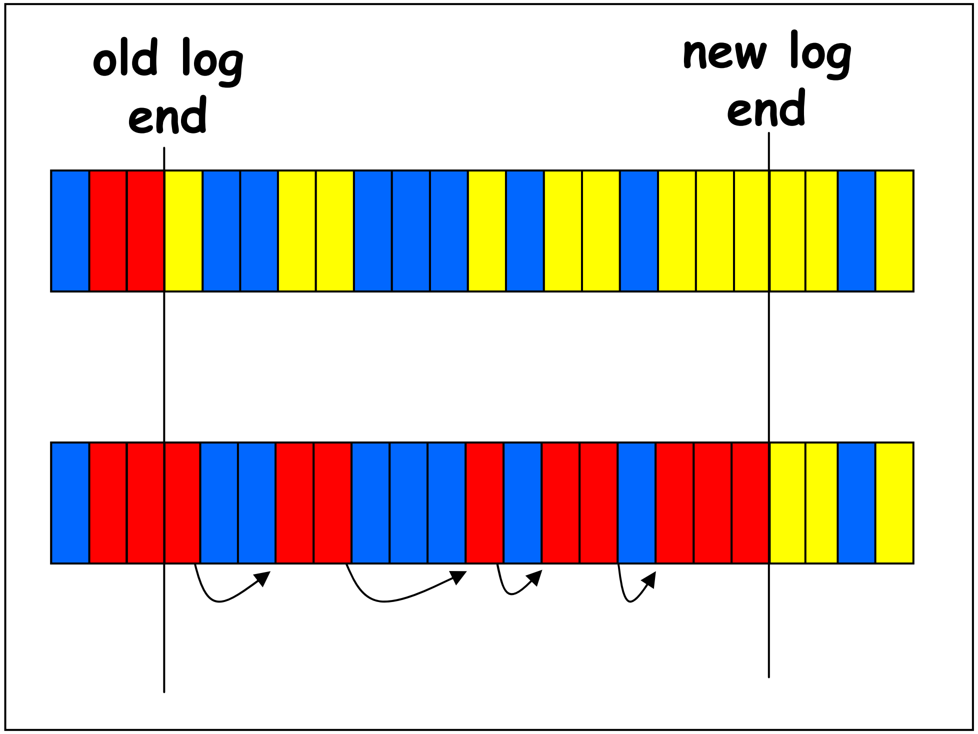
LFS:

new data and metadata is written in a single large write.

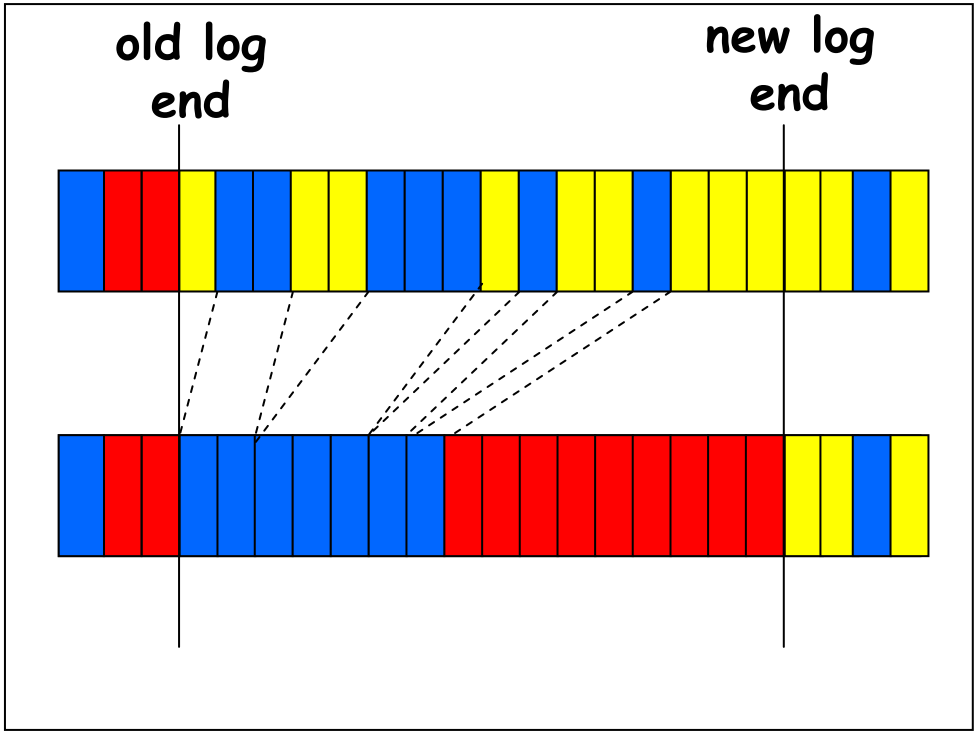
Log structured file systems

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

Free space management alternatives for log-structured file systems:



threaded log



copy and compact log

Log structured file systems

Problems with "Threading":

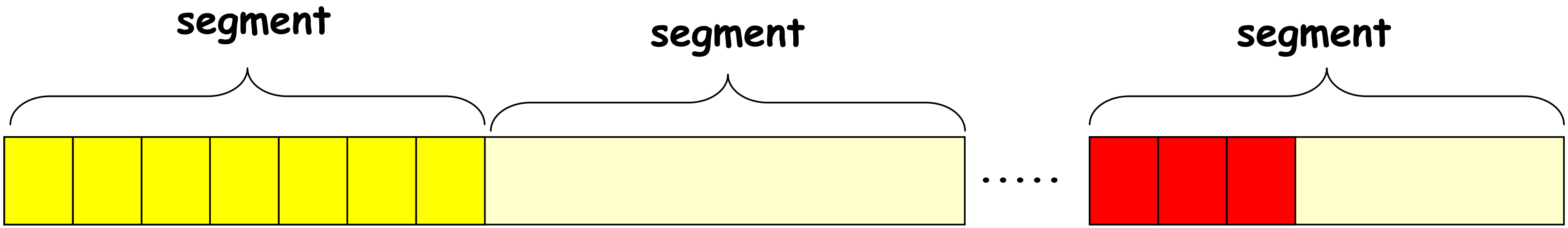
Over time the log becomes fragmented and the benefits are lost

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)



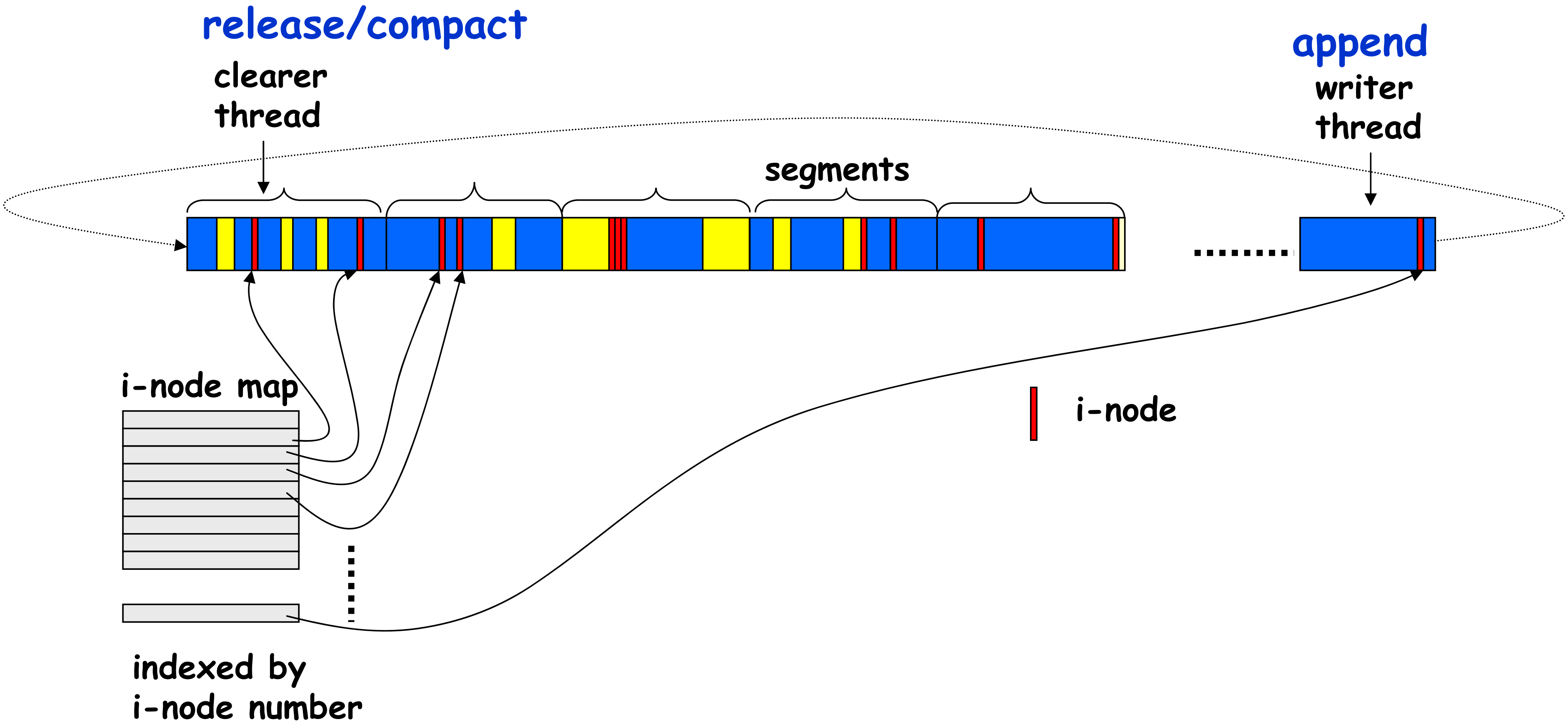
segments: - large number of fixed size contiguous blocks (an extend)

- 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)

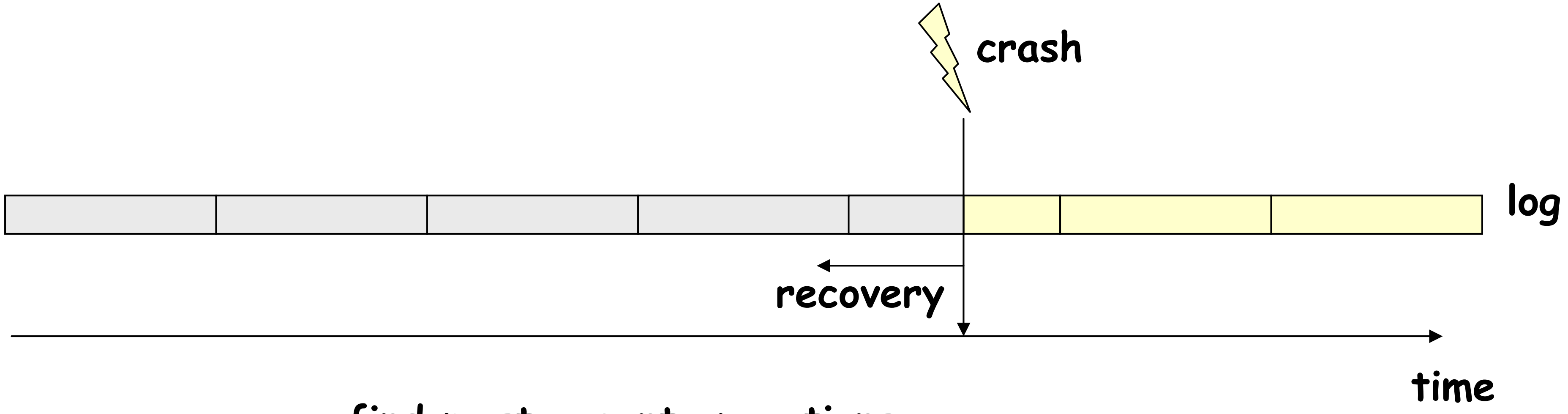
Segmented structure allows a combination of threading and copying.

- All segment 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.

Log-structured File System (LFS)



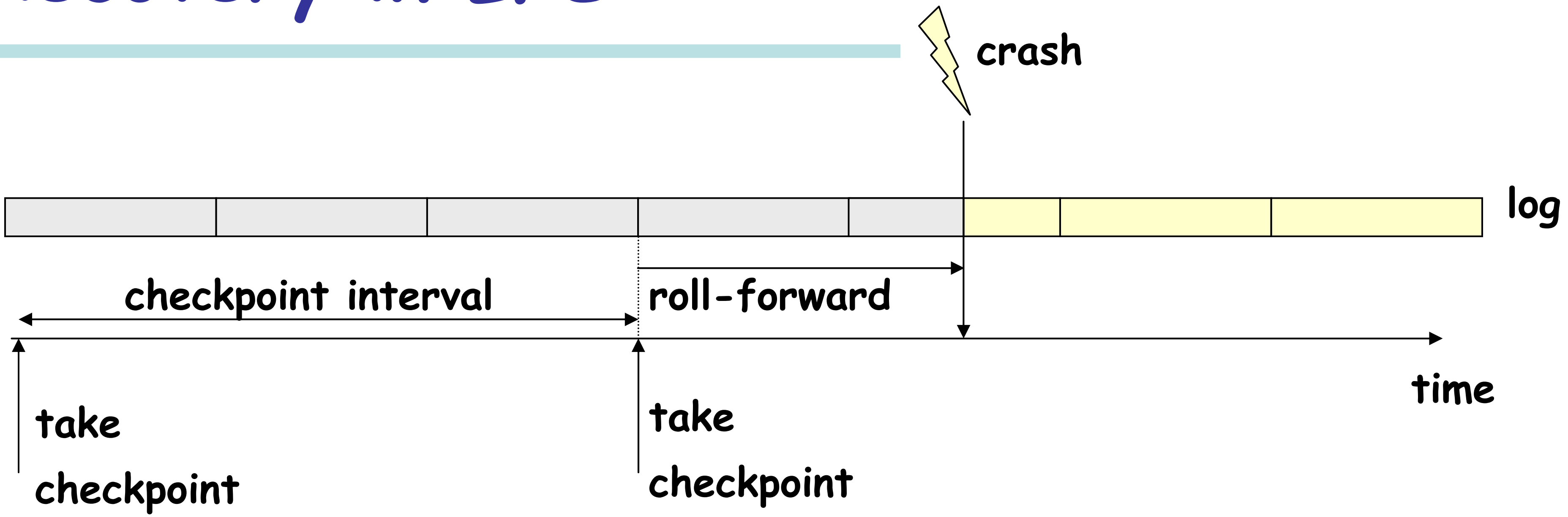
Recovery in LFS



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

Problem: How far to go back?

Recovery in LFS



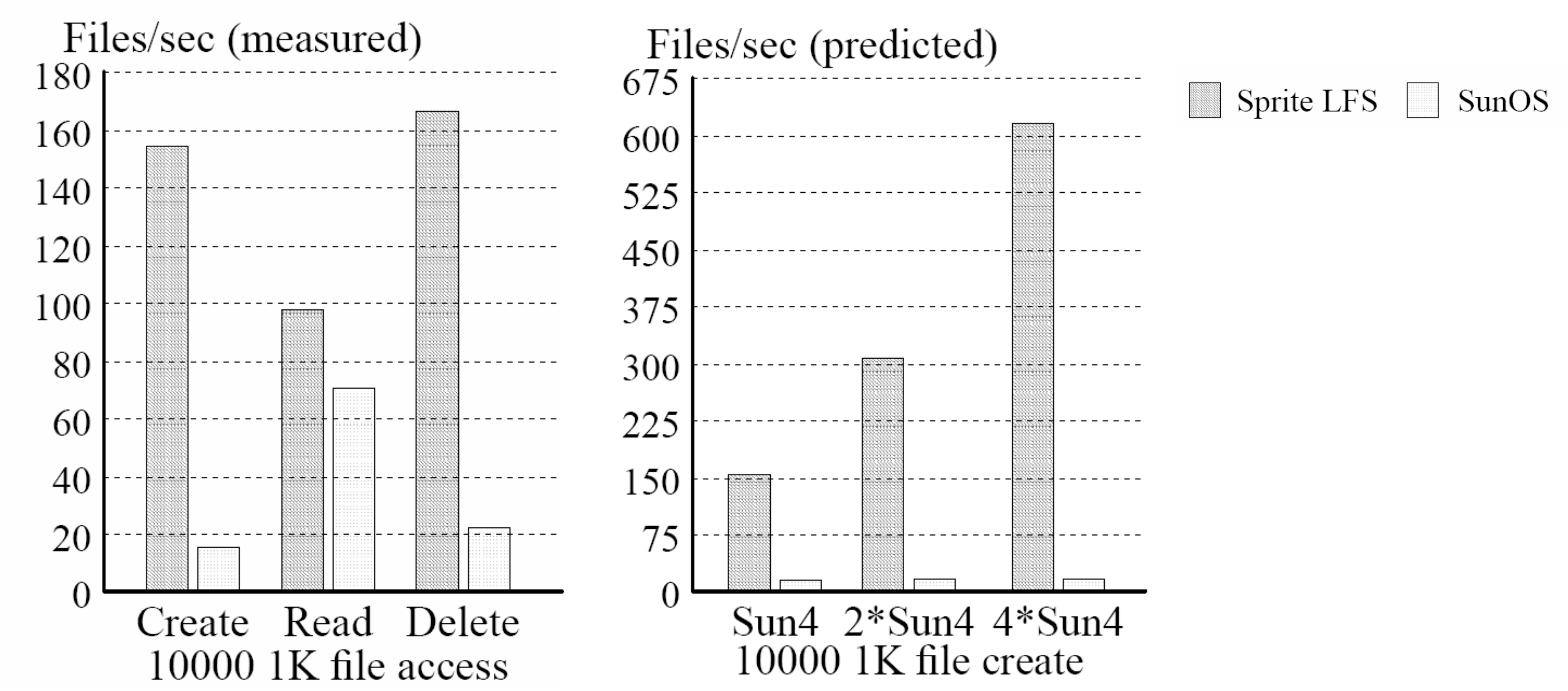
- checkpoint regions are kept in special fixed positions on disk
- checkpoint region contains:
  - addr. of all blocks in the i-node map
  - segment usage tables
  - current time and pointer to last segment written
- two checkpoint regions are maintained to deal with crashes during checkpointing

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)

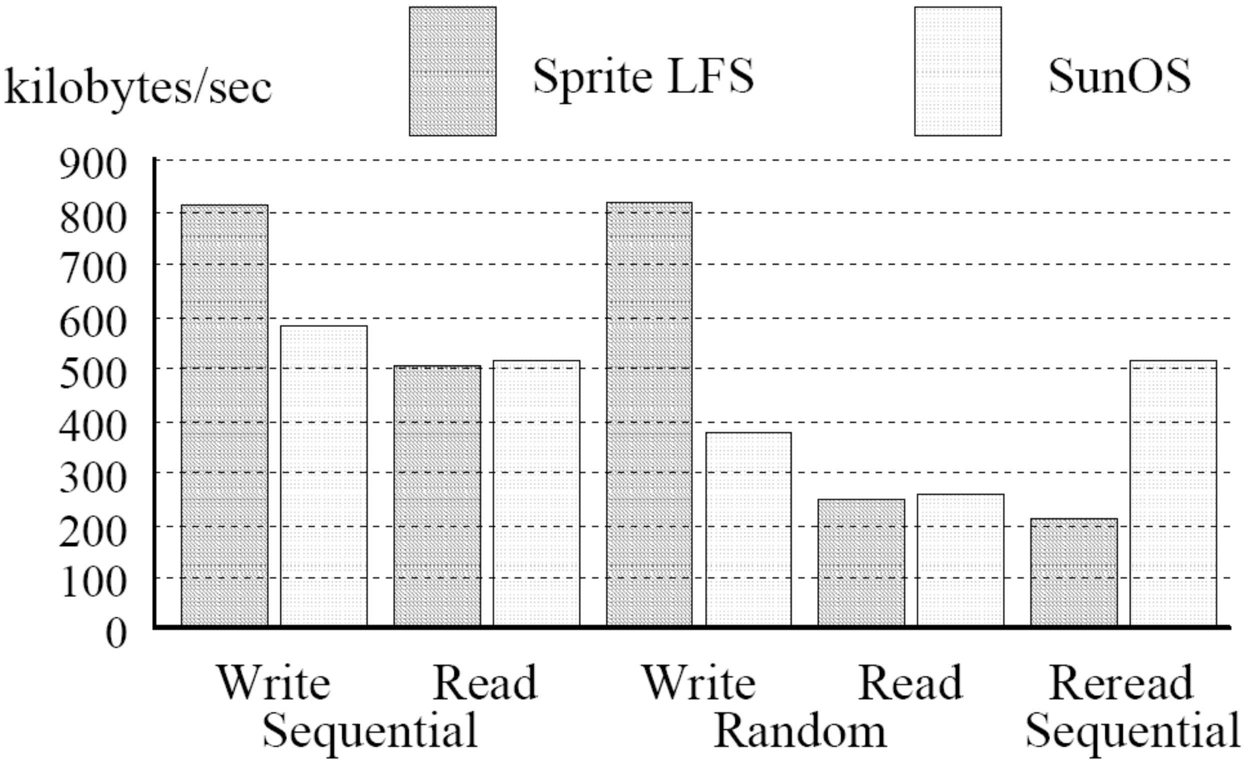
performance comparison: small file performance



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

RAID: Reliable Array of Inexpensive Disks

D.A. Patterson, G.A. Gibson, R. Katz: A Case for Redundand Arrays of Inexpensive Disks (RAID), *Proc. ACM SIGMOD Intern. Conference on Management of Data*, 1988

Goals:

Performance Improvement: parallel disks can be accessed concurrently.

Reliability and availability: RAID exploits redundancy of disks.

Transparency: RAID looks like a single large, fast and reliable disk (SLED).

RAID-level 1

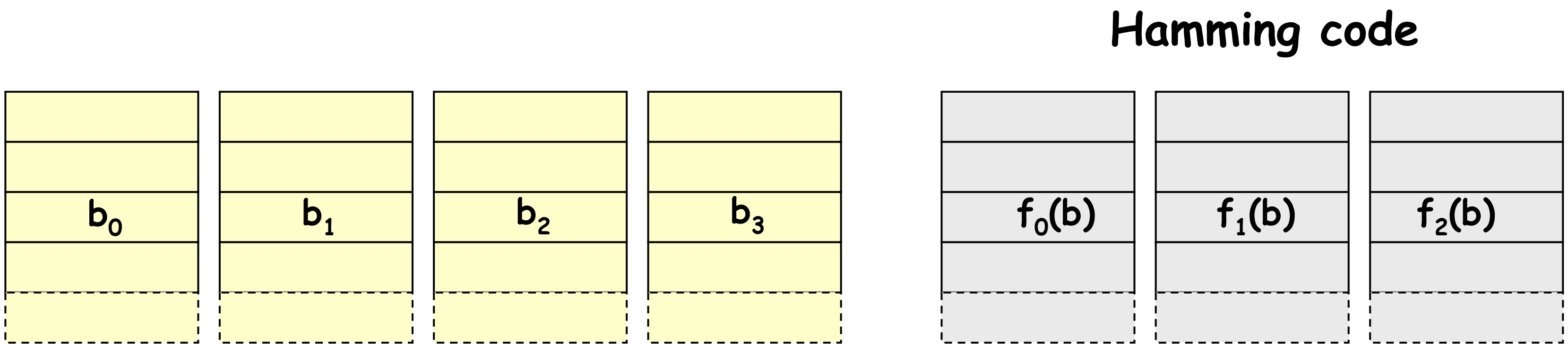
stripe 0	stripe 1	stripe 2	stripe 3
stripe 4	stripe 5	stripe 6	stripe 7
stripe 8	stripe 9	stripe 10	stripe 11
stripe 12	stripe 13	stripe 14	stripe 15

stripe 0	stripe 1	stripe 2	stripe 3
stripe 4	stripe 5	stripe 6	stripe 7
stripe 8	stripe 9	stripe 10	stripe 11
stripe 12	stripe 13	stripe 14	stripe 15

RAID-level 0	RAID-level 1
non-redundant	mirrored disk
high transfer rates	high transfer rates

RAID-level 2

Needs strictly synchronized disks!

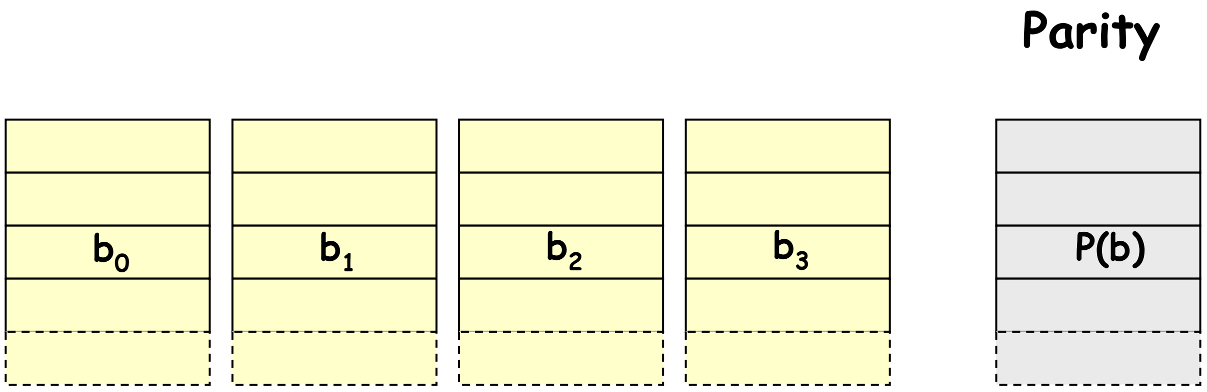


RAID-level 2

word- or byte-oriented

RAID-level 3

Needs strictly synchronized disks!

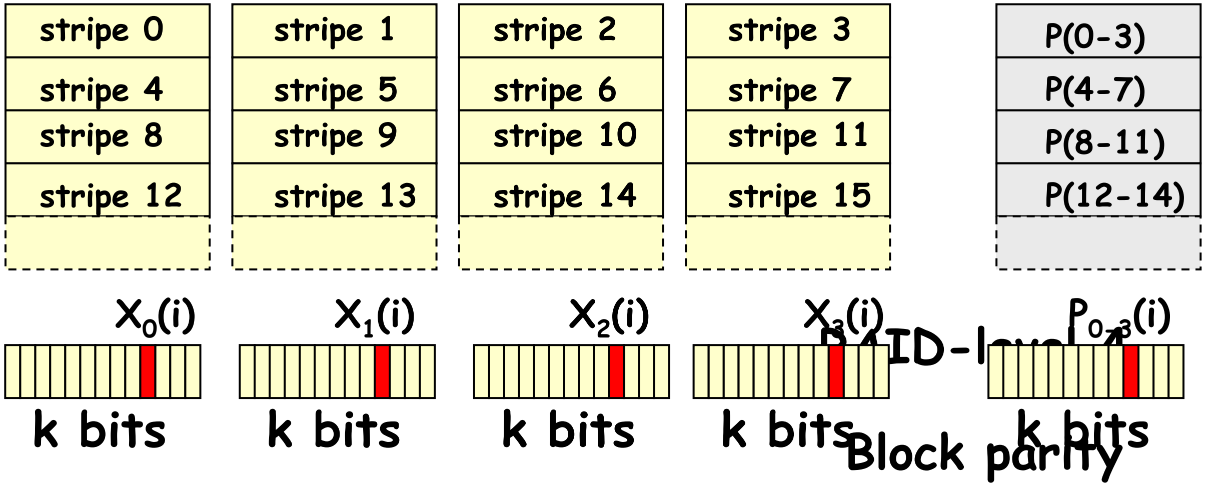


RAID-level 3

word- or byte-oriented

Allows error correction in case of a defective disk because the positon of the defective bit is known !

RAID-level 4



stripe 0	stripe 1	stripe 2	stripe 3	P(0-3)
stripe 4	stripe 5	stripe 6	stripe 7	P(4-7)
stripe 8	stripe 9	stripe 10	stripe 11	P(8-11)
stripe 12	stripe 13	stripe 14	stripe 15	P(12-14)

$P_{0-3}(i) = X_3(i) \oplus X_2(i) \oplus X_1(i) \oplus X_0(i)$ starting point

$P'_{0-3}(i) = X_3(i) \oplus X'_2(i) \oplus X_1(i) \oplus X_0(i)$ changing stripe 2

$P'_{0-3}(i) = X_3(i) \oplus X'_2(i) \oplus X_1(i) \oplus X_0(i) \oplus X_2(i) \oplus X_2(i)$

$P'_{0-3}(i) = P_{0-3}(i) \oplus X'_2(i) \oplus X_2(i)$

➡ A write operation requires 2 reads and 2 writes

RAID-level 5

Problem with RAID-4: Parity disk becomes a bottleneck.

stripe 0	stripe 1	stripe 2	stripe 3	P(0-3)
stripe 4	stripe 5	stripe 6	P(4-7)	stripe 7
stripe 8	stripe 9	P(8-11)	stripe 10	stripe 11
stripe 12	P(12-14)	stripe 13	stripe 14	stripe 15
P(15-19)	stripe 16	stripe 17	stripe 18	stripe 19

RAID-level 5

Block parity

Raid-level 6 tolerates two disk crashes and guarantees a very high availability of data. Needs N+2 disks and has to write 2 Parity blocks on a write operation.

Examples of File Systems

Unix File System

NTFS (NT File System)

Example: Unix file system

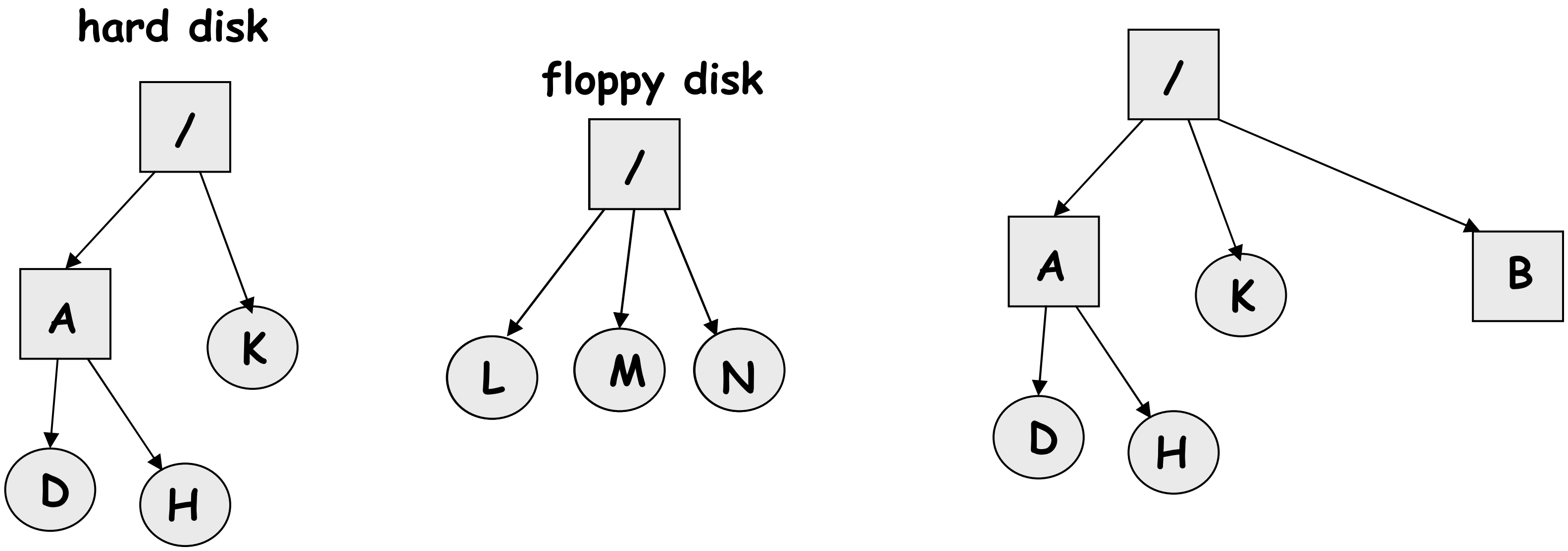
Unix Files:

- File is a sequence of bytes.
- File extensions are conventions.
- Few file types are supported via file type.
- File names up to 255 characters (previously 14 chars.)

Unix supported file types:

- regular files
- directories contains a list of file names and the resp. i-nodes
- named pipes
- character oriented special files used to model serial I/O devices
- block-oriented special files used to model raw disk partitions

Mounting file systems

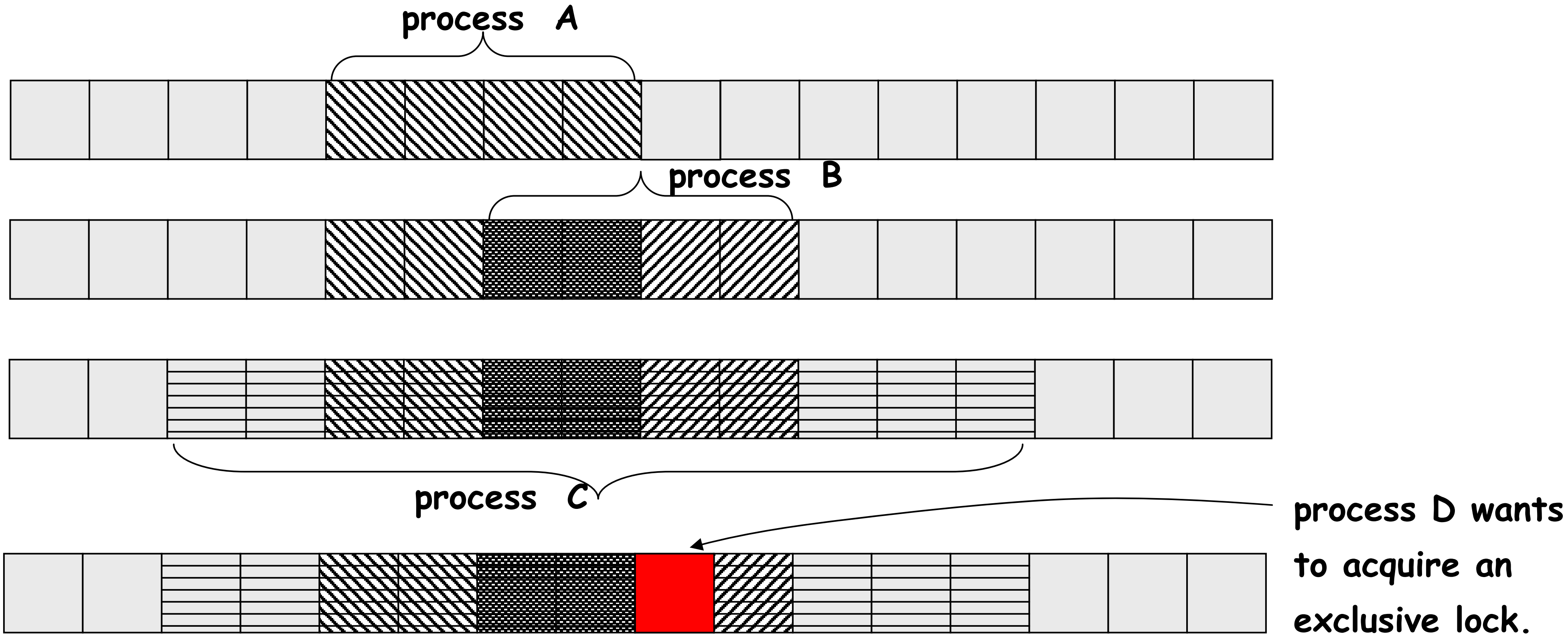


Unix allows a transparent view on different file systems of different storage devices via the mount concept.

Locking file regions

Objective: Improving the granularity of locking down to the byte of a file.

1. shared Locks
2. exclusive locks



process D wants to acquire an exclusive lock.

Unix system calls

File related system calls

```
fd = creat(name, mode)
fd = open(path, how, options...)
s = close(fd)
n = read(fd, buffer, nbytes)
n = write(fd, buffer, nbytes)
position = lseek(fd, offset, whence)
s = stat(name, &buf)
s = fstat(fd, &buf)
s = pipe(&fd[0])
s = fcntl(fd, cmd, . . . )
```

- device which holds the file
- i-node number
- mode
- number of links
- group
- size in bytes
- time of creation
- time of last access
- time of last modification

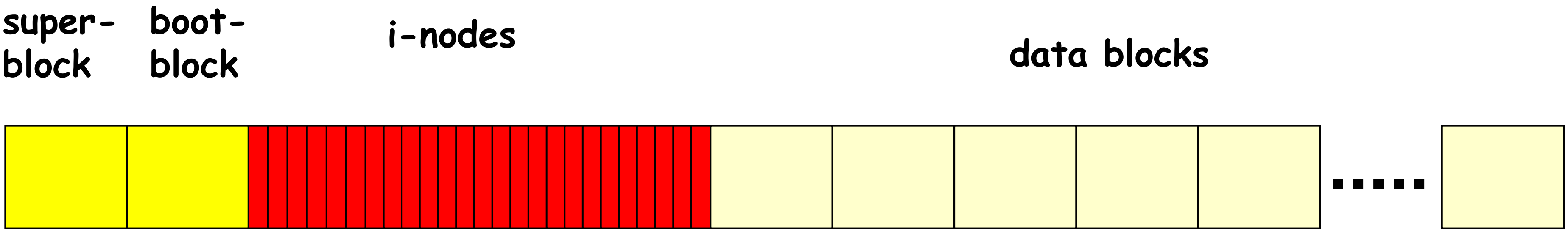
Unix system calls

Directory related system calls

s = mkdir(path, mode)	Create a directory
s = mkdir(path)	delete directory
s = link(oldpath, newpath)	create a link to an exicting file
s = unlink(path)	delete link
s = chdir(path)	change working directory
dir = opendir(path)	open directory for read
s = closedir(dir)	close directory
dirent = readdir(dir)	read a directory entry
rewind(dir) =	rewind und read again

Unix file system management

- Classical Unix System



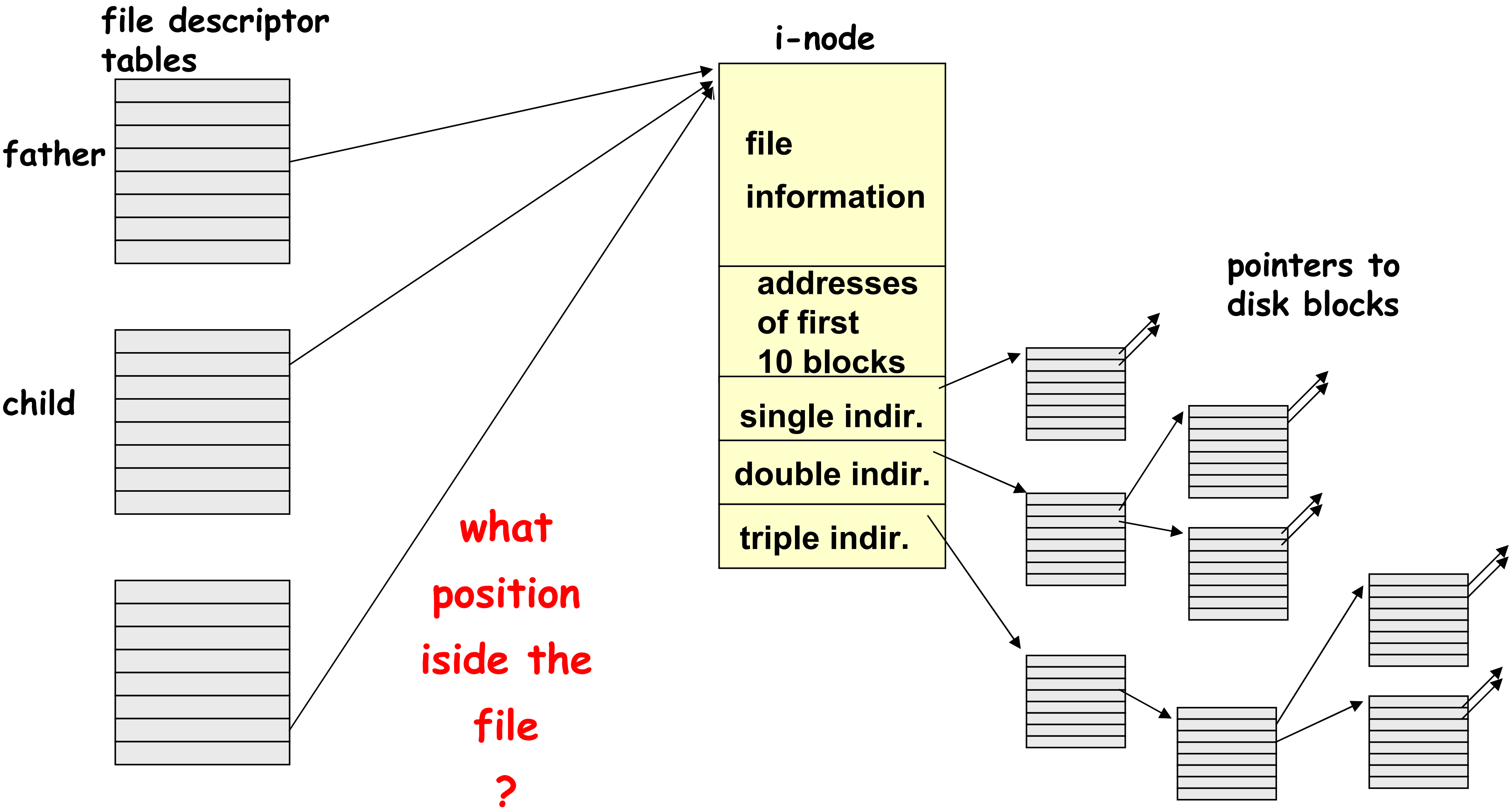
- Berkeley Fast File :
  - long names (255 charakters)
  - structuring the disk in cylinder groups each with own super block, i-nodes and data blocks
  - 2 block sizes

- Linux File System: very similar to 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.

file allocation



file descriptor tables

father

child

i-node

file information

addresses of first 10 blocks

single indir.

double indir.

triple indir.

pointers to disk blocks

what position iside the file ?

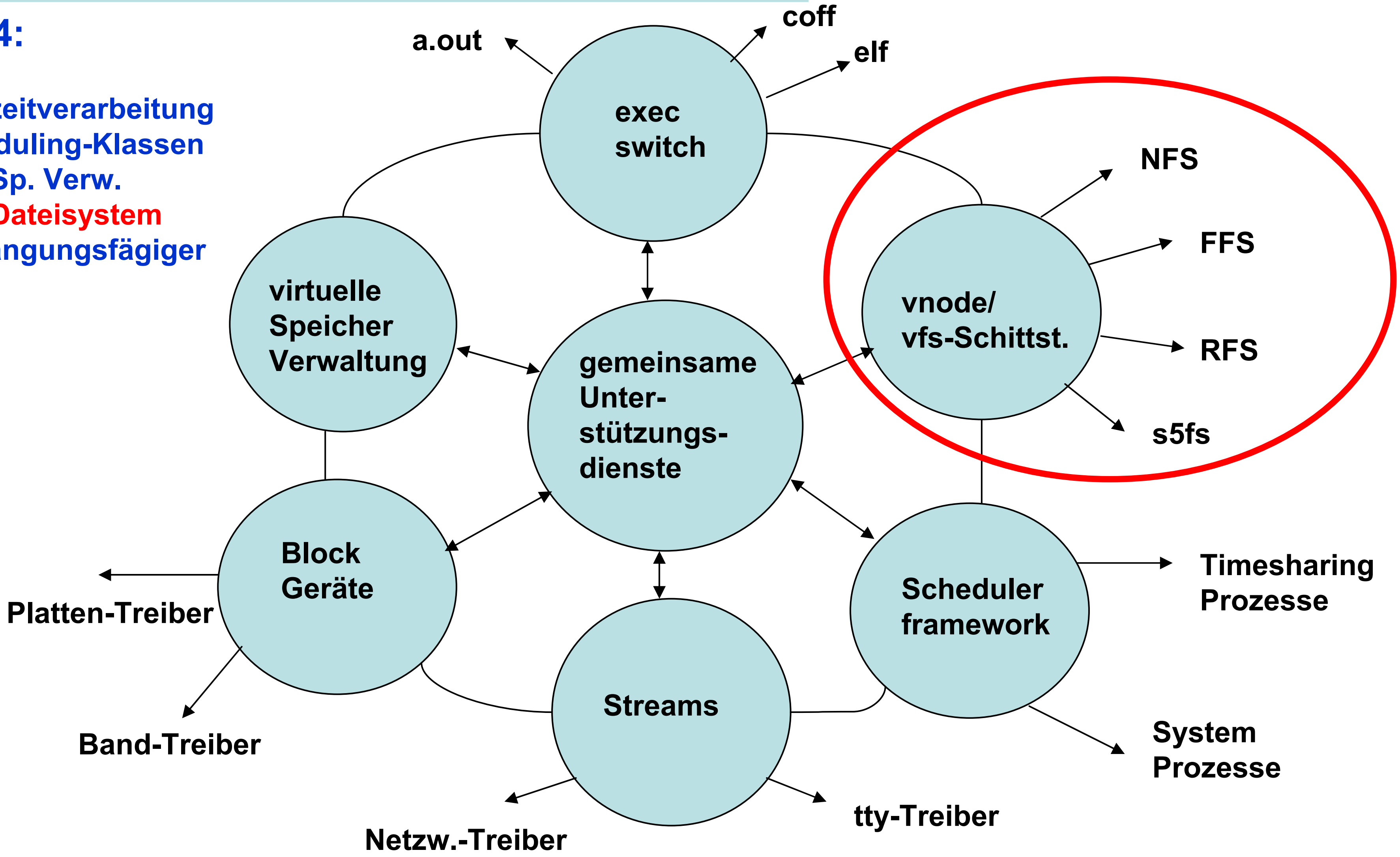
capacity of UNIX file

direct	10 blocks	10 K
single indir.	256 blocks	256 K
double ind.	64K blocks	64 M
triple ind.	256x64K blocks	16 G

Moderner Unix-Kern (Vahalia 1996)

SVR4:

- Echtzeitverarbeitung
- Scheduling-Klassen
- Virt. Sp. Verw.
- Virt. Dateisystem
- verdängungsfägiger Kern



Windows 2k File System

Windows 2k supports 3 File Systems for compatbility reasons:

FAT 16	(partitions ≤ 2G)
FAT 32	
NTFS	(NT File System)

useful website: http://linux-ntfs.sourceforge.net/ntfs/index.html

main features of NTFS

- ☆ Recoverability after system crashes (including fault-tolerance features)
- ☆ Protection and security
- ☆ Very large disks and very large files
- ☆ Multiple datastreams (which can be addressed under a single file name)
- ☆ General indexing possibilities (acc. to file attributes)

main features of NTFS files

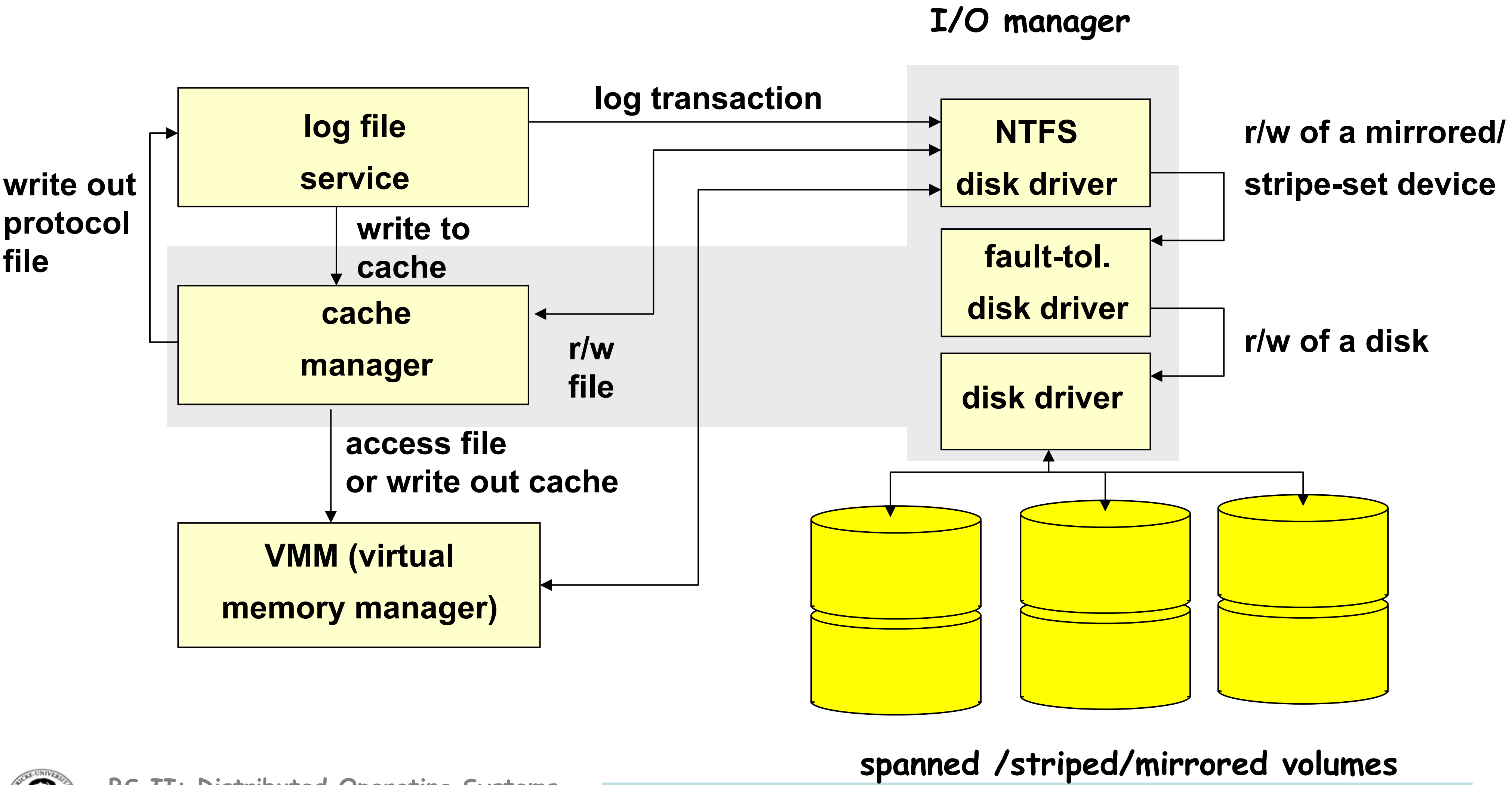
NTFS supports sophisticated naming of files

- long (255 character) file names
- pathnames up to 32767 characters
- unicode representation

NTFS files are not simple byte streams, but..

- comprise multiple byte streams (compatibility with Apple Macintosh FS)
- structured by attributes
- attributes represented by byte streams
- max stream length: 2^{64} bytes (18,4 Exabytes)

W2K components supporting NTFS



I/O manager

log file service

log transaction

NTFS disk driver

r/w of a mirrored/ stripe-set device

fault-tol. disk driver

r/w of a disk

disk driver

write out protocol file

write to cache

cache manager

r/w file

access file or write out cache

VMM (virtual memory manager)

spanned /striped/mirrored volumes

Spanned Volumes:

Logical partitions span multiple physical disks.

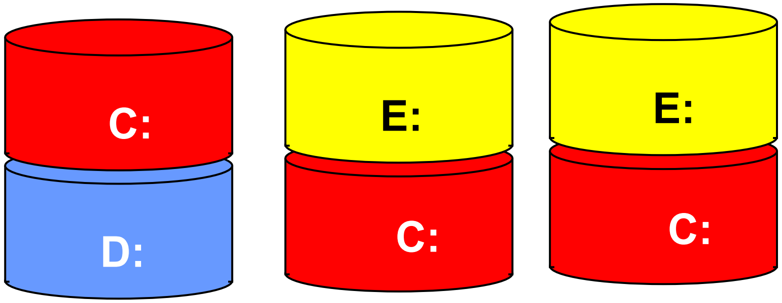
--> Logical Volumes

Motivations:

Volume larger than a physical disk

Transparent extensibility

Concurrent access to multiple physical disks improves performance



Striped Volumes:

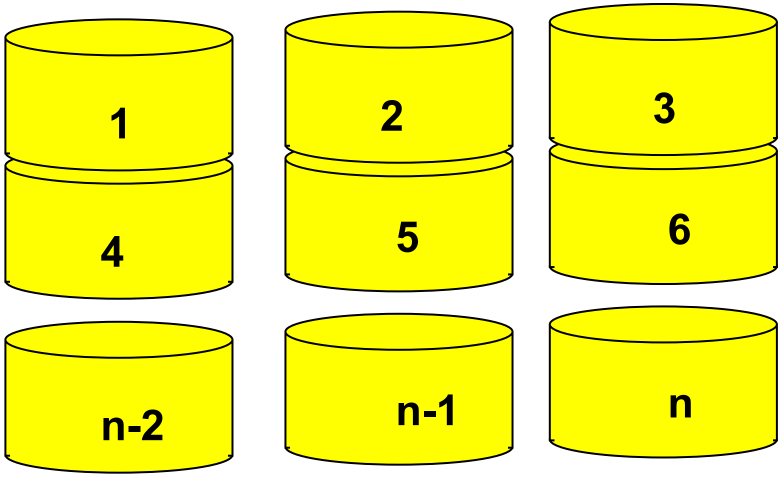
A physical disk drive includes multiple disks. Appears as a single disk for the operating system with improved performance and reliability.

Motivations:

Concurrent access to multiple physical disks improves performance.

Redundant Array of (inexpensive) independent disks for FT.

(RAID-1, RAID-5)



Win32 API

Important API functions for files:

Win32	Unix	
CreateFile	open	Create or open a file; Returns a handle
DeleteFile	unlink	Delete a file
CloseHandle	close	Close a file
ReadFile	read	Read data from file
WriteFile	write	Write data to file
SetFilePointer	lseek	position read pointer
GetFileAttributes	stat	Get File Attributes
LockFile	fcntl	Lock part of a file for multiple access
UnlockFile	fcntl	Release lock

Win32 API

Important API functions for directories:

Win32	Unix	
CreateDirectory	mkdir	Create a directory
RemoveDirectorY	unlink	Delete empty directory
FindFirstFile	opendir	Open directory and read entries
FindNextFile	readdir	Read next entry
MoveFile	rename	move file in another directory
SetCurrentDirectory	chdir	cange current working directory

NTFS basic concepts

Volume and File structure

Volume: Logical disk partition

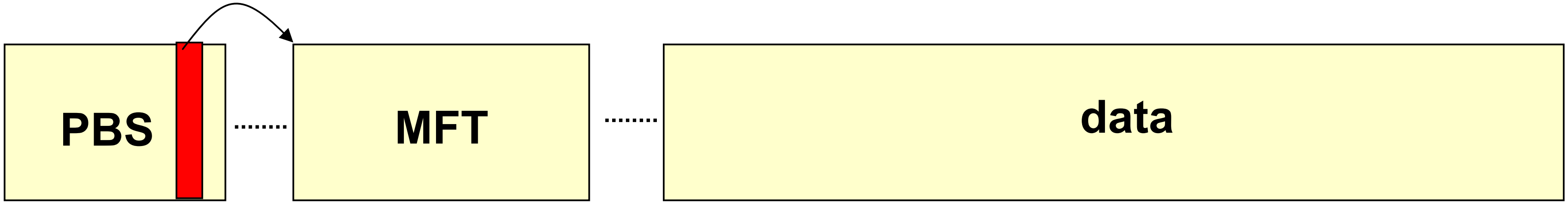
Sector: Smallest physical storage unit (most common size: 512 Byte)

Cluster: One or more consecutive sectors of the same track (corresp. to a block)

The Cluster is the basic unit of storage allocation in NTFS

Volume size	sector/cluster	cluster size
≤ 512 MB	1	512 Byte
512 MB - 1G	2	1 K
1-2 G	4	2 K
2-4 G	8	4 K
4-8 G	16	8 K
⋮	⋮	⋮
> 32 G	128	64K

NTFS volume structure



- PBS: Partition Boot Sector (up to 16 sectors)

- MFT: Master File Table
  - is a file that can be placed freely
  - contains meta data: MFT2, Log file, Cluster bit map,
    Attribute definition table, . . .
  - user file descriptors

Structure of NTFS

Master

File

Table

1 kB

:

16	1. user file	
15	reserved for future use	
14	reserved for future use	
13	reserved for future use	
12	reserved for future use	
11	$Extend	Extensions, Quotas etc.
10	$Upcase	Conversion table for uper/lower case letters
9	$Secure	Security Information
8	$BadClus	List of bad blocks
7	$Boot	Bootstrap Loader
6	$Bitmap	Bitmap of used/free blocks
5	$	Root directory
4	$AttrDef	Attribute specification
3	$Volume	Volume file
2	$LogFile	Log file for recovery
1	$MftMirr	Copy of MFT
0	$MFT	Master File Table

files holding meta-data (0–11)

Structure of NTFS

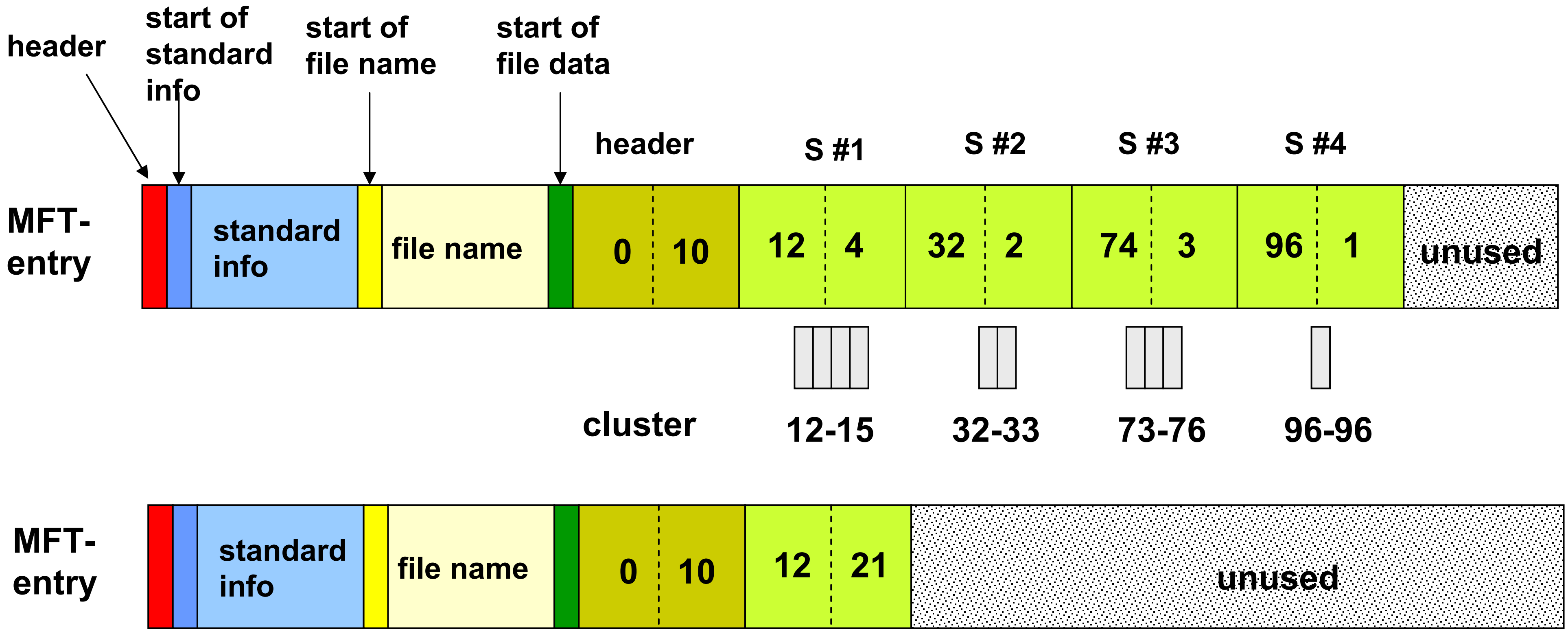
Usual attributes in MFT entries:

Default Information	Owner, protection info, time stamp, link counter, etc....
File name	file name in unicode
Security descriptor	(old) information now in $Extend and $Secure fields
Attribute list	Place wher additional MFT entries are stored if required
Object-ID	64 Bit file ID for internal use (unique for a volume)
Reparse	used for creating symbolic links
Volume name	used in $volume only
Volume attribute	used in $volume only
Index root	used for directories (called index in Microsoft terminology)
Index allocation	used for very large directories
Bitmaps	used for very large directories
Logging-support system	controls the logging in the log file
data	data stream

Structure of NTFS

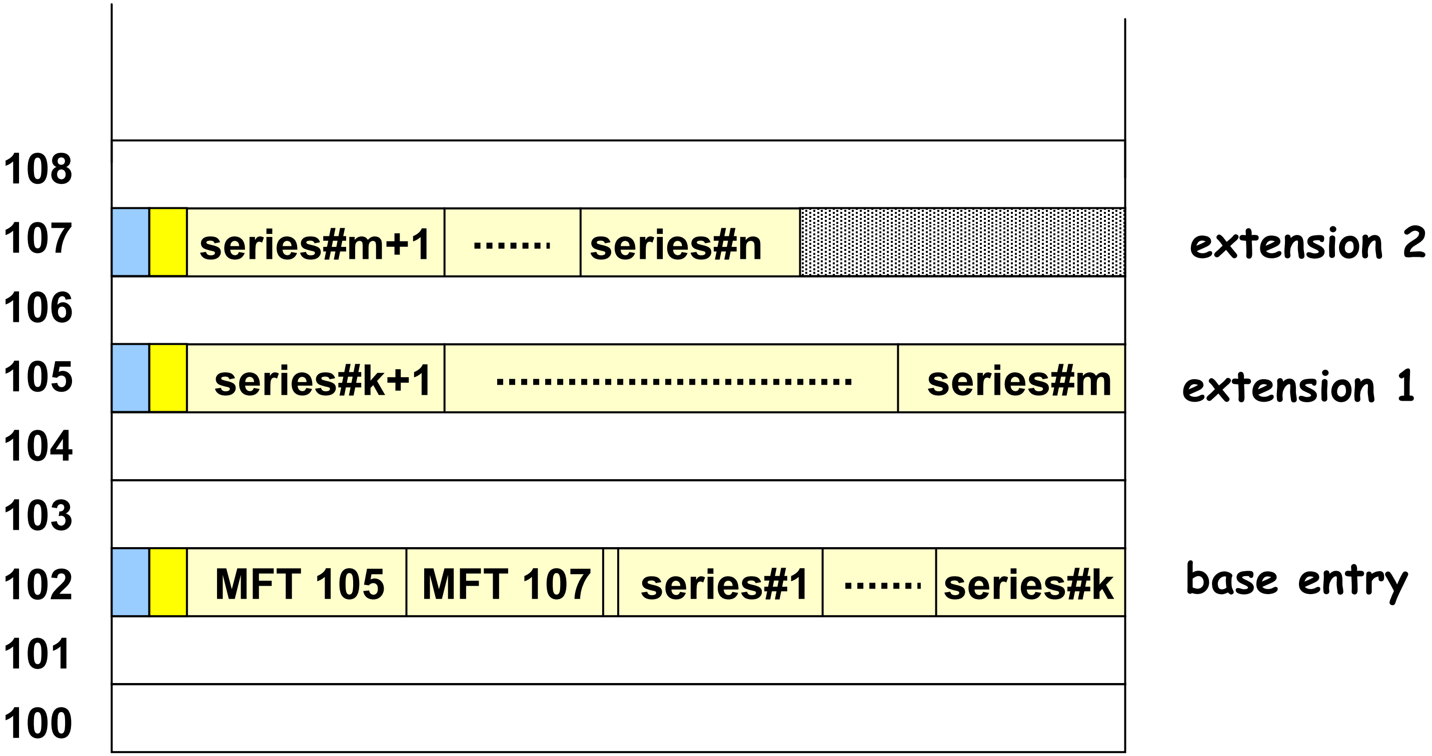
The "data" attribute: obviously, not all data fit in a single entry.

Problem: How to find the associated blocks (clusters)?



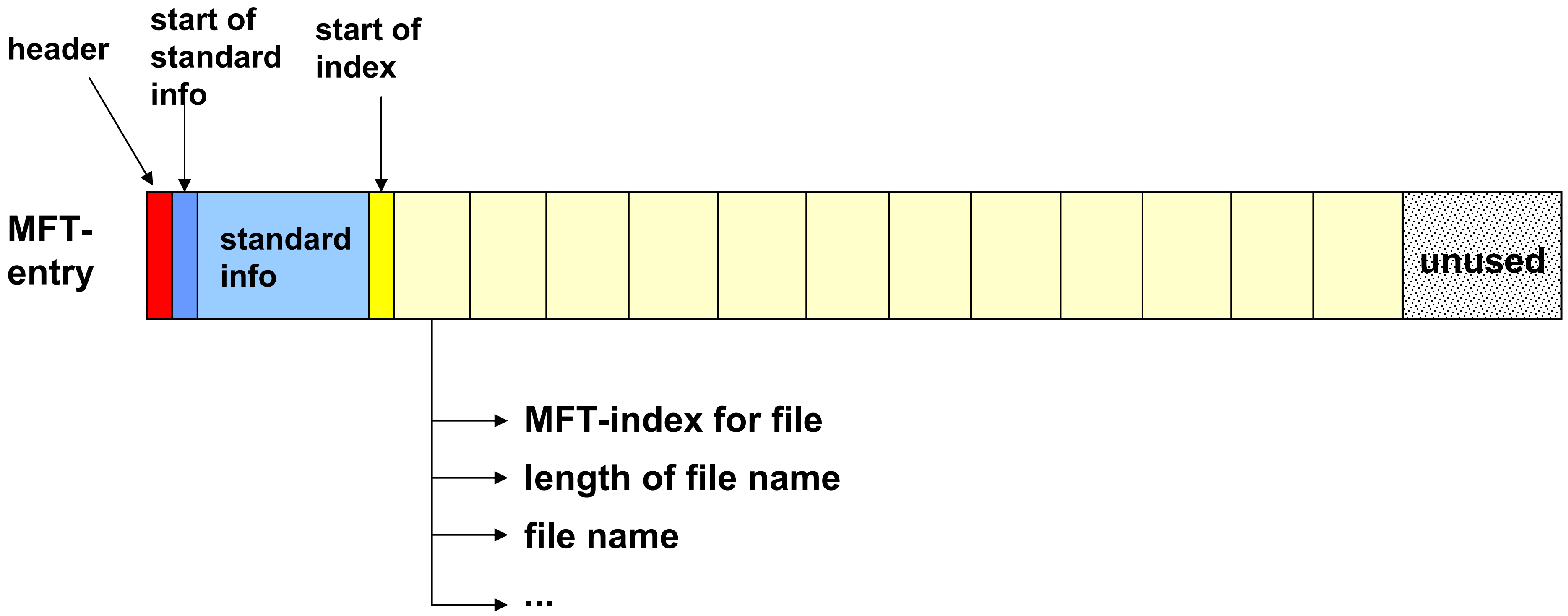
Structure of NTFS

Storing file clusters in multiple MFT entries



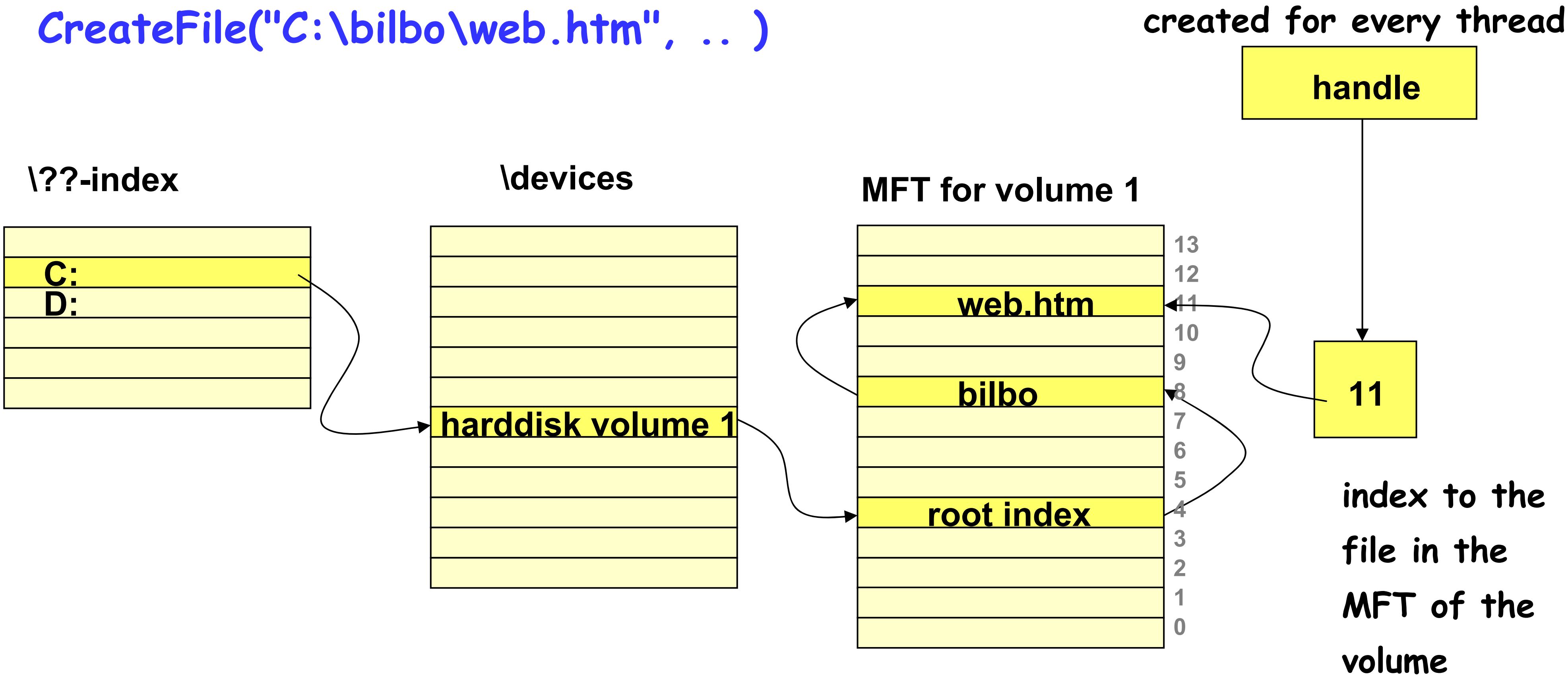
Structure of NTFS

MFT entry for a small index (directory)



Structure of NTFS

CreateFile("C:\bilbo\web.htm", ..)

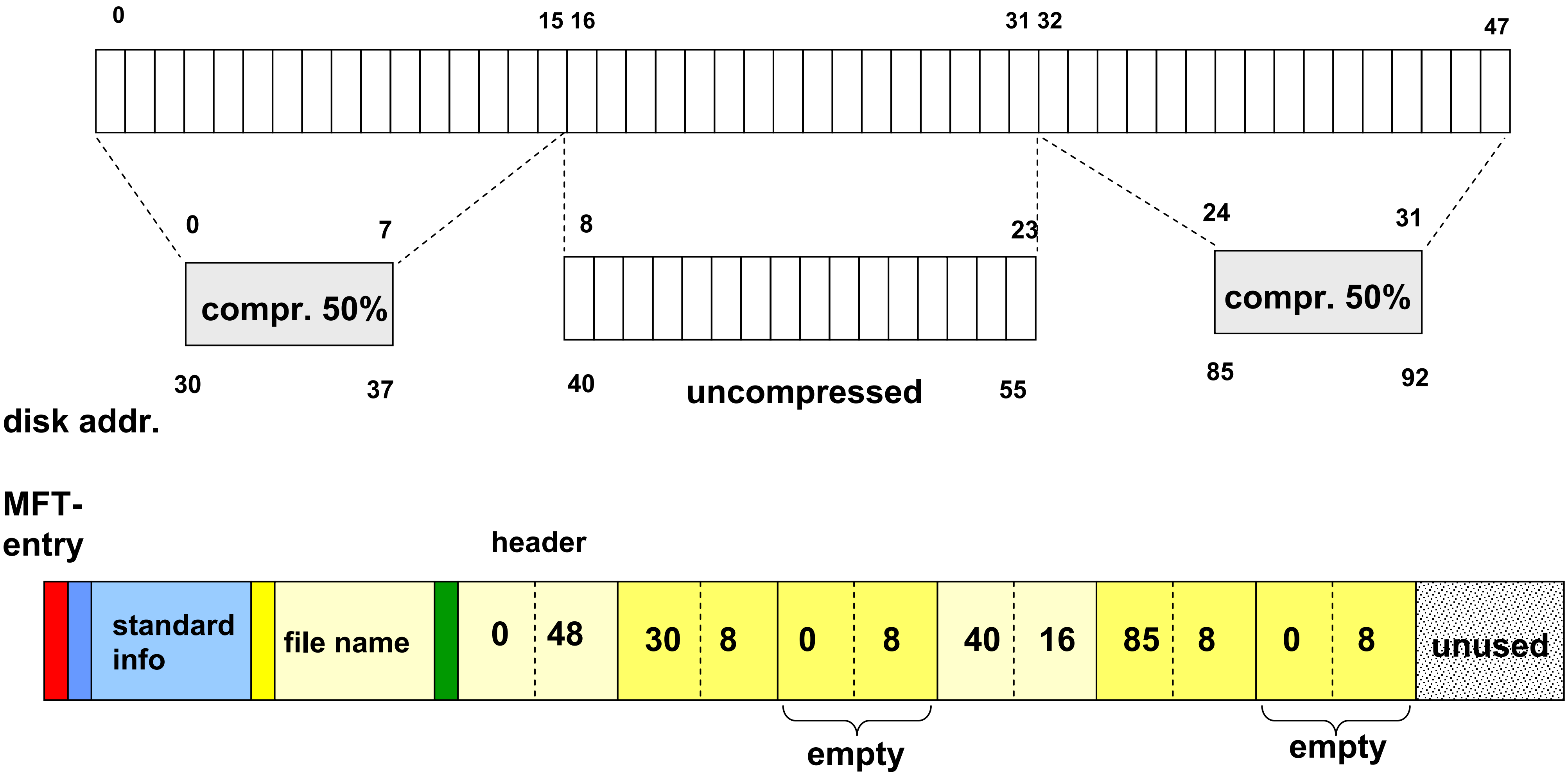


Structure of NTFS

More features:

- Compression of Addresses (16 --> 4)
- Compression of files
- Encrypted files
- Security and Access control

Compression of files

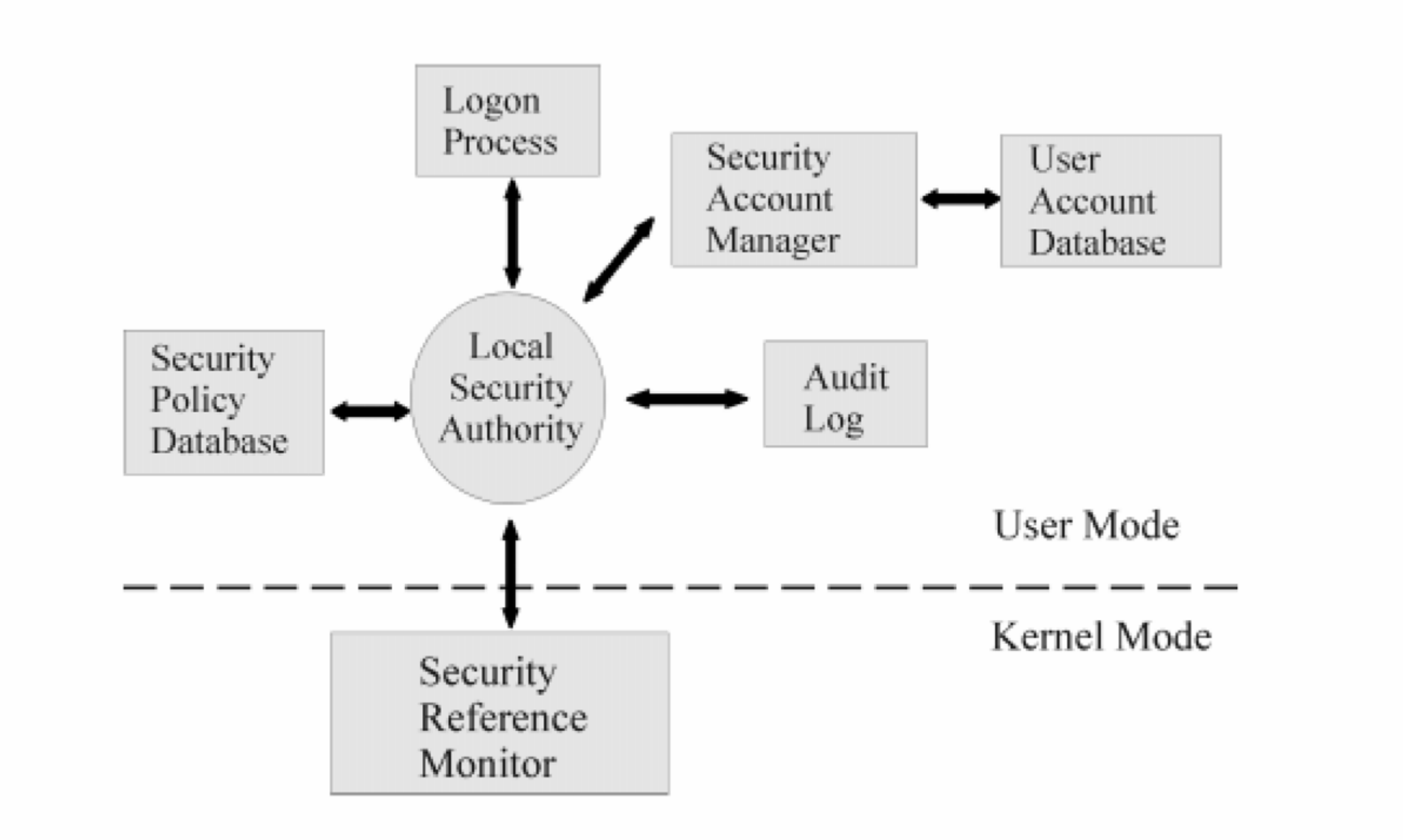


Security and access protection

- secure login and antispoofing
- discretionary access control
- privileged access control
- process address space protection
- prevention of data leaks by zeroing all new pages before loading
- security auditing

overall NT security model

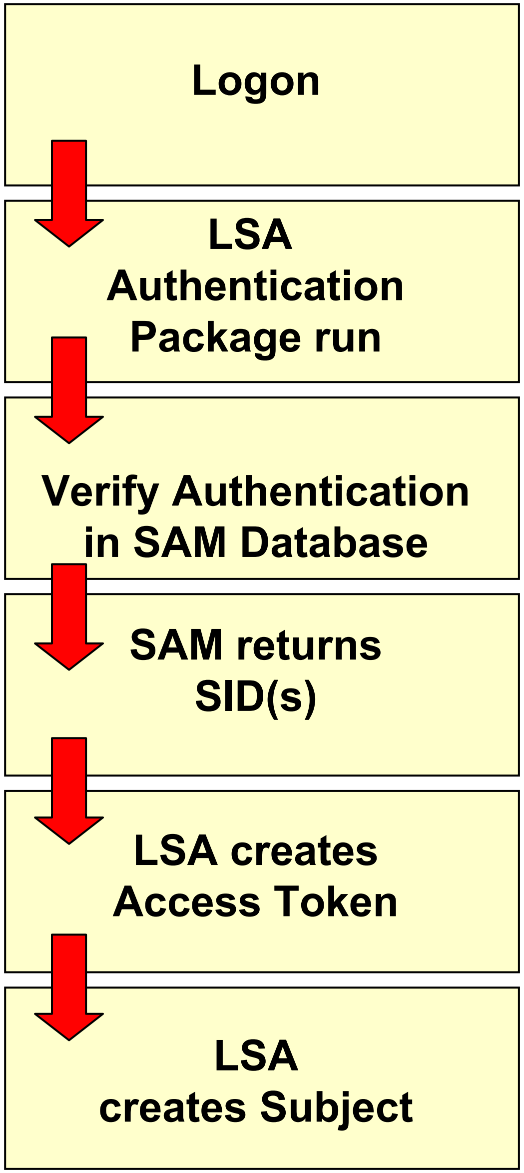
http://www.ciac.org/ciac/documents/CIAC-2317_Windows_NT_Managers_Guide.pdf



NT logon process

Windows NT logon processes provide mandatory logon for user identification and cannot be disabled.

To protect against spoofing, the logon process begins with a Welcome message box that requests the user to press Ctrl, Alt and Del keys before activating the actual logon screen.



Logon

LSA Authentication Package run

Verify Authentication in SAM Database

SAM returns SID(s)

LSA creates Access Token

LSA creates Subject

the access token

header	expir. time	groups	standard DACL	owner SID	group SID	restricted SIDs	privileges

Security ID (SID): The SID is a variable length unique name (alphanumeric character string) that is used to identify an object, such as a user or a group of users in a network of NT/2000 systems.

Expiration time: defines validity interval for the access token (currently not used)

Groups: defines to which group the process belongs (compatibility to Posix Standard)

Discretionary Access Control List (D ACL): Default ACL when they are created by a process and no other ACL is specified.

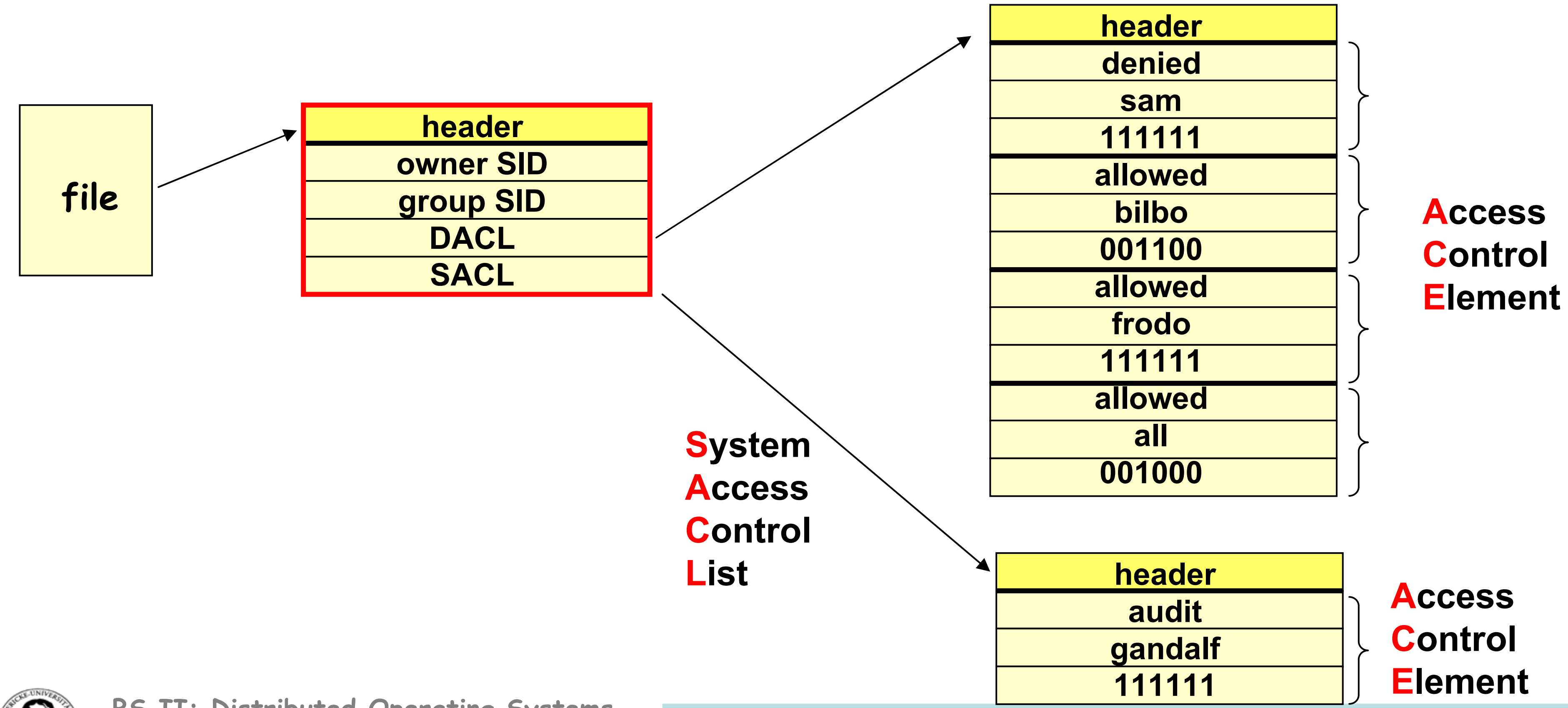
Owner/group SID: indicates the user/group who owns the process.

Restricted SID: enables the cooperation of trusted and non-trusted processes by contraining access for the latter.

Privileges: enable to define "admin rights" in a more fine-grained fashion and associate these with user processes.

the security descriptor

- is associated with every object
- defines who may access the object with which operation



file

header
owner SID
group SID
DACL
SACL

header
denied
sam
111111
allowed
bilbo
001100
allowed
frodo
111111
allowed
all
001000

Access
Control
Element

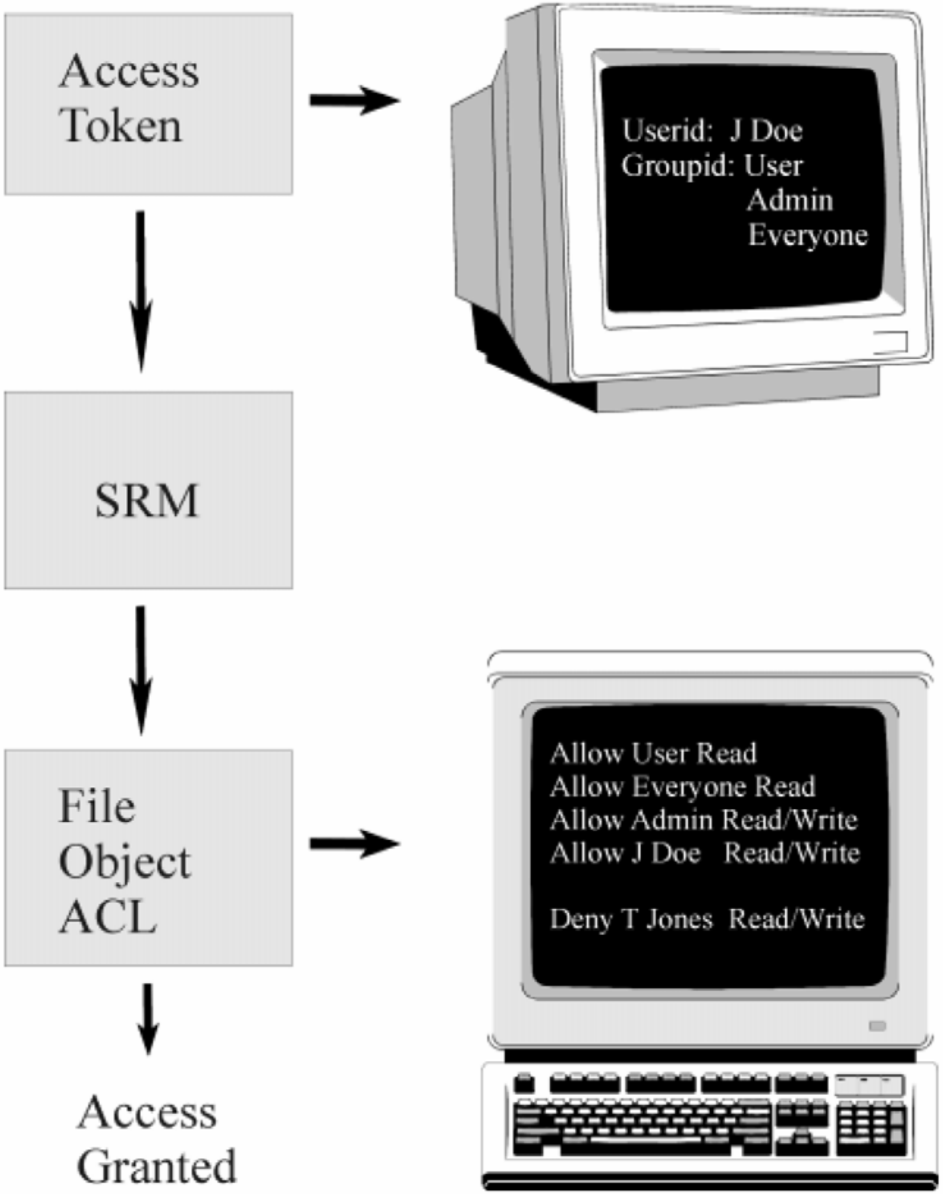
System
Access
Control
List

header
audit
gandalf
111111

Access
Control
Element

SRM access validation

SRM: Security
Reference
Monitor



Access Token

Userid: J Doe
Groupid: User
Admin
Everyone

SRM

File Object ACL

Allow User Read
Allow Everyone Read
Allow Admin Read/Write
Allow J Doe Read/Write

Deny T Jones Read/Write

Access Granted

Summary

- Files are protected by Access Control Lists.
- A security descriptor is associated with every file object comprising the security relevant information and references.
- An access token is associated with every active entity (subject) comprising authentication information and access control information.
- Every access to a shared object is verified by the Security Reference Monitor.