

Operating Systems

Timo Hönig

Bochum Operating Systems and System Software (BOSS)

Ruhr University Bochum (RUB)

VIII. File Systems

June 7, 2023 (Summer Term 2023)



RUHR
UNIVERSITÄT
BOCHUM

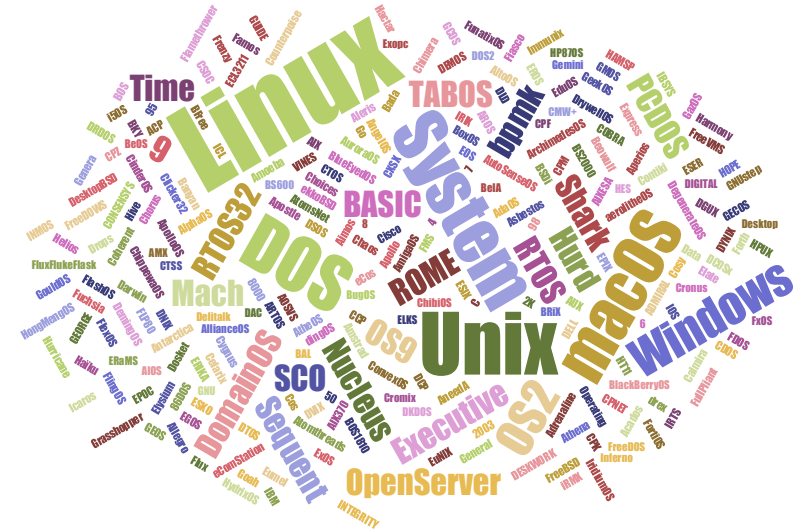
RUB

www.informatik.rub.de

Chair of Operating Systems and System Software

Agenda

- ▶ Recap
- ▶ Organizational Matters
- ▶ Motivation
- ▶ File Allocation Methods
 - ▶ Contiguous Allocation
 - ▶ Linked Allocation
 - ▶ Indexed Allocation
- ▶ File Systems
 - ▶ UNIX File System
 - ▶ Linux ext4
- ▶ Advanced Data Organization
 - ▶ Error Handling, Recovery, Performance
 - ▶ RAID, Journaling, Copy on Write
- ▶ Summary and Outlook



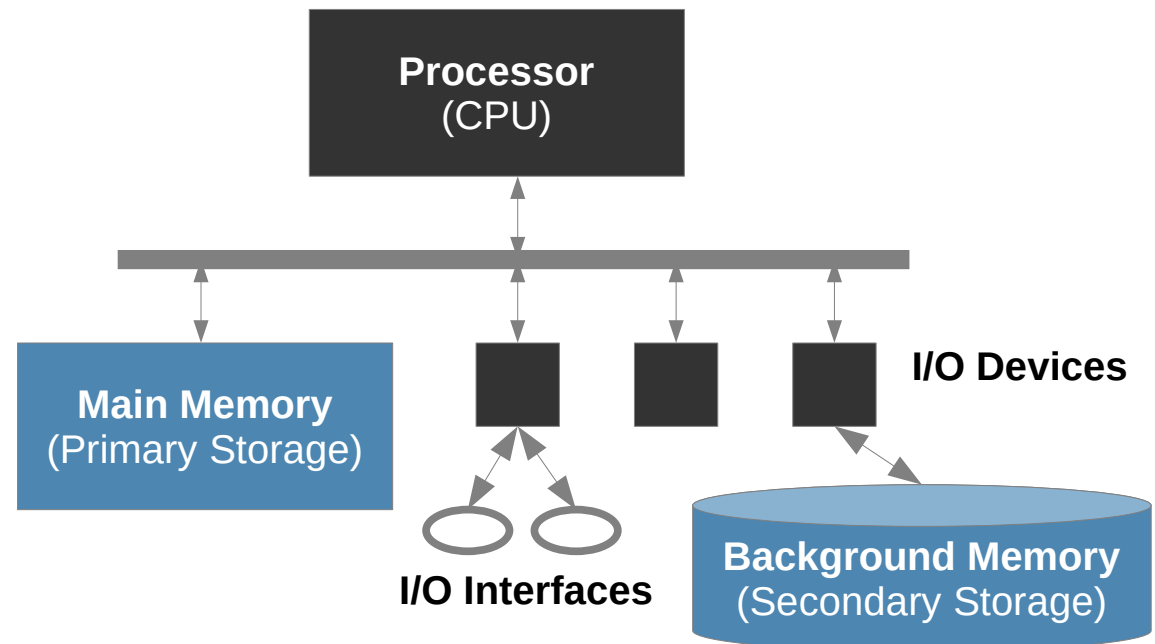
Literature References

Silberschatz, Chapter 13-15

Tanenbaum, Chapter 4

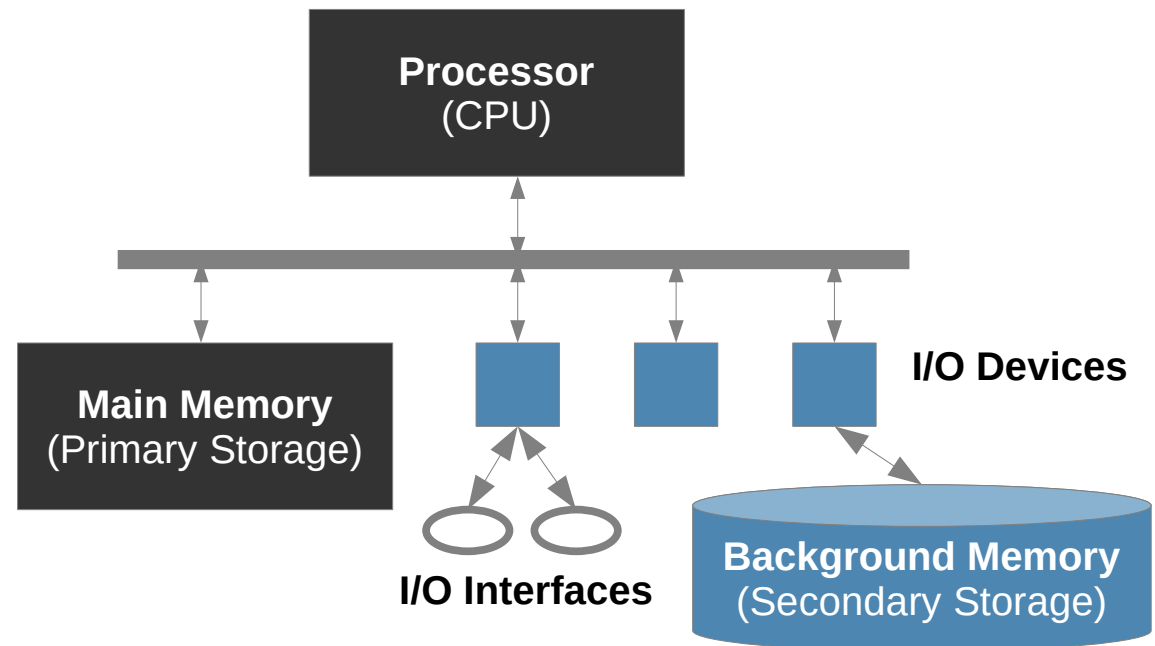
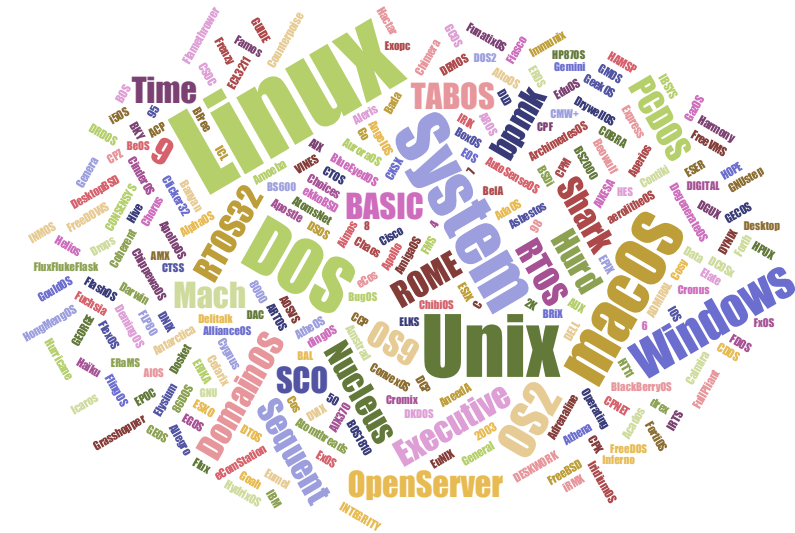
Recap

- memory management
 - operating system resource: byte-addressable main memory
 - paging, placement strategies
- virtual memory
 - demand paging
 - page replacement strategies
 - page frame allocation
- complexity and practical relevance



Recap

- memory management
 - operating system resource: byte-addressable main memory
 - paging, placement strategies
- virtual memory
 - demand paging
 - page replacement strategies
 - page frame allocation
- complexity and practical relevance

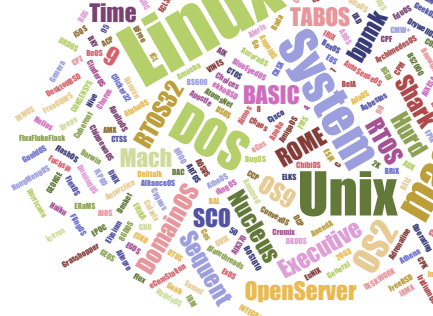


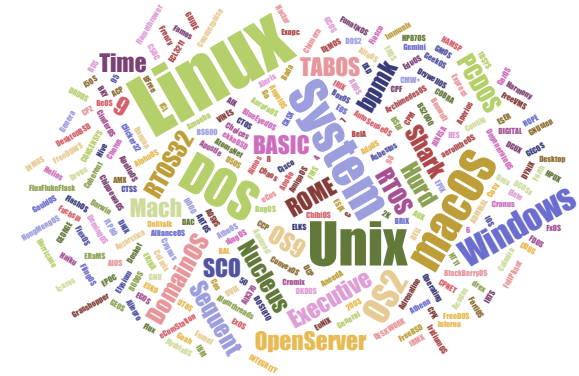
Agenda

- ▶ **Recap**
- ▶ **Organizational Matters**
- ▶ Motivation
- ▶ File Allocation Methods
 - ▶ Contiguous Allocation
 - ▶ Linked Allocation
 - ▶ Indexed Allocation
- ▶ File Systems
 - ▶ UNIX File System
 - ▶ Linux ext4
- ▶ Advanced Data Organization
 - ▶ Error Handling, Recovery, Performance
 - ▶ RAID, Journaling, Copy on Write
- ▶ Summary and Outlook



Organizational Matters

- lecture
 - Wednesday, 10:15 – 11:45
 - format: synchronous, **hybrid**
 - in presence (Room H1D, Building ID)
 - online lecture (Zoom)
 - **exam:** August 7, 2023 (first appointment)
September 25, 2023 (retest appointment)
 - exercises: **group allocation almost complete**
 - make use of group work – for your own benefit!
 - manage course material, asynchronous communication: Moodle
 - <https://moodle.ruhr-uni-bochum.de/course/view.php?id=50698>
- 



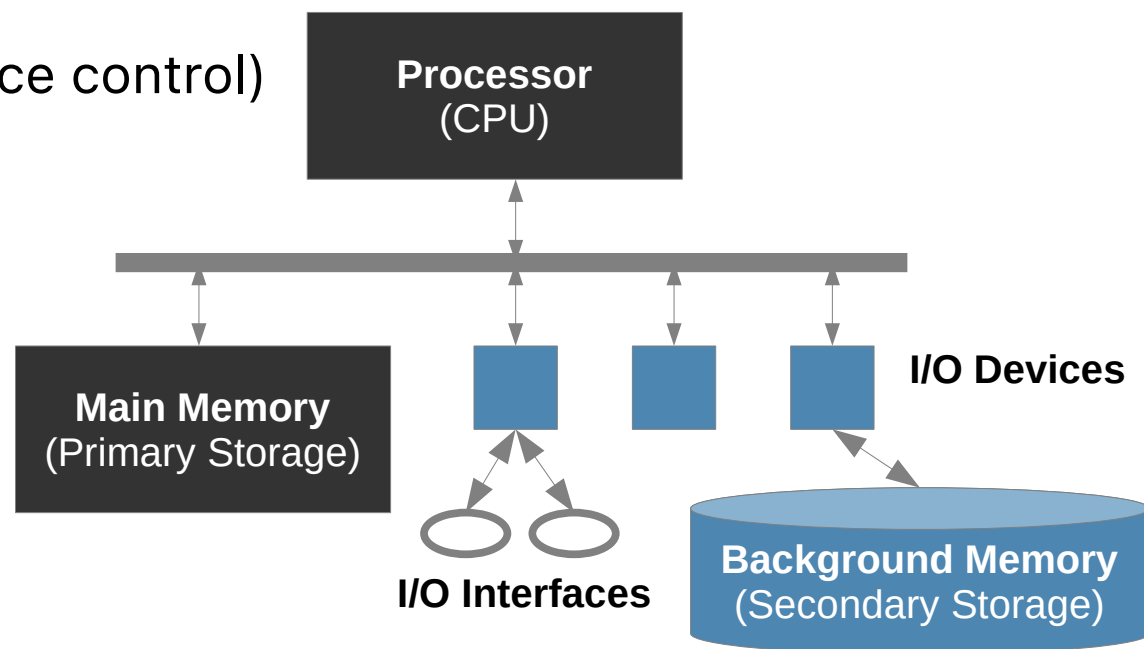
Agenda

- ▶ Recap
- ▶ Organizational Matters
- ▶ Motivation
- ▶ File Allocation Methods
 - ▶ Contiguous Allocation
 - ▶ Linked Allocation
 - ▶ Indexed Allocation
- ▶ File Systems
 - ▶ UNIX File System
 - ▶ Linux ext4
- ▶ Advanced Data Organization
 - ▶ Error Handling, Recovery, Performance
 - ▶ RAID, Journaling, Copy on Write
- ▶ Summary and Outlook

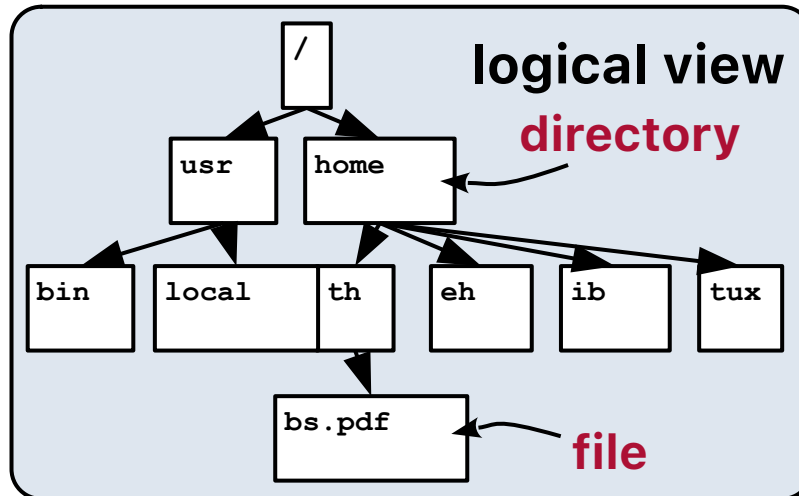


Motivation

- in the operating systems lectures so far
 - CPU, memory (i.e., main memory)
- this and next lecture
 - file systems (i.e., persistent storage on background memory)
 - input/output (i.e., device control)



File Systems on Background Memory



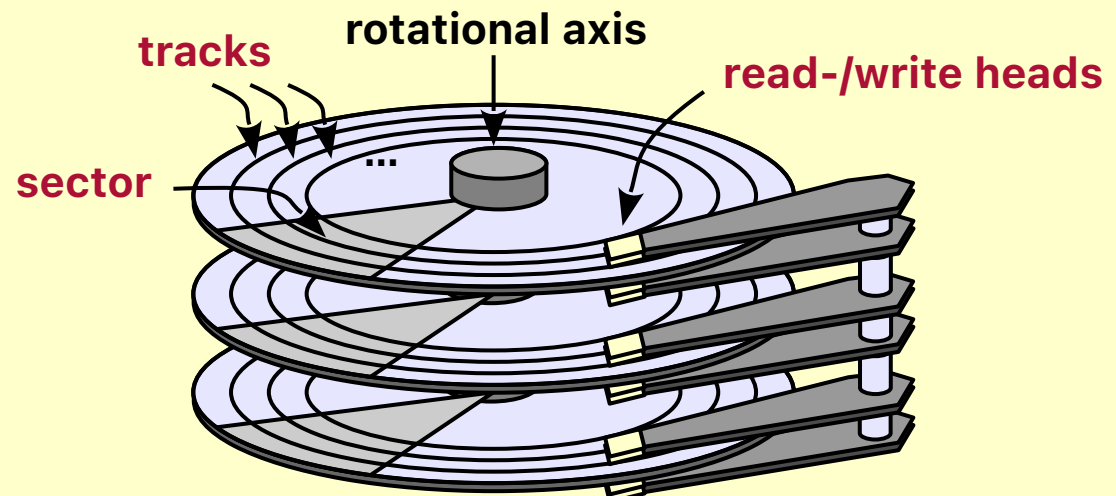
file systems enable the permanent storage of large amounts of data

mapping



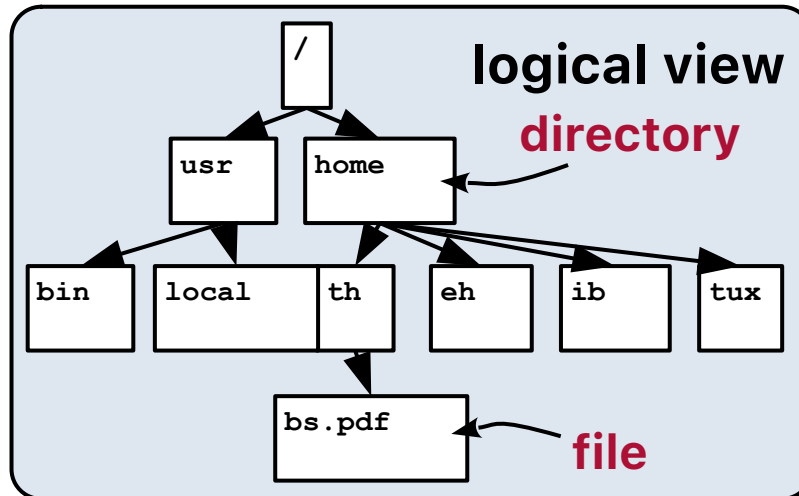
physical view

hard drive with 3 platters (6 surfaces)



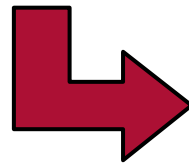
operating systems provide the applications with the logical view and must implement this abstraction efficiently

Data Storage with File Systems



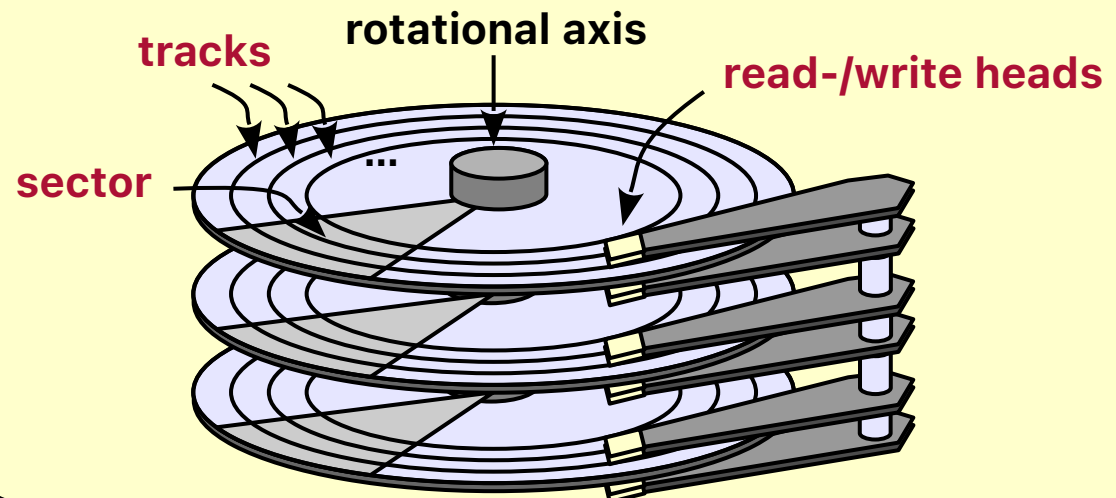
- storage: file system operates on data blocks (block size: 512...32678 bytes)

mapping



a sector is the smallest individual region on a disk that can be referenced (sector size: usually 512, 2048 or 4096 bytes)

physical view



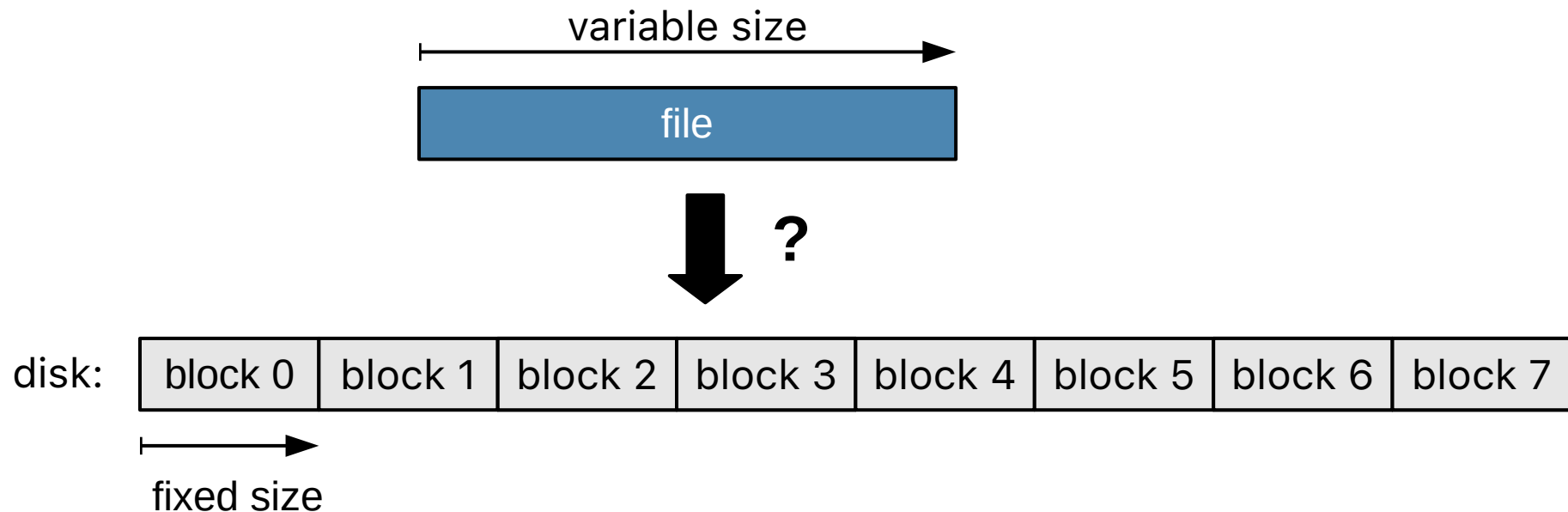
Agenda

- ▶ Recap
- ▶ Organizational Matters
- ▶ Motivation
- ▶ File Allocation Methods
 - ▶ Contiguous Allocation
 - ▶ Linked Allocation
 - ▶ Indexed Allocation
- ▶ File Systems
 - ▶ UNIX File System
 - ▶ Linux ext4
- ▶ Advanced Data Organization
 - ▶ Error Handling, Recovery, Performance
 - ▶ RAID, Journaling, Copy on Write
- ▶ Summary and Outlook



File Allocation

- data files often require more than one block on the hard disk
- data blocks typically span across multiple sectors
 - Which blocks are used to store a file?



logical view

physical view

Contiguous File Allocation

- file is stored in blocks with ascending block numbers
 - number of the first block and number of the following blocks must be stored, e.g., start: block 4; length: 3 blocks



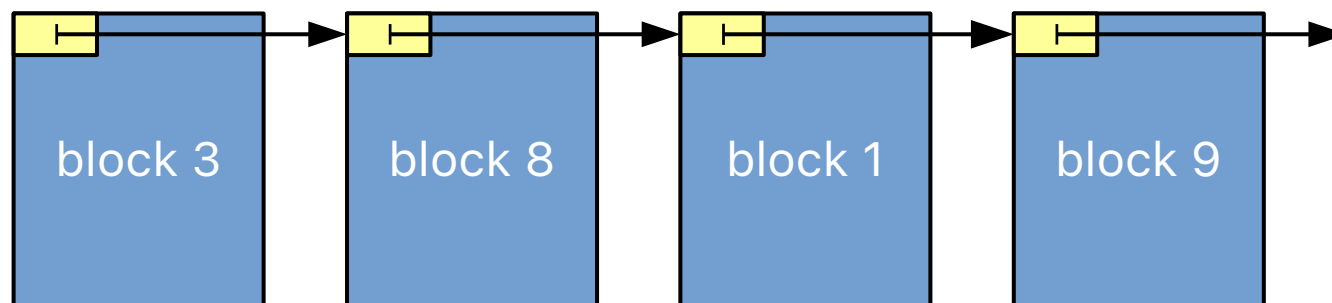
- advantages
 - access to all blocks with minimum positioning time of the arm and read/write heads
 - fast direct access to specific file position
 - particularly suitable for non-modifiable file systems (e.g., CDs/DVDs/Blu-ray discs)

Contiguous File Allocation – Issues

- **finding free space** on the hard disk
 - quantity of consecutive and free blocks on disk
 - cf. placement strategies → first fit, next fit, best/worst fit
- **fragmentation**
 - unusable blocks on hard disk as to fragmentation
 - removal of files → (too) small holes
- **size of new files** often unknown in advance
- **certain operations are problematic** (e.g., appending)
- **recopy** if no more free adjacent block area available

Linked File Allocation

- blocks of a file are linked



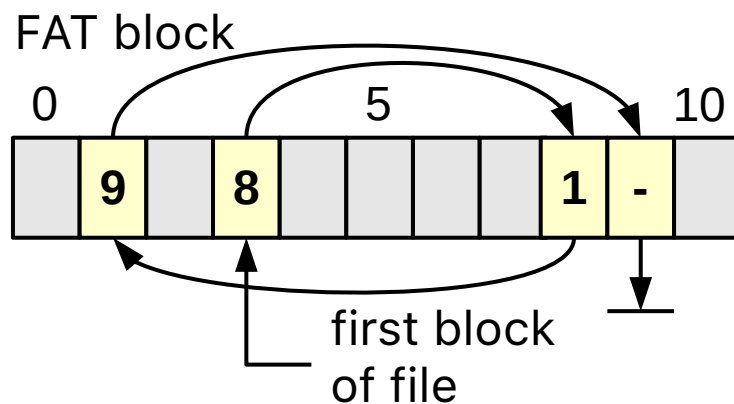
- e.g., Commodore systems (CBM 64)
 - block size 256 bytes
 - the first two bytes denote track/sector of the next block
 - if track number is zero: last block
 - 254 bytes of user data
- file can dynamically be reduced and increased in size

Linked File Allocation – Issues

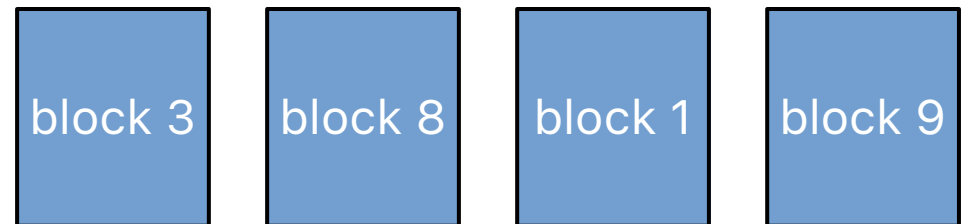
- **linkage: storage demand** reduces user data for each block
 - unfavorable in the context of paging: page would always consist of parts of two disk blocks
- **error-prone**: file is not restorable, if linkage becomes faulty
 - single bit error may destroy file
- **poor direct (random) access** to specific file location
- **frequent positioning** of the read/write head with scattered data blocks – even for sequential data access

Linked File Allocation with FAT

- file links are stored in separate disk blocks
 - file allocation table (FAT) approach (e.g., MS-DOS, Windows 9x)



file blocks: 3, 8, 1, 9



- advantages
 - entire block can be used as user data
 - redundancy: file allocation table can be stored multiple times to reduce errors

Linked File Allocation with FAT – Issues

- **additional loading** of at least one block (caching of the FAT necessary to increase efficiency)
- **loading unnecessary information:** file allocation table contains links for all files
- **time-consuming search** for the associated data block if the position in the file is known
- **frequent positioning** of the read/write head with scattered data blocks

Discussion: Chunks/Extents/Clusters

■ variants

- dividing a file into continuously stored **sequences of blocks** (called a chunk, extent, or cluster), for example: 16 blocks à 512 bytes = 8kB
- reduces the number of positioning operations
- search time for blocks improves as a function of chunk size

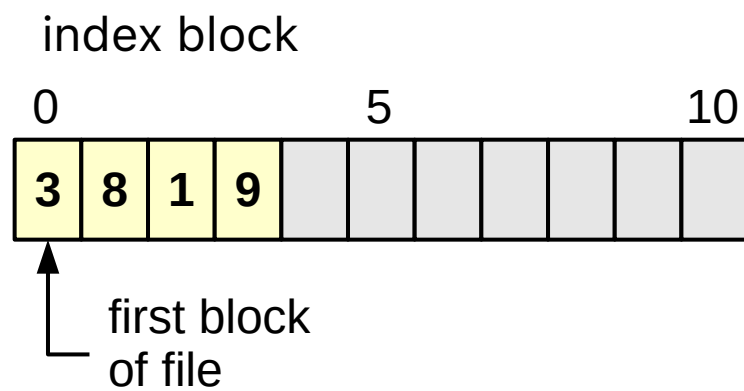
■ problems

- additional management information
- fragmentation
 - ➔ fixed size: within a block sequence (internal fragmentation)
 - ➔ variable size: outside the block sequences (external fragmentation)

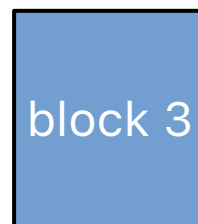
➔ variants are being used, but do not bring fundamental improvement

Indexed File Allocation

- special disk block contains block numbers for the data blocks of a file

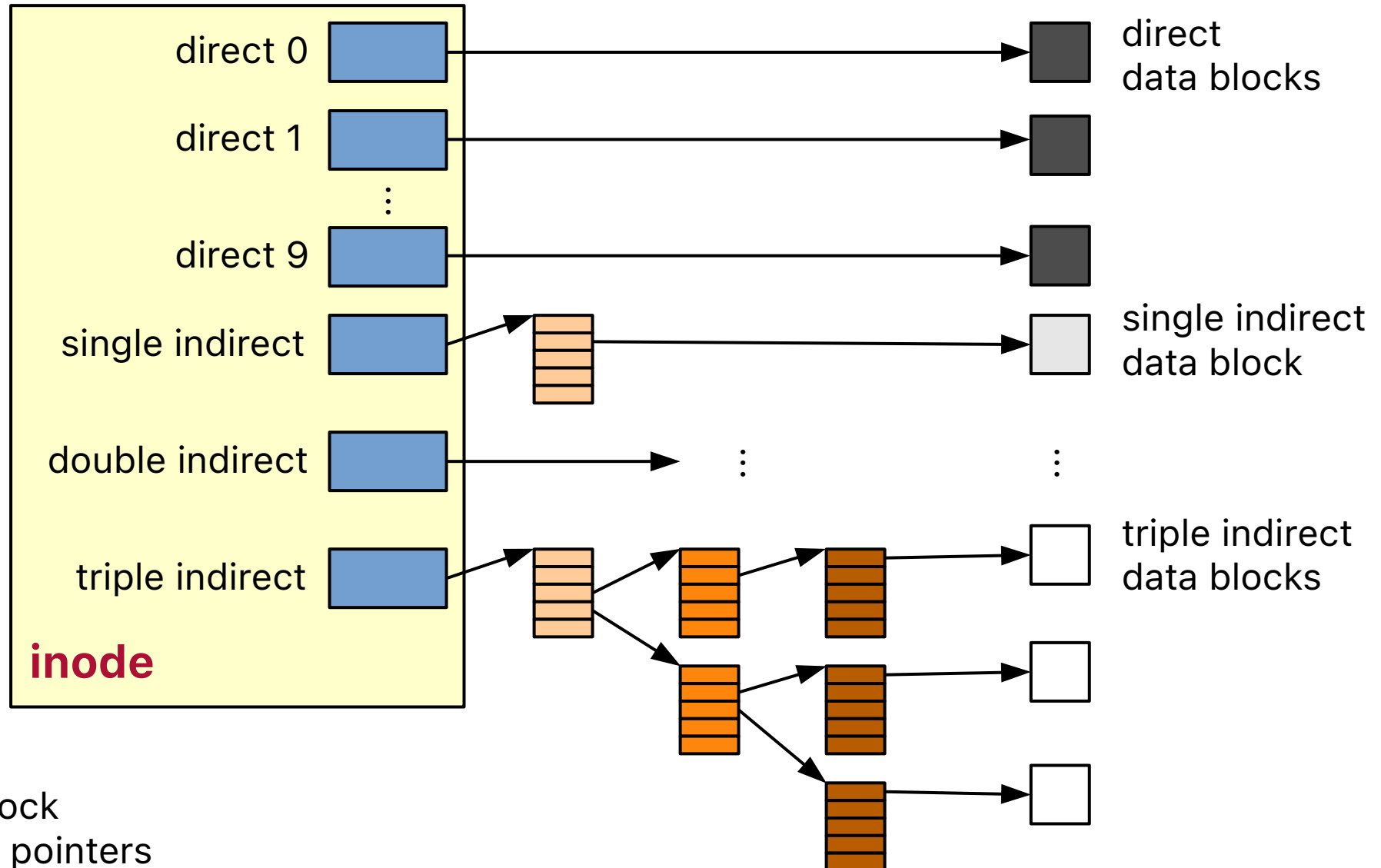


file blocks: 3, 8, 1, 9



- problem
 - fixed number of blocks in the index block
 - fragmentation for small files
 - (functional) extensions for large files mandatory

Indexed File Allocation – UNIX inode

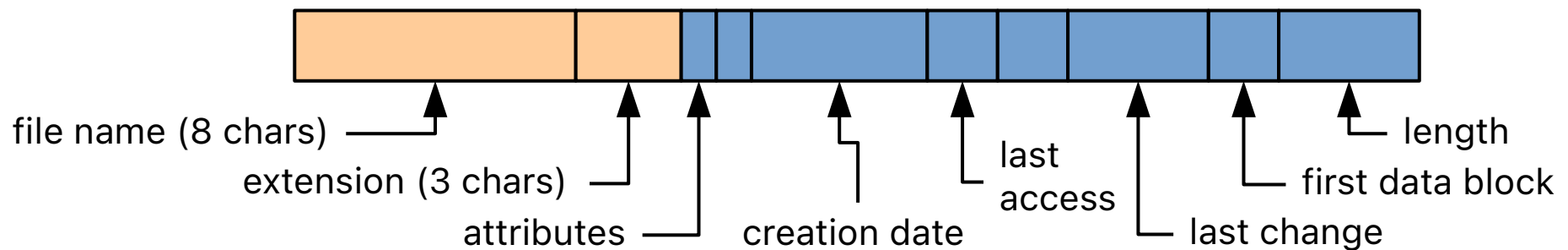


Indexed File Allocation – Discussion

- use of multiple levels of indexing
 - inode needs a block on the disk (fragmentation unproblematic with small files)
 - also (very) large files become addressable due to multiple levels of indexing
- disadvantage
 - several blocks must be loaded for indices (large files, only)

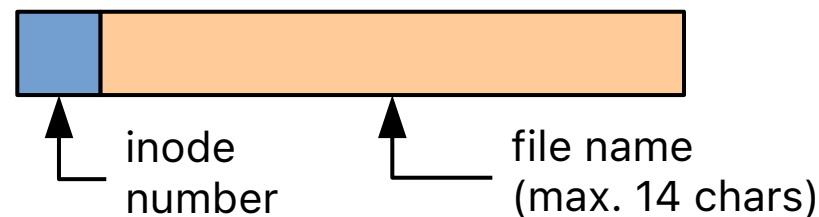
Files and Directories

- directory := list of files and subdirectories, file-system specific
- entries of the same length, one after the other in a list, for example:
 - **FAT based file systems**



→ for VFAT several entries are used together to accommodate long file names, too

- **UNIX System V.3**

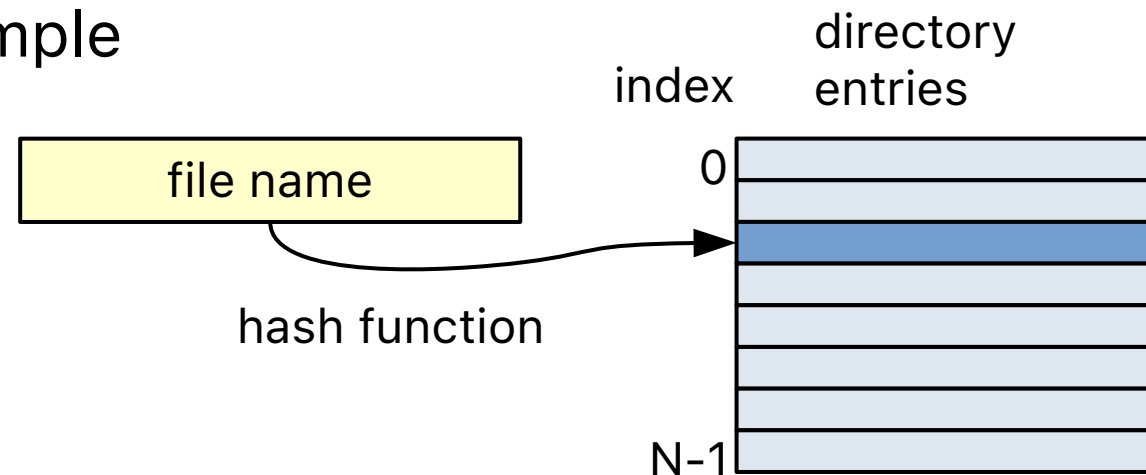


- **issues**

- linear search for specific entry
- when sorting the list: fast search, but high effort when inserting into list data structure

Files and Directories – Hashing

- hash function maps file/directory names to an index in the catalog list
- faster access to the entry (no linear search)
- simple (but naive) example
($\sum \text{chars}$) mod N



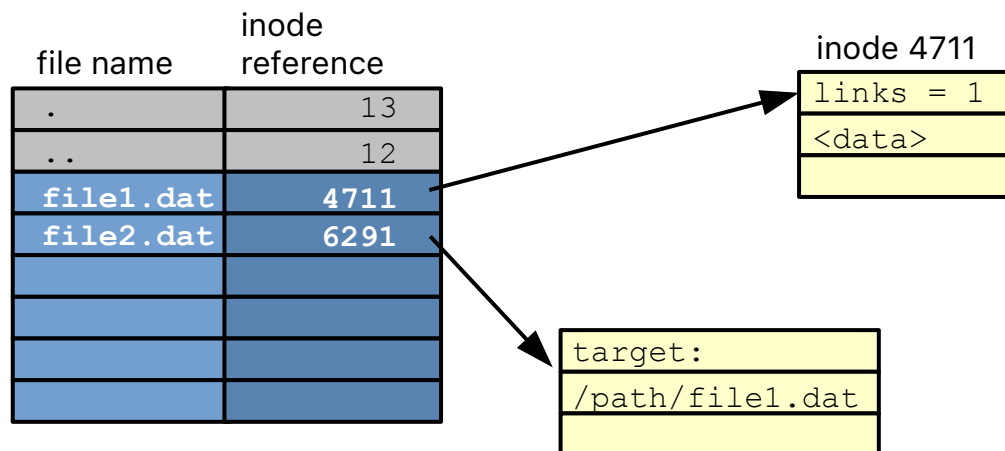
- **problems**

- hash collisions (multiple file names are mapped to a single entry in the catalog)
- adjust catalog size when catalog is full

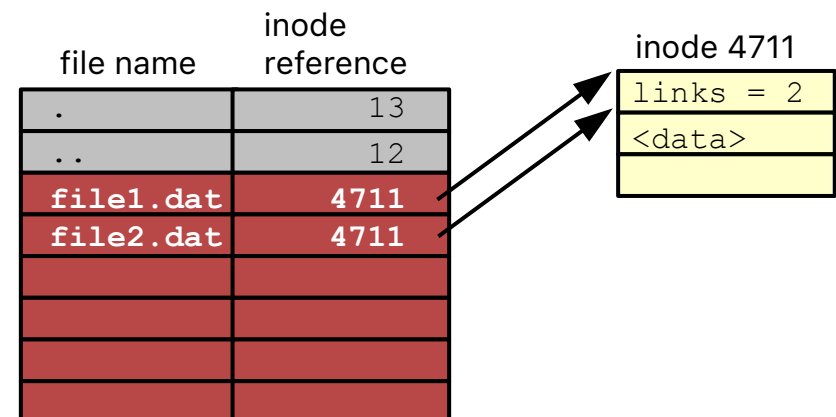
Files and Directories – Links

- symbolic links and hard links
 - symbolic links reference another file or directory using its path
 - hard links connect the hard link's name with the data of another file or directory

1. symbolic link

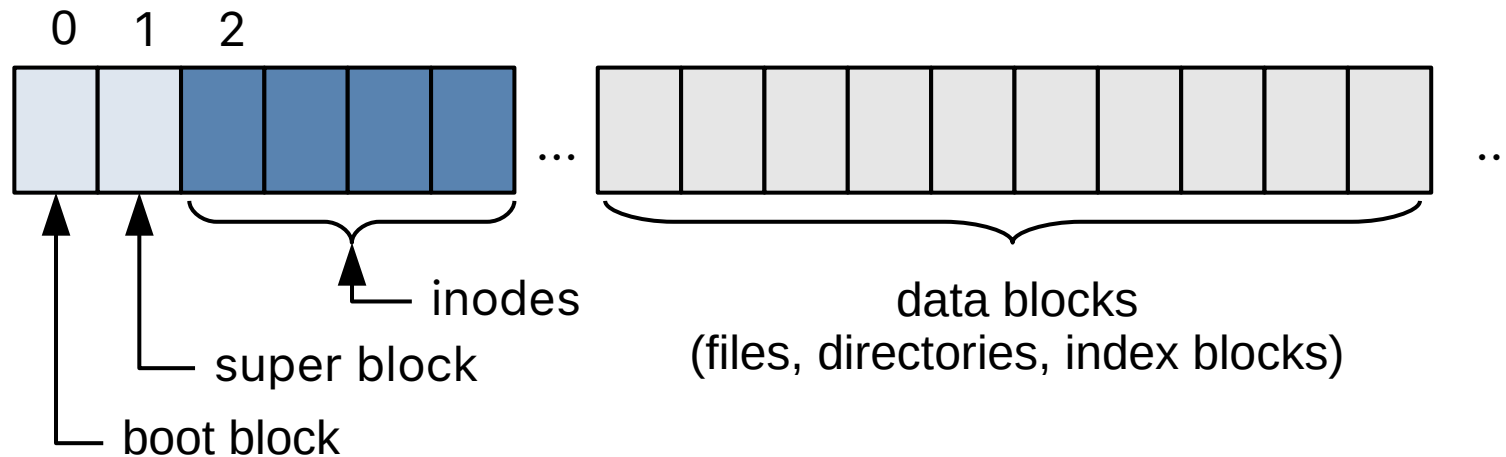


2. hard link



UNIX File System – System V

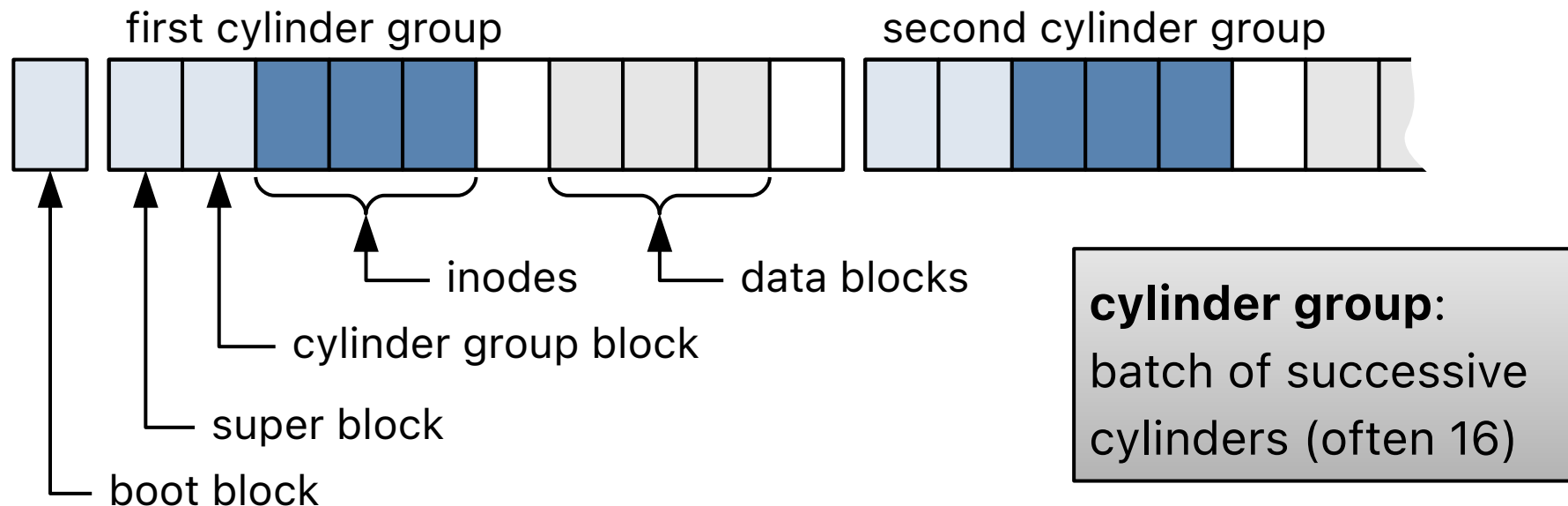
- organisation of blocks



- **boot block** contains information about loading operating system
- **super block** contains management information for the file system
 - block size, number of blocks, number of inodes
 - number and list of free blocks and free inodes
 - attributes (e.g., modified flag)

UNIX File System – Berkeley Fast File System

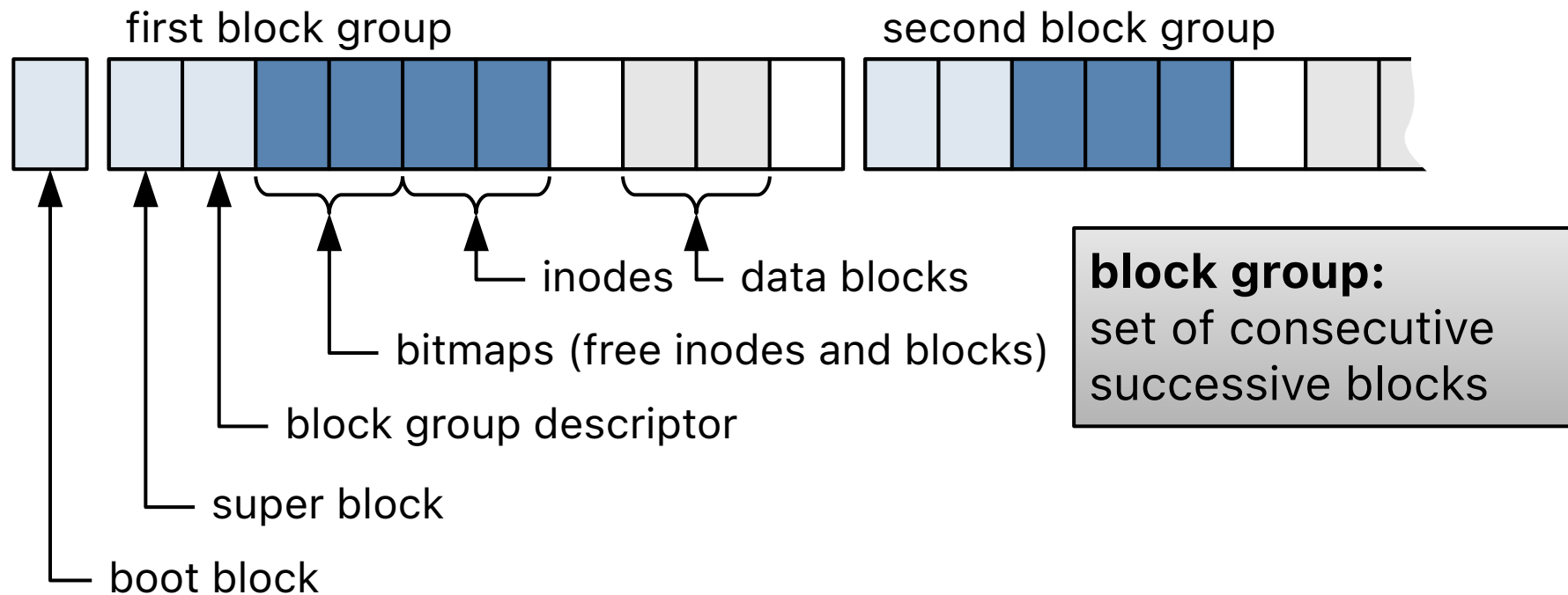
- organisation of blocks



- **copy** of the **super block** in each cylinder group
- a single file is stored within a cylinder group, if possible
- distribution of directories, files of a single directory are grouped together
- advantage: shorter positioning times, higher throughput

Linux Ext4 File System

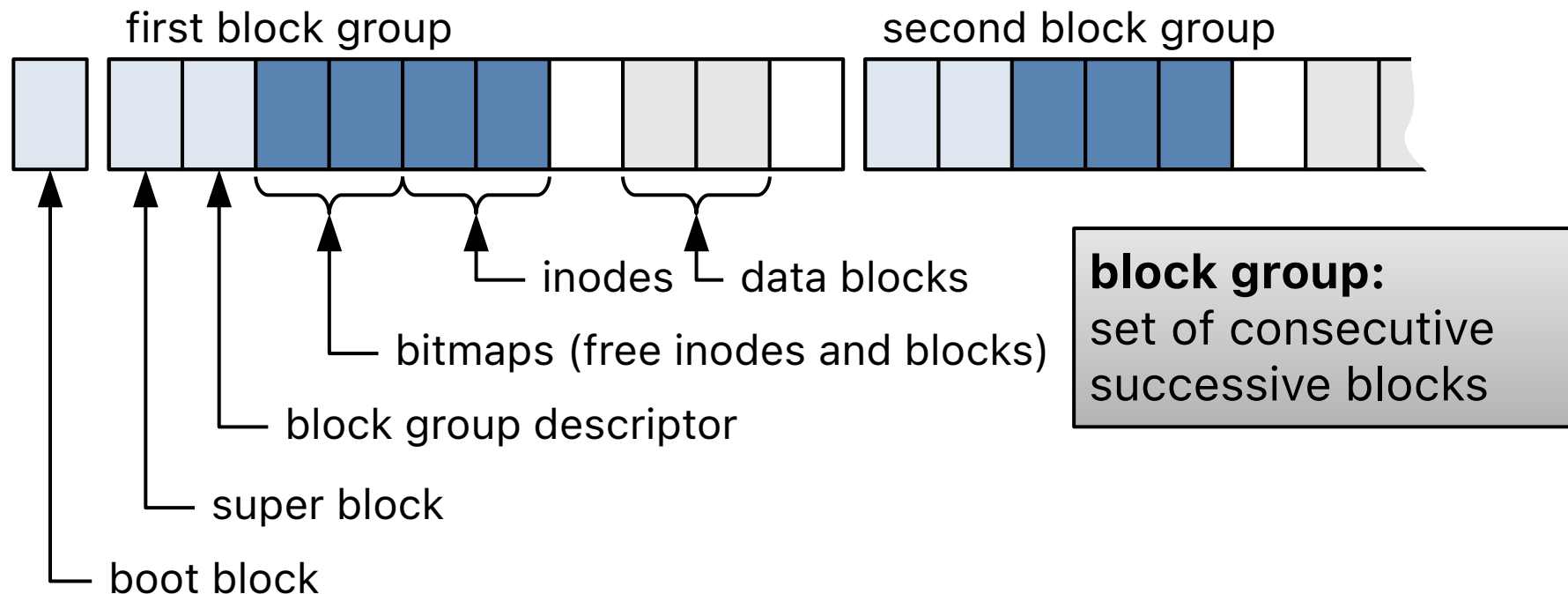
■ organisation of blocks



- similar layout as Berkeley Fast File System
- block groups are independent from cylinders

Linux Ext4 File System

■ organisation of blocks



■ specs

- Ext4 file system size up to 1 exabyte (EB) = 1.000.000 terabytes (TB)
- maximum file size is 16 TB, maximum of 4 billion files

Agenda

- ▶ Recap
- ▶ Organizational Matters
- ▶ Motivation
- ▶ File Allocation Methods
 - ▶ Contiguous Allocation
 - ▶ Linked Allocation
 - ▶ Indexed Allocation
- ▶ File Systems
 - ▶ UNIX File System
 - ▶ Linux ext4
- ▶ Advanced Data Organization
 - ▶ Error Handling, Recovery, Performance
 - ▶ RAID, Journaling, Copy on Write
- ▶ Summary and Outlook



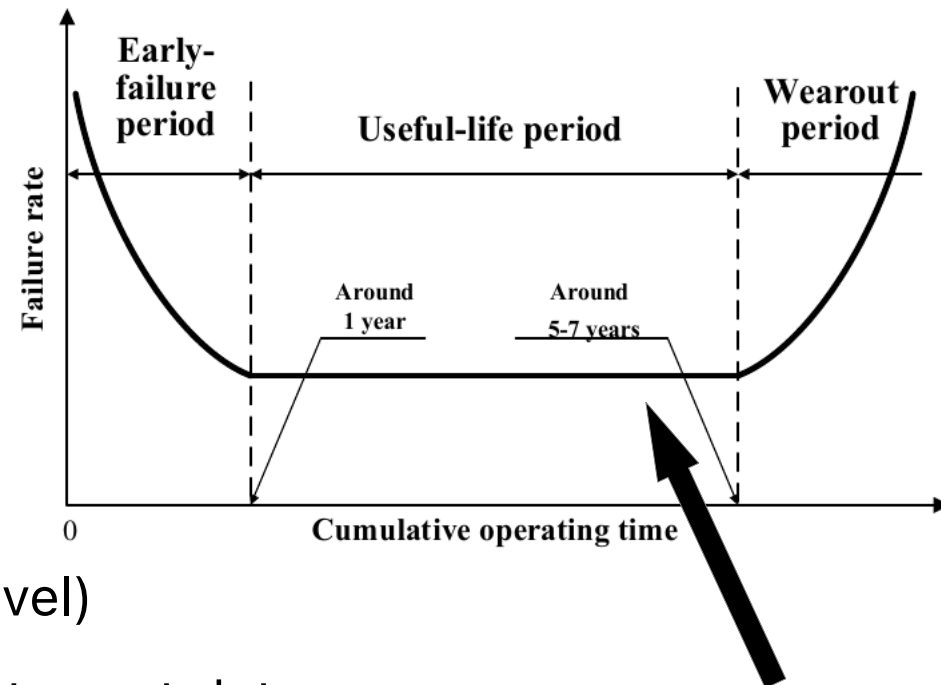
Error Handling and Recovery

■ problems

- block or hard drive error
- computer crash or failure

■ results and effects

- complete data loss
- data corruption (e.g., at block level)
 - applications can no longer interpret data
- inconsistent meta data
 - catalog entry missing to file or vice versa
 - block is used but not marked as such



The "bathtub curve" shows how the error rate of hard disks (and most other technical products) typically develops over their lifetime.

Error Handling and Recovery

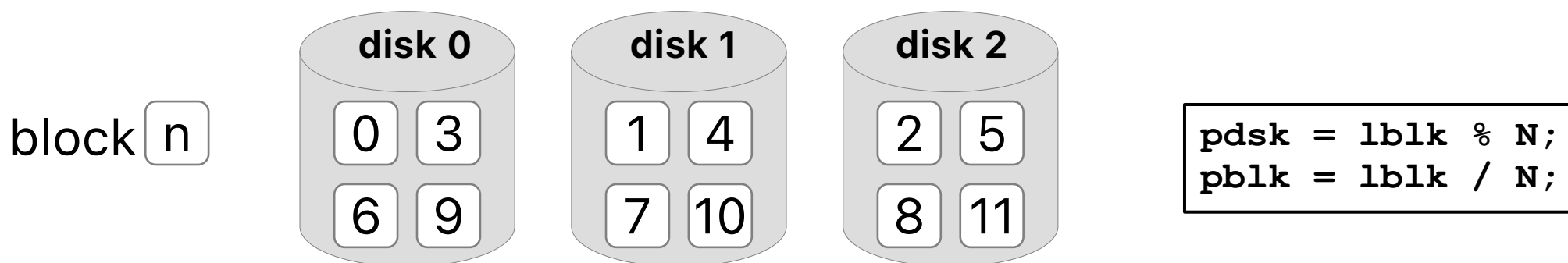
- **approach: backup**
 - regular incremental and full backup of data
 - issues: time and storage requirements
- **approach: checksums**
 - data can be provided with a checksum (detection) or error-correcting code (repair)
 - issues: runtime overhead, storage space requirements
- **approach: repair programs**
 - programs such as **chkdsk**, **scandisk** and **fsck** repair inconsistent meta data
 - issues: data loss possible during repair; long runtimes of repair programs with large disks

Redundant Arrays of Inexpensive Disks (RAID)

- **initial idea:** save costs by realizing large logical drives from cost efficient smaller drives (price per GiB)
- additional features:
 - improved utilization of the available data throughput due to parallel transfers
 - error tolerance with the help of redundancy
- two variants: hardware and software RAID
 - **hardware RAID**
 - ➔ disk controller with special management software
 - **software RAID**
 - ➔ software layer between disk driver and file system code

RAID Level 0 – Striping

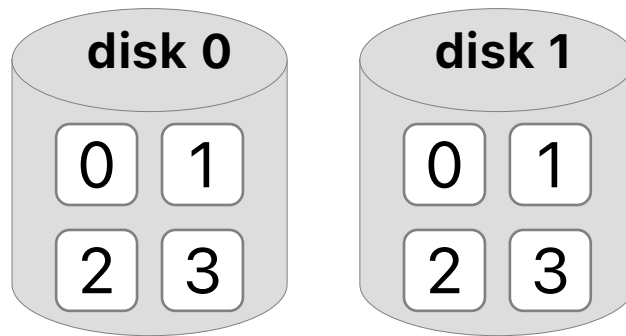
- **approach:** data of a large logical drive is stored in rows distributed over N physical disks:



- **effect:** higher read and write throughput, since several disks can operate in parallel
- **disadvantage:** error probability multiplies by N

RAID Level 1 – Mirroring

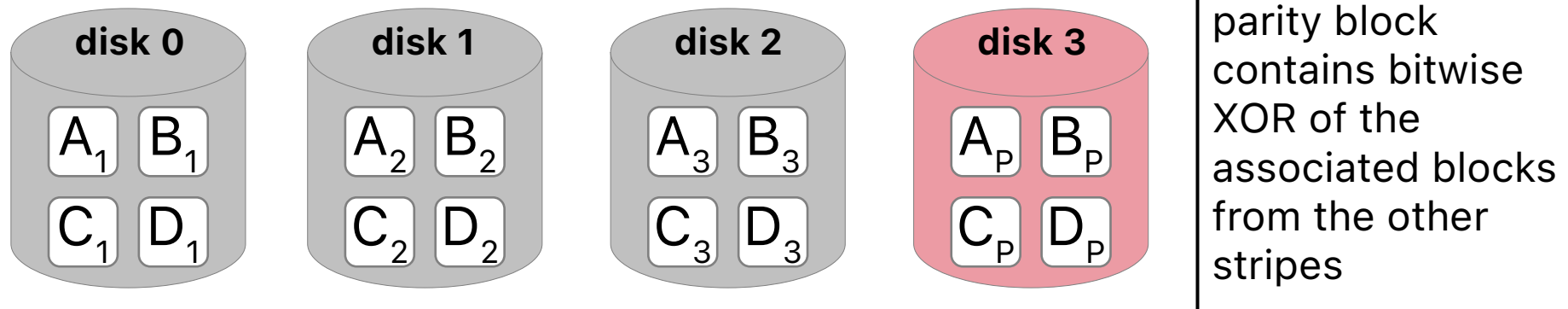
- **approach:** data is stored on two disks simultaneously (redundant storage):



- **effect:** higher read throughput, slightly slower write and increased resilience due to copy
- **disadvantage:** double storage space requirement

RAID Level 4 – Striping with Parity Disk

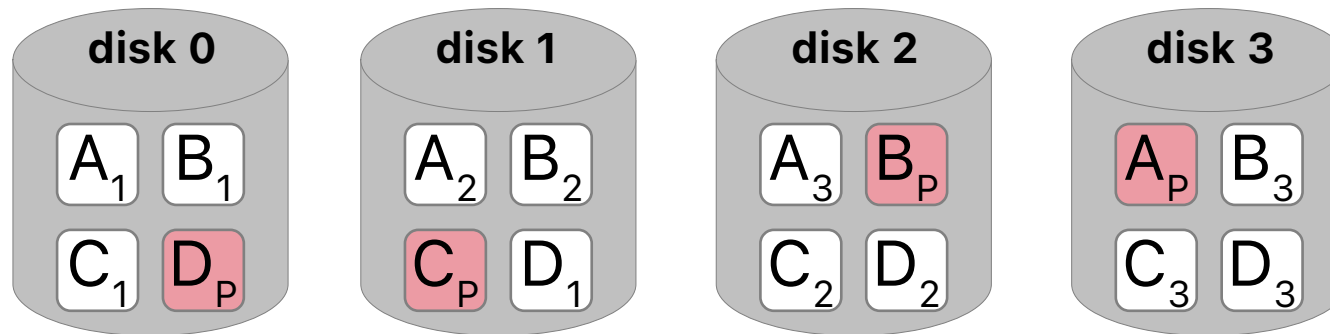
- **approach:** data is striped across multiple disks, one disk contains associated parity



- **effect:** errors of a disk can be detected and corrected without large additional storage space requirements, fast reading
- **disadvantage:** parity disk becomes a bottleneck when writing

RAID Level 5/6 – Striping with Distributed Parity

- **approach:** parity block is distributed over all disks



RAID level 5 is
the most
common RAID
variant today

- **effect:** additional load during writing by updating the parity block is distributed among disks
 - RAID 6: the failure of up to two disks can be handled by an additional parity block
- **disadvantage:** all data is protected elaborately, even if a part of it is not critical

Journalled File Systems

- **general idea:** in addition to writing the data and meta-data (e.g., inodes), a log (i.e., journal) of the changes is recorded
 - all changes occur as part of **transactions**
 - transaction examples:
 - create, delete, expand, shorten files
 - change file attributes
 - rename file
- recording/logging of all changes to the file system **additionally** in a **log file**
- during the boot process, the log file is compared with the current changes, thus avoiding inconsistencies

Journalized File Systems – Protocol

- for each individual operation of a transaction:
 1. a **log entry** is first generated
 2. the **changes to the file system** are made
- thereby applies:
 - the log entry is always written to disk **before** the actual change
 - if something was changed on disk, the log entry for it is also exists on the disk

Journalled File Systems – Recovery

- with a recovery during the **boot** process, the operating system checks whether the logged changes are present:
 - transaction can be repeated or completed if all log entries are available → **redo**
 - started but not finished transactions are undone → **undo**

Journalled File Systems – Discussion

■ benefits

- a transaction is either fully executed or not at all
- users can also define transactions over multiple file accesses if they are also recorded in the log
- inconsistent meta data is impossible
- fast recovery - booting a crashed system requires only the relatively short pass through the log file
 - alternative **chkdsk** requires much time for large disks

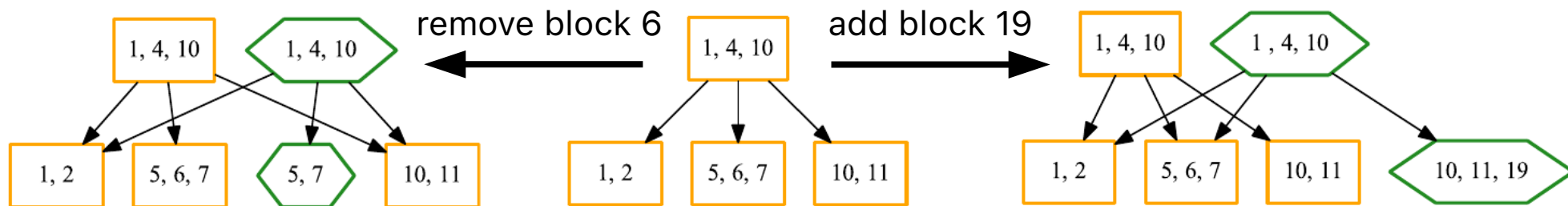
■ disadvantage

- less efficient, as the additional log file needs to be recorded
 - therefore often only "**meta-data journaling**", no "**full journaling**"

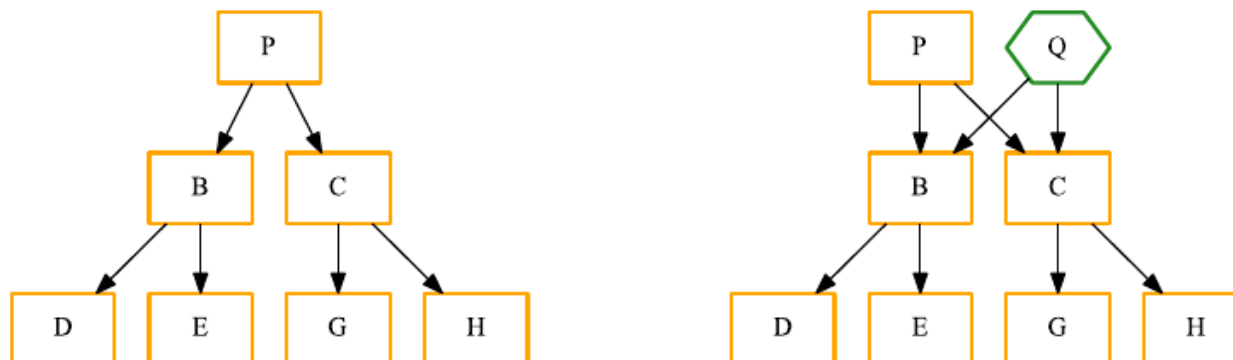
■ examples: NTFS, Ext3/4, ReiserFS

Copy-on-Write File Systems

- many modern file systems refrain from overwriting blocks
 - idea: create and store copies to suitable (free) locations on disk
- example:** modify a file system with B+ tree structure



- example:** copy directory tree recursively



only when P or Q are changed a copy made - the basis for the efficient creation of snapshots

Copy-on-Write File Systems – BTRFS

- widely used in the context of Linux, inspired by ZFS
- **many modern file system features**
 - fast writes: special "CoW-friendly" B+ trees
 - resource efficient **snapshots**
 - **no data loss**
 - atomic changes and checksums for all metadata and data
 - transaction-based, without explicit log file
 - **utilization of multiple drives**
 - implements flexible RAID
 - differentiation of data and metadata (different RAID levels)
 - **online resizing** of file system
 - **supports data compression**

Agenda

- ▶ Recap
- ▶ Organizational Matters
- ▶ Motivation
- ▶ File Allocation Methods
 - ▶ Contiguous Allocation
 - ▶ Linked Allocation
 - ▶ Indexed Allocation
- ▶ File Systems
 - ▶ UNIX File System
 - ▶ Linux ext4
- ▶ Advanced Data Organization
 - ▶ Error Handling, Recovery, Performance
 - ▶ RAID, Journaling, Copy on Write
- ▶ Summary and Outlook



▶▶ Summary and Outlook

■ summary

- file systems are an operating system abstraction to organise persistent data on storage devices
- mapping of logical view (files, directories) to physical view (blocks)
- consider hardware properties for efficient data management
 - head positioning (i.e., for hard drives)
 - wear leveling (i.e., for persistent flash memory)
- reliability by redundancy, better performance with copy-on-write

■ outlook: input/output

- abstractions for accessing input/output devices
- device classes and programming modes

References and Acknowledgments

Lecture

- ▶ Systemnahe Programmierung in C (SPiC), Betriebssysteme (Jürgen Kleinöder, Wolfgang Schröder-Preikschat)
- ▶ Betriebssysteme und Rechnernetze (Olaf Spinczyk, Embedded Software Systems Group, Universität Osnabrück)

Teaching Books and Reference Book

- [1] Avi Silberschatz, Peter Baer Galvin, Greg Gagne: *Operating System Concepts*, John Wiley & Sons, 2018.
- [2] Andrew Tanenbaum, Herbert Bos: *Modern Operating Systems*, Pearson, 2015.
- [3] Wolfgang Schröder-Preikschat: *Grundlage von Betriebssystemen – Sachwortverzeichnis*, 2023.
<https://www4.cs.fau.de/~wosch/glossar.pdf>