NTNU | Norwegian University of Science and Technology

Operating Systems

Lecture overview and Q&A Session 8 – 14.3.2022

Michael Engel

Lectures 15 and 16

File systems

- File abstraction – everything is a file
- Unix file access API
- Virtual file systems and mounting
- Simple file storage: contiguous, FAT, indexed, trees
- Free space and directory management

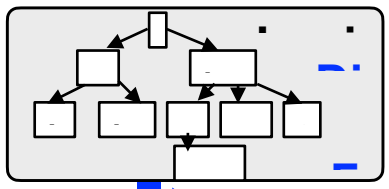
Modern file systems

- Challenges: reliability and performance
- Unix buffer cache
- Logical volume management and RAID
- Journaled, log-structured, CoW file systems

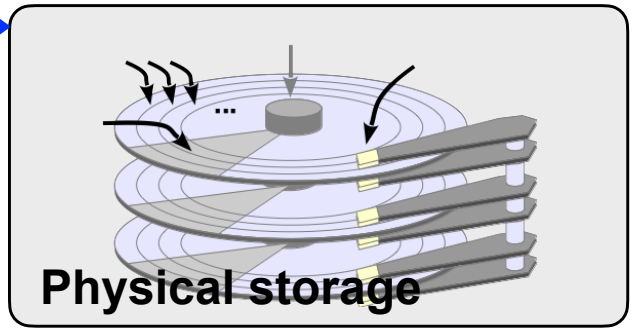
File abstraction – everything is a file

Files

- Direct block access to disks does not work as abstraction
  - OS provides a logical view for applications
- Files abstract from storage location
  - Metadata: owner, access permissions, dates etc.
  - Enables network and virtual file systems
- Unix: **"everything is a file"**
  - every resource in the system can be accessed using a name mapped into a directory hierarchy
- This breaks down for network access
  - see Plan 9, Inferno followup projects



File system and directory hierarchy



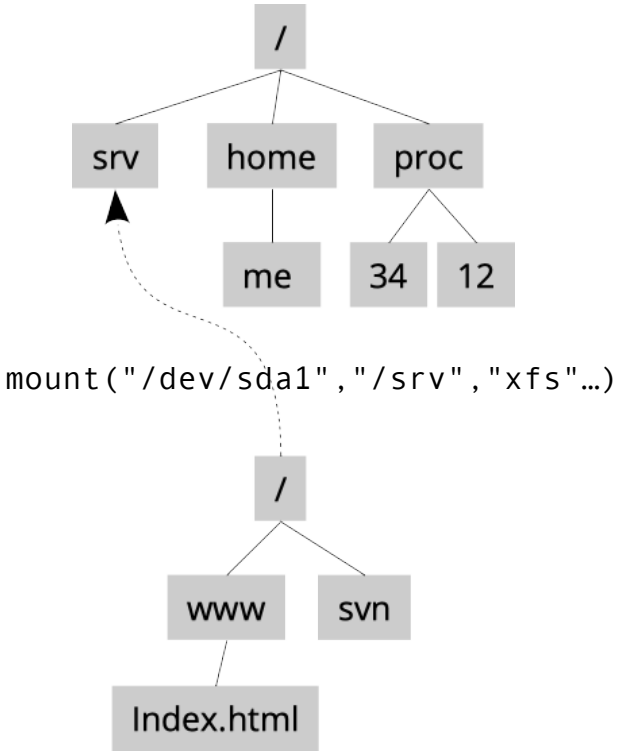
Unix file access API

- Files are identified by per process *file descriptors* in the OS
  - positive integer number, can be reassigned
- The Unix file access API consists of four simple system calls:
- `int open(const char *path, int oflag, ...);`
  - Attempts to open the file with the given path name and options for accessing (read only, read/write etc.)
  - Returns a ***file descriptor*** (fd) refering to the file on success
- `ssize_t read(int fd, void *buf, size_t nbyte);`
- `ssize_t write(int fd, const void *buf, size_t nbyte);`
  - Read (write) `nbyte` bytes from (to) file `fd` into (from) the memory starting at user space memory address `buf`
- `int close(int fildes);`
  - Closes the file: flushes buffers and invalidates file descriptor
- *…plus a large number of supporting functions (access, stat, …)*

Virtual file systems and mounting

System-wide name space for files

- One root of the file system
- Additional file systems are ***mounted*** into the file system hierarchy
  - Overlay an existing directory

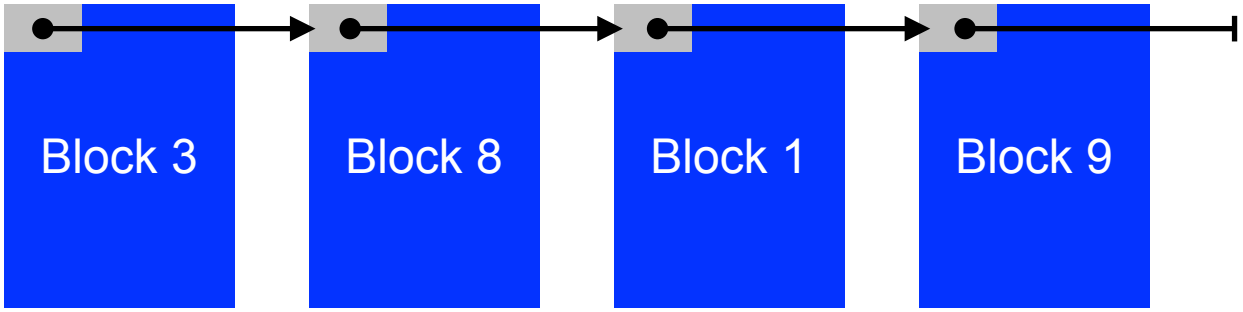


- Enabled by the **virtual file system**
  - All file systems (parts of the OS kernel in a monolithic OS) have to implement the same API
  - Virtual file systems are not backed by storage
    - e.g. `/proc`, `/sys` on Linux

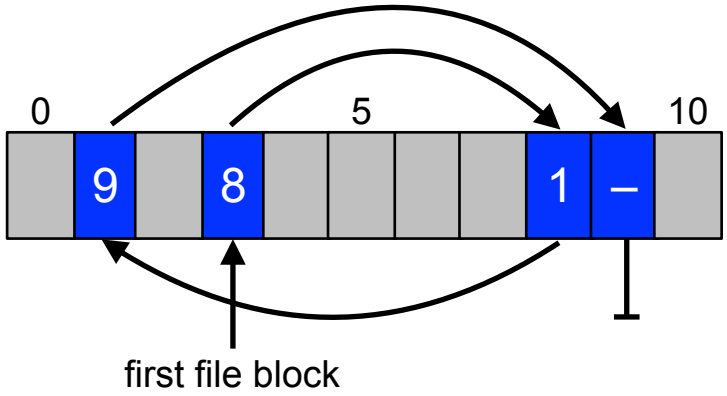
Simple file storage: contiguous, FAT

Contiguous storage: files allocate increasing block numbers

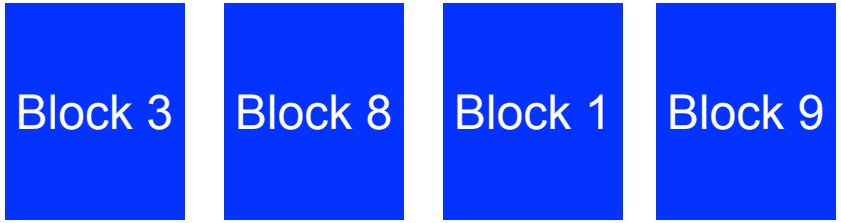
Linked lists: file data blocks contain pointer to following block



Linked lists example: MS-DOS FAT file system

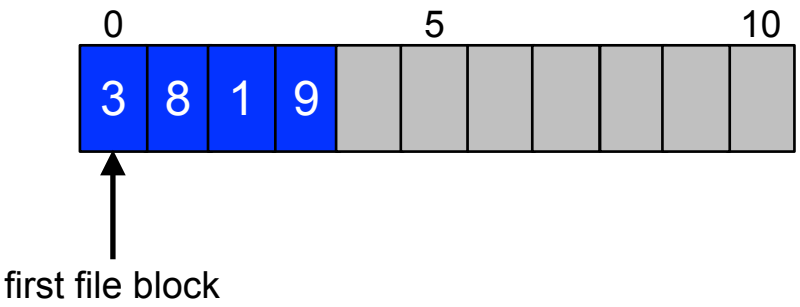


Blocks of the file: 3, 8, 1, 9

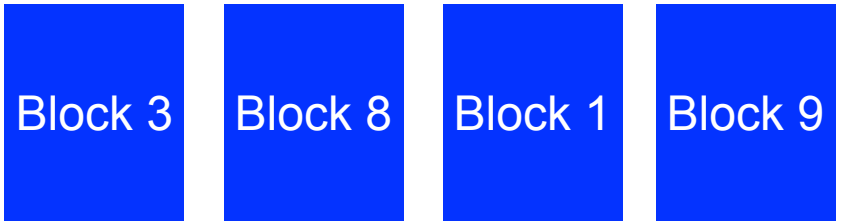


Simple file storage: indexed

Indexed storage: special disk block with numbers of data blocks of a file



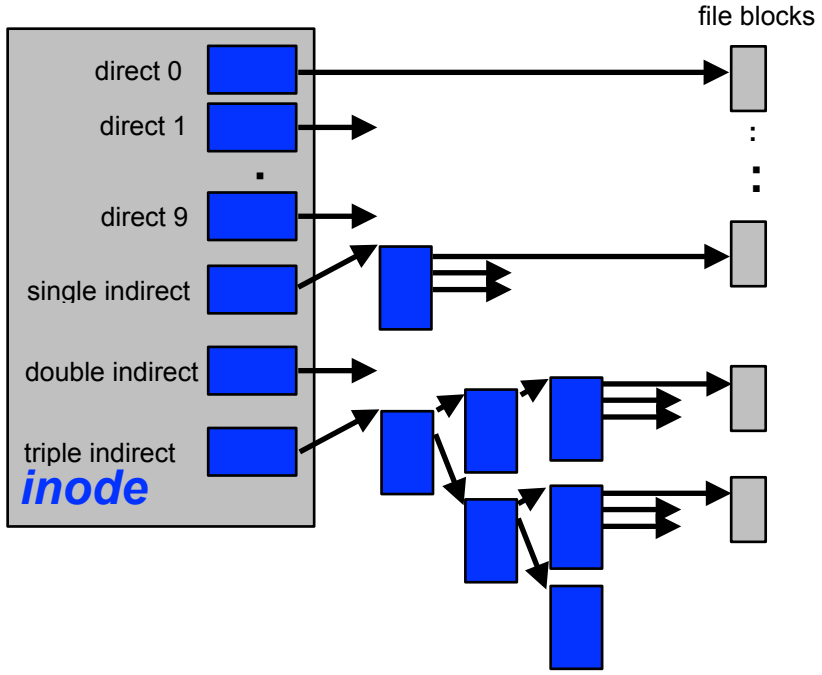
Blocks of the file: 3, 8, 1, 9



Example: classical Unix file system

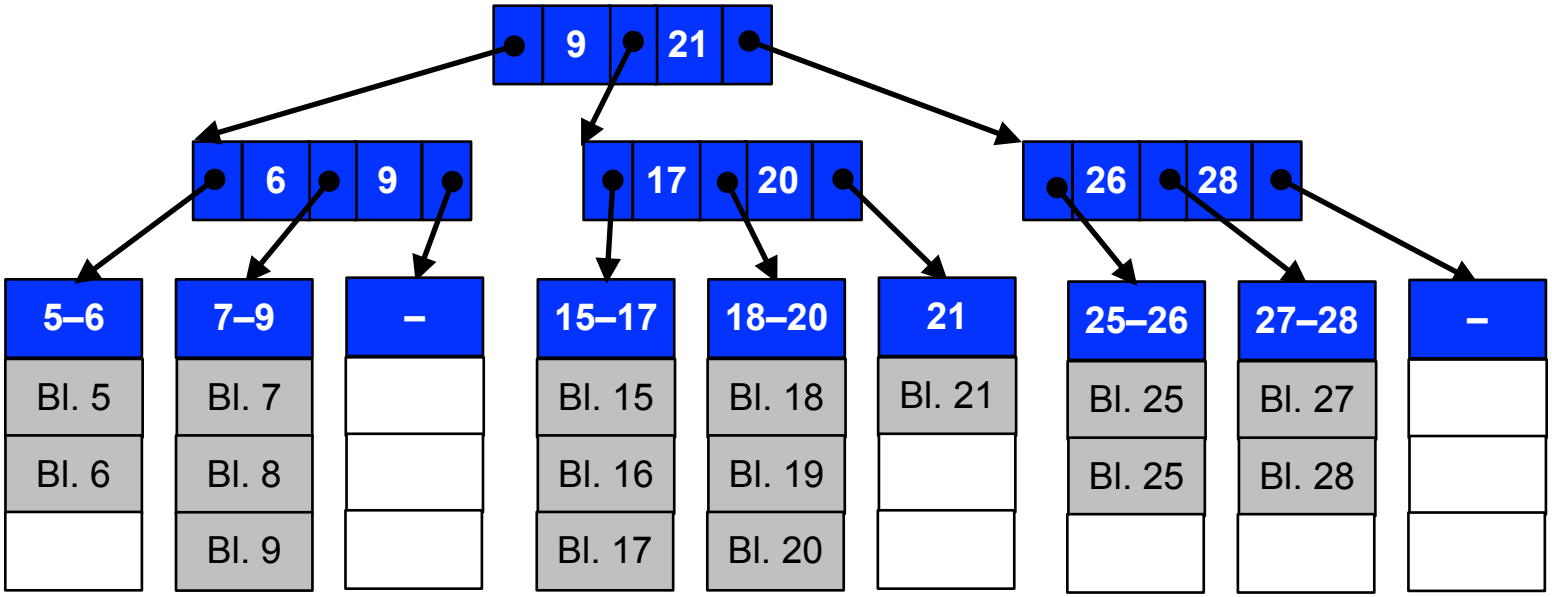
Uses index nodes ("***inodes***")

Multiple levels of indirection to enable efficient indexing of small files while also providing for potentially large files



Simple file storage: trees

Tree-based storage: idea from databases to efficiently retrieve records can also be used to find ***chunks*** of a file in a file system



Example: Apple HFS (available since 1986 for the "classic" MacOS) is based on a B* tree

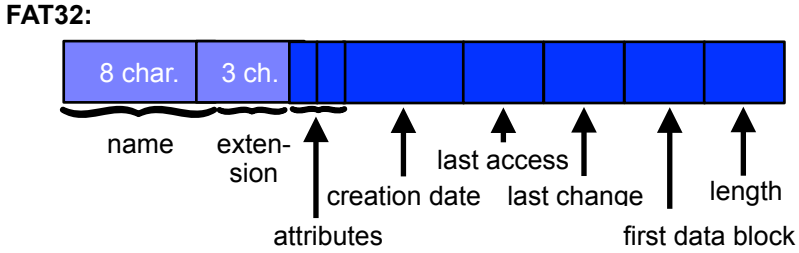
Free space and directory management

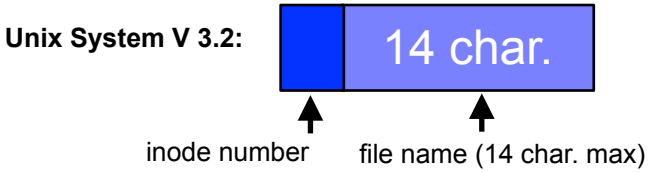
Approaches for free space management

- bit maps, linked lists and trees of free blocks

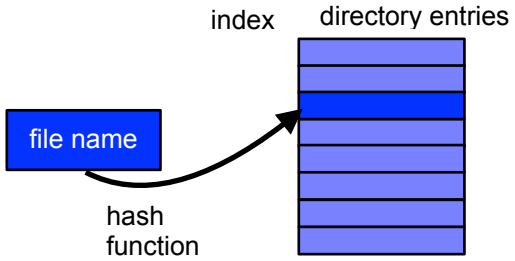
Directory management

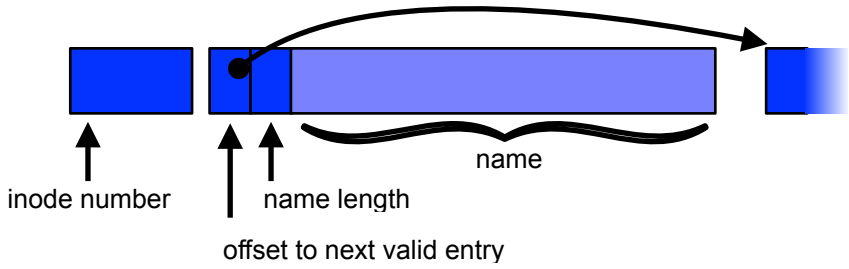
- lists: FAT, traditional Unix





- hash functions and lists with variable length (BSD FFS)

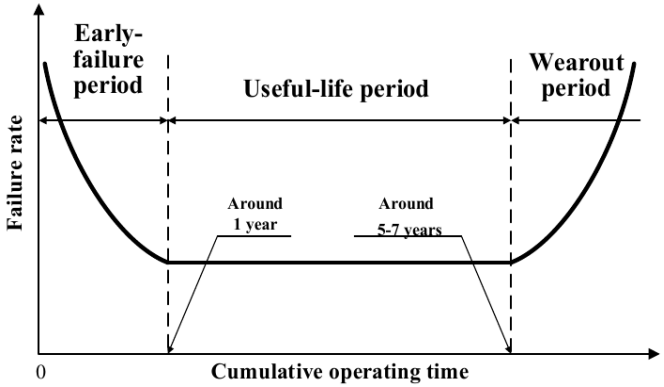




FS challenges: reliability and performance

Reliability

- Disks can degrade or fail, systems can crash
- Prevent data loss, ensure integrity
- Methods: backups, checksums, repair tools



"bathtub curve"

Performance

- Low read/write speeds and high positioning latency in traditional hard disks
- Solutions: caches

Disk management

- Extend file systems beyond the size of a single disk

Unix buffer cache

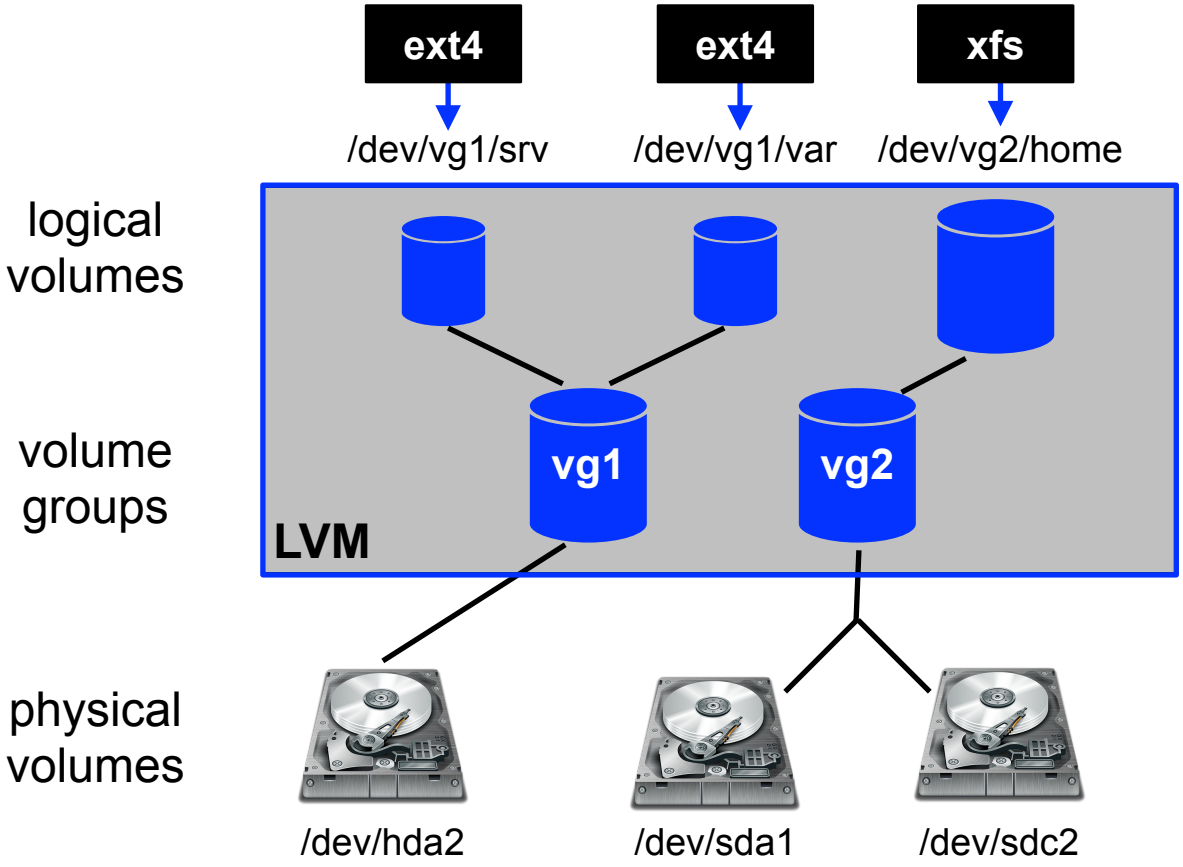
Buffer for disk blocks in main memory

- Algorithms similar to page frame handling for virtual mem.
- Optimizations:
  - **Read ahead**: for sequential reads initiated transfer of subsequent data blocks before they are requested
  - **Lazy write**: a block is not written to disk directly
    - allows optimization of write accesses, does not block writer
- Write back from cache
  - if no more free buffers are available
  - periodically to minimize data loss in case of system crash
  - manually using `sync(2)`
  - option when opening files (`O_SYNC`)

Logical volume management

Remove 1:1 relation between file system and disk

- File systems can span multiple disks



RAID

Idea: save costs by creating large logical disks out of inexpensive smaller disks

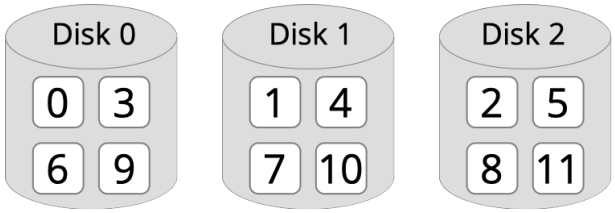
- Additional features:
  - better utilization of the available **data bandwidth** by using parallel transfers
  - fault tolerance using **redundancy**

Variants

- Hardware RAID: disk controller with special management software (+potentially cache)
- Software RAID: layer between disk driver and file system code
- Different RAID levels – differ in throughput and reliability

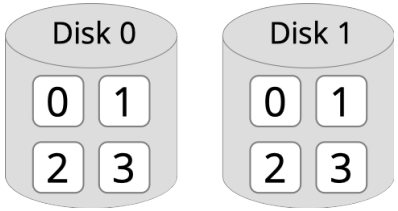
RAID levels

RAID0: striping – increase bandwidth, but increase failure probability by xN

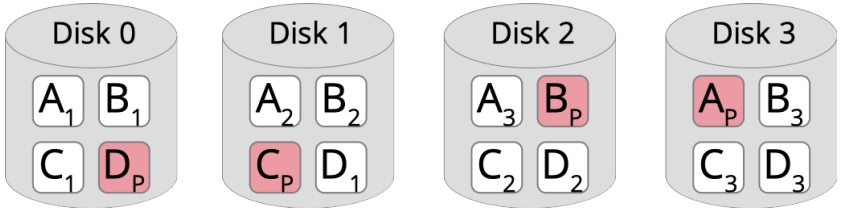



```
pdsk = lblk % N;
pblk = lblk / N;
```

RAID1: mirroring – increased read bandwidth, somewhat lower write bandwidth, higher reliability by having a copy of the data



RAID5+6: distributed parity – additional write overhead (most used RAID)



Journaled & log-structured file systems

Journaled file systems

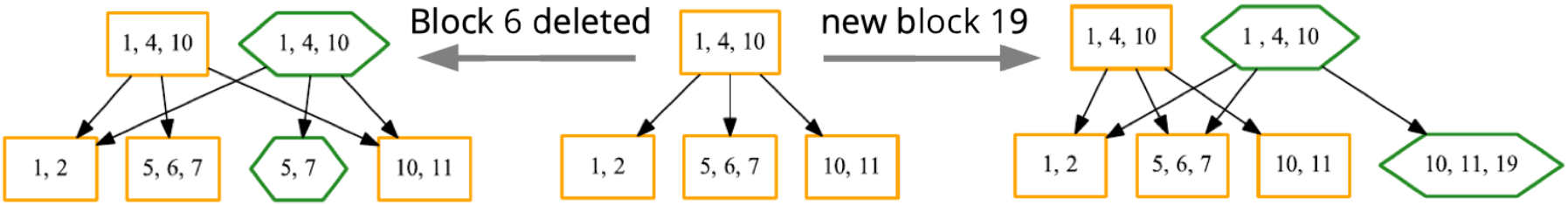
- write a ***protocol of the changes*** in addition to writing data and metadata
- all changes (e.g., create, delete) are part of a ***transaction***
- All changes are also stored in a ***protocol file*** (log file)

Log-structured file systems

- (Radical) approach: one log is sufficient for everything!
- Blocks are not overwritten, but only appended to the log
- Changes to metadata are also stored in the log only
- Write operations are collected in main memory and then written to disk as a single large segment

CoW file systems

- Many modern file systems (e.g. BTRFS) refrain from overwriting
  - Idea from LFS, but more flexible when allocation free areas
- Example: manipulate file (B+ tree)… [2]



- Example: "copy" complete directory tree

Only when P or Q are changed, a copy is performed –
basis for the efficient creation of ***snapshots***

Overview Theoretical Exercise 7

All about file systems…

Why?

- File systems are a central part of most server and desktop operating systems
  - Not necessarily in embedded systems
  - Also – object stores (Smalltalk, Lisp)
- File systems are crucial for *non-functional properties* of an OS
  - Performance (latencies, throughput)
  - Reliability