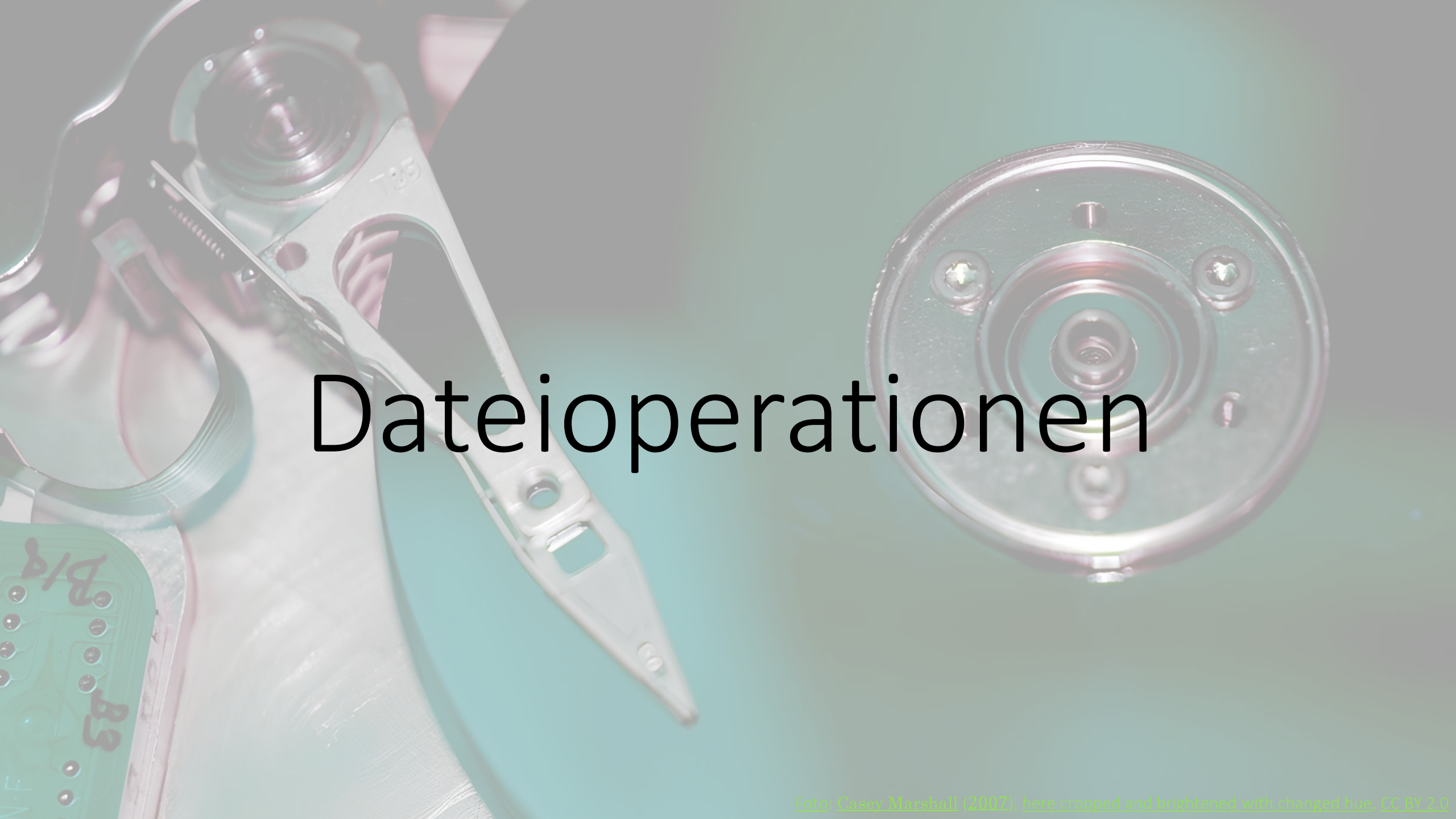


# Dateioperationen

# Dateien und Dateisysteme

- Daten auf einer Festplatte werden meistens zu abgeschlossenen Datenmengen, zu sogenannten **Dateien** abgefasst
- Dateien können je nach Anwendung in unterschiedlichsten **Dateiformaten** vorliegen und besitzen neben einen **Dateinamen** weitere **Meta-Informationen** wie **Dateigröße**, **Besitzer**, etc…
- Das **Dateisystem** legt fest, wie und wo Dateien auf der Festplatte gehalten, geschrieben und gelöscht werden

# Ordner

- Viele Dateisysteme erlauben das organisieren von Dateien in **Verzeichnisstrukturen** (**Ordnerstrukturen**): **Verzeichnisse** (**Ordner**) können jeweils wiederum Dateien und weitere Verzeichnisse enthalte → Baumartige Struktur (**Verzeichnisbaum**)
- Jede Datei besitzt einen eindeutigen **Dateipfad**

# Dateiartige und verzeichnisartige Objekte

- UNIX-Systeme bilden unterschiedliche Betriebsmittel auf das Dateisystem (als '**nichtreguläre** Dateien') ab:
  - Symbolische Links
  - E/A-Geräte und Datenträger
  - Feststehende Programmverbindungen (Pipes)
  - Netzwerkkommunikationsendpunkte (Sockets)
  - Netzwerkverzeichnisse
- Eine wie zuvor definierte Datei wird als **regulär** bezeichnet

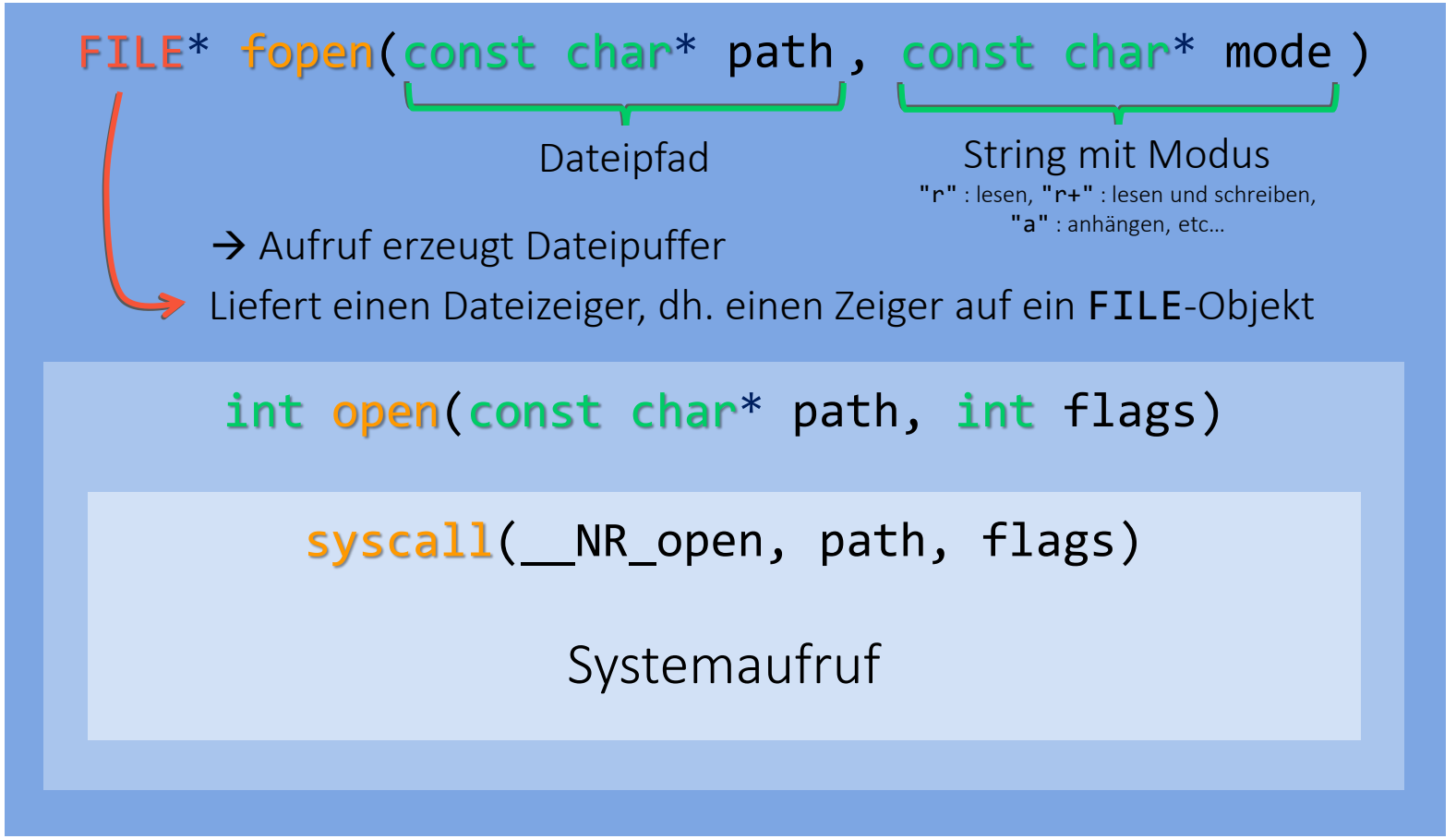
# Dateischnittstelle in C

- `stdio.h` liefert standardisierte Schnittstellen um unabhängig vom Dateisystem oder dem verwendeten Datenträger Dateioperationen sowie Verzeichnisoperationen durchzuführen
- Während es sich bei `open`, `read`, `lseek` und `close` im Grunde genommen eigentlich nur um eine Art 'Verpackung' um die entsprechenden Systemaufrufe handelt, sind `fopen`, `fread`, `flseek` und `fclose` Abstraktionen davon und bieten dank interner Pufferung durch blockweises lesen höhere Geschwindigkeiten → `FILE`-Objekt
- `fstat` und diverse Makros (`S_ISREG`, `S_ISDIR`, …) erlauben die Abfrage diverser Dateiinformationen aus dem Dateisystem

# Dateiinformationen abfragen und auswerten

- `int fstat(const char* path, struct stat* finfo)`
  - Untersucht die Datei bzw. das dateiartige Objekt auf welches der Pfad `path` verweist und liefert bei Erfolg `0`
  - Die Resultate dieser Untersuchung werden in `finfo` gespeichert
- `S_ISREG(finfo)` liefert für eine reguläre Datei `true`
- `S_ISDIR(finfo)` liefert für ein Verzeichnis `true`

# fopen im Detail



# Das `FILE`-Objekt

- Das `FILE`-Objekt enthält neben dem **Dateideskriptor** zum identifizieren einer geöffneten Datei auch den **Modus** sowie Zeiger auf den Puffer inklusive den **Positionszeiger**
- **Achtung**: Auf die Felder des `FILE`-Objekts darf nicht direkt zugegriffen werden, da dies laut C-Standard zu undefinierten Verhalten führt!
  Je nach System unterscheiden sich die Implementierungen von `FILE`
- `DIR`-Objekt als Analogon für Verzeichnisse

# Das `FILE`-Objekt (Mögliche Implementierung)

```c
typedef struct _IO_FILE {
    int _flags; // Modus (z. B. _IO_READ, _IO_WRITE)
    char* _IO_read_ptr; // Aktuelle Lese-Position
    char* _IO_buf_base; // Start des Puffers
    char* _IO_buf_end; // Ende des Puffers
    int _fileno; // Dateideskriptor (z. B. 0 für stdin)
    // ... weitere plattformspezifische Felder
} FILE;
```

# Aus Datei lesen

- `size_t fread(void* buf, size_t size, long int` $n$`, FILE* stream)`
  - Ließt aus der mit `stream` referenzierten, geöffneten Datei $n$ Objekte der Größe `size` in den Puffer `buf`
  - Liefert die Gesamtgröße der erfolgreich gelesenen Objekte zurück
- `int fscanf(FILE* stream, const char* form, …)`
  - Ließt gemäß der Formatierung in `form` Daten aus der Datei
  - Funktioniert ansonsten wie `sscanf`

# In eine Datei schreiben

- `int fputc(char c, FILE* stream)`
  - Schreibt `c` an die aktuelle Position des Positionszeigers in die Datei
- `int fputs(const char* str, FILE* stream)`
  - Schreibt den String `str` beginndend ab der aktuellen Position des Positionszeigers in die geöffnete Datei
  - Funktioniert ansonsten wie `puts`
- `int fprintf(FILE* stream, const char* form,…)`
  - Schreibt den durch `form` formatierten String die geöffnete Datei
  - Funktioniert ansonsten wie `sprintf`

# Positionszeiger auslesen

- `long int ftell(FILE* stream)`
  - Liefert die aktuelle Position des Positionszeigers
  - Kann genutzt werden, um auf naive Art die Dateigröße zu ermitteln

# Leseposition versetzen

- `size_t rewind(FILE* stream)`
  - Setzt den Positionszeiger an den Anfang der Datei
- `int fseek(FILE* stream, long int v, int o)`
  - Versetzt den Positionszeiger um den Versatz `v` relativ zur Referenzposition
  - `o` gibt die Referenzposition an:
    - Dateianfang: `SEEK_SET`,
    - Aktuelle Position: `SEEK_CUR`
    - Dateiende: `SEEK_END`
  - Liefert bei Erfolg `0`

# Schließen einer Datei

- `int fclose(FILE* stream, long int v, int o)`
  - Schließt die Datei: Betriebssystem gibt die Datei frei
  - Speicherplatz für die Puffer wird freigegeben
  - Speicherplatz des `FILE`-Objektes wird freigegeben
  - Liefert bei Erfolg `0`

# Überblick über die Verzeichnisoperationen

- `DIR* opendir(const char* path)`
  - Öffnet das Verzeichnis, auf das der Pfad `path` verweist
  - Liefert für das geöffnete Verzeichnis einen Verzeichniszeiger
- `struct dirent* readdir(DIR* dir)`
  - Gibt einen Zeiger auf eine Datenstruktur zurück,
    die alle aktuellen Einträge des geöffneten Verzeichnisses enthält
- `int closedir(DIR* dir)`
  - Schließt das aktuell geöffnete Verzeichnis