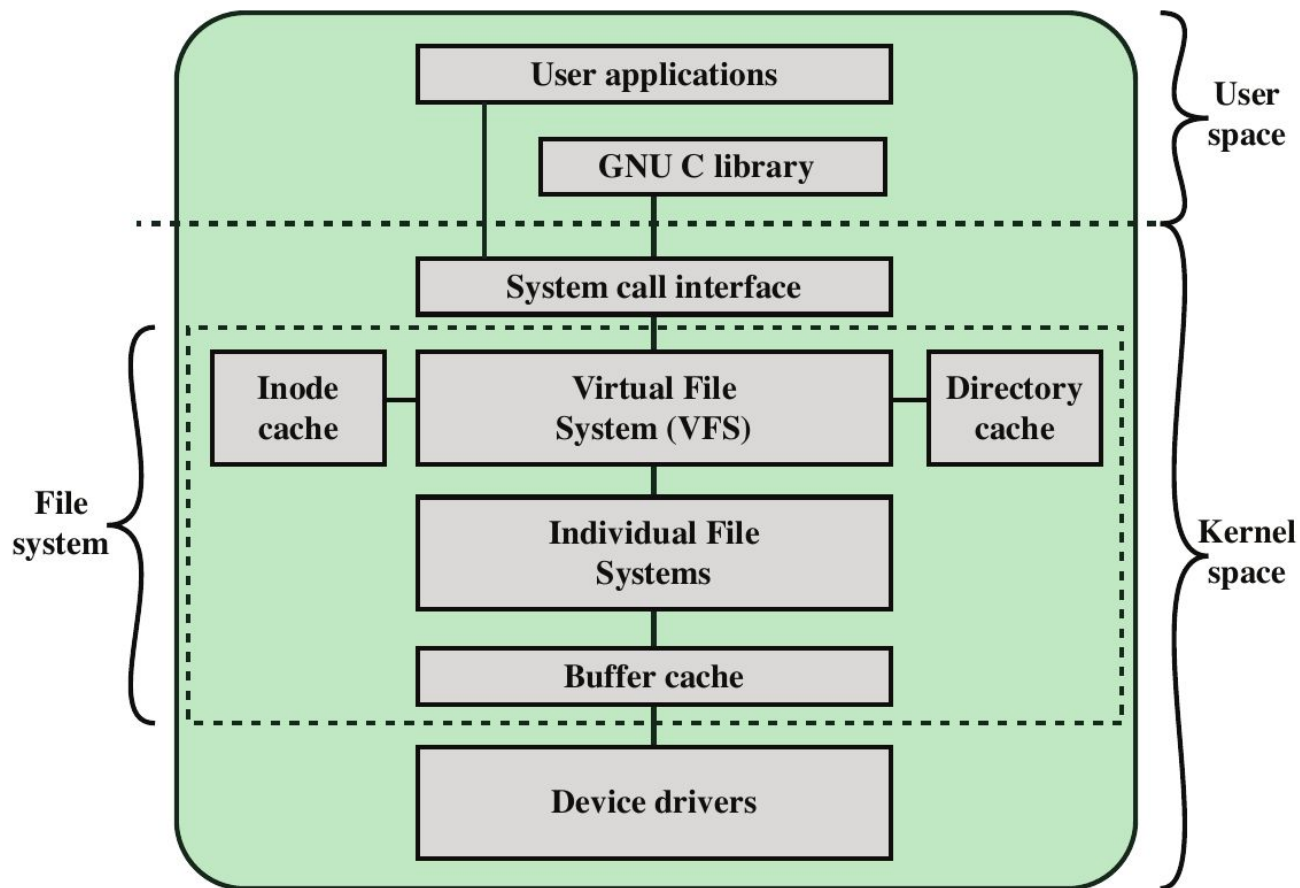


Systemy operacyjne

Wykład 11: Systemy plików



Dziś będziemy mówić o tym jak implementacja wywołań systemowych korzysta z VFS i indywidualnych systemów plików, oraz o organizacji danych na dysku.

The Linux Storage Stack Diagram

Po co nam systemy plików?

1. Przechowywanie danych:

- a. ogólnego przeznaczenia → [NTFS](#), [ext4](#)
- b. dyski półprzewodnikowe → [JFFS2](#), [APFS](#)
- c. dyski optyczne RO → [UDF](#), [ISO 9660](#)
- d. serwerowe → [btrfs](#), [ZFS](#), [XFS](#)
- e. rozproszone → [NFS](#), [lustrefs](#)
- f. pamięć ulotna → [tmpfs](#) (ram dysk)
- g. pamięć ROM → [initramfs](#), [squashfs](#)

2. Reprezentacja zasobów jądra:

- a. procesy → [procfs](#)
- b. stan i konfiguracja systemu → [sysfs](#)
- c. pliki urządzeń → [devfs](#)

3. Manipulacja zasobami:

- a. składanie systemów plików → [unionfs](#)
- b. podłączanie systemów plików użytkownika → [FUSE](#)

Czym zarządzają systemy plików?

Pamięcią, z reguły zewnętrzną! Jednostką adresacji jest **sektor** (z reguły 512 bajtów). Dawniej adresowanie fizyczne używające **geometrii dysku** ([CHS](#)). Dziś **liniowe adresowanie** ([LBA](#)) → adresy logiczne tłumaczone na fizyczne przez **kontroler dysku**.

System uniksowy dostarcza abstrakcji nad pamięcią zewnętrzną w postaci **urządzeń blokowych**, tj. plików o swobodnym dostępie.

Sterowniki takich urządzeń używają bardzo skomplikowanych protokołów komunikacji [ATAPI](#), [SCSI](#), [NVMe](#), [USB](#), itd.

Urządzenia umożliwiają przeprowadzanie diagnostyki: [smartctl\(8\)](#) → czy z dyskiem wszystko ok?

Urządzenia blokowe

```
# sudo lsblk
```

NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPOINT
sda	8:0	0	223,6G	0	disk	
├─sda1	8:1	0	512M	0	part	
├─sda2	8:2	0	207,4G	0	part	/mnt/old
└─sda3	8:3	0	15,7G	0	part	
sr0	11:0	1	1024M	0	rom	
nvme0n1	259:0	0	476,9G	0	disk	
├─nvme0n1p1	259:1	0	512M	0	part	/boot/efi
├─nvme0n1p2	259:2	0	488M	0	part	/boot
├─nvme0n1p3	259:3	0	476G	0	part	
└─nvme0n1p3_crypt	254:0	0	475,9G	0	crypt	
├─rocklobster--vg-root	254:1	0	475G	0	lvm	/
└─rocklobster--vg-swap_1	254:2	0	980M	0	lvm	[SWAP]

loop → urządzenie blokowe, którego zawartość jest odwzorowana w plik zwykły, **part** → partycja, **sr** → pamięć stała

Mapa przestrzeni dyskowej

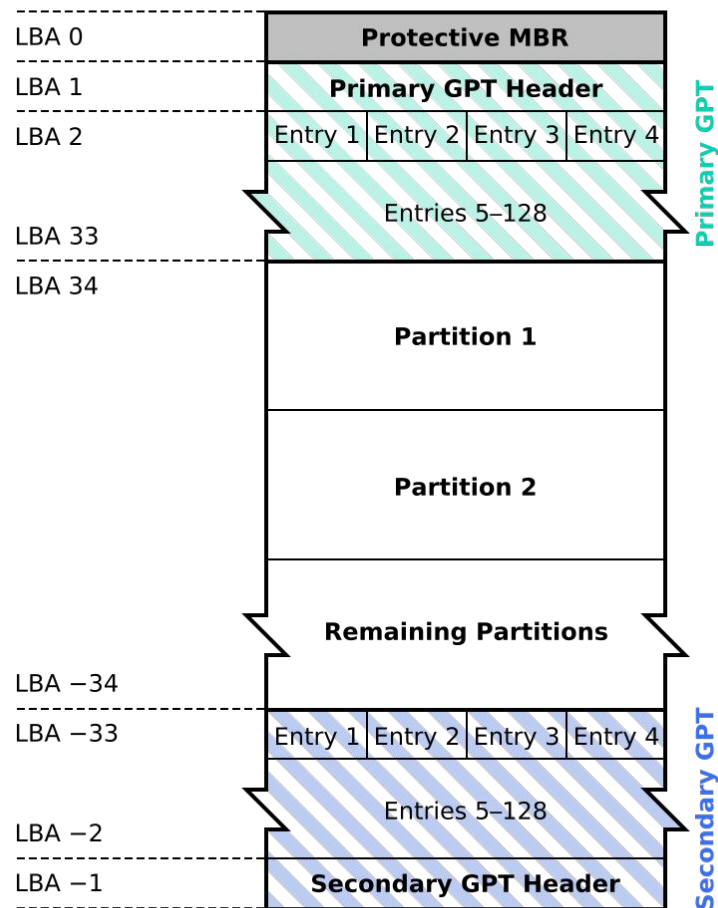
Gdzie umieścić **program rozruchowy** (ang. *boot loader*),
systemy plików (root, home, ...),
przestrzeń wymiany?

Dysk jest podzielony na **woluminy**
zwane **partycjami**.

Tablica partycji (np. [GPT](#))
to dyskowa struktura danych
przechowująca opis partycji.

Na dysku dwie kopie tablicy partycji,
dla każdego wpisu sumy kontrolne.

GUID Partition Table Scheme



Partycje na moim służbowym dysku

```
# sudo fdisk -l -o Device,Start,End,Size,Type /dev/sda
```

```
Disk /dev/sda: 223.6 GiB, 240057409536 bytes, 468862128 sectors
```

```
Disk model: PNY CS1311 240GB
```

```
Units: sectors of 1 * 512 = 512 bytes
```

```
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disklabel type: gpt
```

```
Disk identifier: A66B1B27-99E1-408B-9840-573AA63E2939
```

Device	Start	End	Size	Type
/dev/sda1	2048	1050623	512M	EFI System
/dev/sda2	1050624	435965951	207.4G	Linux filesystem
/dev/sda3	435965952	468860927	15.7G	Linux swap

VFS: motywacja

Systemy plików mogą bardzo różnić się od siebie implementacją, np. dyskowymi strukturami danych, formatem metadanych. Każdy z systemów plików chcemy wkomponować w drzewiastą strukturę. Implementacja wywołań `open(2)`, `read(2)`, `write(2)`, ... dla każdego z systemów plików? Czy da się uwspólnić ten kod?

VFS to infrastruktura jądra do implementacji systemów plików. Na przykład w systemie [NetBSD](#) określono interfejsy:

- udostępniane przez sterownik systemu plików, a wymagane przez wywołania systemowe → [vnodeops\(9\)](#), [vfsops\(9\)](#)
- wykorzystywane przez systemy plików:
 - buforowanie bloków dyskowych → [buffercache\(9\)](#)
 - kolejkovanie operacji dyskowych → [bufq\(9\)](#)
 - pamięć podręczna rozwiązywania ścieżek → [namecache\(9\)](#)

VFS: struktury danych jądra

Struktury danych każdego systemu plików są tłumaczone do:

[vnode](#) uniwersalna reprezentacja i-węzła w pamięci jądra

- typ pliku (VDIR, VREG, VLNK, ...)
- wczytany element struktury dyskowej (np. i-node)
- wskaźnik na tablicę operacji (a'la metody wirtualne) [vnodeops](#)

[vattr](#) uniwersalna reprezentacja metadanych pliku

- używane głównie przez `stat(2)` i `access(2)`

[mount](#) reprezentuje zamontowany system plików

- superblok, grupy bloków i inne prywatne dane systemu plików
- wskaźnik na tablicę operacji [vfsops](#)
- lista v-węzłów należących do instancji tego systemu plików

Fragmentacja systemu plików (1)

Systemy plików nie muszą przechowywać plików w ciągłym obszarze sektorów.

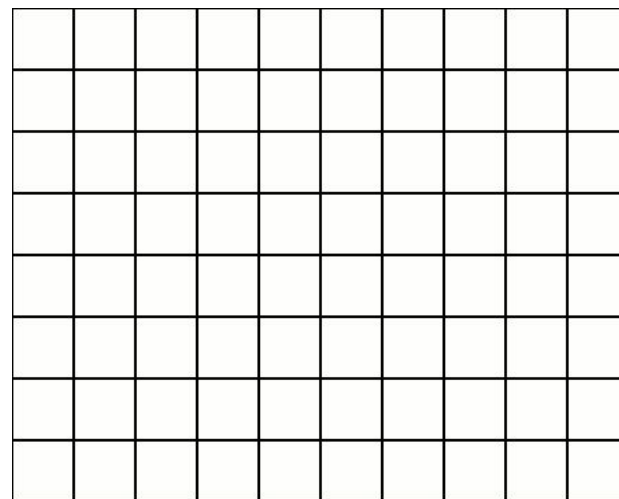
Fragmentacja zewnętrzna to nie problem!

Plik jest pofragmentowany, jeśli jest przechowywany na dysku w więcej niż jednym spójnym fragmencie.

Zmiana pozycji głowicy dysku magnetycznego jest kosztowna.

Losowe dostępy do dysku SSD są również wolniejsze niż sekwencyjne.

Przydział ciągły jest korzystny zawsze!



Fragmentacja systemu plików (2)

Jak radzić sobie z **fragmentacją systemu plików**?

- pozostawienie miejsca za ostatnim blokiem zapobiega powstawaniu fragmentów przy dopisywaniu na koniec pliku
- **odroczone przydział bloków** (ang. *delayed allocation*) na k sekund, grupujemy w pamięci zapisy wymagające przydziału nowych bloków w nadziei na zmniejszenie ogólnej liczby fragmentów
- **usługa defragmentacji** narzędzie użytkownika lub wątek jądra

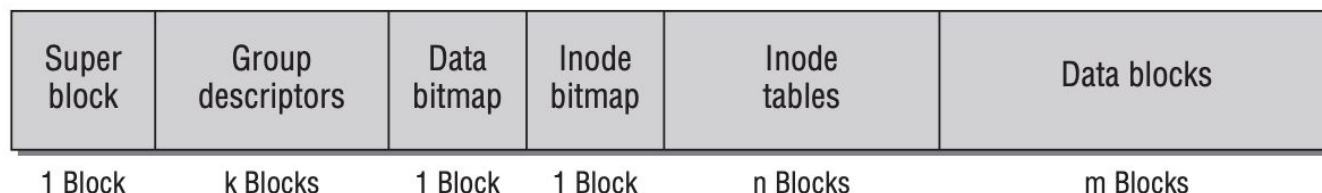
Jak radzić sobie z **fragmentacją wewnętrzną** plików?

- **upakowywanie ogonów** (ang. *tail packing*) końcówki kilku plików pakujemy do jednego bloku, dobre jeśli pliki już nie rosną
- umieszczanie zawartości bardzo krótkich plików w miejscu wskaźników i-węzła (np. dowiązania symboliczne)

Organizacja prostego systemu plików ([ext2](#))



Boot → miejsce na program rozruchowy. Reszta miejsca podzielona na **grupy bloków (BG)** równego rozmiaru (prócz ostatniego).



Superblok (SB) przechowuje najważniejsze właściwości systemu plików (blksize, #blk, #inode). **Deskryptory grup (GDs)** przechowują położenie i rozmiar **bitmapy zajętości** bloków i i-węzłów każdej grupy bloków.

SB i GDs są przechowywane w BG_0 , a ich kopie w wyróżnionych BG_i . System dąży do przydziału pliku w obrębie jednej BG.

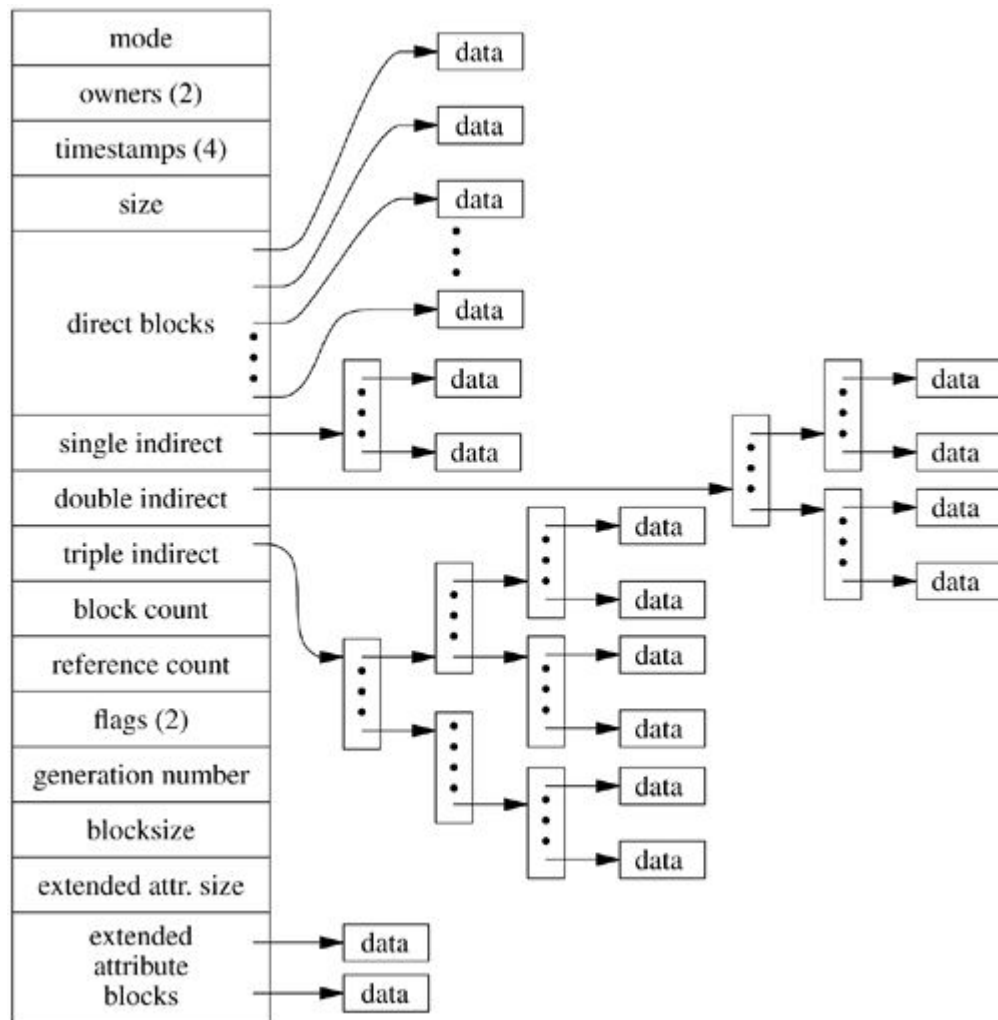
Z góry przydzielona liczba i-węzłów → możliwych plików!

i-węzeł (ang. *i-node*)

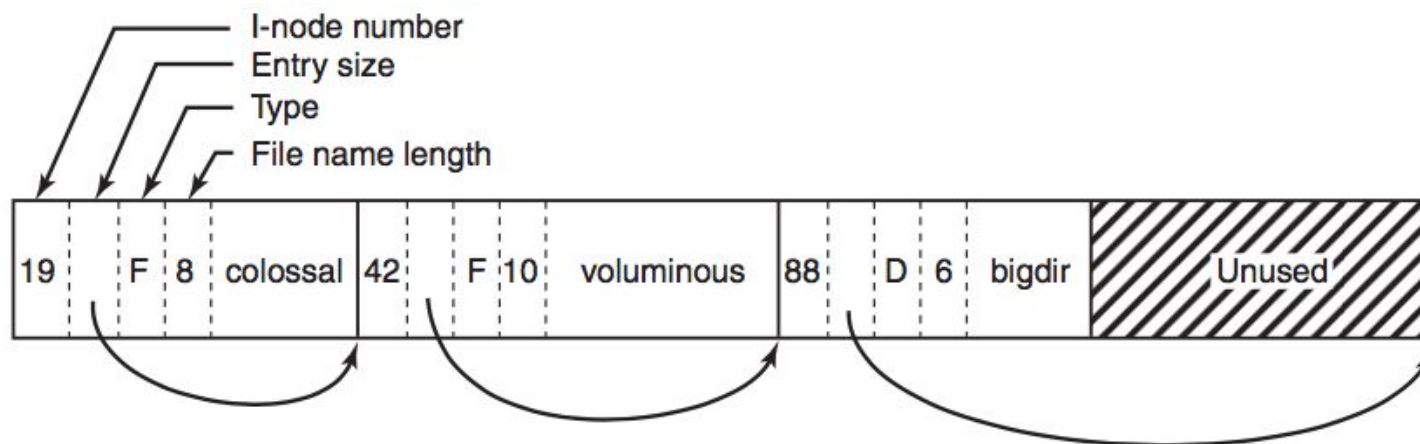
Opis zasobu dyskowego.
Oprócz atrybutów zawiera
wskaźniki na bloki danych
i bloki pośrednie
(ang. *indirect blocks*).

Przydział bloków
w strukturze drzewiastej,
która rośnie wraz
z rozmiarem pliku.

i-węzeł ma 128B lub 256B



Reprezentacja katalogów



Reprezentacja listowa → liniowe wyszukiwanie, koszt niwelowany przez [namecache\(9\)](#). Co jakiś czas potrzebne kompaktowanie! Dodawanie lub usuwanie plików potencjalnie wymaga przejrzania całego katalogu.

Lepsze rozwiązania? [HTree](#) lub [B+Tree](#).

[LKML] Add ext3 indexed directory (htree) support

Creating 100,000 files in a single directory took 38 minutes without directory indexing...
and 11 seconds with the directory indexing turned on.

e2fsprogs: narzędzia systemu plików ext[234]

```
# truncate -s 100M disk.img  
# /sbin/mkfs.ext4 -L testfs disk.img  
# /sbin/dumpe2fs disk.img  
# ls -lhs disk.img
```

fsck naprawianie uszkodzonego systemu plików

resize2fs zmiana rozmiaru systemu plików po zwiększeniu
woluminu lub przed zmniejszeniem wolumenu

tune2fs zmiana parametrów systemu plików

debugfs przeglądanie struktur i właściwości systemu plików

NetBSD: sterownik [ext2fs](#)

Na potrzeby projektu programistycznego “*System plików*” pożyczymy definicje dyskowych struktury danych ext2 z jądra NetBSD (licencja BSD na to pozwala):

- superblok: [ext2fs](#)
- grupa bloków: [ext2_gd](#)
- i-węzeł: [ext2fs_dinode](#)
- wpis katalogu: [ext2fs_direct](#)

Będziecie korzystać z okrojonej wersji tych struktur danych.
Definicje będą w pliku [ext2fs_defs.h](#)

Spójność systemu plików

Operacje na systemie plików nie są atomowe, np. zakładanie pliku:

- znajdź wolny i-węzeł i oznacz go jako zajęty (w bitmapie)
- zainicjalizuj i-węzeł
- wprowadź nowy wpis do katalogu (potencjalnie trzeba go powiększyć)

Jeśli w trakcie zapisu na dysk przydarzy się awaria systemu lub przerwa w dostawie energii, to system plików może zostać w **niespójnym** stanie. Po awarii musimy być w stanie odtworzyć jakiś spójny stan. Przydałoby się również zminimalizować ilość utraconych danych.

Chcemy zachować **spójność metadanych**, czyli danych nadających strukturę systemowi plików (katalogi, i-węzły, mapa wolnych bloków, itd.). Większość programów zakłada, że po wykonaniu operacji **write** dane są bezpieczne na dysku, ale to wymagałoby zachowania **spójności danych**.

Metody zachowania spójności

zapisy synchroniczne wymuszamy czekanie na zakończenie operacji wejścia-wyjścia. Niestety mogą powstawać wycieki pamięci dyskowej! Szybkość operacji ograniczona przepustowością dysku.

wsparcie sprzętowe pamięć NVRAM kontrolera dysku przechowuje niedokończone operacje. Szybkość operacji ograniczona pojemnością.

księgowanie (ang. *journalling*) zmiany metadanych zapisujemy jak **transakcje** do pliku dziennika zanim zostaną wprowadzone do systemu plików. Po awarii czytamy dziennik i odtwarzamy operacje.

copy-on-write trwała struktura systemu plików; poprzednia wersje obiektu nie ulegają zmianom, wskaźniki na bloki zawierają skrót bloku (ang. *hash*), tworzymy nowe wersje obiektu, a stare wersje z czasem zwalniamy; superblok trzyma ostatni spójny stan systemu plików

Kilka reguł zachowania spójności

- nie wolno zapisać wskaźnika na niezainicjowaną strukturę
(i-węzeł musi być zainicjowany zanim ustalimy odpowiadający wpis katalogu)
- nie wolno ponownie użyć zasobu na który jeszcze ktoś wskazuje
(należy wyzerować wskaźnik w mapie bloków i-węzła, zanim ponownie użyjemy tego bloku)
- nie usuwaj starego wskaźnika póki nie ustaliłeś nowego
(zmieniając nazwę pliku – najpierw należy dodać nową nazwę, a potem usunąć starą)

Następujące akcje wymagają odpowiedniego uszeregowania operacji dyskowych, aby umożliwić sensowne wyjście z awarii systemu:

- utworzenie lub usunięcie pliku
- utworzenie lub usunięcie katalogu
- zmiana nazwy katalogu lub pliku
- przydział lub zwolnienie bloku
- operacja na bloku pośrednim (i-węzeł)

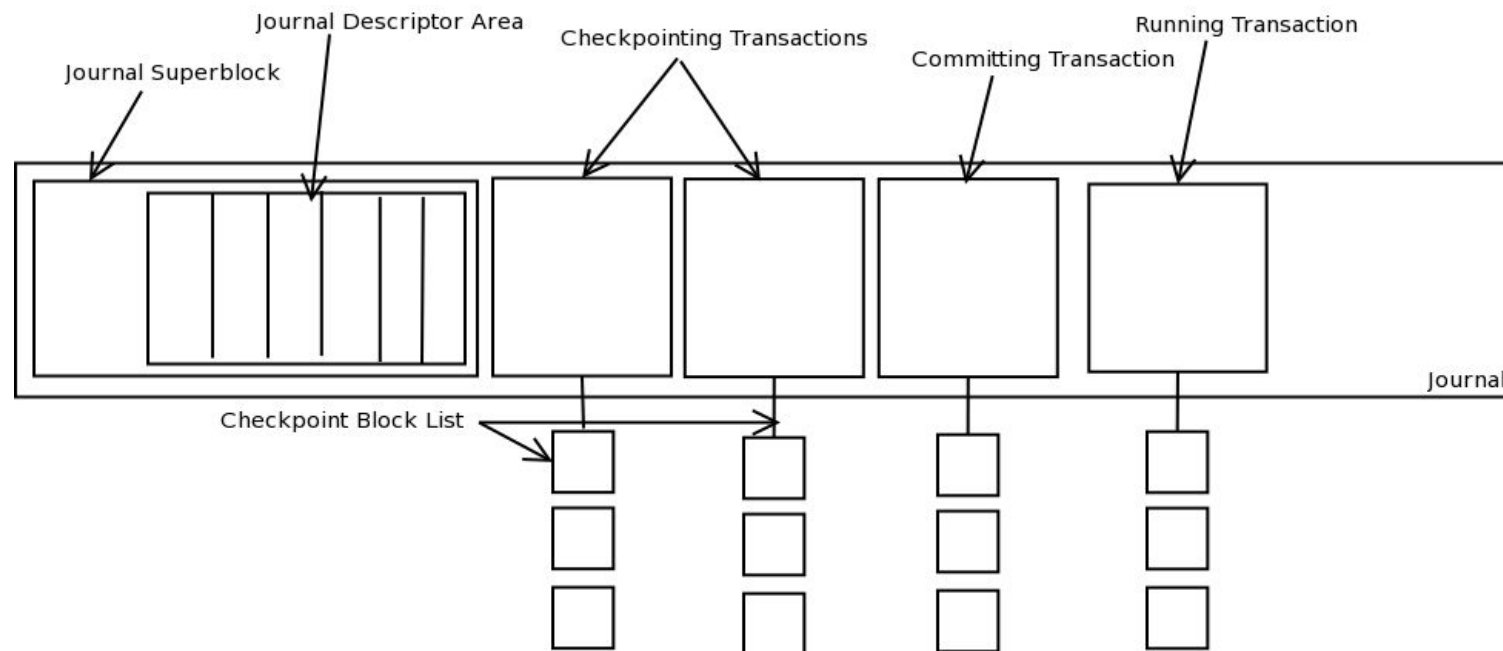
Księgowanie (ang. *journaling*)

Dziennik to miejsce na dysku przechowujące bufor cykliczny **transakcji** składających się z **operacji idempotentnych**. Wielokrotne wykonanie tej samej operacji idempotentnej nie zmienia wyniku: **zwiększ liczbę dowiązań twardych o 1** vs. **oznacz blok k jako używany**.

W trakcie działania system najpierw sekwencyjnie zapisuje modyfikacje (z reguły tylko metadanych) do dziennika. Gdy tylko upewni się, że operacje zaksięgowano, wdraża je w życie, po czym unieważnia odpowiednie wpisy dziennika. Po awarii **fsck** musi zlokalizować czoło i ogon kolejki transakcji, odtworzyć je i sprawdzić spójność metadanych.

Przykładowe operacje wymagające księgowania: zmiana licznika referencji i-węzła, przydział lub zwolnienie bloku, zmiana położenia wpisu katalogu (kompaktowanie).

Journalling Block Device (na podstawie ext3)



Journal: Logical View

JBD jest niezależny od systemu plików. Jedyną idempotentną operacją jest zapis bloku. Każda transakcja składa się z wielu zmodyfikowanych bloków. Blok k może występować w transakcji dokładnie raz.

Journalling Block Device (c.d.)

1. **Wystartuj nową transakcję (stan: in progress)**
zaktualizuj dane jednego lub więcej bloków w obrębie transakcji
2. **Zakończ transakcję (stan: completed)**
transakcja przechowywana w RAM, nie można już modyfikować
3. **Wypisanie transakcji (stan: committed)**
jedną lub więcej transakcji wpisujemy do dziennika i oznaczamy jako kompletną; transakcje mogą być odtworzone z dziennika po awarii
4. **Przechodzenie punktów kontrolnych (stan: checkpointed)**
modyfikacje są wpisywane do systemu plików, po czym transakcje są usuwane z dziennika; metadane powinny zostać zapisane na dysk przed zapisem danych

W **ext3** dziennik to specjalny plik (i-węzeł 8). Domyślnie księgowane są tylko modyfikacje metadanych, ale można również księgować dane.

Pytania?