SYSTEM PLIKÓW EXT

Plan wykładu

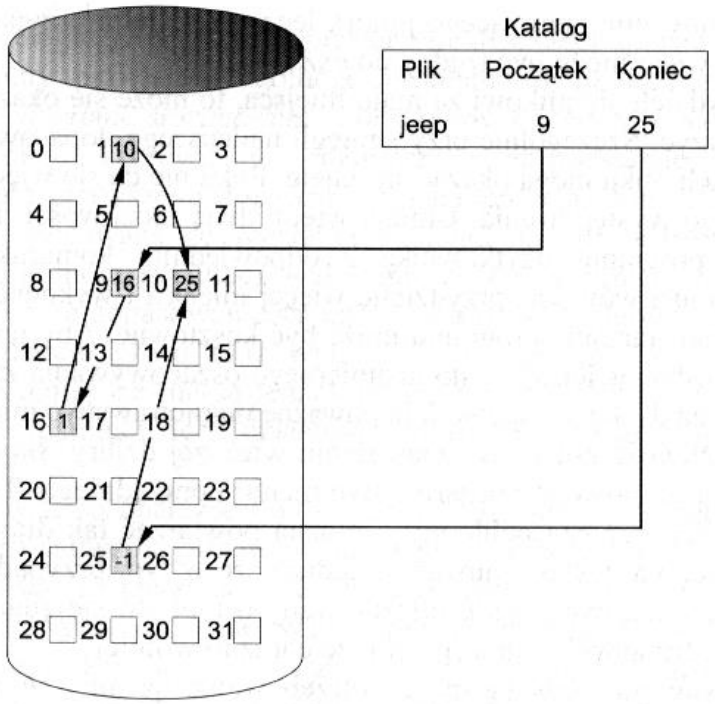
- Wprowadzenie
- Struktury danych na dysku
- Struktury danych w pamięci
- Tworzenie systemu plików
- Operacje
- Zarządzanie przestrzenią dyskową

Daniel P. Bovet, Marco Cesati **Understanding the Linux Kernel**
Mingming Cao, Suparna Bhattacharya, Ted Tso **Ext4: The Next Generation of Ext2/3 Filesystem**
Steve D. Pate **UNIX filesystems – evolution, design and implementation**

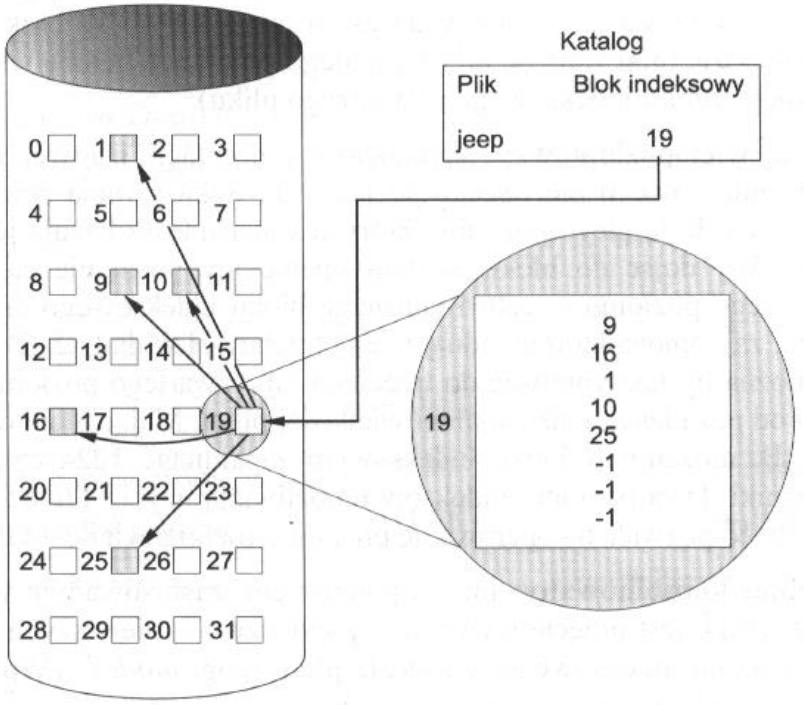
Wprowadzenie

- **Wprowadzenie**
- Struktury danych na dysku
- Struktury danych w pamięci
- Tworzenie systemu plików
- Operacje
- Zarządzanie przestrzenią dyskową

Przydział listowy a przydział indeksowy miejsca na dysku



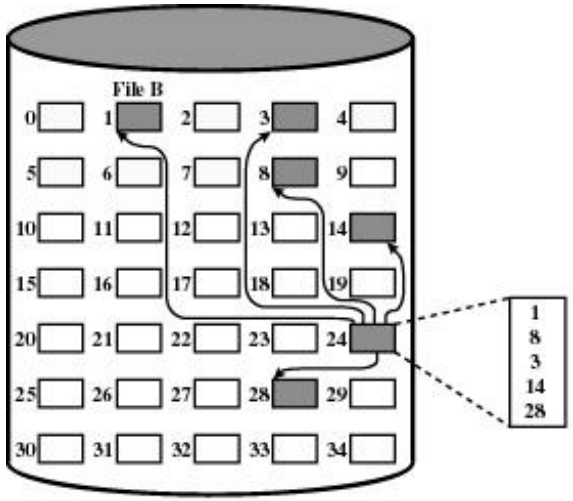
- Katalog zawiera wskaźnik do pierwszego i ostatniego bloku pliku
- Każdy blok pliku zawiera wskaźnik do następnego bloku (poza ostatnim)
- Brak fragmentacji zewnętrznej
- Efektywny tylko dostęp sekwencyjny (dotyczy najprostszej wersji przydziału)
- Podatność na awarie (czemu?)



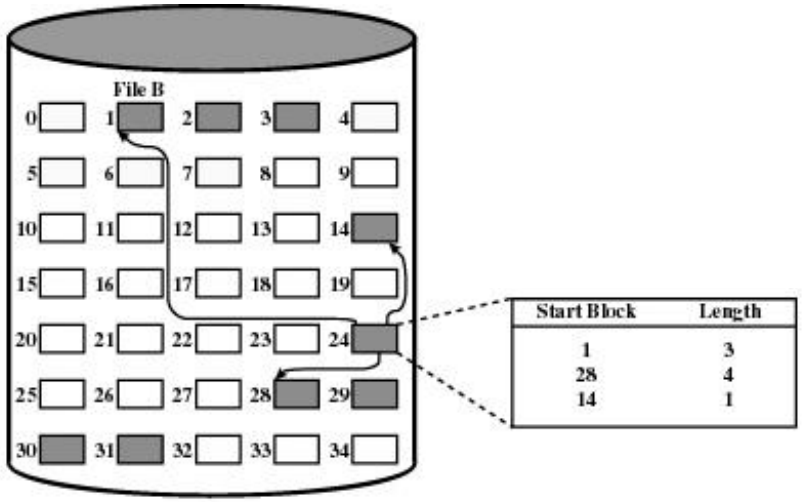
- Skupienie wszystkich wskaźników do bloków w bloku indeksowym
- Każdy plik ma własny blok indeksowy
- Pozycja o numerze *i* w bloku indeksowym wskazuje na blok *i* danego pliku.
- Katalog zawiera adres bloku indeksowego pliku
- Wskaźniki bloku indeksowego zajmują zazwyczaj więcej przestrzeni niż wskaźniki listowe

Przydział indeksowy

Wersja z blokami, np.: ext2, ext3, ext4

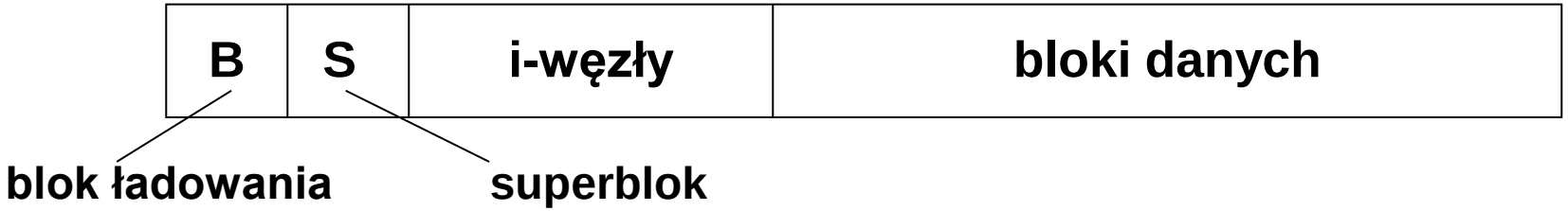


Wersja z zakresami, np.: ext4 (użycie tzw. extentów obok wersji podstawowej z pojedynczymi blokami)



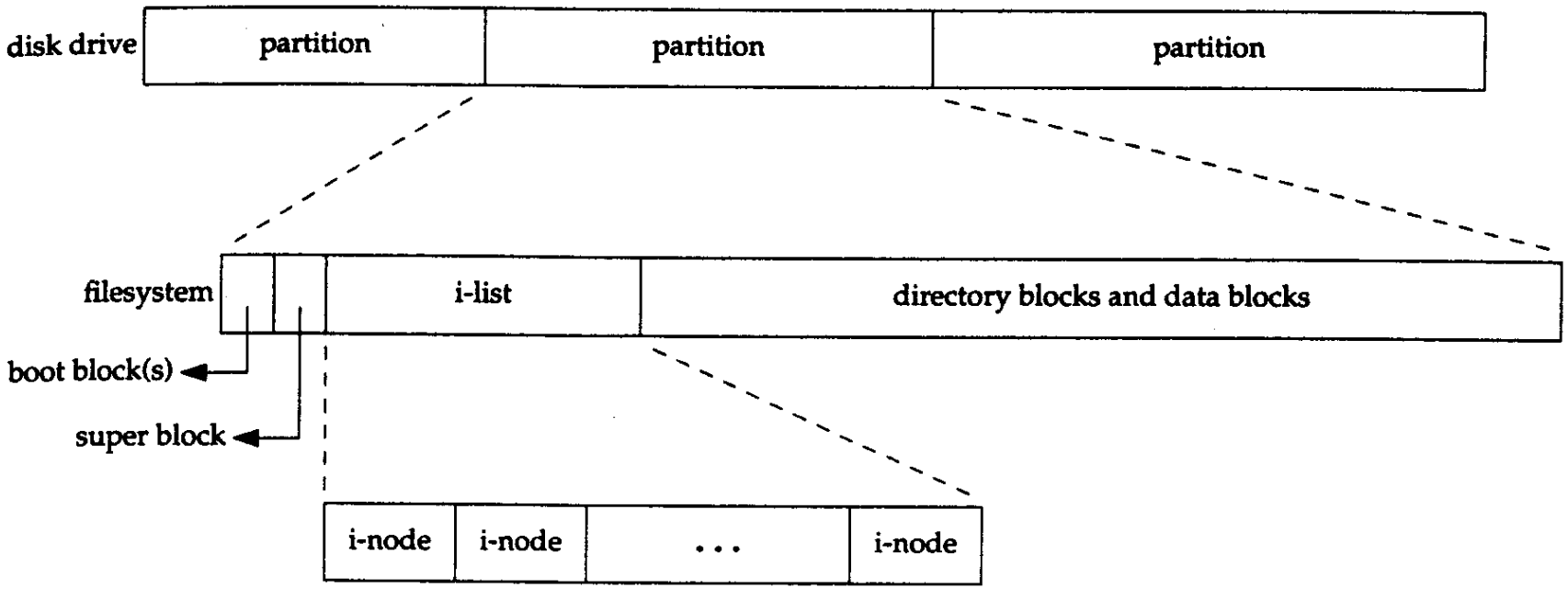
Klasyczna struktura dysku w systemie Unix (indeksowa)

Pojedyncza partycja:

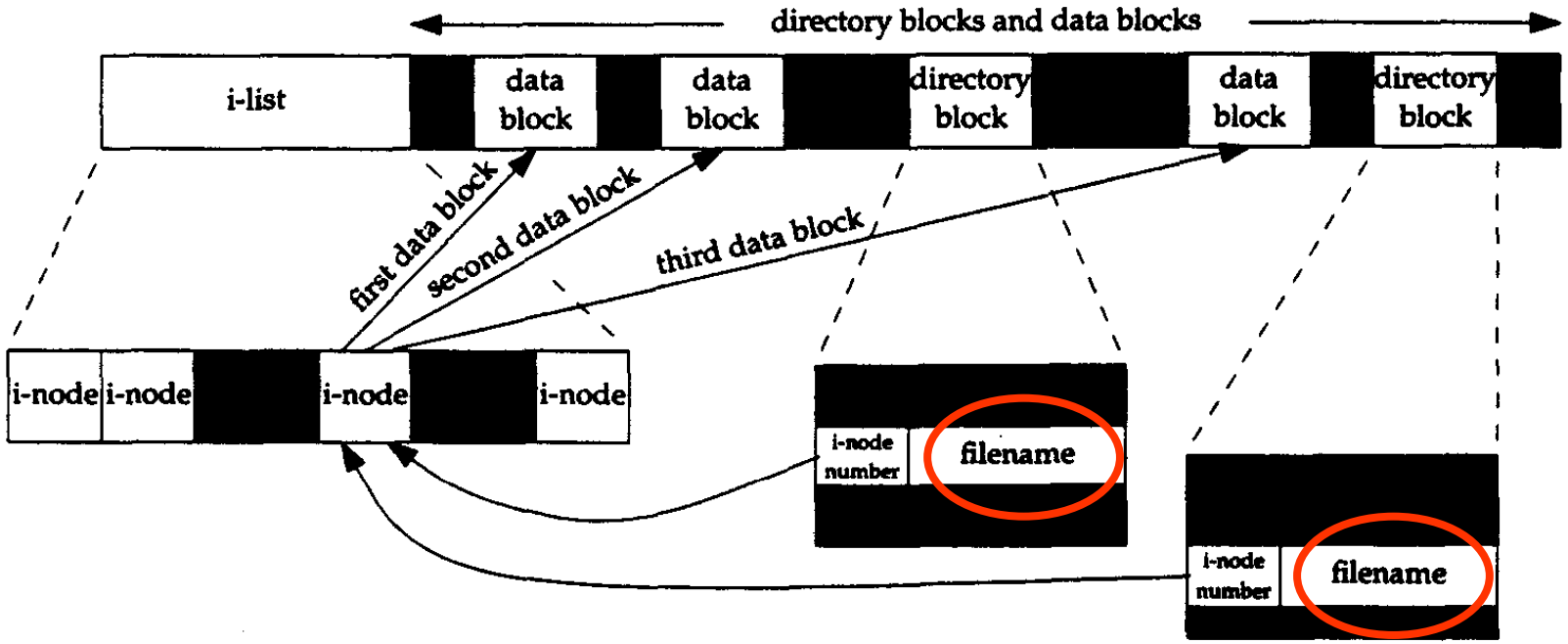


- **blok ładowania** (ang. *boot sector*) – kod ładujący system
- **superblok** – blok zawierający atrybuty danego systemu plików (rozmiar systemu, lista wolnych bloków, lista wolnych i-węzłów, itp.) . Jadro pobiera informacje z superbloku podczas montowania systemu plików.
- **i-węzły** – ustalonej wielkości, zawierają informację o pliku (ograniczona liczba plików, która może zostać utworzona - może się zdarzyć, że na dysku jest miejsce, ale tabela i-węzłów jest pełna)
- **bloki danych** – zawierają pliki, katalogi i bloki adresowania pośredniego

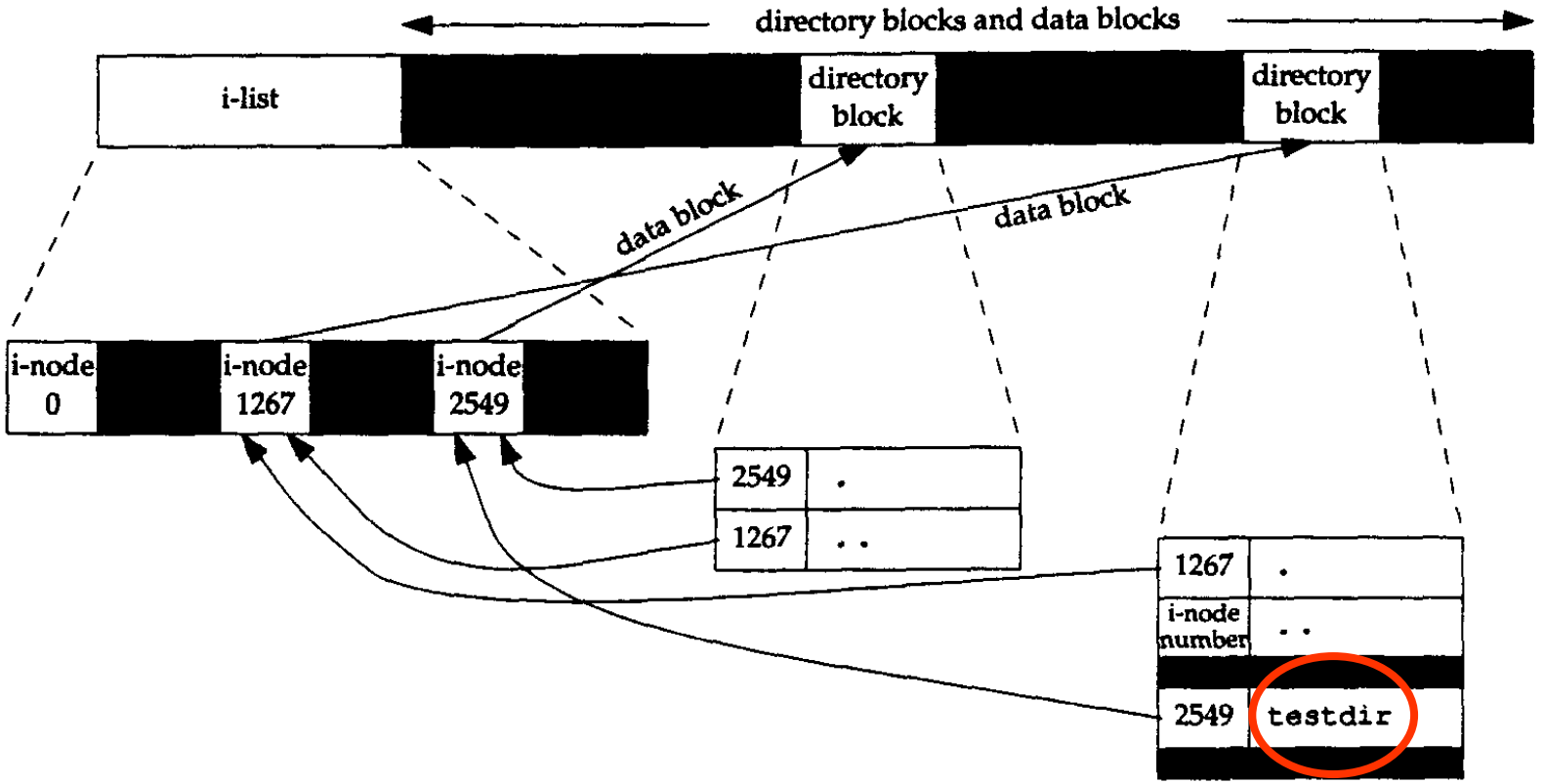
Urządzenie, partycja i system plików



Szczegóły systemu plików

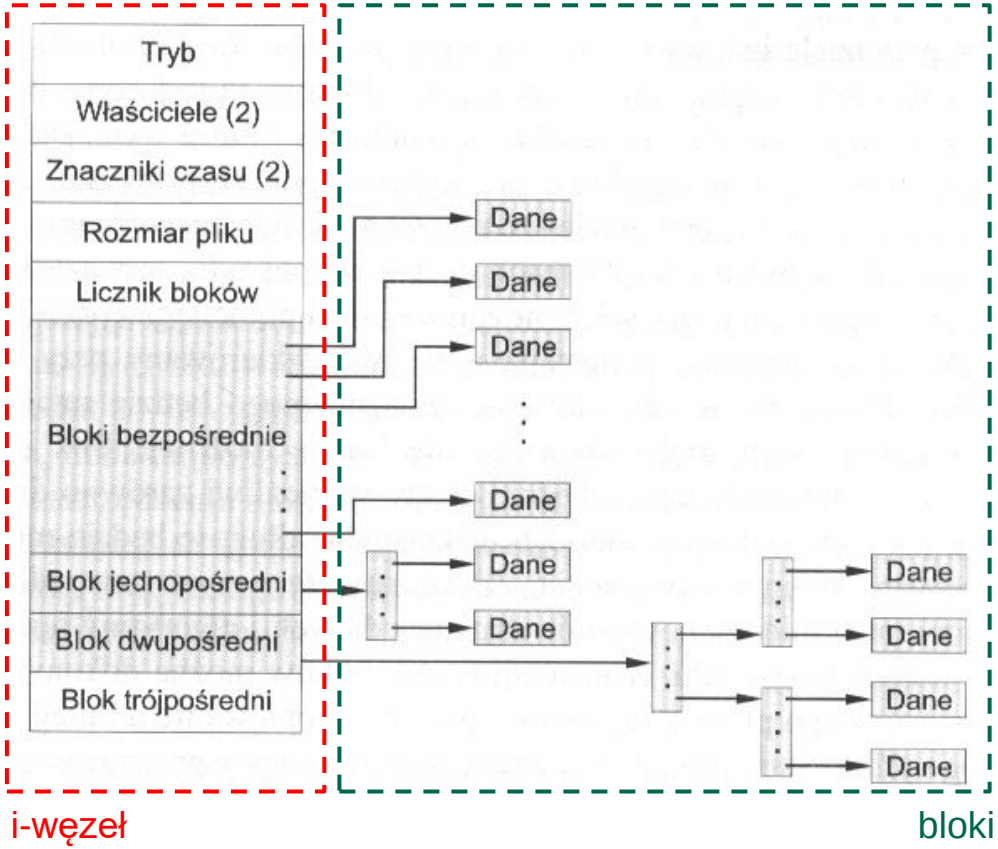


System plików po utworzeniu katalogu **testdir**



Adresowanie bloków danych

Schemat kombinowany: adresowanie bezpośrednie (lista) i pośrednie (wielopoziomowe) – system **BSD**



- **i-węzeł** zawiera m.in. **15 wskaźników** bloku indeksowego
- **12 pierwszych** wskazuje **bezpośrednio** na bloki dyskowe pliku (czyli jeżeli blok ma 4KB to bezpośrednio możemy adresować 48KB plik)
- kolejne 3 wskaźniki to wskaźniki do bloku **jedno-**, **dwu-** i **trójpośredniego**

Systemy plików Ext

Minix

Zalety: istniał i stabilnie działał :)
Wady: bloki adresowane na 16 bitach – rozmiar systemu do **64MB**; wpisy katalogowe mają stały rozmiar – nazwy plików maksymalnie **14** znakowe.

Ext

Zalety: rozmiar systemu do **2GB**, nazwy plików do **255** znaków.
Wady: brak w i-węźle rozróżnienia na czas dostępu, modyfikacji i tworzenia; do zarządzania wolnymi blokami i i-węzłami używane jedynie listy (bez map bitowych); duża fragmentacja.

Ext2

Zalety: rozmiar systemu do **16384GB**, rozmiar pliku do **2048GB**; mniejsza fragmentacja; automatyczne rozpoznawanie uszkodzeń systemu plików (lost+found).
Wady: podatność na awarie całego systemu.

Ext3

Zalety: obsługa mechanizmu księgowania (ang. *journalling*).
Wady: nie ma możliwości prostego odzyskiwania plików (tak jak w ext2).

Ext4

Zalety: obsługa woluminu do **1024PB**; mniejsza fragmentacja – nowe pliki tworzone w ciągłym obszarze dysku (mechanizm rozszerzeń).
Wady: dyskusyjne bezpieczeństwo danych przy opóźnionej alokacji

System plików EXT2

System plików EXT2 na dysku składa się z wielu **grup bloków dyskowych**, przy czym pojedynczy blok uważamy za podstawową jednostkę objętościową danych.

Wielkość bloku dyskowego jest **stała** w ramach całego systemu plików.

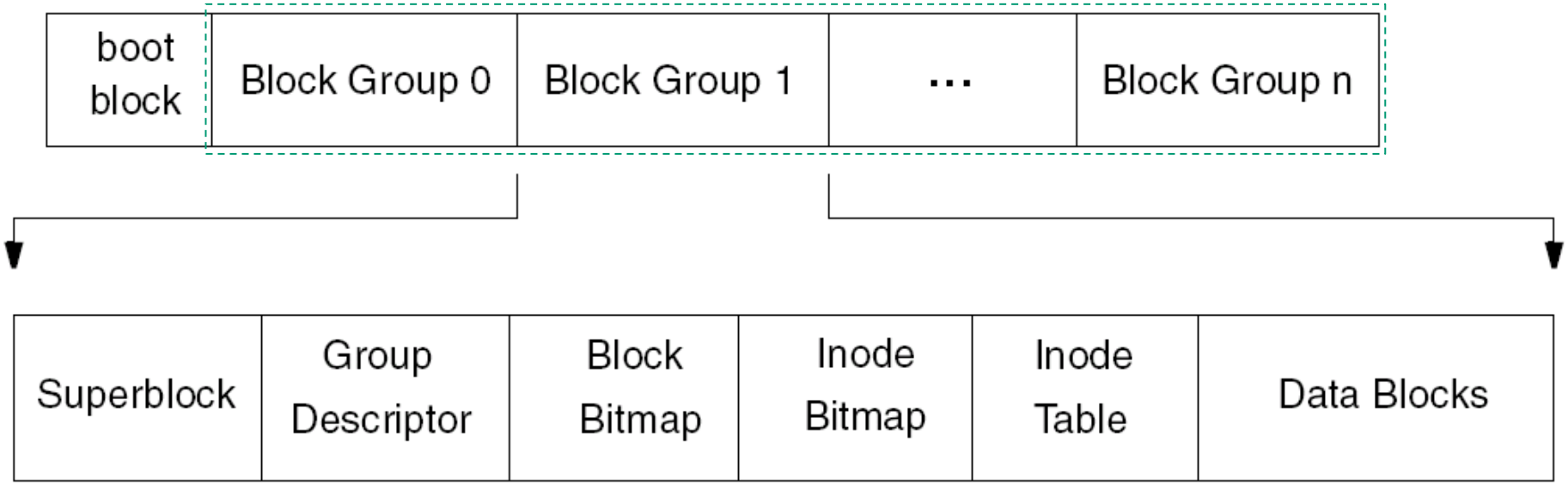
Stałe ograniczające rozmiar bloku dyskowego znajdują się w pliku nagłówkowym **include/linux/ext2_fs.h**:

```
#define   EXT2_MIN_BLOCK_SIZE      1024
#define   EXT2_MAX_BLOCK_SIZE      4096
```

dla architektur 64 bitowych max blok ma **8KB** (system do 32TB)

Struktura partycji

1024B niezależnie od wielkości bloku dyskowego
(*czyli pierwszy superblok może zaczynać się w 0 albo 1 bloku dyskowym*)



Boot blok w partycji typu EXT2 nigdy **nie jest zarządzany przez system** plików EXT2.

Typy bloków

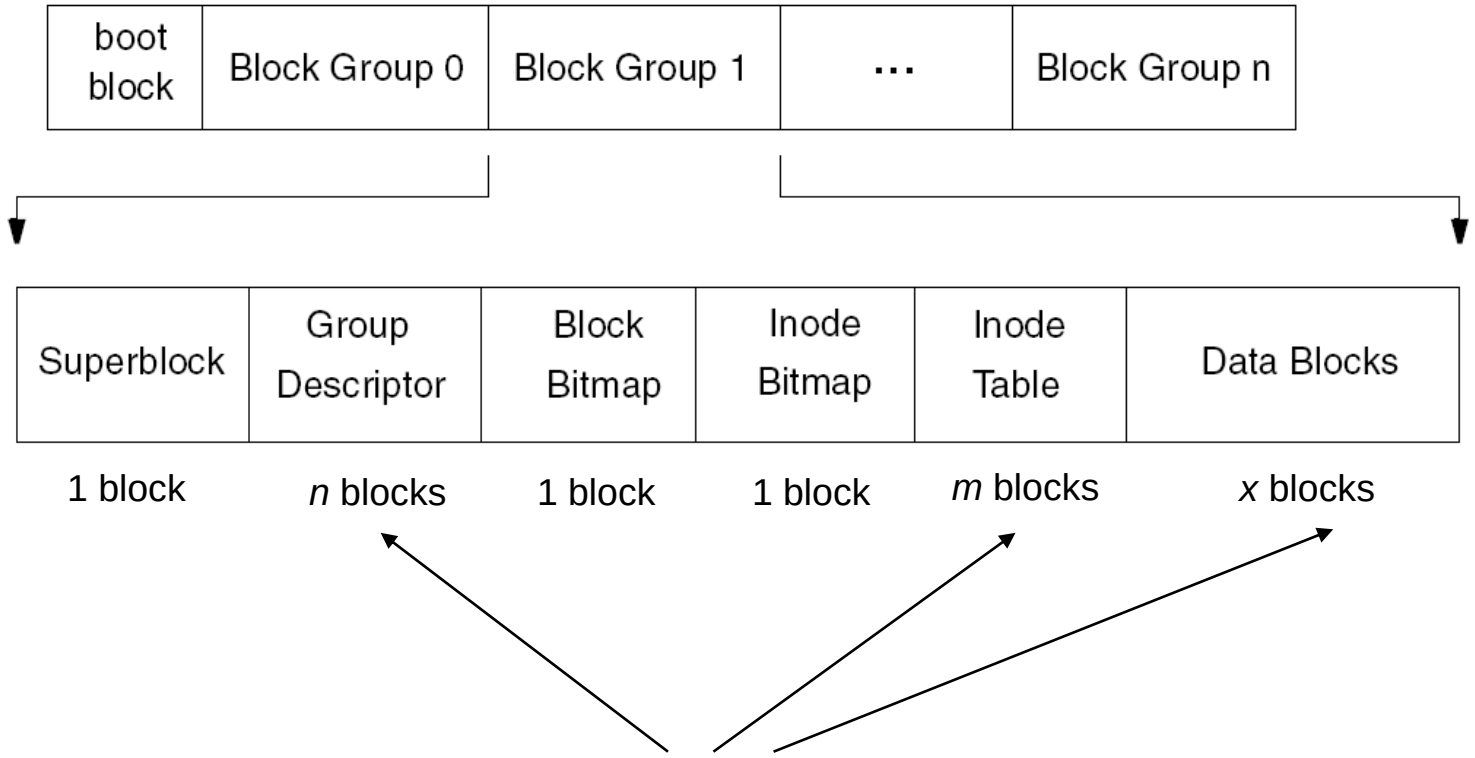
System plików EXT2 na dysku składa się z wielu **grup bloków dyskowych**. **Każda grupa** składa się z następujących części:

- **Superblok** - zawiera kluczowe informacje o partycji i formacie danych (jak np. liczba i-węzłów i bloków na partycji, liczba wolnych i-węzłów i bloków). **Superbloki we wszystkich grupach mają tę samą zawartość.**
- **Deskryptory grup** - zawierają informacje o miejscach, w których zaczynają się pozostałe części grup i o liczbie wolnych i-węzłów i bloków w grupach. **Podobnie jak w przypadku superbloków ich zawartość jest skopiowana do wszystkich grup**
- **Bitmapa bloków** (1 blok dyskowy) - tablica bitowa zawierająca informacje o zajętości bloków należących do grupy. Każdy bit odpowiada najmniejszej jednostce na dysku, która może być zapisana bądź nie, czyli blokowi.
- **Bitmapa i-węzłów** (1 blok dyskowy) - jak wyżej, tylko dotyczy i-węzłów w grupie.
- **Tablica i-węzłów** - obszar na dysku zawierający i-węzły z danej grupy.
- **Bloki danych**.

Struktury danych na dysku

- Wprowadzenie
- **Struktury danych na dysku**
- Struktury danych w pamięci
- Tworzenie systemu plików
- Operacje
- Zarządzanie przestrzenią dyskową

Logiczna struktura dysku w systemie EXT2



struktury mogące zajmować więcej niż 1 blok

Podstawowe zależności 1/2

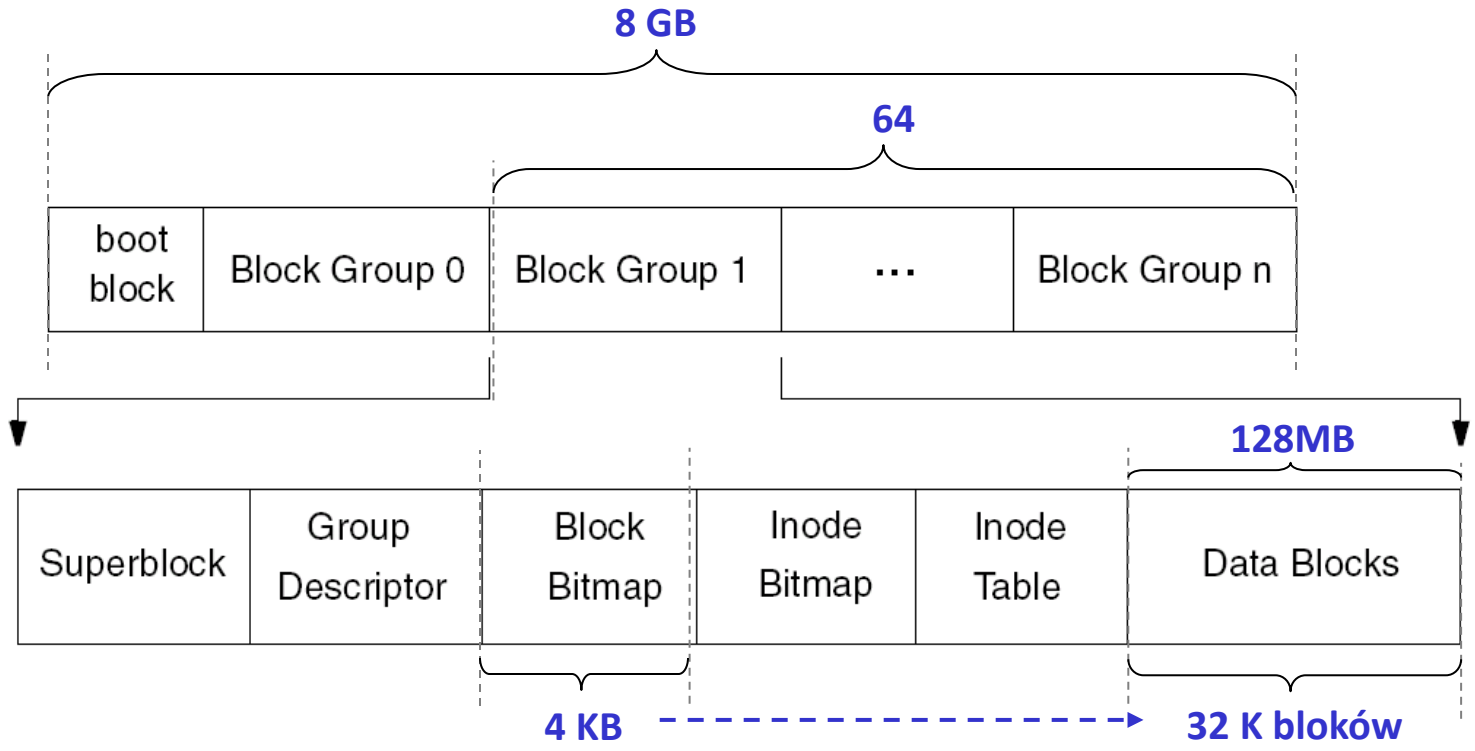
Jądro systemu korzysta jedynie z **superbloku i i deskryptorów grupy** znajdujących się w **zerowej grupie bloków**.

Gdy program **e2fsck** bada spójność systemu plików, odwołuje się do superbloku i zerowego bloku grupy, a następnie tworzy kopie w pozostałych blokach grupy (służą one do odtworzenia systemu w przypadku awarii).

Liczba grup bloków zależy od rozmiaru partycji i wielkości bloku. **Bitmapa** grupy bloków (wolnych i zajętych) musi być przechowywana w pojedynczym bloku. Z tego powodu każda grupa bloków może zawierać $8 \times b$ bloków, gdzie b jest rozmiarem bloku w bajtach. Wynika stąd, że całkowita liczba grup bloków wynosi w przybliżeniu $s/(8 \times b)$, gdzie s jest rozmiarem partycji w blokach.

Przykład

Rozmiar partycji EXT2 - 8 GB, rozmiar bloku **4 KB**. W tym przypadku każda **4 KB** bitmapa bloków opisuje **32K** bloków danych, tj. **128 MB**. Niezbędne są więc **64** grupy bloków. **Zasada**: im mniejszy rozmiar bloku, tym większa liczba grup bloków.



Podstawowe zależności 2/2

i-węzły oraz bloki mają numerację globalną (a nie lokalną w ramach pojedynczej grupy)

i-węzły oraz bloki danych indeksowane są od 1

Katalog główny ma i-węzeł nr **2**

indetifier	**value**	**description**
`EXT2_BAD_INO`	1	Bad blocks inode
`EXT2_ROOT_INO`	**2**	**Root inode**
`EXT2_ACL_IDX_INO`	3	ACL inode
`EXT2_ACL_DATA_INO`	4	ACL inode
`EXT2_BOOT_LOADER_INO`	5	Boot loader inode
`EXT2_UNDEL_DIR_INO`	6	Undelete directory inode
`EXT2_FIRST_INO`	11	First non reserved inode

Superblok EXT2

```
struct ext2_super_block {
   unsigned long s_inodes_count;         //Total number of inodes
   unsigned long s_blocks_count;         //Filesystem size in blocks
   unsigned long s_r_blocks_count;       //Number of reserved blocks
   unsigned long s_free_blocks_count;    //Free blocks counter
   unsigned long s_free_inodes_count;    //Free inodes counter
   unsigned long s_first_data_block;     //Number of first useful block
   unsigned long s_log_block_size;       //Block size
   long s_log_frag_size;                 //Fragment size
   unsigned long s_blocks_per_group;     //Number of blocks per group
   unsigned long s_frags_per_group;      //Number of fragments per group
   unsigned long s_inodes_per_group;     //Number of inodes per group
   unsigned long s_mtime;                //Time of last mount operation
   unsigned long s_wtime;                //Time of last write operation
   unsigned short s_mnt_count;           //Mount operations counter
   short s_max_mnt_count;  //Number of mount operations before check
   unsigned short s_magic;               //Magic signature
   unsigned short s_state;               //Status flag
   unsigned short s_errors;     //Behavior when detecting errors
   unsigned short s_minor_rev_level;     //Minor revision level
   unsigned long s_lastcheck;            //Time of last check
   unsigned long s_checkinterval;        //Time between checks
   unsigned long s_reserved[238];
};
```

Superblok EXT2 – opis niektórych pól

- **s_inodes_count** - pole przechowuje liczbę liczbę i-węzłów, zaś **s_blocks_count** – liczbę bloków w systemie EXT2
- **s_log_block_size** – wyraża rozmiar bloku jako potęgę 2, przy czym jednostką danych jest 1024 bajtów; stąd 0 oznacza bloki 1024 bajtowe, 1 bloki 2048 bajtowe, itd.
- **s_log_frag_size** jest równe **s_log_block_size** (brak implementacji fragmentacji bloków
- pola **s_blocks_per_group**, **s_frags_per_group** i **s_inodes_per_group** przechowują odpowiednio liczbę bloków, fragmentów i i-węzłów przypadająca na jedna grupę bloków
- pola **s_mnt_count**, **s_max_mnt_count**, **s_lastcheck** i **s_checkinterval** wymuszają automatyczną weryfikację EXT2 podczas ładowania systemu za pomocą **e2fsck** wtedy, gdy przekroczona określona liczbę montowań lub minęło wystarczająco dużo czasu od ostatniej weryfikacji plików
- **s_state** – zawiera 0, gdy system jest zamontowany lub niewłaściwie odmontowany, 1 – odmontowany, 2 – system zawiera błędy

Deskryptor grupy i bitmapa

```
struct ext2_group_desc {
  __u32 bg_block_bitmap;/* Block number of bitmap block */
  __u32 bg_inode_bitmap;/* Block number of inode bitmap*/
  __u32 bg_inode_table; /* Block number of first inode table block */
  __u16 bg_free_blocks_count; /* Free blocks count */
  __u16 bg_free_inodes_count; /* Free inodes count */
  __u16 bg_used_dirs_count;   /* Directories count */
  __u16 bg_pad;
  __u32 bg_reserved[3];
};
```

Pola **bg_free_blocks_count**, **bg_free_inodes_count** i **bg_used_dirs_count** są używane wtedy, gdy alokowane są nowe i-węzły i bloki danych;

Bitmapy są ciągami bitów, gdzie wartość 0 oznacza, że odpowiedni i-węzeł lub blok danych są wolne, zaś wartość 1 oznacza, że są zajęte. Ponieważ każda mapa bitowa jest przechowywana w jednym bloku, a blok może mieć rozmiar **1024**, **2048** lub **4096** bajtów, to jedna bitmapa opisuje stan **8192**, **16384** lub **32768** bloków.

I-węzeł

```
struct ext2_inode {
        __u16       i_mode;     /* typ pliku -  zwykły, katalog,
                                   FIFO, specjalny..., 0 gdy wolny */
        __u16       i_uid;      /* identyfikator właściciela pliku */
        __u32       i_size;     /* rozmiar pliku w bajtach – max. 4GB */
        __u32       i_atime;    /* ostatni czas dostępu */
        __u32       i_ctime;    /* czas utworzenia */
        __u32       i_mtime;    /* czas ostatnich zmian */
        __u32       i_dtime;    /* czas skasowania */
        __u16       i_gid;      /* identyfikator grupy właściciela pliku */
        __u16       i_links_count;  /* ilość dowiązań do pliku */
        __u32       i_blocks;    /* ilość bloków dyskowych w pliku */
        __u32       i_file_acl;  /* lista kontroli dostępu pliku */
        __u32       i_dir_acl;   /* lista kontroli dostępu katalogu */
        __u32       i_faddr;     /* adres fragmentu */

        ......
        __u32       i_block[EXT2_N_BLOCKS]; /* tablica opisu bloków pliku
                                               - zawiera adresy bloków
                                                reprezentujących plik */
}
```

Tablica i-węzłów

Tablica i-węzłów składa się z ciągu sąsiadujących bloków, z których każdy zawiera określoną liczbę i-węzłów.

Numer pierwszego bloku tablicy jest przechowywany w polu **bg_inode_table** deskryptora grupy.

Każdy i-węzeł ma rozmiar **128 bajtów**. Stąd 1024 bajtowy blok zwiera 8 i-węzłów, a blok 4096 – 32 i-węzły.

Aby obliczyć liczbę bloków zajmowanych przez i-węzły należy podzielić całkowitą liczbę i-węzłów w grupie (patrz pole **s_inodes_per_group** superbloku) przez ilość i-węzłów w bloku.

Rozmiar pliku

Rozmiar pliku w bajtach (**i_size**) może być różny od rozmiaru obliczonego na podstawie ilości zajmowanych bloków (**i_blocks**).

Standardowo rozmiar pliku jest ograniczony do 2GB (nie są wykorzystywane wszystkie bity pola **i_size**). Istnieje możliwość powiększenia 32 bitowego pola **i_size** o kolejne 32 bity nieużywanego pola **i_dir_acl**. Metoda możliwa do stosowania jedynie w architekturach 64 bitowych.

W strukturze i-węzła nie ma pola zawierającego jego numer. Numer i-węzła jest jednoznacznie określony przez jego położenie na dysku.

Przykład: w systemie gdzie pojedyncza grupa bloków zawiera **4096** i-węzłów chcemy wskazać i-węzeł o numerze **13021**. Odpowiedni i-węzeł będzie to **733** - eci wpis w tablicy i-węzłów **trzeciej grupy bloków**.

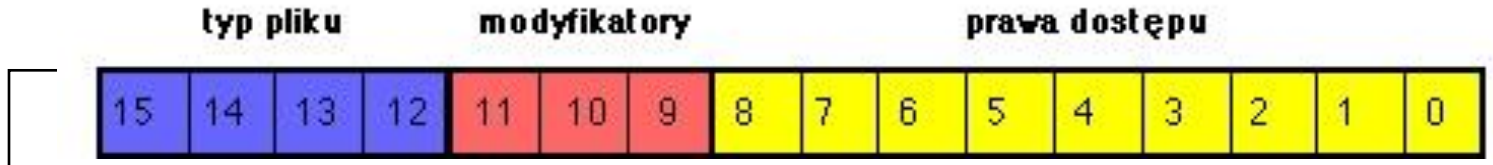
Typy plików (1/2)

W systemie plików EXT2 w każdym i-węźle jest pamiętany typ pliku (zmienna **i_mode** w strukturze **ext2_inode**); zapisywany jest na 4 najstarszych bitach słowa 16 bitowego).

Wartość bitów typu pliku dla różnych rodzajów plików:

```
04h - katalog
08h - plik zwykły
0Ah - dowiązanie symboliczne
06h - plik specjalny blokowy
02h - plik specjalny znakowy
01h - łącze nazwane (FIFO)
```

Typy plików (2/2)



Stała symboliczna	Wartość w kodzie ósemkowym	Znaczenie
S_IFMT	0170000	maska bitowa typu pliku
S_IFSOCK	0140000	gniazdo
S_IFLNK	0120000	dowiązanie symboliczne
S_IFREG	0100000	plik zwykły
S_IFBLK	0060000	plik specjalny blokowy
S_IFDIR	0040000	katalog
S_IFCHR	0020000	plik specjalny znakowy
S_IFIFO	0010000	kolejka FIFO

Nazwa bitu	Stała	Wartość	Znaczenie bitu
bit ustanowienia użytkownika (ang. setuid bit)	S_ISUID	0004000	Bit ma znaczenie tylko dla plików wykonywalnych. Powoduje, że proces wykonuje program z identyfikatorem obowiązującym użytkownika ustawionym na identyfikator właściciela pliku.
bit ustanowienia grupy (ang. setgid bit)	S_ISGID	0002000	Bit ma znaczenie tylko dla plików wykonywalnych. Powoduje, że proces wykonuje program z identyfikatorem obowiązującym grupy ustawionym na identyfikator grupy pliku.
bit lepkosci (ang. sticky bit)	S_ISVTX	0001000	Dla pliku zwykłego: obecnie bit jest ignorowany. Dawniej wymuszał pozostawienie kodu programu w pamięci po zakończeniu jego wykonywania. Dla katalogu: pliki mogą być usuwane tylko przez właściciela lub administratoraniezależnie od posiadania prawa pisania w katalogu.

Pliki zwykłe i „specjalne”

Pliki zwykłe – posiadają zero (puste) lub więcej bloków danych. Możliwa fragmentacja na poziomie całych bloków.

Dowiązania symboliczne – jeżeli rozmiar wskazywanej ścieżki nie jest większy niż 60 znaków, to ścieżka jest przechowywana w polu **i_block** i nie jest wykorzystywany żaden blok danych. W przeciwnym przypadku ścieżka zapisywana jest w jednym bloku danych, wskazywanym jak w zwykłym pliku.

Pliki urządzeń, potoki i gniazda – nie posiadają bloków danych. Wszystkie niezbędne informacje przechowywane w i-węźle.

Katalogi

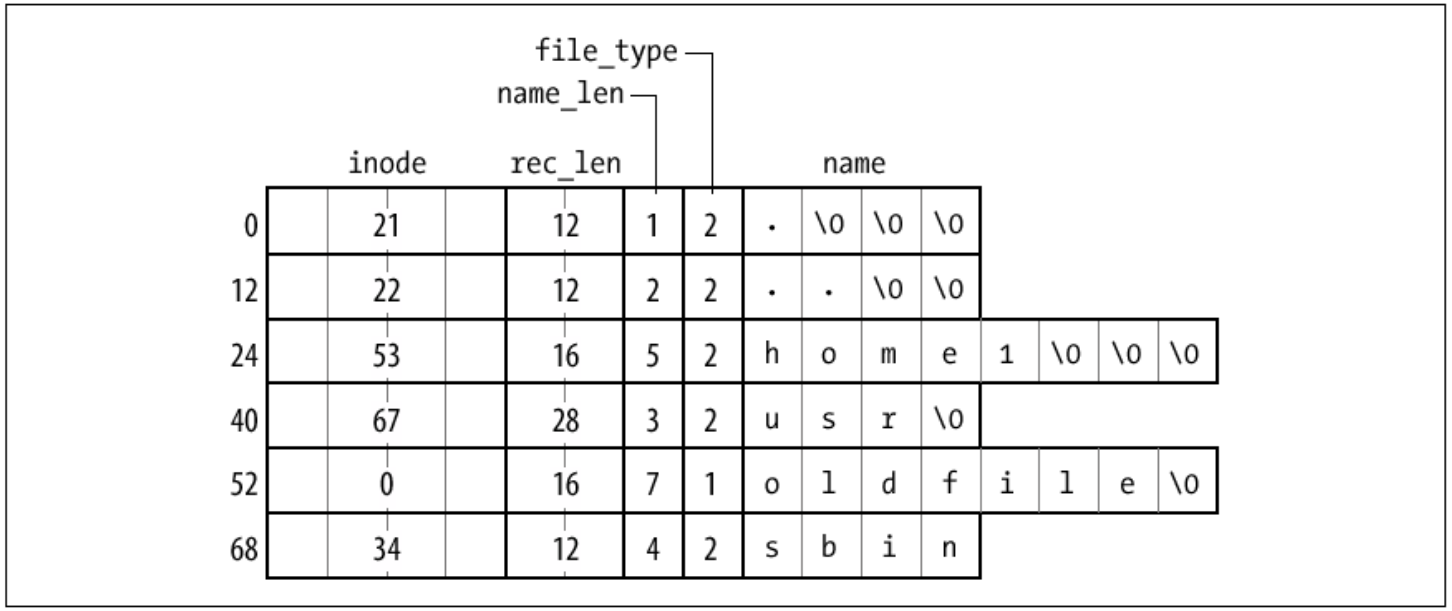
Katalogi – plik zawierające w blokach danych informacje o poszczególnych wpisach – wiązanie nazwy i i-węzła. Poszczególne wpisy są zgodne ze strukturą **ext2_dir_entry_2.**

```
struct ext2_dir_entry_2 {
  u32 inode;                    // Inode number
  u16 rec_len;                  // Directory entry length
  u8 name_len;                  // Filename length
  u8 file_type                  // File type
  char name[EXT2_NAME_LEN ]     //name Filename
};
```

Typy plików (pole **file_type**):

```
0 Unknown          1 Regular file
2 Directory        3 Character device
4 Block device     5 Named pipe
6 Socket           7 Symbolic link
```

Przykład katalogu



Pole **rec_len** wskazuje na następną poprawną pozycję w katalogu: stanowi offest, który należy dodać do adresu początkowego pozycji katalogu, aby otrzymać adres początkowy następnej prawidłowej pozycji.

Pozycja ***oldfile*** została skasowana (**inode** równa się 0), stąd **rec_len** wpisu *usr* jest ustawione na 12+16 (długość pozycji *usr* i *oldfile*)

Struktury danych w pamięci

- Wprowadzenie
- Struktury danych na dysku
- **Struktury danych w pamięci**
- Tworzenie systemu plików
- Operacje
- Zarządzanie przestrzenią dyskową

Pamięciowe struktury danych

Dla zwiększenia wydajności działania systemu plików większość struktur dyskowych jest kopiowana do pamięci.

Type	Disk data structure	Memory data structure	Caching mode
Superblock	`ext2_super_block`	`ext2_sb_info`	Always cached
Group descriptor	`ext2_group_desc`	`ext2_group_desc`	Always cached
Block bitmap	Bit array in block	Bit array in buffer	Fixed limit
Inode bitmap	Bit array in block	Bit array in buffer	Fixed limit
Inode	`ext2_inode`	`ext2_inode_info`	Dynamic
Data block	Unspecified	Buffer page	Dynamic
Free inode	`ext2_inode`	None	Never
Free block	Unspecified	None	Never

fixed limit – ograniczona ilość struktur może być jednocześnie przechowywana w pamięci

dynamic – struktura przechowywana w pamięci tak długo, jak długo używany jest obiekt z którym jest związana

Struktura ext2_sb_info

Po zamontowaniu systemu plikowego EXT2 aktualizowana jest specyficzne dla EXT2 pole **u superbloku VFS** – pole typu **ext2_sb_info**. Struktura ta poza większością pól superbloku dyskowego **ext2_super_block** zawiera informacje związane ze strukturami w pamięci, np.:

- **s_loaded_inodes_bitmaps** liczba wczytanych bitmap zajętości i-węzłów do pamięci,
- **s_inode_bitmap_number** tablica z numerami wczytanych bitmap zajętości i-węzłów,
- **s_inode_bitmap** dowiązanie do bufora zawierającego tablicę wczytanych z dysku bitmap zajętości i-węzłów,
- **s_loaded_blocks_bitmaps** liczba wczytanych bitmap zajętości bloków dyskowych do pamięci,
- **s_block_bitmap_number** tablica z numerami wczytanych bitmap zajętości bloków dyskowych,
- **s_block_bitmap** dowiązanie do bufora zawierającego tablicę wczytanych z dysku bitmap zajętości bloków dyskowych

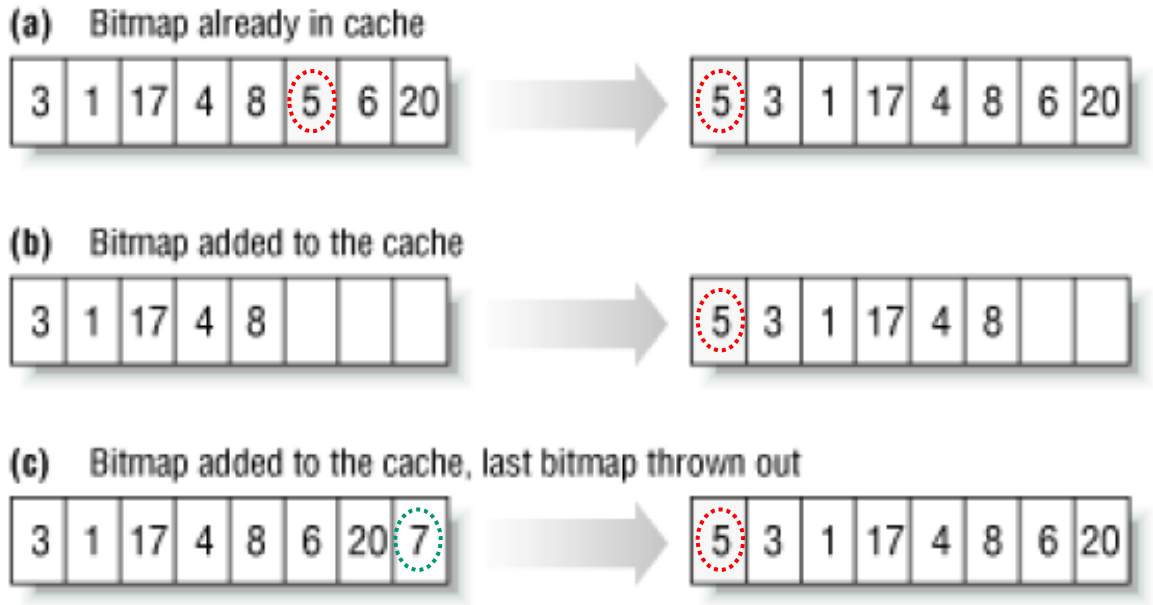
Struktura ext2_inode_info

Podobnie podczas inicjalizowania i-węzła w pamięci pole **u** jest ustawiane na strukturę **ext2_inode_info**. Jest tam umieszczana większość informacji nie przewidziana w domyślnej strukturze i-węzła VFS, m.in.:

- numer i rozmiar **fragmentu**
- indeks grupy bloków do której należy węzeł **block_group**
- pola wykorzystywane do prealokacji – **i_alloc_block** oraz **i_alloc_count**

Bufory bitmap

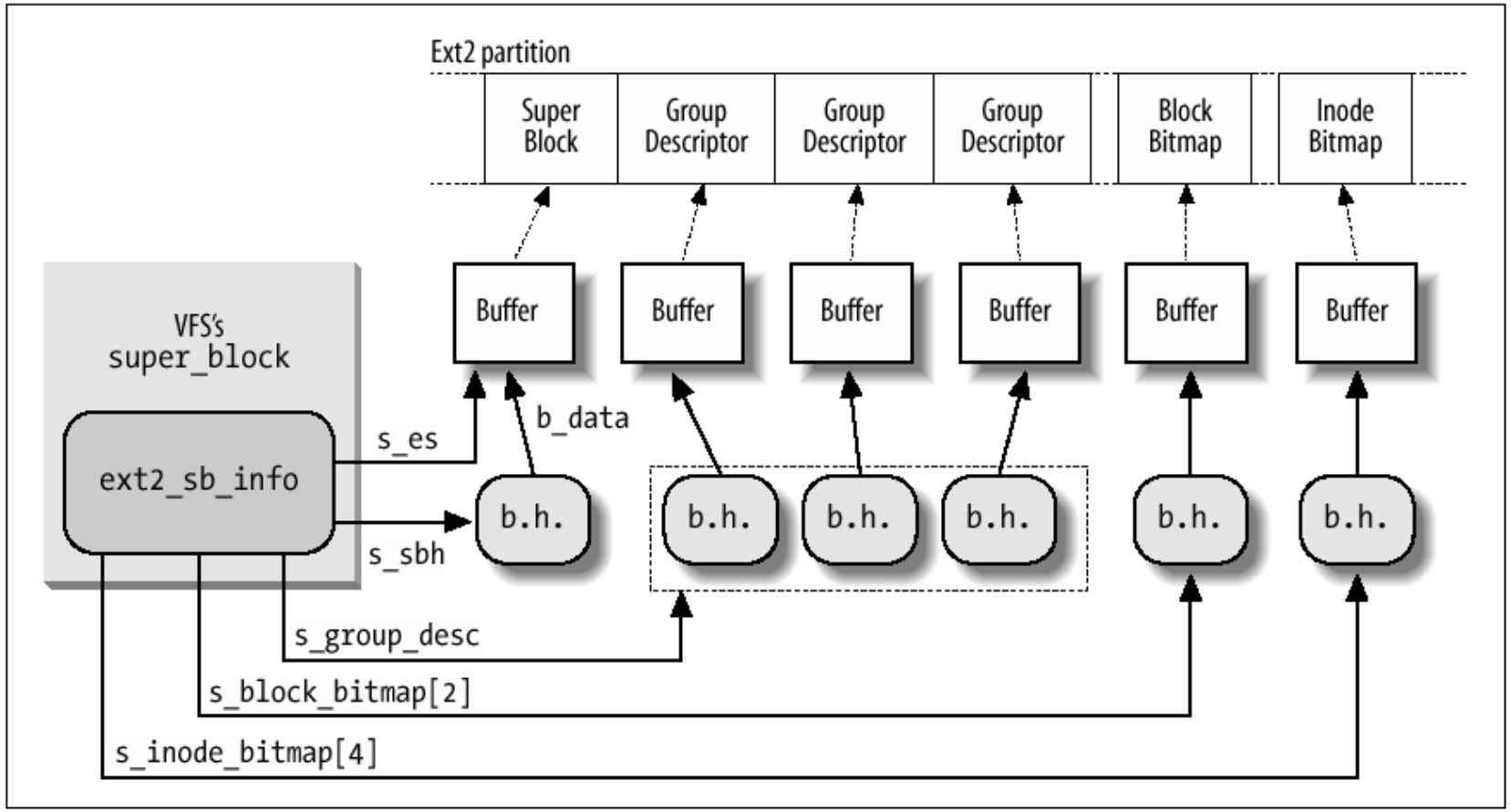
Z powodu ograniczonej ilości pamięci operacyjnej bitmapy nie mogą być w całości buforowane w pamięci. System używa dwa bufory (bitmapy bloków i i-węzłów) o rozmiarze **EXT2_MAX_GROUP_LOADED**.




```
load_block_bitmap(...)
        ⬇
read_block_bitmap(...)
```

```
load_inode_bitmap(...)
        ⬇
read_inode_bitmap(...)
```

Zależności pomiędzy strukturami



Tworzenie systemu plików

- Wprowadzenie
- Struktury danych na dysku
- Struktury danych w pamięci
- **Tworzenie systemu plików**
- Operacje
- Zarządzanie przestrzenią dyskową

Formatowanie

/sbin/mke2fs

1. inicjalizacja superbloku i deskryptorów grup
2. dla każdej grupy rezerwowanie bloków potrzebnych do przechowywania superbloku, deskryptora grup, tablicy i-węzłów i bitmap
3. inicjalizacja wszystkich bitmap
4. inicjalizacja tablic i-węzłów
5. utworzenie katalogu */*
6. utworzenie katalogu *lost+found*
7. aktualizacja bitmap

Przykład

System Ext2 na dyskietce 1.4MB, domyślne ustawienia systemu (blok 1024B, fragmenty wielkości bloków, 4K i-węzłów w grupie, 5% rezerwowanych bloków)

Block	Content
0	Boot block
1	Superblock
2	Block containing a single block group descriptor
3	Data block bitmap
4	Inode bitmap
5-49	Inode table: inodes up to 10: reserved; inode 11: *lost+found*; inodes 12-360: free
50	Root directory (includes `.`, `...`, and *lost+found*)
51	*lost+found* directory (includes `.` and `..`)
52-62	Reserved blocks preallocated for *lost+found* directory
63-1439	Free blocks

Operacje

- Wprowadzenie
- Struktury danych na dysku
- Struktury danych w pamięci
- Tworzenie systemu plików
- **Operacje**
- Zarządzanie przestrzenią dyskową

Operacje na i-węzłach systemu EXT2

- Struktura **ext2_dir_inode_operations** (w pliku fs/ext2/file.c)

VFS inode operation	Ext2 directory inode method
create	ext2_create( )
lookup	ext2_lookup( )
link	ext2_link( )
unlink	ext2_unlink( )
symlink	ext2_symlink( )
mkdir	ext2_mkdir( )
rmdir	ext2_rmdir( )
mknod	ext2_mknod( )
rename	ext2_rename( )

Operacje na plikach systemu EXT2

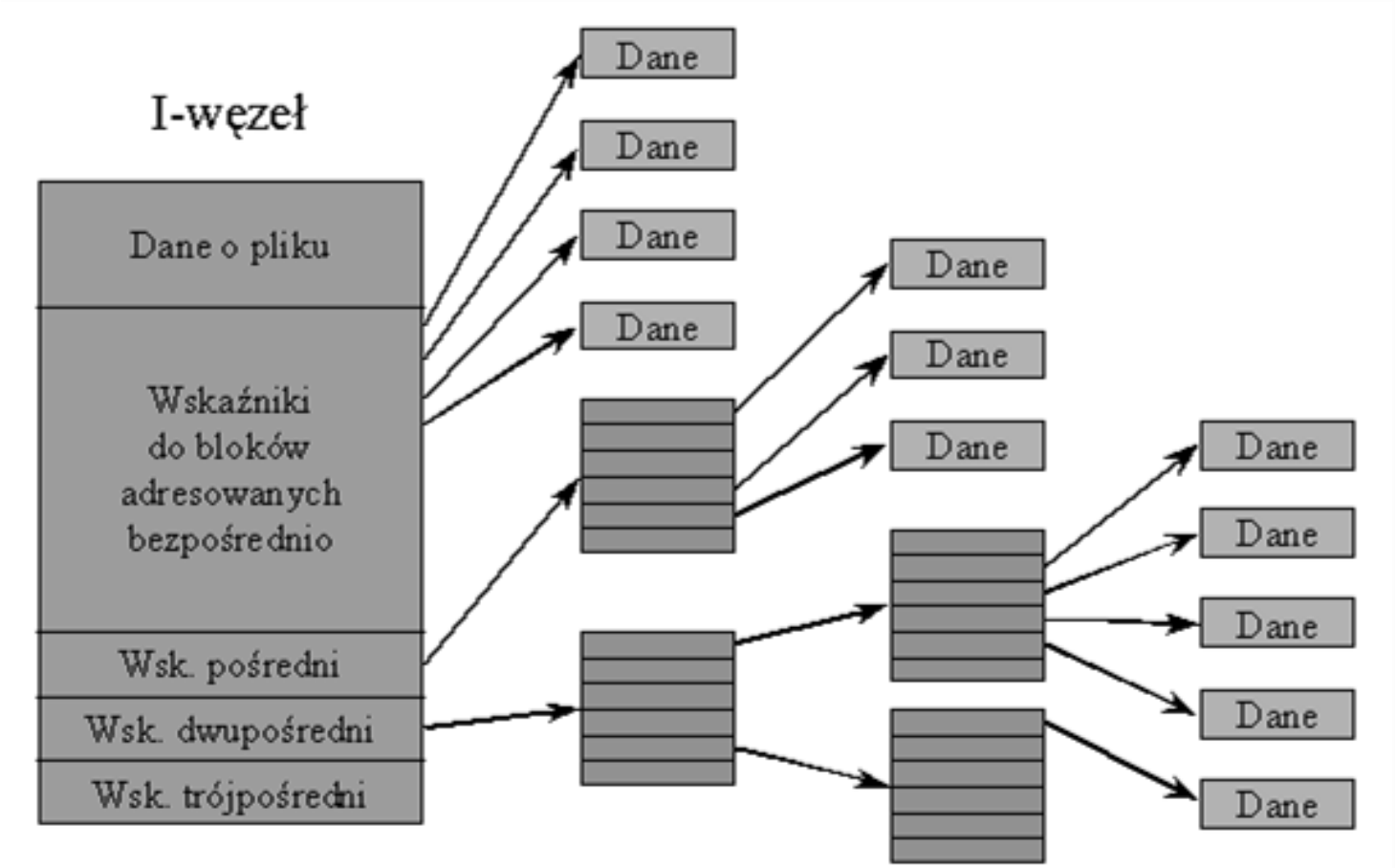
- Struktura **ext2_file_operations** (w pliku fs/ext2/file.c)

VFS file operation	Ext2 method
llseek	generic_file_llseek( )
read	generic_file_read( )
write	generic_file_write( )
ioctl	ext2_ioctl( )
mmap	generic_file_mmap( )
open	generic_file_open( )
release	ext2_release_file( )
fsync	ext2_sync_file( )

Zarządzanie przestrzenią dyskową

- Wprowadzenie
- Struktury danych na dysku
- Struktury danych w pamięci
- Tworzenie systemu plików
- Operacje
- **Zarządzanie przestrzenią dyskową**

Dostęp do system plików EXT2



Adresowanie bloków

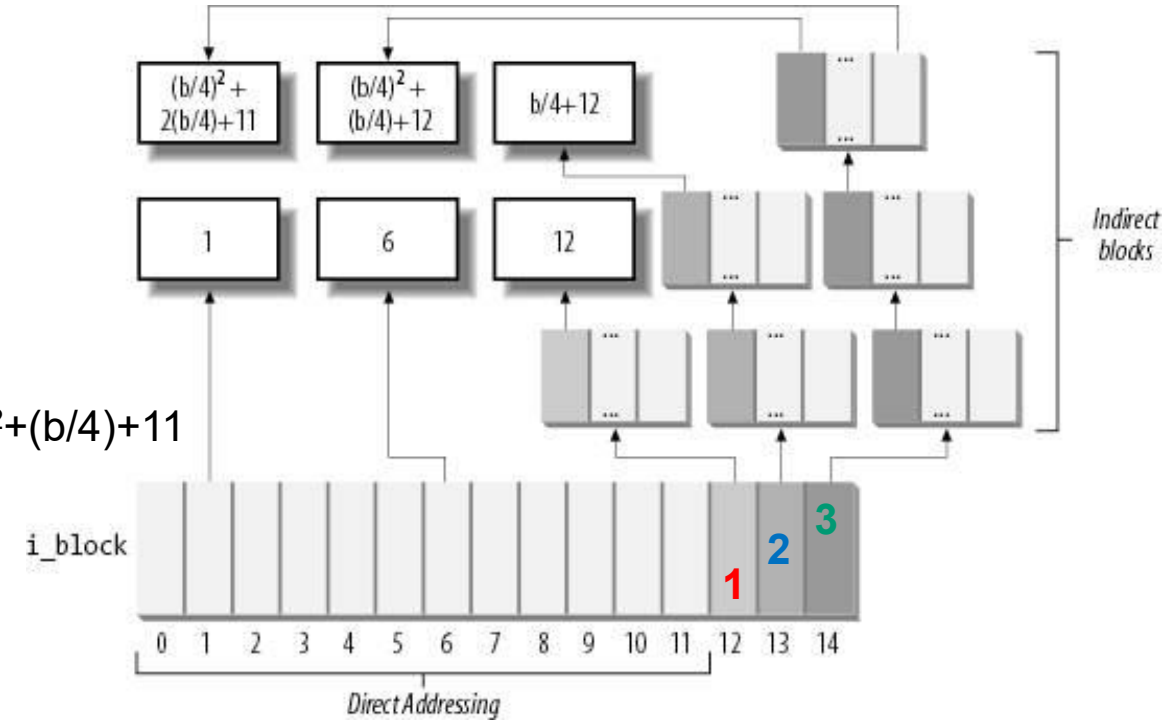
adresy bloków 4 bajtowe

1: 12 ... b/4+11

2: b/4+12 ... $(b/4)^2+(b/4)+11$

3: $(b/4)^2+(b/4)+12$ ... $(b/4)^3+(b/4)^2+(b/4)+11$

b - wielkość bloku w bajtach

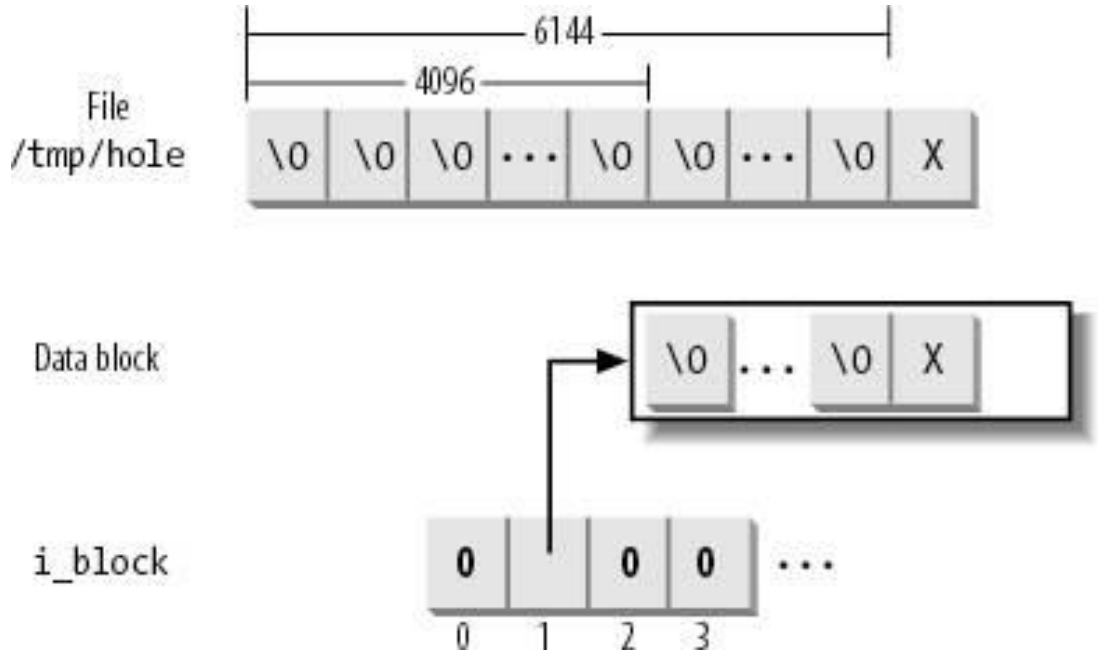


Pliki większe niż 2GB muszą mieć przy otwarciu ustawioną flagę O_LARGEFILE.

Block size	Direct	1-Indirect	2-Indirect	3-Indirect
1,024	12 KB	268 KB	64.26 MB	16.06 GB
2,048	24 KB	1.02 MB	513.02 MB	256.5 GB
4,096	48 KB	4.04 MB	4 GB	~ 4 TB

„Dziury” w plikach

Przykład pliku wypełnionego zerami o rozmiarze 6144 bajtów, zapisanego w jednym bloku danych.



Zarządzanie zasobami

- Alokacja i-węzła: **`ext2_new_inode( )`**
- Zwolnienie i-węzła: **`ext2_free_inode( )`**

- Alokacja bloku: **`ext2_get_block( )`**
- Zwolnienie bloku: **`ext2_free_blocks( )`**

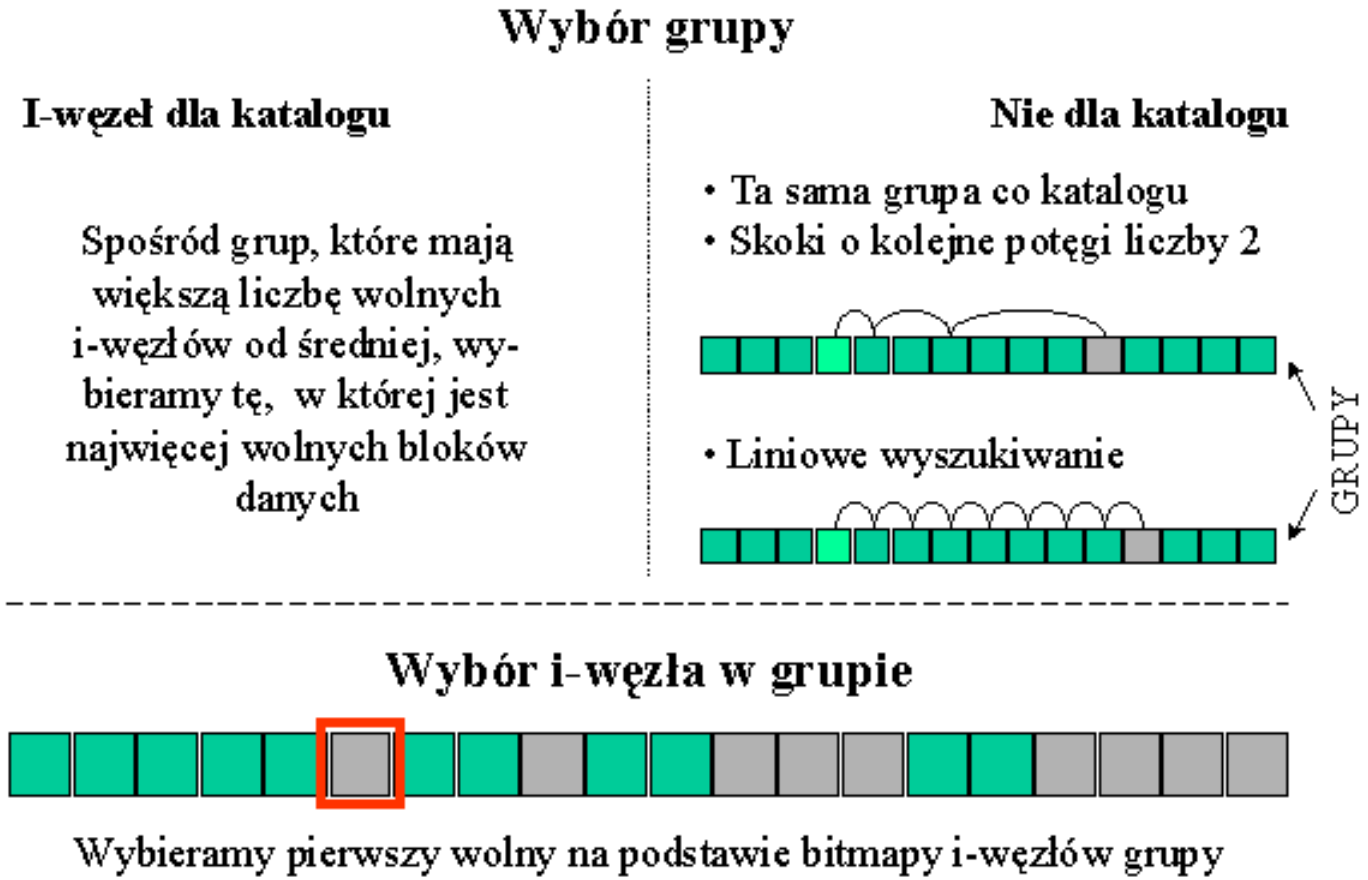
Przydział i-węzła na dysku dla nowego pliku

- Przydziałem i-węzła tworzonemu plikowi zajmuje się funkcja **ext2_new_inode**.
- Algorytm przydziału i-węzła zależy od tego czy przydzielamy i-węzeł **katalogowi**, czy zwykłemu **plikowi**. Jeśli plik **jest katalogiem**, to algorytm przeszukuje wszystkie grupy zawierające liczbę **wolnych i-węzłów** powyżej średniej liczby i-węzłów na całym dysku. Wśród tych grup wybierana jest grupa z największą liczbą wolnych bloków.
- Wymaga to przeszukania wszystkich deskryptorów grup, w których znajdują się liczniki wolnych i-węzłów.
- Jeżeli plik **nie jest katalogiem**, to szukanie wolnego i-węzła zaczynamy od grupy, w której znajduje się katalog tworzonego pliku. Powoduje to, że z większym prawdopodobieństwem pliki leżące w tym samym katalogu będą znajdować się w tej samej grupie bloków.

Przydział i-węzła na dysku dla nowego pliku

- Jeżeli w grupie katalogu **nie istnieją** wolne i-węzły, to przeszukiwanie grup odbywa się przy pomocy algorytmu nazwanego **quadratic hash** (sprawdza się kolejne grupy oddalone o: 1, 2, 4, ..., 2^n < liczba grup (modulo liczba grup).
- Gdy nie da to rezultatu, następuje **liniowe** przeszukiwanie wszystkich grup po kolei począwszy od grupy, w której znajdował się katalog w celu znalezienia grupy z wolnymi i-węzłami.
- Po wybraniu grupy bloków, pozostaje wyszukanie wolnego i-węzła w grupie i przydzielenie go utworzonemu plikowi. Informacja, który i-węzeł w grupie bloków jest wolny, a który zajęty jest przechowywana w mapach bitowych. W odróżnieniu od deskryptorów grup, mapy bitowe zajętości i-węzłów nie są wszystkie przechowywane w pamięci. Maksymalną liczbę załadowanych bitmap w pamięci określa stała EXT2_MAX_GROUP_LOADED i jej wartość wynosi 8.

Alokowanie i-węzła na dysku



Źródło: http://rainbow.mimuw.edu.pl/SO/Wyklady-html/07_pliki/8_pliki.html

Ext3 - księgowanie

Księgowanie lub **kronikowanie** (ang. ***journaling***). Przy użyciu księgowania dane nie są od razu zapisywane na dysk, tylko zapisywane w dzienniku/kronice (ang. *journal*). Dzięki takiemu mechanizmowi działania zmniejsza się prawdopodobieństwo utraty danych.

System **Ext3** umożliwia wybór jednego z trzech trybów księgowania:

- **Journal** - najbezpieczniejszy tryb księgowania, zapisywane są zarówno metadane jak i zwykłe dane (w **Ext4** rozszerzone o **opóźnioną alokację**);
- **Ordered** - tryb domyślny, księgowane są tylko metadane;
- **Writeback** - tryb, w którym księgowane są również tylko metadane, ale jest mniej bezpieczny, bo pozwala na modyfikacje danych objętych metadanymi nie zapisanymi jeszcze na dysk.

e2fsck rozpoznaje dwa rodzaje uszkodzeń:

- uszkodzenie nastąpiło **przed** potwierdzeniem zapisu do kroniki (e2fsck ignoruje zmiany, nowe dane są utracone ale nie został uszkodzony istniejący system plików)
- uszkodzenie nastąpiło **po** potwierdzeniu zapisu do kroniki (e2fsck zapisuje dane z kroniki do systemu plików)

Ext4 – ‘extent’ (1/4)

- Indeksowanie pośrednie w Ext2 i Ext3 (inaczej pośrednie mapowanie bloków) jest bardzo nieefektywne dla dużych plików
  - dodatkowy blok wyszukiwany i odczytywany na każde 1024 bloki
- ‘extent’ jest pojedynczym deskryptorem wskazujący pewien zakres (grupę) kolejno następujących po sobie bloków danych
  - efektywny sposób reprezentacji dużych plików
  - lepsze wykorzystanie CPU, mniej operacji IO dla metadanych

Adres logiczny	Długość	Adres fizyczny
0	1000	200

Mingming Cao, Suparna Bhattacharya, Ted Tso **Ext4: The Next Generation of Ext2/3 Filesystem**

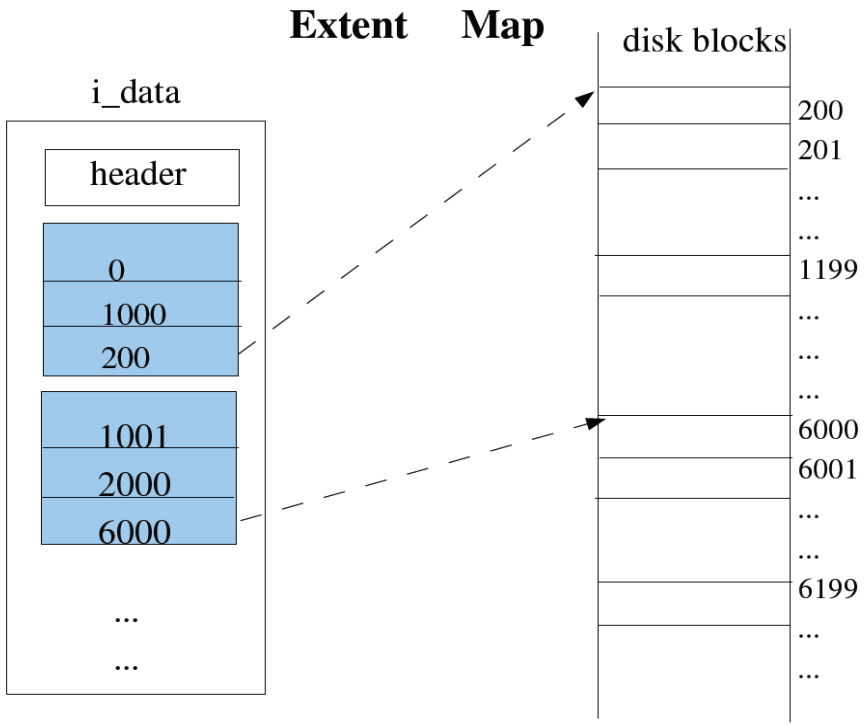
Ext4 – ‘extent’ (2/4)

Na dysku (w i-węźle) mamy 12 bajtową strukturę ext4_extent

- adresowanie bloków fizycznych (rozmiar systemu plików) do 1EB (48 bitowy numer bloku fizycznego)
- maksymalny rozmiar ‘extentu’ 128MB
- Adresowanie pliku o rozmiarze do 16TB (32 bitowy numer bloku logicznego)

```
struct ext4_extent {
__le32 ee_block; /* first logical block extent covers */
__le16 ee_len; /* number of blocks covered by extent */
__le16 ee_start_hi; /* high 16 bits of physical block */
__le32 ee_start; /* low 32 bits of physical block */
};
```

Ext4 – ‘extent’ (3/4)



- do **3** ‘extentów’ przechowywanych **bezpośrednio** w i-węźle (i_data body)
- odpowiednia flaga w i-węźle określa czy używane są ‘extenty’ czy indeksowanie pośrednie typu Ext3
- jeżeli więcej niż 3 ‘extenty’ to adresowanie pośrednie za pomocą B-drzewa

Ext4 – ‘extent’ (4/4)