

Реализация на файловата система

проф. д-р инж. Христо Вълчанов

<http://cs.tu-varna.bg>

Заемане на дисково пространство

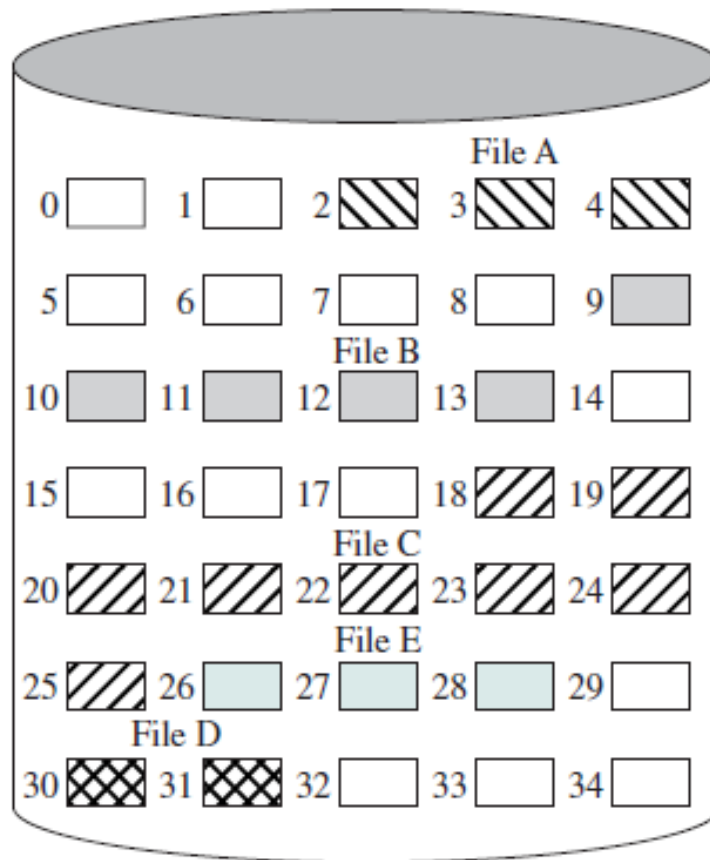
- Диск – устройство с директен достъп.
- Основен проблем – как да се заема дисково пространство за да може дискът да се използва ефективно и файловете да се достъпват бързо.
- Основни методи за заемане на дисково пространство:
 - Непрекъснато заемане
 - Свързано заемане
 - Индексирано заемане

Непрекъснато заемане

Всеки файл заема множество от непрекъснати блокове

- Добра производителност в повечето случаи
- Проста реализация – изисква се само начално местоположение (block #) и дължина (number of blocks)
- Проблеми:
 - Намиране на пространство за файла,
 - Колко е дължината на файла (max size),
 - Външно фрагментиране,
 - Необходимост от дефрагментиране
 - ▶ Off-line
 - ▶ On-line

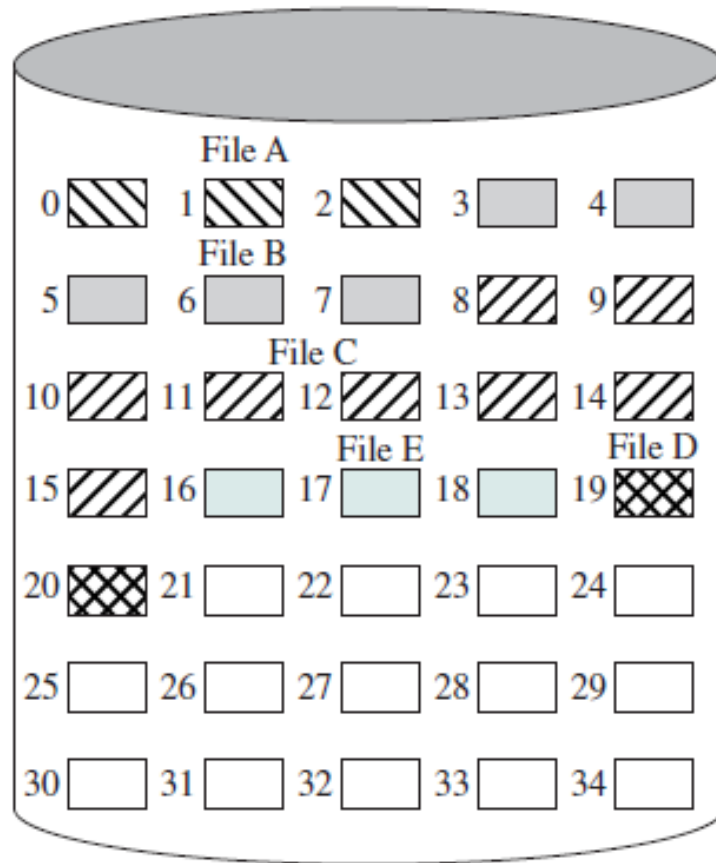
Непрекъснато заемане (2)



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Дефрагментиране



File allocation table

File name	Start block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Системи, базирани на extent

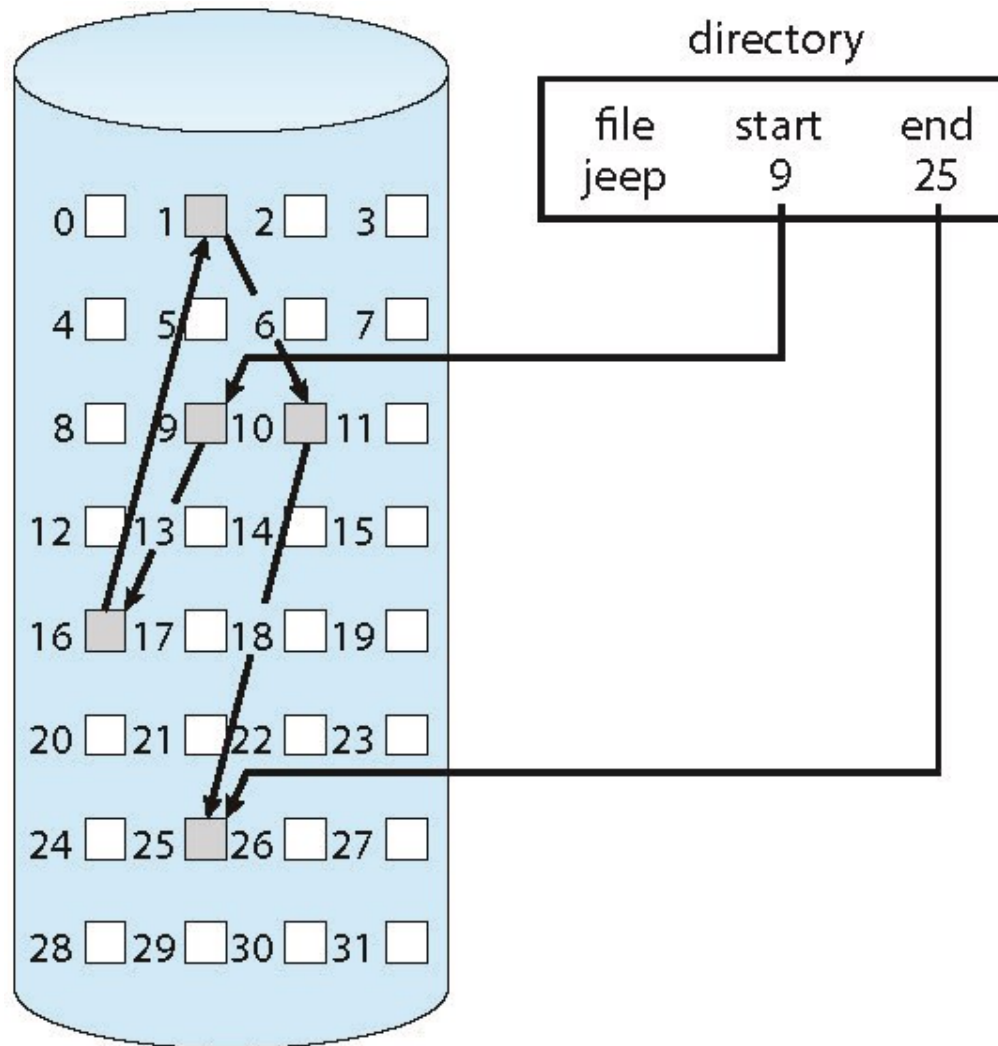
- Модифицирана схема с непрекъснато заемане.
- Заемат се блокове на части - extent.
- **Extent** – верига от блокове
 - Extents се заемат за файловете
 - Файлът може да включва няколко extents.

Свързано заемане

Всеки файл е свързан списък от блокове

- Всеки блок съдържа указател към следващ
- Няма необходимост от дефрагментиране
- Няма външна фрагментация

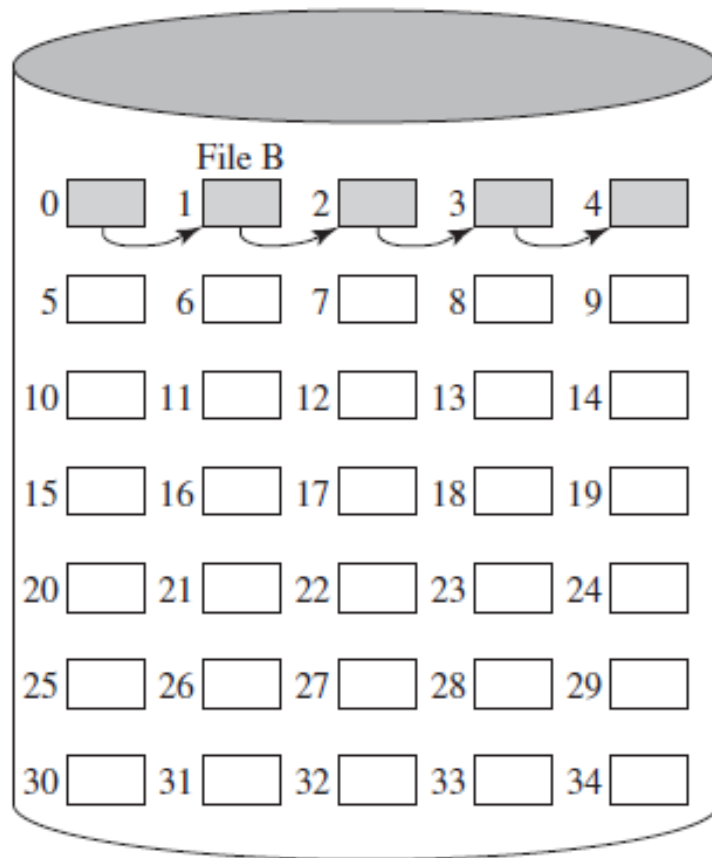
Свързано заемане (2)



Свързано заемане - проблеми

- Директен достъп
- Използва се пространство за указателите
- Клъстеризиране на блокове – повишаване на ефикасността
- Локализиране на блок може да изисква няколко I/O операции и дискови премествания (директен достъп)
- Проблем с надеждността (указателите)

Клъстеризиране

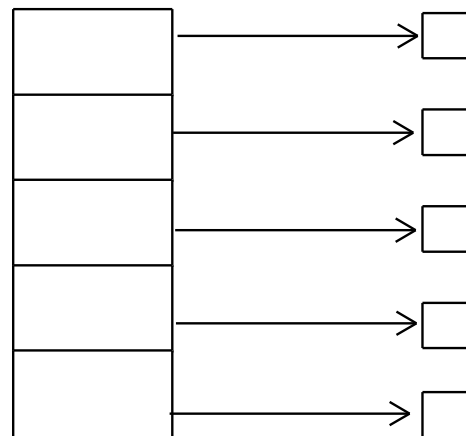


File allocation table

File name	Start block	Length
• • • File B • • •	• • • 0 • • •	• • • 5 • • •

Индексирано заемане

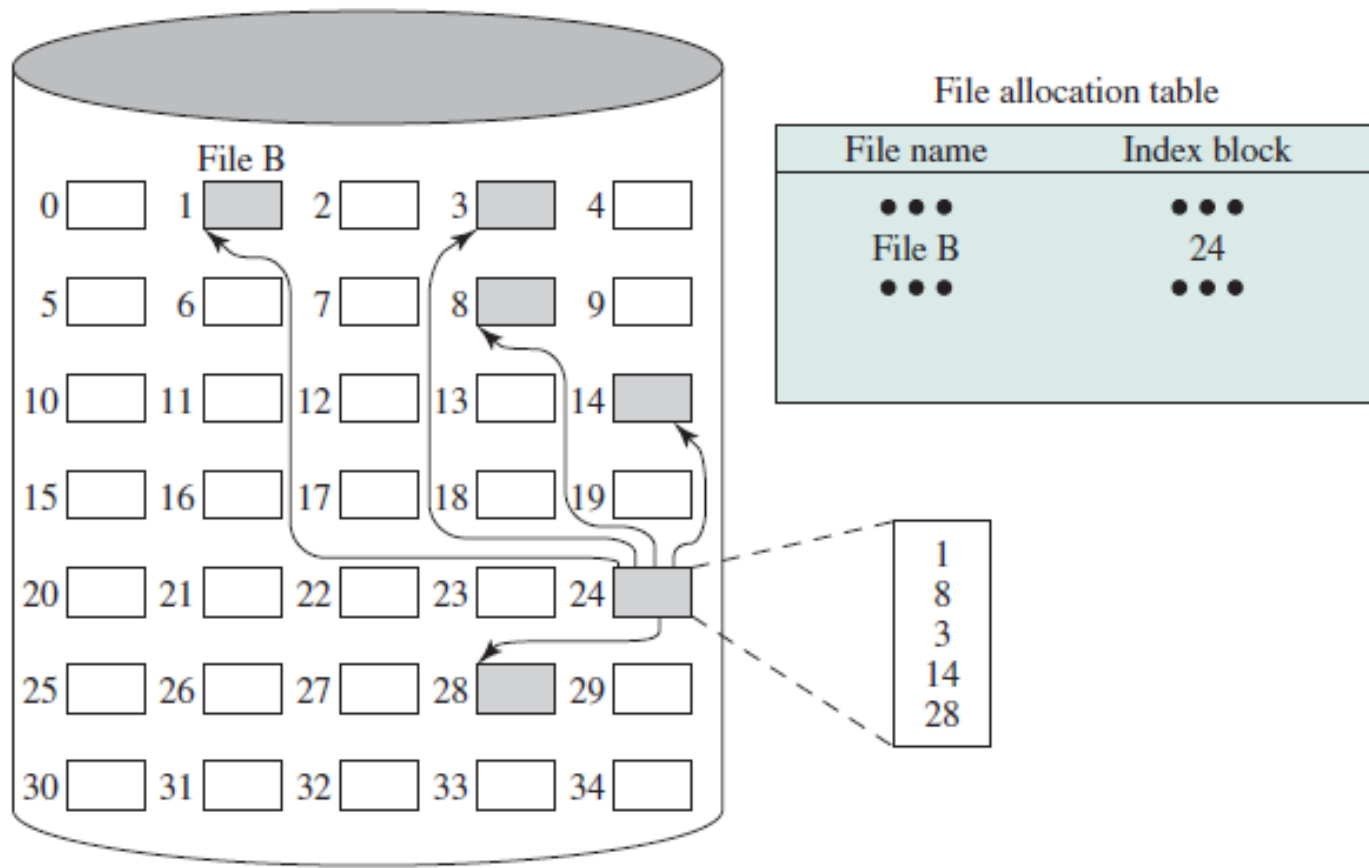
- Всеки файл има собствен **index block** от указатели към неговите блокове
- Логическо представяне



index table

Индексирано заемане (2)

С блокове с фиксиран размер

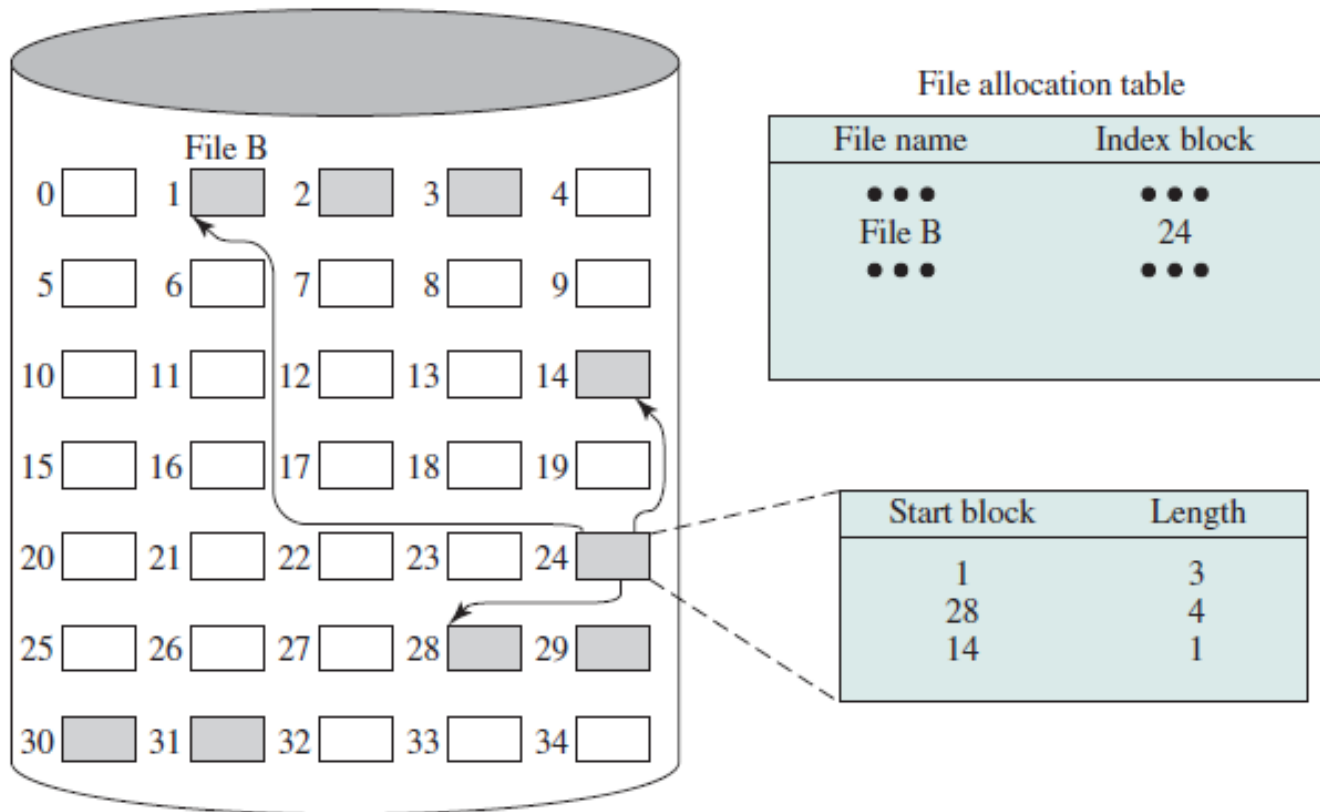


Индексирано заемане (3)

- Поддържане на индексна таблица
- Директен достъп
- Без външна фрагментация, но допуска загуба на пространство

Индексирано заемане (4)

С блокове с променлив размер



Индексирано заемане – големи файлове

- Какъв да е размерът на блок?
- Три основни метода:
 - Свързана схема
 - Индекс на много нива
 - Комбинирана схема

Свързана схема

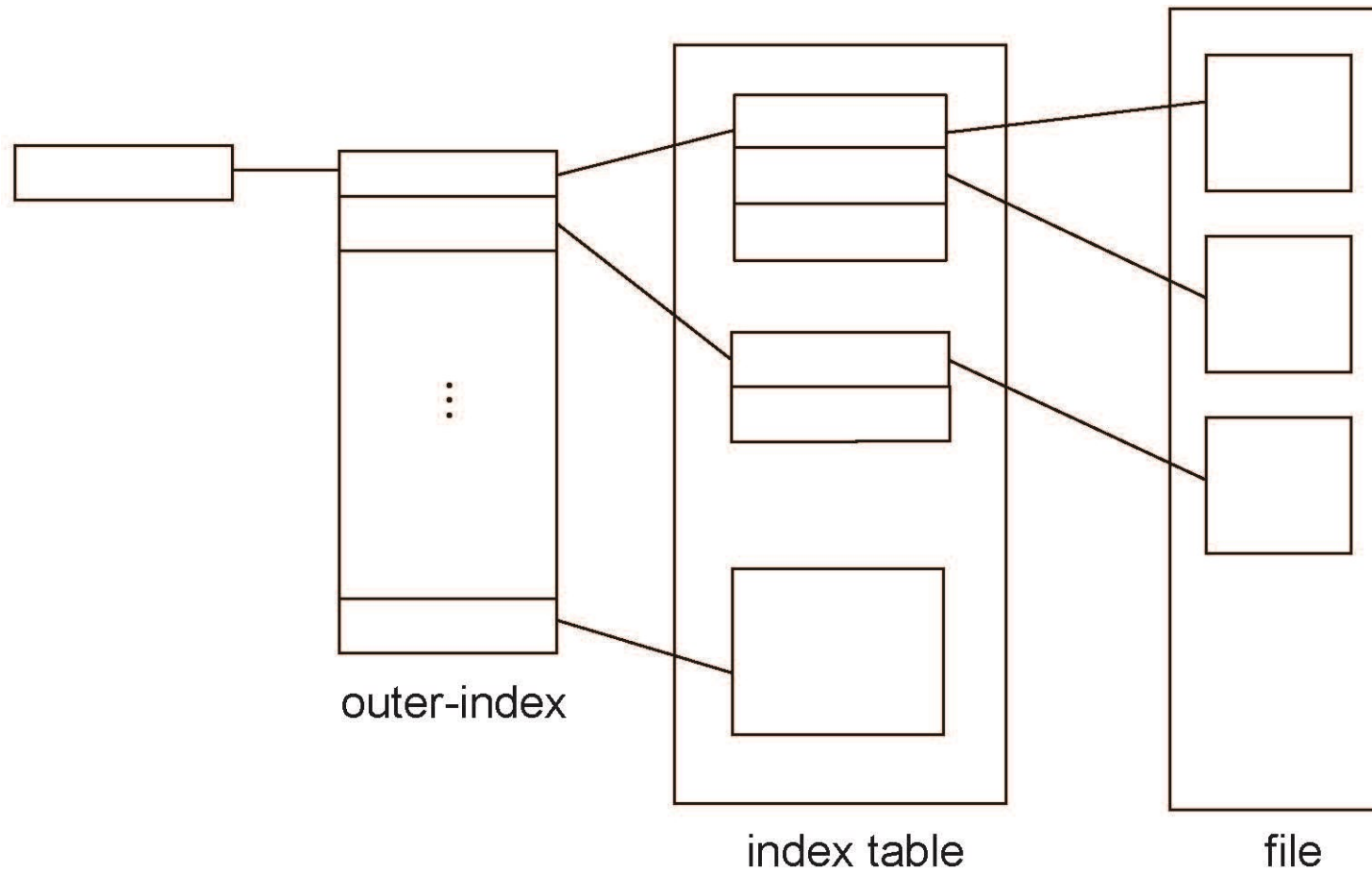
- Свързват се блокове на индексната таблица

Индекс на много нива

Индексен блок от 1-во ниво сочи към индексни блокове от 2-ро ниво. Те сочат към даннови блокове.

Пример: Индекс на 2 нива (при 4K блокове, в индексен блок се съхраняват 1024 на брой 4-байтови указатели → 1 048 567 даннови блокове и размер на файла до 4GB)

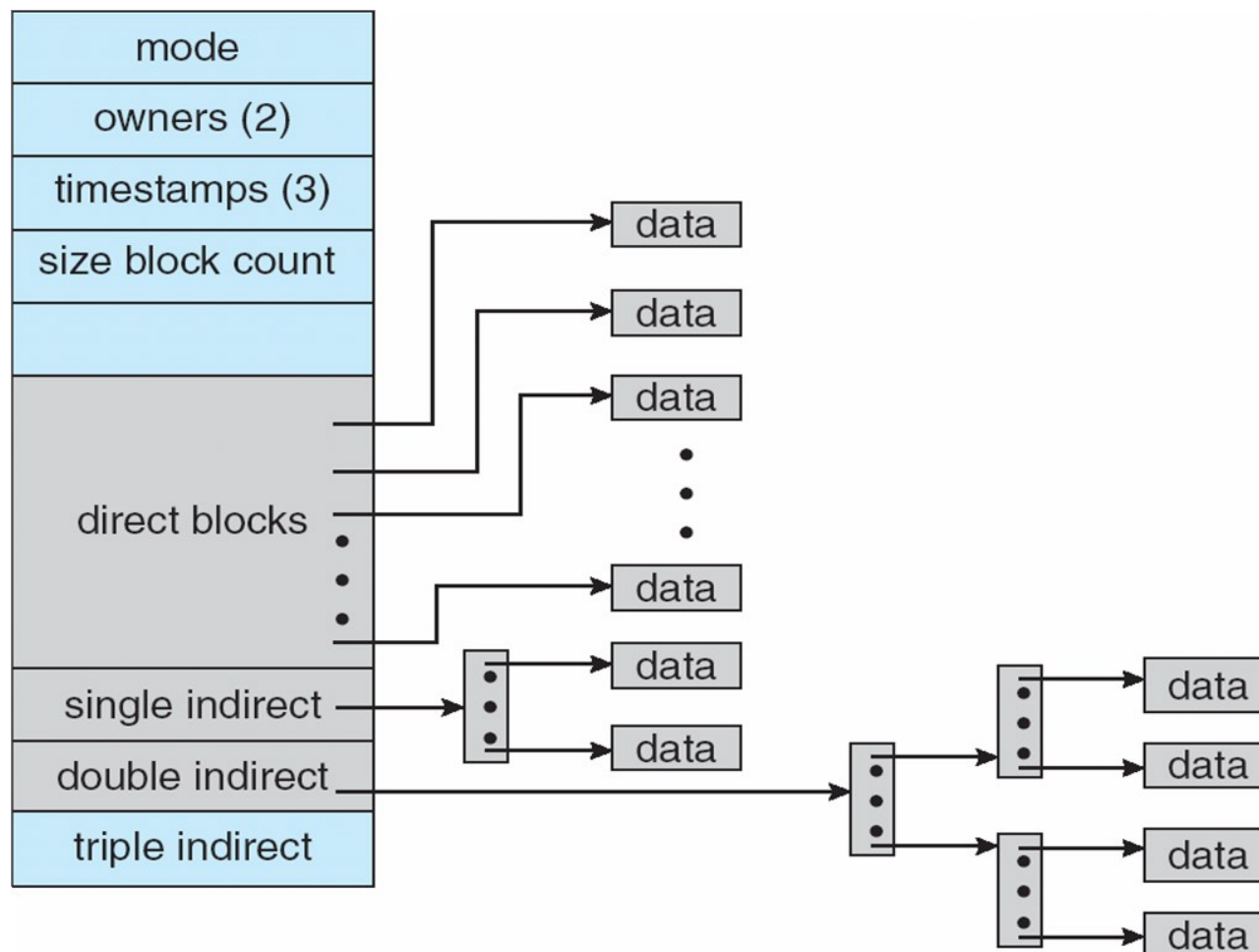
Индекс на много нива (2)



Комбинирана схема - UNIX UFS

- 4K размер на блок, 32-bit адреси
- Първите 15 указателя от индексната таблица са съхраняват в *inode* на файла
- От тях първите 12 сочат директно към даннови блокове. За малки файлове (не повече от 12 блока) няма нужда от отделен индексен блок. Ако размерът на блок е 4K, то директно могат да бъдат достъпни до 48K данни.
- Оставащите 3 указателя:
 - Първият сочи към индиректен индексен блок съдържащ адреси на даннови блокове.
 - Вторият сочи към двоен индиректен блок, съдържащ адрес на блок, който съдържа адреси на даннови блокове.
 - Третият съдържа адрес на троен индиректен индексен блок.

Пример: UNIX UFS



Производителност

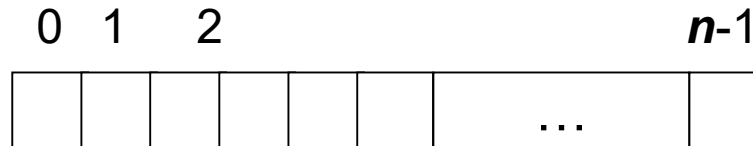
- Кой метод ще се използва зависи от вида на достъп
 - Непрекъснатото заемане е ефикасно за последователен и директен достъп
- Свързаното заемане е ефикасно за последователен, но не за директен достъп
- Да се указва вида на заемане при създаване на файл
- Индексираното заемане е сложно
 - Напр. при 2 индексен блок единичен достъп до блок изисква 2 четения на индексен блок и 1 за четене на даннов

Управление на свободното пространство

- Файловата система трябва да поддържа списък на свободното пространство на диска.

Управление на свободното пространство (2)

- **Bit vector (bit map)**



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block}[i] \text{ е свободен} \\ 0 \Rightarrow \text{block}[i] \text{ е зает} \end{cases}$$

Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

➤ CPU имат инструкции да връщат отместването в думата на първия бит с “1”

Управление на свободното пространство (3)

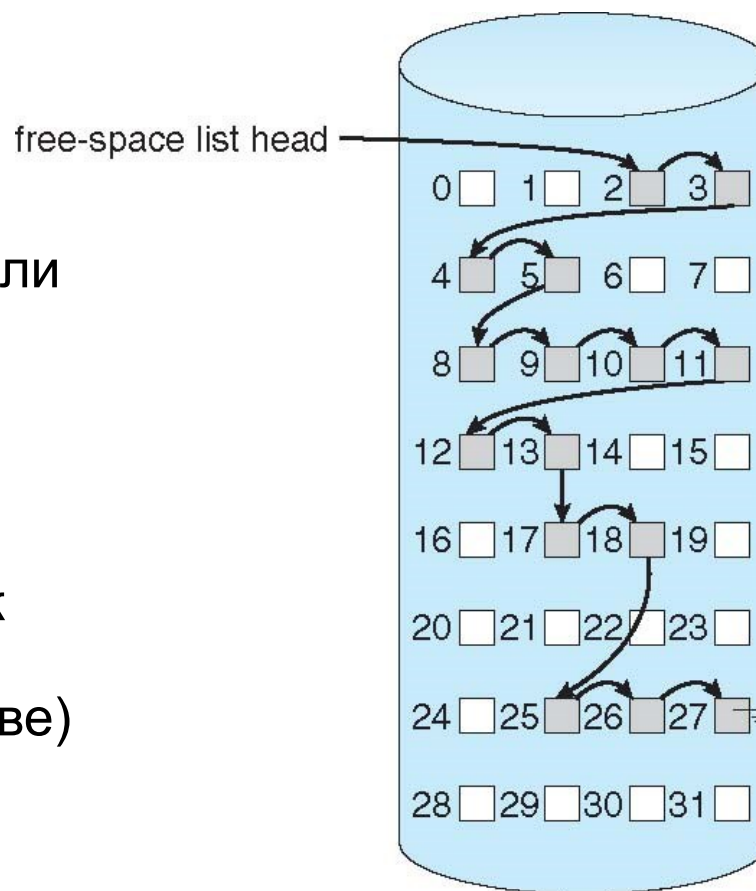
- Bit map изисква допълнително пространство

Пример:

- размер на блок = 4KB = 2^{12} bytes
 - размер на диск = 2^{40} bytes (1 terabyte)
 - $n = 2^{40}/2^{12} = 2^{28}$ bits (or 32MB)
 - при клъстер от 4 блока -> 8MB памет
-
- Удобно за непрекъснати файлове

Списък за свободното пространство на диска

- Свързан списък
 - Не може да се определи лесно непрекъснатото пространство
 - Няма фрагментиране
 - Няма нужда да се обхожда целия списък (при необходимост се вземат първите блокове)



Списък за свободното пространство на диска (2)

- Групиране
 - Свързан списък, съдържащ в първия свободен блок адресите на следващите $n-1$ блока плюс указател към следващ блок, съдържащ аналогично адресите на свободните блокове
- Броене
 - Поради честото използване и освобождаване на непрекъснатото дисково пространство:
 - Съхранява се адреса на първият свободен блок и броя на следващите свободни блокове

Възстановяване след отказ

- **Consistency checking** – сравняват се данните в директорната структура с данните блокове на диска. Фиксиране на нееднородността
 - Бавно
- Използване на програми за архивиране на данните на друго устройство за съхранение
- Възстановяване на загубени данни от архива

Въпроси?