Chapter 12: File System Implementation

- Virtual File Systems.
- Allocation Methods.
- Folder Implementation.
- Free-Space Management.
- Directory Block Placement.
- Recovery.

Virtual File Systems

- An object-oriented way to support multiple file system types:
  - VFS defines API: in pseudo-Java:

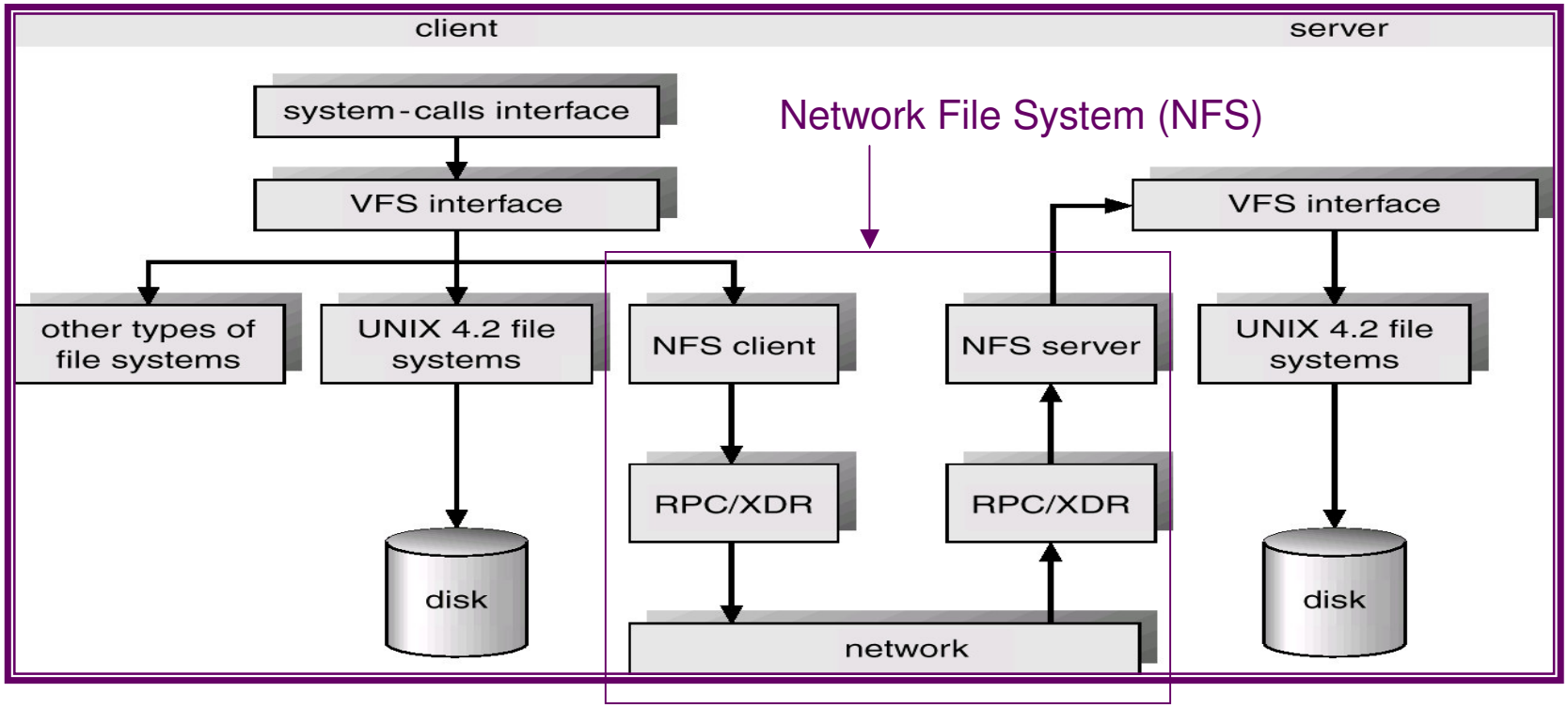
```
public interface VFS {
 void write(File f, Buffer b); }.
```

  - Applications use file system only via this API:

```
myFS.write(myFile,myBuffer);
```

  - API implemented by filesystem drivers:

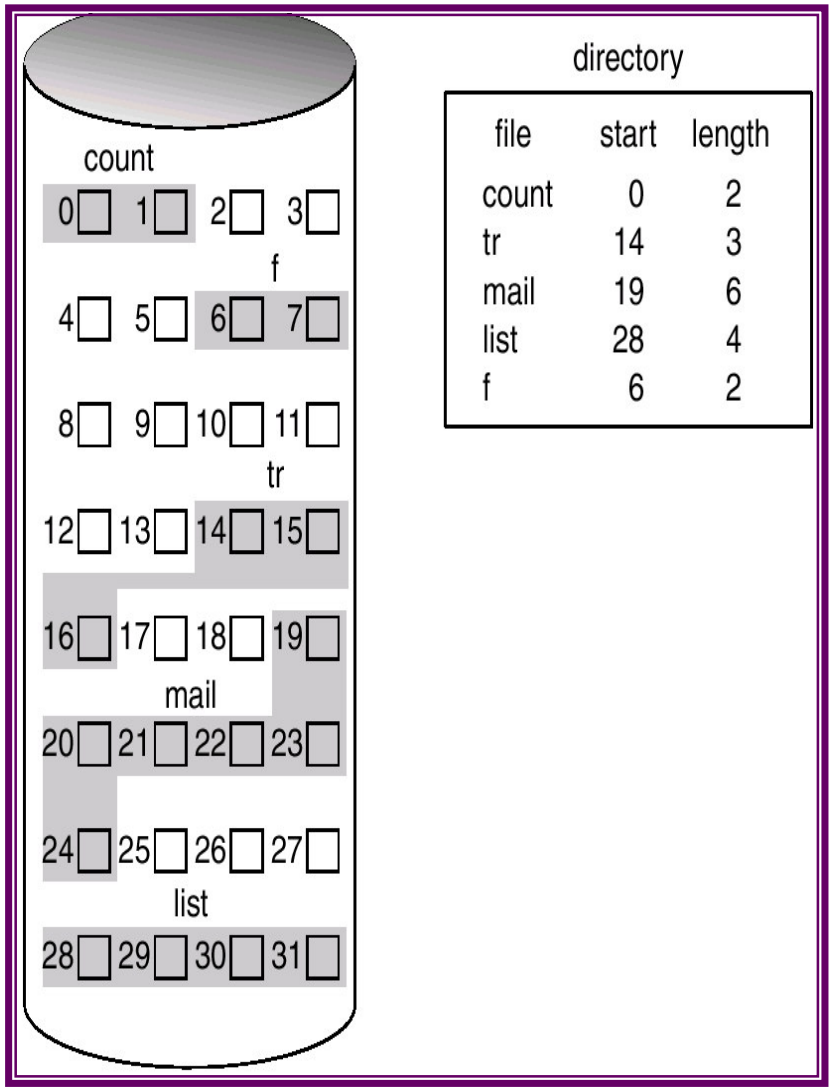
```
class NFS implements VFS { ... }.
```



Allocation Methods

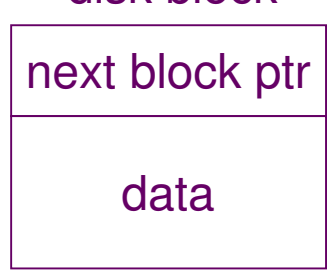
- How are disk blocks allocated for file content? Contiguous, linked, indexed allocation.

- Contiguous:
  - Each file occupies a set of contiguous blocks on the disk.
  - Simple: only starting block index and length (number of blocks) stored in directory.
  - Random access.
  - Wasteful of space: dynamic storage-allocation problem hence external fragmentation.
  - Files cannot grow unless free blocks happen to exist after its present end.



Linked Allocation

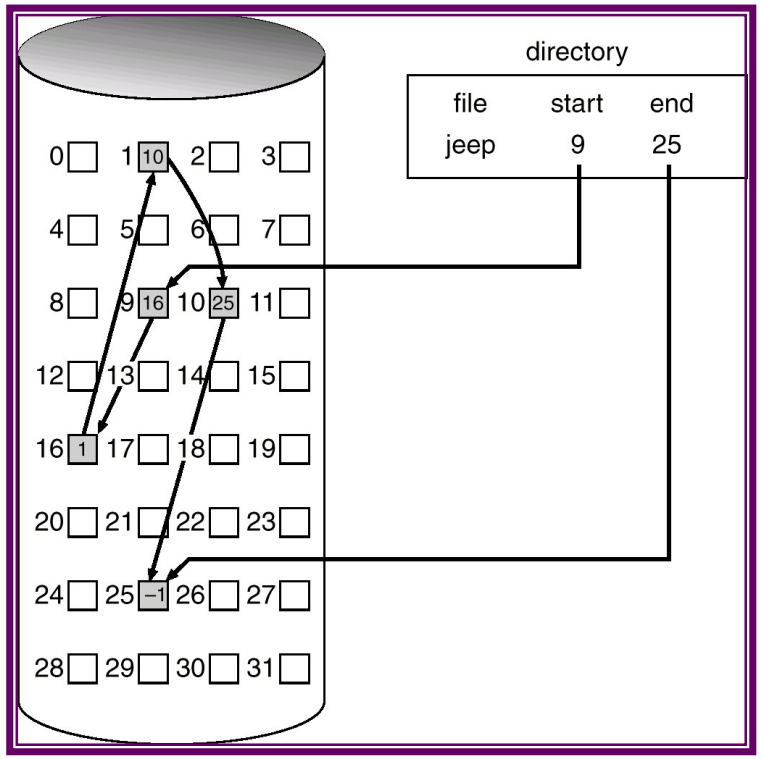
- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



- Simple: store starting block index in file meta-data.
- No random access.
- No waste.
- Growth possible.
- *Defragmentation*: bring data blocks closer together.

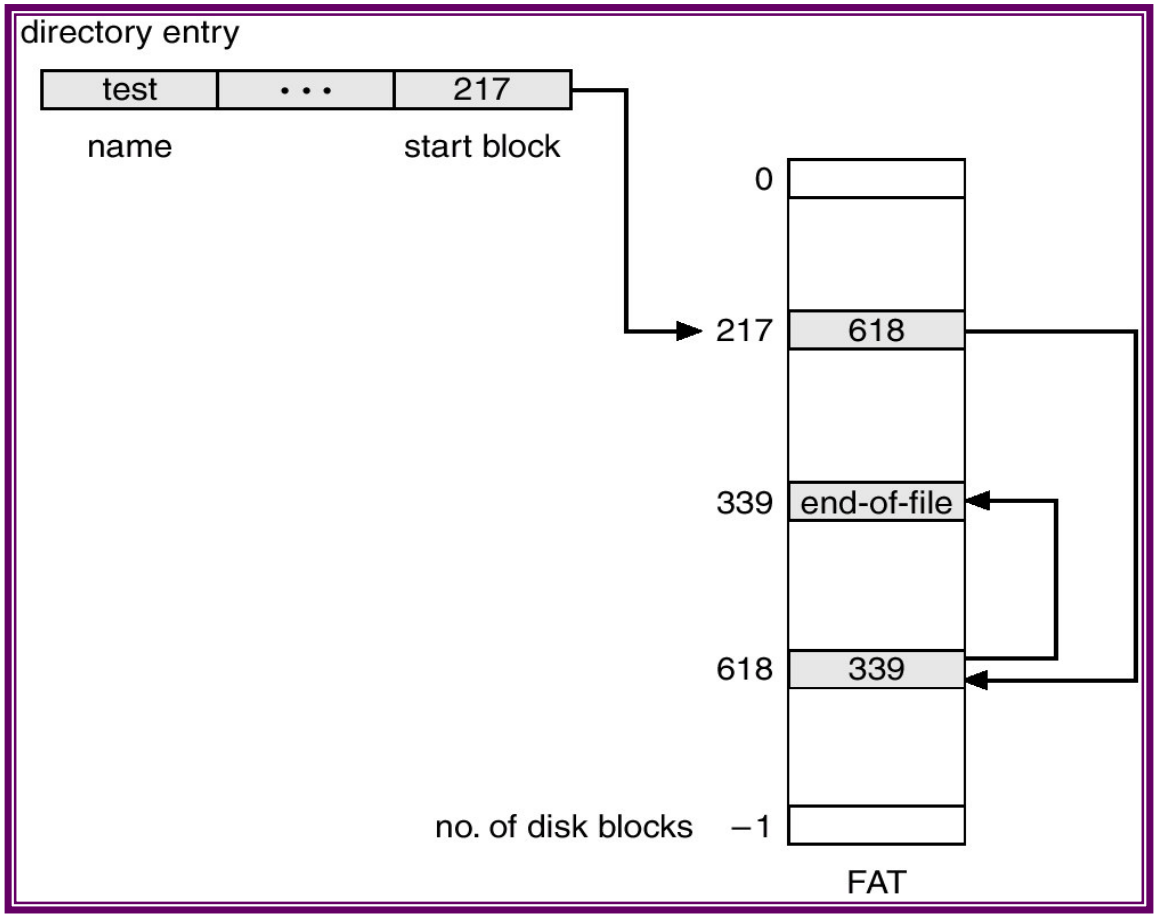
- Variations:
  - FAT (next slide).
  - Extent-based system: like linked list, except that what is linked is contiguous blocks (*extents*).

Veritas



File-Allocation Table

Variation on linked allocation: instead of pointers within blocks, store *all* pointers (from *all* files and blocks) in a single table on disk (FAT). FAT has one entry per disk block, and its contents are the linked list pointers (i.e. block indices).



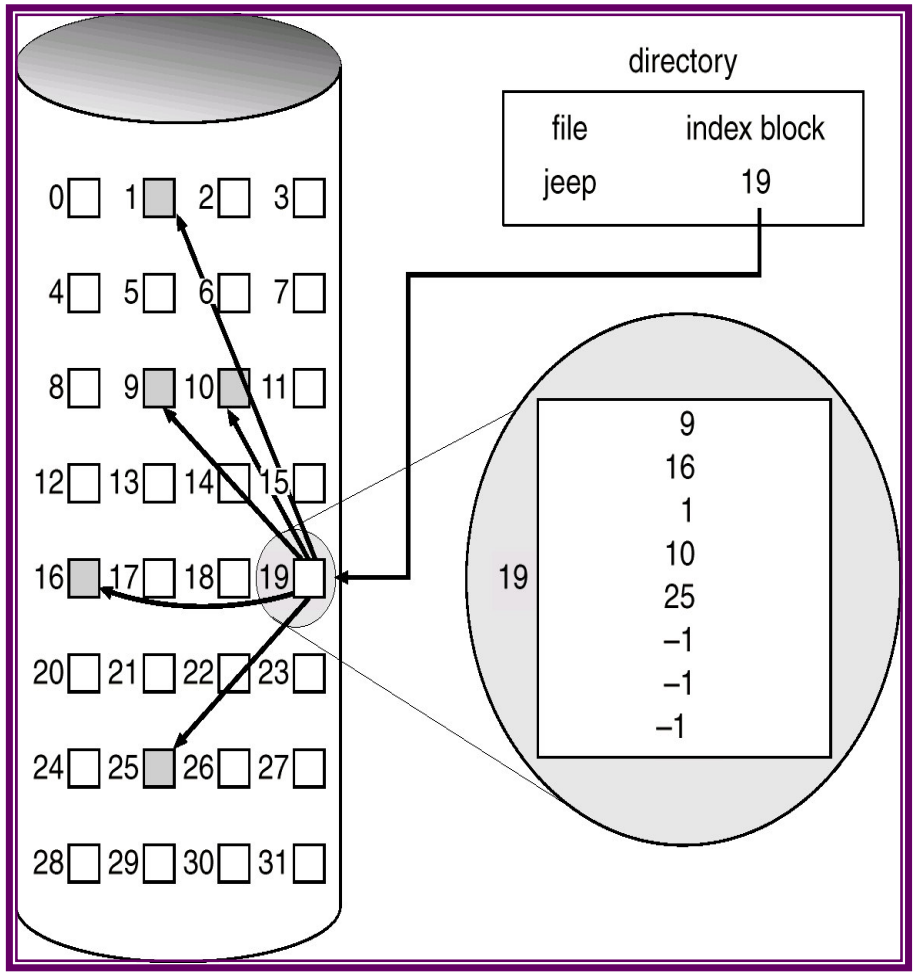
Random access possible because FAT is small and can follow pointers quickly, esp. if FAT is cached in memory.

MS-DOS
OS/2
pre-NT Windows

Indexed Allocation

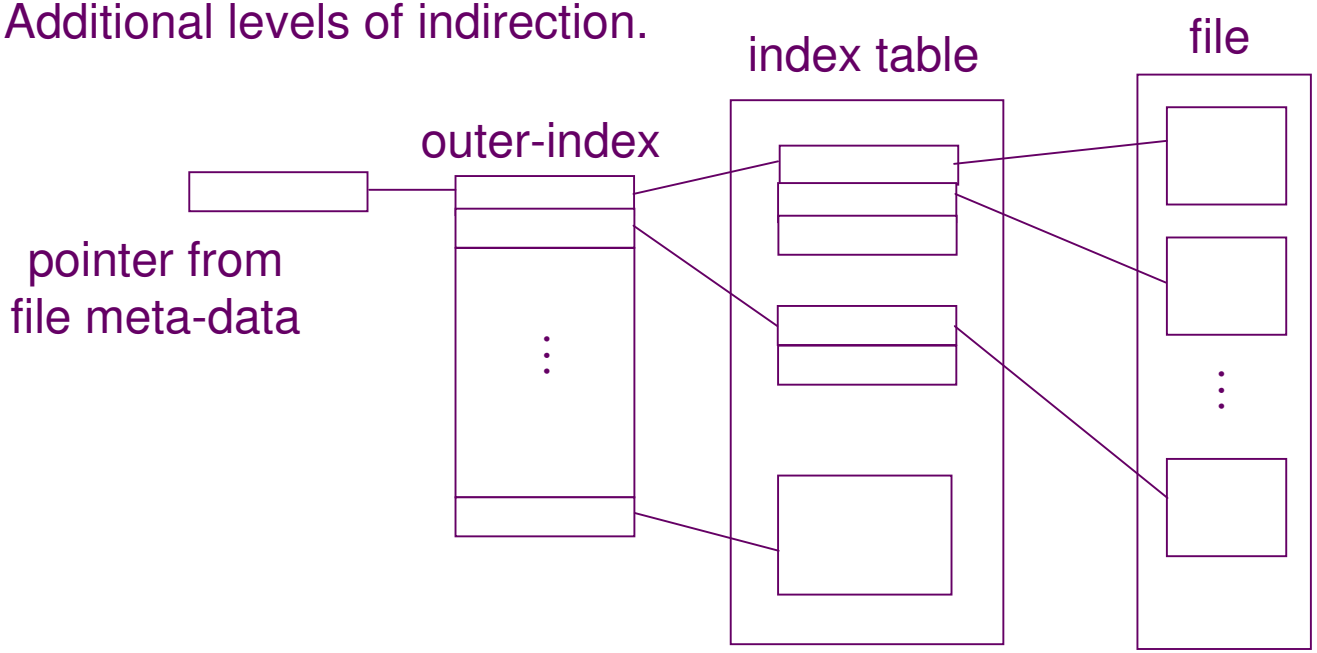
- Store all indices of data blocks into the *index block.*

- Simple: store index of block index in file meta-data.
- Random access.
- No waste, but need index block.
- Growth possible but limited to size of index block.
- Additional level of indirection before we get to file data.



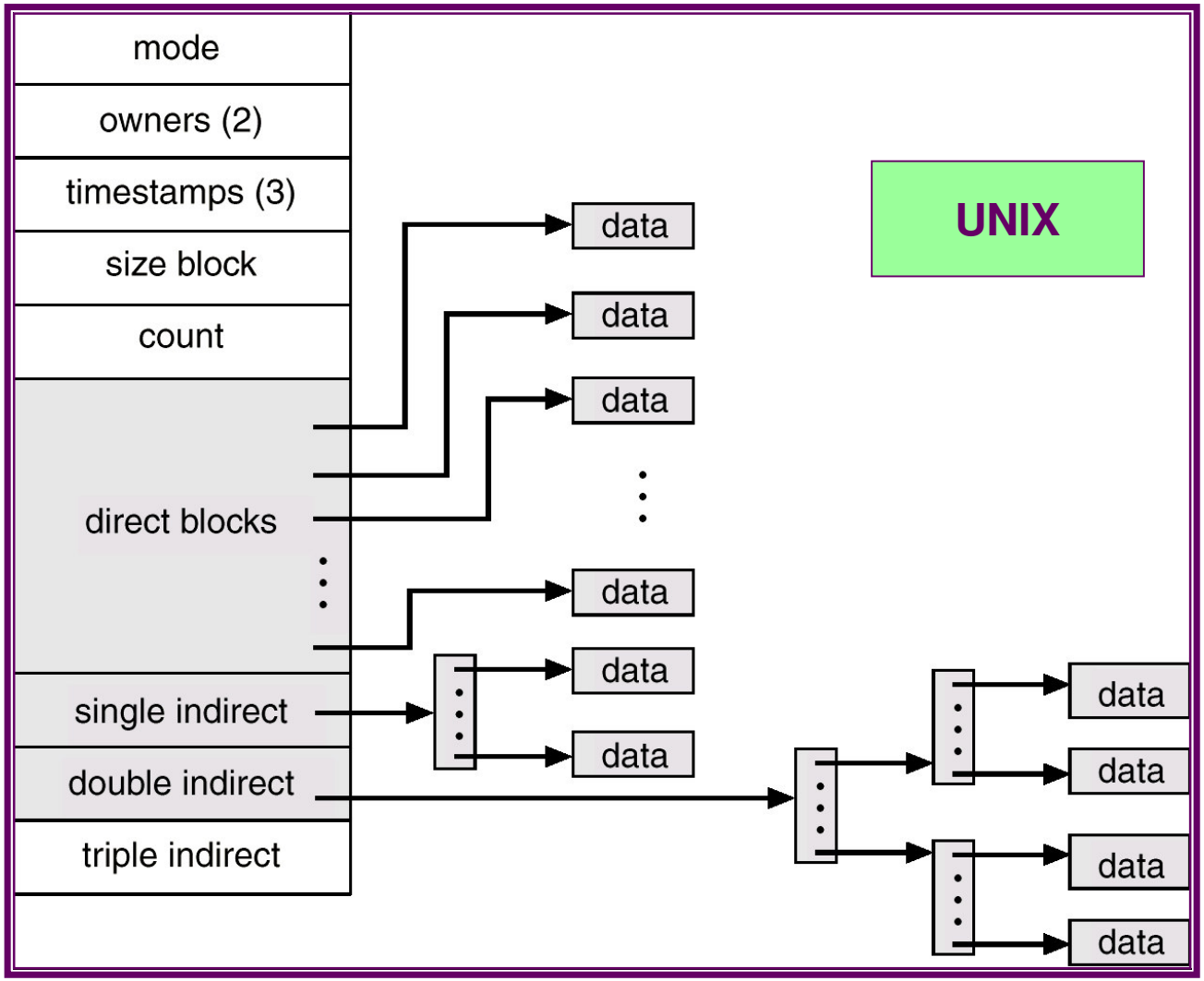
Indexed Allocation (Cont.)

- How can we extend indexed allocation to support large files (growth)?
- Linked scheme: link index blocks to form index *table*:
  - No limit on size.
  - No random access.
- Multi-level index:
  - Large size (still limited).
  - Additional levels of indirection.



- Combined scheme: best of both worlds; next slide.

Combined Scheme

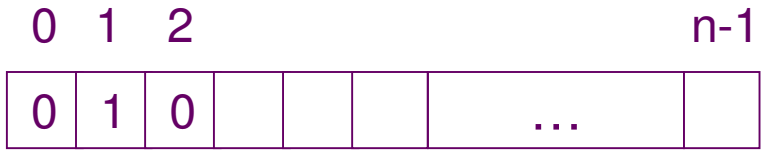


Folder Implementation

- Each folder's contents are stored in one or more disk blocks:
  - Always one block: simple, but fixed number of files/subfolders.
  - Variable: need allocation scheme as for file data.
- Within blocks, folder contents can be organized as:
  - Linear list of file names, each file name pointing to file's data blocks:
    - Simple but requires linear search to find file.
  - List with hash function:
    - Compute hash of file name to get index of list entry.
    - Indexed entry can contain single name (simple but fixed size folder) or linked list of names with pointers to resolve name collisions.

Free-Space Management: Bit Vector

- Bit vector: copied from disk block to memory, and stored in sequential chunks of 32-bit words. Easy to get contiguous space.



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ occupied} \\ 1 \Rightarrow \text{block}[i] \text{ free} \end{cases}$$

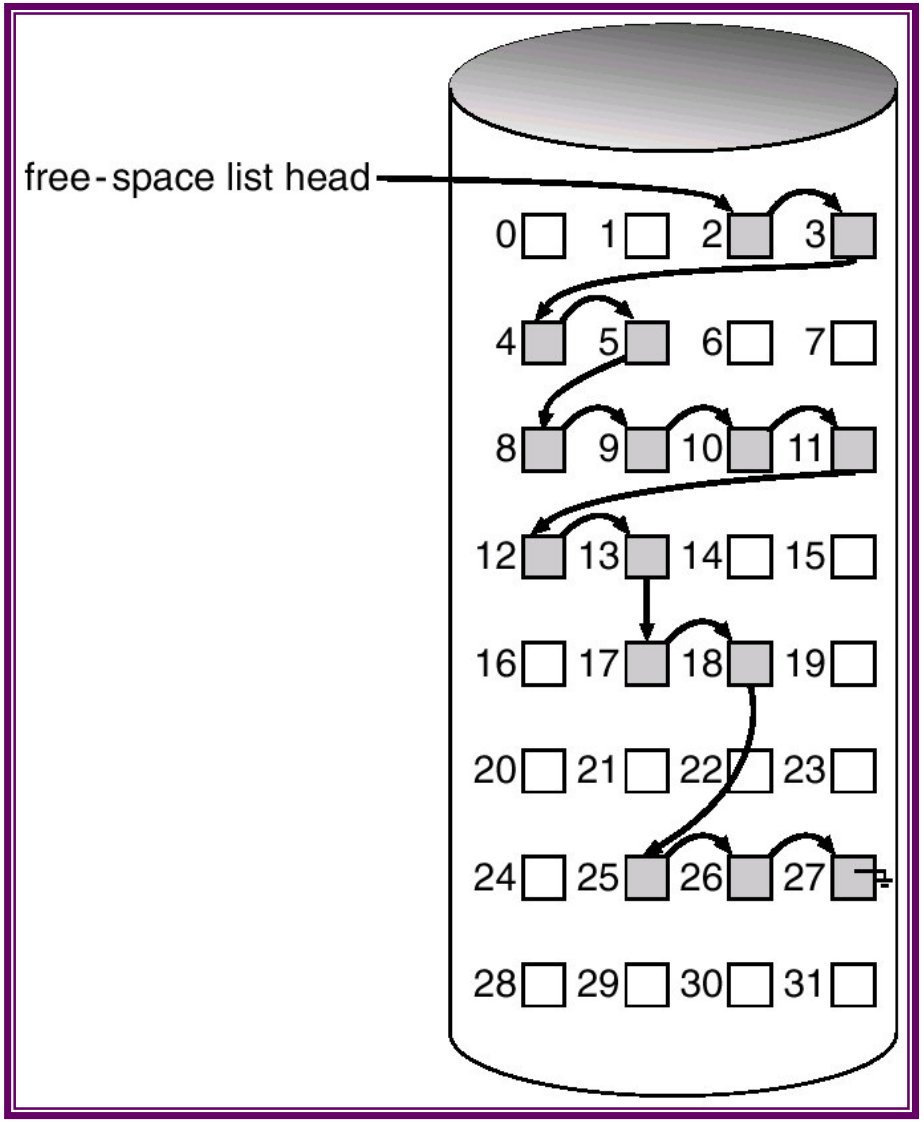
To find free block: scan these words for the first non-zero word, then find the first bit set to 1 in that word. Free block index is then:

32 x (number of words with zero value)
+ (offset of first bit set to 1)

- Bit map space: if block size 4KB (2^{12} bytes), and disk size 256GB (2^8 x 2^{30} bytes), then we need $2^{38}/2^{12} = 2^{26}$ bits = 2^{23} bytes = 8MB.

Free-Space Management: Free List

- No waste for allocated space.
- Basic: each free block points to the next.
  - ☞ Hard to find contiguous space.
- Grouping: some linked “free” blocks set aside to list block indices of other (really) free blocks.
- Counting: free blocks usually contiguous. So store index of first block with count of free ones that follow it in “free” blocks.



Directory Block Placement

- Blocks are used for
  - File data.
  - Directory (file meta-data, incl. folder data).
  - Other: free-space management, partition list, boot block, etc.

- Where should we store the directory?
  - In the first few blocks of the disk, with growth elsewhere as needed:
    - Simple and fast (few seeks if we care about directory data only).
    - Risky: if small platter area is damaged, all meta-data will be lost.
    - Inefficient for short meta-data/data access cycles.
  - In the first few blocks, and copied elsewhere for reliability (see RAID).
  - Unix: pre-allocate blocks for directory scattered throughout disk.

Recovery

- Consistency checking: compare data in directory with data blocks on disk, and try to fix inconsistencies:
  - Example: a block is not marked free in free-list, because OS claimed it for a file right before it crashed. So no file is using the block yet. This is an inconsistency.
  - Windows `chkdsk`: program that does check & fix.
  - Unix `lost+found`: area where lost blocks are placed. Consistency check happens during mounting.
- Log-based recovery: use database-like logging techniques (*transactions*, *journal*) to ensure disk won't be corrupted by system crash.