UNIT V

FILE SYSTEMS

FILE SYSTEMS

Implementing File Systems

- File-System Structure
- File-System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management

Case Study

- Real Time operating systems
- Mobile Operating systems

Directory Implementation

- **Linear list** of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
    - Linear search time
    - Could keep ordered alphabetically via linked list or use B+ tree
- **Hash Table** – linear list with hash data structure
  - Decreases directory search time
  - **Collisions** – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation** – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**

Contiguous Allocation

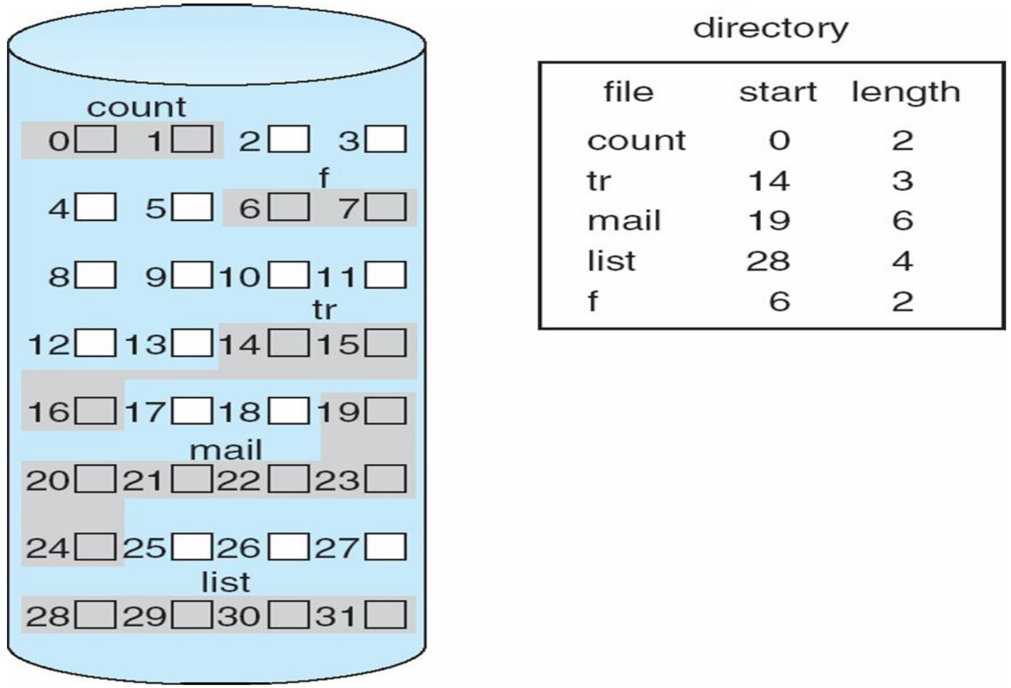
- Mapping from logical to physical

Q

LA/512

R

Block to be accessed = Q + starting address
Displacement into block = R



Extent-Based Systems

- Many newer file systems (i.e., Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in extents
- An **extent** is a contiguous block of disks
  - Extents are allocated for file allocation
  - A file consists of one or more extents

Allocation Methods - Linked

- **Linked allocation** – each file a linked list of blocks
  - File ends at nil pointer
  - No external fragmentation , No compaction
  - Each block contains pointer to next block
  - Free space management system called when new block needed
  - Improve efficiency by clustering blocks into groups but increases internal fragmentation
  - Reliability can be a problem
  - Locating a block can take many I/Os and disk seeks

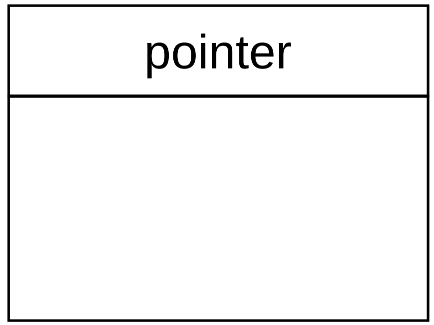
Allocation Methods – Linked (Cont.)

- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

Linked Allocation

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

block =



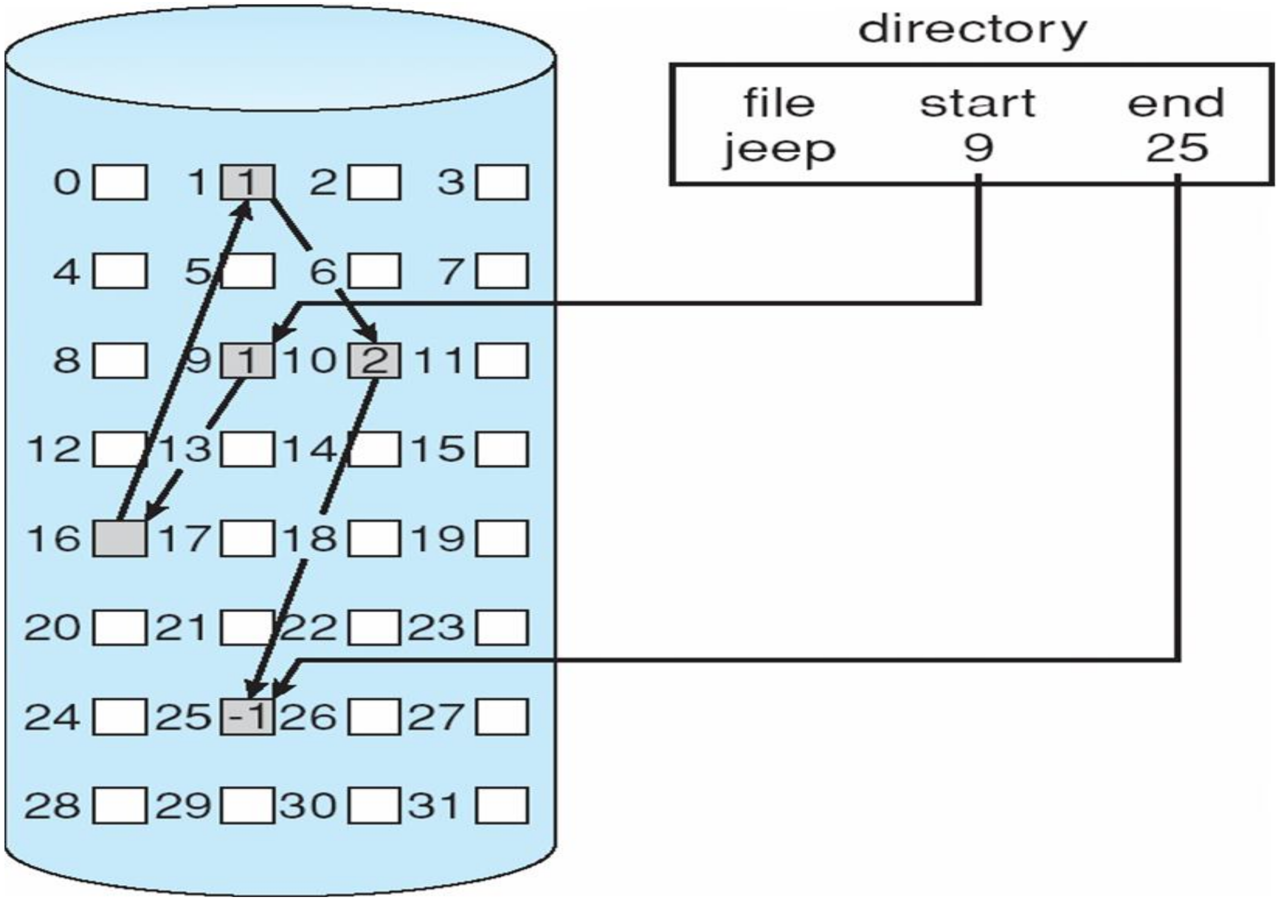
- Mapping

LA/511 → Q, R

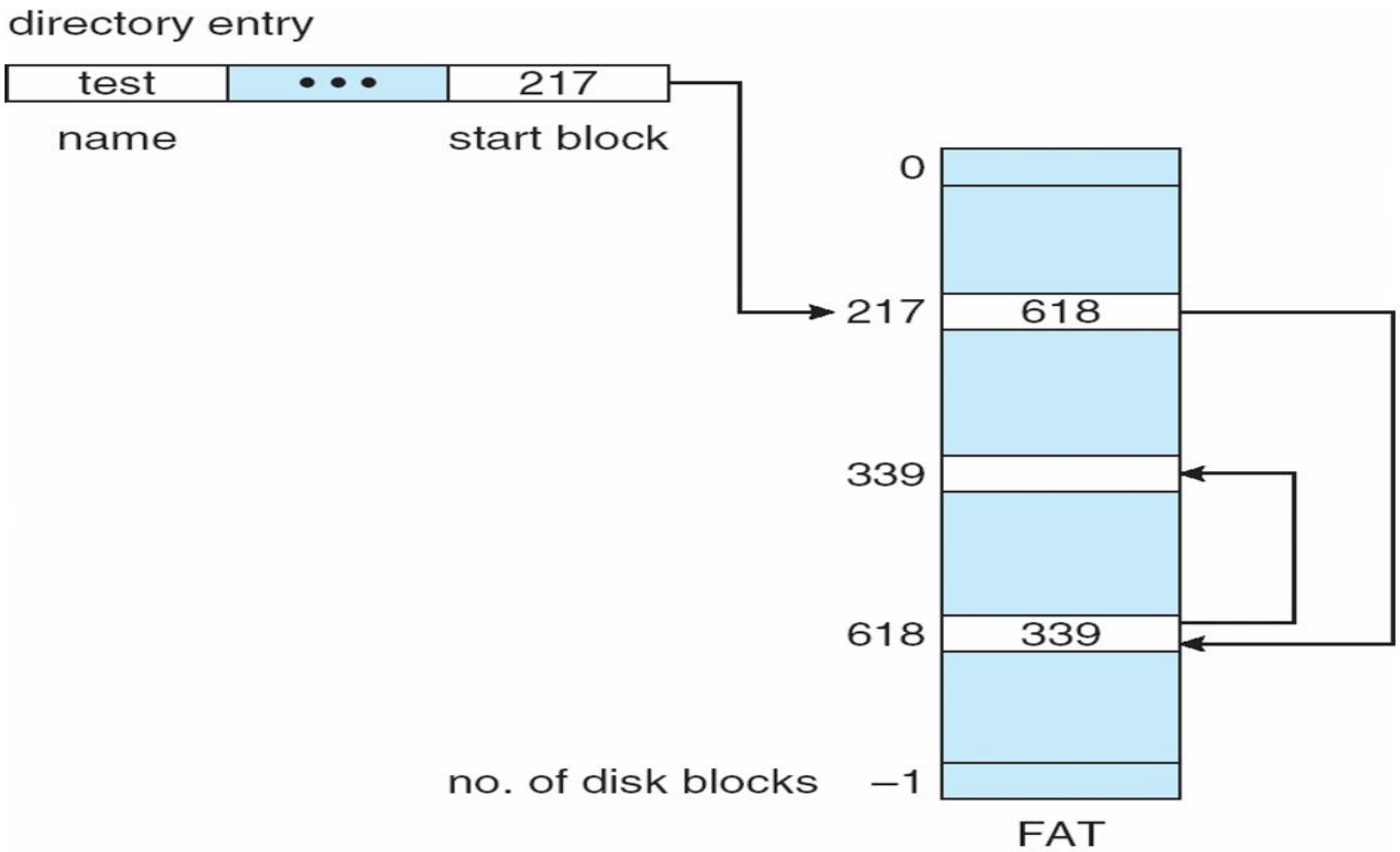
Block to be accessed is the Qth block in the linked chain of blocks representing the file.

Displacement into block = R + 1

Linked Allocation



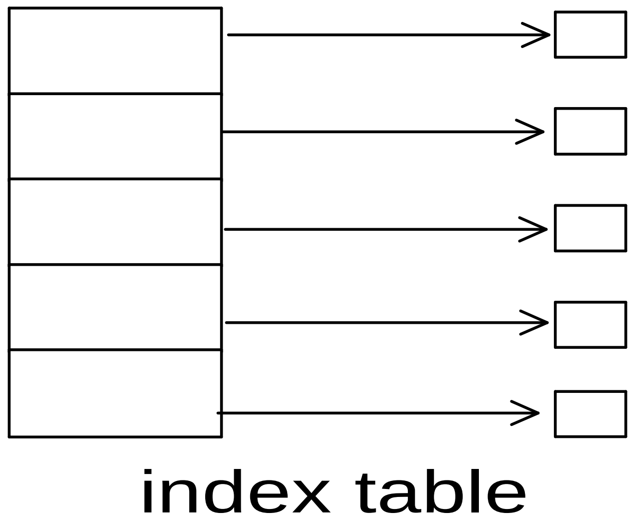
File-Allocation Table



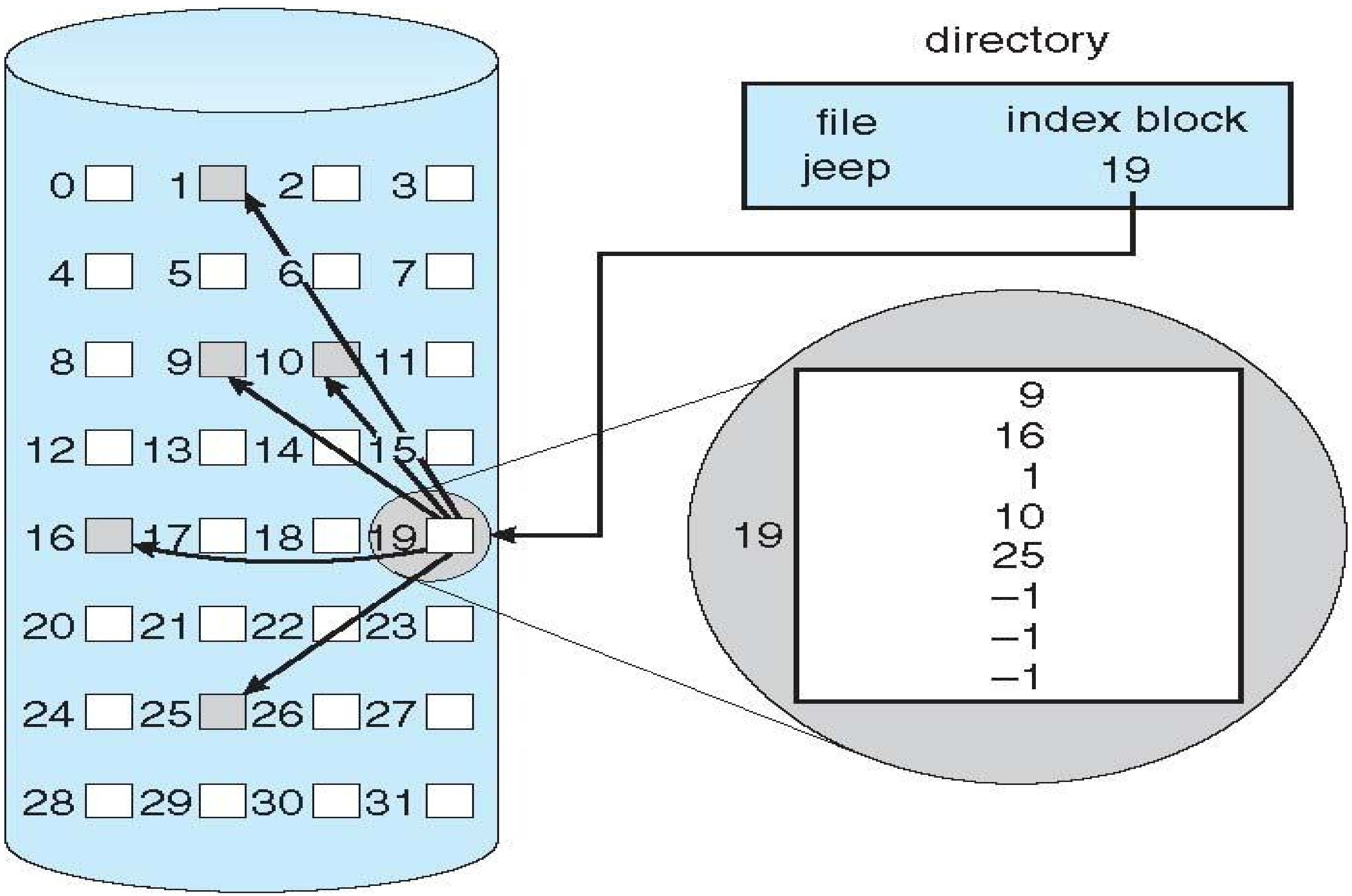
Allocation Methods - Indexed

- **Indexed allocation**
  - Each file has its own **index block**(s) of pointers to its data blocks

- Logical view



Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block
- Mapping from logical to physical in a file of maximum size of 256K bytes and block size of 512 bytes. We need only 1 block for index table

LA/512 → Q, R

Q = displacement into index table
R = displacement into block

Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
- Linked scheme – Link blocks of index table (no limit on size)

Q_1 = block of index table
R_1 is used as follows:

LA / (512 x 511) → Q_1, R_1

Q_2 = displacement into block of index table
R_2 displacement into block of file:

R_1 / 512 → Q_2, R_2

Indexed Allocation – Mapping (Cont.

- Two-level index (4K blocks could store 1,024 four-byte pointers in outer index -> 1,048,567 data blocks and file size of up to 4GB)

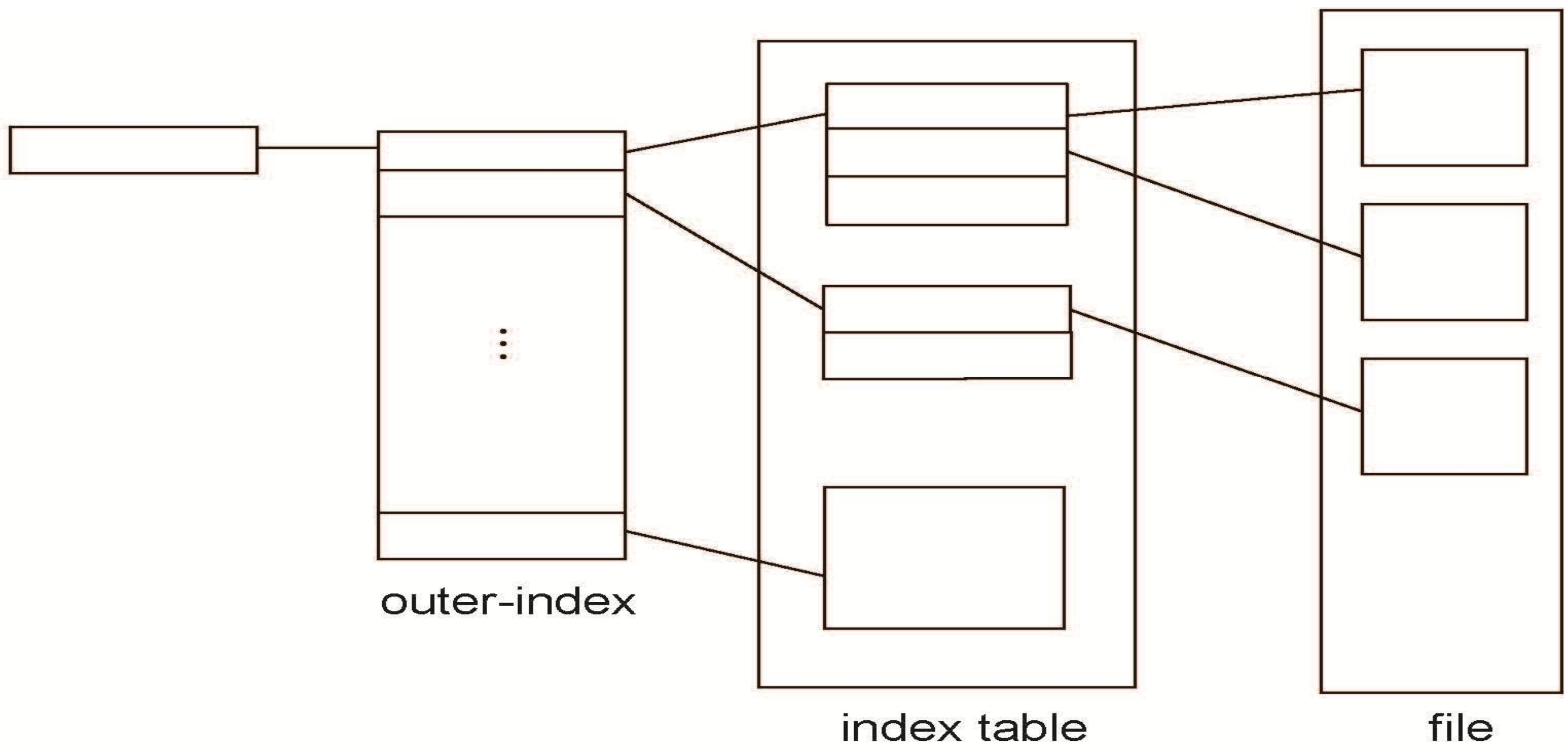
LA / (512 x 512) → Q_1, R_1

Q_1 = displacement into outer-index
R_1 is used as follows:

R_1 / 512 → Q_2, R_2

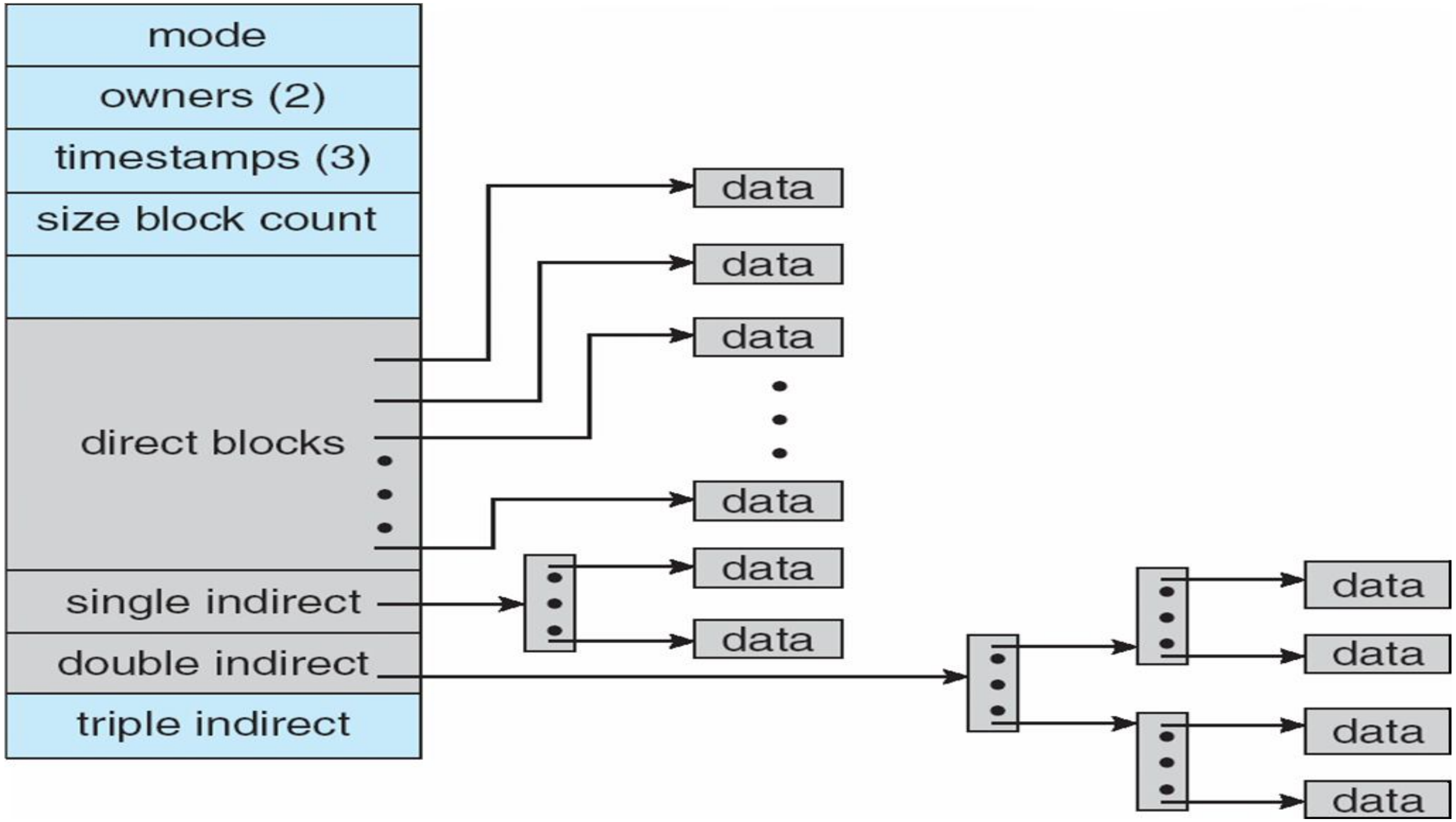
Q_2 = displacement into block of index table
R_2 displacement into block of file:

Indexed Allocation – Mapping (Cont.)



Combined Scheme: UNIX UFS

4K bytes per block, 32-bit addresses



More index blocks than can be addressed with 32-bit file pointer

Performance

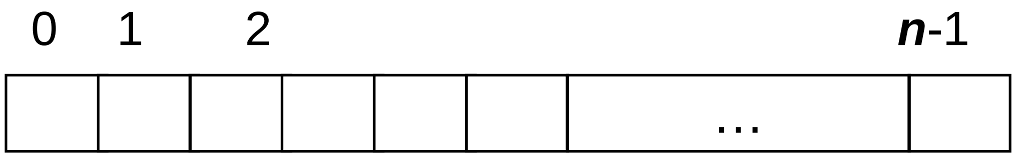
- Best method depends on file access type
    - Contiguous great for sequential and random
- Linked good for sequential, not random
- Declare access type at creation -> select either contiguous or linked
- Indexed more complex
    - Single block access could require 2 index block reads then data block read
    - Clustering can help improve throughput, reduce CPU overhead

Performance (Cont.)

- Adding instructions to the execution path to save one disk I/O is reasonable
  - Intel Core i7 Extreme Edition 990x (2011) at 3.46Ghz = 159,000 MIPS
    - http://en.wikipedia.org/wiki/Instructions_per_second
  - Typical disk drive at 250 I/Os per second
    - 159,000 MIPS / 250 = 630 million instructions during one disk I/O
  - Fast SSD drives provide 60,000 IOPS
    - 159,000 MIPS / 60,000 = 2.65 millions instructions during one disk I/O

Free-Space Management

- File system maintains **free-space list** to track available blocks/clusters
  - (Using term "block" for simplicity)
- **Bit vector** or **bit map** (***n*** blocks)



bit[*i*] =
- 1 ⇒ block[*i*] free
- 0 ⇒ block[*i*] occupied

Block number calculation

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

CPUs have instructions to return offset within word of first "1" bit

Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 4KB = 2^{12} bytes

    disk size = 2^{40} bytes (1 terabyte)

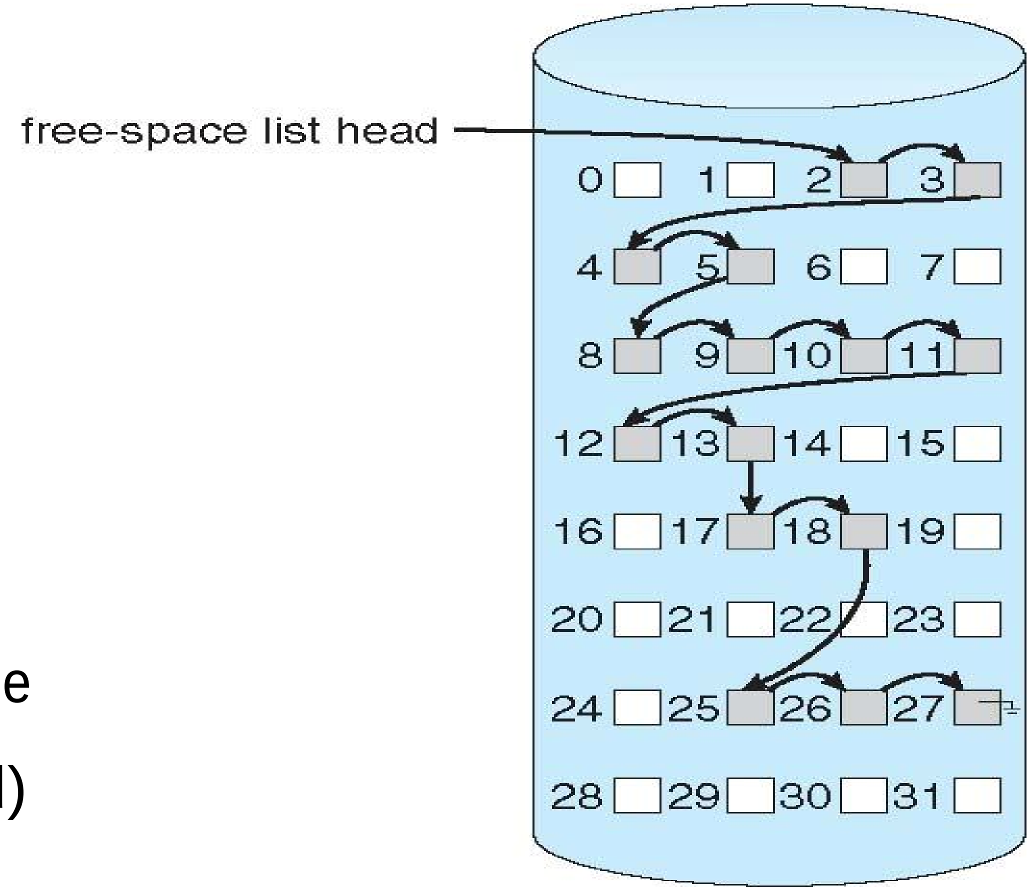
    $\boldsymbol{n} = 2^{40} / 2^{12} = 2^{28}$ bits (or 32MB)

    if clusters of 4 blocks -> 8MB of memory

- Easy to get contiguous files

Linked Free Space List on Disk

- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
  - No need to traverse the entire list (if # free blocks recorded)



Free-Space Management (Cont.)

- **Grouping**
  - Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)
- **Counting**
  - Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering
    - Keep address of first free block and count of following free blocks
    - Free space list then has entries containing addresses and counts

TEXT BOOK

1. Abraham Silberschatz, Peter B. Galvin, “Operating System Concepts”, 10th Edition, John Wiley & Sons, Inc., 2018.

2. Jane W. and S. Liu. “Real-Time Systems”. Prentice Hall of India 2018.

3. Andrew S Tanenbaum, Herbert Bos, Modern Operating Pearson , 2015.

REFERENCES

1. William Stallings, “Operating Systems: Internals and Design Principles”,9th Edition, Prentice Hall of India., 2018.

2. D.M.Dhamdhere, “Operating Systems: A Concept based Approach”, 3rd Edition, Tata McGraw hill 2016.

3. P.C.Bhatt, “An Introduction to Operating Systems–Concepts and Practice",4th Edition, Prentice Hall of India., 2013.

THANK YOU