STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY

CS 492: Operating Systems

File Systems (2)

Instructor: Iraklis Tsekourakis

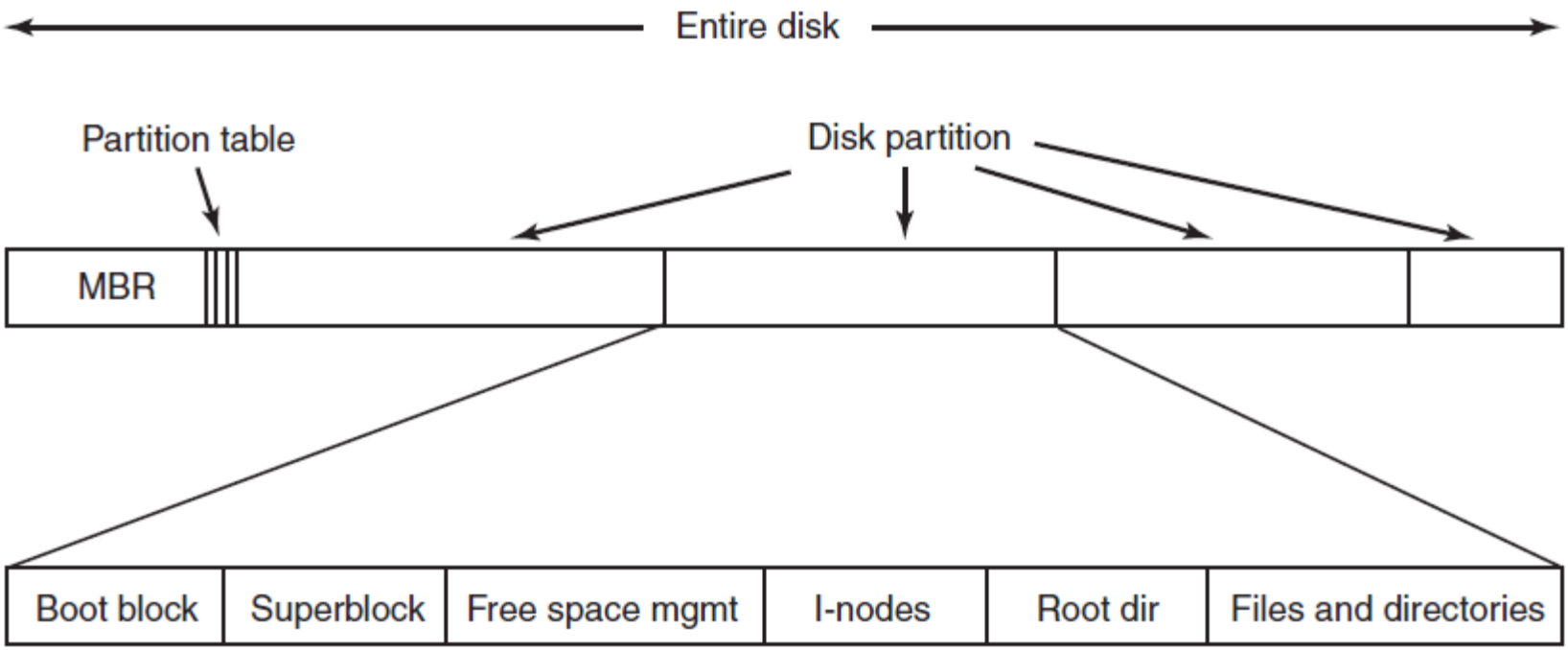
Email: itsekour@stevens.edu

Allocation Methods

Three allocation methods:

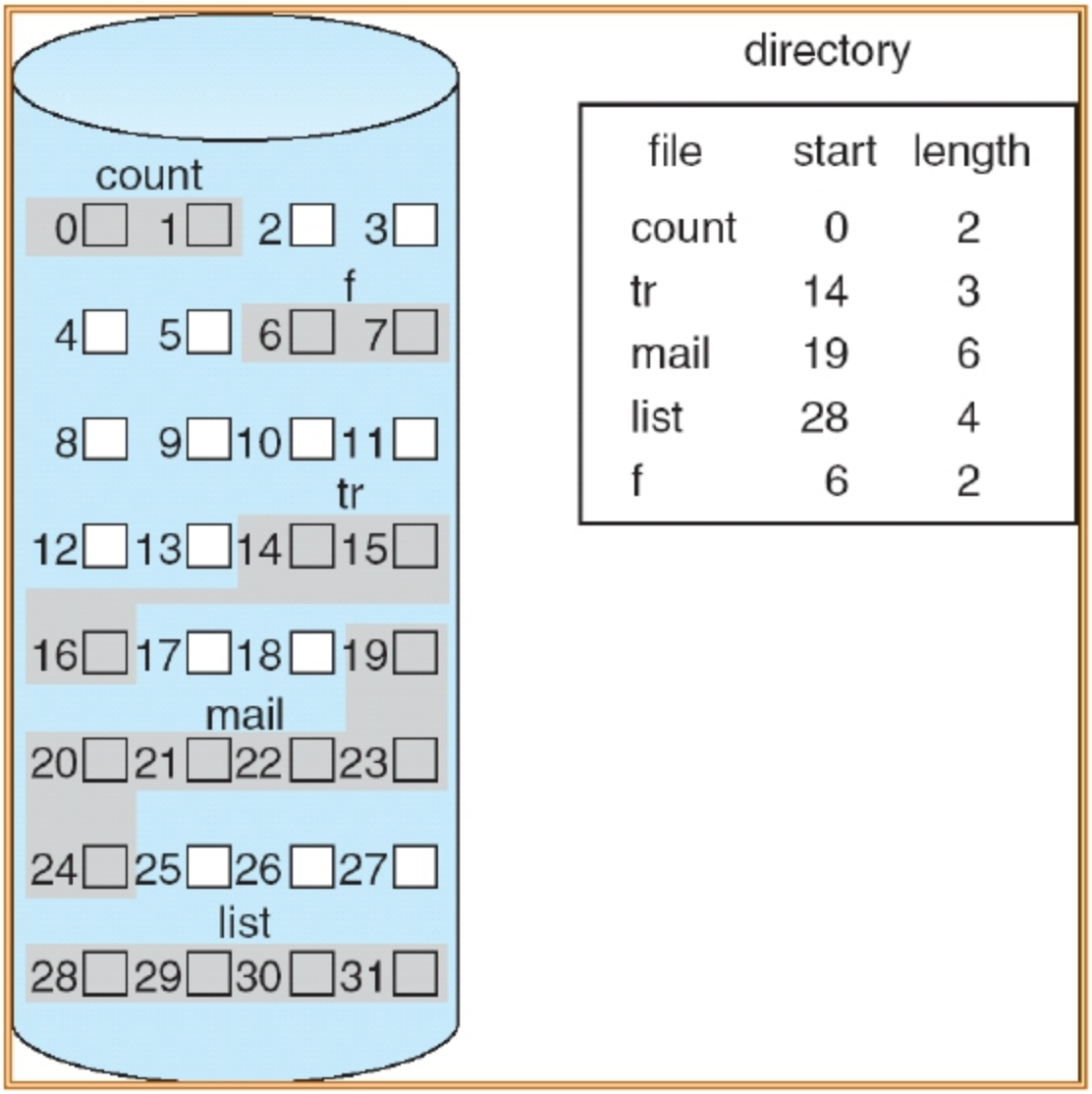
1. **Contiguous allocation**
2. **Linked allocation**
3. **Indexed allocation**

File System Layout



A possible file system layout.

Contiguous Allocation



- Each file occupies a set of contiguous blocks on the disk

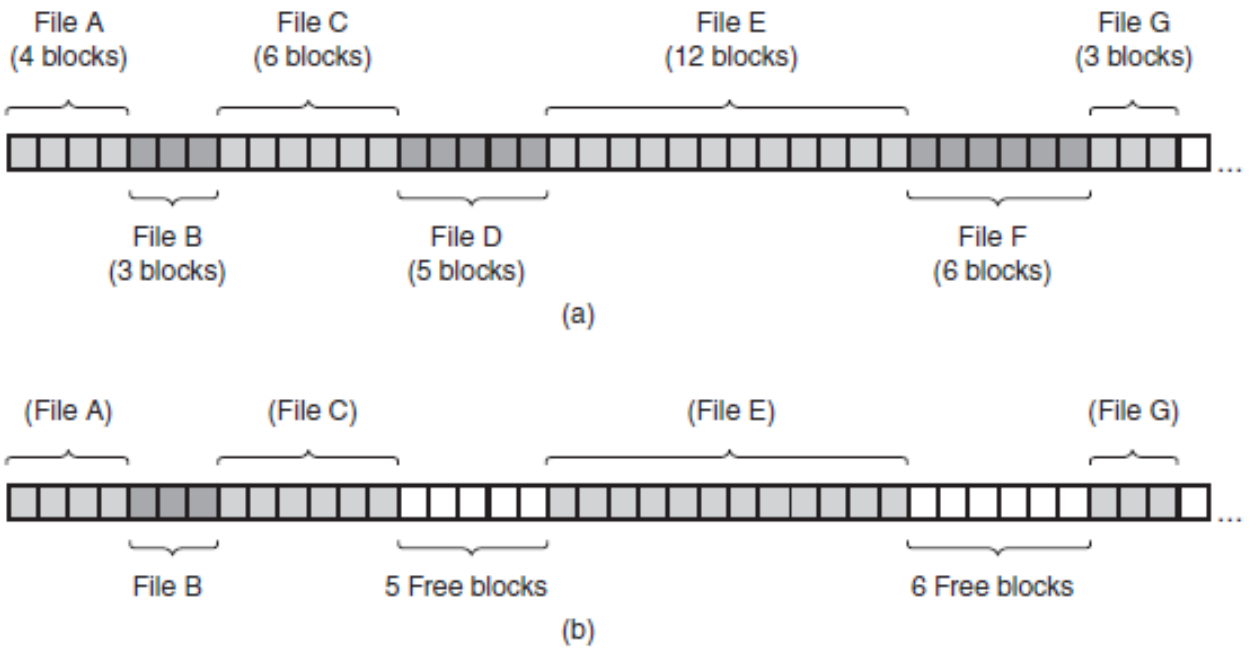
Pros:

- Simple – only starting location (block #) and length (number of blocks) are required
- Random access

Cons:

- Wasteful of space (dynamic storage-allocation problem)
  - External fragmentation: may need to compact space
- Files cannot grow

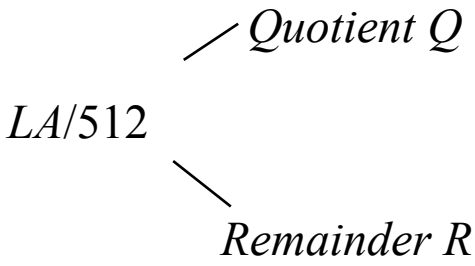
Implementing Files
Contiguous Layout



(a) Contiguous allocation of disk space for seven files. (b) The state of the disk after files *D* and *F* have been removed.

Contiguous Allocation (cont'd)

- Question: Given a logical address *LA* of file A, how to map LA to its physical address (*B*,*D*) (B: block number; D: block offset)*?*
- Suppose the block size is 512 bytes:

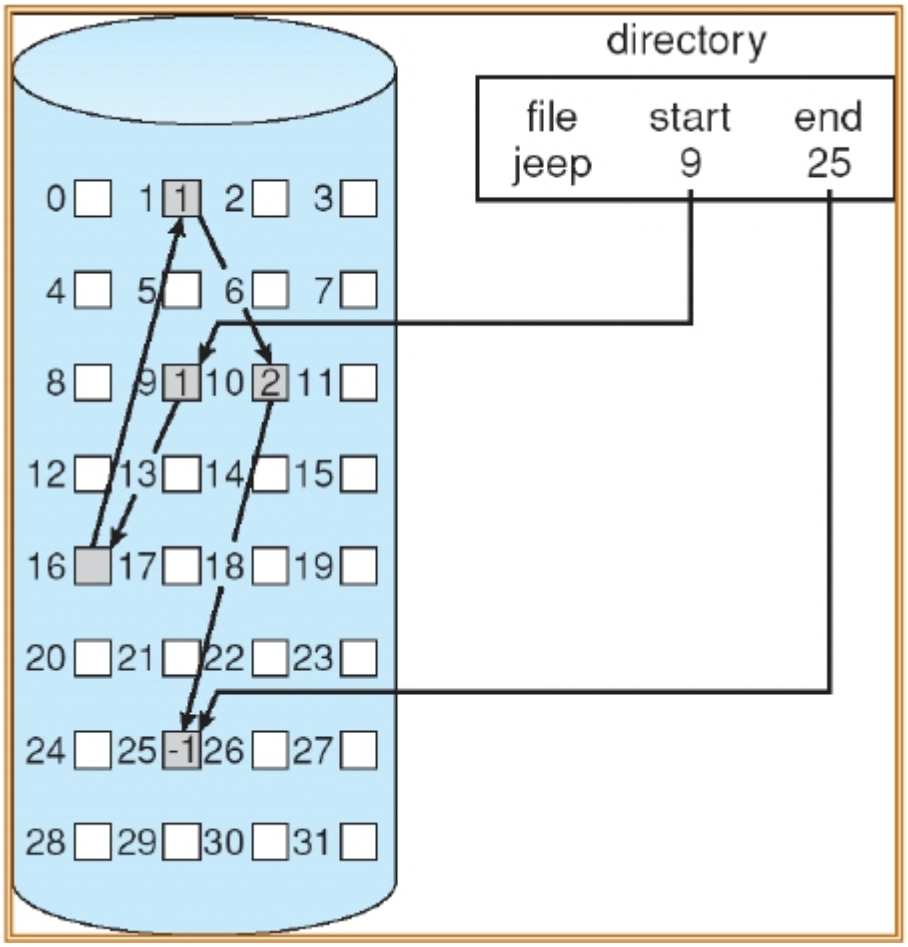


file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

directory

- *Block number B = Q + starting address of file A in directory*
- *Block offset D = R*
- *Number of memory access to get the data at address LA:*
  - *1 access for reading directory to get starting address of file A*
  - *1 access for reading data from block B at offset R.*

Linked Allocation

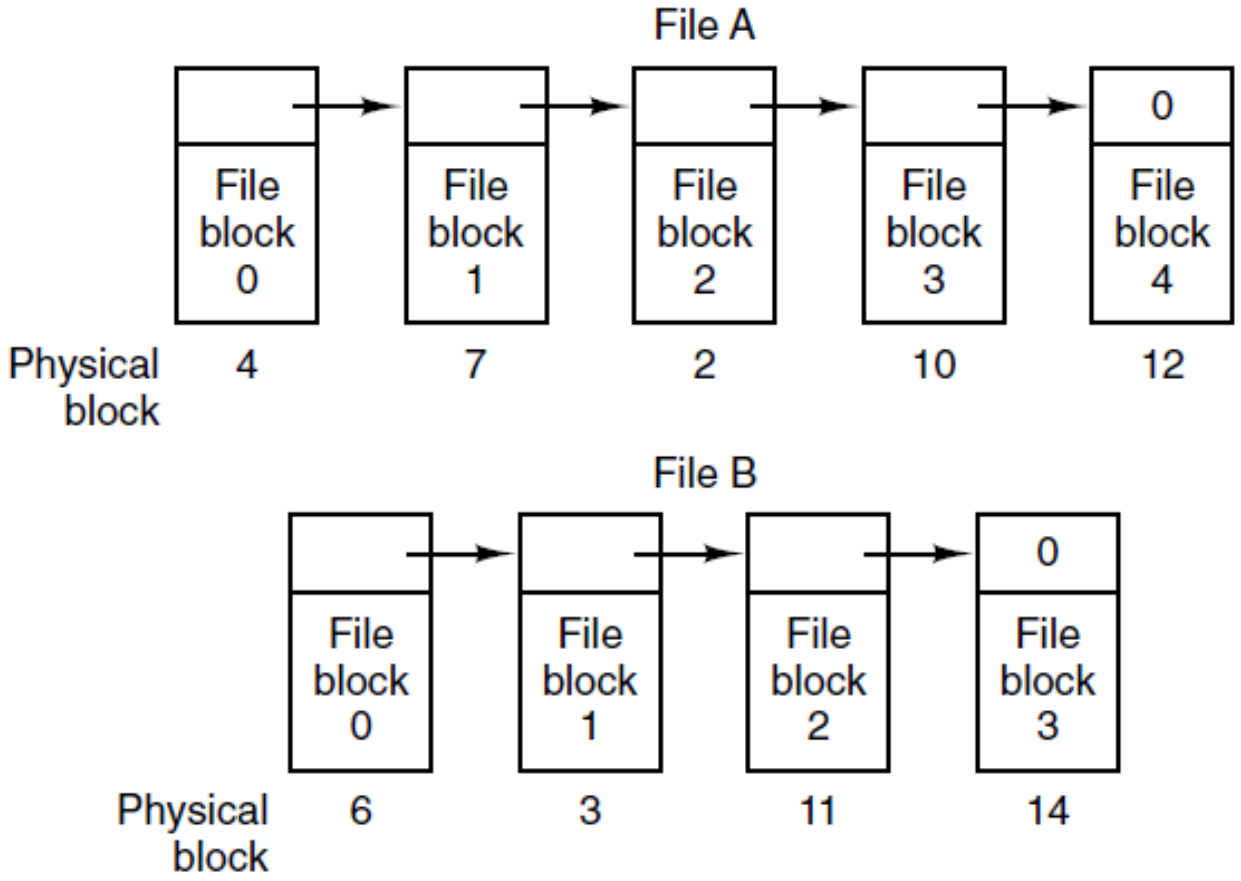


- Each file is a linked list of disk blocks
- Blocks may be scattered anywhere on the disk

block = pointer | Data

- Pros:
  - Simple – need only starting address
  - Free-space management system – no waste of space
- Cons:
  - No efficient random access

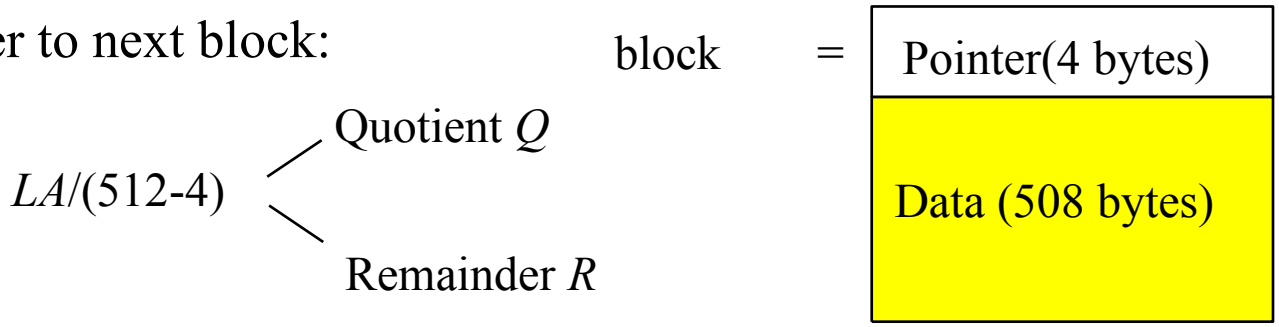
Implementing Files
Linked List Allocation



Storing a file as a linked list of disk blocks.

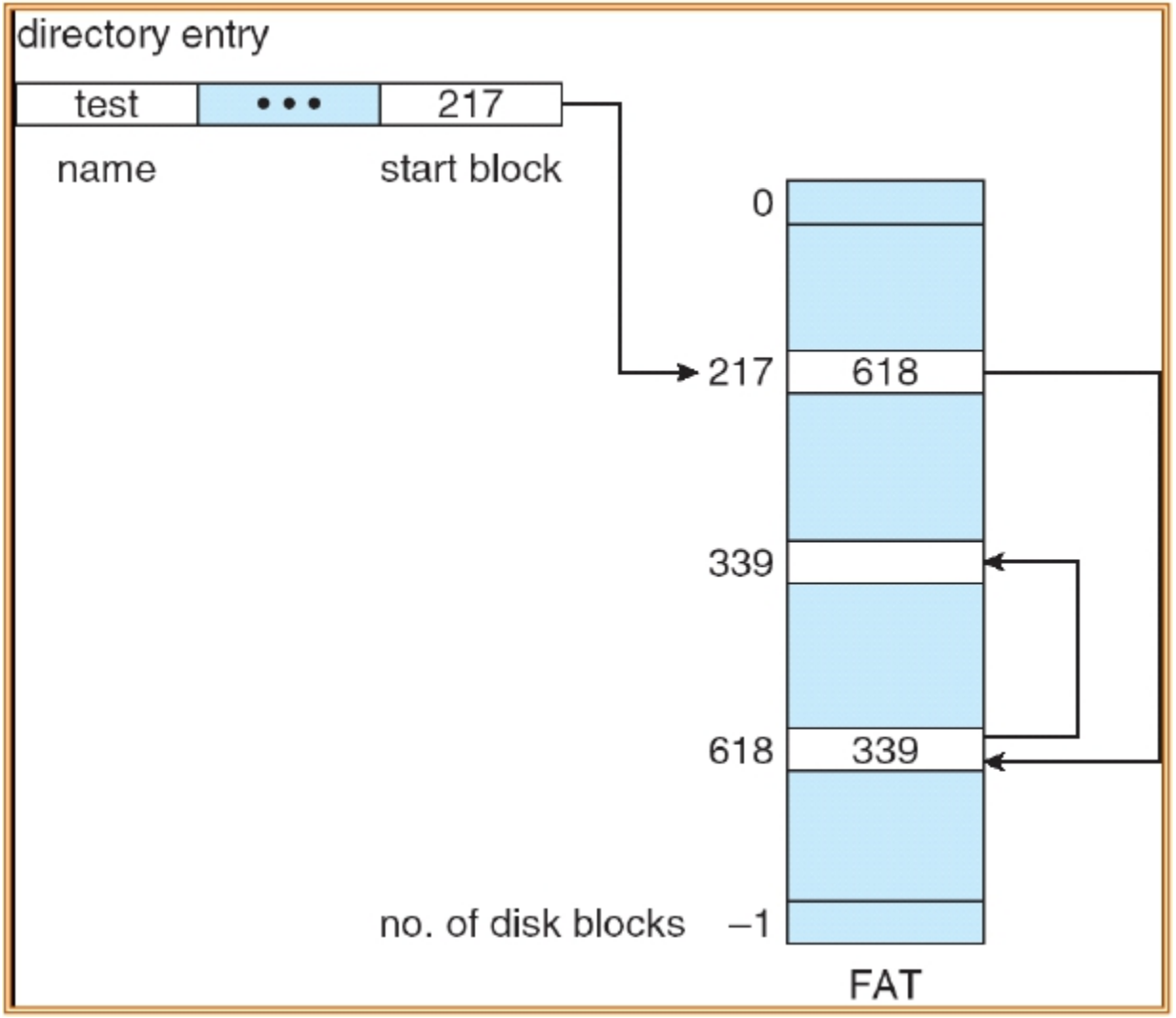
Linked Allocation (Cont.)

- Question: given a logical address *LA* of file A, how to map LA to its physical address (*B*,*D*) (B: block number; D: block offset)*?*
- Suppose block size is 512 bytes and each block contains 4 bytes reserved for pointer to next block:



- *Block number B* = *Q*th block in the linked chain of blocks, starting from the start block in directory
- *Block offset D* = *R* + 4
- *Number of memory access to get the data at address LA:*
  - *1 access for reading directory to get starting block of file A*
  - *Q accesses for traversing Q blocks*

File-Allocation Table (FAT)

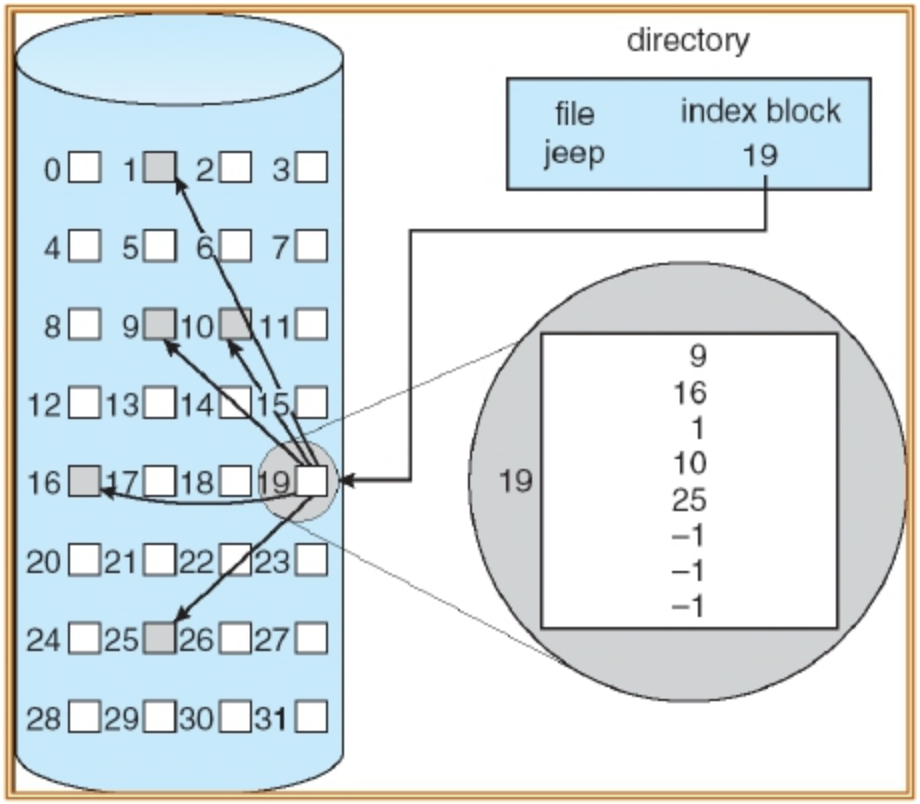


Variant of linked list allocation

Each FAT entry corresponds to disk block number

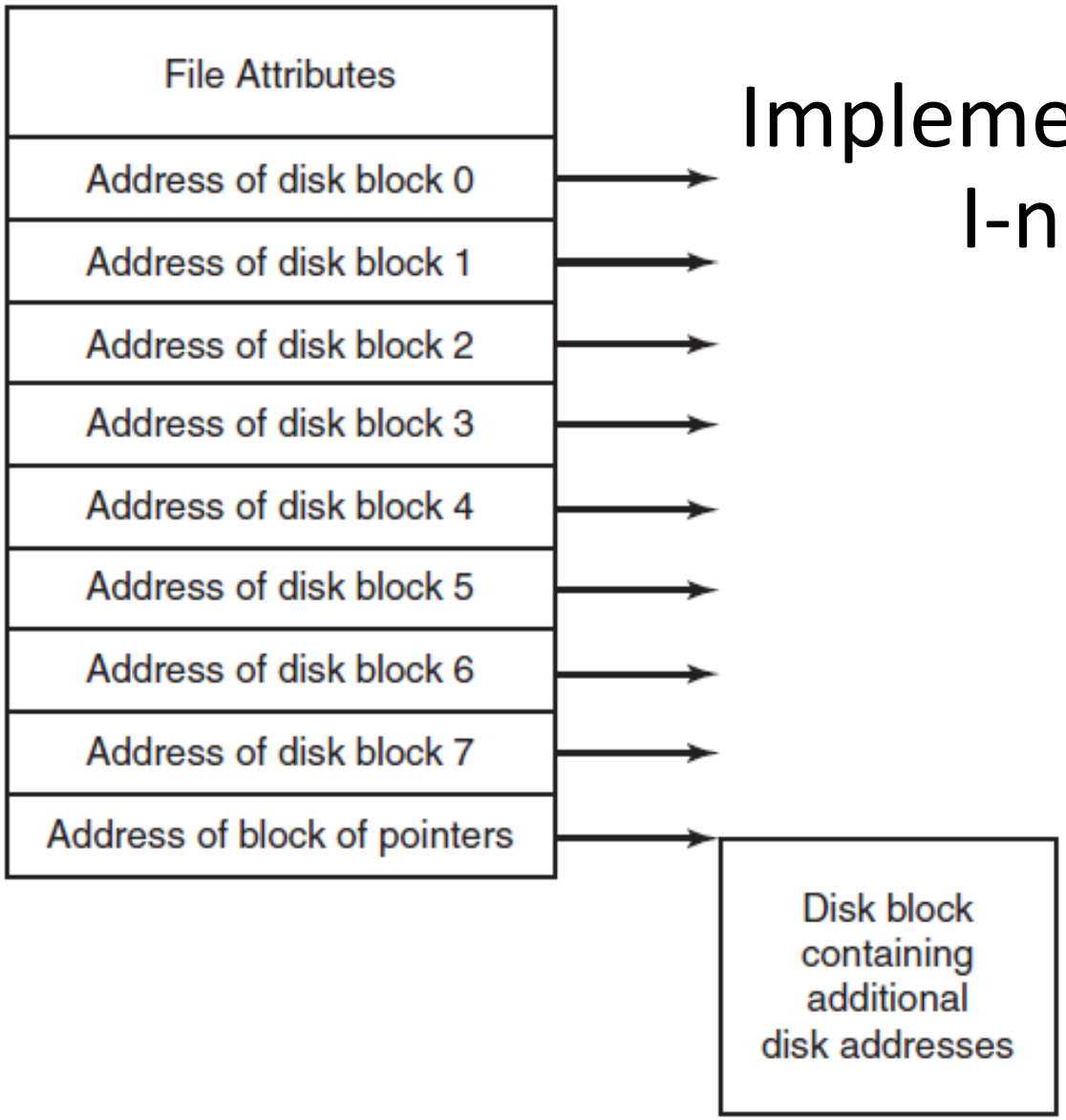
Each FAT entry contains a pointer to the next block or 0

Indexed Allocation



- Brings all pointers together into the **index block (**also called i-node)
- Pros:
  - Efficient random access
  - Dynamic access without external fragmentation
  - Index table storage overhead (way better than FAT)
- Cons:
  - What if i-node doesn't fit all pointers?

Implementing Files
I-nodes

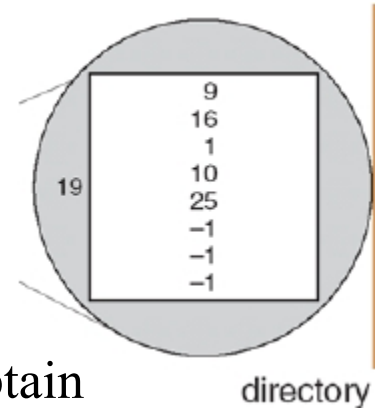


Question?

- If the i-node of the previous slide contains 10 direct addresses and these were 8 bytes each and all disk blocks were 1024B, what would the largest possible file be?

Indexed Allocation (Cont.)

- Question: given a logical address *LA* of file A, how to map LA to its physical address (*B*,*D*) (B: block number; D: block offset *)*?
- Assume the block size is 512 bytes



LA/512 → *Quotient Q*, *Remainder R*

- Block number B: Look up the Q-th entry of the index table to obtain *B*
- *Block offset D = R*
- *Number of memory access to get the data at address LA:*
  - *1 access for reading the index table*
  - *1 access to get data at block offset D.*

Exercise

Consider a hypothetical file system. A disk block contains 512 bytes. Assume:

(1) it takes one disk access to find file X's entry in the directory, and

(2) nothing is cached in memory.

Question:

To read the data at logical address 5116 - 5119 in file X, how many disk block accesses are required (including the initial one to get the directory entry) . Let's consider two file allocation schems:

(1) physically contiguous?

(2) linked in a list, with the "next pointer" from one block to the next embedded in the block and occupying the first 2 bytes of the block?

Question

- Consider a file whose size varies between 4 KB and 4 MB during its lifetime. Which of the three allocation schemes (contiguous, linked and table/indexed) will be most appropriate?