OPERATING SYSTEMS

by
Marwa Yusuf

Lecture 17
Wednesday 6-1-2021

Chapter 4 (4.3 to 4.4.1)
File Systems

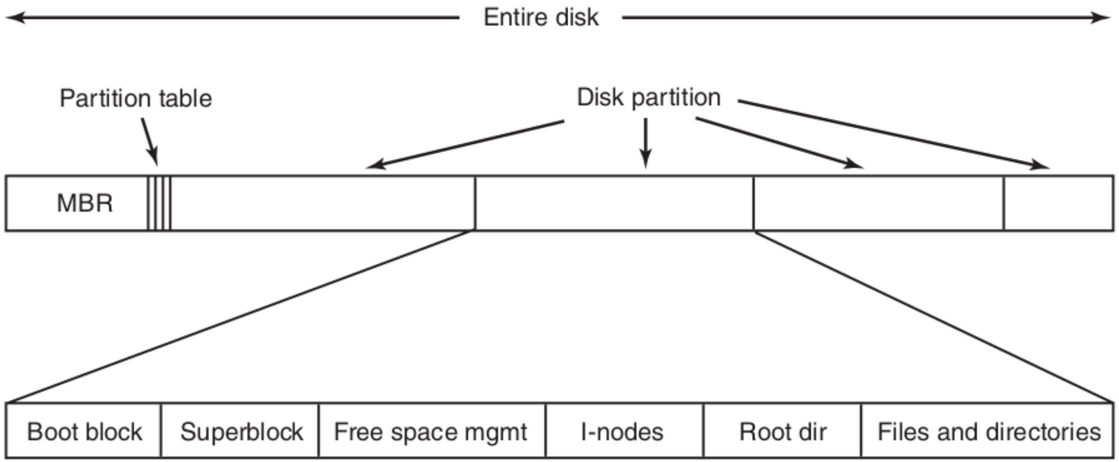
File System Implementation

- Users are interested in naming files, operations, directory trees … etc.
- Implementors (**YOU**) are interested in how to store, disk space management, efficiency and reliability … etc.
  - This is what will be discussed now.

File System Layout

- Disks -usually – are divided into partitions, each with its own file system.
- **MBR** (**Master Boot Record**): on sector 0.
  - Used to boot the computer.
- The end of **MBR** contains **partition table**.
  - Contains start and end of each partition.
  - One of partition is marked **active**.
- When the computer boots:
  - BIOS reads in and executes **MBR**,
  - Which locates the active partition and reads in its 1st block: **boot block**, and executes it,
  - In which exists the program that loads the **OS** stored on that partition.
- Every partition contains a **boot block**:
  - Uniformity.
  - It might get one in the future.

File System Layout cont.

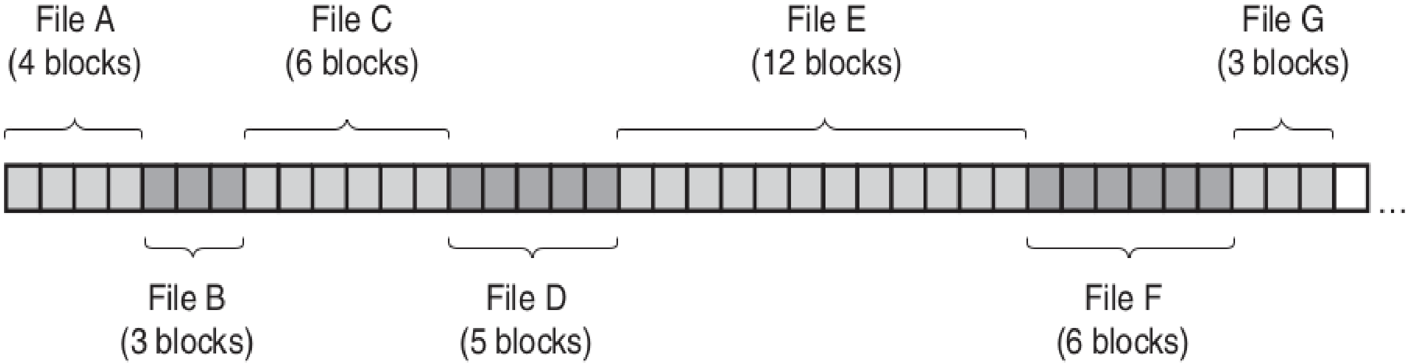


- The layout varies from a file system to another.
- **Superblock**: contains all key parameters about FS.
  - Read into memory at 1st use (may be at boot).
  - May contain: magic number for FS type, # of blocks … etc.
- Free blocks: bitmap or pointers list.
- I-node: one/file, info about the file.

Implementing Files

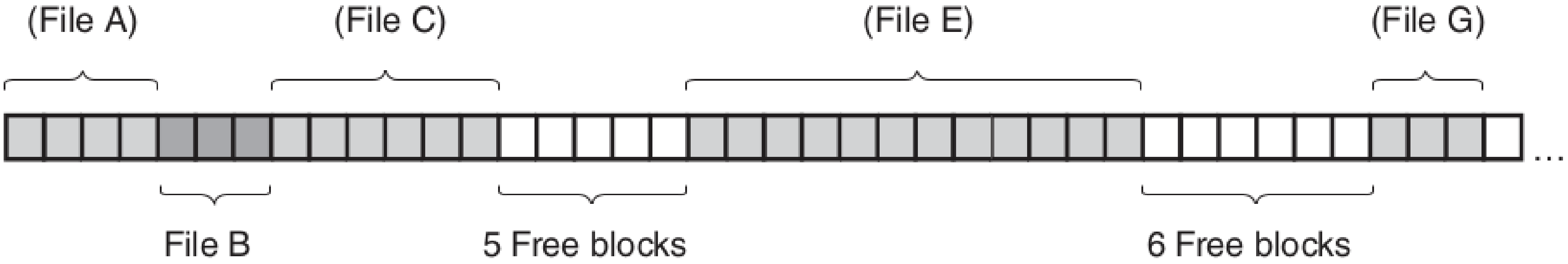
- Important: manage which disk blocks go with which files.
- Different methods.

Contiguous Allocation



- Each file occupies a contiguous (integer) number of blocks.
- Each file starts at a new block.
- Advantages:
  - Simple to implement: first block, # of blocks.
  - Fast to read: only one seek at the 1st block.

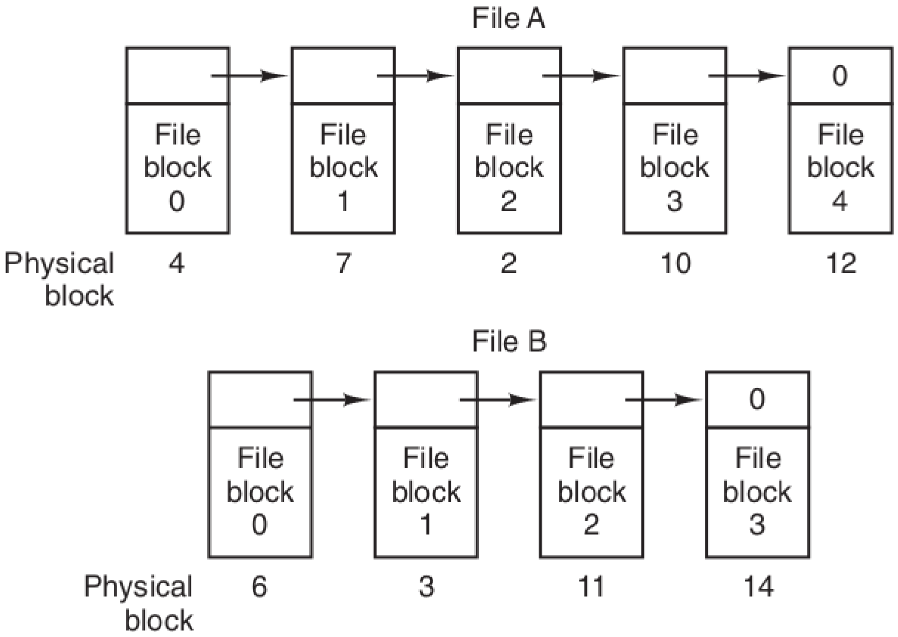
Contiguous Allocation



- Disadvantage: Fragmentation over time.
- Initially not a problem, but gets worse with time.
- Solutions:
  - Compaction: may take hours or even days.
  - Use holes: must keep track of holes and know the final size of file in advance.
    - Imagine that **word** asks you to specify the exact size before starting!
- Acceptable with CD-ROM: files known in advance, read only.
  - Was used in magnetic disk, then abandoned for its problems, then used again in CD-ROM.

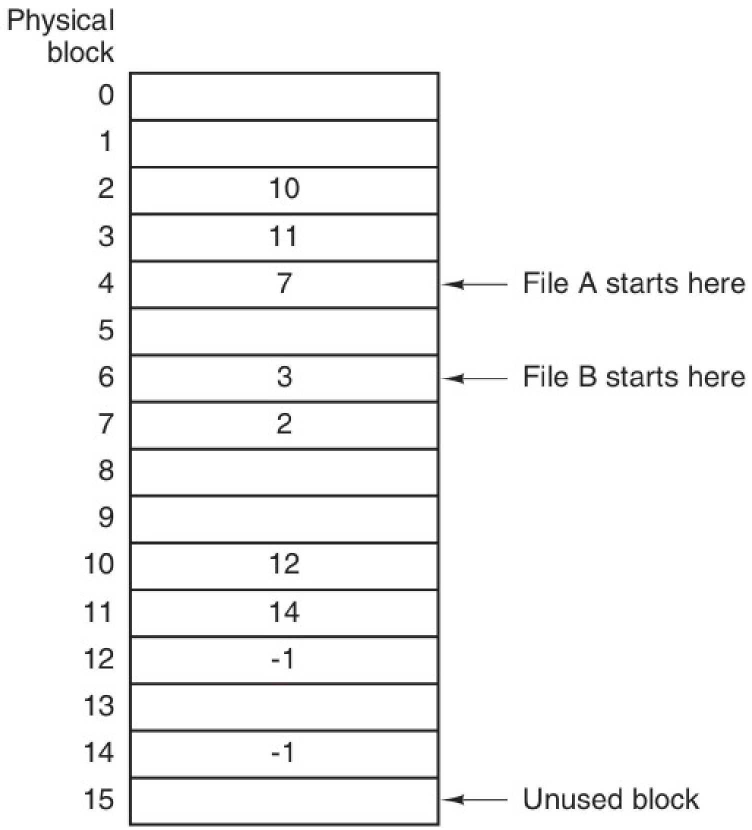
Linked List Allocation

- First word is a **next** pointer.
- No external fragmentation.
- Store only 1st block address in directory.
- Disadvantage:
  - Random access is very slow.
  - Data size in a block not a power of 2: programs reading in units of block need to concatenate from 2 blocks → more overhead.



Linked List Allocation using A table in Memory

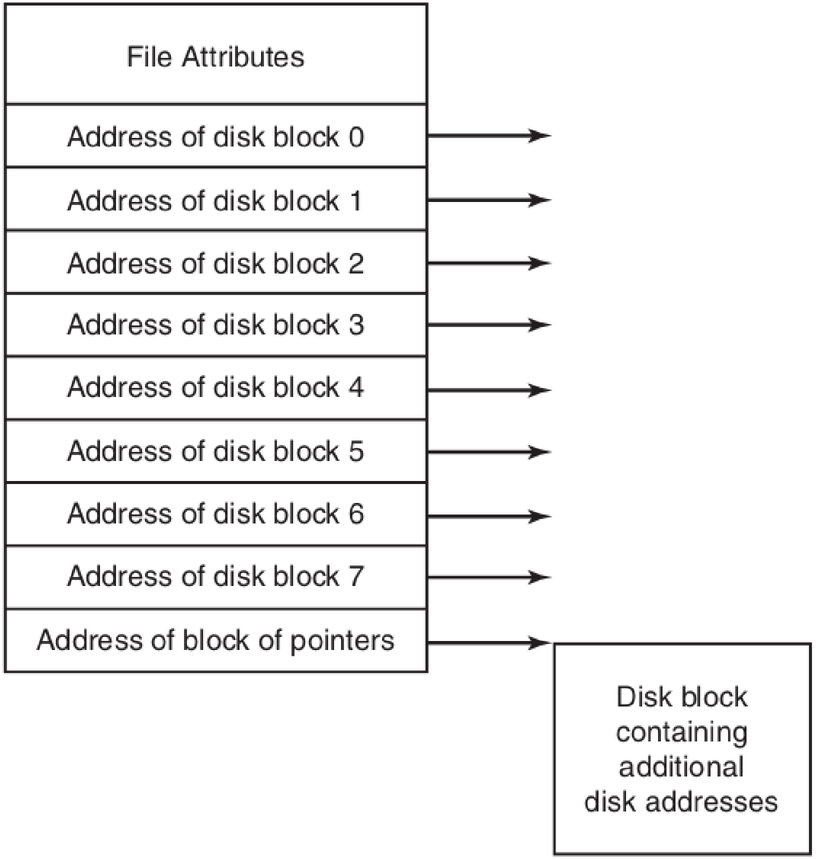
- Put pointers in a table in memory: **FAT** (**File Allocation Table**)
- Entire block used for data.
- Still need to follow the chain, but faster in memory.
- The directory just keeps the starting block number.
- Disadvantage: tables must stay in memory all the time. 1TB disk with 1KB block → 1GB entries.
  - Does not scale well.



Physical block		
0		
1		
2	10	
3	11	
4	7	File A starts here
5		
6	3	File B starts here
7	2	
8		
9		
10	12	
11	14	
12	-1	
13		
14	-1	
15		Unused block

I-nodes

- **I-node** (**index-node**) data structure per file.
- In memory only when the file is open.
  - If i-node size = n, and k files open → only kn memory needed.
  - Size proportional to # of open files, not disk size.
  - If the file grows larger than i-node size limits → reserve last disk address to point to another block that holds addresses.
    - More levels scheme is possible.
    - More last reserved addresses also is possible.



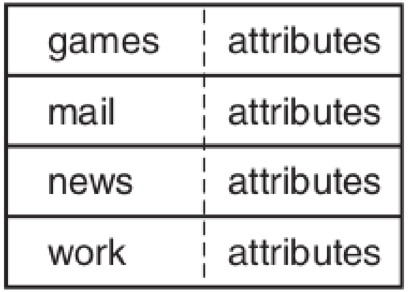
Implementing Directories

- To read a file, open it.
- To open a file, path name is used to locate directory on disk.
- Directory entry provides info to locate file disk blocks.
- Directory system: map ASCII file name into info to locate its blocks.

Implementing Directories - Attributes

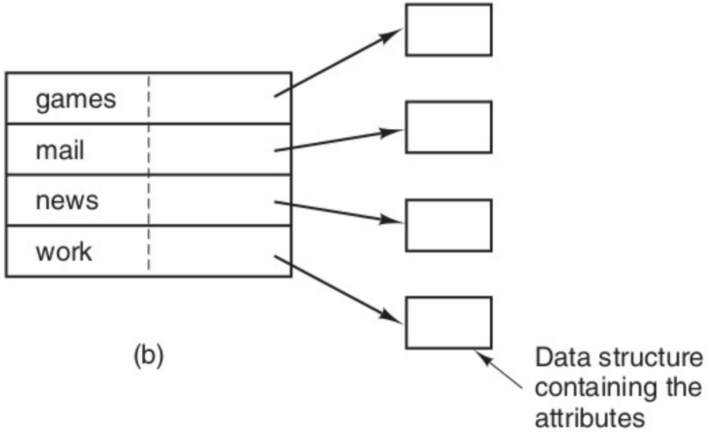
- Where to store file attributes?

- Simple option: in directory entry.
- List of fixed size entries: fixed size name, attributes, address(es).



games	attributes
mail	attributes
news	attributes
work	attributes

- Another option: in i-node.



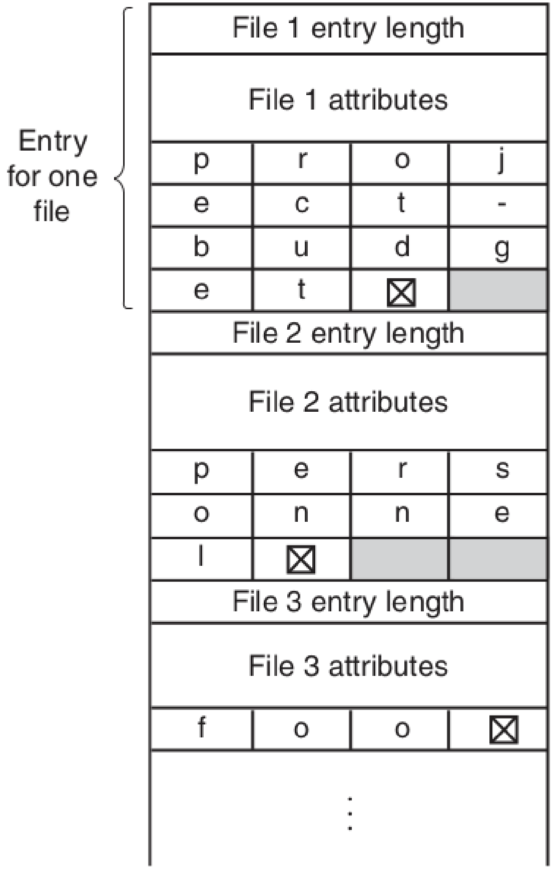
(b)

Implementing Directories – Supporting Long File Names

- With fixed size directory entries, long file names represent a problem.
- Limit file name length : 255 characters, reserve that much length for name in one of the prev. schemes.
  - Simple but wastes disk space with short file names, which are the majority.

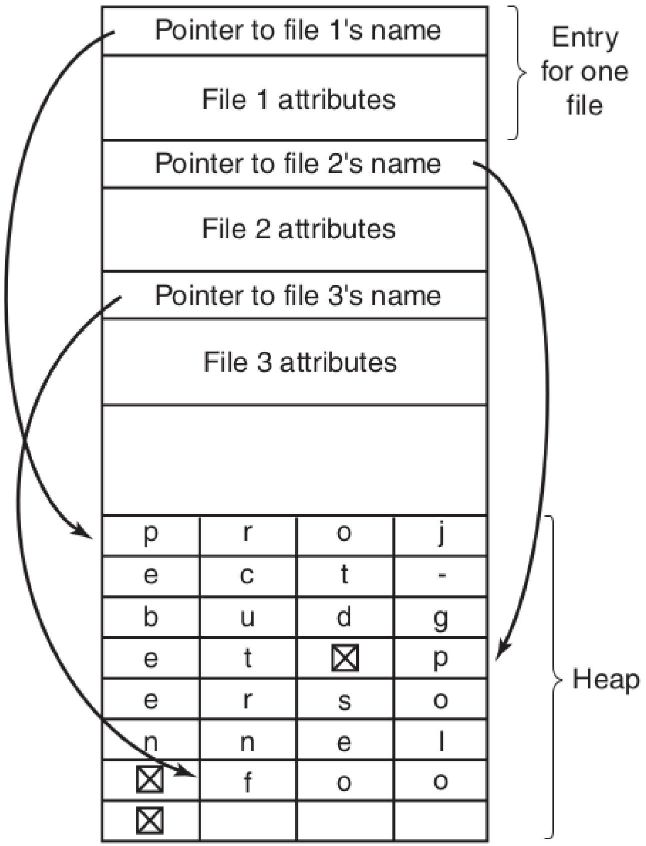
Implementing Directories – Supporting Long File Names

- Each entry:
  - fixed portion (size of entry, attributes such as owner, creation date … etc.)
  - Followed by file name however its length, ending with a special character, padded to align entries with words.
- Disadvantages:
  - Fragmentation with file delete: acceptable because compaction is feasible here.
  - A single entry may span multiple pages → page fault while reading file name.



Implementing Directories – Supporting Long File Names

- Fixed length entries, followed by a heap of file names.
- No fragmentation in entries.
- No padding needed.
- Still need heap management, and page faults can occur.



Implementing Directories – Lookup time

- Files are searched sequentially from the directory beginning → slow lookup.
- One solution: use a hash table.
  - Faster lookup.
  - More complex administration.
  - Used only with large directories.
- Another solution: cache search results.
  - Useful only when there is locality in file search.

Skipped

- Sections 4.3.4, 4.3.5, 4.3.6 and 4.3.7.

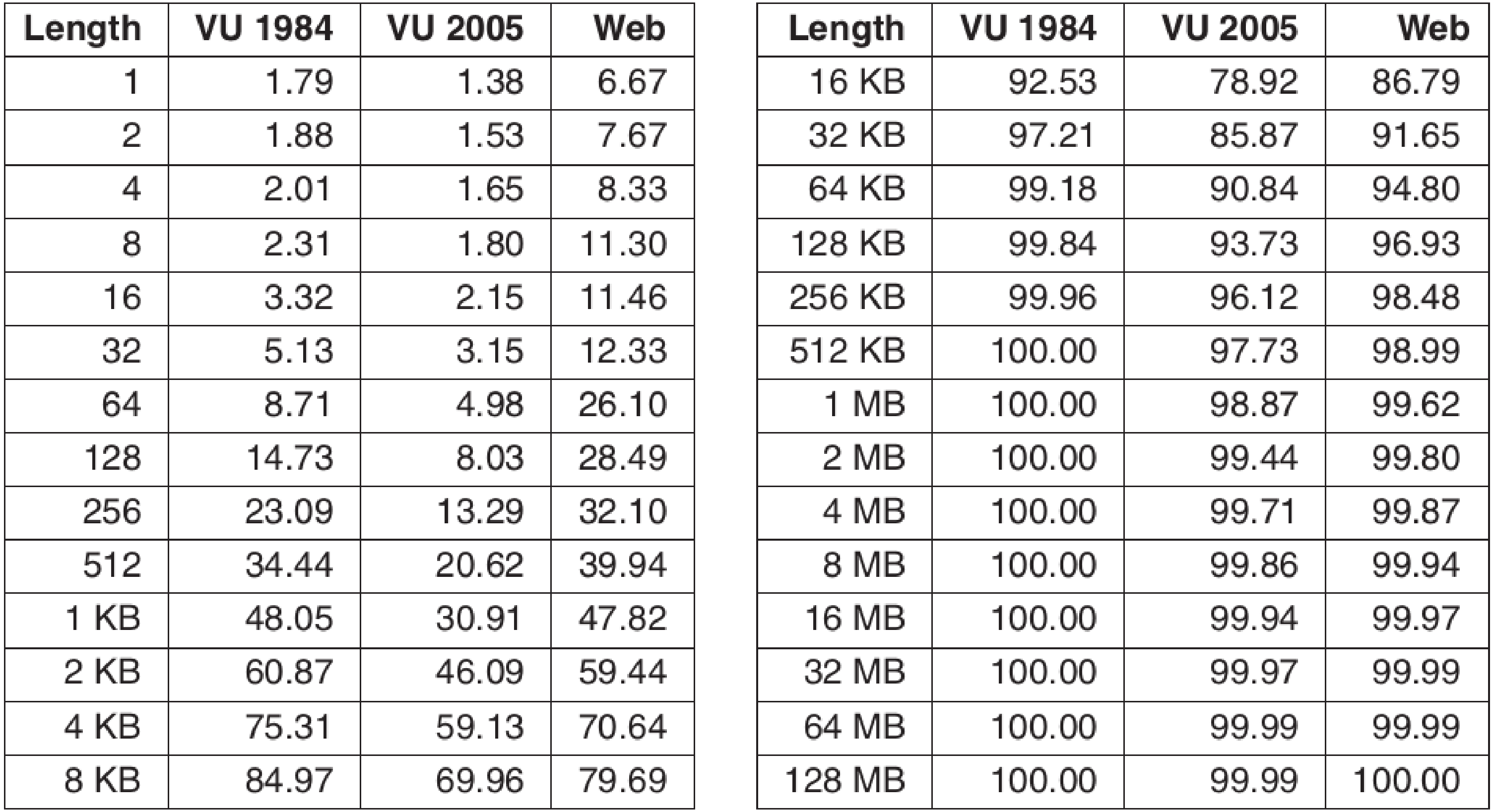
File System Management and Optimization

- How to make FS work efficiently and robustly?

Disk Space Management

- As contiguity is not usually feasible, almost all FSs chop files into blocks.
- Block size:
  - Too large → wastes space
  - Too small → wastes time
- A study about file sizes:
  - 59.13% of all files at the VU were 4 KB or smaller and 90.84% of all files were 64 KB or smaller. The median file size was 2475 bytes.
    - 1KB block → 30% to 50% of files in single block.
    - 4KB block → 60% to 70% of files in single block.
  - 93% of the disk blocks are used by the 10% largest files.
    - Waste in small files is negligible.

Study Results

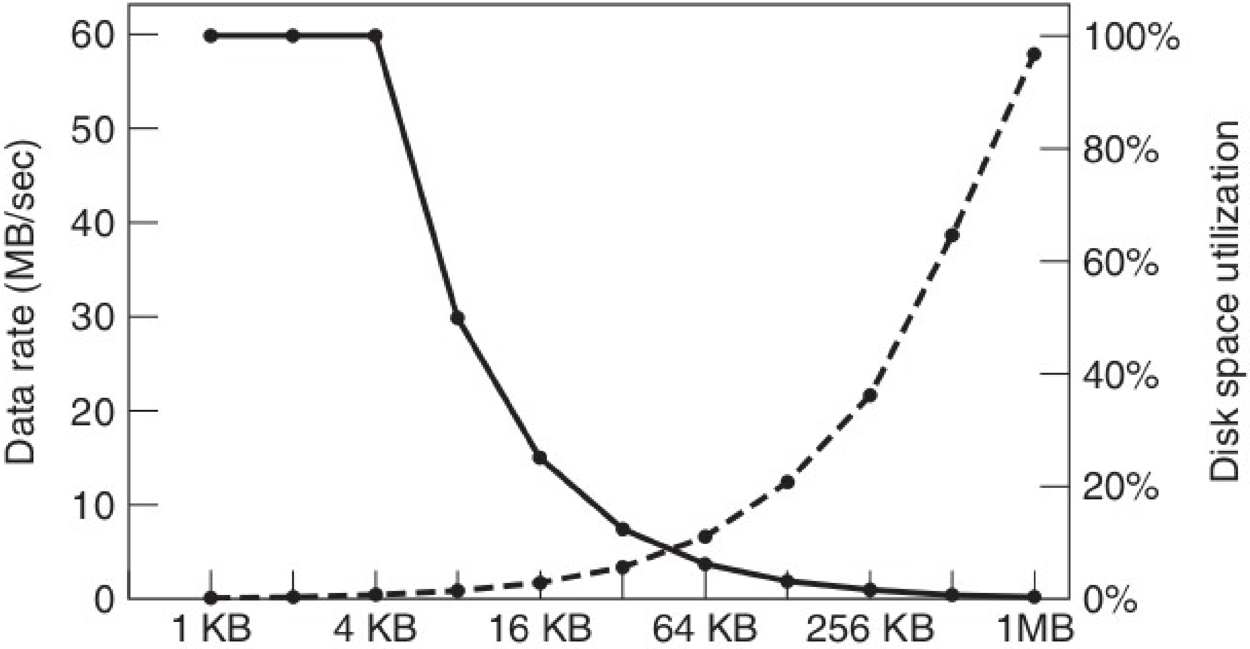


Length	VU 1984	VU 2005	Web
1	1.79	1.38	6.67
2	1.88	1.53	7.67
4	2.01	1.65	8.33
8	2.31	1.80	11.30
16	3.32	2.15	11.46
32	5.13	3.15	12.33
64	8.71	4.98	26.10
128	14.73	8.03	28.49
256	23.09	13.29	32.10
512	34.44	20.62	39.94
1 KB	48.05	30.91	47.82
2 KB	60.87	46.09	59.44
4 KB	75.31	59.13	70.64
8 KB	84.97	69.96	79.69
16 KB	92.53	78.92	86.79
32 KB	97.21	85.87	91.65
64 KB	99.18	90.84	94.80
128 KB	99.84	93.73	96.93
256 KB	99.96	96.12	98.48
512 KB	100.00	97.73	98.99
1 MB	100.00	98.87	99.62
2 MB	100.00	99.44	99.80
4 MB	100.00	99.71	99.87
8 MB	100.00	99.86	99.94
16 MB	100.00	99.94	99.97
32 MB	100.00	99.97	99.99
64 MB	100.00	99.99	99.99
128 MB	100.00	99.99	100.00

Disk Space Management

- Smaller blocks → more blocks/file → more transfers → slower.
- Ex: A disk with 1MB/track, 8.33 msec rotation time, avg 5msec seek time. To read a *k* bytes block:
  - $5 + 4.165 + (k/1000000) \times 8.33$
- A study about block sizes vs data rate and vs space utilization:
  - Assume file size of 4 KB
- With larger disks currently, space is not a limitation, prefer wasting space.

Study Results


Data rate (MB/sec)
1 KB
4 KB
16 KB
64 KB
256 KB
1MB
Disk space utilization