



File System Implementation

CS 241 Lecture 25

S: Ch 12 pp562-569, 535-561

Lawrence Angrave



Prelude

How do create a disk-based file system?

Is the UNIX file system the only way to access permanent data

How is storage allocated for file systems?

Why is the UNIX file system organized the way it is?

Some short answers.

How do create a disk-based file system?

- The data structures and access methods are mapped through *abstraction layers* to the hardware

Is the UNIX file system the only way to access permanent data?

- No.

How is storage allocated for file systems?

- Indexed (and other) allocation algorithms

Why is the UNIX file system organized the way it is?

- Efficient for typical usage patterns

C- More detail required

Why is the UNIX file system organized the way it is?

Most accesses are to small files

Most data is read/written to large files

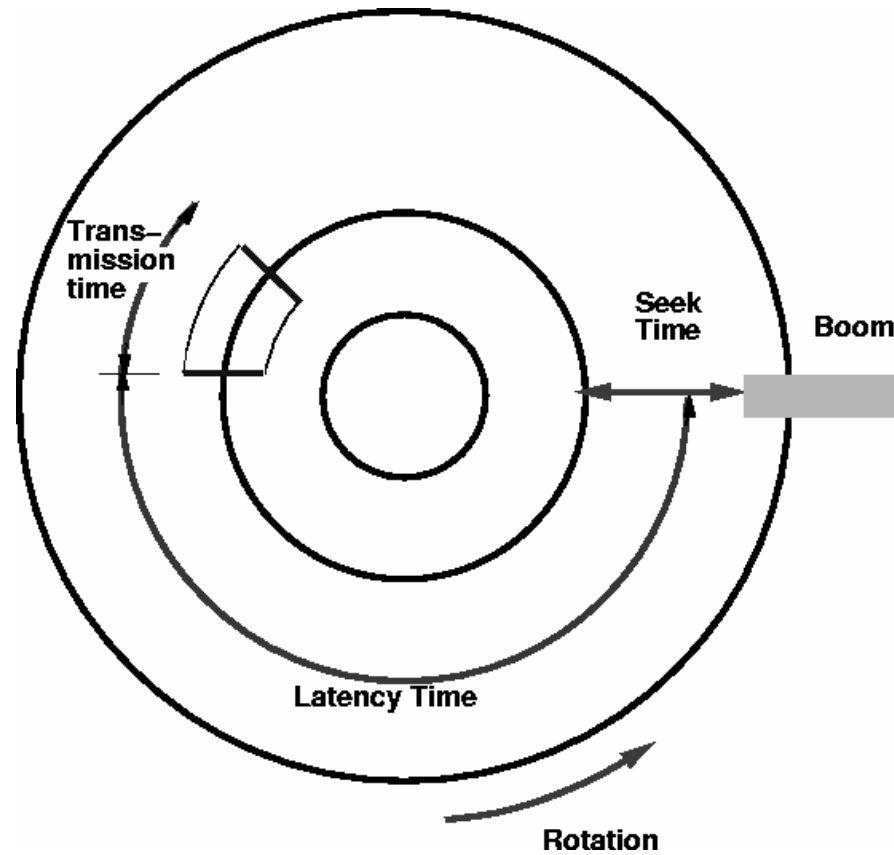
Shell files and Directories mean small files must be accessed fast

Once a file is read (or written) the rest of it is read (or written)

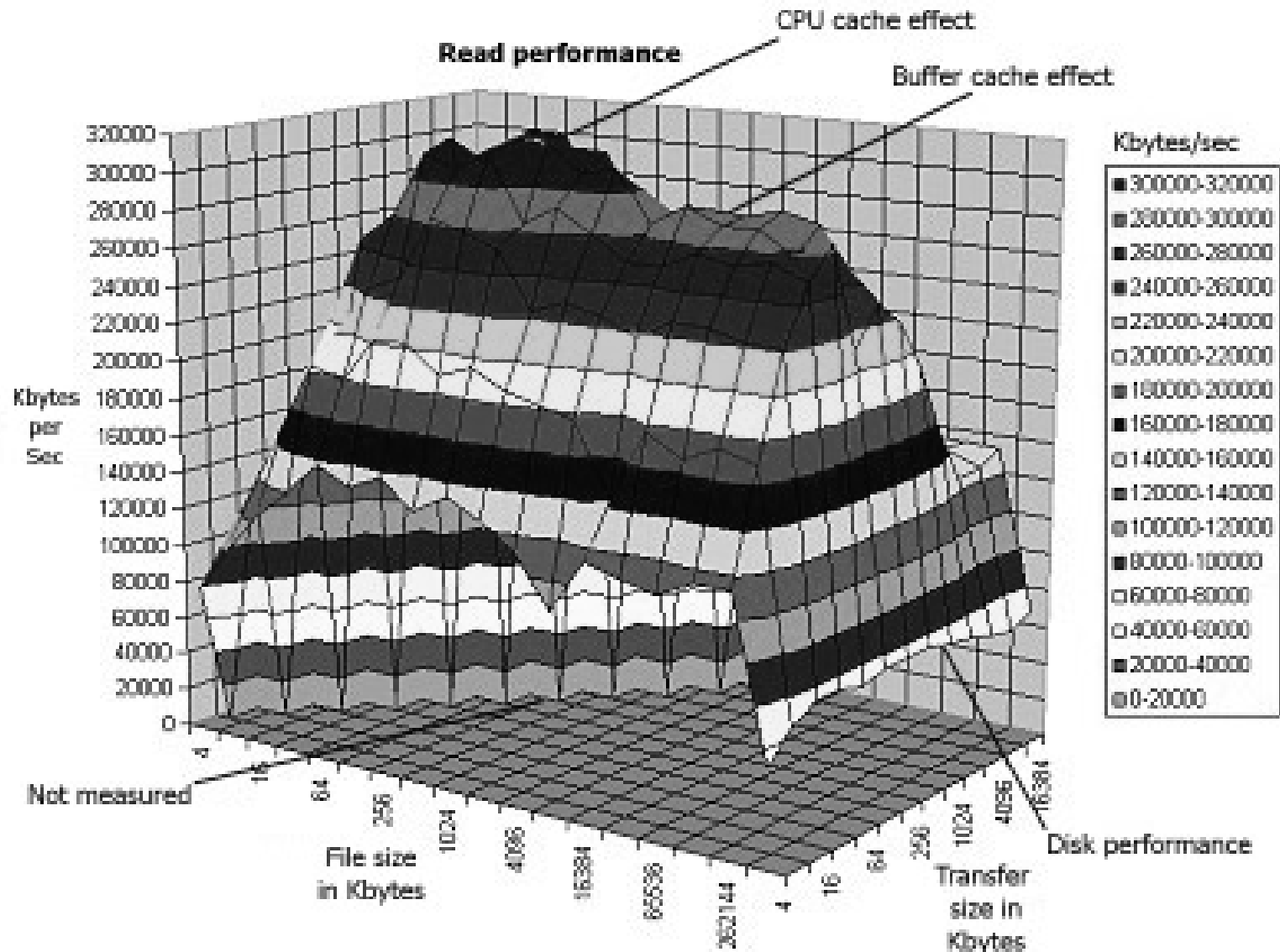
There are more reads than writes

Not all access patterns conform to this (e.g. Database)

Should we measure performance at the disk level?



Better... Investigate actual system performance



Conceptual levels today

File system requirements

Single file access concepts

Disk layout & block access

Partitions

Allocation schemes

File System organization requirements

Long-term

Sharable

Efficient access

Controlled access

Structured

- Field
- Record
- File – Collection of records
- Database

Single File Access Concepts

File

Sequential

Indexed Sequential

Indexed

Direct or Hashed

Single File Access Concepts (S12.2)

Pile – a log of data (usually self describing)

Sequential – fixed format records. Keyed.

- UNIX is sequential, record is byte, key is offset.
- Common operation “Next”. lseek. Additions at end.

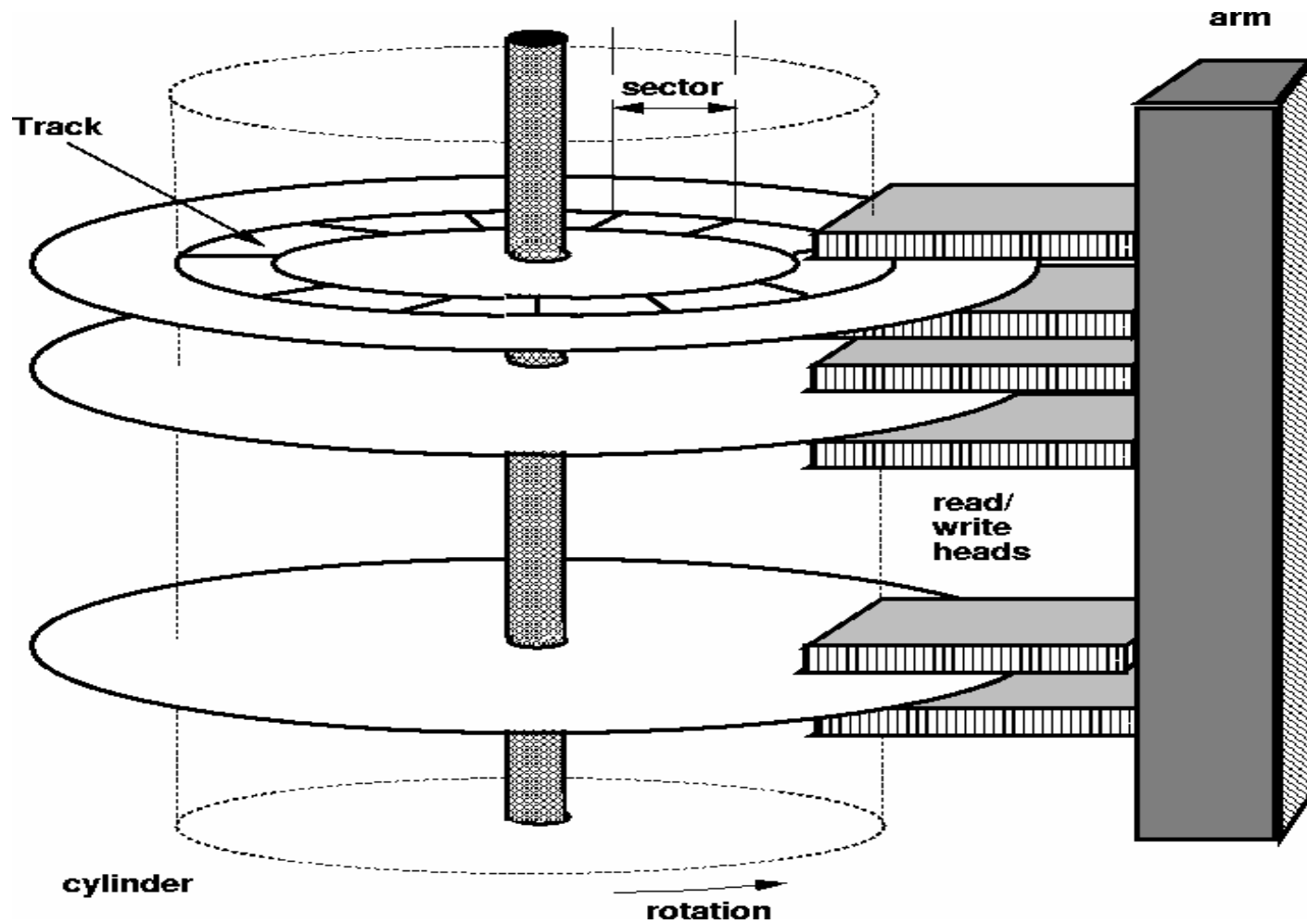
Indexed Sequential. Key and data record.

- Find data with key. Add, Delete, Next.

Indexed. Many fields are indexed

Direct or Hashed. Data hashed to file address

Direct: Go straight to the physical block



Direct. Disk Parameters (Stallings 11A)

Sector has k bytes (typical 32-4096 bytes)

Track has s sectors (size 4-32 sectors)

Surface = j tracks (size 75 to 500 tracks)

Cylinder has t tracks (size 4-32 tracks)

t is equivalent to the number of surfaces

Total storage = $k * s * j * t$

Physical Block Addr = sector * s / surface * cylinder *

Operating System needs to

Read/write to physical block addresses

Present user with a coherent, abstract file system

We need several layers of abstraction (next...)

Logical view: Mounted File System

Allows # files on system to grow for ever

File system must be mounted before it can be available to processes on the system:

The OS is given the name of the device, and the location within the file structure at which to attach the files system (mount point)

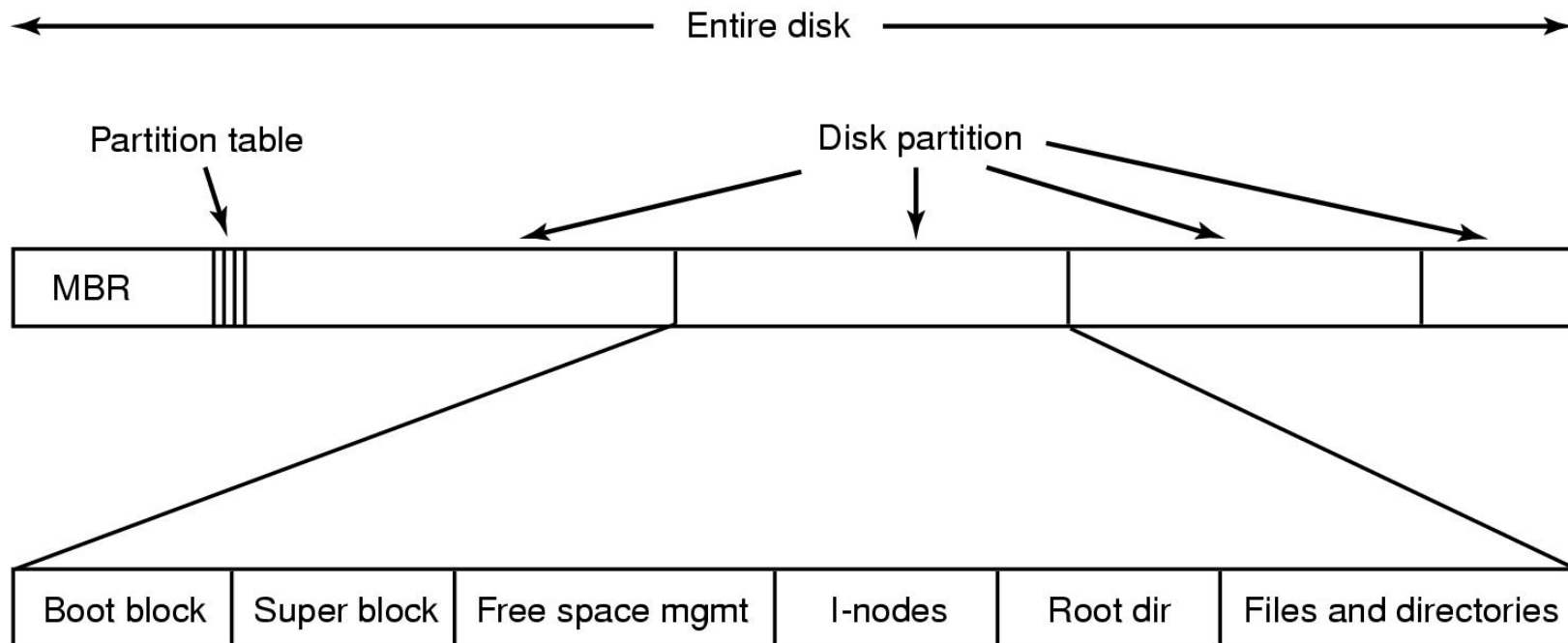
A file system is contained in a partition. A disk can have many partitions.

File System Mounting

OS verifies that the device contains a valid file system

OS notes in its directory structure that a file system is mounted at the specified mount point

File System *Implementation* -Partitions



A typical file system layout
MBR – master boot record

Partitions

Multiple partitions on disk.

Partition can contain file system

File system within partition must be self-describing

Must recover from errors

Must be mountable anywhere

Must describe what sort of file system it contains

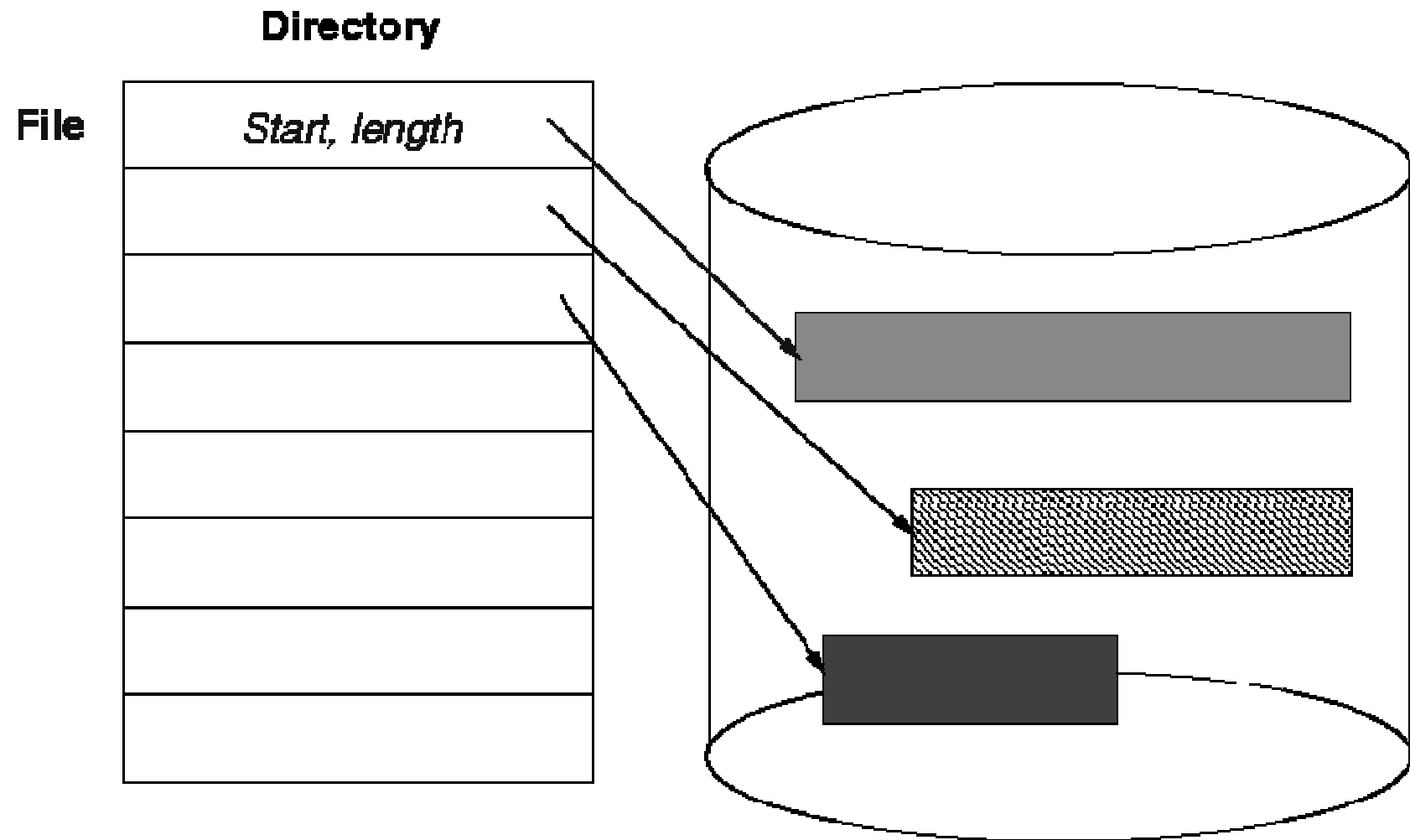
Allocation Schemes

Allocation of Disk Space p554-559

Low level access methods depend upon the disk allocation scheme used to store file data

- Contiguous
- Linked list
- Block or indexed

Contiguous Allocation



Contiguous Allocation Issues

Access method suits sequential and direct access

Directory table maps files into starting physical address and length

Easy to recover in event of system crash

Fast, often requires no head movement and when it does, head only moves one track

Contiguous Allocation Issues

File is allocated large contiguous chunks

User knows size of the file

Expanding the file requires copying

Dynamic storage allocation - first fit, best fit

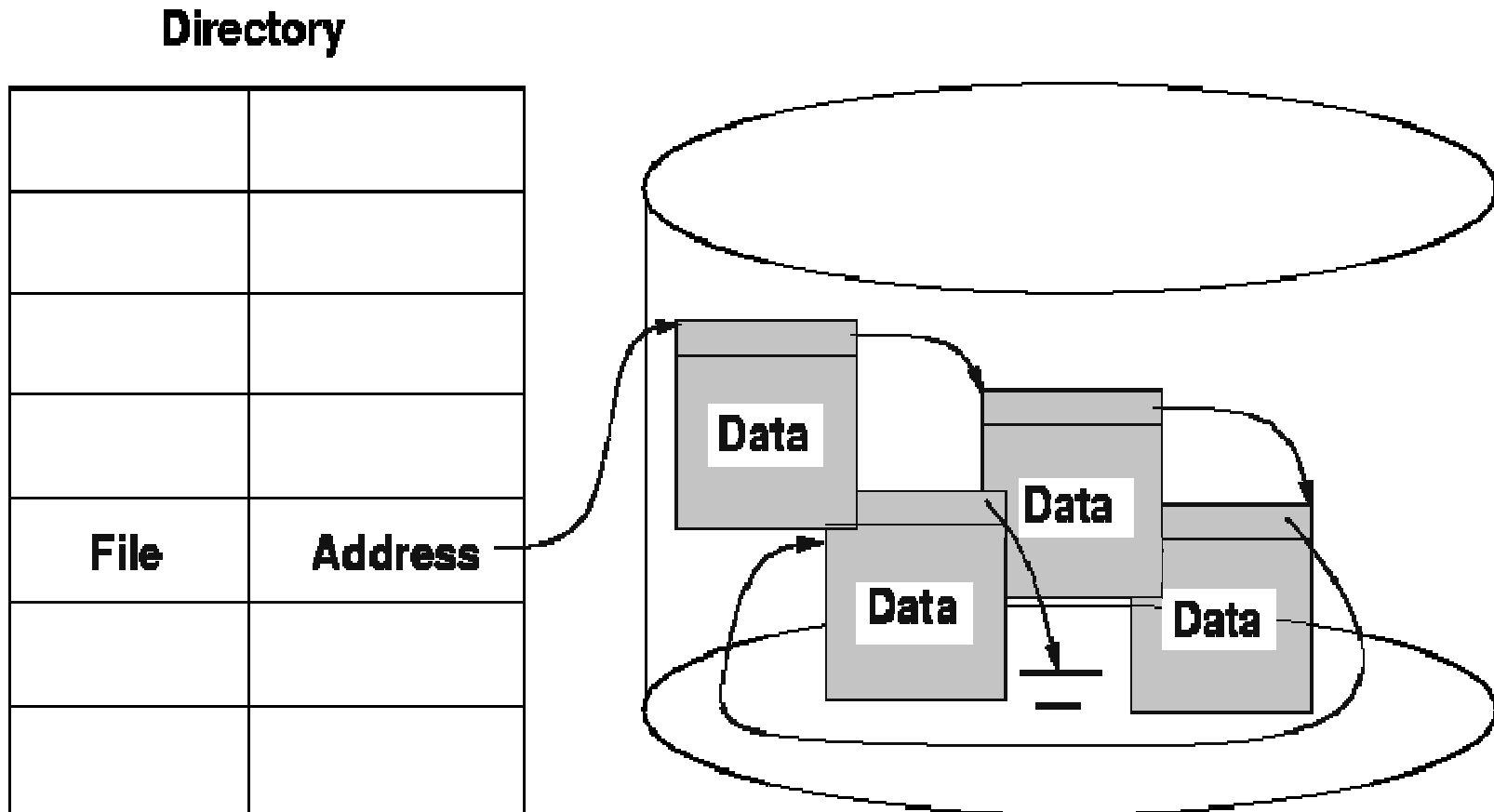
External fragmentation occurs on disk

Users tend to overestimate space => **internal fragmentation**

External Fragmentation

Solution: Linked allocation

Linked Allocation



Linked List Allocation

Each file is a linked list of chunks

Pointers in list are not accessible to user

Directory table maps files into head of list for a file

A node in the list can be a fixed size physical block or a contiguous collection of blocks

Easy to use - no estimation of size necessary

Linked List Allocation

Can grow in middle and at ends

Space efficient, little fragmentation

Slow - defies the principle of locality. Need to read through linked list nodes sequentially to find record of interest blocks

Suited for sequential access but not direct access

Linked List Allocation Issues

Disk space must be used to store pointers (if disk block is 512 bytes, and disk address requires 4 bytes, then the user sees blocks of 508 bytes)

Not very reliable. System crashes can scramble files being updated

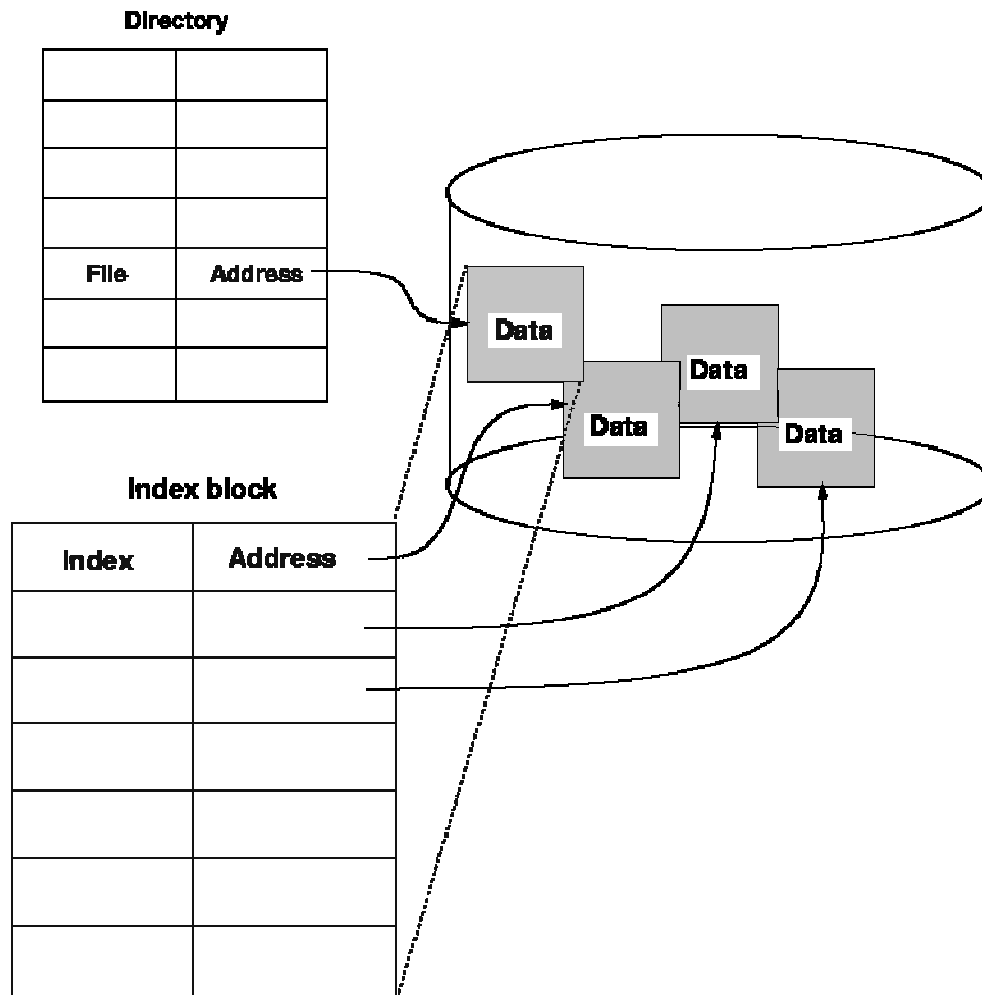
Important variation on linked allocation method: 'file-allocation table' (FAT) - OS/2 and MS-DOS

Linked List Allocation Issues

Summary: linked allocation solves the external fragmentation and size-declaration problems of contiguous allocation,

However, it can't support efficient direct access

#3. Indexed Allocation



Indexed Allocation

Solves external fragmentation

Supports sequential, direct and indexed access

Access requires at most one access to index block first. This can be cached in main memory

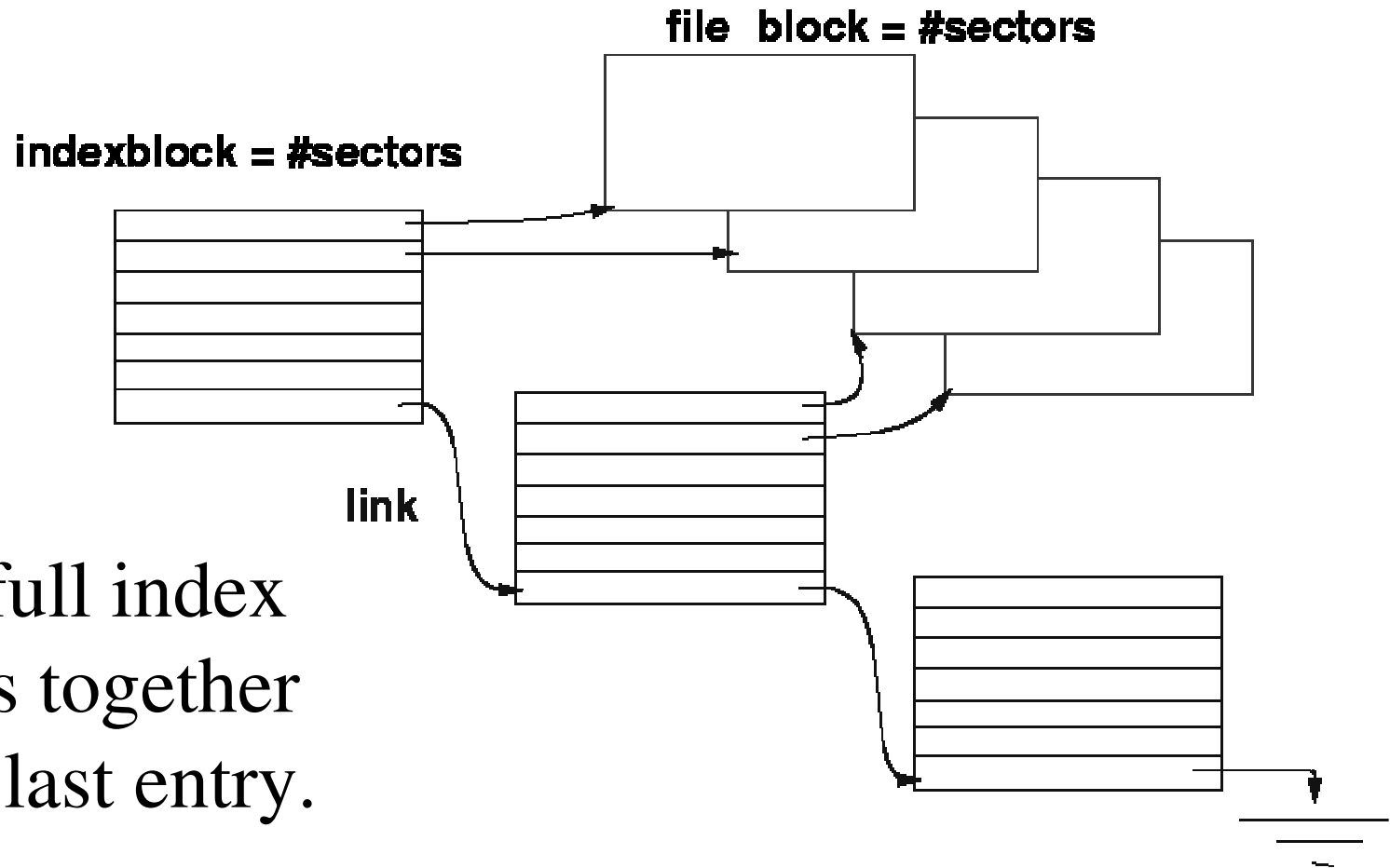
Indexed Allocation

File can be extended by rewriting a few blocks
and index block

Requires extra space for index block, possible
wasted space

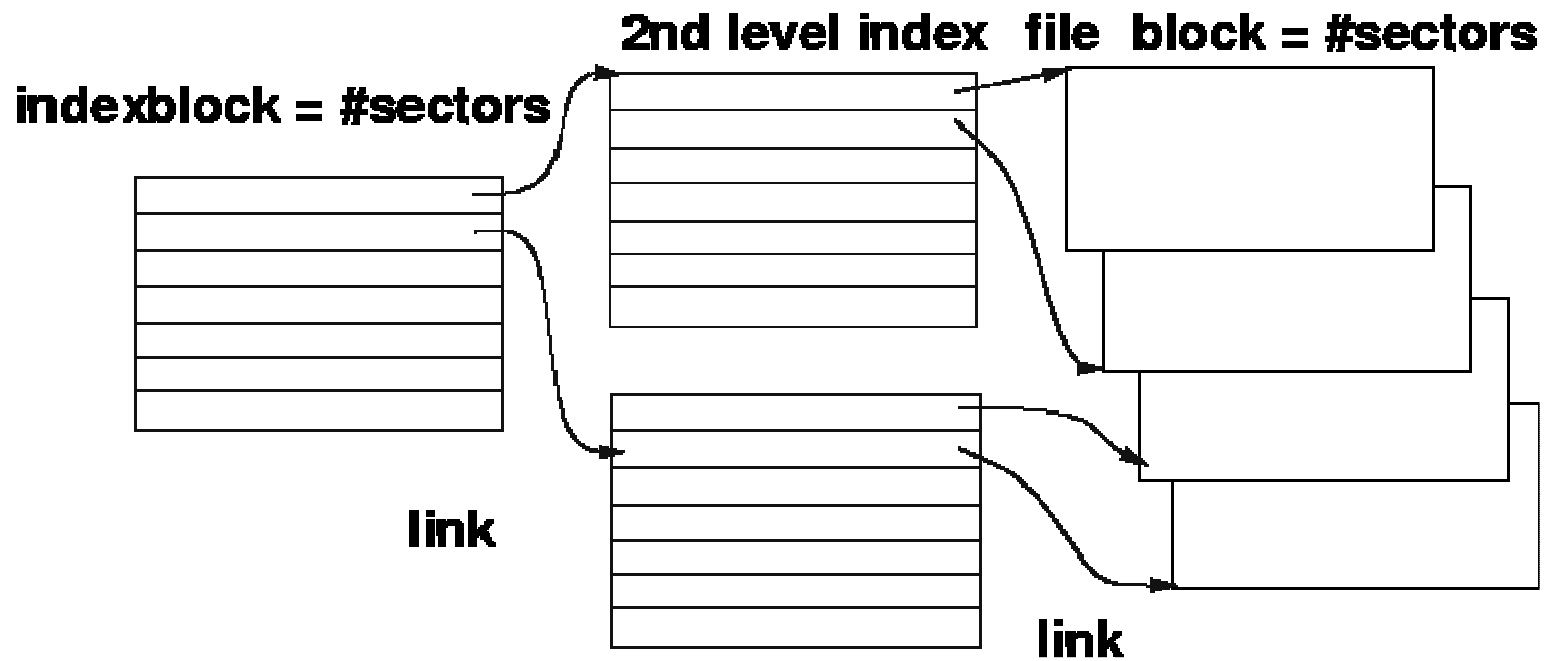
Extension to big files issues

Linked Indexed Files



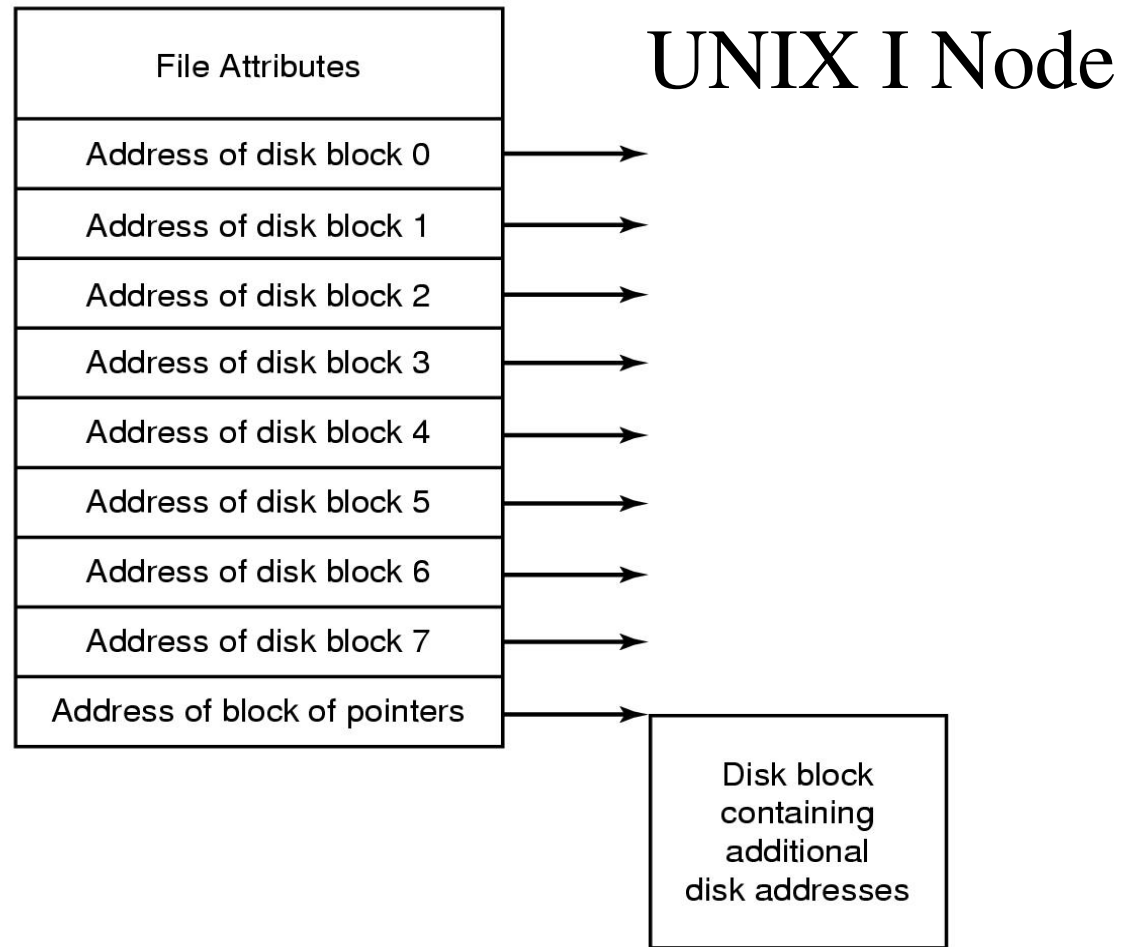
Link full index blocks together using last entry.

Multilevel Indexed File



Multiple levels of index blocks

Implementing Files (4)



Discussion

What is the maximum size of a file?

How many files can be put on a disk?

How do you make disk access fast?

How do you grow and shrink files efficiently?

How are files systems made expandable?

Answers

What is the maximum size of a file?

- 2^{32} blocks

How many files can be put on a disk?

- Each inode is 4096 bytes. An inode must fit in one integer.

How do you make disk access fast?

- Keep a cache. Account for seek, rotation. Make mapping from disk address to logical address not require too many disk reads – use power of trees.

How do you grow and shrink files efficiently?

- Use another level of indirection

How are file systems made expandable?

- Allow multiple file systems to be mounted