Operating Systems: File Management

File management

- Why talk about file management?
  - Files are the central element to most applications
  - The File System is one of the most important parts of the OS to the user
    - Some users tend to think the OS is ONLY the file system!

File management

- Desirable properties of file (systems):
  - Long-term existence
  - Must be able to store very large amounts of information
  - Sharable between processes
  - Want way to information in a way that is easy to access
    - We will need structure
  - Guarantee that data in the file is valid
  - Minimize lost or destroyed data

File management

- Basic functions of a file system
  - Present a logical (abstract) view of files to users by hiding physical details
  - Facilitate the sharing of physical I/O devices
  - Optimize the usage of these I/O devices
  - Provide protection mechanisms for data being transferred or managed by I/O devices

File management

- File management system consists of system utility programs that run as privileged applications
- Concerned with secondary storage
- Usually provide functions like:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write

File management

- Provides services to users and applications in the use of files
- File structure at user level usually given by the following terms:
    - Field
    - Record
    - File
    - Database

Fields and Records

- Fields
  - Basic element of data
  - Contains a single value (such as name or date)
  - May be fixed length or variable
    - If variable, separated by demarcation fields
- Records
  - Collection of related fields, can be fixed or variable length
    - Has a field with a length attribute
  - Treated as a unit
  - Examples: Employee record

Files

- File
  - Collection of similar records
  - Treated as a single entity by users and applications
    - So has a filename
  - Usually the smallest unit with access control restrictions
    - Though this can be down to individual fields

Database

- Collection of related data
- Explicit relationships among data elements and fields
- May contain one or more files
- May be managed by a database management system
  - In other words, could be independent of OS

Requirements for general purpose system

- Each user should be able to create, delete, read, write and modify files
- Each user may have controlled access to other users' files
- Each user may control what type of accesses are allowed to the users' files
- Each user should be able to set up their own structure of files
  - In a form appropriate to their needs
- Each user should be able to move data between files
- Each user should be able to back up and restore data in case of damage
- Should provide a convenient method to access files (symbolic names)

Typical organization

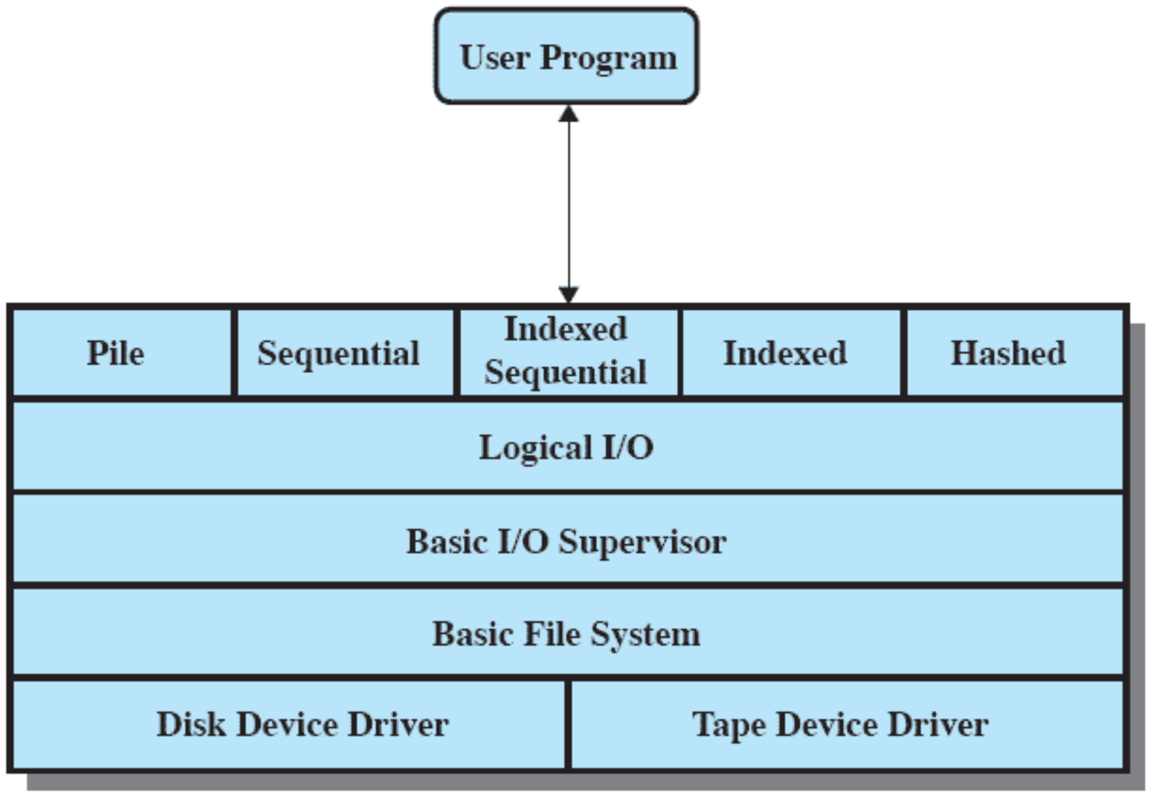


Figure 12.1 File System Software Architecture

Device Drivers

- Lowest level
- Communicates directly with peripherals
- Responsible for starting I/O operations on a device
- Processes the completion of an I/O request
- For file operations, usually either disk or tape drives
- Considered part of the OS

Basic File System

- Physical I/O level, part of OS
- Primary interface with the environment outside the computer
- Deals with exchanging blocks of data
- Concerned with the placement of blocks and buffering blocks in memory
- Does not understand content of the data or structure of the files

Basic I/O Supervisor

- Responsible for all file I/O initiation and termination
- Control structures to deal with:
  - Device I/O
  - Scheduling
  - File status
- Selects and schedules I/O with the device

Logical I/O

- Enables users and applications to access records
- Provides general-purpose record I/O capability
- Maintains basic data about the file

Access method

- Closest to the user
- Reflects different possible file structures
- Provides a standard interface between applications and the file systems and devices that hold the data
- Access method varies depending on the ways to access and process data for the device

File management

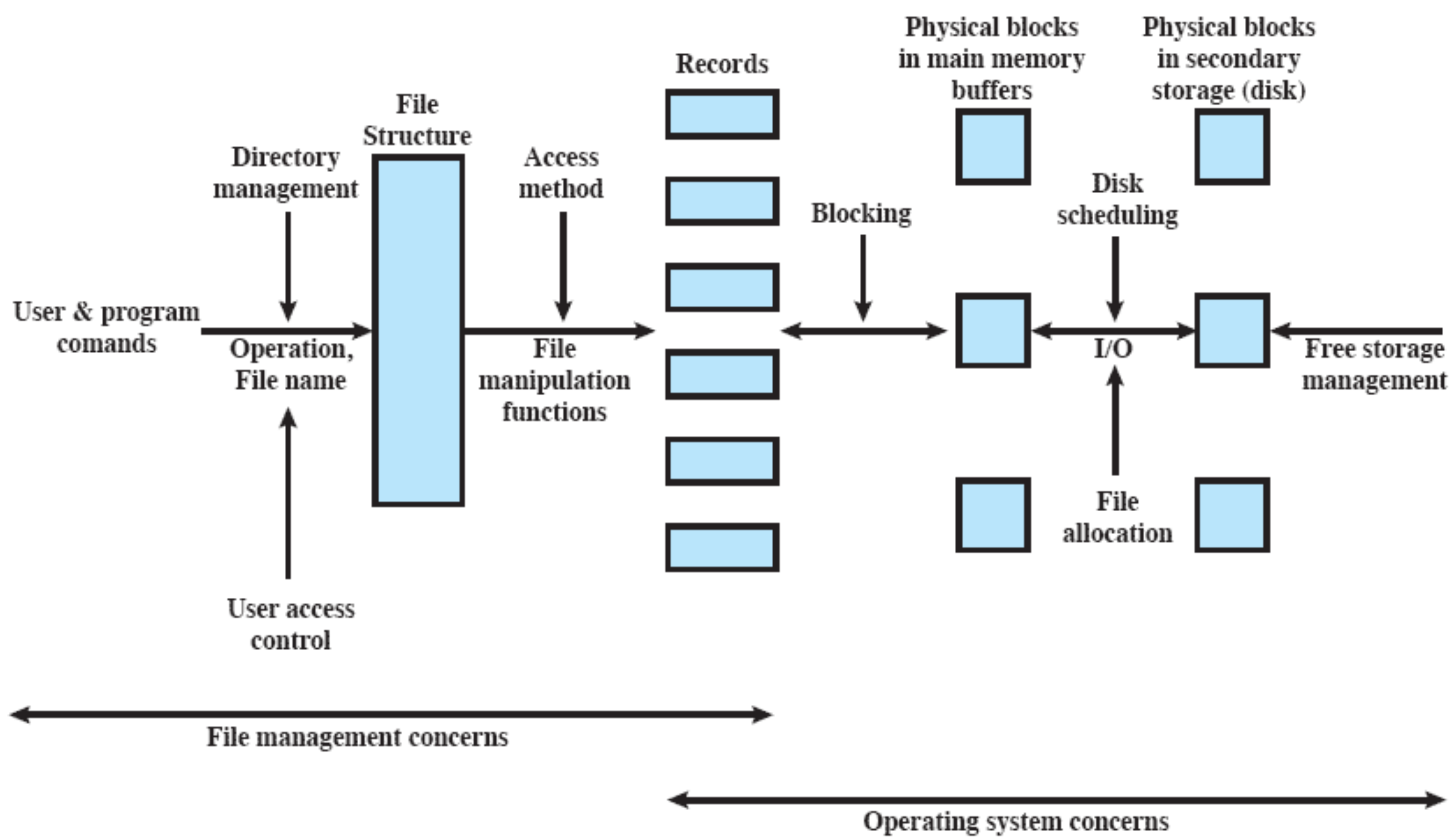


Figure 12.2 Elements of File Management

File Organization

- File organization refers to the logical structure of records
  - Not how they are physically stored
  - Will talk about physical organization later
- Determined by the *way* in which files are accessed
- So what would be desirable properties of a file organization?

File Organization

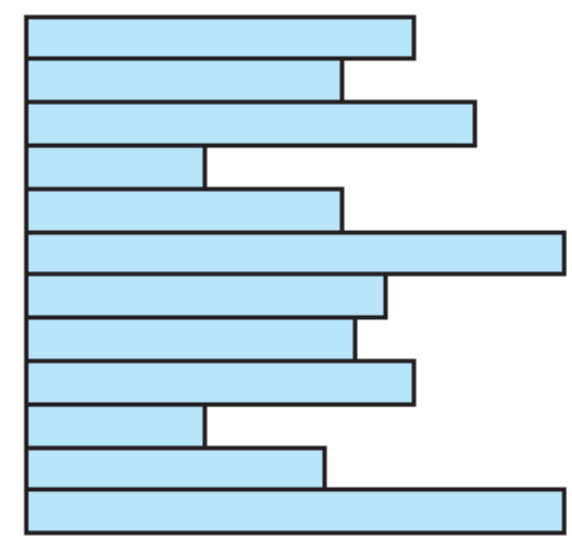
- Important criteria include
  - Short access time
  - Ease of update
  - Economy of storage
  - Simple maintenance
  - Reliability
- Priority of these concerns depends on the use (read-only CD vs hard drive)
  - They could even directly conflict

File Organization

- Many different file organizations, but usually variations of the following:
  - Pile
  - Sequential file
  - Indexed sequential file
  - Indexed file
  - Direct (or hashed) file

The Pile

- Data is collected in the order it arrives
  - No structure
- Purpose is to accumulate a mass of data and save it
- Records may have different fields
- Record access is by exhaustive search

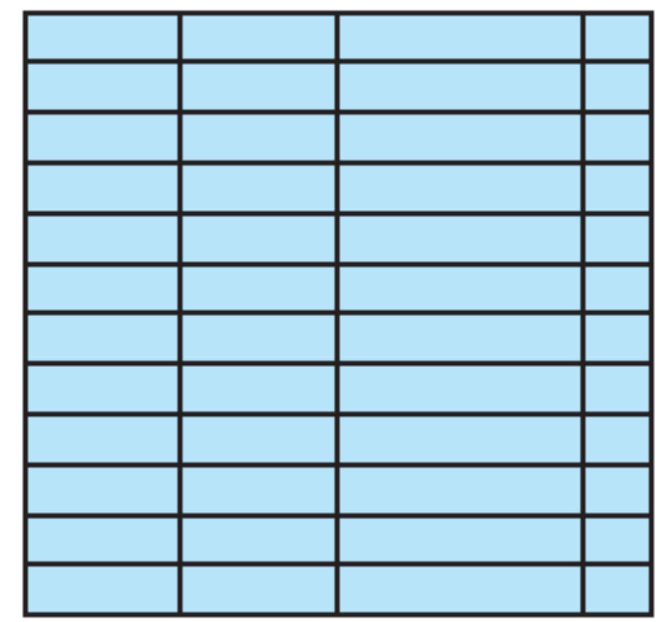


Variable-length records
Variable set of fields
Chronological order

(a) Pile File

Sequential File

- Fixed format used for records
- Records are the same length
- All fields are the same (order and length)
- Field names and lengths are attributes of the file
- Key field
  - Uniquely identifies the record
  - Records are stored in key sequence

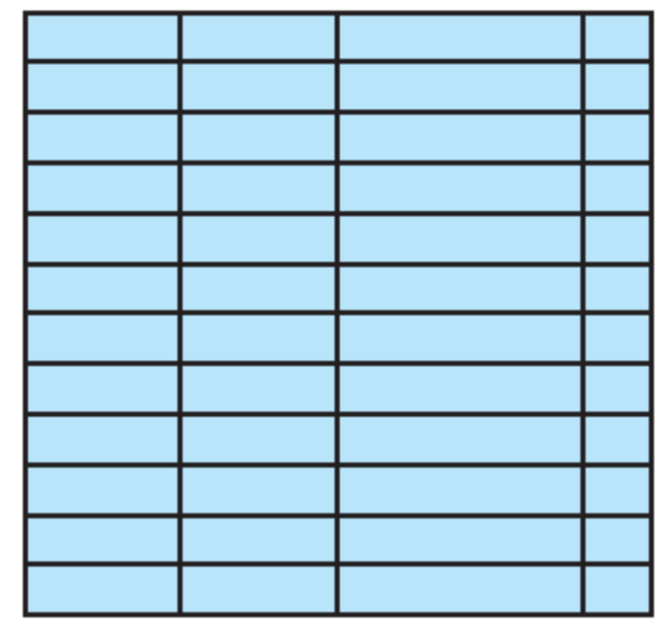


Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

Sequential File

- Performs well on batch operations
- Poorly on individual updates or additions
- Can have separate log or transaction file
  - Periodically merge this file with master
- Alternatively organize file as linked list

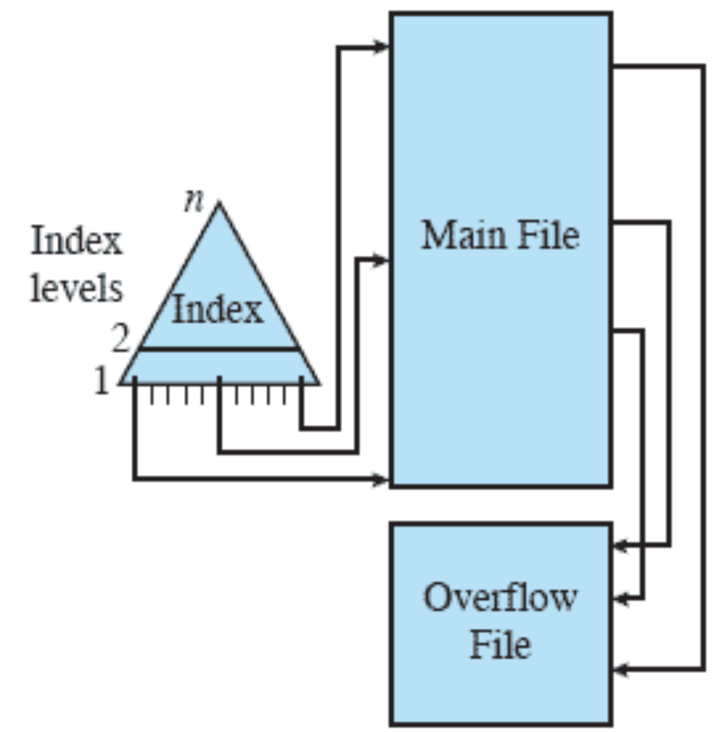


Fixed-length records
Fixed set of fields in fixed order
Sequential order based on key field

(b) Sequential File

Indexed Sequential File

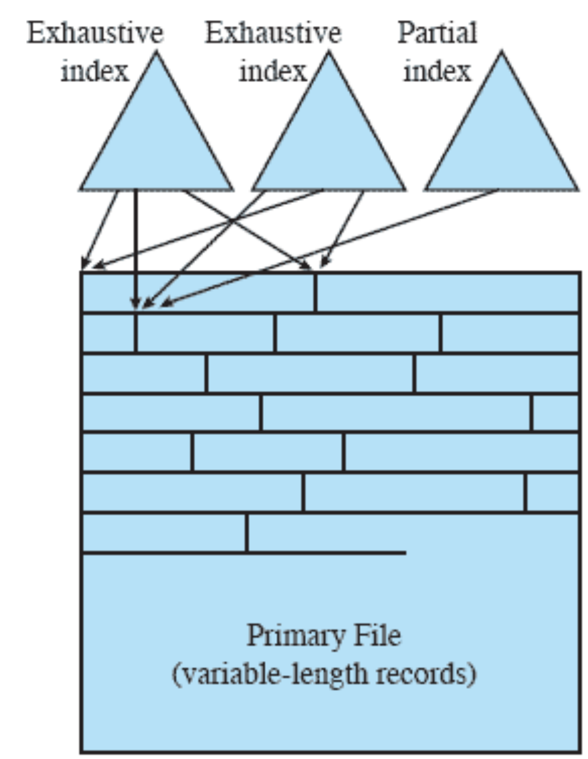
- Maintains key characteristics of sequential file
  - Records are organized in sequence by key
- Adds some additional features
  - Index to the file to support random access
  - Overflow file



(c) Indexed Sequential File

Indexed File

- Uses multiple indexes for different key fields
  - May contain exhaustive index
    - One entry for every record in main file
- When a new record is added to main file
  - All of the index files must be updated



(d) Indexed File

Direct or Hashed File

- Lets us access directly any block of a known address
- Key field required for each record
- Hash tells us where it is stored in the overall file
- Need fixed length records

B-Trees

- If file is large single sequential file of indexes does not provide rapid access
  - Need a more structured index file
- Do it in two levels
  - Original file broken into sections
  - Upper level consists of sequenced pointers to the lower-level sections
  - Can extend this to more levels for a tree structure

B-Trees

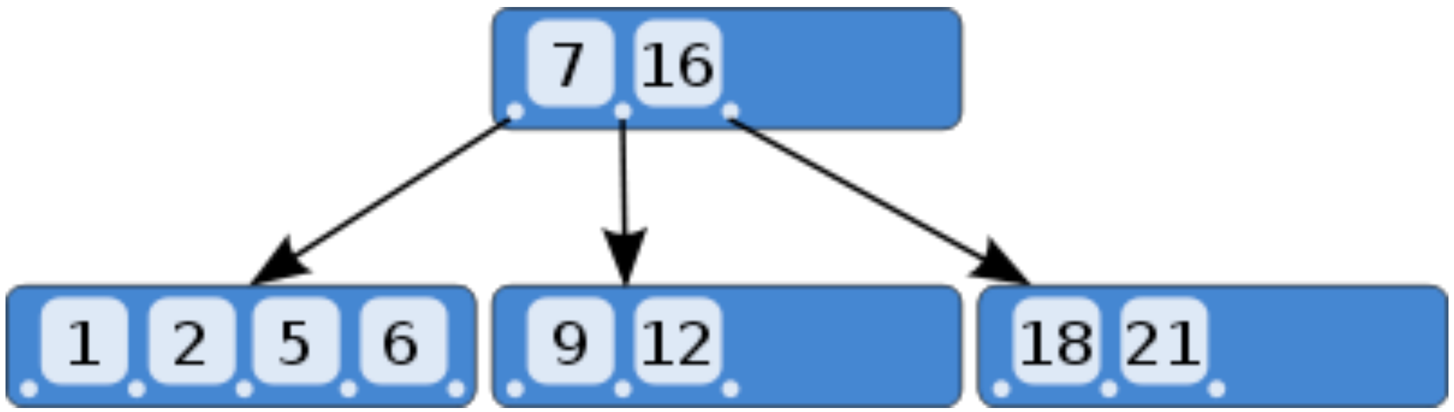
- Searching a tree structure is only efficient if we have structure
  - If some are small and others are large, time to search index is uneven
- Ensure a balanced tree structure
  - All branches of equal length
  - Self-balancing tree structure
- B-tree is standard way to organize indexes for databases and OS

B-Trees

- B-trees have the following characteristics
  - Tree consists of nodes and leaves
  - Each node contains at least one key that uniquely identifies a record
    - Can contain more than one pointer to child nodes or leaves
  - Each node is limited to a maximum number of keys
  - Keys in a node are nondecreasing
- Advantage to B-trees is very shallow, so relatively low depth

B-Trees

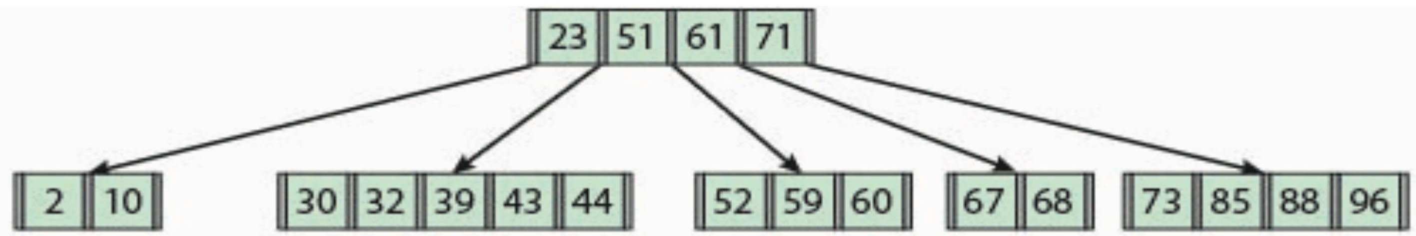
- Might see the *order* of a B-tree
  - Usually defined as the minimum number of keys in a non-root node
  - Knuth defines it as maximum number of children (max keys + 1)



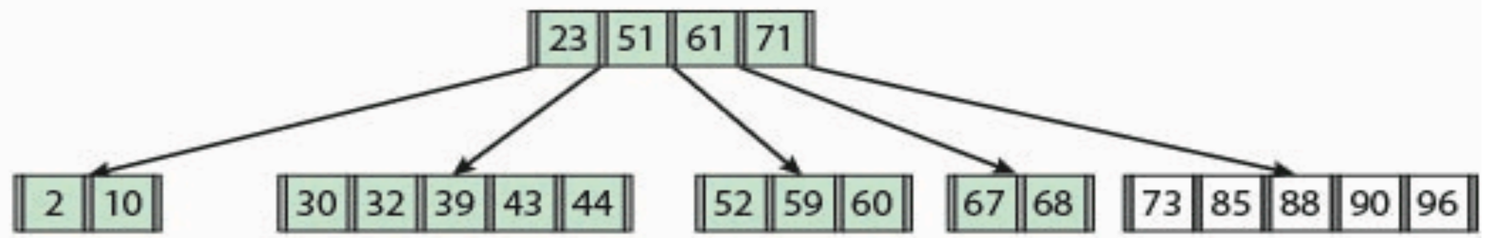
A B-tree of order 5 (Knuth)

Inserting to B-tree

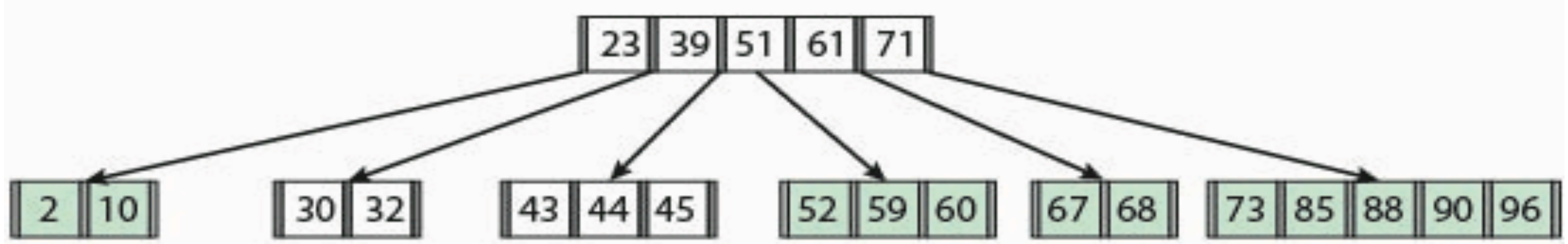
- As if inserting to binary tree, search for key
  - If room in leave node, insert it
- If no room, split it up around median key in node
  - Move median node into parent and hang the now two leaves around it
- If no room for median node in parent, split parent around its median
  - Continue the process to the root if necessary



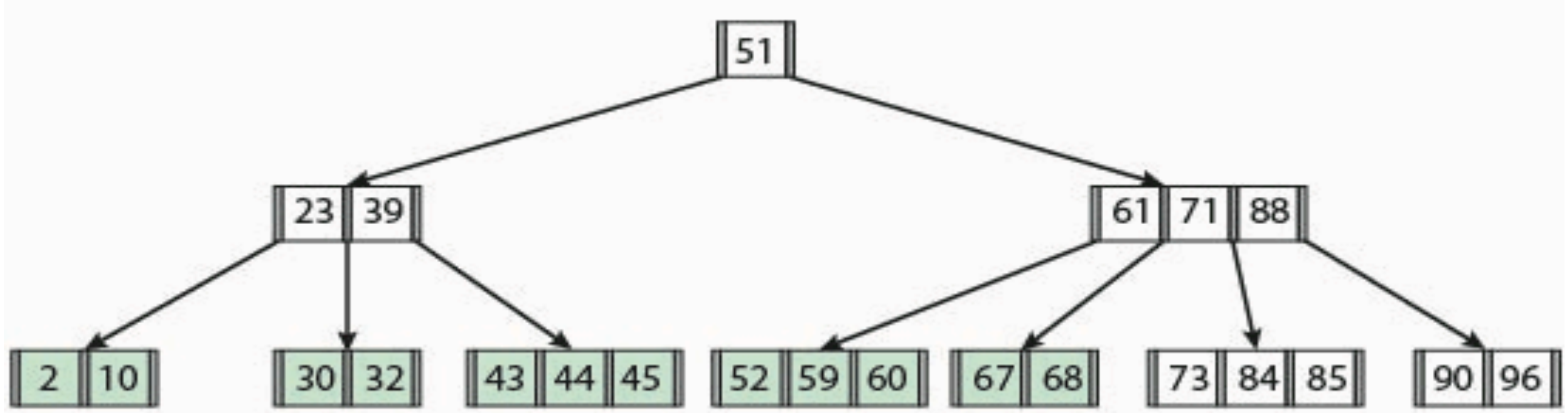
(a) B-tree of minimum degree d = 3.



(b) Key = 90 inserted. This is a simple insertion into a node.



(c) Key = 45 inserted. This requires splitting a node into two parts and promoting one key to the root node.



(d) Key = 84 inserted. This requires splitting a node into two parts and promoting one key to the root node
This then requires the root node to be split and a new root created.

Figure 12.5 Inserting Nodes into a B-tree

Inserting to B-tree

- What if we don't want to have to work our way back up?
- Can perform operations on B-tree always going down
  - Just cannot wait until we find a full parent
  - Instead, while searching to find a new key insertion point
    - Split each full node along the way

File Structure

- At user level we have records, etc
- At lowest level, have alternatives:
  - Byte sequence or stream
    - Maximum flexibility
  - Sequence of records
    - More closely matching user level
  - Tree
    - Faster searches

Blocks and records

- Records are the logical unit of access of a structured file
    - But blocks are the unit for I/O with secondary storage
        - Mismatch
- Three approaches are common:
    - Fixed length blocking
    - Variable length spanned blocking
    - Variable length unspanned blocking

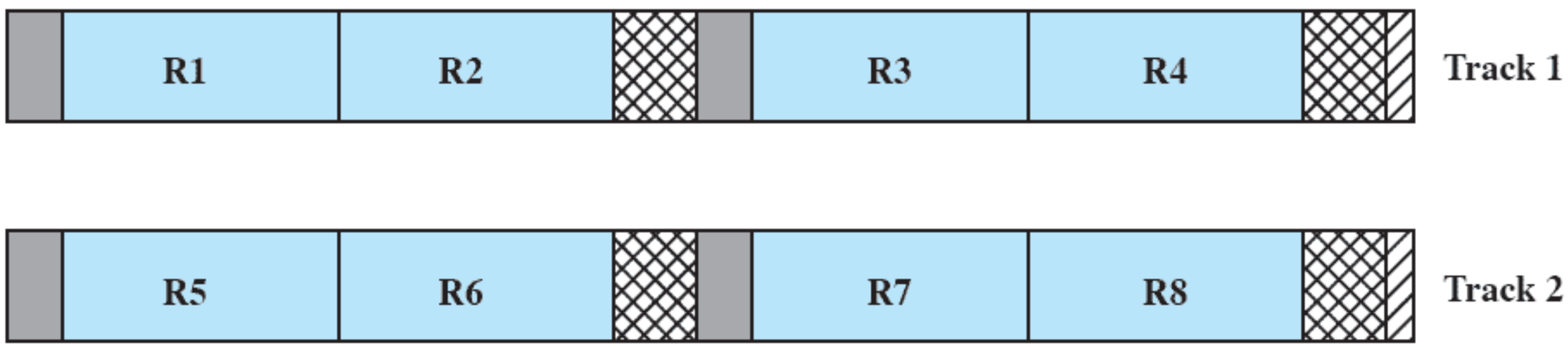
Blocks and records

- Tradeoff
  - Larger block size means more records in one read
    - Good if records are next to each other
  - Larger blocks can result in extra records that we don't need
    - Assuming not strong locality of reference
    - Also requires larger buffers

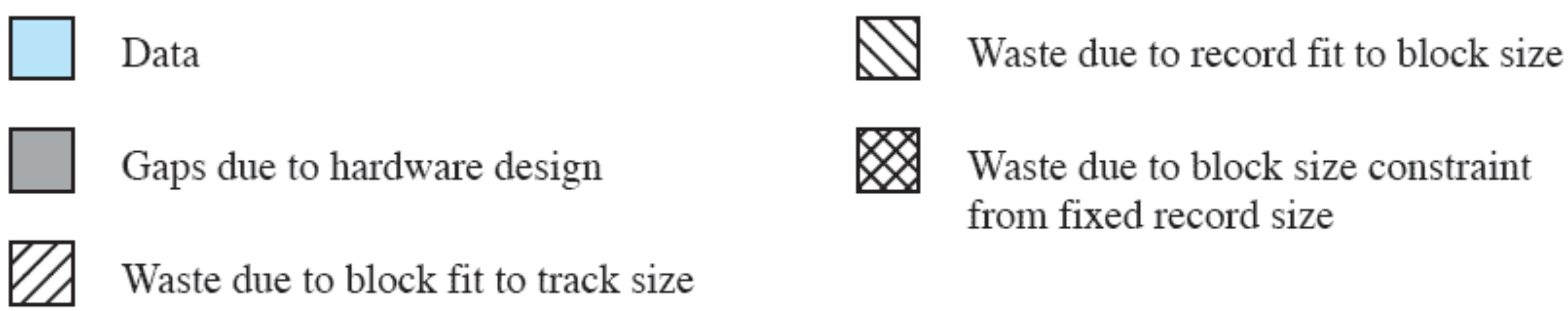
Fixed Blocking

- Fixed-length records are used
  - Integral number of records are stored in a block
- Tends to be default for sequential files with fixed length records
- Unused space at the end of a block is internal fragmentation

Fixed Blocking



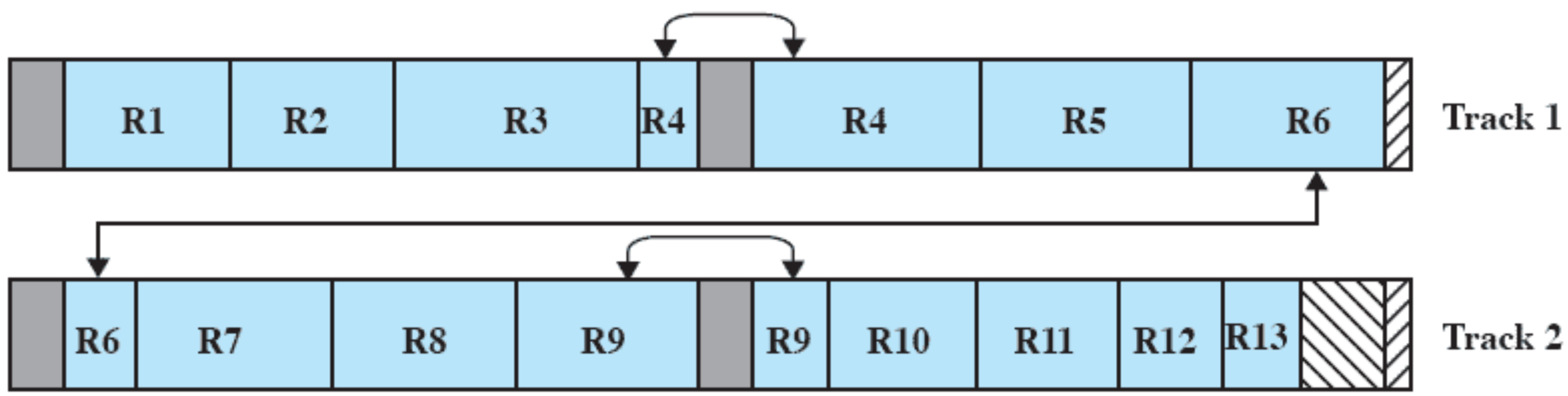
Fixed Blocking



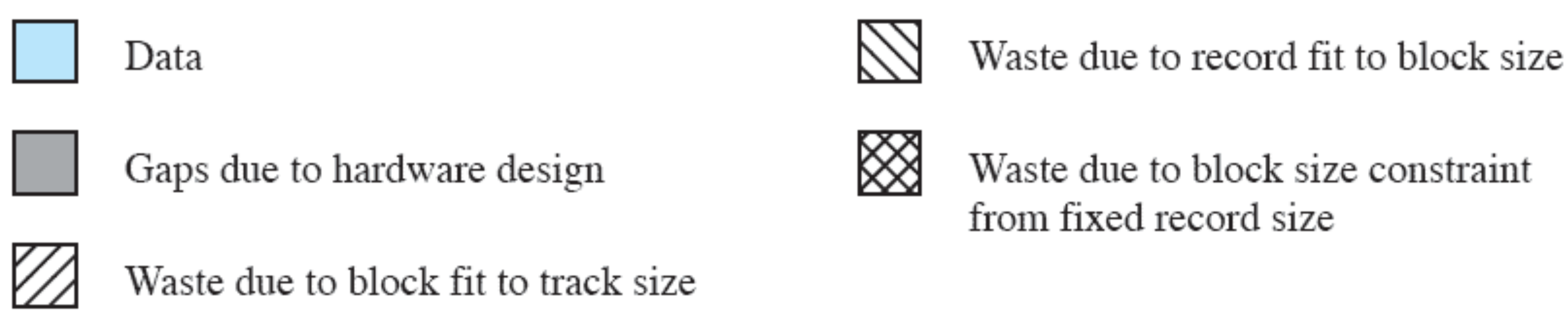
Variable Length Spanned Blocking

- Can use variable-length records
  - Could then pack into blocks with no unused space
- Would need to allow records to span multiple blocks
  - At end of block, need a pointer to the next one
- More efficient in terms of storage
- Difficult to implement
  - Records that span two blocks require two I/O operations
  - Difficult to update records

Variable Length Spanned Blocking



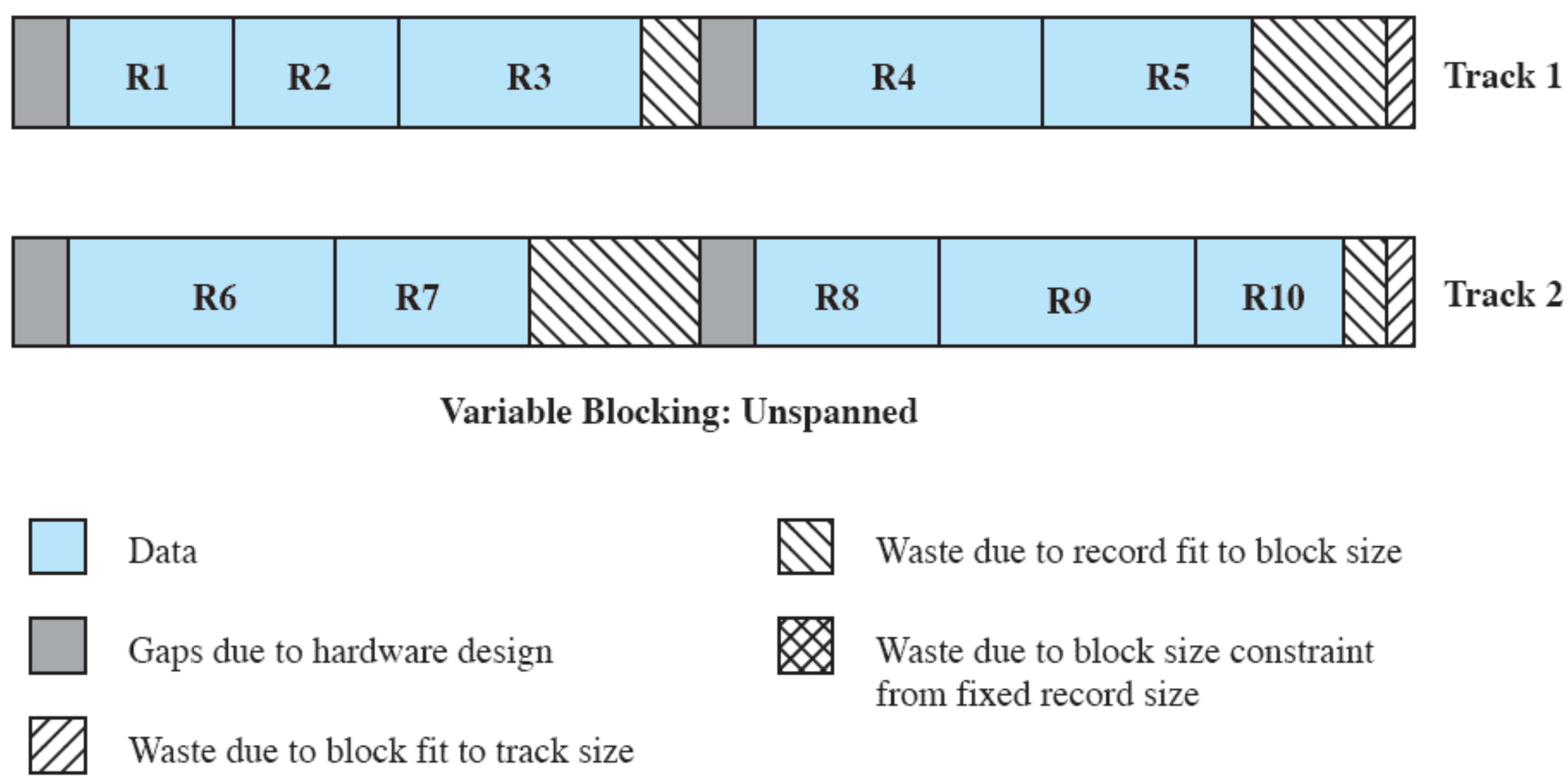
Variable Blocking: Spanned



Variable-length Unspanned Blocking

- Variable length records
  - This time do not allow them to span multiple blocks
- Wasted space in most blocks, as cannot use remainder
- Also cannot have record larger than a block

Variable-length Unspanned Blocking



Filenames

- Files usually accessed by name
  - File types should be known by the OS
    - Should not print binary files, try and compile an image, etc
    - Historically done with file extensions
    - `file` command in UNIX examines magic number in file
- Files can also be accessed by a link

File links

- Links can be either `hard` or `soft` links
- Hard links are additional alias for file
    - Two or more filenames for the same physical thing
        - If you delete one hard link (`rm`), not deleted until all hard links `rm`'ed
    - Share the exact same physical blocks
    - Can only exist on the same filesystem
    - Cannot create hard link to a directory
        - Could get stuck in endless cycle

File links

- Soft links are just a pointer to a filename
  - Treated like the file it is pointing at
    - Distinguishable from the file
  - May point to non-existent files
  - Files on other systems
  - Can only exist on the same filesystem

File types in Unix

- Regular files
  - Most common type
  - Treated as a byte stream, no kernel support of structure
- Directories
  - Binary file containing list of files in it
  - Each entry is file/inode pair
    - Used to associate nodes and directory locations
    - Data itself does not know logical organization

File types in Unix

- Character-special and block-special files
  - Allows applications to community with hardware and peripherals
  - In /dev directory
    - Kernel handles links to device drivers
  - Character-special files
    - Devices perform their own buffering, raw character stream
  - Block-special files
    - Kernel handles buffering

Secondary Storage Management

- Operating system has to allocate blocks to files
  - How do we do this?
- Two related issues
  - Space must be allocated to files
  - Must keep track of the space available for allocation

File Allocation Issues

- When a file is created, is maximum space allocated at once?
- If space is added to a file in contiguous chunks, what should be the size of the portion?
  - What data structure should be used to keep track of the file portions?
    - FAT or inode

File Allocation Issues

- Preallocation
  - Needs maximum size for file at time of creation
  - We don’t have a reliable oracle, so tough to do in practice
  - Tends to result in overestimation at best

Portion Size

- Two extremes:
  - Portion large enough to hold entire file is allocated
  - Allocate space one block at a time
- Trade-off between efficiency from the point of view of:
  - A single file
  - Overall system efficiency

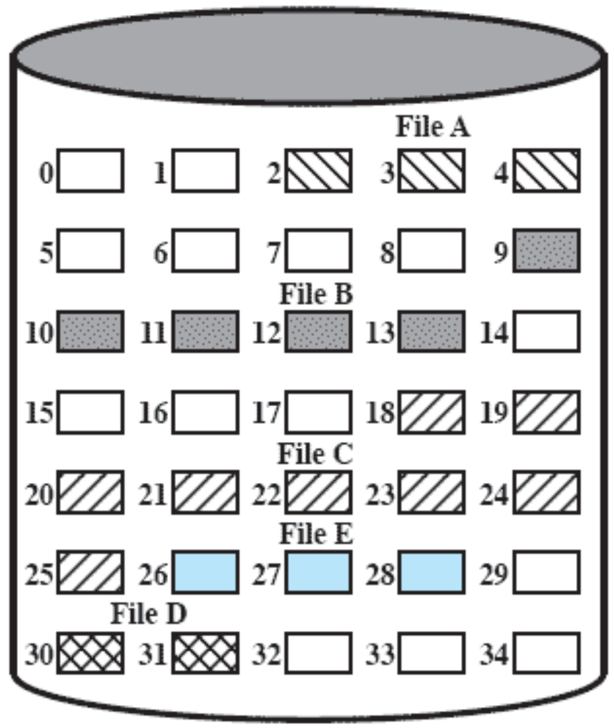
File Allocation Method

- Three methods in common use:
  - Contiguous
  - Chained
  - Indexed

Contiguous Allocation

- Single set of blocks is allocated to a file at the time of creation
- Only a single entry in the file allocation table
  - Starting block and length of the file
- External fragmentation will occur
  - Need to perform compaction

Contiguous File Allocation

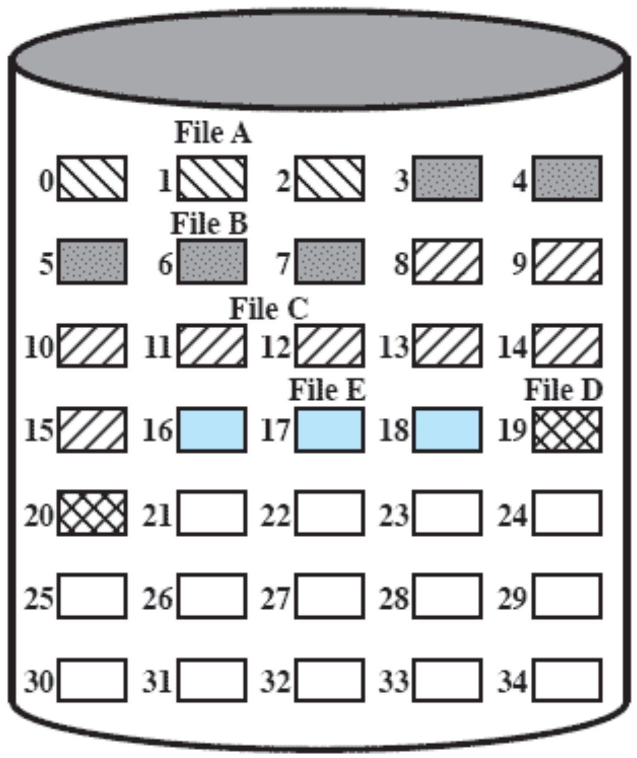


File Allocation Table

File Name	Start Block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Figure 12.7 Contiguous File Allocation

External Fragmentation



File Allocation Table

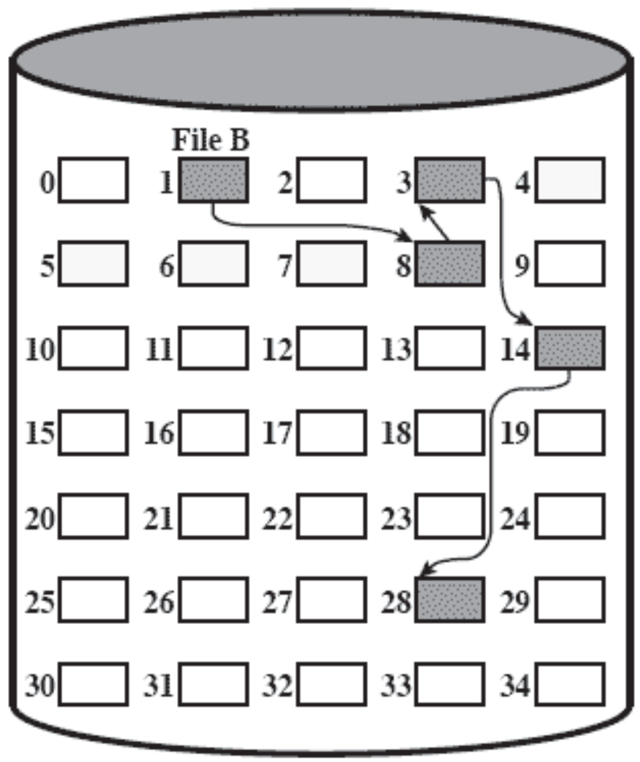
File Name	Start Block	Length
File A	0	3
File B	3	5
File C	8	8
File D	19	2
File E	16	3

Figure 12.8 Contiguous File Allocation (After Compaction)

Chained Allocation

- Allocation on basis of individual block
- Each block contains a pointer to the next block in the chain
- Only single entry in file allocation table
  - Starting block and length of file
- No external fragmentation
- Best for sequential files

Chained Allocation



File Allocation Table

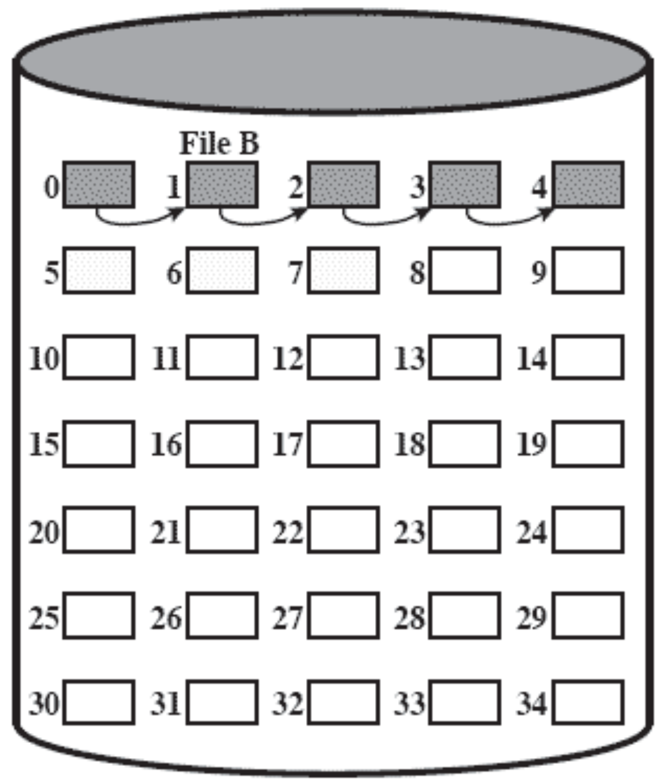
File Name	Start Block	Length
• • •	• • •	• • •
File B	1	5
• • •	• • •	• • •

Figure 12.9 Chained Allocation

Chained Allocation

- How efficient is this from a physical perspective?
- Reading requires constant seeks
- Can consolidate to help with this

Chained Allocation Consolidation



File Allocation Table

File Name	Start Block	Length
• • •	• • •	• • •
File B	0	5
• • •	• • •	• • •

Figure 12.10 Chained Allocation (After Consolidation)

Indexed Allocation

- File allocation table contains separate one-level index for each file
- The index has one entry for each portion allocated to the file
- The file allocation table contains block number for the index

Indexed Allocation Method

- Allocation may be either:
  - Fixed size blocks
  - Variable sized blocks
- Allocating by blocks eliminates external fragmentation
- Variable sized blocks improves locality
- Both cases require occasional consolidation
- Important consideration is that this index needs to be saved in a block!
  - Takes up space, which could be a problem if we want really large files
  - What happens if block size isn't large enough to hold index?

Indexed Allocation with Block Portions

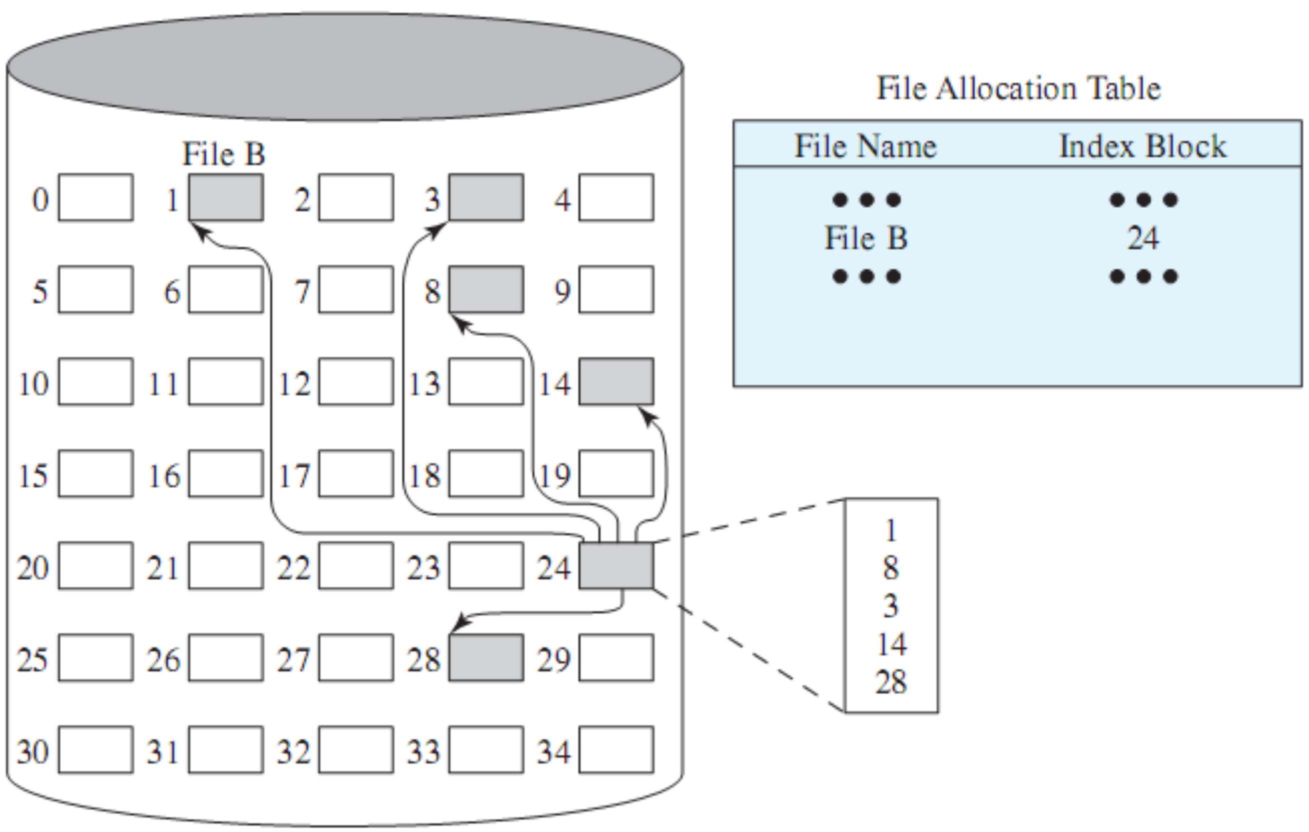


Figure 12.11 **Indexed Allocation with Block Portions**

Indexed Allocation with Variable Length Portions

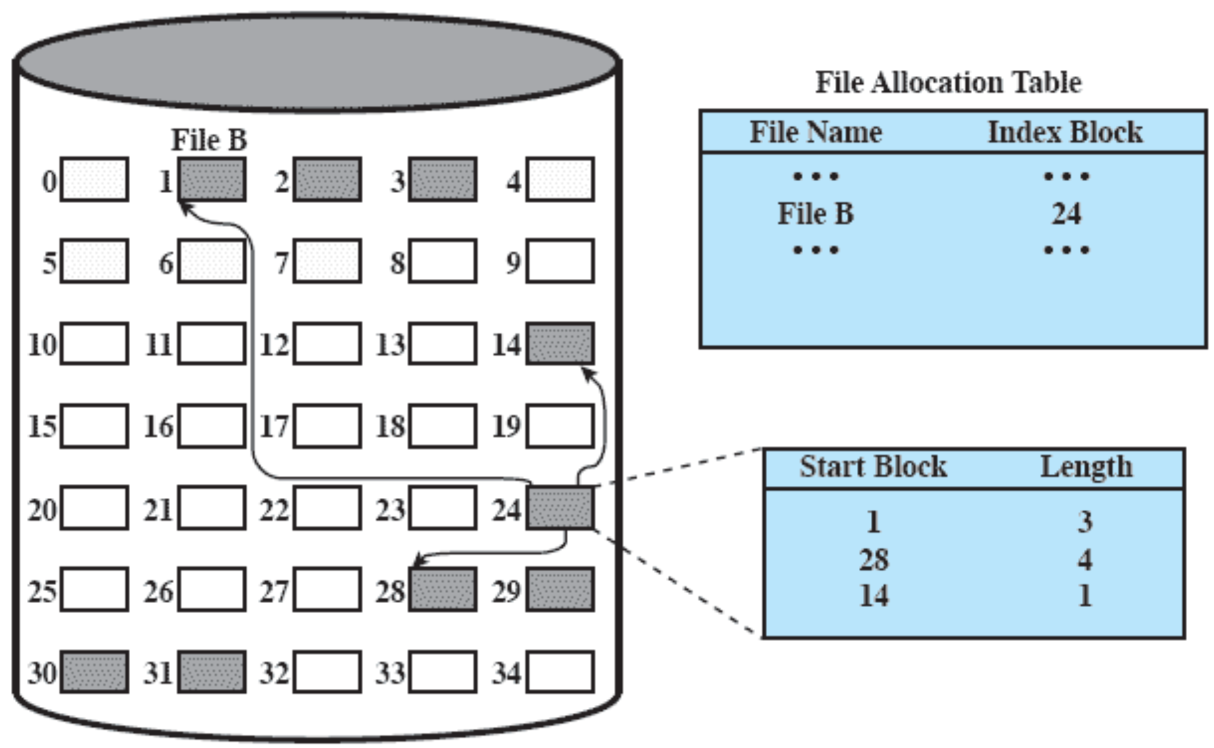


Figure 12.12 Indexed Allocation with Variable-Length Portions

Inodes

- Index node
- Control structure that contains key information for a particular file
- Several filenames may be associated with a single inode
  - But an active inode is associated with only one file
  - Each file is controlled by only one inode

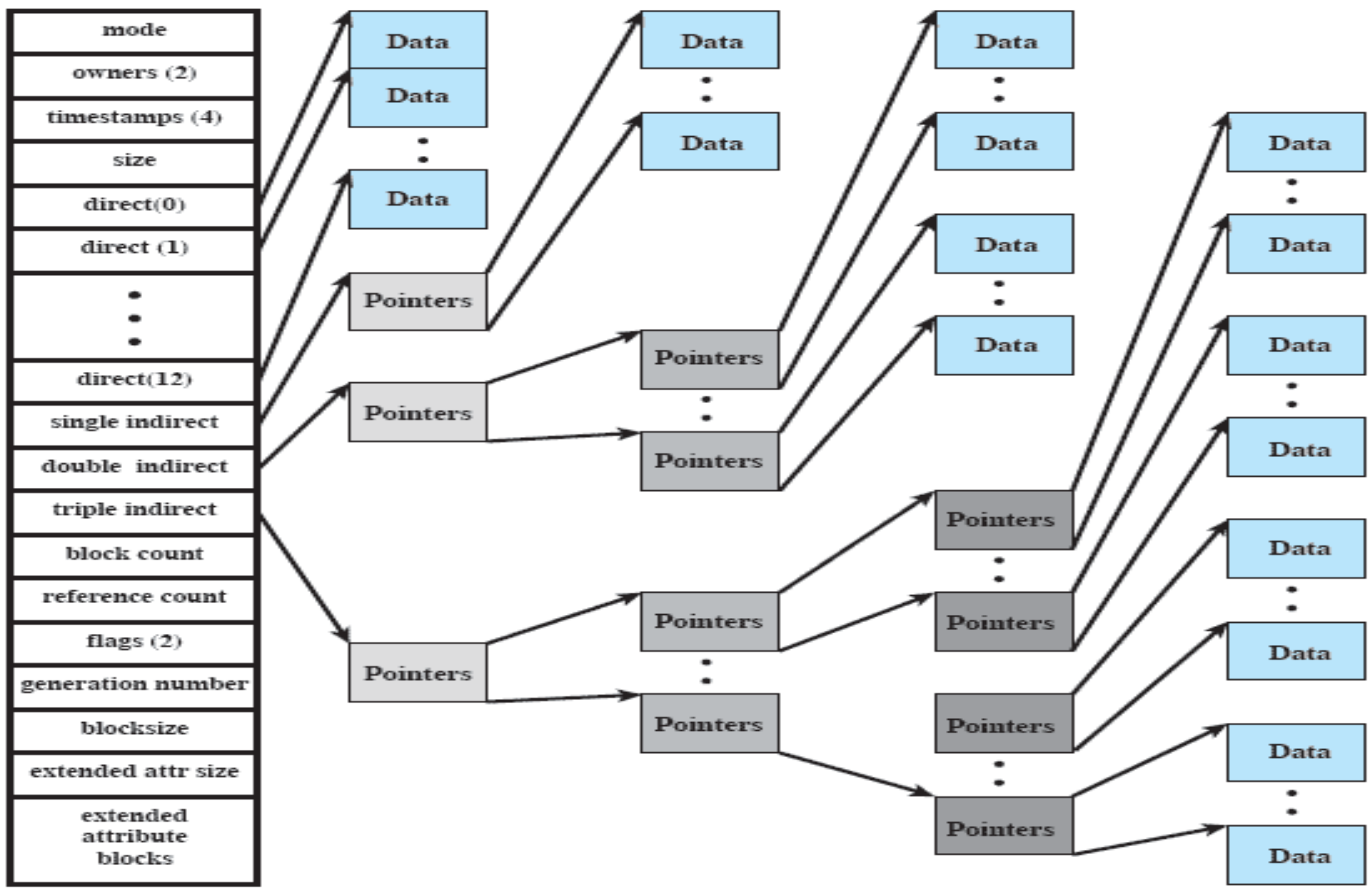
FreeBSD Inodes include:

- The type and access mode of the file
- The file's owner and group-access identifiers
- Creation time, last read/write time
- File size
- Sequence of block pointers
- Number of blocks and Number of directory entries
- Blocksize of the data blocks
- Kernel and user setable flags
- Generation number for the file
- Size of Extended attribute information
- Zero or more extended attribute entries

Inodes

- File allocation is done on a block basis
- Allocation is dynamic
  - Blocks may not be contiguous
- Index method keeps track of files
  - Part of index stored in the file inode
- Inode includes a number of direct pointers
  - Three levels of indirection

Inodes



Inode

Inodes

- File allocation is done on a block basis
- Allocation is dynamic
  - Blocks may not be contiguous
- Index method keeps track of files
  - Part of index stored in the file inode
- Inode includes a number of direct pointers
  - Three levels of indirection

Inodes

- This level of indirection determines the maximum blocks a file can contain
  - Yet still allows us to access particular blocks efficiently
- Multiple levels lets us access small files without indirection
- Inode is fixed in size so can be kept in memory for long periods of time
- Theoretical maximum file size is huge

Inodes

- Consider an example:

```
4Kbytes block size
Each block can hold 512 block addresses

Direct can access 12 blocks                               48k
Single indirect, 512 blocks                                2M
Double indirect, 512 * 512 blocks                          1G
Triple indirect, 512 * 512 * 512 blocks                  512G
```

More on file management!

- Any questions?