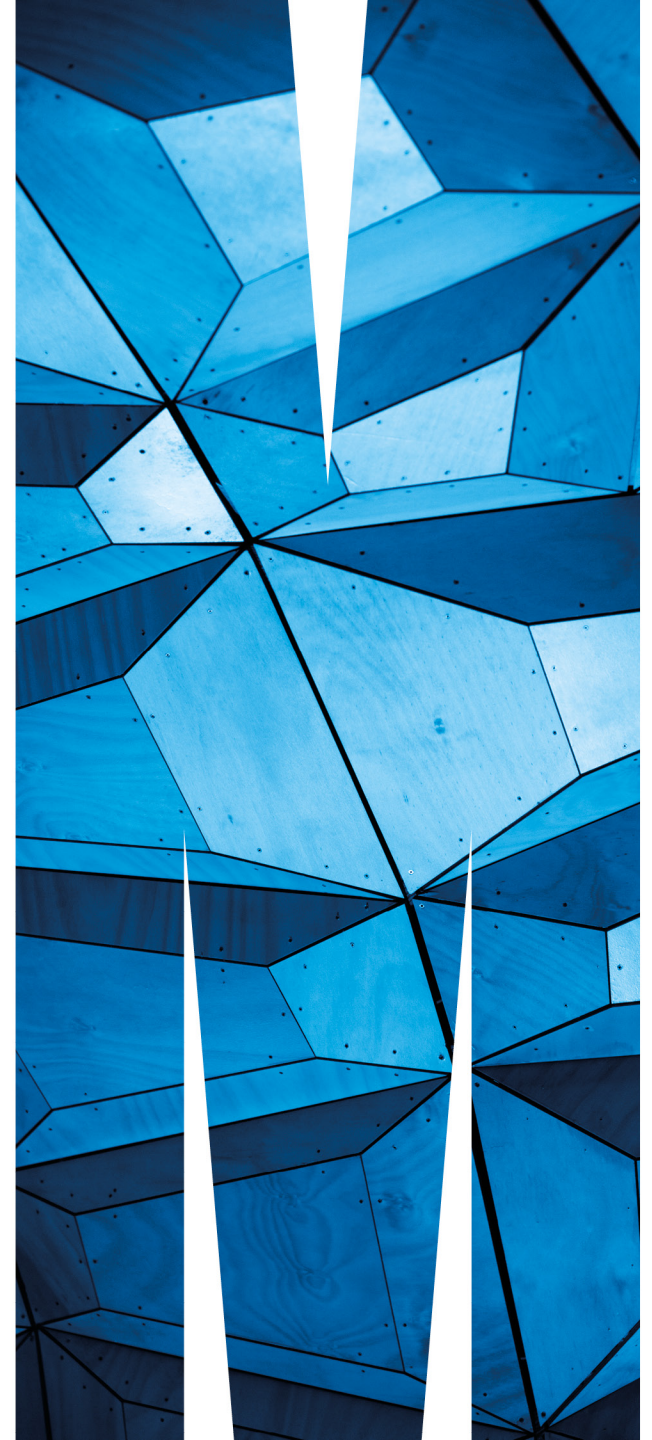


FIT2100 Semester 2 2017

Lecture 11: File Management (Reading: Stallings, Chapter 12)

Jojo Wong



Lecture 11: Learning Outcomes

- Upon the completion of this lecture, you should be able to:
 - Understand the basic concepts of **file** and **file systems**
 - Describe the principal techniques for **file organisation** and **access**
 - Discuss the principal design issues for **secondary storage management**
 - Understand the OS file systems in **Unix**

An overview for files and file systems

Files

- Data collections created by users, with desirable properties or attributes:

Long-term existence:

- File are stored on disk or other secondary storage and do not disappear when a user logs off

Sharable between processes:

- Files have names and can have associated access permissions that permit controlled sharing

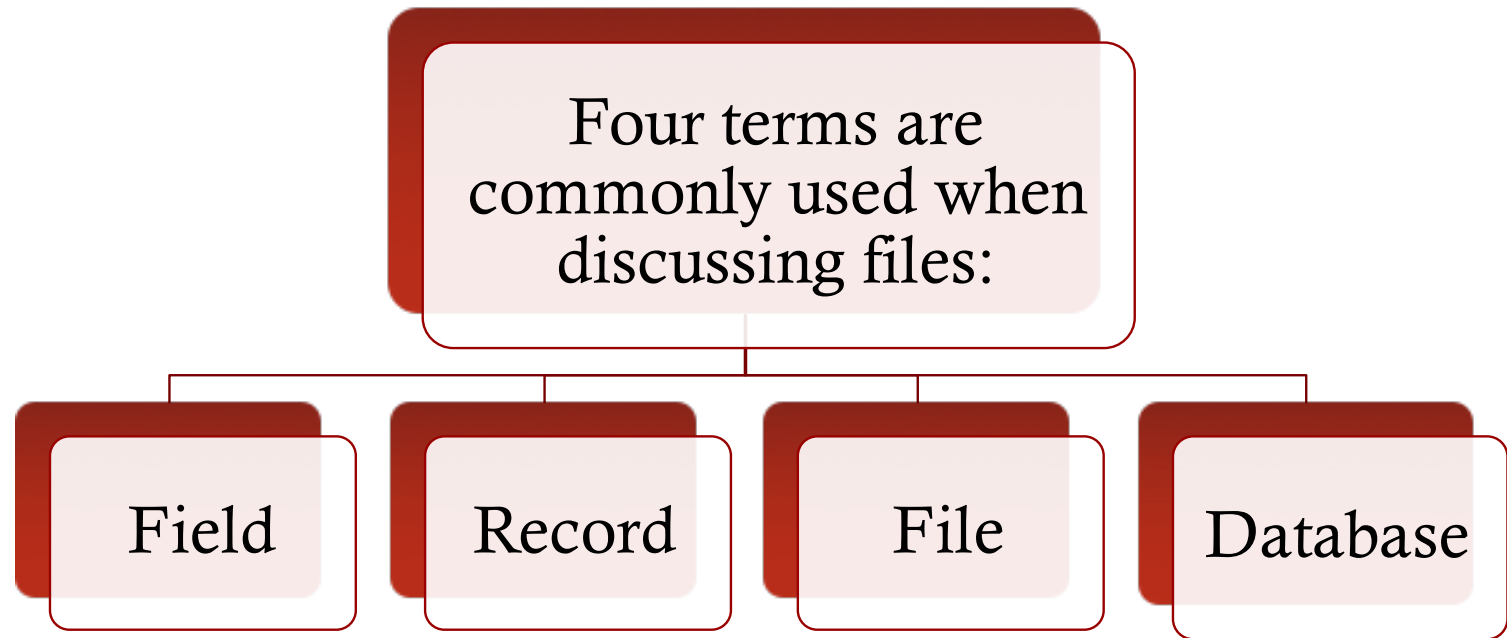
Structure:

- Files can be organised into hierarchical or more complex structure to reflect the relationships among files

File Systems

- Provide a means to store data organised as files as well as a collection of functions that can be performed on files
- Maintain a set of attributes associated with the file
- Typical operations include:
 - Create
 - Delete
 - Open
 - Close
 - Read
 - Write

File Structure



File Structure: Terminology

Field

- Basic element of data
- Contains a single value
- Fixed or variable length



Different
data types

Record

- Collection of related fields
— treated as a unit by
some application program
- Fixed or variable length

File Structure: Terminology

Often
independent
of OS

File

- Collection of similar records
- Treated as a single entity
- May be referenced by name
- Access control restrictions usually apply at the file level

Database

- Collection of related data
- Relationships among elements of data are explicit
- Designed for use by a number of different applications
- Consists of one or more types of files




MONASH
University

What are the objectives of a file management system?

File Management System: Objectives

- Meet the data management needs and requirements of the users
- Guarantee that the data in the file are valid
- Optimise system performance
- Provide I/O support for a variety of storage device types
- Provide a standardised set of I/O interface routines to user processes
- Provide I/O support for multiple users in the case of multiple-user systems
- Minimise the potential for lost or destroyed data



Response time
and system
throughput

User Requirements

- For an interactive general-purpose system, each user:

1

- should be able to create, delete, read, write and modify files

2

- may have controlled access to other users' files

3

- may control what type of accesses are allowed to the files

4

- should be able to restructure the files in a form appropriate to the problem

5

- should be able to move data between files

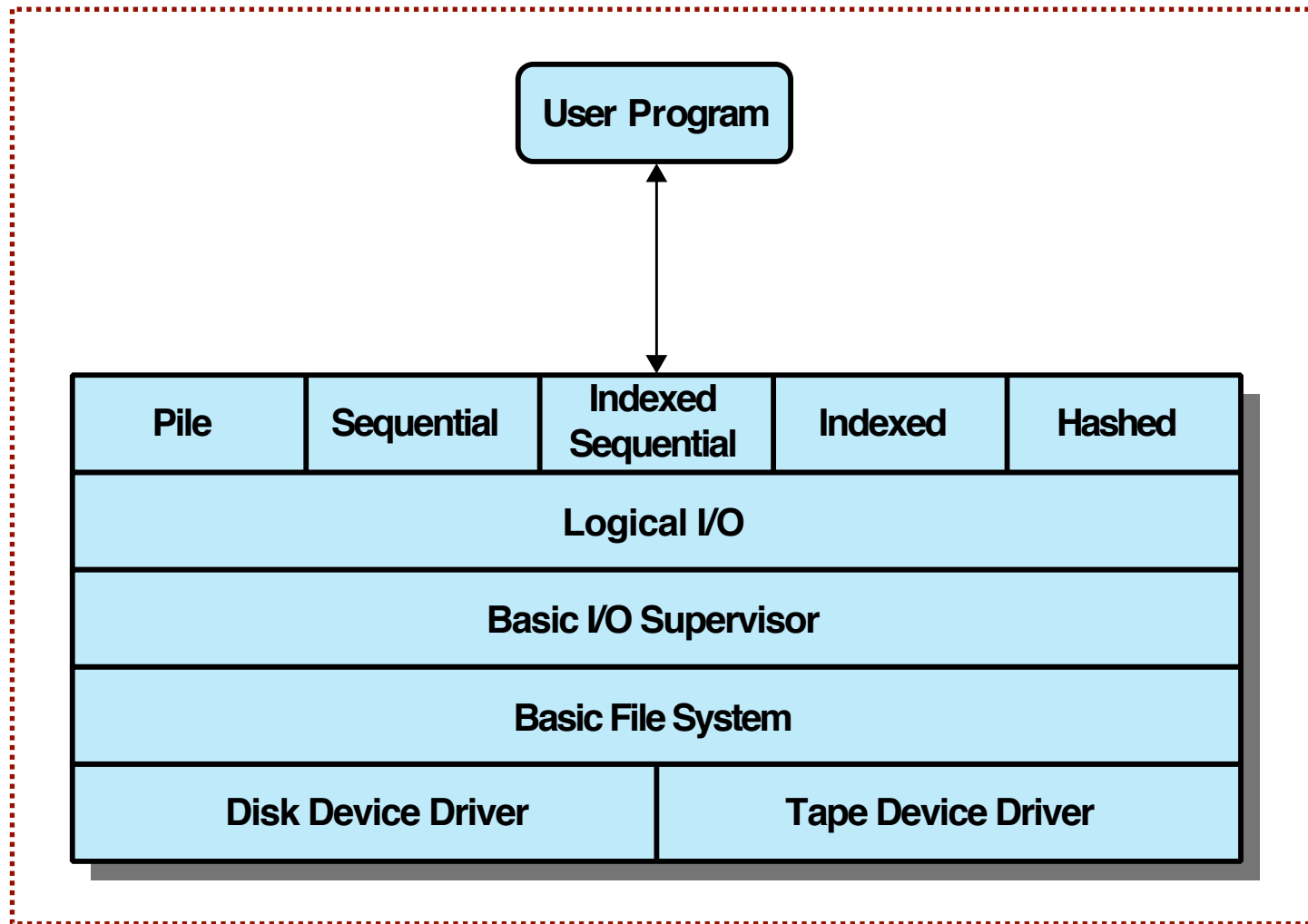
6

- should be able to back up and recover files in case of damage

7

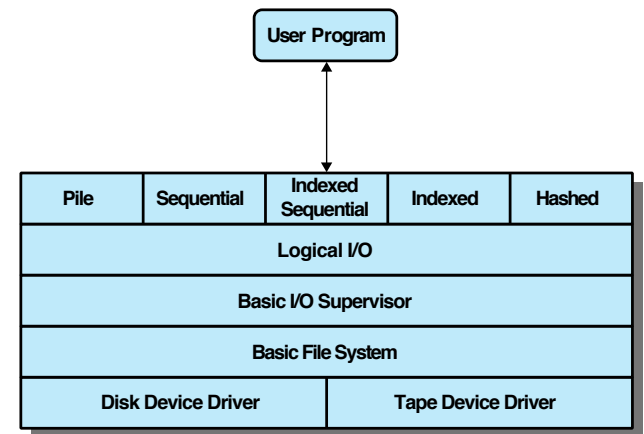
- should be able to access his or her files by name rather than by numeric identifier

File System: Software Architecture



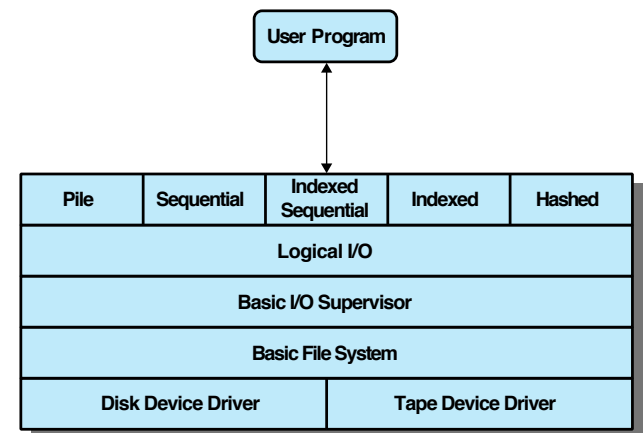
Device Drivers

- The lowest level
- Communicate directly with peripheral devices
- Responsible for starting I/O operations on a device
- Process the completion of an I/O request
- Part of the operating system



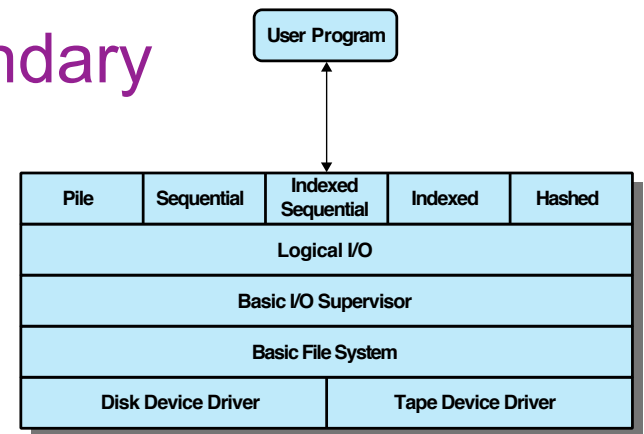
Basic File Systems

- Also referred to as the **physical I/O level**
- Primary interface with the environment outside the computer system
- Deal with blocks of data that are exchanged with disk or tape systems
- Concerned with the **placement of blocks on the secondary storage device**
- Concerned with **buffering blocks in main memory**
- Part of the operating system

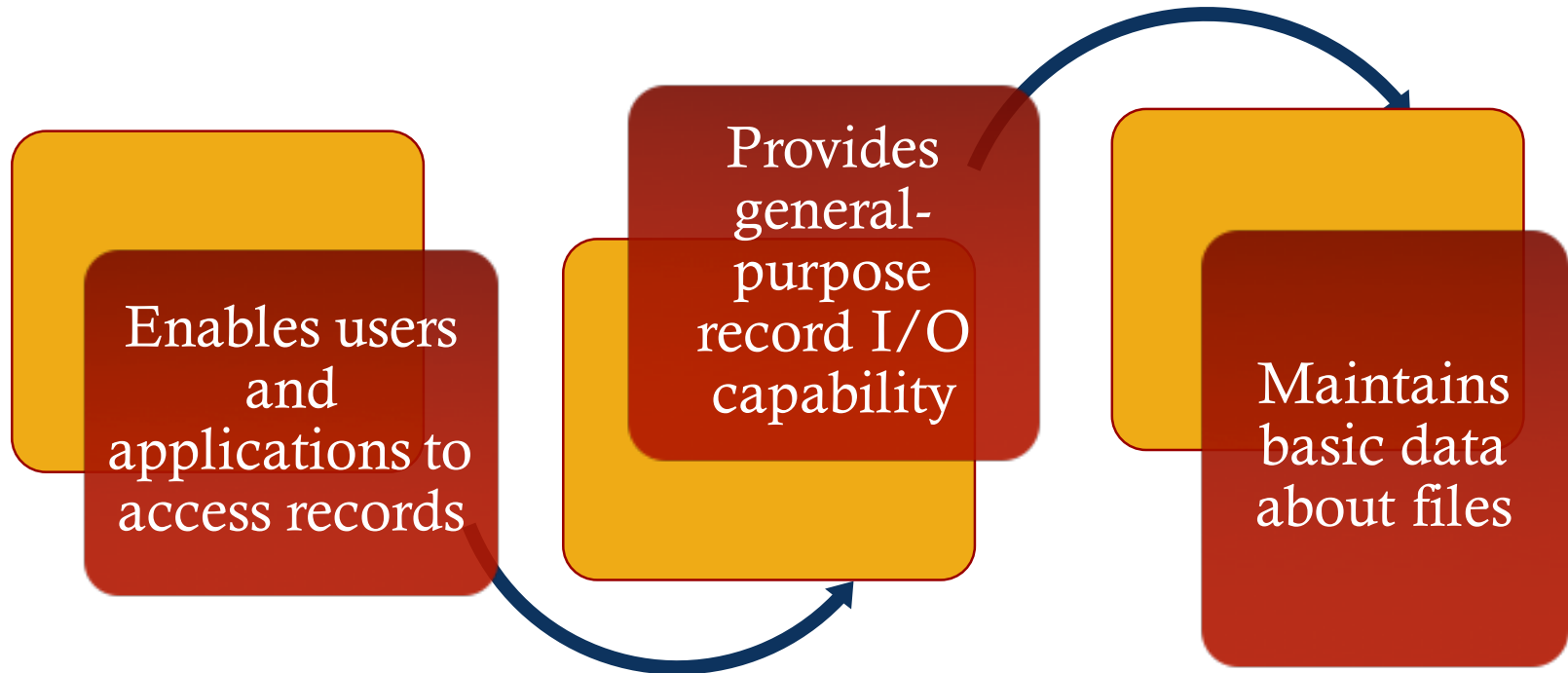


Basic I/O Supervisor

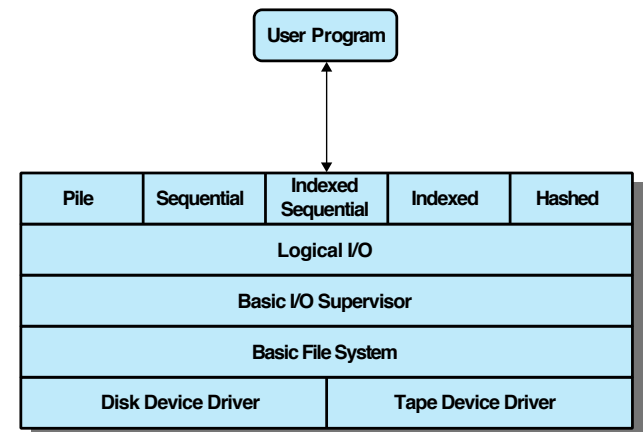
- Responsible for all **file I/O initiation** and **termination**
- Control structures that deal with device I/O, scheduling, and file status are maintained
- Select the device on which I/O is to be performed
- Concerned with **scheduling disk** accesses to optimise performance
- **I/O buffers are assigned** and **secondary memory is allocated** at this level
- Part of the operating system



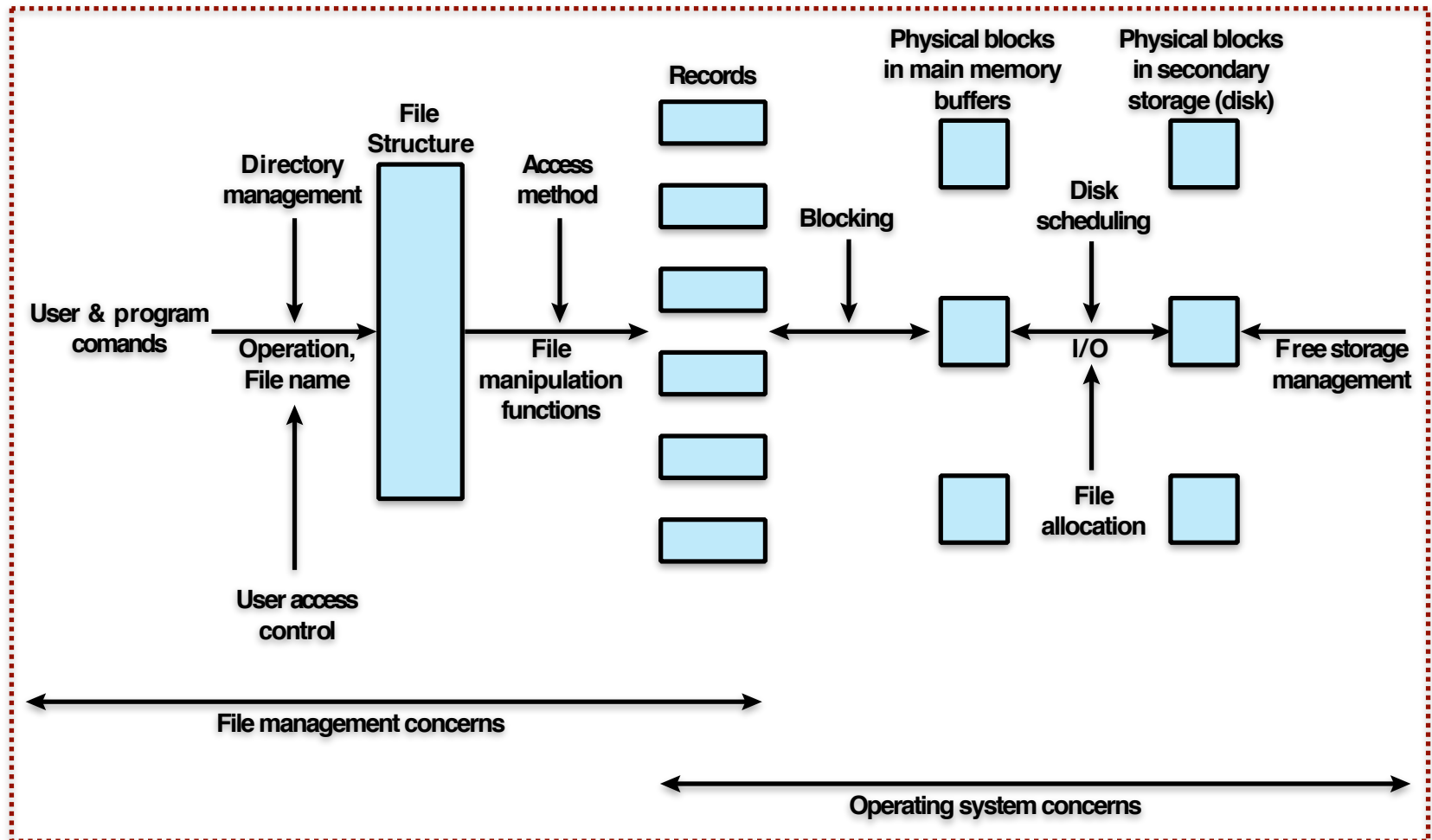
Logical I/O



- The level of the file system **closest to the user**
- Provides a **standard interface** between applications and the file systems and devices that hold the data
- Different access methods reflect different file structures and different ways of accessing and processing the data



File Management: Functions

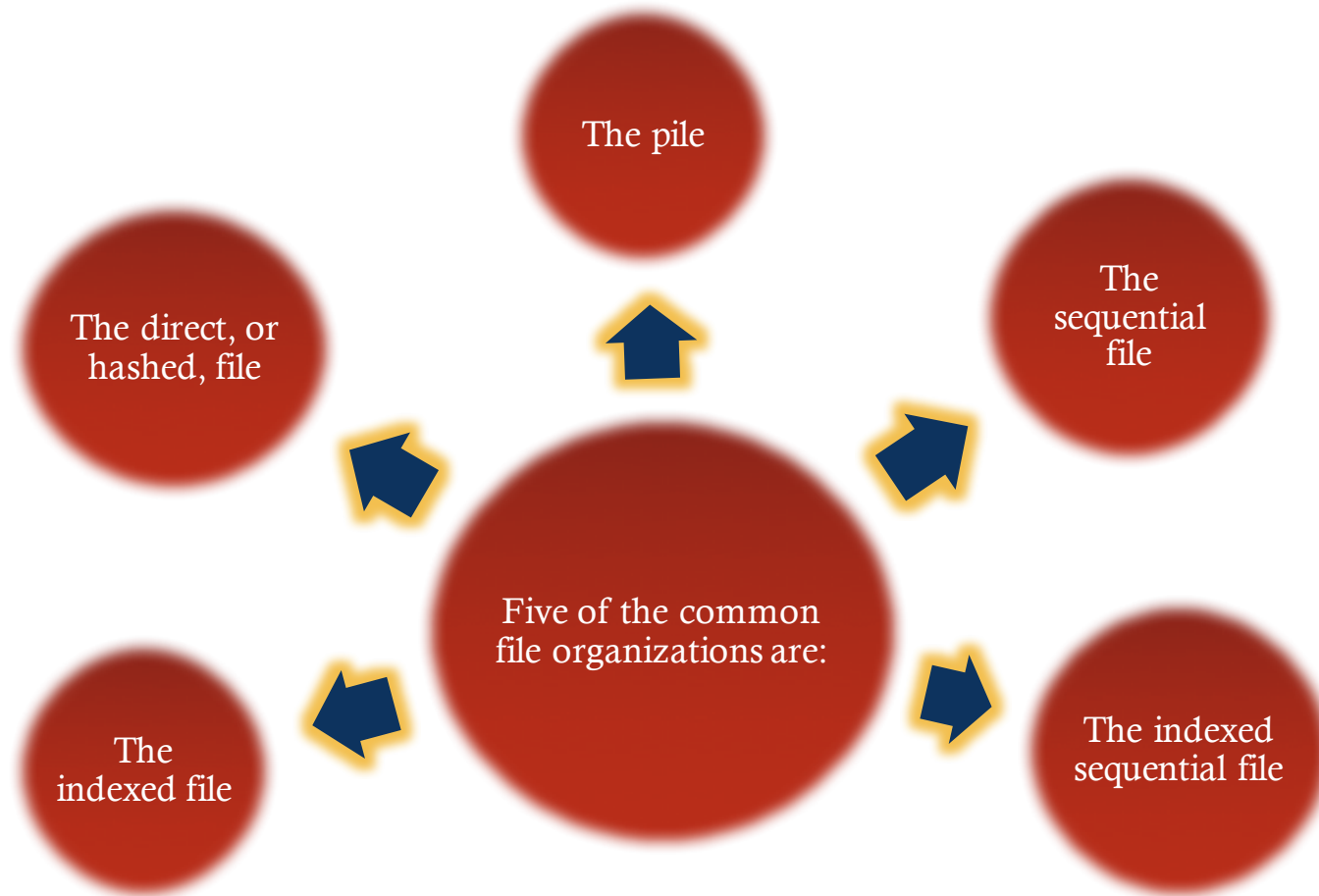


How are files organised and
accessed in a file system?

File Organisation and Access

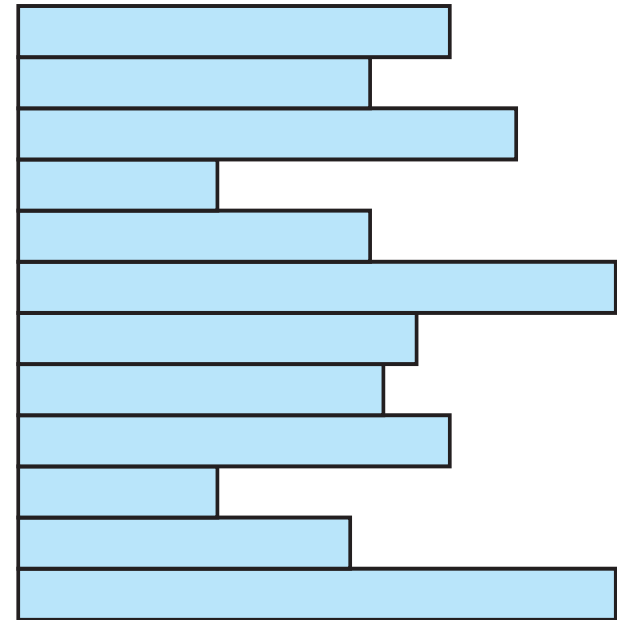
- **File organisation** is the logical structuring of the records as determined by the way in which they are accessed
- In choosing a file organisation, several **criteria** are important:
 - Short access time
 - Ease of update
 - Economy of storage
 - Simple maintenance
 - Reliability
- **Priority** of criteria depends on the application that will use the file

File Organisation: Types



File Organisation: Piles

- Least complicated form of file organisation
- Data are collected in the order they arrive
- Each record consists of one burst of data
- Purpose is simply to accumulate the mass of data and save it
- Record access is by exhaustive search



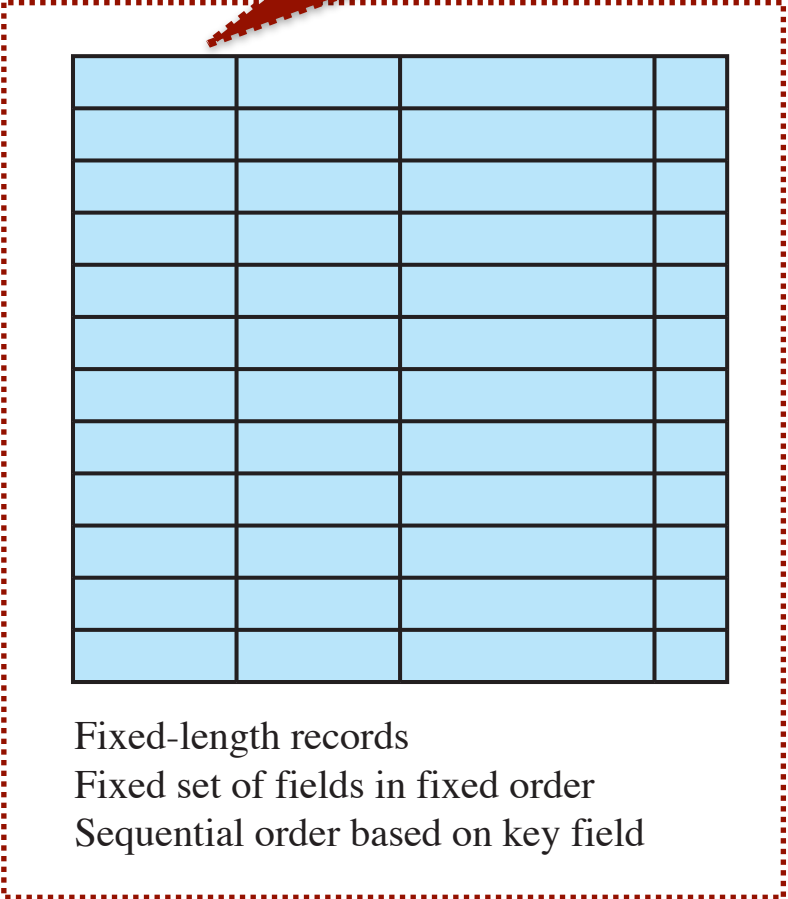
Variable-length records
Variable set of fields
Chronological order

No fixed
structure

File Organisation: Sequential Files

- Most common form of file structure
- A fixed format is used for records
- Key field uniquely identifies the record
- Typically used in batch applications
- Only organisation that is easily stored on disk

Poor performance
due to sequential
search



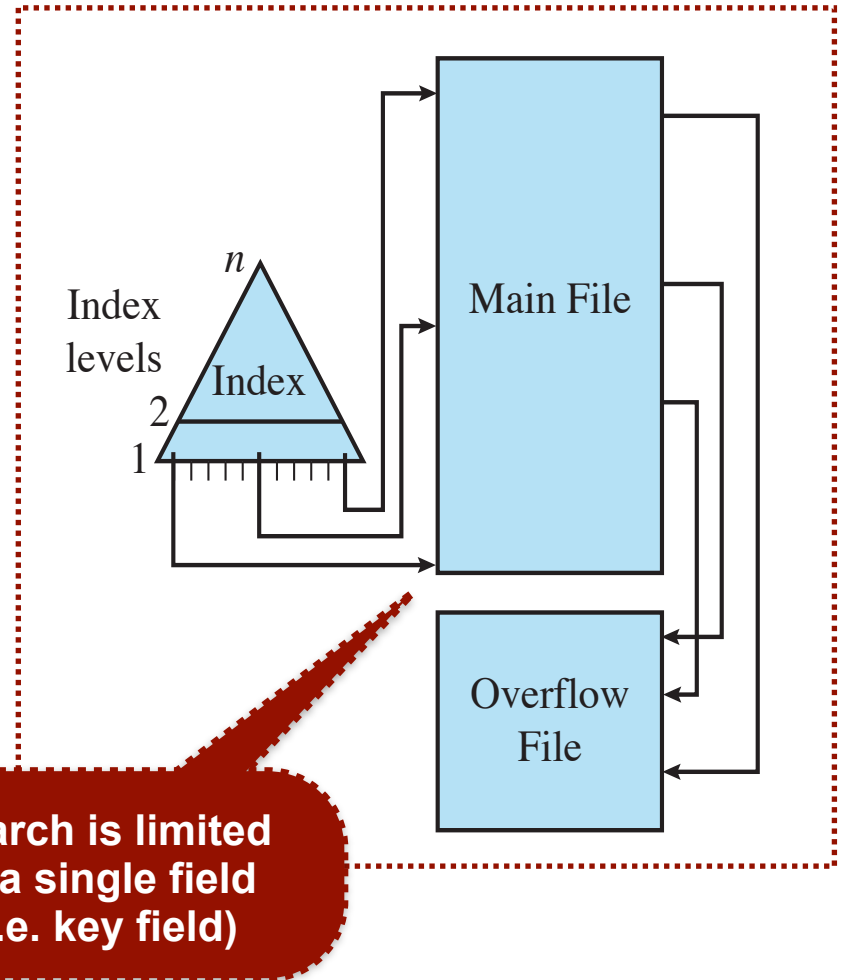
Fixed-length records

Fixed set of fields in fixed order

Sequential order based on key field

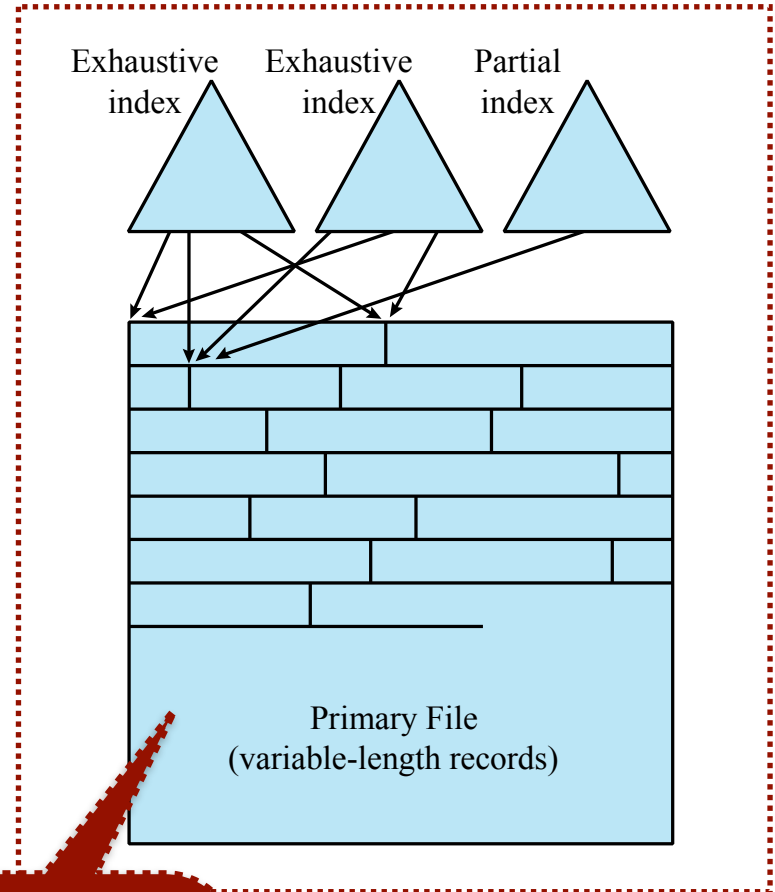
File Organisation: Indexed Sequential Files

- Adds **an index to the file** to support random access
- Adds **an overflow file**
- Greatly reduces the time required to access a single record
- Multiple levels of indexing can be used to provide greater efficiency in access



File Organisation: Indexed Files

- Records are accessed only through their indexes
- Variable-length records can be employed
- Exhaustive index** contains one entry for every record in the main file
- Partial index** contains entries to records where the field of interest exists
- Used mostly in applications where timeliness of information is critical



Search can be based on various fields

File Organisation: Direct or Hashed Files

- Access directly any block of a known address
- Makes use of **hashing** on the **key** value
- Often used where:
 - Very rapid access is required
 - Fixed-length records are used
 - Records are always accessed one at a time

What are the design issues for
secondary storage management?

Record Blocking

- **Blocks** are the unit of I/O with secondary storage
 - For I/O to be performed records must be organised as blocks
- Given the **size** of a block, three methods of **blocking** can be used:
 - Fixed-length blocking
 - Variable-length spanned blocking
 - Variable-length unspanned blocking

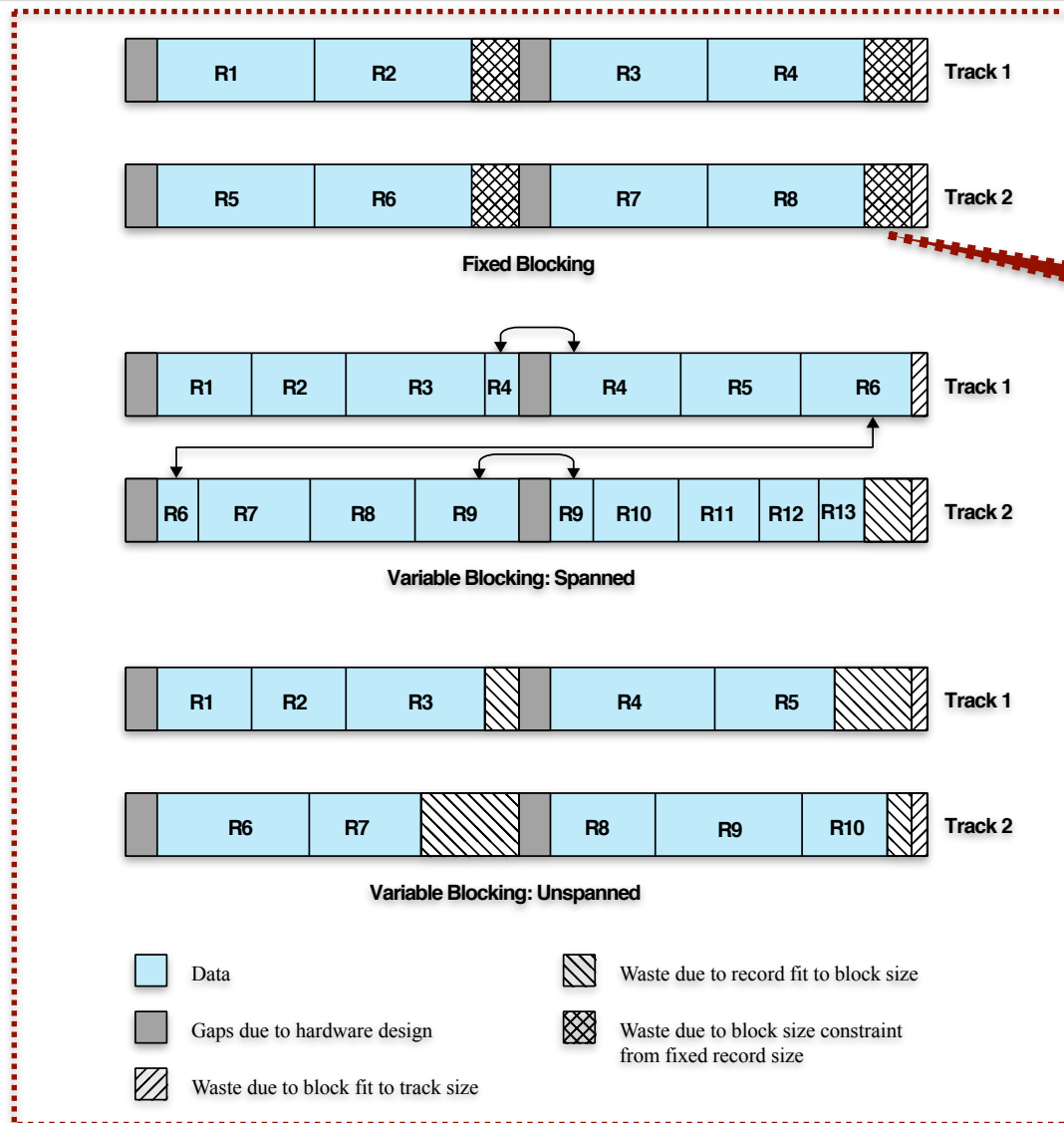
Record Blocking: Methods

- **Fixed-length blocking:**
 - Fixed-length records are used, and an integral number of records are stored in a block
 - Internal fragmentation — unused space at the end of each block
- **Variable-length spanned blocking:**
 - Variable-length records are used and are packed into blocks with no unused space
- **Variable-length unspanned blocking:**
 - Variable-length records are used, but spanning is not employed



Records may span two blocks

Record Blocking: Methods



internal fragmentation

File Allocation

- On secondary storage, a file consists of a collection of **blocks**
- OS or file management system is responsible for allocating blocks to files
- Approaches taken for **file allocation** may influence approaches taken for **free space management**
- Space is allocated to a file as one or more **portions** (a contiguous set of allocated blocks)
- **File allocation table** (FAT): data structure used to keep track of the portions assigned to a file

A block size
or the entire
file size

Pre-Allocation vs Dynamic Allocation

- **Pre-allocation**: requires that the maximum size of a file be declared at the time of the file creation request
- For many applications — it is *difficult* to estimate reliably the maximum potential size of the file
 - Tends to be **wasteful** because users and application programmers tend to overestimate size
- **Dynamic allocation**: allocates space to a file in portions as needed

- Factors to be considered:
 1. **Contiguity of space increases performance**, especially for Retrieve_Next operations, and greatly for transactions running in a transaction-oriented operating system
 2. Having a large number of small portions increases the size of tables needed to manage the allocation information
 3. Having **fixed-size portions** simplifies the reallocation of space
 4. Having **variable-size** or **small fixed-size portions** minimises waste of unused storage due to overallocation

Portion Size: Alternatives

Variable, Large Contiguous Portions

- Provide better performance
- Variable size **avoid wasteful space**
- File allocation tables are small

fragmentation of free space is of concern

Blocks

- **Small-fixed portions** provide greater flexibility
- May required large tables or complex structures for their allocation
- Primary goal: contiguity has been abandoned
- Blocks are allocated as needed

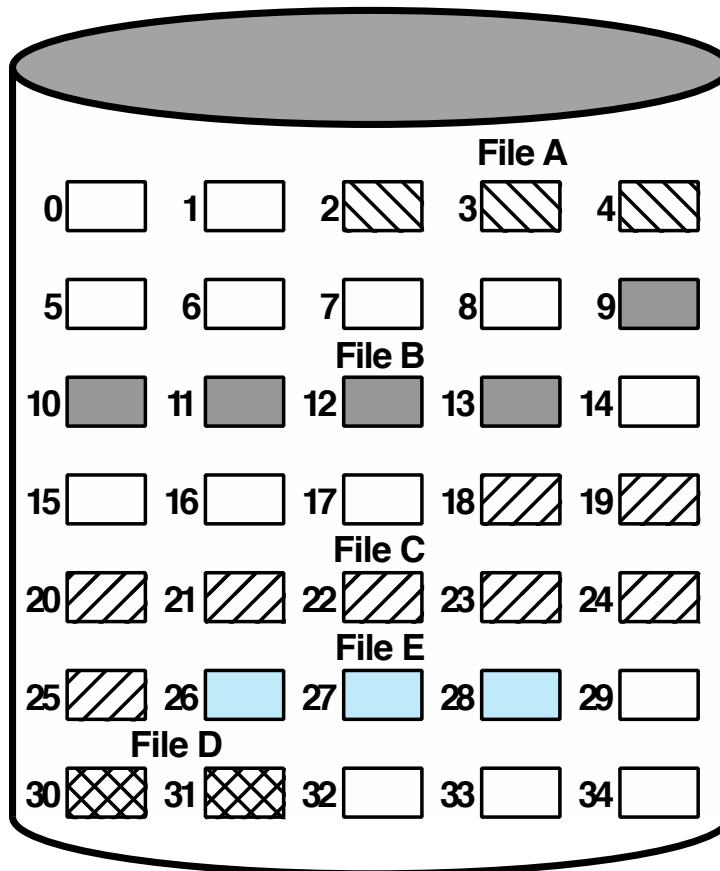
What are the different file allocation methods?

File Allocation: Methods

	Contiguous	Chained	Indexed	
Preallocation?	Necessary	Possible	Possible	
Fixed or variable size portions?	Variable	Fixed blocks	Fixed blocks	Variable
Portion size	Large	Small	Small	Medium
Allocation frequency	Once	Low to high	High	Low
Time to allocate	Medium	Long	Short	Medium
File allocation table size	One entry	One entry	Large	Medium

File Allocation: Contiguous

External
fragmentation

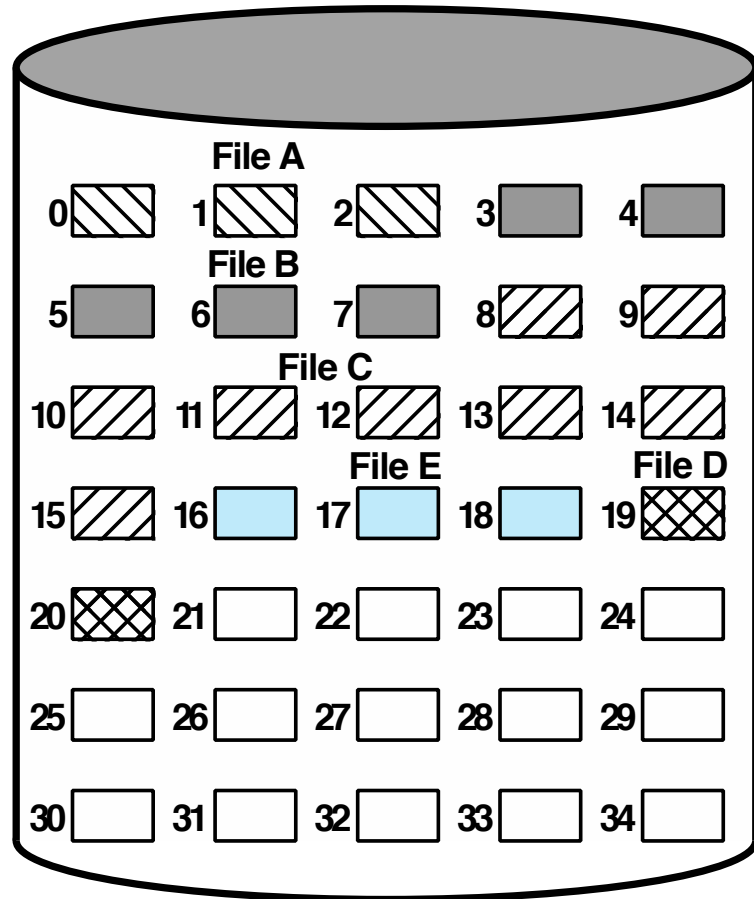


File Allocation Table

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

- A single contiguous block is allocated to a file at the time of file creation
- FAT needs just one single entry for each file, showing the starting block and the length of the file

File Allocation: Contiguous



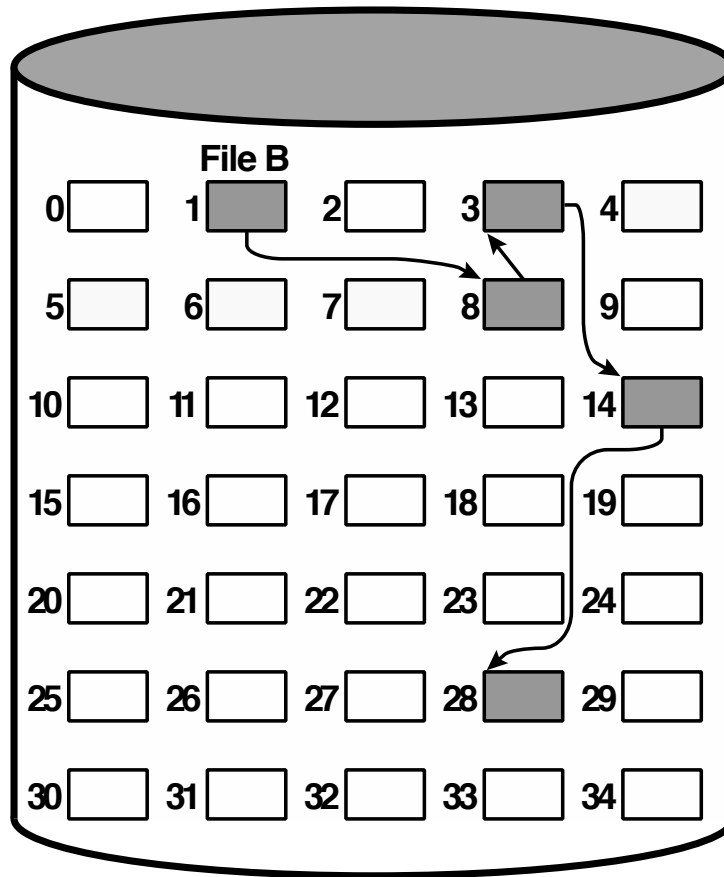
File Allocation Table

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

(after compaction)

File Allocation: Chained

No external fragmentation



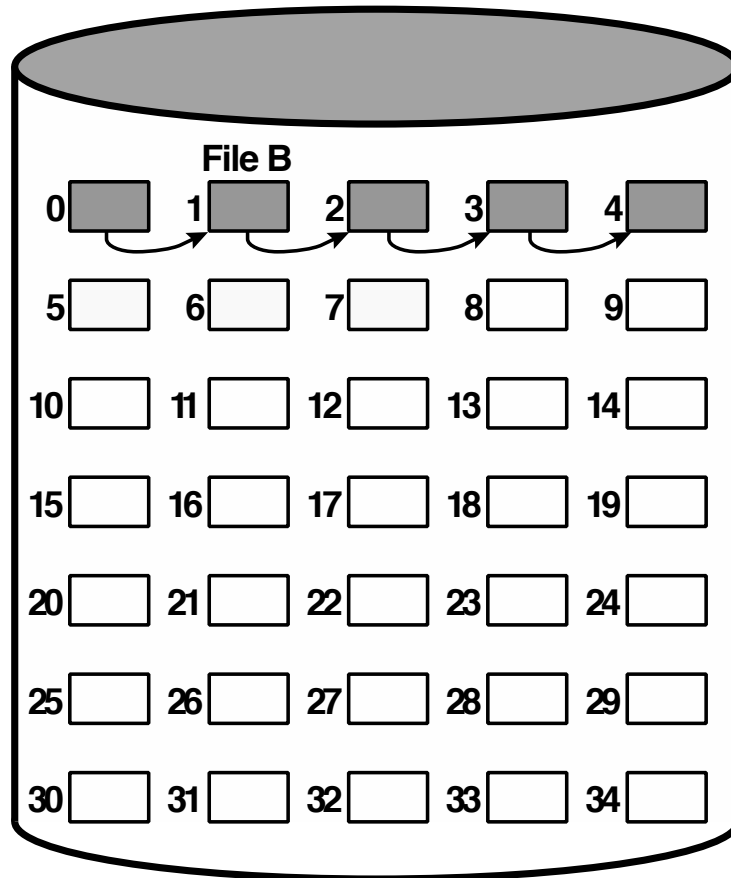
File Allocation Table

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

- Allocation is on an individual block basis
- Each block contains a pointer to the next block in the chain
- FAT needs just a single entry for each file

File Allocation: Chained

No accommodation
for the principle
of locality



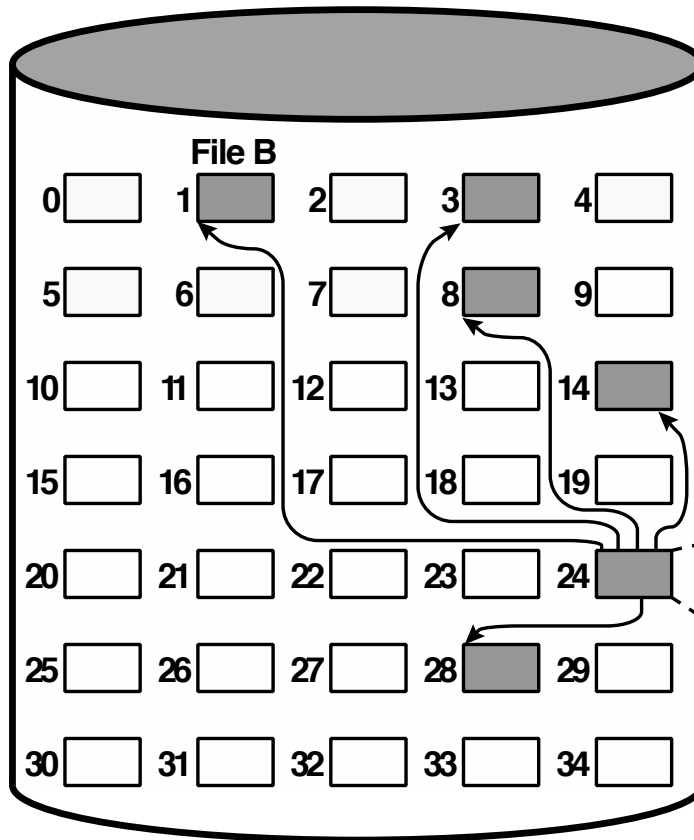
File Allocation Table

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

(after consolidation)

File Allocation: Indexed with Block Portions

**Eliminates
external
fragmentation**



File Allocation Table

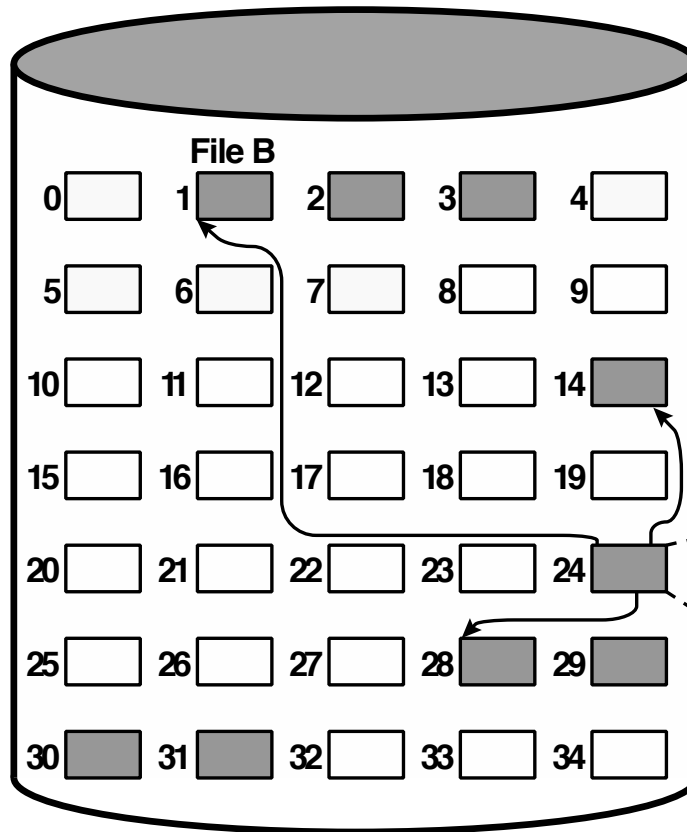
File Name	Index Block
...	...
File B	24
...	...

- FAT contains a separate 1-level index for each file
- The index has one entry for each portion allocated to the file

1
8
3
14
28

File Allocation: Indexed with Variable-Length Portions

Improves
locality



File Allocation Table

File Name	Index Block
...	...
File B	24
...	...

Start Block	Length
1	3
28	4
14	1

Indexed allocation supports
both sequential and direct
access to the file


How should the free space be managed?

Free Space Management

- Just as allocated space must be managed, so must the unallocated space
- To perform file allocation, it is necessary to know which blocks are available
- A disk allocation table is needed in addition to a file allocation table

Bit Tables

- A **vector** containing one bit for each block on the disk is used
- Each entry of a **0** corresponds to a free block, and each **1** corresponds to a block in use
- **Advantages:**
 - It works well with any file allocation method
 - It is as small as possible



Easy to find one or a contiguous group of free blocks

Chained Free Portions

- Free portions may be **chained** together by using a pointer and length value in each free portion
- Negligible space overhead because there is no need for a disk allocation table
- Suited to all file allocation methods
- **Disadvantages:**
 - Every time a block needs to be allocated, the pointer to the new first free block needs to be recovered before writing data to that block



Also leads to fragmentation

Free Block List

Each block is assigned a number sequentially

the list of the numbers of all free blocks is maintained in a reserved portion of the disk

Depending on the size of the disk, either 24 or 32 bits will be needed to store a single block number

the size of the free block list is 24 or 32 times the size of the corresponding bit table and must be stored on disk

There are two effective techniques for storing a small part of the free block list in main memory:

the list can be treated as a push-down stack with the first few thousand elements of the stack kept in main memory

the list can be treated as a FIFO queue, with a few thousand entries from both the head and the tail of the queue in main memory

File Management in Unix

Unix: File Management

Regular, or ordinary

- contains arbitrary data in zero or more data blocks

Directory

- contains a list of file names plus pointers to associated inodes

Special

- contains no data but provides a mechanism to map physical devices to file names

Named pipes

- an interprocess communications facility

Links

- an alternative file name for an existing file

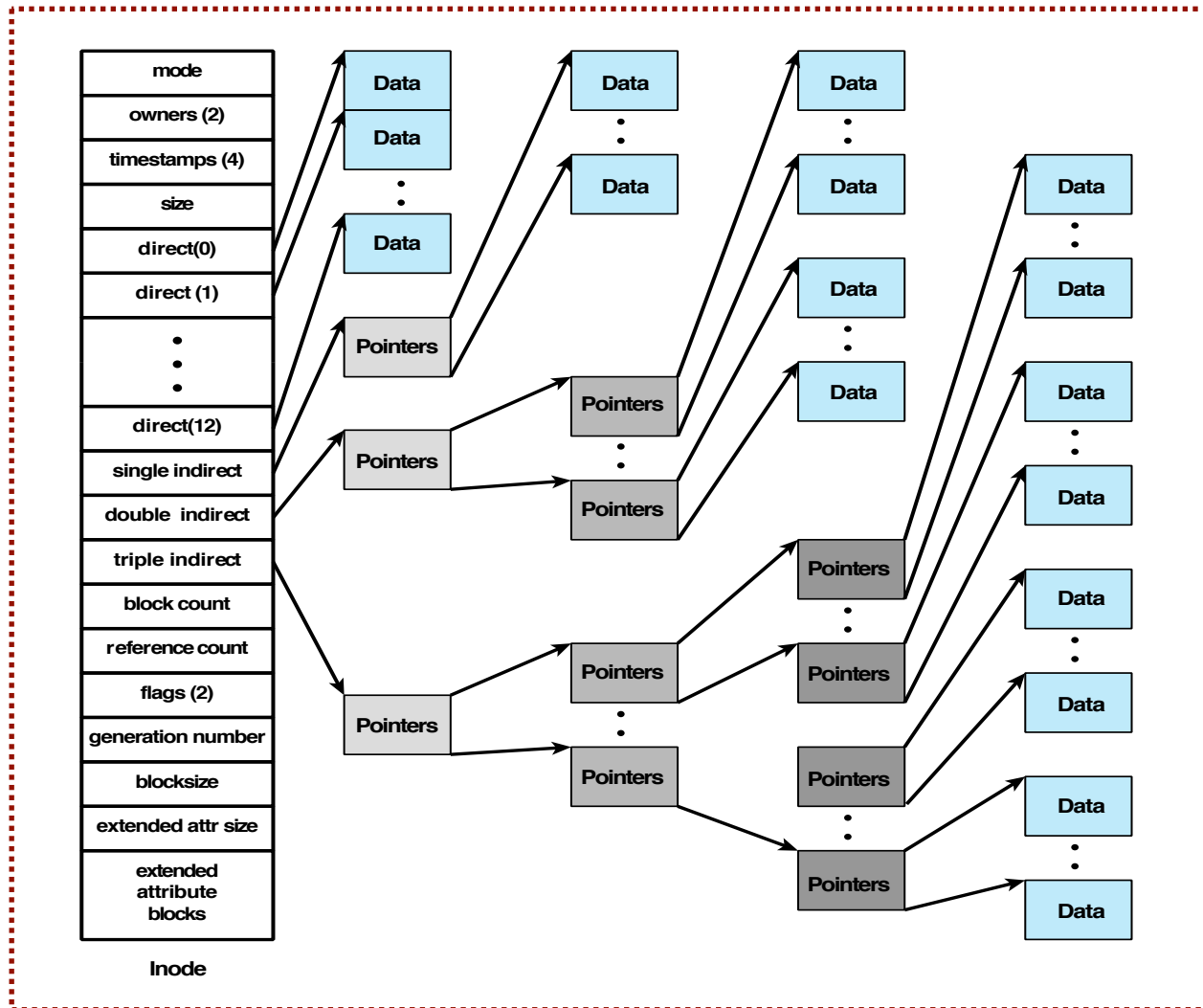
Symbolic links

- a data file that contains the name of the file it is linked to

Unix: Inodes

- All types of UNIX files are administered by the OS by means of **inodes**
- **Index node** (inodes): a *control structure* that contains the key information needed by OS for a particular file
- Several file names may be associated with a single inode
 - **An active inode is associated with exactly one file**
 - Each file is controlled by exactly one inode

Unix: File System Structure (Inode and File)



Unix: File Allocation

- File allocation is done on a **block** basis
- Allocation is **dynamic**, as needed, rather than using preallocation
- An **indexed** method is used to keep track of each file — with part of the index stored in the inode for the file
- In all UNIX implementations, the inode includes — a number of **direct pointers** and **three indirect pointers** (single, double, triple)

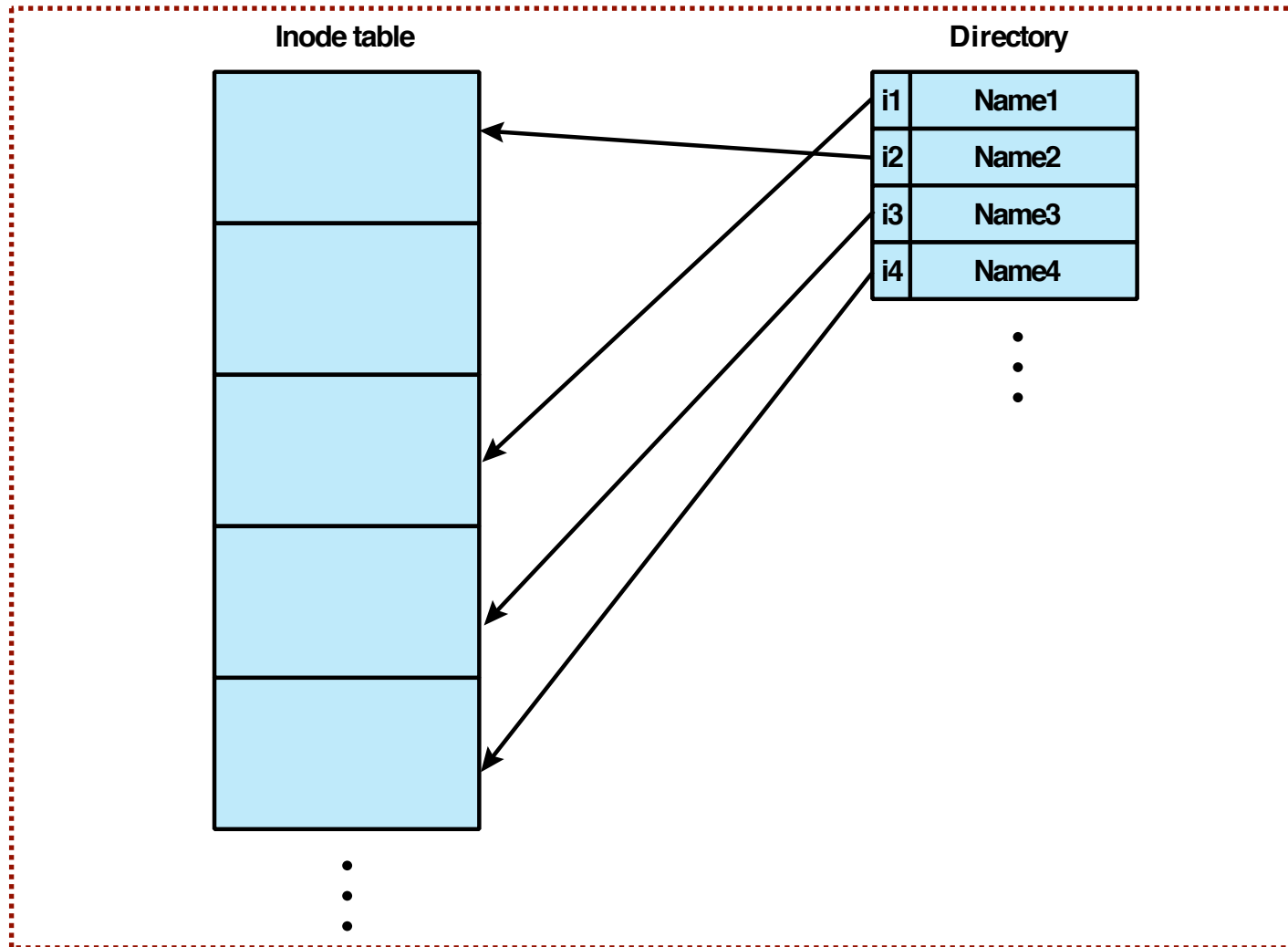
Unix: File Allocation

Each block can
hold 512 block
addresses

Level	Number of Blocks	Number of Bytes
Direct	12	48K
Single Indirect	512	2M
Double Indirect	$512 \times 512 = 256K$	1G
Triple Indirect	$512 \times 256K = 128M$	512G

Capacity of a Unix (FreeBSD) file with 4 kByte block size

Unix: Directories and Inodes



Summary of Lecture 11

- A file management system is a set of system software that provides services to users and applications in the use of files. It is typically viewed as a system service that is served by OS.
- A file consists of a collection of records. The way in which these records are accessed determines its logical organisation as well as its physical organisation on disk.
- A key function of any file management scheme is the management of disk space. It is essential to maintain a disk allocation table to indicate which blocks are available.

Next week: Operating System Security