Files and Streams

What You Will Learn

Create files

Write files

Read files

Update files

Do Random Access File Processing

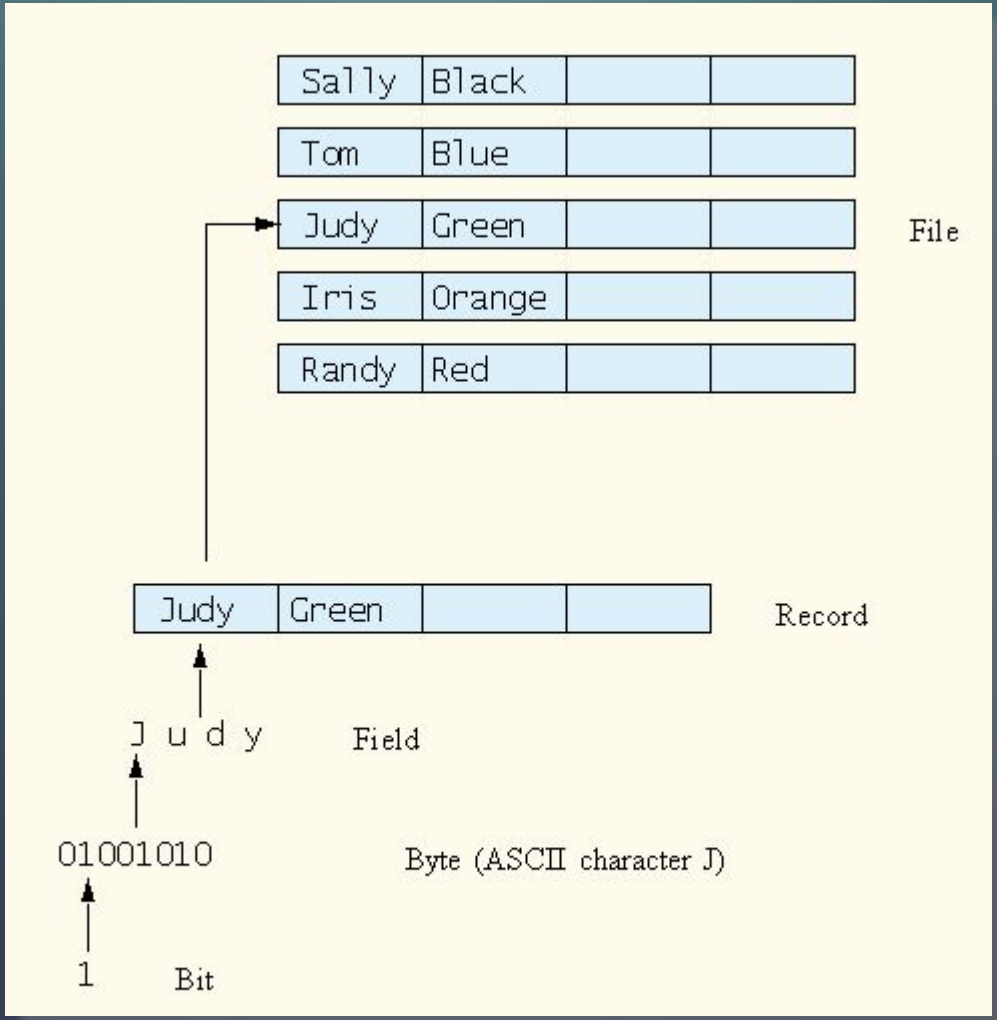
Introduction

- Storage of data in variables is temporary
  - when the program is done running,
    when computer is turned off
    The data is gone!
- Data stored permanently (more or less) on secondary storage devices
  - magnetic disks
  - optical disks
  - tapes
- We consider creation & use of files of data

The Data Hierarchy

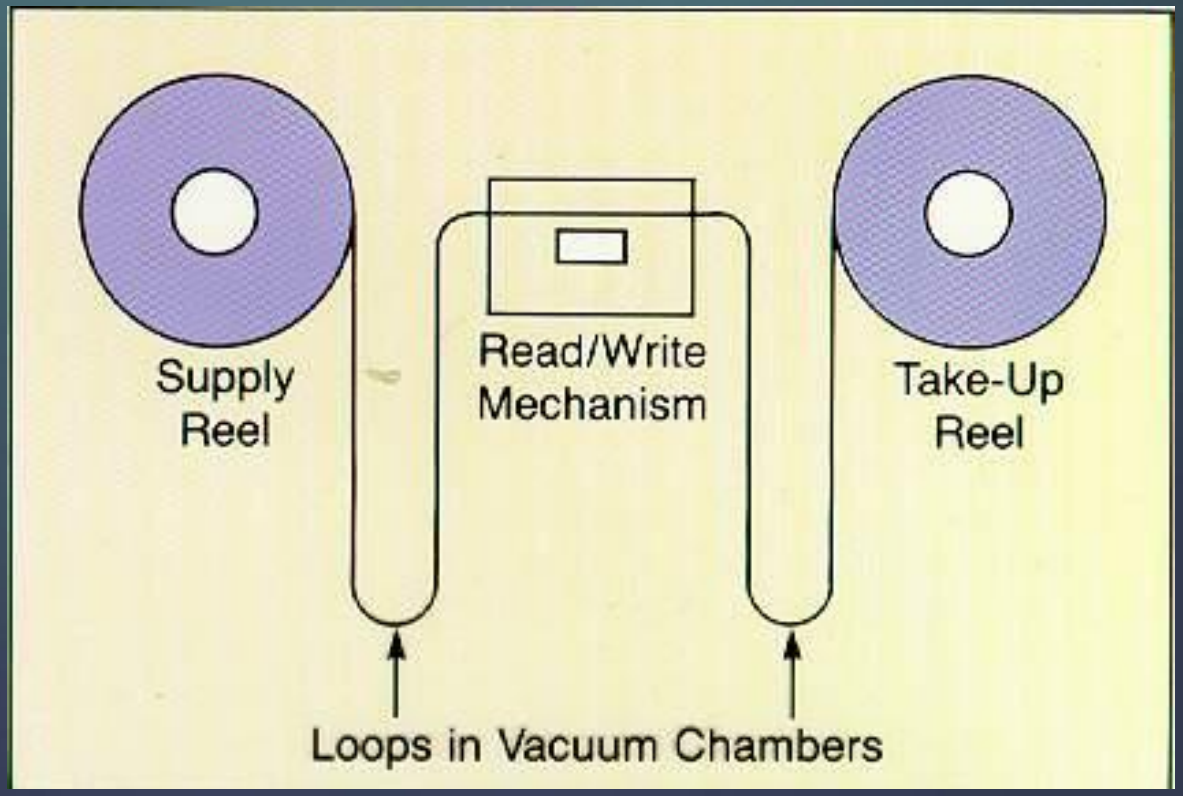
- Lowest level of data storage is in binary digits
  - 0s and 1s -- bits
- Bits grouped together into bytes
- Bytes grouped into characters
- Characters and/or bytes grouped together to form fields
- Fields grouped together to form records
- Records grouped to form files

The Data Hierarchy



Sequential Devices -- Some History

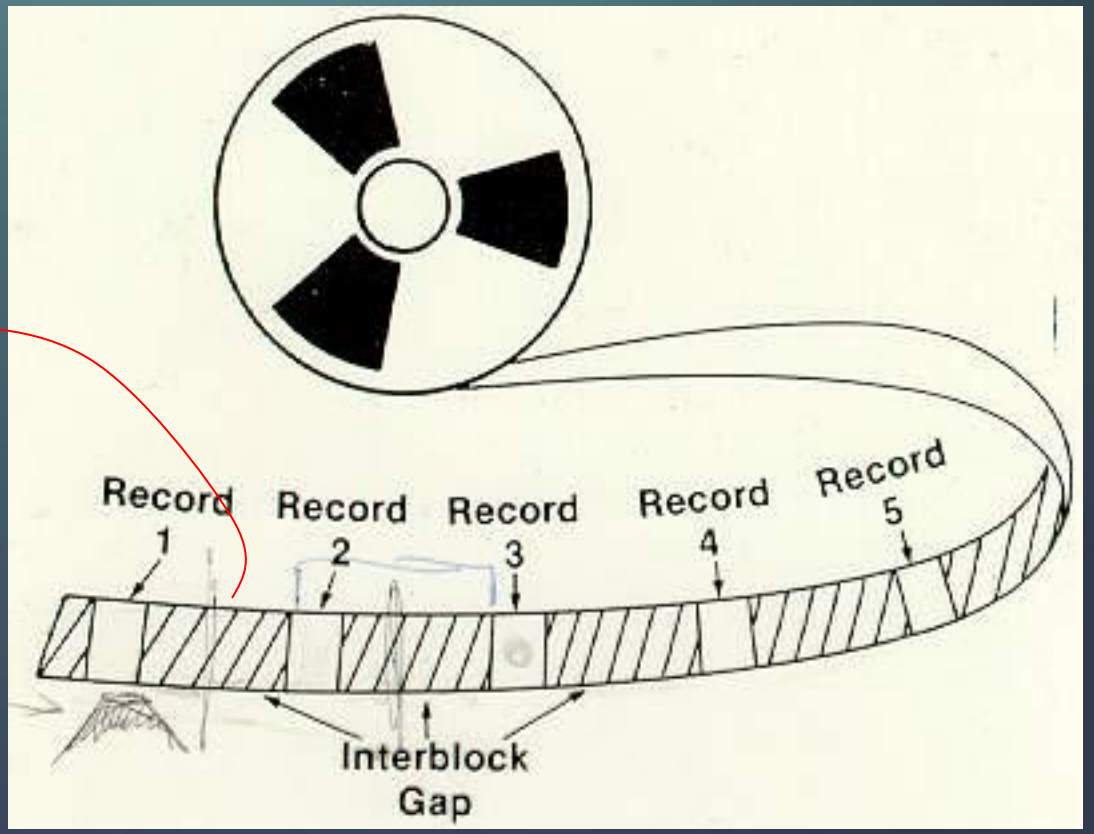
- Tape Drives



Data Stored on Tape

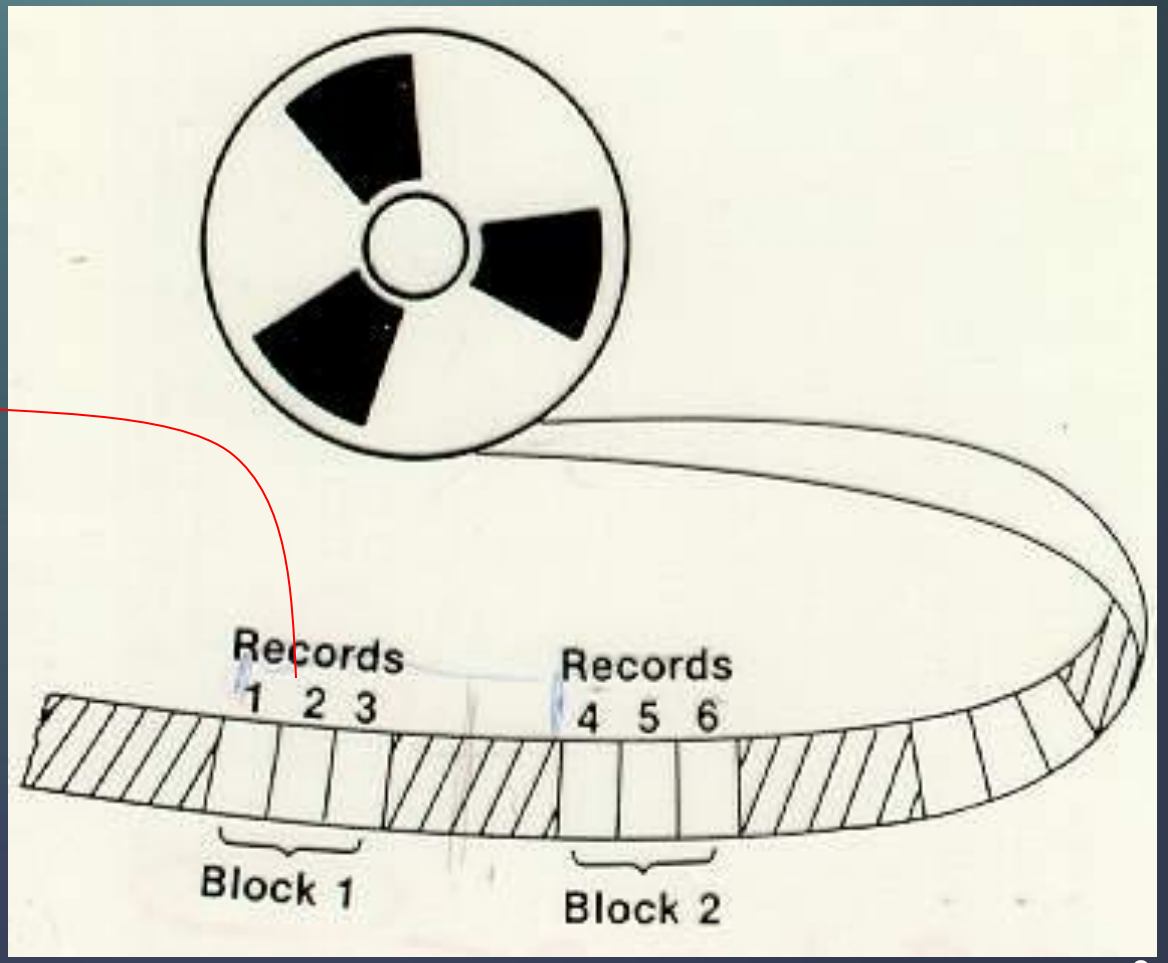
Interblock Gap for purpose of acceleration or deceleration of tape past the read/write head.

Result is large percentage of tape length wasted on the gaps.



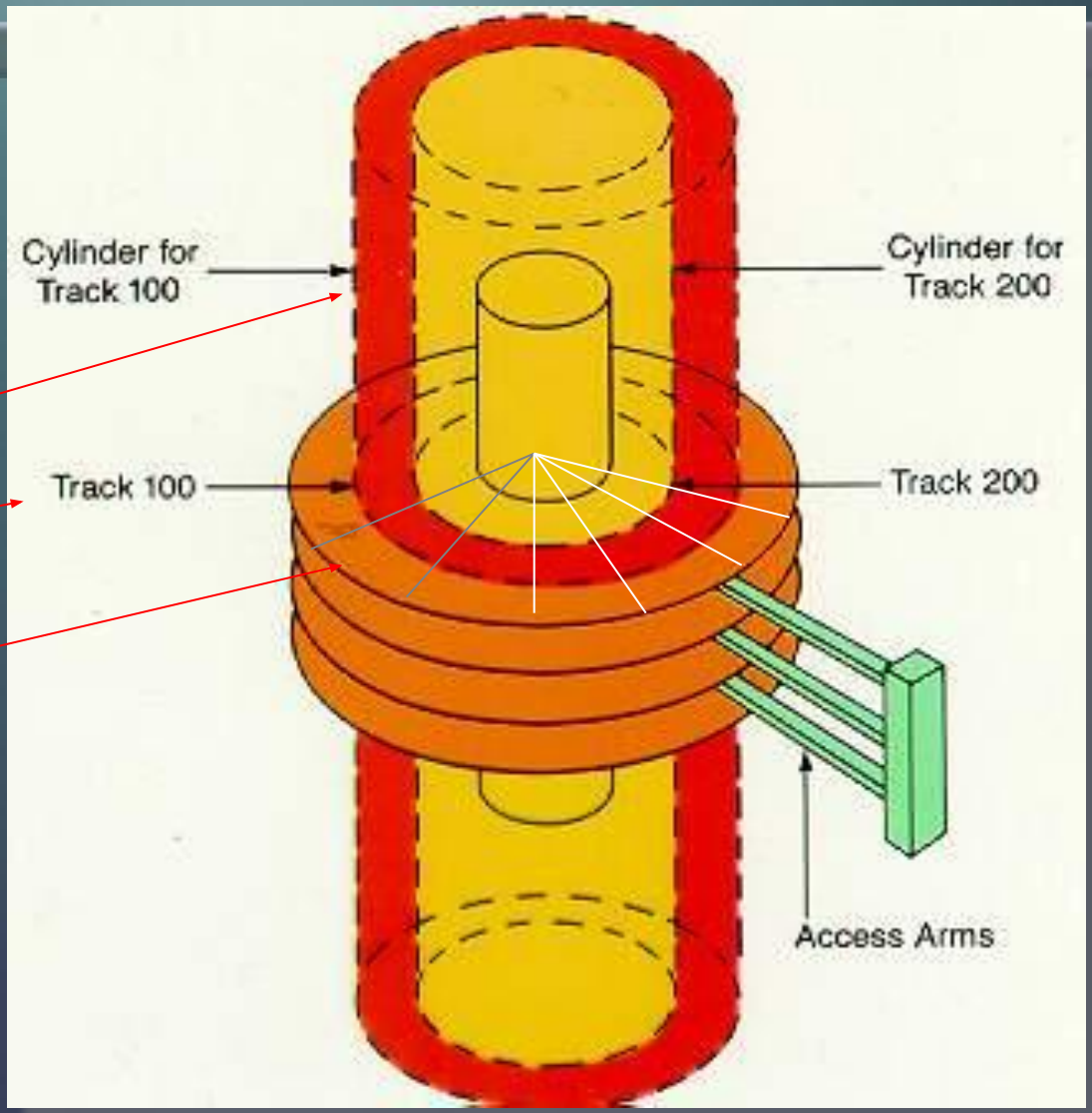
Data Stored on Tape

Records blocked in groups to save space wasted by the required gaps

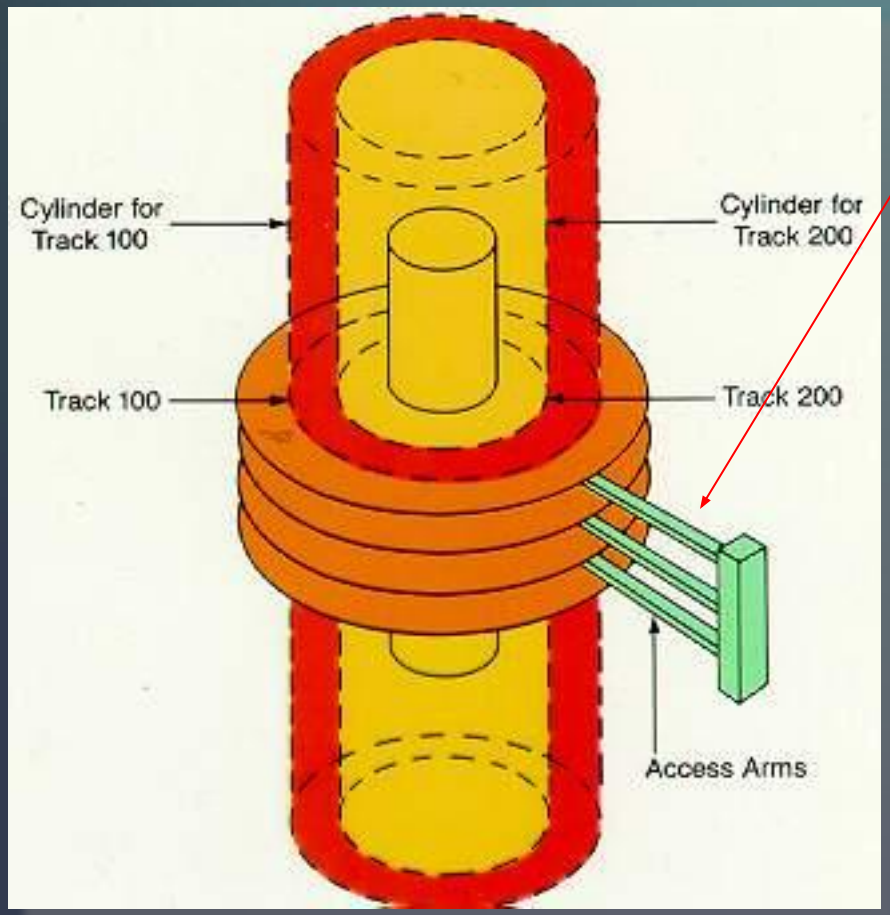


Disk Components

- Cylinders
- Tracks
- Sectors

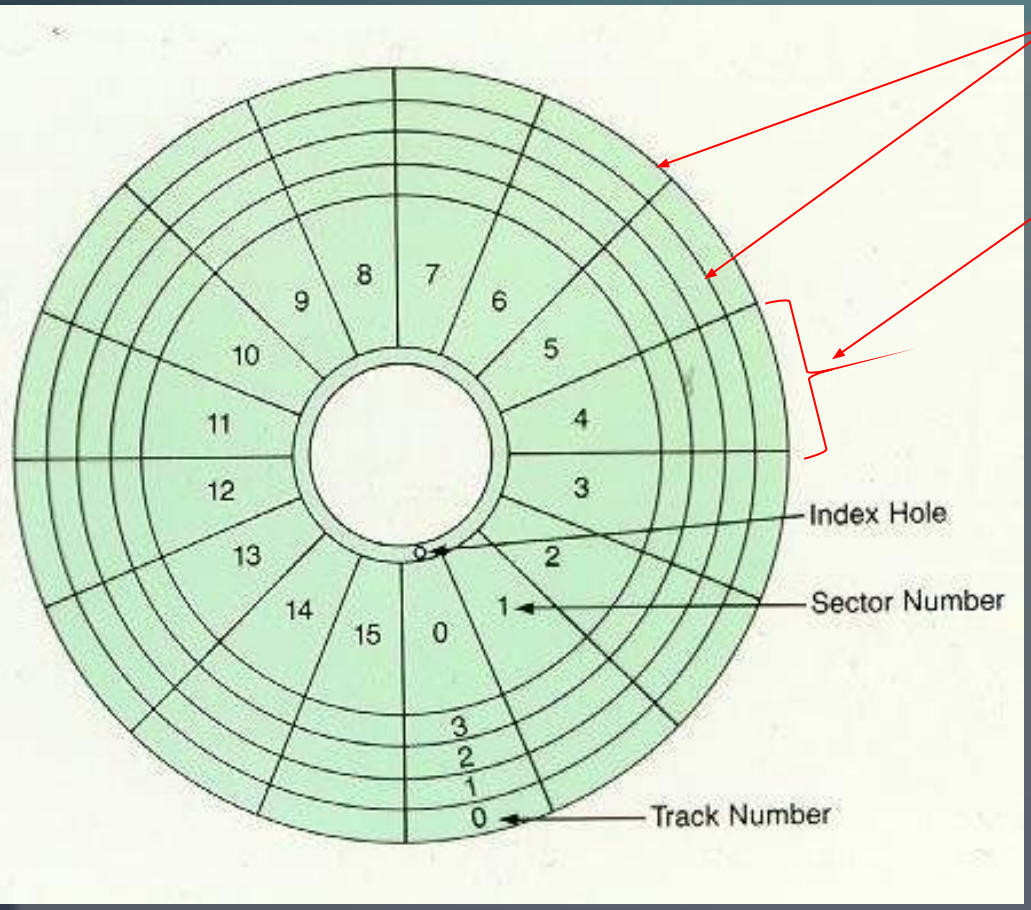


Disk Components



- Read/Write heads move in and out
- Possible to have multiple fixed heads
- Need head for each track surface
- Same amount of data on inside as outside tracks

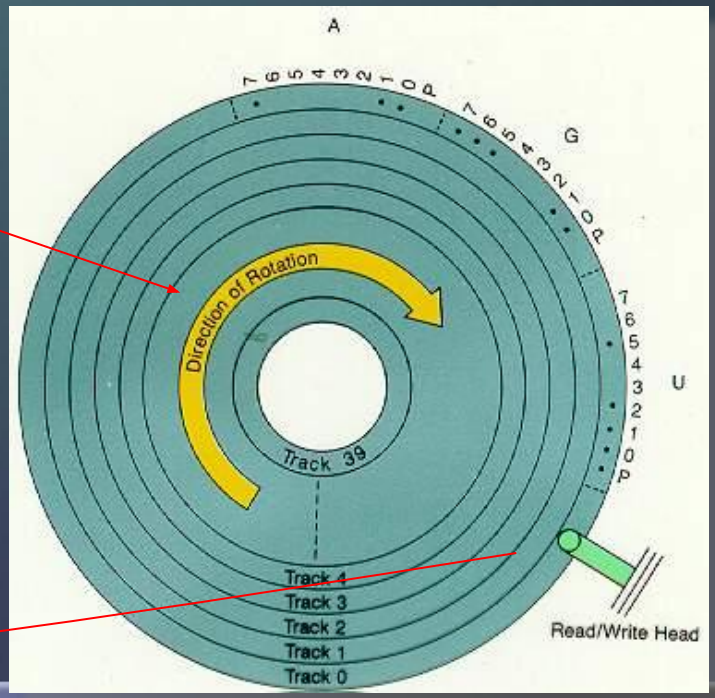
Disk Components



- Track
- Sector
- Inner sectors have same amount of data as outer outer (why?)
- Outer sectors less dense (bits per inch)

Timing Components

- Access motion time (seek time)
  - motion of read/write heads between tracks
  - this is the most lengthy time of the three
- Rotational delay
- Data transfer time

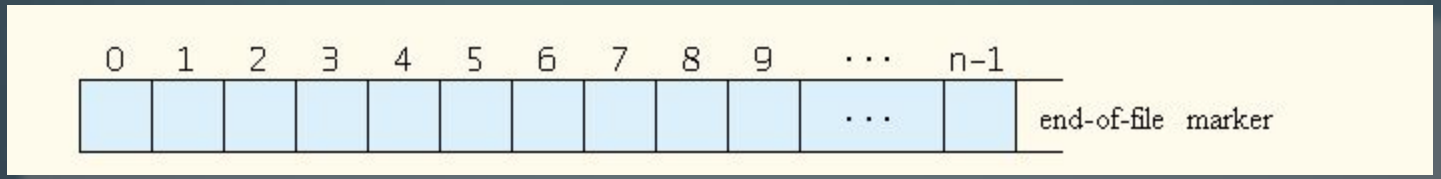


Storage of Sequential Files

- Any type of serial access storage device
  - originally cards
  - magnetic tape
  - even punched paper tape!
- Now would be placed on direct access devices (disks)

Files and Streams

- Java views a file as a stream of bytes
- File ends with end-of-file marker or a specific byte number



- File as a stream of bytes associated with an object
  - Java also associates streams with devices
    - **`System.in`**, **`System.out`**, and **`System.err`**
    - Streams can be redirected

Files and Streams

- File processing with classes in package `java.io`
  - **`FileInputStream`** for byte-based input from a file
  - **`FileOutputStream`** for byte-based output to a file
  - **`FileReader`** for character-based input from a file
  - **`FileWriter`** for character-based output to a file

Files and Streams

- Buffering
  - Improves performance of I/O
  - Copies each output to a region of memory called a buffer
- We can send the entire buffer to disk at once
  - One long disk access takes less time than many smaller ones
  - Due to repeated head seeks required
- **`BufferedOutputStream`** buffers file output
- **`BufferedInputStream`** buffers file input

Class `File`

- Methods from class `File`

Method	Description
`boolean canRead()`	Returns `true` if a file is readable; `false` otherwise.
`boolean canWite()`	Returns `true` if a file is writable; `false` otherwise.
`boolean exists()`	Returns `true` if the name specified as the argument to the `File` constructor is a file or directory in the specified path; `false` otherwise.
`boolean isFile()`	Returns `true` if the name specified as the argument to the `File` constructor is a file; `false` otherwise.
`boolean isDirectory()`	Returns `true` if the name specified as the argument to the `File` constructor is a directory; `false` otherwise.
`boolean isAbsolute()`	Returns `true` if the arguments specified to the `File` constructor indicate an absolute path to a file or directory; `false` otherwise.
`String getAbsolutePath()`	Returns a string with the absolute path of the file or directory.

Class `File`

- Methods from class `File`

Method	Description
`String getName()`	Returns a string with the name of the file or directory.
`String getPath()`	Returns a string with the path of the file or directory.
`String getParent()`	Returns a string with the parent directory of the file or directory—that is, the directory in which the file or directory can be found.
`long length()`	Returns the length of the file, in bytes. If the `File` object represents a directory, 0 is returned.
`long lastModified()`	Returns a platform-dependent representation of the time at which the file or directory was last modified. The value returned is useful only for comparison with other values returned by this method.
`String[] list()`	Returns an array of strings representing the contents of a directory. Returns `null` if the `File` object is not a directory.

Class `File`

- View Figure 17.4
  - Creates a GUI with `JTextField`
  - Enter file, directory name
  - `JTextArea` displays information
- Program features
  - Body of `if` outputs information about file if it exists
  - Test if object is a file, test if file exists
  - Create a reader to gather data from file
  - Read text until no more in file
  - Get list of files in directory

Creating a Sequential-Access File

- Java imposes no structure on a file
- Programmer structures file according to application
- **`AccountRecord`** class specified in Figure 17.6 declares structure for file access examples … note …
  - Class compiled into package
  - Imlements **`Serializable`** for use without I/O streams

Creating a Sequential-Access File

- Program in Figure 17.5 prepares GUI's for file access program
- Note
  - Class compiled as a package
  - Buttons provided for actions in later examples
  - Generic `doTask` buttons

Creating a Sequential-Access File

- Program in Figure 17.7 imports the GUI and record classes
  - Interface and references to buttons created
  - Instantiate, assign `JFileChooser` object
  - Selected file retrieved, opened
  - Method `closeFile` to close current file
  - Data retrieved from text fields
  - New record retrieved, written to file

Reading Data from Sequential Access Files

- Data stored in files
- Retrieved for processing when needed
- Accessing a sequential file
  - Data must be read in same format it was written
- Note program of Figure 17.9
  - Create instance of interface
  - Respond to Open button, Next button
  - `JFileChooser` used as before
  - Method `readObject` reads an Object from the `ObjectInputStream`

Reading Data from Sequential Access Files

- To read from file repeatedly
  - Must close and reopen file
- Each time through
  - Print only records which satisfy condition
- Figure 17.19 allows inquiries
  - Note nesting of `FileInputStream` in `ObjectInputStream`
  - Note use of `catch` for `EOFException`

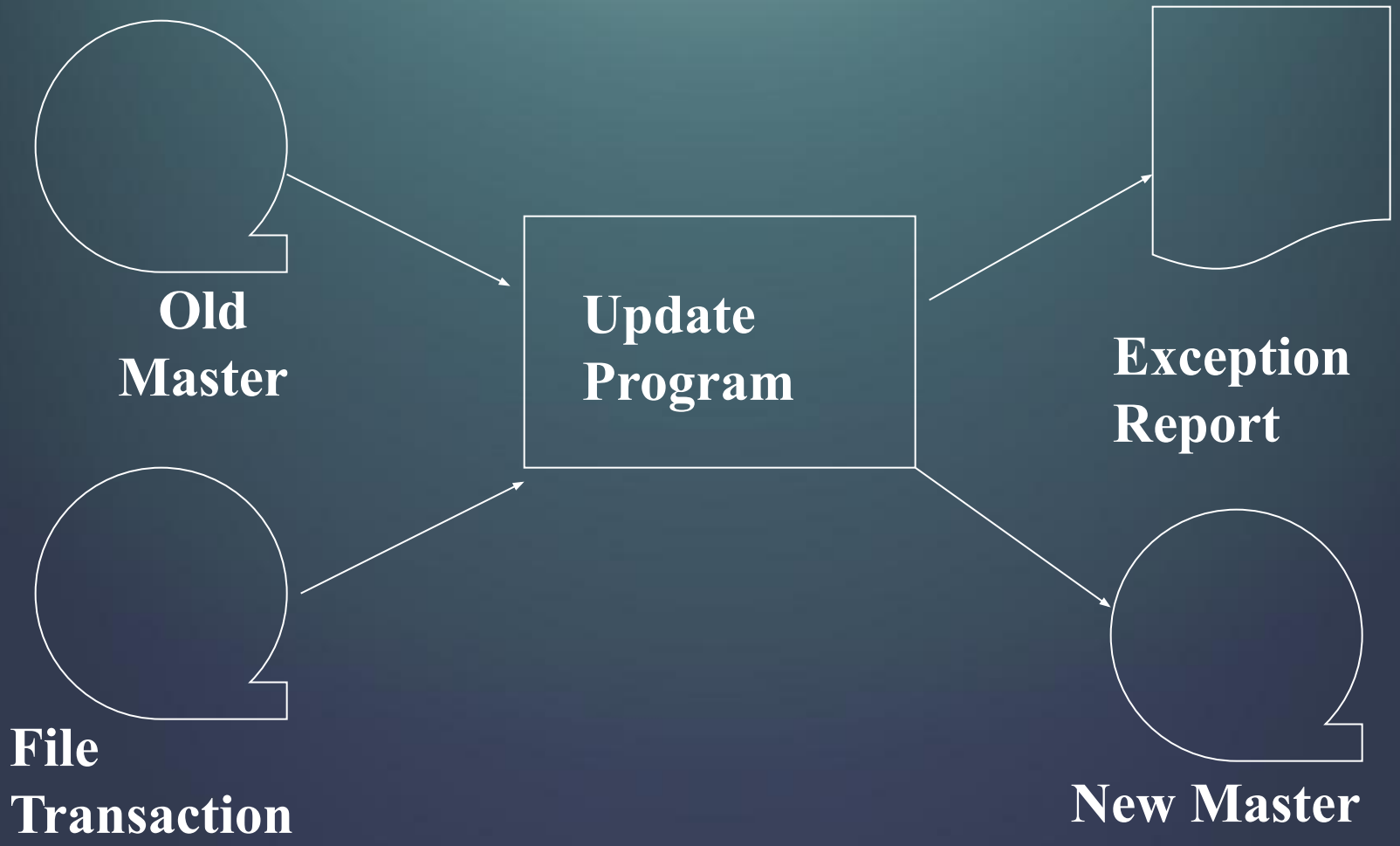
Updating Sequential Access Files

- Difficult to update a sequential-access file
  - Entire file must be rewritten to change one field
  - Only acceptable if many records being updated at once
- Note that this was done in the days of using magnetic tapes

Characteristics of Sequential Systems

- Two types of files
  - Master files -- files with relatively permanent data
  - Transaction files -- file of transitory records
- Updating master files
  - use transaction files
  - combine (merge) old master with transaction file
  - create new master file
  - Old master and Transaction file are the backup for the New Master

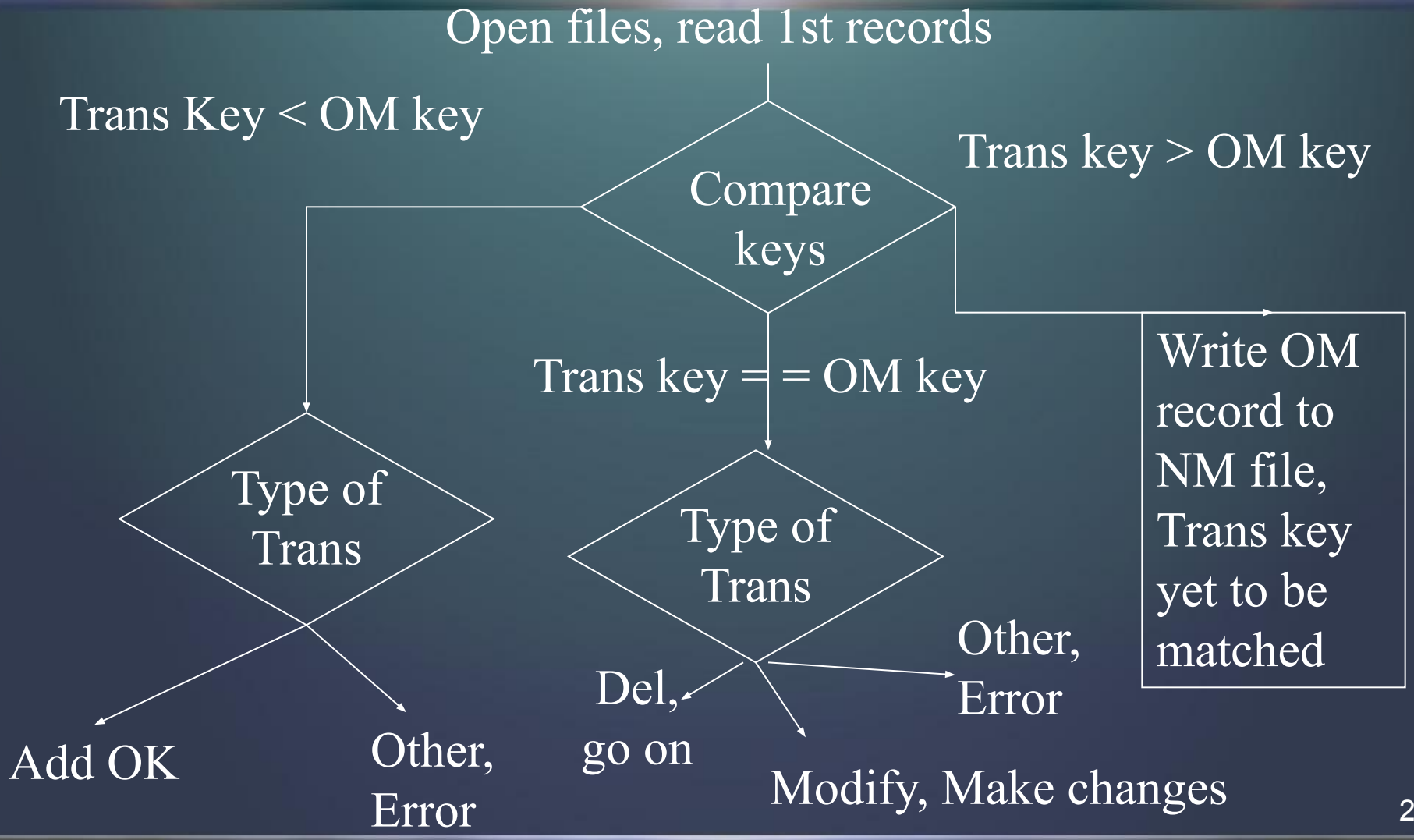
Updating Master Files



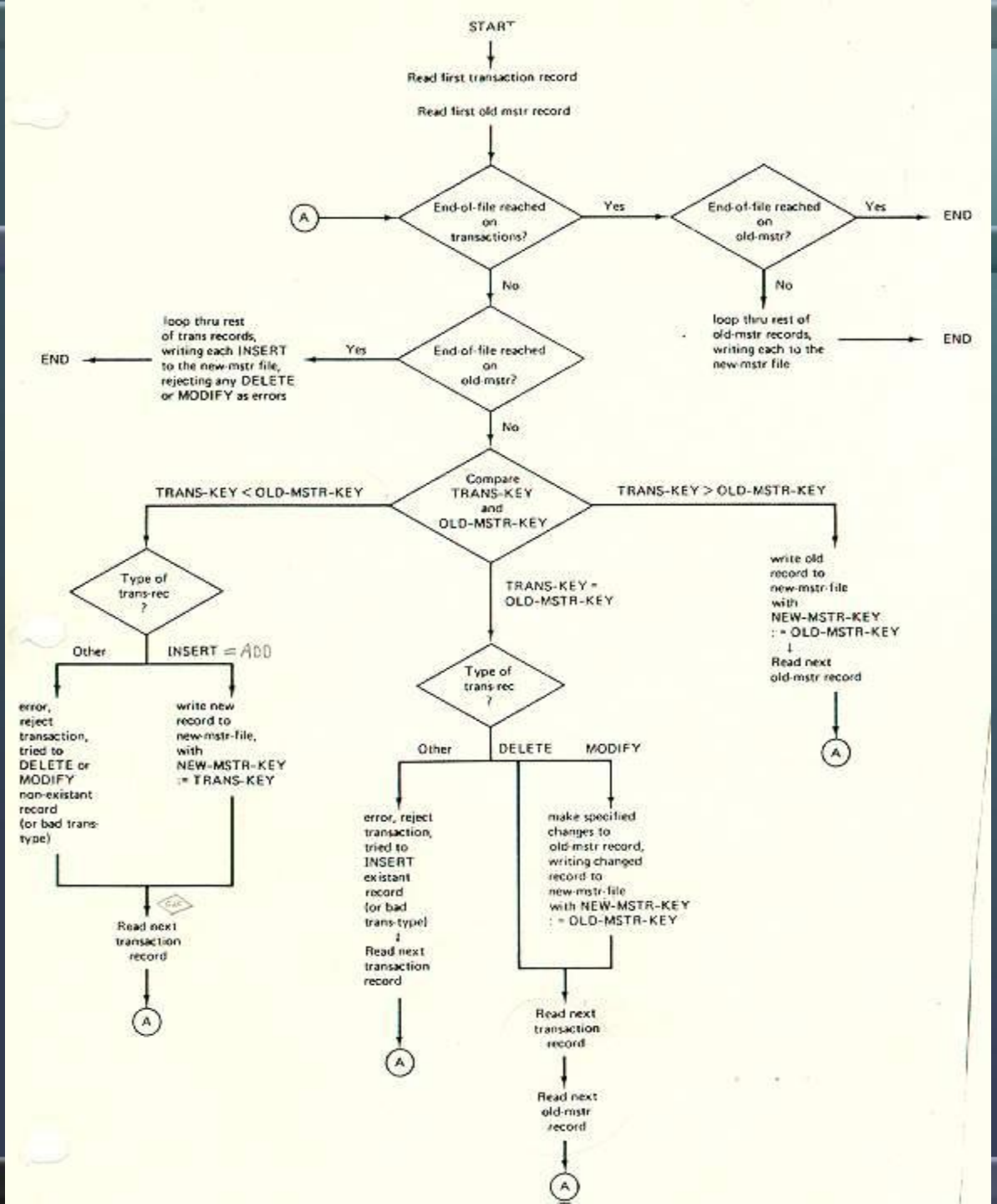
Updating Master Files

- Transaction file contains
  - records to be added
  - modifications to existing records on old master
  - orders for deletions of records on the old master
- Possible errors
  - Adding => record for specified ID already exists
  - Modify => record does not exist
  - Delete => record does not exist

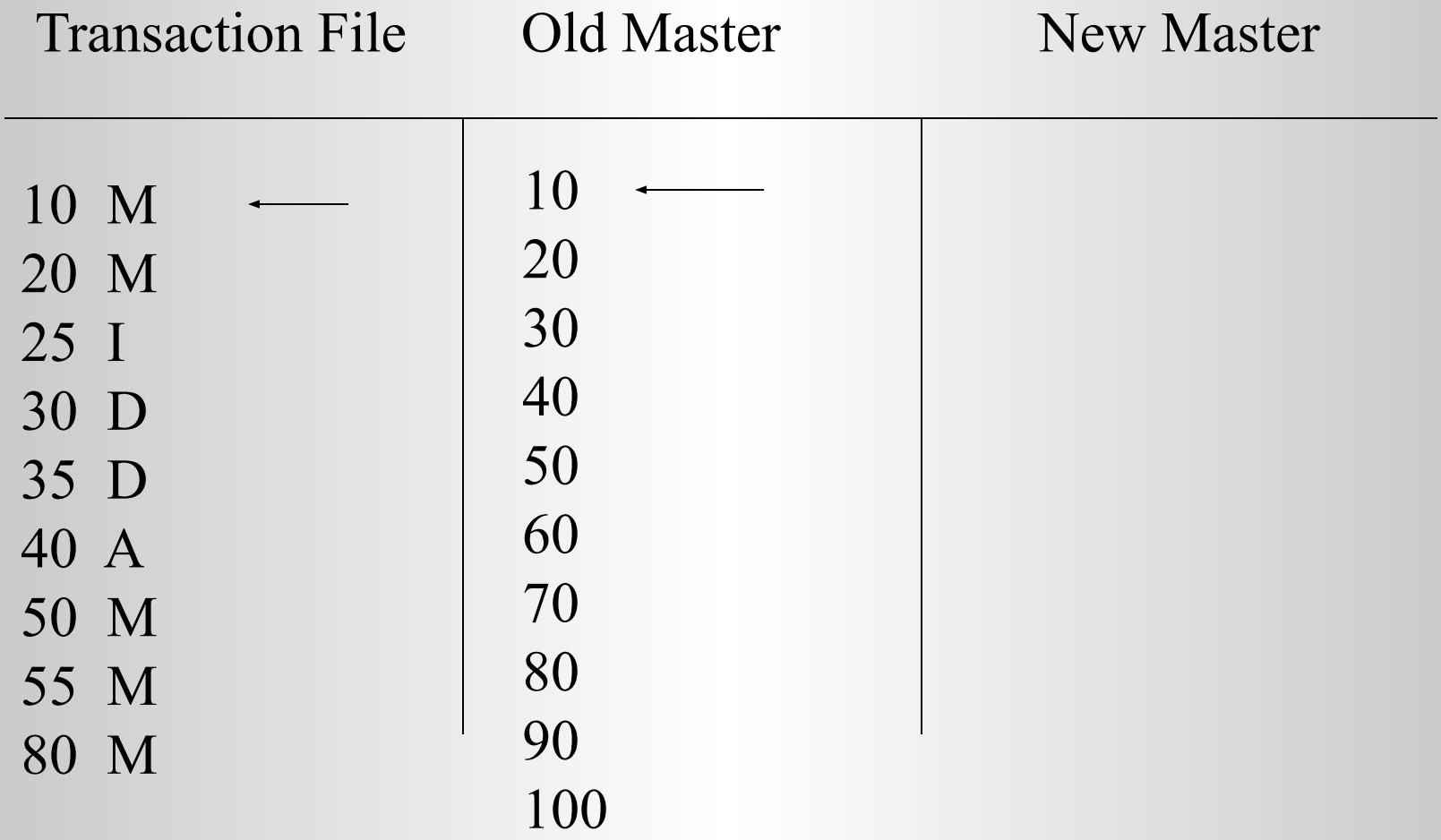
Program Flow Chart



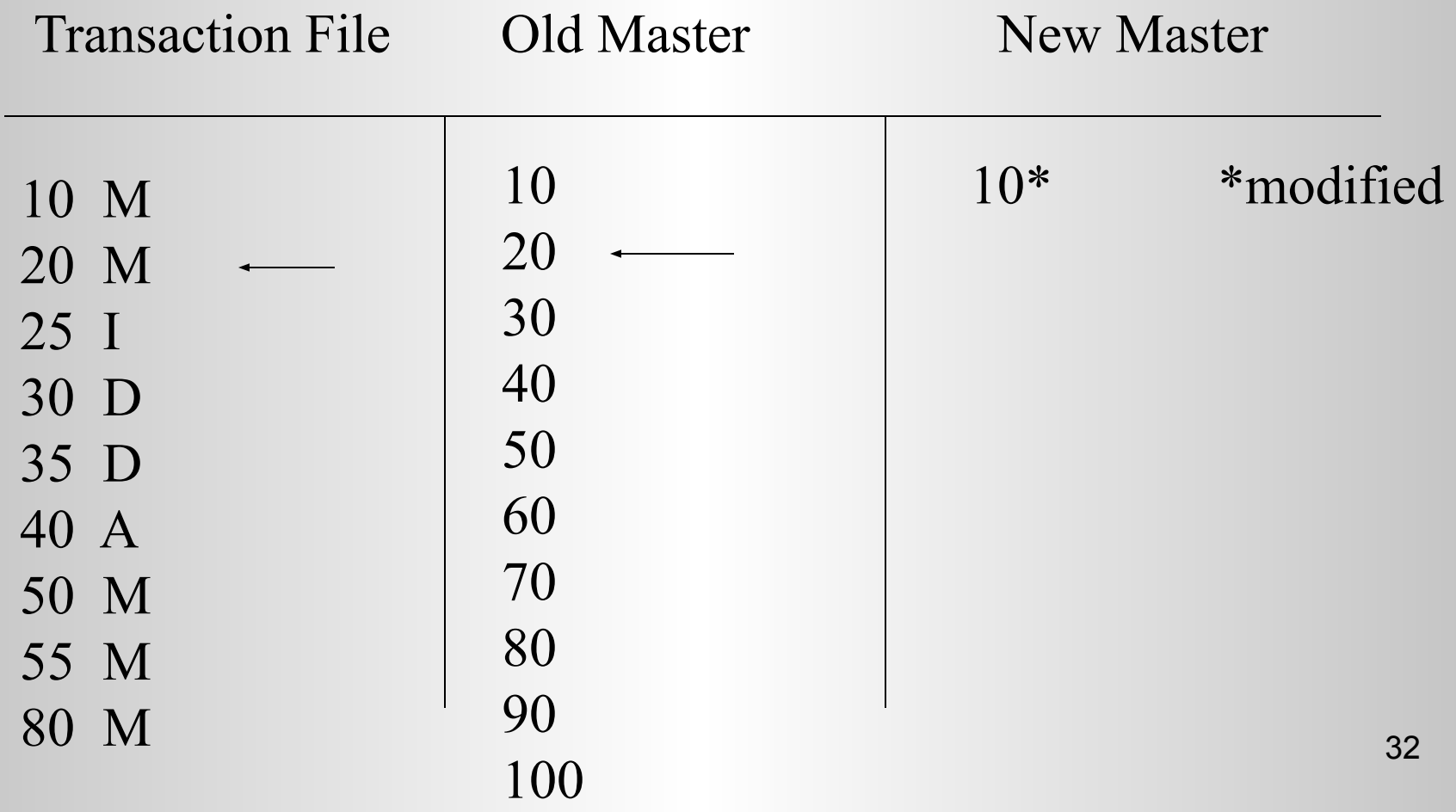
Sequential Update



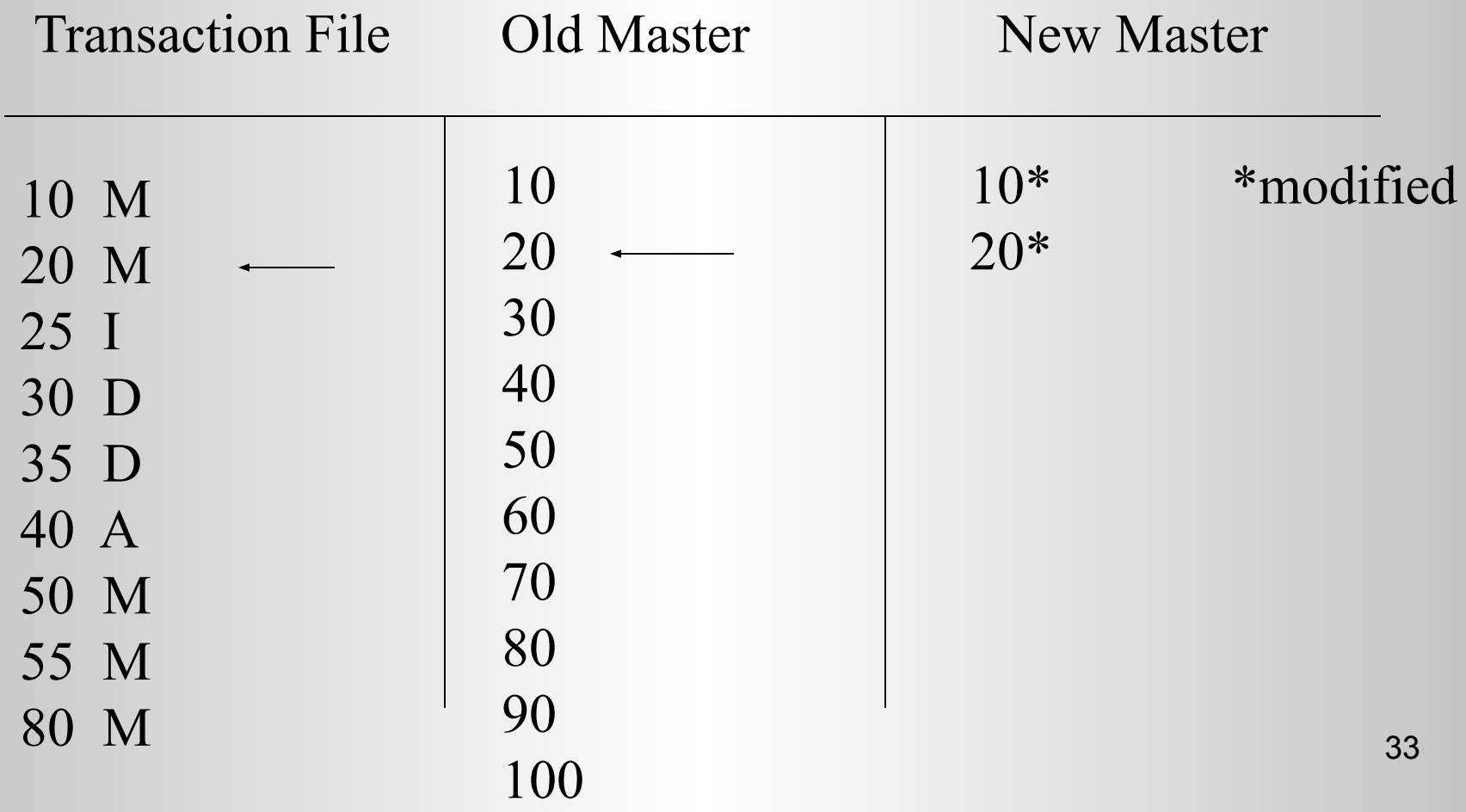
Sequential Update



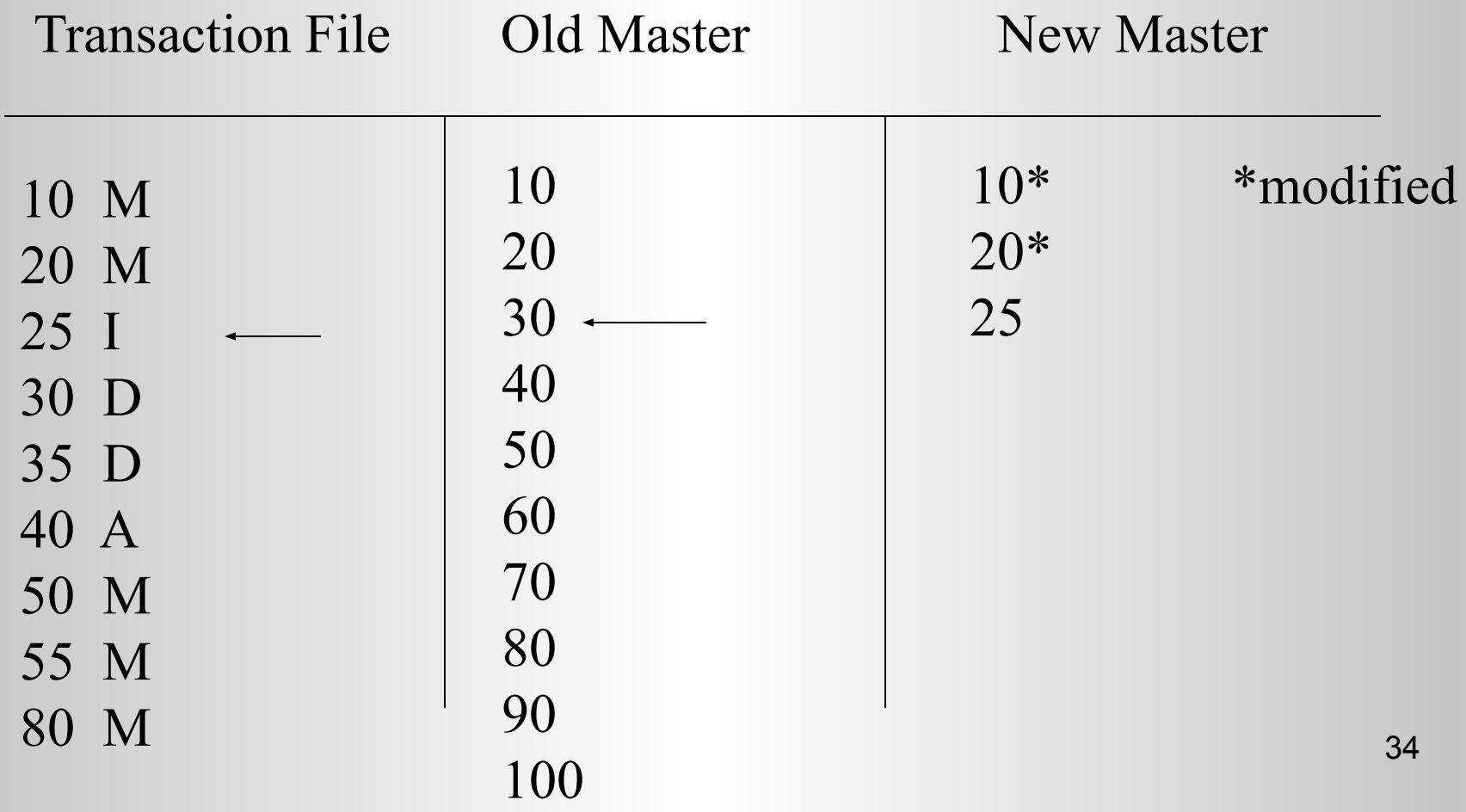
Sequential Update



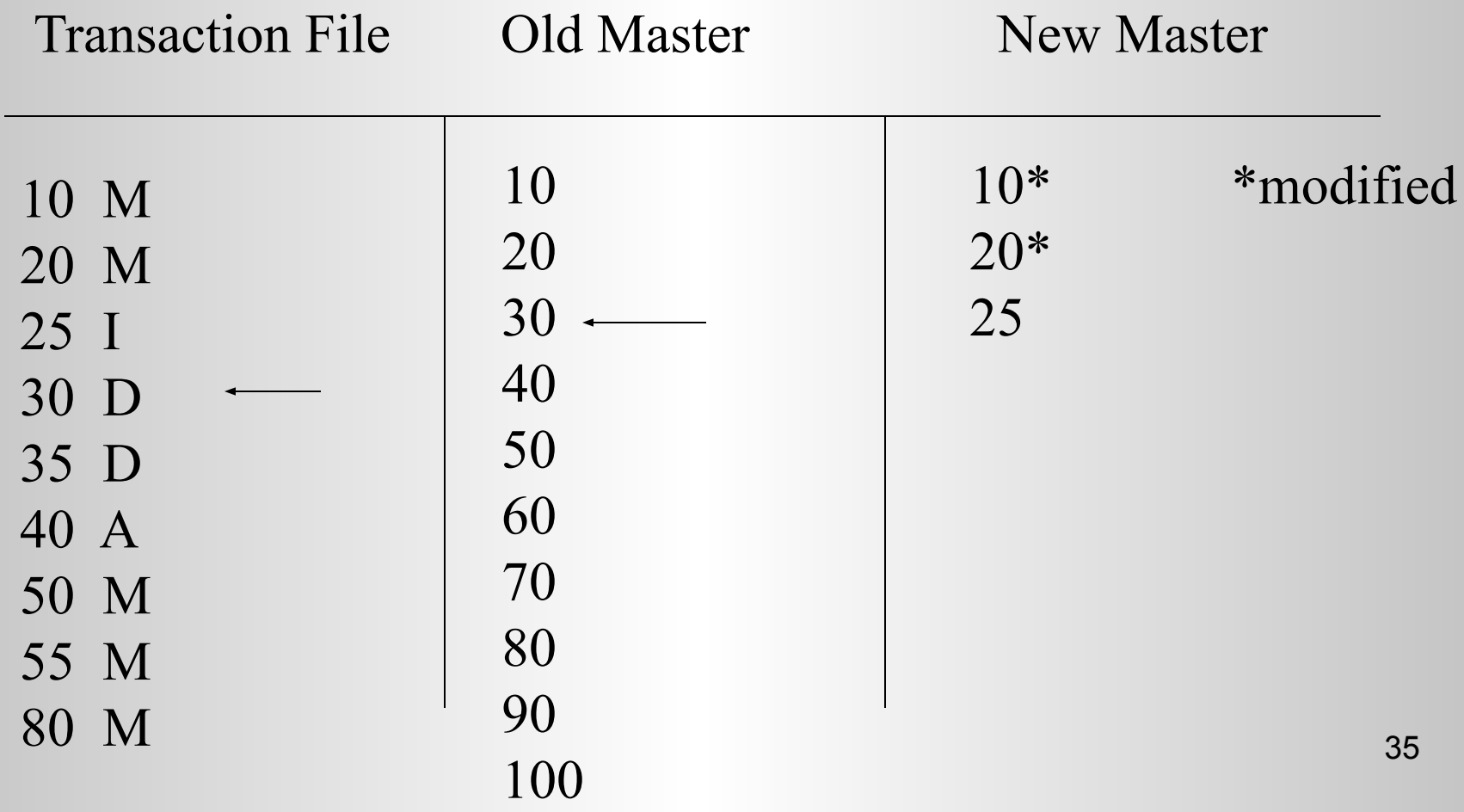
Sequential Update



Sequential Update



Sequential Update



Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40 ←	
35 D ←	50	
40 A	60	
50 M	70	
55 M	80	
80 M	90	
	100	

*modified

?35 D

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40 ←	
35 D	50	
40 A ←	60	
50 M	70	
55 M	80	
80 M	90	
	100	

*modified

?35 D
?40 A

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40 ←	40
35 D	50	
40 A	60	
50 M ←	70	
55 M	80	
80 M	90	
	100	

*modified

?35 D
?40 A

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40	40
35 D	50 ←	50*
40 A	60	
50 M ←	70	
55 M	80	
80 M	90	
	100	

*modified

?35 D
?40 A

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40	40
35 D	50	50*
40 A	60 ←	
50 M	70	
55 M ←	80	
80 M	90	
	100	

*modified

?35 D
?40 A
?55 M

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40	40
35 D	50	50*
40 A	60 ←	60
50 M	70	
55 M	80	
80 M ←	90	
	100	

*modified

?35 D
?40 A
?55 M

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40	40
35 D	50	50*
40 A	60	60
50 M	70 ←	70
55 M	80	
80 M ←	90	
	100	

*modified

Error Rpt

?35 D
?40 A
?55 M

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40	40
35 D	50	50*
40 A	60	60
50 M	70	70
55 M	80 ←	80*
80 M ←	90	
	100	

*modified

Error Rpt

?35 D
?40 A
?55 M

Sequential Update

Transaction File	Old Master	New Master
10 M	10	10*
20 M	20	20*
25 I	30	25
30 D	40	40
35 D	50	50*
40 A	60	60
50 M	70	70
55 M	80	80*
80 M ←	90 ←	90
	100	

*modified

Error Rpt

?35 D
?40 A
?55 M

Sequential Update

Transaction File	Old Master	New Master	
10 M	10	10*	*modified
20 M	20	20*	
25 I	30	25	
30 D	40	40	
35 D	50	50*	
40 A	60	60	
50 M	70	70	
55 M	80	80*	
80 M	90	90	
←	100 ←	100	

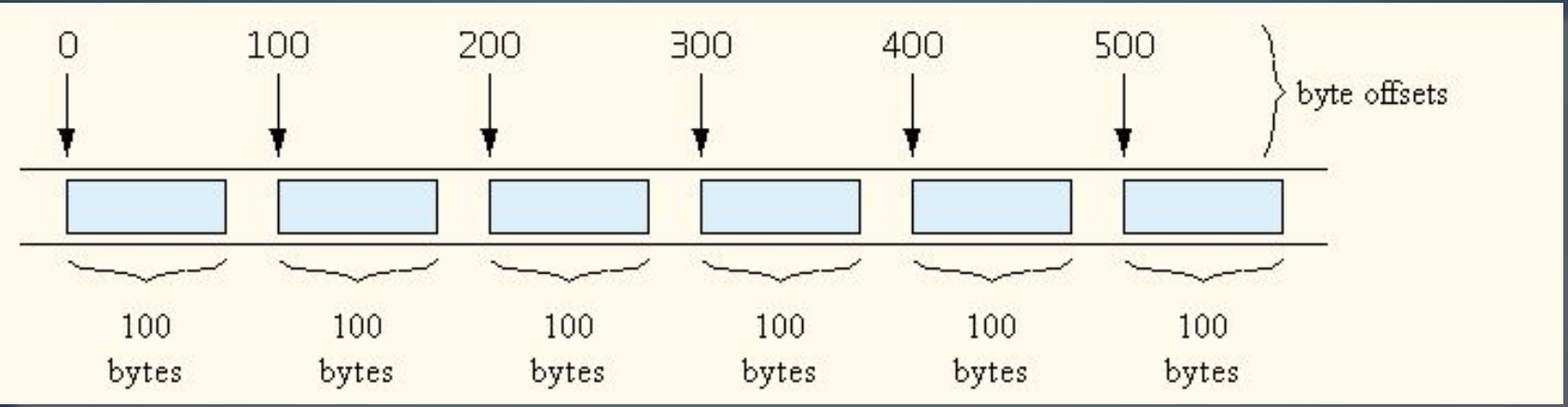
Error Rpt

?35 D
?40 A
?55 M

Random-Access Files

- “Instant-access” applications
  - Record must be located immediately
  - Transaction-processing systems require rapid access
- Random-access files
  - Access individual records directly and quickly
  - Use fixed length for every record
    - Easy to calculate record locations
  - Insert records without destroying other data in file

Java's View of Random-Access File



Creating a Random-Access File

- **`RandomAccessFile`** objects
  - Like **`DataInputStream`** and **`DataOutputstream`**
  - Reads or writes data in spot specified by file-position pointer
    - Manipulates all data as primitive types
    - Normally writes one object at a time to file
- Note Figure 17.12, a new class derived from **`AccountRecord`**

Creating a Random-Access File

- **`RandomAccessAccountRecord`** of Figure 17.12 adds features
  - Two constructors
  - Routine to read a record
  - Routine to write a record
- These will be used in the application to create a Random file, Figure 17.13
  - Window to name and open the new file
  - Program generates 100 empty records

Writing Data Random-Access File

- Figure 17.14
  - Opens the file previously created
  - First window has Open button, Open dialog box appears
  - Uses `BankUI` graphical interface
  - Click Enter button to save the record
- `RandomAccessFile` method `seek`
  - Determines location in file where record is stored
  - Sets file-position pointer to a specific point in file

Reading from Random-Access File

- Read sequentially through random-access file, Figure 17.15
  - Opens the `RandomAccessFile`, specify file name with Open file dialog box
  - Record displayed in `BankUI` window
  - Click on Next button to view next record
  - `try`, `catch` combination handles end of file

Transaction-Processing Program

- Substantial transaction-processing system, Figure 17.21 class with methods for
  - Uses random-access file
  - Updates, adds and deletes accounts
- Note
  - Creation of `FileEditor` object
  - Handling of action buttons
  - Creating new records
  - Updating, deleting records
  - Displaying records in text fields

Transaction-Processing Program

- FileEditor program, Figure 17.22
- Uses the `TransactionProcessor` class
- Features
  - Create `RandomAccessFile` from name provided
  - Position file pointer
  - Read, edit,
  - Reposition file pointer, update record
  - Delete record by overwriting with blank record