Resource - request algo for process $P_i$.

Request = request vector for process $P_i$.

If $Request_i(j) = k$ then process $P_i$ wants k instances of resource type $R_j$.

1) If $Request_i \leq Need_i$ go to step 2. otherwise raise error condition, since process has exceeded its maximum claim.
2) If $Request_i \leq Available$, go to step 3. Otherwise $P_i$ must wait, since resources are not available.
3) Allocate requested resources to $P_i$ by modifying the state as follows.

$Available = Available - Request_i$;
$Allocation_i = Allocation_i + Request_i$;
$Need_i = Need_i - Request_i$;

If safe ⇒ the resources are allocated to $P_i$.

If unsafe ⇒ $P_i$ must wait, and the old resource-allocation state is restored.

4) A → 10 instances B → 5 instances C → 7 instances
($P_1, P_3, P_2, P_0, P_4$)

| | Allocation | | | Max | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| $P_0$ | 0 | 1 | 0 | 7 | 5 | 3 | 2 | 1 | 1 |
| $P_1$ | 2 | 1 | 1 | 3 | 2 | 2 | | | |
| $P_2$ | 3 | 0 | 2 | 9 | 0 | 2 | | | |
| $P_3$ | 2 | 1 | 1 | 2 | 2 | 2 | | | |
| $P_4$ | 1 | 1 | 2 | 4 | 3 | 3 | | | |

Need

| | A | B | C |
|---|---|---|---|
| $P_0$ | 7 | 4 | 3 |
| $P_1$ | 1 | 1 | 1 |
| $P_2$ | 6 | 0 | 0 |
| $P_3$ | 0 | 1 | 1 |
| $P_4$ | 3 | 2 | 1 |

$P_1$

Allocation

| A | B | C |
|---|---|---|
| 2 | 1 | 1 |

Available

| A | B | C |
|---|---|---|
| 2 | 1 | 1 |

Need

| | A | B | C |
|---|---|---|---|
| $P_1$ | 1 | 1 | 1 |

New all.

| A | B | C |
|---|---|---|
| 3 | 2 | 1 |

Available

| A | B | C |
|---|---|---|
| 1 | 0 | 0 |

New Available

| A | B | C |
|---|---|---|
| 4 | 2 | 2 |

$P_3$

Allocation

| A | B | C |
|---|---|---|
| 2 | 1 | 1 |

Available

| A | B | C |
|---|---|---|
| 4 | 2 | 2 |

Need

| A | B | C |
|---|---|---|
| 0 | 1 | 1 |

New All.

| A | B | C |
|---|---|---|
| 2 | 2 | 2 |

Available (during execution)

| A | B | C |
|---|---|---|
| 4 | 1 | 1 |

New Available after execution

| A | B | C |
|---|---|---|
| 6 | 3 | 3 |

| P2 | Allocation | | | Available | | | Need | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| | 3 | 0 | 2 | 6 | 3 | 3 | 6 | 0 | 0 |

New All. — Available

| A | B | C | A | B | C |
|---|---|---|---|---|---|
| 9 | 0 | 2 | 0 | 3 | 3 |

New Avail

| A | B | C |
|---|---|---|
| 9 | 3 | 5 |

| P0 | Allocation | | | Available | | | Need | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| | 0 | 1 | 0 | 9 | 3 | 5 | 7 | 4 | 3 |

Not safe state

| P4 | Allo. | | | Available | | | Need | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| | 1 | 1 | 2 | 9 | 3 | 5 | 3 | 2 | 1 |

New Allo

| A | B | C |
|---|---|---|
| 4 | 3 | 3 |

Avai

| A | B | C |
|---|---|---|
| 6 | 1 | 4 |

New Available

| A | B | C |
|---|---|---|
| 10 | 4 | 7 |

Page No.
Date

Allocation Max Need

| | Allocation | | | Max | | | Need | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| P0 | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 |

New allocation Available

| A | B | C | A | B | C |
|---|---|---|---|---|---|
| 7 | 5 | 3 | 3 | 0 | 4 |

New Available

| A | B | C |
|---|---|---|
| 10 | 5 | 7 |

* methods for handling deadlocks
  - Deadlock prevention
    - mutual exclusion
    - Hold and wait
    - no preemption
    - circular wait

Deadlock avoidance
- Resource allocation graph (with claim edge)

Banker's algorithm:
- safe state
- Data structure used
  - safety algorithm
  - Resource request algorithm

Page No.
Date

- Deadlock detection :- Allow system to enter deadlock state
  Detection algorithm
  Recovery scheme

- Data structures used :-
  Available :- A vector of length m indicates no of available resources of each type.
  Allocated : An n x m matrix defines no of resources of each type currently

  Request :

- Detection algorithm : 1) Let work and Finish be vectors of length m and n, respectively
  Initialize
  a) ~~Work~~ work = Available
  b) For i = 1, 2, ..., n, if Allocation$_i$ ≠ 0 then Finish[i] = false, else Finish[i] = true
  2) find an index i such that both
  Finish[i] == false
  Request$_i$ ≤ work
  If no such i exists, go to step 4
  3 = work = work + Allocation$_i$
  Finish[i] = true
  go to step2
  4) If Finish[i] = false, for some i, 1 ≤ i ≤ n, then the system is in deadlock state

| | Allocation | | | Request | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C |
| P0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P1 | 2 | 0 | 0 | 2 | 0 | 2 | | | |
| P2 | 3 | 0 | 3 | 0 | 0 | 0 | | | |
| P3 | 2 | 1 | 1 | 1 | 0 | 0 | | | |
| P4 | 0 | 0 | 2 | 0 | 0 | 2 | | | |

1) In this, work = [0, 0, 0] and finish = [false, false, false, false, false]

⟨P0, P2, P1, P3, P4⟩

---

P0

| New Available | | |
|---|---|---|
| A | B | C |
| 0 | 1 | 0 |

---

P2

| New Available | | |
|---|---|---|
| A | B | C |
| 3 | 1 | 3 |

---

P1

| New Available | | |
|---|---|---|
| A | B | C |
| 5 | 1 | 3 |

| New Allocation | | | Request | | | Available | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | A | B | C | A | B | C |
| 4 | 0 | 2 | 2 | 0 | 2 | 1 | 1 | 1 |

New Available

| A | B | C |
|---|---|---|
| 5 | 1 | 3 |

B3

| New All. | | | Request | | | Available | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | A | B | C | A | B | C |
| 3 | 1 | 1 | 1 | 0 | 0 | 5 | 1 | 3 |

Available

| A | B | C |
|---|---|---|
| 4 | 1 | 3 |

New Avail

| A | B | C |
|---|---|---|
| 7 | 2 | 4 |

P4

| New All. | | | Req | | | Available | | |
|---|---|---|---|---|---|---|---|---|
| A | B | C | A | B | C | A | B | C |
| 0 | 0 | 4 | 0 | 0 | 2 | 7 | 2 | 4 |

Available

| A | B | C |
|---|---|---|
| 7 | 2 | 2 |

new Avail

| A | B | C |
|---|---|---|
| 7 | 2 | 6 |

* Recovery from Deadlock :

1) Process termination
2) Resource preemption

→ Abort all deadlock processes
Abort one process at a time until deadlock cycle is eliminated

Page No.
Date

In which order to abort?
- Priority of process
- How long process has computed and how much longer to completion.
- Resources the process has used
- Resources the process needs to complete its task.
- How many processes will ~~need~~ to be terminated.
- Is process interactive or batch?

---

Q1) d) ✓

Q2) c) X

Q3) a) X

Q4) a) ✓

Q5) c) ✓

Q6) a) ✓

Q7) a) ✓

Q11) d) X

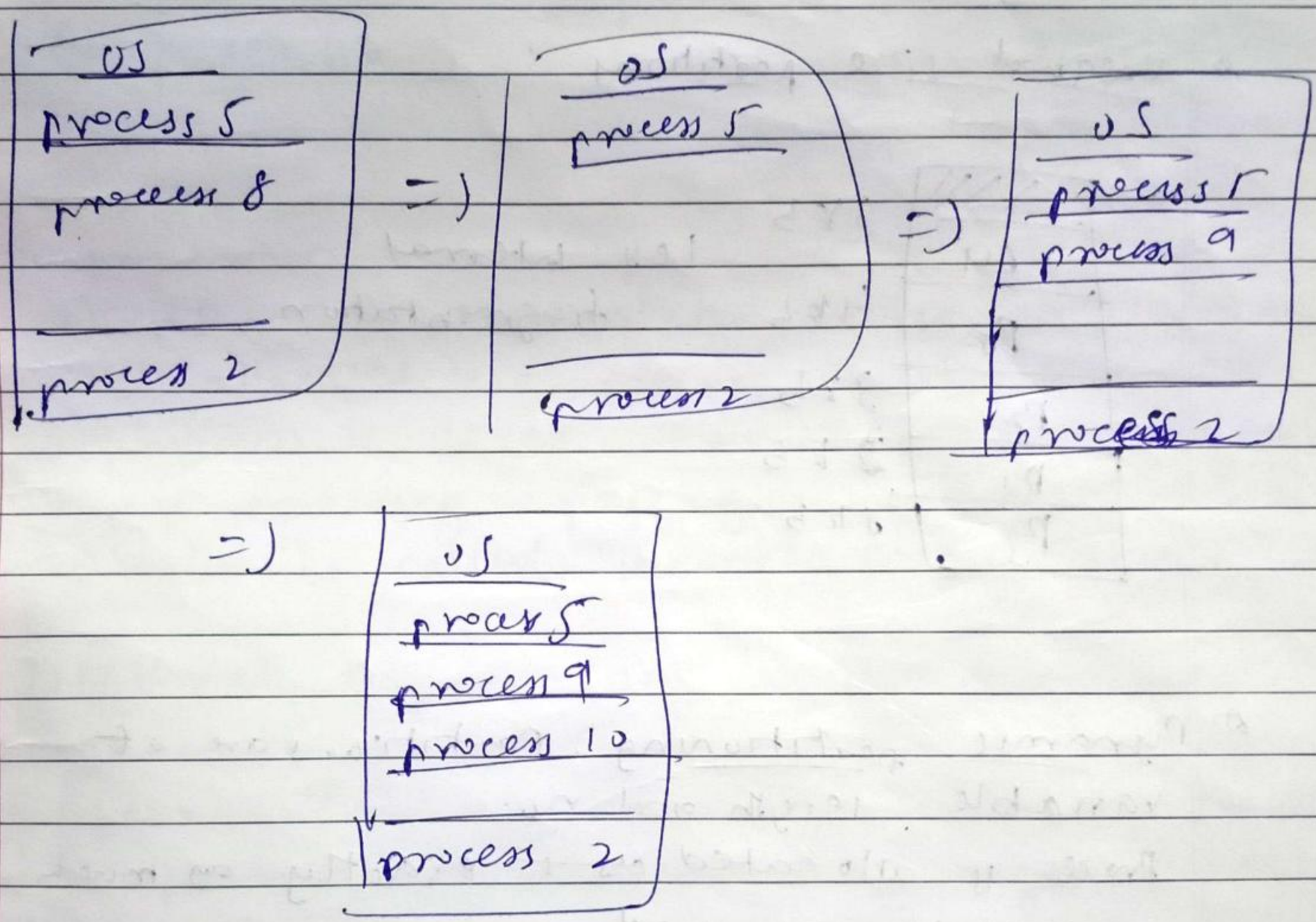
Q12) c) ✓

Q13) a) ✓

Page No.
Date

## Contiguous Allocation

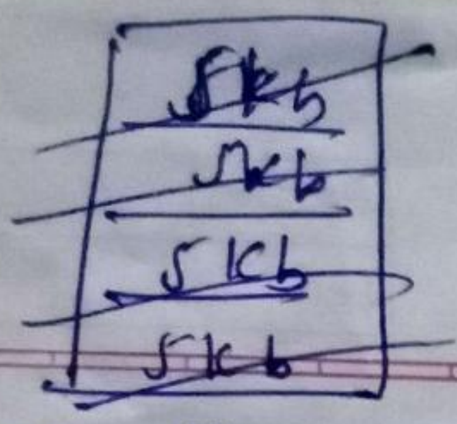
- Hole - block of available memory
when a process arrives, it is allocated memory from a large hole enough to accomodate it
operating system maintains info. about:
a) allocated partitions b) free partitions (holes)

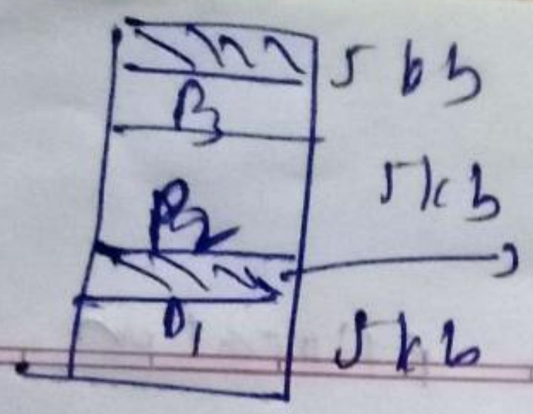



- Partitioning: 1) Fixed partitioning
2) Dynamic partitioning

Fixed partitioning:- 1) Equal-size partitions
2) Unequal-size partitions

Any process whose size is less than or equal to the partition size can be loaded into an available partition

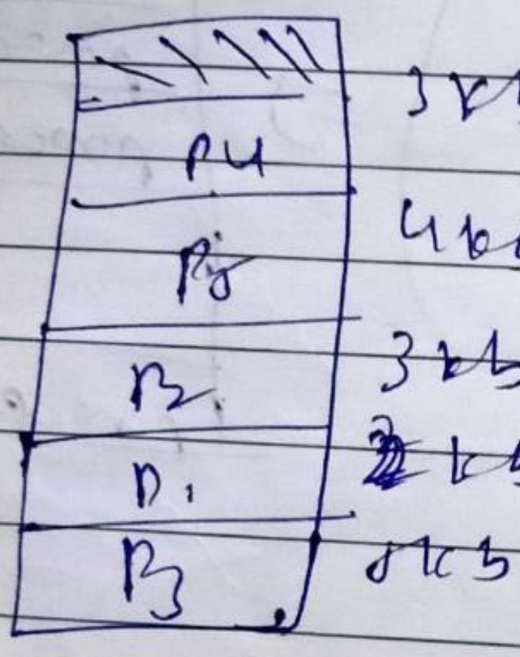





Internal fragmentation

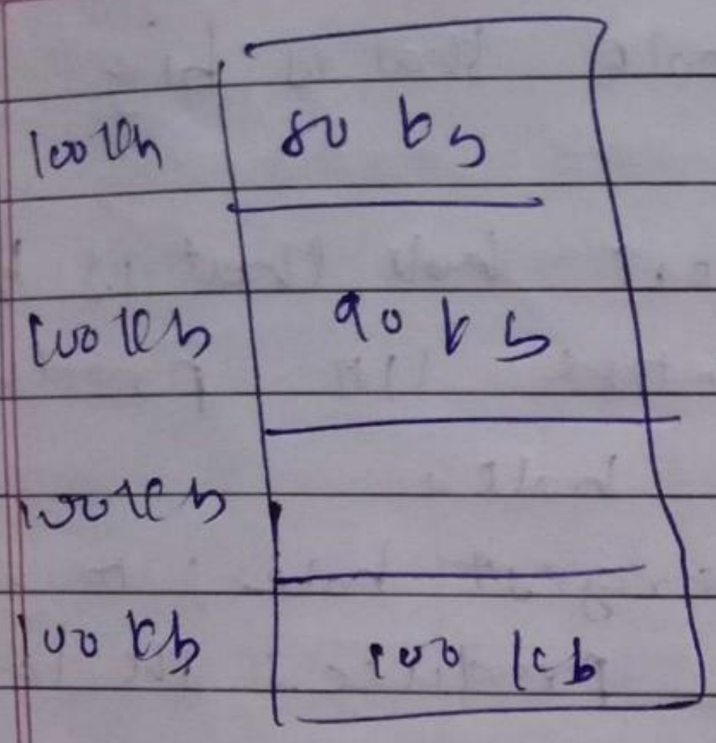
If all partitions are full the OS can swap a process out of a partition.
main memory use is inefficient. Any program no matter how small, occupies an entire partition. This is called internal fragmentation

* Unequal size partitions :




less internal fragmentation

* Dynamic partitioning : Partitions are of variable length and no.
Process is allocated exactly as much memory is required.
Eventually get holes in the memory. This is called external fragmentation.
Must use compaction to shift processes so they are contiguous and all free memory is in one block.




20 kb free
10 kb free
Internal fragmentation

Program size is 110 kb
External fragmentation.

* Fragmentation

1) Internal fragmentation: when a program is allocated to a memory block, if that program is lesser than this memory block and remaining space goes wasted, this situation is called internal fragmentation.

2) External fragmentation :- when a memory cannot be allocated to a requesting process because the memory blocks are very small. The process size is more than any available hole.

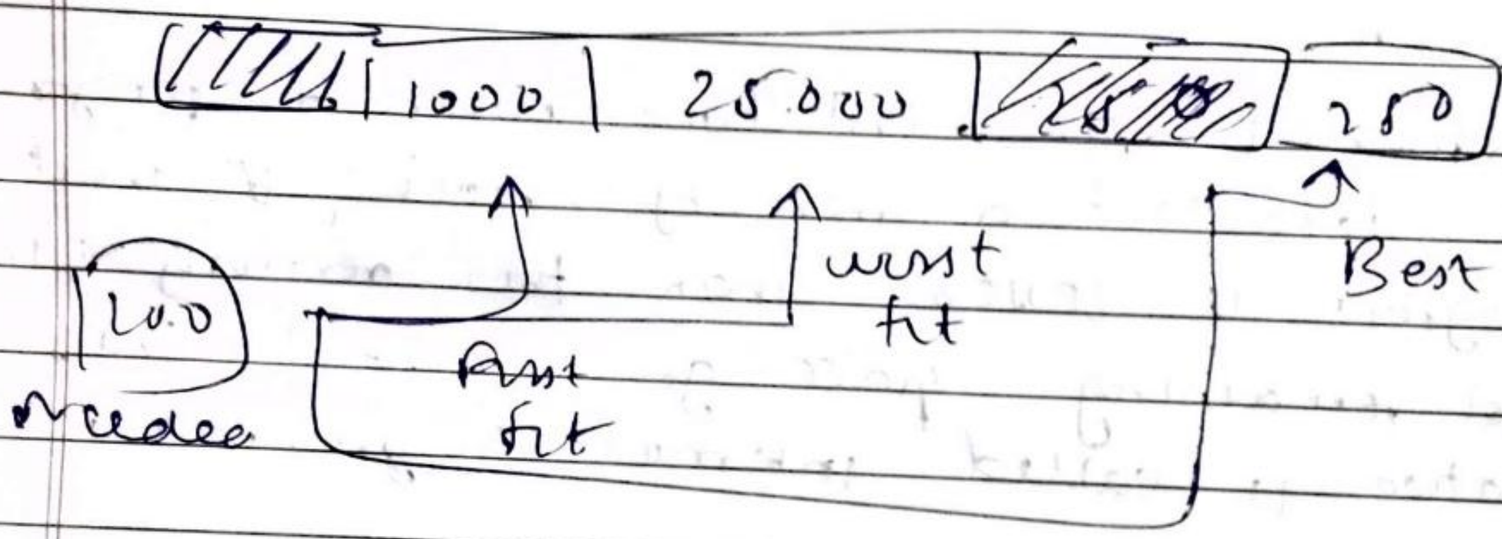
Reduce external fragmentation by compaction
- Shuffle memory contents to place all free memory together in one large block
- Compaction is possible only if relocation is dynamic, and is done at execution time.

* Dynamic storage - Allocation problem :-
How to satisfy a request of size n from a list of free holes

First fit :- Allocate the first hole that is big enough.

Best fit :- Allocate the smallest hole that is big enough. must search entire list. produces the smallest leftover hole.

worst fit :- Allocate the largest hole ; must also search entire list. produces the largest leftover hole



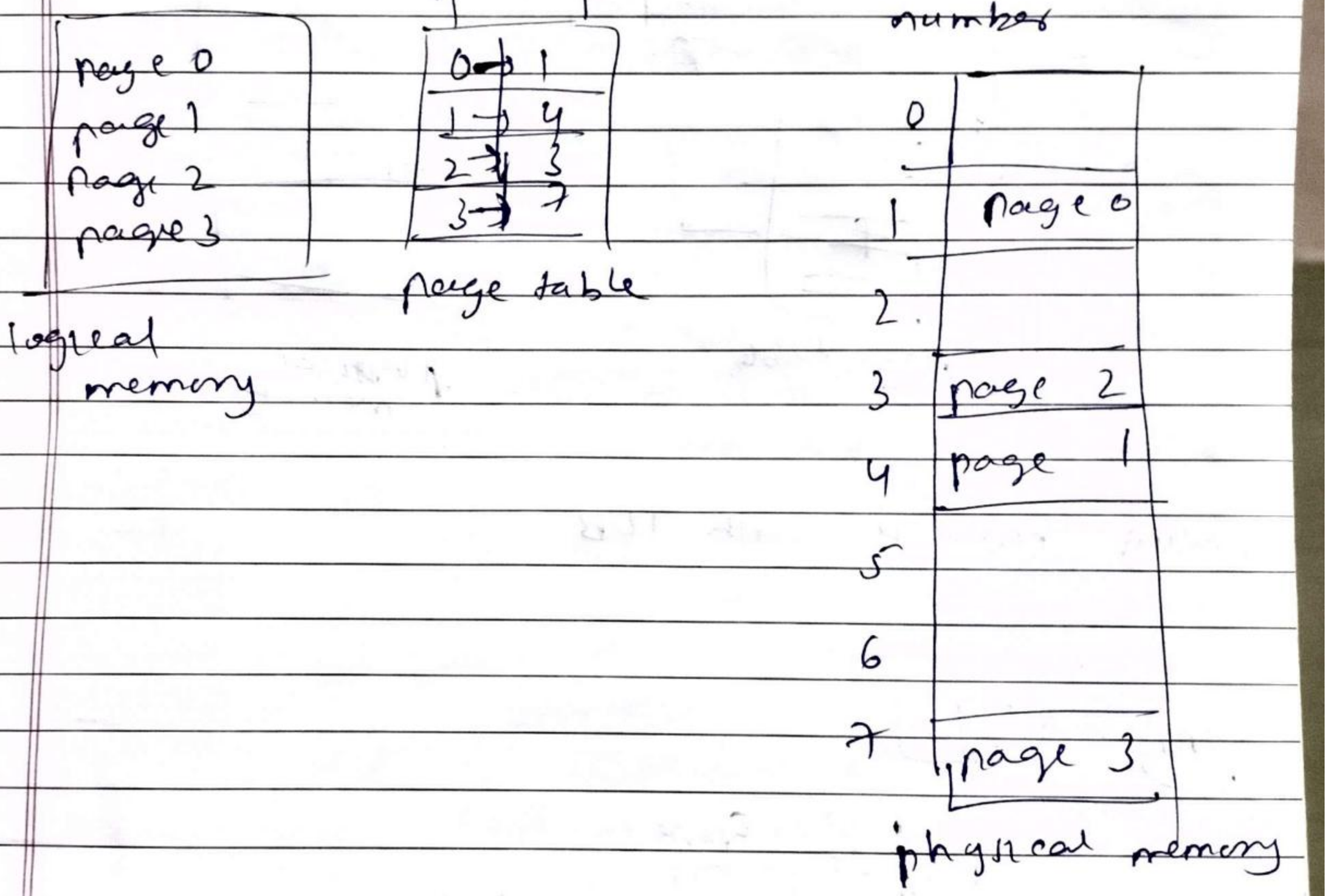

Paging :- Paging is a memory management scheme that permits the physical address space of a process to be non contiguous. Divide physical memory into fixed-size blocks called frames. Divide logical memory into blocks of same size called pages.

when a process is to be executed its pages are loaded loaded into any available memory frames. Set up a page table to translate logical to physical addresses. Reduces internal fragmentation

Address generated by CPU is divided into:

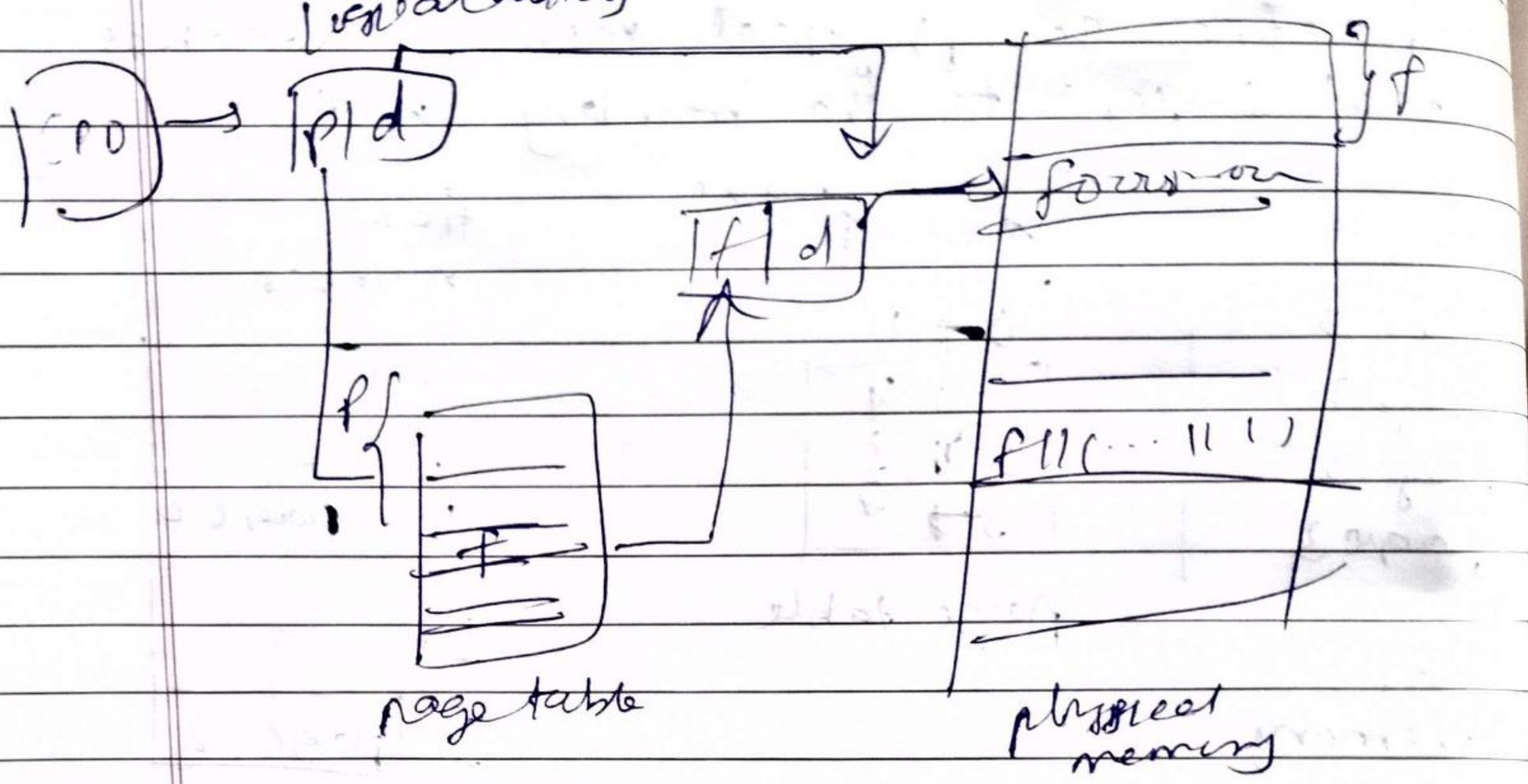
Page number (p) :- used as an index into a page table which contains base address of each Page in physical memory.

page offset (d) - combined with base address to define the physical memory address that is sent to the memory unit.

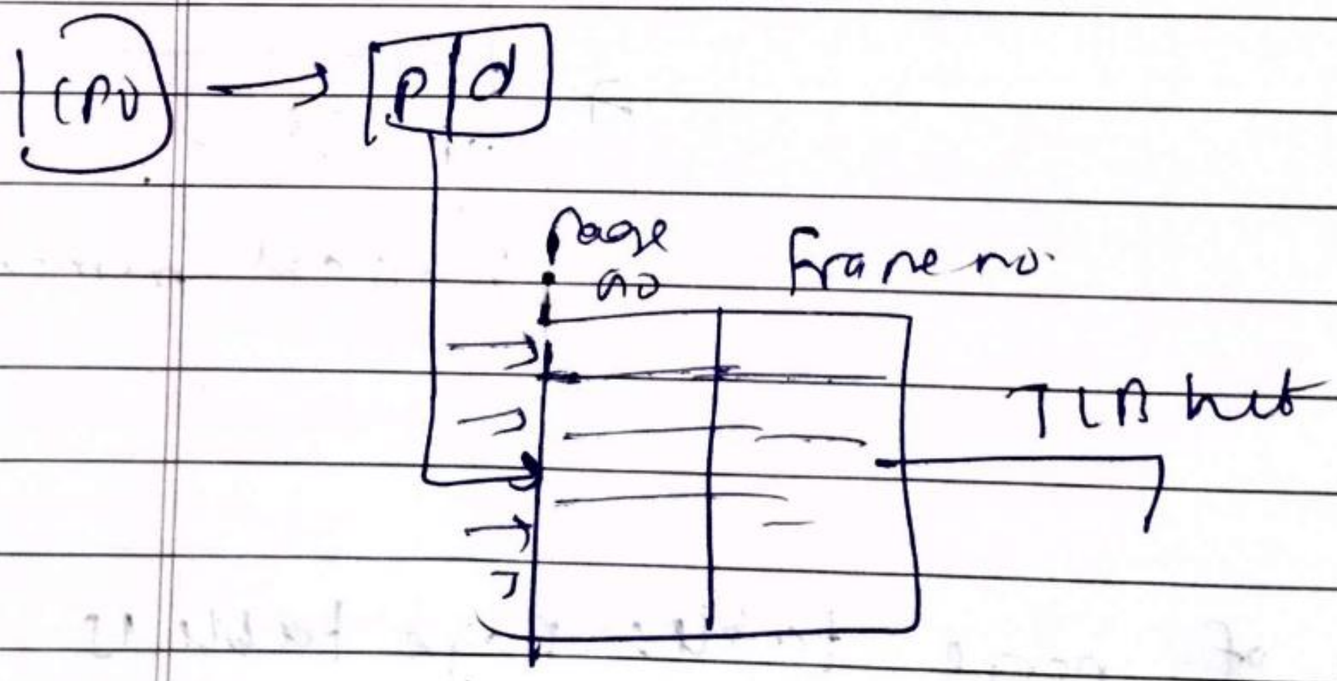



* ~~free frames~~ :-

* Implementation of page table :- Page table is kept in main memory.
Page table base register (PTBR) points to the page table.
Page table length register (PTLR) indicates size of the page table.
In this scheme every data/instruction access requires 2 memory accesses. one for the page table and one for the data/instruction



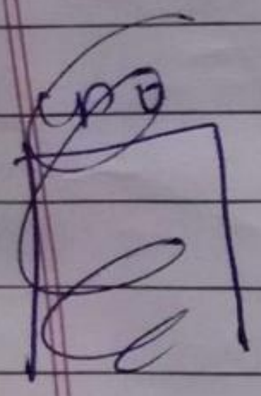

paging hardware with TLB



Page No.
Date

- page table is kept in main memory
page table base register (PTBR) points to the page table
page tube

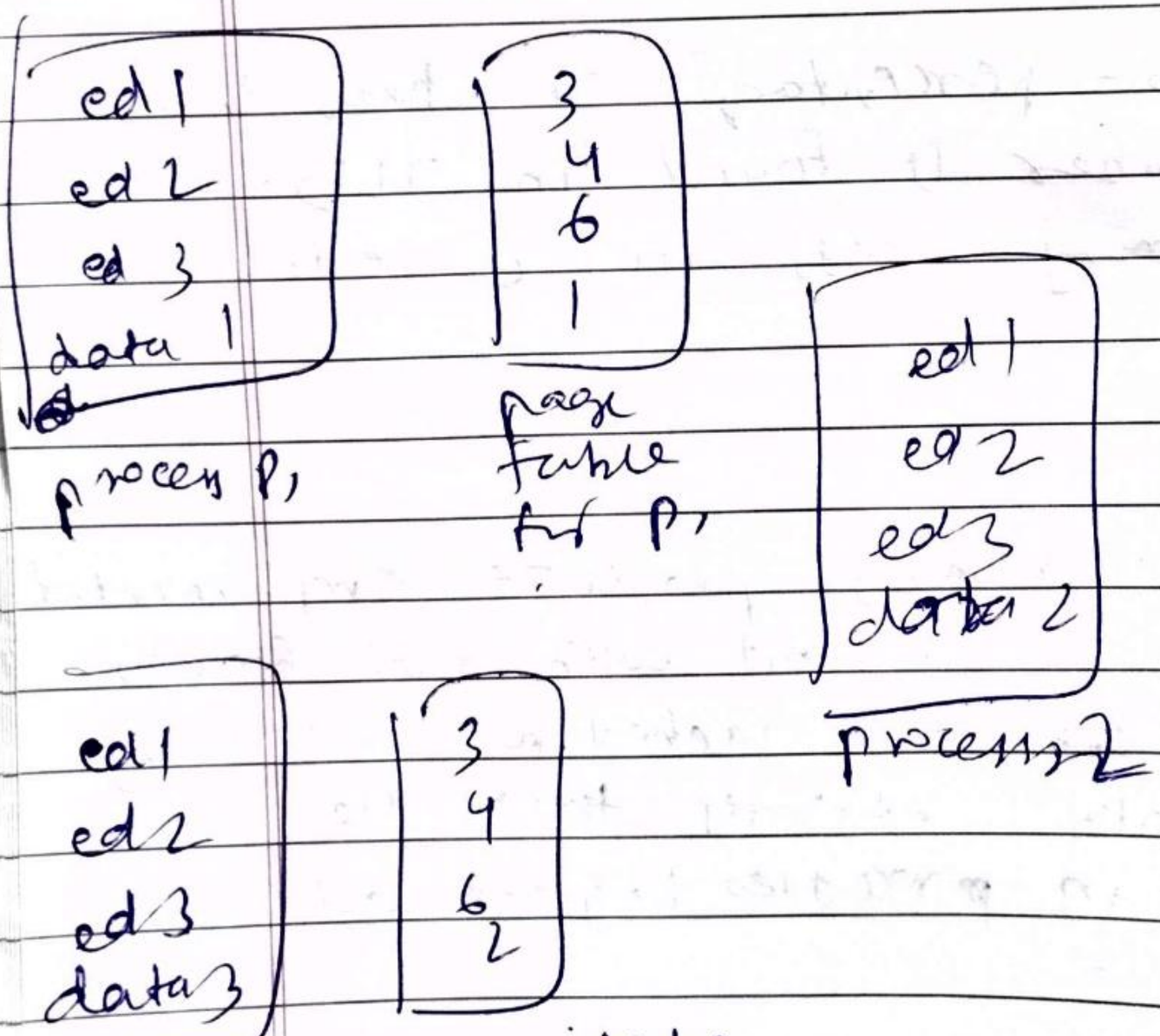
This ↑ the time of access (as size of page table grow)
This problem can be solved by the use of a special lookup hardware called as TLB
Hit ratio - percentage of times that a page number is found in TLB.
Effective memory access time is reduced by TLB.



memory protection: memory protection implemented by associating protection bit with each frames
Valid invalid bit attached to each entry in page table: "valid" indicates that the associated page is in processes logical address space
"invalid" also is present that the associated page is not in processes logical address space.

- Shared pages:- Shared code

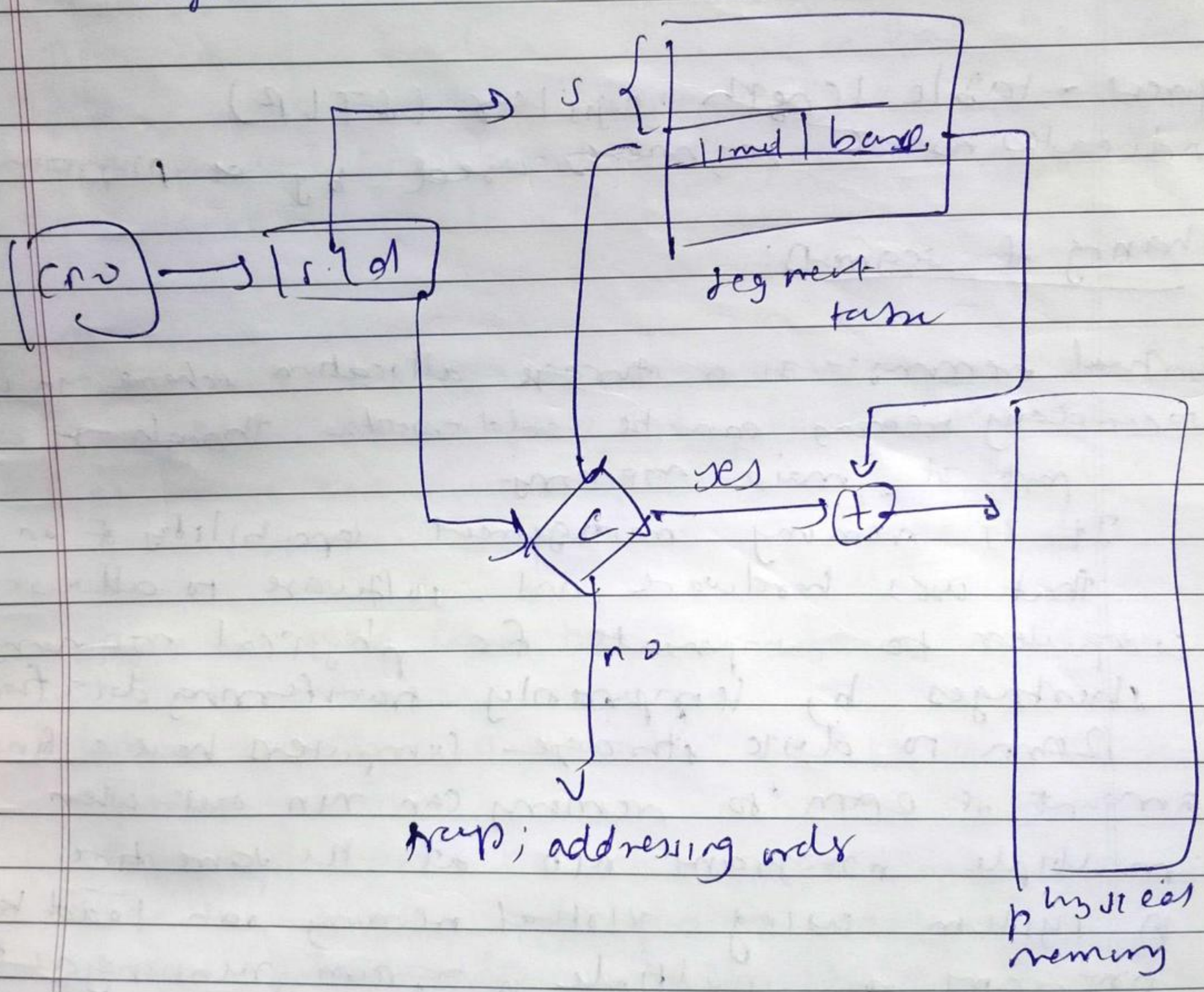
One copy of shared ~~code~~ among processes
shared code must appear in same location in
the logical address space of all processes
private code and data.
- Each process keeps a separate copy of the
code and data.
The pages for private code and data can
appear anywhere in logical address space



Page No.
Date

* Segmentation: A memory management technique in which memory is divided into variable sized chunks which can be allocated to processes each chunk is called a segment

segmentation hardware




segmentation architecture.

logical address consists of 2 tuple
(segment-number, offset)

segment table - maps 2 dimensional physical addresses each table entry has

base - contains the starting physical

address where the segments reside in memory.
limit - specifies the length of the segment.

segment - table base register (STBR)
points to segment table's location in memory.

segment - table length register (STLR)
indicates no. of segments used by a program

* Sharing of segments:

Virtual memory - Is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory.
It is memory management capability of an OS that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from RAM to disk storage. Computers have a finite amount of RAM so memory can run out when multiple programs run at the same time. A system using virtual memory can load larger programs or multiple programs running at the same time, allowing each one to operate as if it has infinite memory and without having to purchase more RAM.
The OS divides memory into page files or swap files. Each page is stored on a disk and when the page is needed, the OS copies it from the disk to main memory and translates virtual

addresses into real addresses.

Page fault :- what happens if a process tries to access page that was not brought into memory? Access to a page marked invalid causes a page fault

The paging hardware will notice that the invalid bit is set, causing a trap to the OS.
This trap is the result of the OS failing to bring the desired page to the memory.

steps
Trap to OS
Save the user registers and process state
Determine that interrupt was a page fault
Determine the location of the page on disk

Issue a read from the disk to a free frame

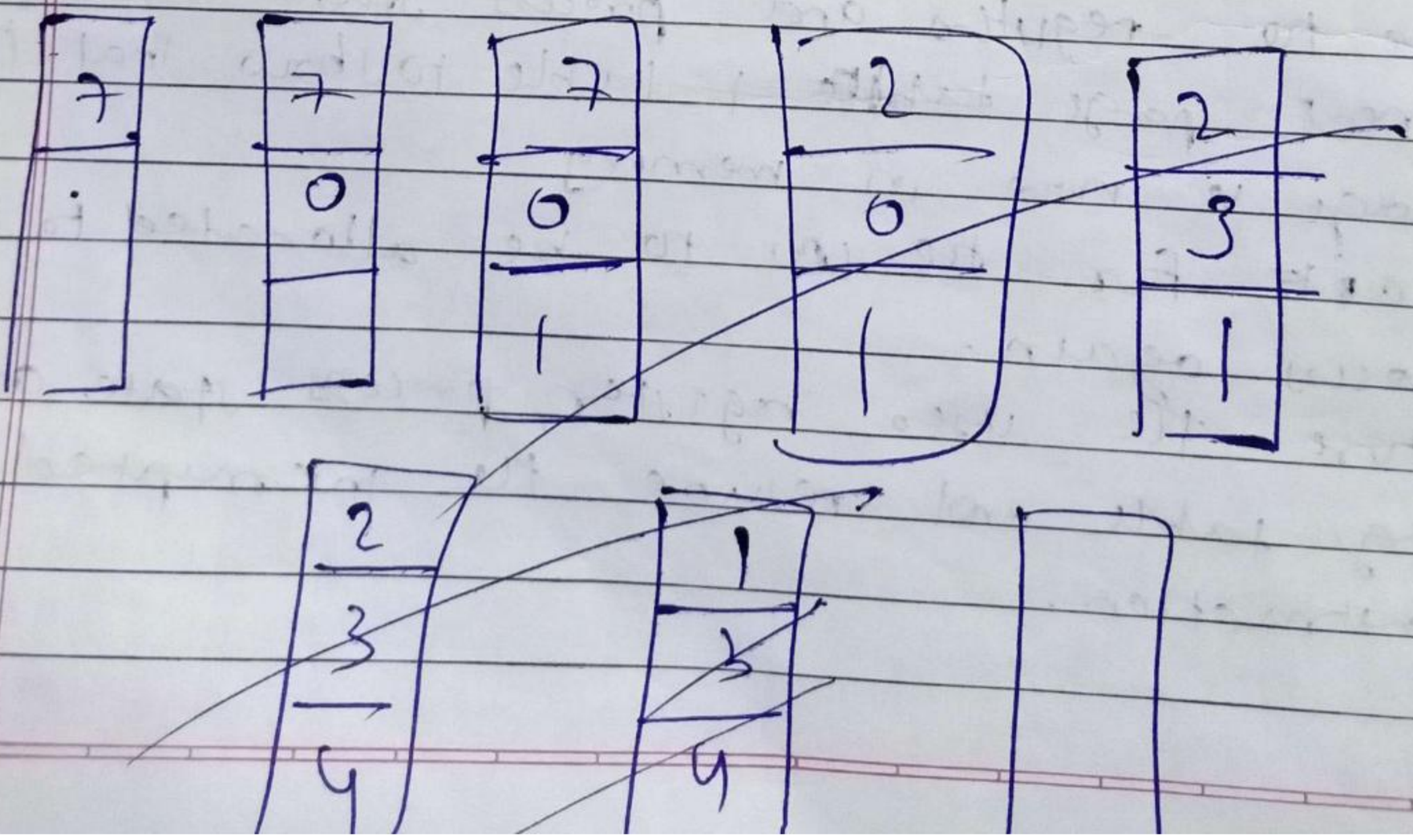
while waiting, allocate the CPU to some other process.
Interrupt from the disk (I/O completed)
Save the registers and process state
Correct page table table to show that the page is now in memory
wait for the CPU to be allocated to the process again.
Restore the user register, process state and new page table and resume the interrupted instruction.

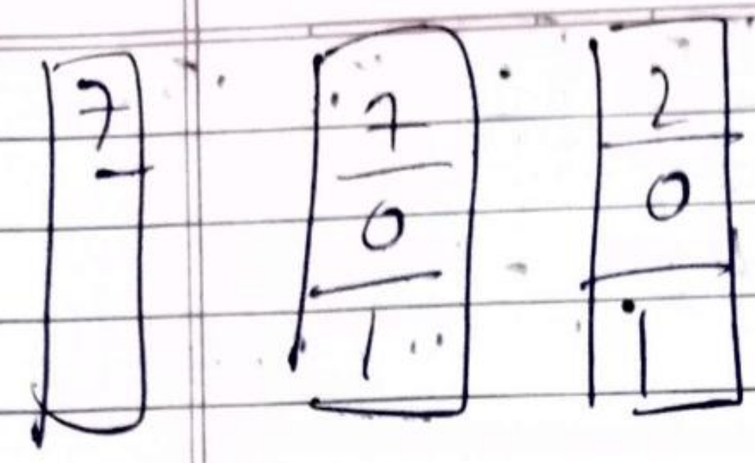
what happens if there is no free frame?
. page replacement - find some page in memory, but not really in use, swap it out
-algorithm
-performance - want an algorithm which will result in min. no. of page faults.

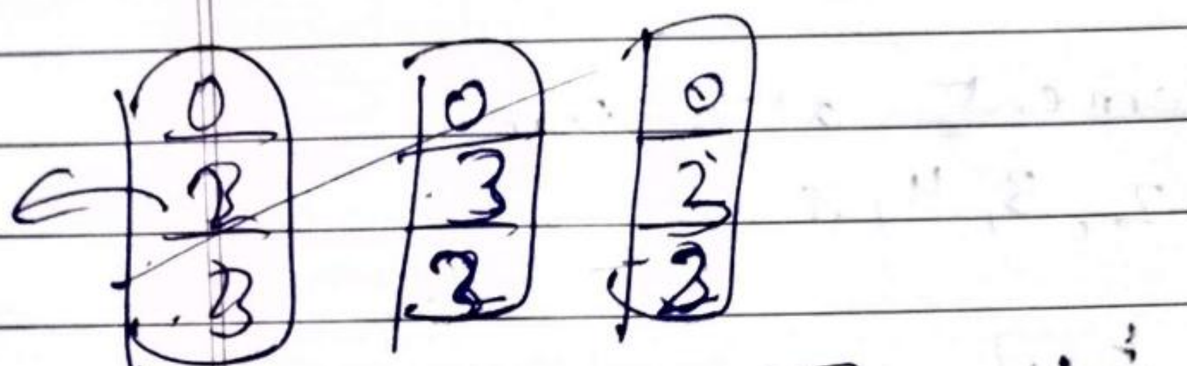
* Basic page replacement :- Find the location of the desired page on disk
And a free frame
- If there is a free frame, use it
- If there is no free frame, use a page replacement algorithm to select a victim frame.
3) Read the desired page into the (newly) free frame. Update the page table.
4) Restart the process

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1





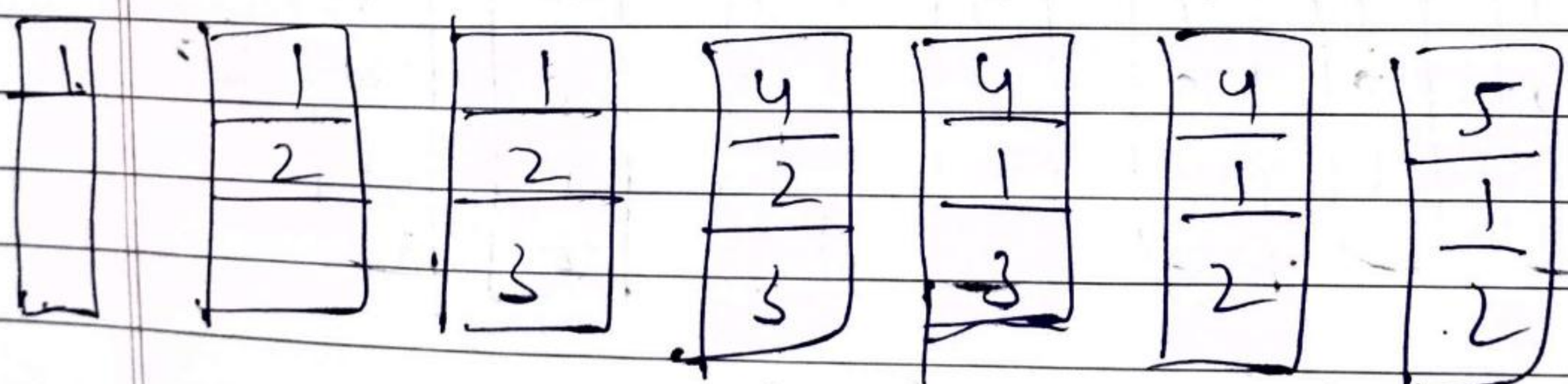


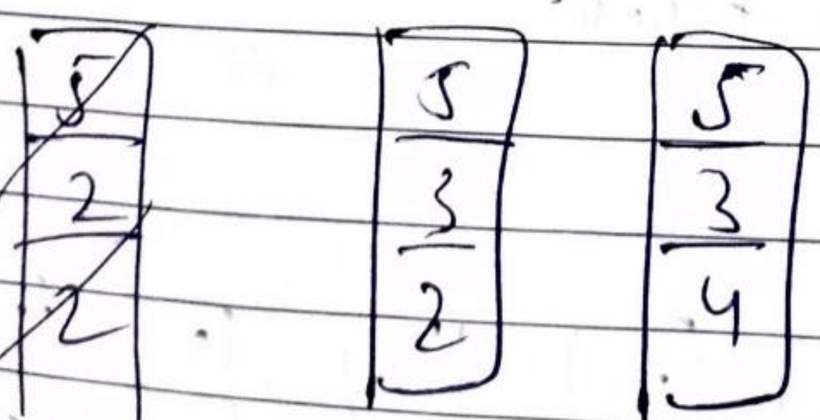
* FIFO

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames

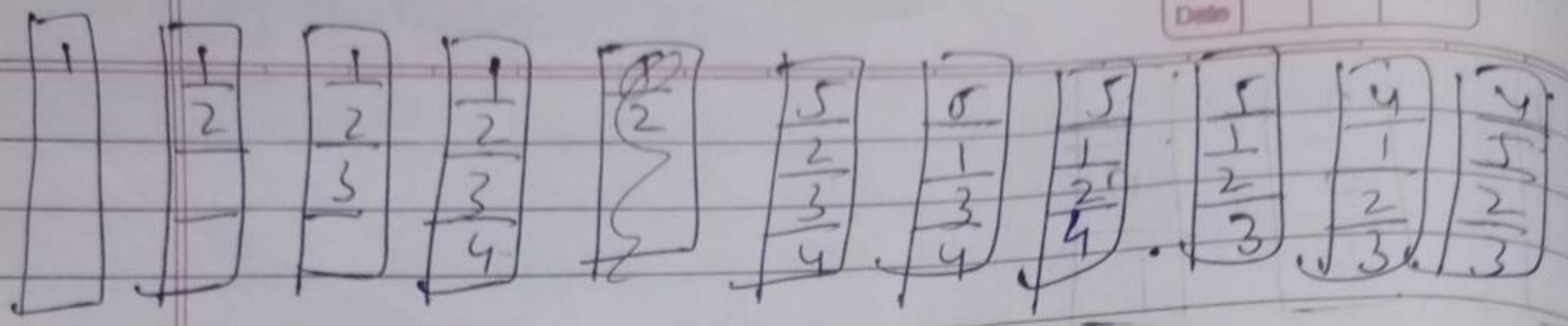
701





= 9 page faults

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 7

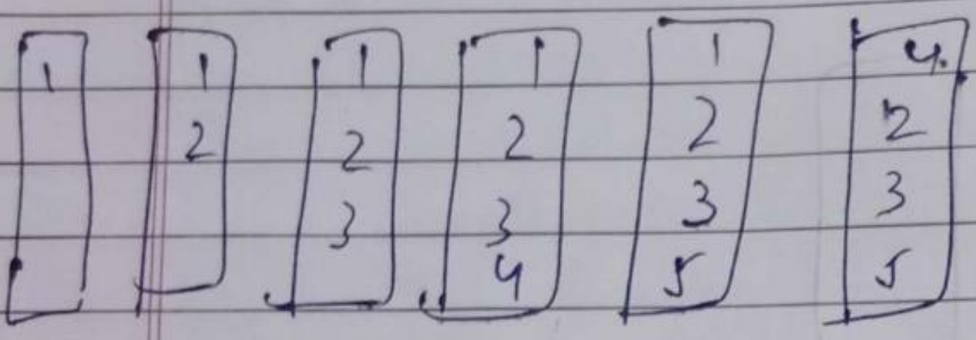


= 10

This is Bilady's anomaly. Page fault increases as frames ↑

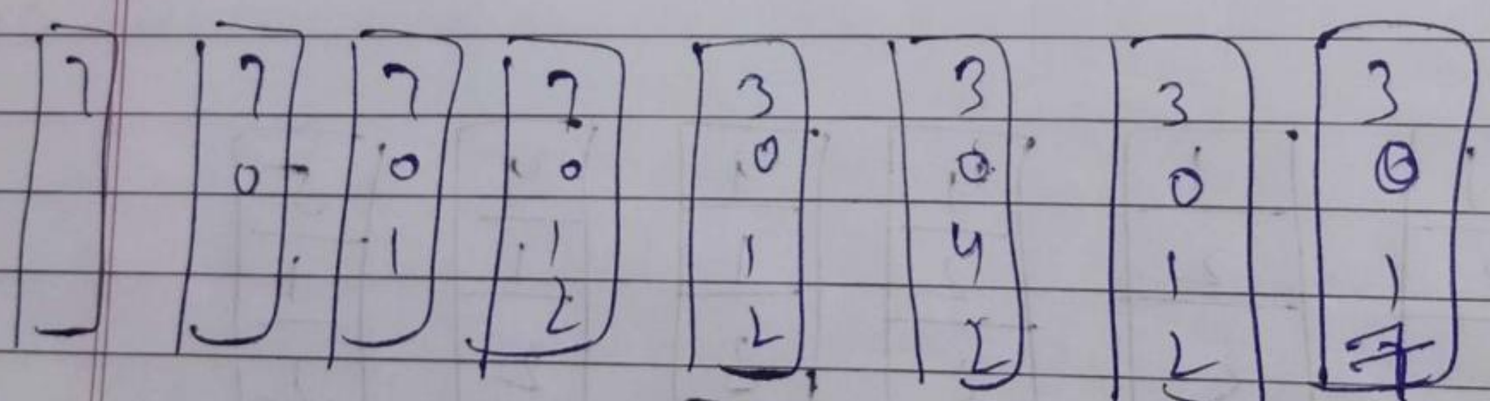
2) Optimal page replacement algorithm

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, 2, 3



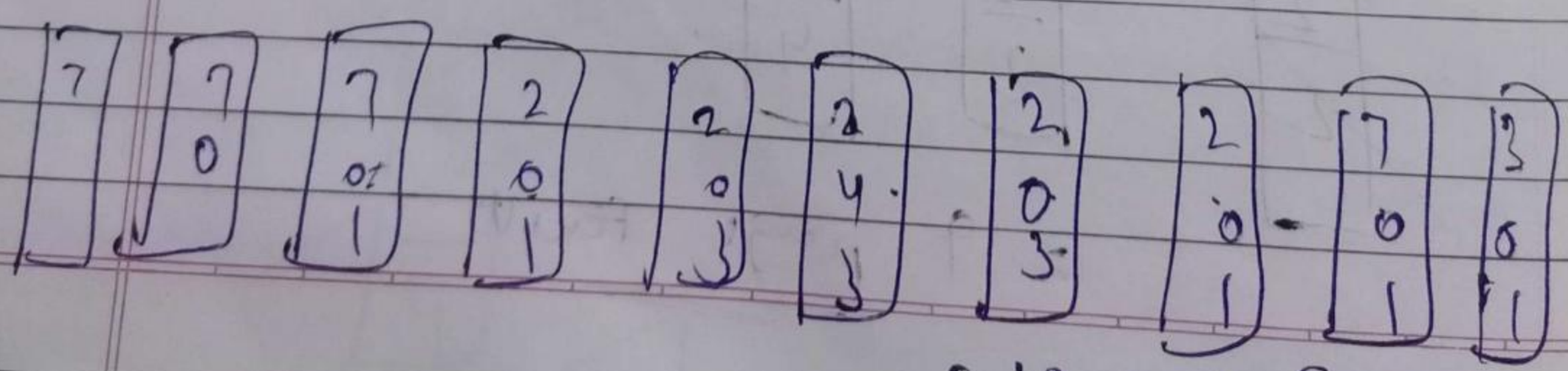
= 6 Page faults

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



= 8 page fault

3 frames

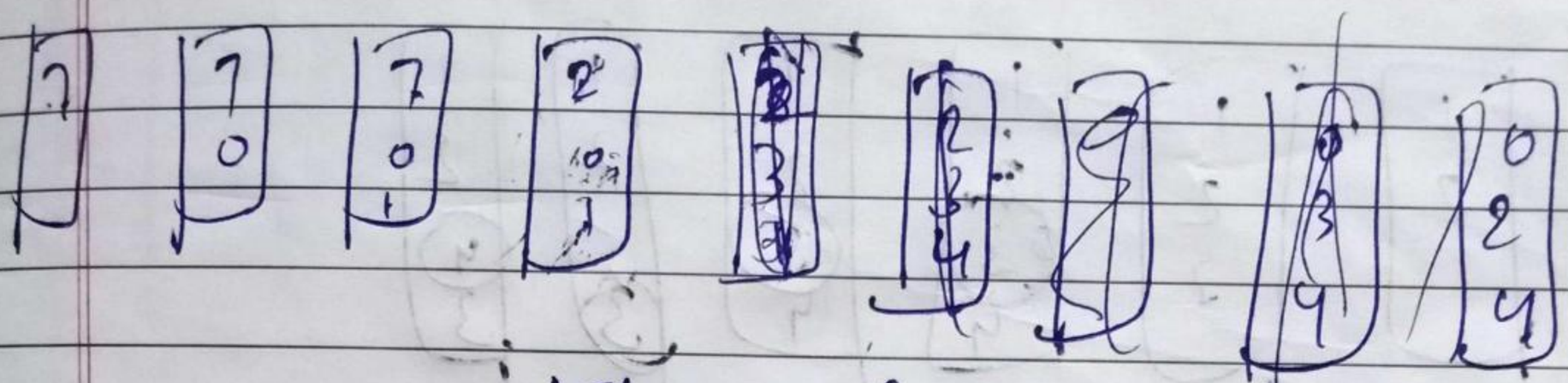


= 10 page faults.

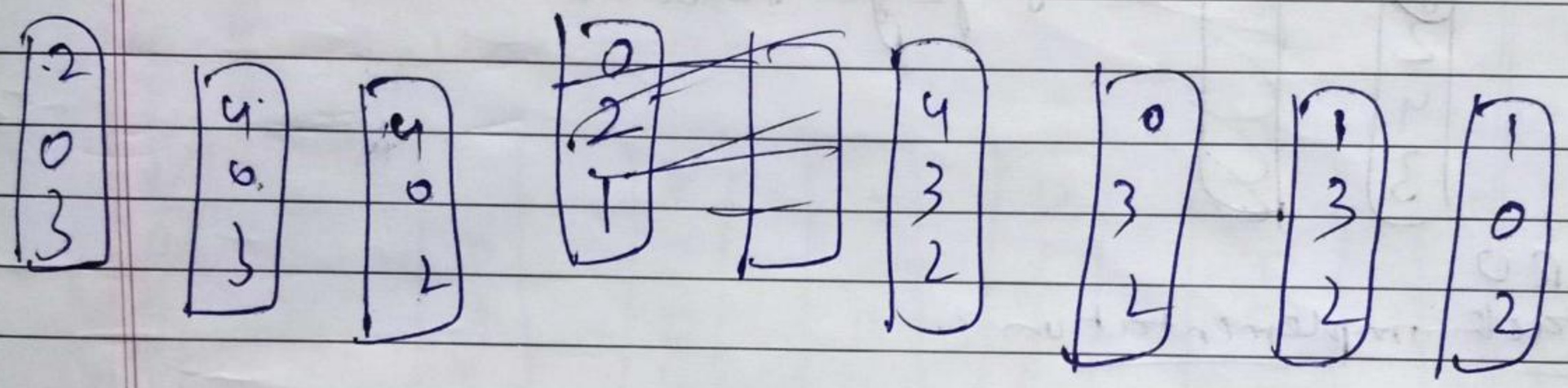
3) LRU (least recently used)

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

3 frames



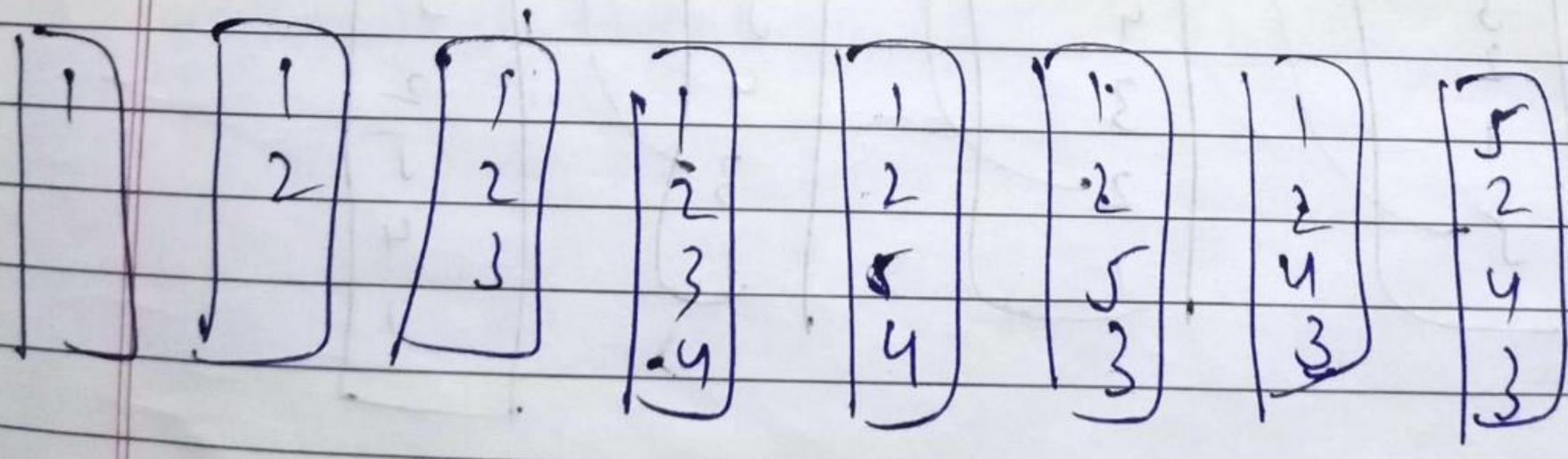
Page faults.



= 12 page faults

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

4 frames

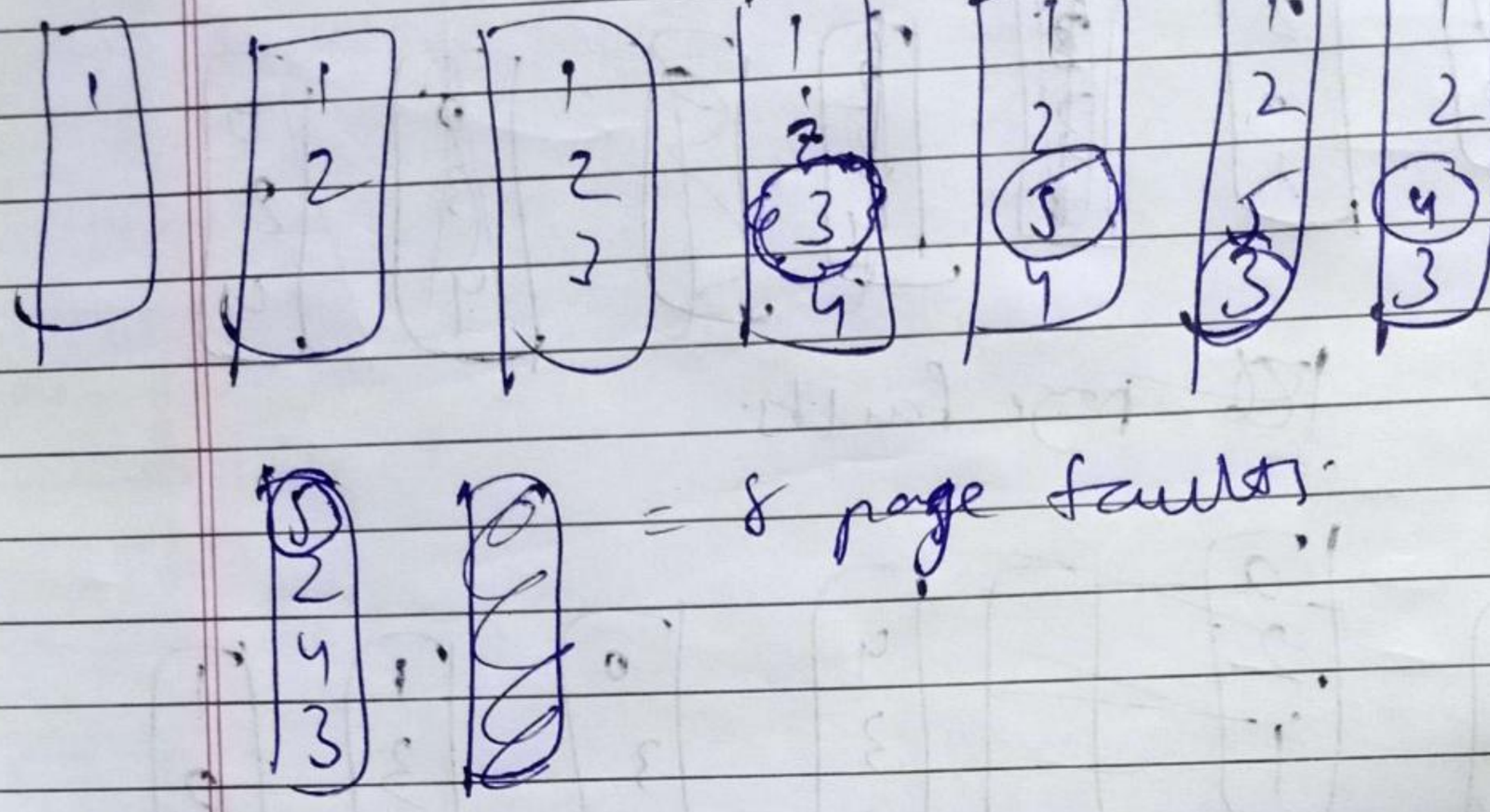


= 8 page faults

LRU implementation.

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, 2, 3

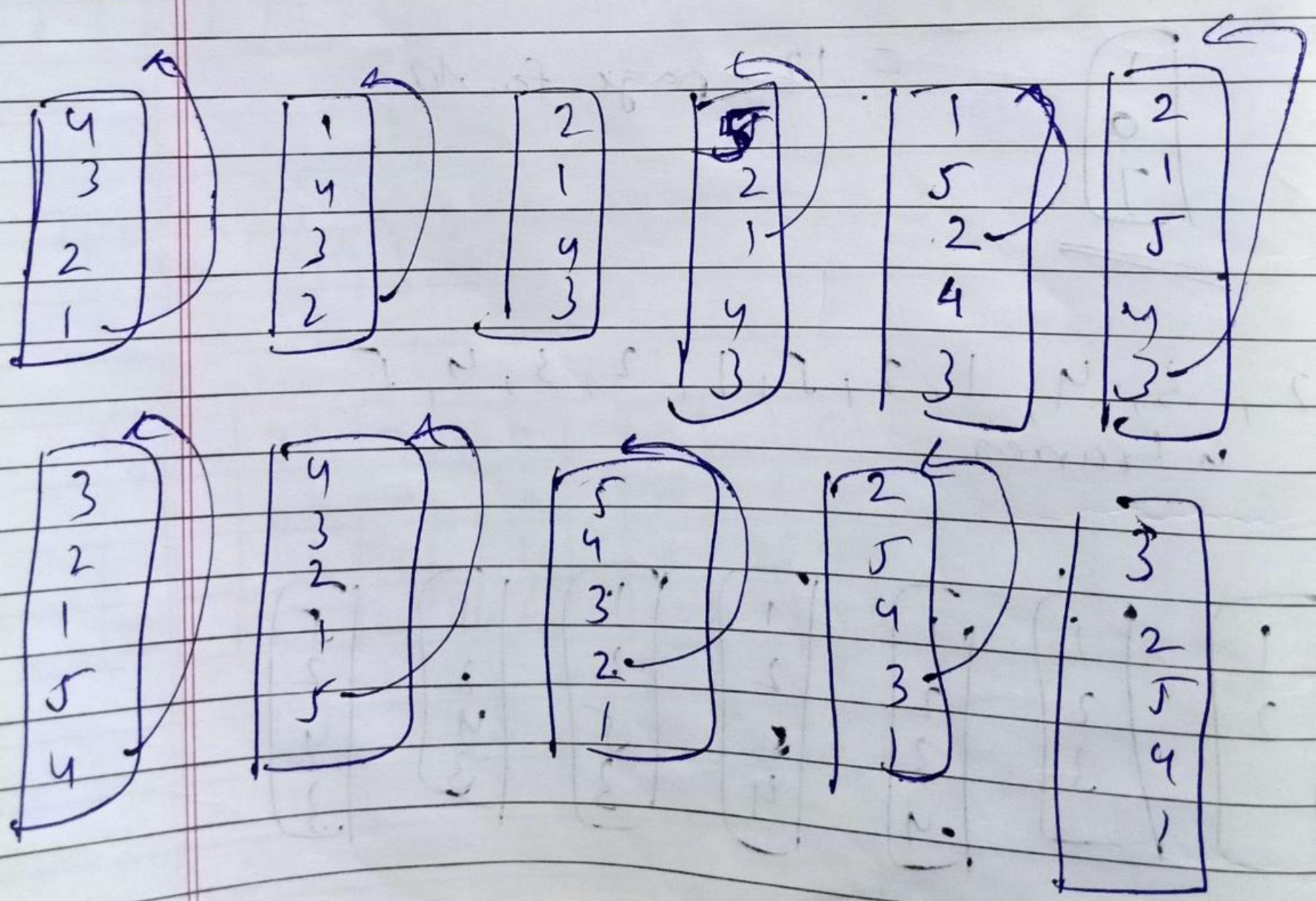
4 frames



= 8 page faults.

LRU

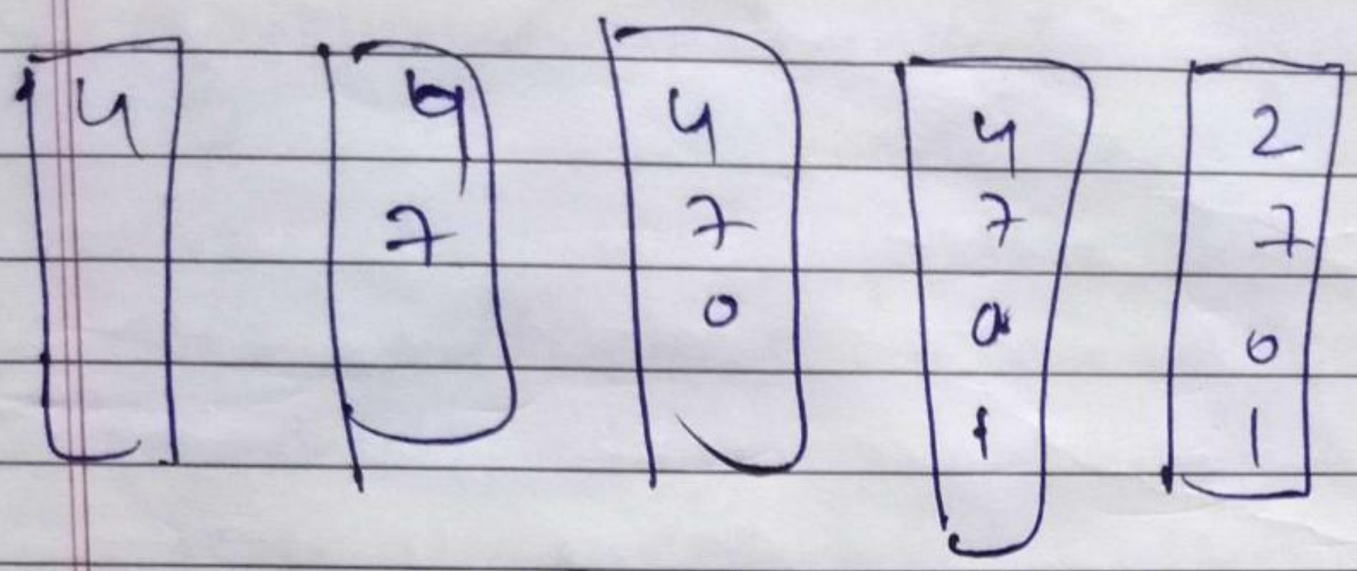
(1) Stack implementation:



LRU implementation

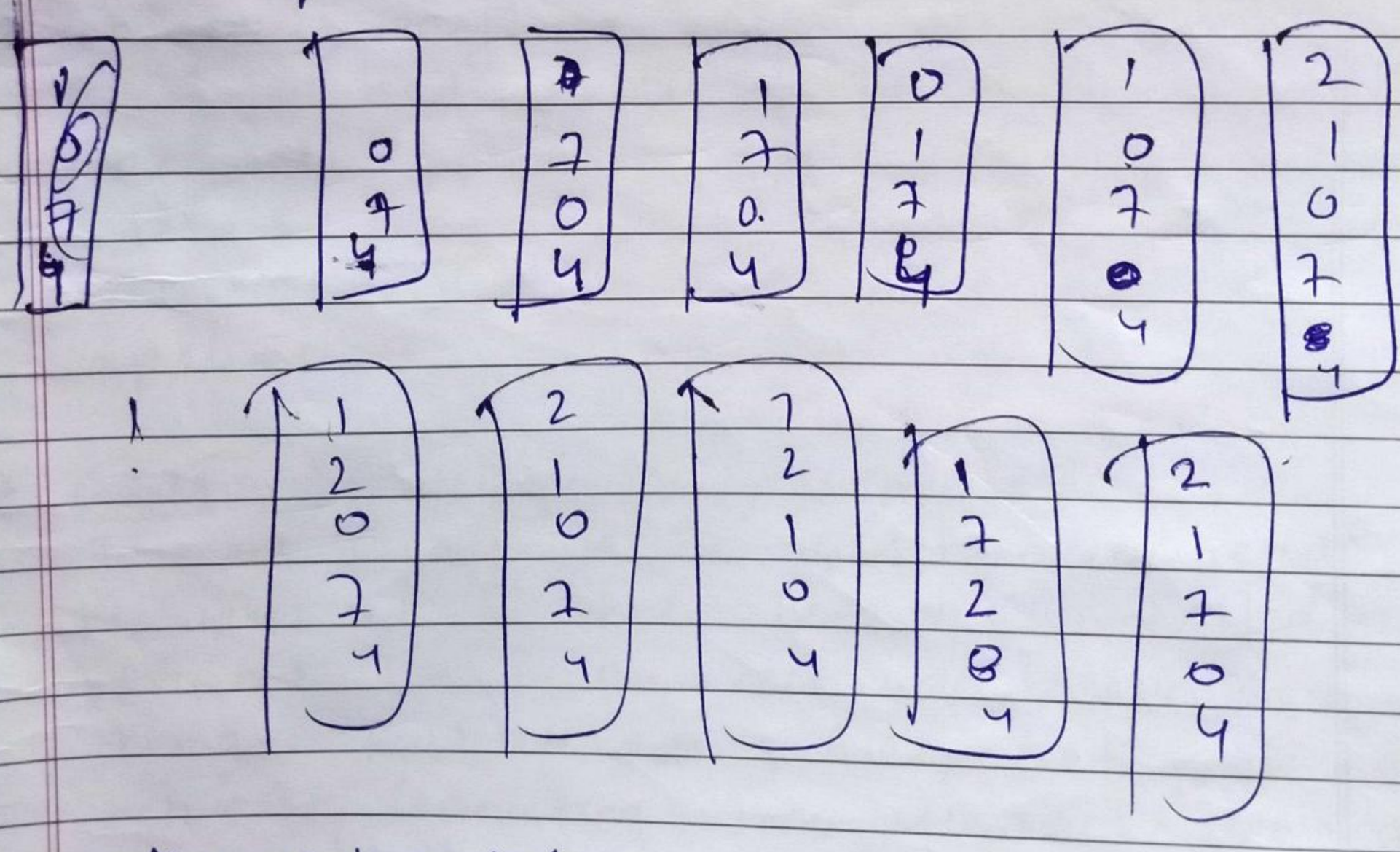
4, 7, 0, 7, 1, 0, 1, 2, 1, 2, 7

Page No. 12
Date



= 5 page fa

Stack implementation



A counter implementation:

Page No.
Date

Second chance algorithm

Page No.
Date

# Distributed system

Collection of independent computers that appears to its users as a single coherent system.
Consist of computers that are autonomous.
The differences between computers and the ways in which they communicate are hidden from the users. Users and applications can interact with a distributed system in a consistent and uniform way regardless of where and when interaction takes place. Distributed systems should also be relatively easy to expand or scale.
A distributed system will be continuously available. Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.

* Goals: 1) making Resources Available
To make it easy for the users to access remote resources and to share them in a controlled and efficient way. Resources can be printers, computers, storage facilities, data, files, web pages and networks. Connecting users and resources also makes it easier to collaborate and exchange information. As connectivity and sharing increases, security becomes important. So systems must provide protection against eavesdropping or intrusion or communication

VVVIP

* Distribution Transparency: To hide that the fact that its processes and resources are physically distributed across multiple computers.

A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

| Transparency | Description |
|---|---|
| Access | Hide differences in data representation and how a resource is accessed. |
| Location | Hide where a resource is located |
| Migration | Hide that a resource may move to another location |
| Relocation | Hide that a resource may be moved to another location while in use. |
| Replication | Hide that a resource is replicated |
| Concurrency | Hide that a resource may be shared by several competitive users |
| Failure | Hide the failure and recovery of a resource. |

2) 1) c) page ✓
2) a) internal ✗
3) b) hit ratio ✓
4) c) valid - invalid ✓
5) : has data in it ✗
6) c) segment length ✓
7 ores addr

16) c) ✓

17) a) ✓

a) b) To not execute anderlin from a deadlock ✗

b) ✓

20) c) ~~max - Allocated~~ ✓

sort by default ~~natural~~ - first whitespace characters then wildcard characters.

3) openness :- An open distributed system is a system that offers services ~~distributed~~ according to standard rules that describe the syntax and semantics of there services - In computer networks, standard rules govern the format, contents and meaning of messages sent and received. Such rules are formalized in protocols. In distributed systems, services are specified through interfaces, which are described in an Interface Definition Language (IDL)

IDLs described an interface in a language - independent way, enabling communication between software components that do not share one language.

Interface definitions specify the names of the functions that are available, with types of the parameters, return values, possible exceptions that can be raised.

If properly specified, an interface definition allows a n arbitrary process that needs a certain interface to talk to another process that provides that interface. An IDL is used to set up communications between clients and servers in remote procedure calls (RPC). Another ~~imp~~ goal for an open distributed system is that it should be easy to configure the system out of different components.

should be easy to add new components or replace existing ones without affecting those components that stay in place. An open distributed system should also be extensible. To achieve flexibility in open distributed systems, it is crucial that the system is organized as a collection of relatively small and easily replaceable or adaptable components.

4) Scalability :- Scalability of a system can be measured along the following dimensions: A system can be scalable with respect to its size we can easily add more users and resources to the system. A geographically scalable system is the one in which the users and resources may lie far apart. A system can be administratively scalable. A system that is scalable in one or more of these dimensions often exhibits some loss of performance as the system scales up.

Scalability problems :- 1) Scaling w.r.t size. The server can become a bottleneck as the no of users and applications grow. Using only a single server is sometimes unavoidable eg: managing highly confidential info such as medical records, bank accounts. In such cases, copying the server to several locations to enhance performance maybe out of the question as it would make the services less secure.

many centralized services are implemented by means of only a single server running on a specific machine in the distributed system.

Ex: Even if one has virtually unlimited processing and storage capacity, communication with that server will eventually prohibit further growth.

Advantages of distributed systems over independent PC's

1) Accessibility.
2) More ~~reliability~~ reliability.
3) More interaction between users.
4) Availability
5) Easily extendable

* Hardware concepts:-

Sealer operation

$A_0 + B_0 = C_0$
$A_1 + B_1 = C_1$
$A_2 + B_2 = C_2$

Vector operation

$A_0 + B_0 = C_0$
$A_1 + B_1 = C_1$
$A_2 + B_2 = C_2$

* Design Issues
1) Transparency

| Kind | Meaning |
|---|---|
| location transparency | users cannot tell where resources are located |
| migration transparency | Resources can move at will without changing their names. |
| Replication transparency | The users cannot tell how many copies exist |
| concurrency transparency | multiple users can share resources automatically |
| parallelism transparency | Activities can happen in parallel without users knowing. |

* Flexibility:
* Reliability:
- Availability

- Reliability
  - Availability
  - Redundancy
  - Security
  - Fault tolerance

- Performance
  - Performance metrices
  - Response time
  - Throughput
  - System utilization
  - Amount of network capacity consumed
  - Fine grained parallelism.
  - Coarse grained parallelism.

- Scalability

* Challenges: The network is not reliable
  The network is not secure
  " not homogeneous.
  The topology changes
  Latency due to distance.
  Bandwidth is not infinite
  Transport cost
  There is no administrator

Page No.
Date

# File

- A named collection of related info. that is recorded on secondary storage
Files represent programs and data.
A file has a certain defined structure according

* Attributes
Name - only info. kept in human-readable form
Identifier - this unique tag identifies the file within the file system
Type - needed
Location -
Size -
Protection -
Time, Data and user identification -

* File operations -
create, write, read, reposition within file
delete, Truncate, ...

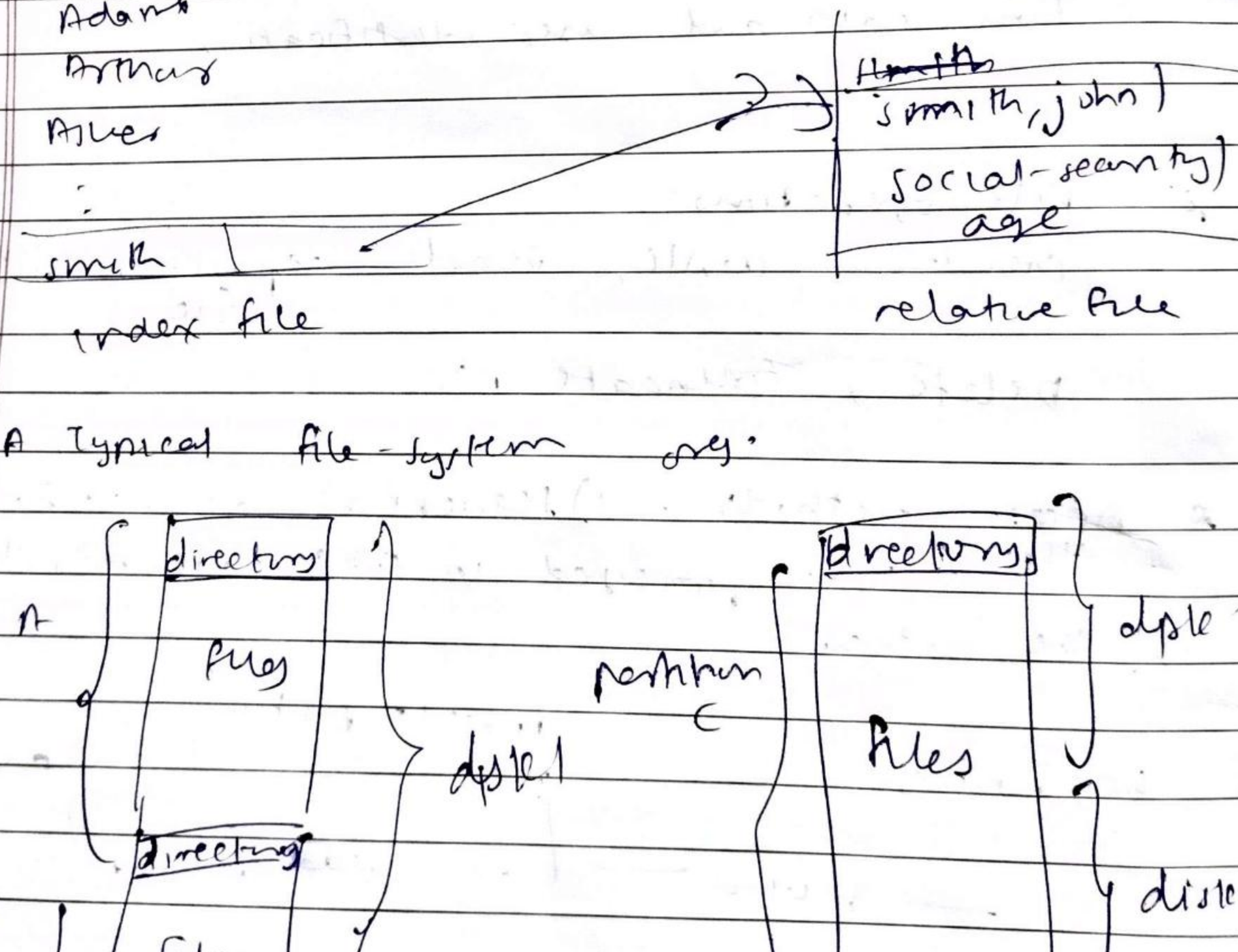
* Access methods - 1) Sequential access - Info. in the file is processed in order, one record after the other.

Page No.
Date

2) Direct access:- A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order. It is viewed as a numbered sequence of blocks or records. The direct access method allows random access to any file block. There is no order in restriction for reading or writing.

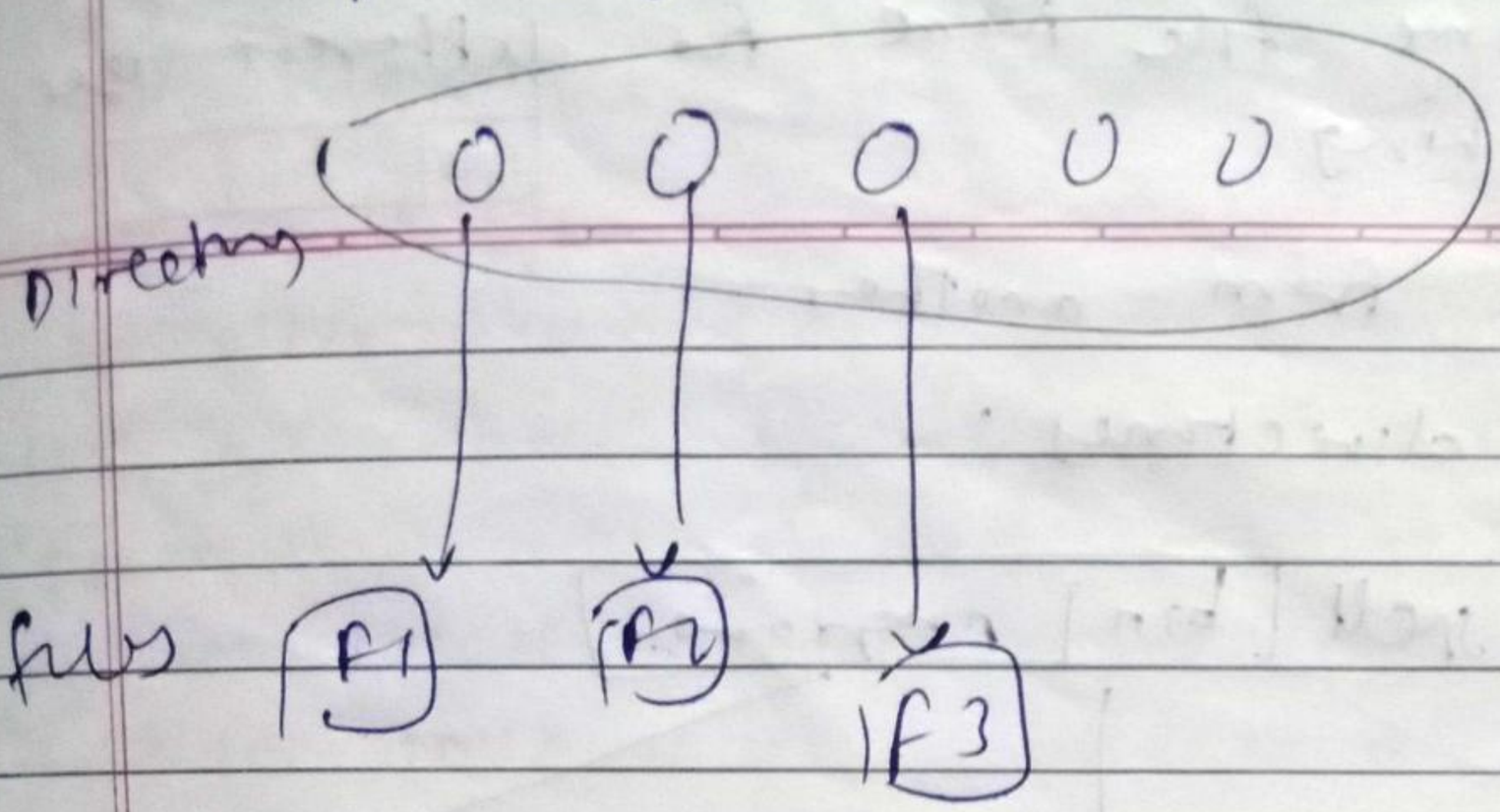
Eg. Index and relative files

It involves the construction of an index for the file. The index contains a pointer to access the file directory and find the desired record.




A Typical file-system org.

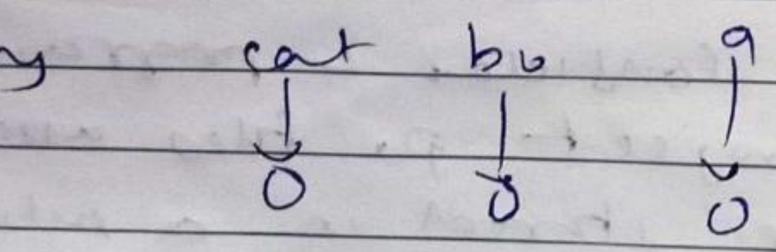
## Directory structure




A collection of nodes containing info about all files

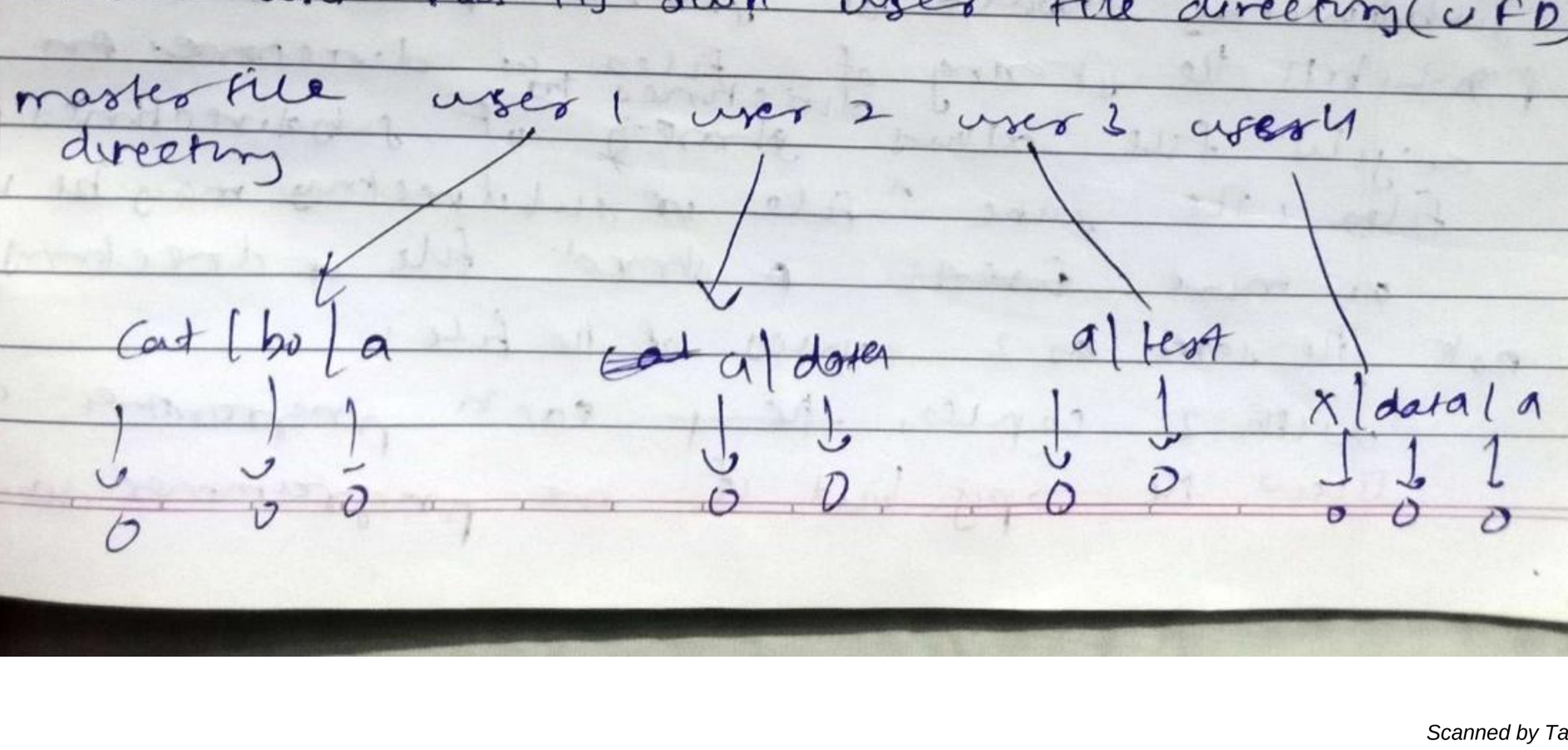
Ops performed on a directory:
1) Search for a file
2) Create a file
3) delete a file
4) list a directory
5) Rename a file
6) Traverse the file system

* Single level directory :- A single directory for all users



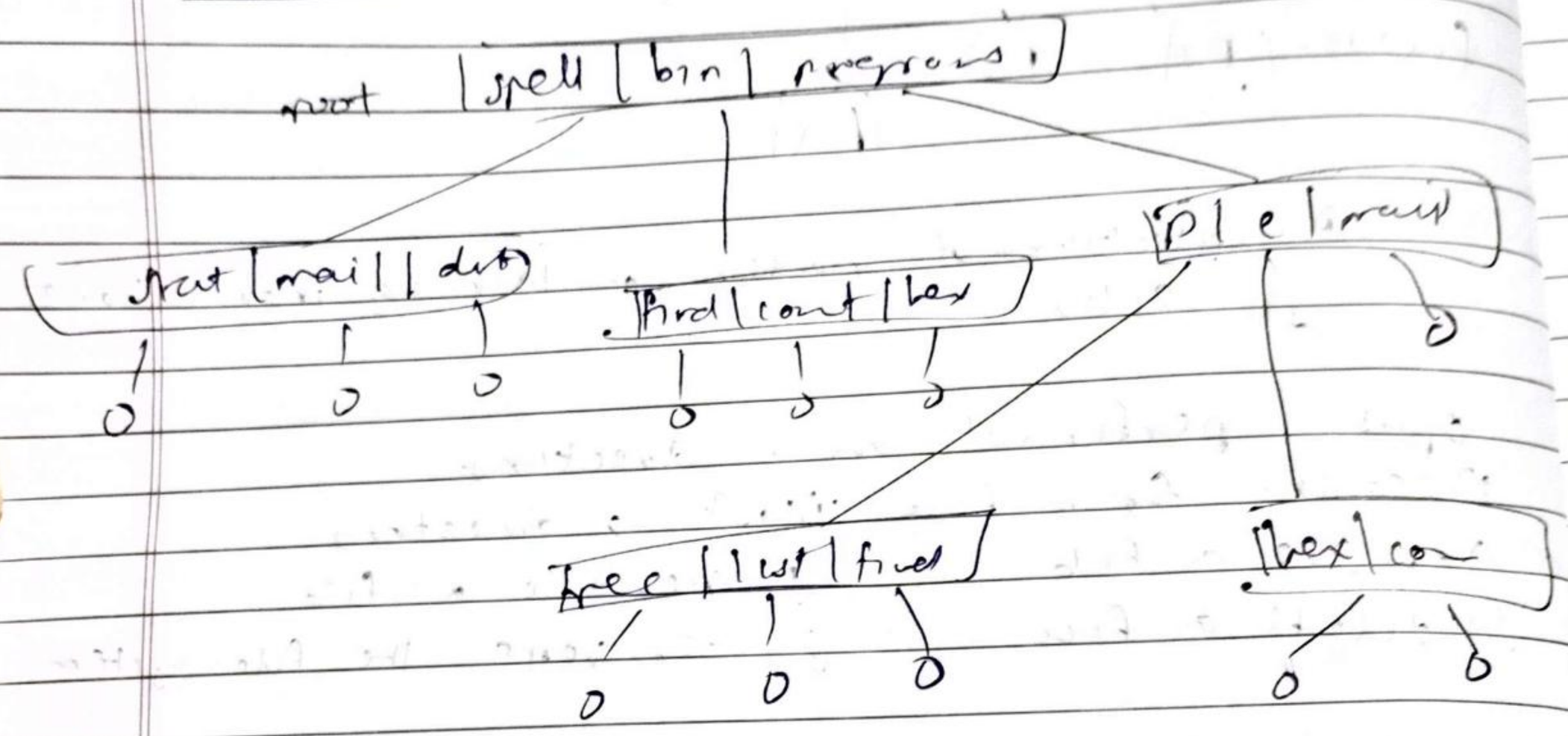

Naming problem
Grouping problem

* Two-level directory :- Separate directory for each user. Each user has its own user file directory (UFD)



Path name
- Can have the same file name for different user
- Efficient searching
- Isolates one user from another

* Tree - structured directories :




Relative path name, Absolute path name.

* Acyclic - graph directories : Consider 2 programmers who are working on a joint project the files associated with that project can be stored in a subdirectory, seperating them from other projects and files of the 2 programmers. But since both programmers are equally responsible for the project, both want the subdirectory to be in their own directories. The common subdirectory should be shared. A tree structure prohibits the sharing of files or directories. An acyclic file allows sharing of subdirectories and files. The same file or subdirectory may be in 2 or more ~~files~~. A shared file is directory is not the same as 2 copies of the file. With 2 copies, each programmer can view the copy but if one programmer changes

the file, the changes will not appear in the other's copy with a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other. Sharing is important for subdirectories; a new file created by one person will automatically appear in all the shared subdirectories. UNIX systems create a new directory entry called a link. A link in UNIX is a pointer to a file. Creating links is a kind of shortcuts to access a file. Links allow more than 1 path name (absolute or relative) to refer to the same file, elsewhere. when a reference to a file is made, we search the directory. If the directory entry is marked as a link, then the name of the real file is included in the link information. we resolve the link by using that path name to locate the real file. Tree structure and soft and hard links - Shirly An acyclic-graph directory structure is more flexible than a simple tree structure, but it is also more complex.