ICE-2261
(Data Structure)

Lecture on
Chapter-4: LINKED LISTS

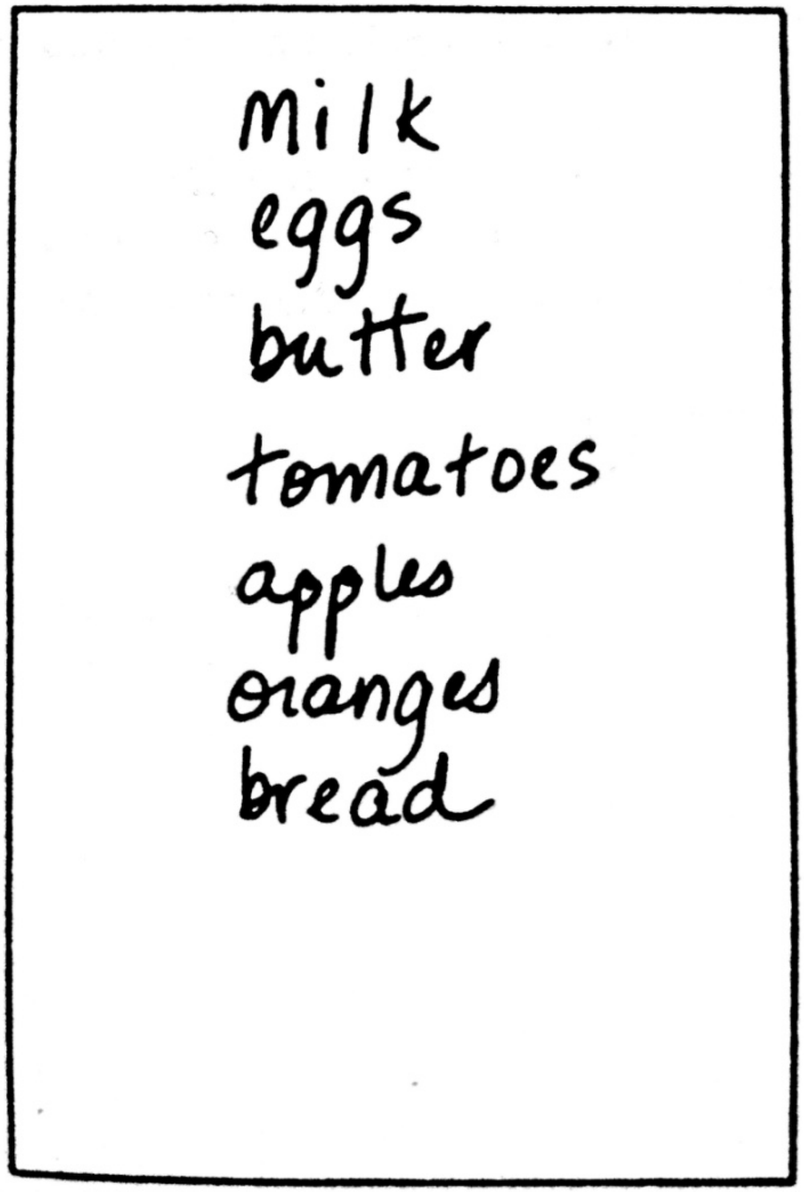
By

Dr. M. Golam Rashed

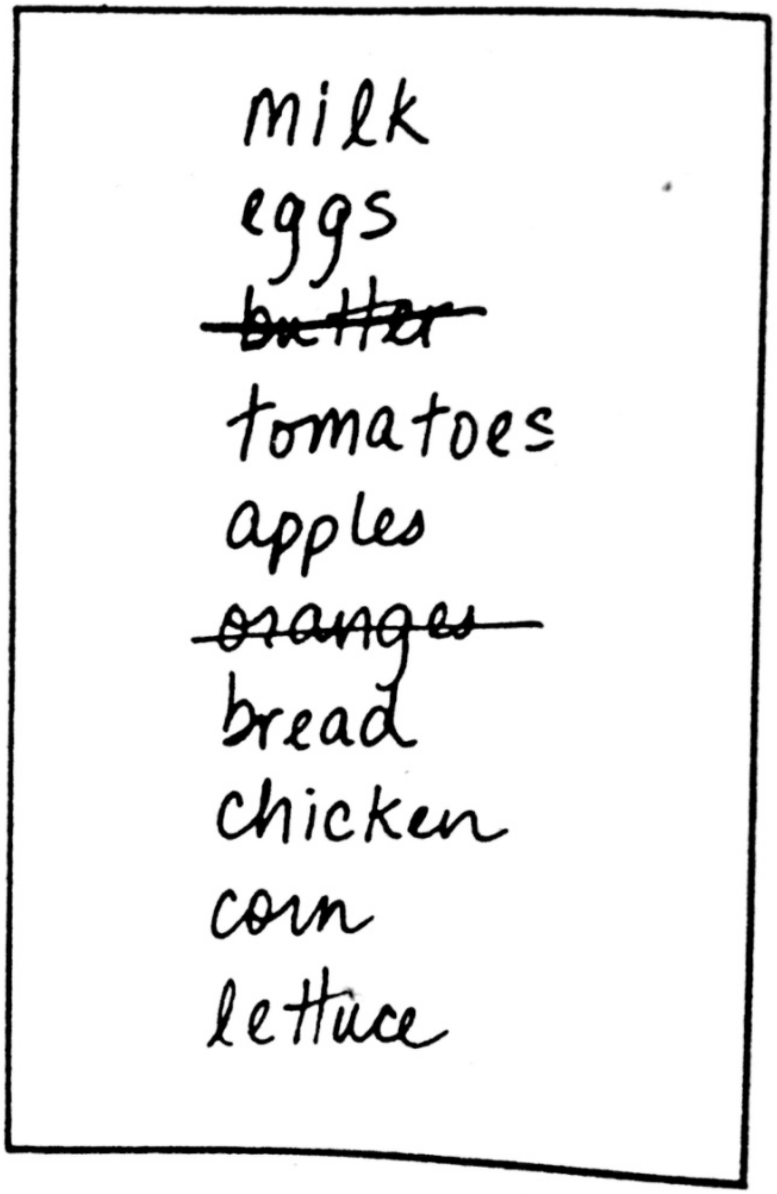
(golamrashed@ru.ac.bd)

Department of Information and Communication Engineering (ICE)
University of Rajshahi, Rajshahi-6205, Bangladesh

LINKED LISTS: Introduction



milk
eggs
butter
tomatoes
apples
oranges
bread



milk
eggs
~~butter~~
tomatoes
apples
~~oranges~~
bread
chicken
corn
lettuce

LINKED LISTS: Introduction

In linear array……,

- ✓ The linear relationship between the data elements of an array is reflected by the physical relationship of the data in memory, not by any information contained in the data elements themselves.
- ✓ It is easy to compute the address of an element in an **Array**.

Arrays have certain disadvantages:

- ➢ It is relatively expensive to insert and delete elements in an array,
- ➢ One can not simply double or triple the size of an array when additional space is required.

For this reason,

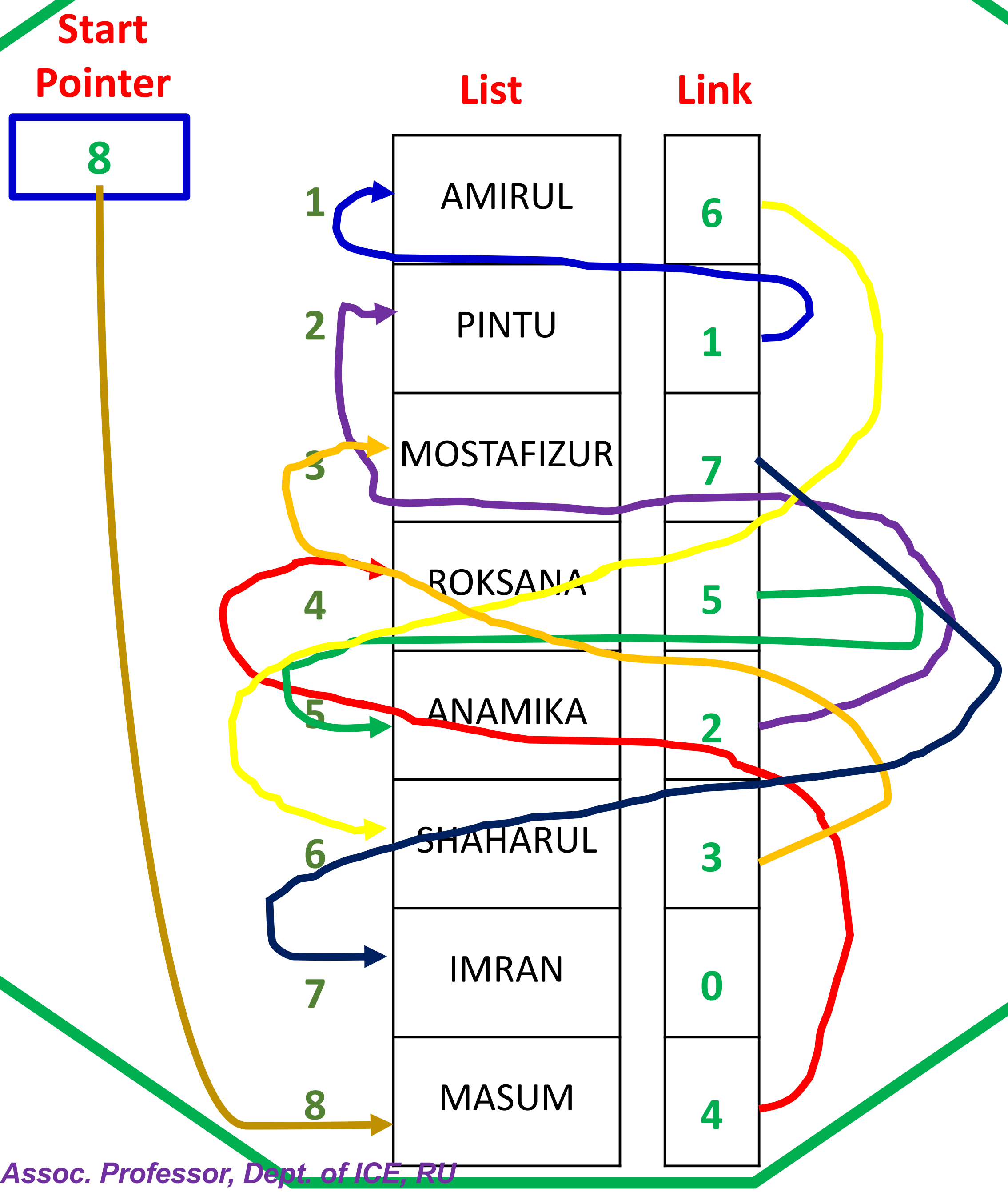
Arrays are

called *Dense lists*, and

said to be *Static* data structure.

LINKED LISTS:

ICE 2261



Start Pointer

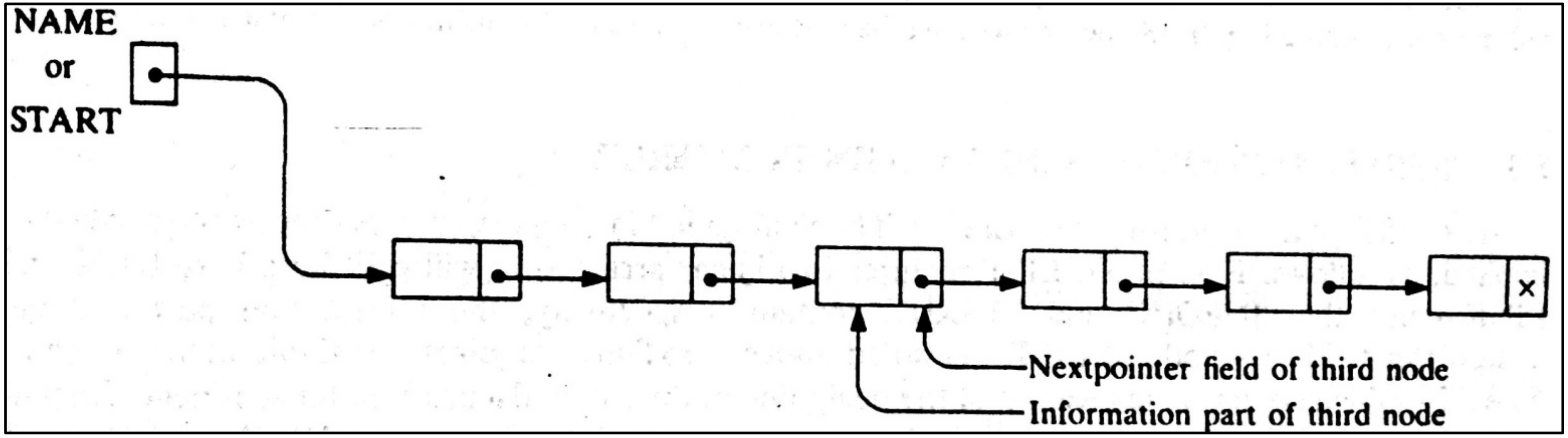
8

	List	Link
1	AMIRUL	6
2	PINTU	1
3	MOSTAFIZUR	7
4	ROKSANA	5
5	ANAMIKA	2
6	SHAHARUL	3
7	IMRAN	0
8	MASUM	4

LINKED LIST

LINKED LISTS: Definition

A *linked list*, or *one-way list*, is a linear collection of data elements, called *nodes*, where the linear order is given by means of *pointers.*


NAME
or
START
×
Nextpointer field of third node
Information part of third node

✓ Each node is divided into two parts:

➢ The first part contains the information of the element,

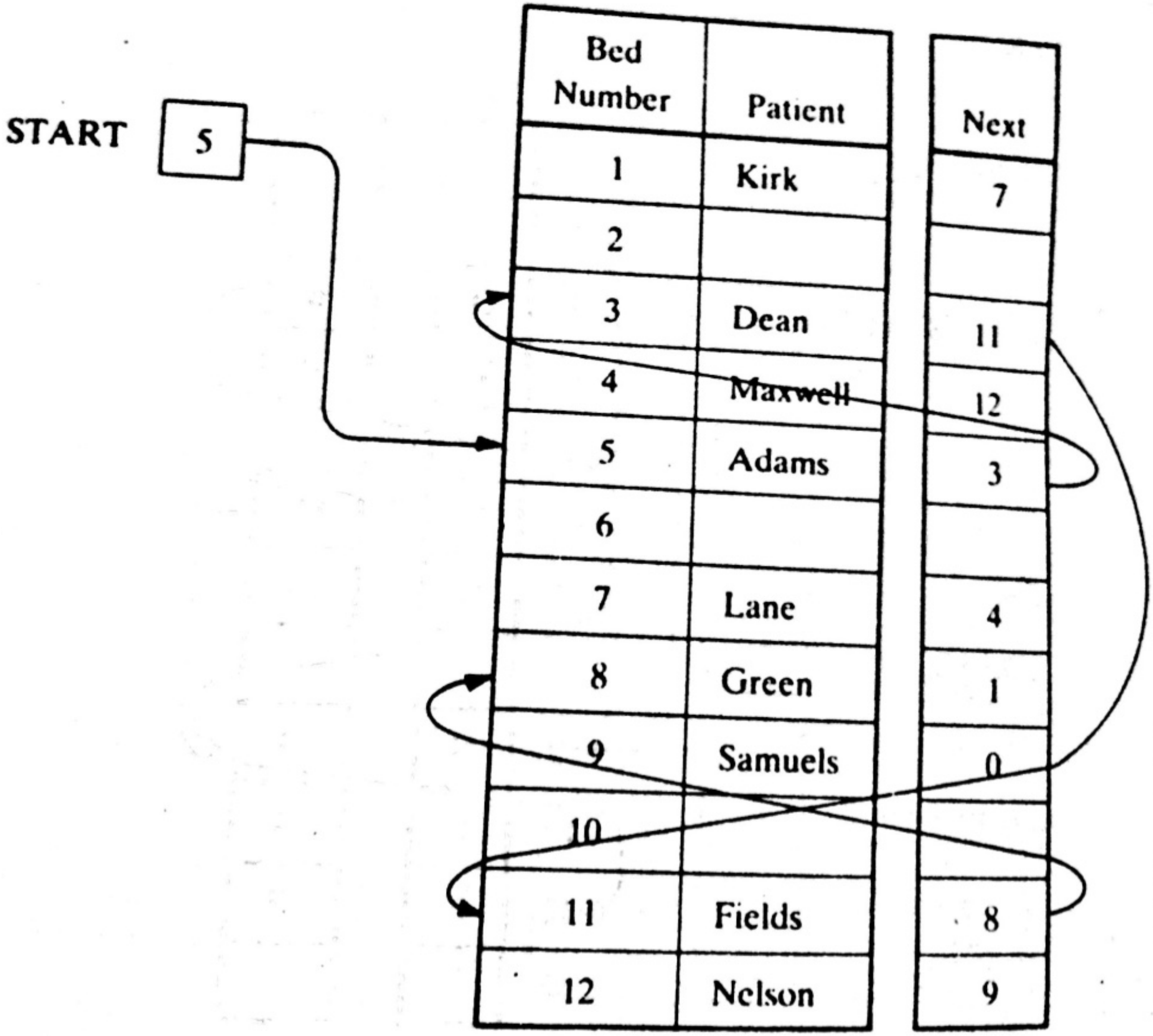
➢ The second part, called the *link field* or *nextpointer field,* contains the address of the next node in the list.

✓ There is an arrow drawn from a node to the next node in the list.

✓ The pointer of the last node contains a special value, called the *null* pointer, which is any invalid address (0 or a negative number), denoted by × in the diagram.

✓ The linked list also contain a *list pointer variable*-called START or NAME- which contain the address of the first node in the list.

LINKED LISTS: Example



START 5

Bed Number	Patient	Next
1	Kirk	7
2		
3	Dean	11
4	Maxwell	12
5	Adams	3
6		
7	Lane	4
8	Green	1
9	Samuels	0
10		
11	Fields	8
12	Nelson	9

LINKED LISTS: Representation in Memory

Let LIST be a linked list which will be maintained in the memory

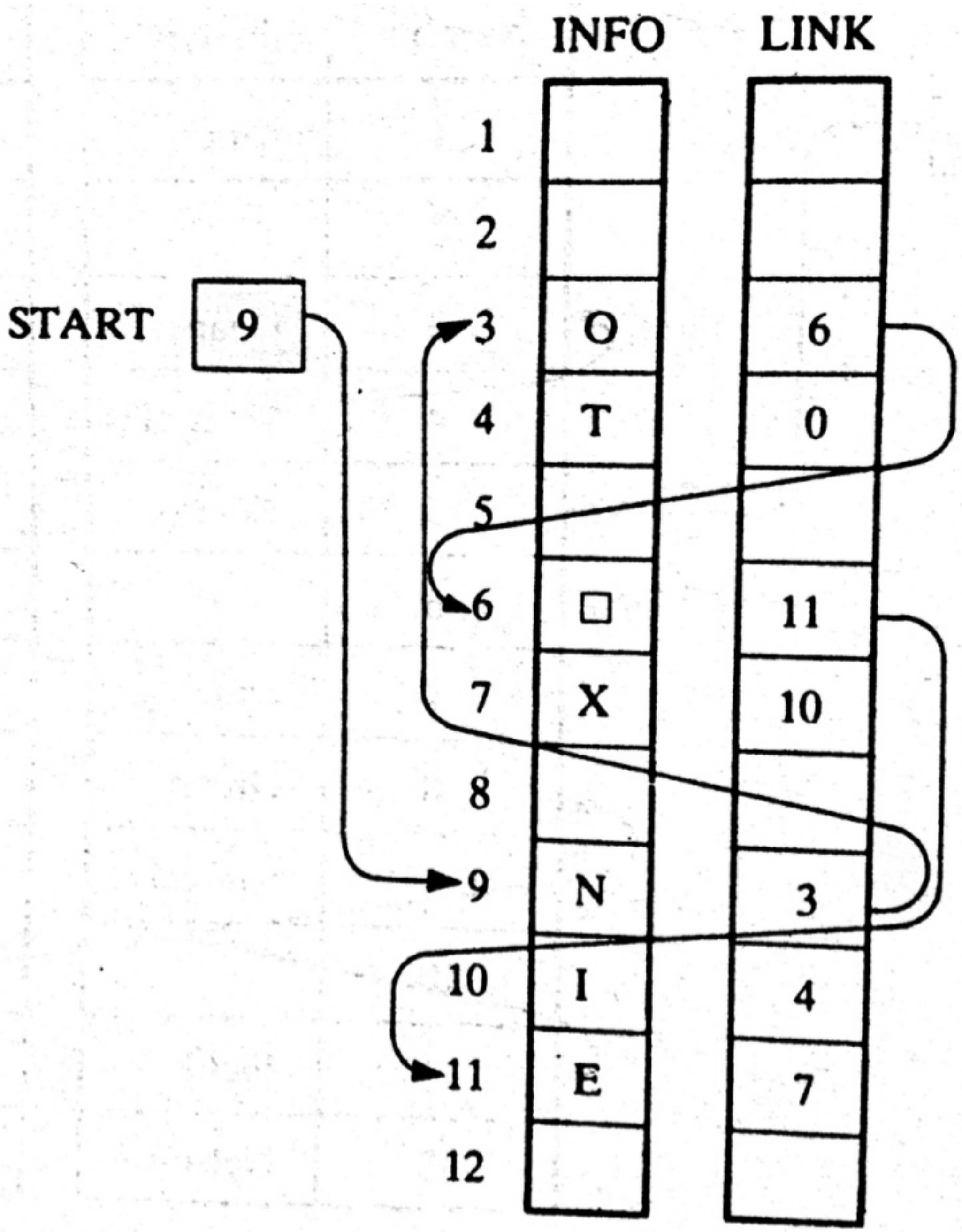
LIST requires…..

- ➢ Two linear array,
  - ❑ INFO[K]-Information part, and
  - ❑ LINK[K]-nextpointer field of a node of LIST
- ➢ A variable name-START, indicating the beginning of the LIST.
- ➢ A nextpointer senitel-NULL, indicates the end of the LIST

❖ Since, the subscripts of the array INFO and LINK will usually be positive, NULL=0, unless otherwise stated

LINKED LISTS: Example

ICE 2261



START=9, so INFO[9]= ? N

LINK[9]=3, so, INFO[3]=? O

LINK[3]=6, so, INFO[6]=? □

LINK[6]=11, so, INFO[11]=? E

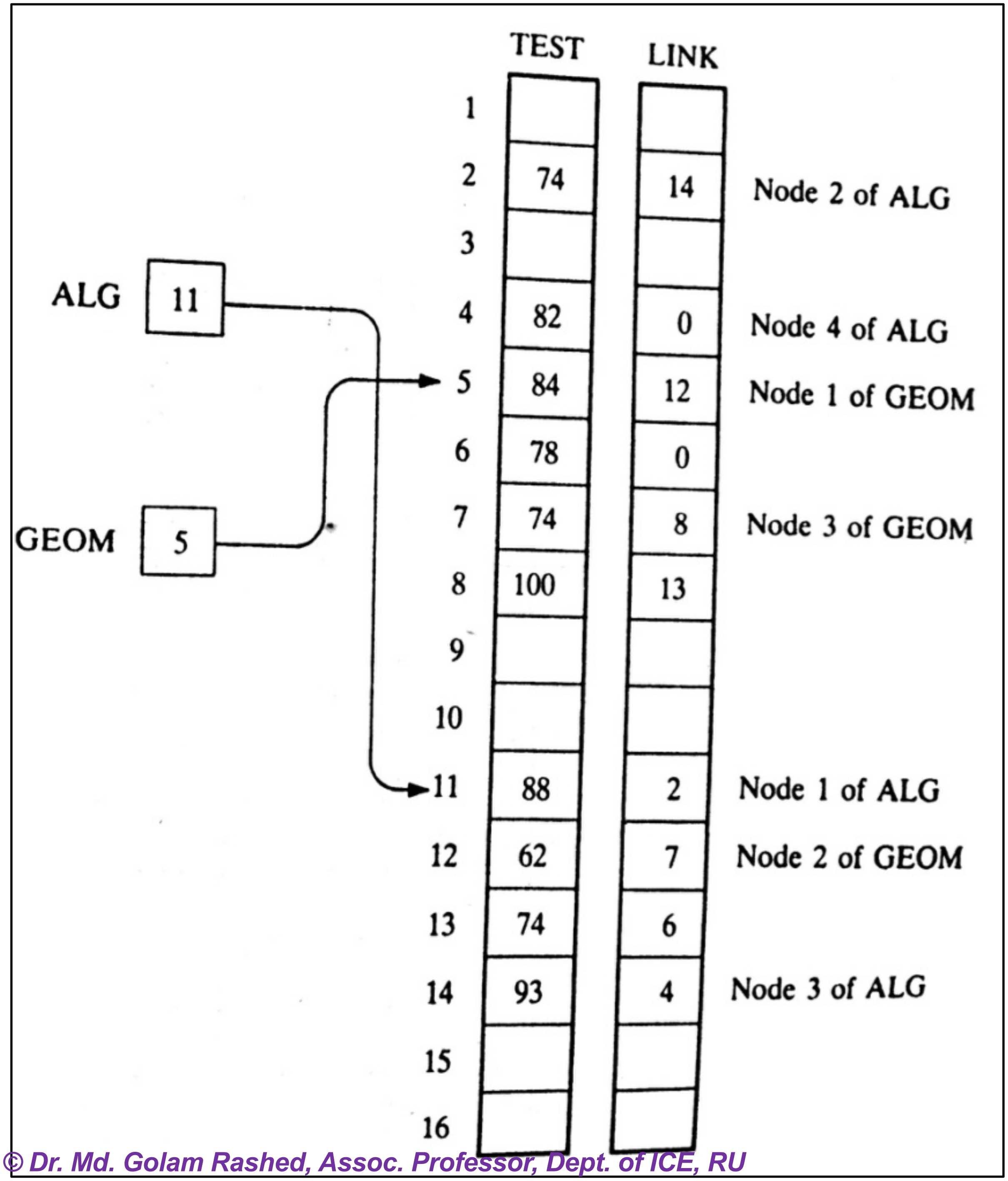
LINK[11]=7, so, INFO[7]= ? X

LINK[7]=10, so, INFO[10]= ? I

LINK[10]=4, so, INFO[4]= ? T

LINK[4]=0, so, INFO[0]= ? NULL

LINKED LISTS: Example



	TEST	LINK	
1			
2	74	14	Node 2 of ALG
3			
4	82	0	Node 4 of ALG
5	84	12	Node 1 of GEOM
6	78	0	
7	74	8	Node 3 of GEOM
8	100	13	
9			
10			
11	88	2	Node 1 of ALG
12	62	7	Node 2 of GEOM
13	74	6	
14	93	4	Node 3 of ALG
15			
16			

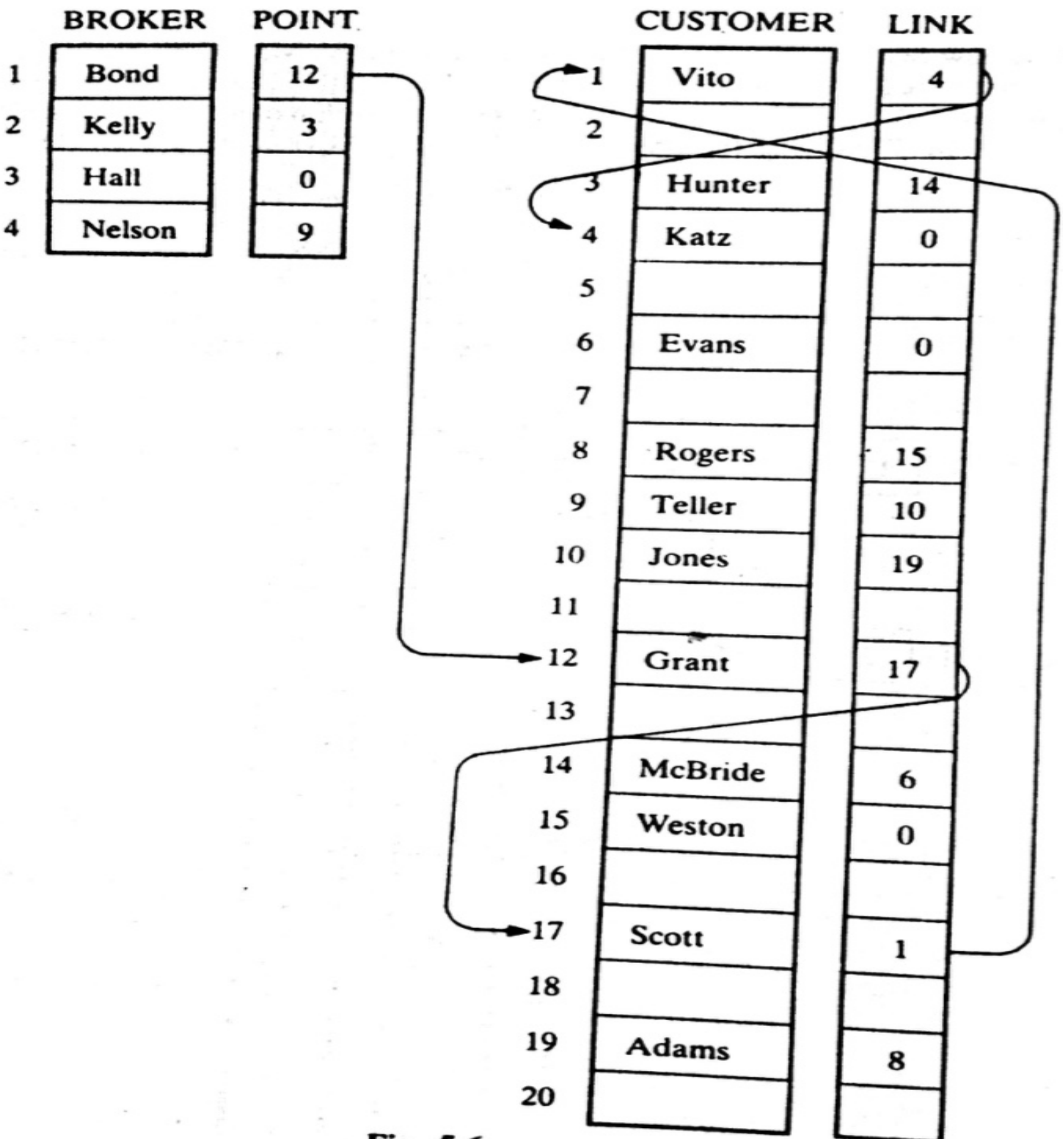
ALG consists of Test score:

88, 74, 93, 82

GEOM consists of Test score:

84, 62, 74, 100,74, 78

LINKED LISTS: Example



	BROKER	POINT
1	Bond	12
2	Kelly	3
3	Hall	0
4	Nelson	9

	CUSTOMER	LINK
1	Vito	4
2		
3	Hunter	14
4	Katz	0
5		
6	Evans	0
7		
8	Rogers	15
9	Teller	10
10	Jones	19
11		
12	Grant	17
13		
14	McBride	6
15	Weston	0
16		
17	Scott	1
18		
19	Adams	8
20		

Fig. 5-6

Bond's list of customers:
Grant, Scott, Vito, Katz

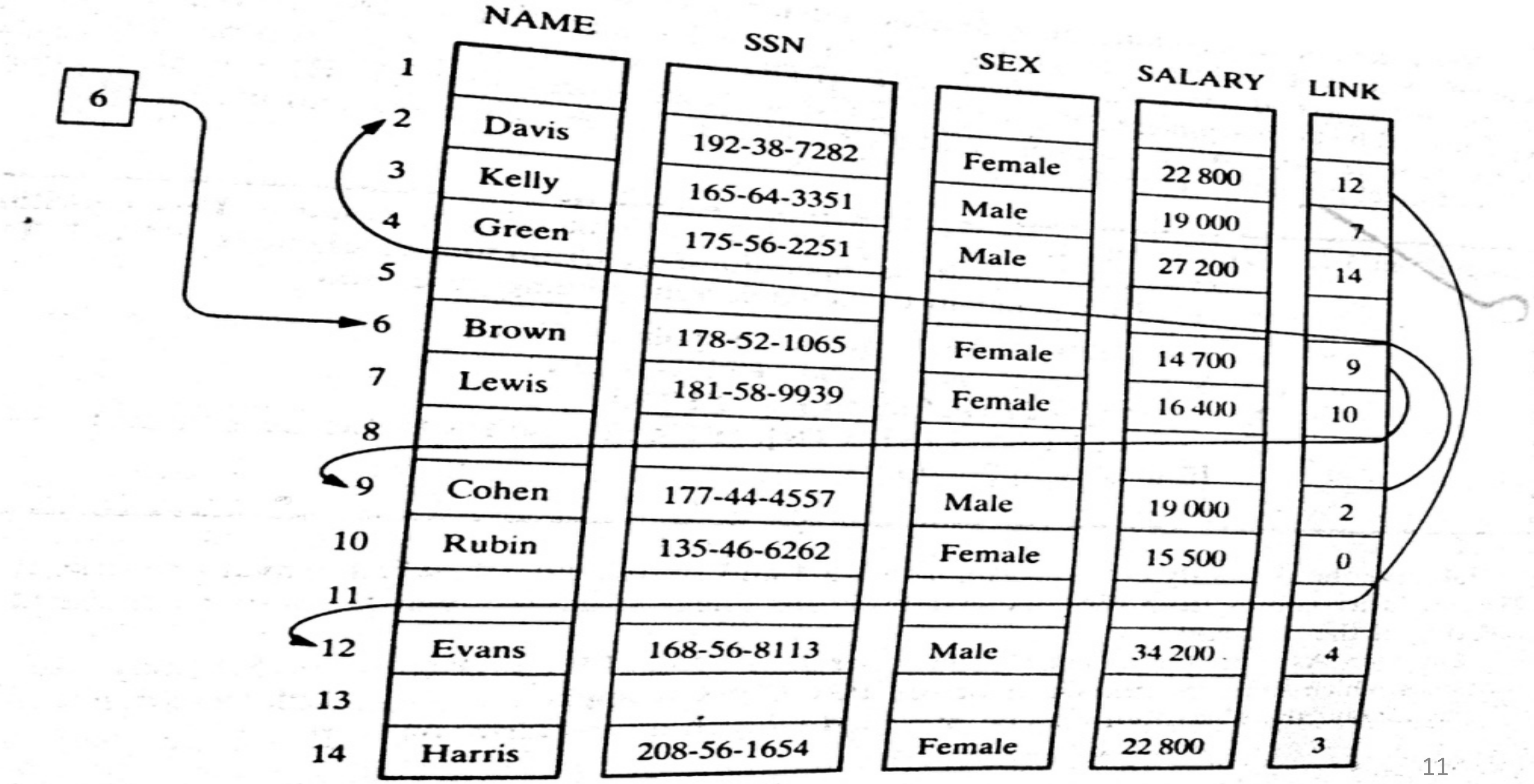
Kelly's list of customers:
Hunter, McBridge, Evans

Hall's list of customers:
NULL / Empty List

Nelson's list of customers:
Teller, Jones, Adams, Rogers, Weston

LINKED LISTS: Example

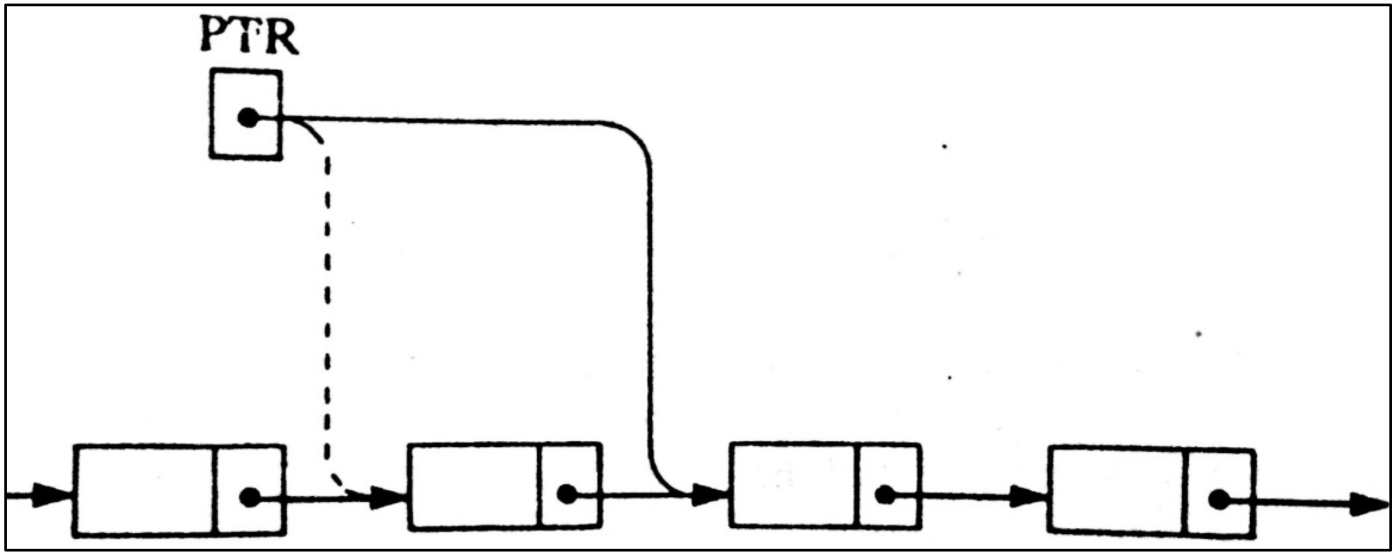
- In general, the information part of a node may be a record with more than one data item.
- In such a case, the data must be stored in some type of record structure or in a collection of parallel arrays.



	NAME	SSN	SEX	SALARY	LINK
1					
2	Davis	192-38-7282	Female	22 800	12
3	Kelly	165-64-3351	Male	19 000	7
4	Green	175-56-2251	Male	27 200	14
5					
6	Brown	178-52-1065	Female	14 700	9
7	Lewis	181-58-9939	Female	16 400	10
8					
9	Cohen	177-44-4557	Male	19 000	2
10	Rubin	135-46-6262	Female	15 500	0
11					
12	Evans	168-56-8113	Male	34 200	4
13					
14	Harris	208-56-1654	Female	22 800	3

LINKED LISTS: Traversing

Let **LIST** be a linked list in memory stored in linear arrays **INFO** and LINK with **START** pointing to the first element and **NULL** indicating the end of **LIST**.



- ✓ Traversing algorithm uses a pointer variable **PTR** which points to the node that is currently being processed.
- ✓ Accordingly, **LINK[PTR]** points to the next node to be processed. Thus,

  PTR:=LINK[PTR], moves the pointer to the next node in the list.

LINKED LISTS: Traversing Algorithm

Let LIST be a linked list in memory.

```
LINKED_LIST_Traversing(INFO, LINK, START)
Step1.  Set PTR:=START.[Initialize pointer PTR.]
Step2.  Repeat Steps 3 and 4 while $PTR \neq NULL$
Step3.          Apply PROCESS to INFO[PTR]
Step4.          Set PTR:=LINK[PTR]. [PTR now point to the next node.]
        [End of step 2 loop]
Step5.  Exit.
```

Any difference with Linear array traversing ??

```
LINEAR_ARRAY_Traversing
Step 1. [Initialize counter] Set K:=LB
Step 2. Repeat Step 3 and 4 while K<=UB [Repeat while loop]
Step 3.         [Visit element] Apply PROCESS to A[K].
Step 4.                 [Increase counter] Set K:=K+1.
        [End of Step 2 loop.]
Step 5. Exit.
```

LINKED LISTS: Practice…

Teach yourself: Example 5.7 (Find the number of elements in a linked list)

LINKED LISTS: Searching

Given:

- LIST- a linked list in memory (Unknown/Unseen)
- ITEM- a specific information.

Objective: Finding the location LOC of the node where ITEM first appears in LIST.

Here, TWO searching algorithm will be discussed for finding the location of LOC of the node where ITEM first appears in LIST.

- Algorithm-1 for LIST is Unsorted
- Algorithm-2 for LIST is Sorted.

LINKED LISTS: Searching Algorithm-1

SEARCH (INFO, LINK, START, ITEM, LOC) [when list is unsorted]

```
Step1.  Set PTR:=START
Step2.  Repeat Step 3 while PTR $\neq$ NULL
Step3.          If ITEM=INFO [PTR], then:
                    Set LOC:=PTR, and Exit.
                Else:
                    Set LOC:=LINK[PTR]. [PTR now points to the next node.]
                [End of If structure.]
        [End of Step 2 loop.]
Step4.  [Search is unsuccessful.] Set LOC:=NULL.
Step5.  Exit.
```

LINKED LISTS: Searching Algorithm-2

SEARCH (INFO, LINK, START, ITEM, LOC) [when list is sorted]

```
Step1.  Set PTR:=START
Step2.  Repeat Step 3 while PTR $\neq$ NULL
Step3.      If ITEM < INFO [PTR], then:
                Set PTR:=LINK[PTR] [PTR now points to next node.]
            Else if ITEM=INFO[PTR], then:
                Set: LOC=PTR. and Exit [Search is successful]
            Else:
                Set LOC:= NULL, and Exit. [ITEM now exceeds INFO[PTR]]
            [End of If structure.]
        [End of Step 2 loop.]
Step4.  Set LOC:=NULL.
Step5.  Exit.
```

Descending Order Sorted LIST

Any change in algorithm for Ascending Order Sorted LIST?

LINKED LISTS: Searching Algorithm

Limitations:

NB. A binary search algorithm cannot be applied to a sorted linked list. **Since, there is no way of indexing the middle element in the list.**

The main drawbacks in using a linked list as data structure

LINKED LISTS: Memory Allocation

- ✓ The maintenance of linked lists in memory assumes the possibility of ……
  - ❖ Inserting new nodes into the lists, and
  - ❖ Deleting nodes from the list.
- ✓ Hence, requires some mechanism which provides:
  - ➢ Unused memory space for the new nodes.
  - ➢ Deleted nodes becomes available for future use.

To meet the necessities……..

LINKED LISTS: Memory Allocation

✓Together with the linked lists in memory, a special list is maintained which consists of unused memory cells.

✓This list, which has its own pointer, is called …

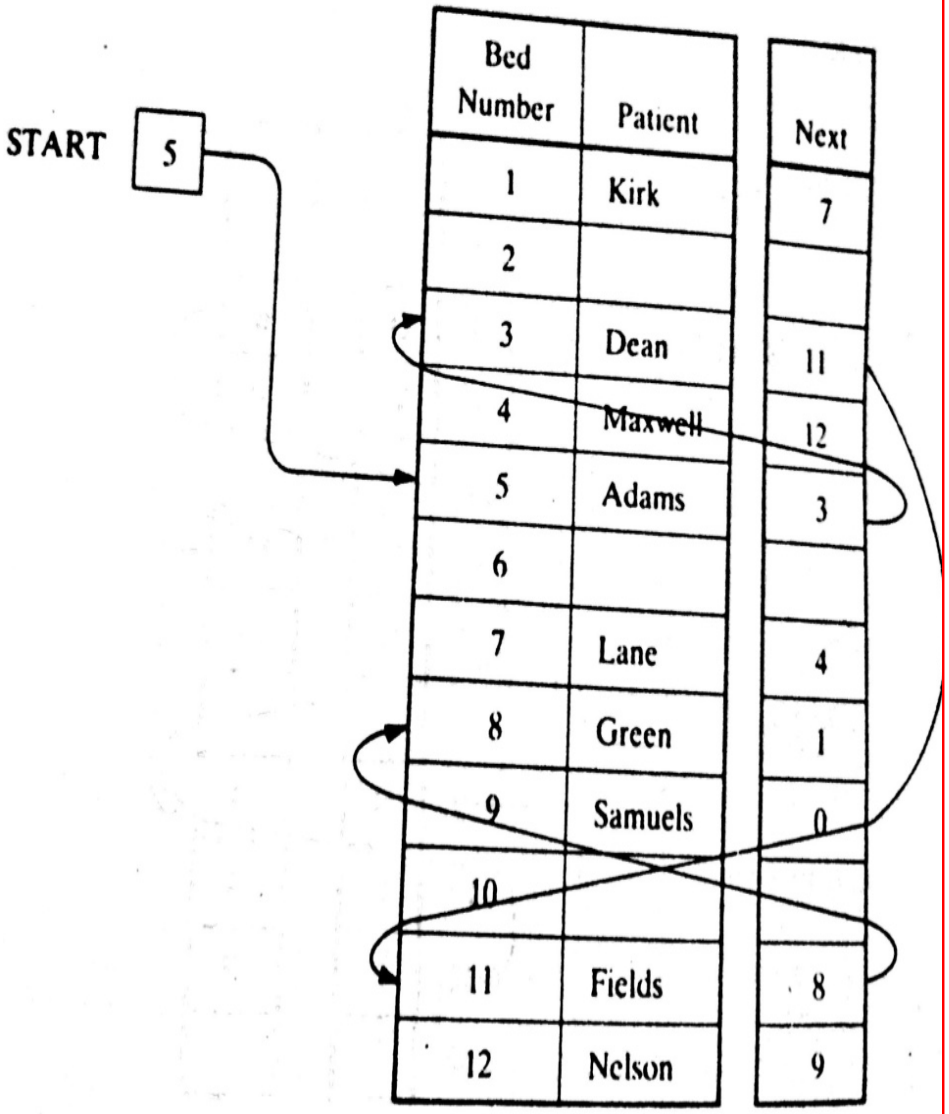
- ✓***The list of available space*** or
- ✓*the free-storage list* or
- ✓*the free pool.*

LINKED LISTS: Memory Allocation

- ✓ Here, linked lists are implemented by parallel arrays.
- ✓ Suppose **INSERTIONS** and **DELETIONS** are to be performed on our linked lists.
- ✓ Then the unused memory cells in the arrays will also be linked together to form a linked list using **AVAIL** as its list pointer variable.

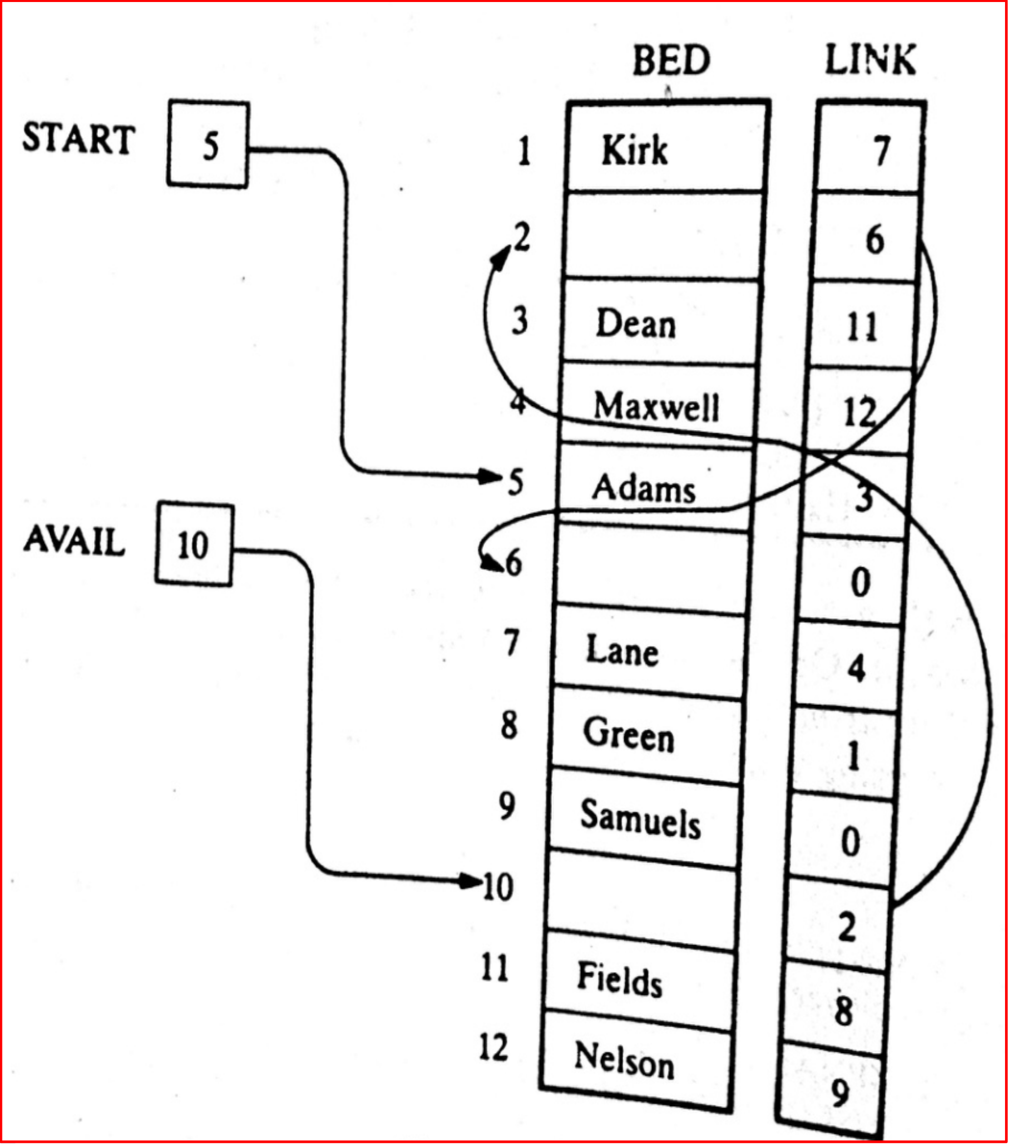
LIST(INFO, LINK, START, AVAIL)

LINKED LISTS: Example 5.10



START 5

Bed Number	Patient	Next
1	Kirk	7
2		
3	Dean	11
4	Maxwell	12
5	Adams	3
6		
7	Lane	4
8	Green	1
9	Samuels	0
10		
11	Fields	8
12	Nelson	9

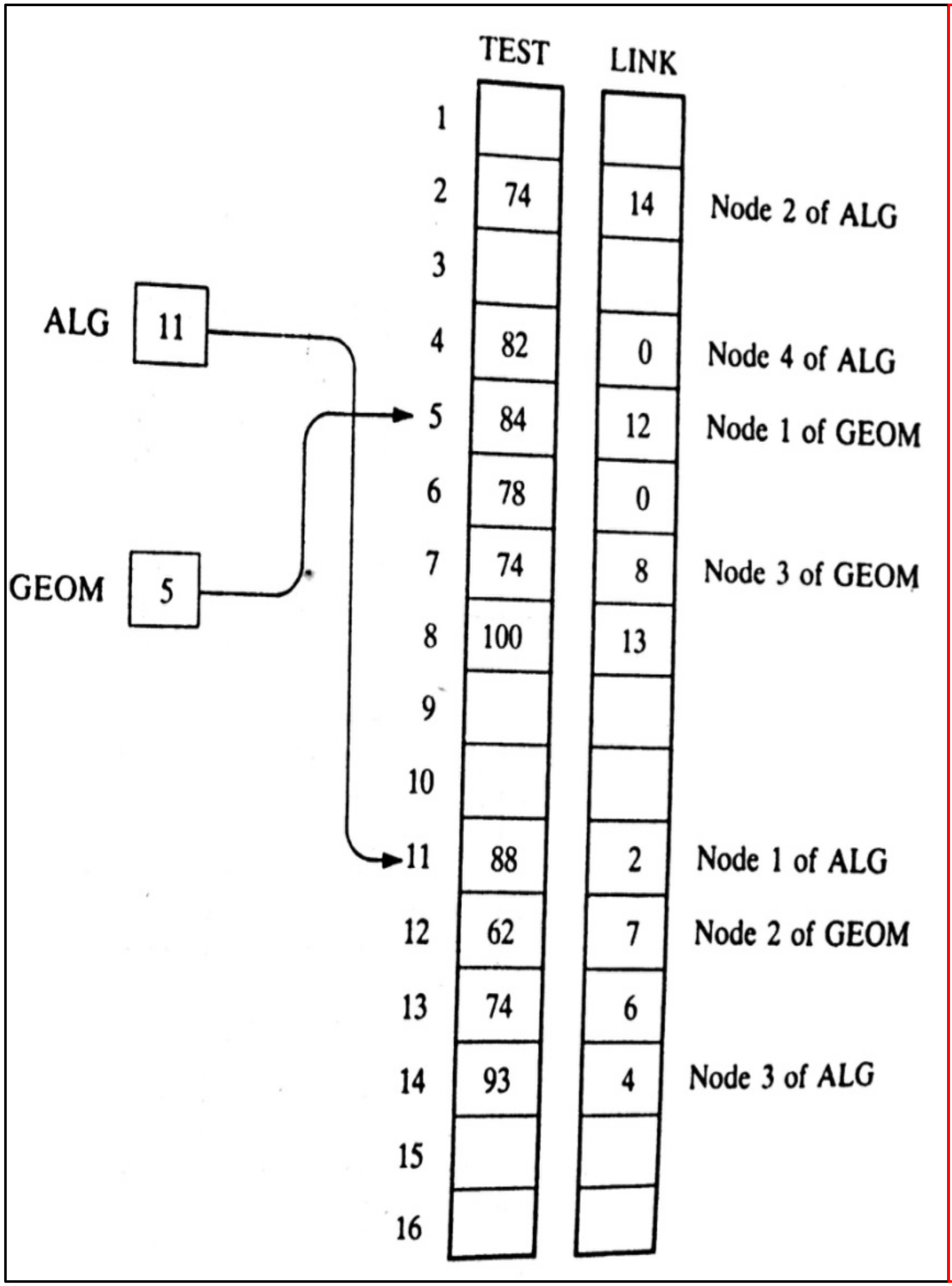


START 5

AVAIL 10

	BED	LINK
1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8	Green	1
9	Samuels	0
10		2
11	Fields	8
12	Nelson	9

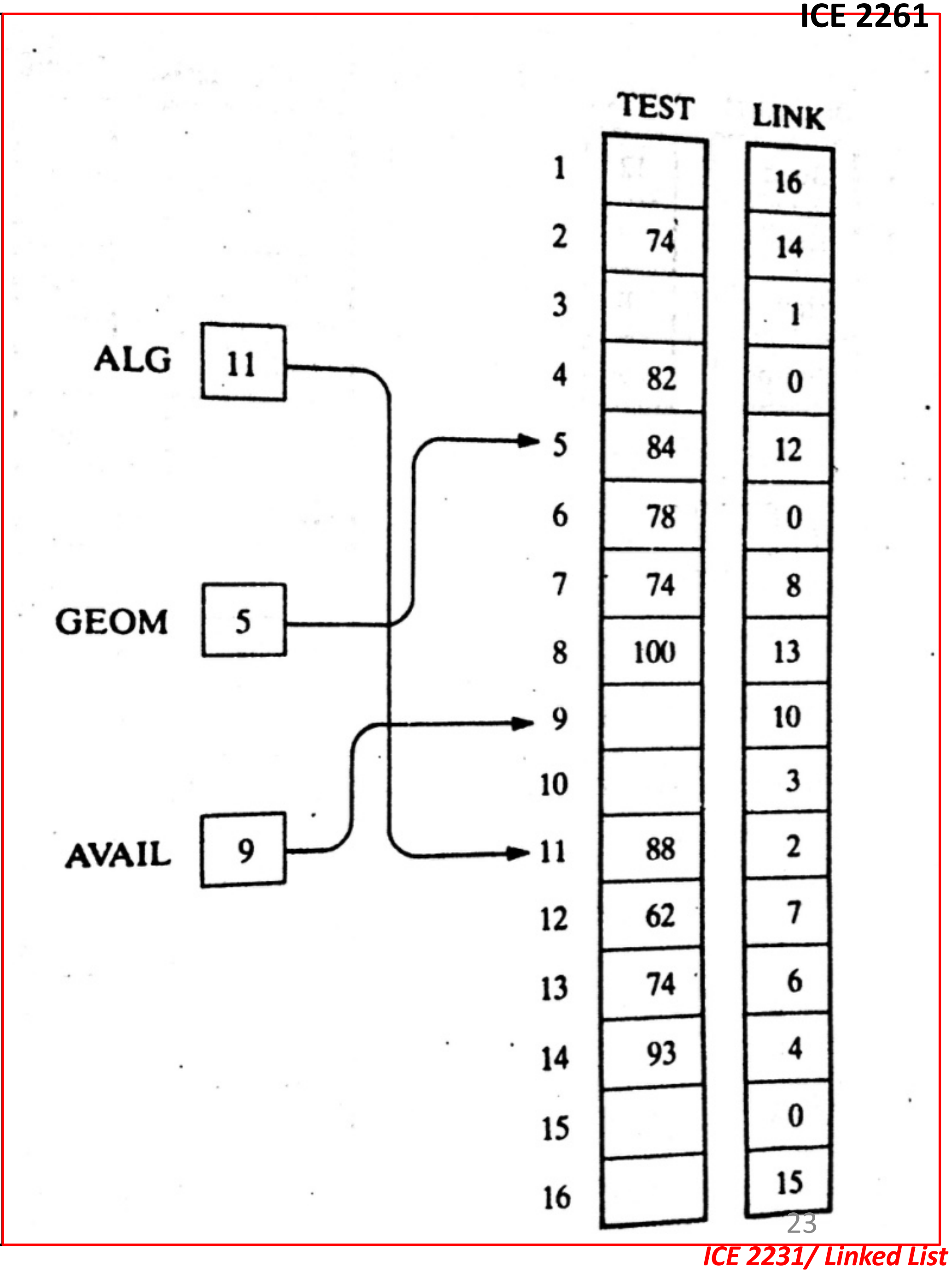
LINKED LISTS: Example 5.11(a)



ALG 11

GEOM 5

	TEST	LINK	
1			
2	74	14	Node 2 of ALG
3			
4	82	0	Node 4 of ALG
5	84	12	Node 1 of GEOM
6	78	0	
7	74	8	Node 3 of GEOM
8	100	13	
9			
10			
11	88	2	Node 1 of ALG
12	62	7	Node 2 of GEOM
13	74	6	
14	93	4	Node 3 of ALG
15			
16			



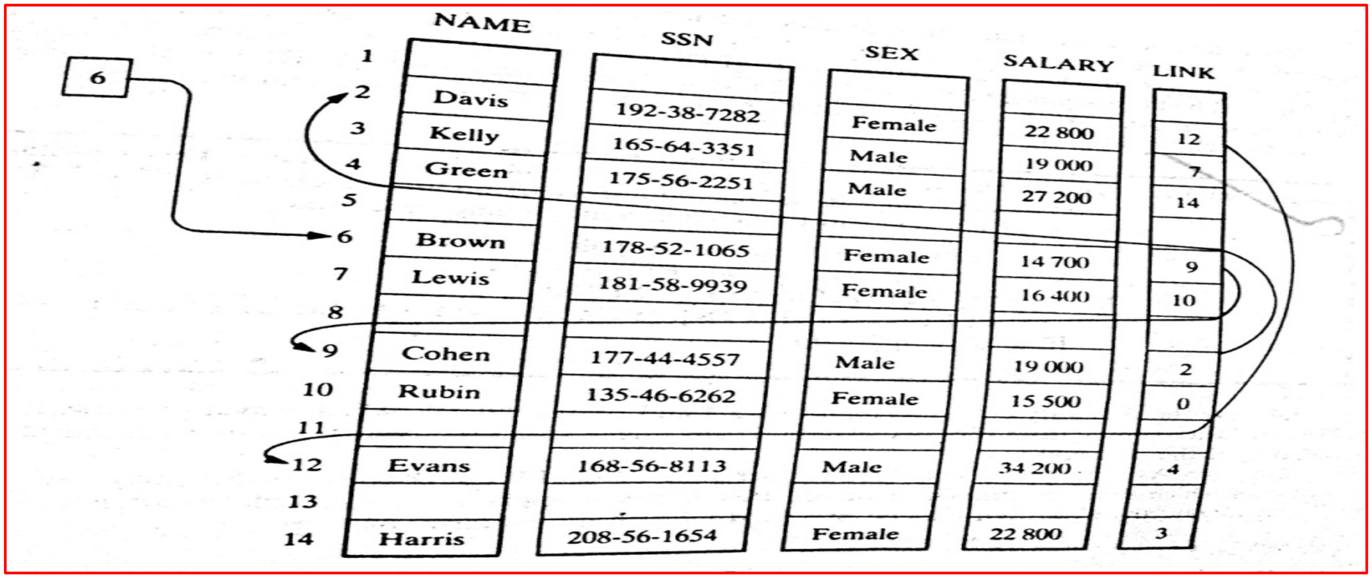
ALG 11

GEOM 5

AVAIL 9

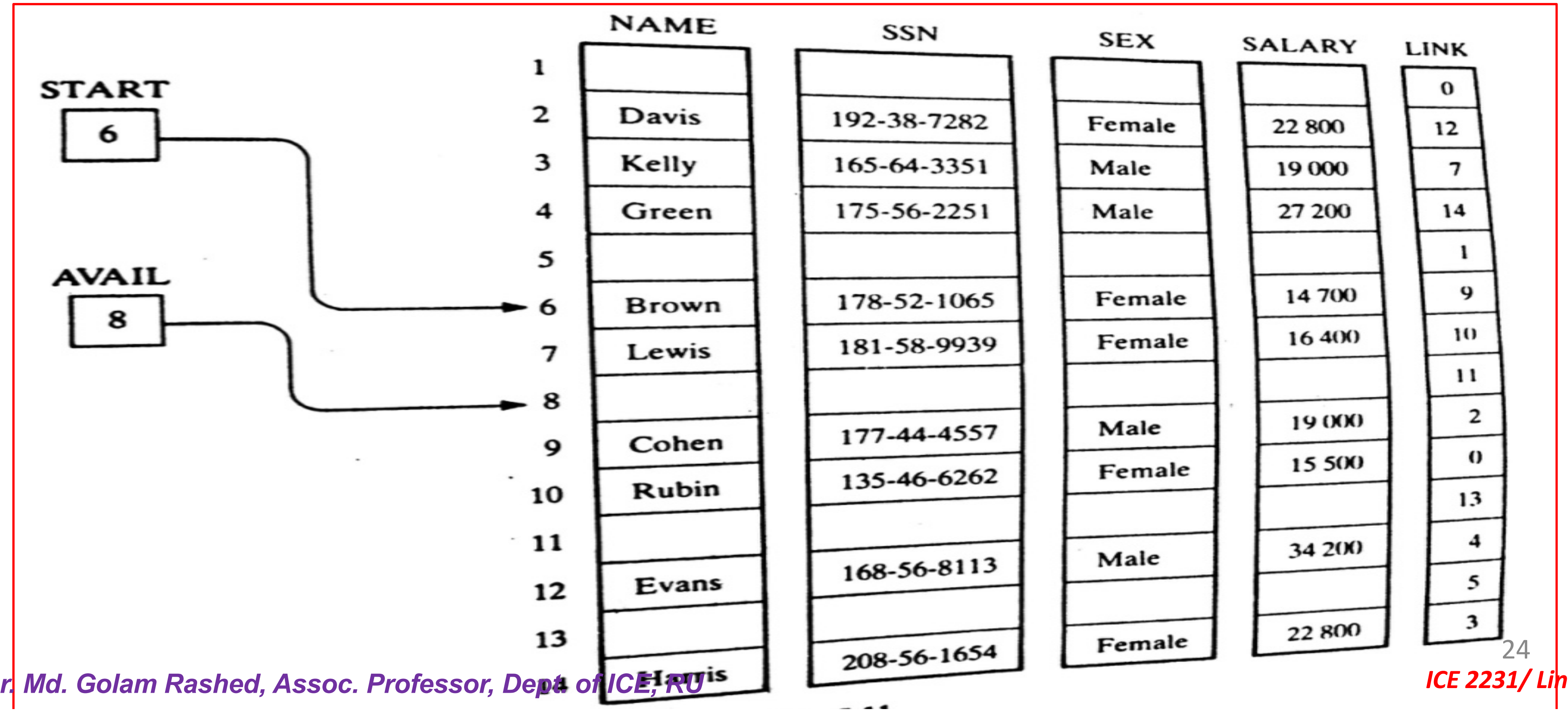
	TEST	LINK
1		16
2	74	14
3		1
4	82	0
5	84	12
6	78	0
7	74	8
8	100	13
9		10
10		3
11	88	2
12	62	7
13	74	6
14	93	4
15		0
16		15

LINKED LISTS: Example 5.11(b)



6

	NAME	SSN	SEX	SALARY	LINK
1					
2	Davis	192-38-7282	Female	22 800	12
3	Kelly	165-64-3351	Male	19 000	7
4	Green	175-56-2251	Male	27 200	14
5					
6	Brown	178-52-1065	Female	14 700	9
7	Lewis	181-58-9939	Female	16 400	10
8					
9	Cohen	177-44-4557	Male	19 000	2
10	Rubin	135-46-6262	Female	15 500	0
11					
12	Evans	168-56-8113	Male	34 200	4
13					
14	Harris	208-56-1654	Female	22 800	3

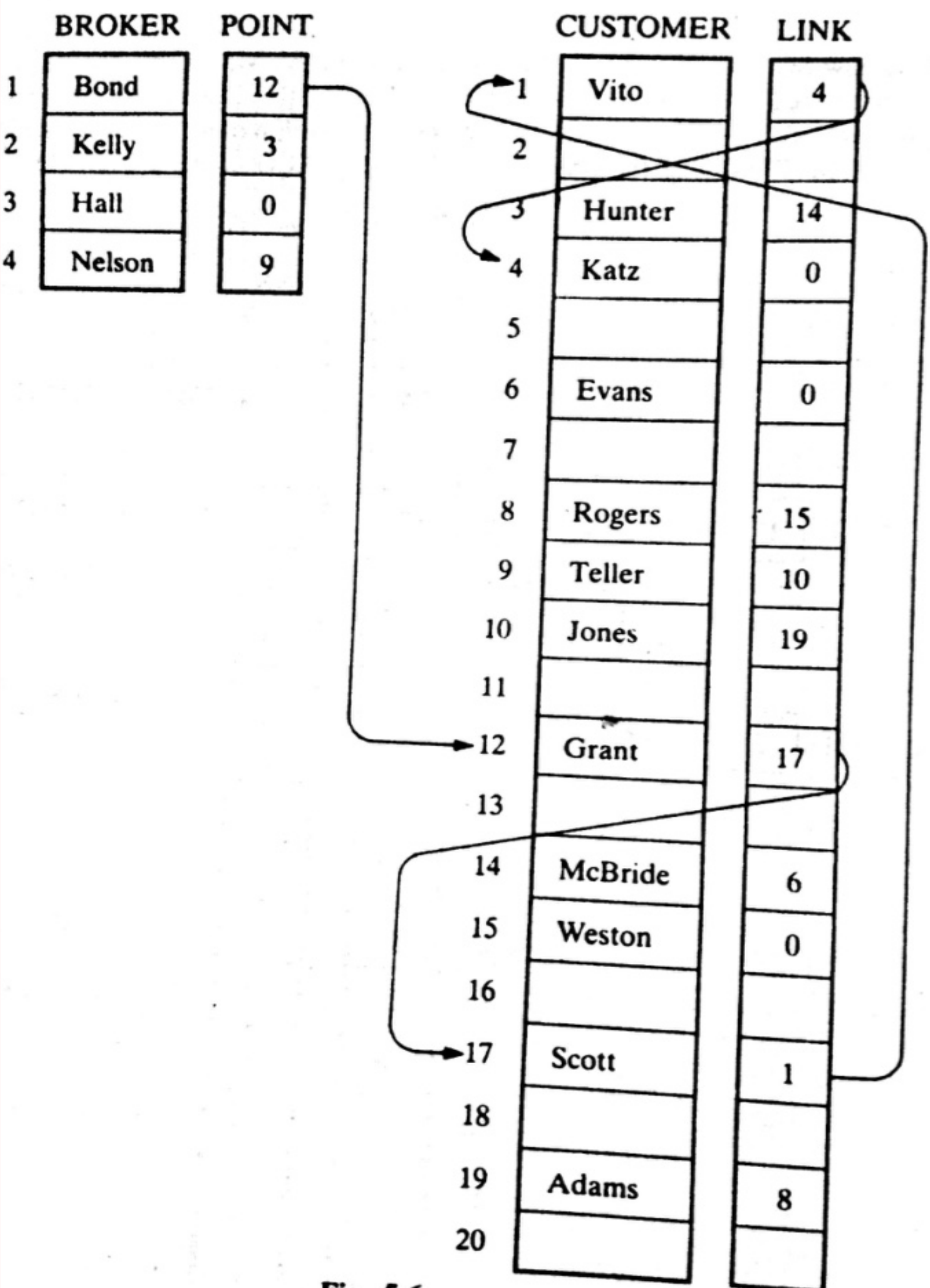


START 6

AVAIL 8

	NAME	SSN	SEX	SALARY	LINK
1					0
2	Davis	192-38-7282	Female	22 800	12
3	Kelly	165-64-3351	Male	19 000	7
4	Green	175-56-2251	Male	27 200	14
5					1
6	Brown	178-52-1065	Female	14 700	9
7	Lewis	181-58-9939	Female	16 400	10
8					11
9	Cohen	177-44-4557	Male	19 000	2
10	Rubin	135-46-6262	Female	15 500	0
11					13
12	Evans	168-56-8113	Male	34 200	4
13					5
14	Harris	208-56-1654	Female	22 800	3

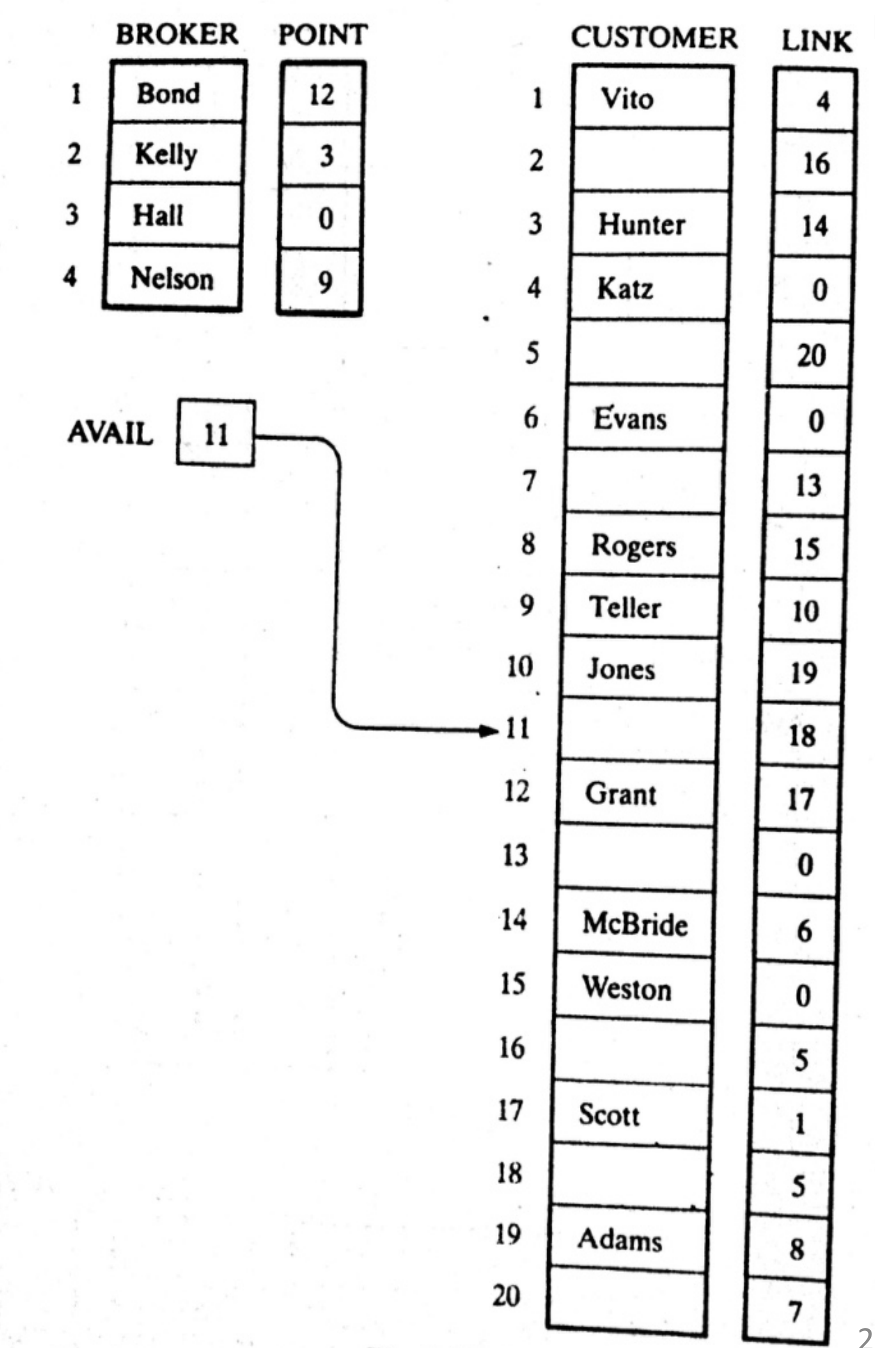
LINKED LISTS: Example 5.11(c)



	BROKER	POINT
1	Bond	12
2	Kelly	3
3	Hall	0
4	Nelson	9

	CUSTOMER	LINK
1	Vito	4
2		
3	Hunter	14
4	Katz	0
5		
6	Evans	0
7		
8	Rogers	15
9	Teller	10
10	Jones	19
11		
12	Grant	17
13		
14	McBride	6
15	Weston	0
16		
17	Scott	1
18		
19	Adams	8
20		

Fig. 5-6



	BROKER	POINT
1	Bond	12
2	Kelly	3
3	Hall	0
4	Nelson	9

AVAIL 11

	CUSTOMER	LINK
1	Vito	4
2		16
3	Hunter	14
4	Katz	0
5		20
6	Evans	0
7		13
8	Rogers	15
9	Teller	10
10	Jones	19
11		18
12	Grant	17
13		0
14	McBride	6
15	Weston	0
16		5
17	Scott	1
18		5
19	Adams	8
20		7

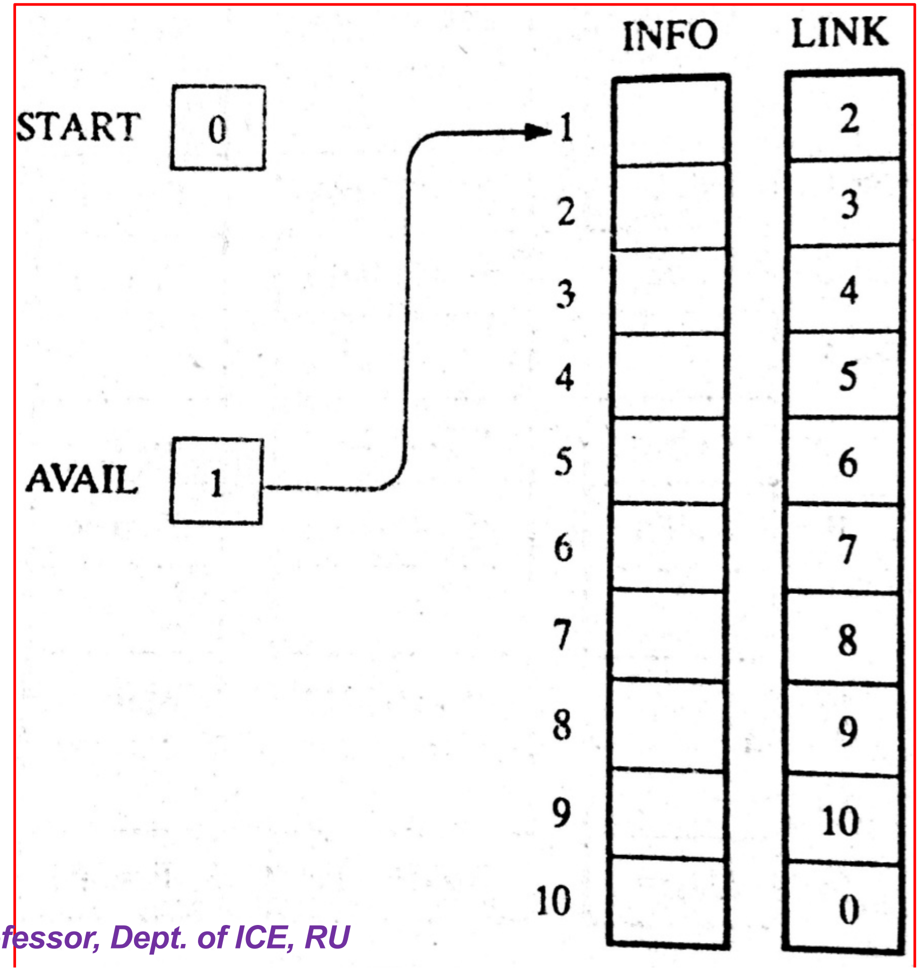
Fig. 5-12

LINKED LISTS: Example 5.12

- Suppose LIST(INFO, LINK, START, AVAIL) has memory space for n=5 nodes,
- Furthermore, suppose LIST is initially empty.

Your INSTATNT Task

Show the LINKED LIST which consists of START, AVAIL, INFO, LINK



	INFO	LINK
1		2
2		3
3		4
4		5
5		6
6		7
7		8
8		9
9		10
10		0

LINKED LISTS: Garbage Collection

Time Consuming Method for OS:

Time Efficient Method for OS:

The OS of a computer may periodically collect all the deleted space onto the free-space list. Any technique which does this collection is called *garbage collection*.

Garbage Collection usually takes place in **TWO** steps*:*

- ✓ Firstly, the Computer runs through the memory, tagging those cells which are currently in use, and
- ✓ Then, computer runs through the memory, collecting all untagged space onto the free-storage list.

LINKED LISTS: Garbage Collection

When garbage collection task executed by Computer?

The garbage collection may take place when …….

- There is only some minimum amount of space in the free-storage list, or
- There is no space at all left in the free-storage list, or
- When the CPU is idle and has time to do the collection

The garbage collection is invisible to the programmer

LINKED LISTS: Overflow

Sometimes new data are to be inserted into a data structure but there is no available space (the free-storage list is empty). This situation is usually called ***Overflow.***

Usually, Overflow will occur with our linked list when AVAIL=NULL and there is an insertion.

How we can handle Overflow situation?

We may handle the *Overflow* situation by.........

- ✓ Printing the message **OVERFLOW**
- ✓ We may modify the program by adding space to the underlying arrays

LINKED LISTS: **Underflow**

Sometime someone wants to delete data from a data structure where the data structure is empty. This situation is usually called **Underflow**.

Usually, underflow will occur with our linked lists when START=NULL and there is a deletion.

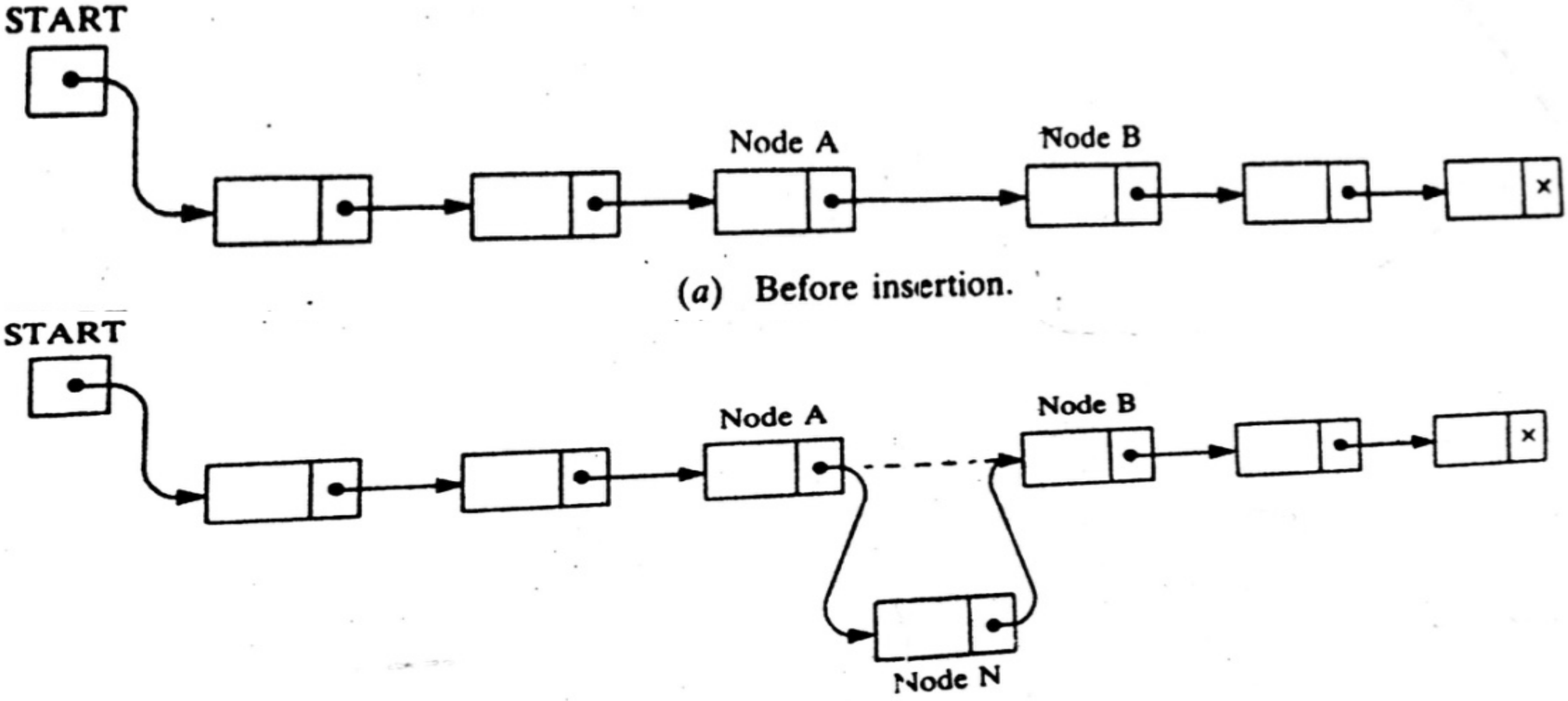
How we can handle Underflow situation?

We may handle the *Overflow* situation by……….

- ✓ Printing the message UNDERFLOW

LINKED LISTS: Insertion

Let LIST be a linked list with successive nodes A and B. Suppose, a node N is to be inserted into the list between nodes A and B.



(a) Before insertion.

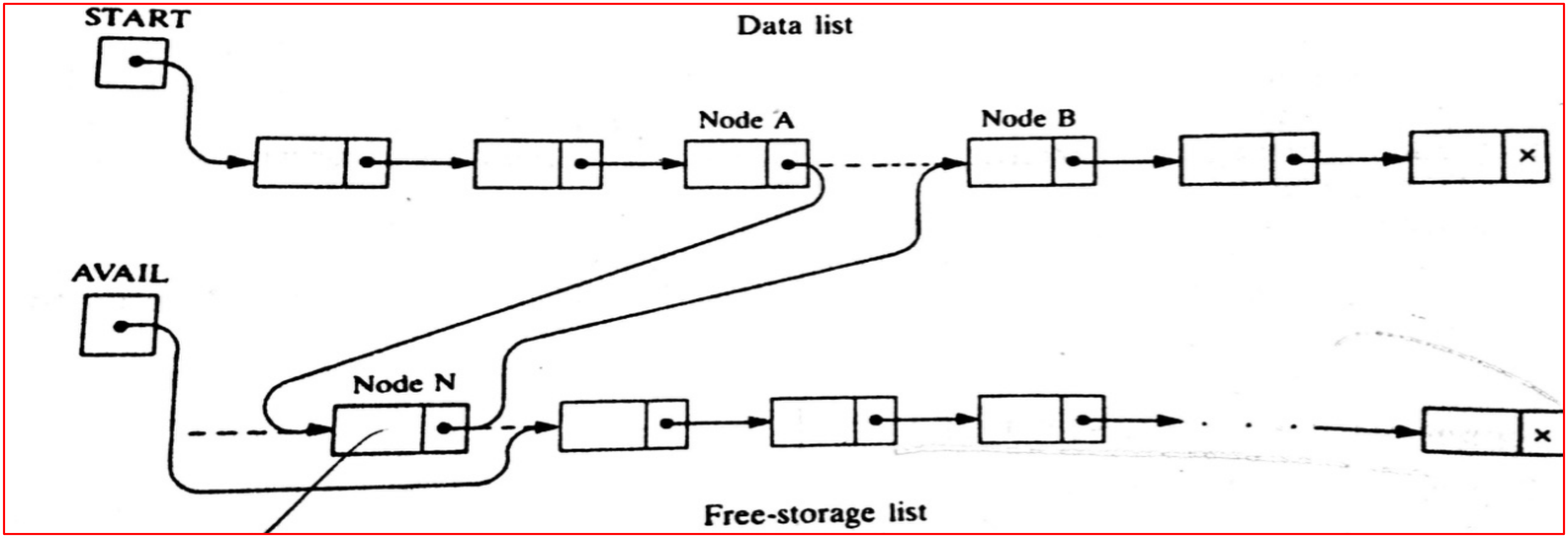
(b) After insertion.

Here, it does not take into account the memory space for the new node N will come from the AVAIL list.

But…..for easier processing, the first node in the AVAIL will be used for the new node N.

LINKED LISTS: Insertion

More exact Illustration of such an insertion



Here, THREE pointer fields are changed: ???

Which are?

- ➢ The nextpointer field of node A now points to the new node N.
- ➢ AVAIL now points to the second node in the free pool.
- ➢ The nextpointer field of node N now point to node B.

LINKED LISTS: Insertion

There are also TWO special cases:

- ➢ If the new node N is the first node in the list, then START will point to N, and
- ➢ If the new node is the last node in the list, then N will contain the null pointer.