ARPA ORDER NO. 189-1

MEMORANDUM
RM-6257-ARPA
MAY 1970

AD708025

THE GRAIL LOGICAL INPUT/OUTPUT PROCESSES

T. O. Ellis, J. F. Heafner and W. L. Sibley

DDC
RETURNED
JUL 6 1970
A

PREPARED FOR:
ADVANCED RESEARCH PROJECTS AGENCY

The RAND Corporation
SANTA MONICA • CALIFORNIA

THIS DOCUMENT HAS BEEN APPROVED FOR PUBLIC RELEASE AND SALE; ITS DISTRIBUTION IS UNLIMITED.

MISSING PAGE
NUMBERS ARE BLANK
AND WERE NOT
FILMED

MEMORANDUM
RM-6257-ARPA
MAY 1970

THE GRAIL LOGICAL INPUT/OUTPUT PROCESSES

T. O. Ellis, J. F. Heafner and W. L. Sibley

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0141. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of Rand or of ARPA.

DISTRIBUTION STATEMENT
This document has been approved for public release and sale; its distribution is unlimited.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

PREFACE

This Memorandum further documents the man-machine interaction studies sponsored by the Advanced Research Projects Agency. The data organization illustrates the needs of a typical on-line relational data file, such as might be found in a military command and control environment.

This Memorandum exemplifies techniques used in the implementation of one functional aspect of the GRAIL (GRAphical Input Language) system. A large part of this system is in continuing use at Rand by the Computer Research for Biomedical Graphics Project, sponsored by the National Institutes of Health, for transforming a portion of GRAIL into a system for continuous systems simulation. In addition, GRAIL communication techniques have been incorporated into several other research projects, both at Rand and elsewhere.

Persons interested in constructing interactive graphical systems may obtain detailed information on other functional aspects of GRAIL by contacting the Computer Systems Group, Computer Sciences Department, The RAND Corporation, 1700 Main St., Santa Monica, California, 90406.

SUMMARY

The data sets transmitted between primary and secondary storage by the logical input/output (I/O) processes of the GRAIL system are ring structures, display frames, text planes, and read-only process groups. The transfer of data is a matter of "reading" from secondary into primary storage and "writing" from primary into secondary storage. In either instance, the data transmission is synchronized whenever necessary with other events in the system, not only to provide good response time, but also to take advantage of the parallel processing organization of the system's software.

Ring structures, which represent storage space allocations and describe man-drawn processes and other system attributes, are of three kinds: file structures, context structures, and plane structures. Their hierarchical relationship is such that for every file structure there are many context structures, for every context structure many plane structures. A file structure serves as an entrance to a man's file. A context structure, associated with a closed-process definition, contains labels and the definition's hierarchy of planes. The plane structure describes the frames in a plane and the connectivity of graphical elements within each frame.

The display frame is the internal representation of the information displayed in a partitioned area of the cathode ray tube face. It comprises sets of display data, a table of code words that drive the display frame, and a software device for correlating stylus position with display graphics and their ring structure representations. By coupling and decoupling data segments, the system either can exchange pictures between primary and secondary storage or can permit the man to create new pictures from primary storage.

Text planes are character strings supplied by the man; read-only process groups are collections of executable programs. The internal organization of neither is examined by the logical I/O process; rather, they are treated as variable length data strings.

CONTENTS

FIGURES

I. INTRODUCTION

This Rand Memorandum is one of several on the operations and organization of the GRAIL system, and presupposes that the reader is already familiar with the first four Rand publications in the series.† It is concerned with the logical input/output (I/O) processes that, in support of other GRAIL operations, transmit data between primary and secondary storage.

The four types of data sets considered are *ring structures*, *display frames*, *text planes*, and *read-only process groups*. These sets may be defined as follows (Fig. 1):

1) Ring structures, i.e., list structures linked unidirectionally to form a ring, are used to represent:
   - o Secondary storage space allocations;
   - o The logical descriptions of man-drawn processes;
   - o Other system attributes.
2) Display frames are the internal representation of the information displayed in a logically partitioned area of the cathode ray tube (CRT) face.
3) Text planes are the internal CRT representation of a text display frame. They may include "off-screen" text if the text in the plane is more than can be displayed in the partition of the screen at any one time.
4) Read-only process groups are collections of reentrant procedures that are treated as a single data string when loading into or releasing from primary storage.

---

†The GRAIL Project: *An Experiment in Man-Machine Communications* (RM-5999-ARPA); *The GRAIL Language and Operations* (RM-6001-ARPA); *The GRAIL System and Implementation* (RM-6002-ARPA); and *The GRAIL System Ring Structures* (RM-6241-ARPA).

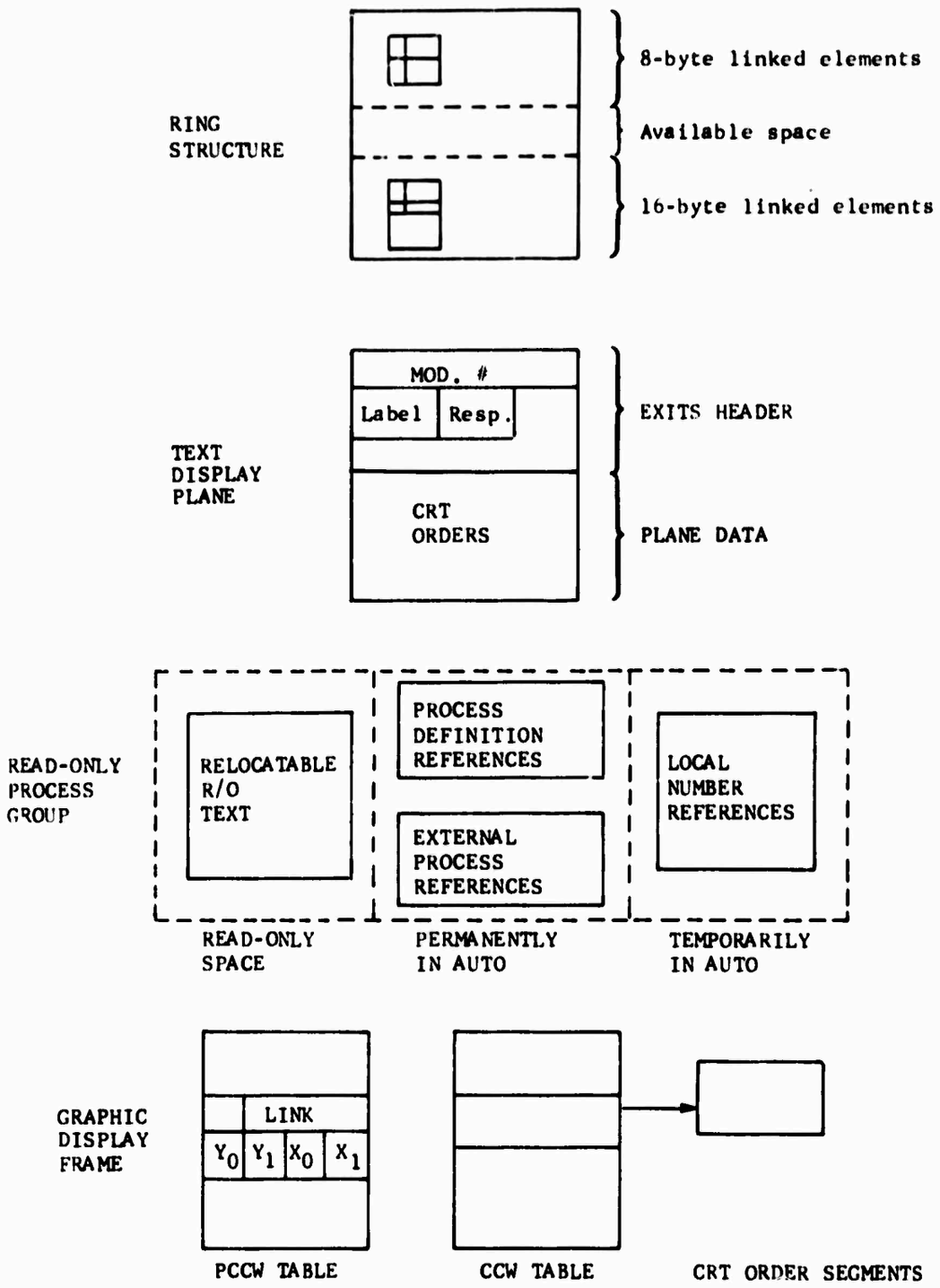


Fig. 1--Data Set Types

The present account deals in detail with the first two kinds of data sets. The last two are treated by the logical I/O processes as "black-boxes."

The data sets in secondary storage have the overall format illustrated in Fig. 2. The 20-byte header is of fixed length but contains information peculiar to each type (Fig. 3). The data set itself, as distinguished from the header, is of variable length.

The secondary storage devices are IBM 2311 disk packs consisting of 200 cylinders, each cylinder being divided into 10 heads, each head containing 3625 data bytes. All transmission is one full head, which means that if GRAIL substitutes one bit of data on a head for another, it does so by rewriting the entire head, including the data that form no part of the substitution. The tracks are formatted with an IBM standard length Record Zero (R0) and Count Field of Record One. The Record One (R1) data length is 3625 bytes (Fig. 4). The data portion of each head is marked logically, rather than physically, into areas designating the number of entries in the head, the relative addresses of the data sets, and the data sets themselves.

The transmission of data is synchronized, wherever necessary, with other events in the system in order to provide good response time and to take advantage of the parallel processing organization of the system software.

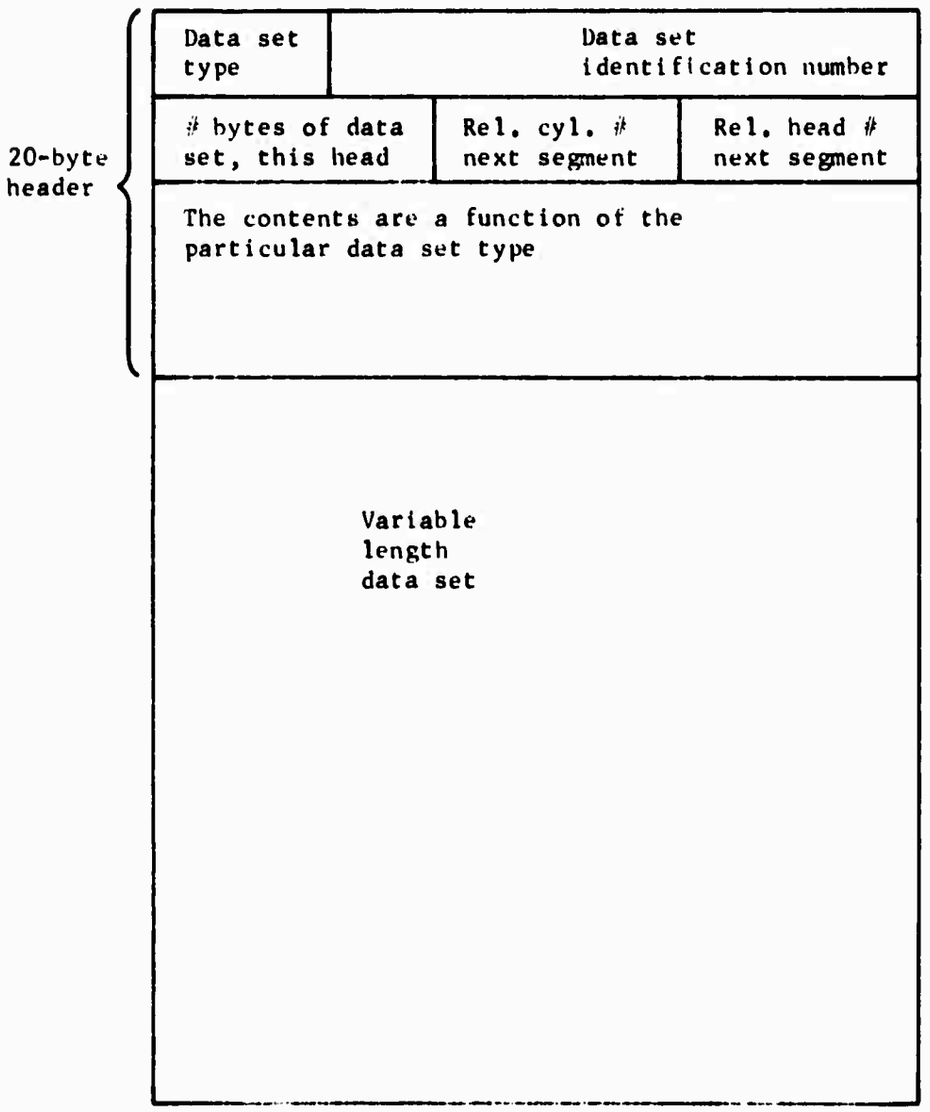


Fig. 2--Secondary Storage Data Set Format

Type-80
PROCESS
GROUP

80	
DT (definition section) length, this head	LR (label ref. section) length, this head
NR (# ref. section) length this head	TXT (text) length this head
TXT (text) length total for P.G.	Not used

TYPE-81
STATIC DISPLAY

Type-85
TRANSIENT
DISPLAY

81/85	
PCCW length this head	CCW length this head
Display length this head	Not used
Not used	Not used

Type-82
RING STRUCTURE

82	
8-byte length this head	16-byte length this head
Structure space length in primary core	Skeleton length this head
8-byte length total	16-byte length total

Type-83
TEXT
PLANE

83	
Data set (this head) length excluding exit header	Not used
Not used	Not used
Not used	Not used

Fig. 3--Individual Secondary Storage Header Formats

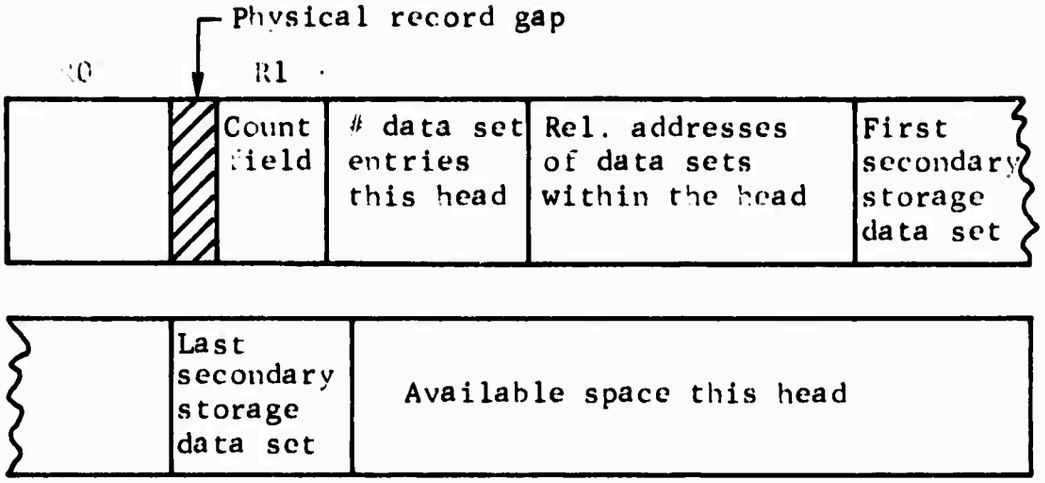


Fig. 4--Disk Head Format

II. RING STRUCTURES

FILE STRUCTURES

Ring structures are of three different kinds: file structures, context structures, and plane structures. We deal first with file structures. The input/output of data in this instance is a matter of reading and writing a file structure.

To understand a file structure, consider Fig. 5. The large outside rectangle represents a disk pack (secondary storage). It contains a number of files. The dashed lines mark a man's file by location and extent. The inner horizontal box contains a series of data sets for that file. The smaller vertical shaded box is the object of our immediate attention--the file structure. It functions as a door or entrance to that particular file. The information the system needs to gain entrance to any file is contained in a directory--the horizontal box at the top of the pack--called the *Files Description Structure* (FDS). It provides three indispensable pieces of information about each file in the pack: its location, its extent, and the location of the door (the file structure) to the file.

The process of reading a file structure is relatively simple. It consists of:

1) Retrieving the directory (Fig. 6);
2) Finding the appropriate information (the location and extent of the file structure;
3) Retrieving the particular file structure by reading it into primary storage (Fig. 7).

It is important to note that retrieving consists of reading not the thing itself, but its image from secondary storage to primary storage. If a machine fault occurs, nothing is lost but the image.

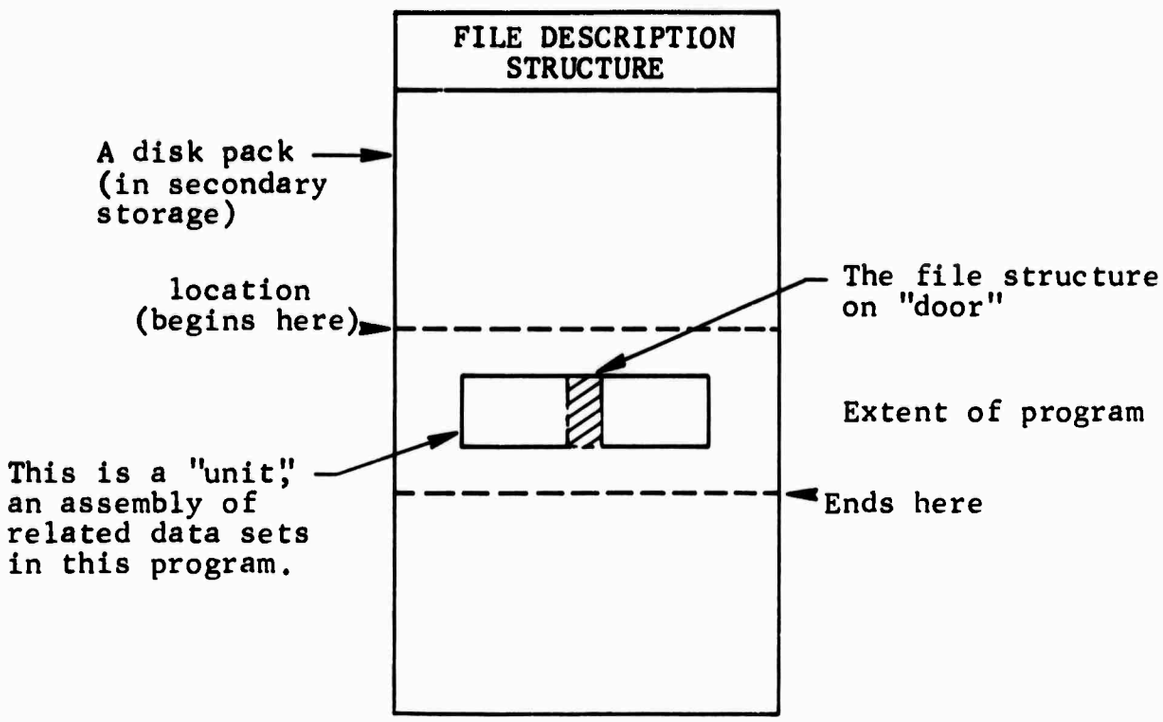


Fig. 5--Secondary Storage Organization

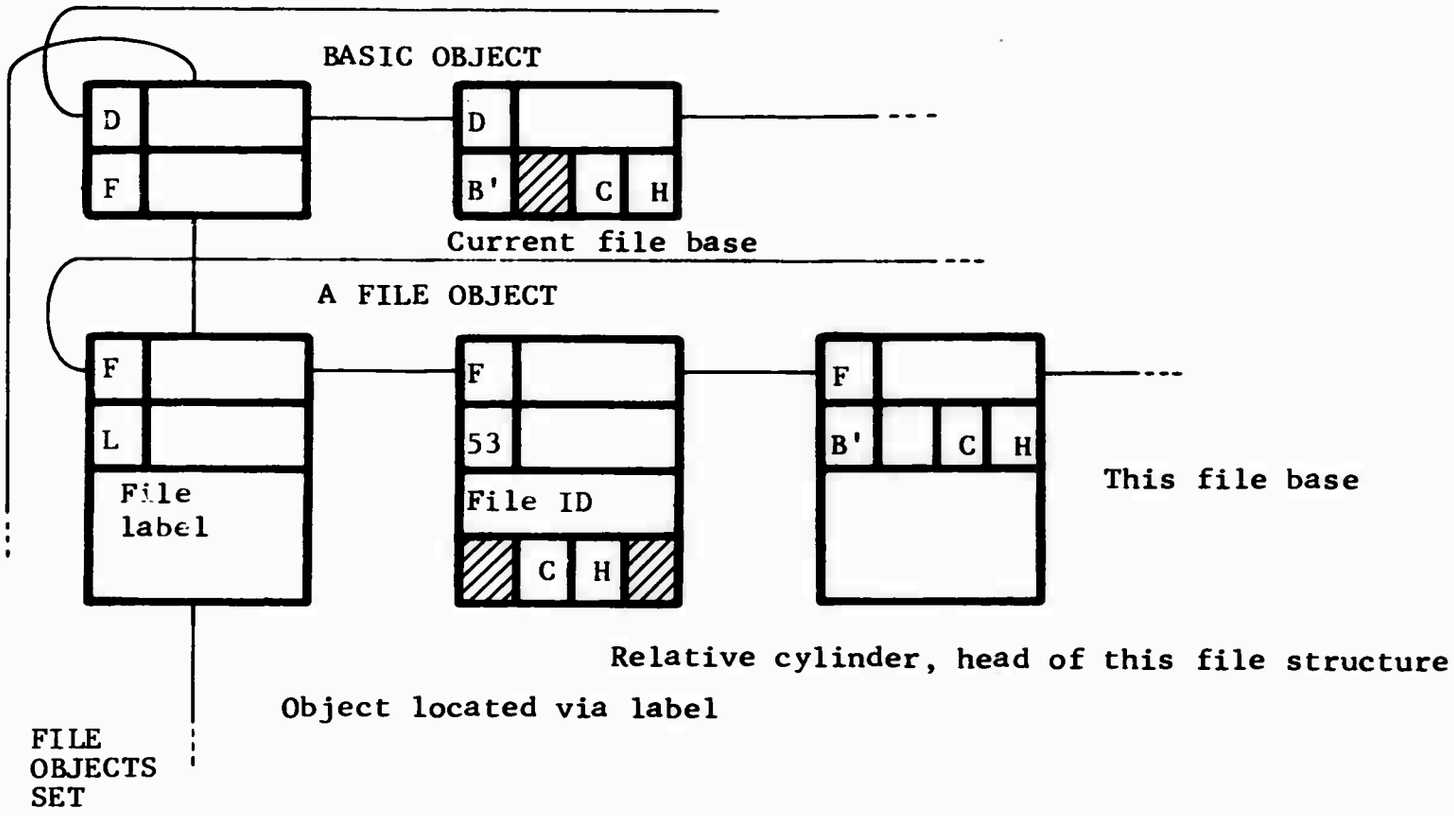
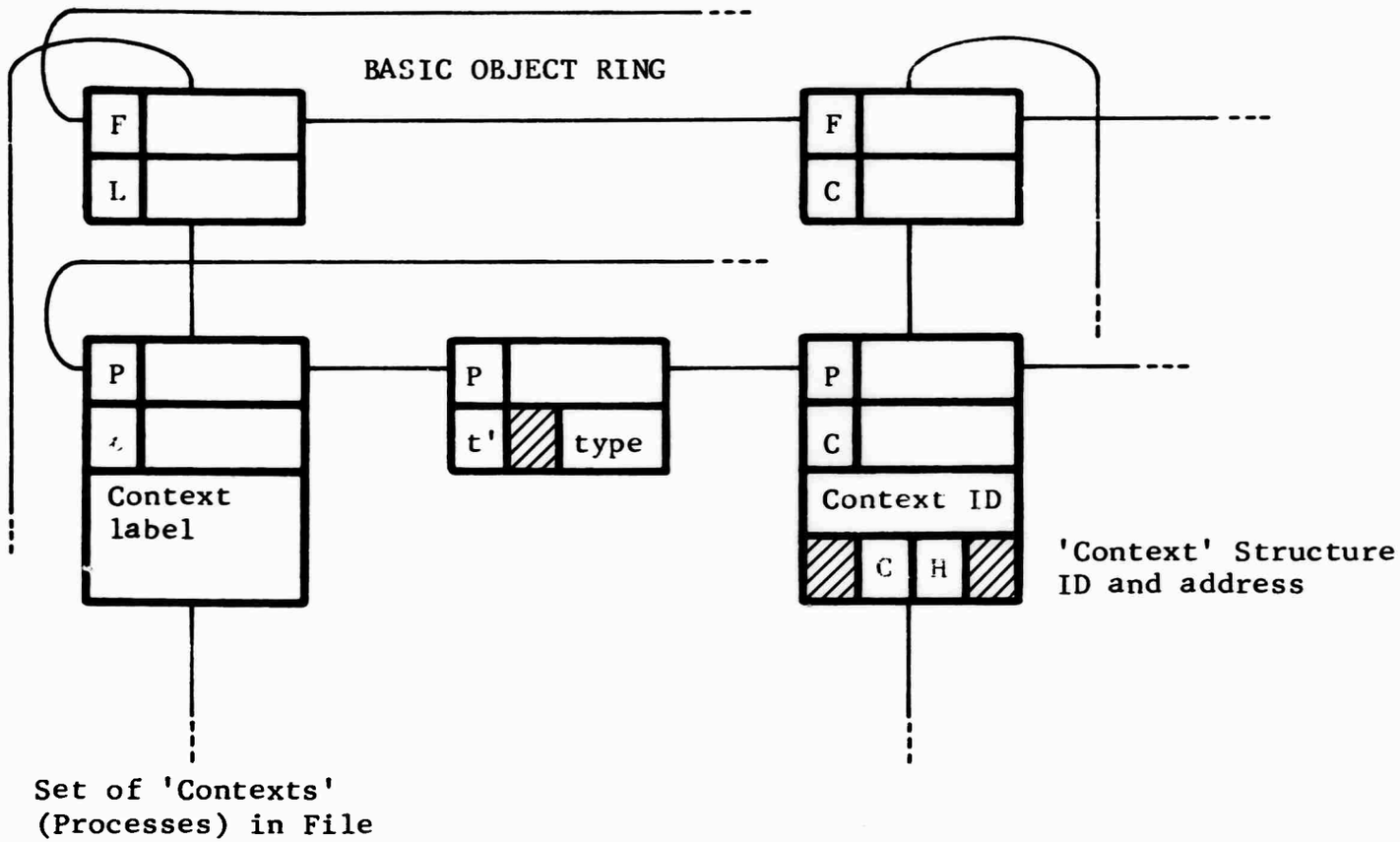


Fig. 6--Files Description Structure



The top 'context' type is output when reading a user File Structure.

Fig. 7--File Structure

Writing a file structure is more complicated and calls for some preliminary information. Understand first that the space for data sets is dynamically allocated within the man's file. He gets what he needs without explicitly requesting it. This means that the length of the new or rewritten file does not necessarily have to be the same as that of the old file structure. Recall again that the file structure occupies a part of a larger receptacle called a disk head.

The first step in writing a file structure is to retrieve the old file structure by the same procedure (via the directory) outlined for reading a file structure. The only significant difference is that, not just the file structure alone, but the head of which it is a part is copied into primary storage. The reason for this is easily seen. Unlike reading a file structure, writing a new one or simply amending the old one may presumably require more or less space than was originally allotted to it on the head. Some adjustments have to be made.

Ignoring the details of this adjustment, described in Sec. IV, and with the image of the head and its inner file structure now read into primary, let us return to the GRAIL user. In response to some stylus movement on the RAND Tablet, the system reads the old file structure in order to delete it, and then transfers the new one to secondary storage. Since the file structure is now likely to occupy a new location, the only remaining task for the system is to update the directory so that the new information can be retrieved when needed.

We come now to context and plane structures. There hierarchical relationships are such that for every file structure there are many context structures; for every context structure, many plane structures.

CONTEXT STRUCTURES

To write a context structure, the system creates a skeleton (Figs. 8 and 9) or abstract of certain data in the context structure. The skeleton exists:

1) To provide the *formal parameters* (dummy data labels) and their descriptions for the *translation display* (the data are put into a display format suitable for the CRT);
2) To update the *exit labels* for the display of a process instance;
3) To compile a process;
4) To interpretatively execute a process.

The system then finds, in the file structure, the address of the old data set in secondary storage. It writes the context structure in secondary, first destroying the old context structure (detailed in Sec. IV). Finally, the address of the new data set, now written into secondary, is recorded in the file structure.

To read a context structure, the system finds its address in the file structure, and then reads it into primary. It may swap context structures, in which case it writes a context structure and then reads a different one. The system then reads a context skeleton (independently of the context structure itself).

PLANE STRUCTURES

Writing a plane structure is analogous to writing a context structure, with two exceptions. The skeleton (Fig. 10) of a plane structure consists only of exit labels, and the address of the plane structure is in the context structure.

Reading a plane structure is like reading a context structure except that the plane structure's address is located in the context structure (Fig. 11).

Mod # (4 bytes)
α { CRT position and intensity information Blanks to be replaced by translated label (fixed length) Formal parameters label (8 bytes) Comments from label description structure (variable length)
(Repeat α for each formal parameter to the context)
Delimiter
non-parallel exits (2 bytes)
parallel exits (2 bytes)
String of 1 character non-parallel exits (variable length)
String of 1 character parallel exits (variable length)

Fig. 8--Context Skeleton Format

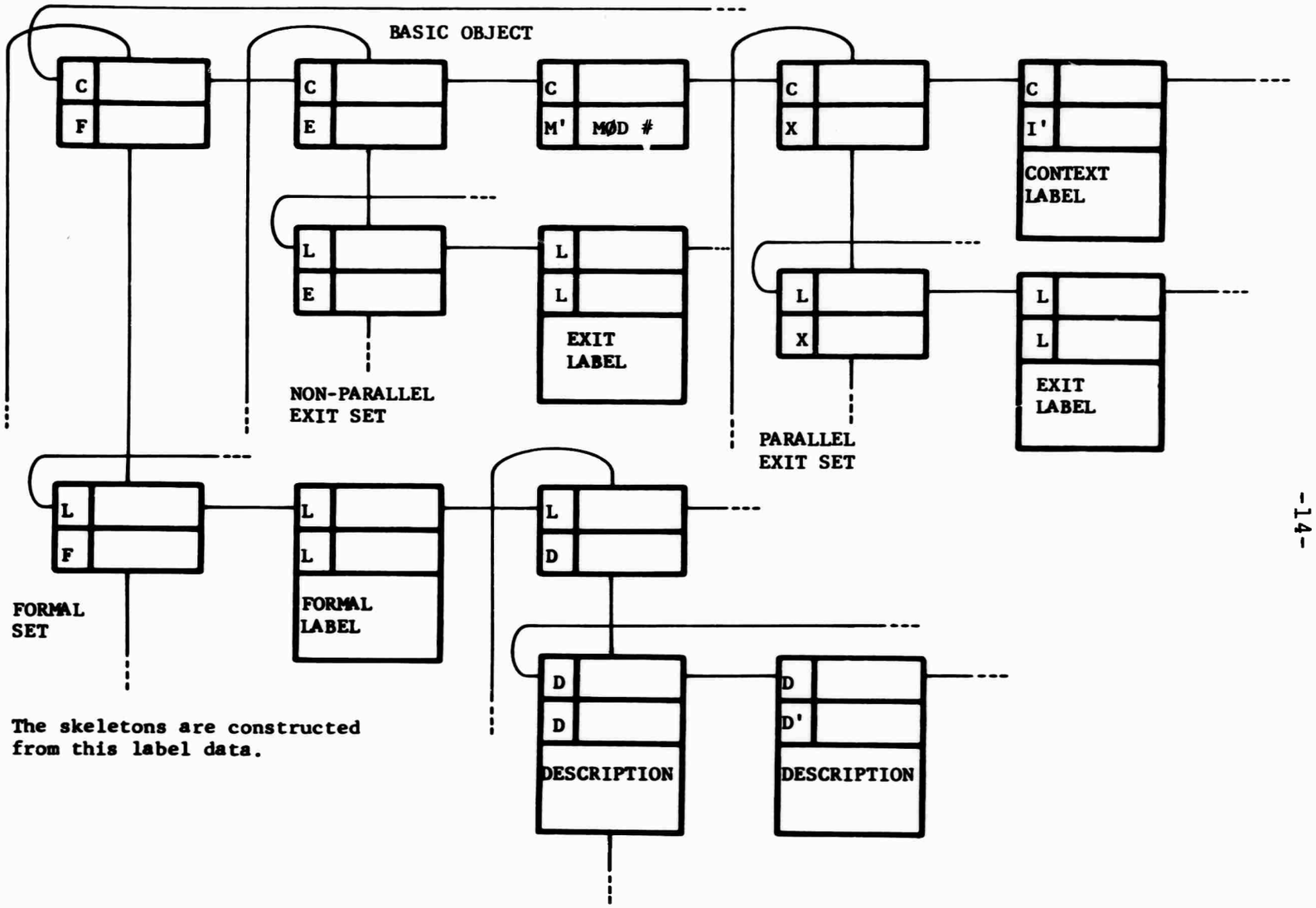
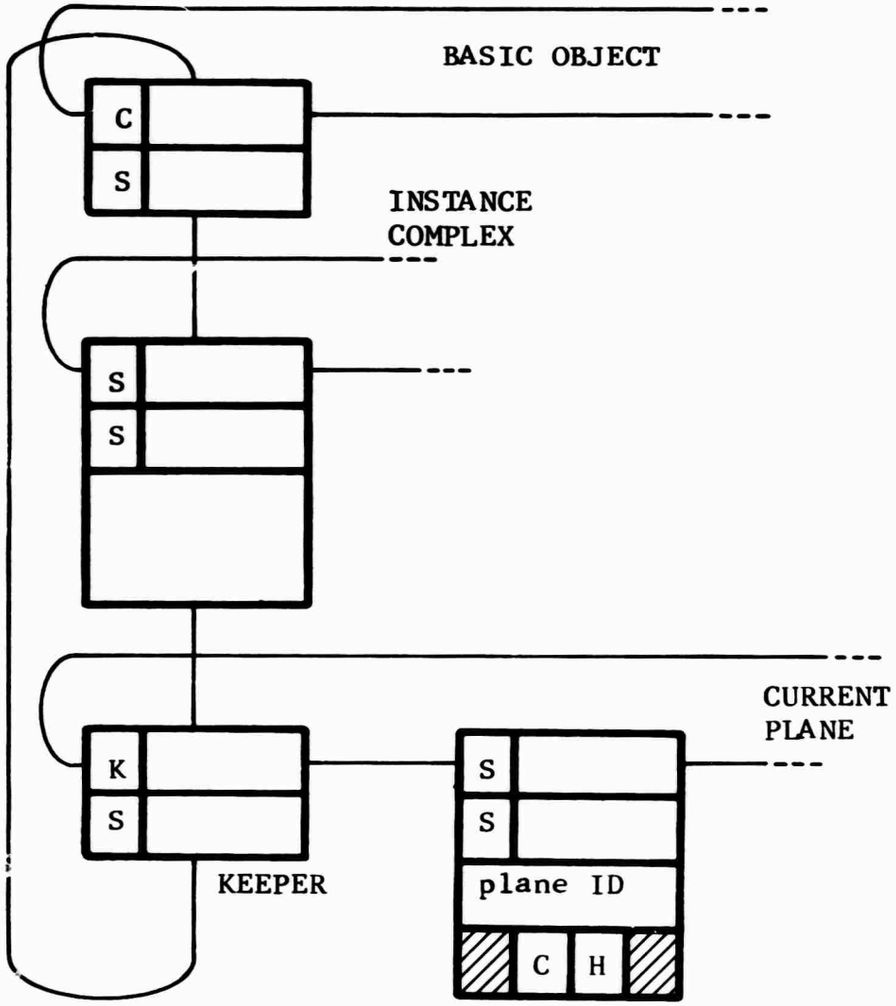


Fig. 9--Context Structure

Mod # (4 bytes)
CRT intensity, position, and delimiter data
non-parallel exits (2 bytes)
parallel exits (2 bytes)
String of 1 character exit labels non-parallel (variable length)
String of 1 character exit labels parallel (variable length)

Fig. 10--Plane Skeleton Format



The KEEPER is used to locate the current plane ID and location. The KEEPER is relocated after a read context.

Fig. 11--Context Structure

Swapping plane structures involves first writing and then reading a plane structure. Finally, the plane skeleton alone may be read independently of the plane structure.

EMPTY STRUCTURES

It may happen that a context or a plane structure is empty of all user-supplied data. A context structure is said to be empty if it has no planes; a plane structure, if it has no frames. If it is empty, then a request to write merely results in the old secondary storage image being deleted.

III. THE DISPLAY FRAME

The CRT face can display several frames at the same time, and each display frame can be made up independently of the others. A display frame can be brought in from secondary storage or coupled to the others already displayed. Conversely, it can be independently removed by decoupling and then recording in secondary storage.

Let us begin with the GRAIL user who needs different information from what he has immediately before him. One way of getting it is to swap pictures. In response to his signal, the GRAIL system automatically relegates the old picture from primary to secondary, and retrieves from secondary storage the picture called for.

Another way of getting more information is to supply it by constructing a new picture in primary. Though pictures may be called from secondary, they are always generated for display purposes from primary. A new picture begins with a blank, something to write on. A blank in this sense is something like an unused bank check. It is not entirely blank, for, like a check, it bears several items of information corresponding to the bank's name and address, code numbers, spaces to be filled in by the payor, etc. Such items are analogous to the static elements in the blank. By filling in the blank, that is, by supplying the dynamic elements, the GRAIL user constructs a new picture. At this point, he can amend it, add to it, or destroy it entirely. When he shuts down, the system files his picture in secondary storage where it can be retrieved when needed.

The foregoing account is very broad-brush. One detail that needs mention is the matter of coupling and decoupling. All the display data segments in primary storage are so interrelated that to remove one of them to secondary storage, or simply to destroy it, one must first break its

connections with all other segments, taking care, of course, to reconnect the chain where it has been broken. An item that has been brought into primary storage, whether it comes from secondary or represents newly created data (a filled-in blank), must be connected to what is already there.

So much for the user's manipulations. Out of sight and out of mind are the inner workings of the system, notably the operations upon the display frame. This is the data set that constitutes the virtual or conceptual image in primary storage, whose real or visual counterpart is mirrored in the CRT display.

Roughly speaking, the display frame consists of three parts. First, there is a table of code double words labeled the "Channel Command Word" (CCW). Each entry in the table contains a two-word code associated with an assortment of data (the second item) and which specifies the address and extent of the data. The CCW table supplies the display channel with the virtual or conceptual image that the CRT hardware can turn into a visual image. The third item in the display frame is the table of *Pseudo Command Channel Words* (PCCWs). Its pseudonymous relation to the CCW is in respect to its structure, not its function. Structurally, it bears a one-to-one relation to the CCW: code word, address, and the geometric boundaries of the displayed element. Its function, however, is that of an interface between the inner workings of the display frame and the outside world, namely, the GRAIL user and his intent. It can be conveniently thought of as a plugboard (Fig. 12). Together, the PCCWs, the CCWs, and the display data compose the virtual picture.

What the display frame does can be made intelligible by considering it in a larger context. Return to GRAIL user.

Assume that the man, stylus in hand, draws on the RAND Tablet what purports to be a square. As he does so, the

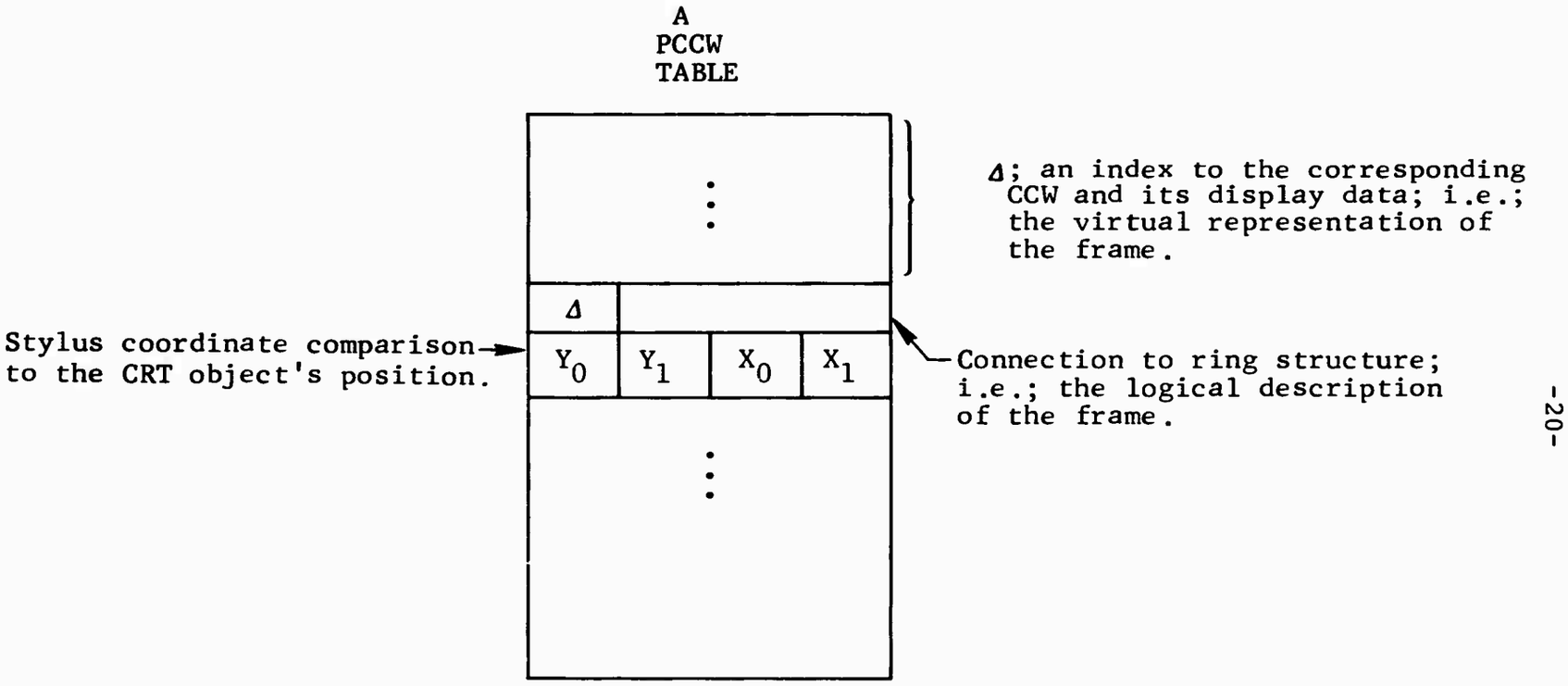


Fig. 12--A PCCW Table

position and movements of the stylus enter the system. At this point, the system programs take over the operation. To function as an operator, the programs need an interface between: the stylus, the logical meaning of the picture being drawn, and the virtual picture; the PCCW provides this interface. The "logical meaning of the picture" means simply that the system must recognize both the drawing for what it is--in this case a square--and the logical relationship of the elements in the picture, that is, the connections, for example, between the square and other graphics. The data for this analysis are provided by a ring structure (Sec. I).

In possession of this information, the program builds the necessary CCW and display data for generating the picture that the user wants.

READING AND COUPLING

It has been said before that the input and output of data involve the transfer of data between primary and secondary storage. We have spoken of the output of data (the transfer of data from primary to secondary) as writing, and the input of data (the transfer of data from secondary to primary) as reading. We have also made the point that reading and writing require coupling and uncoupling. But this is an overall view. Let us consider now the details of these operations, beginning with reading and coupling:

1) The process begins by locating the display frame data in secondary storage via their addresses in the plane structure, currently in primary storage. When found, the display frame is read into primary storage.
2) The next step is to connect the interface (the PCCW) with the logical description of the display frame data, namely, the plane structure. This

connection is made by replacing the internal PCCW elements on the frame's *CCW object rings* by their corresponding, retrieved, external PCCW elements. The correspondence between internal and external elements is identified by the PCCW displacement (Fig. 13).

3) Finally, the display frame is connected to the display channel by means of the CCW link to the active channel program. This is effected by inserting a new display channel segment immediately after the initial segment. The insertion requires three discrete steps (Fig. 14):

   a) An entry for the new segment (the new display frame) is placed in the *Area Control Block* (ACB) (Fig. 14).
   b) The head PCCW for this frame is completed by supplying the address of the CCW table, the address of the CCW Transfer-in-Channel (TIC), and the number of the remaining PCCWs.
   c) The actual channel program linkage is made as follows:
      - o The current TIC is supplied with the code of the next channel-program segment, and the address of the previous channel-program segment's TIC;
      - o The current channel-program segment's TIC address is placed in the TIC of the next channel-program segment;
      - o The next segment's TIC code in the initial segment is updated by inserting the current channel program segment's code;
      - o The channel-program segment's TIC address of the current channel-program segment is placed in the initial segment's TIC.

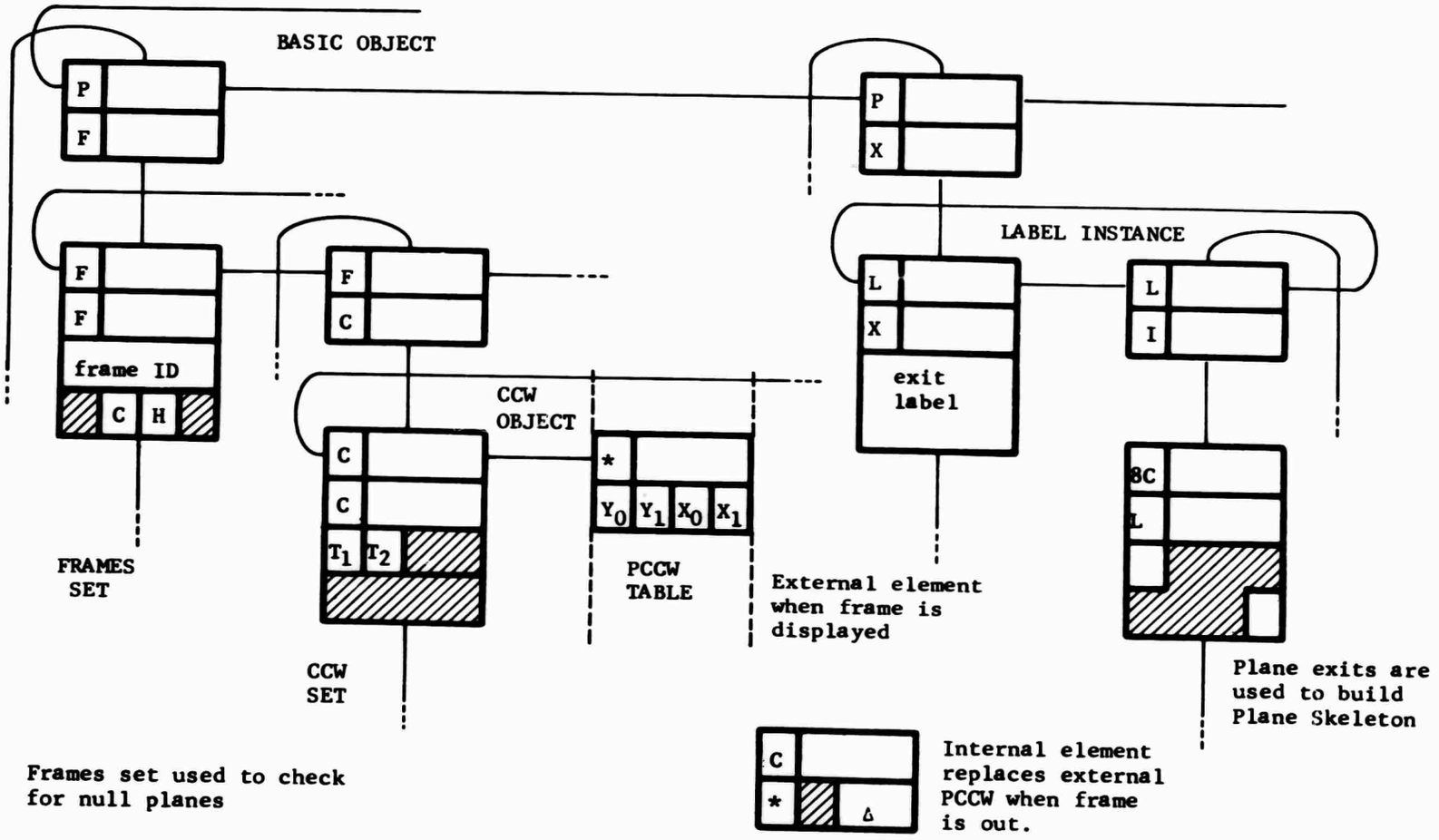


Fig. 13--Plane Structure

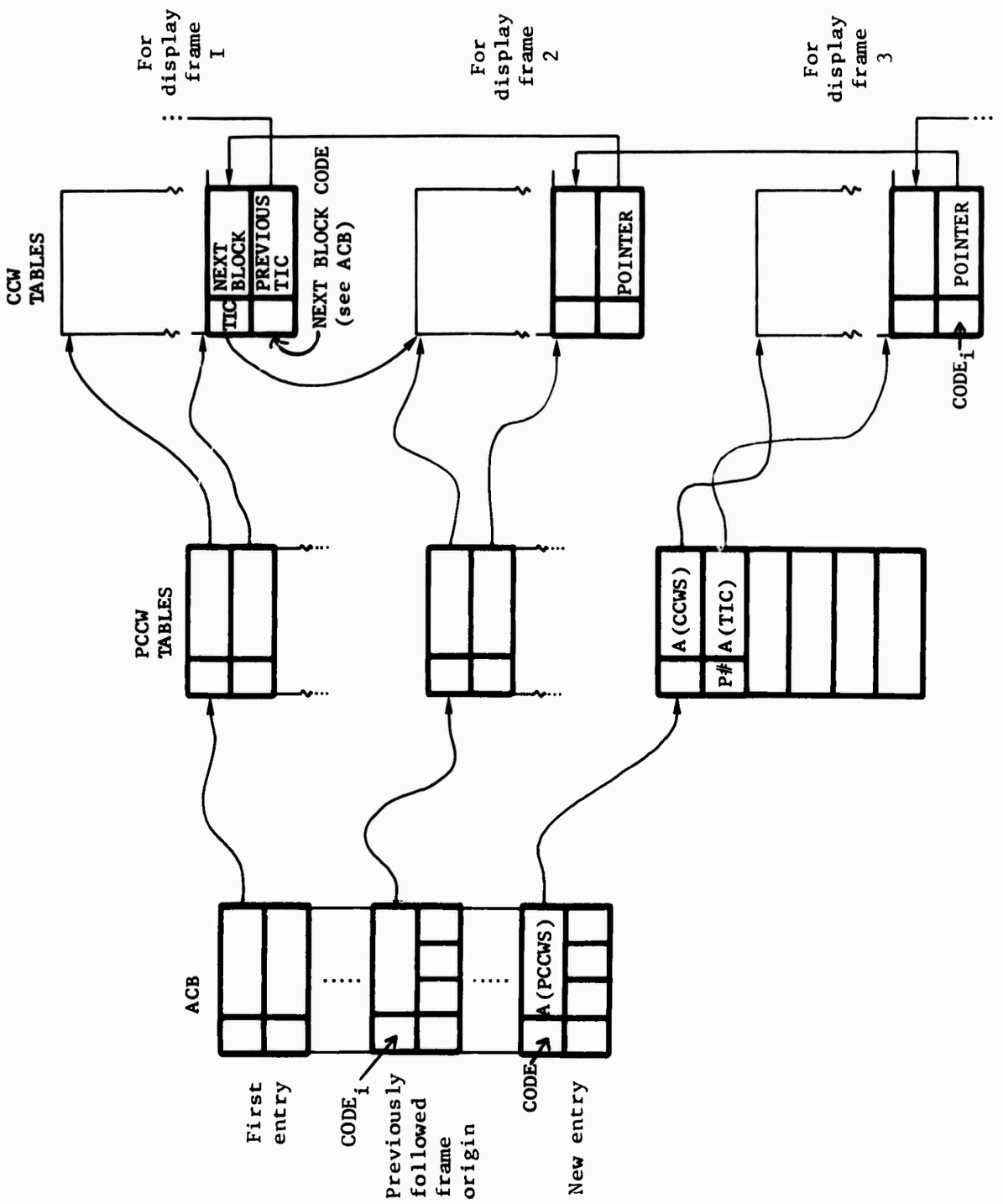


Fig. 14--Display Data Manipulated by MANAGE

Thus far we have been concerned with reading and coupling, the first option. There are two other options: the second is to create a blank frame and couple it into the display frame channel program; the third is to couple in an already given frame. The second option requires only steps 2) and 3) above; the third, only step 3).

Step c) involves synchronizing the occurrence of one event with the non-occurrence of another. There are two instances:

1) While coupling frames, the GRAIL system ignores any stylus gesture, whether advertent or inadvertent.
2) The activity of processing a frame through step c) prevents the linkage of any other display frame. Thus, only one frame at a time can proceed through step c).

DECOUPLING AND WRITING

Decoupling and writing, taken together, is the process of first removing a frame from its position in the chain of channel program segments, and then writing it into secondary storage. Figure 14 shows three program segments or frames, numbered 1-3. Decoupling Number 2 and writing it into secondary are carried out in three steps:

1) The display frame is disconnected from the active display channel program by relinking the CCW TICs to exclude this frame from the display channel program. The relinking of segments 1 and 3 occurs when:
   - o Both the code and the address of Number 3's channel program segment are placed in the TIC of Number 1's channel program segment.
   - o The address of Number 1's channel program segment's TIC is placed in Number 3's channel program segment.

2) The interface is then disconnected from the logical description of the frame as follows: The PCCWs are disconnected from the *CCW object rings* in the accompanying plane structure. The PCCWs' external ring elements on the *CCW object rings* associated with the frame are disconnected, and an internal PCCW element is added to the *CCW object ring* and carries the PCCW displacement in the PCCW table.
3) The frame is then written into the man's file in secondary storage.

Decoupling and writing constitute only the first option. There are two others, both analogous to those discussed under coupling and reading. The second option is decoupling and disconnecting the interface from its logical description in the plane structure. The third is decoupling only. The second option involves steps 1) and 2) above; the third, only 1).

The removal of the segment from the active display channel program is synchronized so that only one frame can be decoupled at a time.

IV. READING AND WRITING: THE DETAILS

WRITING

In writing from primary to secondary, the system economizes on secondary spaces by compacting the data. This also economizes transmission time between primary and secondary. The data sets written from primary to secondary are ring structures, display frames, read-only process groups, code planes, skeleton structures, and code-plane skeletons.

If the data set has already been written, the old version is destroyed, not when it is read out, but when the new one is written. This precaution is taken so as to prevent an accidental loss of the old before the new has come into being. Of course, old data may be destroyed without writing new data. The system identifies the data set to be deleted by comparing identifiers in the header of each data set, and gets the address of the head containing the next segment of the data set (Fig. 2, p. 4).

The *Files Description Structure* and the *System Structure* must be rewritten at their same locations; any other data set will be written at a dynamically assigned location.

The existence of certain data sets defined in the system is relatively permanent; the existence of others created by the user is relatively transient. When written in secondary storage, the address of the permanent-type data set is recorded in the structure by the logical I/O. The relative address of process groups and static displays is inserted on the *occupied disk set*. The secondary storage address of the transient-type data set is passed up to the higher-level process that invokes the logical I/O for recording in the appropriate ring structure.

In writing a new data set, any pre-existing version is destroyed or deleted. Heads containing data that must be kept are a special consideration, for they must be compressed

while being rewritten (Fig. 15). When this occurs, a number of adjustments have to be made.

Consider the following illustration (Fig. 16) of a head in which data set Number 2 (D_2) is deleted--and along with the data set, its address A_2. Four adjustments follow:

1) The count (of data sets in the head) is reduced by one.
2) Because A_2 is also expressed from the head, D_1 will be rewritten in a new position to the left, at a distance equal to the width of A_2. Its new address is therefore noted in A_1 simply by subtracting this distance or length from its old address.
3) The address of any data set beyond D_2 has to be doubly adjusted since it will be rewritten to the left at a distance equal to A_2 plus D_2. In this instance the new address of D_3 is recorded in A_3 by subtracting from its old address the sum of A_2 and D_2.
4) Finally, the sum of A_2 and D_2 is added to the available space at the extreme right.

Since we now have a head that is partially available for future use, this fact is noted in the *Partially Available Heads* (PAH) structure (Fig. 17).

The physical adjustments that have just been described occur whenever the head contains data that must be kept intact. These adjustments may be likened to cutting and splicing audio tape so as to get an uninterrupted playback on a tape recorder.

A logical adjustment has to be made when an entire head already contains data that are no longer needed. Nothing needs to be erased since the system will delete all this while recording new data, just as a tape recorder erases an old recording as it records the new. Accordingly, the logical adjustment consists only of noting in the *Available*

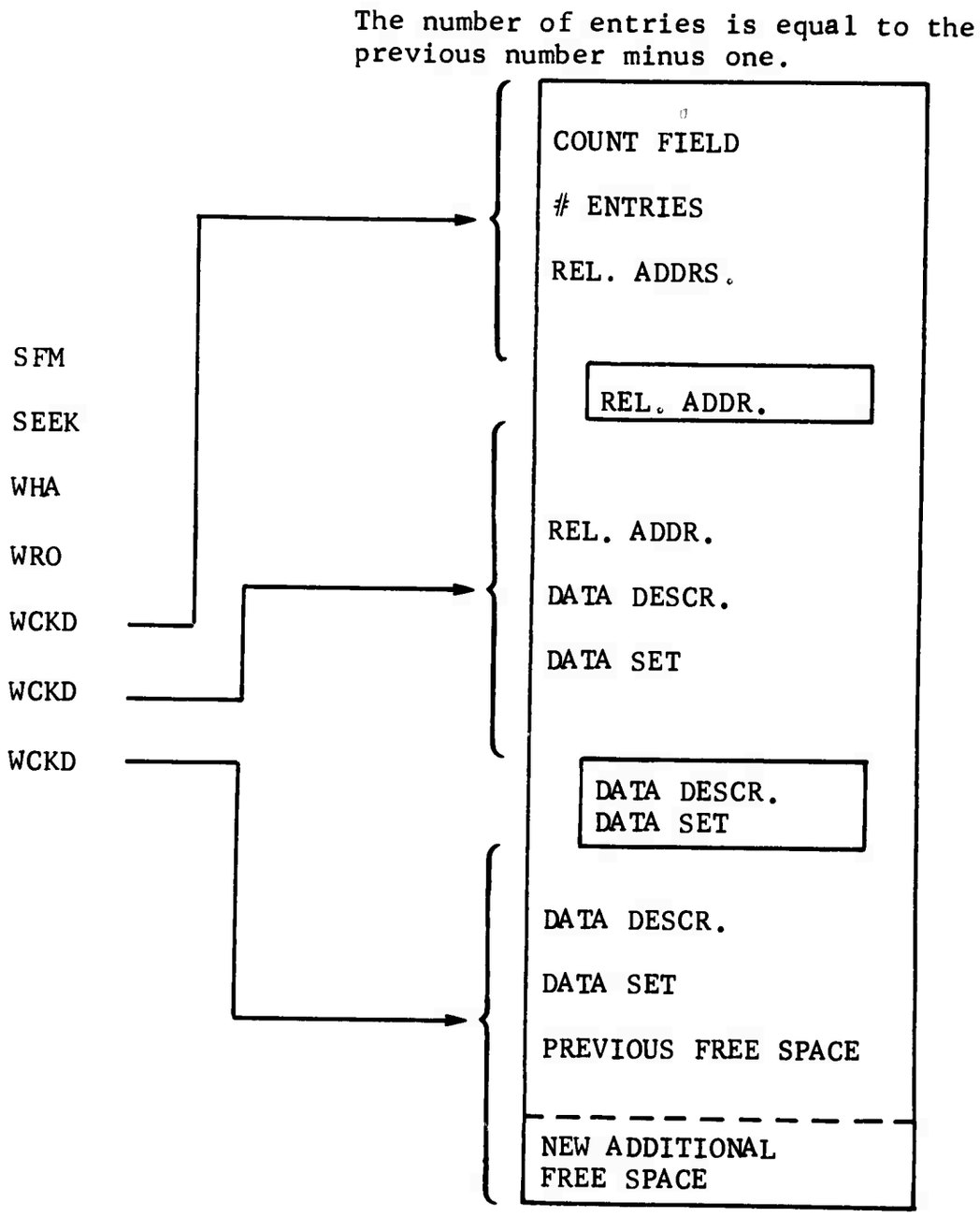


Fig. 15--Write Channel Program

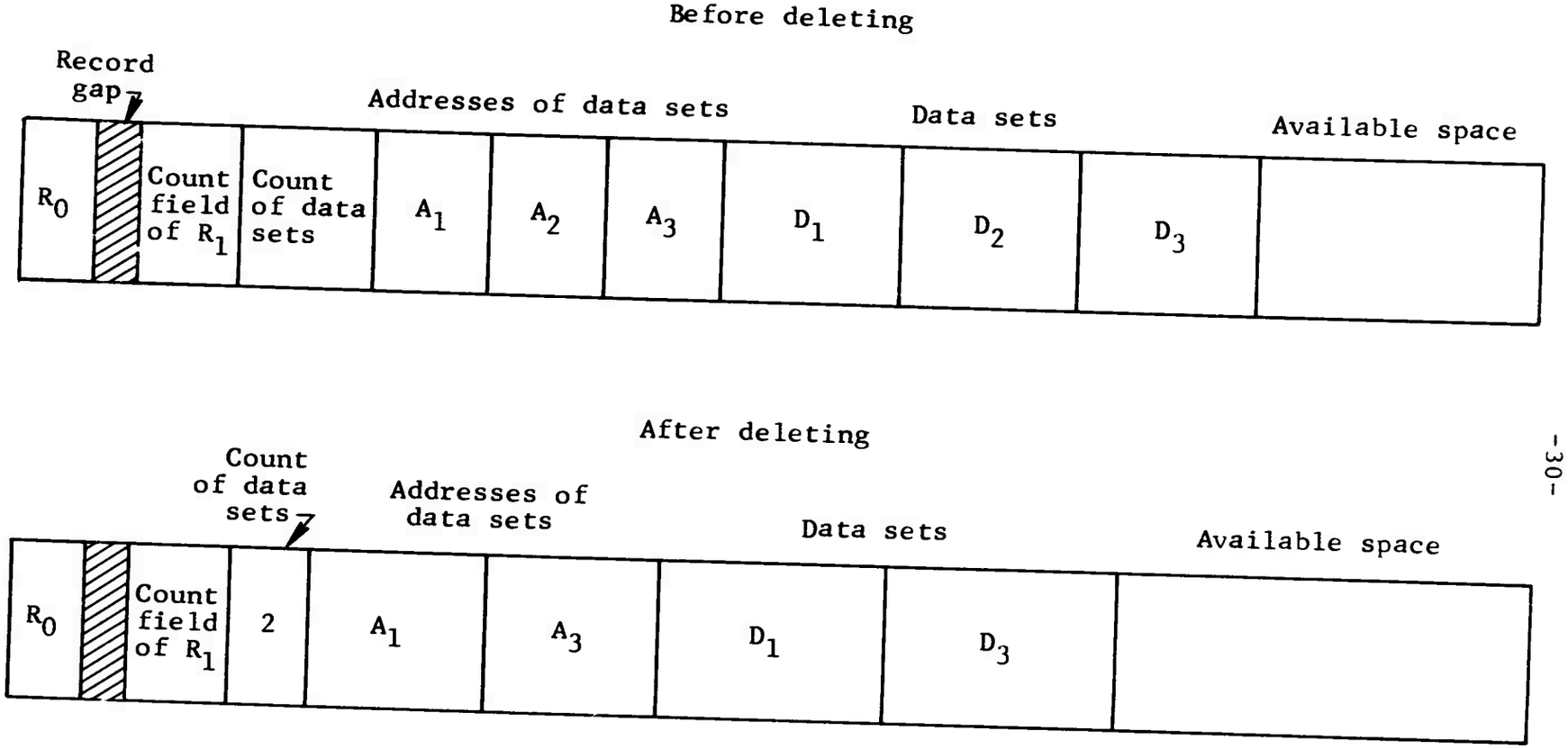


Fig. 16--Secondary Storage Adjustment for Deleting a Data Set

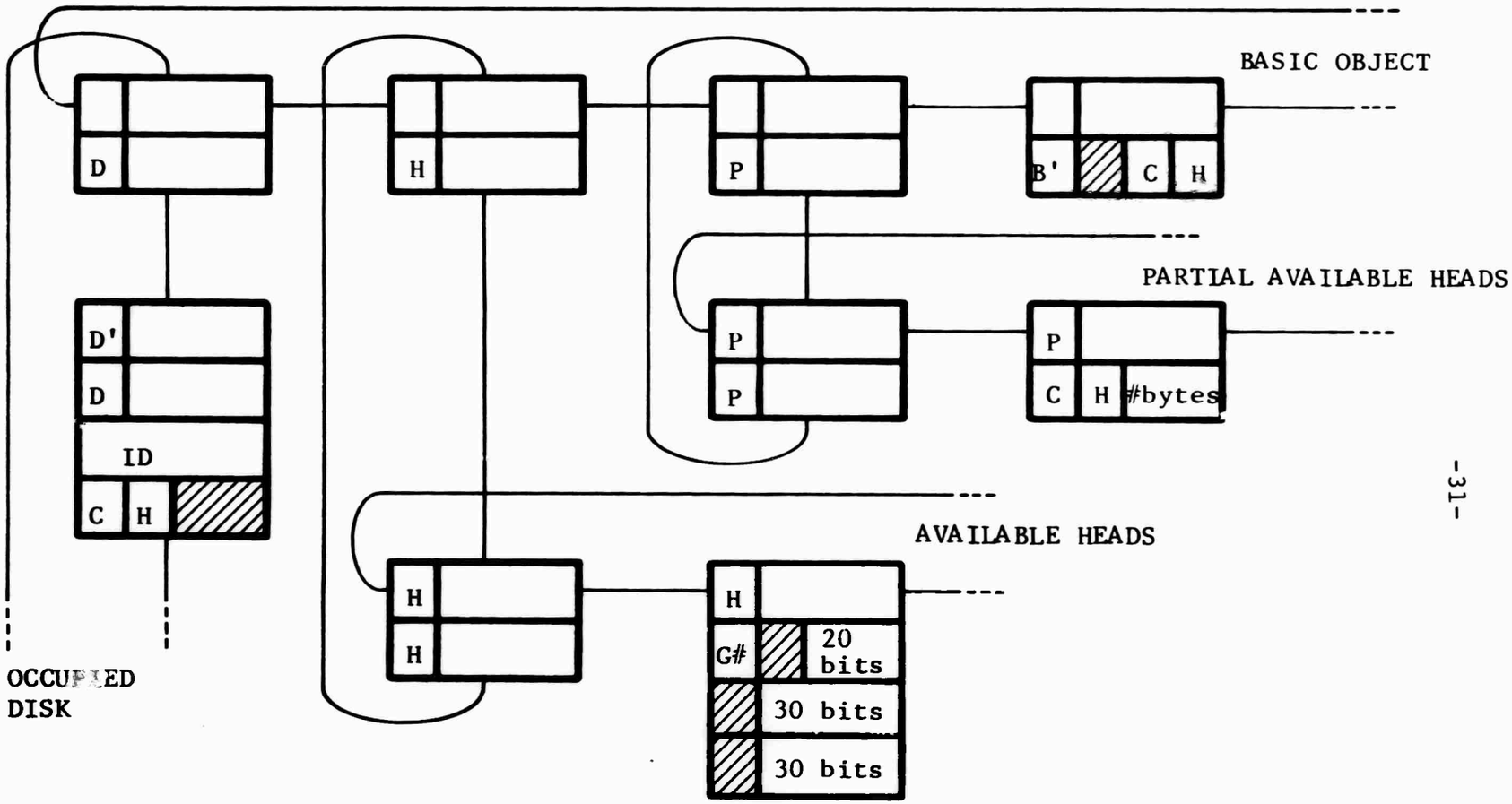


Fig. 17--Disk Allocation Complex in Either System Structure, Files Description Structure, or a File Structure

Heads (AH) structure that this head is completely available for future use. The double logical adjustment is, of course, made whenever a partially filled head contains only data that is no longer needed. First, because it is no longer a partially available head, it is removed as such from the *Partially Available Heads* structure. Second, because the head is now completely available, this fact is noted in the *Available Heads* structure.

Writing necessarily involves not only the allocation of space within, for example, a user's file structure, but also a way of indicating in the PAH and AH structures that this space has been pre-empted.

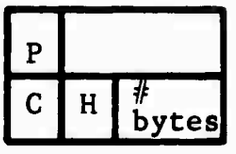
If the space needed is less than a full head, it is allocated, if possible, to a partially filled head, otherwise to a fully available head. Without constructing a complex and lengthy flow chart, one can describe the full-head allocation in only general terms. Two constraints are uppermost: to conserve storage space and, by keeping related items of data as closely together as possible, to conserve transmission time. If the space required is equal to or greater than a complete head, a new head is filled and whatever remains, if less than a complete head, is allocated to a partially filled head, otherwise to an additionally completely available head. This process is repeated so that eventually all space needed is allocated in the interests of the two constraints mentioned above.

The allocation of space is accompanied by an inventory that keeps a running account of all space changes and records them in the AH and PAH structures.

The full-head allocation algorithm operates from right to left and from top to bottom within the element; i.e., relative C,H = 1,9 is the first head assigned, and relative C,H = 5,0 is the last. The G number, the cylinder group index, indicates to which absolute cylinder numbers the elements' head bits refer, as tabulated below (Fig. 18).

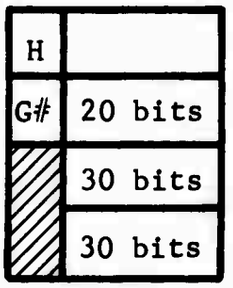
Partial available heads:

The element describes the relative cylinder, head and the # bytes unused at the end of the head.

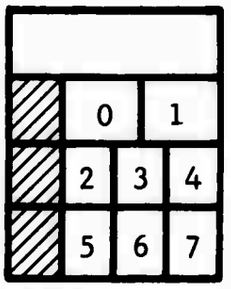


Available heads:

Each element describes the availability or non-availability of 8 consecutive cylinders. For the 80 bits, 1 = available, 0 = occupied or partially available.



Relative cylinder assignments within the 8 cylinders:



Head assignment within a cylinder:

Fig. 18--Element Composition

G No.	Absolute Cylinders
0	0-7
1	8-15
.	.
.	.
.	.
24	192-199 .

Writing new data into secondary requires building a write channel program, which consists of constructing a CCW for each new segment of data recorded in a head. The new data segments in primary are separated from each other, but the data within each segment are contiguous. When written into secondary, a segment's internal continuity need not be preserved. The overriding concern is simply to fill the head. The amount of any data set that goes into a head is limited by the available space in the head itself (measured in bytes). When the head has been filled, any remaining portion of a data set is recorded in another head with available space. A completely available head is always used if the data to be inserted will completely fill it. In this instance, the system provides a *Record Zero* and the *Count Field* of *Record One*.

Anything less than a full head of data, i.e., the last segment of a data set, can be written on a head already containing other data sets. When this occurs, the insertion of a new address for the new data segment requires a change in the old addresses. Because the new address is positioned immediately after the last of the old addresses, the latter are changed by adding to each of them the length of the new. The old *Record Zero* and *Count Field* of *Record One* are left unchanged. As related elsewhere, writing new data first involves reading the old, which is subsequently destroyed, not when it is read out, but only as the new data are written.

READING

An underlying consideration in reading is that the data set may be read into a larger space than it originally occupied so that additions to it may be made. The data sets read from secondary to primary are the same as those enumerated under *Writing*.

When a ring structure is read into a larger space than it previously occupied in primary, some data must be re-addressed because the relative positions of items in the data set change with respect to each other. With a change in position comes a change in address. Accordingly, the new address, signified by the pointer, must be recorded.

The display frame is reconstructed exactly as it appeared when it was written. Specifically, the reconstruction entails that:

1) The PCCW head is pointed to the absolute location of the first CCW in the CCW table and to the TIC within the CCW table.
2) The CCWs are updated so as to address the absolute location of the display data (Fig. 19).

Reading a read-only process group involves synchronization with other processes manipulating the same data once they are brought into primary.

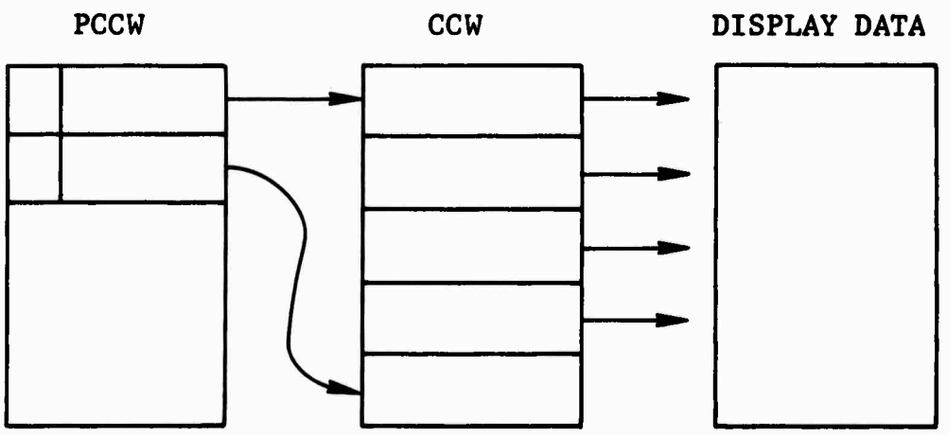


Fig. 19--Display Frame

BIBLIOGRAPHY

Ellis, T. O., J. F. Heafner, and W. L. Sibley, *The GRAIL Project: An Experiment in Man-Machine Communications*, The RAND Corporation, RM-5999-ARPA, September 1969.

-----, *The GRAIL Language and Operations*, The RAND Corporation, RM-6001-ARPA, September 1969.

-----, *The GRAIL System Implementation*, The RAND Corporation, RM-6002-ARPA, September 1969.

-----, *The GRAIL System Ring Structures*, The RAND Corporation, RM-6241-ARPA, February 1970.

DOCUMENT CONTROL DATA

1. ORIGINATING ACTIVITY The Rand Corporation	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

THE GRAIL LOGICAL INPUT/OUTPUT PROCESSES

4. AUTHOR(S) (Last name, first name, initial)

Ellis, T. O., J. F. Heafner and W. L. Sibley

5. REPORT DATE	6a. TOTAL NO. OF PAGES	6b. NO. OF REFS.
May 1970	45	---

7. CONTRACT OR GRANT NO.	8. ORIGINATOR'S REPORT NO.
DAHC 15 67 C 0141	RM-6257-ARPA

9a. AVAILABILITY/LIMITATION NOTICES	9b. SPONSORING AGENCY
DDC-1	Advanced Research Projects Agency

10. ABSTRACT

This is the fifth in a series of memoranda on the operation and organization of the interactive, on-line GRAIL system study. This Memorandum describes the logical I/O processes that transmit data between primary and secondary storage. Such transfers involve "reading" from secondary into primary storage and "writing" from primary into secondary. Four types of data sets are considered: (1) ring structures--list structures linked unidirectionally that represent secondary storage space allocations and the logical descriptions of user-drawn processes; (2) display frames--the internal representation of the displayed information; (3) text planes--the internal representation of user-supplied character strings (text display frames); and (4) read-only process groups--collections of executable programs. Data transmission is synchronized with other system operations to optimize response time and exploit the parallel processing organization of the system software. The GRAIL data organization would be useful for any on-line relational data file, including those used for military command and control.

11. KEY WORDS

Information Processing
Computer Graphics
Computer Programming Languages
GRAIL (Graphical Input Language)