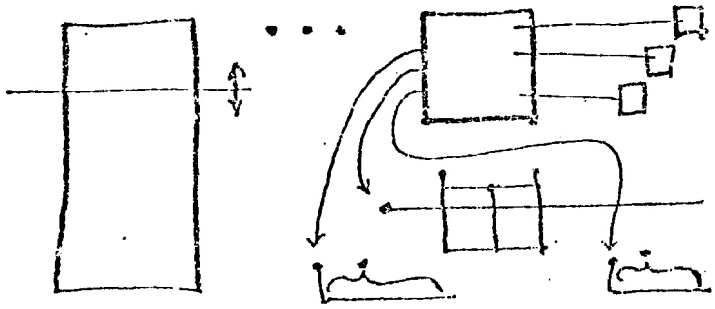
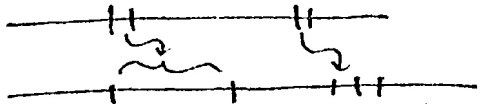


NOT FOR GENERAL DISTRIBUTION.

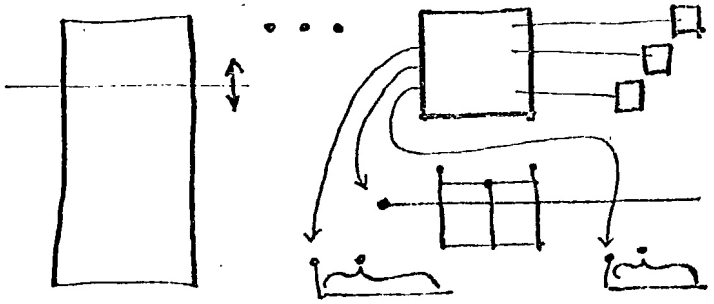
Xanadu is a new way of organizing programs, computers, displays and files. It is intended to permit the construction of extremely complex systems by simple aggregation. It either will or will not work.

You will probably understand Xanadu in general if you understand the following:

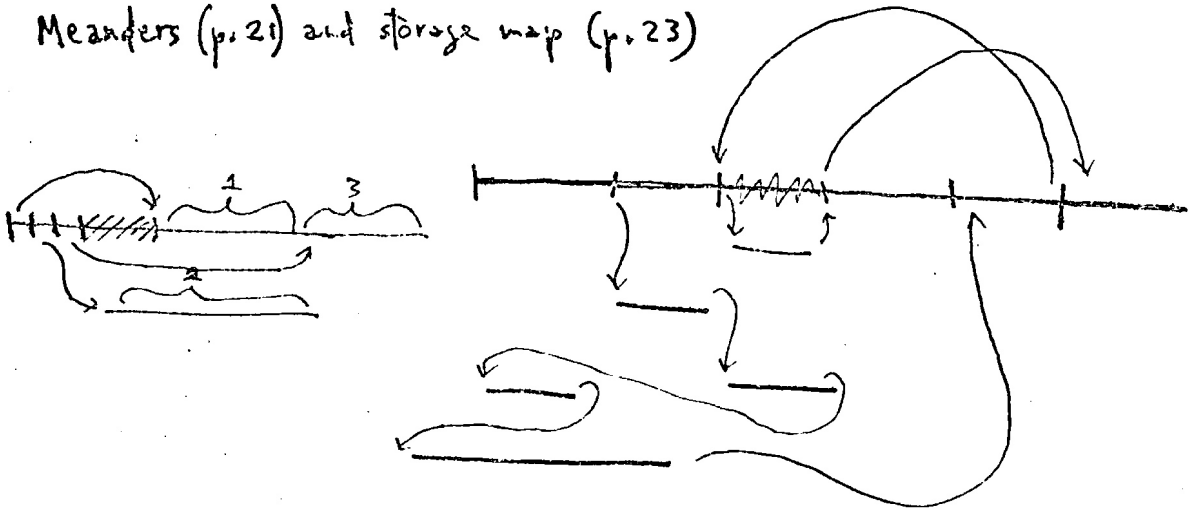
Parallel streams (p. 9)



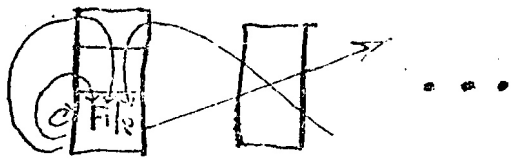
Beds (p. 12) and their pillows (p. 14)



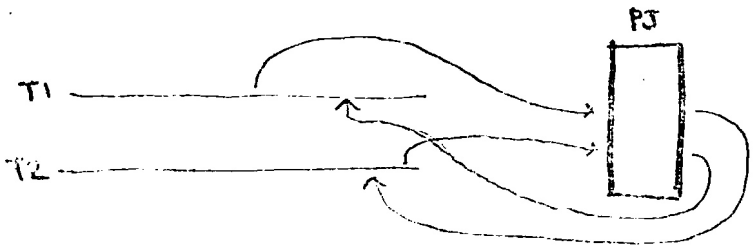
Meanders (p. 21) and storage map (p. 23)



Part map (p. 18) and counterpart map (p. 19)



Plugjumps (p. 33)



2

XTD
Limited Circulation Only

TECHNICAL DESCRIPTION OF THE XANADU SYSTEM
10 February 1971 [this edition completed 22 March]

This is a brief description of a proposed new general-purpose computer environment which is believed to be novel, peculiar and unusually powerful. The reader is expected to understand conventional computers, mini-computers, operating systems, subroutining displays and disk management. Xanadu's unusual features will not be justified, argued or compared with other approaches in this working document.

Xanadu is a unified system for the management of very complex files and displays changing erratically. Originally intended to be an opulent console for text and graphics, it has become generalized and simplified. In concept it is now fully general-purpose.

Implementation in hardware, firmware or software is possible; in some cases we would have to call the result a Xanadu simulator.

It is not particularly easy to describe. Xanadu could also be thought of as a through-designed system architecture, a revised input-output structure, a parallel virtual memory, a radical operating system, a large data break add-on package, or a weird virtual machine.

Implementations are presently contemplated for PDP-11, Nova, Imlac PDS-1, 1130 (with or without 2250), 620 IDIION.

Extensive proprietary rights are claimed to these processes, machines involving these concepts, software implementations, nomenclature, notation, etc., etc., etc.

TERMINOLOGY IS TENTATIVE.

NOT FOR GENERAL RELEASE.

This copy issued to: _____

Theodor H. Nelson, President
The Nelson Organization, Inc.
453 W. 20th St.
New York, N. Y. 10011

XANADU, the mysterious artistic pleasure dome, is intended as a general-purpose console for the creation, revision and viewing of animated and branching text and line drawings. In addition, it will have two automatic facilities nowhere else available: historical trails, permitting backtracking through creative work, and complex coupling among corresponding parts of evolving structures (multicoupler facility). Together these will permit the user to browse in time as well as space, and understand his materials, his choices and his creations much better.

Since I have been told these things "can't be done," I have had to work out the mechanisms for them, here described. Essentially it is one simple method which can, like fingers, be intertwined in complicated ways.

The fundamental mechanism we may call stream complex programming. It seems to be a new genre of computer programming, for which a new type of machine should eventually be built. Meanwhile it can be easily programmed for existing equipment. The particular system described here I tentatively call X-STREAM, Xanadu's stream system.

Xanadu is to be a general-purpose work console with historical and file-coupling features always available. X-Stream is the raw mechanism of it, for which the reader may see other uses.

This stream system consists of a disk system, which provides and edits the streams, and a bed system, which moves the streams through core (without moving strings) and keeps them correctly arranged in relation to one another.

Within the premises of this stream system, programs (streams themselves) can be written for any purpose. However, ordinary programs cannot conveniently function. In a software implementation programs under X-Stream would be wholly interpretive; in hardware not.

The overall Xanadu system comprehends: 1) the stream system (both disk and bed), and 2) particular programs for creating text and pictures, depicting and modifying them on the screen, and continuously refreshing a non-storing CRT display. In addition, 3) what may be called the stewardship functions (multicoupling and historical backtrack) are all times available in conjunction with the creation and revision programs.

A very simple swapping and relocation monitor allows program overlays and space adjustments in core. These may tend, however, to bomb the display briefly or make it flicker.

As it is assumed that the system will be tied by communication link to immense libraries, the operating system will provide for the capture and dismissal of a lot of things to and from remote storage, unendingly. Registry and interchange protocol will not be considered for the time being.

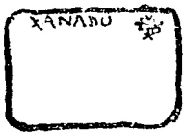
Tape storage we more or less ignore; it is a special local case of remote access.

GENERAL-PURPOSE NAÏVE FRONT END

(Xanadu for the beginner; text functions only)

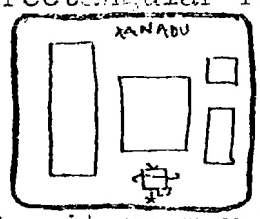
FORMATS AND FURNITURE

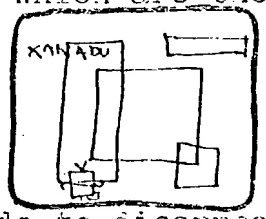
Some stuff appears on the screen automatically: the word XANADU and an "edit rose," a control diagram for the edit functions. These are standard furniture.



(possible edit rose:)

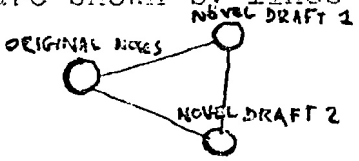
The user may create a number of screen formats, consisting of any arrangement of the standard furniture and the user's own windows-- rectangular frames which are the user's own furniture.

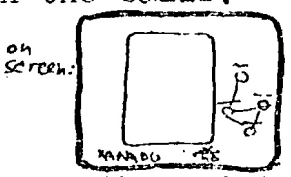




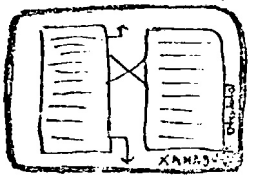
All furniture may be made to disappear, leaving a small symbol to point to if you want it to come back.

A number of files may be associated with the formats. Each file is shown as a little labelled circle or ball. Coupled files are shown by lines between the balls.





To call a file to a frame with the pointer, point at the file ball followed by the desired frame. To show two coupled files, assign two files to two frames, then point at the line connecting two file balls. We then see the multiple couplings between the two files, displayed as lines.



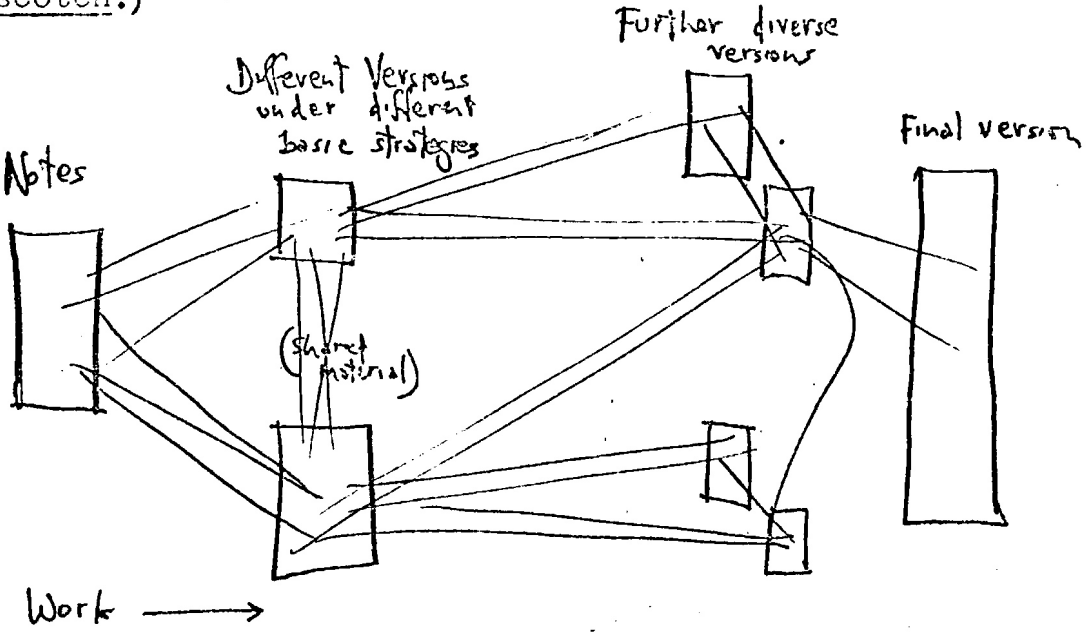
Using pointer controls, it is possible to roll either file smoothly forward or backward within its frame, with links appearing and disappearing appropriately. (The file which is under direct user control is the independent file; the file which moves derivatively is the dependent file.)

Editing is commanded by pointing at parts of the edit rose, then typing the changes and pointing to where they are to take place

Changes may be undone by stepping backward in time (user control means undecided).

EXAMPLE OF USE: NOVELIST'S CONSOLE

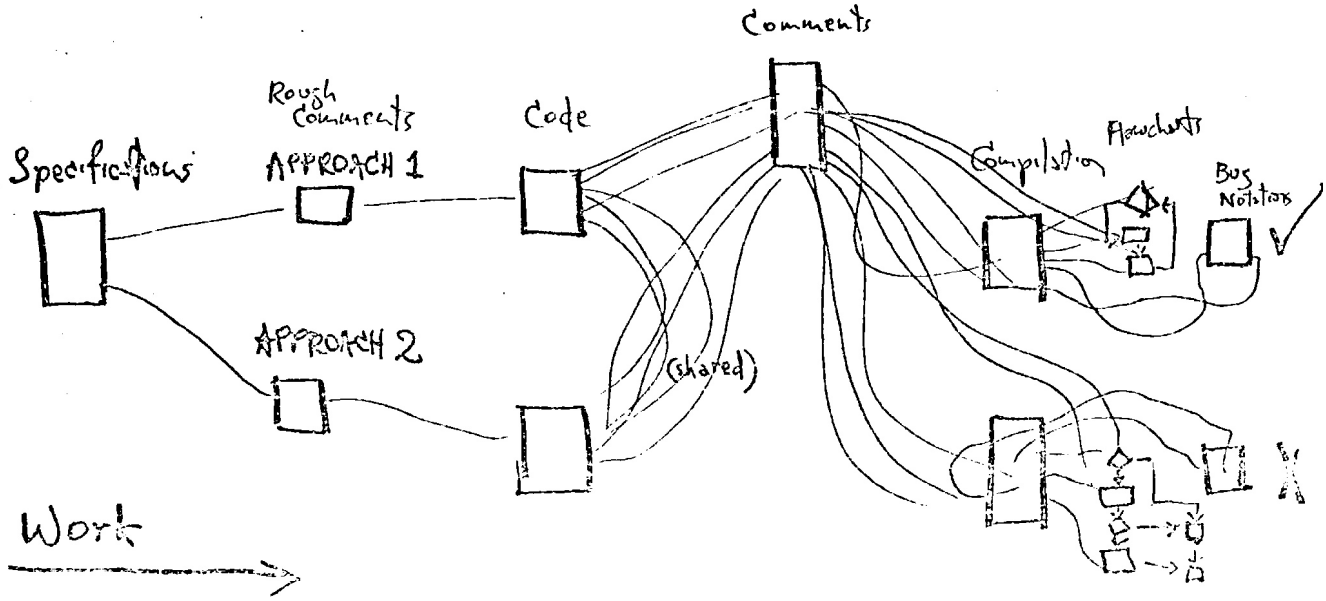
Beginning with raw notes, the novelist may rework his material into alternate and linked drafts to see how they work out, until he is satisfied with a final version. (Or, he may leave it in a linked hypertext version, such as Pale Fire or Hopscotch.)



EXAMPLE OF USE: PROGRAMMER'S CONSOLE

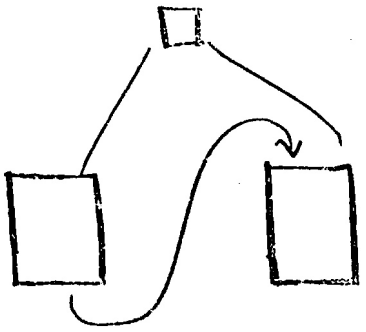
The Xanadu system may serve as a working console and time-sharing terminal for the programmer. Proceeding from written program specifications, he may produce increasingly more specific code, work out the consequences of alternative methods, flowchart, code, convert, compile, run, mark possible bugs and so on.

The system will take care of all inter-indexing among these parts, and store all versions and intercouplings down to the finished program.

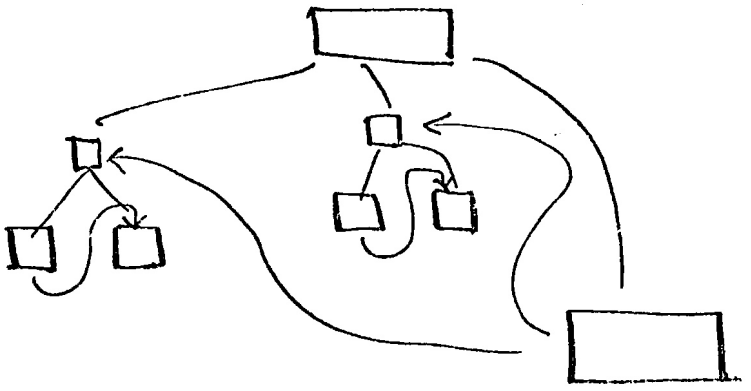


THEORY OF CONVENTIONAL COMPUTING

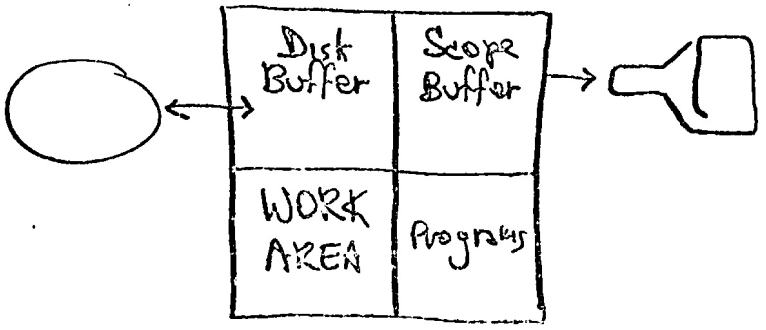
Conventionally, data is stored in chained blocks and indexed by index blocks.



This gets hairy when more than one data type is being employed, as well as change catchers and other accessory information.



Core memory is used in all sorts of tangled and overlaid ways. However, we may distinguish four types of area:



Every change in the display requires the rewriting of the display list in the scope buffer. Indexing is horrific. Programs are big.

XANADU THEORY

Xanadu Theory is an idealized system of conventions for handling data as streams and only as streams, rolling these streams through memory and jumping between them.

PECULIARITIES AND UNIQUE ADVANTAGES OF SYSTEM

This system is so unusual that its special qualities may not be obvious, and conversely, it might be thought to possess advantages which are not actually unique. I think the following are among its strong points and interesting features:

1. NO BUFFERS

The general idea, the most general idea, is that we do away with I/O buffers, work areas and display lists. Everything happens in the same places. I/O and display buffering is out of the same canonical storage areas. Within this constraint, marshalling programs somehow set up and revise the display and permit editing and animation.

2. VIRTUAL MEMORY

Complex virtual memory and indexing are easily specifiable by the programmer. Overhead is low and very straightforward. Becau streams are rolled automatically to and from disk, the programmer need never be concerned about overflow, except for optimization.

3. COMPLICATIONS

Complexities would seem to be additive, rather than multiplicative, for each elaboration.

4. ROLLING AROUND

Data may be rolled smoothly forward and back ad infinitum on a continuously refreshed display.

5. MULTIDIMENSIONALITY

The rolling feature adapts straightforwardly to multidimensional arrays. Movable n-dimensional views of indefinitely large m-dimensional checkerboard-type arrays are possible.

6. CONSTANCY OF CORE ADDRESS

Because a string is never moved in core, its absolute core address remains valid as long as it is in. This is very, very good

STREAMS

All information is in streams rather than blocks. The streams flow smoothly forward and backward through core, and refer to each other a lot.

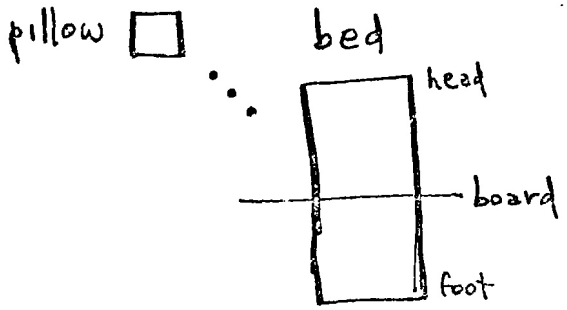
MORE WATER IMAGERY

The area in core through which a stream moves is called a bed. The movement of the stream through its bed is called babbling. The stream is made up of meanders, sections separately stored. The part of a stream currently in core is a view.

BED IMAGERY

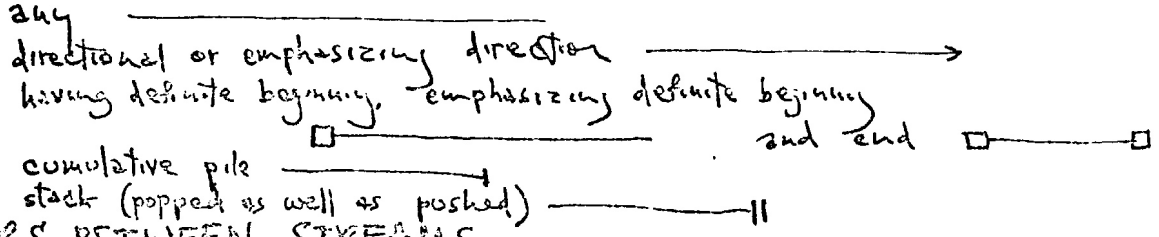
What we do to these streams in their beds is unrelated to their past. Thus we mix metaphors and deal with beds per se.

Each bed has a central area. This is called the pillow. The first space in a bed is the head, the last space is the foot. A curious movable partition divides the bed; this we may call the bundling-board or board. [Also called "terminator" sporadically in this document, and "divider" in certain other documents.]

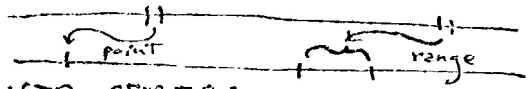


NOTATION:

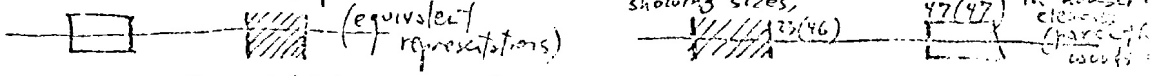
STREAMS



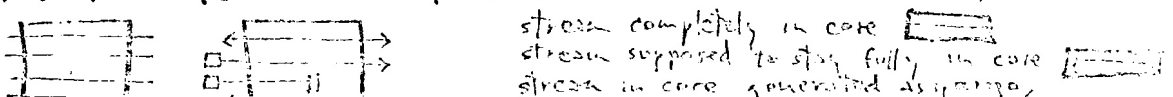
POINTERS BETWEEN STREAMS



WINDOWS INTO STREAM

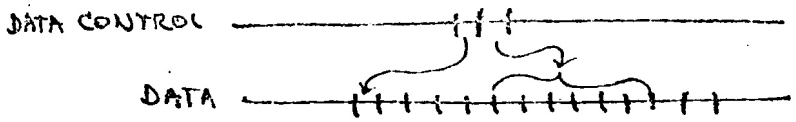


STREAMS PASSING THROUGH CORE



PRINCIPLE OF PARALLEL DATA CONTROL

For various reasons it is nice to have different forms of data kept pure, without control codes (e.g., text with paragraph codes omitted). The stream approach allows you to declare parallel streams for any purpose.



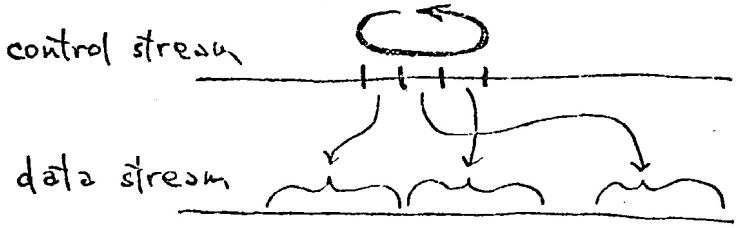
Indexing is in general between two streams. Whenever the programmer needs to distinguish data types, he may break them out into separate streams.

The only space overhead for a new stream, besides the core it occupies, is its pillow (~ 32 words).

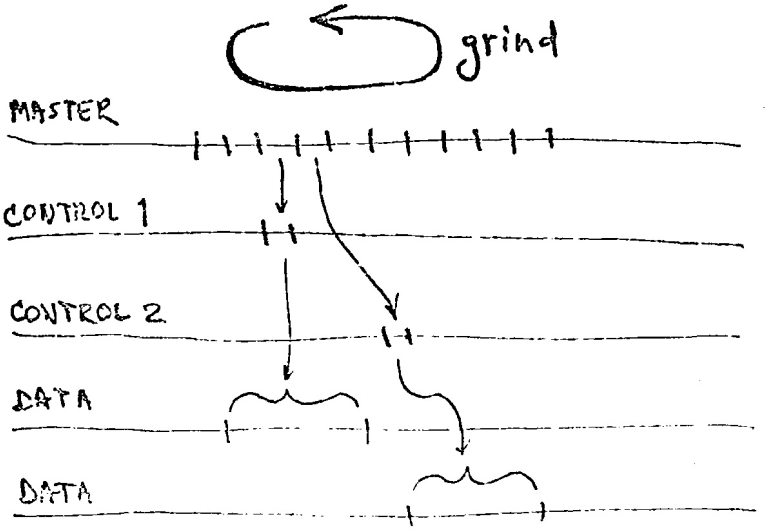
Intercommunication between streams is by stream position, either absolute or relative to current position.

PRINCIPLE OF DISPLAY STREAM-JUMPING

The display control cycles through a sequence of stream pointers, looking first at a control stream to ascertain what elements to display-- where they are and how many-- and then dipping down to take that number of elements from the specified stream. This is the display grind.



The principle can be extended to hierarchies containing any number of control and data streams.



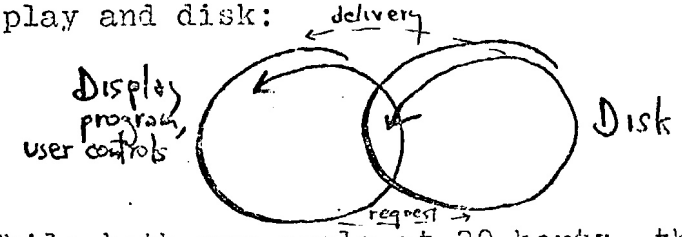
Etc.

DOUBLE GRIND OF DISPLAY AND DISK

For appropriate animation and responsiveness in the system, it must use a refreshed display of at least $(2k)^2$ resolution. This display must always be refreshed without flickering, no matter what else is happening.

Most of the other work is done by the disk system.

Thus we may see the general setup as a double grind of display and disk:

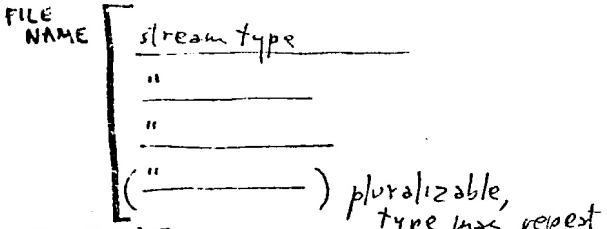


While both may cycle at 30 hertz, they are asynchronous.

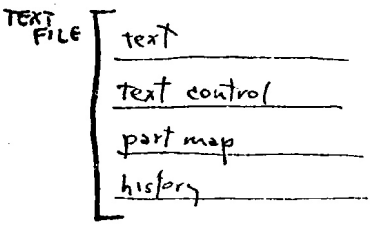
FILES

A file is n parallel streams of different types. The number and types of streams depends on the type of file.

FILE NOTATION



SAMPLE FILE TYPE



FILE SEQUENCE

By "file sequence" we shall mean: first stream in sequence, followed by second stream in sequence, and so on to the end of the last stream.

INTERFILE FILES

Some streams are outside and between files. These we may call dependent files, dependent on other files.

Important dependent files are:

The counterpart map, which points to corresponding and related parts of two or more files.

Interfile history files, which relate the happenings between files over time.

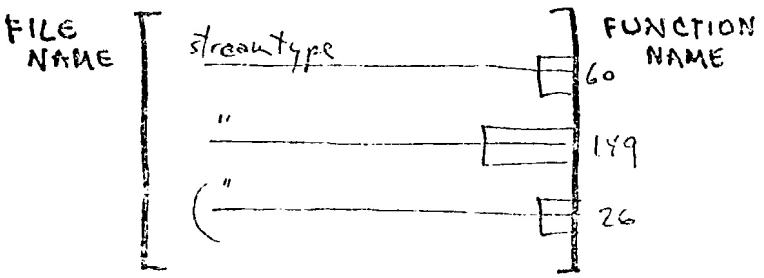
FUNCTIONS

Functions are particular programs, requiring a certain number of streams in a certain number of specific types of beds. A file may bind to a function, meaning that its streams suit the requirements of the function, or not.

FILES AND FUNCTIONS

Files conformable to a given function are babbled in that function's beds.

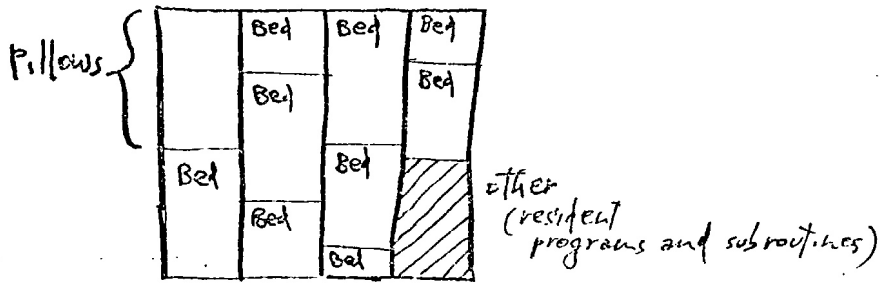
Notation (with sizes)



BED SYSTEM

The present view of a stream resides in a bed, an arbitrary block in core.

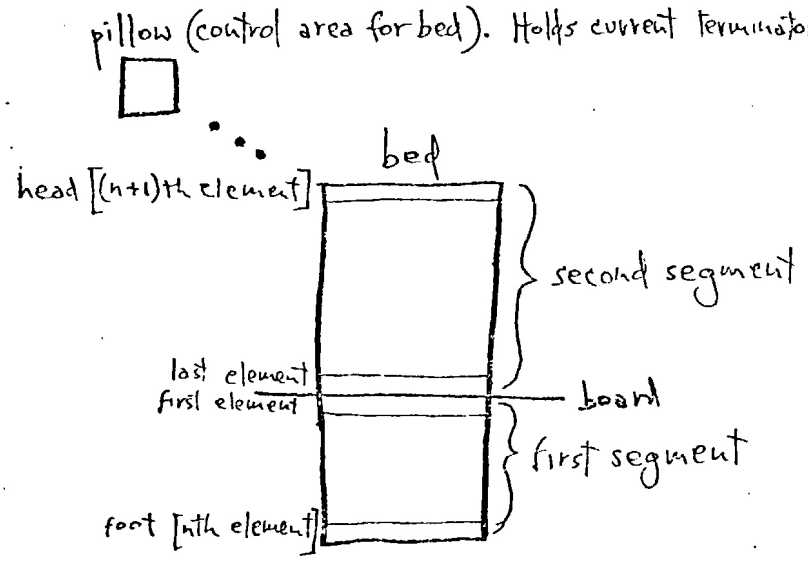
Beds are the main contents of core. Pillows are their resident control blocks. A pillow need not be contiguous to its bed.



When the activity changes, we may change the beds.

A bed always (almost always) holds the same amount of information, which is never (almost never) moved.

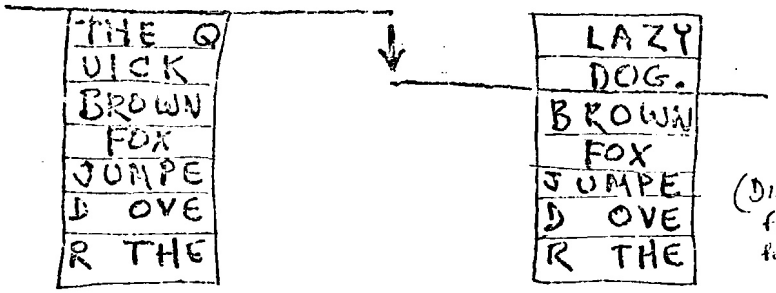
It is divided by an imaginary line, the board, into a first and second segment, which are in reverse order.



The board is actually a count specifying simultaneously the starting-point of the view of data and its termination. This is because the view begins and ends at the same point in core, which is changeable.

Ideally the board should be the data break address register used for that bed, but in most cases this will have to be simulated.

To slide the view forward in the stream, or babble forward, we overlay new information a binary word at a time in a windowshade fashion. The board (a stored count) changes as we go.

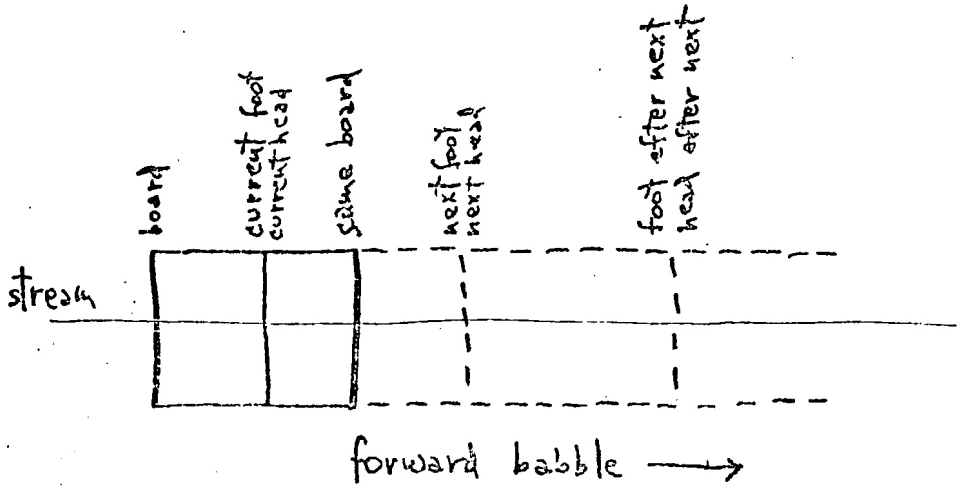


(Diagram is packed five letters across for compactness.)

Ideally this readin is directly to the bed from external memory.

Back-babbling is the same thing in the other direction. (We ignore here the complication of disk rotational direction.)

This movement may be visualized as a view bounded by boards travelling along the stream. Certain elements accidentally fill the bed-end positions as the window moves.



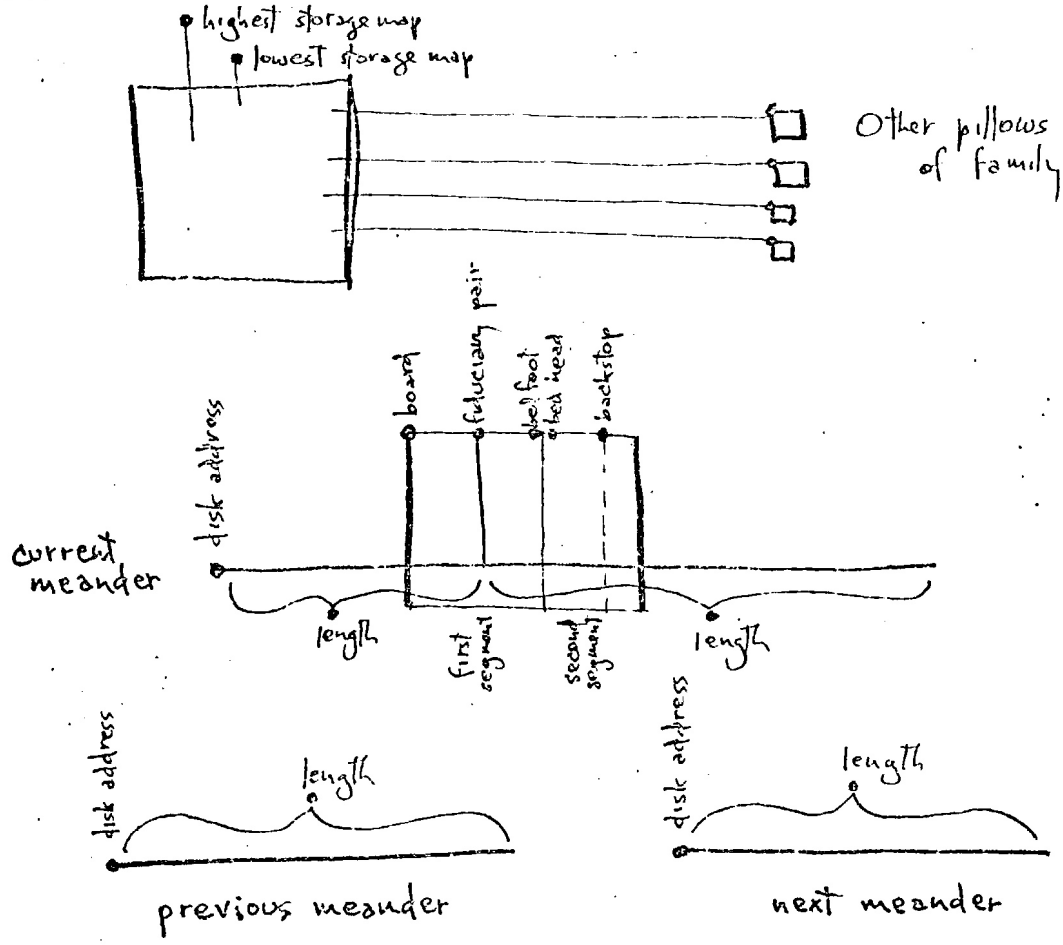
WRAP

It will be noted that if you step through core, the second segment wraps automatically to the first, since you pay no attention to the board, which isn't there anyway.

THE PILLOW

The pillow is a control block fixed in core for each stream in core. The pillow contains information sufficient to find a stream location in the window, and convert all inter-stream references to absolute bed addresses.

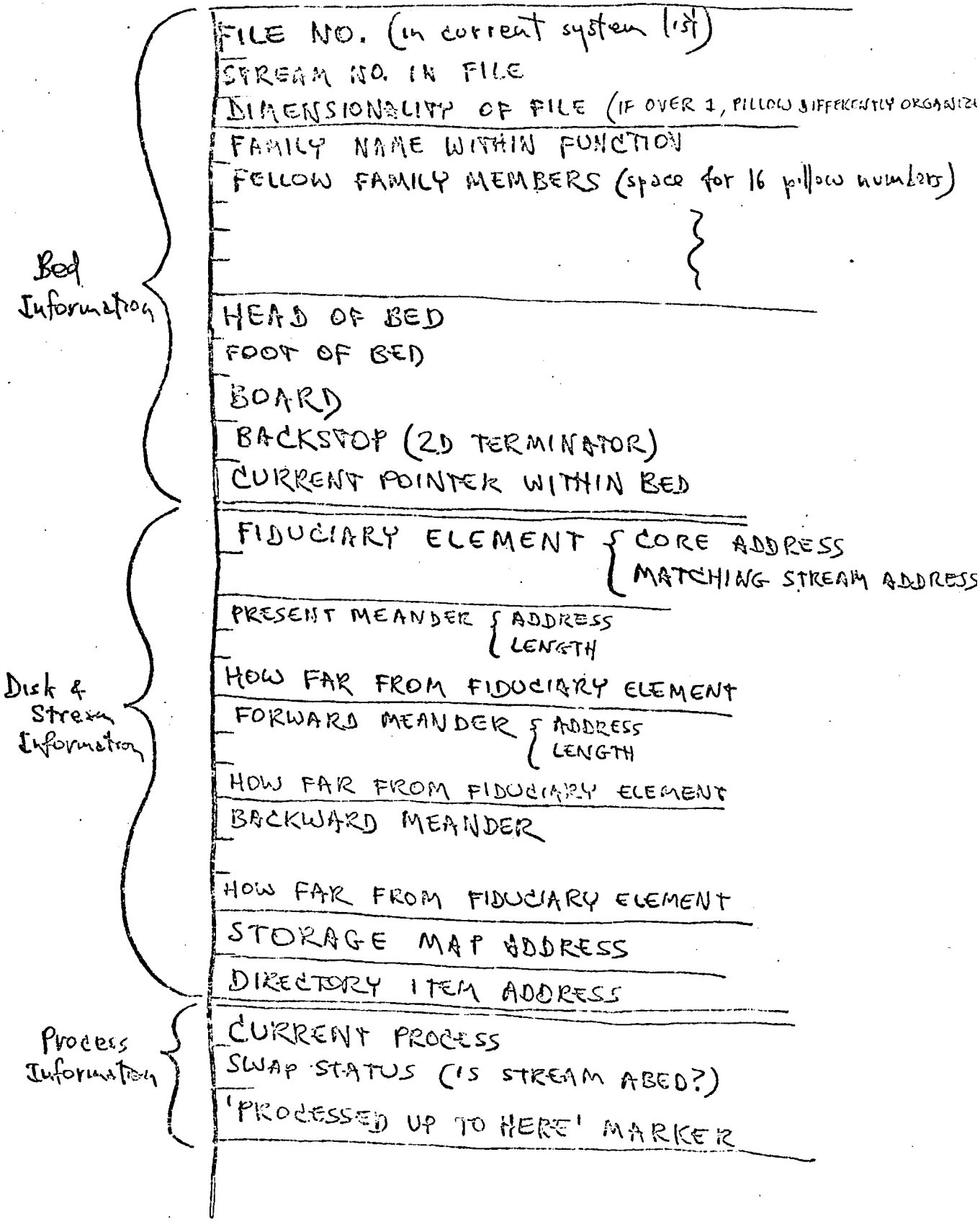
We may visualize this as a system of addresses relating to the stream. Consider each black dot as an address.



The pillow holds for ready reference the disk addresses needed for continued stream motion in either direction, as well as stream jumps to the next meander in either direction. Thus babbling may occur directly for a ways either forward or backward without bothering the monitor.

In other words, the "place" of the core view on the disk is kept by the pillow-- the pillow keeps track (sic) of the disk address. You can therefore babble forward or backward to the end of the next meander in one or two fetches without going to directories first.

THE PILLOW


Bed Information
Disk & Stream Information
Process Information
FILE NO. (in current system list)
STREAM NO. IN FILE
DIMENSIONALITY OF FILE (IF OVER 1, PILLOW DIFFERENTLY ORGANIZED)
FAMILY NAME WITHIN FUNCTION
FELLOW FAMILY MEMBERS (space for 16 pillow numbers)
HEAD OF BED
FOOT OF BED
BOARD
BACKSTOP (2D TERMINATOR)
CURRENT POINTER WITHIN BED
FIDUCIARY ELEMENT
CORE ADDRESS
MATCHING STREAM ADDRESS
PRESENT MEANDER
ADDRESS
LENGTH
HOW FAR FROM FIDUCIARY ELEMENT
FORWARD MEANDER
ADDRESS
LENGTH
HOW FAR FROM FIDUCIARY ELEMENT
BACKWARD MEANDER
HOW FAR FROM FIDUCIARY ELEMENT
STORAGE MAP ADDRESS
DIRECTORY ITEM ADDRESS
CURRENT PROCESS
SWAP STATUS (IS STREAM ABED?)
'PROCESSED UP TO HERE' MARKER

BED FAMILIES

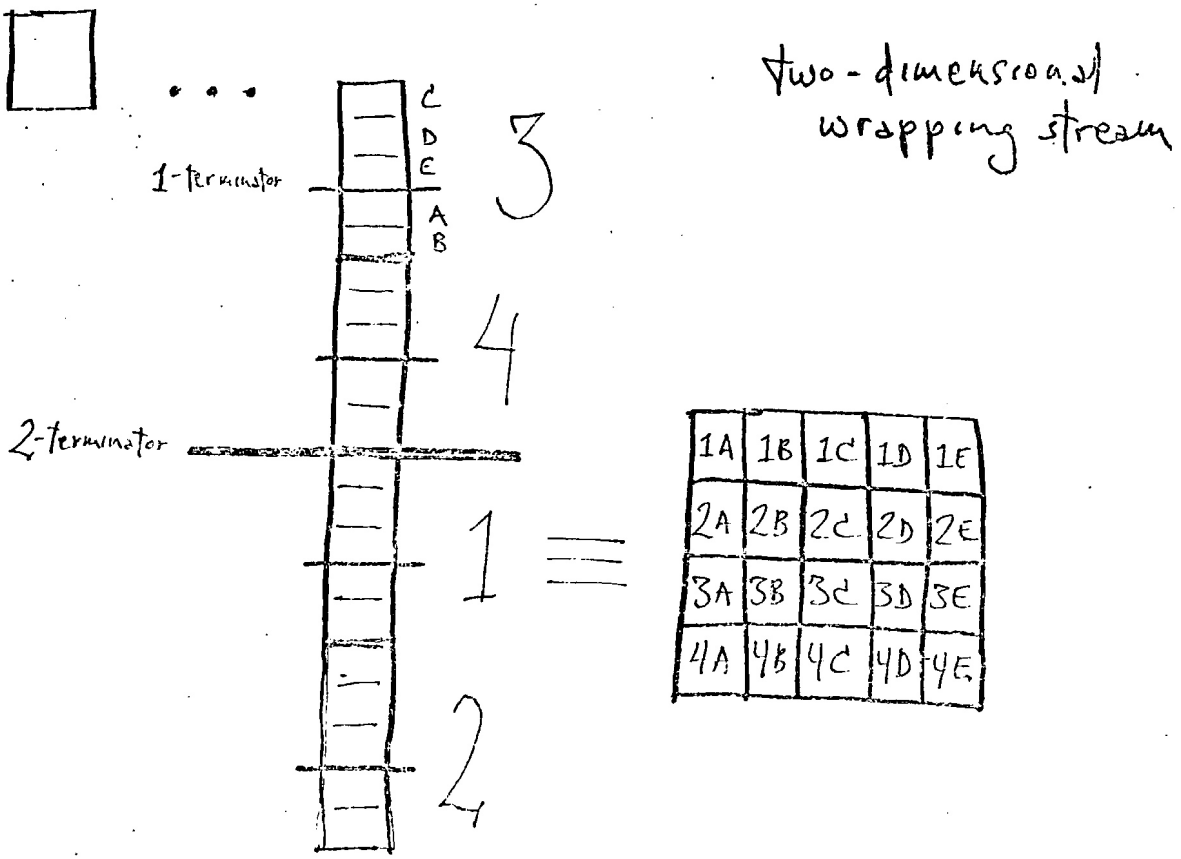
Within a function, several beds hold the component streams of the file. These are the bed "family." Its members are identified among themselves by a short binary number (say, <32), meaning within the family only.

MULTIDIMENSIONALITY

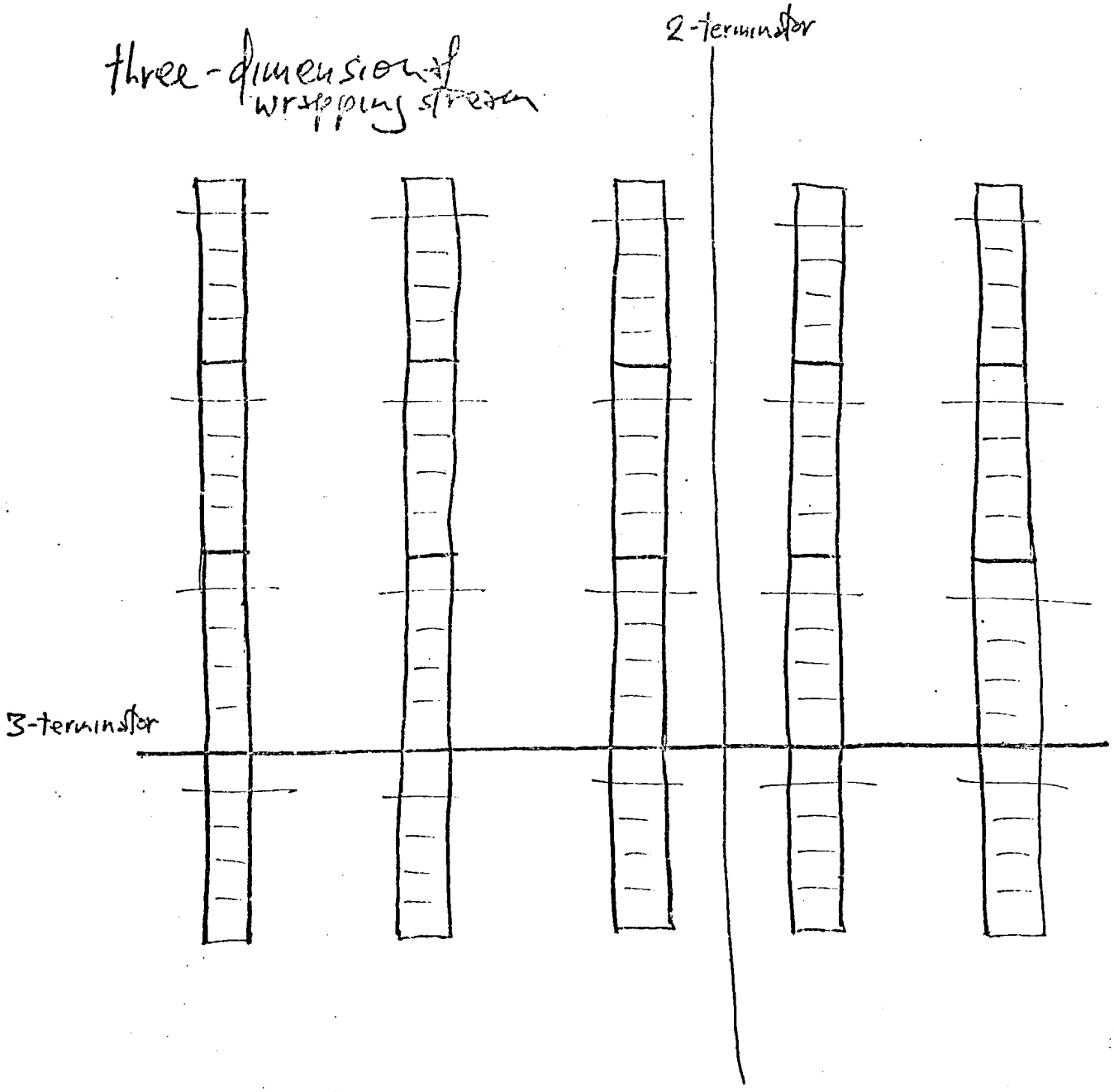
The stream system expands readily to the representation of two-dimensional and higher-dimensional structures. It can provide multidimensional views of these structures in core. These views are incrementally movable through the structures with constant core addressing.

New boards, which we may call n-boards, specify the board positions in the separate dimensions: the 1-board in dimension 1, ... the n-board in dimension n.

The overall view is incrementally babbled by rows (or columns, or whatever) in only one dimension. To babble in another dimension, combinations of beds must be wholly rebabbled.



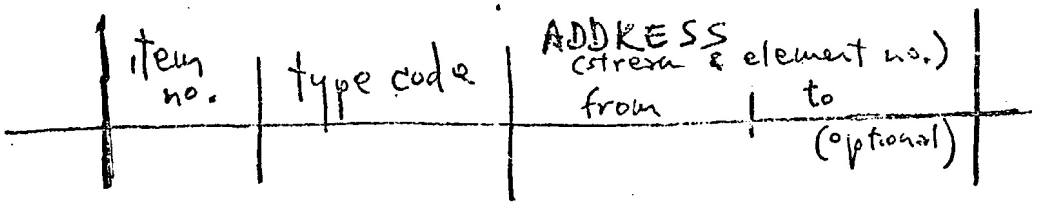
Each bed requires its own pillow. (The variant pillow structure for several dimensions will not be considered here.)



PART MAP

The part map tells where things are in the file, by stream position.

Ordinarily it is used for coupling parts of the file to the outside. It may, however, also be used for internal reference if need be.



Types so far:

point
section
character
heredity subsection
(i.e., a subsection created by dividing a section).

¶ The part numbers are irreversibly assigned in eternally ascending sequence.

ITEM SEQUENCE (of Part Map)

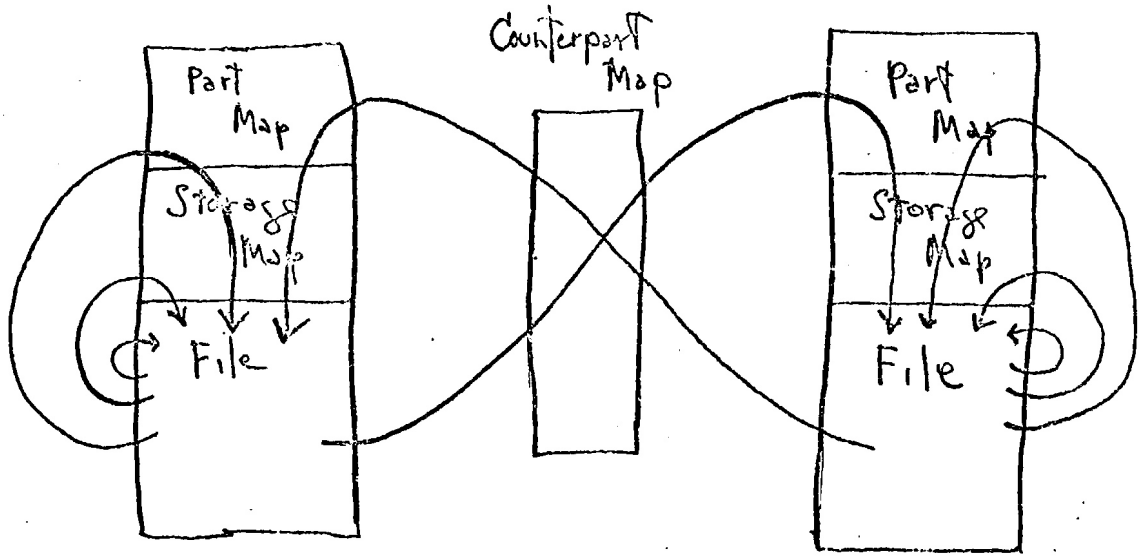
This eternally ascending sequence of item numbers we shall call Item Sequence.

COUNTERPART MAP

The Counterpart Map is a stream specifying correspondences between files, by number. Dispensations must match those of the files

STEPS OF FILE AND INTERFILE COMMUNICATION

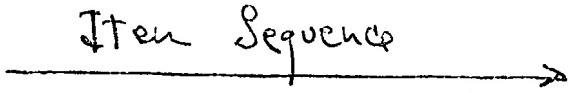
Several steps are involved in file and interfile communication. The separation of these steps is intended to simplify the structure, at some cost in overhead.

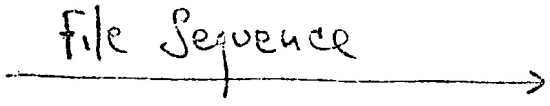


DUAL PART MAPS

In small files it is perfectly reasonable to store part maps in item sequence and natter or eternal-roll through them to update changed stream numbers.

A more elegant method, suitable to long files, is to have two part maps, one in Item Sequence and one in File Sequence.





For update: update items in File Sequence Part Map; make catcher stream-list of affected items; modify affected items in Item Sequence Part Map.

DISK SYSTEM

Most of the work in the stream system is done by the disk system. The general notion of the disk system is to store streams in pieces but deliver them integrally when they are needed. List structures and procedures are here suggested for piece storage and free space. The precise list structures are probably not very important, but the realizations described here will clarify the general mode of operation desirable.

THE DISK MANAGEMENT PROBLEM

The basic disk problem is to pull your desired stream sections off disk as quickly as possibly, preferably in one revolution. Considered more generally, you want to minimize the number of accesses and (more importantly) the number of turns. You also want to avoid chaining techniques which require recursive or deferred accessing. Whenever possible you want to do more than one thing on a given access. (If a moving-arm disk is employed, at great penalty to the system, the motion of the arm is to be avoided whenever possible, and its swing distance to be minimized if it does move.)

GENERAL DESIGN OF DISK SYSTEM

For various reasons, the disk system to be described here has been designed to do quite a lot. Besides fetching pieces of streams, with (it is to be hoped) some optimization, the disk system handles stream motion, stream edit, indexing and storage control.

1. "Stream edit" means it adds sections (or meanders) at desired points in the stream, deletes sections, substitutes sections, reconfigures their storage on an ongoing basis, and keeps track of it all.

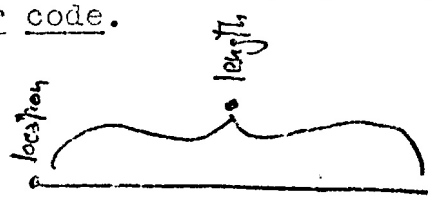
2. "Indexing": the disk system keeps track of the meanders that make up streams. It also maintains a directory. This means maintaining a shifting collection of files, functions and streams on disk.

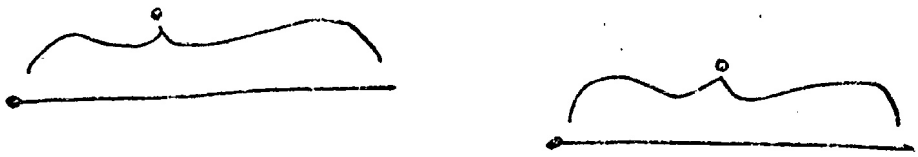
3. "Storage control" means a variety of things. The disk system handles garbage collection, maintaining a free-space list and assimilating dropped meanders to it. Messily-stored streams are rearranged for the better. It also manages certain rearrangements, e.g., sliding meander coles apart for insertion, and sorting items under stream control.

MEANDERS

The general method of stream storage is this: a stream is stored consecutively at first, and as changes occur any additional pieces are scattered around the disk as convenient. Such a piece is stored in a place called a meander.

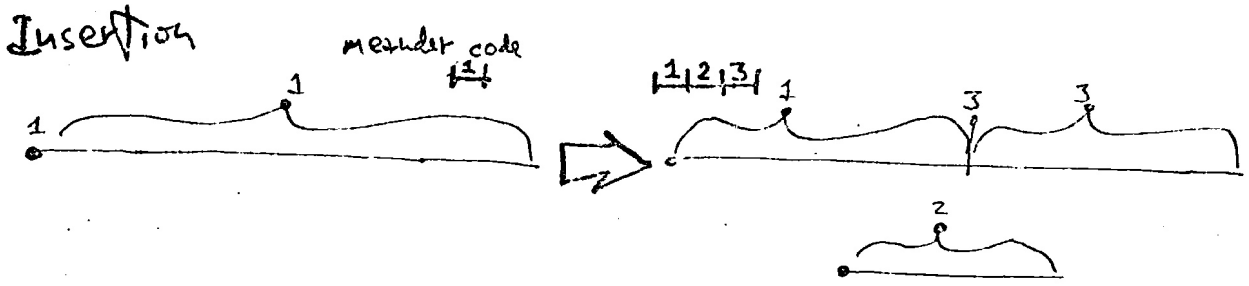
The pointer to a meander on disk, specifying its location and length, is a meander code.

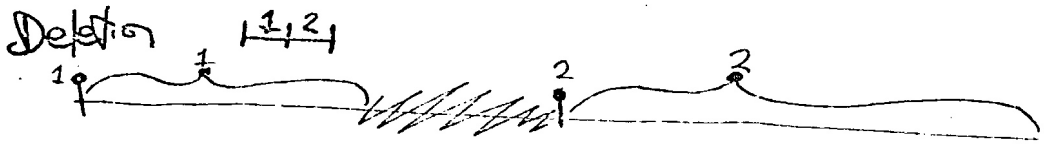


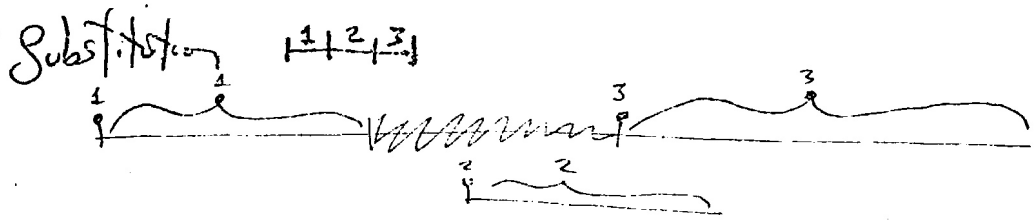


STREAM CHANGE

By changing the meander codes we readily accomplish editing of the virtual stream as represented on disk.



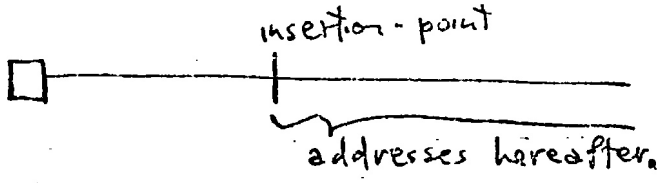




¶ The disk system given here automatically handles stream edit; however, this is to user-level editing; as a loose hair blade is to a knife, and to be treated with similar apprehension.

TELLING THE OTHER STREAMS OF THE CHANGE

When a stream is changed, either by deletion or insertion, all addresses after it in the stream, and in streams pointing at it, including the Part Map, have to be lengthened or shortened by its length.



This means one or more pass through every stream in the file that may be pointing at it. (Since streams outside the file communicate through the Part Map, no streams outside the file need be examined.)

BUFFERED UPDATE

(We assume that changes are completely updated concurrently, one at a time, but plainly buffering schemes are possible and reasonable.)

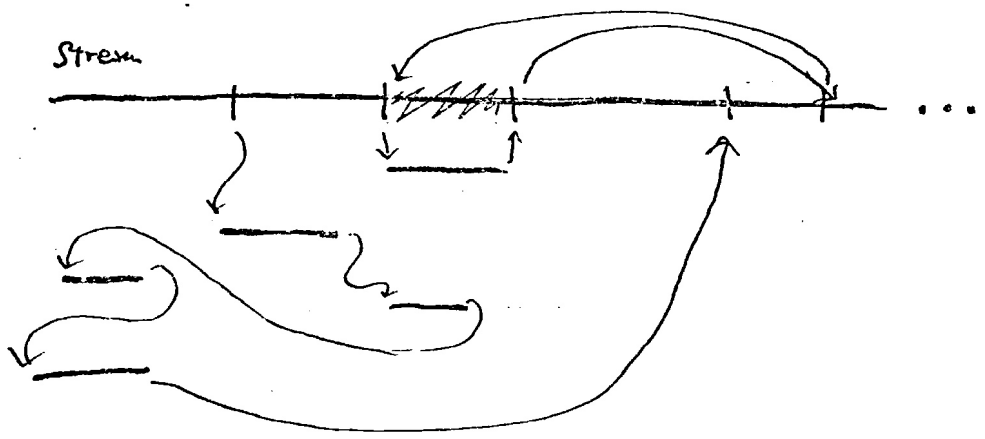
NO BACKPOINTING

As described here, every meander is in only one stream, so that backpointing is not required. When a meander is to be copied to another stream, it must be edit-locked with respect to other changes until the copying operation is finished and all necessary histories and counterpart maps are informed.

STORAGE MAP

The list of meander codes specifying a stream's storage is the storage map.

The intention is to edit streams and revise their storage with as little fuss as possible, regardless of developments.



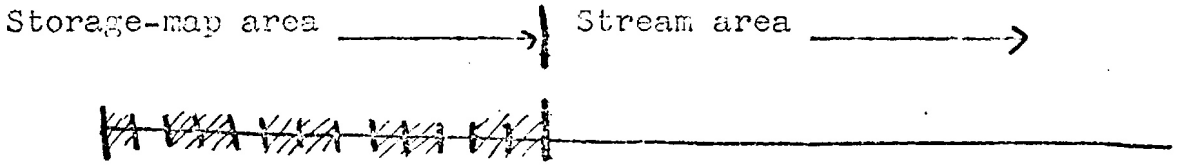
The storage map is not transmitted between storage media or remembered in histories. It is begun fresh whenever a file enters the system and is assigned space.

QUICK-AND-DIRTY STORAGE MAP

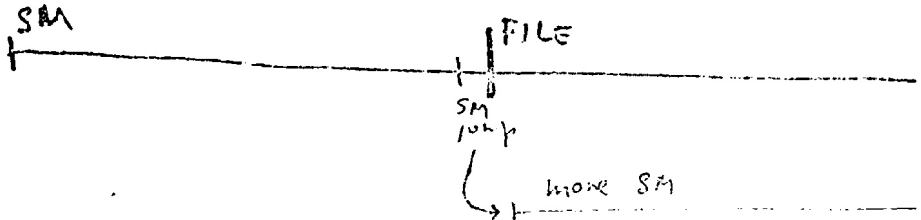
A simple storage map, for either file or stream, would have the meander codes in sequence beginning at the file-start. There is then no need to state the length of a stream anywhere, as a position in the stream may be found by adding the lengths in the meander codes during an eternal roll.

Changes in the storage map must be inserted in the correct position; therefore after each change the codes must be "slid apart" appropriately, i.e., copied a few positions over.

Filler codes (blanks) can be inserted to save sliding later.



A storage-map jump may be added when the storage map overflows. (This permits re-assembling part of a storage map elsewhere, and jumping control to it when it's ready.)



REORGANIZING STORAGE

There are several options for reorganizing storage: moving meanders; moving pointers; and changing the starting point of the file contents (as listed in the directory). These may be experimented with in the operating system.

DISK STORAGE, EFFICIENCY AND REORGANIZATION

It should be noted that the choking of the disk and the filling of the storage map are two main reasons for reorganizing storage. A lesser reason is smoothness of access, but since long rolls through entire streams are expected to be rare, this is less important than might seem at first blush. Not returns to the storage map, but track jumps, cause the greatest penalty.

Putting the forward-and-back disk addresses in the pillow means that a local fetch to babble a stream typically can be done in one turn. Collecting the storage map at the start of a stream means an arbitrary fetch in a stream can be done typically in two turns. A search through Part Map and Storage Map should usually be possible in two. (For moving-head systems, these figures refer to moving-head accesses.)

Suppose there are seven streams to be accessed, and each takes three turns; that means a delay of 21 turns of the disk, or less than one second, which is tolerable in many contexts.

PIECE BABBLE; GRABBING—MEANDERS OUT OF ORDER

Often a bed needs to be filled from more than one meander. Presumably these will be demanded in the most efficient sequence. When a number of meanders is being fetched, the operating system should go for the nearest next string, or otherwise optimize grabs, rather than hold to a forced sequence.

RE-STREAMING

Re-streaming is making new streams out of old: presumably, making neat storage streams out of a tangled bunch of meanders.

CO-STREAMING

Co-streaming is the arrangement of data in more than one set of streams at the same time. This is desirable for data complexes whose different views may not be conveniently displayed from a single separation into streams.

CROSS-STREAMING

Cross-streaming is the co-streaming of data into sets of streams which cross-cut one another: for instance, encoding the contents of a chessboard into a set of North-South streams and a set of East-West streams.

STREAM SORT

One fairly simple method of sorting within this framework uses the stream facility; we may call it "stream sort." A stream is created specifying the parts or things (strings) to be sorted, in the desired order. This sort-stream is then converted to absolute meander codes (usually more than one-for-one). It then constitutes the sorted meander list; if desired it can guide a re-streaming.

IDEALIZED DISK

The way a disk ought to be built for this system is not the way they make them now (although such instruments may be hiding behind their controllers). Desired disk services must presumably be simulated from more conventional methods.

1. ETERNAL ROLL

In the idealized machine, you can set disk input to "constant" between a given track and a given bed: information will roll from the disk into the bed indefinitely. Since the material is rolling into a bed, it never overflows. (However, the system may have to act fast to snag what it wants.)

2. BEDSCATTER

The disk should be able to read a string of arbitrary length in arbitrary position on disk into an arbitrary gap or section of a bed. Moreover, it should be able to jump among a succession of disk requests, scatter-reading and gather-writing the desired strings into their places (i.e., meander pieces into beds), correctly wrapping from first to second bed segment. With respect to a given disk track, it should allow any amount of scatter-read or gather-write, consecutively if need be (nth word to bed A, (n+1)th word to bed B). Scatter-read is necessary for getting parallel streams and odd meanders that happen to be on the same track; gather-write is needed for re-streaming and co-streaming.

SIMULATIONS OF IDEAL DISK: 1. NATTERING

Taking conventional disk controllers as given, we have to simulate eternal roll in software. One good method, if permitted by the hardware, is to keep resetting the terminator count so it never stops reading in, and resetting the address when we get to the bottom of a bed. Because of the yatata, yatata nagging-headache quality of this method, we may call it "nattering."

One slight difficulty with resetting the termination count on the run is the possibility of a change between verification and reset steps. This can be dealt with in various inelegant ways.

2. BEDSCATTER SIMULATION

Scatter-read and gather-write are not convenient on usual disk controllers. To save spins we simulate in software. We natter the material into a buffer bed, then transfer what we wanted to its proper bed on arrival. Natter is continued on a track until we are done scattering or gathering.

LEAVING FILES ON TAPE WHILE YOU WORK

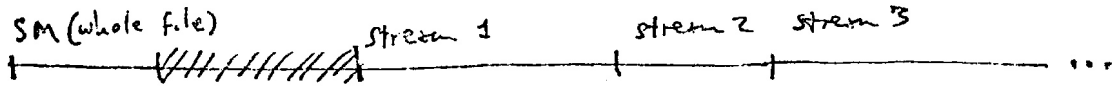
It is the general instinct to read a tape file wholly onto disk before you start anything. That is unnecessary in the present system. Except for the severe time penalty, tape can be treated like disk for general files being browsed, read or added to.

Conventions will need to be developed for files opened but only partly transferred to disk (e.g., fident and other formality material on disk).

THOUGHTS ON DISK ALLOCATION

It will be noted that the storage map can index materials on any tracks; thus expansion of a file to additional tracks is straightforward.

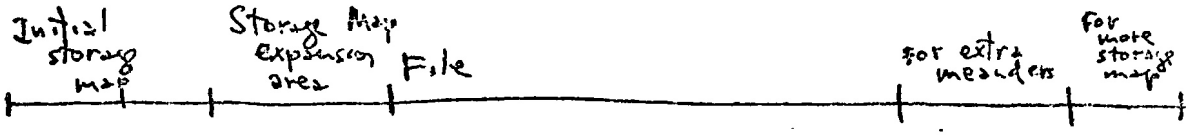
A file smaller than one disk track should be stored all-on-one, if possible.



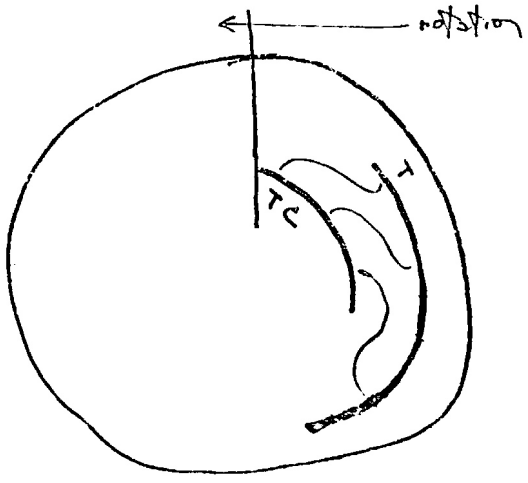
A file larger than one disk track will have to be divided. There are no obvious rules.

It is desirable to have the storage map at the beginning of a track that contains a whole file; that way we can get it all on one turn. But of course storage maps grow as meanders proliferate, and not all files are small enough to fit on one track.

This might work for awhile:



But streams will grow. In order to pick up related pieces of parallel streams in the same turn of the disk, it would be nice to offset them by an appropriate disk angle.



(The same offset should naturally apply to meanders with respect to their insertion-points.) However, since the size ratios and meander quirks are unpredictable, this is mainly a guiding thought.

FREE SPACE

Free space management requires three techniques: keeping a list of free space; regathering newly freed space to this list; and revising storage before it becomes too messy for what you want to do.

When a meander is freed, its endpoint + 1 is compared with the beginning points of other free meanders. If there is a match, the two meanders are combined to make one larger one. If you have a loose string to store, you want a meander equal to or larger than that length, either in a desired angle of the disk or bitten off from a larger piece (which allows the meander to grow, if growth is anticipated). This means the system is to keep track of freed pieces by track address and length. (Track address specifies starting angle). Meander codes of the standard type should thus suffice for the free-spaced list. Presumably this list should be kept on a single track, should list the separate tracks in sequence, and have gaps among the meander codes to simplify rearrangement.

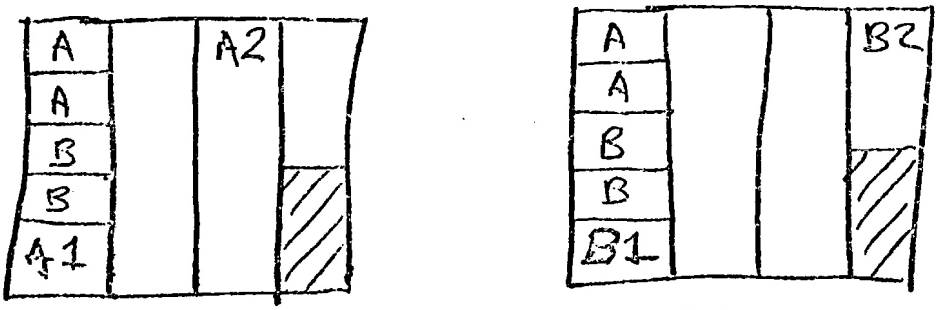
SWAPS

Swaps in the stream system are neat. The pillow contains all information for the restitution of a stream in core.

To swap in a stream, you can put the board at a different position, especially the top of a bed. The terminator can be restarted at any position as long as bed contents are the same. To swap out a stream, only newly-added portions not yet committed to disk and meander code must be assimilated; the rest can be clobbered.

We may therefore distinguish several different types of swap:

1. Bed swaps with resident pillow. The stream and bed are clobbered: the pillow stays. Pillows for several activities may reside in core at once.



2. Pillow swap. The pillows go to a swap file, and are noted in the errand stack.

3. Single bed swap. Contents of individual beds are overlay-swapped for other contents.

4. Multi bed swap. A contiguous lot of beds has its space overlaid by new beds, which may be resized.

ABSOLUTE OVERLAYS

Absolute programs can be run into the bed areas temporarily; the beds may be reconstituted from the pillows.

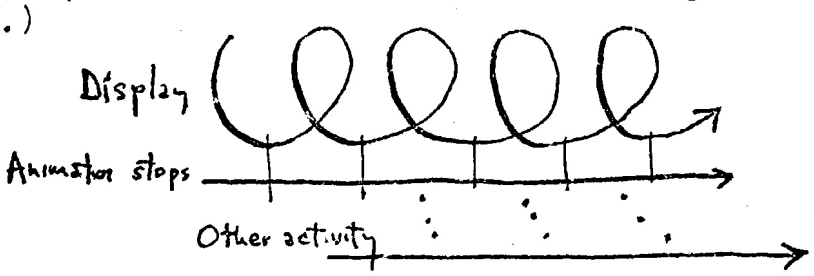
RESIZING AND DYNAMIC REALLOCATION

To resize the beds in core, a resizing routine need only change the bed-boundary words in the pillow. Beds are then rebuilt from the pillows.

OPERATING SYSTEM

The operating system mainly decides what goes in core, when to babble, when to swap. Within the conception of the envisioned text and graphic user commands, this is straightforward.

The urgent problem is the display, which must at all times be refreshed and modifed for any animations at the end of each cycle. Response to the user is the next necessity. Finally there is update. (Remote communication is an exception to this priority scheme.)



OVERALL SYSTEM

The overall system has a large number of obligations, most already mentioned. The following is a basic list. It will be discerned that they are not all that complicated, nor are their interactions unpredictable.

1. BASAL ACTIVITIES (ideally to be done in hardware)
Bedscatter
External roll/flattering
2. DISK FORMALITIES
Meander update (placement of meander on disk; reorganization of storage map)
Storage Map Update
Part Map renumbering (or if no part maps: renumbering of all internal and system references to stream)
Directory update
Free-space update & Garbage Collection
Restreaming
Costreaming
Disk Repacking
Swap: Bed Swap, Pillow Swap
Re-Sizing of beds
3. DISPLAY WORK
Stepping display: line counts, interfile connectors, etc.
Plugging, unplugging, repacking PJ Table
Updating & repacking Animation Table
4. FILE FORMALITIES
File security protocols (as yet undefined)
Registry of new files
Update of geneology
Lockout of edit operations during disk catchup
5. NOT DISCUSSED HERE, BUT MORE OR LESS WHAT YOU'D EXPECT
Servicing user requests
External communications
Remote communication of files, both in and out
Providing small Entertainments during swap, etc.

SYSTEM LISTS

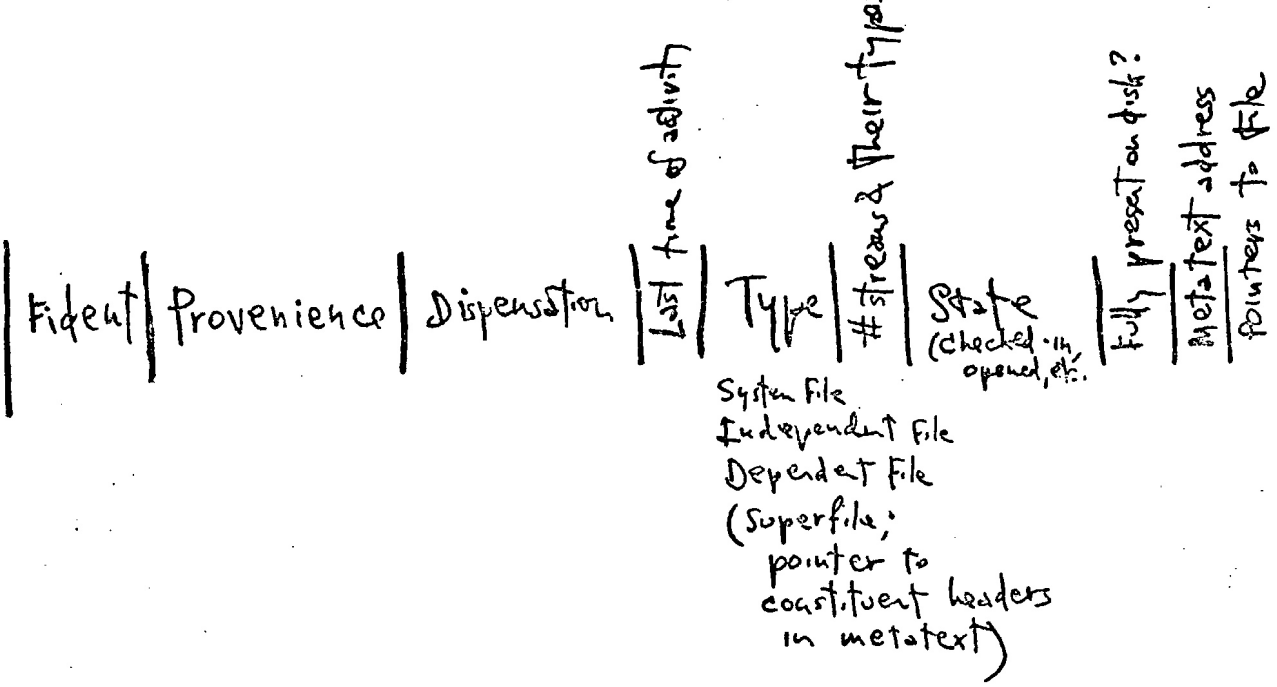
Various system lists need, of course, to be maintained by the operating system. Their exact selection and layout can be worked out later. They are, of course, streams.

NO INTERRUPTS

It will be noted that the system described does not need an interrupt facility to service user requests. Polling all user input registers, either between frames or in a mainframe program concurrent with and independent of the display, should do nicely.

QUICK-AND-DIRTY DIRECTORY CODE

The Directory is composed of a number of fixed-length Directory Items or file heads. They need not be in any sequence, as we can eternal-roll through them. Each has eight addresses pointing to up to eight of the file's stream storage maps (with provision for continuation). It also has a great deal of other information about the file which will eventually be necessary. Variable-length fields of the Directory Items are stored in a "metatext" stream.



FIDENT

Certain information is necessary for file identification, protocols of request, arrival, security and genealogy.

The Fident, or Fiduciary IDENTifier, contains a file name and security and formality material, as yet undefined. The security and formality material will include alphameric checksums and hidden keys, usage status information, file opening and closing codes, etc.

PROVENIENCE

To make sure we don't run out of numbers for a decade or so, we number eras and overall context, calling them Proveniences. The number 00000001, the first Provenience, should hold us for awhile.

DISPENSATIONS AND HISTORY

A numbering of parts in a file is a dispensation. Over time the numbers get higher and higher, and a lot of the lower ones are lost. It becomes time for a New Dispensation.

A new dispensation is recorded as such in the file history. All the existing parts are given lowered numbers, beginning with 1.

FILE GENEALOGY

Dispensations fork out in evolutionary trees; a file may have any number of children, grandchildren, great-grandchildren... and the genealogical derivation of each is stored in the file history. However, dispensations are numbered sequentially in the order of their creation; the history must be consulted for parentage. A file history is kept informed of its descendants.

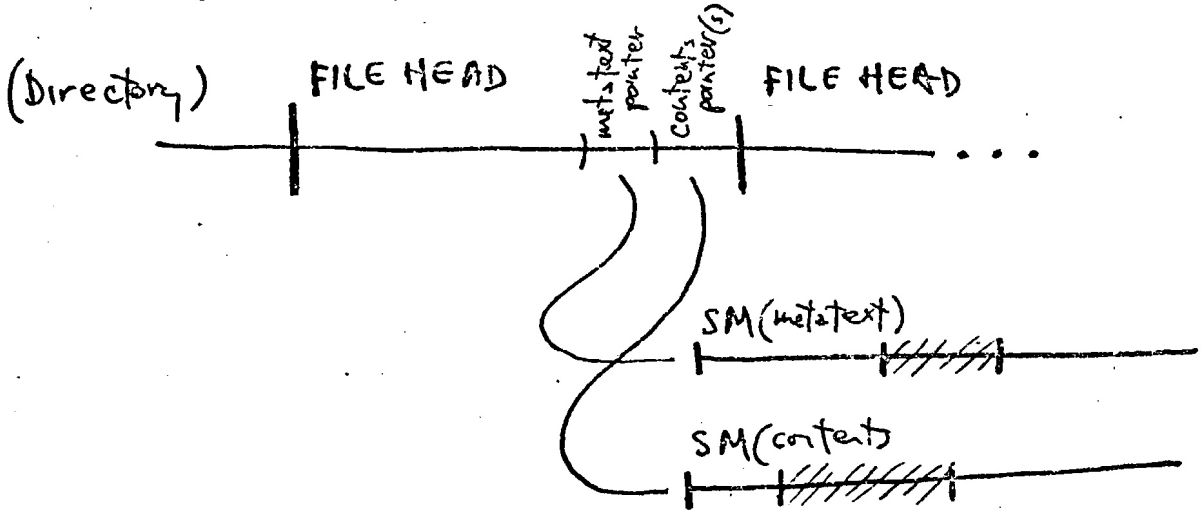
BASTARD FILE GENEALOGY

If, however, a file is not able to communicate with its family history, it is nevertheless able to have descendant dispensations. However, these are called bastards. (Their parents do not know or acknowledge them.) They are supposed eventually to get their genealogy straight by registering with the main history.

METATEXT

Messy text fields about the file, such as the file's name, and other fields indicating authorship, ownership, copyright, exceptions and whatnot, belong off elsewhere. Since they are not "in" the file, but about it, and subject to infrequent citation and change, there is a case for storing them separately from the actual streams comprising the file.

A good arrangement would seem to be:

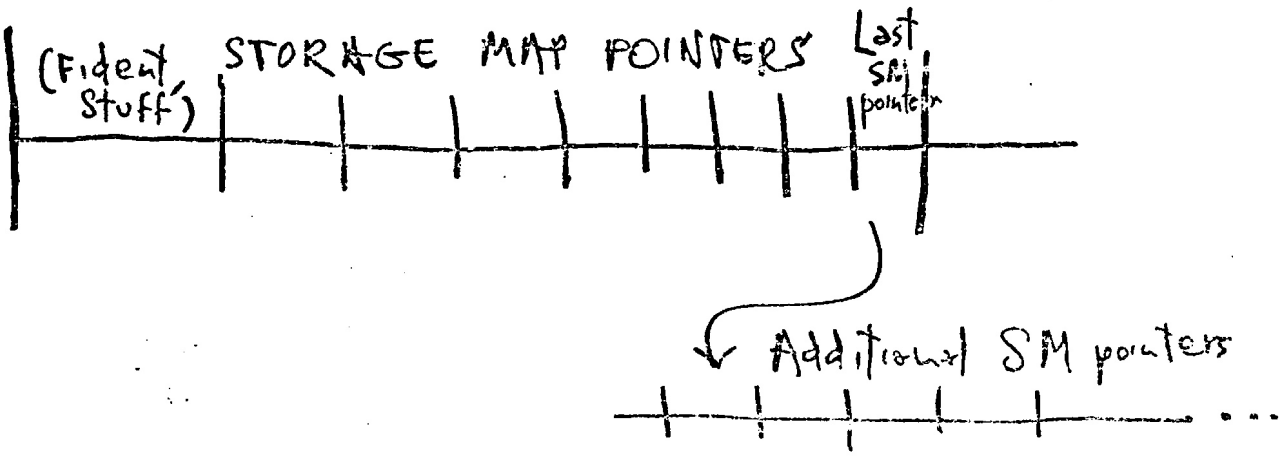


For complete freedom in moving files and their storage maps on disk, it is probably best to have this metatext associated with the main system directory, rather than the content streams. Then everything not in the directory may be relocated on disk by the system without complication.

STORAGE MAP POINTERS

Every stream has to have its own storage map. (The stream then consists of all the meanders pointed to between the beginning and end of that storage map.)

The directory item has space to point at eight streams, as we vaguely expect a file to have eight streams or fewer. But continuation is possible to any number of map pointers.

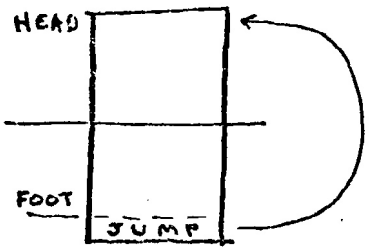


IDEALIZED DISPLAY FOLLOWER

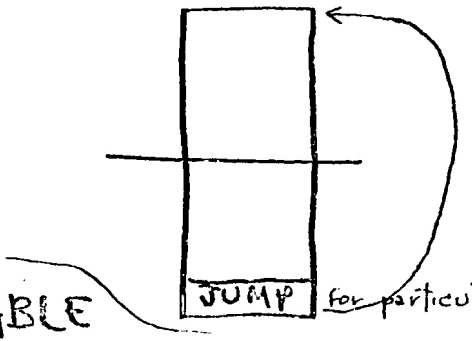
The pure Xanadu machine has a display program follower that leaps from stream to stream, saving counts, decrementing with each element displayed, and returning at termination of the count to the stream that dispatched it, recursively. These returns are naturally consummated by means of a stack.

THE LAST JUMP

The pure Xanadu machine, casually reading down a bed, will want to know when it gets to the end of a bed without checking the pillow. Sometimes this will be by the usual string count. In other cases, this (not pure data) can be done with a standard Xanadu Jump Code (undefined) just below the foot of the bed.



Simulating Xanadu on conventional equipment, we will often want to put a display jump for the particular display below the foot of the bed.



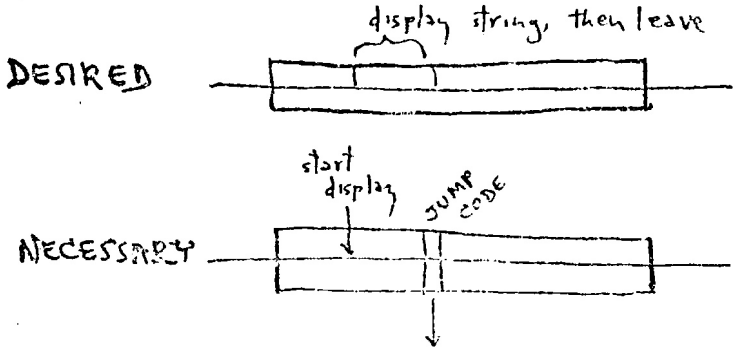
PLUGJUMP TABLE

Classical Xanadu has hardware that jumps between streams. What conventional hardware won't yet do we simulate.

If we simulate Xanadu on conventional equipment, the game changes entirely but stays interesting.

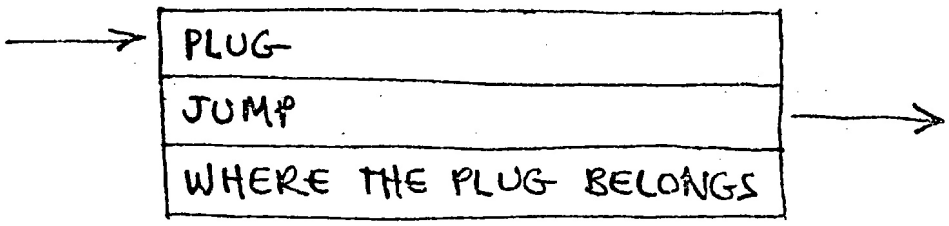
We keep the hierarchial streams, but the display no longer cycles through them. In the first display pass or a pre-pass, we format the display and produce a control table which dispatches all the subdisplays in the separate beds.

A most curious practice becomes desirable. Since we leave all material to be displayed in its beds, and since we usually lack a counter that will jump out of the bed at the terminator of a string of data, we must insert a display-jump code in the data stream.



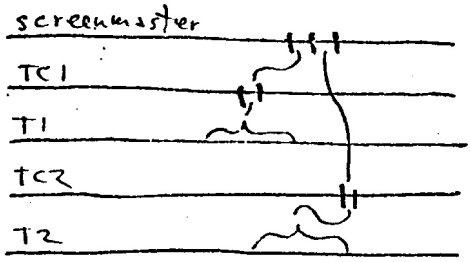
Either we are going to slide stuff apart to make room for this code, which we aren't, or we are going to have to put elsewhere the element(s) clobbered by the jump code. (If the element clobbered were always going to be one alphameric space, we wouldn't need to, but it isn't.)

Thus the plugjump table, made up of plugs and jumps. A "plug" is a little datum that has to be taken out of something else. A jump is a jump code. A plugjump is this:

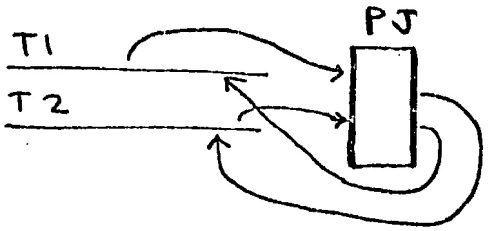


The jump at the end of a string to be displayed points to its plug, which had to be moved to put the jump in. The jump after the plug goes where you actually want to go. The plug is displayed as it should be, and you can, if need be, put it back. (E.g., to re-shake text to new line divisions after an insert.)

The PJ is naturally in a bed. There should be a PJ bed for each text window: thus the views can babble separately, with the PJ babbling like any other bed, to match the changing text and resegment only the new matter, with the old PJs remaining valid till their strings leave the display.

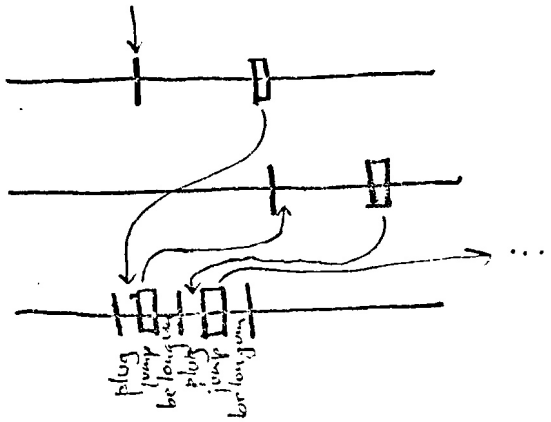


IDEAL (with more streams, actually)



ACTUAL PLUGJUMP

or, more literally,



ANIMATION

In general, text (and line and dot graphics) may be smoothly animated on the screen. What needs to be moved, and how much, is determined by the particular function program. Moving it on the screen is the duty of the animation table.

We will consider only linear motion, for simplicity. To obtain linear motion on the screen of a word or part of a drawing, it is necessary to add fixed increments (positive or negative) to its x and y coordinates before each frame display. (Intermediate rates, involving see-saw increments between frames, we will not consider here.)

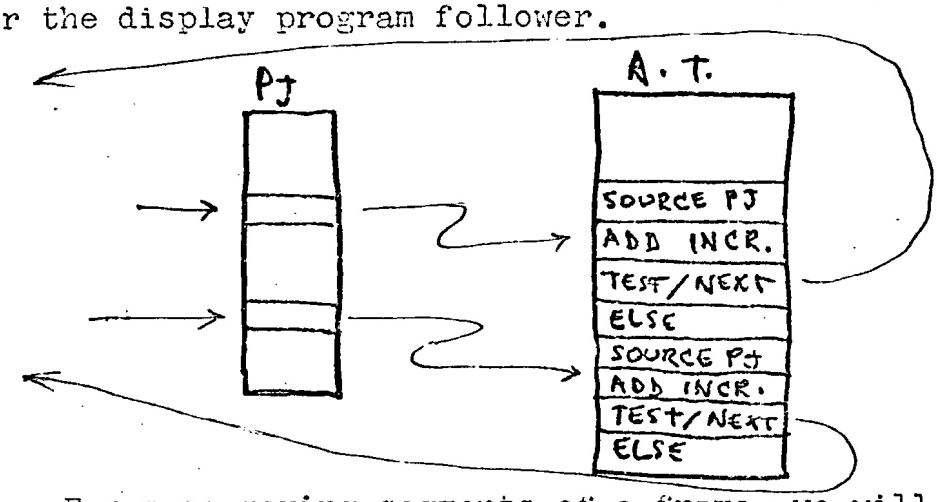
The simplest method is to add the increments at the end of each frame. Thus no interrupts are needed. A table specifies the parts of the picture to be moved.

Various forms of animation table are possible. We will consider here a type that works well with the plugjump table, being a plugjump table itself. There may be a simpler variant.

ANIMATION TABLE

The animation table states the motion increments between presentation-frames for individual parts of the picture.

Let certain plugjumps indicate the beginnings of parts to be animated. These we call moving plugjumps. Let each of these jump to a second table, the animation table, where the increments and tests are stored. These increments specify x and y changes for the display program follower.

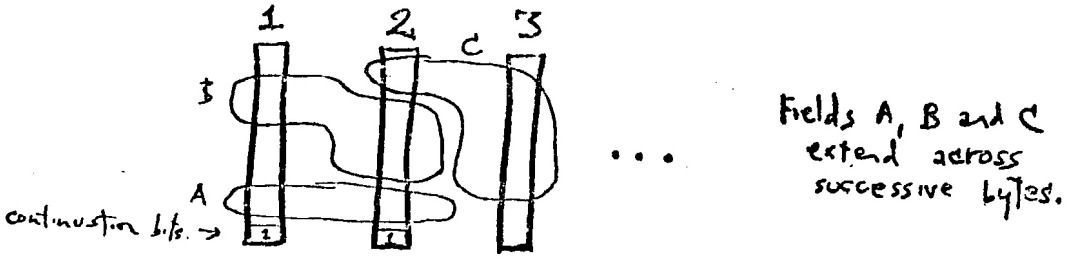


For some moving segments of a frame, we will want to consider whether the motion should change at the completion of the frame. Typically, then, tests will be associated with the end of the frame as to which parts of the picture continue their movement as before, and which enter a new phase of movement.

At the end of each frame, the Animation Test is made for each part in motion, testing completion of the motion. Normally the motion is not completed and the Animation Increment is added to the Animation Step. If the test shows the motion to be completed, the Else Code is followed, presumably a termination of motion or a jump to some other program.

XANADU UNIVERSAL CODE?

It has seemed desirable to contrive a universal addressing code for stream selection, interstream citation, disk addressing, display commands, I/O and other purposes. This would presumably be byte-oriented with a continuation bit. A further desirable feature is what we may call "field continuation," the use of bits in bytes extended after the first to extend and modify the address fields of the first and later bytes, e.g.:



However, this waits on further specification of system functions, and the discovery of further divine design convergences.

HARDWARE

Different aspects of hardware implementation are under investigation, and possible combinations and separations for them.

1. Bed-box. An addressing mechanism we may call the "bed-box" would hold the head and foot addresses of the bed and a current address. It would step the current address forward and back on request, and, on reaching an end of the bed, wrap to the other end.

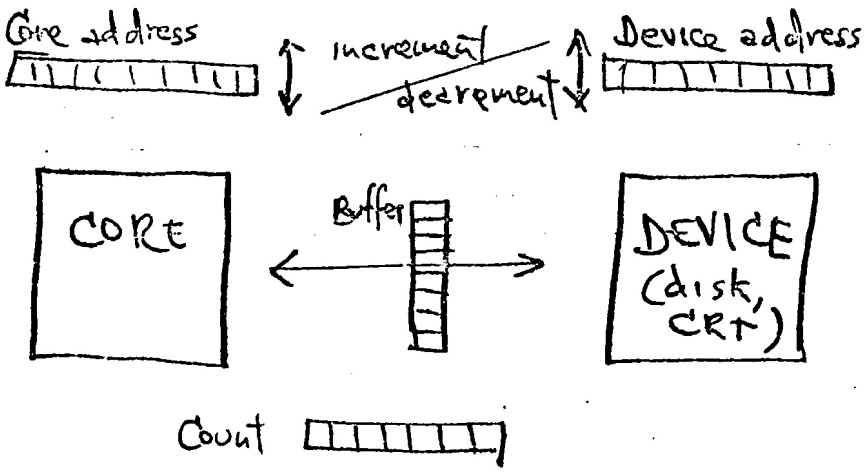
A board check register should perhaps also be included in the unit.

2. Address cruncher. A larger unit built around the bed-box, this would perform an address conversion from a stream address, the ficuciary pair, and current bed information.

¶It would take an absolute or relative stream address, compare it with current stream addresses in the bed (wrapping from first to second segment), and return the absolute bed address of the desired stream element or the advice that the element is not in core, and how far forward or back it is from the fiduciary pair or the current view.

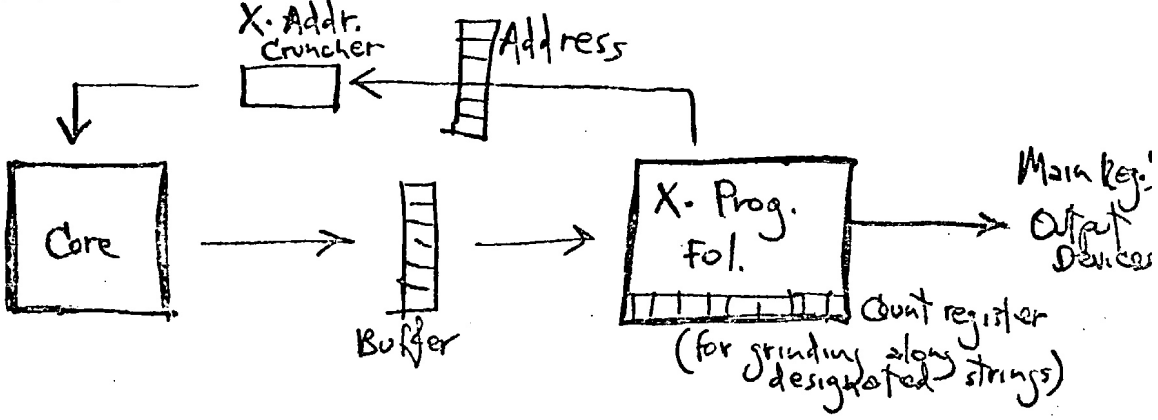
A reasonable approach would be to have this as a fast box or in firmware, dipping into the pillow information in core. In this case the routine could also specify immediately any address in the current, forward and backward meanders.

3. Stream channel to external devices, like data channel but more so. Conventional data channel and controller:



Stream channel: same but with automatic conversion to 2-segment bed, moving terminator. Variations: Bidirectional for forward or back-babbling; multidimensional for array window.

4. Xanadu Program Follower: Device which follows parallel-stream structure.



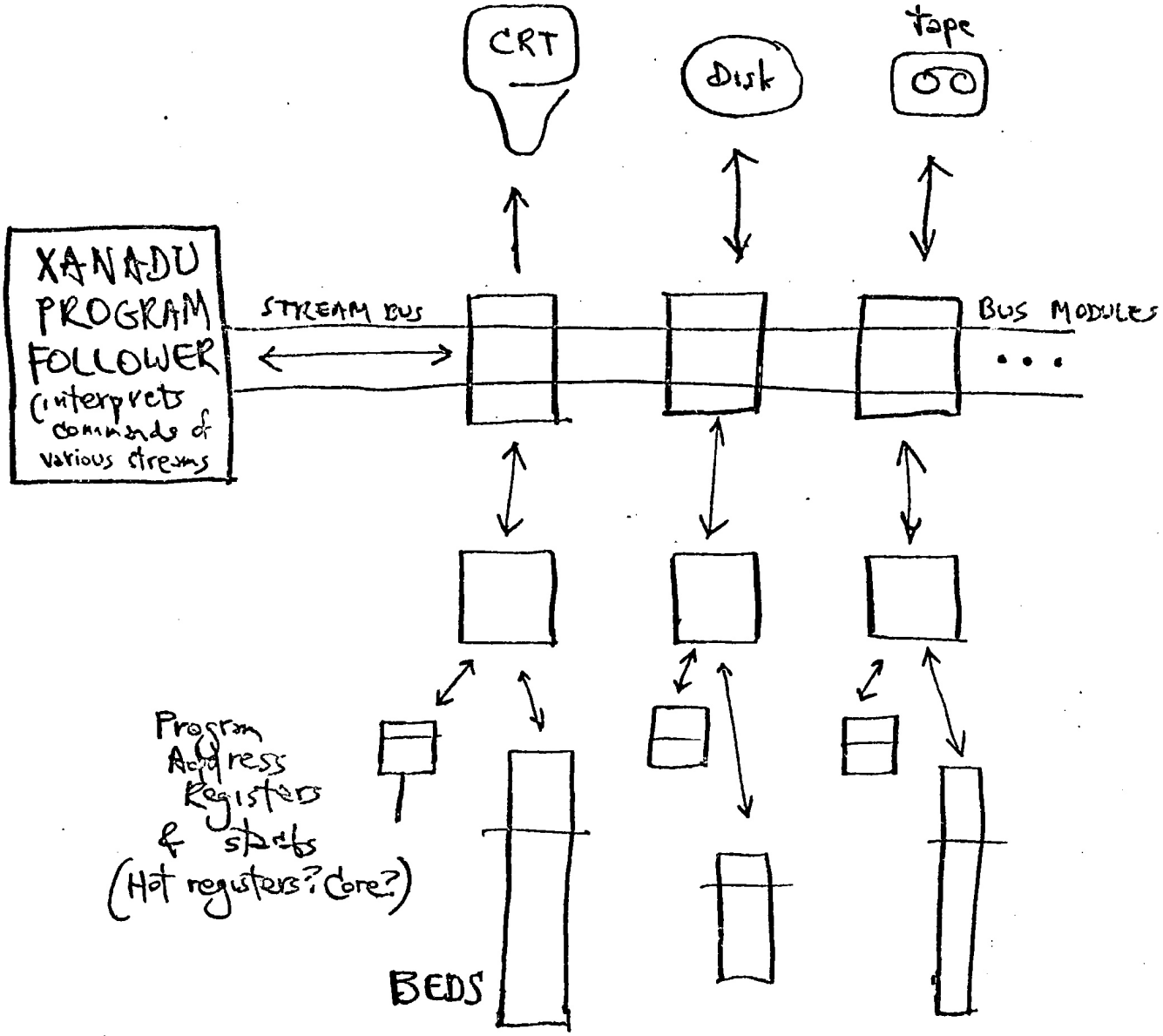
5. Trap system to snag jumps between streams: esp. for generalization of events and programs, e.g. substitution of matrix for arithmetic operations.

6. General-purpose instruction-set, suited to generalized application of parallel-stream monitor to all computing.

7. Data channel add-on for existing computers embracing all of these or coherent subset.

8. Computer embracing all of these or coherent subset.

9. A universal bus-oriented hardware system. This would service many beds in parallel, performing address conversions from stream-absolute to bed-position (the bed-box); step forward and back n positions, scatter reading, writing and displaying.



POSSIBLE ARCHITECTURE FOR XANADU MACHINE EMPLOYING UNIVERSAL CODE (structures of family - addressing, traps and priorities omitted)

10. The program follower (code interpreter) would be time-shared among the several peripherals and programs, converting the stream jumps and steps into address conversions to core for each program currently being followed.

PROGRAMMING

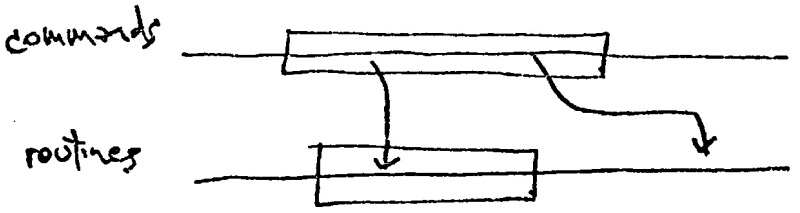
The stream system described is a whole new set of programming conventions. Programs can be written under these conventions, but they are different. (In a software Xanadu implementation they will be wholly interpretive, in a hardware system not.) Where a Fortran programmer declares arrays, the X-stream programmer declares streams, families and bed sizes. He may straightforwardly specify input streams, output streams and buffer streams (both stack and FIFO). Not the nth, but the "next" stream item, is a typical element of concern.

Programming is in general procedural rather than declarative. Methods of coding to roll forward and backward to pre-specified "stops" are under consideration. The different types of swap will be available.

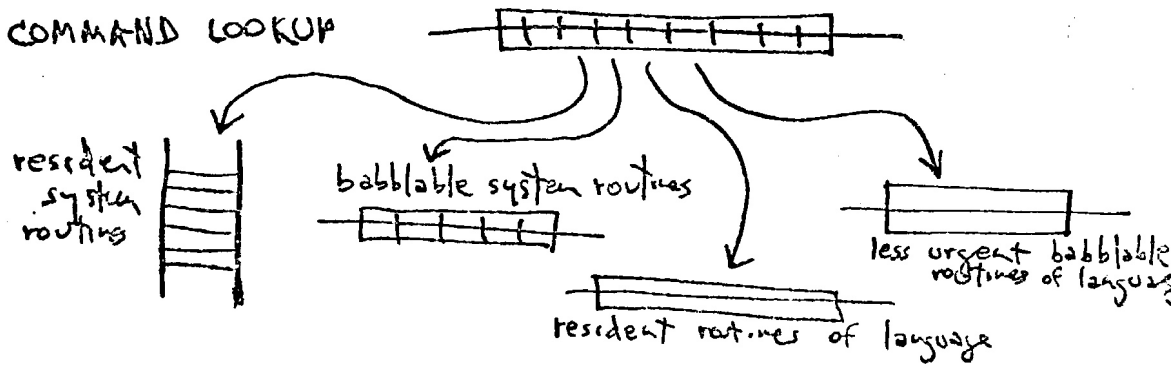
LANGUAGES

The programmer may create new interpretive languages under this system which use and combine old system macros and his new code.

To begin with, we think of the language command as looked up in a table which then branches to the routine. This is then two streams:



except that it is desirable to use code already available to the system. Thus we expand it.



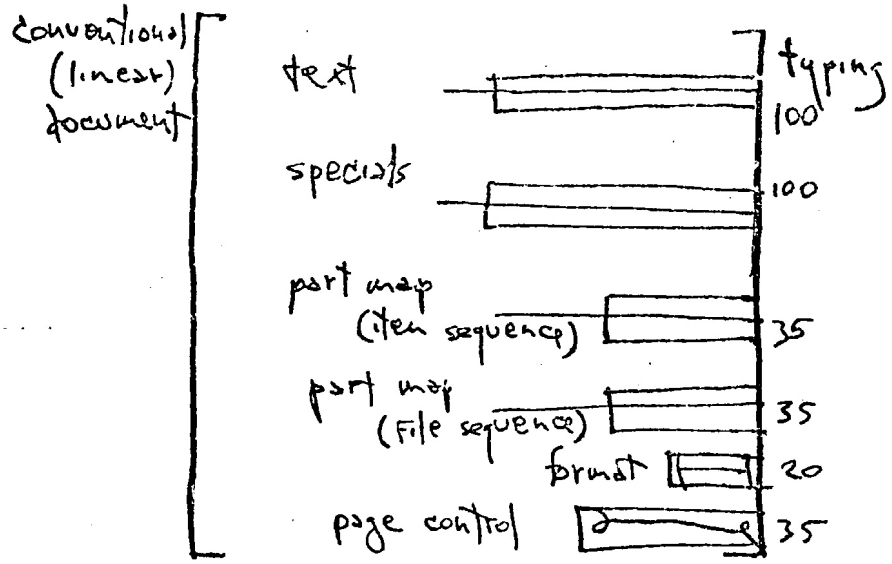
A command in the language causes a lookup of the instruction in a command lookup stream. This either points to a system macro, or to code stored in core (presumably another stream), or to a system macro or subroutine in core or ROM.

Streaming permits a fairly large command structure, at some overhead cost in stream fetches.

Additional streams for more commands, input, output and buffering, may all be declared by the programmer.

TYPEOUT SYSTEM

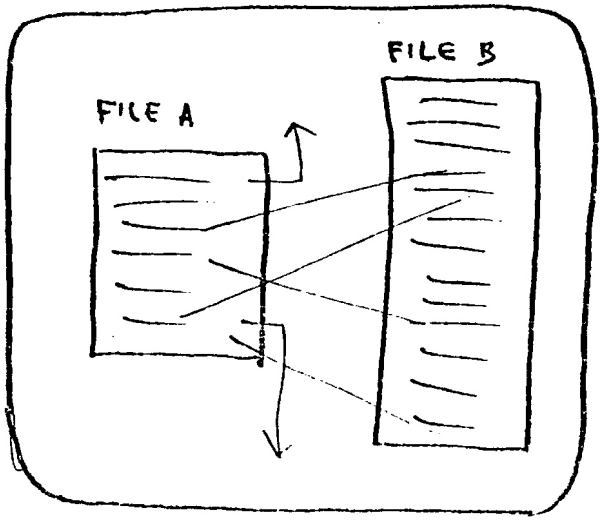
A typing program which handles printout as a shared job may be implemented with six data streams.



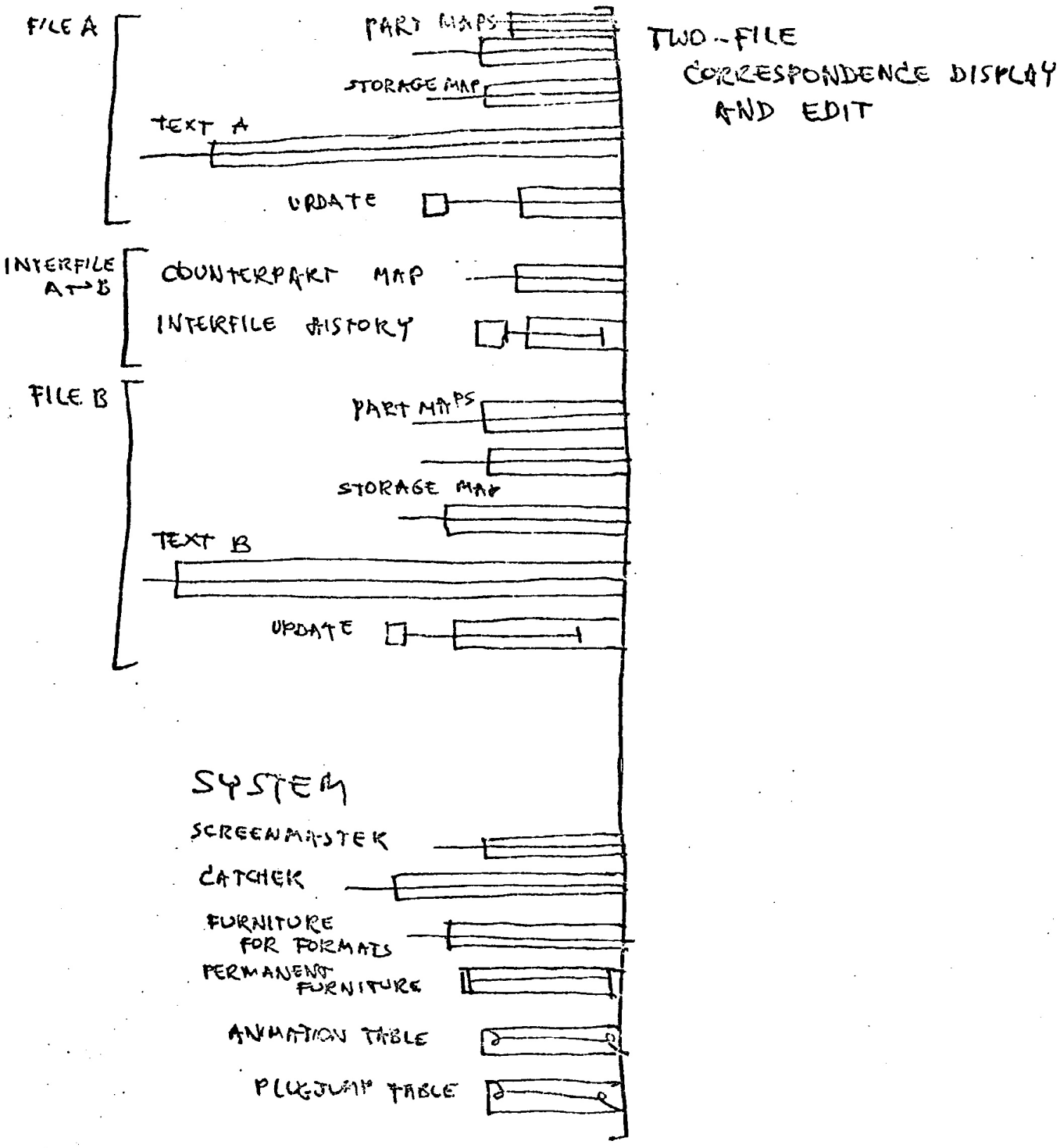
Specials include footnotes, headers, etc.
The part maps indicate paragraphs, sections and specials.
"Format" tells line width, header and footnote positioning.
Page control is generated on the run, formatting from the part map and format stream. If desired, it may be saved for later duplication of the document without examination of part maps.

PROGRAM FOR NAÏVE FRONT END

It will be recalled that in the text system described at the beginning, files may be viewed in separate windows with their couplings displayed, and rolled on a dependent or independent basis.



This system may be implemented with the following function layout of streams and beds.



It will be seen that to create the controlling stream function program for this is relatively straightforward, if not trivial.