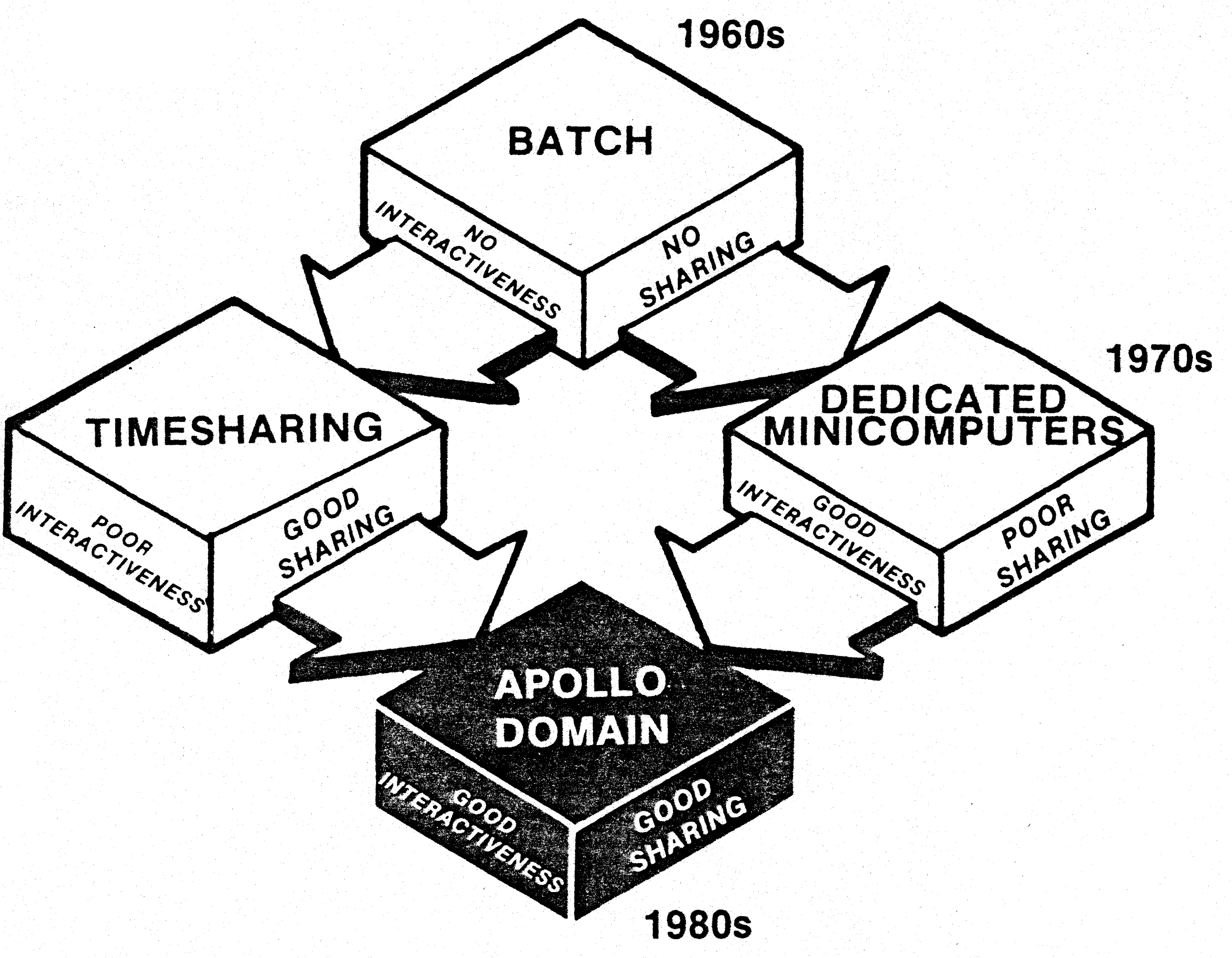
apollo computer

APOLLO DOMAIN ARCHITECTURE

(February 1981)

apollo computer inc.
5 Executive Park Drive, N. Billerica, MA 01862 617-667-8800

ARCHITECTURE EVOLUTION (1960-80)



I.1 ARCHITECTURE EVOLUTION:

This figure depicts the evolution of architecture over the past 20 years. In the center diamond at the top we show batch computing of the 1960's which is characterized by, first, very little or no interactiveness and, second, very little or no sharing of peripherals and data files. In the late 1960's computer architecture evolved into two distinct forms. On the one hand there was timesharing which was intended for people who needed large machine architecture, but could sacrifice certain degrees of performance and interactiveness. Timesharing systems are characterized by poor interactiveness but very good sharing characteristics and also large machine architecture. On the other hand batch evolved into a form called dedicated minicomputers. Minicomputers are characterized by having good interactiveness, good human interfaces, and very good performance, but lacked in the sharing of peripherals and data among a community of users.

The **Apollo DOMAIN** system has evolved as a direct result of improvements in technology and is widely held to be the architecture of the 1980's. It combines the good parts of both timesharing and dedicated minicomputers, but eliminates the disadvantages of both of these earlier forms. The **Apollo DOMAIN** system has good sharing capabilities provided by a high speed interactive network as well as interactiveness provided by a dedicated computer available to each user.

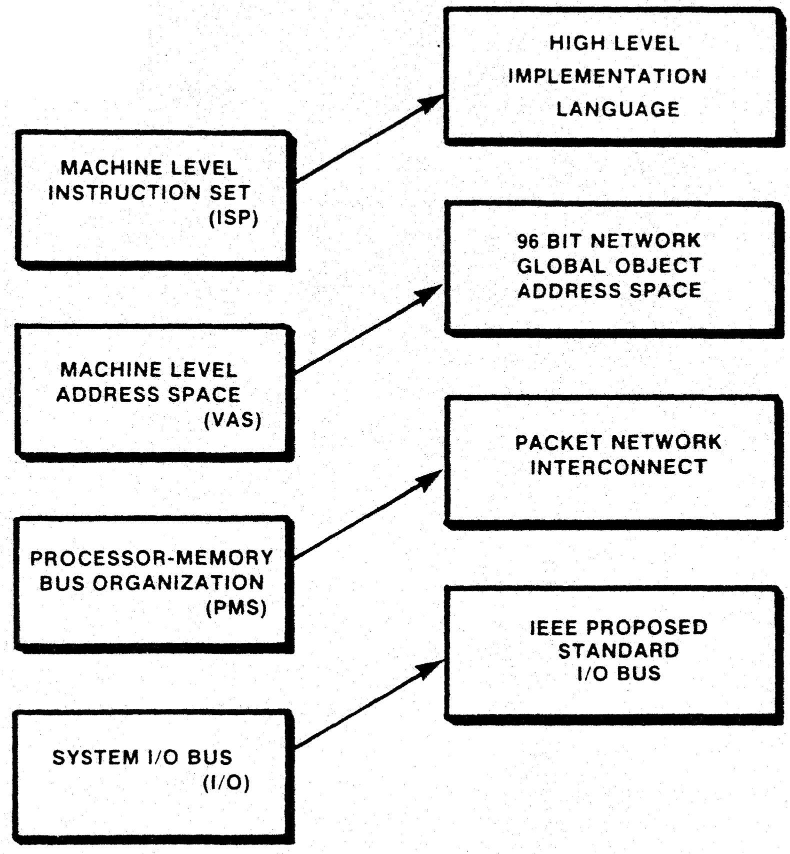
GOVERNING PRINCIPLES

- DEDICATED **CPU** PER USER
- INTEGRAL WIDE BAND LOCAL NETWORK
- HIGH LEVEL DESIGN (**ISP, VAS, PMS** INDEPENDENCE)
- USE OF ADVANCED TECHNOLOGIES
  (**VLSI CPU, WINCHESTER DISKS, etc.**)

I.2 GOVERNING PRINCIPLES:

There are several principles that have been used to govern the design of the Apollo computer system. First, and foremost, is the notion that there exists a dedicated CPU for each user. Second, each user is interconnected with a high performance local area network. Third, the design of the architecture is based on high level abstractions so that we may independently evolve lower level components (such as the instruction set, or internal buses) with minimum impact. Fourth, is the use of advanced technologies, such as VLSI, Winchester disks, and so on.

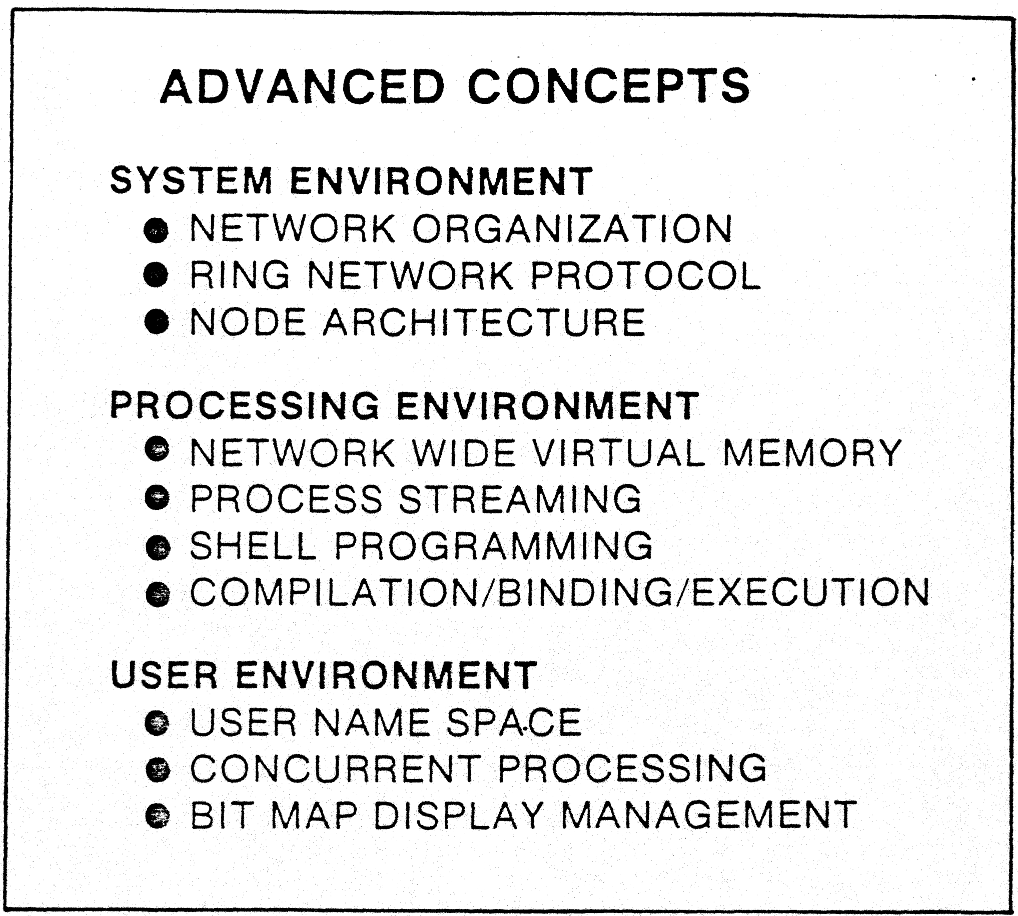
HIGH LEVEL DESIGN/IMPLEMENTATION


MACHINE LEVEL INSTRUCTION SET (ISP)
MACHINE LEVEL ADDRESS SPACE (VAS)
PROCESSOR-MEMORY BUS ORGANIZATION (PMS)
SYSTEM I/O BUS (I/O)
HIGH LEVEL IMPLEMENTATION LANGUAGE
96 BIT NETWORK GLOBAL OBJECT ADDRESS SPACE
PACKET NETWORK INTERCONNECT
IEEE PROPOSED STANDARD I/O BUS

I.3 HIGH LEVEL DESIGN/IMPLEMENTATION:

The Apollo system incorporates designs which are uniformly advanced, or appear at a higher level than conventional computers. A conventional computer is characterized by: (1) a machine level instruction set or what we call an ISP, (2) a machine level address space or a virtual address space which is a measure of the range of addressing that the computer can span, (3) the processor memory bus organization, or what we call PMS, including the memory buses, the attachment of processors, the attachment of multiple memory units and so on, and (4) the I/O system of the computer, or the I/O bus.

The Apollo system is designed around higher level abstractions in each of these particular areas. For example, rather than an instruction set, we talk about a high level language implementation, namely PASCAL. Similarly, instead of a machine level address space, such as the 24 bit address space of the Motorola 68000, we talk about a 96 bit network wide global object address space. Our thinking here is that objects are very large entities that are 32 bits in length and whose location should be anywhere on the network. This 96 bit network wide object address space is the fundamental system address in the **Apollo DOMAIN** system, and is designed to accommodate various machine level address spaces. Similarly, rather than designing the system around a processor memory bus organization, the Apollo system is designed around a two address packet network. This network is used to attach computation units, peripheral units and gateways to other systems. It is the backbone of the system allowing users to intercommunicate, to access shared programs and data files and for access to shared peripherals. Finally, our I/O bus is not an integral part of our internal system, but rather an IEEE proposed standard MULTIBUS which is externally available to users and is widely acknowledged as a standard for small computers in the computer industry.



ADVANCED CONCEPTS

SYSTEM ENVIRONMENT

- NETWORK ORGANIZATION
- RING NETWORK PROTOCOL
- NODE ARCHITECTURE

PROCESSING ENVIRONMENT

- NETWORK WIDE VIRTUAL MEMORY
- PROCESS STREAMING
- SHELL PROGRAMMING
- COMPILATION/BINDING/EXECUTION

USER ENVIRONMENT

- USER NAME SPACE
- CONCURRENT PROCESSING
- BIT MAP DISPLAY MANAGEMENT

I.4 ADVANCED CONCEPTS:

There are many advanced concepts that have been applied to the Apollo architecture and they can be roughly broken down into three general categories: (1) those pertaining to the overall system environment, (2) those pertaining to the program environment, (3) those pertaining to the user environment. It is useful to point out certain particular features that have been incorporated into the **DOMAIN** system in each of these environments.

The Apollo system environment is unique in the sense that the architecture is based on a network as opposed to a central systems architecture. This network allows shared data and peripherals, and is controlled by an object oriented operating system that will be described in more detail later.

The processing environment for the Apollo system includes: (1) a very large linear address space for virtual memory management, (2) advanced concepts, such as stream I/O which will be described later, and (3) new ideas such as shell programming which allow people to build procedures at the command level.

The user environment of the **Apollo DOMAIN** system is radically different from conventional systems. Rather than a character oriented dumb terminal, the Apollo system has for each user an integral bit map display. This parallel device allows many concurrent programs to be executing on behalf of each individual user, which is accomplished by dividing the display into multiple independent window areas.

SYSTEM ENVIRONMENT OBJECTIVES

NETWORK MODULARITY
- WIDE PERFORMANCE RANGE
- HIGH AVAILABILITY

RING NETWORK
- HIGH SPEED / LONG DISTANCE
- MULTIPLE TECHNOLOGIES

MAXIMIZE NETWORK INTERACTIVENESS
- NO SUPERFLUOUS MESSAGE BUFFERING
- MAXIMUM DMA DATA RATES.

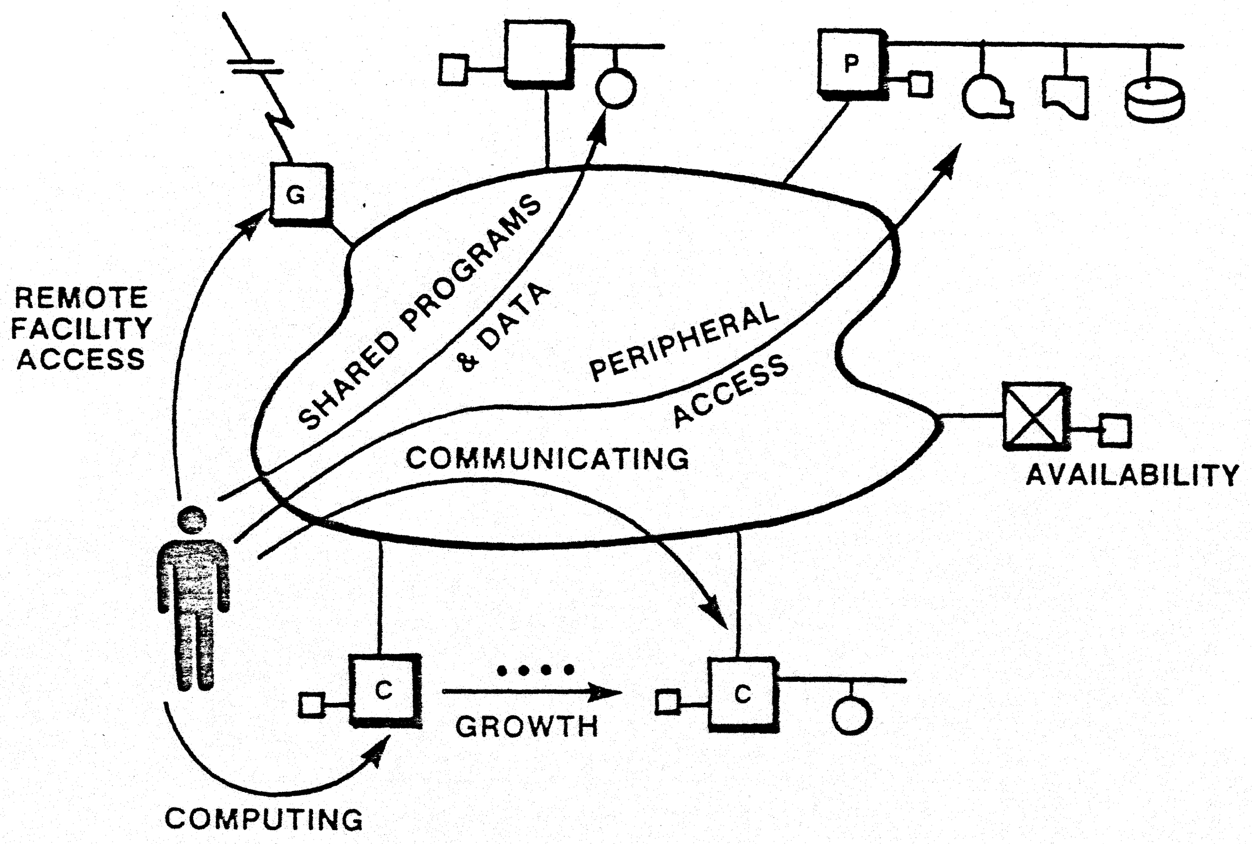
II.1 SYSTEM ENVIRONMENT OBJECTIVES:

Network modularity was a principal design objective of the Apollo computer system, providing a wide range in performance, a wide range in growth capability, and a wide range in system level availability. Modularity at the network level allows users to incrementally expand their system by themselves on their site, and without substantial programming. It means that they can replicate nodes to obtain very high availability. It further means that the overall system configurations can conform to the user's specific application in the most cost effective way he chooses. From a manufacturer's point of view, network modularity significantly eases system maintenance, allowing the replacement of entire nodes as well as the ability for one node to diagnose another.

A second design objective for the Apollo system environment was to incorporate a high performance coaxial local area network. Although our system is designed to accommodate any two address packet transport mechanism, the specific implementation that Apollo has chosen involves a ring topology. Rings have numerous advantages over alternative approaches: They generally allow higher data bandwidths and longer distances, they allow migration to new technologies such as fiber optics, they are very interactive allowing very fast network arbitration, and finally they incorporate a free acknowledgement function with the circulation of each packet.

A third system environment objective was to maximize network interactiveness. In this regard, our design eliminates all superfluous message buffering between nodes, allowing a message generated from one process to be transmitted directly to another process on a separate machine. Secondly, our network controller transmits data through the block multiplexor channel which allows all high performance DMA devices to have access to the total memory bandwidth of both machines. Consequently, when a message is transmitting from one machine to another, the data rate is at the maximum possible permitted by the two memory systems.

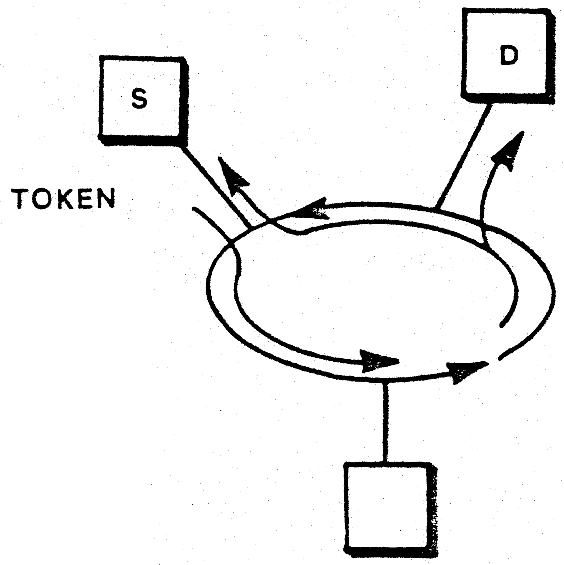
SYSTEM ORGANIZATION



II.2 SYSTEM ORGANIZATION:

The system level organization of the Apollo system is based on the **Apollo DOMAIN** network. This network allows an extremely wide range in performance, growth and system availability. Moreover, users attached to the system can intercommunicate, can access shared programs and data files across the network, can access common pools of peripherals, and can finally access remote facilities, including large foreign machines or other **Apollo DOMAIN** systems. Consequently, the **Apollo DOMAIN** network together with the per user computing node is intended to provide an entire computing facility to each user.

RING NETWORK PROTOCOL



F	DST	SRC	HDR	DATA	CRC	ACK	CRC	T

TOKEN=011111100
FRAME=011111101
MESSAGE HEADER=011111110
MESSAGE SEPARATOR=011111111

II.3 RING NETWORK PROTOCOL:

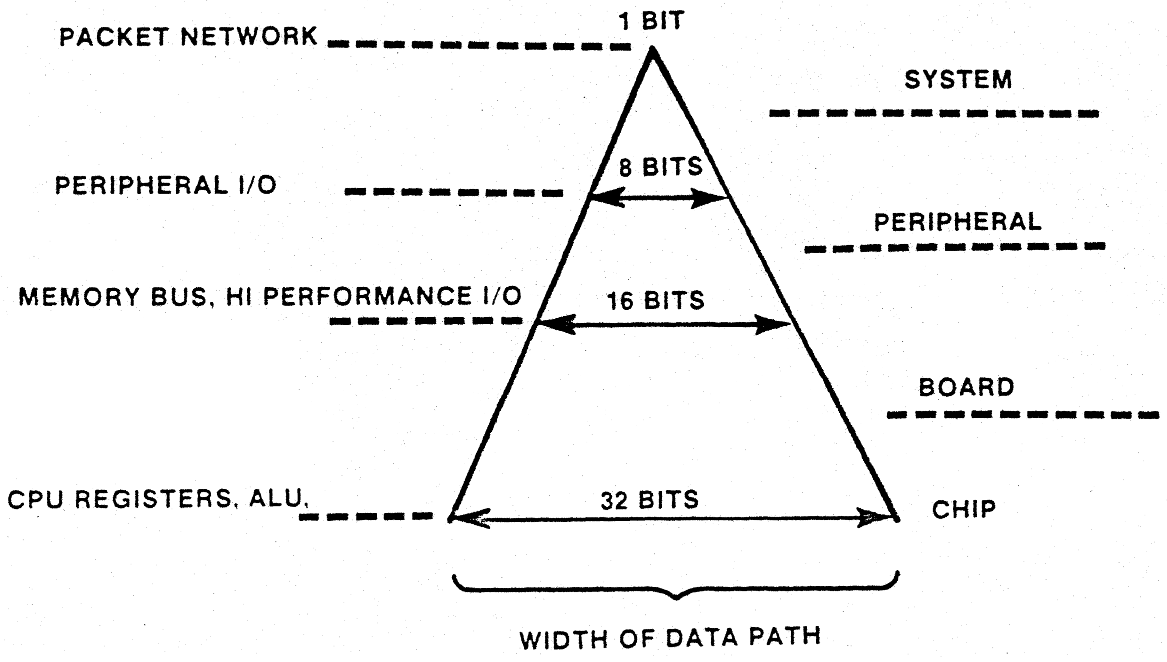
The **Apollo DOMAIN** system is designed around a two address packet transport network. The specific implementation of this network can take various forms, and the system is specifically designed to be able to migrate from one form to another as the technology requires.

The topology of the Apollo network is in the form of a circular ring. Access to this ring is arbitrated through the passing of a TOKEN which is a specific encoding of bits passed from one node on the network to another. The system allows one and only one TOKEN to be on the ring at any given instant, and the possession of this single TOKEN gives a particular node exclusive use of the network for the duration of a message transmission.

The format of the message on the ring includes the destination node address, the source node address, header information, data, a CRC check, and finally an acknowledgement field. The acknowledgement field is adjusted by the destination node, thereby acknowledging the correct receipt of the packet to the source node.

The encoding on the ring uses a conventional bit stuffing technique whereby the occurrence of five consecutive 1's causes the insertion of a 0 on transmission and a corresponding removal of the 0 upon reception. Several special flag characters are used to establish packet synchronization and are encoded as a string of six consecutive 1's followed by two identifier bits. One of these is the TOKEN which deviates from other flag characters by only the last bit thereby allowing a node to exclusively acquire a TOKEN by simply altering a single bit. This allows minimal buffering in each node and therefore maximizes network responsiveness.

32 BIT SYSTEM HIERARCHY

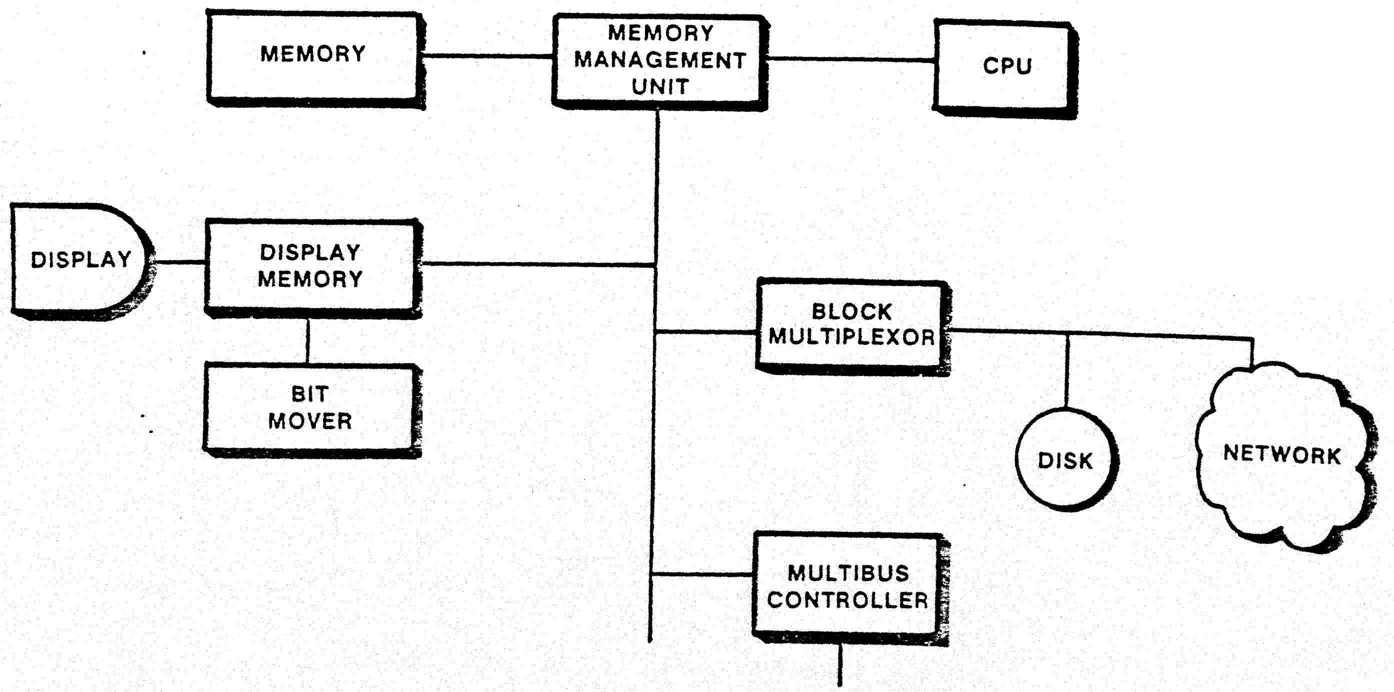


II.4 32 BIT SYSTEM HIERARCHY:

The Apollo central processing unit is built around a VLSI microprocessor with 32 bit architecture. The instruction set of the processor includes both 32 bit data types as well as a 24 bit linear virtual address space. The physical parameters of the system, most notably the width of the data path, can be viewed in a hierarchical arrangement. At the system level computer nodes are interconnected with a 1 bit serial packet network. Certain peripherals attached to an individual computer node are interconnected with 8 bit (1 byte) data paths, whereas, the memory system and high performance peripherals operate on a 16 bit data path. Internal CPU registers and an arithmetic logic unit are all implemented with full 32 bit data paths.

Consequently, the CPU is generally 32 bits wide, the memory system is generally 16 bits wide, while the network system is only a single bit wide. The width of the data path varies inversely with the physical distance from the internal processing registers.

NODE ORGANIZATION



II.5 NODE ORGANIZATION:

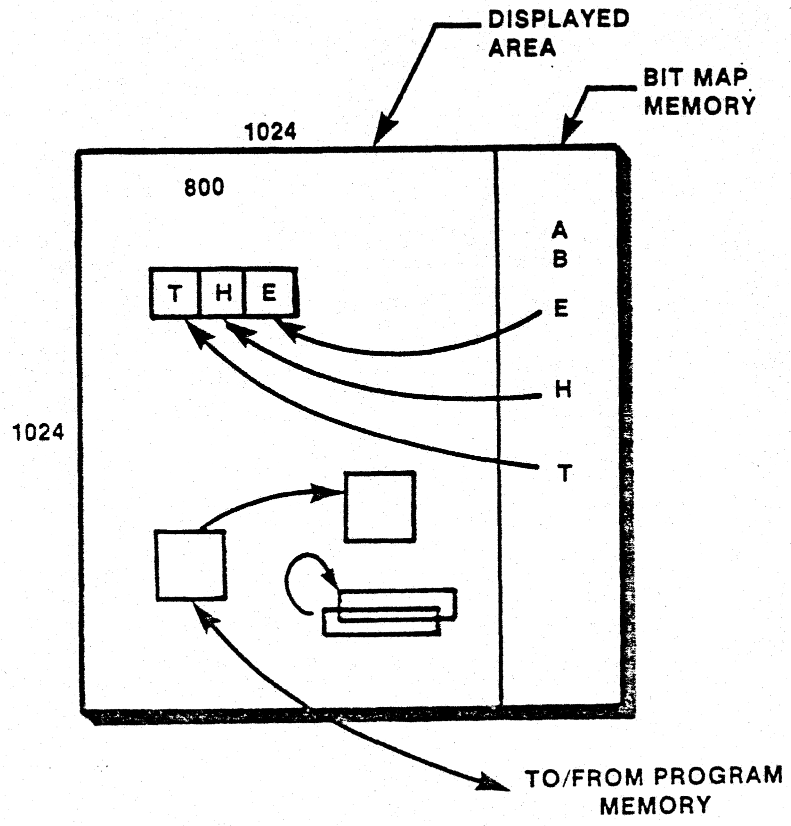
The internal Apollo node organization is comprised of several key parts. First, there is the central processing unit comprised of multiple Motorola 68000's. This central processing unit is connected to a memory management unit which translates the 24 bit virtual address out of the CPU into a 22 bit physical address on the physical memory bus. The memory management unit is actually comprised of two parts: one for the CPU and another part for the I/O system which will be described later. The memory system is comprised of multiple units - each unit containing a ¼ megabyte. This unit is fully protected with error correction codes and is available in sizes up to 1 megabyte. The I/O system of the Apollo node is broken down into two parts. The first part is for those peripherals that are integral to the Apollo system, such as the integral Winchester disk and the integral network node controller. These devices are connected to a block multiplexor channel. Other peripherals, such as user supplied peripherals, line printers, magtapes and so on, are connected to the MULTIBUS controller.

The use of a block multiplexor channel through which all disk and network traffic goes represents an essential part of the Apollo system. The system was designed to specifically maximize the node-to-node responsiveness across the network. To do this we wanted to guarantee that there would be no superfluous buffering of packet messages as they left a transmitting process and entered a receiving process on another machine; and, secondly, we wanted the transfer of this packet to operate at near memory speeds. To accomplish this responsiveness we allow the network full (100%) bandwidth access to primary memory, disallowing all other block transfer devices, such as the Winchester disk. Consequently, the disk and the packet network actually share a common DMA channel into primary memory so that both of these devices can transfer at data rates of nearly 100% memory bandwidth. Occassionally, a disk transfer will overlap a network transfer requiring that either device make one additional revolution. But the system level performance consequences of this interference are negligible.

Finally, the display system is comprised of a separate autonomous 1/8 megabyte bit map memory which is organized into a square array of 1024 bits on each side. The display memory is constantly refreshed onto an 800 x 1024 bit map CRT. There is a separate bit mover which is capable of moving rectangles from one part of the display onto another part of the display at a data rate of 32 megabits per second.

Although the display memory and the program memory are in separate physical bus organizations, they actually share the same address space so that the CPU can instantaneously access display memory and alter its contents. Furthermore, the bit mover can move display areas (rectangles) into and out of program memory. The system is designed so the CPU can access program memory and the display memory can refresh to the CRT display, and the bit mover can be moving rectangles all in parallel and without interference.

BIT MAP DISPLAY



- BIT ALIGNED /BIT RESOLUTION RECTANGLES
- DISPLAY MEMORY

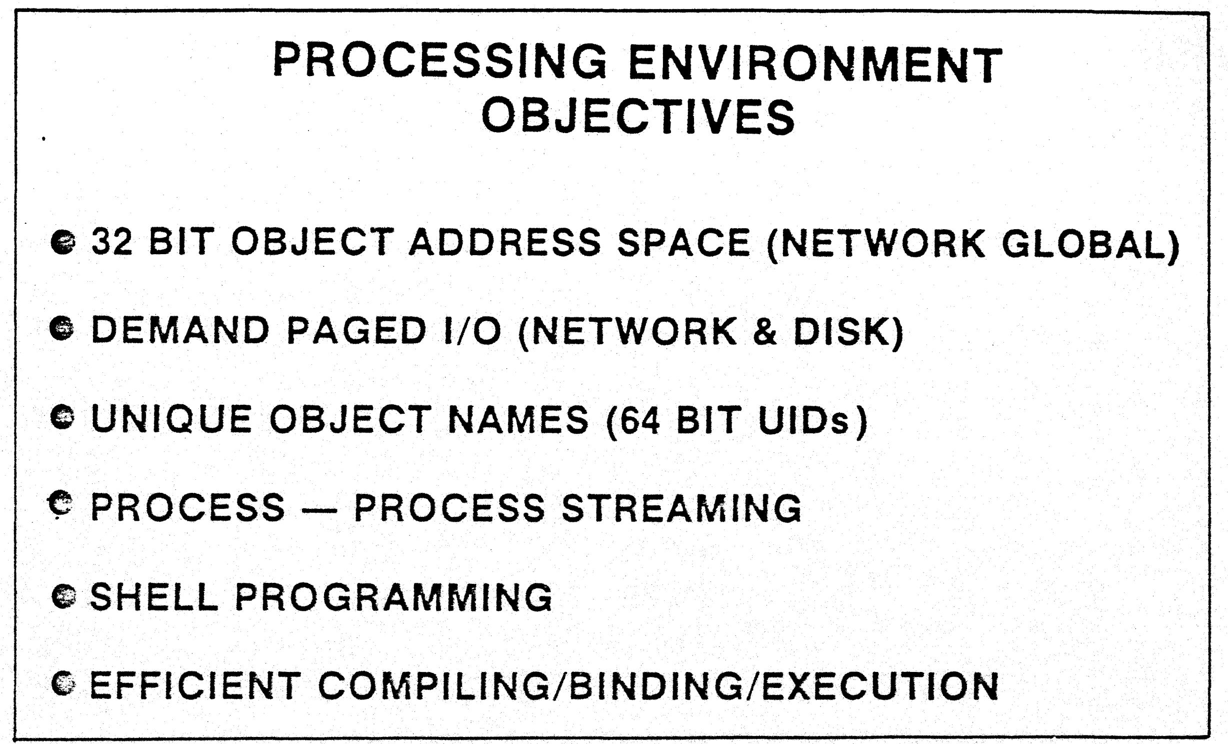
$$\begin{tikzcd}
\text{DISPLAY MEMORY} \arrow[loop left] \arrow[r, shift left] & \text{PROGRAM MEMORY} \arrow[l, shift left] \arrow[loop right]
\end{tikzcd}$$

- 32 MBITS/SEC

II.6 BIT MAP DISPLAY:

The bit map display system is comprised of a 1024 bit by 1024 bit array. A rectangular region of 800 by 1024 is physically transferred onto the CRT display. The remaining area is used as temporary storage for character font tables. The bit mover is a hardware primitive which is capable of moving a rectangular area from any place on the display to any other place on the display. This primitive is used to move windows into and out of main memory, to move them relative to the display itself, to implement scrolling and to create characte strings from character fonts. The bit mover operates at a 32 megabit per second data rate when moving entirel within the display memory.

The bit mover can move bit aligned rectangles fro display memory to/from word aligned buffers in progra memory where the CPU can efficiently perform rast operations, such as exclusive ORing two or more graphi representations.



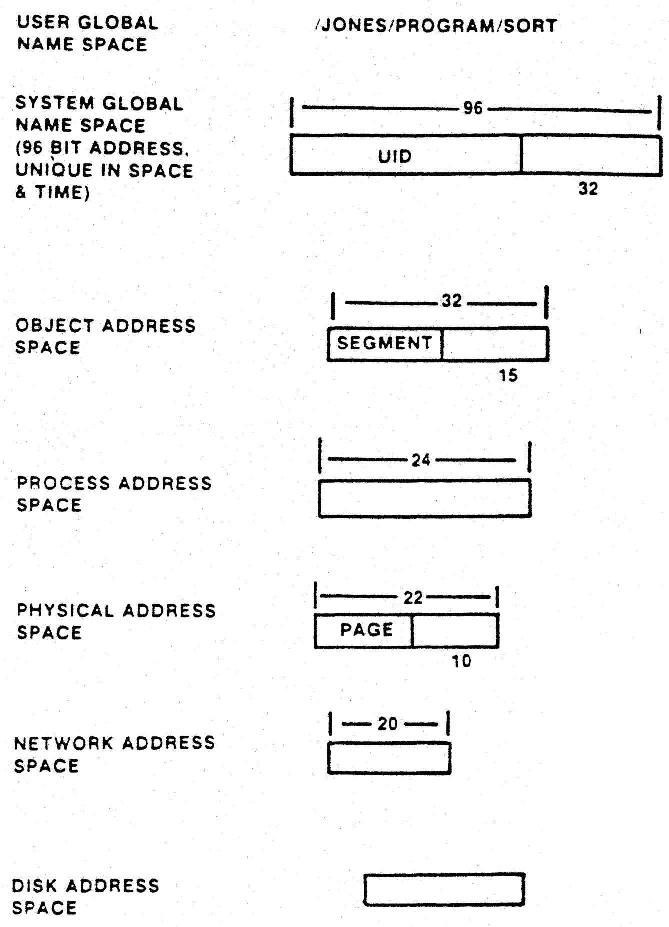
PROCESSING ENVIRONMENT OBJECTIVES

- 32 BIT OBJECT ADDRESS SPACE (NETWORK GLOBAL)
- DEMAND PAGED I/O (NETWORK & DISK)
- UNIQUE OBJECT NAMES (64 BIT UIDs)
- PROCESS — PROCESS STREAMING
- SHELL PROGRAMMING
- EFFICIENT COMPILING/BINDING/EXECUTION

III.1 PROCESSING ENVIRONMENT OBJECTIVES:

A principal objective in designing a system processing environment was to abstract common entities, like programs and data files, into a uniform abstraction which we call an object. The totality of objects across a network forms a 96 bit virtual address space which is comprised of two fields: a unique object name consisting of 64 bits, and a 32 bit byte address within an object. A second objective was to provide a demand paged operating system to implement a network wide virtual memory. A third objective was to provide an environment for efficient process to process streaming and the control of this streaming through shell programs. Finally, an efficient compiler, binding and execution procedure whereby network wide programs can be run interactively.

SYSTEM NAME SPACE



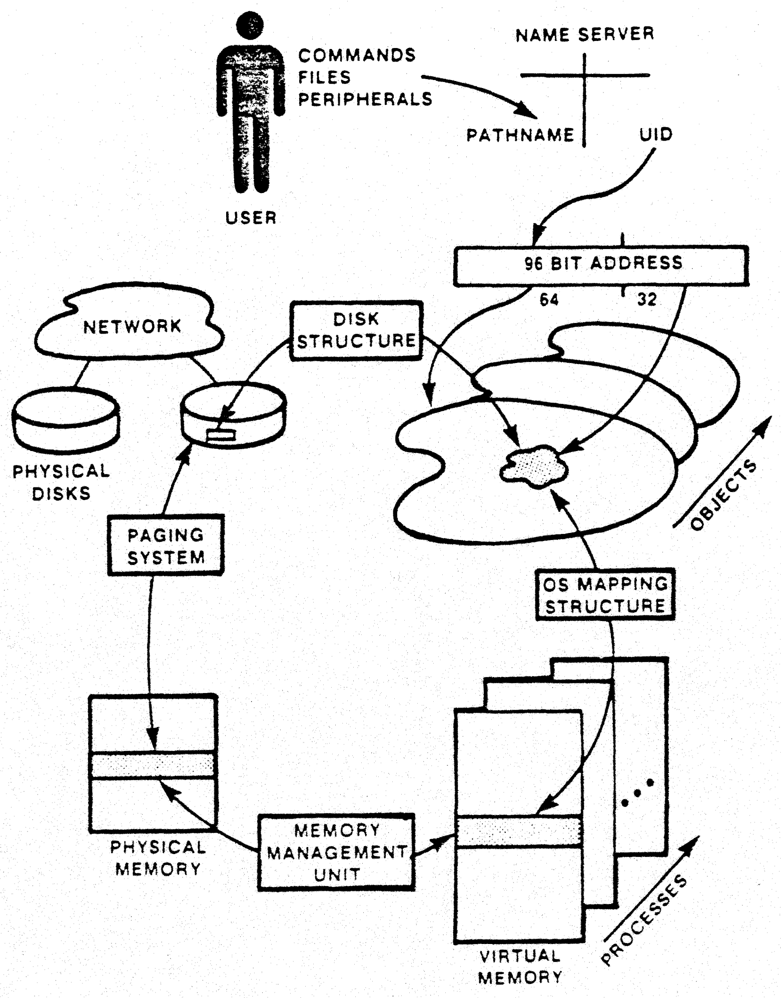
III.2 SYSTEM NAME SPACES:

We now turn to the operating system design in the **Apollo DOMAIN** system. One way of viewing a complex system is to enumerate and describe the various name spaces that occur in the system. First, there is the user global namespace, or what the user would normally type at a terminal to execute a program or access a data file. Second, there is the system global namespace, or the namespace that the operating system uses at a network level. Third, there is an object address space, which is 32 bits long and contains programs and files as well as other entities in the operating system which will be described later. Fourth, there is a process virtual address space that represents an address space in which a Motorola 68000 process executes. Fifth, there is the physical address space which represents the amount of physical memory that can be placed on the system. Sixth, there is the network address space or the maximum number of nodes that can be placed on the network. And, finally, there is the disk address space or the maximum bytes or pages that the disk can hold.

In the Apollo system the user global namespace is syntactically represented as a stream of characters separated by slashes. This actually represents a hierarchical tree space which will be described later. The system global namespace is a 96 bit address space comprised of a unique ID (UID) which is 64 bits and a offset which is 32 bits wide. The 64 bit UID is unique i space and time. It is unique in space in that it includes a encoding of the machine's serial number and it is uniqu in time in the sense that it includes the time at which th name was created. This guarantees that for all time in th future and for all machines that Apollo builds, no tw machines will ever create the same UID, hence the term unique ID.

UID's are names of objects. Objects are used to hol programs, files and various other entities in the Apoll system. An object is a linear 32 bit address space, byt addressable, and can be located generally any place o the network. Objects are the primary focus for the **Apoll DOMAIN** system and are cached into the process addres space provided by the Motorola 68000. This process address space, while very large, is still considerabl smaller than the 32 bit object address space. Consequently, address regions of an object are mapped into regions of a process in much the same way tha regions of physical memory are frequently mapped int regions of a cached memory. The process address spac is a 24 bit virtual address which is converted to a 22 bi physical address by memory management hardware. The unit of allocation in the physical address space is 102 byte pages.

SYSTEM RELATIONSHIPS

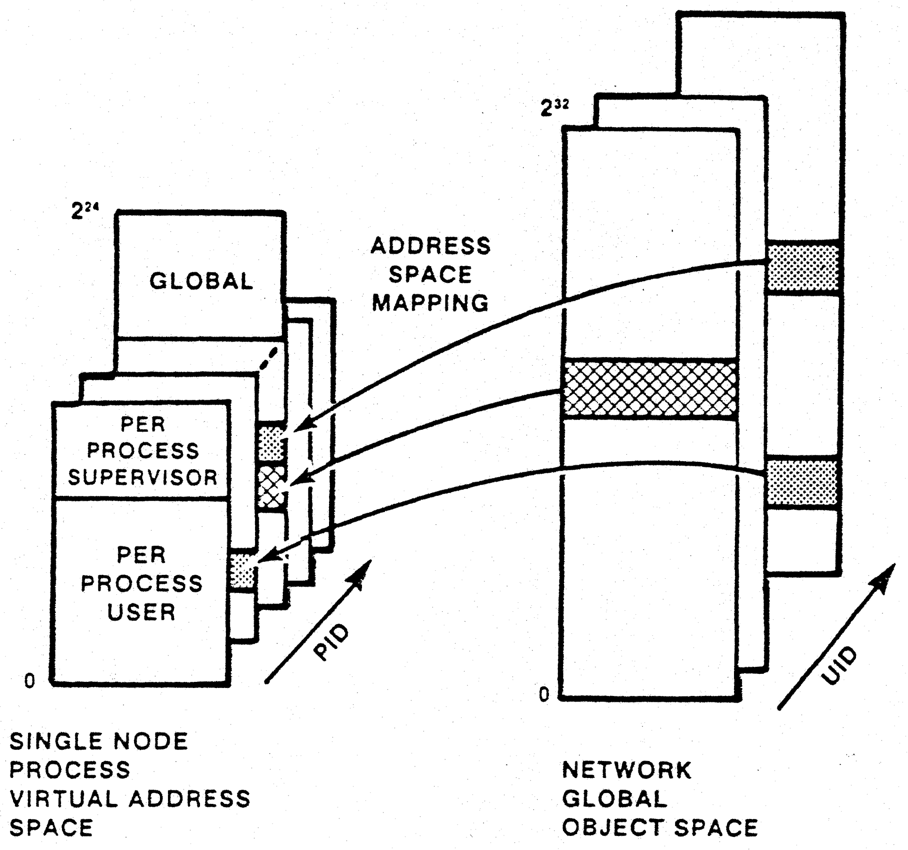


III.3 SYSTEM RELATIONSHIPS:

The execution of a user command on the **Apollo DOMAIN** system is a very complex process and involves many steps. First of all the user types a command which is translated by the naming server into a UID. The UID is a 64 bit address which identifies one particular object on the network. These objects then are dynamically mapped by the operating system into a processes virtual memory. Once mapped no data is transferred until the CPU actually requests it. When a page fault occurs the operating system will retrieve the requested page from some disk structure across the network and transfer it into the physical memory of the local processor. It will then set up the memory management unit to translate the virtual address into the physical address of the requested page and then allow processing to continue.

In this scenario we have four areas which are of interest. First is the operating system mapping structure, which maps object address spaces into process address spaces. Second is the memory management hardware which translates process virtual address spaces into physical memory address spaces. Third is the paging system which transfers pages of physical memory into and out of the memory system onto either local disk or across the network to some remote disk. And, fourth, is the disk structure that physically relates objects onto disk data blocks. These circular relationships are dynamically and under system control managed by the Apollo operating system.

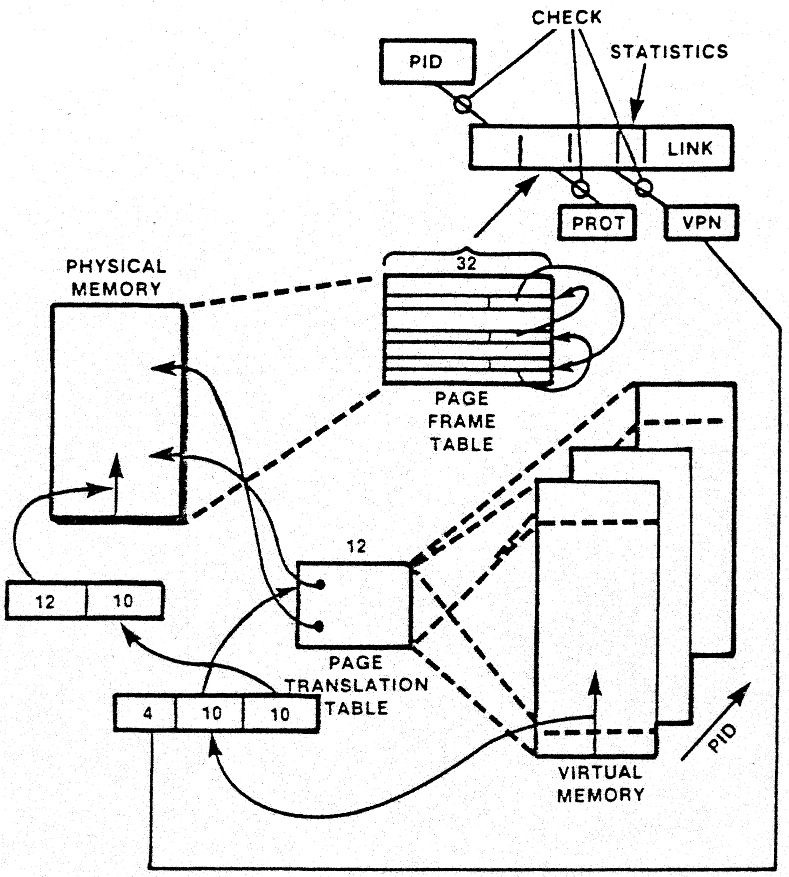
OPERATING SYSTEM MAPPING



III.4 OPERATING SYSTEM MAPPING:

The network global object spaces are mapped selectively into a process virtual address space of a particular node. Once the mapping occurs no data is transferred until the processor actually requests it. Consequently, the mapping of a large address space from an object into a large region of a process is a relatively inexpensive procedure. The objects, of course, are network wide; whereas, the processes are all in a particular node running on behalf of a particular user. The process address space is subdivided into an area which is glob to all processes and then further divided into an are which is per process supervisor and per process use This address space mapping represents the on primitive in which processes can relate to objects. For th most part the operating system and all higher level view of the system relate to objects rather than processes, a consequently a great deal of network transparency attained.

MEMORY MANAGEMENT UNIT



III.5 MEMORY MANAGEMENT UNIT:

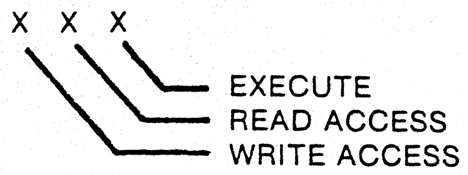
The memory management unit (MMU) is a piece of hardware which translates the 24 bit virtual address spaces out of the Motorola 68000 CPU onto the 22 bit physical address in the Apollo node. The MMU works on 1024 byte physical page sizes and has separate protection and statistics information for each page. There exists a separate entry in a page frame table for each individual page so that when the hardware faults out of the page frame table (i.e. cannot find an appropriate requested page), an interrupt is taken to move the requested page in from secondary storage. The MMU is actually a two level hierarchy, the page frame table being at the highest level . A lower level cache, called the page translation table contains the most recently used pages and acts as a speed up mechanism to search the page frame table.

The translation of a virtual address into a physical address proceeds roughly as follows. The 24 bit virtual address is broken down into three fields: (1) a high order virtual page number, (2) a page number, and (3) a byte offset within the page. The 10 bit page number is used as an index into the page translation table. The page translation table contains a 12 bit pointer which points directly to the physical requested page. Concurrent to the memory system beginning a memory request, this 12 bit pointer is also used to index into the page frame table from which the high order virtual page numbers are checked. If the check is okay, the protection is allowed and the process ID agrees, then the memory reference proceeds uninterrupted. If, however, there is no agreement on any of these accounts, the memory request is aborted and a search is made in the page frame table for all entries corresponding to this particular value of page number. All possible values for this page number are linked together in a circular list and the hardware automatically searches for the requested page number until: (1) it finds it and continues, or (2) does not find it and causes a CPU interrupt. If the requesting page is found in the page frame table, the location within the page frame table is updated to the page translation table so that subsequent references can proceed without researching the page frame table.

MEMORY MANAGEMENT UNIT PROTECTION/STATISTICS

PROTECTION HARDWARE(PER PAGE)

LEVEL	RIGHTS AT THIS LEVEL AND HIGHER
00 USER DOMAIN 0	
01 USER DOMAIN 1	
10 SPVR DOMAIN 0	
11 SPVR DOMAIN 1	



STATISTICS (PER PAGE)

X - ACCESSED
X - MODIFIED

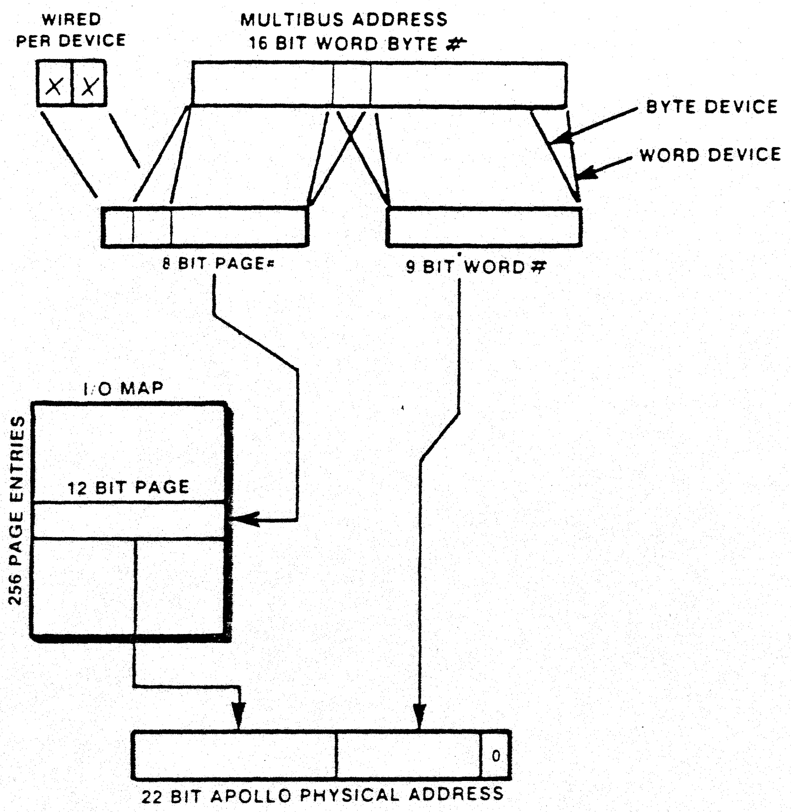
(USED BY PAGE REPLACEMENT LOGIC)

III.6 MEMORY MANAGEMENT UNIT - PROTECTION/STATISTICS:

At each access to a page a set of rights (execute, read, write) are checked as a function of a particular level that the process is running at. The protection hardware specifies the particular rights at this level and all higher levels. The levels are two supervisor levels and two user levels.

The memory management hardware automatically records and maintains certain statistics about the page access. In particular a bit is set every time a page is accessed and a second bit is set when that page is modified. The operating system nucleus scans these bits periodically to maintain knowledge of the statistical usage of the pages for the purpose of page replacement.

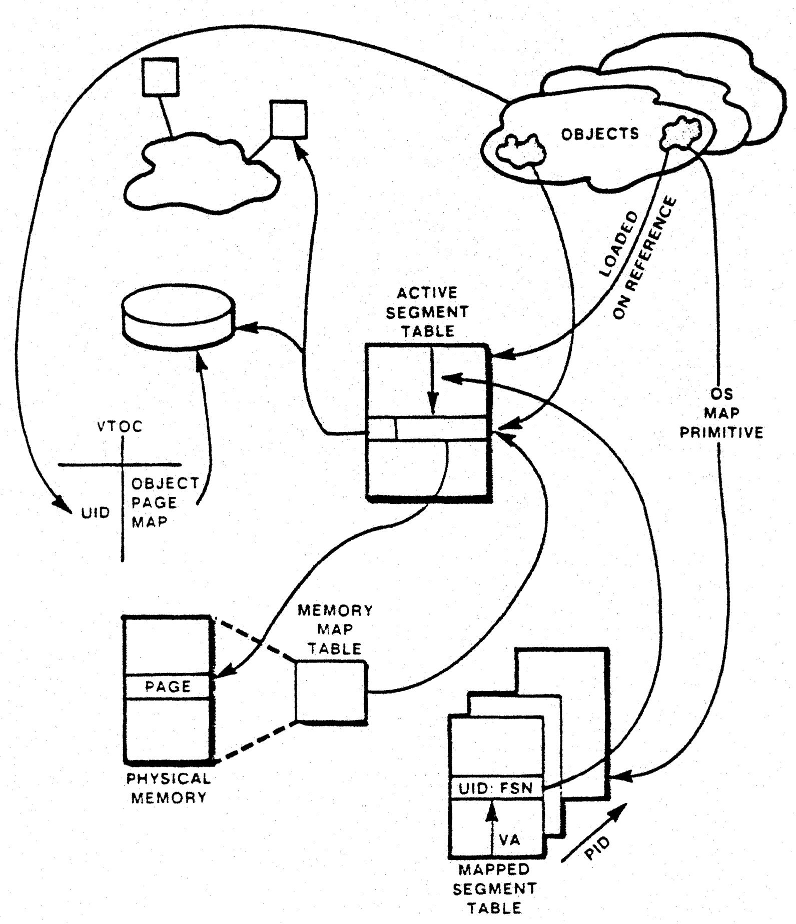
MEMORY MANAGEMENT
UNIT—I/O MAPPING



III.7 MEMORY MANAGEMENT UNIT - I/O MAPPING:

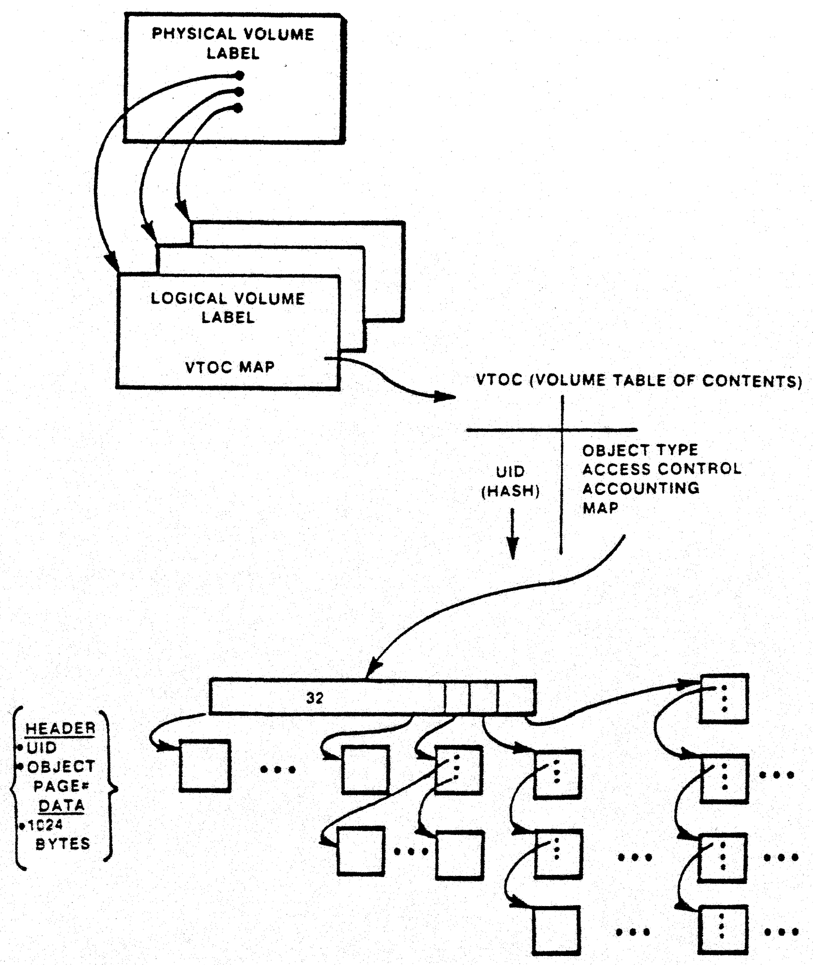
Peripherals on the MULTIBUS are mapped into the 22 bit Apollo physical address bus by means of an I/O map. The I/O map consists of 256 page entries, each entry pointing to a particular Apollo page. A peripheral on the MULTIBUS can generate a 16 bit word or byte address and have the high order bits indexed into the page map and the low order bits indexed relative to the page. In this way MULTIBUS peripherals can directly address themselves into the virtual memory of a process.

PAGING SYSTEM



III.8 PAGING SYSTEM:

To implement the network wide virtual memory system, several tables are maintained within the operating system nucleus. As objects are mapped into process address spaces, entries are made into the mapped segment table (MST). When a CPU fault occurs for that virtual address, the operating system scans the active segment table (AST). This table contains a cache of pointers to the actual location of the pages, be they in physical memory, on local disk or on a remote network node. In this way, objects that are logically mapped into a process are being constantly swapped in and out of memory across the network solely on a demand basis.



III.9 DISK STRUCTURE:

Objects are mapped onto physical disks using a rather dynamic storage allocation. First of all a disk structure contains a physical volume label which is a list of pointers which point to multiple logical volume labels. The division of a physical volume into multiple logical volumes is a means whereby fixed partitions can be created which do not compete for common storage. In other words, one can create a logical volume and guarantee it has a certain minimum amount of allocation.

Each logical volume label contains a volume table of contents map. The volume table of contents is a list of all of the object UID's in that volume and for each object a set of object attributes. The object attributes consist of the object type, access control information, accounting information (last date accessed, last date modified), and a map to all of the various data blocks which comprise the object. The map is comprised of 35 pointers. The first 32 pointers point directly to data blocks each of which consists of a single page. The 33rd pointer points to a block of second level pointers (256 of them) which in turn point to actual data blocks. The 34th pointer expands into three levels of storage and the 35th pointer expands into four levels of storage. Consequently, for small objects data access is very efficient; and for large objects storage allocation is very efficient.

Each block contains not only 1024 bytes of data, but also the UID and object page number that this page represents. Consequently if a failure should occur, the entire mapping structure can be recreated by a single pass over all of the data pages.

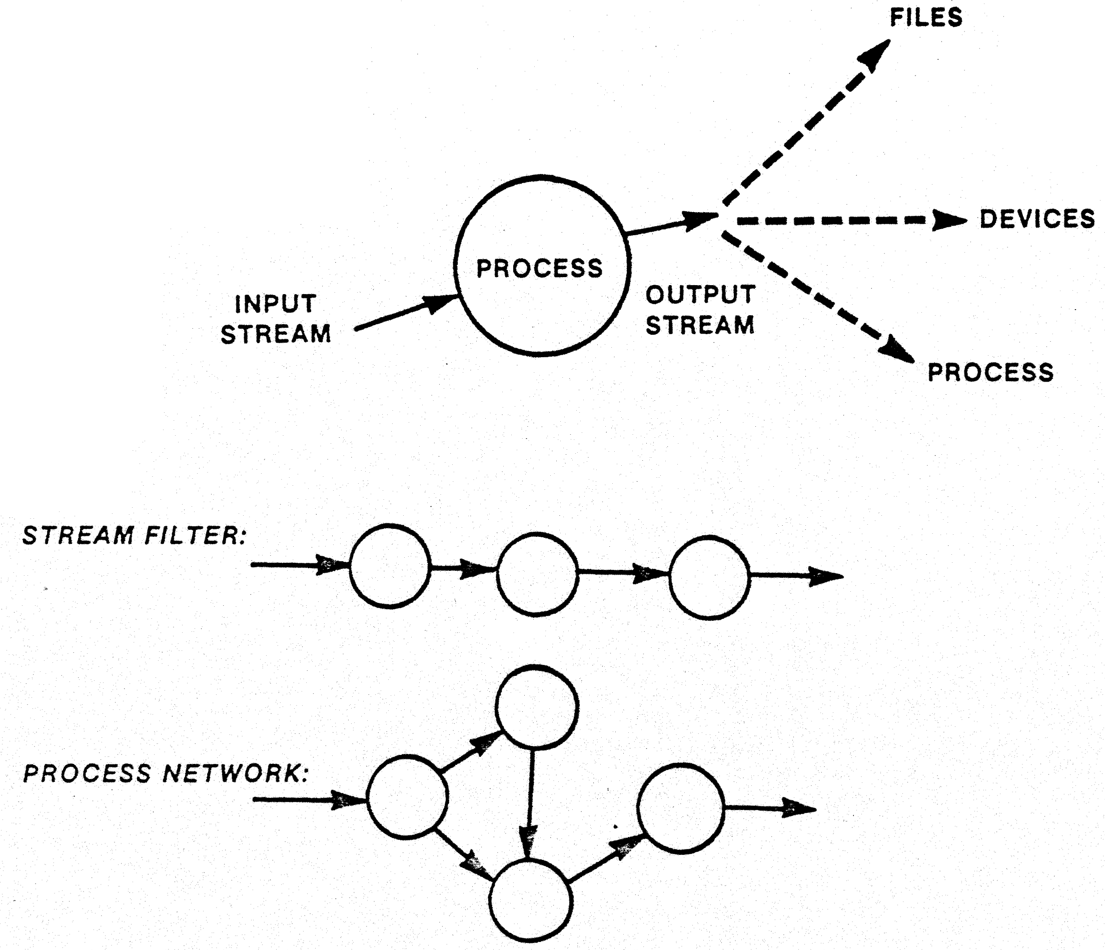
I/O HIERARCHY

LANGUAGE I/O	INDUSTRY COMPATIBLE, SYSTEM INDEPENDENT.
	OBJECT TYPE INDEPENDENT, PROCESS-PROCESS, FILE, DEVICE, ETC.
STREAM I/O	
	OBJECT LOCATION (NETWORK WIDE) INDEPENDENT. ASSOCIATES OBJECT - PROCESS ADDRESSING ONLY, NO DATA TRANSFERRED UNTIL REFERENCE IS MADE.
MAPPED I/O	
	PHYSICAL I/O TO LOCAL AND REMOTE DISKS ACROSS NETWORK. DATA TRANSFERRED "ON DEMAND," RESULTING FROM CPU PAGE FAULT.
PAGE I/O	

III.10 I/O HIERARCHY:

There are four levels of abstraction in the I/O system of the **Apollo DOMAIN.** The highest level is the language level which is supported by the standard language compilers, such as Fortran read and write. The implementation of this language level is done by what we call the stream level. The stream level has the characteristic of being object type independent and can accordingly talk to files, peripheral devices, or to other processes. The implementation of the stream level is accomplished through the map primitives which were described earlier. The map primitives have the characteristic of being object location independent thereby allowing streams to go across the network. The mapped primitive associates object to process addressing only. No data is transferred until the reference is made. All data transfer in the entire system occurs at the page level. The page level is the physical I/O to local and remote disks across the network. This data is transferred on demand, resulting exclusively from a CPU page fault.

STREAM I/O



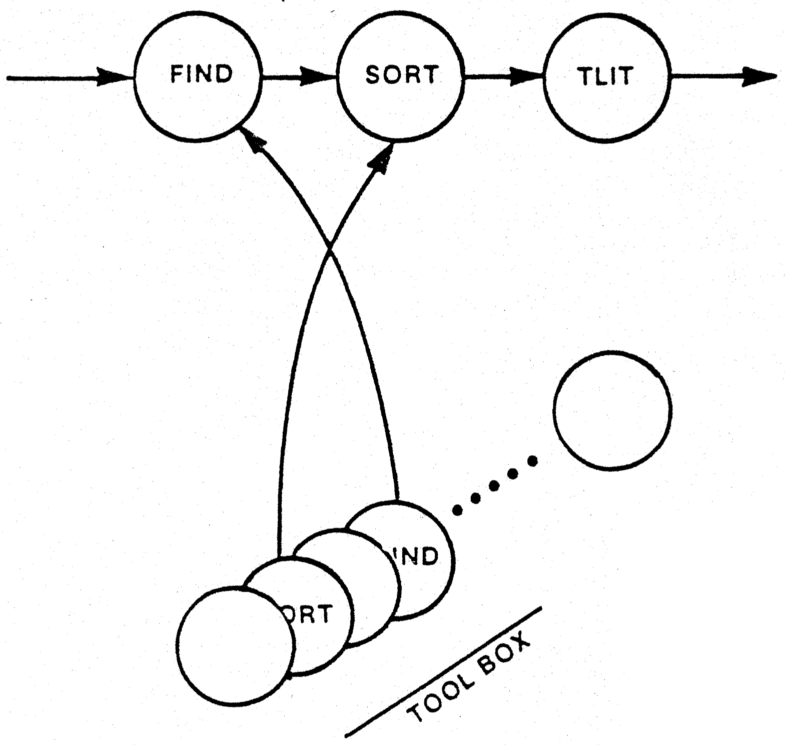
III.11 STREAM I/O:

The stream I/O level deals with the interconnection of objects, including process to file operations, and process to process operations. It has the principal characteristic of being object type independent. Since it is implemented through the mapped I/O level, objects can be conceptually interconnected by streams both within the same node and across the network.

When streams are used to interconnect processes, the output of one process is connected to the input of another process. This multiple process application can acquire the form of a stream filter whereby every process forms some transformation on its input and then passes the output to another process. When applications are encoded in this manner, programmers are encouraged to write processes as simple, modular programs that perform some primitive function. Frequently, these functions can be reused across many applications.

SOFTWARE TOOLS

APPLICATION:

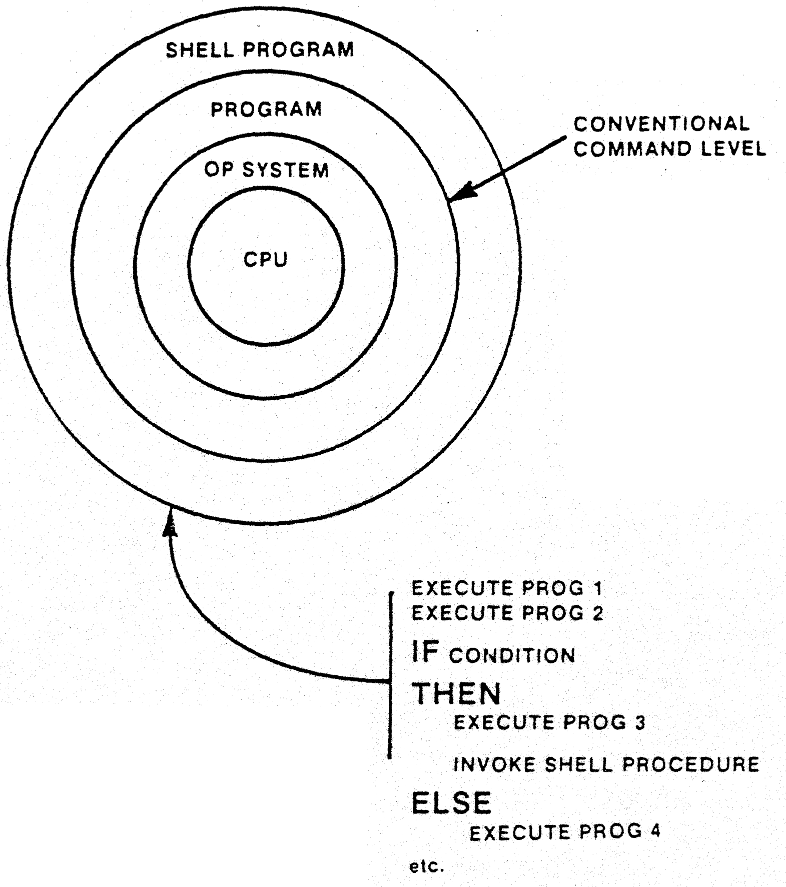


III.12 SOFTWARE TOOLS:

A large collection of program modules designed to perform some primitive function has evolved over years of use by a large collection of users. These modules are referred to as Software Tools and are widely distributed throughout the user community. Software Tools follows the methodology laid out in the book entitled "Software Tools" by Kernigan and Plauger, published by Addison Wesley.

Applications can be easily formed by interconnecting streams of data through a collection of Software Tools. The collection of standard Software Tools is derived from a library of programs - a "toolbox" of Software Tools. In this way complex applications can frequently be formed with little or no programming. The time required to develop a new application is significantly reduced. Furthermore, users are encouraged to write programs that are small, conceptually simple, and usable for many applications and by many users.

SHELL PROGRAMS

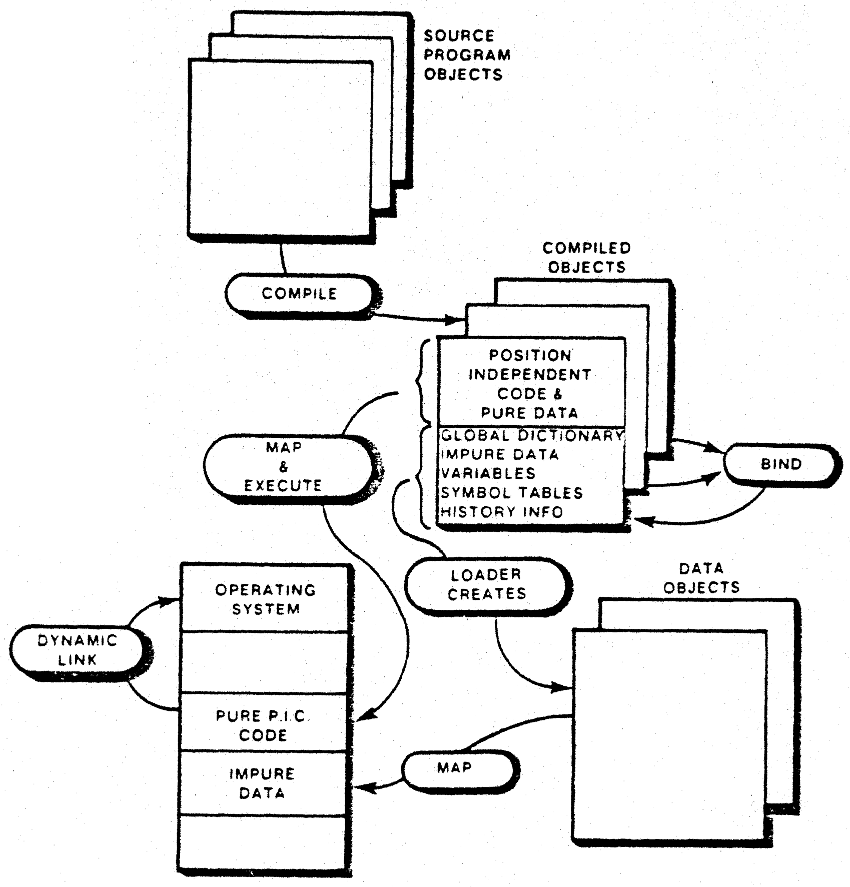


III.13 SHELL PROGRAMS:

A shell program is a higher level flow of control above the conventional program level (e.g. Fortran or Pascal). Shell programs are written in a shell programming language that has a rich set of constructs that are, in many respects, similar to a conventional language. However, an executable statement within a shell program frequently involves the complete execution of one or more conventional programs. In this regard, a shell program can be thought of as a sophisticated command processor which coordinates the execution of multiple program steps.

The ability of users to program applications in a shell programming language relieves a great deal of complexity that would otherwise be required within a Fortran or Pascal program. Consequently, programs written in these languages tend to be simpler and have fewer input options.

The concept of shell programming goes hand-in-hand with the concept of Software Tools. Here, the shell programs represent the interconnect of streams between various programs, and can be extended to richly interconnect small programs in order to form complex applications.



III.14 COMPILATION/BINDING/EXECUTION:

We now shift to the higher level organization of objects in the system as they relate to user programs, compilers, binders and loaders.

The compiler translates a source program object into a compiled object. The compiled object has a format which is suitable for direct execution if there are no unresolved references (i.e., no other subroutines which need to be bound together). If the application contained several source program objects, these compiled objects must be bound together prior to execution, a process accomplished by the BINDER. The process of loading and executing a compiled object consists of: (1) mapping the pure position independent code into a region of a process address space, (2) creating an impure data object and mapping that data object into an impure section of the process address space, and (3) dynamically linking operating system references to the operating system during execution.

There are three important points in this procedure: (1) The output of a compiler can be directly executed if there are no external references to be resolved. (2) A runnable object, once formed, is paged into memory at run time, on demand. (3) Source program objects, compiled objects, and bound objects can be resident anywhere on the network.

The compiled object format is comprised of two parts. The first major part is position independent code and pure data which is directly mapped and executed into a process address space. The second part is a database used by the loader to create an impure temporary data object which is subsequently mapped into the impure part of a process address space.

USER ENVIRONMENT OBJECTIVES

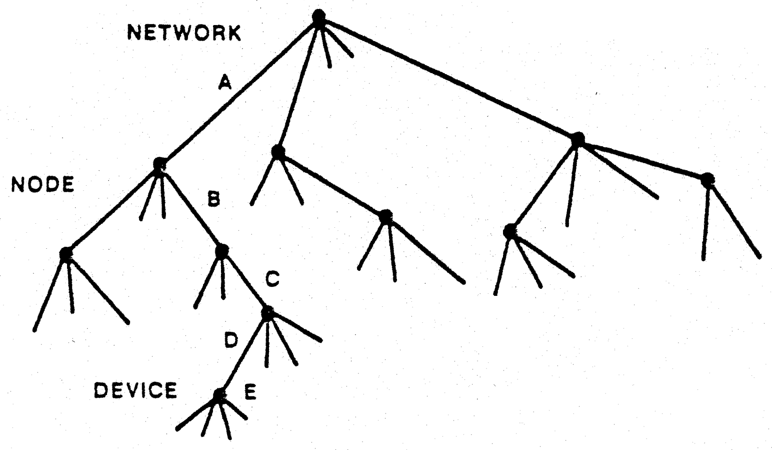
- UNIFORM NAME SPACE
- BIT MAP DISPLAY (TEXT, GRAPHICS)
- CONCURRENT PROCESSING PER USER

IV.1 USER ENVIRONMENT OBJECTIVES:

A key objective in designing the Apollo user environment is to combine simplicity and uniformity with a high degree of functionality.

All objects that the system is capable of referencing can be expressed in a uniform name space that transcends the entire network. Further, a bit map display, as opposed to a character display, is used to represent text and graphics output. The output from multiple programs can be concurrently displayed through multiple windows, thereby providing a degree of functionality unavailable on conventional systems.

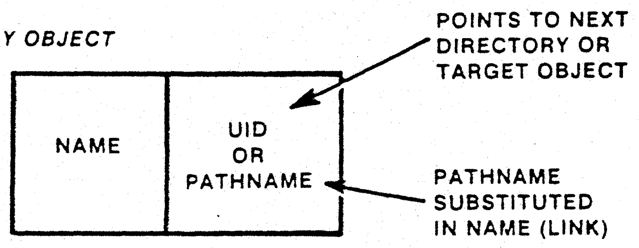
USER NAME SPACE



SYNTAX

//A/B/C...NETWORK WIDE
/B/C/D...LOCAL ROOT RELATIVE
C/D/E...WORKING DIRECTORY RELATIVE
@F/G "@"≡"/user/name_dir/"

DIRECTORY OBJECT

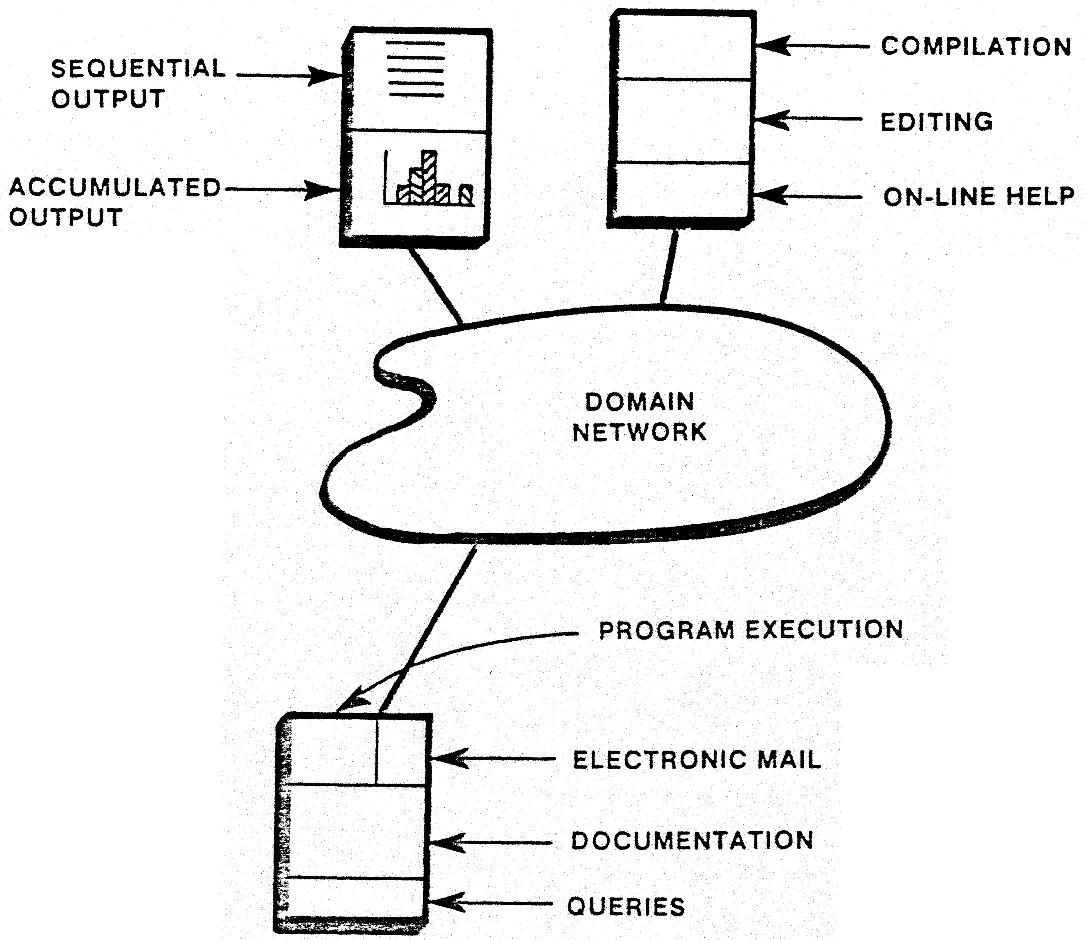


POINTS TO NEXT DIRECTORY OR TARGET OBJECT
NAME
UID OR PATHNAME
PATHNAME SUBSTITUTED IN NAME (LINK)

IV.2 USER NAME SPACE:

The namespace seen by a user is organized as a hierarchical tree structure. The highest node of the network in the tree represents the most global portions of the network. Whereas, the leaves at the bottom of the tree represent particular objects, such as programs, files and devices. Intermediate nodes are used to represent collection of objects that have some common association. For example, an entire node on the network may be represented by an entire subtree in the tree hierarchy. The overall namespace hierarchy is intended to represent a logical organization of the network. All leaves, or the lowest level of the tree, represent objects and the user has a variety of syntactical forms in which to express the location of an object. First of all there is the network wide syntax which is comprised of two leading slashes followed by a full path name to reach the object. Second, there is the local root relative syntax which can be used to express objects that are local to a particular user's node. Syntactically this is expressed by one leading slash followed by a relative path name. For convenience, the user may attach himself or his working directory to any point in the tree name hierarchy; and, consequently, he may express a path name which is relative to his working directory. He does this by expressing the relative path name without a leading slash. Each node in the network is represented as a directory object and contains a list of associations. For each name at a lower level there is contained within the directory a UID or a path name. If it is a UID it points to the next lower level directory or to the object itself. If it is a path name, the path name is syntactically substituted into the name being searched and the search continues. This latter path name is used for linking names across the network.

CONCURRENT USER ENVIRONMENT

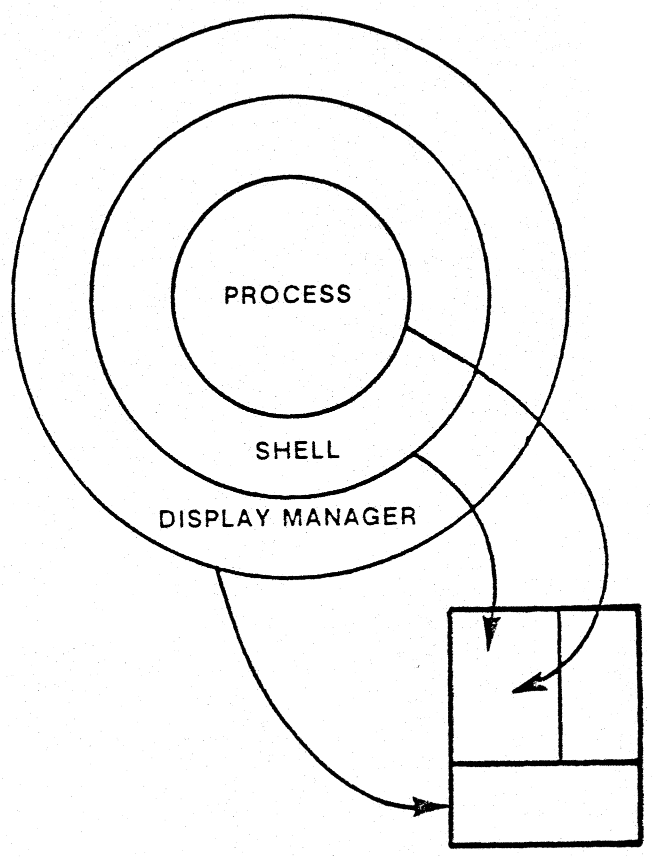


IV.3 CONCURRENT USER ENVIRONMENT:

The notion of concurrency is a new concept on the **Apollo DOMAIN** system unavailable on conventional timesharing systems. On these latter systems users are generally required to execute one function at a time. When a user switches from one function to another, generally the context of the previous function is lost and has to be subsequently recreated. The Apollo integral bit map display provides the user with the capability of displaying multiple windows simultaneously. Each window can contain the output of related or unrelated applications. For example, one window can contain the sequential output of a program while a second window graphically displays the accumulated output of the same program. Similarly, program development, compilation, editing and an on-line help system can all be concurrently displayed.

Consequently, the Apollo system is designed to accommodate a total user environment, which we believe always involves a number of concurrent functions.

DISPLAY MANAGER



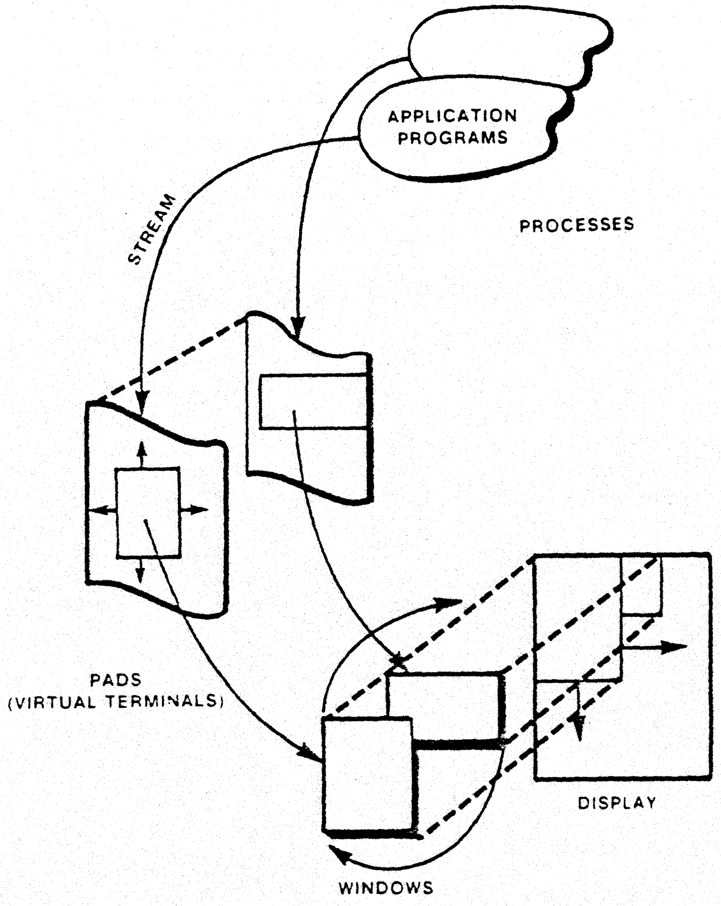
IV.4 DISPLAY MANAGER:

The display manager represents the outer most layer of logic within the Apollo system -- that which controls the relationship among the many windows projected onto the CRT display. Accordingly, the Apollo system adds two additional layers above the conventional programming level. As mentioned earlier, a programmable shell coordinates the activity of many programs (in both parallel and sequential relationships). The output of this shell is written into a virtual terminal, called a PAD. Portions of this PAD are displayed through a rectangular window which is then projected onto the CRT display.

The display manager permits multiple windows to be displayed concurrently, each of which can be executing an independent shell or command environment. The philosophy of the display manager is to allow programs to output data in a logical format, while allowing the user to independently control what is physically displayed.

The display manager is controlled by the use of function keys on the user keyboard. Pushing a function key causes the execution (interpretation) of a user programmable sequence of display manager primitives. Consequently, the user can define function keys to perform complex display manager functions.

USER ENVIRONMENT



IV.5 USER ENVIRONMENT:

The **Apollo DOMAIN** operating system creates a degree of independence between application programs and what is actually viewed on the terminal display. In particular, application programs create virtual terminals which we call pads. The pads are independently windowed onto the CRT display totally under user control. Window images are superimposed on the pads and can be moved relative to the pad in either a horizontal or a vertical direction. Window images from various pads are stacked logically on top of the display so that only the one on top is displayed. Consequently the user environment is actually a three dimensional volume: 800 bits going across, 1024 bits going down and many levels of windows deep. The user can also move window areas up or down relative to the physical display and finally can move window areas into and out of the display relative to other window areas.

Programs create the pad by writing command and data sequences through a stream. The window image created by the display manager from the pad can be placed anywhere in the CRT and can be overlayed by other window images. Window images contain lines and frames. A line is a single line sequence of characters and has only one dimension. A frame has two dimensions and has a rectangular format. It contains characters and/or graphic data. Finally, frames may also contain user created bit maps. These bit maps may reside either within the pad or within a separate user supplied object. Pad information normally accumulates over the life of a process. This allows a user to scroll either in reverse or in forward directions over the entire life of the process. However, for efficiency sake certain commands may be emitted from the program to delete all or part of the pad as appropriate.

SUMMARY OF KEY POINTS

- HIGH PERFORMANCE LOCAL NETWORK OF DEDICATED COMPUTERS
- LARGE MACHINE ARCHITECTURE
- LARGE MACHINE LANGUAGES/COMPILERS
- OBJECT ORIENTED NETWORK OPERATING SYSTEM
- ADVANCED USER INTERFACE COMBINING TEXT, GRAPHICS, CONCURRENT PROCESSING
- ADVANCED **VLSI**, WINCHESTER, COMMUNICATIONS TECHNOLOGIES

V.1 SUMMARY OF KEY POINTS:

An Apollo computer system is comprised of a number of high performance dedicated computers interconnected over a local area network. Each of these nodes contains a large machine architecture which implements a demand paged network wide virtual memory system, allowing a large number of processes for each user, each process having a very large linear virtual address space. Languages that run on the Apollo system include Fortran 77 and Pascal and are implemented to take advantage of the machine's 32 bit orientation.

An object oriented network operating system coordinates the user's access to network wide facilities. Objects themselves, representing programs and data files, etc., are independent of their network location, and given appropriate access rights, can be accessed uniformly by anyone on the system.

The user's display terminal is capable of displaying multi-font text, graphics and can be divided into multiple windows each displaying independent program output.

The Apollo system is designed around high technology. It incorporates VLSI CPU chips, a large capacity Winchester disk, and advanced communication technologies.