Database Machine Morphology[1]

Haran Boral
Steve Redfield

Microelectronics and Computer Technology Corporation
9430 Research Blvd.
Austin, Texas 78759

Abstract

In this paper we classify and analyze approximately twenty database machines (DBMs). We show that (i) research has generally been focused on brute force parallel search methods, (ii) little original work has been done on storage structures, (iii) almost no work has been done on database operating systems, and (iv) most designs are geared towards minimizing the response time of an individual operation, few designs attempt to minimize the response time of an individual query, and almost none are throughput oriented. We also show that the machine architecture of most DBMs can be described using a small number of building blocks.

1. Introduction

The MCC Database Program is studying means for enhancing database management systems to deal with "non-standard" applications, such as knowledge management. In addition, we are exploring the use of a DBM to implement a kernel of the DBMS functions. To provide some order to the assessment of lessons learned from past DBM designs, we undertook a study to classify and analyze DBMs. Our results have been mixed: on the positive side we were able to come up with a "language" for describing DBMs and through its use we have come up with an analysis leading to several important conclusions; on the negative side, the resulting grouping of DBMs into classes is not as robust as we had hoped it would be. In this report we present our language and some of the lessons that we have learned by applying it to approximately twenty DBM designs. Several rather negative conclusions may be drawn at this point. First, the designs we examined tend to emphasize the use of brute force parallelism. Second, little attention seems to have been given to the I/O bottleneck. Third, almost no attempt has been made to examine the effect and benefits of a specialized database operating system. Fourth, almost every design is optimized towards improving the response time of a single request rather than the throughput of the system. On the positive side, some of the more recent designs seem to be bucking these trends (in particular the first and second).

The remainder of this paper is organized as follows. In Section 2 we give an overview of the morphology. In Sections 3 and 4 we describe its two components. In Section 5 we discuss lessons learned from its application and relate our work to other classification efforts. Appendices A and C contain several detailed descriptions of DBMs using the morphology and Appendix B contains a BNF grammar for it.

2. Overview of the Morphology

Our morphology consists of two components: (1) a cataloguing component which is used to discriminate among DBMs based on seven macro characteristics, and (2) an anatomical component in which machine architectures are described systematically. This distinction was made because for the morphology to be useful it must be both simple and complete. Simplicity enables the user of the morphology to capture the essential features and mechanisms of a given DBM whereas completeness enables him to make meaningful comparisons. Simplicity is provided in the catalogue and completeness in the anatomy.

[1] *Morphology:* "... a study of structure or form ..." -- Webster's New Collegiate Dictionary.

Our starting point for the morphology is a simplified model of a DBM. All DBMs are said to be made up of a *Door*, a *Funnel*, and a *Home Repository*. The door and home repository elements of the model are used to describe the passive characteristics of a DBM -- the interface to it (door) and the ultimate location of the data (home repository), whereas the funnel is used to describe its active characteristics. Our choice of a funnel as an abstraction of the processing activity is based on the view that the accomplishment of a database request can be (and typically is) broken down to a series of steps which take place on some base data as it moves from the home repository to the user (assuming a retrieval operation). Each step causes irrelevant data to be filtered out and relevant data to be enhanced to what the user desires. Thus, there is a gradual refinement of the data starting from its "raw" form in the home repository and ending up in its "processed" form in the door. This model of a DBM is shown in Figure 1.

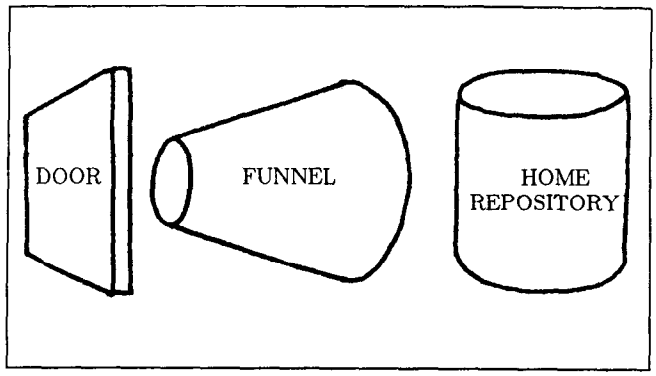


Figure 1 - Model

The catalogue places DBM designs into categories based on seven attributes which can be loosely associated with the three elements of the model. We believe that these seven attributes represent the most significant aspects of a DBM (excluding its machine organization) if one had to describe it succinctly.

While the purpose of the catalogue is to facilitate enumeration of designs, the anatomy enables comparisons among these designs. The anatomy must, thus, capture machine organizational details. Only an anatomy of funnels is given. Sufficient information about the door and home repository is provided in the catalogue. The anatomy is done from the perspective of interconnected subsystems each of which loosely corresponds to a single processing step. Each subsystem has an inner fine structure which is made up of basic hardware modules and their interconnects.

3. Catalogue

3.1. Scheme

What are the salient features of a DBM? We argue that these can be described succinctly using seven attributes that characterize, in a gross manner, each of the three components of the model shown in Figure 1.

The seven attributes we chose are:
- Mission (M)
- Number of Simultaneous Missions (NSM)
- Overlap Type (OT)
- Memory Property (MP)
- Processing Primitives (PP)
- Location Mechanism (LM)
- Storage Structures (SS)

M and NSM characterize the door. OT, MP, PP, and LM characterize the funnel. MP, LM, and SS characterize the home repository. Note that the MP and LM attributes can be reflected as funnel characteristics or as home repository characteristics.

We argue that these attributes enable a complete, albeit high-level, description of a DBM starting from what the machine does (the door characteristics), the basic mechanisms it uses to accomplish its mission (the funnel characteristics), and, not least, how the machine addresses the I/O bottleneck[2] (the home repository characteristics).

A cataloguing involves assigning specific values, or descriptors, to each of the seven attributes.

3.2. Terminology

In this section we describe each of the seven attributes in some detail and give several examples of descriptors that can be assigned to them.

M refers to the level of processing done in the DBM; or, equivalently, how much of the total database system job the DBM performs. Some of the descriptors possible for M are (assuming the relational model): *algebra tree, relational algebra operator,* and *access method.*

The algebra tree descriptor indicates that the DBM receives a tree of relational algebra operators to execute. To be given this classification it must work on

[2] See [4] for a discussion of the impact of the I/O bottleneck on DBM performance.

the tree as a whole and not on an operator at a time. The relational algebra operator descriptor indicates that the DBM works at the level of single operators. The access method descriptor indicates that the primary role of the DBM is to effect the access methods for a database system.

The **NSM** category indicates how many concurrent missions the DBM can be handling at one time instance. Some of the descriptors possible for NSM are: *single, few,* and *several.*

The single mission descriptor obviously indicates that the DBM only works on one mission at one time. We choose not to distinguish between machines that are dedicated to a single mission or those that use timesharing to attain pseudo concurrency -- both fall under the single mission descriptor. A DBM in this category may carry out its mission using some combination of parallelism or pipelining (see the overlap attribute below), but it only does one mission at a time. Some DBMs allow a limited degree of overlap between different missions because of an "assembly line" approach to processing. We distinguish between such machines (where the degree of simultaneity is fixed and typically small) -- the few missions descriptor, and those machines that allow several (in principle an arbitrary number of) missions -- the several missions descriptor.

The **OT** category indicates how much and what manner of concurrency is utilized in processing a *single* mission. A DBM that utilizes some form of overlap in the execution of a single mission must be capable of decomposing a mission into a number of tasks that can be executed in an overlapped manner. Some of the descriptors possible for OT are: *none, pipelined, parallel,* and *pipelined parallel.*

The no overlap descriptor indicates that the DBM has no concurrency with respect to a given mission. It may still have concurrency associated with more than one mission. The pipelined overlap descriptor indicates that the DBM treats a specific mission as a sequence of steps that can be chained linearly. An example is a hardware realization of a pipelined sorter. The parallel overlap descriptor indicates that the DBM divides the work or part of the work it has to do for a specific mission into a number of (possibly identical) activities it can do in parallel. The pipelined parallel overlap descriptor indicates that the DBM does both pipelined and parallel mission execution.

The **MP** category indicates whether the DBM does any staging of data from the home repository for the purpose of speeding the access to staged data. Some of the descriptors possible for MP are: *non staging* and *buffering.*

The non staging descriptor indicates that the DBM has no memory, but just acts as a intelligent conduit between the home repository and the user. The buffering descriptor indicates that the DBM will retain data received from the home repository. The buffered data may be held in a different and more convenient format from the home repository's format.

The **PP** category indicates the primary techniques used by the DBM to achieve its missions. We concentrate on the mission execution rather than on how the data is accessed here. Some of the descriptors possible for PP are: *sorting, hash partitioning, intersection, hash semi-joins, filter semi-joins,* and *nested loops.*

Each of these six descriptors indicates the basic primitive that is used to realize the processing primitives. Some DBMs use more than one of these. DBMs also differ in how they realize these primitives -- several use a software implementation on a general purpose computer and a few have special purpose devices that realize the operation directly in hardware.

The **LM** category indicates the primary technique used by the DBM to associatively access the database. The data may be located in the home repository, or if there is staging it will be in a buffer. Some of the descriptors possible for LM are: *filtering, indexing, indexing with filtering,* and *link based access.*

The filtering descriptor indicates that the DBM uses a scanning and comparison process to locate desired data. The indexing descriptor indicates that the DBM uses a directory or index to identify candidate data based on attribute or key values. The index may specify individual instances or (more typically) blocks which contain qualifying instances. The DBM may do some processing on the indices to further reduce the number of candidates before actually materializing the data. The indexing with filtering descriptor indicates that the DBM uses a combination of indexed and filtered access. The link based access descriptor indicates that the DBM uses pointers to locate desired data. The pointers may be associated with the data and represent interrelationships or they may be a separate meta structure which can be navigated before materializing the data.

The **SS** category indicates the primary storage structure utilized by the DBM for holding the database.

Together with the location mechanism, the storage structure attribute specifies how the DBM deals with the I/O bottleneck. Some of the descriptors possible for SS are: *unordered normalized relations (unr), single attribute clustered normalized relations (sacnr), multiattribute clustered normalized relations (macnr), unnormalized relations (ur), attribute,* and *domain.*

The normalized relations descriptors (unr, sacnr, and macnr) indicate that each relation is stored as a single unit. The items making up a single tuple in the relation are stored in the same place. (Some implementations store a tuple as a contiguous byte string, others associate attribute codes with each field, and others associate mark bits at either the individual field or tuple levels.) Relations may be unclustered, clustered on one, or several, attributes. The unnormalized relations descriptor indicates that each (unnormalized) relation is stored as a single unit. The attribute descriptor indicates that the primary storage structure is an attribute. Associated with this structure must be information to enable the reconstruction of tuples. This can be done explicitly using some meta structure, or implicitly by position (as in transposed files) or tid's. The domain descriptor indicates that the primary storage structure is a domain. All values in a given domain are stored in a single unit. Some additional information for associating values with their relations must exist.

3.3. Examples

Table 1 in Appendix A shows assignment of descriptors to the seven attributes for the twenty DBMs we examined.

4. Anatomy

4.1. Overview

The descriptions of DBM architectures available in the literature are typically so detailed that it is almost impossible to compare two or more DBMs. In this section we propose a "language" for describing the machine organization of the funnel component of DBMs. The language treats a funnel as: (1) a collection of modules, (2) the interconnects which link these modules, and (3) subsystems which are groupings of modules and their associated interconnects. Modules correspond to basic hardware units like sorters and memories. Subsystems correspond to basic units of work and basic units of flow. They indicate the manner of work assignment to the modules.

Modules and interconnects can be "typed" -- that is given a name based on the function they perform (modules) or on their structure (interconnects). We argue that subsystems can also be typed using two properties to be introduced below. This typing is captured in a configuration expression.

Our goal is to describe a funnel in a manner that on the one hand is detailed (by using modules and interconnects) and on the other hand enables "pigeonholing" designs into few categories for the purpose of comparison (using subsystems). Although some precision is lost at the subsystem level of description the advantage is the ability to compare "apples with apples" rather than "apples with oranges".

4.2. Modules and Interconnects

Nine types of modules have been found to cover the DBMs dissected so far. They are (in alphabetical order): *Arithmetic Processor, Buffer, Controller, Filter, General Processor, Hasher, Index Processor, Merger,* and *Sorter.*

Four types of interconnect cover the varieties found in the DBMs studied to date: *Point to Point, Bus, Ring,* and *Switch.*

4.3. Subsystems

We have identified two characteristics that distinguish subsystems from each other: *function* and *flow.* Function refers to the activities carried out by the subsystem. Examination of approximately twenty DBMs yields the following list of possible activities: *filtering, indexing, sorting, partitioning, buffering, metadata processing,* and *general purpose computing.* Metadata processing is distinct from indexing in the sense that indexing simply refers to the operation of mapping a value to a set of addresses whereas metadata processing refers to performing operations on sets of addresses (i.e., the intersection of two inverted lists). By general purpose computing we mean that the subsystem has the capability of performing any computation.

Flow refers to how execution of the activities is mapped to the modules and interconnects that make up the subsystem. Specifically, it indicates how many instruction streams are active and the type of overlap used in their execution. We have identified four types of flow: *single, pipelined, single request parallel (SIMD), and multiple request parallel (MIMD).*

4.4. Dissection Process

The anatomy of a given DBM and the resulting configuration expression can be obtained by following a series of straightforward steps.

First, the work done on the data as it moves form the home repository is examined. It will appear to take place in a number of "steps". A step is a logically complete data location, reduction, ordering, derivation, or computation. The work belonging to a step will be performed by one or more modules and their interconnects. The collection of modules and interconnects associated with a step is identified and isolated as a subsystem.

Next each of the modules and the interconnects in a subsystem is labeled according to its type (see previous sections). Next, each subsystem is labeled using the function and flow descriptors. Finally, the relationships between modules, interconnects, and subsystems are made precise in a configuration expression using the BNF grammar for a funnel anatomy given in Appendix B.

4.5. Examples

Appendix C contains example descriptions of 16 DBMs using the anatomy. The descriptions are given at the subsystem as well as the module and interconnect levels. To illustrate the ideas described in this section we examine the derivation of the anatomy of GRACE [12]. GRACE accomplishes its tasks in three distinct steps. First, the data is retrieved, filtered, and partitioned. Second, the partitions are accumulated in a buffer until all relevant data has been examined. Finally, the data in the buffer is sorted and possibly repartitioned. It then is handed to the host. Repartitioning takes place only if the data is to be used in subsequent operations.

The overall configuration expression is:
host -|=
[order,part:FL:SRTR^{k}] =o= [stage:FL,BUF^{l}] =o=
[index,part,select,FL,GP^{m}] ==== disk^{m}
The host is connected to the DBM using a 1 to many bus (see Appendix B for the definition of the symbols used in the configuration expression). The first subsystem performs indexing, partitioning and selections in a parallel synchronous manner using m general purpose processors, each of which is connected to a disk. The second subsystem stages data in a parallel synchronous manner and uses l buffer units. The third subsystem performs ordering and partitioning, its flow is parallel synchronous, and it is realized using k hardware sorters. Rings are used to connect the various hardware modules.

This is further illustrated in the Figure in Appendix C.

5. Discussion

5.1. Catalogue Observations

The biggest factor in the performance of a database system is the required location and movement of database data in mass storage. Several strategies have been taken by DBMs to deal with this "I/O bottleneck" problem. These include parallel search (i.e., filtering), use of indexing (with and without filtering), clustering, and use of staging to hold frequently used data.

The most popular technique seems to be parallel searching of the data. Fifteen out of the twenty DBMs studied used this technique. In the case of a few of these machines, the search was of the entire database and on behalf of a single request. This greatly limits the potential performance of the machine. Only DIRECT, DBMAC, TERADATA, and SABRE did searching for more than one request at a time.

Many of the DBMs use metadata (typically, an index of some type) to assist in locating data. Two basic approaches are used: multiple single attribute indexing per relation and a single multi-attribute index per relation. In some cases indexing was coupled with filtering. Five of the machines, SABRE, DSDC, GRACE, DELTA, and VERSO[3] adopted the second approach while three machines, DBMAC, TERADTA, and IDM-500, adopted the first approach. Generally, the use of metadata to reduce search space is known to be an excellent technique. It should be one of the major tools in a DBM. It seems reasonable then to consider special hardware to make this tool more efficient. Surprisingly, only DBC and DBMAC provide special hardware for index manipulation (performing intersections between indices). The overhead in maintaining indices can be quite considerable, particularly when they are sophisticated and updated frequently.

Clustering techniques have not been explored much with respect to DBMs. Some organizations could

[3] VERSO actually indexes on the minimal subset of attributes necessary to distinguish the first tuple in a block from the remaining ones. It thus does not employ a "true" multi-attribute indexing scheme.

show a performance gain from clustering and some would not. Those DBMs that filtered the entire database for each request would obviously not benefit. Nothing has been said explicitly about using replicated data (where each copy is clustered on a different attribute) to improve performance.

A fair number of the DBMs employ data staging. However it often seems to be a secondary mechanism. Very few view the staging memory as a central component of their design. DELTA, RAP.2, and DIRECT do use staging in a significant fashion. However, it is not clear whether the size of memory in those machines (and in fact, in any DBM) is large enough to make this effective in reducing the required I/O bandwidth. DELTA which uses a mainframe as a memory hierarchy manager seems particularly ineffective. There is also the recovery problem when data is staged in a memory. Not much seems to have been said about this, though it is solvable, at least for memories that are not enormous.

Almost all DBMs utilize some sort of overlap. In fact, this overlap almost always takes the form of parallel operations. A good example of it is the parallel data searching mentioned above as a popular way for solving the "I/O bottleneck problem". In some machines there is also pipelining. No machine however has tried to take this parallelism to a massive form where there are hundreds of parallel operations happening at once.[4] Also very few machines do parallel work for more than one request at a time.

Amazingly most DBMs lay their database data out in mass storage in the same fashion that the user views it, that is as relations. There are some exceptions, VERSO and RDBM store unnormalized relations. Both machines thus have the potential of supporting molecular objects directly.

DELTA took the approach of storing data as binary relations composed of attribute value and tid. This is a very interesting approach and may have performance benefits in environments where individual attributes are more important than entire relations. A knowledge based system may be an example of this kind of environment. CASSM and DBC associate an attribute identifier with each value in the database allowing them the possibility of implementing transposed files. DBMAC took the approach of storing domains of values in what is called a "data pool" approach. The value of this is unclear.

It is unclear how most DBMs handle large results. The DBMs that do not have significant staging capability will have to pass the entire result to the user at once. DBMs with staging capabilities will be able to send the result in parts (i.e., "stream" it).

The difficulties associated with recovery are often glossed over. Certain DBM organizations are not very amenable to this. Machines that modify the data in place such as RAP.1, VERSO, RARES, EDC, CAFS, DBC, and CASSM will have difficulty recovering from a failure.

Mechanisms for concurrency control are often not addressed. A good many of the DBMs only handle one user request at a time and so this is not a factor. Few consider updates at all.

There seems to have been a trend in DBM design where earlier machines like CASSM and RAP.1 were primarily oriented toward locating data and performing primitive operations on it; while later DBMs such as DELTA, GRACE, and TERADATA deal with high level queries. The higher the level of the interface to the DBM, the more opportunity there is to optimize execution and to take advantage of special hardware for speed improvement.

A final observation is that with time DBM designers have become increasingly sophisticated in their use of processing primitives. The initial designs used filtering and nested loops (with and without broadcast) to realize the various relational algebra operators. Later machines make more use of hashing and sorting with GRACE, DELTA, RDBM, DSDC, and TERADATA realizing these in special purpose hardware. However, few machines have implemented highly parallel sorting algorithms.

5.2. Anatomy Observations

An examination of the anatomy of the DBMs shown in Appendix C has led us to place them into five classes based on similar characteristics. The common characteristics for machines in a group are: 1) the number of subsystems in the DBM, and 2) the functions realized by the subsystems. The five classes are named as follows: Two-stage filter (2F), Two-stage index-filter (2IF), Three-stage computation (3C), Three-stage index-filter-sort (3IFS), and Three-stage

[4] There are two possible exceptions to this statement. Tanaka, in [25] which we had time only to skim, may be the exception here. He discusses the design of MPDC -- a Massively Parallel DBM. The NONVON machine is supposed to employ massive amounts of parallelism. However, only part of the machine has been specified so far and it is thus excluded.

metadata processor-filter (3MF).

2F consists of those DBMs that use two subsystems to realize filtering and data enhancing or derivation (i.e., joins). No indexing is supported by these DBMs. 2IF consists of those DBMs that use two subsystems to realize filtering, indexing, and data derivation. 3C consists of those DBMs that use three subsystems to realize coarse (relation-level) indexing, data manipulation operations (including filtering), and buffering. 3IFS consists of those DBMs that use three subsystems to realize filtering, indexing, and sorting. Finally, 3MF consists of those DBMs that use three subsystems to realize metadata operations and filtering. Note that in all cases we have specified the number of subsystems, the activities performed by the subsystems but not the mapping of the activities onto the subsystems.

We now look at each class in more detail. 2F contains EDC, RAP.1, and CASSM and typifies early DBM efforts. All three DBMs use multiple filters in SIMD fashion, and no indexing. All other operations on the data are performed by a single general purpose processor. Historically, this class of DBMs is important because it helped spur interest and further work in the area. Practically speaking it has little significance.

2IF contains VERSO and the Britton-Lee IDM-500 DBMs. Common to both DBMs is the use of indexing to reduce the search space. However, they differ in the storage structures used and the corresponding indexing scheme. VERSO employs an unnormalized format in which the values within a field are sorted whereas the IDM-500 uses a more conventional structure in which a normalized relation is clustered on its primary key. Neither machine employs parallelism. The IDM-500 does have a buffer whereas VERSO does not. This class is representative of what we call present day potential (and in the case of the IDM-500 existing) commercial DBMs for the DP application environment. Both DBMs are geared towards fast execution of selection type queries (although the storage structure used in VERSO makes it amenable for fast joins in some cases).

3C contains RAP.2 and DIRECT. Both DBMs use relation-level indexing and general purpose processors coupled with a large distributed buffer to realize all the data manipulation operations. RAP.2 associates each buffer unit with a specific processor and uses SIMD type parallelism in the execution of one operation whereas DIRECT allows any processor to access any memory unit and uses MIMD type parallelism within and across operations. Both DBMs are representative of the idea that computation and not I/O are the main problem in DBM design. Both designs were optimized for the "hard" operations, such as join.

3IFS contains GRACE, DSDC, TERADATA, DELTA, and RDBM. It is probably the most controversial of the five classes because there are enough distinguishing features between the five members of the group to place each in a separate class. All five DBMs use indexing to reduce the search space and incorporate hardware sorters. All but TERADATA incorporate hardware filters (in DELTA this is the merge processor within the sorter). DELTA and GRACE use a buffer. DELTA, RDBM, and TERADATA use general purpose processors in addition to special purpose processors. DSDC, TERADATA, RDBM, and DELTA support the parallel execution of several operations whereas GRACE supports the parallel execution of several operations from a single query. Finally, TERADATA uses an intelligent switch.

The last class is 3MF and includes DBC[5] and DBMAC. Both machines are distinguished by their use of special purpose hardware to realize operations on metadata. Both machines also uses parallel filters although DBC can only handle one operation at a time through its filtering subsystem whereas DBMAC can handle several.

5.3. Summary

In this paper we presented a scheme for describing and classifying DBMs. Our scheme is made up of two components: a catalogue in which DBMs are described using seven attributes and an anatomy in which the hardware organization of DBMs is dissected. Using the catalogue a DBM can be described quickly and an overview of the general mechanisms it uses to accomplish its work is given. The anatomy provides a convenient mechanism for capturing the detailed architecture of DBMs.

Two important contributions are made in this paper. The first is the scheme itself and the second is the collection of insights into existing DBM designs (see Sections 5.1 and 5.2).

Previous taxonomies of DBMs seem to be based on two attributes: degree of parallelism used and search strategy employed. For example, Bray and Freeman

[5] We are assuming that a Post Processing Unit makes up the third subsystem of DBC.

[5] propose five categories based on number of processors and type of search. Song [22] proposes distinguishing between machines based on three attributes: placement of logic (secondary versus primary memory), allocation of logic (static versus dynamic), and degree of logic distribution (a range of "low" to "high"). DeWitt and Hawthorn [7] partition machines based on the number of and coupling type of logic and secondary memory. Hsiao [11] distinguishes among machines based on their use of software (i.e., conventional processors) versus hardware (i.e., special purpose processors) and whether parallelism is used. Only Qadah [16] takes a broader look by distinguishing among machines based on the following three attributes: Processor-Memory organization (SISD, SIMD, and MIMD), Query Processing Place (On-Disk, Off-Disk, and Hybrid), and Indexing Level (Page, Relation, and Database).

Our approach has been to concentrate on the design of a scheme to describe DBMs resulting in an open-ended grouping. The grouping itself is only of secondary importance since it seems likely that no single grouping will capture future machine designs. However, we believe our scheme to be sufficiently general to enable the description of future machine designs.

The observations made in Sections 5.1 and 5.2 reveal several important weaknesses and some trends in existing DBM designs which the designer of a future DBM should benefit from. The main negative statements are: the I/O bottleneck has been dealt with naively and emphasis was placed on improving response time for a single operation (or query) rather than increasing the overall throughput of the system. An interesting trend is the use of sorting as a basic processing primitive in most of the recent designs.

6. Acknowledgements

We wish to thank Marc Smith for making several suggestions that improved the contents and presentation.

7. References

[1] Babb E., "Implementing a Relational Database by Means of Specialized Hardware," *ACM TODS,* Vol. 4 No. 1, 1979.

[2] Bancilhon F. et al., "VERSO: A Relational Back-end Database Machine," in [11].

[3] Banerjee J., D.K. Hsiao, and K. Kannan, "DBC - A Database Computer for very Large Databases," *IEEE Trans. Computers* Vol. c-28, No. 6, 1979.

[4] Boral H. and D.J. DeWitt, "Database Machines: An Idea Whose Time Has Passed? A Criticque of the Future of Database Machines," in *Database Machines,* H.-O. Leilich and M. Missikoff (eds), Springer-Verlag, 1983.

[5] Bray O.H. and H.A. Freeman, *Data Base Computers,* Lexington Books, 1979.

[6] DeWitt D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," *IEEE Trans. Computers* Vol. c-28, No. 6, 1979.

[7] DeWitt D.J., and P. Hawthorn, "A Performance Evaluation of Database Machine Architectures," *Proceedings 7th VLDB,* 1981.

[8] Epstein R. and P. Hawthorn, "Design Decisions for the Intelligent Data Base Machine," *Proc. NCC,* 1980.

[9] Gardarin G. et al., "SABRE: A Relational Database System for a Multimicroprocessor Machine," in [11].

[10] Goodman J.R., "An Investigation of Multiprocessor Structures and Algorithms for Data Base Management," Electronics Research Laboratory Memo No. UCB/ERL M81/33, University of California, Berkeley, 1981.

[11] Hsiao D.K., *Advanced Database Machine Architecture,* D.K. Hsiao, ed. Prentice-Hall, 1983.

[12] Kitsuregawa M., H. Tanaka, and T. Moto-Oka, "Application of Hash to Data Base Machine and Its Architecture," *New Generation Computing,* Vol. 1, 1983.

[13] Lin S.C., D.C.P. Smith, and J.M. Smith, "The Design of a Rotating Associative Memory for Relational Database Applications," *ACM TODS,* Vol. 1 No. 1, 1976.

[14] Missikoff M. and M. Terranova, "The Architecture of a Relational Database Computer Known as DBMAC," in [11].

[15] Ozkarahan E.A., S.A. Schuster, and K.C. Smith, "RAP - An Associative Processor for Database Management," *Proc. AFIPS,* Vol. 44, 1975.

[16] Qadah G.Z. and K.B. Irani, "A Database Machine for Very Large Relational Databases," *Proceedings of the 1983 International Conference on Parallel Processing,* 1983.

[17] Schuster S.A. et al., "RAP.2 - An Associative Processor for Databases and Its Applications," *IEEE Trans. Computers* Vol. c-28, No. 6, 1979.

[18] Schultz R.K. and R.J. Zingg, "Response Time Analysis of Multiprocessor Computers for Database Support," *ACM TODS,* Vol. 9 No. 1, 1984.

[19] Schweppe H. et al., "RDBM - A Dedicated Multiprocessor System for Database Management," in [11].

[20] Shaw D.E., Talk given at MCC, May 1984.

[21] Shibayama S. et al., "A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor," *New Generation Computing,* Vol. 2, 1984.

[22] Song S.W., "A Survey and Taxonomy of Database Machines," *Database Engineering,* Vol. 4, No. 2, 1981.

[23] Su S.Y.W. et al., "The Architectural Features and Implementation Techniques of the Multicell CASSM," *IEEE Trans. Computers* Vol. c-28, No. 6, 1979.

[24] Tanaka Y., "A Data Stream Database Computer," in *Japan Annual Reviews in Electronics, Computers & Telecommunications: Computer science & Technologies,* T. Kitagawa, ed., OHM - North Holland, 1982.

[25] Tanaka Y., "MPDC - A Massive Parallelism Database Machine," *Proc. Int'l Conf. on Fifth Generation Computers,* 1984.

[26] "DBC/1012 Data Base Computer Concepts & Capabilities," Document No. C02-0001-00, Release 1.0, Teradata Corp., 1983.

[27] Uemura S. et al., "The Design and Implementation of a Magnetic-Bubble Database Machine," *Proc. IFIP '80,* 1980.

Appendix A - Database Machine Catalogue

In this appendix we describe 20 database machines. Table 1 consists of the descriptors for the seven attributes making up the catalogue for all 20 machines.

DATABASE MACHINE CATALOGUE - SUMMARY

DBM	Mission	# Mis	Ovrlp	Mem	Proc Meth	Locat	Stor Struct	Ref
CAFS	RA Op	1	L	No	Hash Smjn	DB-lvl Filter	unr	[1]
CASSM	RA Op	1	L	No	Smjoin	DB-lvl Filter	Att w. Mk bits	[23]
DBC	RA Op	Few	L	No	Intrsction	Cylndr Index	sacnr	[3]
DBMAC	A Tree	Sev	L&P	No	Intrsction	Inverted Domains	Inverted Domains	[14]
DELTA	RA Op	Few	L&P	Buf	Srt-mrg	2-lvl Index	Att w. TIDs	[21]
DIRECT	A Tree	Sev	L&P	Buf	GP Proc	Reln-lvl Index	unr	[6]
DSDC	RA Op	1	L&P	No	Sort Search	Reln-lvl Index	Encoded Rel'ns	[24]
EDC	RA Op	1	L	No	Nested Loops	DB-lvl Filter	unr	[27]
GRACE	A Tree	1	L&P	Buf	Hash Sort	Multi Att Index	macnr	[12]
HYPER-TREE	RA Op	1	L	Buf	Hash Sort	NA	unr	[10]
IDM-500	A Tree	1	No	Buf	NA	Index	sacnr	[8]

NONVON	RA Op	1	L	No	Brdcst Nstd-lps	NA	unr	[20]
RAP.1	RA Op	1	L	No	Filtered Smjoin	DB-lvl Filter	unr w. Mrk Bits	[15]
RAP.2	RA Op	1	L	No	Filtered Smjoin	Reln-lvl Index	unr w. Mrk Bits	[17]
RARES	Assoc Access	1	L	No	No	DB-lvl Filter	unr	[13]
RDBM	A Tree	Few	L&P	Buf	Sort	Reln-lvl Index	Unnorm Relns	[19]
REPT	RA Op	1	L	NA	GP Proc	NA	unr	[18]
SABRE	RA Op	Few	P	Buf	GP proc	Multi Att Index	macnr	[9]
TERA-DATA	RA Op	Sev	L	Buf	Srt-mrg	NA	unr	[26]
VERSO	RA Op	1	No	No	Sort	Primary Index	Unnorm Sorted Relns	[2]

Table 1[1]

Appendix B - Anatomy BNF

<dbm>	::=	host <ic> <funnel> <ic> <home_rep>
<funnel>	::=	<subsystem> | <ic> <subsystem> <ic> <funnel>
<ic>	::=	ic_type
<subsystem>	::=	[<function> : <flow> : <config>]
<function>	::=	function_type | function_type , <function>
<flow>	::=	flow_type
<config>	::=	<module>$^{<cnt>}$ | <config> <ic> <module>$^{<cnt>}$
<module>	::=	module_type
<cnt>	::=	integer | letter |
<home_rep>	::=	disk$^{<cnt>}$ | disk_cyl$^{<cnt>}$ | disk_track$^{<cnt>}$

ic_type	::=	
	---	pt-to-pt
	-x=	1-to-m switched
	-o=	1-to-m ring
	-|=	1-to-m bus
	=x-	m-to-1 switched
	=o-	m-to-1 ring
	=|-	m-to-1 bus
	=x=	m-to-m switched
	=o=	m-to-m ring
	=|=	m-to-m bus
	===	parallel pt-to-pt

module_type	::=	
	AP	(arithmetic processor)
	BUF	(buffer)
	CTRL	(controller)
	FLTR	(filter)
	GP	(general purpose processor)
	HASH	(hasher)
	IP	(index processor)
	MRGR	(merger)
	SRTR	(sorter)

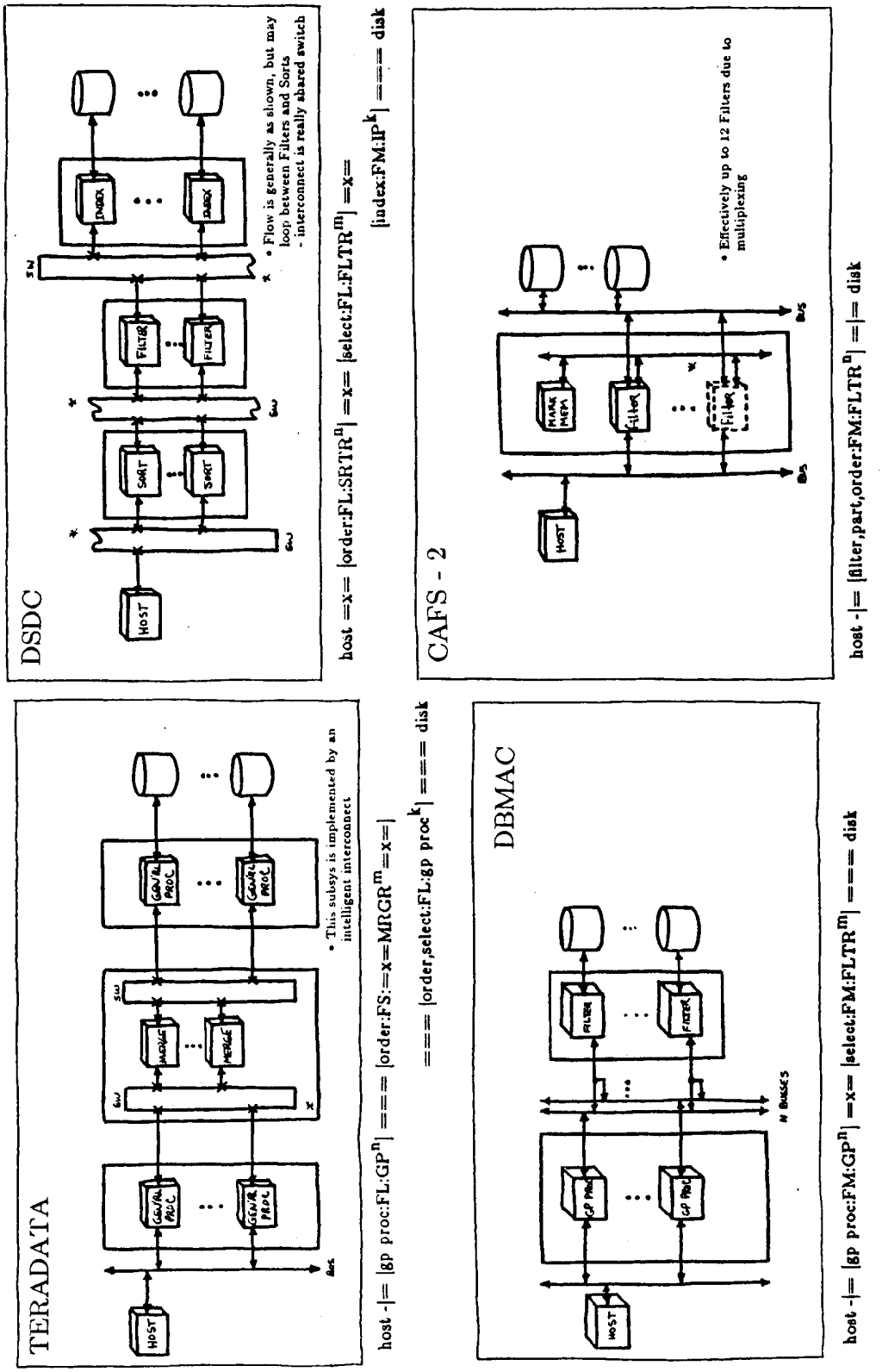
flow_type	::=	
	FS	(single)
	FP	(pipelined)
	FL	(parallel Synchronous)
	FM	(parallel Asynchronous)

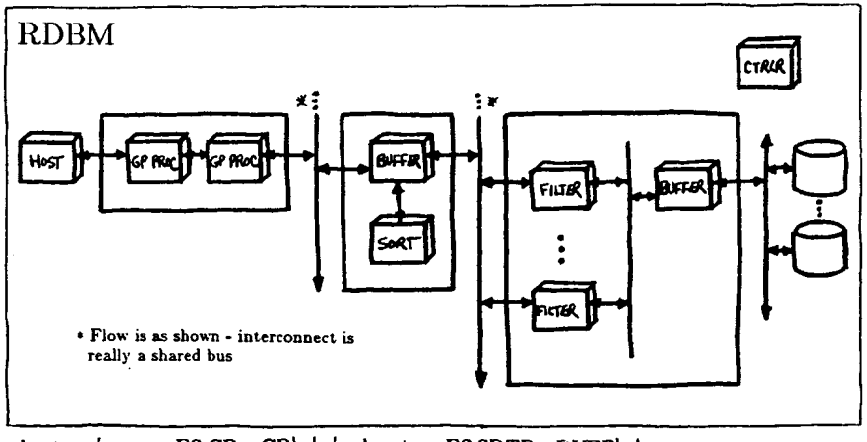
function_type	::=	
	stage	
	order	
	index	
	select	
	gp proc	(general purpose computing)
	part	(partition)
	meta proc	(metadata processing)

[1] The abbreviations used in the Storage Structures column are defined in Section 3.2.

Appendix C - Database Machine Anatomies

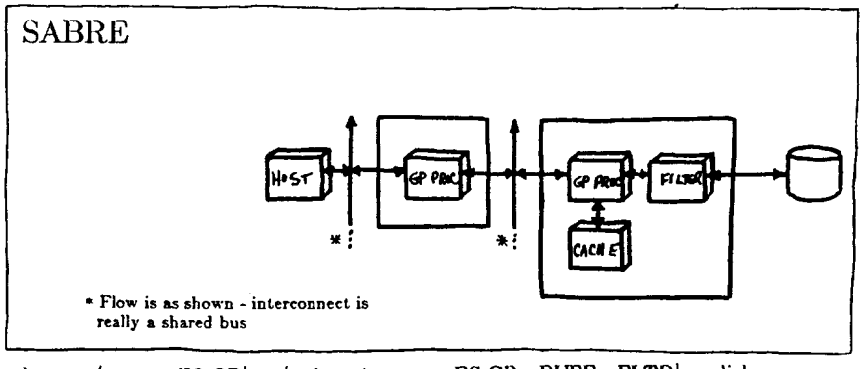
The anatomy of 12 database machines are characterized in this appendix first by a subsystem diagram and then by a configuration expression using the BNF of Appendix B.



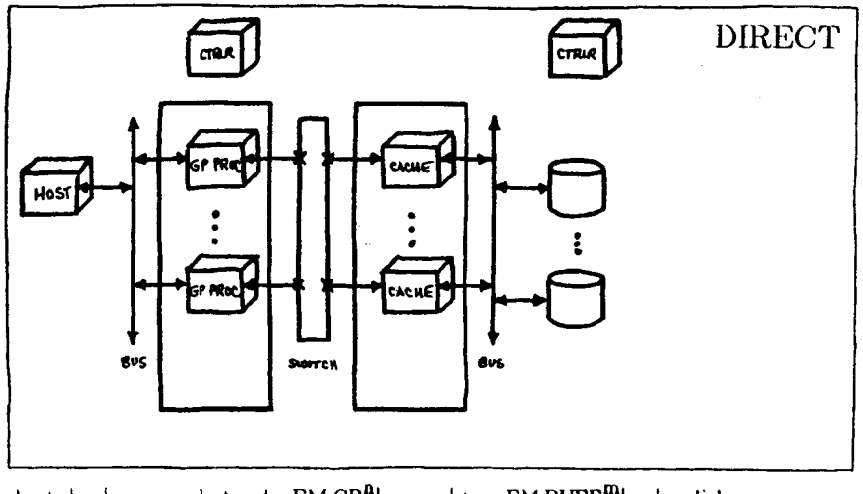


host — |gp proc:FS:GP—GP| -|- |order,stage:FS:SRTR—BUFF| -|=

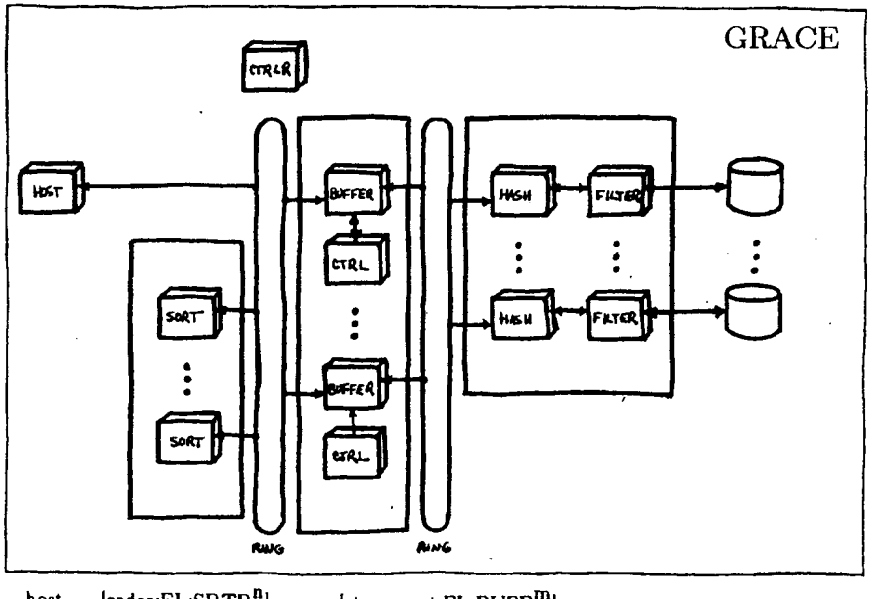
|select,stage:FL:FLTR^{n}=|-BUFF| -|= disk



host — |gp proc:FS:GP| — |order,select,stage:FS:GP—BUFF—FLTR| — disk

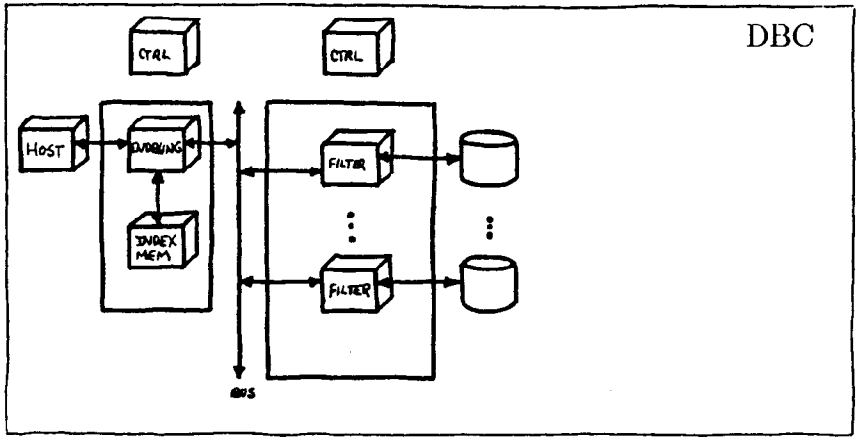


host -|= |gp proc,select,order:FM:GP^{n}| =x= |stage:FM:BUFF^{m}| =|= disk

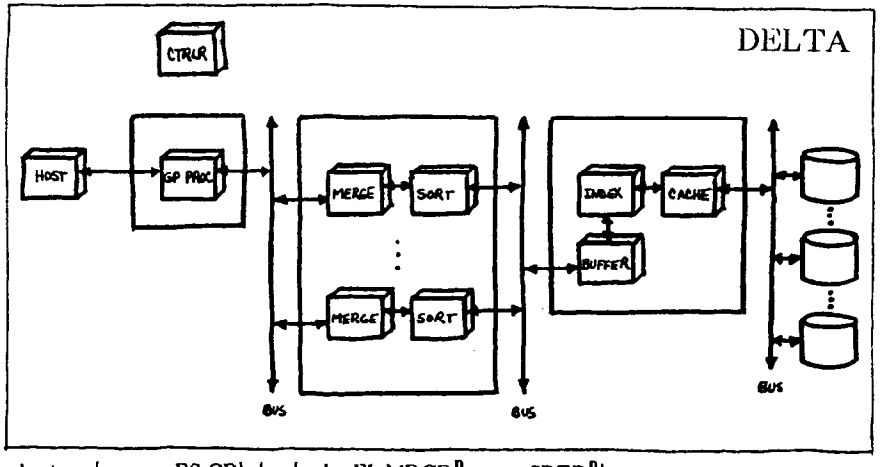


host — |order:FL:SRTR^{n}| =o= |stage,part:FL:BUFF^{m}| =o=

|part,select:FL:HASH^{k}===FLTR^{k}| === disk

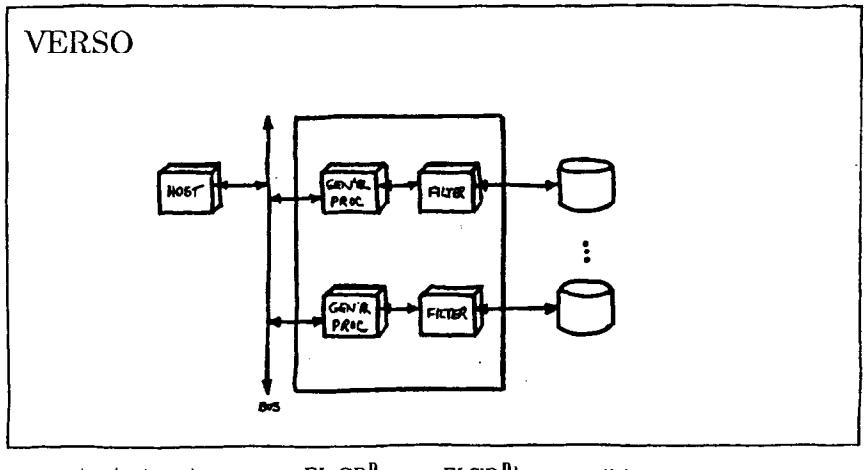


host — |index:FS:IP| -|= |select:FL:FLTR^n| === disk

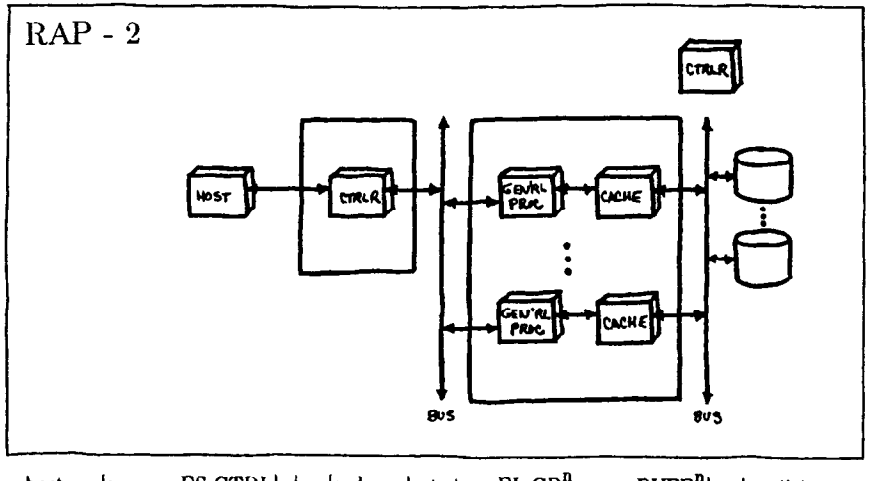


host — |gp proc:FS:GP| -|= |order:FL:MRGR^n===SRTR^n|

=|- |stage,index:FS:BUFF—IP| -|= disk



host -|= |order,select,gp proc:FL:GP^n===FLTR^n| === disk



host — |gp proc:FS:CTRL| -|= |order,select,stage:FL:GP^n===BUFF^n| =|= disk