# HIGH PERFORMANCE DESIGN TECHNIQUE

Dr. C. J. Conti

IBM Corporation,
1000 Westchester Avenue,
White Plains,
New York, 10604,
U.S.A.

Abstract:

This paper discusses the trends in C.P.U. and storage design, illustrated by reference to the IBM 360/195, and predicts some of the future trends.

Rapporteurs:

Mr. J. G. Givens
Mr. I. Mitrani

The talk presented by Mr. Conti was divided into three parts. Firstly, the trends in technology in C.P.U. and storage design up to the present time; secondly, an illustration of the present day state of the art using the IBM Model 360/195 as the example, and thirdly, some predicted future trends in terms of specific design techniques and ideas.

Mr. Conti commenced by illustrating (figure 1) the trend in technology during the past few years in the reduction of circuit delay, and continued by showing (figure 2) that the improvement in performance in memory has been equally impressive. At present the tendency is to move away from film memories towards solid state monolithic memory although these are as yet very expensive. The Model 195 has a conventional ferrite core memory but the buffer store is solid state monolithic which has a cycle time of 54 nanoseconds.

The normalized C.P.U. performance in a mix of jobs plotted against improvements in performance is shown in figure 3, and this plot raises questions concerning the direction the designers must go. It would appear that two ways are possible, the first being the parallel processor and the other the higher overlapped single instruction counter and one of the differences between these two approaches was demonstrated in figure 4.

The theoretical limit of parallelism is a straight line which is not true of the single instruction counter in which a stage can be reached when additional components produce no further improvements. Thus a point is reached when, to achieve improvements, the system must be made parallel at least in part.

One of the main advantages of parallelism is that smaller repetitious elements are used, which in turn produces easier maintenance, simplified field stacking and the ability to produce modular constructions.

One significant disadvantage is the programming difficulty. Many problems do not appear to lend themselves to parallel processing and for those that are capable of being processed in a parallel fashion there is as yet no generalized algorithm for translating programming thought from sequential to parallel mode.

The problem to be faced in any form of parallel machine when attempting a single problem was represented by the dilution curve shown in figure 5.

This was a specific example since there is a different curve for every parallel machine depending upon the degree of parallelism. A choice of N=32 was taken here where N is the number of repetitious elements. The achievement of maximum performance is related to the percentage of the problem which may be done fully in parallel.

If 90% of the problem can be attacked in parallel the performance is approximately 70% of maximum. If N was greater than 32 the performance would drop below 70%. (Here comment was made from a number of members of the audience upon the definition of parallelism and that the dilution curve is purely theoretical and depends upon the definitions used).

The classic forms of parallelism are shown in figure 6. The single instruction counter pipelined approach was then discussed.

Economically there is considerable waste in this type of system. The instruction phases can ideally be completely overlapped (figure 7) and can be regarded as a crude pipeline. In practice, however, not all instructions are of equal length and the 'diameter' of the pipeline changes. To clear this bottleneck a scheme (figure 8) was illustrated. The replication of some elements is due to further bottlenecks having to be overcome. Problems still remain since the clearing of one bottleneck merely creates others. Storage conflict will still remain, mainly due to storage access time being longer than basic machine time. There will be a unit conflict simply because there may not be enough units to achieve all possible combinations. For example, if three adds occur together and only one adder exists then unit conflict occurs. Data dependence is a serious bottleneck. If we wish to add 2 to 2 and divide the result by 2, the divide cannot be done before the addition has been completed. There are severe penalities which must be recognized due to branch on condition instructions. The branch turns out to be the singly most important problem occurring as it does about once in every three instructions in many types of programs.

There are two possible methods of attack. The first, is to use a buffer memory and the second to use a condition mode solution (figure 9). Instructions are stacked following a branch but, no data if fetched on the assumption that in half the cases the branch will not be taken. Additionally, the instruction pointed to by the branch is stacked. Upon verification of the condition either patch can be executed from the stack as illustrated in figure 10.

At this point a question was posed from the audience as to the feasibility of stacking more than one instruction from the 'yes' path, and Mr. Conti replied that he considered this to be a problem of expensive complexity though entirely possible.

The first part of the lecture was concluded by showing (figure 11) a diagram of the storage hierarchy of the Model 195.

The lecture was resumed with a detailed description of the C.P.U. organization of the Model 195 (figure 12) and Mr. Conti shows step by step how a set of instructions are manipulated by the C.P.U. to enable sets of instructions to be concurrently dealt with although within a set they must be consecutive.

Out-of-sequence operations produce their own problems, an example of which is termed 'imprecise interrupt' which occur because some processes are internal to the machine and not precisely defined as to when they are to be processed.

Mr. Conti then examined the method of defining a structure when evaluating the effectiveness of a given proposed C.P.U. design. The method employed is to take a job and run it on a 360 processor under a trace program which records on magnetic tape the step by step work of the C.P.U. together with all other relevant information.

This tape is run on a timing program which is a precise cycle by cycle model of the way in which a proposed C.P.U. would execute a series of instructions.

The result of such a program was shown (figure 13) using 17 tapes for C.P.U. comparisons nomalized to Model 360/65.

The speaker then turned his attention to the possibilities that existed for future improvements, and the potential applications of the single instruction counter machine were then described using the statistics accumulated from the trace programs previously mentioned. If a branch in one direction has been taken then it turns out that with a probability of about 80% that same branch if executed again will be taken in the same direction. Thus, a branch bias table which keeps track of branch instructions will enable up to 80% of correct guesses to be made as to the direction of the next branch. For such a table to be useful it must contain a large number of entries (e.g. 128).

This idea has been discussed for some years but only with the recent technology in monolithic circuitry does it become a practical proposition.

Some time was now spent in describing the existing storage organization in the Model 360/85 and other organizations such as direct mapping, fully associative and set associative which provide better buffer utilities were illustrated (figures 14, 15, 16, 17, 18 and 19).

The set associative system is approximately implemented in Model 195.

The speaker then pointed out that there are further considerations which arise once a basic buffer algorithm has been decided upon, one example of which is the problem of 'store through' versus 'swapping'. Store through is a term used to indicate that every time a store occurs it is performed in backing store as well as buffer storage.

This produces integrity of backing store. There is, however, a performance advantage to swapping. A further consideration is the possibility of fetch anticipation. If a block has just been called it would seem likely that the next sequential block will be needed soon, and so this is anticipated and the block is fetched. This method tends to slow down processing.

Mr. Conti suggested that a future product of buffer use would be the inclusion of a checkpoint re-try (figure 20). This gives a number of advantages.

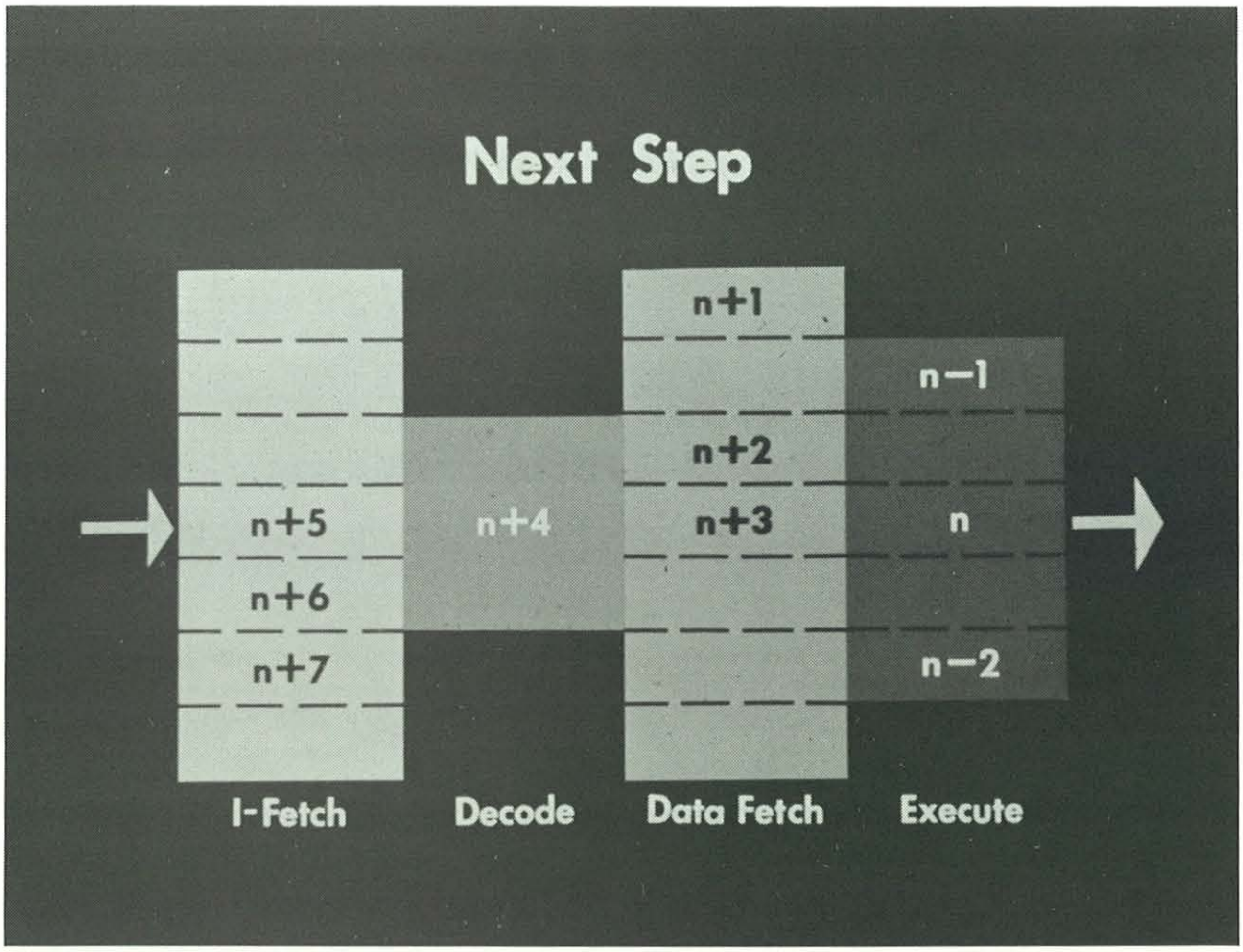


Figure 8

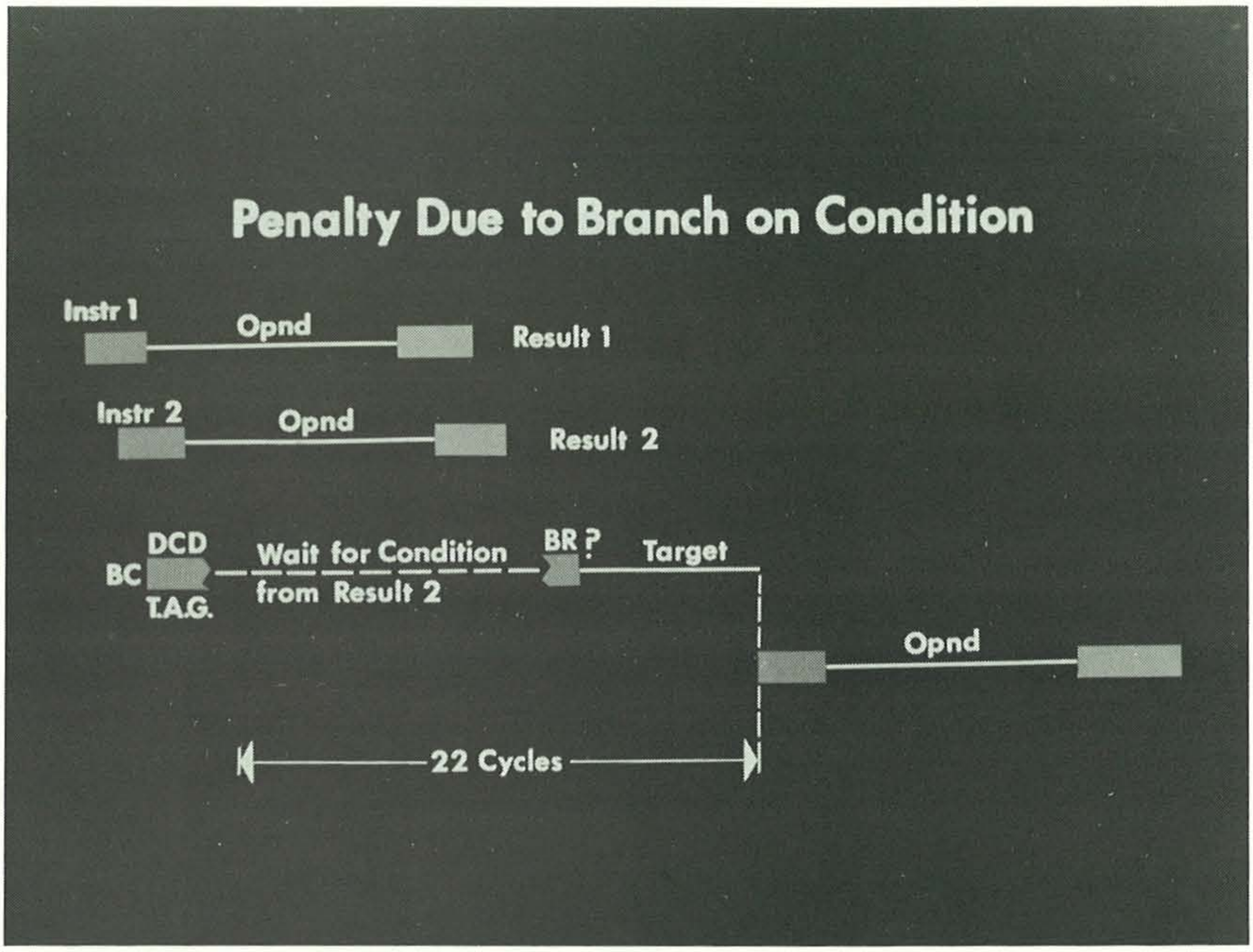


Figure 9

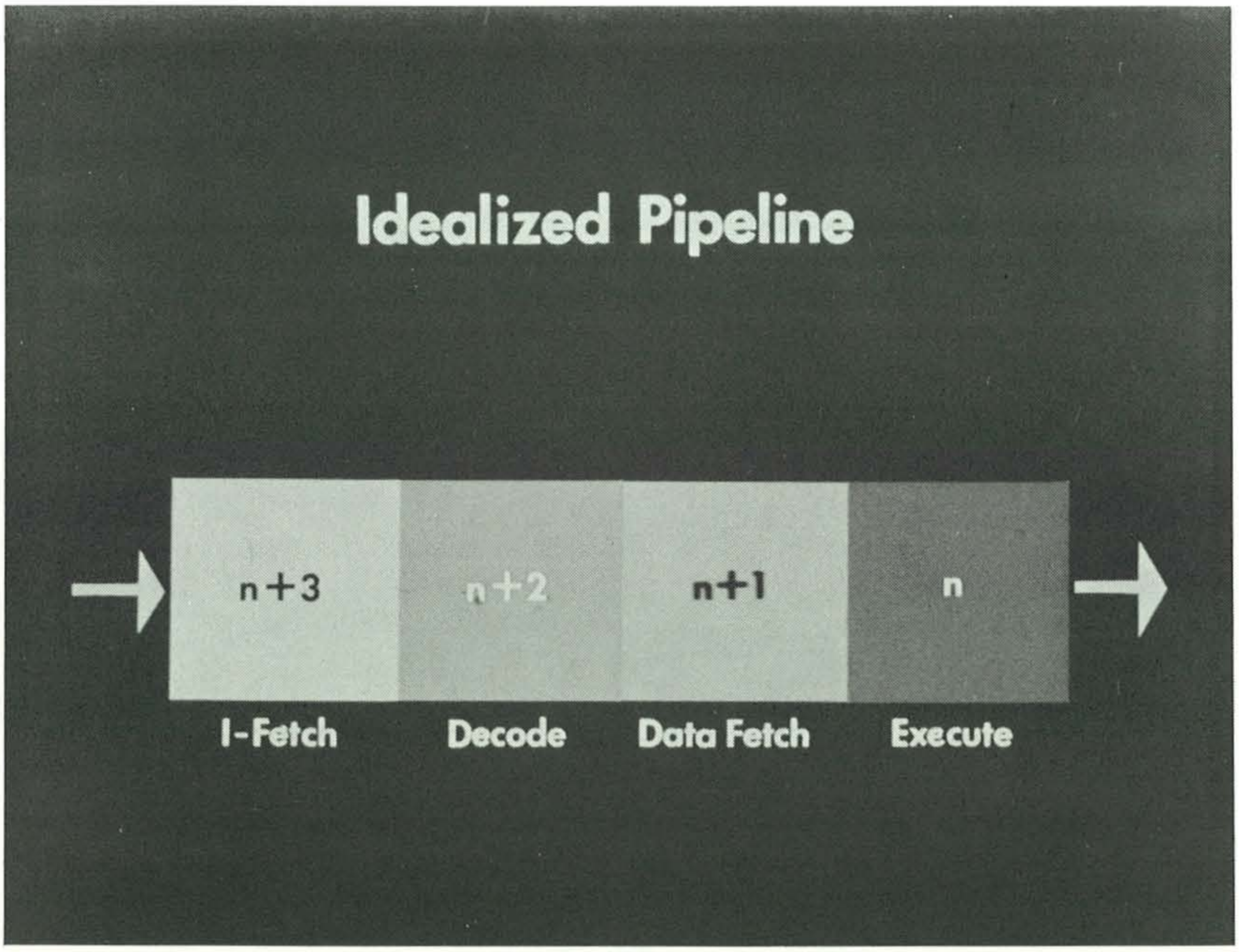


Figure 7 (part 1)

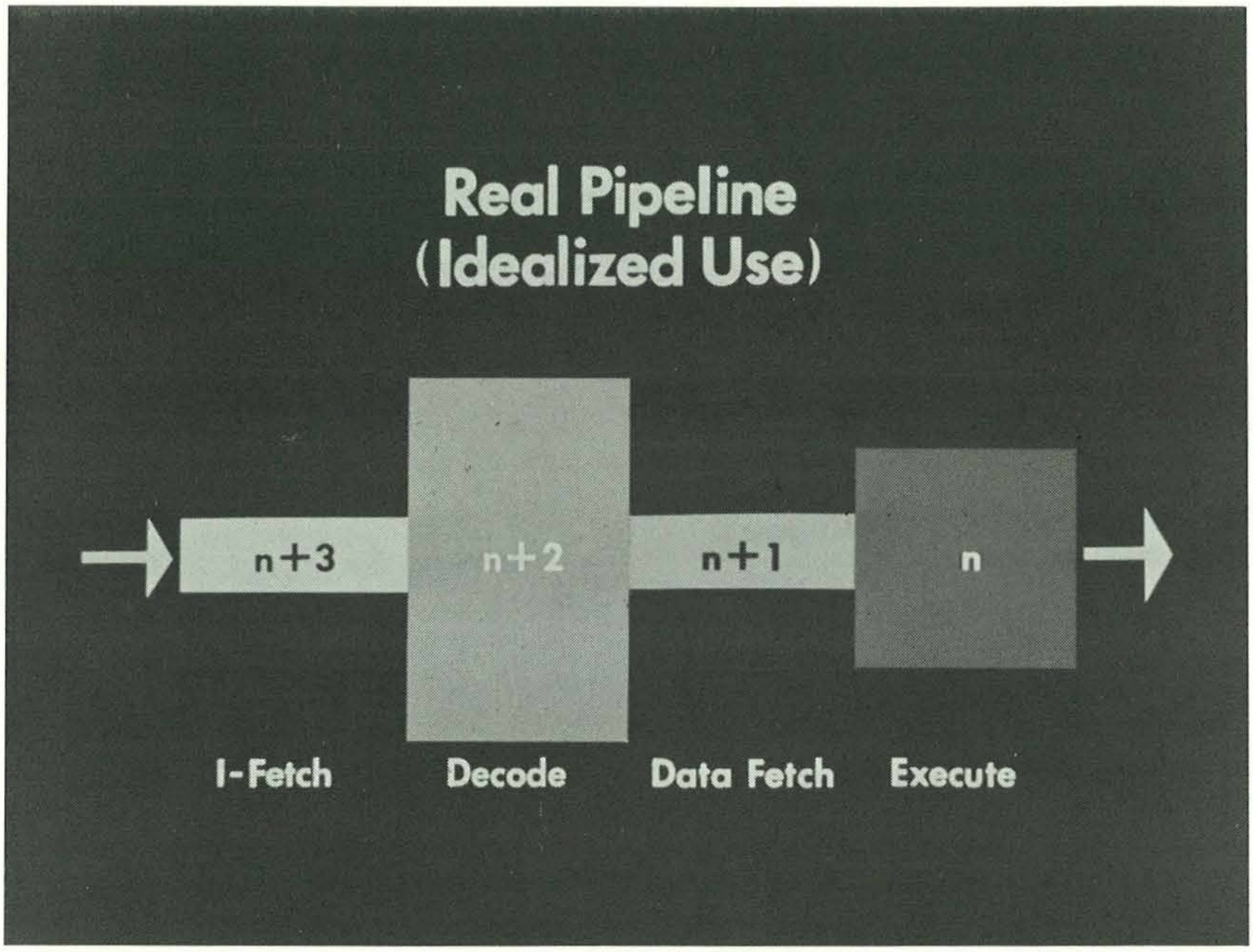


Figure 7 (part 2)

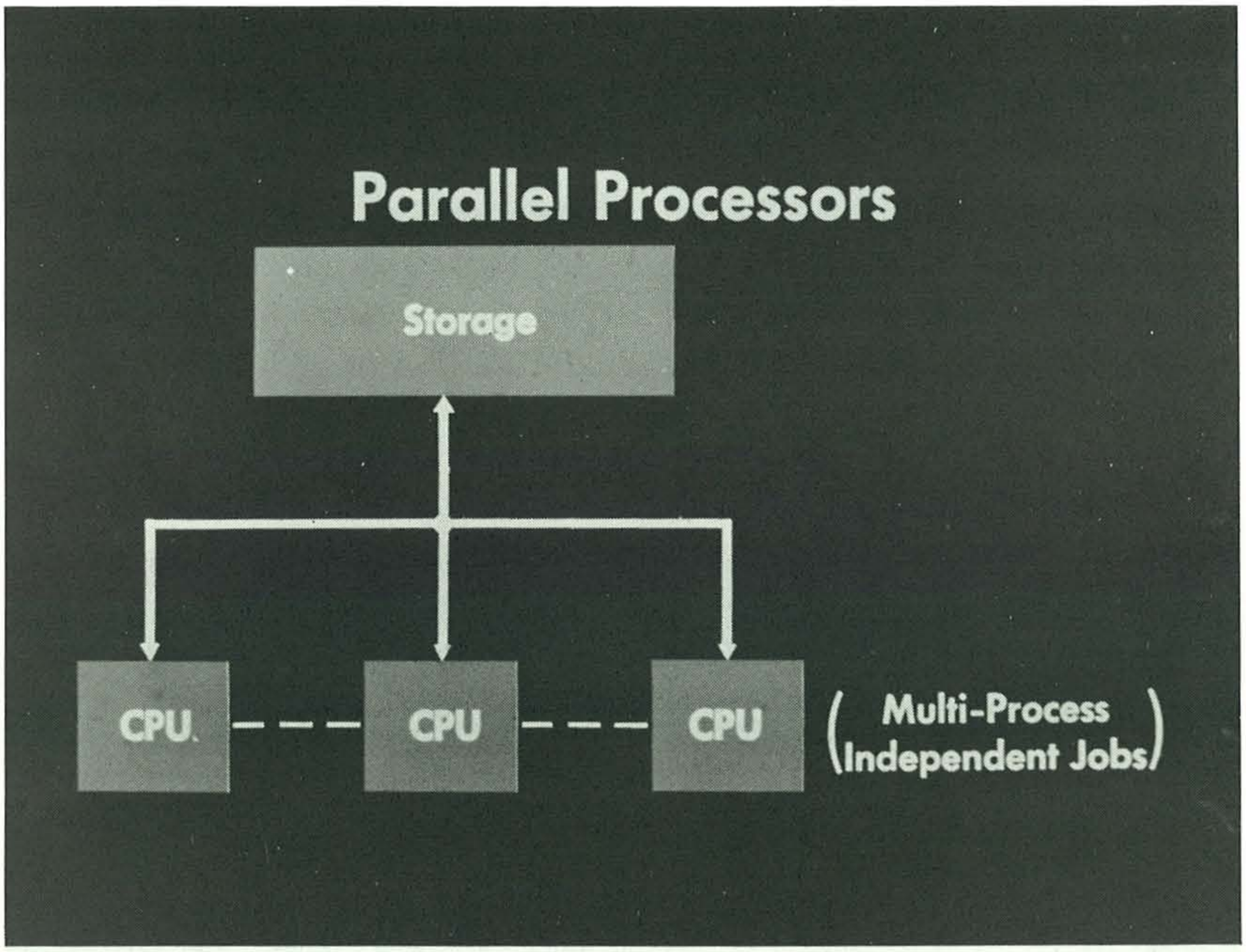


Figure 6 (part 1)

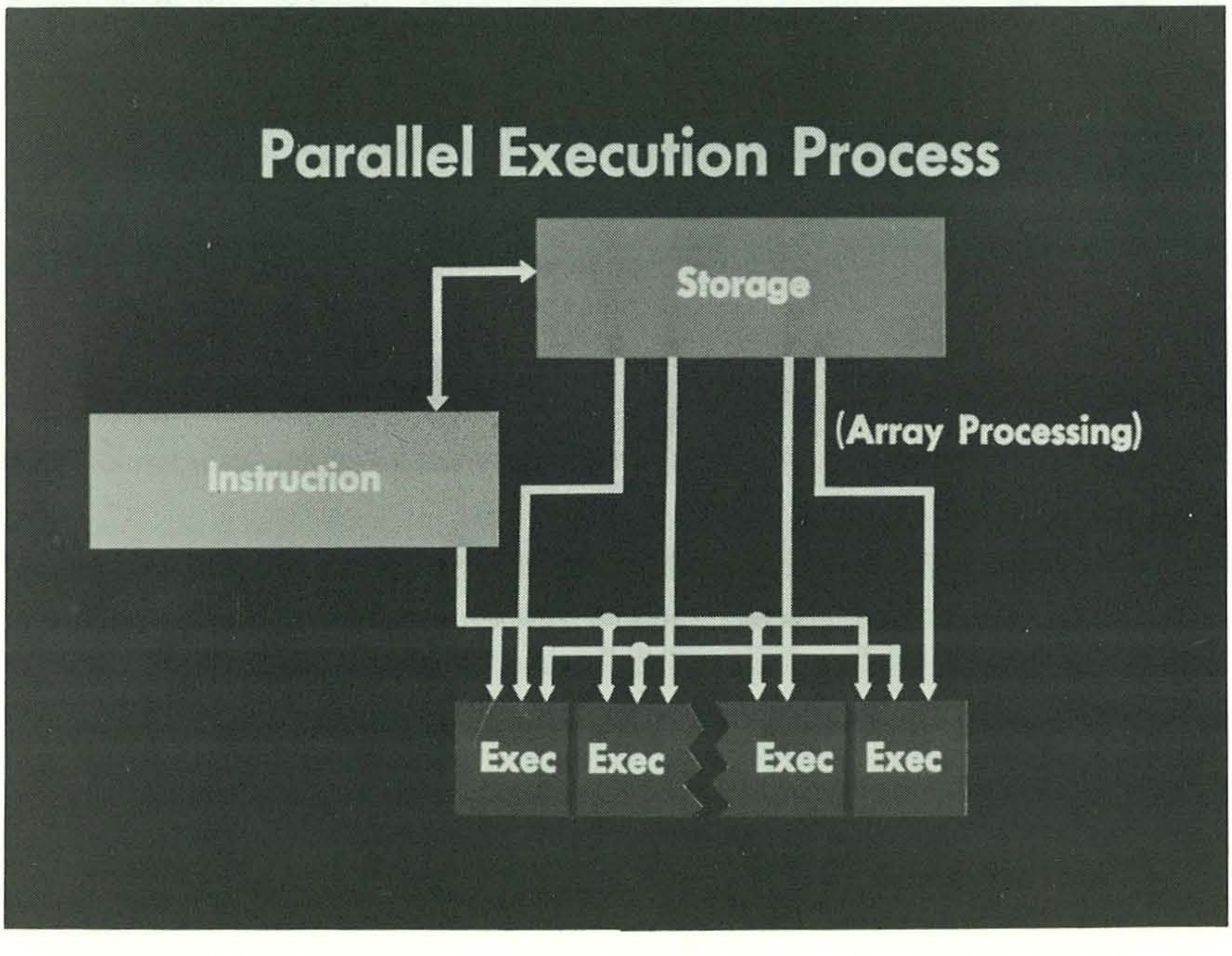
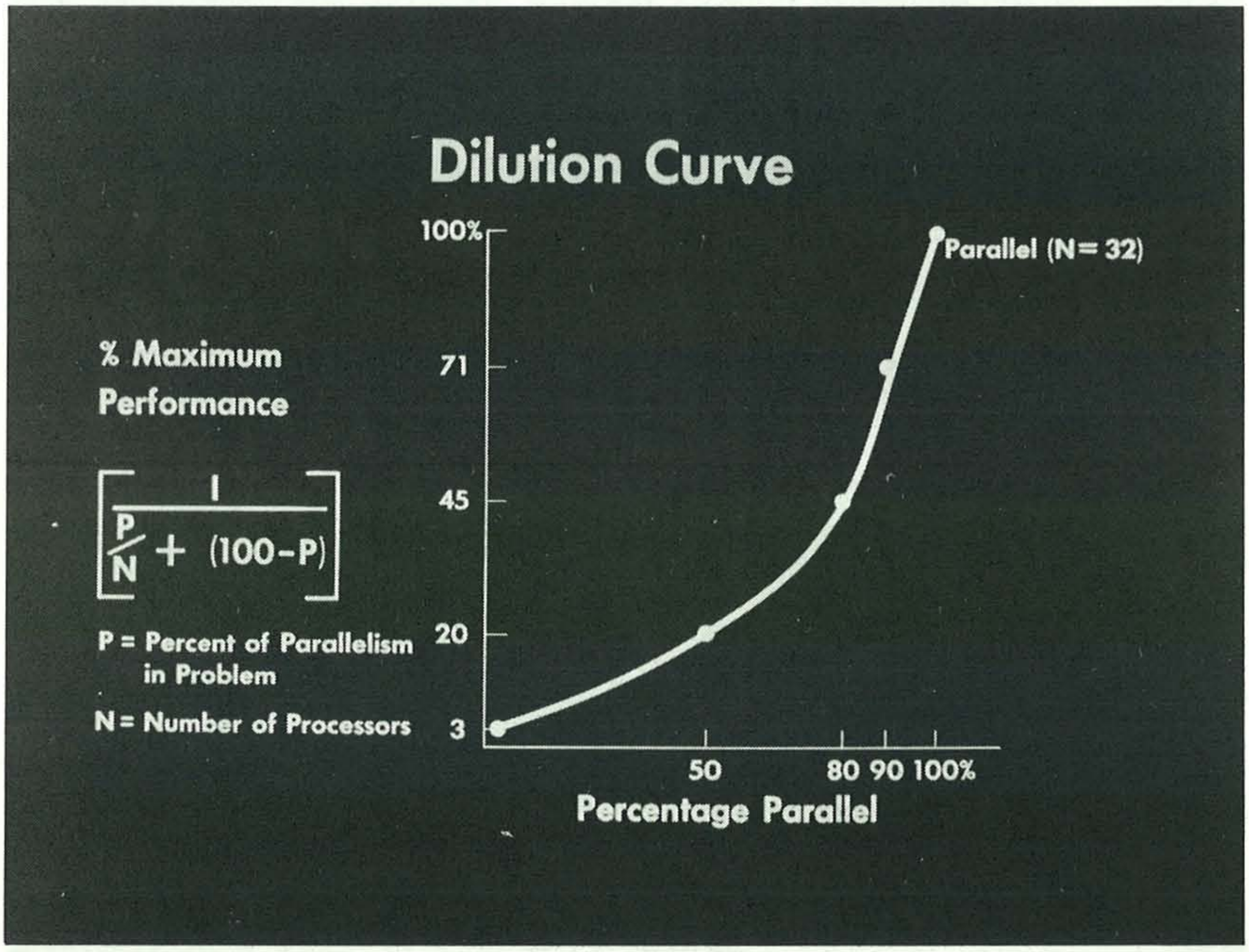


Figure 6 (part 2)



Dilution Curve
100%
71
45
20
3
Parallel (N= 32)
% Maximum Performance
$\left[\frac{1}{\frac{P}{N} + (100-P)}\right]$
P = Percent of Parallelism in Problem
N = Number of Processors
50
80 90 100%
Percentage Parallel

Figure 5

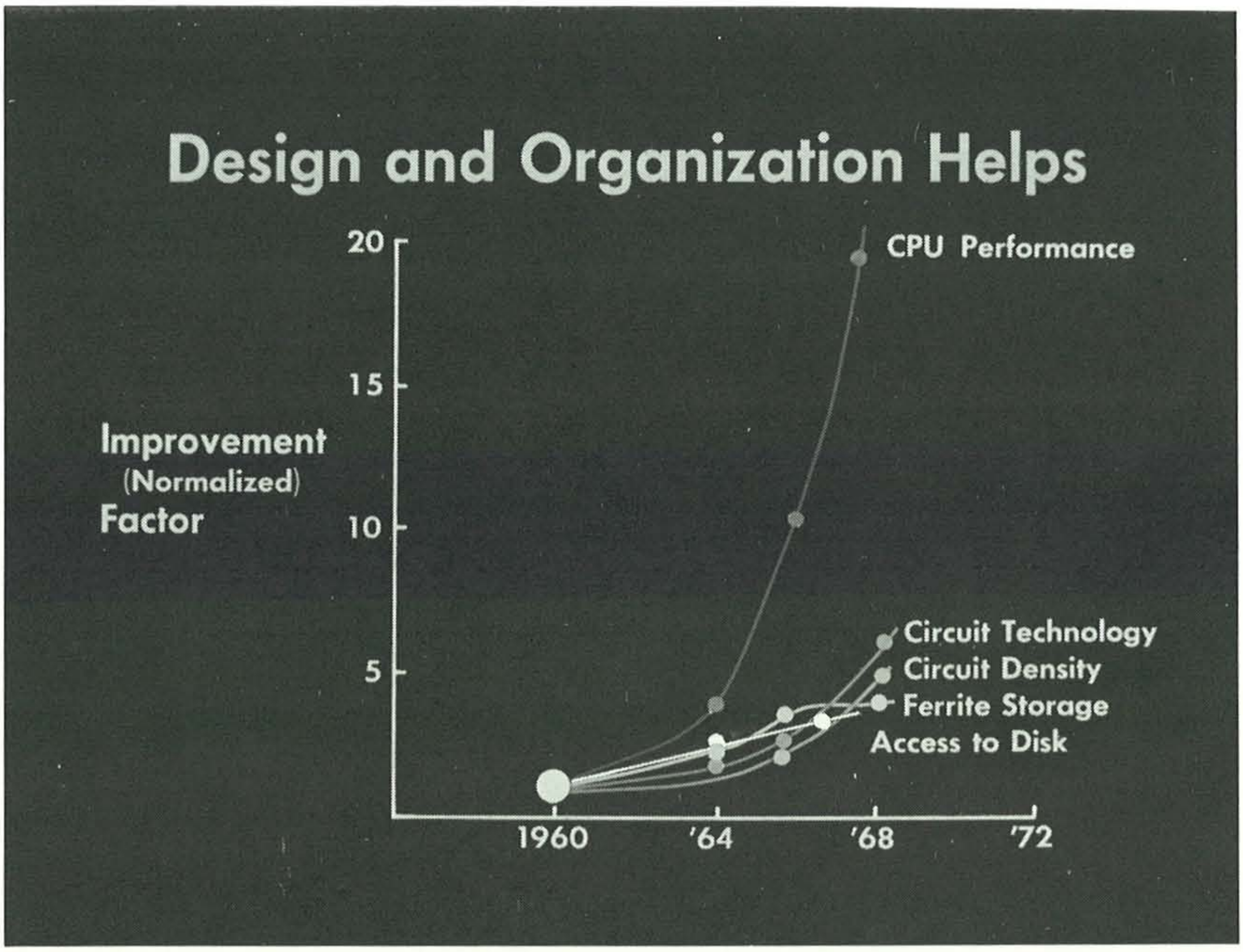


Figure 3

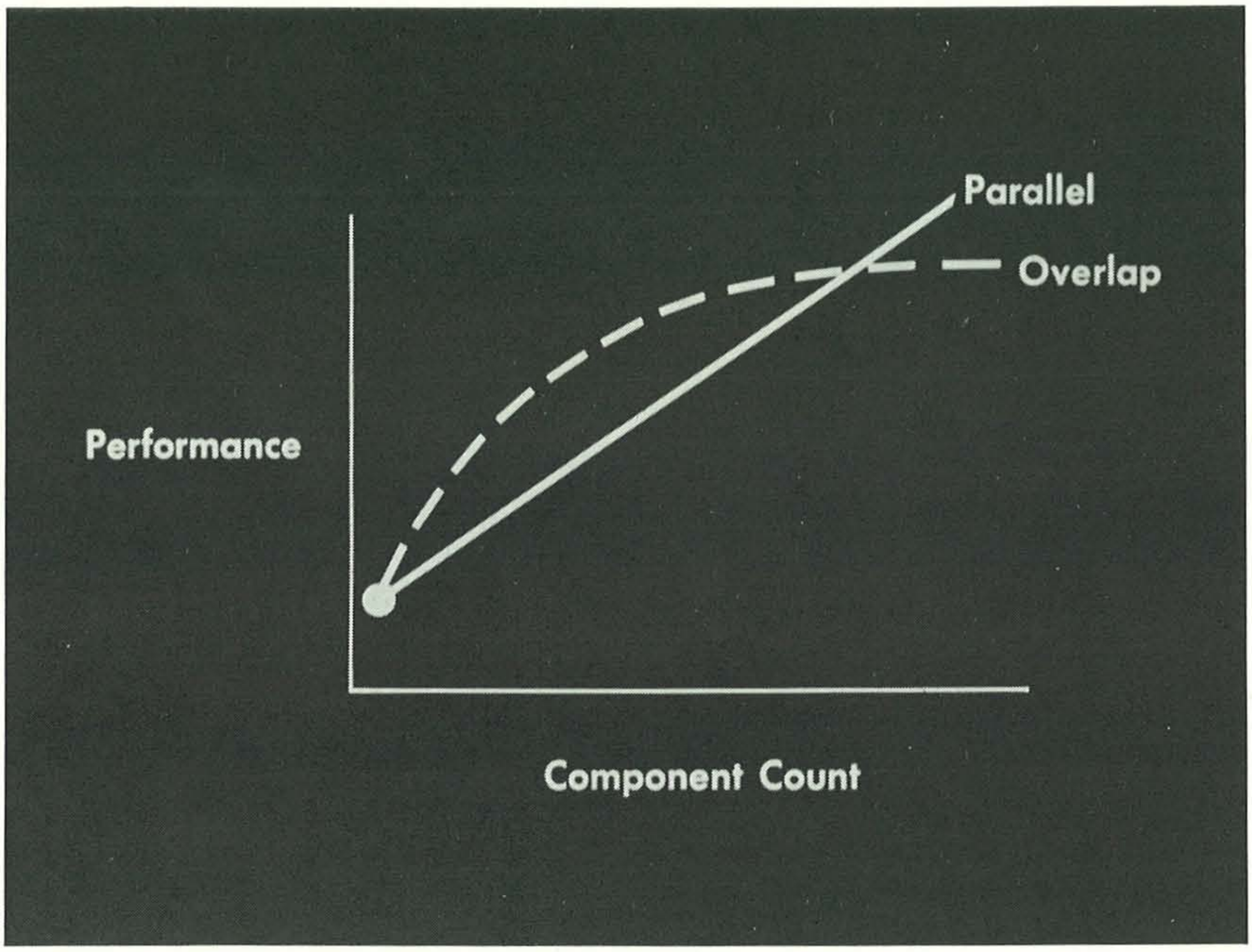


Figure 4

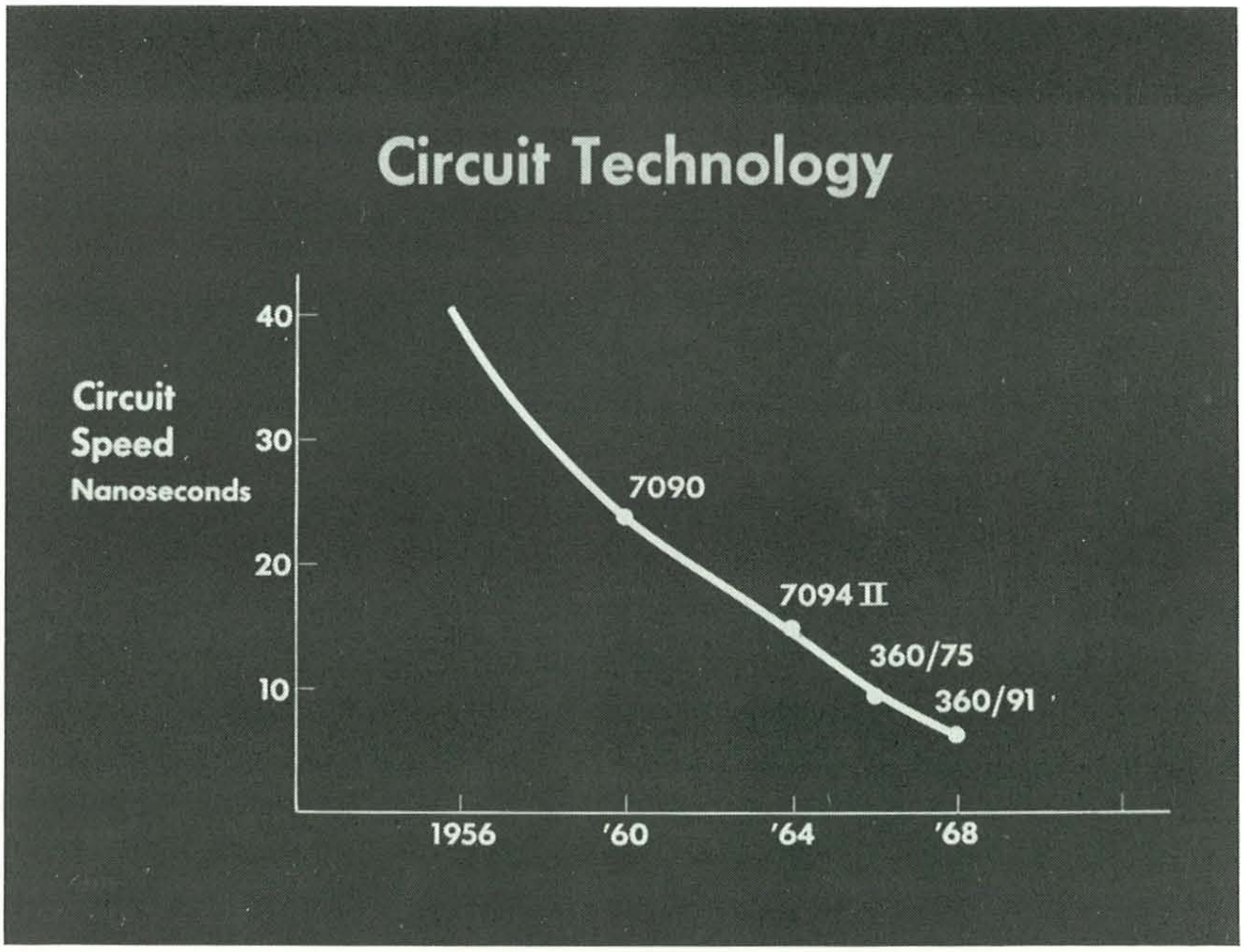


Figure 1

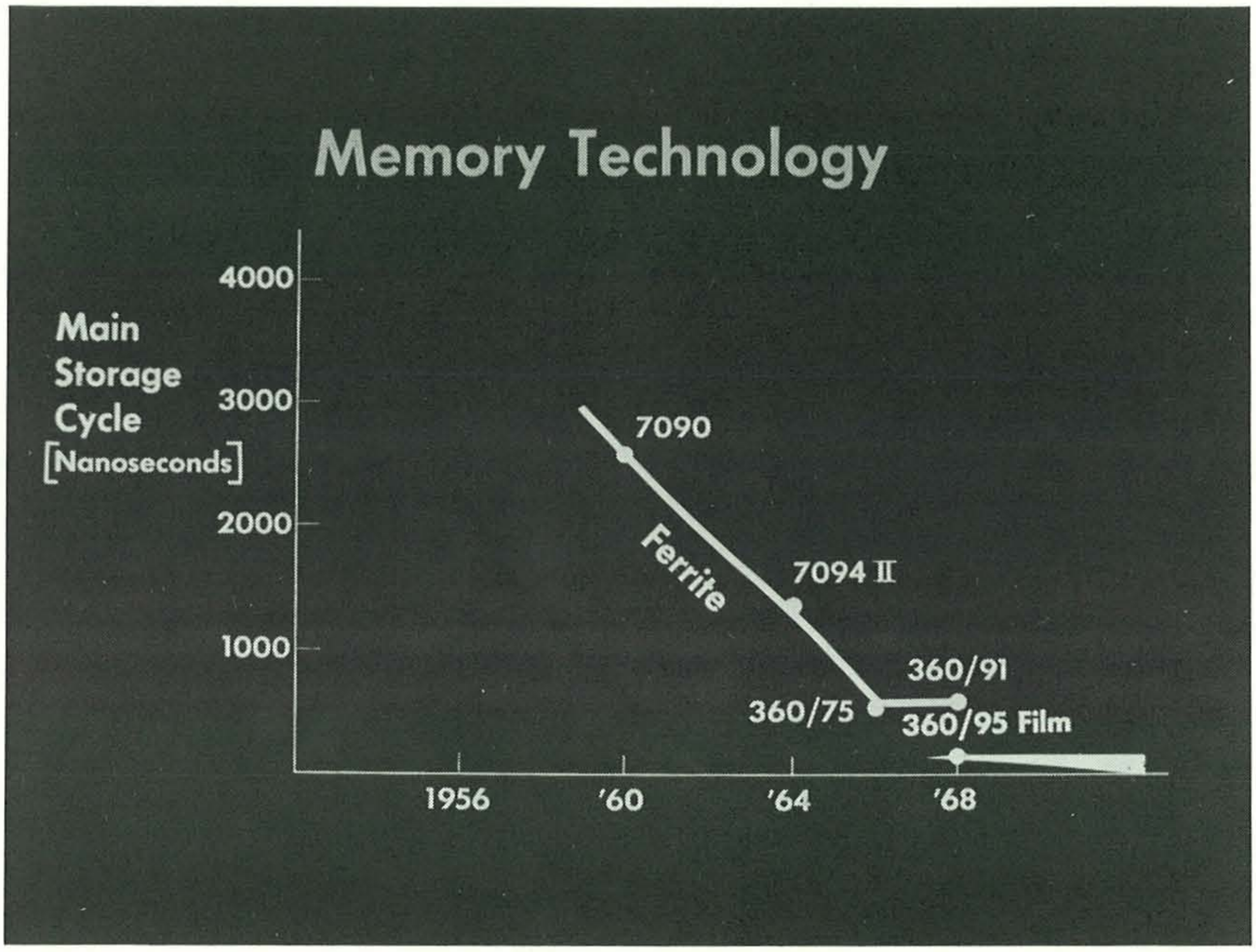


Figure 2

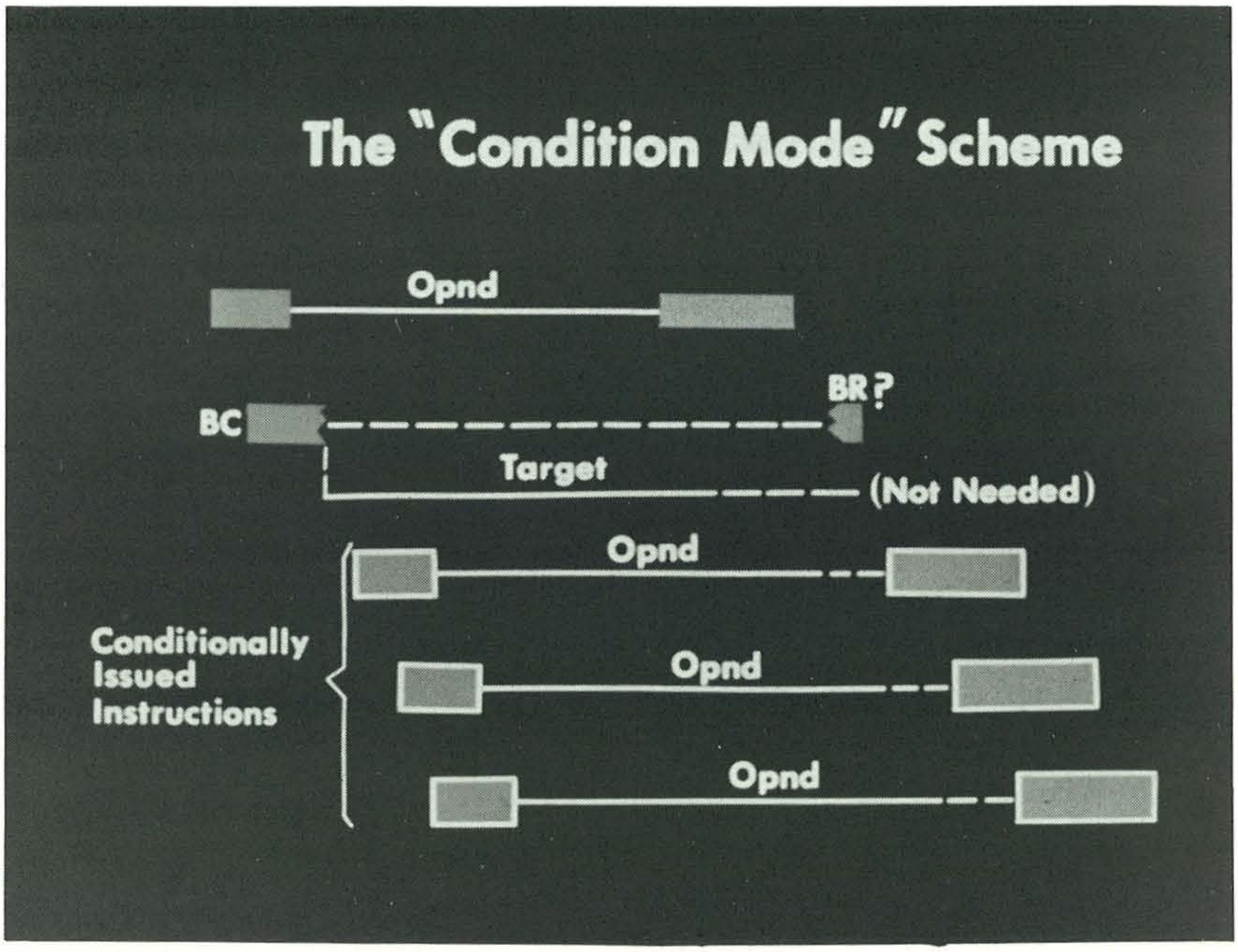

The "Condition Mode" Scheme
Opnd
BC
BR?
Target
(Not Needed)
Conditionally Issued Instructions
Opnd
Opnd
Opnd

Figure 10

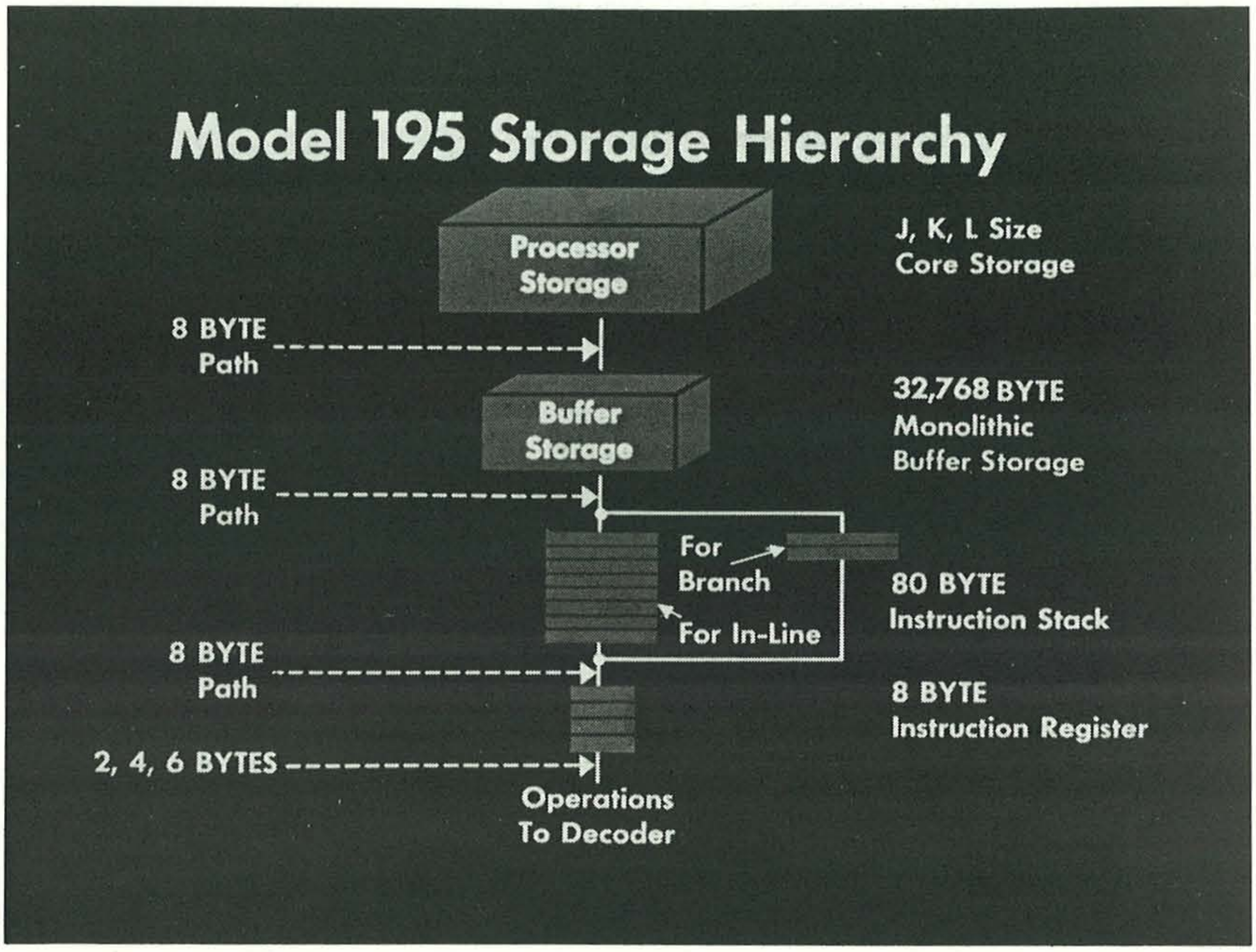

Model 195 Storage Hierarchy
Processor Storage
J, K, L Size Core Storage
8 BYTE Path
Buffer Storage
32,768 BYTE Monolithic Buffer Storage
8 BYTE Path
For Branch
For In-Line
80 BYTE Instruction Stack
8 BYTE Path
8 BYTE Instruction Register
2, 4, 6 BYTES
Operations To Decoder

Figure 11

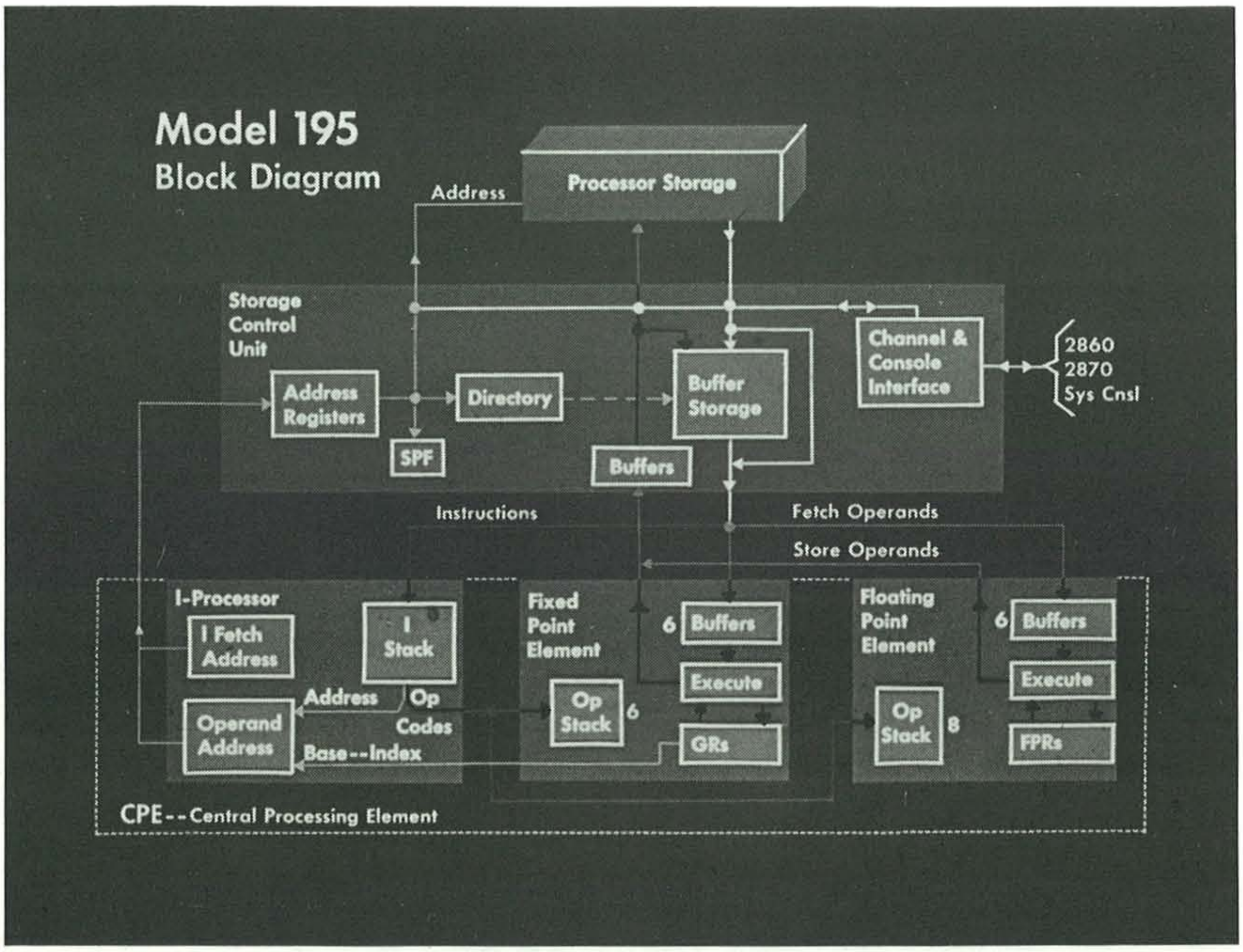


Figure 12

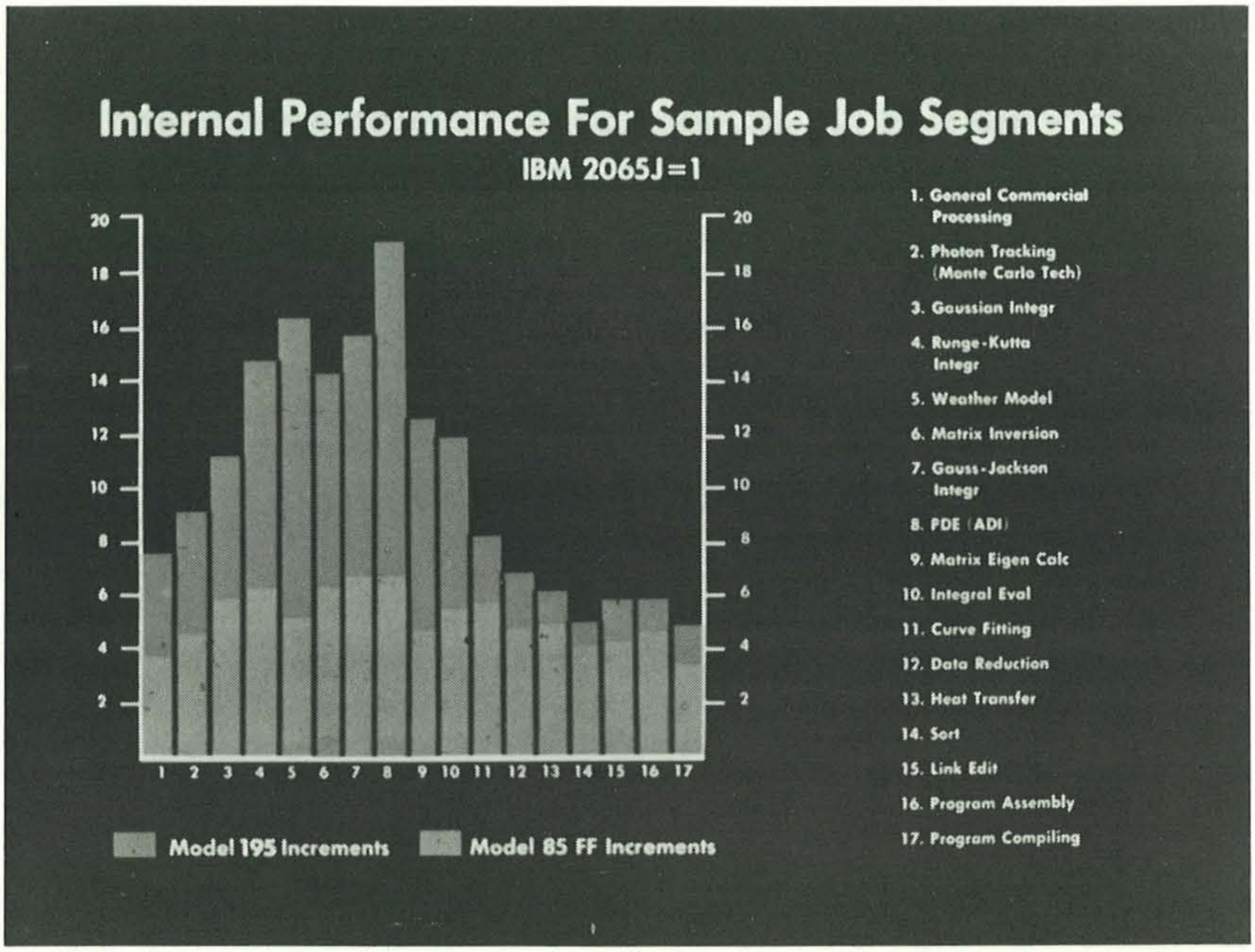


Figure 13

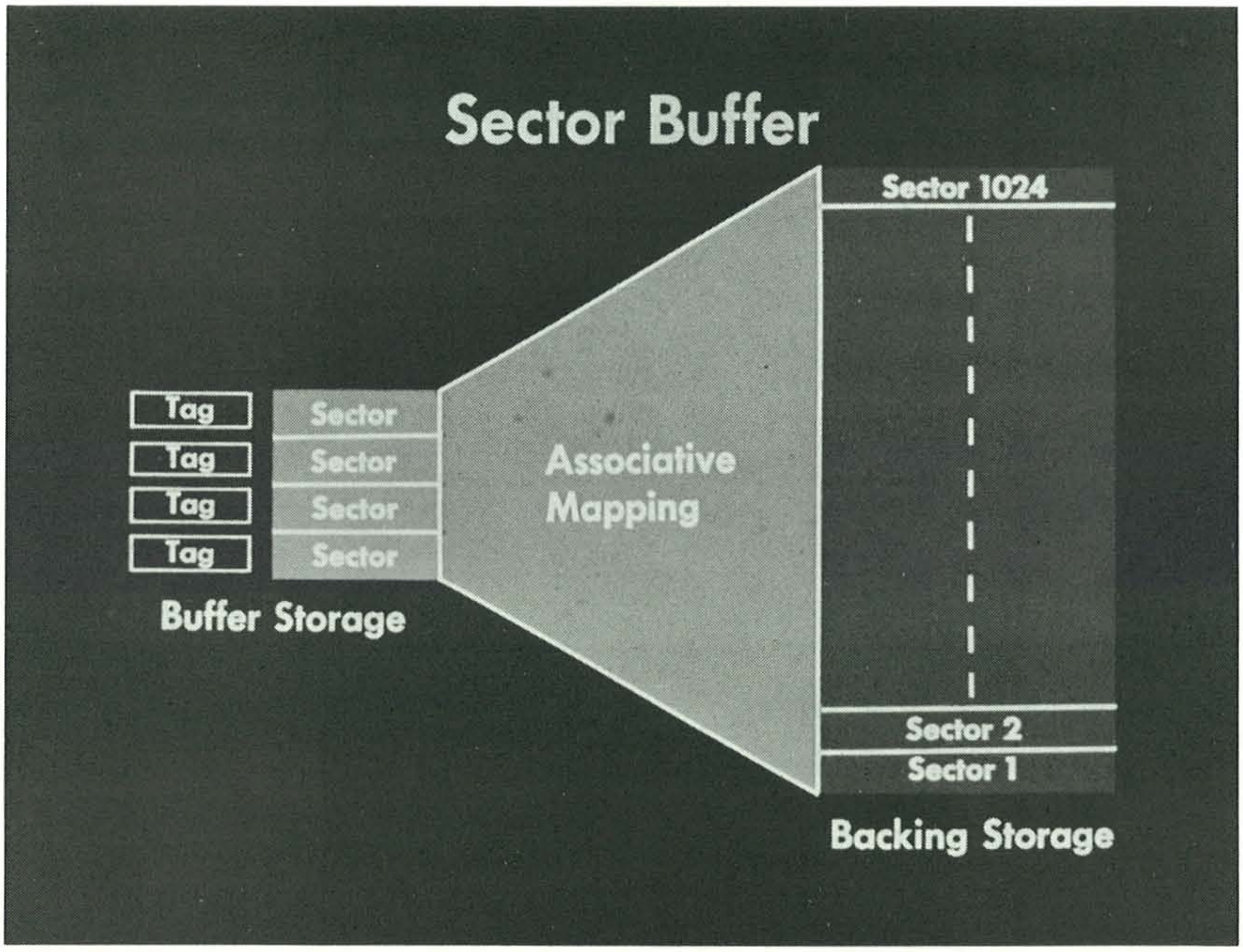


Figure 14

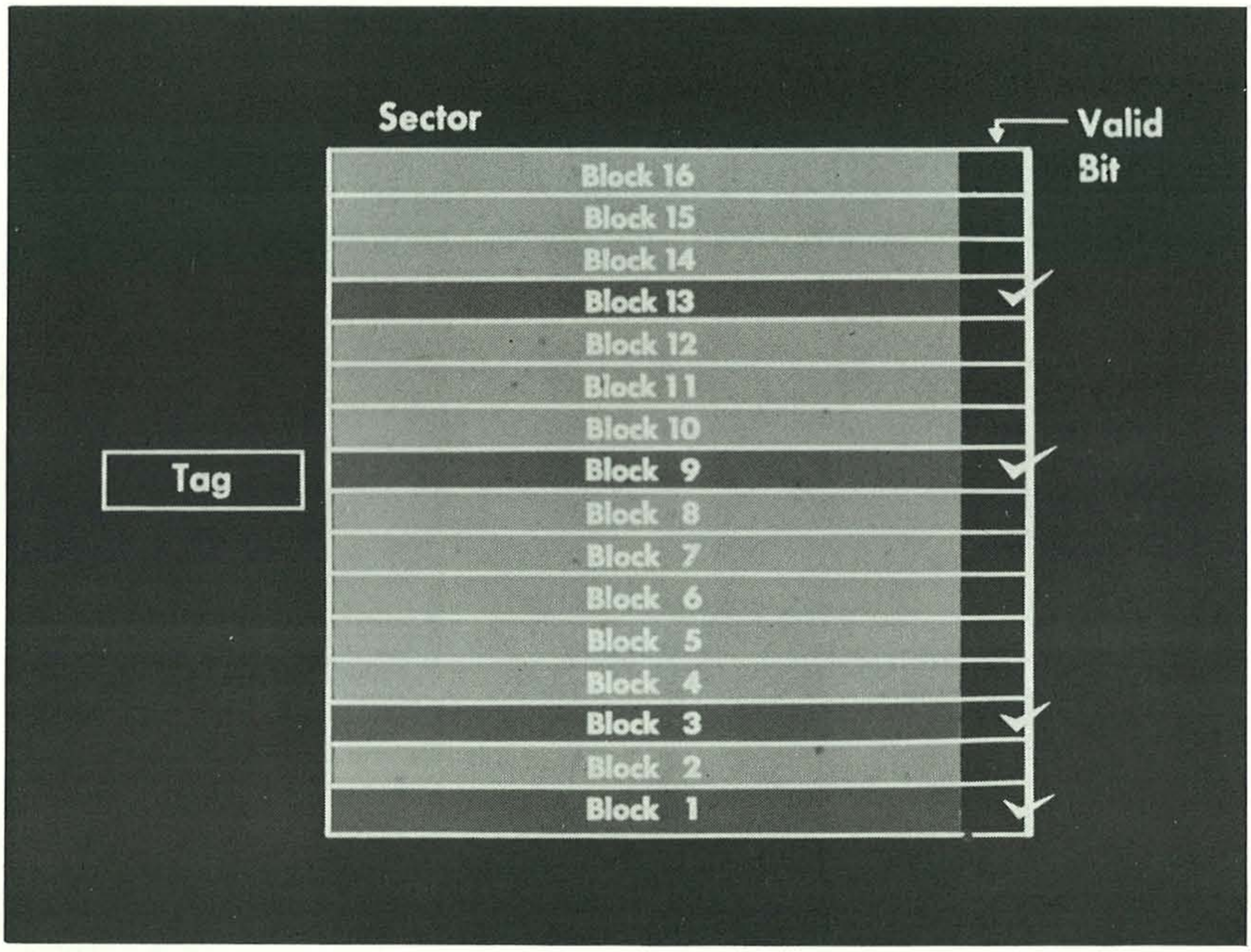


Figure 15

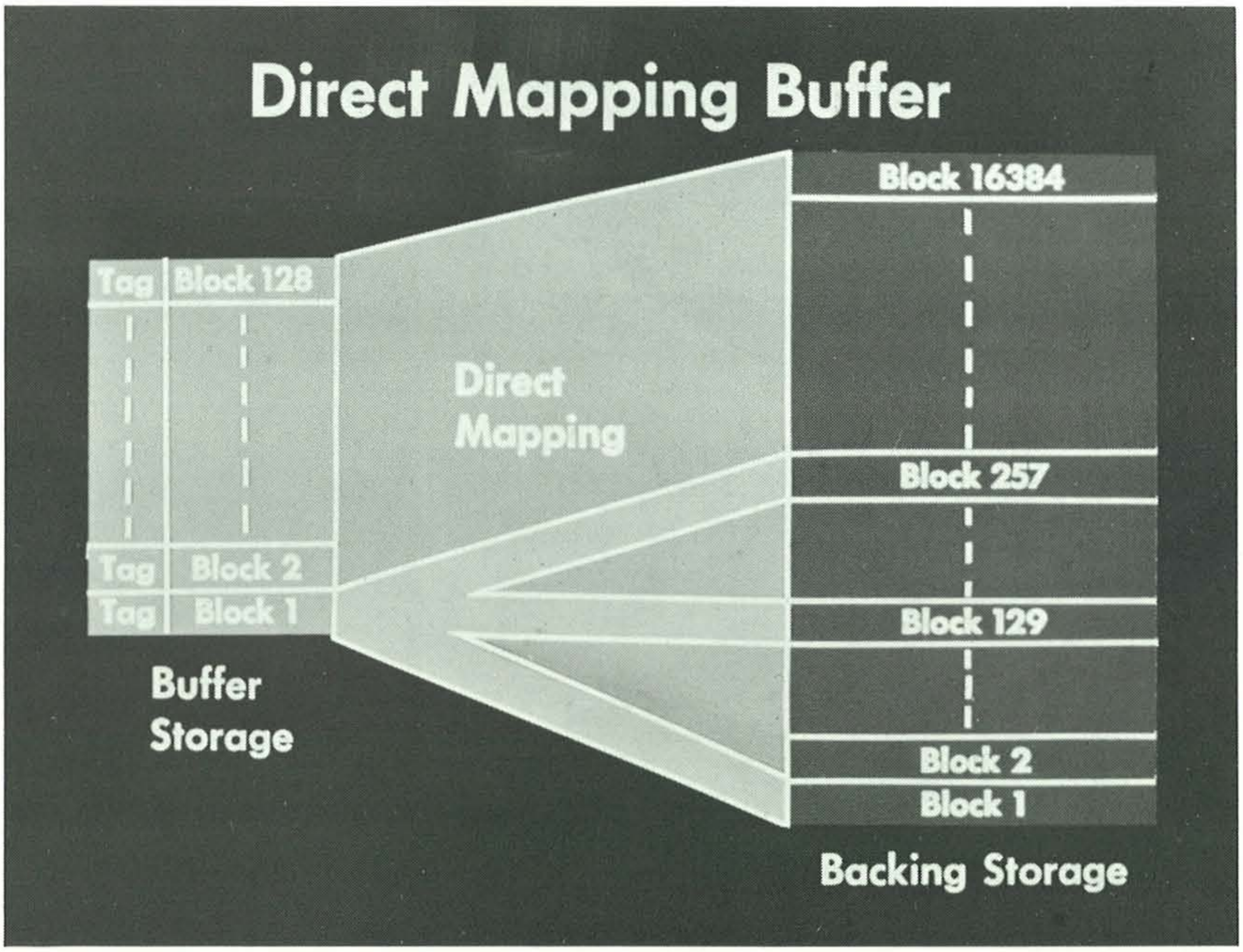


Figure 16

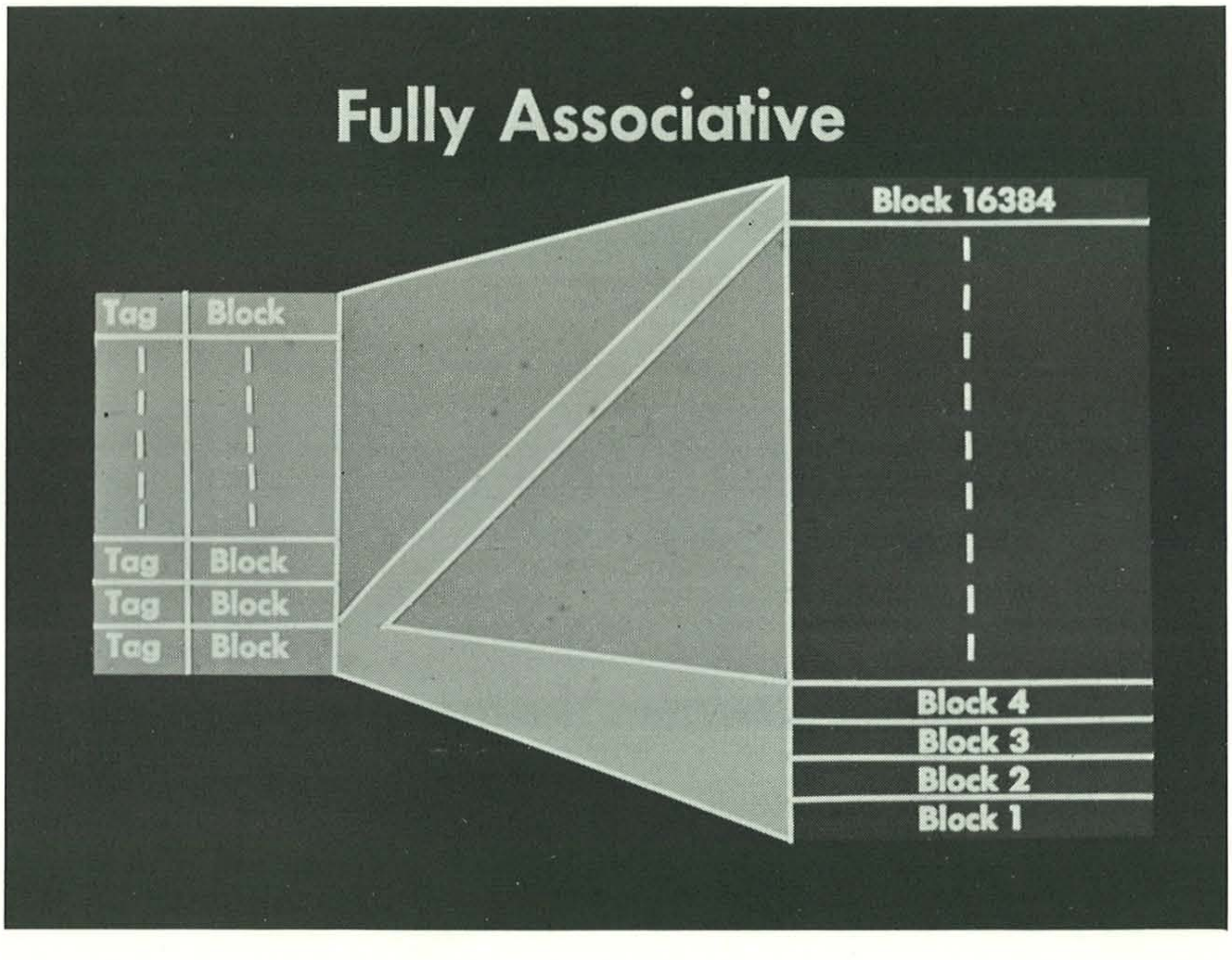


Figure 17

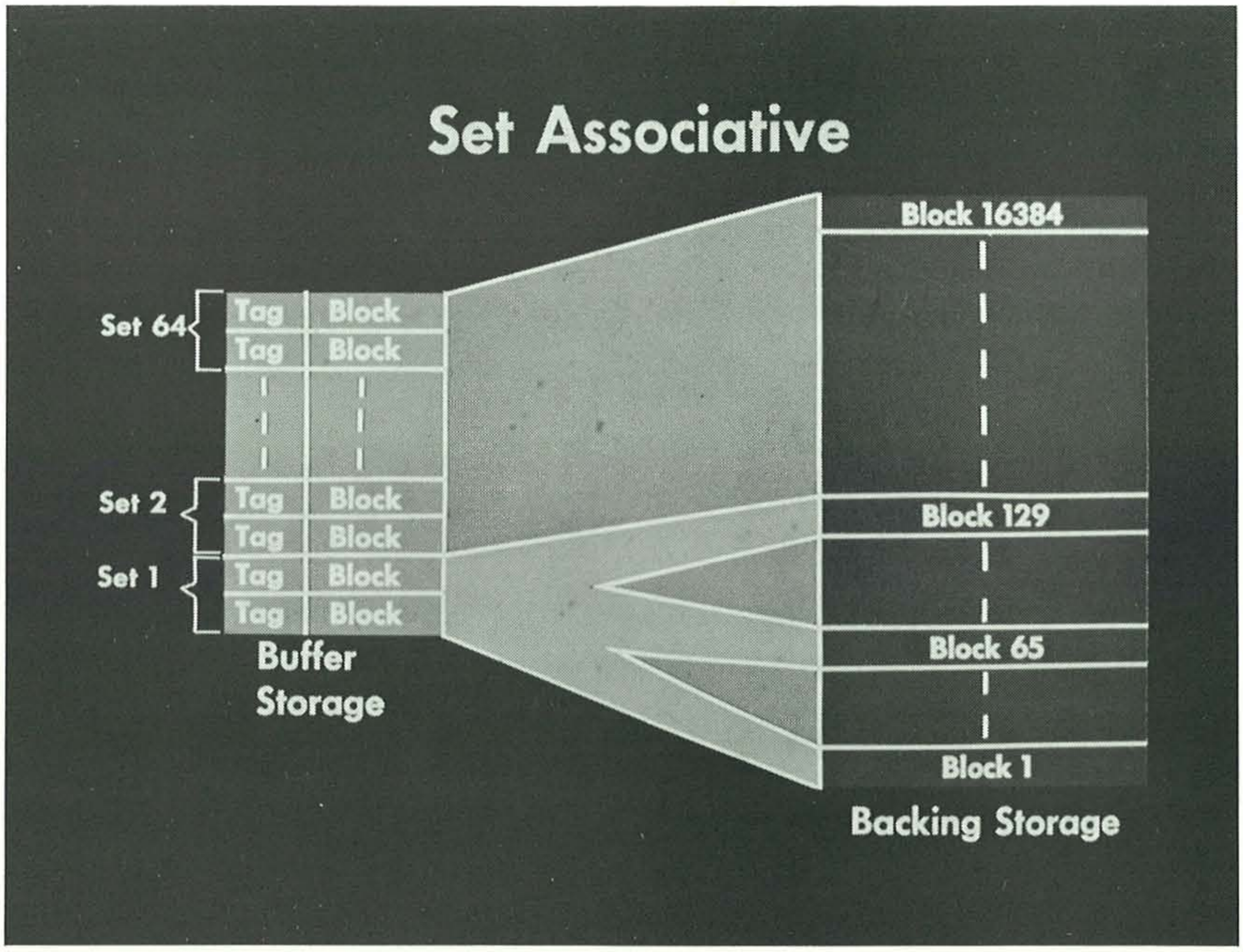


Figure 18

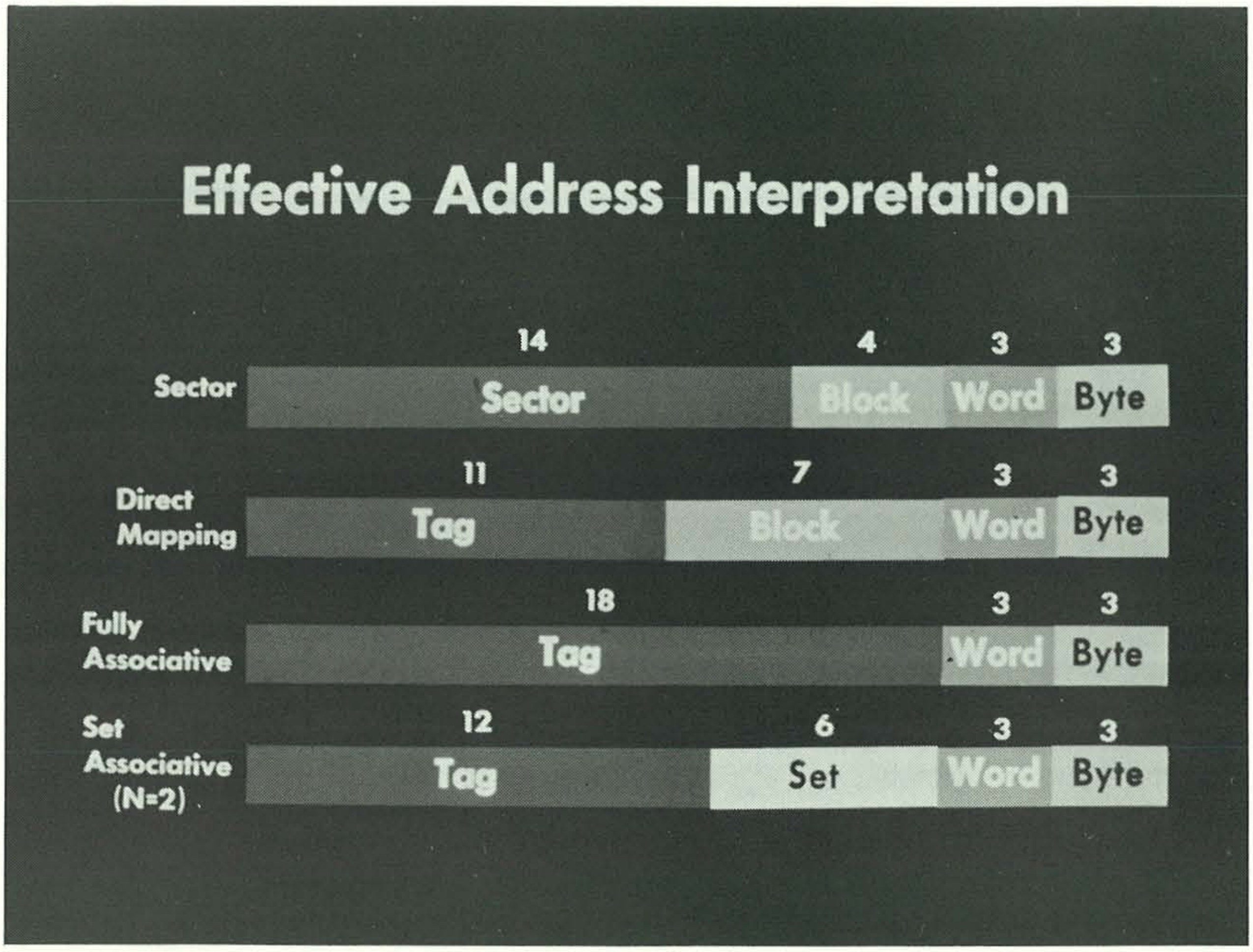


Figure 19

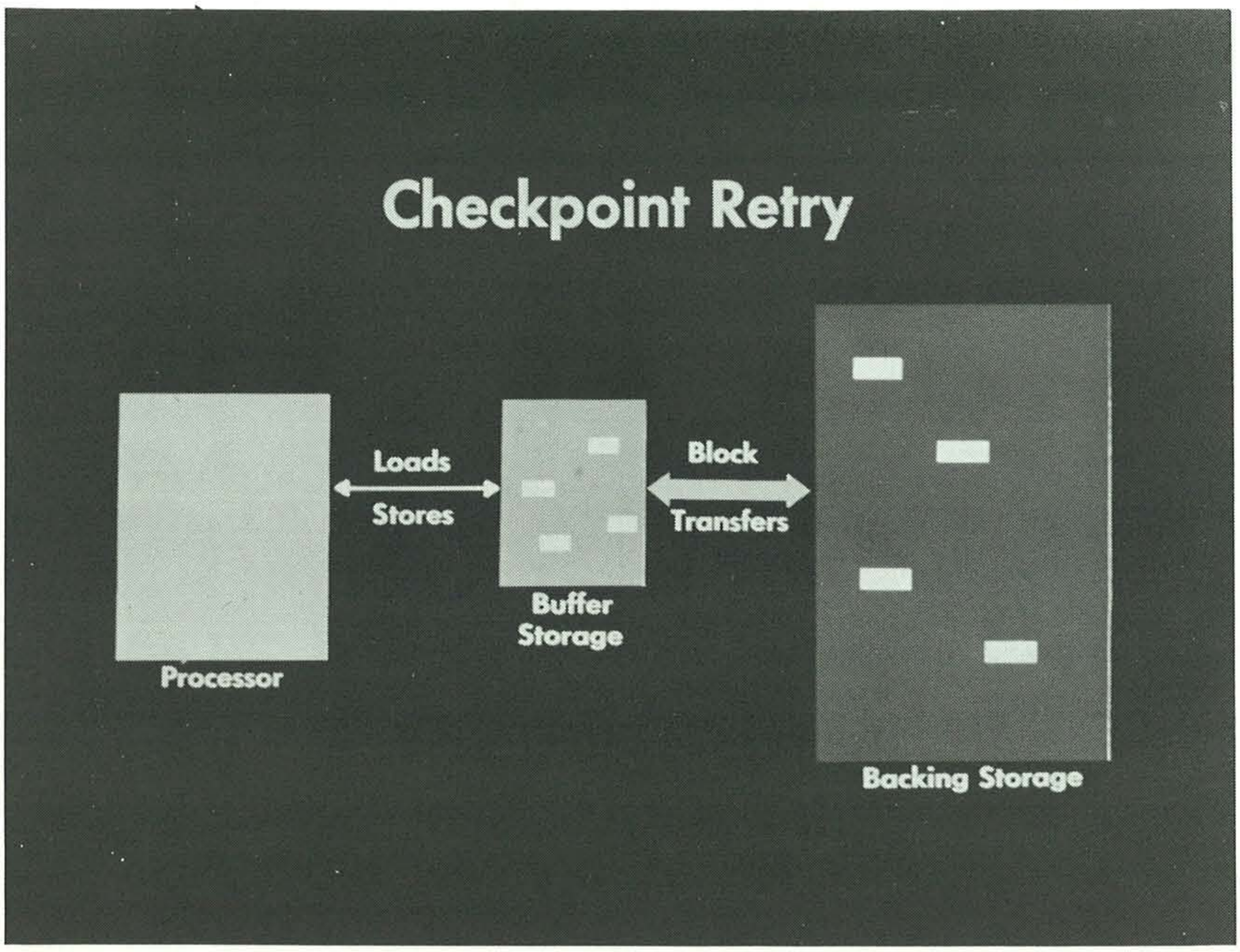


Figure 20

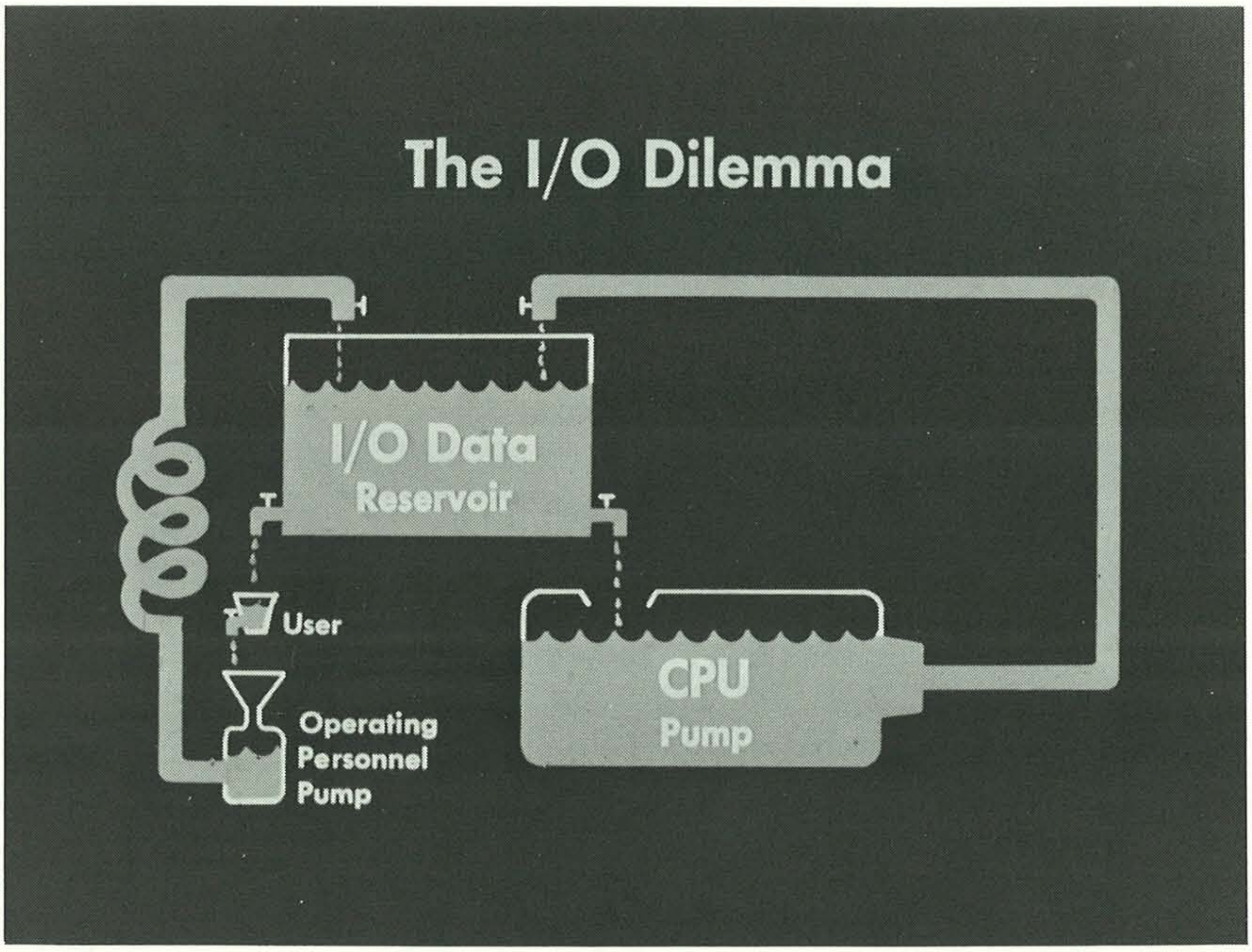


Figure 21