MTA Számitástechnikai és Automatizálási Kutató Intézete, **Közlemények** 15/1975.

# SCHEDULING ALGORITHMS IN CONTROL DATA'S NETWORK OPERATING SYSTEM (NOS)

K. Tantscher

In a modern operating system the two objectives of best utilization of computer resources and reasonable service for time sharing users (response time!), which are to a certain extent exclusive, create the necessity for effective scheduling algorithms.

All really implemented algorithms have to live in a certain environment – the given hardware. Since the hardware architecture of the CDC CYBER 170 Series is rather unique, we shall start with a brief description thereof.

Fig. 1 shows that a CYBER 170 System consists of a Central Processing Unit (CPU), a Central Memory (CM), at least 10 Peripheral Processors (PPU), at least 12 Data Channels and some peripheral equipment. The Extended Core Storage is optional and treated by the Operating System like any rotating mass storage.

The uniqueness of the system starts with PPU's which are independet processors with their own memory (4K 12bit bytes) and access to CM.

Their main purpose is to relieve the CPU from I/0 – work and operating system functions.

This leads to a system lay–out as shown in Fig.2.

Two PPU's are dedicated to the operating system: PPØ contains the MONITOR at all times. This is the routine which keeps track of all time dependent and periodic functions of the system. PP1 is dedicated to drive the CRT–Console,thus giving the operator the various system–status information in clear text.

The remaining 8 PP's form a pool and can be assigned on a demand basis.

According to Fig.3 the CPU is time shared over all active jobs in CM.

Figures 4, 5, 6 and 7 illustrate the Control Point Concept. Central Memory can be assigned in any size (no partitioning or paging!). For convenience in addressing the system assigns memory in increments of 100 words. (1 word = 60 bit, max. memory size = 262K words).

Each Control Point has an area of 200 words in low core which contains status and accounting information of the job presently running at this Control Point. The "Exchange Jump" instruction which is used to switch the CPU from one to another job saves the register contents in the first 16 words of this "Control Point Area" in low core. In case all Control Points are occupied the system has the possibility to swap the field length of a job together with all his information contained in low core to disk in order to free space for a higher priority job.

Figures 8 – 11 show the external effects of the scheduling system on the example of one job.

As jobs are read in (e.g. from a card reader), they are placed in the INPUT–Queue on disk with an initial or entry priority. An aging routine increases this priority according to a given aging rate. After some time, the priority will be higher than the priority of presently running jobs and the job will be scheduled for execution, that means he will be assigned to a Control Point and loaded into Central Memory. There he will stay with the upper bound priority of the INPUT–Queue until his allowed CM–time slice or CPU–time slice has elapsed. At this point in time his priority will be lowered to the lower bound priority of the INPUT– Oueue. This makes him a candidate for being swapped out. When the scheduling program decides that there are other, higher priority jobs to run, it will initiate the swap out of this job. Now he enters the ROLLOUT–Queue (only the first time!). The aging begins again until the schedule decides to swap him in again and so forth.

The reason to give a job the lower bound priority of the INPUT–Queue the first time he enters the ROLLOUT–Queue, is to give short jobs a shorter turnaround time.

Fig.12 shows that there are 5 classes of jobs depending on their origin. For each class the priority parameters for the queues, the ageing rates and the service parameter as CPU time slice and CM time slice can be set independently. The setting of the parameters is done at assembly time of the system but they can be changed by the operator in the running system. Additional "Delay Parameters" control how often scheduler and priority aging program have to be called.

Summarizing the facts up to now we can see that there is a dynamic scheduling of jobs being in INPUT and ROLLOUT–Queue in concurrency with active jobs.

One slightly different strategy is used in the case of interactive jobs performing output–operations. To free memory from these jobs the output is done from disk rather than from a memory buffer. That means an interactive job which has used his time slices and has output data available for the terminal is swapped to disk regardless of the load of the system and the priority level of other jobs. A special PP–routine will then route the output from disk to the terminal.

All these operations described are performed by routines which are dynamically loaded from disk to any free PPU.

The scheduler (1SJ) can be called by several other routines after certain events which have managed the system status. Monitor calls him at least every time his delay period has elapsed (Fig.13).

Figures 14 and 15 give an overview of the schedular folw.

First thing 1SJ does is to check if the priority aging routine 1SP has to be called. If yes, 1SP will increase the priorotes of all jobs in the queues and after completion call 1SJ back.

The next task of 1SJ is to do necessary housekeeping functions of the active jobs. That is mainly to establish a memory map.

If there is any active job requesting more storage we try to fulfil the request. If not, we search the queues for a possible candidate to swap in. This is done by algorithm 1 (Fig.17).

Note that on occurrance of equal priority jobs the one with the bigger memory requirements is preferred ("best fit").

If there is no candidate to be found, 1SJ will perform the EXIT functions (Fig.16).

If there is a candidate, we try to satisfy his memory requirements. Either there is already enough free memory then 1SJ branches to SCJ4, or it tries to free space using algorithm 2. (Fig.18). The interesting fact of this algorithm is that 1SJ searches the active jobs starting with lowest priority (TACP is sorted in descending order of priorities) and collecting their used space. If he found enough but too much, he searches once again downwards the list if one of these lower priority jobs could live in the excess memory, so it would not be necessary to swap him. This, at that point very simple task, ensures best memory usage while avoiding unnecessary swaps.

Starting at SCJ4 finally 1SJ tries to find a Control Point for the candidate to swap in using algorithm 3 (Fig.19). This algorithm looks for a best fit Control Point with minimum swaps or stroage moves.

The EXIT functions of 1SJ try to schedule a candidate (by repearting the whole thing if there was no success the first time), but also to keep scheduling activities in reasonable limits.

The system default values are 1 second recall time for the schedular and 16 seconds for the priority aging routine.

## Összefoglaló

### A CDC NOS – operációs rendszerének scheduling algoritmusai

K. Tantscher

A dolgozat ismerteti a CDC Network Operating System scheduling algoritmusait. Ezek az algoritmusok új, elméleti tudományos eredményeken alapulnak, és egyidejüleg törekednek az erőforrások hatékony kihasználásán és a minél rövidebb kiszolgálás biztositására.

## Резюме

### Алгоритмы расписании операционной системы CDC NOS

К. Танчер

В работе дается обзор об алгоритмах расписании NOS. Эти алгоритмы основываются на новых теоретических научных результатах и одновременно они стараются использовать эффективно ресурсы, и обеспечивать наиболее кароткое обслуживание.

CDC CYBER 170 SERIES

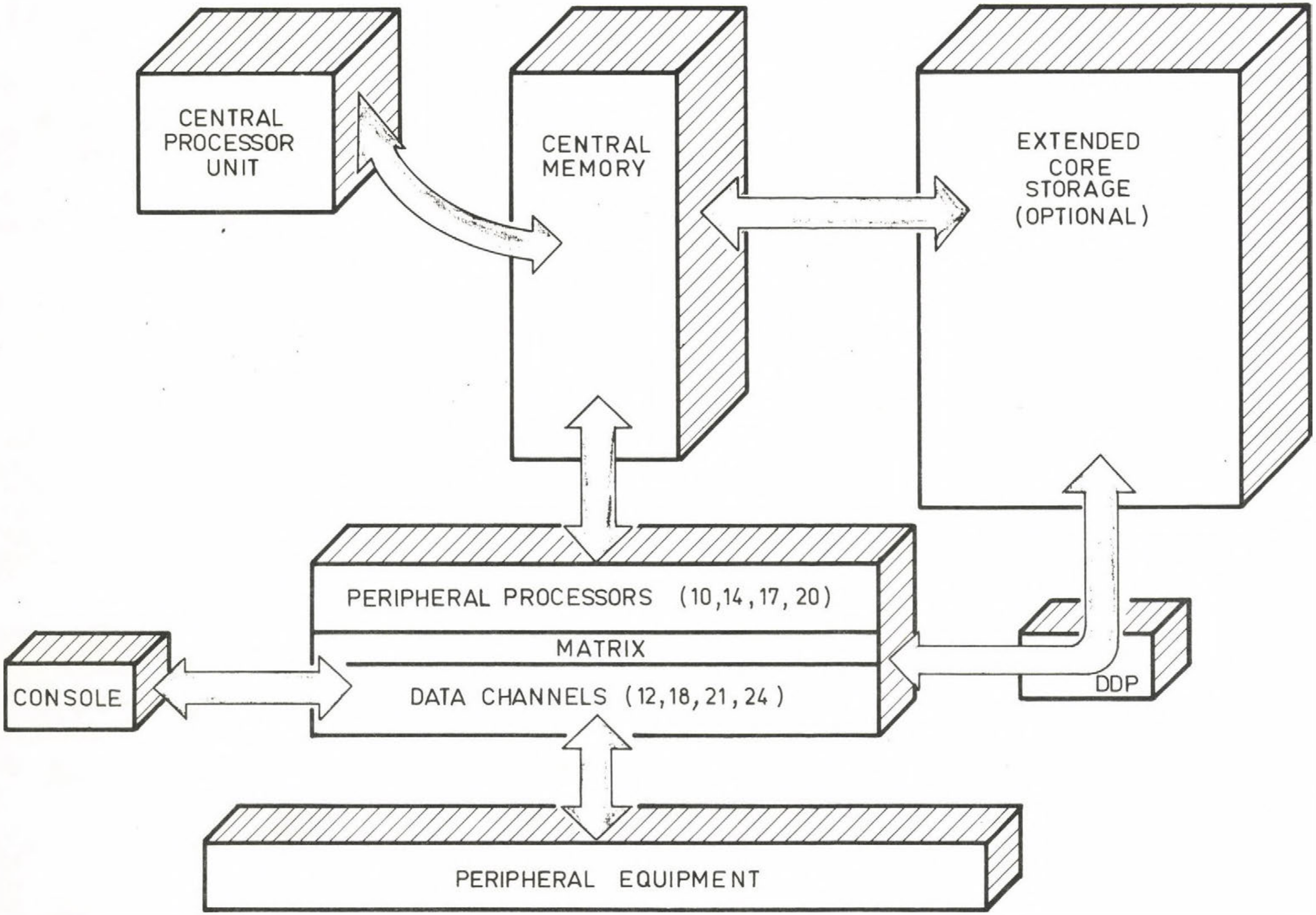

CENTRAL PROCESSOR UNIT
CENTRAL MEMORY
EXTENDED CORE STORAGE (OPTIONAL)
PERIPHERAL PROCESSORS (10,14,17,20)
MATRIX
DATA CHANNELS (12,18,21,24)
CONSOLE
DDP
PERIPHERAL EQUIPMENT

Fig.1.

# SYSTEM LAY-OUT

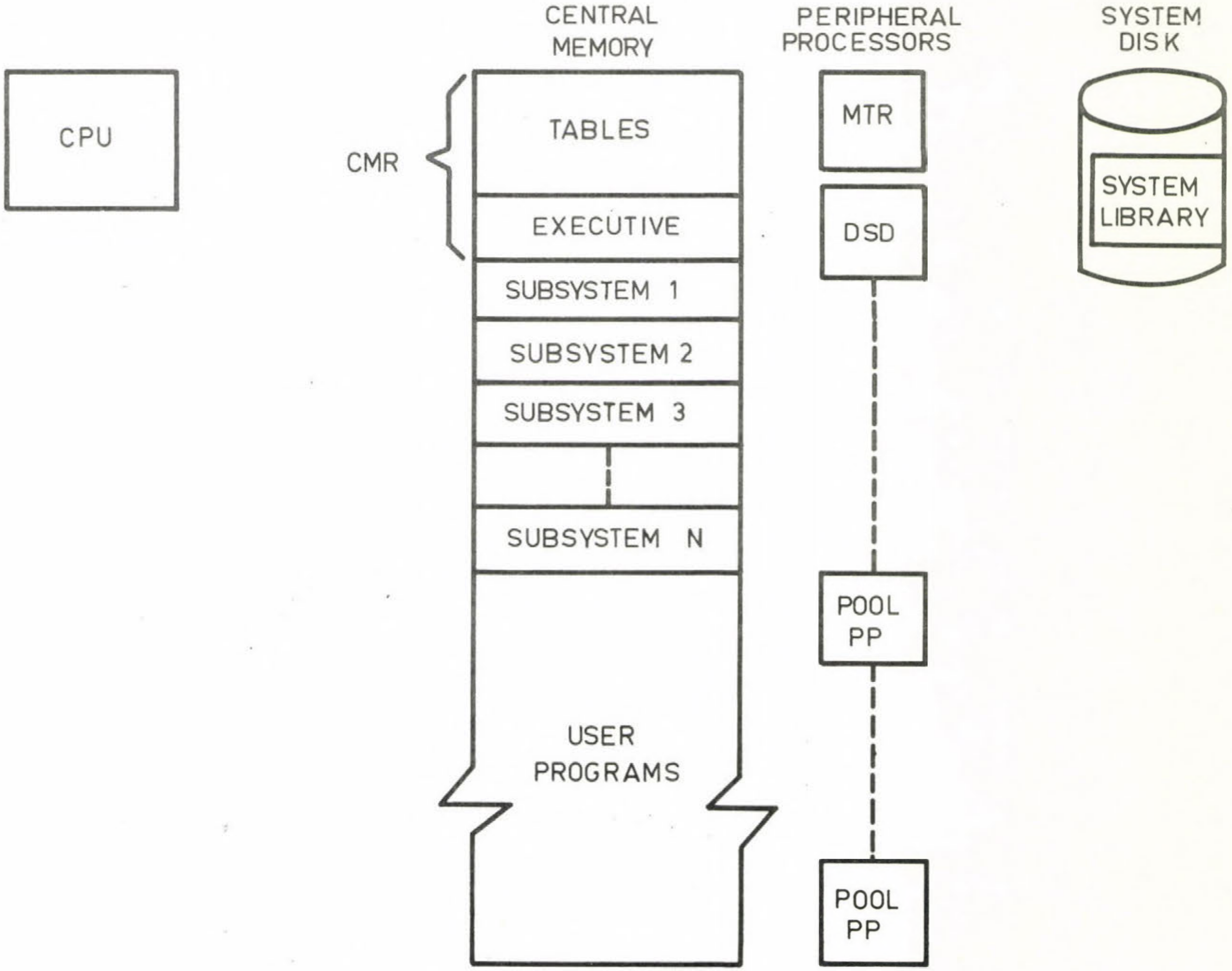
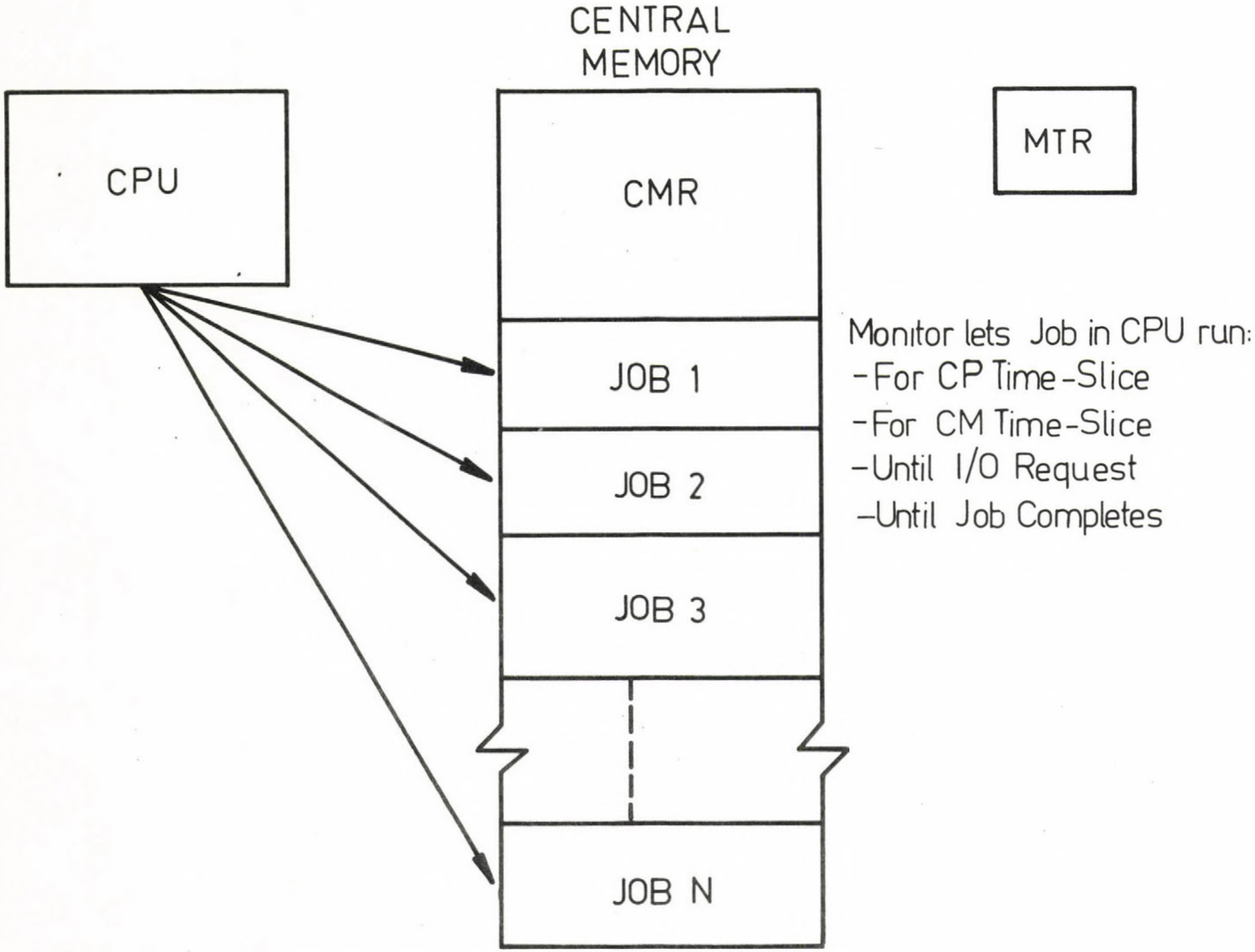


Fig.2.

# MULTI - PROGRAMMING



Monitor lets Job in CPU run:

- For CP Time-Slice
- For CM Time-Slice
- Until I/O Request
- Until Job Completes

Fig.3.

# CONTROL POINT CONCEPT

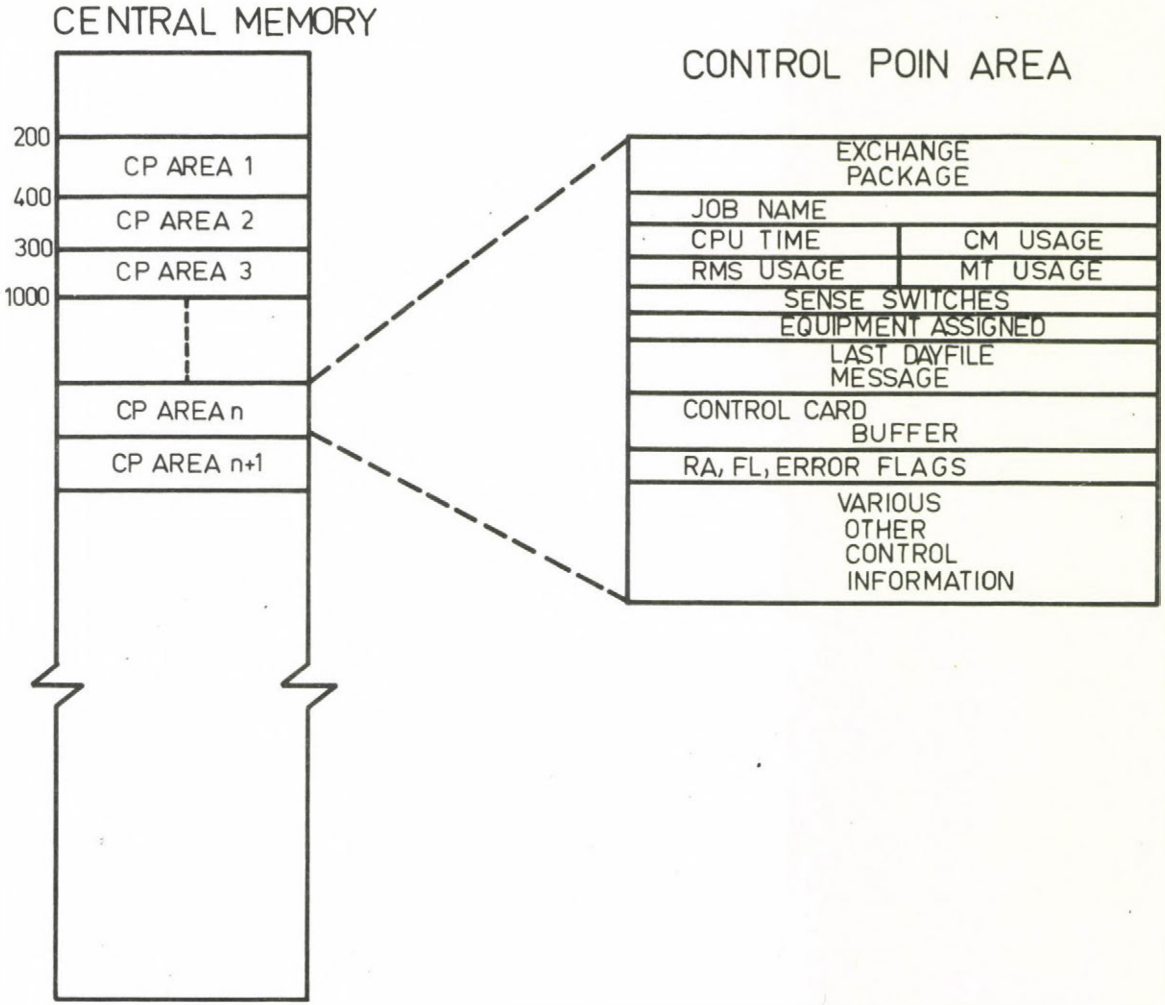


Fig.4.

# MEMORY PROTECTION

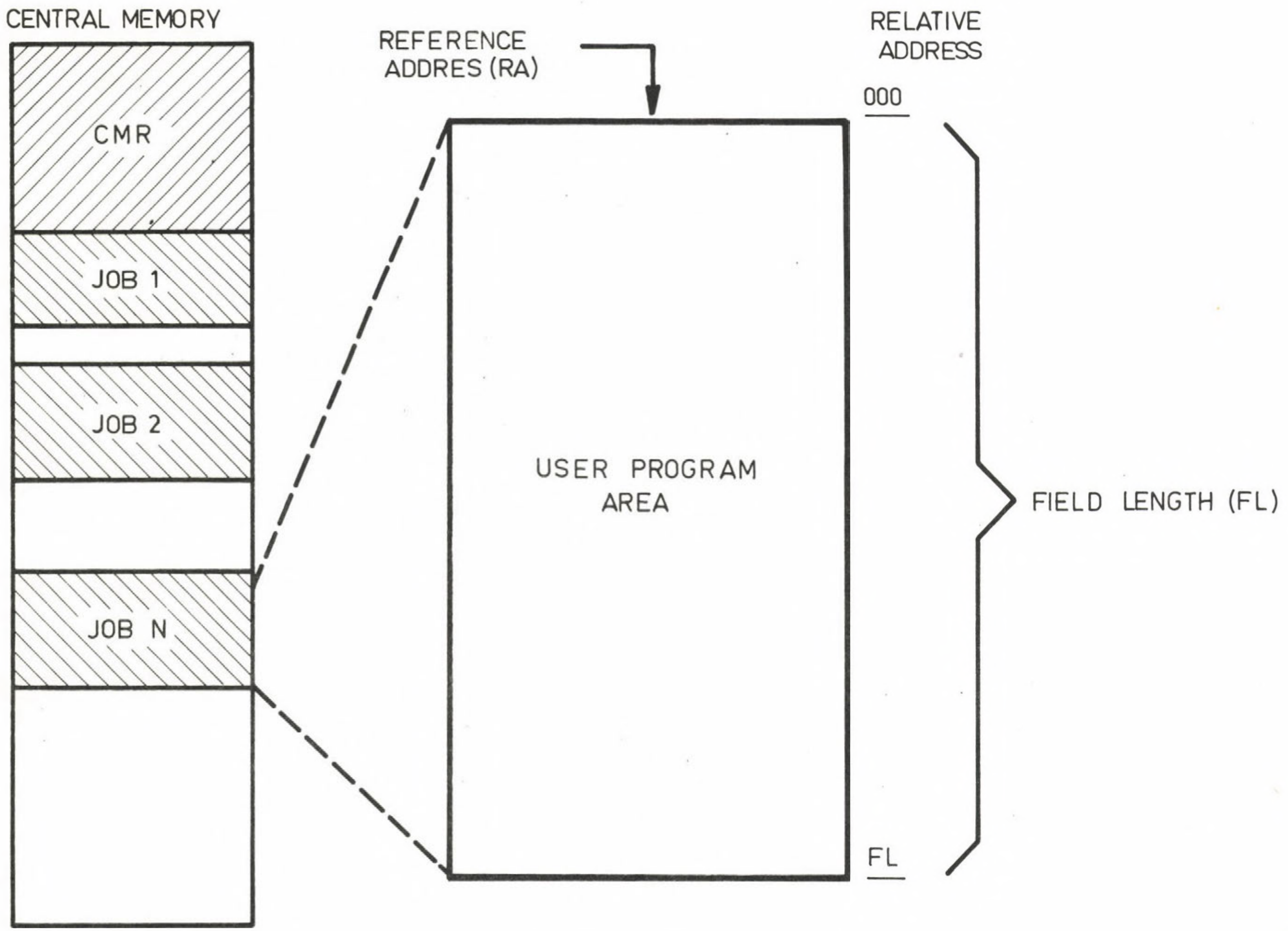


Fig.5.

## STORAGE MOVES

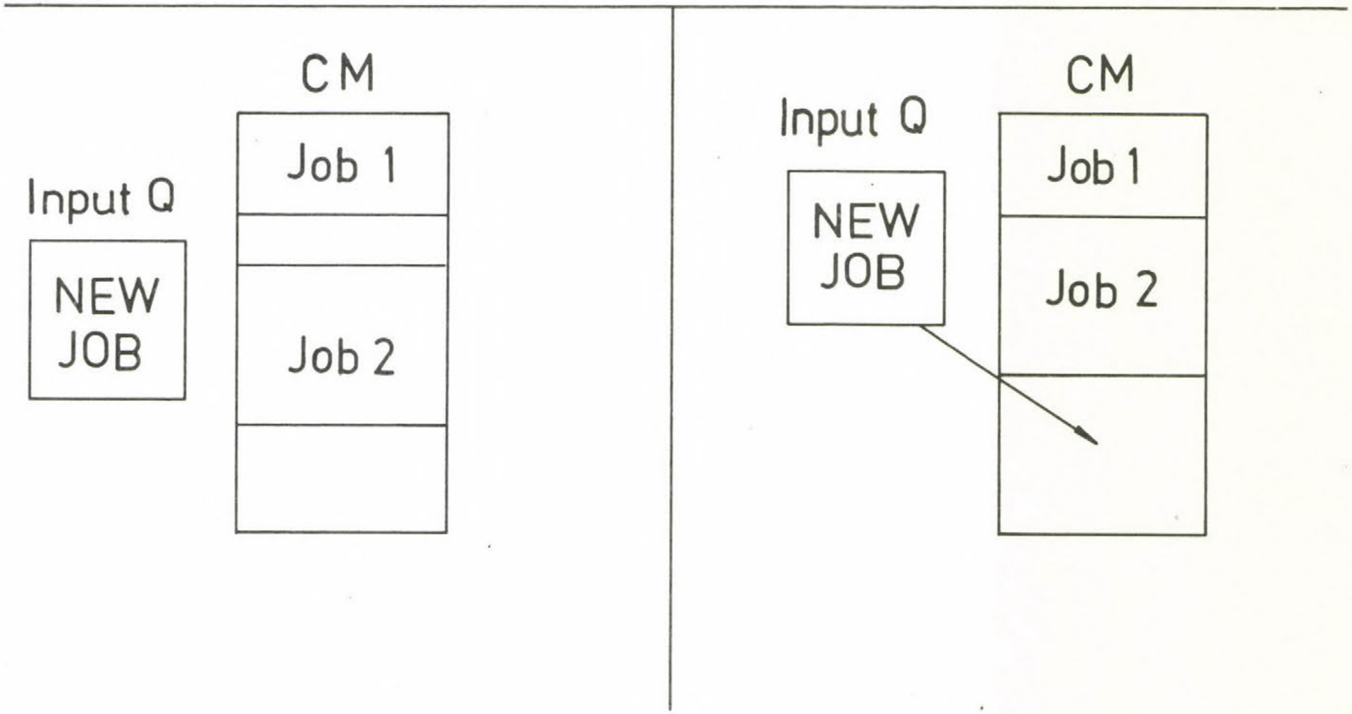


Fig.6.

# SWAPPING

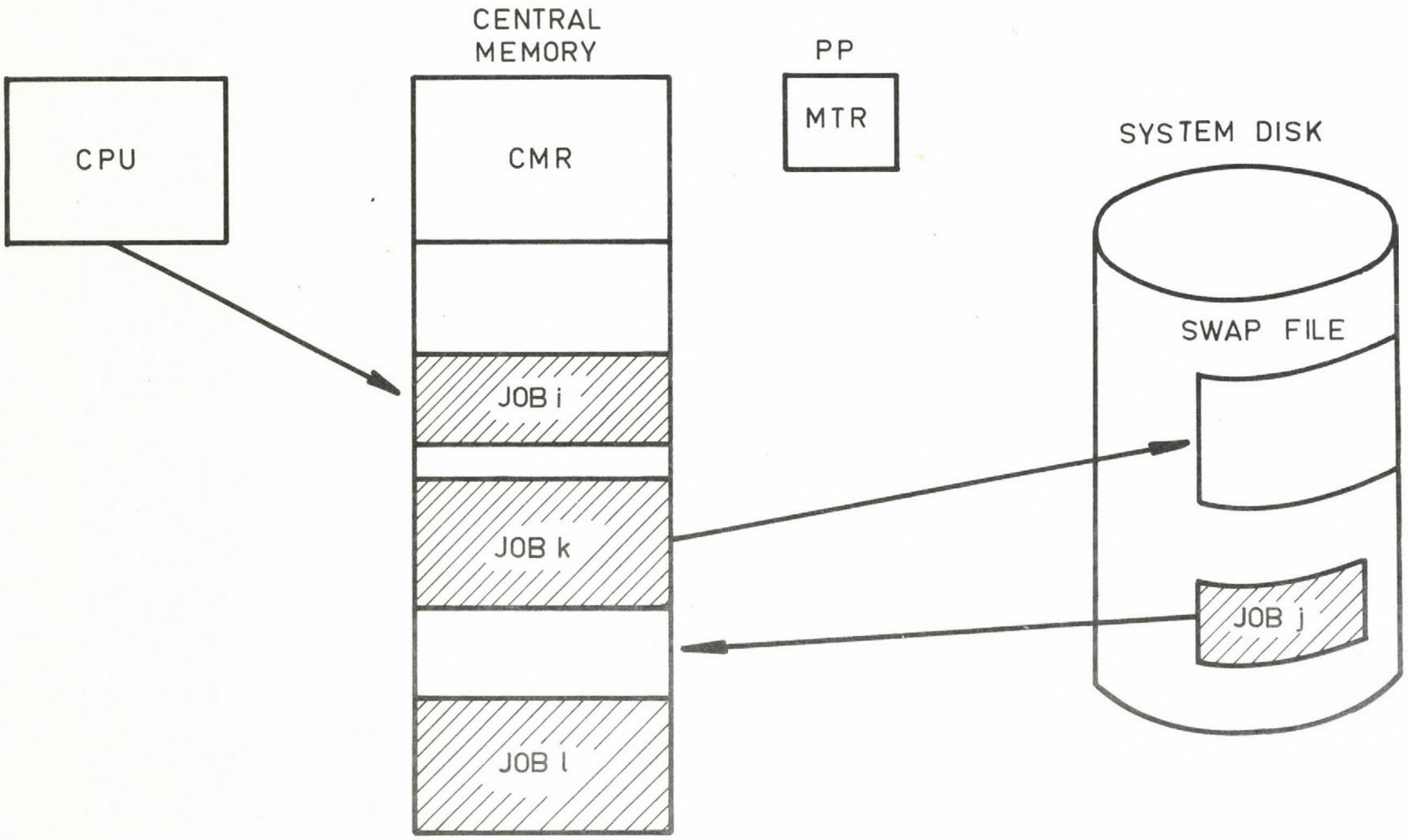


Fig.7.

# JOB SCHEDULING-ENTRY INTO SYSTEM

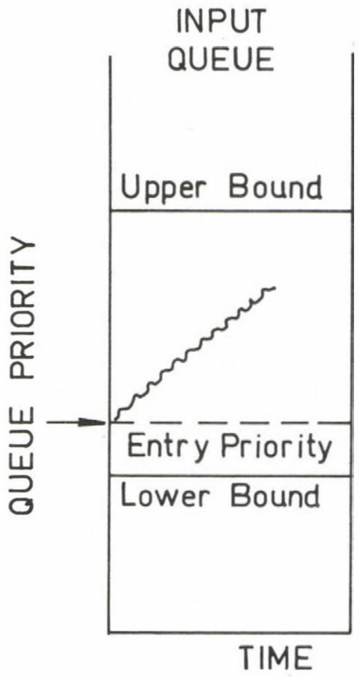


Fig.8.

# JOB SCHEDULING-ENTRY INTO A CONTROL POINT

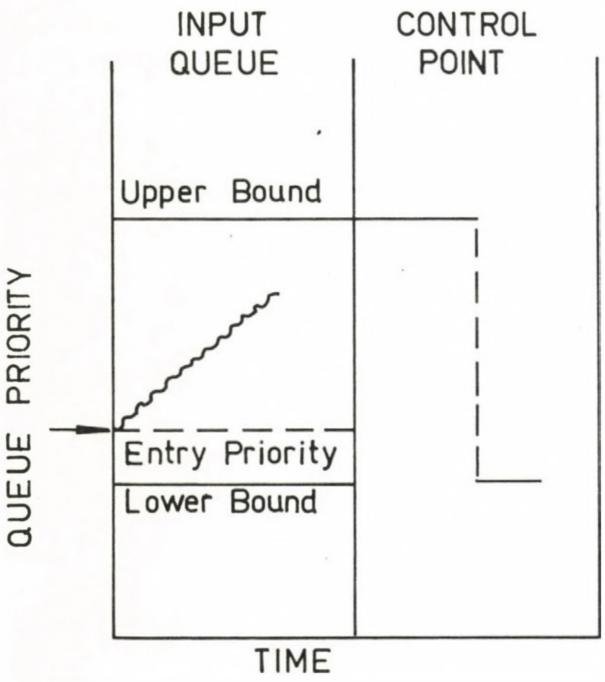


Fig.9.

# JOB SCHEDULING-ENTRY INTO ROLLOUT QUEUE

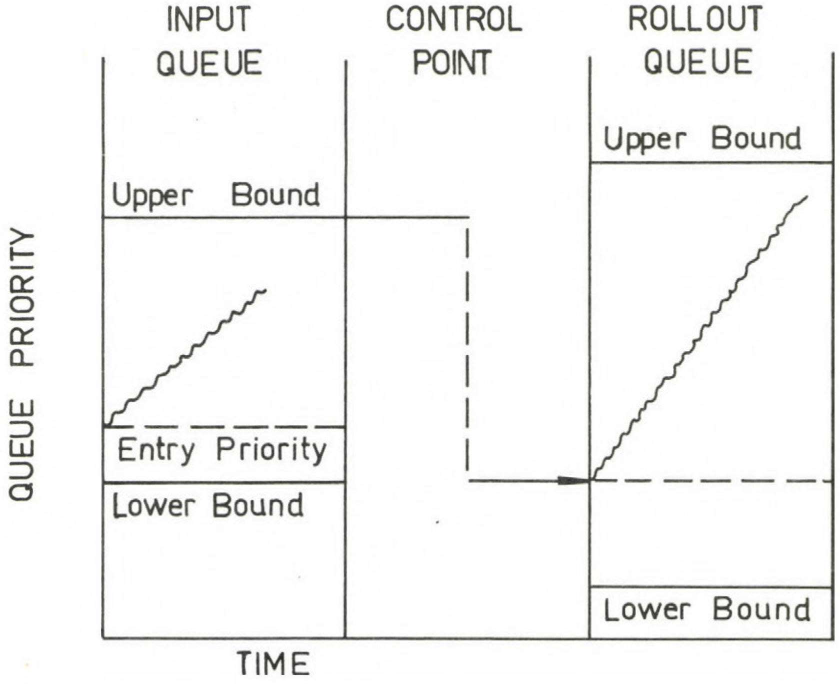


Fig.10.

# JOB SCHEDULING –

# LEAVING AND RE-ENTERING ROLLOUT QUEUE

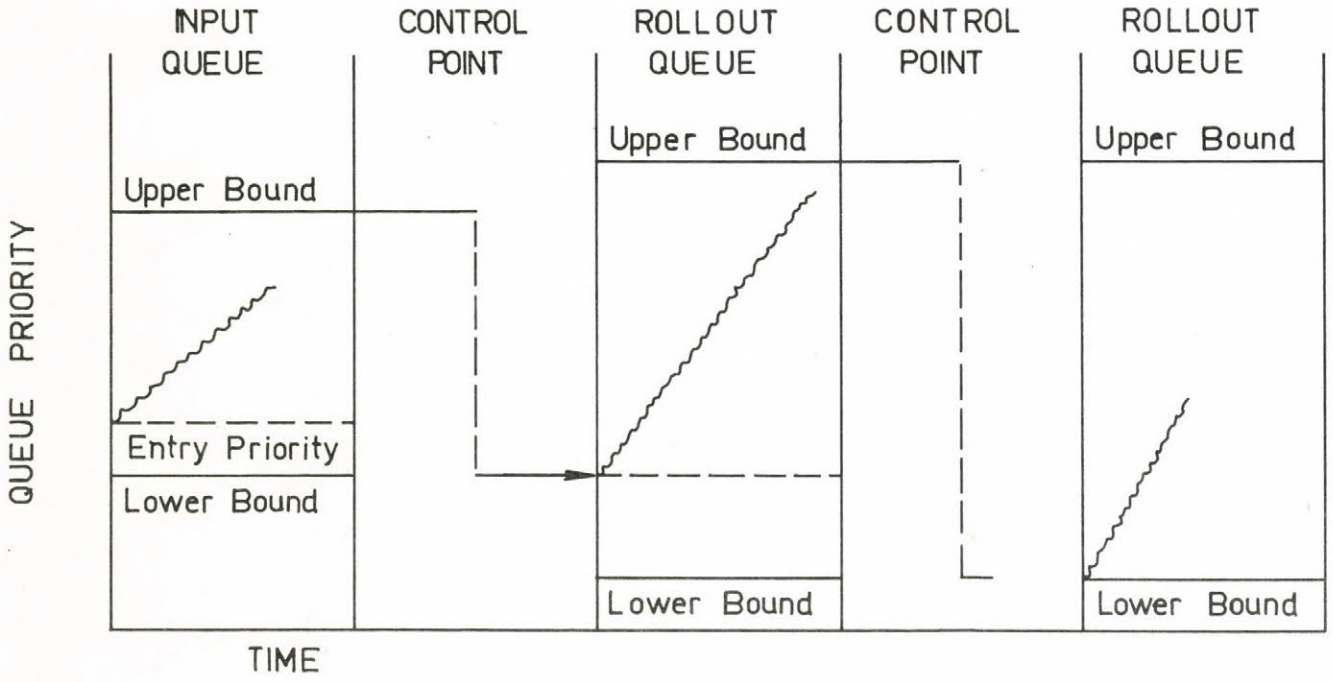


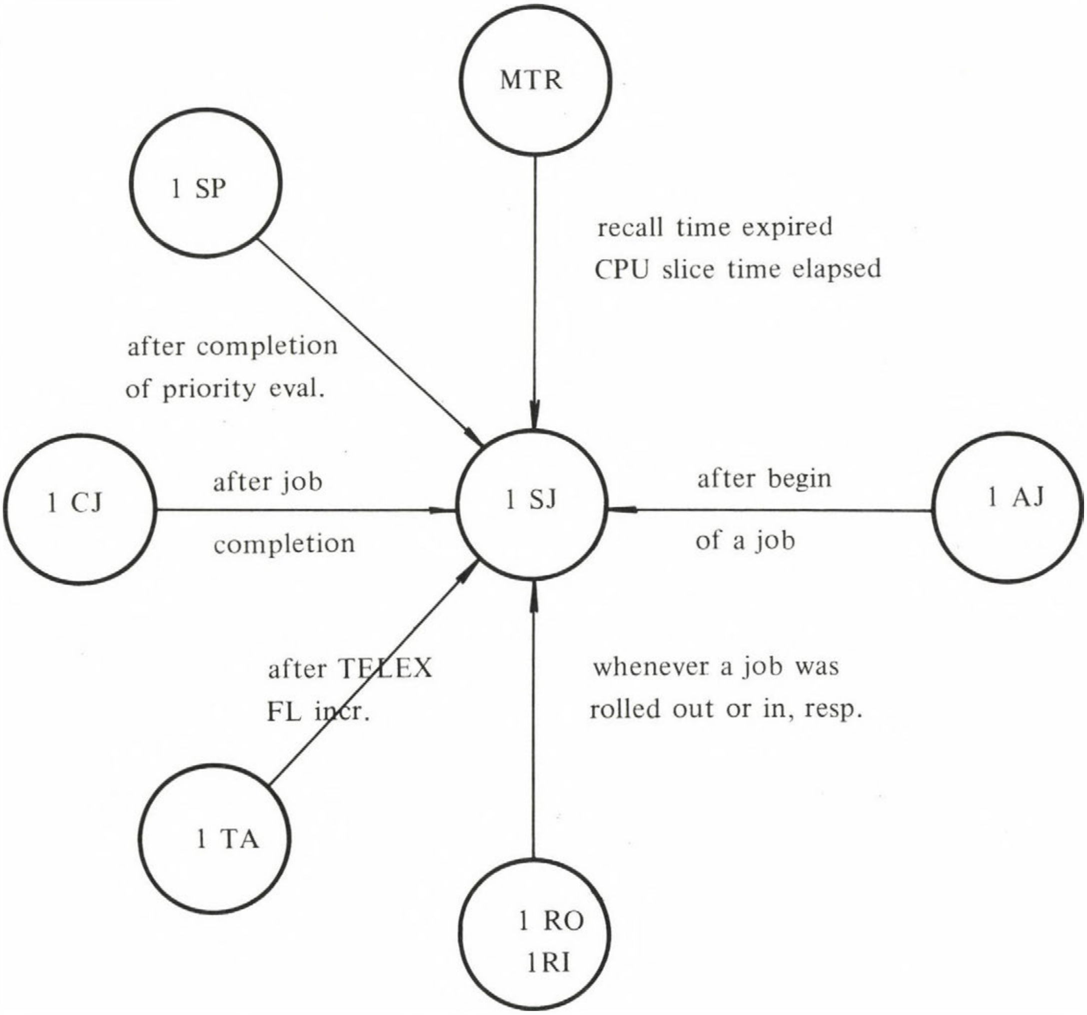
Fig.11.

|  |  | QUEUE PRIORITY |  |  |  | TIME SLICE |  |  |
|---|---|---|---|---|---|---|---|---|
| JOB ORIGIN TYPE | QUEUE TYPE | ENTRY PRIORITY | LOLER BOUND PRIORITY | UPPER BOUND PRIORITY | INCREMENT | CPU | CM | INITIAL CPU PRIORITY |
| SYSTEM | INPUT | 6600 | 700 | 3000 | 1 | 100 | 20 | 1 |
|  | ROLLOUT | 6600 | 100 | 1000 | 2 |  |  |  |
|  | OUTPUT | 400 | 100 | 7700 | 1 |  |  |  |
| BATCH | INPUT | 2400 | 2000 | 4010 | 1 | 400 | 200 | 30 |
|  | ROLLOUT | 2400 | 1010 | 4004 | 1 | 400 |  |  |
|  | OUTPUT | 200 | 100 | 7000 | 2 |  |  |  |
| EXPORT/ IMPORT | INPUT | 3400 | 2400 | 4010 | 1 | 400 | 200 | 30 |
|  | ROLLOUT | 3400 | 1400 | 4006 | 1 | 400 |  |  |
|  | OUTPUT | 200 | 100 | 7600 | 1 |  |  |  |
| TELEX | INPUT | 4000 | 3770 | 7006 | 1 | 40 | 30 | 30 |
|  | ROLLOUT | 4004 | 3740 | 7000 | 1 |  |  |  |
|  | OUTPUT | 200 | 100 | 7000 | 1 |  |  |  |
| MULTI-TERMINAL | INPUT | 6774 | 6700 | 7400 | 1 | 400 | 60 | 31 |
|  | ROLLOUT | 6774 | 4000 | 7400 | 1 |  |  |  |
|  | OUTPUT | 6000 | 100 | 7700 | 1 |  |  |  |

DELAY PARAMETERS

| JS | CR | AR | JA | CS |
|---|---|---|---|---|
| 1 | 10 | 200 | 10 | 10 |

Fig.12.



Philosophy: 1SJ should be called when the system status has changed

Fig.13.

SCHEDULER FLOW

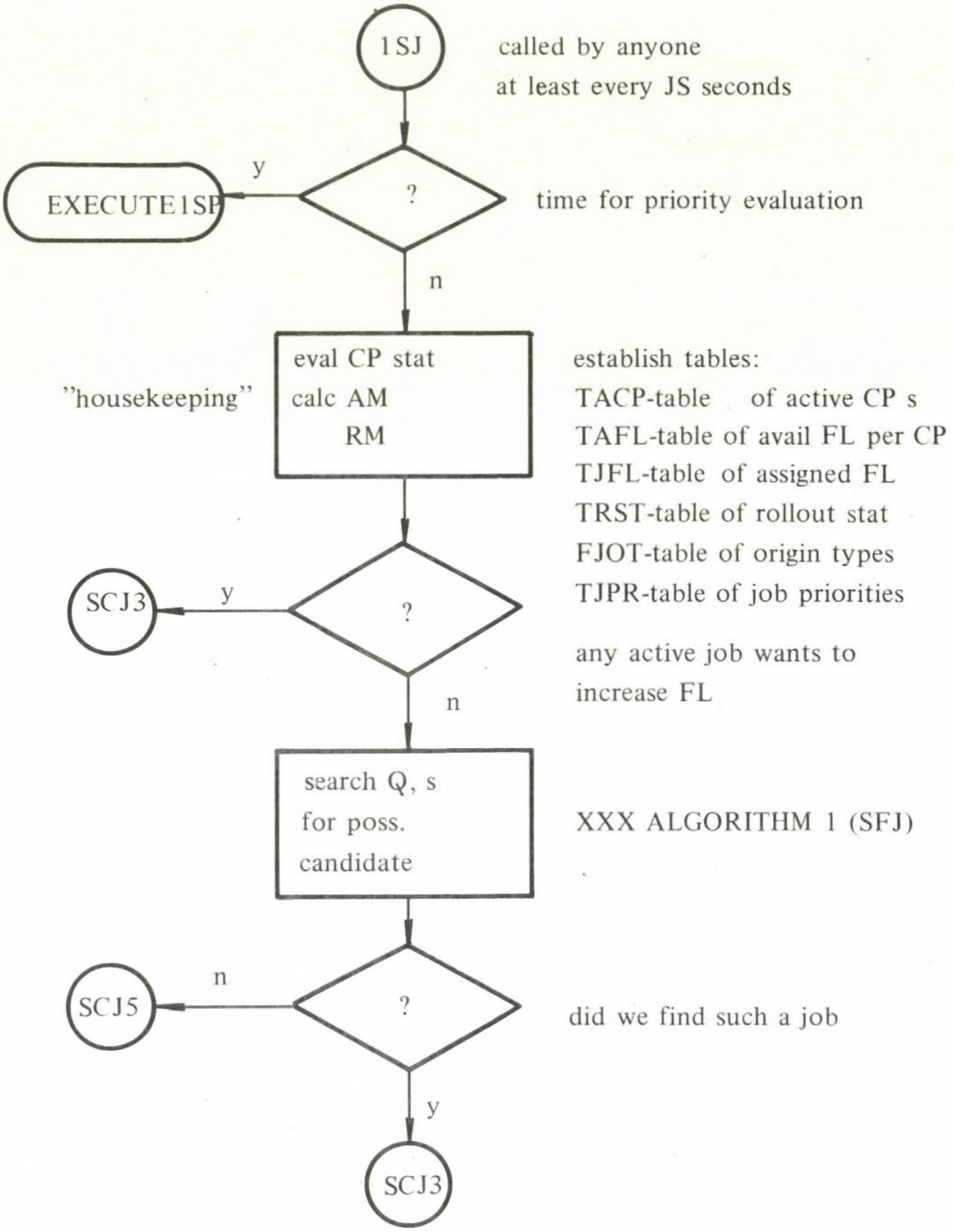

1SJ
called by anyone
at least every JS seconds
y
EXECUTE1SP
?
time for priority evaluation
n
"housekeeping"
eval CP stat
calc AM
RM
establish tables:
TACP-table of active CP s
TAFL-table of avail FL per CP
TJFL-table of assigned FL
TRST-table of rollout stat
FJOT-table of origin types
TJPR-table of job priorities
SCJ3
y
?
any active job wants to
increase FL
n
search Q, s
for poss.
candidate
XXX ALGORITHM 1 (SFJ)
SCJ5
n
?
did we find such a job
y
SCJ3

Fig.14.

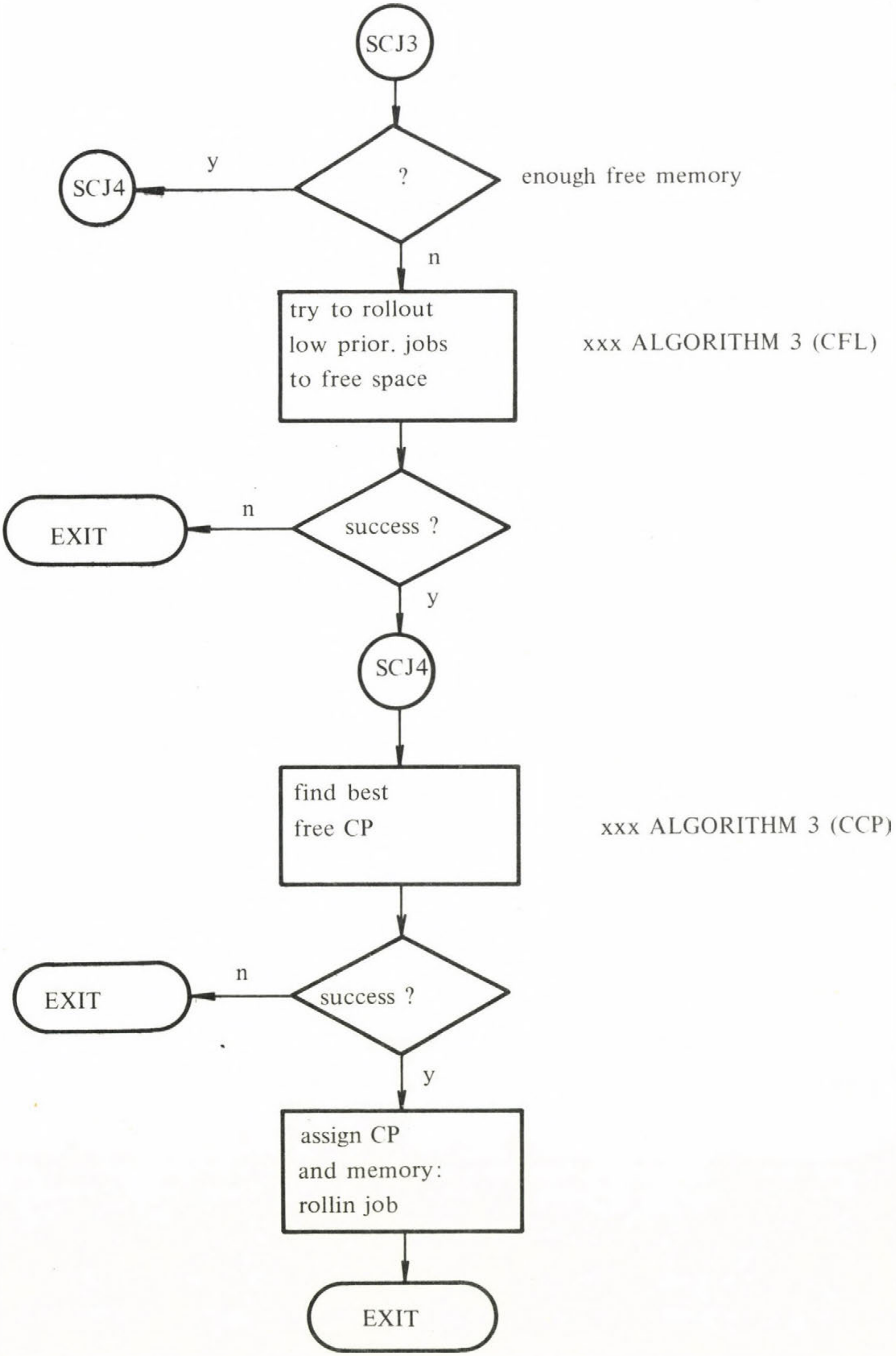

SCJ3
?
y
SCJ4
enough free memory
n
try to rollout low prior. jobs to free space
xxx ALGORITHM 3 (CFL)
success ?
n
EXIT
y
SCJ4
find best free CP
xxx ALGORITHM 3 (CCP)
success ?
n
EXIT
y
assign CP and memory: rollin job
EXIT

Fig.15.

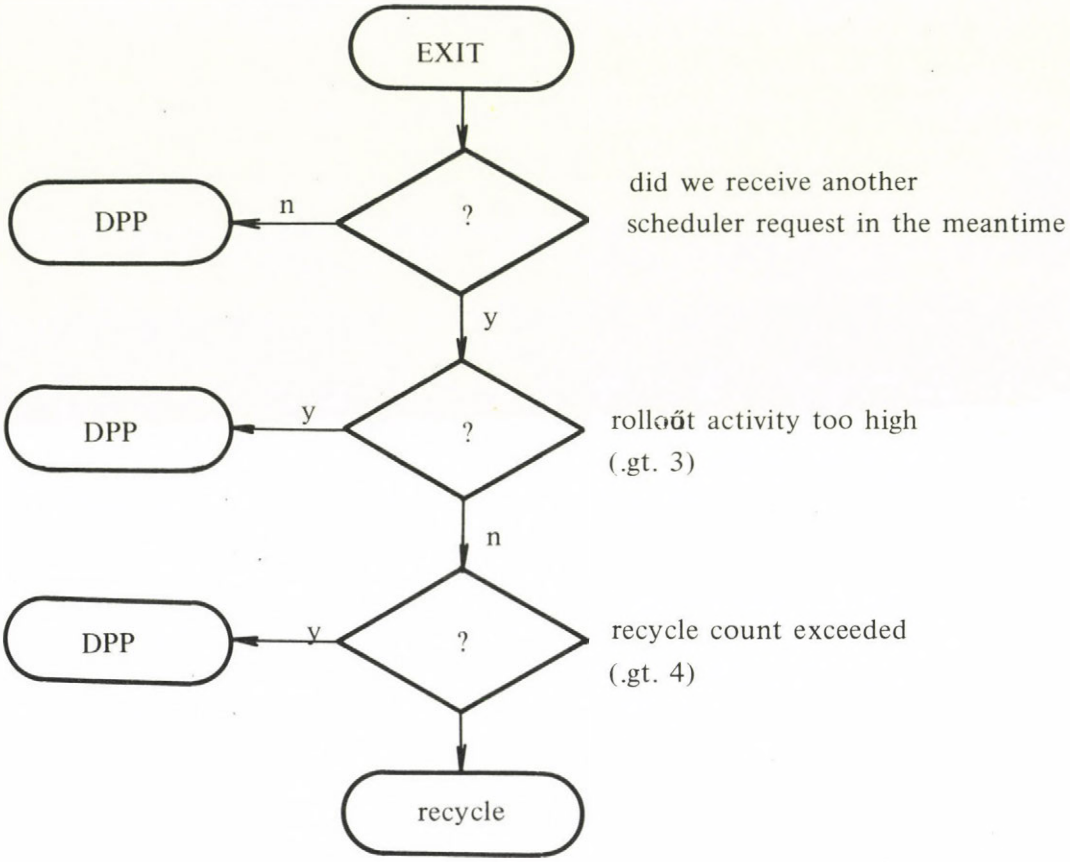
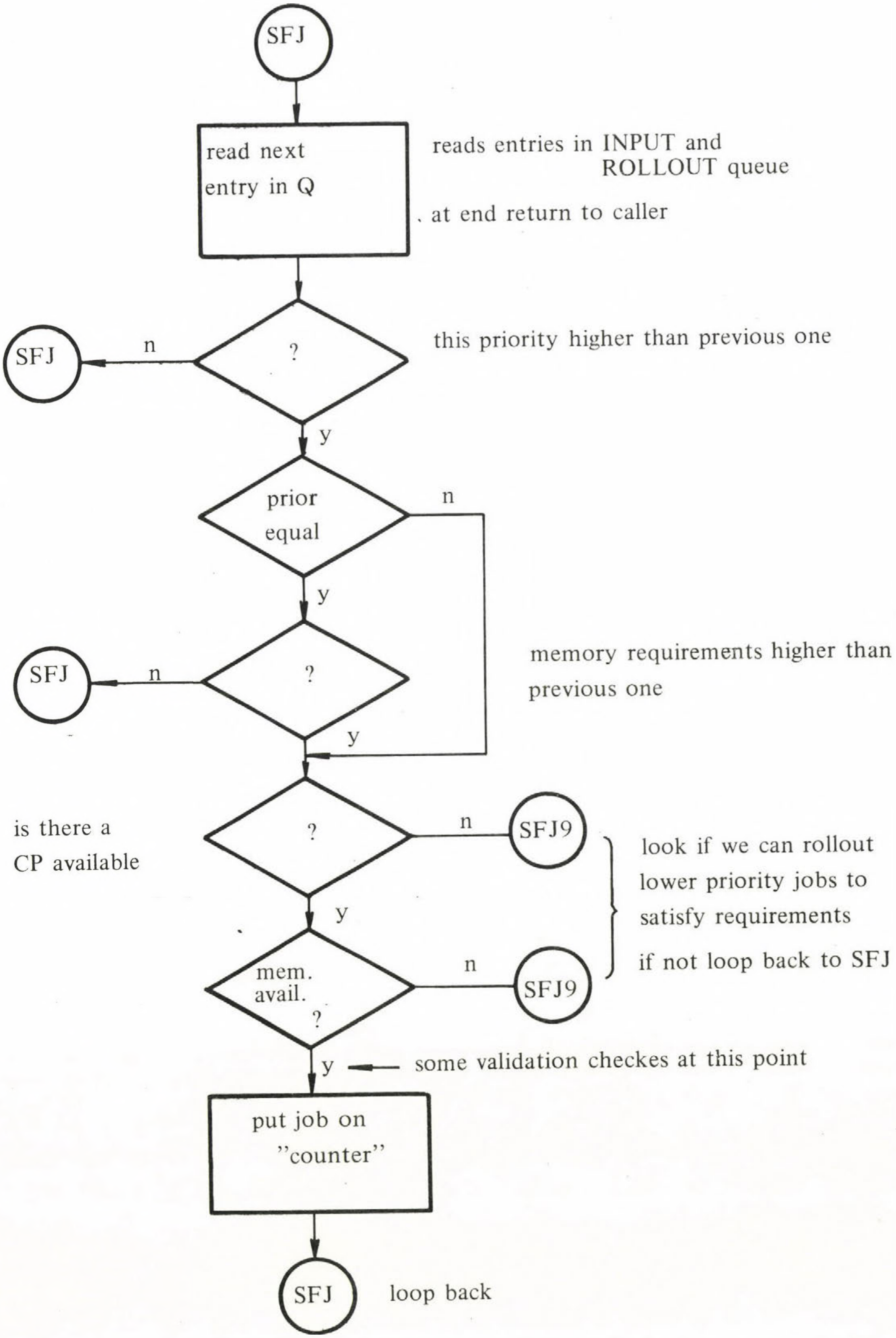

EXIT
DPP
n
?
did we receive another scheduler request in the meantime
y
DPP
y
?
rolloűt activity too high (.gt. 3)
n
DPP
y
?
recycle count exceeded (.gt. 4)
recycle

Fig.16.

ALGORITHM 1

SFJ – search for job



reads entries in INPUT and ROLLOUT queue

. at end return to caller

this priority higher than previous one

memory requirements higher than previous one

is there a CP available

look if we can rollout lower priority jobs to satisfy requirements

if not loop back to SFJ

some validation checkes at this point

loop back

Fig.17.

## ALGORITHM 2

CFL – commit field lenght

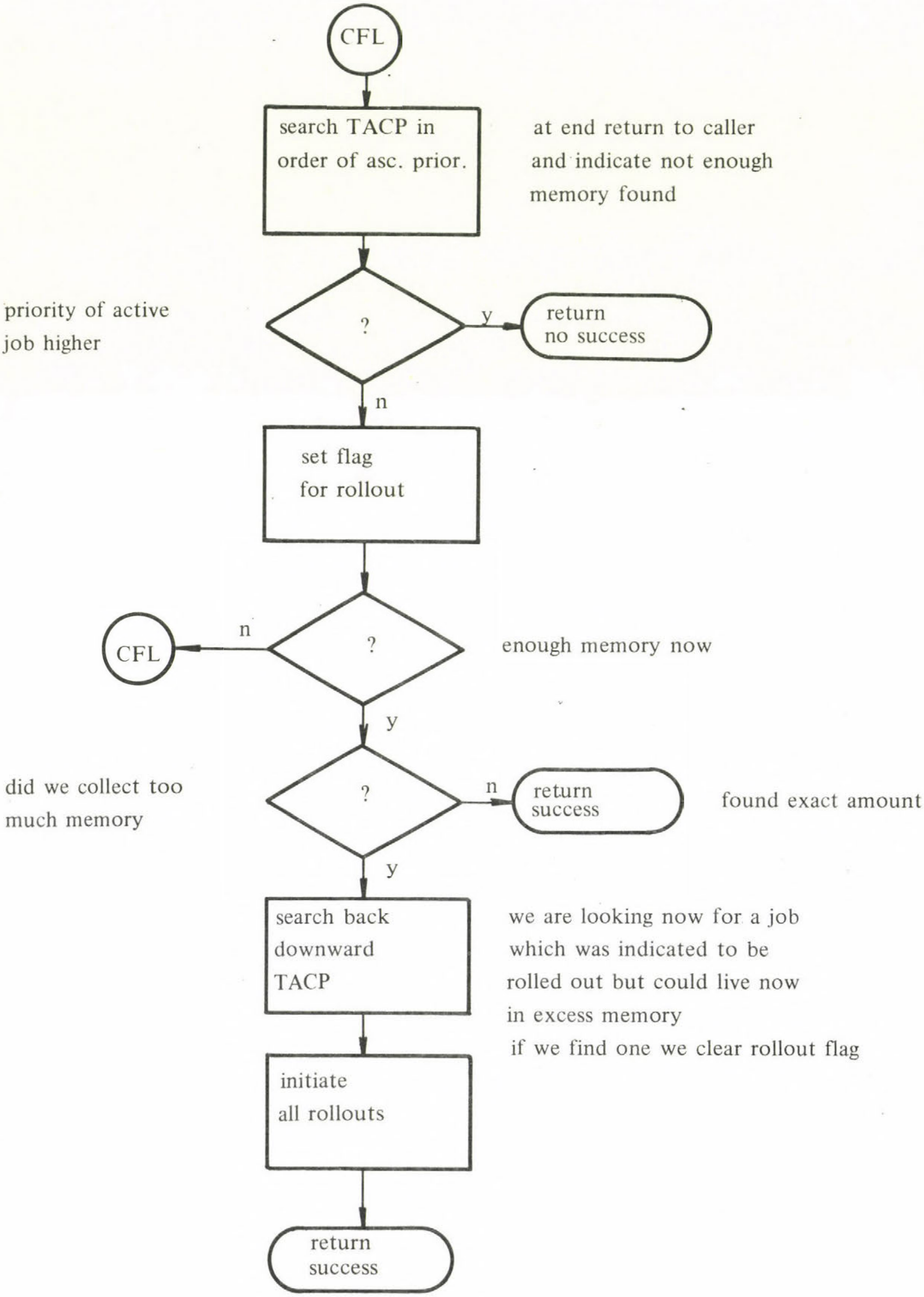

CFL
search TACP in order of asc. prior.
at end return to caller and indicate not enough memory found
priority of active job higher
?
y
return no success
n
set flag for rollout
CFL
n
?
enough memory now
y
did we collect too much memory
?
n
return success
found exact amount
y
search back downward TACP
we are looking now for a job which was indicated to be rolled out but could live now in excess memory if we find one we clear rollout flag
initiate all rollouts
return success

Fig.18.

ALGORITHM 3

CCP – commit control point

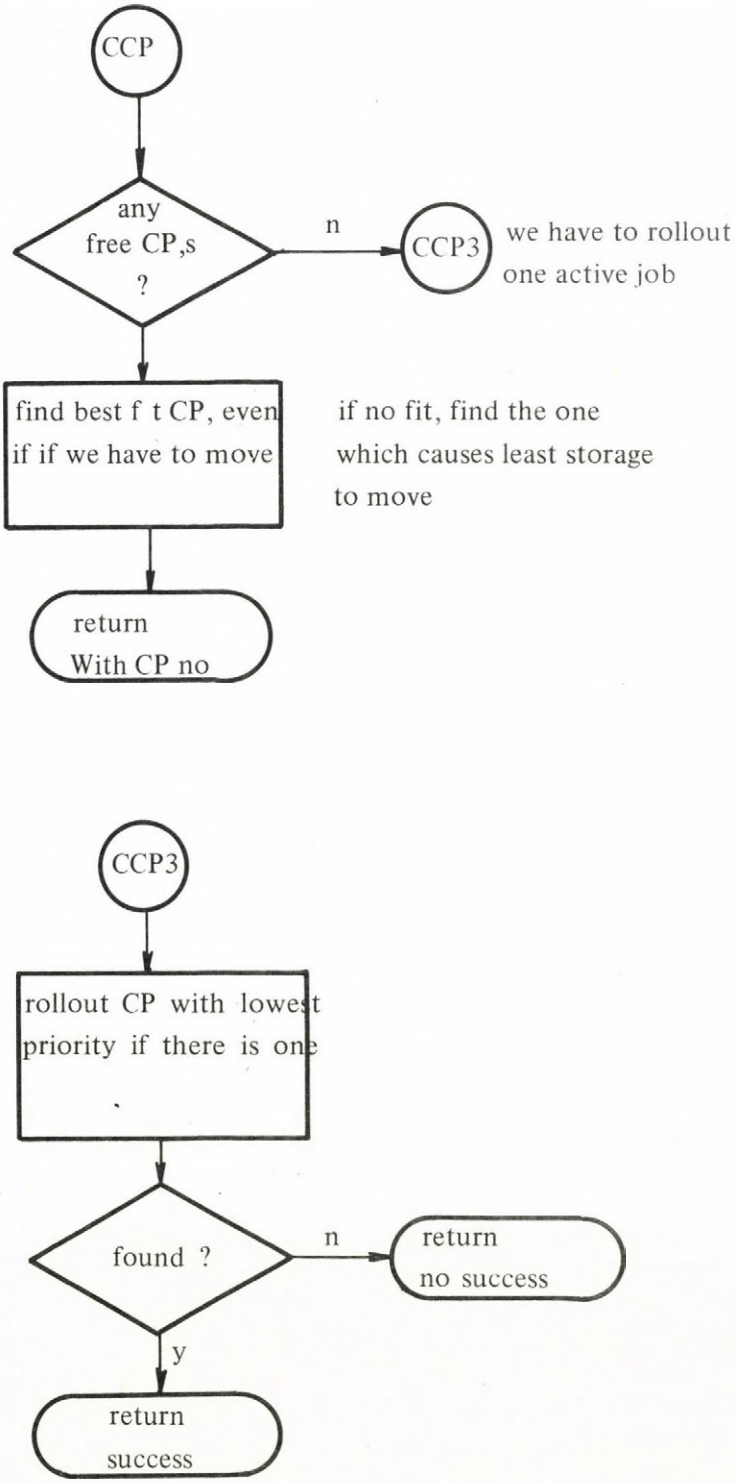


Fig.19.