NOS/VE ANALYSIS

Student Handout

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released. This course reflects the state of the OS as it is expected to be
(3-1-81)	when released in 1983.
Publication No.	

REVISION LETTERS I, O, Q AND X ARE NOT USED

ASF Part No: RW4100

Printed in the United States of America
1 2 3 4 5 6 7 8 9

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Southgate Office Plaza
5001 West 80th Street
Bloomington, MN 55437

or use Comment Sheet in the back of this manual.

TABLE OF CONTENTS

INTRODUCTION

TITLE

NOS/VE Analysis

DESCRIPTION

NOS/VE Analysis is a "detailed overview" of the C180 virtual state operating system. The course will cover system structure thoroughly. Other topics will be covered in somewhat less detail, for example, the executive, SCL interpreter, task manager and logical I/O. This course will also cover the tools, resources and techniques needed to extend, maintain and support NOS/VE.

Note that it is the purpose of this course to provide a solid base for further study and work on NOS/VE, not to make any student an expert in any particular area of the system.

The course will be 5 days long. There will be projects and exercises but no "hands-on" experiments.

PREREQUISITES

The student should be comfortable with CYBIL, the Program Interface and the Command Interface. All three are offered as courses by the Seminar Division as part of the NOS/VE - C180 curriculum.

COURSE CHART

HOUR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5
1	1 INTRO	Review	Review	Review	Review
2	2 Concepts	5 Resources	7 External Communication	9 Program Execution	12 Logical IO
3	3 Job Flow	Exercise			
4	4 File Flow	6 Internal Communication	8 Job Control	10 SCL	Exercise
5				11 Permanent Files	13 Physical IO
6	Exercise		Exercise	Exercise	

COURSE OUTLINE

PART I — CONCEPTS

1. Objectives
    a. Course Objectives
    b. Course Structure
    c. NOS/VE Objectives
2. NOS/VE Structures
    a. Packaging
    b. Table Segments
    c. Components
    d. Memory Layout
3. Job Flow
    a. Initiation
    b. Command Processing
    c. Termination
4. File Flow
    a. Open
    b. Transfer
    c. Close

PART II — COMMUNICATION

5. Resources
    a. Documentation
    b. System Initialization
    c. Load Map
6. Internal Communication
    a. Call/Return
    b. Interrupts
    c. Monitor
    d. Traps
7. External Communication
    a. Dual State
    b. Logs and Statistics
    c. Message Generator
    d. Keypoint

PART III — JOB/PROGRAM MANAGEMENT

8. Job Control
    a. Queued File Management
    b. Job Initiation
    c. Job Termination
9. Program Execution
    a. Task Management
    b. Loader
    c. Condition Handling
10. SCL Interpreter
    a. Control
    b. Command Processors

MATERIALS

PRIMARY REFERENCES

- STUDENT HANDOUT
- NOS/VE PROCEDURES AND CONVENTIONS
- DESIGN SPECIFICATION
  PART II - INTERNAL INTERFACE
  PART III - PACKAGING

SECONDARY REFERENCES

- DESIGN SPECIFICATION
  PART 1 - STRUCTURE CHARTS
- COMMAND INTERFACE (ARH3609)
- PROGRAM INTERFACE (ARH3610)
- GENERAL INTERNAL DESIGN
- INTEGRATION NOTEBOOK

LESSON 1
OBJECTIVES

LESSON PREVIEW

COURSE OBJECTIVES
COURSE STRUCTURES
NOS/VE OBJECTIVES
NOS/VE RELEASE SCHEDULE

REFERENCES

ARCHITECTURAL OBJECTIVES/REQUIREMENTS (SO/R) - ARH1688.
SECTIONS 1.3 and 3.3

GID-PART 1, CHAPTER 2

OBJECTIVES

After completing this lesson the student should be able to --

- o STATE THE GENERAL OBJECTIVES OF THE COURSE
- o UNDERSTAND ENOUGH ABOUT THE STRUCTURE OF THE COURSE TO NOT BE SURPRISED BY ANY NEW TOPIC
- o OUTLINE THE MAIN OBJECTIVES OF NOS/VE AND THE STRATEGIES TO MEET THEM
- o OUTLINE THE RELEASE SCHEDULE FOR NOS/VE

EXERCISES

NONE

COURSE OBJECTIVES

EXTEND	Careful Survey
	Selected Detail
FIX	Available Resources
	Use of Tools
SUPPORT	Methods & Procedures

COURSE STRUCTURE

PART I	CONCEPTS	
	1.	Objectives
	2.	NOS/VE Structure
	3.	Job Flow
	4.	File Flow
PART II	COMMUNICATION	
	1.	Materials
	2.	Internal Communication
	3.	Memory Management
	4.	External Communication
PART III	JOB/PROGRAM MANAGEMENT	
	1.	Permanent Files
	2.	Logical I/O
	3.	Device Management
	4.	Physical I/O
PART IV	FILES	
	1.	Permanent File
	2.	Logical I/O
	3.	Device Management
	4.	Physical I/O

NOS/VE OBJECTIVES

OBJECTIVES	STRATEGIES
RAM	HARDWARE
CONFIGURABILITY	SASD
EXPANDABILITY	CYBIL
USABILITY	STANDARDS
CONSISTENCY	COMMAND INTERFACE
EFFICIENCY	PROGRAM INTERFACT
SECURITY	DUAL STATE
MIGRATION EASE	CP OPERATING SYSTEM
	CODE ISOLATION
	SYSTEM USING ITSELF
	ON-LINE DEVELOPMENT

PHASED RELEASES

Requires NOS

R1 Basic Operating System
Disk and Tape Drivers
FORTRAN and COBOL
Dual-State
Conversion Aids

END 82 availability

R2 Stand-Alone System
Unit Record Drivers
Interactive Facility
Products and Utilities

MID 84 announcement

R3 Competitive System
Networks
Applications
Etc.

END 85

LESSON 2
NOS/VE STRUCTURE

LESSON PREVIEW

MONITOR VS JOB STATE

FUNCTIONAL DIVISION OF NOS/VE

MAP OS TO HARDWARE

REFERENCES

MIDGS
GID-PART 1, CHAPTERS 3

OBJECTIVES

After completing this lesson the student should be able to--

- o DISTINGUISH BETWEEN A MONITOR STATE AND A JOB STATE XP
- o EXPLAIN HOW SYSTEM PACKAGING TAKES ADVANTAGE OF THE RING STRUCTURE TO EFFECT COMPONENT ISOLATION
- o LIST THE MAIN FUNCTIONAL AREAS OF THE SOFTWARE, INDICATE WHERE THE CODE AND TABLES FOR EACH RESIDES

EXERCISES

NONE

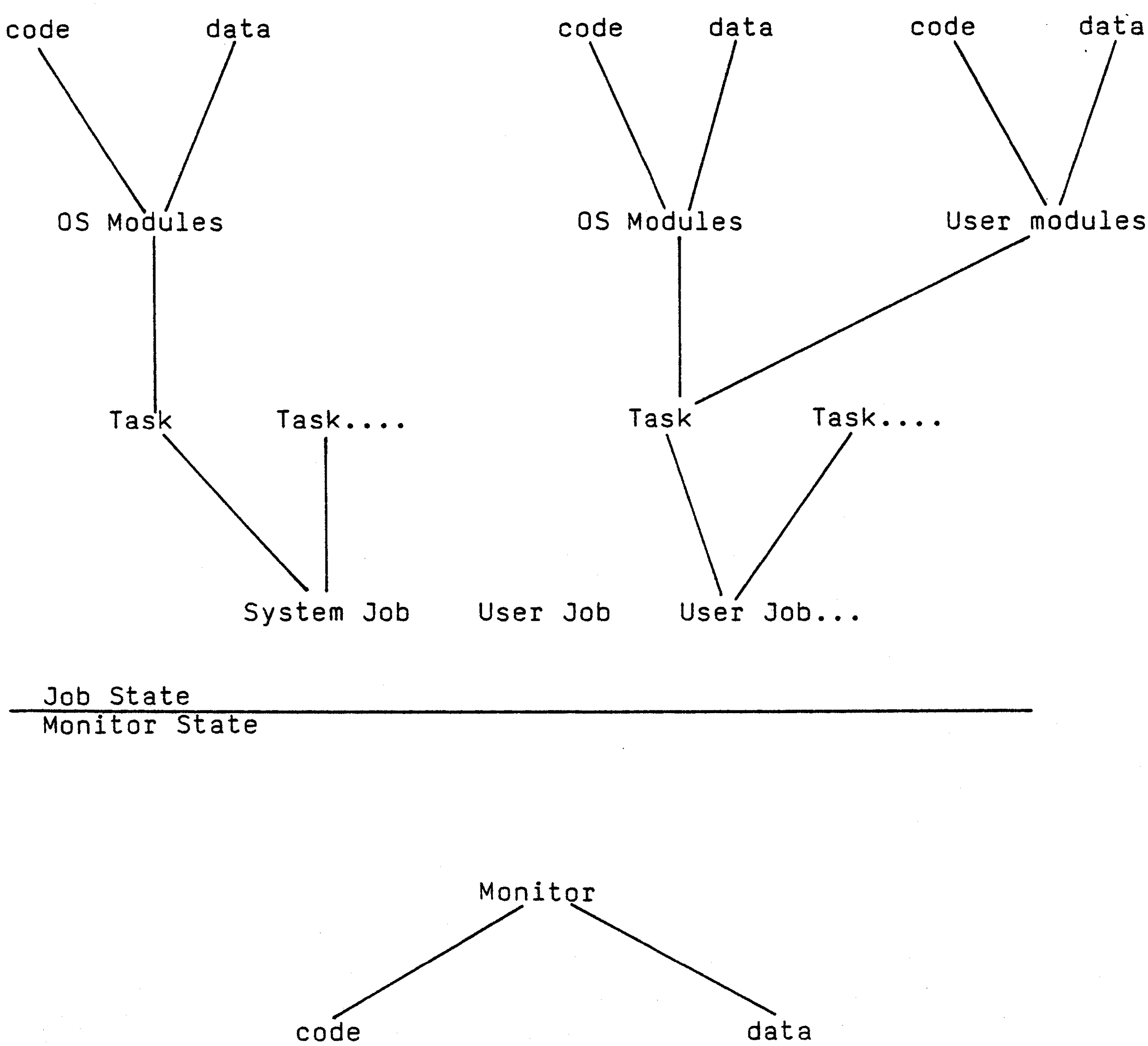
HARDWARE CONSIDERATIONS

- o JOB STATE vs MONITOR STATE
  - Instruction Privilege
  - Interrupts
- o VIRTUAL ADDRESS SPACE
  - Large - 4096 segments 2^{31} bytes/segment
  - Segmented
  - Protected
- o COMMUNICATION
  - Call/Return
  - Exchange - state (MONITOR, JOB)
  - Traps

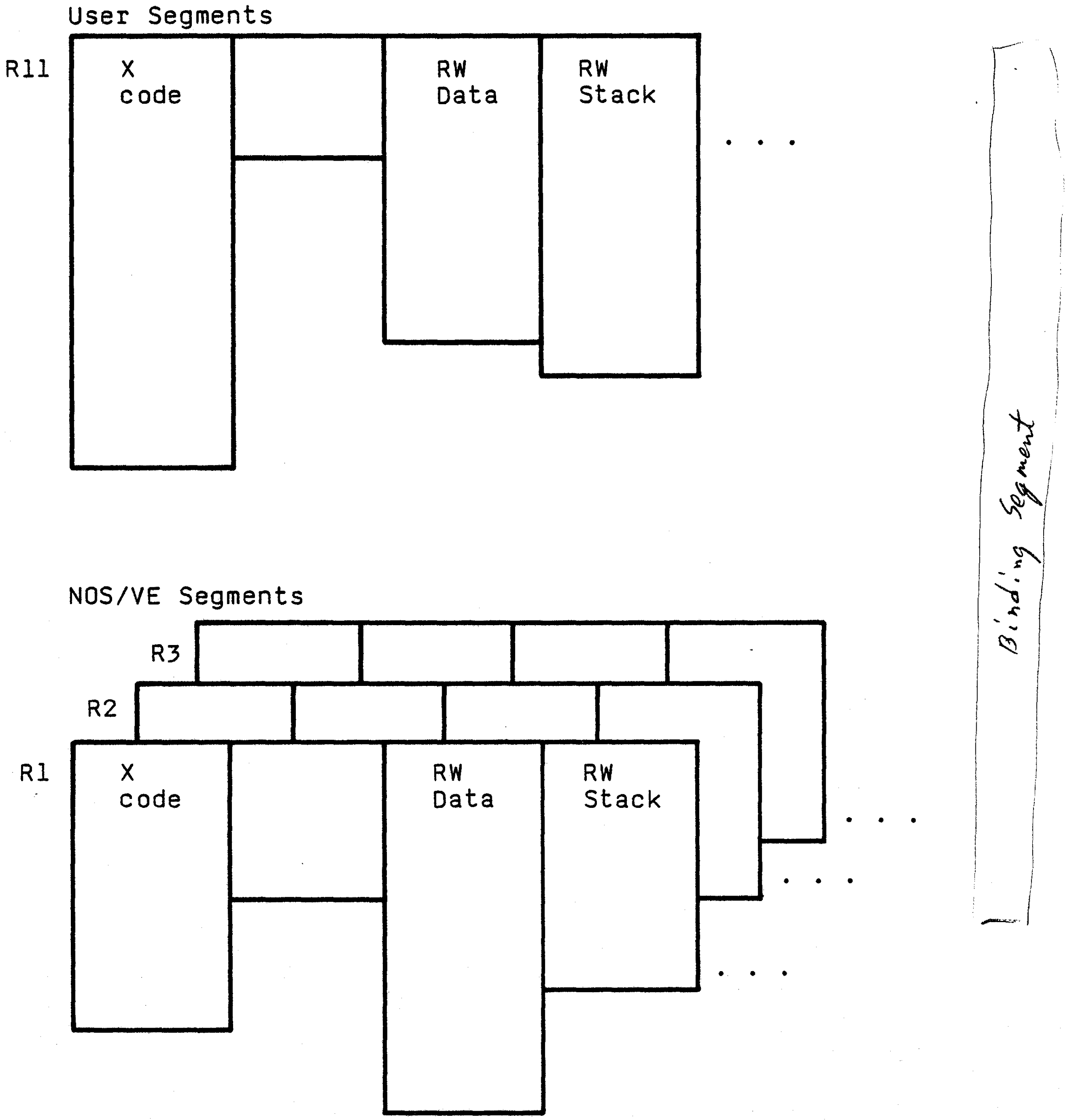
1. Control

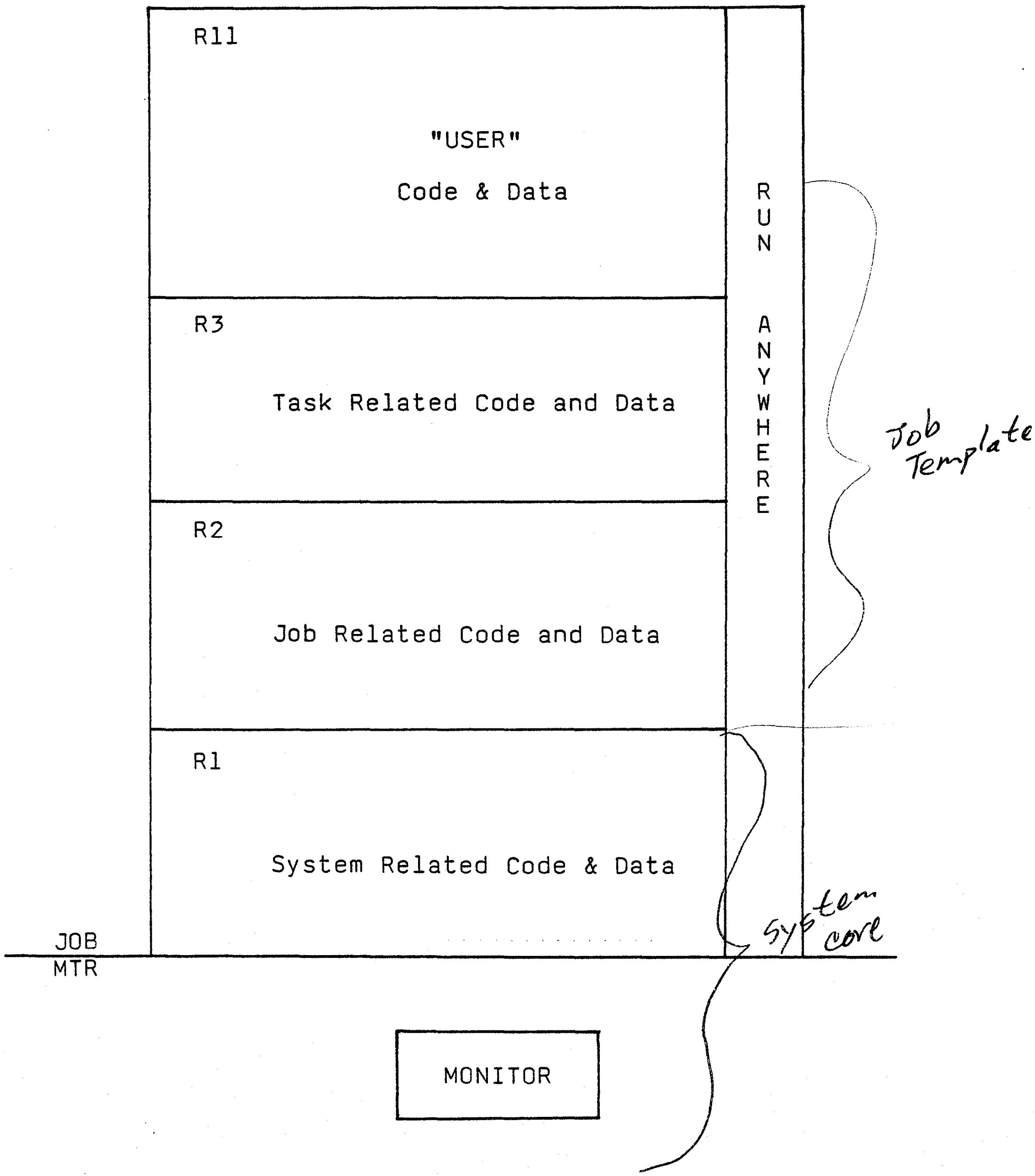
2. Data
   parameter
   shared data
   signal

OS HIERARCHY

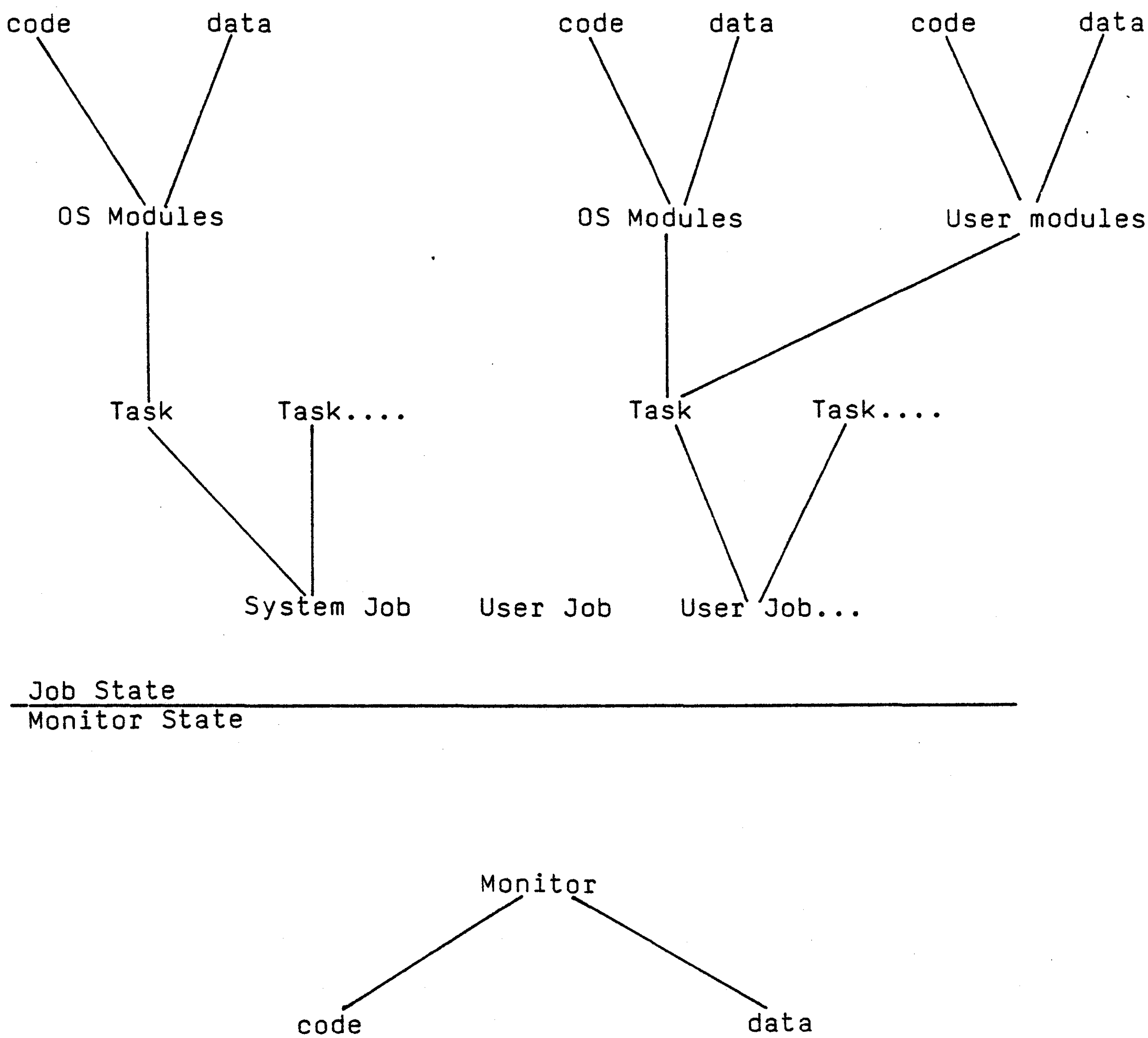


USER TASK ADDRESS SPACE




R11
"USER"
Code & Data
R3
Task Related Code and Data
R2
Job Related Code and Data
R1
System Related Code & Data
R U N A N Y W H E R E
Job Template
System core
JOB
MTR
MONITOR

SHARED DATA


code
data
code
data
code
data
OS Modules
OS Modules
User modules
Task
Task....
Task
Task....
System Job
User Job
User Job...
Job State
Monitor State
Monitor
code
data

GLOBAL TABLES

TABLE RESIDENCE

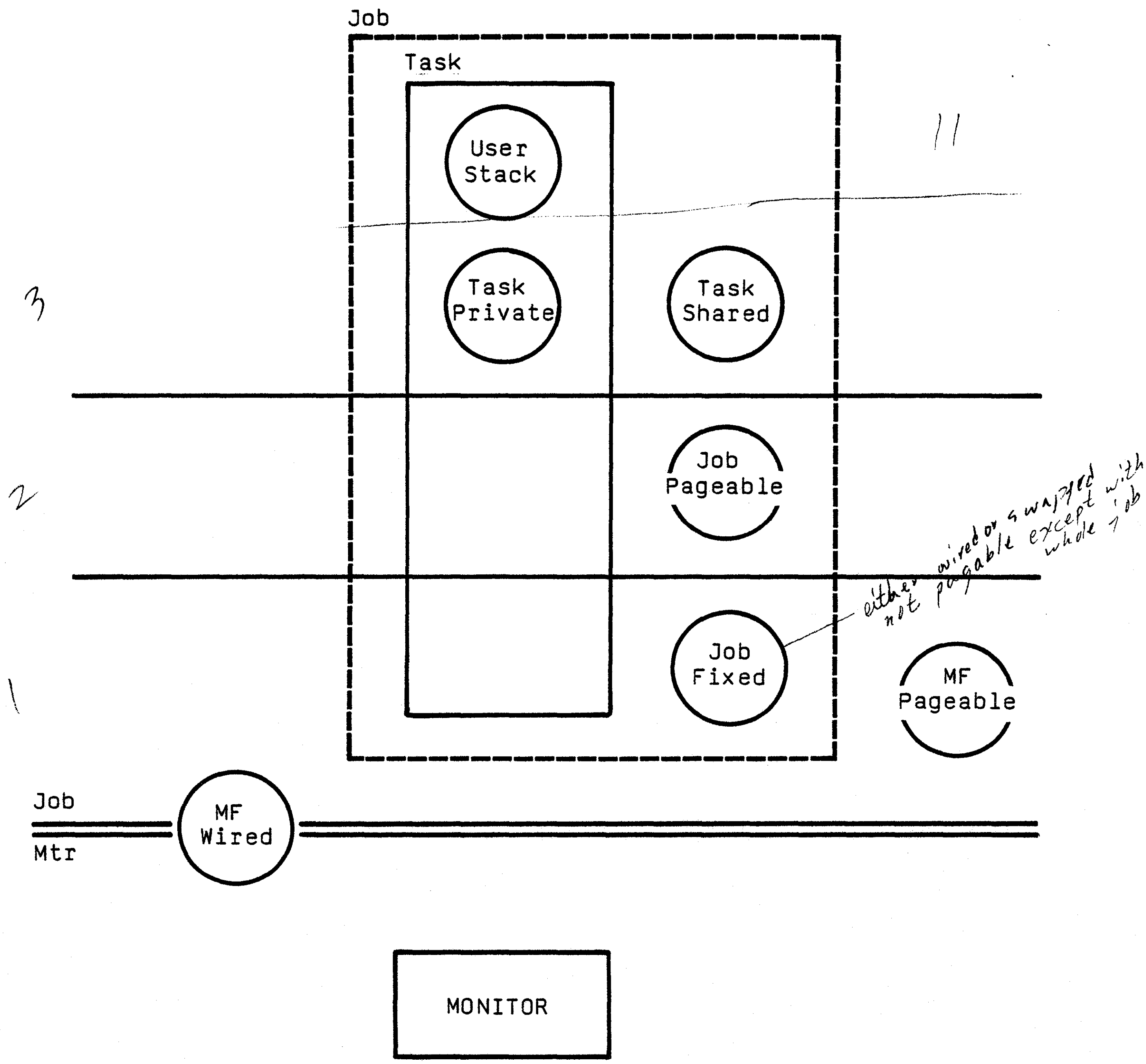
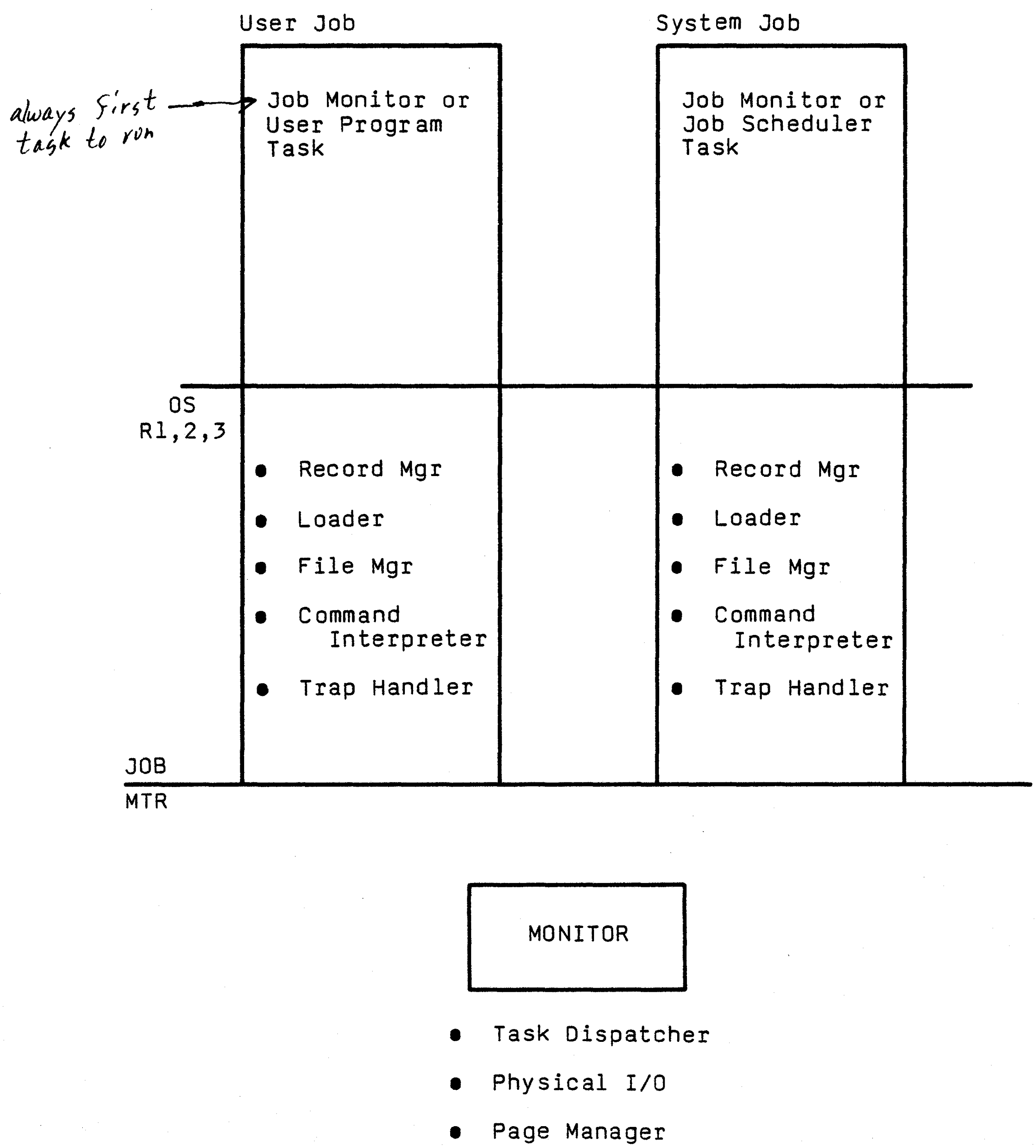
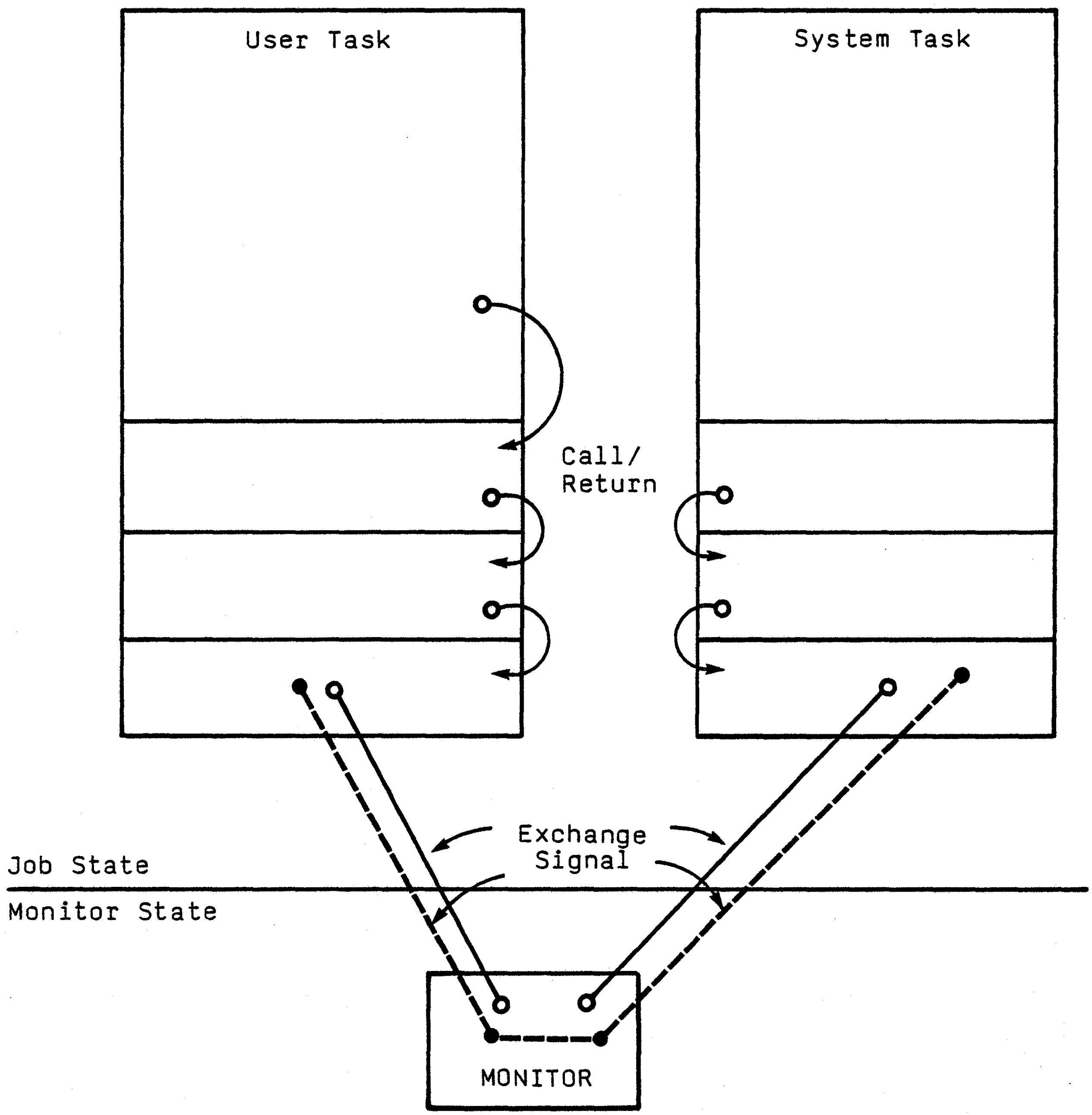


TABLE SEGMENT ATTRIBUTES

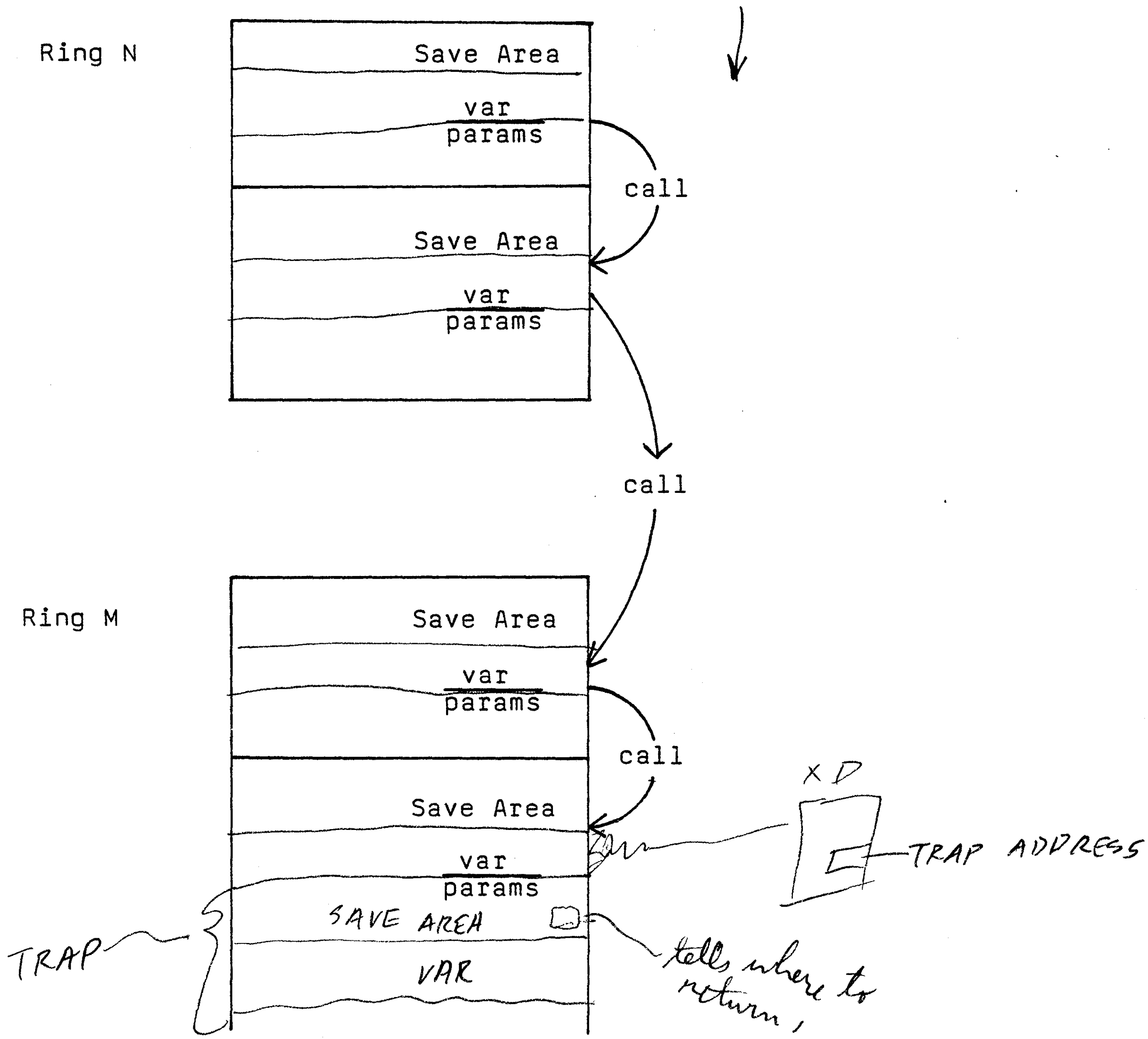
Segment	Name	Rings	
1	OSV$MAINFRAME-WIRED	(1,3)	Always in real memory. One per system. Monitor read and write.
2	OSV$MAINFRAME-PAGEABLE	(1,3)	Pageable. One per system.
3	OSV$JOB-FIXED	(1,3)	Wired when the job is active; swapped when the job is swapped. One per job. Monitor read and write.
4	OSV$JOB-PAGEABLE	(2,3)	One per job.
5	OSV$TASK-PRIVATE	(3,3)	one/task
6	OSV$TASK-SHARED	(3,13)	One per ~~task~~ job. Pageable. Shared with other tasks of the same job.
7	OSV$TASK-PRIVATE-R11	(11,11)	One per task. Pageable. Not shared.

FUNCTIONS


User Job
System Job
always first task to run
Job Monitor or User Program Task
Job Monitor or Job Scheduler Task
OS
R1,2,3
• Record Mgr
• Loader
• File Mgr
• Command Interpreter
• Trap Handler
• Record Mgr
• Loader
• File Mgr
• Command Interpreter
• Trap Handler
JOB
MTR
MONITOR
• Task Dispatcher
• Physical I/O
• Page Manager

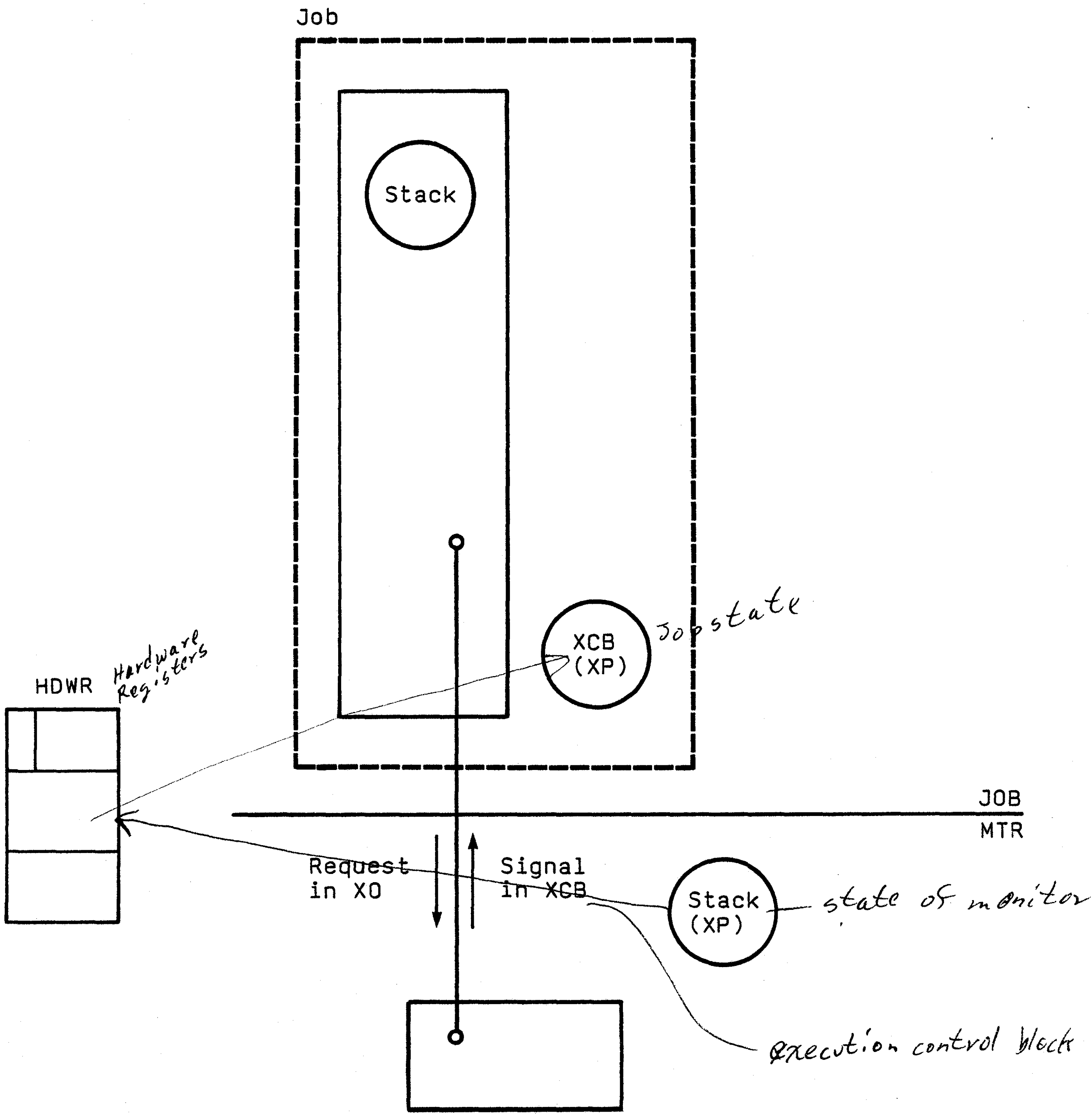

User Task
System Task
Call/
Return
Exchange
Signal
Job State
Monitor State
MONITOR

STACK


Ring N
Save Area
var
params
call
Save Area
var
params
call
Ring M
Save Area
var
params
call
Save Area
var
params
SAVE AREA
VAR
TRAP
XD
TRAP ADDRESS
tells where to return,

mln

EXCHANGE


Job
Stack
XCB
(XP)
Job state
HDWR
Hardware Registers
JOB
MTR
Request
in X0
Signal
in XCB
Stack
(XP)
state of monitor
execution control block

* EXCHANGE JUMP
The exchange jump instruction is used to change state.

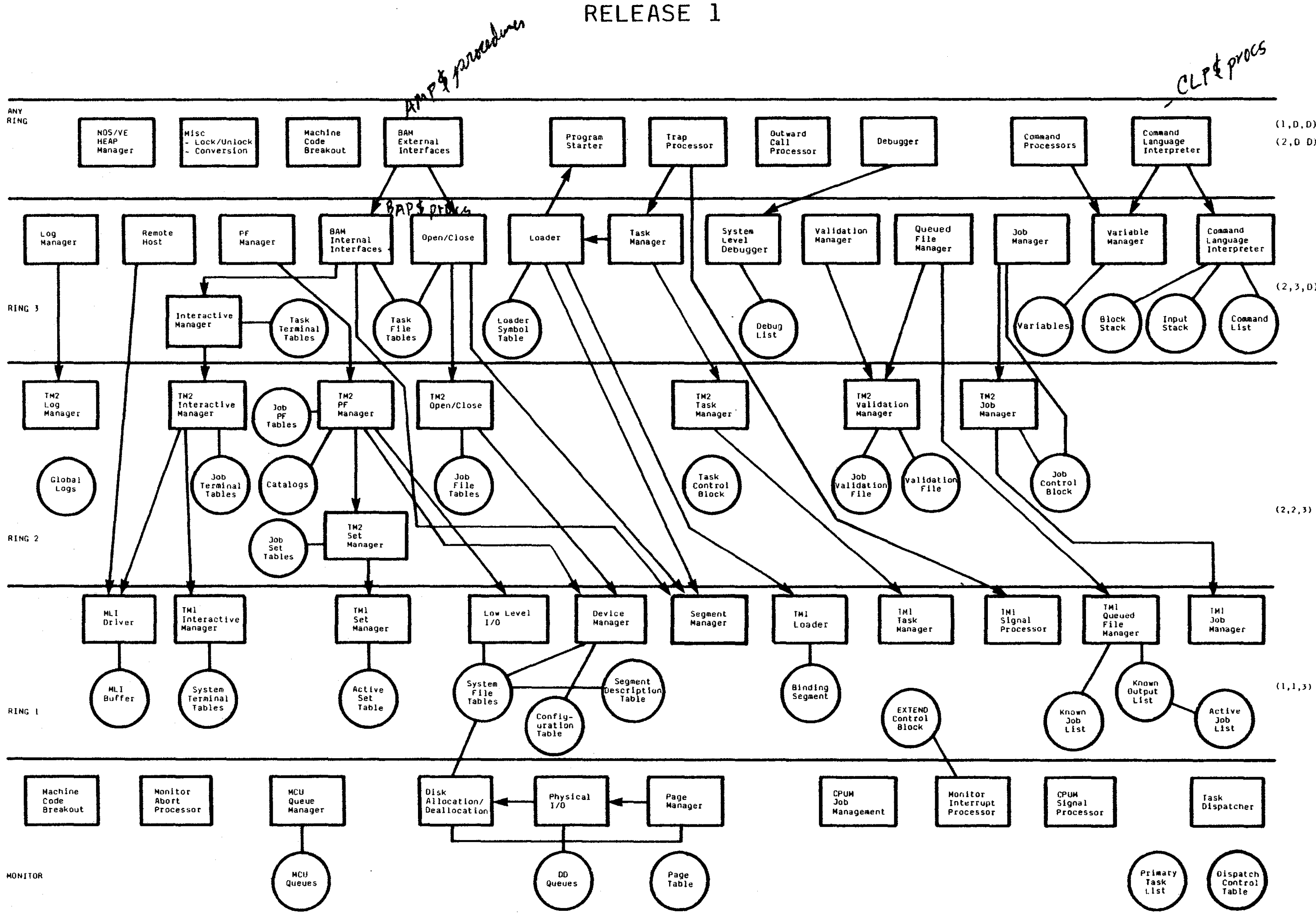
Job state programs will exchange to monitor at the PVA in the monitor state XP P register. The system call bit in the MCR is set and the request will be in X0.

Monitor will find the XP of the appropriate task in the XCB entry for that task and exchange to the XP address. A system signal, a system flag or a MCR condition might indicate a special reason for the entry. In that case, monitor will set the free flag in the job state XP and execute the exchange jump. A trap will occur immediately in job state.

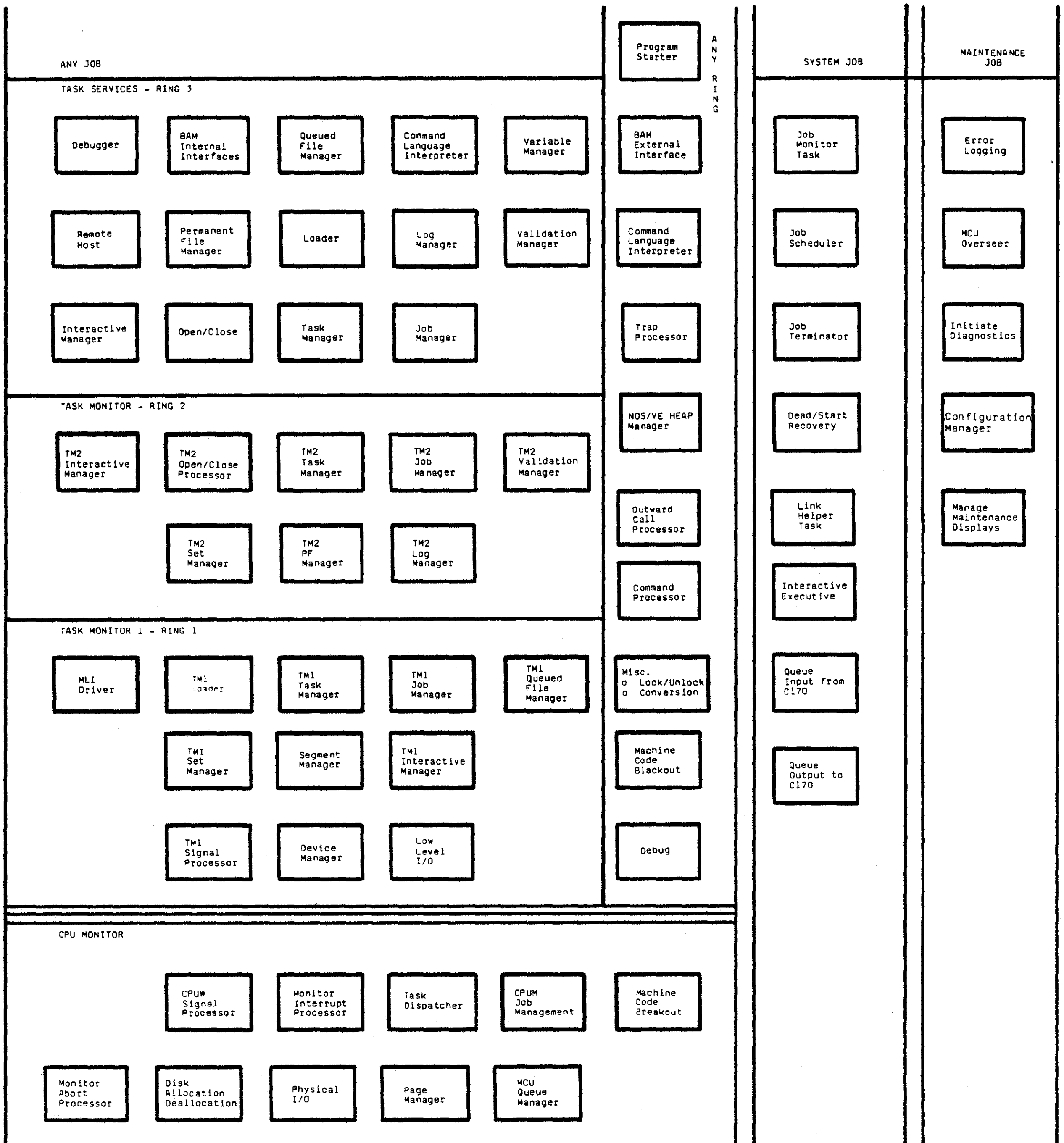
* EXCHANGE INTERRUPT
Exchange interrupts occur in job state when a selected monitor condition occurs. Monitor runs at the PVA in the monitor state XP.

* TRAP INTERRUPT
If traps are enabled, a trap interrupt will occur when a selected user condition occurs in the job state or a selected monitor condition occurs in monitor state. In either case there is no exchange. A stack frame is built and the trap handler is executed.

NOS/VE TARGET
RELEASE 1



NOS/VE TARGET (Cont.)



Control Data Corporation Software Document	Document Class / Mach Type	Project No.	Document Abstract NOS VE TARGET RELEASE 1
Sample Code ☐	Document Title	Project Mgr.	
Flowchart ☐	Page of	Project Name	
Decision Table ☐	Number / Issue Date	Task No.	
Other ☐	Drawn By JHW Date 7/2/81	Task Name	

LESSON 3
JOB FLOW

LESSON PREVIEW

JOB ENTRY
JOB INITIATION
COMMAND PROCESSING
PROGRAM EXECUTION
JOB TERMINATION

REFERENCES

GID-PART 1, CHAPTER 3.1, 3.2, 3.3, 3.6, 3.10, 3.11

OBJECTIVES

After completing this lesson the student should be able to--

- DISTINGUISH BETWEEN JOB AND TASK
- TRACE A JOB FROM INITIATION TO TERMINATION
- EXPLAIN HOW THE SCL INTERPRETER FINDS THE PROCESSOR FOR A COMMAND
- DISTINGUISH BETWEEN JOB SCHEDULING AND TASK DISPATCHING
- LIST THE MAIN TABLES THAT CONTROL JOBS AND TASKS

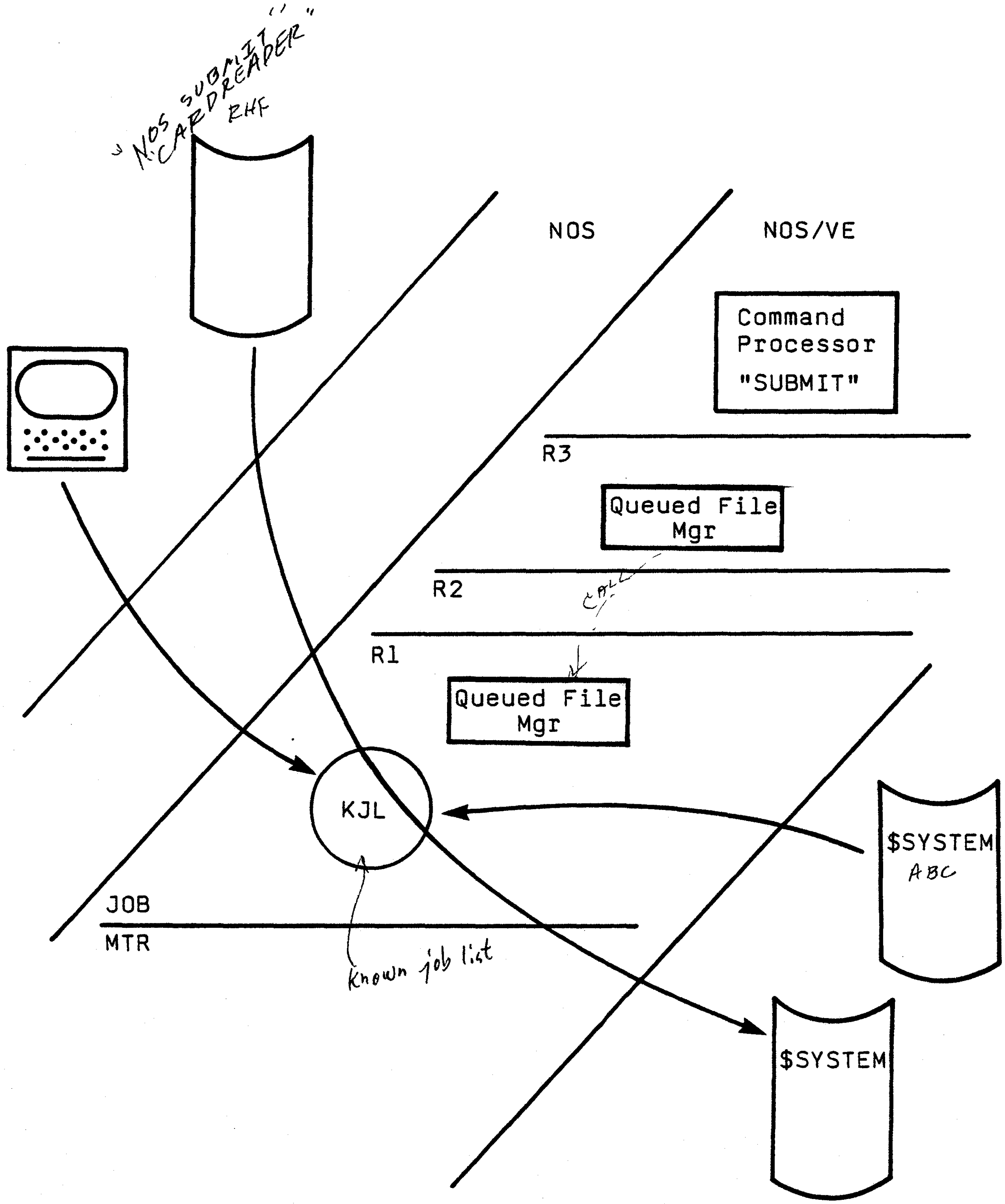
EXERCISE

NONE

SUBMIT JOB

```
   .
   .
   .
/collect_text   ABC
login JHW81
execute DEF                  -----PMP#EXECUTIVE
Logout
**
/submit    ABC               -----JMP$ROUTE
   .
   .
   .
```

1
JOB ENTRY



"NOS SUBMIT"
"CARDREADER"
RHF
NOS
NOS/VE
Command Processor "SUBMIT"
R3
Queued File Mgr
R2
CALL
R1
Queued File Mgr
KJL
$SYSTEM
ABC
JOB
MTR
known job list
$SYSTEM

JOB QUEUEING

* Queued file manager is part of task services. It processes the jmp$route request.

* Queued files are validated and registered in the $SYSTEM catalog and queued through the known job list (KJL).

* The KJL entry for a job is linked into a thread which represents one of the following states.

"Deferred"	waiting for a time interval to elapse
"Queued"	waiting to be initiated
"Initiated"	active, inactive or swapped out but available for execution
"Terminated"	completed but output files queued for disposition

JOB SCHEDULER

* Job scheduler executes as a task in the system job.

* Job scheduler determines:

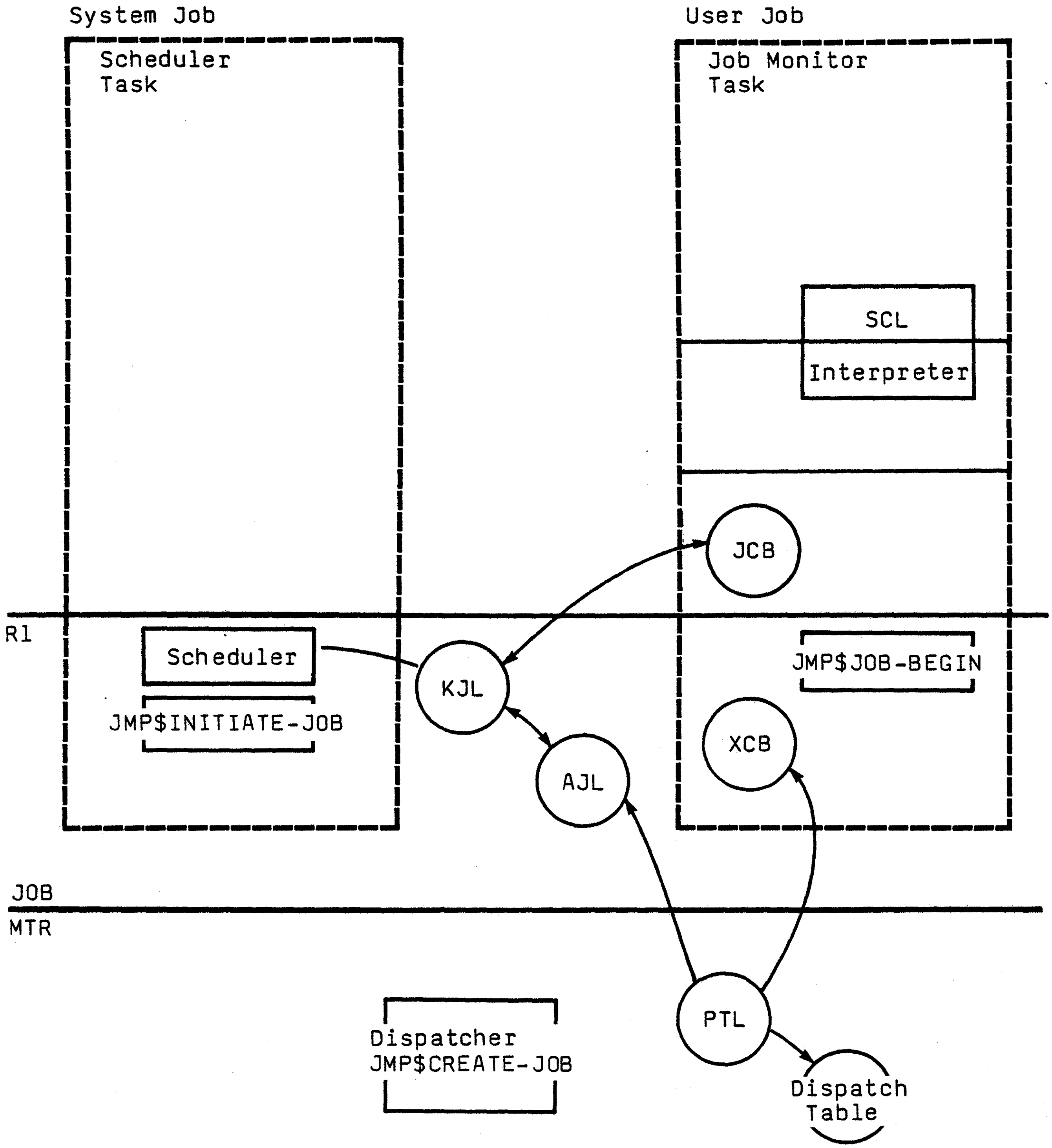
  -Order in which jobs in the input queue should be initiated
  -When a job should be swapped into or out of memory

* Some examples of scheduling criteria are:

  -Current priority within job class
  -Job resource requirements
  -Job class and status
  -Current system resource availability

* Job scheduler monitors the available mix of queued and initiated jobs and prioritizes them based on current system usage.

2
INITIATE JOB



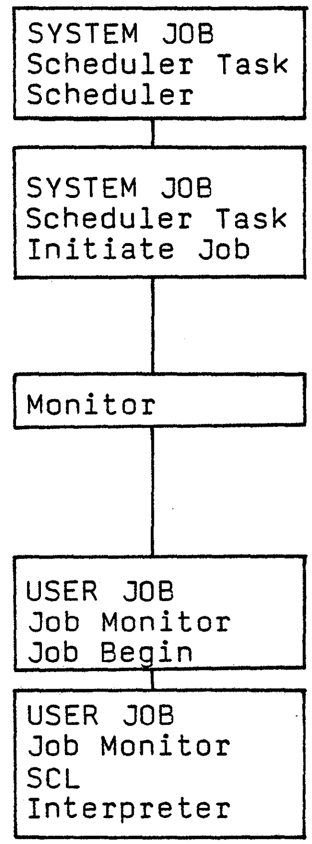
* When a job is selected, it is given an entry in the Active Job List (AJL)

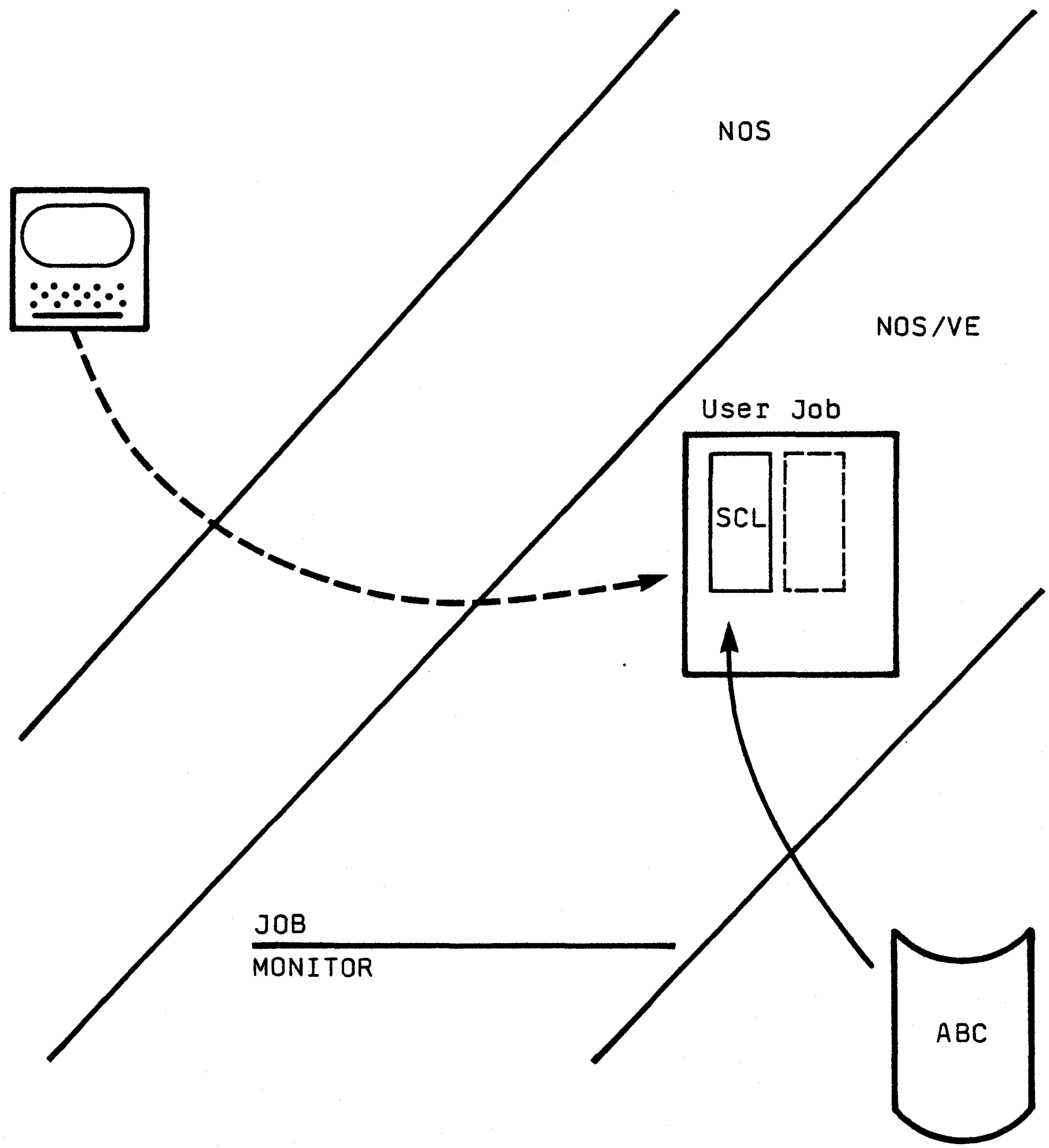
* Initiate_job with the help of monitor initialized the OSS$job_fixed segment for the new job. The Job Control Block (JCB) is built. The Execution Control Block (XCB) is initialized with the XP for the first task to run (Job Monitor)

* Monitor creates a Primary Task List (PTL) entry and logs the job monitor task into the dispatch table. The new job waits its turn. Eventually the dispatcher gives the new job its first time slice.

* Job begin initializes OSS$job_pageable and OSS#task-shared segments. The command file, output file, and job log are also initialized.

* The SCL interpreter interprets the first command (LOGIN). The user is validated and the prolog is executed.




NOS
NOS/VE
User Job
SCL
JOB
MONITOR
ABC

* SCL reads the command from the $COMMAND file.

* SCL searches for the command in the command list, by default -

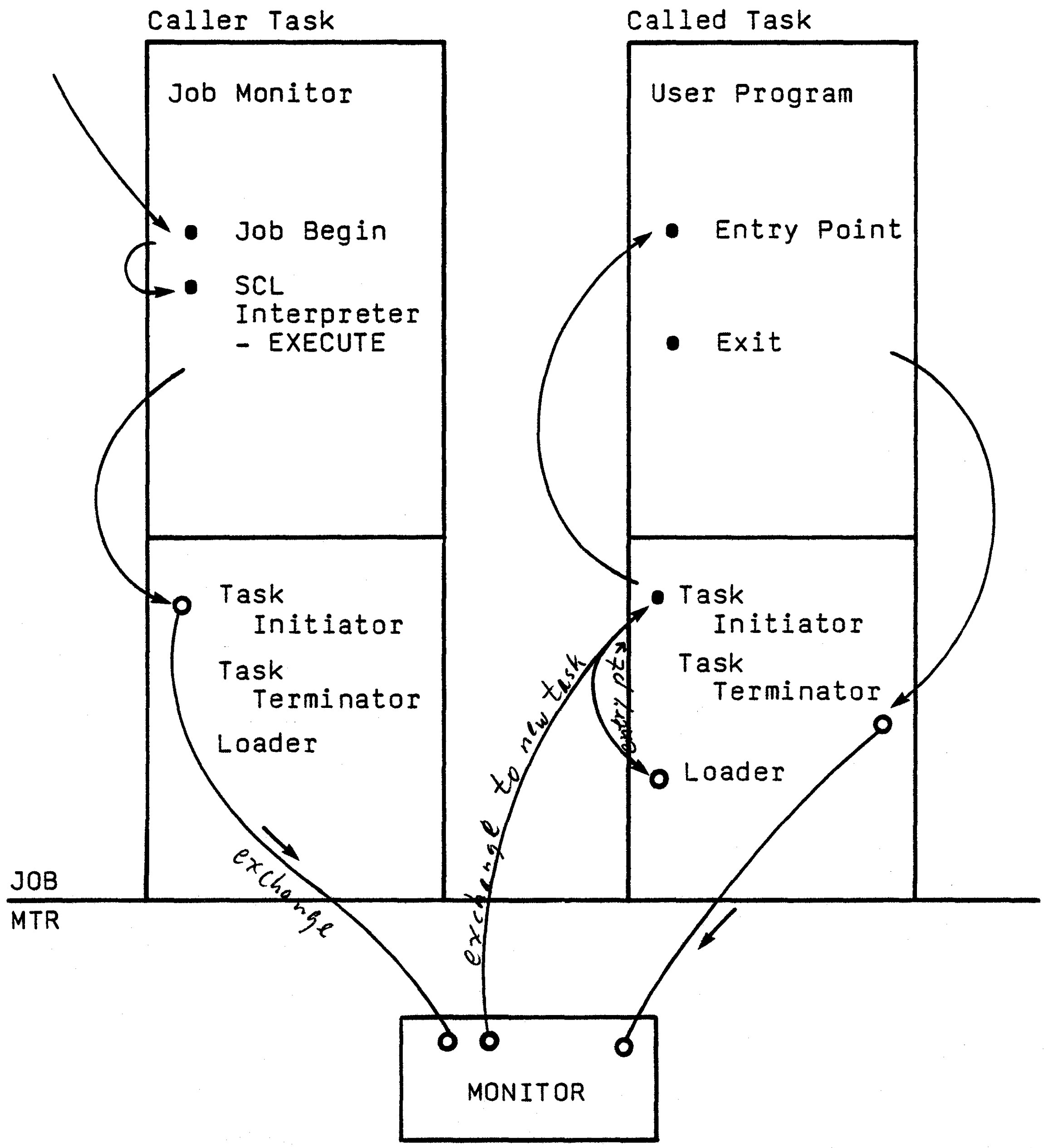
  $LOCAL
  $SYSTEM

* If SCL finds the command it runs the CYBIL procedure which must have been provided to process it. This procedure can run as part of the current task or as a new task.

* If the command is a file name call, it might be a program or an SCL procedure file.

* In all cases, the SCL Interpreter passes the command parameter list to a processor. The processor can now use other SCL interfaces to crack the command.

PROGRAM EXECUTION EXAMPLE



PROGRAM EXECUTION FLOW

* Program or command requests program execution
* Calls task initiator

* Builds tables for the new task

* Exchanges to system monitor to request task initiation

* Links new task into CPU dispatch list

* CPU is dispatched to new task

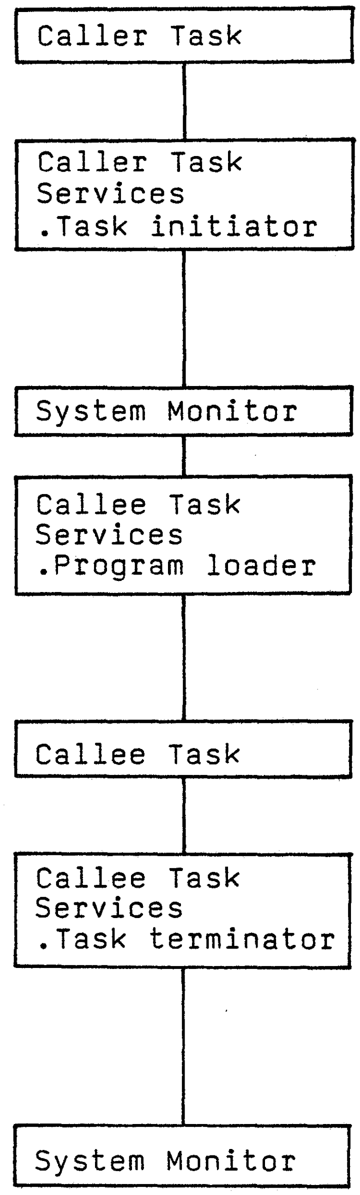
* Loader loads object module
* Loader passes control to initial entry point

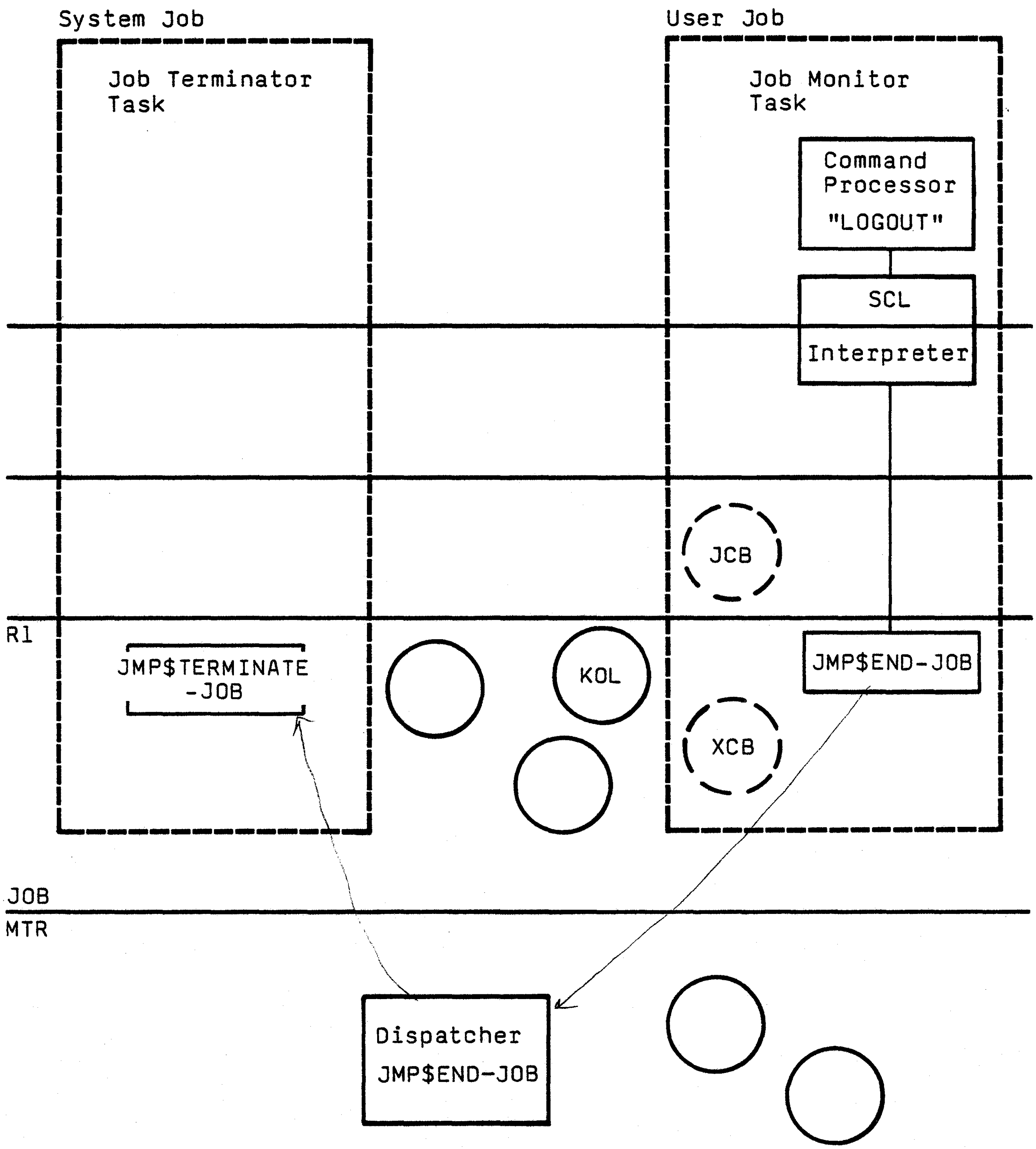
* New task executes asynchronously to caller task

* New task calls exit interface

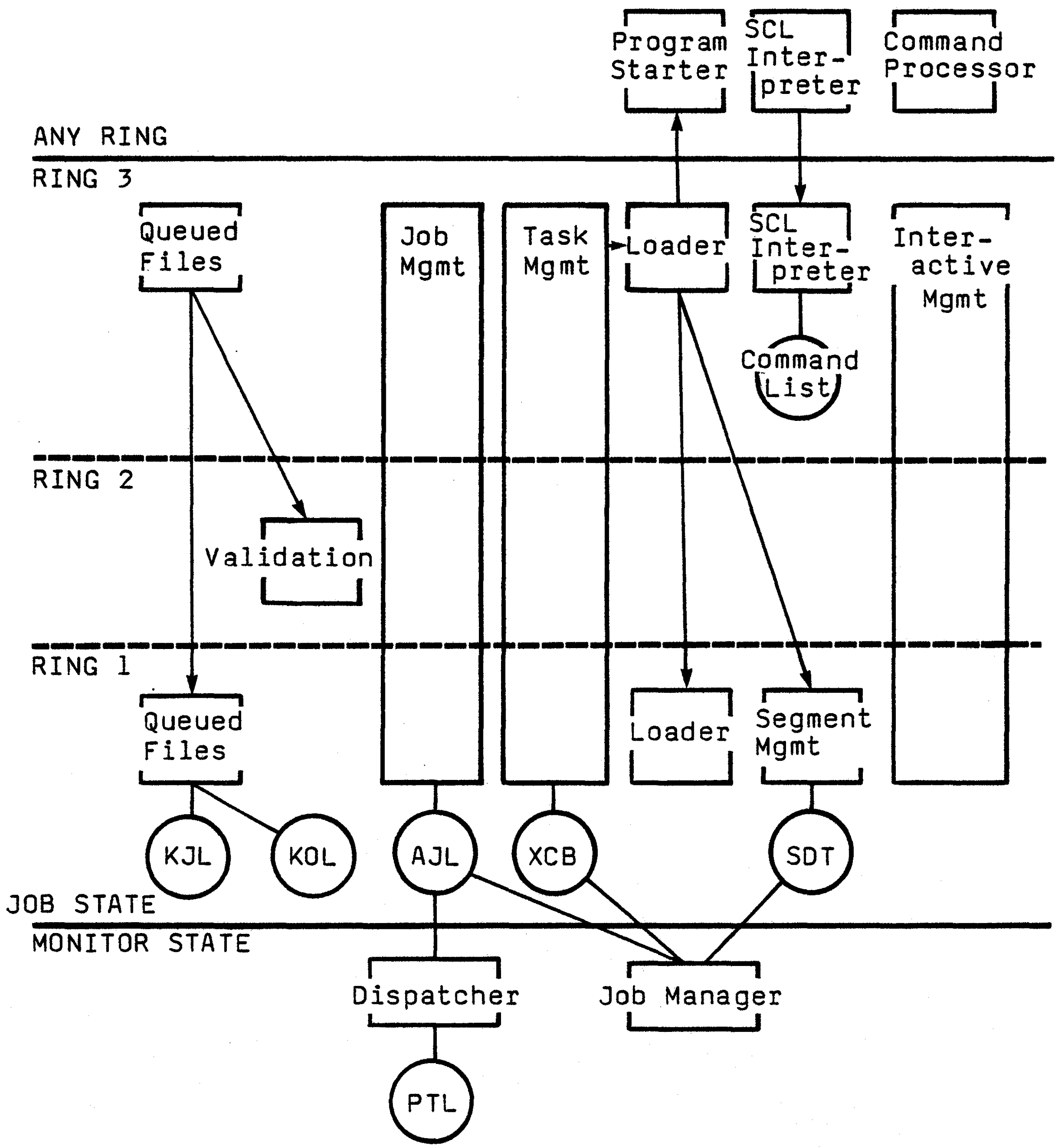
* Cleans up task
* Exchanges to system monitor to request task termination

* Remove task entries from dispatch list
* Informs caller that callee has terminated




System Job
Job Terminator Task
User Job
Job Monitor Task
Command Processor
"LOGOUT"
SCL
Interpreter
JCB
R1
JMP$TERMINATE -JOB
KOL
JMP$END-JOB
XCB
JOB
MTR
Dispatcher
JMP$END-JOB

JOB FLOW
PACKAGING



COMMAND LIST	THIS LIST WILL BE SEARCHED BY SCL INTERPRETER. IF THE COMMAND IS FOUND, A COMMAND PROCESSOR WILL BE CALLED.
KJL-jmv$known-job-list (JMDKJL)	ALL JOBS IN THE SYSTEM HAVE AN ENTRY ON THIS TABLE.
KOL-KNOWN OUTPUT LIST (JMDKOL)	ALL OUTPUT FILES WAITING FOR ROUTING HAVE ENTRIES ON THIS TABLE.
AJL-ACTIVE JOB LIST (JMDAJL)	ALL JOBS THAT HAVE BEEN INITIATED AND ARE NOT SWAPPED HAVE ENTRIES ON THIS LIST.
XCB-EXECUTION CONTROL BLOCK (OSDXCB)	EVERY TASK IN A JOB HAS AN XCB. THIS TABLE CONTAINS THE TASKS EXCHANGE PACKAGE.
SDT-SEGMENT DESCRIPTOR TABLE (OSDSTBL)	ONE SDT PER TASK. EVERY SEGMENT IN EVERY TASK HAS AN ENTRY IN AN SDT. THE STD IS USED BY HARDWARE TO RELATE VM ADDRESS TO REAL MEMORY ADDRESSES.
DISPATCH CONTROL TABLE (TMDDCT)	THE DISPATCHER ORGANIZES TASKS IN THIS TABLE BY PRIORITY AND CHOOSES THE APPROPRIATE CANDIDATE FOR EXECUTION.
PTL-PRIMARY TASK LIST (TMDPTL)	THIS MONITOR TABLE CONTAINS GLOBAL INFORMATION ABOUT EVERY TASK IN THE SYSTEM.
jmv$job-control-block JCB-JOB CONTROL BLOCK (JMDJCB)	THERE IS ONE JCB PER JOB. THE JCB CONTAINS LIMITS, STATISTICS, ETC., FOR THE JOB.

(   ) Common Deck Name

LESSON 4
FILE FLOW

LESSON PREVIEW

OPEN FILE
LOGICAL I/O
PHYSICAL I/O
CLOSE FILE

REFERENCES

GID-PART 1, CHAPTER 3.4, 3.5, 3.7, 3.8

OBJECTIVES

After completing this lesson the student should be able to--

- o TRACE A FILE FROM THE FIRST REFERENCE TO THE FILE TIL IT IS RETURNED
- o DISTINGUISH BETWEEN RECORD AND SEGMENT LEVEL ACCESS TO FILES
- o DESCRIBE THE FLOW OF INFORMATION BETWEEN MEMORY AND DISK OR TAPE
- o LIST THE MAIN TABLES THAT CONTROL FILE

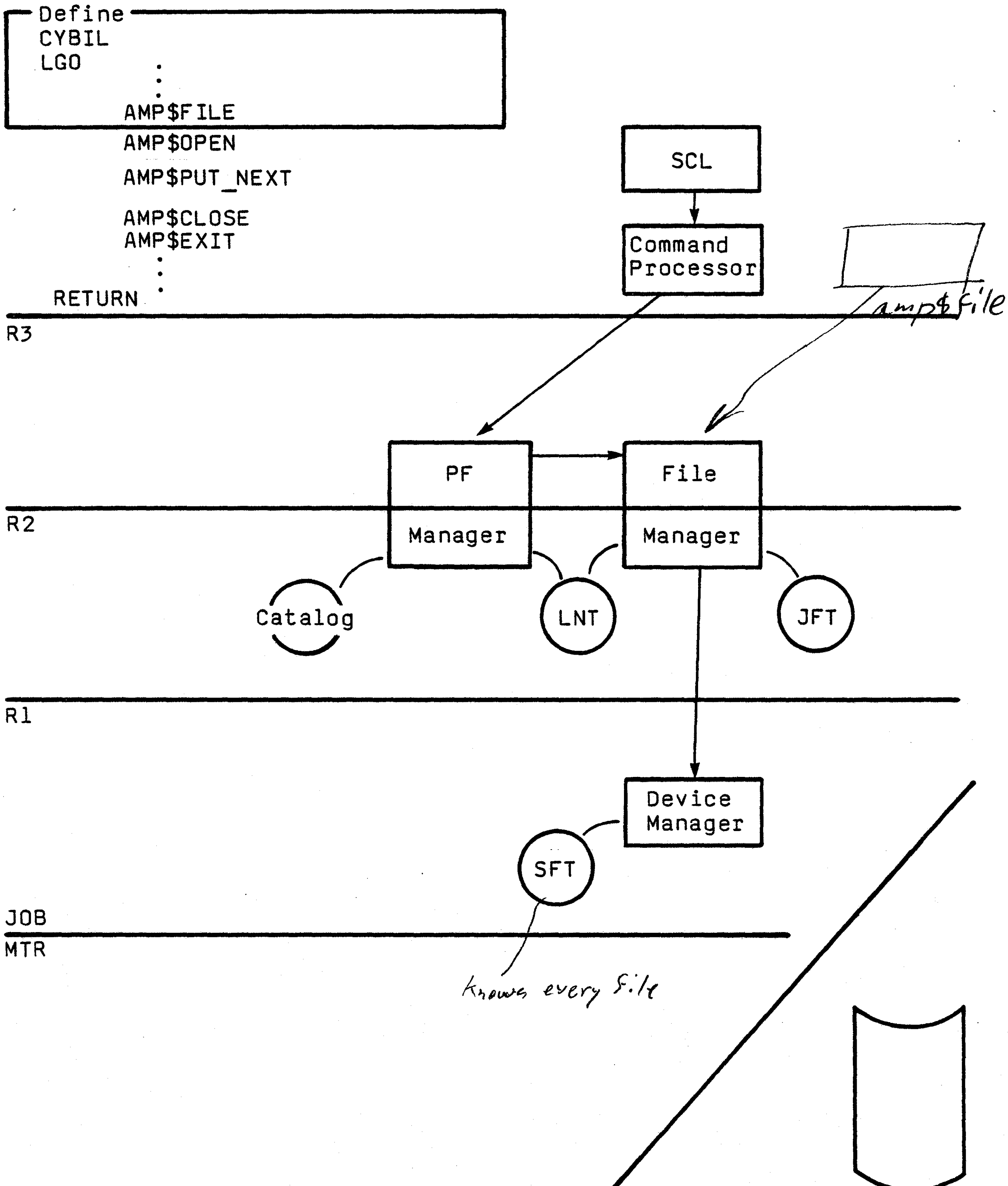
EXERCISES

NONE

CREATE FILE

```
 .
 .
CREATE_FILE
~~DEFINE~~ file = EX
CYBIL ...
LGO

                .
                .
                .
      AMP$FILE (lfn, attributes...
      AMP$OPEN (lfn, amc$record,
        attributes, fid...

      AMP$PUT-NEXT (fid,...

      AMP$CLOSE (fid,...
      PMP$EXIT (status)
RETURN EX   gets rid of local file name
```

1
INITIATE FILE



FILE INITIATION

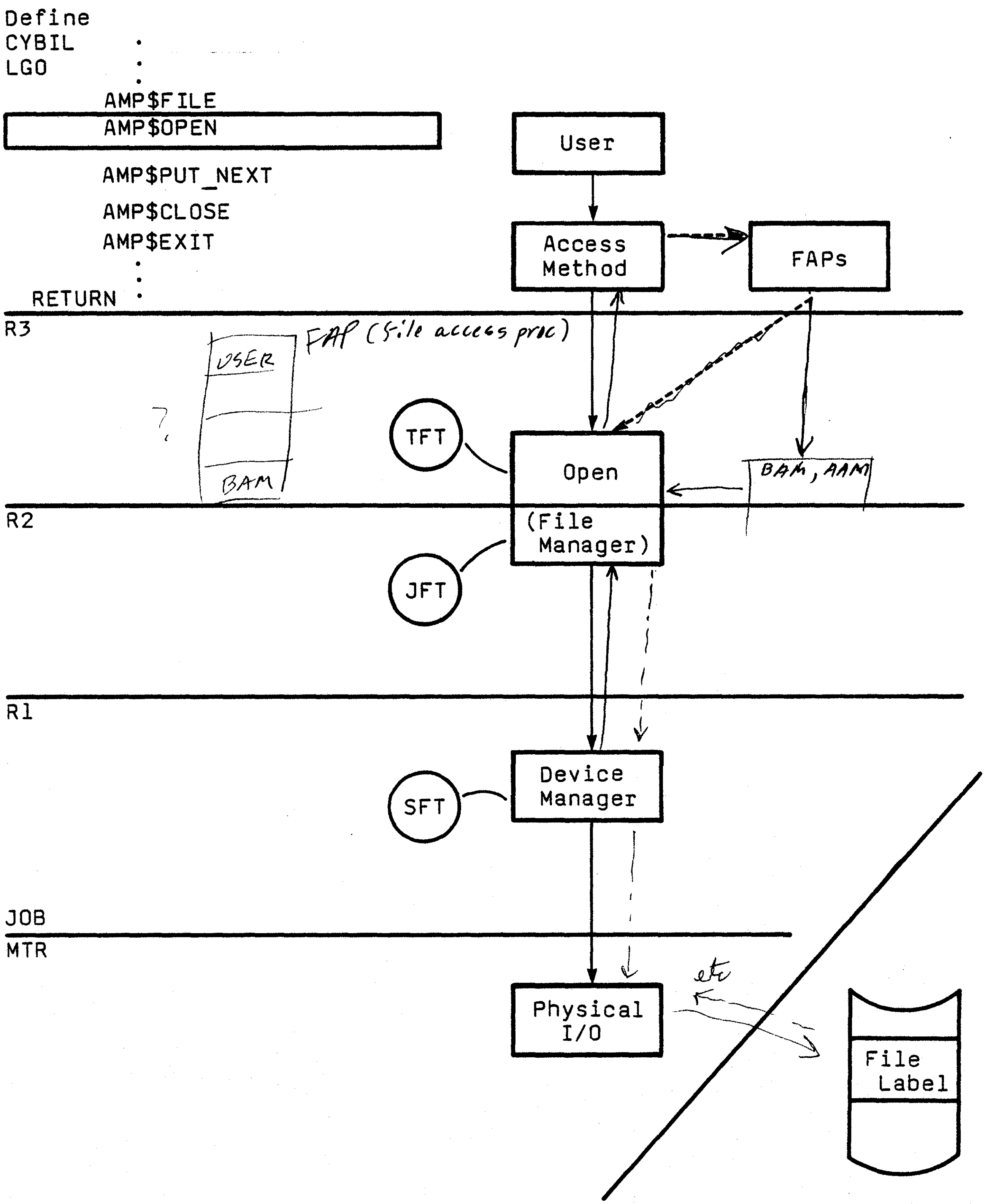
* If the first mention of the file is on a command, then the Job File Table (JFT), the Local Name Table (LNT), and the System File Table (SFT) are built. The commands are:

  REQUEST - TAPE
  REQUEST - TERMINAL
  PRINT
  FILE
  Any PF Command

* For amp$file and some other requests, an auxiliary request table is built. The file tables are built when the file is opened for the first time.

2

OPEN FILE



Define
CYBIL
LGO
AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
AMP$EXIT
RETURN
R3
R2
R1
JOB
MTR
User
Access Method
FAPs
FAP (file access proc)
USER
BAM
?
TFT
Open
(File Manager)
BAM, AAM
JFT
Device Manager
SFT
Physical I/O
etc
File Label

* When the file is opened, the information from commands, program interface requests and AMP$OPEN will be used. The precedence is:

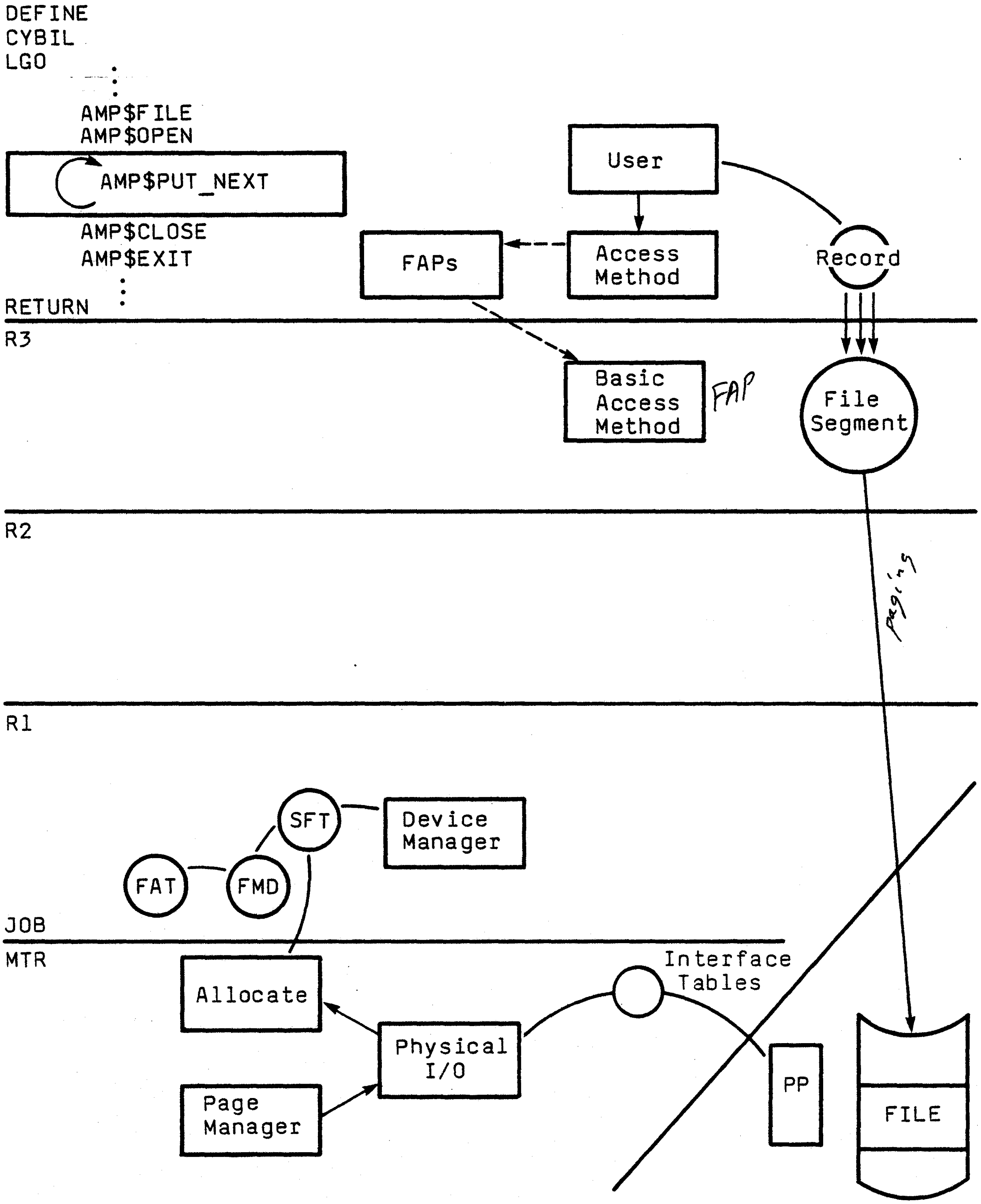
  1. AMP$OPEN
  2. Commands
  3. Requests

* Open entails various processing depending on the file residence and direction of transfer. For example:

DISK	File attributes are read or written
TAPE	Labels are checked or created (R2)
TERMINAL	Attributes are sent to the interactive facility

3

WRITE FILE


DEFINE
CYBIL
LGO
AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
AMP$EXIT
RETURN
R3
R2
R1
JOB
MTR
User
Access Method
FAPs
Record
Basic Access Method
FAP
File Segment
paging
SFT
Device Manager
FAT
FMD
Allocate
Interface Tables
Physical I/O
Page Manager
PP
FILE

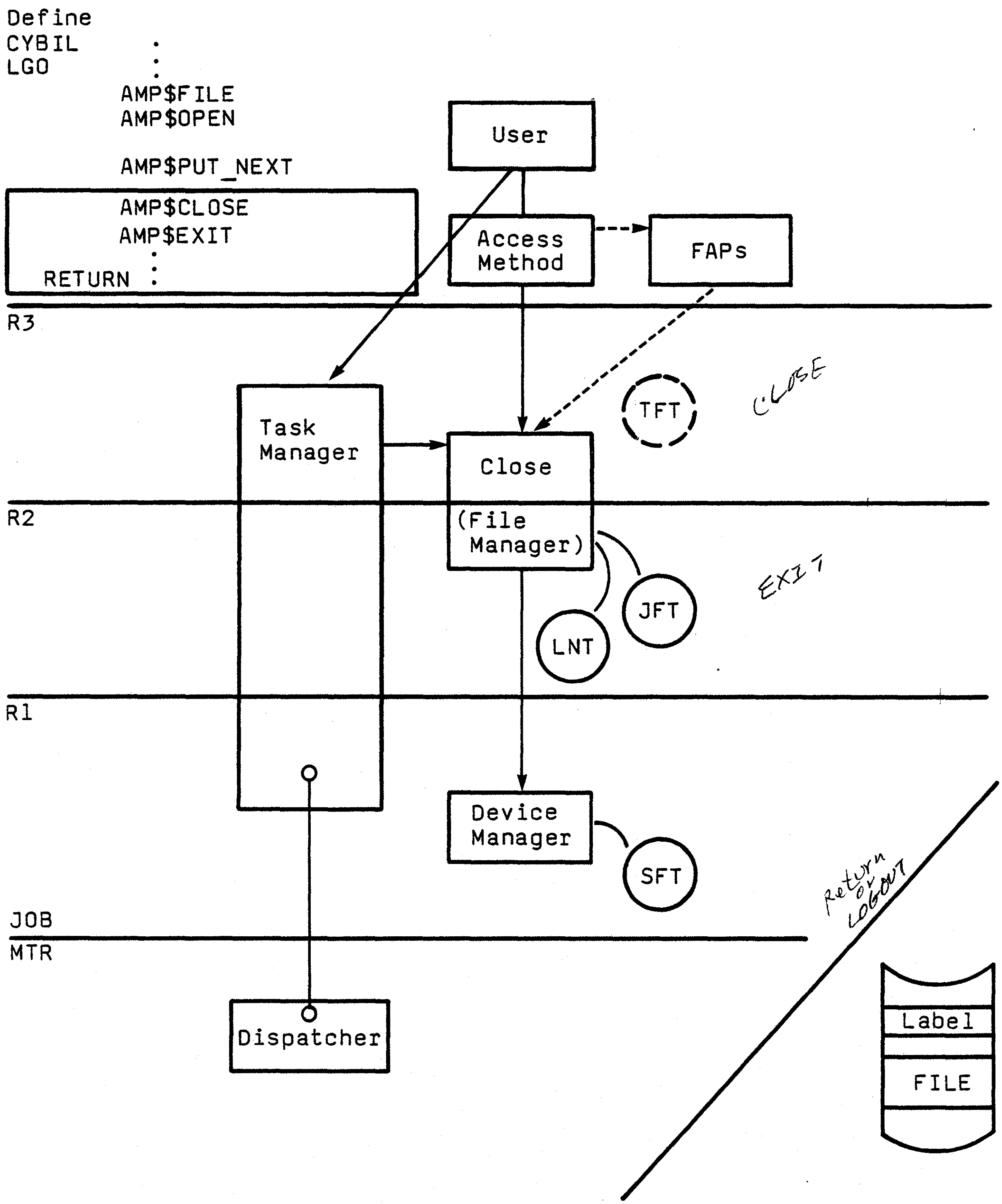
ACCESS LEVELS

* PUT-NEXT
  The access method gets records from the user's buffer and puts them in the file segment that is opened for that purpose (i.e., the system does segment level access). The paging mechanism will take care of real memory and device manager will make sure that space is allocated on the disk. When the filled pages are needed by the system, page manager will instruct the physical I/O component to transfer them.

* GET-NEXT
  Again the access method opens a file segment. Page faults will occur when the needed data is not in real memory. But the access methods is not aware of that; it simply copies the records from the file segment to the user's record buffer.

* Segment Level Access
  If the user opens the file for segment level access, the file segment is directly addressable by the user.

4
CLOSE FILE



Define
CYBIL
LGO
AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
AMP$EXIT
RETURN
User
Access Method
FAPs
R3
Task Manager
Close
TFT
CLOSE
R2
(File Manager)
JFT
LNT
EXIT
R1
Device Manager
SFT
Return or LOGOUT
JOB
MTR
Dispatcher
Label
FILE

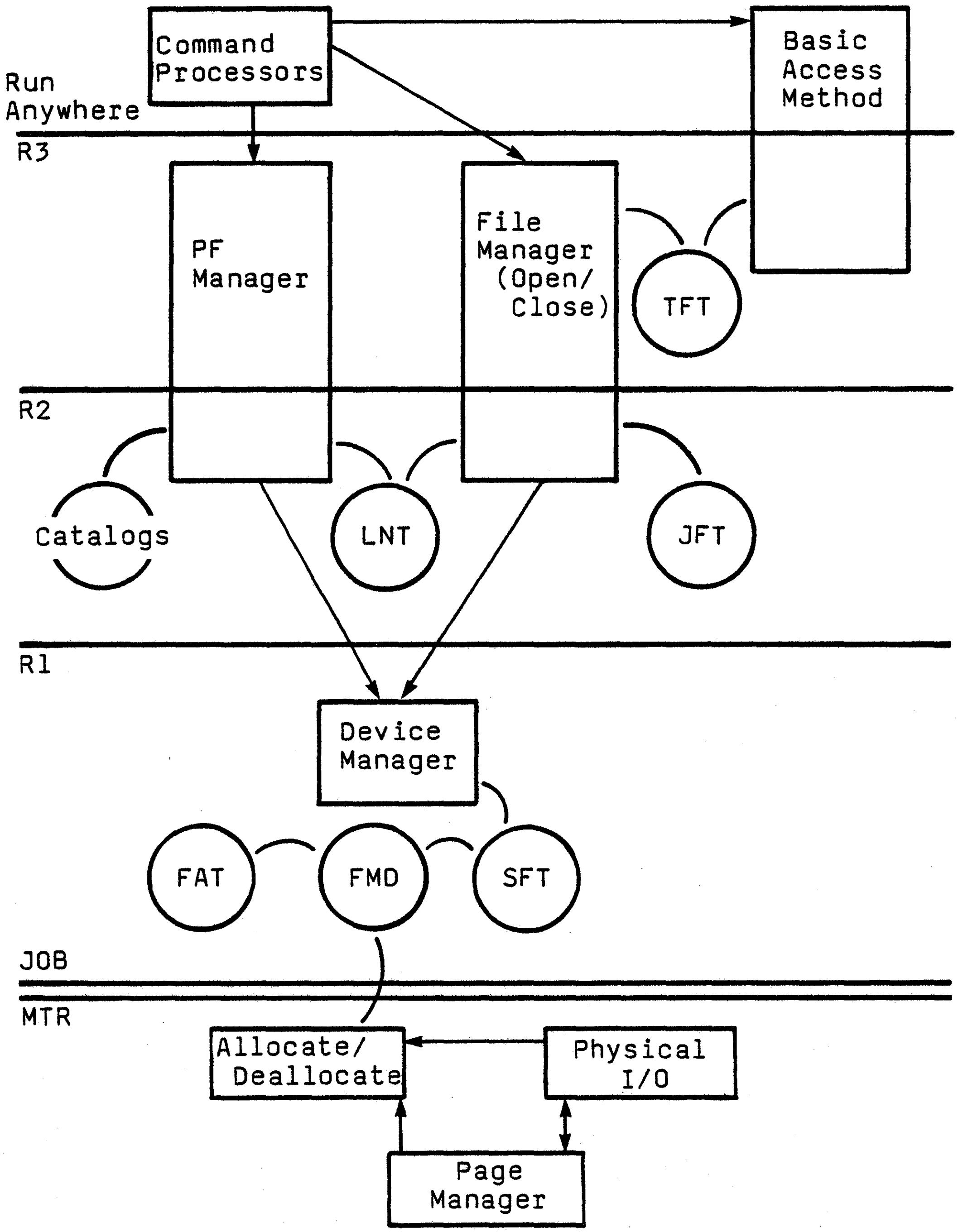
* AMP$CLOSE is a request to close this instance of open. The Task File Table (TFT) will be dismantled if there are no other opens. At task termination (PMP$EXIT, for example) all files in the task will be closed <u>once</u> for each <u>instance of</u> open. Job and System File Tables <u>will</u> remain.

* RETURN will cause all references to the file to be deleted. Examples of file disposition are:

DISK	Temporary files will no longer be accessible. Permanent files will be known through the user's catalog only.
TAPE	Trailer labels will be processed (R2). The volume will be returned.
TERMINAL	Disconnected and returned.

* At job termination all files are closed and returned.

FILE FLOW
PACKAGING



FILE FLOW TABLES

TFT-bat$task_file_table (BADTFT)	All files opened by a task are controlled by this table. Entries contain pointers to record and block descriptors, file attributes and user request tables.
LNT-Local Name Table (FMDLNT)	This table controls the files known to a job by name. It keeps track of the request and attribute info which is global to the job.
JFT-Job File Table (BADJFT)	This table has information about all the files known to the job including unnamed segments like stack and binding.
Catalogs	Each user has a master permanent file catalog.
SFT-System File Tables (DMDSFT)	These tables have entries for all files in the system at a given time. Entries point to tables which describe the file on the device.
FMD-File Medium Descriptor (DMDFMD)	This table lists the volumes on which a file has been allocated.
FAT-File Allocation Table (DMDFAT)	There is one FAT per file. It describes the physical location of the file on the device.
(Common Deck Name)	

LESSON 5
MATERIALS

LESSON PREVIEW

ORGANIZATION OF THE NOS/VE PROJECT
DOCUMENTATION
STRUCTURE AND CONTENT OF SOURCE LIBRARIES
LOAD MAP
SYSTEM INITIALIZATION

OBJECTIVES

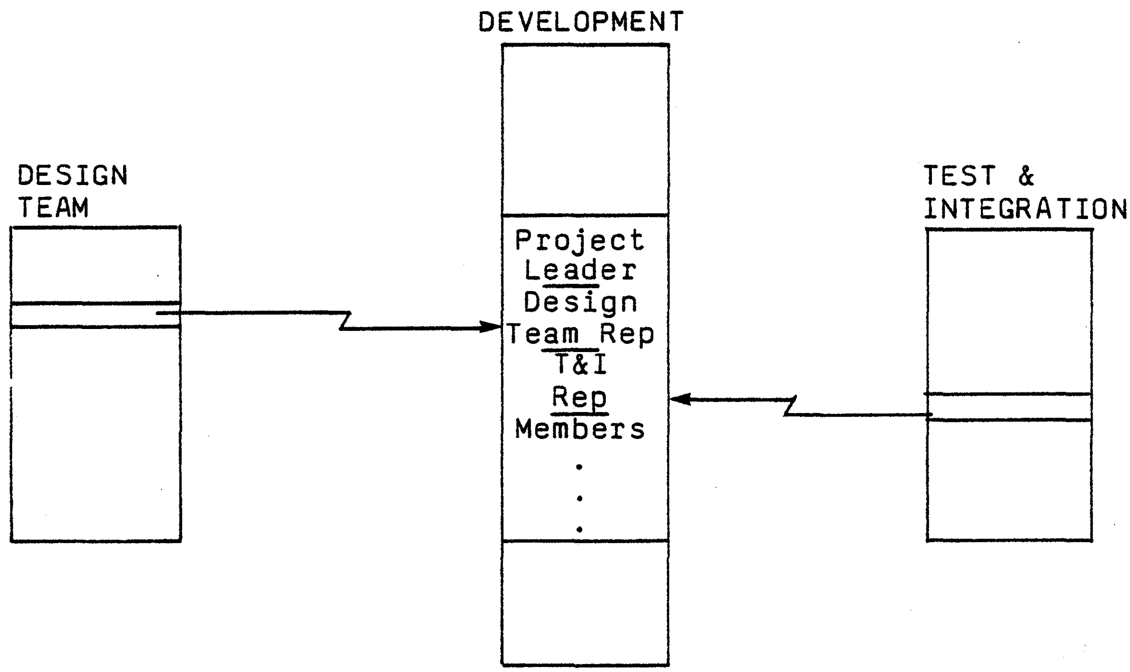
After completing this lesson the student should be able to--

- GET COPIES OF ALL IMPORTANT NOS/VE DOCUMENTS.
- FIND AND LIST NOS/VE SOURCE DECKS
- DESCRIBE HOW THE SOURCE LIBRARIES ARE ORGANIZED
- FIND THE PEOPLE IN THE DEVELOPMENT ORGANIZATION WHO HANDLE CERTAIN AREAS OF THE SYSTEM
- INTERPRET A LOAD MAP
- DESCRIBE NOS/VE DEVELOPMENT ORGANIZATION
- OUTLINE THE SYSTEM INITIALIZATION PROCESS

EXERCISES

1. GIVEN COMMON DECKS AND AN IDENTIFIED TABLE, INTERPRET SOME FIELDS IN THE TABLE.
2. GIVEN AN ADDRESS, FIND THE NAME OF THE MODULE IN A LOAD MAP AND FIND THE CODE IN THE SOURCE LIBRARY

NOS/VE PROJECT ORGANIZATION



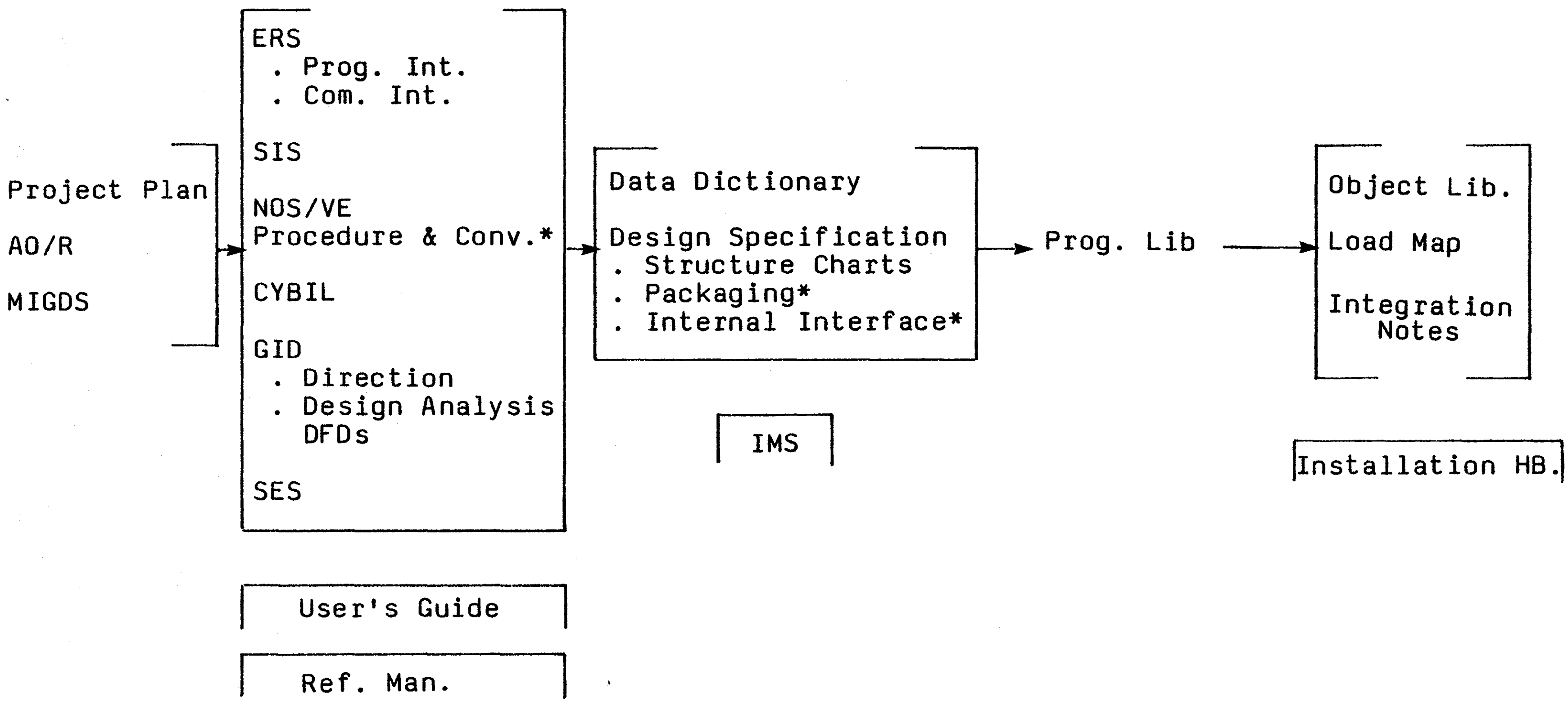
DESIGNERS

- Job Mgmt
- Program Mgmt
- I/O
- Dual State
- Deadstart

DEVELOPMENT GROUPS

- PFs
- Physical I/O
- Logical I/O
- Dual State Communication
- Logs
- Program Control
- Program Execution
- Job Mgmt
- Command Language
- Monitor
- Maintenance
- Deadstart

MATERIALS



*Class Handout

NOTE: Annotated bibliography in appendix.

NOS/VE PROCEDURES & CONVENTIONS

PREFIX NOMENCLATURE

SYNTAX:

XXC$ . . . = Constant

XXT$ . . . = Type

XXE$ . . . = Error Code

XXP$ . . . = Procedure

XXM$ . . . = Module

XXV$ . . . = Variable

XXK$ . . . = Keypoint

ID CODE (XX):

AM = Access Methods

CL = Command Language

IC = Interstate Communication

IF = Interactive Facility

JM = Job Management

OF = Operator Facility

OS = Operating System

PF = Permanent Files

PM = Program Management

RM = Resource Management

SF = Statistics Facility

MM = Memory Management

FM = File Management

MT = Monitor

LO = Loader

CI = Common I/O

CY = CYBIL

IO = Input/output

MS = Maintenance Services

DB = Debug

SH = Signal Handler

BA = Basic Access

RH = Remote Host

ML = Memory Link

II = Interactive Interface

QF = Queued File

DP = Display

SY = System

ST = Sets

TM = Task Management

DM = Device Management

LG = Logs

LN = Local Name

AV = Accounting/Validation

LU = Link User

HP = Heap Processor

DECK NAMING CONVENTION

pptzzzz

pp = two character identifier

t = deck type

zzzz = mnemonic !? name

DECK TYPES

- M = CYBIL
- P = PP Assembler
- A = CP Assembler
- X = XREF declarations*
- D = Type and Constant declarations*
- H = Documentation Header*
- I = In-line procedure*
- E = Example

* = common deck

- Chapter Descriptions
- Procedure Descriptions

  -Request Description
  -Parameter Description
  -XREF Declarations
  -Common Deck Calls

- Common Deck Expansions
- Topics

  CP MONITOR
  Job Management
  Resource Management
  Segment/Memory Mgmt
  Memory Mgmt
  Queued Files
  Program Mgmt
  Preemptive Communication
  File Mgmt
  PF Mgmt
  SCL
  Interstate Com.
  Memory Link
  Log Mgmt
  System Access
  Accounting
  Operator Facility

- Intrinisics

INTRINSICS

1 #CALLER_ID (ID)
2 #CALL_MONITOR (REQBLK)
3 #COMPARE (S1,S2): RESULT
4 #COMPARE_COLLATED (S1, S2, TABLE): RESULT
5 #COMPARE_SWAP (LOCK, INITIAL, NEW, ACTUAL, RESULT
6 #DISABLE_TRAPS (OLD_TE)
7 #ENABLE_TRAPS (OLD_TE)
8 #FREE_RUNNING_CLOCK (PORT): INTEGER
9 #HASH_SVA (SVA, INDEX, COUNT, FOUND)
10 #INTERRUPT_PROCESSOR (PORT_SELECTOR)
11 #KEYPOINT (CLASS, EXPRESSION, CODE)
12 #MOVE (SOURCE, DESTINATION, LENGTH)
13 #OFFSET (PVA): INTEGER
14 #PREVIOUS_SAVE_AREA: POINTER
15 #PROGRAM_ERROR
16 #PTR: (DISP, BASE_POINTER): CELL
17 #PURGE_BUFFER (OPTION, ADDRESS)
18 #READ_REGISTER (REGID): REGISTER_VALUE
19 #REAL_MEMORY_ADDRESS (PVA,RMA)
20 #REL (POINTER, BASE_POINTER): INTEGER
21 #RING (PVA): 0 .. 15
22 # RESTORE_TRAPS (OLD_TE)
23 #SCAN (SELECT, STRING, INDEX, FOUND)
24 #SEGMENT (PVA):)..4995
25 #STORE_BIT (BIT_VALUE, BIT_VARIABLE)
26 #TEST_ALTER_CONDITION_REG (SELOPT, BITNUM, BRANCH_EXIT)
27 #TEST_SET_BIT (BIT_VARIABLE, PREVIOUS_VALUE)
28 #TRANSLATE (TABLE, SOURCE, DESTINATION)
29 #WRITE_REGISTER (REGID, REGISTER_VALUE)

Note: see Internal Interface

SYSTEM INITIALIZATION PROCESS

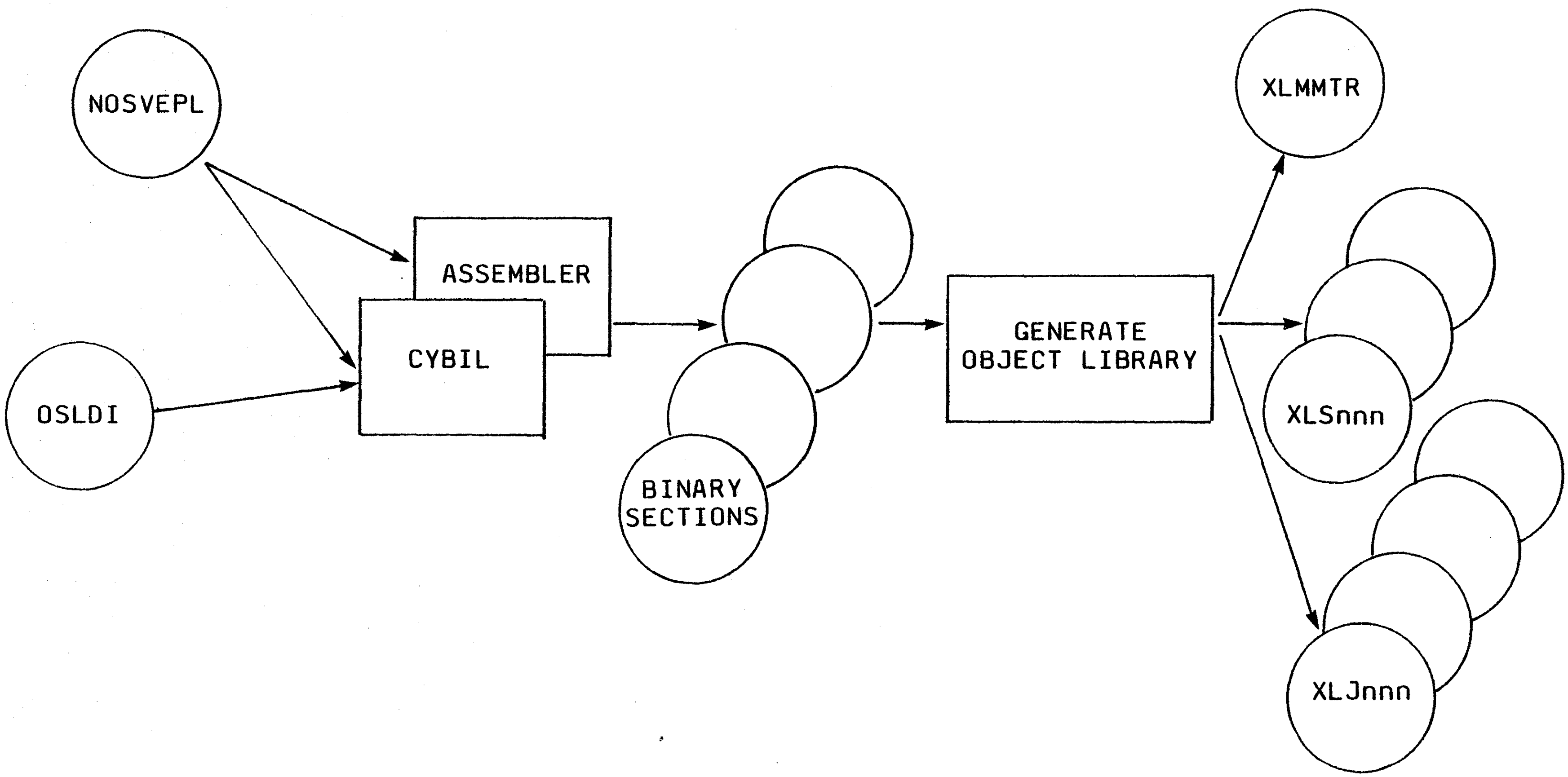
1. Library Generation

2. System Generation

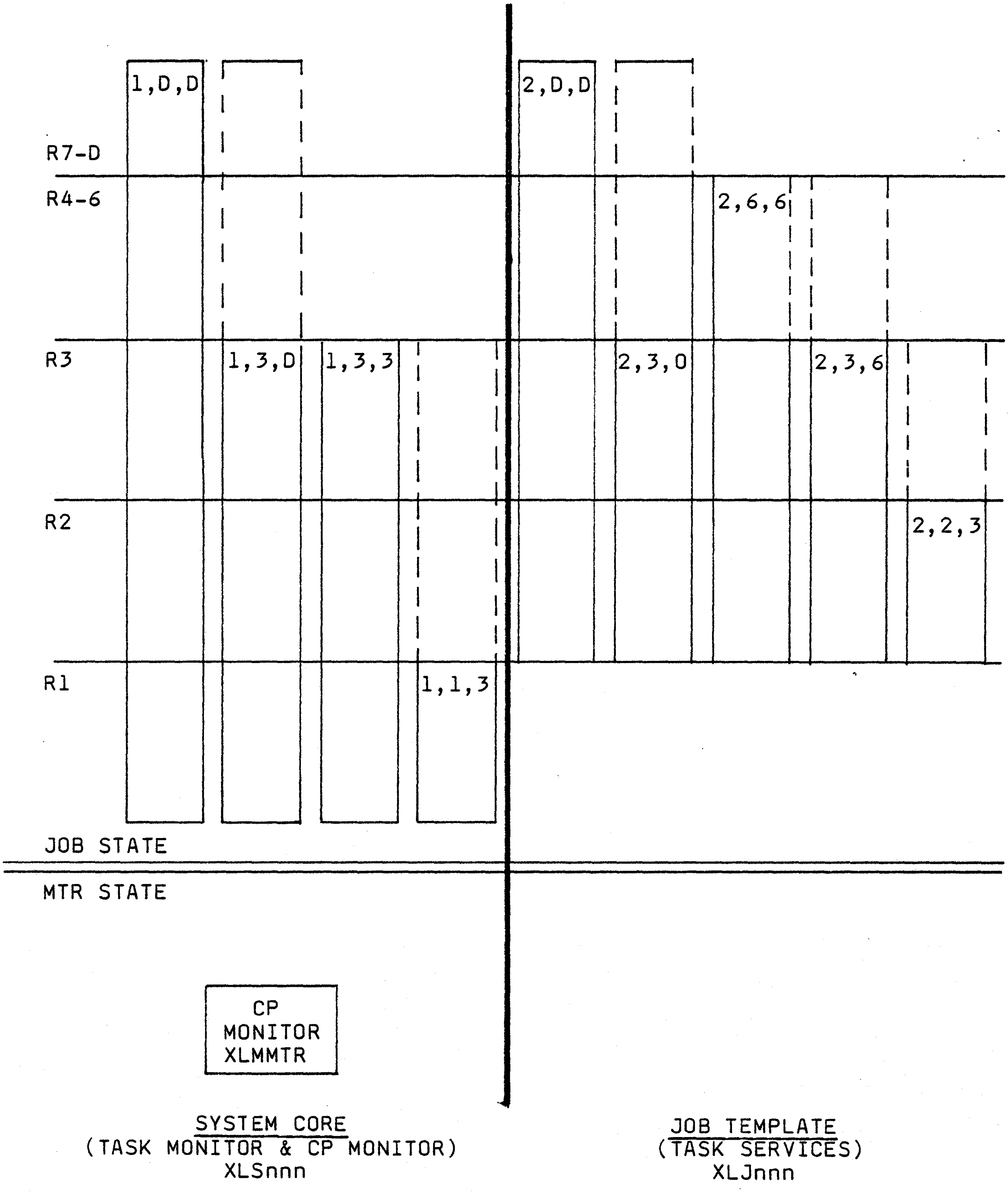
3. System Initialization

4. System Library and Task Initialization

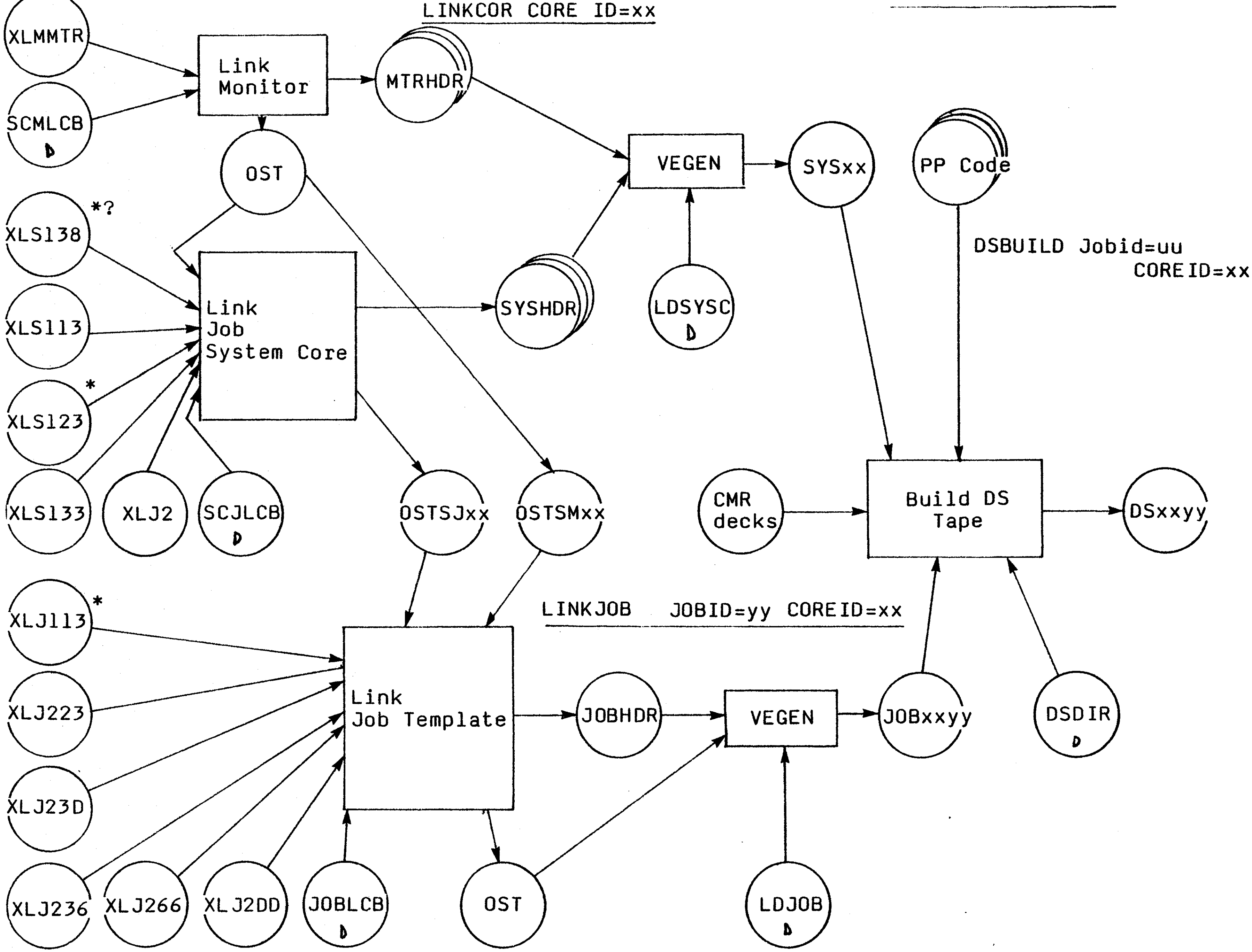
GENERATE LIBRARY



CODE SEGMENTS



SYSTEM GENERATION



Notes:
* - Temporary library, will be deleted.
Ⓓ - directive file.

FILE DESCRIPTIONS

NAME	AREA	TYPE	COMMENTS
NOSVEPL		PL	Contains all code & data source for NOS/VE except program interface.
OSLPI		PL	Program Interface
XLMMTR	Monitor	Object Lib.	Library of monitor modules
XLSnnn	System Core	Object Lib.	Library to run in rings (n,n,n) Task monitor.
XLJnnn	Job Template	Object Lib.	Library to run in rings (n,n,n) Task services.
SCMLCB	Monitor	Directives	System Core/Monitor State Linker Control Block
SCJLCB	System Core	Directives	System Core/Job State Linker Control Block
JOBLCB	Job Template	Directives	Linker Control Block
OST			Outboard Symbol Table. List of gated entry points.
OSTSJxx	System Core	OST	System Core/Job State OST. System with id=xx.
OSTSMxx	Monitor	OST	System Core/Monitor State OST. System id=xx.
MTRHDR	Monitor	Segment Files	This HDR describes a collection of "seed" files with names MTR101, MTR102, etc.
SYSHDR	System Core	Segment Files	This HDR file describes a collection of "seed" files with names SYS101, SYS102, etc.
JOBHDR	Job Template	Segment Files	This HDR file describes a collection of "seed" files with names JOB101, JOB102, etc.

NAME	AREA	TYPE	COMMENTS
LDSYSC	System Core	Directives	Load Directives for use by the Virtual Environment Generator (VEGEN) to build the system core memory image.
LDJOB	Job Template	Directives	Load Directives for use by VEGEN to build a job template memory image.
SYSxx	System Core	Memory Image	This core is sufficient to initialize a system with id=xx in 1M bytes.
JOBxxyy	Job Template	Memory Image	This template will run under the system with id=xx. The id of the template is yy.
CMR			
PP Code			Set of peripheral drivers to run in the PPs.
DSDIR		Directives	Control the building of the DS Tape.
DSxxyy		DS Tape	

Common test and initialization
MCU deadstart monitor
Configuration records
Basic O/S deadstart job (CM image)
System RMS controlware
System RMS driver
Other system drivers and controlware
Balance of NOS/VE (load modules)
Product set libraries

CYBER 180 deadstart file format

- Deadstart function activates NOS/VE to the state in which it is ready to execute user workloads

- Includes recovery or initialization of
  - Permanent file bases
  - System log files
  - I/O queues
  - Hardware configuration information
  - User/system jobs and their transient files

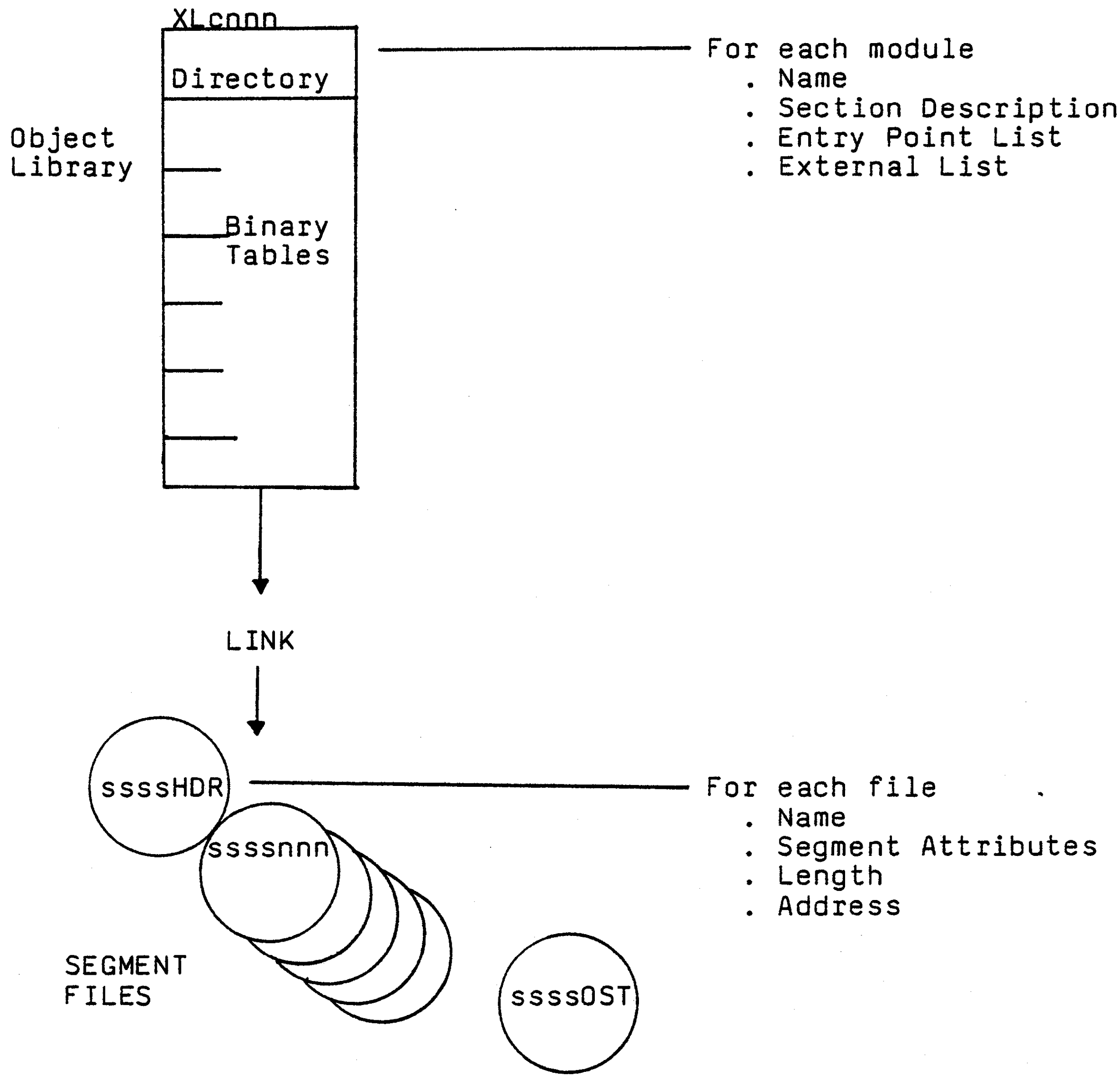
- Deadstart/recovery function supports both dual-state and stand-alone CYBER 180 operation

- Deadstart levels

  1 Installation

  2 Recovery

  3 Continuation


XLcnnn
Directory
Object
Library
Binary
Tables
For each module
. Name
. Section Description
. Entry Point List
. External List
LINK
ssssHDR
ssssnnn
SEGMENT
FILES
ssssOST
For each file
. Name
. Segment Attributes
. Length
. Address

```
MODULE   = CLM$PROGRAM_EXECUTION_COMMANDS   LANGUAGE = CYBIL
FILE     = XLJ200      09/07/81      20:09:30
-------------------------------------------------------------
CODE - RE_200
READ EXECUTE                              29F0    1 018 0002AC60
BINDING - RB_XXX
READ BINDING                              210     1 00C 0000FC40
WORKING STORAGE - RE_200
READ                                      704     1 018 00020650
WORKING STORAGE - OSS$JOB_PAGED_LITERAL
READ                                      90      1 00A 00009048
WORKING STORAGE - CLS$POT
READ                                      12A0    1 00A 00009008
WORKING STORAGE - CLS$ADT
READ                                      9F      1 00A 0000A478

ENTRY POINT DEFINITIONS                           ADDRESS
  CLP$SET_OBJECT_LIST_COMMAND                   1 018 0002AC6C
  CLP$SET_PROGRAM_OPTIONS_COMMAND               1 018 0002B000
  CLP$EXECUTE_COMMAND                           1 018 0002B640
  CLP$TERMINATE_TASK_COMMAND                    1 018 0002C400
  CLP$WAIT_COMMAND                              1 018 0002C5F0
  CLP$$TASK_COMPLETE                            1 018 0002C9A0
  CLP$$TASK_STATUS                              1 018 0002CAE0
  CLP$DISPLAY_PROGRAM_COMMAND                   1 018 0002CF40

EXTERNAL ENTRY POINTS REFERENCED
  CLP$CREATE_NAMED_TASK_ENTRY           CLP$DELETE_NAMED_TASK_ENTRY
  CLP$FIND_NAMED_TASK_ENTRY             CLP$GET_SET_COUNT
  CLP$GET_VALUE                         CLP$SCAN_PARAMETER_LIST
  CLP$SCAN_ARGUMENT_LIST                CLP$CONVERT_INTEGER_TO_STRING
  CLP$CONVERT_VALUE_TO_STRING           CLP$OPEN_DISPLAY
  CLP$CLOSE_DISPLAY                     CLP$PUT_PARTIAL_DISPLAY
  CLP$PUT_DISPLAY                       OSP$SET_STATUS_ABNORMAL
  OSP$APPEND_STATUS_PARAMETER           OSP$AWAIT_ACTIVITY_COMPLETION
  PMP$CHANGE_DEFAULT_PROG_OPTIONS       PMP$CHANGE_JOB_LIBRARY_LIST
  PMP$ESTABLISH_CONDITION_HANDLER       PMP$EXECUTE
  PMP$GET_JOB_LIBRARY_LIST              PMP$GET_DEBUG_LIBRARY_LIST
  PMP$GET_NUMBER_OF_JOB_LIBRARIES       PMP$GET_NUMBER_OF_DEBUG_LIBS
  PMP$GET_DEFAULT_PROGRAM_OPTIONS       PMV$PRESET_CONVERSION_TABLE
  PMP$TERMINATE                         CYP$ERROR
  PMP$CONTINUE_TO_CAUSE                 CYP$NIL
```

```
SES/C180 LINKER OUTPUT
FILE NAME/                                  LOAD/
  SECTION NAMES
------------------------------------------------------------
HDWX101
READ WRITE                          1721    * 002 00000000
  RW_113
  OSS$MAINFRAME_PAGEABLE

HDWX102
WRITE EXTENSIBLE                    2328    * 003 00000000

HDWX103
READ WRITE                          1AC8    * 004 00000000
  OSS$JOB_PAGEABLE

HDWX104
READ WRITE                          3CEA    * 005 00000000
  OSS$TASK_PRIVATE

HDWX105
READ WRITE                          183A    * 006 00000000
  OSS$TASK_SHARED

HDWX106
READ                               171A3    * 00A 00000000
  OSS$MAINFRAME_PAGED_LITERAL
  OSS$JOB_PAGED_LITERAL
  CLS$PDT
  CLS$ADT

HDWX107
READ EXECUTE-LOCAL PRIV            2F487    * 00B 00000000
  RE_113

HDWX108
BINDING                            12542    * 00C 00000000
  RB_XXX
```

```
HDWX109
READ WRITE EXTENSIBLE                    400    * 00D 00000000

HDWX110
READ WRITE EXTENSIBLE                    800    * 00E 00000000

HDWX111
READ WRITE EXTENSIBLE                    200    * 00F 00000000
HDWX112
READ WRITE EXTENSIBLE                      8    * 010 00000000

HDWX113
READ EXECUTE-LOCAL PRIV                    8    * 011 00000000
  RE_123

HDWX114
READ WRITE EXTENSIBLE                      9    * 012 00000000

HDWX115
READ EXECUTE-LOCAL PRIV                 3700    * 013 00000000
  RE_13X

HDWX116
READ WRITE                               180    * 014 00000000
  RW_13X

HDWX117
READ EXECUTE-LOCAL PRIV                 2448    * 015 00000000
  RE_10D

HDWX118
READ EXECUTE-LOCAL PRIV                37300    * 016 00000000
  RE_223

HDWX119
READ EXECUTE-LOCAL PRIV                77548    * 017 00000000
  RE_23X

HDWX120
READ EXECUTE-LOCAL PRIV                7D96E    * 018 00000000
  RE_20D
```

LESSON 6
INTERNAL COMMUNICATION

LESSON PREVIEW

- REGISTERS AND EXCHANGE PACKAGES
- INTERPRET THE STACK
- SIGNALS, SYSTEM FLAGS, AND MONITOR FAULTS
- INTERRUPTS AND TRAPS
- CP MONITOR
- Trap Handler

REFERENCES

MIGDS
GID-PART 3 (PACKAGING)

OBJECTIVES

After completing this lesson the student should be able to--

- INTERPRET THE CONTENTS OF EXCHANGE PACKAGE REGISTERS
- INTERPRET THE SAVE AREA, AUTOMATIC VARIABLES AND PARAMETERS IN A STACK
- EXPLAIN HOW THE EXCHANGE INTERRUPT IS PROCESSED AND HOW THE SIGNALS, SYSTEM FLAGS AND MONITOR CONDITIONS ARE PASSED TO A TASK
- EXPLAIN HOW A TRAP INTERRUPT IS PROCESSED
- GIVEN A CRASH DUMP, DETERMINE WHAT WAS RUNNING WHEN THE CRASH OCCURRED
- OVERVIEW THE CP MONITOR FUNCTIONS
- EXPLAIN WHAT ACTION IS TAKEN BY CP MONITOR ON EACH OF THE MONITOR CONDITIONS
- SHOW HOW A MONITOR REQUEST IS MADE FROM JOB STATE
- LIST THE REQUESTS THAT MONITOR IS PREPARED TO PROCESS

EXERCISES

1. GIVEN A DUMP, DETERMINE WHERE THE SYSTEM WAS PROCESSING WHEN THE DUMP WAS TAKEN.
2. GIVEN A STACK, FIND THE CURRENT FRAME AND TRACE THE CALL CHAIN.
3. DETERMINE WHAT PARAMETERS WERE PASSED TO A PROCEDURE AND WHAT THE VALUE OF EACH PARAMETER IS.

COMMUNICATION

MECHANISMS

Call
Return
Exchange Jump
Exchange Interrupt
Trap Interrupt

PROCESSORS

Monitor Interrupt Processor (MIP)
- Request Processors

Trap Handler
- Signal Handler
- System Flag Handler
- Monitor Fault Handlers

STACK DATA MAPPING

1. SFSA-stack frame save area

    - Typically words 0-4. length in word 2
    - See diagram and CYBIL definition

2. Automatic Variables

    - First two words not used
    - Each variable starts a new word
    - Array and record components are byte aligned unless packed
    - Packed components are bit aligned except characters, integers, and pointers

3. Parameters

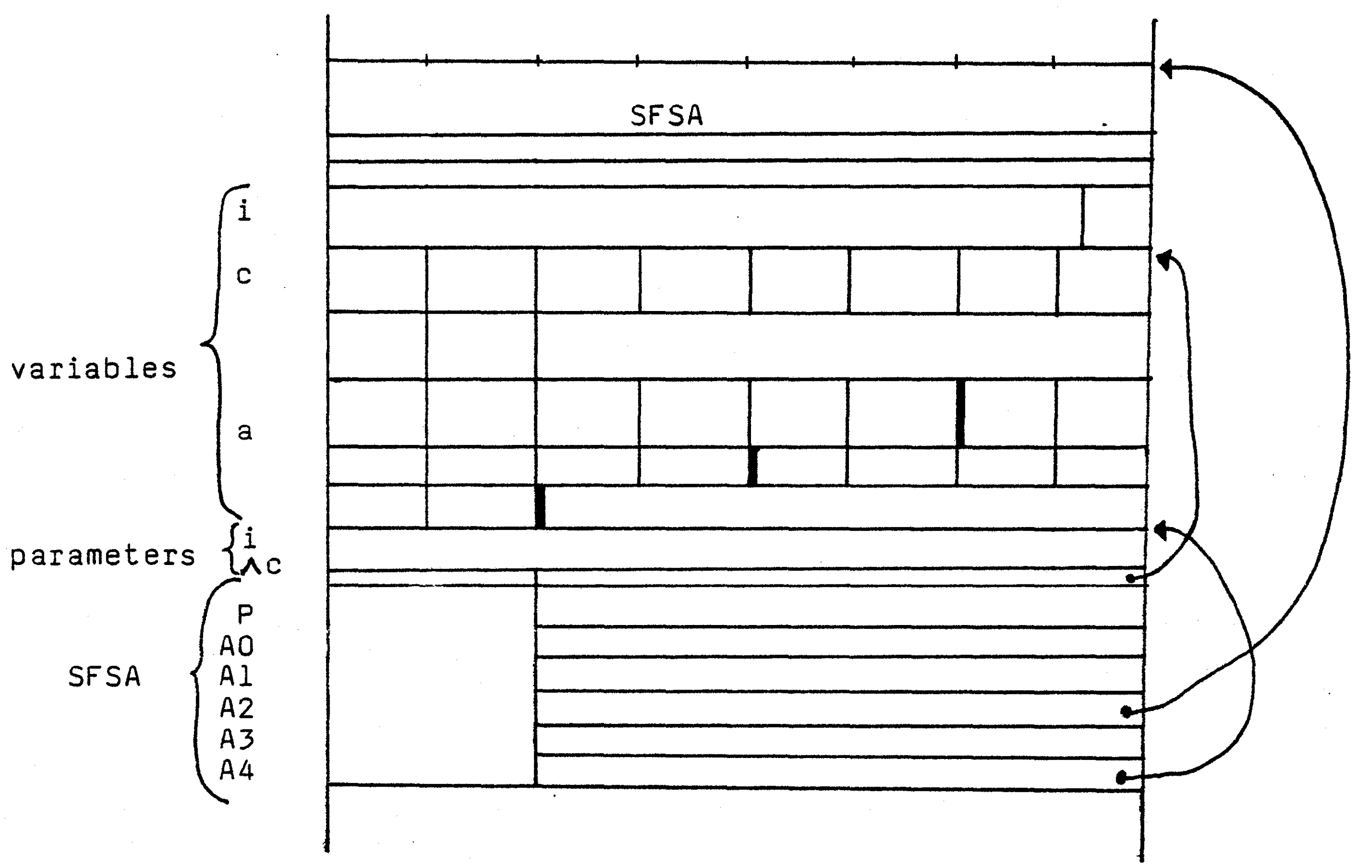
    - Each parameter starts a new word
    - VAR parameters are passed as pointers
    - The pointer to the parameters is in A4

STACK FRAME SAVE AREA

Save area	Word		
MINIMUM SAVE AREA	0	P REGISTER	
	1	VMID	A0 REGISTER (DYNAMIC SPACE POINTER)
	2	FRAME DESCRIPTION	A1 REGISTER (CURRENT STACK FRAME POINTER)
	3	USER MASK	A2 REGISTER (PREVIOUS SAVE AREA POINTER)
MAXIMUM SAVE AREA	4		A3 REGISTER (BINDING SECTION POINTER)
	5	USER CONDITION	A4 REGISTER (ARGUMENT POINTER)
	6	MONITOR CONDITION	A5 REGISTER
	7		A6 REGISTER
	8		A7 REGISTER
			• • •
	16	00 → 15	AF REGISTER
	17	XO REGISTER	
		• • •	
	32	XF REGISTER	
		00 → 63	

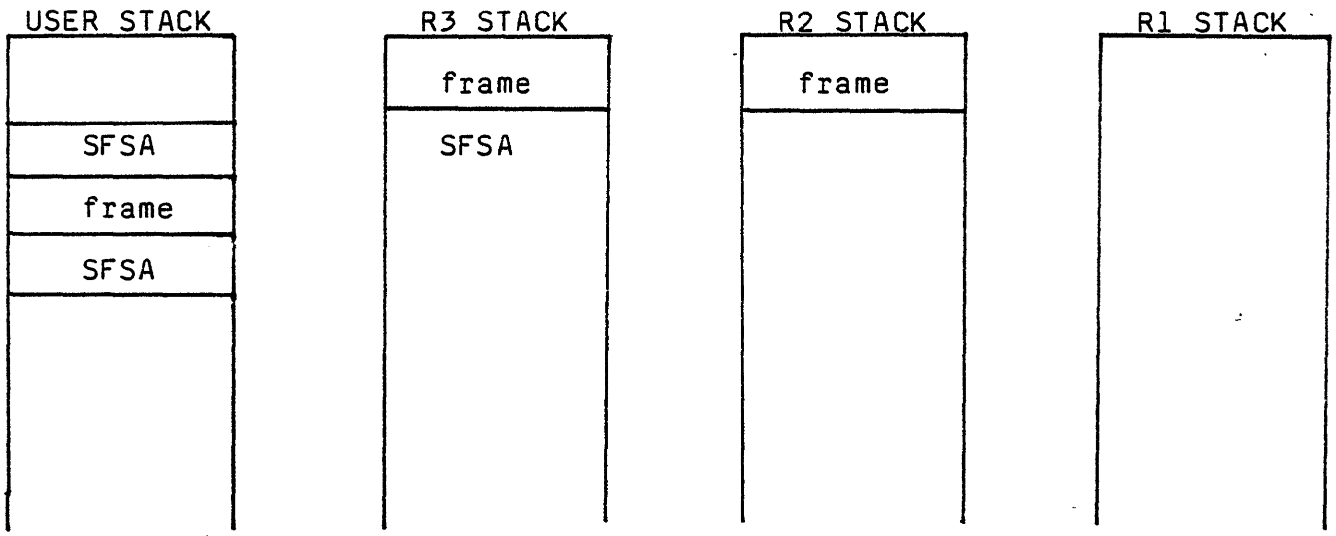
DATA MAPPING
EXAMPLE

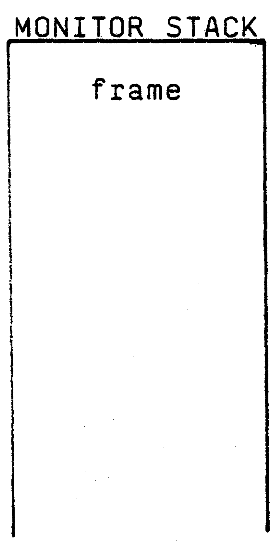
```
MODULE x;
   PROCEDURE ABC,
     VAR
       i: 1..10,
       c: string(10),
       a: array  1..3  of string(6),
                      .
                      .
                      .
     XYZ(i,c);
                      .
                      .
                      .
   PROCEND ABC;

   PROCEDURE XYZ (i:1..10; VAR s:string(10));
                      .
                      .
                      .
```

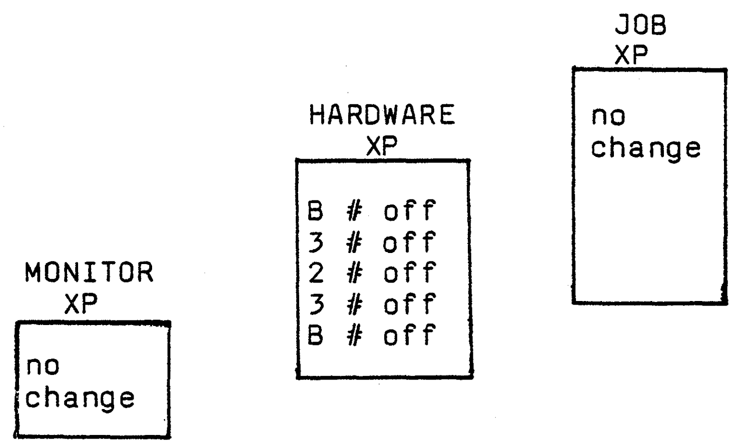


TRANSFER OF CONTROL

AMP&OPEN







1. User makes a request using program interface e.g. AMP$OPEN.
2. AMP$OPEN checks parameters and calls BAP$OPEN
3. BAP$OPEN creates task tables and calls FMP$OPEN to update job files.
4. FMP$OPEN returns
5. BAP$OPEN returns
6. AMP$OPEN returns

EXCHANGE PACKAGE

Word No.	00–07	08–15	16–63
0	P		
1	VMID*	UVMID**	A0
2	Flags	Traps Enables	A1
3	User Mask		A2
4	Monitor Mask		A3
5	User Condition		A4
6	Monitor Condition		A5
7	Kypt Class	LPID*	A6
8	Keypoint Mask		A7
9	Keypoint Code		A8
10			A9
11	Process Int. Timer		AA
12			AB
13	Base Constant		AC
14			AD
15	Model Dependent Flags		AE
16	Segment Table Length		AF
17	X0		
≈			
32	XF		
33	Model Dependent Word		
34	Segment Table Address		Untranslatable Pointer
35			Trap Pointer
36	Debug Index	Debug Mask	Debug List Pointer
37	Largest Ring Number		Top of Stack Ring No. 1
≈			
51			Top of Stack Ring No. 15

00 07 08 15 16 63

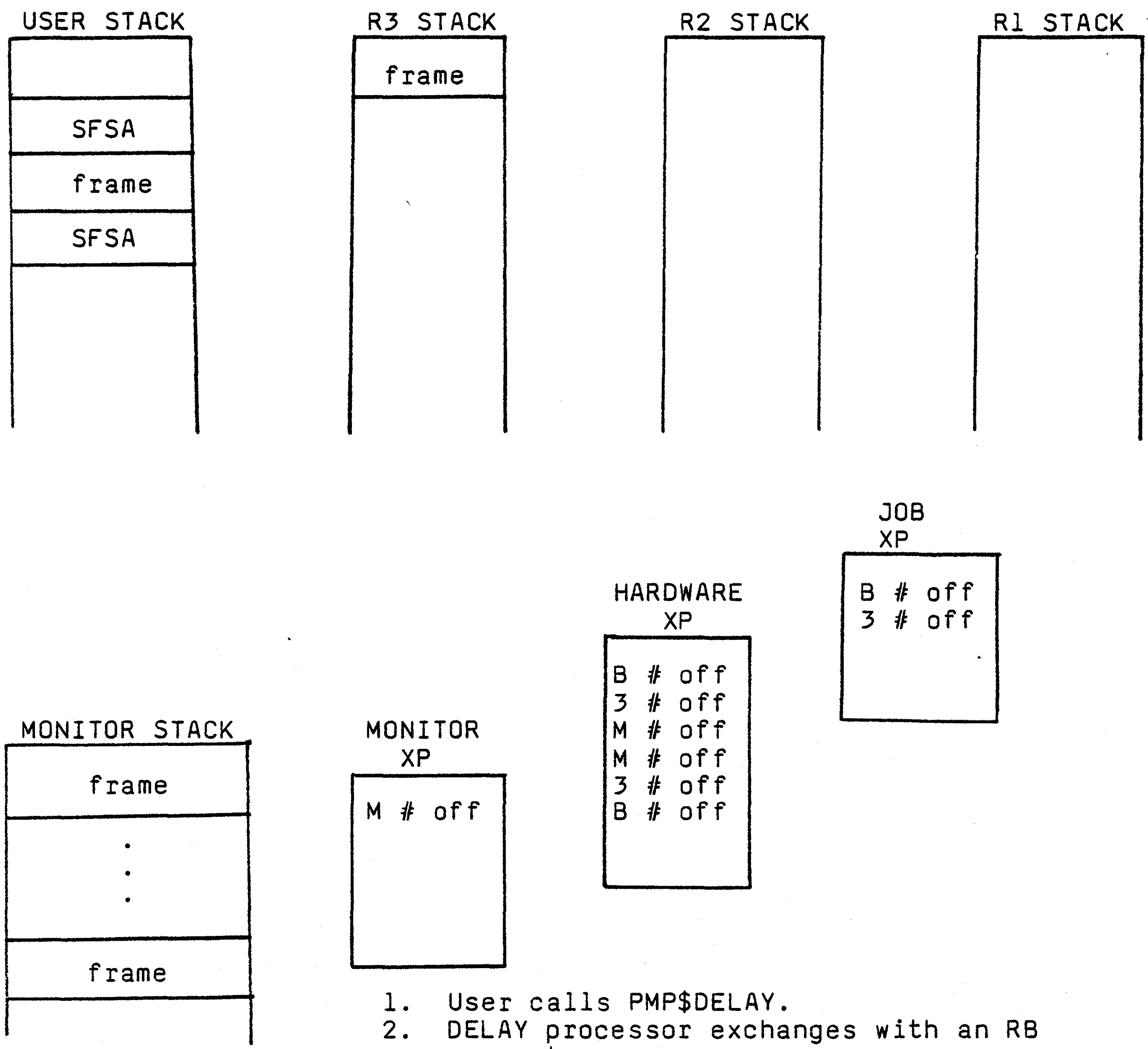
* Virtual Machine Identifier

** Untranslatable Virtual Machine Identifier

*** Last Processor Identification

TRANSFER OF CONTROL

DMP&DELAY



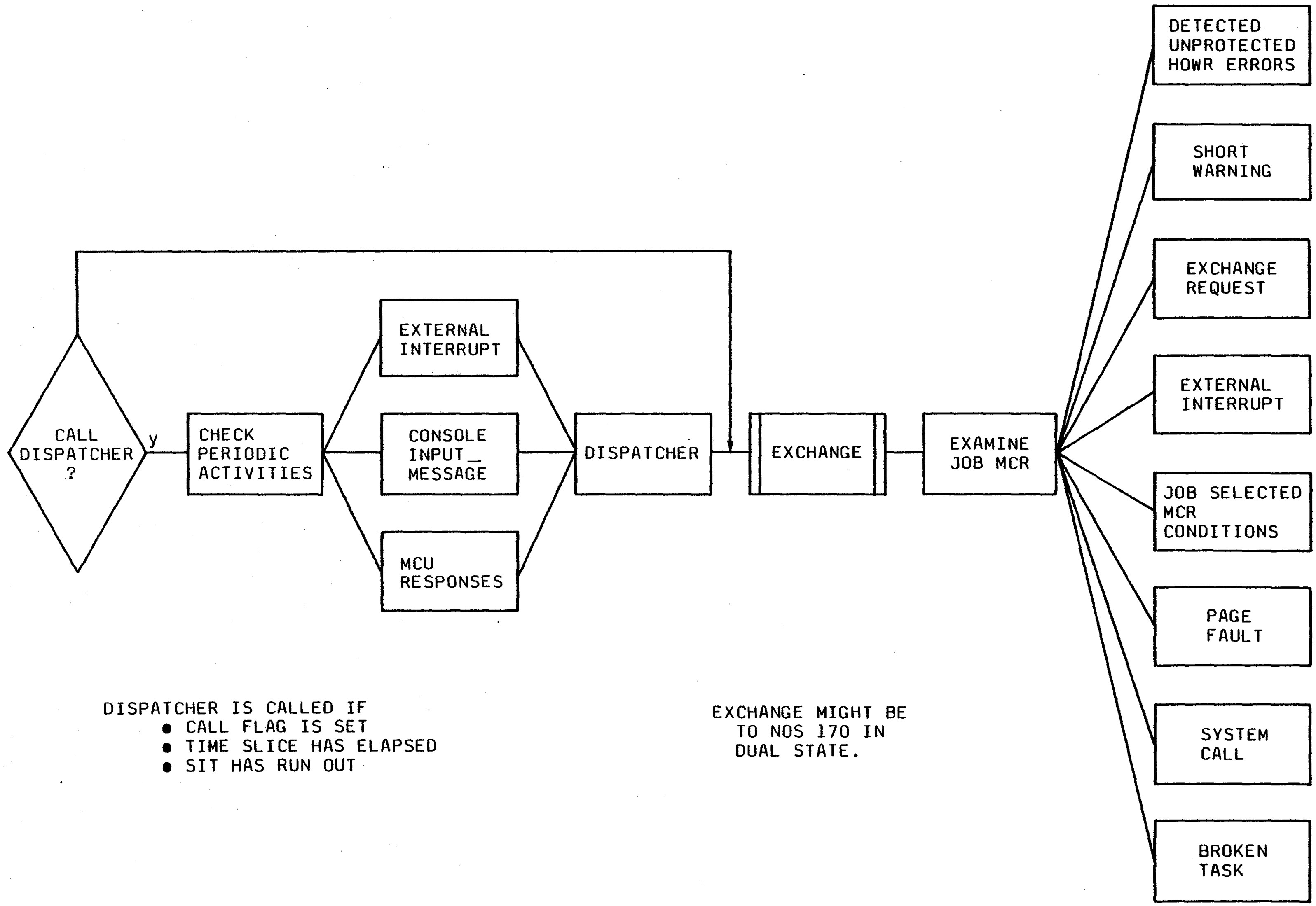
1. User calls PMP$DELAY.
2. DELAY processor exchanges with an RB request.
3. Monitor Interrupt processor delays the task.

.
.
.

4. Monitor returns to the task.
5. Delay processor returns.

Time	Hardware XP	R11 Stack	R3 Stack	R2 Stack	R1 Stack	Monitor Stack		Task XP Area	Monitor XP Event	Event Program Interface Request
T_0	B# n	B.frame$_1$	empty	empty	empty	M.frame$_1$		Program Starter	MIP	
T_1	3# n	SFSA	3.frame$_1$							P.I. request eg AMP+ RETURN
T_2	1# n		SFSA		1.frame					I.I. request to update system file tables (R1)
T_3	3# n		3.frame$_1$							R1 returns
T_4	B# n	B.frame$_1$								R3 returns
T_5										Page Fault
T_6	M# n					M.frame$_1$		Interrupt user B# n		Page fault
T_7	"					SFSA M.frame$_2$				Call
T_8	"					M.frame$_1$				Return From Page Manager
T_9	"					SFSA M.frame$_3$				Call Dispatcher
T_{10}	"					M.frame$_1$				Return from Dispatcher
T_{11}	?									Exchange to new task
T_{12}										⋮
T_{13}	B# n					M.frame$_1$				Exchange to this task

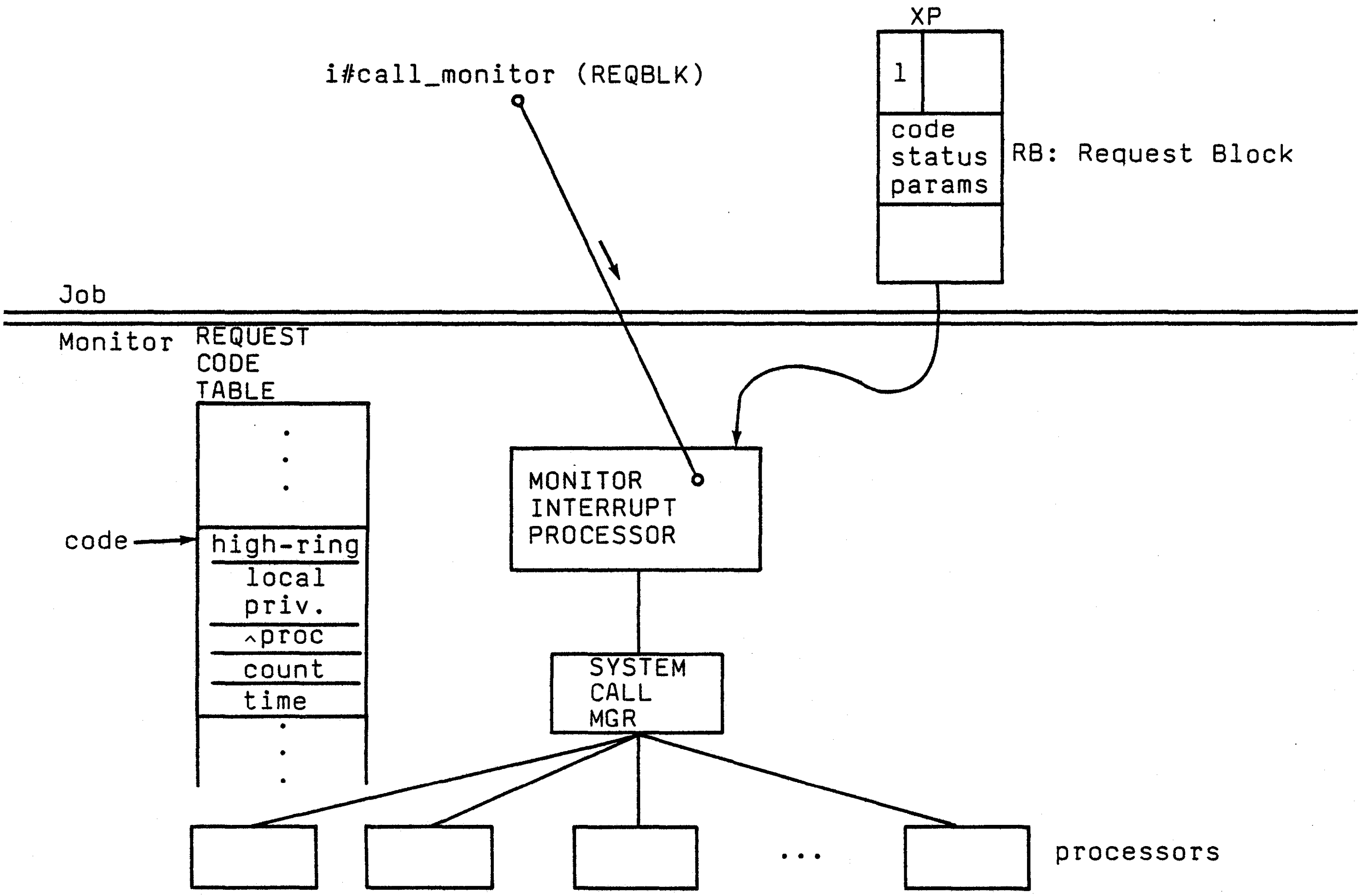
MONITOR INTERRUPT PROCESSOR



DISPATCHER IS CALLED IF
- CALL FLAG IS SET
- TIME SLICE HAS ELAPSED
- SIT HAS RUN OUT

EXCHANGE MIGHT BE TO NOS 170 IN DUAL STATE.

SYSTEM CALL PROCESSING



PROCEDURE [XDCL] xxp$yyyy (VAR rb:request-block)

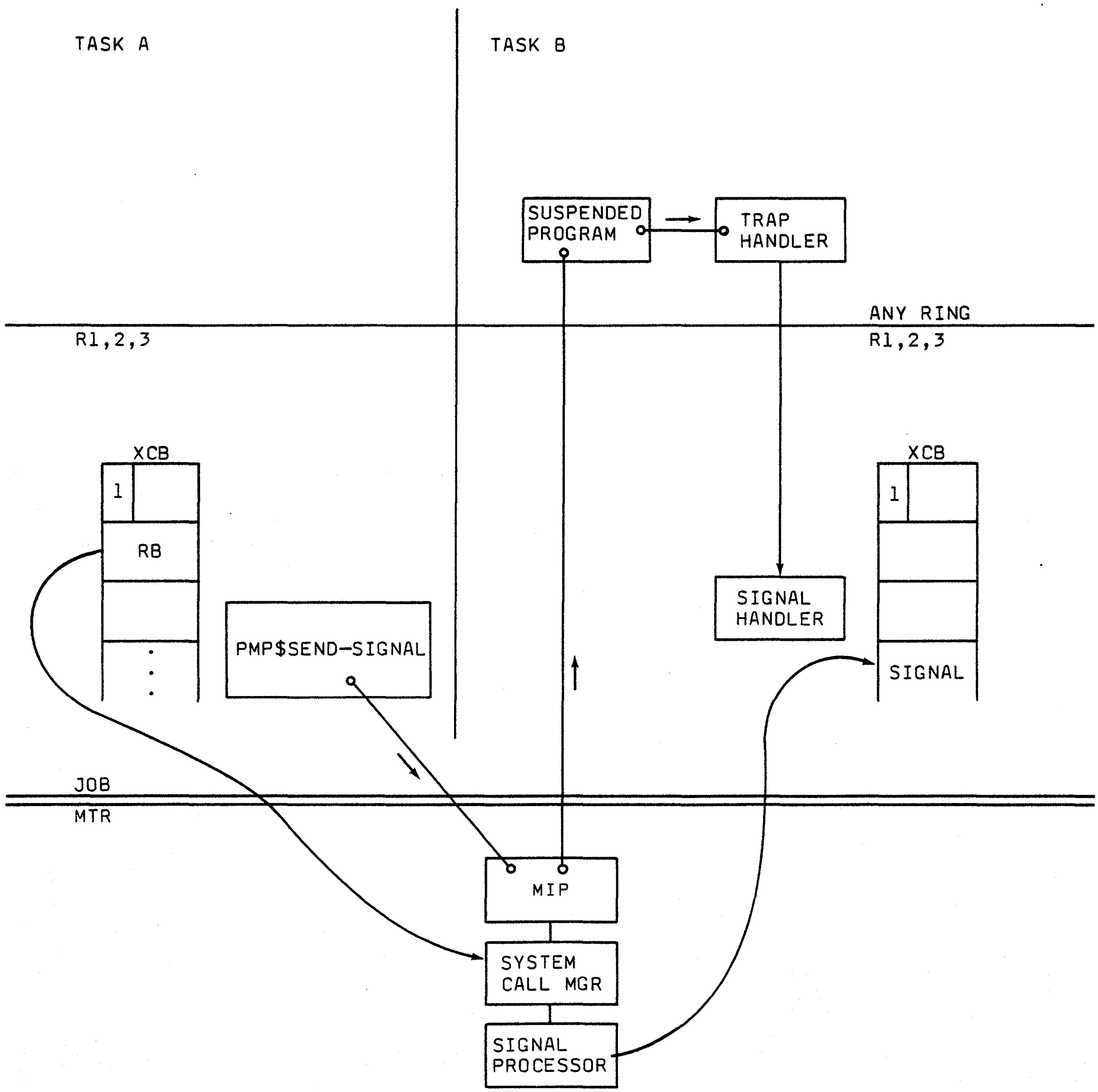
REQUEST BLOCK

MEMORY MANAGER REQUEST DEFINITIONS
1 MMT$RB_ADVISE
2 MMT$RB_ASSIGN_FLAWED_MEMORY
3 MMT$RB_ASSIGN_REAL_PAGE
4 MMT$RB_FLAW_PAGE
5 MMT$RB_FREE_FLUSH
6 MMT$RB_UNFLAW_PAGE
TASK MANAGER REQUEST DEFINITIONS
1 TMT$RB_INITIATE_JOB
2 TMT$RB_INITIATE_TASK
3 TMT$RB_CYCLE
4 TMT$RB_DELAY
5 TMT$RB_EXIT_JOB
6 TMT$RB_EXIT_TASK
7 TMT$RB_SEND_SIGNAL
8 TMT$RB_WAIT_SIGNAL
9 TMT$RB_CHANGE_SEGMENT_TABLE

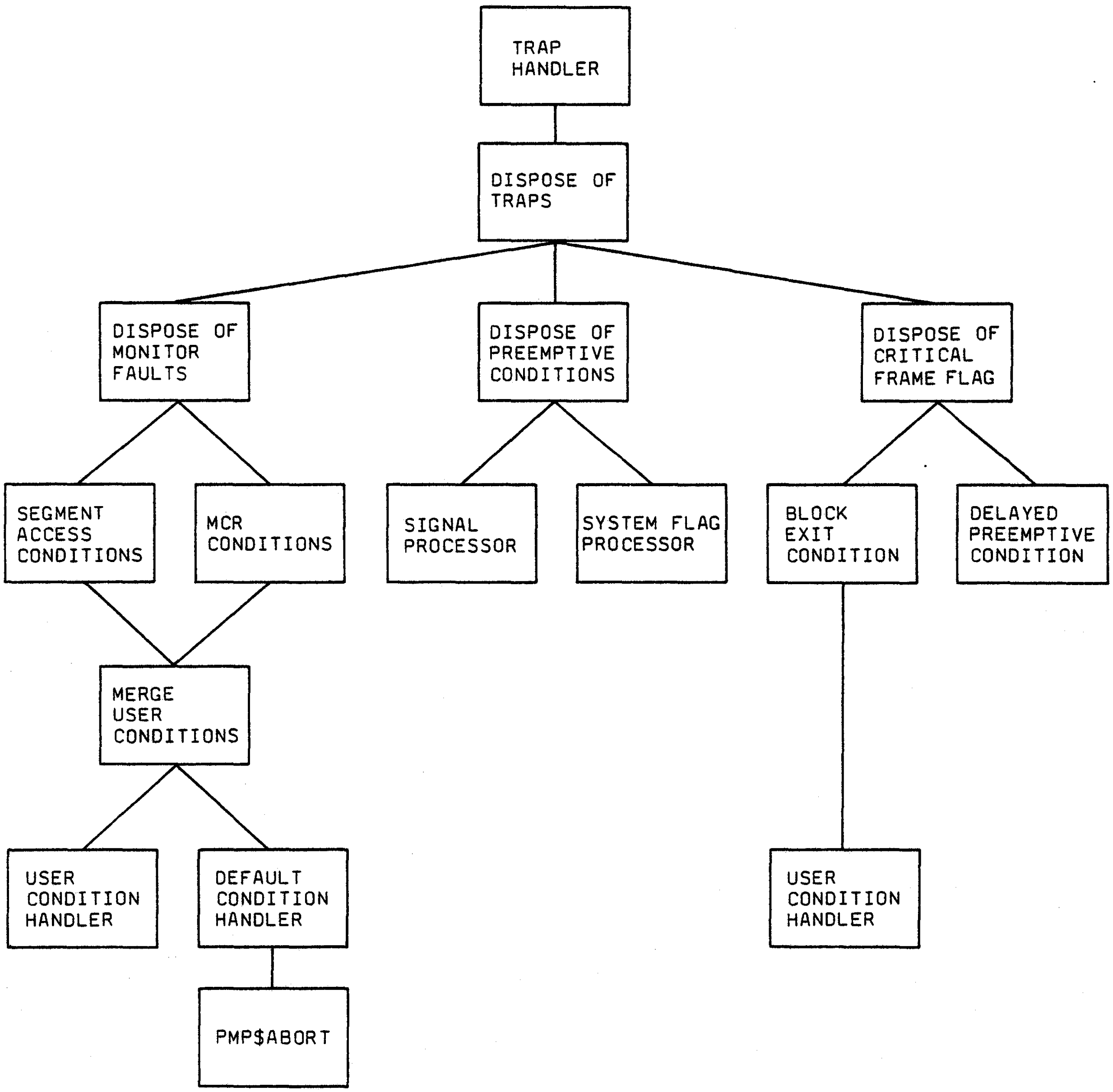
Note: see Internal Interface.

Time	Hardware XP	R11 Stack	R3 Stack	R2 Stack	R1 Stack	Monitor Stack		Task XP Area	Monitor XP Event	Event Access Violation
T_0	B# n_1	B.frame$_1$	empty	empty	empty	M.frame$_1$		Program starter	MIP	
T_1	M# n_1							B# n_1		Access Violation
T_2	"									Set free flag in UCR
T_3	B# n_1								MIP	Exchange
T_4	B# n_2	SFSA B.frame$_2$								Trap
T_5	3# n_1	SFSA	3.frame$_1$							Call default Condition Handler
T_6	3# n_2		SFSA 3.frame$_2$							Call PMP$ABORT
T_7	3# n_2									Pop R11 stack
T_8	M# n_1							3# n_2		Request Task Termination
T_9	M# n_2					SFSA M.frame$_2$				Call System Call processor
T_{10}						M.frame$_1$				Return after terminating task
T_{11}										Call dispatcher to pick new task
T_{12}										
T_{13}										

SIGNAL HANDLING



TRAP HANDLER



- Trap Handler runs at the ring of the interrupted program.
- Dispose-of-traps runs in ring 1.
- Dispose-of-traps checks the reason for entry in this order:
  1. Monitor Faults
     Segment Access Conditions, MCR conditions and User conditions are merged together. If a user condition handler exists it will be found in the stack. The default condition handler will abort the task in all cases.
  2. Preemptive Conditions
     All signals and flags will be processed if the free flag is set. If the ring of execution is lower than the recognition ring, the critical frame flag will be set in the first stack frame above the recognition ring. In all other cases the signal or flag handler will be called.
  3. Critical Frame Flag
     If the critical frame flag indicates a delayed signal or flag it will be resolved by dispose-of-preemptive-conditions. If the user has established a block exit handler, the handler will be found in the stack frame and called.
- A Daley diagram of these modules is included in the chapter on program management.

SEND SIGNAL

```
PMP$SEND_SIGNAL(recipient,signal,status)
```

SIGNAL HANDLER

```
ppP$HANDLE_SIGNAL_xxx(originator,signal)
```

DEFINE HANDLER (for test only)

```
PMP$DEFINE_SIGNAL_HANDLER(id,handler,recog_ring,status)
```

SIGNALS

Memory Link	MLP$ handle_signal interprets 'sub_signals' and calls a handler.
Interactive	IFP$hande_signal passes info between the interactive exec., job monitor, and user tasks.
Callend	PMP$child_terminator_handler.
Scheduler	JMP$_handle_gfm_ia_signal processes the signal from QF manager that interactive job has been routed.

SET FLAG

PMP$SET_SYSTEM_FLAG(flag_id,recipient,status)

FLAG HANDLER

ppP$HANDLE_FLAG_xxx(flag_id)

DEFINE HANDLER (for test only)

PMP$DEFINE_SYSTEM_FLAG_HANDLER(id,handler,recog_ring,st)

FLAGS

Statistics	AVP$monitor_statistics_handler
Terminate	PMP$terminate_flag_handler
Drop	JMP$handle_drop_job_flag
Linked Signals	TMP$dispose_mainframe_signals This flag indicates that a signal occurred while the task was swapped.

MONITOR FAULTS

FAULT HANDLER

ppP$HANDLE_FAULT_xxx(fault,save_area)

DEFINE HANDLER (for test only)

PMP$DEFINE_MONITOR_FAULT(id,handler,status)

FAULTS

Instruction Specification Error
Address Specification Error
Access Violation
Environment Specification Error
Outward Call/Inward Return

SEGMENT ACCESS CONDITIONS

Read beyond EOI
Write beyond msl
Segment access error
Key lock violation
Ring violation
I/O read error

MONITOR CONDITION REGISTER

BIT NUMBER AND DEFINITION			TRAPS ENABLED		TRAPS DISABLED		
			TRAP ENABLE F/F SET AND TRAP ENABLE DELAY F/F CLEAR AND MASK BIT SET		TRAP ENABLE F/F CLEAR OR TRAP ENABLE DELAY F/F SET AND MASK BIT SET		MASK BIT CLEAR
			JOB MODE	MONITOR MODE	JOB MODE	MONITOR MODE	JOB OR MONITOR MODE
0	Processor Detected Malfunction	Mon	EXCH	TRAP	EXCH	HALT	HALT
1	Memory Detected Malfunction	Mon	EXCH	TRAP	EXCH	HALT	HALT
2	Power Warning	Sys	EXCH	TRAP	EXCH	STACK	STACK
3	· Instruction Specification Error	Mon	EXCH	TRAP	EXCH	HALT	HALT
4	· Address Specification Error	Mon	EXCH	TRAP	EXCH	HALT	HALT
5	Exchange Request	Sys	EXCH	TRAP	EXCH	STACK	STACK
6	· Access Violation	Mon	EXCH	TRAP	EXCH	HALT	HALT
7	· Environment Specification Error	Mon	EXCH	TRAP	EXCH	HALT	HALT
8	External Interrupt	Sys	EXCH	TRAP	EXCH	STACK	STACK
9	Page Table Search Without Find	Mon	EXCH	TRAP	EXCH	HALT	HALT
10	System Call	Status - This bit is a flag only and does not cause any hardware action.					
11	System Interval Timer	Sys	EXCH	TRAP	EXCH	STACK	STACK
12	Invalid Segment	Mon	EXCH	TRAP	EXCH	HALT	HALT
13	· Outward Call/Inward Return	Mon	EXCH	TRAP	EXCH	HALT	HALT
14	Soft Error Log	Sys	EXCH	TRAP	EXCH	STACK	STACK
15	Trap Exception	Status - This bit is a flag only and does not cause any hardware action.					

USER CONDITION REGISTER

BIT NUMBER AND DEFINITION			TRAPS ENABLED		TRAPS DISABLED		
			TRAP ENABLE F/F SET AND TRAP ENABLE DELAY F/F CLEAR AND MASK BIT SET		TRAP ENABLE F/F CLEAR OR TRAP ENABLE DELAY F/F SET AND MASK BIT SET		MASK BIT CLEAR
			JOB MODE	MONITOR MODE	JOB MODE	MONITOR MODE	JOB OR MONITOR MODE
0	Privileged Instruction Fault	Mon	TRAP	TRAP	EXCH	HALT	These mask bits are permanently set.
1	Unimplemented Instruction	Mon	TRAP	TRAP	EXCH	HALT	
2	Free Flag	User	TRAP	TRAP	STACK	STACK	
3	Process Interval Timer	User	TRAP	TRAP	STACK	STACK	
4	Inter-ring Pop	Mon	TRAP	TRAP	EXCH	HALT	
5	Critical Frame Flag	Mon	TRAP	TRAP	EXCH	HALT	
6	Keypoint	User	TRAP	TRAP	STACK	STACK	
7	Divide Fault	User	TRAP	TRAP	STACK	STACK	STACK
8	Debug	User	TRAP	TRAP	Debug bit will not set when traps disabled.		
9	Arithmetic Overflow	User	TRAP	TRAP	STACK	STACK	STACK
10	Exponent Overflow	User	TRAP	TRAP	STACK	STACK	STACK
11	Exponent Underflow	User	TRAP	TRAP	STACK	STACK	STACK
12	F. P. Loss of Significance	User	TRAP	TRAP	STACK	STACK	STACK
13	F. P. Indefinite	User	TRAP	TRAP	STACK	STACK	STACK
14	Arithmetic Loss of Significance	User	TRAP	TRAP	STACK	STACK	STACK
15	Invalid BDP Data	User	TRAP	TRAP	STACK	STACK	STACK

LESSON 7
EXTERNAL COMMUNICATION

LESSON PREVIEW

- MEMORY LINK (DUAL STATE)
- INTERACTIVE FACILITY
- OPERATOR FACILITY
- STATISTICS FACILITY
- MESSAGE GENERATOR
- KEYPOINTS
- LOGS

REFERENCES

PROGRAM INTERFACE

OBJECTIVES

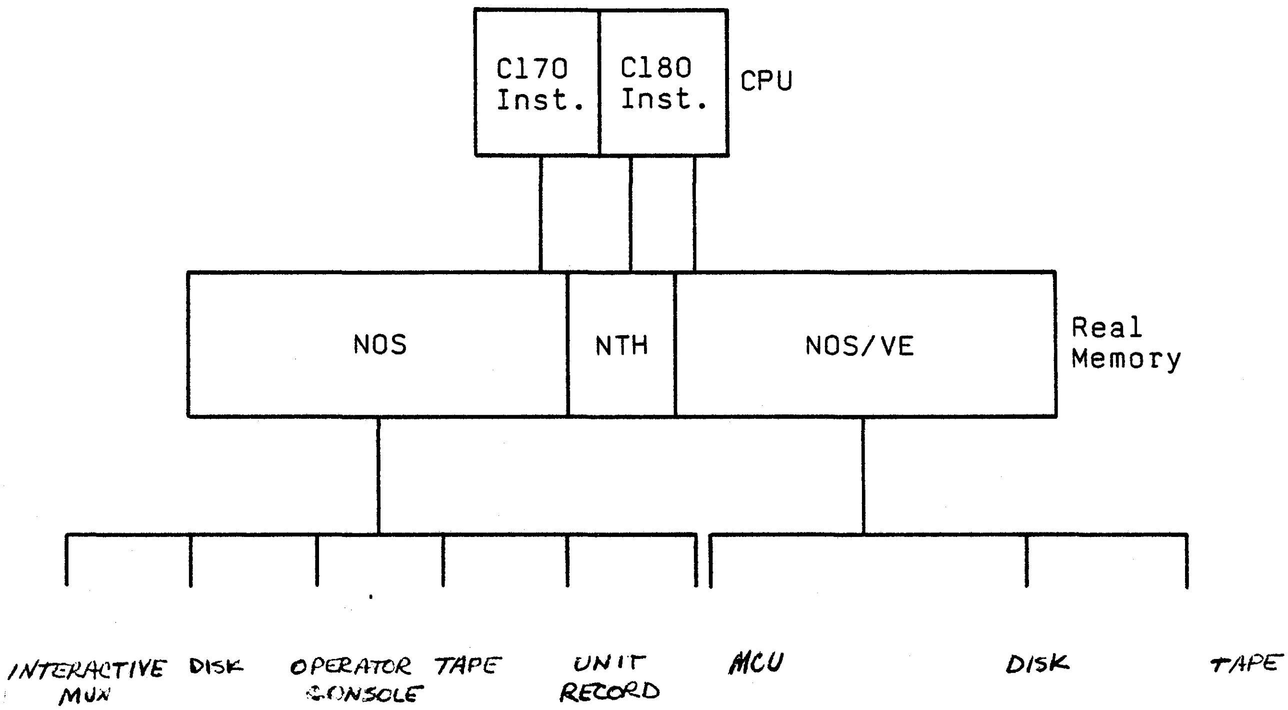
After completing this lesson the student should be able to--

- DESCRIBE THE MEMORY LINK INTERFACE
- EXPLAIN HOW THE MEMORY LINK IS USED BY THE INTERACTIVE FACILITY, THE OPERATOR FACILITY AND THE REMOTE HOST FACILITY
- OUTLINE HOW THE NOS DEPENDENT CAPABILITIES WILL BE CHANGED TO BE INDEPENDENT OF NOS
- EMIT, ENABLE, AND ESTABLISH SYSTEM STATISTICS
- ADD MESSAGE TEMPLATES TO THE TEMPLATE TABLE
- GENERATE KEYPOINT DATA
- USE THE LOG MANAGER INTERFACES TO MANIPULATE LOG FILES

EXERCISES

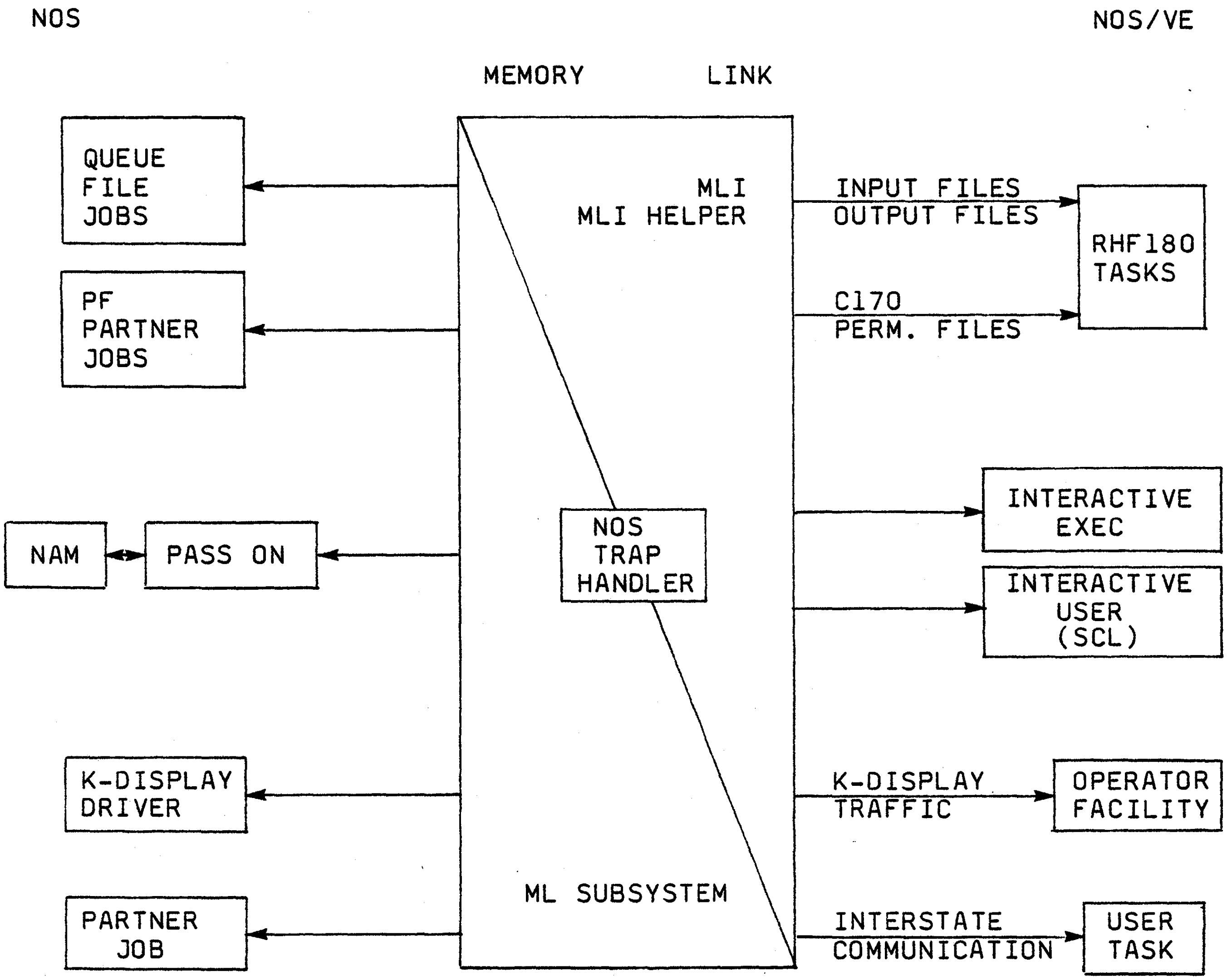
1. ADD A MESSAGE TEMPLATE TO THE SYSTEM. USE IT.
2. ESTABLISH, ENABLE, AND EMIT A NEW STATISTICS
3. INTERPRET KEYPOINT OUTPUT.

DUAL STATE CONFIGURATION



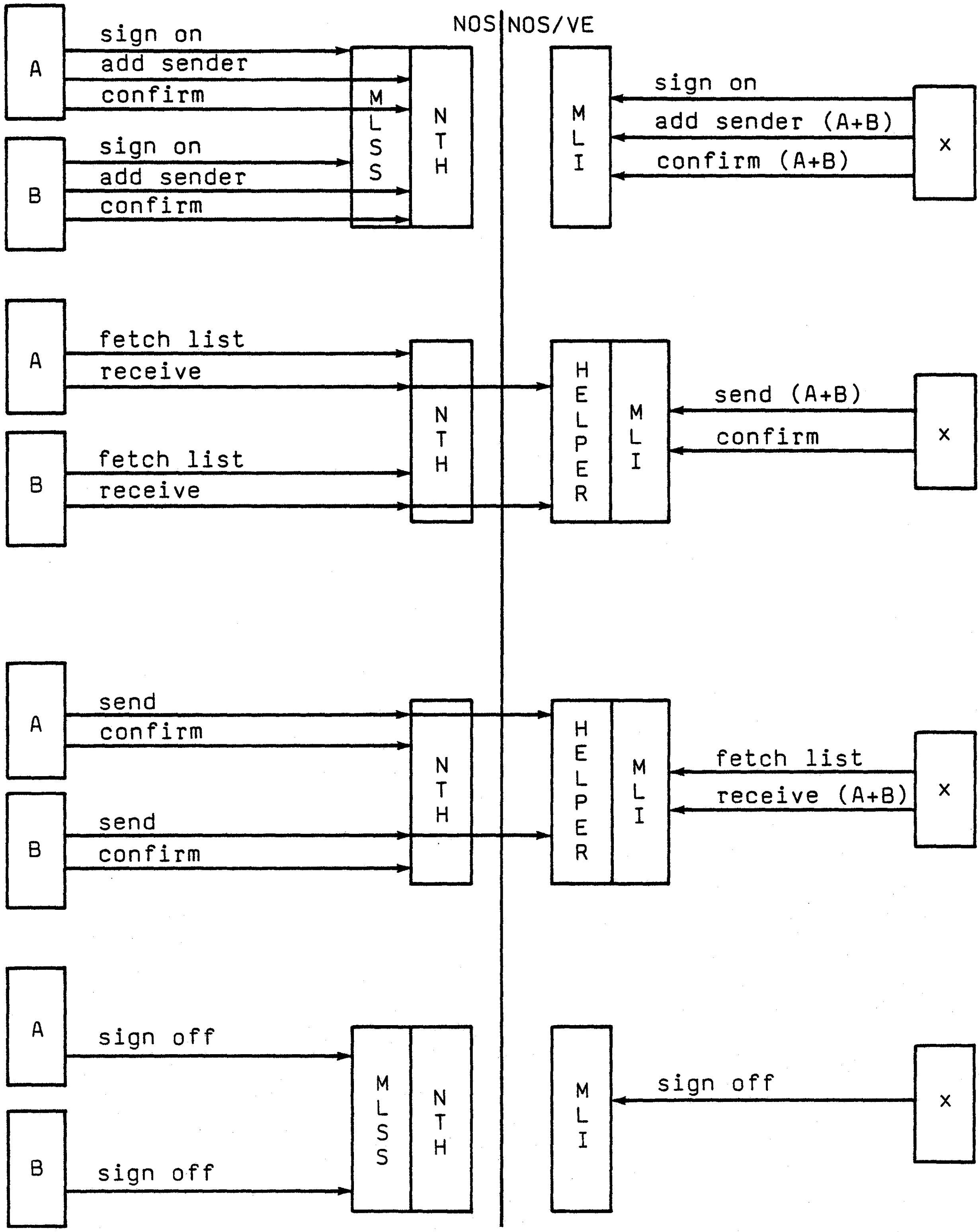
- The system resources are partitioned between CYBER 170 and CYBER 180 logical machines
- CPU is partitioned using the VMID field in the exchange package. Determines how the CPU will
  - Fetch and interpret instructions
  - Interpret the register file
  - Interpret interrupts
- CPU access to central memory
  - CYBER 17Ø addresses map into real memory addresses Ø-N
  - CYBER 18Ø addresses map into (N+1) - (memory size-1)
- PPU access to central memory
  - PPUs are assigned to either 17Ø system or NOS/VE
  - IOU bounds register limits write access to CM
- Channels are software partitioned to access only CYBER 17Ø or CYBER 18Ø peripheral devices (except maintenance channel).

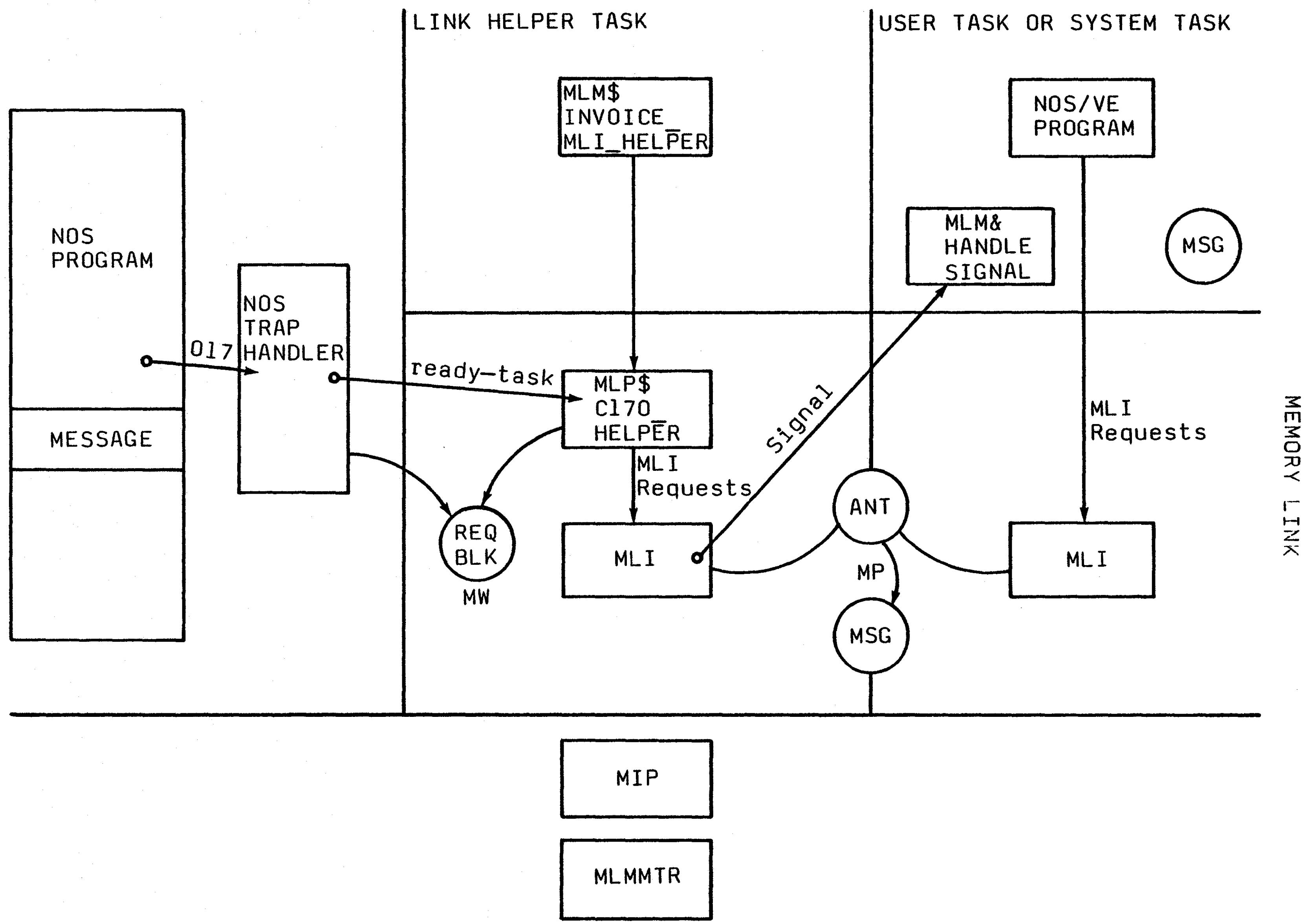
LINK FACILITY



```
MLP$SIGN_ON
        (name,max_msgs,unique_name,status)

MLP$SIGN_OFF
        (name,status)

MLP$ADD_SENDER
        (name,sender_name,status)

MLP$DELETE_SENDER
        (name,sender_name,status)

MLP$CONFIRM_SEND
        (name,destination_name,status)

MLP$SEND_MESSAGE
        (name,info,signal,message_area,message_length,destination
        name,status)

MLP$FETCH_RECEIVE_LIST
        (name,sender_name,list,count,status)

MLP$RECEIVE_MESSAGE
        (name,info,signal,message_area,msg_length,msg_area
        length,receive_index,sender_name,status)
```

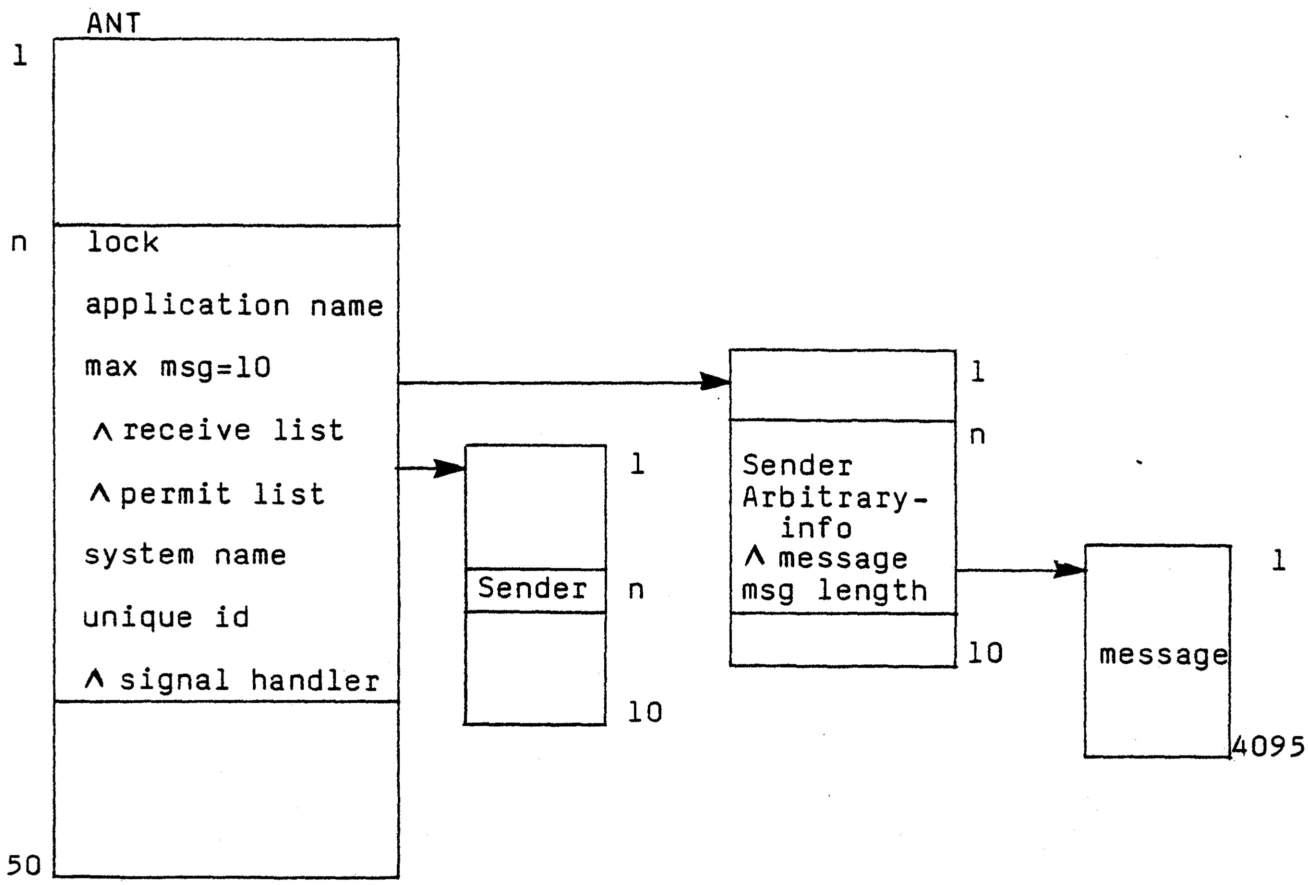
MLI PROTOCOL




LINK HELPER TASK
USER TASK OR SYSTEM TASK
MLM$ INVOICE_ MLI_HELPER
NOS/VE PROGRAM
NOS PROGRAM
MLM& HANDLE SIGNAL
MSG
NOS TRAP HANDLER
017
ready-task
MLP$ C170_ HELPER
Signal
MLI Requests
MEMORY LINK
MESSAGE
MLI Requests
ANT
REQ BLK
MW
MLI
MP
MLI
MSG
MIP
MLMMTR

- The NOS program uses CYBIL procedures or COMPASS macros to communicate with the NOS/VE job. These translate to 017 instruction which are trapped by the NOS trap handler (NOS T.H.).
- When an 017 instruction is executed, the NOS trap handler runs in NOS/VE instruction mode. It moves the message into a circular buffer. The entries are called request blocks. Trap handler issues a monitor request to ready the link_helper task.
- MLI helper transfers the message from the circular buffer to the mainframe pageable segment using MLI transfer requests. Some NOS/VE applications are signaled when there is a message received for that application.
- MLI manages the queue and services the requests issued by MLI helper, Interactive Facility, and so on.
- The NOS/VE program transfers the message into its buffer using MLI requests. Applications which use the facility are:

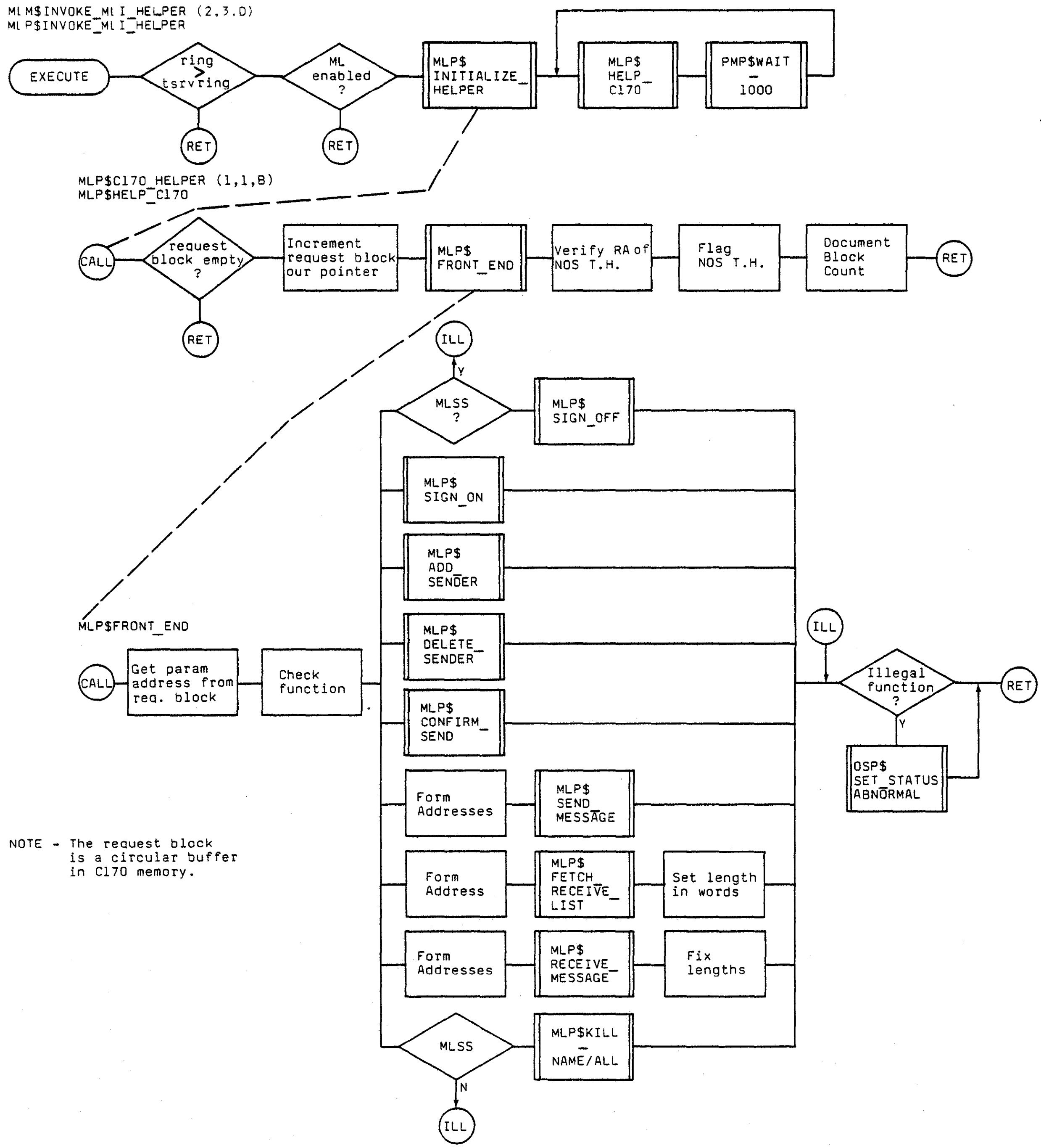
  RHF180
  Interactive Exec.
  Interactive Facility
  Operator Facilty
  Interstate Communication (users)



ANT=Application_name_table

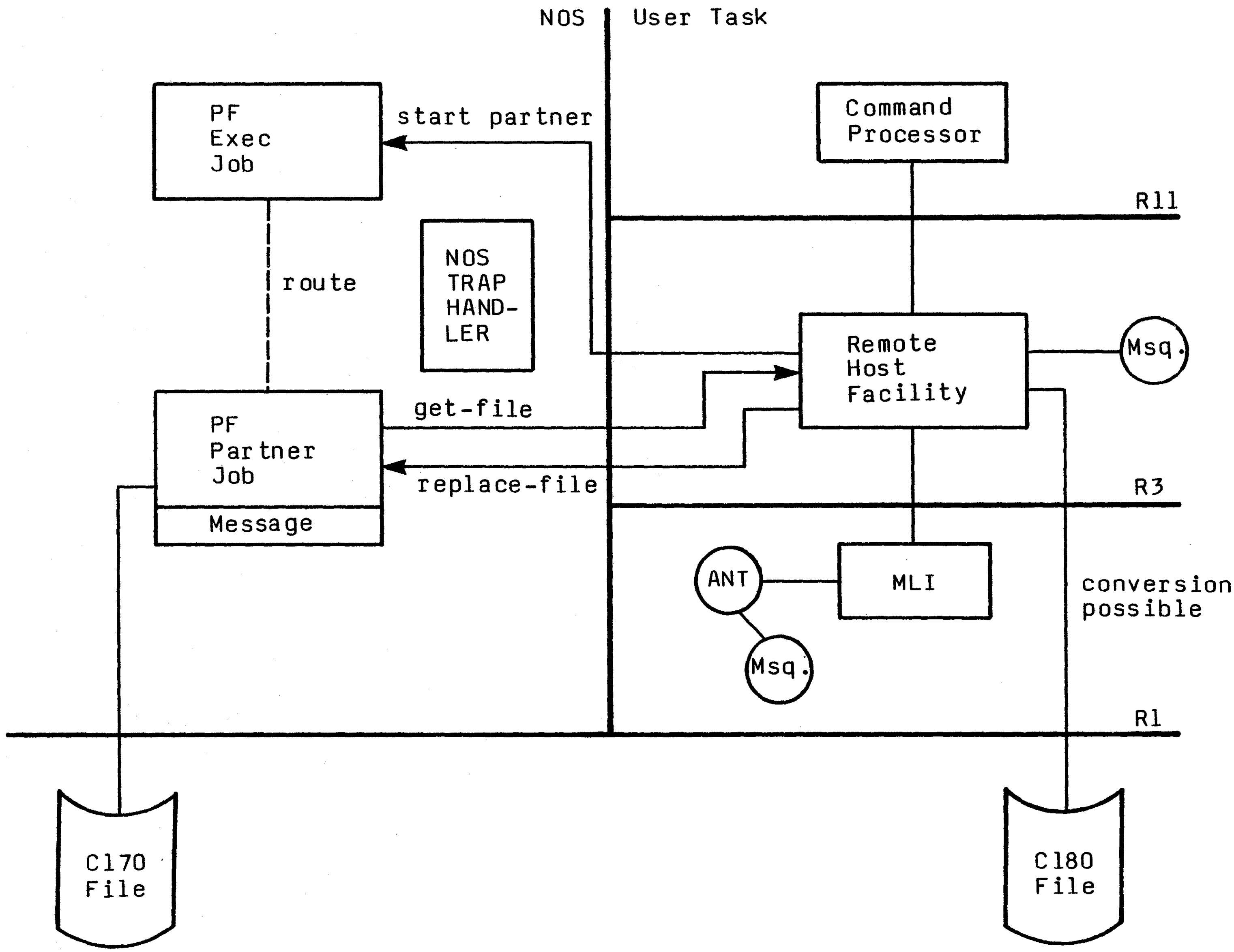
All tables are in mainframe pageable segment.

ML HELPER



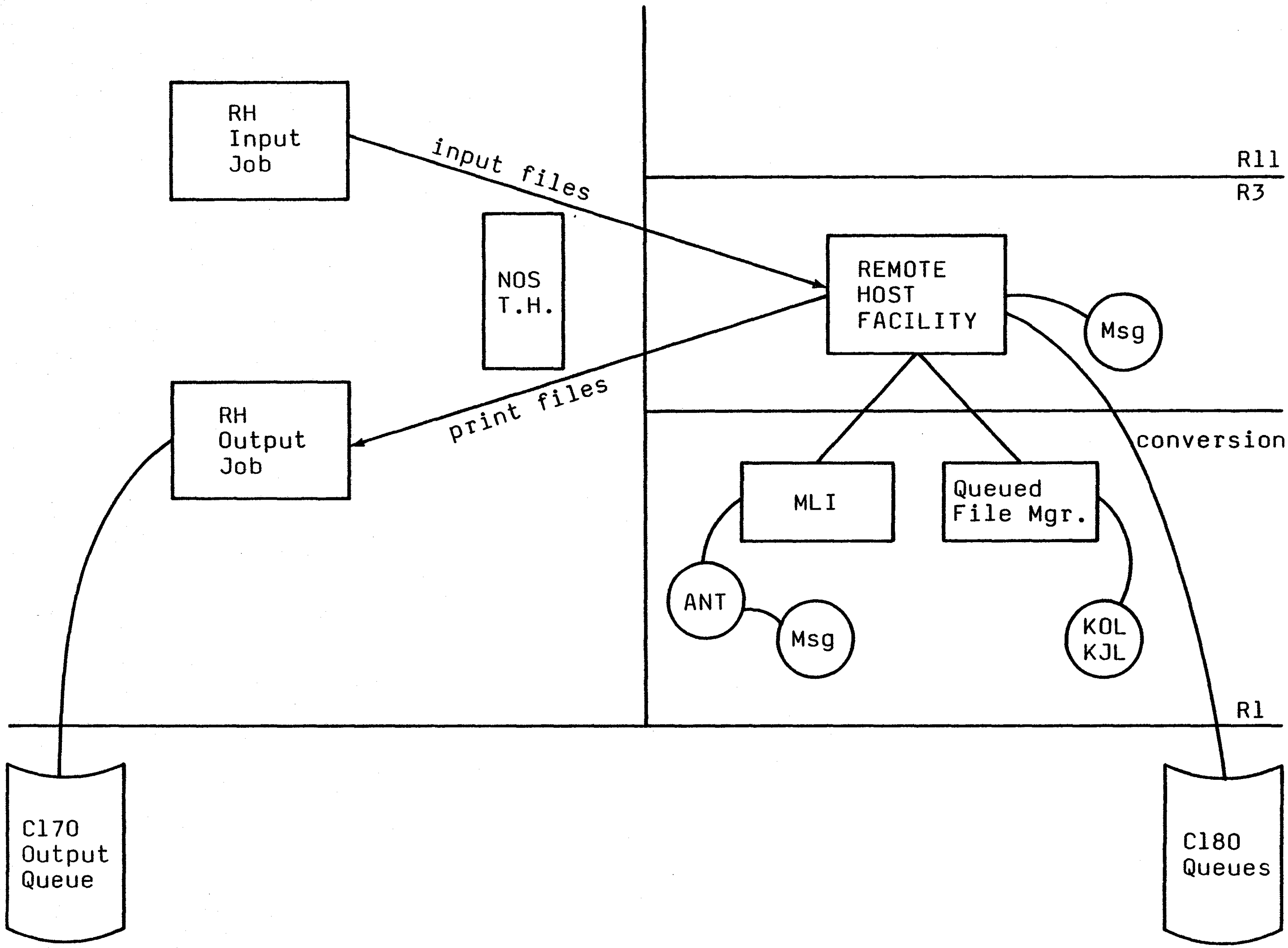
NOTE - The request block is a circular buffer in C170 memory.

RHF-PERM FILES

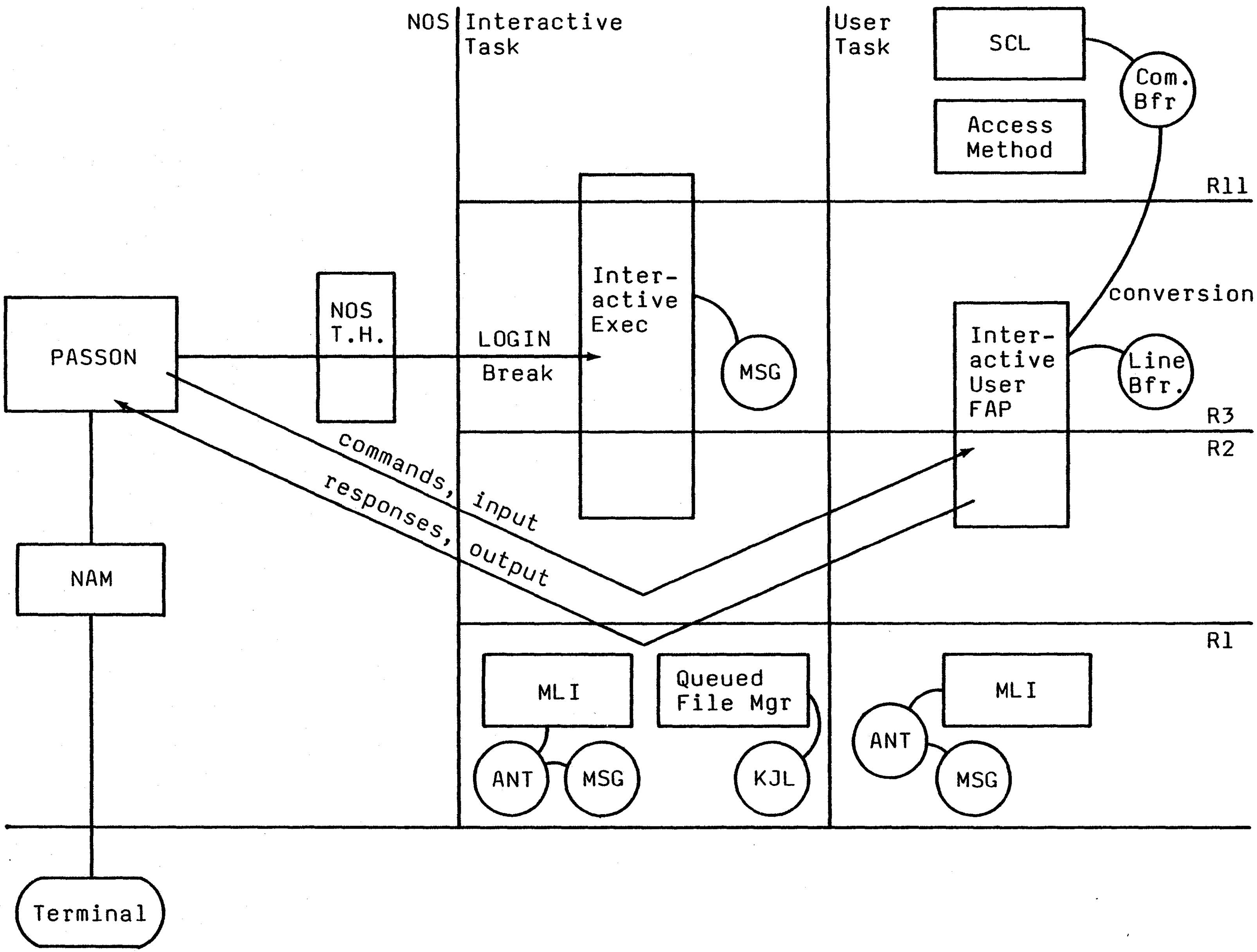


- Users must be validated for access to the remote host facilty
  - NOS/VE uses family name for mainframe ID.
  - Requests to access permanent files via the RHF include user validation.
- File size limitations will be associated with each linked family to restrict transfers via the RHF
- NOS/VE remote host facility job
  - Communicates with the linked system
  - Receives input jobs and sends output files
- Linked communication services
  - User interface for permanent file handling via the link (get, save, replace, purge, permit and catist commands)
- Linked file conversion
  - Link files are interchange format
  - Queue files and permanent files are converted before and after transfer

RHF-SPOOLED FILES



INTERACTIVE COMMUNICATION

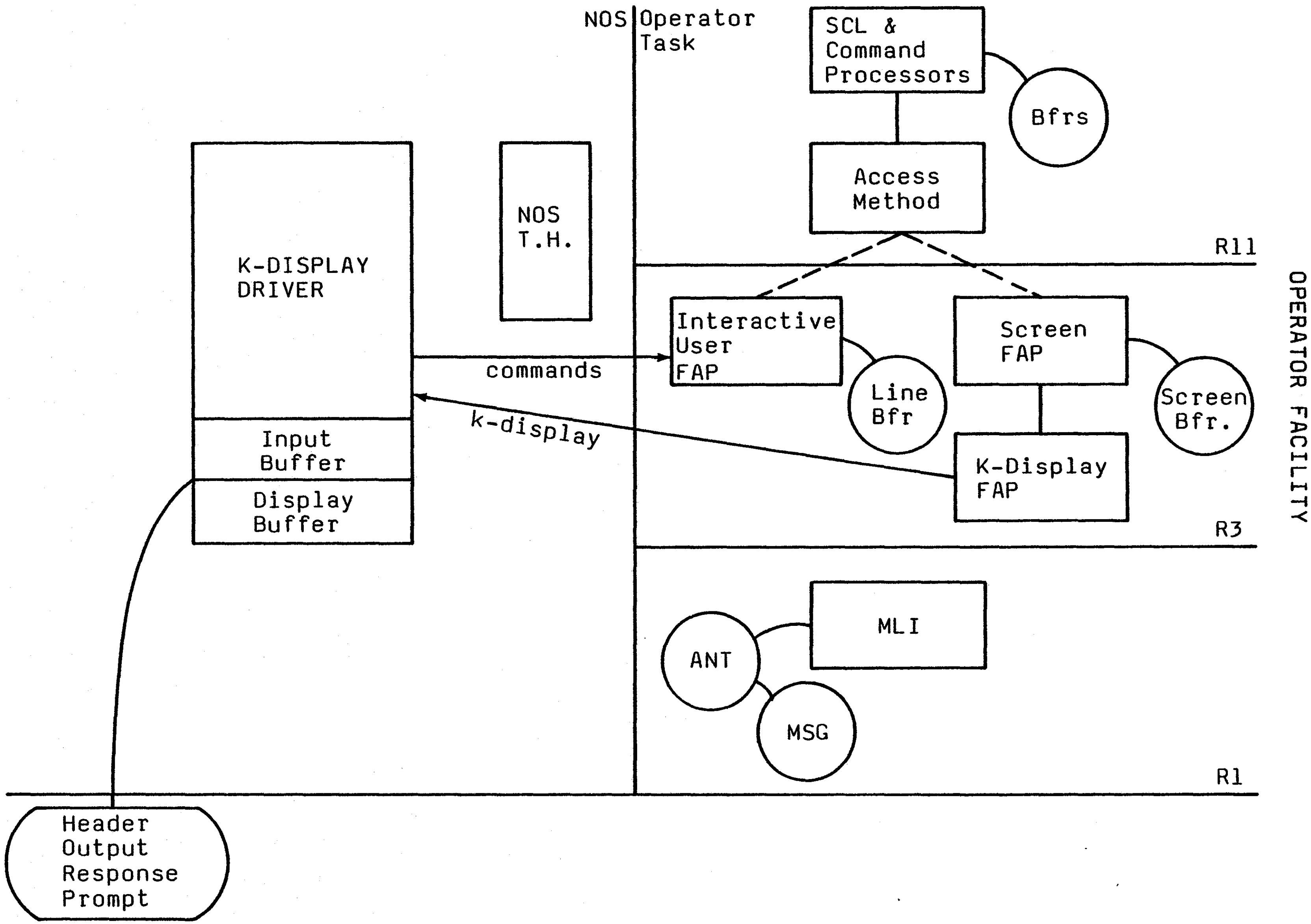


- NOS/VE interactive terminal access is performed through the Network Access Method (NAM). Its interactive facilities are a superset of those of NOS/170. The terminal user may:

  - Enter commands
  - Enter data to programs
  - Interrupt the execution of interactive jobs
  - Define terminal attributes
  - Receive command status messages
  - Receive program output data
  - Disconnect a terminal from a running interactive job thus freeing the terminal for other work
  - Recover an interactive job that was disconnected from its terminal

- NOS/VE treats terminal I/O as normal file I/O through Basic Access Methods (BAM). BAM allows the name of an I/O processing (FAP) to be substituted into the file attributes at open time. This is done for terminal I/O.

- NOS/VE treats NAM as an external interface. The basic handling of terminals will not change whether it is done through CYBER 170 NAM or through future versions of NAM on the CYBER 180 side.


NOS
Operator Task
SCL & Command Processors
Bfrs
Access Method
NOS T.H.
K-DISPLAY DRIVER
R11
Interactive User FAP
Screen FAP
commands
Line Bfr
Screen Bfr.
Input Buffer
k-display
K-Display FAP
OPERATOR FACILITY
Display Buffer
R3
MLI
ANT
MSG
R1
Header
Output
Response
Prompt

OPERATOR COMMUNICATION OVERVIEW

- Supports communication between:
  - NOS/VE and system operators
  - User jobs and system operators

- An operator console is a terminal "logged-in" with system operator privileges
  - Operator commands and displays are processed by NOS/VE interactive jobs having system operator privileges granted by the NOS/VE user validation
  - The installation may distribute access privileges between users

    Status and control of hardware components

    Status and control of NOS/VE user jobs and their resource allocation

    Status and control for the operating system, system jobs and special applications

    Allows NOS/VE to request operator assistance for tape mounts

    Provides visible information on system operation, current parameter values, etc.

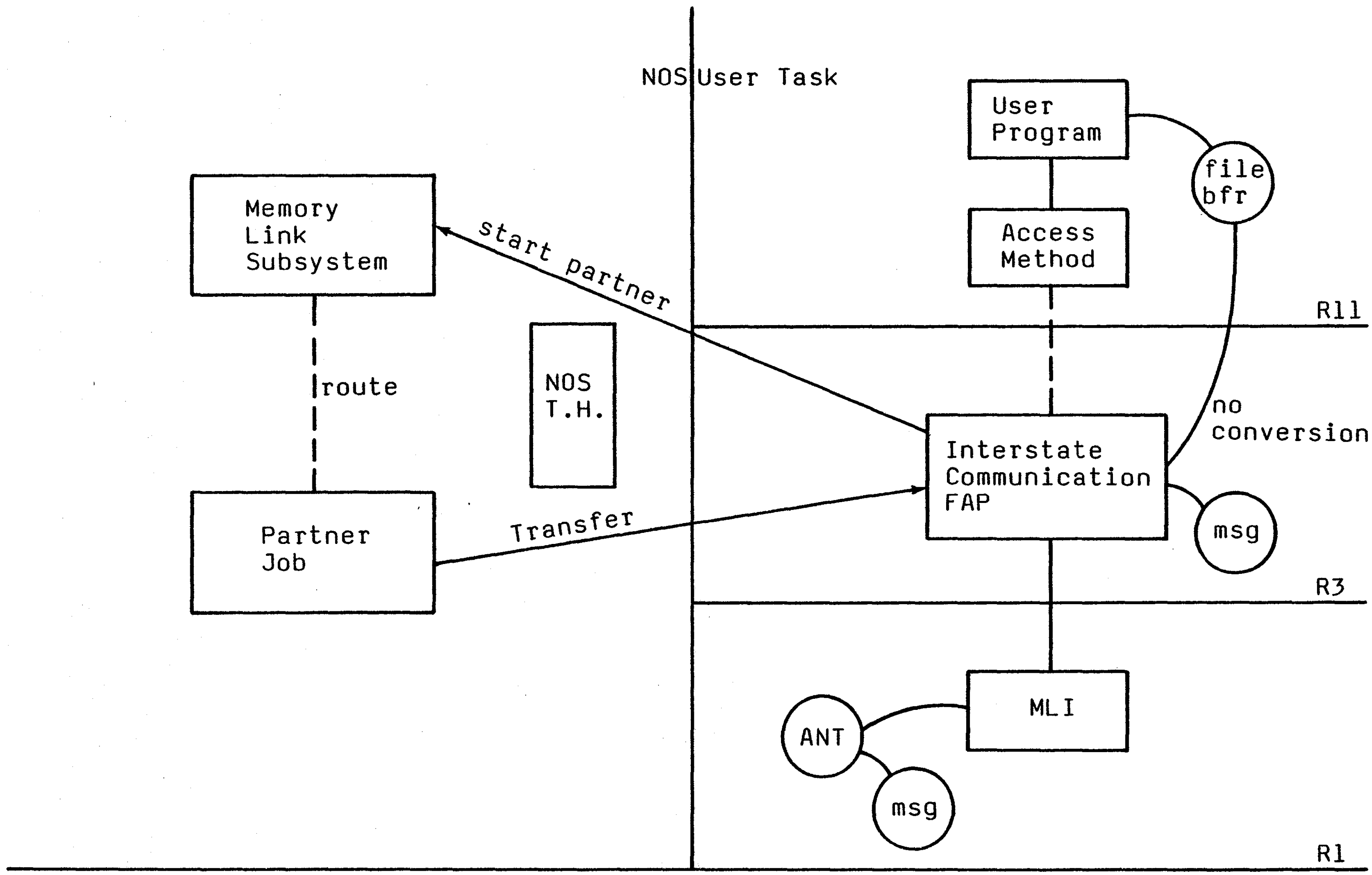
    Reports hardware and software problems

    Allows operator-job and operator-terminal communication

    Supports on-line system debugging

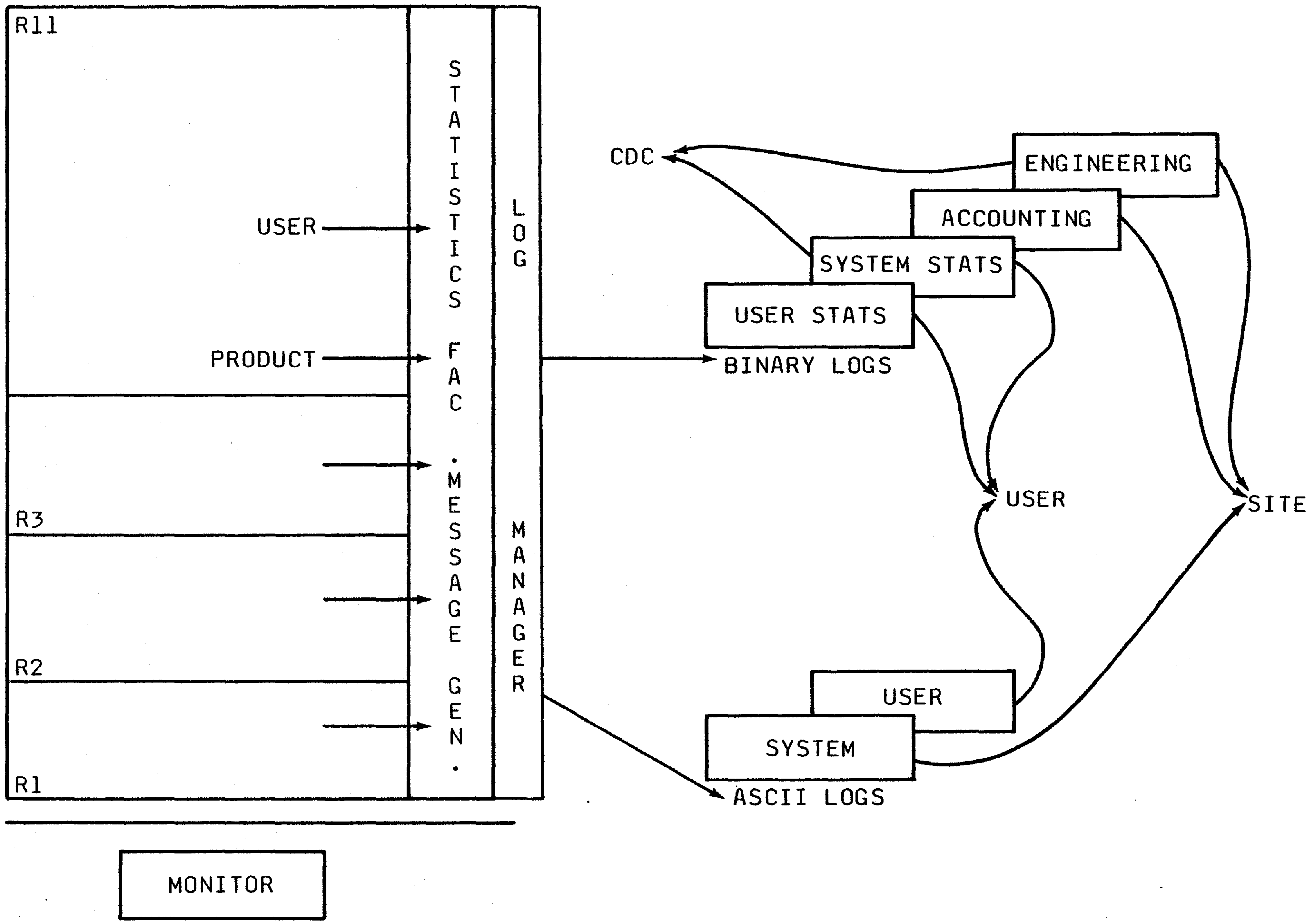
    Supports on-line diagnostic initiation and control

```
OFP$SET_DATE(m,d,y,status)

OFP$SET_TIME(h,m,s,status)

OFP$SET_SYSTEM_STATE(type,value,status)
    Types: security
           attended
           maintenance
           debug

OFP$SET_JOB_CLASS_LIMIT(class,limit,status)
    Classes: interactive
             batch

OFP$GET_SYSTEM_INFORMATION(info,status)
    Info: header
          version
          batch count and limit
          interactive count and limit

OFP$GET_OPERATOR_ACTIONS(actions,status)
    Action entry: ordinal
                  job name and id
                  task id
                  response boolean
                  message
```

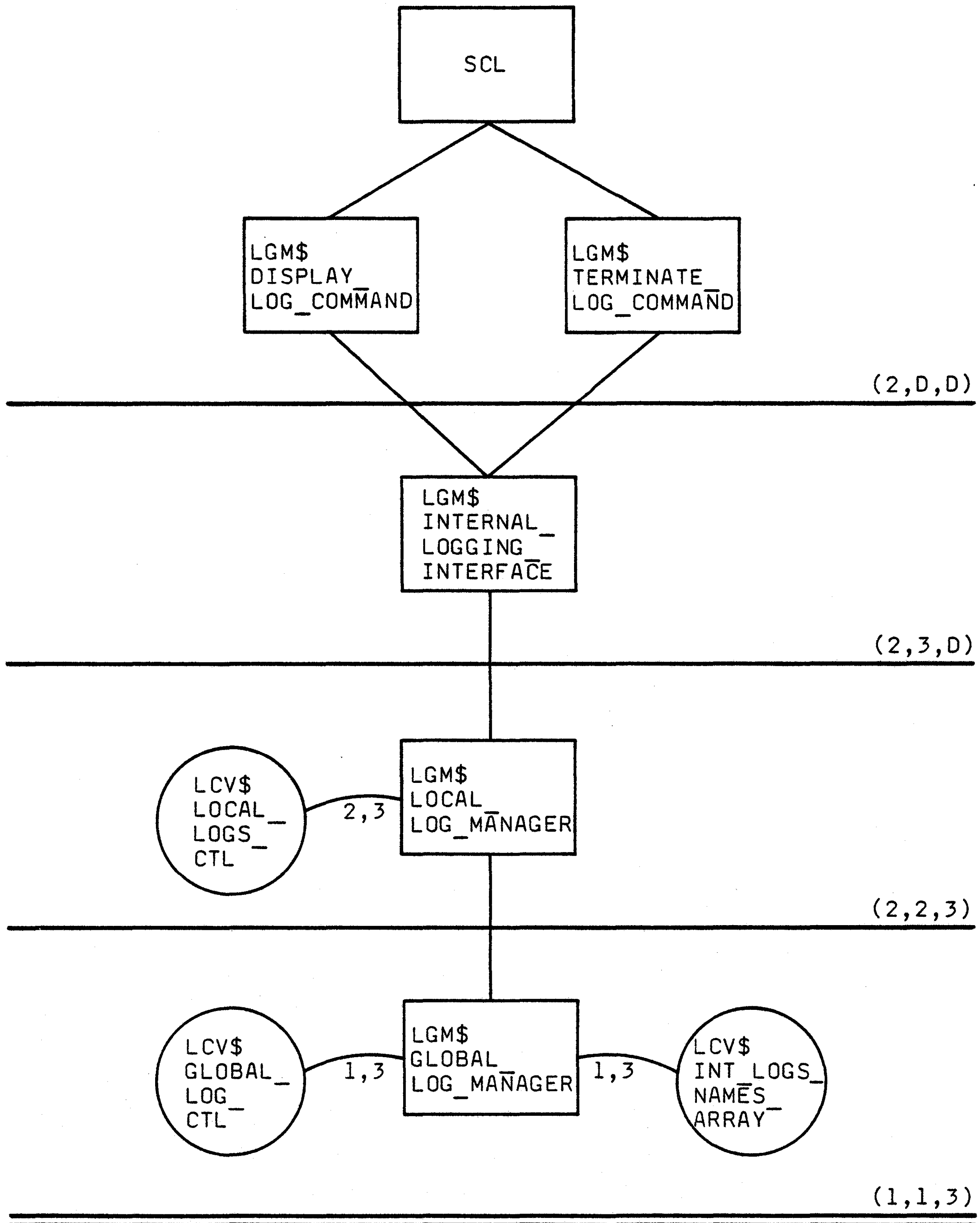


INTERSTATE COMMUNICATION

LOGS


R11
USER
PRODUCT
R3
R2
R1
STATISTICS FAC .MESSAGE GEN.
LOG MANAGER
MONITOR
CDC
ENGINEERING
ACCOUNTING
SYSTEM STATS
USER STATS
BINARY LOGS
USER
SITE
USER
SYSTEM
ASCII LOGS

LOG MANAGER


SCL
LGM$ DISPLAY_ LOG_COMMAND
LGM$ TERMINATE_ LOG_COMMAND
(2,D,D)
LGM$ INTERNAL_ LOGGING_ INTERFACE
(2,3,D)
LCV$ LOCAL_ LOGS_ CTL
2,3
LGM$ LOCAL_ LOG_MANAGER
(2,2,3)
LCV$ GLOBAL_ LOG_ CTL
1,3
LGM$ GLOBAL_ LOG_MANAGER
1,3
LCV$ INT_LOGS_ NAMES_ ARRAY
(1,1,3)

LGP$ADD_ENTRY_TO_BINARY_LOG
(log,entry_address,log_address,cycle,status)

LGP$APPEND_JOB_LOG_TO_OUTPUT
(status)

LGP$BUILD_DISPLAY_OF_ASCII_LOG
(log,scroll_size,status)

LGP$INTERCEPT_LOG_IO_REQUEST

(fid,call_block,layer_no,status)
This is a FAP.

LGP$SETUP_ACCESS_TO_LOCAL_LOGS(status)

LGP$SETUP_ACCESS_TO_GLOBAL_LOGS(logs,status)

PROBE

A PROBE IS THAT PORTION OF SOFTWARE RESPONSIBLE FOR COLLECTING AND EMITTING A STATISTIC TO THE STATISTICS FACILITY.

- PROBES ARE EMBEDDED IN KEY AREAS OF THE SYSTEM, BUT ARE NOT SUBJECT TO GUIDELINES LIKE KEYPOINTS.
- THE PRECISE LOCATION OF PROBES AND THE INFORMATION REPORTED WILL BE DETERMINED BY THE REQUIREMENTS OF THE COMPONENTS IN WHICH THEY LIE.
- THE FREQUENCY AT WHICH A PROBE EMITS A STATISTIC TO THE STATISTICS FACILITY IS DETERMINED BY THE SUPERORDINATE COMPONENT.
- A PROBE DOES NOT ASCRIBE ANY INHERENT QUALITIES TO A STATISTIC.
- THERE SHOULD BE A ONE-TO-ONE CORRESPONDENCE BETWEEN A PROBE AND STATISTIC.

NOS/VE STATISTIC

A NOS/VE STATISTIC HAS THREE COMPONENTS:

- STATISTIC CODE : AN ORDINAL THAT UNIQUELY IDENTIFIES THE STATISTIC.
- DESCRIPTIVE DATA: A STRING INDICATING THE OCCURRENCE OF A SYSTEM OR JOB EVENT.
- COUNTERS : A SEQUENCE OF COUNTERS CONTAINING REPORTED VALUES OF SYSTEM OR JOB VARIABLES.

PRODUCT STATISTICS COLLECTED BY NOS/VE

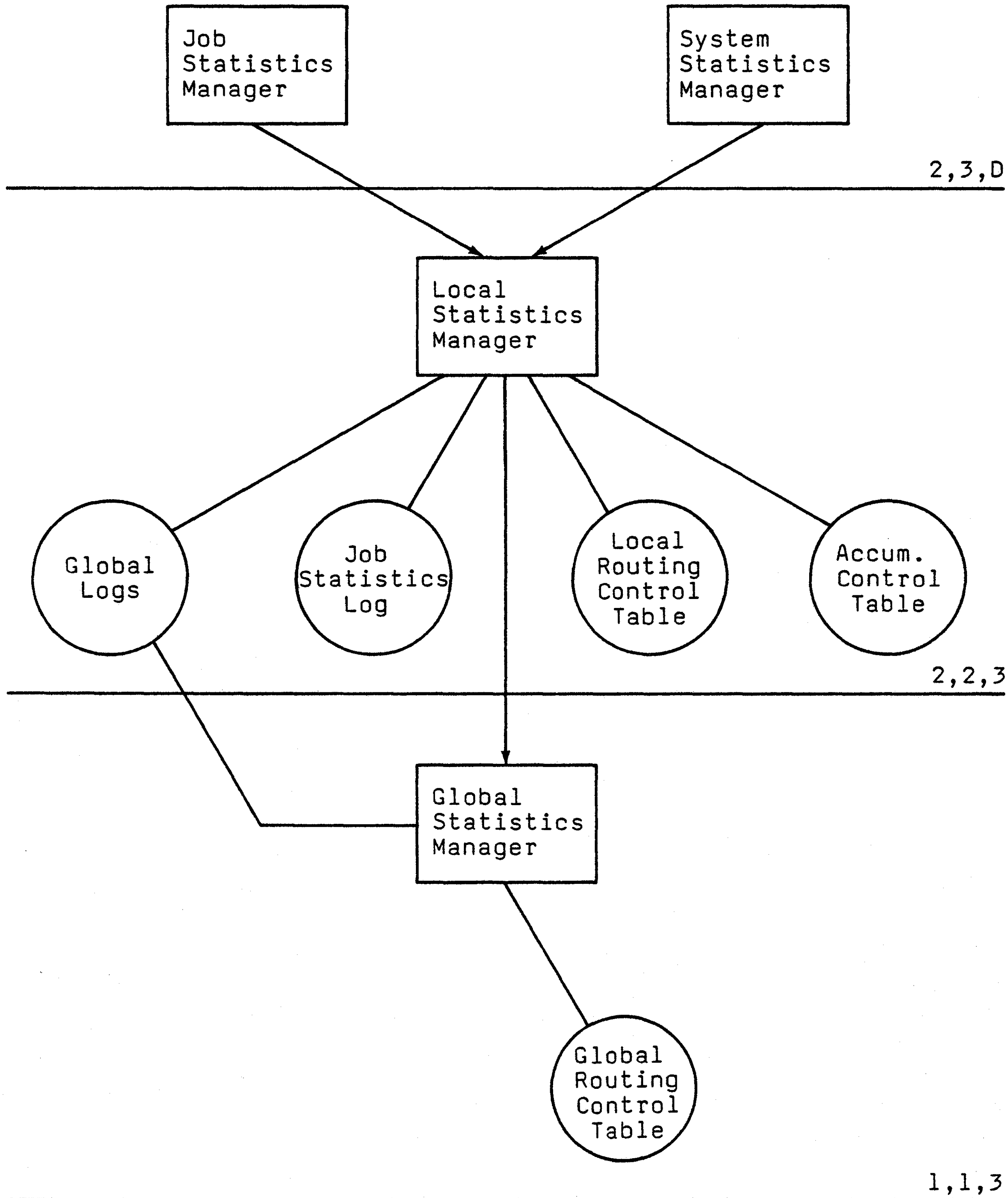
In general, the O/S is responsible for collecting job step statistics that can be determined external to the product, that is statistics that the O/S is capable of determining.

For each product identified in SIS section 4.1 that is directly invoked by the user, e.g., via command or as a program initiated task, NOS/VE will record resources used per invocation. Resources accounted for include:

- Total CP-time
- Maximum virtual memory used
- Maximum real memory used
- Average working set size
- CP-time per memory size used
- Number of I/O requests
- Number of data read/written to files

Additional data to be collected for each invocation of a product include:

- Origin of job step - batch command, terminal command, procedure file, executing job.
- Type of termination - normal, product error, time limit, invalid memory request, operator drop, and so on. A recovered condition does not cause product termination.
- Abnormal conditions recovered from.
- Average interactive response time for interactive products - the average elapsed time between input data available and output data issued to terminal.
- The fact that the product was invoked (added to count of the number of separate invocations).
- Number of modules loaded (input units for the loader)
- Source languages of modules loaded (added to language usage count).


Job
Statistics
Manager
System
Statistics
Manager
2,3,D
Local
Statistics
Manager
Global
Logs
Job
Statistics
Log
Local
Routing
Control
Table
Accum.
Control
Table
2,2,3
Global
Statistics
Manager
Global
Routing
Control
Table
1,1,3

GLOBAL BINARY LOG FORMAT

BINARY DATE AND TIME
STATISTIC CODE
STATISTIC IDENTIFIER
JOB SEQUENCE NUMBER
GLOBAL TASK ID
CONDENSING FREQUENCY
NUMBER OF COUNTERS
DESCRIPTIVE DATA SIZE
COUNTER_1
COUNTER_2
. . .
COUNTER_N
DESCRIPTIVE DATA
. . . .

ACCUMULATION CONTROL

STATISTIC CODE
ACCUMULATION CONTROL TYPE
ACCUMULATION ADDRESS
FREQUENCY ADDRESS
THRESHOLD
FORWARD LINK
BACKWARD LINK

GLOBAL AND LOCAL ROUTING CONTROL TABLE

STATISTIC CODE
IDENTIFIER
ROUTING CONTROL TYPE
ENABLED
CONDENSING_ADDRESS
THRESHOLD
INTERVAL SIZE
INTERVAL_END_TIME
LOG_CYCLE
FORWARD_LINK
BACKWARD LINK

CONDENSING

The first counter of a statistic can be condensed, that is, the information will be collected in the counter until either time runs out or a certain number occur. When the condensing threshold is reached, a new entry is logged and collecting starts again. This might be used to count page faults or total monitor time.

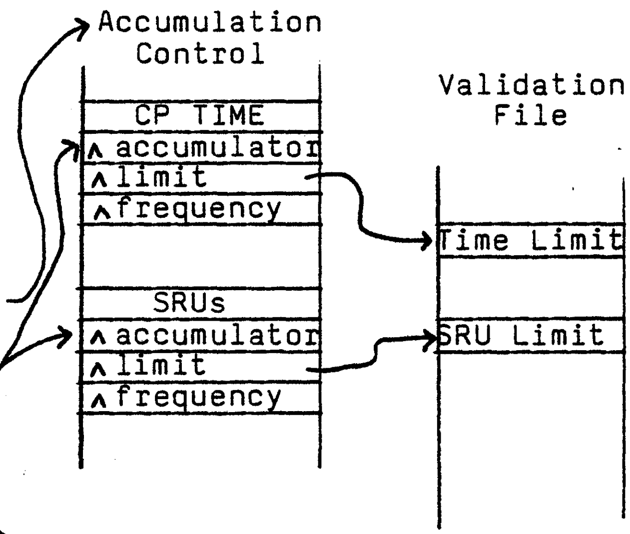
ACCUMULATING

Accumulation also involves collecting occurrences of an event. When the threshold (limit) is reached, some action is taken. Typically the job monitor will be signaled and will take further action. Currently this is being used for CP time and SRUs.

BREAKOUT

Sometimes it is necessary to seek local and global statistics of the same thing. An example might be job time. It would be necessary to have total job time as well as the time for individual jobs to compute standard deviation. If breakout is established, the statistics manager will enter in both the local and the global logs.

```
PROCEDURE [XREF] sfp$establish_system_statistic (identifier:
    sft$statistic_identifier;

    statistic_code: sft$statistic_code;

    log_name: pmt$global_binary_logs;

    breakout: boolean;

    condensing_control: sft$condensing_control;

    VAR status: ost$status);


PROCEDURE [XREF] sfp$enable_system_statistic (statistic_group:
    sft$statistic_group;

    VAR status: ost$status;


PROCEDURE [XREF] sfp$disable_system_statistic (statistic_group;
    sft$statistic_group;

    VAR status: ost$status);


PROCEDURE [XREF] Sfp$disestablish_system_stat (identifier:
    sft$statistic_identifier;

    statistic_code: sft$statistic_code;

    log_name: pmt$global_binary_logs;

    breakout: boolean;

    VAR status: ost$status);


PROCEDURE [XREF] sfp$emit_system_statistic (identifier:
    sft$statistic_identifier;

    statistic_code: sft$statistic_code:

    descriptive_data: sft$descriptive_data;

    counter: sft$counters;

    VAR status: ost$status);
```

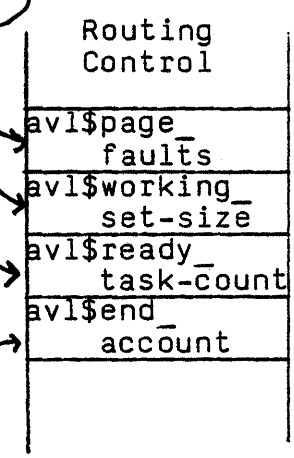


AVP$BEGIN_ACCOUNT

- Establish Accounting Stats

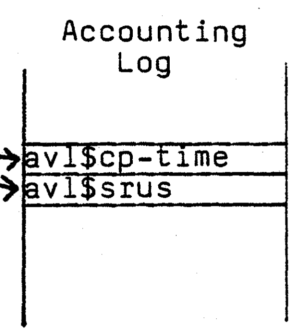
AVP$MONITOR_STATISTICS_HANDLER (every2FFFFF m.sec)

- Emit Accounting Stats
- Emit System Stats
- If Accounting Stat exceeds limit, signal job monitor

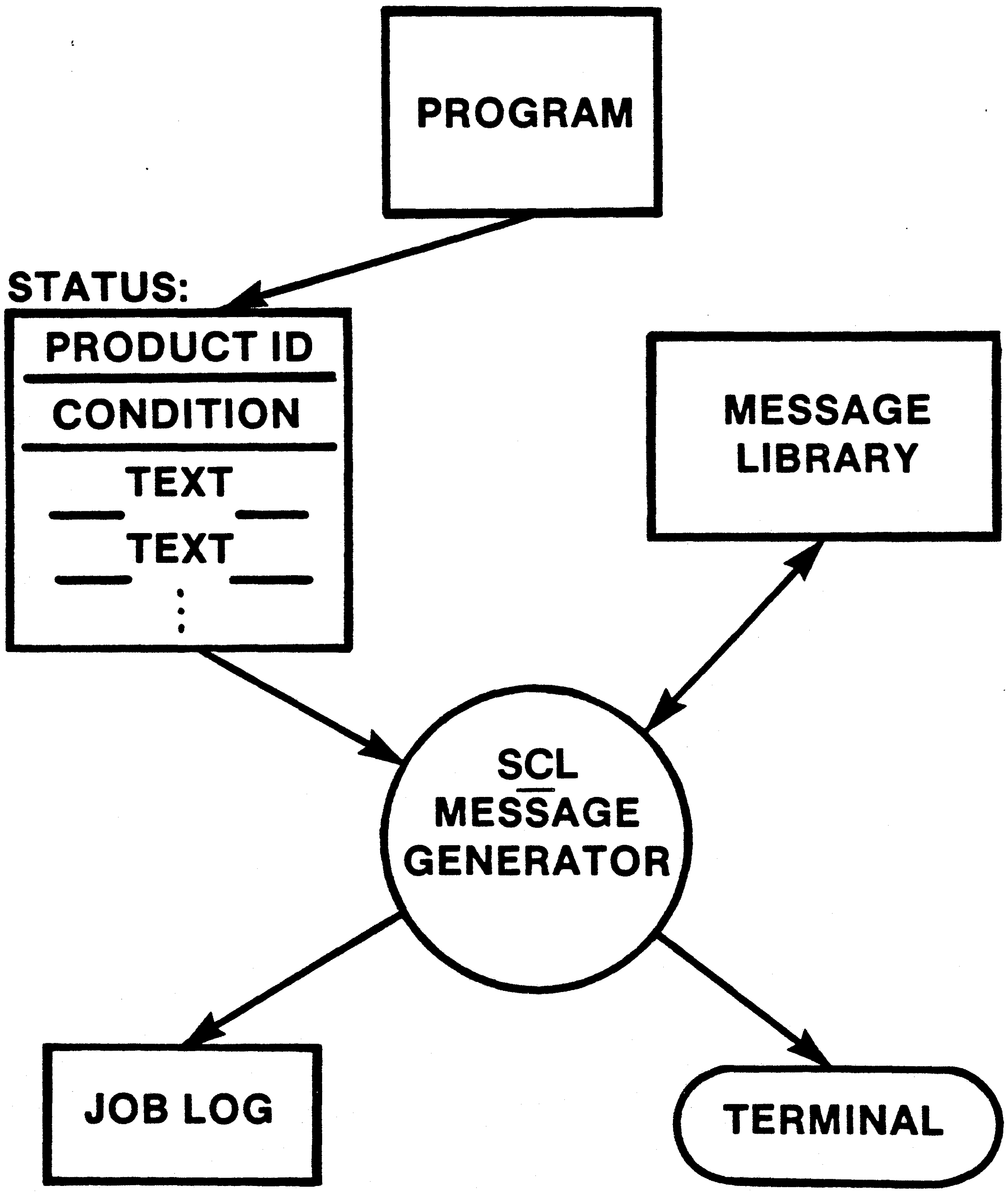


AVP$END_ACCOUNT

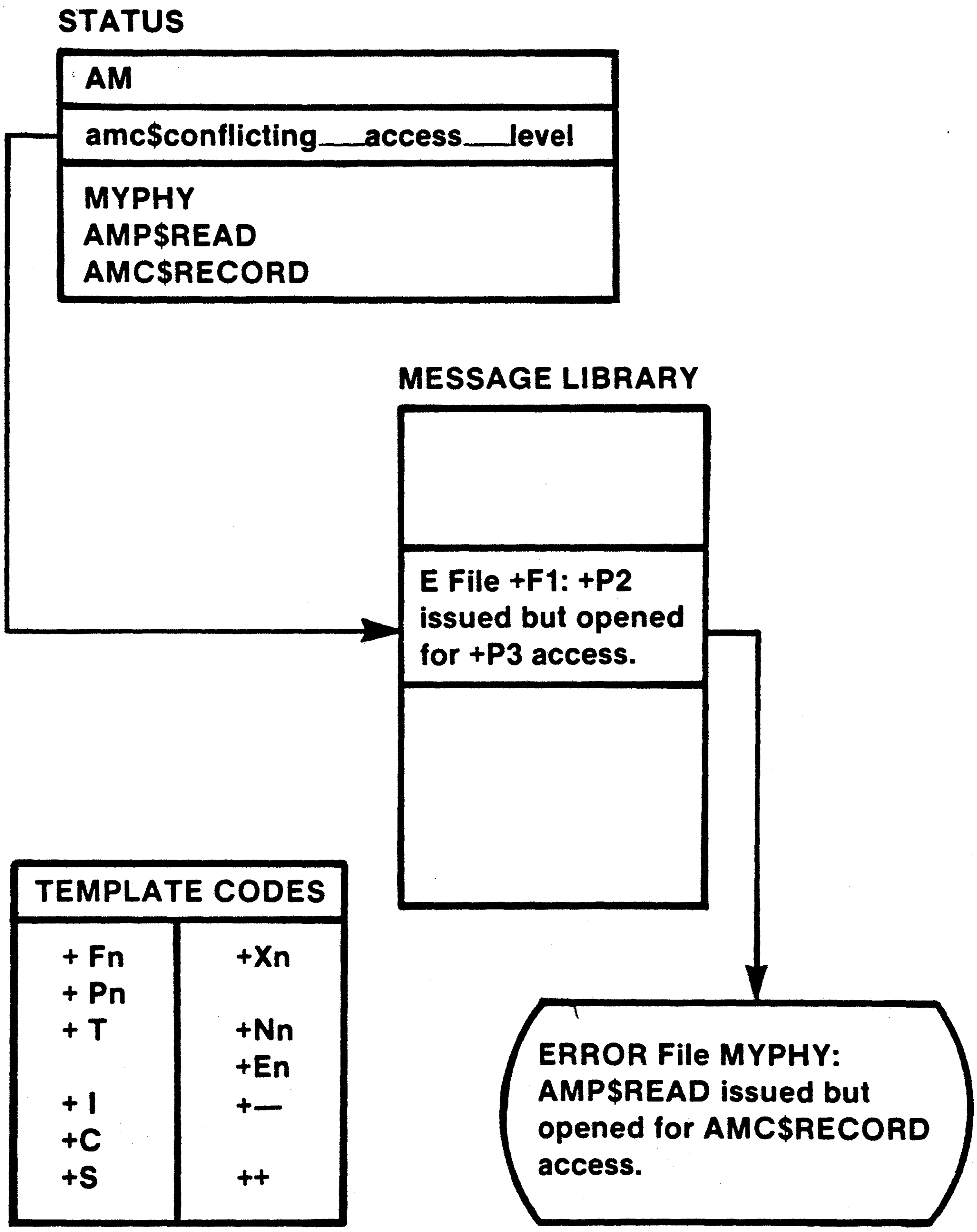
- Call AVP$MONITOR_STATISTICS_HANDLER
- Establish Local Statistic
- Emit System Statistic
- Disestablish Local Statistic



MESSAGE GENERATOR



MESSAGE GENERATION



- OSP$GENERATE_MESSAGE
  (message_status, status)

- OSP$FORMAT_MESSATE
  (message_status, message_level, max_message_line, message, status)

- OSP$GET_STATUS_SEVERITY
  (condition, severity, status)

- OSP$SET_STATUS_ABNORMAL
  (id, condition, text, status)

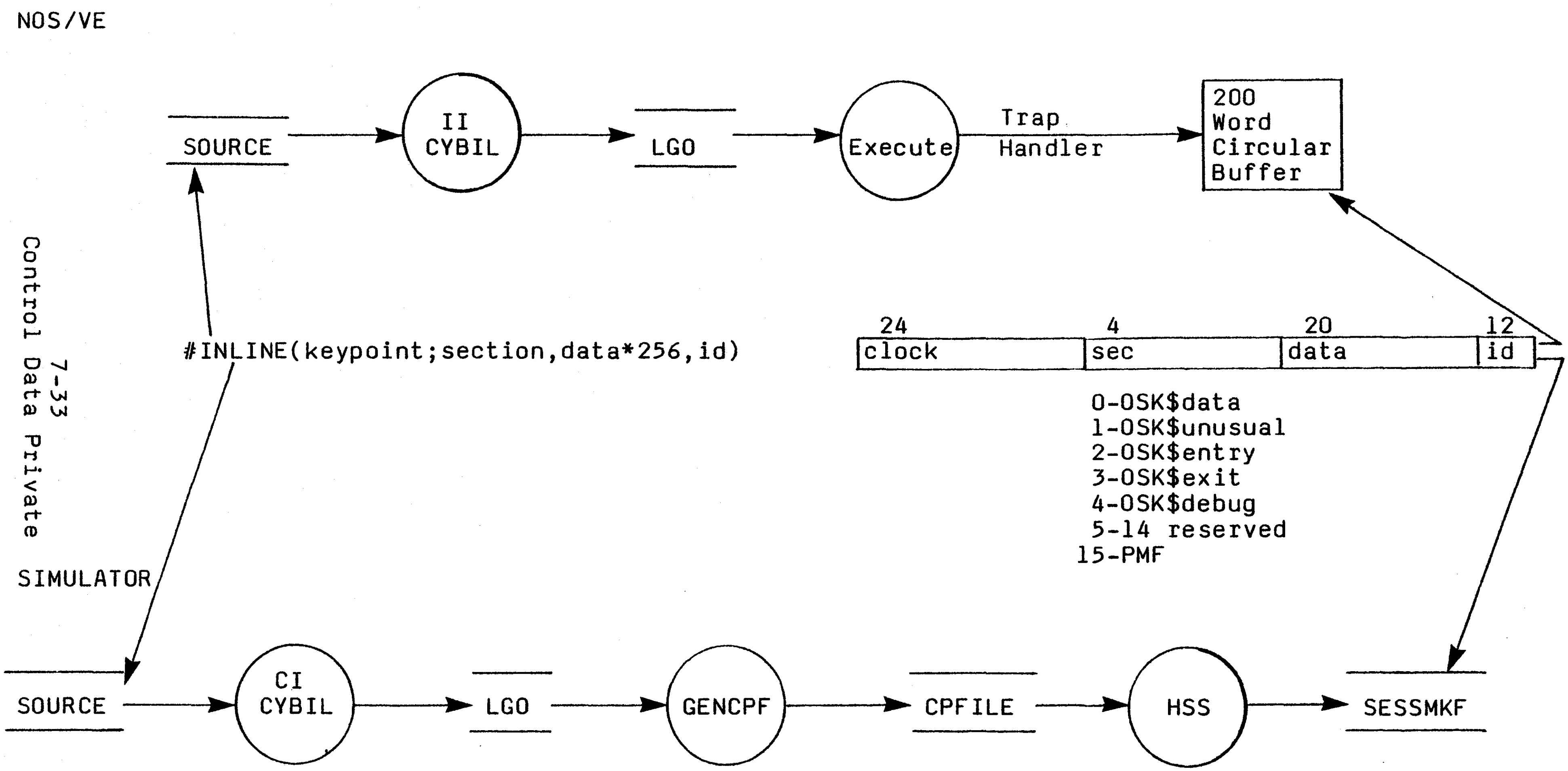
- OSP$APPEND_STATUS_PARAMETER
  (delimiter, text, status)

- OSP$APPEND_STATUS_INTEGER
  (delimiter, int, radix, include_radix_specifier, status)

- Parameters

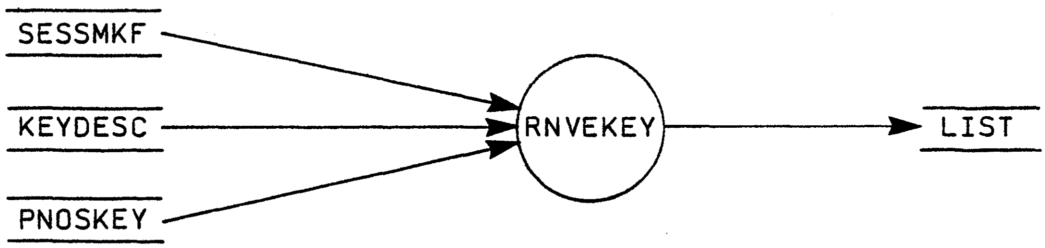
  - Message-level: full, brief, explain
  - Severity: informative, warning, error, fatal
  - Message: sequence, # lines, # char/line, text
  - Delimiter: osc$status_parameter_delimiter or any other
  - Include_radix_specifier: radix will be part of text added to status

KEYPOINT FLOW



See Procedures and Conventions CH.9.

KEYPOINT FILES



KEYDESC

Section id
Section #
Proc id
Special Marks
Length of Data
Data Description
Format
Text

LIST

- o Summary
- o Itemized List
  - clock
  - elapsed time
  - data
  - text
  - mode
  - task id
  - section id

RNOSKEY

CV maxprocid n
CV undefined
CV defined
CV ident
RUN
END

LESSON 8
JOB CONTROL

LESSON PREVIEW

QUEUED FILE MANAGEMENT
JOB MANAGEMENT
JOB RELATED TABLES
SCHEDULING JOBS
DISPATCHING TASKS

OBJECTIVES

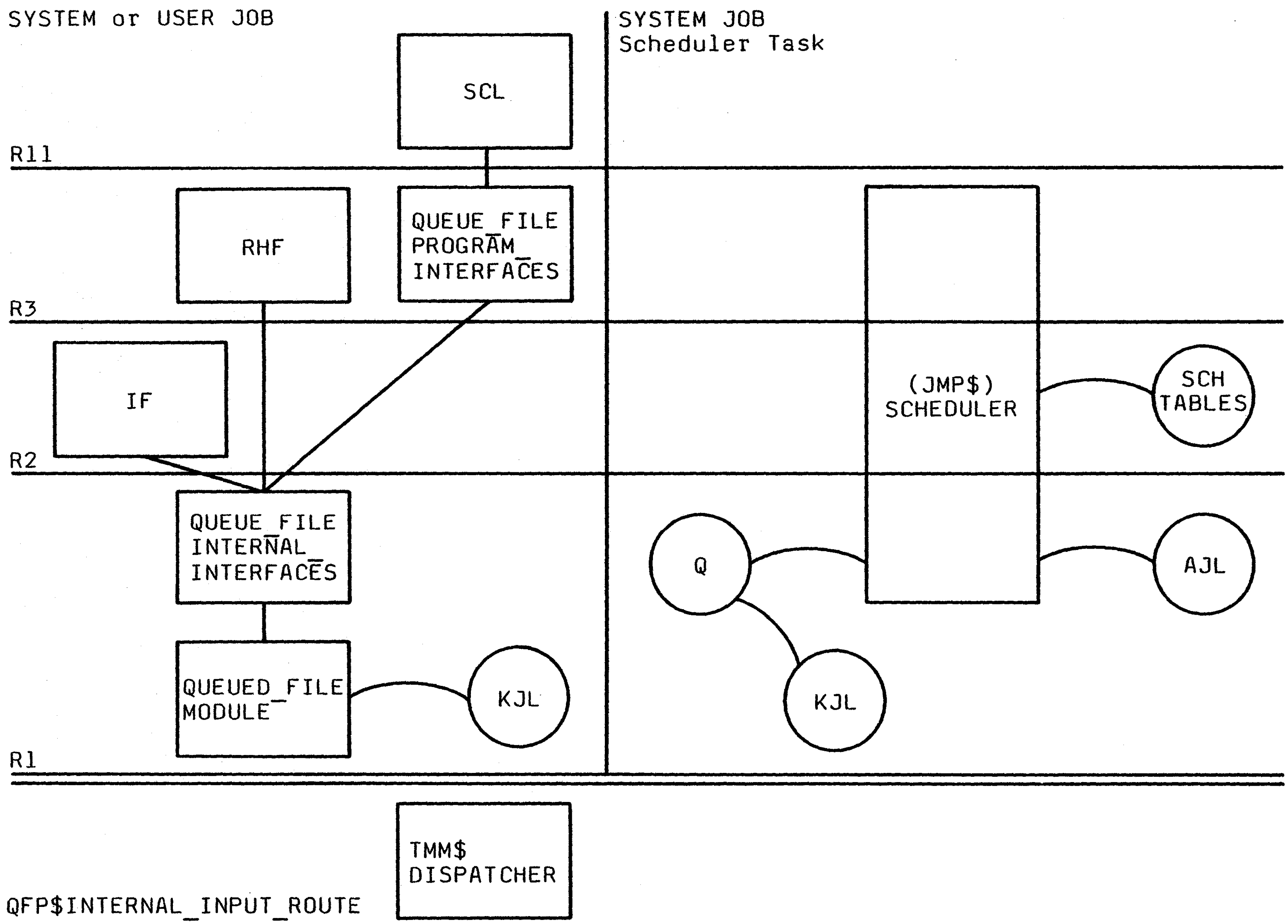
After completing this lesson the student should be able to--

- EXPLAIN THE LINKAGE AND HANDLING OF THE KJL AND KOL
- EXPLAIN THE LINKAGE AND HANDLING OF THE MAJOR JOB TABLES--AJL & JCB
- EXPLAIN HOW BATCH AND INTERACTIVE JOBS ARE VALIDATED
- EXPLAIN THE JOB SCHEDULING ALGORITHM
- EXPLAIN THE TASK DISPATCHING ALGORITHM
- EXPLAIN THE PROCESS OF BEGINNING AND TERMINATING JOBS.

EXERCISE

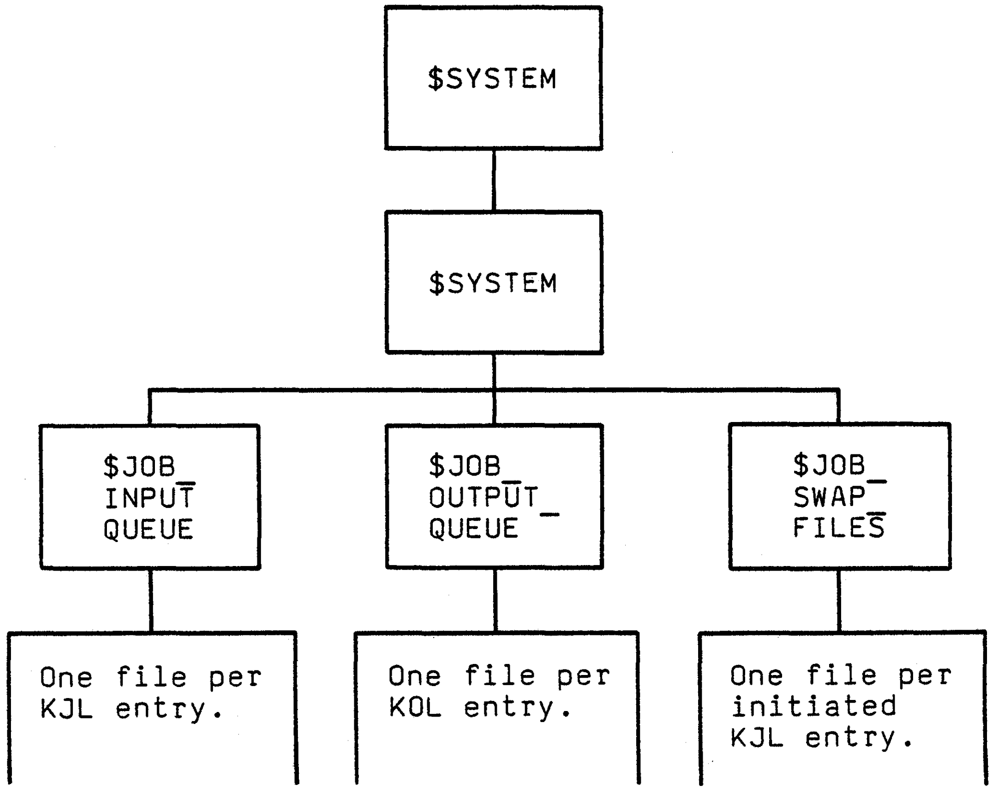
GIVEN A DUMP, DETERMINE THE STATUS OF JOBS IN THE SYSTEM.

JOB ENTRY



SYSTEM or USER JOB
SYSTEM JOB
Scheduler Task
SCL
R11
RHF
QUEUE_FILE
PROGRAM_
INTERFACES
R3
IF
(JMP$)
SCHEDULER
SCH
TABLES
R2
QUEUE_FILE
INTERNAL_
INTERFACES
Q
AJL
QUEUED_FILE
MODULE
KJL
KJL
R1
TMM$
DISPATCHER
QFP$INTERNAL_INPUT_ROUTE

QUEUED FILES



FILE NAMES: 1. user_job_name
2. system_job_name
AAAA$,AAAB$,...

RECOVERY: The $SYSTEM catalog is recovered like any PF catalog. Information in the system file labels (SFL) of the files is sufficient to reconstruct the KJL and the KOL.

SCHEDULING OVERVIEW

1) Jobs can be divided into (currently) one of three classes: system, batch, and interactive. Scheduler's class attribute table is used to delineate the classes.

   Low, high, and initial priorities are defined as are memory values. The exclude class flag will inhibit the initiation of jobs from this class. The self-terminating capability will allow queued jobs of a class to be initiated even though the maximum active jobs for that class have been exceeded. The job will be up long enough to bring itself down. Currently interactive class jobs have this capability.

   The initiator is within the Job Scheduler task. When a job is routed, it will be queued and the scheduler is signaled.

2) Job swapping is controlled by two parameters:

   a) The maximum number of swapped jobs in a class.

   b) The maximum overall number of swapped jobs.

   Swapping is initiated as a result of three conditions:

   a) If the scheduler determines that the system is thrashing, a candidate will be chosen and swapout will be performed. The two rules given above will be overridden.

   b) The scheduler will periodically examine the input and active job queues. If a job in input has a higher priority than one executing, a swap request will be issued for the active job. This swap request obeys the two parameters governing the swap function.

   c) If memory contention is high and a terminal break occurs, that job will be swapped.

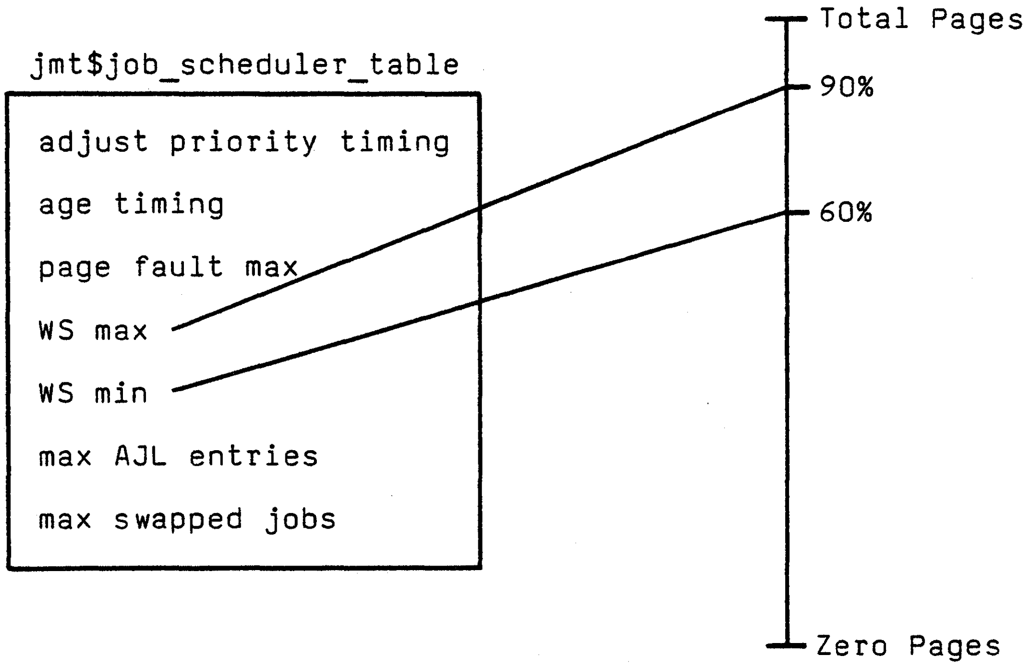
3) Job priority adjustment is limited to aging queued jobs, aging swapped jobs, and adjusting the priorities of executing jobs.

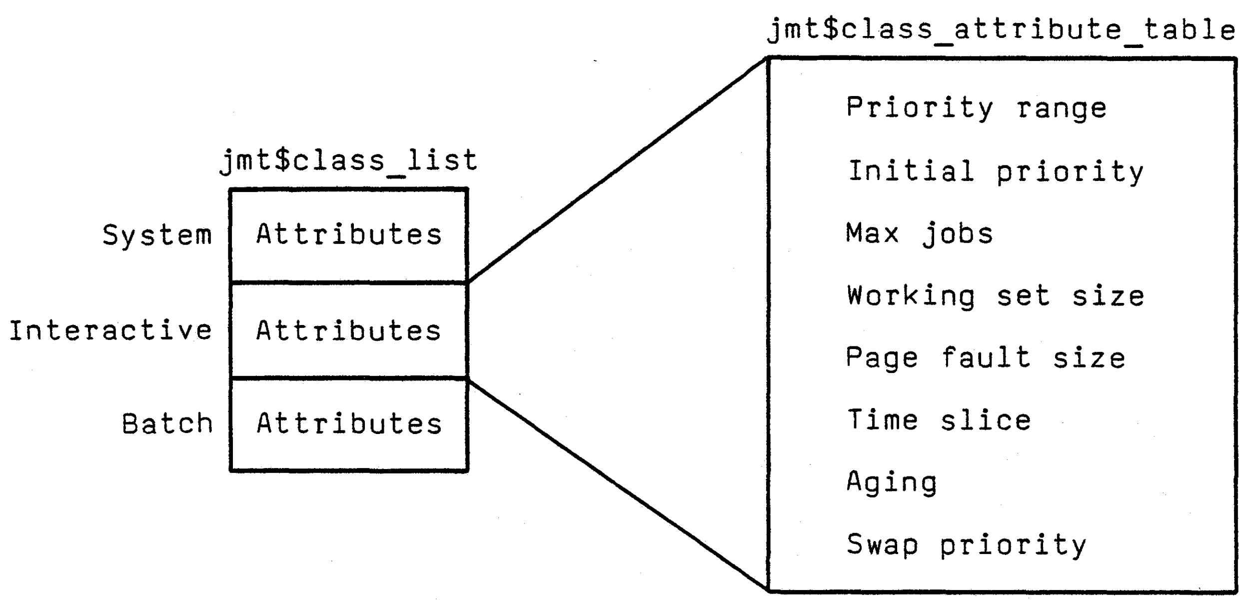
   The aging function will increment job priority based on values local to the class. There are two aging increments for each class: one for input list and the other for swap list. The aging function will be performed on a periodic basis.

   Executing jobs will have their priorities adjusted according to several factors. If the job has just been swapped in, it will be given a priority boost to prevent it from being swapped out immediately. If the job's ready task count falls to zero, it will lose priority points. (This may or may not initiate swapping.) If the job's time or memory limits are exceeded, it will be switched to another internal class. Currently there are secondary interactive and batch classes.

   When a swapped job receives a signal, the scheduler will increase that job's priority which will result in the job being activated sooner.

SCHEDULING TABLES





SCHEDULING PROCESS

1. Check for Thrashing

   - Add Working Set (ws) from all AJL entries.

   - If the sum is in the thrashing range, swap jobs til the sum is out of that range. Start with the job with largest ws.

   - Stop.

2. Check page fault rate (R2)

   - If page fault rate > page fault max in jmt$job_scheduler_table, increase memory manager's aging internal.

3. Fill Free Memory

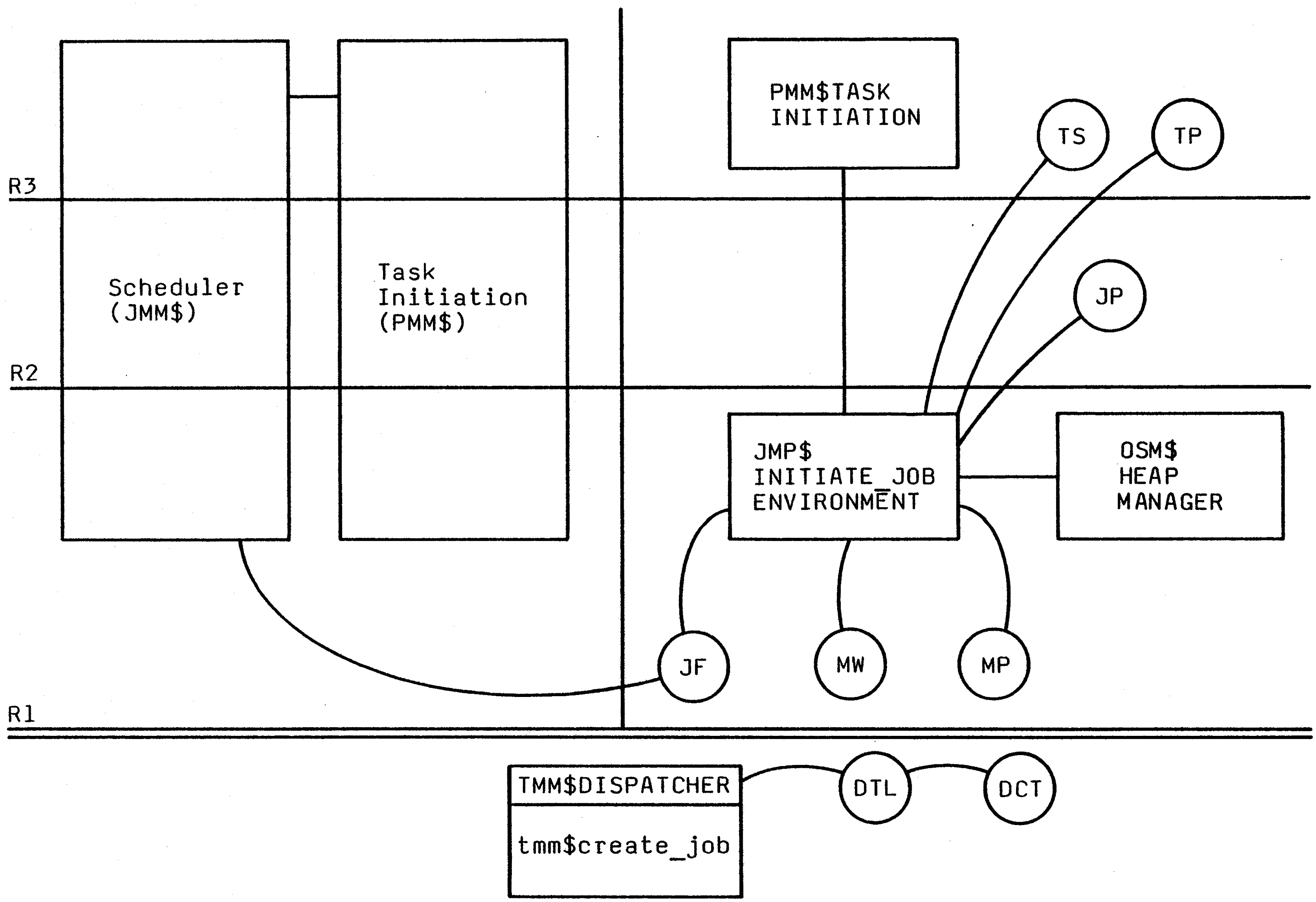
   - Built temporary queues for each state (active, queued, swapped) for each class (batch, interactive, system, etc.).

   - Calculate the number of free pages between the current value and ws_max.

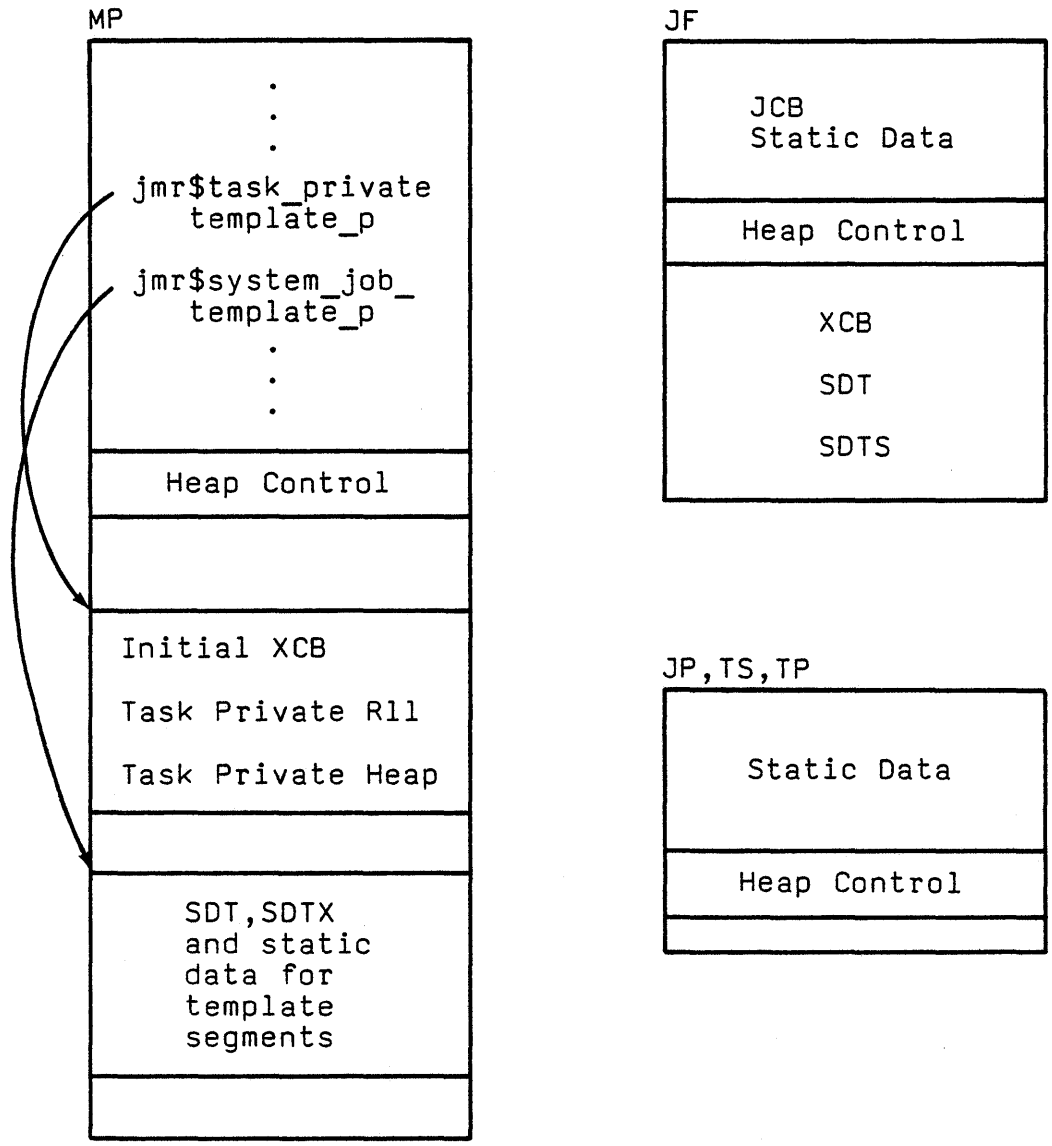
   - Select the algorithm (∧proc). The only R1 algorithm gets the highest priority queued job from each class and compares it with the highest priority swapped job. If the queued job wins it is initiated, otherwise swap. Continue until ws_min $\leqslant$ ws $\leqslant$ ws_max.

   - Stop.

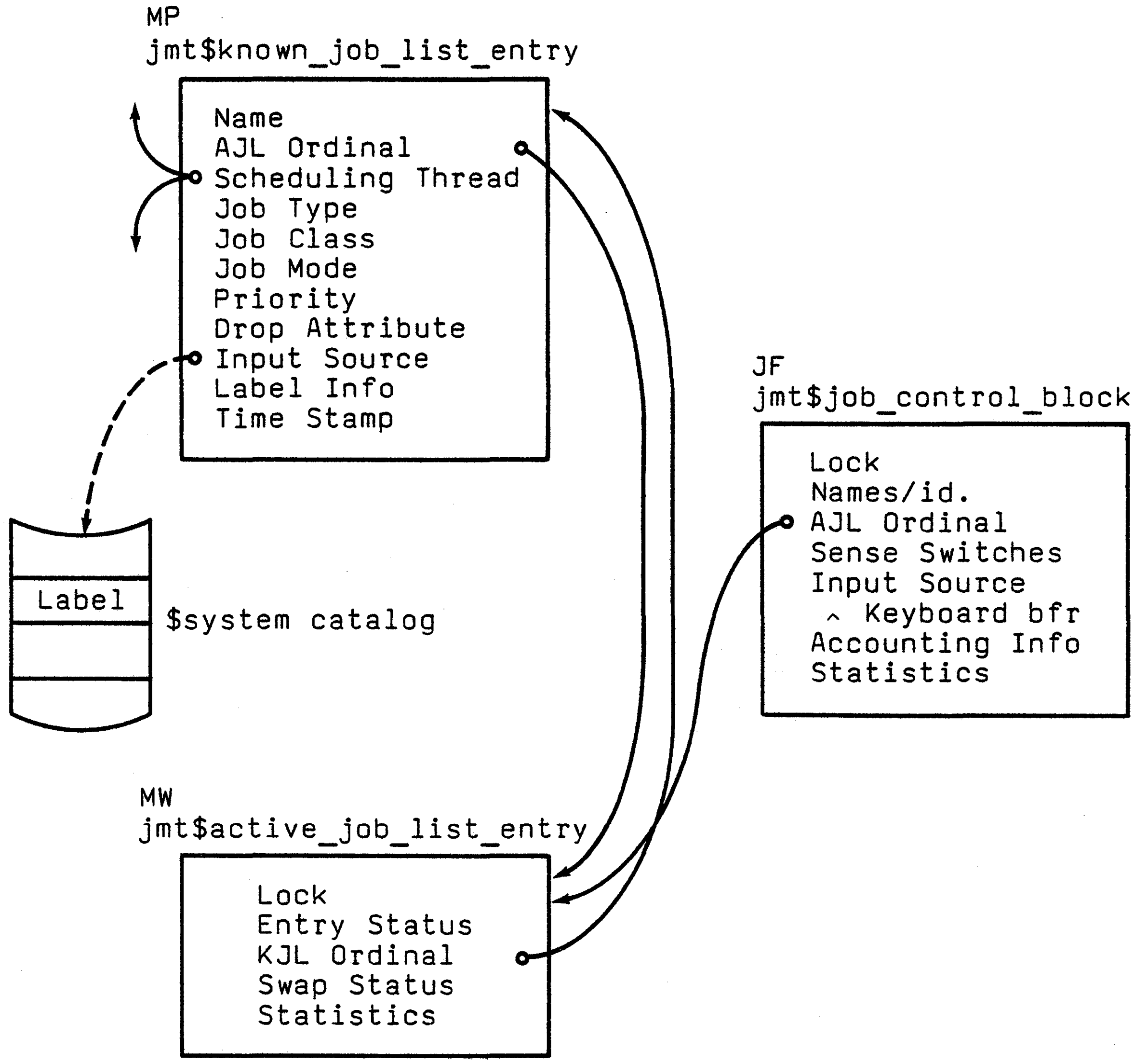
INITIALIZE JOB ENVIRONMENT



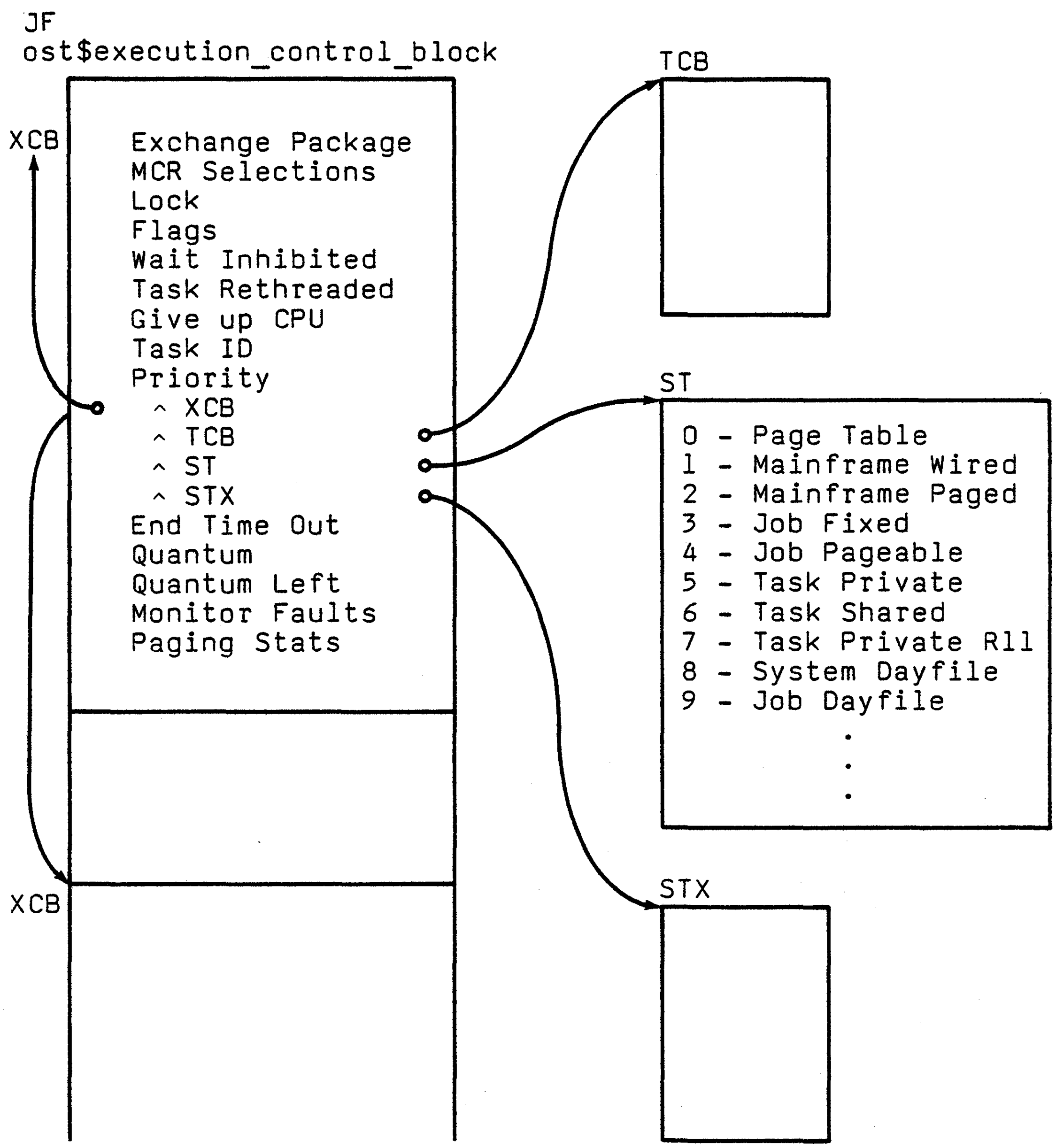
JOB TEMPLATES



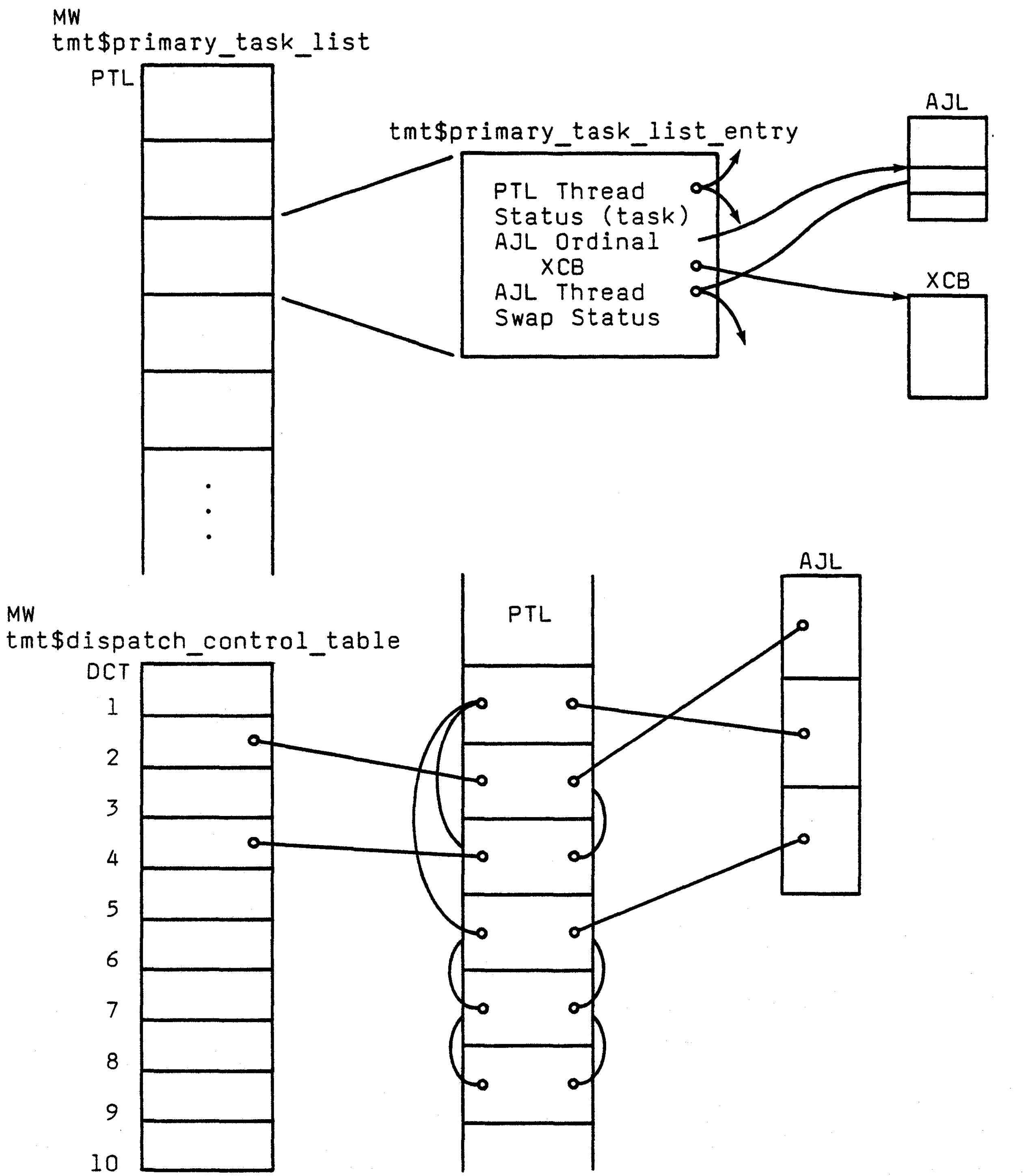
Scheduler creates all segments.
Scheduler initializes Job Fixed.
Initialize_Job_Environment initializes other segments.


MP
jmt$known_job_list_entry
Name
AJL Ordinal
Scheduling Thread
Job Type
Job Class
Job Mode
Priority
Drop Attribute
Input Source
Label Info
Time Stamp
JF
jmt$job_control_block
Lock
Names/id.
AJL Ordinal
Sense Switches
Input Source
^ Keyboard bfr
Accounting Info
Statistics
Label
$system catalog
MW
jmt$active_job_list_entry
Lock
Entry Status
KJL Ordinal
Swap Status
Statistics

EXECUTION CONTROL BLOCK



TASK DISPATCHING TABLES



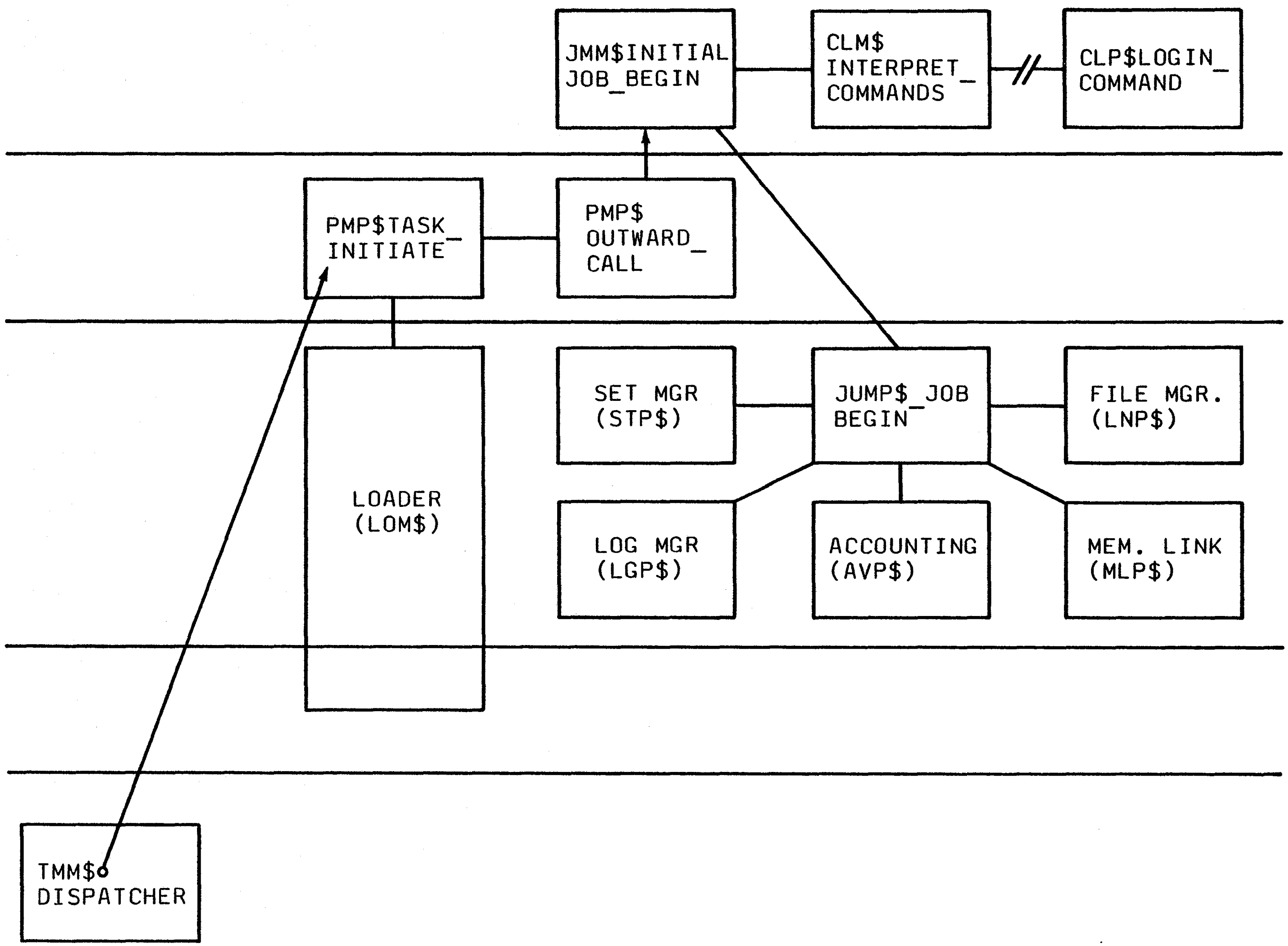
* Currently (R1) all tasks are on DCT thread 4 unless they have a system table locked. Tasks with a system table locked are put on thread 2, and the rethreaded field in the XCB is marked true.

* All tasks on the highest priority thread get 50 m-sec time slice in a round robin fashion as long as there are active tasks on that thread.

* In future releases, all 10 threads will be used. Different threads will have different time slices. These algorithms have not been defined yet.

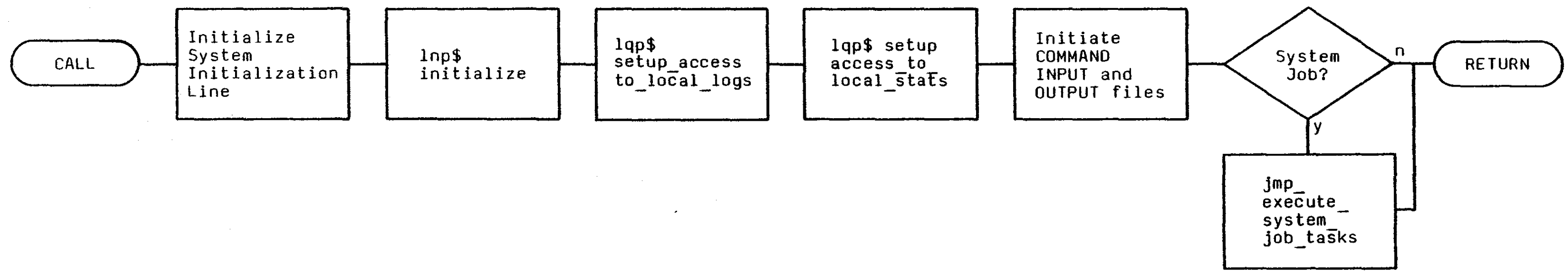
* NOS - NOS/VE scheduling is done in NOS and MIP. If the current NOS job has higher priority, NOS runs; if the current NOS/VE task has higher priority, NOS/VE runs. If the priorities are equal (NOS job default = NOS/VE task = 30) then the CPU is toggled between states. Currently 50 ms are awarded to each side but that can be changed to favor one side or the other. NOS trap handler does the timing. Idle is in NOS.

JOB MONITOR

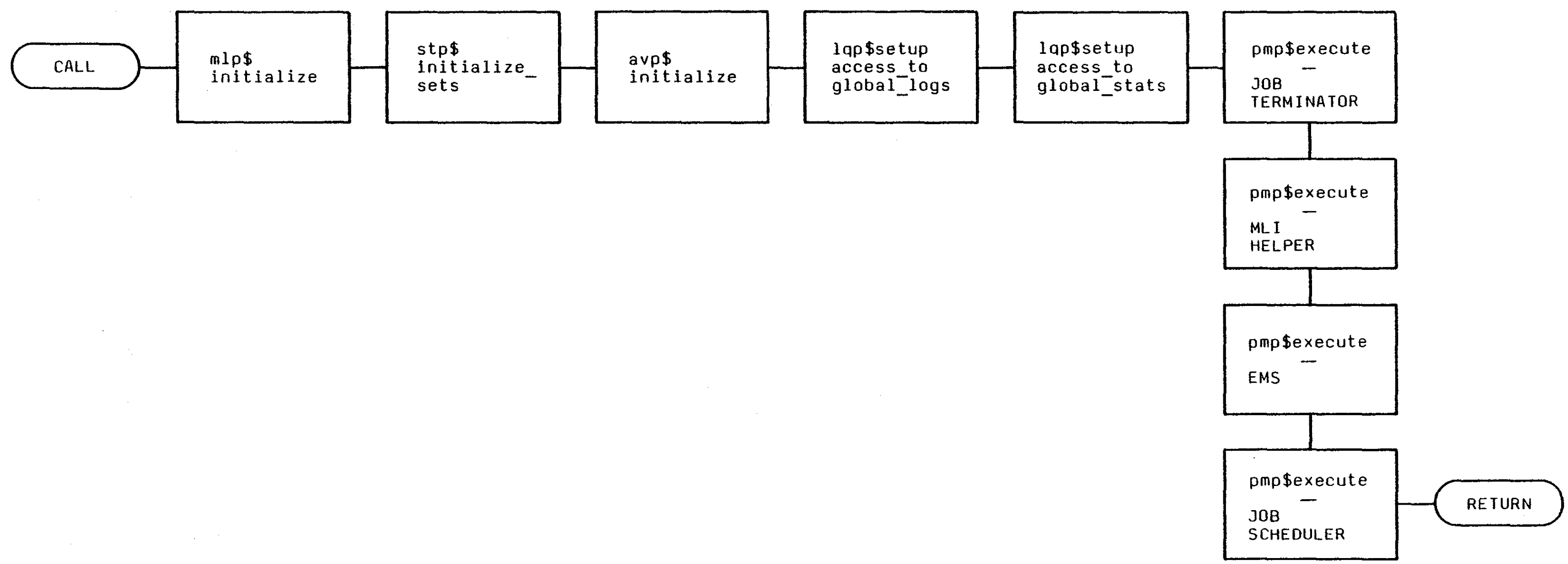


JMM$INITIAL_
JOB_BEGIN
CLM$
INTERPRET_
COMMANDS
CLP$LOGIN_
COMMAND
PMP$TASK_
INITIATE
PMP$
OUTWARD_
CALL
LOADER
(LOM$)
SET MGR
(STP$)
JUMP$_JOB
BEGIN
FILE MGR.
(LNP$)
LOG MGR
(LGP$)
ACCOUNTING
(AVP$)
MEM. LINK
(MLP$)
TMM$
DISPATCHER

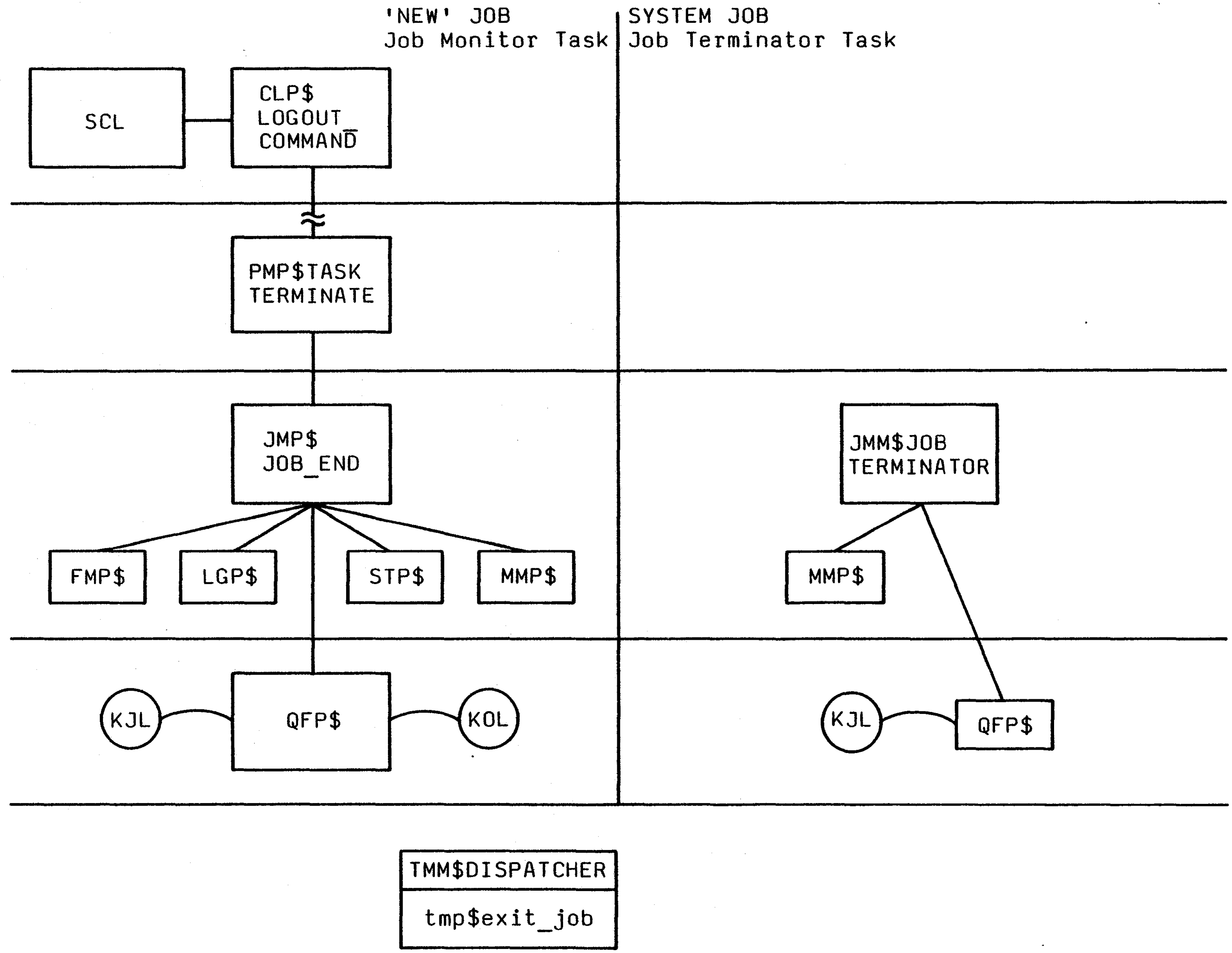
jmp$job_begin



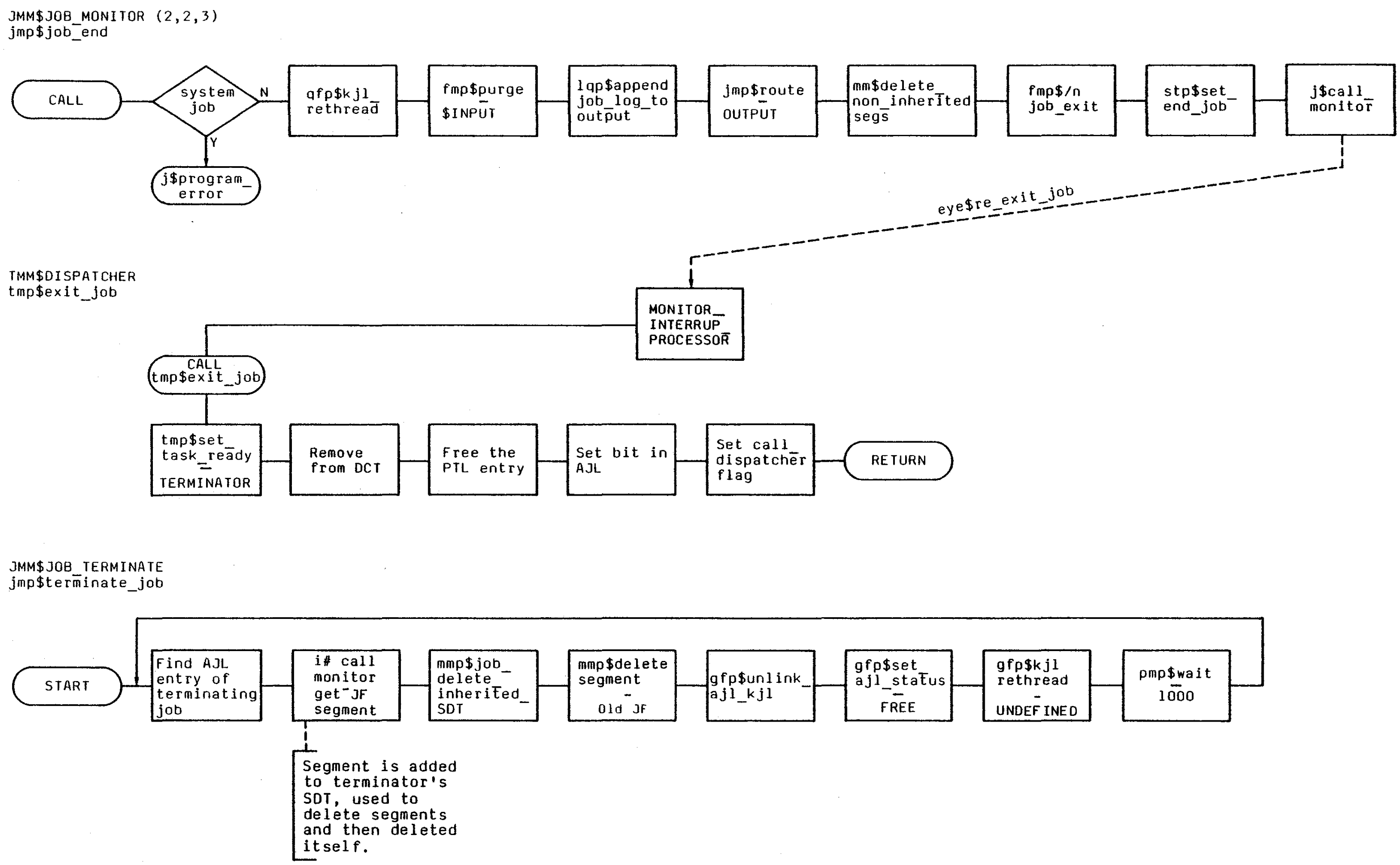
jmp_execute_system_job_task



JOB TERMINATION



JOB END



JMM$JOB_MONITOR (2,2,3)
jmp$job_end
CALL
system job
N
Y
j$program_ error
qfp$kjl_ rethread
fmp$purge
$INPUT
lqp$append job_log_to output
jmp$route
OUTPUT
mm$delete_ non_inherited segs
fmp$/n job_exit
stp$set_ end_job
j$call_ monitor
eye$re_exit_job
TMM$DISPATCHER
tmp$exit_job
MONITOR_ INTERRUP PROCESSOR
CALL tmp$exit_job
tmp$set_ task_ready
TERMINATOR
Remove from DCT
Free the PTL entry
Set bit in AJL
Set call_ dispatcher flag
RETURN
JMM$JOB_TERMINATE
jmp$terminate_job
START
Find AJL entry of terminating job
i# call monitor get JF segment
mmp$job_ delete inherited_ SDT
mmp$delete segment
Old JF
gfp$unlink_ ajl_kjl
gfp$set_ ajl_status
FREE
gfp$kjl rethread
UNDEFINED
pmp$wait
1000
Segment is added to terminator's SDT, used to delete segments and then deleted itself.

LESSON 9
PROGRAM EXECUTION

LESSON PREVIEW

TASK INITIATION
SYNCHRONOUS AND ASYNCHRONOUS EXECUTION
JOB LOCAL QUEUES
DEBUGGER
LOADER

OBJECTIVES

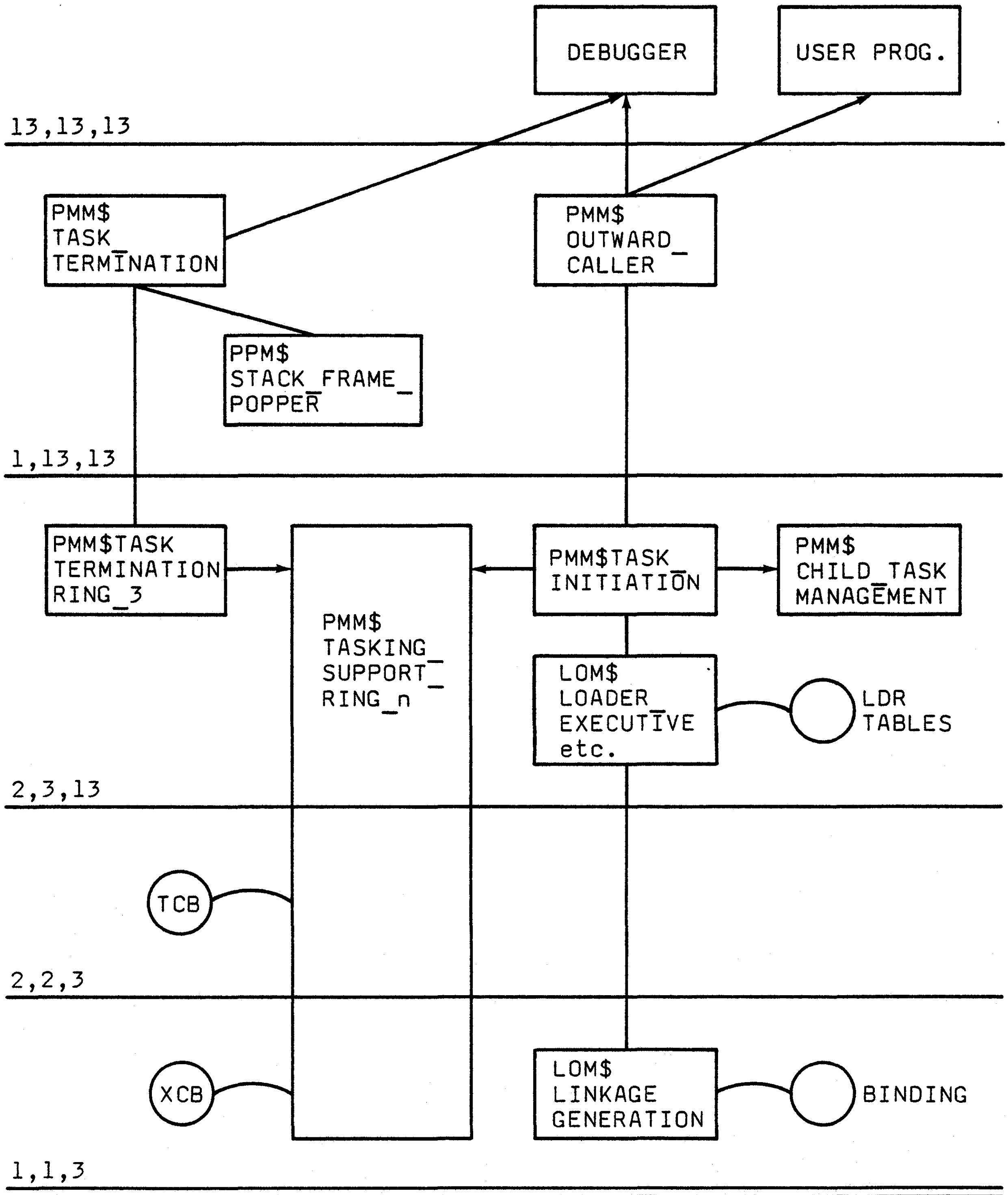
After completing this lesson the student should be able to--

- OVERVIEW THE MODULES AND TABLES THAT CONTROL TASK INITIATION AND EXECUTION
- EXPLAIN THE LINKAGE AND HANDLING OF THE TCB AND XCB
- DESCRIBE THE STRUCTURE OF OBJECT MODULES AND OBJECT LIBRARIES
- OUTLINE THE PROCESSING OF THE CATEGORIES OF CONDITIONS

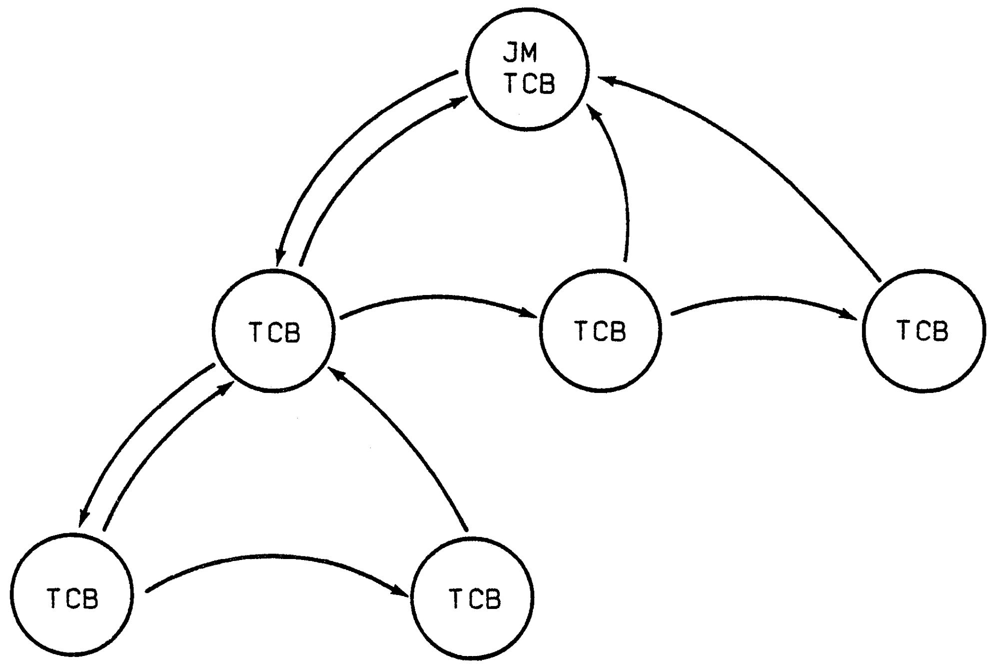
EXERCISE

NONE

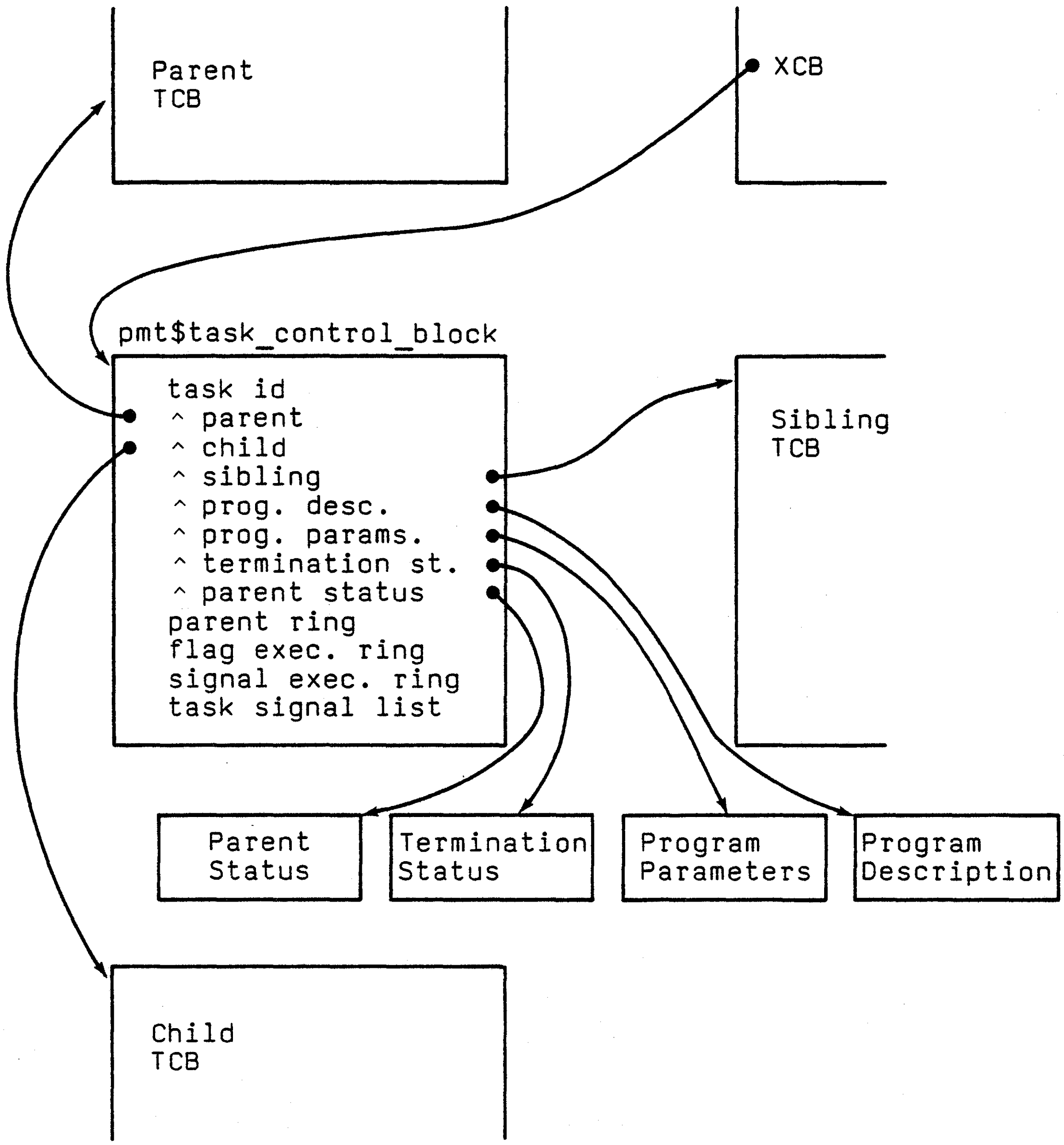
PROGRAM CONTROL AND LOADER



PARENT/CHILD/SIBLING



TASK CONTROL BLOCK



TASK WAIT

PMP$CYCLE (status)	Task waits till the next cycle of the dispatcher.
PMP$DELAY (ms,status)	Task waits ms milliseconds.
PMP$WAIT (ms)	Task waits for signal, flag, PMP$READY_TASK or ms milliseconds.
PMP$READY_TASK (task,status)	Cause a waiting task to be made ready for execution.

TASK IDENTIFICATION

PMP$FIND_EXECUTING_TASK_XCB (xcb)

PMP$FIND_TASK_XCB (tid,xcb)

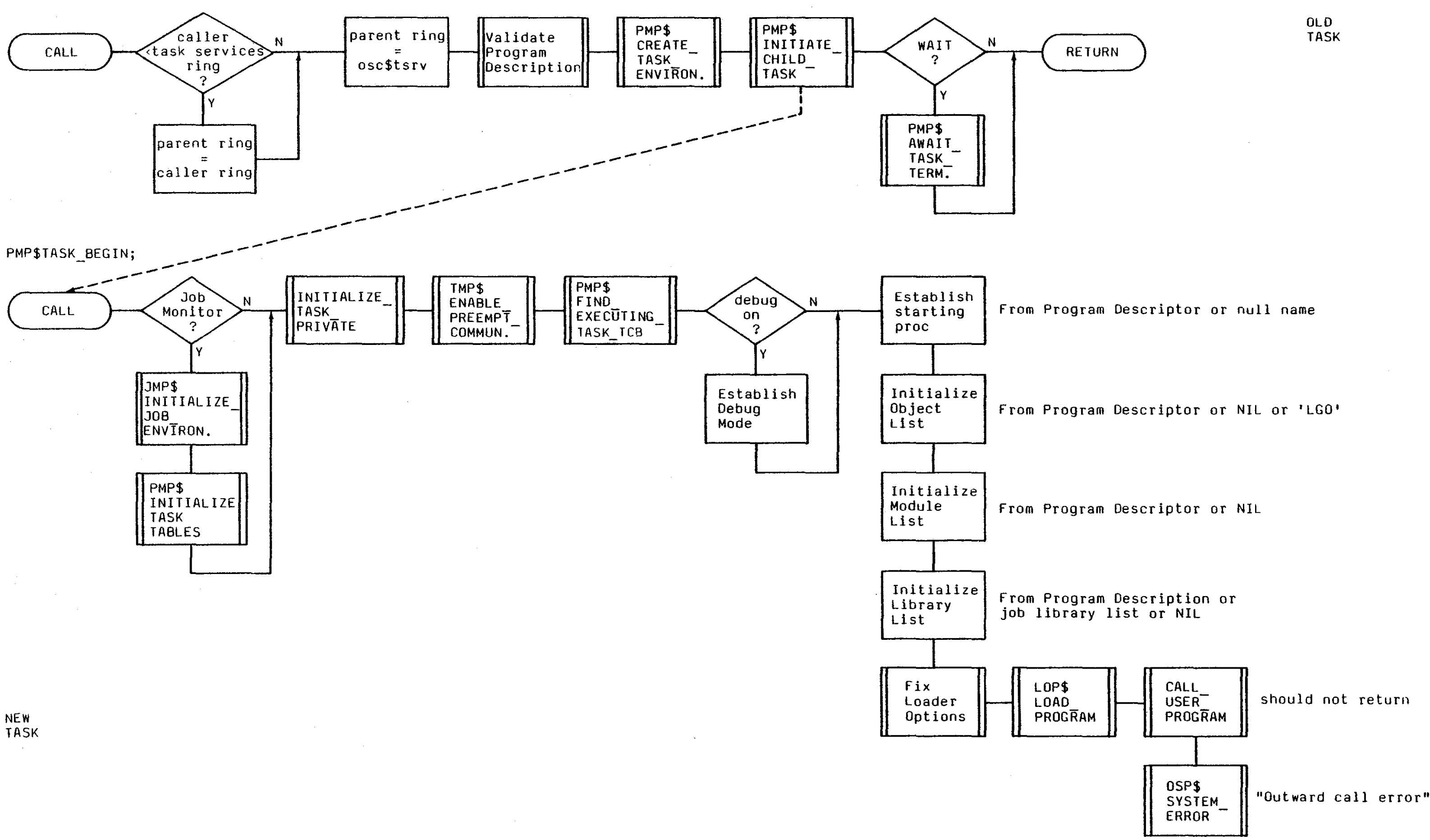
PMP$GET_EXECUTING_TASK_GTID (gtid)

PMP$GET_GLOBAL_TASK_ID (tid,gtid,status)

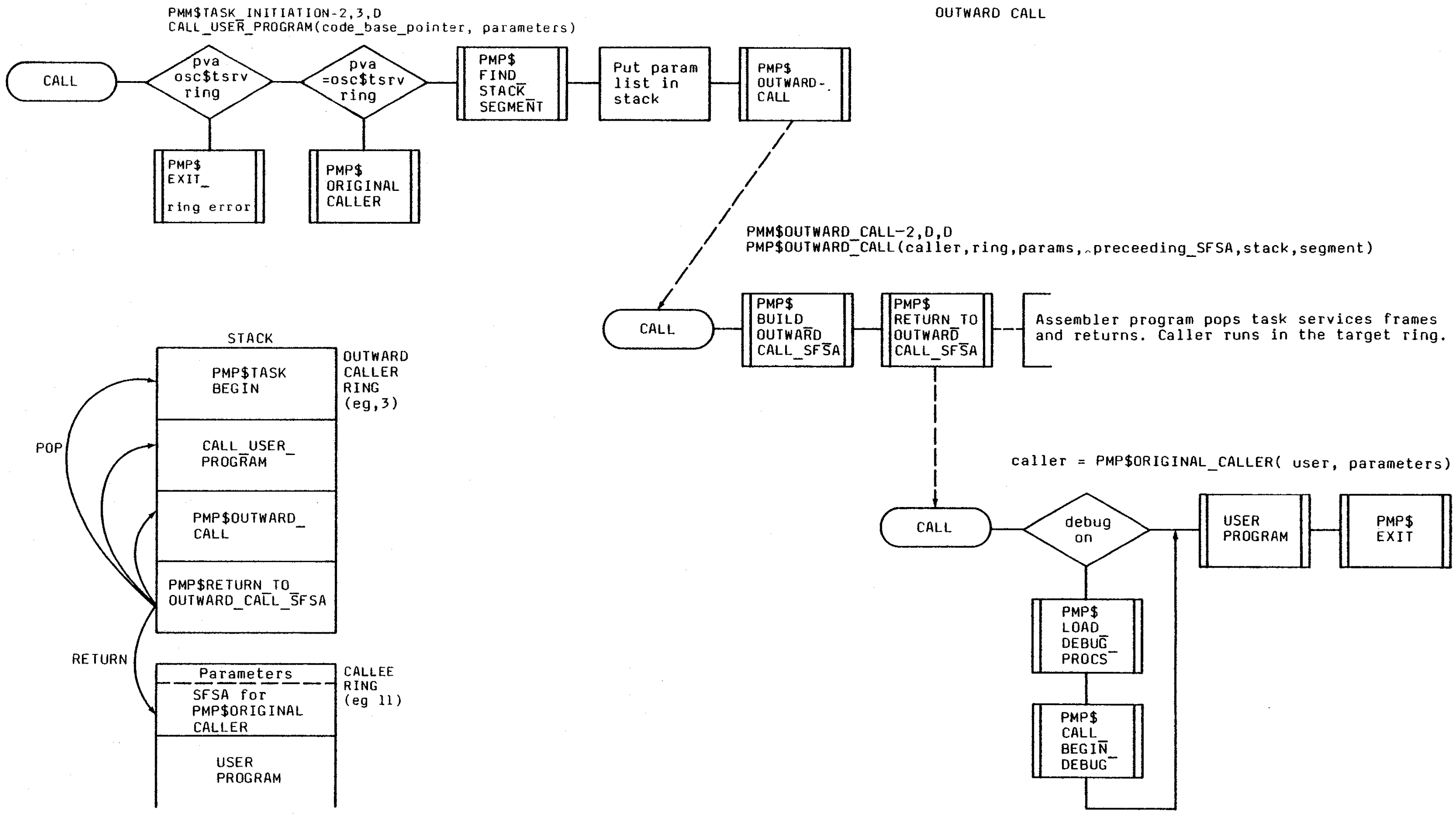
PMP$TASK_STATE

TASK INITIATION

PMM$TASK_INITIATION-2,3,D
PMP$EXECUTE(prog_desc,prog_parameters,wait,task_id,task_status,status);



TASK INITIATION (Continued)



TASK TERMINATION LEVELS

1. Unwinding

   - Revoke program termination (Debugger)
   - Pop stock frames--block exit processing
   - Close files at each 'active ring' to ring 3
   - Child Task Cleanup
     - Abnormal--kill all child tasks
     - Normal--await child termination
   - Clean up task environment

2. Unwinding Impossible

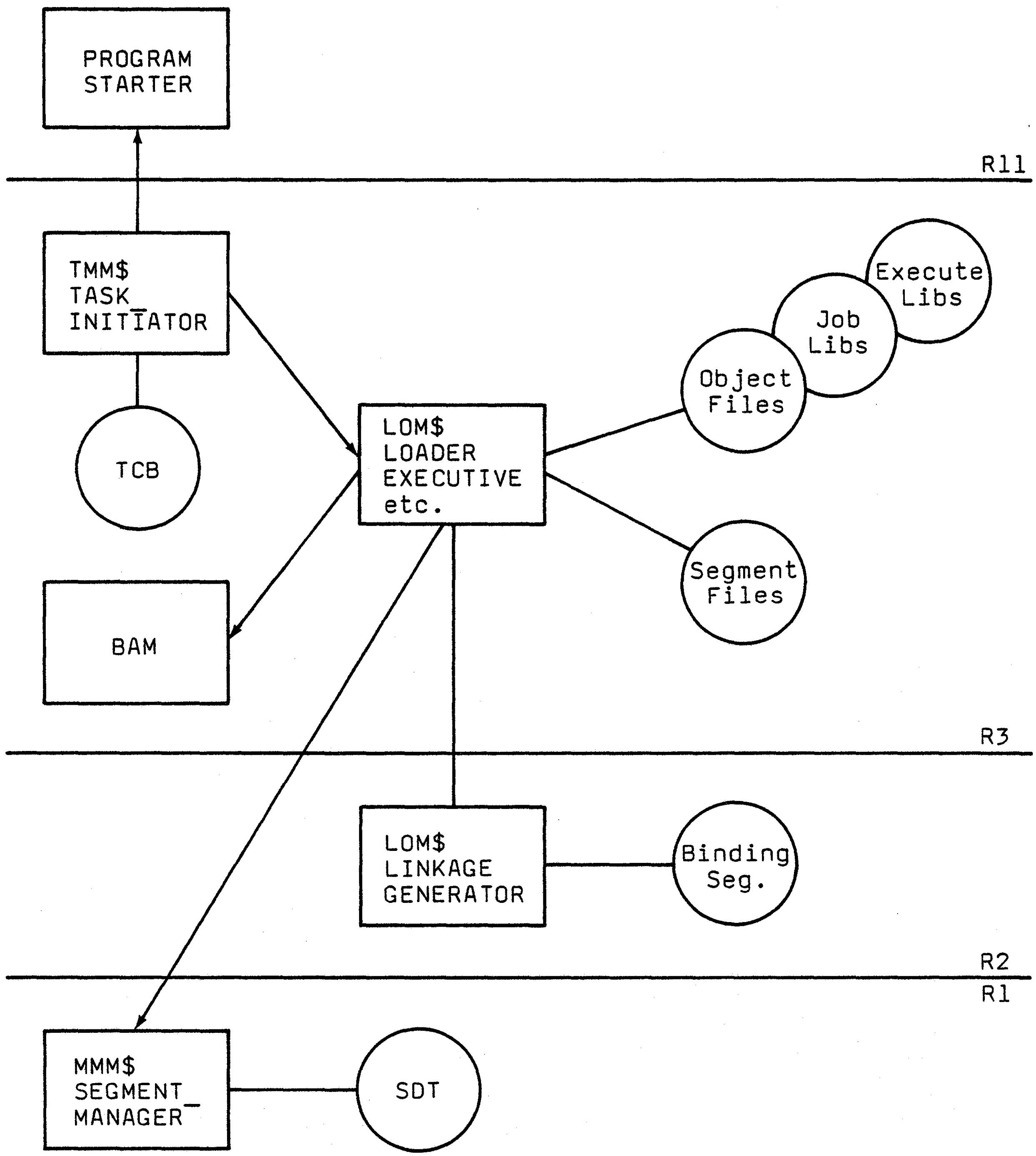
   - Stack, for example, is bad
   - Child task cleanup
   - Clean up task environment

3. Broken Task

   - Monitor detects monitor fault with traps disabled
   - Fix trap handler tables to see broken task flag

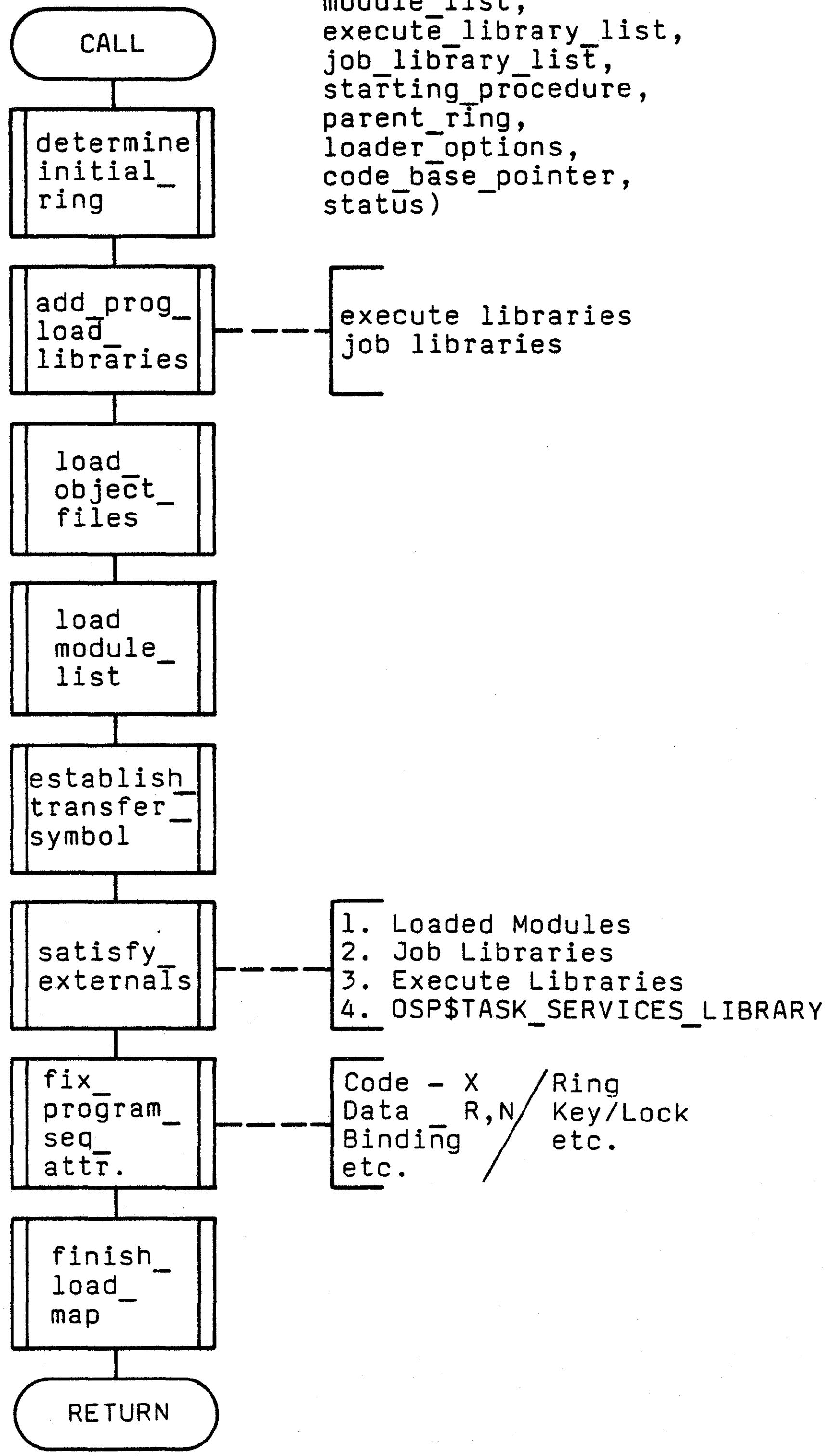
4. Monitor Kill.

LOADER CONTEXT



LOADER EXECUTIVE

LOM$LOADER_EXECUTIVE
LOP$LOAD_PROGRAM (object_file_list,
module_list,
execute_library_list,
job_library_list,
starting_procedure,
parent_ring,
loader_options,
code_base_pointer,
status)



LOADER OPTIONS

PMP$CHANGE_DEFAULT_PROG_OPTIONS (change,status)

PMP$GET_DEFAULT_PROGRAM_OPTIONS (options)

PMP$CHANGE_JOB_LIBRARY_LIST (change,status)

PMP$GET_NUMBER_OF_JOB_LIBRARIES (n)

PMP$GET_JOB_LIBRARY_LIST (list,status)

OBJECT MODULE INTERNAL FORMAT

* Each object module is a set of records on the object file

* The object record descriptor contains

  - Item type

  - Record length

* Item types

  IDR: Identification of module and attributes

  LIB: Libraries from which to satisfy external references

  SDC: Length and attributes of each section, code, working storage, binding, and all common blocks

  TEX: Text to be placed in each section

  RPL: Text to be repetitively placed in each section

  BIT: Inserts bit-level data into a section

  EPT: Defines an address in a section as an entry point

  RIF: Identifies addresses that must be relocated by the library generator when binding modules together

  ADR: Allows PVAs to be built at load time (when ring, segment number, and offset are known)

  XRL: List of external references to be satisfied

  BTI: Binding template describes the contents of a location in the binding section

  TRA: Terminates the object module and gives the primary entry point

USER
COMMAND
STREAM
(VALIDATED FOR
RING 11)

•
•
•

FTN,I=MAIN,B=LGO
FTN,I=SUB,B=LGO
LGO

•
•
•

LOCAL FILE LGO R1=11, R2=11, R3=11

IDR	• NAME • TIME & DATE CREATED • ETC.
LIB	• FTNLIB
SDC	CODE SECTION
SDC	BINDING SECTION
SDC	WORKING STORAGE SECTION
SDC	COMMON BLOCKS
TEX, RPL BIT, REL ADR, XRL EPT, BIN RECORDS FOR CODE, BINDING AND WORKING STORAGE SECTIONS	
TRA	• STARTING ADDRESS • END OF MODULE
IDR	
LIB	• FTNLIB
SDC	• CODE
SDC	• BINDING
SDC	• WORKING STORAGE
SDC	• COMMON BLOCKS
TEX, RPL BIT, REL ADR, XRL EPT, BIN RECORDS FOR CODE BINDING AND WORKING STORAGE SECTIONS	
TRA	

OBJECT
MODULE
FOR
MAIN

OBJECT
MODULE
FOR
SUB

OBJECT LIST (1)

```
MODULE SQACI;

  CONST
    MIN = 1,
    MAX = 30;

  TYPE
    T_ARRAY = ARRAY [MIN .. MAX] OF INTEGER;

  PROCEDURE SQUARER (B: T_ARRAY;
    VAR SQ: T_ARRAY);

    VAR
      J: MIN .. MAX;

    FOR J := MIN TO MAX DO
      SQ [J] := B [J] * B [J];
    FOREND;
  PROCEND SQUARER;

  PROGRAM MAIN;

    VAR
      BASE: T_ARRAY,
      SQUARE: T_ARRAY,
      I: MIN .. MAX;

    FOR I := MIN TO MAX DO
      BASE [I] := I;
    FOREND;
    SQUARER (BASE, SQUARE);
  PROCEND MAIN;
MODEND SQACI;
```

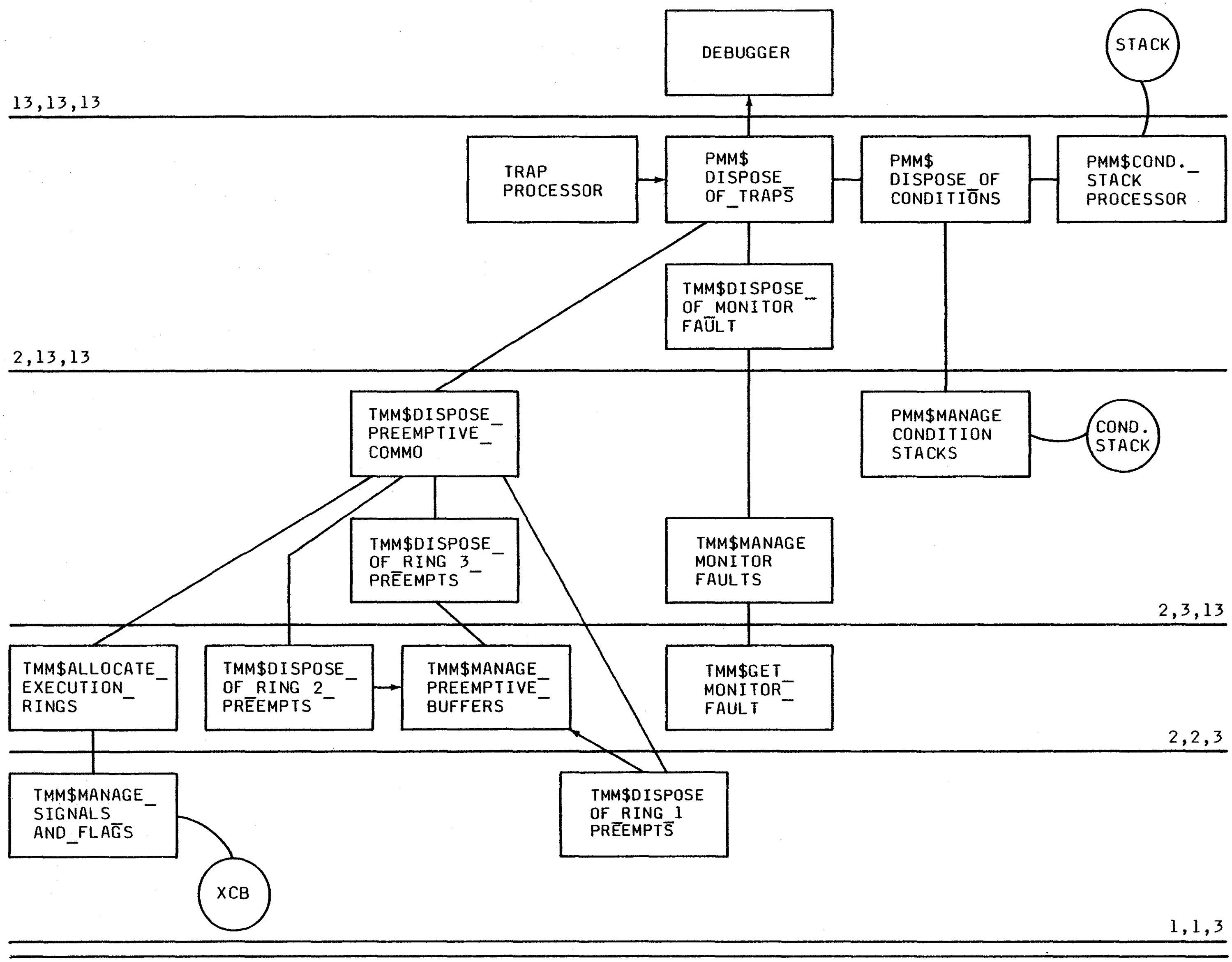
OBJECT LIST (2)

```
/ses.cybil sqaci b*bbb ci
*    COMPILING  SQACI
*    END CYBIL     SQACI -> LISTING, BBB
/ses.objlist bbb
1IDR  RN= 1  SQACI                          V1.2 MVS 10:02:14 10/13/81
      GREATEST SECTION ORD= 2  GEN ID=CYBIL
      GEN NAME VERS=C180 CYBIL 1.0          LEVEL 81188
      COMMENTARY=

 LIB  RN= 2  CYBILIB

 SDC  RN= 3  KIND=CODE     ATTRIBUTES=RX ORDINAL= 0 LENGTH=00000118
      OFFSET= 0 ALIGNMENT= 8

 SDC  RN= 4  KIND=BINDING  ATTRIBUTES=RB ORDINAL= 1 LENGTH=00000020
      OFFSET= 0 ALIGNMENT= 8

 SDC  RN= 5  KIND=WORKING  ATTRIBUTES=R ORDINAL= 2 LENGTH=0000001F
      OFFSET= 0 ALIGNMENT= 8

 EPT  RN= 6  SECTION= 0  OFFSET=000000B8  ATTRIBUTES=
        NAME=MAIN                            LANGUAGE=CYBIL
      DECLARATION MATCHING:  REQUIRED-TRUE VALUE-0A9A10EC520777

 BTI  RN= 7  BINDING OFFSET=0000000A  CURRENT MODULE
        SECTION= 2  OFFSET=00000000  KIND=POINTER

 ADR  RN= 8  VALUE SECTION= 2  DEST. SECTION= 1
      KIND=POINTER   VALUE OFFSET=00000000  DEST. OFFSET=0000000A

 BTI  RN= 9  BINDING OFFSET=00000010  EXTERNAL REF
        NAME=CYP$ERROR                       ADDRESS=EXT PROC

 EXT  RN= 10  NAME=CYP$ERROR                         LANGUAGE=CYBIL
      DECLARATION MATCHING:  REQUIRED-FALSE VALUE-00000000000000
      SECTION ORDINAL= 1 OFFSET=00000010  KIND=EXT PROC  OFFSET OPERAND=00000000
```

OBJECT LIST (3)

```
RIF   RN= 11   SECTION= 0 OFFSET=000000A6 R-SECTION= 1 CONTAINER=Q FIELD  ADDRESS=BYTE-

RIF   RN= 12   SECTION= 0 OFFSET=000000B2 R-SECTION= 1 CONTAINER=Q FIELD  ADDRESS=WORD-

TEX   RN= 13   LENGTH=001F SECTION= 2 OFFSET=00000000
      53514143  49202020  20202020  20202020  20202020  20202020
      20202020  202020

TEX   RN= 14   LENGTH=00B4 SECTION= 0 OFFSET=00000000
      8E100020  84450000  09068E00  00F08516  00127656  09F00000
      09F00000  3D128D03  001E8D0E  00103D14  96420036  96230034
      83120003  D015001F  84450008  A9560003  8D5E0011  96450025
      96530023  D015001F  84160012  A9570003  9645001B  96530019
      D015001F  84170012  A9580003  96450011  9653000F  1187D765
      70001188  D7778000  26571186  DF576000  9C32FFCD  04323D2D
      94000003  3D3D0905  8E000018  835D0000  835E0001  8436000A
      85560012  B30000FF  B5350002

TEX   RN= 15   LENGTH=005A SECTION= 0 OFFSET=000000B8
      8E100208  3D128D03  001E8D0E  001C3D14  9642FFE6  9623FFE4
      8312003E  D01501F7  A9560003  8D5E001D  9645FFD7  9653FFD5
      D01501F7  DF156008  9C32FFE9  8E1501F8  8E160010  85560000
      8E160100  85560008  8D000020  B035FFDF  0435

TRA   RN= 16   NAME=MAIN
```

PREEMPTIVE COMMUNICATION AND CONDITIONS

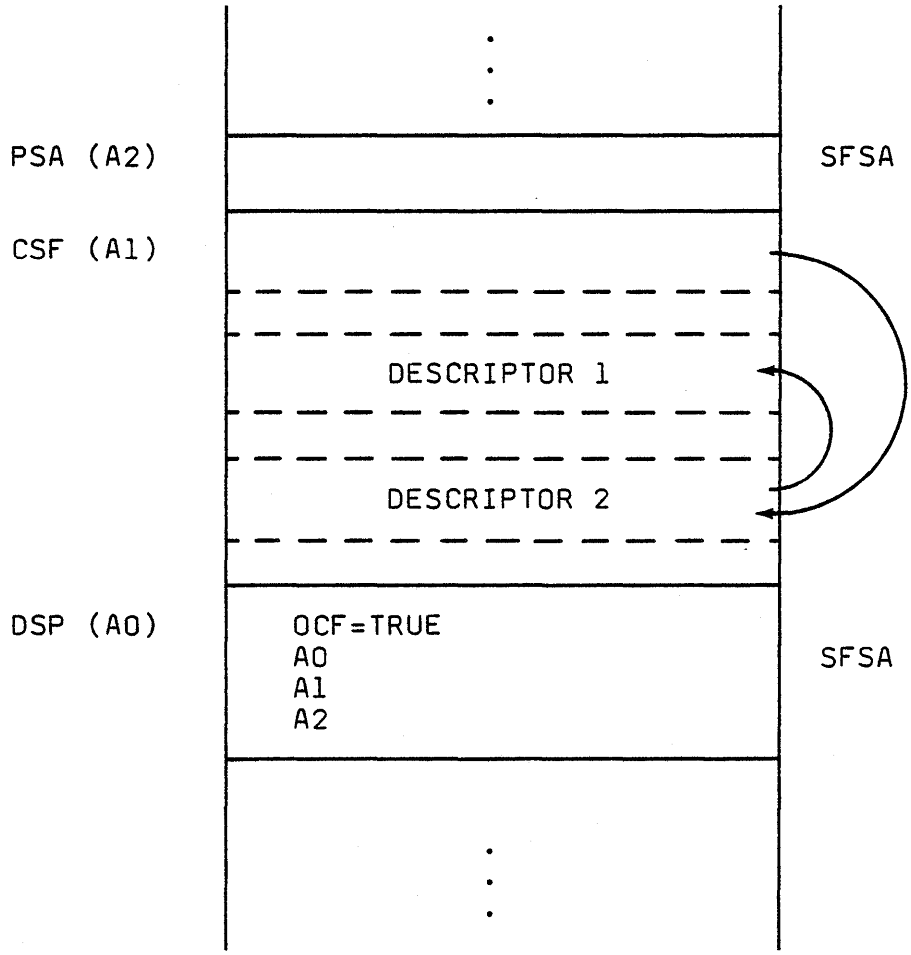


PMP$VERIFY_CURRENT_CHILD (tid,current)

PMP$SIGNAL_ALL_CHILD_TASKS (signal,status)

PMP$FLAG_ALL_CHILD_TASKS (flag,status)

PMP$REVOKE_PROGRAM_TERMINATION



PSA (A2)
SFSA
CSF (A1)
DESCRIPTOR 1
DESCRIPTOR 2
DSP (A0)
OCF=TRUE
A0
A1
A2
SFSA

CONDITIONS

	DESCRIPTION	SCOPE	INFO RETURNED
USER	Selector name ^ handler	Current Ring	Condition Condition Descriptor passed on PMP$CAUSE_CONDITION
INTERACTIVE	Selector id:0..255 ^ handler	All Rings	Condition
SYSTEM	Selector Set of MCR, UCR Loop Prevention ^ handler	Current Ring	Condition Save Area of frame that caused the condition
BLOCK	Selector Set of reason CFF ^ handler	Frame	Condition Save Area of frame attempting Return, Pop or non-local exit
JOB RESOURCE	Selector id:0..255 ^ handler	All Rings	Condition
SEGMENT ACCESS	Selector id:0..255 Segment Number Loop Prevention ^ handler	Current Ring	Condition Save Area of frame that caused the condition
COMBINATION	Selector Set of category ^ handler	-	-

NOTE: See Program Interface

LESSON 10
SCL INTERPRETER

LESSON PREVIEW

COMMAND VS PROGRAM INTERFACE
LOGIN, LOGOUT PROCESSING
COMMAND SEARCH
COMMAND PROCESSING
SUB COMMANDS
PROLOG AND EPILOG PROCESSING

OBJECTIVES

After completing this lesson the student should be able to--

- ADD A COMMAND PROCESSOR THAT WILL RUN IN THE CURRENT TASK; IN A NEW TASK

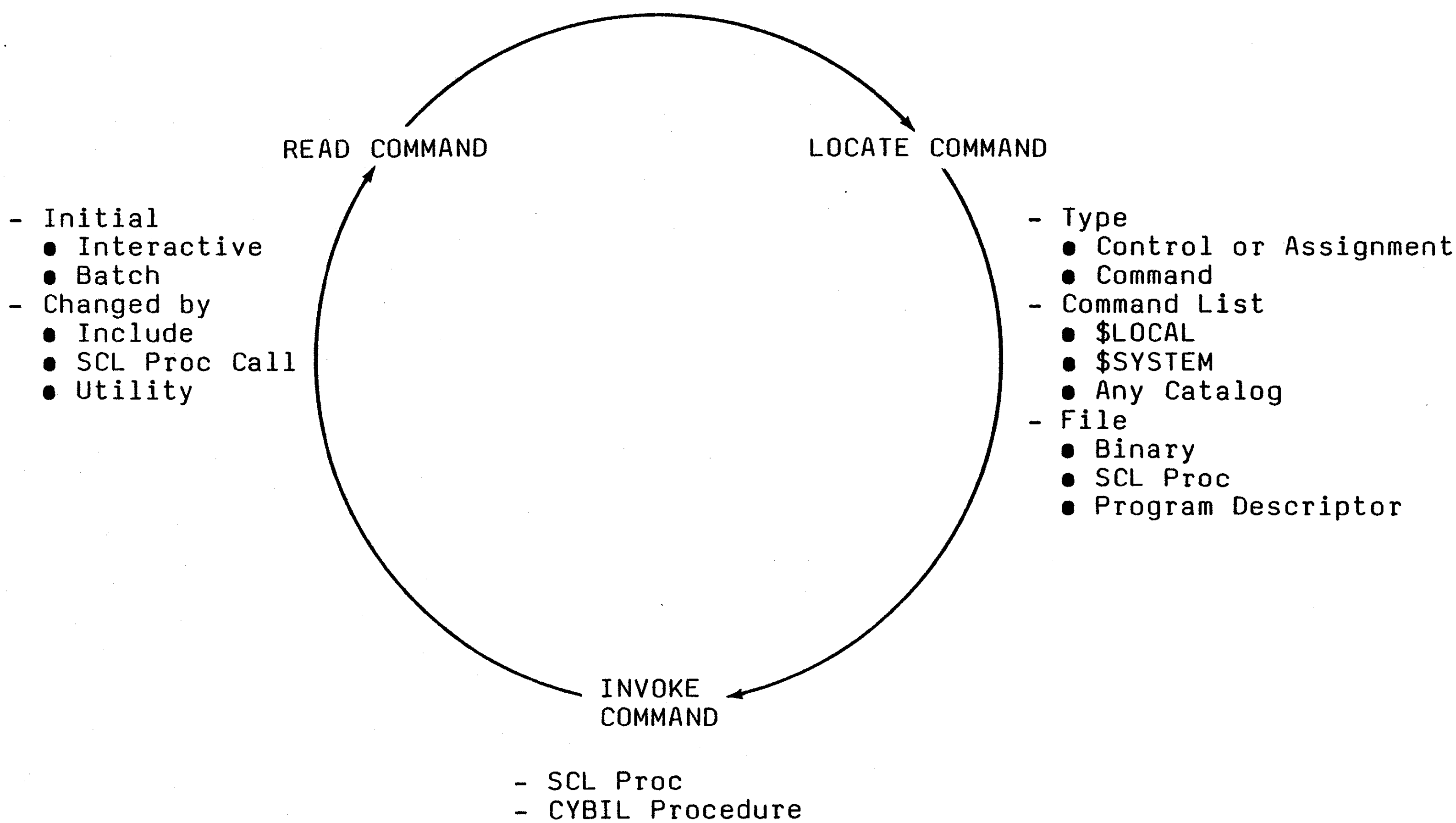
- EXPLAIN THE USE OF THE BLOCK STACK AND THE INPUT STACK TO CONTROL THE PROCESSING OF COMMANDS

- OUTLINE THE PROCESSING OF LOGIN AND LOGOUT

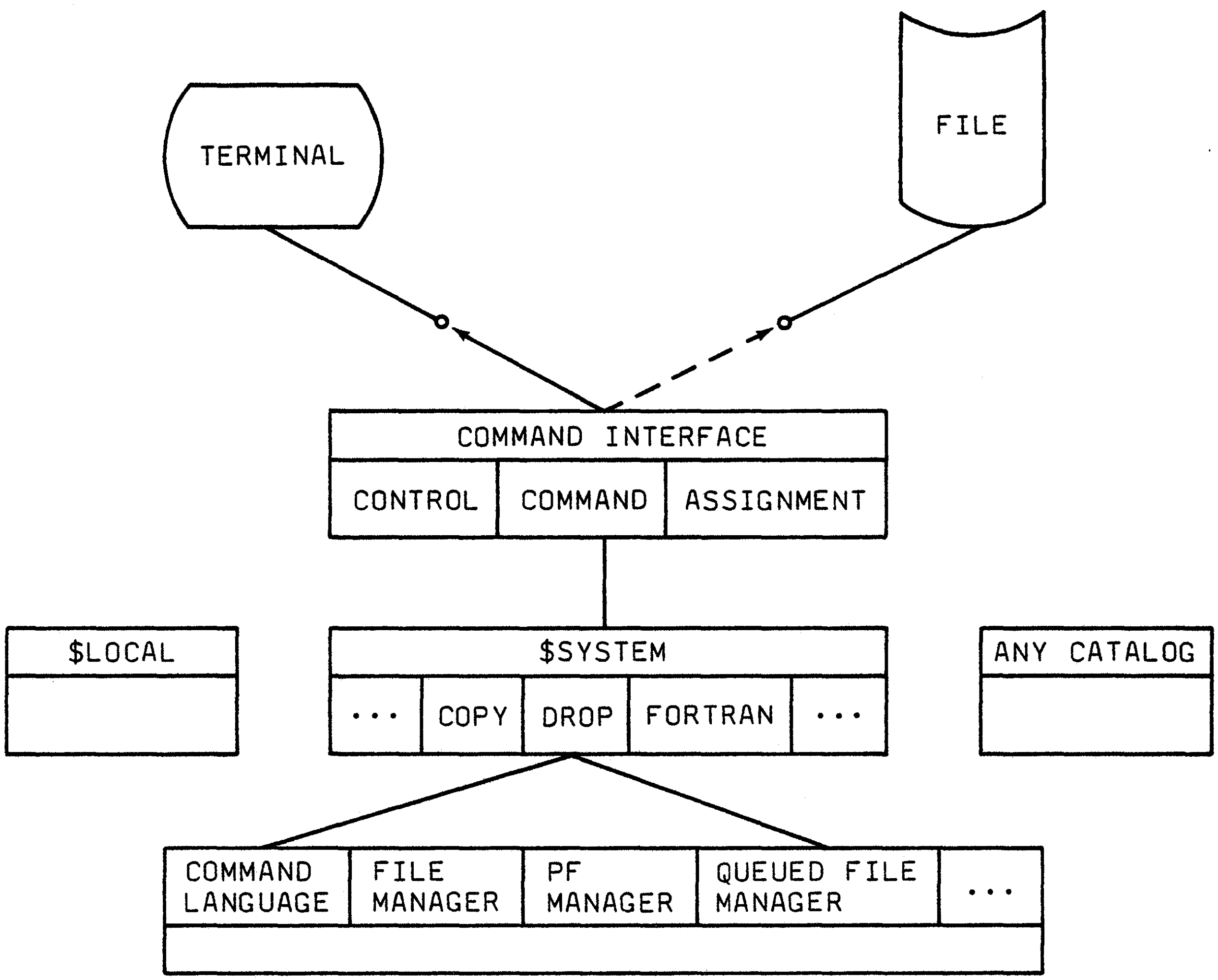
EXERCISE

ADD A COMMAND PROCESSOR TO THE SYSTEM

CLI BASIC LOOP



COMMAND PROCESSING

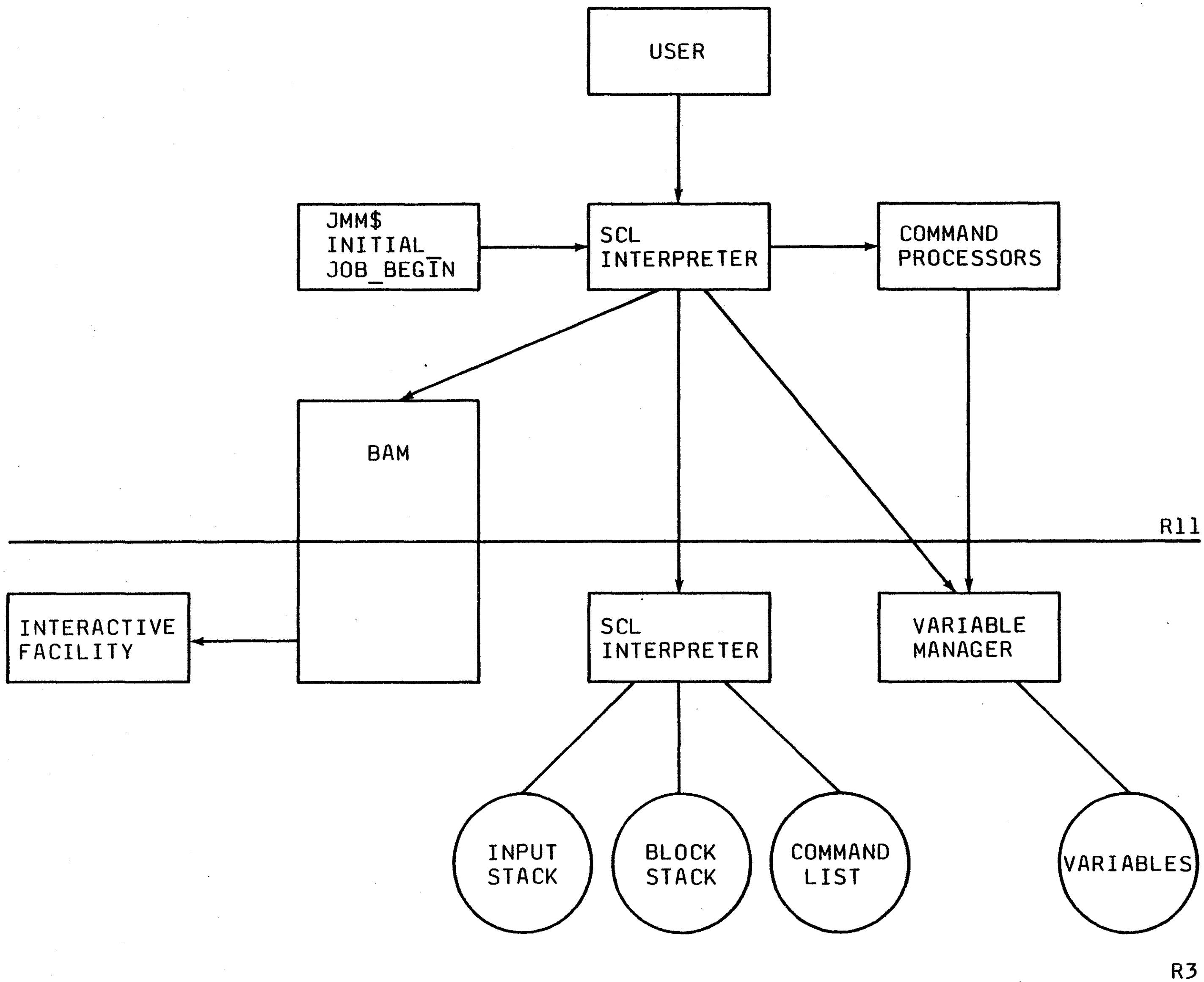


```
{ PDT copy_pdt(
{ from:FILEREF = $required
{ to:FILEREF = $output
{ count:INTEGER 0..amc$file_byte_limit = 1
{ unit:KEY file,partition,record
{ status)

  VAR
    copy_pdt: [STATIC, READ, cls$pdt] clt$parameter_descriptor_table := [^copy_pdt_names, ^copy_pdt_params];

  VAR
    copy_pdt_names: [STATIC, READ, cls$pdt] array [1 .. 5] of clt$parameter_name_descriptor := [['FROM', 1],
      ['TO', 2], ['COUNT', 3], ['UNIT', 4], ['STATUS', 5]];

  VAR
    copy_pdt_params: [STATIC, READ, cls$pdt] array [1 .. 5] of clt$parameter_descriptor := [

{ FROM }
    [[clc$required], 1, 1, 1, 1, clc$value_range_not_allowed, [NIL, clc$file_value, clc$position_allowed]],

{ TO }
    [[clc$optional_with_default, ^copy_pdt_dv2], 1, 1, 1, 1, clc$value_range_not_allowed, [NIL,
      clc$file_value, clc$position_allowed]],

{ COUNT }
    [[clc$optional_with_default, ^copy_pdt_dv3], 1, 1, 1, 1, clc$value_range_not_allowed, [NIL,
      clc$integer_value, 0, amc$file_byte_limit]],

{ UNIT }
    [[clc$optional], 1, 1, 1, 1, clc$value_range_not_allowed, [^copy_pdt_kv4, clc$keyword_value]],

{ STATUS }
    [[clc$optional], 1, 1, 1, 1, clc$value_range_not_allowed, [NIL, clc$variable_reference,
      clc$array_not_allowed, clc$status_value]]];

  VAR
    copy_pdt_kv4: [STATIC, READ, cls$pdt] array [1 .. 3] of ost$name := ['FILE', 'PARTITION', 'RECORD'];

  VAR
    copy_pdt_dv2: [STATIC, READ, cls$pdt] string (7) := '$output';

  VAR
    copy_pdt_dv3: [STATIC, READ, cls$pdt] string (1) := '1';
```

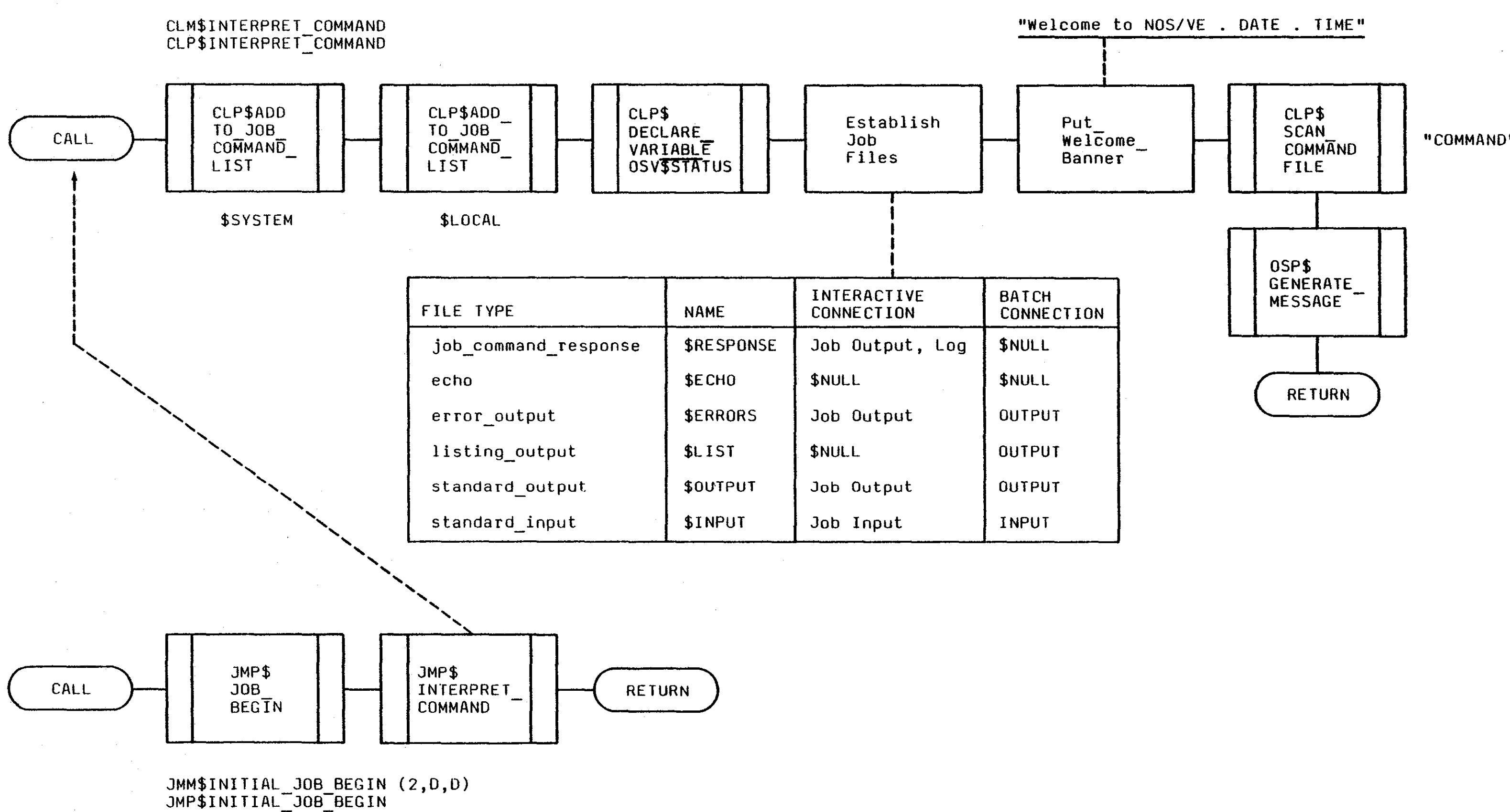
```
PROCEDURE [XDCL] clp$copy_command (parameter_list: clt$parameter_list;
  VAR status: ost$status);

  VAR
    count_specified: boolean,
    unit_specified: boolean,
    from_value: clt$value,
    to_value: clt$value,
    set_attributes: array [1 .. 1] of amt$file_item;

  clp$scan_parameter_list (parameter_list, copy_pdt, status);
  IF NOT status.normal THEN
    RETURN;
  IFEND;

  clp$test_parameter ('UNIT', unit_specified, status);
  IF NOT status.normal THEN
    RETURN;
  IFEND;
  IF unit_specified THEN
    osp$set_status_abnormal ('CL', cle$not_yet_implemented, 'UNIT parameter', status);
    RETURN;
  IFEND;

  clp$test_parameter ('COUNT', count_specified, status);
  IF NOT status.normal THEN
    RETURN;
  IFEND;
  IF count_specified THEN
    osp$set_status_abnormal ('CL', cle$not_yet_implemented, 'COUNT parameter', status);
    RETURN;
  IFEND;
```

```
    clp$get_value ('FROM', 1, 1, clc$low, from_value, status);
    IF NOT status.normal THEN
      RETURN;
    IFEND;
    IF from_value.file.open_position.specified THEN
      set_attributes [1].key := amc$open_position;
      set_attributes [1].open_position := from_value.file.open_position.value;
      amp$file (from_value.file.local_file_name, set_attributes, status);
      IF NOT status.normal THEN
        RETURN;
      IFEND;
    IFEND;

    clp$get_value ('TO', 1, 1, clc$low, to_value, status);
    IF NOT status.normal THEN
      RETURN;
    IFEND;
    IF to_value.file.open_position.specified THEN
      set_attributes [1].key := amc$open_position;
      set_attributes [1].open_position := to_value.file.open_position.value;
      amp$file (to_value.file.local_file_name, set_attributes, status);
      IF NOT status.normal THEN
        RETURN;
      IFEND;
    IFEND;

    amp$copy_file (from_value.file.local_file_name, to_value.file.local_file_name, status);
    IF NOT status.normal THEN
      RETURN;
    IFEND;

  PROCEND clp$copy_command;

MODEND clm$copy_command;
```

SCL CONTEXT


USER
JMM$ INITIAL_ JOB_BEGIN
SCL INTERPRETER
COMMAND PROCESSORS
BAM
R11
INTERACTIVE FACILITY
SCL INTERPRETER
VARIABLE MANAGER
INPUT STACK
BLOCK STACK
COMMAND LIST
VARIABLES
R3

INTERPRET COMMAND



FILE TYPE	NAME	INTERACTIVE CONNECTION	BATCH CONNECTION
job_command_response	$RESPONSE	Job Output, Log	$NULL
echo	$ECHO	$NULL	$NULL
error_output	$ERRORS	Job Output	OUTPUT
listing_output	$LIST	$NULL	OUTPUT
standard_output	$OUTPUT	Job Output	OUTPUT
standard_input	$INPUT	Job Input	INPUT

BLOCK STACK

THE PRIMARY PURPOSE OF THE BLOCK STACK IS TO MAINTAIN:

CURRENT PARAMETER VALUES (PVT)

COMMAND LANGUAGE VARIABLES

TASK LOCAL
PROC OR WHEN LOCAL

BLOCK STRUCTURE

INFORMATION FOR LOOPNG STATEMENTS

INPUT STACK

THE INPUT STACK IS USED TO MANAGE THE COMMAND STREAM KNOWN AS $COMMAND.

FILE NAME
BYTE ADDRESS OF CURRENT LINE
COMMAND LINE
LINE INDEX
FILE ID (PER TASK)

EXAMPLES:

```
/INCLUDE_FILE file = abc
/CREATE_OBJECT_LIBRARY
```

LESSON 11
PERMANENT FILES

LESSON PREVIEW

PF CAPABILITIES
PF TABLES
SETS

OBJECTIVES

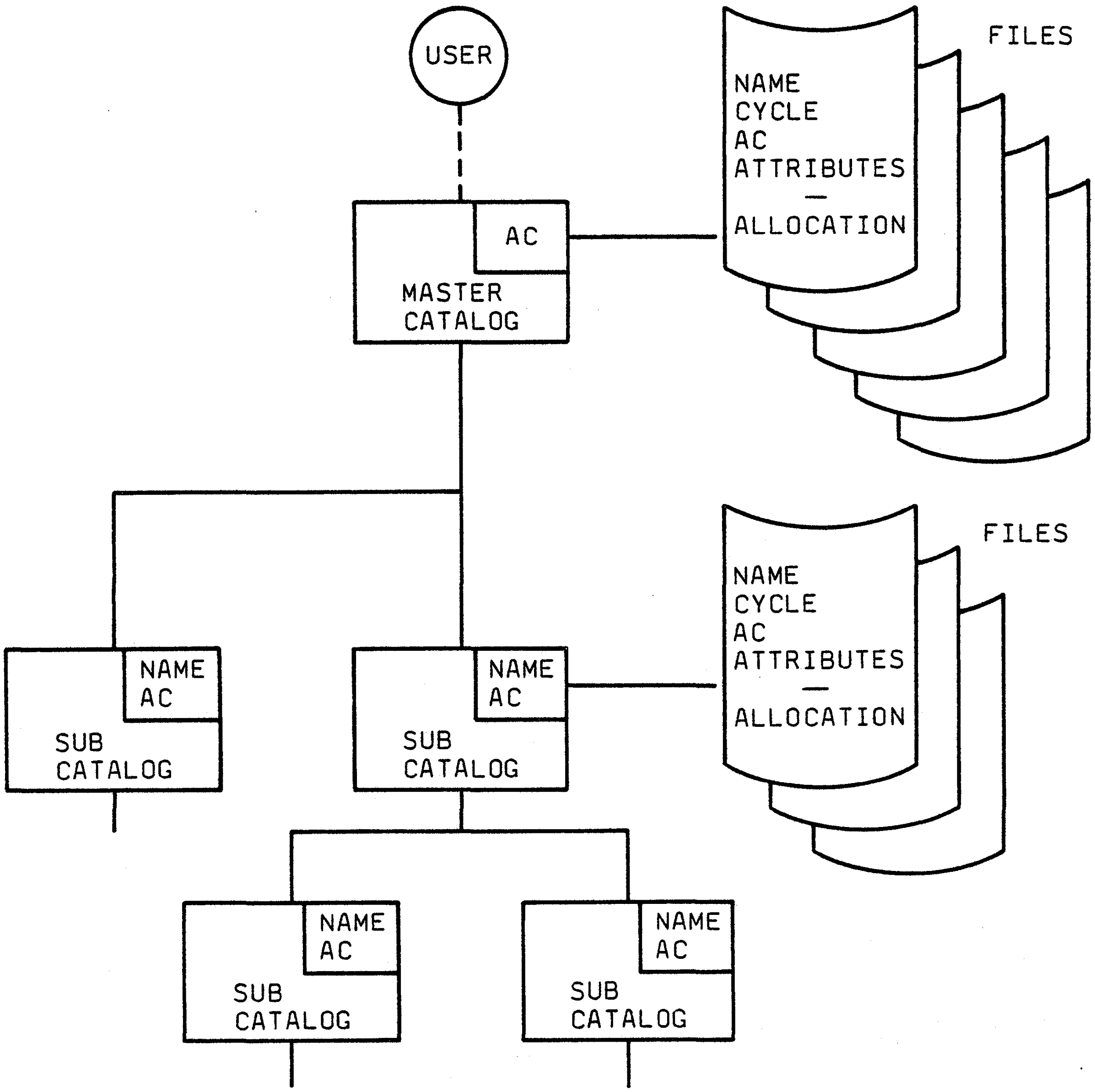
After completing this lesson the student should be able to--

- OUTLINE THE CAPABILITIES OF THE PF SYSTEM
- DESCRIBE THE RELATIONSHIPS BETWEEN PF MANAGEMENT AND OTHER FUNCTIONAL AREAS OF THE SYSTEM
- DESCRIBE THE CONTENTS AND LINKAGE OF THE PF TABLES

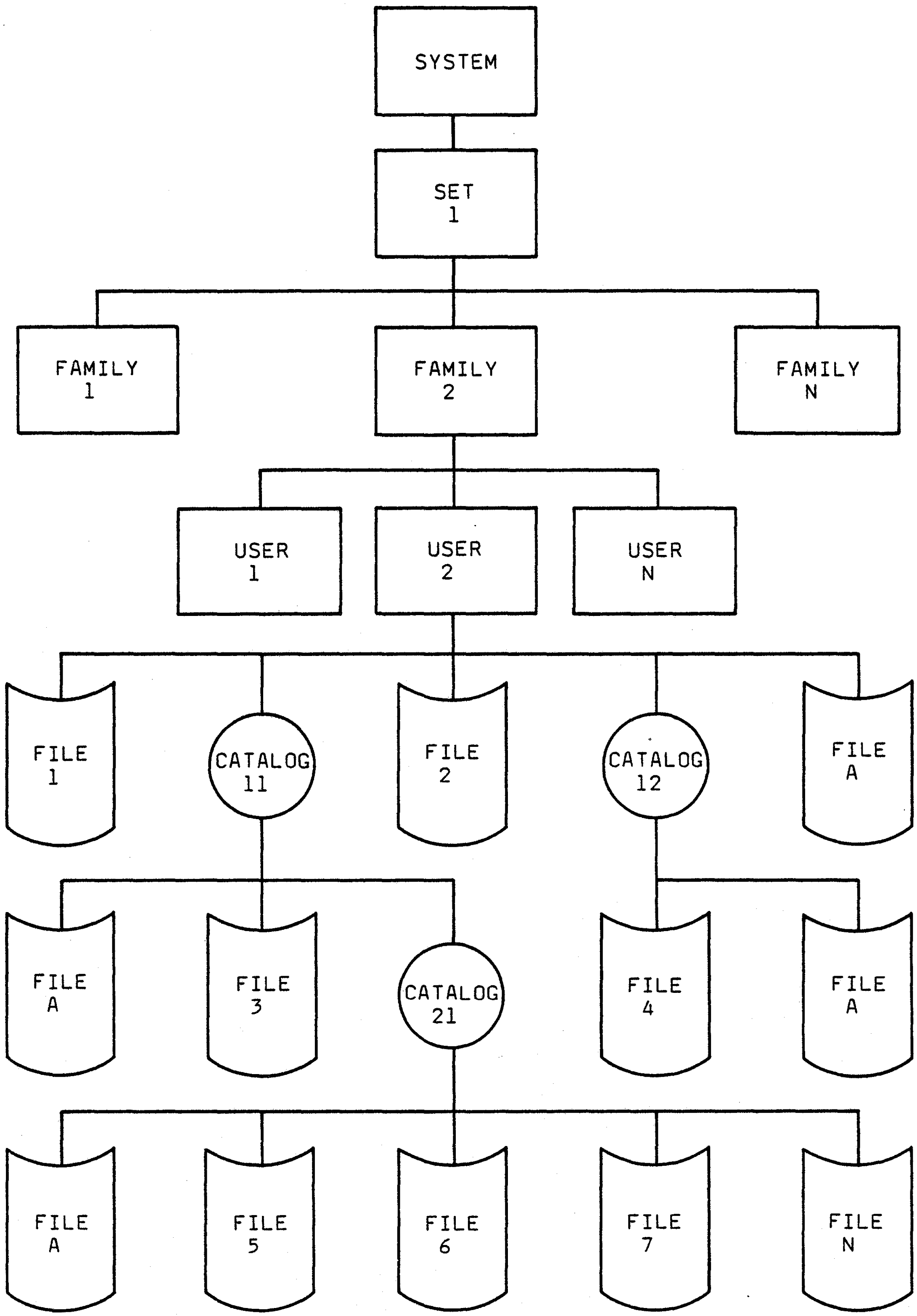
EXERCISE

NONE

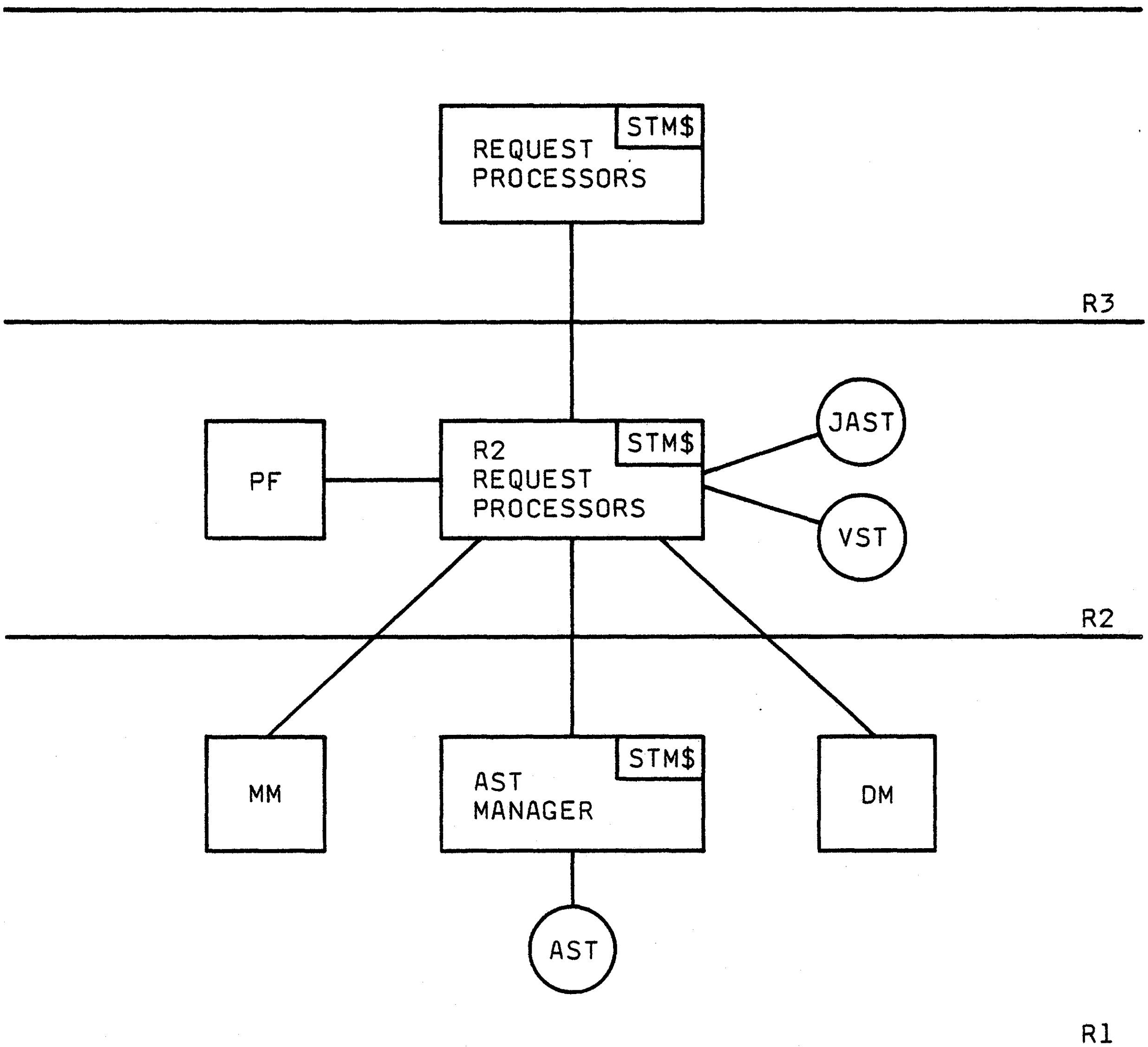
CATALOG/PERM FILE



CATALOG TREE STRUCTURE



SET MANAGER



STM$
REQUEST
PROCESSORS
R3
PF
STM$
R2
REQUEST
PROCESSORS
JAST
VST
R2
MM
STM$
AST
MANAGER
DM
AST
R1

SET MANAGER TABLES

AST stt$active_set_table STDAST	Name Master VSN ^ Member VSNs Set Owner Number of Jobs Using Set Root object list locator
JAST stt$job_active_set_table STDJAST	Name Set Owner Root object list locator

VST
stt$vol_set_table
STDVST
	VSN
	Name
	Member
	   Master VSN
	Master
	   Set Owner
	   Root object list locator
	   ^ Member VSNs
	   VST heap
	   Segment size fixer

SET INTERNAL INTERFACES

FROM OUTSIDE SET MGR.

STP$CREATE_SET

STP$ADD_MEMBER_VOL_TO_SET

STP$PURGE_SET

STP$REMOVE_MEMBER_VOL_FROM_SET

STP$ASSOCIATE_CATALOG

FROM WITHIN SET MGR.

STP$CREATE_VOL_SET_TABLE

STP$GET_ROOT_OBJECT_LOCATOR

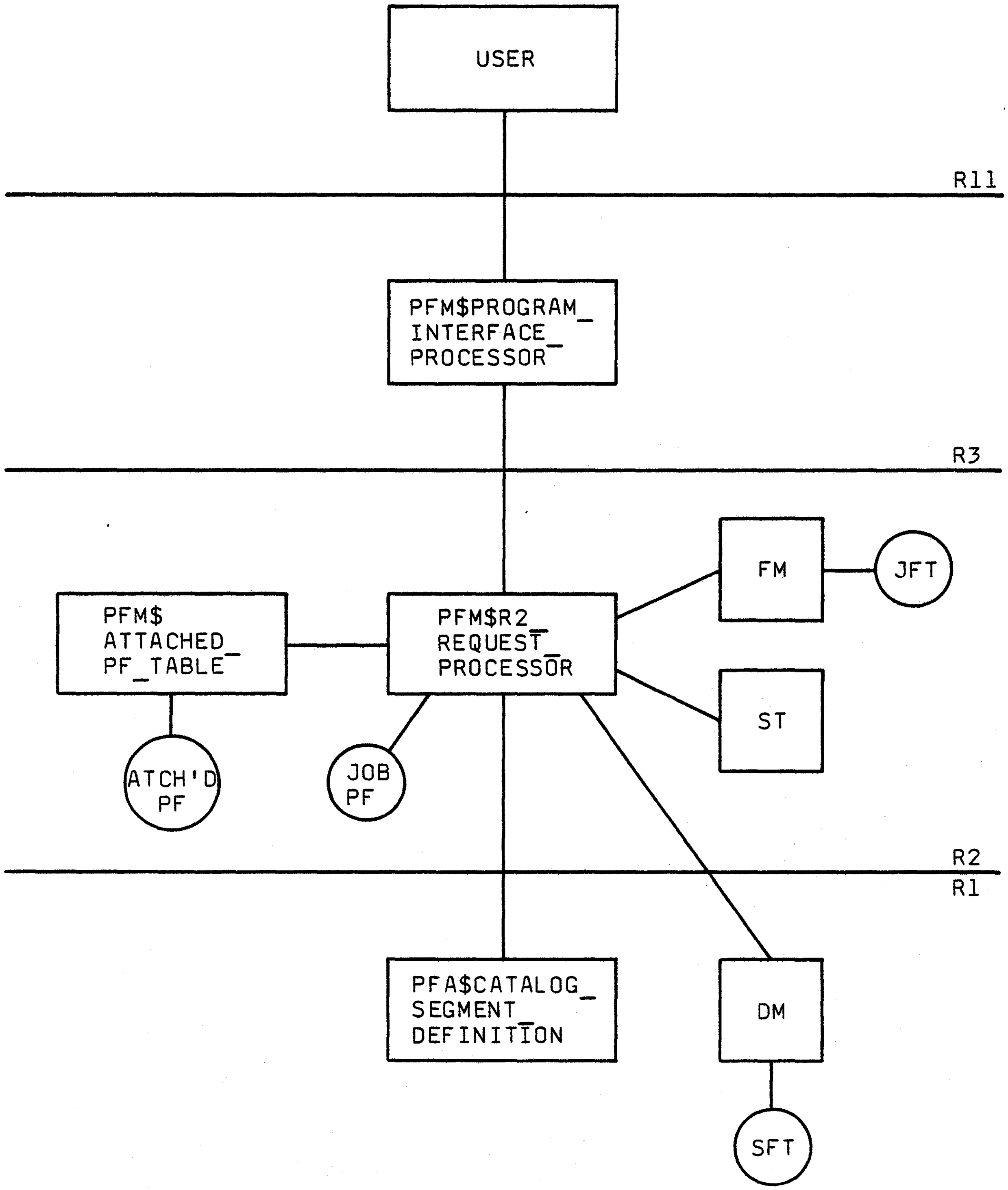
STP$GET_SET_OWNER

STP$CHECK_CATALOG_ASSOCIATION

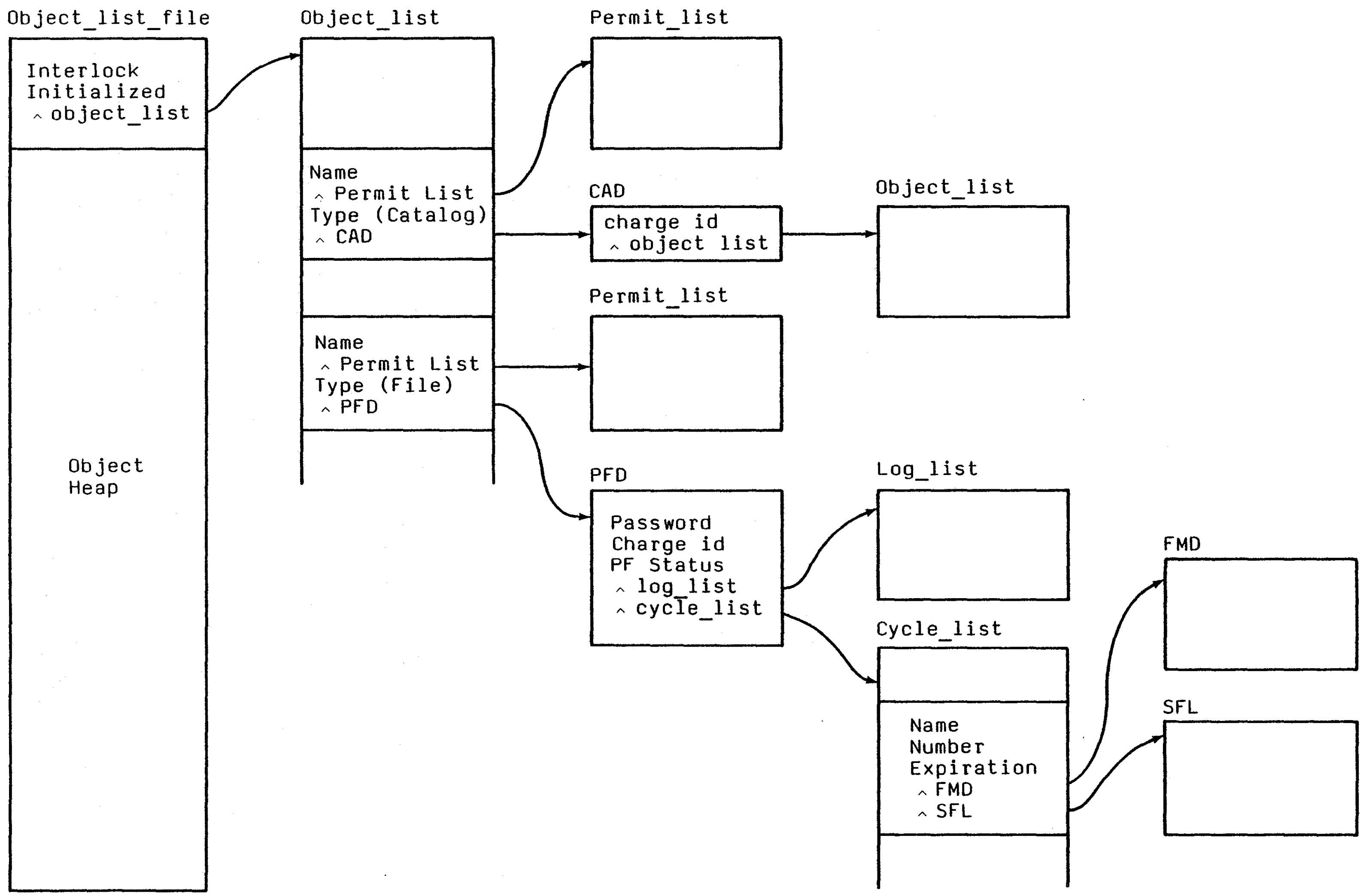
STP$CHANGE_ACCESS_TO_SET

STP$SET_END_JOB

PF MANAGER


USER
R11
PFM$PROGRAM_
INTERFACE_
PROCESSOR
R3
FM
JFT
PFM$
ATTACHED_
PF_TABLE
PFM$R2_
REQUEST_
PROCESSOR
ST
ATCH'D
PF
JOB
PF
R2
R1
PFA$CATALOG_
SEGMENT_
DEFINITION
DM
SFT

PF CATALOG STRUCTURE



LESSON 12
LOGICAL I/O

LESSON PREVIEW

OPEN/CLOSE
RECORD VS. SEGMENT LEVEL ACCESS
DEVICE MANAGEMENT
FAPS
FILE ATTRIBUTES
FILE TABLES

OBJECTIVES

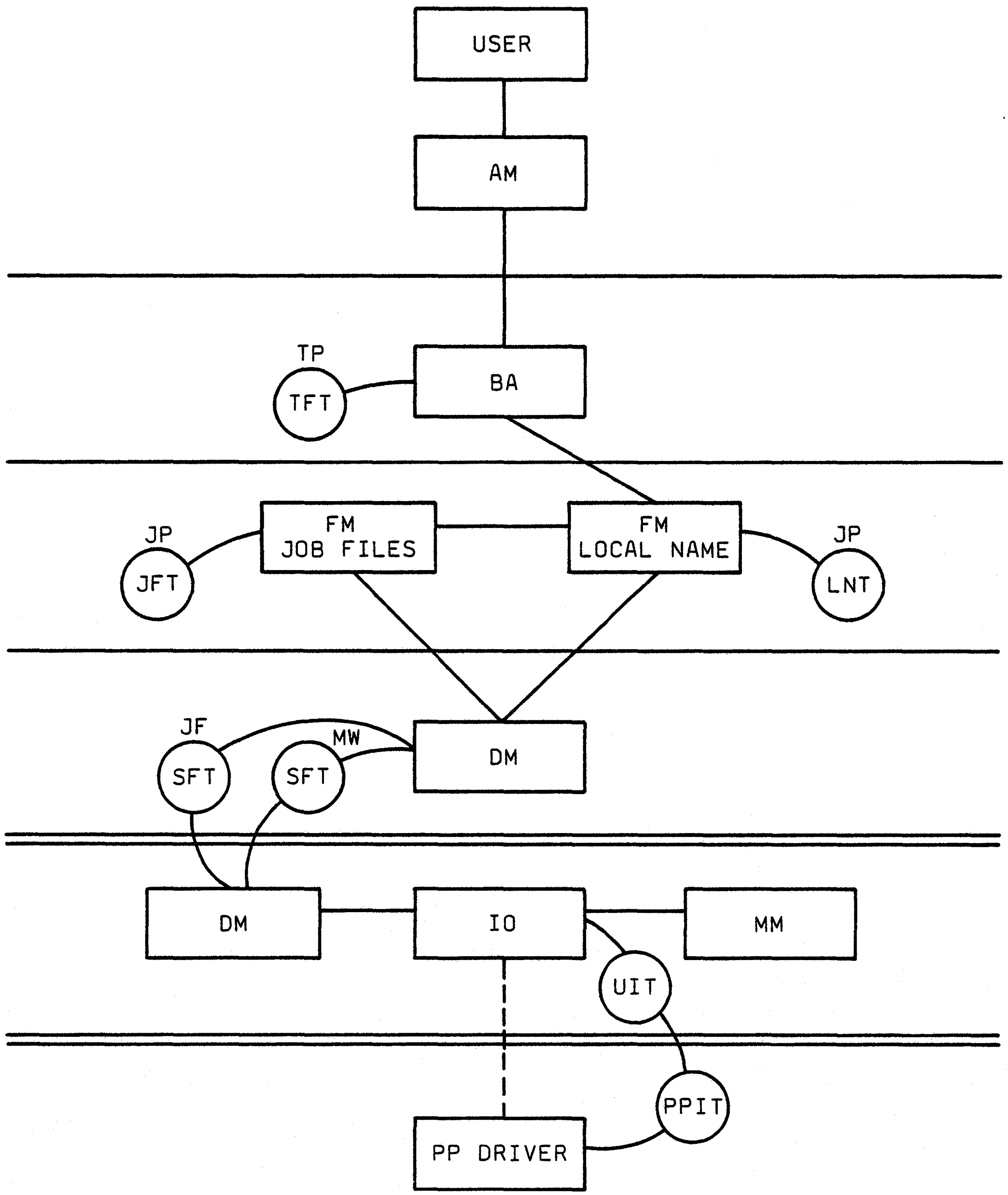
After completing this lesson the student should be able to--

- TRACE THE PROCESSING OF A RECORD LEVEL FILE FROM OPEN TO CLOSE
- TRACE THE PROCESSING OF A SEGMENT LEVEL FILE FROM OPEN TO CLOSE
- EXPLAIN HOW FAPS ARE HANDLED
- EXPLAIN THE USE OF THE MAIN FILE TABLES-LNT,JFT,TFT,SFT
- DESCRIBE THE ALGORITHMS FOR ASSIGNING DEVICES AND ALLOCATING SPACE ON DEVICES
- EXPLAIN THE USE OF THE MAIN DEVICE MANAGEMENT TABLES-FMD,FAT

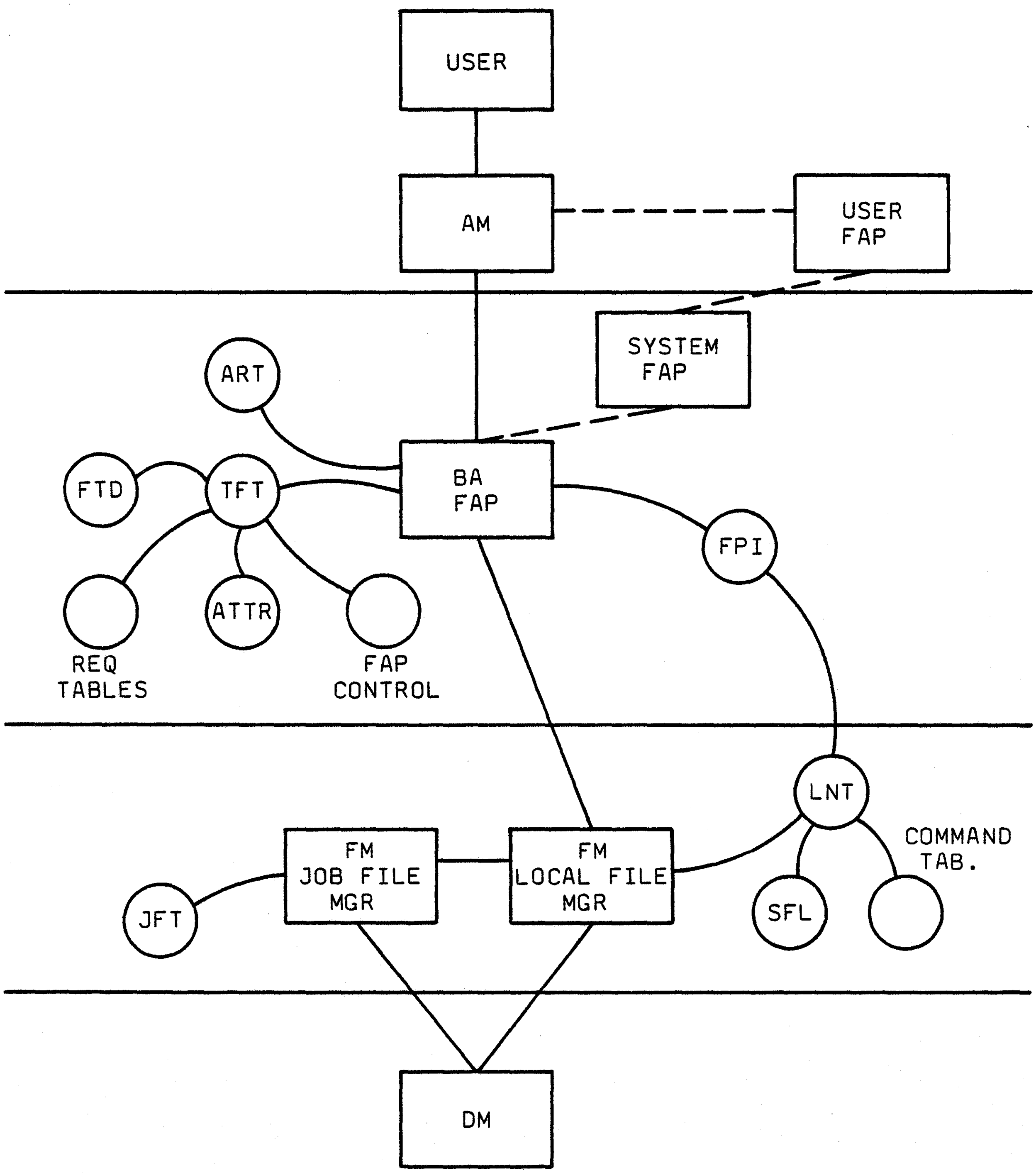
EXERCISE

TRACE THE DISK ALLOCATION OF A FILE

FILE MANAGEMENT COMPONENTS



BASIC ACCESS METHOD



FILE ACCESS PROCEDURES

1. USER

2. SYSTEM

   Advanced Access Method
   Connected File
   Operator Facility
   Interactive Facility
   Interstate Communication
   Logging

3. BASIC ACCESS METHOD

ATTRIBUTES

* Permanent attributes are established on the first open of a new file.

* Permanent attributes are never changed (R1).

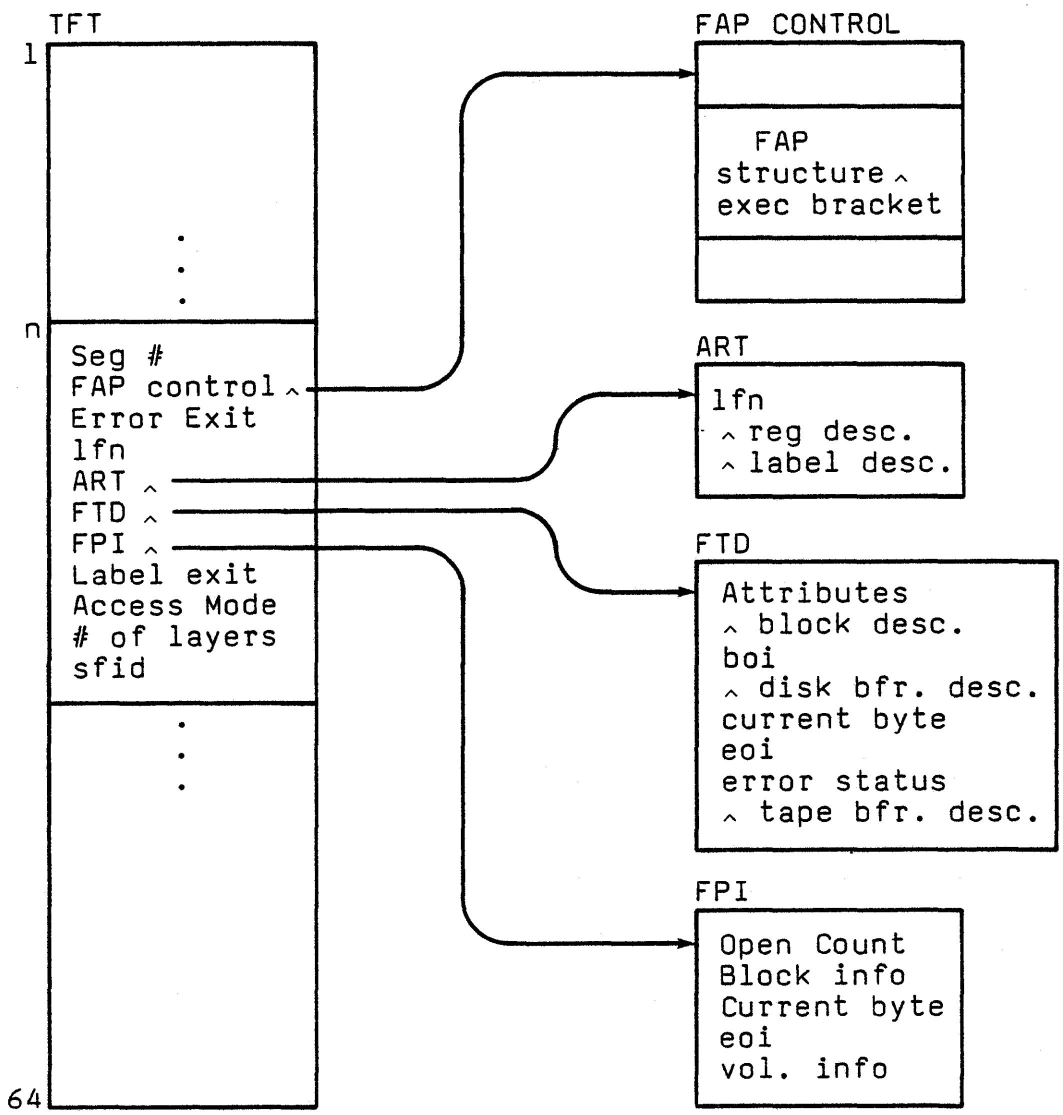
* Source of permanent attributes:

  FAP Request
  Open Request
  Commands
  Other program interface requests
  Defaults

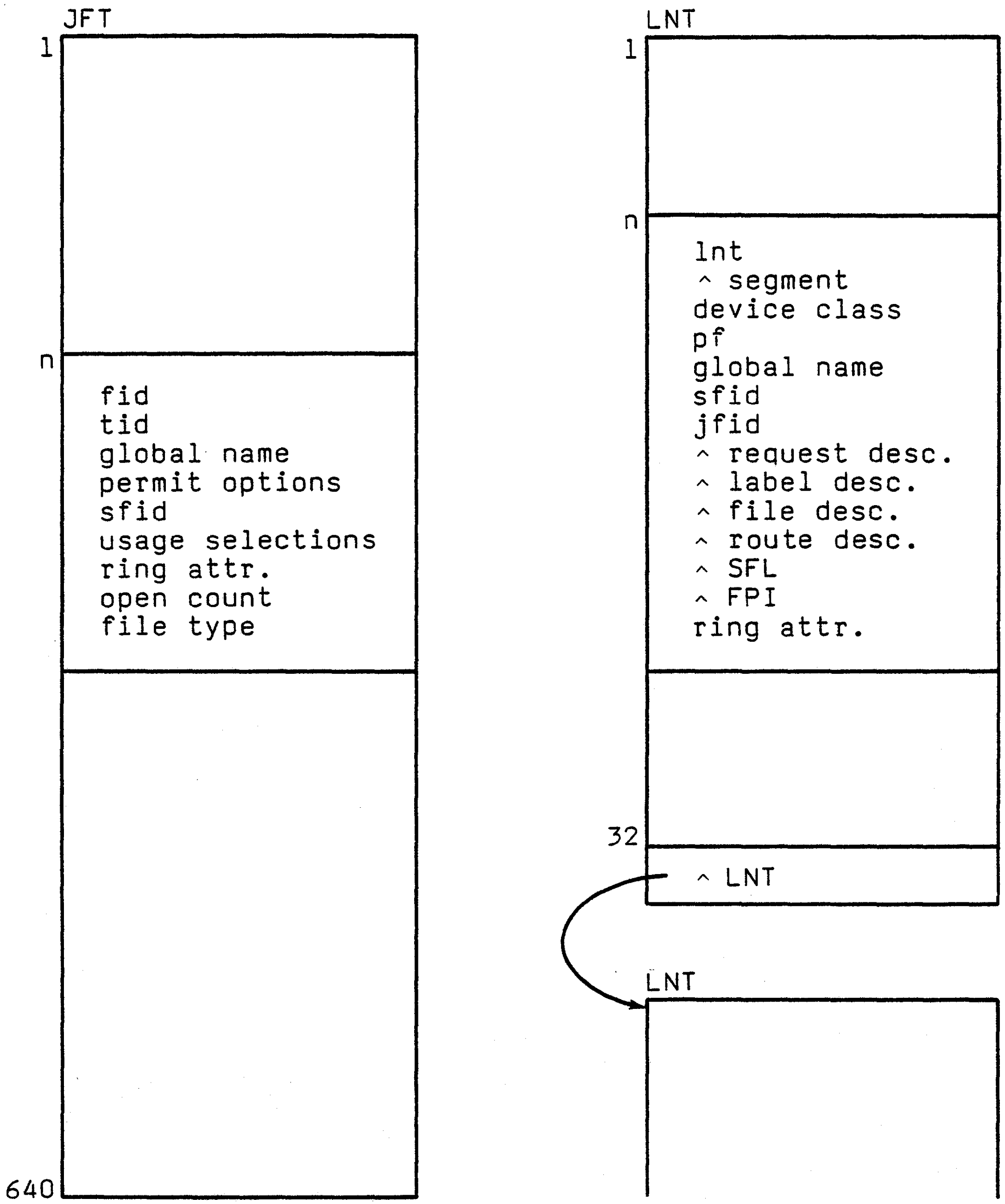
* Source of temporary attributes:

  Store request
  Open
  Commands
  Other program interface requests
  SFL
  Defaults

TASK FILE TABLES



FILE MANAGER TABLES



Local Name Mgr.

FMP$GET_JFID_SFID (lfn,jfid,sfid,status)

FMP$LN_ATTACH (lfn,sfid,usage_mode,share_mode,rings,status)

FMP$LN_OPEN_CHAPTER (lfn,chapter_number,validation_ring,
segment_attr,pointer_type,pointer,status)

FMP$LN_RENAME (old,new,validation_ring,status)

FMP$LN_RETURN (lfn,ring,returned,status)

FMP$LN_OPEN_NAME_TABLE (lfn,ring,chapter,access_level,
request_desc,label_desc,file_desc,new_file_desc,
system_attr,position_info,status)

FMP$LN_CREATE (lfn,file_attr,sfid,global_name,status)

FMP$GET_FILE_ATTRIBUTES (lfn,request_desc,lable_desc,file_desc,
new_file_desc,system_attr,position_info,status)

FMP$LN_GET_JOB_FILE_ID (lfn,jfid,status)

Job File Manager

FMP$CREATE_OPEN_CHAPTER (attributes,ring,access_mode,chapter,
jfid,sfid,status)

FMP$CREATE_JOB_FILE_ENTRY (attributes,global_name,jfid,sfid,
status)

FMP$RETURN_JOB_FILE (jfid,ring,returned,status)

FMP$ATTACH_JOB_FILE (sfid,attributes,jfid,status)

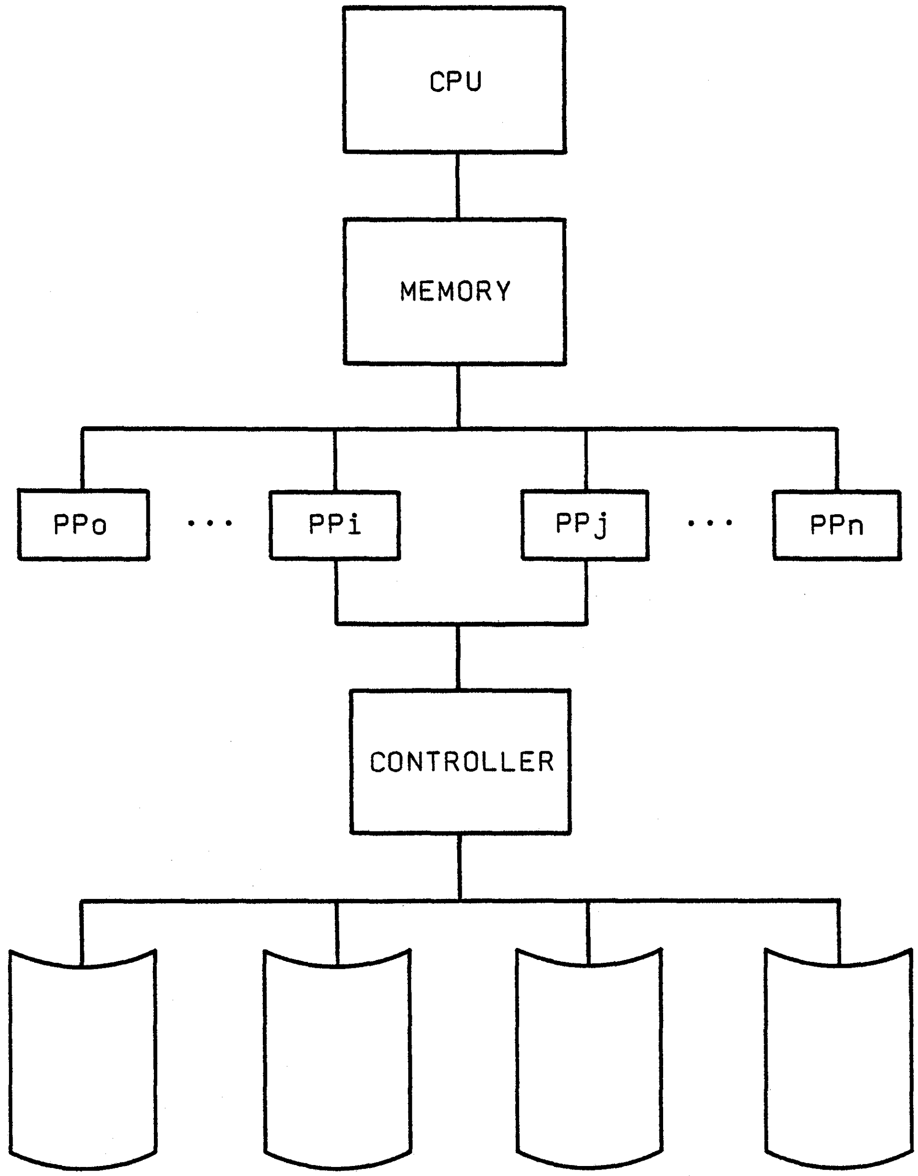
FMP$OPEN_PHYSICAL (jfid,ring,access_mode,status)

FMP$CLOSE_PHYSICAL (jfid,ring,status)

FMP$OPEN_CHAPTER (jfid,ring,access_mode,chapter,sfid,status)

FMP$CLOSE_CHAPTER (jfid,ring,chapter,status)

MASS STORAGE DEVICES



844-4x
885-1x
885-4x

DEFINITIONS

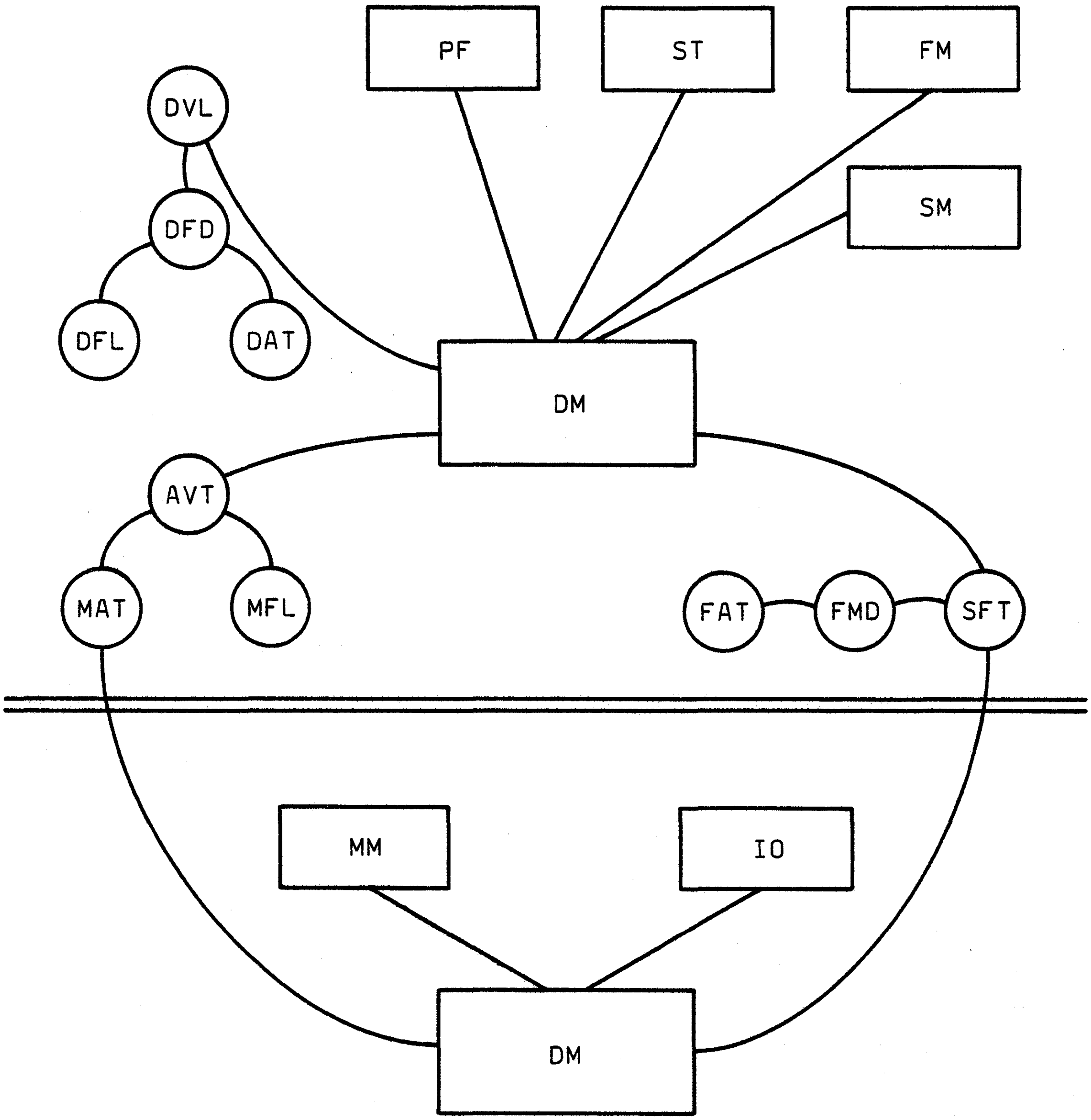
MAU--The minimum addressable unit is the quantum of data transfer between a driver and a mass storage device. It is a constant 2048 bytes in length. Standard software is released with page size ≥ 2048 bytes (MAU). Special systems could have page size < 2048 bytes but page size could never be changed without file conversion.

DAU--The device allocation unit is the quantum of device allocation. It is a device dependent, integral multiple of MAU.

ALLOCATION UNIT--A power of 2 multiples of contiguous DAUs on a device. An allocation unit does not span cylinders on a device. A physical I/O request does not span allocation units. Expressed as A1, A2, A4, A8, A16, A32, A64, A128, A256.

	844-4x	885-1x	885-4x
Capacity			
Cylinders/Spindle	823	843	843
Tracks/Cylinder	19	40	10
MAU/DAU (bytes)	(4096) 2	(4096) 2	(4096) 2
Total ($*10^6$ bytes)	151.6	552.5	552.5
Performance			
Seconds/Revolution	.0167	.0167	.0167
Transfer rate (bytes/sec)	$.589 \times 10^6$	$.981 \times 10^6$	3.924×10^6
Allocation			
DAU/A1 (Bytes)	(4096) 1	(4096) 1	(4096) 1
DAU/A2 (Bytes)	(8192) 2	(8192) 2	(8192) 2
.	.	.	.
.	.	.	.
.	.	.	.
DAU/A32 "	32	32	32
DAU/A64 "	(180224) 44	64	64
DAU/A128 "	44	128	128
DAU/A256 "	44	(655360)160	(655360)160

DEVICE MANAGER CONTEXT



DM TABLES

SFT	SYSTEM FILE TABLE	1 entry/file
FMD	FILE MEDIUM DESCRIPTOR	1 entry/subfile
FAT	FILE ALLOCATION TABLE	1 entry/allocation

DVL	DEVICE LABLE	1/volume
DFD	DEVICE FILE DIRECTORY	1 entry/device file
DFL	DEVICE FILE LIST	1 entry/subfile
DAT	DEVICE ALLOCATION TABLE	1 entry/AU

AVT	ACTIVE VOLUME TABLE	1 entry/volume
MFL	MAINFRAME FILE LIST	1 entry/new file
MAT	MAINFRAME ALLOCATION TABLE	1 entry/available AU

DEVICE MANAGER USERS

- FILE MANAGER
  - Locally Named File Mgr.
    - File Allocation
    - Set Mass Storage Limit
  - Job File Mgr.
    - Create File
    - Assign File to Device
    - Destroy File

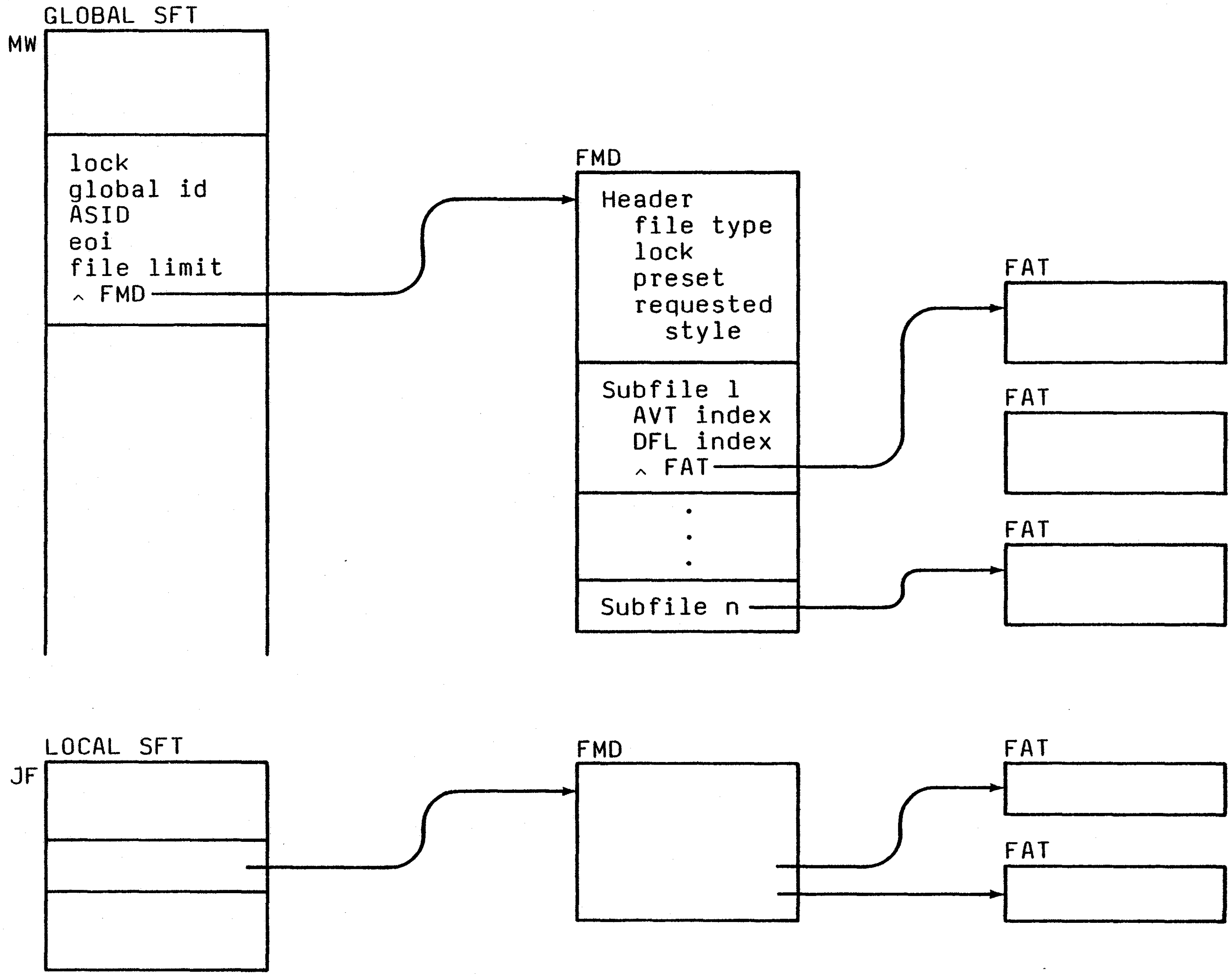
- MEMORY MANAGER
  - Store ASID in SFT for Sharing
  - Provide transfer unit offset and length

- PHYSICAL IO
  - Device Address for IO transfers
  - Check initial write of new allocation
  - Flaws

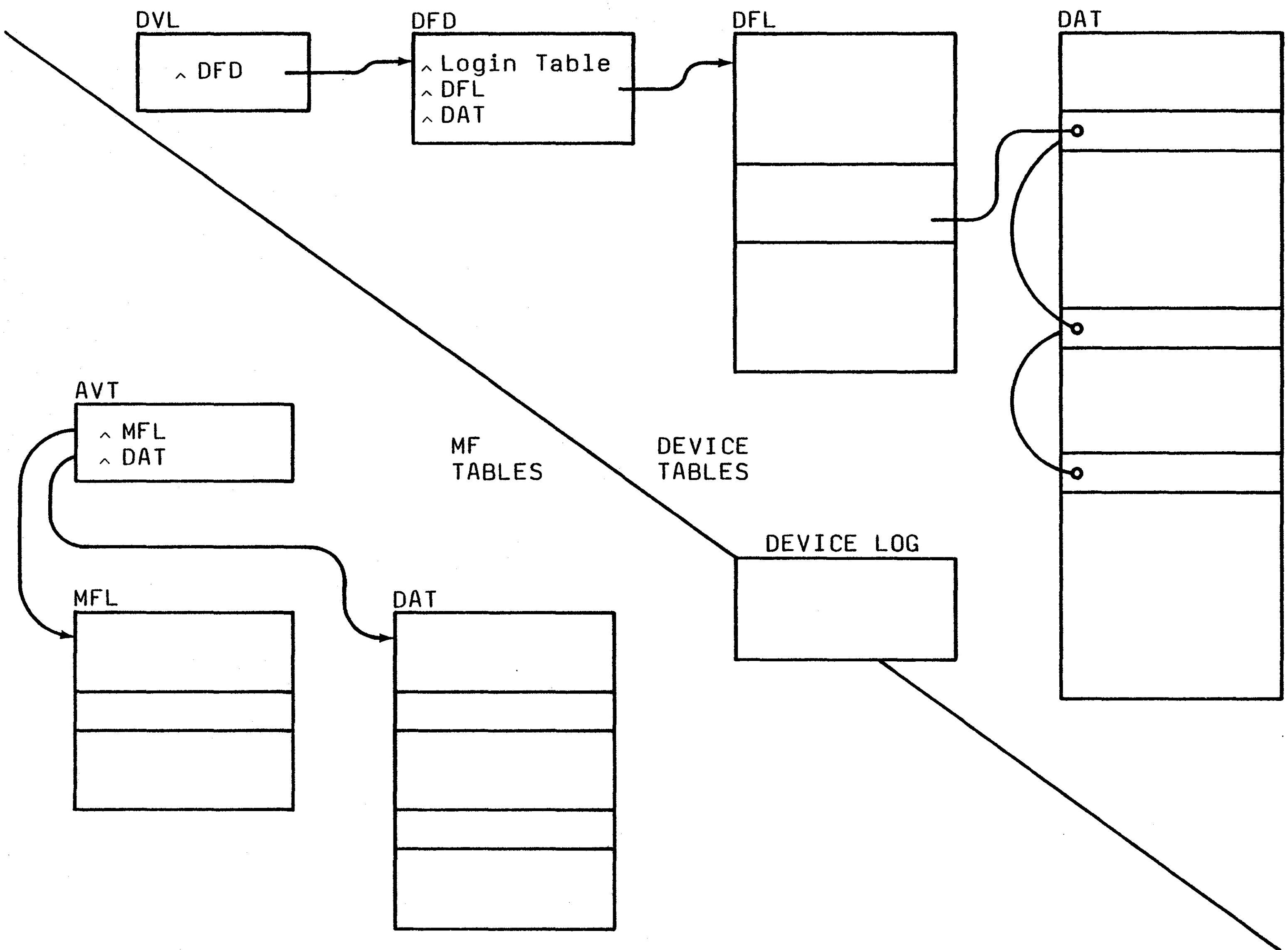
- MANAGE SETS
  - Add volume to Set
  - Remove volume from Set

- MANAGE PFs
  - Get FMD for storage in PF Catalog
  - Manage FMD on attach/detach
  - Destroy PF
  - Lock and Unlock Catalog File

DM FILE TABLES



DM DEVICE AND MANAGEMENT TABLES



LESSON 13
PHYSICAL I/O

LESSON PREVIEW

MEMORY MANAGEMENT
SEGMENT MANAGEMENT
PAGE FAULTS
PP COMMUNICATION
WORKING SET

OBJECTIVES

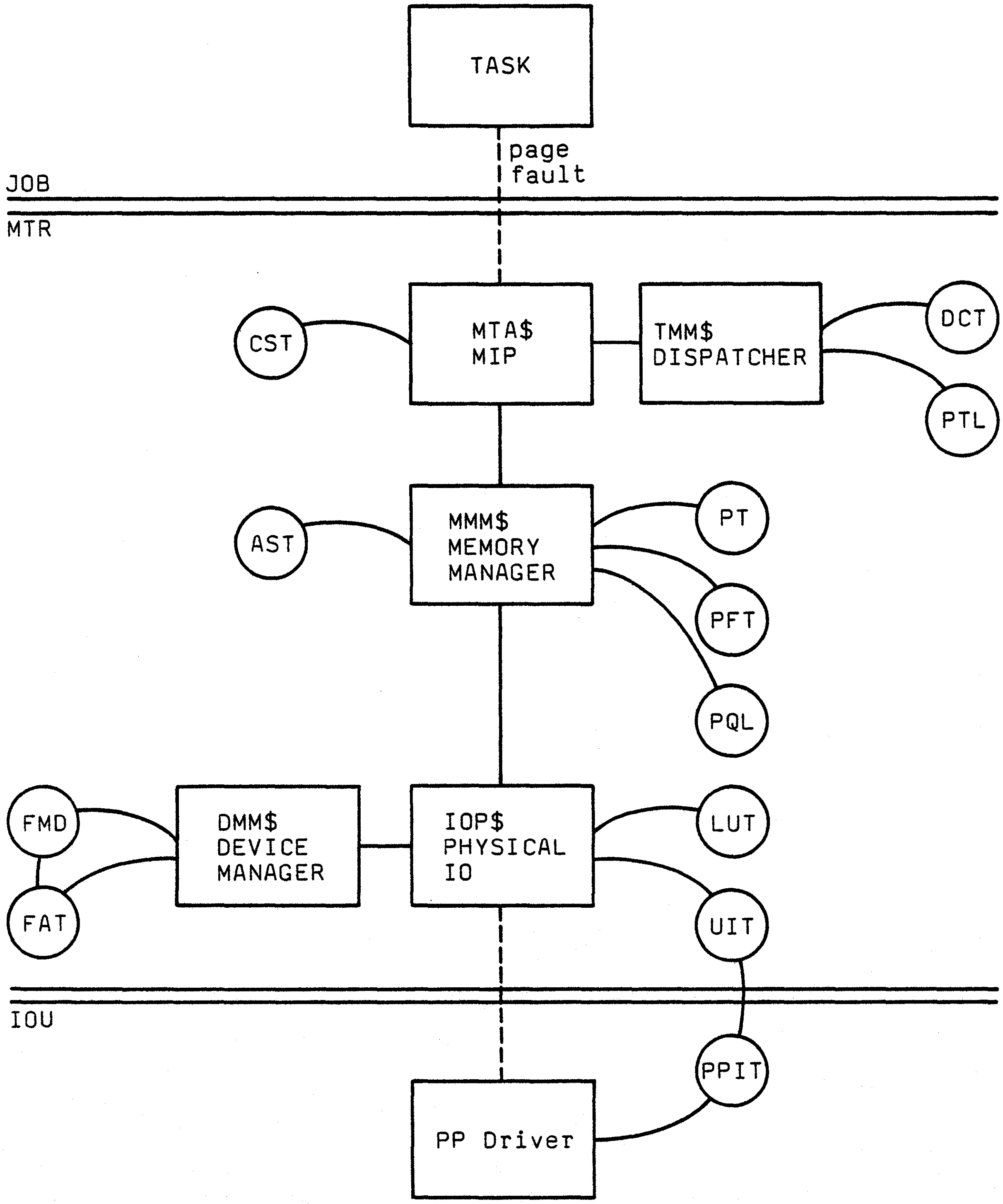
After completing this lesson the student should be able to--

- TRACE THE PROCESS OF RESOLVING PAGE FAULTS
- DESCRIBE THE WORKING SET ALGORITHMS
- TRACE THE PHYSICAL I/O PROCESSES FROM INITIAL REQUEST TIL THE TRANSFER IS COMPLETE

EXERCISE

NONE

MEMORY MANAGER CONTEXT



TABLES

CST	CPU State Table-MT ^XCB, JCB,statistics	1/CPU
PTL	Primary Task List-TM	1 entry/task
DCT	Dispatch Control Table-TM	1/mainframe
PT	Page Table-SY Hardware	1 entry/active page
PFT	Page Frame Table Software	1 entry/page
PQL	Page Queue List PFT tops of threads	1/mainframe
AST	Active Segment Table AST index → ASID	1/active segment
FMD	File Medium Descriptor	1/file
FAT	File Allocation Table	1/subfile
LUT	Logical Unit Table	1/drive
UIT	Unit Information Table IO request queue	1/drive
PPIT	PP Interface Table	1/drive

MODULES

MONITOR INTERRUPT HANDLER

- Receive Page Fault
- Call Memory Manager to process fault
- Call Physical IO Mgr to process completion

DISPATCHER

- Adjust wait status
- Pick next task to execute

MEMORY MANAGER

- Process Page Fault
- Manage Working Set
- Lock/Unlock pages

PHYSICAL IO

- Link requests
- Alert PP
- Process IO completion status

DEVICE MANAGER

- Provide physical addresses
- Allocate space

PP DRIVER

- Function and status the device
- Read/Write the device
- Read/Write Real Memory

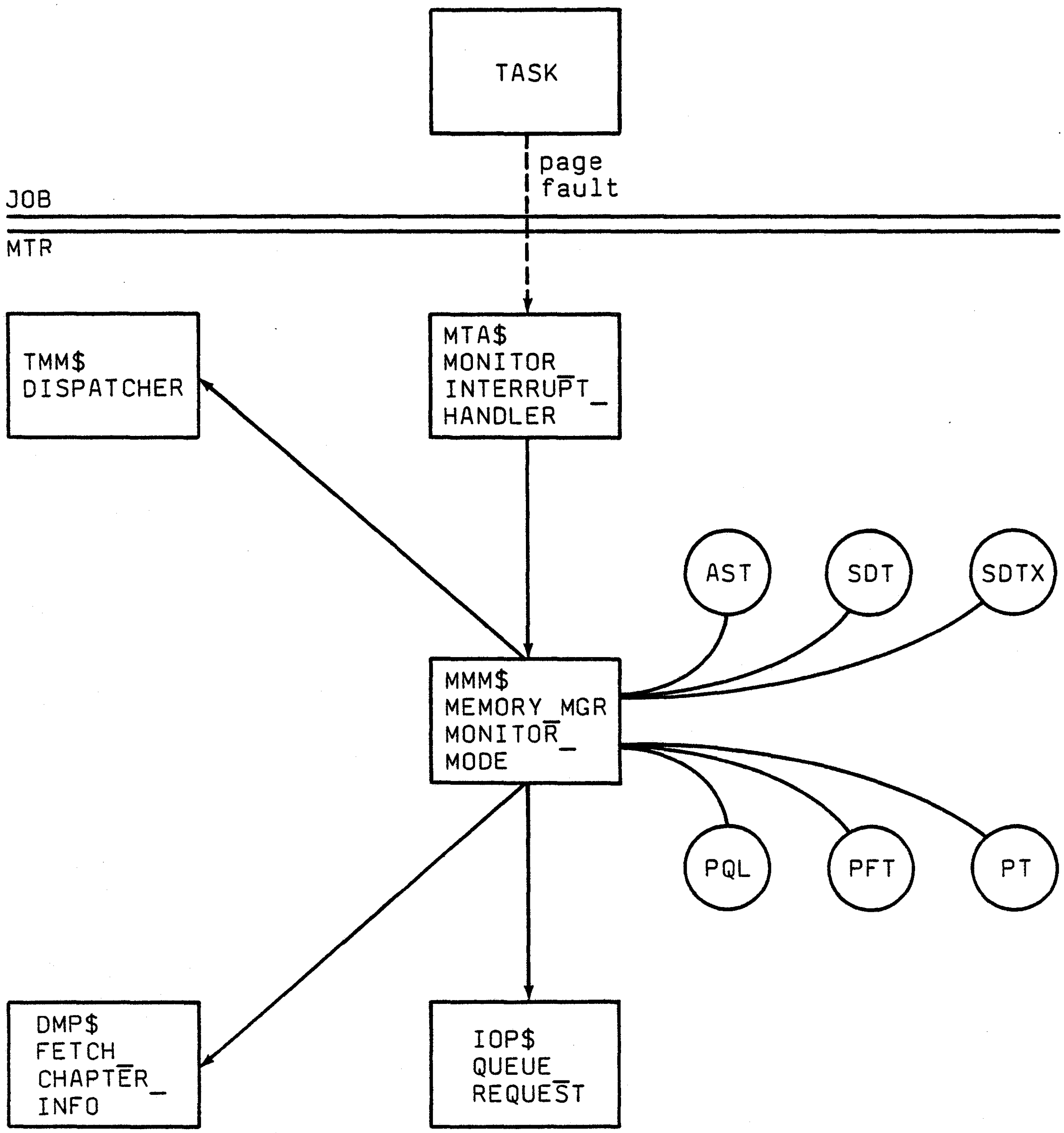
PHYSICAL IO

1. PROCESS PAGE FAULT

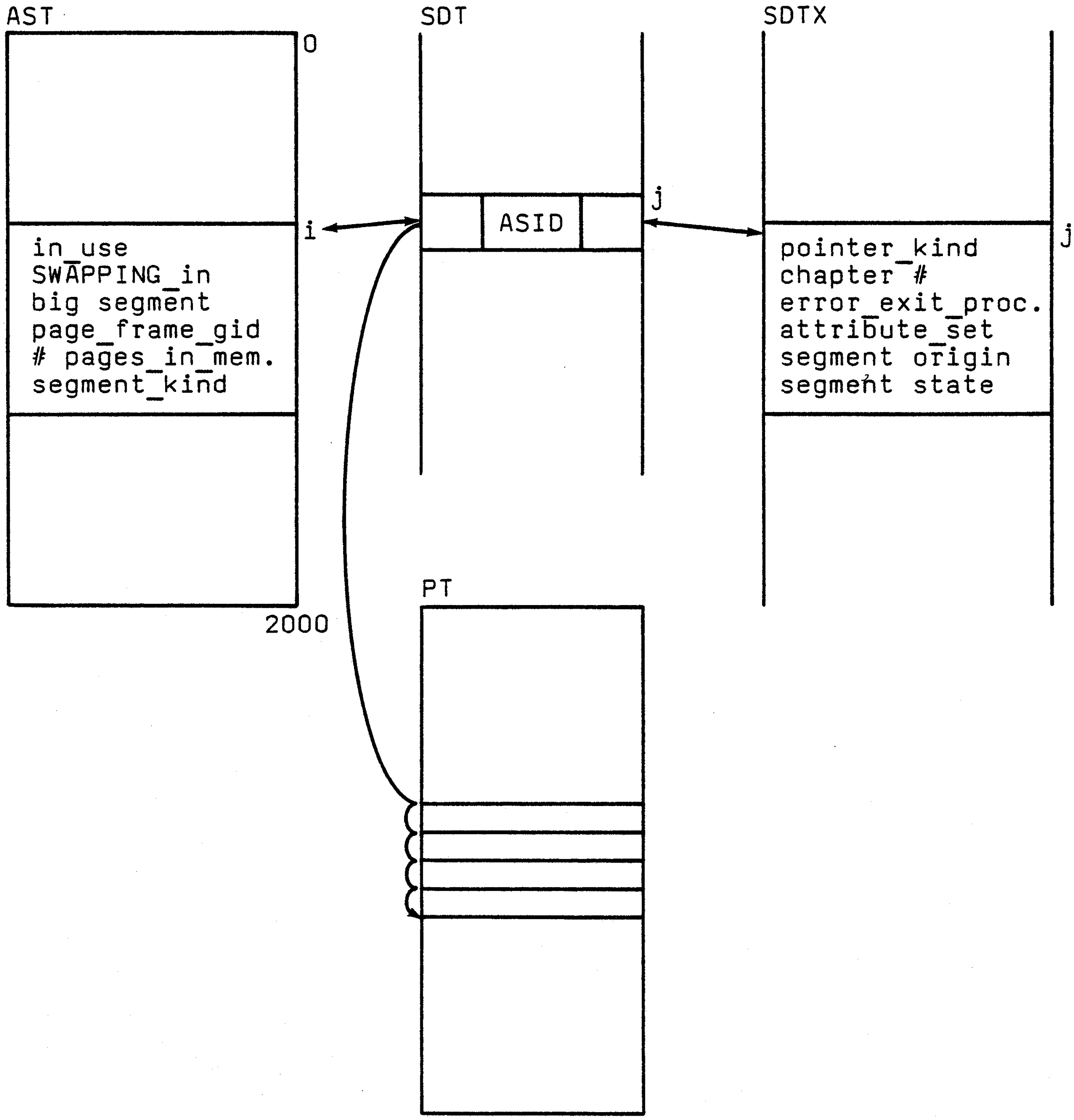
2. INITIATE PHYSICAL IO

3. PROCESS IO COMPLETION

PROCESS PAGE FAULT



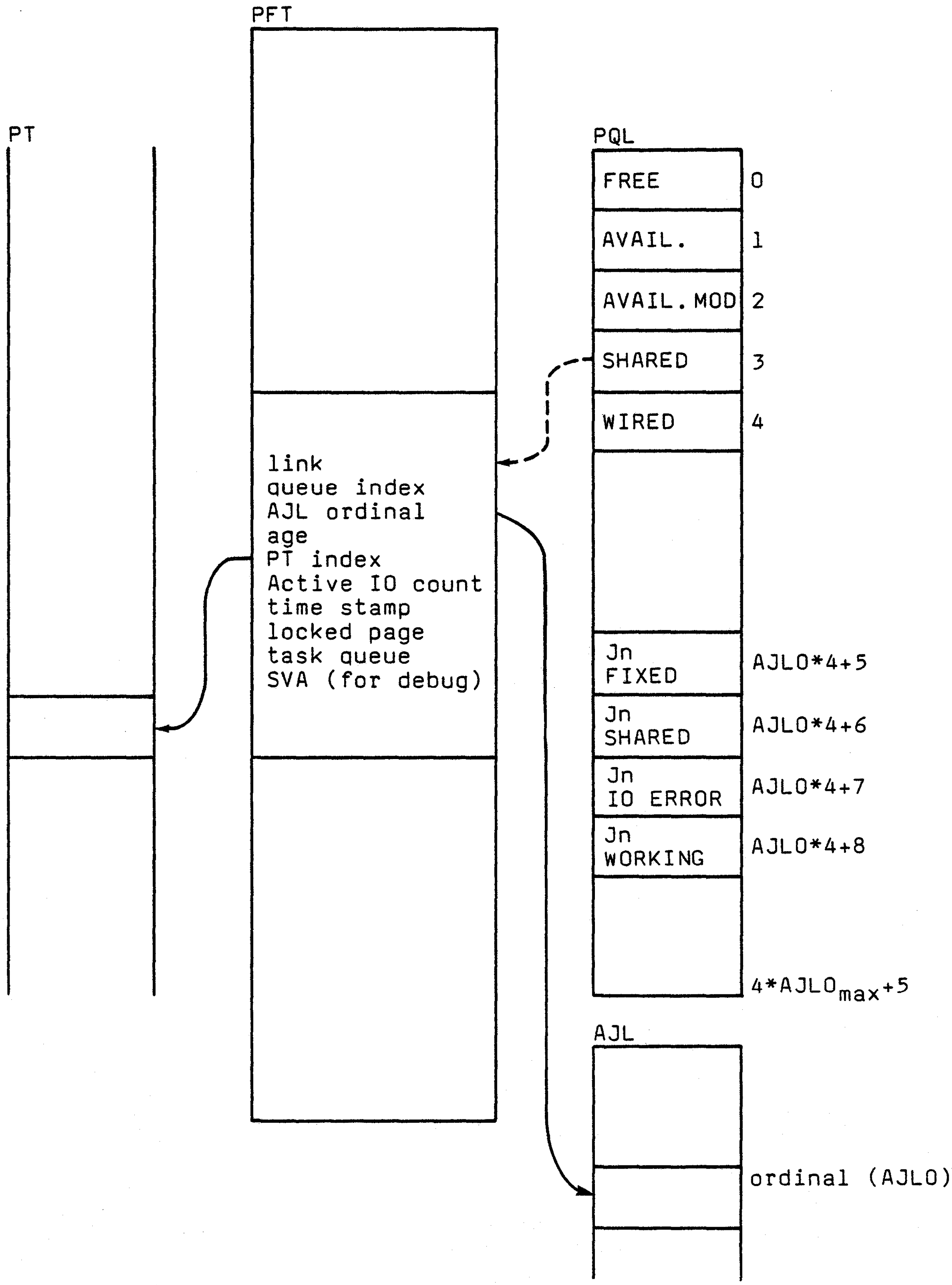
SEGMENT TABLES



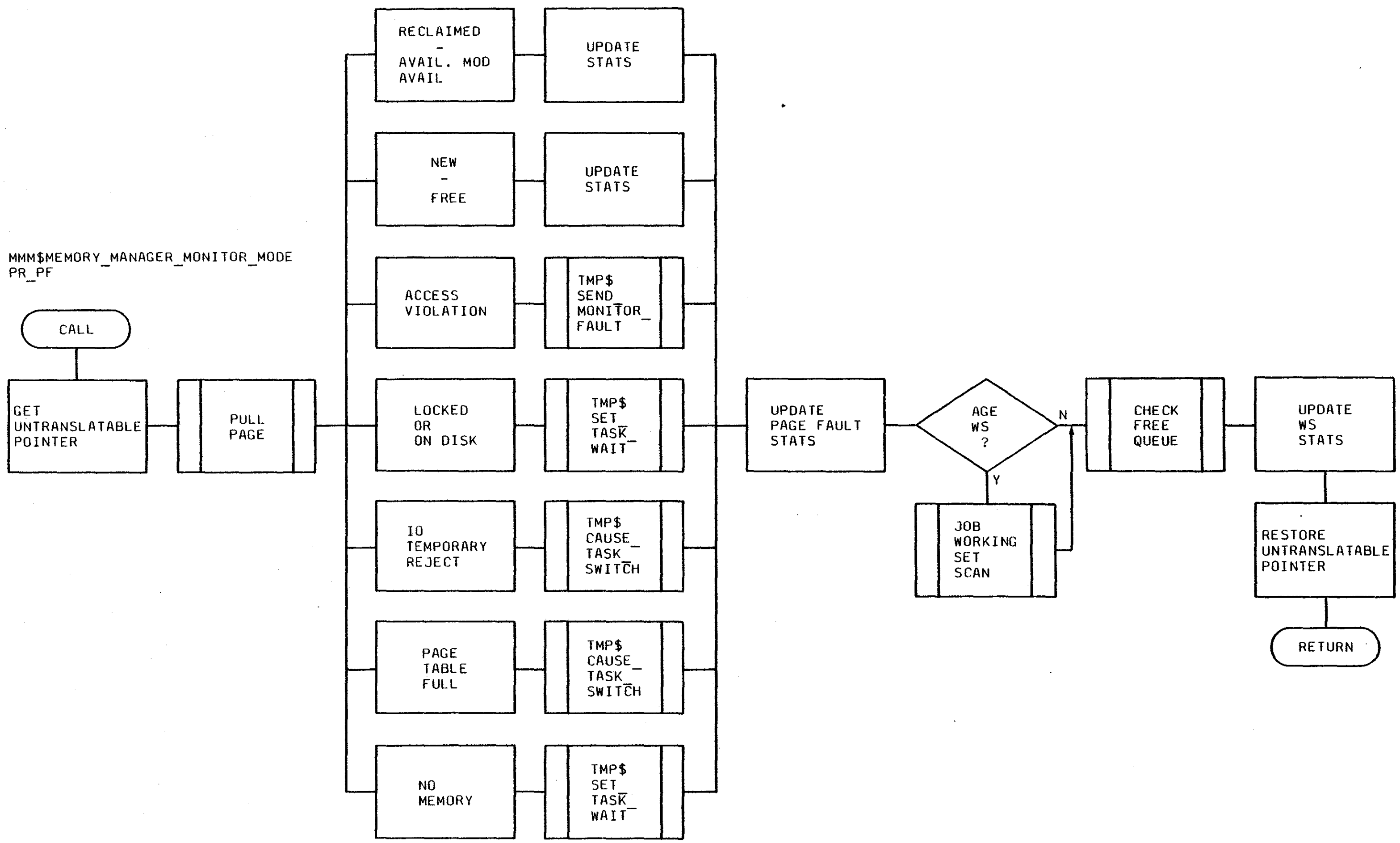
Software hashes the AST index to assign the ASID.

Hardware hashes the ASID and page offset to find the page table index. A sequential search of the next 24 entries might follow.

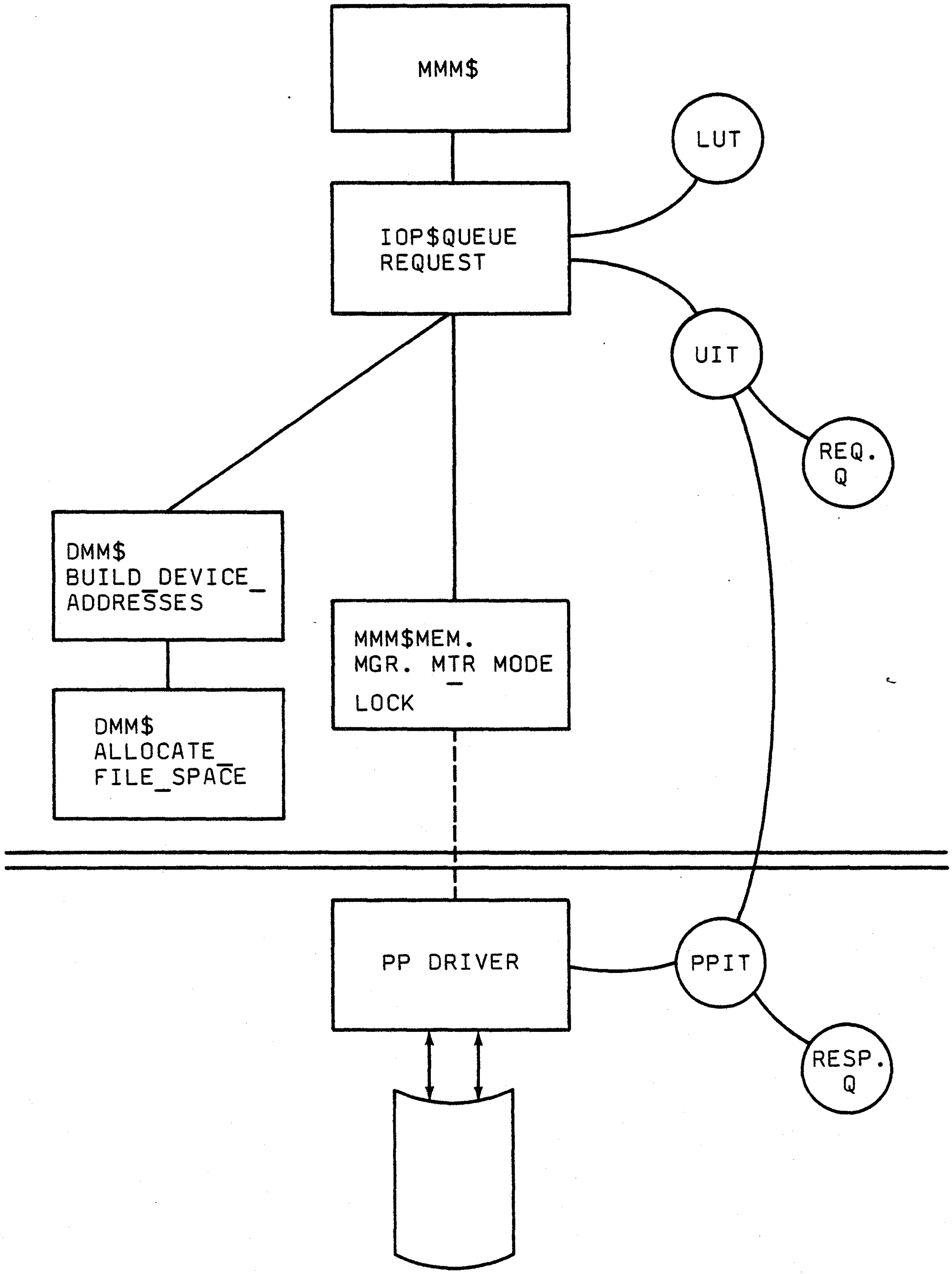
PAGING TABLES



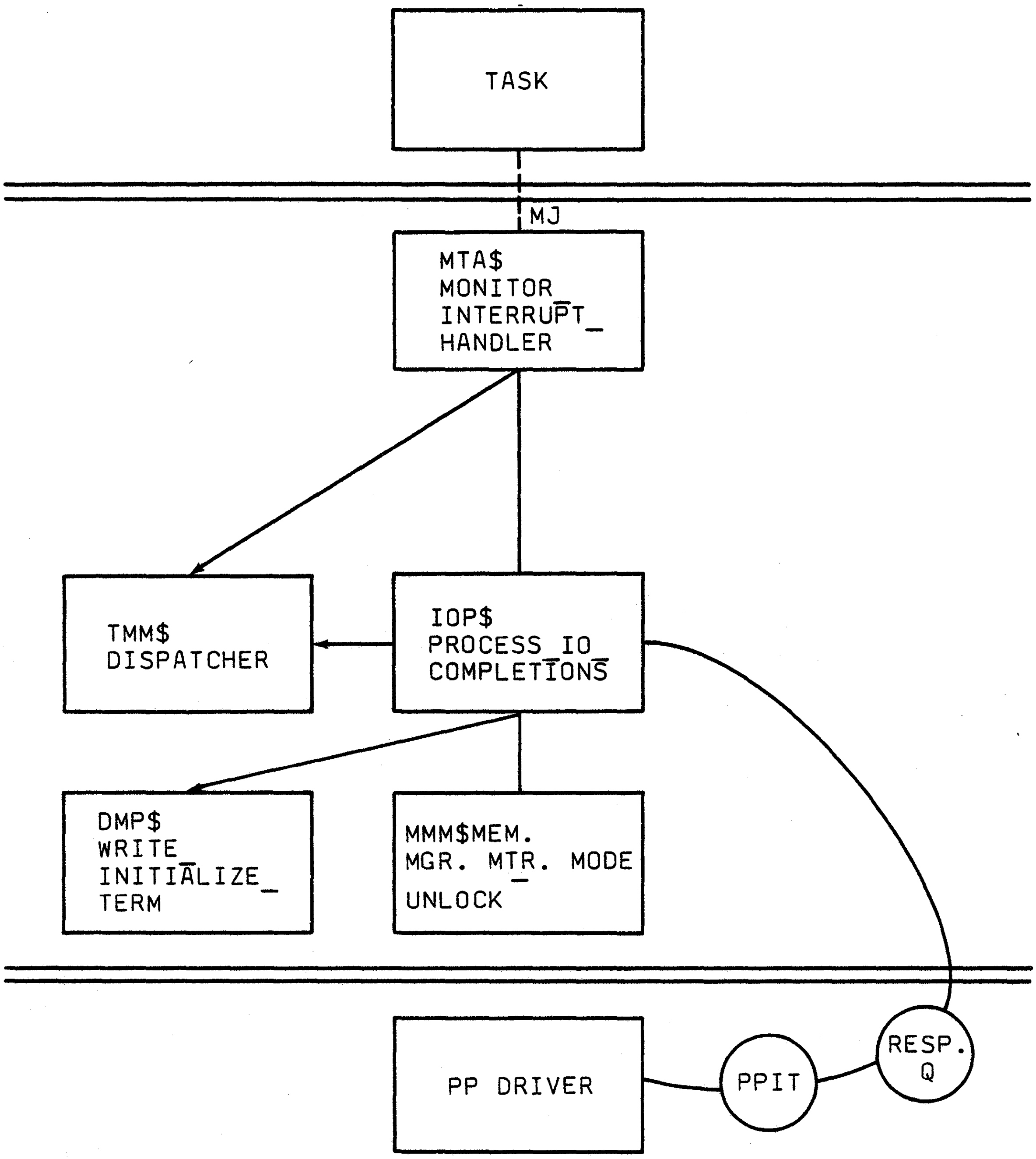
PROCESS PAGE FAULT



INITIATE PHYSICAL IO



COMPLETE IO REQUEST



NO ERRORS--Ready Task
PF ERROR--Notify PF Manager
READ ERROR--Abort Task
WRITE ERROR--Leave page in memory

NOS/VE DESIGN SPECIFICATION

PART III

SYSTEM PACKAGING

TABLE OF CONTENTS

1.0 SYSTEM STRUCTURE

A basic objective is to provide a well defined system structure which will result in a highly reliable system and one that can grow over time in an orderly and cost effective manner.

In order to meet this objective, a set of hardware and software conventions are imposed on both user and system code. This allows the normal protection, debugging, loading, code maintenance, accounting, and error handling methods of the user and the system to be the same. This also facilitates movement of services between user and system.

1.1 GENERAL STRUCTURE ELEMENTS

Jobs, tasks and modules represent the basic structure elements for all services provided by NOS/VE. They have the general relationship shown in figure 1. Each element has a set of unique execution attributes, interface conventions and resource requirements. System and application programmers make services available to users with combinations of these elements.

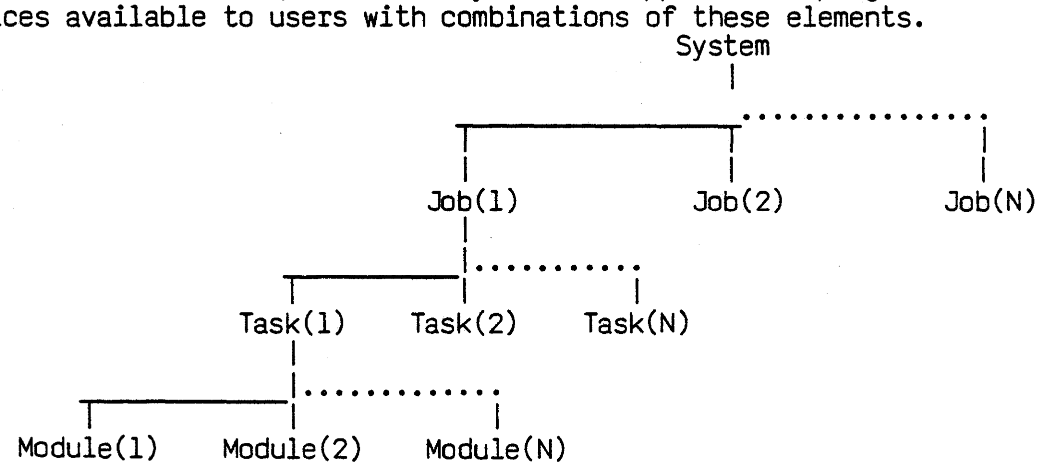


Figure 1 - Structure Elements

Each level contains a system element which monitors the progress of other elements within that level. The job level contains a system job which schedules, initiates, and terminates (normal or abnormal) user and system jobs. Within each job resides a system task which initiates and terminates tasks of the job. Within each task resides a collection of system modules which assist in the initiation and termination of the task.

1.1.1 JOB ELEMENT

The general facility for presenting work to the system is a job. Jobs run on behalf of a specific user whose identification is the basis of the system access control mechanisms. In addition to batch or interactive jobs that are submitted by end users, the operating system and various subsystems not initiated by end users also run as jobs. Since all jobs are protected and compete for resources via the same mechanism, it is anticipated that the addition of new subsystem jobs will be quite straightforward.

Every job consists of multiple tasks. An important characteristic of a job is that all tasks esecuting within the job share a common set of operating system services that are determined at the time of job initiation. These service modules, called task services, are the mechanism through which operating system functions are made available. They are constructed from a job templete that is selected based on job type. This allows different jobs to have different services.

1.1.2 TASK ELEMENT

A task is the execution of a program. A program is a set of modules organized to perform some specific function (e.g. compile COBOL statements, copy a file). Tasks are protected from one another, can be dynamically created and destroyed, can communicate with other tasks and can execute asynchronous with other tasks. Tasks are the only asynchronous execution unit supported by NOS/VE.

Tasks then are the environment for providing functions that are natural to place outside of the requesting environment. Tasks are requested via an operating system request. They have their own (clock) accounting, scheduling, and execution characteristics. Tasks can come and go independently and represent a mechanism which is used to control memory usage (e.g., each pass of a compiler as a separate task). Protection is enforced by different segment descriptor tables for the caller and callee.

The figure below illustrates a task environment.

```
                    ANY
                    TASK

               +-----------+
               |           |
               |   USER    |
               |  MODULES  |
               |           |
               |-----------|-----------+
               | PROTECTED |           |
               | RUN TIME  |           |  RING BOUNDARY
               |  MODULES  |           |
               |           |           |
               |-----------|-----------+
               |           |
               |    OS     |
               |  MODULES  |
               |           |
               +-----------+
                 ^   ^   ^
                 |   |   |
          +------+   |   +--------+
          |          |            |
          v          v            v
   +-----------+ +-----------+ +-----------+
   |           | |           | |           |
   | EXCHANGE  | |  SEGMENT  | |           |
   | PACKAGE   | |DESCRIPTOR | |   QUEUE   |
   |           | |   TABLE   | |           |
   |           | |           | |           |
   +-----------+ +-----------+ +-----------+

    EXECUTION     PROTECTION   COMMUNICATION
```

Every task looks similar to NOS/VE in that it has an exchange package which defines execution status, a segment descriptor table which defines protection, a queue which defines a communication path and a collection of modules which define the program. The collection of modules can include "user" modules, application or run time service modules and operating system modules. The address space of each task is subdivided by a ring protection hierarchy. An attribute of a module is its ring of execution. Each task will include modules which are protected from each other by executing in different ring brackets.

All tasks, regardless of the type of function they perform, have the same appearance as illustrated below.

```
     USER            PRODUCT SET          OS
     TASK               TASK             TASK

+------------+    +------------+    +------------+
|            |    |            |    |            |
|USER PROGRAM|    |  COMPILER  |    |OS  PROGRAM |
|            |    |            |    |            |
|  MODULES   |    |  MODULES   |    |  MODULES   |

|            |    |            |    |            |
|            |    |            |    |            |
+------------+    +------------+    +------------+
| PROTECTED  |    | PROTECTED  |    | PROTECTED  |

|            |    |            |    |            |
| RUN TIME   |    | RUN  TIME  |    | RUN  TIME  |
|            |    |            |    |            |
|  MODULES   |    |  MODULES   |    |  MODULES   |

|            |    |            |    |            |
|            |    |            |    |            |
+------------+    +------------+    +------------+
|            |    |            |    |            |
|    OS      |    |    OS      |    |    OS      |
|  MODULES   |    |  MODULES   |    |  MODULES   |

|            |    |            |    |            |
|            |    |            |    |            |
+------------+    +------------+    +------------+
```

1.1.3 MODULE ELEMENT

Modules are the environment for the set of services that are natural to place within the environment of the caller. These services are provided as procedures and are interfaced via the standard procedure call. They have the same (clock) accounting, scheduling, and execution characteristics as the caller. Examples include file access methods, loading, table handling and Fortran object time. The available services can be added dynamically by explicit requests of the loader. Protection enforced by the ring hardware may exist between the caller and callee.

1.2 NOS/VE STRUCTURE

NOS/VE utilizes the task and module structure elements to package the operating system services. Some of its tasks execute as part of the "user" jobs and some execute as part of NOS/VE system jobs. NOS/VE also collects together a set of modules that perform the lowest level operating system functions into a special environment called the CPU Monitor. The operating system services are provided within three basic environments:

CPU Monitor (one per system)
NOS/VE Modules (modules within each task)
Operating System Tasks (executing within "user" jobs, and executing within "system" jobs)

Every request a user makes of the system is translated into communication with one or more of these environments. Whenever operating system extensions are being implemented, the conventions and interfaces of these environments must be understood and used.

1.2.1 CPU MONITOR ENVIRONMENT

CPU Monitor is that portion of the operating system that is most directly related to the hardware environment. It provides:

- Basic intertask communication (signals)
- CPU Dispatching
- Basic CPU Scheduling
- Changing Task Status
- Interrupt Handling
- Page Management
- Basic Physical I/O Management

CPU Monitor is interrupt driven, nonpageable, and represents the most thoroughly debugged, least frequently changed code within the operating system.

1.2.1.1 CPU Monitor Request Handling

CPU monitor requests are only made by Task Services and Task Monitor functions. These requests are made using the hardware exchange instruction. Parameters are passed in the hardware registers.

1.2.2 NOS/VE MODULES ENVIRONMENT

NOS/VE modules are the set of operating system modules that execute within the environment of a task. These modules perform the operating system functions that are most directly related to the requestor's environment. To provide for maximum protection and RAM these modules are divided into Task Services modules and Task Monitor modules.

1.2.2.1 Task Services Modules

Task services modules provide the user interface to NOS/VE capabilities for:

- File Management
- Access Methods
- Program Management
- Job Management
- Resource Allocation

Task services is a collection of protected procedures. These procedures are directly callable by user code via the call instruction. The call causes a change in privilege for the called procedure, allowing these operating system services to execute with more or different privileges than the calling procedure. This type of structure allows protected operating system services to execute within the user environment. Task services provide a central interface for all requests and responses made and received by a task. If the requested service is not supported directly by task services, the request is passed on to CPU Monitor or to an operating system task. Task services occupies rings 3 to 6 within each address space. Only ring 3 is used for release 1 of NOS/VE.

1.2.2.1.1 TASK SERVICES REQUEST HANDLING

There are multiple task service entry points gated to requestors. Every call to a task service must supply a status variable of type ost$status. The parameter rules will conform to those of CYBIL.

1.2.2.2 Task Monitor Modules

Task monitor modules perform the more privileged functions of NOS/VE and execute at rings 1 and 2. These modules are a collection of procedures that interface to NOS/VE basic system tables (e.g. segment table, system file tables, catalogs, execution control tables) and to the CPU Monitor. The ring 2 procedures manage job global tables (i.e. accessible in all tasks of a job). The ring 1 procedures manage system wide tables (i.e. accessible in all tasks of all jobs) and are more privileged and critical to the integrity of the system. Task Monitor procedures are not directly callable by "users"; only NOS/VE Task Services procedures can directly interface to Task Monitor procedures.

1.2.3 OPERATING SYSTEM TASKS

Operating system tasks are those portions of the operating system that are relatively independent of the requestor's environment. They may execute asynchronous to the requestor and provide major portions of:

- Job Management
- Job Scheduling
- Operator Communications
- Device Drivers
- Hardware Maintenance

Execution of a system task is triggered by a signal passed into its communication queue. Tasks may execute in different processors. The device drivers, for example, are system tasks which execute on the IOU.

1.2.4 OPERATING SYSTEM COMMUNICATION

The operating system functions communicate using a basic signal handling service. The signals have a fixed format, a maximum size and are used by the operating system primarily for communication between address spaces. CPU Monitor is responsible for placing signals into the proper signal queue and for notifying the proper Task Monitor that a signal exists. Task Monitor is responsible for taking signals out of the communication queue and passing it to a Task Services signal handler. Routing, based on signal type, to a signal processor within Task Services will be effected by the Signal Handler.

1.2.5 OPERATING SYSTEM ENVIRONMENT SUMMARY

The following figure summarizes the basic environments and interfaces of NOS/VE.

```
              TASK(1)                           TASK(N)

          +---------------+                +---------------+
          |     USER      |                |               |
          |    MODULES    |                |               |
        1 +---------------+                +---------------+
          | PROTECTED RUN |                |               |
          | TIME MODULES  |                |               |
        1 +---------------+                +---------------+
          | TASK SERVICES | -----TRAP      |               | -----TRAP
          | TASK MONITOR  |                |               |
          +---------------+                +---------------+
                 ^2                               ^
                 |                                |
                 |                                |
                 +----------------+---------------+
                                  |
                                  2v
                          +---------------+
                          |
          CPU      -------INTERRUPTS 3    |
                          |    MONITOR    |
                          +---------------+
```

1 - INTERFACED VIA THE CALL INSTRUCTION, CYBIL PARAMETERS FOR COMMUNICATION, RINGS FOR PROTECTION

2 - INTERFACED VIA THE SYSTEM CALL, SIGNALS FOR COMMUNICATION, SEGMENT TABLES FOR PROTECTION

3 - INTERRUPTS ARE PROCESSED BY CPU MONITOR OR ARE TRANSLATED INTO SIGNALS

1.2.6 SEGMENT USAGE

1.2.6.1 Ring Assignment for a User Task

AREA	DATA PORTION	CODE PORTION	WHEN CREATED
USER APPLICATION PROGRAM	WORKING STORAGE, STACK, USER DATA	APPLICATION PROGRAM	AT LOAD TIME ACCORDING TO LIBRARY LIST IN PROGRAM DESCRIPTOR
PROTECTED RUN TIME MODULES	WORKING STORAGE, STACK	DATA BASE MANAGER	
TASK SERVICES/ TASK MONITOR MODULES	WORKING STORAGE, STACK, TABLES FOR JOB, TABLES FOR SYSTEM	RECORD MANAGER LOADER, PROGRAM COMM., TRAP HANDLING	A JOB TYPE TEMPLATE SUPPLIED BY SYSTEM GENERATION WHICH IS USED BY JOB INITIATION

This diagram illustrates:

1. Examples of code which exist at each ring bracket

2. Examples of private data at each ring bracket

3. When the data and code segments are created

Entry points to task services are created by system generation within the loader symbol table and are dynamically linked to external references from user and protected run time procedures by the loader.

1.2.6.2 Segment Assignments for User Modules

The following example demonstrates how the loader allocates and initializes segments based on information contained in compiler generated object text.

- Object Text Topology

RECORD TYPE	SAMPLE CONTENTS
(identificaton record)	name, date, generator name
(section definition)	code, binding, working storage, protection
(interpretive text)	text, replication, bit, entry, external
(transfer...end of text)	

- Generated Object Text

CODE SECTION (R,X)	STATIC SECTION (R,W)
. Non selfmodifying instructions	. Modifiable data

BINDING SECTION (B)	LITERAL SECTION (R)
. Base address of other sections . All procedure descriptions	. Constant data

DYNAMIC WORKING STORAGE SECTIONS (R,W)

. Common blocks
. Data allocated at run time

- Mapping sections to segments (assume 2 modules) providing an executable entity.

```
LGO file  Module 1
          Module 2
          EOF
```

Segments

Segment N (R,X)	Segment N+1 (B)	Segment N+2 (R,W)
Code Section M(1)	Binding Section M(1)	Static Data M(1)
Code Section M(2)	Binding Section M(2)	Static Data M(2)
		Any Named Common

Segment N+3 (R)	Segment N+4 (R,W,E)	Segment N+5 (R,W,E)
Literals M(1)	Universal Heap	Run Time Stack
Literals M(2)	(Grow)	(Grow)

The binding segment contains pointers to static, literals, code and other binding sections. The advantages of using segments include:

- Independent growth
- Integrity by separation
- Supports code sharing
- Non rewrite of code and constants (paging or swapping)

R - Read
E - Extensible
B - Binding
W - Write
X - Execute

2.0 SYSTEM TABLES AND INTERFACES

2.1 GENERAL GUIDELINES

The operating system is dependent on the use of tables to provide interfaces between different system modules and between the system and the user, and to describe the basic objects supported by the system and how these objects are related. When a table is defined within the system, consideration must be given to the following six general characteristics.

- Protection - Should the information be protected by hardware from inadvertent write operations? Must the information be protected from malicious write/read operations?

- Scope - Should the information be local to a user or should it be made global and shareable by other users? In general, information should be globally defined only when required. Keeping information local to a user has two advantages: 1) this information is private and no other user can interfere with it, and 2) if most of the tables required by a Job are collected locally, it is easier for the system to keep track of a user (swapping, restart, paging critical tables, etc.).

- Residence - Should the information be pageable or locked down? Whenever possible, information should be pageable. It should be locked down only when an obvious efficiency case exists. Three points can be made: 1) System Monitor cannot tolerate access interrupts, so any information referenced by System Monitor must be in real memory at the time of reference, 2) I/O channels use absolute addresses and require that real memory exists when in operation, and 3) there are degrees of paging, that is, some information must be present if a task is to use the CPU and can only be explicitly removed.

- Life Cycle - When will the table come into existence and when will it disappear? The data to describe a job is divided into environments which will go away, when the job terminates, when a task terminates, when the system crashes, and environments which will live forever unless explicitly removed.

- Crash Resistance - When the system crashes, how will the tables be reconstructed? What impact will there be on recovery if the tables cannot be reconstructed? Will the corrupting of the tables cause a system crash? What protection will be provided to detect corruption?

- Structure - The general structure of each of the NOS/VE Tables Area is the same and allocation of entries within a particular table is the same.

The contents (entries) of NOS/VE are position independent, that is,

a) the order and number of static entries in tables areas can vary from build to build;

b) the order and number of static entries in tables areas (task and job private) can vary among job types; and

c) the order and number of dynamic entries in the tables areas can vary among instances of execution.

The allocation of entries in NOS/VE tables should require minimal interaction among development projects; is controlled at the source level; via CYBIL; and is managed by execution and the system generator.

The general structure, allocation technique or order, value assignment tactics of NOS/VE tables should not impose undue constraints on the structure of entries contained in tables areas.

The allocation of entries and the assignment of values to entries in NOS/VE tables should be postponed as long as is feasible - priority order:

a) execution time
    1) first use time
    2) task initiation time
    3) job initiation time
    4) system initiation time
b) system generation time
c) source (compile) time.

2.2 TABLES AREAS

TABLES AREA	R1, R1
TASK SHARED	3, 13
TASK PRIVATE	3, 13
JOB PRIVATE PAGEABLE	2, 13
JOB PRIVATE FIXED	1, 3
MAINFRAME PAGEABLE	1, 3
MAINFRAME WIRED	1, 3

2.3 TABLES AREA GUIDELINES

2.3.1 JOB PRIVATE FIXED

The Job Private Fixed tables area is the container for tables shared among monitor and all tasks of a job. Job Private Fixed tables reside in non-pageable memory because of monitor access. Therefore, care should be exercised to minimize the amount of space allocated to entries which are not accessed by monitor.

2.3.1.1 Job Private Fixed Static Section

The Job Private Fixed static section is the container for statically allocated tables entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Job Private Fixed tables area. Statically allocated table entries are those which are somewhat constant in nature for the duration of the job. Such entries may also be "root" pointers to dynamically allocated entries in the Job Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.1.2 Job Private Fixed Dynamic Section

The Job Private Fixed dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) tables entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the job. Dynamic entries whose lifetime is less than that of the job must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.2 JOB PRIVATE PAGEABLE

The Job Private Pageable tables area is the container for tables <u>shared</u> among all tasks of a job. Table entries residing in this tables area are not accessible by monitor.

2.3.2.1 Job Private Pageable Static Section

The Job Private Pageable static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Job Private Pageable tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the job. Such entries may also be "root" pointers to dynamically allocated entries in the Job Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.2.2 Job Private Pageable Dynamic Section

The Job Private Pageable dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the job. Dynamic entries whose lifetime is less than that of the job must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.3 TASK PRIVATE

The Task Private tables area is the container for tables <u>shared</u> among procedures in task services and task monitor of a task. Task Private is pageable. Table entries residing in this tables area are not accessible by other tasks or monitor.

2.3.3.1 Task Private Static Section

The Task Private static section is the container for statically allocated tables entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Task Private tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the task. Such entries may also be "root" pointers to dynamically allocated entries in the Task Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.3.2 Task Private Dynamic Section

The Task Private dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the task. Dynamic entries whose lifetime is less than that of the task must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.4 MAINFRAME PAGEABLE

The Mainframe Pageable tables area is the container for tables shared among all jobs in the system. This tables area is writable by R1 task monitor and readable up to task services. The mainframe pageable tables area is not accessible to monitor.

2.3.4.1 Mainframe Pageable Static Section

The Mainframe Pageable static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Mainframe Pageable tables area.

Statically allocated table entries for those which are somewhat constant in nature for the duration of the system. Such entries may also be "root" pointers to dynamically allocated entries in the System Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.4.2 Mainframe Pageable Dynamic Section

The Mainframe Pageable dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the system. Dynamic entries whose lifetime is less than that of the system must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.5 MAINFRAME WIRED

The Mainframe Wired tables area is the container for tables shared among monitor and all jobs in the system. The Mainframe Wired tables reside in wired memory due to monitor access. Therefore, care should be exercised to minimize the amount of space allocated to entries which are not accessed by monitor.

2.3.5.1 Mainframe Wired Static Section

Only monitor software can allocate static table entries in the Mainframe Wired static section.

The Mainframe Wired static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Mainframe Wired tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the system. Such entries may also be "root" pointers to dynamically allocated entries in the System Private tables area. The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.5.2 Mainframe Wired Dynamic Section

The Mainframe Wired dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the system. Dynamic entries whose lifetime is less than that of the system must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

CONTROL DATA CORPORATION

CLASS EVALUATION

CLASS CYBIL DATE ______

INSTRUCTOR ______

CLASS OBJECTIVES Upon completion of this course the student will be prepared to implement software designs in CYBIL.

I. OBJECTIVES

A. Were the stated objectives the same as your objectives in attending this class?

____ Yes ____ No - Please explain the differences.

B. In your opinion, did you attain the stated objectives?

____ Yes ____ No - Please explain.

C. What topics do you feel were the most important?

D. What topics do you feel were the least important?

E. In your opinion, were any topics omitted? If so, what are they?

II. INSTRUCTION

A. Was the instructor effective in presenting the class material? Please explain.

B. Was the instructor knowledgeable in the subject material? Please explain.

C. In your opinion, were the instructor's examples effective in clarifying topic areas?

III. REFERENCE MATERIALS

How do you rate the reference materials and handouts used in the class?

IV. COURSE IN GENERAL

A. Were the assigned projects meaningful, and were they good exercises for the material covered?

B. List any suggestions you have for improvement concerning classroom facilities and materials.

C. What changes in the class would you make if you were the instructor?

D. Would you recommend this class to others in your company or department? Why?

Optional: ______________________________

Name and/or Company