DENNIS DINAN
GLENN ENTIS

MORGAN GUARANTY TRUST COMPANY OF NEW YORK

AN APPROACH TO ON-LINE APPLICATIONS MANAGEMENT

PRESENTED AT THE
HEWLETT-PACKARD GENERAL SYSTEMS USERS GROUP
7th INTERNATIONAL MEETING

OCTOBER 30 - NOVEMBER 3, 1978 DENVER, COLORADO

TERMINAL AND APPLICATIONS MANAGEMENT SYSTEM

TAMP (for Terminal and Applications Management System) is an integrated system of processes and subprograms which are used to define and manage an on-line application system. Each application to be run under TAMP includes one TAMPFILE, which is a multi-key, variable-length KSAM file designed to provide a compact, flexible, and easy-to-use interface to the applications environment. This TAMPFILE, which will be discussed below, defines all processing to be done within the monitor, and includes TAMP terminal configuration, user/function security matrix, and all data files to be managed by the system.

The first step in implementing an application under TAMP is the creation of the TAMP KSAM file. Here the user is asked questions about the size of his application, such as the maximum number of functions which he expects to include in the system. If the user miscalculates the dimensions of his application, he also has the option of later expanding his TAMPFILE, without affecting its data. The user also enters the Security Manager's name and password, and in so doing, restricts all further modification of the file to the bearer of that password.

The next stage in preparing a TAMPFILE for use is the definition of the application. This data is entered through a terminal interactive program and is grouped in logical records which represent the resources managed by TAMP. These resources are user functions, system utilities, data files, terminals, and users. A brief description of each of these records, their relationships to each other, and their use in on-line processing may be found below.

Although the following outline of TAMP record types greatly simplifies their structure, it does provide the essential content and purpose of each type of record.

FUNCTION - There is one function record for each callable application entry point. Each function may be one of several types (program subprogram, SL subprogram, independent process, MPE stream) and is invoked accordingly. If the function expects data file numbers or data from the terminal as parameters, that is also included here.

UTILITY - Utility records are almost identical to function records. However, those functions which are designated as utilities are initiated automatically by the TAMP software and may not receive parameters or be called by users.

FILE - This record contains a filename qualified by group and account. The file which this record represents may be of any type, including MPE, KSAM, IMAGE, and SMOG (described elsewhere in this paper). Once defined on a TAMP record, a file may be opened by the TAMP software and passed as a parameter to one or more functions.

TERMINAL - This record contains the logical device number of a terminal to be included in on-line TAMP. For each active terminal TAMP record, a process will be created by the on-line software which opens and allocates the corresponding logical device.

USER - The user's record includes his name, encoded password, processing values, and maps indicating which functions he may display on his menu and which functions he may actually call (these two types of function access are completely independent). Also included in this record are the assignments by which a user can expect each of four terminal soft-keys to call for him a selected function.

The program which defines a TAMPFILE also serves as the TAMPFILE maintenance program, and is where the Security Manager really does his job. As each user (and user password) is added to the system, the Security Manager also enters which functions that user may see on his menu and/or actually call. The Security Manager may also assign for each user two application-defined processing values, which are handy for such things as data-base user class codes, maximum dollar amount which may be entered by the user, or whatever else fits the application.

Other features of the TAMPFILE maintenance program include terminal configuration, assignment of data-files to functions (the filenumbers or file-tables of these files will be passed as parameters to their assigned functions during on-line processing), and definition

of system utilities, such as start-up, batch processing or logging routines.

The on-line portion of TAMP is the "outer block" within which all application programs will run. The TAMP software will call application defined start-up routines, open and allocate terminals, display sign-on screens, provide all first-level menus, and call requested application functions. This is accomplished through the use of a set of related processes which communicate with each other through a commonly held extra data segment.

On-line TAMP is actually a simple, two-level process tree consisting of a father process (FATHER'TAMP), and his various kinds of sons. FATHER'TAMP is the process which is created upon running TAMP. His first duties include initiating start-of-day application processing, which may also include restart and recovery procedures, and creating the extra data segment which will be used as the common communication area between processes. FATHER'TAMP next reads all terminal records from the TAMPFILE, and for each creates a terminal sign-on process. FATHER'TAMP also creates a process whose sole responsibility is communication with the system console, which here becomes the TAMP console as well. At this point, FATHER'TAMP suspends himself, and is reactivated only by sons with specific requests. All further processing depends on input from the user and console terminals.

The role of the common communication area in the extra data segment, as well as the orderly regulation of processes through use of Resource Identification Numbers (RINS) is a vital component of the on-line TAMP software. However, this processing is invisible to the user and beyond the scope of this paper. It should just be considered here that processes do not haphazardly suspend, terminate, and activate one another.

The task of the terminal sign-on processes is simple. Each displays a user sign-on screen, and then validates, by user name and password, attempted entries into the system. Upon a successful sign-on by a user, the terminal sign-on process for that user's terminal activates FATHER'TAMP with the appropriate message and then terminates.

Upon notice that a user has signed-on successfully, FATHER'TAMP creates a new process for that user. This process displays a welcome screen for the user, and, depending on the user's response, may display a menu of those functions which the Security Manager has permitted this user to see. The user may now request entry into a function in one of several ways, including selection by menu position, function name, or the user's own soft-key assignments.

The user's process calls the selected function in the manner specified in the function's record (e.g., if the selected function's type indicates that the function is a subprogram in one of the Segmented Libraries, the user's process will find, load, and jump to this subprogram). Also, if the function's record indicates that it expects data files and/or data from the terminal, the user process will open the appropriate files (if not already open) and pass the necessary data as parameters to the function.

When a user signs-off, control of his terminal is passed back to a terminal sign-on process. At the end of the day, FATHER'TAMP again takes control, signs remaining users off of the system, initiates application-defined end-of-day processing, and terminates.

SMOG is a HP2645A terminal interface and screen processing subsystem. It provides a convenient access to local and remote terminal devices using a wide range of the HP2645A terminal capabilities while keeping necessary terminal knowledge to a minimum.

SMOG makes very few demands on the application programmer. Developing screen images requires no knowledge of escape sequence characters. There are no stringent programming requirements necessary for accessing terminals in an interactive mode. All housekeeping, work areas, buffers and data transfers are handled transparently within the subsystem.

Screens exist in the system as logical records in a KSAM file. These records consist of the screen image in displayable form, control data and reporting data. Each record has an associated screen name which is used as the "key" when adding a screen record to a file or recalling an existing screen within a file. Also, there is a control record in the file. This contains global file information and sizing requirements necessary for allocating buffer space for screen records.

Working with screens, it is entirely up to the user how they will be designed. Then a screen file maintenance program assists the user in defining and entering the screen. The user must supply the row and column numbers of the start of each field, their lengths, their field type: protected or unprotected, any display enhancements and edit checking, plus any data which is to be used to initialize the fields. The maintenance program then interprets and assembles this data into a displayable character string with all appropriate escape sequences.

Counters and indexes are kept internally to accumulate read and write counts, field descriptions and status information about a screen.

Once entered, a screen may be easily modified, deleted, displayed, duplicated or renamed. Several reports are available including one containing a reproduction of the screen image. This report also contains a description of each field along with entry numbers which are used to address fields in subsequent screen maintenance or to programmatically access these fields for changing their enhancements, protection, or modifying their data content.

Screens in one file may be copied in entirety or selectively to new screen files. They may also be merged in the same manner with screens in other existing screen files. Provisions are included for both keeping and deleting another version of the same screen found during a merge.

Programmatically, SMOG has many useful features. One call to an initialization program sets up everything necessary to process. It is even smart enough to know how its process was started so that a terminal can be initialized properly. If a user father process was responsible for its creation, it expects that he also assigns a terminal to use, otherwise, it defaults to using the logon terminal. Also at this time, the caller's screen file is opened, the break key is disabled, buffer areas are dynamically allocated and a global work area is set up to maintain file numbers and status information which is necessary for on-line processing. None of this is ever seen by the programmer.

The programmer can now interact with the terminal using the screens from the screen file. Intrinsics are available to locate screens, to modify protected and unprotected data, display enhancements, and protection, to output a screen image, to read a screen's unprotected data, and to

dynamically create and read from individual fields on a screen. The outputting of a screen can optionally include the field at which the cursor is to be positioned; all reads from the terminal can be armed with a function key interrupt feature which returns an indicator to the program, or can optionally be timed out. The programmer has the responsibility of calling an intrinsic to reset the terminal before exiting from a program.

SMOG is very versatile. It can alternate the use of both block and character mode of I/O. It is callable from COBOL, FORTRAN, and SPL. In each of these languages, it allows a simple approach to screen processing.

Keyed Extra Data Segments (KEDS) is a method of organizing ordered sets of tables within extra data segments. A set consists of one or more tables. A table is comprised of one or more elements. Access to these tables is permitted by table element within table within set.

This access method allows an application to keep all necessary tables "virtually" memory resident while permitting them to be shared by several processes within the same job or session. It allows for elements to be deposited into a table by a relative element number and retrieved by the same element number or by searching for a table element with a specific key value.

KEDS is relatively simple to use. Initially, table space must be allocated. This is done by an intrinsic call with which the program must supply a set name, the size of the data segments to be used, the number of tables in the set and a list containing each table description. Each table in a set must be assigned a name, an element size, and a maximum number of elements.

Using this information, KEDS calculates the total amount of memory needed and creates as many extra data segments necessary to contain all of the user's tables. A parent segment is also created through which the tables will be accessed. Therefore, by accessing the parent segment, one can determine if an element exists in a table, the segment in which it exists, its location within the segment, plus its length. Once created, the programmer must load table elements in the order in which they are to be used.

Elements can be deposited and retrieved very efficiently by KEDS, using relative numbers, but, this is not always the easiest or best way of accessing tables. If a programmer so desires, KEDS will search a table for an element with a specific key value. Keys are not restricted to any size or location within the element. Tables can exist which have several keys and the choice of which will be used in the search is made by the calling program. Tables can be ordered in ascending or random key sequence.

Data transfers are very flexible. One can specify that all, part or none of an element is to be transferred if it is found during a search. This is a powerful tool for validation tables or tables whose elements are suited for multiple purposes.

The complete capabilities available to the programmer are: create tables, append element, update element, read element, read element by key (random or sequential), and delete tables. Combined, they provide a simplified approach to memory resident table management.

Morgan Guaranty Trust Company of New York

- Automated Banking System
- Euro·currency Automation
- Global Exposure System
- N.Y. Profitability System

ENVIRONMENT

- HP 3000 Series II/III
- On-line & Batch
- Multi-user

REQUIREMENTS

- Self-contained system
- Computer management
- User & function oriented security
- Terminal management
- Full screen support

TERMINAL APPLICATION MANAGEMENT PROGRAM

G-09.16

G-09.17

SCREEN MANAGEMENT OFF-LINE GENERATION

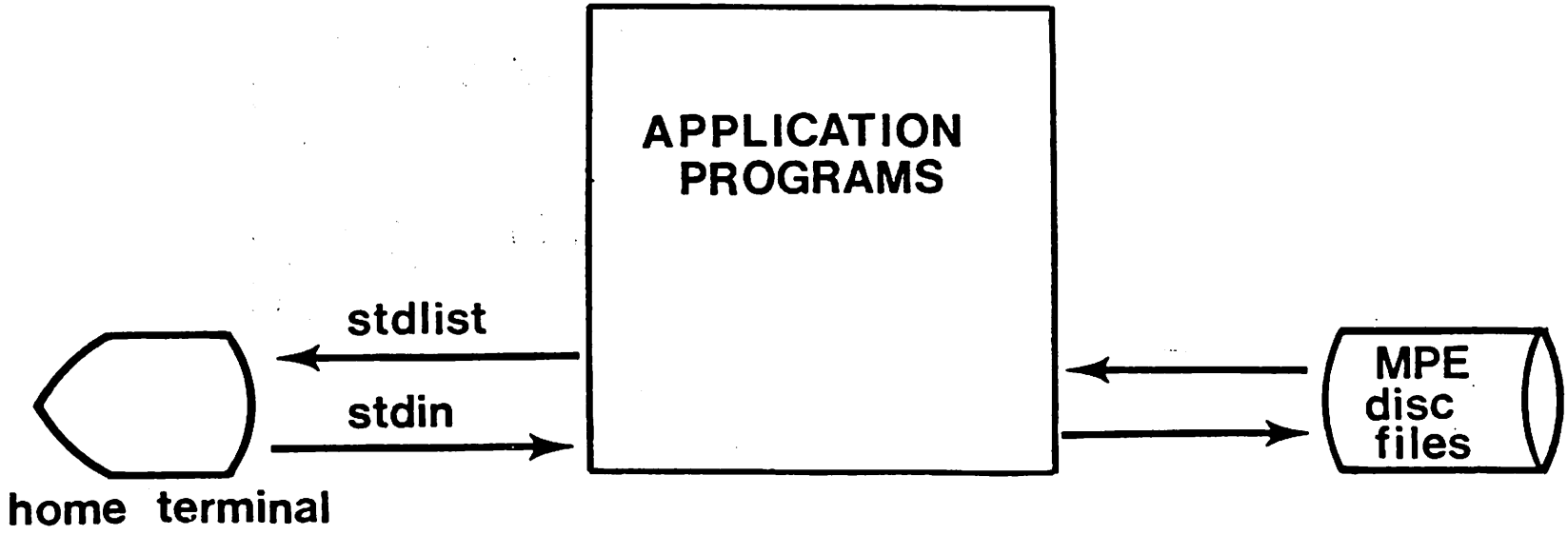
KEYED

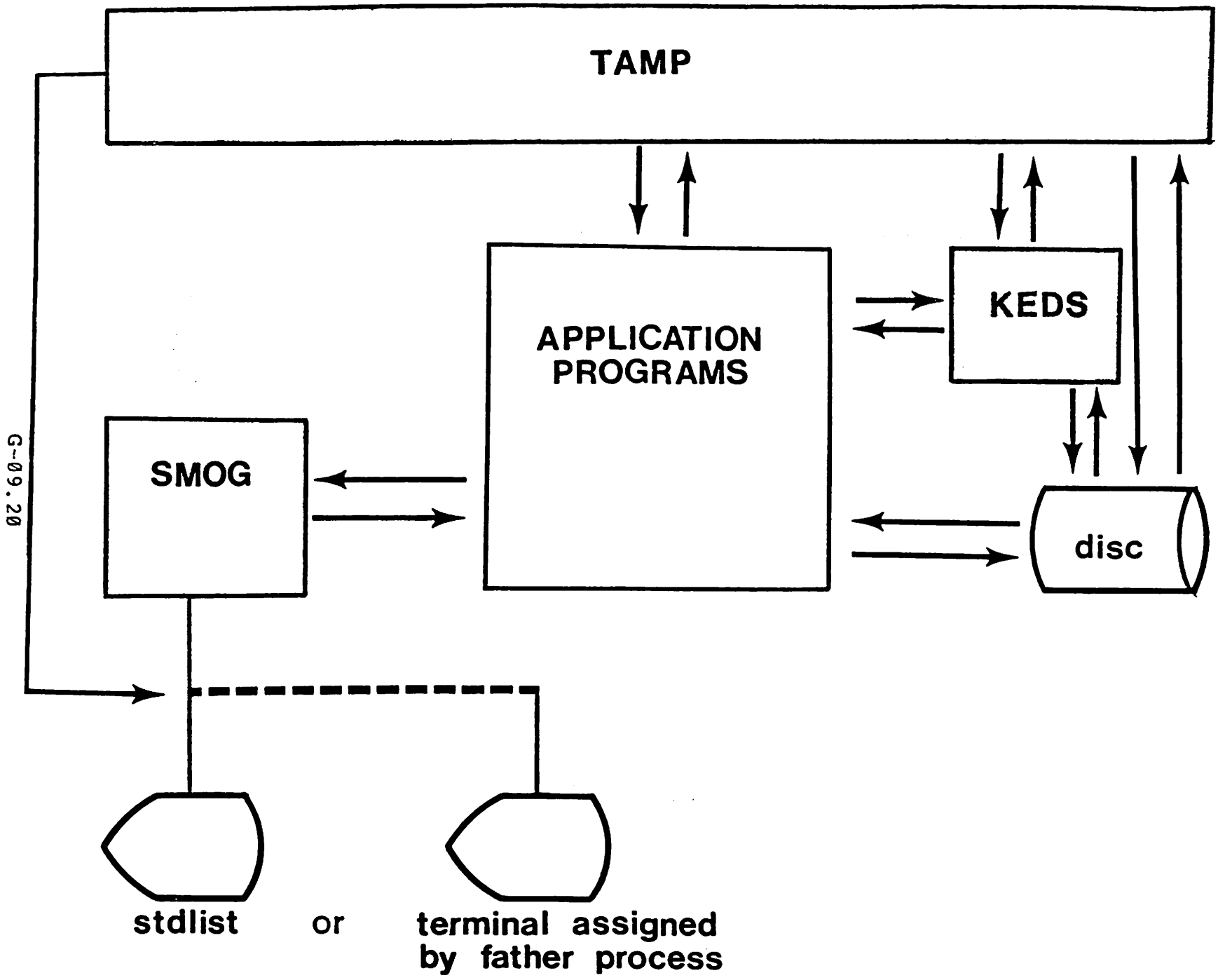
EXTRA

DATA

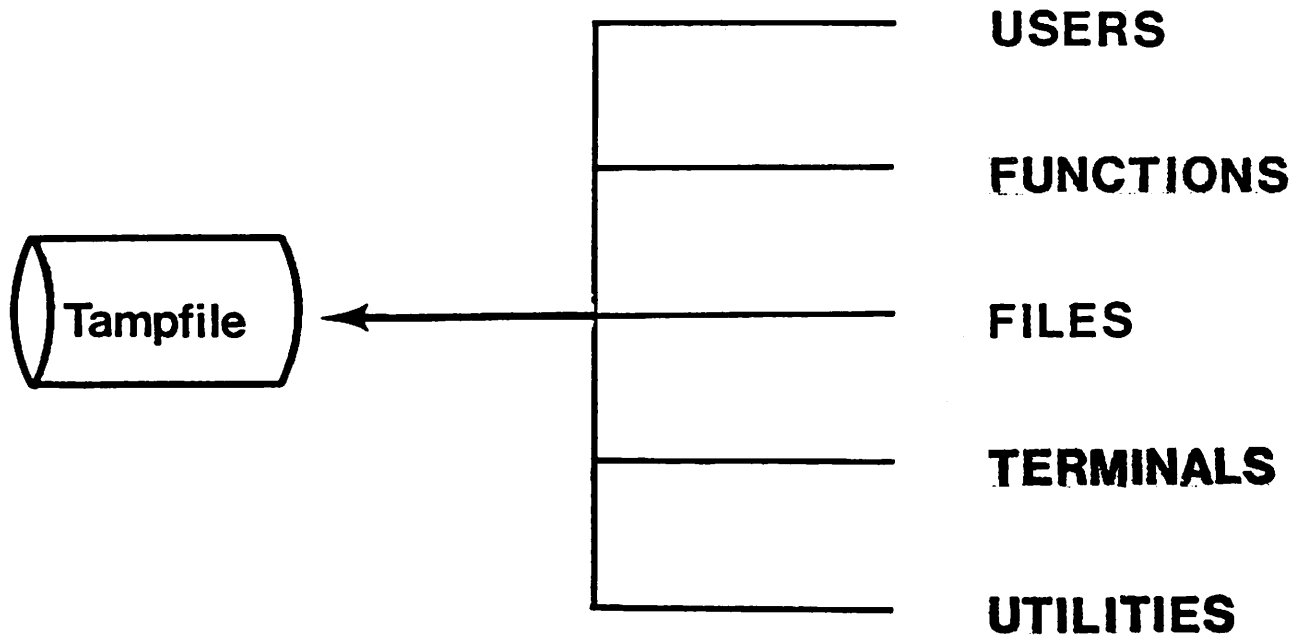
SEGMENTS

Applications environment




TAMP
APPLICATION PROGRAMS
KEDS
SMOG
disc
stdlist
or
terminal assigned by father process
G-09.20

TAMPFILE



USER

- name
- password
- function (view)
- function (call)
- processing values
- softkeys

FUNCTION

- name
- type
- entry point
- segmented library
- run priority
- description
- data files

FILE

- name
- type
- multi-access

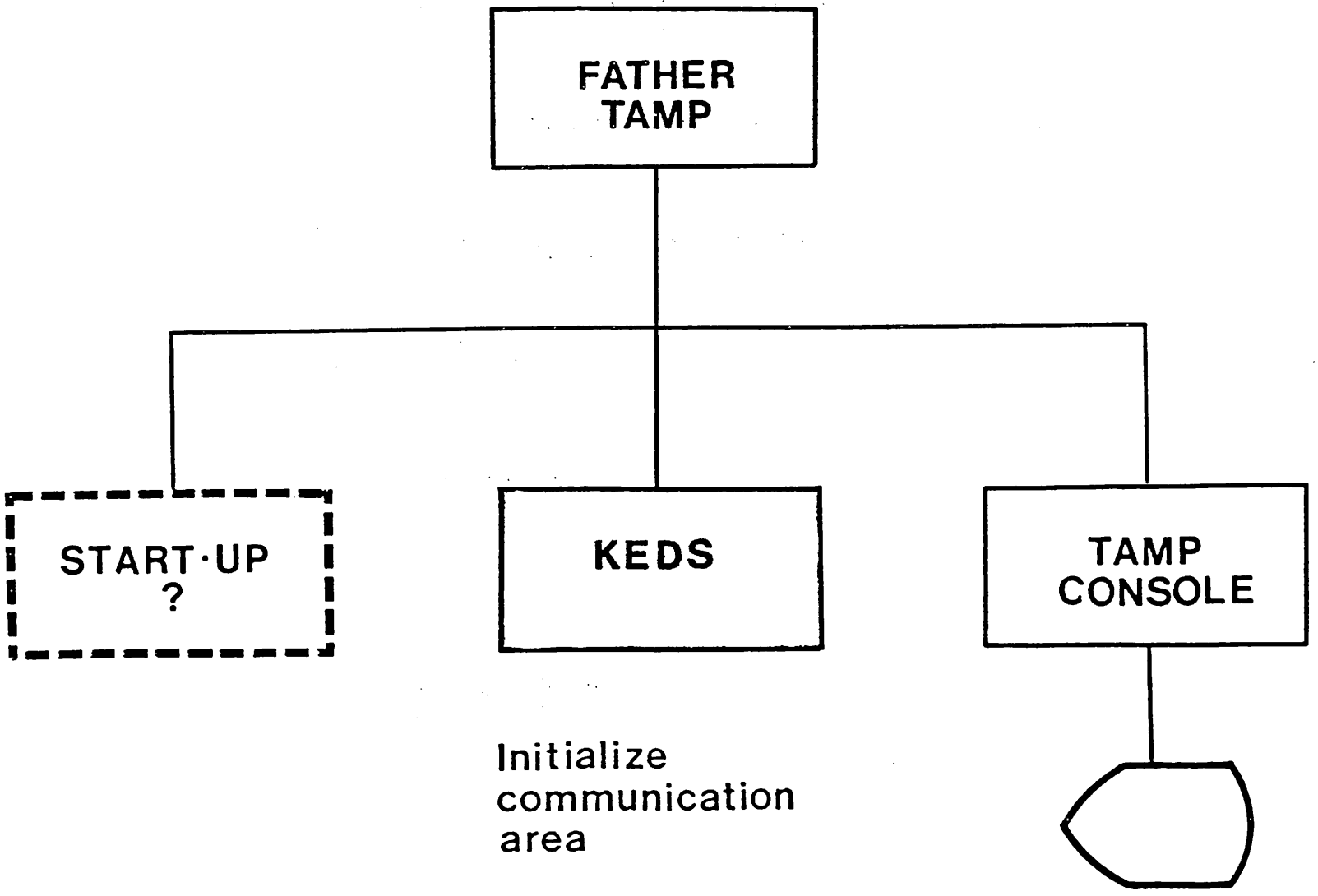
TERMINAL

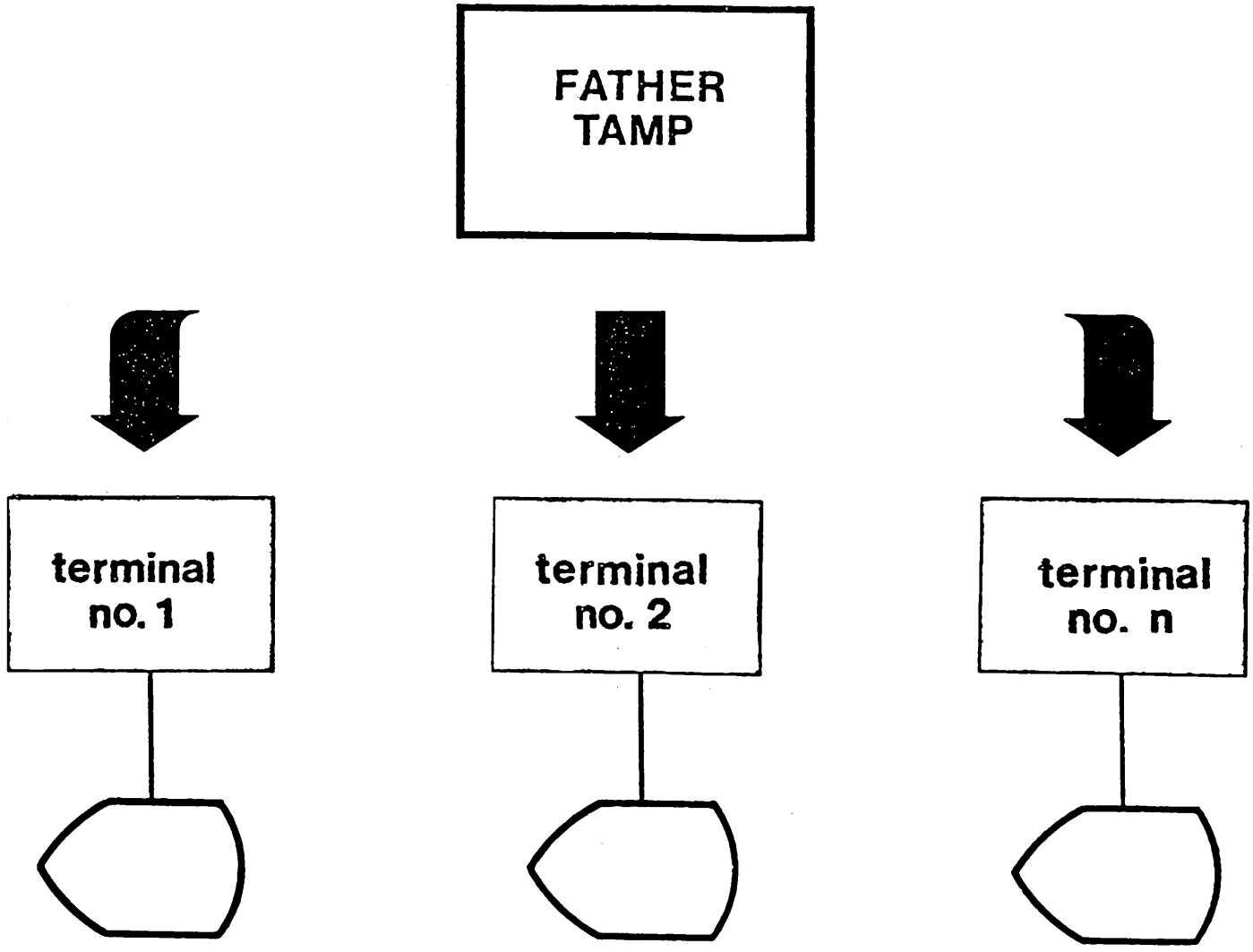
- name
- logical device no.
- hardware data

UTILITY

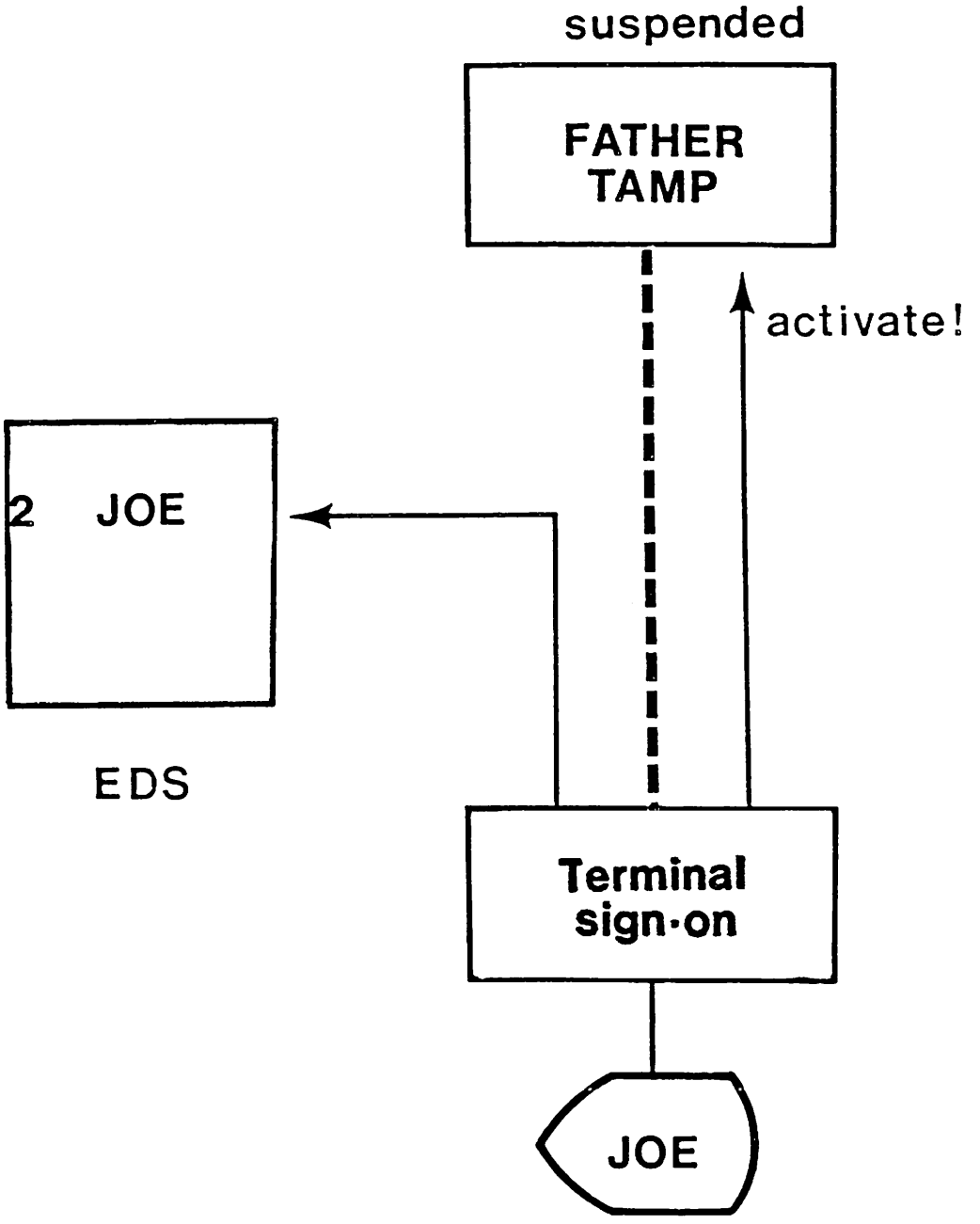
- name
- type
- entry point
- segmented library
- run priority

:RUN TAMP


FATHER
TAMP
START·UP
?
KEDS
TAMP
CONSOLE
Initialize
communication
area



Terminal sign·on processes


suspended
FATHER
TAMP
activate!
2 JOE
EDS
Terminal
sign-on
JOE

FATHER
TAMP
USER
SON
terminal
sign-on
terminate
JOE

USER MENU

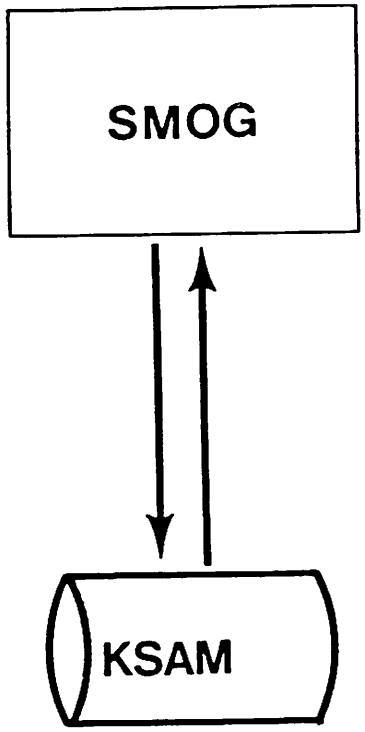
99/99/99
USERNAME

	1	ADDCUST	Add a customer
sk1	2	FUNCNAME	This is a function description
	3	CHNGCUST	Change a customer
sk2	4	REPTCUST	Report on a customer
	5	DELCUST	Delete a customer

G-09.31

SMOG

- Block mode transfers
- Modify protected & unprotected fields
- Modify & reset display enhancements
- Cursor positioning
- Cancel notification
- Field level I/O



Build files

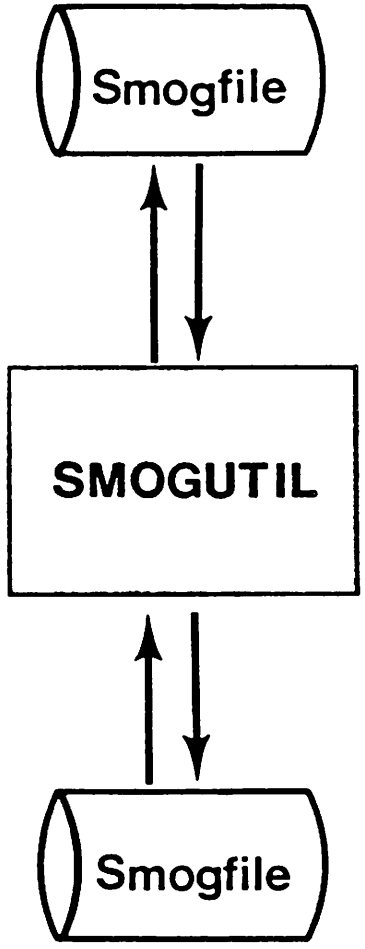
Add screens

Display ”

Modify ”

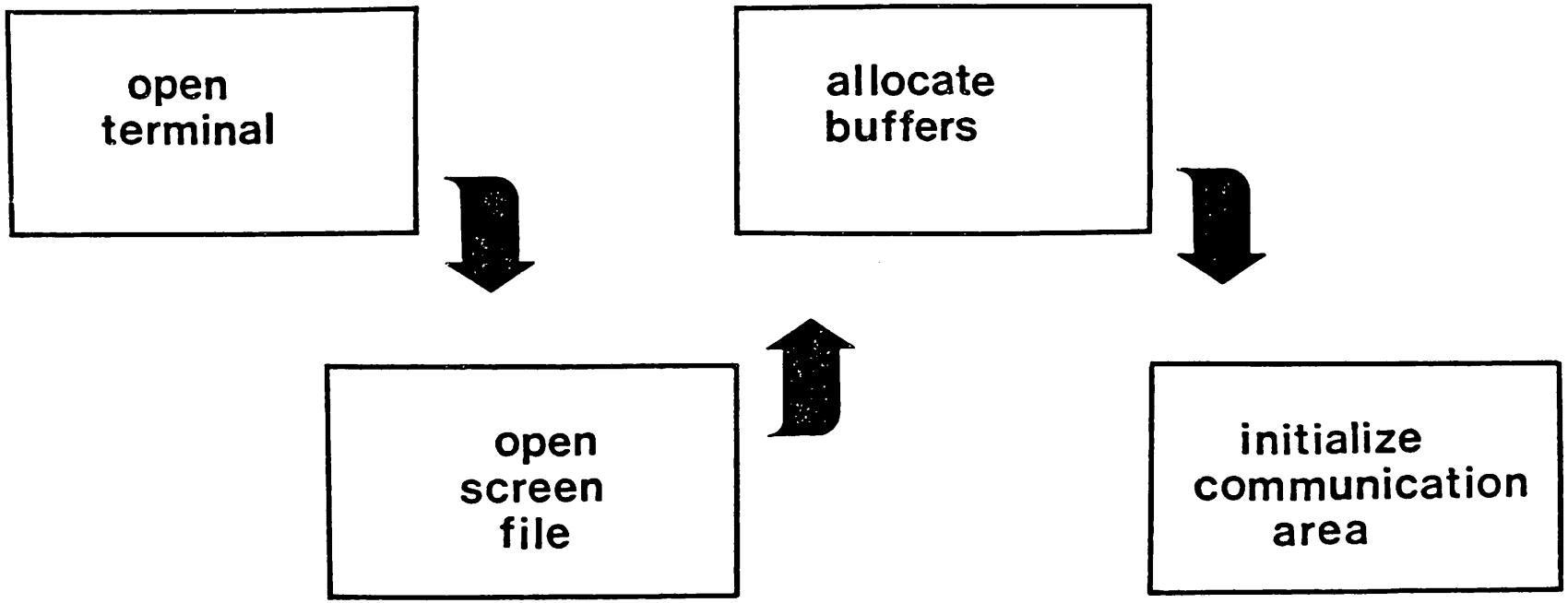
Delete ”

Report ”

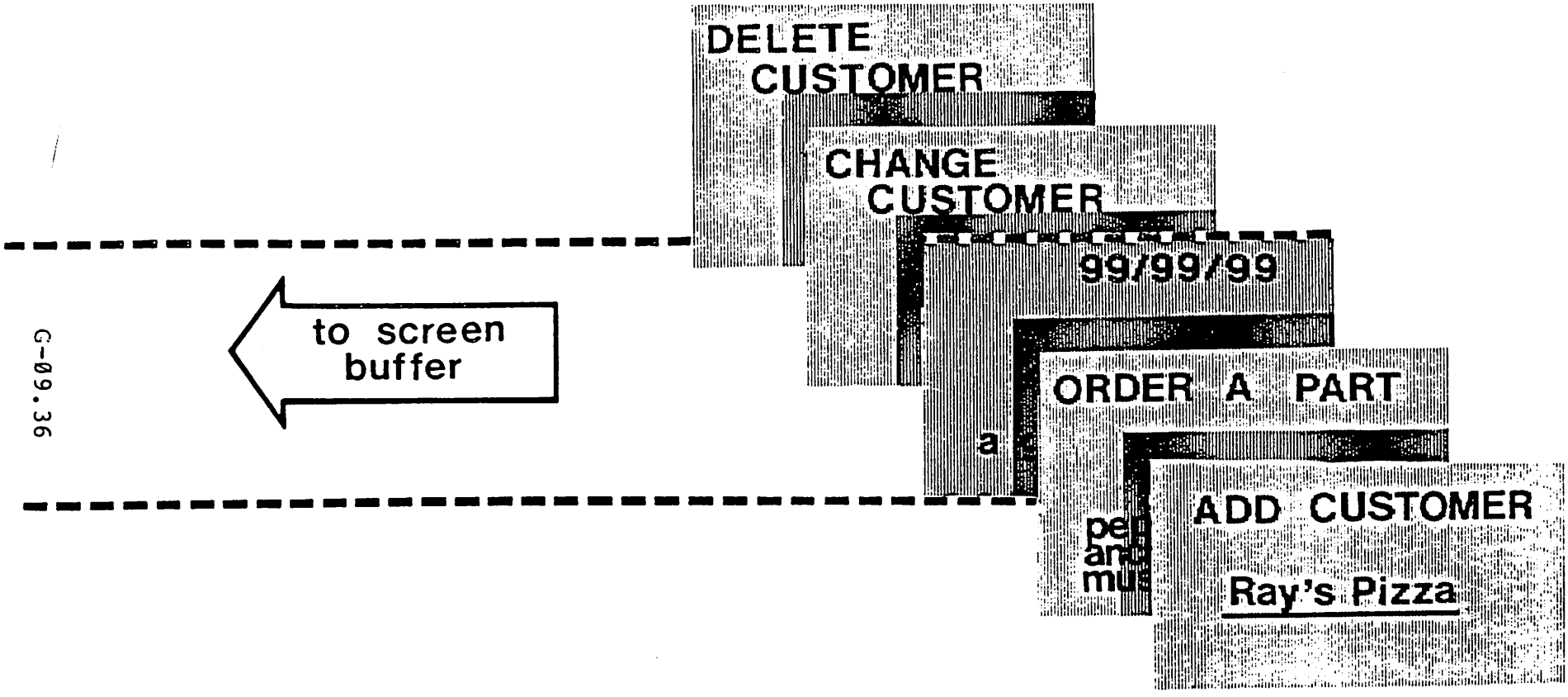


- Copies screens
- Duplicates screens
- Renames screens
- Merges files

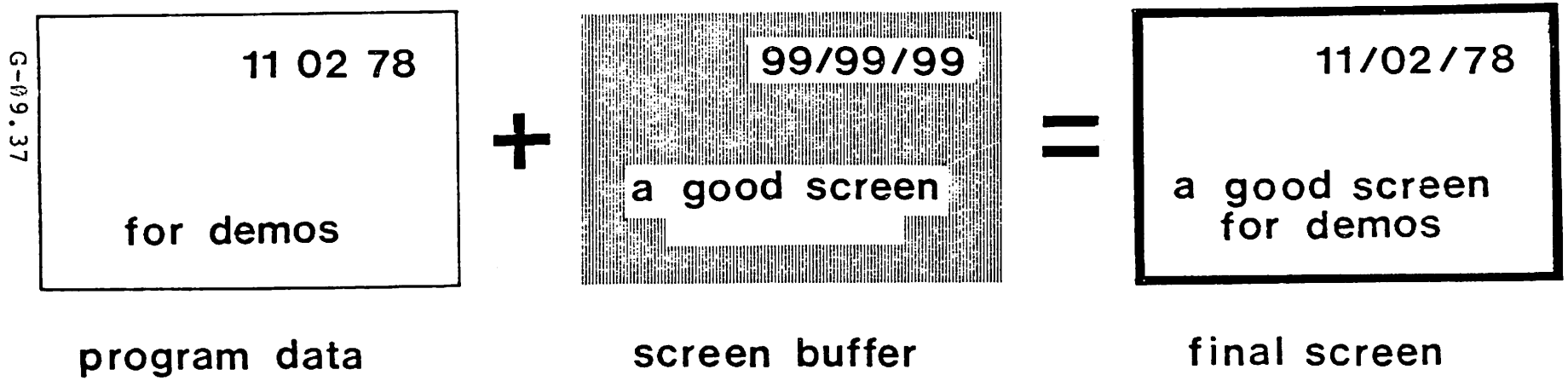
INITERM


open terminal
allocate buffers
open screen file
initialize communication area

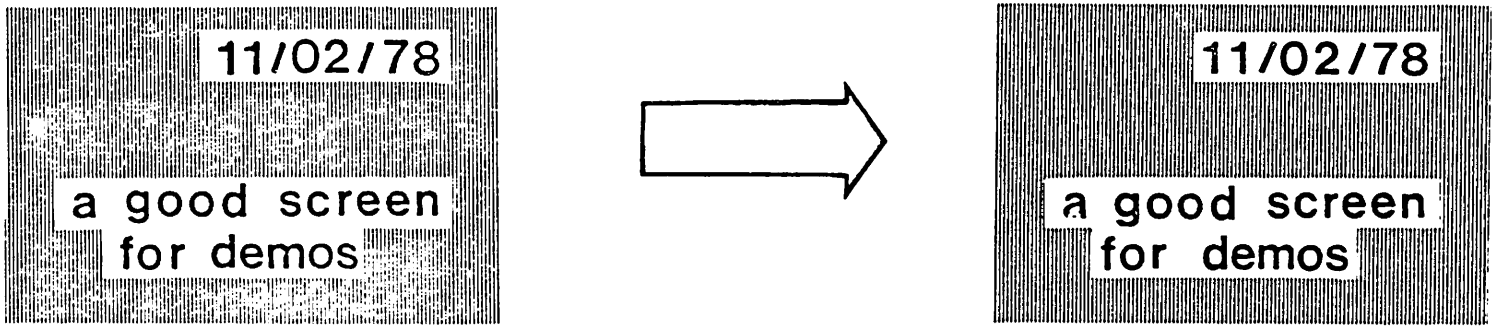
GETSCREEN


DELETE CUSTOMER
CHANGE CUSTOMER
99/99/99
ORDER A PART
ADD CUSTOMER
Ray's Pizza
to screen buffer
G-09.36

MOVEDATA & ENHANCE



PUTERM

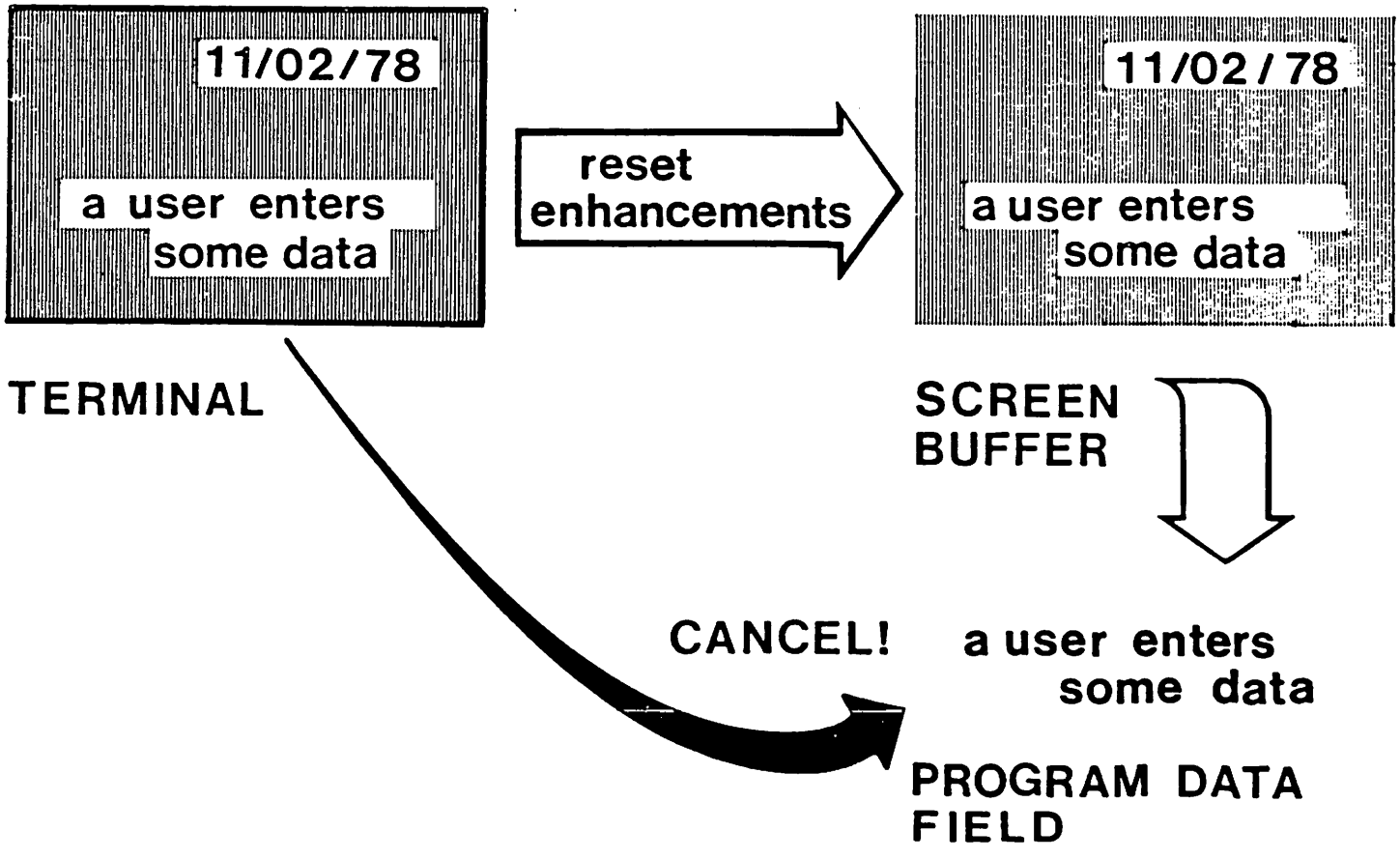


SCREEN BUFFER

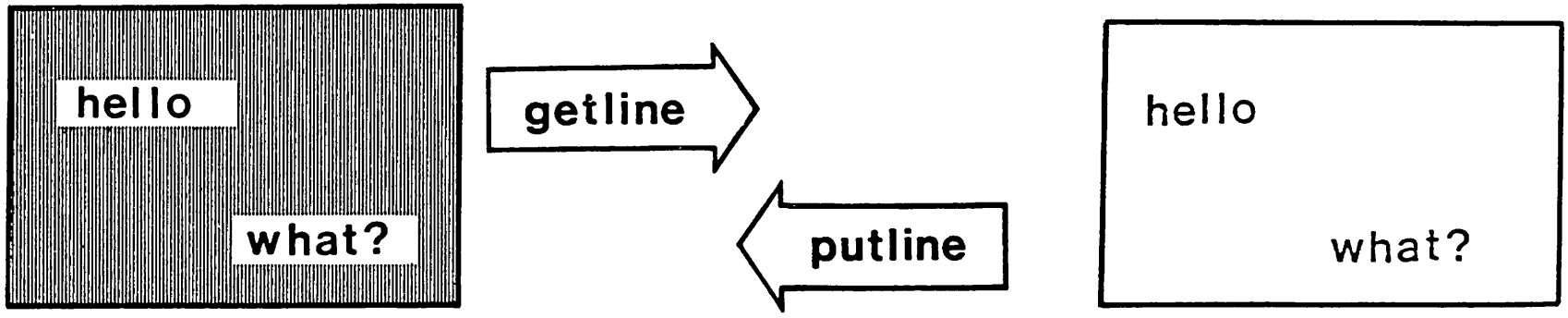
display screen
&
position cursor

TERMINAL DISPLAY

GETERM


11/02/78
a user enters
some data
reset
enhancements
11/02/78
a user enters
some data
TERMINAL
SCREEN
BUFFER
CANCEL!
a user enters
some data
PROGRAM DATA
FIELD

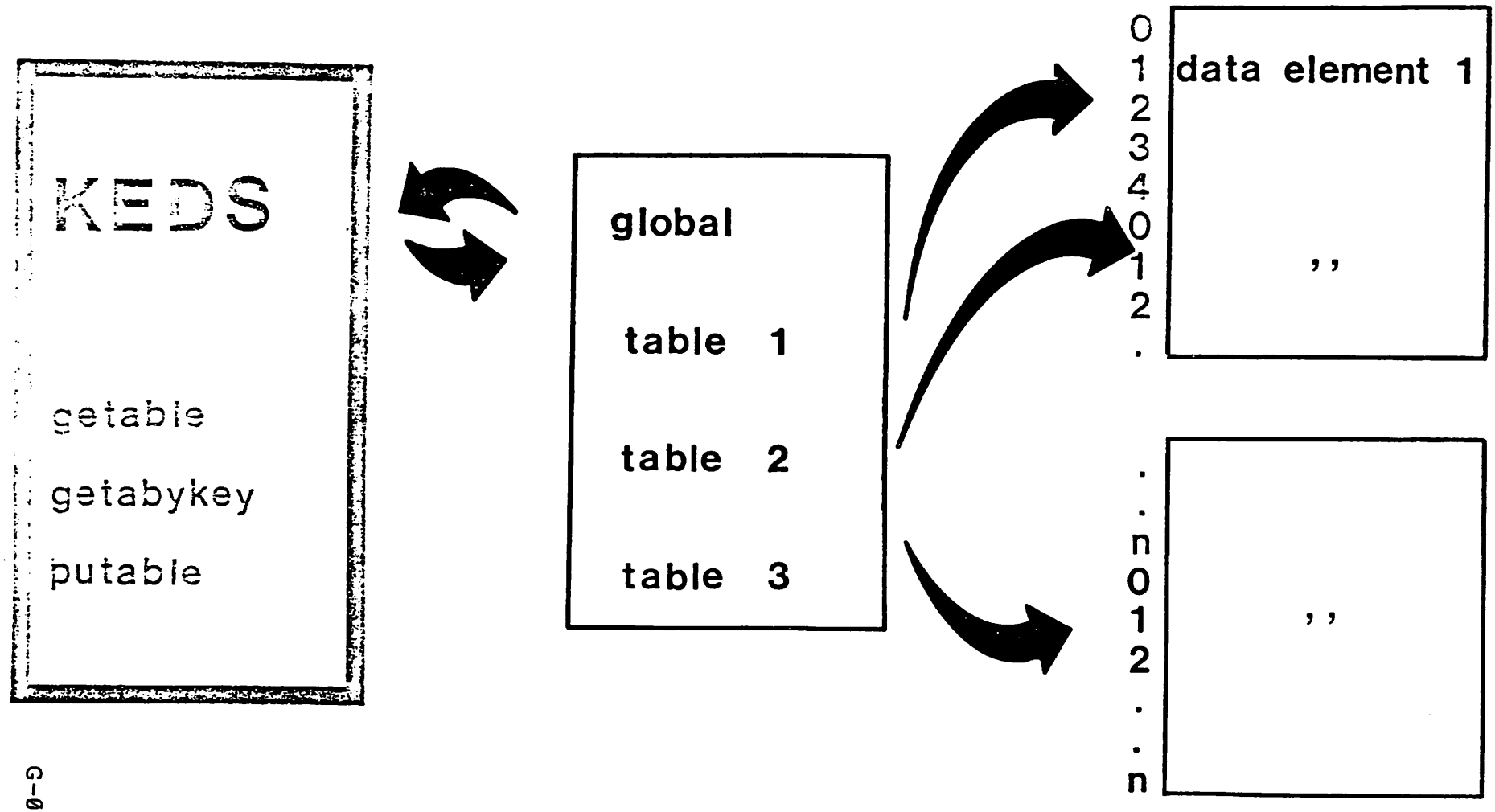
GETLINE & PUTLINE



KEDS

- **MANAGES TABLES IN VIRTUAL MEMORY**

  - Builds tables
  - Shares tables
  - Reads elements
  - Writes/updates elements
  - Reads elements by keyvalues
  - Purges tables



G-09.42

Program Interface

Parent EDS

Children EDS