TPF Users Group
Grapevine Texas

IBM Software Group

Introducing z/TPF!

October 2004

Bob Dryfoos
Jason Keenaghan
Michael Shershin
Mark Gambino
Colette Manoni

AIM Core and Enterprise Solutions

IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Opening

- New application model
  - Large memory
  - Large programs
  - Subroutines
  - Stacks

Opening

- Other z/TPF presentations
  - Applications: Debuggers
  - Applications: Coding today for z/TPF tomorrow - Assembler
  - Languages: GNU Compiler Collection (GCC) Overview
  - Languages: Coding today for z/TPF tomorrow - C/C++
  - Languages: C/C++ single source APARs
  - Operations: Positive feedback enhancements
  - SCP: Scheduler enhancements
  - Database: TPFDF status
- Other demonstrations and education
  - TPF Toolkit
  - Make environment

Agenda

- z/Architecture Overview
- Development Environment
- Application Productivity Enhancements
  - Debug and Test Enhancements
  - Design and Coding Enhancements - z/TPF Globals Support
  - Design and Coding Enhancements
- Availability Enhancements
- New Tuning Capabilities
- Additional Enhancements
- Migration to z/TPF: Concepts and Tools
- Closing

z/Architecture Overview

Architecture Definition

- z900, z990, z800, and z890 machines have 2 architecture modes
  - ESA/390 - TPF 4.1 runs in this mode
    - Limited to 2 GB of storage
  - z/Architecture - z/TPF runs in this mode
    - Designed to exploit larger storage spaces
      - For example, 32 GB, 64 GB, 128 GB, and so on

What is z/Architecture?

- 64-bit addresses
  - Real
  - Virtual
    - Region tables
    - 64-bit entries in page - segment - region table entries
- 64-bit registers
  - 16 general registers
  - 16 control registers
  - 16 floating point registers
- 128-bit PSW
- 8-KB prefix area
- Format 2 IDAWs - 64-bit entries

What is z/Architecture (continued)?

- z/Architecture instruction set is a superset of ESA/390
  - ESA/390 instructions work in z/Architecture
    - Except: No vector instructions
- Addressing modes
  - 64-bit addressing mode
  - 31-bit addressing mode
  - 24-bit addressing mode (not supported in z/TPF)
- Reference...
  - z/Architecture Principles of Operation (SA22-7832)
  - z/Architecture Reference Summary (SA22-7871)

Use of z/Architecture Enables Use of New Features

- Relative Instructions - BRC, LARL
  - Program expansion is easier
- Immediate Instructions
  - Halfword Immediate - AHI, CHI, LHI, and MHI
  - For portions of 64-bit register - IIxx, NIxx, OIxx, TMxx
    - For example TMHH, TMHL, TMLH, TMLL
  - Programs easier to write and more efficient
- Storage-protection override
- C/C++ benefits as well as assembler because compiler takes advantage of these instructions

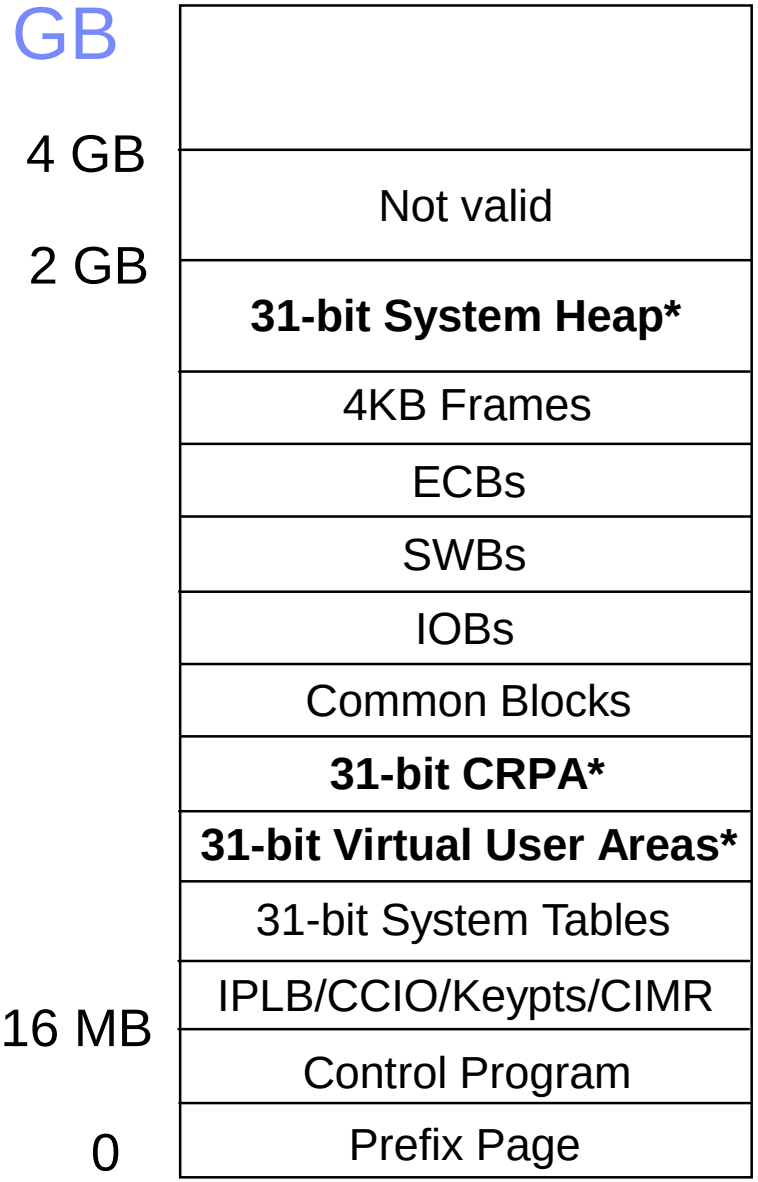
z/TPF Memory Model

- z/TPF has ability to exploit > 2 GB of memory
  - Several existing large system tables moved above 2 GB
  - Most new system tables reside above 2 GB
  - Customer tables can reside above 2 GB
  - Programs can reside and execute above 2 GB
  - Additional diagnostics
  - New tuning capabilities

z/TPF Memory Model (continued)

- Program execution
  - C programs execute in 64-bit addressing mode
    - Mostly recompile of TPF 4.1 source code
  - Assembler programs can execute in 31-bit or 64-bit addressing mode
    - Assembler programs brought from TPF 4.1 with no changes will execute in 31-bit addressing mode
    - Most IBM realtime assembler system programs did not require change and still run in 31-bit addressing mode
    - Expect most assembler programs will not require source changes

z/TPF SVM < 4 GB



...see next page

*** - Use 1-MB frames**

z/TPF SVM > 4 GB

Other I-Streams SVM region/segment/page
VFA
64-bit System Heap*
System Heap Control*
Dump Buffer Area
EVM region/segment/page
1-MB Frames
ECB Trace Tables
ECB Preallocated Areas
64-bit CRPA*
64-bit Virtual User Areas*
TCP/IP Areas*
64-bit System Tables
LDEV Trace
Main I-Stream SVM region/segment/page

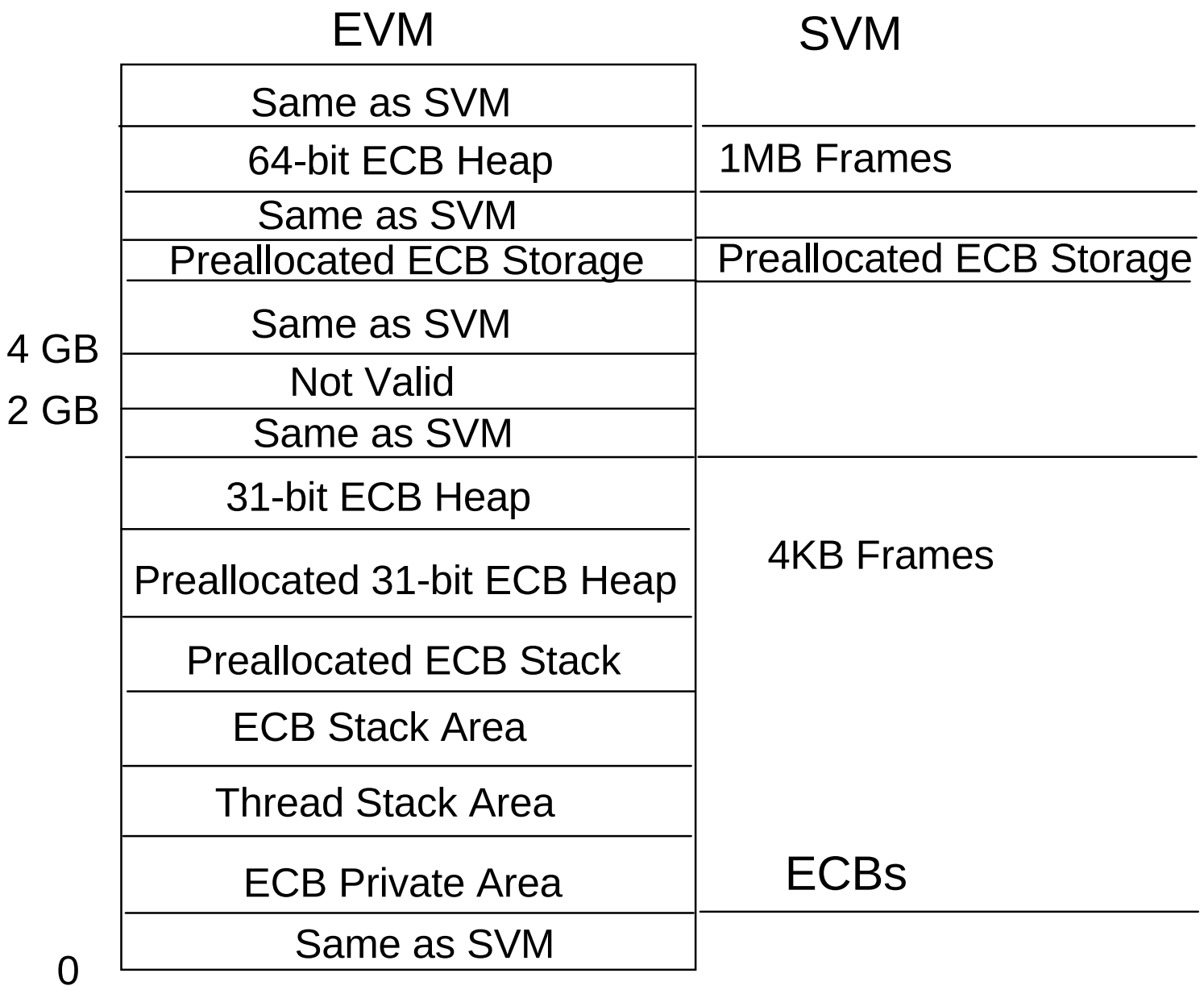
*** - Use 1-MB frames**

4 GB

0

...see preceding page

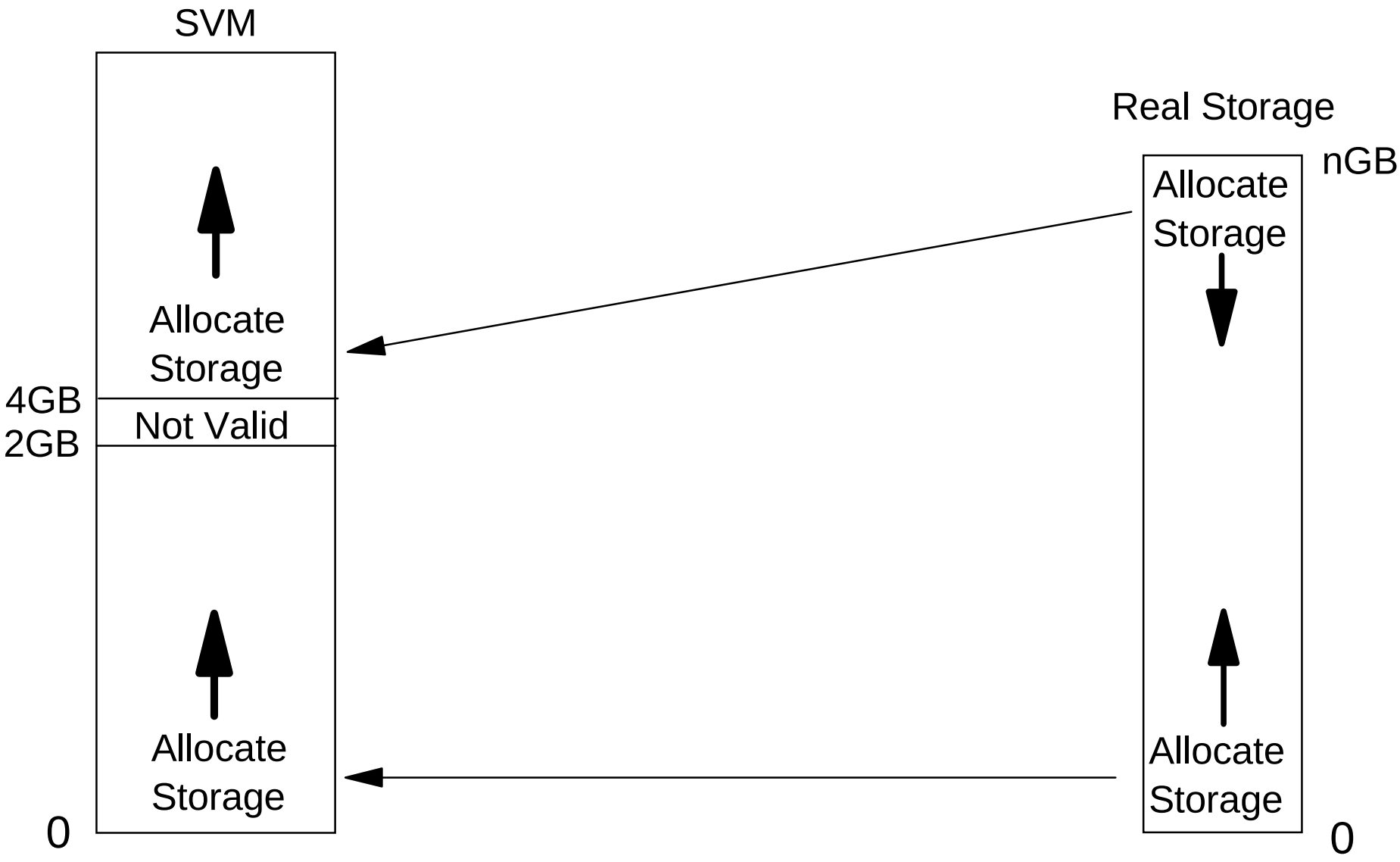
z/TPF EVM



z/TPF Memory Model - Real Address Handling

- SVM address does not equal real address
  - In TPF 4.1 most SVM addresses equaled real addresses
  - In z/TPF SVM is truly virtual
- Do not need > 2 GB of real storage to run z/TPF
  - Important for VM test systems
- z/TPF SVM and EVM do not map 2 GB - 4 GB
  - Real storage is not discarded
  - Intent is to find problems where programs in 31-bit addressing mode pass an incorrect pointer to programs in 64-bit addressing mode
    - Pointer created using BAL, BAS, BALR, BASR instructions.

z/TPF Real Address Mapping



Development Environment and Program Management

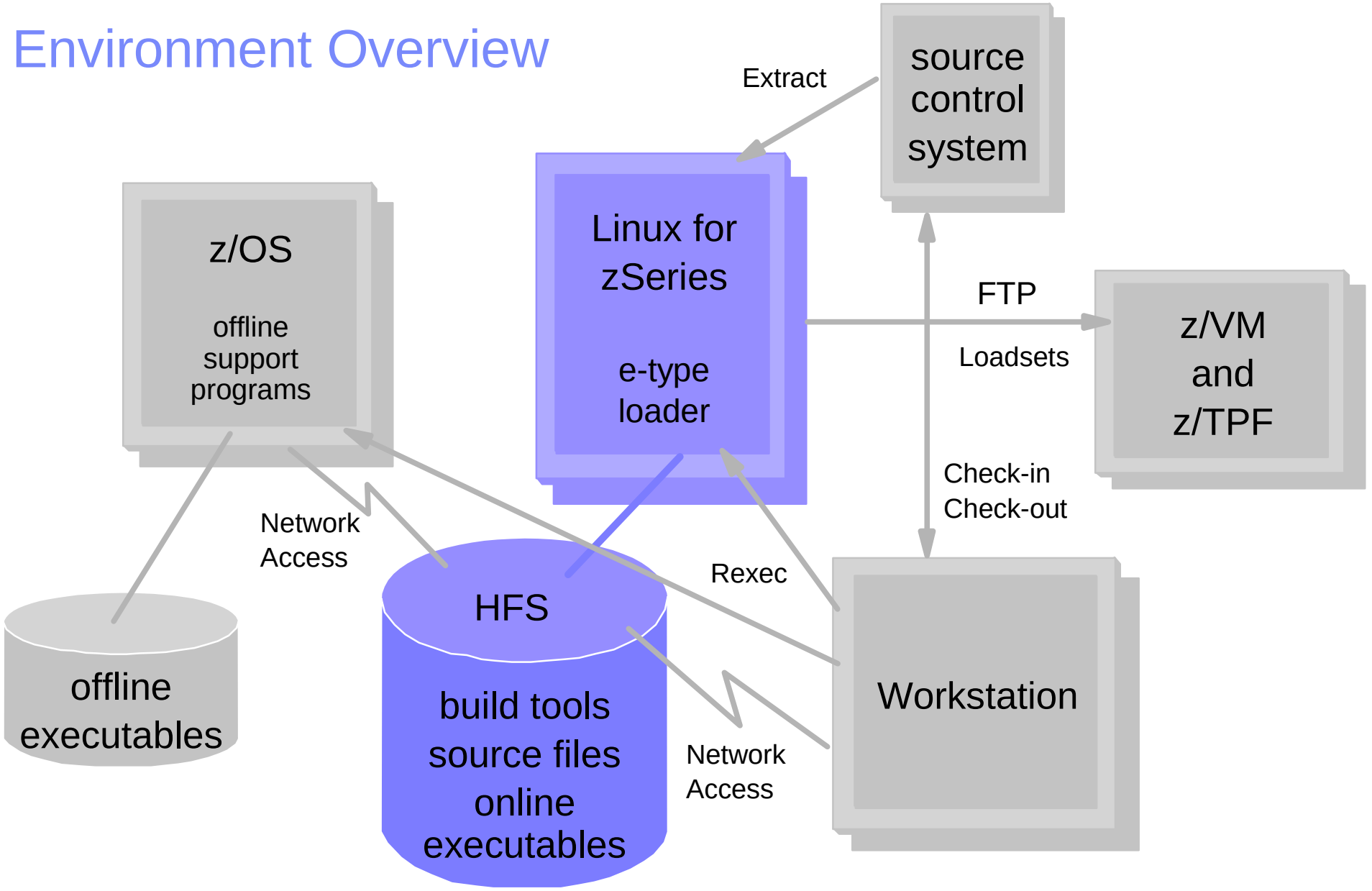
Useful Terms

- FSF : Free Software Foundation
  - Organization that sponsors and controls GNU software
- GNU : GNU's Not Unix
  - Collaborative effort to provide free software
- GPL : GNU Public License
  - License used to ensure that free software remains free
- GCC : GNU Compiler Collection
  - Set of tools used to build programs
  - Supports a number of languages
- ELF : Executable and Linking Format
- SO : Shared Object

Development Environment Objectives

- Open
  - Based on open tools
    - gnumake, korn shell
  - Easy to modify or enhance
- Consistent
  - System and development builds use the same tools
  - MakeTPF integrated with TPF Toolkit for WebSphere Studio
- Configurable
  - Different TPF configurations
  - Layers for development
- Easy to use
  - HFS based

Environment Overview



System and Application Build

- Driven from Linux
  - Compiler, Assembler, Linker, and Loader*
- Uses maketpf makefiles
  - All compile, assemble, and link options
  - Used to build entire system
- Managed by a control file
  - Central point of information regarding all programs
  - Replaces sppgml.mac, ibmpal.cpy, and usrtpf.cpy
- Generates load deck
- Easy to build and maintain the system

C/C++ Compiler Support

- GNU Compiler Collection (GCC)
  - C/C++, assembler, linker, and other utilities
  - Open Source
    - All changes incorporated by the FSF
  - Cross compiler
  - Open standards
    - Executable and Link Format (ELF)
    - Standard Application Binary Interface (ABI)
  - Extensions
    - Imbedded assembler

z/TPF Transformation

- z/TPF modified to match open standards
  - All programs are loaded as ELF
    - C/C++ and TPF Assembler
    - DLMs, DLLs, LLMs, BAL Segments, and CIMR Components
  - Enter/Back modified to be a dynamic loader
  - All program linkages modified to match the ABI
  - All programs use stack as linkage mechanism
- Better integration of C/C++ and BAL
- Improved performance of program linkages
- Easier to manage

Executable and Linking Format (ELF)

- Describes binary file format used by LINUX and documented as part of System V Application Binary Interface (ABI)
  - Tool Interface Standard (TIS) Portable Formats Specification
    - http://www.linuxbase.org/spec/refspecs/elf/TIS1.1.pdf
  - System V ABI Edition with 64-bit ELF update
    - http://www.linuxbase.org/spec/refspecs/elf/gabi41.pdf
- Must be used in conjunction with a processor-specific ABI supplement
  - LINUX for zSeries
    - oss.software.ibm.com/developerworks/opensource/linux390/docu/lzsabi0.pdf

Program Linkage

- Dynamic Linker / Loader
  - Transforms Enter/Back to standard processing
    - Retrieves programs from file
    - Performs relocation
  - Resolves external references to a specific stub at *system level*
    - Handles any interface transformations
    - Provides dynamic hooks
      - E-type loader
      - User exits
      - Trace tools
  - Standard processing with z/TPF efficiency

PAT : Changing Concept of Program Allocation

- No offline allocator
  - SALTBL or TABLE40 are obsolete
  - No more timestamp mismatches
- No need to maintain program order
  - System detects new programs
  - Adds new entries to end
- Dynamically change via command (ZAPAT)
- Maintenance handled via positive feedback controls
  - Online capture of IPAT via command (ZDECK)
  - Offline tool to merge captured information into control file
- Easier to maintain and manage program base

IPAT : Program Attribute Table

- Defined in the control file
  - Fetch type: pre-load, on demand, and default
  - Load type: BSS only, SS only, ALL, SS Shared
  - Debug
  - Timeout value: caller, default, interval
  - Affinity
  - Dump group name
  - Trace group name
  - Authorizations: restrict, montc, key0, cmb, bypass
  - User attributes
- Built from control file using a makefile
- Maintenance can be automated

Load Deck Changes

- No fixed column restrictions
- More freedom in ordering cards
  - using control statements
- More options for search paths
  - Multiple paths for a given component type
  - Override path with program lists
- Output report
  - shows directory in which components were found
  - more helpful diagnostic messages
- New patch format
  - allows validation of data
  - consistent with ZAPGM format
- Option to verify programs against IPAT

Sample TLDR Load Deck

```
@DEFINE
 SYSID=BSS
 &SYS1=/u/ztpf/linuxbin/cur/bss/load
 &APP1=/u/cmast/proj1

@CIMR &SYS1
 CPS0.so (my patches to fix displacement problem)
 @@CCNUCL B2A D0C8 VALDATA-D0C5

@KEYPOINT &SYS1
* Processor Shared Keypoints
CTK6.kpt
* Processor Unique Keypoints for Processor C
CTKA.kpt%C

@VERIFY &SYS1
IPAT.so

@APPLICATION &APP1
@INCLUDE /u/cmast/base/PROJ.BASE
MYP1.so  /* fix for defect 12345 */
MYP2.so
```

Sample TLDR Output

```
@DEFINE
SYSID=BSS
&SYS1 = /u/ztpf/linuxbin/cur/bss/load/
&APP1 = /u/cmast/proj1/
@CIMR &SYS1
CPS0.so (my patches to fix displacement problem)
@@CCNUCL b2a D0C8 VALDATA-D0C5
@KEYPOINT &SYS1
CTK6.kpt
CTKA.kpt %C
@VERIFY &SYS1
IPAT.so
@APPLICATION &APP1
@INCLUDE /u/cmast/base/PROJ.BASE
>QZZ1.so
>QZZ2.so
>QZZ3.so
MYP1.so
MYP2.so

LOAD0007I The default of ELDRCLEAR=NO was accepted.  Any loadsets that were
          loaded previously to the image will remain unchanged.
LOAD0010I Parsing step ended with return code 0.
```

Sample TLDR Output

```
**********************    END PARSING PHASE    **********************

LOAD8070E MYP1.so is not present in the specified IPAT
          and will not be loaded.
LOAD8070E MYP2.so is not present in the specified IPAT
          and will not be loaded.

*****************************************************************
**************************    SUMMARY REPORT    **************************
*****************************************************************

Status of DEFINE Flags:
  SYSID=BSS
  OVERLAY_IPAT=NO
  PROGCLEAR=NO
  ELDRCLEAR=NO
  DEBUGFILES=YES
  WARN_BSO_RELO=NO
  &APP1 = /u/cmast/proj1/
  &SYS1 = /u/ztpf/linuxbin/cur/bss/load/

THE FOLLOWING PAT WAS USED FOR VERIFICATION
/u/ztpf/linuxbin/cur/bss/load/IPAT.so
```

Sample TLDR Output

```
CIMR Compnts      Path              Found In
  CPS0.so         &SYS1             /u/ztpf/linuxbin/cur/bss/load/
          (my patches to fix displacement problem)
     @@CCNUCL b2a D0C8

Keypoints       P Path              Found In
  CTKA.kpt    %C &SYS1              /u/ztpf/linuxbin/cur/bss/load/
  CTK6.kpt      &SYS1               /u/ztpf/linuxbin/cur/bss/load/

App. Pgms         Path              Found In
  MYP1.so         &APP1             /u/cmast/proj1/
  MYP2.so         &APP1             /u/cmast/proj1/
  QZZ1.so         &APP1             /u/cmast/proj1/
  QZZ2.so         &APP1             /u/cmast/proj1/
  QZZ3.so         &APP1             /u/cmast/proj1/

THE FOLLOWING COMPONENTS WERE NOT LOADED.
  APP programs:
    MYP1.so         - NOT PRESENT IN THE SPECIFIED IPAT
    MYP2.so         - NOT PRESENT IN THE SPECIFIED IPAT

LOAD0020I Load step ended with return code 8.
```

Sample OLDR Load Deck

```
@DEFINE
   SYSID=BSS
   &MYPGMS=/u/cmast/proj1         /* define search path */
   &SYS1=/u/ztpf/linuxbin/cur/bss/load   /* path that contains IPAT.so */

@VERIFY &SYS1
 IPAT.so

@LOADSET  BASE1  &MYPGMS
  MYP1.so, MYP2.so
@INCLUDE /u/cmast/base/PROJ.BASE
```

Sample OLDR Output

```
@DEFINE
SYSID=BSS
&MYPGMS = /u/cmast/proj1/
&SYS1 = /u/ztpf/linuxbin/cur/bss/load/
@VERIFY &SYS1
IPAT.so
@LOADSET BASE1 &MYPGMS
MYP1.so
MYP2.so
@INCLUDE /u/cmast/base/PROJ.BASE
>QZZ1.so
>QZZ2.so
>QZZ3.so
LOAD0010I Parsing step ended with return code 0.

**********************    END PARSING PHASE    ******************

LOAD4070W MYP1.so is not present in the specified IPAT.
          The program will be loaded anyway.
LOAD4070W MYP2.so is not present in the specified IPAT.
          The program will be loaded anyway.
```

Sample OLDR Output

```
************************************************************************
************************    SUMMARY REPORT    ************************
************************************************************************

Status of DEFINE Values:
  SYSID=BSS
  WARN_BSO_RELO=NO
  DEBUGFILES=YES
  &MYPGMS = /u/cmast/proj1/
  &SYS1 = /u/ztpf/linuxbin/cur/bss/load/

LOADSETS SUCCESSFULLY PROCESSED:
 Loadset BASE1:
  MYP1.so         &MYPGMS          /u/cmast/proj1/
  MYP2.so         &MYPGMS          /u/cmast/proj1/
  QZZ1.so         &MYPGMS          /u/cmast/proj1/
  QZZ2.so         &MYPGMS          /u/cmast/proj1/
  QZZ3.so         &MYPGMS          /u/cmast/proj1/

LOADSETS SUCCESSFULLY PROCESSED WITH WARNINGS:
 Loadset BASE1:
    MYP1.so         - NOT PRESENT IN THE SPECIFIED IPAT
    MYP2.so         - NOT PRESENT IN THE SPECIFIED IPAT

LOAD0020I Load step ended with return code 4.
```

Applications Productivity Enhancements
Debug and Test Enhancements

ECB Trace Enhancements - C Function Trace

- C function trace is always available
  - Can be used in both test and production environments
  - Enabled by default compiler option
- Function entry and exit can be traced
  - Function entry and macro trace are equivalent
  - Function exit is a separate option
- Formatted macro and C function trace are collated together
- Function trace entries include:
  - Prototype
  - Parameter values
- Ability exists to add free-form text into function trace.

ECB Trace Example

```
ECB TRACE
TRACE     PROG  IS  LOADMOD  MACRO        PARAMETERS
GROUP     MODE        OR           OR
                    OFFSET   PROGRAM INFORMATION
IBM_DEFT            CTAL
               01   +001E1D0 void __serrc(t_serrc status=00000000,int number=0092EA6C,const char* msg=
                             0000000000000000,void ** slist=0000000000000000,char prefix=0xE4,+0001
                              MORE PARAMETERS)
                    CTIS
               01   +000C124 return(void * 0000000480000000) errno=00000
                              from void * malloc64(size_t)
         64PU1 01   +000046C QZZ8  MALOC  S-00000000000F0F78 B-0000000480000000
   R0  00000000 00000000  00000000 00000000  R1    R2  00000000 00000000  00000000 00000500  R3
   R4  00000003 8DBF1110  00000000 000F0F78  R5    R6  00000000 00000082  00000003 8C5E2CF4  R7
   R8  00000003 8C5E2DF0  00000000 CC022800  R14   R15 00000000 0AA0039A

               01   +000C110 void * malloc64(size_t size=00000000000F0F78)
                    QZZ8
               01   +0000B6C return(long int 00000000000F0F78) errno=00000
                              from long int get_size_of_table(char*)
               01   +0000B62 long int get_size_of_table(char* name=000000038C5E2EEA)
                    CTAL
               01   +0023452 return(int 00000049) errno=00000
                              from int tpf_trace_info(char*)
               01   +00233AC This is a test to show that free form text can be added to function trace
```

ECB Trace Enhancements - More Trace Entries

- Number of trace entries per ECB can be set by the user
  - Maximum number of trace entries 65,535
- Provides more diagnostic information for each ECB
  - Can see more of the flow through an application
- Makes debugging easier.

ECB Trace Enhancements - Trace Groups

- Problem: middleware programs can dominate ECB trace
- Solution: trace groups
  - Each ECB has multiple trace buffers
    - Trace buffers per ECB is user settable (min = 4, max = 254)
  - Trace Group = set of programs which share an ECB trace buffer
    - User settable by load module
      - Control File (offline) PAT (online)
    - IBM use
      - MQ has its own trace group = IMQSGRP
      - z/TPFDF has its own trace group = DATABASE
- Makes debugging easier.

ECB Trace Group Example

```
ECB TRACE
TRACE      PROG   IS   LOADMOD   MACRO          PARAMETERS
GROUP      MODE          OR              OR
                       OFFSET    PROGRAM INFORMATION
IBM_DEFT               QXDH
                  01   +00000C6 QXDH   ENTRC   ENTER TO UAC0
   R0   00000000 00000000   00000000 00000000   R1    R2   00000000 00000000   00000000 00000004   R3
   R4   00000000 00000001   00000000 0A308000   R5    R6   00000000 0C6001C0   00000000 0A306EA1   R7
   R8   00000004 00000064   00000000 0A308150   R14   R15  00000000 83745CCA

                       QXCE
                  01   +0000172 QXCE   ENTRC   ENTER TO QXDH
   R0   00000000 00000000   00000000 00000000   R1    R2   00000000 00000000   00000000 00000004   R3
   R4   00000000 00000001   00000000 0A308000   R5    R6   00000000 0C6001C0   00000000 0A306EA1   R7
   R8   00000004 00000064   00000000 0A308150   R14   R15  00000000 83745CCA

DATABASE               UTDF
                  01   +0000E98 UAA0   BACKC   RETURN TO QXCE
   R0   00000000 00000000   00000000 00000000   R1    R2   00000000 00000000   00000000 00000004   R3
   R4   00000000 00000001   00000000 0A308000   R5    R6   00000000 0C6001C0   00000000 0A306EA1   R7
   R8   00000004 00000064   00000000 0A308150   R14   R15  00000000 83745CCA

                       UFTH
                  01   +0009582 return(void) errno=00000
                            from void ufth_btree_interface(void)
```

ECB Trace Enhancements - New ECB Trace Capabilities

- ECB heap trace
  - Records all ECB heap requests (malloc, calloc, realloc, free)
  - Formatted in dumps
  - Number of entries is user settable (max = 32,767)
  - Ability to track heap usage even if ECB trace has wrapped
- ECB data level trace
  - Records name of load module
    - Obtained 4K frame in ECB private area
    - Released 4K frame in ECB private area
  - Formatted in dumps
  - Ability to track ECB private area usage even if ECB trace has wrapped

New ECB Trace Examples

ECB heap trace:

```
ECB HEAP TRACE
LOADMOD   LOADSET          OBJECT NAME
  OR
OBJ DSP   IS ECB SVM  THRDID   HEAP BUFFER ADDR API
CTS7      LOADSET-BASE     OBJECT-cts7.goff
+000012F4 01 0ABE4000 00000000                  FREEC BLOCK=000000000CE00FB0
+000011BA 01 0ABE4000 00000000 000000000CE00FB0 MALOC SIZE=0000000000000044
+00000500 01 0ABE4000 00000000                  FREEC BLOCK=000000000CE00FF0
+000012F4 01 0ABE4000 00000000                  FREEC BLOCK=000000000CE0CFB0
+000011BA 01 0ABE4000 00000000 000000000CE0CFB0 MALOC SIZE=0000000000000044
CMQO      LOADSET-BASE     OBJECT-cxcsmw.cpp
+00000046 01 0ABE4000 00000000 000000000CE02FF8 malloc(size=0000000000007FFE)
CTSC      LOADSET-BASE     OBJECT-ctsc.goff
+0000042C 01 0ABE4000 00000000 000000000CE00FF0 MALOC SIZE=0000000000000008
```

ECB data level trace:

```
*DATA BLK, LEVEL 5 OBTAINED BY PROGRAM-BXAM RELEASED BY PROGRAM-N/A

*AVAIL L1 BLK OBTAINED BY PROGRAM-CVAA RELEASED BY PROGRAM-CVAU
```

ECB Trace Enhancements - New ECB Trace Capabilities

- Trace log
  - Ability to record all macro and C function trace items for an ECB
  - Started by API call - TLOGC, tpf_tracelog_on()
  - Stopped by API call or ECB exit - tpf_tracelog_off()
  - Data written to tape or HFS file
  - Post processed on Linux
    - Output to HFS file

Socket API Trace

- Trace socket API calls
- Entries contain detailed information, including:
  - API input parameters
    - Includes implied parameters like time out values
  - Output, including the API return code
    - *sockerrno()* value included if error return code
  - How long it took the API to be completed
  - Data is displayed in user friendly (readable) format
- Two trace entries created if ECB becomes blocked while processing the API:
  - First entry shows the input and when the API was issued
    - Allows you to determine if an API is pending
  - Second entry shows the output and when the API was completed

There are Actually Two Socket API Traces

- Trace at a per-ECB level:
  - Contains socket APIs issued by this ECB
  - Independent of C function trace
  - Useful in debugging socket application programs
  - Included and formatted in dumps
- Trace at a per-socket level:
  - Contains APIs issued for a particular socket
  - Useful because multiple ECBs can share a socket
  - A socket can be passed from one ECB to another via *activate_on_receipt* (AOR)
  - Display online via new ZSOCK API command
  - Allows you to view recent history of the socket for faster online debugging
    - Can be used in conjunction with individual IP trace

Socket API Trace Example - Compact Format

```
ZSOCK TRACE SOCK-C000002
SOCK0035I 15.26.53 BEGIN SOCKET TRACE FOR 00C00002
                                                  COMPLETION
ECB       API          RC                 PROGRAM  TIME (SEC)  TIME STAMP
07650000  socket       00C00002           QZZQ          0.003  May 15 15:25:39
07650000  bind         0                  QZZQ          0.011  May 15 15:25:39
07650000  listen       0                  QZZQ          0.002  May 15 15:25:39

07650000  accept                          QZZQ                 May 15 15:25:39
07650000  accept       00C00008           QZZQ         13.634  May 15 15:25:52

07650000  accept                          QZZQ                 May 15 15:25:52
07650000  accept       00C00013           QZZQ          5.213  May 15 15:25:57

07650000  accept       00C00014           QZZQ          0.016  May 15 14:25:57

07650000  accept                          QZZQ                 May 15 15:25:57
END OF DISPLAY
```

Socket API Trace Example - Detailed Format

```
ECB-07650000  API-socket        PROG-QZZQ  OFFSET-000058  IS-02  May 15 15:25:39
  type-SOCK_STREAM  prot-IPPROTO_TCP
  RC-00C00002          COMPLTIME-0.003sec

ECB-07650000  API-bind          PROG-QZZQ  OFFSET-000148  IS-02  May 15 15:25:39
  port-5004  ip-9.117.241.1  addrlen-16
  RC-0                 COMPLTIME-0.011sec

ECB-07650000  API-listen        PROG-QZZQ  OFFSET-000334  IS-02  May 15 15:25:39
  backlog-15
  RC-0                 COMPLTIME-0.002sec

ECB-07650000  API-accept        PROG-QZZQ  OFFSET-000520  IS-02  May 15 15:25:39
  addrlen-16  timeout-0
  BLOCKED

ECB-07650000  API-accept        PROG-QZZQ  OFFSET-000520  IS-02  May 15 15:25:52
  port-1027  ip-9.117.232.167
  RC-00C00008          COMPLTIME-13.634sec
```

Network Management - Start with SNMP

- Simple Network Management Protocol (SNMP) tracks exception conditions at the system-wide level for TCP/IP nodes (hosts and routers). For example:
  - MIB variable *ipInDiscards* contains the number of datagrams received by this node that contained no errors, but were discarded anyway (for example, due to lack of buffer space)
    - Useful for determining which routers are overloaded
- SNMP can tell you if a server is having problems. For example:
  - MIB variable *tcpRestransSegs* contains the number of segments retransmitted by this node and *ipReasmReqds* contains the number of IP fragments received by this node
- However, SNMP will not tell you which sockets are causing the most retransmits, or have the most IP fragments flowing
- In addition, SNMP does not track all exception conditions
  - No MIB variable for TCP data received out of order

Network Management - Add Socket Exceptions

- z/TPF tracks exception conditions on a per-socket basis
- Exceptions for a given socket are shown when that socket is displayed (via ZSOCK DISPLAY with FORMAT)
- Use the new ZSOCK EXCEPTION command to determine the sockets with the most total exceptions or most exceptions of a specific type
  - Allows you to identify troubled sockets in realtime rather than having to examine trace data after the fact
  - Helps identity the scope of the problem. For example:
    - One socket
    - Sockets with one local or remote application
    - Sockets with one business partner
    - Many/all sockets
- Turns SNMP into a Superior Network Management Protocol

Problem Determination Example Using Socket Exceptions

1. Large amount of retransmitted packets on z/TPF is detected by:
   - An SNMP manager monitoring z/TPF
   - ZSNMP or ZTTCP DISPLAY STATS command
2. Issue ZSOCK EXCEPTION to gather further information

```
ZSOCK EXCEPTION RETRANS TOP-5
CSMP0097I 12.48.57 CPU-B SS-BSS  SSU-HPN  IS-01
SOCK0042I 12.48.57 BEGIN SOCKET EXCEPTIONS DISPLAY

  RANK     FD          TOTAL    RETRANS  OUT ORDER  FRAG OUT  FRAG IN
  ----  --------  ----------  -------  ---------  --------  -------
     1  00C00011         376      366         10         0        0
     2  00C00256          25       23          2         0        0
     3  00C0014B           7        7          0         0        0
     4  00C00033         935        5          0         0      930
     5  00C001CD           8        5          3         0        0
END OF DISPLAY
```

Dump Formatting Enhancements

- Storage formatting always has:
  - 8-byte core address on left
  - 16-byte hex display in middle
  - Translated display on right
- Translation can be
  - EBCDIC
  - ASCII
  - User defined code page
  - Specified on IDATG or LISTC macro

Dump Formatting Translation Example

```
*AREA REFERENCED BY REGISTER 9  000000000ED37D14  PROTECT KEY     9
000000000ED37000     00000000     00000000     00000000     0ED39000  .............L..
000000000ED37010     00000000     00000000     C3C8D3F1     40404040  ........CHL1
000000000ED37020     40404040     40404040     40404040     C3C8D3F1              CHL1
000000000ED37030     40404040     40404040     40404040     40404040
000000000ED37040     00000004     00000003     00000002     00000000  ................

*DATA BLK          OBTAINED BY PROGRAM-BXDY RELEASED BY PROGRAM-N/A
*BLOCK IN USE
0000000010A08000     C1F277C0     00000000     00000000     00060000  A2..............
0000000010A08010     00000000     00000000     00000000     00000000  ................


*MSG BLK, LEVEL 0
000000000AB06E80     00000040     00212850     00000000     00000000  ... ............
000000000AB06E90     00410100     00E9C9D5     C5E340C1     C4C440E2  .....ZINET ADD S
000000000AB06EA0     60D4E8E2     C5D9E540     D7D9D6C3     60E740D7  -MYSERV PROC-X P
000000000AB06EB0     C7D460D4     E8D7C740     D4D6C4C5     D360E6C1  GM-MYPG MODEL-WA
000000000AB06EC0     C9E340D7     D6D9E360     F9F9F9F9     F9F9F9F9  IT PORT-99999999
```

Dump Formatting Enhancements - Extensions

- Dump processing has ability to call dump formatting extension
- Extension allows customization
  - What is to be dumped
  - Actual formatting of the data
- User extensions are available
- IBM uses extensions
    - MQ uses to dump queue and channel information

Dump Formatting Extensions Example

```
*MQ Profile (QMGR) in Memory:
000000000ECF6000     00000064     E3D7C6D8     D4404040     40404040  ....TPFQM
000000000ECF6010     40404040     40404040     40404040     40404040
000000000ECF6030     40404040     415DB9D6     00000001     00000000      ...O........
000000000ECF6040     00000000     000000B4     0000000A     0000001E  ................
000000000ECF6050     00000005     00001388     00000100     00000032  .......h........
000000000ECF6060     C4C5C1C4     4BD3C5E3     E3C5D94B     D8E4C5E4  DEAD.LETTER.QUEU
000000000ECF6070     C5404040     40404040     40404040     40404040  E


Local Normal Queue DEAD.LETTER.QUEUE, Common=No:

*
000000000ED32048     C4C5C1C4     4BD3C5E3     E3C5D94B     D8E4C5E4  DEAD.LETTER.QUEU
000000000ED32058     C5404040     40404040     40404040     40404040  E
000000000ED32068     40404040     40404040     40404040     40404040
000000000ED32078     00000000     D4D8D8C4     00000001     00000000  ....MQQD........
```

Dump Formatting Enhancements - Branch Trace

- Branch Trace is formatted for last 10 entries
  - Gives name of program and displacement into program
  - For CP, gives name of CP CSECT and displacement into CSECT

```
*LAST 10 BRANCH TABLE ENTRIES          CSECT    OFFSET*
              00000000 801C70BE        CCTAPE   000060BE
              00000000 8102C060        CIO      000014E8
              00000000 801C4852        CCTAPE   00003852
              00000000 8102BD98        CIO      00001220
              00000000 8020D528        CCOSAE   00004528
              00000000 8004C3E8        CCCLHR   000003E8
              00000000 8004CB44        CCCLHR   00000B44
              00000000 8004C330        CCCLHR   00000330
              00000000 80002780        CCNUCL   00002780
              00000000 8011A16C LAST   CCDBAF   0000116C
```

Dump Formatting Enhancements - Application Stack

- Each in use stack frame is formatted in a dump
- Formatting contains:
  - Name of the function using the stack
    - C function name includes prototype
  - Tags to identify fields in the stack, for example:
    - BCH = Back Chain
    - PAT = PAT address
    - ...many more...

Dump Application Stack Formatting Example - C Function

```
*STACK FRAME FUNCTION- void __serrc(t_serrc status,int number,const char* msg,
                                    void ** slist,char prefix,+0001 MORE PARAMETERS)


000000000CC0F490 BCH 00000000     0CC0FC48     00000000     0CD00400  ................
000000000CC0F4A0 R2  00000000     0CC0F710 R3  00000000     0CC0F430  ......7.......4.
000000000CC0F4B0 R4  00000000     0CC0F788 R5  00000000     0013306C  ......7h........
000000000CC0F4C0 R6  00000000     00000002 R7  00000000     0CC0FBF0  ................0
000000000CC0F4D0 R8  00000000     00000001 R9  00000000     0AA00000  ................
000000000CC0F4E0 R10 00000000     00133012 R11 00000000     0092EA6C  .............k..
000000000CC0F4F0 R12 00000000     00002000 R13 00000003     8D8BB888  .............h
000000000CC0F500 R14 00000000     0012990A R15 00000000     0CC0F490  ......r.......4.
000000000CC0F510 F0  00000000     00000001 F2  00000003     8D8BBA10  ................
000000000CC0F520 F4  00000003     8D8B8DE4 F6  00000000     0CC0F4B0  .......U......4.
000000000CC0F530 PAT 00000000     0A201420 CRE 00000003     8D8B7B40  ..............
000000000CC0F540 BAS 00000000     0013306C TRN D8E9E9F8     0CD00500  ........QZZ8....
000000000CC0F550     00000000     00002000     0CC0FC48     8D8BB7F8  ..............8
```

Dump Application Stack Formatting Example - Assembler

```
*STACK FRAME FUNCTION- BXEY
0000000012C139C0 BCH 00000000     12C13C00      00000000     00000001  .....A.........
0000000012C139D0 R2  00000000     10A00000 R3   00000000     00135012  ................
0000000012C139E0 R4  00000000     00000020 R5   00000000     000000F8  ................8
0000000012C139F0 R6  00000000     12C13BC8 R7   0A320010     07F10008  .....A.H.....1..
0000000012C13A00 R8  00000000     10A00008 R9   00000000     12C13BC8  .............A.H
0000000012C13A10 R10 00000000     000000F2 R11  00000000     0013506C  .......2........
0000000012C13A20 R12 00000000     00000008 R13  00000000     00000001  ................
0000000012C13A30 R14 00000000     8D3C6B2A R15  00000000     0FA00000  ................
0000000012C13A40 F0  00000000     12C13C00 F2   00000000     000000F8  .....A.........8
0000000012C13A50 F4  00000000     10A00000 F6   00000000     001351FE  ................
0000000012C13A60 PAT 00000000     0FE030A0 CRE  00000000     00002000  ................
0000000012C13A70 BAS 00000000     0D3C6AD0 TRN  C2E7C5E8     80000000  ........BXEY....
0000000012C13A80     00010000     00C13A08      00000000     0000E5F0  .....A........V0
```

Dump Formatting Enhancements - ECB Heap

- ECB Heap formatting includes:
  - Buffer address
  - Size of buffer
  - Obtaining load module
  - Displacement into load module where malloc() request was done
  - Buffer contents

```
*IN USE HEAP STORAGE
 BUFFADDR-000000000CE02FF8  SIZE-0000000000008008  ECB SVA-0ABE4000  THRD ID-00000000
 GET PROGRAM-CMQO  OFFSET-000000000002F36E  REL PROGRAM-N/A   OFFSET-N/A

000000000CE02FF8     E3E2C840     00000240     01043000     BBE78D09  TSH ... .....X..
000000000CE03008     285644E2     00000111     01F40000     D4E2C840  ...S.....4..MSH
000000000CE03018     000297DE     00000064     00000000     00000210  ..p.............
000000000CE03028     E7D8C840     00000001     D9D8F140     40404040  XQH ...RQ1
```

Dump Formatting Enhancements - z/TPFDF

- Current SW00SR block is formatted
- Pointer to Database Interface Block (DBIFB) in ECB is labeled
  - DBS is the tag name
  - DBIFB is in ECB Heap - included in the dump

Dump Formatting Enhancements - z/TPFDF SW00SR

```
*CURRENT SW00SR BLOCK
000000000C6001C0 BID E2E60000 PGM D8E7C4C8 KLS 00000000 KLS 036152AC  SW..QXDH........
000000000C6001D0 DET 00000000 WID B0750001 INB 001A00BF ILT 000100A5  ..............v
000000000C6001E0 EOR 0000004D     00320000 RBV 0001D980 OP2 06040000  ..........R.....
000000000C6001F0     00000000 ALG 0A30816C     D1000000     00000000  ......a.J.......
000000000C600200     00000000     00000000 FAD 00000000 FAD 435801EE  ................
000000000C600210 IPT 00000000 IPT 0A308150     0000001A     00000000  ......a.........
000000000C600220     00000000     00000000     00000000     00000000  ................
000000000C600230     00000000     00000000 UKY 00000000 ID1 08082600  ................
000000000C600240 ID5 00010000 ID9 00000000 IDF 00000000     00000050  ................
000000000C600250 RTN 00000400 PNB 001A0005     00000000 REC 0A30A61A  ..............w.
000000000C600260 IPA 00000000     001AA000 NLR 00000000 ORD 0000003D  ................
000000000C600270     00000000     00000000 SRV 00000000 BWF 00000000  ................
000000000C600280 BPT 00000000     00000000     00000000     00000000  ................
000000000C6002A0     00000000     00000000     0000B075     00000CE7  ..............X
000000000C6002B0     C4C88001     FDAE0000     00000000     00000000  DH..............
000000000C6002C0     00000000     00000000     00000000     00000000  ................
000000000C600300     00000000     00000000     00000000     C4C6E3C4  ............DFTD
000000000C600310 CAL 00000004 RET 00000000 PM1 00000002 PM2 800200BE  ................
000000000C600320 PM3 0A308150     00000000     00000000     00000000  ..a.............
```

Dump Formatting Enhancements - Program Link Map

- Failing load module's link edit map is dumped for:
  - CTL-1, CTL-2, CTL-3, and OPR-4 dumps
  - SYSDUMP=YES

```
********* ASSOCIATED LINK MAP DATA
        0000000008450E30 OBJECT FILE: /ztpf/bld/bas/obj/uaa0.o
                         COMPILED ON: 2004/10/04 AT 19.12.26
                         COMPILER: HLASM V10 R52
 OFFSET           ADDRESS FUNCTION NAME
00000000 0000000008450E30 UAA0_BSO
00000008 0000000008450E38 UAA9_BSO
00000010 0000000008450E40 UAA2_BSO
00000040 0000000008450E70 UAA0_INT31
00000066 0000000008450E96 UAA0_INTN31
000000BE 0000000008450EEE UAA9_INT31
000000E4 0000000008450F14 UAA9_INTN31
0000013C 0000000008450F6C UAA2_INT31
00000162 0000000008450F92 UAA2_INTN31
```

Dump Formatting Enhancements - Architectural Changes

- 64-bit Registers
- 128-bit PSWs
- 64-bit core addresses

```
*GENERAL REGISTERS
R0  00000000 00000004 00000000 00000001 R1  R2  00000000 0A308000 00000000 0C6001C0  R3
R4  00000000 0A306EA1 00000004 00000064 R5  R6  00000000 0A308150 00000000 83745CCA  R7
R8  00000000 08461408 00000000 0A300000 R9  R10 00000000 00000010 00000000 00001000  R11
R12 00000000 00002000 00000000 FFFFFFFF R13 R14 00000000 00000000 00000000 0C50F4C0  R15

*CONTROL REGISTERS
C0  00000000 1CB4BEF0 00000000 33261007 C1  C2  00000000 00000000 00000000 70000000  C3
C4  00000000 00000000 00000000 00000000 C5  C6  00000000 81000000 00000000 63FF1007  C7
C8  00000000 0000FFFF 00000000 C0000000 C9  C10 00000000 03600000 00000003 92BFFFFF  C11
C12 80000000 010FA760 00000000 63FF1007 C13 C14 00000000 D7000000 00000000 00000000  C15
```

Dump Formatting Architectural Example (continued)

```
*FLOATING POINT REGISTERS
Y0  43277D94 AE415D24 43277DB5 128336B6 Y1  Y2  43277DB5 128336B6 43277DB5 128336B6 Y3
Y4  00000000 00000000 43277DB5 128336B6 Y5  Y6  43277D94 AD7DEDE0 00000000 00000000 Y7
Y8  00000000 00000000 00000000 00000000 Y9  Y10 00000000 00000000 00000000 00000000 Y11
Y12 00000000 00000000 00000000 00000000 Y13 Y14 00000000 00000000 00000000 00000000 Y15
*FLOATING POINT CONTROL REGISTER  C0000000

*CURRENT PSW AT TIME OF DUMP - 07150000 80000000 00000000 08461C54

*PSWS                 OLD                                          NEW
*   RESTART   00000000 00000000 00000000 00000000   0004C001 80000000 00000000 00000F80
*   EXTERNL   07150000 80000000 00000000 08461C50   0404C001 80000000 00000000 0001A488
*   SVC       07150000 80000000 00000000 08482BD4   05040001 80000000 00000000 00094098
*   PROGRAM   07150000 80000000 00000000 08461C54   0004C001 80000000 00000000 000A4008
*   MACHINE   0706C001 80000000 00000000 0004E71C   0000C001 80000000 00000000 000A0000
*   I/O       07150000 80000000 00000000 08461C50   0404C001 80000000 00000000 0102C1C0
```

Owners

- A new mechanism to track users of system resources
  - 32-byte name is given to users of system resources
    - CMBs, SWBs, 4-KB Frames, 1-MB Frames
    - Name associated when resource obtained (GETBC, GSWBC)
  - The name is comprised of
    - 8-byte high level qualifier
    - 8-byte mid level qualifier
    - 16-byte low level qualifier
    - For example ITAPE.CCWTRANS.REAL_CCW_BLOCK.

Owners (continued)

- ECBs can be registered to an owner
  - New API will register the ECB - EOWNRC, tpf_eownrc()
  - All resources obtained by the ECB after registration will use the registered owner name
- Online displays provided
  - ZSTAT OWNER.

ZSTAT OWNER Example

```
ZSTAT OWNER BLOCK-FRM1MB
CSMP0097I 21.49.05 CPU-B SS-BSS  SSU-HPN  IS-01
STAT0023I 21.49.05 BLOCK OWNER DISPLAY
                       IOB    FRAME  COMMON      SWB      ECB      FRM1MB
ALLOCATED             2704     5000     250     1416      300         300
AVAILABLE             2704     4952     247     1376      293         233
 _
                       IOB    FRAME  COMMON      SWB      ECB      FRM1MB
ICRPA                                                                  53
ISYSHEAP                                                               13
IECB                                                                    1
ITCPIP                                                                  1
END OF DISPLAY+
```

Program Trace Name Enhancements

- Standardized mechanism to modify program name used for tracing
  - No longer need to modify program name at 4(,R8)
  - PNAMC, tpf_pnamc() can be used to modify trace name
    - Set your own trace name
    - Use the caller of this routine
    - Use the name in the PAT
  - Automatically setup on enter
- Used:
  - Macro trace
  - Dump messages
  - File a record
  - Logging pool gets and releases.

Dump Message Enhancements

- Message includes:
  - Trace name
  - Load module name
  - Object name
  - Displacement into the object where dump happened
  - If dump is in CP, CP CSECT name
  - If CTL-1, CTL-2, CTL-3, or OPR-4
    - Intruction length code + Program-interruption code
    - Failing instruction
  - If no core available dump
    - Summary of Owners information
- Dump Tape VSN where dump started is displayed.

Dump Message Examples

Standard OPR dump message:

```
CPSE0152E 12.07.10 IS-0001 SS-BSS  SSU-BSS  SE-000499 OPR-IDBC161
010000A TRC-QXEN
CTDF    OBJ-ufgq.goff       000003E6  LOADSET-BASE
DFOPN -- REFERENCE NAME SHORTER THAN 8 BYTES +
```

OPR-4 dump message:

```
CSMP0097I 09.06.59 CPU-B SS-BSS  SSU-HPN  IS-01
CPSE0156I 09.06.59 DUMP SEQUENCE NUMBER 2380 STARTED ON VSN 000230+
CSMP0097I 09.07.56 CPU-B SS-BSS  SSU-HPN  IS-01
CPSE0152E 09.06.59 IS-0001 SS-BSS  SSU-HPN  SE-002380 OPR-I000004
010000B TRC-CVXS
CVXS    OBJ-cvxsf4.goff     00000164  LOADSET-BASE
PSW    07150001 80000000  00000003 8C5EF4CC  PIC 003B ILC 0004 I-50F02070
R0-2   00000000 00000008  00000003 8C5EF3BE  00000000 8CD04780
R3-5   00000000 0CD07028  BBF0067F ED5CAA2A  00000000 0AA0F000
R6-8   00000000 0AA101D8  00000000 0AA00390  00000000 0CC0E3E0
R9-11  00000000 0AA00000  00000000 00014BB0  00000000 00001000
R12-14 00000000 00002000  00000003 8C5EF4AC  00000000 00000000   _
R15    00000000 00000003
```

Dump Message Examples (continued)

No core dump message:

```
CPSE0151T 09.21.21 IS-0001 SS-BSS  SSU-HPN  SE-002579 CTL-I064C02 CATASTROPHIC
                                      CSECT-CCNUCL    0000CA12
NO FRAMES AVAILABLE
                        IOB   FRAME  COMMON     SWB     ECB     FRM1MB
ALLOCATED           0002504 0001000 0000100 0001092 0000050 0000000220
AVAILABLE           0002502 0000000 0000096 0000065 0000047 0000000153
                        IOB   FRAME  COMMON     SWB     ECB     FRM1MB
ITAPE                       0000993
IECB                        0000004
ISYSTEM                     0000003
END OF OWNERS DISPLAY
```

Dump Enhancements - OPR dumps

- Enhanced ability to select core areas in an OPR dump
  - Ability to dump 4-KB of core around each register
  - Ability to include collated macro trace and I/O trace
  - Ability to include prefix pages of all I-streams
  - ZIDOT extended to OPR dumps
    - Ability to include core areas in OPR dumps
      - System tables
      - User tables.

Dump Enhancements - Dump Groups

- Problem: If a core area is to be included in OPR dumps for 50 programs, doing 50 ZIDOTs is time consuming and error prone.
- Solution: Dump Groups
  - Set of programs which share ZIDOT dump characteristics
    - One ZIDOT command includes tables in OPR dumps for many programs
  - Defined in control file
  - Online maintained in PAT
    - ZDPAT to display
    - ZAPAT to change.

Dump Enhancements - Select by CP CSECT

- Problem: If a core area is to be included in all dumps in a functional area of the CP, ZIDOTs for each dump is time consuming and error prone.
- Solution: Select core areas for dumps in the CP by CP CSECT
  - ZIDOT enhanced to support additions and alters
  - IDOTC macro created
  - Some examples...
    - CCTAPE dumps include ITAPE and ISWBUSE
    - CCSONS dumps include IMFST and IIOBS
    - CCSONP dumps include IPOOL and IRIAT
  - Selection used for CTL-1, CTL-2, CTL-3, OPR-4, and SYSDUMP=YES dumps.

Dump Enhancements - Dump by Owner

- Ability to dump blocks in a resource by Owner name
- Example, if ISWBUSE is selected...
  - A tape dump only needs SWBs which are owned by ITAPE
  - A MPIF dump only needs SWBs which are owned by MPIF
- ZIDOT provides ability to select by Owner
  - IDOTB provides same capability.

Dump Enhancements - Named Manual Dumps

- Problem: ZDUMP does not give enough information and ZDUMP ALL gives too much information
- Solution: Named manual dumps
  - ZDUMP has list of manual dumps
    - Predefined list targets specific functions
      - Example: Tape or SNA
    - ZIDOT can alter existing named manual dumps
    - IDOTM macro defines named manual dumps
  - Create your own manual dump
    - ZIDOT can create new named manual dumps
    - Or include IDOTM definitions in CUDP.CPY
    - You target the data that is dumped.

Named Manual Dump Example

Predefined Manual dump:

```
ZDUMP IBMSNA
CSMP0097I 12.47.19 CPU-B SS-BSS  SSU-HPN  IS-01
CPSE0156I 12.47.19 DUMP SEQUENCE NUMBER 2778 STARTED ON VSN 000230+
CSMP0097I 12.47.19 CPU-B SS-BSS  SSU-HPN  IS-01
DUMP0000I 12.47.19 ZDUMP-OK+
CSMP0097I 12.47.19 CPU-B SS-BSS  SSU-HPN  IS-01
CPSE0115I 12.47.19 IS-0001 SS-BSS  SSU-HPN  SE-002778 MANUAL DUMP
010000B TRC-CVFM
CVFM    OBJ-cvfm.goff       0000024E  LOADSET-BASE
NAMED MANUAL DUMP-     IBMSNA   +
```

Named Manual Dump Example (continued)

Create my own Named Manual dump:

```
ZIDOT INCLUDE MYMANUAL IMFST
CSMP0097I 12.54.45 CPU-B SS-BSS  SSU-HPN  IS-01
IDOT0002I 12.54.45 KEYWORD IMFST INCLUDED FOR NAMED MANUAL DUMP - MYMANUAL
IDOT0001I 12.54.45 COMPLETED+

ZDUMP MYMANUAL
CSMP0097I 12.55.44 CPU-B SS-BSS  SSU-HPN  IS-01
CPSE0156I 12.55.44 DUMP SEQUENCE NUMBER 2779 STARTED ON VSN 000230+
CSMP0097I 12.55.44 CPU-B SS-BSS  SSU-HPN  IS-01
DUMP0000I 12.55.44 ZDUMP-OK+
CSMP0097I 12.55.44 CPU-B SS-BSS  SSU-HPN  IS-01
CPSE0115I 12.55.44 IS-0001 SS-BSS  SSU-HPN  SE-002779 MANUAL DUMP
010000B TRC-CVFM
CVFM    OBJ-cvfm.goff       0000024E  LOADSET-BASE
NAMED MANUAL DUMP-     MYMANUAL +
```

Dump Enhancements - No Core Dumps

- CTL-C dump number no longer used
- Unique dump number for each resource
  - 064C00 - System out of SWBs
  - 064C01 - System out of Common Blocks
  - 064C02 - System out of 4-KB frames
  - 064C03 - System out of 1-MB frames
  - 064C04 - System out of ECBs
- Allows customization
  - In TPF 4.1 need to dump all possible storage areas on a CTL-C
  - Ability to dump only what is needed
    - Example ... 064C00 - do not need to dump common blocks.

Dump Enhancements - Additional ZIDOT Enhancements

- Provide ability to take a dump if SERRC is in NODUMP table
  - ZIDOT FORCE
- Provide ability to take a dump if SNAPC is in NODUMP table
  - ZIDOT SNAP FORCE

LDEV Trace Enhancements

- Ability to define the number of trace entries for all LDEVs
  - Does not require a reassembly of IPLB
- Ability to dynamically increase the number of trace entries for a range of SDAs
- Ability to display LDEV trace online
- ZIOTR provides all of these new capabilities.

```
ZIOTR SET COUNT 420 TO 422 575
CSMP0097I 13.34.33 CPU-B SS-BSS  SSU-HPN  IS-01
IOTR0002I 13.34.33 TRACE COUNT SET DISPLAY
THE NUMBER OF TRACE ENTRIES FOR SDA 0420 IS SET TO  575
THE NUMBER OF TRACE ENTRIES FOR SDA 0421 IS SET TO  575
THE NUMBER OF TRACE ENTRIES FOR SDA 0422 IS SET TO  575
END OF DISPLAY+
```

LDEV Online Trace Display Example

```
ZIOTR DISPLAY TRACE 420 15
CSMP0097I 13.37.47 CPU-B SS-BSS  SSU-HPN  IS-01
IOTR0010I 13.37.47 TRACE ENTRIES, SUBCHANNEL 0001000F, LDEV 00000000012E5E00
ISDAC BBF04310 E2D68C17 01 00 00013994 00000004 01D2AC00
ISDAC BBF042C4 C914C41A 01 00 00013994 00000004 01B3CC00                _
INT   BBF0429A 58E8A6DE 01    00C04007 01F5F020 0C000000    001C44E8
SIOSC BBF0429A 58E5A79E 01 00 001C428A 01F5F018          00 80
ISDAC BBF0429A 58E5999E 01 00 001C6176 00000000 001D1720
INT   BBF0429A 58E57A5E 01    00C04007 01F5F020 0C000000    001C44E8
SIOSC BBF0429A 58E2109E 01 00 001C428A 01F5F018          00 80 _
ISDAC BBF0429A 58E2065E 01 00 001C6176 00000000 001D1720
INT   BBF0429A 58E1E15E 01    00C04007 01F5F020 0C000000    001C44E8
SIOSC BBF0429A 58DE6C9E 01 00 001C428A 01F5F018          00 80
ISDAC BBF0429A 58DE609E 01 00 001C6176 00000000 001D1720                _
INT   BBF0429A 58DE355E 01    00C04007 01F5F020 0C000000    001C44E8
SIOSC BBF0429A 58D9421E 01 00 001C428A 01F5F018          00 80
ISDAC BBF0429A 58D934DE 01 00 001C6176 00000000 001D1720
INT   BBF0429A 58D9145E 01    00C04007 01F5F020 0C000000    001C44E8
END OF DISPLAY+
```

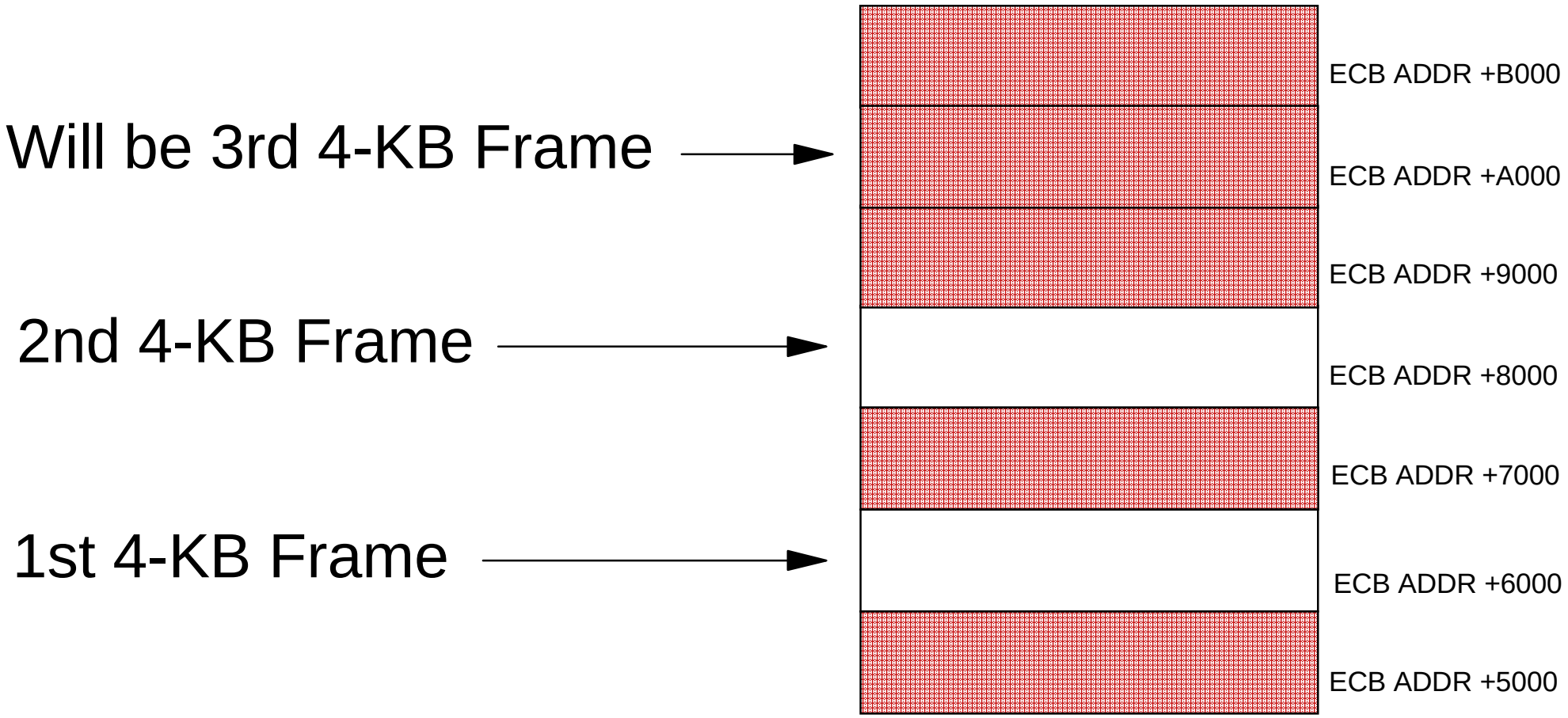
SNAPC Enhancement

- ECB trace can be included in a SNAPC
  - Collated macro and C function trace
  - Last 40 trace items.

Added Protection in ECB Private Area

- 4-KB frame allocation in the ECB Private Area alternates:
  - Valid 4-KB frame
  - Unmapped 4-KB frame
- If a program references beyond the end of a 4-KB block, an OPR-4 will happen
- This is one capability of TPF 4.1 BLKCHK mode.
  - Running BLKCHK mode is processor and memory intensive
- Using this allocation method z/TPF gains some capabilities of BLKCHK mode without the overhead
- Improved ability to detect logic errors as they happen
  - Prevents core corruption.

ECB Private Area Representation



New Diagnostic Tool - ECB Heap HEAPCHECK Mode

- System-wide debugging mode intended for test system use only
- Similar to block check mode in the ECB private area (EPA)
- Intended to detect common programming errors in the usage of the ECB heap *before* applications are released to production environments
  - Applications that write beyond the end of allocated ECB heap buffers
  - Applications that continue to access ECB heap buffers after they have been released
- Allows ECB heap corruption and unauthorized accesses to be detected at the exact moment they occur
- ZSTRC ALTER (NO)HEAPCHECK
  - When turned on/off, the change takes effect immediately for all newly created ECBs. No IPL necessary!
  - All existing ECBs continue to operate with the HEAPCHECK setting that was in effect when they were created.

ZDMAP - Locate a Program

- Displays where a program resides in memory and program size
- Displays the objects in that program, object location, object size, and compile date and time

```
ZDMAP CTS7
CSMP0097I 13.05.38 CPU-B SS-BSS  SSU-HPN  IS-01
DMAP0003I 13.05.38 LINK MAP DATA DISPLAY
                   CTS7   ACTIVE IN LOADSET BASE IN SUBSYSTEM BSS
                   PROGRAM ADDRESS                          000000038CBEE000
                   PROGRAM SIZE                                     00007000

cts7                                      - OBJ FILE AT ADDR 000000038CBEF550
  OBJECT FILE SIZE                                                00001B7C
  COMPILED ON 2004/09/27 AT 08.56.21

END OF DISPLAY
```

ZDMAP - Locate an Object Within a Program

```
ZDMAP COMX OBJ-cgthbn
CSMP0097I 13.05.21 CPU-B SS-BSS  SSU-HPN  IS-01
DMAP0003I 13.05.21 LINK MAP DATA DISPLAY
                   COMX   ACTIVE IN LOADSET BASE IN SUBSYSTEM BSS
                   PROGRAM ADDRESS                          000000038D86A000
                   PROGRAM SIZE                                    00018000

cgthbn                                    - OBJ FILE AT ADDR 000000038D870710
  OBJECT FILE SIZE                                           00000C10
  COMPILED ON 2004/09/27 AT 08.45.24

END OF DISPLAY
```

ZDMAP - Locate the Object Where a Function Resides

```
ZDMAP COMX FUNC-codepage_conv
CSMP0097I 13.07.58 CPU-B SS-BSS  SSU-HPN  IS-01
DMAP0003I 13.07.58 LINK MAP DATA DISPLAY
                   COMX   ACTIVE IN LOADSET BASE IN SUBSYSTEM BSS
                   PROGRAM ADDRESS                          000000038D86A000
                   PROGRAM SIZE                                    00018000

cgthba                                    - OBJ FILE AT ADDR 000000038D86F25C
  OBJECT FILE SIZE                                           00001120
  COMPILED ON 2004/09/27 AT 08.45.25
  OFFSET    ADDRESS           FUNCTION NAME
  00000000  000000038D86F25C  codepage_conv

END OF DISPLAY
```

ZDMAP - Determine the Object and Function Given an Address

```
ZDMAP ADDR-38D86F333
CSMP0097I 13.03.08 CPU-B SS-BSS  SSU-HPN  IS-01
DMAP0003I 13.03.08 LINK MAP DATA DISPLAY
                   COMX   ACTIVE IN LOADSET BASE IN SUBSYSTEM BSS
                   PROGRAM ADDRESS                          000000038D86A000
                   PROGRAM SIZE                                    00018000

cgthba                                    - OBJ FILE AT ADDR 000000038D86F25C
  OBJECT FILE SIZE                                           00001120
  COMPILED ON 2004/09/27 AT 08.45.25
  OFFSET    ADDRESS           FUNCTION NAME
  00000000  000000038D86F25C  codepage_conv

000000038D86F333 IS AT OFFSET 000000D7 INTO OBJECT FILE cgthba

END OF DISPLAY
```

ZDMAP - Display Object File Source Information

```
ZDMAP CTSR INFO-spath
CSMP0097I 13.09.53 CPU-B SS-BSS  SSU-HPN  IS-01
DMAP0003I 13.09.53 LINK MAP DATA DISPLAY
                   CTSR   ACTIVE IN LOADSET BASE IN SUBSYSTEM BSS
                   PROGRAM ADDRESS                          000000038CB95000
                   PROGRAM SIZE                                    00006000

/tpf51/bld/base/obj                        - OBJ FILE AT ADDR 000000038CB96690
  OBJECT FILE SIZE                                             000012FC
  COMPILER INFO - HLASM V10 R52
  COMPILED ON 2004/09/27 AT 08.56.34

END OF DISPLAY
```

ZDMAP - Display Object File Source Information

```
ZDMAP CNSD INFO-SPATH
CSMP0097I 11.51.29 CPU-B SS-BSS  SSU-HPN  IS-01
DMAP0003I 11.51.29 LINK MAP DATA DISPLAY
                   CNSD   ACTIVE IN LOADSET BASE IN SUBSYSTEM BSS
                   PROGRAM ADDRESS                          000000038C77B000
                   PROGRAM SIZE                                     0002E000

/tpf51/bld/base/obj                          - OBJ FILE AT ADDR 000000038C77DABC
  OBJECT FILE SIZE                                                000000C0
  COMPILER INFO -  GCC: (GNU) 3.4-gnupro-04r1-3
  COMPILED ON 2004/09/27 AT 10.17.14
```

Enhanced Disassembler

- Ability to disassemble code in core storage using:
  - ZDCOR
  - ZDPGM
  - TPF Debugger
- Updated to support the z/Architecture instruction set
- Macro parameters are disassembled as well now.

Disassembler Example

```
ZDCOR 38CB96D96.44 INST
CSMP0097I 12.59.54 CPU-B SS-BSS  SSU-HPN  IS-01
DCOR0010I 12.59.54 BEGIN DISPLAY
000000038CB96D96  0A32 0008       RELCC D1
000000038CB96D9A  0A2C 0851       GETCC D1,L4
000000038CB96D9E  E360 1250 0016  LLGF  R6,592(,R1)
000000038CB96DA4  A779 0FFF       LGHI  R7,X'FFF'
000000038CB96DA8  A7F9 0FFF       LGHI  R15,X'FFF'
000000038CB96DAC  0EE6            MVCL  R14,R6
000000038CB96DAE  D703 9080 9080  XC    128(4,R9),128(R9)
000000038CB96DB4  D201 9080 89D4  MVC   128(2,R9),2516(R8)
000000038CB96DBA  0A16 0800       FILEC D1,TAG=Y,GDS=Y
000000038CB96DBE  E3E0 0C28 0017  LLGT  R14,3112
000000038CB96DC4  A7EB 0780       AGHI  R14,X'780'
000000038CB96DC8  E3E0 E000 0004  LG    R14,0(,R14)
000000038CB96DCE  B904 001E       LGR   R1,R14
000000038CB96DD2  A775 00DC       JAS   R7,X'DC'
000000038CB96DD6  D601 1290 1290  OC    656(2,R1),656(R1)
END OF DISPLAY
```

Ability to View Program Listings Online

```
ZDPGM XODD 78.20 LIST
CSMP0097I 09.18.06 CPU-B SS-BSS  SSU-HPN  IS-01
DPGM0010I 09.18.06 BEGIN DISPLAY OF FILE COPY FOR XODD.XODD
                   VERSION TA IN PROGRAM BASE 2 ACTIVE IN LOADSET TESTZ
0078 58F0 9298         3364          GLOBZ REGR=R15
007C 9015 9014         3896          STM   R1,R5,EBW012      SAVE REGS
0080 58E0 9058         3897          L     R14,EBW080        LOAD XTAT TERML
0084 50E0 9064         3898          ST    R14,EBW092        STORE XTAT ENTRY
0088 4860 82C4         3899          LH    R6,XGAH45         LOAD MAX MSG COU
008C 9516 E012         3900          CLI   XZ1IN1,X'16'      IS THIS 23RD XOLD
0090 4770 8028         3901          BNE   XGNLST            NO, BRANCH
0094 4160 000A         3902          LA    R6,10             LOAD MAX OF 10 O
END OF DISPLAY - ZEROED LINES NOT DISPLAYED
```

View Program Listing - Example 2

- Can specify the starting column of the listing to display
- Enables you to view the desired sections of the listing.

```
ZDPGM XODD 78.20 LIST STCOL-30
CSMP0097I 11.50.26 CPU-B SS-BSS  SSU-HPN  IS-01
DPGM0010I 11.50.26 BEGIN DISPLAY OF FILE COPY FOR XODD.XODD
                   VERSION TA IN PROGRAM BASE 2 ACTIVE IN LOADSET TESTZ
         GLOBZ REGR=R15
         STM   R1,R5,EBW012          SAVE REGS
         L     R14,EBW080            LOAD XTAT TERML FLD CORE ADDRESS
         ST    R14,EBW092            STORE XTAT ENTRY ADDR
         LH    R6,XGAH45             LOAD MAX MSG COUNT OF XOLD
         CLI   XZ1IN1,X'16'          IS THIS 23RD XOLD RCD
         BNE   XGNLST                NO, BRANCH
         LA    R6,10                 LOAD MAX OF 10 ON RCD 23
END OF DISPLAY - ZEROED LINES NOT DISPLAYED
```

New Debugger Capabilities

- Can now debug programs built with optimization
  - Can debug same code loaded to production system
- Linkage into DF library standardized
  - Can trace calls into the DF library
- Can use debugger against a captured file
  - Dumps and snapshots of running ECBs can be written to HFS file
  - Debugger makes it easier to locate program variables
    - And other data using expression evaluator
- Can use the trace log utility in conjunction with debugger
  - Can turn trace log on and off from the debugger
  - Allows you to create a trace log between two breakpoints.

Applications Productivity Enhancements
Design and Coding Enhancements
z/TPF Globals Support

Overview of Existing Globals Support

- In TPF 4.1, globals are a portion of fixed main storage that are designed to contain application records; globals permit fast and efficient communication among application programs and between application programs and the control program
- In z/TPF, globals support as it existed in TPF 4.1, remains essentially the same, with a few modifications and enhancements
- Modifications:
  - No longer support 24-bit globals
    - Existing TPF 4.1 global layout is still supported
      - Separate primary and extended global areas, both containing 31-bit globals
  - Global attribute table (GAT) required even if extended global areas are not present

Enhancements to Existing Globals Support in z/TPF

- Easier for application programs to update globals
  - C/C++ applications now have the ability to modify protected globals directly
  - Existing TPF 4.1 methods to modify globals are still supported
    - glob_modify( ), glob_update( ), and global( )
- Faster global restart
  - Reduces the time required to IPL the z/TPF system
- New synchronization option
  - Provides capability for applications to receive notification when global synchronization has completed across all processors in the z/TPF complex
    - Enhances data integrity and system availability
  - New assembler and C/C++ API options
    - ASM: SYNCC macro with WAIT=YES|NO
    - C/C++: tpf_glob_sync_wait( ) function

Globals Support in z/TPF

- ***Format-1 globals:*** The same globals support that exists in TPF 4.1
  - 4 global areas (GL1, GL2, GL3, and GL4)
  - Global storage allocator (GOA) records on DASD
  - Global directories (GL0BA and GL0BY) in main storage
  - Application macros: GLOBZ, GLOUC, FILKW, SYNCC, etc.
- ***Format-2 globals:*** New globals support for z/TPF
  - Used and managed independently of format-1 globals
  - Enhanced features unique to z/TPF

Format-2 Globals Design Initiatives

- Exploit 64-bit architecture
- No impacts to existing applications
- Make globals easy to use
- Provide robust functionality
- Enable globals to be highly extensible
- Offer an easy method for migrating existing applications to the new globals support

Similar Attributes Between Format-1 and Format-2 Globals

- SSU-unique or SSU-shared
- IS-unique or IS-shared
- Processor-unique or processor-shared
- Protected or unprotected
- Keypointable or non-keypointable
- Synchronizable or non-synchronizable

Unique Features of Format-2 Globals Support

- Global records only; no global fields
- Reside in dynamically allocated main storage
- Can reside above or below 2-GB bar
- Managed by a new set of z/TPF commands
- Accessed by new z/TPF-unique APIs
- Keypointable globals may be > 1055 bytes
- Synchronizable globals may be > 1055 bytes
- No limit on the size of global data
- No limit on the number of global definitions
- Access format-1 global data with new format-2 globals APIs

Much Easier to Manage Format-2 Globals

- New set of z/TPF commands that are used to manage format-2 globals
  - No SIP requirements
  - No PILOT tapes
  - No macros or header files that must be kept current
- ZGLBL command
  - Define, alter, and delete format-2 globals definitions
  - Initialize format-2 globals data
  - Define and display z/TPF system control information
- ZDGBL command
  - Display attributes for a format-2 global
  - Display data for a format-2 global
- ZAGBL command
  - Alter data for a format-2 global

Defining Format-2 Globals

- ZGLBL GLOBAL DEFINE
  - Create a format-2 global definition
  - Global names are 8-characters long
    - Valid characters: A-Z, a-z, 0-9, and '_'
    - Padded on the right with blanks if less than 8 characters
  - Attributes
    - Uniqueness: SSU, IS, processor
    - Load: restart, cycle-up, on-demand
    - Protected or not protected
    - Location: 31-bit or 64-bit
    - Keypointable, synchronizable, or neither
  - LIKE parameter: define a new global with same attributes as an existing global

Defining Format-2 Globals Example

- Only need to issue the ZGLBL GLOBAL DEFINE command once
  - The global definition is viewable by all SSUs within the SS, all processors in the L/C complex, and all I-streams
  - The global *cannot* be accessed by an application until it has been initialized with data
- ZGLBL GLOBAL DEFINE example:

```
ZGLBL GLOBAL DEFINE _Glob2 LOC-64 SSU-N IS-N PROC-Y KEY-Y SYNC-N PROT-Y
LOAD-DEMAND
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
GLBL0050I 13.35.22 FORMAT-2 GLOBAL RECORD _Glob2 DEFINED SUCCESSFULLY


ZGLBL GL DEF _myglob LIKE-_Glob2
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
GLBL0050I 13.35.22 FORMAT-2 GLOBAL RECORD _myglob DEFINED SUCCESSFULLY
```

Initializing Format-2 Globals

- ZGLBL GLOBAL INITIALIZE
  - Initialize the data for a format-2 global
  - Must initialize global before it can be used
  - Specify the source from which the global is to be initialized
    - ASDEFINED: reserves specified amount of storage only
    - ZERO: reserves specified amount of storage and clears it
    - GLOBAL: copies data from another format-2 global
    - INPUTDECK: uses global data input deck created online or offline
  - z/TPF system reserves storage, both in memory and on DASD, for the new global data

Creating Format-2 Globals Data Input Deck

- Use a global data input deck with ZGLBL GLOBAL INITIALIZE
- User can use any mechanism to create the input deck
  - Create the input deck on any offline platform and transfer it to the online z/TPF system
    - GLINIT: sample tool available to build global data input deck
      - z/OS: JCL interface creates a general data set (GDS), HFS, tape, or VRDR
      - LINUX: command line interface creates HFS
  - Create the input deck on the online z/TPF system itself

Initializing Format-2 Globals Example

- Create global data input deck on LINUX, and FTP it to the z/TPF file system into file /temp/globaldata/_myglob.data
- Define a temporary data definition name (DDNAME) on z/TPF

```
ZDSMG DEF MYGLOBDDNAME HFS-'/temp/globaldata/_myglob.data'
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
DSMG0001I 13.35.22 DDNAME MYGLOBDDNAME   DEFINED
```

- Use the DDNAME as input to ZGLBL GLOBAL INITIALIZE command

```
ZGLBL GLOBAL INITIALIZE _myglob SOURCE-INPUT INPUT-MYGLOBDDNAME PROC-ALL
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
GLBL0200I 13.35.22 FORMAT-2 GLOBAL RECORD _myglob SUCCESSFULLY INITIALIZED
```

Reinitializing Format-2 Globals

- ZGLBL GLOBAL INITIALIZE can also be used to dynamically reinitialize the contents of a format-2 global while the z/TPF system is running (even in NORM state)
- Allows global data to be reloaded multiple times in a month, week, or even a single day, without disrupting the flow of the z/TPF system
- Backup copy of the original data is stored on DASD
- Backup copy of the original data remains in memory until it is no longer being used
- After the reinitialization, all new accesses of the format-2 global will occur for the new data

Miscellaneous Format-2 Globals Commands

- ZGLBL GLOBAL ALTER
  - Alter attributes for an existing format-2 global definition
- ZGLBL GLOBAL DELETE
  - Delete an unused format-2 global definition
- ZGLBL GLOBAL UNDO
  - Reverse an unwanted or unintentional initialization or deletion
  - Undo of initialization or deletion can be processed up until the user-specified expiration time limit has elapsed
- ZGLBL GLOBAL RELEASE
  - Force release of backup copies of global data on DASD

More Miscellaneous Format-2 Globals Commands

- ZGLBL CANCEL and ZGLBL CONTINUE
  - Operator confirmation for certain actions
- ZGLBL SET
  - Set subsystem-wide format-2 globals control values (eg, keypoint interval, expiration time limit, and load preference)
- ZGLBL DISPLAY
  - Display subsystem-wide control values
  - Monitor resource utilization

Displaying Format-2 Globals

- ZDGBL
  - Display an individual format-2 global's attributes
  - Display all format-2 globals defined to a subsystem
  - Display the data for an individual format-2 global
  - Redirect output of display to an HFS file
    - Create log of global definitions and attributes
    - Capture an individual global's data on-demand

Displaying a Format-2 Global's Attributes Example

```
ZDGBL _glob1
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
DGBL0025I 13.35.22 GLOBAL RECORD DISPLAY
        GLOBAL NAME:        _glob1
        SUBSYSTEM USER:     HPN
        ISTREAM NUMBER:     1
        ADDRESS:            0000000880184000
        SIZE (BYTES):       0000000000001000
        RECORD TYPE:        #IF2G
        GORD ORDINAL:       0000000000000002

        LOCATION:           64BIT
        PROTECTED:          NO
        SSU:                SHARED
        PROC:               SHARED
        IS:                 MEMORY
        LOAD:               RESTART
        KEYPOINTABLE:       NO
        SYNCHRONIZABLE:     YES
        CONTROLLED BY:      SYSTEM
END OF DISPLAY+
```

Displaying All Format-2 Globals Example

```
ZDGBL ALL
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
DGBL0026I 13.35.22 BEGIN DISPLAY ALL FORMAT-2 GLOBAL RECORDS IN BSS SUBSYSTEM

GLOBAL    PROC  SSU  IS  PROT  LOC  LOAD  KEYPT  SYNC  CONTROL
_glob1      N    N    M    N    64   RES    N      Y    SYSTEM
_myglob     Y    Y    Y    Y    64   DEM    Y      N    SYSTEM
_Glob2      Y    N    N    Y    31   CYC    Y      N    SYSTEM

END OF DISPLAY +
```

Displaying Format-2 Global Data Example

```
ZDGBL _glob1 0.30 COPY-B
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
DGBL0010I 13.35.22 BEGIN DISPLAY
                   FILE COPY FOR _glob1 PROC-B SSU-HPN IS-1
 0000000000000000- A9E3D7C6 89A2C799 8581A3A9 E3D7C689 zTPFisGr eatzTPFi
 0000000000000010- A2C79985 81A3A9E3 D7C689A2 C7998581 sGreatzT PFisGrea
 0000000000000020- A3A9E3D7 C689A2C7 998581A3 A9E3D7C6 tzTPFisG reatzTPF
                   CORE COPY FOR _glob1 PROC-B SSU-HPN IS-1
 0000000880184000- A9E3D7C6 89A2C799 8581A3A9 E3D7C689 zTPFisGr eatzTPFi
 0000000880184010- A2C79985 81A3A9E3 D7C689A2 C7998581 sGreatzT PFisGrea
 0000000880184020- A3A9E3D7 C689A2C7 998581A3 A9E3D7C6 tzTPFisG reatzTPF
END OF DISPLAY - ZEROED LINES NOT DISPLAYED+
```

Modifying Format-2 Global Data Example

- ZAGBL
  - Change the data for an individual format-2 global
  - Change the data in memory, on DASD, or both

```
ZAGBL _glob1 0 C1C2C3C4 VALDATA-A9E3D7C689A2C799 COPY-c
CSMP0097I 13.35.22 CPU-B SS-BSS  SSU-HPN  IS-01
AGBL0010I 13.35.22 BEGIN DISPLAY
                   CORE COPY FOR _glob1 PROC-B SSU-HPN IS-1
 0000000880184000- A9E3D7C6 89A2C799 8581A3A9 E3D7C689 zTPFisGr eatzTPFi
ALTERED TO -  _
 0000000880184000- C1C2C3C4 89A2C799 8581A3A9 E3D7C689 ABCDisGr eatzTPFi
END OF DISPLAY - ZEROED LINES NOT DISPLAYED+
```

Format-2 Globals Subsystem User (SSU) Groups

- Allow SSU-unique globals to be shared among a subset of SSUs
- Data resides in the "owning" SSU
- Manage SSU groups with z/TPF commands
  - ZGLBL SSUGROUP DEFINE
  - ZGLBL SSUGROUP ALTER
  - ZGLBL SSUGROUP DELETE
  - ZGLBL DISPLAY SSUGROUP
- SSU-unique globals may include as many predefined SSU groups as necessary
  - ADDGROUP parameter on ZGLBL GLOBAL DEFINE and ZGLBL GLOBAL ALTER
  - REMGROUP parameter on ZGLBL GLOBAL ALTER

Using Format-2 Globals SSU Groups

```
WP/ZGLBL SSUGROUP DEFINE group1 OWNER-WP2 ADDSSU-WP1.WP2
CSMP0097I 12.25.50 CPU-B SS-WP   SSU-WP1   IS-01
GLBL0700I 12.25.50 FORMAT-2 GLOBAL SSU GROUP group1 DEFINED SUCCESSFULLY

WP/ZGLBL GLOBAL DEFINE _globwp LOC-31 SSU-Y PROC-N IS-N KEY-N SYNC-N
ADDGROUP-group1
CSMP0097I 12.25.50 CPU-B SS-WP   SSU-WP1   IS-01
GLBL0050I 12.25.50 FORMAT-2 GLOBAL RECORD _globwp DEFINED SUCCESSFULLY

WP/ZGLBL GLOBAL INIT _globwp SO-ZERO SIZE-5000 SSUGROUP-group1
CSMP0097I 12.25.50 CPU-B SS-WP   SSU-WP1   IS-01
GLBL0200I 12.25.50 FORMAT-2 GLOBAL RECORD _globwp SUCCESSFULLY INITIALIZED
```

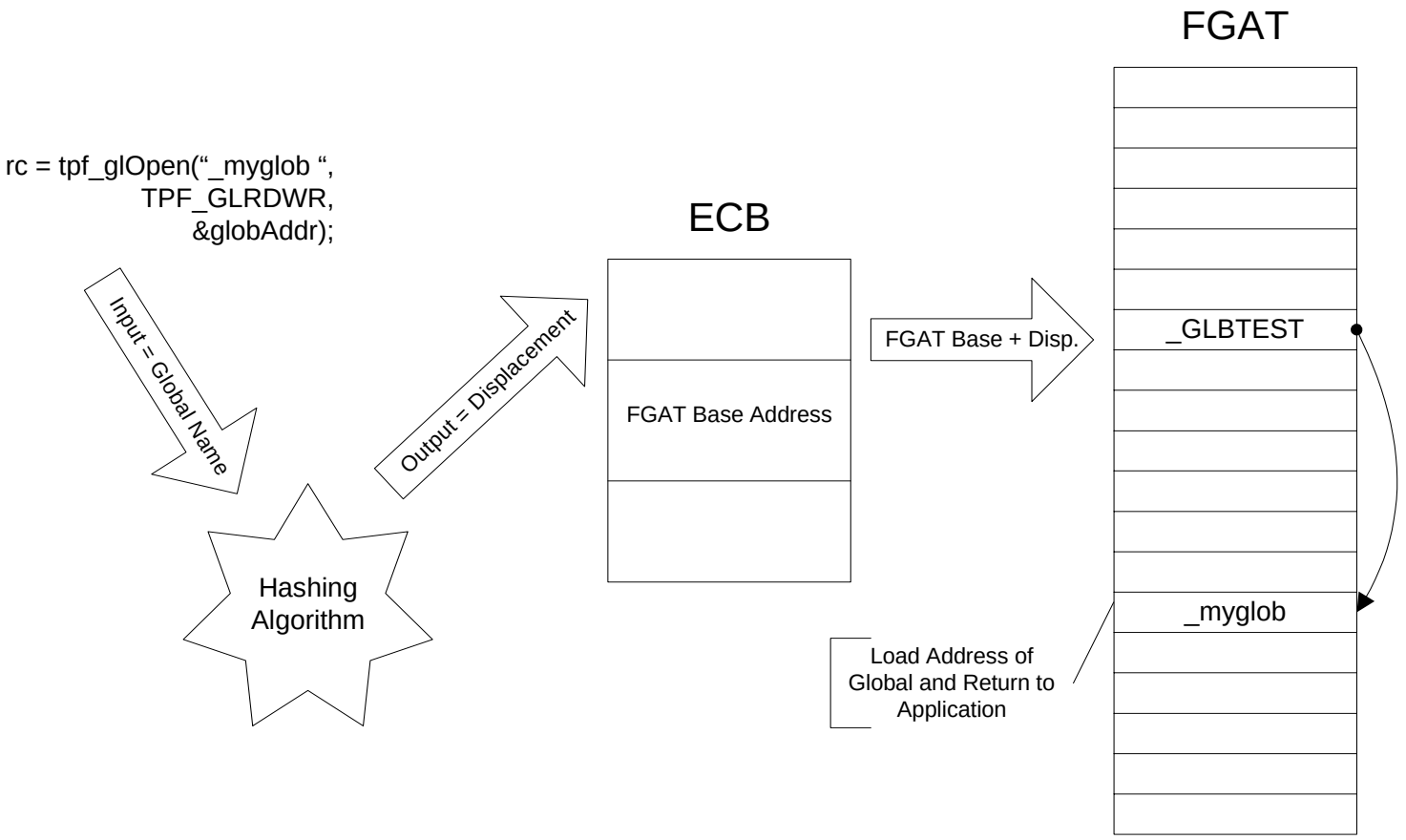
New z/TPF APIs for Format-2 Globals

- Abstract APIs determine behavior based on the defined attributes of the global
- C/C++ language APIs modeled after standard file system APIs
- Perform various actions on globals
  - Open a global:
    tpf_glOpen( ) and GLOBLC FUNC=OPEN
  - Request updates made to a global to be written to the database:
    tpf_glWrite( ) and GLOBLC FUNC=WRITE
  - Change options on how a global was opened:
    tpf_glCntl( ) and GLOBLC FUNC=CNTL
  - Obtain information about an open global:
    tpf_glStat( ) and GLOBLC FUNC=STAT
  - Close a global:
    tpf_glClose( ) and GLOBLC FUNC=CLOSE

Opening Format-2 Globals

- tpf_glOpen(char *globalname, enum t_glopt options, void **globaladdr);
  - Returns main storage address of requested global for the current SSU and IS
  - Specify options for opening a global
    - TPF_GLRD: open for read-only access
    - TPF_GLRDWR: open for read/write access (ie, lock global for exclusive use)
    - TPF_GLRDFST: open for read-only access using fast path
  - Creates a global descriptor (non fast path only)
    - Keeps track of all globals currently open for an ECB
    - Used as input for subsequent tpf_gl-type API calls
- Assembler equivalent: GLOBLC FUNC=OPEN
  - OPT=RDONLY
  - OPT=RDWRITE
  - OPT=RDFAST

Opening Format-2 Globals Example



Opening Format-2 Globals for Read-Only or Read/Write

- If the global is opened for read/write access, it will be locked for exclusive use by this ECB
  - Synchronizable: perform equivalent of SYNC LOCK operation to lock the global on DASD and refresh main storage copy
  - Non-synchronizable: perform CORHC on global's main storage address
- If the global is opened for read-only access, application can select one of two methods:
  - Fast Path
    - Best performance for accessing read-only data that will not be updated by the application
  - Non-Fast Path
    - Provides capability of efficiently changing a read-only access to read/write access
    - Provides diagnostic capabilities to keep track of all globals that have been accessed by this application

Updating Format-2 Global Data

- tpf_glWrite(int globaldesc, enum t_glopt options);
  tpf_glWrite(int globaldesc, enum t_glopt options, long offset, long length);
  - File out updates made to a global that was previously opened for read/write access
  - Only allowed for keypointable or synchronizable globals
  - Global remains open for read/write access for this ECB
  - Specify options for writing a global
    - TPF_GLALL: update the entire global
    - TPF_GLPART: update only the specified part of the global (using "offset" and "length" parameters)
- Assembler equivalent: GLOBLC FUNC=WRITE
  - Optionally specify: OFFSET=, LENGTH=

Changing the Open Options for Format-2 Globals

- tpf_glCntl(int globaldesc, enum t_glopt options);
  - Change open options for a previously opened global
    - Read/write access to read-only access
    - Read-only access to read/write access
  - Specify options for changing global access
    - TPF_GLRDWR: change access of global to read/write
    - TPF_GLUNLK: unlock a global previously opened for read/write access
    - TPF_GLUNLKWT: unlock a global previously opened for read/write access, and wait for any previous updates to be filed out
- Assembler equivalent: GLOBLC FUNC=CNTL
  - OPT=RDWRITE
  - OPT=UNLOCK
  - OPT=UNLOCKWT

Obtaining Information About Format-2 Globals

- tpf_glStat(int globaldesc, struct iglst *statinfo);
  - Obtain information about a previously opened global
  - Get attributes of global from format-2 global attribute table (FGAT)
    - Global's main storage address and size
    - SSU-unique or SSU-shared
    - IS-unique or IS-shared
    - Processor-unique or processor-shared
    - Keypointable or non-keypointable
    - Synchronizable or non-synchronizable
    - Protected or not protected
    - Opened for read-only or read/write access
- Assembler equivalent: GLOBLC FUNC=STAT

Closing Format-2 Globals

- tpf_glClose(int globaldesc, enum t_glopt options);
  tpf_glClose(int globaldesc, enum t_glopt options, long offset,
  long length);
  - Close a global and optionally write updates to the global to DASD
  - Specify options for closing a previously opened global
    - TPF_GLUPD: file out updates made to the global (ie, keypoint or synchronize)
    - TPF_GLUPDWT: file out updates made to the global and wait for completion (only for synchronizable)
    - TPF_GLNOU: close global without updating it
    - TPF_GLPART: only update the specified portion of the global (using "offset" and "length" parameters)
  - Remove format-2 global descriptor from ECB
- Assembler equivalent: GLOBLC FUNC=CLOSE
  - OPT=UPDATE
  - OPT=UPDATEWT
  - OPT=NOUPDATE

Considerations for Updating Format-2 Globals

- Updating a keypointable format-2 global
  - Action taken depends on state of subsystem
    - 1052 state: file the updated global to the database before returning to the application
    - Above 1052 state: turn on keypoint request indicator. Time-initiated keypoint process will scan through all keypointable format-2 globals; those that have requested a keypoint will be written to the database
  - Size limit on keypointable format-2 globals is determined by the user
- Updating a synchronizable format-2 global
  - Synchronize tightly-coupled and loosely-coupled based on IS-uniqueness and processor-uniqueness attributes of global
  - Size limit on synchronizable format-2 globals is determined by the user
  - Ability to update a portion of the global
  - Ability to have issuing ECB wait for all other processors in loosely coupled complex to acknowledge completion of synchronization

Modifying Format-2 Globals is Easier

- Both unprotected *and* protected format-2 globals can be updated in main storage directly by both assembler and C/C++ applications
- No longer need to call intermediate function to perform the update on behalf of the application
- tpf_glmod(enum t_glmod key);
  - Sets the correct storage protection access for the ECB to be able to alter protected format-2 globals
  - Use keyrc( ) to return the ECB to working storage protection when done modifying protected format-2 globals

Migrating Format-1 Global Data to Format-2 Globals

- Applications can be migrated one at a time to use new format-2 globals APIs to access format-1 globals data
- ZGLBL GLOBAL DEFINE with CONTROL-FORMAT1
- Allows user to define a format-2 global record that points at existing format-1 global data (ie, *format-1 global in migration state*)
  - All format-2 globals APIs can be used on format-1 globals in migration state.
  - New UGLM user exit to help the migration process
- ZGLBL GLOBAL MIGRATE
  - Migrate data from format-1 to format-2
  - Issue after all applications have been updated to use format-2 globals APIs
  - Global will become a system-controlled format-2 global, and will no longer reference the format-1 global data

Customizing Format-2 Globals Support

- When the requirements for a particular global record do not fit into the definition of a system-controlled global record, it can be defined as user-controlled
  - Large tables that have unique loading, keypointing, or synchronization requirements
  - Data that has unique storage requirements or may not even physically reside on the z/TPF system
- ZGLBL GLOBAL DEFINE with CONTROL-USER
- Easy customization for user-controlled format-2 globals with various user exits
- User-controlled or system-controlled is transparent to the application
  - Same format-2 globals APIs are used by the application

Operators Can Customize Format-2 Global Data in Dumps

- Using format-2 global names as dump keywords
  - 2 ways to create a dump keyword that references a format-2 global
    - ZIDOT CREATE command
    - IDATG macro
  - Use the ZIDOT INCLUDE command to specify that this global should be included for specific system errors. The copy of the global for the failing SSU and IS is dumped
  - "IF2GLOPN": pre-defined dump keyword that will dump all format-2 globals currently opened by the failing ECB

Programs Can Customize Format-2 Global Data in Dumps

- Specifying format-2 globals to dump at system error time
  - LISTC macro
    - Specify name of a format-2 global and maximum length to dump
    - Can be used for SERRC and SNAPC dumps
  - serrc_list and snapc_list structures updated for C/C++
    - SERRC_F2GLOBAL and SNAPC_F2GLOBAL indicators tell the z/TPF system that the "tag" field is a format-2 global name
  - "IF2GLOPN" can be used as the global name, as noted for dump keywords

Tuning Format-2 Globals Usage and Characteristics

- Data collection and data reduction
  - Data is collected and reported for the usage of synchronizable format-2 globals
  - Rate of locks, unlocks, synchronizations, and sychronizations with wait option

```
 FORMAT-2 GLOBALS SUMMARY                    BSS    SUBSYSTEM          14 JUN        09:18:32
     79 OBSERVATIONS

 GLOBAL NAME                 LOCKS/SEC     UNLOCKS/SEC       SYNCS/SEC  SYNC WAITS/SEC
ADDED/DELETED
 _globs                          0.124           0.051           0.057           0.016
 _myGlob                         0.115           0.056           0.043           0.016
 _newGlob                        0.009           0.008           0.000           0.001         Y
 _Delete                         0.000           0.000           0.000           0.000         Y
                            ----------      ----------      ----------      ----------
 TOTAL                           0.248           0.114           0.100           0.034
```

Applications Productivity Enhancements
Design and Coding Enhancements

Expanded ECB Private Area

- Private area above 16-MB and below 2-GB
- Size increased
  - Minimum size is 4 MB
  - Absolute maximum size is 16 MB
  - Usable maximum size is user settable from 4 MB to 16 MB
    - For all ECBs
    - For individual ECBs
      - EBMAXC, tpf_ebmaxc() can override usable maximum
- Easier management of traditional TPF I/O.

Expanded ECB Heap

- 31-bit ECB Heap - below 2-GB
  - Default heap area - malloc() uses this area
  - With z/TPF memory model , 31-bit ECB Heap size can be large
    - 100 MB ... 500 MB
- 64-bit ECB Heap - above 2-GB
  - New API - malloc64() uses this area
  - Same virtual size as 1-MB frame area
  - Intended as very large work area
    - Large ECB unique in core tables
    - Large sort area
    - ECB unique cache.

Expanded System Heap

- Dynamically allocate memory available to all ECBs
- Infrastructure which other function is built upon
  - Format-2 Globals
  - Memory File System
- 31-bit System Heap - below 2 GB
  - Virtual area
  - Uses from top of ECB virtual area to 2 GB
- 64-bit System Heap - above 2 GB
  - Virtual area - same size as 1 MB frames
  - Preallocated - backed with real memory
    - Intended for known large tables.

Assembler Program Enhancements

- Size no longer limited to 4 K
- BEGIN macro allows
  - Multiple base registers ... R1 - R8
  - No base register (baseless)
- Architecture enablements
  - Relative instructions
  - Long displacement instructions
- Macros updated to have baseless capabilities
- New standard linkage macros for E-Type programs
  - Based on CP linkage macros
  - CLINKC, SLINKC, RLINKC, ELINKC
  - Easier subroutine coding.

Assembler Program Enhancements - Register Handling

- R8 saved / restored across all SVCs
  - Does not need to be the program base register
  - Enables baseless coding
- R10, R11, R12, R13 available for use in E-type programs
  - Not guaranteed across macro calls
- Extended Register Save
  - Will save R10, R11, R12, R13 over general macro calls
  - Can be enabled for all general macros in a program
  - Can be enabled for general macros in a specific section of code.

C Environment Enhancements

- Every ECB is initialized with a C Environment
  - Stack is always allocated
- Dynamically allocate stack storage - ALLOC and alloca()
  - Efficient way to allocate temporary storage
  - Valid during scope of function
  - Accessible from both TPF Assembler and C/C++
- TPF Assembler programs can define space on stack frame upon entry
  - DSECT defined in program

```
         BEGIN NAME=ABCD,DSECT=WORKAREA,USEREG=Rz
REG_SAVE DS   8FD
TOD_SAVE DS   2FD
         APSTKC
```

C Environment Enhancements (continued)

- Ability to call C functions from TPF Assembler
  - New API - CALLC
  - Write function once and call it from C and Assembler
    - New IBM functions using this capability
  - Enhance existing assembler applications using C functions
    - Don't need to rewrite application in C
    - Gain benefits of using higher level language.

New File Systems

- Processor and Subsystem Unique
- Memory File System (MFS)
  - Based on system heap storage
  - Used for temporary files
  - Deleted on IPL or file system dismount
  - Operates at memory speeds
- Fixed and Pool File Systems (FFS and PFS)
  - Based on pool and fixed files
  - Used for persistent files
  - Dramatically different performance characteristics
    - More than 100x faster
    - Creating and deleting files
    - Appending to existing files

New Device Support

- Can use standard APIs to access data independent of the device type
- Device drivers under the TPF File System (TFS)
  - General Data Sets
  - General Tapes
  - Virtual Readers
  - Sockets
  - User written driver
- Standard Streams (stdin, stdout, stderr)
  - Special character files
  - Eliminated initialization overhead

New File Systems APIs

- All file system APIs are now supported including the following:
  - Full file and byte locks
  - File attributes
    - Record ID to assign to data records
    - File Service Level
    - User assigned

File System Check Utility

- Scandisk-like function with fix capability for all file systems
  - TFS, MFS, FFS, PFS
- Ability to correct a file system without requiring a re-initialization of the file system or a system IPL
- Performs checking of specified file system while the file system is in use
- Invoked via the zfile fsck comand
  - Use can be automated

Additional New Functionality - Storage-Protection Override

- Problem: When updating a protected global in TPF 4.1 the program can only write into the global area. It cannot write into the ECB or Stack
- Solution: Use hardware feature called storage-protection override
  - When on, write to storage in key 9 as well as the PSW key
  - Allows programs to:
    - Write into protected globals
    - And write into EVM work areas (ECB, ECB private area, ECB Heap, and Application Stack)
  - New parameters on GLMOD, KEYCC allow use of storage-protection override
  - New C API - tpf_stpoc()

Additional New Functionality

- Do not need to load a RIAT to use a new record ID
  - Simply input ==> ZRDTM MOD RECID-xxxx,....
  - Reduces bureaucracy for new development

- Improved addressibility to system Heap areas
  - Unique tokens can be assigned to system heap areas
    - Assigned when system heap is obtained
      - GSYSC or tpf_gsysc()
    - Address of in use system heap can be obtained
      - New API - FSYSC or tpf_fsysc()
    - System heap can be released using a unique token
      - RSYSC or tpf_rsysc()
  - Simplifies inter-program and inter-ECB communication via system heap storage.
  - Eliminates need to allocate space to save system heap address

Additional NEW APIs

- mallinfo() and EHEAPC FUNC=INFO
  - Gives total storage being used for 31-bit ECB heap
  - Gives maximum size 31-bit ECB heap buffer that can currently be allocated
- tpf_eheap_tag(), tpf_eheap_locate() and EHEAPC FUNC=TAG|LOCATE
  - Applications can assign names to ECB heap buffers
    - Alternative to passing ECB heap addresses between programs or routines.
    - Simplifies inter-program and inter-routine communication
- malloc64() and MALOC HEAP=64
  - Allocate storage from 64-bit ECB Heap
  - Similar APIs for calloc64() and realloc64()

Additional New APIs (continued)

- SWISC IMMEDIATE and swisc_immediate()
  - Move to another I-stream and start work where the SWISC is coded
- SYNCC SYNC,WAIT=YES
  - On return global update is in core on all processors
- TAMCC
  - Test Addressing mode
- tpf_tgetc(), tpf_tputc and TGETC, TPUTC
  - Ability for tape to read into or write out of discontiguous areas of memory
  - Ability to use any storage area such as ECB heap

Additional New APIs (continued)

- tpf_STCKE()
  - Obtain 128-bit extended clock
  - Provides C programs access to STCKE instruction
- ERRNOC
  - Provides assembler programs access to C variable errno
- DEBUGC
  - Debug facility for assembler programs
  - Can be used in the CP
  - Example ... take a dump if a routine is executed in 31-bit addressing mode

Availability Improvements

TCP/IP Tables - They're Movin' On Up

- Large tables reside above 2-GB bar, including:
  - Socket block table
  - IP message table (IPMT)
- Experience has shown there are many factors in determining what size IPMT is needed
  - Can make IPMT very large and not a concern
  - Larger IPMT helps enable OSA polling changes
- Can dynamically increase the size of these tables via ZNKEY command
  - Change takes effect immediately
  - No longer requires reloading CTK2 and an IPL

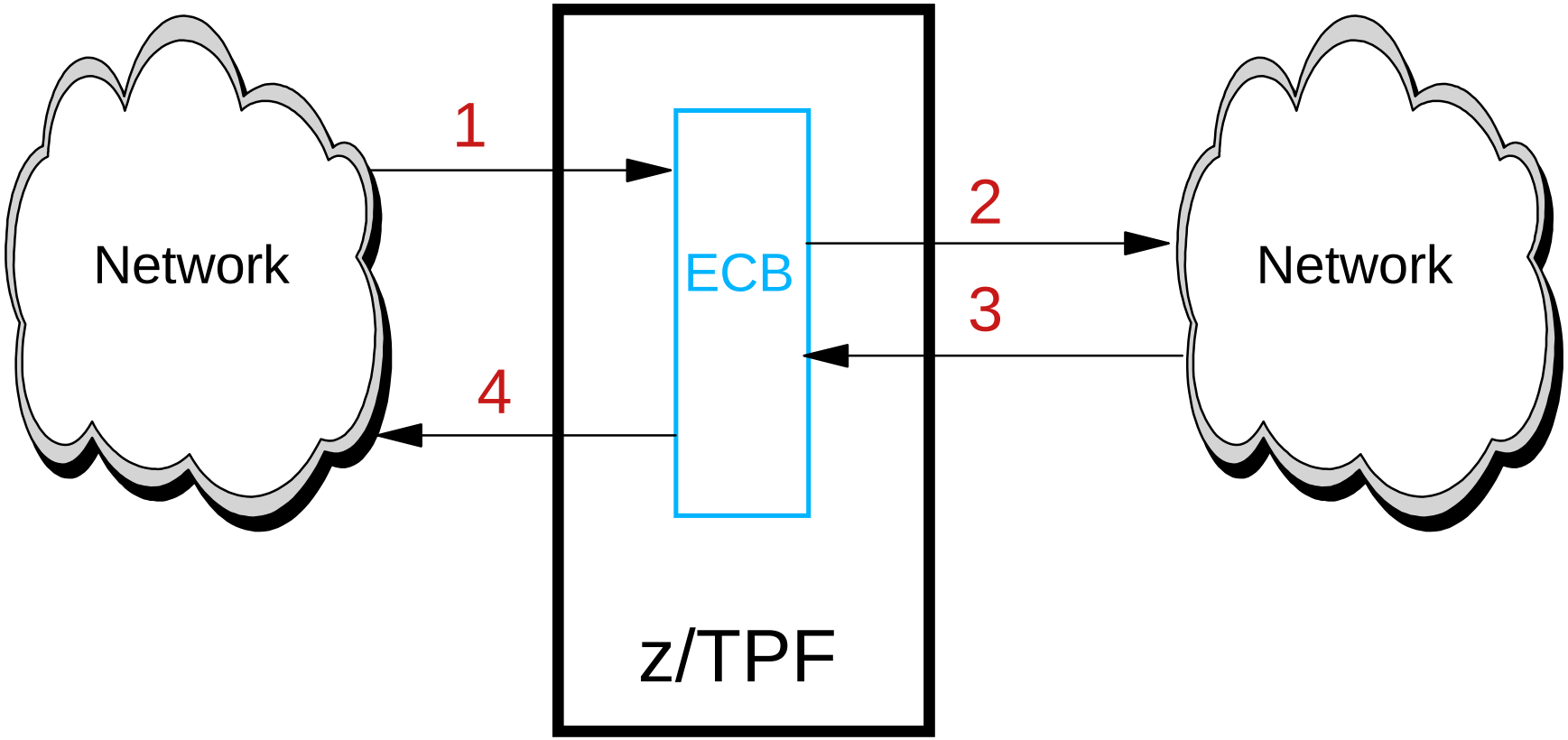
TCP/IP Input Message Priority

- Allows you to define the priority of TCP/IP input messages
- Determines how a message is processed by z/TPF
  - Independent of network priority
- Priority value is defined at the application level
- Application program can:
  - Override the priority value for a given socket
  - Change the priority value during the life of a socket
- "High Priority" input messages
  - Processed via the ready list rather than the input list
  - These messages are read in from the network and processed even when in input list shutdown
- Other priority values exist
  - Determines what messages are discarded first if the system runs low on TCP/IP block resources

OSA Polling Changes

- TPF polls (reads input messages from) OSA even when in an input list shutdown condition
  - High priority input messages continue to be processed
  - Other input messages are queued in memory (in the IP message table) and do not begin processing until system resources return to an acceptable level
- TPF polls OSA during dump processing
  - All messages are queued in memory and do not begin processing until after the dump is completed
- Reading input messages from the network during shutdown and dump processing reduces the likelihood of lost messages
  - Not polling would result in routers discarding messages that are destined for z/TPF
    - Lost messages would result in exception handling overhead following the shutdown or dump

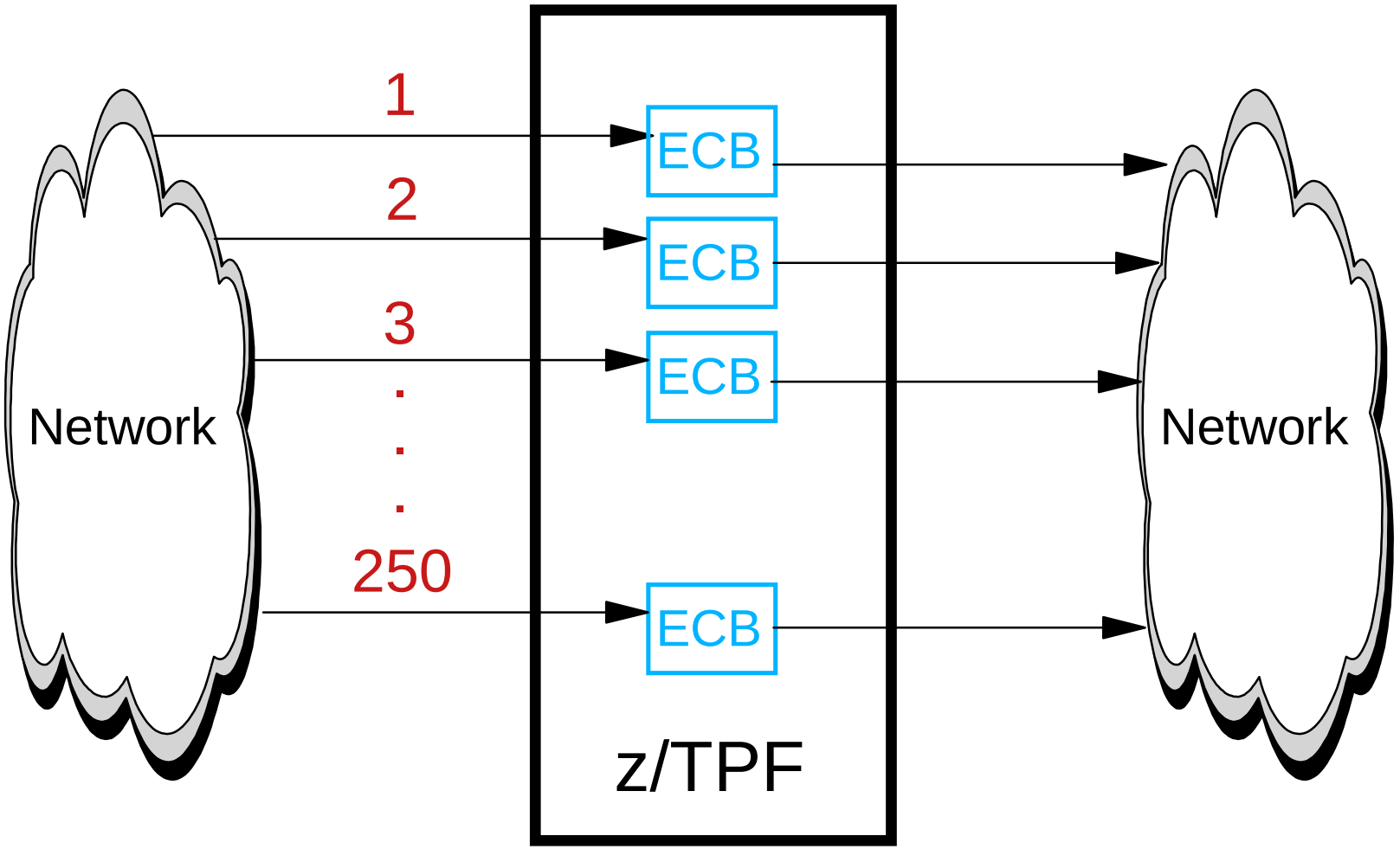
Distributed Transaction Processing



Distributed Transaction Details

1. z/TPF receives an input message from a remote client and an ECB is created to process the message.
   - How the message was received from the network (over TCP/IP, SNA, and so on) does not matter.
2. As part of the transaction, the z/TPF application sends a request to a remote server over TCP/IP. The application then issues a socket read() API causing the ECB to be suspended while waiting for the reply from the remote server.
3. The remote server sends the reply back over TCP/IP causing the application ECB to be dispatched again.
4. The application sends the output message to the remote client and the ECB exits.

Example of When to Use High Priority Input Messages



High Priority Input Message Example Details

1. Input message #1 is received, ECB #1 is created and sends a request to a remote server over TCP/IP. ECB #1 is then suspended waiting for the reply from the remote server.
   - The socket between z/TPF and the remote server is defined as high priority.
2. Step 1 is repeated 249 more times. There are now 250 active ECBs all waiting for replies from the remote server.
   - In this example, there are enough active ECBs to drive z/TPF into input list shutdown.
   - These ECBs will not go away until replies are received over the TCP/IP network from the remote server.
   - If the sockets between z/TPF and the remote server were not high priority, there would be a deadlock condition.
   - However, because these sockets are high priority, replies from the remote server are read in and processed, enabling the ECBs to exit. This in turn reduces the number of active ECBs are gets the system out of input list shutdown.

New and Improved Scheduler

- TPF scheduler
  - Also known as the CPU loop or task dispatcher
- Now uses a combination of shared lists along with the existing I-stream unique lists
  - Replaces the TPF 4.1 routing weight based scheduler
- Better suited for z/TPF processors running a larger number of I-streams
- Enables more efficient use of compute resources within a processor, especially at very high utilizations
  - May result in better response time

How to Take Advantage of the New Scheduler

- Program affinity attribute
  - Determines whether an ECB can dynamically switch I-streams while processing in a given program/application
    - For example, after a DASD I/O operation is completed
    - Default is not to dynamically switch
      - Behavior is consistent with TPF 4.1 to make migration easier
- Certain middleware and system services will be defined to take advantage of dynamic balancing, including:
  - MQSeries, mail server
  - TCP/IP native stack, file systems
- Application programs that are most frequently used are prime candidates to exploit the dynamic balancing capability
  - Assuming the application is tightly-coupled capable

Dynamic Program Allocation

- New ADD option on ZAPAT command
  - Allocates a new program in the PAT
  - Replaces PATU option on ZOLDR LOAD
- Different ways to define the attributes (like KEY0, MONTC, RESTRICT) of the new program:
  - Specify the attributes on the ZAPAT ADD
  - Use the same attributes as an existing program
    - Use the LIKE option on ZAPAT ADD
  - Use the attributes that are assigned to unallocated programs
    - Do not specify any attributes or the LIKE option on the ZAPAT ADD command

ZAPAT ADD Example

```
ZAPAT ADD CLDW LIKE CLDB
APAT0200I 16.54.42 PROGRAM CLDW ADDED
                   BEGIN DISPLAY OF FILE COPY FOR IMAGE CUR51B

PROGRAM          CLDW
VERSION

LINKAGE TYPE     ---
BASE PAT SLOT    00000000
FILE ADDRESS     00000000D43014D5
CORE ADDRESS     0000000000000000
AUTHORIZATION    KEY0 MONTC RESTRICT
FETCH            DEFAULT
QUALIFIERS       NONE
TIMEOUT          50
DUMP GROUP       NONE
TRACE GROUP      IBM_DEFT
AFFINITY         PROGRAM
```

Enable More Utilities to Run in 1052 State

- More system services can be enabled in 1052 state:
  - File systems
  - Pools (GFS)
  - TCP/IP native stack, including Internet Daemon (INETD)
- Allows certain utilities to run in 1052 state, including:
  - FTP
    - Upload configuration files needed by applications before cycling to NORM and starting those applications
  - TPF Debugger
    - Can debug code in 1052 state now

1052 State Enabled System Services Details

- GFS does not automatically start until cycle up above 1052 state (same behavior as TPF 4.1)
  - Can manually start pools in 1052 state
    - New ZPOOL 1052 UP command
- An OSA-Express connection can be defined as automatically starting in 1052 state (default is still CRAS state)
  - Do not need additional OSA-Express adapters
    - Can define multiple logical connections to the same physical OSA-Express adapter
      - For example, one connection that starts in 1052 state and another connection that does not start until CRAS state
- Sockets can be started in 1052 state using new *SetTCP1052()* API
  - Default is still that sockets cannot be started until CRAS state
- INETD can be set up to automatically start in 1052 state and start certain applications

Reduce Time Spent Dumping

- When a dump is in progress, all I-streams are stopped
  - No work can be done; the system is unavailable
- In TPF 4.1 data is written to tape during dump processing
- The amount of time that the system is unavailable is a function of:
  - Tape speed
  - Amount of storage being dumped.

- z/TPF addresses both tape speed and amount of storage dumped.

Reduced Time Spent Dumping - Tape Speed

- In z/TPF data is written to a dump buffer area during the dump
  - Memory to memory move during dump processing
  - Write to tape is started after the dump has completed
  - Size of the dump buffer area is set by users
    - Make it as big as you need - 2 GB, 5 GB, 10 GB.

- If dump buffer area is full, dump processing will write to tape
  - RTL tape has been changed to increase speed
    - Must be blocked
    - Dump blocks are 128-KB
      - Increased from 32,760 bytes
      - 128-KB is optimal size of writing to tape devices.

Reduce Time Spent Dumping - Reduce Amount of Data

- Ability to dump percentage of frames and SWBs
  - When dumping frames or SWBs, rather than dumping all blocks, dump a sample of each block
  - Specify from 0% to 100%
    - ZASER MAXBLKS.
- Ability to limit number of bytes dumped for an ECB heap buffer
  - If an ECB has a 100-MB ECB heap buffer, an OPR dump could take a long time
    - Is all 100 MB needed?
  - Specify number of bytes to dump for each ECB heap buffer
    - ZASER MAXEHEAP.

Reduce Dump Impacts

- Prevent streaming dumps causing an outage
  - Ability to limit the number of CTL dumps
    - If more than a specified number of CTL dumps are taken in 1 minute, NODUMP remaining CTL dumps in that minute
      - ZASER MAXCTL.
  - Ability to limit the number of dump messages sent to prime cras
    - If more than a specified number of dump messages are sent to prime cras in 1 minute, do not send any more messages in that minute
      - Dumps will still happen
      - Dump message will be written to tape
      - ZASER MAXCPSE.

Identify Run Away ECBs

- ECB Resource Manager
  - Based on ECB Resource Policeman
- Identify ECBs which:
  - Do too many FINDs, FILEs, SERRCs, or create ECB-type macros
  - Get too many SWBs, Common Blocks, pool file addresses, or system heap
- Actions taken can be:
  - Send a message
  - Take a SERRC with return
  - Take a SERRC and exit the ECB.

ECB Resource Manager Example

```
ZECBM DISPLAY
CSMP0097I 18.34.59 CPU-B SS-BSS  SSU-HPN  IS-01
ECBM0010I 18.34.59 ECB RESOURCE LIMIT TABLE DISPLAY FOR CORE COPY

         1ST LIMIT  2ND LIMIT
RESOURCE WARNING    PERCENTAGE
FIND         5000           0
FILE         5000           0
SERR           10           0
SWBK          200           0 _
GRFS       300000         300
CMBK           50           0
SYSH            0           0
CRET          100           0
FILT= 1000 FILR= 1000
ecbRM MONITORING IS: OFF
1ST LIMIT ACTION IS: MESSAGE
2ND LIMIT ACTION IS: DUMPEXIT
END OF DISPLAY+
```

Additional Enhancements

- Ability to include registers in macro trace without an IPL
  - Provides ability to get more diagnostics information quickly and easily.
- New ECB Heap Control Table (EHCT).
  - Each ECB has its own unique EHCT used to store information required in the management of the ECB heap.
  - Separates system control information from user data.
  - Applications overwriting ECB heap buffers will not overwrite system control information.

New Tuning Capabilities

BAL Packaging

- Assembler programs are linked
- Ability to package many assembler programs into one load module
  - For example, the following programs are link edited into CVAA
    - CVAA, CVAD, CVAH, CVAI, CVAO, CVAU
- Reduces linkage costs
- No need to update or reassemble programs
- Loading is done with ZOLDR
  - No need to IPL
- Further linkage cost reductions can be done by modifying ENTRC to programs that are in the same load module
  - ENTRC aaaa,TYPE=INT
  - Most efficient linkage

Tuning Performance of the ECB Heap

- Most ECB heap requests are for small amounts of storage
  - New algorithm enables users to optimize allocation performance for these smaller buffer requests.
  - 4 fixed-size available lists are used to fulfill these smaller buffer requests.
    - Virtually eliminates the search time for small buffer requests.
    - Reduces amount of fragmented, or "unusable", ECB heap storage
- Larger requests are fulfilled from a 5th variable-sized available list.
- Users can customize
  - Size of each fixed-size buffer
  - Number of fixed-sized buffers allocated at ECB initialization.

Sample ECB Heap Statistics Report from Data Collection

- ECB Heap Statistics data collection report provides information for setting counts of buffers on the fixed-size available lists.

```
ECB HEAP STATISTICS

  MAXIMUM FRAMES FOR 31-BIT ECB HEAP                12 1MB FRAMES
  MAXIMUM FRAMES FOR 64-BIT ECB HEAP                10 1MB FRAMES
  SIZE OF 31-BIT PREALLOCATED ECB HEAP           65536 BYTES

                                      AVL1         AVL2         AVL3         AVL4

  BUFFER SIZE (BYTES)                   64          256         1024         4096
  BUFFER COUNT (INITIAL)                64           16           16            2
  ECBS WITH BUFFER MISSES                0           32            0          156

  FOR THOSE ECBS WITH BUFFER MISSES:
     MEAN BUFFER MISSES              0.00        80.53         0.00         1.15
     STANDARD DEVIATION              0.00        56.95         0.00         0.53
```

Sample ECB Heap Request Size Report from Data Collection

- ECB Heap Request Size data collection report provides information for setting sizes of buffers on the fixed-size available lists.

```
TPF ECB HEAP REQUEST SIZE REPORT

CURRENT BUFFER SIZES IN BYTES:       AVL1 -     64  AVL2 -    256  AVL3 -   1024  AVL4 -   4096

   CLASS UPPER    FREQUENCY    PERCENT     FREQUENCY DIAGRAM  (SCALE =     1/1)
      LIMIT       OBSERVED     OF TOTAL

       128            33       17.37%    |*********************************
       256             0        0.00%    |
       384            38       20.00%    |************************************
       512             0        0.00%    |
       640             1        0.53%    |*
       768            30       15.79%    |*****************************
       896             0        0.00%    |
      1024             0        0.00%    |
      1152             0        0.00%    |
      1280             0        0.00%    |
      1408             0        0.00%    |
      1536             0        0.00%    |
      1664             0        0.00%    |
```

ECB Preallocated Storage

- Storage permanently assigned to an ECB
  - Private Area
  - ECB Heap
  - Application Stack
- Reduces overhead of updating EVM when added storage is needed
  - Also, reduces cleanup at exit
- Size can be set by the user
  - Intent is to set size of preallocated areas so that storage requests will be satisfied with preallocated for the majority of your ECBs
- Efficient use of storage based on application usage patterns
- Optimizes performance based on your unique applications.

Additional Enhancements

IP Scan

- Performs many TCP/IP stack functions on a time-initiated basis, including:
  - Sending TCP delayed acknowledgments
  - Retransmitting messages lost by the network
  - Sends output messages from partially full output buffers
- Now can run on any I-stream
  - Used to run on the main I-stream only
  - Enables a more balanced load on z/TPF
- Can be invoked more frequently
  - May improve response time

Reduce IPL Time and Reduce VM Impact

- z/TPF does not test block (TB) every 4-KB in IPLB
  - TB 4K pages when they are needed
    - VFA buffers (if VFA is rebuilt)
    - Resources: SWBs, Common Blocks, 4-KB frames, 1-MB frames
  - Use of Available lists and Allocated lists for resources
    - All resources are on the allocated list initially
    - No resources on available list initially
    - Resource moved to available list on get when list is empty
      - TB done when resource moved to available list
    - All releases go to available list
    - All gets look at available list first
  - Both combined minimizes VM working set size.

Reduce IPL Time

- Enhancement to Format-1 Globals restart
  - Global restart runs in parallel with
    - TCP/IP restart
    - SNA restart
    - TPFDF restart
    - CCP restart

Enhanced Memory Management

- Ability to define up to 8 memory configurations in CTKA
  - Memory configuration consists of:
    - Number of IOBs, CMBs, SWBs, ECBs, 4-KB and 1-MB frames
    - Size of dump buffer area, preallocated ECB areas, and more
    - Minimize size needed for 31-bit system heap and VFA
  - On IPL CCCTIN selects the configuration to use based on:
    - Defined preference
    - Best fit based on available memory
- Allows predefined memory configurations for
  - New faster processors with more memory
  - Various sizes of test systems
- Easier management of CTKA.

Improved Timeout Processing

- Each load module has its own timeout value
  - Defined offline in control file, online in PAT
  - Ability to use the calling program's timeout value
    - C libraries such as CTAL uses this option
  - Ability to set a default timeout value for user programs
- Internal CP table which allows unlimited processing has been eliminated
- Timeout processing enabled in all states
  - Including restart and 1052.

Constraint Relief

- Support up to 255 SSUs
- Support SDA addresses to x'FFFF'
  - Including tape
- Support up to 40,000 DASD devices

Improved Coordination of Online Changes to Offline Source

- Ability to put online tables into offline source format
  - ZDECK command
    - RIAT
    - PAT
- Enhancements to Positive Feedback
  - Improved log management
  - Ability to select any command to be added to positive feedback.

New Definable Options Requested by Customers

- Prime / Dupe module pairing
- DASD least queuing option
  - When DASD queue length is equal, always go to dupe rather than flip flop
- Set in SIP.

New User Exits Requested by Customers

- Online DBR
- Midnight processing
- Copy member of user fields in ECB page 1
- Dump error message processing (CPSA)
- Dump data for OPR dumps
- Duplicate dump processing
- ZSTAT command

Various Command Changes Requested by Customers

- New command to display CINFC information
  - ZDCNF / ZACNF
- Display state of all subsystems - ZDSYS ALL
- Display TOD clock sync information - ZPSMS DISPLAY TOD
- Ability to get macro name from SVC number - ZDSVC

Various Enhancements Requested by Customers

- Eliminate assembly and compile warnings
- Improved lock release routine in dump processing
- Prevent looping catastrophic dumps
- Increase shutdown levels to 4 bytes
- Counts of DASD finds from DASD and VFA in the ECB
- Repeat tape mount request in restart
- Ability to request keypoint of in core user keypoint
- C function for FLVFC - tpf_flvfc()
- Improve CXFRC Parent processing
- RLCHA program RLCH to use 4-KB block

Enhancements to LODIC Support

- Additional 4 user classes supported
- Provide ability to specify a maximum amount of time that an ECB will lose control on initial LODIC call due to low resources
- When marked ECBs create child ECBs, allow the child ECBs to use a different class
- Supports shutdown for 1-MB frames
- Enhanced lose of control support
  - Save and restore new floating point registers.

Larger Tape Blocking Support

- Support tape blocking up to 128-KB
- Any tape can use this
  - Command ZTLBL updates tape label mask
  - Command ZTMNT can override tape label mask
- 128-KB is most efficient size to write to tape devices
- RTL tape handling
  - Always write dump blocks at 128-KB
  - Blocking for other writes (TOURC) is user setable
- RTA tape handling
  - Always blocks at 32,760 bytes - dumps and user data
  - Most traces go to RTA.

More VFA

- VFA gets remaining storage after all tables are allocated
  - Could be a large amount of storage - gigs and gigs
- New function added to support large number of delayed file records
  - Do not file delayed filed short term record if it has not been accessed in a specified period of time
    - Likely file address is not used anymore, a release file was not done for it
    - User setable by record ID using command
      - ZRTDM MOD RECID-xxxx,PDELAY-yyy

Larger Keypoints

- Support 48-KB keypoints
- IBM keypoints moved to 48-KB are: CTKA, CTK2
- User keypoints moved to 48-KB are: CTKD, CTK3
- New APIs to access keypoints
  - GETKC / tpf_getkc() and UPDKC / tpf_updkc()

Migration to z/TPF: Concepts and Tools

Single Source Concept

- There are some changes which have to be made to application programs in order to migrate from TPF 4.1 to z/TPF
- Single Source support allows these changes to be made to applications while they are running in TPF 4.1
  - Don't need two copies of application source
  - Provides capability to reassemble / recompile TPF 4.1 applications for z/TPF.

Architectural Concepts

- For assembler programs, 31-bit addressing mode is supported
- Storage areas which are shared among C and assembler programs are below 2-GB
  - ECB private area
  - Default ECB heap
  - Application stack
  - Format-1 Globals
  - Format-2 Globals are supported below 2-GB
- 31-bit pointer is available to C programs
  - Provides access to addresses stored in 4-byte fields

Closing

Comments

- Performance
  - Impact is estimated to be around 10% though individual results may vary.
- Migration to z/TPF from TPF 4.1
  - PUT 15 or higher installed on TPF 4.1
    - 32LC conversion complete (ZPMIG & ZMIGR done)
    - FARF6 conversion complete (ZMODE 6 done)
  - Coexistence supported for loosely coupled complexes

Software not supported in z/TPF

- Target(TPF) C
- 24-bit addressing mode
- 24-bit Global support
- Fallback keypoint extents
- File resident programs
- PTV
- RTT
- VEQR mode
- Old comms: SLC and bisync
- CLAW offload support for TCP/IP
- SNA PU 2.1 Logon Manager

Hardware not supported in z/TPF

- 3350, 3375, 3380, 3390, 9345, RAMAC DASD
- 3880, 3990-2, 3990-3, 3990-6 DASD CU
- 3480, 3495 tape
- 3705, 3725 CCU
- 3172, 3174 Terminal CU
- 3088 CTC
- 3505 card reader
- LLF, ELLF
- TOD RPQ
- STR subset
- 1403 printer

Closing

- Questions???

Legal

IBM, z/Architecture, z/VM, zSeries, z/OS, and Websphere are trademarks of International Business Machines Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.