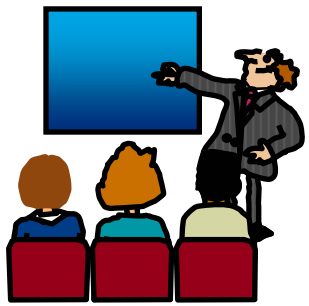
A sysprog view of z/OS 64-bit Virtual Application Support

Thomas Petrolino
IBM Poughkeepsie
tapetro@us.ibm.com



Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- CICS®
- Hiperspace
- IMS
- Language Environment®
- MVS
- OS/390®
- z/Architecture
- z/OS®
- z/Series®

® Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.

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

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

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

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Agenda

- Overview
- 64-bit Application Stack
- Application Environment
  - Migration/coexistence
  - Installation
- Run time options
- 64 bit DLL Support
- Programming environment
  - Compiler support
- Pre-initialized Environment
- 64 bit exploitation considerations
- Appendix

Overview

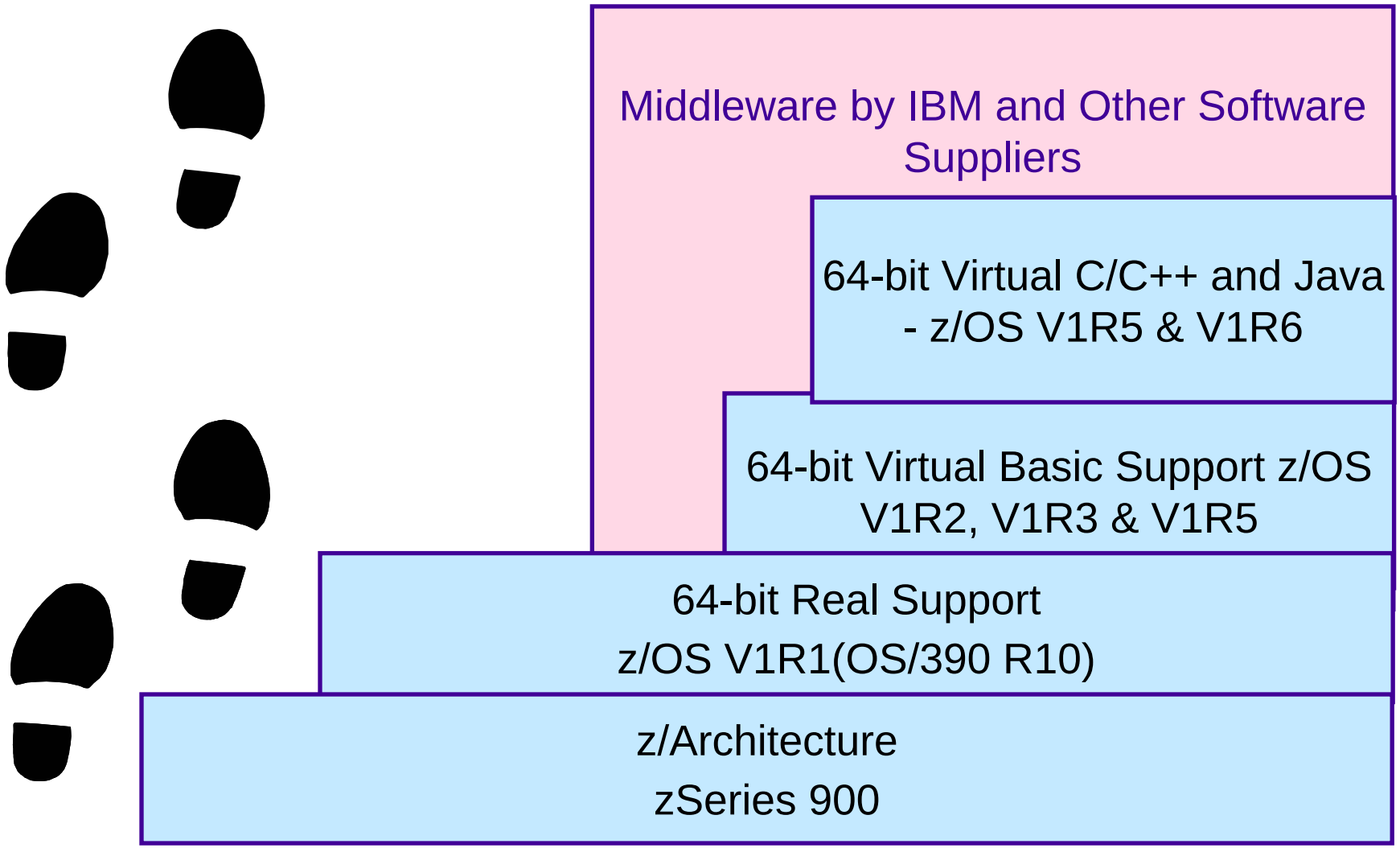
- High level languages need to address storage above the 2G Bar
  - Both application programs and middleware
- Solution: Language Environment creates a separate 64bit run-time environment
  - The support is for C/C++ (and Assembler) only.

Overview

- Using 64bit Virtual support, the customer can
  - Access data above the 2G Bar
  - Allocate large data areas (even those in excess of 2G)
  - More easily port applications from other 64bit platforms

- Advantage:
  - Consolidation of data within a single address space (especially for databases and large buffer pools)
  - Manipulating large amounts data within storage

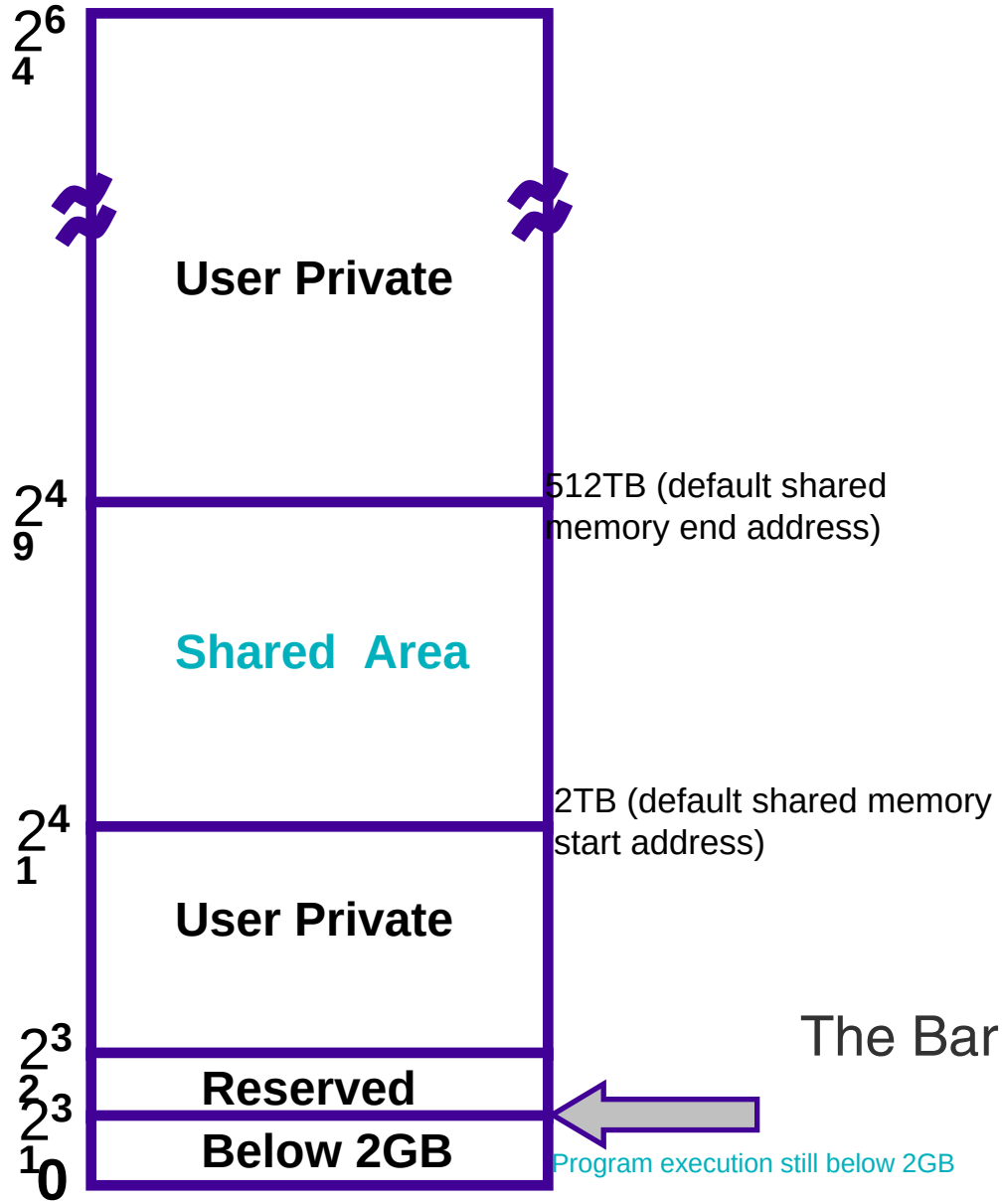
z/OS 64-bit virtual support



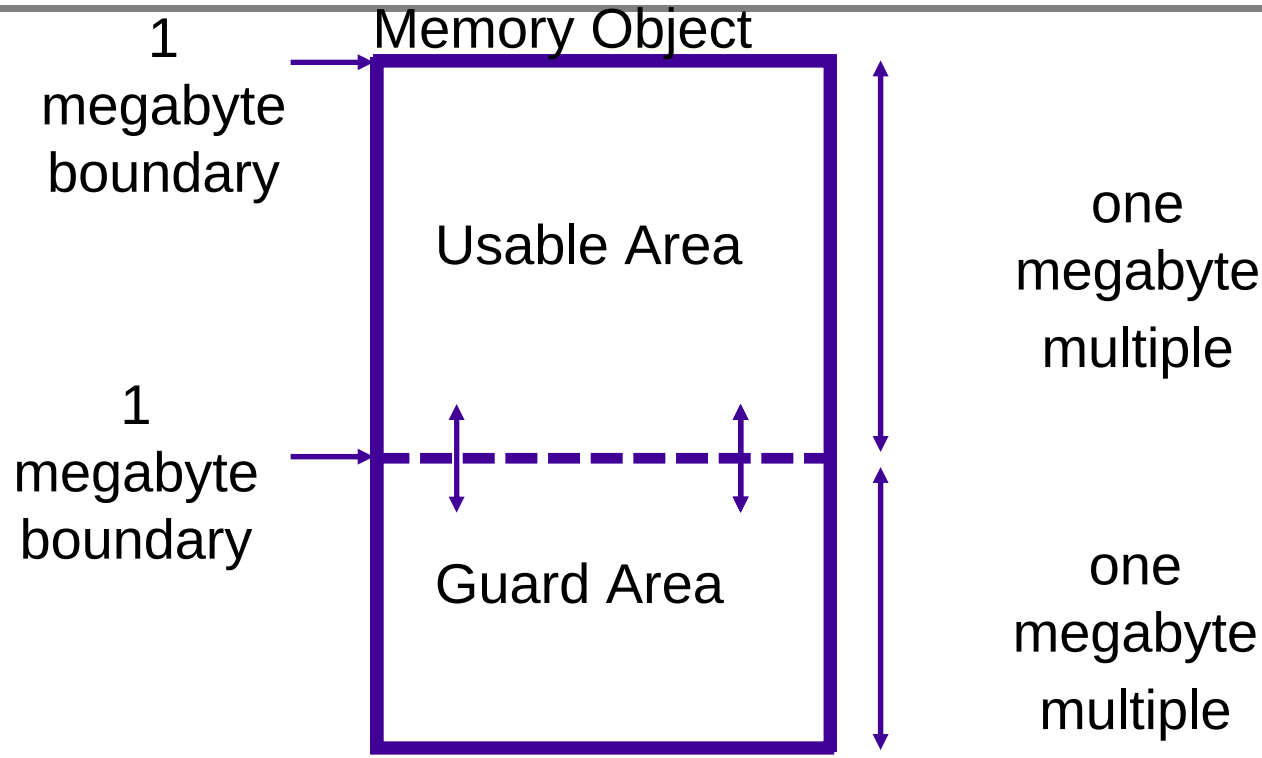
What are the 64-bit virtual releases?

- z/OS V1.2 - 9/2001 GA
  - basic 64-bit virtual
- z/OS V1.3 - 3/2002 GA
  - can bind/load/execute AMODE64 assembler programs
- z/OS V1.5 - 3/2004 GA
  - basic 64-bit shared virtual storage across address spaces
  - can experiment with compilation only for 64-bit C/C++ programs
- z/OS V1.6 - 9/2004 GA
  - 64-bit C/C++/Java programming environment
  - 64-bit C/C++ compiler and debugger

64-bit Address Space Memory Map



64-bit Memory Object



z/OS manages virtual storage above 2GBs differently. Memory requests cause the creation of "memory objects" which impose discipline on the virtual storage. It's not just bytes.

64 bit Application Stack

- C/C++ Compiler
  - 1.6 compiler produces amode 64 program objects
  - 1.5 compiler only for test compiles to fix compile-time problems. Does not produce object code for 64-bit.
- Binder
  - require the z/OS 1.6 binder for 64 bit virtual programs
- Unix System Services
  - 64-bit API for standard UNIX functions in C/C++
  - New interfaces for 64-bit assembler code (BPX4*)

64 bit Application Stack (cont.)

- Runtime
  - V1R6 Language environment
  - C/C++/Assembler support available
- Debugger
  - DBX is the only debugger available
- z/OS Base (BCP)
  - various services provide 64-bit APIs where needed
  - most traditional MVS services do not support 64-bit mode

Overview

- With the addition of this 64-bit form of LE, the Language Environment component of z/OS will consist of three forms:
  - Base form (31-Bit, EBCDIC, Standard LE linkage, Hex, IEEE)
  - XPLink form (31-Bit, EBCDIC, Enhanced ASCII, XPLink linkage, Hex, IEEE)
  - 64-Bit form (64-Bit, EBCDIC, Enhanced ASCII, XPLink linkage, Hex (z/OS V1.7), IEEE)

Application Environment

Application Environment

- Hardware
  - z/Architecture (z/900 and later)
- Software
  - z/OS 1.6
  - z/OS 1.6 C/C++ compiler
  - z/OS 1.6 Binder

Installation

- Prerequisites for installation:
  - New members CELQ* shipped in existing Language Environment datasets:
    - SCEERUN2
    - SCEEBND2
    - SCEESAMP
  - A new CELQDOPT usermod is available to customize default run-time options for AMODE 64 applications
  - To enable Language Environment AMODE 64 support you must set MEMLIMIT
    - U4093 RC=548 if MEMLIMIT is not set
    - see Appendix for details to set MEMLIMIT
- Publications References, e.g.:
  - SA22-7564 z/OS V1R6.0 Language Environment Customization

Migration and Coexistence

- 24-bit and 31-bit support is not affected
- Existing applications will continue to run and be supported
- LE will continue to enhance and ship existing 24/31 bit libraries
- A new library was added to provide the 64-bit virtual support
- There is no support for bimodal applications
  - 64-bit and 31-bit programs must run in separate processes.
  - The appropriate Run Time Library, 31-bit or 64-bit, is loaded at execution.
  - All High Level Language code within a process must run in a single mode.
  - Assembler language routines can switch modes if desired, but must not invoke the Run Time Library in the "other mode".
  - Communications between 31-bit and 64-bit processes is through the standard Unix interprocess functions.

Which Run-time will be used?

- The form of LE selected is based on the name of the LE bootstrap routine that is referenced in the first csect emitted by the compiler.
  - In 64-Bit LE, the C compiler emits a csect CELQSTRT instead of CEESTART.
  - The names of the bootstrap routines are as follows:
    - CEEROOTA - Base form of LE
    - CEEROOTD - XPLink form of LE
    - CELQBST - 64-Bit form of LE
- The application load module
  - fewer library modules than an LE-enabled application module today (details in appendix)
  - All library-provided 64-bit CSECTs renamed to CELQ*

Which Run-time will be used?

- AMODE 31 or AMODE 24 main program
  - The "regular" Language Environment run-time will be loaded and initialized
  - These programs may have CEESTART as the entry point with CEEROOTA (non-XPLINK) and CEEROOTD(XPLINK) as the boot strap routines
- AMODE 64 main program
  - The AMODE 64 specific Language Environment run-time will be loaded and initialized (CELQLIB)
  - AMODE 64 main programs will have CELQSTRT as the entry point and CELQBST as the boot strap routine

Amode64 Environment

- 64 bit form of Language Environment supports
  - data **above** the "bar' (>= 2G address)
    - user stack above the 'bar'
    - user heap above the 'bar'
      - capability to obtain heap below the 'bar'
    - data objects are obtained in Megabyte increments
  - Writable Static Area (WSA) **above** the 'bar'
  - code execution **below** the 'bar' (< 2G address)
    - no execution support above the 'bar'
- C Runtime environment is always initialized
  - XPLink is the only linkage model for 64-bit.
- For the initial release (R6)
  - Supported languages are C, C++ and LE conforming assembler

Run Time Options

LE Runtime Options

- The set of supported Runtime options is reduced and some defaults are changed when LP64 is specified.
- New run-time options to support both stack and heap storage above-the-bar
  - heap64, stack64, threadstack64, heappools64, ioheap64, libheap64
- Options not required for 64-bit support have been removed (i.e. ALL31, RTLS)
  - if 31-bit only option specified, message issued
- See appendix for details

Run-Time Options for AMODE64

- Where options can be specified for LP64

- CELQDOPT - Installation wide amode 64 run-time options (similar to CEEDOPT)
  - link-edited with CELQLIB
- CELQUOPT- Application specific amode 64 run-time options (similar to CEEUOPT)
  - link-edited with user application
- System defaults - CEEPRMxx parmlib member, SETCEE command (V1.7)
- #pragma runopts - C program source, C/C++ compiler builds CELQUOPT included with program
- DD:CEEOPTS - Execution time run-time options from data set (V1.7)
- PARM card - PARM="run-time options/program parms"
- _CEE_RUNOPTS Environment variable for spawn/exec family of functions (USS)

Run Time Options

- Propagation via spawn() and exec()
  - When going from 31-Bit to 64-Bit and vice versa through the spawn or exec functions, Language Environment rebuilds the _CEE_RUNOPTS environment variable as a means to propagate run-time options to the new program.
  - In the situations where 31-Bit specific options or 64-Bit specific options would be passed across to the other mode, options processing will *ignore* these options.
  - No messages will be issued. For example, when the STACK option is sent across from 31-Bit to 64-Bit, it will be ignored. This is because the 64-Bit program uses the STACK64 option.
  - No attempt to convert the 31-Bit option to the new 64-Bit option will be performed.

Programming Environment

- C data model and data neutrality
- Amode64 APIs
- External control structures
- Exception handling

C/C++ Data Model

- ILP32 used for AMODE24/31 programs
  - integer 32 bits (4 bytes)
  - long 32 bits (4 bytes)
  - pointer 32 bits (4 bytes)
- LP64 used for AMODE64 programs
  - integer 32 bits (4 bytes)
  - long 64 bits (8 bytes)
  - pointer 64 bits (8 bytes)

☞NOTE: with ILP32 it was "ok" to assign pointers to integers and long to integers. With LP64 this is no longer the case.

64 bit C/C++ Compiler Support

- The default addressing mode is 31-bit
- The addressing mode is controlled by the **LP64** compiler option
- There is a **warn64** option allows to detect portability errors from 32-bit to 64-bit
  - e.g truncation, wrong results, implicit conversions of data types due to casting,etc.
- The only object file format supported is GOFF
- Must use PDSE or z/OS UNIX file system
- The only linkage convention is XPLINK
- Defines _LP64 feature test macro

C/C++ Support - compiler

- OS-linkage
  - OS_NOSTACK
  - #pragma linkage(func_name,OS_NOSTACK)
  - provides 64 bit OS linkage
  - 144 byte savearea pointed by register 13 (on application's 64 bit stack)
  - Register 1 pointing to OS style parameter list
- great for calling assembler stubs from a C/C++ program to invoke system services
  - e.g C system APIs

C Data Neutrality

- The objective is to have common source code.
- The code can then be compiled 31-bit or 64-bit based on the **LP64** compiler option.
- The ***_LP64*** feature test macro can be used for mode-specific code.
  - When you have hard coded offsets or want different logic flow
- Function prototype are highly recommended since the default return type is still ***int***.
- The compiler reserves two new pointer qualifiers, ***__ptr32*** and ***__ptr64***. ***__ptr32*** can be used to define a 31-bit pointer in a 64-bit program.
- Structures and control blocks have different lengths and field offsets when compiled for 31 bit vs. 64 bit

Data neutrality issues

- What to look out for when making the code data neutral....
  - Loss or truncation of data due to size/type mismatch
  - Assigning pointer to integer
  - casting of long, int, pointer
  - constant definition - max range
  - implicit vs. explicit padding
  - doubleword address alignment
  - array sizes, hardcoded offsets
  - size_t and return values or input parms
  - Lots of documentation out there
  - The **WARN64** compiler option can be used to have the compiler identify potential problem areas.

AMODE64 Conforming Assembler

- New AMODE 64 macros are available
  - CELQPRLG Entry (prolog) macro
    - EXPORT=YES to export function
  - CELQEPLG Exit (epilog) macro
  - CELQCALL Call macro
    - Only supports “call-by-reference”
    - Floating point parameters not allowed
      - You may code that based on XPLINK design
  - CEEPDDA define imported/exported data
  - CEEPLDA locate imported/exported data

Programming Environment

- Stack and User heap are above the bar
  - increments are on Meg boundaries
  - Maximum stack size specified
  - Overall above the bar storage allocation limited by MEMLIMIT specification
- APIs are also provided in C/C++ to obtain storage below-the-bar and below-the-line
  - __malloc31() - obtain 31-bit addressable storage
  - __malloc24() - obtain 24-bit addressable storage

AMODE64 APIs

- Callable services are not supported in their current form (CEExxxx). C/C++ functions are used instead.
  - In some cases a similar C API already existed
    - CEEGTST and malloc()
  - In some cases a new C API is introduced
    - CEE3ABD and __cabend()
  - Most require __le_api.h be included at compile time
  - In some cases there is no corresponding C API or the function is not supported in AMODE 64.
    - Additional APIs will be added as required
- LE anchor is changed, from CAA to LAA, and R12 not required to contain address of CAA
- See Appendix for more details.

AMODE64 Message Handling

- Message Handling
  - Messages written using Language Environment services will now be sent to stderr
    - When stderr is redirected so are these messages
  - stderr will default to the DD SYSOUT except in the USS shell
    - The MSGFILE run-time option is not honored
  - The following services are available to the user to write to the LE style message (see Appendix):
    - __le_message_add_insert()
    - __le_message_get()
    - __le_message_get_and_write()
    - __le_message_write()

AMODE64 I/O Support

- Non-message Language Environment AMODE 64 I/O support
  - All access methods supported by the AMODE 31 C/C++ run-time continue to be supported… except
    - HFS I/O supports user buffers allocated above the bar
      - All other I/O must use system/library buffers below the bar
    - HIPERSPACE is not supported for AMODE 64
      - All requests to open a file "type=memory(hiperspace)" will be treated as a regular memory file
- Global streams are not supported (not required without nested enclaves)

AMODE64 user exit

- Language Environment AMODE 64 User Exit
  - The only user exit supported in AMODE 64 is the abnormal termination user exit (ATUE).
  - New parts (SCEESAMP)
    - CELQXTAN AMODE 64 ATUE CSECT
    - CELQWEXT Sample job to install exit
    - CEEQWATX Sample ATUE exit
  - The exit will be called AMODE 64 and XPLINK
    - Must be LE-conforming assembler
    - R1 contains the address of the CIB

LE Exception Handling

- Provide support for a 'stack frame' (thread) based exception handler in 64-Bit LE.
  - Multiple Exception Handlers may be registered for a given thread. However, only one may be registered per stack frame
  - A maximum of one Exception Handler is active at any point in time
  - Exception Handlers are only driven for program exceptions and abends
  - The Exception Handler should not 'return' to LE. If it does, the thread or the environment will be abnormally terminated
- Two new APIs allow one to register and unregister an Exception Handler
  - __set_exception_handler() -- Register an Exception Handler Routine
  - __reset_exception_handler() -- Unregister an Exception Handler Routine

AMODE 64 DLLs

AMODE 64 DLL Support

- A DLL runs in the environment of the code that links to it
  - Either AMODE 31 or AMODE 64
- It is not possible to create a DLL which will run in both AMODE 31 and AMODE 64
  - You may have common source
- Therefore, as with most other platforms, 2 separate DLLs are required to support AMODE31 applications and AMODE 64 applications
  - DLL providers may need to provide both a 31-bit and 64-bit DLL to support both modes.
- Assembler DLL support is also now available.

Assembler DLL Support - new

- Support is being provided to:
  - Export Assembler entry points as DLL functions
  - Export Assembler data as DLL data
  - Import DLL functions into Assembler code
  - Import DLL data into Assembler code
- Support provided for these environments:
  - 31-bit "standard" LE
  - 31-bit XPLINK LE
  - 64-bit LE
- XPLINK Asm parameter passing restricted to "by reference"

Pre-initialized Environments

AMODE64 Pre-Initialization

- Language Environment AMODE 64 Pre-Initialization support (PIPI)
  - CELQPIPI (alias for CELQLIB) resides in SCEERUN2
    - supports unauthorized, problem state programs only
  - Three pieces
    - A non-LE conforming assembler driver (AMODE 64)
      - User written program to control the environment
      - Calls CELQPIPI with 64bit OS linkage (not XPLINK)
    - A PIPI Table
      - Identifies the routines to be executed in the environment
      - Routines must be AMODE 64 (XPLINK)
    - CELQPIPI services
      - Calls from the assembler driver use the services to control the environment

AMODE64 Pre-Initialization

- Language Environment AMODE 64 Pre-Initialization support (PIPI)
  - Two types of PIPI environments
    - Main
      - A new pristine environment is created for each call
      - Run-time options can be specified on each call
      - Support for multiple main environments
        - only one can be POSIX(ON)
    - Sub
      - Best performance
      - Environment is left in state of previous call
      - Run-time options can only be specified at init
      - Routine must be declared "fetchable"
      - Call sub by address supported

64 bit Debugging

Debugger support

- CEEDUMP
  - Content of CEEDUMP is being reduced in this environment.
  - SYSMDUMPs and IPCS verb exits will provide equivalent or greater debugging capabilities
    - VERBEXIT available for LE called LEDATA
  - The following is a list of what will be in a CEEDUMP:
    - Traceback of the routines on each stack
    - Condition information
    - Entry information
    - The entire content of each stackframe, including formatting of certain sections
    - The data around registers (96 bytes)
    - The run-time options report
    - The storage diagnostic report (if the HEAPCHK option was active)

Debugger Support

- Debugger support
  - IBM Debuggers – DBX
    - A 64 bit application capable of debugging 31bit and 64bit applications
    - uses industry standard DWARF architecture
    - Supports MVS 64 bit binary dumps
  - Non-IBM Debuggers are supported
  - Debug event handler – CELQVDBG
    - Is a DLL with an exported function called CELQVDBG
    - May be loaded from PDS or HFS (new for AMODE 31 too)
    - Controlled by environment variable
      - __CEE_DEBUG_FILENAME64
- Separate presentation on 64bit debugging

64 bit Exploitation Considerations

64 bit Exploitation considerations

- Pure C/C++ application
  - compile and build C/C++ application code w/LP64
    - make data neutral changes where needed
    - add conditional 64 bit logic to take advantage of larger addressable storage in same address space (_LP64 FTM)
  - package the 64 bit application / DLL
    - same name in different library/path
    - different name in same library/path

64-bit virtual exploitation considerations

- 64-bit virtual for large data buffer
  - create a new AMODE 64 program
    - do you have all the necessary 64-bit APIs support?
    - if not, may end up with a lot of amode switching
  - use existing AMODE 31 program and switch amode as required
    - may end up with a lot of amode switching
- 64-bit API support considerations - mixed HLL
  - use a thin layer of AMODE 64 code if all code cannot be converted to AMODE 64
  - parameter list can be 31-bit or 64-bit
    - avoid additional data move
  - different API name/same API name consideration

64-bit virtual exploitation considerations

- What if all your code cannot be converted to AMODE 64?
  - interface must be 64 bit
    - can build parameter list below the bar (31 bit storage)
  - switch to assembler
  - must be 64 bit on entry
  - switch to 31 bit and call the 31 bit code
    - the 31 bit code cannot run on the same stack
      - since stack is above the bar
    - 31 bit code cannot call the runtime
  - switch back to 64 bit
  - return to mainline application code

Appendix

Appendix - Publications

- New book
  - z/OS V1R6.0 Language Environment Programming Guide for 64-bit Virtual Addressing Mode, SA22-7569
- Existing books which include AMODE 64 information
  - SA22-7567 z/OS V1R6.0 Language Environment Concepts Guide
  - SA22-7564 z/OS V1R6.0 Language Environment Customization
  - GA22-7560 z/OS V1R6.0 Language Environment Debugging Guide
  - SA22-7562 z/OS V1R6.0 Language Environment Programming Reference
  - SA22-7566 z/OS V1R6.0 Language Environment Run-time Messages
  - GA22-7565 z/OS V1R6.0 Language Environment Migration Guide
  - SA22-7568 z/OS V1R6.0 Language Environment Vendor Interfaces
  - SA22-7563 z/OS V1R6.0 Language Environment Writing ILC applications
  - SA22-7821 z/OS V1R6.0 C/C++ Run-time Library Reference
- Red Paper
  - z/OS 64-bit C/C++ and Java Programming Environment - http://www.redbooks.ibm.com/abstracts/redp9110.html

AMODE 64 load modules

- All forms of Language Environment will continue to reside in datasets beginning with SCEE
  - All parts associated with AMODE 64 support will start with the CELQ prefix

64 bit module	31 bit module	source	function
CELQSTRT	CEESTART	Compiler emitted	module entry point
CELQMAIN	CEEMAIN	Compiler emitted	address of main(), env., inpl
CELQFMAN	CEEFMAIN	Compiler emitted	addr of FETCHABLE main()
CELQBST	CEEROOTA/B CEEROOTD/ CEEINT/ CEEBPIRA	SCEEBND2/SCEELKED	Bootstrap module.
CELQETBL	CEEBETBL	SCEEBND2/SCEELKED	Externals Routine Table
CELQSG03	CEESG003	SCEEBND2/SCEELKED	C Signature CSECT
CELQLLST	CEEBLLST	SCEEBND2/SCEELKED	Language List Table
CELQINPL	EDCINPL	SCEEBND2/SCEELKED	Initialization parameter list
CELQTRM	CEEBTRM	SCEEBND2/SCEELKED	termination stub

Run-Time Options for Amode64

- Some AMODE 24/31 run-time options remain
- Several AMODE 64 specific run-time options are added
  - HEAPPOOLS64 (HP64)
  - HEAP64 (H64)
  - IOHEAP64 (IH64)
  - LIBHEAP64 (LH64)
  - STACK64 (S64)
  - THREADSTACK64 (TS64)
- The list of available run-time options is reduced

Run-Time Options for Amode64

- Existing run-time options supported in AMODE 64
  - All other existing run-time options are not supported
  - * indicates some suboptions have changed or are not supported

ARGPARSE/NOARGPARSE
ENVAR
EXECOPTS/NOEXECOPS
FILETAG
HEAPCHK
INFOMSGFILTER
NATLANG
POSIX
PROFILE
REDIR/NOREDIR
RPTOPTS
RPTSTG
STORAGE *
TERMTHDACT *
TEST/NOTEST
TRACE *
TRAP

Run-time options

- HEAP64(init64,inc64,disp64,init31, inc31, disp31, init24, inc24, disp24)
  - Controls user heap storage
    - init64 Initial size of above the bar storage (in MB)
    - inc64 Increment size of above the bar storage (in MB)
    - disp64 KEEP or FREE (how to treat 64 increments)
    - init31 Initial size of above the line storage (in bytes)
    - inc31 Increment size of above the line storage (bytes)
    - disp31 KEEP or FREE (how to treat 31 increments)
    - init24 Initial size of below the line storage (in bytes)
    - inc24 Increment size of below the line storage (bytes)
    - disp24 KEEP or FREE (how to treat 24 increments)

Run-time options

- HEAPPOOLS64(ON|OFF,cell1 size, cell1 count,…,cell12 size, cell12 count)
  - ON|OFF Are heappools on?
  - cellX size Size of cells with this pool (8 to 64K)
  - cellX count Number of cells in this pool (min 4)
  - NOTE: Different from AMODE 24/31 (HEAPPOOLS) – was percentage (not changing in AMODE 24/31)

Run-time options

- IOHEAP64(init64,inc64,disp64,init31, inc31, disp31,
  init24, inc24, disp24)
  - Controls I/O storage for the run-time
    - init64 Initial size of above the bar storage (in MB)
    - inc64 Increment size of above the bar storage (in MB)
    - disp64 KEEP or FREE (how to treat 64 increments)
    - init31 Initial size of above the line storage (in bytes)
    - inc31 Increment size of above the line storage (bytes)
    - disp31 KEEP or FREE (how to treat 31 increments)
    - init24 Initial size of below the line storage (in bytes)
    - inc24 Increment size of below the line storage (bytes)
    - disp24 KEEP or FREE (how to treat 24 increments)

Run-time options

- LIBHEAP64(init64,inc64,disp64,init31, inc31, disp31, init24, inc24, disp24)
  - Controls heap storage usage for the runtime (non I/O)
    - init64 Initial size of above the bar storage (in MB)
    - inc64 Increment size of above the bar storage (in MB)
    - disp64 KEEP or FREE (how to treat 64 increments)
    - init31 Initial size of above the line storage (in bytes)
    - inc31 Increment size of above the line storage (bytes)
    - disp31 KEEP or FREE (how to treat 31 increments)
    - init24 Initial size of below the line storage (in bytes)
    - inc24 Increment size of below the line storage (bytes)
    - disp24 KEEP or FREE (how to treat 24 increments)

Run-time options

- THREADSTACK64(initial, increment, maximum)
  - Controls the allocation of user stack
    - initial Size of initial stack (in MB)
    - increment Size of increments of stack (in MB)
    - maximum Maximum stack size (in MB)
  - NOTES:
    - THREADSTACK64 is always above the bar
    - THREADSTACK64 is always one contiguous segment
      - Initially we reserve "maximum" space
      - Only use initial size and increase in increments until maximum is reached.
  - THREADSTACK64 is downward growing (XPLINK)

Run-time options

- STACK64(initial, increment, maximum)
  - Controls the allocation of user stack
    - initial — Size of initial stack (in MB)
    - increment — Size of increments of stack (in MB)
    - maximum — Maximum stack size (in MB)
  - NOTES:
    - STACK64 is always above the bar
    - STACK64 is always one contiguous segment
      - Initially we reserve "maximum" space
      - Only use initial size and increase in increments until maximum is reached.
      - Only what is actually used, counts towards MEMLIMIT
  - STACK64 is downward growing (XPLINK)

Data sizes in LP64 data model

Data Type	Size in LP64	Size in ILP32
int	4	4
long	**8**	**4**
pointer	**8**	**4**
size_t	**8**	**4**
ssize_t	**8**	**4**
off_t	**8**	4
ino_t	4	4
useconds_t	4	4
int_addr_t	4	4
ptrdiff_t	**8**	**4**
rlim_t	**8**	4

AMODE64 APIs

- New AMODE 64 C APIs
  - Two additional C functions have been added to complement the malloc() function.
    - __malloc24() Obtain heap storage below the line
    - __malloc31() Obtain heap storage below the bar
  - __cabend() Similar to CEE3ABD
  - __le_condition_token_build() Similar to CEENCOD
  - __le_debug_set_resume_mch() Similar to CEEMRCM
  - __le_get_cib() Similar to CEE3CIB
  - __le_msg_add_insert() Similar to CEECMI
  - __le_msg_get() Similar to CEEMGET
  - __le_msg_get_and_write() Similar to CEEMSG
  - __le_msg_write() Similar to CEEMOUT

APPENDIX - C APIs

Callable Service	Description	AMODE 64 name
CEE3ABD	Terminate with ABEND code	__cabend() *
CEE3CIB	Return pointer to Condition Information Blk.	__le_get_cib() *
CEE3CTY	Set default country	setlocale()
CEE3DMP	Generate a Language Environment dump	cdump(), ctrace() or csnap()
CEE3GRN	Get routine name that incurred condition	sigaction()
CEE3MCS	Get default currency symbol	localeconv()
CEE3MDS	Get default decimal separator	localeconv()
CEE3MTS	Get default thousands separator	localeconv()
CEE3PRM	Query parameter string	argv[], argc
CEE3CMI	Store and load message insert data	

APPENDIX - C APIs

Callable Service	Description
AMODE 64 name	
CEEGMT	Get GMT time
	gmtime(time())
CEEGMTO	Get offset from GMT
	getenv(TZ)
CEEGPID	Get LE version and release
__librel() (existing func)	
CEEGTST	Get heap storage
	malloc()
CEELCNV	Query local numeric conventions
localeconv()	
CEELOCT	Get local time
	time()
CEEMGET	Get a message
	__le_msg_get() *
CEEMOUT	Write a message
	__le_msg_write() *
CEEMRCM	Move Resume Cursor – Machine
__le_debug_set_resume_mch() *	

LE Anchor

- LE Anchor
  - Register 12 can no longer be relied upon to contain the address of LE's Common Anchor Area (CAA).
  - A new LE anchor, Library Anchor Area - LAA, is being defined that is anchored in the PSA (prefix save area) field PSALAA at +x'4B8'.
  - The LAA points to a new 'Library Common Area - LCA' which has a pointer to the existing updated CAA.
  - This basing now maps through the PSA->LAA->LCA->CAA.
  - All / most external fields now moved to LAA or LCA

External Control Structures

- CEELAA
  - New anchor for 64 bit Language Environment
    - CAA is not the LE anchor in 64 bit
  - The PSALAA field (+ x'4B8') always points to the LAA for the currently active unit of work
  - It is a key 0 control block, anchored from the STCB.
- CEELCA
  - allocated in the key of the caller, when a LE environment is actually initialized.
  - It is allocated along with the LE control blocks and is pointed to by the CEELAA (+ x'60').
- CEEDIA
  - A new LE control block has been defined for the Debugger hooks, called the Debugger Interface Area (DIA)
  - The DIA is anchored off the LCA at an architected offset (+16 or 0x10).
  - The DIA is allocated in 31-bit addressable storage during LE initialization and will remain for the duration of the process.
  - The size and order of the hooks remains consistent with the CAA debugger section.

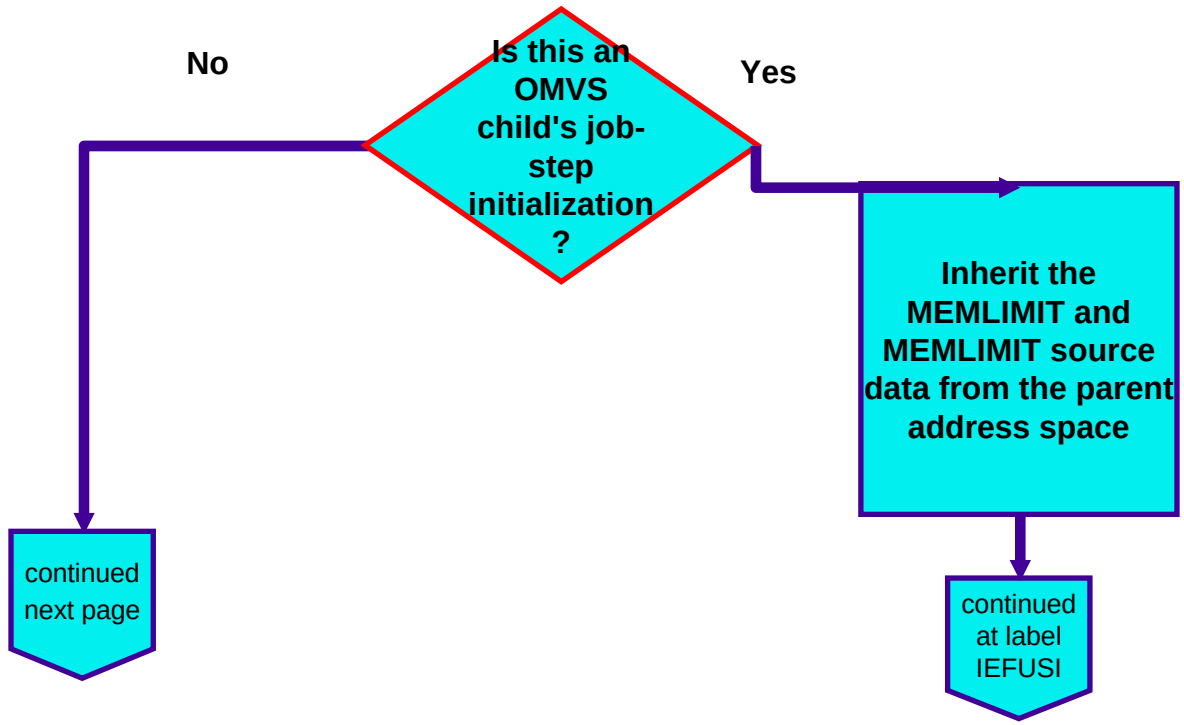
External Control Structures

- CAA – Common Anchor Area (external section)
  - Pointed to from LCA (CEELCA_CAA)
  - Note: R12 no longer points to CAA
- EDB – Enclave Data Block (external section)
  - Pointed to from CAA (CEECAAEDB)
- PCB – Process Control Block (external section)
  - Pointed to from CAA (CEECAAPCB)
- RCB – Region Control Block (external section)
  - Pointed to from CAA (CEECAARCB)

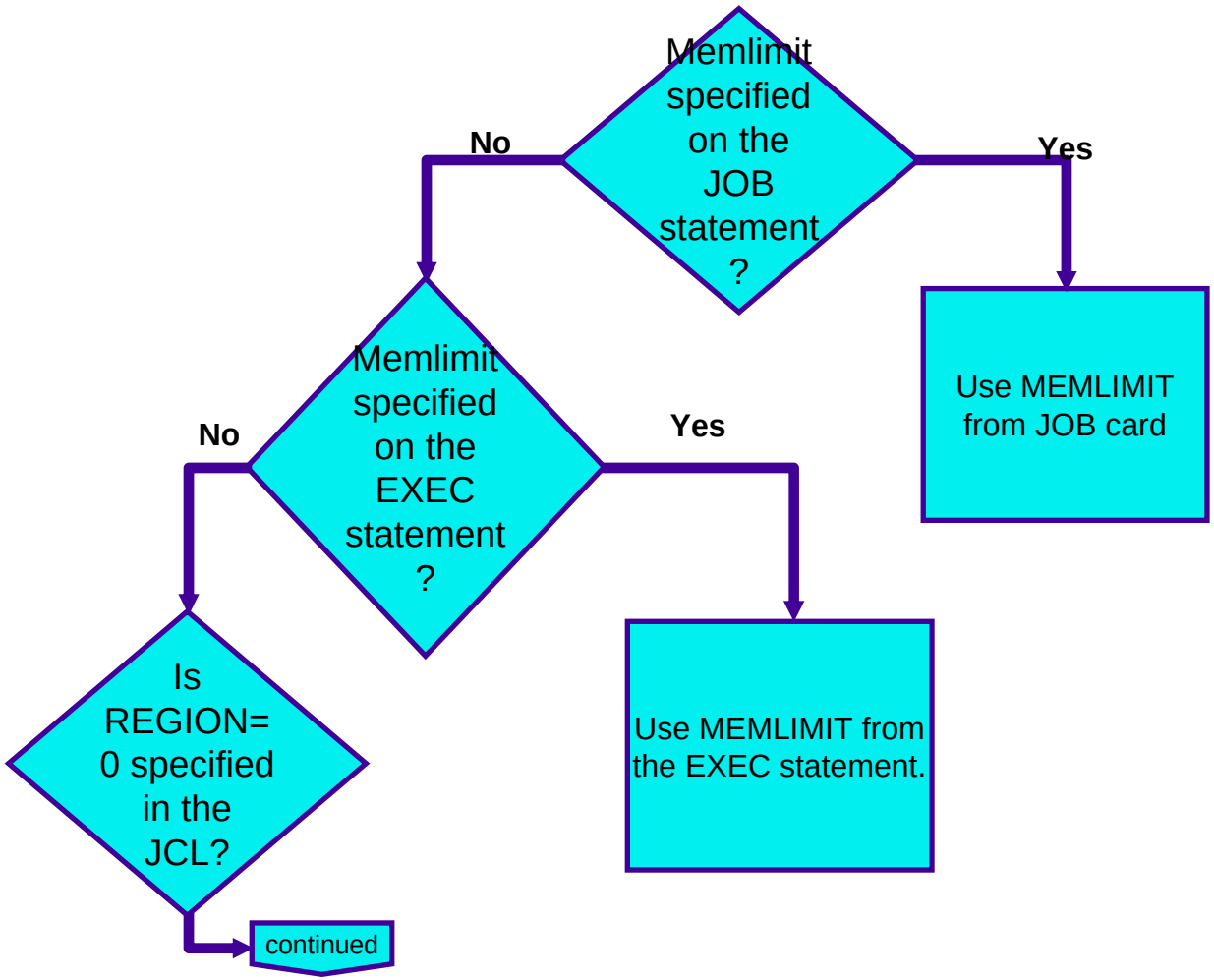
Setting MEMLIMIT

- How is 64-bit virtual limit(MEMLIMIT)?
  - SMFPRMxx parameter MEMLIMIT sets the system defaults for each address space
  - Use SETSMF command to change limit online
  - MEMLIMIT keyword in JOB & EXEC JCL
  - MEMLIMIT in RACF OMVS segment for Unix System Services users
  - MEMLIMIT on spawn system call using inheritance structure
- Note: System programmer must set the SMFPRMxx parameter as system default based on their system requirement
- How is the final MEMLIMIT is chosen?
  - JOB MEMLIMIT overrides EXEC MEMLIMIT
  - If no JOB or EXEC MEMLIMIT specified
    - and REGION=0, MEMLIMIT=no limit
    - and REGION^=0, use MEMLIMIT in SMFPRMxx
  - IEFUSI exit can override the SMFPRMxx MEMLIMIT value
- Setting MEMLIMIT
  - MEMLIMIT(NOLIMIT|nnnnnM| nnnnnG| nnnnnT| nnnnnP

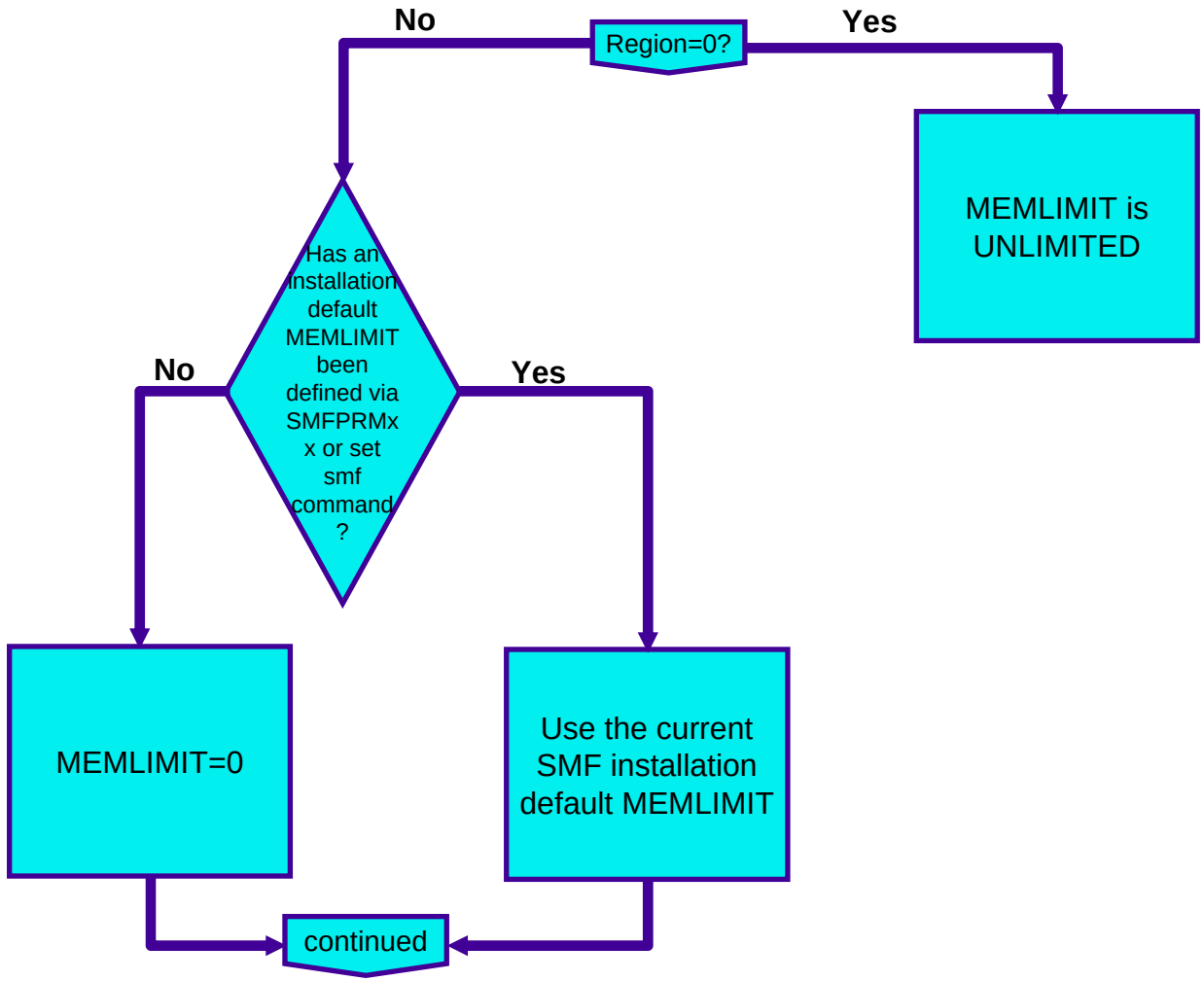
Determining MEMLIMIT



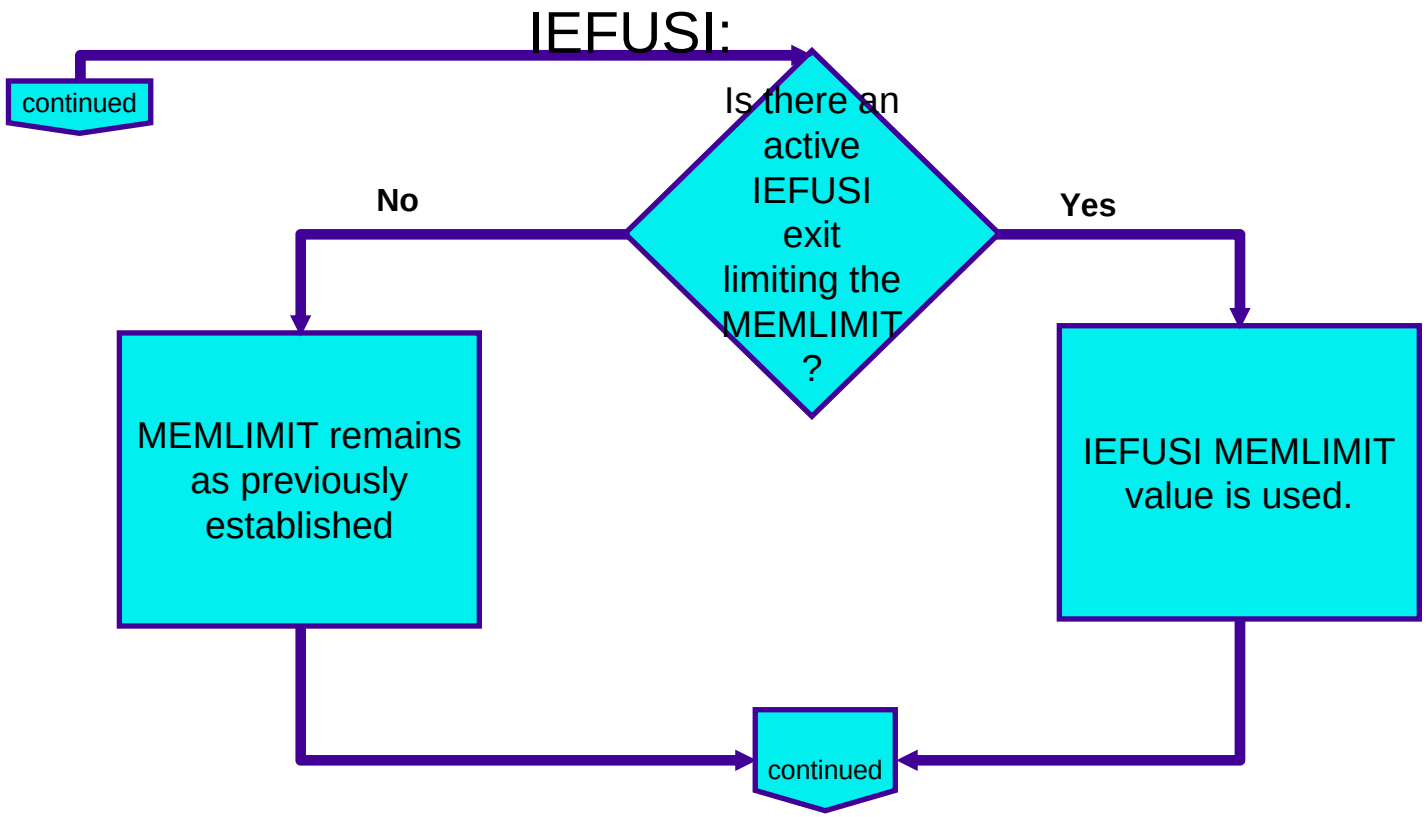
Determining MEMLIMIT (continued)



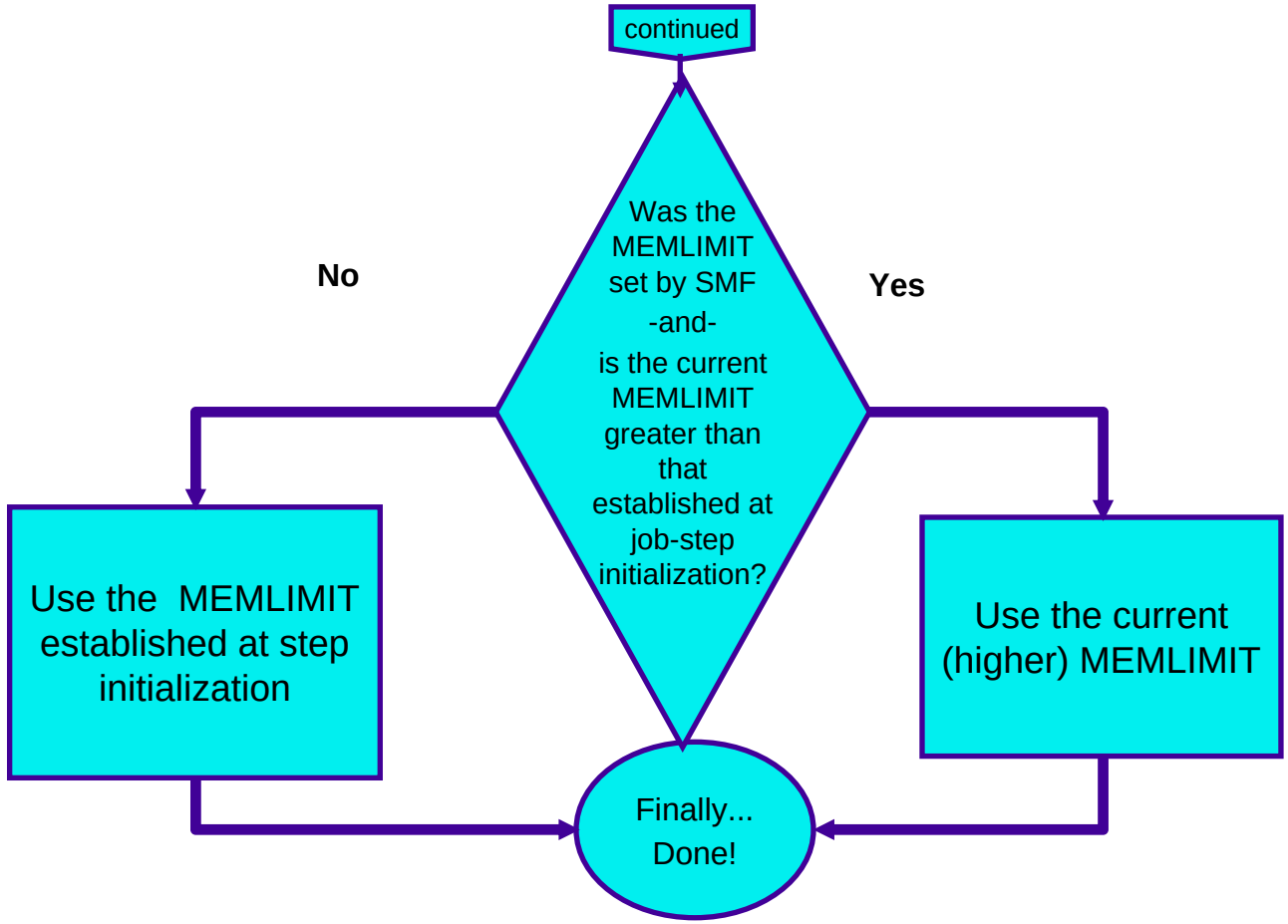
Determining MEMLIMIT (continued)



Determining MEMLIMIT (continued)



Determining MEMLIMIT (continued)



Additional Information

- Language Environment AMODE 64
  - Items not supported initially
    - HLL languages other than C and C++
    - Nested Enclaves
    - Hex math (added in z/OS V1.7)
    - CICS
    - IMS
    - LRR
    - SPC
    - PICI