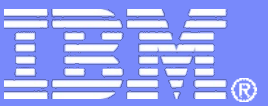


Language Environment for Dummies

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®
- DB2®
- Language Environment®
- OS/390®
- z/OS®

* 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.

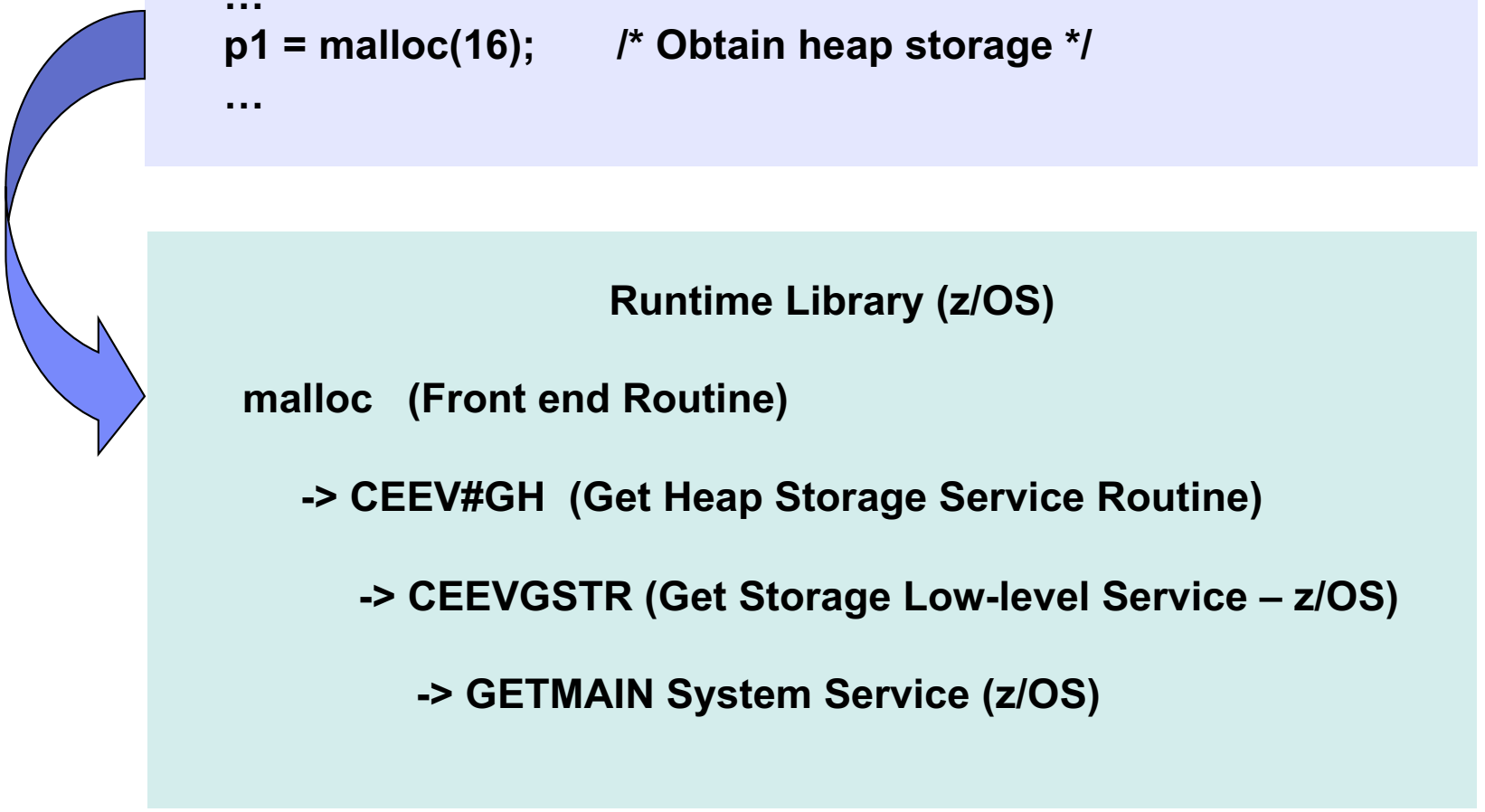
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products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

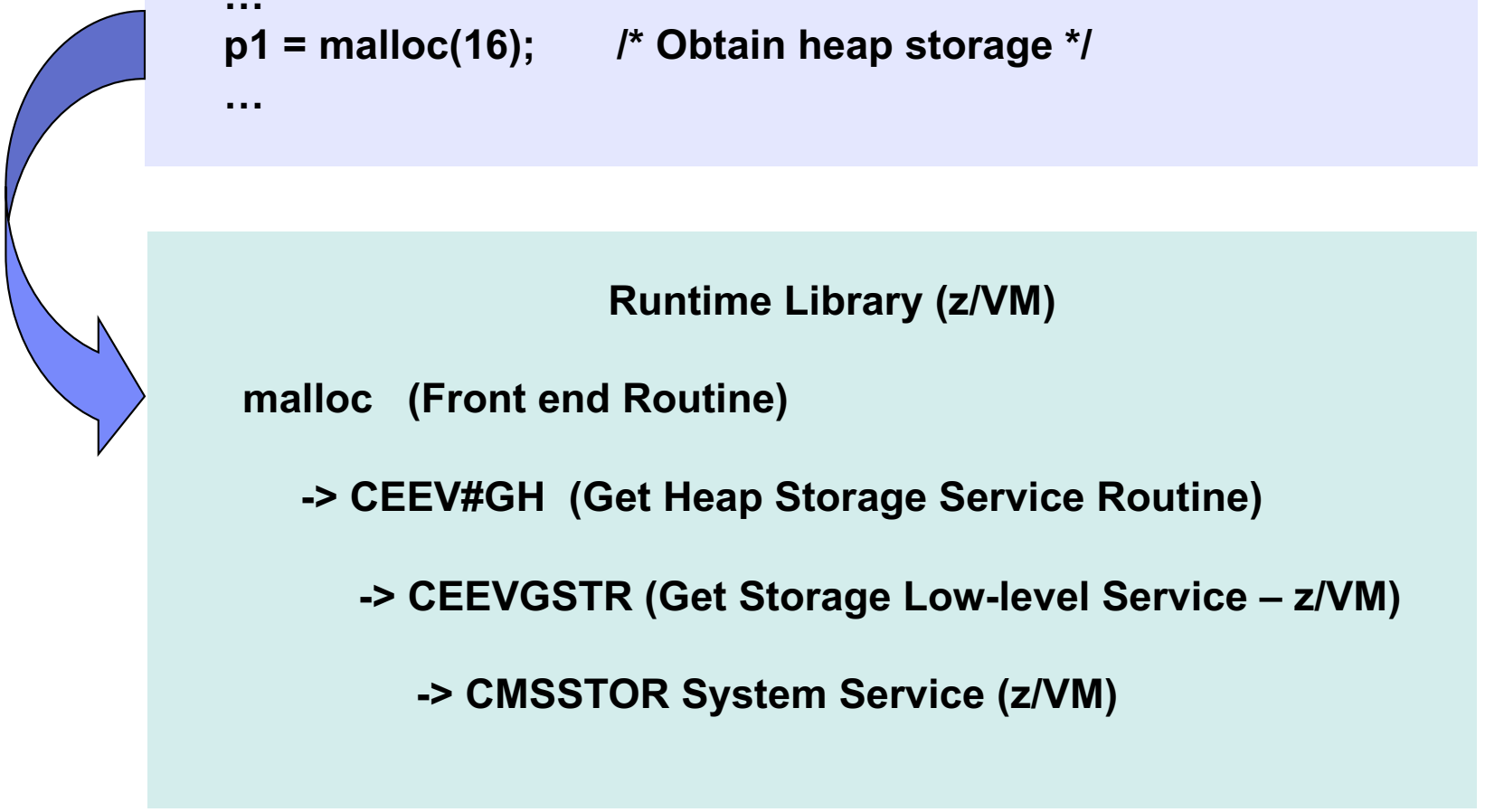
Agenda

- What is a Runtime Library?
- Why LE?
- LE Terminology
- LE CEL Functions
- Setting Runtime Options
- Appendix

What is a Runtime Library?



What is a Runtime Library?



What is a Runtime Library?

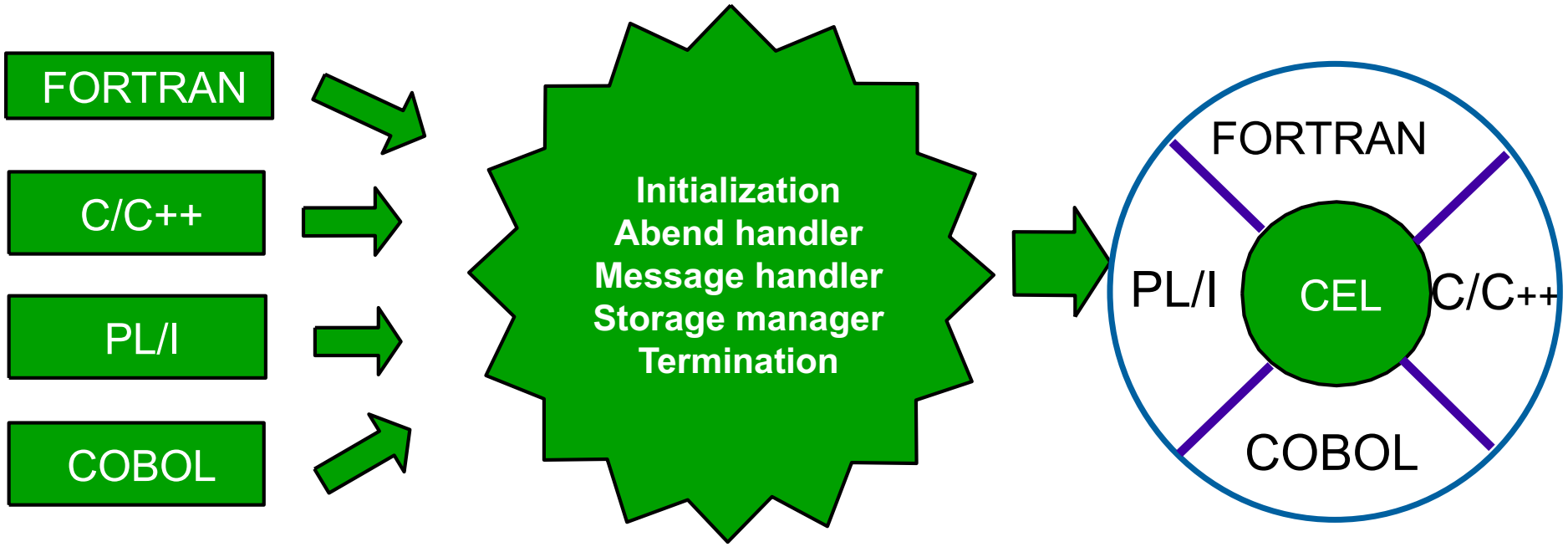
- A Runtime Library works together with the code produced by a compiler to provide functionality for an application
  - Obtain and manage storage
  - Read and write data
  - Perform math calculations

- There are advantages to providing function in a Runtime Library
  - Greatly reduces need for the compilers to generate the code
  - Shields the languages from needing detailed knowledge of the underlying operating system and hardware
  - Greatly reduces the need to recompile and re-link when fixes are required to runtime functions

So, Why Language Environment?

- Since their creation, customers were having trouble getting COBOL and PL/I to play nicely together
  - COBOL and PL/I each designed to be stand-alone, unaware of each other
    - When leaving a COBOL program to return to a PL/I program, the COBOL library might free storage that PL/I still wanted
    - Language-specific Math Libraries produced different results
- Customers at GUIDE and SHARE worked with IBM to design a solution
  - The result: **Language Environment**

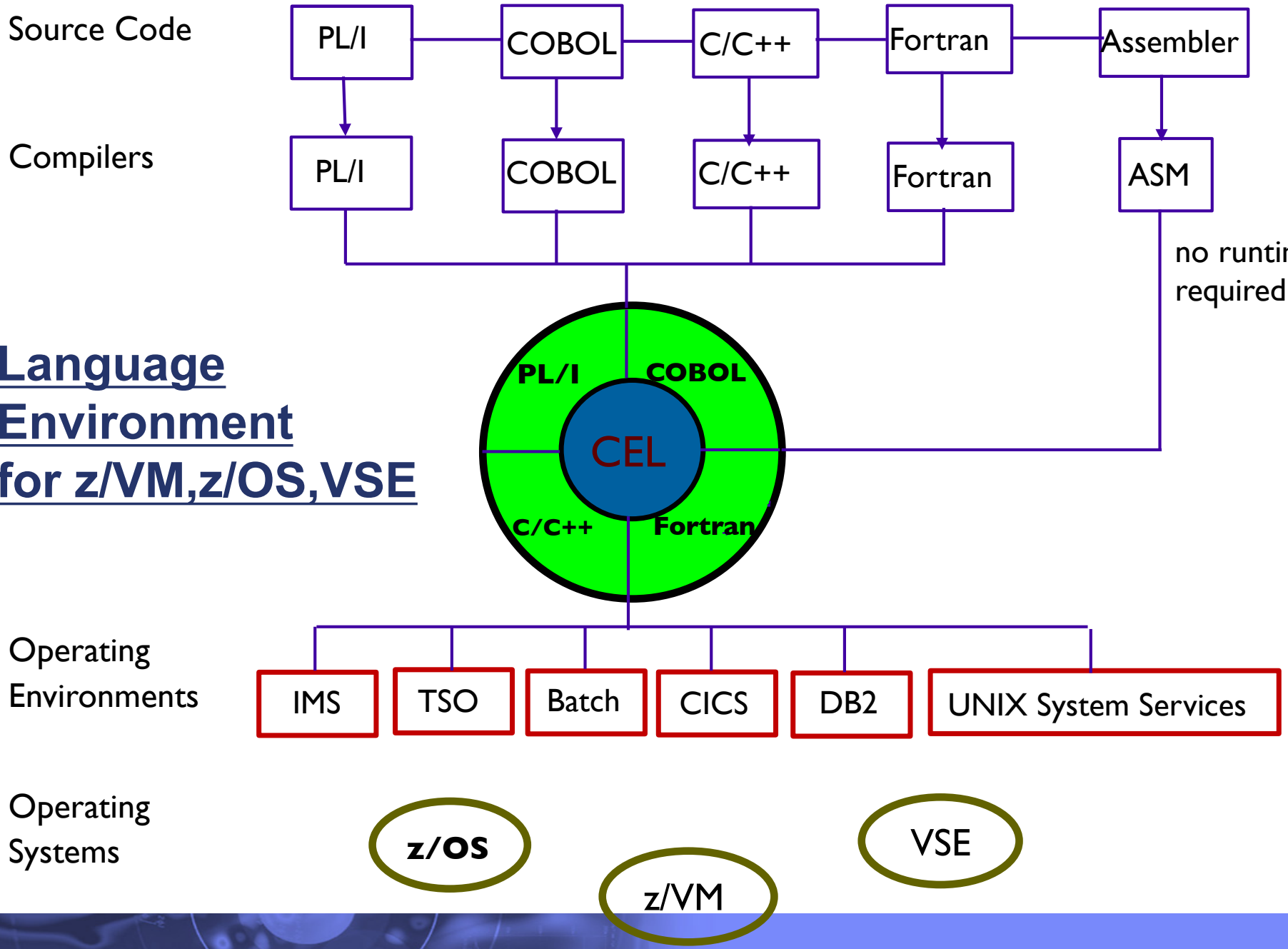
Time to make the doughnut…



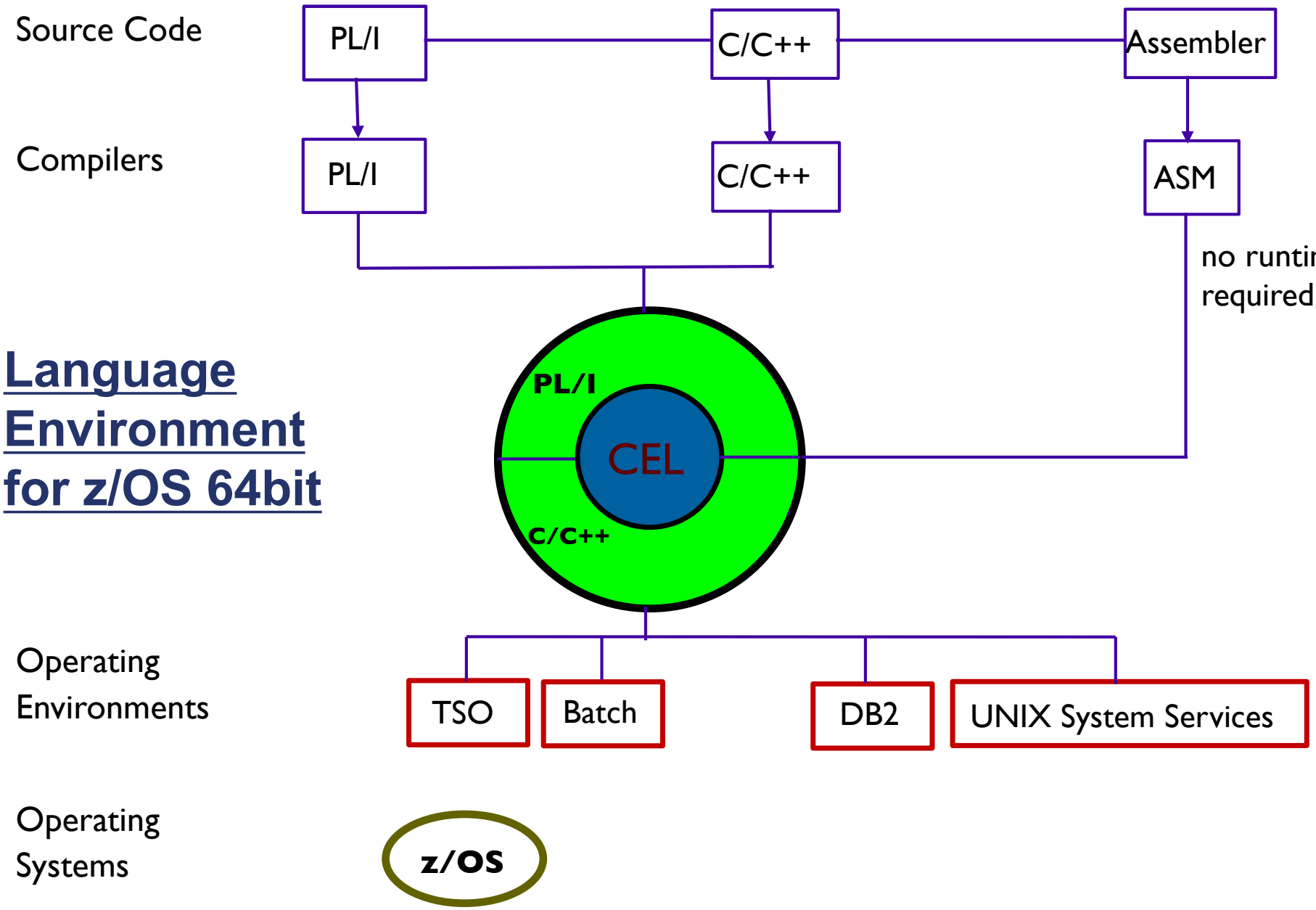
- Pre-LE environment
  - 4 independent products
  - upward incompatibilities
  - loose adherence to standards
  - purely a customer application enabler
- LE environment
  - 1 product for z/OS (also z/VM and VSE)
  - 100% upward/downward compatibility
  - strict adherence to standards
  - part of the z/OS base
  - exploiters include USS, TCP/IP, BCPii, LOTUS Domino, WebSphere, etc...

Other Advantages

- Language Environment not only helped the languages to cooperate with each other, but also allowed member languages to share each other's features. For example:
  - COBOL can use the C and PL/I condition handling infrastructure
  - Storage managed in a 'common' fashion
  - All languages now access the excellent Fortran library math routines
  - "hybrid" languages – Enterprise PL/I


Source Code
PL/I
COBOL
C/C++
Fortran
Assembler
Compilers
PL/I
COBOL
C/C++
Fortran
ASM
no runtime required
Language Environment for z/VM,z/OS,VSE
PL/I
COBOL
CEL
C/C++
Fortran
Operating Environments
IMS
TSO
Batch
CICS
DB2
UNIX System Services
Operating Systems
z/OS
z/VM
VSE

Language Environment for z/OS 64bit



LE Terminology - Program Management

- **main program** – the routine that causes the LE environment to be initialized
- **routine** either a procedure, function, or subroutine

  Equivalent HLL terms:
  - COBOL - program
  - C/C++ - function
  - PL/I - procedure, BEGIN block
- **ILC** – inter-language communication – application contains a mixture of languages, which introduces special issues
  - how the languages' data maps across load module boundaries
  - how conditions are handled
  - how data can be passed and received by each language

LE Terminology - Program Management

- **member language** – a high-level language that is compiled with an LE-supported compiler
- **member event handler** - member-supplied routine that is called at various times as a program runs when a significant event has occurred, or when the environment needs some information that is held by the member
- **LE-Enabled** - Routine that can run with LE runtime, and may also run with previous runtimes. Cannot make use of Language Environment callable services.
- **LE-Conforming** - Routine that can run only with the LE runtime library. Can make use of LE callable services.

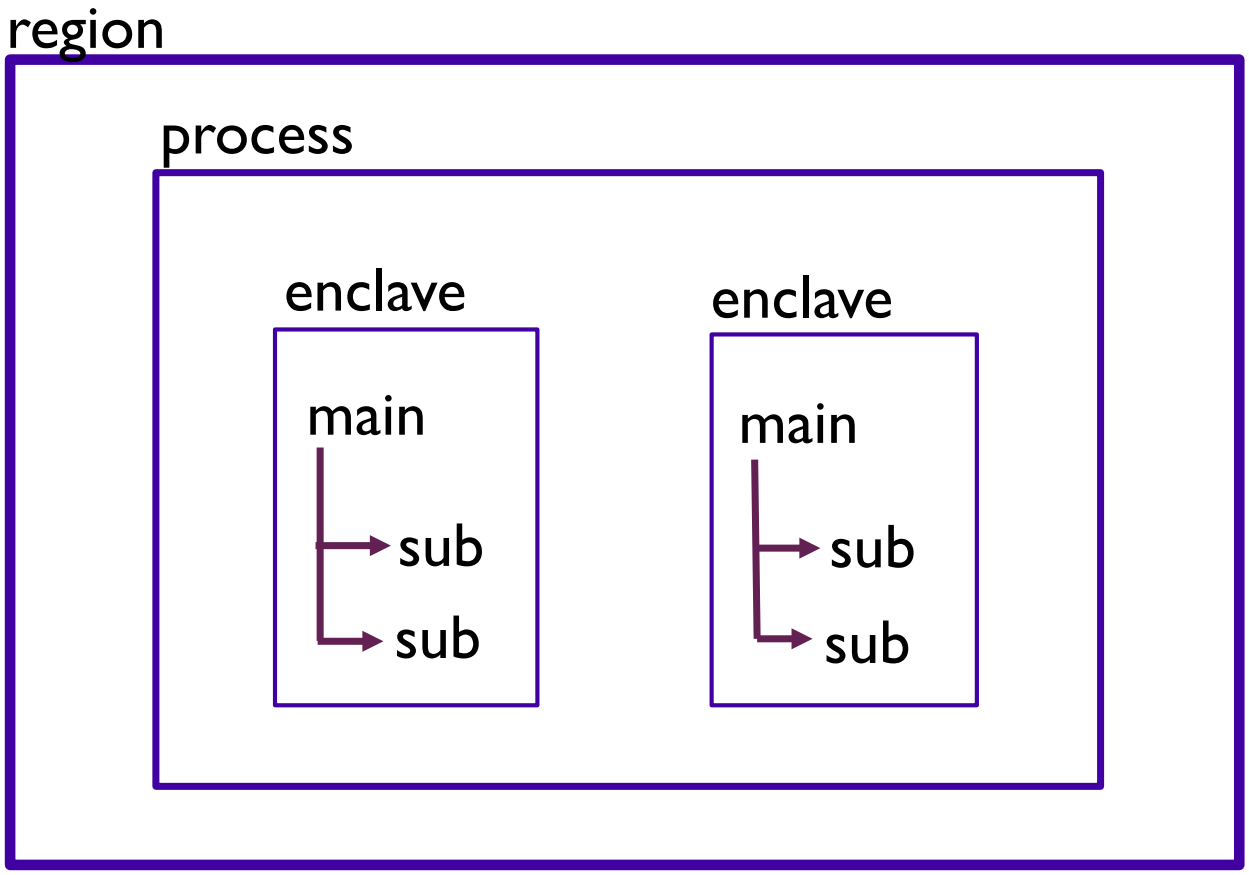
LE Terminology – Callable Services

- **LE Callable Services** – programmatic way of utilizing LE services
  - AWI - Application Writer Interface
  - CWI - Compiler Writer Interface
  - CEE prefixed – general to all platforms
  - CEE3 prefixed – specific to only z/OS

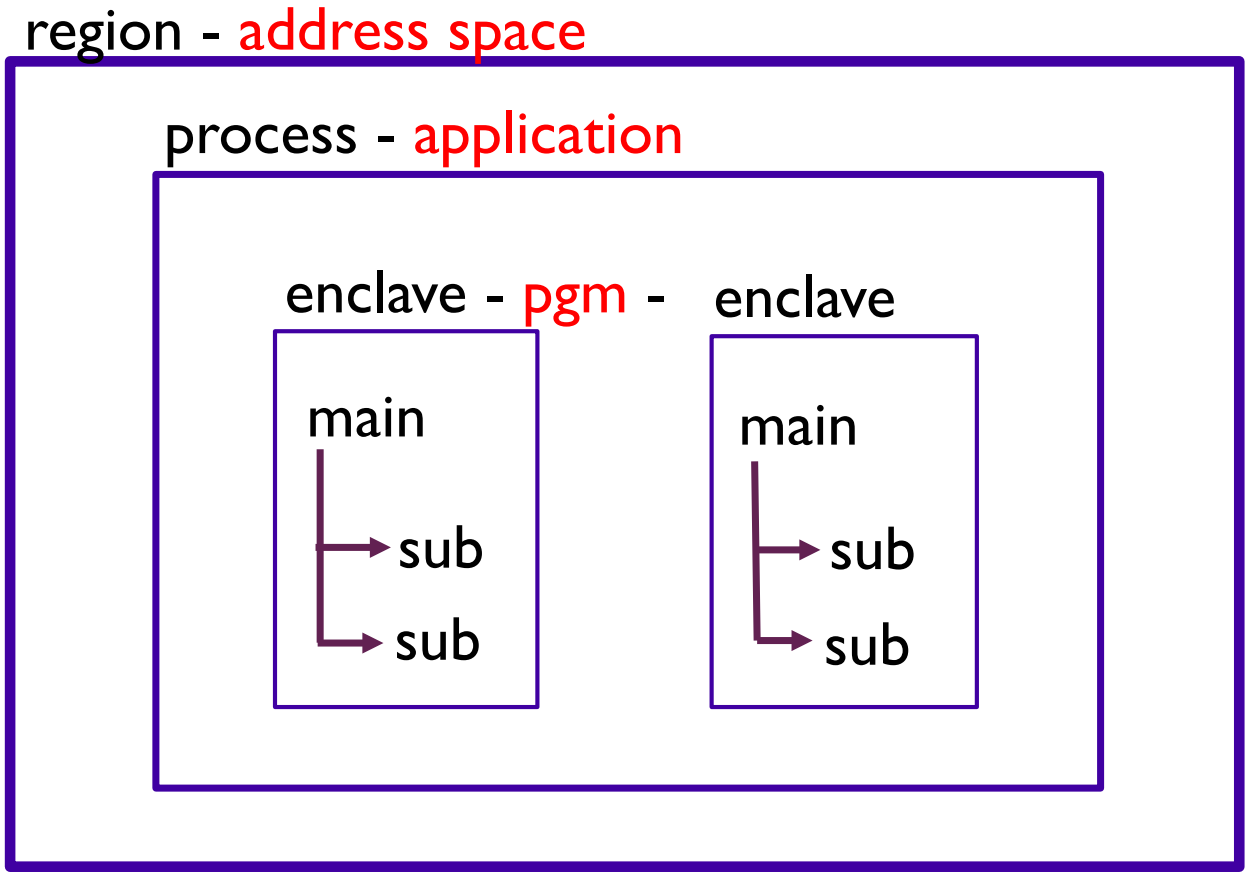
LE Terminology – Program Model

- **region** - the range of storage the application set runs in
- **process** - set of applications that accomplish a task
- **enclave** - an application - set of modules that accomplish some subtask
- **thread** - dispatchable unit of work that shares storage with others in the enclave

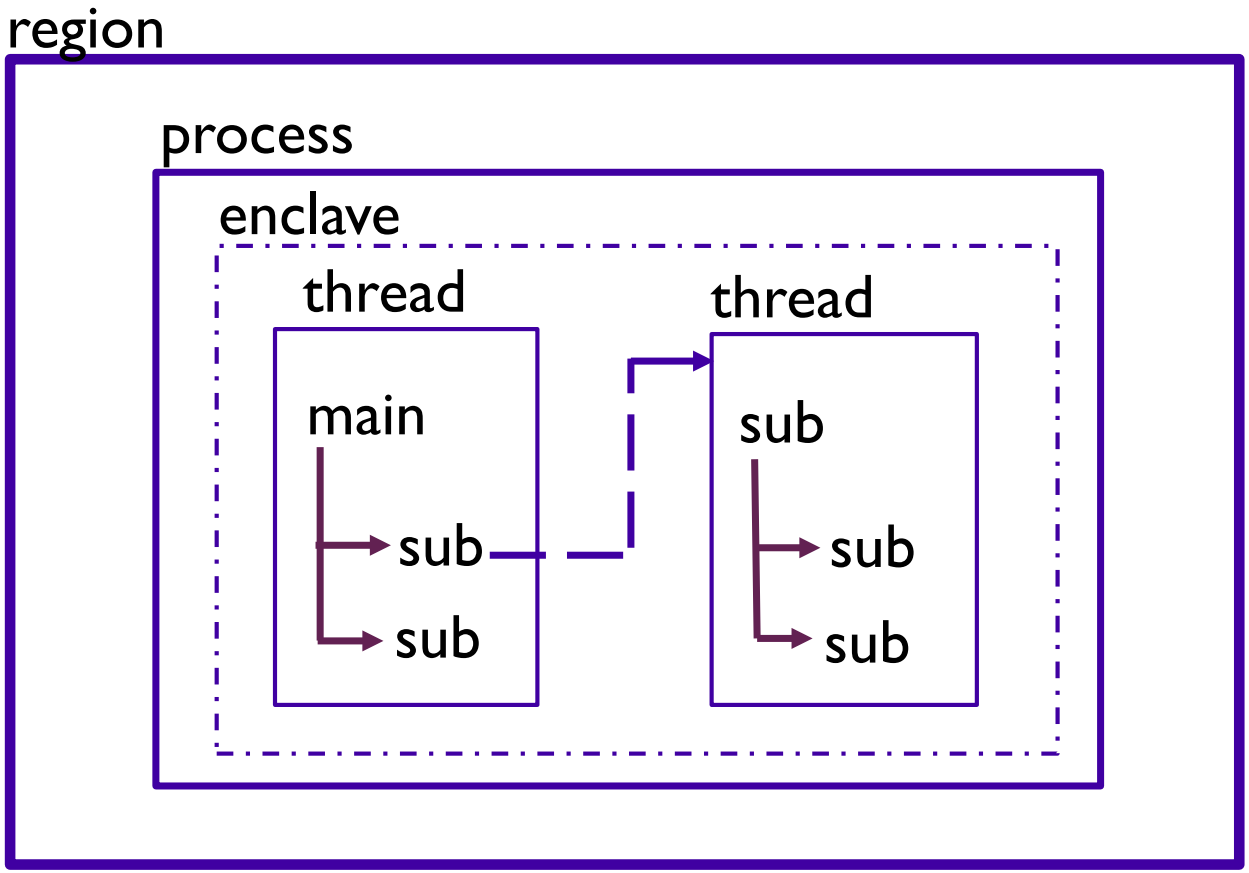
LE Terminology - Program Model



LE Terminology - MVS 'Model'



LE Terminology – Multi-threading 'Model'



LE CEL Functions

- CEL is a set of common functions and routines used by all member languages of LE
  - Initialization/Termination
  - Storage Management
  - Condition Handling
  - Message Services
  - Date/Time Services
  - Math Functions
- Behavior customizable by the use of Runtime Options

Common LE Functions – Initialization/Termination

- LE code linked with the module begins a bootstrap process to initialize LE
  - initial storage is obtained
  - LE Program Model levels are built
    - active member language specific runtime is initialized via event handler calls
- Control is given to the application code
- Once the application ends and 'returns' to LE
  - The LE environment is terminated via cleanup of Program Model levels
  - System resources obtained during initialization and throughout the execution of the application are cleaned up

Common LE Functions - Storage Management

- LE manages two types of storage for use by the application (and itself):
  - HEAP - used for COBOL WORKING-STORAGE, C malloc, PL/I ALLOCATE, and COBOL ALLOCATE (as of V6.1)
  - STACK - module linkage (save areas), C and PL/I automatic variables, COBOL LOCAL-STORAGE
- Initial storage is obtained with one GETMAIN and managed internal to LE

Common LE Functions - Condition Handling

- Condition - Any change to the normal flow of a program
  - a.k.a. exception, interruption
  - Could be detected by hardware or software (ours or yours)
- Condition Handler – A routine called by LE to respond to a condition
  - Registered by application using CEEHDLR, or part of a member language semantics, such as PL/I ON statements
- Condition Handler Response
  - Resume – after corrective action taken, control returns to a ‘resume cursor‘
    - Either back to point of failure, or to a new resume point set by the condition handler
  - Percolate - decline to handle the condition, LE calls next condition handler
  - Promote - change condition meaning and percolate

Common LE Functions - Condition Handling

- Diagnostic Documentation
  - Messages (same as module prefixes)
    - CEE CEL
    - IGZ COBOL
    - IBM PL/I
    - AFH FORTRAN
    - EDC C/C++
  - ABEND Codes
    - User ABENDs U4000-4095 reserved by LE
    - Usually have reason codes to help isolate the problem
  - CEEDUMP and/or system dump
  - Runtime Options Report
  - Storage Report

Common LE Functions - Message Services

- allows HLLs to issue common messages
- messages written to a common place - LE's MSGFILE
- can be formatted in:
  - Mixed-case American English (ENU)
  - Uppercase American English (UEN)
  - Japanese (JPN)

Common LE Functions – Date/Time Services

- provides a consistent answer when requesting date and time from the running system
- format date and time by country code
- parse date and time values
- convert between different formats (Gregorian, Julian, Asian, etc)
- calculate days between dates, elapsed time
- get local time

Common LE Functions – Math Services

- derived from FORTRAN math functions
- binary, single floating point, double floating point, IEEE support
- See the LE Programming Reference for a complete list

Runtime Options

- Allows users to specify how Language Environment behaves when an application runs
  - Performance tuning
  - Error handling characteristics
  - Storage management
  - Production of debugging information
- May be set in many different locations with varying scopes

Setting Runtime Options

- The default RTOs for applications across all systems
  - **IBM-supplied defaults**
    - Base set of values for Language Environment RTOs
- To set default RTOs for applications on one or more systems
  - **System defaults**
    - Options specified in a PARMLIB member (CEEPRMxx)
    - Options specified with an operator command (SETCEE)
- To affect applications running within a region
  - **Region Level Overrides** (CEEROPT/CELQROPT)
    - CICS TS, LRR users (e.g. IMS), also Batch
    - Separate module loaded at runtime during region initialization
    - CLER transaction for CICS environment (RTO subset)

Setting Runtime Options…

- To provide RTO settings for a specific application:
  - **Application Level Overrides** (CEEUOPT/CELQUOPT)
    - CSECT linked with the application
  - **Programmer Overrides**
    - #pragma runopts for C/C++
    - PLIXOPT for PL/I
- To provide RTO settings for a given run of an application:
  - **Program Invocation Overrides**
    - USS shell: export _CEE_RUNOPTS='runtime options'
    - In batch, on EXEC card: PARM=
  - **DD:CEEOPTS Overrides**
    - Optional data set in which runtime options may be specified

Setting Runtime Options…

- Options Merge (priority)
  - Program Invocation Overrides
  - DD:CEEOPTS Overrides
  - Programmer Overrides
  - Application Level Overrides
  - Region Level Overrides (where applicable)
  - System Defaults (CEEPRMxx and SETCEE)
  - IBM-Supplied Defaults

Key Runtime Options

- Subtopics
  - Tuning
  - Diagnostics

Key Runtime Options - Tuning

- ALL31
  - Indicates whether application runs entirely AMODE 31
- HEAP / ANYHEAP / BELOWHEAP
  - Controls size, location, and disposition of heap segments
- STACK
  - Controls size, location, and disposition of stack segments
- RPTSTG
  - Produces a report that aids in tuning storage usage

Key Runtime Options - Diagnostics

- TERMTHDACT
  - Tells LE what type of diagnostic information to produce
- DYNDUMP
  - Tells LE to produce a dynamically-allocated dump for diagnostics
- HEAPCHK
  - Performs diagnostic checks of the user heap

Key Runtime Options – Diagnostics…

- STORAGE
  - Controls initial contents of storage when obtained or freed
- TRAP
  - Controls LE's condition handling
- RPTOPTS
  - Produces a report of the runtime options settings for a specific run of an application


LE
PL/I
C/C++
CEL
COBOL
Fortran