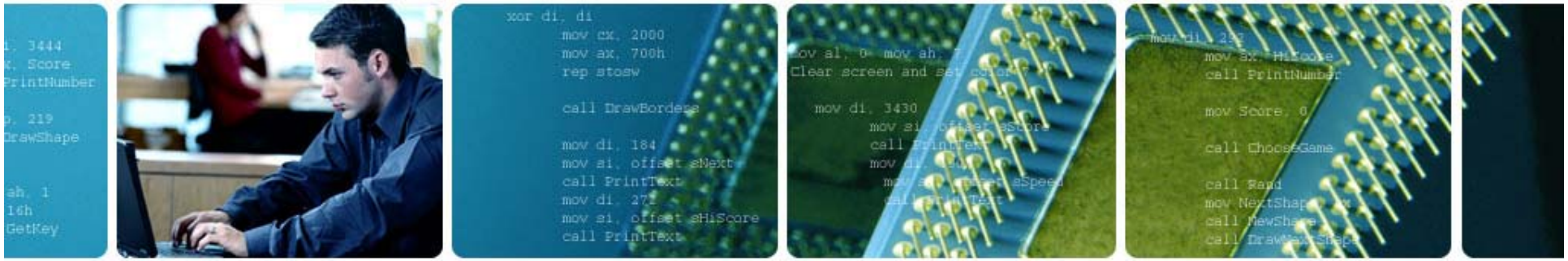
IBM

Introduction to the new mainframe

Chapter 9: Using programming languages on z/OS



Chapter 9 objectives

Be able to:

- List several common programming languages for the mainframe
- Explain the differences between a compiled language and an interpreted language
- Create a simple CLIST or REXX program
- Choose an appropriate data file organization for an online application
- Compare the advantages of a high level language to those of Assembler language
- Explain the relationship between a data set name, a DD name, and the file name within a program
- Explain how the use of z/OS Language Environment affects the decisions made by the application designer

Classification of programming languages (continued)

3rd generation

- Procedural languages, known as high-level languages (HLL)
- Example: COBOL
- Must be translated (compiled) before execution
- Usually portable (to an extent) across hardware and software platforms with a recompile

4th generation – 4GL

- Non-procedural languages
- Report generators
- Query languages
- Examples:
  - RPG, CSP, OMF, SQL

Classification of programming languages (continued)

Visual programming languages (or event-driven languages)

- Visual Basic, Visual C++

Object-Oriented language

- used in OO technology, e.g. Smalltalk, Java, C++

Other languages

- 3D applications

Scripting languages

- Perl
- REXX
- HTML

Choosing a programming language for z/OS

Which language to use? Factors to consider include:

- Response time requirements for the application
- Budget allotted for development and ongoing support
- Time constraints of the project
- Whether subroutines will be coded in different languages
- Whether to use a compiled or an interpreted language

Using Assembler language on z/OS

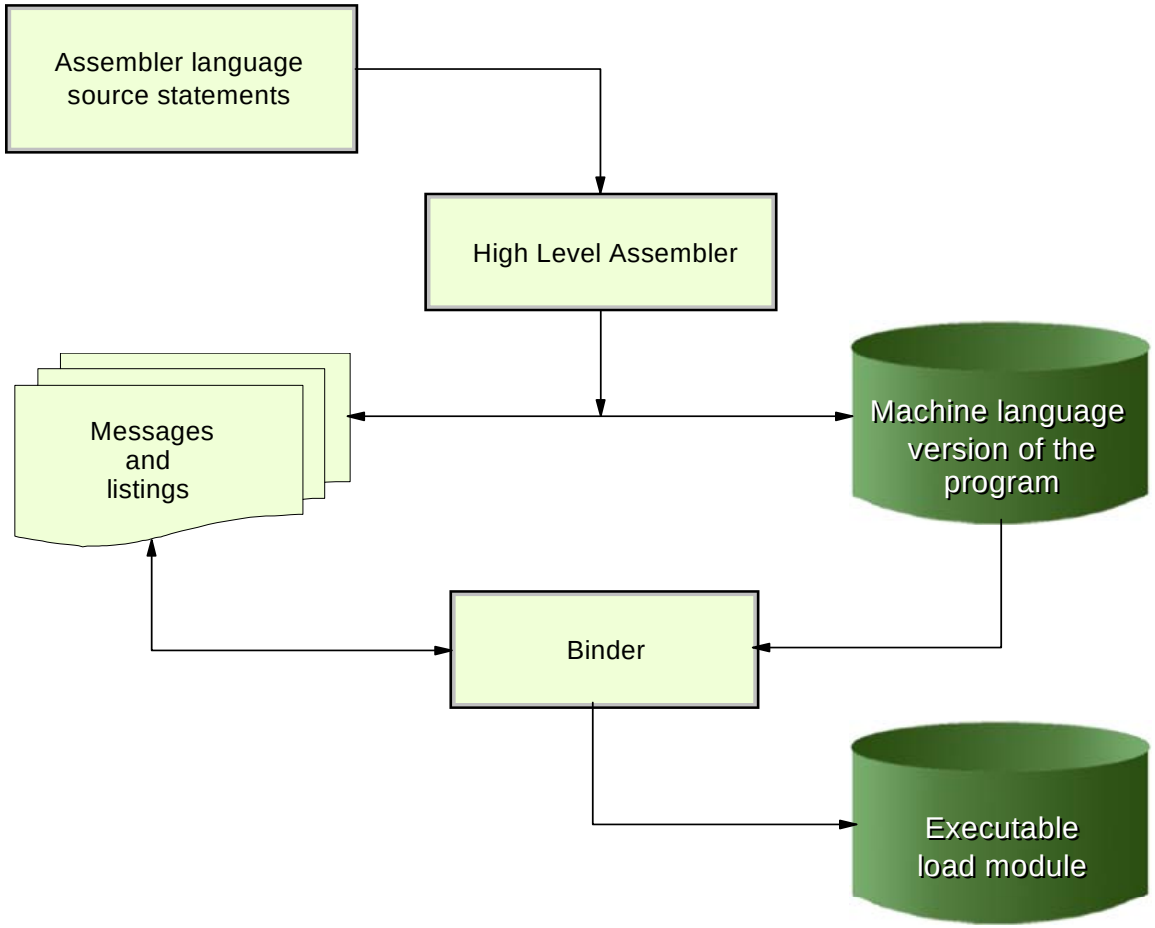
Assembler language

- Not usually used for application development
- Machine-specific

Used when:

- Accessing bits or bytes
- Accessing system control blocks
- Execution efficiency is needed (performance)
- Require high performance subroutines that can be called from HLL programs

From Assembler source to executable module



Using COBOL on z/OS

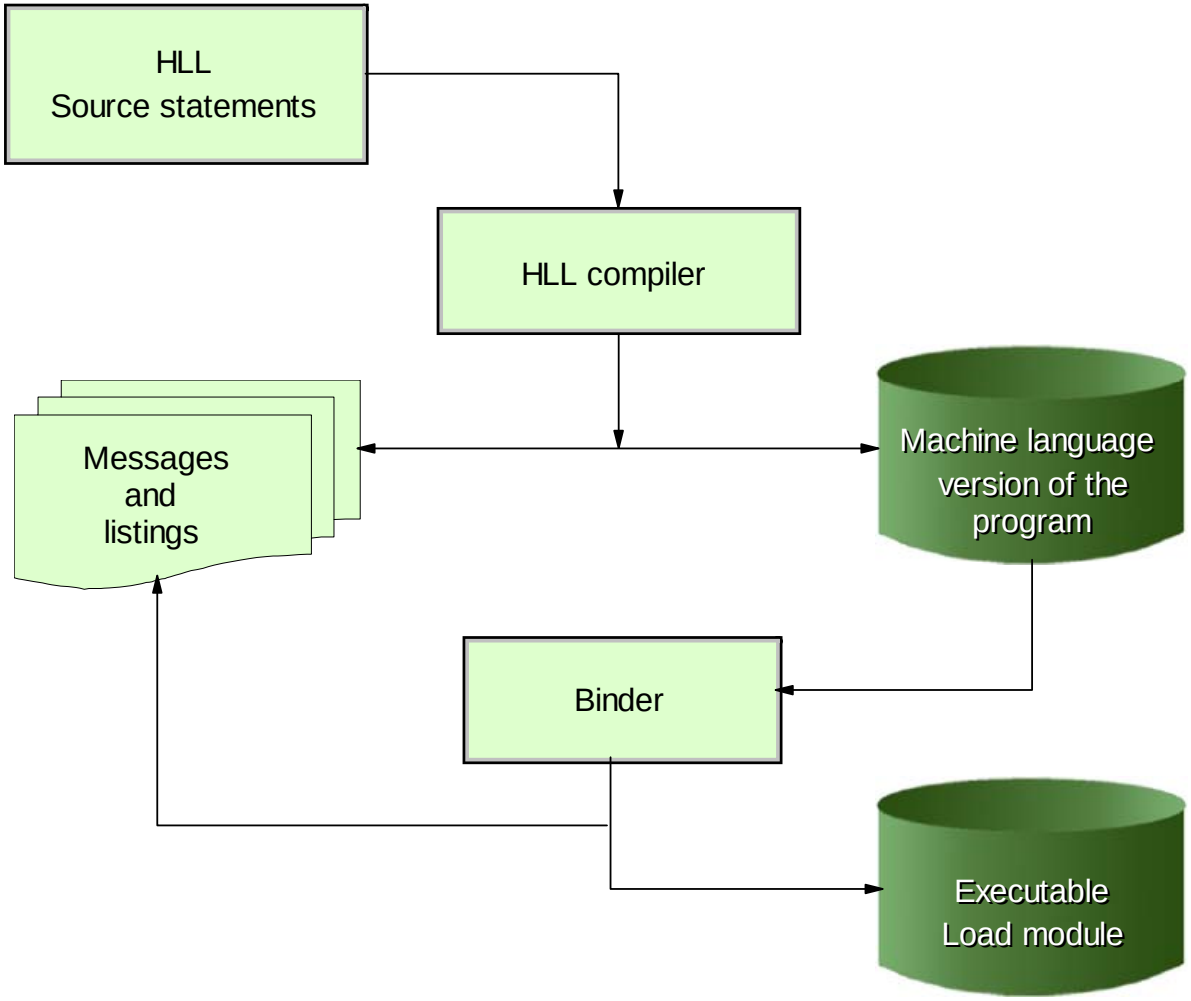
COBOL is an English-like programming language

Used for business-oriented applications

Capabilities of IBM Enterprise COBOL for z/OS and OS/390

- **Integrate COBOL applications into Web-oriented business processes**
- **Inter-operability with Java**
- **Parsing of data in XML and Unicode formats**

From HLL source to executable module



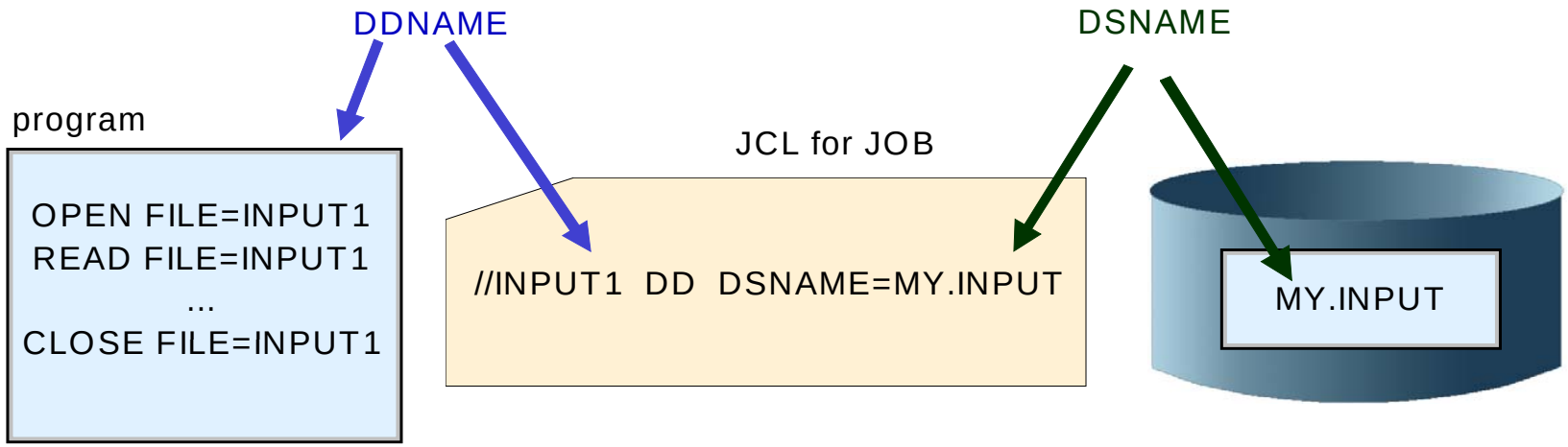
HLL relationship between JCL and program files

```
//MYJOB    JOB
//STEP1    EXEC IGYWCLG
...
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT INPUT ASSIGN TO INPUT1 .....
     SELECT DISKOUT ASSIGN TO OUTPUT1 ...
   FILE SECTION.
     FD INPUT1
        BLOCK CONTAINS...
        DATA RECORD IS RECORD-IN
     01 INPUT-RECORD
...
     FD OUTPUT1
        DATA RECORD IS RECOUT
     01 OUTPUT-RECORD
...
/*
//GO.INPUT1  DD DSN=MY.INPUT,DISP=SHR
//GO.OUTPUT1 DD DSN=MY.OUTPUT,DISP=OLD
```

HLL relationship between JCL and program (continued)

- COBOL SELECT statement makes the link between the DDNAMEs INPUT1 and OUTPUT1 and the COBOL FDs INPUT1 and OUTPUT1 respectively
- The COBOL FDs are associated with group items INPUT-RECORD and OUTPUT-RECORD
- The program is completely independent of the location of the data or the name of the data sets.

Relationship between JCL, program, and data set



Using PL/I on z/OS

Full-function, general-purpose high-level programming language

Suited for

- **System programming**
- **Commercial**
- **Engineering/scientific, etc.**

Less verbose than COBOL

Less English-like

HLL relationship between JCL and program files

- **Referring to physical files by a symbolic file name is used by all of the HLLs**
  - **even Assembler language**
- **Isolates your program from changes to data set name and data set location**
  - **data set name and location can change without recompiling program**
- **“Hard-coding” data set names or other such information in a program is not usually considered a good programming practice**
  - **Externalize these values from programs**

HLL relationship between JCL and program files

```
//MYJOB    JOB
//STEP1    EXEC CLG
...
   OPEN FILE=INPUT1
   OPEN FILE=OUTPUT1
   READ FILE=INPUT1
...
   WRITE FILE=OUTPUT1
...
   CLOSE FILE=INPUT1
   CLOSE FILE=OUTPUT1
/*
//GO.INPUT1  DD DSN=MY.INPUT,DISP=SHR
//GO.OUTPUT1 DD DSN=MY.OUTPUT,DISP=OLD
```

Using C/C++ on z/OS

C is a multi-purpose programming language

Suited for:

- **System-level code**
- **Text processing**
- **Graphics, etc.**

C language contains concise set of statements, with functionality added through its library

C is highly consistent across different platforms

Using Java on z/OS

- Java is an object-oriented programming language
- Enterprise COBOL and Enterprise PL/I provide interfaces to programs written in Java Language. Also, DB2 and IMS.
- Java is pervasive across the zSeries platform.
- Java Native Interface (JNI) allows your program to call programs written in other languages. The JNI is part of the Java Development Kit.

Using CLISTs on z/OS

CLIST (pronounced "see list") is short for command list, because the most basic CLISTs are lists of TSO/E commands

CLIST language is an interpreted language (that is, you don't have to compile and link-edit it)

CLISTs are easy to write and test

CLIST and REXX languages:

- **Two command languages available in TSO/E**

CLIST programming language is used for:

- **Performing routine tasks (entering TSO/E commands)**
- **Invoking other CLISTs**
- **Invoking applications written in other languages**
- **ISPF applications (displaying panels, controlling application flow)**
- **One-time quick solutions to problems**

Using REXX on z/OS

Restructured Extended Executor (REXX) language is a procedural language

REXX is an interpreted and compiled language

REXX is a more full-function language than CLIST

REXX can be used for:

- **Performing routine tasks (entering TSO/E commands)**
- **Invoking other REXX execs**
- **Invoking applications written in other languages**
- **ISPF applications (displaying panels, controlling application flow)**
- **One-time quick solutions to problems**
- **System programming**
- **Anywhere that we might use another HLL compiled language**

Using REXX on z/OS

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       LUTZ.CLIST(SYSPLEX) - 01.00                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> PAGE
****** ***************************** Top of Data ********************************
000001 /* REXX ------------------------------------------------------------*/
000002 /*                                                                  */
000003 /* function : checks whether z/OS UNIX works with SYSPLEX(YES)      */
000004 /*                                                                  */
000005 /*------------------------------------------------------------------*/
000006  numeric digits 12
000007  cvt=c2x(storage(10,4))                    /* get asdress of cvt    */
000008  ecvt=c2x(storage(d2x(x2d(cvt)+140),4))    /* get asdress of ecvt   */
000009  ocvt=c2x(storage(d2x(x2d(ecvt)+240),4))   /* get BPXYOEXT address  */
000010  oext=c2x(storage(d2x(x2d(ocvt)+12),4))
000011  sysplexflag=storage(d2x(x2d(oext)+8),1)
000012
000013  if bitand(sysplexflag ,'20'x ) = '20'x
000014    then say 'z/OS UNIX works with file sharing.'
000015    else say 'z/OS UNIX works without file sharing.'
000016
000017 exit
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
MA    A                                                        04/015
```

Compiled versus interpreted languages

Compiled versus interpreted:

- **A design-stage decision**
- **Performance is slower with interpreted languages**

Both compiled and interpreted languages have their strengths and weaknesses

No simple answer as to which is better -- it depends on the application. Within a single application, we might decide to use several programming languages.

Advantages of compiled languages

- Assembler, COBOL, PL/I, C/C++ are translated by running the source code through a compiler
- This results in very efficient code that can be executed any number of times
- Often, the overhead for the translation is incurred just once, when the source is compiled; thereafter, it need only be loaded and executed
- Compiled programs will be more efficient and performing
- Interpreted languages are often parsed, interpreted, and executed each time that the program is run, increasing the cost of running the program

Advantages of interpreted languages

- An interpretive language is relatively easy to code, test, and change
- Good for one-time solutions
- Good for developing application prototypes
- Ad hoc versus fixed requests
- Time saver for command strings that are entered repeatedly

Overview of Language Environment

Goals of application development today:

- Modularize and share code
- Develop applications on a Web-based front end

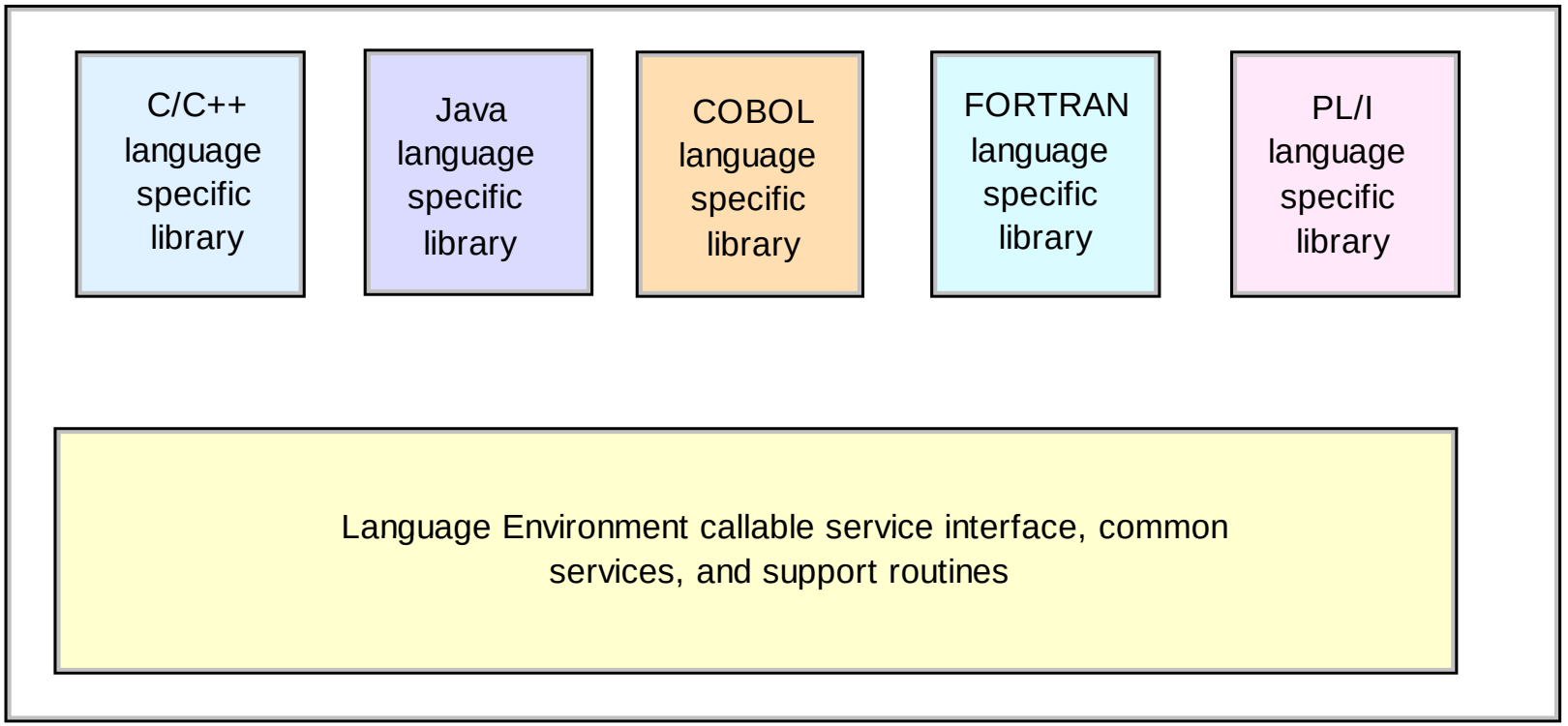
z/OS Language Environment product provides a common environment for all conforming high-level language (HLL) products:

- Establishes a common language development and execution environment for application programmers on z/OS
- Consolidates in a common runtime library, function previously provided in individual library products eliminating the need to maintain separate language libraries

Advantages of z/OS Language Environment

- Establishes a common run-time environment for all participating HLLs
- Combines essential run-time services, such as routines for run-time message handling, condition handling, and storage management
- All of these services are available through a set of interfaces that are consistent across programming languages
- You can use one run-time environment for your applications, regardless of the application's programming language or system resource needs
- Your program can seamlessly call one language from another, to exploit the functions and features in each language

Language Environment components



Summary

- The mainframe supports most programming languages in use today.
- Your choice of a programming language depends on several factors, including the requirements of the application and the installation's ability to maintain the application.
- Depending on the application requirements, you might use multiple languages or assembler subroutines for certain parts.
- Remember: When it is time to update the application, other people must be able to program these languages as well.
- Complexity in design must always be weighed against ease of maintenance.