COMP 3400 Mainframe Administration[1]

Christian Grothoff
christian@grothoff.org

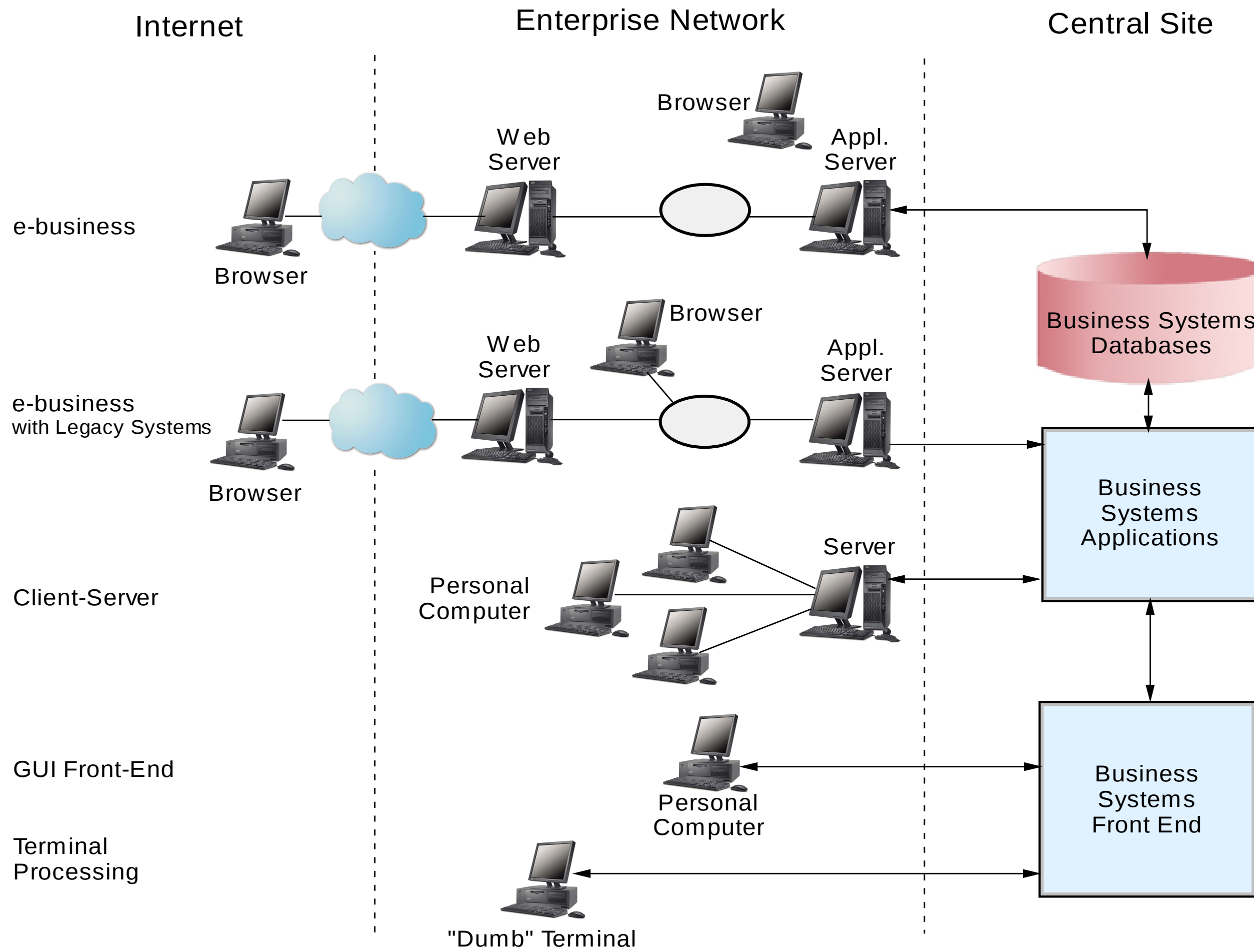
http://grothoff.org/christian/

[1]These slides are based in part on materials provided by IBM's Academic Initiative.

Today

- Application Programming on z/OS:
  - Software Engineering Review
  - Common languages on z/OS
  - Build process

Applications Are Everywhere



UNIVERSITY OF DENVER

Roles in Application Development

- Application Designer
- Application Programmer

UNIVERSITY OF DENVER

Application Designer

Determines the best programming solution for a business requirement using his understanding of:

- Business objectives of the company
- Company's hardware and software
- Other roles in the mainframe IT organization

$\Rightarrow$ Must have a global view of the entire project

UNIVERSITY OF DENVER

Types of Requirements

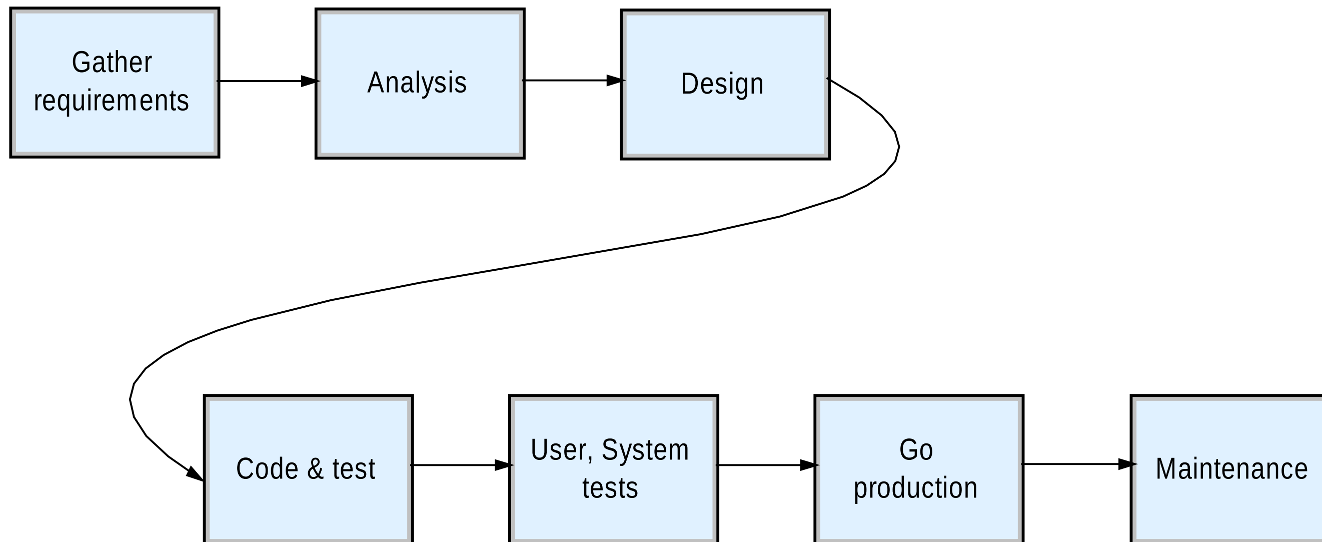
- Accessibility
- Interoperability
- Usability
- Managability
- Performance
- Serviceability
- Portability
- Availability
- Recoverability
- Fault-tolerance

UNIVERSITY OF DENVER

Key Decisions

- Online processing or batch processing?
- Storage model: Database, tape, flat file?
- Programming language: Java, Cobol, PL/1, Assembler?
- Platform: z/OS, Linux, UNIX, Windows?
- Hardware: type of server, capacity?
- Develop or purchase or both?

The Waterfall Model

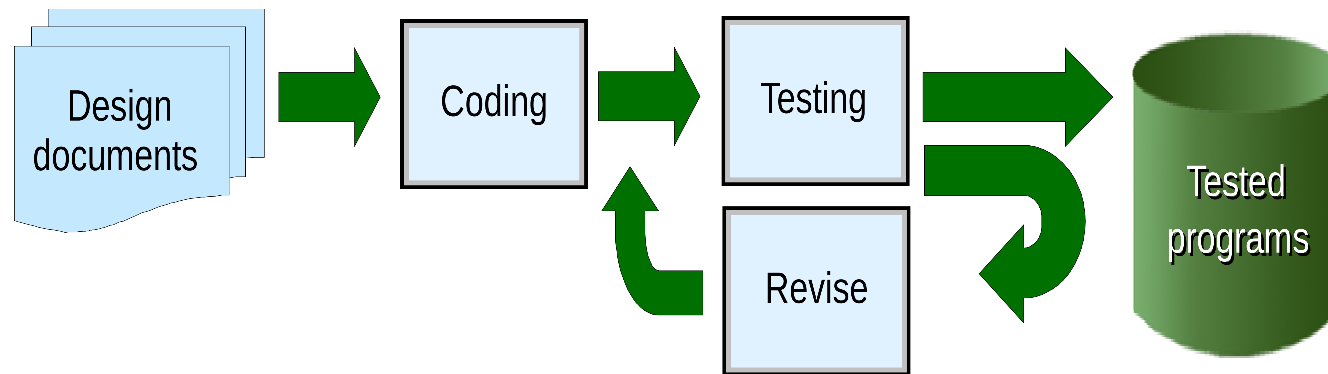


UNIVERSITY OF DENVER

Application Developer

- Builds, tests and delivers applications (for end users)
- Works from application designer's specifications
- Uses tools to change code, compile, build and test applications

The Grind (of the Application Developer)



UNIVERSITY OF DENVER

Traditional Application Development

1. Edit source and make modifications
2. Submit compile job to JCL (verifies syntax, compiles, links, runs tests)
3. View job output in SDSF, check for errors
4. If there are errors, go back to step 1
5. Save source code in repository

Programming tools

- ISPF Editor (or remote using WebSphere/Eclipse/Rational)
- Repository for source code (PDS, SCLM or other repository)
- Job monitoring and viewing (SDSF)
- Debugging tools (WebSphere, Rational, etc.)

IDEs accelerate development – also on the mainframe!

IDE features

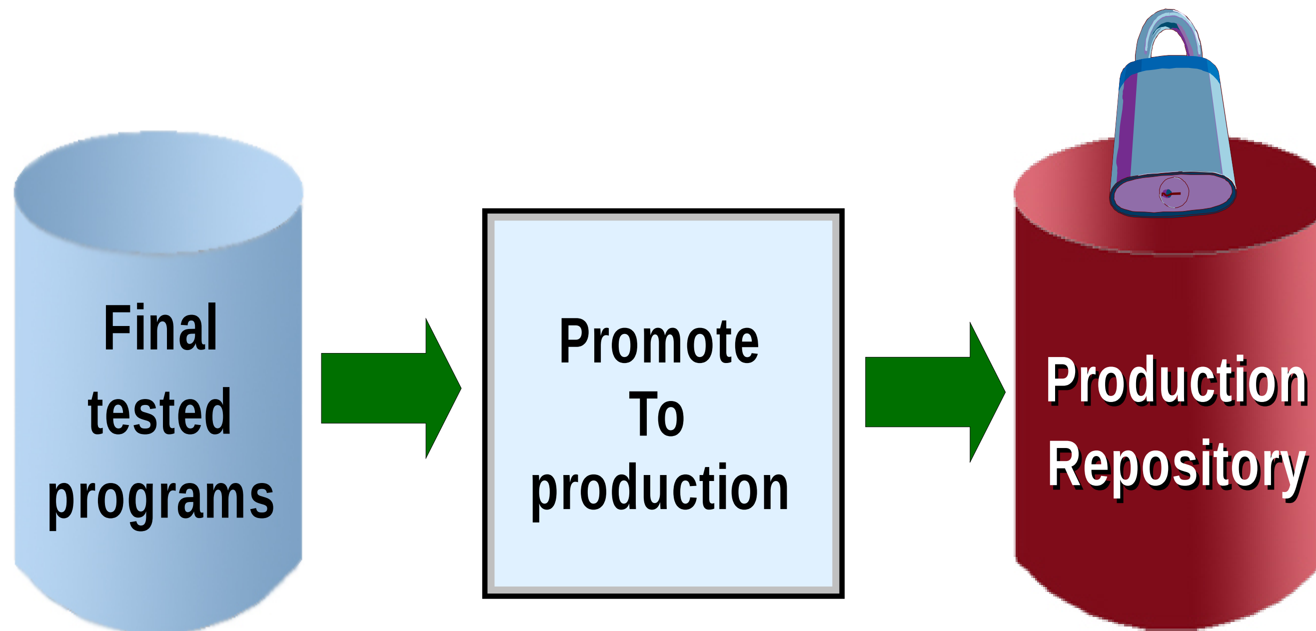
- Edit source on workstation
- Compile on workstation or on platform
- Remote debugging (program on mainframe, debugger on workstation)
- Support for many programming languages
- Integration with source code repositories

Testing

Before production use, code must pass many types of tests:

- Unit testing
- Functionality and acceptance testing
- Performance (stress) testing
- Integration testing

Moving to Production



UNIVERSITY OF DENVER

More than code

- Document operational procedures
- Provide training manuals (for users and administrators)
- Implement change control process
- Help handing code over to system operators
- Responsibility for maintenance may change to others or stay with developers

Maintenance

- Maintenance and enhancement is the primary role of most application programmers for mainframes
- Existing applications are often written in COBOL and PL/1
- New applications are in Java, COBOL and PL/1. Note that COBOL and PL/1 continue to be enhanced.

Types of Programming Languages

- Low-level languages: Assembler and C – direct correspondance to physical machine
- High-level languages: COBOL, PL/1, Java – require complex compilation and/or runtime environments
- Special-purpose languages: RPG, CSP, QMF, SQL – usually used for a specific subproblem
- Scripting languages: Perl, REXX – fast development, write-only code

UNIVERSITY OF DENVER

Choosing a programming language

- Performance requirements
- Interaction with code in other languages
- Knowledge of development team
- Scope of the project
- Tool support

Using Assembler on z/OS

Assembler is not usually used for application development, but for:

- Accessing system control blocks
- High performance subroutines where extreme execution efficiency is needed

UNIVERSITY OF DENVER

COBOL on z/OS

- Traditional language for business applications
- Can be integrated with web-oriented business processes
- Interoperability with Java
- Support for XML and Unicode

PL/1 on z/OS

- Used for system programming
- Used for engineering and scientific applications
- Less verbose and English-like compared to COBOL
- Can use symbolic file names just like COBOL[2]

[2] In fact, all z/OS languages support the use of symbolic file names, even Assembler.

UNIVERSITY OF DENVER

C/C++ on z/OS

- Used for system-level code, text processing, graphics, etc.
- z/OS is POSIX compliant!
- C language is standardized, but z/OS uses EBCDIC

⇒ C strings are in EBCDIC, not ASCII!

- The z/OS C compiler is not gcc!

⇒ No gcc extensions, for example, // is not a comment in C!

Java on z/OS

- Interfaces with COBOL and PL/1
- Interfaces with DB2 and IMS
- Support for JNI (interface with C/C++ and other languages)
- Good IDE support (Eclipse, WebSphere)

UNIVERSITY OF DENVER

CLISTs on z/OS

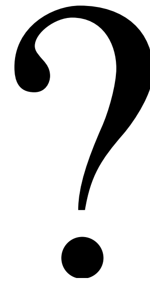
- Interpreted language
- Most basic CLISTs are lists of TSO/E commands
- Commonly used for writing ISPF panels
- Commonly used for one-time quick solutions for small problems

UNIVERSITY OF DENVER

REXX on z/OS

- Can be compiled or interpreted
- More expressive than CLIST
- Also available on other platforms (GNU/Linux, W32, z/VM)
- Used for routine tasks (submitting TSO/E commands), ISPF panels, system programming, etc.

Questions



Question!

Which programming language is the best to use?

z/OS Language Environment

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

- Common language development and execution environment for application programmers on z/OS
- Consolidates runtime library functions previously provided in individual library products

⇒ Similar to Microsoft's .NET framework, but without IR

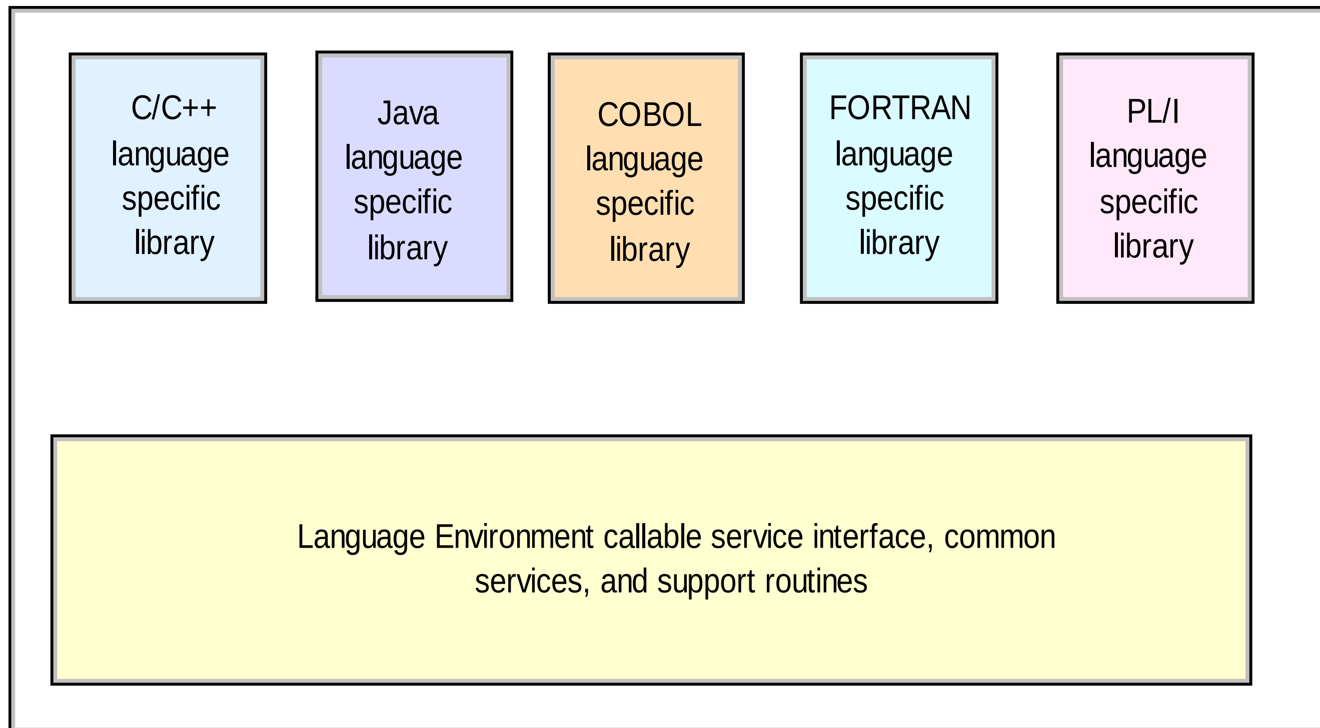
UNIVERSITY OF DENVER

Advantages of z/OS Language Environment

Having a common run-time environment for all paritipating HLLs...

- allows programs to seamlessly call one language from another
- avoids replication of essential run-time services such as message handling and storage management
- provides consistent interfaces across programming languages

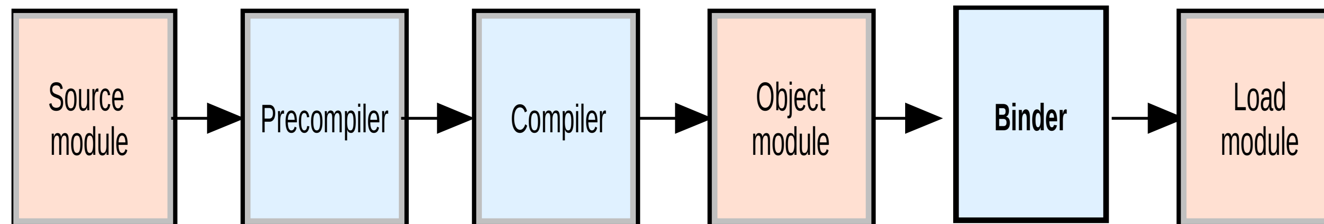
Language Environment Overview



Compiling on z/OS

- A source program is divided into logical units called *modules*
- Each *source module* is assembled or compiled by the respective language translator
- The compiler generates an *object module*
- Object modules are process by the binder to create a *load module* which can be executed

Compilation Overview



Separate Compilation and Relocation

When compilers translate source code into object code, they:

- Assign *relative* addresses to all instructions, data elements and labels, starting from zero
- Run-time addresses are in the form of a base address plus a *displacement* (to allow programs to be relocated)
- References to external programs or subroutines are left as *unresolved*

Relocatability

Even the final load modules are relocatable:

- The code can be located at any address in virtual storage[3]

$\Rightarrow$ An identical copy of a program can be loaded in many different address spaces at different starting addresses

$\Rightarrow$ Physical copies in memory can be reused!

[3]Within the confines of the residency mode

Source modules

- Source code written in the respective programming language
- Source programs are often stored in a PDS known as a *source library*
- A *copybook* is a source library containing prewritten text; it is used to copy text into a source program, at compile time.

$\Rightarrow$ *copybook* $\approx$ `/usr/include/`

UNIVERSITY OF DENVER

Object modules

- Collection of one or more compilation units
- Contains machine code (and program data) in relocateable format
- Contains *control dictionaries* to resolve cross-references between sections of different modules
- Not executable
- Multiple object modules can be stored in an *object library*

UNIVERSITY OF DENVER

Load modules

- Contains machine code (and program data) in relocateable format
- Contains *control dictionaries* to resolve cross-references between sections of different modules
- Can be loaded into virtual storage and relocated by the *program manager*

$\Rightarrow$ "executable" (with help of program manager)

UNIVERSITY OF DENVER

Load libraries

Load libraries contain programs ready to be executed (usually load modules processed by the binder or linkage editor).

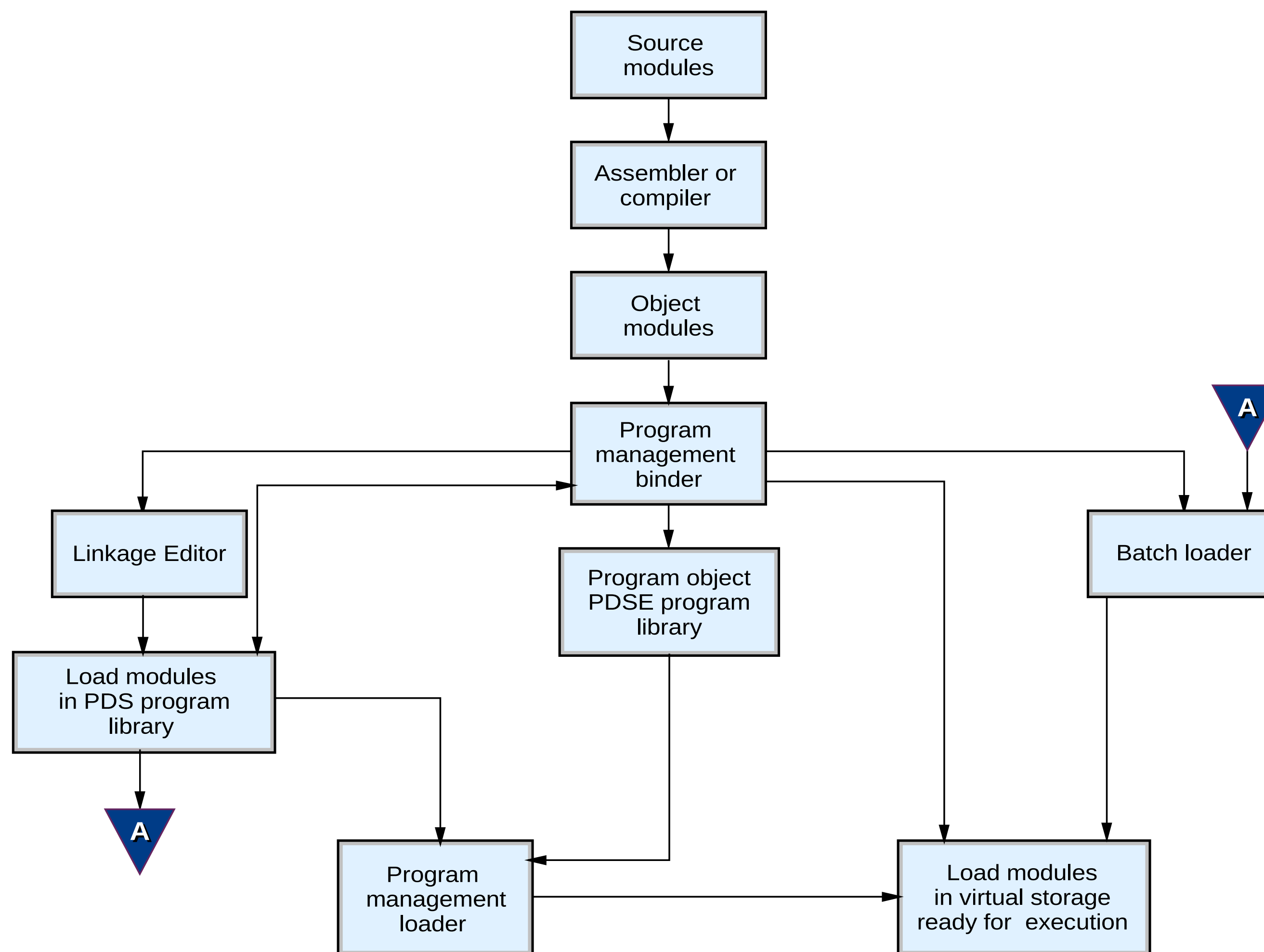
We distinguish:

- System libraries – unless a job specifies a private library, the system libraries are searched when a job specifies `//step EXEC PGM=program-name`
- Private libraries – user-written programs, searched only when JCL specifies DD statement defining `JOBLIB` or `STEPLIB`
- Temporary libraries – see IGYWCLG (later)

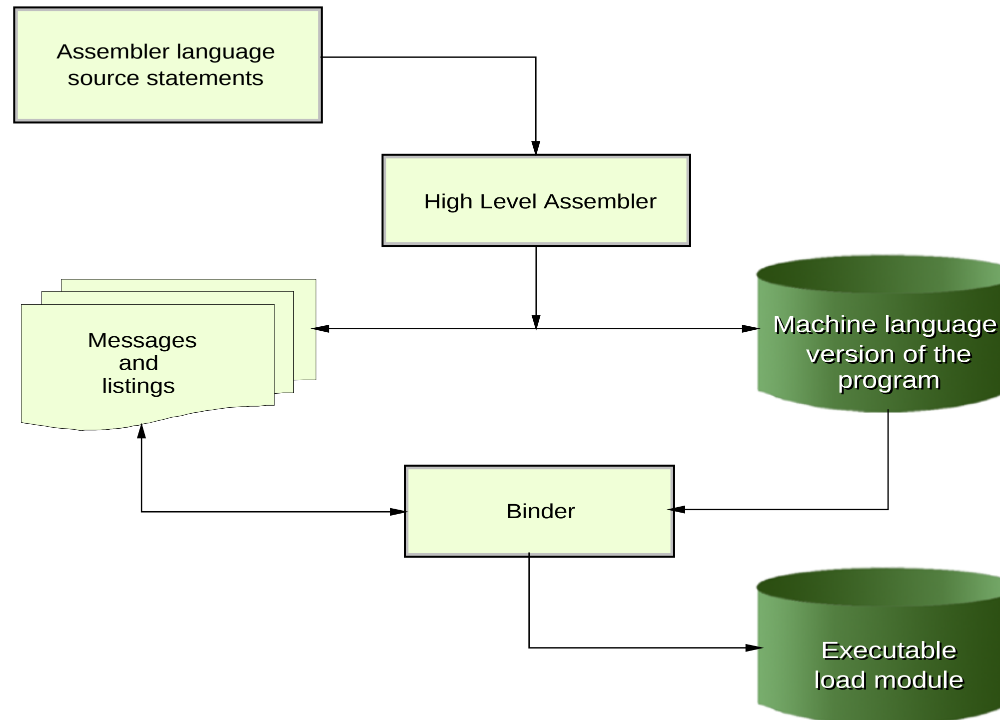
Binder

- Assigns virtual storage addresses to sections of the module and resolves references between modules
- Can process traditional data sets (PDS, PDSE) and z/OS UNIX files
- An older, more restricted version of the binder was called the *linkage editor*
- The *batch loader* and the *program management loader* can also be used to create a load module to execute only (but not to store the result in a library)
- The batch loader is replaced by the binder in later releases of z/OS

Compilation Overview II

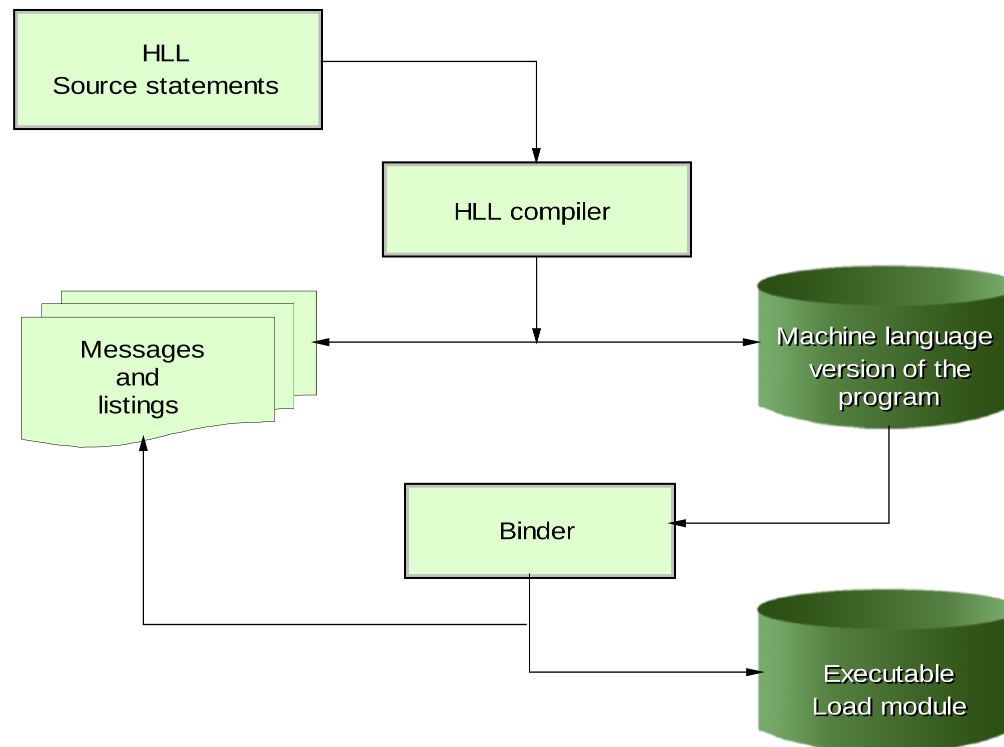


Translating Assembler Code



UNIVERSITY OF DENVER

Translating COBOL Code



University of Denver

Ways for compiling COBOL

- Use a batch job (JCL), often with cataloged procedures
- Use TSO/E commands in CLISTs or ISPF panels
- Use the `cob2` command in the z/OS UNIX shell

UNIVERSITY OF DENVER

IGYWCLG

IGYWCLG is a three-step cataloged procedure to:

- Compile (step is called "COBOL")
- Link-Edit and
- Run (step is called "GO")

a COBOL application. You must supply:

```
//COBOL.SYSIN DD  *
```

UNIVERSITY OF DENVER

Example: Compiling COBOL

```
//MYJOB   JOB
//STEP1   EXEC IGYWCLG
//COBOL.SYSIN DD  *
  ...
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT INPUT ASSIGN TO INPUT1 ...
    SELECT DISKOUT ASSIGN TO OUTPUT1 ...
  ...
/*
//GO.INPUT1 DD DSN=MY.INPUT,DISP=SHR
//GO.OUTPUT1 DD DSN=MY.OUTPUT,DISP=OLD
```

Example: Compiling PL/1

```
//MYJOB    JOB
//STEP1    EXEC IBMZCLG
//PLKED.SYSIN DD  *
   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
```

Questions

UNIVERSITY OF DENVER

Tasks!

- Find IGYWCLG on your z/OS system!
- Do the exercises in section 10.9