The Once and Future COBOL

James K. Lowden
Cobolworx

Symas Corporation

foss-north 2026

Monday 27 April 2026, 10:00 am

The COBOL Origin Story

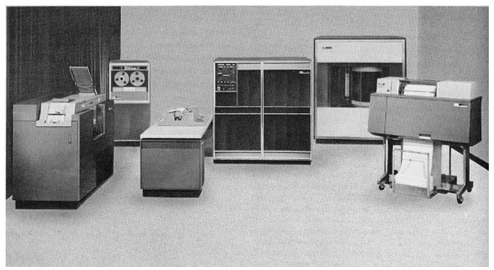
Cool COBOL Features

GCC COBOL

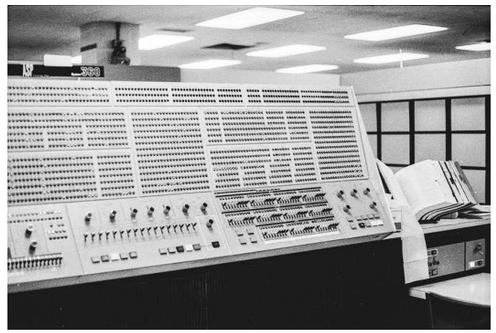
One Compiler to Rule Them All

The COBOL Story

How did we get here?



1960, IBM 1401
16 KB memory
COBOL
FORTRAN



1970, IBM 360
1 MB memory
COBOL
FORTRAN
PL/1

1980, IBM 370
32 MB memory
COBOL
FORTRAN
PL/1
IBM rank on S&P 500: #1

The COBOL Story

1 Consistent Interface

Today

- Version Control
- Networks
- Concurrency
- Security
- Virtualization
- Databases
- Libraries
- Frameworks

Then

COBOL

The COBOL Story

How did we get here? (1960-1990)

Unique moment in the history of computing.

- IBM was 3/4 of computer industry for over 3 decades
- 95% of programmers employed by non-tech industry
- Automation offered historic productivity improvement
- 30 years $\times$ 0.75 $\times$ 0.95 $\times$ programmers
  = *billions* of lines of COBOL

Not only IBM

- Most vendors offered a COBOL compiler
- Stonebraker remembers early Ingres turned down by University of Arizona for lack of COBOL interface (on Unix!)

They Weren't Stupid

Why did we get here?

Highest level compiled language, still today

COBOL programmer deals only in the business-problem domain

- Read input
- Process
- Write output
- Profit! (No joke)

6 Killer COBOL Features

Why did we get here?

1. Record-oriented
2. Definite Data
3. Nothing External
   - No OS API
   - No event loop
4. Error Handling
5. Exception Conditions
6. Exact Computation

Killer COBOL Feature #1

Record-oriented

NIST NC101A.CBL

```
146 01   MULTIPLY-DATA.
147       02 MULT1 PICTURE IS 999V99 VALUE IS 80.12.
148       02 MULT2 PICTURE IS 999V999.
149       02 MULT3 PICTURE IS $$99.99.
150       02 MULT4 PICTURE IS S99 VALUE IS -56.
151       02 MULT5 PICTURE IS 9 VALUE IS 4.
152       02 MULT6 PICTURE IS 99 VALUE IS 20.

276 01   CCVS-E-2.
277       02 FILLER            PIC X(31)  VALUE SPACE.
278       02 FILLER            PIC X(21)  VALUE SPACE.
279       02 CCVS-E-2-2.
280           03 ERROR-TOTAL  PIC XXX    VALUE SPACE.
281           03 FILLER       PIC X      VALUE SPACE.
282           03 ENDER-DESC   PIC X(44)  VALUE "ERRORS ENCOUNTERED".
```

Killer COBOL Feature #1

Record-oriented (2)

Exact data layout

- Defined in-memory and on-disk representation
- Exact numerical representation
- Formatted money

Great names

- Convenient hyphenation
- Duplicate names use space but cannot be referenced for computation
- Unique names may be used without qualification in hierarchy

All valid:

```
ENDER-DESC
ENDER-DESC OF CCVS-E-2-2
ENDER-DESC OF CCVS-E-2
ENDER-DESC OF CCVS-E-2-2 OF CCVS-E-2
```

Killer COBOL Feature #1

Record-oriented (3)

Great names

The Hyphenated name, and the fate of subtraction

Just use a little whitespace

Name

- `x-y`

Subtraction

- `x - y`
- `x - 1`

Invalid

- `x -1` (`-1` is a literal)
- `-y` (a name may not start with a hyphen)

Killer COBOL Feature #1

Record-oriented (4)

Great I/O

NIST IX101A.CBL

```
510 INPUT-OUTPUT SECTION.
511 FILE-CONTROL.
512      SELECT    IX-FS1 ASSIGN TO "IX101-2A.indexed"
513      ORGANIZATION IS INDEXED
514      RECORD KEY IS IX-FS1-KEY
515      ACCESS MODE IS SEQUENTIAL.
516 DATA DIVISION.
517 FILE SECTION.
518 FD   IX-FS1
519      BLOCK CONTAINS 1 RECORDS
520      RECORD CONTAINS 240 CHARACTERS.
521 01   IX-FS1R1-F-G-240.
522      03 IX-FS1-WRK-120 PIC X(120).
523      03 IX-FS1-GRP-120.
524      05 FILLER    PIC    X(8).
525      05 IX-FS1-KEY   PIC X(29).
526      05 FILLER PIC X(83).
```

Killer COBOL Feature #1

Record-oriented (5)

Great I/O

NIST IX101A.CBL

```
461      MOVE     FILE-RECORD-INFO (1) TO IX-FS1R1-F-G-240.
462      WRITE    IX-FS1R1-F-G-240
463                 INVALID KEY GO TO WRITE-FAIL-GF-01.
464      IF       XRECORD-NUMBER (1) EQUAL TO 500
465               GO TO WRITE-PASS-GF-01.
466      ADD       000001 TO XRECORD-NUMBER (1).
467      GO        TO WRITE-TEST-GF-01.
468 WRITE-FAIL-GF-01.
469      MOVE      "IX-41 4.9.2          " TO RE-MARK.
470      PERFORM FAIL.
471      GO TO WRITE-WRITE-GF-01.
472 WRITE-PASS-GF-01.
473      PERFORM PASS.
474 WRITE-WRITE-GF-01.
475      MOVE      "WRITE-TEST-GF-01" TO PAR-NAME
476      MOVE      "FILE CREATED, LFILE "  TO COMPUTED-A.
477      MOVE     XRECORD-NUMBER (1) TO CORRECT-18V0.
478      PERFORM   PRINT-DETAIL.
479      CLOSE     IX-FS1.
```

Killer COBOL Feature #2

Definite Data

All data statically defined (usually)

- no pointers, no dynamic allocation, no stack

File type and structure (and mode of access) defined to compiler

- Sequential, Random (by record number), or Indexed (by key)
- Record layout and size defined to compiler

Type, storage, and format of each field defined to compiler

- Decimal arithmetic on integers with implied decimal point (that displays!)
- Computation on character digits, or half-character (packed decimal)
- Strings fixed size, blank padded

Killer COBOL Feature #2

DISPLAY and MOVE

`DISPLAY` just ... displays

- No conversion, no formatting characters
- No Mojibake
- No buffer overruns

`MOVE` converts between types

- Any to any: integer, fixed point, floating point, character
- No **strcpy**(3), no **sprintf**(3)
- Copy whole arrays, or substrings, with a single statement
- Convert between character encodings

Killer COBOL Feature #2

MOVE CORRESPONDING

`MOVE CORRESPONDING` copies like-named members

```
186 01   CORR-DATA-1.
187      03 XYZ-1                    PICTURE IS 99      VALUE IS ZERO.
188      03 XYZ-2                    PICTURE IS 99      VALUE IS ZERO.
189      03 XYZ-3                    PICTURE IS 99      VALUE IS ZERO.
190      03 XYZ-4                    PICTURE IS 99      VALUE IS ZERO.
191      03 XYZ-5                    PICTURE IS 99      VALUE IS ZERO.
192      03 XYZ-6                    PICTURE IS 99      VALUE IS ZERO.
...
200 01   CORR-DATA-3.
201      03 XYZ-4                    PICTURE IS 99      VALUE IS ZERO.
202      03 XYZ-3                    PICTURE IS 99      VALUE IS ZERO.
203      03 XYZ-6                    PICTURE IS 99      VALUE IS ZERO.
204      03 XYZ-5                    PICTURE IS 99      VALUE IS ZERO.
205      03 XYZ-2                    PICTURE IS 99      VALUE IS ZERO.
206      03 XYZ-1                    PICTURE IS 99      VALUE IS ZERO.
...
982 MOVE CORRESPONDING CORR-DATA-1 TO CORR-DATA-3.
```

Killer COBOL Feature #3

Nothing External

All I/O mediated through the compiler

- `OPEN` cannot e.g. open a sequential file for random access
- `READ` and `WRITE` typically do not specify record size

Terminal interaction mediated by CICS

- Basic Mapping Support (BMS) defines
  - UI form for terminal
  - COBOL record structure
- COBOL `READ MAP` and `WRITE MAP` work with *records*
- No parsing, no formatting, no surprises, no errors

Killer COBOL Feature #4

Error Handling

Most COBOL statements have a *conditional form*

- Each statement may raise one of a class of errors, known as an “Exception Condition”
- NIST NC101A.CBL:

```
485 MPY-TEST-F1-3-0.
486      MULTIPLY MULT5 BY MULT5 ON SIZE ERROR
487                   MOVE "K" TO XRAY.
```

Conditional code under `ON SIZE ERROR` handles *statement-specific* errors

Compare e.g. Python

```
1 try:
2     foo()
3 except mumble as oops:
4     pass
```

Killer COBOL Feature #4

Standard Declarative Error Handling

"Declarative" procedures

- handle Exception Conditions centrally
- if not handled by the statement

NIST IX103A.CBL

```
317 PROCEDURE DIVISION.
318 DECLARATIVES.
319 USE-IX103-TEST   SECTION.
320      USE    AFTER   STANDARD   EXCEPTION PROCEDURE
321                 IX-FS1.
322 USE-PAR-001.
323      ADD    1    TO WRK-CS-09V00-009.
324 USE-PAR-EXIT.
325      EXIT.
326 END DECLARATIVES.
```

Killer COBOL Feature #4

Standard Declarative Error Handling (2)

Declarative for any I/O error

```
USE [GLOBAL] AFTER STANDARD { EXCEPTION | ERROR } PROCEDURE ON { file-name-1 ... | INPUT | OUTPUT | I-O | EXTEND }
```

Declarative for any named exception

```
USE AFTER { EXCEPTION CONDITION | EC } { exception-name-1 | exception-name-2 {FILE file-name-2} ... } ...
```

Killer COBOL Feature #5

100 Exception Conditions

Example: EC-SIZE

- At runtime, computation may produce a result that cannot be represented by the target field.
- Instead of *undefined behavior*, COBOL defines it

In gcc	Exception-name	Cat	Description
	EC-SIZE		Size error exception
	EC-SIZE-ADDRESS	Fatal	Invalid pointer arithmetic
☞	EC-SIZE-EXPONENTIATION	Fatal	Exponentiation rules violated
	EC-SIZE-IMP	Imp	Implementor-defined size error exception
	EC-SIZE-OVERFLOW	Fatal	Arithmetic overflow in calculation
☞	EC-SIZE-TRUNCATION	Fatal	Significant digits truncated in store
	EC-SIZE-UNDERFLOW	Fatal	Floating-point underflow
☞	EC-SIZE-ZERO-DIVIDE	Fatal	Division by zero

Of 6 defined size errors, GCC COBOL implements 3

Killer COBOL Feature #6

Exact Computation

Business, not science

- Integer quantities with implied decimal point
- 7 Rounding rules

DEFAULT ROUNDED MODE IS
- AWAY-FROM-ZERO
- NEAREST-AWAY-FROM-ZERO
- NEAREST-EVEN
- NEAREST-TOWARD-ZERO
- PROHIBITED
- TOWARD-GREATER
- TOWARD-LESSER
- TRUNCATION

6 Killer COBOL Features

Let's review

1. Record-oriented
2. Definite Data
3. Business Focus, Nothing External
4. Error Handling
5. Exception Conditions
6. Exact Computation

So, what's the problem?

The COBOL Problem

The mainframe is a weird place

Supporting environment unlike POSIX

- No shell, no hierarchical file system, no process/file permissions
- JCL, RACF, CICS, IMS, SYNCSORT

☜ Adaptation to POSIX is like *drawing a horse*

Recompilation is only Step 1

GCC COBOL

The origin story

Symas

- OpenLDAP support
- Refugees from big companies

A COBOL compiler?

- Is it possible?
    - Seems that way
- Is it profitable?
    - It’s possible

GCC COBOL

Technical Strategerizing

Technology survey: Where to start?

- GnuCOBOL (transpiler)
- GCC
- Clang
- Abandonware
- Roll your own

GCC COBOL

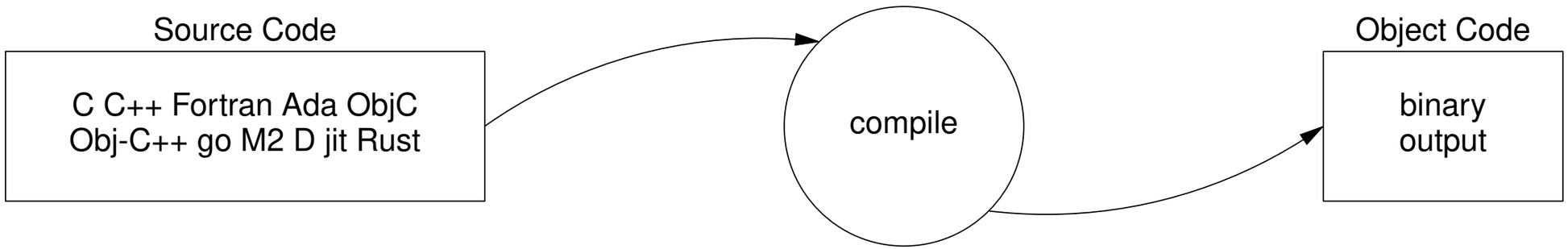
Why GCC?

GCC

- Only compiler with many languages
- Established path to add COBOL
- Good culture: Low-temperature, high quality mailing list
- Built-in support for portability and debugging

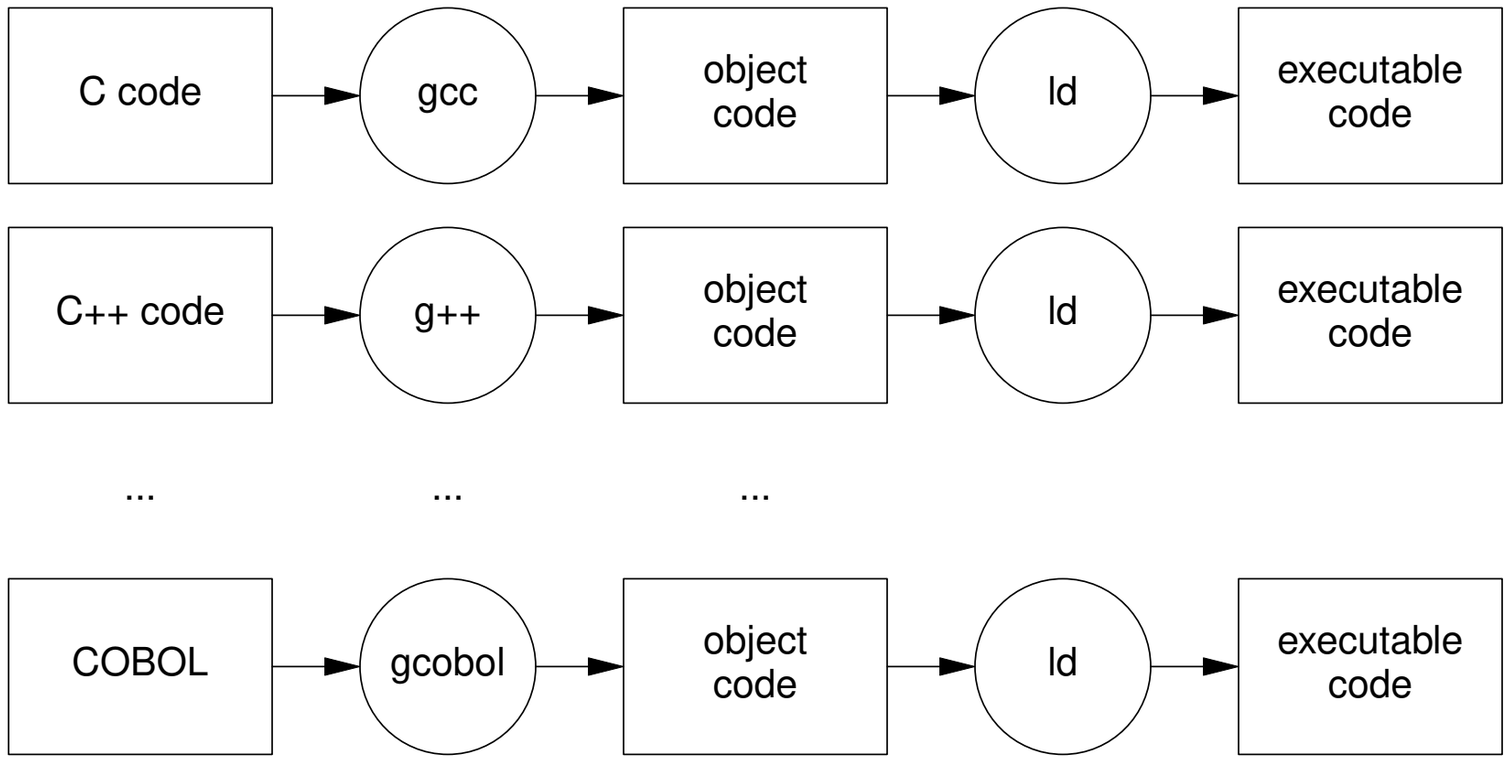
GCC

Overview



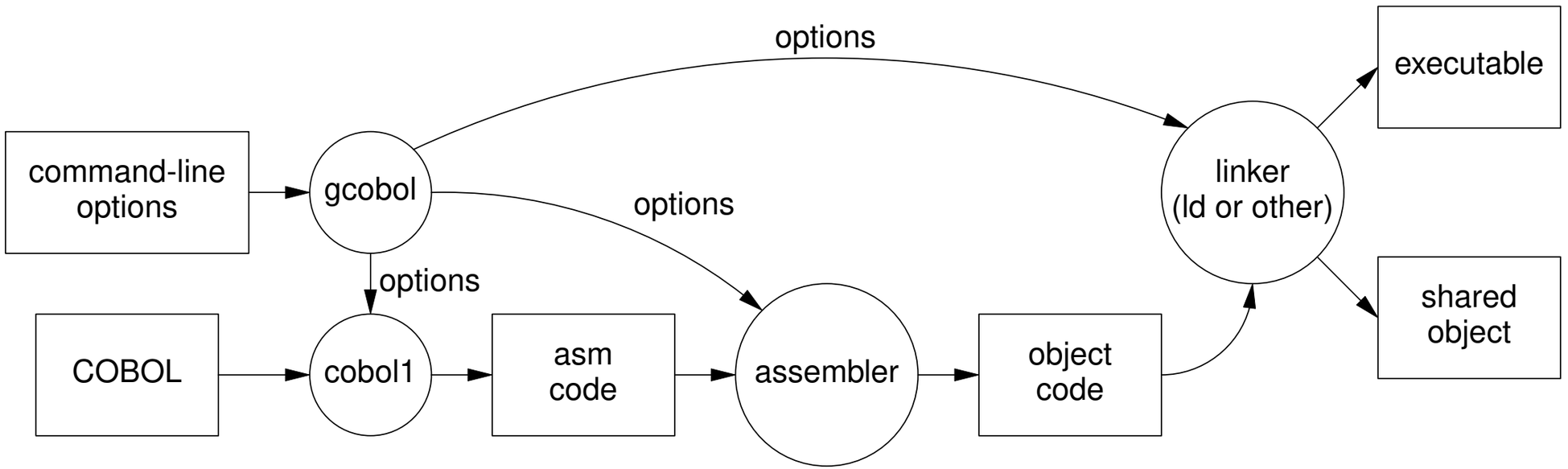
GCC

A Compiler Collective, er, Collection



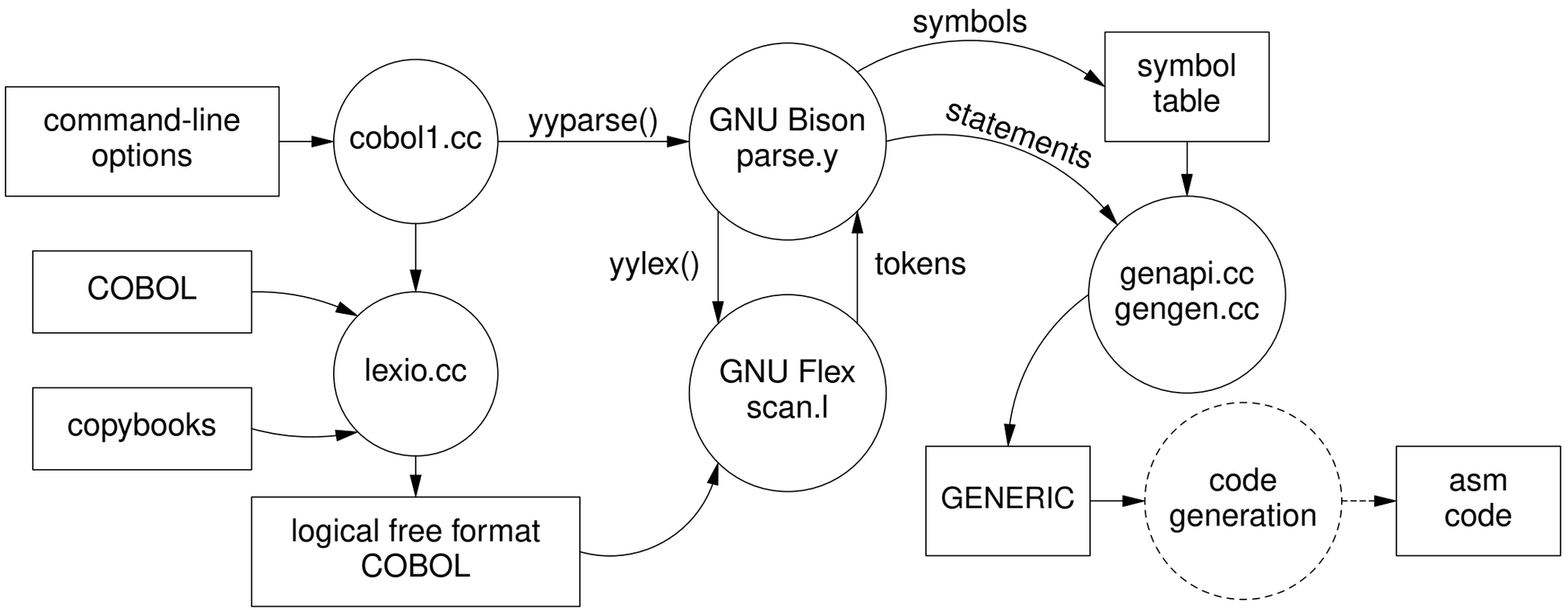
gcobol

compiler driver and compiler



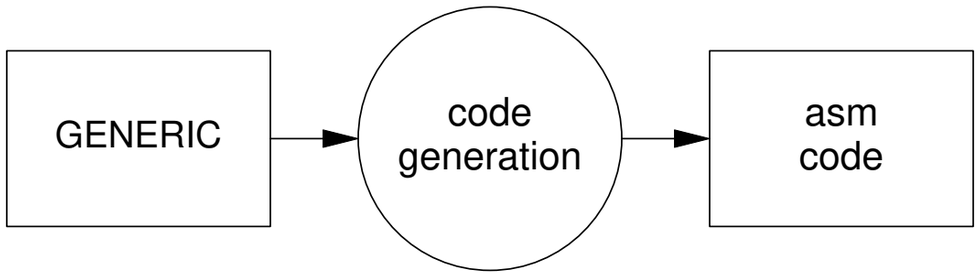
cobol1

The compiler



GCC Internals

Front-end perspective



Alternative view

The Present COBOL

The tale thus far

GnuCOBOL

- Mature, in commercial use
- Emulates many COBOL dialects
- Transpiler: Converts COBOL to C (then compiles C with GCC)
  - Efficiency depends on translation
  - Debugging requires mediation

GCC COBOL

- Overnight success, after 4 years
- Emulates 2 COBOL dialects, partially
- Compiles to an executable, of course
  - Efficiency depends on us
  - Native support for gdb

The Future COBOL

The future is unwritten

GCC COBOL: User feedback drives development

- Defined encoding choices for string variables and files
- XML support
- Runtime tracing
- More flexible support for warnings, errors, and dialects
- Portable to more architectures

The Once and Future COBOL

The billion-line secret

GCC 15 first free *direct-to-executable* COBOL compiler

More tools and support needed

2026 Prediction: 1st production use of GCC COBOL

The best way to predict the future is to invent it.

—Alan Kay