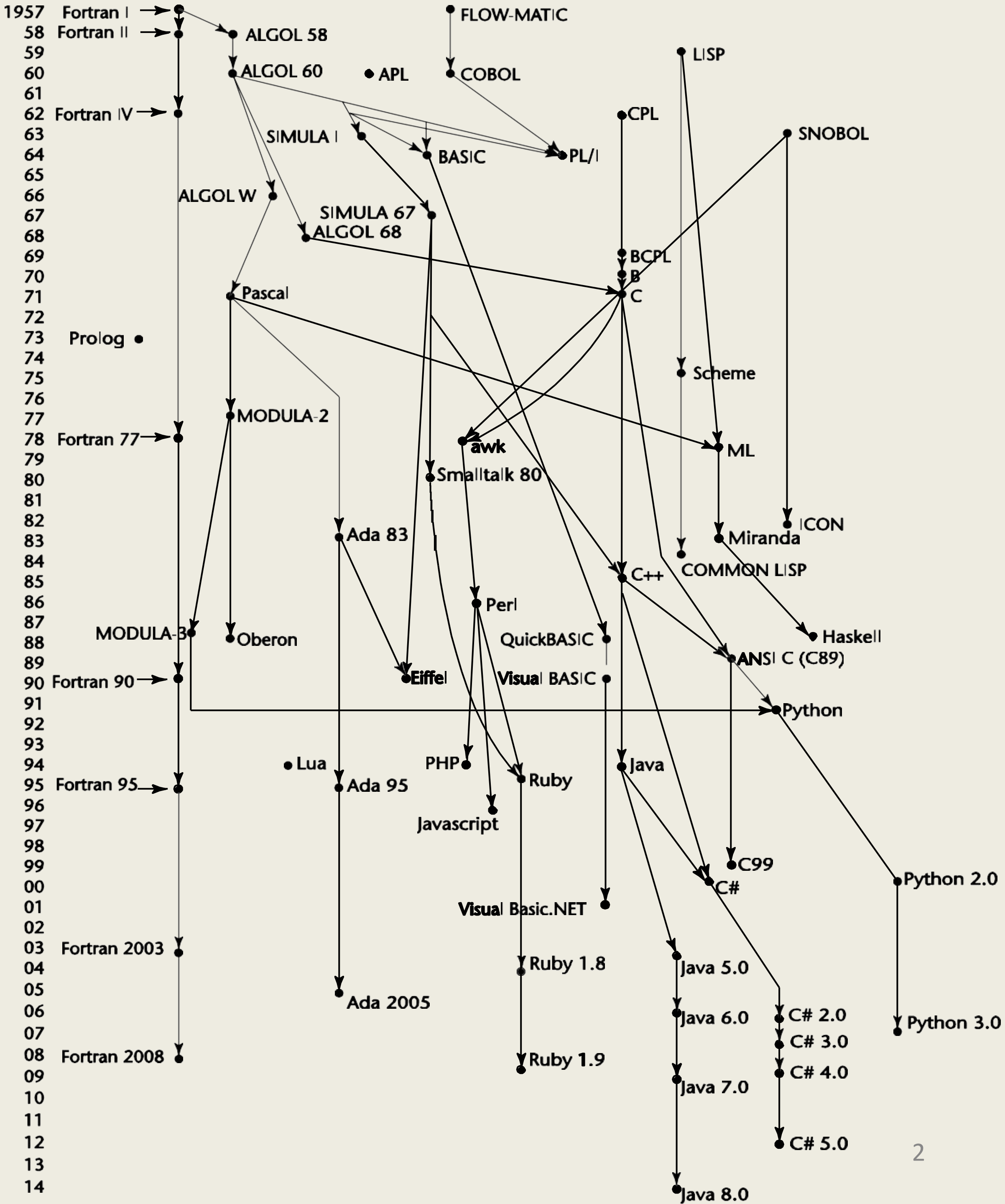
The Evolution of Programming Languages

Programming Language Genealogy



Zuse'sPlankalkül

- Designed in 1945, but not published until 1972
- Invented by a German scientist
- Never implemented
- Advanced data structures
  - floating point, arrays, records
- Invariants

Plankalkül Syntax

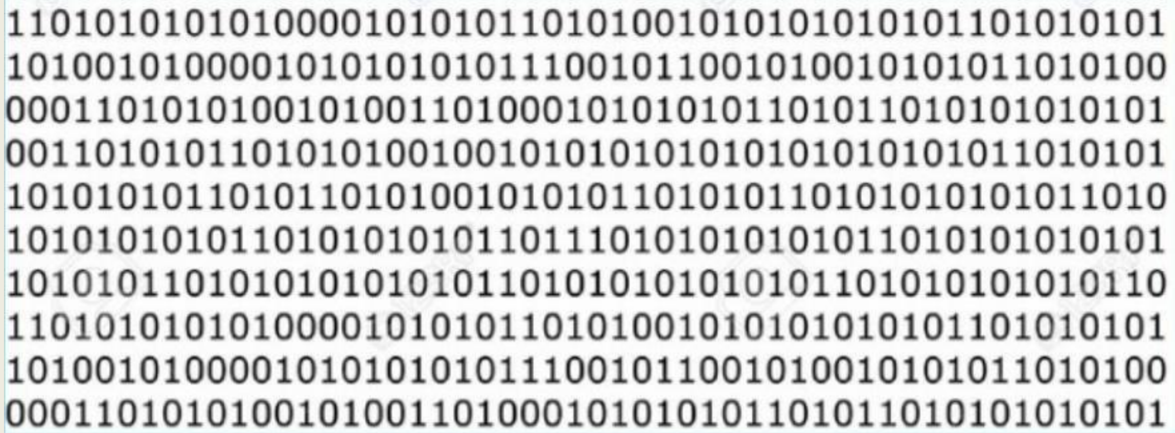
- An assignment statement to assign the expression *A*[4] + 1 to *A*[5]

```
    |  A + 1 => A
  V |   4        5              (subscripts)
  S |  1.n     1.n               (data types)
```

Minimal Hardware Programming: Pseudocodes

- Pseudocodes were developed and used in the late 1940s and early 1950s
- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies--no indexing or floating point

Machine Code



```
1101010101010000101010110101001010101010101101010101
1010010100001010101010111001011001010010101011010100
0001101010100101001101000101010101101011010101010101
0011010101101010100100101010101010101010101011010101
1010101011010110101001010101101010110101010101011010
1010101010110101010101101110101010101011010101010101
1010101101010101010101101010101010101101010101010110
1101010101010000101010110101001010101010101101010101
1010010100001010101010111001011001010010101011010100
0001101010100101001101000101010101101011010101010101
```

- Any binary instruction which the computer's CPU will read and execute
  - *e.g.*, 10001000 01010111 11000101 11110001 10100001 00010101
- Each instruction performs a very specific task, such as loading a value into a register, or adding two binary numbers together

Short Code: The First Pseudocode

- Short Code developed by Mauchly in 1949 for BINAC computers
  - Expressions were coded, left to right
  - Example of operations:

01	-	06	abs value	1n	(n+2)nd power
02	)	07	+	2n	(n+2)nd root
03	=	08	pause	4n	if <= n
04	/	09	(	58	print and tab

- Variables were named with byte-pair codes
  - E.g., X0 = SQRT(ABS(Y0))
  - 00 X0 03 20 06 Y0
  - 00 was used as padding to fill the word

Short Code

- Implementation: pure interpretation
- (called atomic programming)
- High execution time
- 50 times slower than machine code
- Simple for programming

IBM 704 and Fortran

- Fortran 0: 1954 - not implemented
- Fortran I: 1957
  - Designed for the new IBM 704, which had index registers and floating point hardware
    - This led to the idea of compiled programming languages, because there was no place to hide the cost of interpretation (no floating-point software)
  - Includes
    - Formatted I/O, variable names of up to six characters, user-defined subroutines, three-way selection statement (arithmetic IF), do-loop
- Environment of development
  - Computers were small and unreliable
  - Applications were scientific
  - No programming methodology or tools
  - Machine efficiency was the most important concern

–Limitations

- No separate compilation
  - Subroutines could not be separately compiled
- No data typing statements
  - Variables whose names began with I, J , K, L, M, and N were implicitly integer type, and all others were implicitly floating-point.
- Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of 704

Fortran

- Fortran II: 1958
  - Independent compilation
  - Fixed the bugs
- Fortran IV: 1960-62 (Fortran 66)
  - Explicit type declarations
  - Logical if-construct
  - The capability of passing subprograms as parameters

Fortran

- Fortran 77: 1978
  - Character string handling
  - Logical loop control statement
  - IF-THEN-ELSE statement
- Fortran 90
  - Modules, dynamic arrays, pointers, recursion, CASE statement, parameter type checking

Fortran

- Fortran 95
  - relatively minor additions, plus some deletions
- Fortran 2003
  - support for OOP, procedure pointers, interoperability with C
- Fortran 2008
  - blocks for local scopes, co-arrays (for parallel execution), Do Concurrent

Fortran Evaluation

- Highly optimizing compilers (all versions before 90)
- Types and storage of all variables are fixed before runtime
- Dramatically changed forever the way computers are used (first used high-level language)

The First Step Toward Sophistication: ALGOL 60

- Environment of development
  - FORTRAN had (barely) arrived for IBM 70x
  - Many other languages were being developed, all for specific machines
  - No portable language; all were machine-dependent
  - No universal language for communicating algorithms
- ALGOL 60 was the result of efforts to design a universal language

Early Design Process

- ACM and GAMM met for four days for design (May 27 to June 1, 1958)

- Goals of the language
  - Close to mathematical notation
  - Good for describing algorithms
  - Must be translatable to machine code

ALGOL 58

- Concept of type was formalized
- Names could be any length
- Arrays could have any number of subscripts
- Parameters were separated by mode (in & out)
- Subscripts were placed in brackets
- Compound statements (**begin** ... **end**)
- Semicolon as a statement separator
- Assignment operator was :=
- **if** had an **else-if** clause
- No I/O - “would make it machine dependent”

ALGOL 58 Implementation

- Not meant to be implemented, but variations of it were (MAD, JOVIAL)
- Although IBM was initially enthusiastic, all support was dropped by mid 1959

ALGOL 60 Overview

- Modified ALGOL 58 at 6-day meeting in Paris
- New features
  - Block structure (local scope)
  - Two parameter passing methods
  - Subprogram recursion
  - Stack-dynamic arrays
  - Still no I/O and no string handling

ALGOL 60 Evaluation

- Successes
  - It was the standard way to publish algorithms for over 20 years
  - All subsequent imperative languages are based on it
  - First machine-independent language
  - First language whose syntax was formally defined (BNF: Backus-Naur form)

ALGOL 60 Evaluation (continued)

- Failure
  - Never widely used, especially in U.S.
- Reasons
  - Lack of I/O and the character set made programs non-portable
  - Too flexible--hard to implement
  - Entrenchment of Fortran
  - Formal syntax description
  - Lack of support from IBM

ALGOL 68

- From the continued development of ALGOL 60 but not a superset of that language
- Source of several new ideas (even though the language itself never achieved widespread use)
- Design is based on the concept of orthogonality
  - A few basic concepts, plus a few combining mechanisms

ALGOL 68 Evaluation

- Contributions
  - User-defined data structures
  - Reference types
  - Dynamic arrays (called flex arrays)

- Comments
  - Less usage than ALGOL 60
  - Had strong influence on subsequent languages, especially Pascal, C, and Ada

Pascal - 1971

- Developed by Wirth (a former member of the ALGOL 68 committee)
- Designed for teaching structured programming
- Small, simple, nothing really new
- Largest impact was on teaching programming
  - From mid-1970s until the late 1990s, it was the most widely used language for teaching programming

C - 1972

- Designed for system programming (at Bell Labs by Dennis Richie)
- Evolved primarily from BCLP and B, but also ALGOL 68
- Powerful set of operators, but poor type checking
- Initially spread through UNIX
- Though designed as a system language, it has been used in many application areas

History's Largest Design Effort: **Ada**

- Huge design effort, involving hundreds of people, much money, and about eight years
- Sequence of requirements document for the new language (1975-1978)
  - (Strawman, Woodenman, Tinman, Ironman, Steelman)
  - Four finalist language design proposals were chosen, all of which were based on Pascal
  - The Cii Honeywell/Bull language design proposal was selected

Ada Evaluation

- Named Ada after Augusta Ada Byron, the first programmer
- Contributions
  - Packages - support for data abstraction
  - Exception handling
  - Generic program units
  - Concurrency - through the tasking model

Ada Evaluation

- Comments
  - Competitive design
  - Included all concepts that was then known about software engineering and language design
  - First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

- Ada 95 (began in 1988)
  - Support for OOP through type derivation
  - Better control mechanisms for shared data
  - New concurrency features
  - More flexible libraries
- Ada 2005
  - Interfaces and synchronizing interfaces

Ada

- Popularity suffered because the DoD no longer requires its use but also because of popularity of C++

Object-Oriented Programming: **Smalltalk**

- Developed at Xerox PARC, initially by Alan Kay, later by Adele Goldberg
- First full implementation of an object-oriented language (data abstraction, inheritance, and dynamic binding)
- Pioneered the graphical user interface design
- Promoted OOP

Combining Imperative and Object- Oriented Programming: C++

- Developed at Bell Labs by Stroustrup in 1980
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67
- A large and complex language, in part because it supports both procedural and OO programming

C++

- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November 1997
- Microsoft's version: MC++
  - Properties, delegates, interfaces, no multiple inheritance

A Related OOP Language

- Objective-C (designed by Brad Cox – early 1980s)
  - C plus support for OOP based on Smalltalk
  - Uses Smalltalk's method calling syntax
  - Used by Apple for system programs

An Imperative-Based Object-Oriented Language: Java

- Developed at Sun in the early 1990s
  - C and C++ were not satisfactory for embedded electronic devices
- Based on C++
  - Significantly simplified (does not include struct, union, enum, pointer arithmetic, and half of the assignment coercions of C++)
  - Supports *only* OOP
  - Has references, but not pointers
  - Includes support for applets and a form of concurrency

Java Evaluation

- Eliminated many unsafe features of C++
- Supports concurrency
- Libraries for applets, GUIs, database access
- Portable: Java Virtual Machine concept, JIT compilers
- Widely used for Web programming
- Use increased faster than any previous language
- Most recent version, 8, released in 2014

Scripting Languages for the Web

- Perl
  - Designed by Larry Wall—first released in 1987
  - Variables are statically typed but implicitly declared
  - Three distinctive namespaces, denoted by the first character of a variable's name
    - All scalar variable names begin with dollar signs ($),
    - All array names begin with at signs (@), and
    - all hash names (hashes are briefly described below) begin with percent signs (%).
  - Powerful, but somewhat dangerous
  - Gained widespread use for CGI programming on the Web
  - Also used for a replacement for UNIX system administration language

Scripting Languages for the Web

- JavaScript
  - Began at Netscape, but later became a joint venture of Netscape and Sun Microsystems
  - A client-side HTML-embedded scripting language
  - Used to dynamically create and modify HTML documents
  - Purely interpreted
  - Related to Java only through similar syntax

Scripting Languages for the Web

- PHP
  - PHP: Hypertext Preprocessor, designed by Rasmus Lerdorf
  - A server-side HTML-embedded scripting language, often used for form processing and database access through the Web
  - Purely interpreted

Scripting Languages for the Web

- Python
  - An OO interpreted scripting language
  - Type checked but dynamically typed
  - Used for CGI programming and form processing
  - Supports lists, tuples, and hashes

Scripting Languages for the Web

- Lua
  - 1990s, Brazil
  - Supports procedural and functional programming
  - An OO interpreted scripting language
  - Type checked but dynamically typed
  - Used for CGI programming and form processing
  - Supports lists, tuples, and hashes, all with its single data structure—the table
  - Easily extendable

Scripting Languages for the Web

- Ruby
  - Designed in Japan by Yukihiro Matsumoto (a.k.a, “Matz”)
  - Began as a replacement for Perl and Python
  - A pure object-oriented scripting language
    - All data are objects
  - Most operators are implemented as methods, which can be redefined by user code
  - Purely interpreted

The Flagship .NET Language: C#

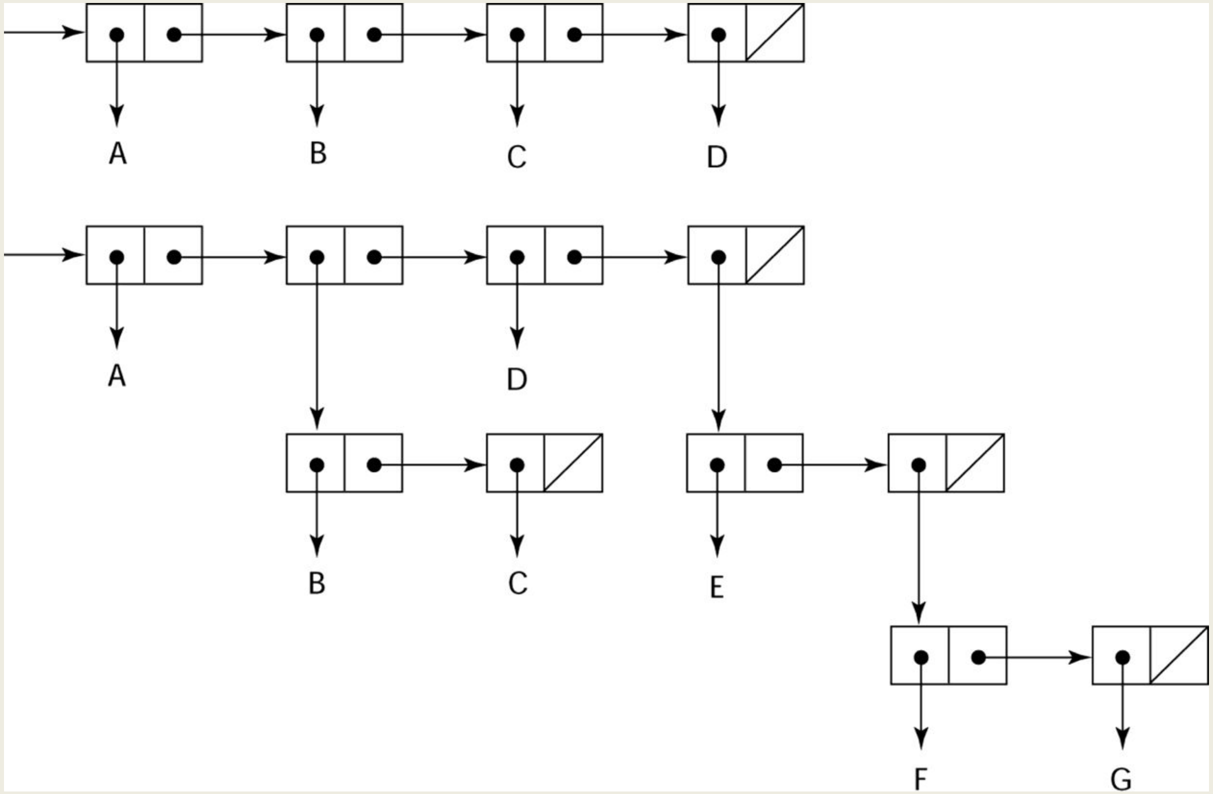
- Part of the .NET development platform (2000)
- Based on C++ , Java, and Delphi
- Includes pointers, delegates, properties, enumeration types, a limited kind of dynamic typing, and anonymous types
- Is evolving rapidly

Functional Programming: Lisp

- LISt Processing language
  - Designed at MIT by John McCarthy
- AI research needed a language to
  - Process data in lists (rather than arrays)
  - Symbolic computation (rather than numeric)
- Only two data types: atoms and lists
- Syntax is based on *lambda calculus*

Representation of Two Lisp Lists

(A B C D)



Representing the lists **`(A B C D)`**
and **`(A (B C) D (E (F G)))`**

Lisp Evaluation

- Pioneered functional programming
  - No need for variables or assignments
  - Control via recursion and conditional expressions
- Still the dominant language for AI
- Common Lisp and Scheme are contemporary dialects of Lisp
- ML (Meta-Language), Haskell, and F# are also functional programming languages, but use very different syntax

Scheme

- Developed at MIT in mid 1970s
- Small
- Extensive use of static scoping
- Functions as first-class entities
- Simple syntax and small size make it ideal for educational applications

Common Lisp

- An effort to combine features of several dialects of Lisp into a single language
- Large, complex, used in industry for some large applications

Programming Based on Logic: Prolog

- Prolog: *Programming Logic*
- Developed by Comerauer and Roussel (University of Aix-Marseille), with help from Kowalski (University of Edinburgh)
- Based on formal logic
- Non-procedural
- Can be summarized as being an intelligent database system that uses an inference process to infer the truth of given queries
- Comparatively inefficient
- Few application areas

Markup/Programming Hybrid Languages

- XSLT
  - eXtensible Markup Language (XML): a metamarkup language
  - eXtensible Stylesheet Language Transformation (XSLT) transforms XML documents (to HTML) for display
  - Programming constructs (e.g., looping and control flow)

Markup/Programming Hybrid Languages

- JSP
  - Java Server Pages: a collection of technologies to support/create *dynamic Web documents*
  - JSTL, a JSP library, includes programming constructs in the form of HTML elements

Computerizing Business Records: COBOL

- COBOL design process
  - First Design Meeting (Pentagon) - May 1959
  - Design goals
    - Must look like simple English
    - Must be easy to use, even if that means it will be less powerful
    - Must broaden the base of computer users
    - Must not be biased by current compiler problems
  - Design committee members were all from computer manufacturers and DoD branches
  - Design Problems: arithmetic expressions? subscripts? Fights among manufacturers

COBOL Evaluation

- Contributions
  - First macro facility in a high-level language (DEFINE verb)
  - Hierarchical data structures (records)
  - Nested selection statements
  - Long names (up to 30 characters), with hyphens
  - Separate data division

COBOL: DoD Influence

- First language required by DoD
  - would have failed without DoD
- Still the most widely used business applications language

The Beginning of Timesharing: Basic

- Designed by Kemeny & Kurtz at Dartmouth
- Design Goals:
    - Easy to learn and use for non-science students
    - Must be “pleasant and friendly”
    - Fast turnaround for homework
    - Free and private access
    - User time is more important than computer time
- Current popular dialect: Visual Basic
- First widely used language with time sharing

Everything for Everybody: PL/I

- Designed by IBM and SHARE
- Computing situation in 1964 (IBM's point of view)
  - Scientific computing
    - IBM 1620 and 7090 computers
    - FORTRAN
    - SHARE user group
  - Business computing
    - IBM 1401, 7080 computers
    - COBOL
    - GUIDE user group

PL/I: Background

- By 1963
  - Scientific users began to used floating-point data and arrays extensively; business users began to need more elaborate I/O
  - It looked like many shops would begin to need two kinds of computers, languages, and support staff--too costly

PL/I: Background

- The obvious solution
  - Build a new computer to do both kinds of applications
  - Design a new language to do both kinds of applications

PL/I: Design Process

- Designed in five months by the 3 X 3 Committee
  - Three members from IBM, three members from SHARE
- Initial concept
  - An extension of Fortran IV
- Initially called NPL (New Programming Language)
- Name changed to PL/I in 1965

PL/I: Evaluation

- PL/I contributions
  - First unit-level concurrency
  - First exception handling
  - Switch-selectable recursion
  - First pointer data type
  - First array cross sections
- Concerns
  - Many new features were poorly designed
  - Too large and too complex