ICC043/IEC582 – Paradigmas de Linguagens de Programação

História e Evolução da Computação e das LP

Prof. Dr. Rafael Giusti
rgiusti@icomp.ufam.edu.br

Referências

- **Leitura recomendada**
  - SEBESTA. *Concepts of Programming Languages*
    - Capítulos 1 e 2
- **Leitura complementar**
  - SCOTT. *Programming Language Pragmatics*
    - Capítulo 1, seções 1.1, 1.2 e 1.3
- **Na web**
  - www.computinghistory.org.uk/cgi/computing-timeline.pl
  - www.tiobe.com
  - en.wikipedia.org/wiki/Von_Neumann_architecture
  - en.wikipedia.org/wiki/IBM_704
  - www.americanscientist.org/article/the-semicolon-wars
  - www.rosettacode.org

Agenda

- Motivação e conceitos introdutórios
- Breve história da computação
- Breve história das linguagens de programação

Motivação

- Sobre a riqueza da história da computação
- Sobre a evolução das linguagens de programação
  - Por que existem tantas linguagens?
  - Por que tantas linguagens são tão parecidas?
  - ~~Qual é a melhor linguagem~~?
    - Por que perguntar "qual é a melhor linguagem" não faz nenhum sentido?
  - Por que estudar paradigmas e conceitos de linguagens de programação?

Motivação

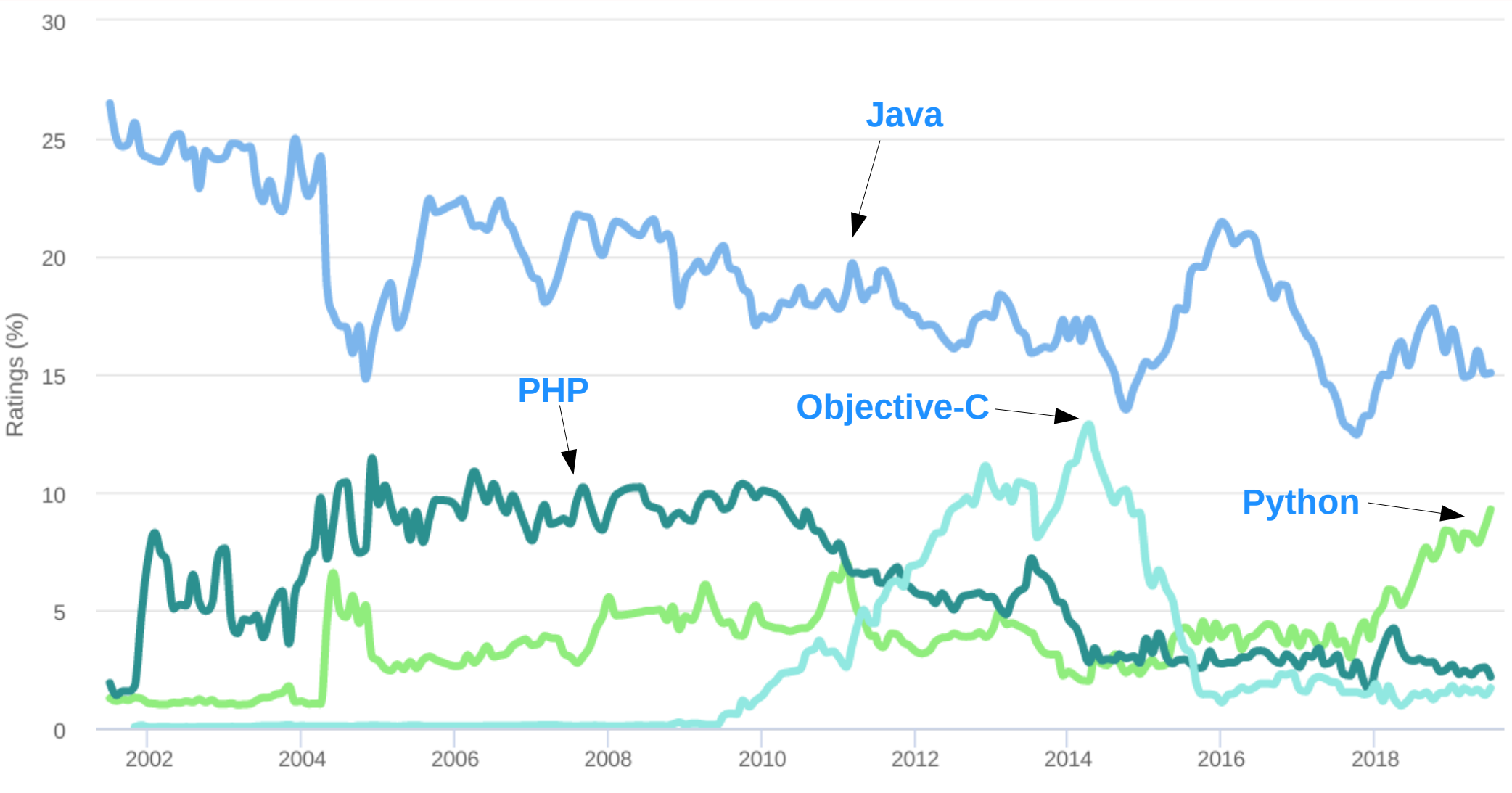
- Por que estudar linguagens de programação?
  - Aumento da capacidade de expressar ideias
  - Facilidade em entender linguagens
  - Embasamento para escolher linguagens
  - Entender a importância da implementação
  - Utilizar melhor linguagens conhecidas
  - Participar do avanço geral da computação

Aprender novas linguagens é importante?

Jul 2019	Jul 2018	Change	Programming Language
1	1		Java
2	2		C
3	4	↑	Python
4	3	↓	C++
5	6	↑	C#
6	5	↓	Visual Basic .NET
7	8	↑	JavaScript
8	7	↓	PHP
9	9		SQL
10	10		Objective-C
11	12	↑	Ruby
12	13	↑	Assembly language
13	11	↓	Swift
14	15	↑	MATLAB
15	81	↑↑	Groovy
16	18	↑	Go
17	19	↑	Visual Basic
18	16	↓	Delphi/Object Pascal
19	17	↓	Perl
20	14	↓↓	R

https://www.tiobe.com/tiobe-index/

Aprender novas linguagens é importante?



https://www.tiobe.com/tiobe-index/

Aprender novas linguagens é importante?

- A **capacidade de aprender** é mais importante do que o rol de linguagens que você conhece
- O IComp não forma programadores
  - Queremos que vocês desenvolvam sua própria capacidade de se adaptar a um cenário em constante evolução
  - Compreender como as linguagens se desenvolveram e como elas são criadas faz diferença?
    - Uma pergunta pra ser respondida daqui a uns quatro meses...

Linguagens de programação

- O que é uma linguagem de programação?
  - Uma linguagem de programação é um conjunto de regras sintáticas e semânticas usadas para descrever algoritmos
    - Sintaxe: refere-se à **forma** dos programas
      - Exemplo: uma atribuição tem a forma `<variável> = <expressão>`
    - Semântica: refere-se ao **significado** dos programas
      - Exemplo: após uma atribuição, a variável assume o valor da expressão

Linguagens de programação

- A natureza das LP tem íntima relação com a história da computação
- Essa história é bem mais antiga do que costumamos pensar
- Vamos ver que se passaram muitos anos até que um método realmente adequado de programar computadores fosse disseminado (FORTRAN)

Agenda

- Motivação e conceitos introdutórios
- Breve história da computação
- Breve história das linguagens de programação

"Pré-história"

- Antes de haver linguagens de programação, havia computação

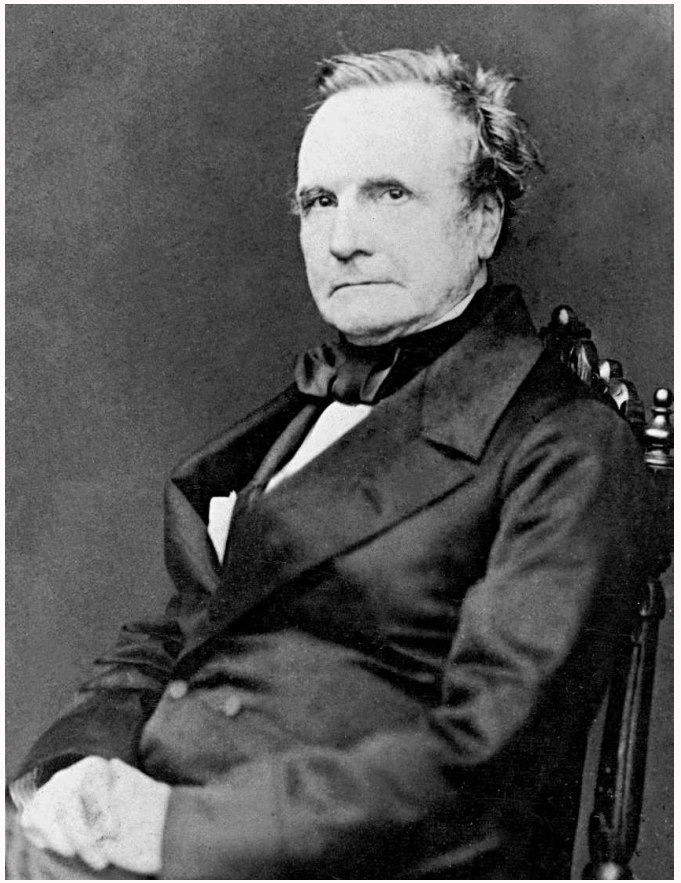
Uma computadora em 1952 [1]

Pascalina, uma máquina de calcular construída por Blaise Pascal no séc. XVII [2]

[1] https://en.wikipedia.org/wiki/File:X-4_with_Female_Computer_-_GPN-2000-001932.jpg
[2] https://en.wikipedia.org/wiki/Pascal%27s_calculator#/media/File:Pascaline-CnAM_823-1-IMG_1506-black.jpg

Máquina diferencial

- Charles Babbage concebeu e *começou* a construção da **máquina diferencial** no século XIX
  - Tratava-se de uma máquina de calcular para logaritmos e funções trigonométricas
  - Durante o projeto da máquina diferencial, Babbage concebeu um projeto para a **máquina analítica**



Charles Babbage [1]

[1] https://en.wikipedia.org/wiki/File:Charles_Babbage_-_1860.jpg

Máquina analítica e a primeira programadora

Analytical Engine Mill – a "Unidade Aritmética" da máquina analítica [1]

Ada Lovelace (ca. 1843) [2]

[1] https://en.wikipedia.org/wiki/Analytical_Engine#/media/File:Analytical_Engine_(2290032530).jpg
[2] https://en.wikipedia.org/wiki/Ada_Lovelace#/media/File:Ada_Byron_daguerreotype_by_Antoine_Claudet_1843_or_1850.jpg

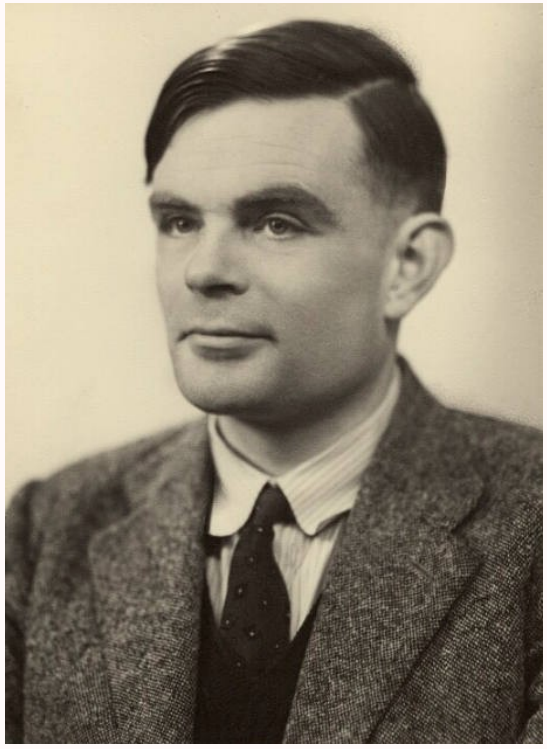
A primeira programadora

- **Ada Lovelace** foi a primeira pessoa a enxergar o verdadeiro potencial da máquina analítica
  - Mais que uma máquina de fazer cálculos, a máquina analítica seria, de fato, um **computador multi-propósito**
  - A máquina analítica era **Turing-completa**: qualquer cálculo que um computador pode fazer, a máquina analítica também poderia
- Ada também escreveu os primeiros programas para a máquina analítica (embora alguns suspeitem que Babbage tenha feito isso primeiro)

Formalização matemática

- Na primeira metade do século XX, matemáticos formalizavam o conceito de **computabilidade** e estabeleciam as fundações da computação

Alonzo Church, inventor do cálculo lambda [1]



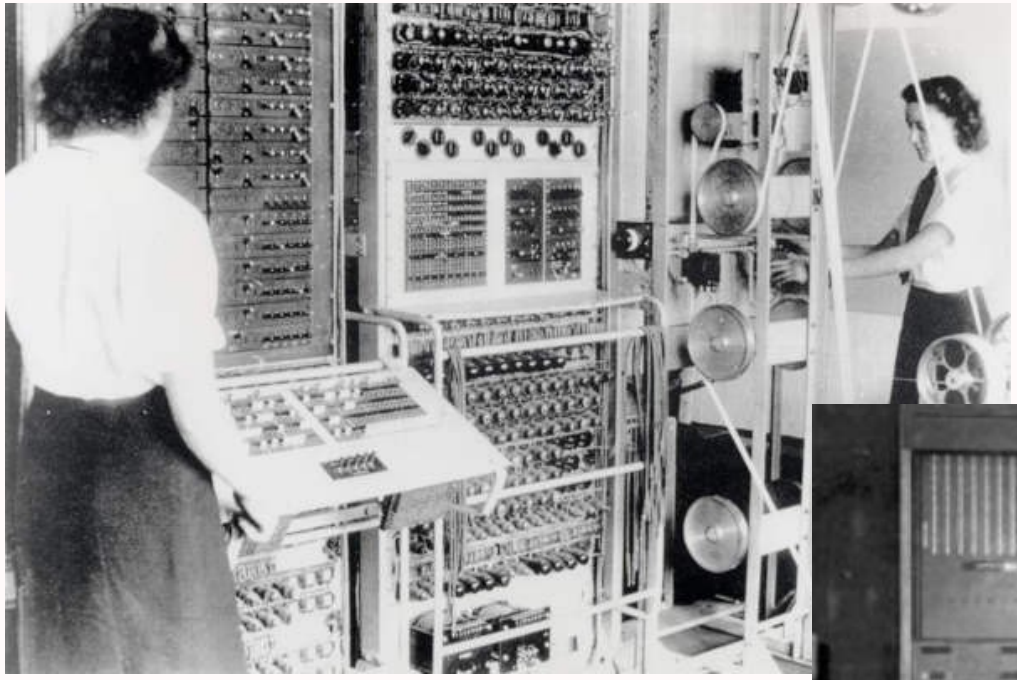
Alan Turing, criador da máquina de Turing [2]

[1] https://en.wikipedia.org/wiki/File:Alonzo_Church.jpg
[2] https://en.wikipedia.org/wiki/File:Alan-Turing.jpg

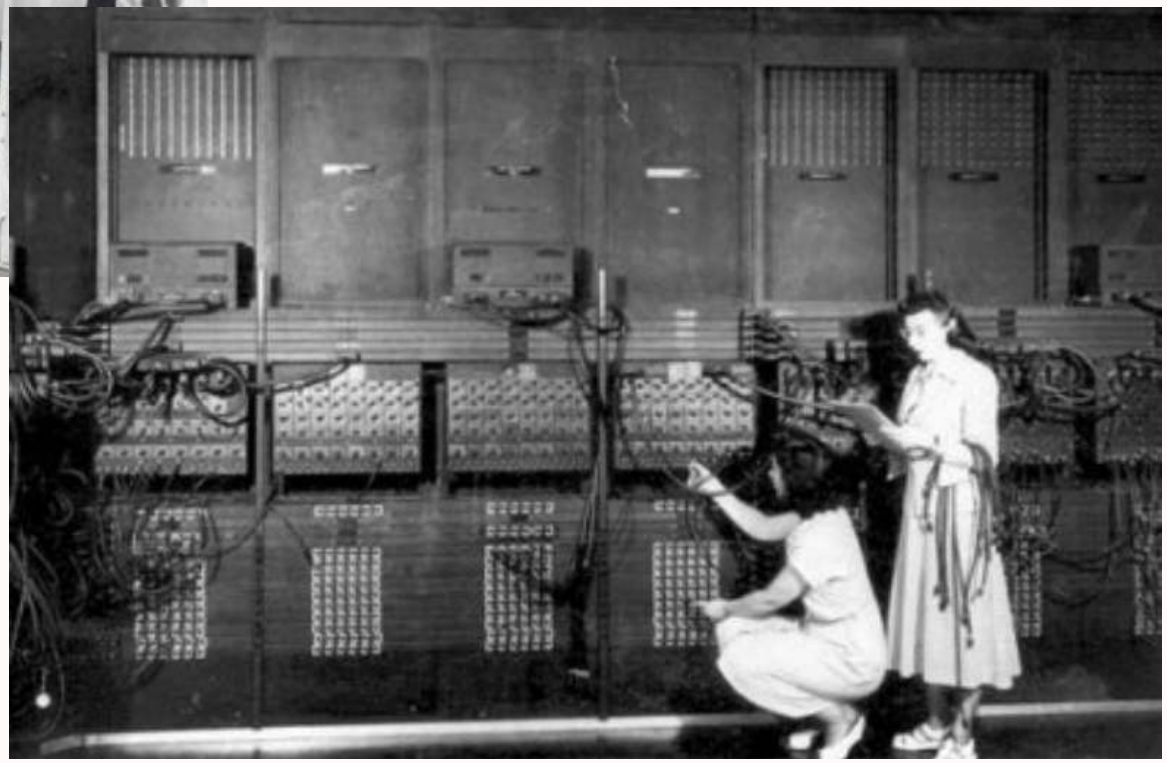
Formalização matemática

- Church e Turing procuravam uma solução para um desafio proposto por David Hilber em 1928
- O ***Entscheidungsproblem*** (**problema de decisão**)
  - É possível conceber um algoritmo que recebe uma expressão lógica de primeira ordem e um conjunto de axiomas...
  - E identifica se a expressão é universalmente válida?
    - Isto é, pode-se provar a expressão através dos axiomas usando as regras da lógica?

Computadores



Colossus (*ca*. 1944) [1]



ENIAC (*ca*. 1945) [2]

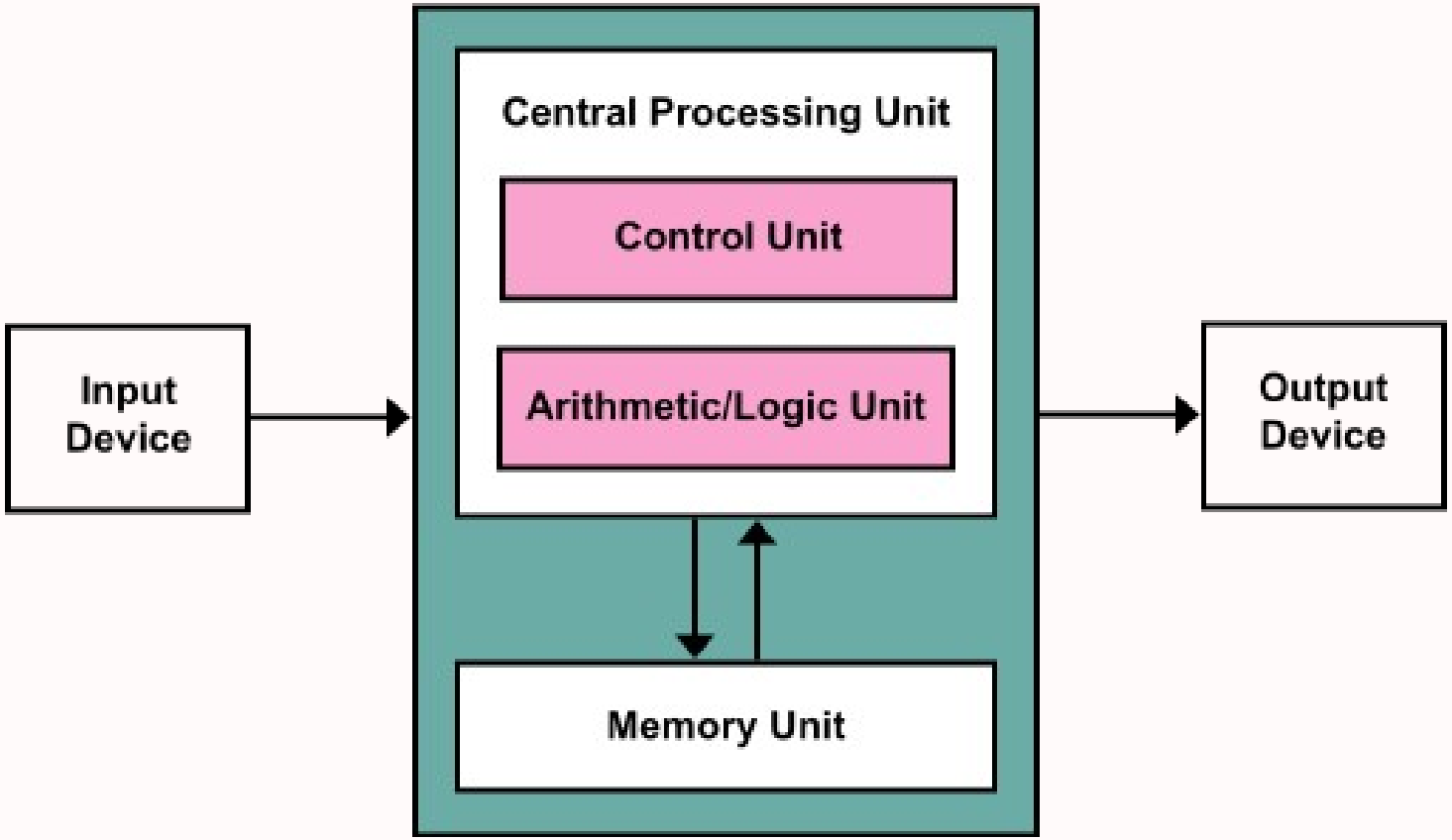
[1] https://en.wikipedia.org/wiki/Colossus_computer
[2] http://zuse-z1.zib.de/simulations/eniac/history.html

ENIAC

- O ENIAC era uma "calculadora para algoritmos"
  - Podia executar laços de repetição, comandos de seleção (`if`) e sub-rotinas
  - Mas não possuía um programa em memória
  - O algoritmo era programado diretamente no *hardware*, através de cabos e relês
    - Simulador em Java http://zuse-z1.zib.de/simulations/eniac/

Programa armazenado

- O primeiro computador capaz de executar um **programa armazenado** foi o EDSAC (1949)



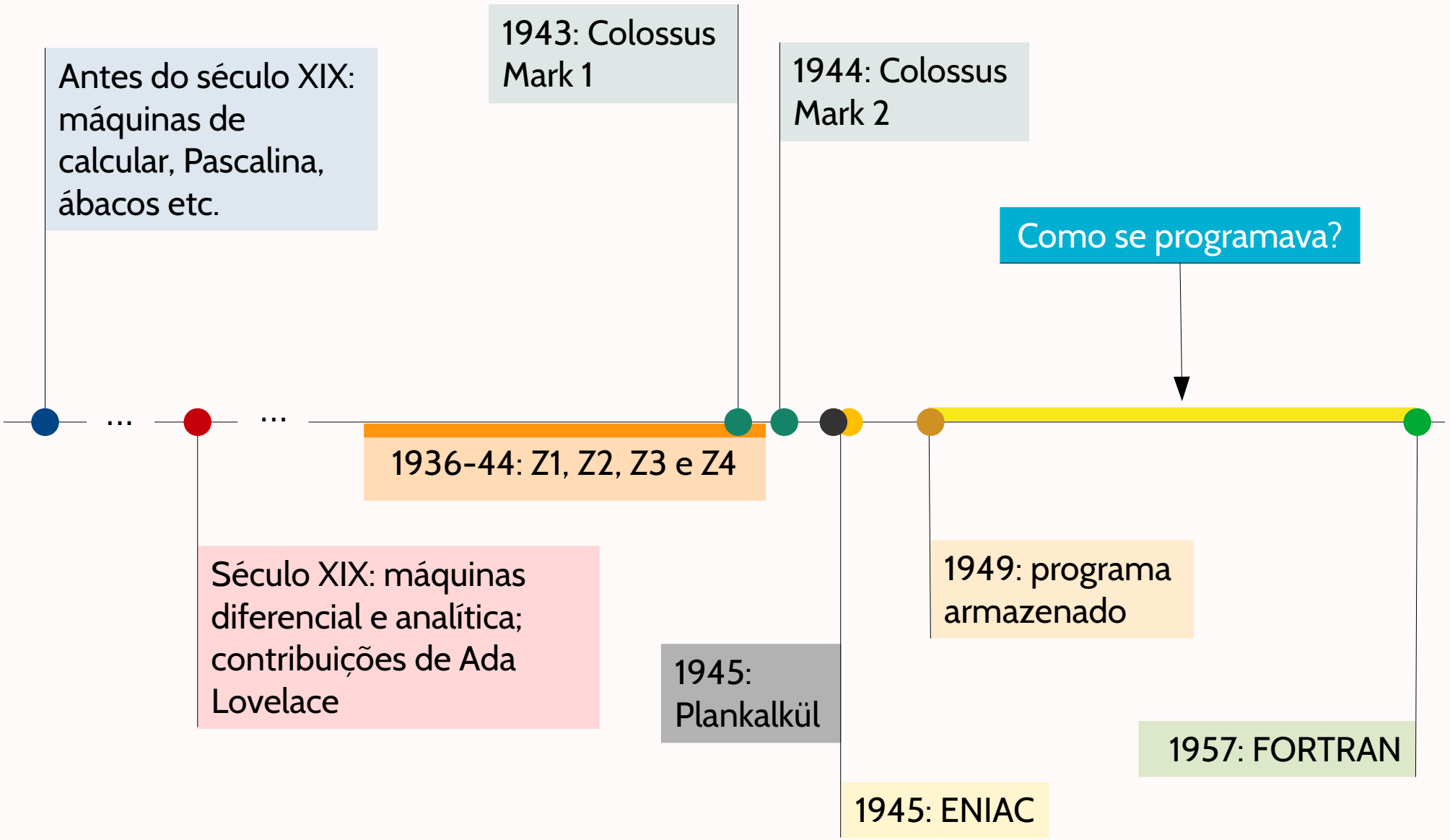
Programa armazenado e arquitetura de von Neumann

https://en.wikipedia.org/wiki/Von_Neumann_architecture

Enquanto isso, na Alemanha...

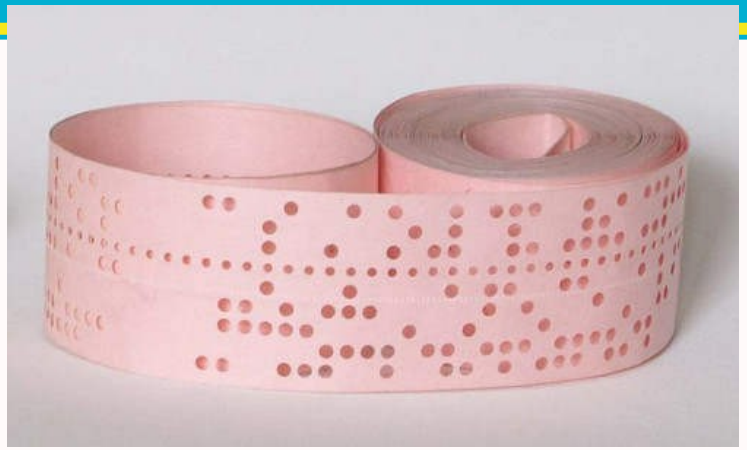
- Konrad Zuse construiu uma série de computadores eletromecânicos entre 1936-45
  - O Z3 foi o primeiro computador multi-propósito da história (1941)
    - Todos os modelos de Zuse foram destruídos durante a guerra, exceto o Z4
  - Em 1945, isolado em um vilarejo na Bavária, Zuse finalizou um manuscrito descrevendo sua linguagem de programação **Plankalkül**
    - Os manuscritos foram esquecidos e publicados apenas em 1972

Linha do tempo

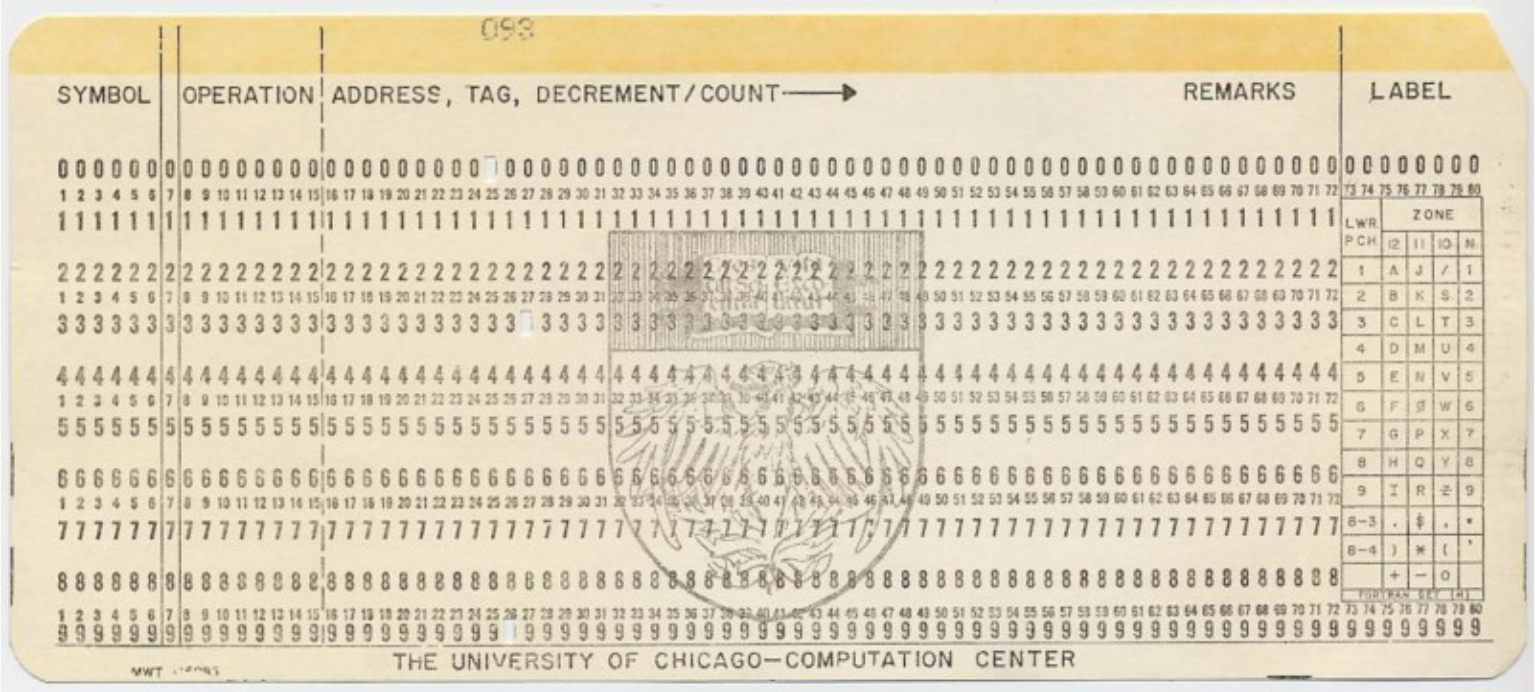


Código de máquina e Assembly

- Programas eram escritos em código de máquina ou Assembly, usando cartões ou fitas perfuradas



Fita perfurada [1]



Cartão perfurado para Assembly (computadores da série IBM 709) [2]

[1] https://simple.wikipedia.org/wiki/Punched_tape#/media/File:PaperTapes-5and8Hole.jpg
[2] https://homepage.divms.uiowa.edu/~jones/cards/collection/i-program.html

Código de máquina e Assembly

- O termo **Assembly** designa uma família de linguagens que usam mnemônicos para representar comandos da linguagem de máquina
  - Exemplo

Arquitetura: 6502 (NES)		
Código	**Assembly**	**Efeito**
A9 80	LDA #$80	Carrega o valor 0x80 para o acumulador
85 B7	STA $83	Guarda o valor do acumulador no endereço 0x83
38	SEC	Limpa a *flag* de carga (*carry bit*)

Assembly

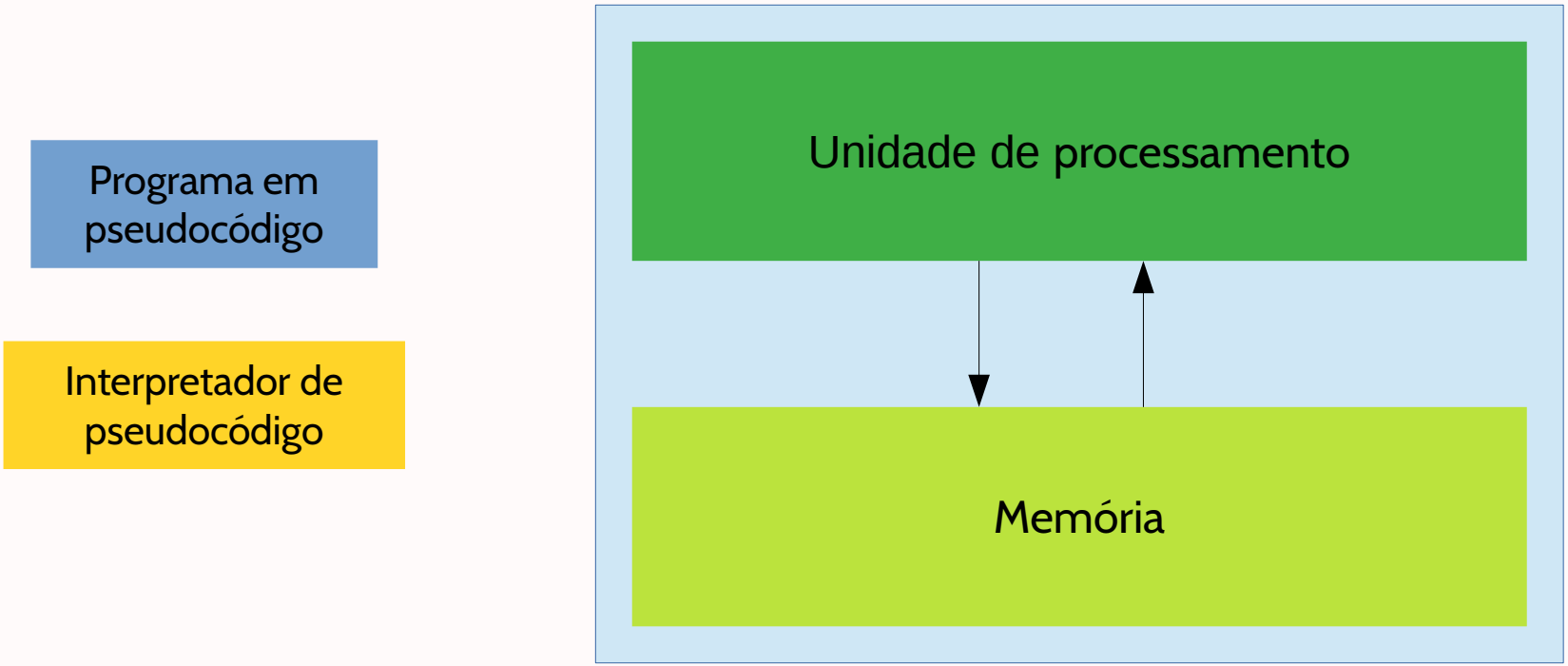
- Inicialmente, Assembly fornecia apenas mnemônicos
  - Rapidamente incorporou recursos como macros e nomes simbólicos
  - Exemplo hipotético

Código de máquina	Assembly
`0080 0010 A0130744`	`.DATA 1 var` `ADD A, var`

  - O programa que traduz Assembly é chamado **montador** ou **assembler**

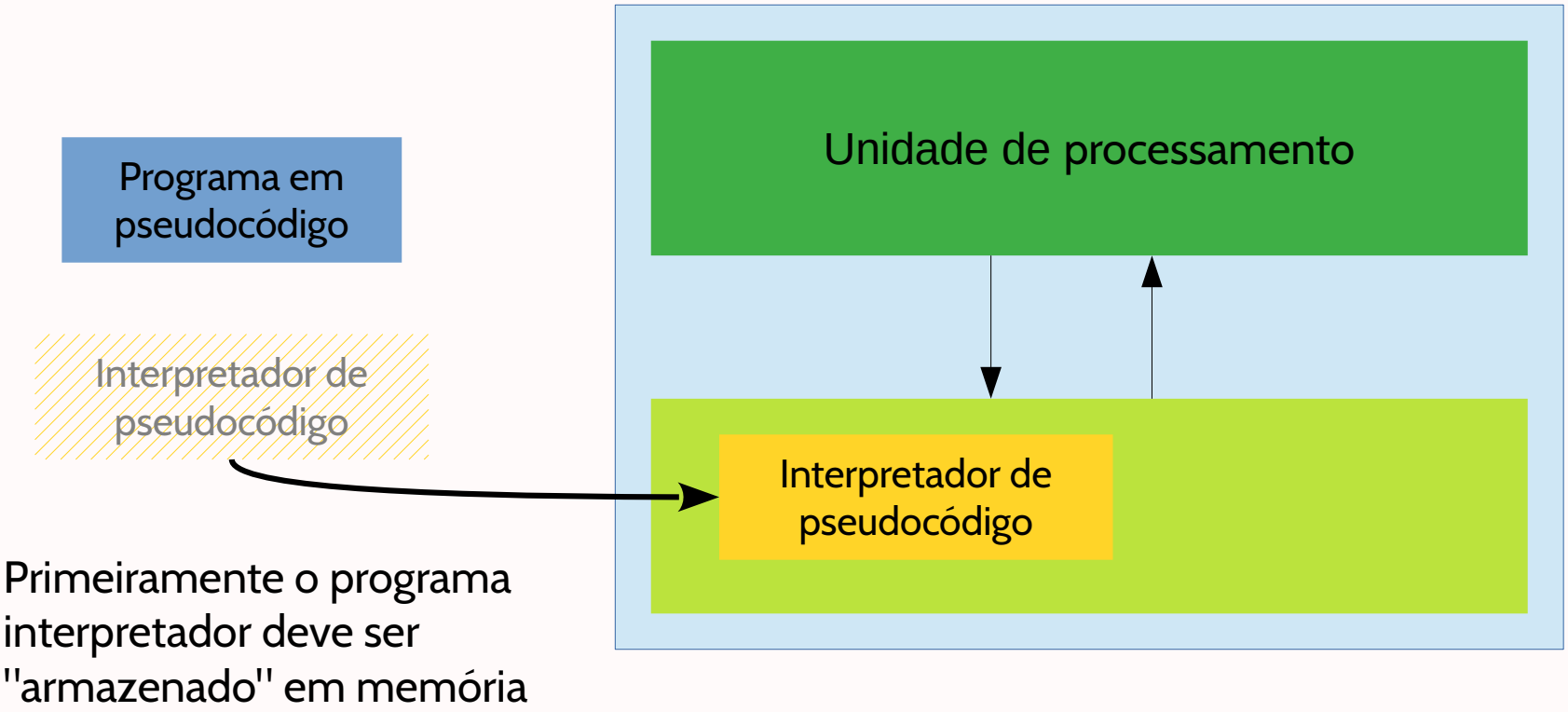
Pseudocódigo

- Linguagens de **pseudocódigo** eram interpretadas e especificavam sub-rotinas para serem utilizadas em um programa



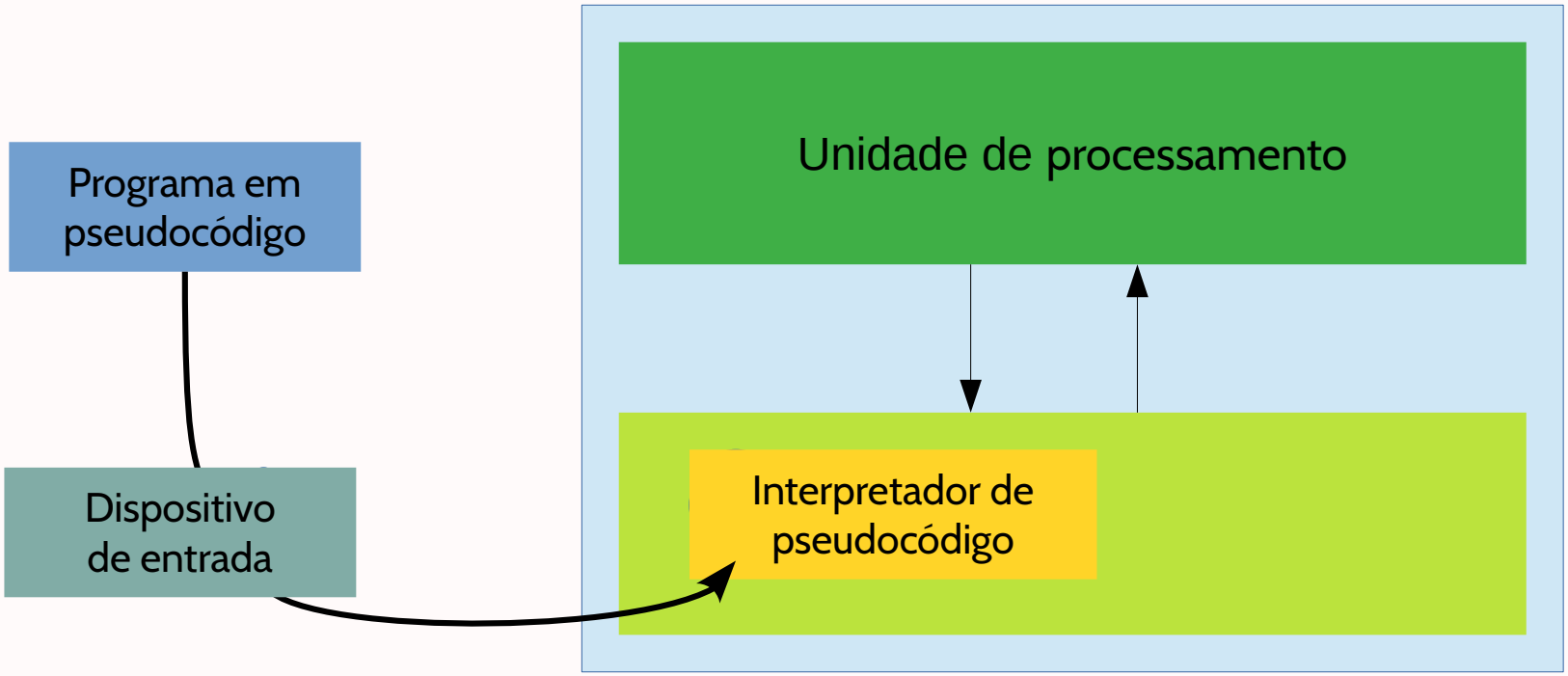
Pseudocódigo

- Linguagens de **pseudocódigo** eram interpretadas e especificavam sub-rotinas para serem utilizadas em um programa



Pseudocódigo

- Linguagens de **pseudocódigo** eram interpretadas e especificavam sub-rotinas para serem utilizadas em um programa



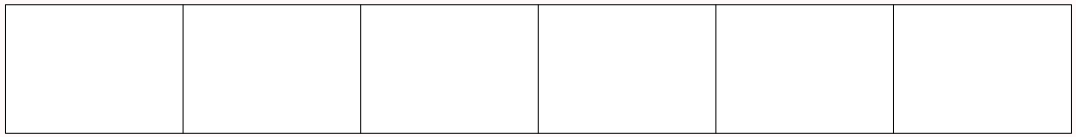
A partir daí, o interpreador lê o pseudocódigo por meio do dispositivo de entrada (*e.g.*, fita magnética ou cartão perfurado) e executa as sub-rotinas

Short code

- O ***Short Code*** foi um pseudocódigo para expressões algébricas criado em 1949
- Comandos e variáveis eram pares de bytes
  - Um byte em *Short Code* são seis bits
  - Exemplos
    - X0, Y0, X1, Y1: nomes de variáveis
    - 01: código para operação de subtração
    - 07: código para adição
    - 09 e 02: códigos para abrir e fechar um nível de precedência

Short code

- O programador de *Short Code* devia primeiramente escrever a expressão e traduzi-la de acordo com os códigos
- Exemplo
  - Expressão:
    - `X0 = SQRT(ABS(Y0))`
  - Código:



Short code

- O programador de *Short Code* devia primeiramente escrever a expressão e traduzi-la de acordo com os códigos
- Exemplo
  - Expressão:
    - `X0 = SQRT(ABS(Y0))`
  - Código:

	X0				

Short code

- O programador de *Short Code* devia primeiramente escrever a expressão e traduzi-la de acordo com os códigos
- Exemplo
  - Expressão:
    - `X0 = SQRT(ABS(Y0))`
  - Código:

	X0	03			

Short code

- O programador de *Short Code* devia primeiramente escrever a expressão e traduzi-la de acordo com os códigos
- Exemplo
  - Expressão:
    - `X0 = SQRT(ABS(Y0))`
  - Código:

	X0	03	20		

Short code

- O programador de *Short Code* devia primeiramente escrever a expressão e traduzi-la de acordo com os códigos
- Exemplo
  - Expressão:
    - `X0 = SQRT(ABS(Y0))`
  - Código:

	X0	03	20	06	

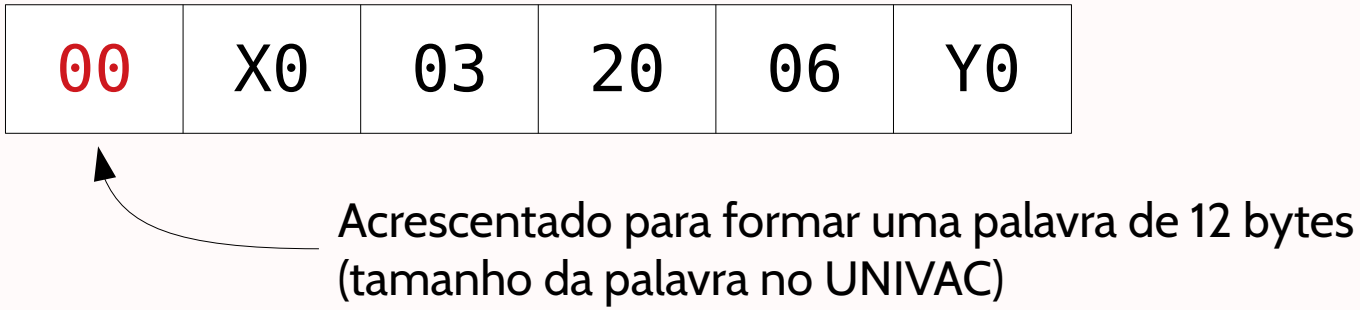
Short code

- O programador de *Short Code* devia primeiramente escrever a expressão e traduzi-la de acordo com os códigos
- Exemplo
  - Expressão:
    - `X0 = SQRT(ABS(Y0))`
  - Código:

	X0	03	20	06	Y0

Short code

- O programador de *Short Code* devia primeiramente escrever a expressão e traduzi-la de acordo com os códigos
- Exemplo
  - Expressão:
    - `X0 = SQRT(ABS(Y0))`
  - Código:



Short code

- Programar em *Short Code* era mais simples do que utilizar linguagem de máquina ou Assembly
- Mas gerava problemas
  - O programa em *Short Code* e o interpretador exigiam uma grande quantidade de memória
  - O programa em *Short Code* tinha que ser lido em tempo de execução
    - Normalmente a partir de fita magnética
  - O processo de interpretação podia ser até 50 vezes mais lento que Assembly

Sistema de compilação do UNIVAC

- Entre 1951 e 1953, uma equipe liderada por Grace Hopper desenvolveu diversos sistemas de compilação para o UNIVAC
- Chamados A-0, A-1 e A-2, esses sistemas se asse-melhavam a macros
  - Os programas em pseudocódigo eram expandidos em código

Grace Hooper e um painel do UNIVAC [1]

https://en.wikipedia.org/wiki/Grace_Hopper#/media/File:Grace_Hopper_and_UNIVAC.jpg

Pontos positivos e negativos

- Todas as partes tinham seus argumentos

☺ Programas em Assembly eram muito mais eficientes

☹ O desenvolvimento era caro e propenso a erros

☹ A manutenção era muito cara

☺ Pseudocódigo era intuitivo e simples

☹ O interpretador consumia muita memória e era lento

☺ Os compiladores permitiam desenvolvimento com pseudocódigo

☺ Programas compilados tinham compromisso razoável entre custo de desenvolvimento e manutenção e tempo de execução

☹ Ainda assim, não eram tão eficientes quanto Assembly

Pontos flutuantes

- Na prática, o custo dos sistemas de interpretação era tolerado pela necessidade de "simular" cáclulos de ponto flutuante
  - Pontos flutuantes
    - Números na forma $mantissa \times base^{expoente}$
  - O *overhead* causado pelo interpretador era relativamente pequeno
- Unidades aritméticas de ponto flutuante e suporte do *hardware* para operações de indexação mudaram isso

IBM 704

- Lançado em 1954, foi o primeiro computador produzido em larga escala com suporte a aritmética de ponto flutuante em *hardware*

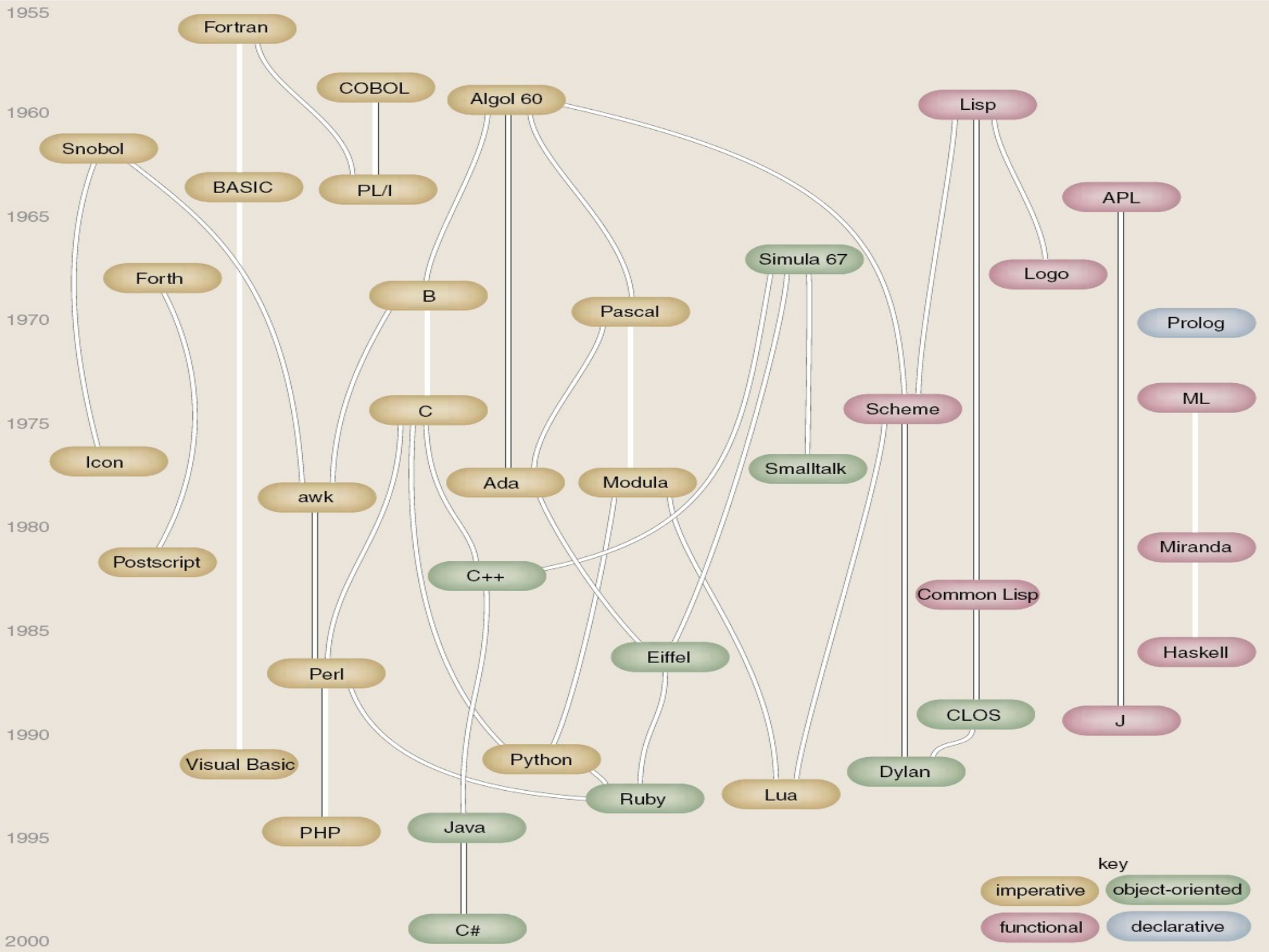
IBM 704 e unidades de fita[1]

IBM 704 em 1957 [2]

[1] https://en.wikipedia.org/wiki/IBM_704#/media/File:IBM_704_mainframe.gif
[2] https://en.wikipedia.org/wiki/IBM_704#/media/File:IBM_Electronic_Data_Processing_Machine_-_GPN-2000-001881.jpg

Agenda

- Motivação e conceitos introdutórios
- Breve história da computação
- Breve história das linguagens de programação


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Simula 67
Forth
Logo
B
Pascal
Prolog
ML
C
Scheme
Icon
Smalltalk
Ada
Modula
awk
Postscript
Miranda
C++
Common Lisp
Haskell
Eiffel
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

FORTRAN (1957)

- **For**mula **Tran**slator
  - Originalmente proposto por John Backus
  - Motivações
    - Ser expressivo como linguagens de programação de alto nível (interpretadas)
    - Ser eficiente quanto código de máquina e Assembly
  - Foi impulsionada pelo IBM 704 e sua capacidade de fazer operações de ponto flutuante e indexação em *hardware*

FORTRAN (1957)

- Cenário da época
  - Memória lenta, pouco confiável e escassa
  - A finalidade era computação científica
  - Programação "eficiente" era inexistente
  - Velocidade do programa era o objetivo principal

FORTRAN (1957)

- Variáveis
  - Identificadores limitados a 6 caracteres
  - Primeira letra `I`, `J`, `K`, `L`, `M` ou `N`: inteiros
  - O restante: ponto flutuante
- Funções para E/S
  - FORMAT, READ, READ INPUT TAPE, WRITE, WRITE OUTPUT TAPE, READ TAPE...
  - GO TO
    - Condicional
    - Incondicional

FORTRAN (1957)

- O desenvolvimento de FORTRAN foi bastante influenciado pelas capacidades do IBM 704
  - E potencialmente vice-versa
- IF em 3 vias
  - `IF (`*expressão*`) n1 n2 n3`
    - Se expressão < 0, GOTO n1
    - Se expressão = 0, GOTO n2
    - Se expressão > 0, GOTO n3

FORTRAN (1957)

C ← FOR COMMENT / STATEMENT NUMBER (1–5)	CONTINUATION (6)	FORTRAN STATEMENT (7–72)	IDENTIFICATION (73–80)
C		PROGRAM FOR FINDING THE LARGEST VALUE	
C	X	ATTAINED BY A SET OF NUMBERS	
		DIMENSION A(999)	
		FREQUENCY 30(2,1,10), 5(100)	
		READ 1, N, (A(I), I = 1,N)	
1		FORMAT (I3/(12F6.2))	
		BIGA = A(1)	
5		DO 20 I = 2,N	
30		IF (BIGA-A(I)) 10,20,20	
10		BIGA = A(I)	
20		CONTINUE	
		PRINT 2, N, BIGA	
2		FORMAT (22H1THE LARGEST OF THESE I3, 12H NUMBERS IS F7.2)	
		STOP 77777	

http://www.paulgraham.com/history.html

FORTRAN II (1958)

- Corrige muitos *bugs* de FORTRAN I
- Acrescenta a possibilidade de compilação separada de sub-rotinas
  - Em FORTRAN I, todo o programa precisa ser compilado de uma vez
  - Programas eram limitados a aproximadamente 300~400 linhas de códigos
    - Programas mais longos tinham pouca chance de serem compilados

FORTRAN III

- Nunca foi distribuído em larga escala

COBOL (1959)

- FORTRAN preencheu muito bem um nicho
  - Computação científica
  - Deixou em aberto o nicho da computação comercial
  - Grace Hopper:
    - “Programas matemáticos devem ser escritos em notação matemática; programas para processamento de dados devem ser escritos em inglês”.

COBOL (1959)

- COBOL foi desenvolvida por um comitê financiado pelo Departamento de Defesa dos Estados Unidos
- Objetivos
  - Usar inglês tanto quanto possível
  - Facilidade de uso
- Expectativa
  - Que a linguagem fosse utilizada por não programadores

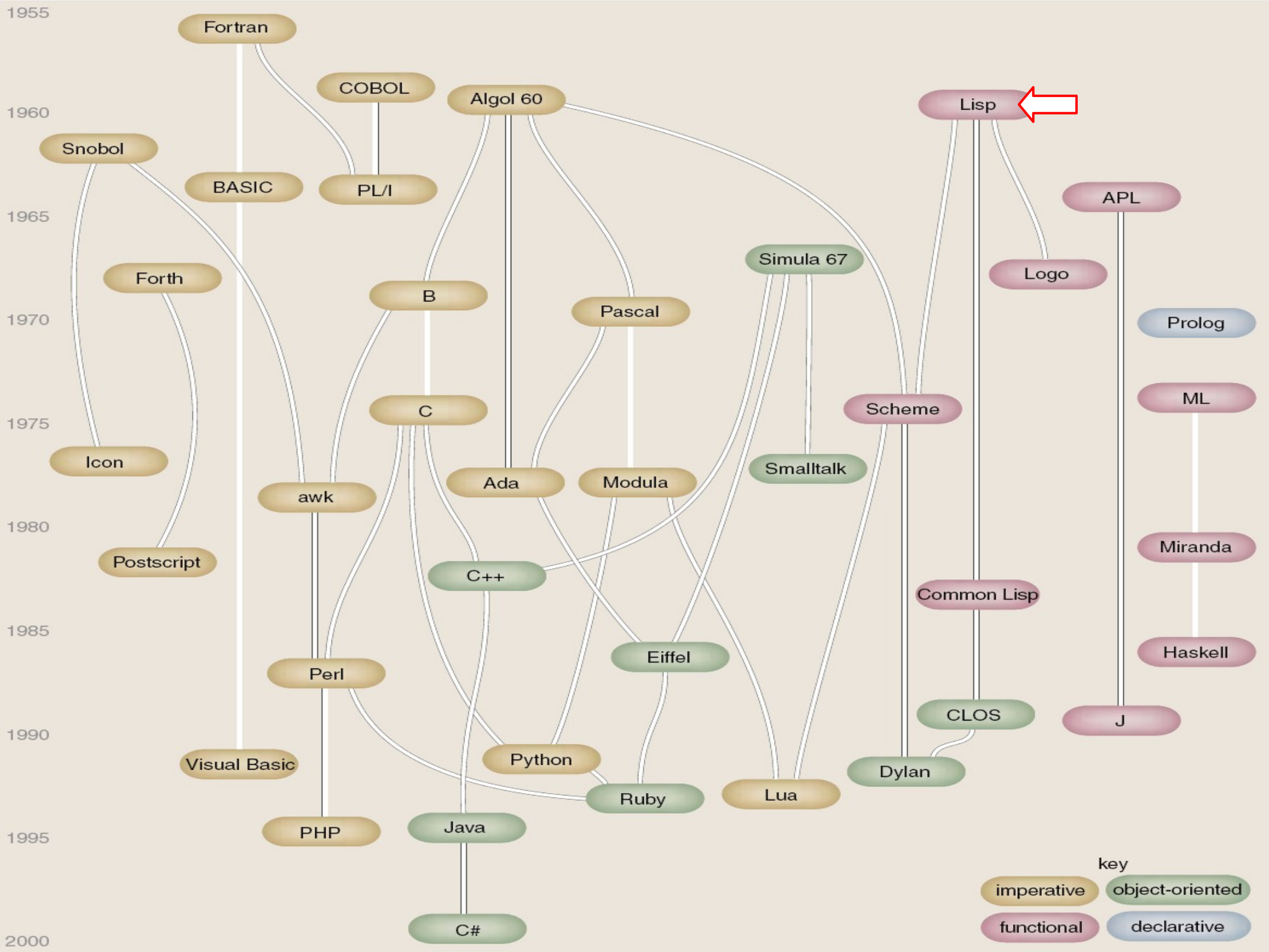
COBOL (1959)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. A-Plus-B.

DATA DIVISION.
WORKING-STORAGE SECTION.
01  A        PIC S9(5).
01  B        PIC S9(5).

01  A-B-Sum PIC S9(5).

PROCEDURE DIVISION.
    ACCEPT A
    ACCEPT B

    ADD A TO B GIVING A-B-Sum

    DISPLAY A-B-Sum

    GOBACK
    .
```

```
identification division.
program-id. caesar.
data division.
1 msg pic x(50)
    value "The quick brown fox jumped over the lazy dog.".
1 offset binary pic 9(4) value 7.
1 from-chars pic x(52).
1 to-chars pic x(52).
1 tabl.
 2 pic x(26) value "abcdefghijklmnopqrstuvwxyz".
 2 pic x(26) value "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
 2 pic x(26) value "abcdefghijklmnopqrstuvwxyz".
 2 pic x(26) value "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
procedure division.
begin.
    display msg
    perform encrypt
    display msg
    perform decrypt
    display msg
    stop run
    .

encrypt.
    move tabl (1:52) to from-chars
    move tabl (1 + offset:52) to to-chars
    inspect msg converting from-chars
        to to-chars
    .

decrypt.
    move tabl (1 + offset:52) to from-chars
    move tabl (1:52) to to-chars
    inspect msg converting from-chars
        to to-chars
    .
end program caesar.
```

FORTRAN IV (1962)

```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      INTEGER A,B,C
      READ(5,501) A,B,C
  501 FORMAT(3I5)
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C))
      WRITE(6,601) A,B,C,AREA
  601 FORMAT(4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,12HSQUARE UNITS)
      STOP
      END
```


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Simula 67
Forth
Logo
B
Pascal
Prolog
C
Scheme
ML
Icon
Smalltalk
Ada
Modula
awk
Postscript
Miranda
C++
Common Lisp
Eiffel
Haskell
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

LISP (1958)

- Primeira linguagem do paradigma funcional
- Motivada por pesquisa em Inteligência Artificial
  - Demonstração automática de teoremas, modelagem do pensamento humano etc.
  - Necessidade de uma linguagem adequada para processamento de listas
    - LISP → List Processor
  - Todas as estruturas de LISP são objetos atômicos ou listas
    - **Ortogonalidade**

LISP (1958)

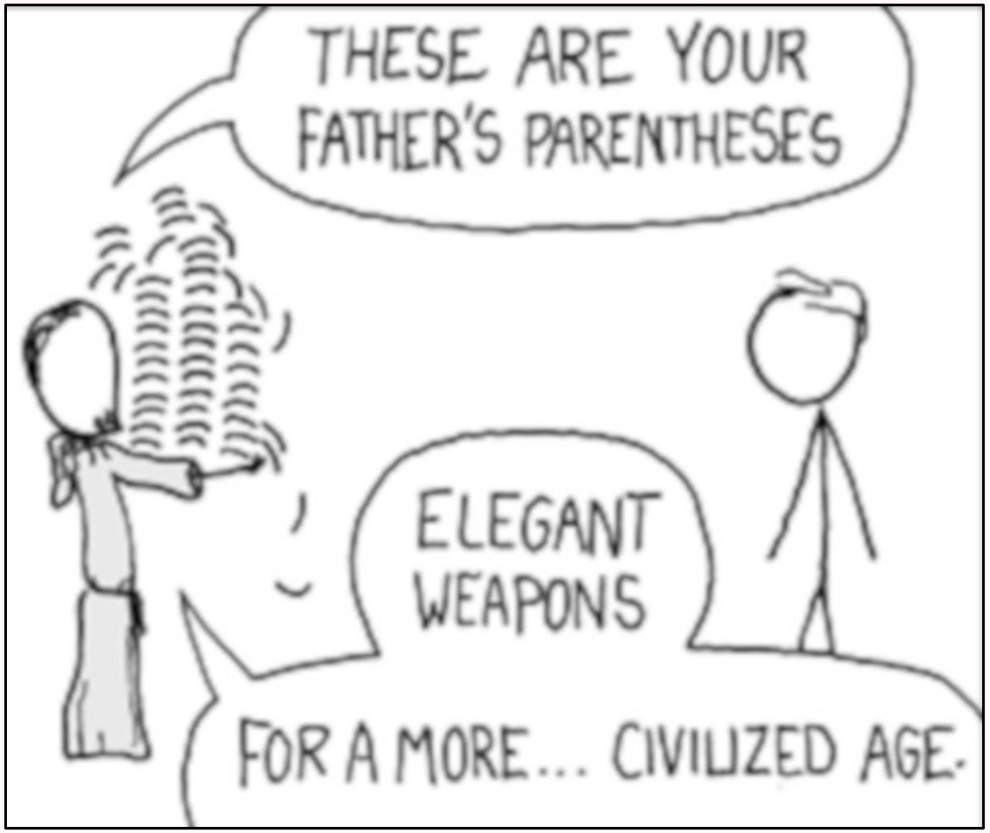
- Uma lista pode ser interpretada como dados ou como código
- Exemplo
  - (A B C D)
    - Como dado: uma lista que contém quatro elementos
    - Como códgo: uma função de nome "A" que recebe três argumentos

LISP (1958)

- Uma lista pode ser interpretada como dados ou como código
- Exemplo
  - (A (B C) (D E))
    - Como dado: uma lista que contém três elementos: "A", "(B C)" e "(D E)"
    - Como códgo: uma função de nome "A" que recebe dois argumentos

LISP (1958)

- LISP é uma linguagem expressiva e simples...
- ...frequentemente criticada pela quantidade de parênteses aninhados



https://xkcd.com/297/

LISP (1958)

```
; LISP Example function
;  The following code defines a LISP predicate function
;  that takes two lists as arguments and returns True
;  if the two lists are equal, and NIL (false) otherwise
(DEFUN equal_lists (lis1 lis2)
    (COND
        ((ATOM lis1) (EQ lis1 lis2))
        ((ATOM lis2) NIL)
        ((equal_lists (CAR lis1) (CAR lis2))
                (equal_lists (CDR lis1) (CDR lis2)))
        (T NIL)
    )
)
```

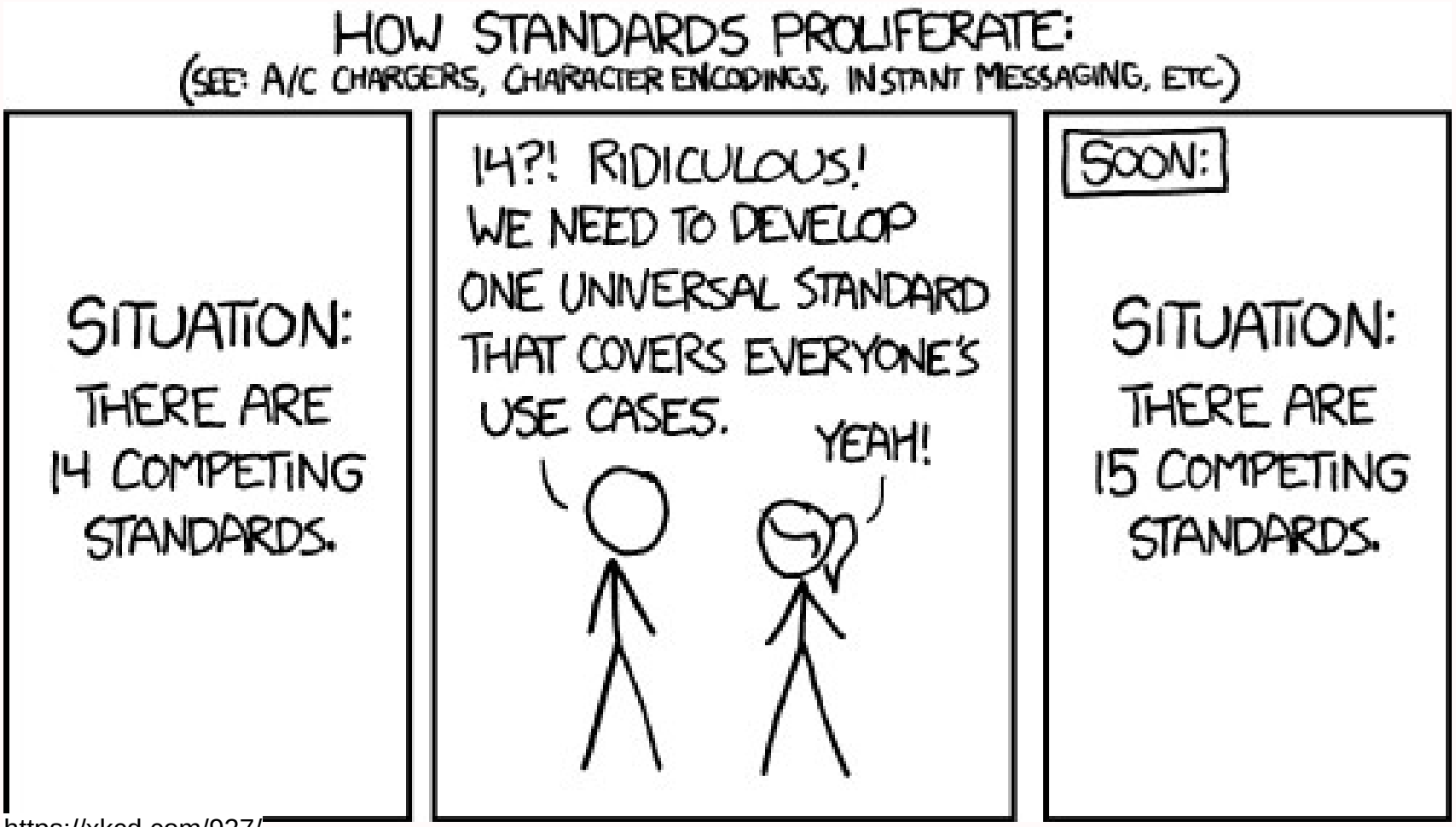
LISP (1958)

- Foi marcante na definição de diversas tecnologias
  - Primeira linguagem de programação com suporte adequado para recursão
  - Primeira linguagem com gerenciamento de memória automático
  - Primeira linguagem que incentiva o desenvolvimento em um paradigma diferente do imperativo
  - Primeira linguagem a suportar funções de alta ordem

ALGOL 58

- Fruto de um desejo de uma linguagem de programação universal
  - Novas linguagens de programação estavam aparecendo
    - FORTRAN em 1957
    - Outras menos notórias, como IT para o UNIVAC, MATH-MATIC e UNICODE
  - A proliferação de novas linguagens dificultava o compartilhamento de código
  - Algumas linguagens focavam em plataformas específicas

ALGOL 58



https://xkcd.com/927/

ALGOL 58

- Um comitê internacional inicial decidiu
  - A sintaxe deveria ser tão próxima quanto possível da notação matemática
  - Os programas fáceis de ler (a linguagem deveria ser intuitiva)
  - Uma linguagem que fosse adequada para programação e comunicação científica
  - Compilável
    - Tão eficiente quanto FORTRAN e Assembly

ALGOL 58

- O trabalho do comitê foi complicado
- Muitos concessões tiveram que ser feitas
  - Disputa entre separação de casas decimais: pontos ou vírgulas?
  - A sintaxe da atribuição deveria seguir a forma do Plankalkül ou do FORTRAN?
    - `expressão => variável` (sugestão europeia)
    - `variável := expressão` (americana)

ALGOL 58

- Ainda assim o comitê definiu diversos pontos--chave
  - Formalizou o conceito de tipo de dados
  - Definiu o conceito de comando composto
  - Identificadores poderiam ter número arbtirário de caracteres
  - Matrizes poderiam ter número arbitrário de dimensões
  - O primeiro índice dos vetores poderia ser especificado pelo programador

ALGOL 58

- Mas a linguagem não saiu do papel....

ALGOL 60

- Em 1959 houve grande debate em torno de ALGOL
- John Backus apresentou sua nova notação para linguagens
  - Posteriormente, essa notação ficou conhecida como Backus-Naur Form (BNF)
- O comitê chegou em acordos e publicou a especificação de ALGOL 60

ALGOL 60

- Foi extremamente influente
  - Introduziu o conceito de blocos
  - Passagem de parâmetros por valor e nome
  - Inventou if-then-else
  - Vetores dinâmicos de pilha
    - Tamanho máximo do vetor definido no momento em que uma função é chamada
  - Não era claro se permitia recursão
  - Tornou-se a primeira linguagem padrão *de facto* para descrever algoritmos

ALGOL 60

- Man or boy
  - Proposto por Donald Knuth
  - Objetivo: testar se uma implementação/linguagem suportava adequadamente recursão
  - Um bom exemplo de como ALGOL 60 era usado para divulgar algoritmos

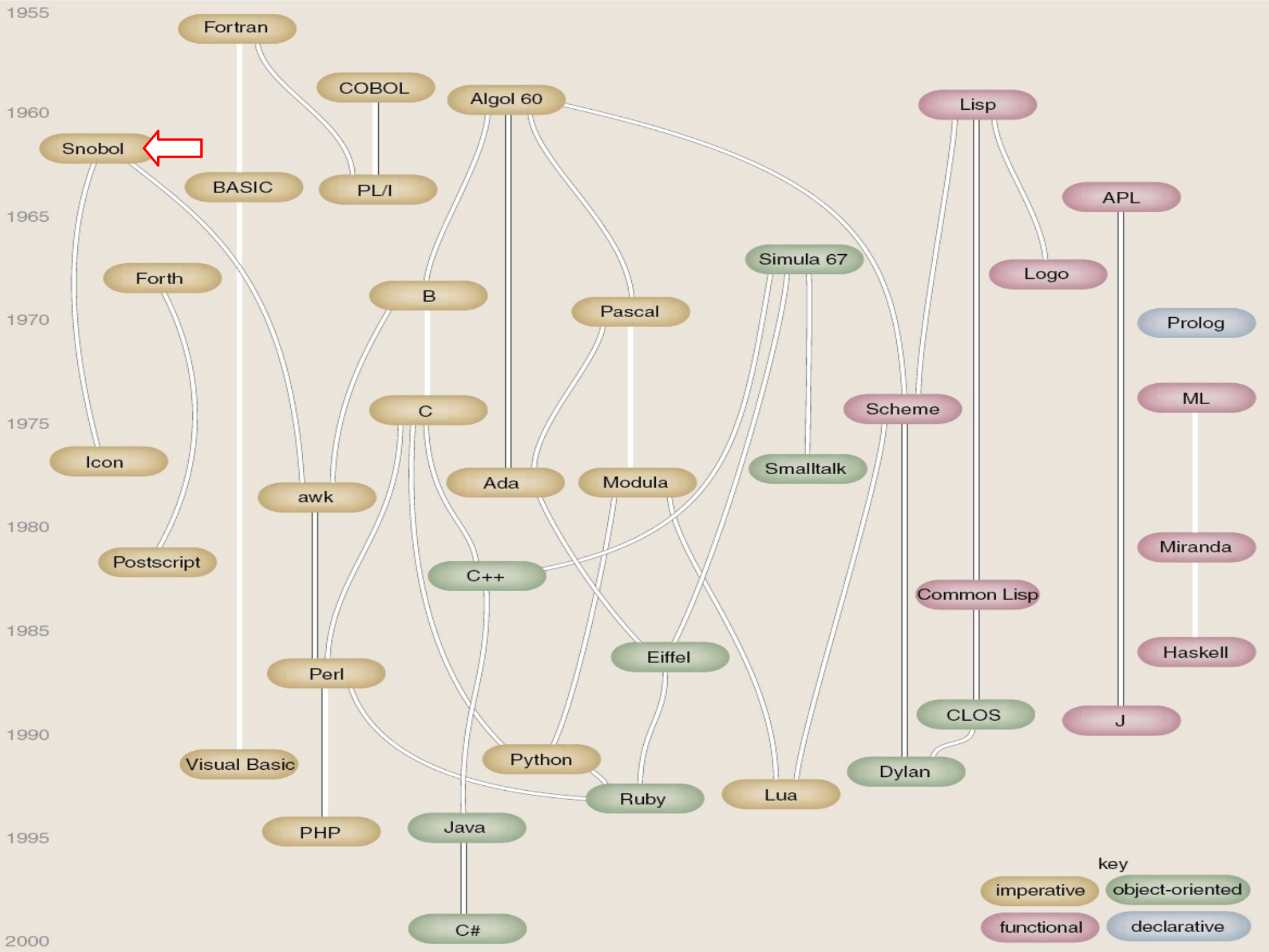
ALGOL 60

- Man or boy

```
begin
  real procedure A (k, x1, x2, x3, x4, x5);
  value k; integer k;
  real x1, x2, x3, x4, x5;
  begin
    real procedure B;
    begin k:= k - 1;
          B:= A := A (k, B, x1, x2, x3, x4)
    end;
    if k <= 0 then A:= x4 + x5 else B
  end;
  outreal (A (10, 1, -1, -1, 1, 0))
end
```

ALGOL 60

```
'BEGIN' 'COMMENT' Loops/Break - ALGOL60 - 18/06/2018;
  'INTEGER' SEED;
  'INTEGER' 'PROCEDURE' RANDOM(N);
  'VALUE' N; 'INTEGER' N;
  'BEGIN'
    SEED:=(SEED*19157+12347) '/' 21647;
    RANDOM:=SEED-(SEED '/' N)*N+1
  'END' RANDOM;
  'INTEGER' I,J,K;
  SYSACT(1,6,120);SYSACT(1,8,60);SYSACT(1,12,1);'COMMENT' open print;
  SEED:=31567;
  J:=0;
  'FOR' I:=1, I+1 'WHILE' I 'LESS' 100 'DO' 'BEGIN'
    J:=J+1;
    K:=RANDOM(20);
    OUTINTEGER(1,K);
    'IF' J=8 'THEN' 'BEGIN'
       SYSACT(1,14,1);  'COMMENT' skip line;
       J:=0
    'END';
    'IF' K=10 'THEN' 'GOTO' LAB
  'END';
LAB:
  SYSACT(1,14,1);  'COMMENT' skip line;
'END'
```


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Forth
Simula 67
Logo
B
Pascal
Prolog
C
Scheme
ML
Icon
Ada
Modula
Smalltalk
awk
Postscript
Miranda
C++
Common Lisp
Eiffel
Haskell
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

SNOBOL

- String Oriented and Symbolic Language
  - Linguagem para processamento de cadeias de caracteres
  - SNOBOL 4
    - Comandos são na forma

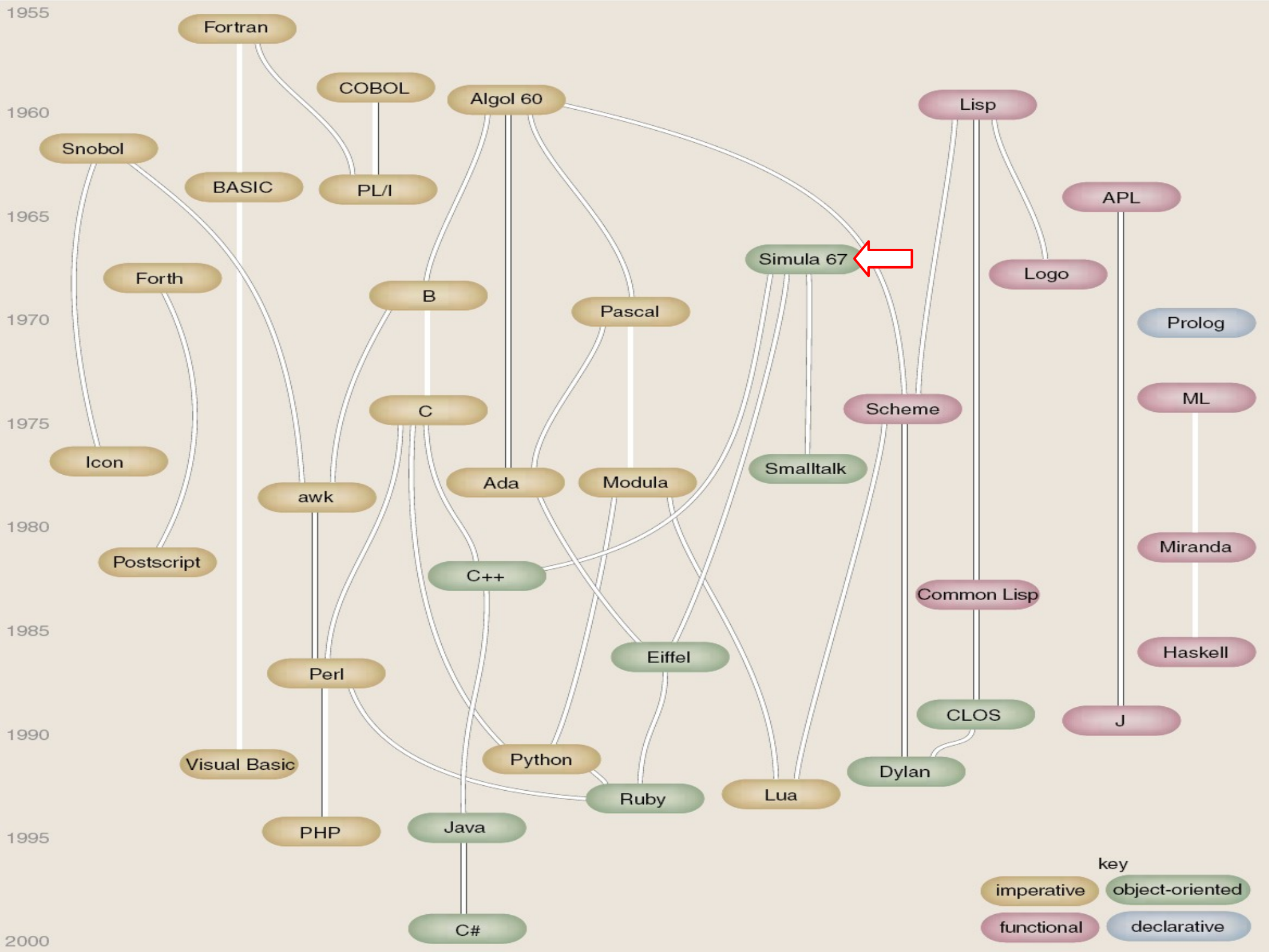
rótulo sujeito padrão = objeto : transferência

SNOBOL

```
        OUTPUT = "Hello world"
END
```

```
        OUTPUT = "What is your name?"
        Username = INPUT
        OUTPUT = "Thank you, " Username
END
```

```
        OUTPUT = "What is your name?"
        Username = INPUT
        Username "J"                                    :S(LOVE)
        Username "K"                                    :S(HATE)
MEH     OUTPUT = "Hi, " Username                        :(END)
LOVE    OUTPUT = "How nice to meet you, " Username      :(END)
HATE    OUTPUT = "Oh. It's you, " Username
END
```

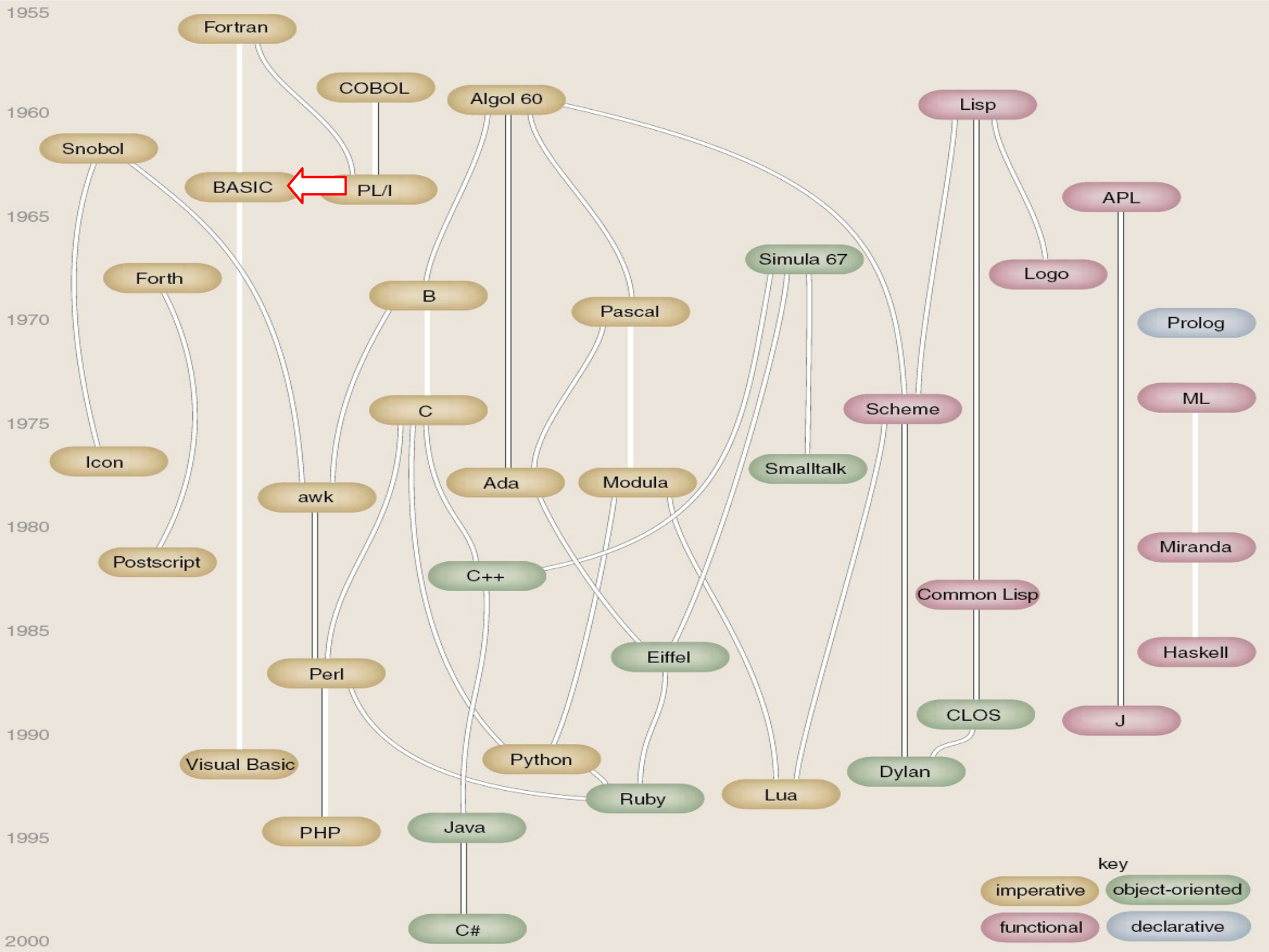

1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Simula 67
Logo
Forth
B
Pascal
Prolog
ML
C
Scheme
Icon
Smalltalk
Ada
Modula
awk
Postscript
Miranda
C++
Common Lisp
Haskell
Eiffel
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

Simula 67

- Uma extensão de ALGOL 60
- Introduziu o conceito de classes e objetos e deu origem à Programação Orientada a Objetos
- Introduziu o conceito de co-rotinas
  - Uma função pode ser interrompida e depois continuar sua execução do ponto em que parou
  - Útil para simulações

```
Begin
   Class Glyph;
      Virtual: Procedure print Is Procedure print;
   Begin
   End;

   Glyph Class Char (c);
      Character c;
   Begin
      Procedure print;
        OutChar(c);
   End;
```

```
   Glyph Class Line (elements);
      Ref (Glyph) Array elements;
   Begin
      Procedure print;
      Begin
         Integer i;
         For i:= 1 Step 1 Until UpperBound (elements, 1) Do
            elements (i).print;
         OutImage;
      End;
   End;

   Ref (Glyph) rg;
   Ref (Glyph) Array rgs (1 : 4);

   ! Main program;
   rgs (1):- New Char ('A');
   rgs (2):- New Char ('b');
   rgs (3):- New Char ('b');
   rgs (4):- New Char ('a');
   rg:- New Line (rgs);
   rg.print;
End;
```


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Forth
Simula 67
Logo
B
Pascal
Prolog
C
Scheme
ML
Icon
Smalltalk
Ada
Modula
awk
Postscript
Miranda
C++
Common Lisp
Eiffel
Haskell
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

BASIC (1964)

- Beginner's All-purpose Symbolic Instruction Code
  - Uma linguagem de programação para principiantes
  - Criada para alunos da Universidade de Dartmouth
    - Na época 75% eram de artes
    - Uma linguagem para não-engenheiros

BASIC (1964)

- Simples
  - Não tinha instruções de E/S
  - Variáveis só tinham uma letra
  - Todos os tipos eram de ponto flutuantes
    - Os criadores de BASIC acreditavam que seu público alvo não compreenderia a distinção entre inteiros e ponto flutuante

BASIC (1964)

- Ideias inovadoras
  - Primeira linguagem que não foi criada para um público especializado
  - Primeira linguagem a considerar que o tempo do usuário é mais importante que o da máquina
    - Os autores apostaram que os preços dos computadores cairia

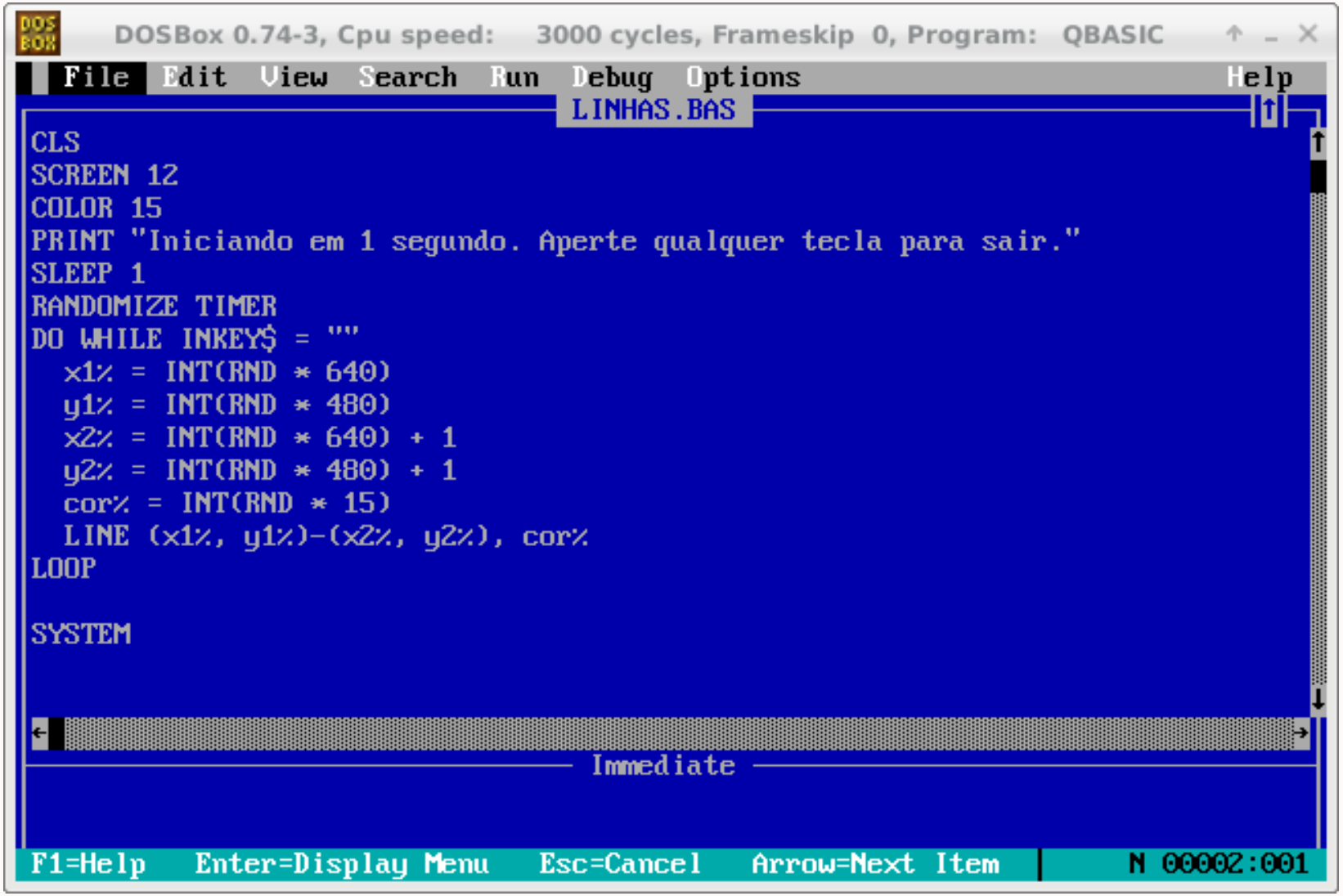
BASIC (1964)

- Muito popular nas décadas de 1970 e 1980
- Dialetos famosos
  - Commodore BASIC
  - Apple BASIC
  - Microsoft BASIC
    - GW-BASIC
    - QBasic

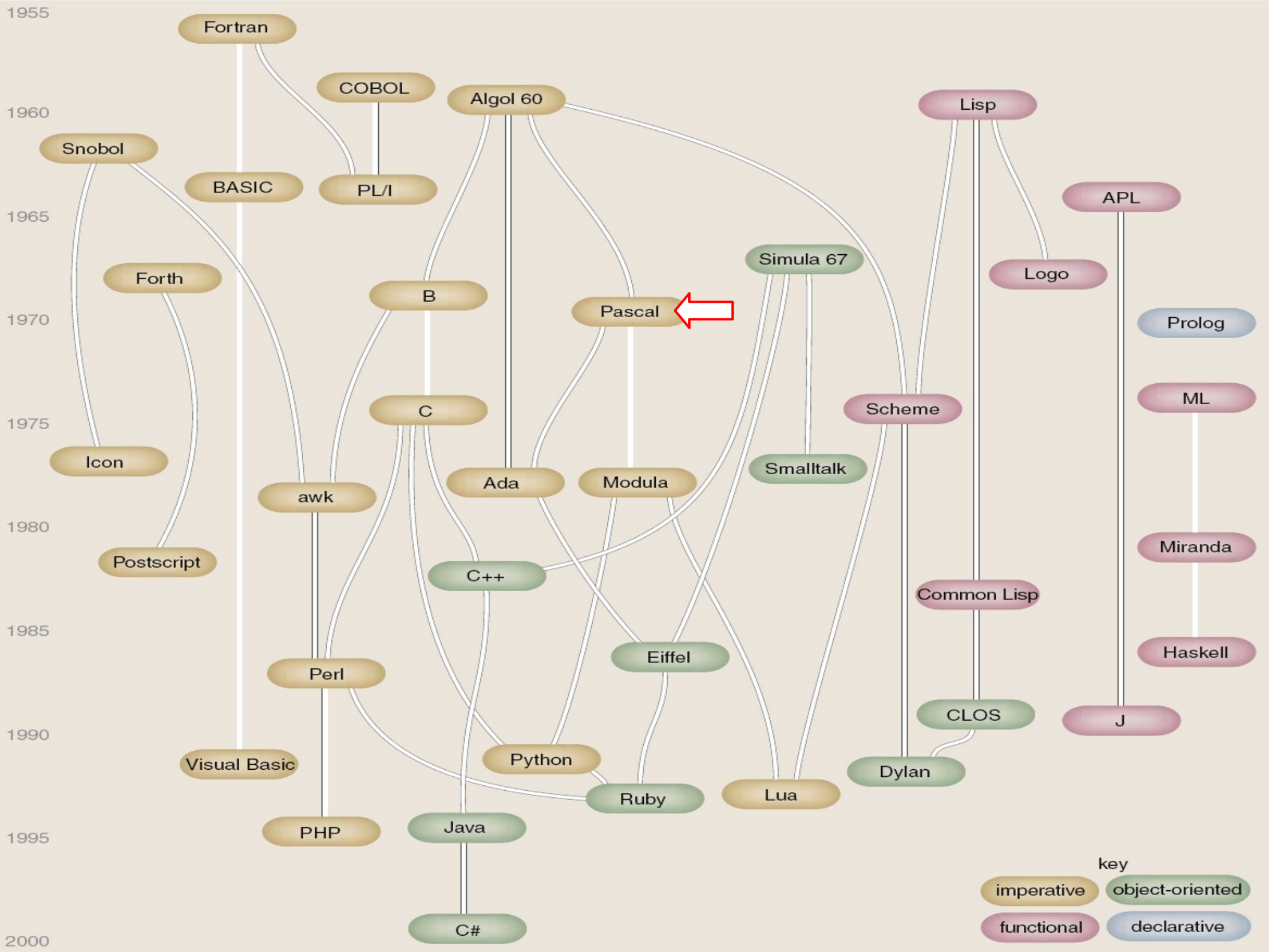
BASIC (1964)

```
10 INPUT "What is your name: ", U$
20 PRINT "Hello "; U$
30 INPUT "How many stars do you want: ", N
40 S$ = ""
50 FOR I = 1 TO N
60 S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Do you want more stars? ", A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFT$(A$, 1)
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30
130 PRINT "Goodbye "; U$
140 END
```

BASIC (1964)



Microsoft QBasic rodando no emulador DOSBox


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Forth
Simula 67
Logo
B
Pascal
Prolog
C
Scheme
ML
Icon
Smalltalk
Ada
Modula
awk
Postscript
Miranda
C++
Common Lisp
Eiffel
Haskell
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

Pascal (1970)

- Criada por Niklaus Wirth
  - Voltada para o ensino de programação
  - Simples por *design*
  - Extremamente popular e influente
    - Wirth distribuiu a especificação da linguagem para universidades em todo o mundo

Pascal (1970)

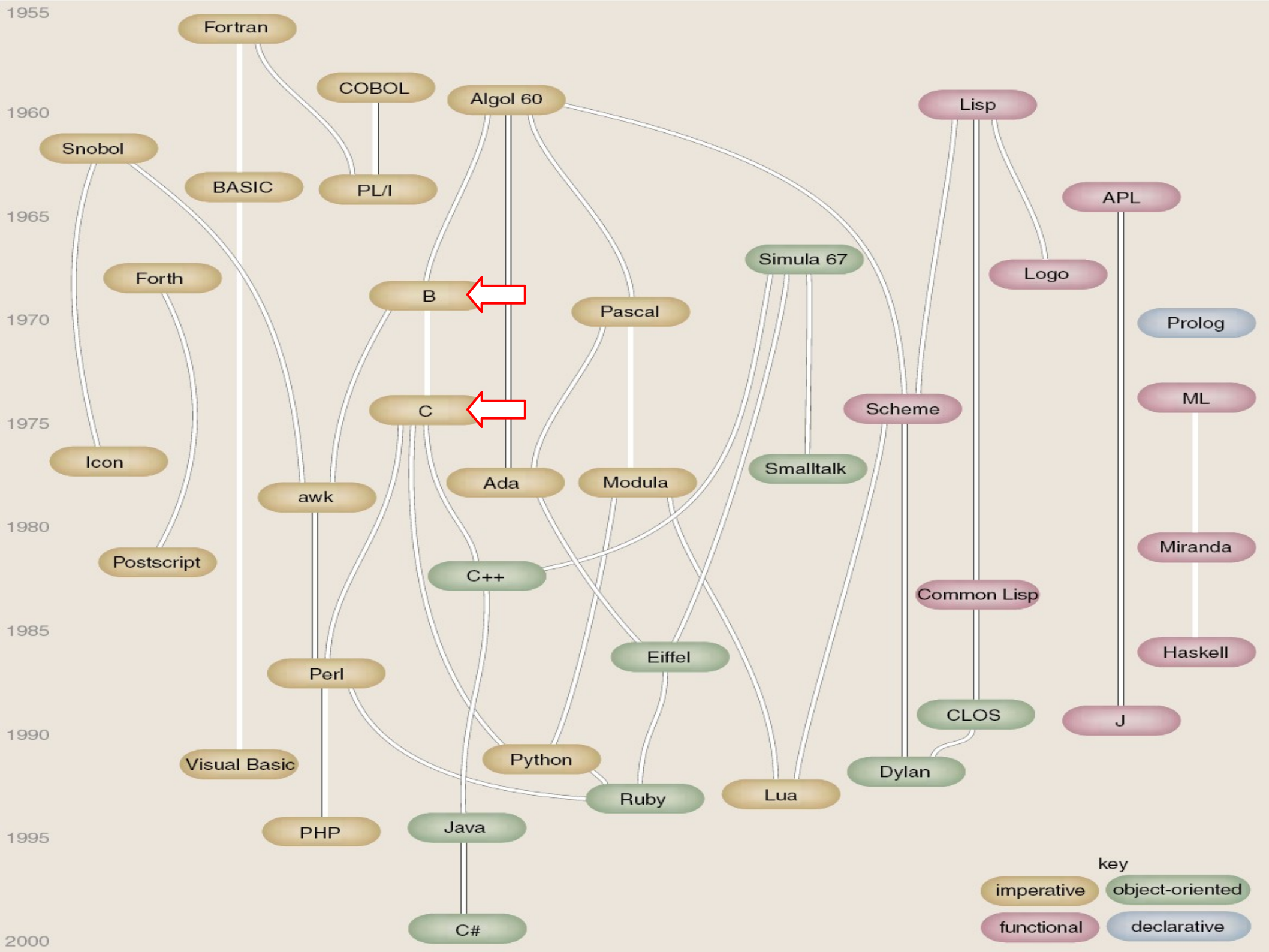
- Características de Pascal
  - Não permitia algumas funcionalidades importantes para o desenvolvimento de programas “sérios”
    - Uma função não pode receber um vetor de tamanho indeterminado
    - Não permitia compilação separada de subprogramas
  - Relativamente segura, especialmente se comparada com outras linguagens da época, como Fortran

Pascal (1970)

```
program BottlesOfBeer;

var
    i: integer;

begin
    for i := 99 downto 1 do
        if i <> 1 then
            begin
                writeln(i, ' bottles of beer on the wall');
                writeln(i, ' bottles of beer');
                writeln('Take one down, pass it around');
                if i = 2 then
                    writeln('One bottle of beer on the wall')
                else
                    writeln(i - 1, ' bottles of beer on the wall');
                writeln;
            end
        else
            begin
                writeln('One bottle of beer on the wall');
                writeln('One bottle of beer');
                writeln('Take one down, pass it around');
                writeln('No more bottles of beer on the wall');
            end
end.
```

Pasca

```
program sum(Input, Output);

type
    IntList = array [ 1 .. 99 ] of Integer;

var
    List: IntList;
    i: Integer;
    Sum: Integer;
begin
    for i := 1 to 99 do
        ReadLn(List[i]);
    Sum := 0;
    for i := 1 to 99 do
    begin
        Sum := Sum + List[i];
    end;
    WriteLn(Sum);
end.
```


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Forth
Simula 67
Logo
B
Pascal
Prolog
C
Scheme
ML
Icon
Smalltalk
Ada
Modula
awk
Postscript
Miranda
C++
Common Lisp
Haskell
Eiffel
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

B (1969)

- Motivada pelo desenvolvimento do sistema operacional UNIX
  - Criada por Ken Thompson no Bell Labs em 1970
  - Linguagem sem tipo de dados, extremamente baixo nível
    - Todos os dados são palavras da máquina
  - Uma nova linguagem baseada em B foi criada, incluindo tipos de dados
    - NB
    - Depois C

C (1972)

- Motivada pelo desenvolvimento do sistema operacional UNIX
  - Criada por Dennis Ritchie e Brian Kernighan em 1972
  - Baixo nível
  - Tipos de dados
    - Inteiros, pontos flutuantes e caracteres (byte)
    - Não especifica o tamanho de um byte
    - Não especifica o tamanho dos tipos inteiros

C (1972)

- Inicialmente, parâmetros de funções não tinham tipos definidos
- Também não havia verificação de número de argumentos passados para uma função
  - Uma função poderia receber um número diferente de argumentos do que seus parâmetros
    - **Argumento** (também chamado parâmetro real): expressão passada para a função
    - **Parâmetro** (ou parâmetro formal): nome da variável que vincula com o argumento

K&R C (1978)

- Criada por Brian Kernighan e Dennis Ritchie
  - Livro: "*The C Programming Language*"
- Recursos
  - Estabelece funções para E/S
  - Acrescenta os tipos de dados `(long int)` e `(unsigned int)`
  - Altera operadores de atribuição e operação da forma =+, =* etc. para +=, *= etc.
  - Funções sem retorno declaram o tipo do "retorno" como `void`

K&R C (1978)

```
soma(x, y)
    int x;
    int y;
{
    return x + y;
}

main()
{
    x, y;
    scanf(“%d %d”, &x, &y);
    printf(“%d\n”, soma(x, y);
}
```

K&R C (1978)

```
soma(vetor, tamanho)
    int *vetor;
    tamanho;
{
    register int i;
    s = 0;
    for (i = 0; i < tamanho; i++)
        s += vetor[i];
    return s;
}
```

ANSI C (1989)

- Primeira padronização da linguagem C
  - ANSI C (1989)
  - ISO C (1990) – às vezes chamada C90
- Inovações
  - O comitê ANSI definiu rigorosamente a linguagem, promovendo portabilidade
  - Adicionou protótipos de funções que definem tipos de parâmetros
  - Introduz qualificadores como `const` e `volatile`

ANSI C (1989)

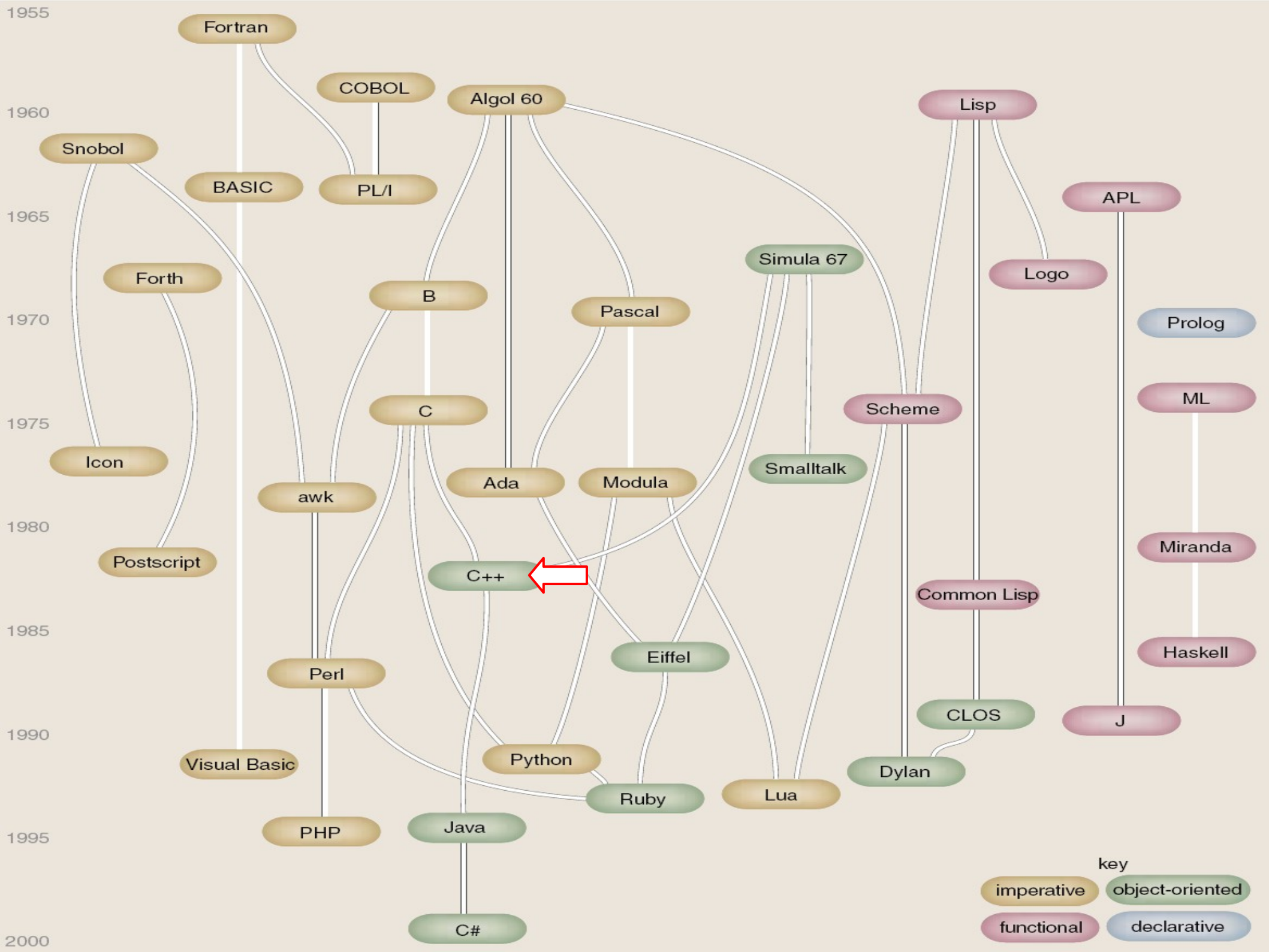
```
int soma(int x, int y)
{
    return x + y;
}

int main(void)
{
    x, y;
    scanf(“%d %d”, &x, &y);
    printf(“%d\n”, soma(x, y);
    return 0;
}
```

ANSI C (1989)

```
int printf(const char *s, ...);
int scanf(const char *s, ...);

int soma(int x, int y)
{
    return x + y;
}

int main(void)
{
    int x, y;
    scanf(“%d %d”, &x, &y);
    printf(“%d\n”, soma(x, y);
    return 0;
}
```

ANSI C (1989)

```c
#include <stdio.h>

int soma(int x, int y)
{
    return x + y;
}

int main(void)
{
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d\n", soma(x, y);
    return 0;
}
```


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Simula 67
Logo
Forth
B
Pascal
Prolog
ML
C
Scheme
Icon
Smalltalk
Ada
Modula
awk
Postscript
Miranda
C++
Common Lisp
Eiffel
Haskell
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

C++ (1985)

- Inicialmente uma extensão da lingagem C criada por Bjarne Stroustrup
  - C com um incremento (orientação a objetos)
- O primeiro compilador de C++ reescrevia o código em C
- As classes eram “estruturas com funções”

C++ (1985)

```
struct Hello {
    void sayHello() {
        printf("Hello, world!\n");
    }
};

int main()
{
    Hello h();
    h.sayHello();
    return 0;
}
```

C++98

- Padrão ISO/IEC 1482:1998
  - Espaços de nomes
  - Incorporação da STL (*Standard Template* Library) como componente oficial da linguagem
    - *Templates* (gabaritos) já existiam como extensão da linguagem
    - A STL foi criada por desenvolvedores da HP no início da década de 1990

C++98

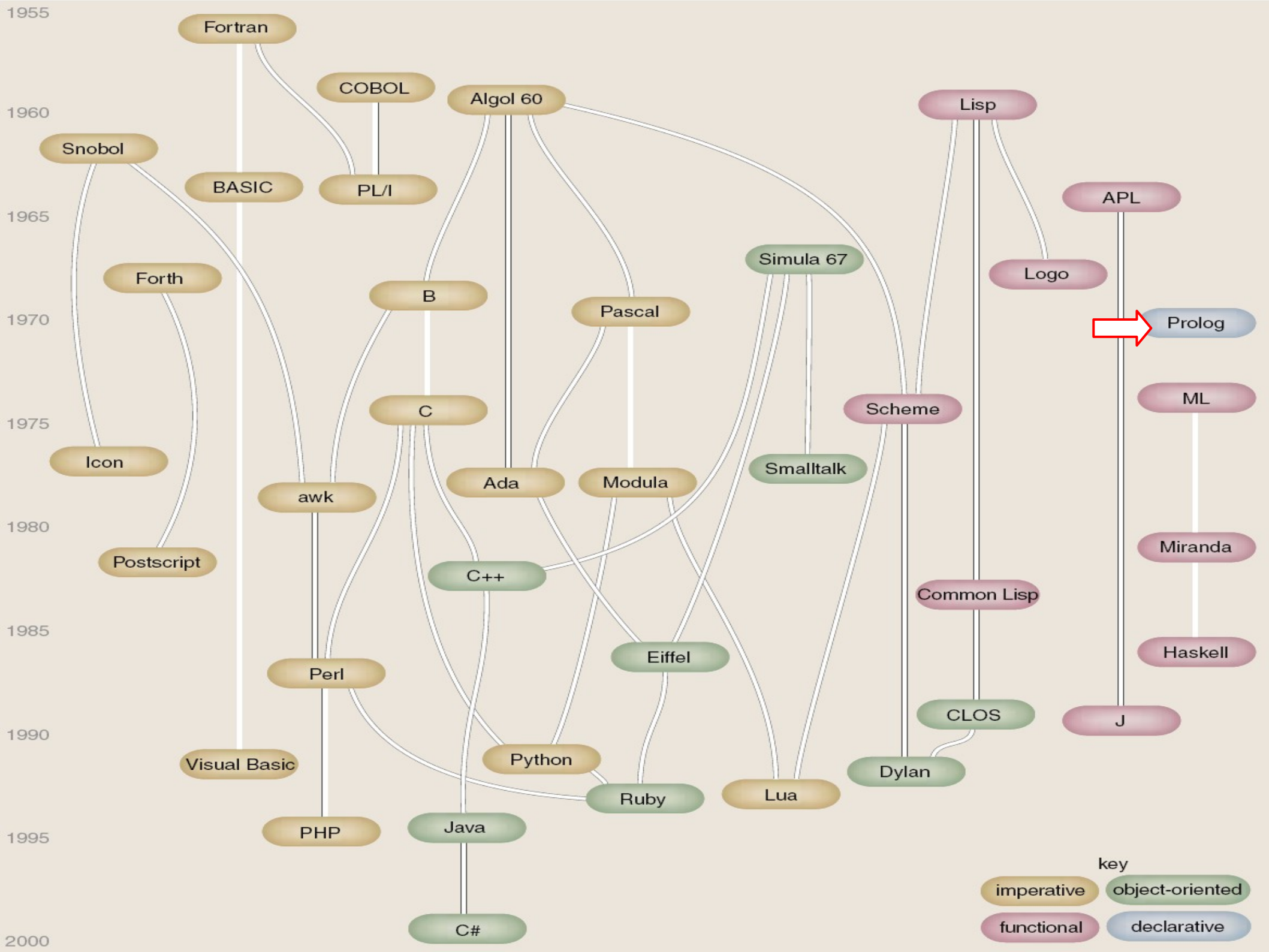
```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <utility>

int main() {
  std::vector<std::pair<int, std::string> > pq;
  pq.push_back(std::make_pair(3, "Clear drains"));
  pq.push_back(std::make_pair(4, "Feed cat"));
  pq.push_back(std::make_pair(5, "Make tea"));
  pq.push_back(std::make_pair(1, "Solve RC tasks"));

  // heapify
  std::make_heap(pq.begin(), pq.end());

  // enqueue
  pq.push_back(std::make_pair(2, "Tax return"));
  std::push_heap(pq.begin(), pq.end());

  while (!pq.empty()) {
    // peek
    std::cout << pq[0].first << ", " << pq[0].second << std::endl;
    // dequeue
    std::pop_heap(pq.begin(), pq.end());
    pq.pop_back();
  }

  return 0;
}
```

C++11

- Padrão ISO/IEC 14822:2011
  - Conhecida anteriormente como C++0x
  - Modifica muitos recursos fundamentais da linguagem
  - Introduz inferência de tipos
    - `auto variavel = 42;`
    - `auto *outraVariavel = &variavel;`
  - Expressões *lambda* para programação funcional

C++11

```cpp
#include <iostream>
#include <algorithm>

void display(int a)
{
    std::cout<<a<<" ";
}
int main() {
    int arr[] = { 1, 2, 3, 4, 5 };

    std::for_each(arr, arr + sizeof(arr) / sizeof(int), &display);

    std::cout << std::endl;

}
```


1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Fortran
COBOL
Algol 60
Lisp
Snobol
BASIC
PL/I
APL
Simula 67
Forth
Logo
B
Pascal
Prolog
C
ML
Scheme
Icon
Smalltalk
Ada
Modula
awk
Miranda
Postscript
C++
Common Lisp
Haskell
Eiffel
Perl
CLOS
J
Visual Basic
Python
Dylan
Ruby
Lua
PHP
Java
C#
key
imperative
object-oriented
functional
declarative

Prolog

- Fruto de necessidades de IA para resolver problemas de cálculo de predicados
  - Programas estabelecem fatos
  - Descrevem o domínio de discurso
  - Estabelecem proposições
  - Estabelecem argumentos
  - Um programa em Prolog tem que encontrar conjuntos de variáveis para satisfazer os argumentos

Prolog

- Comandos factuais descrevem uma base de dados (domínio de discurso)

```
mae(maria, joaozinho).
pai(jose, joaozinho).
```

- Um programa em Prolog é uma consulta à base de dados

```
mae(maria, jose).
```

Prolog

- Letras maiúsculas indicam variáveis

```
mae(maria, joaozinho).
pai(jose, joaozinho).
```

- Consulta:

```
mae(maria, X).
```

Prolog

- Regras podem ser utilizadas para escrever proposições quantificadas

```
mae(maria, joaozinho).
pai(jose, joazinho).
pai(jorge, jose).

avo(X, Y) :- pai(X, Z), pai(Z, Y).
```

$avo(x, y) \rightarrow (\exists z)[\, pai(x, z) \wedge pai(z, y)\,]$

Prolog

The Ackermann function is usually defined as follows:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

```
ack(0, N, Ans) :- Ans is N+1.
ack(M, 0, Ans) :- M>0, X is M-1, ack(X, 1, Ans).
ack(M, N, Ans) :- M>0, N>0, X is M-1, Y is N-1, ack(M, Y, Ans2), ack(X, Ans2, Ans).
```