C11

ISO/IEC 9899:2011

07.11.2013, Daniel Martens

Einführung

in C

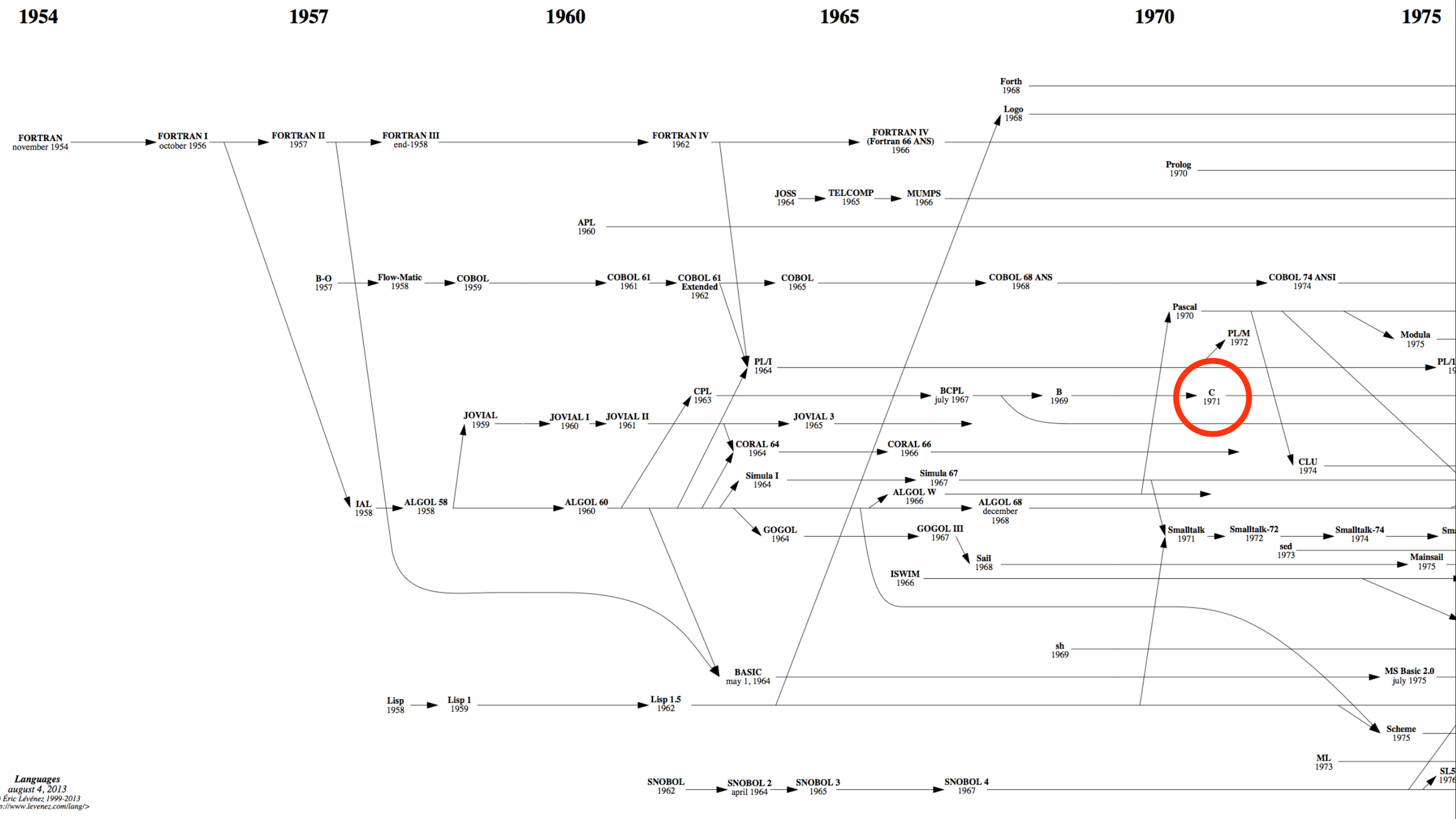
Hello World!

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   printf("Hello world!\n");

   return 0;
}
```

Entstehung

- 1969 - 1973 in den AT&T Bell Labs von Dennis Ritchie entwickelt
- Imperative Programmiersprache
- Beeinflusst von B, BCPL und ALGOL 68

Entstehung



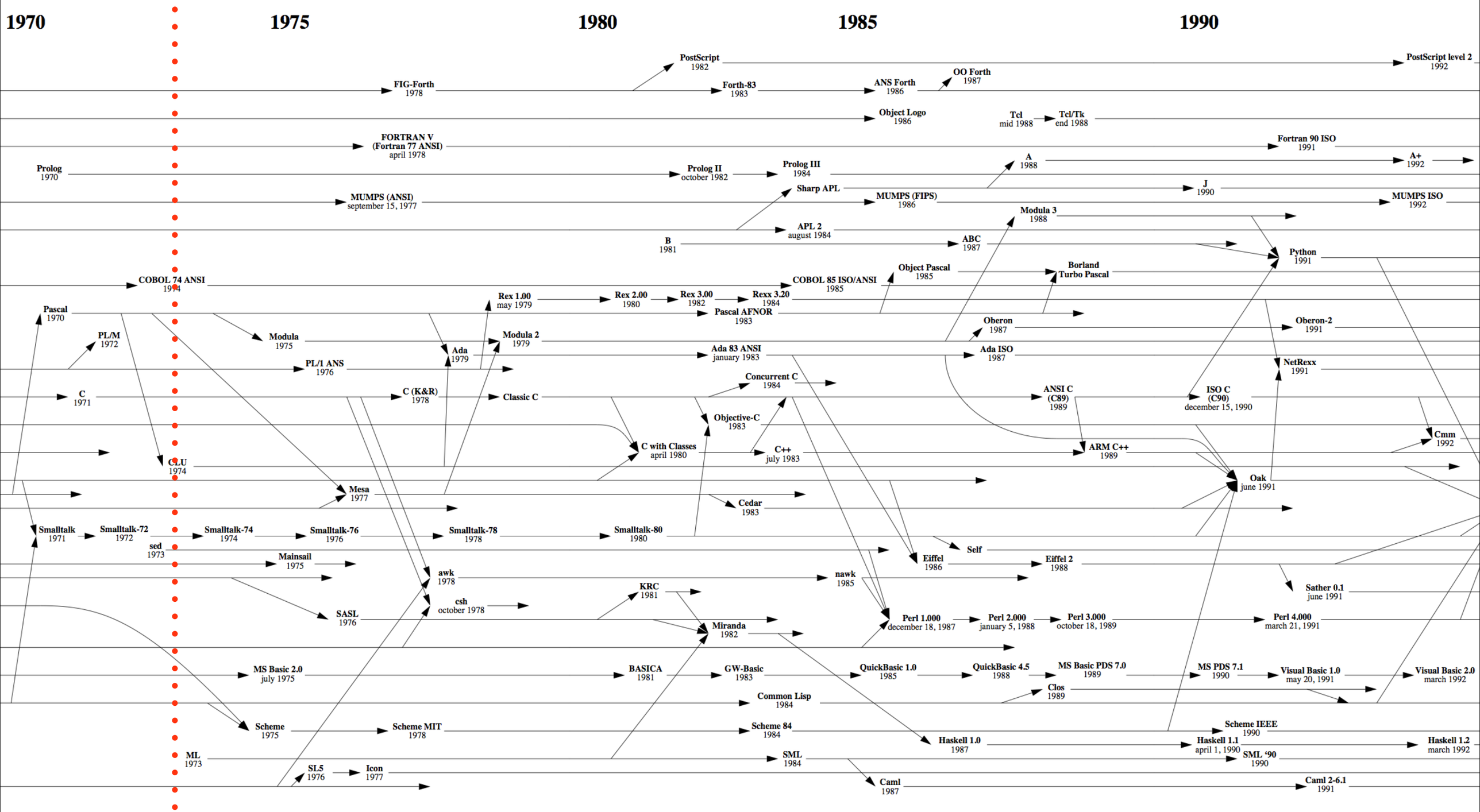
Languages
august 4, 2013

<http://www.levenez.com/lang/>

Evolution

- 1973 ist C so weit ausgereift, dass Unix-Kernel, der PDP-11, in C neu geschrieben werden kann
- Erster C-Compiler ebenfalls von Dennis Ritchie entwickelt

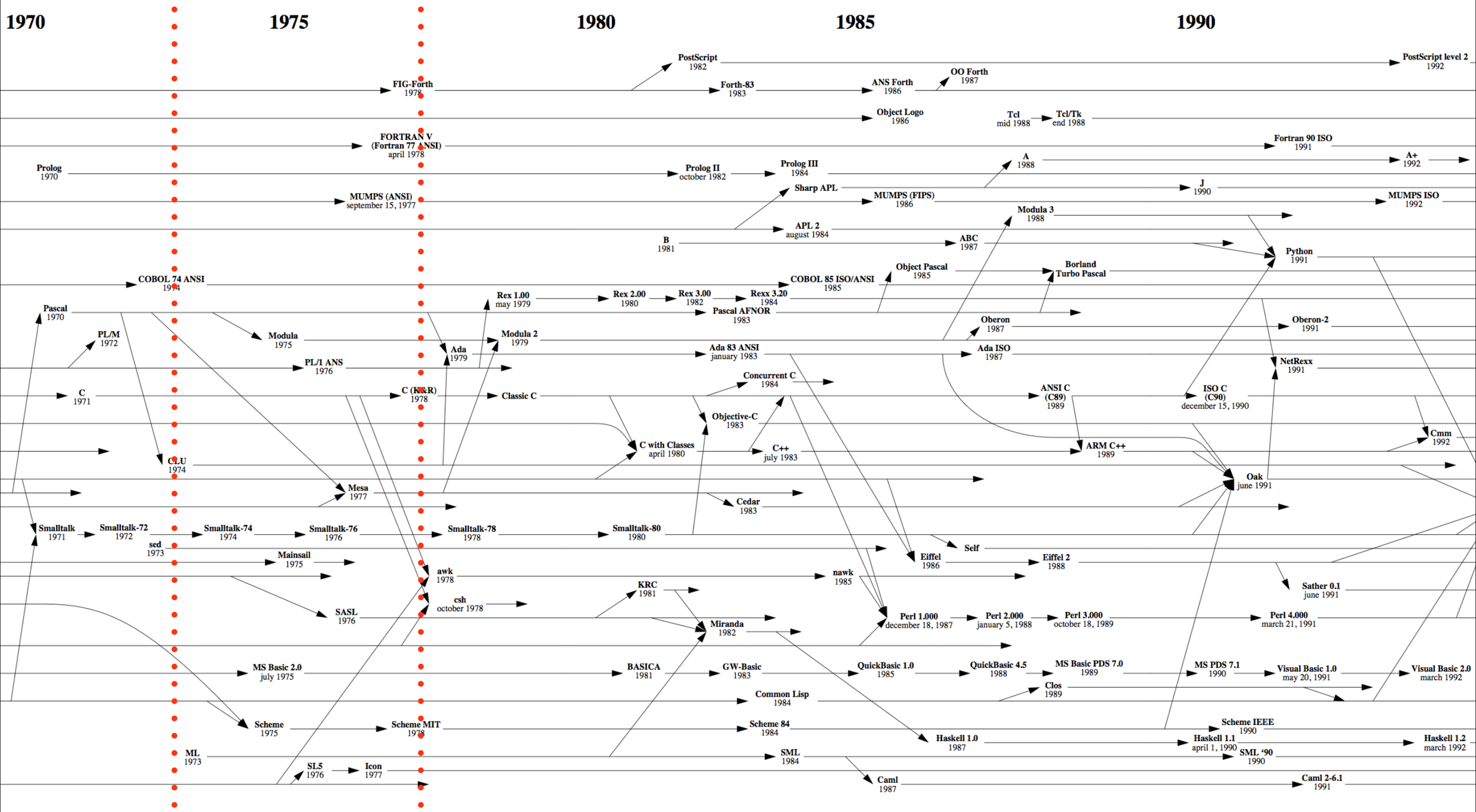
Evolution


1970
1975
1980
1985
1990
Prolog
1970
COBOL 74 ANSI
1974
Pascal
1970
PL/M
1972
C
1971
Smalltalk
1971
Smalltalk-72
1972
sed
1973
ML
1973
FIG-Forth
1978
FORTRAN V
(Fortran 77 ANSI)
april 1978
MUMPS (ANSI)
september 15, 1977
Modula
1975
PL/1 ANS
1976
C (K&R)
1978
CLU
1974
Mesa
1977
Smalltalk-74
1974
Mainsail
1975
SASL
1976
MS Basic 2.0
july 1975
Scheme
1975
Scheme MIT
1978
SL5
1976
Icon
1977
Smalltalk-76
1976
awk
1978
csh
october 1978
Rex 1.00
may 1979
Ada
1979
Modula 2
1979
Classic C
Smalltalk-78
1978
PostScript
1982
Forth-83
1983
Prolog II
october 1982
B
1981
Rex 2.00
1980
Rex 3.00
1982
Pascal AFNOR
1983
Ada 83 ANSI
january 1983
C with Classes
april 1980
Smalltalk-80
1980
KRC
1981
Miranda
1982
BASICA
1981
GW-Basic
1983
Common Lisp
1984
Scheme 84
1984
SML
1984
Rexx 3.20
1984
Concurrent C
1984
Objective-C
1983
C++
july 1983
Cedar
1983
Prolog III
1984
Sharp APL
APL 2
august 1984
COBOL 85 ISO/ANSI
1985
Object Pascal
1985
nawk
1985
QuickBasic 1.0
1985
Haskell 1.0
1987
Caml
1987
Object Logo
1986
ANS Forth
1986
MUMPS (FIPS)
1986
Eiffel
1986
Perl 1.000
december 18, 1987
OO Forth
1987
ABC
1987
Oberon
1987
Ada ISO
1987
ANSI C
(C89)
1989
ARM C++
1989
QuickBasic 4.5
1988
Tcl
mid 1988
Tcl/Tk
end 1988
A
1988
Modula 3
1988
Borland
Turbo Pascal
MS Basic PDS 7.0
1989
Clos
1989
Perl 2.000
january 5, 1988
J
1990
Perl 3.000
october 18, 1989
ISO C
(C90)
december 15, 1990
Oak
june 1991
Eiffel 2
1988
Self
Fortran 90 ISO
1991
A+
1992
MUMPS ISO
1992
Python
1991
Oberon-2
1991
NetRexx
1991
Cmm
1992
Sather 0.1
june 1991
Perl 4.000
march 21, 1991
MS PDS 7.1
1990
Visual Basic 1.0
may 20, 1991
Visual Basic 2.0
march 1992
Scheme IEEE
1990
Haskell 1.1
april 1, 1990
SML '90
1990
Haskell 1.2
march 1992
Caml 2-6.1
1991
PostScript level 2
1992

Evolution

- 1973 ist C so weit ausgereift, dass Unix-Kernel, der PDP-11, in C neu geschrieben werden kann
- Erster C-Compiler ebenfalls von Dennis Ritchie entwickelt
- 1978 Brian W. Kernighan und Dennis Ritchie veröffentlichen erste Spezifikation („The C Programming Language“)

Evolution


1970
1975
1980
1985
1990
Prolog
1970
FIG-Forth
1978
FORTRAN V
(Fortran 77 ANSI)
april 1978
MUMPS (ANSI)
september 15, 1977
COBOL 74 ANSI
1974
Pascal
1970
PL/M
1972
Modula
1975
PL/1 ANS
1976
C
1971
C (K&R)
1978
CLU
1974
Mesa
1977
Smalltalk
1971
Smalltalk-72
1972
Smalltalk-74
1974
Smalltalk-76
1976
sed
1973
Mainsail
1975
SASL
1976
MS Basic 2.0
july 1975
Scheme
1975
ML
1973
SL5
1976
Icon
1977
awk
1978
csh
october 1978
Scheme MIT
1978
Rex 1.00
may 1979
Modula 2
1979
Ada
1979
Smalltalk-78
1978
PostScript
1982
Forth-83
1983
Prolog II
october 1982
B
1981
Sharp APL
APL 2
august 1984
COBOL 85 ISO/ANSI
1985
Rex 2.00
1980
Rex 3.00
1982
Rexx 3.20
1984
Pascal AFNOR
1983
Ada 83 ANSI
january 1983
Concurrent C
1984
Objective-C
1983
C with Classes
april 1980
C++
july 1983
Cedar
1983
Smalltalk-80
1980
KRC
1981
Miranda
1982
BASICA
1981
GW-Basic
1983
Common Lisp
1984
Scheme 84
1984
SML
1984
Caml
1987
nawk
1985
ANS Forth
1986
Object Logo
1986
Prolog III
1984
MUMPS (FIPS)
1986
ABC
1987
Object Pascal
1985
Oberon
1987
Ada ISO
1987
OO Forth
1987
Tcl
mid 1988
Tcl/Tk
end 1988
A
1988
Modula 3
1988
Borland
Turbo Pascal
ANSI C
(C89)
1989
ARM C++
1989
Eiffel
1986
Perl 1.000
december 18, 1987
Perl 2.000
january 5, 1988
Perl 3.000
october 18, 1989
QuickBasic 1.0
1985
QuickBasic 4.5
1988
MS Basic PDS 7.0
1989
Clos
1989
Haskell 1.0
1987
PostScript level 2
1992
Fortran 90 ISO
1991
A+
1992
J
1990
MUMPS ISO
1992
Python
1991
Oberon-2
1991
NetRexx
1991
ISO C
(C90)
december 15, 1990
Cmm
1992
Oak
june 1991
Self
Eiffel 2
1988
Sather 0.1
june 1991
Perl 4.000
march 21, 1991
MS PDS 7.1
1990
Visual Basic 1.0
may 20, 1991
Visual Basic 2.0
march 1992
Scheme IEEE
1990
Haskell 1.1
april 1, 1990
SML '90
1990
Haskell 1.2
march 1992
Caml 2-6.1
1991

Features

- Wenige Schlüsselwörter
- Standard Bibliothek
- Kompiliert zu nativem Code
- Systemnah, plattformunabhängig
- Präprozessor
- Internationaler Standard
- Compiler für nahezu jede Plattform

Anwendungsgebiete

- Systemprogrammierung
  - Betriebssysteme
- Mikrocontroller
- Eingebettete Prozessoren
- DSP Prozessoren
- Als Interpreter anderer Programmiersprachen (z.B. Java Virtual Machine)
- als Zwischencode einiger Programmiersprachen (z.B. Vala)

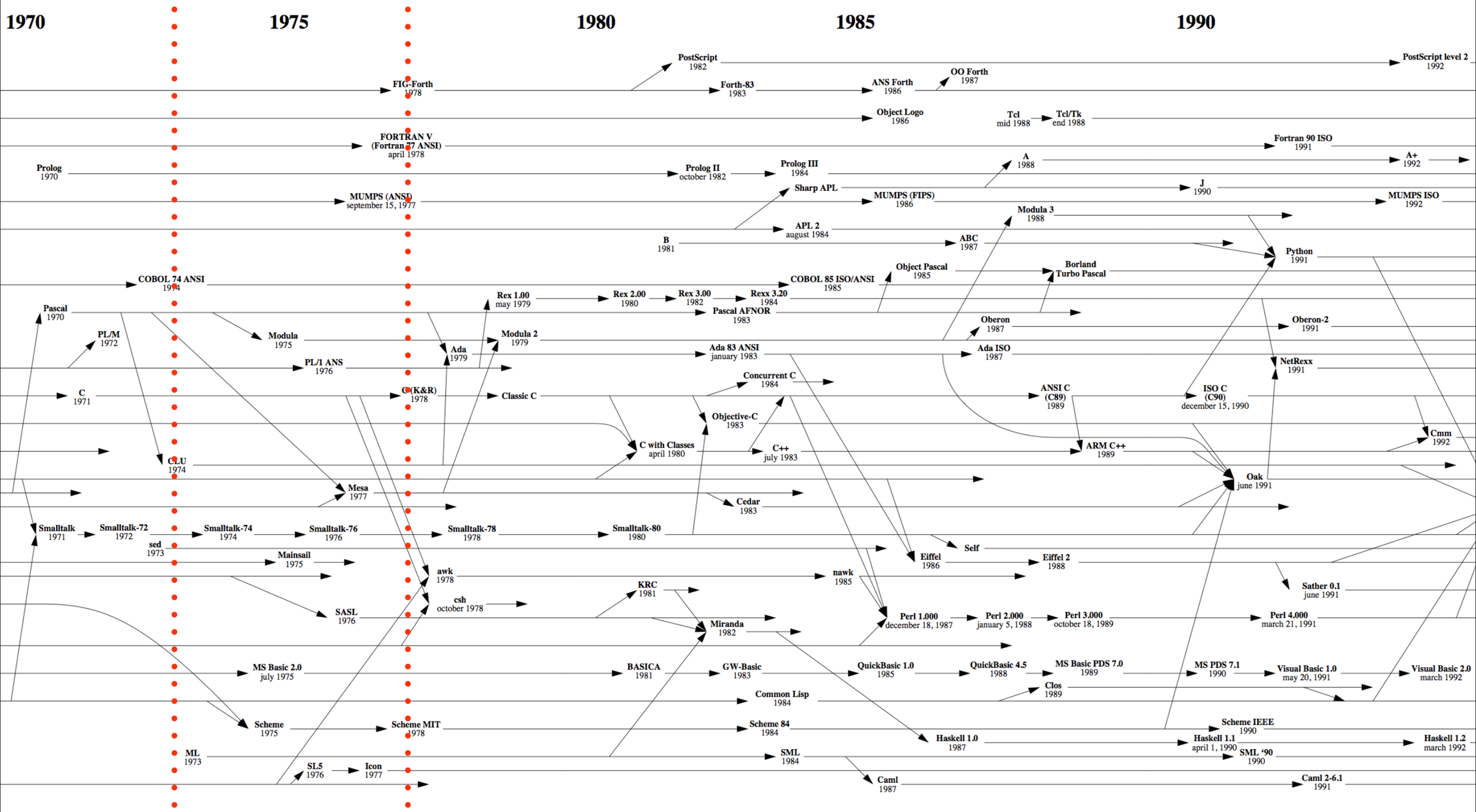
Warum C?

- Portierbarkeit
- Geschwindigkeit
- Größe
- Low-Level-Access/Hardware ansprechen
- Keine Laufzeitumgebung erforderlich
- Wird bereits lange in der Entwicklung von Betriebssystemen eingesetzt

C und andere

- Abwandlungen
  - C++, C#, Objective-C
- Auswirkungen auf
  - Java, Perl, D, PHP, Vala
- Nachteile
  - Exceptions
  - Range checking
  - Garbage collection
  - OOP

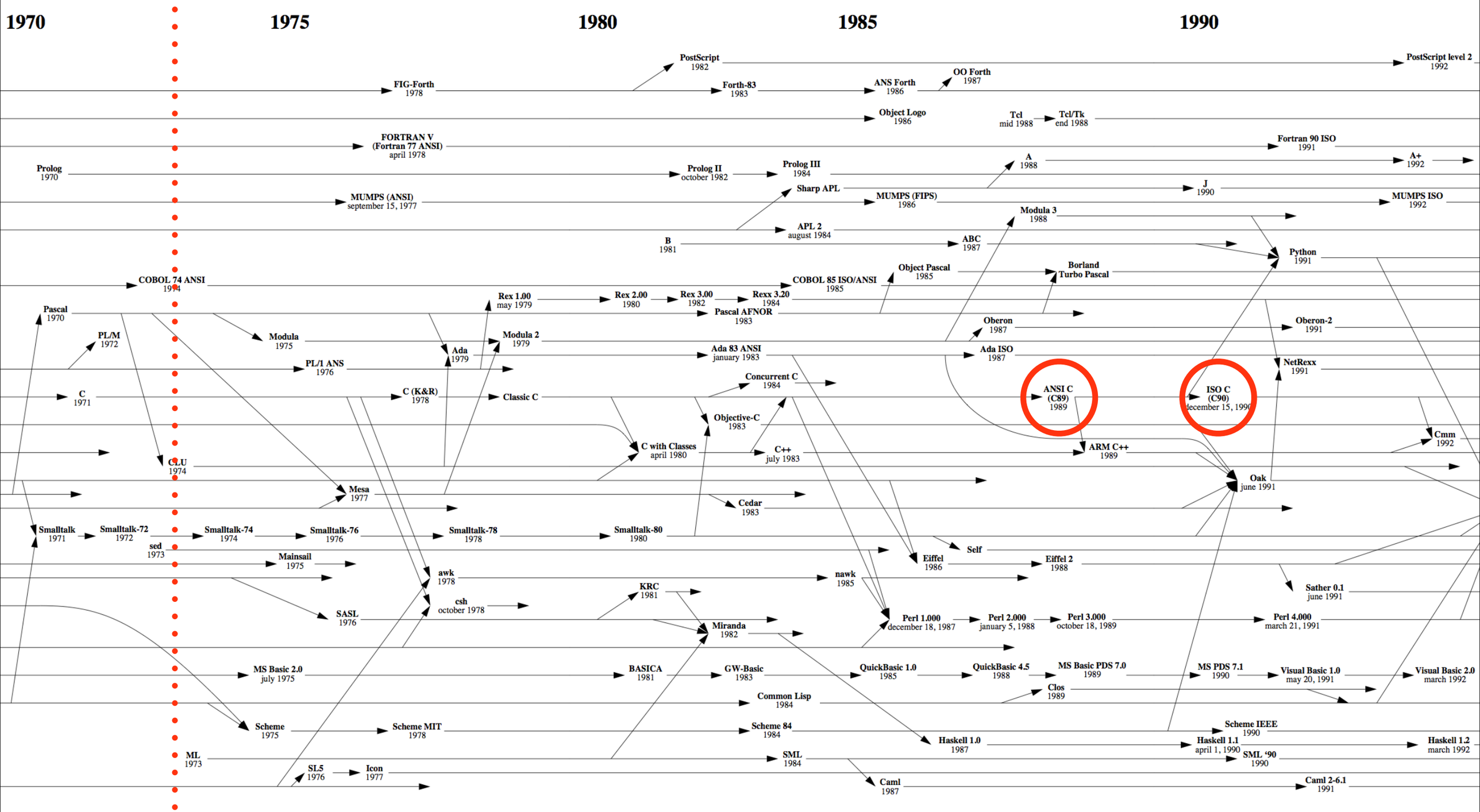
Evolution


1970
1975
1980
1985
1990
PostScript
1982
PostScript level 2
1992
FIG-Forth
1978
Forth-83
1983
ANS Forth
1986
OO Forth
1987
Object Logo
1986
Tcl
mid 1988
Tcl/Tk
end 1988
FORTRAN V
(Fortran 77 ANSI)
april 1978
Fortran 90 ISO
1991
Prolog
1970
Prolog II
october 1982
Prolog III
1984
A
1988
A+
1992
Sharp APL
J
1990
MUMPS (ANSI)
september 15, 1977
MUMPS (FIPS)
1986
MUMPS ISO
1992
APL 2
august 1984
Modula 3
1988
B
1981
ABC
1987
Python
1991
COBOL 74 ANSI
1974
COBOL 85 ISO/ANSI
1985
Object Pascal
1985
Borland
Turbo Pascal
Rex 1.00
may 1979
Rex 2.00
1980
Rex 3.00
1982
Rexx 3.20
1984
Pascal
1970
Pascal AFNOR
1983
Oberon
1987
Oberon-2
1991
Modula
1975
Modula 2
1979
PL/M
1972
Ada
1979
Ada 83 ANSI
january 1983
Ada ISO
1987
NetRexx
1991
PL/1 ANS
1976
Concurrent C
1984
C
1971
C(K&R)
1978
Classic C
ANSI C
(C89)
1989
ISO C
(C90)
december 15, 1990
Objective-C
1983
C with Classes
april 1980
C++
july 1983
ARM C++
1989
Cmm
1992
CLU
1974
Oak
june 1991
Mesa
1977
Cedar
1983
Smalltalk
1971
Smalltalk-72
1972
Smalltalk-74
1974
Smalltalk-76
1976
Smalltalk-78
1978
Smalltalk-80
1980
Self
sed
1973
Eiffel
1986
Eiffel 2
1988
Mainsail
1975
awk
1978
nawk
1985
KRC
1981
Sather 0.1
june 1991
csh
october 1978
SASL
1976
Perl 1.000
december 18, 1987
Perl 2.000
january 5, 1988
Perl 3.000
october 18, 1989
Perl 4.000
march 21, 1991
Miranda
1982
MS Basic 2.0
july 1975
BASICA
1981
GW-Basic
1983
QuickBasic 1.0
1985
QuickBasic 4.5
1988
MS Basic PDS 7.0
1989
MS PDS 7.1
1990
Visual Basic 1.0
may 20, 1991
Visual Basic 2.0
march 1992
Clos
1989
Common Lisp
1984
Scheme
1975
Scheme MIT
1978
Scheme 84
1984
Scheme IEEE
1990
Haskell 1.0
1987
Haskell 1.1
april 1, 1990
Haskell 1.2
march 1992
ML
1973
SML
1984
SML '90
1990
SL5
1976
Icon
1977
Caml
1987
Caml 2-6.1
1991

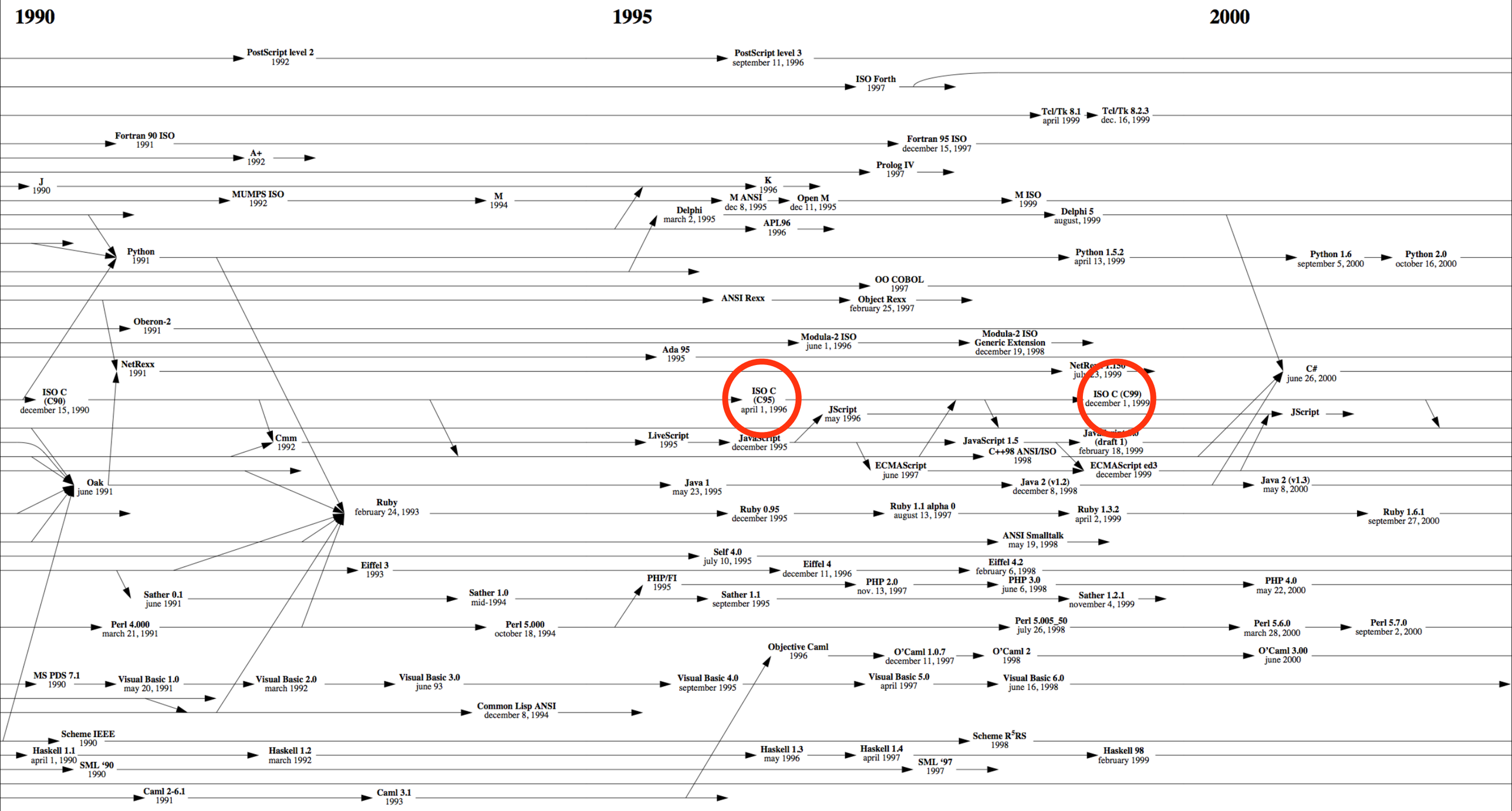
Evolution

- 1973 ist C so weit ausgereift, dass Unix-Kernel, der PDP-11, in C neu geschrieben werden kann
- Erster C-Compiler ebenfalls von Dennis Ritchie entwickelt
- 1978 Brian W. Kernighan und Dennis Ritchie veröffentlichen erste Spezifikation („The C Programming Language“)
- Standards
  - C89/C90, C95, C99, C11

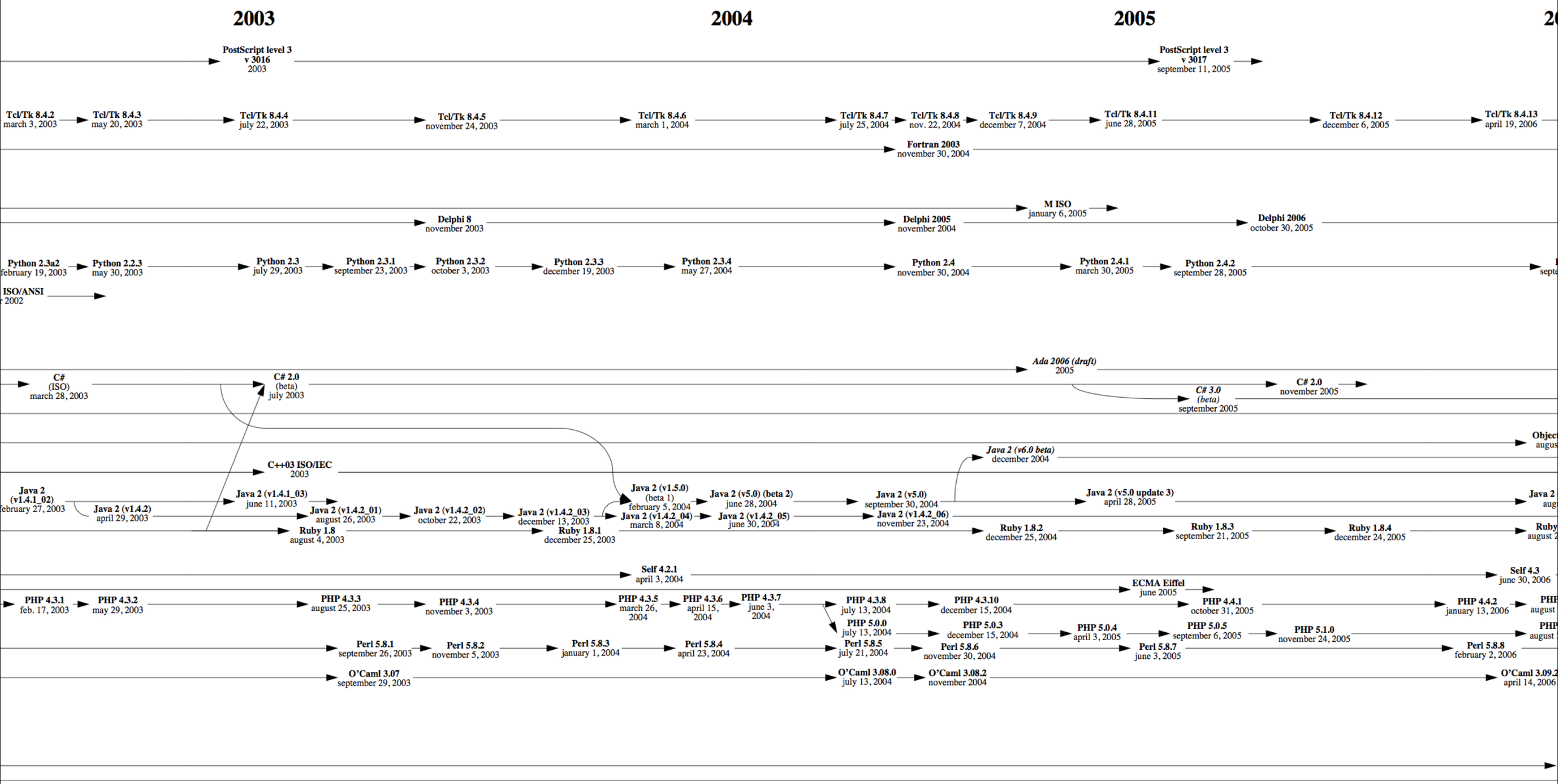
Evolution


1970
1975
1980
1985
1990
Prolog
1970
FIG-Forth
1978
FORTRAN V
(Fortran 77 ANSI)
april 1978
MUMPS (ANSI)
september 15, 1977
COBOL 74 ANSI
1974
Pascal
1970
PL/M
1972
Modula
1975
PL/1 ANS
1976
C
1971
C (K&R)
1978
CLU
1974
Mesa
1977
Smalltalk
1971
Smalltalk-72
1972
Smalltalk-74
1974
Smalltalk-76
1976
sed
1973
Mainsail
1975
SASL
1976
MS Basic 2.0
july 1975
Scheme
1975
ML
1973
SL5
1976
Icon
1977
awk
1978
csh
october 1978
Scheme MIT
1978
Rex 1.00
may 1979
Modula 2
1979
Ada
1979
Smalltalk-78
1978
PostScript
1982
Forth-83
1983
Prolog II
october 1982
B
1981
Sharp APL
APL 2
august 1984
COBOL 85 ISO/ANSI
1985
Rex 2.00
1980
Rex 3.00
1982
Rexx 3.20
1984
Pascal AFNOR
1983
Ada 83 ANSI
january 1983
Concurrent C
1984
Classic C
Objective-C
1983
C with Classes
april 1980
C++
july 1983
Cedar
1983
Smalltalk-80
1980
KRC
1981
Miranda
1982
BASICA
1981
GW-Basic
1983
Common Lisp
1984
Scheme 84
1984
SML
1984
Caml
1987
nawk
1985
Prolog III
1984
Object Logo
1986
ANS Forth
1986
MUMPS (FIPS)
1986
Object Pascal
1985
Oberon
1987
Ada ISO
1987
OO Forth
1987
Tcl
mid 1988
Tcl/Tk
end 1988
A
1988
Modula 3
1988
ABC
1987
Borland
Turbo Pascal
ANSI C
(C89)
1989
ARM C++
1989
Eiffel
1986
Haskell 1.0
1987
QuickBasic 1.0
1985
QuickBasic 4.5
1988
Clos
1989
Perl 1.000
december 18, 1987
Perl 2.000
january 5, 1988
Perl 3.000
october 18, 1989
Self
PostScript level 2
1992
Fortran 90 ISO
1991
A+
1992
J
1990
MUMPS ISO
1992
Python
1991
Oberon-2
1991
NetRexx
1991
ISO C
(C90)
december 15, 1990
Cmm
1992
Oak
june 1991
Eiffel 2
1988
Sather 0.1
june 1991
Perl 4.000
march 21, 1991
MS Basic PDS 7.0
1989
MS PDS 7.1
1990
Visual Basic 1.0
may 20, 1991
Visual Basic 2.0
march 1992
Scheme IEEE
1990
Haskell 1.1
april 1, 1990
SML '90
1990
Haskell 1.2
march 1992
Caml 2-6.1
1991

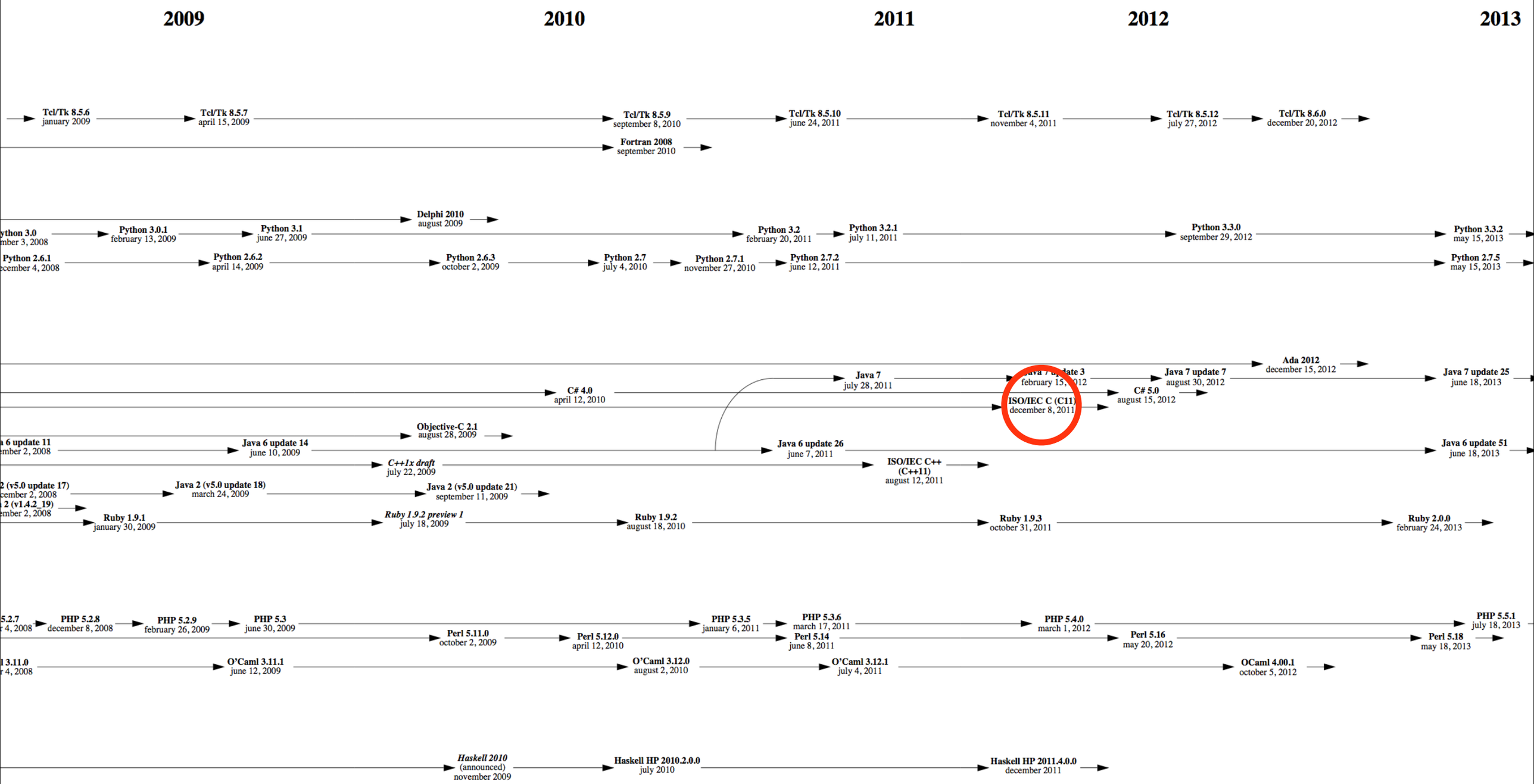
Evolution


1990
1995
2000
PostScript level 2
1992
PostScript level 3
september 11, 1996
ISO Forth
1997
Tcl/Tk 8.1
april 1999
Tcl/Tk 8.2.3
dec. 16, 1999
Fortran 90 ISO
1991
Fortran 95 ISO
december 15, 1997
A+
1992
Prolog IV
1997
J
1990
K
1996
MUMPS ISO
1992
M
1994
M ANSI
dec 8, 1995
Open M
dec 11, 1995
M ISO
1999
Delphi
march 2, 1995
Delphi 5
august, 1999
APL96
1996
Python
1991
Python 1.5.2
april 13, 1999
Python 1.6
september 5, 2000
Python 2.0
october 16, 2000
OO COBOL
1997
ANSI Rexx
Object Rexx
february 25, 1997
Oberon-2
1991
Modula-2 ISO
june 1, 1996
Modula-2 ISO
Generic Extension
december 19, 1998
Ada 95
1995
NetRexx
1991
C#
june 26, 2000
ISO C
(C90)
december 15, 1990
ISO C
(C95)
april 1, 1996
ISO C (C99)
december 1, 1999
JScript
may 1996
JScript
Cmm
1992
LiveScript
1995
JavaScript
december 1995
JavaScript 1.5
(draft 1)
february 18, 1999
C++98 ANSI/ISO
1998
ECMAScript
june 1997
ECMAScript ed3
december 1999
Oak
june 1991
Java 1
may 23, 1995
Java 2 (v1.2)
december 8, 1998
Java 2 (v1.3)
may 8, 2000
Ruby
february 24, 1993
Ruby 0.95
december 1995
Ruby 1.1 alpha 0
august 13, 1997
Ruby 1.3.2
april 2, 1999
Ruby 1.6.1
september 27, 2000
ANSI Smalltalk
may 19, 1998
Self 4.0
july 10, 1995
Eiffel 3
1993
Eiffel 4
december 11, 1996
Eiffel 4.2
february 6, 1998
PHP/FI
1995
PHP 2.0
nov. 13, 1997
PHP 3.0
june 6, 1998
PHP 4.0
may 22, 2000
Sather 0.1
june 1991
Sather 1.0
mid-1994
Sather 1.1
september 1995
Sather 1.2.1
november 4, 1999
Perl 4.000
march 21, 1991
Perl 5.000
october 18, 1994
Perl 5.005_50
july 26, 1998
Perl 5.6.0
march 28, 2000
Perl 5.7.0
september 2, 2000
Objective Caml
1996
O'Caml 1.0.7
december 11, 1997
O'Caml 2
1998
O'Caml 3.00
june 2000
MS PDS 7.1
1990
Visual Basic 1.0
may 20, 1991
Visual Basic 2.0
march 1992
Visual Basic 3.0
june 93
Visual Basic 4.0
september 1995
Visual Basic 5.0
april 1997
Visual Basic 6.0
june 16, 1998
Common Lisp ANSI
december 8, 1994
Scheme IEEE
1990
Scheme R5RS
1998
Haskell 1.1
april 1, 1990
Haskell 1.2
march 1992
Haskell 1.3
may 1996
Haskell 1.4
april 1997
Haskell 98
february 1999
SML '90
1990
SML '97
1997
Caml 2-6.1
1991
Caml 3.1
1993

Evolution


2003
2004
2005
PostScript level 3
v 3016
2003
PostScript level 3
v 3017
september 11, 2005
Tcl/Tk 8.4.2
march 3, 2003
Tcl/Tk 8.4.3
may 20, 2003
Tcl/Tk 8.4.4
july 22, 2003
Tcl/Tk 8.4.5
november 24, 2003
Tcl/Tk 8.4.6
march 1, 2004
Tcl/Tk 8.4.7
july 25, 2004
Tcl/Tk 8.4.8
nov. 22, 2004
Tcl/Tk 8.4.9
december 7, 2004
Tcl/Tk 8.4.11
june 28, 2005
Tcl/Tk 8.4.12
december 6, 2005
Tcl/Tk 8.4.13
april 19, 2006
Fortran 2003
november 30, 2004
M ISO
january 6, 2005
Delphi 8
november 2003
Delphi 2005
november 2004
Delphi 2006
october 30, 2005
Python 2.3a2
february 19, 2003
Python 2.2.3
may 30, 2003
Python 2.3
july 29, 2003
Python 2.3.1
september 23, 2003
Python 2.3.2
october 3, 2003
Python 2.3.3
december 19, 2003
Python 2.3.4
may 27, 2004
Python 2.4
november 30, 2004
Python 2.4.1
march 30, 2005
Python 2.4.2
september 28, 2005
ISO/ANSI
r 2002
Ada 2006 (draft)
2005
C#
(ISO)
march 28, 2003
C# 2.0
(beta)
july 2003
C# 3.0
(beta)
september 2005
C# 2.0
november 2005
C++03 ISO/IEC
2003
Java 2 (v6.0 beta)
december 2004
Java 2
(v1.4.1_02)
february 27, 2003
Java 2 (v1.4.2)
april 29, 2003
Java 2 (v1.4.1_03)
june 11, 2003
Java 2 (v1.4.2_01)
august 26, 2003
Java 2 (v1.4.2_02)
october 22, 2003
Java 2 (v1.4.2_03)
december 13, 2003
Java 2 (v1.5.0)
(beta 1)
february 5, 2004
Java 2 (v1.4.2_04)
march 8, 2004
Java 2 (v5.0) (beta 2)
june 28, 2004
Java 2 (v1.4.2_05)
june 30, 2004
Java 2 (v5.0)
september 30, 2004
Java 2 (v1.4.2_06)
november 23, 2004
Java 2 (v5.0 update 3)
april 28, 2005
Ruby 1.8
august 4, 2003
Ruby 1.8.1
december 25, 2003
Ruby 1.8.2
december 25, 2004
Ruby 1.8.3
september 21, 2005
Ruby 1.8.4
december 24, 2005
Self 4.2.1
april 3, 2004
ECMA Eiffel
june 2005
PHP 4.3.1
feb. 17, 2003
PHP 4.3.2
may 29, 2003
PHP 4.3.3
august 25, 2003
PHP 4.3.4
november 3, 2003
PHP 4.3.5
march 26, 2004
PHP 4.3.6
april 15, 2004
PHP 4.3.7
june 3, 2004
PHP 4.3.8
july 13, 2004
PHP 4.3.10
december 15, 2004
PHP 4.4.1
october 31, 2005
PHP 5.0.0
july 13, 2004
PHP 5.0.3
december 15, 2004
PHP 5.0.4
april 3, 2005
PHP 5.0.5
september 6, 2005
PHP 5.1.0
november 24, 2005
Perl 5.8.1
september 26, 2003
Perl 5.8.2
november 5, 2003
Perl 5.8.3
january 1, 2004
Perl 5.8.4
april 23, 2004
Perl 5.8.5
july 21, 2004
Perl 5.8.6
november 30, 2004
Perl 5.8.7
june 3, 2005
O'Caml 3.07
september 29, 2003
O'Caml 3.08.0
july 13, 2004
O'Caml 3.08.2
november 2004

Evolution


2009
2010
2011
2012
2013
Tcl/Tk 8.5.6 january 2009
Tcl/Tk 8.5.7 april 15, 2009
Tcl/Tk 8.5.9 september 8, 2010
Tcl/Tk 8.5.10 june 24, 2011
Tcl/Tk 8.5.11 november 4, 2011
Tcl/Tk 8.5.12 july 27, 2012
Tcl/Tk 8.6.0 december 20, 2012
Fortran 2008 september 2010
Delphi 2010 august 2009
Python 3.0.1 february 13, 2009
Python 3.1 june 27, 2009
Python 3.2 february 20, 2011
Python 3.2.1 july 11, 2011
Python 3.3.0 september 29, 2012
Python 3.3.2 may 15, 2013
Python 2.6.1
Python 2.6.2 april 14, 2009
Python 2.6.3 october 2, 2009
Python 2.7 july 4, 2010
Python 2.7.1 november 27, 2010
Python 2.7.2 june 12, 2011
Python 2.7.5 may 15, 2013
Ada 2012 december 15, 2012
Java 7 july 28, 2011
Java 7 update 7 august 30, 2012
Java 7 update 25 june 18, 2013
C# 4.0 april 12, 2010
C# 5.0 august 15, 2012
ISO/IEC C (C11) december 8, 2011
Objective-C 2.1 august 28, 2009
Java 6 update 14 june 10, 2009
Java 6 update 26 june 7, 2011
Java 6 update 51 june 18, 2013
C++1x draft july 22, 2009
ISO/IEC C++ (C++11) august 12, 2011
Java 2 (v5.0 update 18) march 24, 2009
Java 2 (v5.0 update 21) september 11, 2009
Ruby 1.9.1 january 30, 2009
Ruby 1.9.2 preview 1 july 18, 2009
Ruby 1.9.2 august 18, 2010
Ruby 1.9.3 october 31, 2011
Ruby 2.0.0 february 24, 2013
PHP 5.2.8 december 8, 2008
PHP 5.2.9 february 26, 2009
PHP 5.3 june 30, 2009
PHP 5.3.5 january 6, 2011
PHP 5.3.6 march 17, 2011
PHP 5.4.0 march 1, 2012
PHP 5.5.1 july 18, 2013
Perl 5.11.0 october 2, 2009
Perl 5.12.0 april 12, 2010
Perl 5.14 june 8, 2011
Perl 5.16 may 20, 2012
Perl 5.18 may 18, 2013
O'Caml 3.11.1 june 12, 2009
O'Caml 3.12.0 august 2, 2010
O'Caml 3.12.1 july 4, 2011
OCaml 4.00.1 october 5, 2012
Haskell 2010 (announced) november 2009
Haskell HP 2010.2.0.0 july 2010
Haskell HP 2011.4.0.0 december 2011

Bücher

- „The C Programming Language“ (2nd Edition)
  - Brian W. Kernighan
  - Dennis M. Ritchie

C11

Standard

Warum Standardisierung?

- C entwickelt sich schnell
  - Entspricht nicht mehr dem von Kernighan und Ritche beschriebenen Spezifikation
  - Viele Erweiterungen, fehlende Vereinheitlichung
- Ziel ist eine Normierung der Sprache

C Standard

- C wurde mehrfach standardisiert
  - C89/C90, C95, C99, C11
- 1983 ANSI (American National Standards Institute) Komitee X3J11 nimmt sich der Ausarbeitung eines Standards an
- 1989 erster Standard „ANSI X3.159-1989 Programming Language C“
- 1990 übernimmt ISO den Standard mit kleinen Änderungen

C Standard

- 1995 ergänzt ISO den Standard C90
- 1999 wird ISO/IEC 9899 verabschiedet (C99), in diesen Standard fließen aus C++ bekannte Erweiterungen zurück in C
- Seitdem Arbeit an C1X durch Komitee WG14, heute bekannt als C11

Regeln für Änderungen

- Vollständige Kompatibilität zu vorherigem Standard
- Bestehender Code hat hohen Stellenwert
- Modifikationen sollen Geist der Sprache nicht verändern
- Das Prinzip des Vertrauens (z.B. Programmierer muss sicherstellen, dass der Speicher das Ergebnis einer Operation aufnehmen kann), wird durch optionale Erweiterungen ergänzt
  - z.B. Bounds-Checking (Annex K, folgt später)

Features

Überblick

- Thread Unterstützung
- Atomare Operationen
- Bounds-Checking
- Static Assertions
- Weitere Neuerungen
- Compiler-Unterstützung

Multithreading

- `<threads.h>` Bestandteil von C seit C11
- Zuvor Rückgriff auf
  - POSIX Threads (im UNIX Umfeld)
  - Windows Threads
- Anlehnung an POSIX Threads
  - Bereits gut dokumentiert
  - Anwender nicht verwirren

Multithreading

- Problem vor C11
  - C Spezifikation sollte eine abstrakte Maschine referenzieren, die eine Generalisierung der benutzten Maschine darstellt
  - Bezieht sich weder auf einen speziellen Compiler, noch auf ein Betriebssystem, noch auf eine CPU
  - Programmiert man entsprechend der Spezifikationen, sollte ein Programm zu jeder Zeit auf jedem System ohne Modifikationen kompilierbar und ausführbar sein

Multithreading

- Problem vor C11
  - Die abstrakte Maschine, die C referenziert, ist vor C11 singlethreaded
  - Es gibt folglich keine Spezifikation für Multithreading und das Programm ist nicht auf jeder Maschine ohne Modifikationen ausführbar
    - Man unterscheidet bei Multithreading sogar zwischen Betriebssystemen
- C11 löst mit einer neuen Spezifikation dieses Problem

Multithreading

- **`int thrd_create(thrd_t *thr, thrd_start_t func, void *arg);`**
  - `thrd_create` erzeugt einen neuen Thread
  - Führt `func(arg)` aus
  - Bei Erfolg wird `thr` der Identifier des neu erzeugten Threads

Multithreading

- **`_Noreturn void thrd_exit(int res);`**
  - `thrd_exit` beendet einen Thread
  - Gibt das Resultat an `res`
  - Das Programm terminiert nach beenden des letzten Threads normal

Multithreading

- **`int thrd_join(thrd_t thr, int *res);`**
  - `thrd_join` führt den aktuellen Thread mit dem Thread `thr` durch blockieren zusammen, bis der Thread terminiert ist
  - Wenn `res` kein Nullpointer ist, wird in `res` das Resultat des Threads gespeichert

Atomare Operationen

- Beispiel **vor** C11 Spezifikation
  - `int a; int b;`

Thread 1	**Thread 2**
`a = 1;`	`printf(„a = %d“, a);`
`b = 2;`	`printf(„b = %d, b);`

- **Vor** C11 ist das Verhalten undefiniert, da es keine Spezifikation für Threads gibt

Atomare Operationen

- Problemstellung: Mehrere Threads sollen die selben Variablen nutzen
- Operationen sollen serialisiert werden
- Ausführungsreihenfolge kann festgelegt werden

- Vor C11 bereits möglich durch Rückgriff auf externe Bibliotheken
  - z.B. GLib

Atomare Operationen

- Mutex
  - `mtx_lock(&mutex);`
  - `a=1;`
  - `mtx_unlock(&mutex);`
- Für einfache Operationen zu komplex
- Es gibt Prozessoren die solche Operationen als einen Schritt durchführen
  - Diese Operationen sind teurer, da Speicherzugriff für andere Prozesse kurzzeitig blockiert werden muss

Atomare Operationen

- C11 abstrahiert diese Prozessorbefehle
- `<stdatomic.h>` stellt Typen, Makros und Funktionen für atomare Operationen bereit
- Atomare Operationen werden häufig im Linux-Kernel verwendet, da sie performanter als Lockingmechanismen sind

Atomare Operationen

- **`void atomic_init(volatile A *obj, C value);`**
  - `atomic_init` initialisiert ein atomares Objekt des Zeigers **`obj`** mit dem Wert **`value`**
  - Beispiel
    - `atomic_int a;`
    - `atomic_init(&a, 1);`

Atomare Operationen

- **`void atomic_store(volatile A *object, C desired);`**
- **`void atomic_store_explicit(volatile A *object, C desired, memory_order order);`**
  - `atomic_store` ersetzt den Wert von `object` mit dem Wert aus `desired`
  - `order` gibt das Verhalten beim Ersetzen an

Atomare Operationen

- **`C atomic_load(volatile A *object);`**
- **`C atomic_load_explicit(volatile A *object, memory_order order);`**
  - `atomic_load` lädt den Wert eines Objektes und gibt ihn zurück
  - `order` gibt an wie der Wert geladen werden soll

Atomare Operationen

- Beispiel mit C11 Spezifikation
  - `atomic_int a; atomic_int b;`

Thread 1	Thread 2
`atomic_store(&a, 1);`	`atomic_load(&b);`
`atomic_store(&b, 2);`	`atomic_load(&a);`

- Seit der C11 Spezifikation ist das Verhalten definiert

Atomare Operationen

- Es gibt 3 mögliche Verhalten, **Thread2** lädt
  - (1) **b=0** und **a=0**, wenn dieser **vor Thread1** ausgeführt wird
  - (2) **b=2** und **a=1**, wenn dieser **nach Thread1** ausgeführt wird
  - (3) **b=0** und **a=1**, wenn dieser ausgeführt wird **nachdem Thread1 a** gespeichert hat und **bevor b** gespeichert wurde

Atomare Operationen

- Was nicht passieren kann ist
  - **b=2** und **a=0**, weil die Standardeinstellungen für atomares Laden/Speichern sequentielle Konsistenz vorschreibt
- Das bedeutet, dass sie in der Reihenfolge geschehen müssen wie sie in den Threads beschrieben sind

Atomare Operationen

- Nachteil
  - Dieses Verhalten ist auf einer CPU sehr teuer
- Lösung
  - Wenn der Algorithmus keine Ordnung benötigt, also beispielsweise auch den Zustand **b=2** und **a=0** akzeptiert, kann für Laden/Speichern eine bestimmte Ordnung angegeben werden

Atomare Operationen

- Beispiel
  - `atomic_int a; atomic_int b;`

Thread 1	Thread 2
`atomic_store(&a, 1, memory_order_relaxed)`	`atomic_load(&b, memory_order_relaxed)`
`atomic_store(&b, 2, memory_order_relaxed)`	`atomic_load(&a, memory_order_relaxed)`

- Dieses Verhalten ist auf modernen CPUs wesentlich schneller

Atomare Operationen

- Zusammenfassend standardisiert C11 Mutexe
- C11 bietet in seiner Spezifikation Möglichkeiten an, wie beim Laden/Speichern vorzugehen ist
- Resultat
  - Es lassen sich performante Routinen entlang der Spezifikationen schreiben
  - Der Code lässt sich jetzt und in Zukunft auf allen Systemen ohne Modifikationen ausführen

Bounds-Checking

- Annex K des ISO/IEC 9899:2011 Standards
- Sicherheitsrelevante Erweiterungen
- Sicherstellung von genügend Speicher für das Ergebnis einer Operation lag zuvor beim Entwickler
  - Folge: z.B. Buffer-Overflows

- Seit C11 unterstützende Methoden mit Suffix **_s**, um vor Sicherheitslücken zu schützen

Bounds-Checking

- Beispiel `strcat`
  - **`errno_t strcat_s(char * restrict s1, rsize_t s1max, const char * restrict s2);`**
- Erwartet einen zusätzlichen Parameter für die Puffergröße
- `strcat_s` kopiert nicht mehr als **`s1max`** Bytes zu `s1`

Bounds-Checking

- Beispiel `strcopy`
  - **`errno_t strcopy_s(char * restrict s1, rsize_t s1max, const char * restrict s2);`**
  - `strcopy_s` erfordert, dass **`s1max`** größer ist als die Länge von **`s2`**

Bounds-Checking

- Beispiel `gets`
  - **`char *gets_s(char * restrict buffer, size_t nch);`**
  - `gets_s` ließt höchstens **`nch`** Zeichen von der Standardeingabe
- Zuvor war **`gets`** eine häufig genutzte Sicherheitslücke
  - In C99 wurde es bereits deprecated
  - In C11 komplett entfernt

Bounds-Checking

- Methoden angelehnt an Visual C
  - Dort gab es Methoden mit Suffix `_s` schon vor C11
  - Implementierung in C11 ähnlich, aber nicht identisch

Static Assertions

- Ausdruck, der zur Übersetzungszeit ausgewertet wird und bei Auswertung gleich **0** einen String/ Fehler ausgibt
- Hilfreich zum Debuggen/Fehleranalyse
- In Java bereits lange vorhanden

Static Assertions

- Zur Zeit der C89 Spezifikation wurden Tests vom Präprozessor ausgeführt
  - Ausgabe der Fehler durch Präprozessoranweisung `#error`
- Tests durch Präprozessor sind sehr limitiert
  - `sizeof` wird beispielsweise nicht korrekt ausgewertet
  - Das liegt daran, dass diese Methoden erst konvertiert werden wenn der Präprozessor ausgeführt wurde

Static Assertions

- C89 Beispiele
  - `#if __STDC__ != 1`
  - `#error "Fehlerbeschreibung"`
  - `#endif`

- `#if sizeof(long) < 8`
- `#error "Fehlerbeschreibung"`
- `#endif`

Static Assertions

- Benötigt wird eine Anweisung, die diese Funktion zur Übersetzungszeit übernimmt (nachdem der Präprozessor ausgeführt wurde)
  - C11 stellt `_Static_assert` bereit

Static Assertions

- `_Static_assert` besteht aus
  - Einem Ausdruck, der zur Übersetzungszeit ausgewertet wird
  - Einem String, der ausgegeben wird, wenn die Anweisung zu **`0`** ausgewertet wird

- **`_Static_assert(`**`constant-expression, string-literal`**`);`**

Static Assertions

- Beispiel
  - `_Static_assert(sizeof(size_t) >= 8, "Fehlerbeschreibung");`

Weitere Neuerungen

- Exklusiver Dateizugriff
- Generic-Selections
- Unicode-Unterstützung
- Speicherzugriffe und Speicherausrichtung
- ...

Compiler-Unterstützung

- Nicht alle Compiler unterstützen die C11 Spezifikation vollständig
- Gilt ebenfalls, auch heute noch, für C99

Compiler-Unterstützung

	C99	C11
LLVM/ Clang	Nahezu alle Features (keine C99 Floating Point Pragmas)	Teilweise
Pelles C	**Vollständige Unterstützung**	**Vollständige Unterstützung**
IBM XL	**Vollständige Unterstützung**	**Vollständige Unterstützung**
GCC	Nahezu alle Features (1 Feature falsch, 6 nicht vorhanden)	Teilweise

Quellen

Quellen

- http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf
- http://stackoverflow.com/questions/6319146/c11-introduced-a-standardized-memory-model-what-does-it-mean-and-how-is-it-g
- http://bartoszmilewski.com/2008/12/01/c-atomics-and-memory-ordering/
- http://blog.smartbear.com/codereviewer/c11-a-new-c-standard-aiming-at-safer-programming/

Quellen

- https://www.securecoding.cert.org/confluence/display/seccode/STR07-C.+Use+the+bounds-checking+interfaces+for+remediation+of+existing+string+manipulation+code
- http://www.ibm.com/developerworks/rational/library/support-iso-c11/
- http://www.smorgasbordet.com/pellesc/
- http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an&subtype=ca&supplier=897&appname=IBMLinkRedirect&letternum=ENUS202-161

Quellen

- http://gcc.gnu.org/c99status.html
- http://gcc.gnu.org/wiki/C11Status