Software ideals and history

Bjarne Stroustrup

www.stroustrup.com/Programming

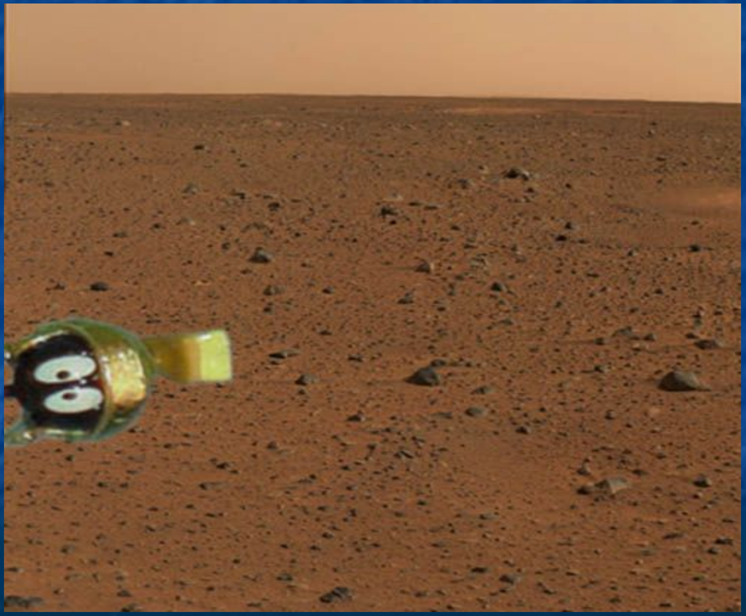
Abstract

- This is a very brief and very selective history of software as it relates to programming, and especially as it relates to programming languages and C++. The aim is to give a background and a perspective to the ideas presented in this course.
- We would have loved to talk about operating systems, data bases, networking, the web, scripting, etc., but you'll have to find those important and useful areas of software and programming in other courses.

Overview

- Ideals
  - Aims, heroes, techniques
- Languages and language designers
  - Early languages to C++

(There is so much more than what we can cover)



History and ideas

- One opinion
  - "History is bunk"
- Another opinion
  - "He who does not know history is condemned to repeat it"
- Our view
  - There can be no professionalism without history
    - If you know too little of the background of your field you are gullible
      - History is littered with plausible ideas that didn't work
        - "I have a bridge I'd like to sell you"
  - Ideas and ideals are crucial for practical use
    - And they are the real "meat" of history

What is a programming language?

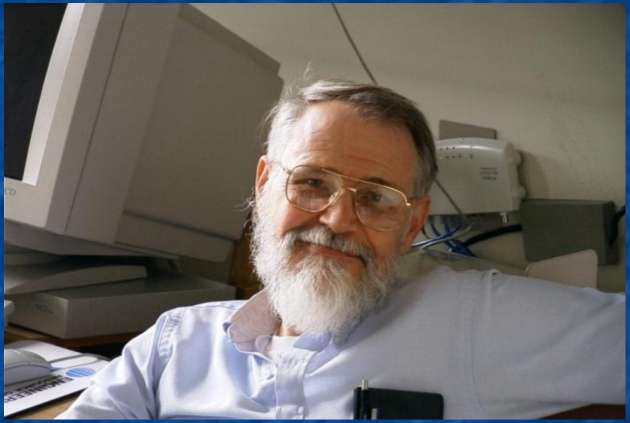
- A tool for instructing machines
- A notation for algorithms
- A means for communication among programmers
- A tool for experimentation
- A means for controlling computer-controlled gadgets
- A means for controlling computerized devices
- A way of expressing relationships among concepts
- A means for expressing high-level designs

- All of the above!
    - And more

Greek heroes

- Every culture and profession must have ideals and heroes
  - Physics: Newton, Einstein, Bohr, Feynman
  - Math: Euclid, Euler, Hilbert
  - Medicine: Hippocrates, Pasteur, Fleming

Geek heroes



- Brian Kernighan
  - Programmer and writer extraordinaire
- Dennis Ritchie
  - Designer and original implementer of C

Another geek hero

- Kristen Nygaard
  - Co-inventor (with Ole-Johan Dahl) of Simula67 and of object-oriented programming and object-oriented design

Yet another geek hero

- Alex Stepanov
  - Inventor of the STL and generic programming pioneer

Two extremes

- Academic beauty/perfection/purity
- Commercial expediency

- The pressures towards both are immense
- Both extremes must be avoided for serious progress to occur
  - Both extremes encourage overstatement of results (hype) and understatement (or worse) of alternatives and ancestor languages

Ideals

- The fundamental aims of good design
  - Represent ideas directly in code
  - Represent independent ideas independently in code
  - Represent relationships among ideas directly in code
  - Combine ideas expressed in code freely
    - where and only where combinations make sense
- From these follow
  - Correctness
  - Maintainability
  - Performance
- Apply these to the widest possible range of applications

Ideals have practical uses

- During the start of a project, reviews them to get ideas
- When you are stuck late at night, step back and see where your code has most departed from the ideals – this is where the bugs are most likely to lurk and the design problems are most likely to occur
    - Don’t just keep looking in the same place and trying the same techniques to find the bug
        - “The bug is always where you are *not* looking – or you would have found it already”

Ideals are personal

- Chose yours well

Styles/paradigms

- Procedural programming
- Data abstraction
- Object-oriented programming
- Generic programming

- Functional programming, logic programming, rule-based programming, constraints-based programming, aspect-oriented programming, …

Styles/paradigms

```
template<class Iter> void draw_all(Iter b, Iter e)
{
    for_each(b,e,mem_fun(&Shape::draw));    // draw all shapes in [b:e)
}


Point p(100,100);
Shape* a[] = { new Circle(p,50), new Rectangle(p, 250, 250) };
draw_all(a,a+2);
```

- Which programming styles/paradigms did we use here?
  - Procedural, data abstractions, OOP, and GP

Styles/paradigms

```
template<class Cont> void draw_all(Cont& c) // C++11
{
    for_each(Shape* p : c) p->draw();     // draw all shapes in c
}


void draw_all(Container& c) // C++14
{
    for_each(Shape* p : c) p->draw();     // draw all shapes in c
}
```

- It’s all just programming!

Some fundamentals

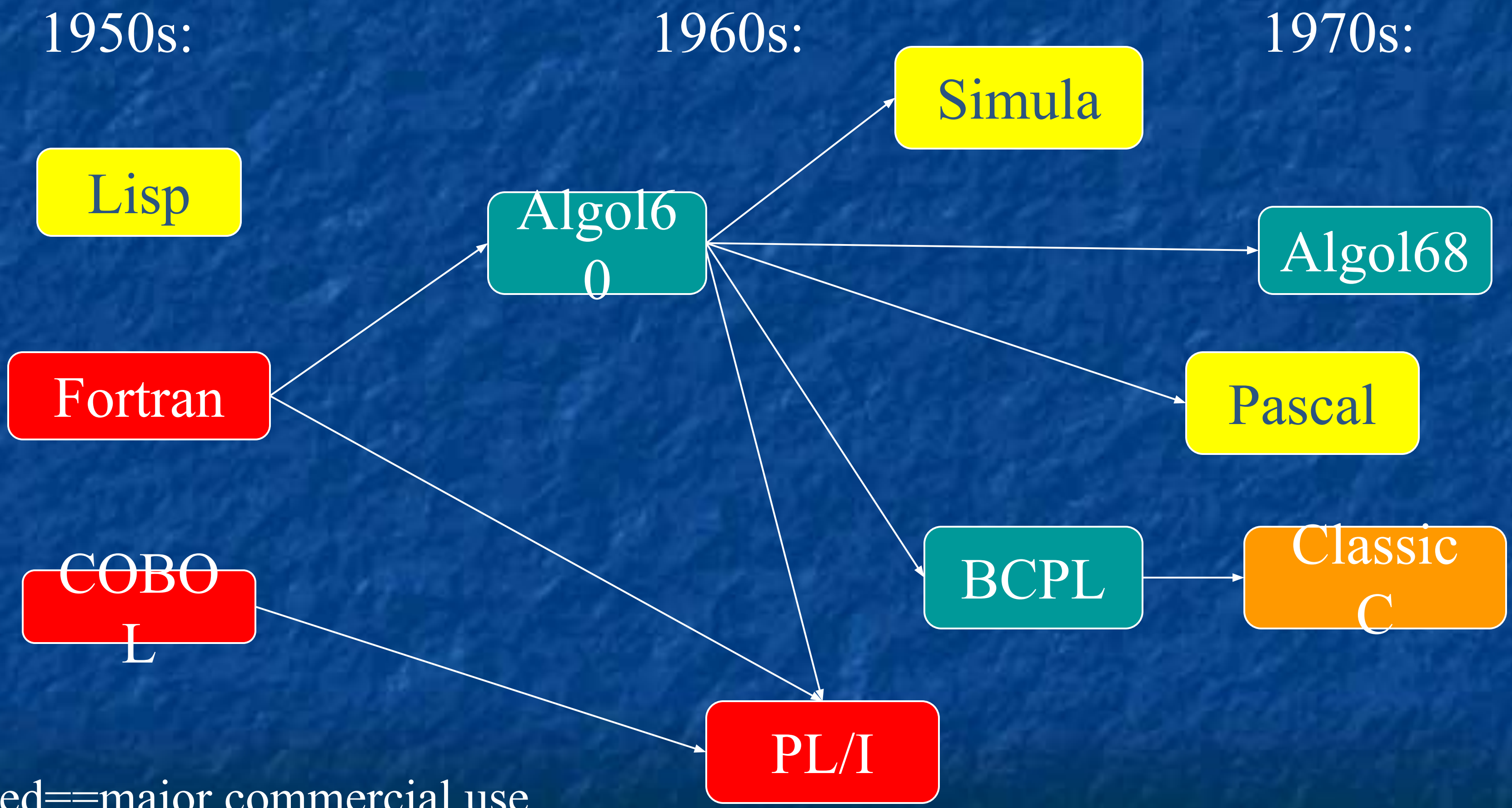
- Portability is good
- Type safety is good
- High performance is good
- Anything that eases debugging is good
- Access to system resources is good
- Stability over decades is good
- Ease of learning is good
- Small is good
- Whatever helps analysis is good
- Having lots of facilities is good

- You can’t have all at the same time: engineering tradeoffs

Programming languages

- Machine code
  - Bits, octal, or at most decimal numbers
- Assembler
  - Registers, load, store, integer add, floating point add, …
  - Each new machine had its own assembler
- Higher level languages
  - First: Fortran and COBOL
- Rate of language invention
  - At least 2000 a decade
- Major languages today
  - Really solid statistics are hard to come by
    - IDS: about 9 million professional programmers
  - COBOL, Fortran, C, C++, Visual Basic, PERL, Java, Javascript
    - Ada, C#, PHP, …

Early programming languages



1950s:

1960s:

1970s:

Lisp

Fortran

COBOL

Algol60

Simula

Algol68

Pascal

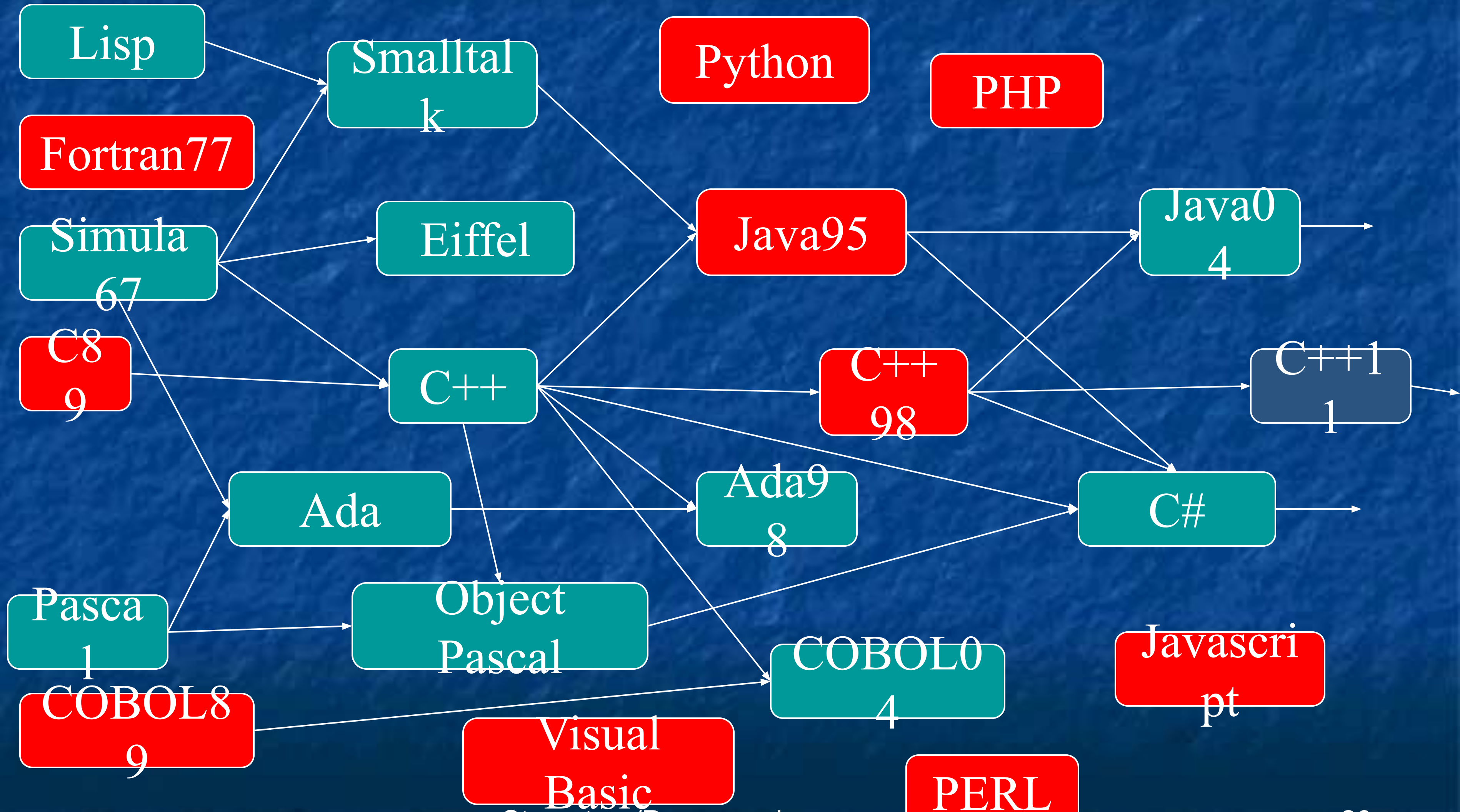
BCPL

Classic C

PL/I

Red==major commercial use
Yellow==will produce important “offspring”

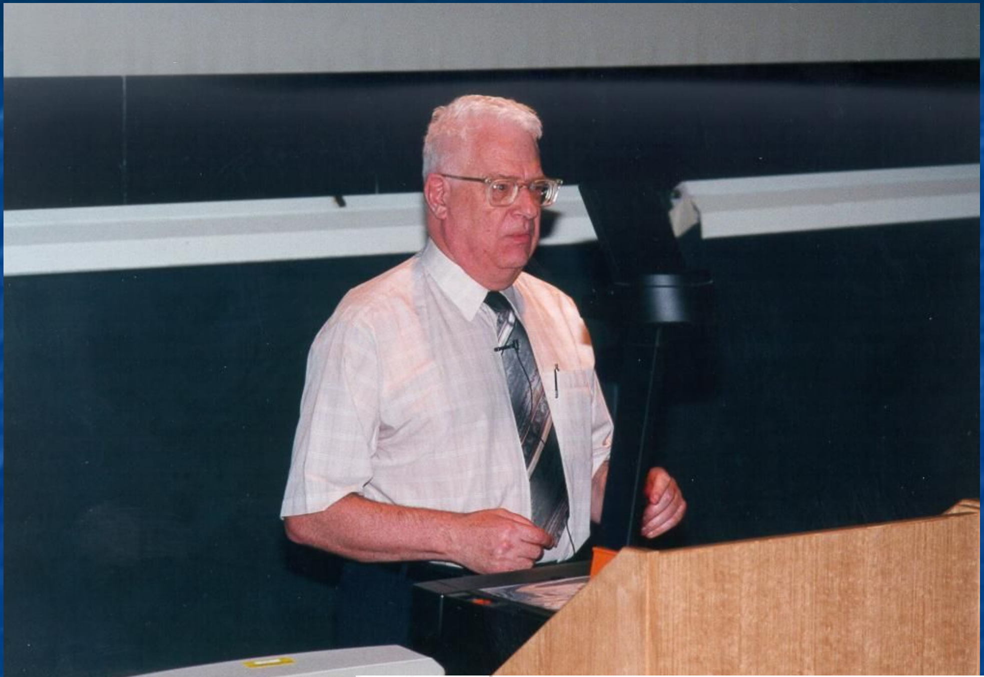
Modern programming languages

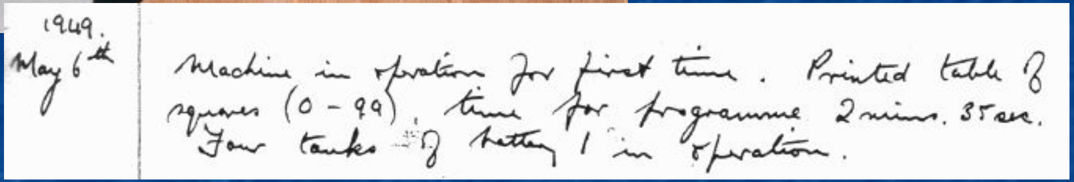


Why do we design and evolve languages?

- There are many diverse applications areas
  - No one language can be the best for everything
- Programmers have diverse backgrounds and skills
  - No one language can be best for everybody
- Problems change
  - Over the years, computers are applied in new areas and to new problems
- Computers change
  - Over the decades, hardware characteristics and tradeoffs change
- Progress happens
  - Over the decades, we learn better ways to design and implement languages

First modern computer – first compiler





1949.
May 6th

Machine in operation for first time. Printed table of squares (0 - 99), time for programme 2 mins. 35 sec. Four tanks of battery 1 in operation.

- David Wheeler (1927-2004)
  - University of Cambridge
  - Exceptional problem solver: hardware, software, algorithms, libraries
  - First computer science Ph.D. (1951)
  - First paper on how to write correct, reusable, and maintainable code (1951)
  - (Thesis advisor for Bjarne Stroustrup ☺)

Early languages – 1952

- One language for each machine
  - Special features for processor
  - Special features for “operating system”
  - Most had very assembler-like facilities
    - It was easy to understand which instructions would be generated
  - No portability of code

Fortran

- John Backus (1924-2007)
  - IBM
  - FORTRAN, the first high level computer language to be developed.
    - *We did not know what we wanted and how to do it. It just sort of grew.*
  - The Backus-Naur Form (BNF), a standard notation to describe the syntax of a high level programming language.
  - A functional programming language called FP, which advocates a mathematical approach to programming.

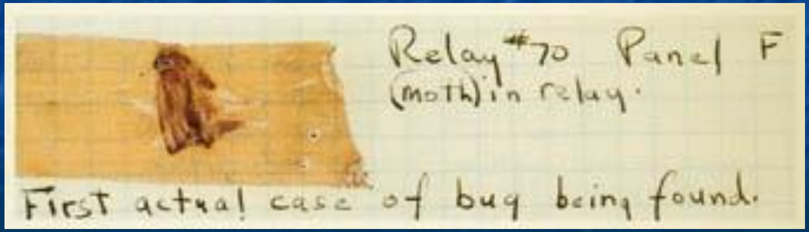
Fortran – 1956

- Allowed programmers to write linear algebra much as they found it in textbooks
  - Arrays and loops
  - Standard mathematical functions
    - libraries
  - Users’ own functions
- The notation was largely machine independent
  - Fortran code could often be moved from computer to computer with only minor modification
- This was a ***huge*** improvement
  - Arguably the largest single improvement in the history of programming languages
- Continuous evolution: II, IV, 77, 90, 95, 03, 08, [15]

COBOL

- “Rear Admiral Dr. Grace Murray Hopper (US Navy) was a remarkable woman who grandly rose to the challenges of programming the first computers. During her lifetime as a leader in the field of software development concepts, she contributed to the transition from primitive programming techniques to the use of sophisticated compilers. She believed that ‘we've always done it that way’ was not necessarily a good reason to continue to do so.”

(1906-1992)



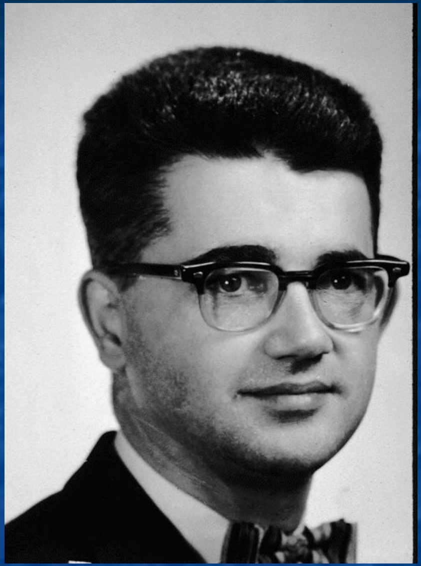
Relay #70 Panel F
(moth) in relay.

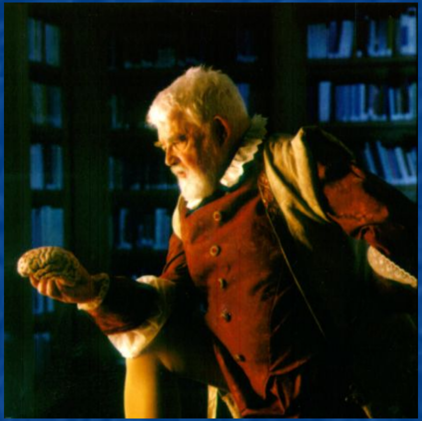
First actual case of bug being found.

Cobol – 1960

- Cobol was (and sometimes still is) for business programmers what Fortran was (and sometimes still is) for scientific programmers
- The emphasis was on data manipulation
  - Copying
  - Storing and retrieving (record keeping)
  - Printing (reports)
- Calculation/computation was seen as a minor matter
- It was hoped/claimed that Cobol was so close to business English that managers could program and programmers would soon become redundant
- Continuous evolution: 60, 61, 65, 68, 74, 85, 02

Lisp

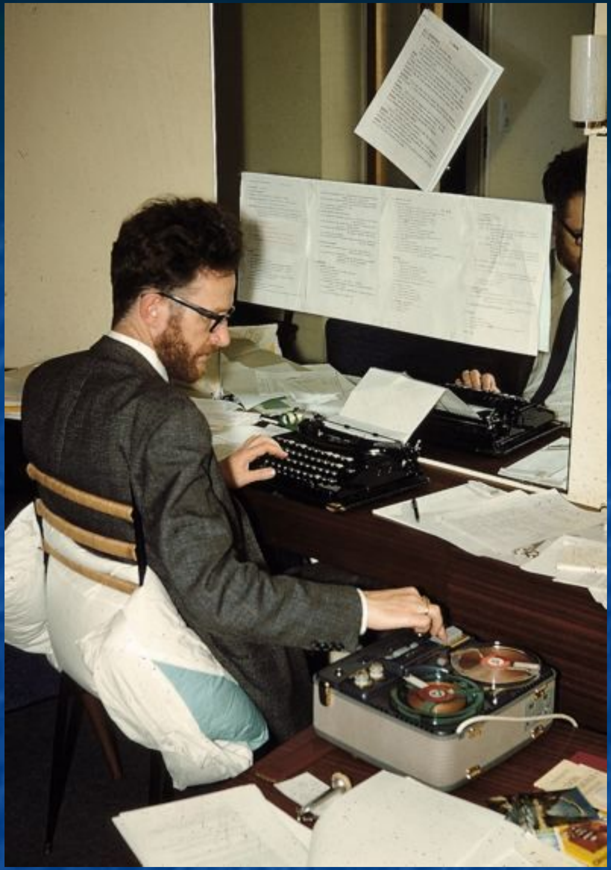




- John McCarthy (1927-2011)
  - Stanford University
  - AI pioneer

Lisp – 1960

- List/symbolic processing
- Initially (and often still) interpreted
- Dozens (most likely hundreds) of dialects
  - “Lisp has an implied plural”
  - Common Lisp
  - Scheme
- This family of languages has been (and is) the mainstay of artificial intelligence (AI) research
  - though delivered products have often been in C or C++

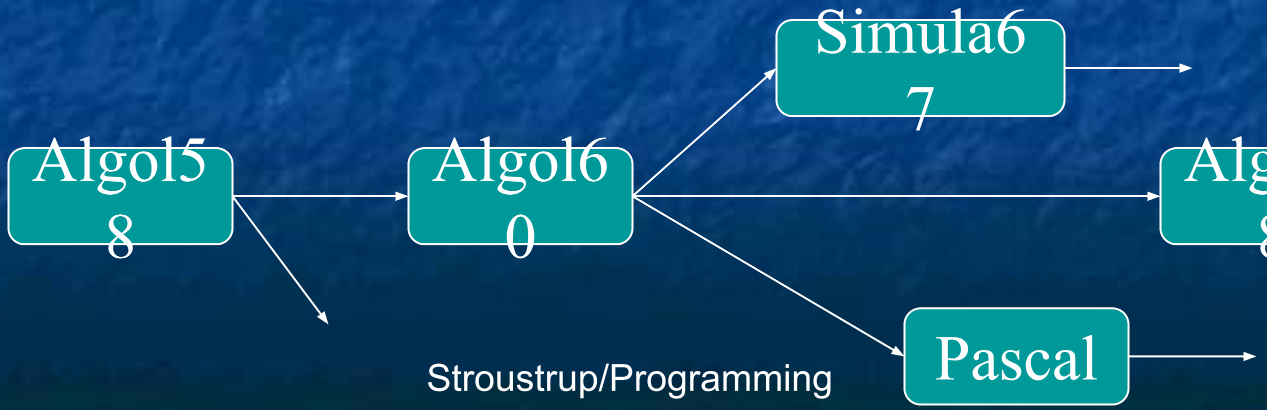


Algol

- Peter Naur (b. 1928)
  - Danish Technical University and Regnecentralen
  - BNF
- Edsger Dijkstra (1930-2002)
  - Mathematisch Centrum, Amsterdam, Eindhoven University of Technology, Burroughs Corporation , University of Texas (Austin)
  - Mathematical logic in programming, algorithms
  - THE operating system

Algol – 1960

- The breakthrough of modern programming language concepts
  - Language description
    - BNF; separation of lexical, syntactic, and semantic concerns
  - Scope
  - Type
  - The notion of "general purpose programming language"
    - Before that languages were either scientific (e.g., Fortran), business (e.g., Cobol), string manipulation (e.g., Lisp), simulation, …
- Never reached major non-academic use



Simula 67

- Kristen Nygaard (1926-2002) and Ole-Johan Dahl (1931-2002)
  - Norwegian Computing Center
  - Oslo University
  - The start of object-oriented programming and object-oriented design

Simula 1967

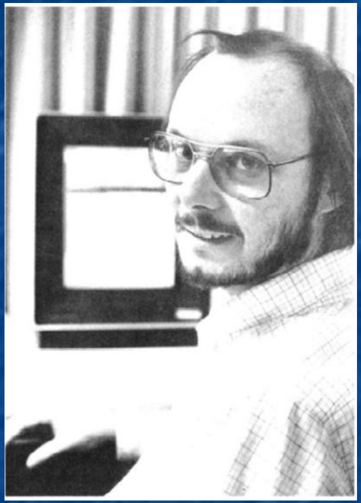
- Address all applications domains rather then a specific domain
  - As Fortran, COBOL, etc. did
  - Aims to become a true general-purpose programming language
- Model real-world phenomena in code
  - represent ideas as classes and class objects
  - represent hierarchical relations as class hierarchies
- Classes, inheritance, virtual functions, object-oriented design
- A program becomes a set of interacting objects rather than a monolith
  - Has major (positive) implications for error rates

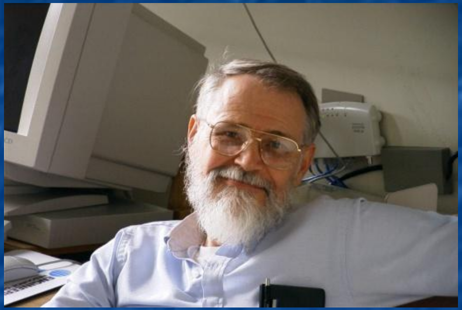
C

- Dennis Ritchie (1941-2011)
  - Bell Labs
  - C and helped with Unix
- Ken Thompson (b. 1943)
  - Bell Labs
  - Unix

- Doug McIlroy (b. 1932)
  - Bell Labs
  - Everybody’s favorite critic, discussion partner, and ideas man (influenced C, C++, Unix, and much more)

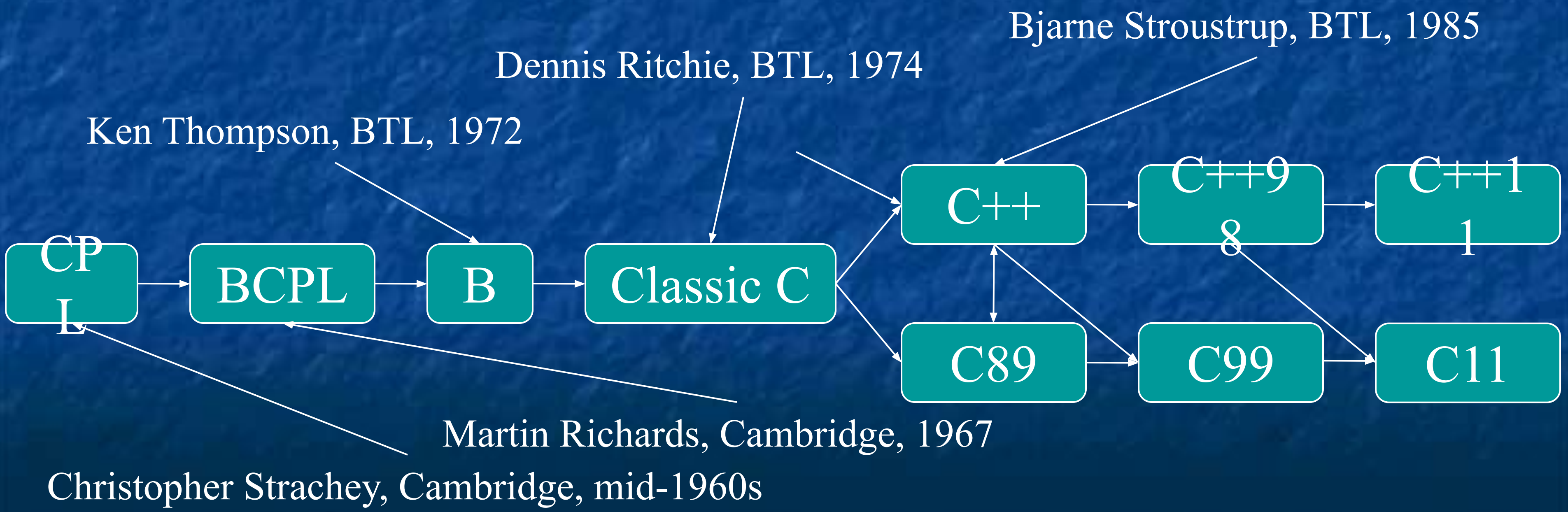
Bell Labs – Murray Hill





C – 1978

- (Relatively) high-level programming language for systems programming
  - Very widely used, weakly checked, systems programming language
  - Associated with Unix and through that with Linux and the open source movement
  - Direct map to hardware
  - Performance becomes somewhat portable
  - Designed and implemented by Dennis Ritchie 1974-78


Bjarne Stroustrup, BTL, 1985
Dennis Ritchie, BTL, 1974
Ken Thompson, BTL, 1972
CPL
BCPL
B
Classic C
C++
C++98
C++11
C89
C99
C11
Martin Richards, Cambridge, 1967
Christopher Strachey, Cambridge, mid-1960s

Parasol
Smarter computing.
Texas A&M University

C++

- Bjarne Stroustrup
  - AT&T Bell labs
  - Texas A&M University
  - making abstraction techniques affordable and manageable for mainstream projects
  - pioneered the use of object-oriented and generic programming techniques in application areas where efficiency is a premium

My ideals – in 1980 and more so in 2013

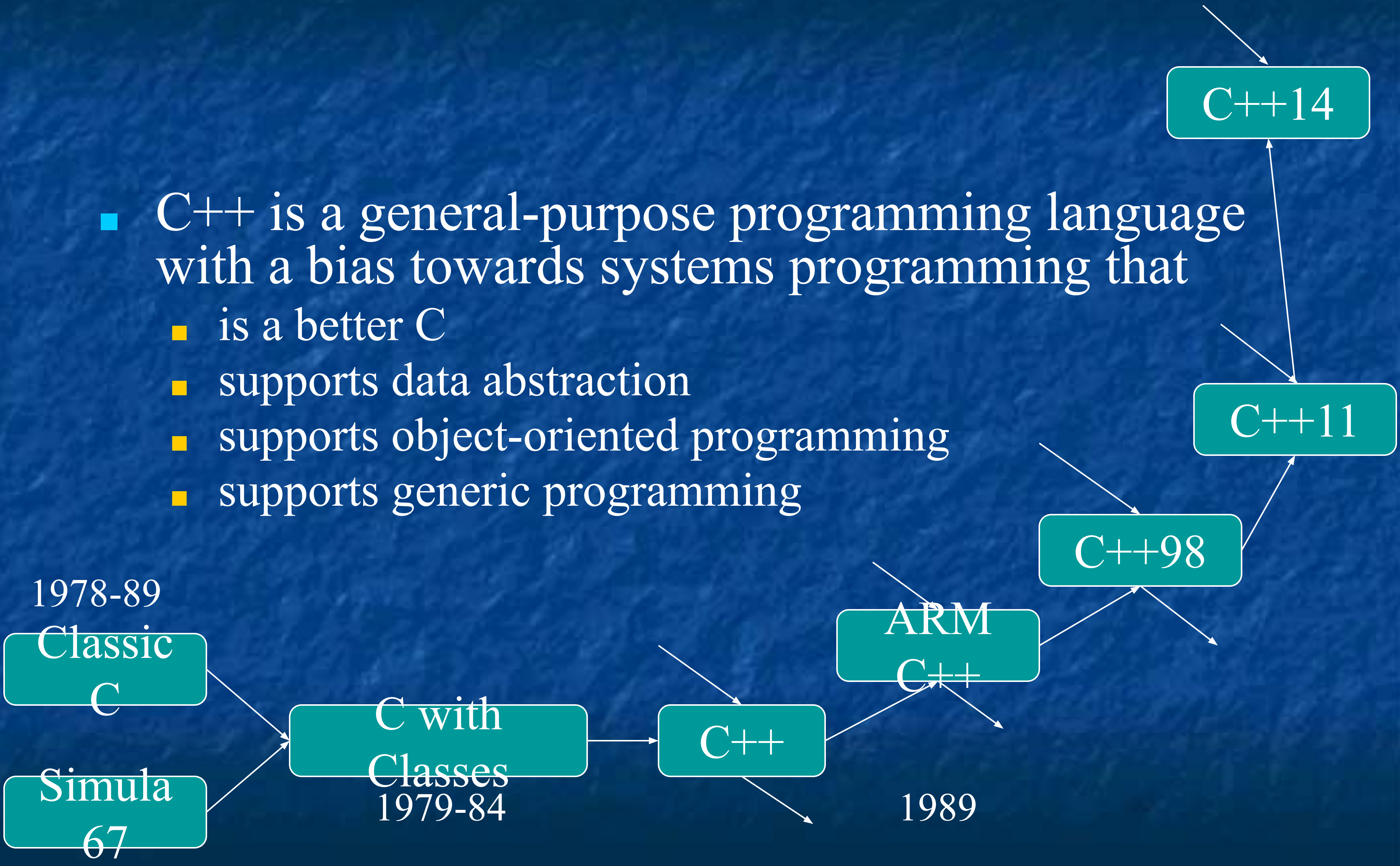
Parasol
Smarter computing.
Texas A&M University

- "To make life easier for the serious programmer"
    - i.e., primarily me and my friends/colleagues
  - I love writing code
  - I like reading code
  - I hate debugging

- Elegant and efficient code
  - I really dislike choosing between the two
  - Elegance, efficiency, and correctness are closely related in many application domains
    - Inelegance/verbosity is a major source of bugs and inefficiencies

C++ – 1985

- C++ is a general-purpose programming language with a bias towards systems programming that
  - is a better C
  - supports data abstraction
  - supports object-oriented programming
  - supports generic programming



1978-89

Classic C

Simula 67

C with Classes

1979-84

C++

ARM C++

1989

C++98

C++11

C++14

More information

- More language designer links/photos
  - http://www.angelfire.com/tx4/cus/people/
- A few examples of languages:
  - http://dmoz.org/Computers/Programming/Languages/
- Textbooks
  - Michael L. Scott, *Programming Language Pragmatics*, Morgan Kaufmann, 2000, ISBN 1-55860-442-1
  - Robert W. Sebesta, *Concepts of programming languages*, Addison-Wesley, 2003, ISBN 0-321-19362-8
- History books
  - Jean Sammet, *Programming Languages: History and Fundamentals*, Prentice-Hall, 1969, ISBN 0-13-729988-5
  - Richard L. Wexelblat, *History of Programming Languages*, Academic Press, 1981, ISBN 0-12-745040-8
  - T. J. Bergin and R. G. Gibson, *History of Programming Languages – II*, Addison-Wesley, 1996, ISBN 0-201-89502-1