A Brief History of Computing

Xmas Lecture

Prof. Dr. Jörg Schäfer

23 December, 2010

Content

Prologue

Disclaimer

The following slides contain an *entirely subjective* view of (computer) history!

Due to laziness, some notes contain Wikipedia as a "reference" – please consider this as an anti-pattern and *never* use Wikipedia as a "reference" in any serious scientific report!

Famous (Computer) Scientists Quotes

Famous (Computer) Scientists Quotes

Computer science is no more about computers than astronomy is about telescopes.

Edsger Dijkstra

Famous (Computer) Scientists Quotes

Computer science is no more about computers than astronomy is about telescopes.

Edsger Dijkstra

Computer science also differs from physics in that it is not actually a science. It does not study natural objects. Neither is it, as you might think, mathematics; although it does use mathematical reasoning pretty extensively. Rather, computer science is like engineering; it is all about getting something to do something, rather than just dealing with abstractions, as in the pre-Smith geology.

Richard Feynman, Feynman Lectures on Computation, 1970

Famous (Computer) Engineers Quotes

Famous (Computer) Engineers Quotes

I think there is a world market for maybe five computers.
Thomas Watson, Chairman of IBM, 1943

Famous (Computer) Engineers Quotes

I think there is a world market for maybe five computers.

Thomas Watson, Chairman of IBM, 1943

There is no reason anyone in the right state of mind will want a computer in their home.

Ken Olson, President of Digital Equipment Corp, 1977

Famous (Computer) Engineers Quotes

I think there is a world market for maybe five computers.

Thomas Watson, Chairman of IBM, 1943

There is no reason anyone in the right state of mind will want a computer in their home.

Ken Olson, President of Digital Equipment Corp, 1977

640k is enough for anyone, and by the way, what's a network?

William Gates III, President of Microsoft Corporation, 1984

Famous (Computer) Engineers Quotes

I think there is a world market for maybe five computers.

Thomas Watson, Chairman of IBM, 1943

There is no reason anyone in the right state of mind will want a computer in their home.

Ken Olson, President of Digital Equipment Corp, 1977

640k is enough for anyone, and by the way, what's a network?

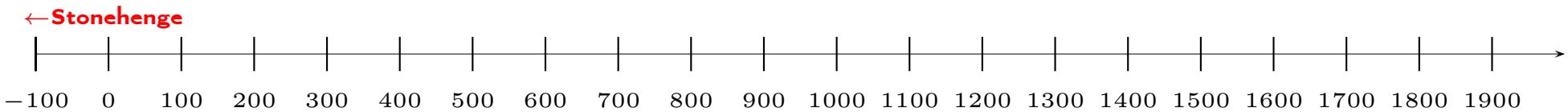
William Gates III, President of Microsoft Corporation, 1984

The most profound technologies are those that disappear: they weave themselves into fabric of everyday life until are indistinguishable from it.

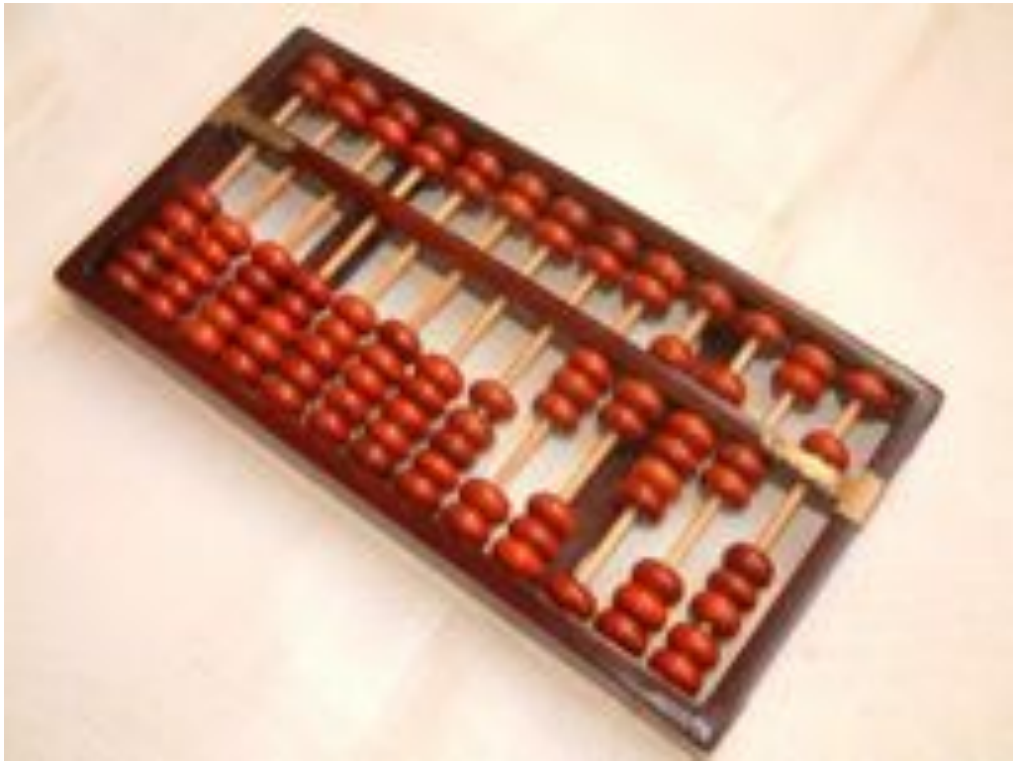
Mark Weiser, The Computer for the 21st Century, Scientific American, 1991

A Brief History of Computers

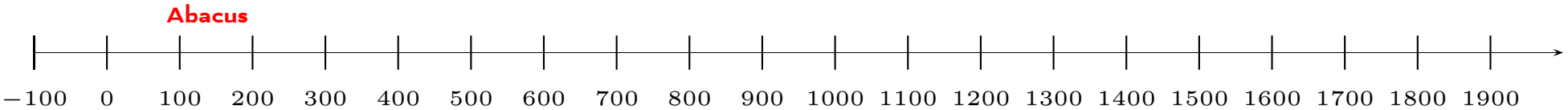
Stonehenge


←Stonehenge
−100
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900

Abacus

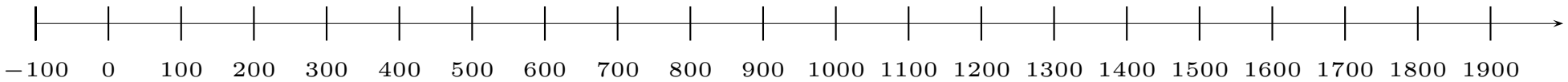


Used in many cultures including Egypt, Persia, Greek, Roman, Chinese, India, Japan, Native American, ...



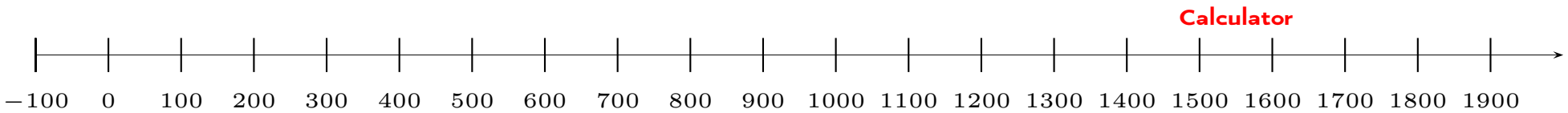
Al-Chwarizmi

Algorithms



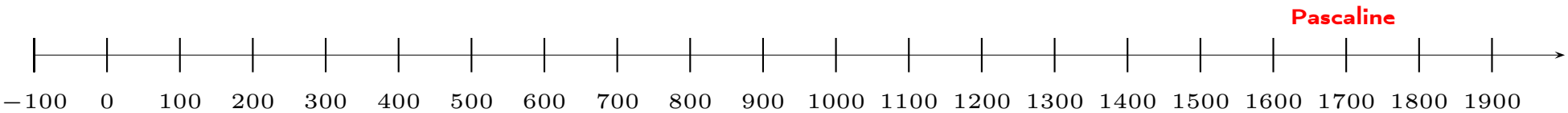
Leonardo da Vinci

Content (Possibly) Copyright Protected



Pascaline

In 1642 Blaise Pascal, at age 19, invented the Pascaline.



Leibniz 01

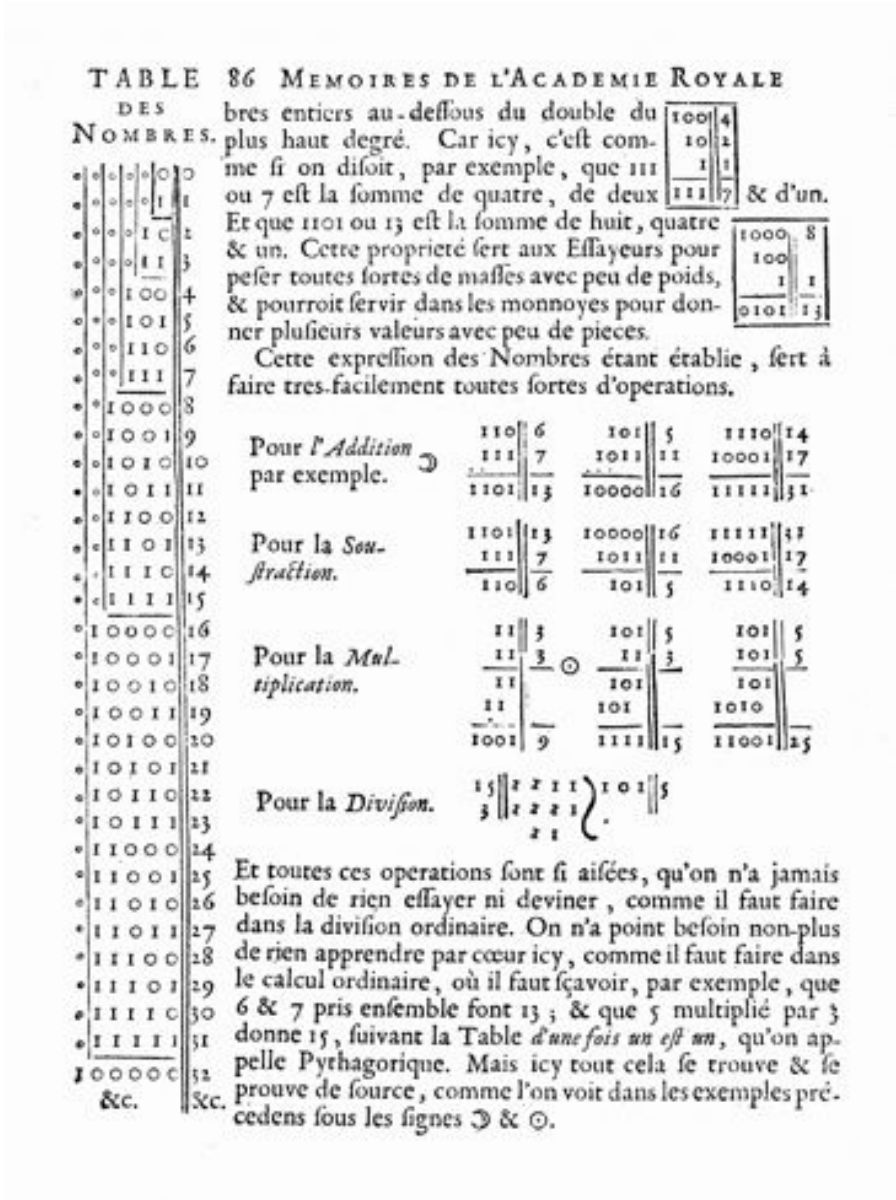


TABLE DES NOMBRES.

86 MEMOIRES DE L'ACADEMIE ROYALE

bres entiers au-deſſous du double du plus haut degré. Car icy, c'eſt comme ſi on diſoit, par exemple, que 111 ou 7 eſt la ſomme de quatre, de deux & d'un. Et que 1101 ou 13 eſt la ſomme de huit, quatre & un. Cette proprieté ſert aux Eſſayeurs pour peſer toutes ſortes de maſſes avec peu de poids, & pourroit ſervir dans les monnoyes pour donner pluſieurs valeurs avec peu de pieces.

Cette expreſſion des Nombres étant établie, ſert à faire tres-facilement toutes ſortes d'operations.

Pour *l'Addition* par exemple. ☽

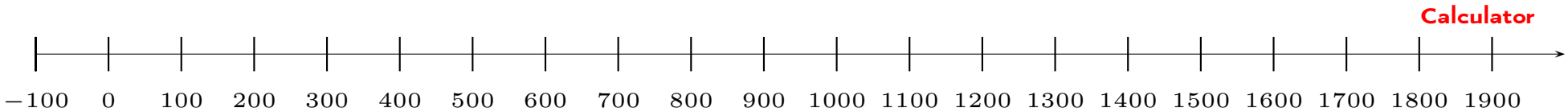
Pour la *Souſtraction.*

Pour la *Multiplication.* ⊙

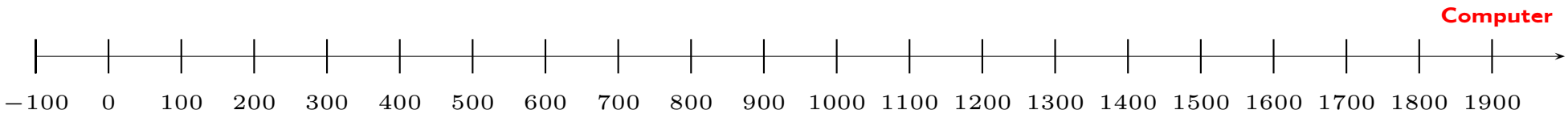
Pour la *Diviſion.*

Et toutes ces operations ſont ſi aiſées, qu'on n'a jamais beſoin de rien eſſayer ni deviner, comme il faut faire dans la diviſion ordinaire. On n'a point beſoin non-plus de rien apprendre par cœur icy, comme il faut faire dans le calcul ordinaire, où il faut ſçavoir, par exemple, que 6 & 7 pris enſemble font 13; & que 5 multiplié par 3 donne 15, ſuivant la Table *d'une fois un eſt un*, qu'on appelle Pythagorique. Mais icy tout cela ſe trouve & ſe prouve de ſource, comme l'on voit dans les exemples précedens ſous les ſignes ☽ & ⊙.

Charles Babbage's Difference Engine

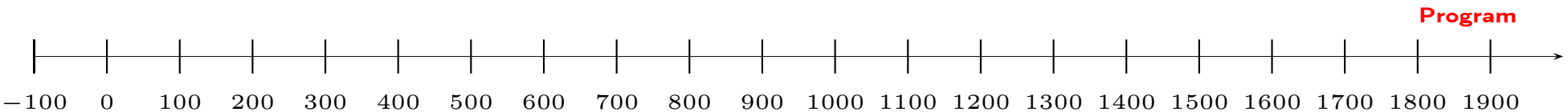


Charles Babbage's Analytical Engine



The First Programmer

Augusta **Ada** King Byron, Countess of **Lovelace**



Inspiration for Cyberpunk

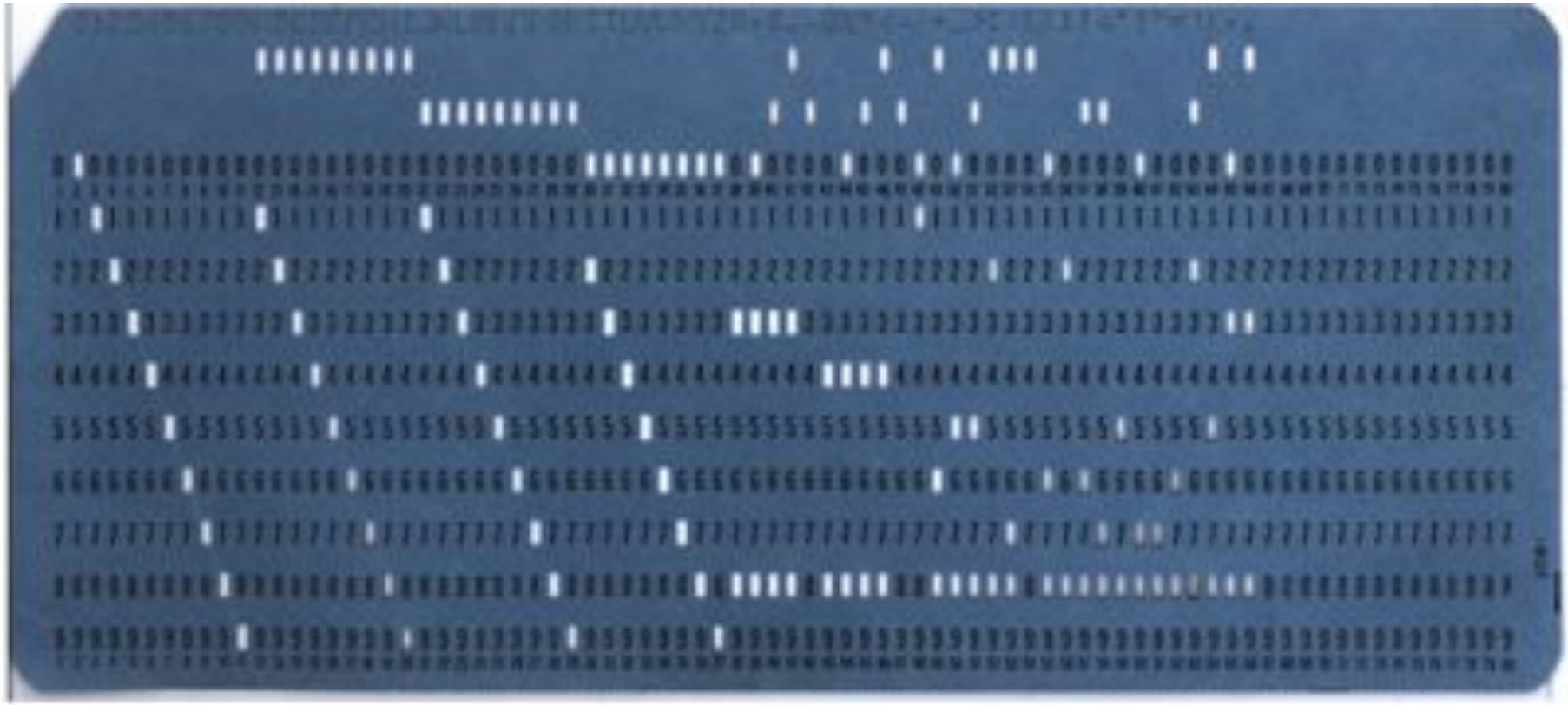
Content (Possibly) Copyright Protected

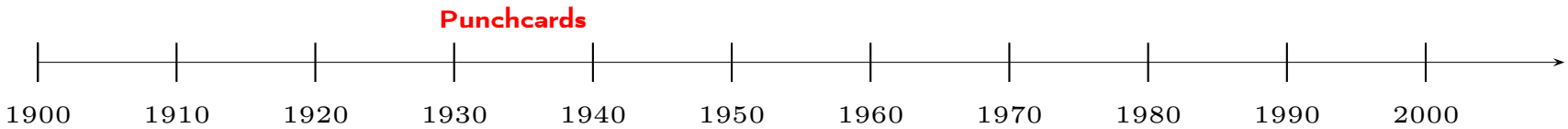
Inspiration for Steampunk

Power (Jacquard) Looms

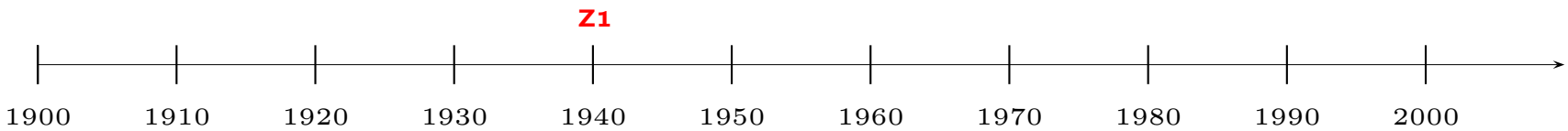
We may say most aptly, that the Analytical Engine weaves algebraical patterns just as the Jacquard-loom weaves flowers and leaves. (Lady Ada Lovelace)

IBM, formerly known as Hollerith



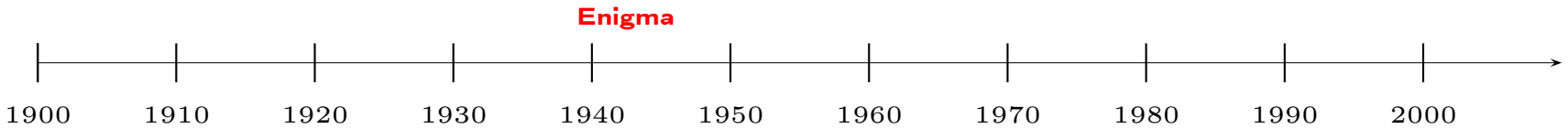


Konrad Zuse – Z1

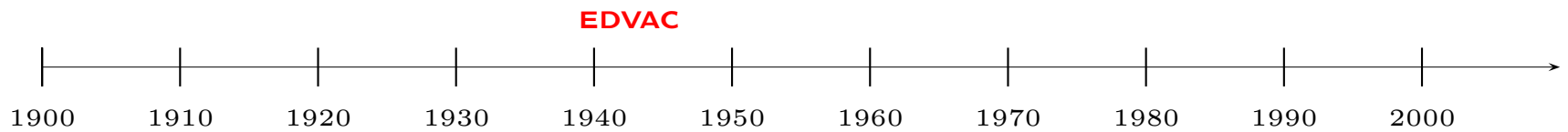


Alan Turing [Tur01]

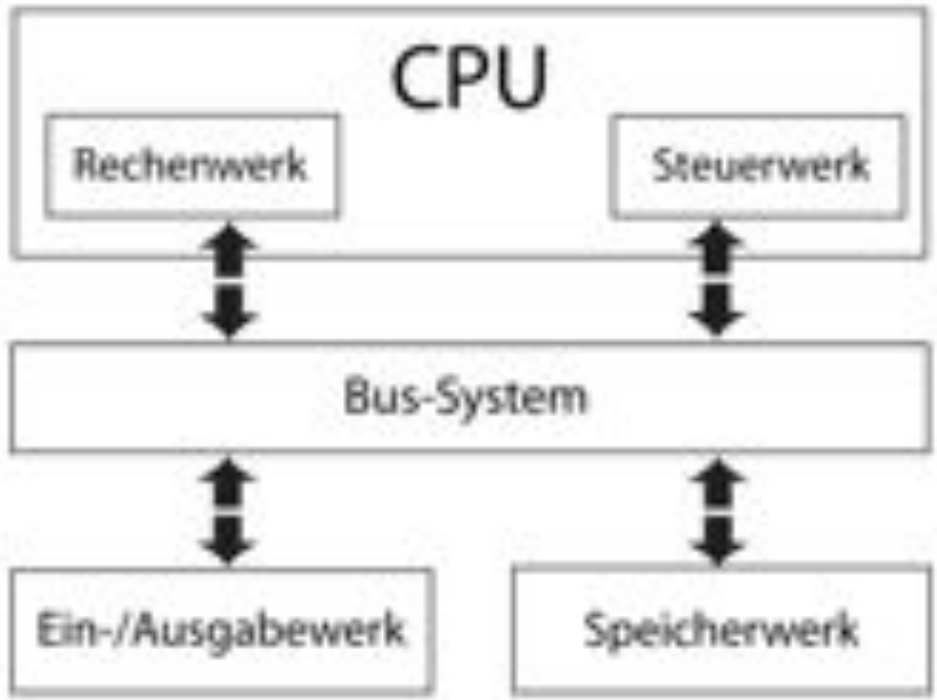
Content (Possibly) Copyright Protected

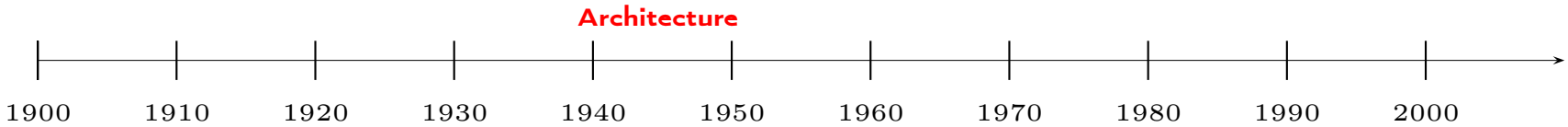


John von Neumann [Neu45]



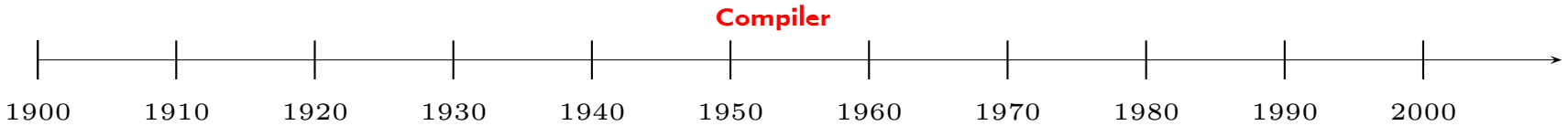
Von Neumann Architecture



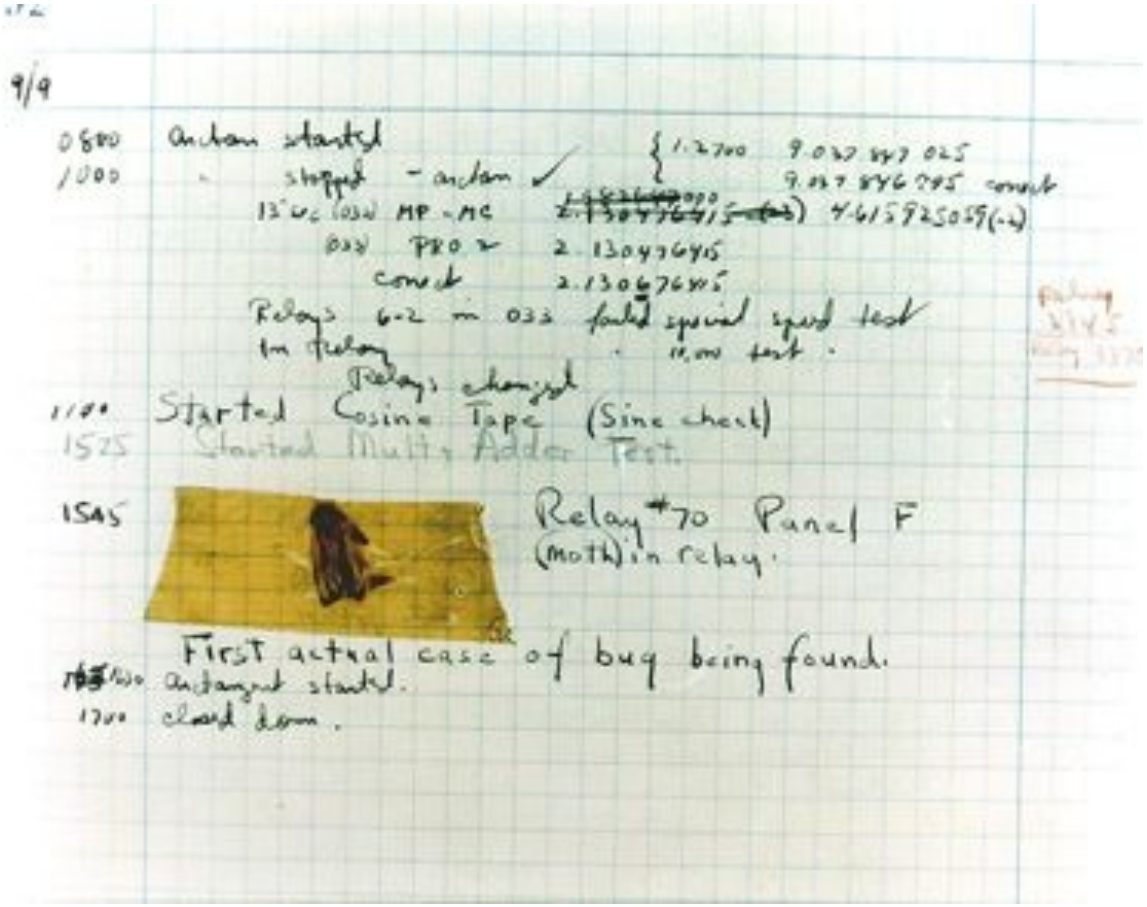


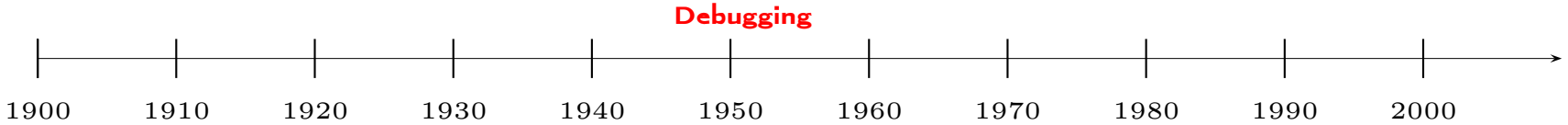
Grace Hopper

Grace Murray Hopper was an American computer scientist and United States Naval officer. One of the first programmers of the Harvard Mark I computer. She developed the first compiler for a computer programming language.

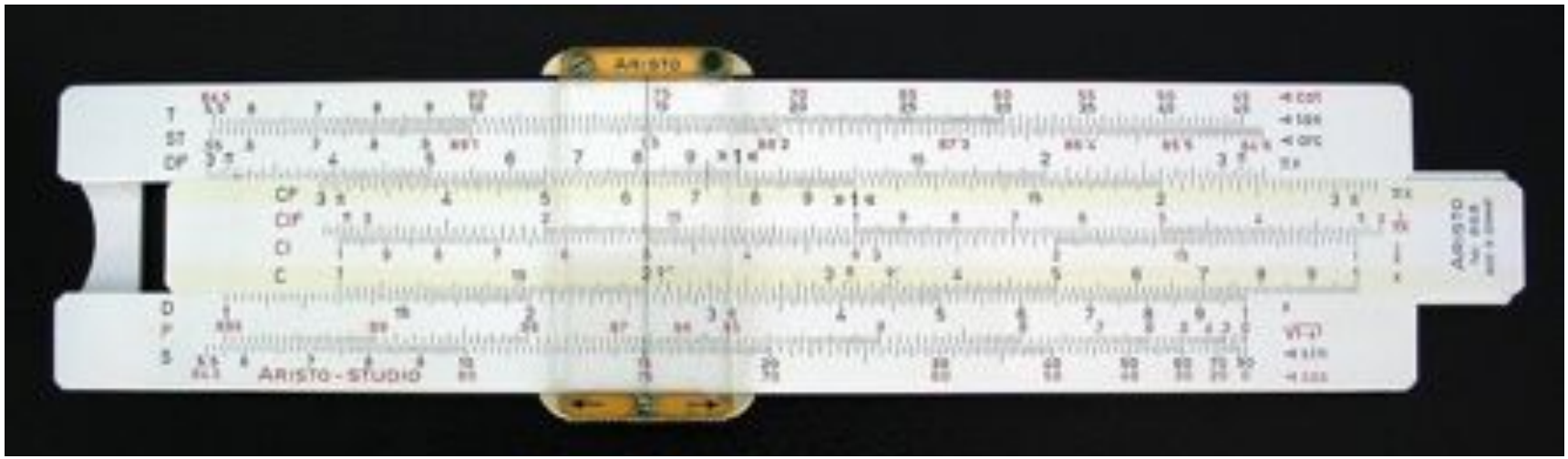


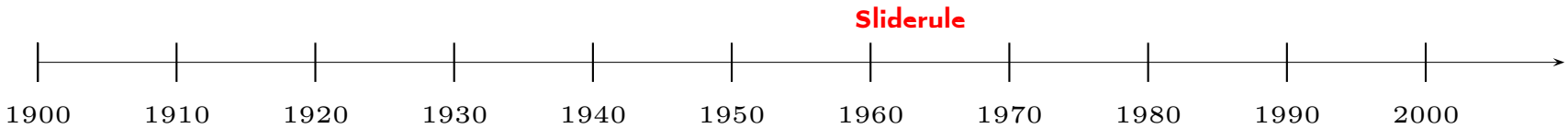
The First Bug



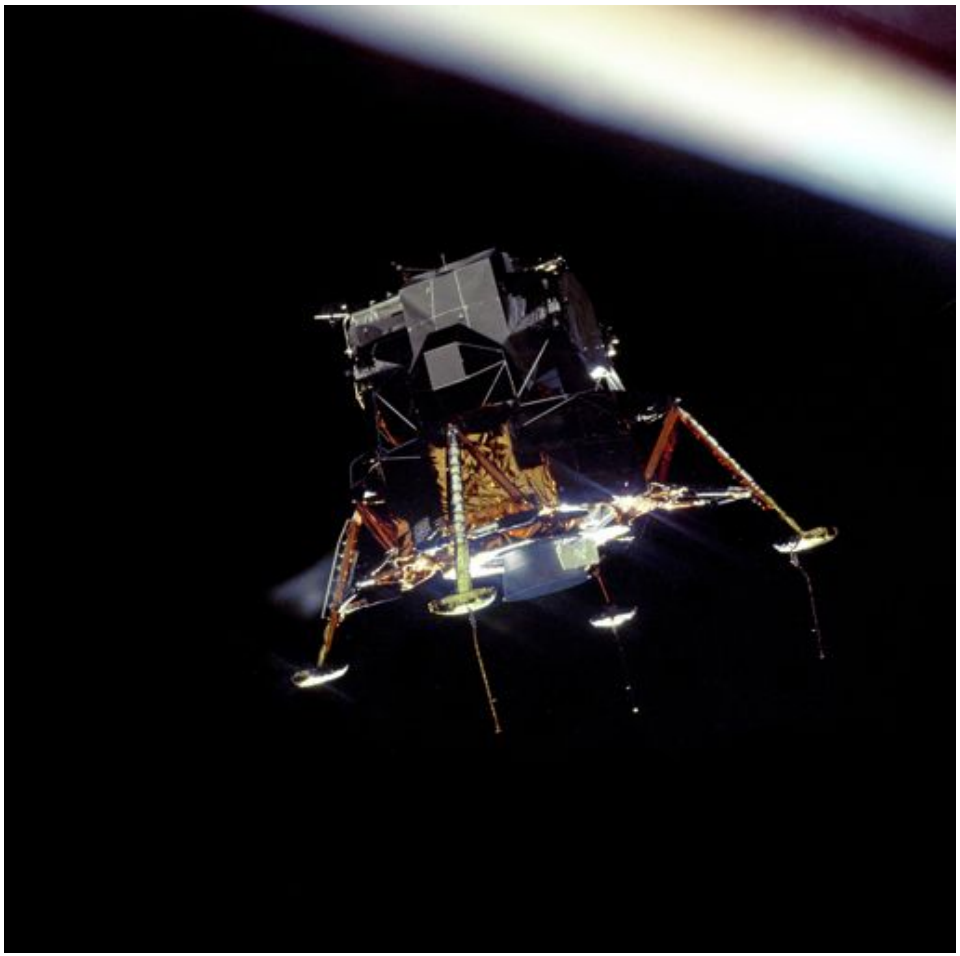


Analog Computers

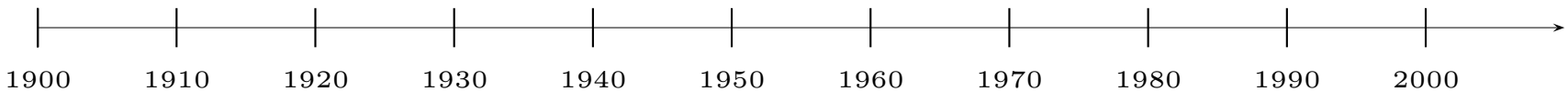




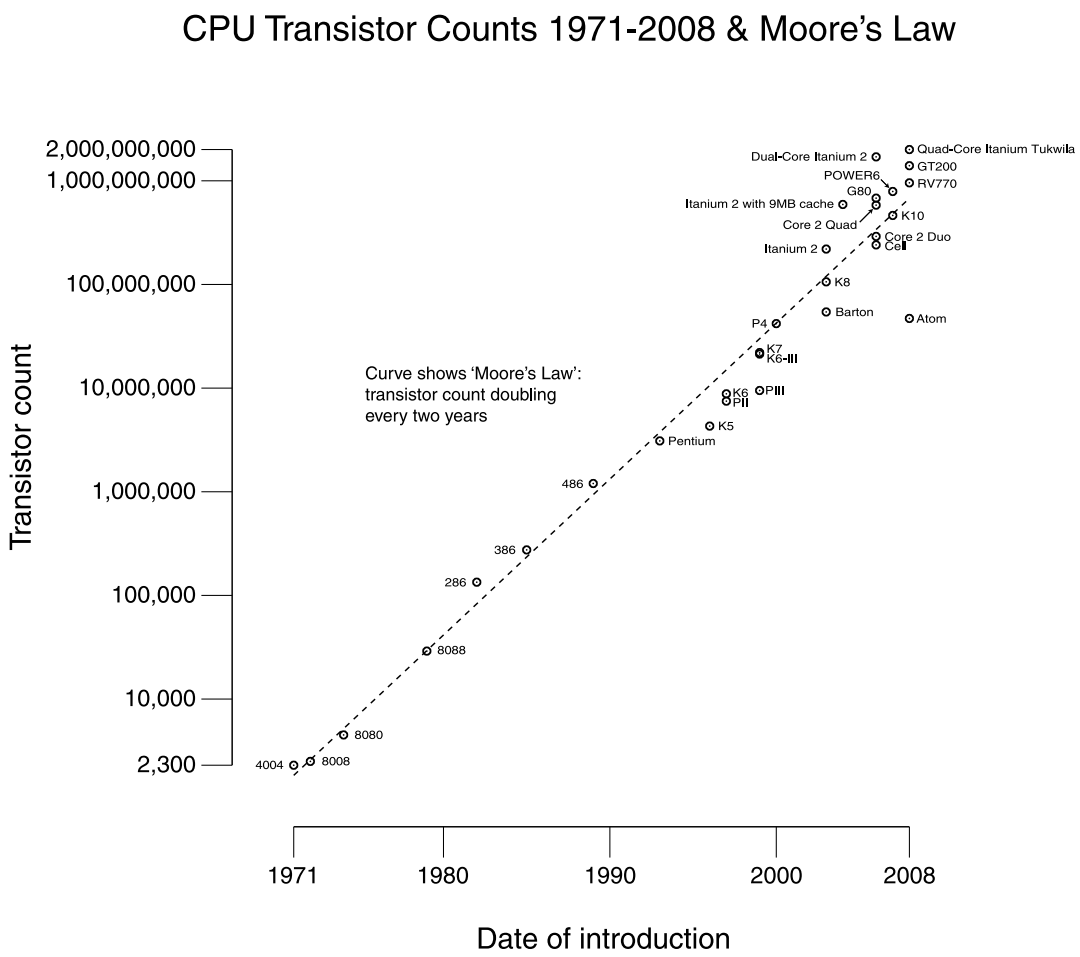
Other Analog Computers

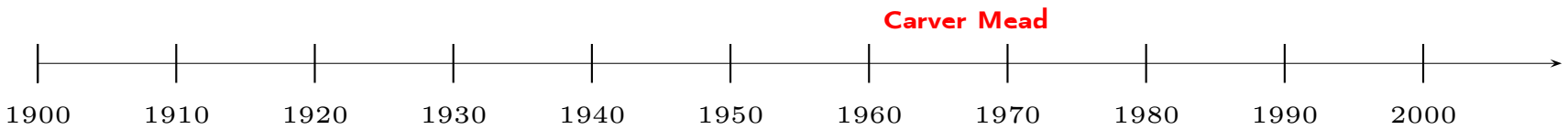


Apollo Lunar Module

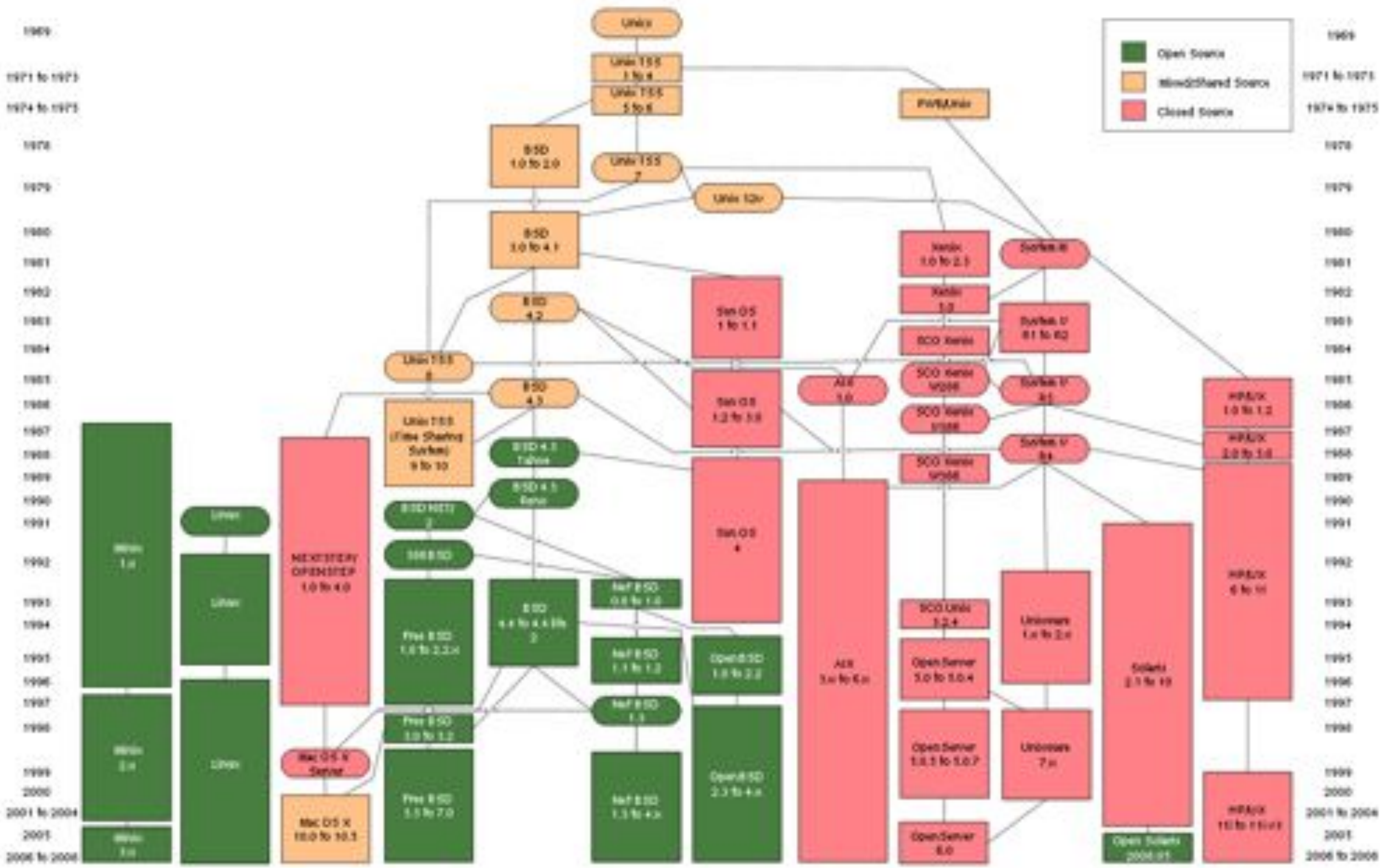


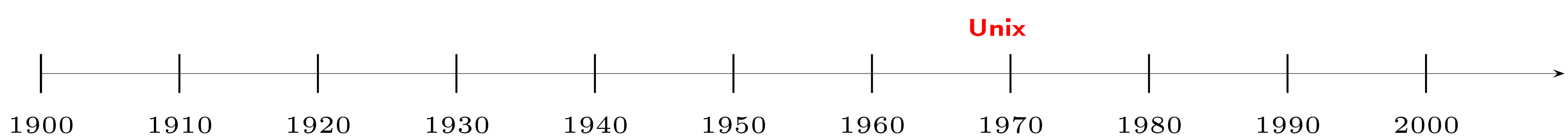
Moore's Law


CPU Transistor Counts 1971-2008 & Moore's Law
Transistor count
2,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
Curve shows 'Moore's Law':
transistor count doubling
every two years
Dual-Core Itanium 2
Quad-Core Itanium Tukwila
POWER6
GT200
G80
RV770
Itanium 2 with 9MB cache
K10
Core 2 Quad
Core 2 Duo
Itanium 2
K8
Barton
Atom
P4
K7
K6-III
K6
PII
PIII
K5
Pentium
486
386
286
8088
8080
4004
8008
1971
1980
1990
2000
2008
Date of introduction

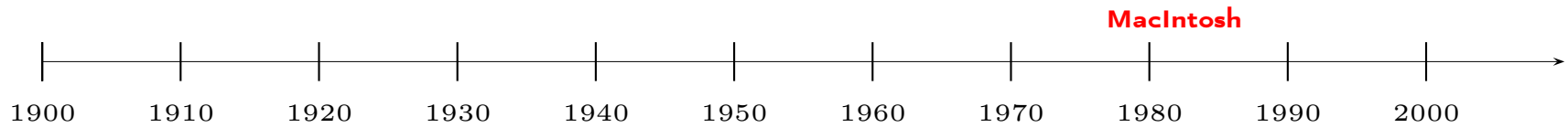

Carver Mead
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000

Unix [RT74]

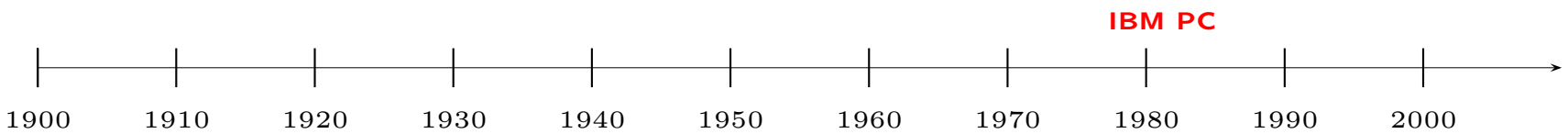




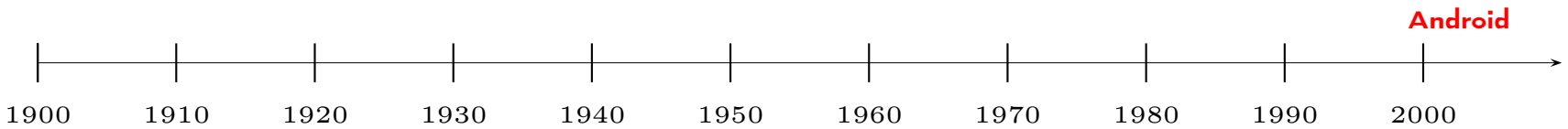
Macs


MacIntosh
1900
1910
1920
1930
1940
1950
1960
1970
1980
1990
2000

PCs

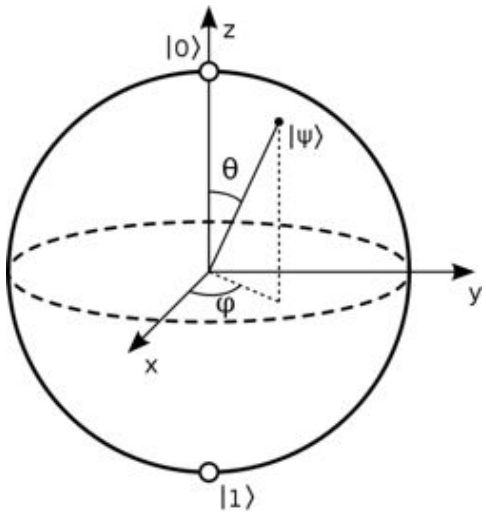


Ubiquitous Computers



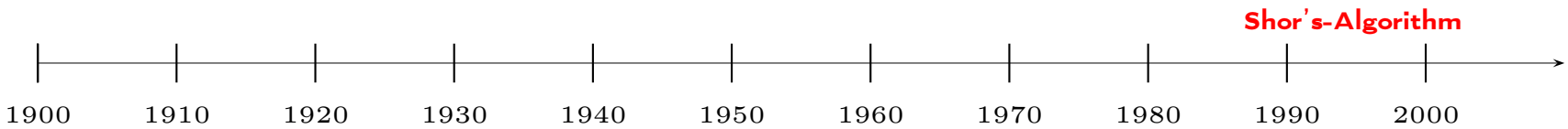
Quantum Computing

Quantum Computing:



Shor's-Algorithm [Sho97]:

$$q^{-1} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} e^{2\pi i a c/q} |c\rangle |x^a \bmod n\rangle$$



A Brief History of Programming Languages

Why are Languages Important?

Human language appears to be a unique phenomenon, without significant analogue in the animal world.

Noam Chomsky

Language is a process of free creation; its laws and principles are fixed, but the manner in which the principles of generation are used is free and infinitely varied. Even the interpretation and use of words involves a process of free creation.

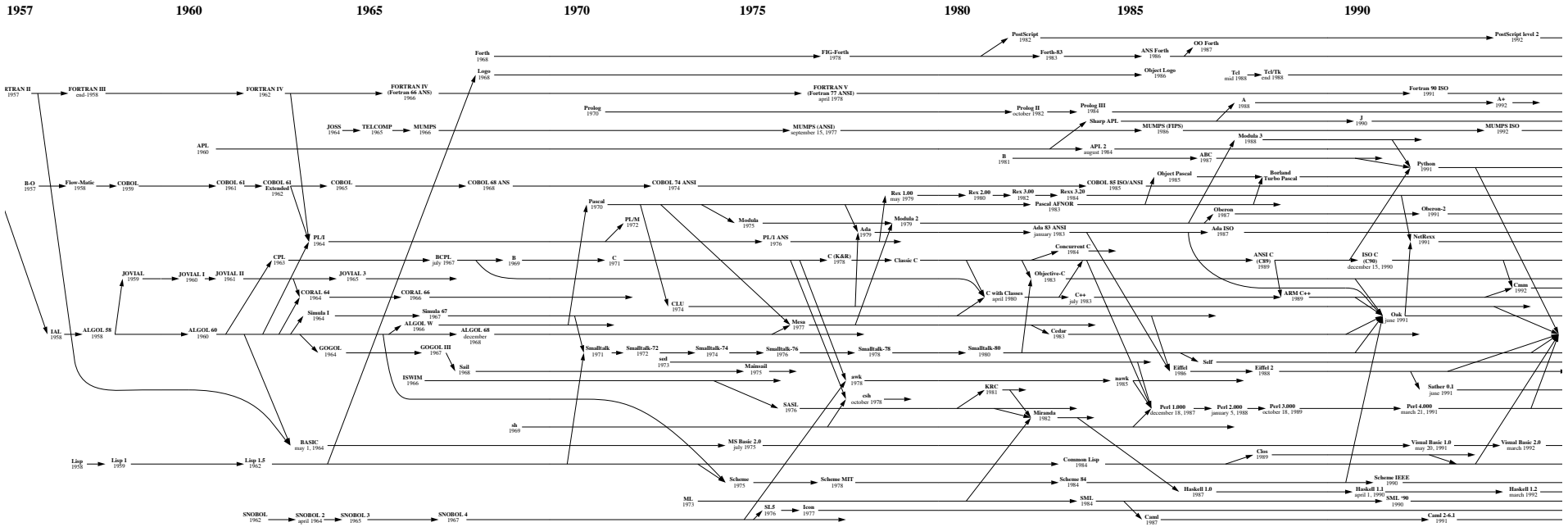
Noam Chomsky

Languages as Tools for Thinking

Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.

Ludwig Wittgenstein [Wit22]

Language Evolution



Messy!

1940s: It is War...

1940s cont.

- No languages or
- Machine Language (i.e. no language)

Machine Language

```
11000111101110101001010010010010101011100110101010011000001111001
10110101011111010100111111111110010110110000000010100100010010000
01100101011011000110110001101111001000000101011101101111011100101
01101100011001000010000101000010011011110110111111000111101110101
10010100100100101010111001101010100110000011110010110101011111101
01001111111111100101101100000000101001000100100001100101011011001
01101100011011110010000001010111011011110111001001101100011001001
00100001010000100110111101101111110001111011101010010100100100010
10101110011010101001100000111100101101010111110101001111111111110
01011011000000001010010001001000011001010110110001101100011011111
00100000010101110110111101110010011011000110010000100001010000010
01101111011011111100011110111010100101001001001010101110011010101
10011000001111001011010101111101010011111111111100101101100000000
10100100010010000110010101101100011011000110111110010000001010111
01101111011100100110110001100100110001111011101010010100100100010
10101110011010101001100000111100101101010111110101001111111111110
01011011000000001010010001001000011001010110110001101100011011111
```

1950s: Innovation Big Bang!

1950s cont.

- Assembly languages were first developed in the 1950s, when they were referred to as second generation programming languages. For example, SOAP (Symbolic Optimal Assembly Program) was a 1957 assembly language for the IBM 650 computer.
- Fortran, invented by John W. Backus optimized for speed (similar to assembly), used for numerical calculations in e.g. physics until today.
- Algol, the mother of procedural languages (such as Pascal)
- Lisp, the mother of functional languages

Fortran and Algol are *imperative* and *procedural*. Lisp is primarily *functional*.

Assembly

```
_partition:
LFB3:
        pushq   %rbp
LCFI0:
        movq    %rsp, %rbp
LCFI1:
        pushq   %rbx
LCFI2:
        movl    %edx, %r10d
        movslq  %edx,%rax
        leaq    (%rdi,%rax,4), %rbx
        movl    (%rbx), %r11d
        leal    -1(%rsi), %r9d
        cmpl    %esi, %edx
        jle L2
        movslq  %esi,%rax
        leaq    (%rdi,%rax,4), %rcx
        .align 4,0x90
L4:
        movl    (%rcx), %r8d
        cmpl    %r8d, %r11d
        jl  L5
        incl    %r9d
        movslq  %r9d,%rax
        leaq    (%rdi,%rax,4), %rax
        movl    (%rax), %edx
        movl    %edx, (%rcx)
...
```

Fortran – I

```
MODULE Qsort_Module

IMPLICIT NONE

CONTAINS

RECURSIVE SUBROUTINE Qsort(a)

  INTEGER, INTENT(IN OUT) :: a(:)
  INTEGER :: split

  IF(size(a) > 1) THEN
     CALL Partition(a, split)
     CALL Qsort(a(:split-1))
     CALL Qsort(a(split:))
  END IF

END SUBROUTINE Qsort
```

Fortran – II

```
SUBROUTINE Partition(a, marker)

  INTEGER, INTENT(IN OUT) :: a(:)
  INTEGER, INTENT(OUT) :: marker
  INTEGER :: left, right, pivot, temp

  pivot = (a(1) + a(size(a))) / 2  ! Average of first and last elements to prevent quadratic
  left = 0                         ! behavior with sorted or reverse sorted data
  right = size(a) + 1

  DO WHILE (left < right)
     right = right - 1
     DO WHILE (a(right) > pivot)
        right = right-1
     END DO
     left = left + 1
     DO WHILE (a(left) < pivot)
        left = left + 1
     END DO
     IF (left < right) THEN
        temp = a(left)
        a(left) = a(right)
        a(right) = temp
     END IF
  END DO

  IF (left == right) THEN
     marker = left + 1
  ELSE
     marker = left
  END IF

END SUBROUTINE Partition
END MODULE Qsort_Module
```

Algol68 – I

```
PROC partition =(REF [] DATA array, PROC (REF DATA, REF DATA) BOOL cmp)INT: (
  INT begin:=LWB array;
  INT end:=UPB array;
  WHILE begin < end DO
     WHILE begin < end DO
      IF cmp(array[begin], array[end]) THEN
        DATA tmp=array[begin];
        array[begin]:=array[end];
        array[end]:=tmp;
        GO TO break while decr end
      FI;
      end -:= 1
    OD;
    break while decr end: SKIP;
     WHILE begin < end DO
      IF cmp(array[begin], array[end]) THEN
        DATA tmp=array[begin];
         array[begin]:=array[end];
        array[end]:=tmp;
        GO TO break while incr begin
      FI;
      begin +:= 1
     OD;
     break while incr begin: SKIP
  OD;
  begin
);
```

Algol68 – II

```
PROC qsort=(REF [] DATA array, PROC (REF DATA, REF DATA) BOOL cmp)VOID: (
  IF LWB array < UPB array THEN
    INT i := partition(array, cmp);
    PAR ( # remove PAR for single threaded sort #
      qsort(array[:i-1], cmp),
      qsort(array[i+1:], cmp)
    )
  FI
);

MODE DATA = INT;
PROC cmp=(REF DATA a,b)BOOL: a>b;

main:(
  []DATA const l=(5,4,3,2,1);
  [UPB const l]DATA l:=const l;
  qsort(l,cmp);
  printf(($g(3)$,l))
)
```

Lisp

```
(defun qs (list)
  (if (<= (length list) 1)
    list
    (let ((pivot (first list)))
      (append (qs (remove-if-not #'(lambda (x) (< x pivot)) list))
              (remove-if-not #'(lambda (x) (= x pivot)) list)
              (qs (remove-if-not #'(lambda (x) (> x pivot)) list))))))
```

Content (Possibly) Copyright Protected

1960s: Swinging London (but not in IT)!

1960s

- Basic was developed by John George Kemeny and Thomas Eugene Kurtz at the Dartmouth College
- COBOL

Both are *imperative* and *procedural*.

Basic

```
Procedure qSort(Array a(1), firstIndex, lastIndex)
  Protected  low, high, pivotValue

  low = firstIndex
  high = lastIndex
  pivotValue = a((firstIndex + lastIndex) / 2)

  Repeat

    While a(low) < pivotValue
      low + 1
    Wend

    While a(high) > pivotValue
      high - 1
    Wend

    If low <= high
      Swap a(low), a(high)
      low + 1
      high - 1
    EndIf

  Until low > high

  If firstIndex < high
    qSort(a(), firstIndex, high)
  EndIf

  If low < lastIndex
    qSort(a(), low, lastIndex)
  EndIf
EndProcedure
```

Cobol

Implementation is not shown, because

1. Cobol is too ugly
2. Originally, you could not implement recursion in Cobol at all (it is a language for business people, after all)

1970s: Revolution – New Frontiers!

1970s

- C, developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system. The one and only systems language. Imperative and procedural, a better high level assembly.
- Prolog , invented by Alain Colmeraue, is a general purpose declarative logic programming language mainly used for artificial intelligence and computational linguistics.
- Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Scott Wallace, and other created Smalltalk-80 at Xerox PARC, the mother of all (good) object-oriented languages, based on Simula (from the 60s!)[1]

[1] So OO is indeed a very old technology.

Hackers

C

```
void quick(int *left, int *right)
{
  if (right > left) {
    int pivot = left[(right-left)/2];
    int *r = right, *l = left;
    do {
      while (*l < pivot) l++;
      while (*r > pivot) r--;
      if (l <= r) {
        int t = *l;
        *l++ = *r;
        *r-- = t;
      }
    } while (l <= r);
    quick(left, r);
    quick(l, right);
  }
}

void sort(int *array, int length)
{
  quick(array, array+length-1);
}
```

Prolog

```
sort( [], [] ).
qsort( [X], [X] ).
qsort( [H|U], S ) :- splitBy(H, U, L, R), qsort(L, SL), qsort(R, SR),
                     combine(H, SL, SR, S).

% splitBy( H, U, LS, RS )
% True if LS = { L in U | L <= H }; RS = { R in U | R > H }
splitBy( H, [], LS, RS).
splitBy( H, [U|T], [U|LS], RS ) :- U =< H, splitBy(H, T, LS, RS).
splitBy( H, [U|T], LS, [U|RS] ) :- U  > H, splitBy(H, T, LS, RS).

% combine( H, L, R, S )
% True if S is L ++ [H] ++ R (in Haskell notation)
combine( H, L, R, S ) :- append(L, [H|R], S).
```

Smalltalk

```
" This must be inserted into the class Array as an instance method "

quicksort: l to: r
" sorts the array between l and r after the quicksort method
  by Michael Neumann 1998
"
| m il ir temp |
   (r > l) ifTrue: [
      m ← self at: ((l + r) // 2).
      il ← l.
      ir ← r.
      [
         [(self at: il) < m] whileTrue: [il ← il + 1.].
         [(self at: ir) > m] whileTrue: [ir ← ir - 1.].
         (il < ir) ifTrue: [
            " swap "
            temp ← self at: il.
            self at: il put: (self at: ir).
            self at: ir put: temp.
         ].
      ] doUntil: [il >= ir.].
      self quicksort: l to: (il - 1).
      self quicksort: (il + 1) to: r.
   ].
↑self.
```

1980s: OO is in the Air!

1980s

- C++ was invented as a better (really?!) C by Bjarne Stroustrup. It is a multi-paradigm language supporting procedural programming and OO concepts.
- Guido van Rossum created Python, a modern script language with functional and OO elements.

C++ – I

```
#include <iterator>
#include <algorithm> // for std::partition
#include <functional> // for std::less

// helper function for median of three
template<typename T>
 T median(T t1, T t2, T t3)
{
  if (t1 < t2)
  {
    if (t2 < t3)
      return t2;
    else if (t1 < t3)
      return t3;
    else
      return t1;
  }
  else
  {
    if (t1 < t3)
      return t1;
    else if (t2 < t3)
      return t3;
    else
      return t2;
  }
}
```

C++ – II

```
// helper object to get <= from <
template<typename Order> struct non_strict_op:
  public std::binary_function<typename Order::second_argument_type,
                              typename Order::first_argument_type,
                              bool>
{
  non_strict_op(Order o): order(o) {}
  bool operator()(typename Order::second_argument_type arg1,
                  typename Order::first_argument_type arg2) const
  {
    return !order(arg2, arg1);
  }
private:
  Order order;
};

template<typename Order> non_strict_op<Order> non_strict(Order o)
{
  return non_strict_op<Order>(o);
}

...

template<typename RandomAccessIterator>
 void quicksort(RandomAccessIterator first, RandomAccessIterator last)
{
  quicksort(first, last,
    std::less<typename std::iterator_traits<RandomAccessIterator>::value_type>());
}
```

Python

```
def qsort(list):
    if not list:
        return []
    else:
        pivot = list[0]
        less = [x for x in list      if x <  pivot]
        more = [x for x in list[1:] if x >= pivot]
        return qsort(less) + [pivot] + qsort(more)
```

1990s: Evolutionary Improvements!

1990s cont.

http://info.cern.ch/

1990s cont.

- Haskell, named after logician Haskell Curry is a standardized, general-purpose purely functional programming language, with non-strict semantics and strong static typing. “A proof is a program; the formula it proves is a type for the program”. Designed by Simon Peyton Jones, Paul Hudak, Philip Wadler, et al. Used in the research community.
- Java invented by SUN (James Gosling, Mike Sheridan, and Patrick Naughton), is a general purpose OO language with C++ syntax.

Haskell

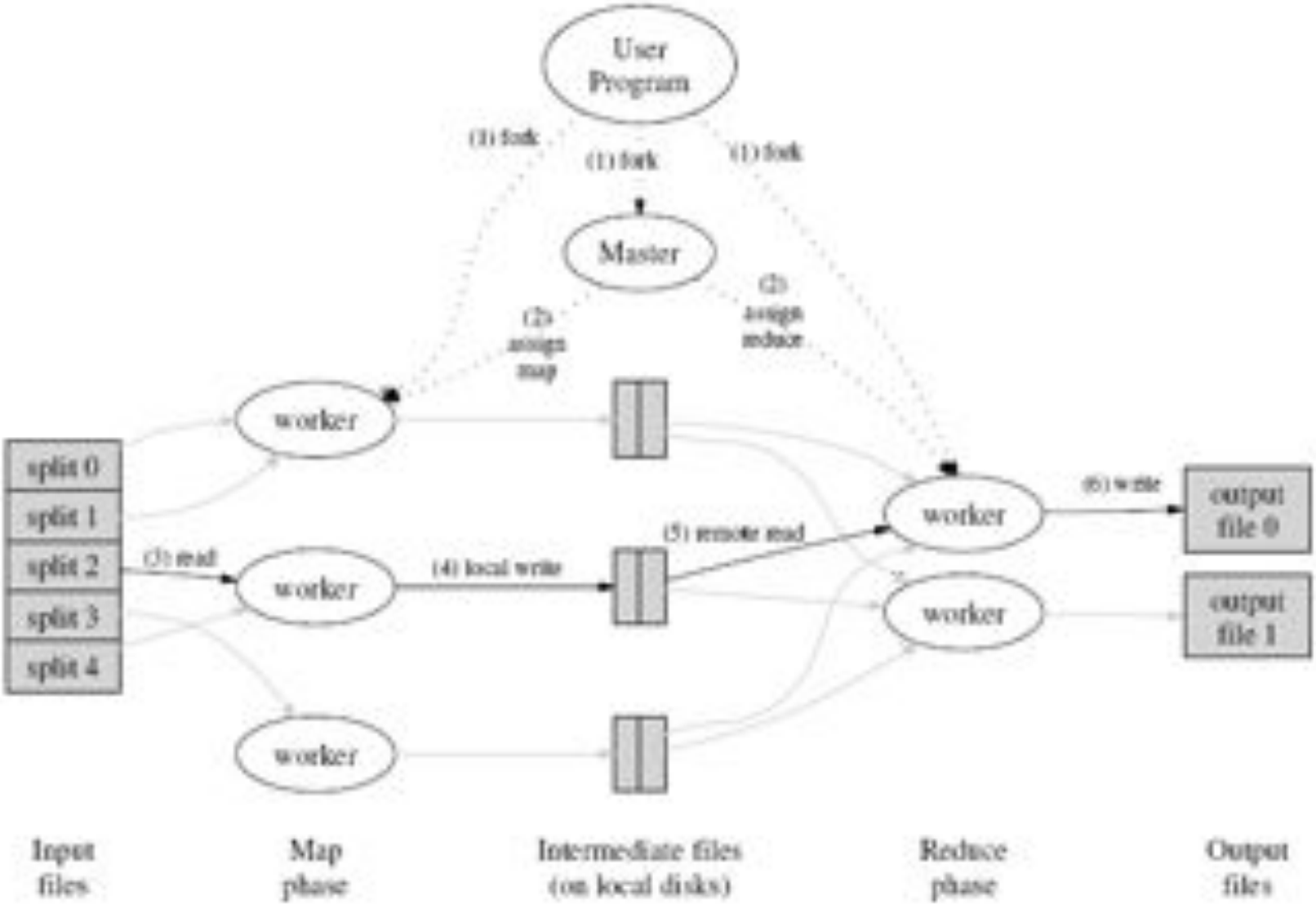
```
qsort :: Ord a => [a] -> [a]
qsort [] = []
qsort (p:xs) = qsort lesser ++ [p] ++ qsort greater
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

Java

```
public static <E extends Comparable<? super E>> List<E> quickSort(List<E> arr) {
    if (arr.size() <= 1)
        return arr;
    E pivot = arr.getFirst(); //This pivot can change to get faster results

    List<E> less = new LinkedList<E>();
    List<E> pivotList = new LinkedList<E>();
    List<E> more = new LinkedList<E>();

    // Partition
    for (E i: arr) {
        if (i.compareTo(pivot) < 0)
            less.add(i);
        else if (i.compareTo(pivot) > 0)
            more.add(i);
        else
            pivotList.add(i);
    }

    // Recursively sort sublists
    less = quickSort(less);
    more = quickSort(more);

    // Concatenate results
    less.addAll(pivotList);
    less.addAll(more);
    return less;
}
```

2000s: Clouds...

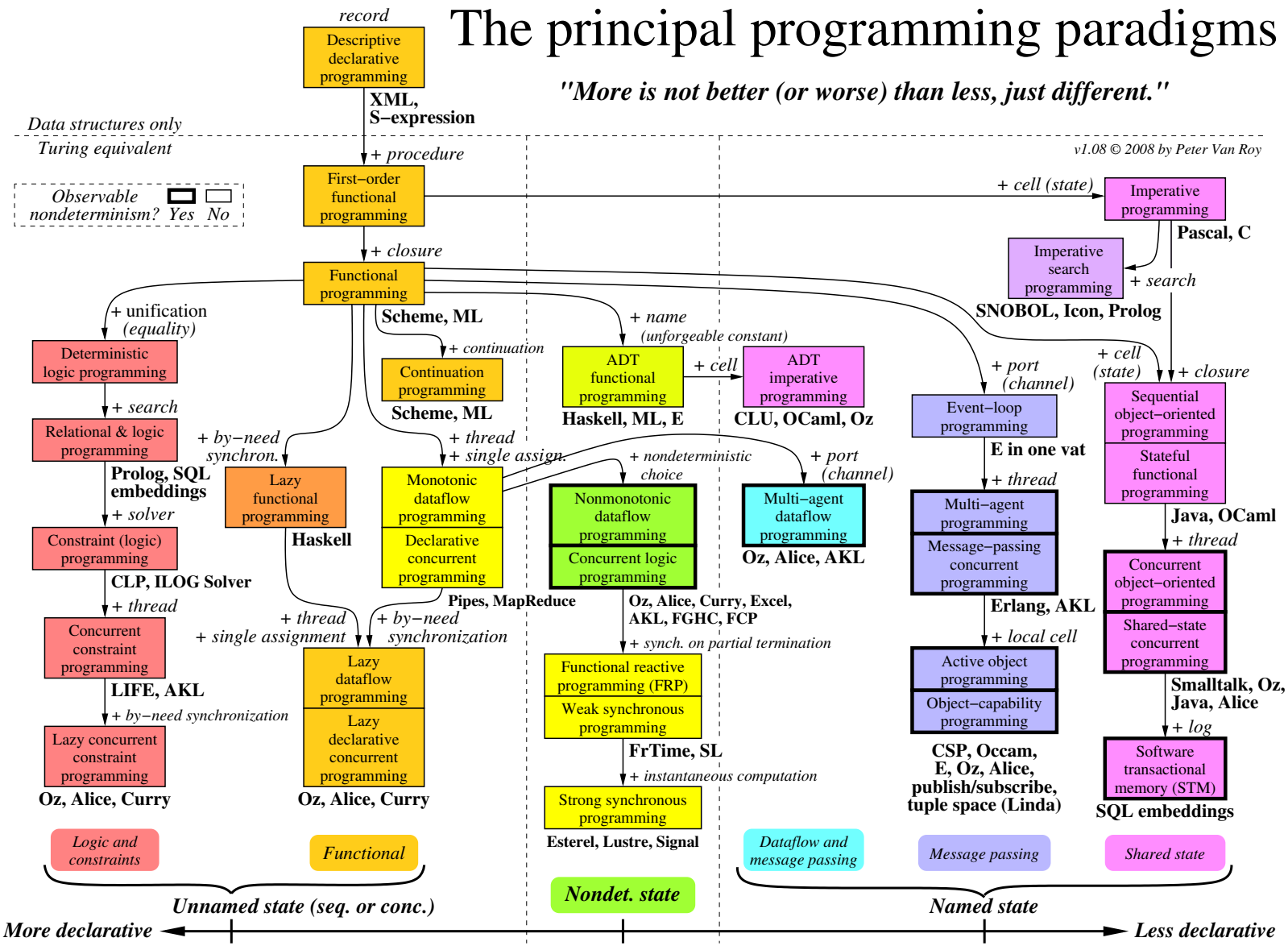
MapReduce


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Summary: Programming Languages' Paradigms

- Imperative: Computation in terms of program state and statements that change the program state. Specifies steps the program must take to reach the desired state. Examples: Assembly, C, C++, Java
- Declarative: Describes *what* rather than *how*. Examples: HTML, SQL, Prolog
- Procedural: A subset of imperative languages consisting of procedures (aka routines, subroutines, methods, or functions), each containing a series of computational steps to be carried out. Examples: C, C++, Java
- Functional: Computation is the evaluation of mathematical functions. Avoids state and mutable data. Examples: Lisp, ML, Haskell
- Constraint: Relations between variables are stated in the form of constraints. Constraints do not specify a sequence of steps to execute, but rather the properties of a solution to be found. Searches for a state of the world in which a number of constraints are satisfied at the same time. Examples: Prolog
- Logic: Pattern-directed invocation of procedures from assertions and goals. When a set of predicates is satisfied, an action is taken. Examples: Prolog

Interpretation of Paradigms



Shooting Yourself

How to Shoot Yourself In the Foot Using Any Programming Language:

- **C:** You shoot yourself in the foot and then nobody else can figure out what you did.
- **C++:** You accidentally create a dozen clones of yourself and shoot them all in the foot. Emergency medical assistance is impossible since you can't tell which are bitwise copies and which are just pointing at others and saying, "That's me, over there."
- **Java:** After importing java.awt.right.foot.* and java.awt.gun.right.hand.*, and writing the classes and methods of those classes needed, you've forgotten what the hell you're doing.
- **Haskell:** You shoot yourself in the foot very elegantly, and wonder why the whole world isn't shooting itself this way.
- **Lisp:** You shoot yourself in the appendage which holds the gun with which you shoot yourself in the appendage which holds the gun with which you shoot yourself in the appendage which holds the gun with which you shoot...
- **Smalltalk:** You shoot yourself in the foot and your foot sends doesNotUnderstand: Pain to your brain.
- **XML:** You can't actually shoot yourself in the foot; all you can do is describe the gun in painful detail.

(Sources: `http://www.toodarkpark.org/computers/humor/shoot-self-in-foot.html` **and** `http://thealmightyguru.com/Humor/Docs/ShootYourselfInTheFoot.html`**)**


Java
C
PHP
Ruby
Haskell
Lisp
as seen by...
Java fans
C fans
PHP fans
Ruby fans
Haskell fans
Lisp fans

Languages as Tools for Programming

A language that doesn't affect the way you think about programming, is not worth knowing.

Alan Perlis

A Brief History of Networks

The World is Connected

The network is the computer

Scott McNealy, co-founder of Sun Microsystems

The World is Connected

The network is the computer

Scott McNealy, co-founder of Sun Microsystems

Unfortunately, this is the topic for another lecture, please ask Prof. Hoffmann for next year's Xmas lecture!

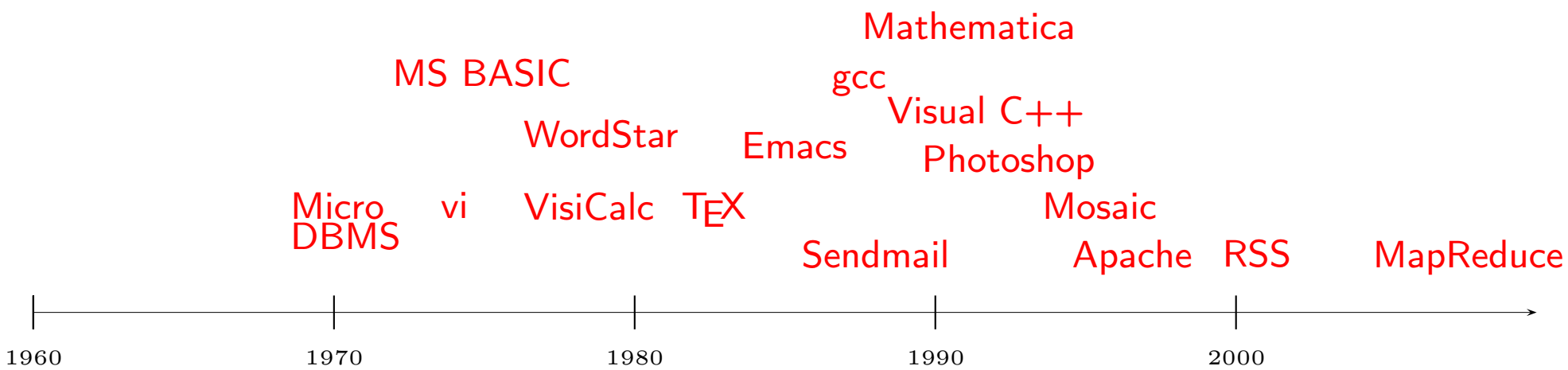
The Power of the Network

213.251.145.96

A Brief History of Applications

Important Applications

Warning: The following choice is ridiculously subjective!



A Brief History of Computer Scientists and Engineers

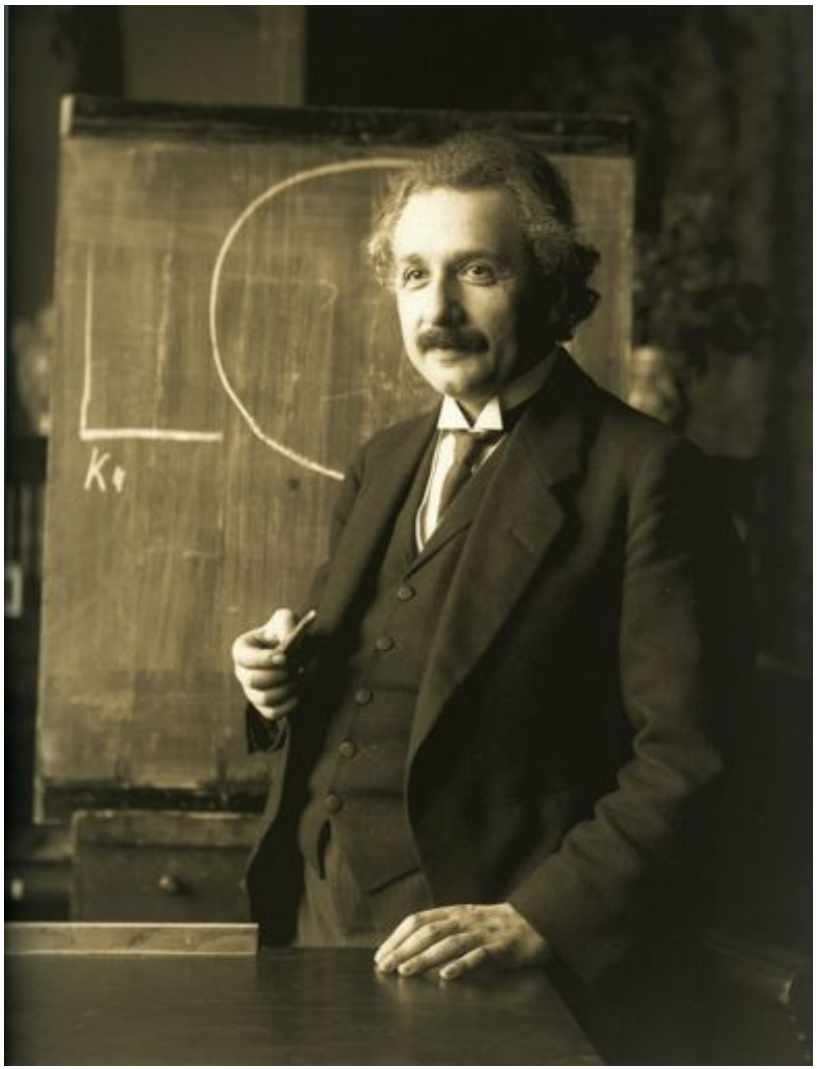
Famous Computer Scientists and Engineers

Gottfried Wilhelm Leibniz Charles Babbage Ada Lovelace John von Neumann Alan Turing Kurt Gödel John Atanasoff Alonzo Church Claude Shannon Konrad Zuse Vannevar Bush C. A. R. Hoare Stephen Cole Kleene Edgar F. Codd John Backus Marvin Minsky Barbara Liskov Ron Rivest Adi Shamir Leonard Adleman Dennis M. Ritchie Alan Kay Bill Joy Peter Naur Ken Thompson Gordon E. Moore Fred Brooks Niklaus Wirth Donald Knuth Linus Torvalds Andrew S. Tanenbaum Eric S. Raymond Steve Jobs Richard M. Stallman William Gates III Joseph Weizenbaum Tim Berners-Lee...

A Brief History of Challenges

Different Disciplines

Engineering



Science

Speaking Different Languages – Example I

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!
SOFTWARE ENGINEER: ?

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Aren't both Turing complete?

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Aren't both Turing complete?
SOFTWARE ENGINEER: ?

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Aren't both Turing complete?
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, I guess they both support loops or recursion, conditional statements and the like?

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Aren't both Turing complete?
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, I guess they both support loops or recursion, conditional statements and the like?
SOFTWARE ENGINEER: Well, yes, of course!

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Aren't both Turing complete?
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, I guess they both support loops or recursion, conditional statements and the like?
SOFTWARE ENGINEER: Well, yes, of course!
COMPUTER SCIENTIST: Then it does not matter!

Speaking Different Languages – Example I

SOFTWARE ENGINEER: Shall I use C++ or Lisp?
COMPUTER SCIENTIST: Does not matter!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Aren't both Turing complete?
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, I guess they both support loops or recursion, conditional statements and the like?
SOFTWARE ENGINEER: Well, yes, of course!
COMPUTER SCIENTIST: Then it does not matter!
SOFTWARE ENGINEER: WTF?!

Speaking Different Languages – Example II

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?
COMPUTER SCIENTIST: You do not understand - there is no state!

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?
COMPUTER SCIENTIST: You do not understand - there is no state!
SOFTWARE ENGINEER: So, please – how do I implement input/output?

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?
COMPUTER SCIENTIST: You do not understand - there is no state!
SOFTWARE ENGINEER: So, please – how do I implement input/output?
COMPUTER SCIENTIST: You have to use *monads*!

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?
COMPUTER SCIENTIST: You do not understand - there is no state!
SOFTWARE ENGINEER: So, please – how do I implement input/output?
COMPUTER SCIENTIST: You have to use *monads*!
SOFTWARE ENGINEER: For goodness' sake, what is a *monad*?

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?
COMPUTER SCIENTIST: You do not understand - there is no state!
SOFTWARE ENGINEER: So, please – how do I implement input/output?
COMPUTER SCIENTIST: You have to use *monads*!
SOFTWARE ENGINEER: For goodness' sake, what is a *monad*?
COMPUTER SCIENTIST: A monad is a monoid in the category of endofunctors, what's the problem?

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?
COMPUTER SCIENTIST: You do not understand - there is no state!
SOFTWARE ENGINEER: So, please – how do I implement input/output?
COMPUTER SCIENTIST: You have to use *monads*!
SOFTWARE ENGINEER: For goodness' sake, what is a *monad*?
COMPUTER SCIENTIST: A monad is a monoid in the category of endofunctors, what's the problem?
SOFTWARE ENGINEER: WTF?!

Speaking Different Languages – Example II

SOFTWARE ENGINEER: Haskell does not have side effects?
COMPUTER SCIENTIST: Haskell is a pure functional language!
SOFTWARE ENGINEER: ?
COMPUTER SCIENTIST: Sigh, a pure functional language does not have *any* side effects, so it was designed!
SOFTWARE ENGINEER: So I cannot change state?
COMPUTER SCIENTIST: You do not understand - there is no state!
SOFTWARE ENGINEER: So, please – how do I implement input/output?
COMPUTER SCIENTIST: You have to use *monads*!
SOFTWARE ENGINEER: For goodness' sake, what is a *monad*?
COMPUTER SCIENTIST: A monad is a monoid in the category of endofunctors, what's the problem?
SOFTWARE ENGINEER: WTF?!

We teach computer science, but the industry desperately needs (software) engineers [Par97]. The effect is roughly the same as it would be if you assigned a physics Ph.D. to design electrical equipment!

What can we Compute?

Two important questions:

1. "What does it mean for a function from the natural numbers to themselves to be computable?"
2. "Can noncomputable functions be classified into a hierarchy based on their level of noncomputability?"

Related to the Church–Turing thesis, which states that any function that is computable by an algorithm is a computable function.

Big open question:

$$P = NP?$$

Further aspects:

> "Can (quantum) physics be (efficiently) simulated by (classical) computers?" Richard Feynman (1981)

This led to theory of quantum computers.

How to deal with Complexity?

Politically correct answer:

How to deal with Complexity?

Politically correct answer:

Software Engineering!

How to deal with Complexity?

Politically correct answer:

Software Engineering!

The truth is:

How to deal with Complexity?

Politically correct answer:

Software Engineering!

The truth is:

We still have no clue!

Outlook

The Future of Computing

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

Alan Kay

Appendix

References

[Neu45] John von Neumann. First draft of a report on the edvac. *IEEE Ann. Hist. Comput.*, 15(4):27–75, October 1945.

[Par97] David Lorge Parnas. Software engineering (extended abstract): an unconsummated marriage. In *Proceedings of the 6th European SOFTWARE ENGINEERING conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*, ESEC '97/FSE-5, pages 1–3, New York, NY, USA, 1997. Springer-Verlag New York, Inc.

[RT74] D. M. Ritchie and K. Thompson. The unix time-sharing system. *Communications of the ACM*, 17:365–375, 1974.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26:1484–1509, October 1997.

[Tur01] Alan M. Turing. *Collected works of A. M. Turing. Mathematical logic*. Elsevier, Amsterdam, The Netherlands, 2001. Edited by R. O. Gandy and C. E. M. Yates. Including prefaces by Solomon Feferman.

[Wit22] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Kegan Paul, Trench, Trubner & Co., 1922.

Credits – I/III

- **Stonehenge**: ©2009 David Ball – www.davidball.net
- **Abacus**: public domain
- **Al-Chwarizmi**: John L. Esposito. The Oxford History of Islam. Oxford University Press, copyright expired
- **Da Vinci**: ©IBM
- **Pascaline**: CC 2005 David Monniaux
- **Leibniz**: Page of the Article "Explication de l'Arithmétique Binaire", 1703/1705, copyright expired
- **Babbage Diff. Engine**: CC 2005 Carsten Ullrich
- **Babbage Ana. Engine**: CC 2009 Bruno Barral
- **Ada Lovelace**: Margaret Carpenter (1793-1872), copyright expired
- **Steampunk**: ©2007 Jake von Slatt
- **Power (Jacquard) Looms**: ©2009 Lutz Marx
- **IBM**: public domain
- **Zuse**: CC ComputerGeek, public domain
- **Alan Turing**: ©1951 National Portrait Gallery, London, http://www.npg.org.uk/
- **Von Neumann**: public domain
- **Von Neumann Architektur**: CC 2007 Lukas Grossar

Credits – II/III

- **Grace Hopper**: **public domain**
- **Bug**: **public domain**
- **Sliderule**: **Creative Commons Attribution-Share Alike 3.0 Unported**
- **Lunar Module**: **public domain**
- **Moore's Law**: **Creative Commons Attribution-Share Alike 3.0 Unported**
- **Unix**: **Creative Commons Attribution-Share Alike 3.0 Unported**
- **MacIntosh**: **Creative Commons Attribution-Share Alike 2.5 Italy** http://www.allaboutapple.com/
- **PC**: **Creative Commons Attribution-Share Alike 3.0 Unported**
- **Android**: **Creative Commons Attribution-Share Alike 3.0**
- **Quantum Computing**: **Smite-Meister, Creative Commons Attribution-Share Alike 3.0 Unported**
- **Programming Languages ©1999-2010 Éric Lévénez** http://www.levenez.com/lang/
- **Bletchley Park**: **Matt Crypto, public domain**
- **1950s**: **2006 Hans Weingartz** http://www.pass-weingartz.de/hw.htm **CC-by-sa 2.0/de**
- **Fortran and all further programming examples taken from** http://rosettacode.org/wiki/Sorting_algorithms/Quicksort
- **Twiggy**: **©2001-2005** www.twiggylawson.co.uk
- **Apollo 11**: **public domain**

Credits – III/III

- **Thompson Ritchie: public domain**
- **Berlin Wall: © Unknown photographer, Creative Commons-Lizenz Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Unported**
- **Bill Clinton, Yitzhak Rabin, Yasser Arafat at the White House: public domain**
- **Clouds: ©2008 Jörg Schäfer**
- **Languages: Source** `http://i.imgur.com/1gF1j.jpg`**, copyright unknown**
- **MapReduce: ©2004, Jeffrey Dean and Sanjay Ghemawat, Google Inc.**
- **Principles Programming Paradigms: ©2007 Peter Van Roy** `http://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng.pdf`
- **Engineers:** `http://s882.photobucket.com/albums/ac30/Pvt_Hawkins/`
- **Einstein: ©1921, Ferdinand Schmutzer, copyright expired**