



founderisis.com

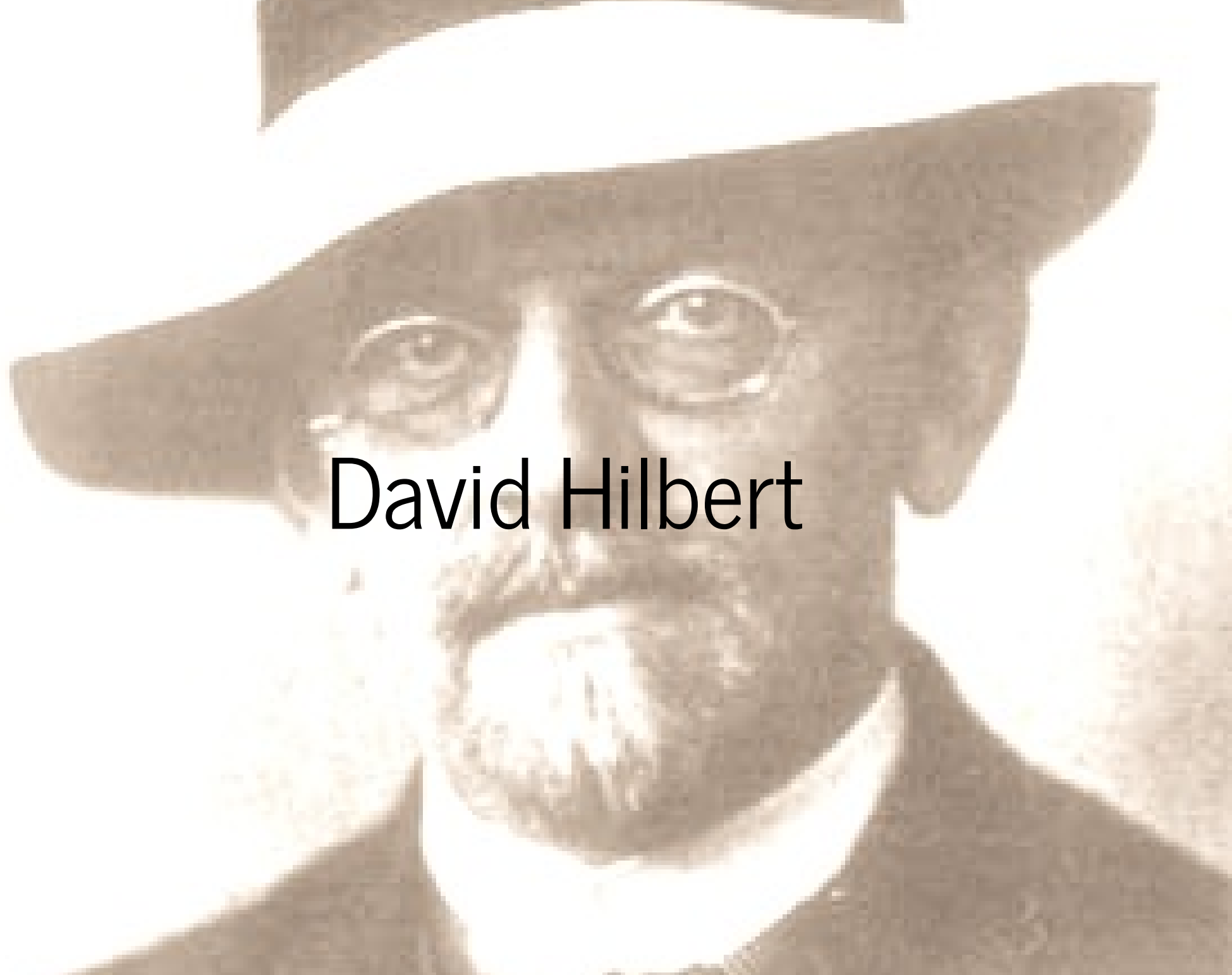
Andrea Magnorsky

Scala Exchange



A brief (and incomplete) history
of programming languages

Scala Exchange - December 2016



David Hilbert

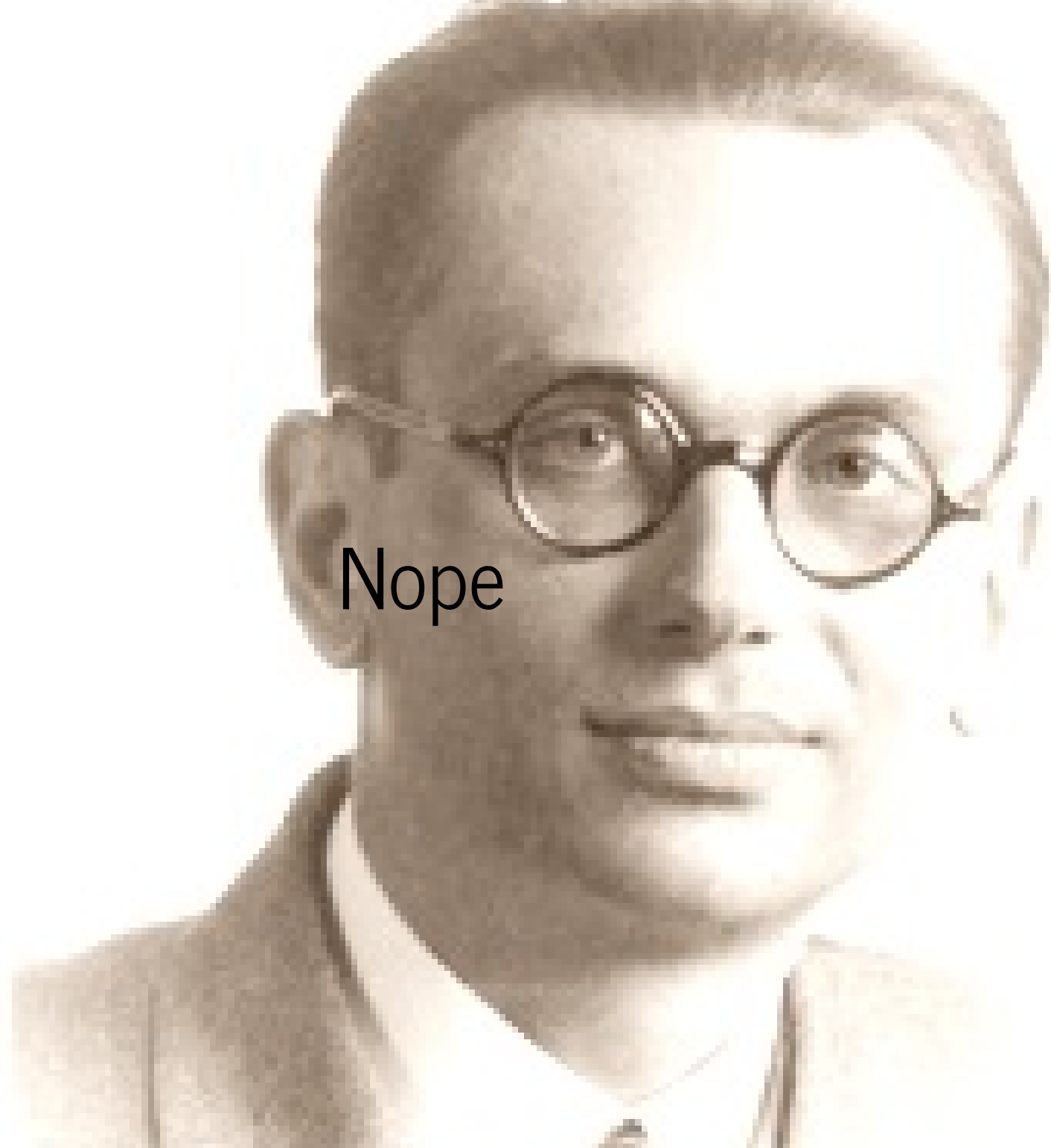
We must know. We will know.

I HAZ A QUESTION



1903.jpg

Can we devise a process to determine in a finite number of operations, whether a first order logic statement is valid?



Nope

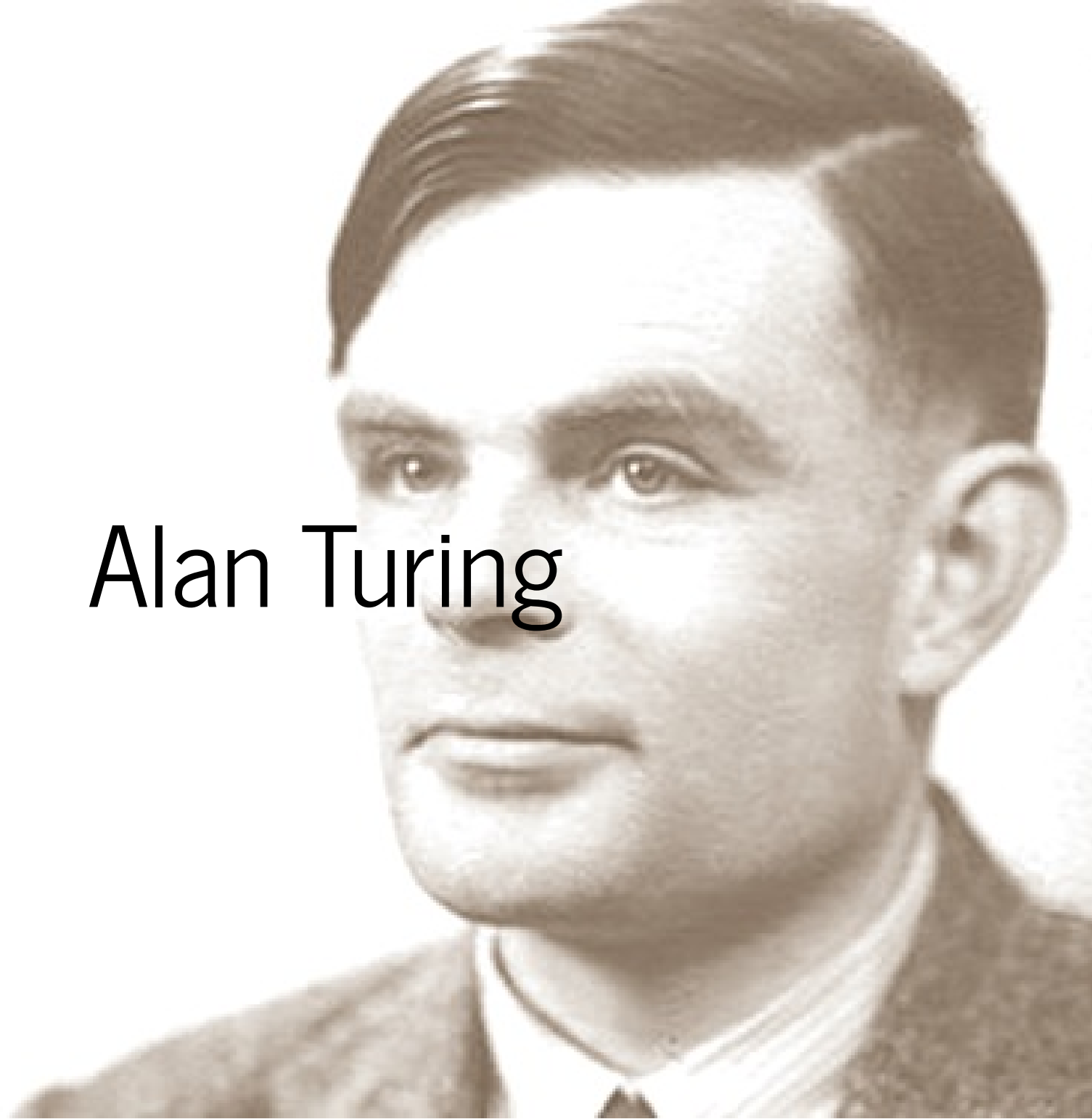
Kurt Gödel

Alonzo Church



λ Calculus

Alan Turing



Turing Machine




sauce

Church-Turing Thesis



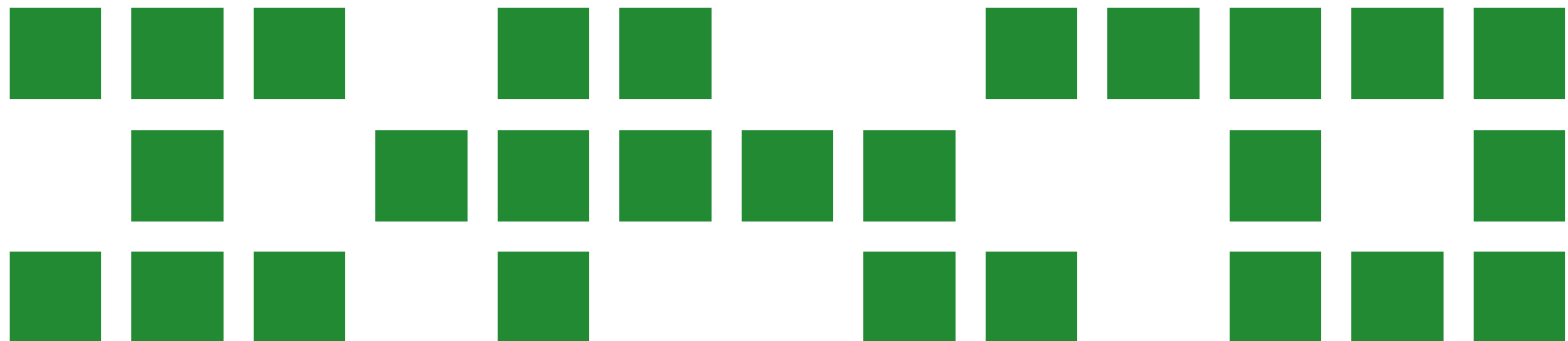
World war II

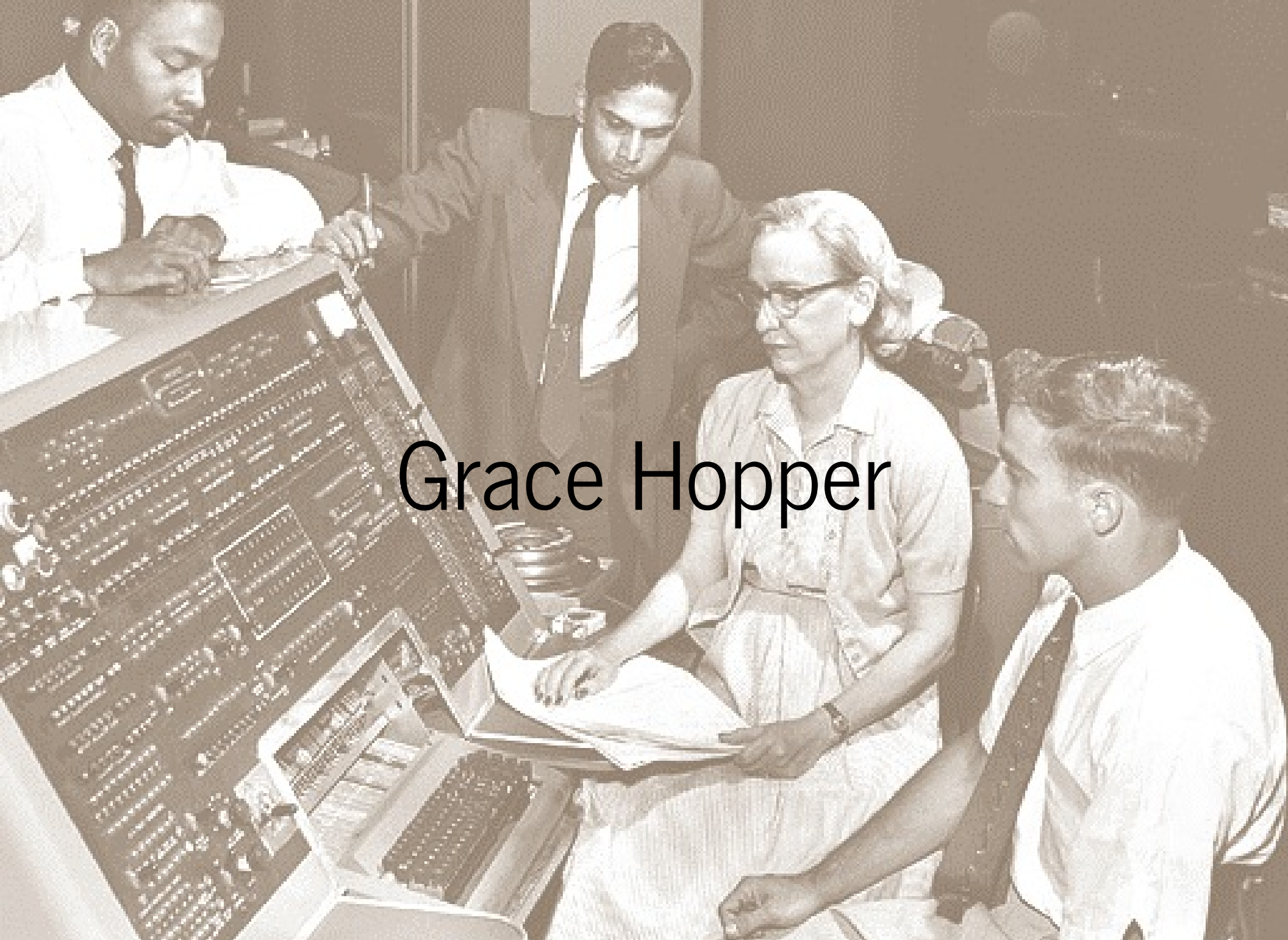
A photograph of five men standing in a room filled with large, complex electronic equipment, which is the Colossus computer. The man on the far left is older, wearing a brown hat, a red shirt, and a dark jacket. The other four men are younger, wearing various shirts and sweaters. They are all looking towards the camera. The background shows the intricate wiring and components of the Colossus machine, which is a large, multi-story structure made of metal frames and electronic components. The room has large windows on the right side, letting in natural light.

Colossus

THE NATIONAL MUSEUM

The National Museum of Computing





Grace Hopper

The first compiler: A-0

FLOW-MATIC

```
1: 0) INPUT INVENTORY FILE=A
2: PRICE FILE=B,
3: OUTPUT PRICED-INV FILE=C
4: UNPRICED-INV FILE=D,
5: HSP D.
6: 1) COMPARE PRODUCT-NO(A) WITH PRODUCT-NO(B)
7: IF GREATER GO TO OPERATION 10;
8: IF EQUAL GO TO OPERATION 5;
9: OTHERWISE GO TO OPERATION 2.
10: 2) TRANSFER A TO D.
11: 3) WRITE ITEM D.
12: 4) JUMP TO OPERATION 8.
13: 5) TRANSFER A TO C.
```

```
1:  6) MOVE UNIT-PRICE(B) TO UNIT-PRICE(C).
2:  7) WRITE ITEM C.
3:  8) READ ITEM A; IF END OF DATA GO TO OPERATION 14.
4:  9) JUMP TO OPERATION 1.
5: 10) READ ITEM B; IF END OF DATA GO TO OPERATION 12.
6: 11) JUMP TO OPERATION 1.
7: 12) SET OPERATION 9 TO GO TO OPERATION 2.
8: 13) JUMP TO OPERATION 2.
9: 14) TEST PRODUCT-NO(B) AGAINST ZZZZZZZZZZZZ;
10: IF EQUAL GO TO OPERATION 16;
11: OTHERWISE GO TO OPERATION 15.
12: 15) REWIND B.
13: 16) CLOSE-OUT FILES C, D.
14: 17) STOP. (END)
```



John Backus

Speedcoding

BNF

FORTRAN

C FOR COMMENT		CONTINUATION	FORTRAN STATEMENT	IDENTIFICATION		
STATEMENT NUMBER						
1	5	6	7	72	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 13, 12H NUMBERS IS F7.2)			
			STOP 77777			

A sepia-toned portrait of John McCarthy. He is an older man with a full, white beard and mustache, wearing dark-rimmed glasses. He is looking slightly to the right of the camera. He is seated at a desk, and a desk lamp with a white shade is visible to his left. Several papers and a pen are on the desk in front of him. The background is a wooden bookshelf filled with books.

John McCarthy

Lisp

AI, time-sharing



ALGOL

LANGUAGE HISTORY CHART

First letter of each name has been aligned with the approximate date on which work began.

THIS TYPE STYLE indicates languages of major importance, because of their wide usage or technical significance.

THIS TYPE STYLE indicates languages of moderate importance.

THIS TYPE STYLE is used for all other languages.

Parentheses were used to indicate alternate names, or the later addition of the sequence number "1."

- indicates that the second language is a direct extension of the first
- indicates that the second language is an approximate extension of the first, i.e., very similar to the first, but not completely upward compatible
- indicates strong influence; sometimes the second language is "like, or in the style of" the first
- indicates an approximate subset

Each of the following marks is associated with the language above or to its left:

- indicates preliminary or informal specifications or manual
- indicates a public manual, or formal publication via technical paper, or public presentation
- ▲ release for use outside development group

General Comments

This chart represents only the personal opinions of the author as far as value judgments are involved, and the author's best estimate in many cases as far as dates are involved. The indications of the start of the work are the most questionable.

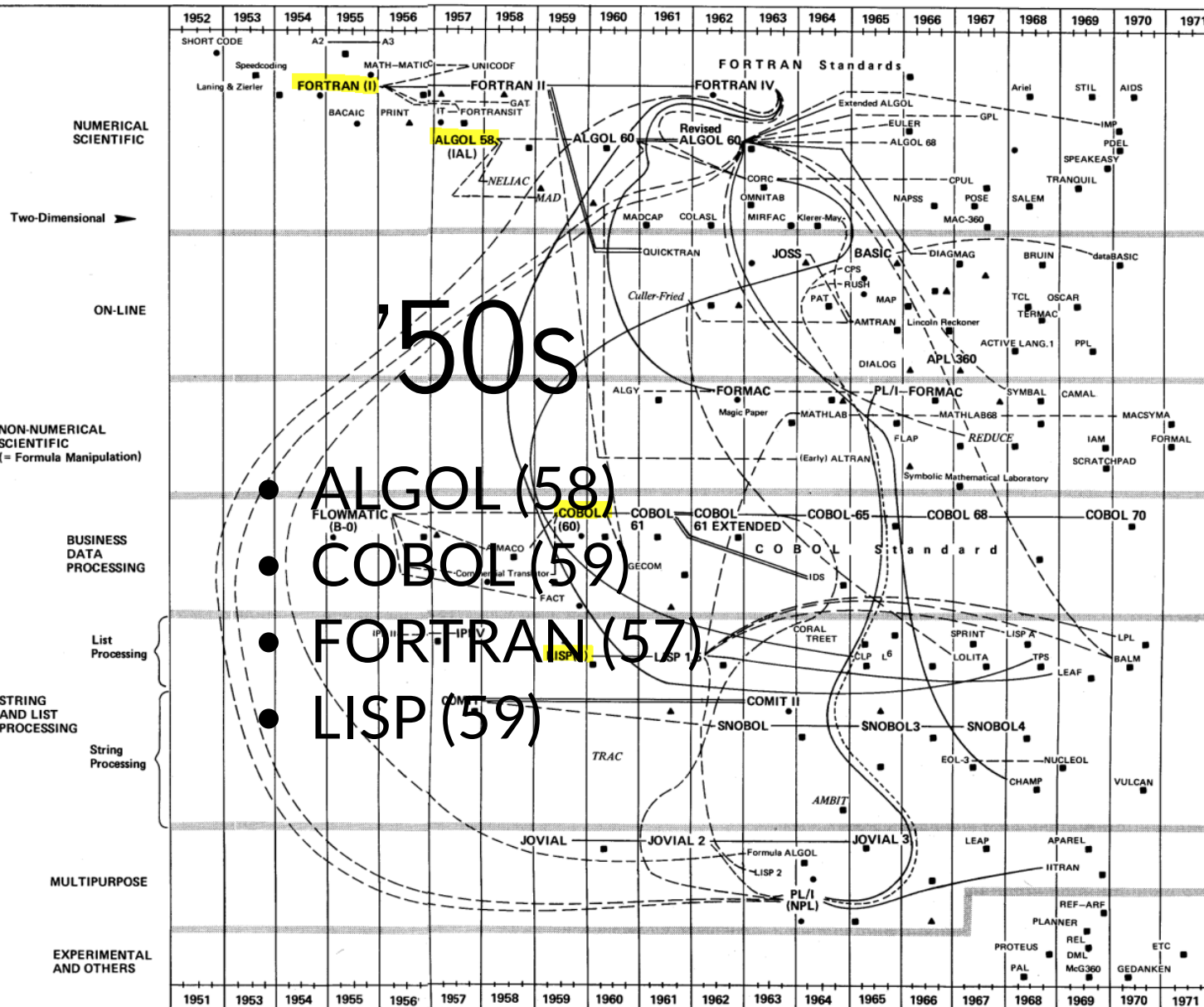
The information for languages in 1971 is based solely on those listed in "Roster of Programming Languages-1971," Computers and Automation, Vol. 20, No. 6B (June 1971), pp. 6-13.

In most cases, dialects with differing names have been omitted. This has the unfortunate effect of appearing to minimize the importance of some languages which spawned numerous versions under differing names (e.g., JOSS).

Languages for specialized application areas (e.g., simulation, machine tool control, civil engineering, systems programming) have not been included because of space considerations. This explains the absence of such obvious languages as APT, GPSS, SIMSCRIPT, COGO, BLISS.

Acknowledgment: The idea for such a chart in such a format came from the one by Christopher J. Shaw entitled "Milestones in Computer Programming" and included with the [ACM Los Angeles Chapter] SIGPLAN notices, February 1965.

"Programming Languages: History and Future"
by Jean E. Sammet
Communications of the ACM, Vol. 15, July 1972
© 1972, Association for Computing Machinery, Inc.



LISP

```
1: (defun is-prime (n)
2:   (cond ((= 2 n) t)
3:         ((= 3 n) t)
4:         ((evenp n) nil)
5:         (t
6:          (loop for i from 3 to (isqrt n) by 2
7:                never (zerop (mod n i))))))
```

'60s

- APL (62)
- BASIC (64)
- LOGO (67)
- Pascal (69)

APL

$$(\sim T \in T \circ . \times T) / T \leftarrow 1 \downarrow \iota R$$

'70s

- Smalltalk (72)
- ML (73)
- Prolog (72)
- C (72)

Prolog

```
1: mother_child(trude, sally).
2:
3: father_child(tom, sally).
4: father_child(tom, erica).
5: father_child(mike, tom).
6:
7: sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).
8:
9: parent_child(X, Y) :- father_child(X, Y).
10: parent_child(X, Y) :- mother_child(X, Y).
```

'80s

- Erlang (86)
- SQL (83)
- Miranda (85)
- C++ (83)
- Perl (87)

Erlang

```
1: -module(mymath).  
2: -export([square/1,fib/1]).  
3:  
4: square(Value) -> Value*Value.  
5:  
6: fib(0) -> 0;  
7: fib(1) -> 1;  
8: fib(N) when N>1 -> fib(N-1) + fib(N-2).
```

'90s

- Haskell (90)
- Ruby (95)
- Python(91)
- Delphi (95)
- Java (95)
- Visual Basic (91)
- Javascript (95)

Javascript

```
1: function factorial(n) {  
2:     if (n == 0) {  
3:         return 1;  
4:     }  
5:     return n * factorial(n - 1);  
6: }
```

'00s

- C# (00)
- Scala (03)
- F# (05)
- Clojure (07)
- D (01)
- Go(07)

D

```
1: void Quack(Animal)(Animal a)
2:     if( __traits(compiles, a.Quack()))
3: {
4:     a.Quack();
5: }
6:
7: struct Duck { void Quack(){ "Quack".writeln; }}
8:
9: int main(string[] argv) {
10:     Duck d;
11:     Quack(d); // good
12:     Quack(5); // compile time error
13: }
```

'10s

- Elixir (12)
- Elm (12)
- Rust (10)
- Pony (14)
- Idris (12)

Idris

```
1: data Vect : Nat -> Type -> Type where
2:   Nil      : Vect 0 a
3:   (:::)    : (x : a) -> (xs : Vect n a) -> Vect (n + 1) a
4:
5: total
6: append : Vect n a -> Vect m a -> Vect (n + m) a
7: append Nil      ys = ys
8: append (x ::: xs) ys = x ::: append xs ys
```

"More is not better (or worse) than less, just different."

The principal programming paradigms

"More is not better (or worse) than less, just different."

v1.08 © 2008 by Peter Van Roy

Legend:

- Observable:
- nondeterminism?:

Paradigms and Languages:

- Descriptive declarative programming** (XML, S-expression)
- First-order functional programming** (+ procedure)
- Functional programming** (+ closure)
 - Deterministic logic programming** (+ unification (equality))
 - Relational & logic programming** (+ search)
 - Prolog, SQL embeddings** (+ solver)
 - Constraint (logic) programming**
 - CLP, ILOG Solver** (+ thread)
 - Concurrent constraint programming**
 - LIFE, AKL** (+ by-need synchronization)
 - Lazy concurrent constraint programming** (Oz, Alice, Curry)
 - Lazy functional programming** (+ by-need synchroniz.)
 - Lazy concurrent concurrent programming** (Oz, Alice, Curry)
 - Monotonic dataflow programming** (+ thread, + single assign.)
 - Declarative concurrent programming** (+ by-need synchronization)
 - Lazy dataflow programming**
 - Lazy declarative concurrent programming** (Oz, Alice, Curry)
 - Continuation programming** (+ thread, + single assign.)
 - Nonmonotonic dataflow programming** (+ nondeterministic choice)
 - Concurrent logic programming**
 - Oz, Alice, Curry, Excel, AKL, FGHC, FCP** (+ synch. on partial termination)
 - Functional reactive programming (FRP)**
 - Weak synchronous programming**
 - FrTime, SL** (+ instantaneous computation)
 - Strong synchronous programming** (Esterel, Lustre, Signal)
 - ADT functional programming** (+ name (unforgeable constant), + cell)
 - ADT imperative programming** (+ port (channel))
 - Multi-agent dataflow programming** (Oz, Alice, AKL)
 - Imperative programming** (+ cell (state))
 - Imperative search programming** (+ search)
 - SNOBOL, Icon, Prolog**
 - Event-loop programming** (+ port (channel))
 - E in one vat** (+ thread)
 - Multi-agent programming**
 - Message-passing concurrent programming**
 - Erlang, AKL** (+ local cell)
 - Active object programming**
 - Object-capability programming**
 - CSP, Occam, E, Oz, Alice, publish/subscribe, tuple space (Linda)**
 - Sequential object-oriented programming** (+ closure)
 - Stateful functional programming**
 - Java, OCaml** (+ thread)
 - Concurrent object-oriented programming**
 - Shared-state concurrent programming**
 - Smalltalk, Oz, Java, Alice** (+ log)
 - Software transactional memory (STM)**
 - SQL embeddings**

Horizontal Axis:

 - More declarative** ← **Unnamed state (seq. or conc.)** | **Nondet. state** | **Named state** → **Less declarative**

Vertical Axis:

 - Data structures only** (Turing equivalent) | **More is not better (or worse) than less, just different.**

See "Concepts, Techniques, and Models of Computer Programming".

When a language is mentioned under a paradigm, it means that part of the language is intended (by its designers) to support the paradigm without interference from other paradigms. It does not mean that there is a perfect fit between the language and the paradigm. It is not enough that libraries have been written in the language to support the paradigm. The language's kernel language should support the paradigm. When there is a family of related languages, usually only one member of the family is mentioned to avoid clutter. The absence of a language does not imply any kind of value judgment.

Axes orthogonal to this chart are typing, aspects, and domain-specificity. Typing is not completely orthogonal: it has some effect on expressiveness. Aspects should be completely orthogonal, since they are part of a program's specification. A domain-specific language should be definable in any paradigm (except when the domain needs a particular concept).

Metaprogramming is another way to increase the expressiveness of a language. The term covers many different approaches, from higher-order programming, syntactic extensibility (e.g., macros), to higher-order programming combined with syntactic support (e.g., meta-object protocols and generics), to full-fledged tinkering with the kernel language (introspection and reflection). Syntactic extensibility and kernel language tinkering in particular are orthogonal to this chart. Some languages, such as Scheme, are flexible enough to implement many paradigms in almost native fashion. This flexibility is not shown in the chart.



A silhouette of a cowboy on a horse herding three cats at sunset. The cowboy is on the left, wearing a hat and holding a lasso. The horse is in the center, facing right. Three cats are on the right, walking towards the horse. The background is a sunset sky with a bright orange glow on the horizon.

Herding the cats

Extra Attention to tooling & developer productivity.

*Tension between "good languages" and
"getting stuff done".*

*People (and communities) developing
languages... how to get funding???*

We must know. We Will know



Thanks

¿O RLMENTE?

@SilverSpoon - roundcrisis.com

A non exhaustive list of the Resources

- Programming languages: History and future (1972 Jean E. Sammet)
- Definition of Turing Machines - Stanford Encyclopedia of Philosophy
- This has happened before and will happen again - Strange Loop conference recording- Video
- David Hilbert
- Alan Kay: Computer Applications: A Dynamic Medium for Creative Thought 1972
- The APL Programming Language Source Code
- Roots of computer languages through the ages
- A Science of Operations, M. Priestley- Book

- Some History of Functional Programming Languages - D. A. Turner
- History of ML - David McQueen
- Visualizing influence relations of programming languages
- Freebase programming language collection
- Turing on computable numbers
- A Programming Language
- Principal programming paradigms

Photo credits

- [history main starting the talk](#)
- "Alonzo Church" by Princeton University. Licensed under Fair use via [Wikipedia](#)