UNIVERSITÄT BERN

Einführung in die Informatik

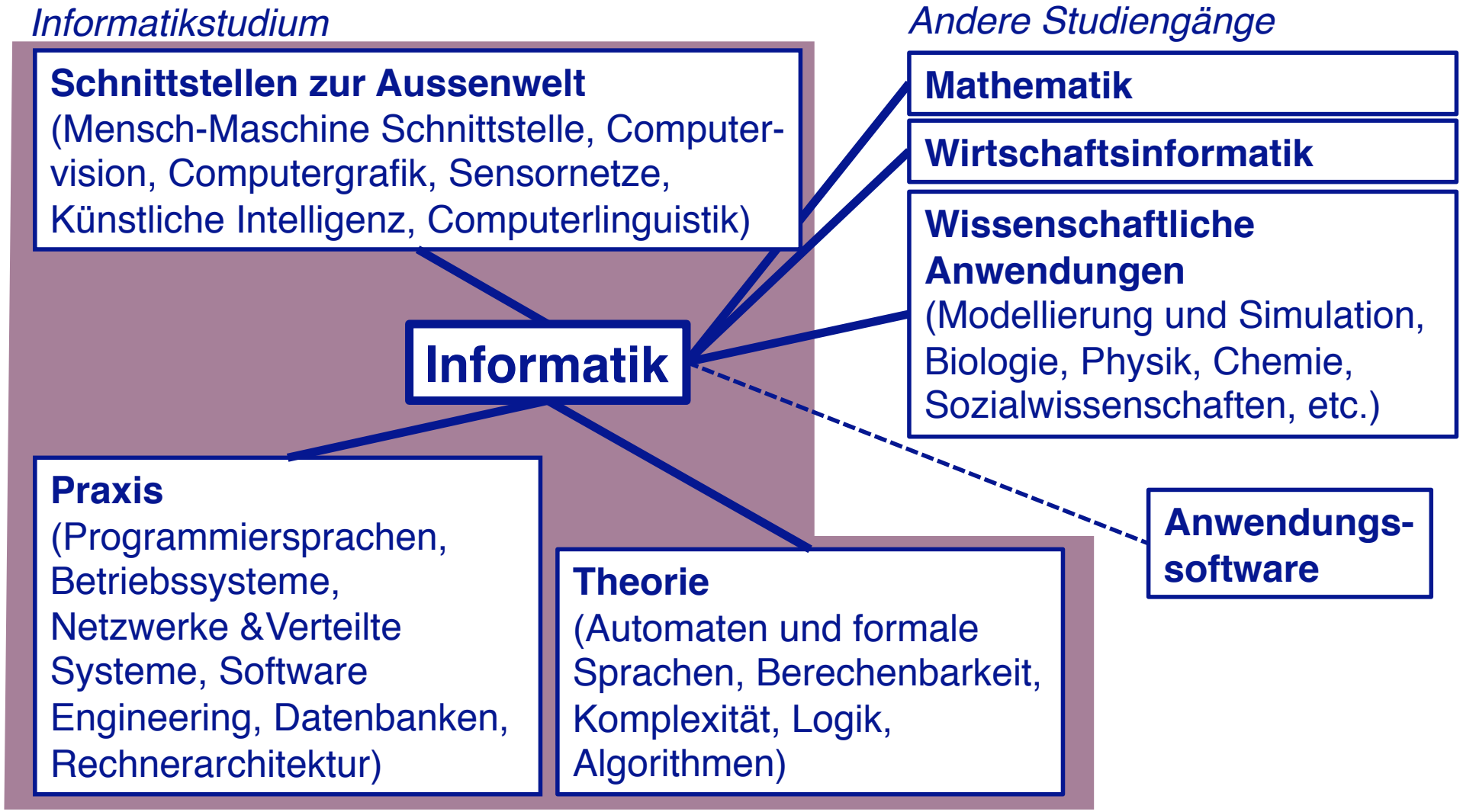
2. Programming Languages, Paradigms and Technology

Institut für Informatik und angewandte Mathematik

Roadmap

- What is a programming language?
- Programming = modeling
- Evolution
- Trends and Challenges

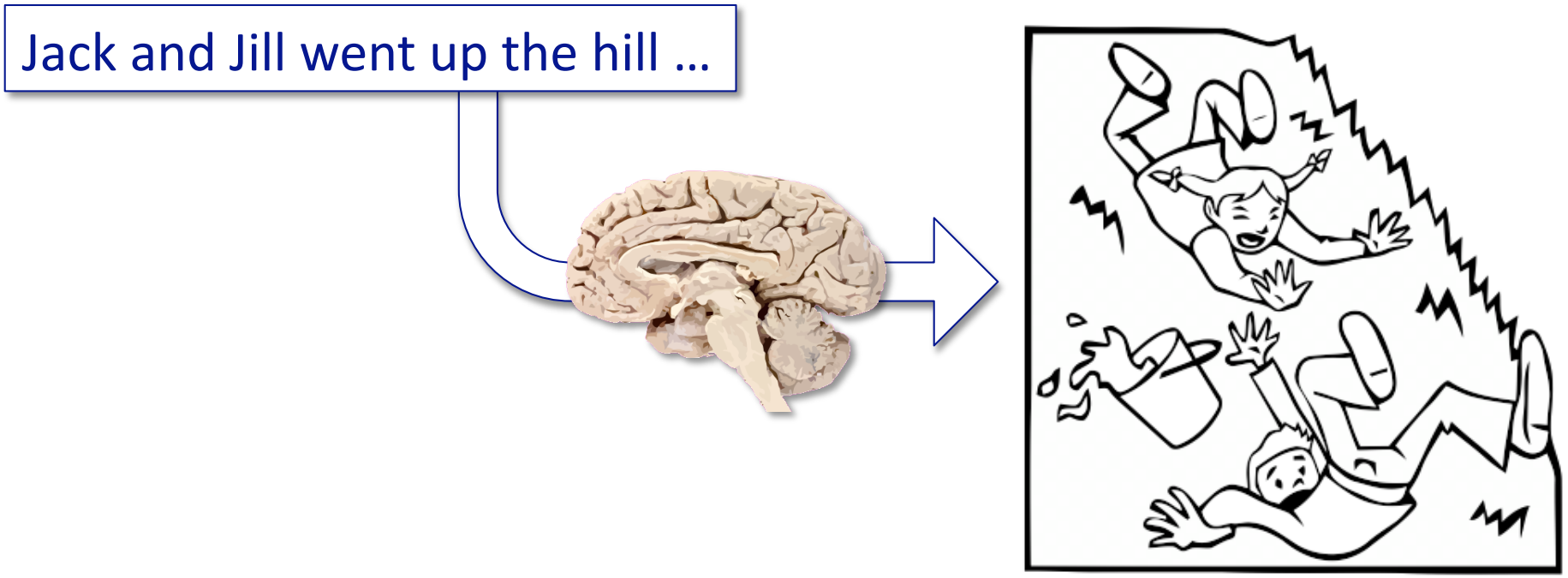
Übersicht



Roadmap

- **What is a programming language?**
- Programming = modeling
- Evolution
- Trends and Challenges

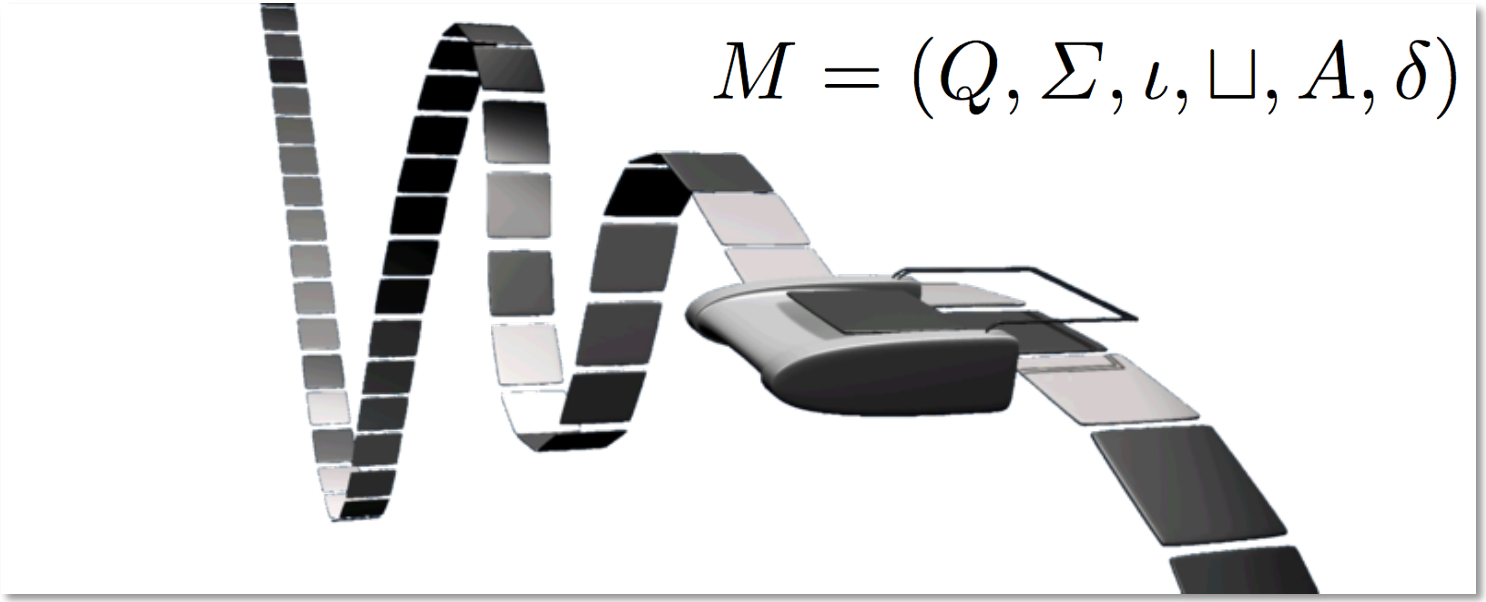
What is language?



Language = *Sequences* of symbols (or sounds) which we *interpret* to attribute *meaning*

What is a formal language?

A *Turing machine* reads (and writes) a tape of 0s and 1s



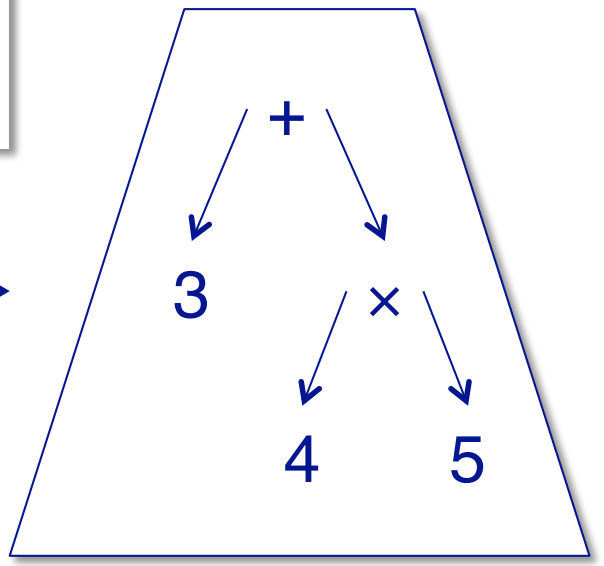
The *language* it accepts is the set of strings that leave it in an *accepting state*

How can we describe formal languages?

Use a set of *rules* ($\alpha \rightarrow \beta$) to describe the *structure* of the language

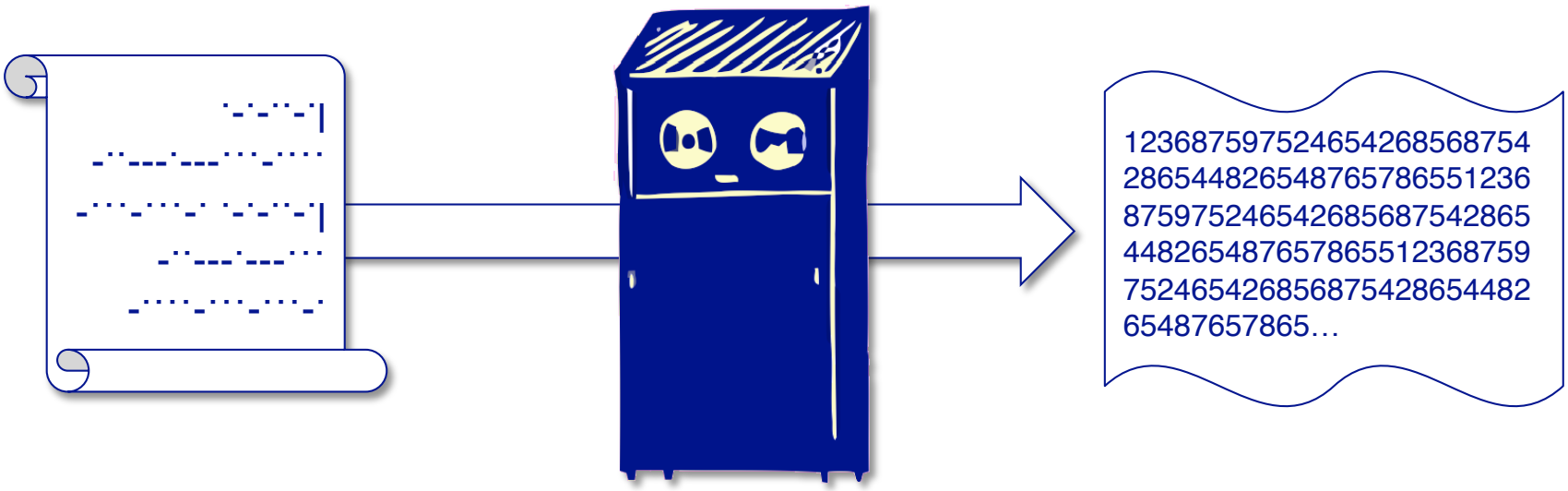
expression → number
expression → expression + expression
expression → expression × expression
number → digit
number → digit number

3 + 4 × 5 →



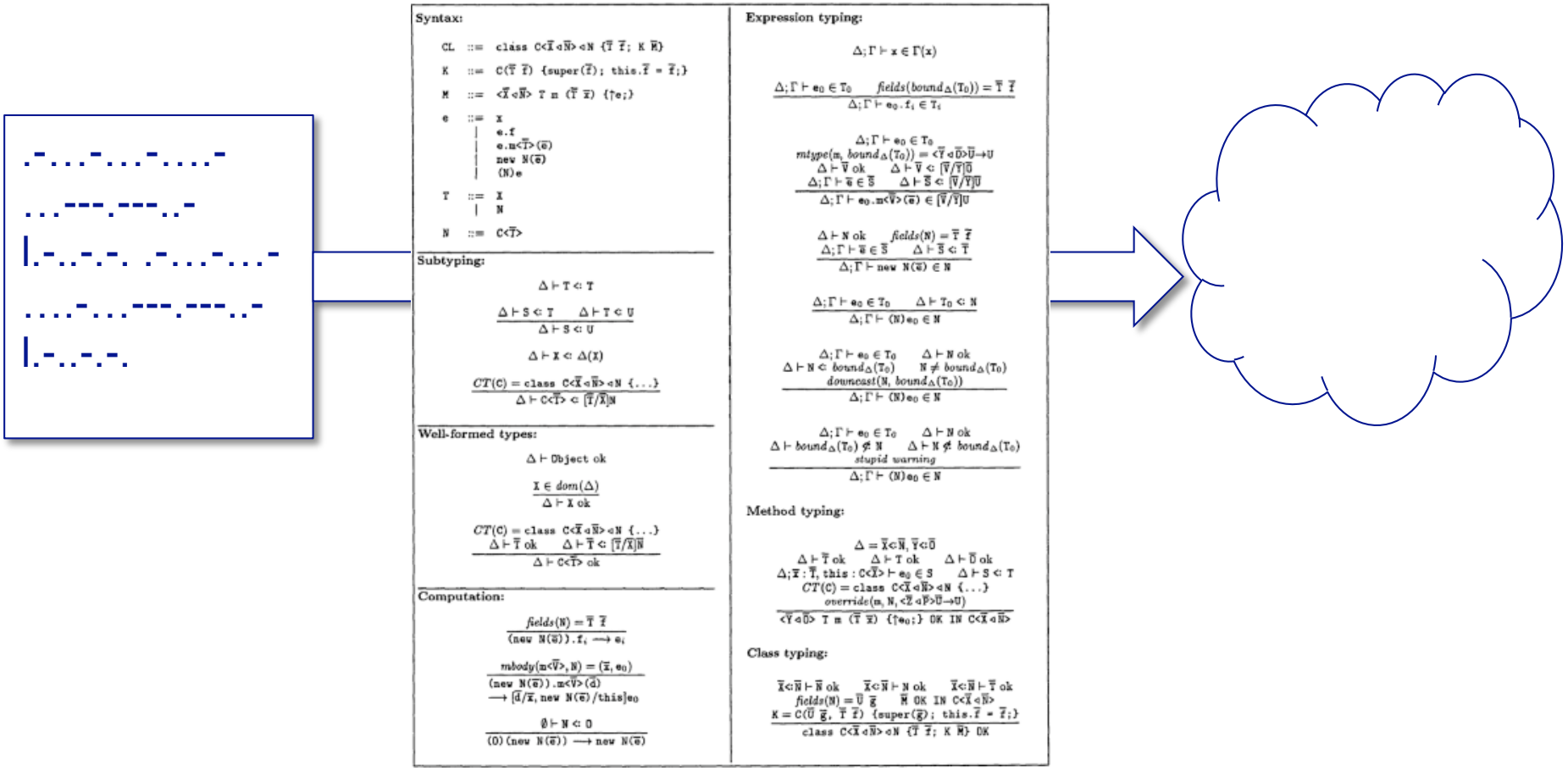
cf. Chomsky

What is a Programming Language? (take 1)



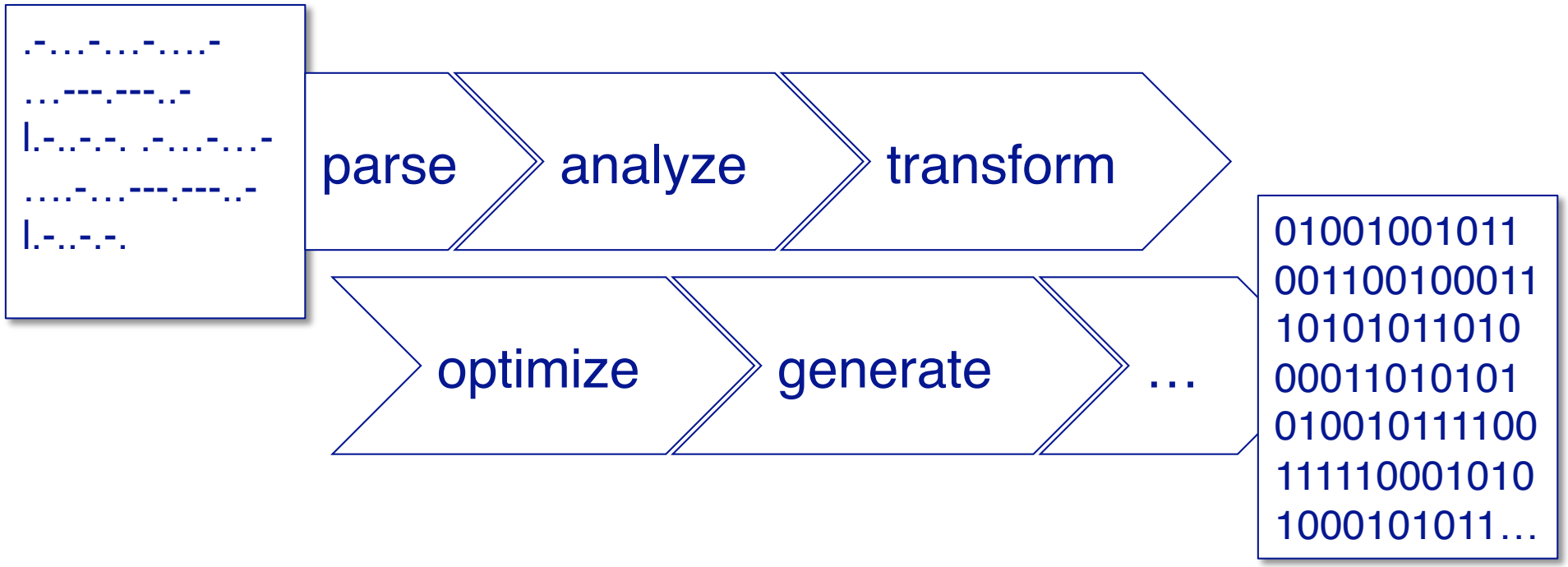
A language to instruct a computer to compute “stuff” …

What is a Programming Language? (take 2)



Syntax and semantics in a mathematical domain …

What is a Programming Language? (take 3)



What the compiler will translate …

What is a Programming Language? (take 4)

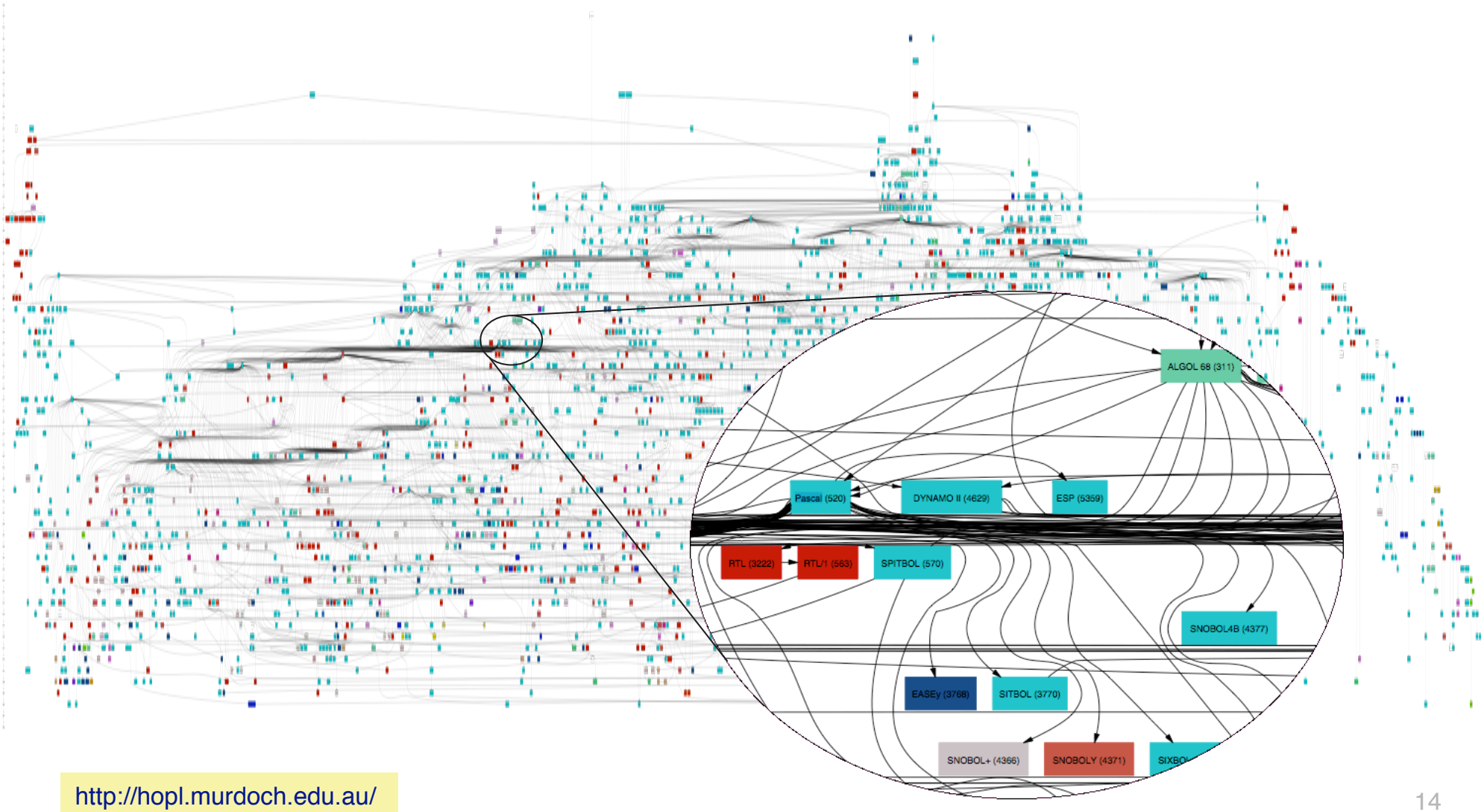
A language for *communicating* software designs

Roadmap

- What is a programming language?
- **Programming = modeling**
- Evolution
- Trends and Challenges

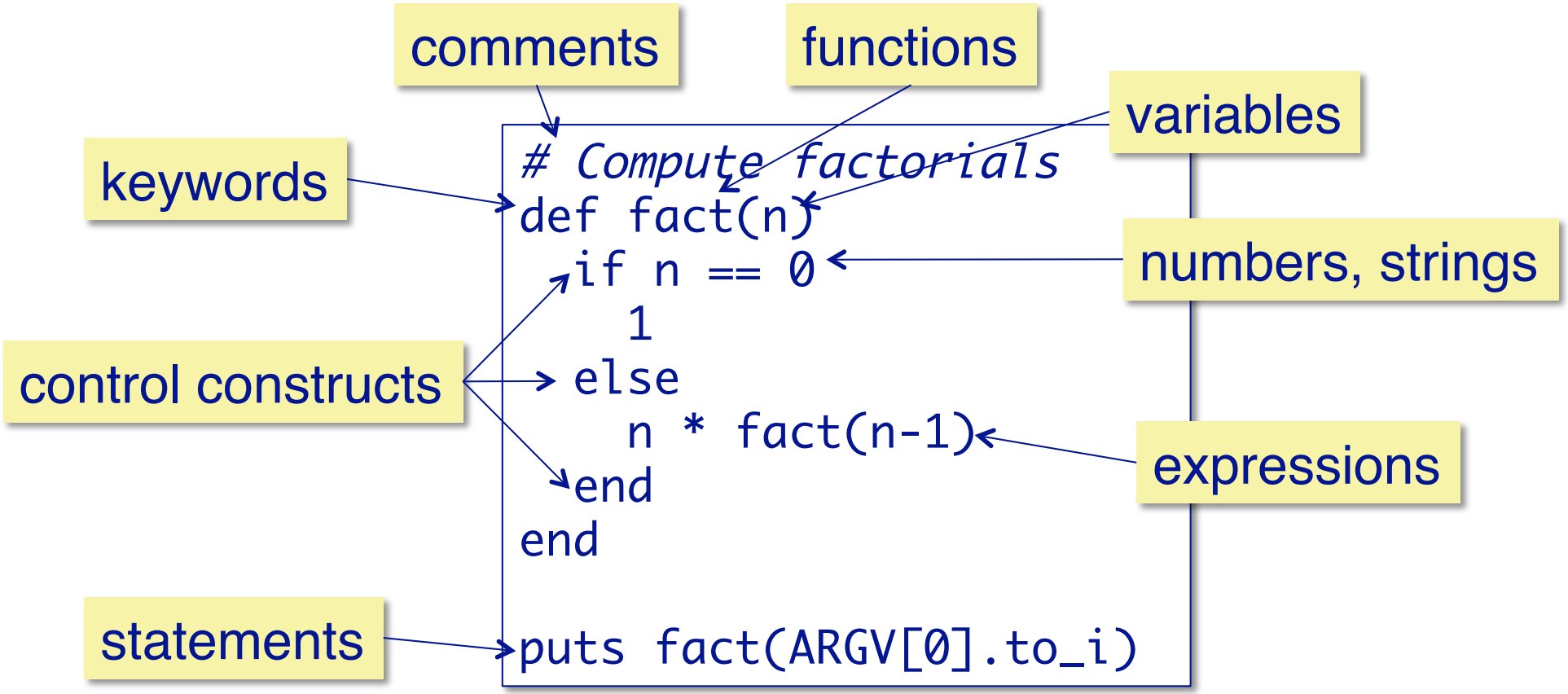
Programming = Modeling

Over 8000 recorded programming languages



http://hopl.murdoch.edu.au/

What do programming languages have in common?

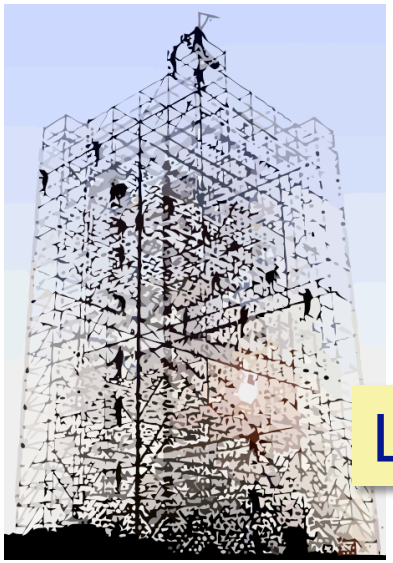


A fragment of Ruby code

How do these languages differ?

Imperative

Functional

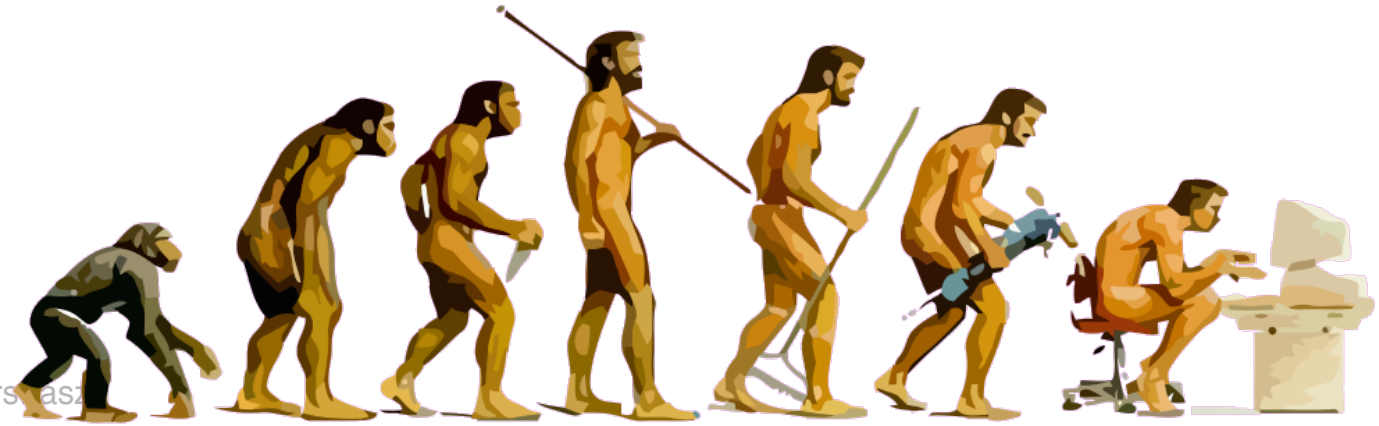


Logic

Object-oriented

Roadmap

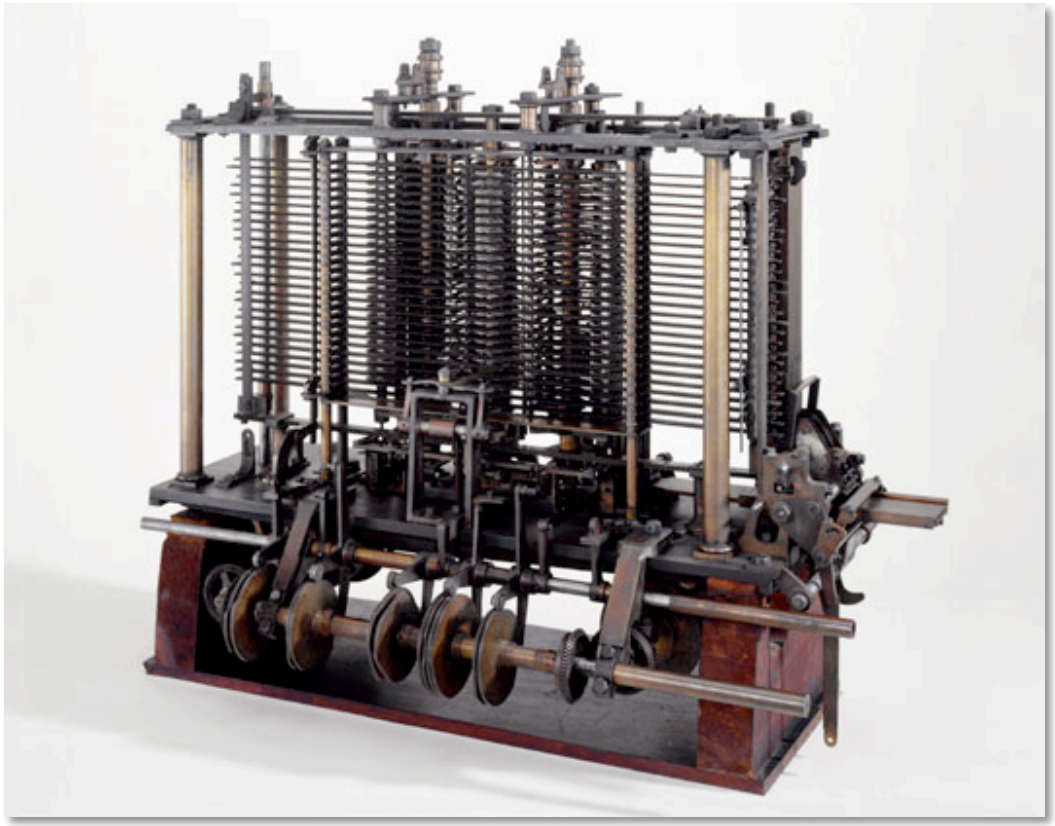
- What is a programming language?
- Programming = modeling
- **Evolution**
- Trends and Challenges



Jacquard loom — 1801

Punch cards are invented

Babbage's Analytical Engine — 1822



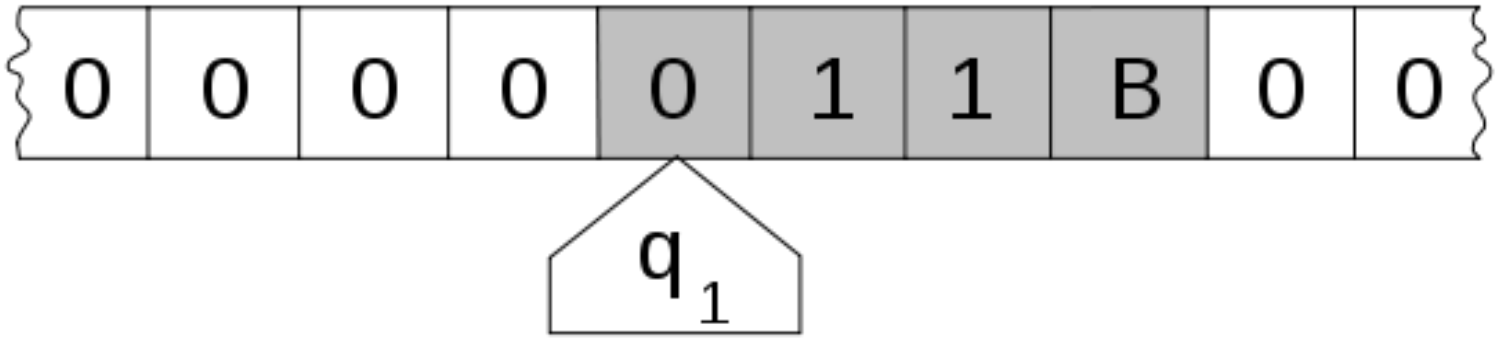
The first mechanical computer

Church's Lambda Calculus — 1932

$$(\lambda\, x.\ (\lambda\, y.\ x))\ a\ b \rightarrow a$$

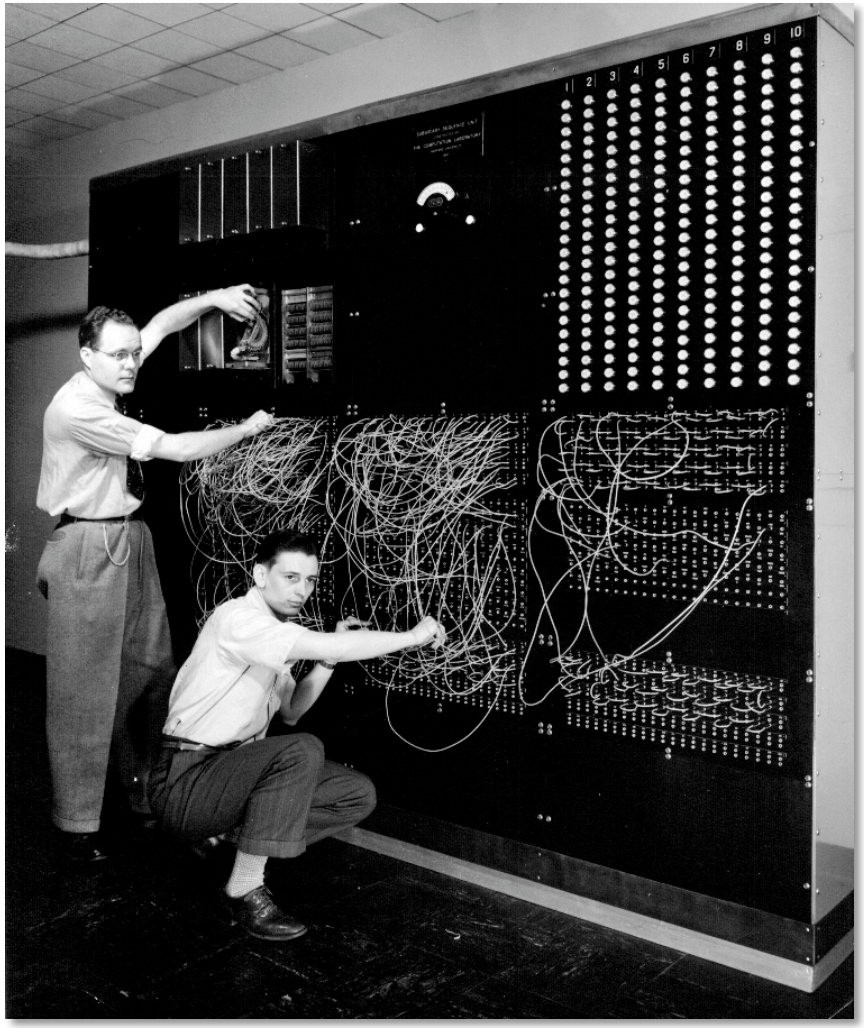
The first (minimal) language for studying computation

Turing machine — 1936



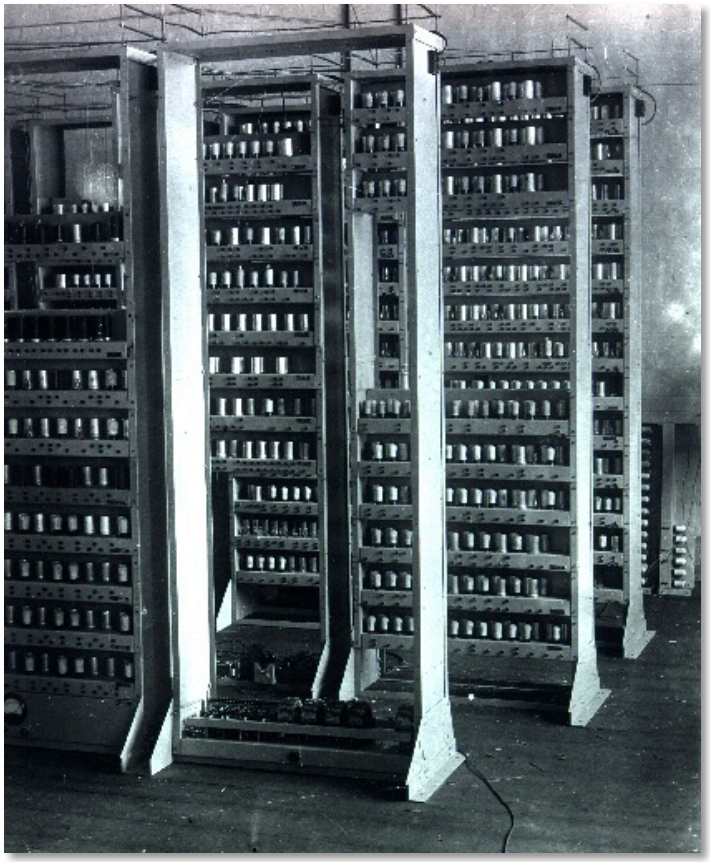
The first abstract model of a computer

1st generation: Machine code — 1944

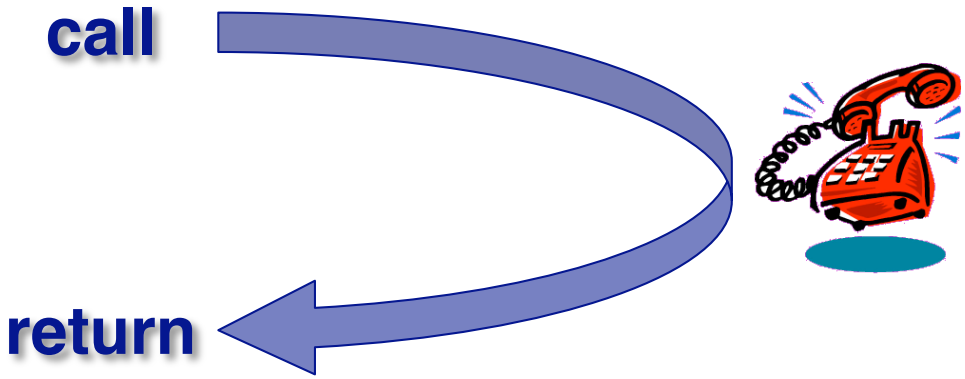


Machine code is only meant to be read by … machines

Subroutines — 1949



The subroutine is one of the key concepts of programming

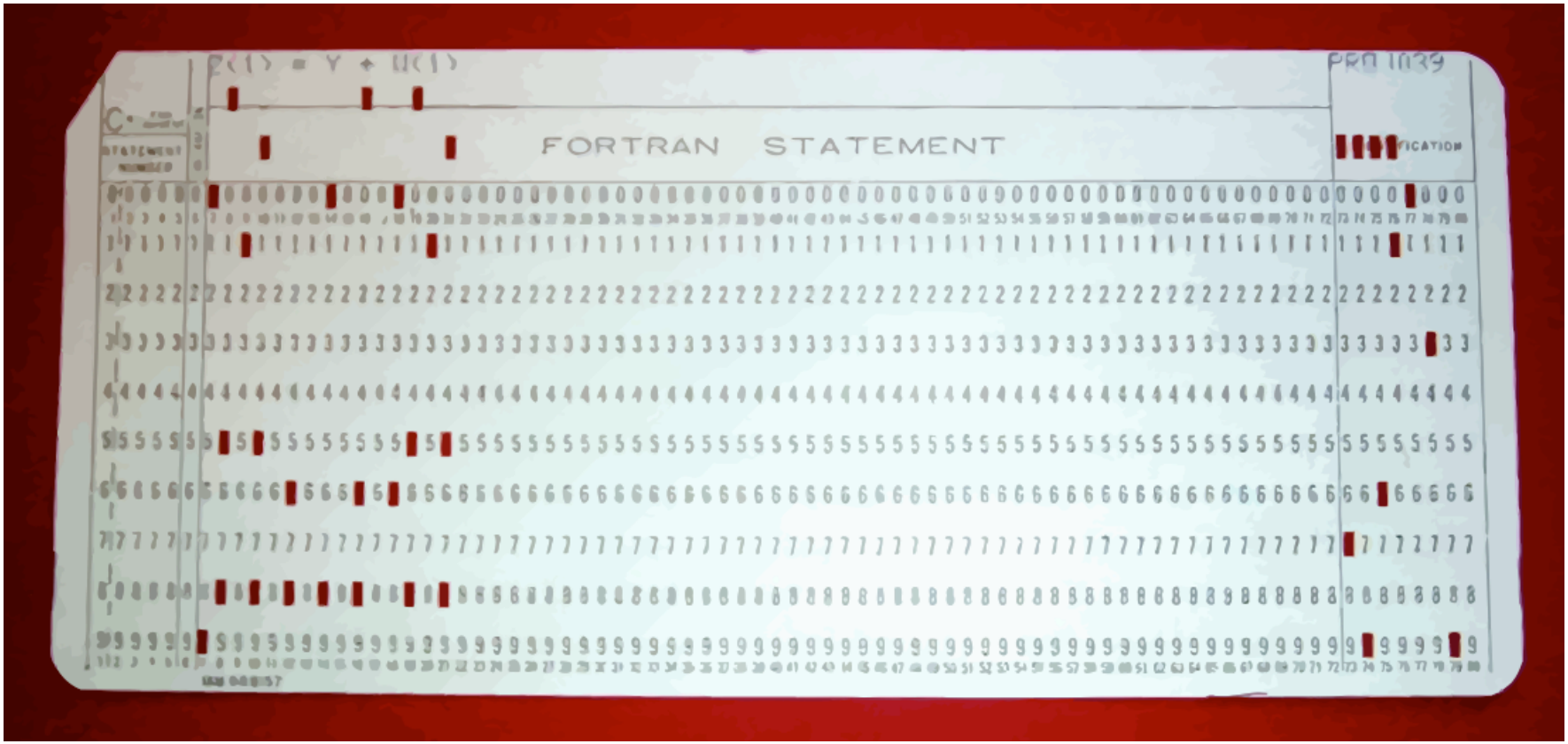


2nd generation: assembler — early 1950s

Address	Label	Instruction (AT&T syntax)	Object code[16]
		.begin	
		.org 2048	
	a_start	.equ 3000	
2048		ld length,%	
2064		be done	00000010 10000000 00000000 00000110
2068		addcc %r1,-4,%r1	10000010 10000000 01111111 11111100
2072		addcc %r1,%r2,%r4	10001000 10000000 01000000 00000010
2076		ld %r4,%r5	11001010 00000001 00000000 00000000
2080		ba loop	00010000 10111111 11111111 11111011
2084		addcc %r3,%r5,%r3	10000110 10000000 11000000 00000101
2088	done:	jmpl %r15+4,%r0	10000001 11000011 11100000 00000100
2092	length:	20	00000000 00000000 00000000 00010100
2096	address:	a_start	00000000 00000000 00001011 10111000
		.org a_start	
3000	a:		

Assembly code introduces symbolic names (for humans!)

3rd generation: FORTRAN — 1955



High-level languages are born

ALGOL — 1958

```
<statement> ::= <unconditional statement>
    | <conditional statement>
    | <for statement>
...
```

BNF

```
begin
    ...
end
```

block structure

recursion

Lisp — 1958

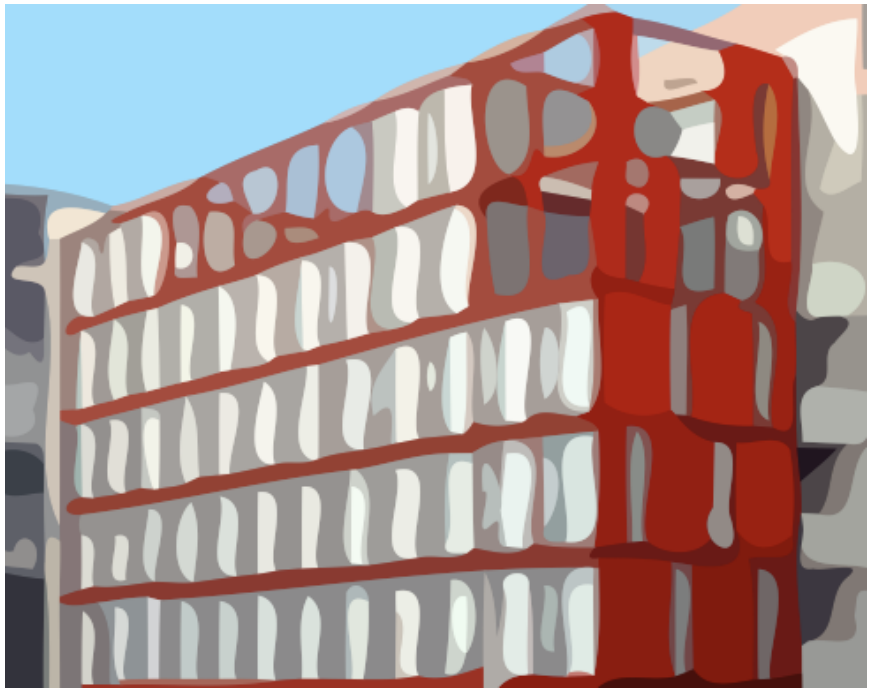
```
(defun factorial (n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

programs as data

garbage collection

COBOL — 1959

```
ADD YEARS TO AGE.
MULTIPLY PRICE BY
QUANTITY GIVING COST.
SUBTRACT DISCOUNT FROM
COST GIVING FINAL-COST.
```



modules

BASIC — 1964

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

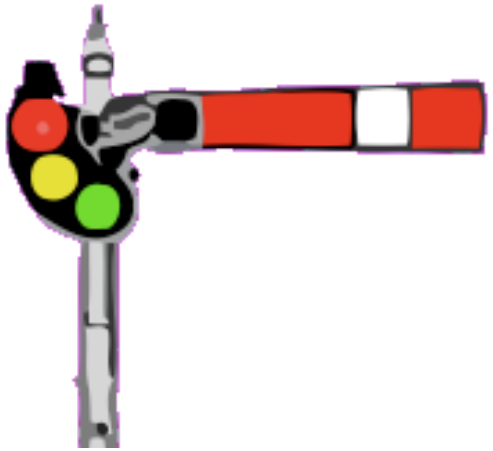
interactive programming for the masses

JCL — 1964

```
//IS198CPY JOB (IS198T30500),'COPY JOB',CLASS=L,MSGCLASS=X
//COPY01   EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=OLDFILE,DISP=SHR
//SYSUT2   DD DSN=NEWFILE,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(40,5),RLSE),
//            DCB=(LRECL=115,BLKSIZE=1150)
//SYSIN    DD DUMMY
```

invented *scripting* for IBM 360

Semaphores — 1965



```
P – acquire resource
… critical section
V – release resource
```

radically simplified concurrency control

Planner — 1969
Prolog — 1972

```
man(socrates).
mortal(X) :- man(X).
```

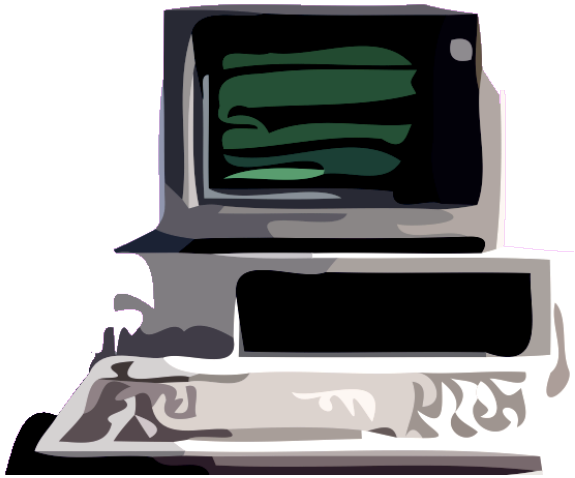
facts and rules

```
?- mortal(socrates).
Yes
?- mortal(elvis).
No
```

queries and inferences

Pascal — 1970

designed for teaching



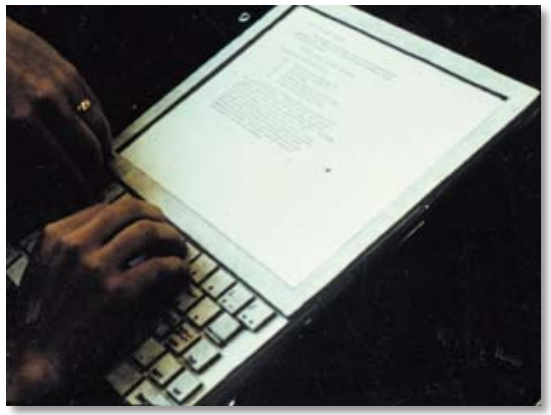
successful with PCs

C — 1972


```
#include <stdio.h>
//echo the command line arguments
int main (int argc, char* argv[]) {
   int i;
   for (i=1; i<argc; i++) {
      printf("%s ", argv[i]);
   }
   printf("\n");
   return 0;
}
```

portable systems programming

Smalltalk — 1972



"Dynabook" vision

Everything is an object

Everything happens by sending messages

```
5 factorial → 120
```

```
Integer»factorial
   self = 0 ifTrue: [^ 1].
   self > 0 ifTrue: [^ self * (self - 1) factorial].
   self error: 'Not valid for negative integers'
```

ML polymorphic type inference — 1973

"Now! *That* should clear up a few things around here!"

```
length [ ] = 0
length (x:xs) = 1 + length xs

length :: [a] -> Int

length "hello"  → 5
length [10..20] → 11
```

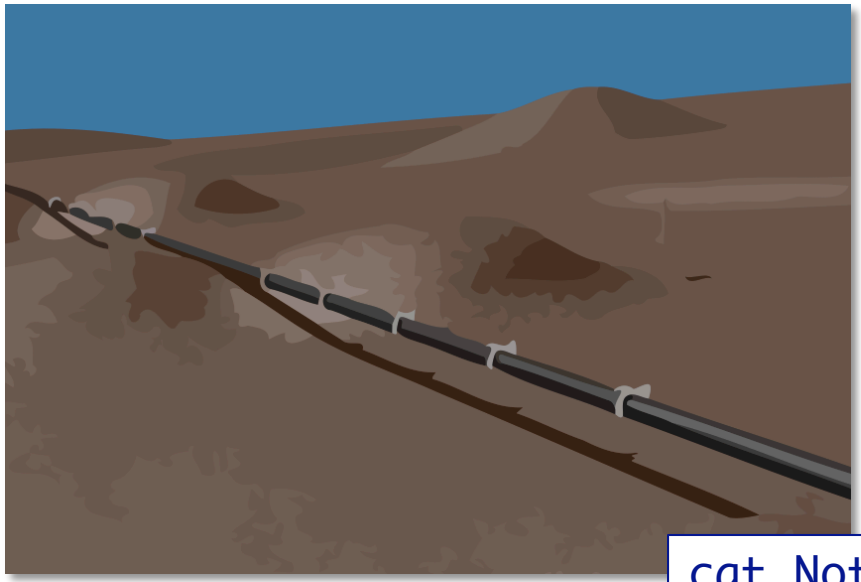
generic functions may be applied to *many types* of arguments

Monitors — 1974

```
public class Account {
    protected int assets = 0;
    ...
    public synchronized void withdraw(int amount) {
        while (amount > assets) {
            try {
                wait();
            } catch(InterruptedException e) { }
        }
        assets -= amount;
    }
    ...
}
```

structured concurrency control

Bourne shell — 1977



scripting pipelines of commands

```
cat Notes.txt | tr -c '[:alpha:]' '\012'
| sed '/^$/d' | sort | uniq -c | sort -rn
| head -5

→    14 programming
     14 languages
      9 of
      7 for
      5 the
```

SQL— 1978

```
SELECT *
    FROM Book
    WHERE price > 100.00
    ORDER BY title;
```

domain-specific language for relational databases

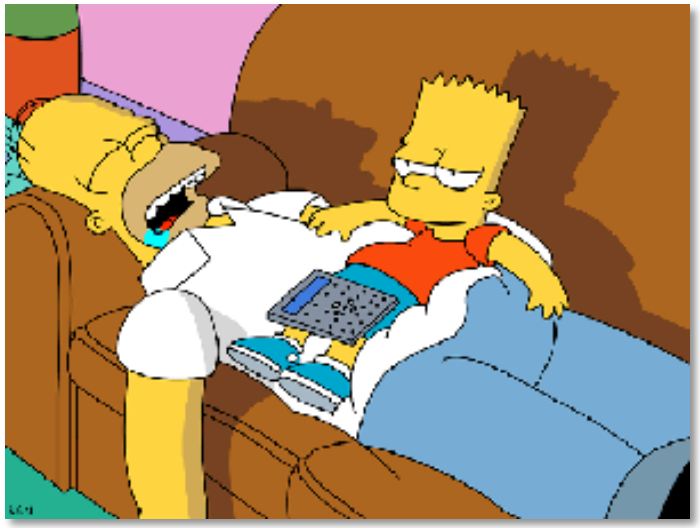
Miranda — 1985

"pure" functional programming

```
fibs = 1 : 1 : fibsAfter 1 1
fibsAfter a b = (a+b) : fibsAfter b (a+b)
```

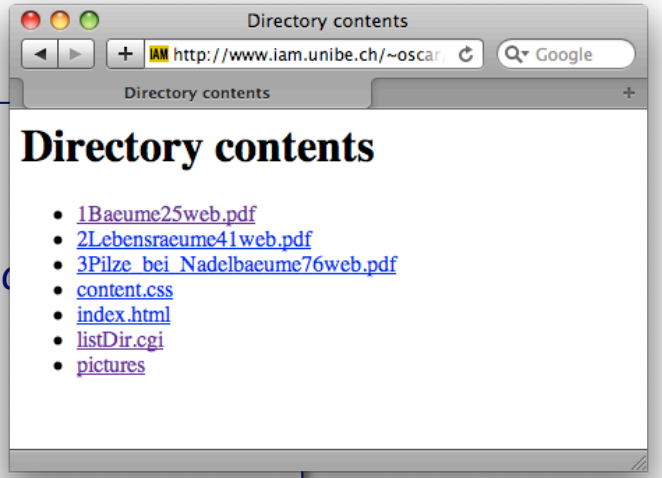
```
take 10 fibs
→ [1,1,2,3,5,8,13,21,34,55]
```

lazy evaluation



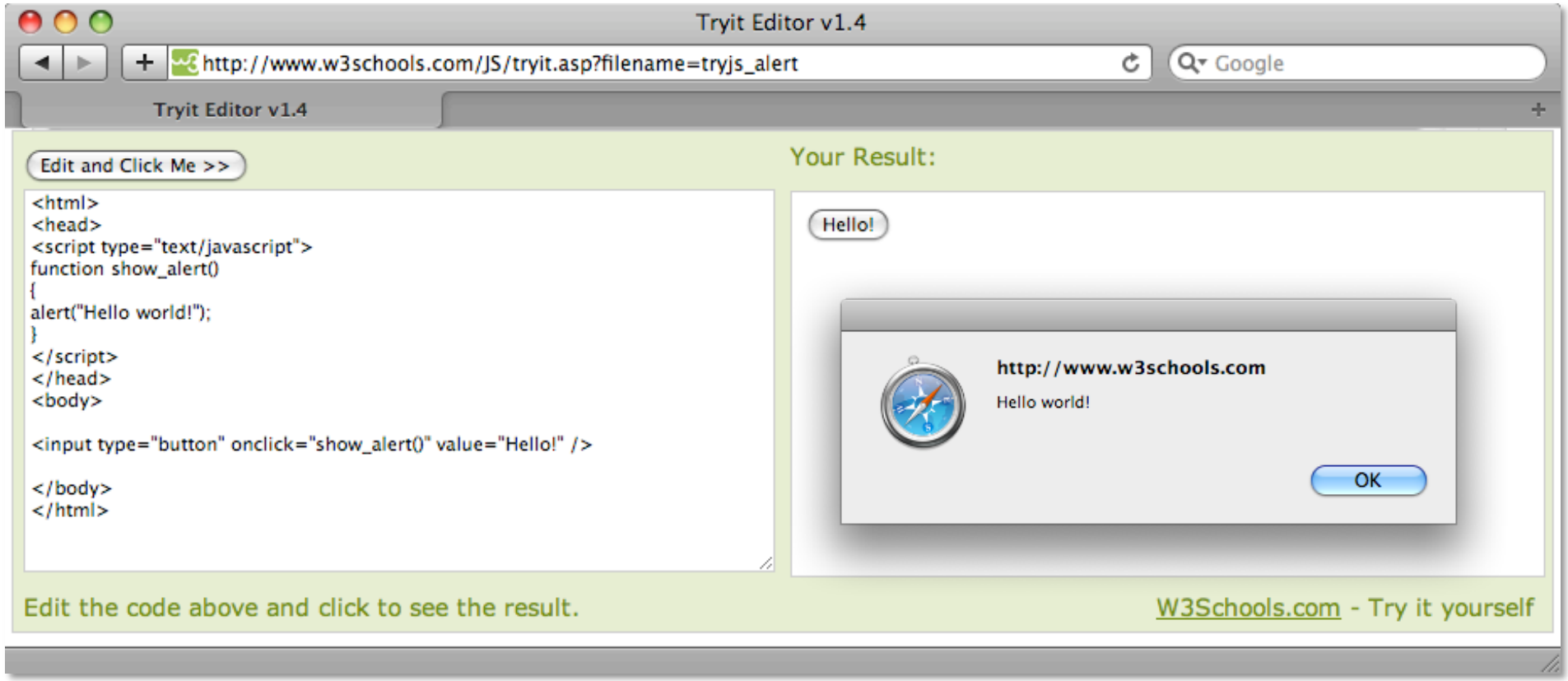
Perl — 1987

```
#!/usr/bin/perl -w
print "Content-type: text/html\n\n";
print <<'eof'
<html><head><title>Directory contents</title></hea
<body>
<h1>Directory contents</h1><ul>
eof
;
@files = <*>;
foreach $file (@files) {
    print '<li><a href="' . $file . '">' . $file . "</li>\n";
}
print "</ul></body></html>\n";
__END__
```



text manipulating, then web scripting

JavaScript — 1995



client-side browser scripting

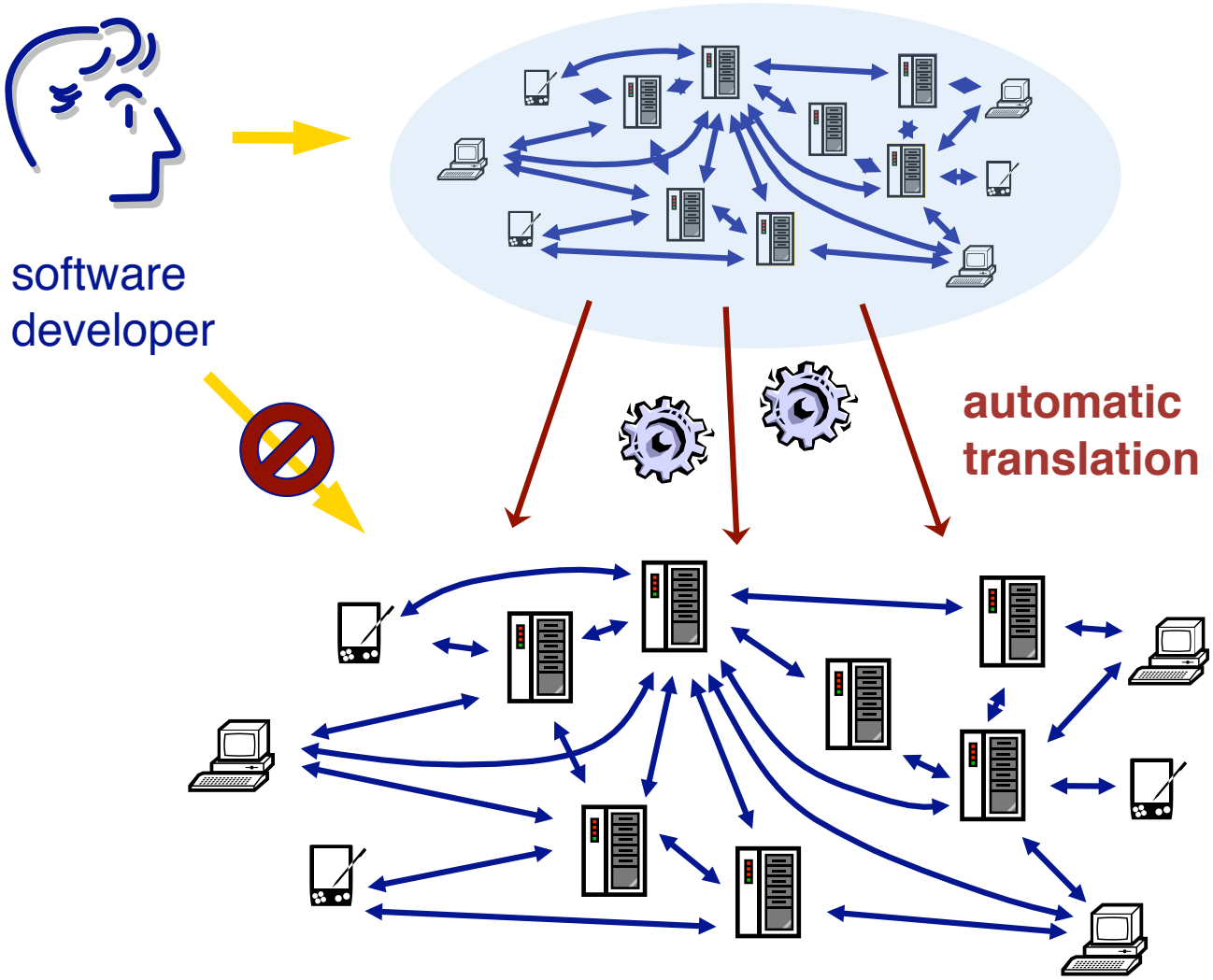
Roadmap

- What is a programming language?
- Programming = modeling
- Evolution
- **Trends and Challenges**

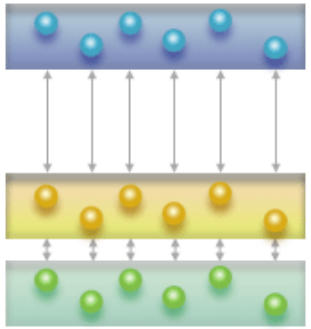
Components, Frameworks, Patterns — 1990s

So far, limited impact on programming languages …

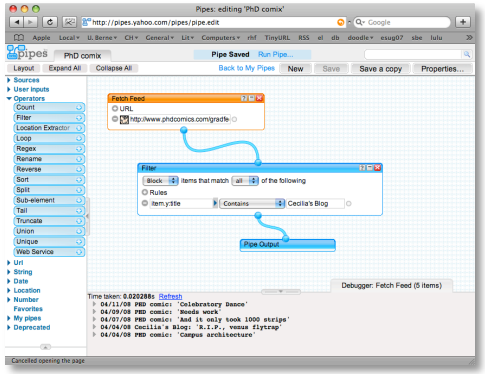
Model-Driven Development — 1980s, 1990s



Trends and Challenges

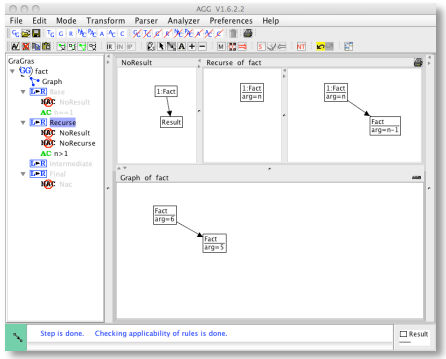


naked objects

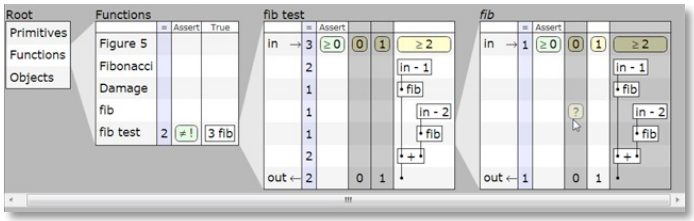


yahoo pipes

Bridging gap between users and technology



graph transformation



subtext

Conclusions

Programming languages are for *humans* not just computers

Programming is *modeling*

Programming languages have always evolved to bring programming *closer to the users' problems*

We are still *very early* in the history of programming