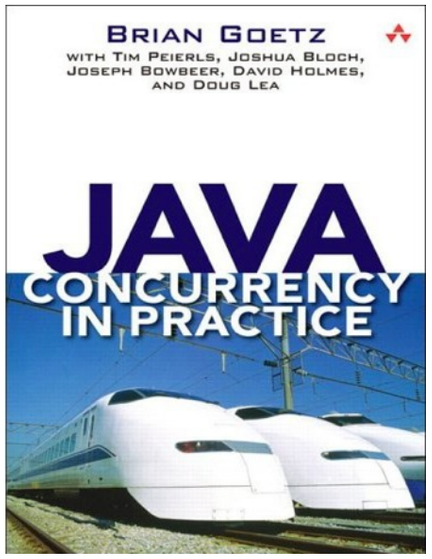
CMSC 330
Summer 2015

Kristopher Micinski



CMSC 330 Summer 2015

Kristopher Micinski

$$\frac{\Gamma \vdash e \Downarrow \texttt{true} \qquad \Gamma \vdash e_1 \Downarrow v}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v}$$

$$\frac{\Gamma \vdash e \Downarrow \texttt{false} \qquad \Gamma \vdash e_2 \Downarrow v}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Downarrow v}$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}Var$$

$$\frac{\Gamma, x : \tau_p \vdash t : \tau_r}{\Gamma \vdash (\lambda x : \tau_p . t) : \tau_p \to \tau_r}Lam$$

$$\frac{\Gamma \vdash t_f : \tau_p \to \tau_r \qquad \Gamma \vdash t_p : \tau_p}{\Gamma \vdash t_f\ t_p : \tau_r}App$$

What people are saying…

“For CMSC330 I loved learning the languages…

“You also learn Ruby, but it's essentially Java except instead of having a main function the entire file you run (a script) is your main function and you declare other functions inside of that.”

What people are saying…

“For CMSC330 I loved learning the languages…

(looking back at it… **durring** OCaml projects was a whole different story).”

“You also learn Ruby, but it's essentially Java except instead of having a main function the entire file you run (a script) is your main function and you declare other functions inside of that.”

Mechanics & Tools

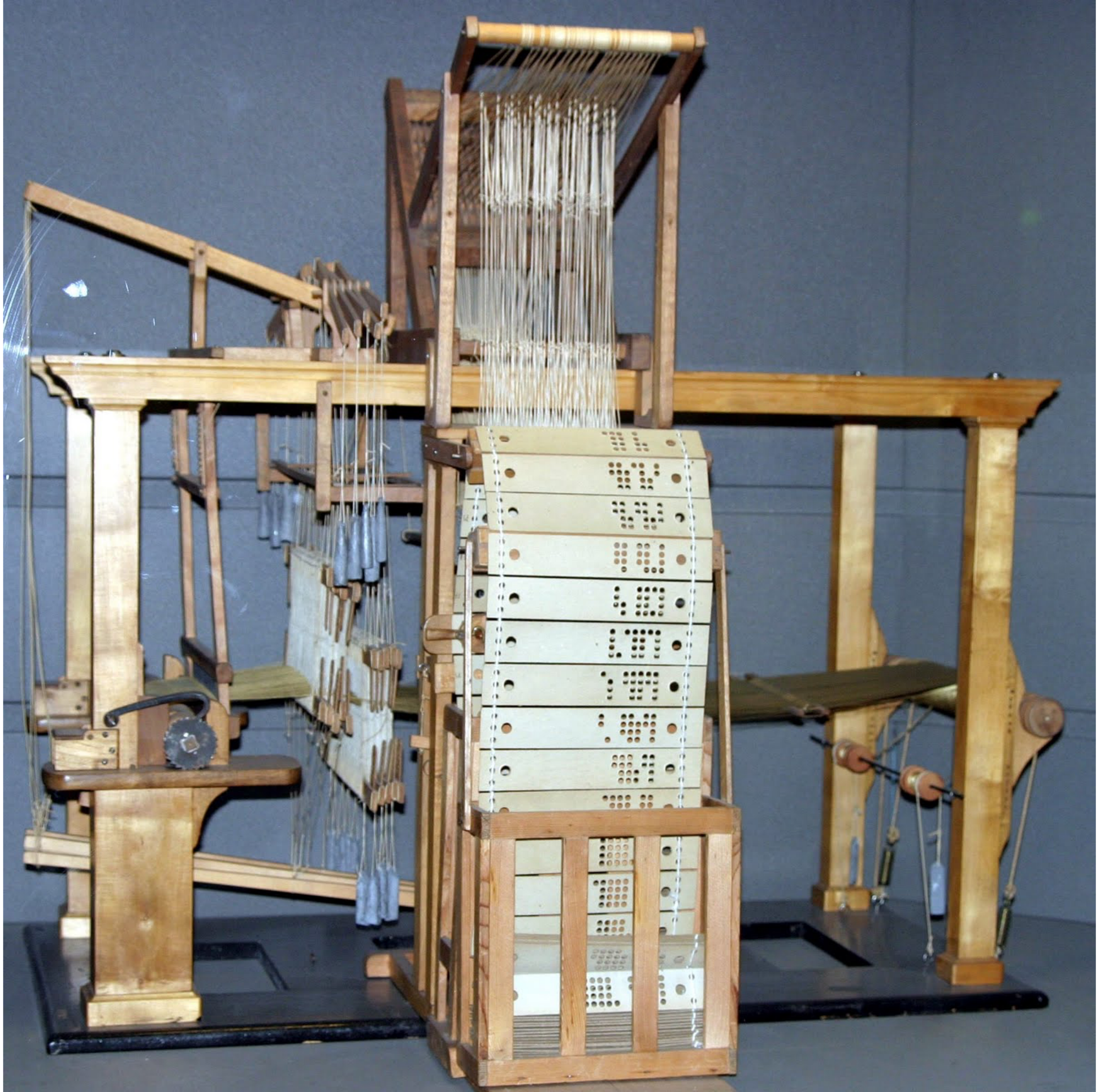
- Ruby
- OCaml
- Logic programming

Concepts too…

- Lambda calculus
- Type systems
- Semantics

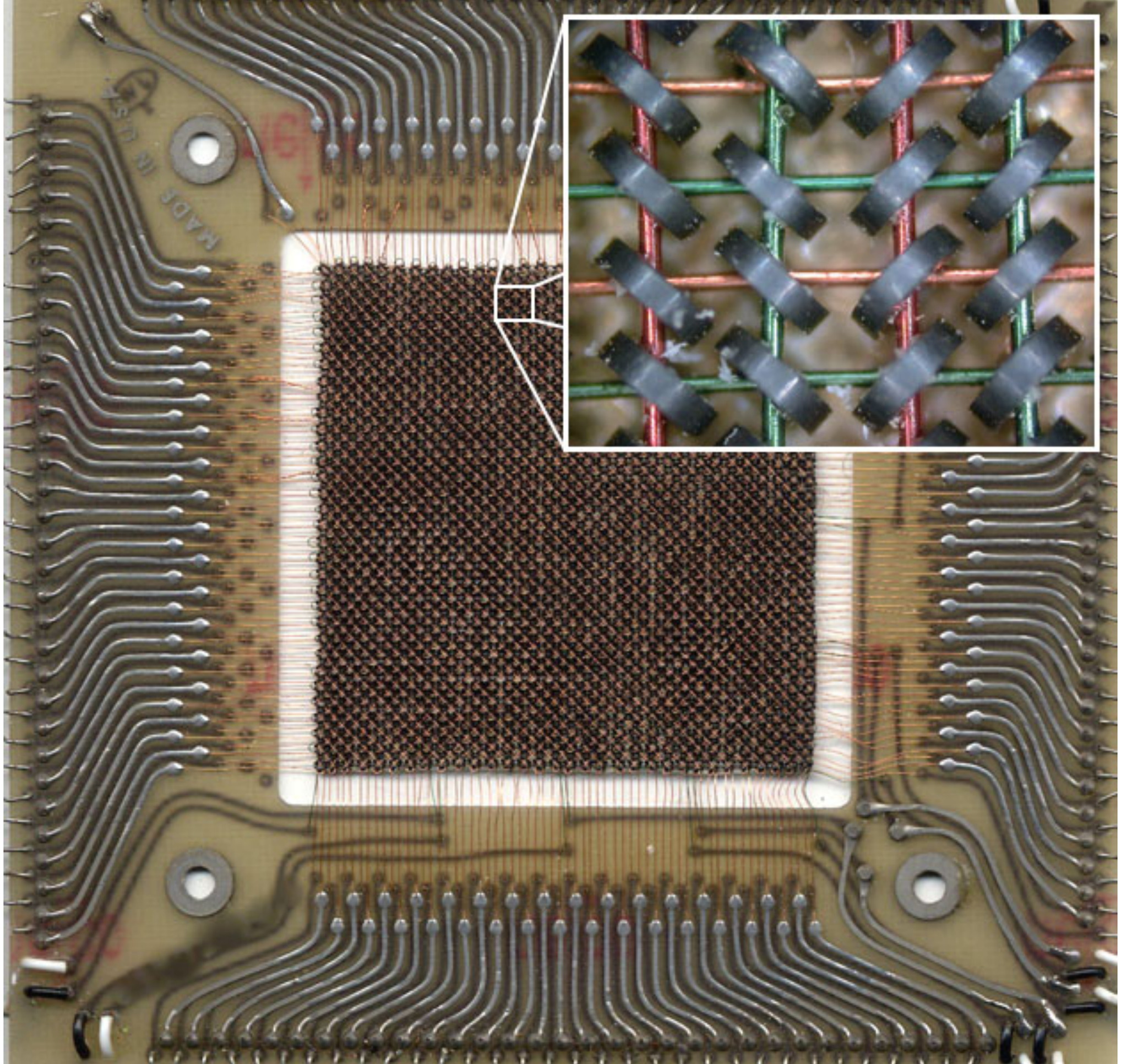
What is a Programming Language

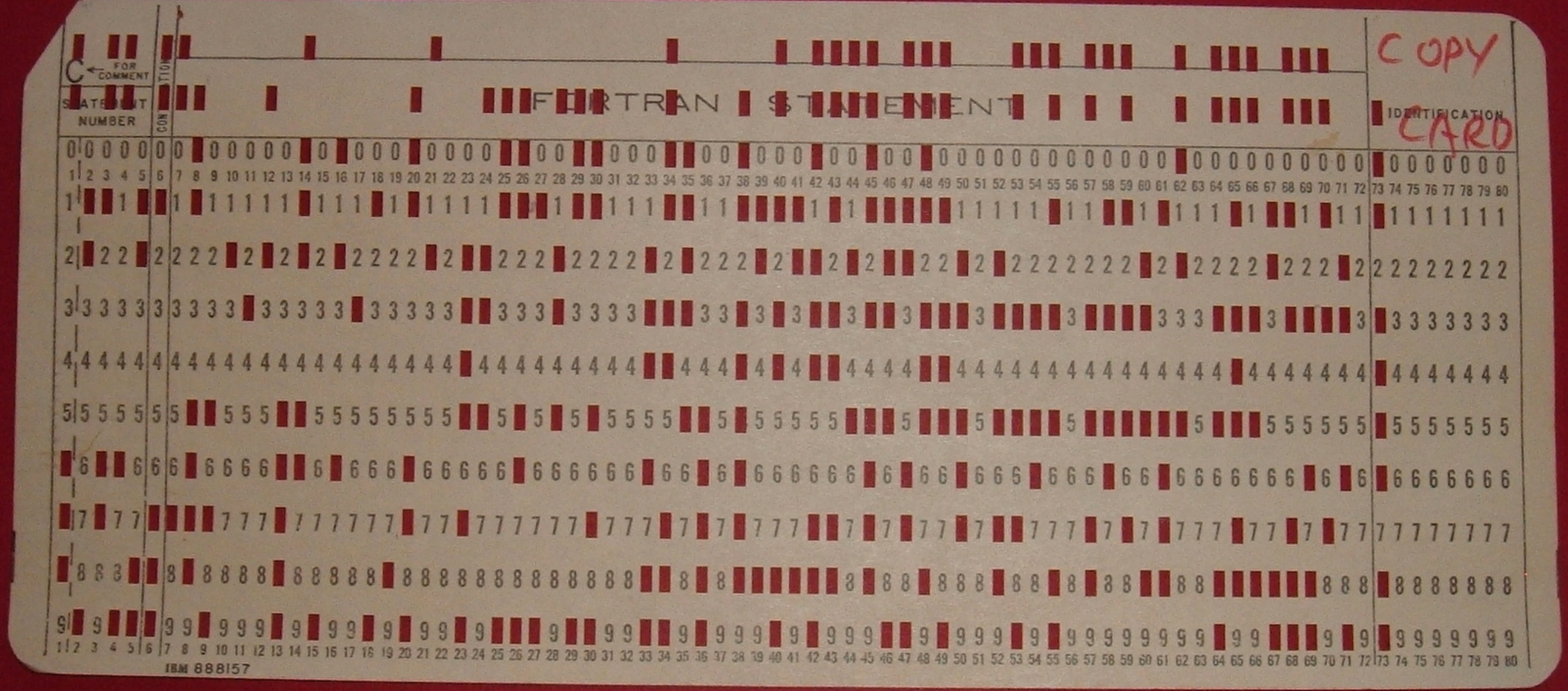
And how do we study it?



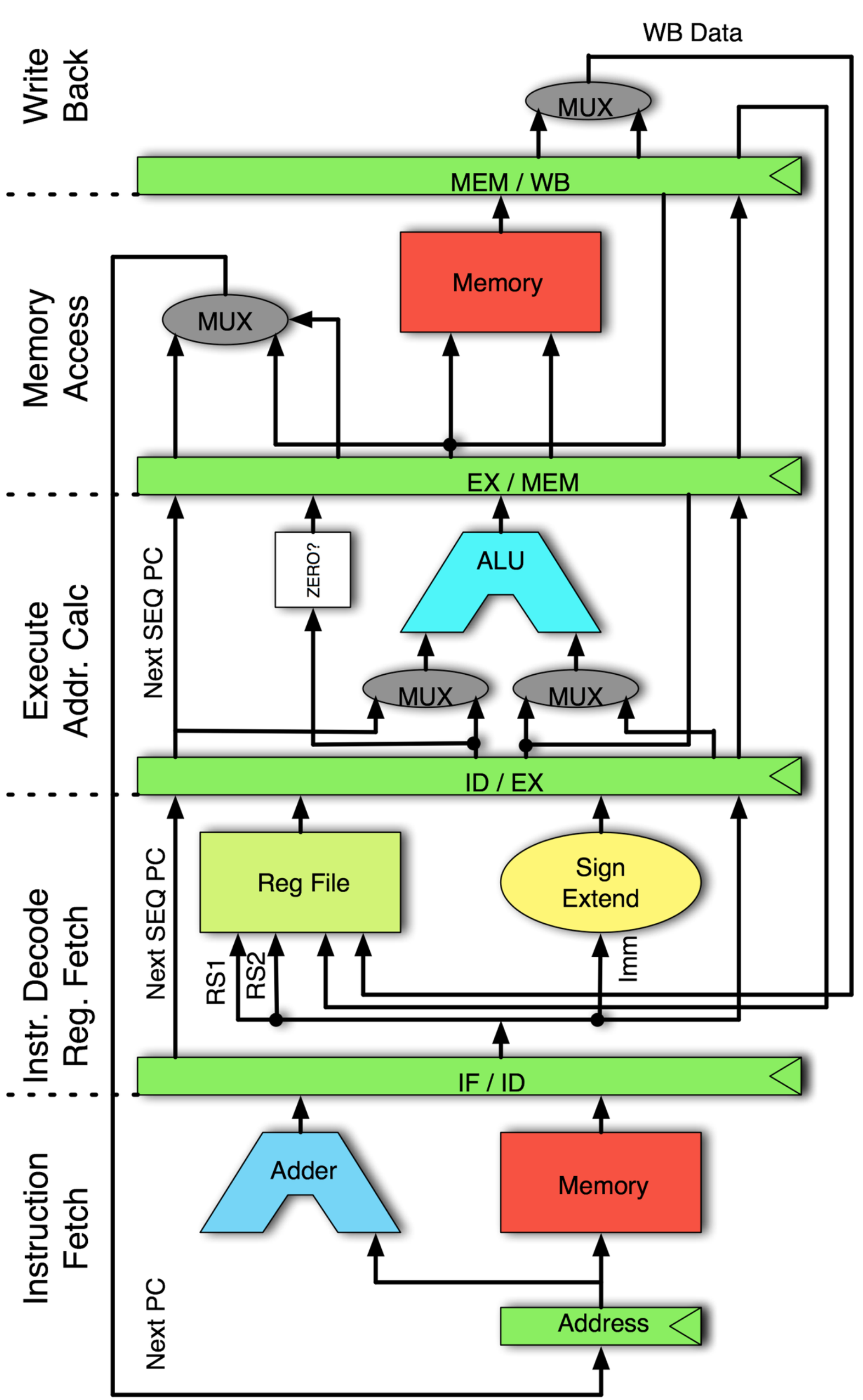
- PLs provide **abstraction mechanisms**
  - Control flow (goto/if/else/fail/etc…)
  - Structure (classes/modules/etc…)
  - Effects (memory mapped registers, concurrency)

- We think about the problem we want to solve
- Then we design an algorithm
- Then we implement, test, debug, and deploy it
- We want to use a PL that enables us to be succinct
- We want to write programs that are *obvious*


MADE IN USA



I don’t know how these work but they are apparently a thing.


WB Data
Write Back
MUX
MEM / WB
Memory
Memory Access
MUX
EX / MEM
ZERO?
ALU
Next SEQ PC
Execute Addr. Calc
MUX
MUX
ID / EX
Reg File
Sign Extend
Next SEQ PC
RS1
RS2
Imm
Instr. Decode Reg. Fetch
IF / ID
Adder
Memory
Instruction Fetch
Next PC
Address

- Realized that writing in binary wasn't very efficient
- Mispunching a card: causes program to crash
- (But misplacing a semicolon still does, too…)

```
C000                    ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START    LDS    #STACK

                        ***************************************
                        * FUNCTION: INITA - Initialize ACIA
                        * INPUT: none
                        * OUTPUT: none
                        * CALLS: none
                        * DESTROYS: acc A

0013            RESETA  EQU    %00010011
0011            CTLREG  EQU    %00010001

C003 86 13      INITA   LDA A  #RESETA   RESET ACIA
C005 B7 80 04           STA A  ACIA
C008 86 11              LDA A  #CTLREG   SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A  ACIA

C00D 7E C0 F1           JMP    SIGNON    GO TO START OF MONITOR

                        ***************************************
                        * FUNCTION: INCH - Input character
                        * INPUT: none
                        * OUTPUT: char in acc A
                        * DESTROYS: acc A
                        * CALLS: none
                        * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH     LDA A  ACIA      GET STATUS
C013 47                 ASR A            SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC    INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A  ACIA+1    GET CHAR
C019 84 7F              AND A  #$7F      MASK PARITY
C01B 7E C0 79           JMP    OUTCH     ECHO & RTS

                        ***************************************
                        * FUNCTION: INHEX - INPUT HEX DIGIT
                        * INPUT: none
                        * OUTPUT: Digit in acc A
                        * CALLS: INCH
                        * DESTROYS: acc A
                        * Returns to monitor if not HEX input

C01E 8D F0      INHEX   BSR    INCH      GET A CHAR
C020 81 30              CMP A  #'0       ZERO
C022 2B 11              BMI    HEXERR    NOT HEX
C024 81 39              CMP A  #'9       NINE
C026 2F 0A              BLE    HEXRTS    GOOD HEX
C028 81 41              CMP A  #'A
C02A 2B 09              BMI    HEXERR    NOT HEX
C02C 81 46              CMP A  #'F
C02E 2E 05              BGT    HEXERR
C030 80 07              SUB A  #7        FIX A-F
C032 84 0F      HEXRTS  AND A  #$0F      CONVERT ASCII TO DIGIT
C034 39                 RTS

C035 7E C0 AF  HEXERR   JMP    CTRL      RETURN TO CONTROL LOOP
```

```
0013            RESETA  EQU     %00010011
0011            CTLREG  EQU     %00010001

C003 86 13      INITA   LDA A   #RESETA    RESET ACIA
C005 B7 80 04           STA A   ACIA
C008 86 11              LDA A   #CTLREG    SET 8 BITS AND 2 STOP
C00A B7 80 04           STA A   ACIA

C00D 7E C0 F1           JMP     SIGNON     GO TO START OF MONITOR

                ****************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04   INCH    LDA A   ACIA       GET STATUS
C013 47                 ASR A              SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC     INCH       RECIEVE NOT READY
C016 B6 80 05           LDA A   ACIA+1     GET CHAR
C019 84 7F              AND A   #$7F       MASK PARITY
C01B 7E C0 79           JMP     OUTCH      ECHO & RTS

                ****************************************
                * FUNCTION: INHEX - INPUT HEX DIGIT
                * INPUT: none
                * OUTPUT: Digit in acc A
                * CALLS: INCH
                * DESTROYS: acc A
                * Returns to monitor if not HEX input
```

- Added a human readable form
    - With basic structure
    - Abstracting the binary code
    - But we still didn’t have modularity figured out

- Then a bunch of programming languages
  - Fortran (still used today!?)
  - Algol (code blocks, nested scope)
  - And perhaps most popular…

SECOND EDITION

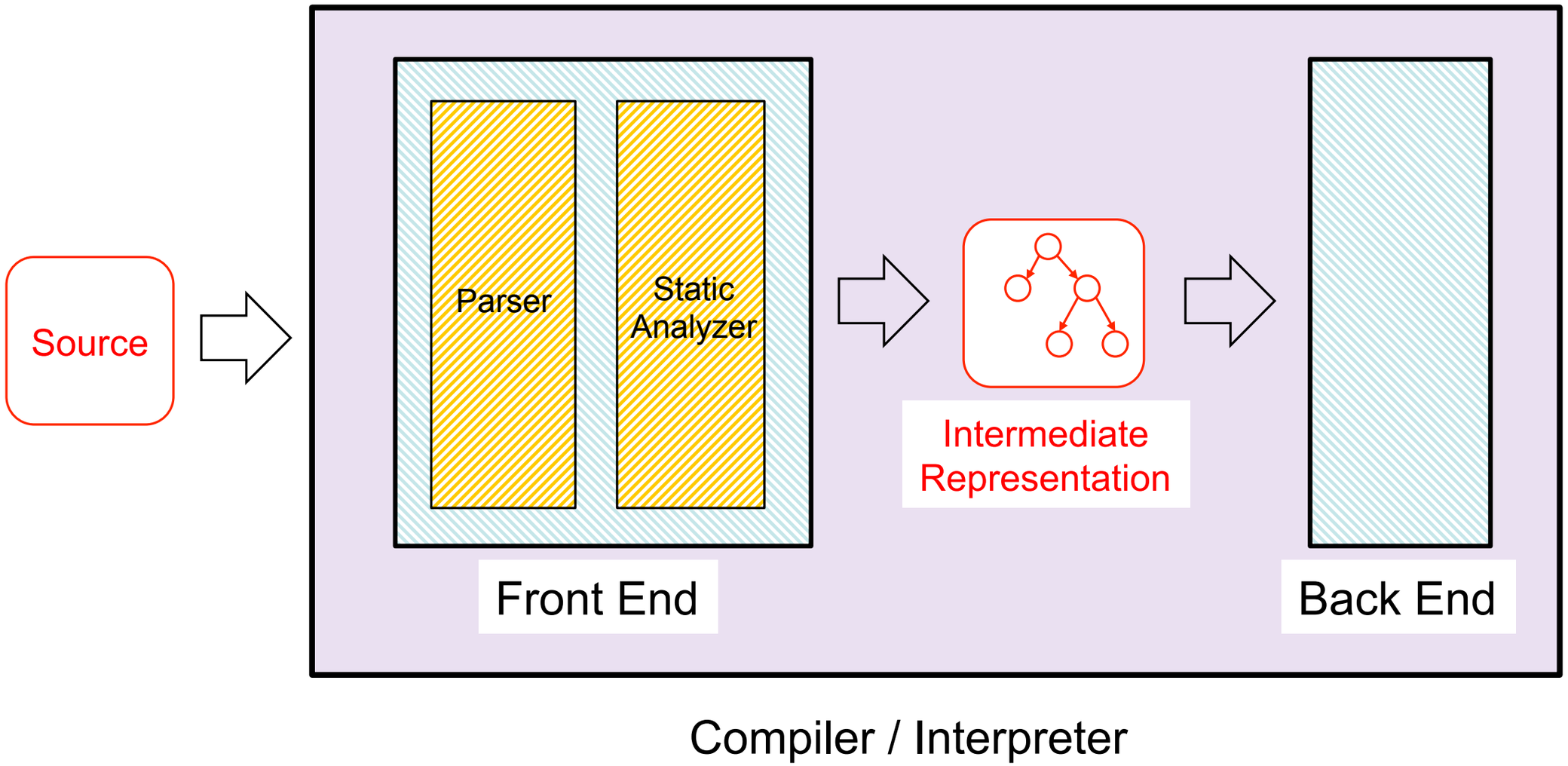
THE C PROGRAMMING LANGUAGE

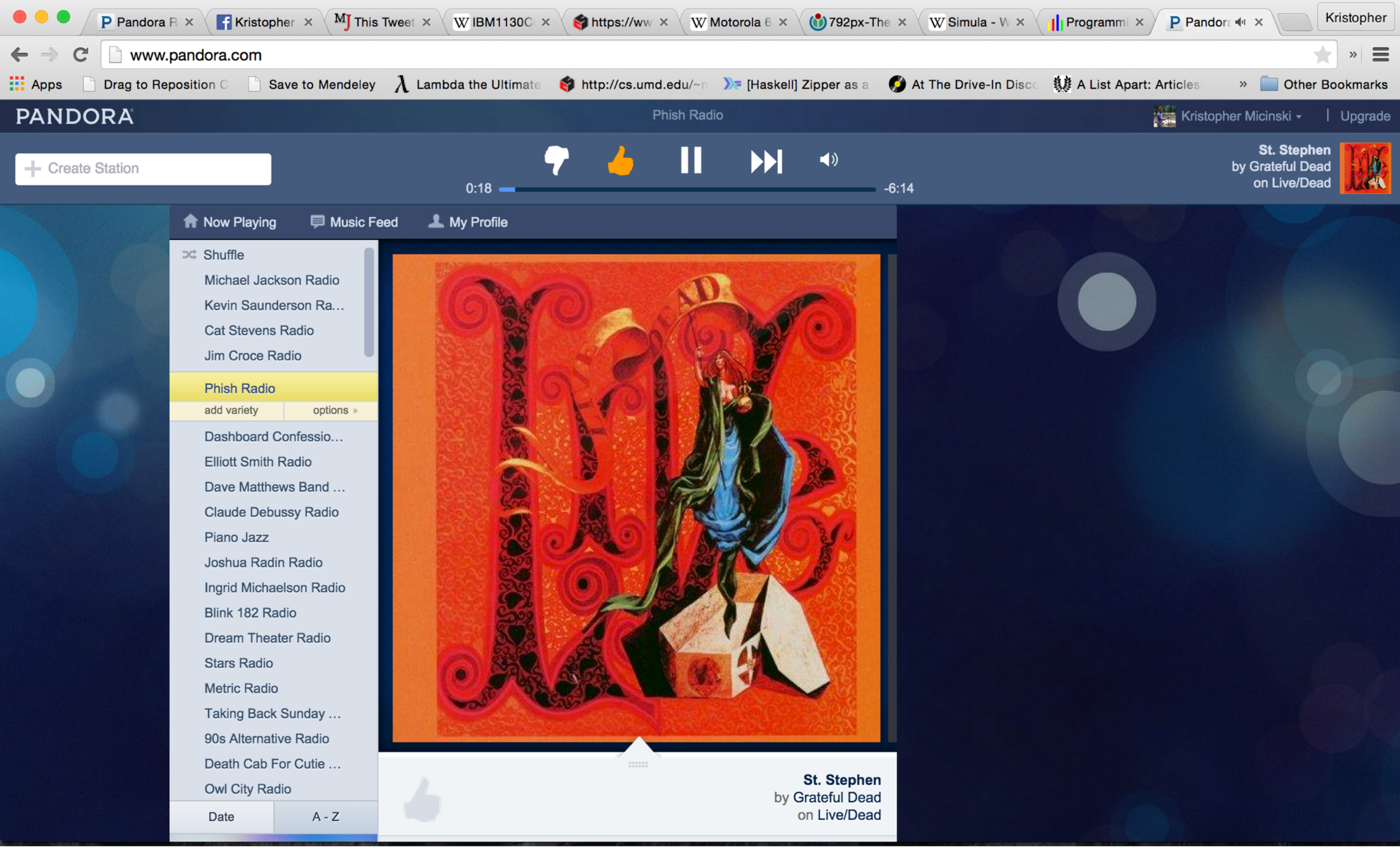
BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

PRENTICE HALL SOFTWARE SERIES

- Ancestor of most modern programming languages
- The abstraction model is over *memory*
  - I.e., a pointer represents a place in memory
- Very fast,
  - decades of work in optimizing compilers
- **C is macro assembly**

Code is **translated** to assembly


Source
Parser
Static Analyzer
Intermediate Representation
Front End
Back End
Compiler / Interpreter



What runs this?


CSS
JS
HTML

- JavaScript is **interpreted**
- An interpreter *reads* and *evaluates* its code
    - Instead of being compiled

Why not compiled?

- Needs to be portable, it runs in a browser
    - Browser written for each architecture
- Running untrusted binary code is *dangerous*

Why not compiled?

- JavaScript was initially for very small programs
  - Animated or blinking text, form validation
- GMail wasn’t even imagined at its inception
- And by that time, computers were fast enough

- Who writes Javascript?
  - Initially, not systems programmers, web dev
  - The abstraction of a *bunch of bits* was **wrong**
  - Abstraction: The DOM, event loop

Compilers vs. Interpreters

- Compilers
    - Generate fast code
- Interpreters
    - Great for debugging
    - *Typically* slower
- Typically:
    - Code that needs to be fast, written in C etc…
    - Scripting languages for scripts

Why study PL?

- Because it's fun?
- Study cross cutting building blocks
- Helps understand systems in a *foundational* way

Logistics

- Course webpage
- Piazza
  - Please post most questions here
- Office hours

Recitations

- On Fridays
- Serves as *review* time, but also **quiz / exam** time
  - Important to come to these!
  - First quiz **this Friday**

Grading

- 4 Projects (40%)
- 5 Quizzes (8%)
- 2 Exams (Midterm, final)
    - Final is comprehensive, emphasizes second half

Programming Projects

- Submitted on Submit server

- Similar to previous classes: public / private tests