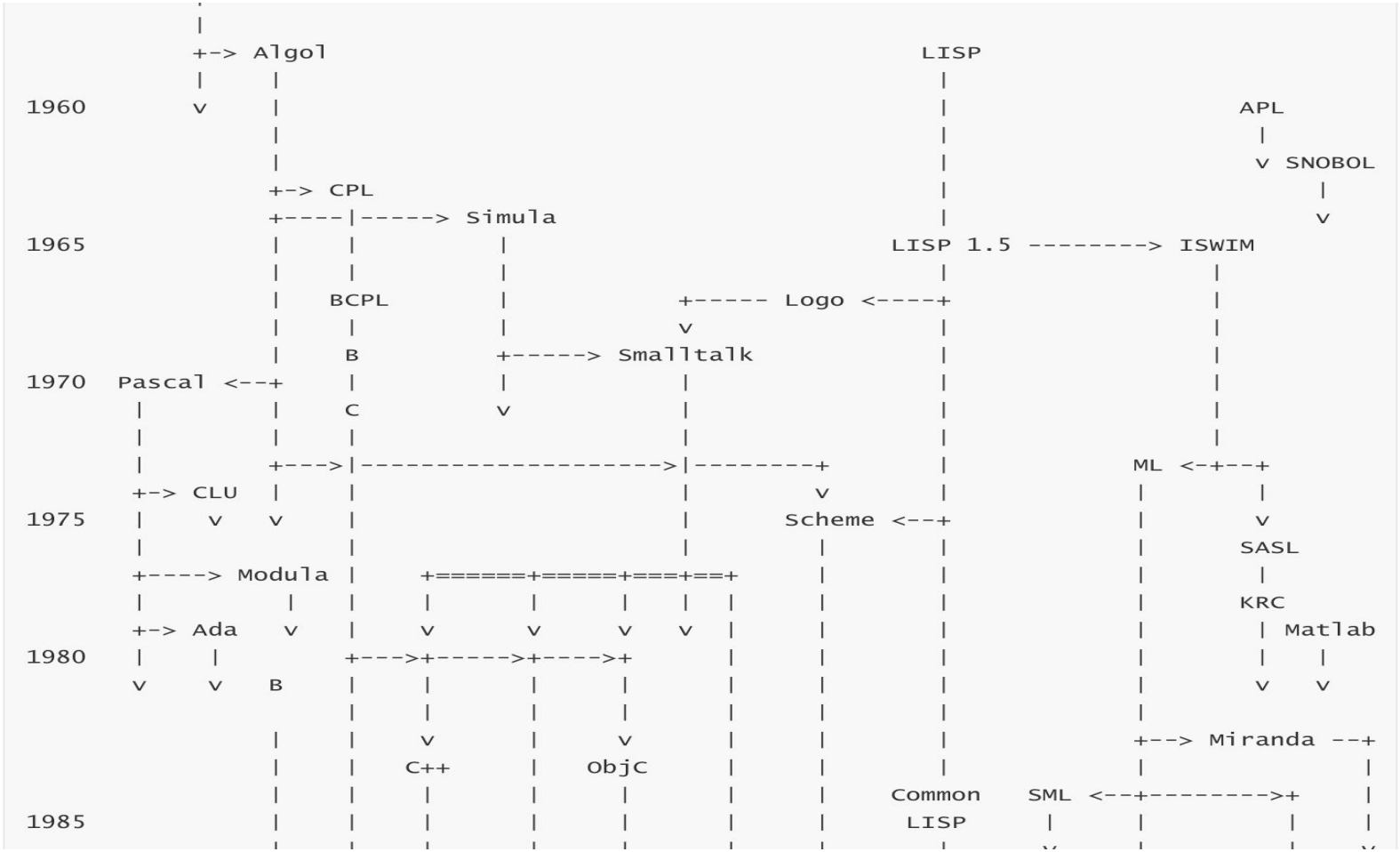
CSCI 3366 Programming Languages

Spring 2021


JavaScript
Python
Java
PHP
SQL
Ruby
C#
Bash
Delphi
C
C++
Go
Clojure
Haskell
Scala
Objective-C
Swift
LiveCode
CoffeeScript
F#
Erlang
Rust
R
Sass
Perl
Groovy
Lua
ColdFusio
Less
Elixir
Haxe
PowerShe
Lisp
Pascal
Action
TypeSc
Racket
GLSL
Emacs L
Ocaml
Awk
WLangua
Matlab
Prolog
Scheme
Assemb
Elm
Dart
Julia
Lingo
Basic
D
zsh
Octav
Small
Scrat
ML
Red
Appl
Fort
Kotl
Math

PLs don’t appear out of nowhere



About

- Design,
- Specification &
- Implementation of PLs

Main Goals

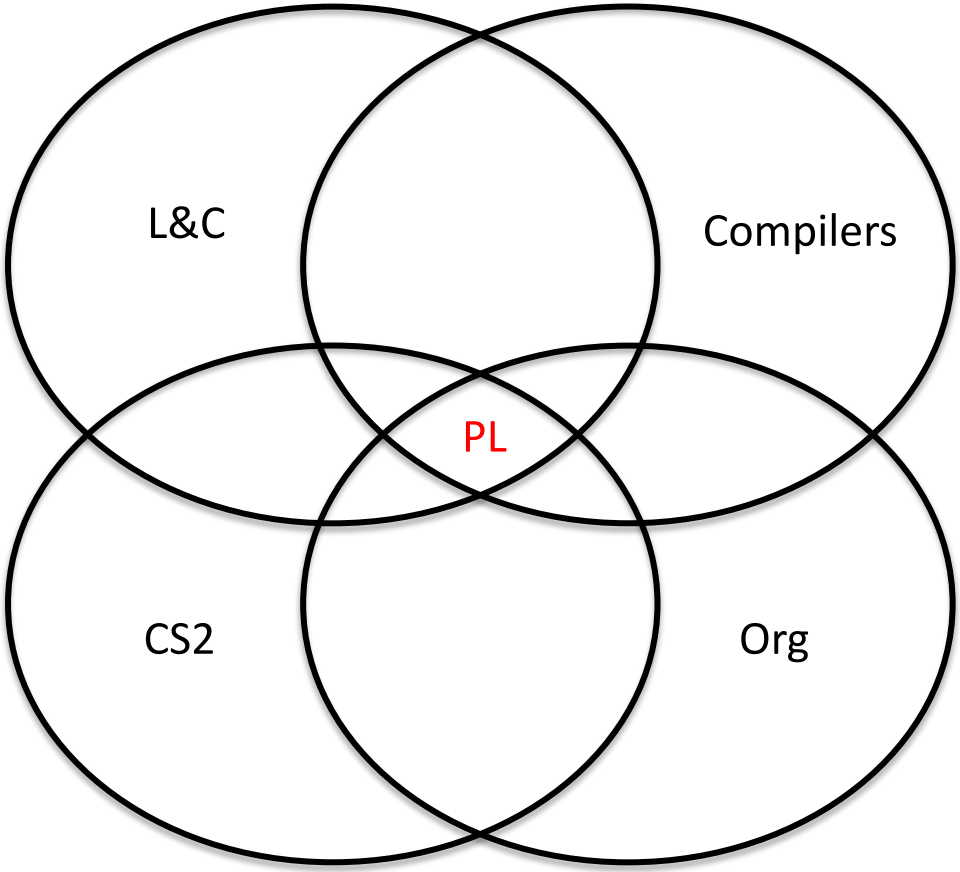
- Help students develop a better understanding of how programming languages “work”
- Help students develop a better sense of the relative strengths and weaknesses of PL choices for a given application
- Help students develop a better understanding of software.

Prerequisites

- CSCI 1102 Computer Science 2
- CSCI 2243 Logic & Computation

- CSCI 2272 Computer Organization (Recommended)

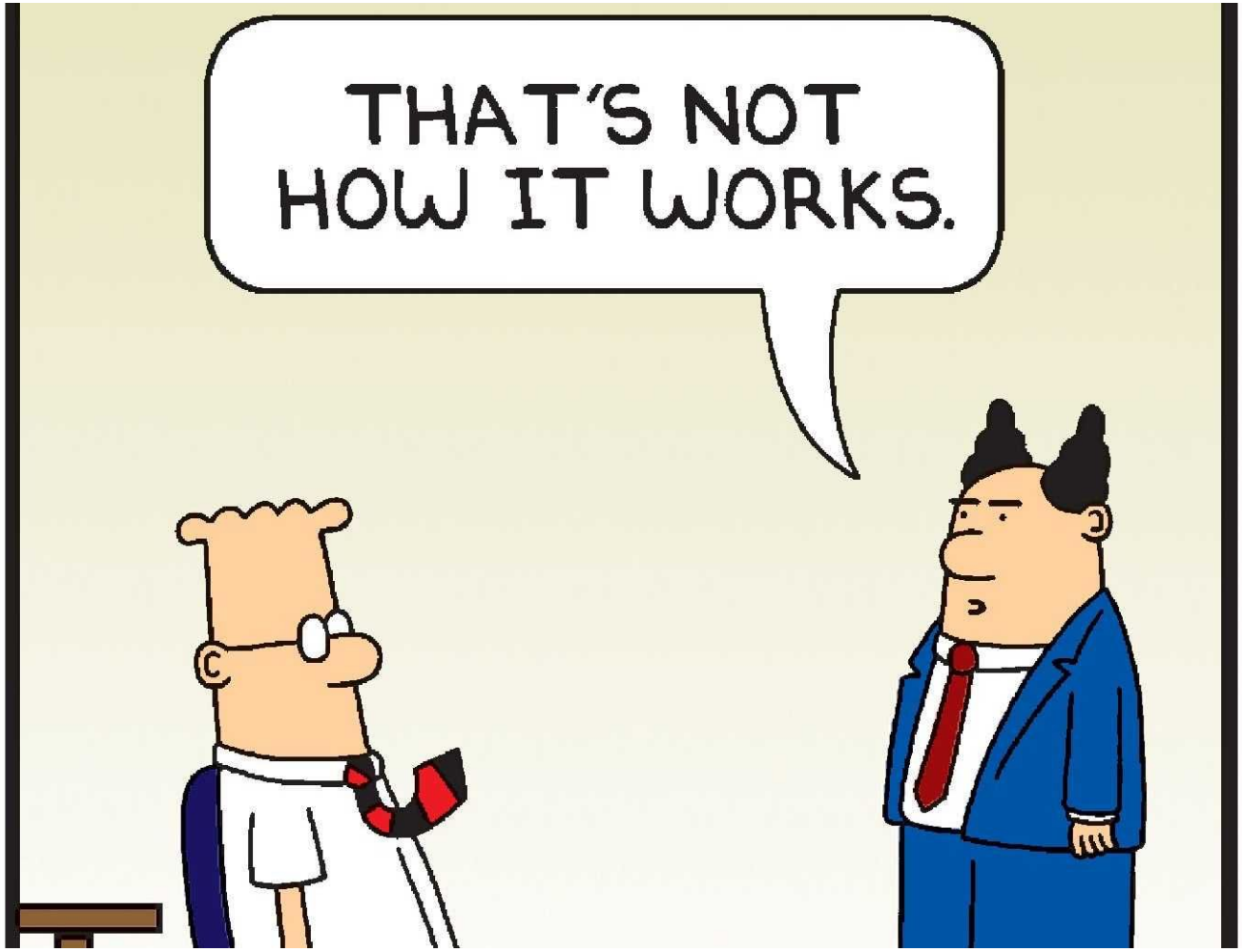
If you haven't had these, see me.


L&C
Compilers
PL
CS2
Org

What are reasonable design goals for a programming language?

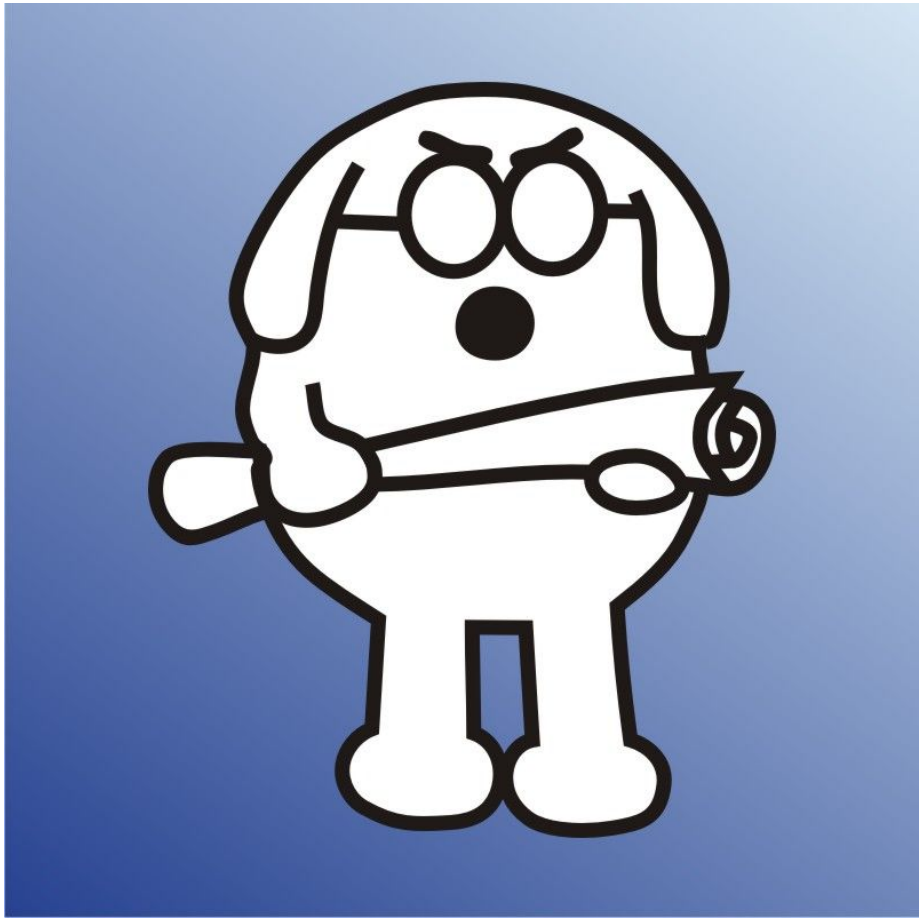
Four parties – software consumer

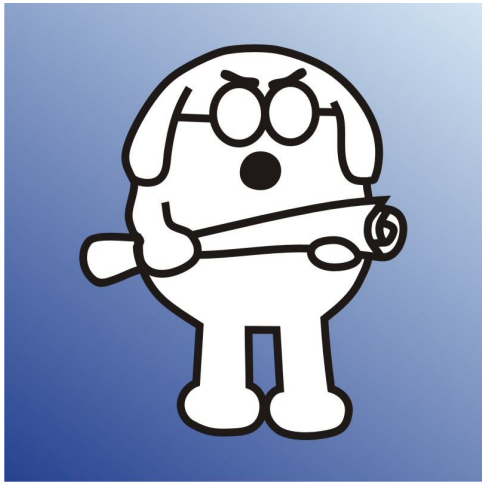
Four parties – software developer



Four parties – language implementor

Four parties – language designer





Spec

Application

Compiler

Tools

Application

~~Spec~~

Compiler

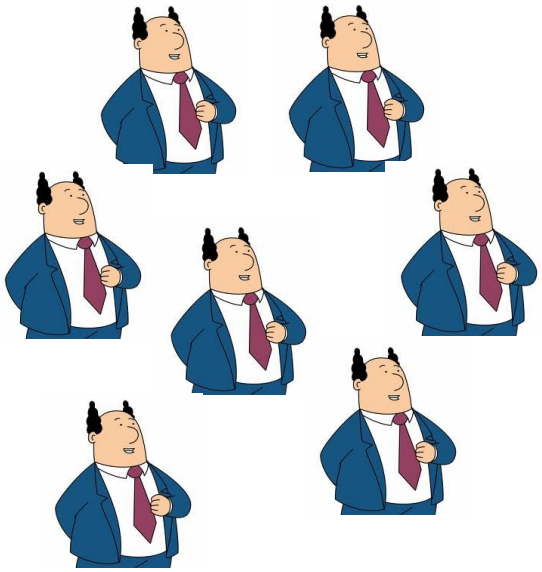
Tools

Application

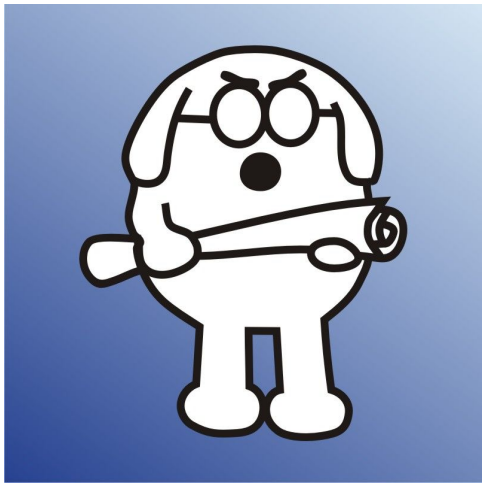
You're winging it …

Compiler

Tools



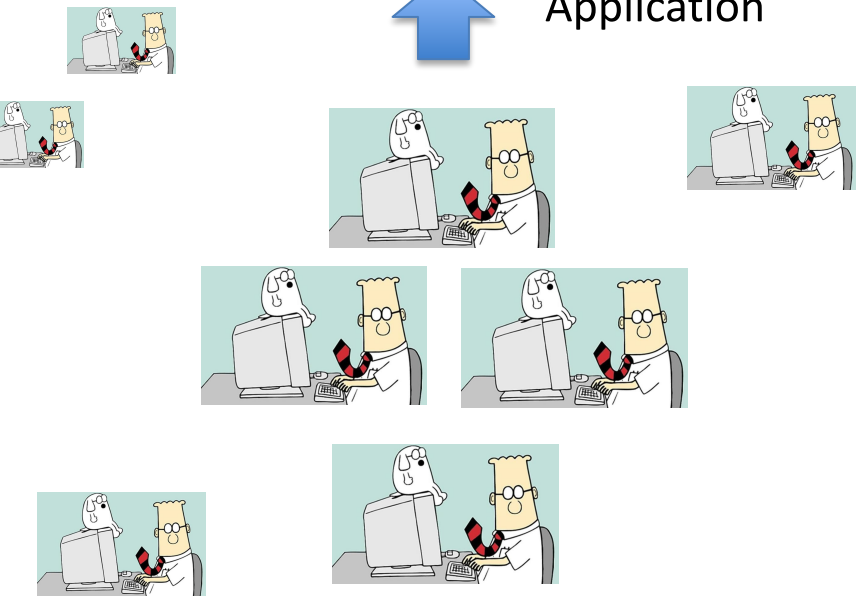
Application



Spec

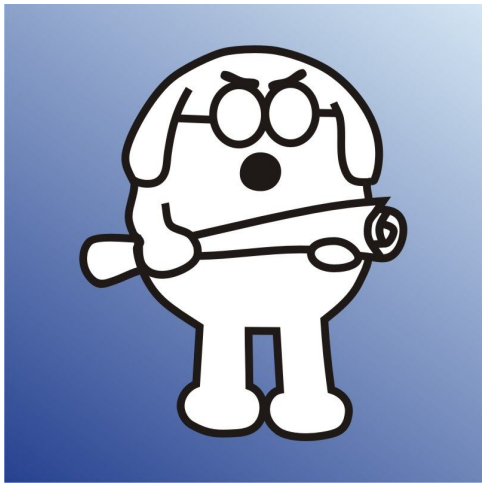
Compiler

Tools



Application

Correct?

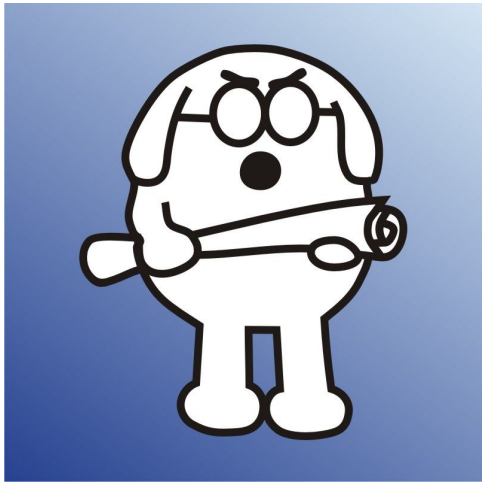


Spec

Compiler

Tools

Correct?



Application

Spec

Compiler

Tools

A Few Modest Design Goals

Consumer: PL is designed in such a way that it can be implemented in such a way that software consumers are justified in their confidence that the application code

1. does what the software developers think it does and
2. meets performance constraints.

A Few Modest Design Goals

Developer: PL is designed in such a way that it can be implemented in such a way that the software developers

1. have some confidence that their code “works”
   - correct
   - performant
2. can develop solutions efficiently
3. can manage the software life cycle efficiently - scale

Specification

Language = Syntax + Semantics

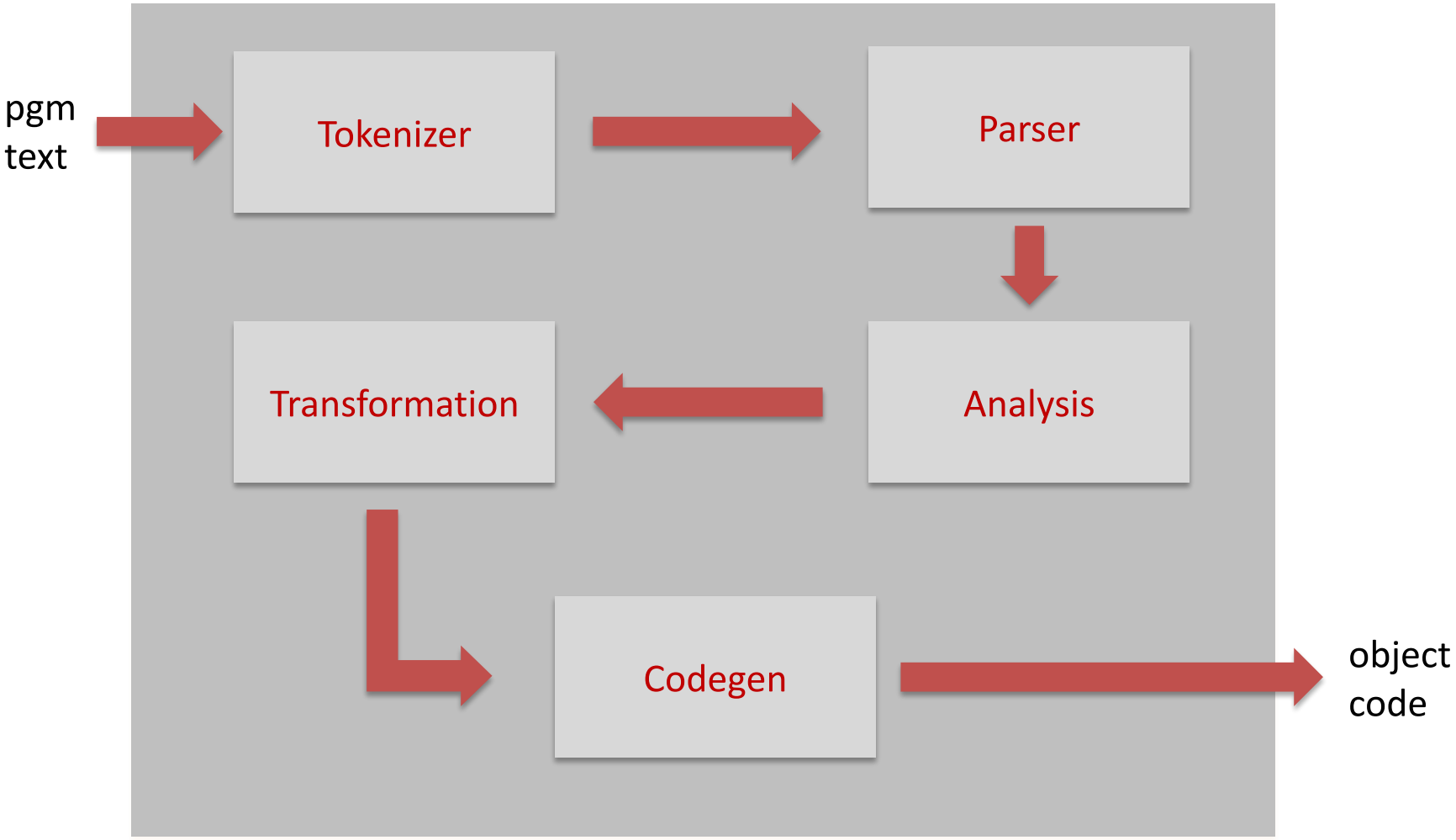
Syntax: specified with context free grammars

+ Trees

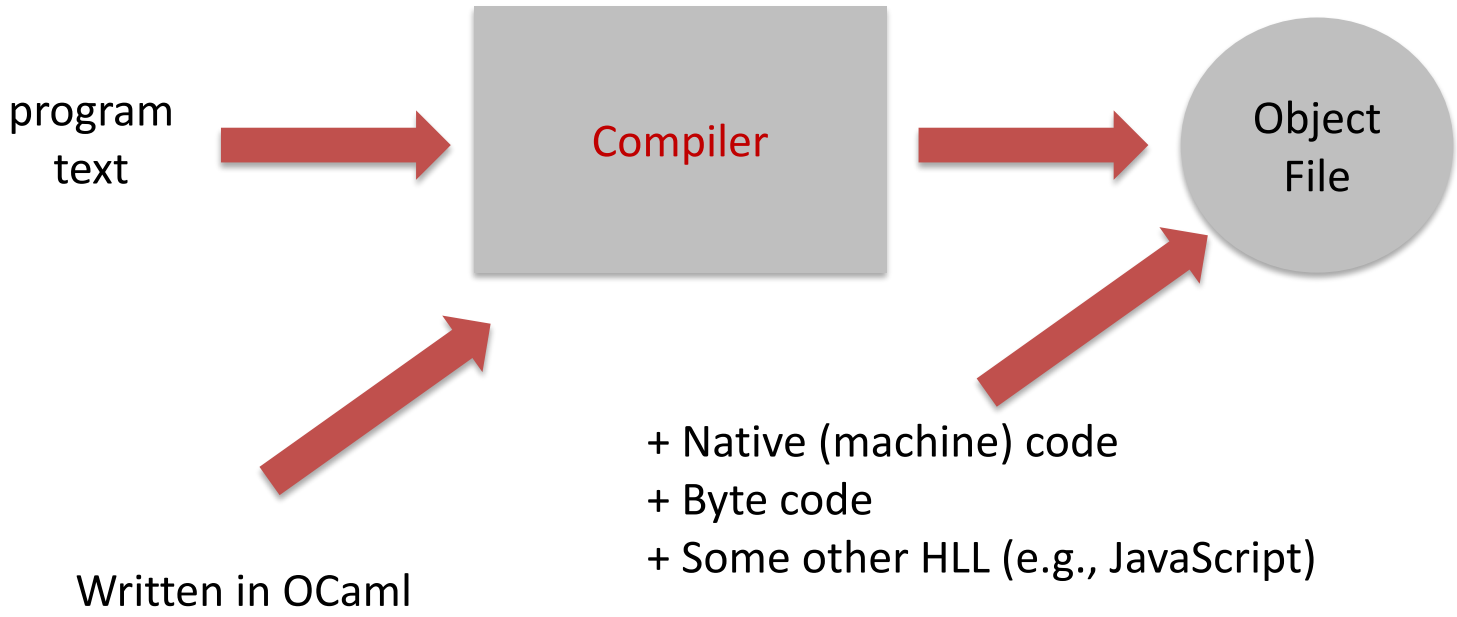
Semantics: specified with proof systems

+ More Trees

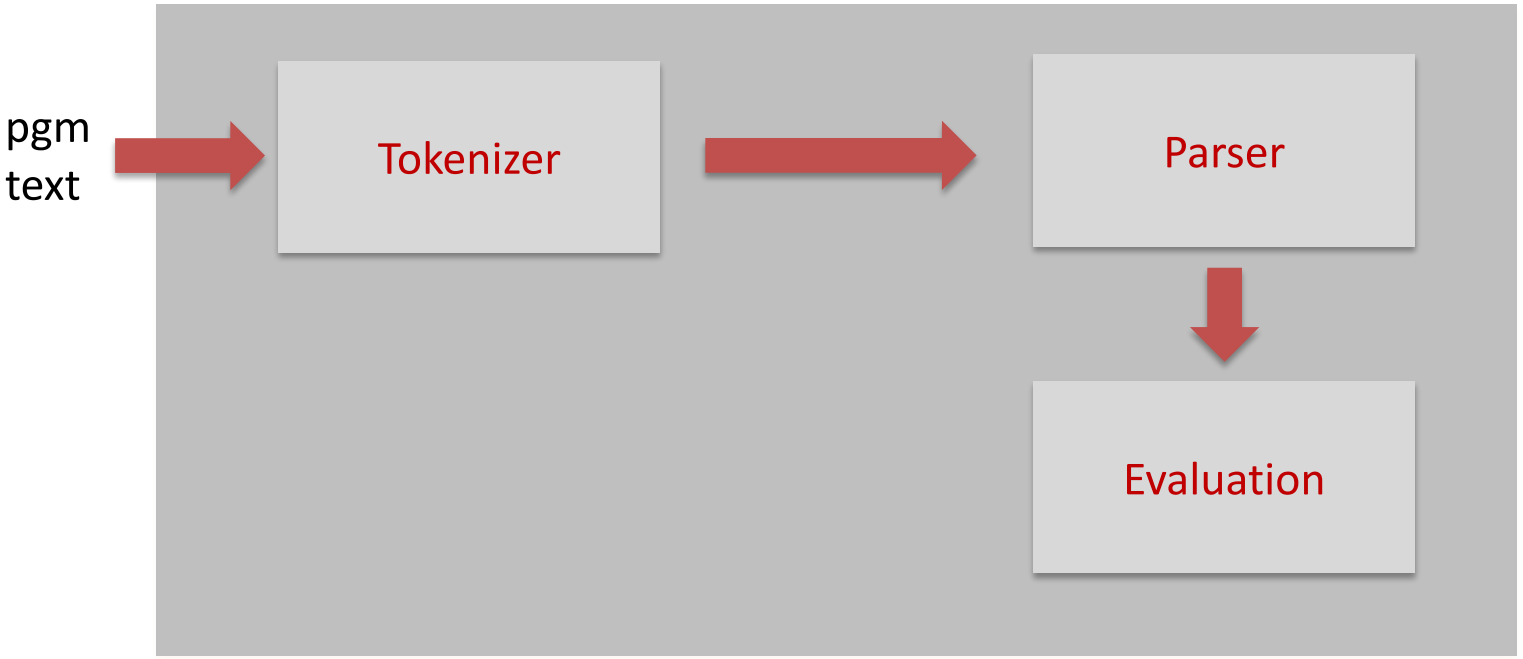
Language Implementation: Compiler



Language Implementation: Compiler



Language Implementation: Interpreter



Four Simple Languages

- Mercury – just integer expressions
- Venus – extend Mercury with a second type (reals) and names
- Earth – extend Venus with structured types and block-structured recursive functions
- Mars – restrict Earth to make it C-like, compiler

Types and Type Systems

- Types can be thought of as an *annotation* describing a piece of data.

- Useful form of communication:
  - Between compiler and programmer
  - Between programmers

- Framework for studying languages and their parts

Course Admin

Teaching Assistants

Brian Ward

Work

10 problem sets, mostly coding in OCaml (120 points), you'll implement most of the important phases of a simple compiler

Problem Sets
(Due Tuesdays at 6PM)

- PS0 (2): Join
- PS1 (6): Getting Started
- PS2 (10): Getting Started with OCaml
- PS3 (10): Tokens & Trees
- PS4 (12): Grammars & Parsing
- PS5 (12): A Call-by-Name Interpreter
- PS6 (12): Type Checking
- PS7 (14): MiniC Compiler Name; Unify
- PS8 (14): MiniC Compiler Lift; Infer
- PS9 (14): MiniC Compiler Control
- PS10 (14): MiniC Compiler Codegen

Draft Schedule

1. Introduction & background
2. Introduction to OCaml
3. More OCaml
4. Syntax & grammars
5. Mercury: parsing; natural semantics & evaluation
6. Venus: variables & multiple types; binding & scope; substitution & environments
7. Venus evaluation order: call-by-value & call-by-name

Draft Schedule

8. Typing Venus, Type Inference
9. Compiling Venus to MIPS
10. Earth: Block-Structured recursive functions, structured types
11. MiniC: Compiling to MIPS
12. Lambda Calculus
13. System F; Generics
14. Dependent Types, Review & wrap-up

Tools

OPAM

Tools

- git, GitHub and GitHub Classroom

  https://github.com/BC-CSCI3366/s21

- Text editor with OCaml Merlin support

- Canvas (for grades & panopto recordings)