The Tao of PLCC

Stoney Jackson
2024

Common Approaches to Programming Languages

- Tour of languages:
  - all breadth, no depth
  - Language *du jour* changes (but the fundamentals don't)
- Tour of language constructs (with example languages):
  - A little more depth, more focus on language fundamentals
- Build compiler/interpreter (assembly and/or a simulated machine):
  - Requires more details than is necessary, which means less time for the fundamentals
- Build language implementations using industrial strength tools
  lex, yacc, javacc, antlr
  - Generated code is complicated, so tools must be used without understanding

PLCC Approach

Provide a tool set that

- Generates understandable code
  students are encouraged to read the generated code
- Targets Java for semantics implementation
  abstracting away instruction sets, assembly, memory, architecture, linking/loading, etc.
- Easy to learn
  about 1-2 standard course weeks to learn the basics
- Allows courses to focus on semantics
  the essence of programming languages

PLCC Design Principle

Understandability

above all else
(performance, expressiveness, ...)

Use well known languages

- **Regex** for tokens
- **BNF** for syntax
- **Java** for semantics

Use easily understood algorithms

- First, longest-match for scanning
- Top-down, recursive descent parser

Use a single file to specify a language's:

- Lexical structure
- Syntactic structure
- Semantics

Support an incremental pedagogical approach

Use dynamic dispatch to simplify implementation of semantics across alternative BNF rules.

How I Teach PL using PLCC

I = Stoney Jackson
2024

Context

- Junior-level course
- Students have had
  - CS1&2 in Python (they can programming and know data structures)
  - Design using Java (they know Java and OO)
  - Software Development (they know tools like Git, GitLab, and GitPod)
    - Git + GitLab => Submission System
    - GitPod => Standard development environment
- Required for CS majors
- 15 week semester
- two 80m sessions per week
- 20-30 students

Schedule

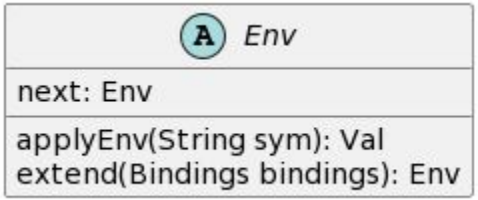
- W1-2: Language Construction
  - Tokens, Scanner, Regex, Parser, BNF, AST, PLCC toolset
  - **Variable binding/lookup (essence of semantics)**
- W3-6: Functional Paradigm
  - Static vs dynamic scoping
  - let, proc, letrec, +, -, *, add1, sub1, zero?, if
- W7-9: Call semantics
  - Side-effects
  - Pass-by-value, -reference, -name, -need, -copy-in-copy-out
- W10-11: Static Type System
- W12-15: Object-Oriented Paradigm
  - Classes, objects, static fields/methods, fields, methods, instantiation/object-construction, inheritance, shadowing

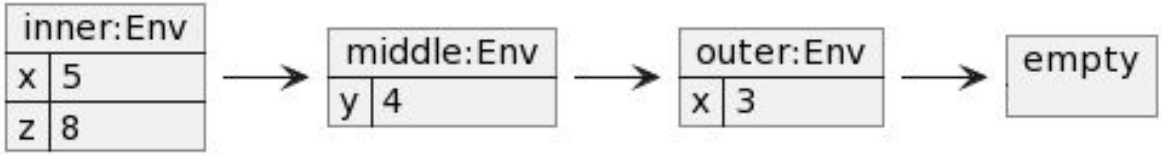
W	Session 1	Session 2	Due
1	Syllabus	Lexical Analysis	
2	Syntactic Analysis	Semantic Analysis	
3	Semantic Analysis	Environments	
4	V0	V1	**A1**
5	V2 - V3	V4	
6	V5	V6	**A2**
7	HOLIDAY	SET	
8	Review for Exam 1	NO CLASS	
9	**EXAM1** through V6	REF/NAME	**A3**
10	NAME/NEED	TYPE0	
11	TYPE1	TYPE1	
12	OBJ	Review for Exam 2	**A4**
13	**EXAM 2** through TYPE1	HOLIDAY	
14	OBJ	OBJ	
15	OBJ	Review for Exam 3	
16		**EXAM 3** through OBJ	**A5**

W1-2: Language Construction

- PLCC as a toolset
- **Lexical analysis** - tokens, regex, lexemes, scanner, first-longest match rule
- **Syntactic analysis** - BNF, left-most-derivation, top-down recursive-descent parser, mapping between rules and generated Java classes, AST construction
- **Semantic analysis** - walking ASTs with small methods and dynamic dispatch
- **Environments** - Data structure for variable binding and lookup (next slide)

Environments





Env.apply() semantics:

- inner.apply("x"); // 5
- inner.apply("y"); // 4
- inner.apply("z"); // 8
- middle.apply("x"); // 3
- middle.apply("y"); // 4
- middle.apply("z"); // Exception

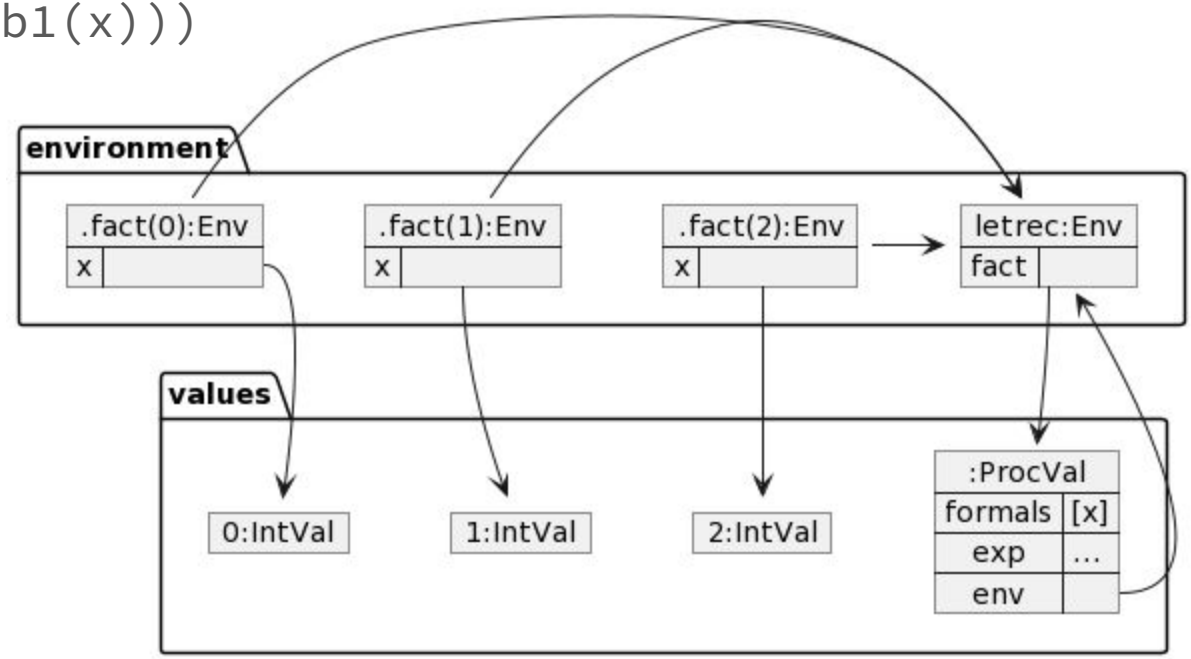
Weeks 3-6: Functional Paradigm

- Functional, expression language
- Closures
- Static-scoping
- Supports recursion
- Selection through if expressions
- Integer primitives
- Boolean values defined as 0 is False, non-0 is True
- Two types: integers and procs

Students practice understanding language concepts in terms of regex, BNF, semantics in Java, parse trees, and environment diagrams.

Weeks 3-6: Functional Paradigm: V0-V6

```
letrec
  fact = proc(x)
    if zero?(x) then 1
    else *(x, .fact(sub1(x)))
in
  .fact(2)
```



Weeks 7-9: Call Semantics

- SET - pass-by-value (adds side effects)
- REF - pass-by-reference
- NAME - pass-by-name
- NEED - pass-by-need
- CICO - pass-by-copy-in-copy-out (in homework)

Weeks 7-9: SET: add side-effects

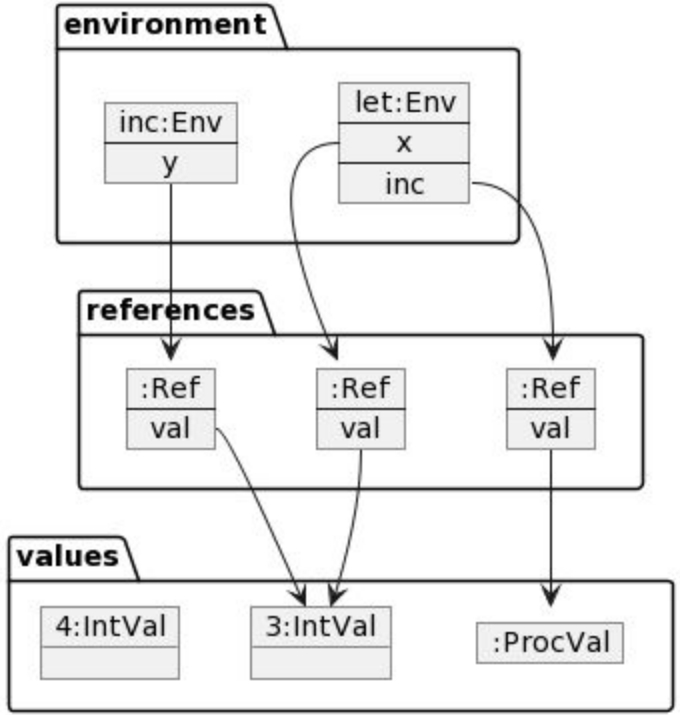
```
let
    x = 42
in {
    set x = +(x, 2) ;
    x
}

% => 44
```

Symbols are now bound to ***mutable*** Refs instead of Vals.
Enables side effects.

Weeks 7-9: SET - Pass-by-Value

```
let
       x = 3
       inc = proc(y) set y = add1(y)
in {
      . inc(x) ;
      x % => ??
}
```

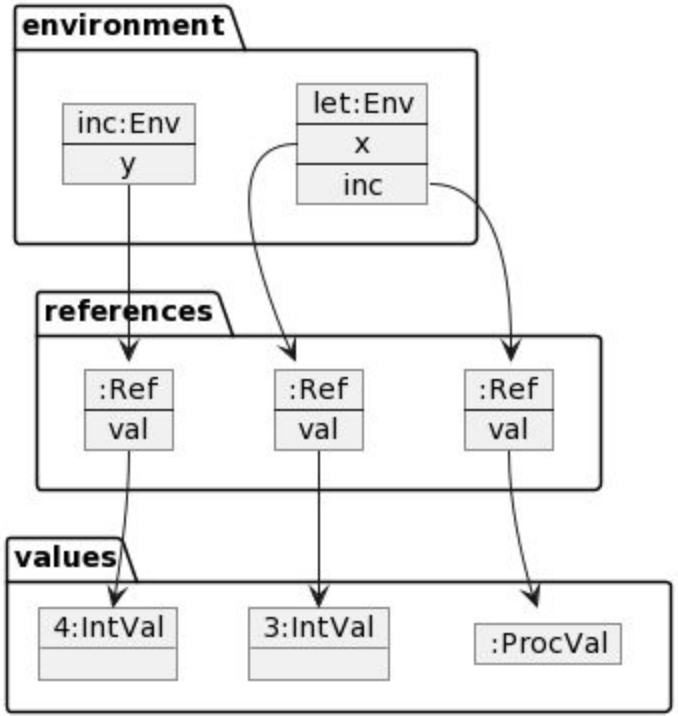
Before body of .inc(x) evaluated



Weeks 7-9: SET - Pass-by-Value

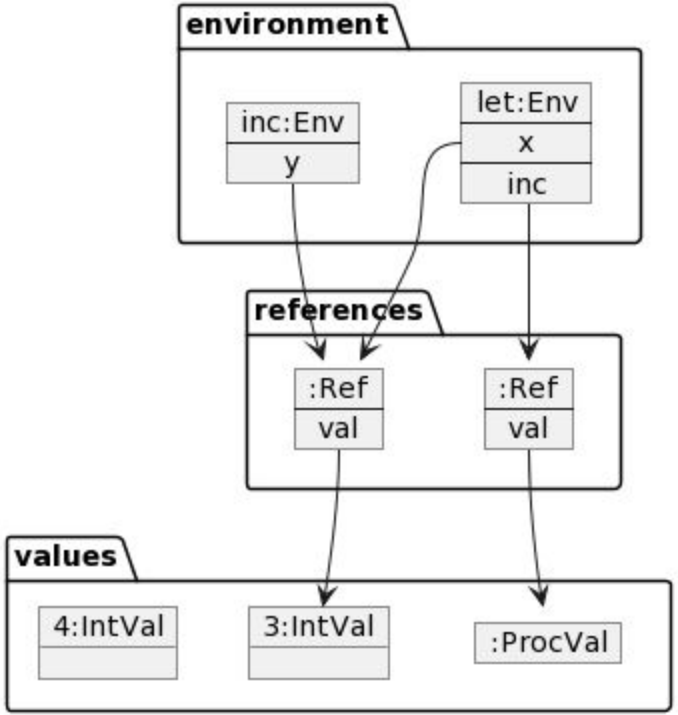
```
let
       x = 3
       inc = proc(y) set y = add1(y)
in {
      . inc(x) ;
      x % => 3
}
```

After .inc(x)



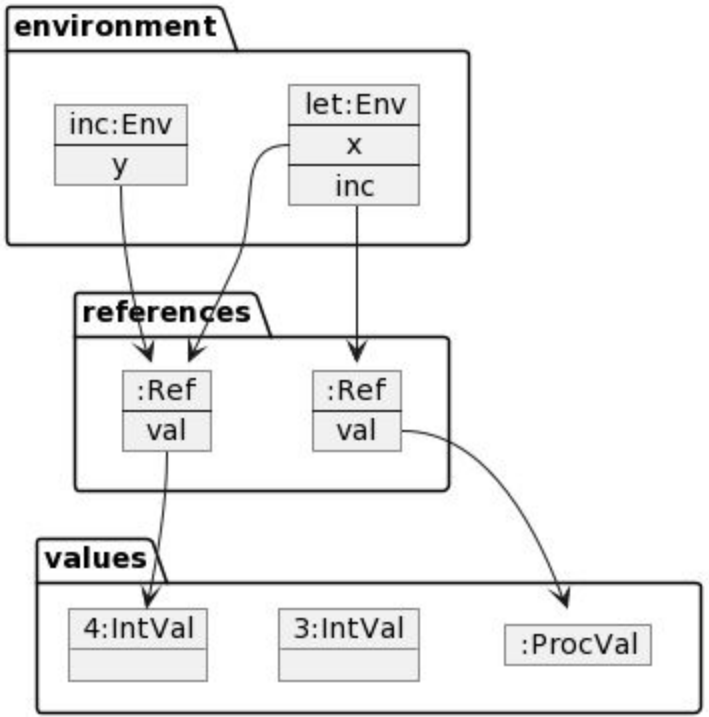
Weeks 7-9: REF - Pass-by-Reference

```
let
        x = 3
        inc = proc(y) set y = add1(y)
in {
       . inc(x) ;
       x % => ??
}
```



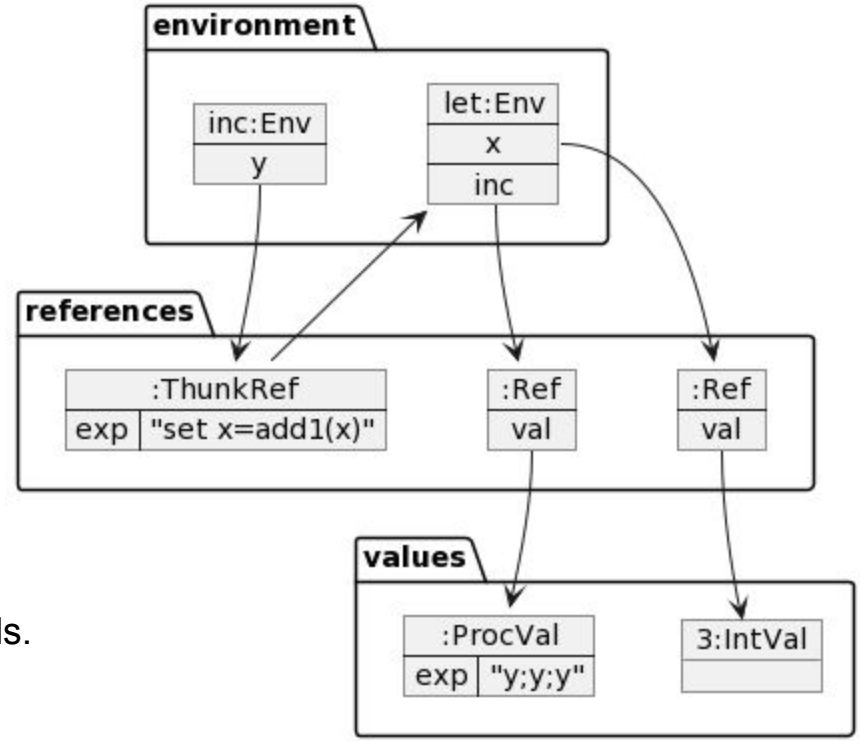
Weeks 7-9: REF - Pass-by-Reference

```
let
       x = 3
       inc = proc(y) set y = add1(y)
in {
      . inc(x) ;
      x % => 4
}
```



Weeks 7-9: NAME - Pass-by-Name (thunk)

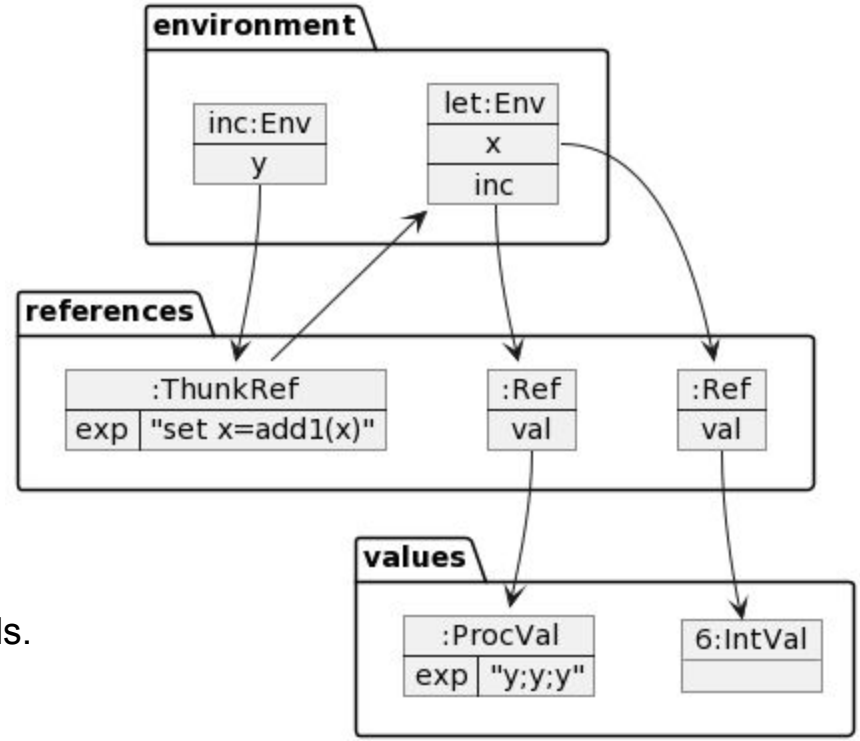
```
let
    x = 3
    inc = proc(y) { y; y; y }
in {
    . inc(set x = add1(x)) ;
    x % => ??
}
```



A thunk is similar to a proc without formals.
But it's a Ref.
thunk.deRef() evals its exp in calling env.

Weeks 7-9: NAME - Pass-by-Name (thunk)

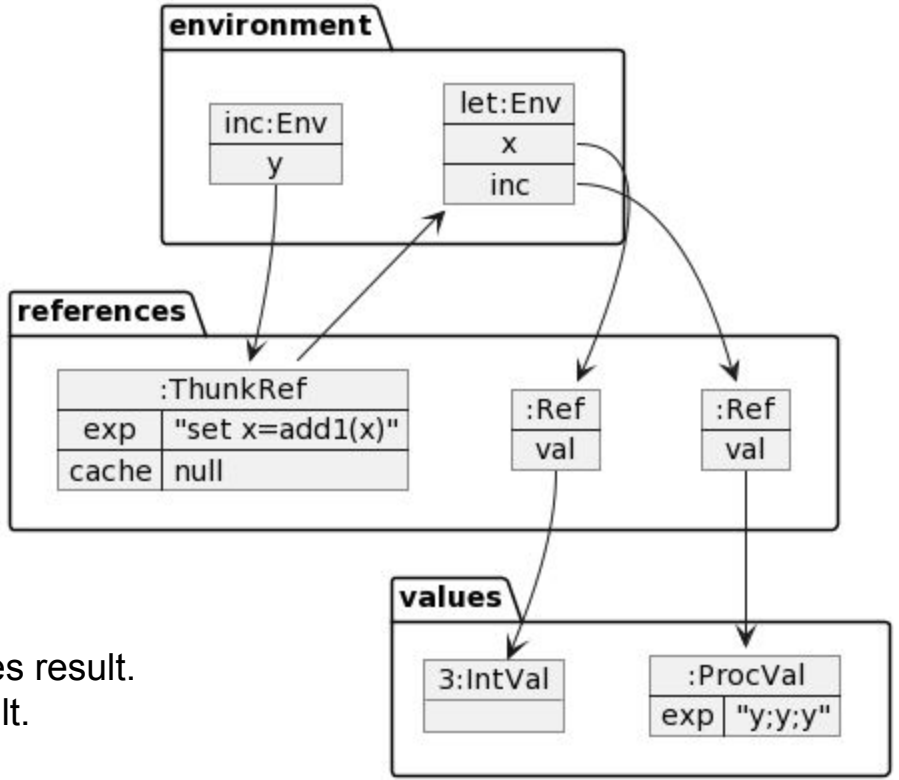
```
let
    x = 3
    inc = proc(y) { y; y; y }
in {
    . inc(set x = add1(x)) ;
    x % => 6
}
```



A thunk is similar to a proc without formals.
But it's a Ref.
thunk.deRef() evals its exp in calling env.

Weeks 7-9: NEED - Pass-by-Need

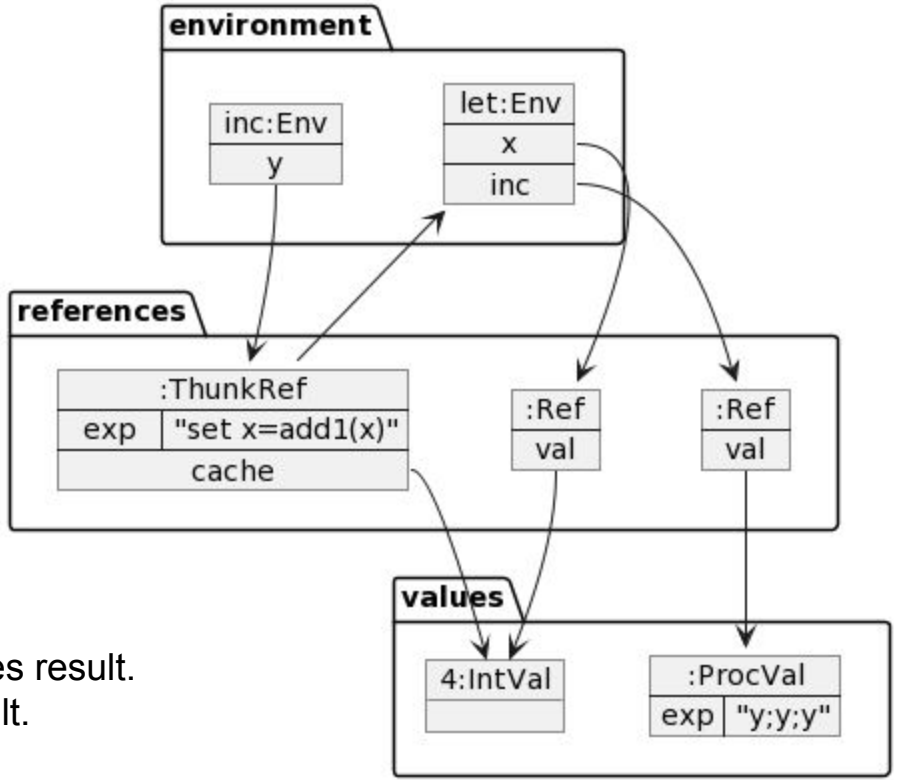
```
let
      x = 3
      inc = proc(y) { y; y; y }
in {
     . inc(set x = add1(x)) ;
     x % => ??
}
```



On first deRef(), thunk evaluates exp and caches result.
Subsequent deRef(), thunk returns cached result.
Lazy evaluation.

Weeks 7-9: NEED - Pass-by-Need

```
let
      x = 3
      inc = proc(y) { y; y; y }
in {
     . inc(set x = add1(x)) ;
     x % => 4
}
```



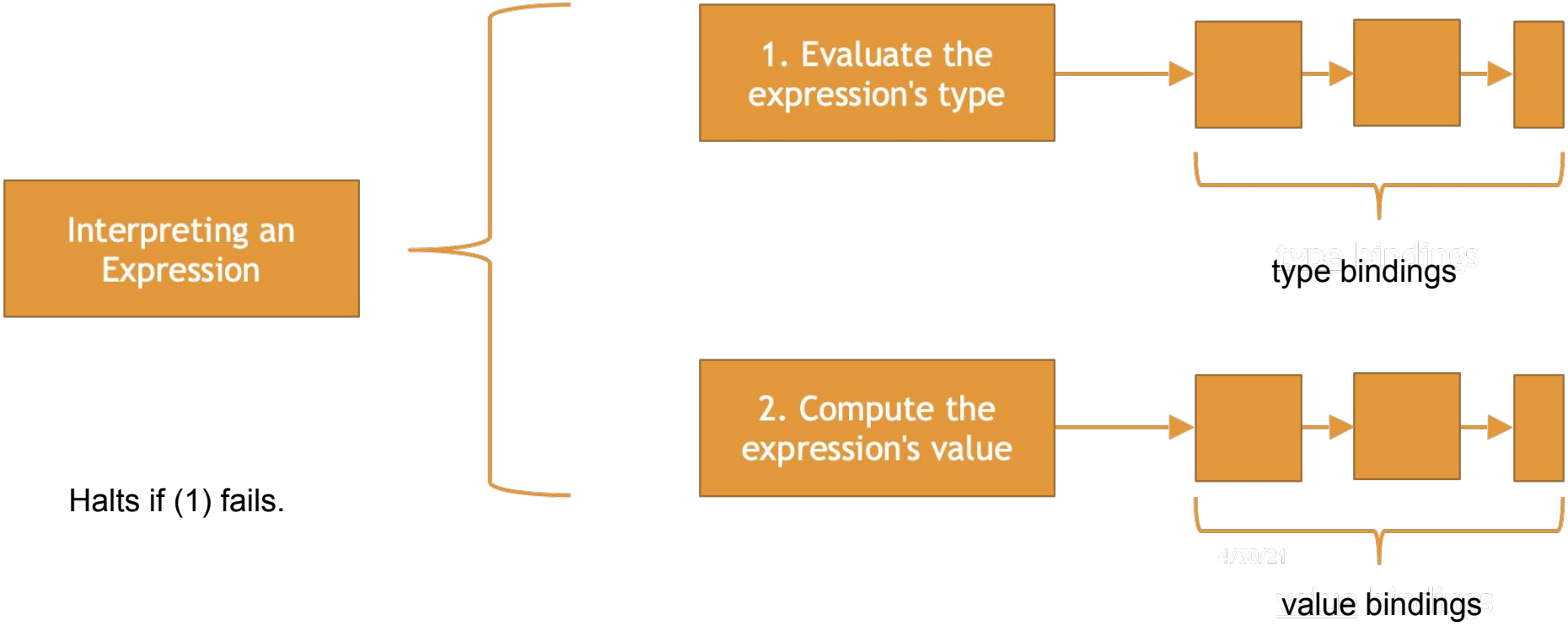
On first deRef(), thunk evaluates exp and caches result.
Subsequent deRef(), thunk returns cached result.
Lazy evaluation.

Weeks 10-11: Static Type Checking: TYPE0 and TYPE1:

```
letrec
    fact = proc(x: int): int
        if zero?(x) then 1
        else *(x, .fact(sub1(x)))
in
    .fact(5)
```

```
let
    twice = proc(f: [int => int], x: int): int {
        f(f(x))
    }
    add5 = proc(x: int): int  +(5, x)
in
    twice(add5, 10)   % => 20
```

Type Checking Design



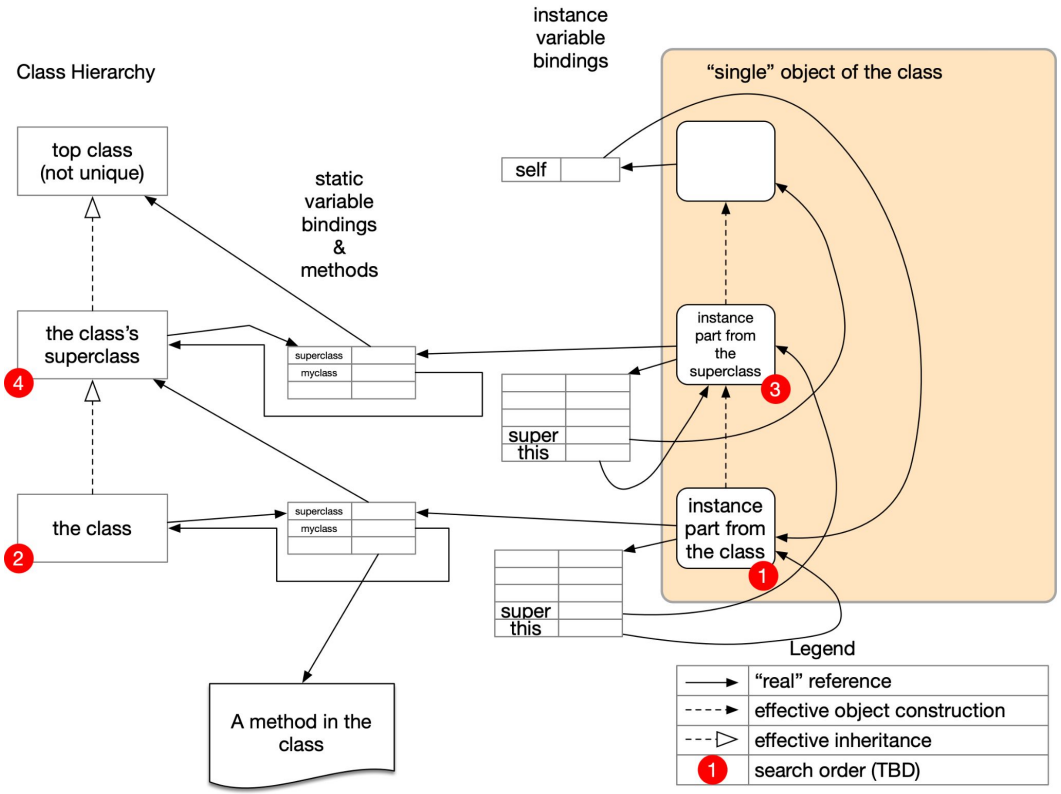
Halts if (1) fails.

Weeks 12-15: OBJ

```
define shape = class
    method area = proc() -1
end

define rectangle = class extends shape
    field lenn % length
    field widd % width

    method init = proc(lenn,widd) {set <self>lenn=lenn ; set <self>widd=widd ; self}
    method area = proc() *(lenn,widd)
end

define s = new shape
define r = .<new rectangle>init(4,5)
.<r>area() % => 20
.<s>area() % => -1
```

What is an Object?
An Environment!!!!

Weeks 12-15: OBJ: Making objects "Environments, Environments, Environments"



Schedule

- W1-2: Language Construction
  - Tokens, Scanner, Regex, Parser, BNF, AST, PLCC toolset
  - **Variable binding/lookup (essence of semantics)**
- W3-6: Functional Paradigm
  - Static vs dynamic scoping
  - let, proc, letrec, +, -, *, add1, sub1, zero?, if
- W7-9: Call semantics
  - Side-effects
  - Pass-by-value, -reference, -name, -need, -copy-in-copy-out
- W10-11: Static Type System
- W12-15: Object-Oriented Paradigm
  - Classes, objects, static fields/methods, fields, methods, instantiation/object-construction, inheritance, shadowing

W	Session 1	Session 2	Due
1	Syllabus	Lexical Analysis	
2	Syntactic Analysis	Semantic Analysis	
3	Semantic Analysis	Environments	
4	V0	V1	**A1**
5	V2 - V3	V4	
6	V5	V6	**A2**
7	HOLIDAY	SET	
8	Review for Exam 1	NO CLASS	
9	**EXAM1** through V6	REF/NAME	**A3**
10	NAME/NEED	TYPE0	
11	TYPE1	TYPE1	
12	OBJ	Review for Exam 2	**A4**
13	**EXAM 2** through TYPE1	HOLIDAY	
14	OBJ	OBJ	
15	OBJ	Review for Exam 3	
16		**EXAM 3** through OBJ	**A5**

Wrap Up

Other Languages/Topics

- Logic Based Languages (ABC)

  PLCC implementation of ABCDatalog
  <http://abcdatalog.seas.harvard.edu/>
  a subset of Prolog

  ```
  bear(fuzzy).
  bear(wuzzy).
  bear(X)?    % yields "bear(fuzzy)" and "bear(wuzzy)"
  ```

- Other languages: https://github.com/ourPLCC/languages

More Resources

https://github.com/ourPLCC/
https://plcc.pithon.net/

Let's look at what's there...

Future Work

- Targeting Python for semantics
- Generalize to target X for semantics

Join us!

Discord: https://discord.gg/EVtNSxS9E2

All resources: https://github.com/ourPLCC