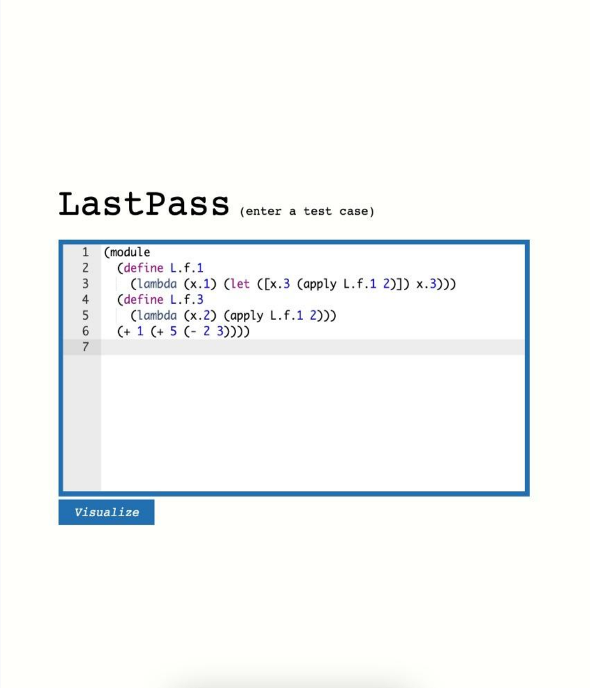
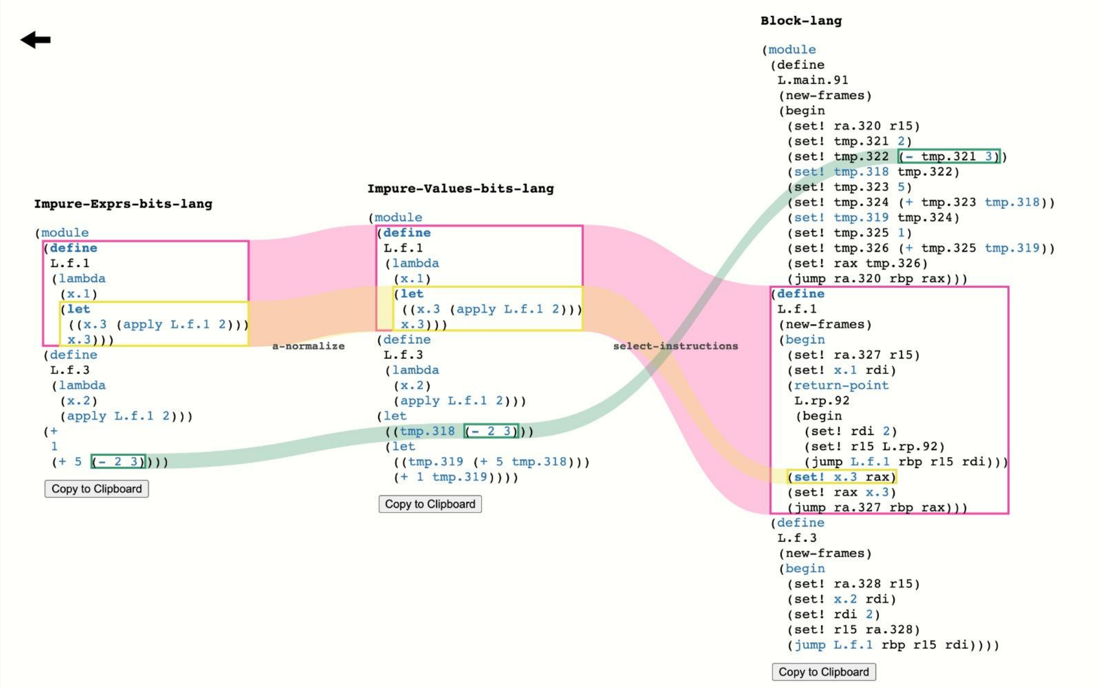
# Visualizing Compiler Passes with LastPass*

## Paulette, Braxton, and Jonathan (PB & J)



LastPass (enter a test case)

```
(module
  (define L.f.1
    (lambda (x.1) (let ([x.3 (apply L.f.1 2)]) x.3)))
  (define L.f.3
    (lambda (x.2) (apply L.f.1 2)))
  (+ 1 (+ 5 (- 2 3))))
```



Impure-Exprs-bits-lang

```
(module
 (define
  L.f.1
  (lambda
   (x.1)
   (let
    ((x.3 (apply L.f.1 2)))
    x.3)))
 (define
  L.f.3
  (lambda
   (x.2)
   (apply L.f.1 2)))
 (+
  1
  (+ 5 (- 2 3))))
```

Impure-Values-bits-lang

```
(module
 (define
  L.f.1
  (lambda
   (x.1)
   (let
    ((x.3 (apply L.f.1 2)))
    x.3)))
 (define
  L.f.3
  (lambda
   (x.2)
   (apply L.f.1 2)))
 (let
  ((tmp.318 (- 2 3)))
  (let
   ((tmp.319 (+ 5 tmp.318)))
   (+ 1 tmp.319))))
```

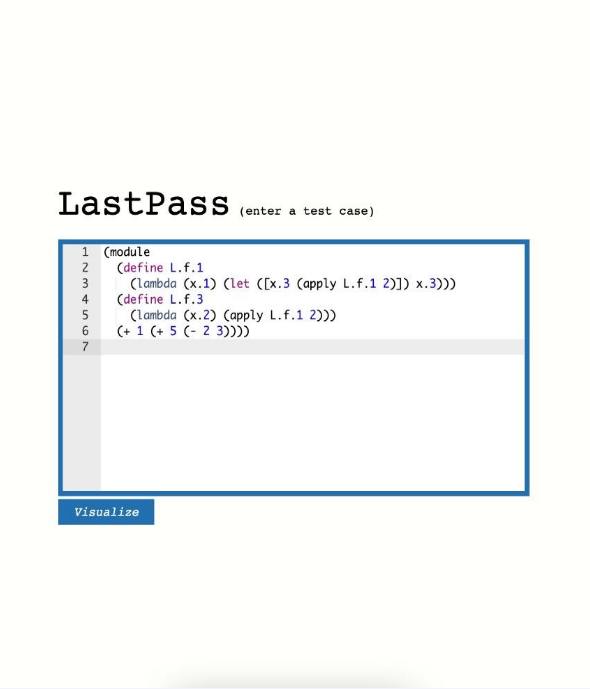
Block-lang

```
(module
 (define
  L.main.91
  (new-frames)
  (begin
   (set! ra.320 r15)
   (set! tmp.321 2)
   (set! tmp.322 (- tmp.321 3))
   (set! tmp.318 tmp.322)
   (set! tmp.323 5)
   (set! tmp.324 (+ tmp.323 tmp.318))
   (set! tmp.319 tmp.324)
   (set! tmp.325 1)
   (set! tmp.326 (+ tmp.325 tmp.319))
   (set! rax tmp.326)
   (jump ra.320 rbp rax)))
 (define
  L.f.1
  (new-frames)
  (begin
   (set! ra.327 r15)
   (set! x.1 rdi)
   (return-point
    L.rp.92
    (begin
     (set! rdi 2)
     (set! r15 L.rp.92)
     (jump L.f.1 rbp r15 rdi)))
   (set! x.3 rax)
   (set! rax x.3)
   (jump ra.327 rbp rax)))
 (define
  L.f.3
  (new-frames)
  (begin
   (set! ra.328 r15)
   (set! x.2 rdi)
   (set! rdi 2)
   (set! r15 ra.328)
   (jump L.f.1 rbp r15 rdi))))
```

* formerly SecondPass†
† formerly FirstPass‡
‡ formerly Untitled Compiler Pass Visualization

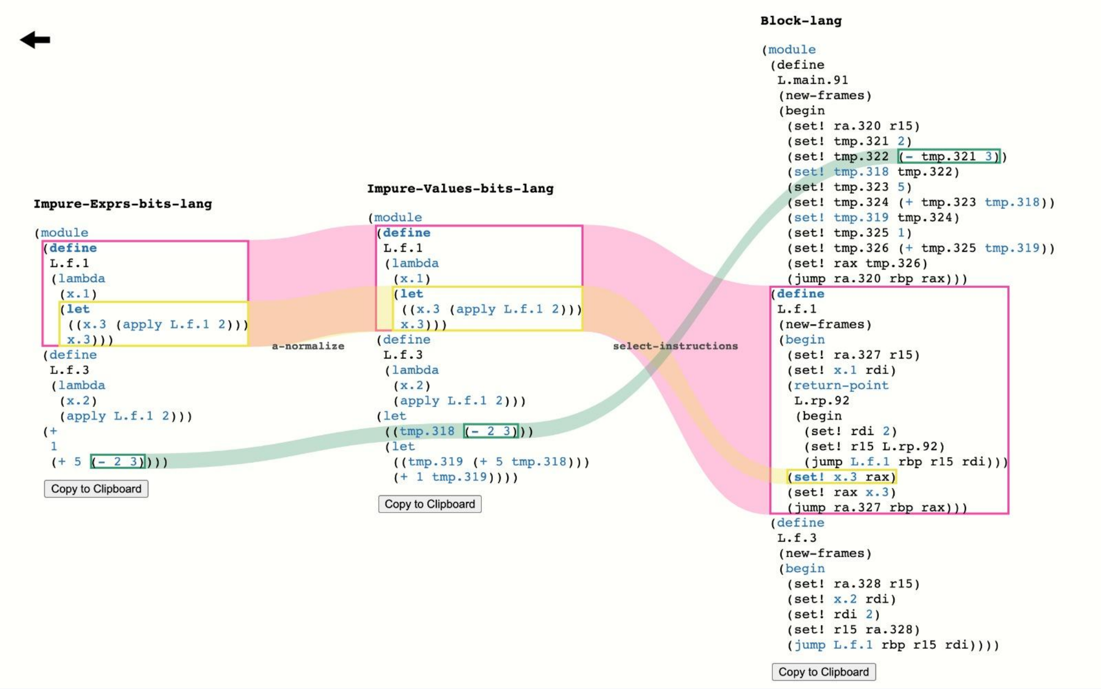
# Visualizing Compiler Passes with LastPass*

what? →



LastPass (enter a test case)

```
(module
  (define L.f.1
    (lambda (x.1) (let ([x.3 (apply L.f.1 2)]) x.3)))
  (define L.f.3
    (lambda (x.2) (apply L.f.1 2)))
  (+ 1 (+ 5 (- 2 3))))
```



Impure-Exprs-bits-lang

```
(module
 (define
  L.f.1
  (lambda
   (x.1)
   (let
    ((x.3 (apply L.f.1 2)))
    x.3)))
 (define
  L.f.3
  (lambda
   (x.2)
   (apply L.f.1 2)))
 (+
  1
  (+ 5 (- 2 3))))
```

Impure-Values-bits-lang

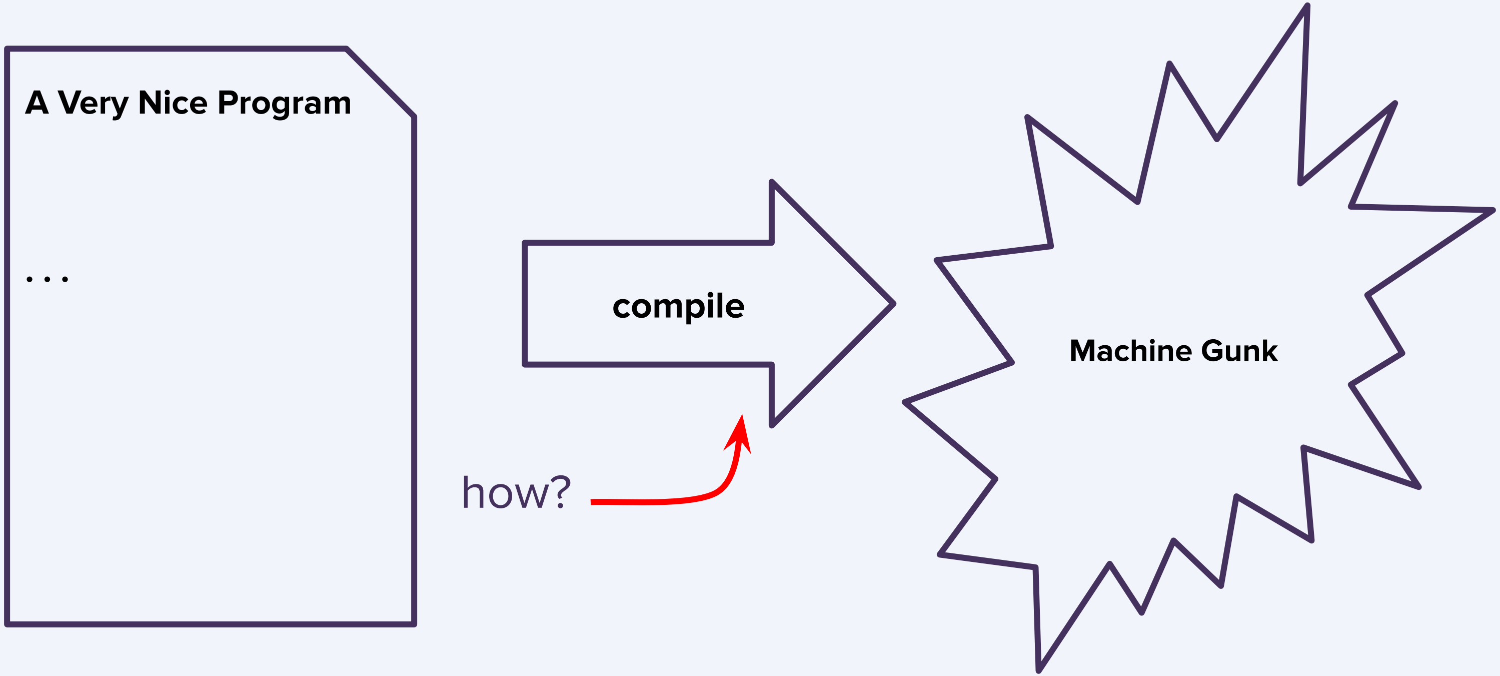
```
(module
 (define
  L.f.1
  (lambda
   (x.1)
   (let
    ((x.3 (apply L.f.1 2)))
    x.3)))
 (define
  L.f.3
  (lambda
   (x.2)
   (apply L.f.1 2)))
 (let
  ((tmp.318 (- 2 3)))
  (let
   ((tmp.319 (+ 5 tmp.318)))
   (+ 1 tmp.319))))
```

Block-lang

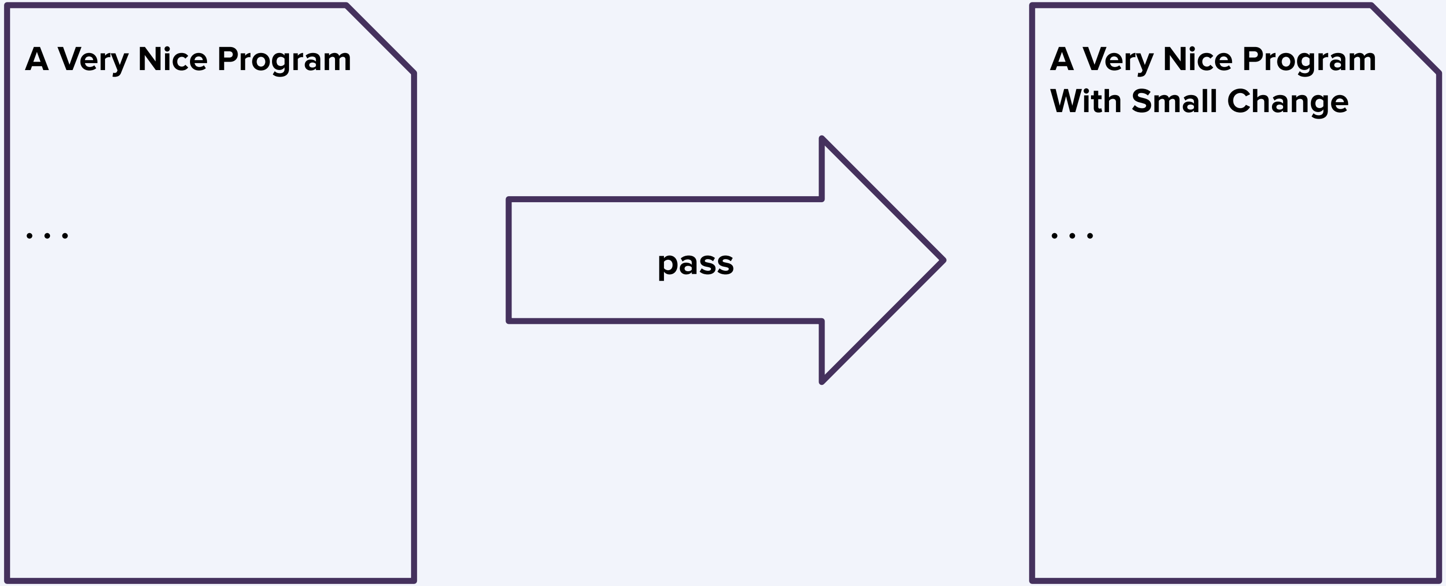
```
(module
 (define
  L.main.91
  (new-frames)
  (begin
   (set! ra.320 r15)
   (set! tmp.321 2)
   (set! tmp.322 (- tmp.321 3))
   (set! tmp.318 tmp.322)
   (set! tmp.323 5)
   (set! tmp.324 (+ tmp.323 tmp.318))
   (set! tmp.319 tmp.324)
   (set! tmp.325 1)
   (set! tmp.326 (+ tmp.325 tmp.319))
   (set! rax tmp.326)
   (jump ra.320 rbp rax)))
 (define
  L.f.1
  (new-frames)
  (begin
   (set! ra.327 r15)
   (set! x.1 rdi)
   (return-point
    L.rp.92
    (begin
     (set! rdi 2)
     (set! r15 L.rp.92)
     (jump L.f.1 rbp r15 rdi)))
   (set! x.3 rax)
   (set! rax x.3)
   (jump ra.327 rbp rax)))
 (define
  L.f.3
  (new-frames)
  (begin
   (set! ra.328 r15)
   (set! x.2 rdi)
   (set! rdi 2)
   (set! r15 ra.328)
   (jump L.f.1 rbp r15 rdi))))
```

* formerly SecondPass[†]
† formerly FirstPass[‡]
‡ formerly Untitled Compiler Pass Visualization

# What are compiler passes?



# What are compiler passes?



**A Very Nice Program**

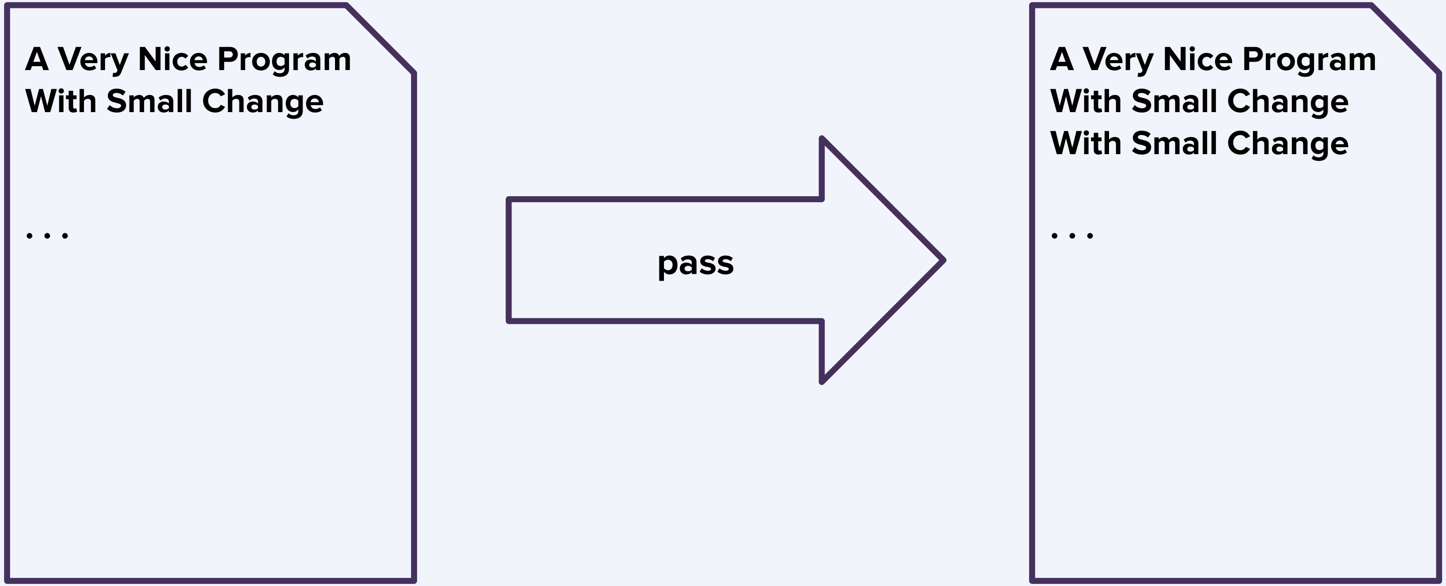
. . .

**pass**

**A Very Nice Program With Small Change**

. . .

# What are compiler passes?



A Very Nice Program
With Small Change

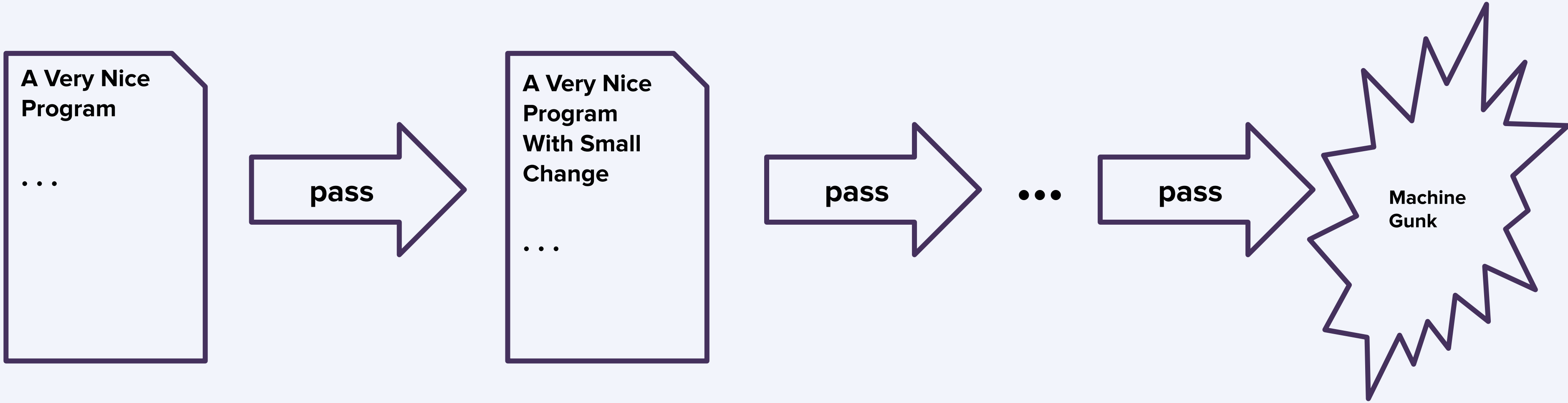
. . .

pass

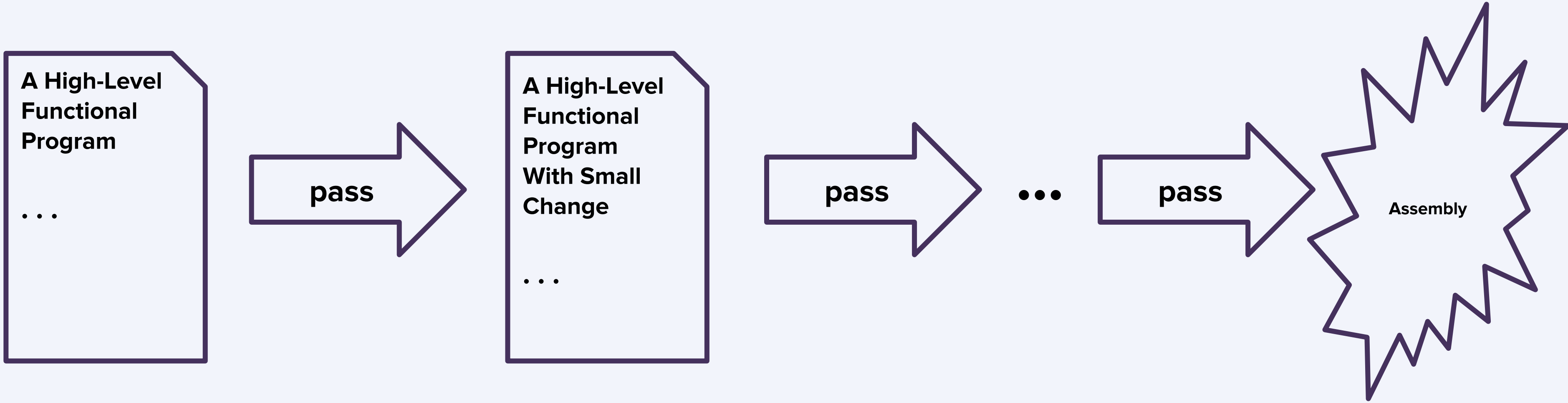
A Very Nice Program
With Small Change
With Small Change

. . .

# Multi-pass compiler

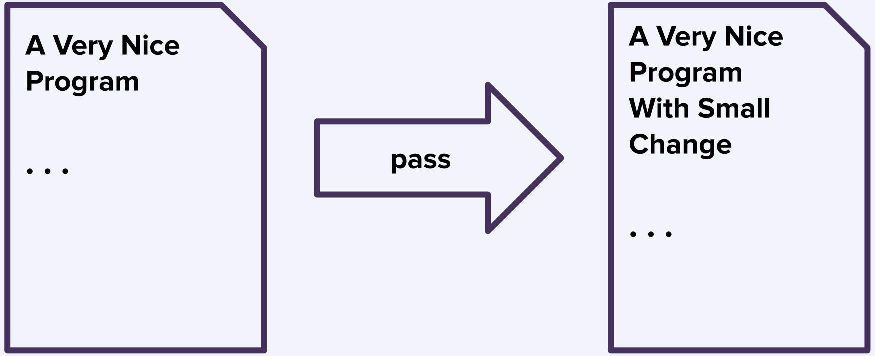


# Goal of CPSC 411


A High-Level Functional Program
. . .
pass
A High-Level Functional Program With Small Change
. . .
pass
•••
pass
Assembly

# Tasks

## understanding an individual pass



A Very Nice Program

. . .

pass

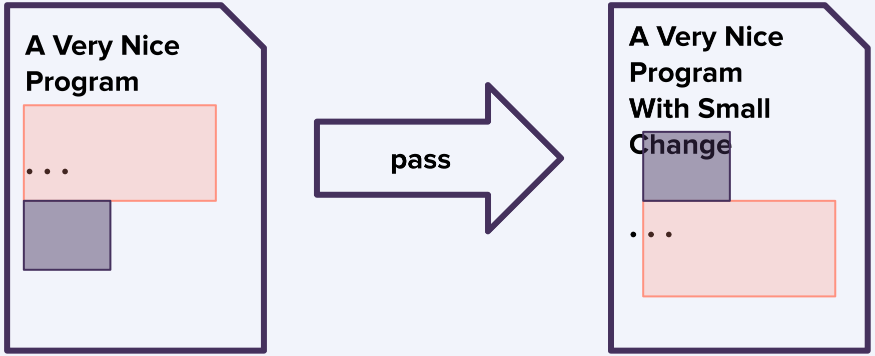
A Very Nice Program With Small Change

. . .

# Tasks

understanding an individual pass

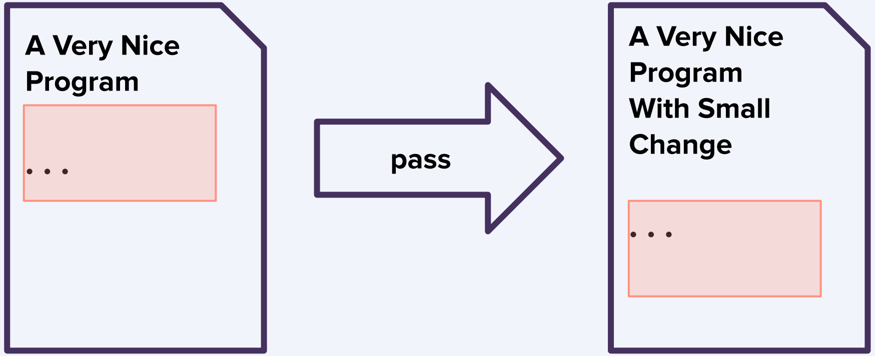
**analyzing its effect on different program expressions**



# Tasks

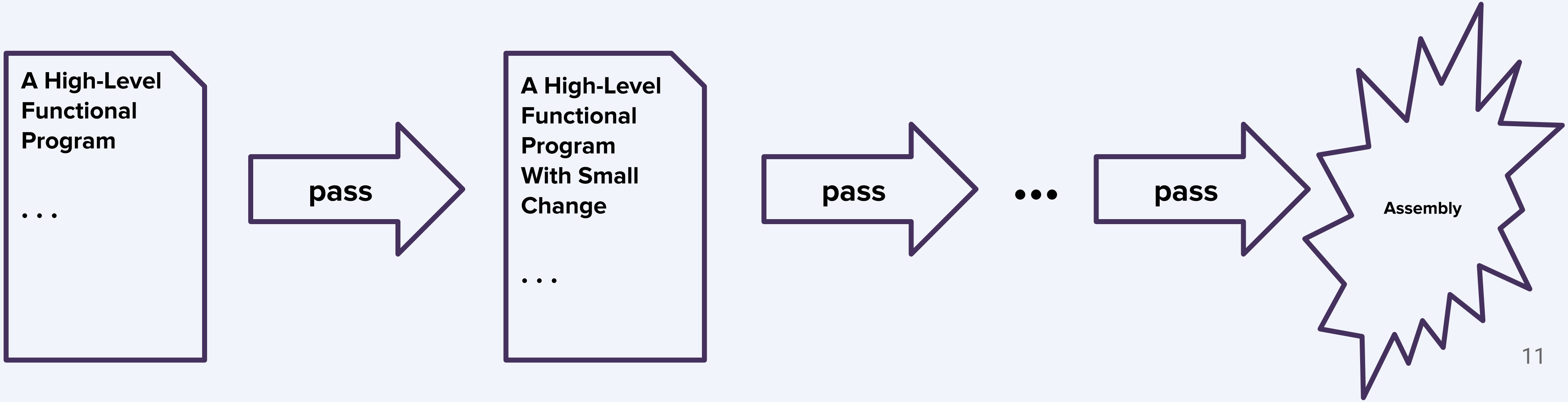
understanding an individual pass

- analyzing its effect on different program expressions
- **identifying compiled expressions in context**



# Tasks

## identifying and comparing complex compiler passes

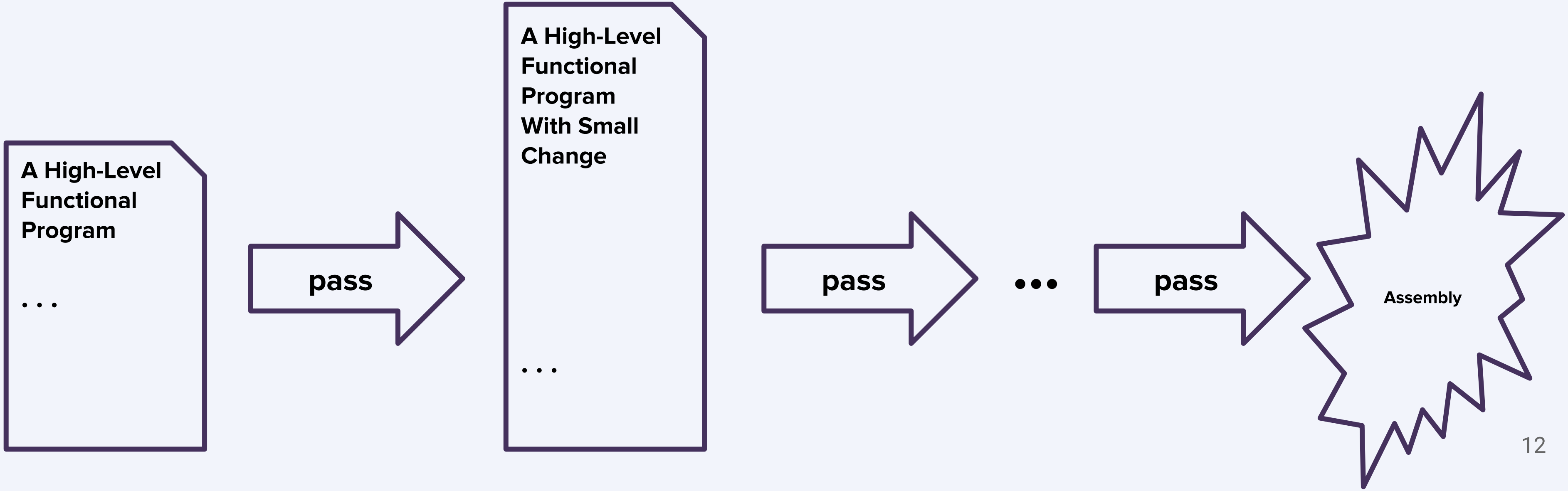


A High-Level Functional Program
. . .
pass
A High-Level Functional Program With Small Change
. . .
pass
•••
pass
Assembly

# Tasks

identifying and comparing complex compiler passes

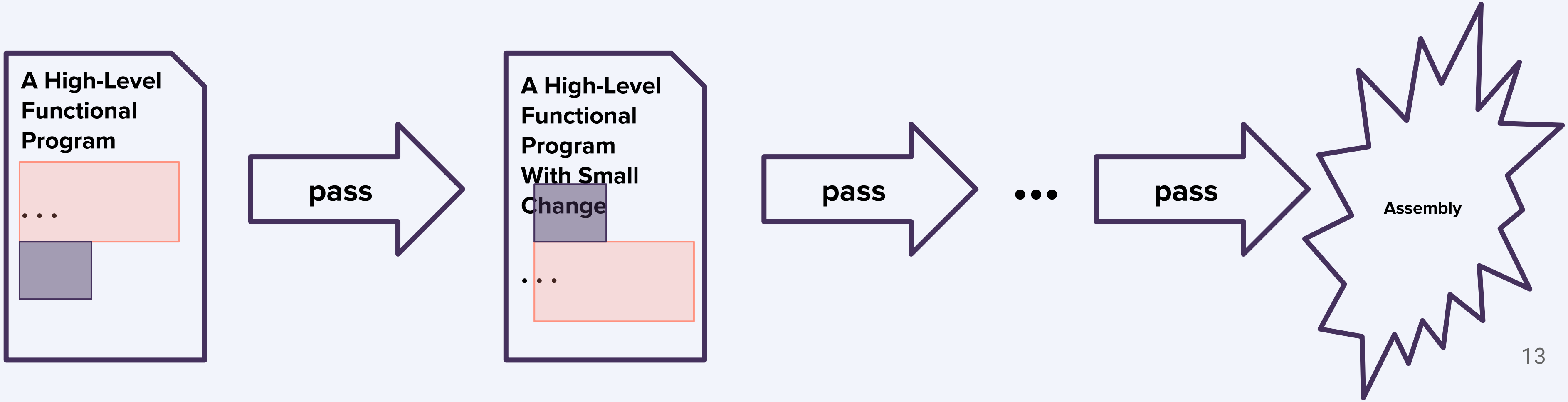
**comparing amount of generated code**


A High-Level Functional Program
. . .
pass
A High-Level Functional Program With Small Change
. . .
pass
•••
pass
Assembly

# Tasks

identifying and comparing complex compiler passes

- comparing amount of generated code
- **identifying structural changes**

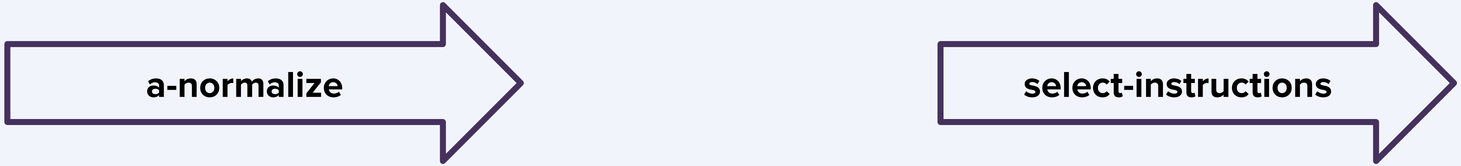


A High-Level Functional Program
. . .
pass
A High-Level Functional Program With Small Change
. . .
pass
•••
pass
Assembly

# LastPass Scope

CPSC 411 reference compiler: 33 passes!
LastPass: 2 passes

# LastPass Scope



a-normalize

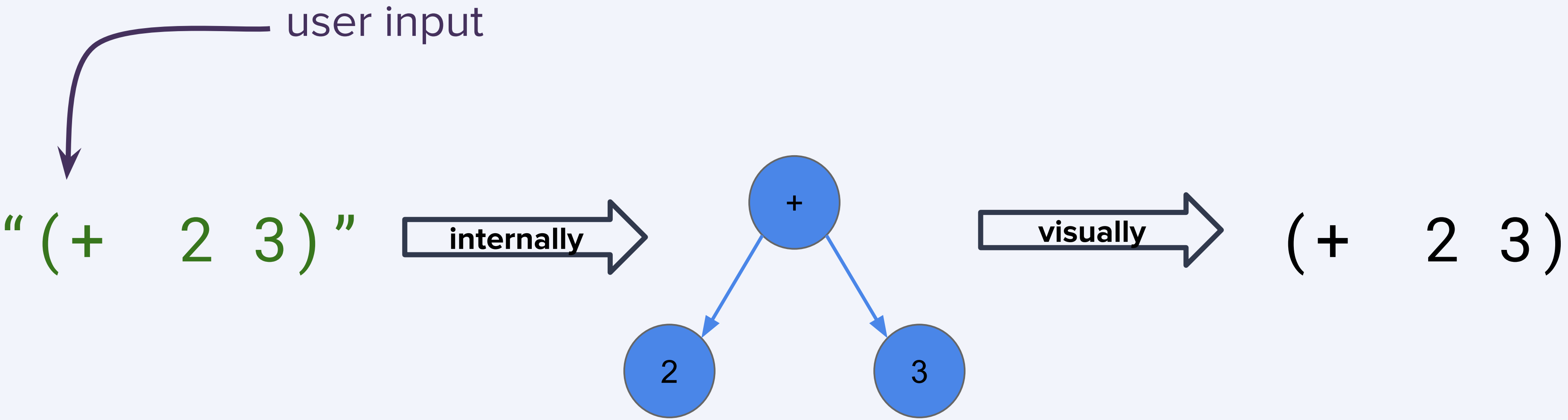
select-instructions

```
(* (+ 2 3) 5)
```

```
(let (tmp (+ 2 3))
  (* tmp 5))
```

```
(set! tmp.99 2)
(set! tmp.100 (+ tmp.99 3))
(set! tmp tmp.100)
(set! tmp.101 (* tmp 5))
(set! rax tmp.101)
(jump ra.98 rbp rax)
```

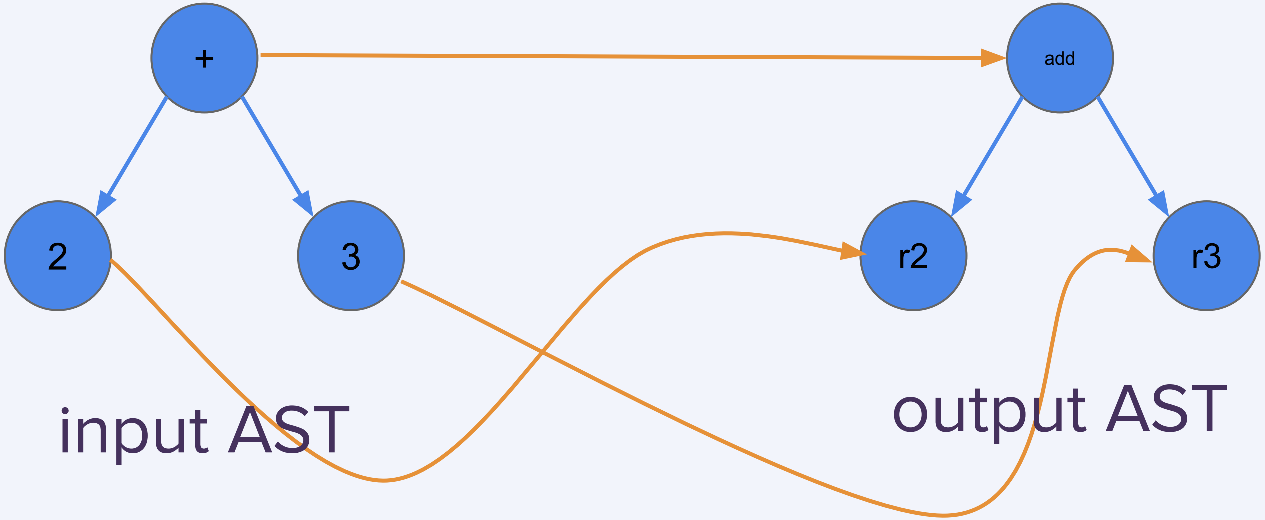
# Data: Programs



## Abstract Syntax Tree (AST)

directed acyclic graph

# Data: Pass



# Data: Summary

**3 program ASTs**

- source program AST
- output AST from `a-normalize`
- output AST from `select-instructions`

**2 sets of directed edges**

- `a-normalize` and `select-instructions`

**exactly what we want to visualize!**

# Limitations

**false positives and negatives on complex passes**

`a-normalize` doesn’t move a lot of code, but requires
a new style of programming to implement

**scale, 33 >> 2**

**pinning overload**

pinning too many flows blends colours

# LastPass: A tool for CPSC 411

**query multiple test programs**

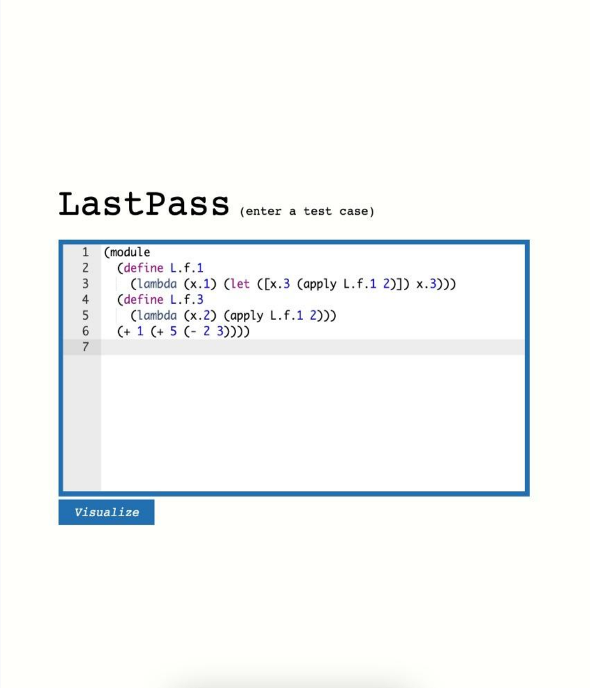
**see an overview of compilation changes**

**compare programs after any number of passes**

**track how expressions change in context**

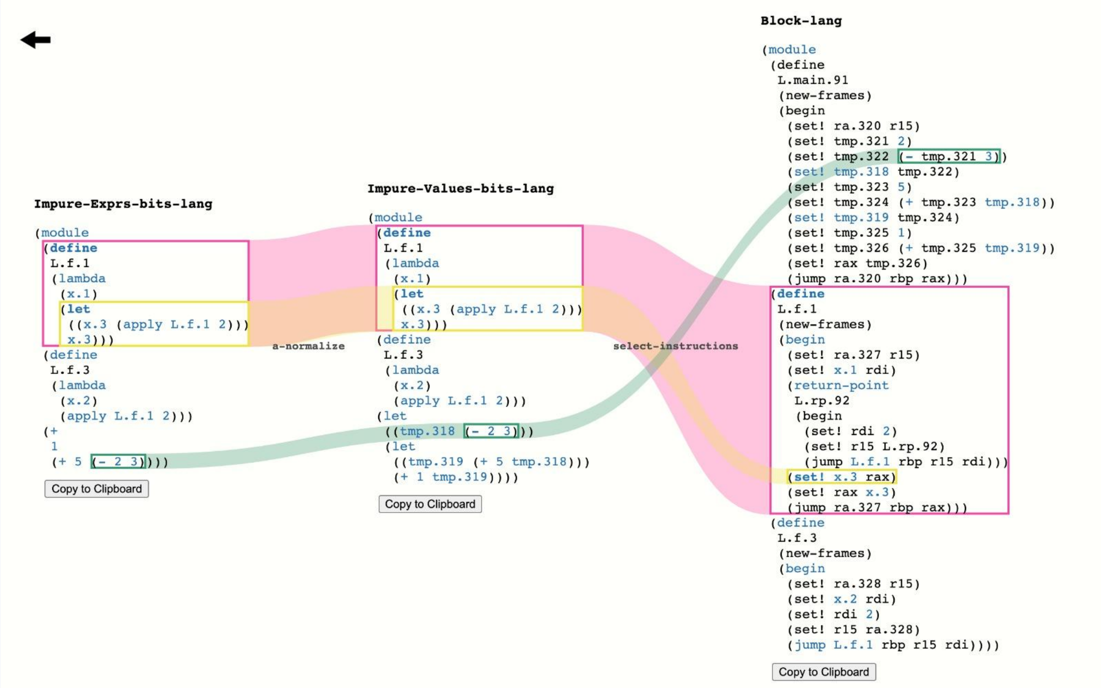
# Thank you!

https://se.cs.ubc.ca/compiler-viz/index.html



**LastPass** (enter a test case)

```
(module
  (define L.f.1
    (lambda (x.1) (let ([x.3 (apply L.f.1 2)]) x.3)))
  (define L.f.3
    (lambda (x.2) (apply L.f.1 2)))
  (+ 1 (+ 5 (- 2 3))))
```



**Impure-Exprs-bits-lang**

```
(module
 (define
  L.f.1
  (lambda
   (x.1)
   (let
    ((x.3 (apply L.f.1 2)))
    x.3)))
 (define
  L.f.3
  (lambda
   (x.2)
   (apply L.f.1 2)))
 (+
  1
  (+ 5 (- 2 3))))
```

**Impure-Values-bits-lang**

```
(module
 (define
  L.f.1
  (lambda
   (x.1)
   (let
    ((x.3 (apply L.f.1 2)))
    x.3)))
 (define
  L.f.3
  (lambda
   (x.2)
   (apply L.f.1 2)))
 (let
  ((tmp.318 (- 2 3)))
  (let
   ((tmp.319 (+ 5 tmp.318)))
   (+ 1 tmp.319))))
```

**Block-lang**

```
(module
 (define
  L.main.91
  (new-frames)
  (begin
   (set! ra.320 r15)
   (set! tmp.321 2)
   (set! tmp.322 (- tmp.321 3))
   (set! tmp.318 tmp.322)
   (set! tmp.323 5)
   (set! tmp.324 (+ tmp.323 tmp.318))
   (set! tmp.319 tmp.324)
   (set! tmp.325 1)
   (set! tmp.326 (+ tmp.325 tmp.319))
   (set! rax tmp.326)
   (jump ra.320 rbp rax)))
 (define
  L.f.1
  (new-frames)
  (begin
   (set! ra.327 r15)
   (set! x.1 rdi)
   (return-point
    L.rp.92
    (begin
     (set! rdi 2)
     (set! r15 L.rp.92)
     (jump L.f.1 rbp r15 rdi)))
   (set! x.3 rax)
   (set! rax x.3)
   (jump ra.327 rbp rax)))
 (define
  L.f.3
  (new-frames)
  (begin
   (set! ra.328 r15)
   (set! x.2 rdi)
   (set! rdi 2)
   (set! r15 ra.328)
   (jump L.f.1 rbp r15 rdi))))
```