INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER DYNAMIC LANGUAGES WITHOUT STATIC ANALYSIS

ECOOP
June 22, 2017

Baptiste Saleil
baptiste.saleil@umontreal.ca
Marc Feeley
feeley@iro.umontreal.ca

Université de Montréal

INTRODUCTION

- Research on JIT compilation
  - Dynamic languages
  - Dynamic techniques

- *LC*: Research oriented Scheme compiler
  - Scheme and Functional Programming Workshop 2014 & 2015
  - https://github.com/bsaleil/lc

INTRODUCTION

- Dynamic languages
  - Work done at compilation ↓
  - Work done at execution ↑
- Dynamic type checking
  - Ensures safety of the primitives :)
  - Impact on performance :(

DYNAMIC TYPE CHECKING

- Example

```
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))
```

DYNAMIC TYPE CHECKING

- Example

```
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))
```

4 type checks in this code

DYNAMIC TYPE CHECKING

- Example

```
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))
```

4 type checks in this code

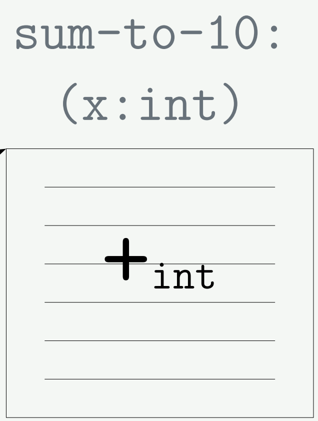
- How can we remove them ?

INTERPROCEDURAL SPECIALIZATION

```
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))


(print (sum-to-10 6))
(print (sum-to-10 7.5))
```

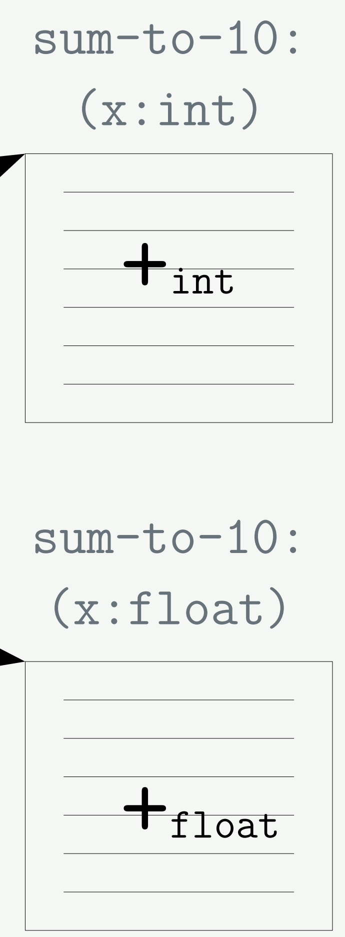
INTERPROCEDURAL SPECIALIZATION

```
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))


(print (sum-to-10 6))
(print (sum-to-10 7.5))
```



INTERPROCEDURAL SPECIALIZATION

```
(define (sum-to-10 x)
  (if (> x 10)
      0
      (+ x (sum-to-10 (+ x 1)))))


(print (sum-to-10 6))
(print (sum-to-10 7.5))
```



RUNNING EXAMPLE: HIGHER ORDER FUNCTION

```
(define (make-sumer n)
  (letrec ((f (lambda (x)      ← Closure that captures n
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

RUNNING EXAMPLE: HIGHER ORDER FUNCTION

```
(define (make-sumer n)
  (letrec ((f (lambda (x)       ← Closure that captures n
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

RUNNING EXAMPLE: HIGHER ORDER FUNCTION

→ Static analysis (e.g. *0*-CFA)

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

Running Example: Higher Order Function

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

→ Static analysis (e.g. *0*-CFA)

- JIT incompatible
- Lacks precision

RUNNING EXAMPLE: HIGHER ORDER FUNCTION

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
               (if (> x n)
                   0
                   (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

→ ~~Static analysis (e.g. 0-CFA)~~
- ~~JIT incompatible~~
- ~~Lacks precision~~

INTERPROCEDURAL SPECIALIZATION

INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER LANGUAGES

INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER DYNAMIC LANGUAGES

INTERPROCEDURAL SPECIALIZATION OF HIGHER-ORDER DYNAMIC LANGUAGES WITHOUT STATIC ANALYSIS

Interprocedural Specialization of Higher-Order Dynamic Languages Without Static Analysis

→ Basic Block Versioning (BBV)

- *Simple and Effective Type Check Removal through Lazy Basic Block Versioning*
  Maxime Chevalier-Boisvert and Marc Feeley, ECOOP 2015
- Lazy intraprocedural code specialization (JIT)
- No static analysis
- Dynamic languages (JavaScript, Scheme, …)

NAIVE COMPILATION

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

NAIVE COMPILATION

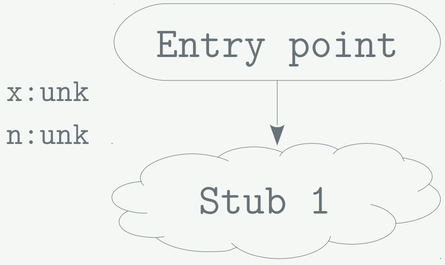
```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))
```

5 checks needed

```
(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))


; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

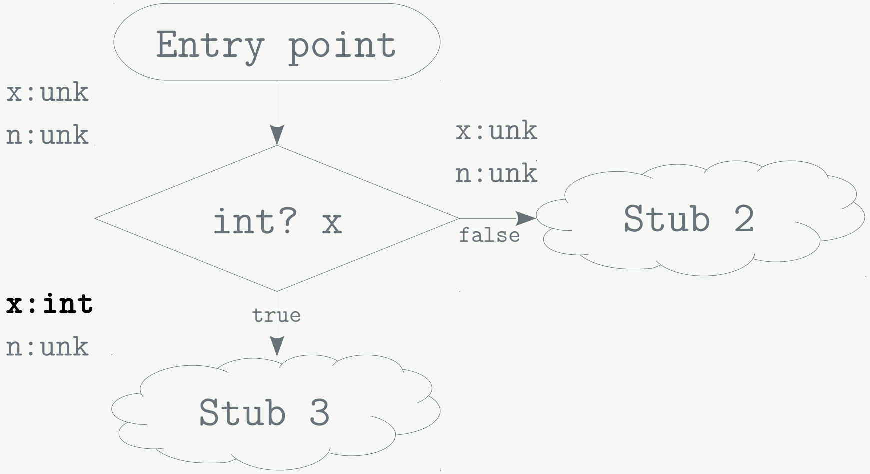
BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



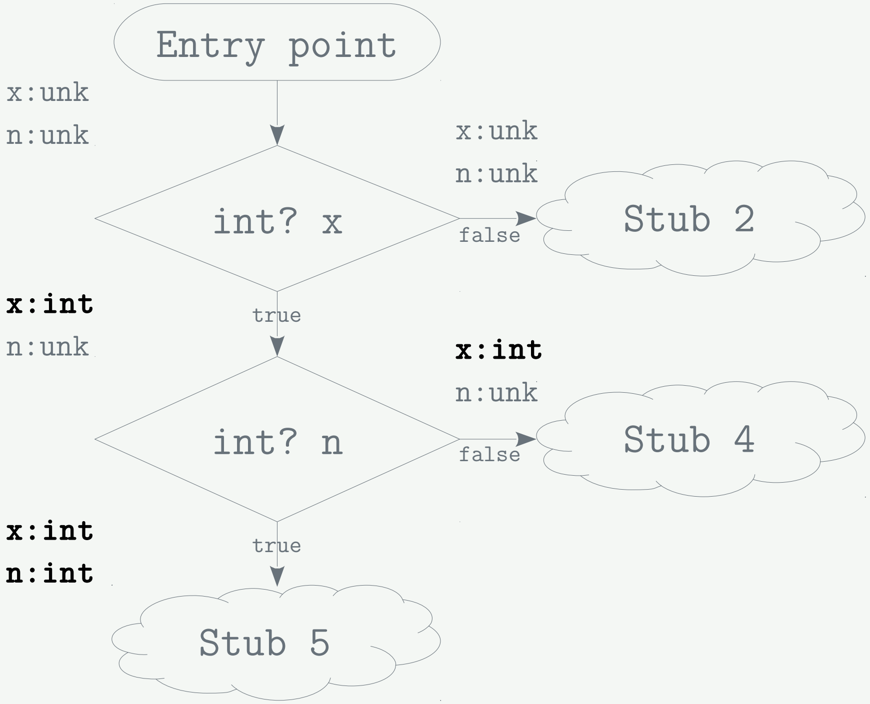
BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



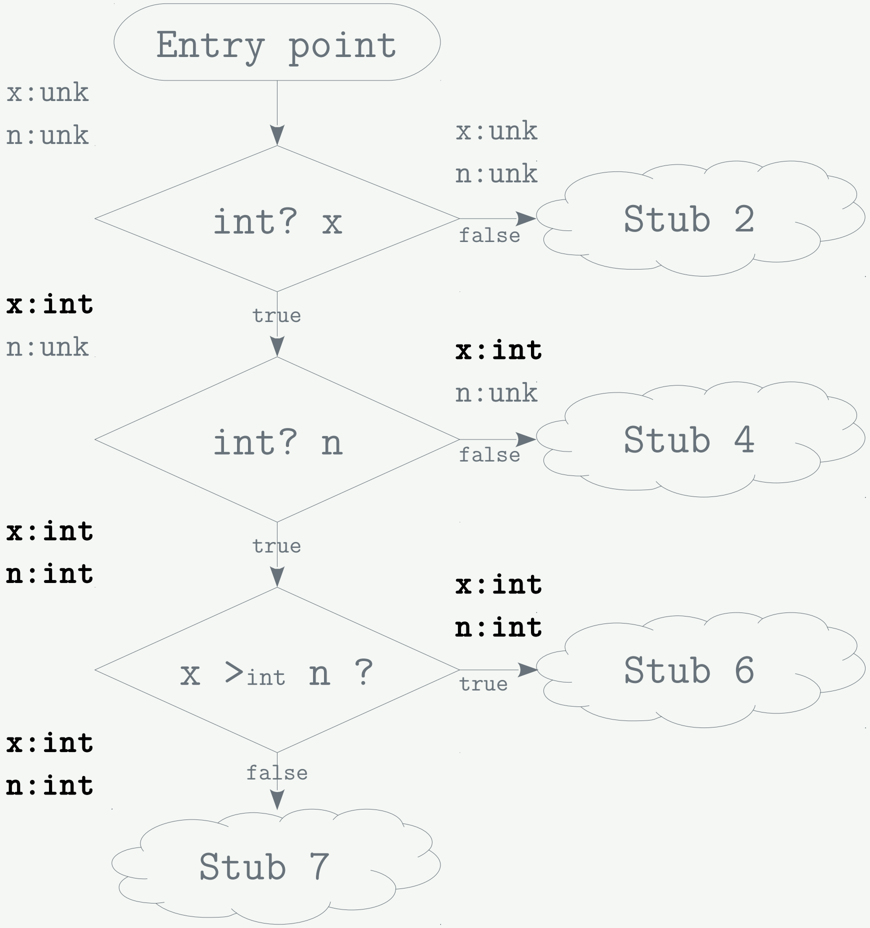
BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



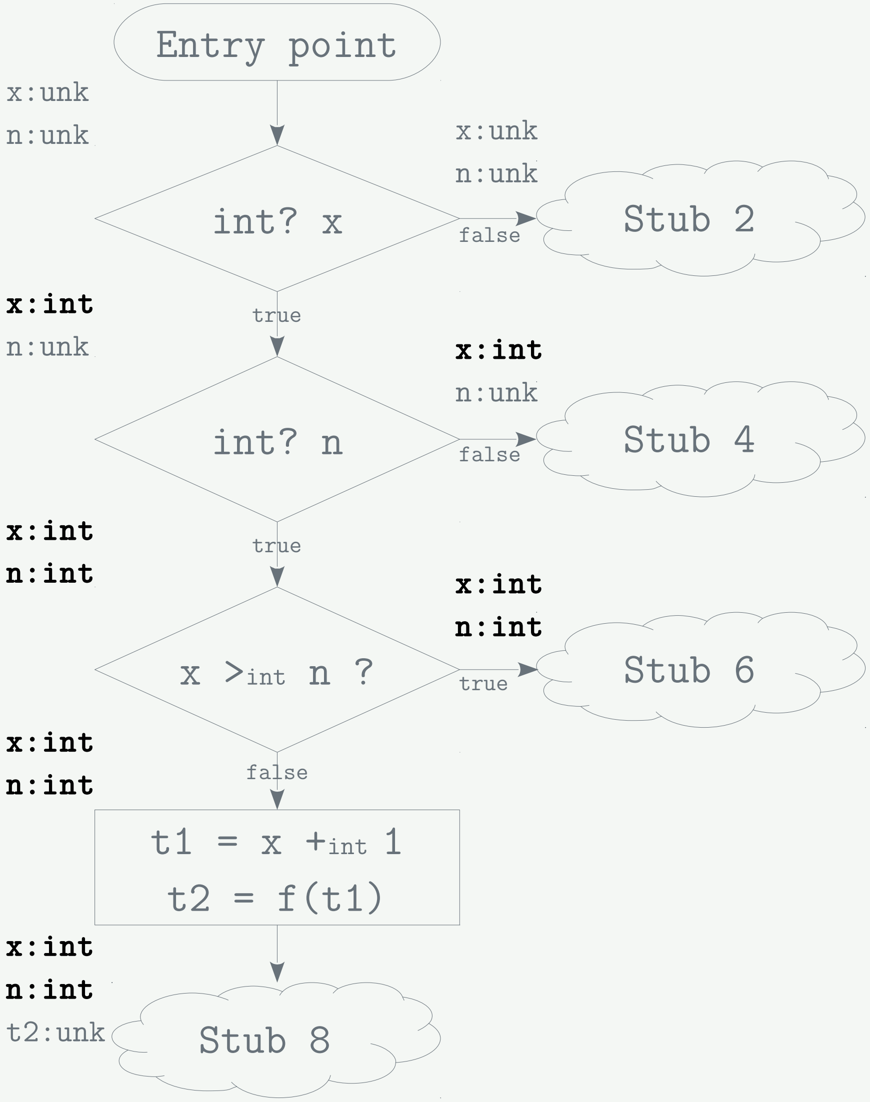
BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



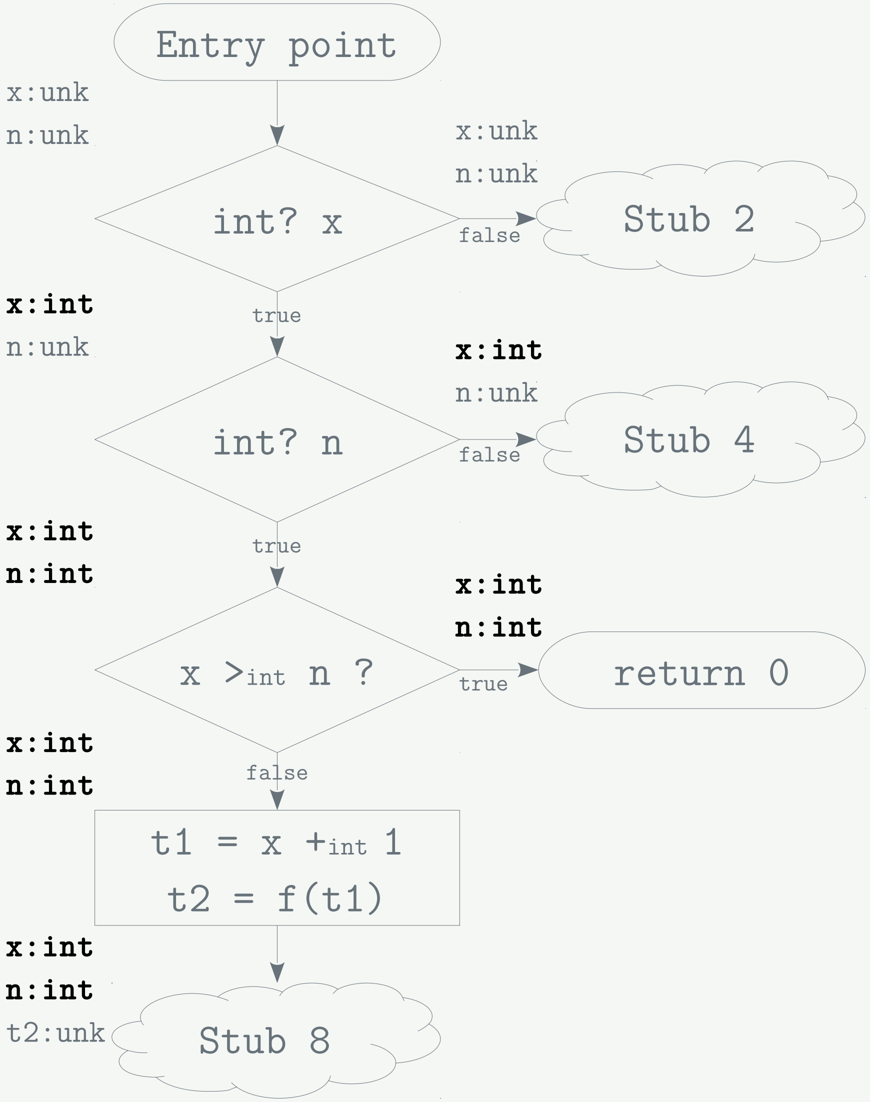
BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



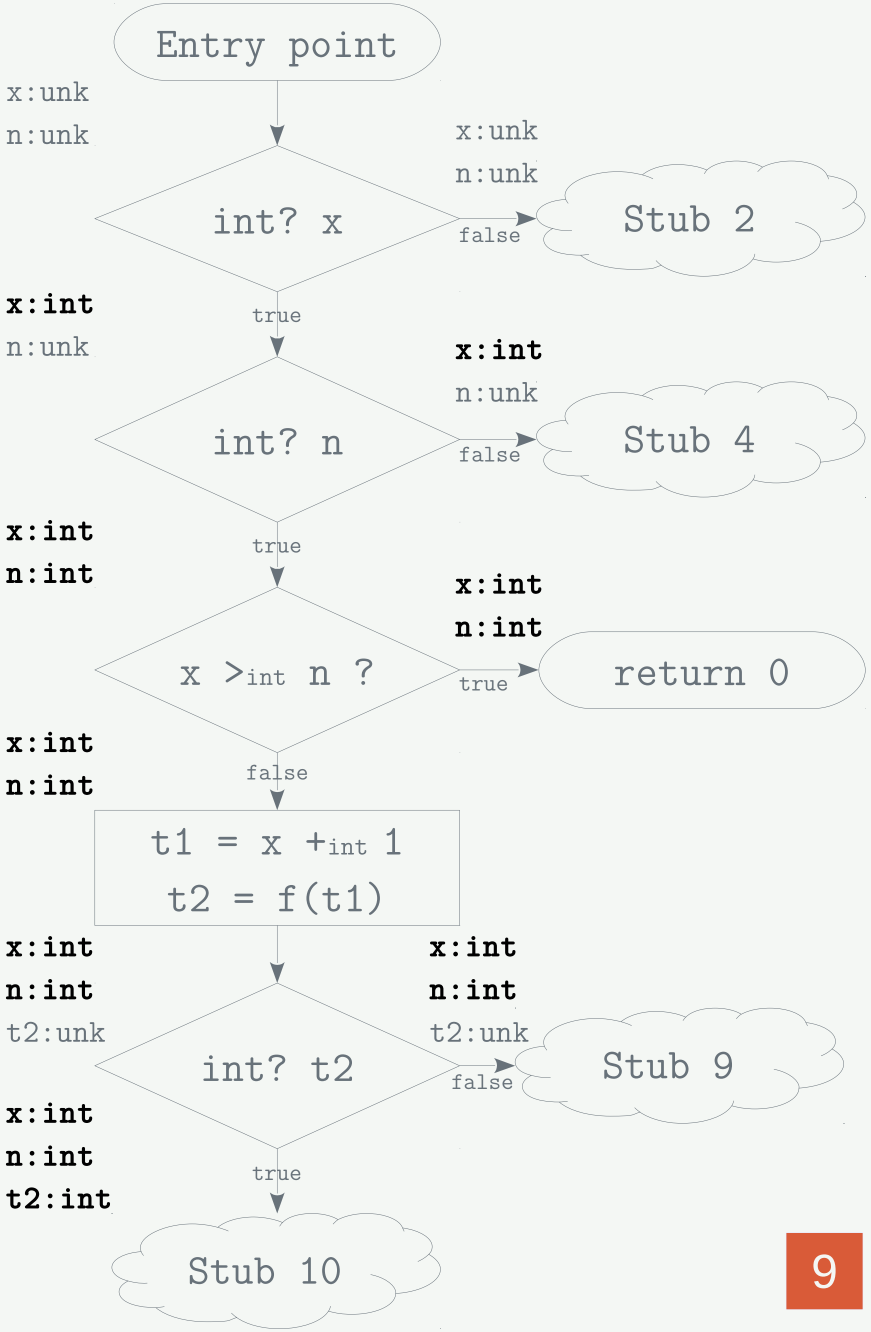
BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



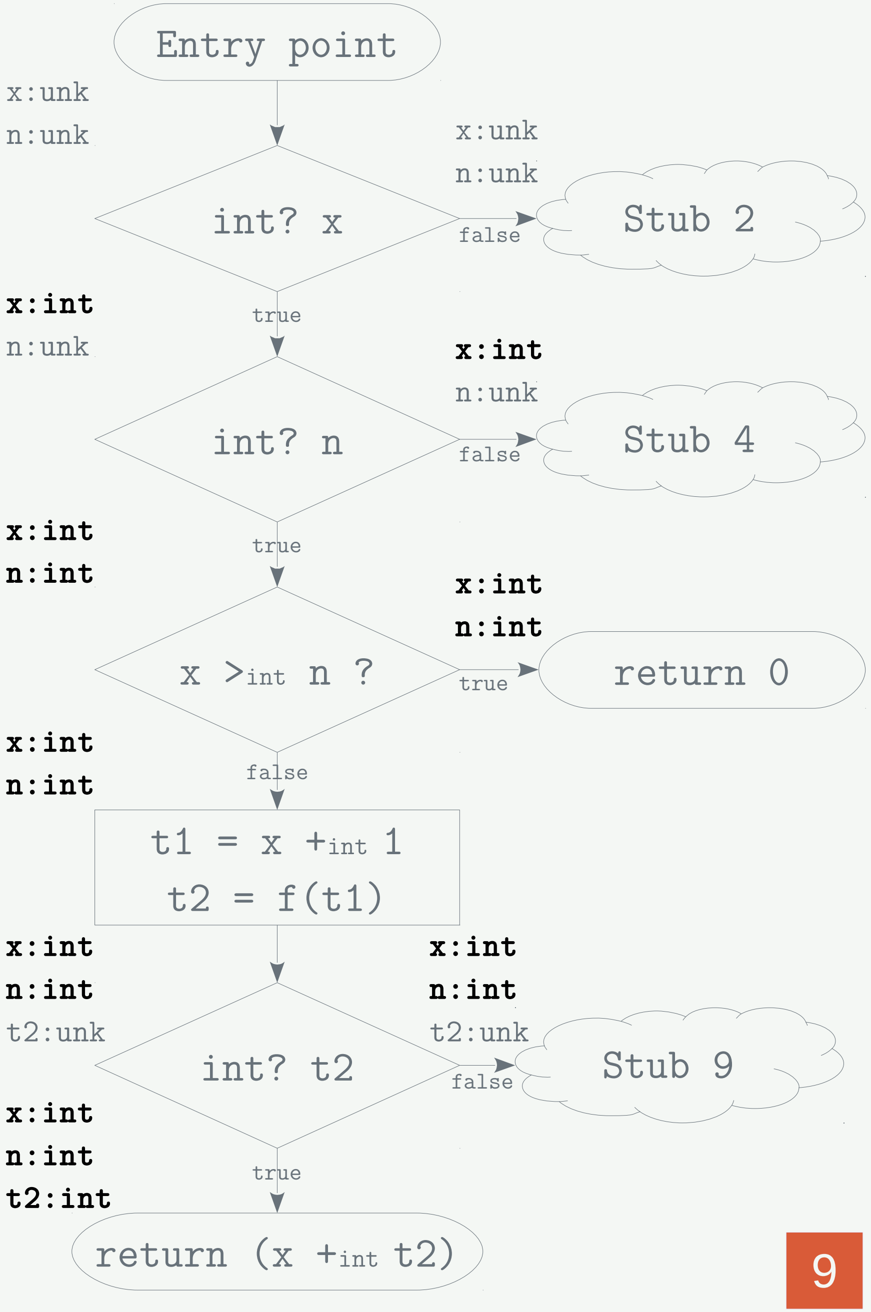
BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

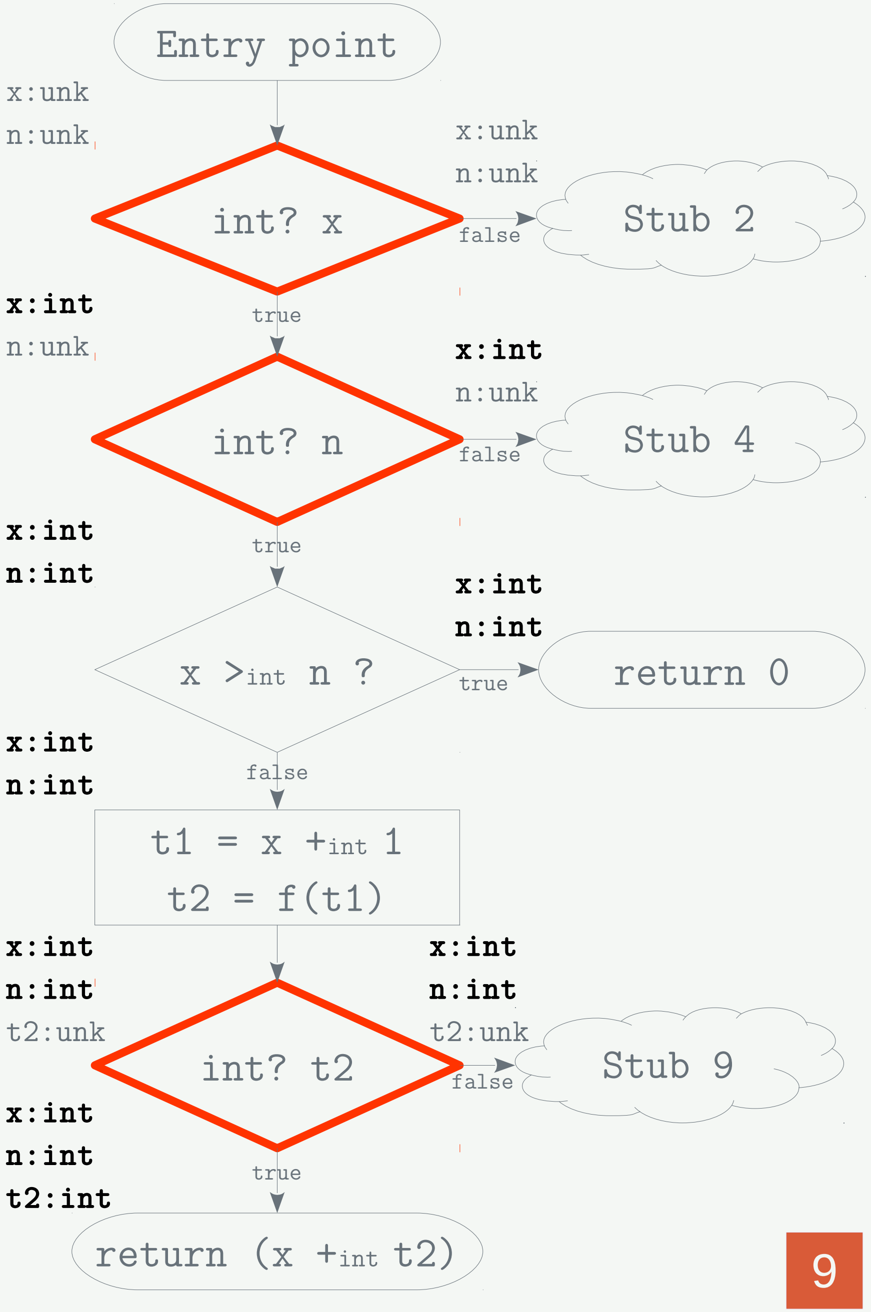


BASIC BLOCK VERSIONING

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))
```

3 checks needed
(2 removed)

```
(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```



OUR WORK: INTERPROCEDURAL EXTENSIONS

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))
```

2 checks needed
(3 removed)

```
(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))


; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

- Propagate the types through function calls
  → no check on x

OUR WORK: INTERPROCEDURAL EXTENSIONS

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))
```

1 check needed
(4 removed)

```
(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

- Propagate the types through function calls
  → no check on x
- Propagate the types through function returns
  → no check on the returned value

OUR WORK: INTERPROCEDURAL EXTENSIONS

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

no checks needed
(5 removed)

- Propagate the types through function calls
  → no check on x
- Propagate the types through function returns
  → no check on the returned value
- Specialize the code using captured information
  → no check on n

OUR WORK: INTERPROCEDURAL EXTENSIONS

→ Several entry points, each specialized for a type combination

OUR WORK: INTERPROCEDURAL EXTENSIONS

→ Several entry points, each specialized for a type combination

1. Extend the closure representation

- Allow storing multiple entry points instead of 1

OUR WORK: INTERPROCEDURAL EXTENSIONS

→ Several entry points, each specialized for a type combination

1. Extend the closure representation
   - Allow storing multiple entry points instead of 1
2. Dynamic dispatch to jump to the entry point
   - Each function call
   - Each function return (each continuation call)

OUR WORK: INTERPROCEDURAL EXTENSIONS

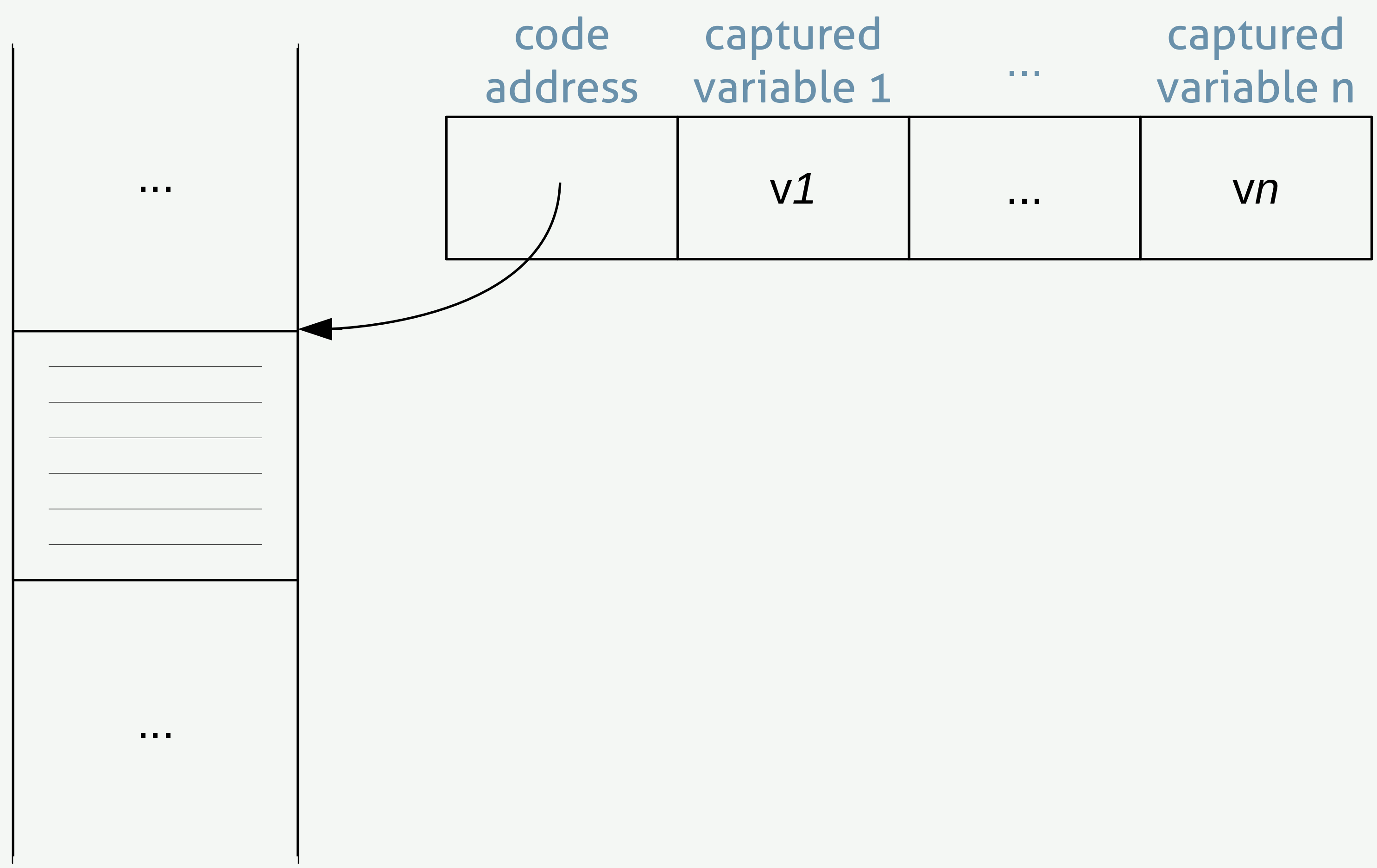
→ Several entry points, each specialized for a type combination

1. Extend the closure representation
   - Allow storing multiple entry points instead of 1
2. Dynamic dispatch to jump to the entry point
   - Each function call
   - Each function return (each continuation call)
3. Specialize using captured information

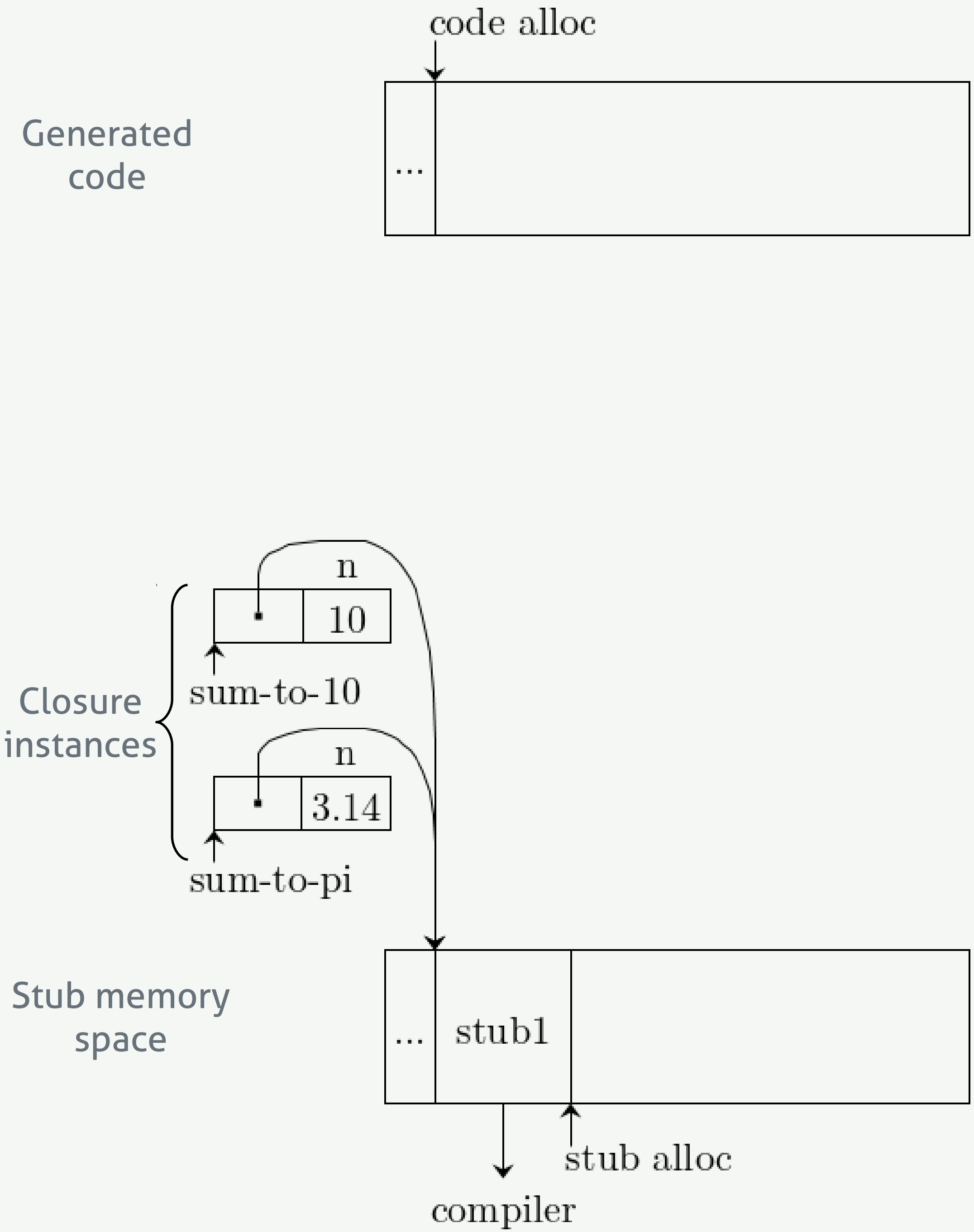
IMPLEMENTATION

Closure representation extension

FLAT CLOSURE REPRESENTATION

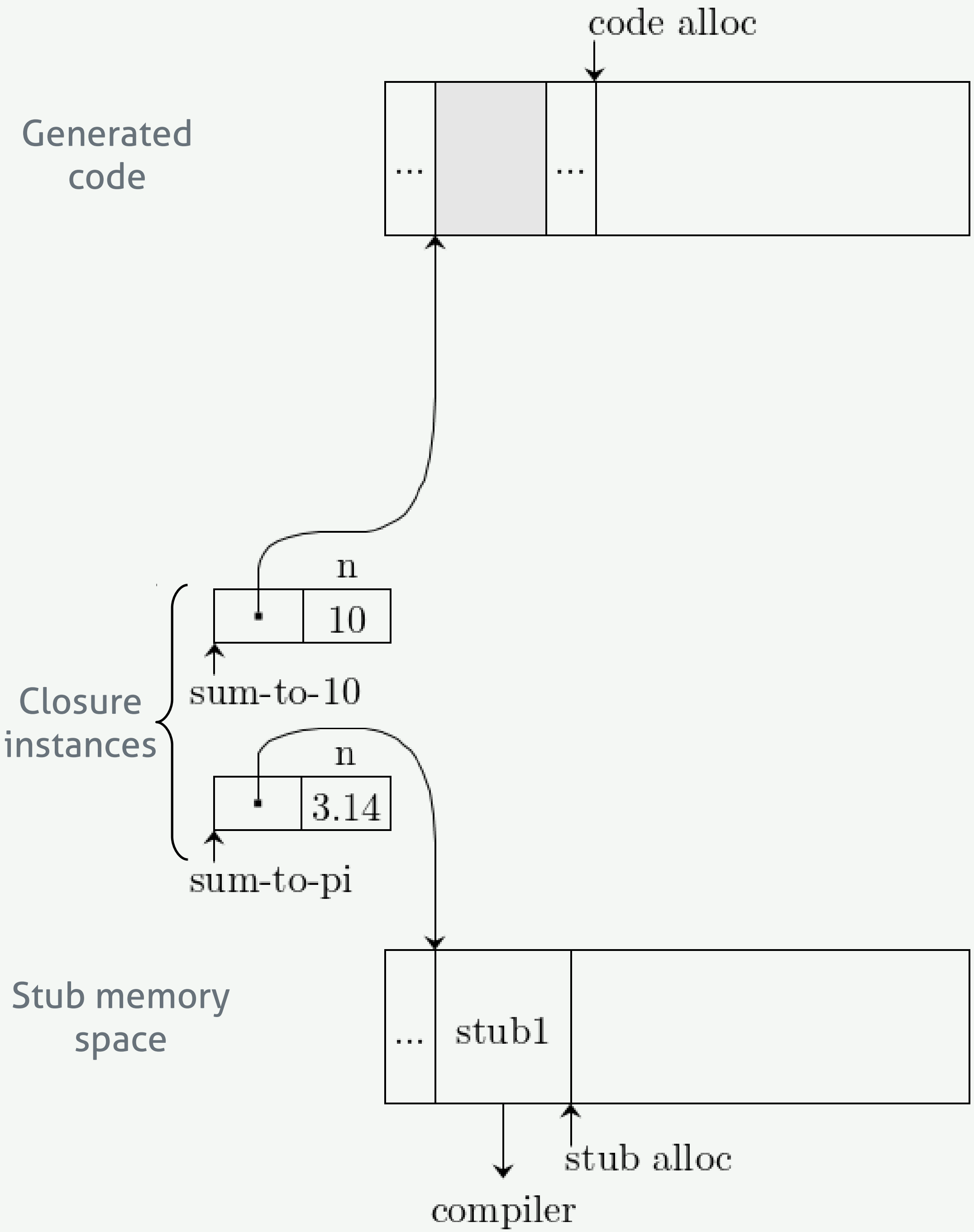


FLAT CLOSURE REPRESENTATION



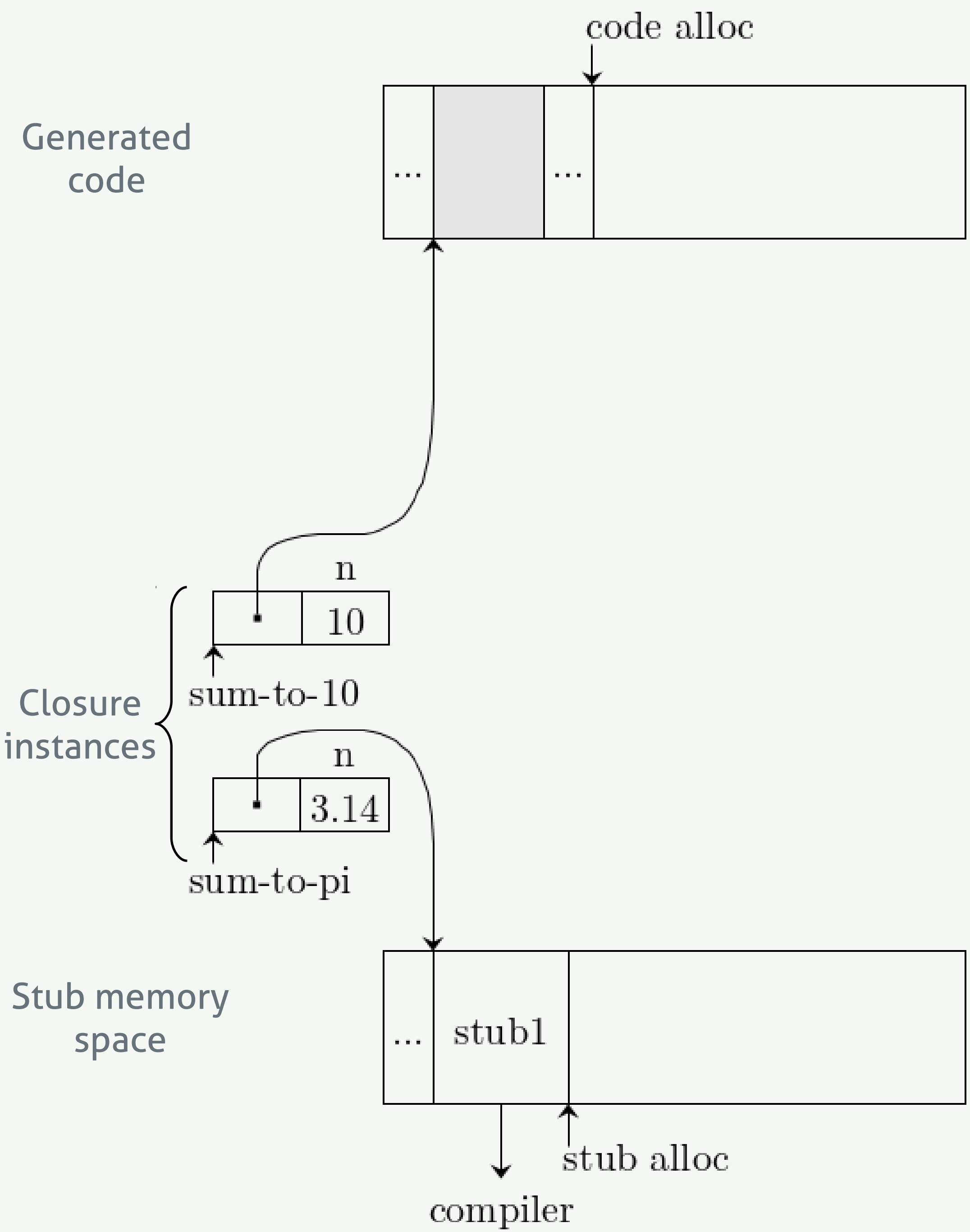

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

FLAT CLOSURE REPRESENTATION



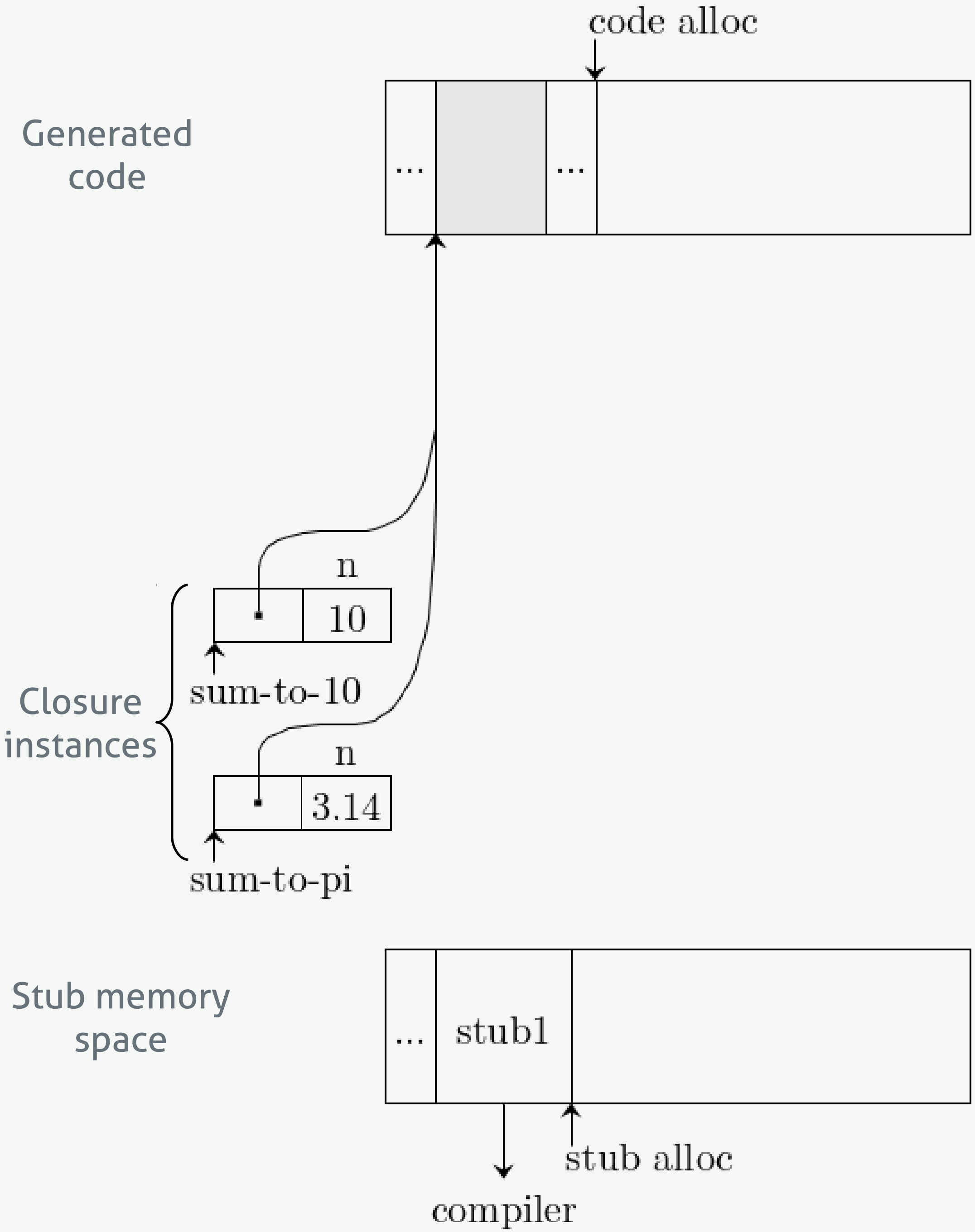

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

FLAT CLOSURE REPRESENTATION



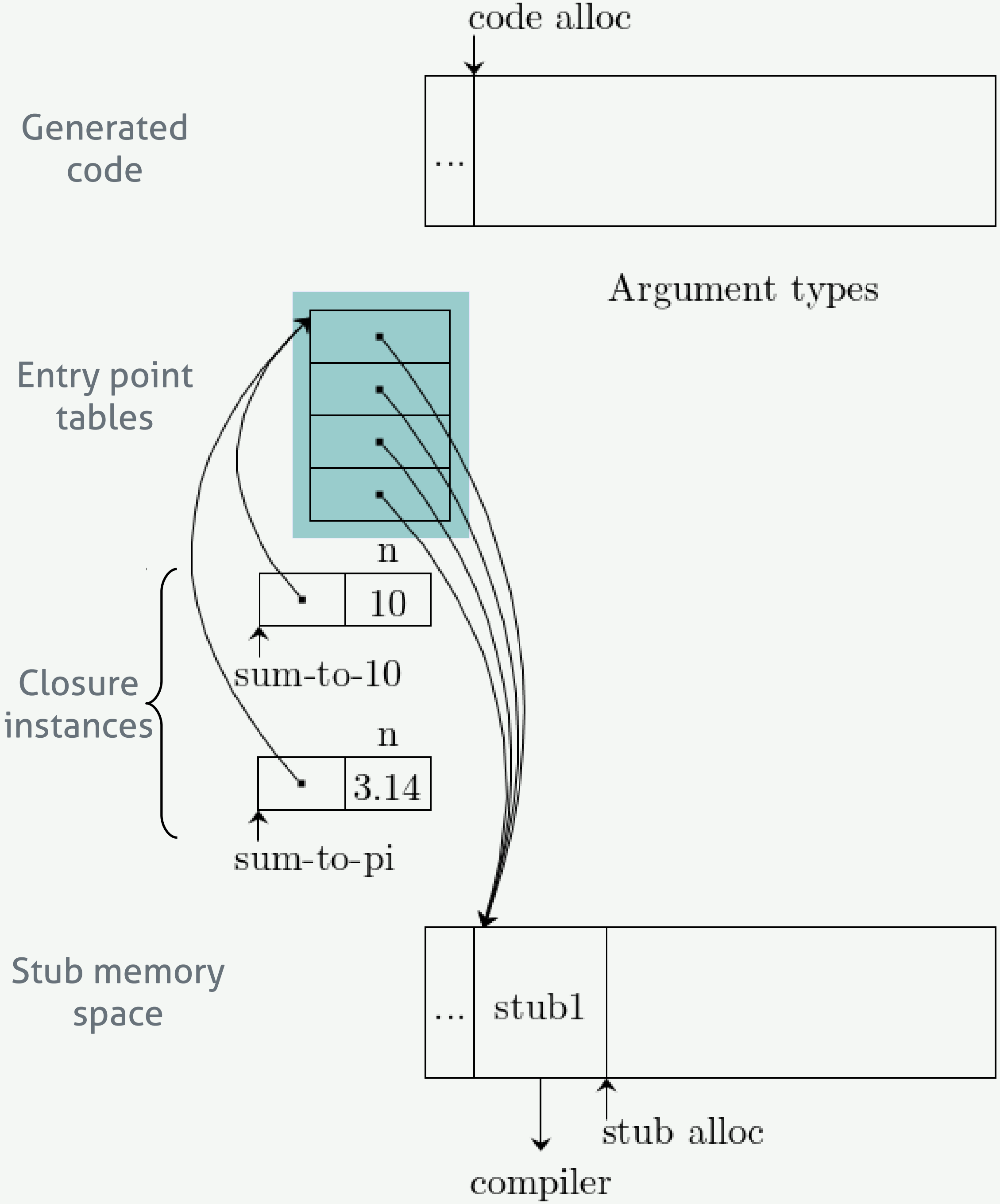

```scheme
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

FLAT CLOSURE REPRESENTATION



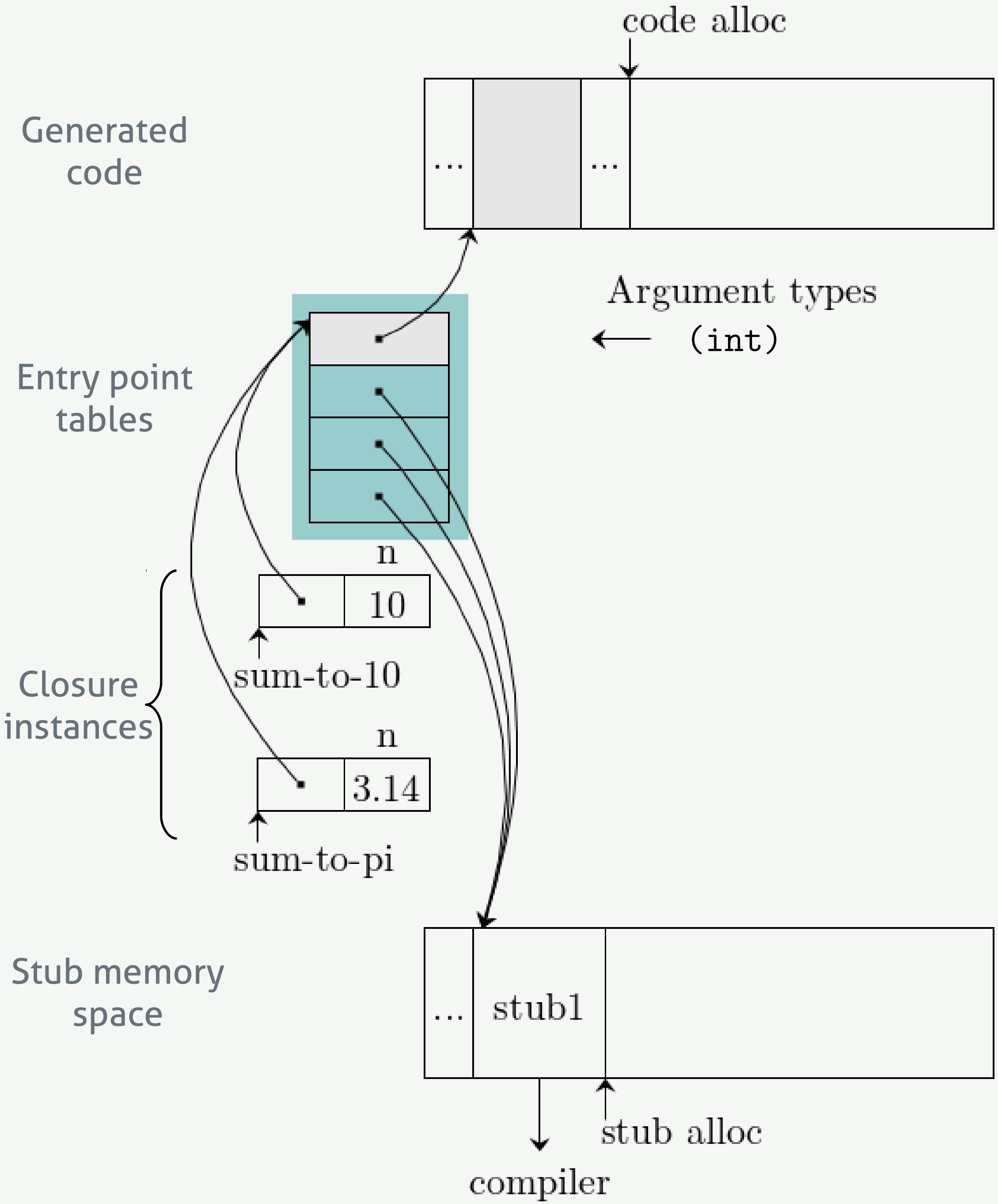

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: FLAT CLOSURE EXTENSION



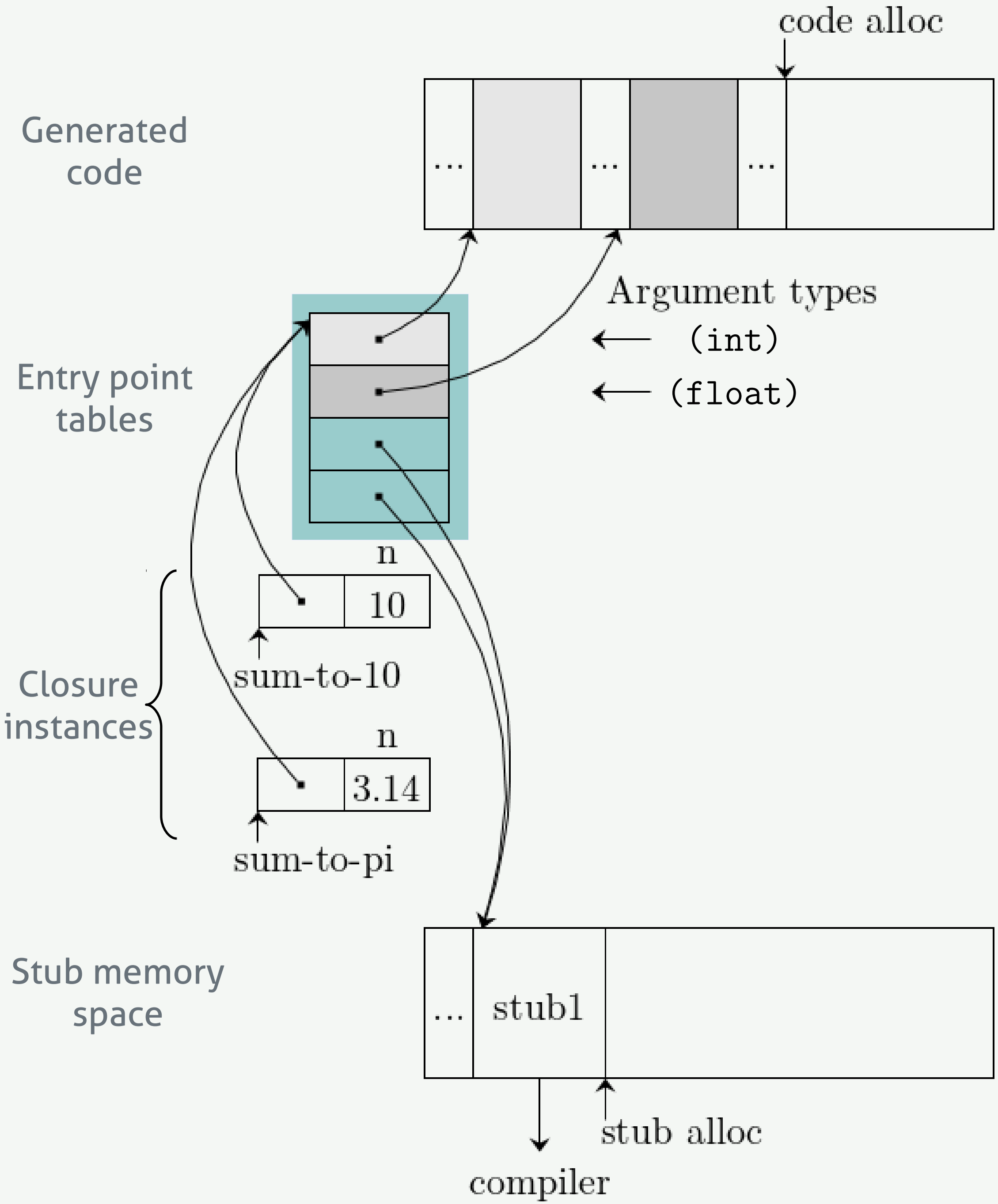

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: FLAT CLOSURE EXTENSION



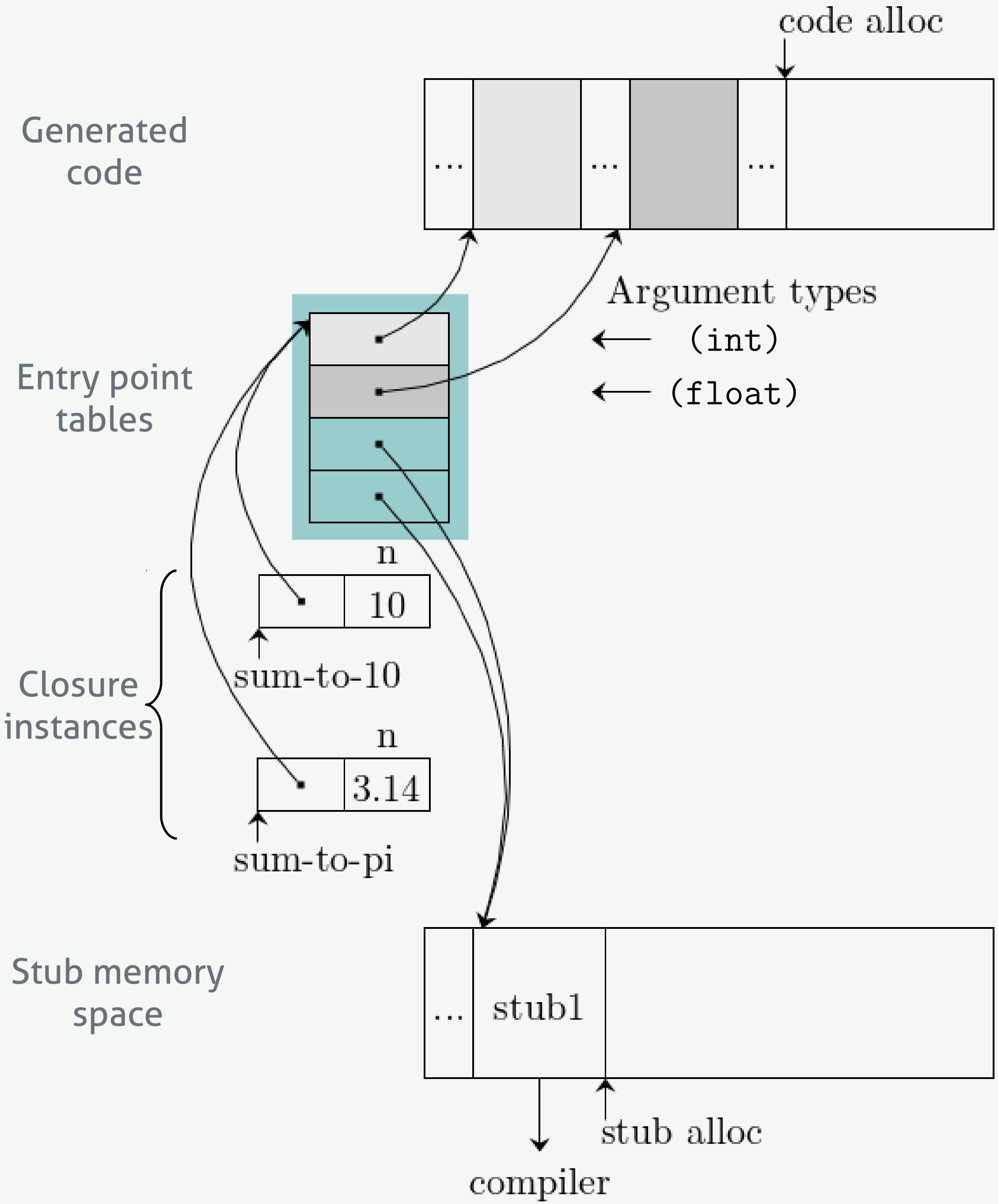

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: FLAT CLOSURE EXTENSION




```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: FLAT CLOSURE EXTENSION

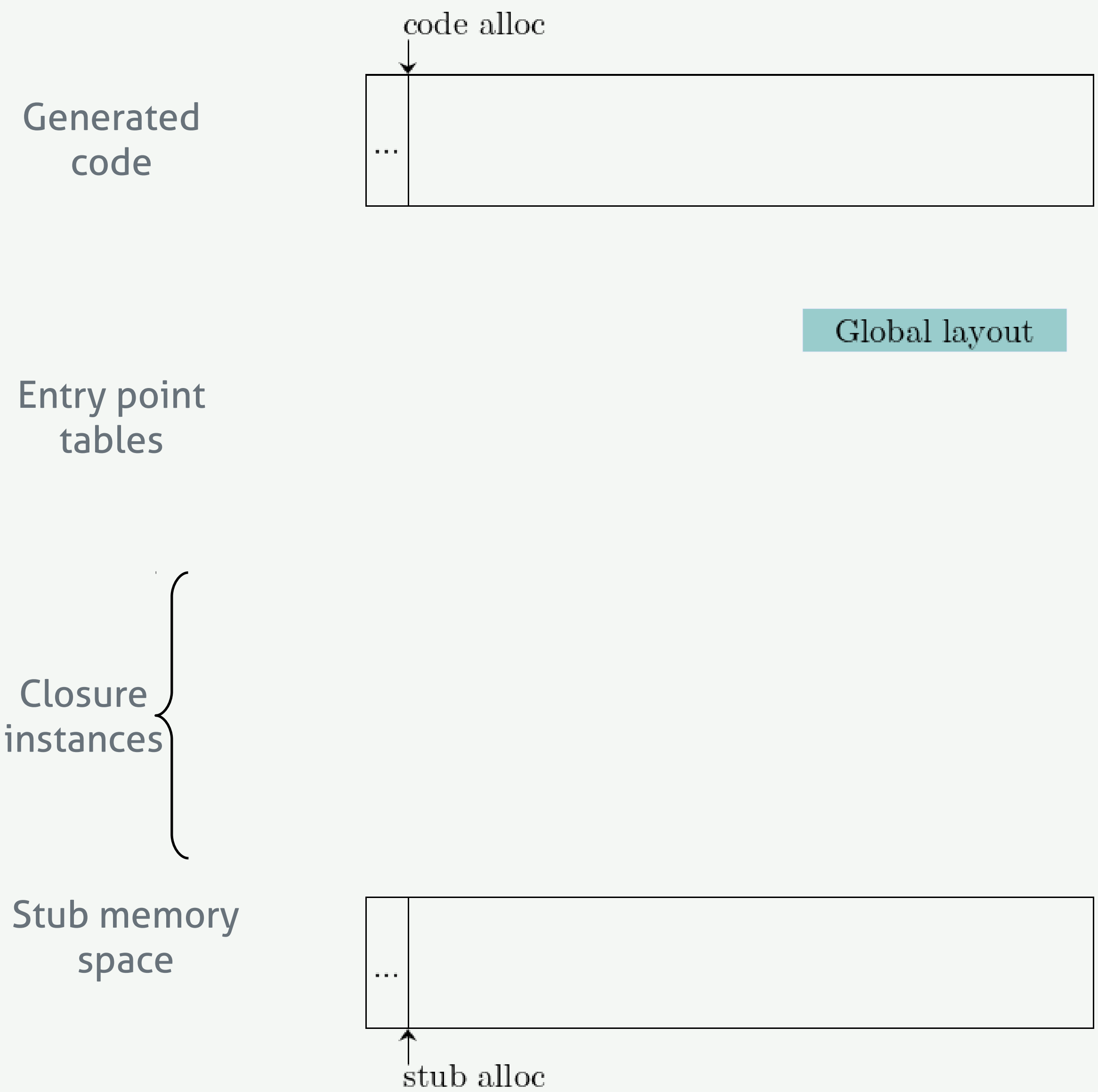



```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION

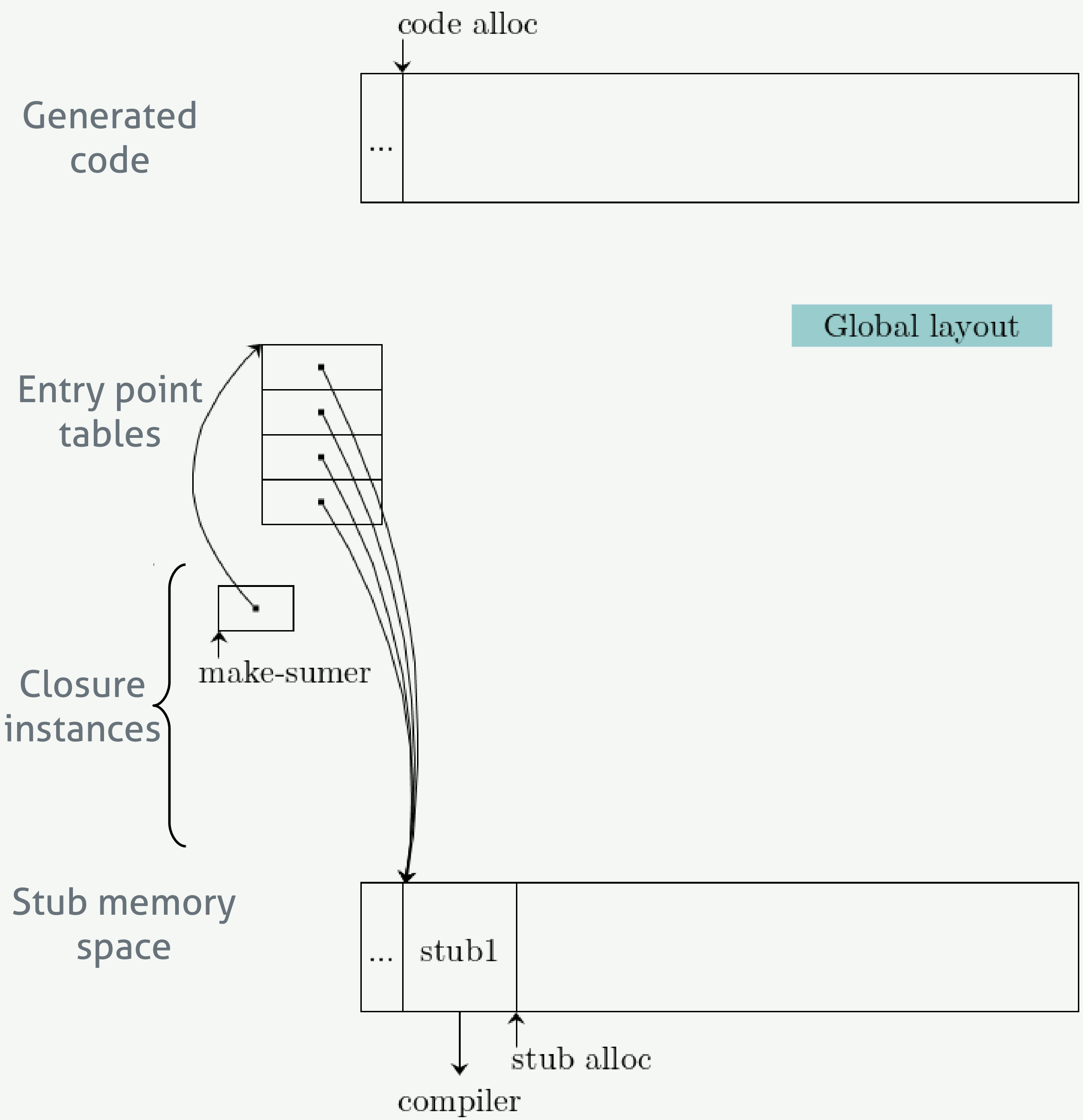
Dynamic dispatch

IMPLEMENTATION: DYNAMIC DISPATCH



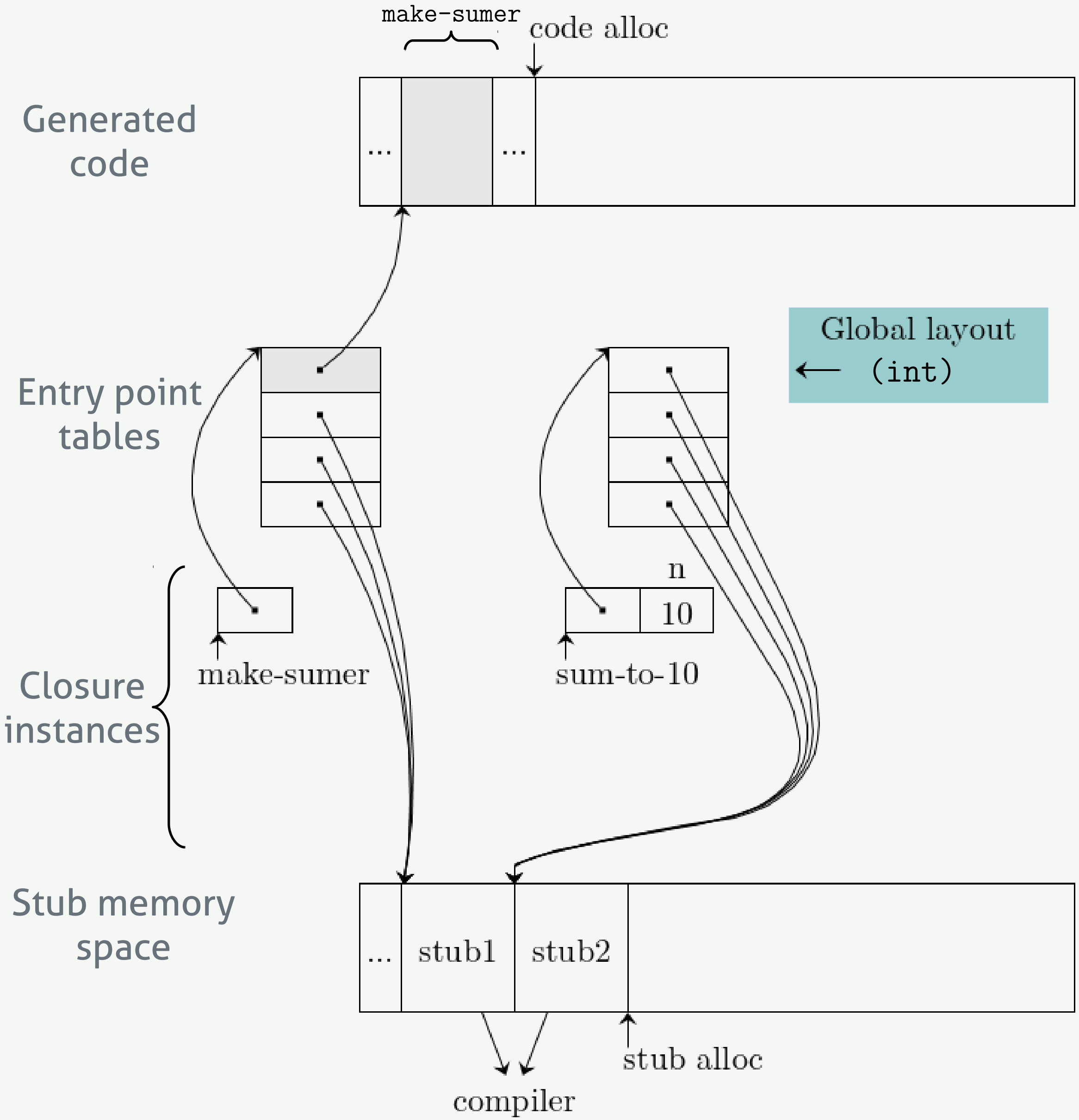

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: DYNAMIC DISPATCH



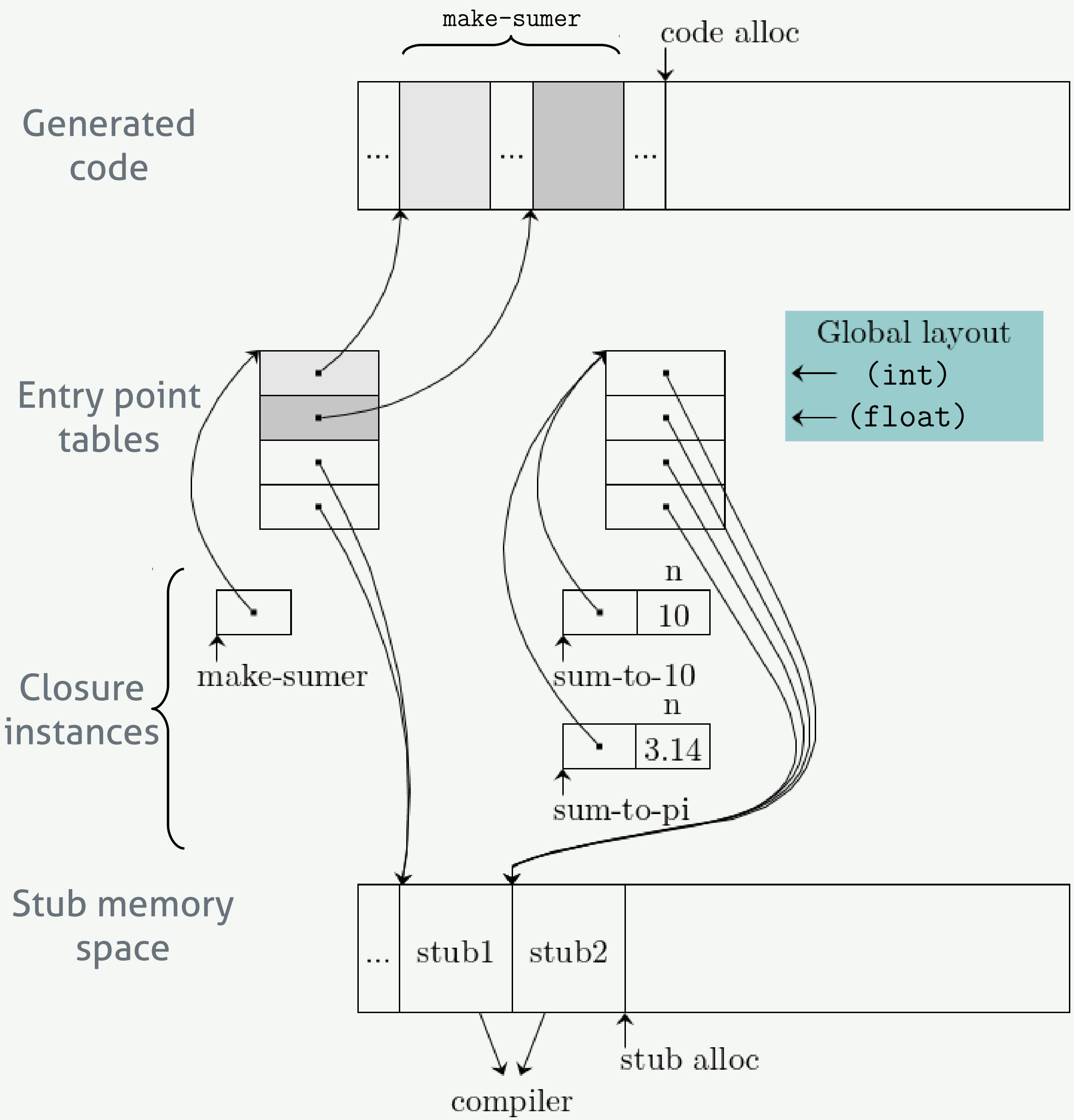

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: DYNAMIC DISPATCH



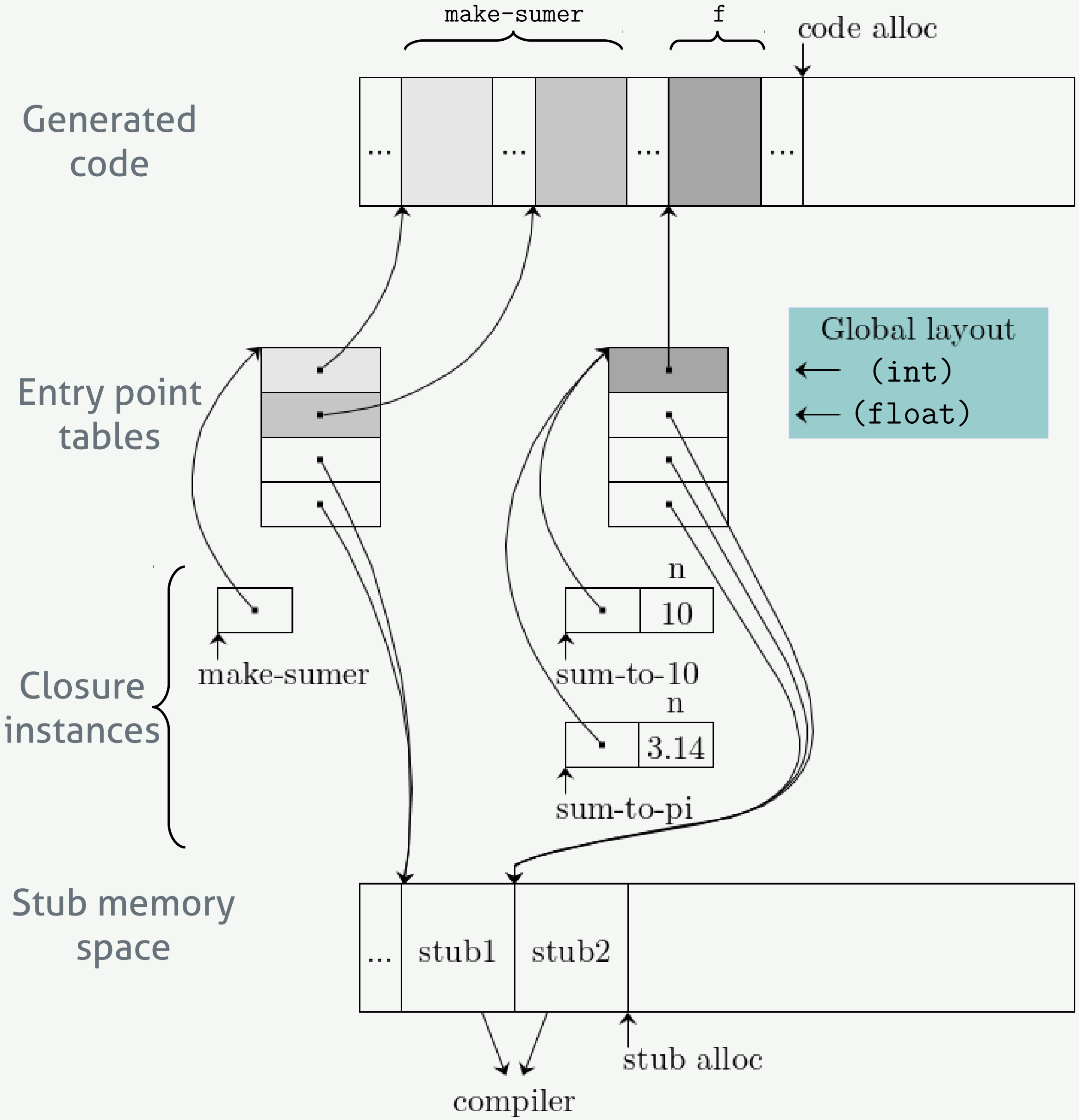

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: DYNAMIC DISPATCH



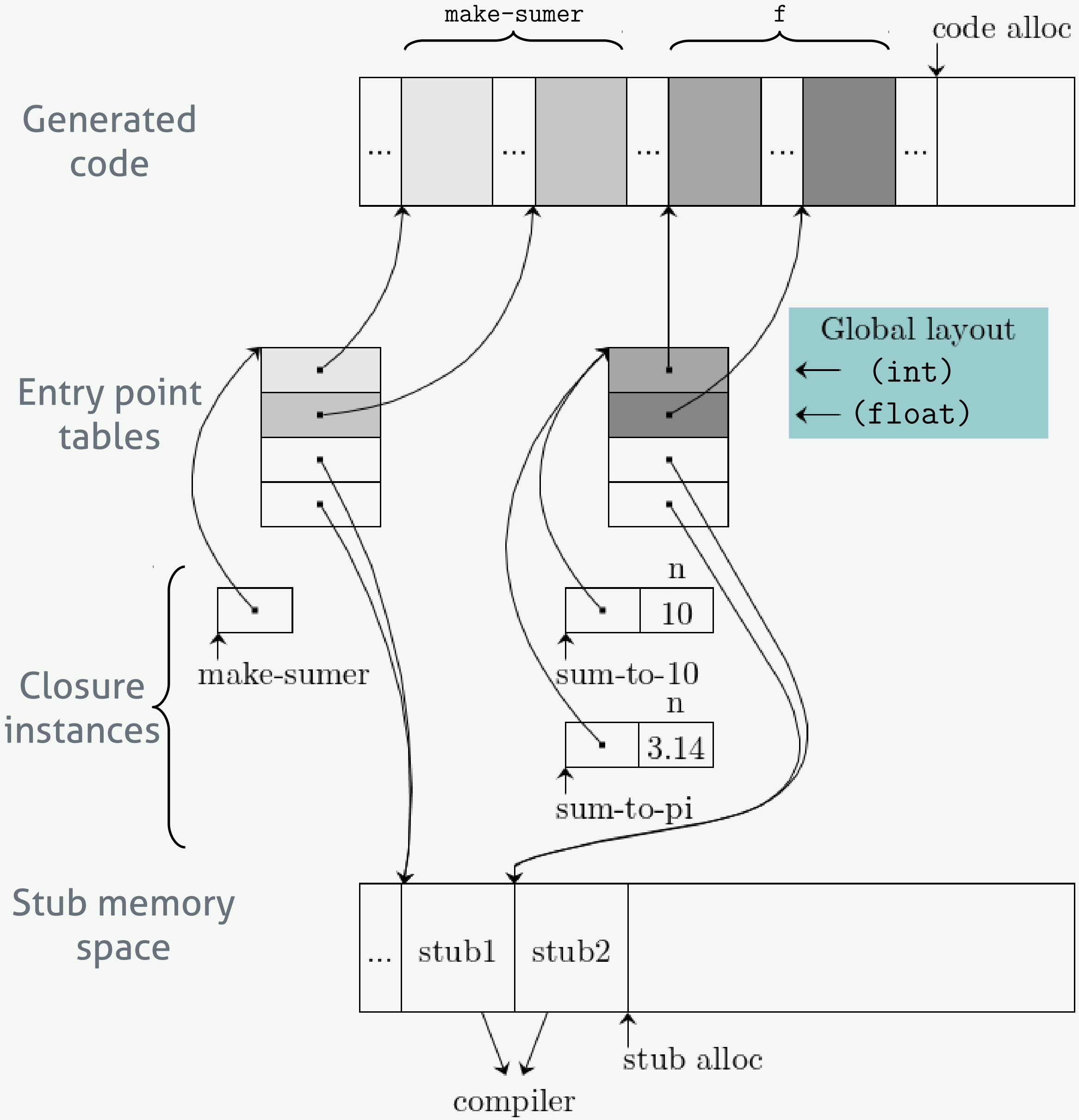

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: DYNAMIC DISPATCH



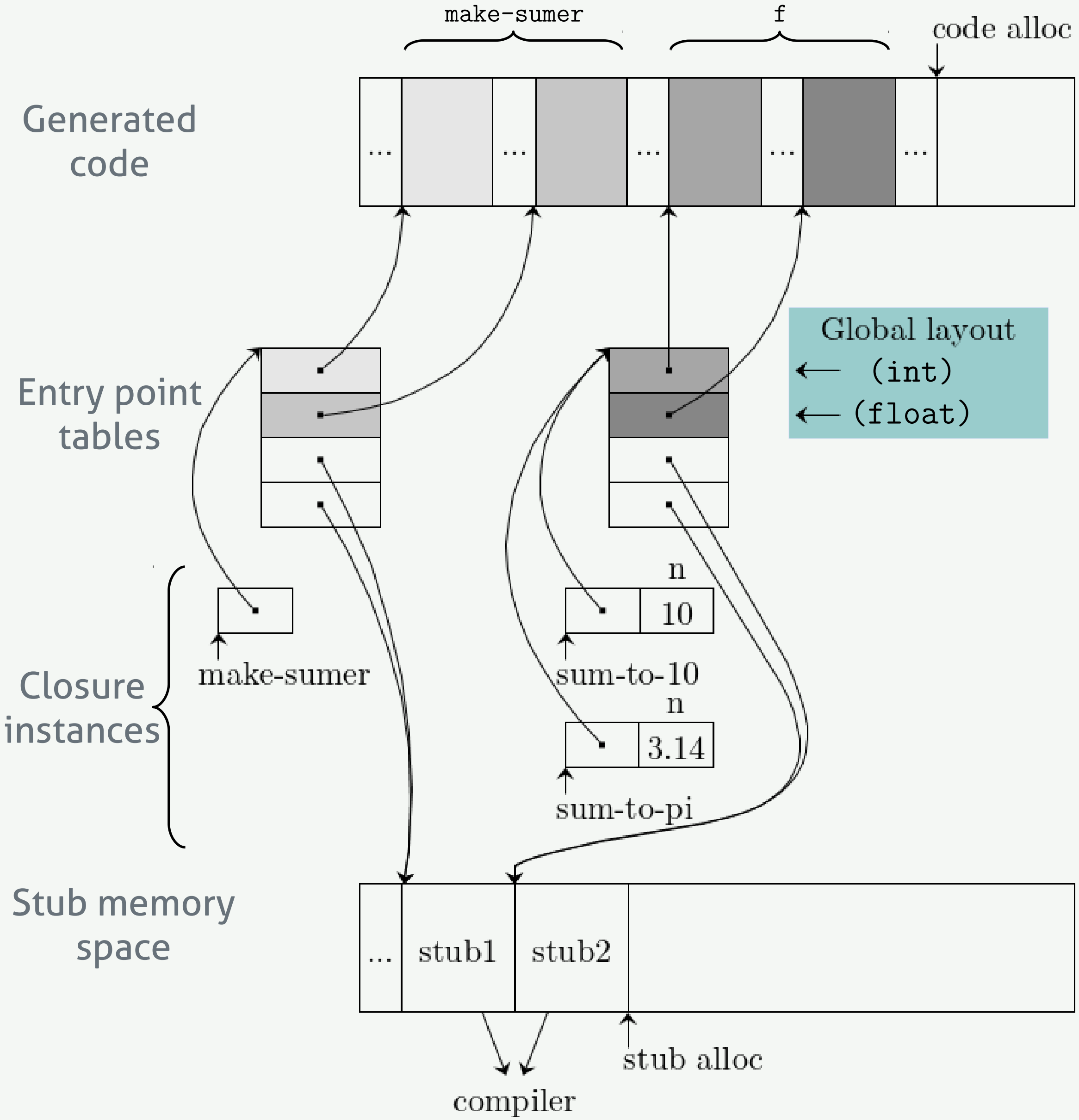

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: DYNAMIC DISPATCH




```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: DYNAMIC DISPATCH




```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION

Captured information

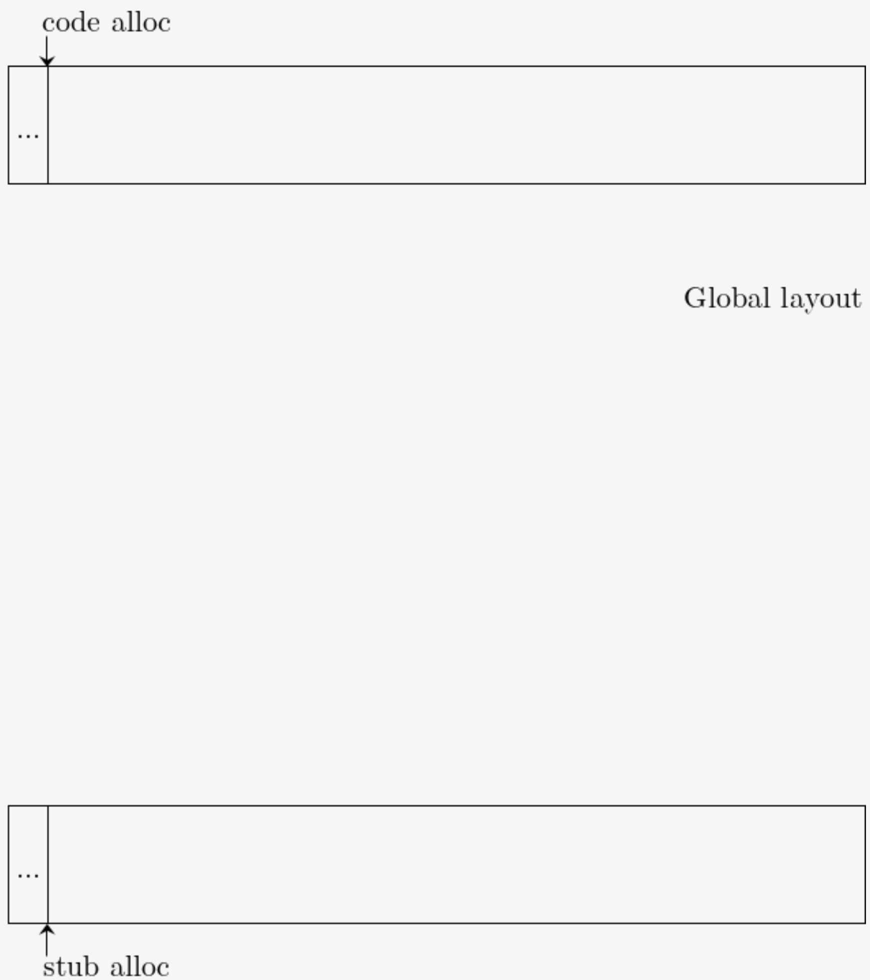
IMPLEMENTATION: CAPTURED INFORMATION

Generated code

Entry point tables

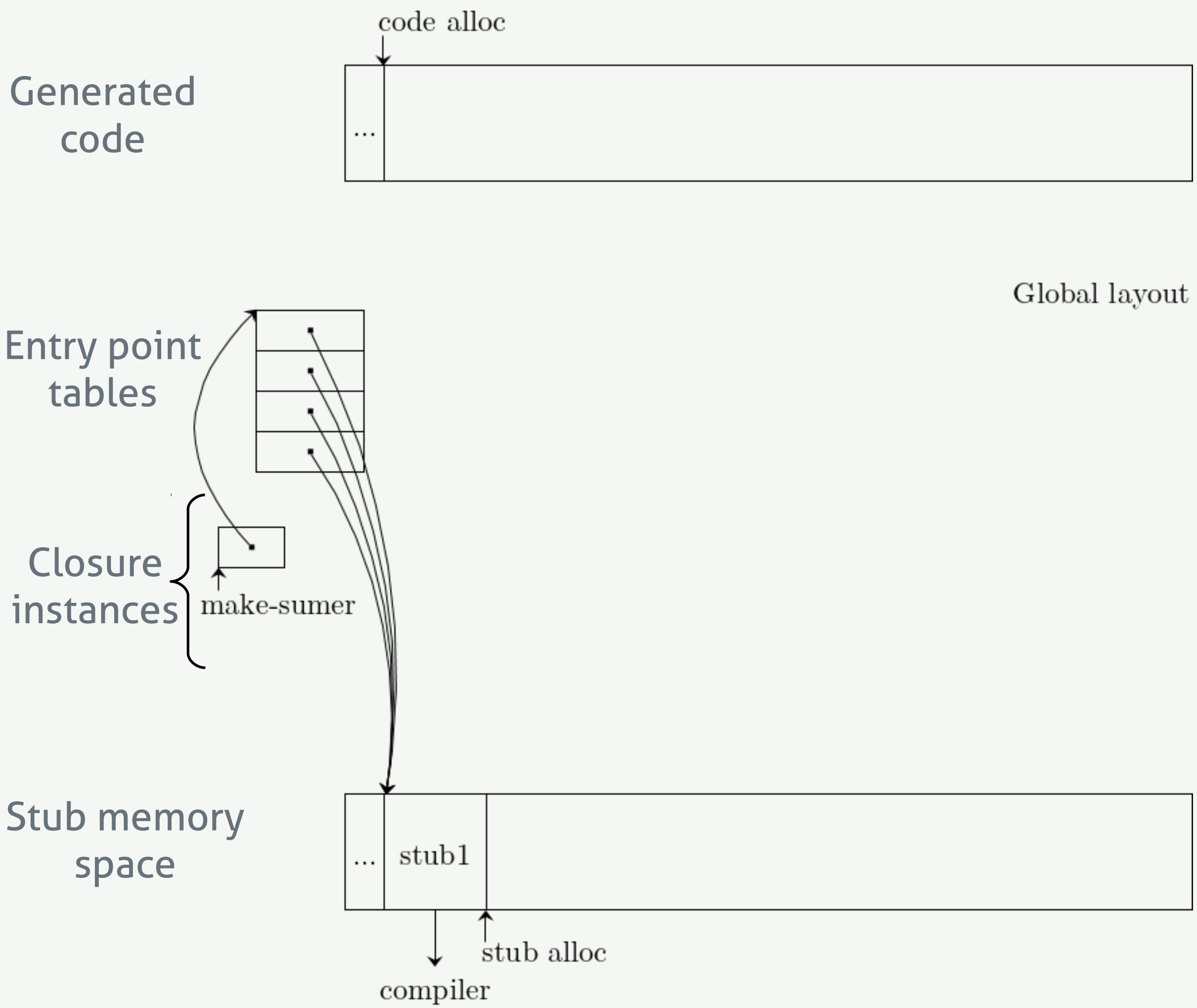
Closure instances

Stub memory space



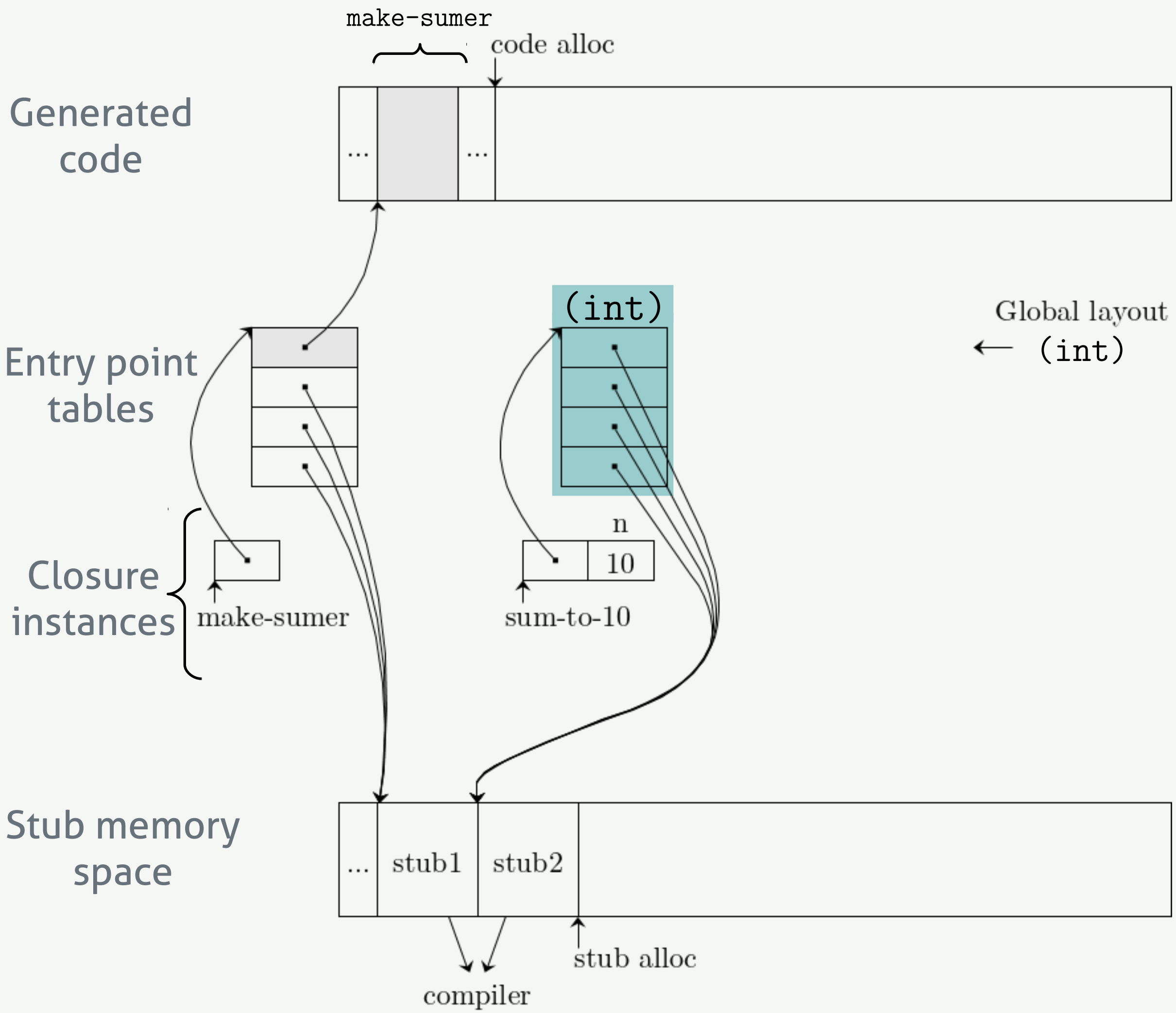

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: CAPTURED INFORMATION



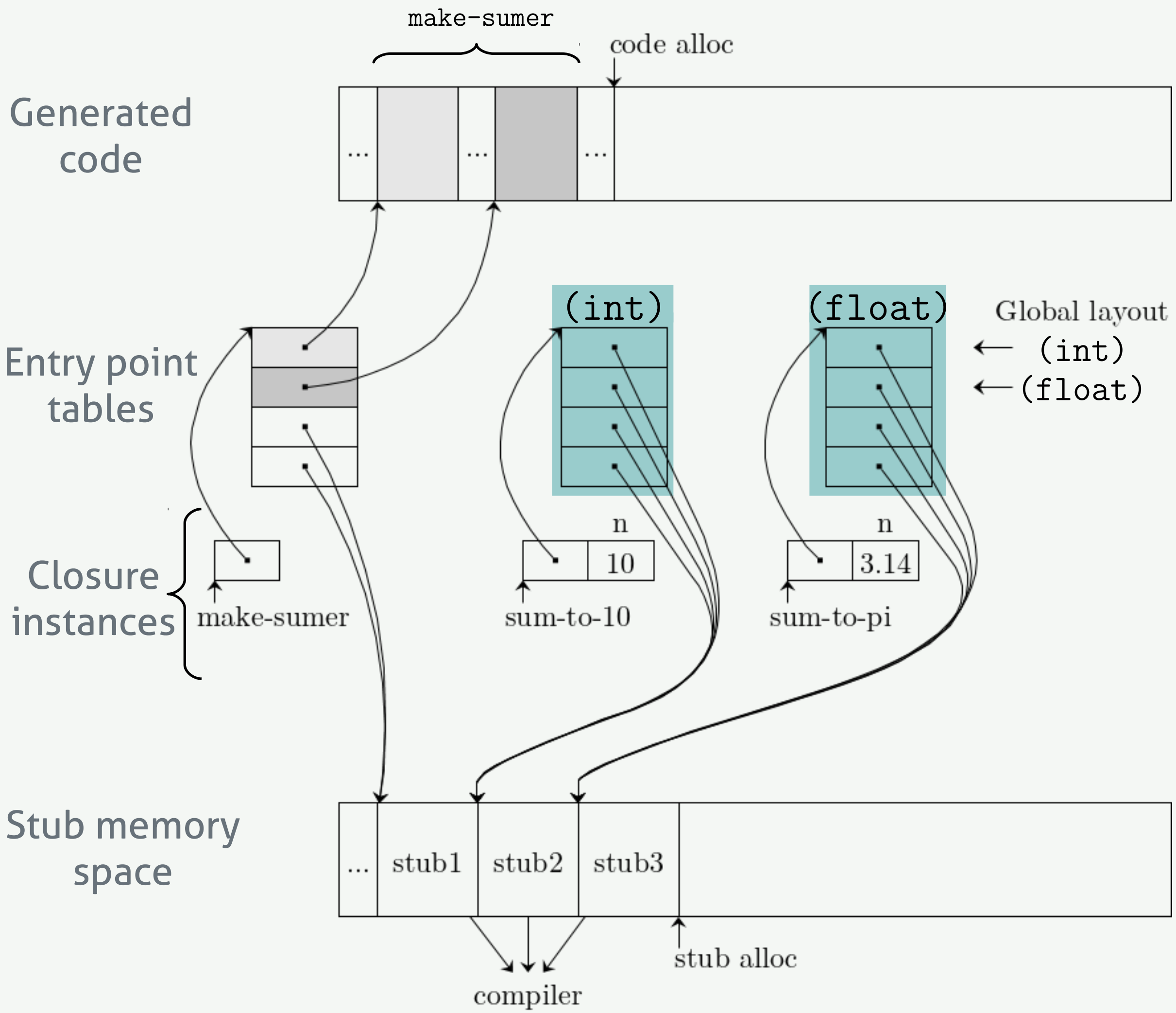

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: CAPTURED INFORMATION



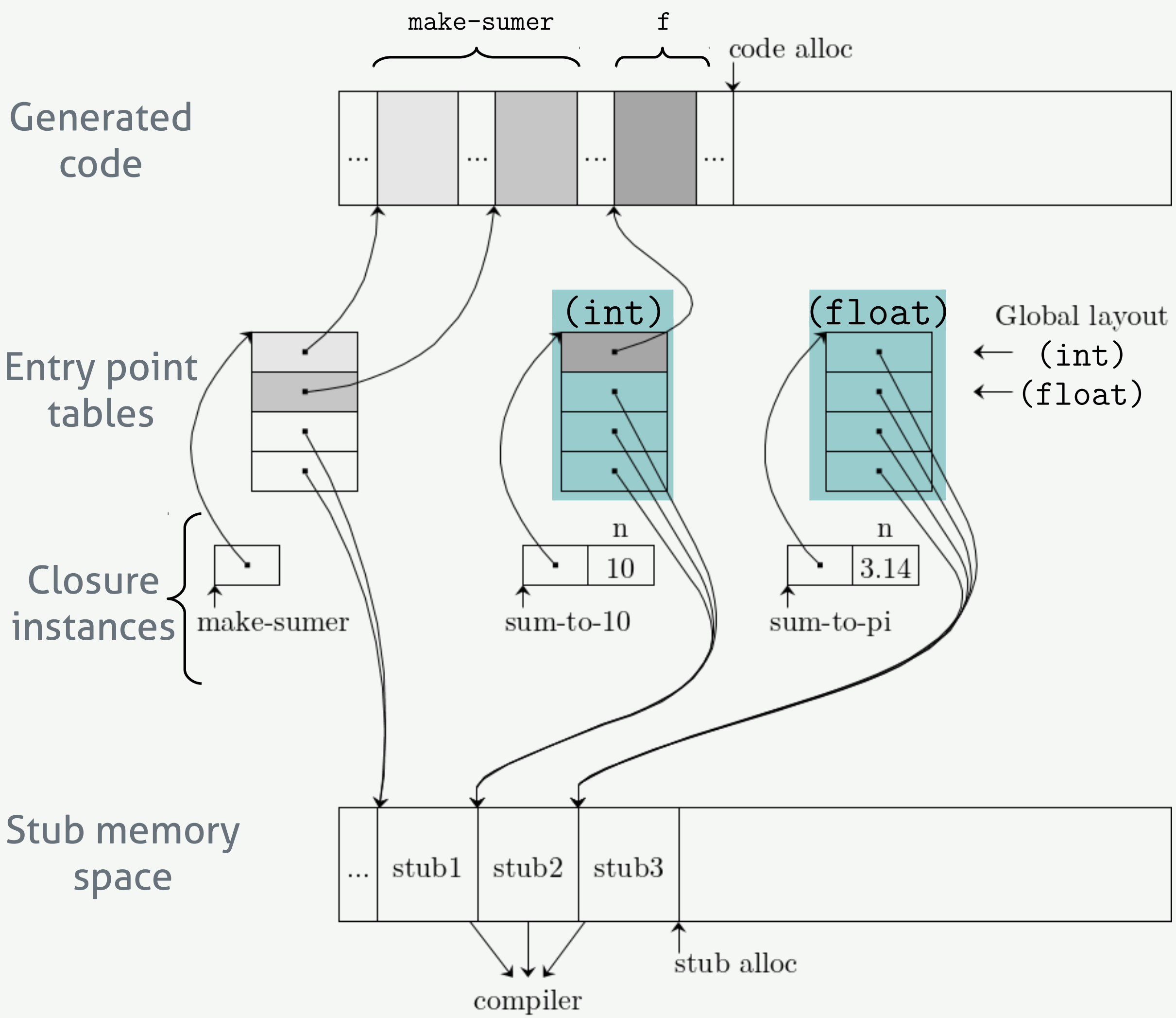

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: CAPTURED INFORMATION



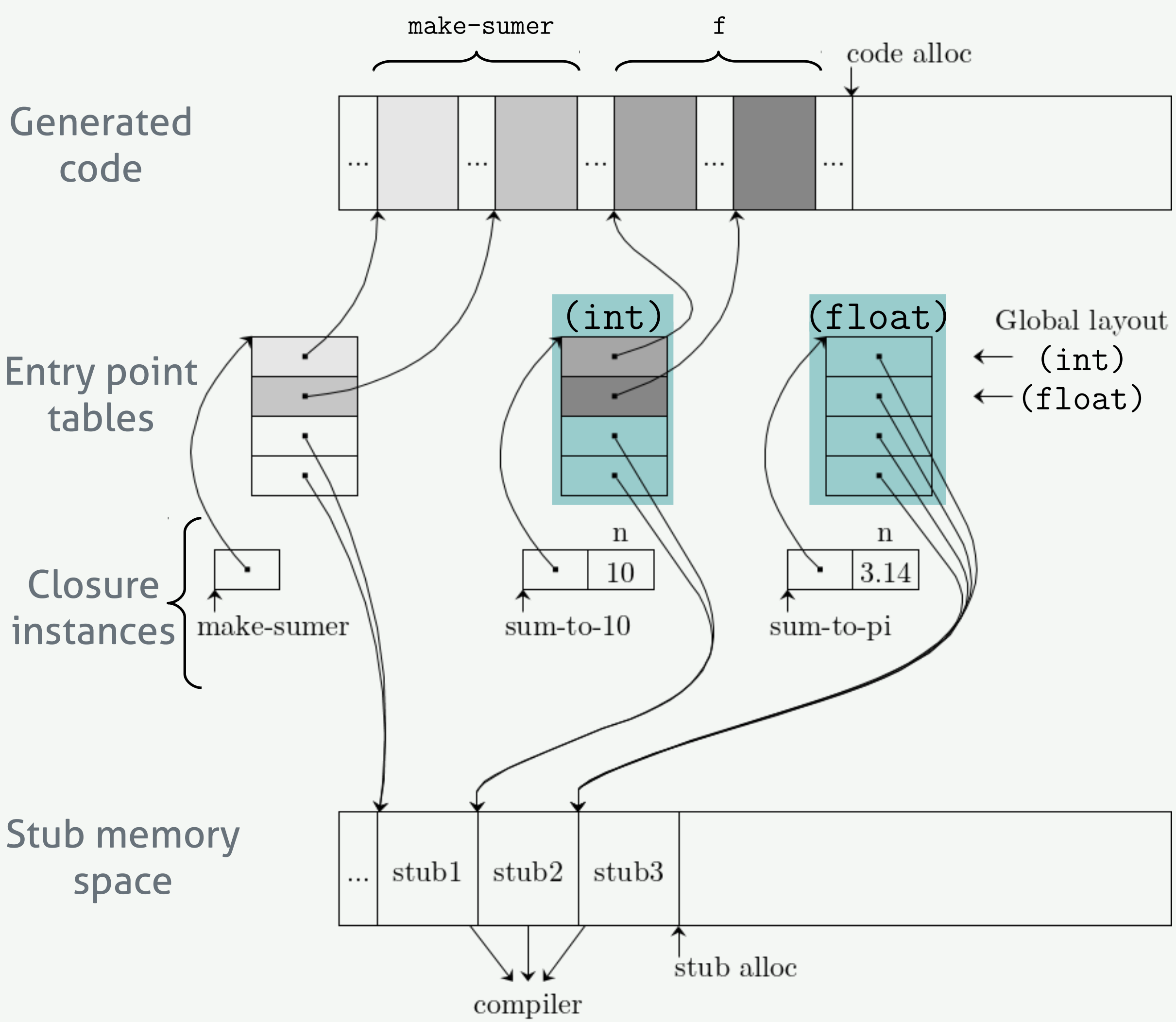

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: CAPTURED INFORMATION



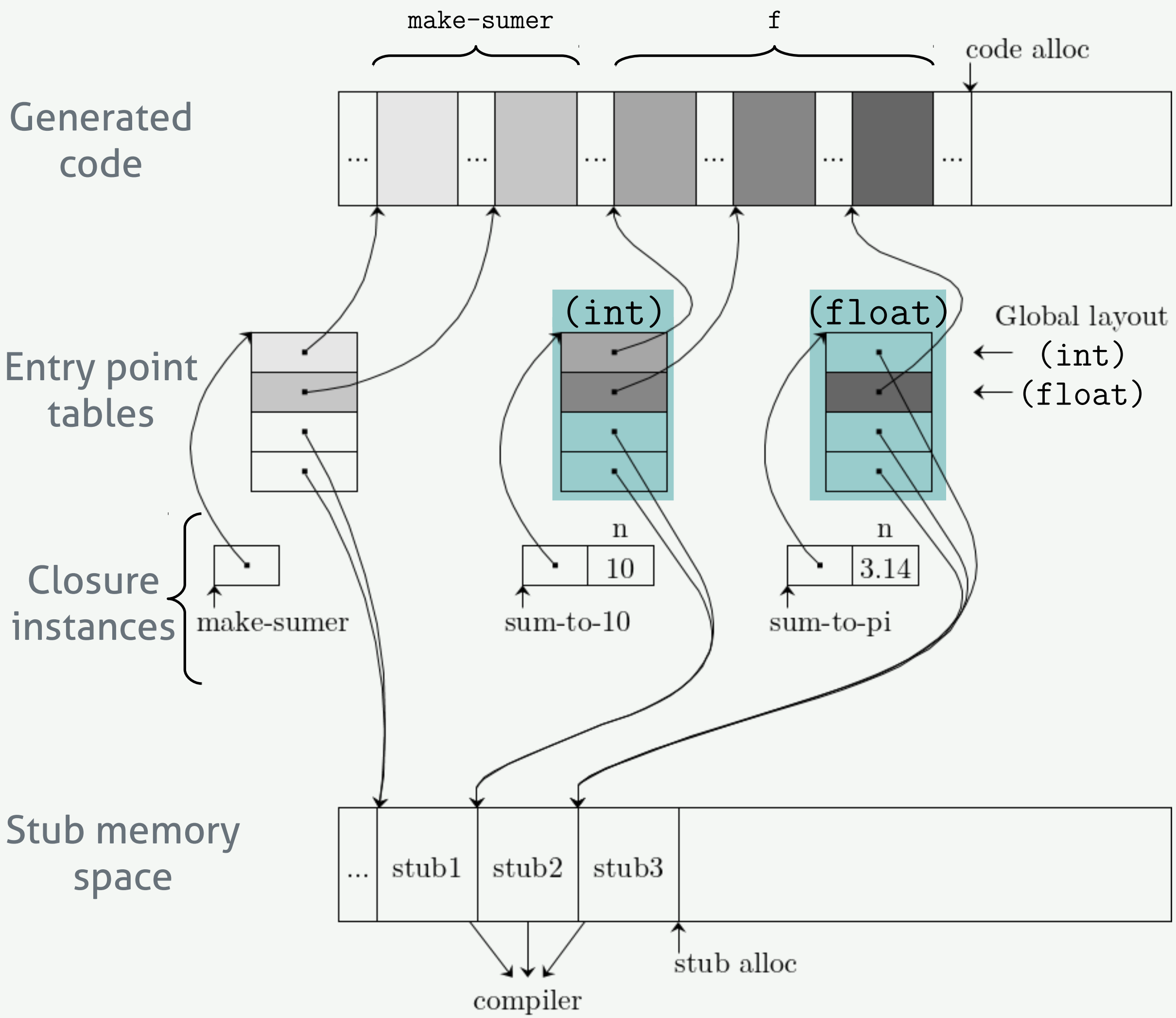

```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: CAPTURED INFORMATION




```
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION: CAPTURED INFORMATION




```scheme
(define (make-sumer n)
  (letrec ((f (lambda (x)
                (if (> x n)
                    0
                    (+ x (f (+ x 1)))))))
    f))

(define sum-to-10 (make-sumer 10))
(define sum-to-pi (make-sumer 3.14))

; 6 + 7 + 8 + 9 + 10
(print (sum-to-10 6))
; 7.5 + 8.5 + 9.5
(print (sum-to-10 7.5))
; 1.10 + 2.10 + 3.10
(print (sum-to-pi 1.10))
```

IMPLEMENTATION

- What about continuations ?

IMPLEMENTATION

- What about continuations ?

→ Function return is a call to the continuation
→ Conceptually same implementation !

IMPLEMENTATION

- What about continuations ?

  → Function return is a call to the continuation
  → Conceptually same implementation !

  - Propagated types → Type of the returned value
  - Captured types → Type of the local variables

RESULTS

RESULTS

- 35 benchmarks: Standard Scheme benchmarks

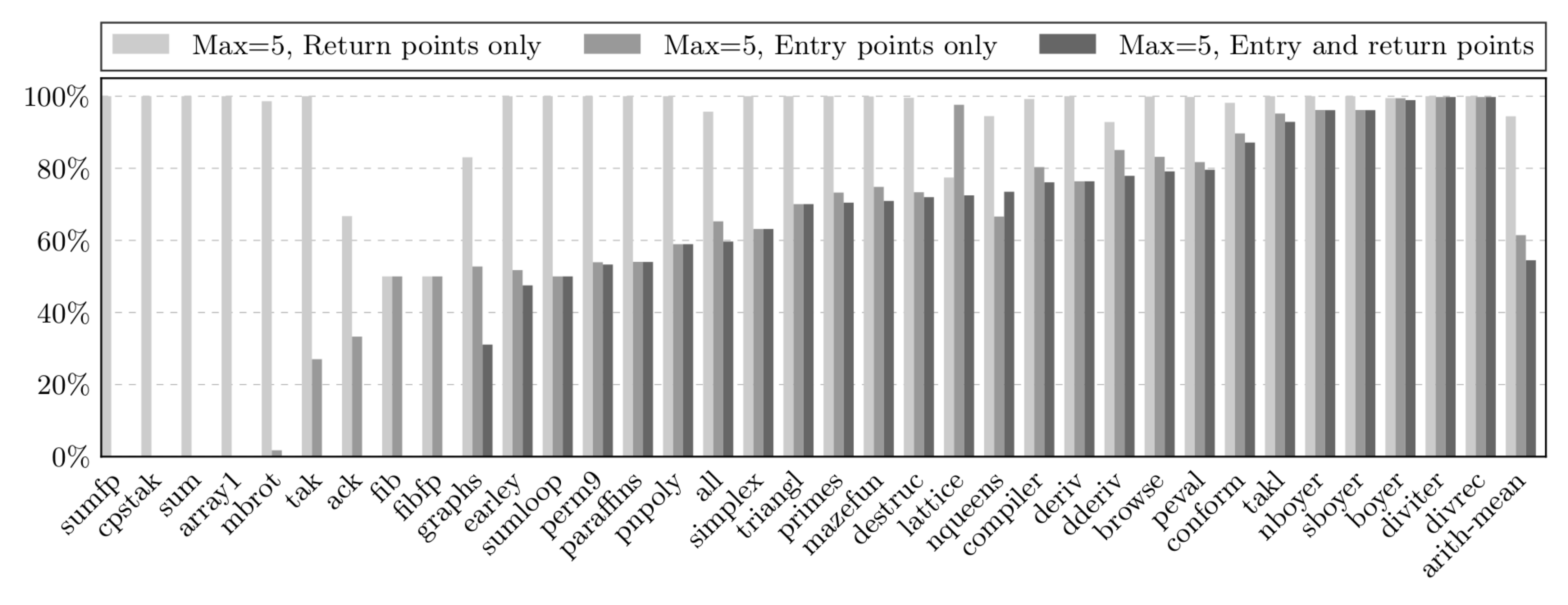
- Configurations

  - **Intraprocedural BBV**
  - Interprocedural: function specialization
  - Interprocedural: continuation specialization
  - **Interprocedural: function and continuation specialization**

- Metrics

  - Number of type checks
  - Generated code size
  - Execution / Compilation / Total time

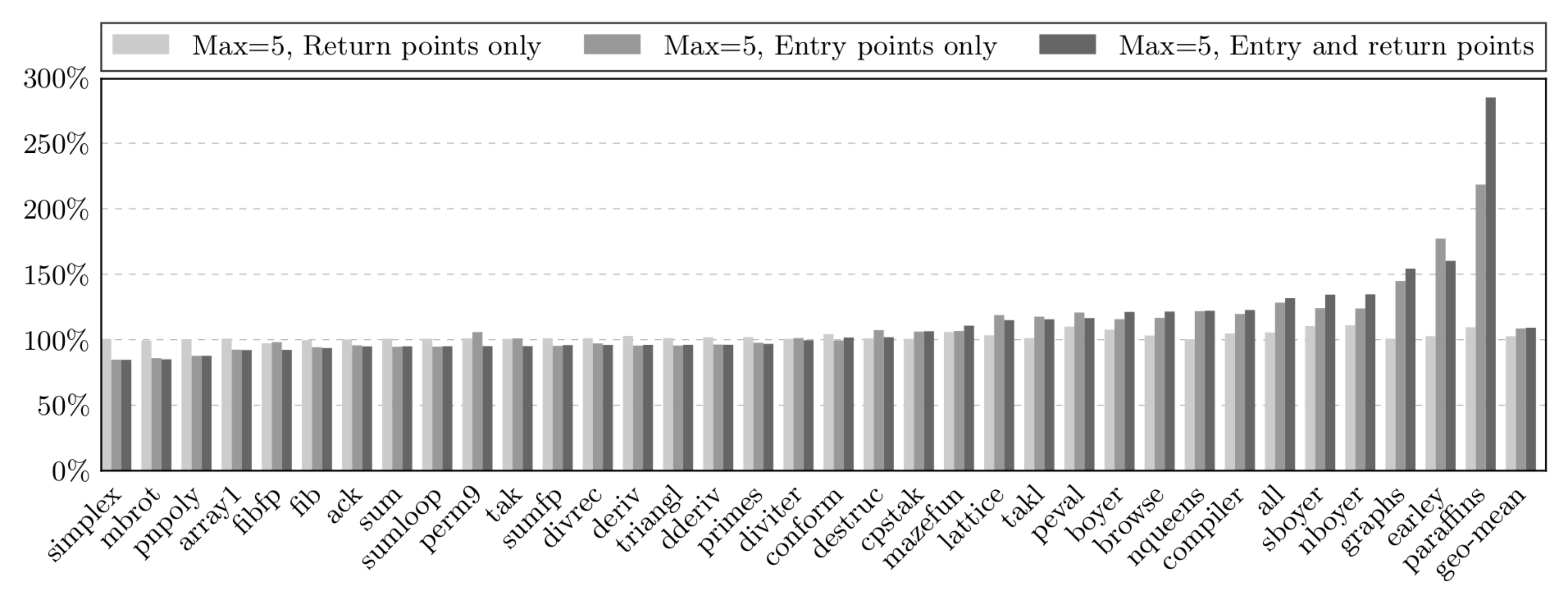
NUMBER OF TYPE CHECKS



- Number of executed type checks relative to pure intraprocedural specialization

- No checks for 9 benchmarks
- Significantly fewer checks for most of the benchmarks

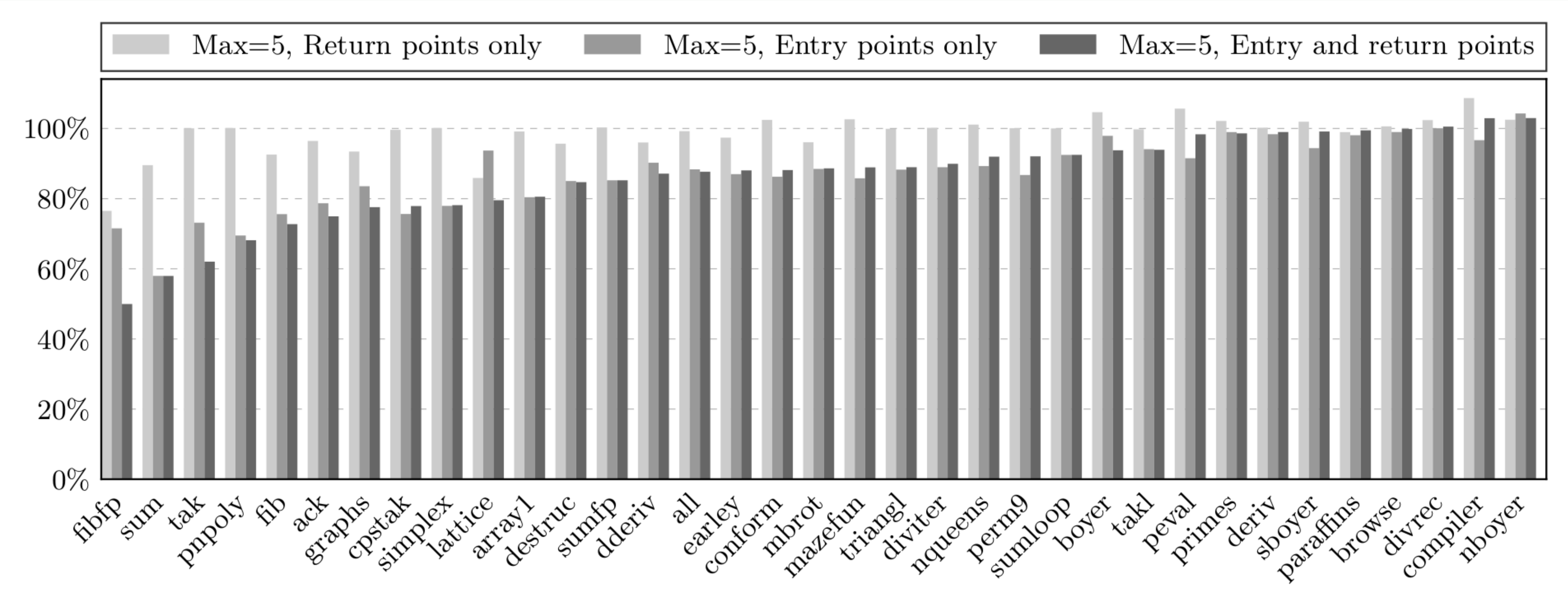
CODE SIZE



- Generated code size relative to pure intraprocedural specialization

- Less code generated for half of the benchmarks
- Just 9% more code generated on average

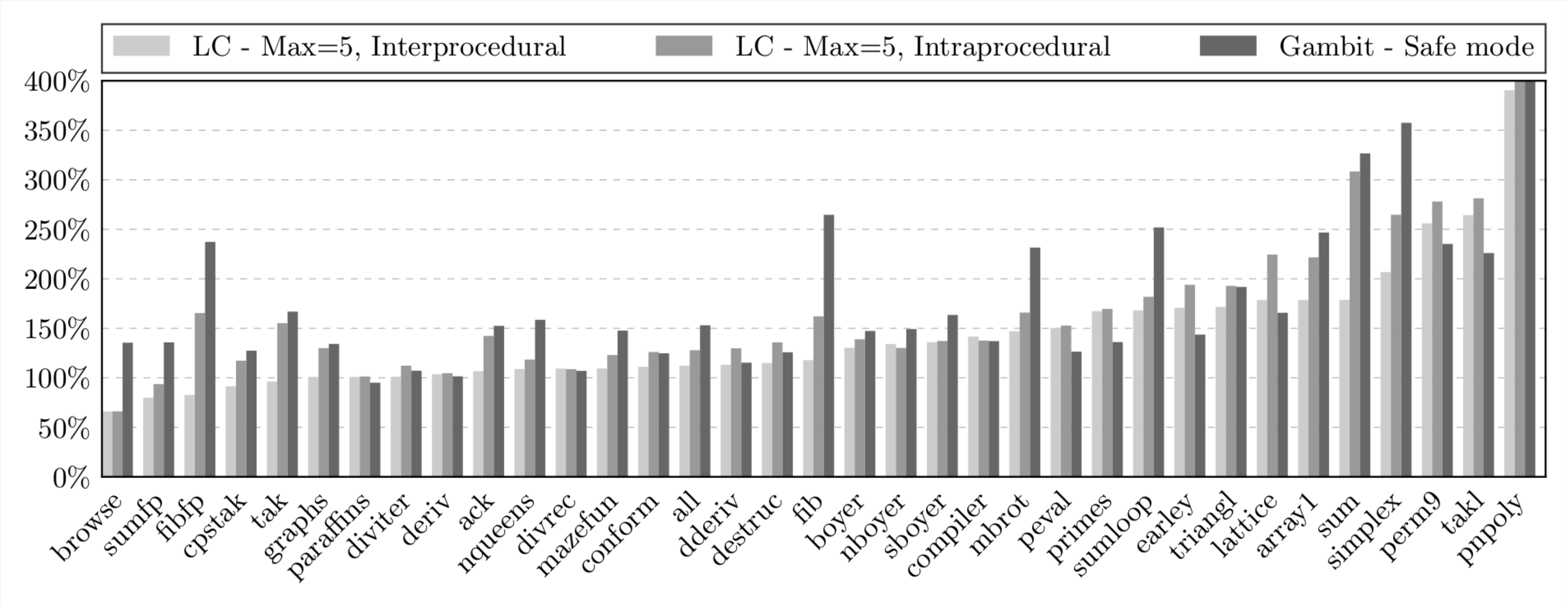
EXECUTION TIME



- Execution time relative to pure intraprocedural specialization

- Almost all benchmarks are faster
- No benchmark is significantly slower
- Up to 2x faster

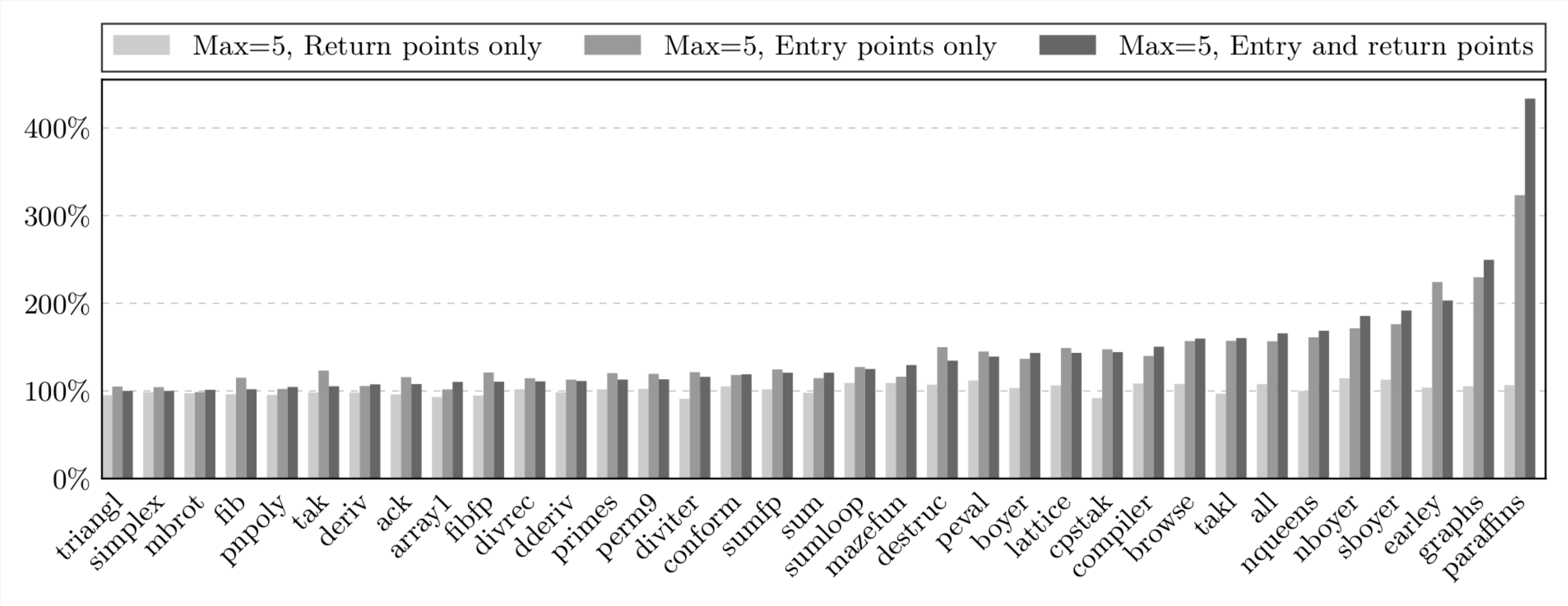
EXECUTION TIME (VS GAMBIT)



- Execution time relative to the execution time of the code generated by the Gambit Scheme compiler with dynamic checks turned off (capped at 400%)

- Faster than Gambit with dynamic checks turned on
- Varies from 0.5x to 4x (6x slower with Gambit executing all the checks)

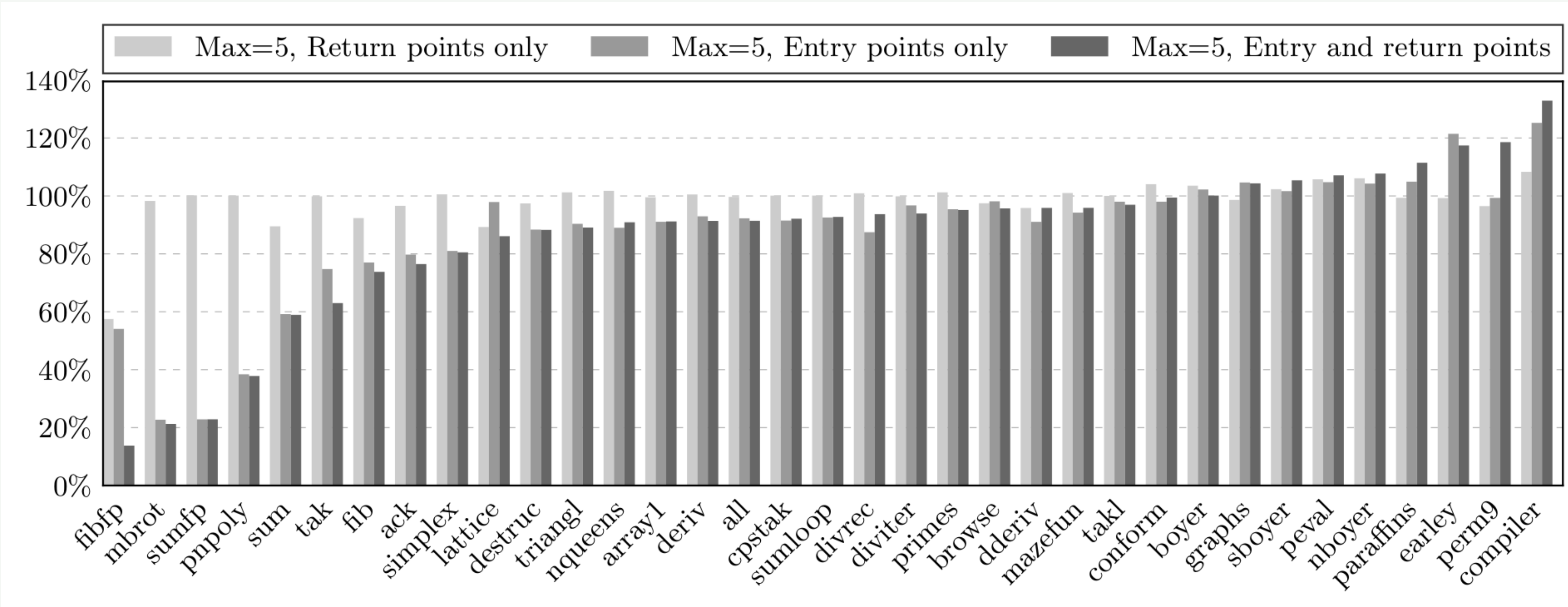
COMPILATION TIME



- Compilation time relative to pure intraprocedural BBV

- Compilation time increase for most of the benchmarks
- Mostly small increase in compilation time, but up to 4x

TOTAL TIME



- Total time relative to pure intraprocedural BBV

- Large speedup for floating point (avoids boxing)
- Most of the benchmarks are faster
- From 0.14x to 1.33x

CONCLUSION

CONCLUSION

- Lazy interprocedural specialization
- Works well
  - Checks removed (up to 100%)
  - Faster code (up to 50%)
- Simple
  - Simple to implement
  - Does not require complex architecture

Future Work

- Propagating other properties
  - Value (e.g. `x = 10`)
  - Variable relationship (e.g. `x < y`)
- Function identity and return address propagation
  - Dynamic function inlining
- Register allocation information
  - Arguments
  - Returned value