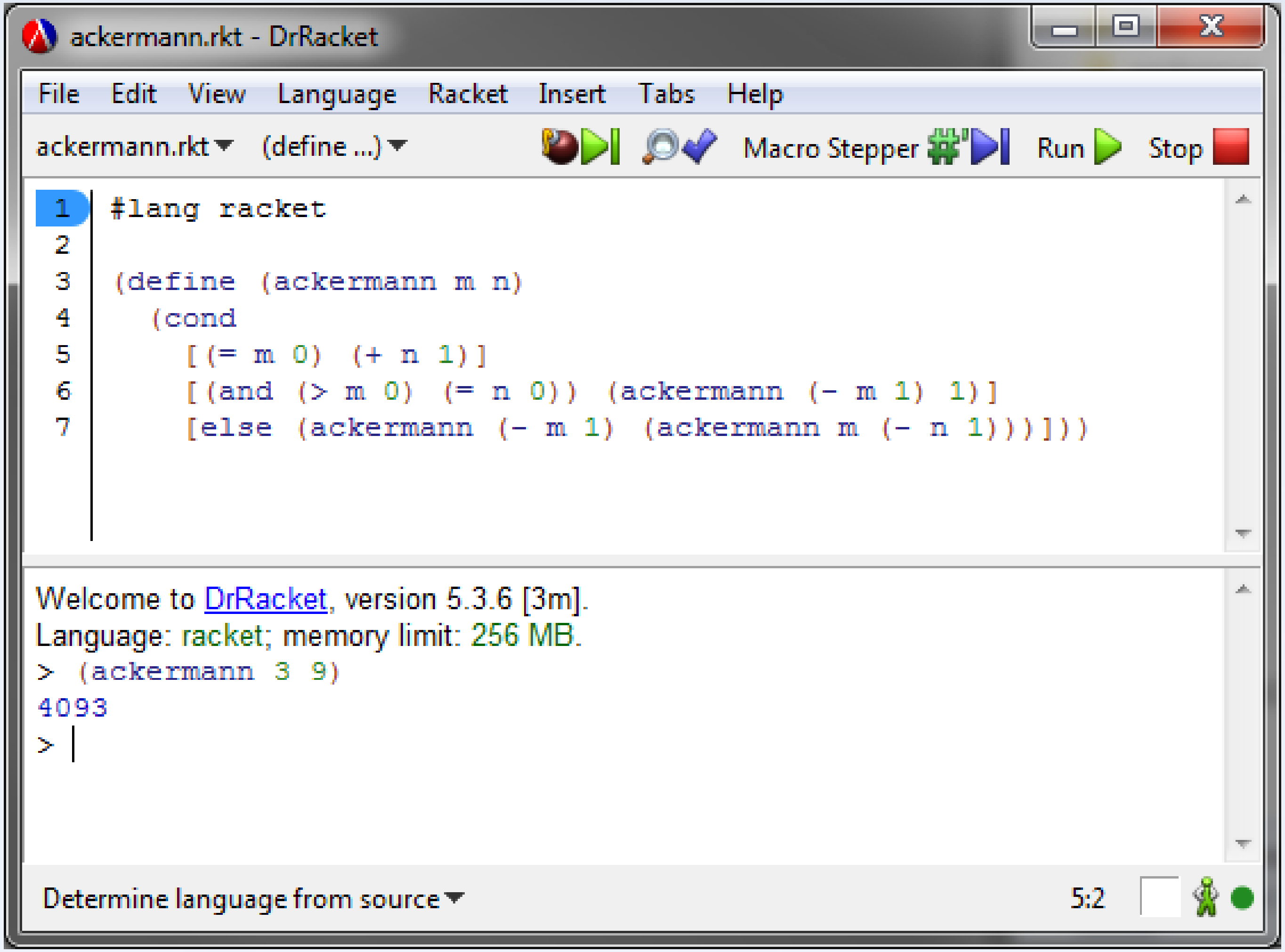
An Implementation of Python for Racket

Pedro Palma Ramos
António Menezes Leitão

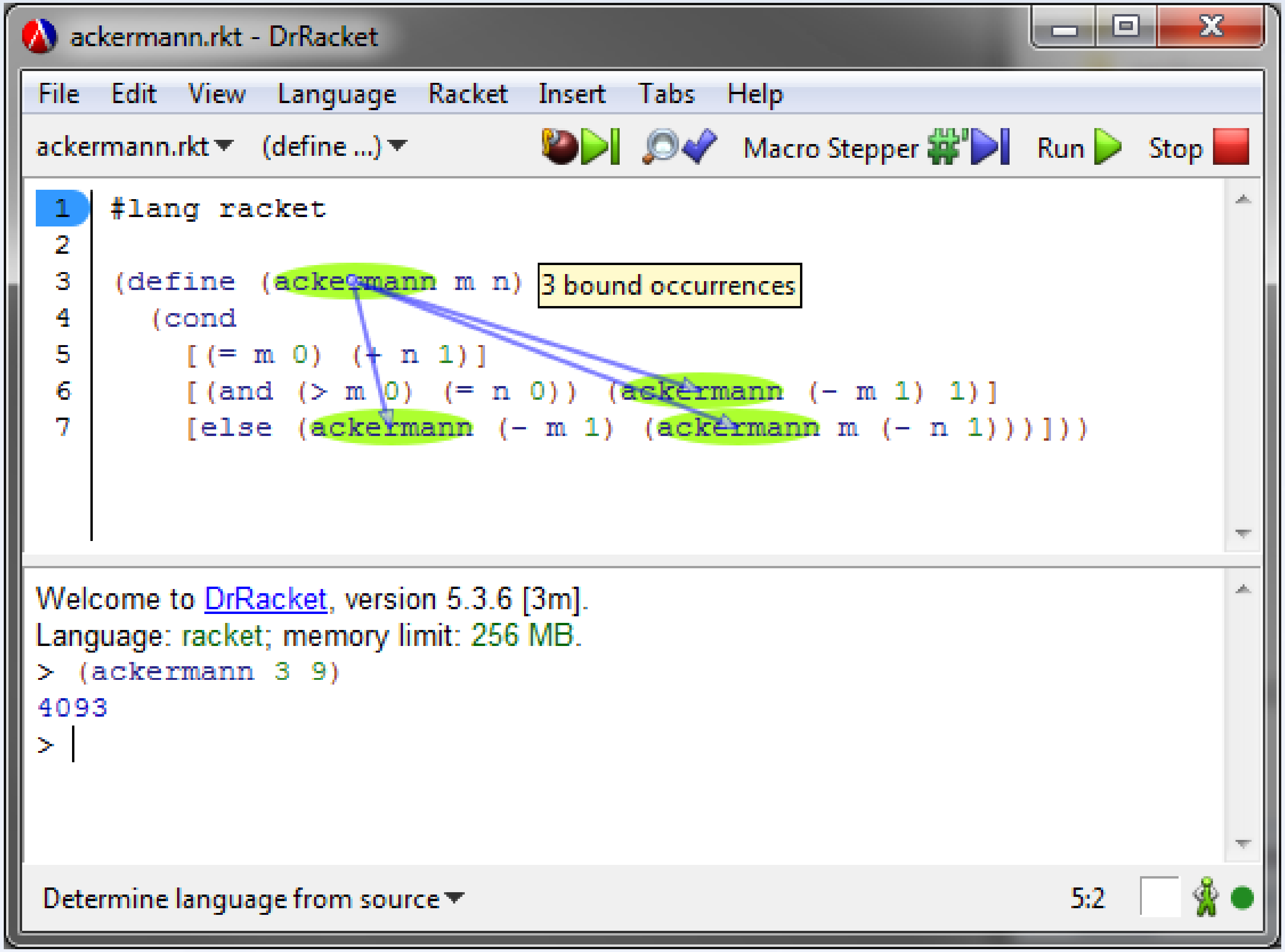
Contents

- Motivation
- Goals
- Related Work
- Solution
- Performance Benchmarks
- Future Work

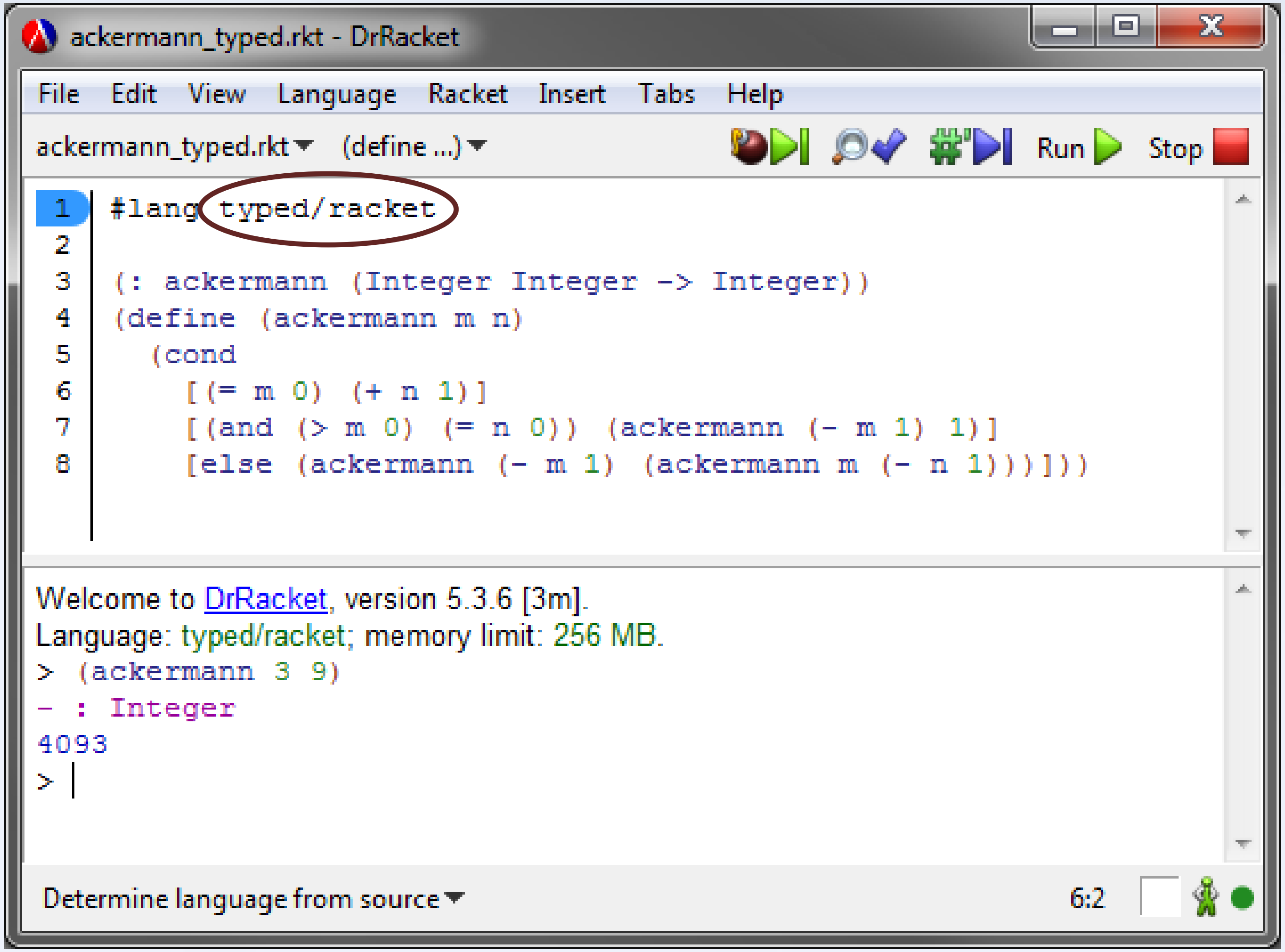
Racket + DrRacket



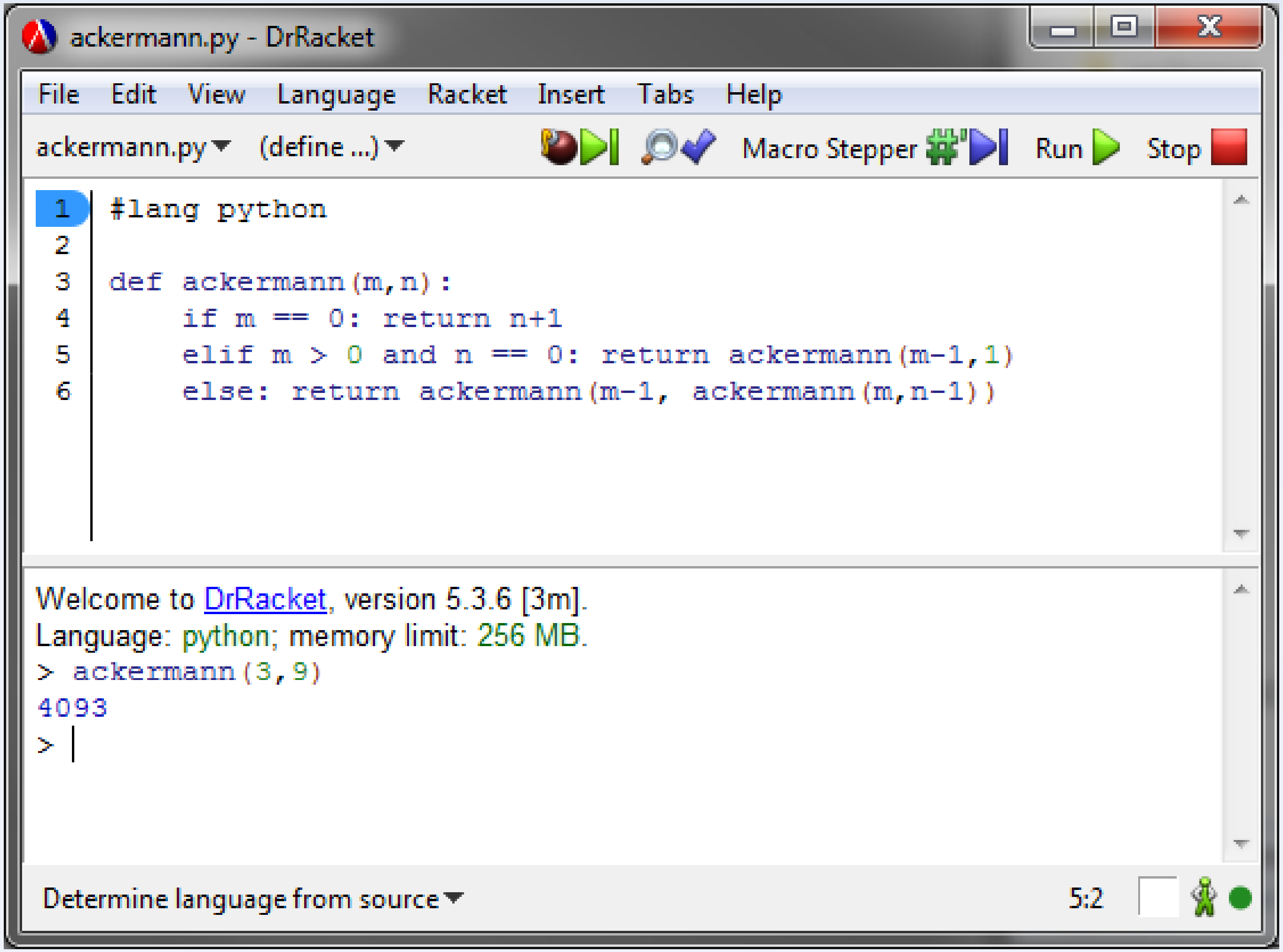
Racket + DrRacket



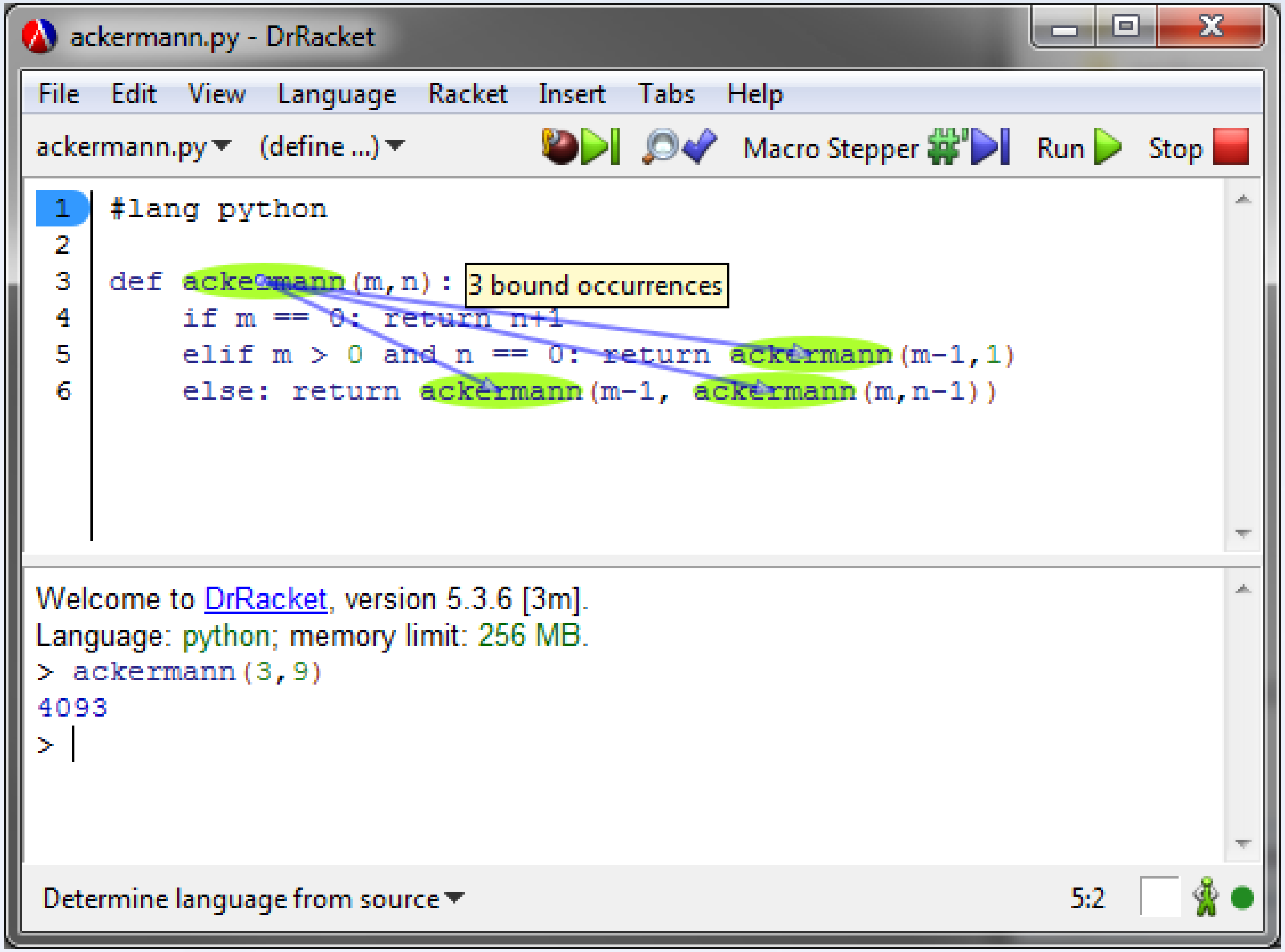
Racket + DrRacket



Our goal...


ackermann.py - DrRacket
File Edit View Language Racket Insert Tabs Help
ackermann.py (define ...)
Macro Stepper
Run
Stop
1 #lang python
2
3 def ackermann(m,n):
4 if m == 0: return n+1
5 elif m > 0 and n == 0: return ackermann(m-1,1)
6 else: return ackermann(m-1, ackermann(m,n-1))
Welcome to DrRacket, version 5.3.6 [3m].
Language: python; memory limit: 256 MB.
> ackermann(3,9)
4093
>
Determine language from source
5:2

Our goal...

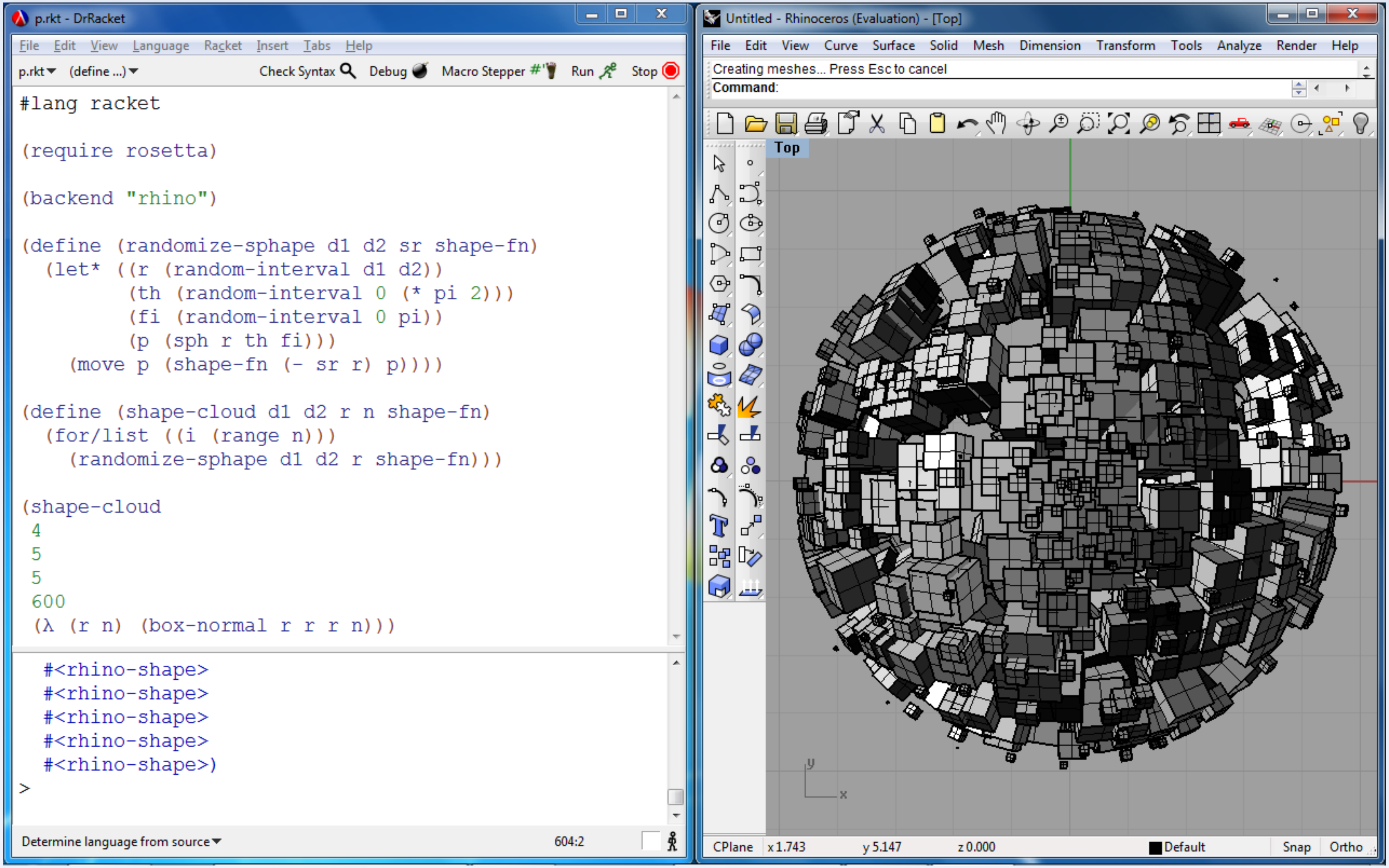


ackermann.py - DrRacket
File Edit View Language Racket Insert Tabs Help
ackermann.py (define ...)
Macro Stepper
Run
Stop
1 #lang python
2
3 def ackermann(m,n):
4   if m == 0: return n+1
5   elif m > 0 and n == 0: return ackermann(m-1,1)
6   else: return ackermann(m-1, ackermann(m,n-1))
3 bound occurrences
Welcome to DrRacket, version 5.3.6 [3m].
Language: python; memory limit: 256 MB.
> ackermann(3,9)
4093
>
Determine language from source
5:2

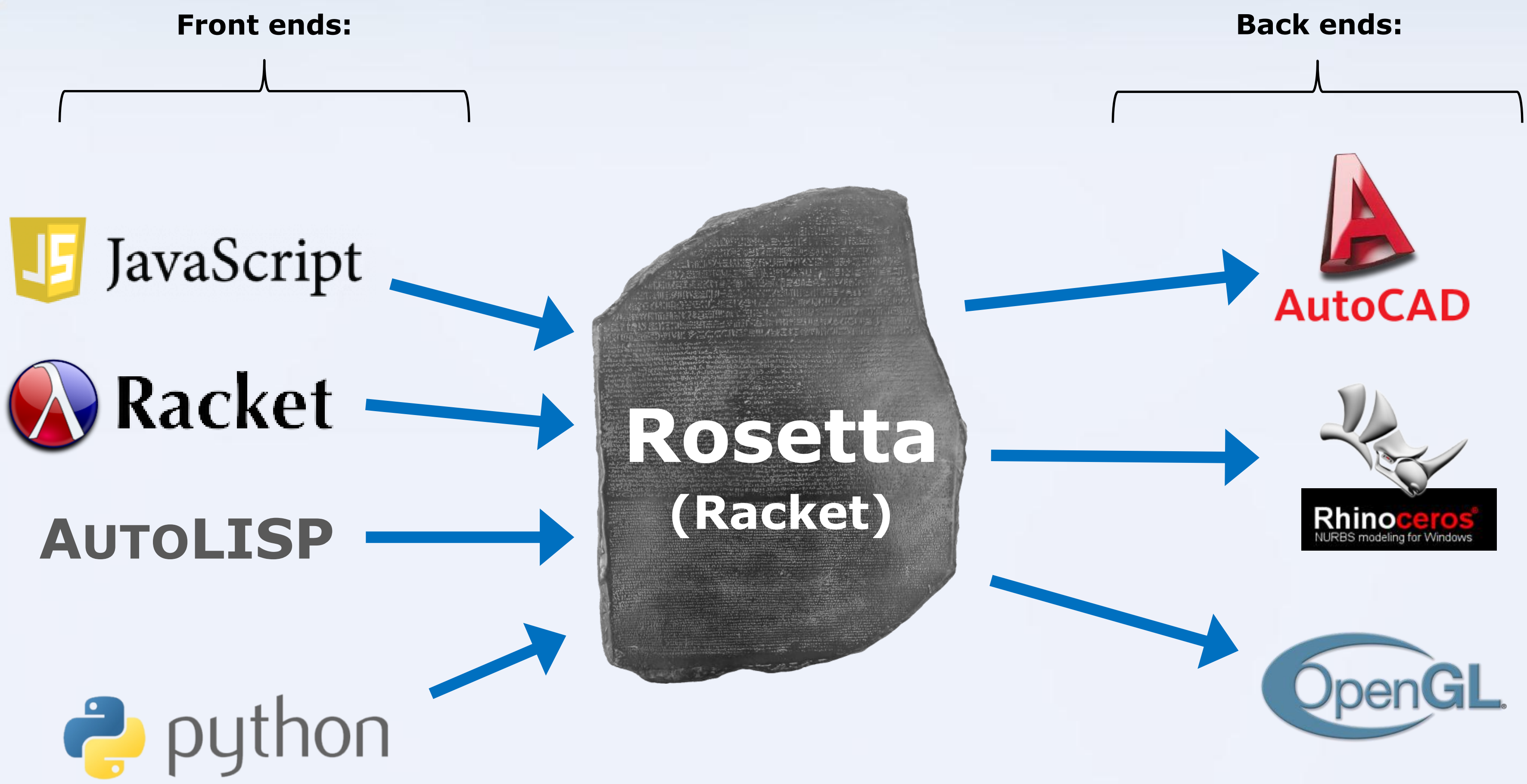
Why Python?

Python is replacing Scheme in introductory programming courses

Rosetta IDE


p.rkt - DrRacket
File Edit View Language Racket Insert Tabs Help
p.rkt (define ...) Check Syntax Debug Macro Stepper Run Stop
#lang racket
(require rosetta)
(backend "rhino")
(define (randomize-sphape d1 d2 sr shape-fn)
(let* ((r (random-interval d1 d2))
(th (random-interval 0 (* pi 2)))
(fi (random-interval 0 pi))
(p (sph r th fi)))
(move p (shape-fn (- sr r) p))))
(define (shape-cloud d1 d2 r n shape-fn)
(for/list ((i (range n)))
(randomize-sphape d1 d2 r shape-fn)))
(shape-cloud
4
5
5
600
(λ (r n) (box-normal r r r n)))
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>
#<rhino-shape>)
>
Determine language from source
604:2
Untitled - Rhinoceros (Evaluation) - [Top]
File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Help
Creating meshes... Press Esc to cancel
Command:
Top
CPlane x 1.743 y 5.147 z 0.000 Default Snap Ortho

Rosetta IDE


Front ends:
Back ends:
JavaScript
Racket
AUTOLISP
python
Rosetta
(Racket)
AutoCAD
Rhinoceros
NURBS modeling for Windows
OpenGL

- Borrows influences from Lisp
- High level, dynamically typed, GC’d
- Multiple paradigms
- Huge standard library + third-party libraries

Goals

- Correctness + Completeness
- Performance
- DrRacket Integration
- Interoperability with Racket

Related implementations

	Language(s) written	Platform(s) targetted	Speedup (vs. CPython)	Std. library support
CPython	C	CPython's VM	1x	Full

Related implementations

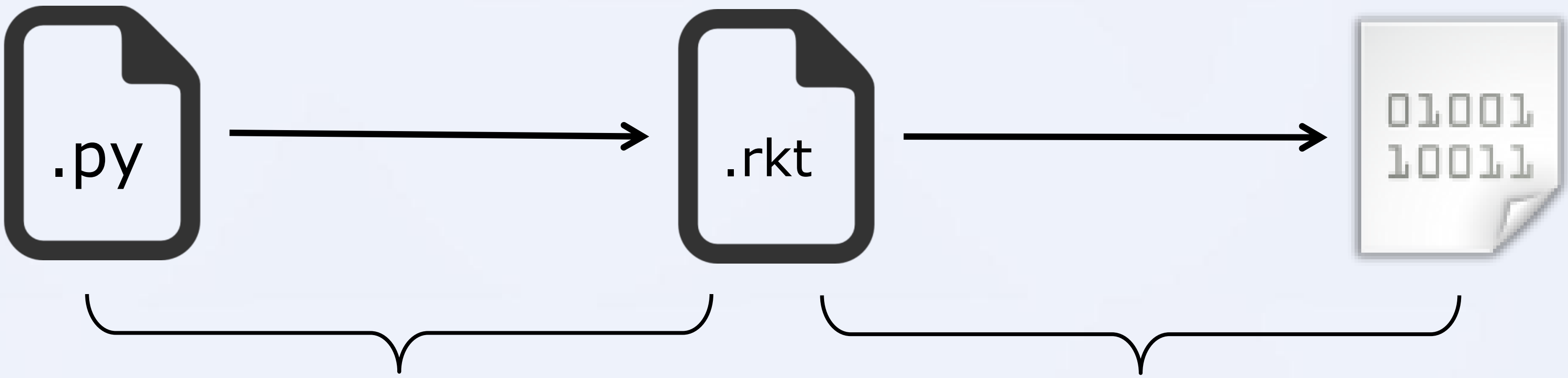
	Language(s) written	Platform(s) targetted	Speedup (vs. CPython)	Std. library support
CPython	C	CPython's VM	1x	Full
Jython	Java	JVM	~1x	Most
IronPython	C#	CLI	~1.8x	Most
CLPython	Common Lisp	Common Lisp	~0.5x	Most

Related implementations

	Language(s) written	Platform(s) targetted	Speedup (vs. CPython)	Std. library support
CPython	C	CPython's VM	1x	Full
Jython	Java	JVM	~1x	Most
IronPython	C#	CLI	~1.8x	Most
CLPython	Common Lisp	Common Lisp	~0.5x	Most
PLT Spy	Scheme, C	Scheme	~0.001x	Full

Our solution…

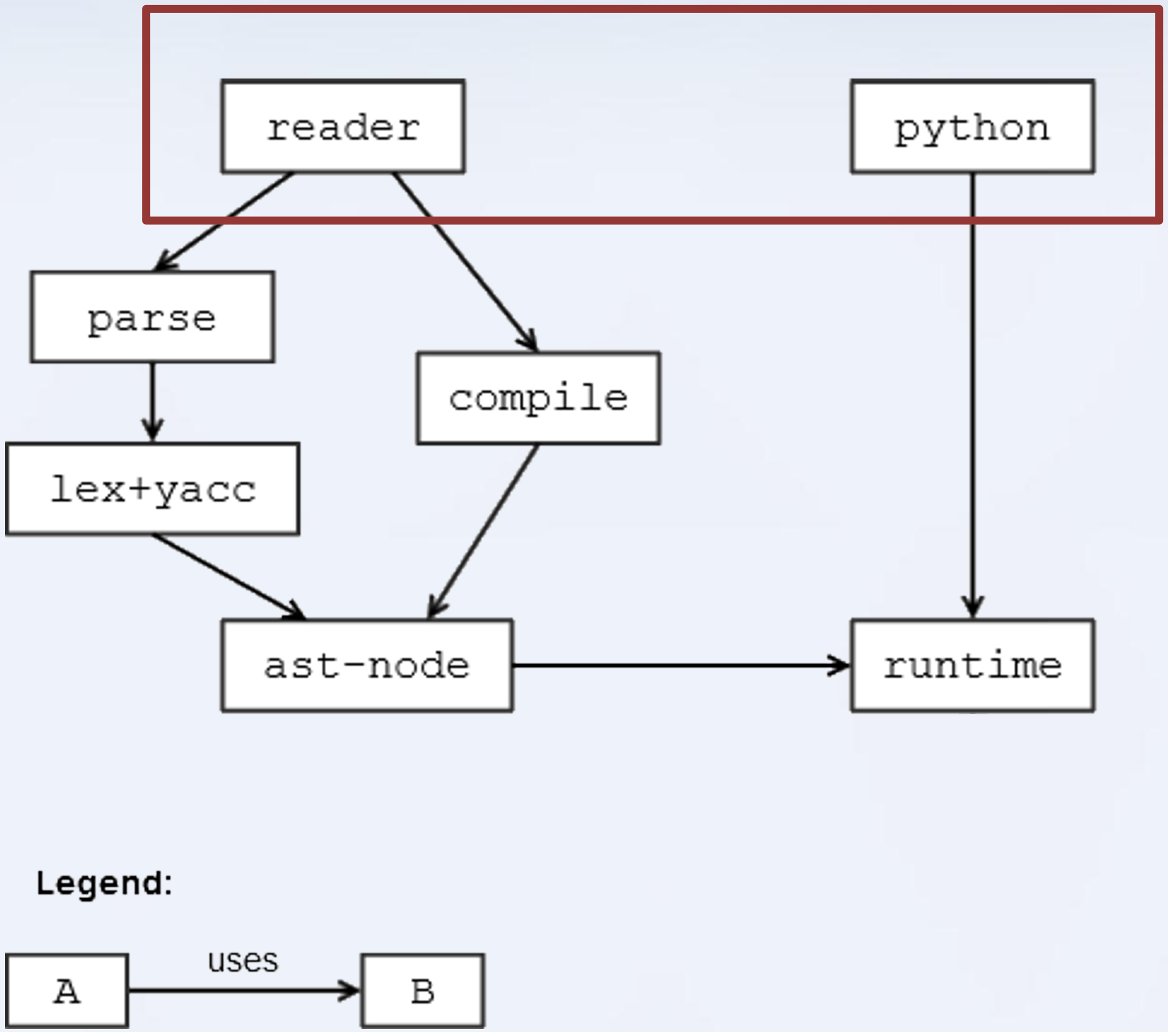
Pipeline



Python → Racket
(source-to-source compiler)

Racket → Bytecode
(Racket compiler + JIT)

Architecture



Racket Modules

- `reader` module (for compilation)
  - `read`: input-port → (listof s-expression)
  - `read-syntax`: input-port → (listof $\boxed{\text{syntax-object}}$) ?
- `python` module (for runtime behaviour)
  - Provides functions/macros used in compiled code

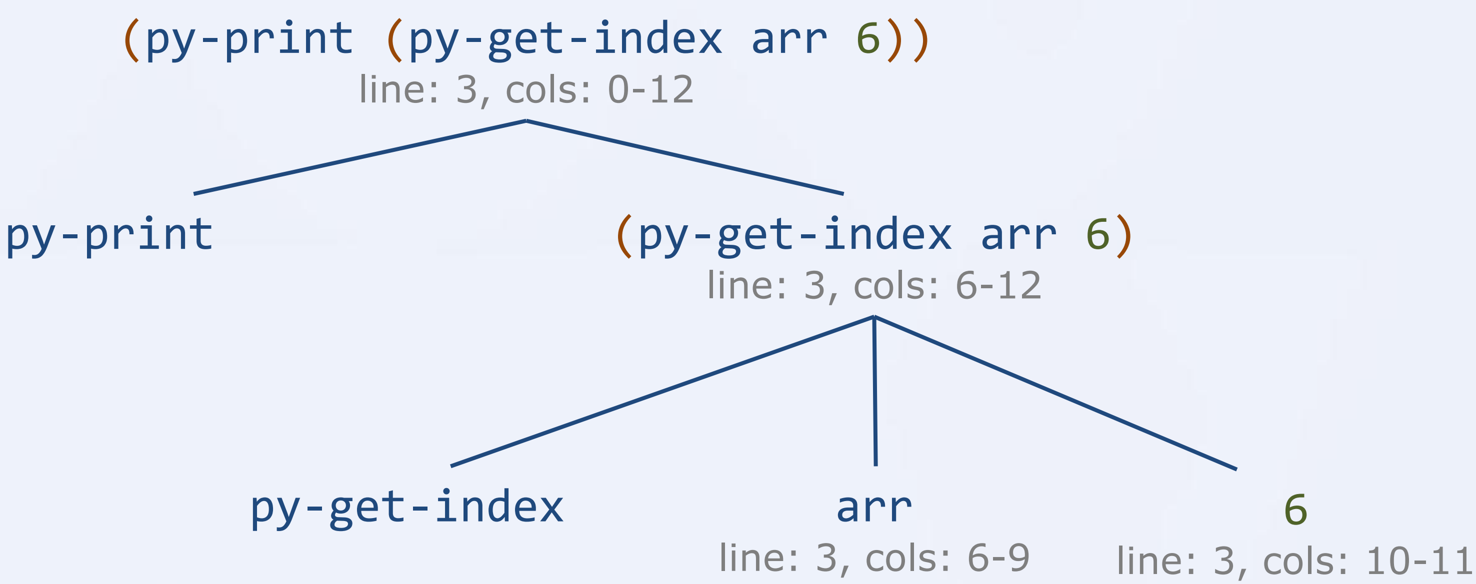
Syntax-objects

- S-expression
- Source location information
  - File, line number, column number, span
- Lexical-binding information

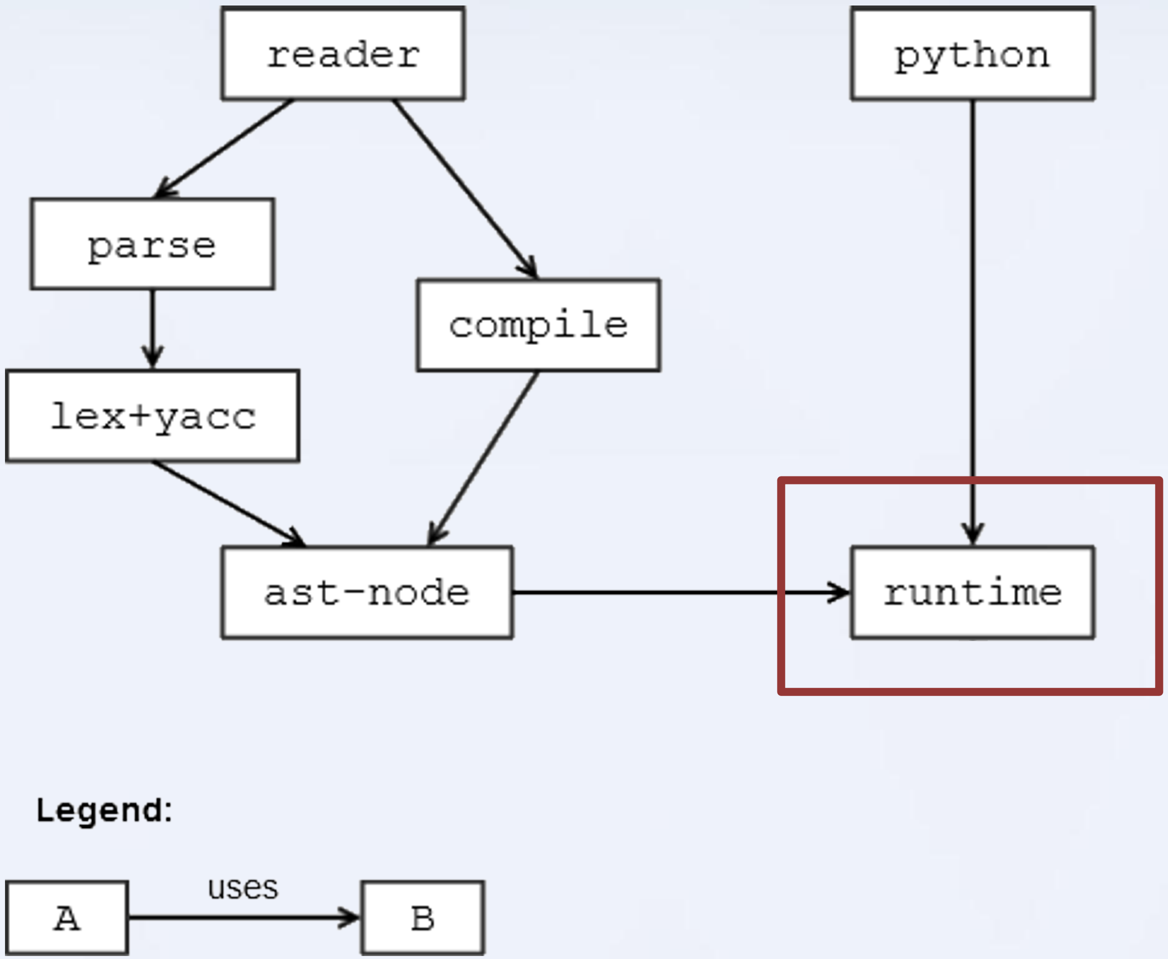
Syntax-objects

```
#lang python
arr = [1,1,2,3,5,8,13,21]
print arr[6]
```

↓



Architecture



How to implement Python's behaviour?

Runtime implementation

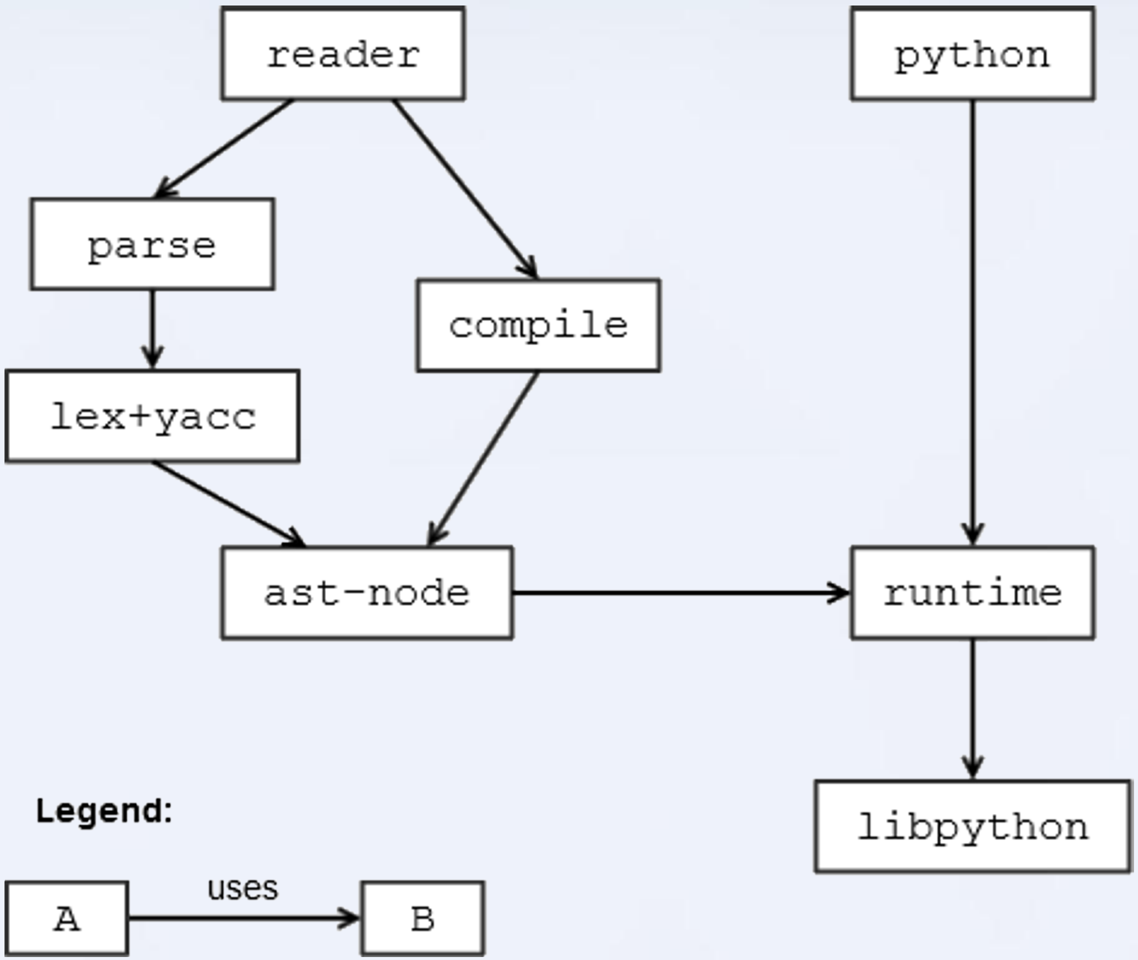
Two alternatives:

- Mapping to Python/C API

  (via Racket Foreign Function Interface)

- Racket reimplementation

Architecture



FFI Approach

libpython module

Racket

Racket FFI
(Foreign Function Interface)

foreign calls
on C pointers

CPython VM

Python/C API

FFI Runtime - Example

```
x + y
```

```
(define (py-add x y)
  (PyObject_CallObject (PyObject_GetAttrString x "__add__")
                       (make-py-tuple y)))


(define (make-py-tuple . elems)
    (let ([py-tuple (PyTuple_New (length elems))])
      (for ([i (range (length elems))]
            [elem elems])
        (PyTuple_SetItem py-tuple i elem))
      py-tuple))
```

FFI Runtime - Disadvantages

- Bad Performance
  - Expensive type conversions + FFI calls
  - Finalizers for GC
- Clumsy Interoperability with Racket
  - Wrappers/Unwrappers

What about implementing it over Racket data types?

We must first understand Python’s data model

Python’s Data Model

- Every value is an object
- Every object has a reference to its *type-object*
- Type-objects hold hash-table for method dispatching
  - Maps method names to function objects
- Operator behaviour is mapped to methods

Optimizations

- Basic types mapped to Racket types
  - int, long, float, complex, string, dict
  - Avoids wrapping/unwrapping
- Early method dispatching for operators
  - Avoids expensive method dispatching for common uses

Racket Runtime - Example

```
x + y
```

```
(define (py-add x y)
  (py-method-call x "__add__" y))
```

↓

```
(define (py-add x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (string? x) (string? y)) (string-append x y)]
    [else (py-method-call x "__add__" y)]))
```

How are modules imported?

Python import system

- `import <module>`
  - <module> is imported as a module object
- `from <module> import <id>`
  - <id> is imported as a new binding
- `from <module> import *`
  - All bindings from <module> are imported

`module->exports`
+
`dynamic-require`

`require`

- Special syntax for Racket imports

Import - Example

```
#lang python
import "racket" as rkt

def add_cons(c):
  return rkt.car(c) + rkt.cdr(c)

c1 = rkt.cons(2, 3)
c2 = rkt.cons("abc", "def")
```

```
> add_cons(c1)
5
> add_cons(c2)
"abcdef"
```

Import - Example

```
#lang python
from "racket" import cons, car, cdr

def add_cons(c):
  return car(c) + cdr(c)

c1 = cons(2, 3)
c2 = cons("abc", "def")
```

```
> add_cons(c1)
5
> add_cons(c2)
"abcdef"
```

Import - Example (Macros)

```
#lang python
from "racket/trace" import trace

def factorial(n):
  if n == 0: return 1
  else: return n * factorial(n-1)

trace(factorial)
```

```
> factorial(5)
>(factorial 5)
> (factorial 4)
> >(factorial 3)
> > (factorial 2)
> > >(factorial 1)
> > > (factorial 0)
< < < 1
< < <1
< < 2
< <6
< 24
<120
120
```

Other Features

- Class definitions
  - `class` statement → new type object
- Exception handling
  - `raise`, `try...except` statements → `raise`, `with-handlers` forms
- Flow control statements
  - `return`, `break`, `continue`, `yield` → escape continuations

Benchmarks

- Ackermann
  - computing the Ackermann function

- Mandelbrot
  - computing if a complex sequence diverges after a limited number of iterations

Ackermann

```
(define (ackermann m n)
  (cond
    [(= m 0) (+ n 1)]
    [(and (> m 0) (= n 0)) (ackermann (- m 1) 1)]
    [else (ackermann (- m 1) (ackermann m (- n 1)))]))

(ackermann 3 9)
```

```
def ackermann(m,n):
    if m == 0: return n+1
    elif m > 0 and n == 0: return ackermann(m-1,1)
    else: return ackermann(m-1, ackermann(m,n-1))

print ackermann(3,9)
```

Mandelbrot

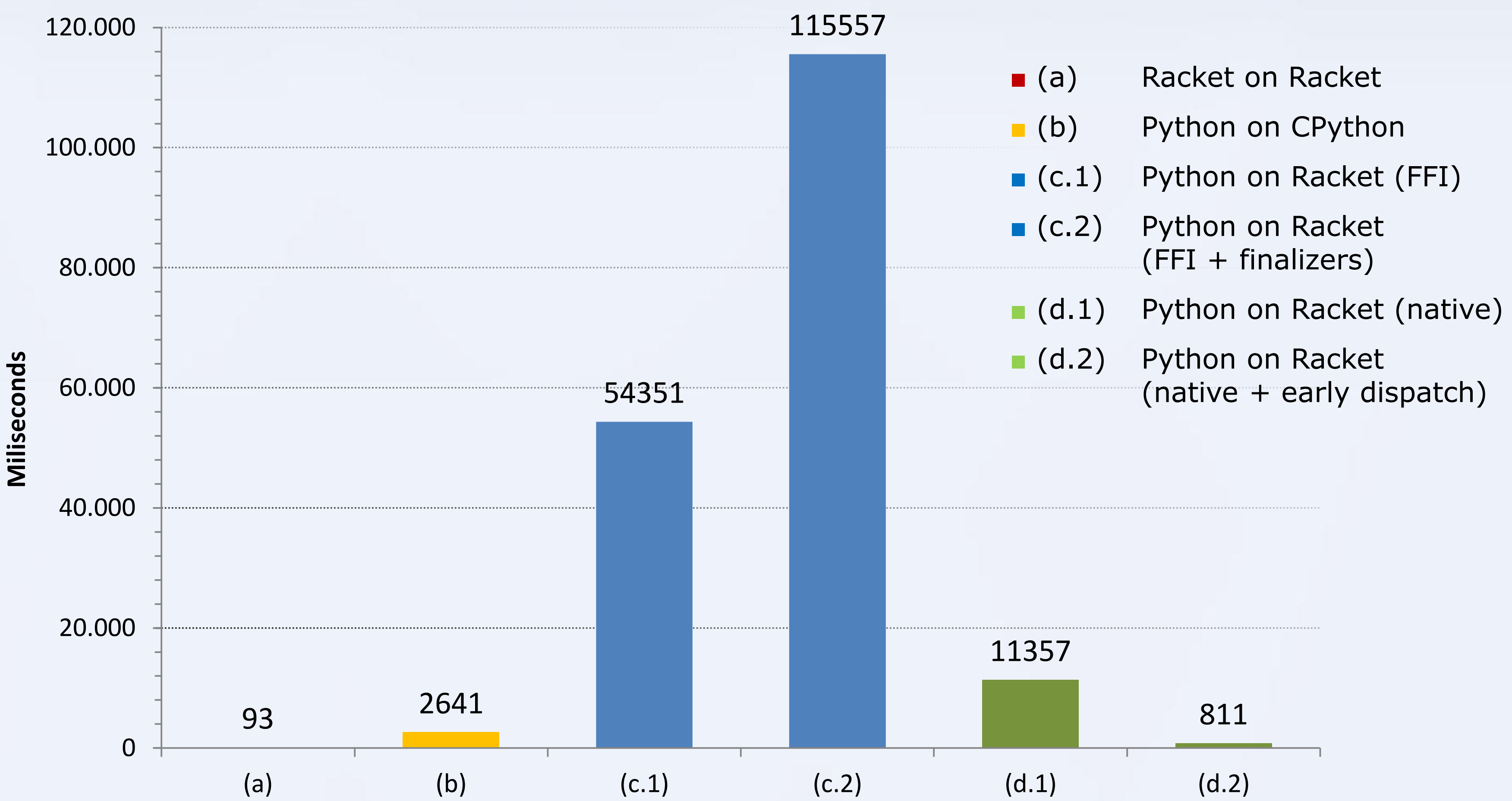
```racket
(define (mandelbrot limit c)
  (let loop ([i 0] [z 0+0i])
    (cond
      [(> i limit) i]
      [(> (magnitude z) 2) i]
      [else (loop (add1 i)
                  (+ (* z z) c))])))

(mandelbrot 1000000 .2+.3i)
```

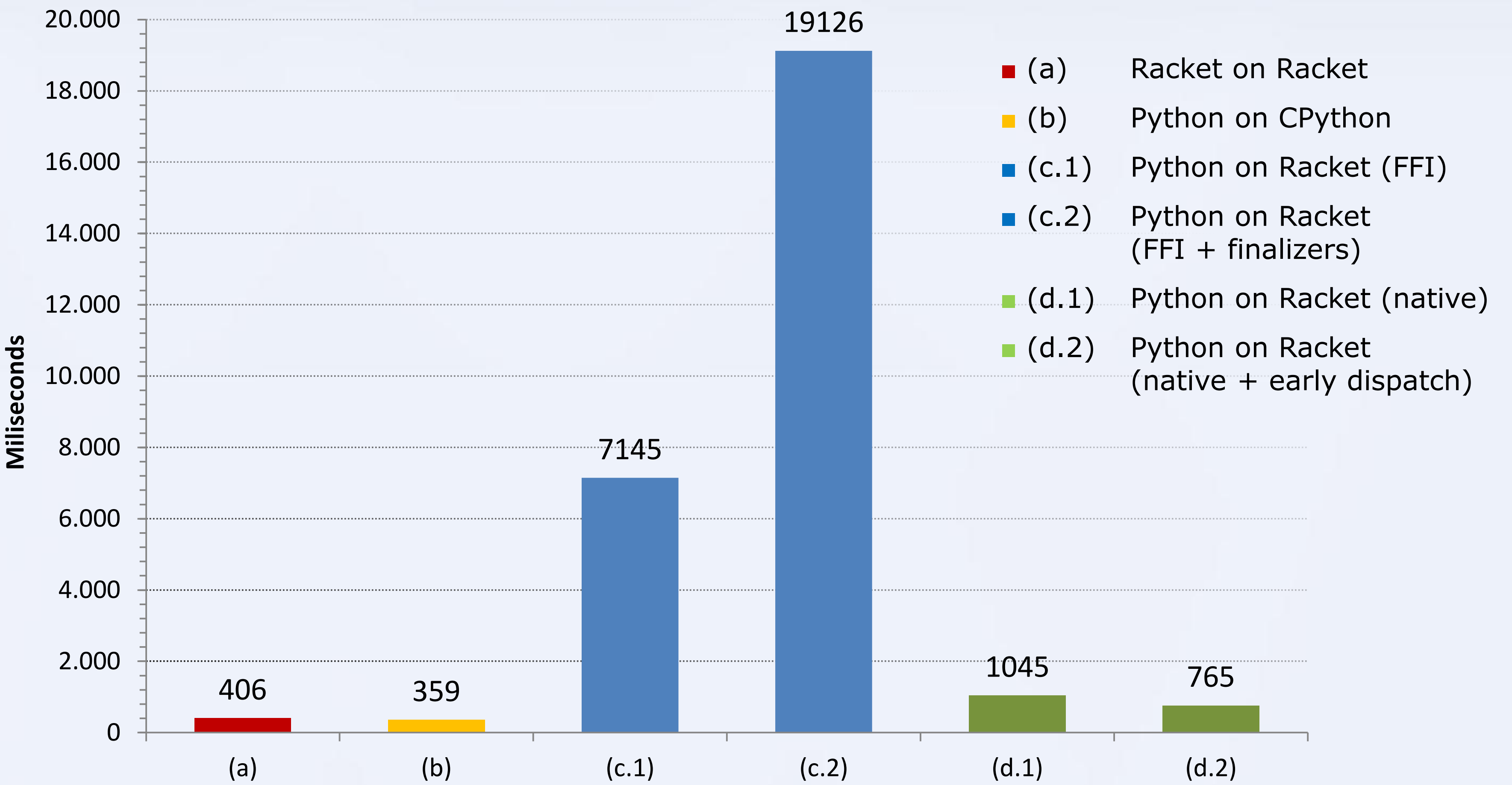
```python
def mandelbrot(limit, c):
    z = 0+0j
    for i in range(limit+1):
        if abs(z) > 2: return i
        z = z*z + c
    return i+1

print mandelbrot(1000000, .2+.3j)
```

Ackermann - Results



Mandelbrot - Results



Future Work

- Fully implement compilation process
- Implement behaviour for built-in types
- Integrate FFI calls with current data model
- Formal testing for correctness

Thank you for listening!

Questions? Comments?