Tracing Comes To Racket!

Spenser Bauman[1] Carl Friedrich Bolz[2] Robert Hirschfeld[3]
Vasily Kirilichev[3] Tobias Pape[3] Jeremy G. Siek[1]
Sam Tobin-Hochstadt[1]

[1]Indiana University Bloomington, USA

[2]King's College London, UK

[3]Hasso-Plattner-Institut, University of Potsdam, Germany

RacketCon
September 27th 2015

```
(define/contract (dot-safe v1 v2)
 ((vectorof flonum?) (vectorof flonum?) . -> . flonum?)
 (for/sum ([e1 v1] [e2 v2]) (* e1 e2)))

        (time (dot-safe v1 v2)) ;; 8888 ms
```

```
(define (dot v1 v2)
  (for/sum ([e1 v1] [e2 v2])
    (* e1 e2)))

(time (dot v1 v2)) ;; 3864 ms
```

```
(define (dot2 v1 v2)
  (for/sum ([e1 (in-vector v1)]
            [e2 (in-vector v2)])
    (fl* e1 e2)))

(time (dot2 v1 v2)) ;; 984 ms
```

```
(define (dot-fast v1 v2)
  (define len (flvector-length v1))
  (unless (= len (flvector-length v2))
    (error 'fail))
  (let loop ([n 0] [sum 0.0])
    (if (unsafe-fx= len n) sum
        (loop (unsafe-fx+ n 1)
              (unsafe-fl+ sum (unsafe-fl*
                               (unsafe-flvector-ref v1 n)
                               (unsafe-flvector-ref v2 n)))))))
```

```
(time (dot-fast v1 v2)) ;; 268 ms
```

Pycket is a tracing JIT compiler which significantly reduces the need for manual specialization

```
(time (dot-safe v1 v2)) ;; 95 ms vs 8888 ms
(time (dot v1 v2))      ;; 74 ms vs 3864 ms
(time (dot2 v1 v2))     ;; 78 ms vs 984  ms
(time (dot-fast v1 v2)) ;; 74 ms vs 268  ms
                        ;;-----------------
                        ;; 20 %  vs 3000 %
```

Idea: Apply dynamic language JIT compiler to Racket

Take: Racket

+

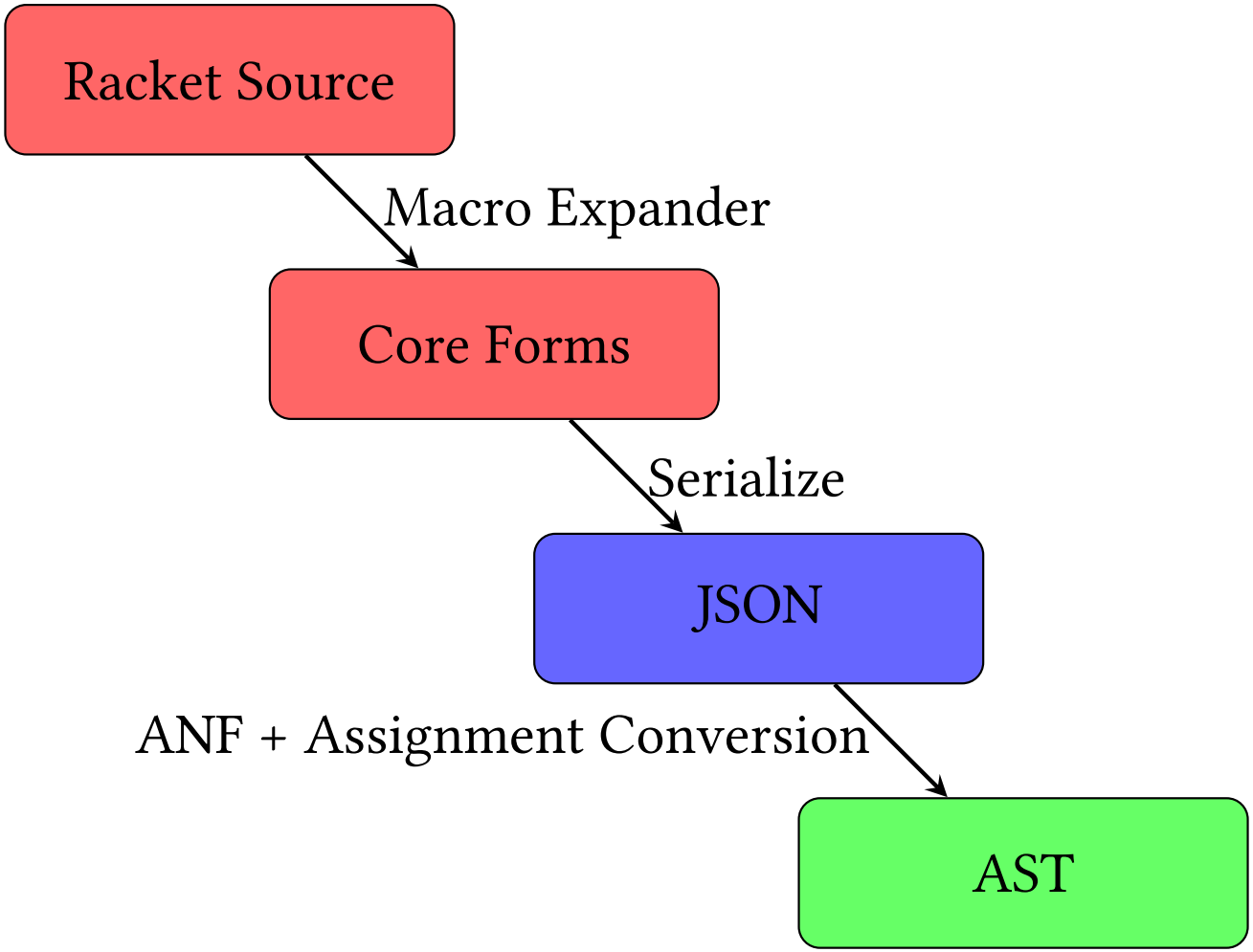
Apply: RPython Project

pypy

=

Design

Bootstrapping Process



CEK Machine

Use CEK machine to interpret Racket's core forms

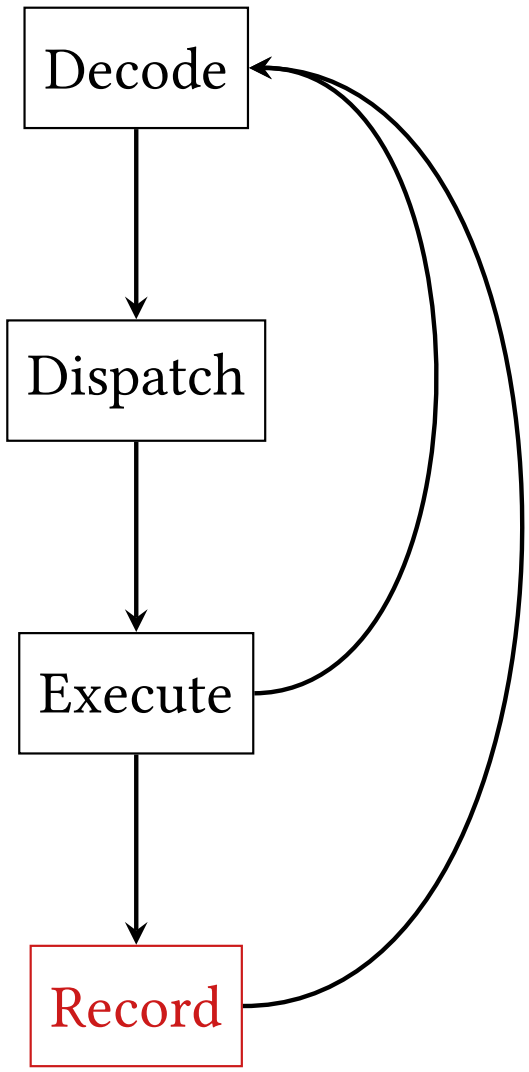
```
(define (app-κ exp Γ κ)
  (match κ
    [`()                                exp]
    [`(arg ,e ,Γ^ ,κ^)                  `(,e ,Γ^ (fun ,exp ,κ^))]
    [`(fun (proc ,x ,e ,Γ^) ,κ^) `(,e ,(ext x exp Γ^) ,κ^)]))

(define (step exp Γ κ)
  (match exp
    [x #:when (symbol? x) `(,(lookup x Γ) ,Γ ,κ)]
    [`(lambda (,x) ,e)    (app-κ `(proc ,x ,e ,Γ) Γ κ)]
    [`(,e1 ,e2)           `(,e1 ,Γ (arg ,e2 ,Γ ,κ))]
    [x                    (app-κ x Γ κ)]))
```

Trace Compilation

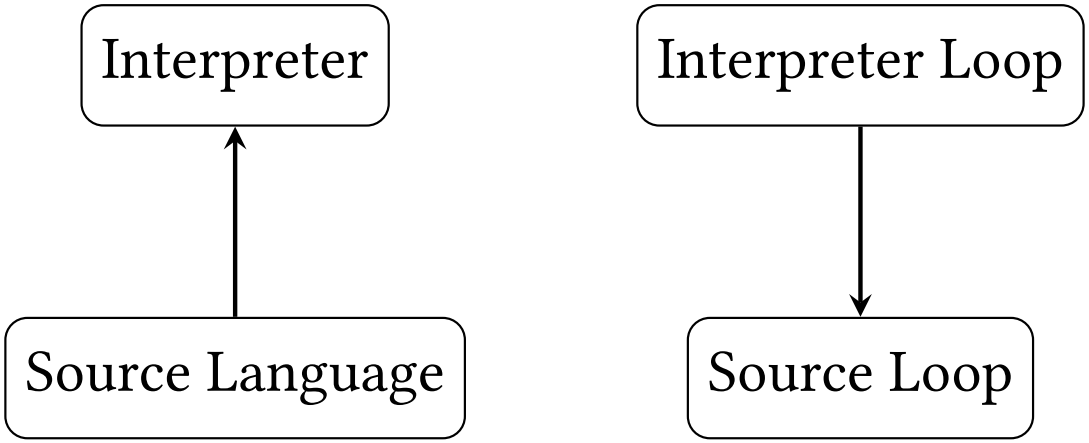
Trace Compilation

1. Instrument dispatch loop
2. Record emulation instructions during interpretation
3. Generate code for recorded instruction sequence
4. Bail to interpreter when control flow diverges



Meta-Tracing

Meta-tracing: Trace the interpreter rather than the source language



[Bolz, Cuni, Fijałkowski, Rigo 2009]

Dot in Detail

Unit of compilation = loops

`dot` becomes

```
label(acc, idx1, idx2, len1, len2, arr1, arr2)
# Bail to interpreter if false
guard(idx1 < len1)
guard(idx2 < len2)
val1     = getarrayitem_gc(arr1, idx1)
val2     = getarrayitem_gc(arr2, idx2)
prod     = val1 * val2
acc_new  = acc + prod
idx1_new = idx1 + 1
idx2_new = idx2 + 1
jump(acc_new, idx1_new, idx2_new, len1, len2, arr1, arr2)
```

Functionality

What Works?

- File IO

```
(open-input-file "list.txt")
(open-output-file "brain.dat")
```

- Numeric tower

```
number? complex? real? rational? integer? ...
```

- Contracts

```
(define-contract ...)
```

- Typed Racket

```
#lang typed/racket
```

- Primitve Function ($\sim 900/1400$)

What Doesn't Work?

- FFI
- Scribble

```
#lang scribble/base
```

- DrRacket
- Web

```
#lang web-server/insta
```

- Threads

```
(thread (λ () ...))
```

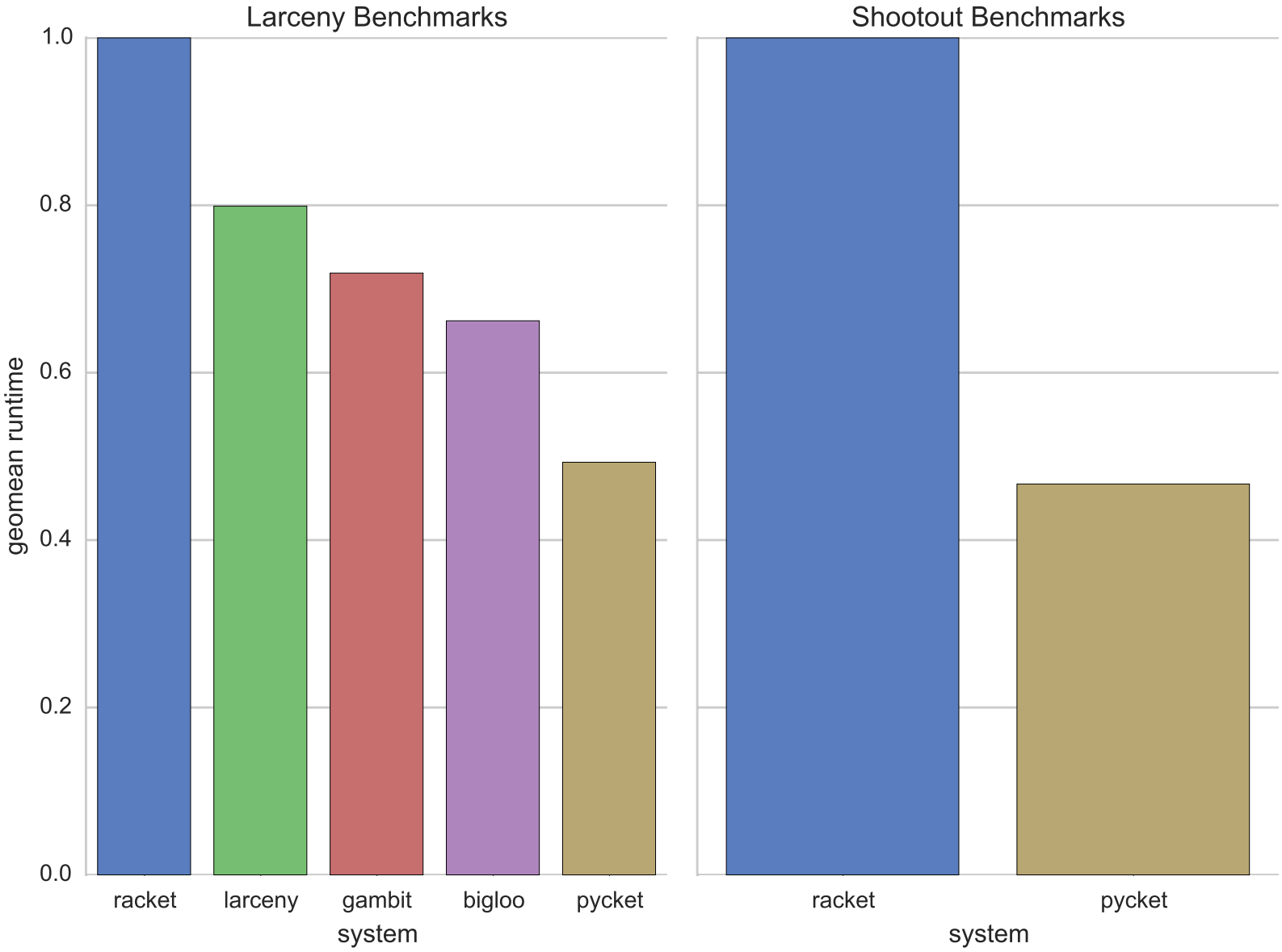
- Lesser used primitives

Performance Caveats

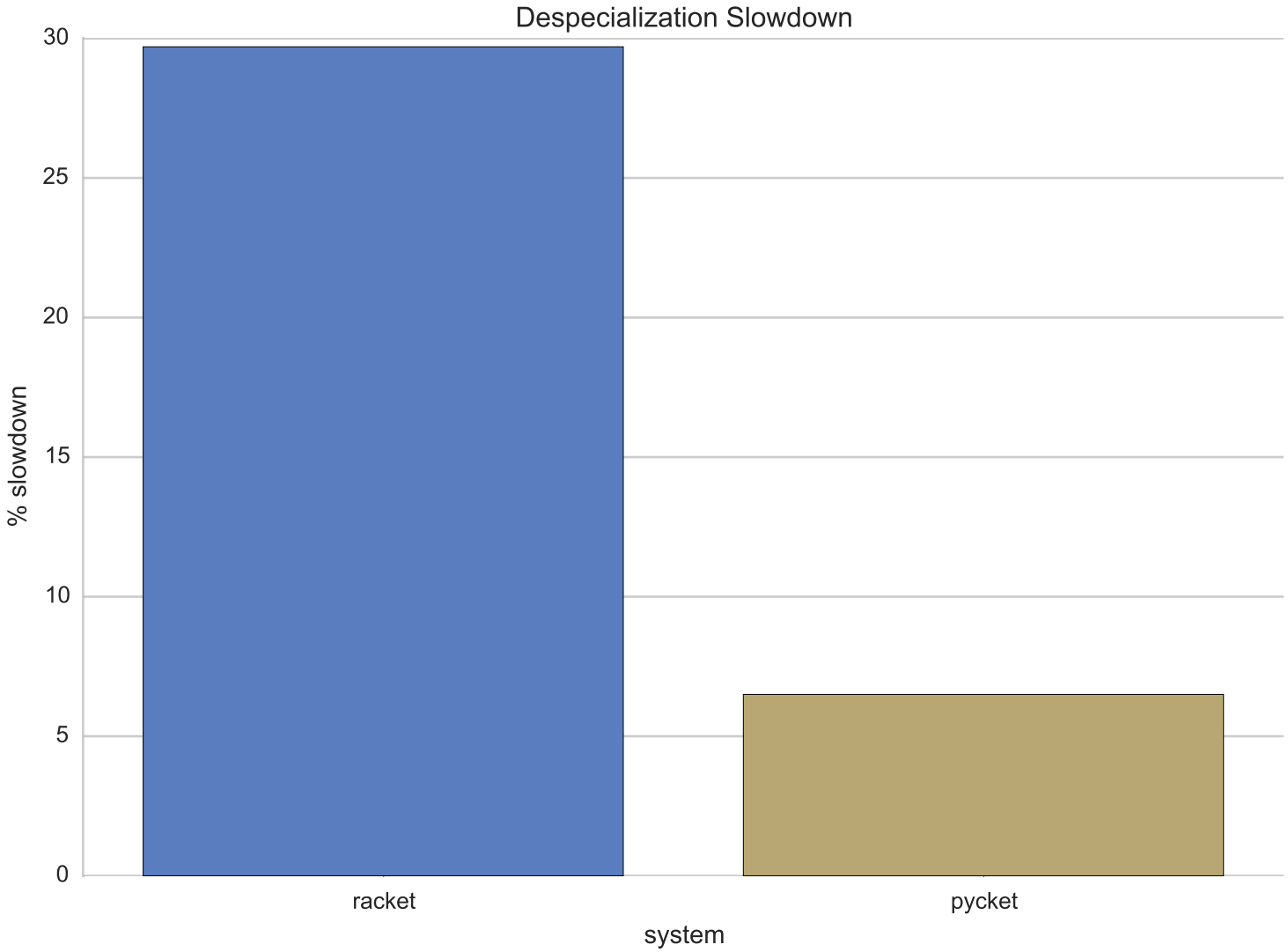
Fast	Slow
Straight loops	Branchy/irregular control flow
Numeric Computations	Code not easily expressed as loops
	Interpreters

Benchmarks

Overall Performance



Specialization



Future Improvements

- Improve chaperone/impersonator performance and space usage
- Explore interaction between ahead-of-time and just-in-time optimizations
- Green threads and inter-thread optimizations
- Improve performance on complicated control flow
- Support more of Racket

Thank You

- Performance competitive Racket implementation
- Small amount of code ($\sim$ 14000 LOC)
- High level interpreter implementation
- Supports a large subset of Racket

https://github.com/samth/pycket