# From Racket to GnoSys

## Racket is already

- expressive
- extensible
- performant
- reliable
- cross-platform

## Languages

```
#lang racket

(define (twice f x)
  (f (f x)))
```

## Languages

```
#lang typed/racket

(: twice : (All (A) (A -> A) A -> A))
(define (twice f x)
  (f (f x)))
```

## Languages

```
#lang typed/racket

(: twice : (All (A) (A -> A) A -> A))
(define (twice f x)
  (f (f x)))
```

Integrating Static Semantics with Optimization

## Languages

```
#lang lazy

(define (twice f x)
  (f (f x)))
```

## Languages

```
#lang lazy

(define (twice f x)
  (f (f x)))
```

Optimzied Extensible Semantics

## Languages

```
#lang web-server

(define (twice f x)
  (f (f x)))
```

## Languages

```
#lang web-server

(define (twice f x)
  (f (f x)))
```

Static Semantics for Code Transformation

## Languages

```
#lang datalog

parent(john, douglas)
ancestor(A, B) :-
  parent(A, B)
ancestor(A, B) :-
  parent(A, C),
  ancestor(C, B)
```

## Languages

```
#lang datalog

parent(john, douglas)
ancestor(A, B) :-
  parent(A, B)
ancestor(A, B) :-
  parent(A, C),
  ancestor(C, B)
```

Restricted Languages for Improved Security

## Racket

```
#lang racket

(require net/url net/uri-codec)

;  let-me-google-that-for-you : string  -> [listof bytes]
(define (let-me-google-that-for-you query)
  (define base "http://www.google.com/search?q=")
  (define url (string->url
                 (string-append base (uri-encode query))))
  (define rx #rx"(?<=<h3 class=\"r\">).*?(?=</h3>)")
  (regexp-match* rx (get-pure-port url)))
```

## Racket

```
#lang racket

(require net/url net/uri-codec)

;  let-me-google-that-for-you : string  -> [listof bytes]
(define (let-me-google-that-for-you query)
  (define base "http://www.google.com/search?q=")
  (define url (string->url
               (string-append base (uri-encode query))))
  (define rx #rx"(?<=<h3 class=\"r\">).*?(?=</h3>)")
  (regexp-match* rx (get-pure-port url)))
```

## Racket with Contracts

```
#lang racket

(require net/url net/uri-codec)
(provide/contract
  [let-me-google-that-for-you  (string? -> [listof bytes?])])
(define (let-me-google-that-for-you query)
  (define base "http://www.google.com/search?q=")
  (define url (string->url
               (string-append base (uri-encode query))))
  (define rx #rx"(?<=<h3 class=\"r\">).*?(?=</h3>)")
  (regexp-match* rx (get-pure-port url)))
```

## Racket with Contracts

```
#lang racket

(require net/url net/uri-codec)
(provide/contract
  [let-me-google-that-for-you  (string? -> [listof bytes?])])
(define (let-me-google-that-for-you query)
  (define base "http://www.google.com/search?q=")
  (define url (string->url
               (string-append base (uri-encode query))))
  (define rx #rx"(?<=<h3 class=\"r\">).*?(?=</h3>)")
  (regexp-match* rx (get-pure-port url)))
```

Static contract validation

## Typed Racket

```
#lang typed/racket

(require typed/net/url typed/net/uri-codec)

(: let-me-google-that-for-you : String -> (Listof Bytes))
(define (let-me-google-that-for-you query)
  (define base "http://www.google.com/search?q=")
  (define url (string->url
               (string-append base (uri-encode query))))
  (define rx #rx"(?<=<h3 class=\"r\">).*?(?=</h3>)")
  (regexp-match* rx (get-pure-port url)))
```

## Typed Racket

```
#lang typed/racket

(require typed/net/url typed/net/uri-codec)

(: let-me-google-that-for-you : String -> (Listof Bytes))
(define (let-me-google-that-for-you query)
  (define base "http://www.google.com/search?q=")
  (define url (string->url
               (string-append base (uri-encode query))))
  (define rx #rx"(?<=<h3 class=\"r\">).*?(?=</h3>)")
  (regexp-match* rx (get-pure-port url)))
```

Types and Little Languages

## DSLs for Language Specification

```
(define-syntax (let stx)
  (syntax-parse stx
    [(let bs:distinct-bindings body:expr)
     #'((λ (bs.var ...) body) bs.rhs ...)]))
```

## DSLs for Language Specification

```
(define-syntax (let stx)
  (syntax-parse stx
    [(let bs:distinct-bindings body:expr)
     #'((λ (bs.var ...) body) bs.rhs ...)]))
```

Automated Semantic Tools

## High-level Operating Systems

```
(define (run-bounded thunk timeout)
  (define user-cust (make-custodian))
  (parameterize ([current-custodian user-cust])
    (thread thunk))
  (sleep timeout)
  (custodian-shutdown-all user-cust))
```

## High-level Operating Systems

```
(define (run-bounded thunk timeout)
  (define user-cust (make-custodian))
  (parameterize ([current-custodian user-cust])
    (thread thunk))
  (sleep timeout)
  (custodian-shutdown-all user-cust))
```

Semantics-based resource control