Pico: Scheme for Mere Mortals

1st European Lisp and Scheme Workshop

13 June 2004

Theo D'Hondt – Wolfgang De Meuter – Jessie Dedecker

Programming Technology Lab
Computer Science Department
Faculty of Sciences
Vrije Universiteit Brussel

Timeline

‘98	intro to programming
‘99	virtual machines
‘00	mobility & migration
‘02	prototypes
‘03	virtual^2 machines

Scheme vs. Pico

```scheme
(define (QuickSort V Low High)
  (define Left Low)
  (define Right High)
  (define Pivot (vector-ref V (quotient (+ Left Right) 2 )))
  (define Save 0)
  (do ((stop #f (> Left Right))) (stop)
    (do () ((>= (vector-ref V Left) Pivot))
      (set! Left (+ Left 1)))
    (do () ((<= (vector-ref V Right) Pivot))
      (set! Right (- Right 1)))
    (if (<= Left Right)
        (begin
          (set! Save (vector-ref V Left))
          (vector-set! V Left (vector-ref V Right))
          (vector-set! V Right Save)
          (set! Left (+ Left 1))
          (set! Right (- Right 1)))))
  (if (< Low Right) (QuickSort V Low Right))
  (if (> High Left) (QuickSort V Left High)))
```

special forms

lambda's

syntax

...

∀ functions

∀names

canonical

...

```
QuickSort(V, Low, High):
 { Left: Low;
   Right: High;
   Pivot: V[(Left + Right) // 2];
   Save: 0;
   until(Left > Right,
     { while(V[Left] < Pivot, Left:= Left+1);
       while(V[Right] > Pivot, Right:= Right-1);
       if(Left <= Right,
         { Save:= V[Left];
           V[Left]:= V[Right];
           V[Right]:= Save;
           Left:= Left+1;
           Right:= Right-1 }) });
   if(Low < Right, QuickSort(V, Low, Right));
   if(High > Left, QuickSort(V, Left, High)) }
```

Pico basics

- minimal&regular syntax
- infix operators
- tables everywhere
- first-class everything
- call-by-name
- abstract syntax

Pico basics

- minimal&regular syntax

variable	tabulation	application	
`x` variable/constant reference	`t[idx]` table indexing	`f(1, x)` function call	invocation
`v: 123` variable definition	`t[10]: x()` variable table definition	`f(x): x+x` variable function definition	invocation: expression
`c:: 123` constant definition	`t[10]:: y()` constant table definition	`f(x):: x*x` constant function definition	invocation:: expression
`v:= 123` variable assignment	`t[10]:= 0` table modification	`f(x):= -x` function redefinition	invocation:= expression

Pico basics

- minimal&regular syntax
- infix operators
- tables everywhere
- first-class eve
- call-by-name
- abstract syntax

```
a++b: a+b+1
<function ++>
a**b: a*b*2
<function **>
p<=>q: abs(p-q)<1
<function <=>>
1++2<=>1**2
<native true>
```

Pico basics

```
counter():
  { count:0;
    counter():=
      count:=count+1;
    counter() }
<function counter>
tab[10]: counter()
<table>
display(tab)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

regular syntax

ators

- tables everywhere
- first-clas
- call-by-
- abstract

```
table@arguments: arguments
<function table>
begin@arguments: arguments[size(arguments)]
<function begin>
T: table(1,2,3,4,5)
<table>
display(T)
[1, 2, 3, 4, 5]
begin(X: 1, Y: 2, X+Y)
3
```

Pico basics

number
fraction
text
function
table
dictionary
continuation
void

- minimal ular syntax
- infix ope s
- tables ev here
- first-class everything
- call-by-name
- abstract syntax

basics

```
{ true(p, q())::
    p;
  false(p(), q)::
    q;
  if(p, c(), a())::
    p(c(), a());
  while(p(), e())::
    { loop(value, boolean)::
        boolean(loop(e(), p()), value);
      loop(void, p()) }}
```

- egular syntax
- infix operators
- tables eve
- first-class

```
map(filter(item), table):
  { index: 0;
    filtered_table[size(table)]:
      filter(table[index:= index+1]) }
<function map>
display(map(item^2, [1,2,3,5,7,11]))
[1, 4, 9, 25, 49, 121]
```

- call-by-name
- abstract syntax

Pico basics

```
<expression>    ::= <void> |  ... | <number>

<void>          ::= VOI
<reference>     ::= REF <name>
<application>   ::= APL <expression> <arguments>
<tabulation>    ::= TBL <expression> <indexation>
<declaration>   ::= DCL <invocation> <expression>
<definition>    ::= DEF <invocation> <expression>
<assignment>    ::= SET <invocation> <expression>
<constant>      ::= CST <name> <expression> <dictionary>
<variable>      ::= VAR <name> <expression> <dictionary>
<continuation>  ::= CNT <dictionary> <number> <number> <table>
<native>        ::= NAT <name> <number>
<function>      ::= FUN <name> <arguments> <expression> <dictionary>
<table>         ::= TAB <table>
<text>          ::= TXT <text>
<fraction>      ::= FRC <fraction>
<number>        ::= NBR <number>

<name>          ::= <text>
<indexation>    ::= <table>
<arguments>     ::= <table>
<arguments>     ::= <invocation>
<dictionary>    ::= <variable>
<dictionary>    ::= <constant>
<dictionary>    ::= <void>
<invocation>    ::= <reference>
<invocation>    ::= <application>
<invocation>    ::= <tabulation>
```

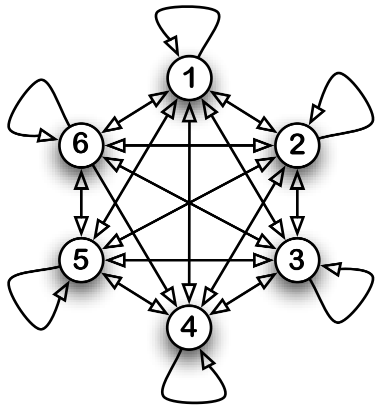
- m x
- in
- to
- fi
- c
- abstract syntax

Pico internals

- uniform memory + gc
- abstract grammar driven
- environments as lists
- thread/continuation style
- tail recursion
- smart caching

Pico internals

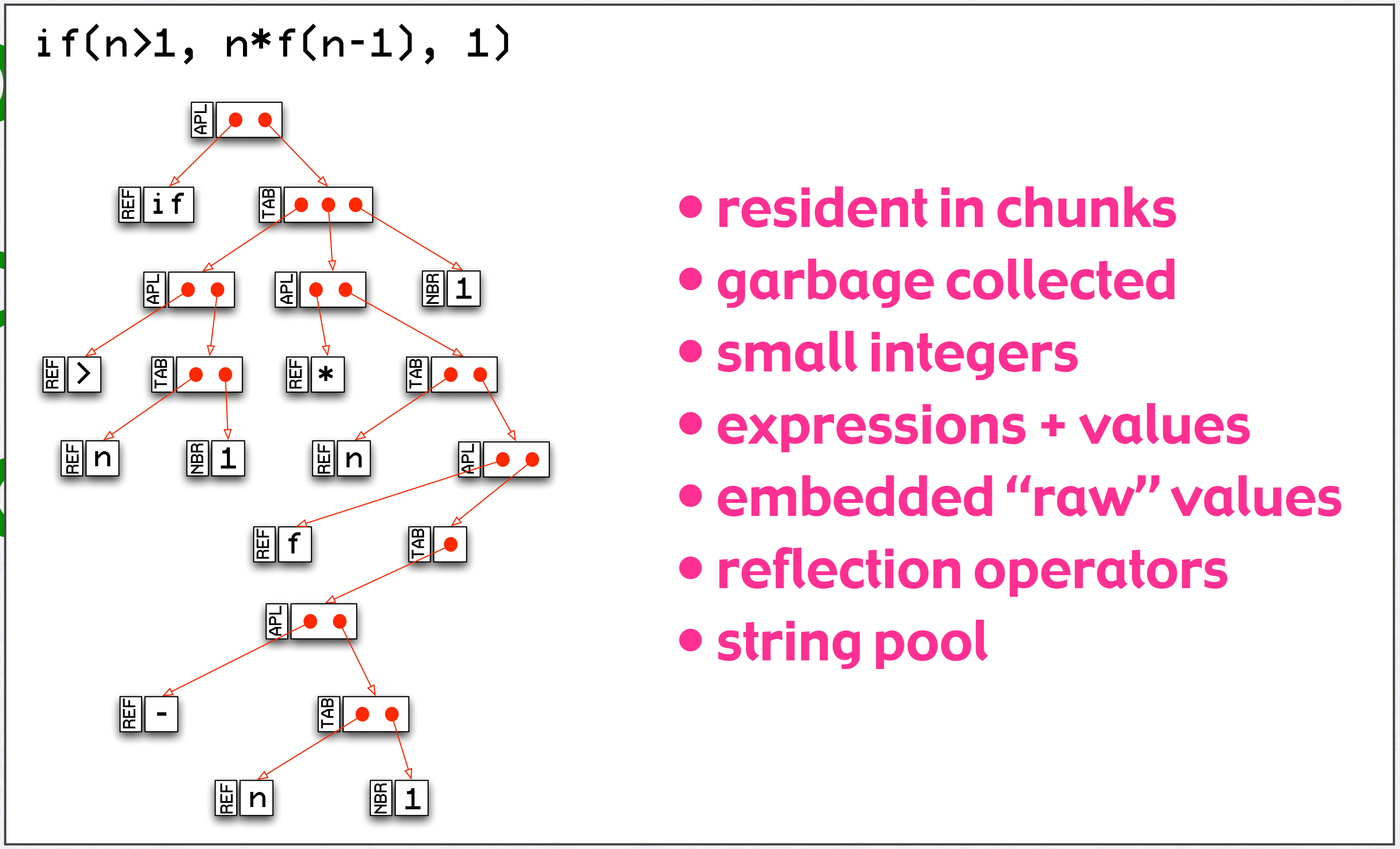
- uniform memory + gc
- abstract grammar driven
- environments as lists
- threa yle
- tail re
- smart caching



- variable-length chunks
- mark-and-compact gc
- tagged size-headers
- single bit per cell for gc
- programs -> chunks
- values -> chunks
- environments -> chunks
- threads -> chunks

Pico internals

- uniform memory + gc
- abstract grammar driven
- enviro
- threac
- tail rec
- smart



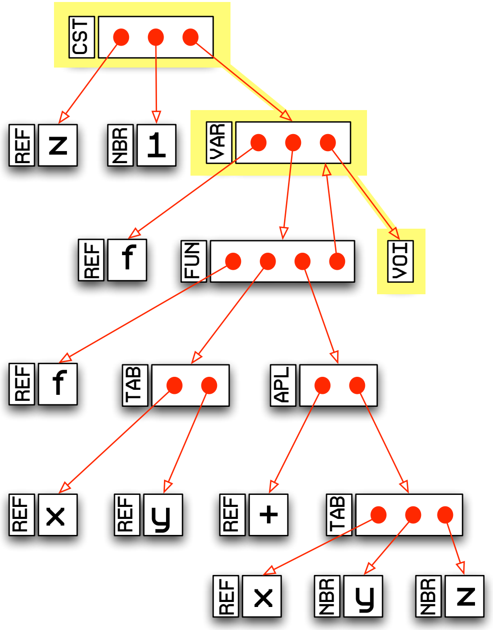
```
if(n>1, n*f(n-1), 1)
```

- resident in chunks
- garbage collected
- small integers
- expressions + values
- embedded “raw” values
- reflection operators
- string pool

Pico internals

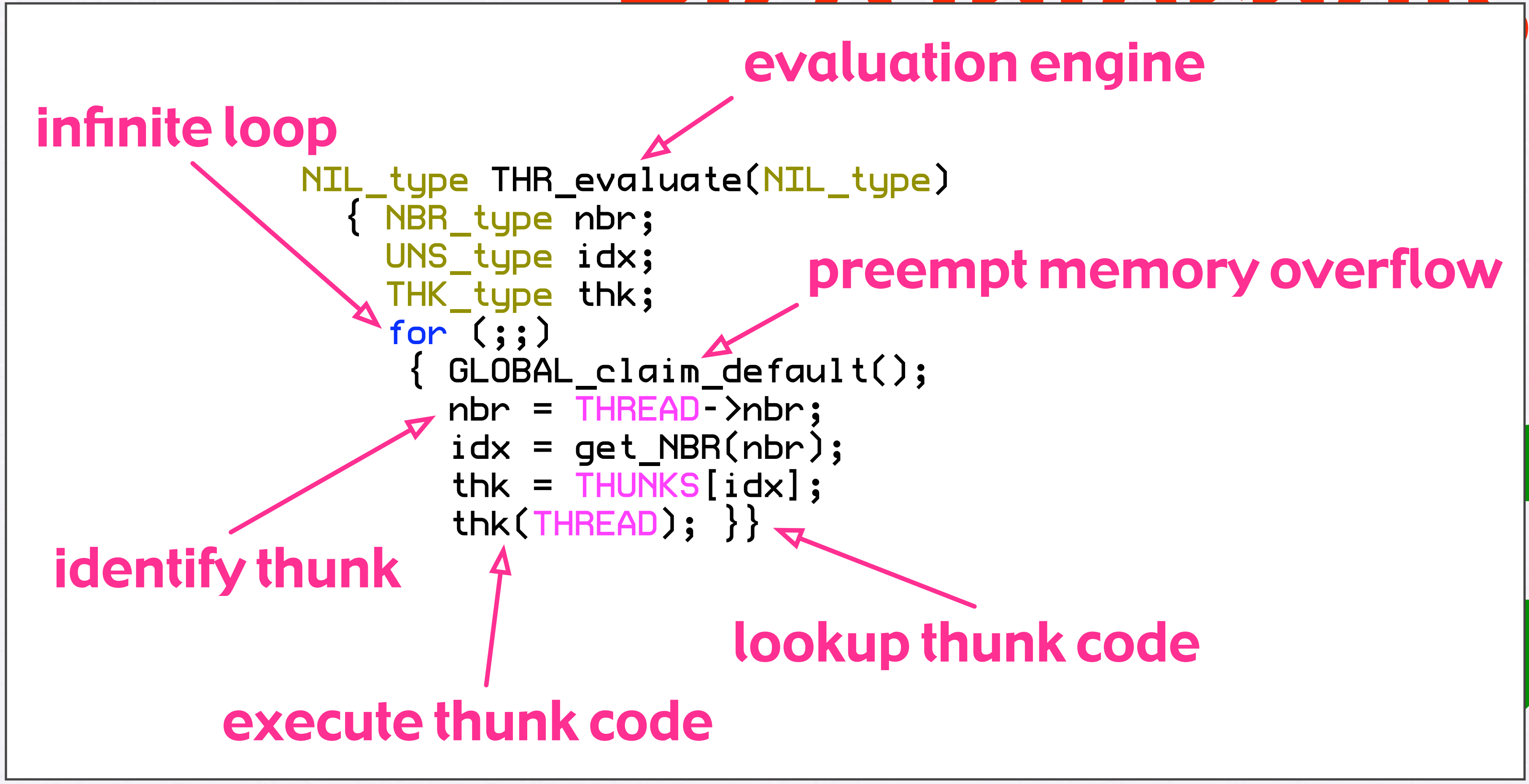
- * uniform memory + gc
- * abstract grammar driven
- * environments as lists
- * thread/
- * tail recu
- * smart c

```
{z:: 1; f(x,y): x+y+z}
```



- association list
- linked list
- circular closures
- static scoping
- apply => shallow copy
- duplicate variables
- performance <=> cache

Pico internals



```
NIL_type THR_evaluate(NIL_type)
  { NBR_type nbr;
    UNS_type idx;
    THK_type thk;
    for (;;)
     { GLOBAL_claim_default();
       nbr = THREAD->nbr;
       idx = get_NBR(nbr);
       thk = THUNKS[idx];
       thk(THREAD); }}
```

evaluation engine

infinite loop

preempt memory overflow

identify thunk

lookup thunk code

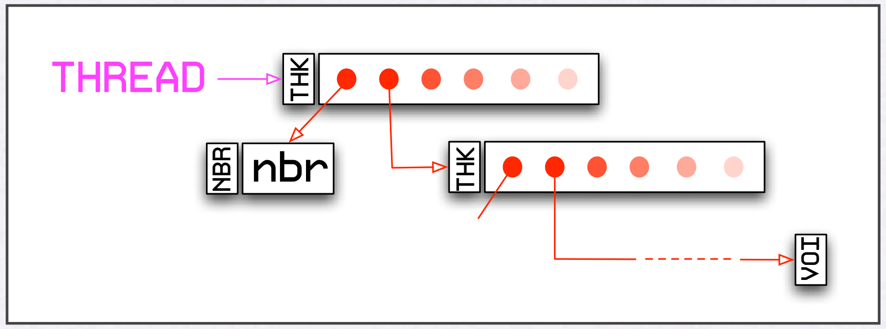
execute thunk code

+ gc

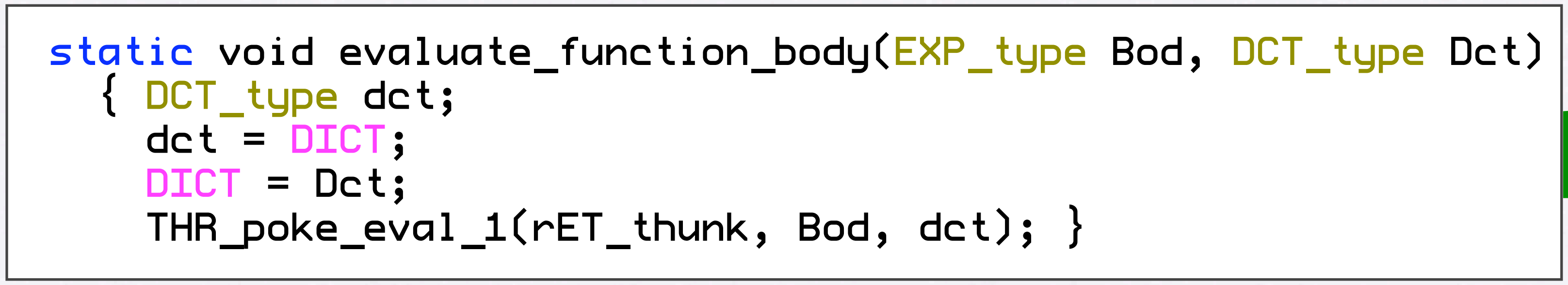
r driven

lists

- thread/continuation style
- tail recursion
- smart caching



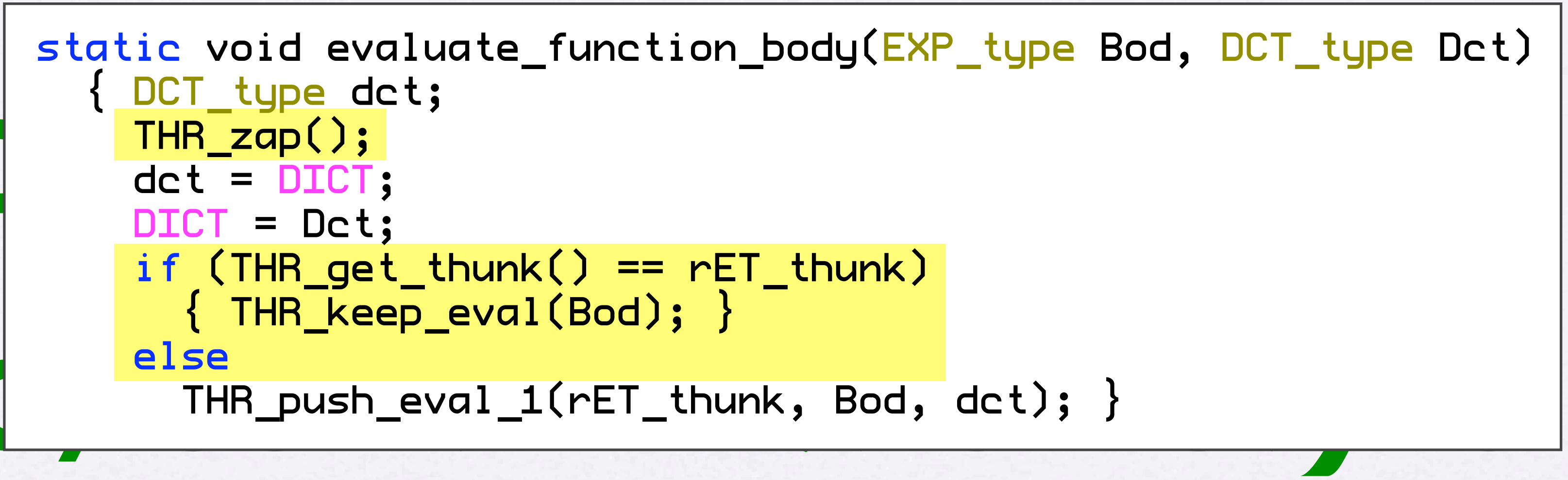
Pico internals



```
static void evaluate_function_body(EXP_type Bod, DCT_type Dct)
  { DCT_type dct;
    dct = DICT;
    DICT = Dct;
    THR_poke_eval_1(rET_thunk, Bod, dct); }
```

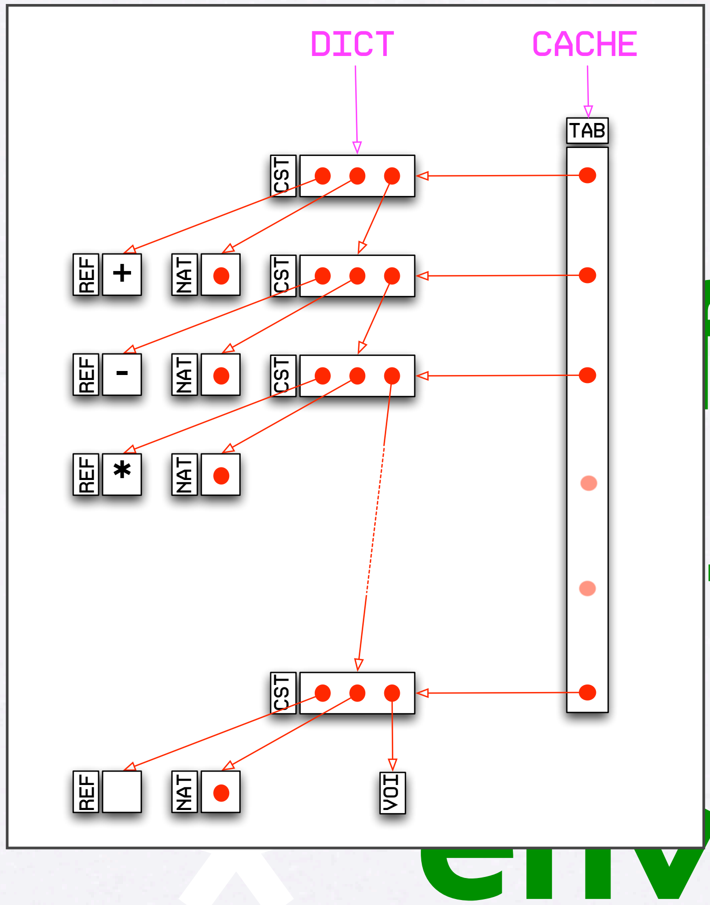
ry + gc

- abstract grammar driven
- enviro
- threa

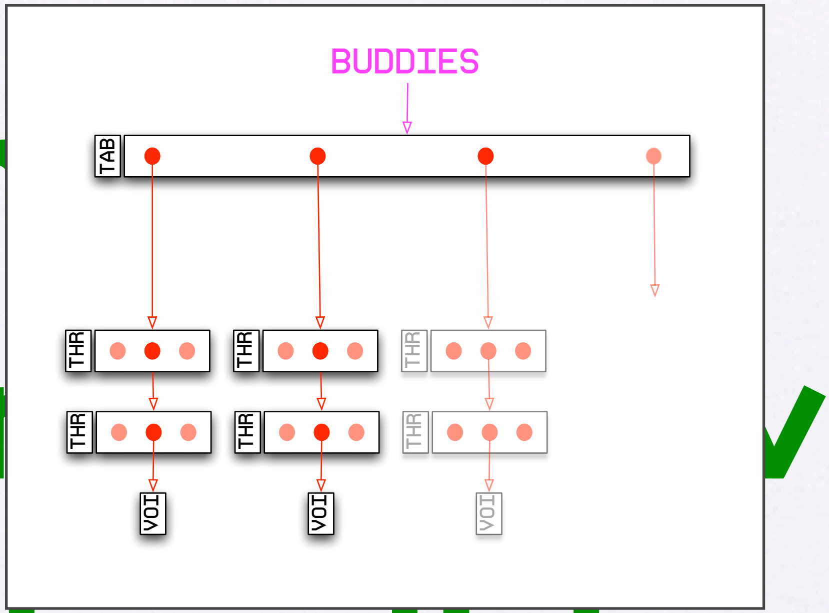


```
static void evaluate_function_body(EXP_type Bod, DCT_type Dct)
  { DCT_type dct;
    THR_zap();
    dct = DICT;
    DICT = Dct;
    if (THR_get_thunk() == rET_thunk)
      { THR_keep_eval(Bod); }
    else
      THR_push_eval_1(rET_thunk, Bod, dct); }
```

- tail recursion
- smart caching



Pico internals



form mer

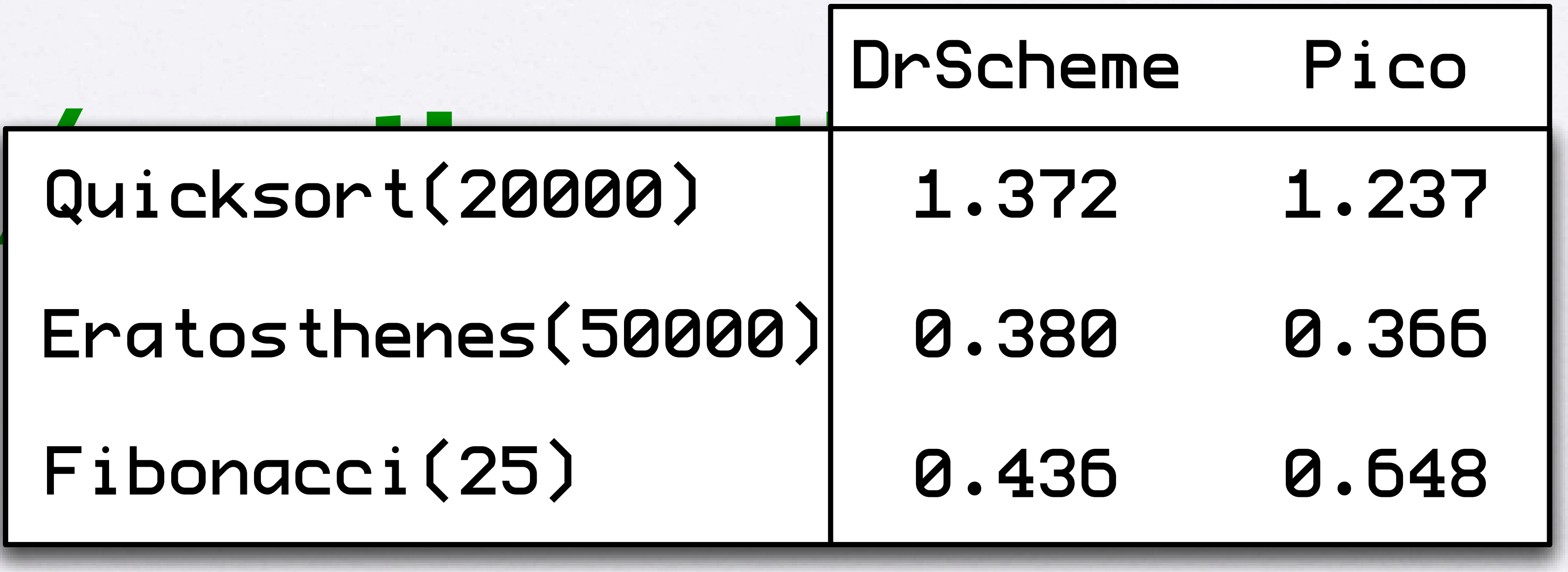
tract gra ven

environments as lists

- thread

- tail rec

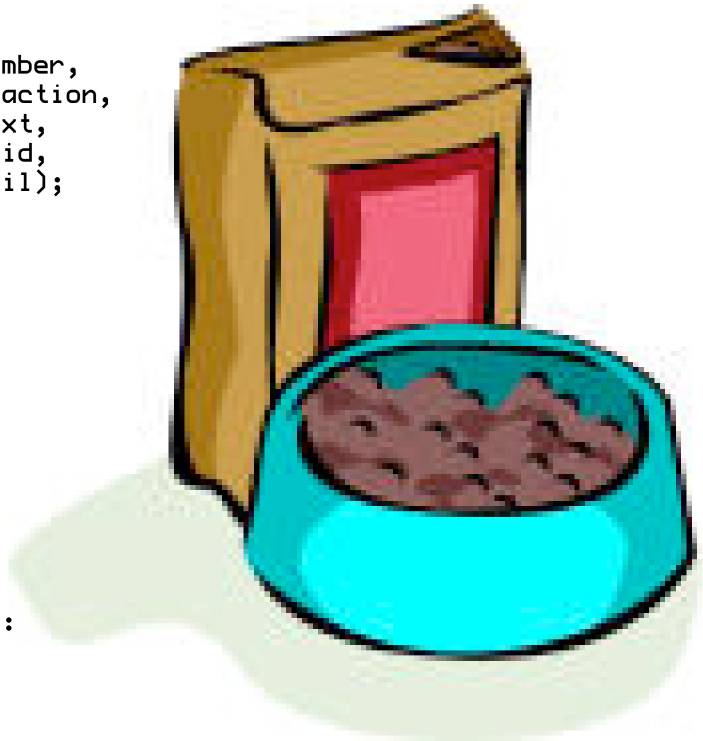
- smart caching



	DrScheme	Pico
Quicksort(20000)	1.372	1.237
Eratosthenes(50000)	0.380	0.366
Fibonacci(25)	0.436	0.648

```
unify(Ex1, Ex2, Frm):
  void;

unify_fail@Any:
  void;

same_number(Nb1, Nb
  Nb1[NBR_NBR_idx]

same_fraction(Fr1,
  Fr1[FRC_FRC_idx]

same_text(Tx1, Tx2):
  Tx1[TXT_TXT_idx] = Tx2[TXT_TXT_idx];

same_void(Vo1, Vo2):
  true;

same_fail(Vo1, Vo2):
  false;

unify_values_case: case(NBR_tag # same_number,
                        FRC_tag # same_fraction,
                        TXT_tag # same_text,
                        VOI_tag # same_void,
                           void # same_fail);

unify_values(Val, Exp, Frm):
  { tg1: Val[TAG_idx];
    tg2: Exp[TAG_idx];
    if(tg1 = tg2,
      { cas: unify_values_case(tg1);
        if(cas(Val, Exp),
          Frm,
          void) }) };

referenced(Var, Exp):
  { referenced_variable(Va1, Va2):
      same_variable(Va1, Va2);

    referenced_table_items(Var, Tab, Idx):
      if(Idx > size(Tab),
        false,
        if(referenced(Var, Tab[Idx]),
          true,
          referenced_table_items(Var, Tab, Idx+1)));

    referenced_table(Var, Tab):
      referenced_table_items(Var, Tab[TAB_TAB_idx], 1);

    referenced_pattern(Var, Pat):
      referenced_table(Var, Pat[PAT_TMS_idx]);

    referenced_value(Var, Val):
      false;

    referenced_case: case(VAR_tag # referenced_variable,
                          TAB_tag # referenced_table,
                          PAT_tag # referenced_pattern,
                             void # referenced_value);

    referenced(Var, Exp):=
      { tag: Exp[TAG_idx];
        cas: referenced_case(tag);
        cas(Var, Exp) };

    referenced(Var, Exp) };
```

http://pico.vub.ac.be



```
unify_table_items(Ta1, Ta2, Frm, Idx):
  if(is_void(Frm) | (Idx>size(Ta1)),
    Frm,
    unify_table_items(Ta1, Ta2, unify(Ta1[Idx], Ta2[Idx], Frm), Idx+1));

unify_2_tables(Ta1, Ta2, Frm):
  { ta1: Ta1[TAB_TAB_idx];
    ta2: Ta2[TAB_TAB_idx];
    if(size(ta1) = size(ta2),
      unify_table_items(ta1, ta2, Frm, 1),
      void) };

unify_table_case: case(VAR_tag # unify_variable,
                       TAB_tag # unify_2_tables,
                          void # unify_fail);

unify_table(Tab, Exp, Frm):
  { tag: Exp[TAG_idx];
    cas: unify_table_case(tag);
    cas(Exp, Tab, Frm) };

unify_2_patterns(Pa1, Pa2, Frm):
  if(Pa1[PAT_SYM_idx] = Pa2[PAT_SYM_idx],
    unify(Pa1[PAT_TMS_idx], Pa2[PAT_TMS_idx], Frm),
    void);

unify_pattern_case: case(VAR_tag # unify_variable,
                         PAT_tag # unify_2_patterns,
                            void # unify_fail);

unify_pattern(Pat, Exp, Frm):
  { tag: Exp[TAG_idx];
    cas: unify_pattern_case(tag);
    cas(Exp, Pat, Frm) };

unify_value(Val, Exp, Frm):
  { tag: Exp[TAG_idx];
    if(tag = VAR_tag,
      unify_variable(Exp, Val, Frm),
      unify_values(Val, Exp, Frm)) };

unify_case: case(VAR_tag # unify_variable,
                 TAB_tag # unify_table,
                 PAT_tag # unify_pattern,
                    void # unify_value);

unify(Tm1, Tm2, Frm):=
  { tag: Tm1[TAG_idx];
    cas: unify_case(tag);
    cas(Tm1, Tm2, Frm) };
```