Vrije Universiteit Brussel

Software Languages.Lab

Programming Language Engineering
Master of Computer Science

Faculty of Science and Bio-Engineering Sciences
Vrije Universiteit Brussel

Section 10: Optimization

Theo D'Hondt
Software Languages Lab

"… an act, process, or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible …"

Further Optimizations

	functional	non functional
algorithms	lexical addressing	caching
technology	continuations	inlining

Non-functional Aspects

- space constraints
- time constraints
- no stop-the-world
- ...

Smart Caching

- capture free memory chunks of specific category
- store them in specialized freelists
- intercept standard allocations from heap
- reduce garbage collection

Smart Caching

- capture free memory chunks of specific category
- store them in specialized freelists
- intercept standard allocations from heap
- reduce garbage collection

Apply to the Thread stack

Thread Allocation (recap)

evaluator module

```
static EXP_type evaluate_set_local(STL_type Set)
  { sTL_type set_thread;
    EXP_type expression;
    NBR_type offset;
    offset     = Set->ofs;
    expression = Set->exp;
    set_thread = (sTL_type)Thread_Push(Continue_set_local,
                                       Main_False,
                                       sTL_size);
    set_thread->ofs = offset;
    return evaluate_expression(expression,
                               Main_False); }
```

Thread Allocation (recap)

thread interface

```
THR_type        Thread_Patch(NBR_type);
THR_type          Thread_Pop(NIL_type);
THR_type         Thread_Push(NBR_type,
                             EXP_type,
                             UNS_type);
NBR_type    Thread_Register(CCC_type);
NIL_type     Thread_Restore(NIL_type);
```

evaluator module

```
static EXP_type evaluate_set_local(STL_ty
  { sTL_type set_thread;
    EXP_type expression;
    NBR_type offset;
    offset     = Set->ofs;
    expression = Set->exp;
    set_thread = (sTL_type)Thread_Push(Continue_set_local,
                                       Main_False,
                                       sTL_size);
    set_thread->ofs = offset;
    return evaluate_expression(expression,
                               Main_False); }
```

Thread Allocation (recap)

thread interface

```
THR_type        Thread_Patch(NBR_type);
THR_type          Thread_Pop(NIL_type);
THR_type         Thread_Push(NBR_type,
                             EXP_type,
                             UNS_type);
NBR_type    Thread_Register(CCC_type);
NIL_type     Thread_Restore(NIL_type);
```

evaluator module

```
static EXP_type evaluate_set_local(STL_ty
  { sTL_type set_thread;
    EXP_type expression:
    NBR_type of
    offset
    expressi
    set_thre                                      .,


    set_thre
    return e
```

thread module

```
THR_type Thread_Push(NBR_type Thread_id,
                     EXP_type Call_status,
                     UNS_type Size)
  { THR_type thread;
    thread = make_THR(Thread_id,
                      Threaded_continuation,
                      Call_status,
                      Size);
    Threaded_continuation = thread;
    return thread; }
```

Thread Allocation (recap)

thread interface

```
THR_type        Thread_Patch(NBR_type);
THR_type          Thread_Pop(NIL_type);
THR_type         Thread_Push(NBR_type,
                             EXP_type,
                             UNS_type);
NBR_type    Thread_Register(CCC_type);
NIL_type     Thread_Restore(NIL_type);
```

evaluator module

```
static EXP_type evaluate_set_local(STL_ty
  { sTL_type set_thread;
    EXP_type expression:
    NBR_type of
    offset
    expressi
    set_thre


    set_thre
    return e
```

thread module

```
THR_type Thread_Push(NBR_type Thread_id,
                     EXP_type Call_s
                     UNS_type Size)
  { THR_type thread;
    thread = make_THR(Thread_id,
                      Threaded_cont
                      Call_status,
                      Size);
    Threaded_continuation = thread
    return thread; }
```

grammar interface

```
typedef
  struct THR { CEL_type hdr;
               NBR_type tid;
               THR_type thr;
               EXP_type tcs;
               EXP_type exp[]; } THR;
BYT_type   is_THR(EXP_type);
BYT_type marked_THR(THR_type);
THR_type make_THR(NBR_type,
                  THR_type,
                  EXP_type,
                  UNS_type);
NIL_type mark_THR(THR_type);
```

Thread Allocation (recap)

thread interface

```
   Thread_Patch(NBR_type);
     Thread_Pop(NIL_type);
    Thread_Push(NBR_type,
                EXP_type,
                UNS_type);
Thread_Register(CCC_type);
 Thread_Restore(NIL_type);
```

grammar module

```
THR_type make_THR(NBR_type Thread_id,
                  THR_type Thread,
                  EXP_type Call_status,
                  UNS_type Size)
  { THR_type thread;
    UNS_type index;
    thread = make_chunk_with_offset(THR,
                                    Size);
    thread->tid = Thread_id;
    thread->thr = Thread;
    thread->tcs = Call_status;
    for (index = 0;
```

eva

```
static
  { sTL
    EXP
    NBR
    off
    exp
    set


    set
    ret
```

```
id,
                            Call_status,
                            Size);
    Threaded_continuation = thread
    return thread; }
```

grammar interface

```
edef
truct THR { CEL_type hdr;
            NBR_type tid;
            THR_type thr;
            EXP_type tcs;
            EXP_type exp[]; } THR;
BYT_type   is_THR(EXP_type);
BYT_type marked_THR(THR_type);
THR_type make_THR(NBR_type,
                  THR_type,
                  EXP_type,
                  UNS_type);
NIL_type mark_THR(THR_type);
```

Thread Allocation (recap)

eva

```
static
  { sTL
    EXP
    NBR
    off
    exp
    set


    set
    ret
```

grammar module

```
THR_type make_THR(NBR_type Thread_id,
                  THR_type Thread,
                  EXP_type Call_status,
                  UNS_type Size)
 { THR_type thread;
   UNS_type index;
   thread = make_chunk_with_offset(THR,
                                   Size);
   thread->tid = Thread_id;
   thread->thr = Thread;
   thread->tcs = Call_status;
   for (index = 0;
```

```
                          id,
                                    Call_status,
                                    Size);
     Threaded_continuation = thread;
     return thread; }
```

thread interface

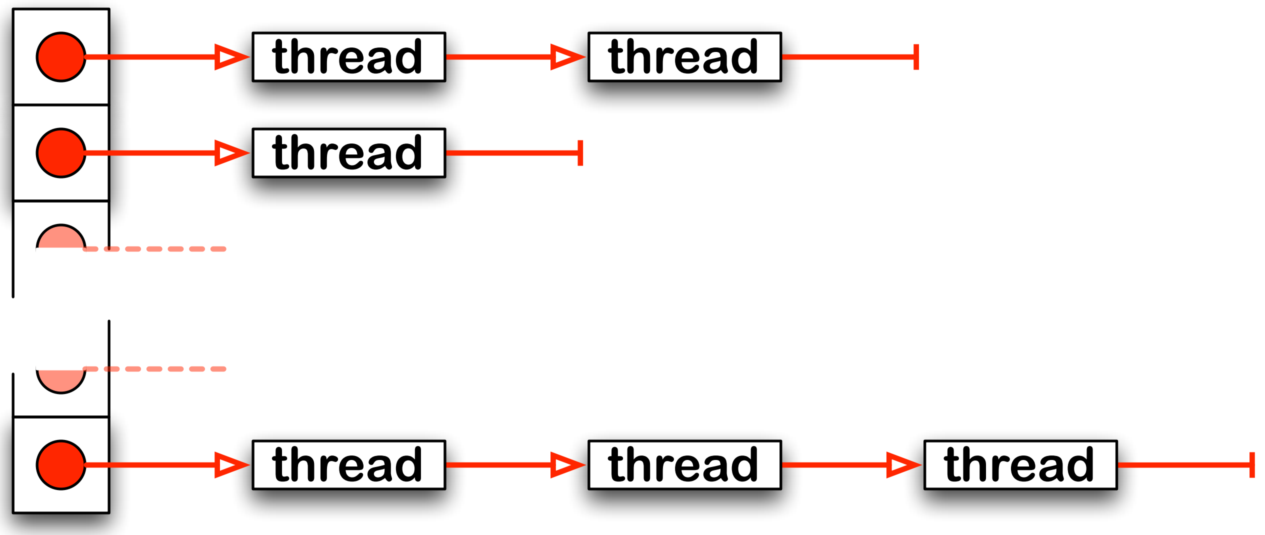
```
   Thread_Patch(NBR_type);
     Thread_Pop(NIL_type);
    Thread_Push(NBR_type,
                EXP_type,
                UNS_type);
Thread_Register(CCC_type);
 Thread_Restore(NIL_type);
```

grammar interface

```
typedef
 struct THR { CEL_type hdr;
              NBR_type tid;
              THR_type thr;
              EXP_type tcs;
              EXP_type exp[]; } THR;
BYT_type   is_THR(EXP_type);
BYT_type marked_THR(THR_type);
THR_type make_THR(NBR_type,
                  THR_type,
                  EXP_type,
                  UNS_type);
NIL_type mark_THR(THR_type);
```

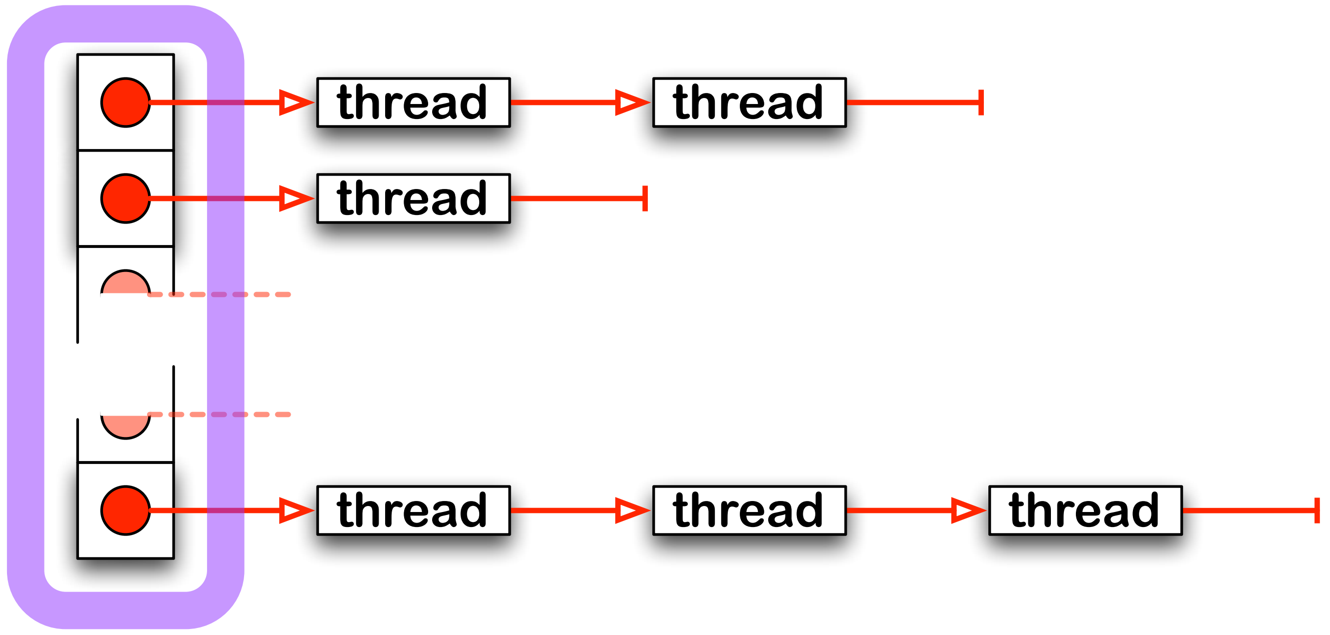
invoke the memory manager

A simple Thread pool



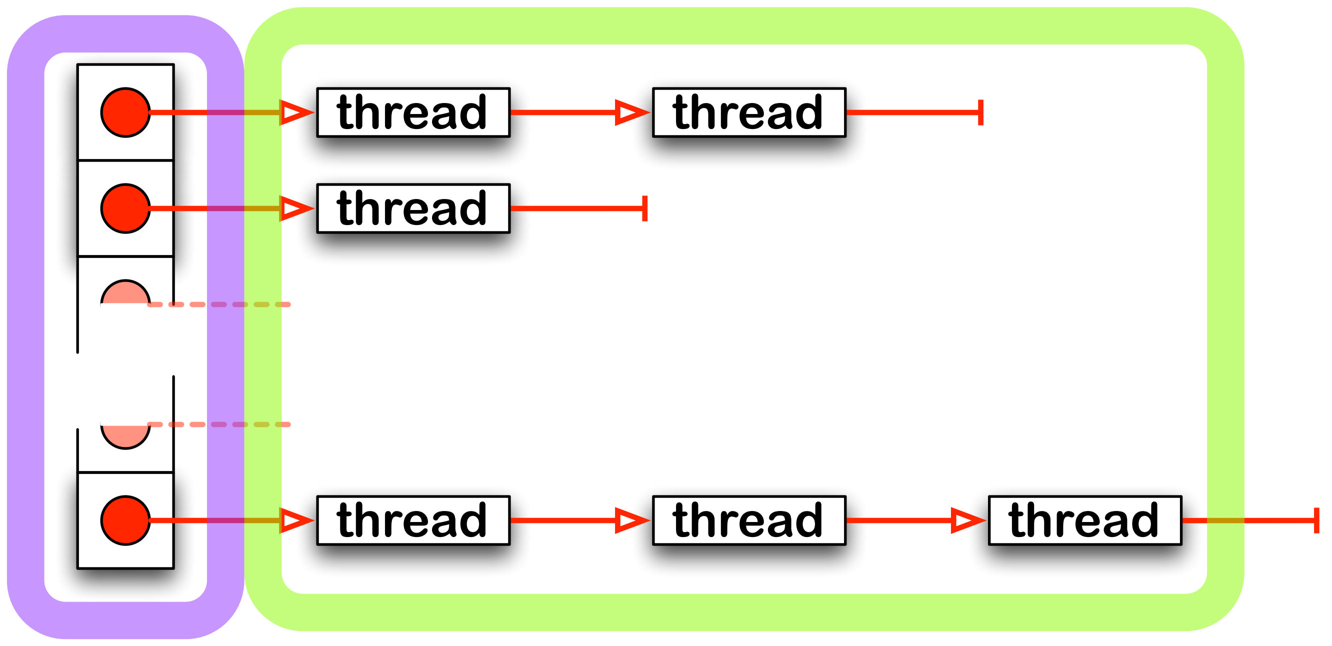
A simple Thread pool

freelists grouped by size



A simple Thread pool

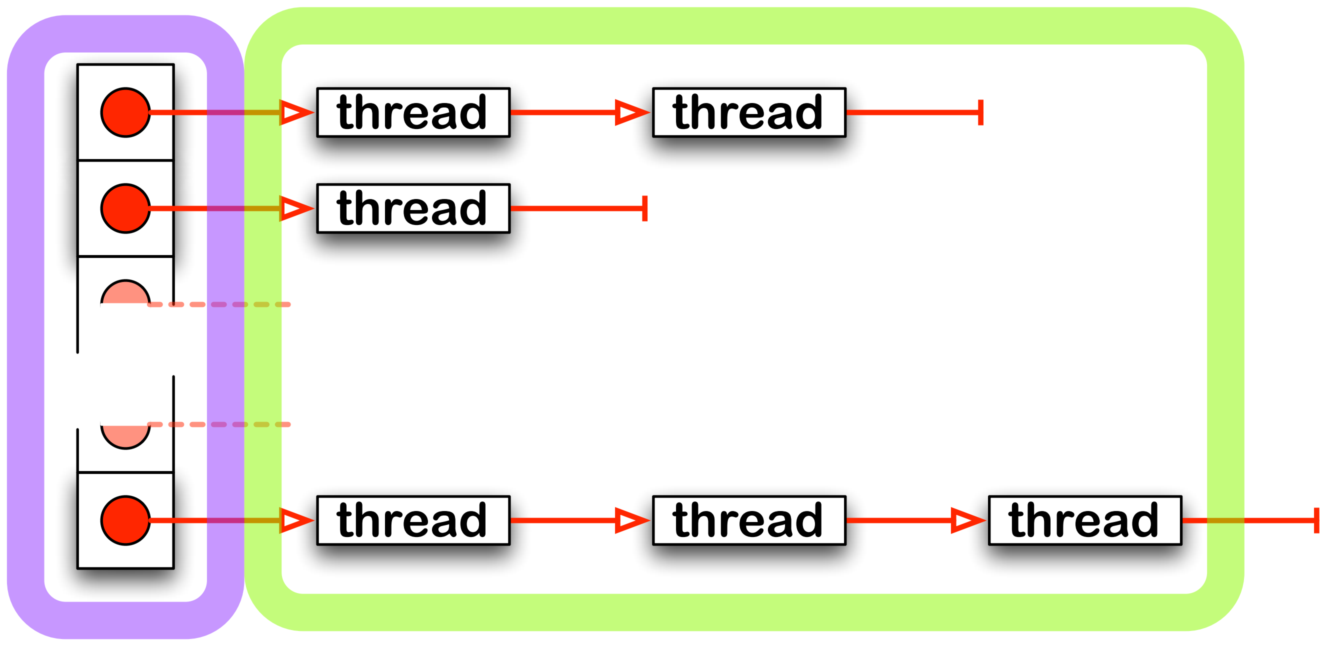
freelists grouped by size



zapped threads

A simple Thread pool

freelists grouped by size

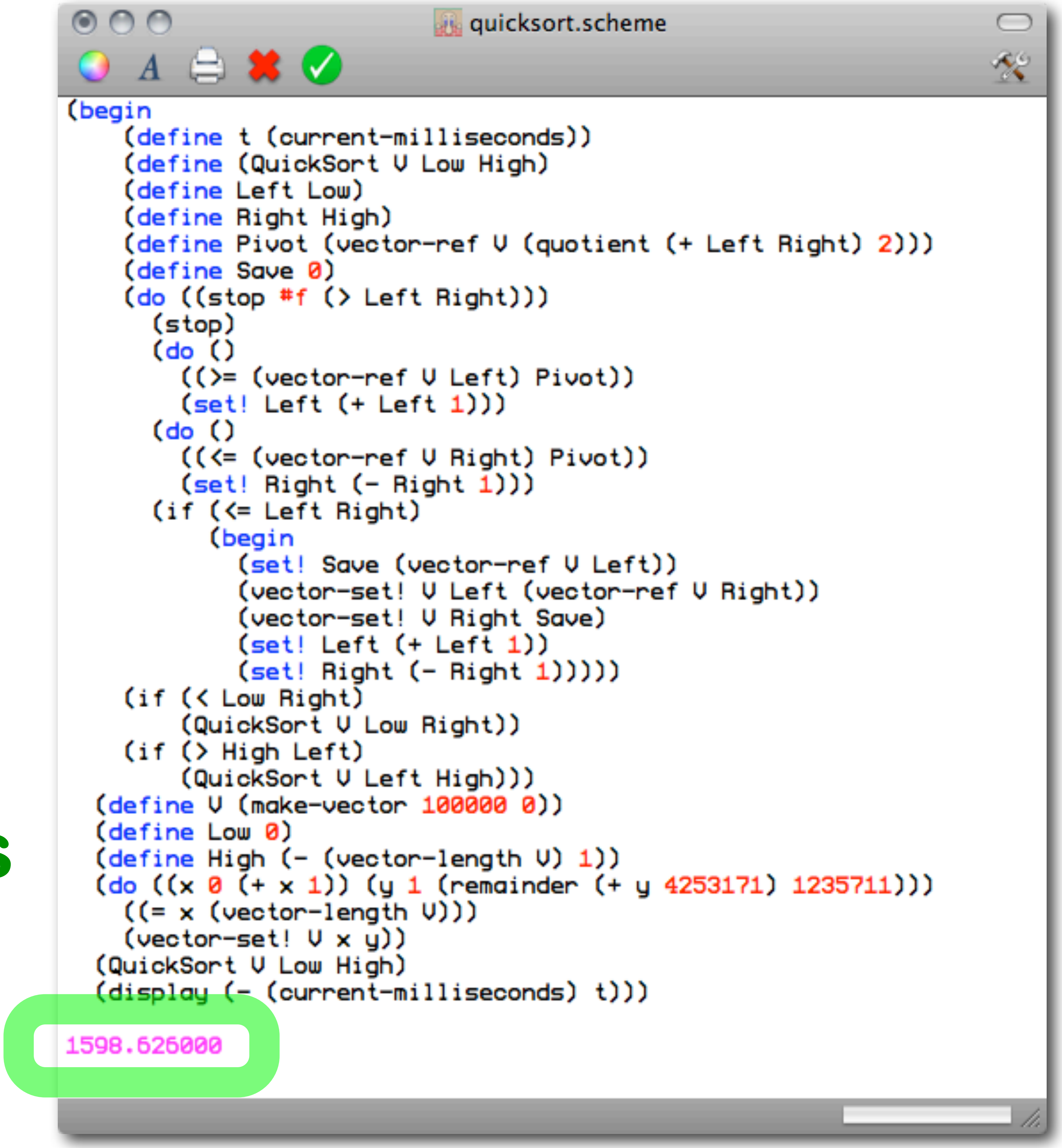


zapped threads

allocation from freelist
else memory manager call

A simple timing experiment

\'skēm\

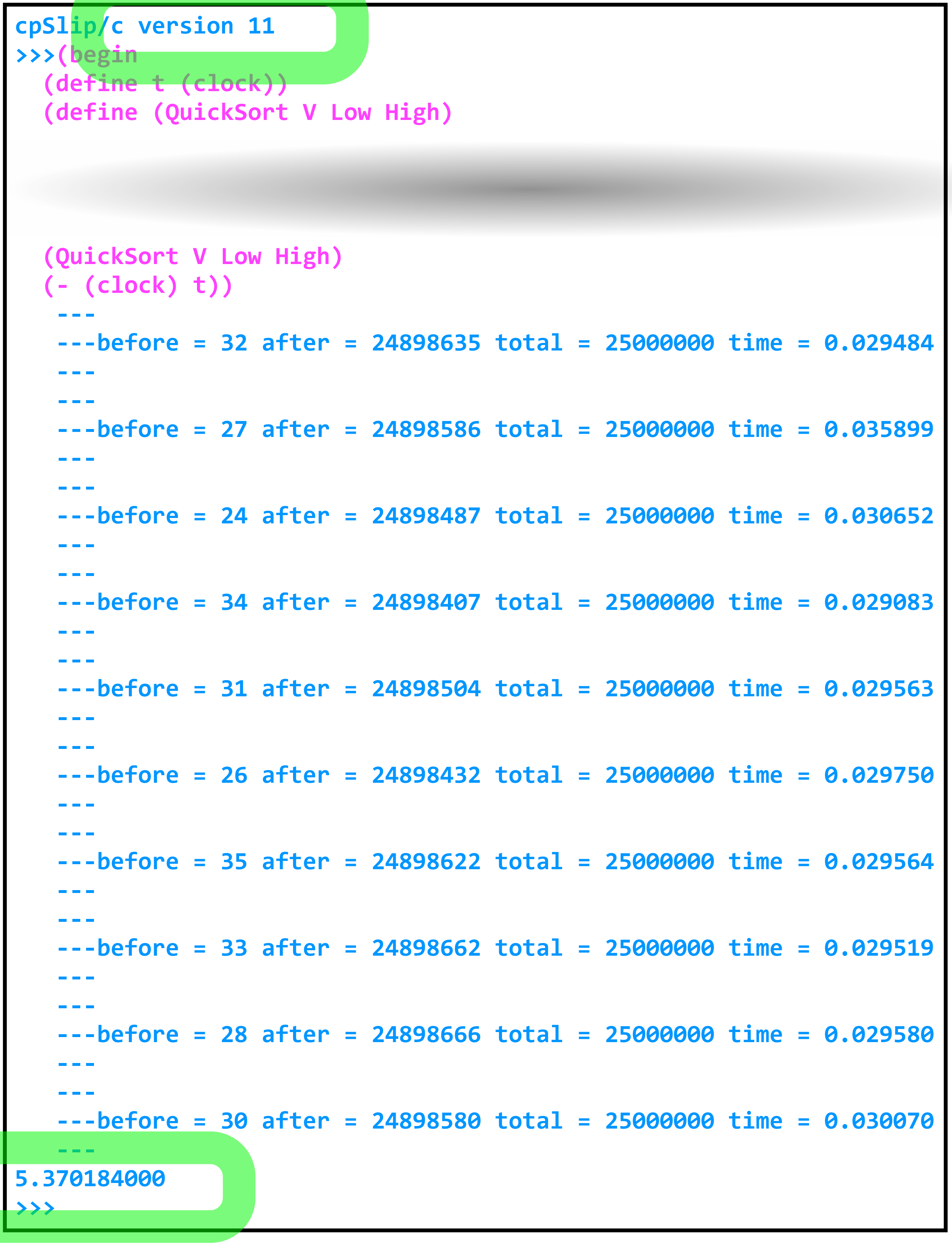


```
(begin
    (define t (current-milliseconds))
    (define (QuickSort V Low High)
    (define Left Low)
    (define Right High)
    (define Pivot (vector-ref V (quotient (+ Left Right) 2)))
    (define Save 0)
    (do ((stop #f (> Left Right)))
      (stop)
      (do ()
        ((>= (vector-ref V Left) 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)))
  (define V (make-vector 100000 0))
  (define Low 0)
  (define High (- (vector-length V) 1))
  (do ((x 0 (+ x 1)) (y 1 (remainder (+ y 4253171) 1235711)))
    ((= x (vector-length V)))
    (vector-set! V x y))
  (QuickSort V Low High)
  (display (- (current-milliseconds) t)))

1598.626000
```

16secs

quicksort of 100000 random integers in 1000 Mb

A simple timing experiment: no cache

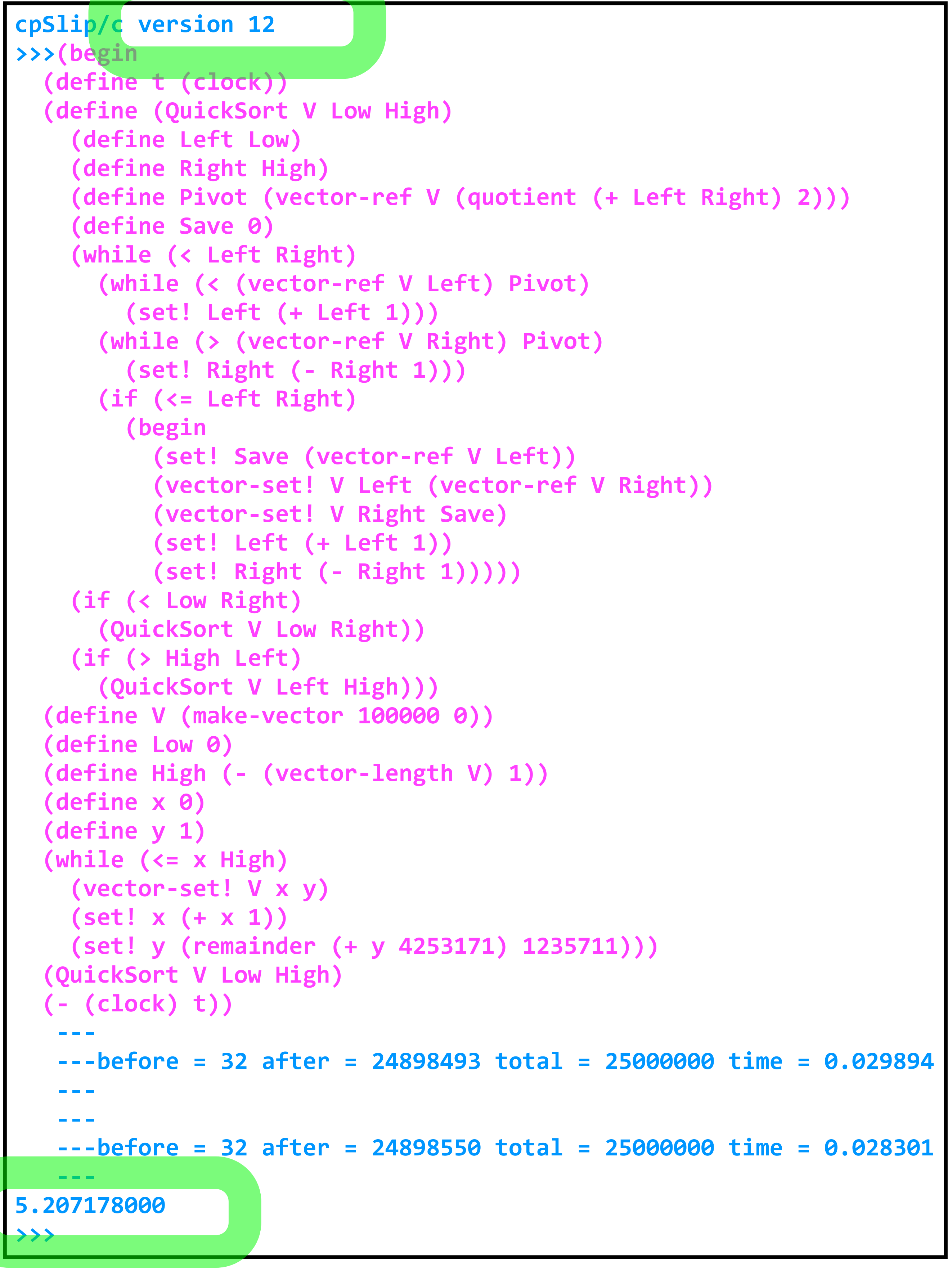


```
cpSlip/c version 11
>>>(begin
  (define t (clock))
  (define (QuickSort V Low High)


  (QuickSort V Low High)
  (- (clock) t))
   ---
   ---before = 32 after = 24898635 total = 25000000 time = 0.029484
   ---
   ---
   ---before = 27 after = 24898586 total = 25000000 time = 0.035899
   ---
   ---
   ---before = 24 after = 24898487 total = 25000000 time = 0.030652
   ---
   ---
   ---before = 34 after = 24898407 total = 25000000 time = 0.029083
   ---
   ---
   ---before = 31 after = 24898504 total = 25000000 time = 0.029563
   ---
   ---
   ---before = 26 after = 24898432 total = 25000000 time = 0.029750
   ---
   ---
   ---before = 35 after = 24898622 total = 25000000 time = 0.029564
   ---
   ---
   ---before = 33 after = 24898662 total = 25000000 time = 0.029519
   ---
   ---
   ---before = 28 after = 24898666 total = 25000000 time = 0.029580
   ---
   ---
   ---before = 30 after = 24898580 total = 25000000 time = 0.030070
   ---
5.370184000
>>>
```

10 GC's

5.3 secs

A simple timing experiment: with cache



```
cpSlip/c version 12
>>>(begin
  (define t (clock))
  (define (QuickSort V Low High)
    (define Left Low)
    (define Right High)
    (define Pivot (vector-ref V (quotient (+ Left Right) 2)))
    (define Save 0)
    (while (< Left Right)
      (while (< (vector-ref V Left) Pivot)
        (set! Left (+ Left 1)))
      (while (> (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)))
  (define V (make-vector 100000 0))
  (define Low 0)
  (define High (- (vector-length V) 1))
  (define x 0)
  (define y 1)
  (while (<= x High)
    (vector-set! V x y)
    (set! x (+ x 1))
    (set! y (remainder (+ y 4253171) 1235711)))
  (QuickSort V Low High)
  (- (clock) t))
   ---
   ---before = 32 after = 24898493 total = 25000000 time = 0.029894
   ---
   ---
   ---before = 32 after = 24898550 total = 25000000 time = 0.028301
   ---
5.207178000
>>>
```

5.2 secs!

2 GC's

Separate interface/implementation

evaluator module

```
static EXP_type continue_set_global(EXP_type Value,
                                    EXP_type Tail_call)
  { sTG_type set_thread;
    NBR_type offset,
             scope;
    UNS_type raw_offset,
             raw_scope;
    set_thread = (sTG_type)Thread_Pop();
    scope  = set_thread->scp;
    offset = set_thread->ofs;
    raw_scope = get_NBR(scope);
    raw_offset = get_NBR(offset);
    Environment_Global_Set(raw_scope,
                           raw_offset,
                           Value);
    return Value; }
```

Separate interface/implementation

environment interface

```
UNS_type       Environment_Get_Environment_size(NIL_type);
VEC_type                   Environment_Get_Frame(NIL_type);
VEC_type                Environment_Global_Frame(NIL_type);
EXP_type                  Environment_Global_Get(UNS_type,
                                                 UNS_type);
BYT_type             Environment_Global_Overflow(UNS_type);
NIL_type                  Environment_Global_Set(UNS_type,
                                                 UNS_type,
                                                 EXP_type);
VEC_type            Environment_Grow_Environment(NIL_type);
NIL_type                  Environment_Initialize(NIL_type);
EXP_type                   Environment_Local_Get(UNS_type);
```

evaluator mo

```
static EXP_type contin

  { sTG_type set_threa
    NBR_type offset,
             scope;
    UNS_type raw_offse
             raw_scope
    set_thread = (sTG_
    scope   = set_threa
    offset = set_threa
    raw_scope = get_NBR(scope);
    raw_offset = get_NBR(offset);
    Environment_Global_Set(raw_scope,
                           raw_offset,
                           Value);
    return Value; }
```

Separate interface/implementation

evaluator mo

```
static EXP_type contir

  { sTG_type set_threa
    NBR_type offset,
             scope;
    UNS_type raw_offse
             raw_scope
    set_thread = (sTG_
    scope  = set_threa
    offset = set_threa
    raw_scope = get_NBR(scope);
    raw_offset = get_NBR(offset);
    Environment_Global_Set(raw_sco
                           raw_off
                           Value);
    return Value; }
```

environment interface

```
UNS_type      Environment_Get_Environment_size(NIL_type);
VEC_type                 Environment_Get_Frame(NIL_type);
VEC_type              Environment_Global_Frame(NIL_type);
EXP_type                Environment_Global_Get(UNS_type,
                                               UNS_type);
BYT_type           Environment_Global_Overflow(UNS_type);
NIL_type                Environment_Global_Set(UNS_type,
                                               UNS_type,
                                               EXP_type);
VEC_type          Environment_Grow_Environment(NIL_type);
NIL_type                                          type);
EXP_type
```

environment module

```
static VEC_type Current_environment;
static VEC_type Current_frame;



NIL_type Environment_Global_Set(UNS_type Scope,
                                UNS_type Offset,
                                EXP_type Value)
  { VEC_type frame;
    frame = Current_environment[Scope];
    frame[Offset] = Value; }
```

Inlining code

evaluator mo

```
static EXP_type contir

  { sTG_type set_threa
    NBR_type offset,
             scope;
    UNS_type raw_offse
             raw_scope
    set_thread = (sTG_
    scope  = set_threa
    offset = set_threa
    raw_scope = get_NE
    raw_offset = get_N
    Environment_Global


    return Value; }
```

environment interface

```
UNS_type       Environment_Get_Environment_size(NIL_type);
VEC_type                  Environment_Get_Frame(NIL_type);
VEC_type               Environment_Global_Frame(NIL_type);
EXP_type                 Environment_Global_Get(UNS_type,
                                                UNS_type);
BYT_type            Environment_Global_Overflow(UNS_type);

extern VEC_type Current_environment;
extern VEC_type Current_frame;

NIL_type Environment_Global_Set(UNS_type Scope,
                                UNS_type Offset,
                                EXP_type Value)
  { VEC_type frame;
    frame = Current_environment[Scope];
    frame[Offset] = Value; }

VEC_type          Environment_Grow_Environment(NIL_type);
NIL_type                 Environment_Initialize(NIL_type);
EXP_type                  Environment_Local_Get(UNS_type);
```

Inlining code

evaluator mo

```
static EXP_type contir

  { sTG_type set_threa
    NBR_type offset,
             scope;
    UNS_type raw_offse
             raw_scope
    set_thread = (sTG_
    scope  = set_threa
    offset = set_threa
    raw_scope = get_NB
    raw_offset = get_N
    Environment_Global


    return Value; }
```

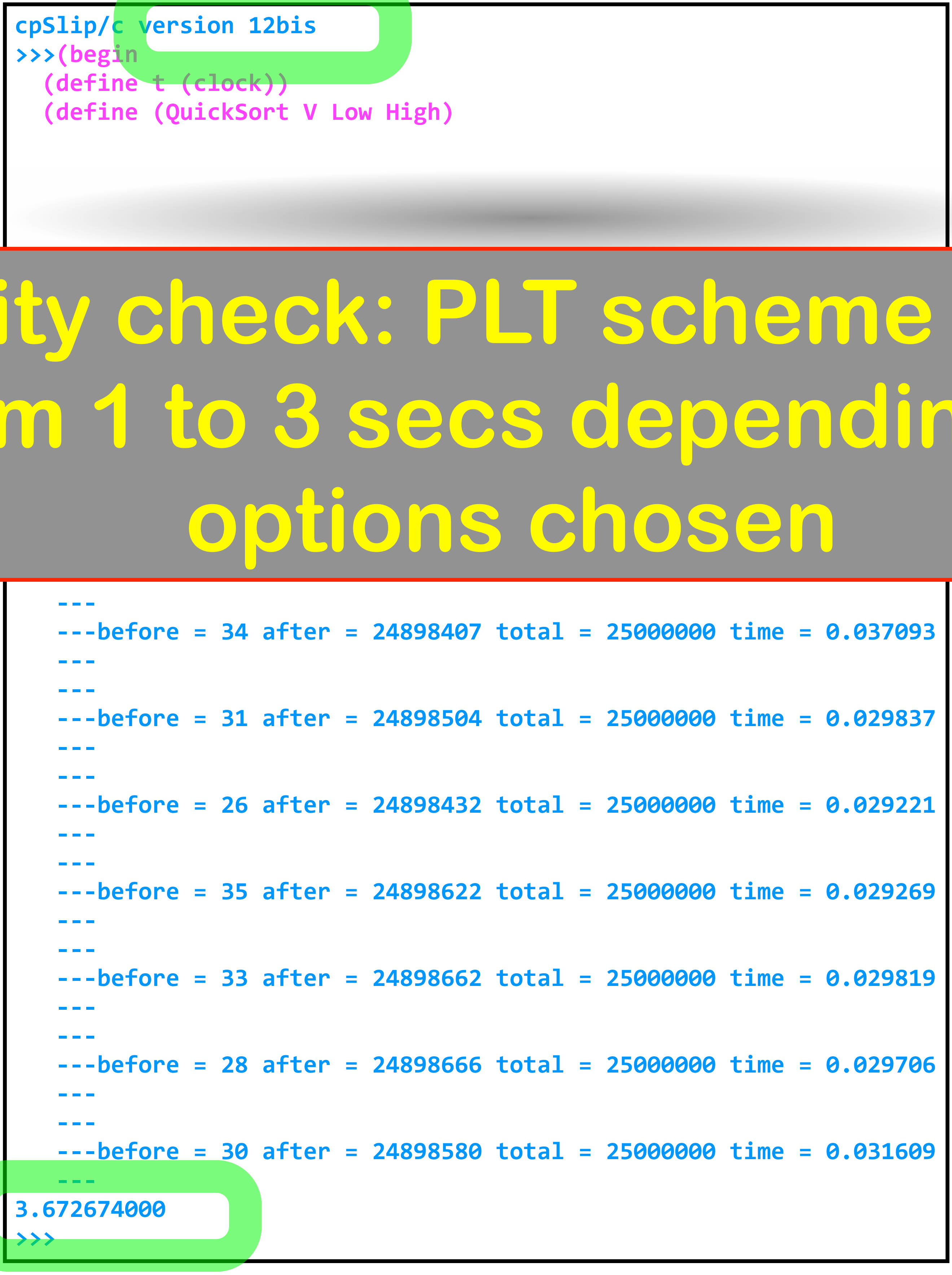
environment interface

```
UNS_type      Environment_Get_Environment_size(NIL_type);
VEC_type                Environment_Get_Frame(NIL_type);
VEC_type             Environment_Global_Frame(NIL_type);
EXP_type               Environment_Global_Get(UNS_type,
                                              UNS_type);
BYT_type          Environment_Global_Overflow(UNS_type);

extern VEC_type Current_environment;
extern VEC_type Current_frame;

NIL_type Environment_Global_Set(UNS_type Scope,
                                UNS_type Offset,
                                EXP_type Value)
  { VEC_type frame;
    frame = Current_environment[Scope];
    frame[Offset] = Value; }

VEC_type          Environment Grow Environment(NIL type);
NIL_type                                             e);
EXP_type
```

environment module

```
VEC_type Current_environment;
VEC_type Current_frame;
```

A simple timing experiment: inlining



```
cpSlip/c version 12bis
>>>(begin
  (define t (clock))
  (define (QuickSort V Low High)

    ---
    ---before = 34 after = 24898407 total = 25000000 time = 0.037093
    ---
    ---
    ---before = 31 after = 24898504 total = 25000000 time = 0.029837
    ---
    ---
    ---before = 26 after = 24898432 total = 25000000 time = 0.029221
    ---
    ---
    ---before = 35 after = 24898622 total = 25000000 time = 0.029269
    ---
    ---
    ---before = 33 after = 24898662 total = 25000000 time = 0.029819
    ---
    ---
    ---before = 28 after = 24898666 total = 25000000 time = 0.029706
    ---
    ---
    ---before = 30 after = 24898580 total = 25000000 time = 0.031609
    ---
3.672674000
>>>
```

sanity check: PLT scheme takes from 1 to 3 secs depending on options chosen

evaluator code goes from 167k to 272k

10 GC's

3.7 secs

What’s next?

- dynamic compilation
- modular interpreters
- a real successor to C
- a very high-level macro language ... Haskell?

Top 3 conclusions

- it’s becoming harder to write interpreters than to write compilers
- interpreters are the way to go - compilation is optimization
- we won’t be doing manycore language engineering on top of bytecodes