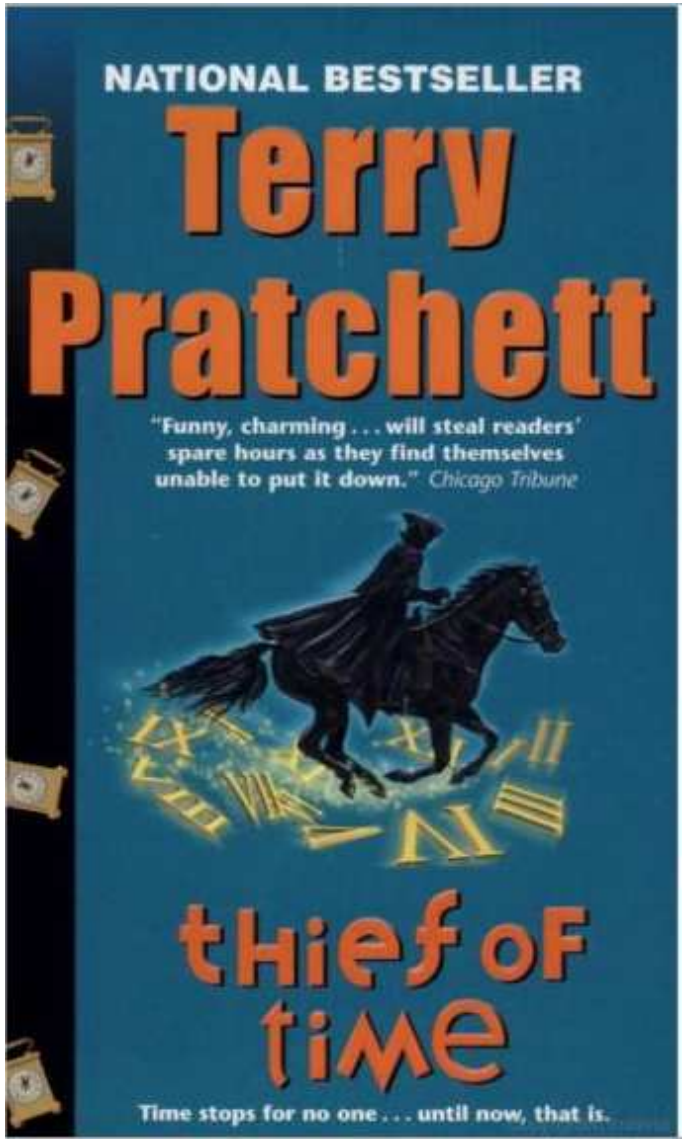
NATIONAL BESTSELLER
Terry Pratchett
"Funny, charming . . . will steal readers' spare hours as they find themselves unable to put it down." Chicago Tribune
thief of time
Time stops for no one . . . until now, that is.

# Page 1

*According to the First Scroll of Wen the Eternally Surprised*, Wen stepped out of the cave where he had received enlightenment and into the dawning light of the first day of the rest of his life. He stared at the rising sun for some time, because he had never seen it before.

He prodded with a sandal the dozing form of Clodpool the Apprentice, and said: "I have seen. Now I understand."

Then he stopped and looked at the thing next to Clodpool.

"What is that amazing thing?" he said.

"Er . . . er . . . it's a tree, master," said Clodpool, still not quite awake. "Remember? It was there yesterday."

"There was no yesterday."

"Er . . . er . . . I think there *was*, master," said Clodpool, struggling to his feet. "Remember? We came up here, and I cooked a meal, and had the rind off your *sklang* because you didn't want it."

"I *remember* yesterday," said Wen, thoughtfully. "But the memory is in my head *now*. Was yesterday real? Or is it only the memory that is real? Truly, yesterday I was not born."

Clodpool's face became a mask of agonized incomprehension.

"Dear stupid Clodpool, I have learned everything," said Wen. "In the cup of the hand there is no past, no future. There is only now. There is no time but the present. We have a great deal to do."

# Page 27

The first question they ask is: "Why was he eternally surprised?"

And they are told: "Wen considered the nature of time and understood that the universe is, instant by instant, re-created anew. Therefore, he understood, there is, in truth, no Past, only a memory of the Past. Blink your eyes, and the world you see next did not exist when you closed them. Therefore, he said, the only appropriate state of the mind is surprise. The only appropriate state of the heart is joy. The sky you see now, you have never seen before. The perfect moment is now. Be glad of it."

# Store-Passing Interpreters

Our **BCFAE** interpreter explains state by representing the store as a value

- Every step in computation produces a new store
- The interpreter itself is purely functional

It's a ***store-passing interpreter***

# Variables

Boxes don’t explain one of our earlier Scheme examples:

```
(define counter 0)
(define (f x)
  (begin
    (set! counter (+ x counter))
    counter))
```

In a program like this, an identifier no longer stands for a ***value***; instead, an identifier stands for a ***variable***

# Implementing Variables

Option 1:

```
(define counter 0)
(define (f x)
  (begin
    (set! counter (+ x counter))
    counter))
(f 10)
```

⇒

```
(define counter (box 0))
(define (f x)
  (begin
    (set-box! counter (+ (unbox x)
                         (unbox counter)))
    (unbox counter)))
(f (box 10))
```

Option 2:

- Essentially the same, but hide the boxes in the interpreter

# BMCFAE = BCFAE + variables

```
<BMCFAE>  ::=  <num>
           |   {+ <BMCFAE> <BMCFAE>}
           |   {- <BMCFAE> <BMCFAE>}
           |   <id>
           |   {fun {<id>} <BMCFAE>}
           |   {<BMCFAE> <BMCFAE>}
           |   {if0 <BMCFAE> <BMCFAE> <BMCFAE>}
           |   {newbox <BMCFAE>}
           |   {setbox <BMCFAE> <BMCFAE>}
           |   {openbox <BMCFAE>}
           |   {seqn <BMCFAE> <BMCFAE>}
           |   {set <id> <BMCFAE>}
```

# Implementing Variables

```
(define-type DefrdSub
  [mtSub]
  [aSub (name symbol?)
        (address integer?)
        (ds DefrdSub?)])
```

# Implementing Variables

```
; interp : BCFAE DefrdSub Store -> Value*Store
(define (interp expr ds st)
  ...
  [id (name) (v*s (store-lookup (lookup name ds) st)
                  st)]
  ...)
```

# Implementing Variables

```
; interp : BCFAE DefrdSub Store -> Value*Store
(define (interp expr ds st)
  ...
  [app (fun-expr arg-expr)
       (interp-two fun-expr arg-expr ds st
                   (lambda (fun-val arg-val st)
                     (local [(define a (malloc st))]
                       (interp (closureV-body fun-val)
                               (aSub (closureV-param fun-val)
                                     a
                                     (closureV-sc fun-val))
                               (aSto a
                                     arg-val
                                     st)))))]
  ...)
```

# Implementing Variables

```
; interp : BCFAE DefrdSub Store -> Value*Store
(define (interp expr ds st)
  ...
  [set (id val-expr)
       (local [(define a (lookup id ds))]
         (type-case Store*Value (interp val-expr ds st)
           [v*s (val st)
                (v*s val
                     (aSto a
                           val
                           st))]))]
  ...)
```

# Variables and Function Calls

```
(define (swap x y)
  (local [(define z y)]
    (set! y x)
    (set! x z)))

(local [(define a 10)
        (define b 20)]
  (begin
    (swap a b)
    a))
```

Result is **10**; assignment in **swap** cannot affect **a**

# Call-by-Reference

What if we wanted **`swap`** to change **`a`**?

```
(define (swap x y)
  (local [(define z y)]
    (set! y x)
    (set! x z)))

(local [(define a 10)
        (define b 20)]
  (begin
    (swap a b)
    a))
```

⇒

```
(define (swap x y)
  (local [(define z (box (unbox y)))]
    (set-box! y (unbox x))
    (set-box! x (unbox z))))

(local [(define a (box 10))
        (define b (box 20))]
  (begin
    ; (swap (box (unbox a))
    ;       (box (unbox b)))
    (swap a b)
    (unbox a)))
```

This is called ***call-by-reference***, as opposed to ***call-by-value***

*Terminology alert:* this “call-by-value” is orthogonal to the use in “call-by-value” vs. “call-by-name”

# Implementing Call-by-Reference

```
; interp : BCFAE DefrdSub Store -> Value*Store
(define (interp expr ds st)
  ...
  [app (fun-expr arg-expr)
       (if (id? arg-expr)
           ; call-by-ref handling for id arg:
           (type-case Value*Store (interp fun-expr ds st)
             [v*s (fun-val st)
                  (local [(define a
                                  (lookup (id-name arg-expr) ds))]
                    (interp (closureV-body fun-val)
                            (aSub name
                                  a
                                  (closureV-sc fun-val))
                            st))])
           ; as before:
           ...)]
  ...)
```