Tailcalls in 8.6

Miguel Sofer

Tcl2008
Manassas, VA

Tailcalls: what?

- a proc/lambda invokes **tailcall foo [bar] $soom**
- **bar** is invoked and the value of soom fetched, as if **[list foo [bar] $soom]** had been called
- a command named **foo** is looked up in the current context
- the current proc/lambda replaces itself in the call stack with a call to the just found command

Tailcalls: what?

tailcall foo [bar] $soom

is very similar to

return [uplevel 1 [list foo [bar] $soom]]

with two exceptions:

- **foo** is looked up in the current context, not in the caller's
- the stack frame is really gone, not just virtually. This has positive effects on memory and a possibly confusing effect on stack traces.

Tailcalls: what?

A pseudo-implementation (may be useful for debugging purposes: stack trace!):

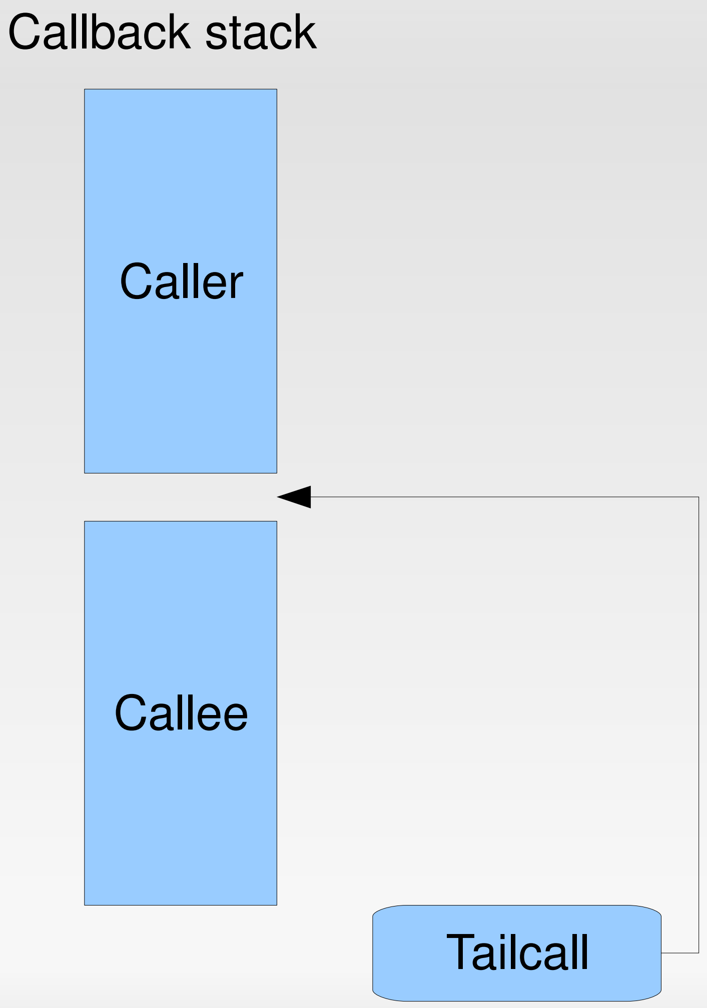
```
proc tailcall args {
  set cmd [uplevel 1 [\
     list namespace which [lindex $args 0]]]
  lset args 0 $cmd
  uplevel 2 $args
}
```

Taicalls: why?

- Infinite recursion depth possible
- Functional Programming constructs
- Pseudo-macros (without the speed gain)
- Continuation Passing Style
- State machines
- ”Delegation”

Tailcalls: tech?

[tailcall] arranges for the command to be run **right after** all callbacks that were scheduled by the callee's implementation


Callback stack
Caller
Callee
Tailcall