TclOO

Past, Present, *Future…*

Donal K. Fellows
University of Manchester / Tcl Core Team

Or "What I've Been Doing for the Past Few Years Instead of Watching Bad TV"…

TclOO: Past

Where it came from, how it was developed

What is TclOO?

- New Standard Object System
- Part of Tcl 8.6
  - Available as extension for Tcl 8.5

```
oo::class create Toaster {
        variable useCount
        constructor {} { set useCount 0 }
        method makeToast {{slices 1}} {
                incr useCount
                for {set i 0} {$i<$slices} {incr i} {
                        puts "made slice $i from use $useCount"
                }
        }
}
set t [Toaster new]
$t makeToast; # → made slice 1 from use 1
```

Why?

- Existing Object Systems had Issues
  - Some were too slow
  - Some were messy internally
  - Some were entangled with class libraries
  - All were poorly integrated with Tcl
    - Except Tk, which is plain inflexible
- TclOO Exists to be a *Small* Object System
  - But deeply integrated with Tcl
  - Allow other OO systems to be built on top

Where Did TclOO Come From?

- Many Principal Lineages
  - Tk
    - General style of method calling
  - [incr Tcl]
    - Way of declaring classes, much syntax
  - XOTcl
    - Semantics of class system
  - Snit
    - Support delegation and scriptability

Who and When?

- First Glimmerings at Tcl 2003
  - Will Duquette and myself
    - Also lead to 8.5's ensembles
- Nearly 2 Years Later (2005)
  - Steve Landers, Will and myself wrote spec
- First prototype by me in 2006
  - Started getting much useful TCT input
- Full prototype in 2007
- Committed to Tcl in 2008
  - Few small things added since with usage experience
- Overall Gestation was 3-5 Years!

Development

- Scripted Prototype
  - Very difficult, so scrapped in favour of...
- CVS Feature Branch of Tcl
  - Written in C
  - Planned for deep integration
  - No unrelated changes

Key Milestones

1. Initial Specification
2. Working Method Invocation
3. Definition Command
4. Starting the Test Suite
5. Generic Method C API
6. Converting to a Package
7. Doing a Release

Lessons Learned...

- Make a Plan
  - Define what success is!
- Study the Competition
  - Steal their good ideas!
- Be Clean in Development
  - No releases until people will be able to run it
  - Don't mix in irrelevant features
- Testing is Vital

TclOO: Present

Features, Use and Performance

What does TclOO Offer?

- Powerful Object System
- Small Feature Set
- Stability
- Deep Integration with Tcl
- Can be Driven from C API
- Fast Core

The Example Again


```
oo::class create Toaster {
    variable useCount
    constructor {} {
        set useCount 0
    }
    method makeToast {{slices 1}} {
        incr useCount
        for {set i 0} {$i < $slices} {incr i} {
            puts "made slice $i from use $useCount”
        }
    }
}

set toastie [Toaster new]
$toastie makeToast;              # → made slice 1 from use 1
```

Making our Toaster Fly with a Mixin


```
oo::class create FlyingObject {
    method takeOff! {} { ... }
    method land {} { ... }
    method getAltitude {} { ... }
}

oo::objdefine $toastie mixin FlyingObject
$toastie takeOff!
$toastie makeToast
```

TclOO Power

- Same Basic Semantic Model as XOTcl
- Single Rooted Multiple Inheritance
  - Subclassable Class Objects in Object System
- Mixins (“ad hoc” classes) and Filters
  - Enables things like Prototypes and Aspects
- Two Types of Methods
  - Procedure-like
  - Forwarded/Delegated

TclOO Features

- As Few as Possible
  - Scriptable and Composable
- Every Object has its own Namespace
  - Holds all variables
- Objects can be Renamed
- Objects can be Reclassed
- Definitions by Scripts
  - Somewhat similar to [incr Tcl] and Snit definitions
- Introspection Integrated into [info]

Scripting: Class Variables


```
proc ::oo::Helpers::classvar {name args} {
    # Get reference to class's namespace
    set ns [info object namespace [uplevel 1 {self class}]]

    # Double up the list of variable names
    set vs [list $name $name]
    foreach v $args {lappend vs $v $v}

    # Link the caller's locals to the class's variables
    tailcall namespace upvar $ns {*}$vs
}
```

Scripting: Class Methods


```
proc ::oo::define::classmethod {name {args ""} {body ""}} {
    # Create the method on the class if the caller gave
    # arguments and body
    if {[llength [info level 0]] == 4} {
        uplevel 1 [list self method $name $args $body]
    }
    # Get the name of the class being defined
    set cls [lindex [info level -1] 1]
    # Make connection to private class “my” command by
    # forwarding
    tailcall forward $name  [info object namespace $cls]::my $name
}
```

Stability and Testing

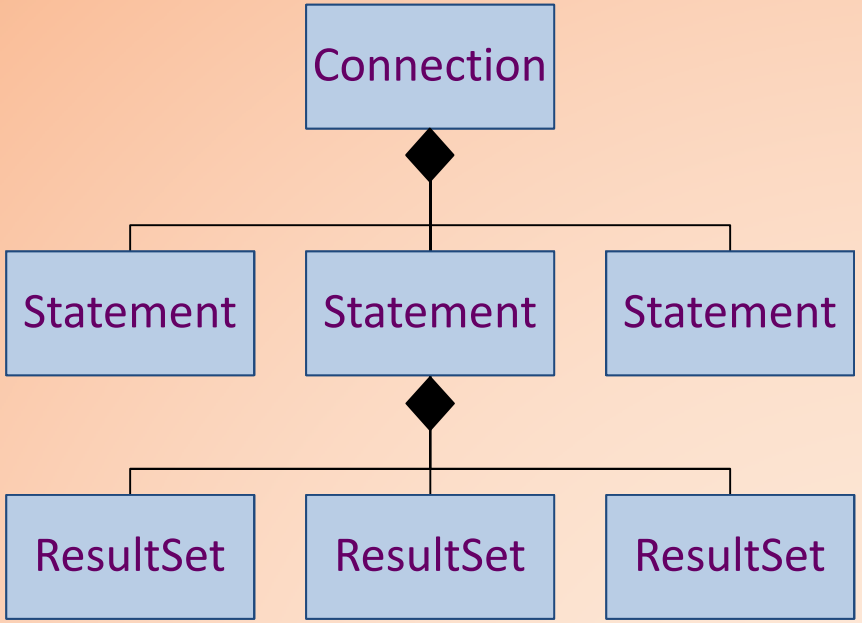
- Test Suite Covers Normal and Error Cases
  - Includes checks for various clean teardowns
  - Includes checks for leaking memory
- Goal: As Stable and Robust as Tcl
  - Should be *no* unchecked failures ever

Production Use of TclOO

- Powers TDBC
  - Also show that TclOO supports UML Composition
- Supports itcl-ng
  - This is [incr Tcl] in Tcl as contrib. package
  - Uses TclOO to provide basic OO framework
- Commercial Uses
  - Ansaldo STS use it in their railway maintenance support product
    - Reported at EuroTcl 2009

Tricks in TDBC: Lifetime

- TDBC uses Cunning Lifetime Management Techniques
  - UML Class Composition
- Based on Ownership
  - Each Statement owned by one Connection
  - Each ResultSet owned by one Statement
- Implemented by Placing Owned into Owner's Namespace
  - Automatic deletion when owner goes away



Tcl Integration

- Available as Package for 8.5
  - Thanks to ActiveState for build support
- Part of Tcl 8.6
  - Fully supports NRE
    - Doesn't blow C stack
    - Can [yield] inside method calls and constructors
- Connection to Tcl Procedure Engine
  - Other OO extensions that use TclOO to implement methods are no longer tightly coupled to Tcl

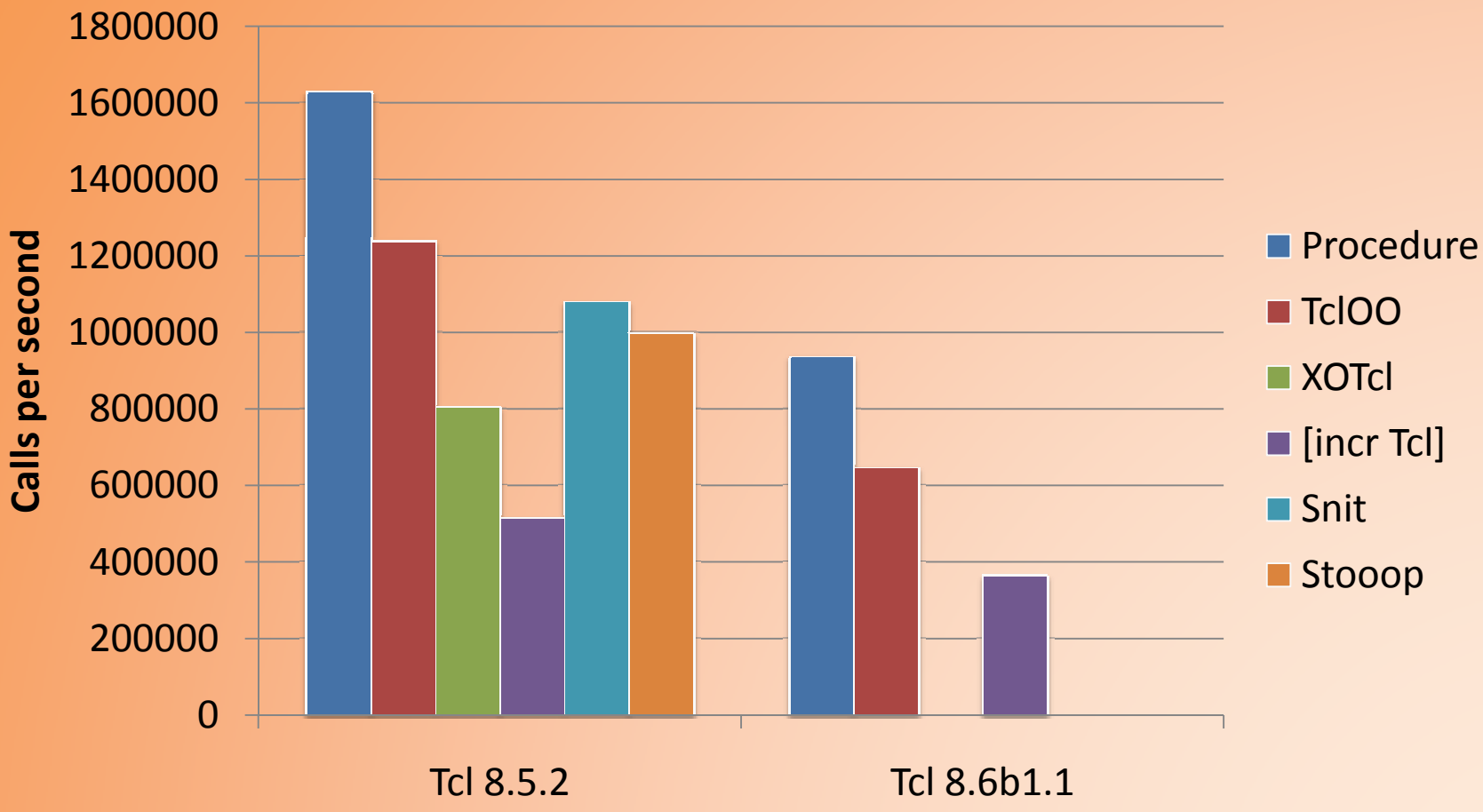
The TclOO C API

- TclOO uses its own C API
  - Generic Method Definition Interface
    - All standard methods built on top of that
  - Construction API
    - No destruction API; use existing facilities
  - Introspection API
    - Information about objects
    - Information about current method
- The C API is a Vital Part of TclOO

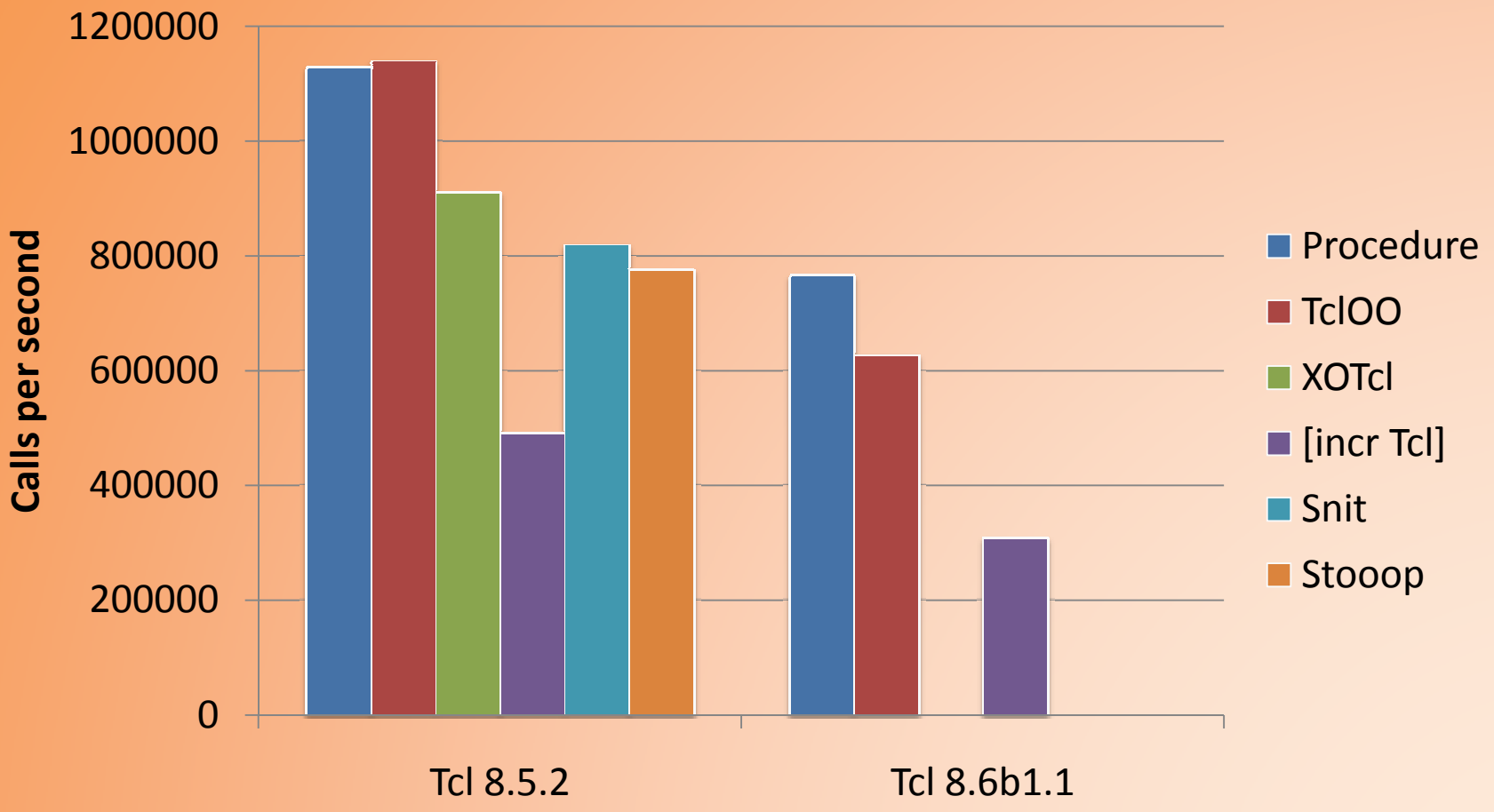
TclOO Performance

- TclOO is Fast
  - Fastest Object System for Tcl
- Massive Amount of Caching
  - Especially of method interpretations
    - In object, class and Tcl_Obj internal representation
  - Caches flushed conservatively
- Critical Paths Analysed to Reduce Hot Allocs
  - Object creation
  - Method call

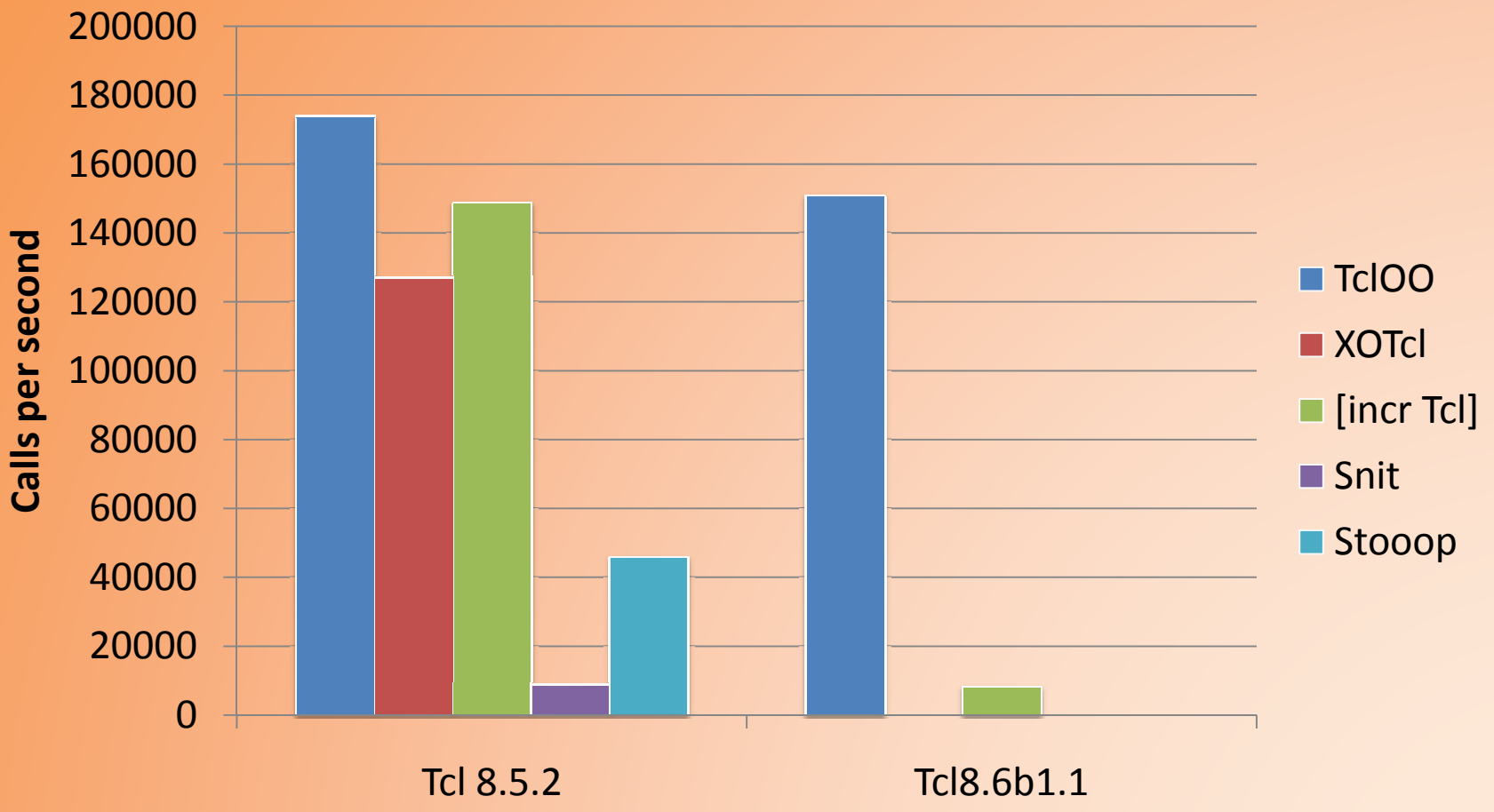
Performance: Basic Call



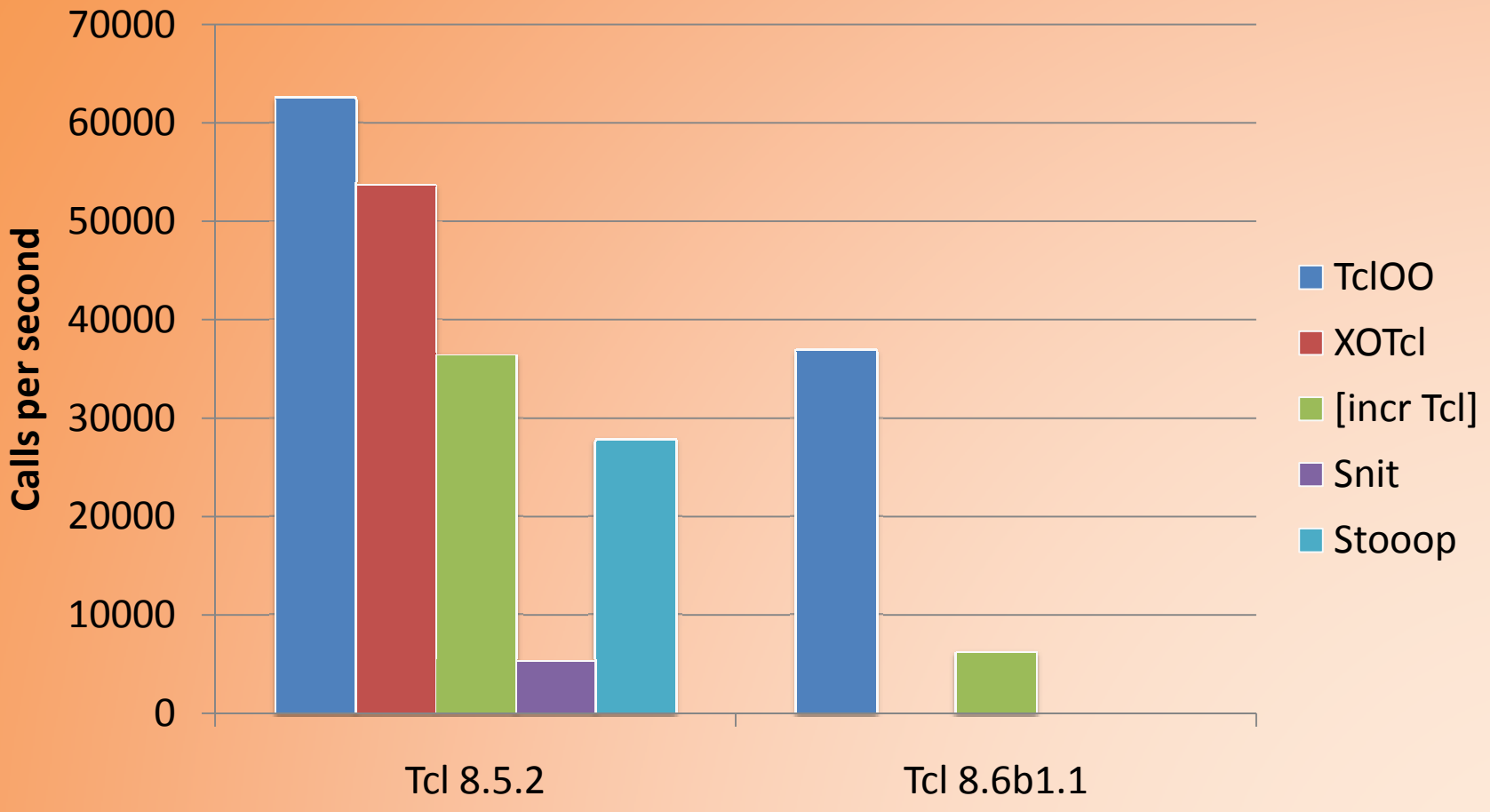
Performance: Stateful Call



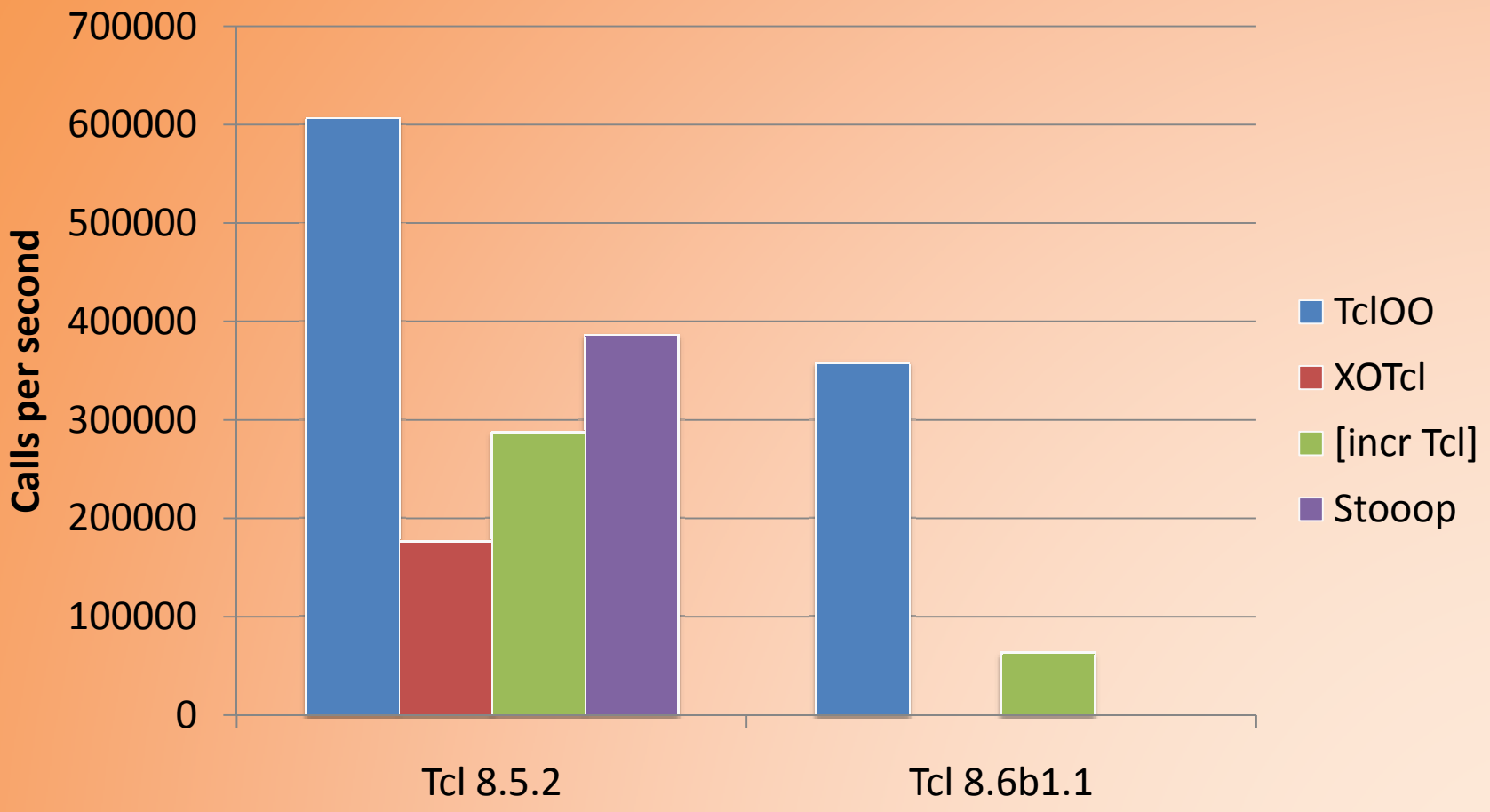
Performance: Create/Delete

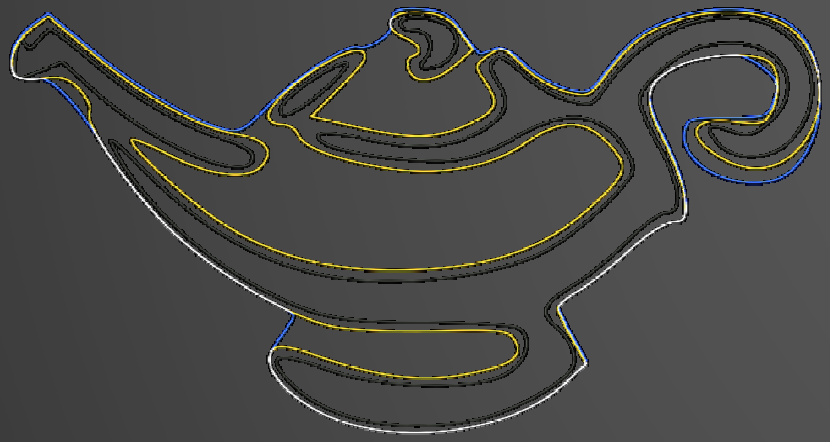


Performance: Make/Call 10/Del.



Performance: Superclass Call





TclOO: Future

Possible future directions

New TclOO Features?

- Garbage Collection
  - Only of unrenamed objects from "*new*" method
  - Problematic because it doesn't fit Tcl's semantics
    - Unlikely to break scripts that weren't leaking
- Submethods
  - More like Tk/Snit
  - Very nice to use, but some tricky issues
    - How does inheritance work?
  - Portable scripts will make method names be single words

Jazzing Up TclOO's Internals

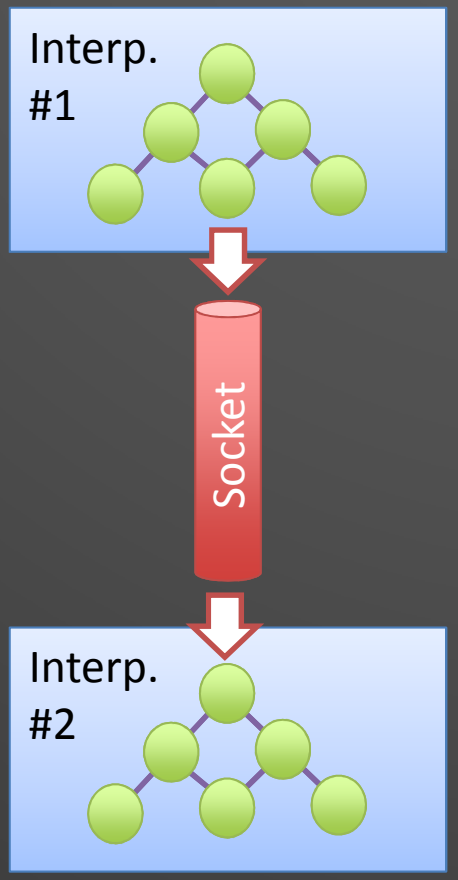
- Slots
  - Better way to manage configuration
    - Likely to cause issues with anything named starting with a "-" character
  - But slots are objects
  - Should methods and variables be objects?
    - Needs investigation
- Poking in the Guts
  - e.g., ways to change how methods are looked up
    - Currently some hacks for itcl-ng; want better…

Building a Class Library

- Already prototyped on Wiki
  - Serialization
  - Channel engine
  - Megawidgets
- How to distribute between packages?
  - Which in Tcl?
  - Which in Tk?
  - Which in Tcllib?

Object Serialization

- Write Objects to a String and Read Back
  - Can also go over sockets or through files or …
- Experimental Package on Wiki
  - http://wiki.tcl.tk/23444
  - Does not serialize classes
  - Does not deal with object name clashes
  - Needs cooperation from objects to explore object graph



Channel Engine Classes

- Classes to Make Writing Channels Easy
  - Based on Andreas Kupries's channel API
  - Both full channels and transformations
- Prototype Code on Wiki
  - http://wiki.tcl.tk/21723
  - Introspection to find what features to support

Megawidgets

- Make Tk Widgets with TclOO Objects
  - Work in “the same” way
  - Wrap actual widgets
- Prototype Code on Wiki
  - http://wiki.tcl.tk/21103
- Already Driven Two Features Added
  - The ‘variable’ declaration
  - Improved method forwarding to object-local commands

Where Next?

- TclOO Intended to be Foundation
  - Fast, light, small, stable, and above all ***Tcl-ish***
  - It deals with the really complicated bits so you don't have to
- Features to Add Should be Community-Driven
  - If you want it, let us know!