Tcl/Tk 2011

Jim Tcl

A Small Footprint Tcl Implementation

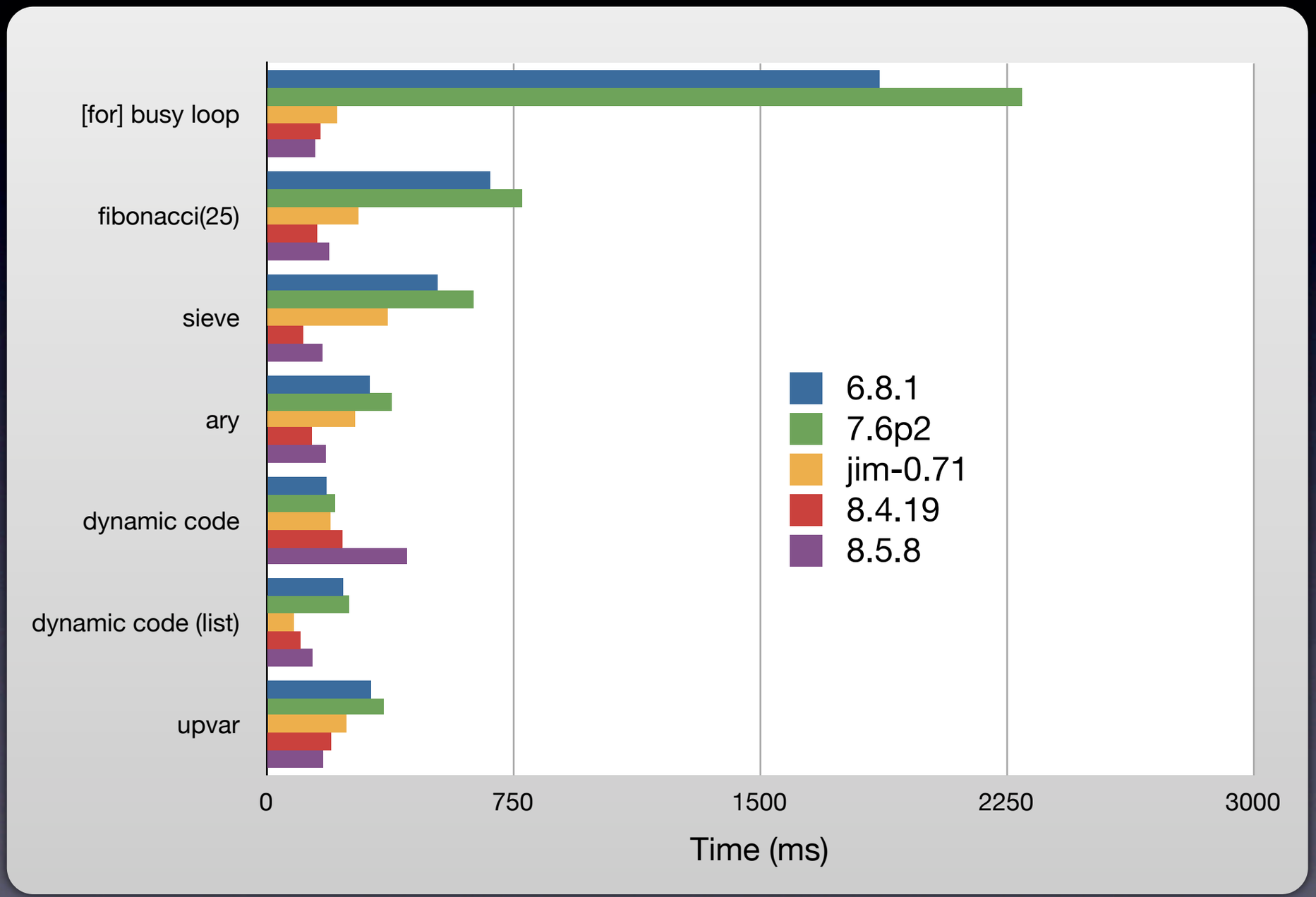
Steve Bennett

WorkWare Systems

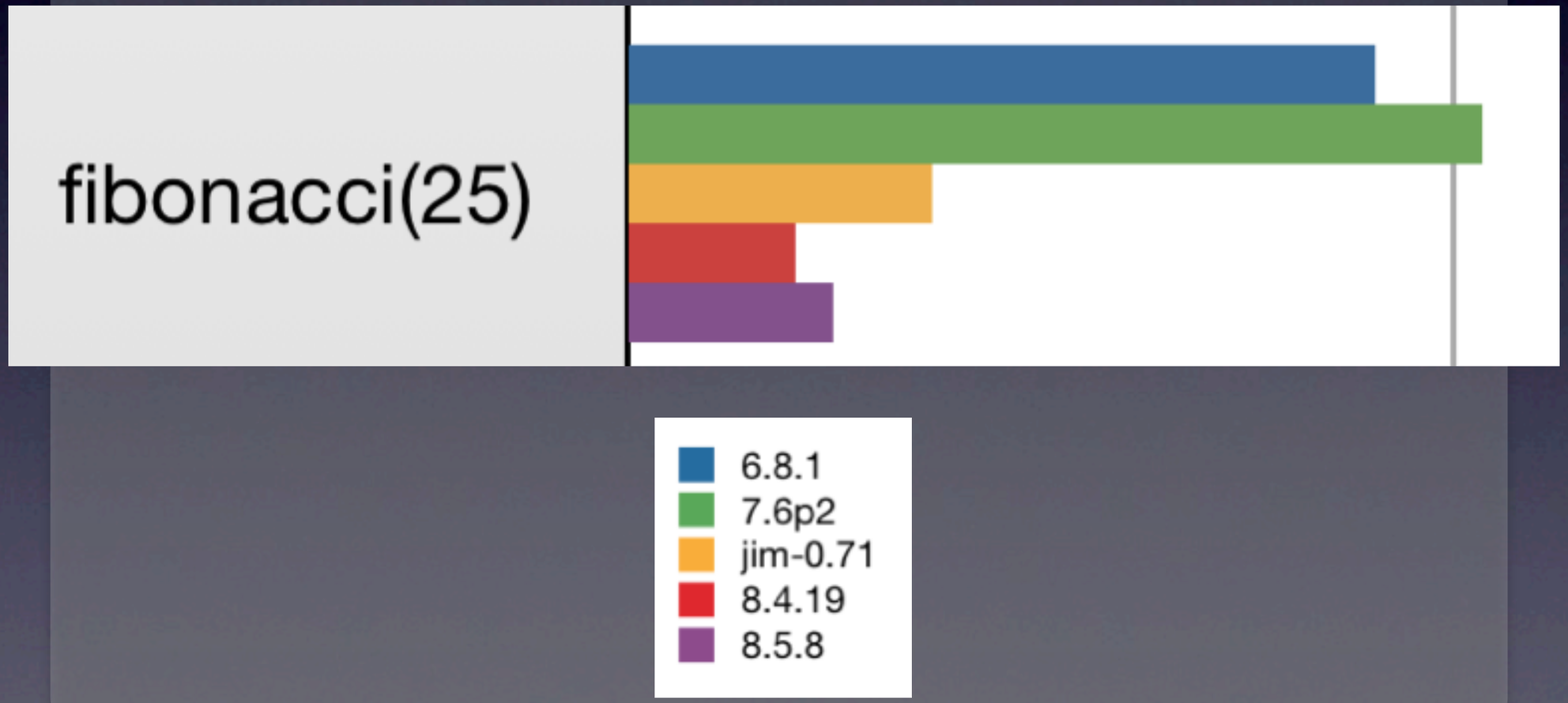
What is Jim Tcl?

- Another C Implementation of the Tcl language (~Tcl 8.6)
- Small, modular
- Does not focus on large Tcl applications, Tk
- Designed for embedded applications
- Functional programming and other enhancements

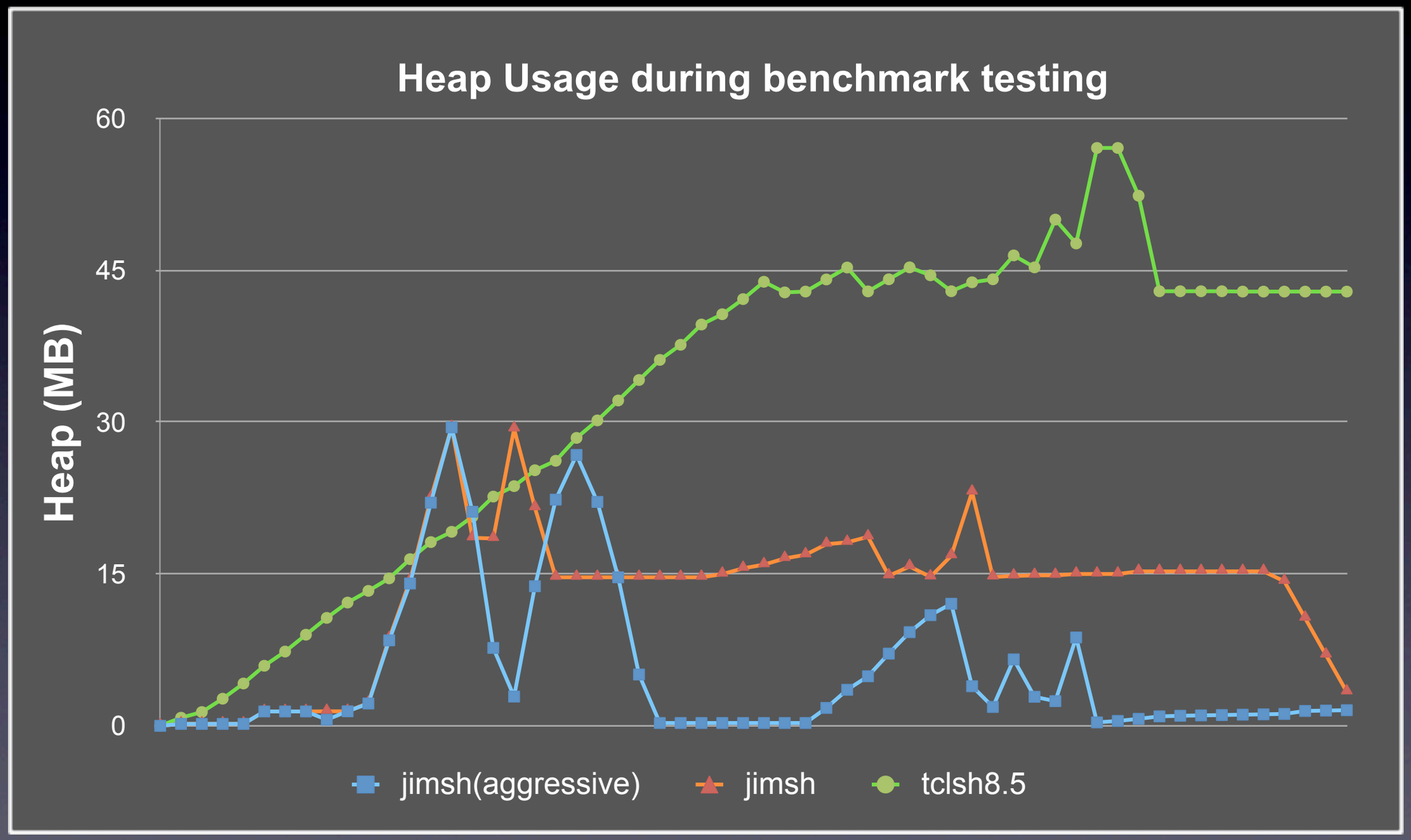
Performance



Performance



Memory Usage



Size

Where possible, Jim Tcl uses system (libc) features to provide a small footprint at the cost of features and/or compatibility

System/Configuration	Size (bytes)
Jim Tcl, system regex	3500
Jim Tcl, built-in regex	9878
Jim Tcl, built-in regex + utf-8	9929
Tcl 8.5.8 regex	54892

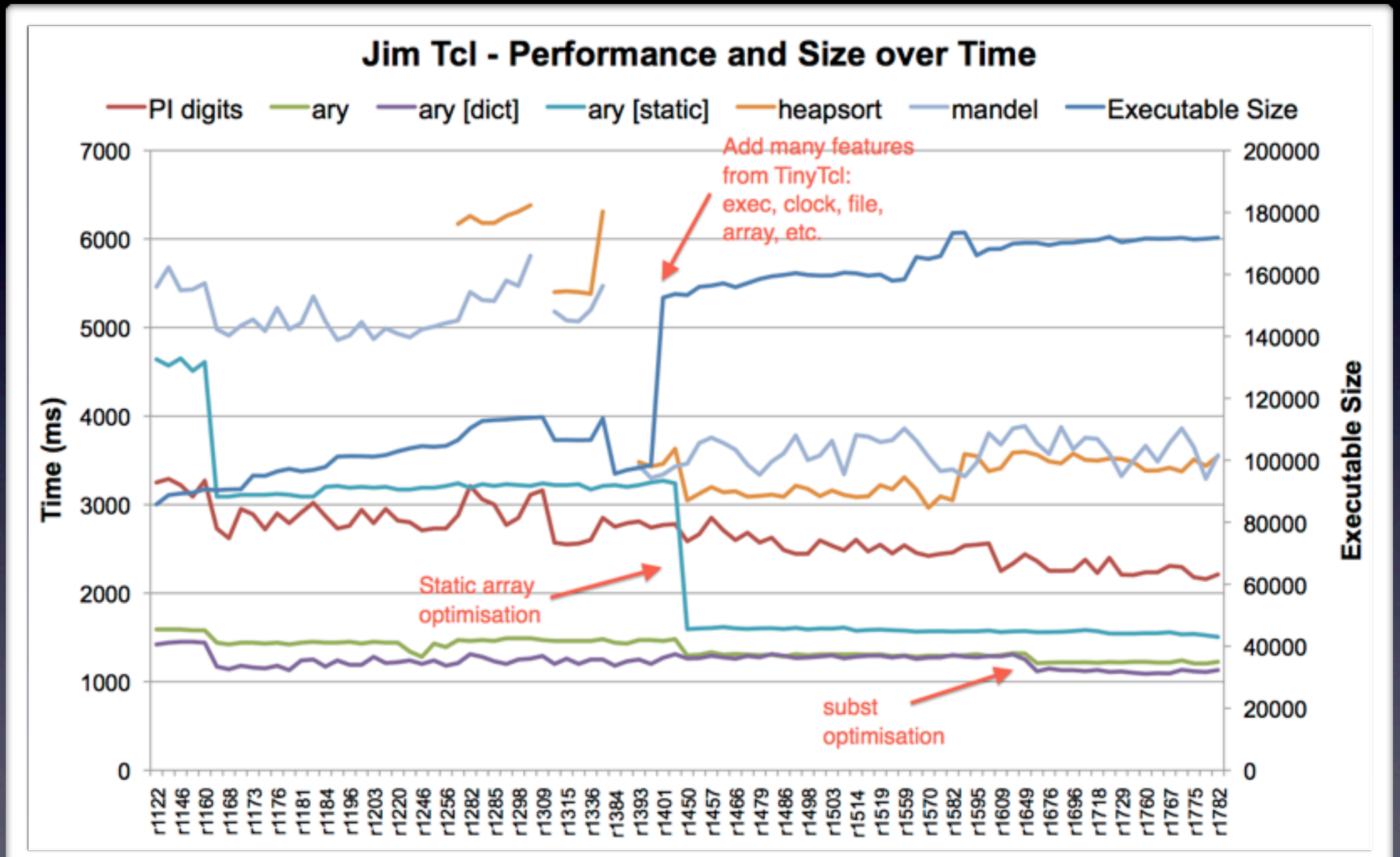
regexp + regsub implementation

Size - Modular

Component	Size
core	89957
tclcompat	4641
load	633
package	1594
readdir	509
glob	2002
array	1696
clock	1437
exec	5290

Component	Size
file	6606
posix	1548
regexp	3508
signal	3689
aio	7629
eventloop	5060
pack	2724
binary	5289
utf-8	16563

History


Jim Tcl - Performance and Size over Time
PI digits
ary
ary [dict]
ary [static]
heapsort
mandel
Executable Size
Add many features from TinyTcl: exec, clock, file, array, etc.
Static array optimisation
subst optimisation
Time (ms)
Executable Size
7000
6000
5000
4000
3000
2000
1000
0
200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
r1122
r1146
r1160
r1168
r1173
r1176
r1181
r1184
r1196
r1203
r1220
r1246
r1256
r1282
r1285
r1298
r1309
r1315
r1336
r1384
r1393
r1401
r1450
r1457
r1466
r1479
r1486
r1498
r1503
r1514
r1519
r1559
r1570
r1582
r1595
r1609
r1649
r1676
r1696
r1718
r1729
r1760
r1767
r1775
r1782

Cross Compile

- Easy to cross compile on many platforms
- Select components, options with 'configure'

```
$ ./configure --ipv6 --with-ext=binary --host=arm-linux
Host System...arm-unknown-linux-gnu
Build System...x86_64-apple-darwin11.1.0
C compiler...ccache arm-linux-gcc -g -O2
C++ compiler...ccache arm-linux-c++ -g -O2
Build C compiler...cc
Checking for stdlib.h...ok
Checking for long long...ok
..etc..
$ make
```

Source Tracking

- Tcl is a very dynamic language
- Tracking source location is not trivial
- Error messages can be hard to interpret

```
$ tclsh8.6 dbgtest.tcl
can't use non-numeric string as operand of "+"
    while executing
"expr 1+$x"
    (procedure "p4" line 3)
    invoked from within
"p4 y"
    (procedure "p2" line 4)
    invoked from within
"p2"
    (procedure "p1" line 3)
    invoked from within
"p1"
    (file "dbgtest.tcl" line 33)
```

Source Tracking

Jim Tcl gives absolute line numbers

```
$ jimsh dbgtest.tcl
Runtime Error: dbgtest.tcl:8: syntax error in expression: "1+y"
in procedure 'p1' called at file "dbgtest.tcl", line 33
in procedure 'p2' called at file "dbgtest.tcl", line 28
in procedure 'p4' called at file "dbgtest.tcl", line 22
at file "dbgtest.tcl", line 8
```

- Error messages are easier to interpret
- Source location introspection allows for parsers, debuggers, code coverage tools

Source Tracking

Token #	Token Type	Token Value	Object Type
[0]	LIN		scriptline line=1
[1]	ESC	set	source (test.tcl:1)
[2]	ESC	x	source (test.tcl:1)
[3]	ESC	abc	source (test.tcl:1)
[4]	LIN		scriptline line=2
[5]	ESC	if	source (test.tcl:2)
[6]	STR	[string match -x* $x]	source (test.tcl:2)
[7]	STR	\nputs "$x matches"\n	source (test.tcl:2)
[8]	ESC	else	source (test.tcl:4)
[9]	STR	\nputs "$x does not match"\n	source (test.tcl:4)

```
1: set x abc
2: if {[string match -x* $x]} {
3:     puts "$x matches"
4: } else {
5:     puts "$x does not match"
6: }
```

test.tcl

Initial parse preserves source location of each token plus the line

Source Tracking

Token #	Token Type	Token Value	Object Type
[0]	LIN		scriptline line=1
[1]	ESC	set	command
[2]	ESC	x	variable
[3]	ESC	abc	source (test.tcl:1)
[4]	LIN		scriptline line=2
[5]	ESC	if	command
[6]	STR	[string match -x* $x]	expression
[7]	STR	\nputs "$x matches"\n	source (test.tcl:2)
[8]	ESC	else	compared-string
[9]	STR	\nputs "$x does not match"\n	script (test.tcl:4)

```
1: set x abc
2: if {[string match -x* $x]} {
3:     puts "$x matches"
4: } else {
5:     puts "$x does not match"
6: }
```

test.tcl

As script is evaluated, tokens are converted to internal representation

Source Tracking

Token #	Token Type	Token Value	Object Type
		...	
[9]	STR	\nputs "$x does not match"\n	script (test.tcl:4)

```
1: set x abc
2: if {[string match -x* $x]} {
3:     puts "$x matches"
4: } else {
5:     puts "$x does not match"
6: }
```

test.tcl

Token #	Token Type	Token Value	Object Type
[0]	LIN		scriptline line=5
[1]	ESC	puts	source (test.tcl:5)
[2]	ESC	$x does not match	source (test.tcl:5)

Token #	Token Type	Token Value	Object Type
[0]	LIN		scriptline line=5
[1]	ESC	puts	command
[2]	ESC	$x does not match	source (test.tcl:5)

Source information propagates as scripts are parsed and evaluated

Source Tracking

Access source information for any string

```
 1: # test3.tcl
 2: puts [info source {}]
 3:
 4: proc a {} {
 5: }
 6:
 7: puts [info source [info body a]]
 8:
 9: set b {
10:     one
11:     two
12:     three
13: }
14: puts [info source [lindex $b 1]]
```

Source Tracking

Access source information for any string

```
 1: # test3.tcl
 2: puts [info source {}]
 3:
 4: proc a {} {
 5: }
 6:
 7: puts [info source [info body a]]
 8:
 9: set b {
10:     one
11:     two
12:     three
13: }
14: puts [info source [lindex $b 1]]
```

➡ Location of current line

```
test3.tcl 2
```

➡ Location of proc body

```
test3.tcl 4
```

➡ Location of list element

```
test3.tcl 11
```

Source Tracking

```
$ ./jimdb test.tcl
Jim Tcl debugger v1.0  -  Use ? for help

@ test.tcl:1 set x abc
>    1 set x abc
     2 if {[string match -x* $x]} {
dbg> n
=> abc
@ test.tcl:2 if {[string match -x* $x]} ...
     1 set x abc
>    2 if {[string match -x* $x]} {
     3    puts "$x matches"
dbg> p $x
abc
dbg> ?
 s        step into    w        where
 n        step over    l [loc] list source
 r        step out     v        local vars
 c        continue     u        up frame
 p [exp] print         d        down frame
 b [loc] breakpoints  t [n]    trace
 ? [cmd] help         q        quit
dbg> l alias
@ stdlib.tcl
     1 # Create a single word alias (proc)
     2 # e.g. alias x info exists
     3 # if {[x var]} ...
*    4 proc alias {name args} {
     5    set prefix $args
```

Pure-Tcl debugger can display source location and set source-based breakpoints

Source Tracking

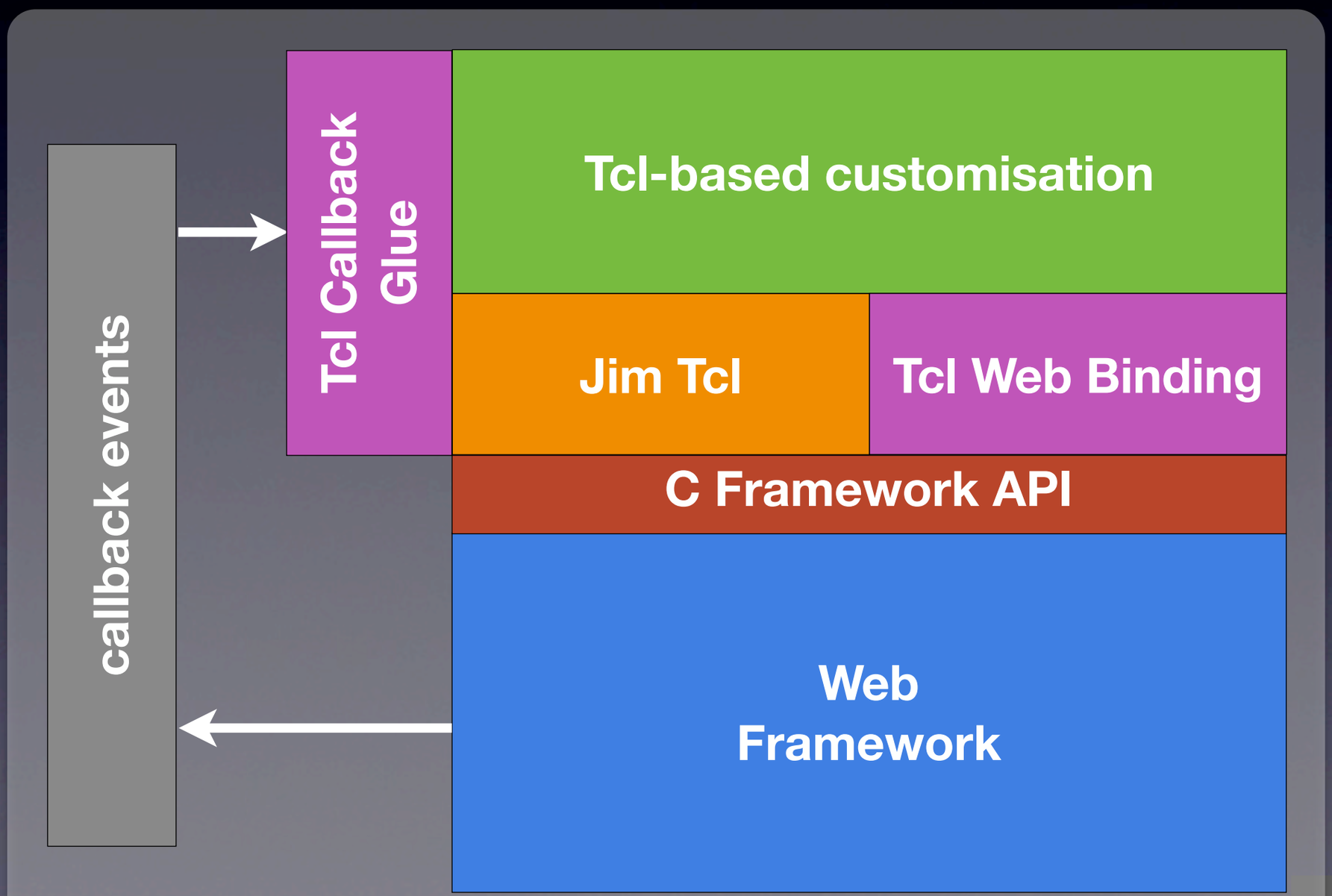
```
$ ./jcov testcov.tcl
a(1)  = 2
a(4)  = 1

   1: foreach i {abc def ghi} {
   3:    switch -glob -- $i {
####:       {[a-d]*} {
   2:          incr a(1)
   -:       }
####:       def {
####:          incr a(2)
   -:       }
####:       g*h {
####:          incr a(3)
   -:       }
####:       g*i {
   1:          incr a(4)
   -:       }
####:       default {
####:          incr a(5)
   -:       }
   -:    }
   -: }
   -:
   2: parray a
```

Source information can be used to track code coverage in pure-Tcl

Source Tracking

µWeb Embedded Web Framework



µWeb Source Location Preservation with Jim Tcl

The Jim Tcl interpreter for the target platform is linked into the application.

libjim

The µWeb compiler is a Jim Tcl script. It uses the live stack trace information to provide source-accurate error messages and also ‘info source’ to record the original source location of “scriptlets”.

µWeb “compiler”

generated C code

C compiler, Linker

```
static const struct elem_button_t elem15[] = {
   {
      ...
      .submit_script.script = "\n"
"cgi success \"Message log cleared\"\n"
"file delete /var/log/messages\n"
"\n",
      .submit_script.filename = “syslog.page”,
      .submit_script.line = 41,
   }
};
```

page files

Page files are Tcl scripts parsed as a DSL. They include “scriptlets” which are executed at runtime

Web Application

“scriptlets” are executed at runtime by the Jim Tcl interpreter via Jim_Eval_Named(). Runtime errors can therefore provide accurate source information.

```
37: button clear {
38:    label "Clear Log"
39:    help "Clear the log display"
40:    editmode newline
41:    submit -tcl {
42:       cgi success "Message log cleared"
43:       file delete /var/log/messages
44:    }
45: }
```

µWeb

Page files are Tcl scripts

Jim Tcl parser gives accurate error messages

Tcl/Tk 2011

WorkWare Systems

µWeb Source Location Preservation with Jim Tcl

The Jim Tcl interpreter for the target platform is linked into the application.

libjim

The µWeb compiler is a Jim Tcl script. It uses the live stack trace information to provide source-accurate error messages and also ‘info source’ to record the original source location of “scriptlets”.

µWeb “compiler”

generated C code

C compiler, Linker

```
static const struct elem_button_t elem15[] = {
   {
        ...
        .submit_script.script = "\n"
"cgi success \"Message log cleared\"\n"
"file delete /var/log/messages\n"
"\n",
        .submit_script.filename = “syslog.page”,
        .submit_script.line = 41,
   }
};
```

page files

Page files are Tcl scripts parsed as a DSL. They include “scriptlets” which are executed at runtime

Web Application

“scriptlets” are executed at runtime by the Jim Tcl interpreter via Jim_Eval_Named(). Runtime errors can therefore provide accurate source information.

```
37: button clear {
38:    label "Clear Log"
39:    help "Clear the log display"
40:    editmode newline
41:    submit -tcl {
42:       cgi success "Message log cleared"
43:       file delete /var/log/messages
44:    }
45: }
```

µWeb

Source information of embedded Tcl Scripts is preserved within the generated C code

Tcl/Tk 2011

WorkWare Systems

µWeb Source Location Preservation with Jim Tcl

The Jim Tcl interpreter for the target platform is linked into the application.

libjim

The µWeb compiler is a Jim Tcl script. It uses the live stack trace information to provide source-accurate error messages and also ‘info source’ to record the original source location of “scriptlets”.

µWeb “compiler”

generated C code

```
static const struct elem_button_t elem15[] = {
    {
        ...
        .submit_script.script = "\n"
"cgi success \"Message log cleared\"\n"
"file delete /var/log/messages\n"
"\n",
        .submit_script.filename = “syslog.page”,
        .submit_script.line = 41,
    }
};
```

C compiler, Linker

page files

Page files are Tcl scripts parsed as a DSL. They include “scriptlets” which are executed at runtime

Web Application

“scriptlets” are executed at runtime by the Jim Tcl interpreter via Jim_Eval_Named(). Runtime errors can therefore provide accurate source information.

```
37: button clear {
38:     label "Clear Log"
39:     help "Clear the log display"
40:     editmode newline
41:     submit -tcl {
42:         cgi success "Message log cleared"
43:         file delete /var/log/messages
44:     }
45: }
```

µWeb

The Jim Tcl runtime uses the source location for accurate error messages

Tcl/Tk 2011

WorkWare Systems

Open On-Chip Debugger

Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing

- Jim Tcl as configuration, commands
- Provides a full-featured, well-known language
- Easy build integration, cross compilation

```
jtag newtap $_CHIPNAME cpu -irlen 4 -ircapture 0x1 \
  -irmask 0xf -expected-id $_CPUTAPID
set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME arm7tdmi -endian $_ENDIAN \
  -chain-position $_TARGETNAME -variant arm7tdmi

$_TARGETNAME configure -event reset-start {
    # start off real slow when we're running off internal RC oscillator
    jtag_khz 32
}
proc peek32 {address} {
    mem2array t 32 $address 1
    return $t(0)
}
# Wait for an expression to be true with a timeout
proc wait_state {expression} {
    for {set i 0} {$i < 1000} {incr i} {
        if {[uplevel 1 $expression] == 0} {
            return
        }
    }
    return -code error "Timed out"
}
```

autosetup

A build environment "autoconfigurator"

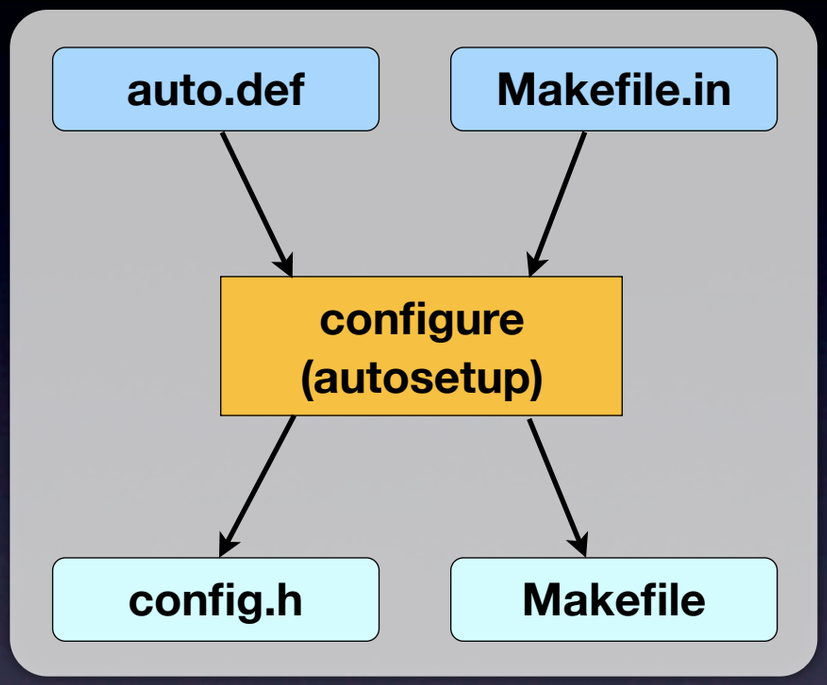
- autoconf replacement

  "Tcl 8.5.8 configure is 20162 lines long"

- Written in Tcl

- Tcl8.5 or Jim Tcl

- Includes bootstrap Jim Tcl, single source file Tcl interpreter




```
$ ./configure
No installed jimsh or tclsh, building local bootstrap jimsh0
Host System...i686-pc-mingw32
Build System...i686-pc-mingw32
C compiler... gcc -g -O2
C++ compiler... c++ -g -O2
```

- Used by Fossil SCM, Jim Tcl

Lambdas, Garbage Collection and more

Jim Tcl allows procs with static variables

```
. proc a {x} {{adder 5}} {
    return [incr x $adder]
  }
. a 3
8
```

And garbage-collected references

```
. set r [ref "One String" test]
<reference.<test___>.00000000000000000000>
```

With finalizers (destructors)

```
. finalize $r myfinalizer
. set r ""
. collect
myfinalizer called with <reference.<test___>.00000000000 123
```

Lambdas, Garbage Collection and more

Which allows garbage-collected lambdas

```
# Implementation of lambda with Jim Tcl
proc lambda {arglist args} {
    set name [ref {} func lambda.finalizer]
    tailcall proc $name $arglist {*}$args
}
proc lambda.finalizer {name val} {
    rename $name {}
}
```

Which can be used like any other command

```
. set list {1 50 20 -4 2}
1 50 20 -4 2
. lsort -command [lambda {a b} {expr {$a - $b}}] $list
-4 1 2 20 50
```

Lambdas, Garbage Collection and more

Lambdas can include static variables, thus creating closures

```
. proc make-adder {x} {
    lambda p x { incr p $x }
  }
. set add5 [make-adder 5]
. $add5 10
15
. $add5 3
8
```

Jim Tcl Unique Features

- array/dict/list conversion
- built-in line editing
- modular, optional utf-8
- object-oriented I/O
- garbage collected references
- proc "static" variables
- accurate source tracking
- signal handling
- stacking local procs, upcall
- 64 bit integers
- expr shorthand: $(...)
- simplified packaging system
- proc &upvar
- proc default args in any order, rename args
- udp, IPV6, unix domain sockets, pipes

http://jim.berlios.de/