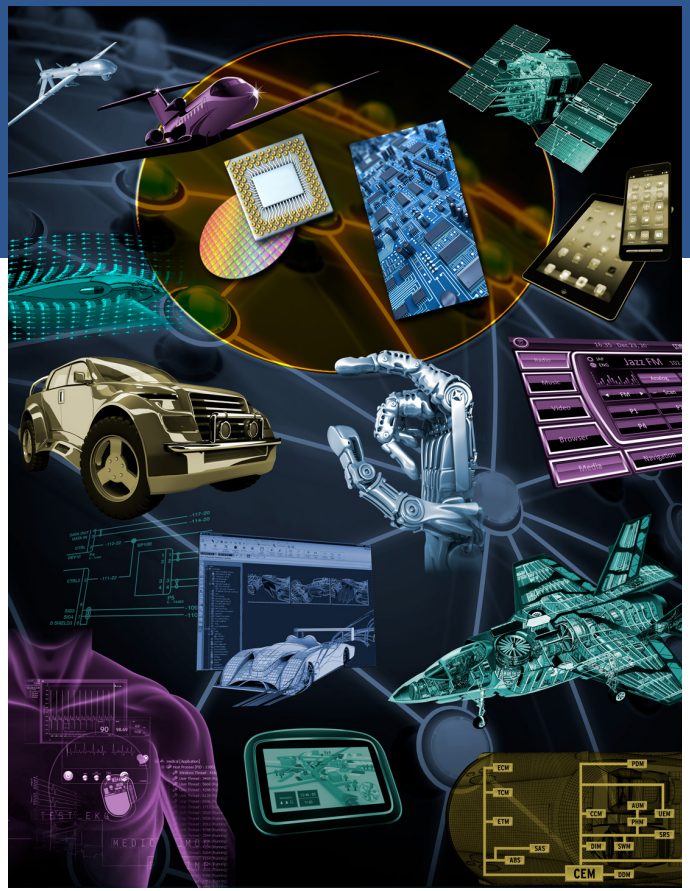


Tcl and the Qt Event Loop

Brian S Griffin

Using Tcl in a Qt Application

IC Verification Solutions

June 2019

Mentor®
A Siemens Business

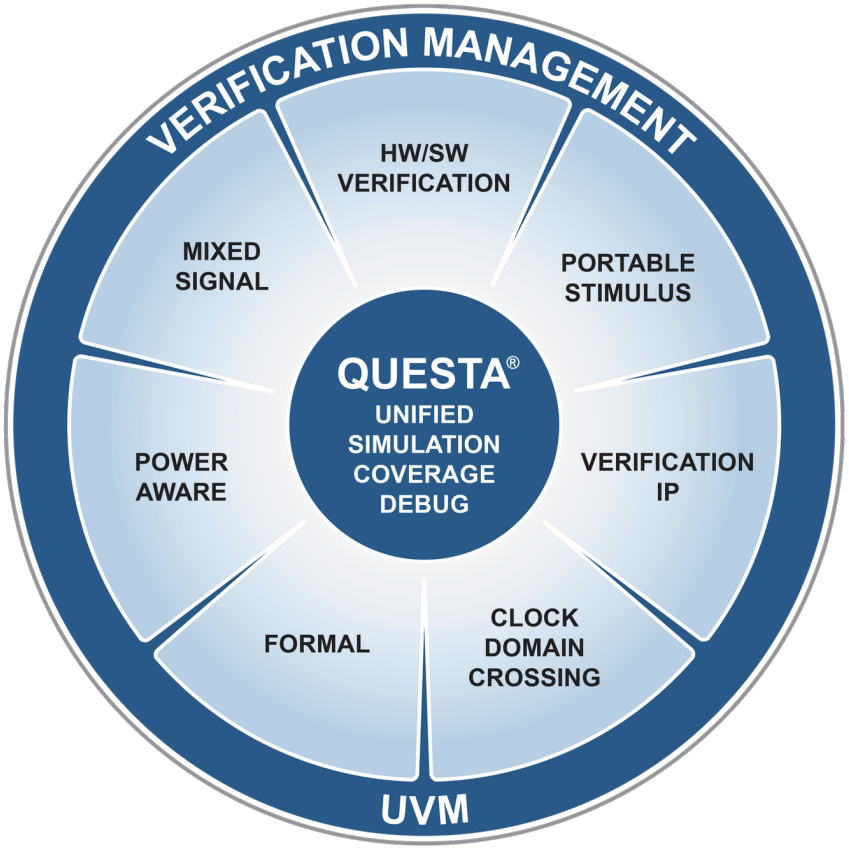
A little bit about me

- Mentor Graphics - A Siemens Business
- HDL Simulation
  - Debugger
  - Infrastructure (plumbing)
  - GUI
- Using Tcl/Tk since ~1996
- TCT member since 2012

IC Verification Solutions

Questa Simulation

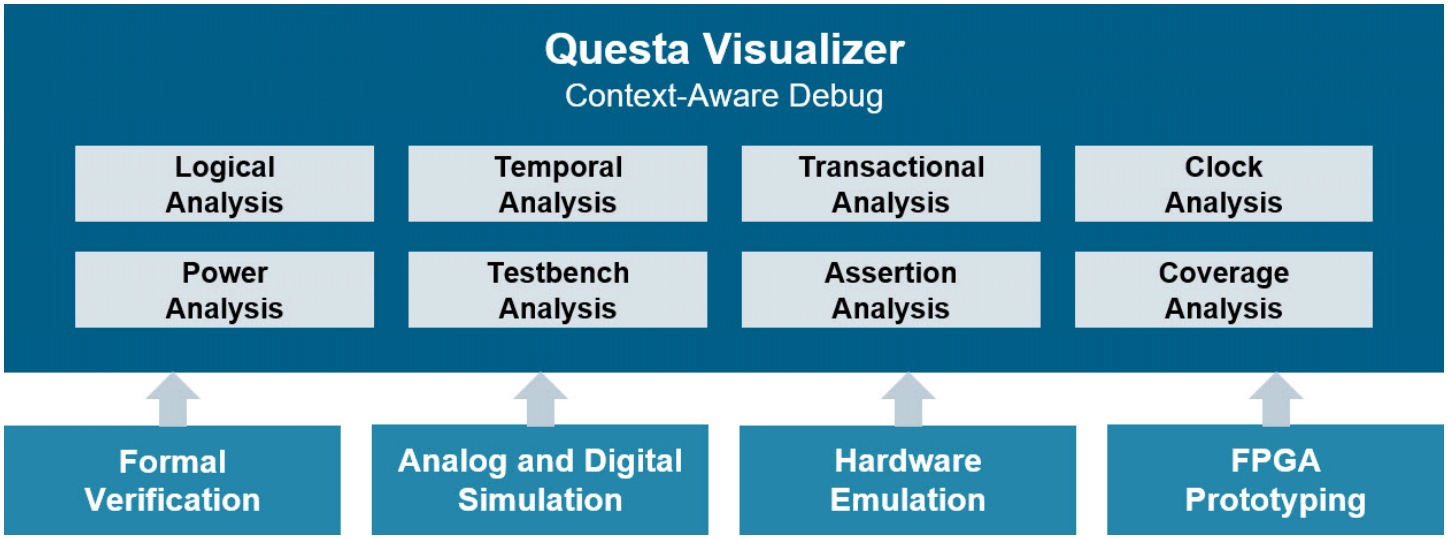
- RTL Simulation
- Mixed Signal
- HW/SW verification
- Power Domain verification
- Formal analysis
- Clock Domain verification
- Portable Stimulus
- Verification IP
- Gate thru SoC
- …



Mentor® A Siemens Business

GUI debug environment

- Design data + debug analysis engines + Qt based GUI
- Analyze results post-simulation
- Interactive debug of a running simulation
- Debug
  - Device Under Test
  - Testbench
  - Coverage
  - with 3rd party add-ons



The Problem

- New Qt based debug environment
  - Provided limited scripting control via Tcl
  - No general purpose way to interface with external tools or services
- Existing Tcl integration has problems with
  - [after]
  - [vwait]
  - [fileevent]
- Because
  - Qt event loop unaware of pending Tcl events
  - Tcl event loop unaware of pending Qt events

Solution

- Replace Tcl's native event loop with Qt's event loop
- Tcl is designed for this

1) Read the Tcl_SetNotifier() man page.

2) Implement the Notifier using Qt.

3) Have a beer.

- However, the devil is in the details.

What is the Event Loop?

- Pseudo code for Tcl_DoOneEvent(flags)

```
forever {
  if (ServiceEvent()) { break }
  foreach Event-Source { src->setup() }
  WaitForEvent( WAIT ? &blocktime : NULL )
  foreach Event-Source { src->check() }
  if (ServiceEvent()) { break }
  if (ServiceIdle()) { break }
  if ( ! WAIT ) { break }
}
```

What's in the Notifier

```
typedef struct Tcl_NotifierProcs {
            Tcl_SetTimerProc *setTimerProc;
         Tcl_WaitForEventProc *waitForEventProc;
    Tcl_CreateFileHandlerProc *createFileHandlerProc;
    Tcl_DeleteFileHandlerProc *deleteFileHandlerProc;
         Tcl_InitNotifierProc *initNotifierProc;
     Tcl_FinalizeNotifierProc *finalizeNotifierProc;
        Tcl_AlertNotifierProc *alertNotifierProc;
      Tcl_ServiceModeHookProc *serviceModeHookProc;
} Tcl_NotifierProcs;
```

Notifier Procs — Timer Proc

```
set ahndl [after 1000 updateClock]
```

```
typedef struct Tcl_NotifierProcs {
            Tcl_SetTimerProc *setTimerProc;
        Tcl_WaitForEventProc *waitForEventProc;
   Tcl_CreateFileHandlerProc *createFileHandlerProc;
   Tcl_DeleteFileHandlerProc *deleteFileHandlerProc;
        Tcl_InitNotifierProc *initNotifierProc;
    Tcl_FinalizeNotifierProc *finalizeNotifierProc;
       Tcl_AlertNotifierProc *alertNotifierProc;
     Tcl_ServiceModeHookProc *serviceModeHookProc;
} Tcl_NotifierProcs;
```

SetTimerProc

Tcl (unix):

```
if (notifier.currentTimeout != 0) {
  XtRemoveTimeOut(notifier.currentTimeout);
}

if (timePtr) {
  timeout = timePtr->sec * 1000 +
            timePtr->usec / 1000;
  notifier.currentTimeout =
    XtAppAddTimeOut(notifier.appContext,
    (unsigned long) timeout,
    TimerProc, NULL);
} else {
  notifier.currentTimeout = 0;
}
```

Qt:

```
if ( ! timer) {
  timer = new QTimer(this);
  connect(timer, SIGNAL(timeout()),
          SLOT(TimerProc(void)));
}

timer->stop();

if (timePtr) {
  long timeout = timePtr->sec * 1000 +
                 timePtr->usec / 1000;
  currentTimeout = timer->start(timeout, true);
}
```

SetTimerProc

Tcl (unix):

```
static void
TimerProc(
  XtPointer clientData, /* Not used. */
  XtIntervalId *id)
{
  if (*id != notifier.currentTimeout) {
    return;
  }
  notifier.currentTimeout = 0;

  Tcl_ServiceAll();
}
```

Qt:

```
void
qtclNotifierCls::Timerproc(
  void)

{
  if (timerPtr) {
    timerPtr->stop();
    theTclNotifier->myexit(-2);
 }
}
```

Notifier Procs — Create File Handler Proc

fileevent $fd **readable** readit

```
typedef struct Tcl_NotifierProcs {
            Tcl_SetTimerProc *setTimerProc;
         Tcl_WaitForEventProc *waitForEventProc;
    Tcl_CreateFileHandlerProc *createFileHandlerProc;
    Tcl_DeleteFileHandlerProc *deleteFileHandlerProc;
         Tcl_InitNotifierProc *initNotifierProc;
     Tcl_FinalizeNotifierProc *finalizeNotifierProc;
        Tcl_AlertNotifierProc *alertNotifierProc;
      Tcl_ServiceModeHookProc *serviceModeHookProc;
} Tcl_NotifierProcs;
```

CreateFilehandlerProc

Tcl (unix):

```
FileHandler *filePtr;

for (filePtr = tsdPtr->firstFileHandlerPtr;
     filePtr != NULL;
     filePtr = filePtr->nextPtr) {
  if (filePtr->fd == fd) {
    break;
  }
}
if (filePtr == NULL) {
  filePtr = ckalloc(sizeof(FileHandler));
  filePtr->fd = fd;
  filePtr->readyMask = 0;
  filePtr->nextPtr = tsdPtr->firstFileHandlerPtr;
  tsdPtr->firstFileHandlerPtr = filePtr;
}
filePtr->proc = proc;
filePtr->clientData = clientData;
filePtr->mask = mask;
```

Qt:

```
filePtr = FirstFileHandlerPtr ?
          FirstFileHandlerPtr->GetHandler(fd) :
          new tclFileHandlerCls(fd,
                        &FirstFileHandlerPtr);

if (mask & TCL_READABLE) {
  if (!(filePtr->mask & TCL_READABLE)) {
    filePtr->SetNotifier(QSocketNotifier::Read,
                         mask, proc, clientData);
  }
} else {
  if (filePtr->mask & TCL_READABLE) {
    delete filePtr->read;
    filePtr->read = 0;
  }
}
// ... repeat for each mode
```

CreateFilehandlerProc (continued)

Qt:

```
void tclFileHandler::SetNotifier(QSocketNotifier::Type type, int mask,
                                 Tcl_FileProc *proc, ClientData clientData)
{
      switch (type) {
        case QSocketNotifier::Read:
            read = new QSocketNotifier(fd, type, this, "tclReadNotifier");
            read->setEnabled(true);
            connect(read, SIGNAL(activated(int)), SLOT(FileProc(int)));
            break;
     // ... all other modes

      }
// ...
}
```

CreateFilehandlerProc (continued)

Tcl (unix):

```
if (filePtr->readyMask == 0) {
  FileHandlerEvent *fileEvPtr =
         ckalloc(sizeof(FileHandlerEvent));
  fileEvPtr->header.proc =
                        FileHandlerEventProc;
  fileEvPtr->fd = filePtr->fd;
  Tcl_QueueEvent((Tcl_Event *) fileEvPtr,
                             TCL_QUEUE_TAIL);
}
filePtr->readyMask = mask;
```

Qt:

```
void tclFileHandlerCls::FileProc (int xFd)
{
 …
 filePtr->readyMask |= mask;
 FileHandlerEvent *fileEvPtr =
    (FileHandlerEvent *)ckalloc(
                      sizeof(FileHandlerEvent));
 fileEvPtr->header.proc = FileHandlerEventProc;
 fileEvPtr->fd = lFilePtr->fd;
 Tcl_QueueEvent((Tcl_Event *) lFileEvPtr,
                TCL_QUEUE_TAIL);
```

The FileHandlerEventProc is practically identical to Tcl's FileHandlerEventProc.

Notifier Procs — Delete File Handler Proc

filehandler $fp **readable {}**

```
typedef struct Tcl_NotifierProcs {
            Tcl_SetTimerProc *setTimerProc;
         Tcl_WaitForEventProc *waitForEventProc;
    Tcl_CreateFileHandlerProc *createFileHandlerProc;
    Tcl_DeleteFileHandlerProc *deleteFileHandlerProc;
         Tcl_InitNotifierProc *initNotifierProc;
     Tcl_FinalizeNotifierProc *finalizeNotifierProc;
        Tcl_AlertNotifierProc *alertNotifierProc;
      Tcl_ServiceModeHookProc *serviceModeHookProc;
} Tcl_NotifierProcs;
```

DeleteFilehandlerProc

Qt:

```
filePtr = FirstFileHandlerPtr ? FirstFileHandlerPtr->GetHandler(fd) : NULL;

if (filePtr) {
  FirstFileHandlerPtr = FirstFileHandlerPtr->removeHandler(fd);
  if (filePtr->read) delete filePtr->read;
  if (filePtr->write) delete filePtr->write;
  if (filePtr->except) delete filePtr->except;
  delete lFilePtr;
}
```

Notifier Procs

```
typedef struct Tcl_NotifierProcs {
            Tcl_SetTimerProc *setTimerProc;
         Tcl_WaitForEventProc *waitForEventProc;
    Tcl_CreateFileHandlerProc *createFileHandlerProc;
    Tcl_DeleteFileHandlerProc *deleteFileHandlerProc;
         Tcl_InitNotifierProc *initNotifierProc;
     Tcl_FinalizeNotifierProc *finalizeNotifierProc;
        Tcl_AlertNotifierProc *alertNotifierProc;
      Tcl_ServiceModeHookProc *serviceModeHookProc;
} Tcl_NotifierProcs;
```

WaitForEventProc

From the man page:

Ideally, **Tcl_WaitForEvent** should only wait for an event to occur;
it should not actually process the event in any way.

Later on, the event sources will process the raw events and create Tcl_Events on the event queue in their checkProc procedures.

WaitForEventProc (continued)

Qt:

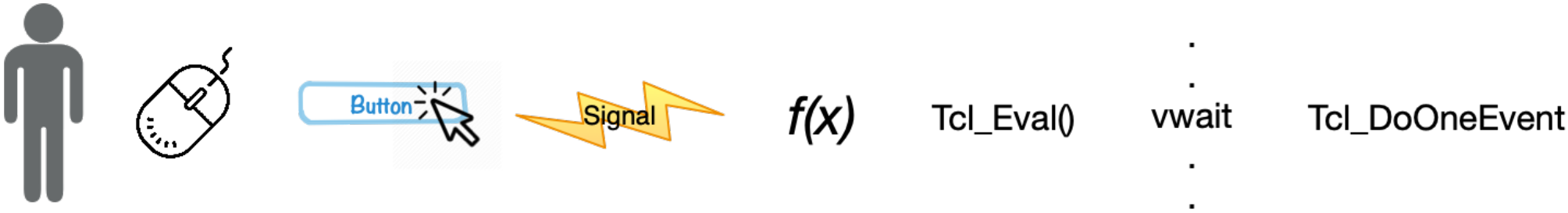
```
QEventLoop *waitLoop = new QEventLoop();

if (timeout && !app->inExit()) {
  // Call Qt to do the waiting, with a timeout determined by Tcl
  QTimer::singleShot(timeout, QtclNotifier, SLOT(myexit()));
  didOne = waitLoop->exec(QEventLoop::WaitForMoreEvents);
} else {
  didOne = waitLoop->processEvents(QEventLoop::AllEvents);
}

return ! didOne;
```

How does it all work?

- Qt event loop gets started by calling QApplication->**exec**() method
- Tcl event loop gets started by calling Tcl_DoOneEvent(), usually via [vwait]
- Since we have a Qt Application, the Qt event loop will be started first via the **exec**() method
- The Tcl event loop gets involved by way of something like this:



WaitForEventProc (continued)

Qt:

To handle recursion correctly, keep track of nested event loops so the timeout will exit the appropriate one.

```
QEventLoop *prevWaitLoop = currentWaitLoop;
QEventLoop *waitLoop = new QEventLoop();
currentWaitLoop = waitLoop;
if (timeout && !app->inExit()) {
  QTimer::singleShot(timeout, QtclNotifier, SLOT(myexit()));
  didOne = waitLoop->exec(QEventLoop::WaitForMoreEvents);
} else {
  didOne = waitLoop->processEvents(QEventLoop::AllEvents);
}
currentWaitLoop = prevWaitLoop;
return ! didOne;
```

WaitForEventProc (continued)

Qt:

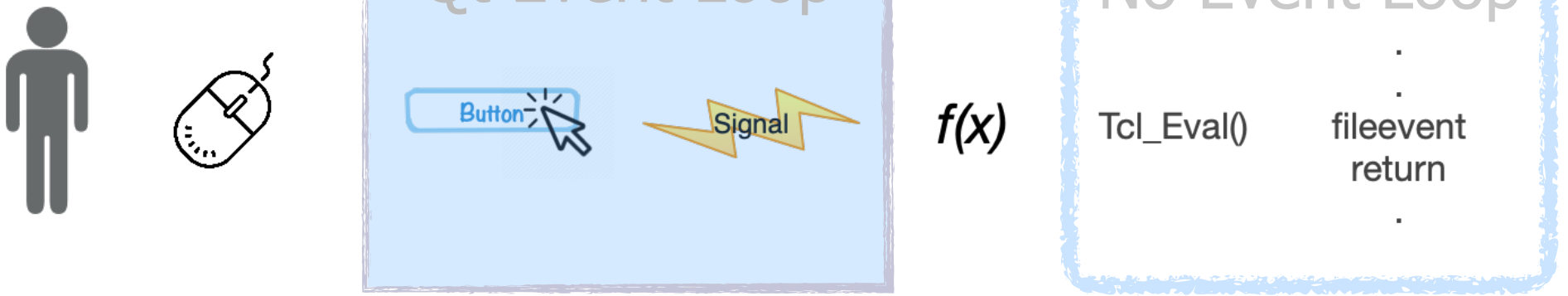
```
void qtclNotifierCls::myexit(int status = -1)
{
    if (currentWaitLoop) currentWaitLoop->exit(status);
}
```

From Qt documentation:

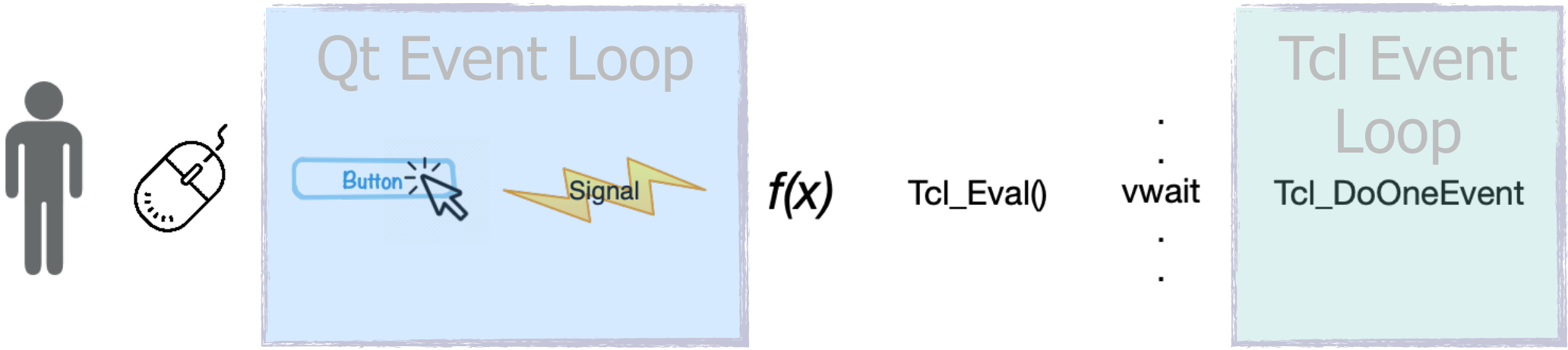
void QEventLoop::exit(int *returnCode* = 0)

Tells the event loop to exit with a return code.
After this function has been called, the event loop returns from the call to exec(). The exec() function returns *returnCode*.
By convention, a *returnCode* of 0 means success, and any non-zero value indicates an error.
Note that unlike the C library function of the same name, this function *does* return to the caller -- it is event processing that stops.

No recursion

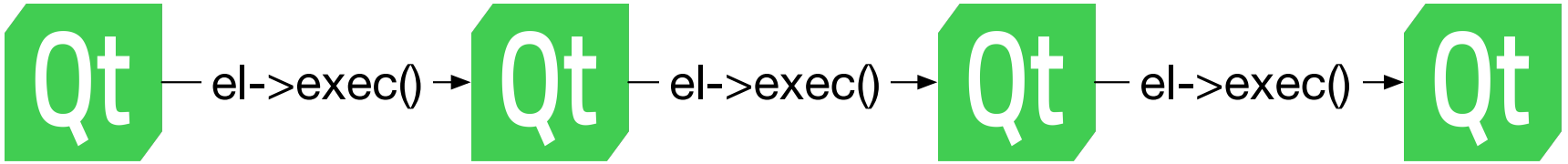


Event Loop Recursion



Event Loop Recursion (continued)

Qt allows for recursive calls to the event loop. — This does happen in applications.



Tcl also allows for recursion.

So too, recursion happens when mixing the two

Mentor® A Siemens Business

Event Loop Recursion (continued)

- WaitForEvent is designed to work recursively — hence the use of the a local QEventLoop / exit strategy
- There is, however, one last issue to address

Getting to Tcl

- Events queued for the Tcl event loop will only be processed when Tcl_DoOneEvent() is called
- Qt applications run from app->exec()
- The default ->exec() method will not call Tcl_DoOneEvent()
- This means idle tasks and timer events may never be processed
- To solve this, app->exec() must be overloaded to call Tcl_DoOneEvent() instead of Qt's event loop exec() method.

```
int myApplication::exec()
{
    while ( ! inExit ) {
        Tcl_DoOneEvent(TCL_ALL_EVENTS);
    }
    return exit_status;
}
```

What did we forget?

Notifier Procs

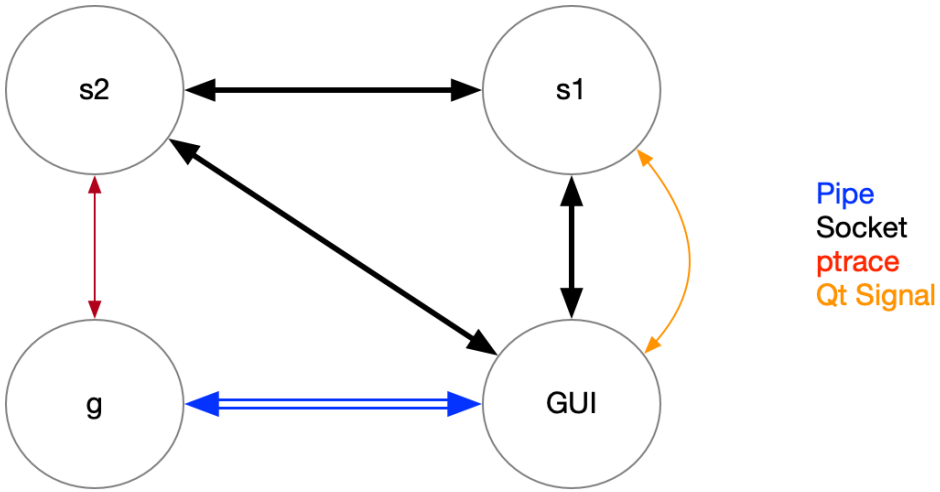
```
typedef struct Tcl_NotifierProcs {
            Tcl_SetTimerProc *setTimerProc;
         Tcl_WaitForEventProc *waitForEventProc;
    Tcl_CreateFileHandlerProc *createFileHandlerProc;
    Tcl_DeleteFileHandlerProc *deleteFileHandlerProc;
         Tcl_InitNotifierProc *initNotifierProc;
     Tcl_FinalizeNotifierProc *finalizeNotifierProc;
        Tcl_AlertNotifierProc *alertNotifierProc;
      Tcl_ServiceModeHookProc *serviceModeHookProc;
} Tcl_NotifierProcs;
```

The Remaining Procs

- initNotifierProc — Creates notifier state, calls SetNotifier, and returns the notifier state.
- serviceModeHookProc — save the service mode value
- alertNotifierProc — leave NULL, handled by Tcl
- finalizeNotifierProc — clean up all your resources (i.e. delete theQtclNotifier)

Results

- Successfully reuse Tcl with minor changes
- Drop-in Tcl-based RPC library with no changes
- Recently ”stress tested” configuration with 4 different processes using RPC (sockets), command pipes, and Qt Signals.



A Siemens Business

www.mentor.com