AI & Robotics

GUIs continued

PXL AI & ROBOTICS LAB

Goals

The **junior-colleague**

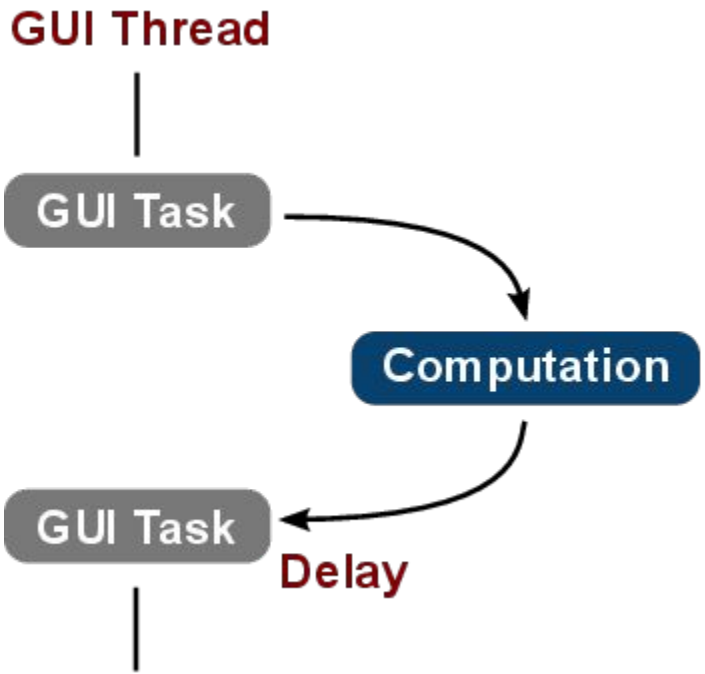
- can create multithreaded GUIs using QT and Python.

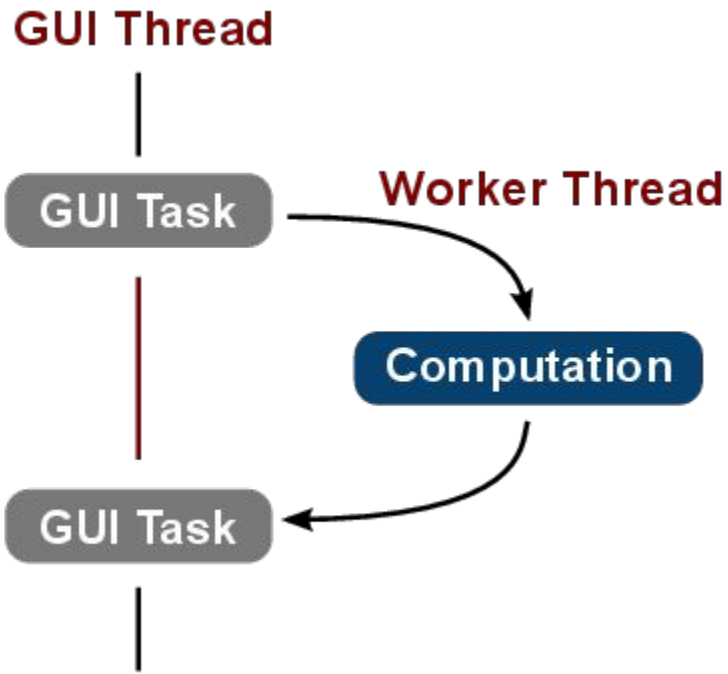
GUIs and threads

- Important to keep track of your threads when GUI programming

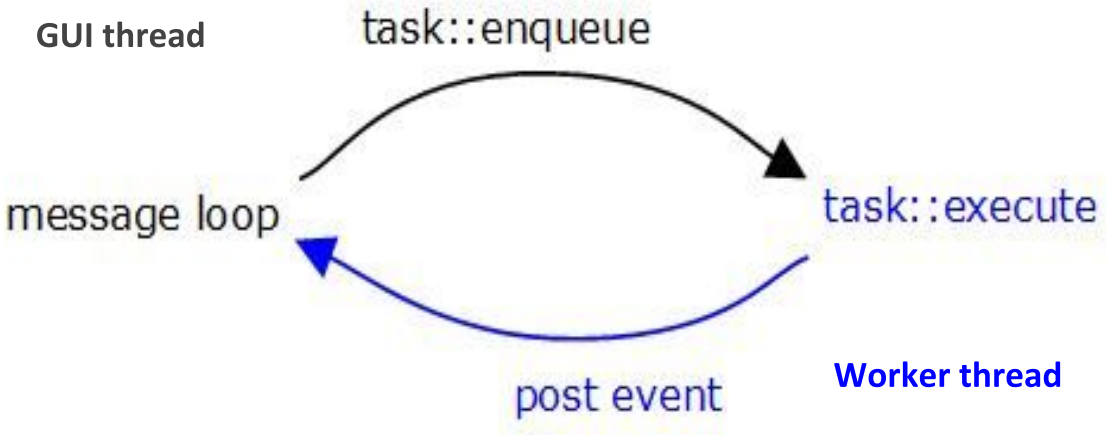
  => user interface thread must remain responsive to user requests
- The GUI thread:
  - services an event loop
  - needs to offload work onto other threads without waiting for the work to complete
  - must be responsive to the event loop and not become dedicated to doing the offloaded work
  - Is usually the main thread (but not always!)

GUIs and threads





GUIs and threads



Qt Event loop

```
while (is_active)
{
 while (!event_queue_is_empty)
 dispatch_next_event();

wait_for_more_events();
}
```

- wait_for_more_events() function blocks until some event is generated
- Reacts to:
  - Paint events
  - Timer events
  - User actions
  - …

Qt Event loop

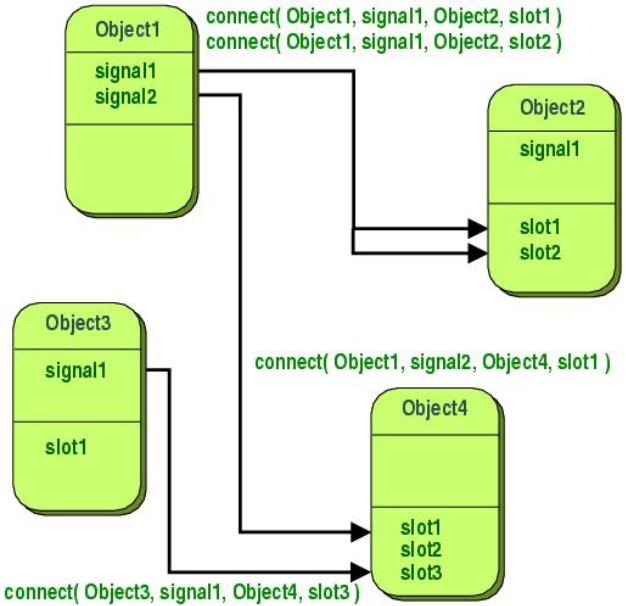
```
while (is_active)
{
 while (!event_queue_is_empty)
 dispatch_next_event();

wait_for_more_events();
}
```

- GUI thread creates main event loop
- Qt forces GUI-related operations to only happen in the GUI thread

Qt Signals & Slots

- Used to support event-driven nature of a GUI application
- Functions or methods are executed in response to events:
  - Clicking a button
  - Key press
  - Timer
- Widgets:
  - Source of these events
  - Can emit “signal”

Qt Signals & Slots




```
QtCore.QObject.connect(widget,
QtCore.SIGNAL('signalname'), slot_function)

Or

widget.signal.connect(slot_function)
```

Qt Signals & Slots: example


```
Button 1 clicked
Button 2 clicked
```

```python
def window():
   app = QApplication(sys.argv)
   win = QDialog()
   b1 = QPushButton(win)
   b1.setText("Button1")
   b1.move(50,20)
   b1.clicked.connect(b1_clicked)

   b2 = QPushButton(win)
   b2.setText("Button2")
   b2.move(50,50)
   QObject.connect(b2,SIGNAL("clicked()"),b2_clicked)

   win.setGeometry(100,100,200,100)
   win.setWindowTitle("PyQt")
   win.show()
   sys.exit(app.exec_())

def b1_clicked():
   print "Button 1 clicked"

def b2_clicked():
   print "Button 2 clicked"

if __name__ == '__main__':
   window()
```

Qt Signals & Slots across threads

```
class WorkerThread : public QThread
{
    Q_OBJECT
    void run() {
        QString result;
        /* expensive or blocking operation  */
        emit resultReady(result);
    }
signals:
    void resultReady(const QString &s);
};

void MyObject::startWorkInAThread()
{
    WorkerThread *workerThread = new WorkerThread(this);
    connect(workerThread, SIGNAL(resultReady(QString)), this, SLOT(handleResults(QString)));
    connect(workerThread, SIGNAL(finished()), workerThread, SLOT(deleteLater()));
    workerThread->start();
}
```

[INFO] https://wiki.qt.io/Threads_Events_QObjects

Qt Signals & Slots across threads

- **Direct connection**: slot is always invoked directly by the thread the signal is emitted from
- **Queued connection** an event is posted in the event queue of the thread the receiver is living in, which will be picked up by the event loop and will cause the slot invocation sometime later
- **Blocking queued connection:** like a queued connection, but the sender thread blocks until the event is picked up by the event loop of the thread the receiver is living in, the slot is invoked, and it returns
- **Automatic connection** (the default): if the thread the receiver is living in is the same as the current thread, a direct connection is used; otherwise, a queued connection is used.

[INFO] https://wiki.qt.io/Threads_Events_QObjects