For some time I was thinking about Mirek`s question on CoWork. And couldn`t find how to apply queue approach to Reference/CoWork example natively. The only way I think of is QueueCoWork as pool manager for QueueThread objects which decides which Thread should execute next action depending on their business (i.e. queue lengths). On highest hierarchy level QueueCoWork is received PaintLines message from main thread`s Paint. This event executes main class callback function DoRepaint which manages pooling of low-level callbacks painting the lines. IMO, looks quite ugly:

```
void QueueCoWork::Manage(Callback1 &cb)
{
    int mostFreeThread = -1;
    //find most free (unbusy) Thread

    queueThreads[mostFreeThread] << cb;
}

void QueueCoWork::ManageTypical(Callback1 &cb)
{
    //simply rotate through threads to average tasks count
    lastUsedThread = (++lastUsedThread) % queueThreads.GetCount();

    queueThreads[lastUsedThread] << cb;
}

//---------------------------------------------------
void MainWindow::OnPaint
{
    queueCoWork << THISBACK(PaintLines);
}

void MainWindow::PaintLines()
{
    for (int y=0; y<height; ++y)
        queueCoWork.ManageTypical(THISBACK1(PaintLine, y));
}
void MainWindow::PaintLine(int y)
{
    //paint the y-th line
}
```