

Медленные алгоритмы и интерфейс пользователя

Простейшая анимация

С помощью задержки

```
#define FRAME_TIME 500

while(вычисление не закончено)
{
    ОдинШагВычислений();
    ОтрисоватьНовоеСостояние();

    QThread::msleep(FRAME_TIME);
}
```

С точным контролем времени

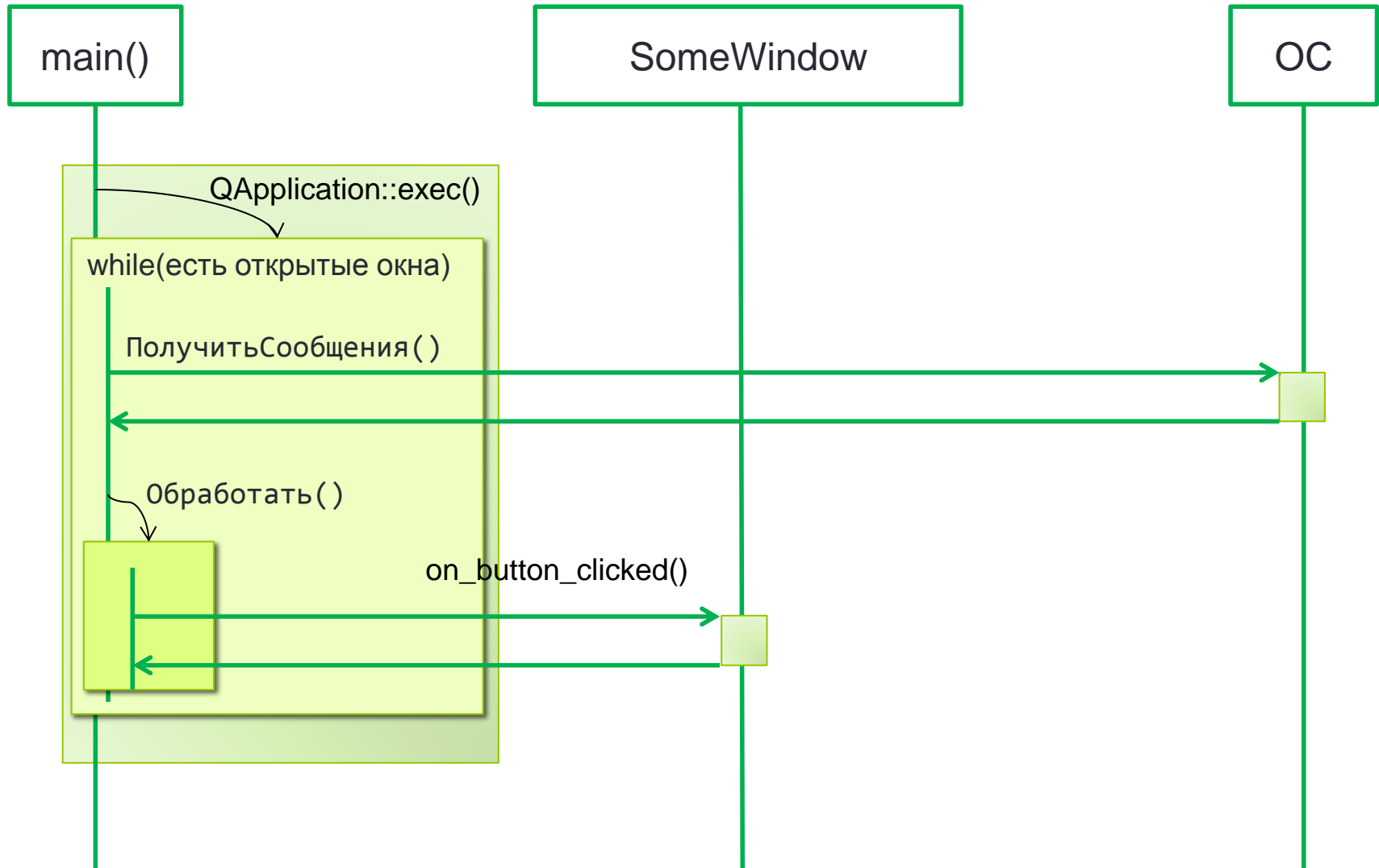
```
#define FRAME_TIME 500
QElapsedTimer timer;

while(вычисление не закончено)
{
    timer.start();

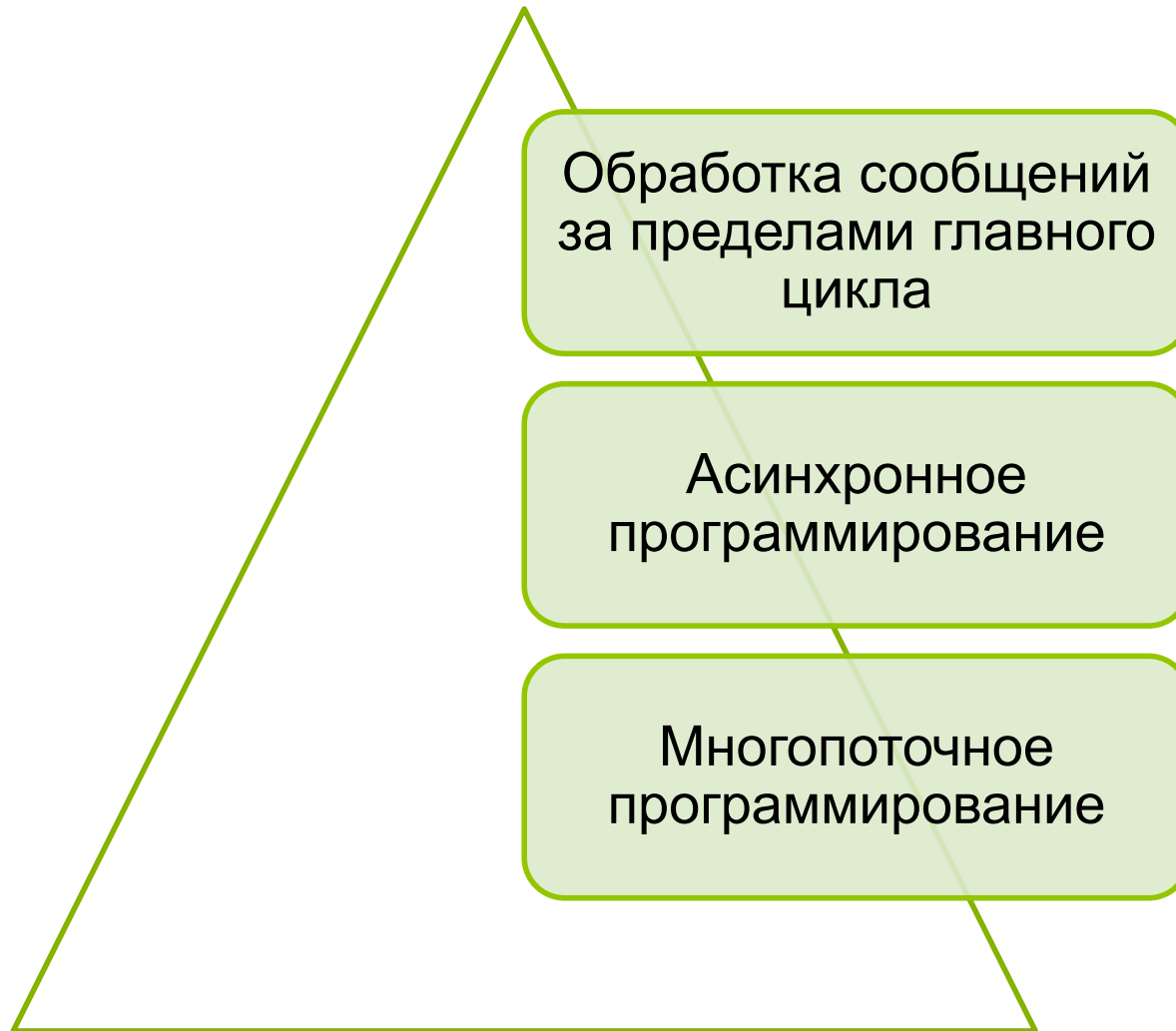
    ОдинШагВычислений();
    ОтрисоватьНовоеСостояние();

    int delay=FRAME_TIME-timer.elapsed();
    if(delay>0)
        QThread::msleep(delay);
}
```

Графический интерфейс пользователя и цикл сообщений



Как совместить интерфейс и долгие вычисления



Обработка сообщений за пределами главного цикла сообщений

Решение

```
#define FRAME_TIME 500
bool stop;

...::on_start_clicked()
{
    stop=false;
    while(вычисление не закончено && !stop)
    {
        ОдинШагВычислений();
        ОтрисоватьНовоеСостояние();

        QApplication::ProcessEvents();

        QThread::msleep(FRAME_TIME);
    }
}

...::on_stop_clicked()
{
    stop=true;
}
```

Аргументы

За

- легко реализовать.

Против

- обработку сообщений надо вызывать часто;
- остаются моменты, в которые приложение не обрабатывает сообщения и не выполняет полезной работы;
- в Qt есть ограничения при использовании `QApplication::processEvents()`

Асинхронное программирование

Решение

```
#define FRAME_TIME 500
int ...::timer_id=0;

...::on_start_clicked()
{
    timer_id=startTimer(FRAME_TIME);
}

...::timerEvent(QTimerEvent *event)
{
    ОдинШагВычислений();
    ОтрисоватьНовоеСостояние();
    if(ВычислениеЗакончено)
        on_stop_clicked();
}

...::on_stop_clicked()
{
    if(timer_id!=0)
    {
        killTimer(timer_id);
        timer_id=0;
    }
}
```

Аргументы

За

- устраняются все возможные задержки обработки сообщений ОС;
- не нужно выносить обработку сообщений из основного цикла.

Против

- может быть сложно реализовать алгоритм в виде отдельных небольших шагов.