



```
messageThreadExecutor.callAsync([] () { std::cout << "Hello World!" << std::endl; }));
```

```
class AsyncCaller {  
public:  
    void callAsync(std::function<void()> && lambda) {  
        auto success = queue.push(std::move(lambda));  
        assert (success);  
    }  
};
```

```
private:  
    fifo<std::function<void()>> queue;  
};
```

```
AsyncCaller messageThreadExecutor;
```

```
class AsyncCaller {
public:
    void callAsync(std::function<void()> && lambda) {
        auto success = queue.push(std::move(lambda));
        assert (success);
    }

    void process() {
        std::function<void()> lambda;
        while (queue.pop (lambda))
            lambda();
    }
private:
    fifo<std::function<void()>> queue;
};

AsyncCaller messageThreadExecutor;
```

```
class AsyncCaller {
public:
    void callAsync(std::function<void()> && lambda) {
        auto success = queue.push(std::move(lambda));
        assert (success);
    }

    void process() {
        std::function<void()> lambda;
        while (queue.pop (lambda))
            lambda();
    }
private:
    fifo<std::function<void()>> queue;
};
```

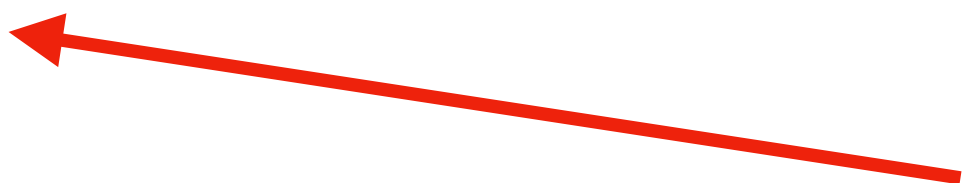
```
AsyncCaller messageThreadExecutor;
```

```
void timerCallback() {
    messageThreadExecutor.process();
}
```

Farbot's Async Callable



**Called on realtime threads**



**Must not wake-up non-  
realtime thread as signalling  
another thread is not lock-free**



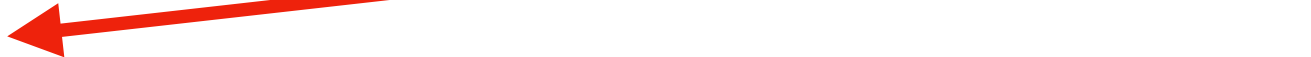
**} Lambdas are executed on a timer  
on the non-realtime thread**



**Pops lambdas from the queue**

**User needs to ensure that**

**$\lambda$  is real-time movable**















# Farbot's AsyncCaller

```
class AsyncCaller {
public:
    void callAsync(std::function<void()> && lambda) {
        auto success = queue.push(std::move(lambda));
        assert (success);
    }

    void process() {
        std::function<void()> lambda;
        while (queue.pop (lambda))
            lambda();
    }

private:
    fifo<std::function<void()>> queue;
};
```

```
AsyncCaller messageThreadExecutor;
```

```
void timerCallback() {
    messageThreadExecutor.process();
}
```

```
messageThreadExecutor.callAsync([] () { std::cout << "Hello World!" << std::endl; });
```

User needs to ensure that  
lambda is real-time movable

Called on realtime threads

Must not wake-up non-  
realtime thread as signalling  
another thread is not lock-free

Pops lambdas from the queue

Lambdas are executed on a timer  
on the non-realtime thread

# FIFO Summary

- Scenario:
  - Data is big: `std::atomic<>::is_always_lock_free == false`
  - Transferring objects between real-time and non-real-time threads
- Trade-off:
  - Static FIFO size
  - Behaviour when FIFO full (block/drop/overwrite)
  - Potential overhead of copying when writing and reading from the FIFO
- Examples:
  - Logging, writing input to disk (recording), reading from disk, dispatching