

# Callbacks and Spinning

- ⇒ While roscpp may use threads behind the scenes to do network management, scheduling etc., it will never expose its threads to your application.
- ⇒ roscpp does, however, allow your callbacks to be called from any number of threads.
- ⇒ The end result is that without a little bit of work from the user your subscription, service and other callbacks will never be called.

## ★ Single-threaded Spinning

- ⇒ The simplest (and most common) version of single-threaded spinning is `ros::spin()`:
  - ↳ `ros::spin()` will not return until the node has been shutdown, either through a call to `ros::shutdown()` or a Ctrl-C.
- ⇒ Another common pattern is to call `ros::spinOnce()` periodically.
  - ↳ `ros::spinOnce()` will call all the callbacks waiting to be called at that point in time.

Note: `spin()` and `spinOnce()` are really meant for single-threaded applications, and are not optimized for being called from multiple threads at once.

## ★ Multi-threaded Spinning

- ⇒ roscpp provides some built-in support for calling callbacks from multiple threads.

- ⇒ There are two built-in options for this:

- `ros::MultiThreadedSpinner`

- `MultiThreadedSpinner` is a blocking spinner, similar to `ros::spin()`.

- You can specify a number of threads in its constructor, but if unspecified (or set to 0), it will use a thread for each CPU core.

```
1 ros::MultiThreadedSpinner spinner(4); // Use 4 threads
2 spinner.spin(); // spin() will not return until the node has been shutdown
```

- `ros::AsyncSpinner`

- A more useful threaded spinner is the `AsyncSpinner`.

- Instead of a blocking `spin()` call, it has `start()` and `stop()` calls, and will automatically stop when it is destroyed.

## ★ CallbackQueue::callAvailable() and callOne()

- ⇒ You can create callback queues this way:

```
#include <ros/callback_queue.h>
...
ros::CallbackQueue my_queue;
```

- ⇒ The `CallbackQueue` class has two ways of invoking the callbacks inside it:
  - `callAvailable()`
  - `callOne()`

⇒ Both `callAvailable()` and `callOne()` can take in an optional timeout, which is the amount of time they will wait for a callback to become available before returning.

↳ If this is zero and there are no callbacks in the queue the method will return immediately.

## ★ Advanced: Using Different Callback Queues

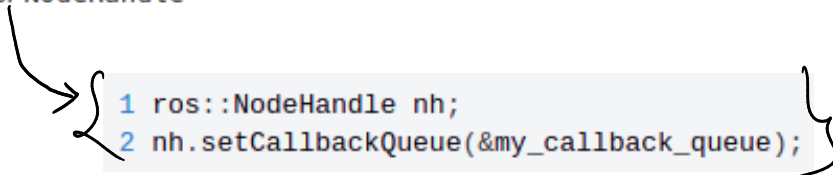
By default, all callbacks get assigned into that global queue, which is then processed by `ros::spin()` or one of the alternatives.

You can get pointer to global queue by calling `ros::getGlobalCallbackQueue()`.

`roscpp` also lets you assign custom callback queues and service them separately.

This can be done in one of two granularities:

1. Per `subscribe()`, `advertise()`, `advertiseService()`, etc.
2. Per `NodeHandle`



```
1 ros::NodeHandle nh;  
2 nh.setCallbackQueue(&my_callback_queue);
```

The various `*Spinner` objects can also take a pointer to a callback queue to use rather than the default one:

```
1 ros::AsyncSpinner spinner(0, &my_callback_queue);  
2 spinner.start();
```

```
1 ros::MultiThreadedSpinner spinner(0);  
2 spinner.spin(&my_callback_queue);
```