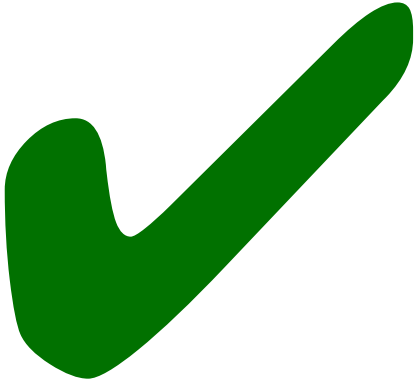
```cpp
struct BiquadCoeffecients  {  float b0, b1, b2, a1, a2; };
std::unique_ptr<BiquadCoeffecients> storage { std::make_unique<BiquadCoeffecients>() };
std::atomic<BiquadCoeffecients*> biquadCoeffs;

void processAudio (float* buffer)
{
    auto* coeffs = biquadCoeffs.exchange (nullptr); // set biquadCoeffs to nullptr while in processing audio

    processBiquad (*coeffs, buffer);
    coeffs->b0 *= 2.0;

    biquadCoeffs = coeffs;
}

void changeBiquadParameters (BiquadCoeffecients newCoeffs)
{
    auto newBiquad = std::make_unique<BiquadCoeffecients> (newCoeffs);

    for (auto* expected = storage.get(); // spin while the realtime thread is processing
         ! biquadCoeffs.compare_exchange_strong (expected, newBiquad.get());
         expected = storage.get());

    storage = std::move (newBiquad);
}
```

The CAS Exchange Loop



Works!

Old storage now deleted

Changes on real-time thread will be lost

48

The CAS Exchange Loop

```
struct BiquadCoeffecients  {  float b0, b1, b2, a1, a2; };
std::unique_ptr<BiquadCoeffecients> storage { std::make_unique<BiquadCoeffecients>() };
std::atomic<BiquadCoeffecients*> biquadCoeffs;

void processAudio (float* buffer)
{
    auto* coeffs = biquadCoeffs.exchange (nullptr); // set biquadCoeffs to nullptr while in processing audio

    processBiquad (*coeffs, buffer);
    coeffs->b0 *= 2.0;

    biquadCoeffs = coeffs;
}

void changeBiquadParameters (BiquadCoeffecients newCoeffs)
{
    auto newBiquad = std::make_unique<BiquadCoeffecients> (newCoeffs);

    for (auto* expected = storage.get(); // spin while the realtime thread is processing
         ! biquadCoeffs.compare_exchange_strong (expected, newBiquad.get());
         expected = storage.get());

    storage = std::move (newBiquad);
}
```

Changes on real-time thread will be lost

Old storage now deleted

✔ **Works!**

farbot's NonRealtimeMutatable

```
struct BiquadCoeffecients  {  float b0, b1, b2, a1, a2; };
NonRealtimeMutatable<BiquadCoeffecients> biquadCoeffs;

void processAudio (float* buffer)
{
    auto& coeffs = biquadCoeffs.realtimeAcquire();

    processBiquad (coeffs, buffer);

    biquadCoeffs.realtimeRelease();
}

void changeBiquadParameters (BiquadCoeffecients newCoeffs)
{
    auto& coeffs = biquadCoeffs.nonRealtimeAcquire();

    coeffs = newCoeffs;

    biquadCoeffs.nonRealtimeRelease();
}
```