





```
struct BiquadCoefficients { float b0, b1, b2, a1, a2; };
std::unique_ptr<BiquadCoefficients> storage { std::make_unique<BiquadCoefficients>() };
std::atomic<BiquadCoefficients*> 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 (BiquadCoefficients newCoeffs)
{
    auto newBiquad = std::make_unique<BiquadCoefficients> (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**









The CAS Exchange Loop

```
struct BiquadCoefficients { float b0, b1, b2, a1, a2; };  
std::unique_ptr<BiquadCoefficients> storage { std::make_unique<BiquadCoefficients>() };  
std::atomic<BiquadCoefficients*> 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;  Changes on real-time thread will be lost  
  
    biquadCoeffs = coeffs;  
}
```

```
void changeBiquadParameters (BiquadCoefficients newCoeffs)  
{  
    auto newBiquad = std::make_unique<BiquadCoefficients> (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);  Old storage now deleted  
}
```



Works!

farbot's NonRealtimeMutable

```
struct BiquadCoefficients { float b0, b1, b2, a1, a2; };  
NonRealtimeMutable<BiquadCoefficients> biquadCoeffs;
```

```
void processAudio (float* buffer)  
{  
    auto& coeffs = biquadCoeffs.realtimeAcquire();  
    processBiquad (coeffs, buffer);  
    biquadCoeffs.realtimeRelease();  
}
```

```
void changeBiquadParameters (BiquadCoefficients newCoeffs)  
{  
    auto& coeffs = biquadCoeffs.nonRealtimeAcquire();  
    coeffs = newCoeffs;  
    biquadCoeffs.nonRealtimeRelease();  
}
```