Mutating on realtime and non-realtime

```
void mixAllSources (float* output, char* realtimeEventMessages, int n) {
    processRealtimeEvents(realtimeEventMessages); // may add and remove sources
    realtimeThreadCaller.process(); // process all the lambdas

    RealtimeMutatable<SourceList>::ScopedAccess<true> sourceList (sharedSourceList);
    for (int i = 0; i < sourceList->numSources; ++i)
        mixSource (output, sourceList->buffers[i]);
}


void printSources() {
    RealtimeMutatable<SourceList>::ScopedAccess<false> sourceList (sharedSourceList);
    for (int i = 0; i < sourceList->numSources; ++i)
        std::cout << (void*)sourceList->buffers[i] << std::endl;
}
```

76

Mutating on realtime and non-realtime

```
void mixAllSources (float* output, char* realtimeEventMessages, int n) {
    processRealtimeEvents(realtimeEventMessages); // may add and remove sources
    realtimeThreadCaller.process(); // process all the lambdas

    RealtimeMutatable<SourceList>::ScopedAccess<true> sourceList (sharedSourceList);
    for (int i = 0; i < sourceList->numSources; ++i)
        mixSource (output, sourceList->buffers[i]);
}
```

```
void printSources() {
    RealtimeMutatable<SourceList>::ScopedAccess<false> sourceList (sharedSourceList);
    for (int i = 0; i < sourceList->numSources; ++i)
        std::cout << (void*)sourceList->buffers[i] << std::endl;
}
```

Real-time & Non-real-time Summary

- Scenario:
  - Data is big: `std::atomic<>::is_always_lock_free == false`
  - Sharing data between real-time and non-real-time threads
  - Both threads can mutate data
- Trade-off:
  - One thread needs to own the data
  - Same trade-offs as FIFOs & `(Non)RealTimeMutatableObjects`
  - Complexity
- Examples:
  - Managing lists and dynamic streams where losing packets is not acceptable