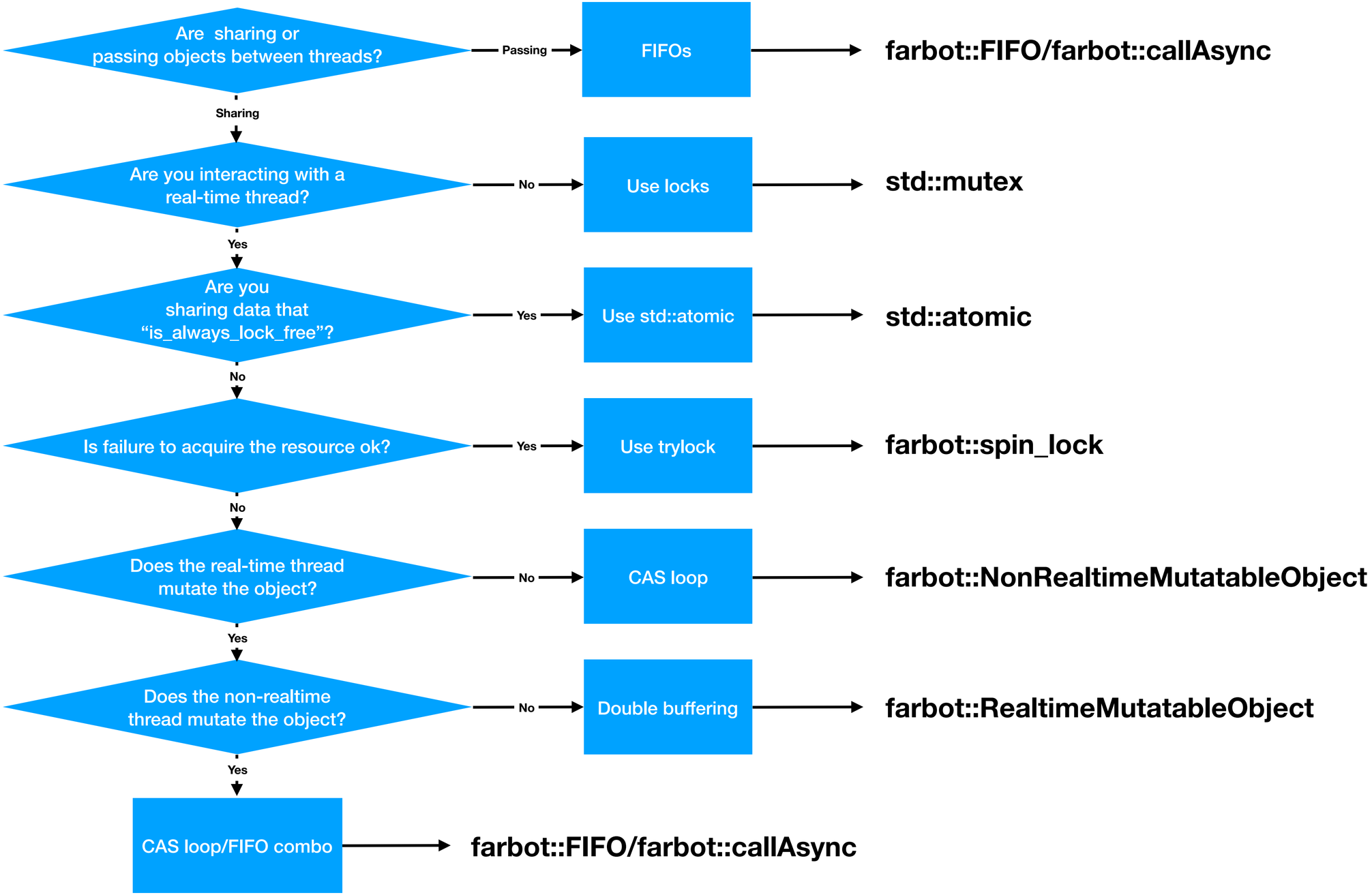
Real-time & Non-real-time Summary

- 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


Are sharing or passing objects between threads?
Passing
FIFOs
farbot::FIFO/farbot::callAsync
Sharing
Are you interacting with a real-time thread?
No
Use locks
std::mutex
Yes
Are you sharing data that "is_always_lock_free"?
Yes
Use std::atomic
std::atomic
No
Is failure to acquire the resource ok?
Yes
Use trylock
farbot::spin_lock
No
Does the real-time thread mutate the object?
No
CAS loop
farbot::NonRealtimeMutatableObject
Yes
Does the non-realtime thread mutate the object?
No
Double buffering
farbot::RealtimeMutatableObject
Yes
CAS loop/FIFO combo
farbot::FIFO/farbot::callAsync