70

```
template<class T+>
class
[[unsafe::send(T~is_send), unsafe::sync(T~is_send)]]
mutex
{
  using mutex_type = unsafe_cell<std::mutex>;

  unsafe_cell<T> data_;
  box<mutex_type> mtx_;



public:
  class lock_guard/(a)
  {
    friend class mutex;

    mutex const^/a m_;

    lock_guard(mutex const^/a m) noexcept safe
      : m_(m)
    {
    }
```

```
template<class T+>
class vector
{
public:
  using value_type = T;
  using size_type = std::size_t;

  //…

  [[unsafe::drop_only(T)]]
  ~vector() safe {
    // TODO: std::destroy_n() doesn't seem to
    // like `int^` as a value_type
    // eventually we should fix this

    unsafe {
      auto const* end = self.data() + self.size();
      auto* pos = self^.data();

      while (pos < end) {
        auto t = __rel_read(pos);
        drp t;
        ++pos;
      }

      ::operator delete(p_);
    }
  }
```

```
template<class T+>
class vector
{
public:
  using value_type = T;
  using size_type = std::size_t;

  //…

  [[unsafe::drop_only(T)]]
  ~vector() safe {
    // TODO: std::destroy_n() doesn't seem to
    // like `int^` as a value_type
    // eventually we should fix this

    unsafe {
      auto const* end = self.data() + self.size();
      auto* pos = self^.data();

      while (pos < end) {
        auto t = __rel_read(pos);
        drp t;
        ++pos;
      }

      ::operator delete(p_);
    }
  }
```

```
template<class T+>
class
[[unsafe::send(T~is_send), unsafe::sync(T~is_send)]]
mutex
{
  using mutex_type = unsafe_cell<std::mutex>;

  unsafe_cell<T> data_;
  box<mutex_type> mtx_;


public:
  class lock_guard/(a)
  {
    friend class mutex;

    mutex const^/a m_;

    lock_guard(mutex const^/a m) noexcept safe
      : m_(m)
    {
    }
```

Rust

```rust
pub(super) struct PthreadMutexAttr<'a>(pub &'a mut MaybeUninit<libc::pthread_mutexattr_t>);

impl Drop for PthreadMutexAttr<'_> {
    fn drop(&mut self) {
        unsafe {
            let result = libc::pthread_mutexattr_destroy(self.0.as_mut_ptr());
            debug_assert_eq!(result, 0);
        }
    }
}
```