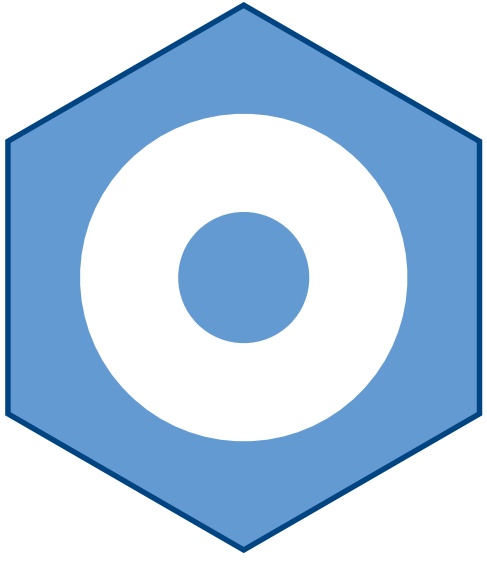

Circle

69

```
auto numbers = std2::vector<int> { 1, 2, 3 };
auto iterator = numbers.iter();
numbers = std2::vector<int> { 4, 5, 6 };

for (auto number : iterator)
    std2::println (number);
```

```
safety: during safety checking of int main() safe
  borrow checking: example.cpp:10:24
      for (auto number : iterator)
                         ^

  use of iterator depends on expired loan
  drop of numbers between its shared borrow and its use
  invalidating operation at example.cpp:8:13
      numbers = std2::vector<int> { 4, 5, 6 };
              ^

  loan created at example.cpp:7:21
      auto iterator = numbers.iter();
```

https://godbolt.org/z/h71xo3b54

Circle

```
auto numbers = std2::vector<int> { 1, 2, 3 };
auto iterator = numbers.iter();
numbers = std2::vector<int> { 4, 5, 6 };

for (auto number : iterator)
    std2::println (number);
```

```
safety: during safety checking of int main() safe
  borrow checking: example.cpp:10:24
      for (auto number : iterator)
                         ^
  use of iterator depends on expired loan
  drop of numbers between its shared borrow and its use
  invalidating operation at example.cpp:8:13
      numbers = std2::vector<int> { 4, 5, 6 };
                ^
  loan created at example.cpp:7:21
      auto iterator = numbers.iter();
```

```
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)
    {
    }
```