

# Examples

Pieter P

## Acquire-release

```
1 unsigned value = 0;
2 std::atomic_bool value_ready{false};
```

### Release

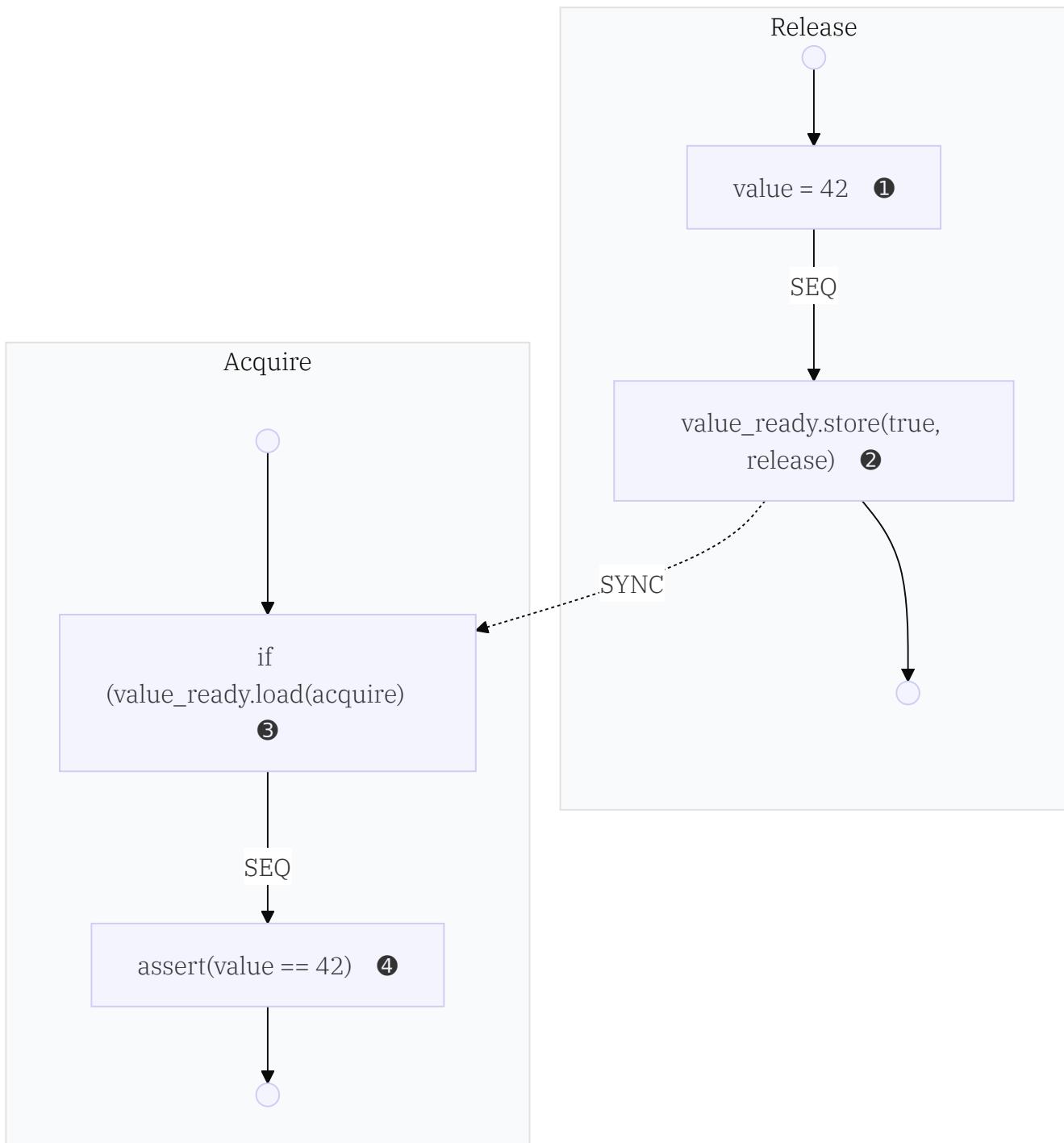
```
1 value = 42; //  
2 value_ready.store(true, std::memory_order_release); //
```

①  
② ▲▲▲

### Acquire

```
1 if (value_ready.load(std::memory_order_acquire)) //  
2     assert(value == 42); //
```

③ ▼▼▼▼  
④



If ③ reads the value written by ②, then the load-acquire ③ synchronizes with the store-release ② on the same atomic variable. Therefore, ② simply happens before ③. Combined with the fact that ① is sequenced before ② and ③ is sequenced before ④, we conclude that ① strongly happens before ④.

If ③ does not read the value written by ②, then there is no synchronization, but this is not an issue because in that case, the branch is not taken and ④ is never executed.

## Interrupt handle once

```

1 std::atomic_bool handled{false};
2 unsigned value = 0;
3 std::atomic<const unsigned *> value_ptr{nullptr};
4 std::atomic_uint total{0};

```

## Main

```
1 // Store the value to be handled
2 value = 1; //
3 value_ptr.store(&value, std::memory_order_seq_cst); // ② ▲▲▲
4 // Check if the interrupt was handled before we ran
5 auto h = handled.load(std::memory_order_seq_cst); // ③ ▼▼▼
6 // If the interrupt ran earlier, it might not have seen our value
7 if (h) {
8     // See if our value is still there
9     auto v = value_ptr.exchange(nullptr, std::memory_order_relaxed); // ④
10    // And handle it ourselves
11    if (v)
12        total.fetch_add(*v, std::memory_order_relaxed); // ⑤
13 }
```

## Interrupt

```
1 // Notify the main thread that the interrupt ran
2 handled.store(true, std::memory_order_seq_cst); // ⑥ ▲▲▲
3 // Read the value from the main thread
4 auto v = value_ptr.exchange(nullptr, std::memory_order_seq_cst); // ⑦ ▼▼▼
5 // Handle it
6 if (v)
7     total.fetch_add(*v, std::memory_order_relaxed); // ⑧
```

