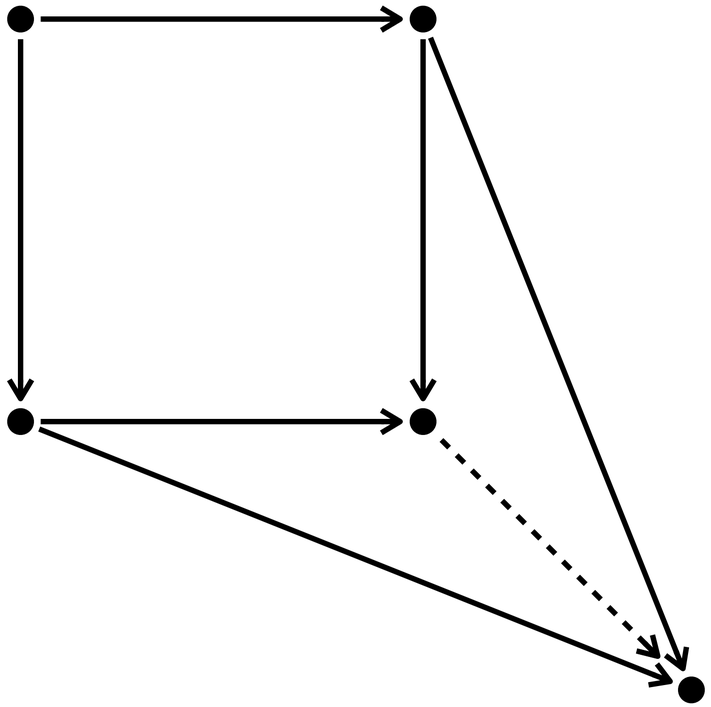
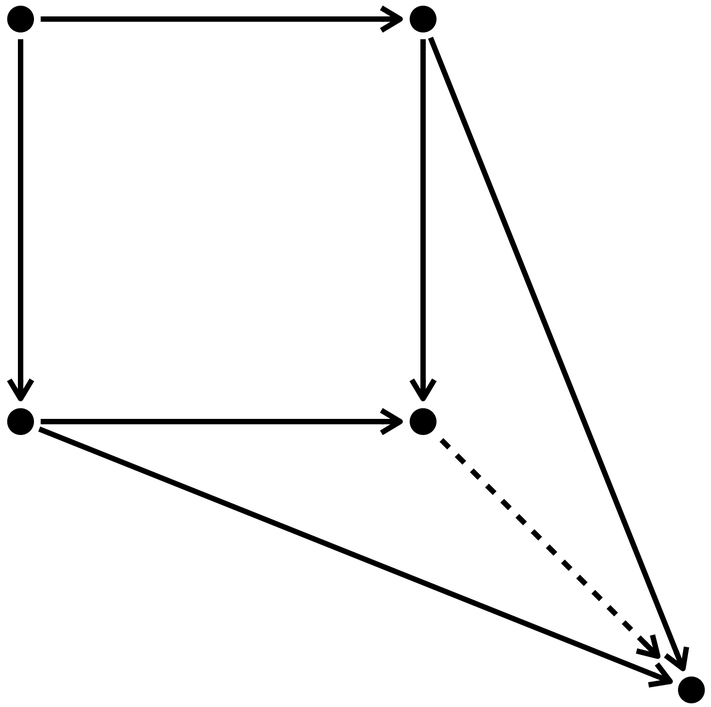
The Vector Challenge

Lisa Lippincott

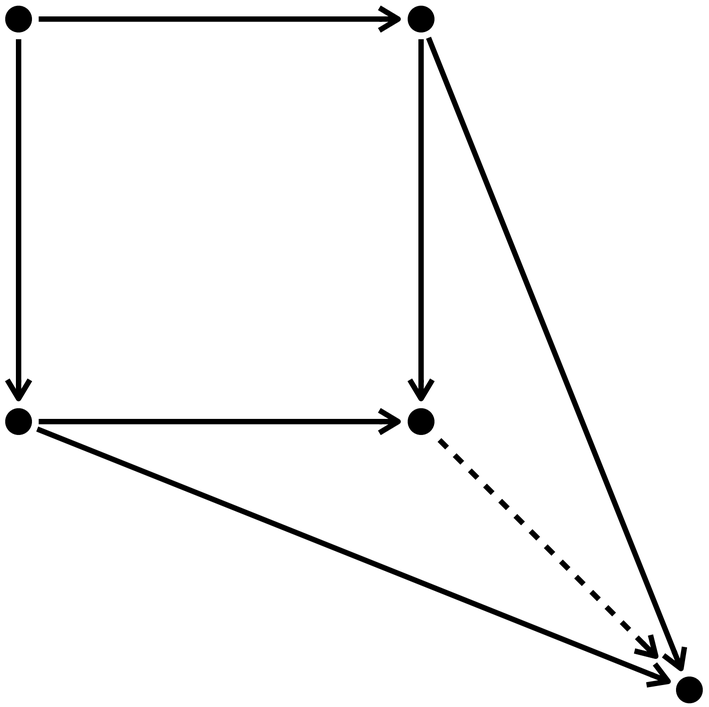
C++now, May 2024



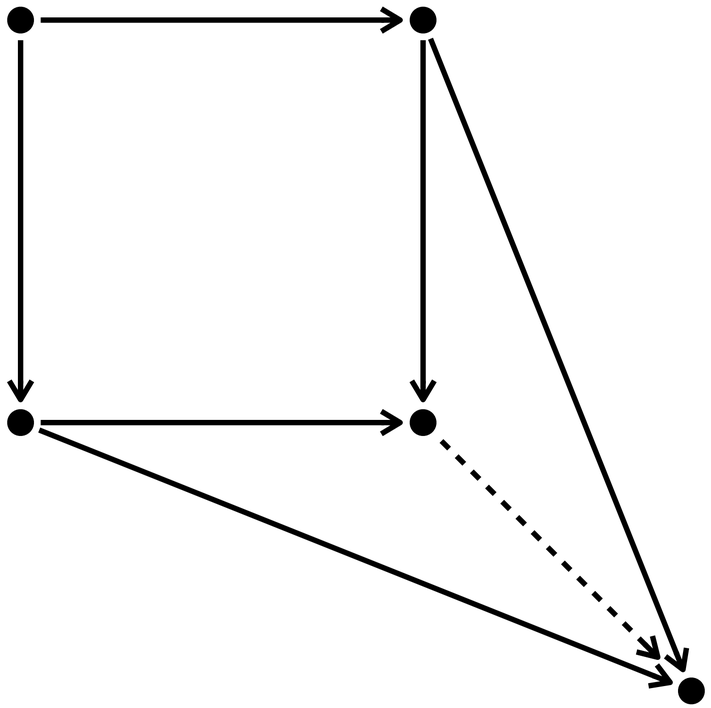


Ludwig Wittgenstein
(1889-1951)

Tractatus Logico-Philosophicus




CDRA
TVLA
ATA
LLA
CTM
Margaret Hamilton



Learn to explain your reasoning

Write down your explanation

Get someone to check it

Alexander Stepanov
(1950-)

```
template < class... Ts >
```

```
boost::bind( std::less, _1, x )
std::bind( std::less, _1, x )
[ x ]( auto y ){ return y < x; }
```

We know why we expect the programs we write to work.

We should explain our reasoning in a formal language, and have computers check that our reasoning is correct.

For C++ programs, that formal language should be C++.

```
pop is
    require nb_elements > 0

    do      nb_elements := nb_elements - 1

    ensure  nb_elements < max_size;
            nb_elements = old nb_elements - 1
    end;
```

Bertrand Meyer,
Object-oriented Software Construction
1988

The ability to formally express assertions of a general nature (powerful enough to describe realistic abstract data types) requires an assertion language in which one can directly manipulate sets, sequences, functions, relations and first-order predicates with quantifiers (“for all” and “there exists”).

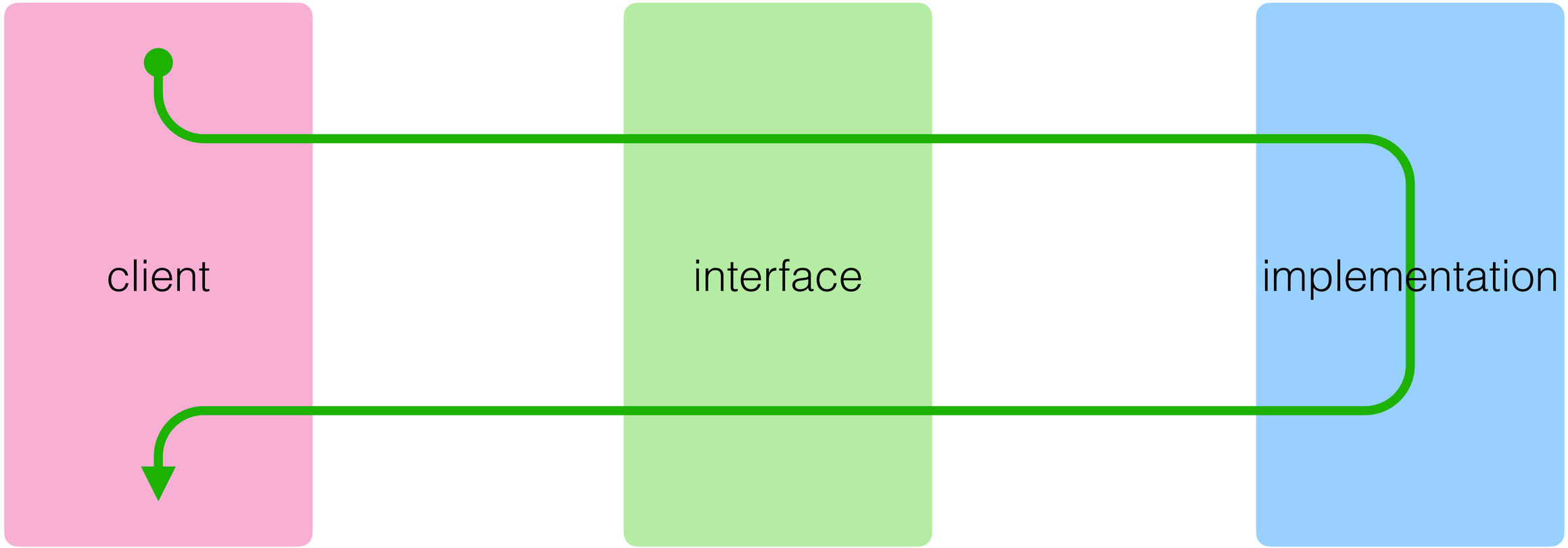
Bertrand Meyer,
Object-oriented Software Construction
1988

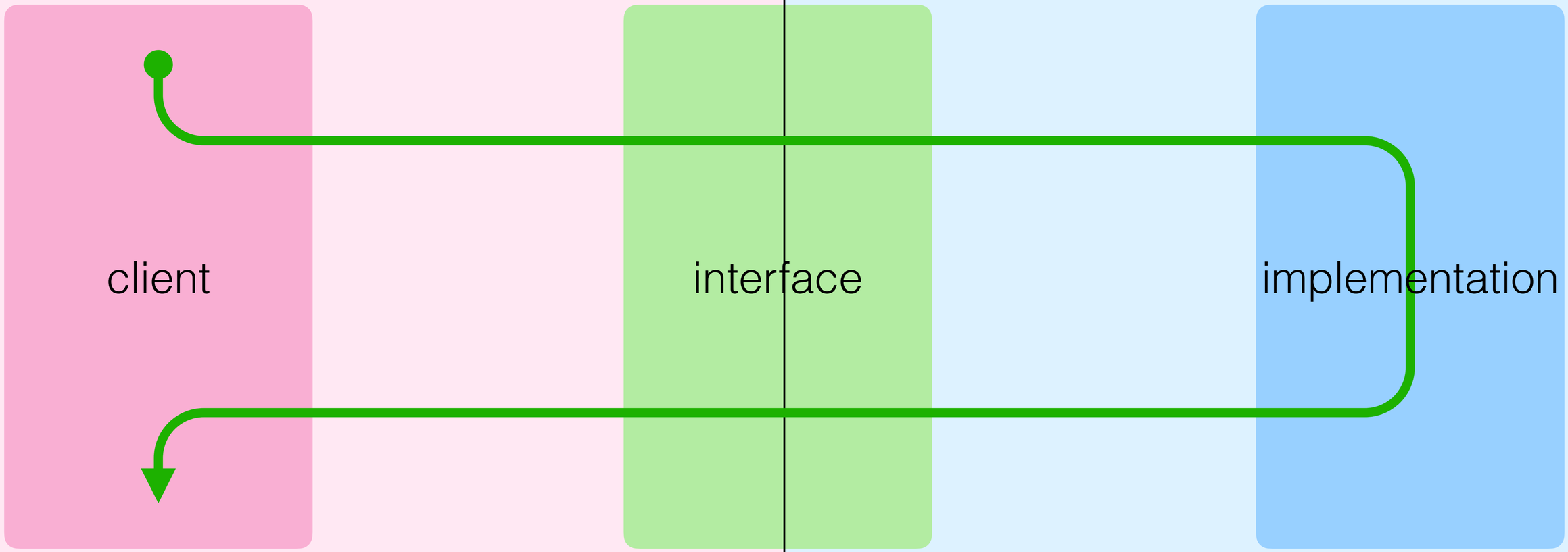
```
pop is
    require nb_elements > 0

    do      nb_elements := nb_elements - 1

    ensure  nb_elements < max_size;
            nb_elements = old nb_elements - 1
    end;
```

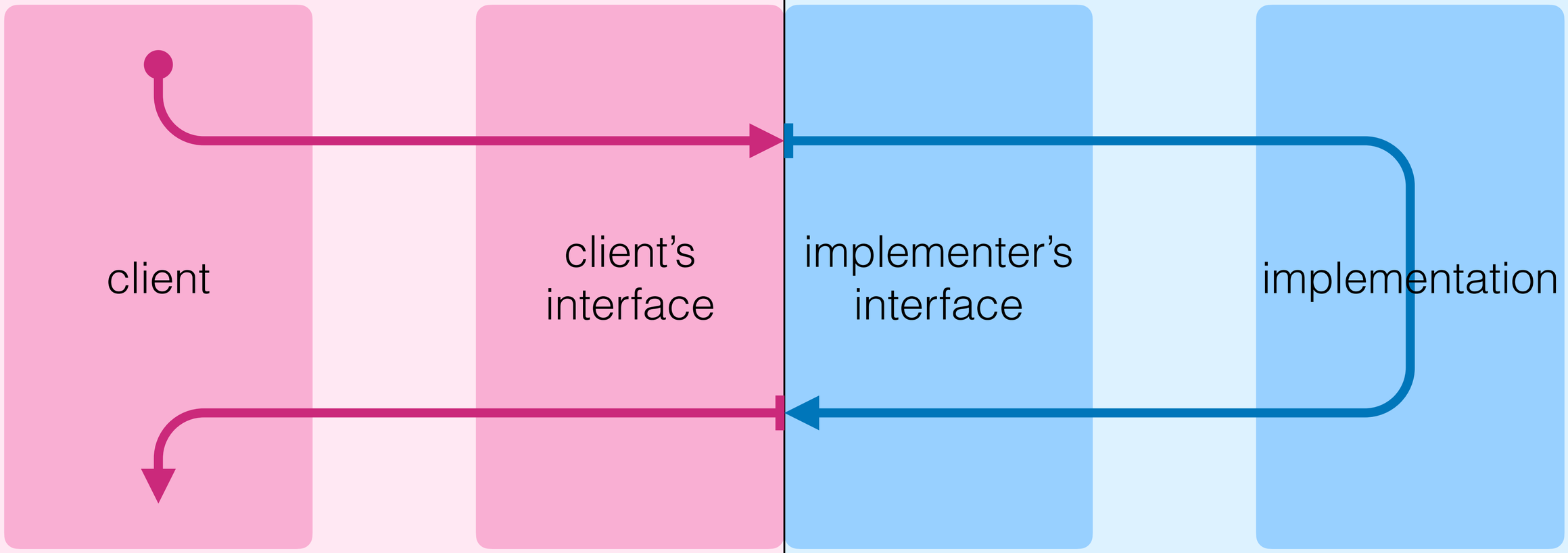
Bertrand Meyer,
Object-oriented Software Construction
1988

```
void pop()
interface
 {
   claim nb_elements > 0;

   const auto old_nb = nb_elements;

   implementation;

   claim nb_elements  <  max_size;
   claim nb_elements == old_nb - 1;
 }
```

```
void pop()
implementation
 {
   --nb_elements;
 }
```


client
interface
implementation


client
interface
implementation


client
client's interface
implementer's interface
implementation

Alexander Stepanov
(1950-)

“It would be really nice to have that for vector.”

vector type client

vector type client's interface

vector type implementer's interface

vector type implementation

vector type client

vector type client's interface

vector type implementer's interface

vector type implementation

vector object client

vector object client's interface

vector object implementer's interface

vector object implementation

```
template < class T, class A >
class vector
interface
 {
   // . . .
 };
```

```
template < class T, class A >
class vector
implementation
 {
   // . . .
 };
```

```
template < class T >
class vector
interface
 {
   // . . .
 };
```

```
template < class T >
class vector
implementation
 {
   // . . .
 };
```

```
template < class T >
class vector
interface
 {
   public:
      using value_type             = T;
      using reference              = T&;
      using pointer                = T*;
      using const_reference        = const T&;
      using const_pointer          = const T*;









    // . . .
 };
```

```
template < class T >
class vector
interface
 {
   public:
      using value_type             = T;
      using reference              = T&;
      using pointer                = T*;
      using const_reference        = const T&;
      using const_pointer          = const T*;
      using size_type              = implementation::size_type;
      using difference_type        = implementation::difference_type;
      using iterator               = implementation::iterator;
      using const_iterator         = implementation::const_iterator;



    // ...
 };
```

```
template < class T >
class vector
interface
 {
  public:
     using value_type             = T;
     using reference              = T&;
     using pointer                = T*;
     using const_reference        = const T&;
     using const_pointer          = const T*;
     using size_type              = implementation::size_type;
     using difference_type        = implementation::difference_type;
     using iterator               = implementation::iterator;
     using const_iterator         = implementation::const_iterator;

     static_assert( !is_signed_v< size_type > );
     static_assert(  is_signed_v< difference_type > );
   // ...
 };
```

```
template < class T >
class vector
interface
 {




  public:
     size_type size() const
     interface
      {
        implementation;


      }

   // . . .
 };
```

```
template < class T >
class vector
interface
 {

   private:
      size_type expected_size;



   public:
      size_type size() const
      interface
       {
         implementation;
         claim result == expected_size;
       }

   // . . .
 };
```

```
template < class T >
class vector
interface
 {

   private:
      size_type expected_size;



   public:
      size_type size() const
      interface
       {
         implementation;
         claim result == expected_size;
       }

   // . . .
 };
```

```
bool empty() const
interface
 {
   implementation;
   claim result == ( expected_size == 0 );
 }
```

object
interface
data

postconditions

```
bool empty() const
interface
 {
  implementation;
  claim result == ( expected_size == 0 );
 }
```

preconditions

inspector interface data

object interface data

postconditions

```
const value_type&
operator[]( size_type index ) const
interface
 {
  const auto i = index;
  claim i < expected_size;

  implementation;

  const auto r = expected_data + i;
  substitutable &result, r;
 }
```

preconditions

inspector interface data

object interface data

postconditions

```
value_type&
operator[]( size_type index )
interface
 {
  const auto i = index;
  claim i < expected_size;

  implementation;

  const auto r = expected_data + i;
  substitutable &result, r;
 }
```

preconditions

inspector interface data

object interface data

postconditions

```
value_type&
operator[]( size_type index )
interface
 {
  const auto i = index;
  claim i < expected_size;

  implementation const;

  const auto r = expected_data + i;
  substitutable &result, r;
 }
```

preconditions

inspector interface data

object interface data

postconditions

```
value_type&
at( size_type index )
interface
 {
  const auto i = index;
  claim i < expected_size;

  implementation const;

  const auto r = expected_data + i;
  substitutable &result, r;
 }
```

preconditions

inspector interface data

object interface data

postconditions

```
value_type&
at( size_type index )
interface
 {
  const auto i = index;

  implementation const;

  claim i < expected_size;

  const auto r = expected_data + i;
  substitutable &result, r;
 }
```

preconditions

inspector interface data

object interface data

postconditions

```
value_type&
at( size_type index )
interface
 {
  const auto i = index;

  if ( i < expected_size )
    implementation const;
  else
    implementation noreturn;

  const auto r = expected_data + i;
  substitutable &result, r;
 }
```

construct object interface

preconditions

constructor
interface
data

construct object interface

postconditions

```
vector()
interface
 {
  // uninitialized memory consumed…

  implementation;

  claim expected_size == 0;
 }
```

```
size_type       expected_size       = implementation.size();
size_type       expected_max_size = implementation.max_size();
size_type       expected_capacity   = implementation.capacity();
const_pointer expected_data       = implementation.data();
```

preconditions

constructor
interface
data

construct object interface

postconditions

```
vector()
interface
 {
  // uninitialized memory consumed…

  implementation;

  claim expected_size == 0;
 }
```

preconditions

constructor
interface
data

construct object interface

postconditions

preconditions

constructor
interface
data

construct object interface

postconditions

destroy object interface

preconditions

inspector interface data

object interface data

postconditions

construct object interface

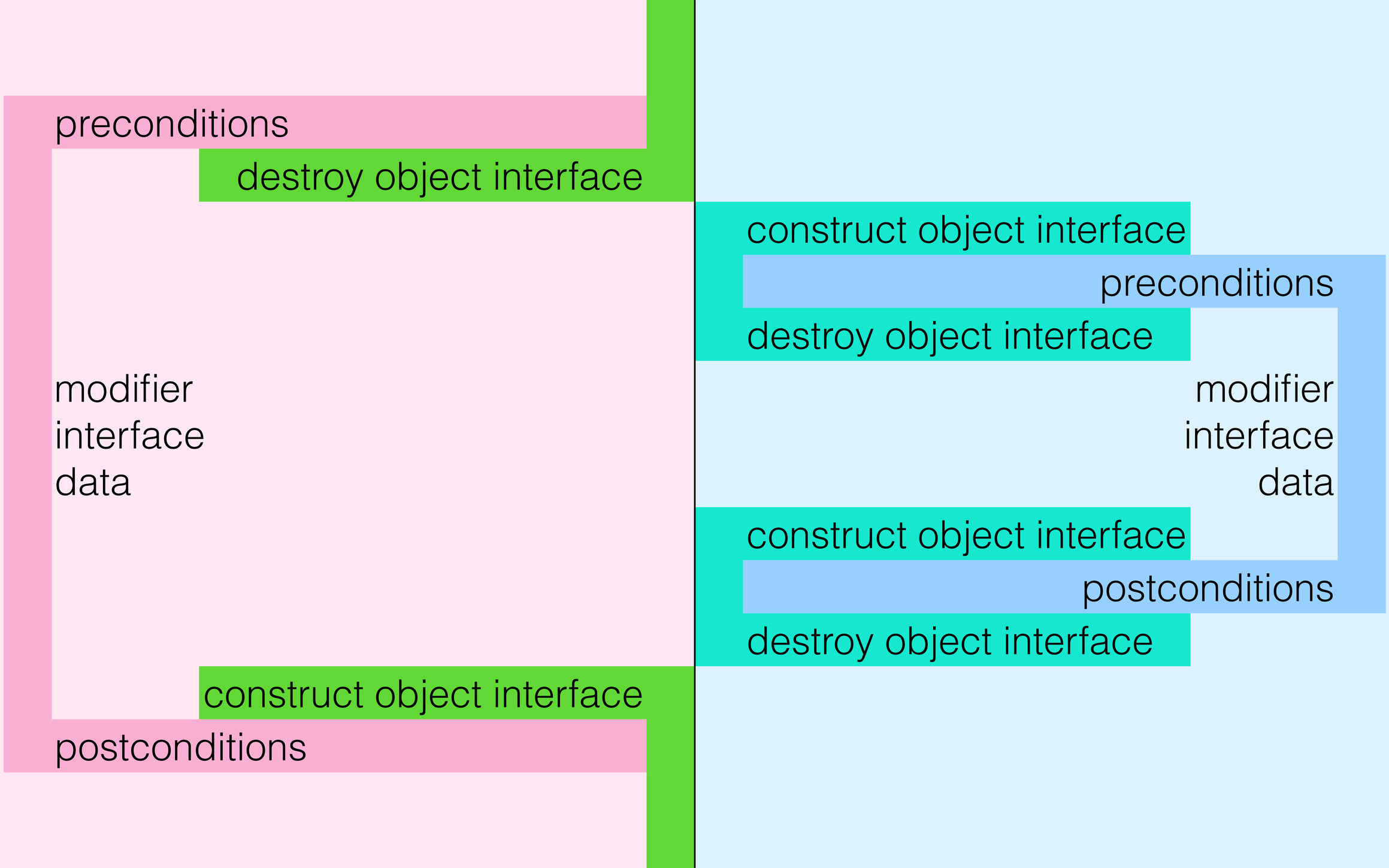
preconditions

object interface data

inspector interface data

postconditions

destroy object interface


preconditions
destroy object interface
modifier
interface
data
construct object interface
postconditions
construct object interface
preconditions
destroy object interface
modifier
interface
data
construct object interface
postconditions
destroy object interface

preconditions

destroy object interface

destructor
interface
data

postconditions

construct object interface

preconditions

destroy object interface

destructor
interface
data

postconditions

preconditions

destroy object interface

destructor
interface
data

postconditions

```
~vector()
interface
 {
  implementation;

  // uninitialized memory left behind…
 }
```

preconditions

destroy object interface

destructor
interface
data

postconditions

```
~vector()
interface
 {
  // elements are consumed…

  implementation;

  // uninitialized memory left behind…
 }
```

```
int counted_object::count = 0;
```

```
counted_object::counted_object()
interface
 {
  const auto one_more = count + 1;

  implementation;

  claim count == one_more;
 }
```

```
counted_object::~counted_object()
interface
 {
  const auto old_count = count;

  implementation;

  claim count == old_count - 1;
 }
```

preconditions

destroy object interface

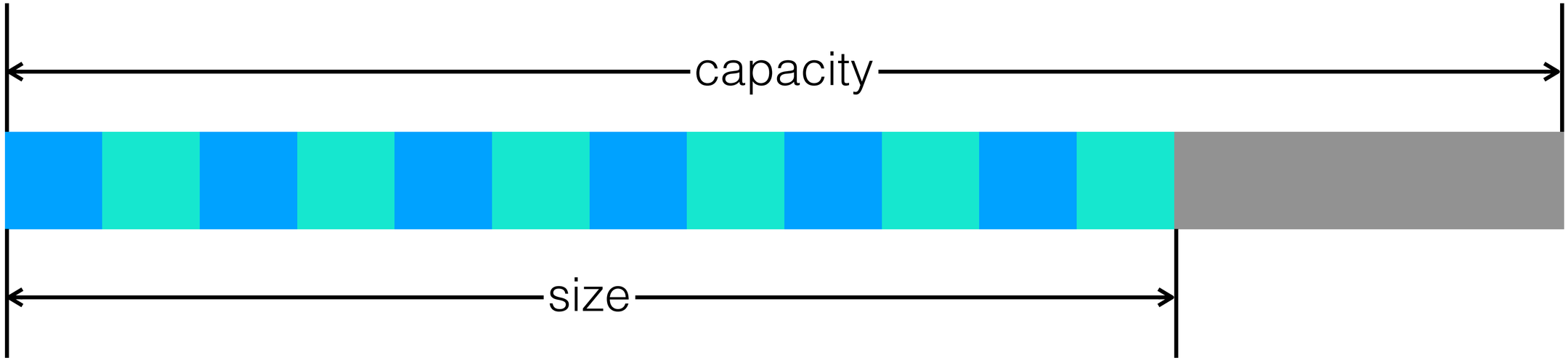
destructor
interface
data

postconditions

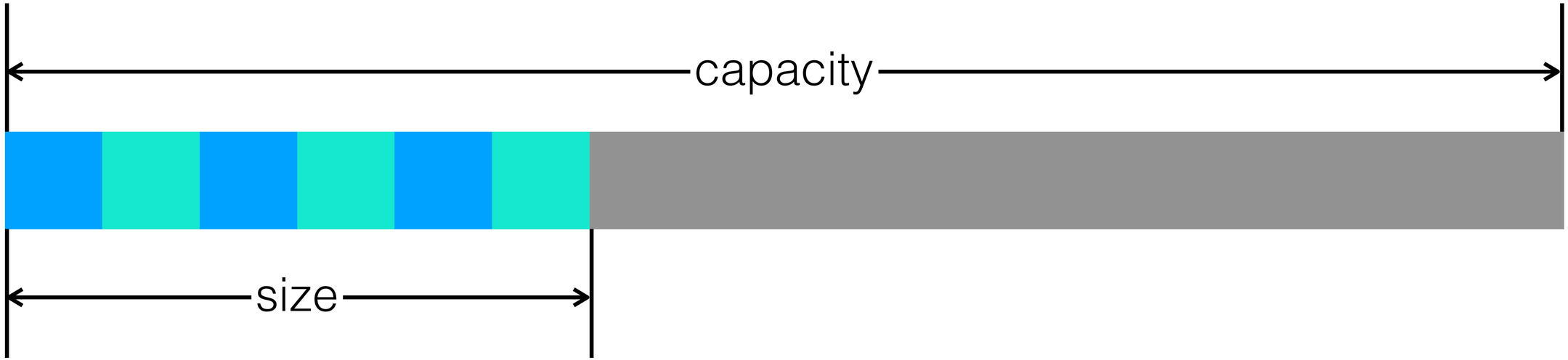
```
inline
~vector()
 {
  destroy( rbegin(), rend() );
  destroy_internal_parts();
 }


void destroy_internal_parts()
interface
 {
  implementation;

  // uninitialized memory left behind…
 }
```

```
const auto a = /* whatever */

const auto b1 = a + 10;
// …
const auto b2 = a + 10;
```

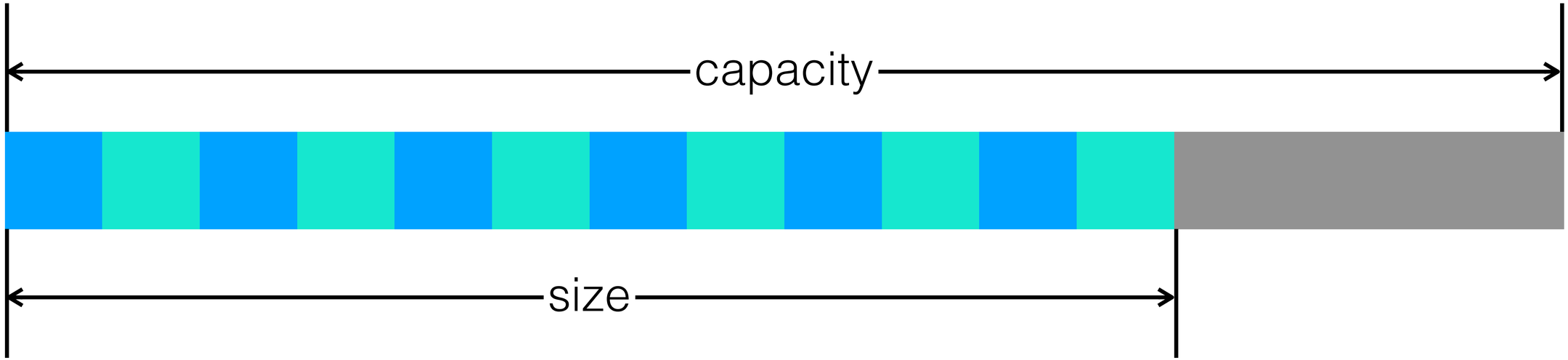
```
const auto a = v.data();

const auto b1 = a + 10;
// …
const auto b2 = a + 10;
```



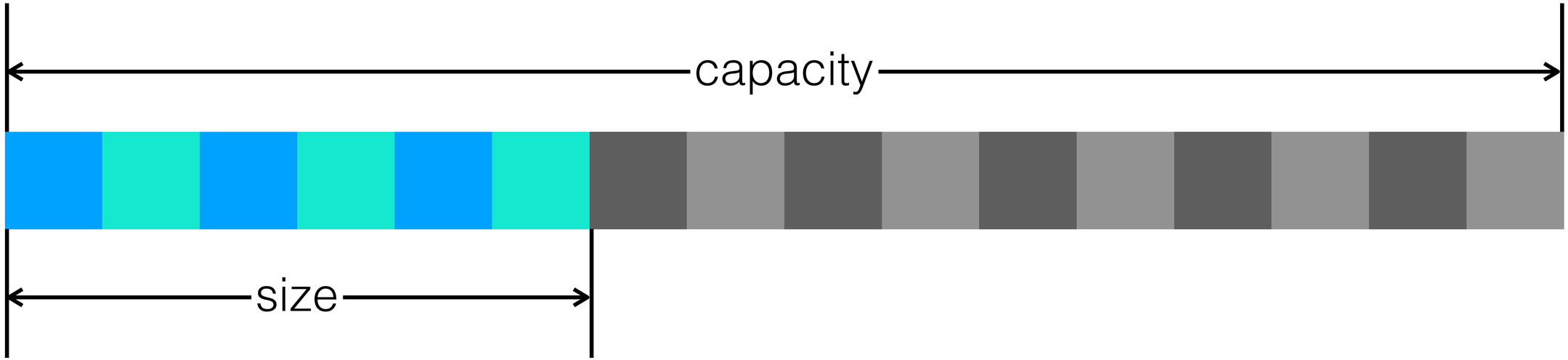
```
const auto a = v.data();

const auto b1 = a + 10;
// ...
const auto b2 = a + 10;
```



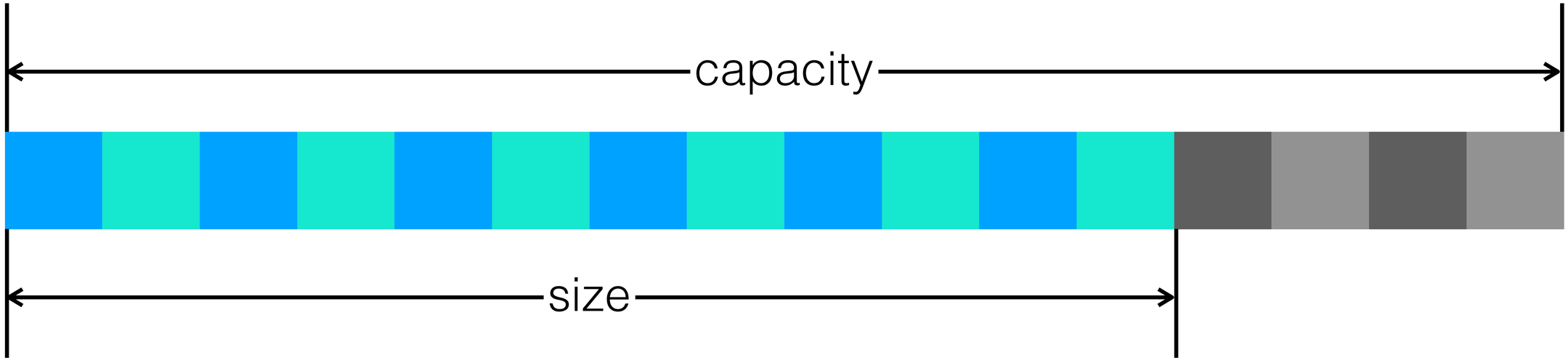
```
const auto a = v.data();

const auto b1 = a + 10;
// …
const auto b2 = a + 10;
```

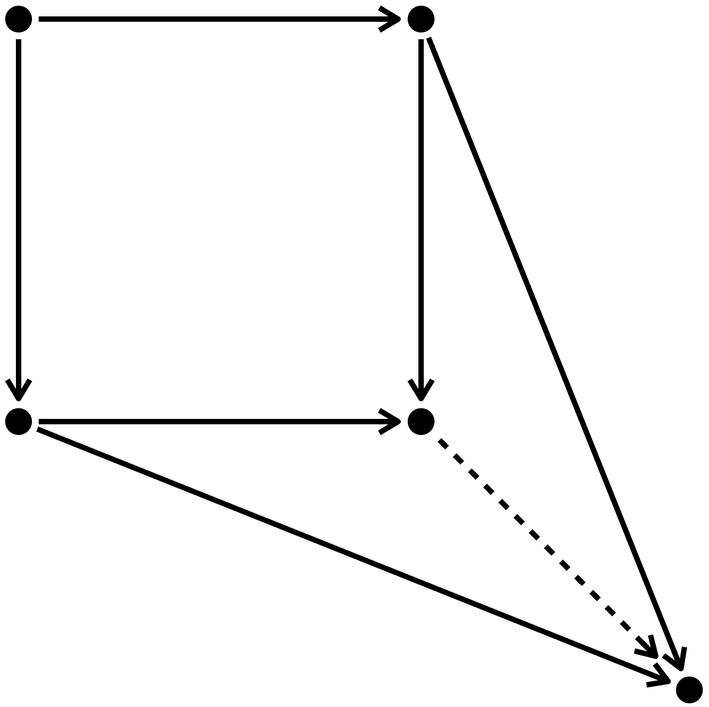


```
const auto a = v.data();

const auto b1 = a + 10;
// …
const auto b2 = a + 10;
```

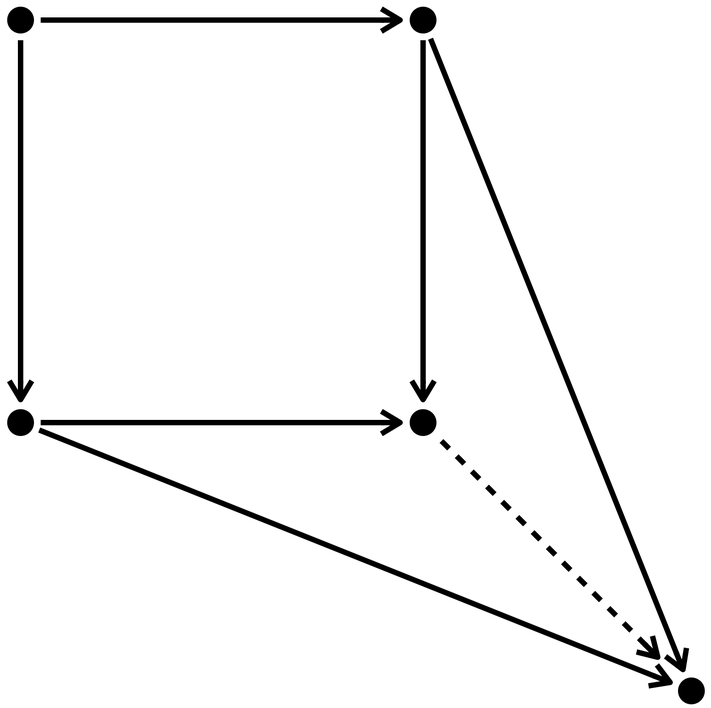


```
const auto a = v.data();

const auto b1 = a + 10;
// …
const auto b2 = a + 10;
```





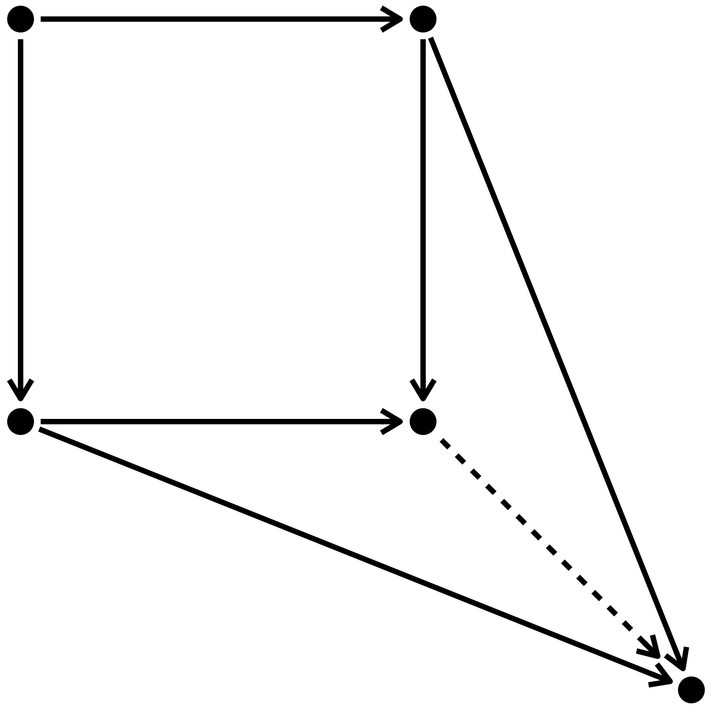

CDRA
TVLA
ATA
LLA
CTM
Margaret Hamilton



Learn to explain your reasoning

Write down your explanation

Get someone to check it



Learn to explain your reasoning

Write down your explanation
in your programming language

Get a computer to check it

Thank you for listening.

Questions?