C++ moderno

¿Qué nos hemos dejado en el tintero?

Atanasio José Rubio Gil

Universidad de Granada

24 de noviembre de 2021

Esta obra está bajo una licencia Creative Commons "Reconocimiento-NoCommercial-CompartirIgual 4.0 Internacional".

1: RAII (C++*)

Atrhur O'Dwyer

Back to Basics: RAII and the Rule of Zero

Resource **A**dquisition **I**s **I**nitialisation

- Encapsular cada recurso en una clase donde:
  - El constructor adquiere el recurso y establece los invariantes de la clase o lanza una excepción si no es posible.
  - El destructor libera el recurso y nunca lanza excepciones.
- Usar siembre el recurso mediante una instancia de una clase RAII que:
  - Tenga una duración de almacenamiento automática o una vida temporal en sí misma.
  - O tenga una vida ligada a la vida de un objeto automático o temporal.

RAII

Trabajando con un `std::mutex m`.

Peligros de usar una clase no RAII

```
void bad()
{
    m.lock(); // Adquire the mutex
    f(); // Exceptions in f()? Mutex isn't released!
    if(!everything_ok()) return; // Return without releasing the mutex
    m.unlock(); // Release the mutex
}
```

Usando RAII dejamos de preocuparnos por las pérdidas de memoria

```
void good()
{
    std::lock_guard<std::mutex> lk(m); // Mutex adquisition is initialisation
    f(); // Exceptions in f()? Release the mutex when destroying lk
    if(!everything_ok()) return; // Release the mutex when returning
}
```

cppreference

RAII

Trabajando con un `std::mutex m`.

Peligros de usar una clase no RAII

```
void bad()
{
    m.lock(); // Adquire the mutex
    f(); // Exceptions in f()? Mutex isn't released!
    if(!everything_ok()) return; // Return without releasing the mutex
    m.unlock(); // Release the mutex
}
```

Usando RAII dejamos de preocuparnos por las pérdidas de memoria

```
void good()
{
    std::lock_guard<std::mutex> lk(m); // Mutex adquisition is initialisation
    f(); // Exceptions in f()? Release the mutex when destroying lk
    if(!everything_ok()) return; // Release the mutex when returning
}
```

cppreference

2: Copia profunda segura (C++*)

Arthur O'Dwyer

Back to Basics: RAII and the Rule of Zero

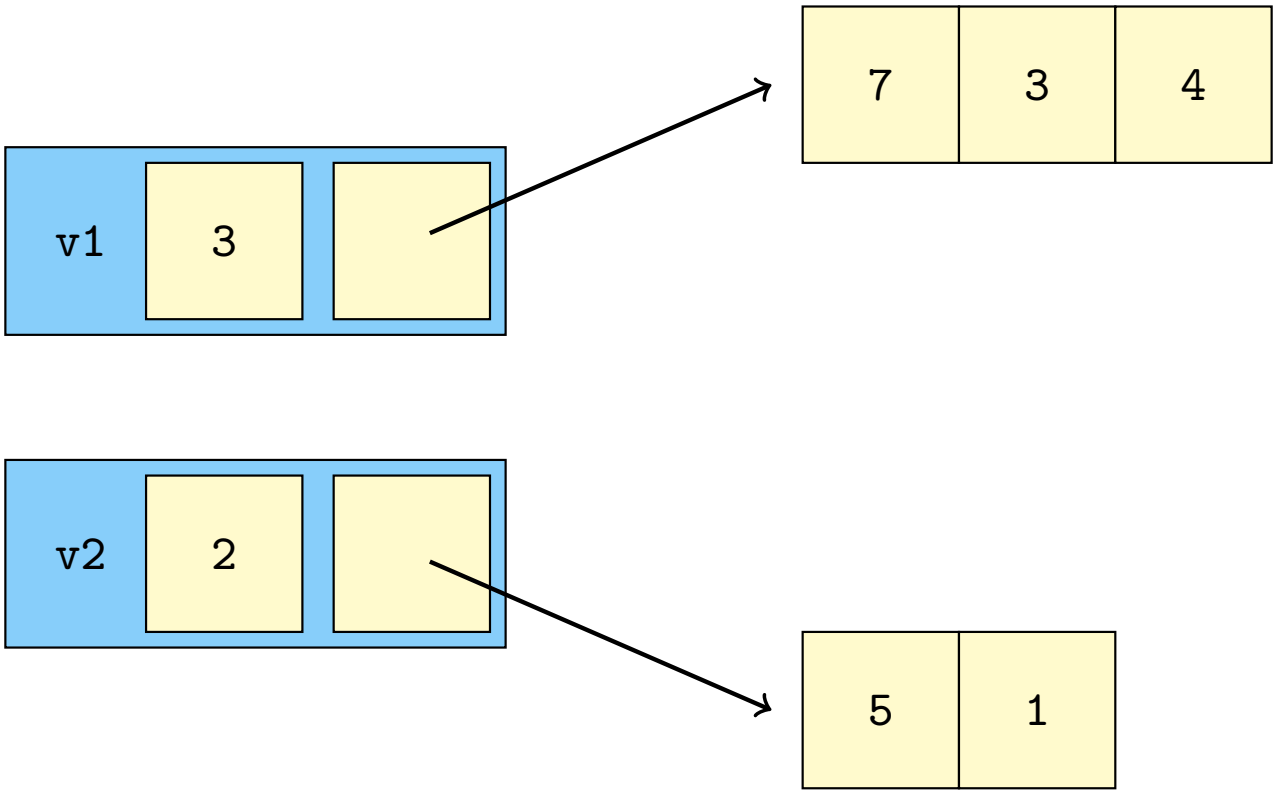
Copia profunda segura

Asignación de un vector por referencia constante

```
class vector
{
private:
   int * array_;
   size_t size_;

public:
   vector () : array_ (nullptr), size_ (0) { }
   vector (const vector & rhs) { *this = rhs; }
   ~vector () { delete [] array_; }

   vector & operator = (const vector & rhs)
   {
      delete [] array_;
      size_  = rhs.size_;
      array_ = new int[size_];
      std::copy(rhs.array_, rhs.array_ + size_, array_);
      return *this;
   }
};
```

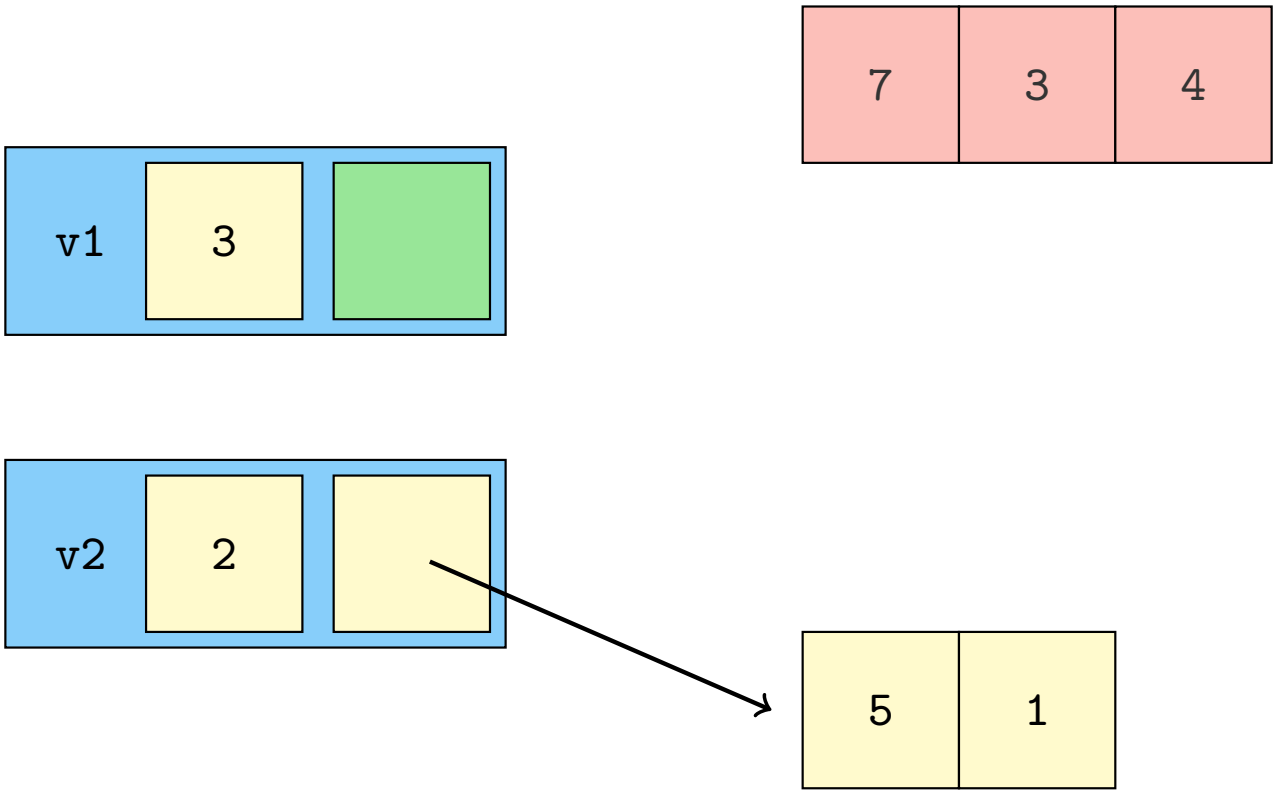
Copia profunda segura

```
v1 = v2;
```



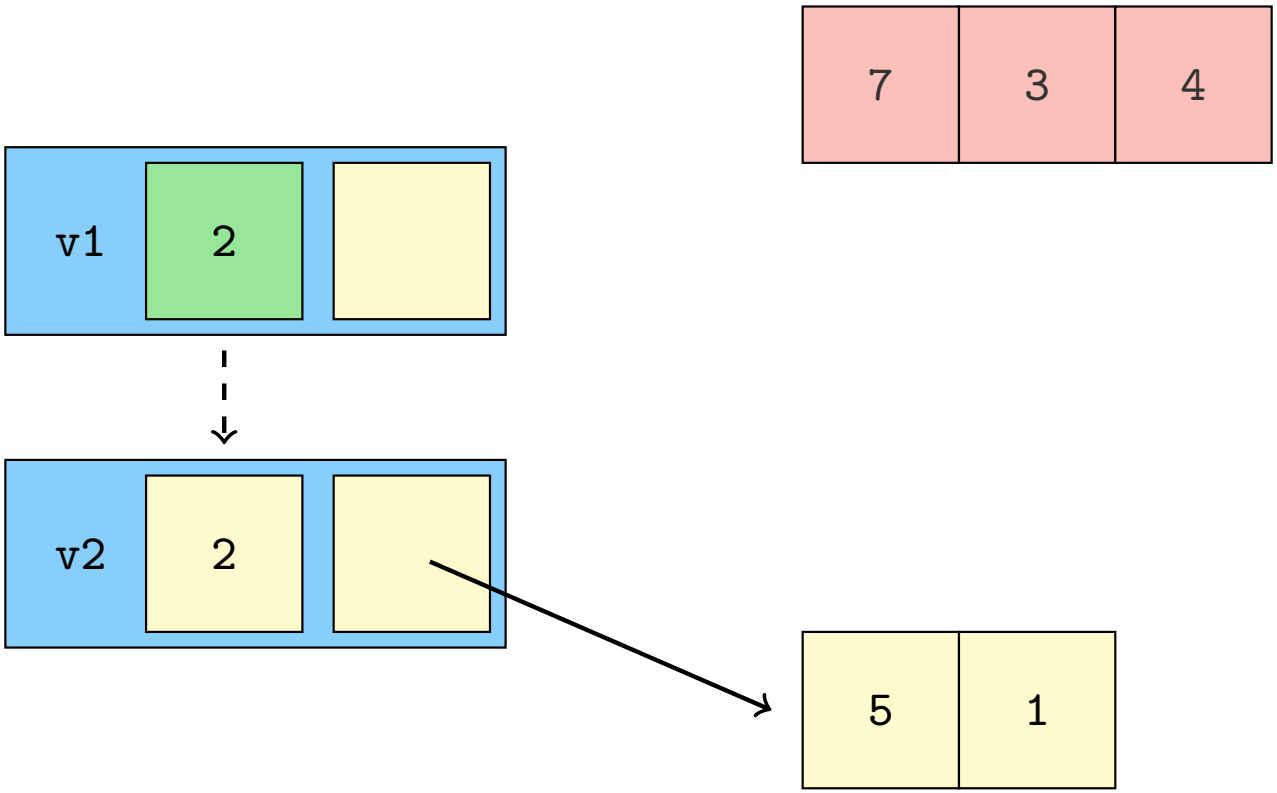
Copia profunda segura

```
v1 = v2;
```



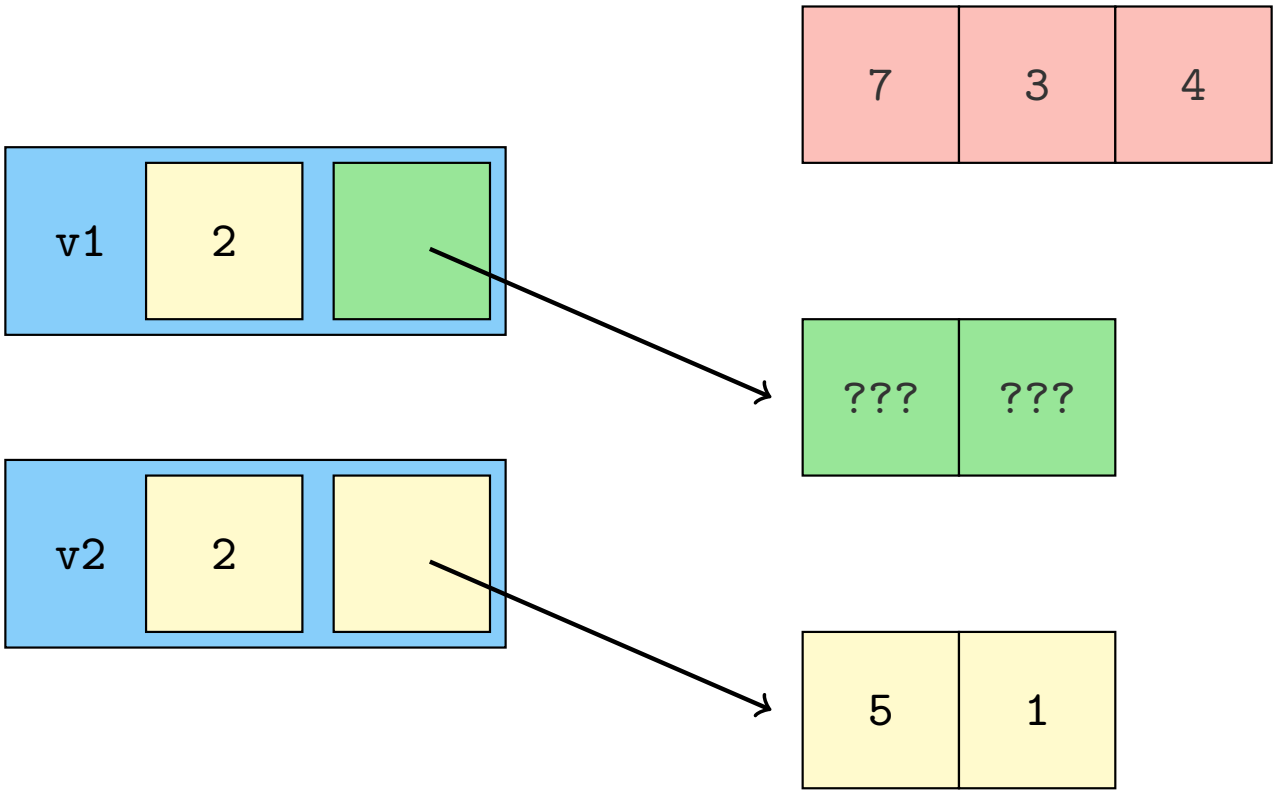
Copia profunda segura

```
v1 = v2;
```



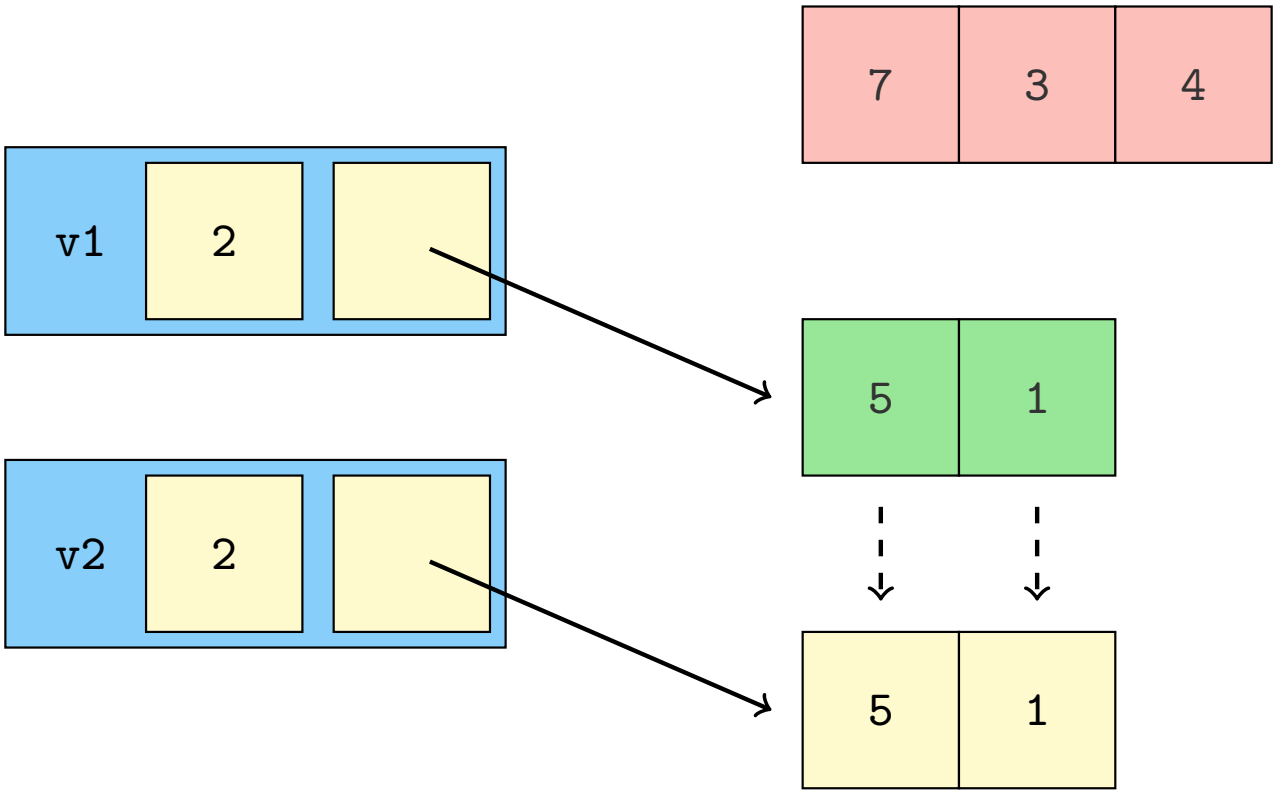
Copia profunda segura

```
v1 = v2;
```



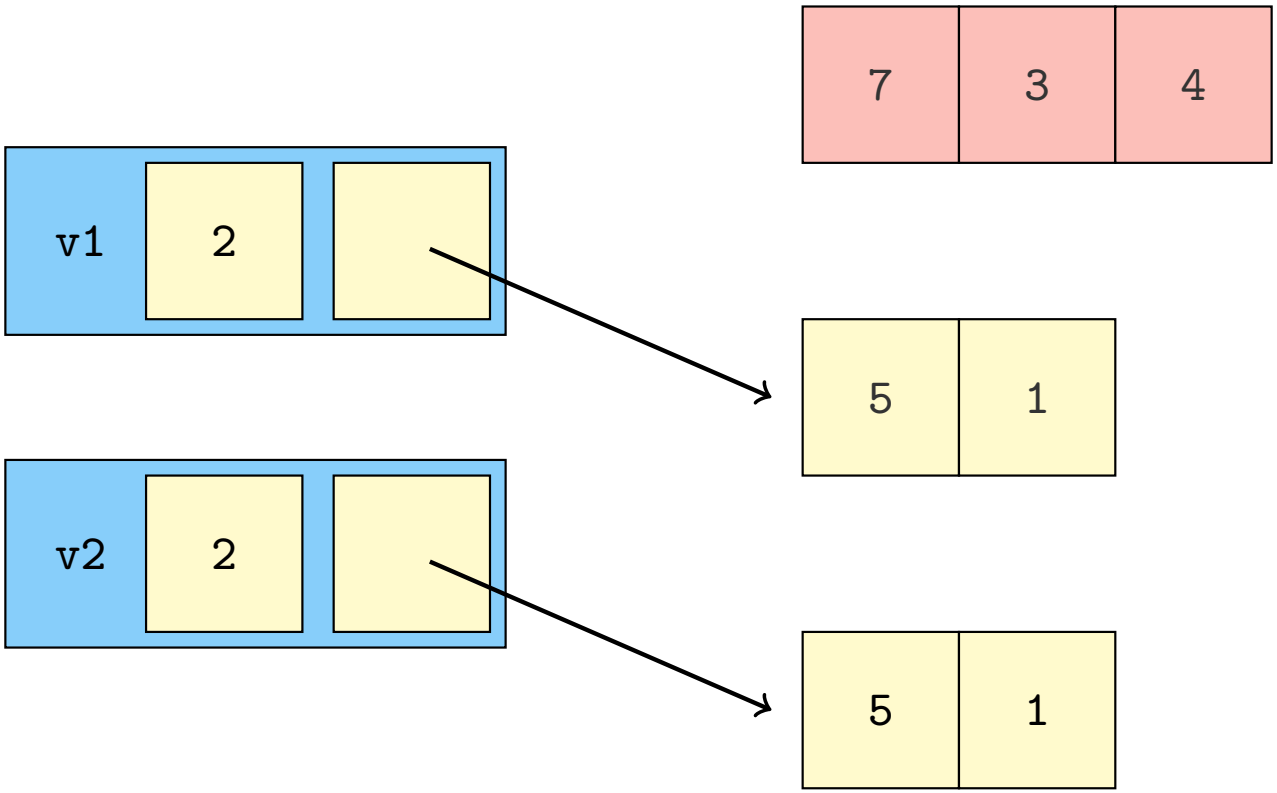
Copia profunda segura

```
v1 = v2;
```



Copia profunda segura

```
v1 = v2;
```



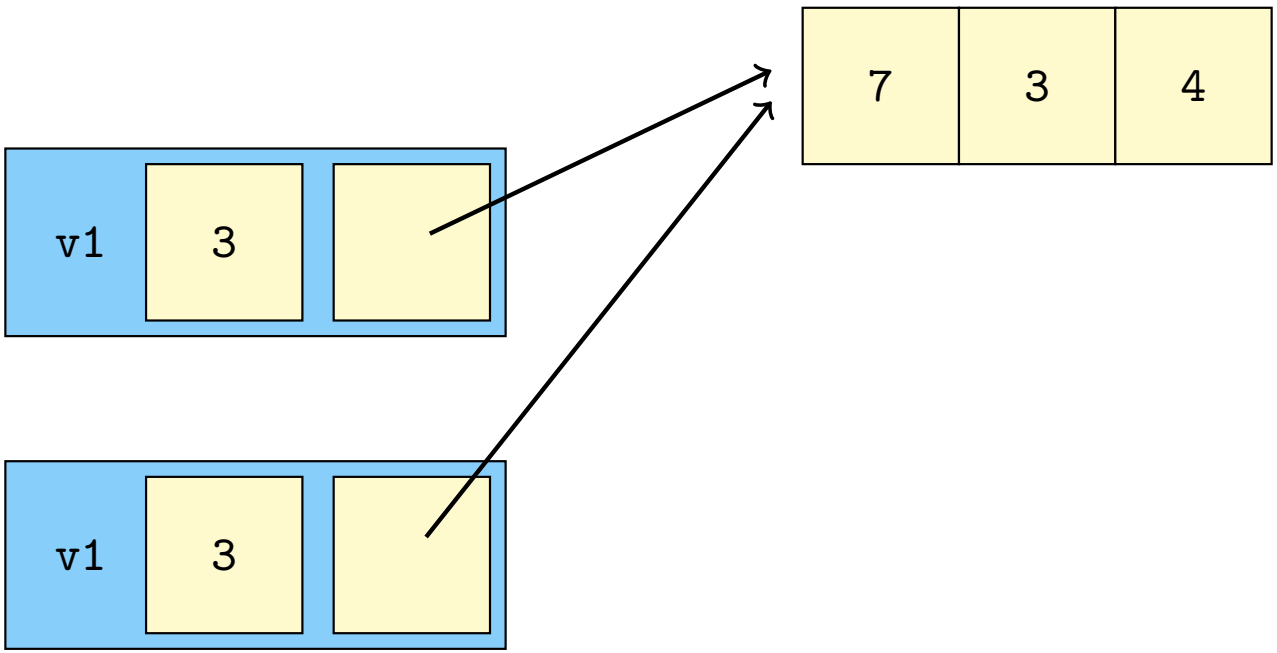
Copia profunda segura

Autoasignación de un vector por referencia constante

```
vector & operator = (const vector & rhs)
{
    delete [] array_;
    size_  = rhs.size_;
    array_ = new T[size_];
    std::copy(rhs.array_, rhs.array_ + size_, array_);
    return *this;
}

// ...

int main ()
{
    vector v1;
    v1 = v1;
}
```

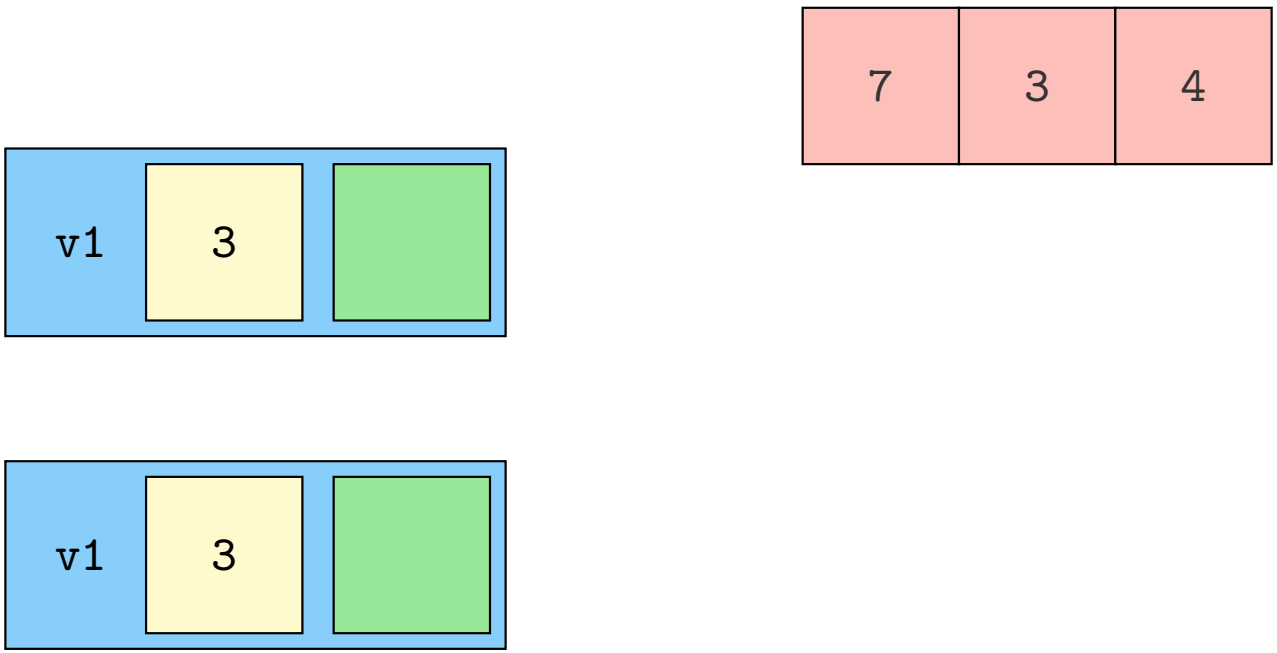
Copia profunda segura

```
v1 = v1;
```



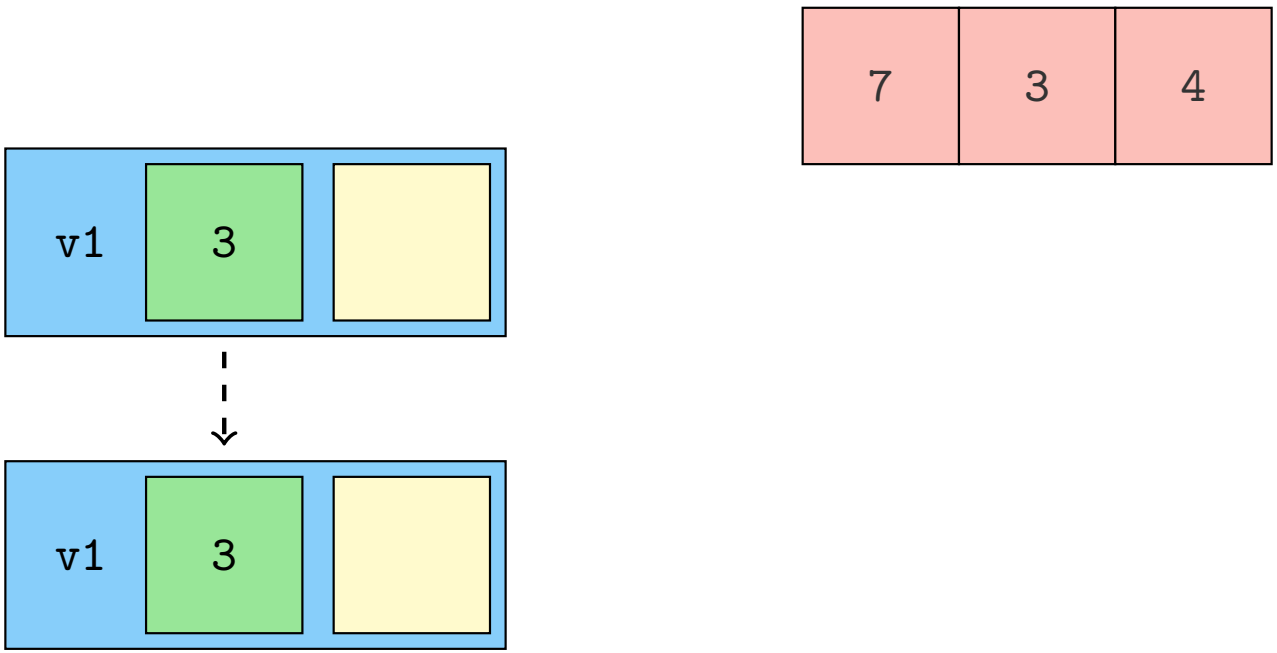
Copia profunda segura

```
v1 = v1;
```



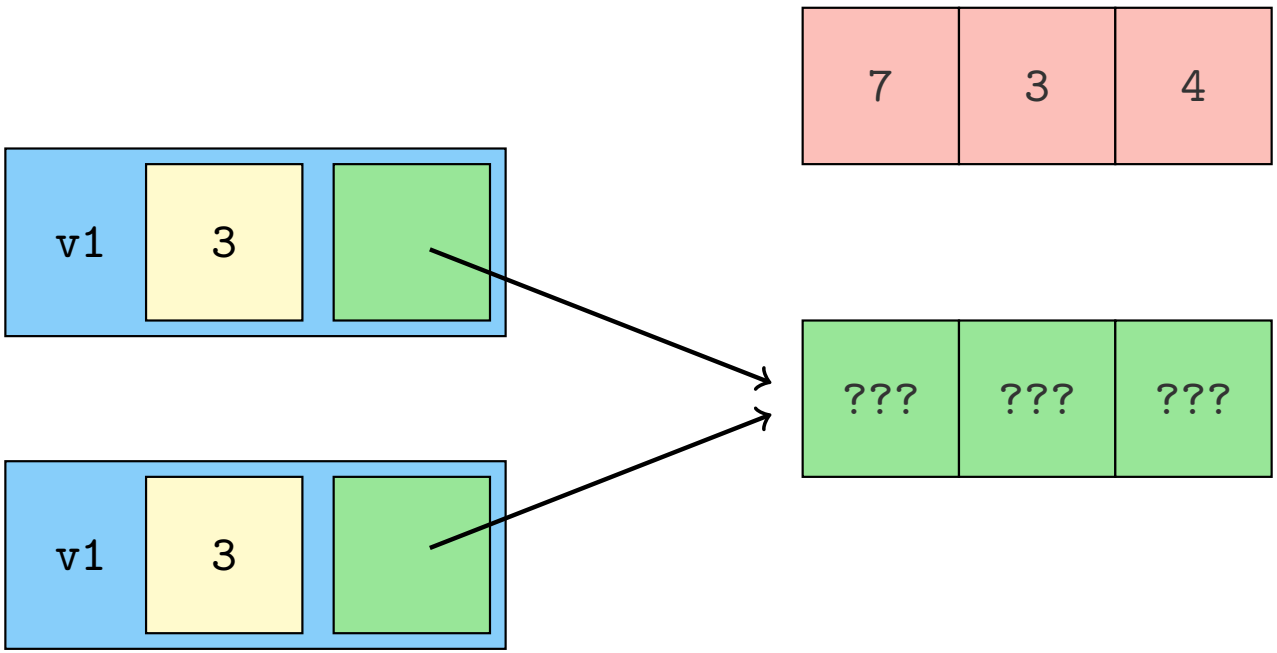
Copia profunda segura

```
v1 = v1;
```



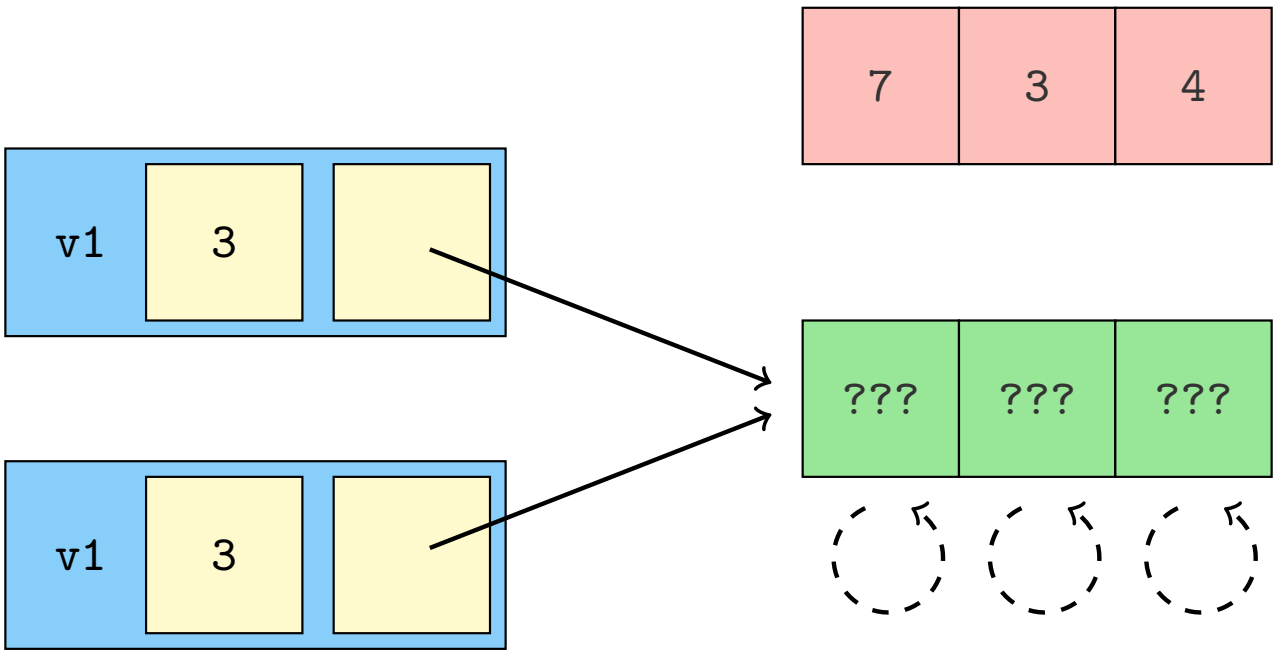
Copia profunda segura

```
v1 = v1;
```



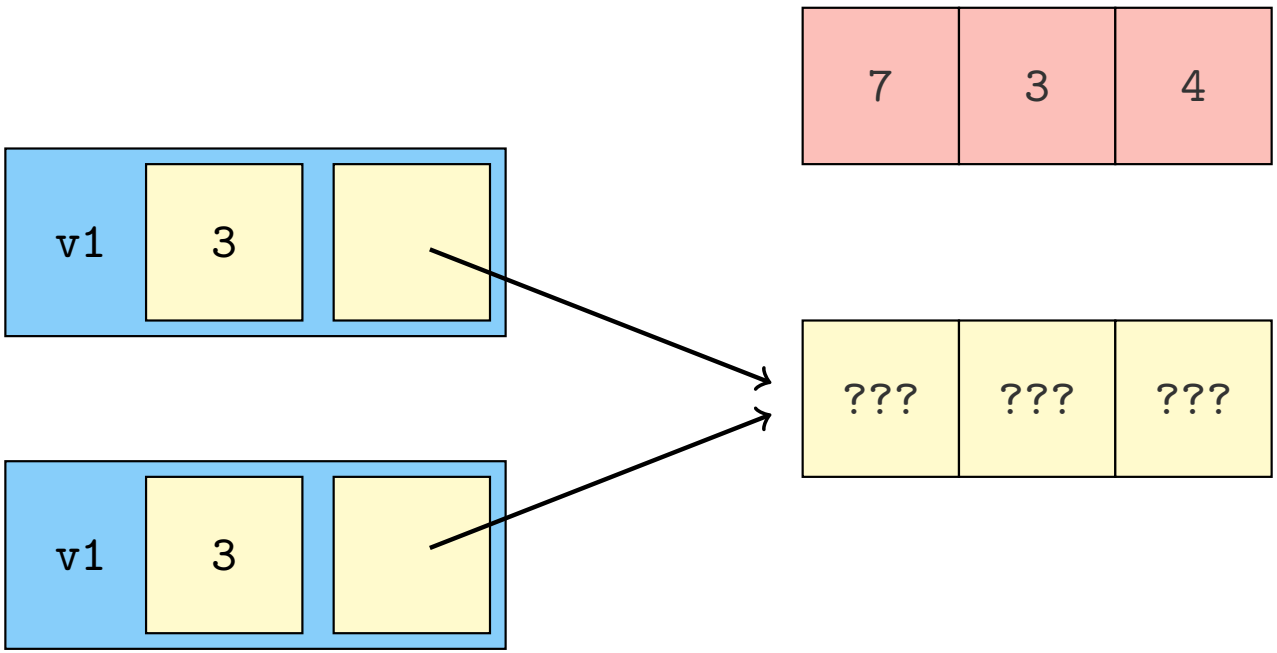
Copia profunda segura

```
v1 = v1;
```



Copia profunda segura

```
v1 = v1;
```



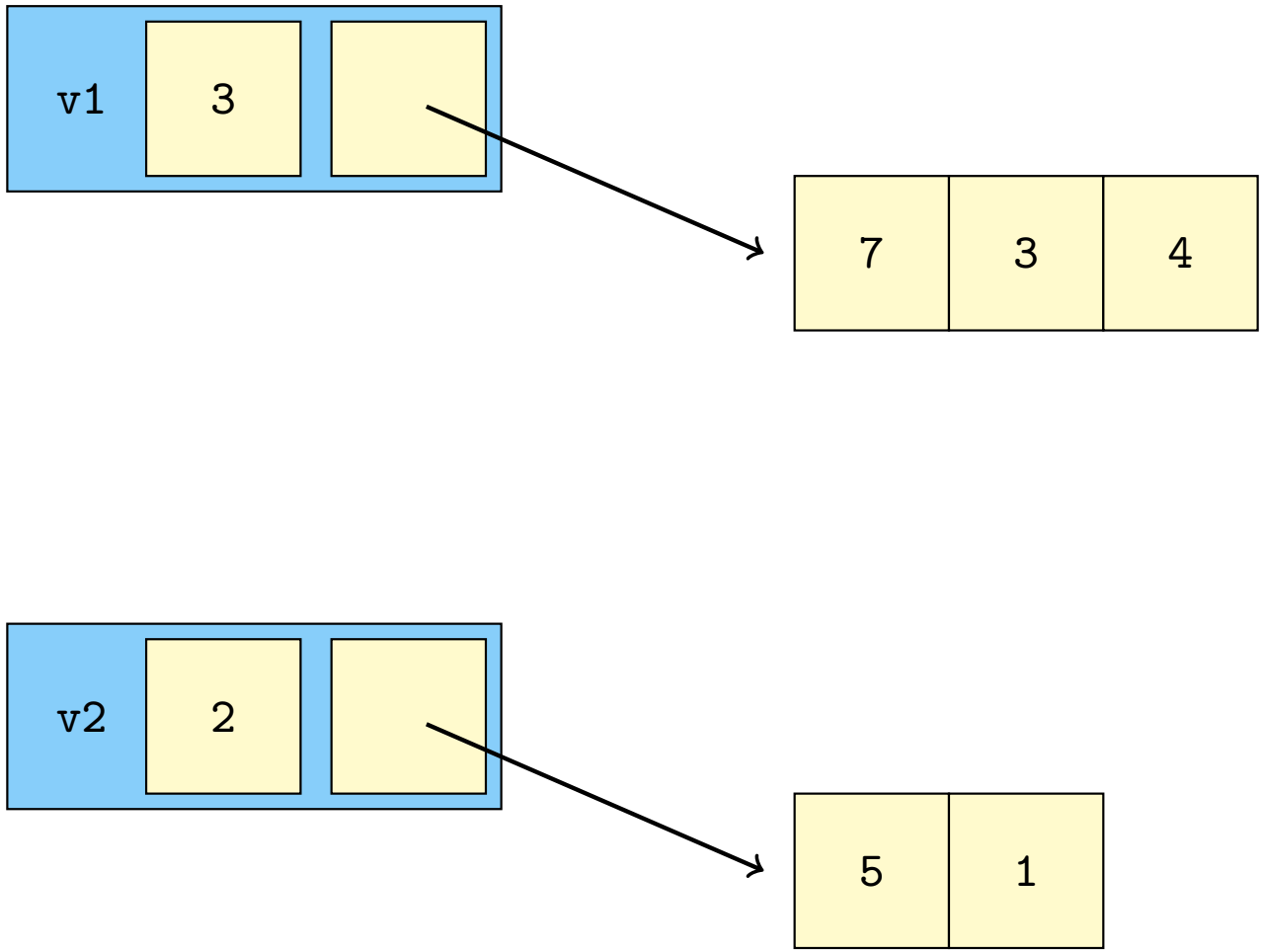
Copia profunda segura

Copia por copy-swap

```
class vector
{
private:
    int * array_;
    size_t size_;

public:
    vector () : array_ (nullptr), size_ (0) { }
    vector (const vector & rhs) { /* deepcopy */ }
    ~vector () { delete [] array_; }

    vector & operator = (const vector & rhs)
    {
        vector copy = rhs;
        copy.swap(*this);
        return *this;
    }

    void swap (vector & rhs)
    {
        std::swap(array_, rhs.array_);
        std::swap(size_, rhs.size_);
    }
};
```

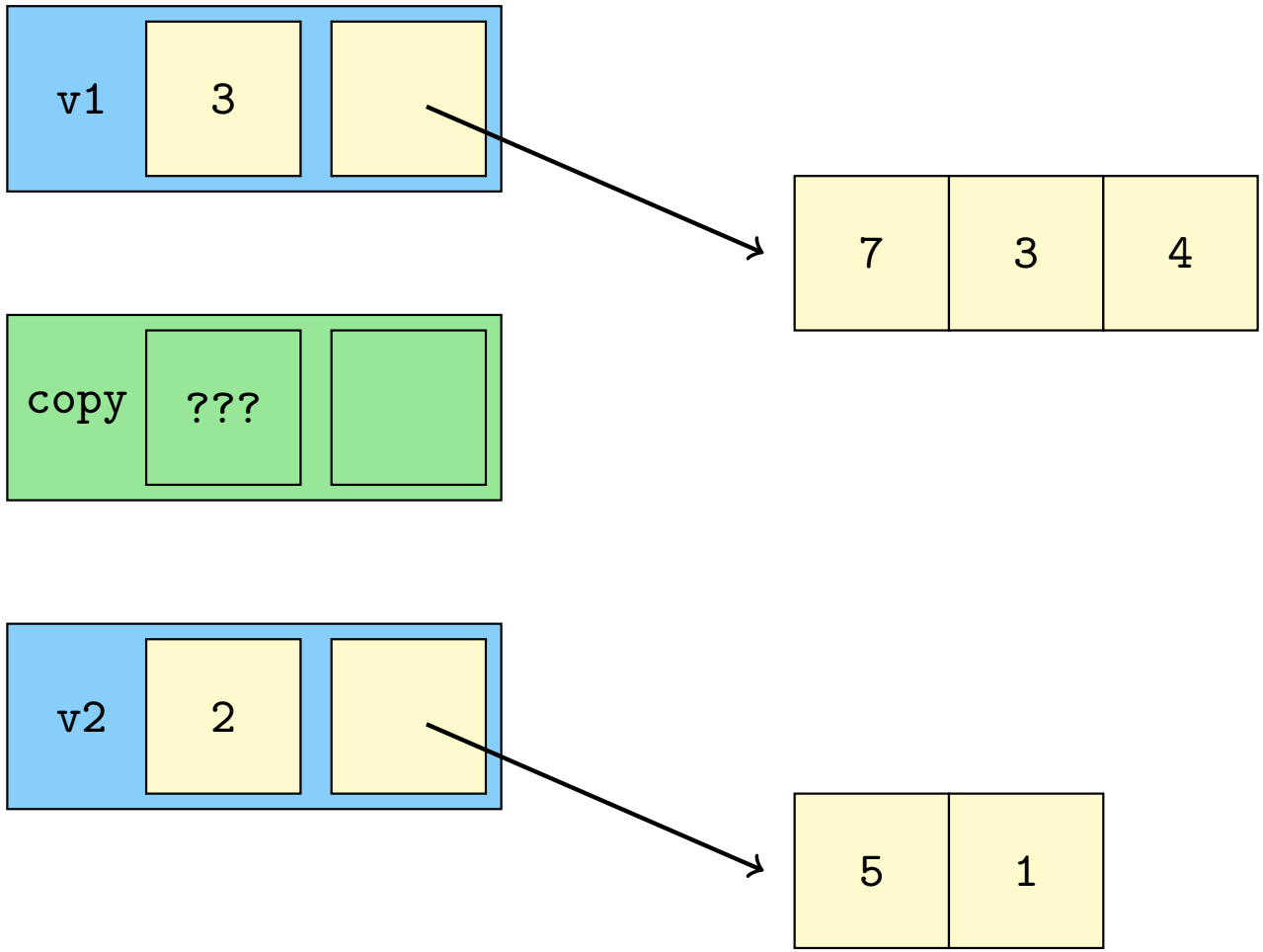
Copia profunda segura

```
v1 = v2; // copy-swap
```



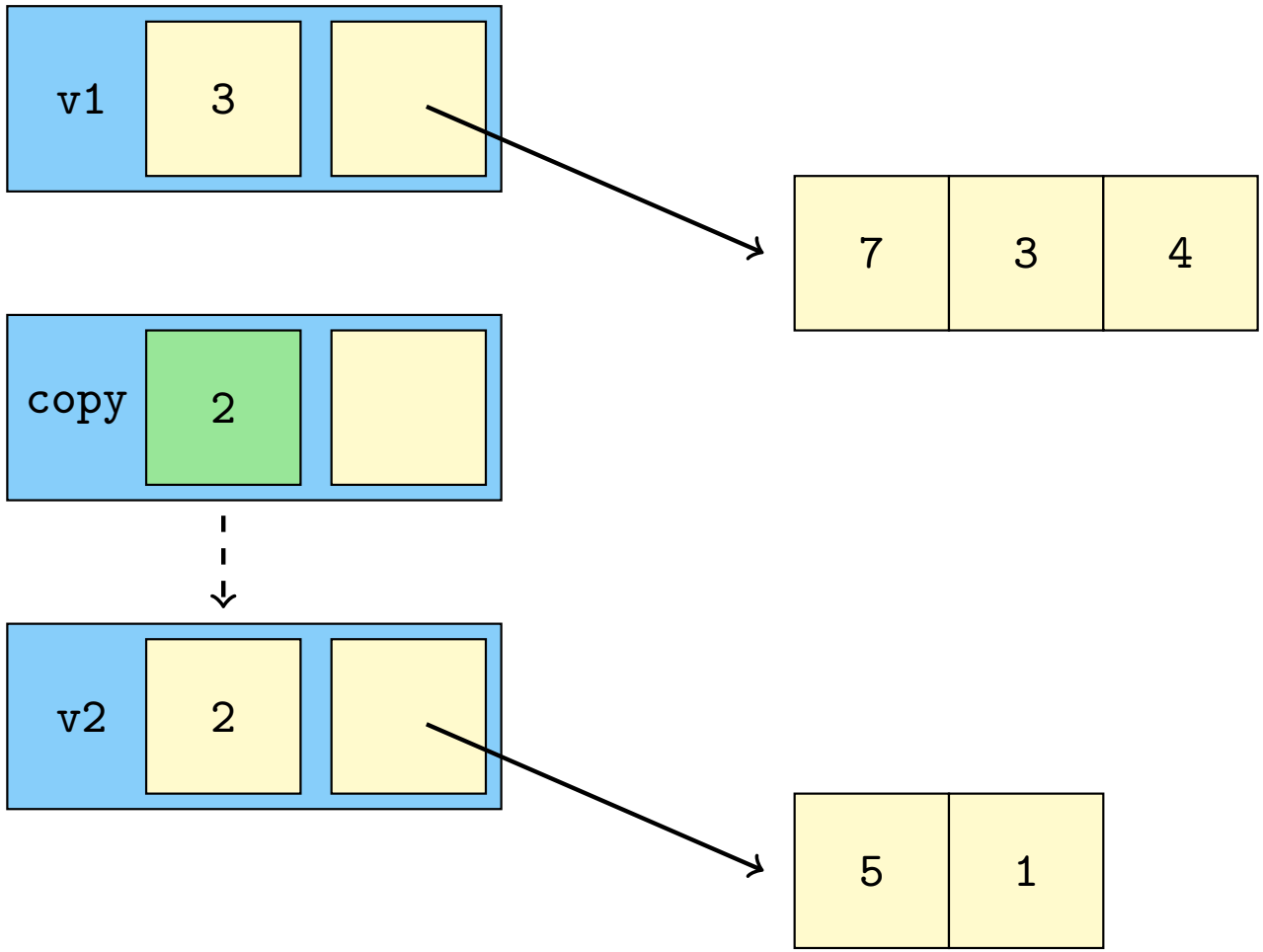
Copia profunda segura

```
v1 = v2; // copy-swap
```



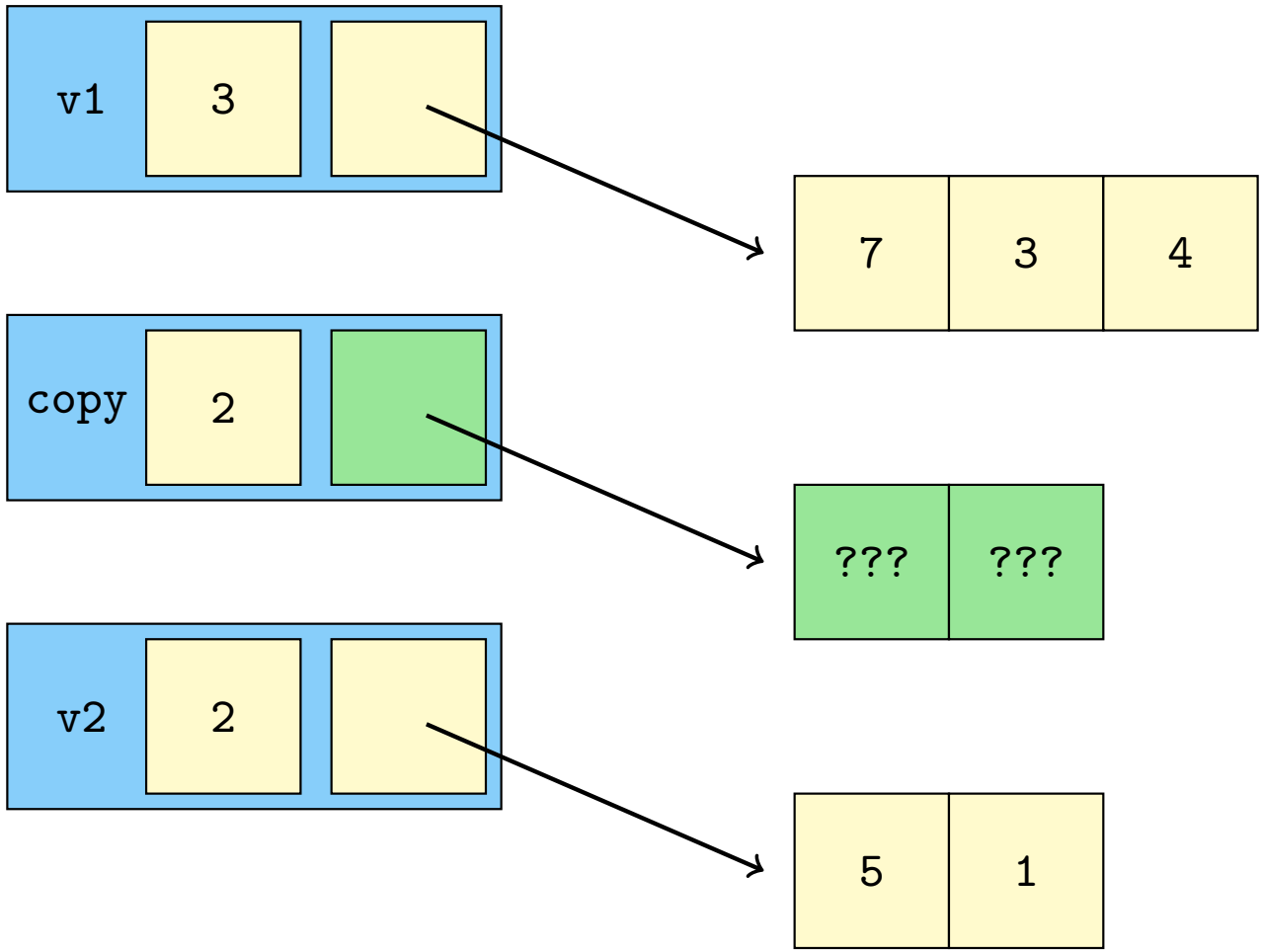
Copia profunda segura

```
v1 = v2; // copy-swap
```



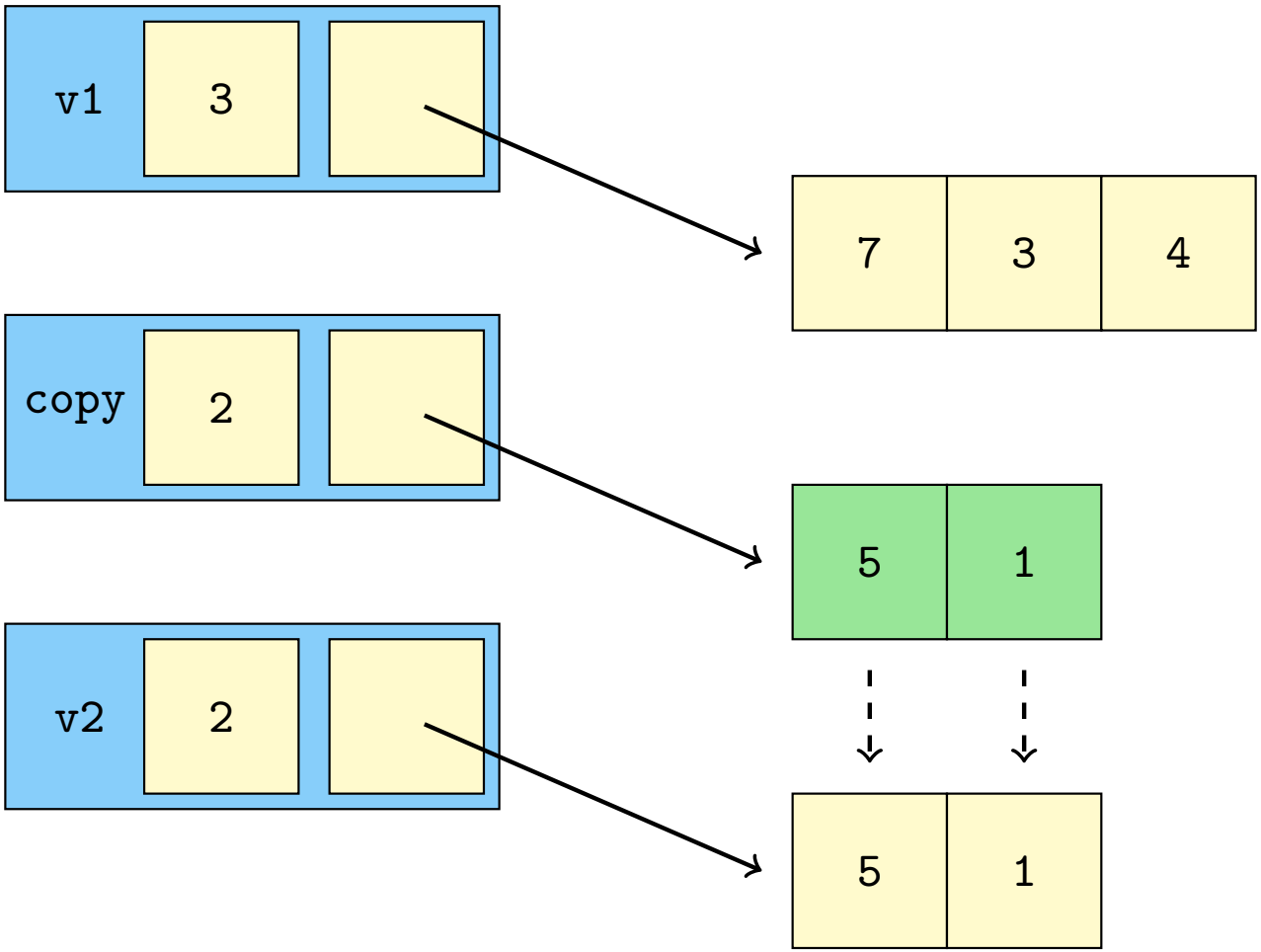
Copia profunda segura

```
v1 = v2; // copy-swap
```



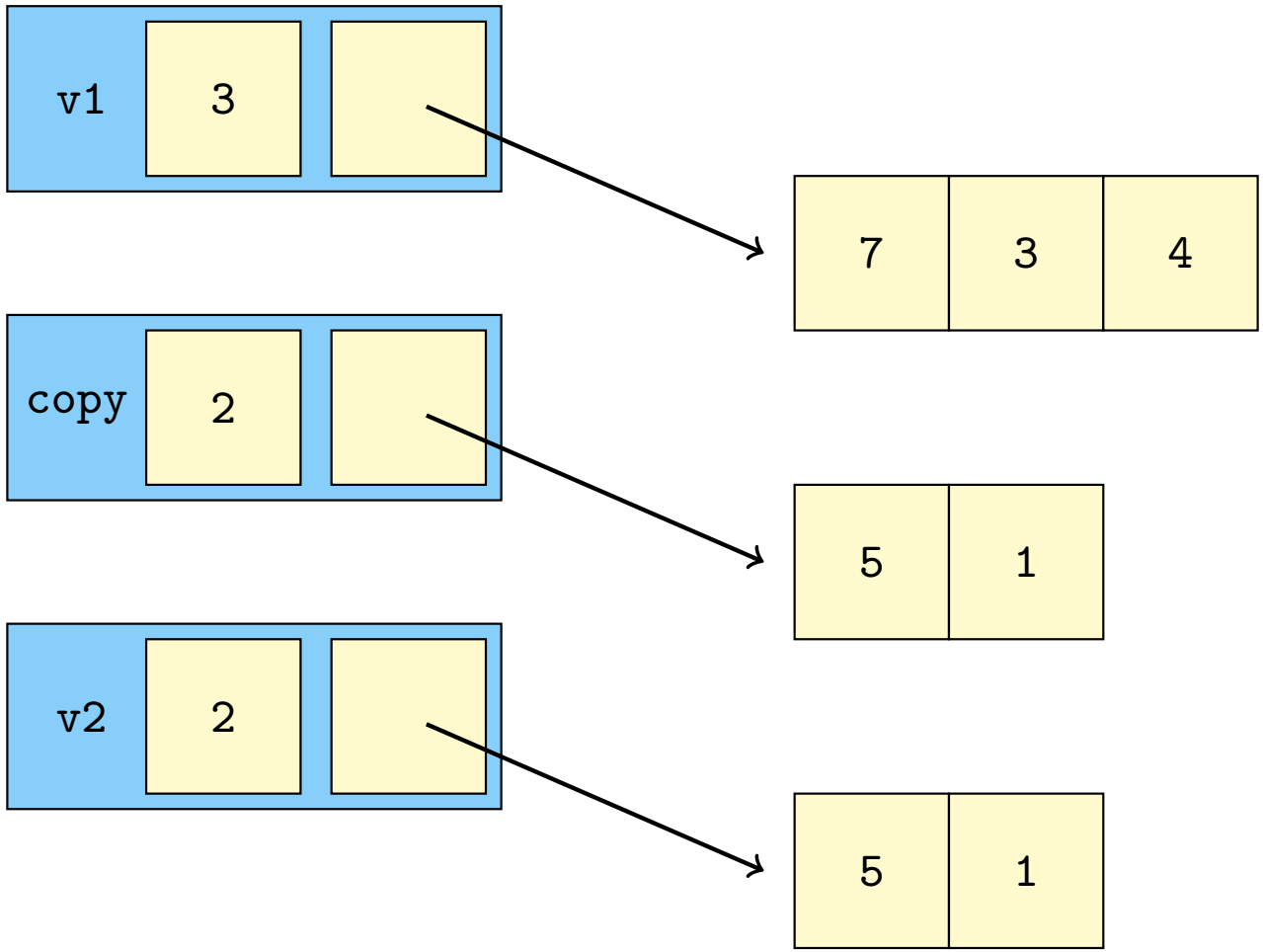
Copia profunda segura

```
v1 = v2; // copy-swap
```



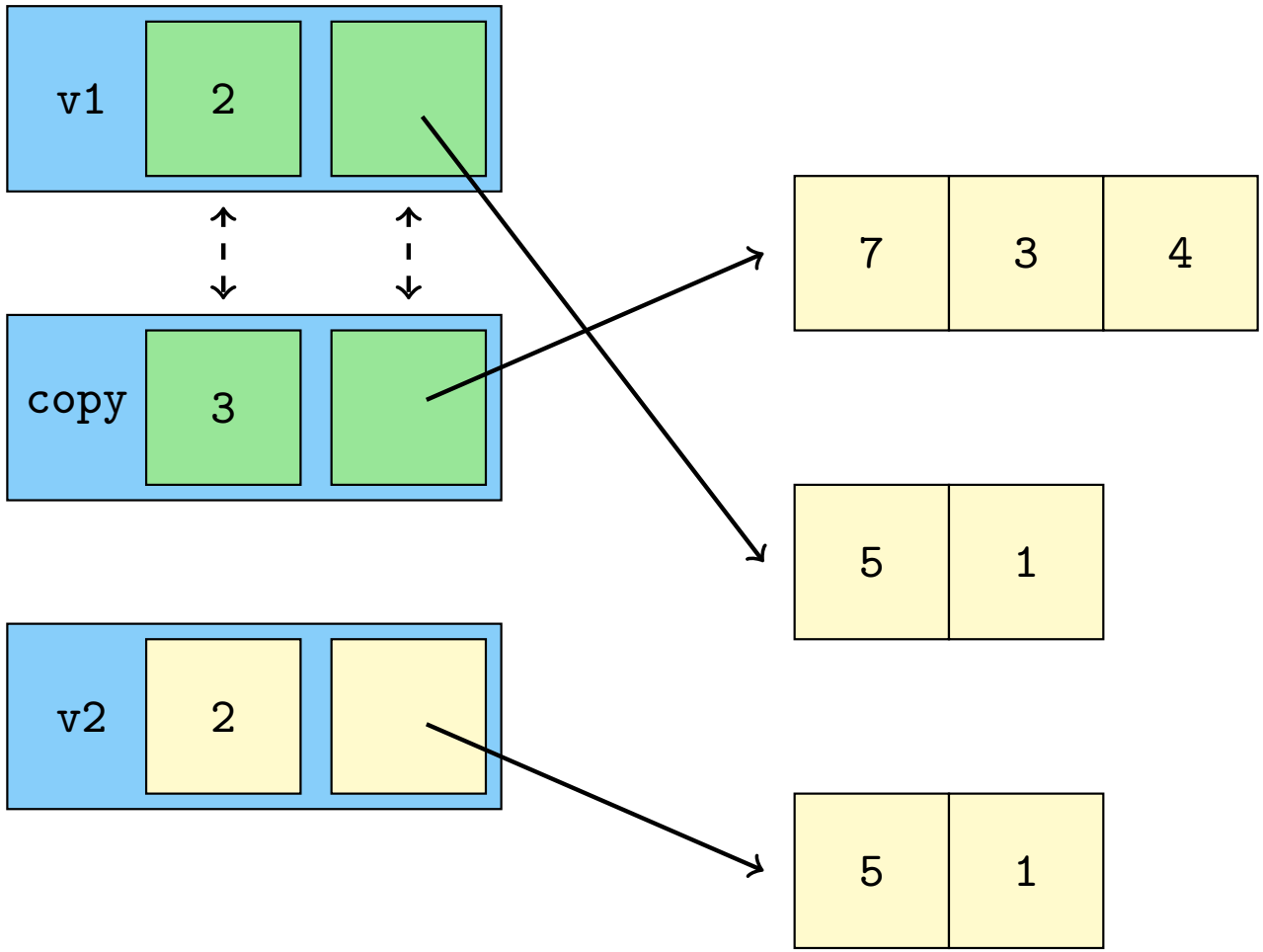
Copia profunda segura

```
v1 = v2; // copy-swap
```



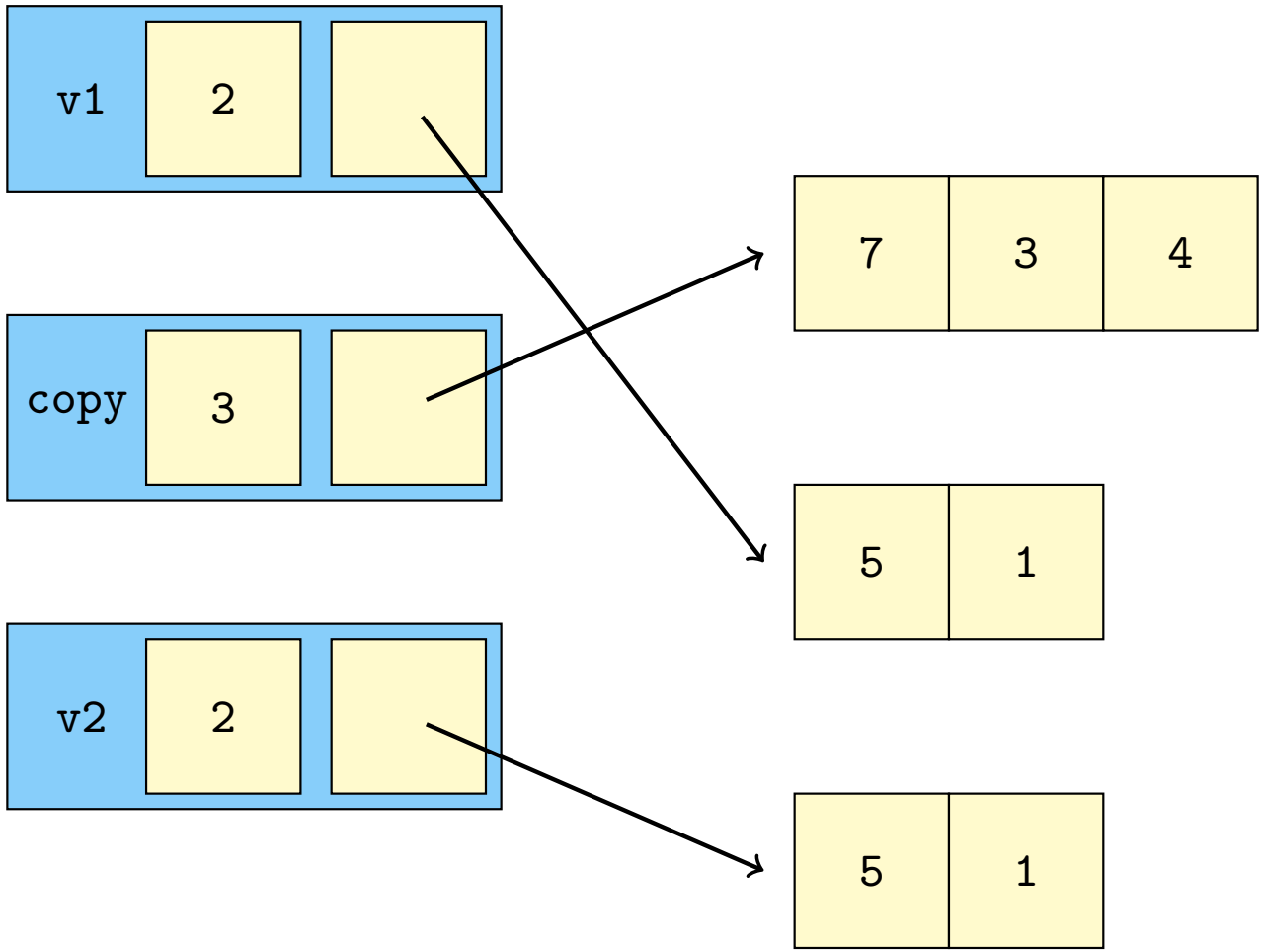
Copia profunda segura

```
v1 = v2; // copy-swap
```



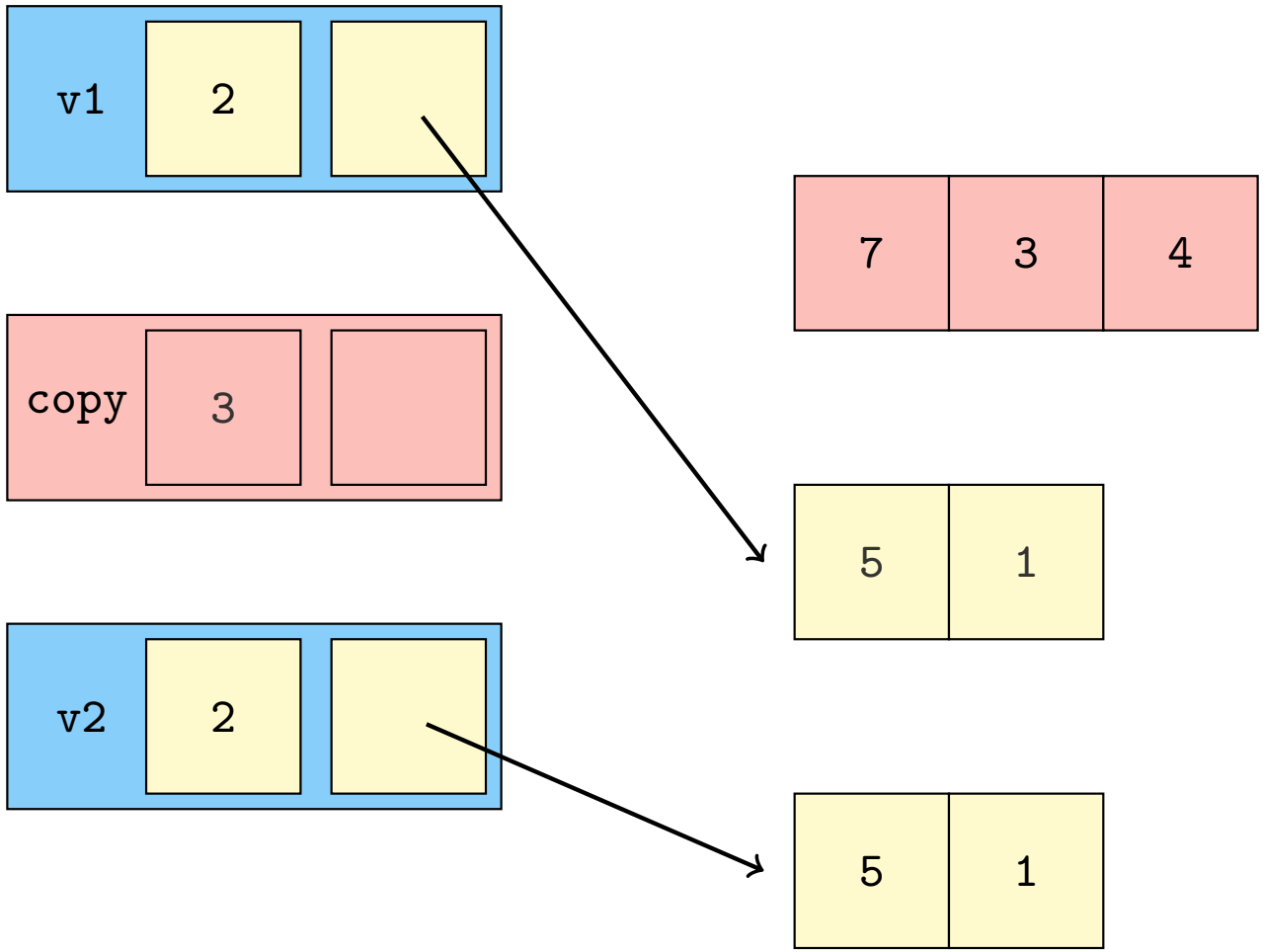
Copia profunda segura

```
v1 = v2; // copy-swap
```



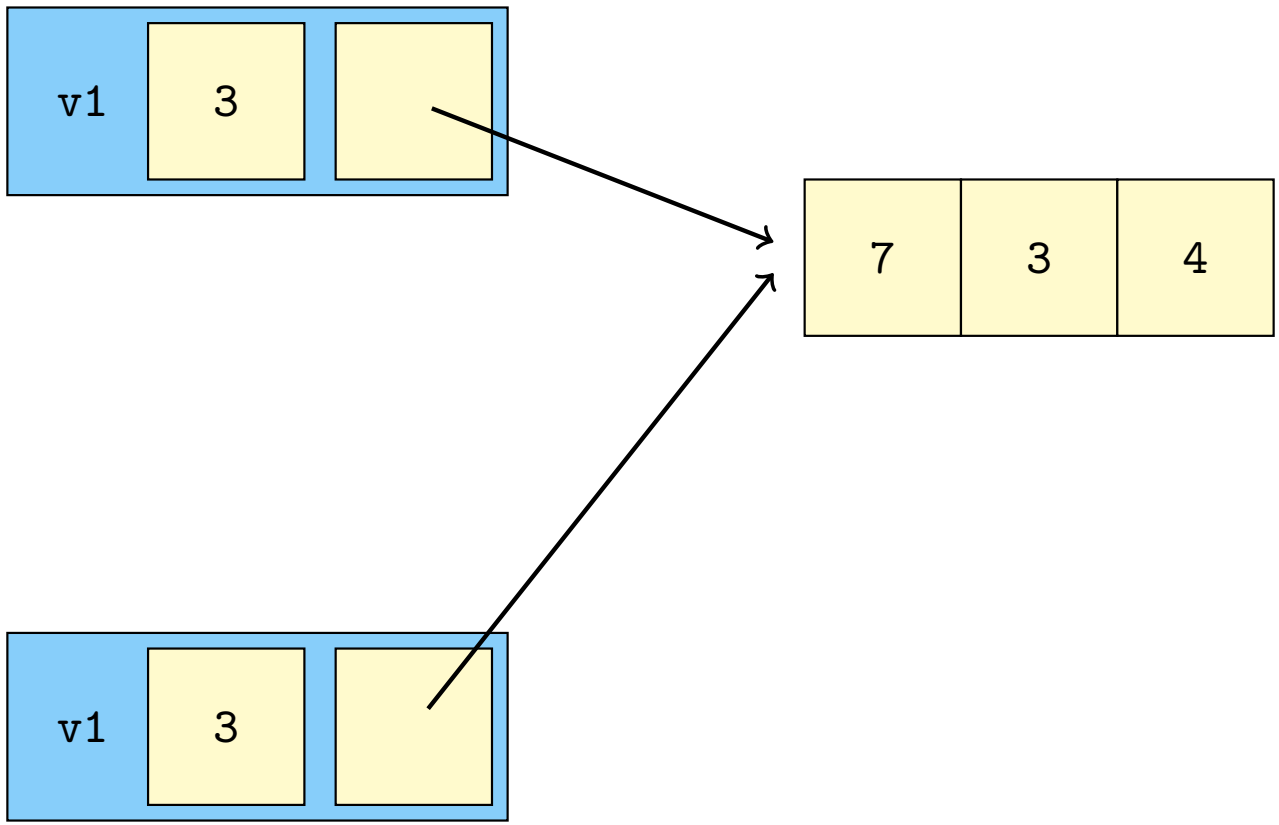
Copia profunda segura

```
v1 = v2; // copy-swap
```



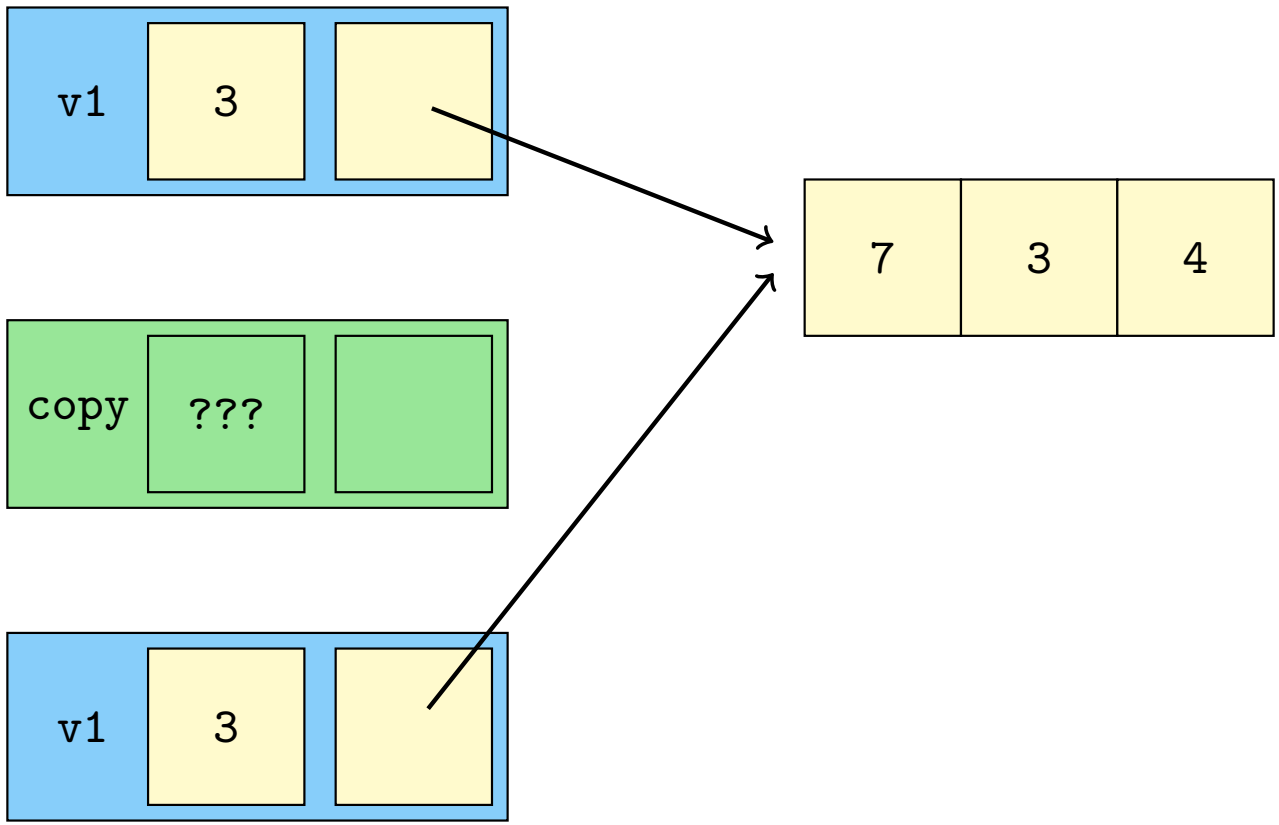
Copia profunda segura

```
v1 = v1; // copy-swap
```



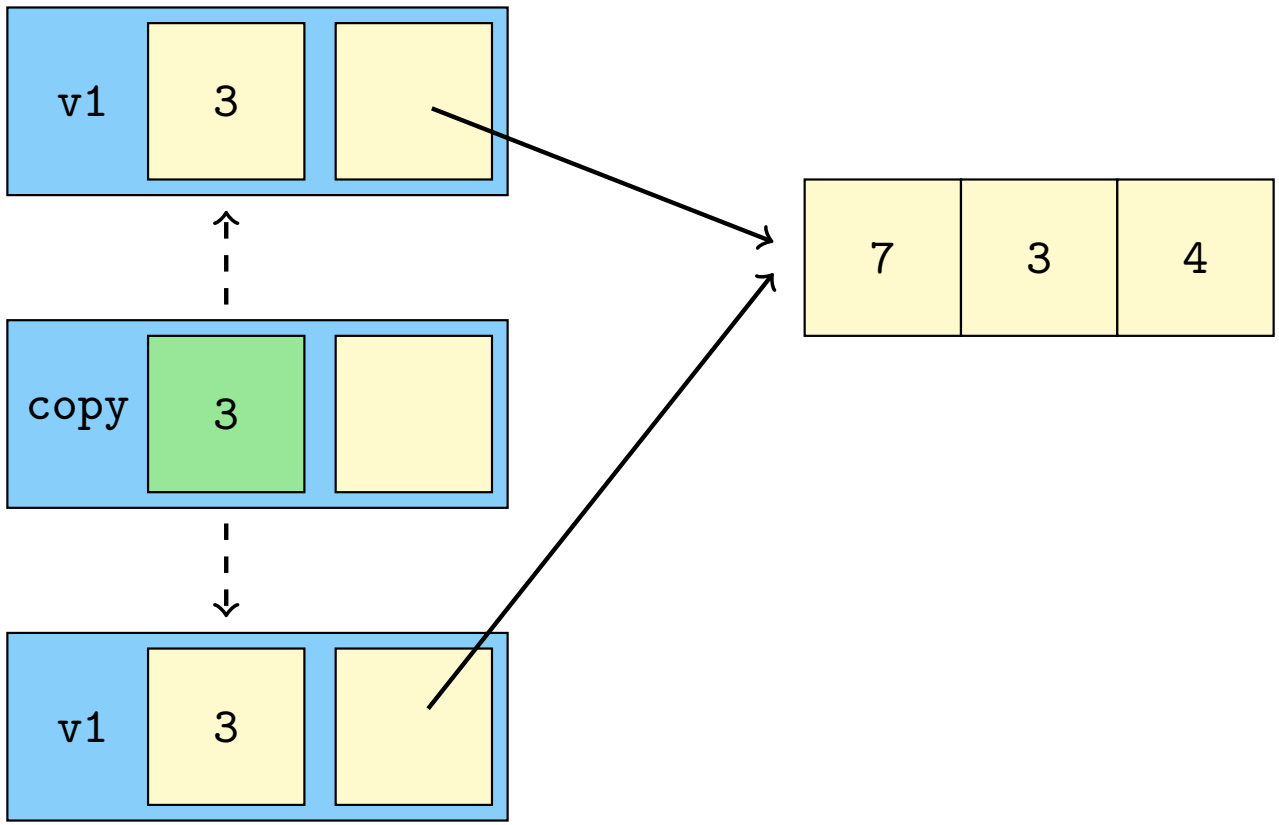
Copia profunda segura

```
v1 = v1; // copy-swap
```



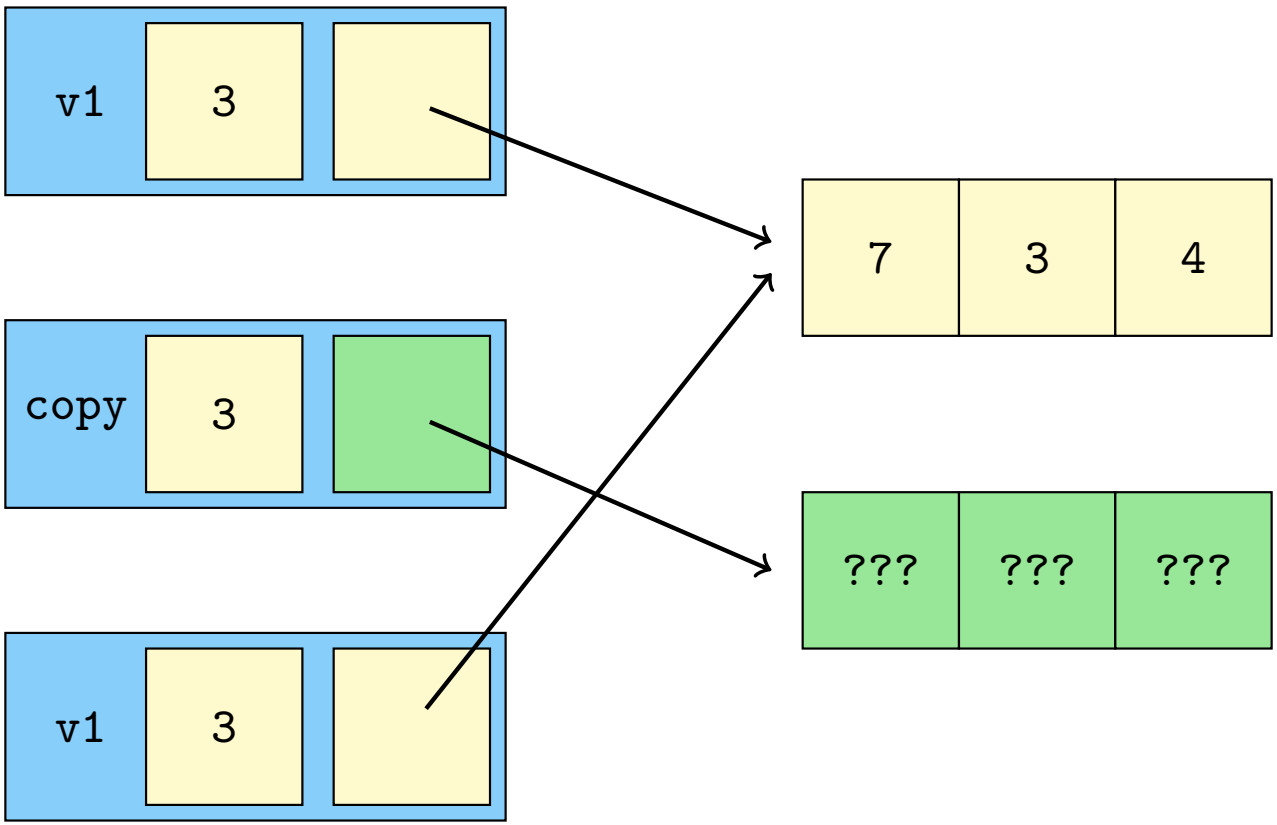
Copia profunda segura

```
v1 = v1; // copy-swap
```



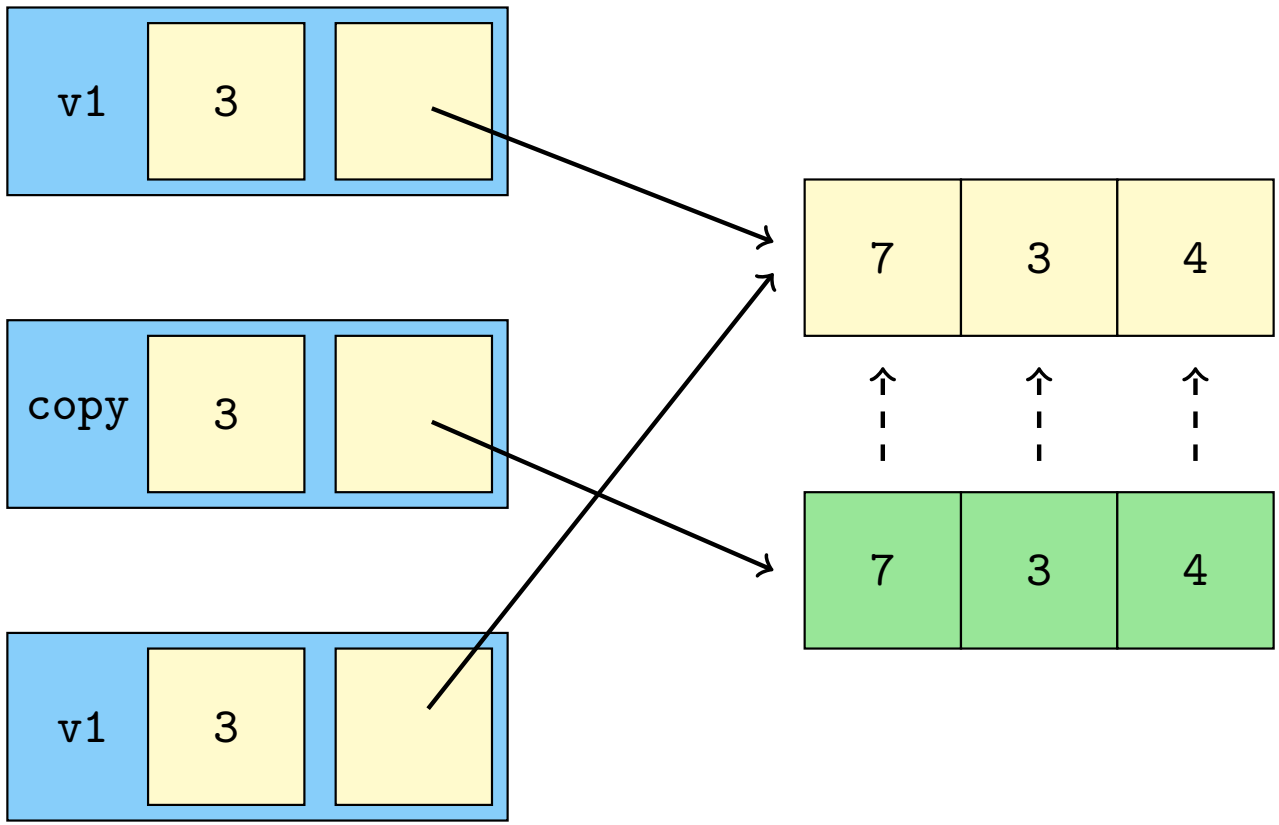
Copia profunda segura

```
v1 = v1; // copy-swap
```



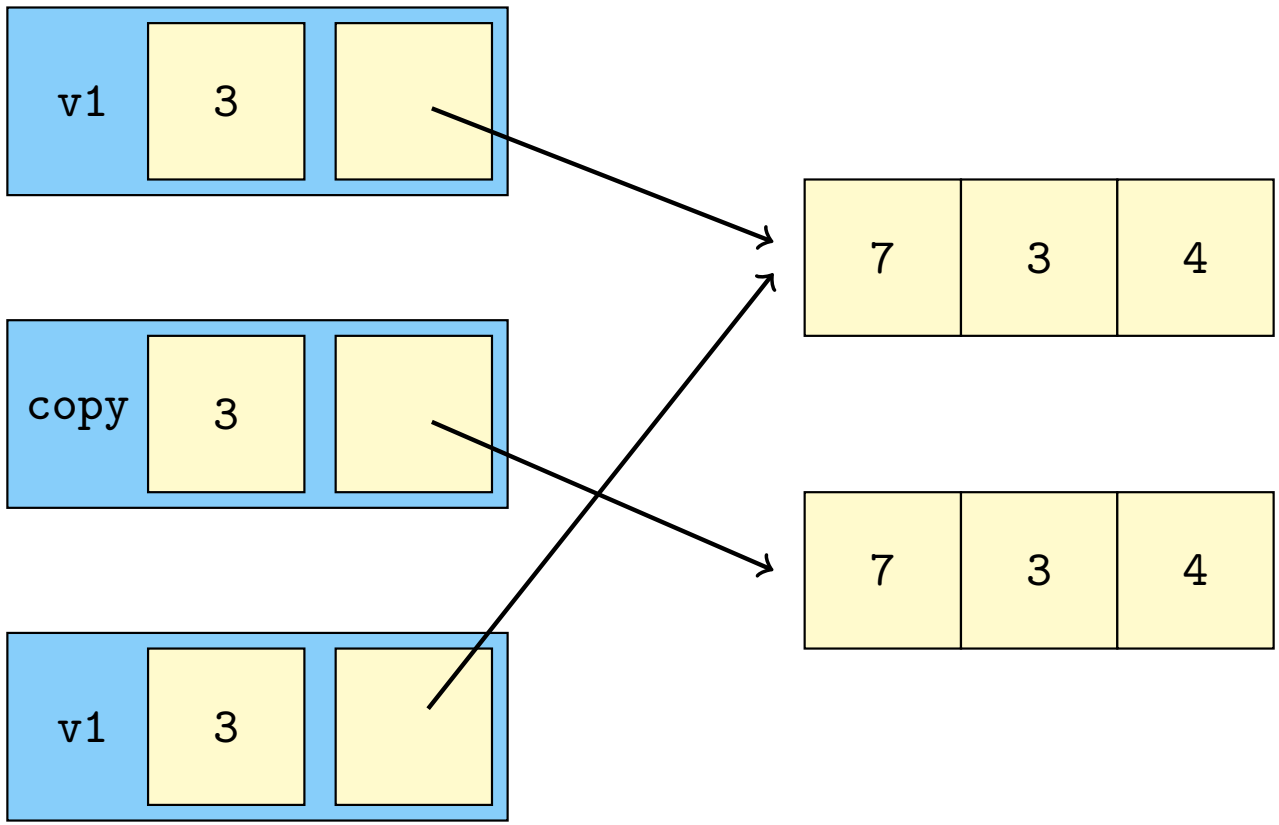
Copia profunda segura

```
v1 = v1; // copy-swap
```



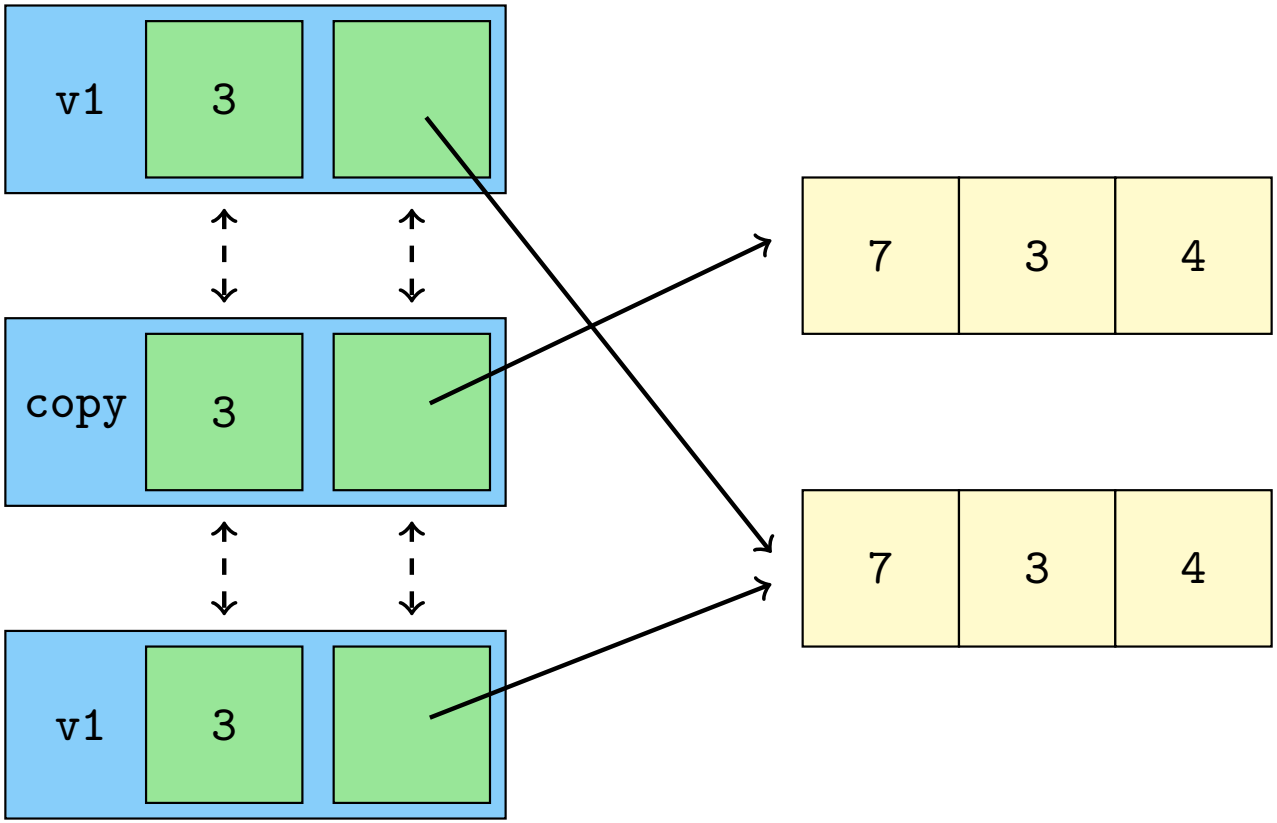
Copia profunda segura

```
v1 = v1; // copy-swap
```



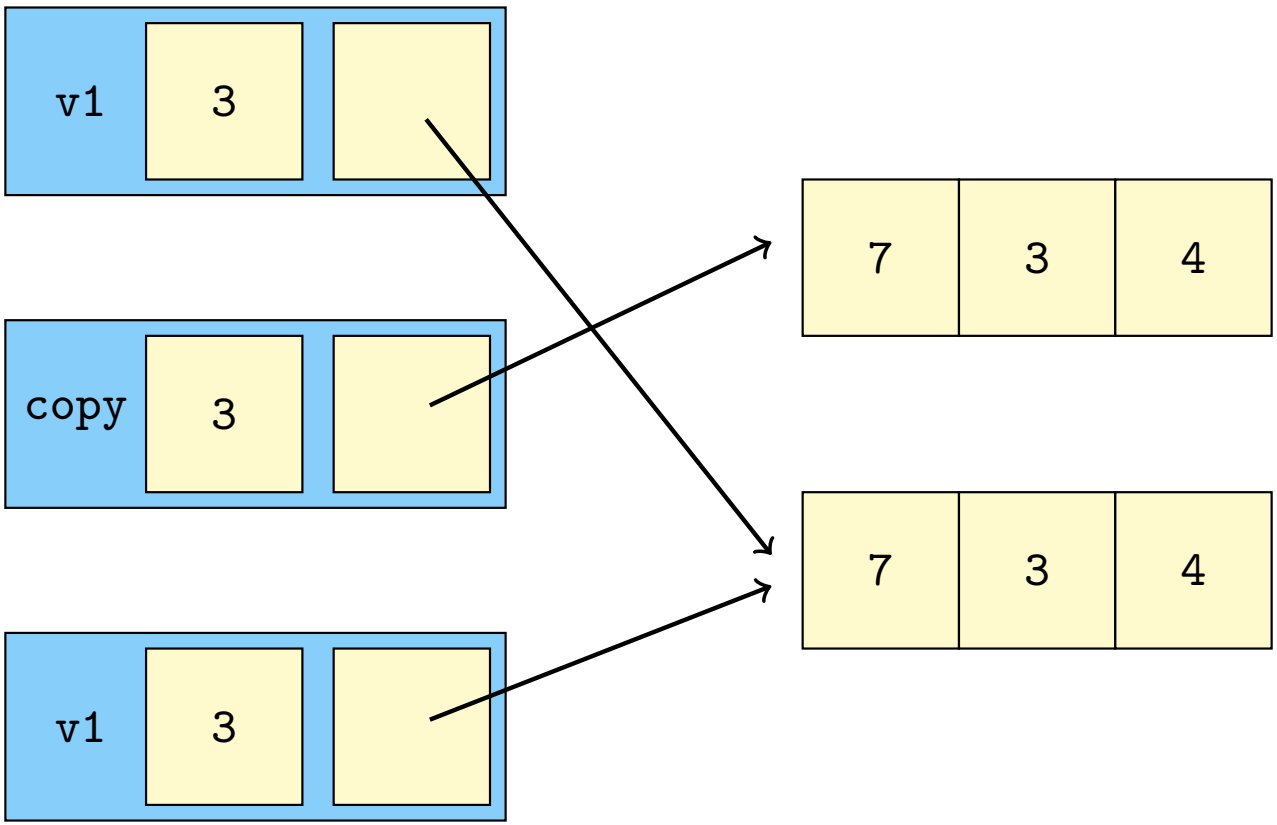
Copia profunda segura

```
v1 = v1; // copy-swap
```



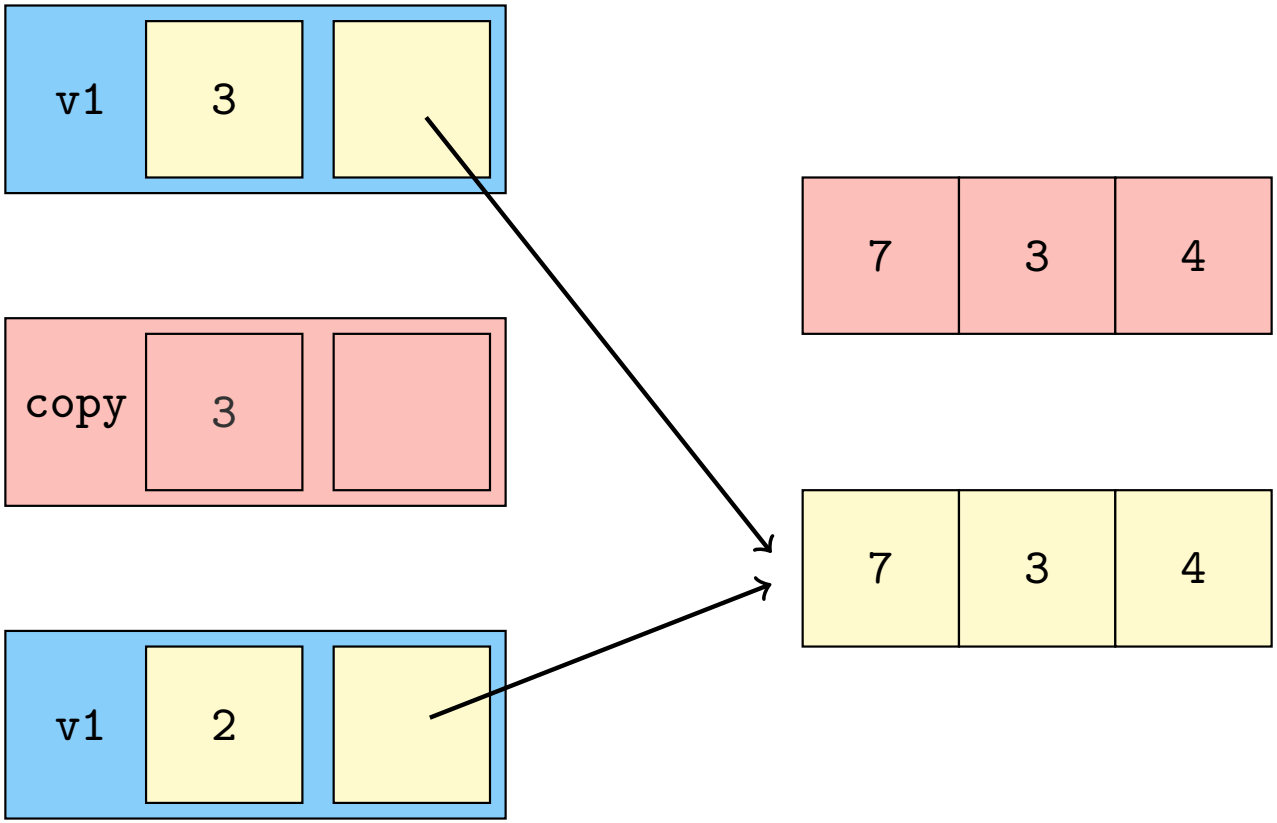
Copia profunda segura

```
v1 = v1; // copy-swap
```



Copia profunda segura

```
v1 = v1; // copy-swap
```



3: *Ranged* `for` (C++11)

Kate Gregory

What Do We Mean When We Say Nothing At All?

Ranged `for`

Impresión de los elementos de un `std::vector<T> vec`:

Bucle `for` al estilo C tradicional

```
for (size_t i = 0; i < vec.size(); i++)
    std::cout << vec[i] << " ";
```

`for (size_t i = 0; i <= vec.size(); i++)` @LocalPinkRobin

Bucle `for` con iteradores constantes

```
for (std::vector<T>::const_iterator it = vec.cbegin(); it != vec.cend(); ++it)
    std::cout << *it << " ";
```

Bucle `for` con iteradores constantes e inferencia de tipos

```
for (auto it = vec.cbegin(); it != vec.cend(); ++it)
    std::cout << *it << " ";
```

Ranged `for`

Impresión de los elementos de un `std::vector<T> vec`:

Bucle `for` al estilo C tradicional

```
for (size_t i = 0; i < vec.size(); i++)
   std::cout << vec[i] << " ";
```

`for (size_t i = 0; i <= vec.size(); i++)` @LocalPinkRobin

Bucle `for` con iteradores constantes

```
for (std::vector<T>::const_iterator it = vec.cbegin(); it != vec.cend(); ++it)
   std::cout << *it << " ";
```

Bucle `for` con iteradores constantes e inferencia de tipos

```
for (auto it = vec.cbegin(); it != vec.cend(); ++it)
   std::cout << *it << " ";
```

Ranged `for`

Impresión de los elementos de un `std::vector<T> vec`:

Bucle `for` al estilo C tradicional

```
for (size_t i = 0; i < vec.size(); i++)
   std::cout << vec[i] << " ";
```

`for (size_t i = 0; i <= vec.size(); i++)` @LocalPinkRobin

Bucle `for` con iteradores constantes

```
for (std::vector<T>::const_iterator it = vec.cbegin(); it != vec.cend(); ++it)
   std::cout << *it << " ";
```

Bucle `for` con iteradores constantes e inferencia de tipos

```
for (auto it = vec.cbegin(); it != vec.cend(); ++it)
   std::cout << *it << " ";
```

Ranged `for`

Impresión de los elementos de un `std::vector<T> vec`:

Bucle `for` al estilo C tradicional

```
for (size_t i = 0; i < vec.size(); i++)
   std::cout << vec[i] << " ";
```

`for (size_t i = 0; i <= vec.size(); i++)` @LocalPinkRobin

Bucle `for` con iteradores constantes

```
for (std::vector<T>::const_iterator it = vec.cbegin(); it != vec.cend(); ++it)
   std::cout << *it << " ";
```

Bucle `for` con iteradores constantes e inferencia de tipos

```
for (auto it = vec.cbegin(); it != vec.cend(); ++it)
   std::cout << *it << " ";
```

Ranged for

Bucle *ranged* for

```
for (auto item : vec)
   std::cout << item << " ";
```

Bucle *ranged* for constante

```
for (const auto item : vec)
   std::cout << item << " ";
```

Bucle *ranged* for sobre referencias

```
for (auto & item : vec)
   std::cout << item << " ";
```

Bucle *ranged* for sobre referencias constantes

```
for (auto const & item : vec)
   std::cout << item << " ";
```

Ranged `for`

Bucle *ranged* `for`

```
for (auto item : vec)
   std::cout << item << " ";
```

Bucle *ranged* `for` constante

```
for (const auto item : vec)
   std::cout << item << " ";
```

Bucle *ranged* `for` sobre referencias

```
for (auto & item : vec)
   std::cout << item << " ";
```

Bucle *ranged* `for` sobre referencias constantes

```
for (auto const & item : vec)
   std::cout << item << " ";
```

Ranged `for`

Bucle *ranged* `for`

```
for (auto item : vec)
    std::cout << item << " ";
```

Bucle *ranged* `for` constante

```
for (const auto item : vec)
    std::cout << item << " ";
```

Bucle *ranged* `for` sobre referencias

```
for (auto & item : vec)
    std::cout << item << " ";
```

Bucle *ranged* `for` sobre referencias constantes

```
for (auto const & item : vec)
    std::cout << item << " ";
```

Bucle *ranged* `for`

```
for (auto item : vec)
    std::cout << item << " ";
```

Bucle *ranged* `for` constante

```
for (const auto item : vec)
    std::cout << item << " ";
```

Bucle *ranged* `for` sobre referencias

```
for (auto & item : vec)
    std::cout << item << " ";
```

Bucle *ranged* `for` sobre referencias constantes

```
for (auto const & item : vec)
    std::cout << item << " ";
```

Mucho cuidado con el paso por valor

```
#include <iostream>
#include <vector>

void print_vec (const std::vector<int> & vec)
{
   for (auto v : vec)
      std::cout << v << ', ';

   std::cout << " :: ";
}

int main ()
{
   std::vector<int> vec{ -1, 0, 1 };
   print_vec(vec);

   for (auto v : vec)
      v = 2;

   print_vec(vec);
}
```

Salida del programa: `-1, 0, 1 :: -1, 0, 1` @advy99

4: `constexpr` (C++11/17)

Muchas veces encontramos funciones que son caras de ejecutar

```
int nth_triangle (const int x)
{
    int result = 0;

    for (int i = 0; i < x; i++)
        result += i;

    return result;
}

int factorial (const int x)
{
    int result = 1;

    if (x > 1)
        result = x * factorial(x-1);

    return result;
}

int main ()
{
   return nth_triangle(20) + factorial(5);
}
```

El compilador no es mágico y únicamente optimiza la ejecución

```
nth_triangle(int):
        mov     eax, 0
        mov     edx, 0
        jmp     .L2
.L3:
        add     edx, eax
        add     eax, 1
.L2:
        cmp     eax, edi
        jl      .L3
        mov     eax, edx
        ret
factorial(int):
        cmp     edi, 1
        jg      .L11
        mov     eax, 1
        ret
.L11:
        push    rbx
        mov     ebx, edi
        lea     edi, [rdi-1]
        call    factorial(int)
        imul    eax, ebx
        pop     rbx
        ret
main:
        push    rbx
        mov     edi, 20
        call    nth_triangle(int)
        mov     ebx, eax
        mov     edi, 5
        call    factorial(int)
        add     eax, ebx
        pop     rbx
        ret
```

Solución: Indicar que las funciones son constexpr

```
constexpr int nth_triangle (const int x)
{
    int result = 0;

    for (int i = 0; i < x; i++)
        result += i;

    return result;
}

constexpr int factorial (const int x)
{
    int result = 1;

    if (x > 1)
        result = x * factorial(x-1);

    return result;
}

int main ()
{
   return nth_triangle(20) + factorial(5);
}
```

El compilador se encarga de ahorrarle el trabajo al usuario

```
main:
        mov     eax, 310
        ret
```

Las macros dificultan la lectura y dependen del preprocesador

```
void Widget :: show ()
{
    #ifdef LEXI_WIDGET_DEBUG
        check_widget_state_valid_or_abort();
    #endif

    inner_element_.show();
}
```

Peor todavía: Programar con macros es programar con maldad

```
void Widget :: show ()
{
    #ifdef LEXI_WIDGET_DEBUG
        if (widget_state_is_valid())
    #endif

    inner_element_.show();
}
```

En este último caso -Wmisleading-indentation no se activa. @Melchor629

Las macros dificultan la lectura y dependen del preprocesador

```
void Widget :: show ()
{
    #ifdef LEXI_WIDGET_DEBUG
        check_widget_state_valid_or_abort();
    #endif

    inner_element_.show();
}
```

Peor todavía: Programar con macros es programar con maldad

```
void Widget :: show ()
{
    #ifdef LEXI_WIDGET_DEBUG
        if (widget_state_is_valid())
    #endif

    inner_element_.show();
}
```

En este último caso `-Wmisleading-indentation` no se activa. @Melchor629

Solución: if constexpr

```
class Widget
{
public:
    #ifdef LEXI_WIDGET_DEBUG
        static constexpr bool DEBUG = true;
    #else
        static constexpr bool DEBUG = false;
    #endif

    // ...
};

void Widget :: show ()
{
    if constexpr (Widget::DEBUG)
        check_widget_state_valid_or_abort();

    inner_element_.show();
}
```

5: [[nodiscard]] (C++17)

Descarte de valor por accidente

```
int Foo :: find_nearest (
    const int candidate,
    const std::vector<int>::const_iterator begin,
    const std::vector<int>::const_iterator end
{
    int nearest = candidate;
    // ...
    return nearest;
}

void Foo :: update_on_nearest ()
{
    // ...
    find_nearest(bar_, qux_.cbegin(), qux_.cend()); // Discarded :(
    // ...
}
```

Usando [[nodiscard]] el compilador lanza un warning

```
[[nodiscard]]
int Foo :: find_nearest (
    const int candidate,
    const std::vector<int>::const_iterator begin,
    const std::vector<int>::const_iterator end
{
    int nearest = candidate;
    // ...
    return nearest;
}

void Foo :: update_on_nearest ()
{
    // ...

    // warning: ignoring return value of function
    // declared with 'nodiscard' attribute
    find_nearest(bar_, qux_.cbegin(), qux_.cend());
    // ...
}
```

6: [[fallthrough]] (C++17)

Kate Gregory

What Do We Mean When We Say Nothing At All?

[[fallthrough]]

switch con casos sin break

```
switch (menu)
{
    case Menu::Inactive:
        MenuSelection(key);
    break;

    case Menu::AnimationSelection:
        AnimationSelection(key);

    case Menu::MovementSelection:
        MovementSelection(key);

    case Menu::ObjectSelection:
        ObjectSelection(key);
    break;

    case Menu::CameraSelection:
        CameraSelection(key);
    break;
};
```

[[fallthrough]]

Usando [[fallthrough]] comunicamos nuestras intenciones

```
switch (menu)
{
    case Menu::Inactive:
        MenuSelection(key);
    break;

    case Menu::AnimationSelection:
        AnimationSelection(key);
        [[fallthrough]];

    case Menu::MovementSelection:
        MovementSelection(key);
        [[fallthrough]];

    case Menu::ObjectSelection:
        ObjectSelection(key);
    break;

    case Menu::CameraSelection:
        CameraSelection(key);
    break;
};
```

7: Categorías de valores (C++11)

Ben Saks

Back to Basics: Understanding Value Categories

Klaus Iglberger

Back to Basics: Move Semantics (part 1 of 2)

Categorías de valores

Lvalue: Un objeto que ocupa un espacio en memoria.
Rvalue: Un objeto que no ocupa espacio en memoria.

Uso clásico de ambas categorías en C

```
int x, y;

// Lvalue = Rvalue
x = 1;
x = x + 1;

// Lvalue = Lvalue
y = x;
```

Categorías de valores

Lvalue: Un objeto que ocupa un espacio en memoria.
Rvalue: Un objeto que no ocupa espacio en memoria.

Uso clásico de ambas categorías en C

```
int x, y;

// Lvalue = Rvalue
x = 1;
x = x + 1;

// Lvalue = Lvalue
y = x;
```

Copiar objetos es caro: Usamos referencias de lvalues

```
#include <iostream>
#include <string>

void print (const std::string & str)
{
    std::cout << str << "\n";
}

// This is an example, please avoid output params (F.20)
void duplicate (std::string & str)
{
    str += (" " + str);
}

int main ()
{
    std::string str = "Hello";
    duplicate(str);
    print(str) // "Hello Hello"
}
```

Devolver objetos locales por referencia crea errores

```
#include <iostream>
#include <string>

std::string & generate_string_from_file (const std::string_view & path)
{
    std::string file_contents;
    // Populate the string...
    return file_contents;
}

int main (int argc, char ** argv)
{
    std::string file_string = generate_string_from_file(argv[1]);
}
```

¡Pero el objeto es caro de copiar!
(Ignorando RVO)

Podemos mover el objeto

```
#include <iostream>
#include <string>

std::string generate_string_from_file (const std::string_view & path)
{
   std::string file_contents;
   // Populate the string...
   return std::move(file_contents);
}

int main (int argc, char ** argv)
{
   std::string file_string = generate_string_from_file(argv[1]);
}
```

Mover un objeto implica que no podemos utilizarlo más en el ámbito del que se ha movido sin inicializar de nuevo su contenido.

¡Pero el objeto es caro de copiar!
(Ignorando RVO)

Podemos mover el objeto

```
#include <iostream>
#include <string>

std::string generate_string_from_file (const std::string_view & path)
{
    std::string file_contents;
    // Populate the string...
    return std::move(file_contents);
}

int main (int argc, char ** argv)
{
    std::string file_string = generate_string_from_file(argv[1]);
}
```

Mover un objeto implica que no podemos utilizarlo más en el ámbito del que se ha movido sin inicializar de nuevo su contenido.

Categorías de valores

Tenemos que introducir un nuevo concepto: **Referencias rvalue**.
`std::move` devuelve una referencia rvalue.

Pero... `file_contents` sigue ocupando espacio antes del `move`.
Dividimos rvalue en dos categorías:

- **Prvalue:** Rvalues puros, los que describimos en C.
- **Xvalue:** Objetos que están a punto de finalizar su tiempo de vida (expirando).

¡Pero los xvalues también pueden ser lvalues!
Introducimos la categoría **glvalue**: Valores lvalue globales. Pueden ser lvalues o xvalues.

Categorías de valores

Tenemos que introducir un nuevo concepto: **Referencias rvalue**.
`std::move` devuelve una referencia rvalue.

Pero... `file_contents` sigue ocupando espacio antes del `move`.
Dividimos rvalue en dos categorías:

- **Prvalue:** Rvalues puros, los que describimos en C.
- **Xvalue:** Objetos que están a punto de finalizar su tiempo de vida (expirando).

¡Pero los xvalues también pueden ser lvalues!
Introducimos la categoría **glvalue**: Valores lvalue globales. Pueden ser lvalues o xvalues.

Categorías de valores

Tenemos que introducir un nuevo concepto: **Referencias rvalue**.
`std::move` devuelve una referencia rvalue.

Pero... `file_contents` sigue ocupando espacio antes del `move`.
Dividimos rvalue en dos categorías:

- **Prvalue:** Rvalues puros, los que describimos en C.
- **Xvalue:** Objetos que están a punto de finalizar su tiempo de vida (expirando).

¡Pero los xvalues también pueden ser lvalues!
Introducimos la categoría **glvalue**: Valores lvalue globales. Pueden ser lvalues o xvalues.

Categorías de valores

Tenemos que introducir un nuevo concepto: **Referencias rvalue**.
`std::move` devuelve una referencia rvalue.

Pero... `file_contents` sigue ocupando espacio antes del `move`.
Dividimos rvalue en dos categorías:

- **Prvalue:** Rvalues puros, los que describimos en C.
- **Xvalue:** Objetos que están a punto de finalizar su tiempo de vida (expirando).

¡Pero los xvalues también pueden ser lvalues!
Introducimos la categoría **glvalue**: Valores lvalue globales. Pueden ser lvalues o xvalues.

Categorías de valores

Tenemos que introducir un nuevo concepto: **Referencias rvalue**.
`std::move` devuelve una referencia rvalue.

Pero... `file_contents` sigue ocupando espacio antes del `move`.
Dividimos rvalue en dos categorías:

- **Prvalue:** Rvalues puros, los que describimos en C.
- **Xvalue:** Objetos que están a punto de finalizar su tiempo de vida (expirando).

¡Pero los xvalues también pueden ser lvalues!
Introducimos la categoría **glvalue**: Valores lvalue globales. Pueden ser lvalues o xvalues.

Categorías de valores

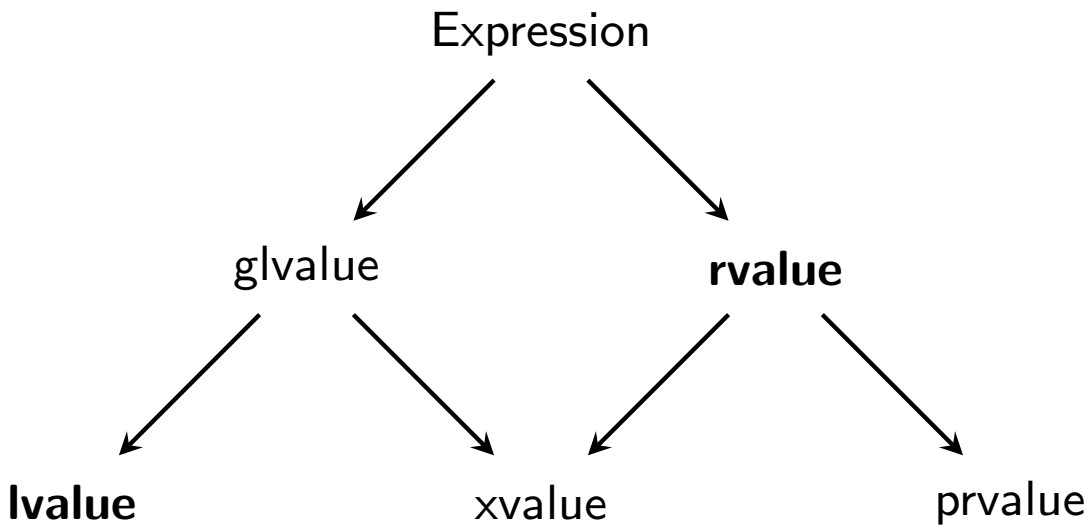
Tenemos que introducir un nuevo concepto: **Referencias rvalue**.
`std::move` devuelve una referencia rvalue.

Pero... `file_contents` sigue ocupando espacio antes del `move`.
Dividimos rvalue en dos categorías:

- **Prvalue:** Rvalues puros, los que describimos en C.
- **Xvalue:** Objetos que están a punto de finalizar su tiempo de vida (expirando).

¡Pero los xvalues también pueden ser lvalues!
Introducimos la categoría **glvalue**: Valores lvalue globales. Pueden ser lvalues o xvalues.

Categorías de valores



Programming Languages — C++, ISO

Una vez movido un objeto no podemos asegurar su estado

```
#include <iostream>
#include <vector>

int main ()
{
    std::vector<int> v1 {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};

    // OK
    for (const auto item : v1)
        std::cout << item << " ";

    std::vector<int> v2 = std::move(v1);

    // OK
    for (const auto item : v2)
        std::cout << item << " ";

    // UB
    std::cout << *v1.cbegin() << "\n";
}
```

8: Punteros inteligentes (C++14)

Atrhur O'Dwyer

Back to Basics: Smart Pointers

Error garrafal: Usar punteros brutos para poseer un recurso

```
template <typename T>
class Vector
{
private:
    T * begin_ = nullptr;
    T * end_   = nullptr;
    T * cap_   = nullptr;

public:
    Vector (const size_t capacity)
    {
        begin_ = new T[capacity];
        end_   = begin_ + 1;
        cap_   = begin_ + capacity + 1;
    }

    ~Vector ()
    {
        delete [] begin_;
    }
};
```

Usando unique_ptr indicamos la posesión del recurso

```
#include <memory>

template <typename T>
class Vector
{
private:
    std::unique_ptr<T[]> begin_ = nullptr;
    T * end_ = nullptr;
    T * cap_ = nullptr;

public:
    Vector (const size_t capacity)
    {
        begin_ = std::make_unique<T[]>(capacity);
        end_   = begin_.get() + 1;
        cap_   = begin_.get() + capacity + 1;
    }
};
```

Los punteros inteligentes liberan el recurso automáticamente al destruirse.

Ahora usamos punteros brutos sin miedo a transferir la propiedad del recurso

```
#include <memory>
#include <iostream>

void print_int     (const int i)       { std::cout << i      << "\n"; }
void print_int_ptr (const int * iptr) { std::cout << *iptr << "\n"; }

void adquire_then_print_int_ptr (std::unique_ptr<int> && iptr)
{
   std::cout << *iptr << "\n";
}

int main ()
{
   std::unique_ptr<int> my_int_ptr = std::make_unique<int>(3);

   print_int(*my_int_ptr);
   print_int_ptr(my_int_ptr.get());
   adquire_then_print_int_ptr(std::move(my_int_ptr));

   std::cout << *my_int_ptr << "\n"; // UB
}
```

9: Conceptos (C++20)

En Haskell indicamos que a debe pertenecer a la clase Show (ser imprimible)

```haskell
printDebug :: Show a => a -> String
printDebug a = "[DEBUG] " ++ show a
```

En C++ implementamos el concepto Showable

```cpp
template <typename T>
concept Showable = requires (T obj)
{
    std::cout << obj;
};

template <Showable T>
void print_debug (T obj)
{
   std::cout << "[DEBUG] " << obj << "\n";
}
```

En Haskell indicamos que a debe pertenecer a la clase Show (ser imprimible)

```haskell
printDebug :: Show a => a -> String
printDebug a = "[DEBUG] " ++ show a
```

En C++ implementamos el concepto Showable

```cpp
template <typename T>
concept Showable = requires (T obj)
{
    std::cout << obj;
};

template <Showable T>
void print_debug (T obj)
{
   std::cout << "[DEBUG] " << obj << "\n";
}
```

No podemos llamar a print_debug si no podemos imprimir el objeto

```
struct Square
{
    int lower_left_ = 0;
    int length_ = 0;
};

int main ()
{
    Square square;

    print_debug(3);
    print_debug("Hello!");
    print_debug(square); // No matching function for call to 'print_debug'
}
```

¿Qué nos grita gcc?

```
a.cpp: In function 'int main ()':
a.cpp:27:20: error: no matching function for call to 'print_debug(Square&)'
   27 |         print_debug(square);
      |         ~~~~~~~~~~~~~~~~~~~
a.cpp:10:6: note: candidate: 'template<class T>  requires  Showable<T>
           void print_debug(const T&)'
   10 | void print_debug (const T  & obj)
      |      ^~~~~~~~~~~
a.cpp:10:6: note:   template argument deduction/substitution failed:
a.cpp:10:6: note: constraints not satisfied
a.cpp: In substitution of 'template<class T>  requires  Showable<T>
       void print_debug(const T&) [with T = Square]':
a.cpp:27:13:   required from here
a.cpp:4:9:   required for the satisfaction of 'Showable<T>' [with T = Square]
a.cpp:4:20:   in requirements with 'T obj' [with T = Square]
a.cpp:6:20: note: the required expression '(std::cout << obj)' is invalid
    6 |         std::cout << obj;
      |         ~~~~~~~~~~~~~~~~
```

10: Lambdas (C++11)

Atrhur O'Dwyer

Back to Basics: Lambdas from Scratch

Clase para llamar a una función

```
class Plus
{
private:
    int num;

public:
    Plus (int number) : num (number) { }
    int plus_number (const int x) { return x + num; };
};

int main ()
{
    Plus plus(1);
    static_assert(plus.plus_number(21) == 22);
}
```

La llamada a la función es una sobrecarga de operator ()

```
class Plus
{
private:
    int num;

public:
    Plus (int number) : num (number) { }
    int operator () (const int x) { return x + num; };
};

int main ()
{
    Plus plus(1);
    static_assert(plus(21) == 22);
}
```

Lambdas

¿De verdad tengo que escribir todo esto para sumar 1?

```
class Plus
{
private:
    int num;

public:
    Plus (int number) : num (number) { }
    int operator () (const int x) { return x + num; };
};
```

Misma expresión con una lambda

```
int main ()
{
    auto plus = [num = 1] (int x) { return x + num; };
    static_assert(plus(21) == 22);
}
```

¿De verdad tengo que escribir todo esto para sumar 1?

```
class Plus
{
private:
    int num;

public:
    Plus (int number) : num (number) { }
    int operator () (const int x) { return x + num; };
};
```

Misma expresión con una lambda

```
int main ()
{
    auto plus = [num = 1] (int x) { return x + num; };
    static_assert(plus(21) == 22);
}
```

Lambdas en acción

```cpp
#include <iostream>
#include <regex>
#include <string>
#include <vector>

int main ()
{
    std::vector<std::string> films {
        "1421 - Starship Troopers",   "4920 - Whiplash",
        "3912 - Catch Me If You Can", "9857 - Ferrus Bueller's Day Off",
    };

    auto title_regex = [] (const std::string & title)
    {
        return std::regex_match(title, std::regex("[0-9]{4} - .+"));
    }

    bool all_titles_correct = std::all_of(
        films.cbegin(), films.cend(), title_regex
    );

    return static_cast<int>(all_titles_correct);
}
```

El mismo ejemplo pero *inline*

```
#include <iostream>
#include <regex>
#include <string>
#include <vector>

int main ()
{
    std::vector<std::string> films {
        "1421 - Starship Troopers",   "4920 - Whiplash",
        "3912 - Catch Me If You Can", "9857 - Ferris Bueller's Day Off",
    };

    bool all_titles_correct = std::all_of(
        films.cbegin(), films.cend(),
        [] (const std::string & title)
        {
            return std::regex_match(title, std::regex("[0-9]{4} - .+"));
        }
    );

    return static_cast<int>(all_titles_correct);
}
```

Usar bibliotecas no RAII nunca es fiable

```
#include <3rdparty/non-raii-file-handler/fhandler.hpp>

void print_file (const std::string_view & path)
{
    FHandler handler(path);
    handler.open();
    // Print every line
    handler.close();
}

void print_vowels (const std::string_view & path)
{
    FHandler handler(path);
    handler.open();
    // Print all vowels one by one
}

int main (int argc, char ** argv)
{
    count_file_lines(argv[1]);
    print_file(argv[1]);
}
```

Pasamos una lambda para asegurarnos de que siempre se cierra el fichero

```
#include <3rdparty/non-raii-file-handler/fhandler.hpp>

template <typename Function>
void run_on_file (const std::string_view & path, Function fn)
{
    FHandler handler(path);
    handler.open();
    fn(handler);
    handler.close();
}

int main (int argc, char ** argv)
{
    auto print_file   = [] (const FHandler & f) { /* Print file */ };
    auto print_vowels = [] (const FHandler & f) { /* Print all vowels */ };

    run_on_file(argv[1], print_file);
    run_on_file(argv[1], print_vowels);
}
```

Clean Code - Robert Martin

11: Ronda relámpago (C++*)

Cabeceras siempre sin espacios de nombres

¡¡¡No uséis `namespace std` en las cabeceras!!!

`vector.hpp`: Empieza a usar `std`

```
using namespace std;

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

`main.cpp`: ¿A qué `vector` nos estamos refiriendo?

```
#include <vector>
#include "parser.hpp" // Includes "vector.hpp" somewhere deep

int main ()
{
   vector vec(5);
}
```

¡¡¡No uséis `namespace std` en las cabeceras!!!

`vector.hpp`: Empieza a usar `std`

```
using namespace std;

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

`main.cpp`: ¿A qué `vector` nos estamos refiriendo?

```
#include <vector>
#include "parser.hpp" // Includes "vector.hpp" somewhere deep

int main ()
{
   vector vec(5);
}
```

¡¡¡No uséis `namespace std` en las cabeceras!!!

`vector.hpp`: Empieza a usar `std`

```
using namespace std;

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

`main.cpp`: ¿A qué `vector` nos estamos refiriendo?

```
#include <vector>
#include "parser.hpp" // Includes "vector.hpp" somewhere deep

int main ()
{
   vector vec(5);
}
```

Warnings de gcc que recomiendo

Declaraciones de banderas y *warnings* en mis `Makefiles`

```
WBASIC := -Wall -Wconversion -Wextra -Wpedantic
WCAST  := -Wnoexcept -Wuseless-cast
WMATHS := -Wfloat-equal -Wsign-promo
WOPTIM := -Wdisabled-optimization -Winline
WPALEO := -Wold-style-cast -Wzero-as-null-pointer-constant
WFLAGS := $(WBASIC) $(WCAST) $(WMATHS) $(WOPTIM) $(WPALEO)

# -Wpadded

ifdef NDEBUG
   CXXFLAGS := $(CXXFLAGS) -DNDEBUG -O2
else
   CXXFLAGS := $(CXXFLAGS) $(WFLAGS) -g
endif
```

ISOC++ Core Guidelines

https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines.html

- **C.4:** Make a function a member only if it needs direct access to the representation of a class.
- **Enum.6:** Avoid unnamed enumerations.
- **ES.41:** If in doubt about operator precedence, parenthesize.
- **ES.47:** Use `nullptr` rather than `0` or `NULL`.
- **ES.49:** If you must use a cast, use a named cast.
- **ES.106:** Don't try to avoid negative values by using `unsigned`.
- **F.8:** Prefer pure functions.
- **I.11:** Never transfer ownership by a raw pointer (`T*`) or reference (`T&`).
- **P.3:** Express intent.
- **P.5:** Prefer compile-time checking to run-time checking.
- **Per.7:** Design to enable optimization.
- **R.3:** A raw pointer (a `T*`) is non-owning.
- **R.11:** Avoid calling `new` and `delete` explicitly.

Compiler Explorer

https://godbolt.org/

Orden de calificativos

Arthur O'Dwyer:
`static constexpr unsigned long` is C++'s "lovely little old French whittling knife"

https://quuxplusone.github.io/blog/2021/04/03/static-constexpr-whittling-knife/

¡Gracias por vuestra atención!

https://twitter.com/Groctel
https://gitlab.com/Groctel

https://gitlab.com/Groctel/modern-cpp-workshop

CppCon Likes:
https://www.youtube.com/playlist?list=PLnRwtuFx1xpeFeEMDIDLkOron0Fs8tHDj