STL Algorithms

Reid Watson
(rawatson@stanford.edu)

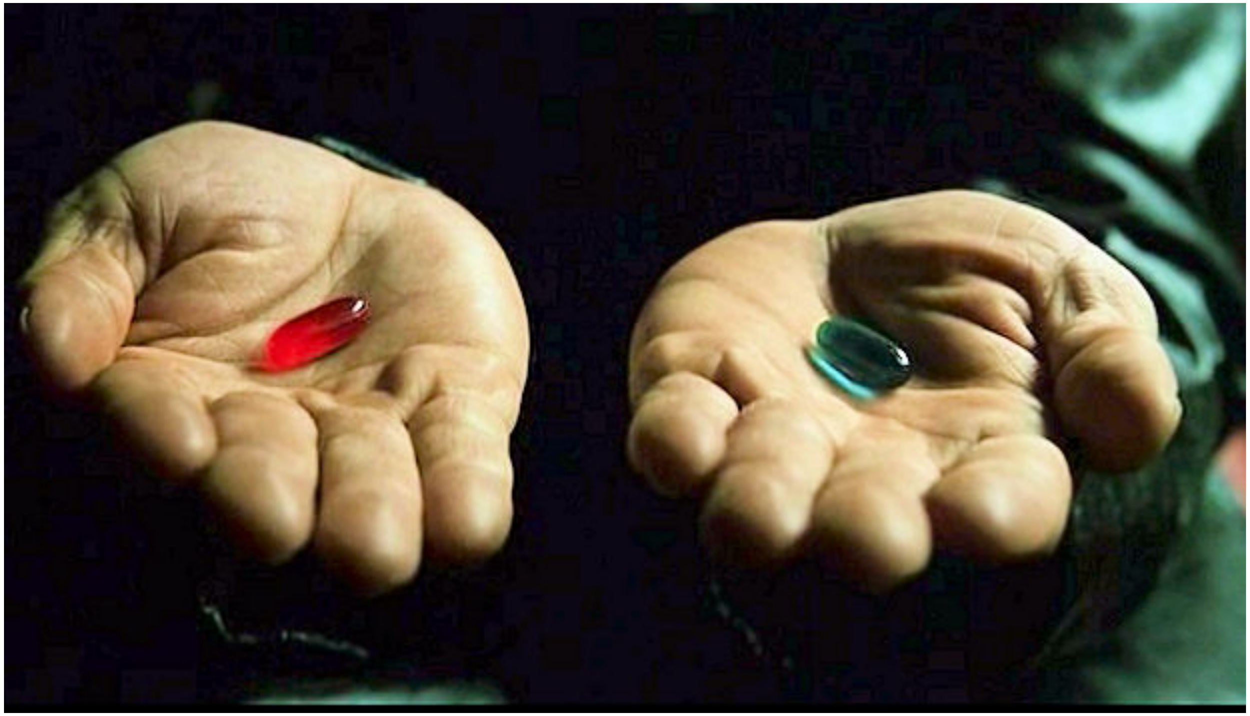
Administrivia

- Office hours: Tuesday 6-12, Thursday 8-12
- Assignment one is still out
- Qt Creator

STL Algorithms

- Iterators and why they exist
- STL <algorithm>
- Iterator adapters

STL Iterators



STL Iterators

The Blue Pill:

Iterators are an inconvenient way to access all the elements in a set

The Blue Pill:

Iterators are the building block for accessing, modifying, and using all sorts of collections

STL Iterators

- Last class we talked about iterators
- Iterators allowed us to access all the elements in a container in a linear order, even if the container was unordered
- This was convenient, but the syntax seemed a bit awkward, especially looking at Stanford foreach or C++11 range based for

STL Iterators

Printing the contents of a vector of integers (say the vector was named 'v') using iterators.

```
vector<int>::iterator i = v.begin();
while (i != v.end()) {
  cout << *i << endl;
  ++i;
}
```

STL Iterators

Printing the contents of a vector of integers (say the vector was named 'v') using C++11 range based for.

```
for (int x : v) {
  cout << x << endl;
}
```

STL Iterators

Why do we still use iterators if the syntax is so awkward?

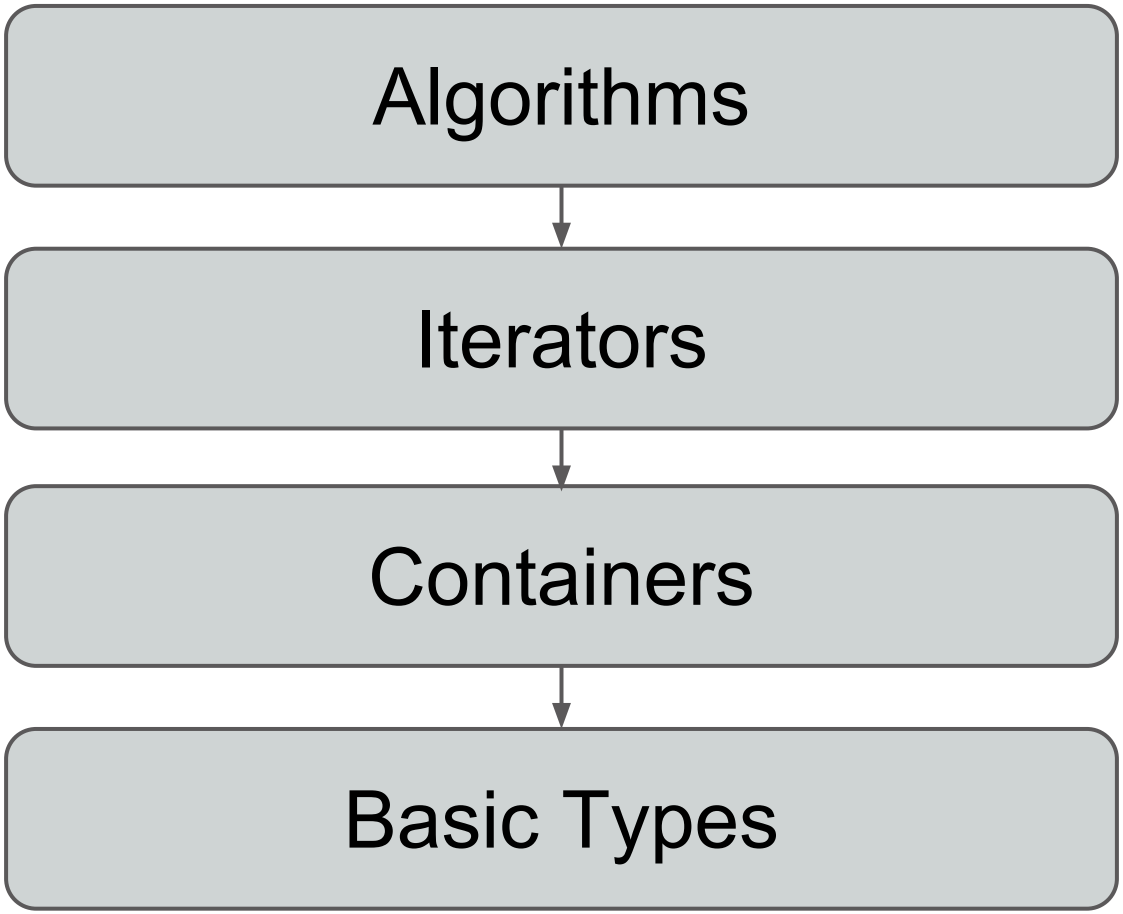
STL Iterators

Let's introduce a different picture of the STL, in terms of **abstraction**

Abstraction in the STL

abstraction allows us to express the general structure of a problem instead of the particulars of implementation

Abstraction in the STL



Abstraction in the STL

- We began by talking about **basic types**.
- **char**, **int**, **double**, **string**, others.
- Each of these types held what was conceptually a "single value"

Abstraction in the STL

- Many programs require a number of variables of the same basic type
  - A vector of integers representing student's ages
  - A mapping translating between names and addresses
- **Containers** allow a programmer to use the same collection regardless of the underlying basic type
- The same `<vector>` implementation can be used for `int`s as well as `string`s

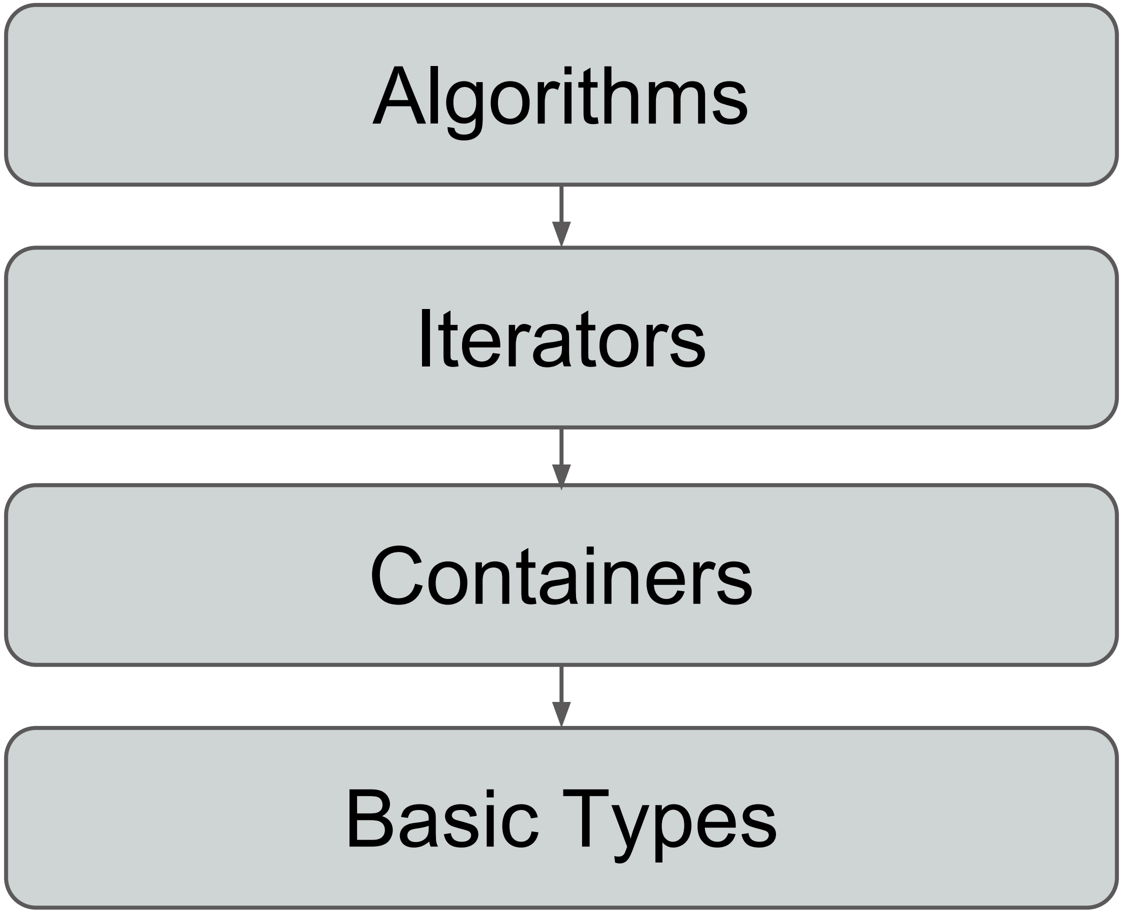
Abstraction in the STL

- The same `<vector>` implementation can be used for `int`s as well as `string`s
- This means we can use containers to perform various operations on **basic types**, regardless of what the basic type is?
- Is it possible to perform various operations on **containers** regardless of what the container is?

Abstraction in the STL

- Iterators allow us to abstract away which container was used
  - Similar to how containers allow us to abstract away which basic type was used
- Operations like sorting, partitioning, filtering, searching, etc., can be written to work with a **vector**, **deque**, **set**, or any other data type.
- We call these operations the STL **algorithms**

Abstraction in the STL



Examples of Algorithms

Let's take a look at why this is cool.

See AlgorithmFun.pro

Why Algorithm

Why do we need this complex model of abstraction?

- Don't **duplicate** code
- Write **correct** code
- Write **efficient** code
- Write **clear** code

Why Algorithm

To take a look at what's possible with <algorithm>, let's write a quick magic square solver.

Why Algorithm

A "magic square" is a 3x3 grid in which all rows, columns, and 3-element diagonals sum to the same number.

2	7	6
9	5	1
4	3	8

Why Algorithm

A "magic square" is a 3x3 grid in which all rows, columns, and 3-element diagonals sum to the same number.

2	7	6
9	5	1
4	3	8

Why Algorithm

A "magic square" is a 3x3 grid in which all rows, columns, and 3-element diagonals sum to the same number.

2	7	6
9	5	1
4	3	8

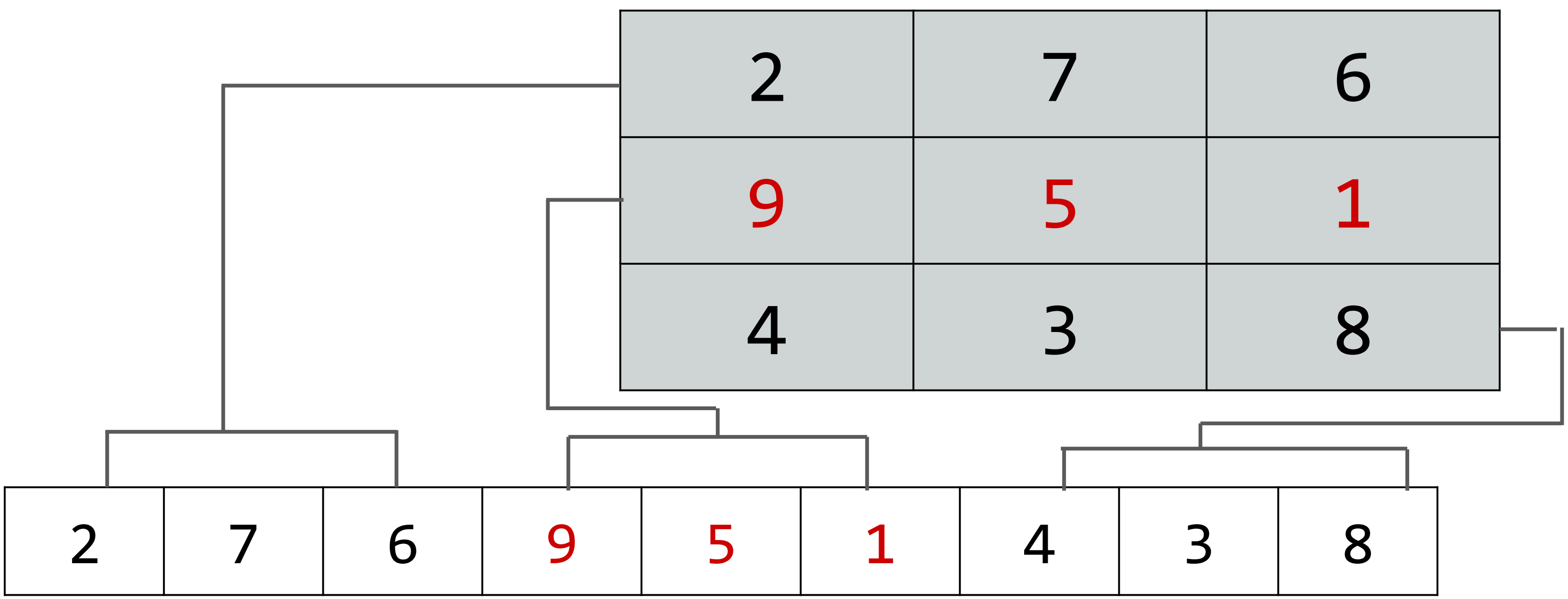
Why Algorithm

A "magic square" is a 3x3 grid in which all rows, columns, and 3-element diagonals sum to the same number.

2	7	6
9	5	1
4	3	8

Why Algorithm

We'll represent a magic square as a linear vector of elements



Why Algorithm

If we could enumerate through every permutation of the numbers 1-9 in a vector, we could find every magic square which uses only the numbers 1-9...

If only we had an `<algorithm>` to do that...

Why Algorithm

Let's take a look at some code to solve this in MagicSquares.pro

In-depth: std::copy

To understand iterators and algorithms a bit better, let's take a look at the copy function.

```
vector<int> v;
v.push_back(1);
v.push_back(650);
v.push_back(867);
v.push_back(5309);

vector<int> vcopy(4);

copy(v.begin(), v.end(), vcopy.begin());
```

In-depth: std::copy

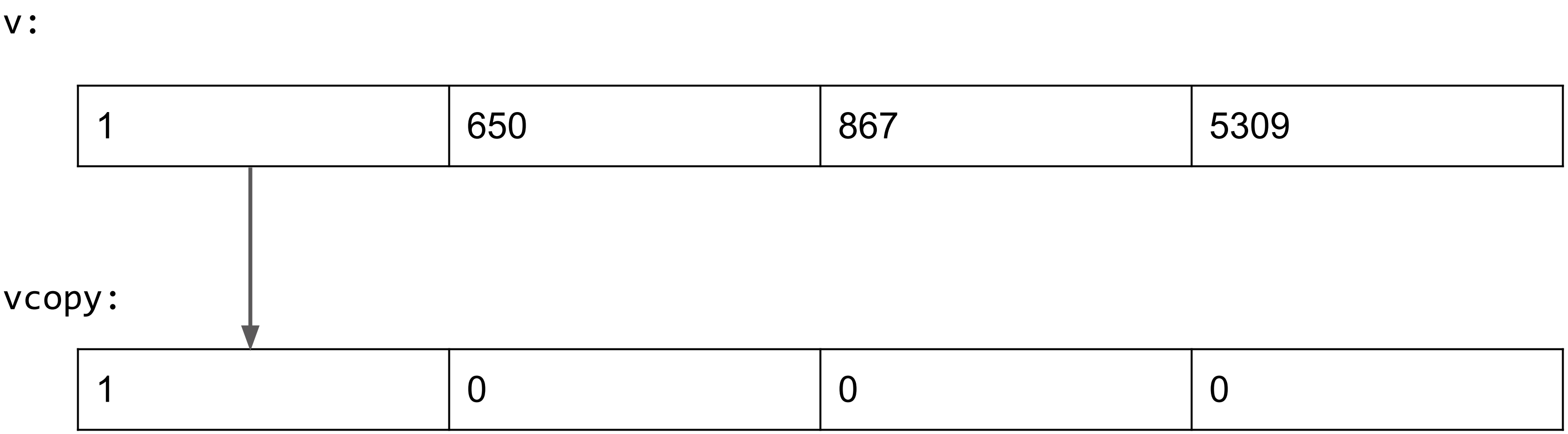
`v:`

1	650	867	5309

`vcopy:`

0	0	0	0

In-depth: std::copy



In-depth: std::copy

`v:`

1	650	867	5309

`vcopy:`

1	650	0	0

In-depth: std::copy

`v:`

1	650	867	5309

`vcopy:`

1	650	867	0

In-depth: std::copy

`v:`

1	650	867	5309

`vcopy:`

1	650	867	5309

In-depth std::copy

What happens if we didn't allocate enough space?

In-depth: std::copy

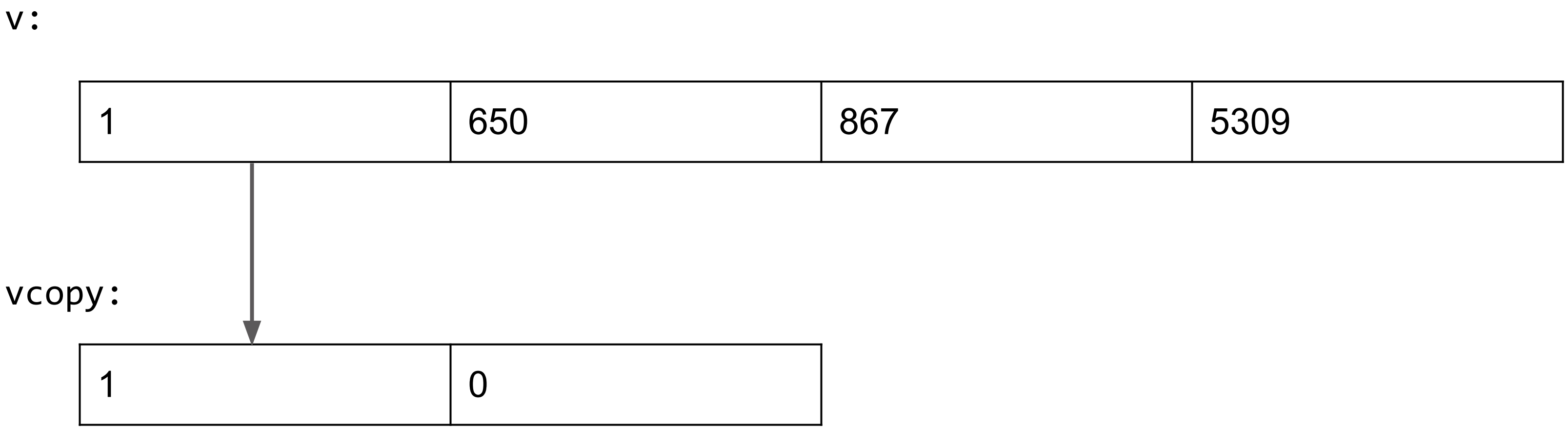
`v:`

1	650	867	5309

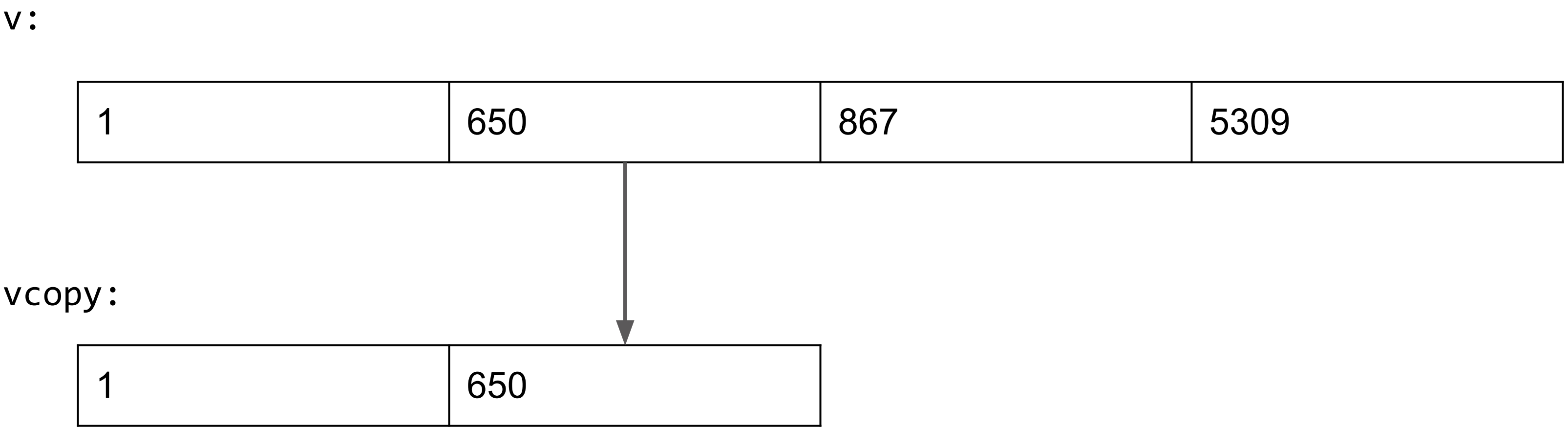
`vcopy:`

0	0

In-depth: std::copy



In-depth: std::copy



In-depth: std::copy

`v:`

1	650	867	5309

`vcopy:`

1	650

In-depth: std::copy

How can we avoid running into this problem?

Iterator Adapters

Sometimes we need to form "weird" iterators.

- We don't just want to iterate over elements, we want to retrieve them from an istream
- We don't just want to iterate over elements we want to add them to a vector

Iterator Adapters

Stream iterators are a fun way to simplify code. When you want to repeatedly read values from an input streams.

You can also form iterators which write values to a stream for you.

It's easiest to explain these with a quick bit of code demonstrating how they work.

Iterator Adapters

See code in Sum.pro

Iterator Adapters

Inserters create an iterator which inserts values into a container for you.

These are useful when using something like std::copy.

Iterator Adapters

Using a back inserter will push the elements to the end of vcopy, so you don't have to worry about vcopy not having enough space.

```
vector<int> v;
v.push_back(1);
v.push_back(650);
v.push_back(867);
v.push_back(5309);

vector<int> vcopy;

copy(v.begin(), v.end(), back_inserter(vcopy));
```

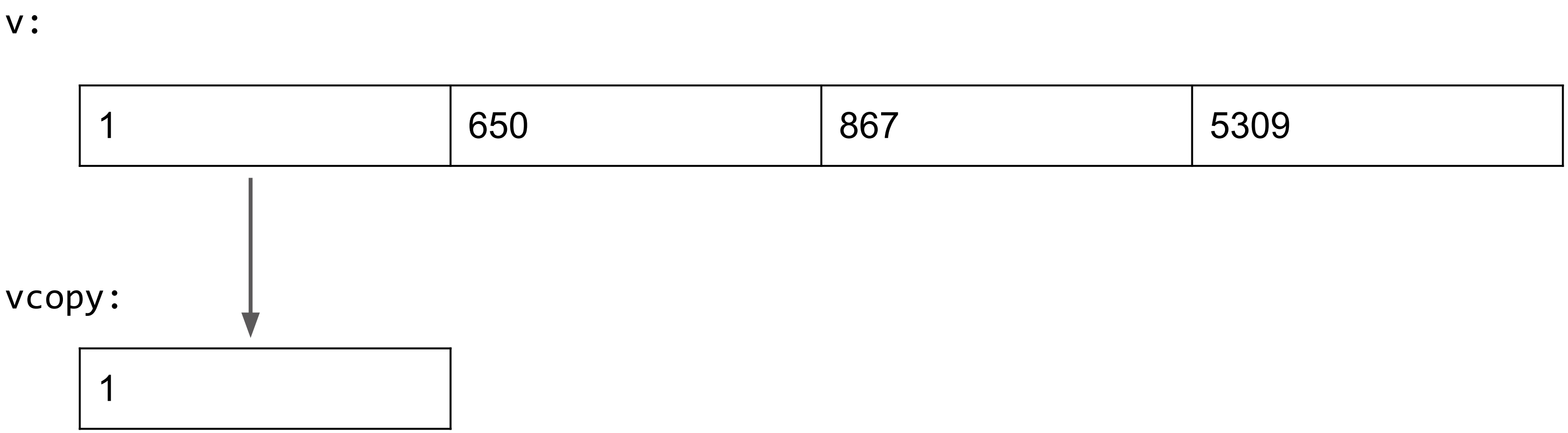
Iterator Adapters

`v:`

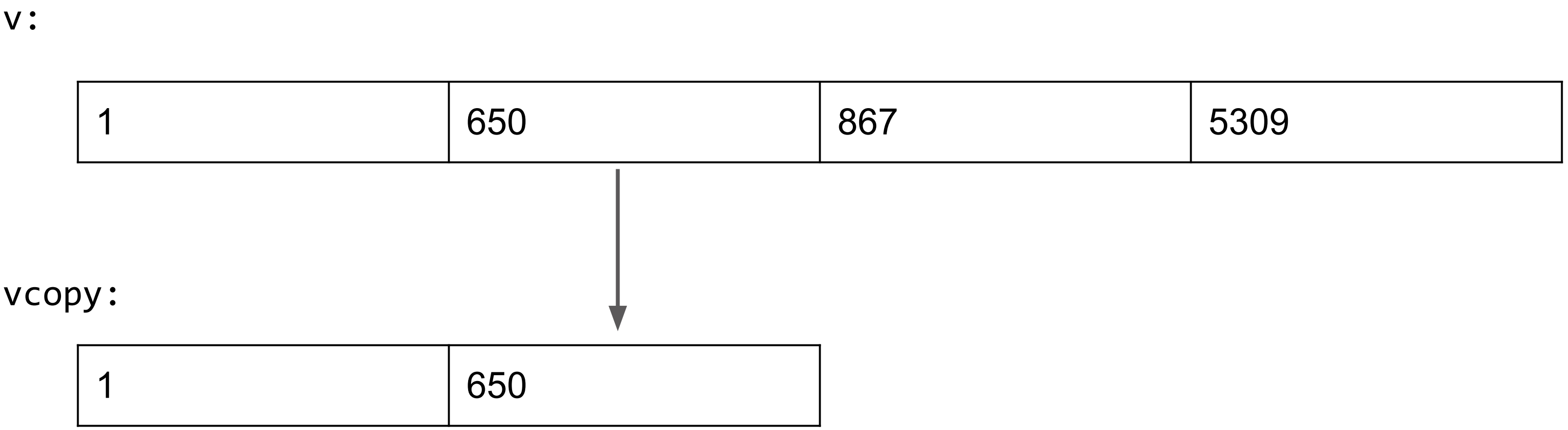
1	650	867	5309

`vcopy:`

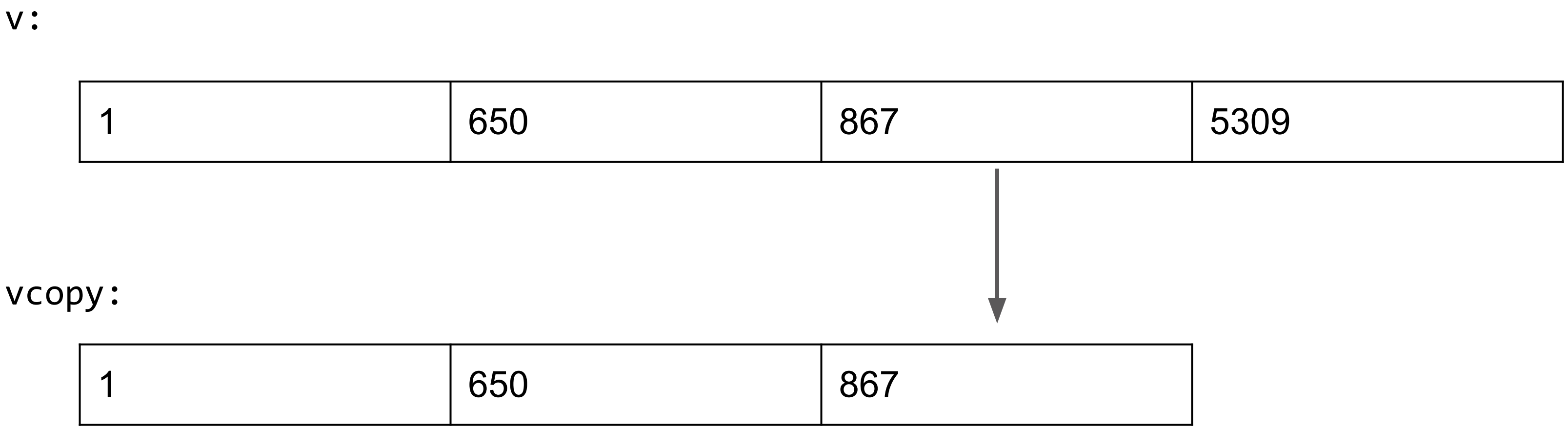
Iterator Adapters



Iterator Adapters



Iterator Adapters



Iterator Adapters

`v:`

1	650	867	5309

`vcopy:`

1	650	867	5309