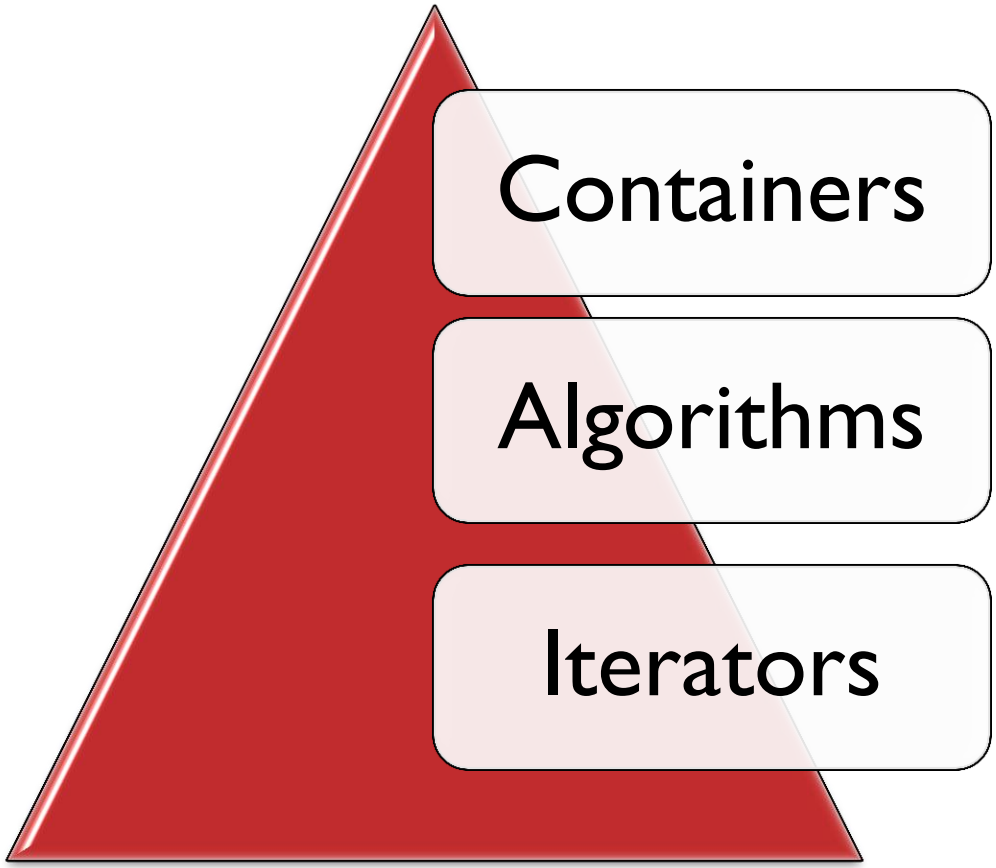
STANDARD TEMPLATE LIBRARY (STL)

Developed by Alexander Stepanov and Meng Lee of HP in 1979.

Standard template library accepted in July 1994 into C++ ANSI Standard

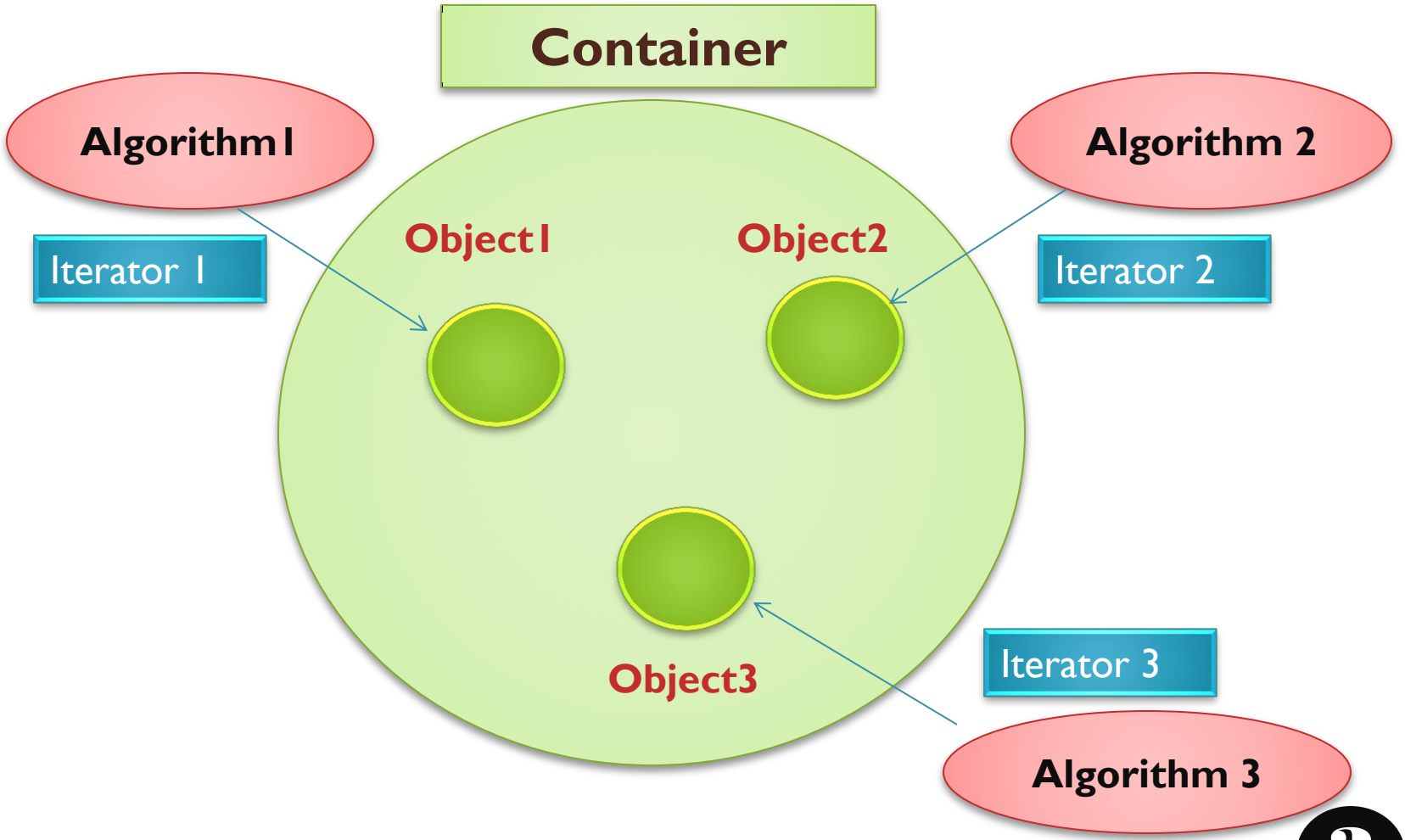
These are called as collection of General-purpose template classes(data structures) and functions

COMPONENTS OF STL



COMPONENTS OF STL

Algorithms use iterators to interact with objects stored in containers



CONTAINER

Objects that hold data (of same type)

Example : Array

Implemented by Template Classes

ALGORITHM

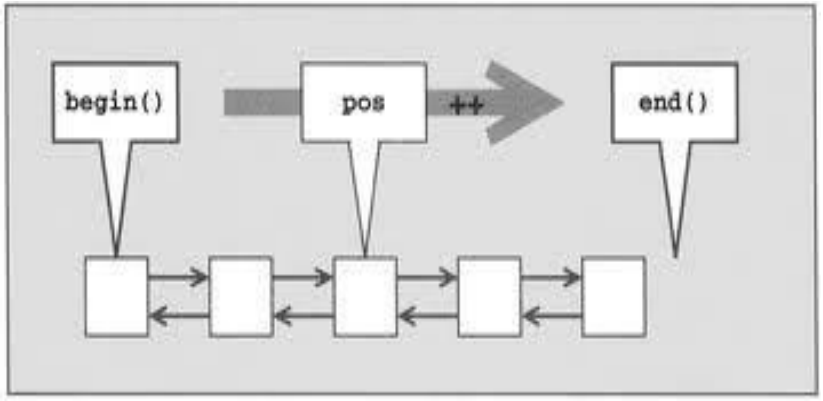
These are procedures used to process the data contained in containers.

Example :

Searching, Sorting, Merging, Copying, Initializing

Implemented by template functions

ITERATOR



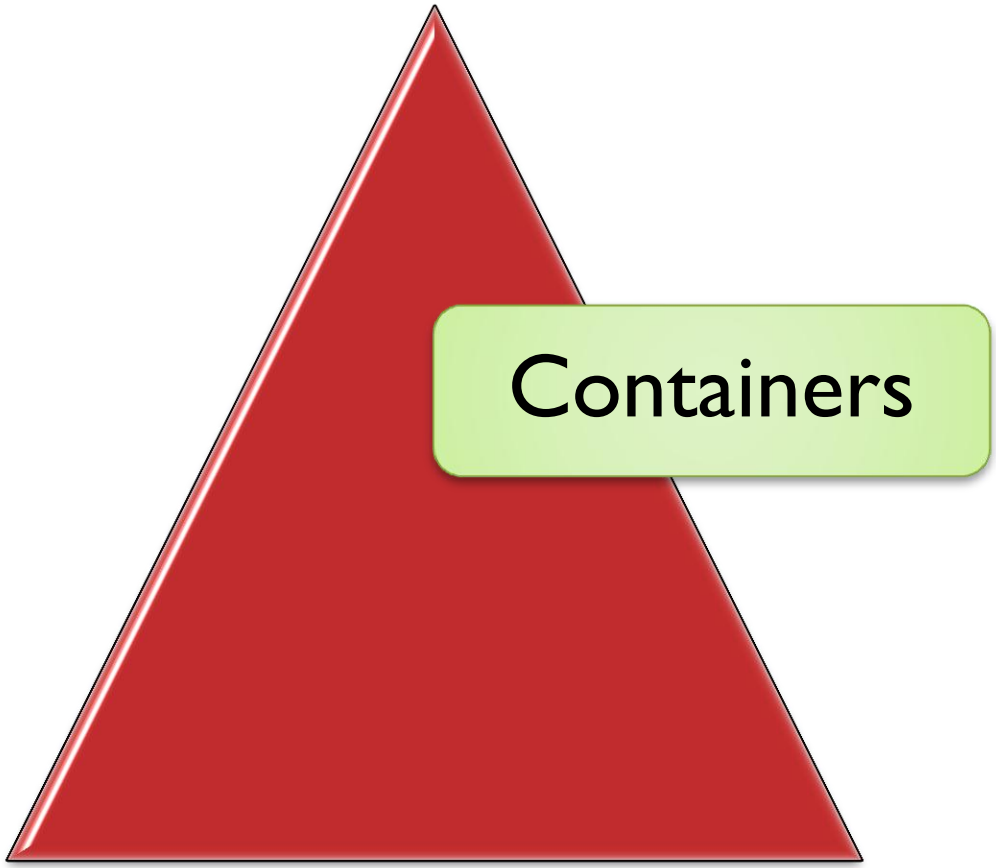
It is an object that points to an element in a container

Used to move through the contents of container

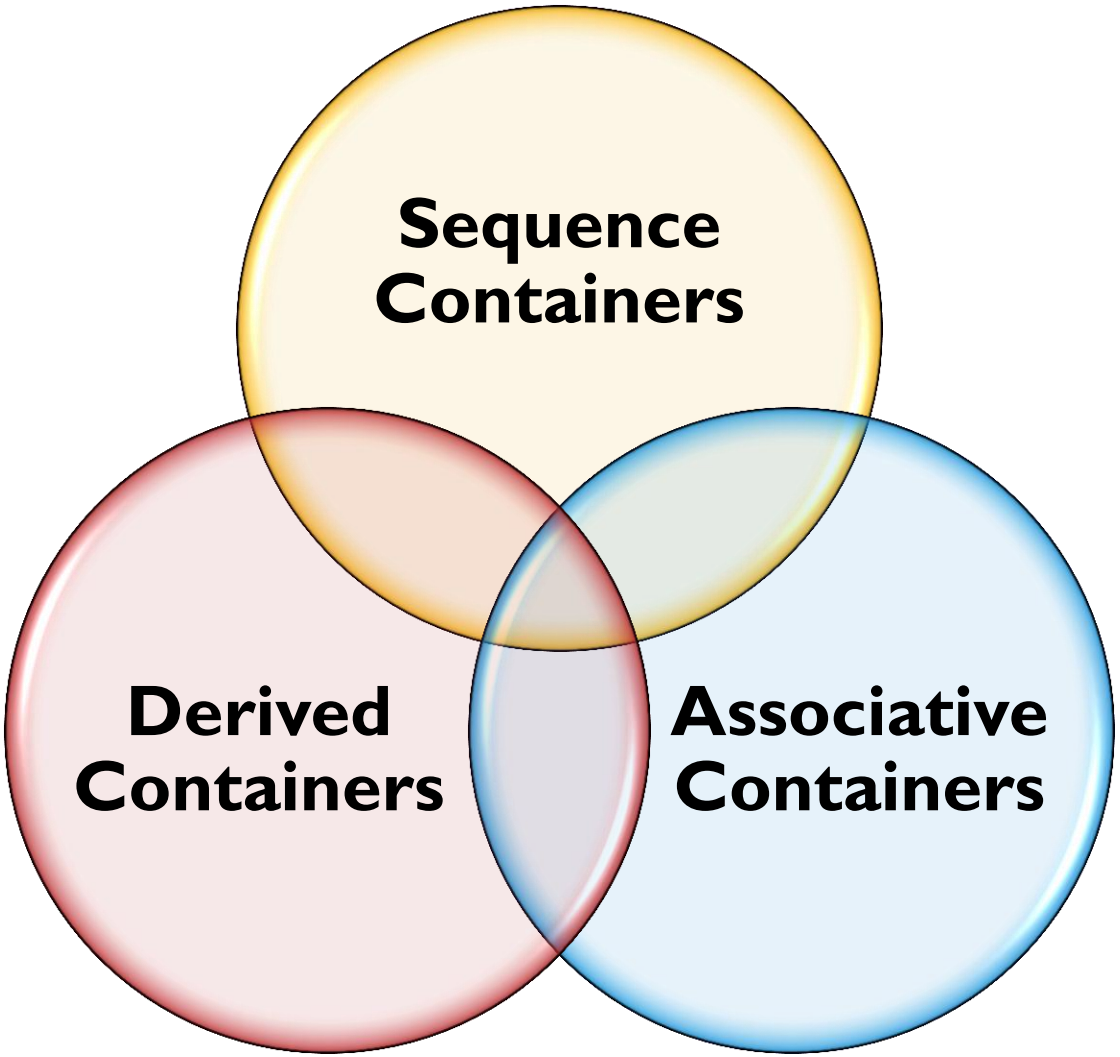
They can be incremented and decremented

Connect Algorithms with Containers

COMPONENTS OF STL



CATEGORIES OF CONTAINERS



CONTAINERS

STL Defines **10** Containers

CATEGORIES OF CONTAINERS

Sequence	Associative	Derived
• vector • deque • list	• set • multiset • map • multimap	• stack • queue • Priority_queue

SEQUENCE CONTAINERS

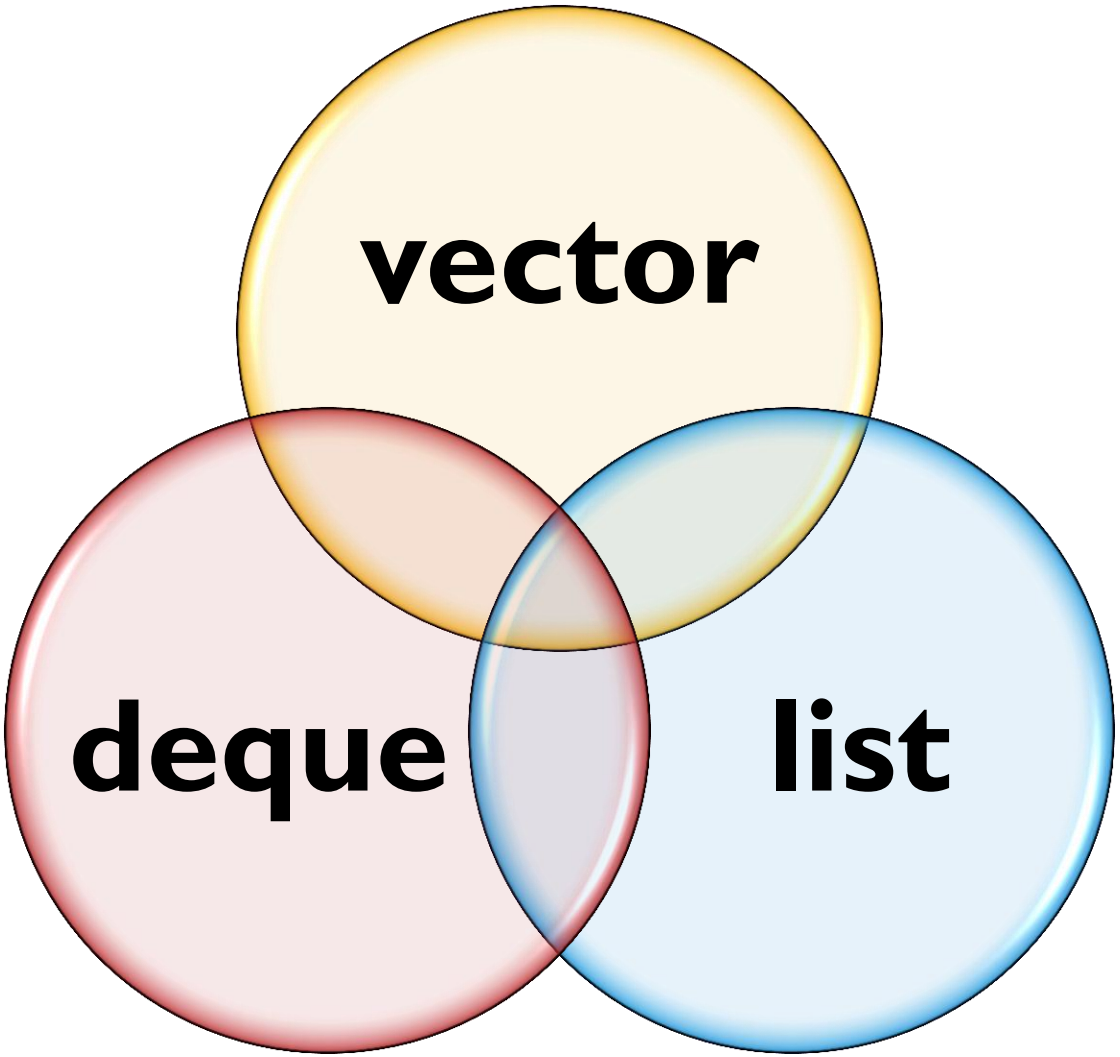
Stores elements in a linear sequence

Each element is related to other elements by its position along the line

They allow insertion of elements

Example

THREE TYPES OF SEQUENCE CONTAINERS



Vector : Sequence Container

- Expandable and dynamic array
- Grows and shrinks in size
- Insertion / Deletion of elements at back
- Permits direct access to any element

Vector : Sequence Container

Container	Header File	Iterator
vector	<vector>	Random Access

vector Sequence Container

- Declarations
  - **`vector <type> v;`**
    - ***`type`***: **`int`**, **`float`**, etc.
- Iterators
  - **`vector<type>::const_iterator iterVar;`**
    - **`const_iterator`** cannot modify elements
  - **`vector<type>::reverse_iterator iterVar;`**
    - Visits elements in reverse order (end to beginning)
    - Use **`rbegin`** to get starting point
    - Use **`rend`** to get ending point

vector Sequence Container

- **`vector`** functions
  - **`v.push_back(value)`**
    - Add element to end (found in all sequence containers).
  - **`v.size()`**
    - Current size of vector
  - **`v.capacity()`**
    - How much vector can hold before reallocating memory
    - Reallocation doubles size
  - **`vector<type> v(a, a + SIZE)`**
    - Creates **`vector v`** with elements from array **`a`** up to (not including) **`a + SIZE`**

vector Sequence Container

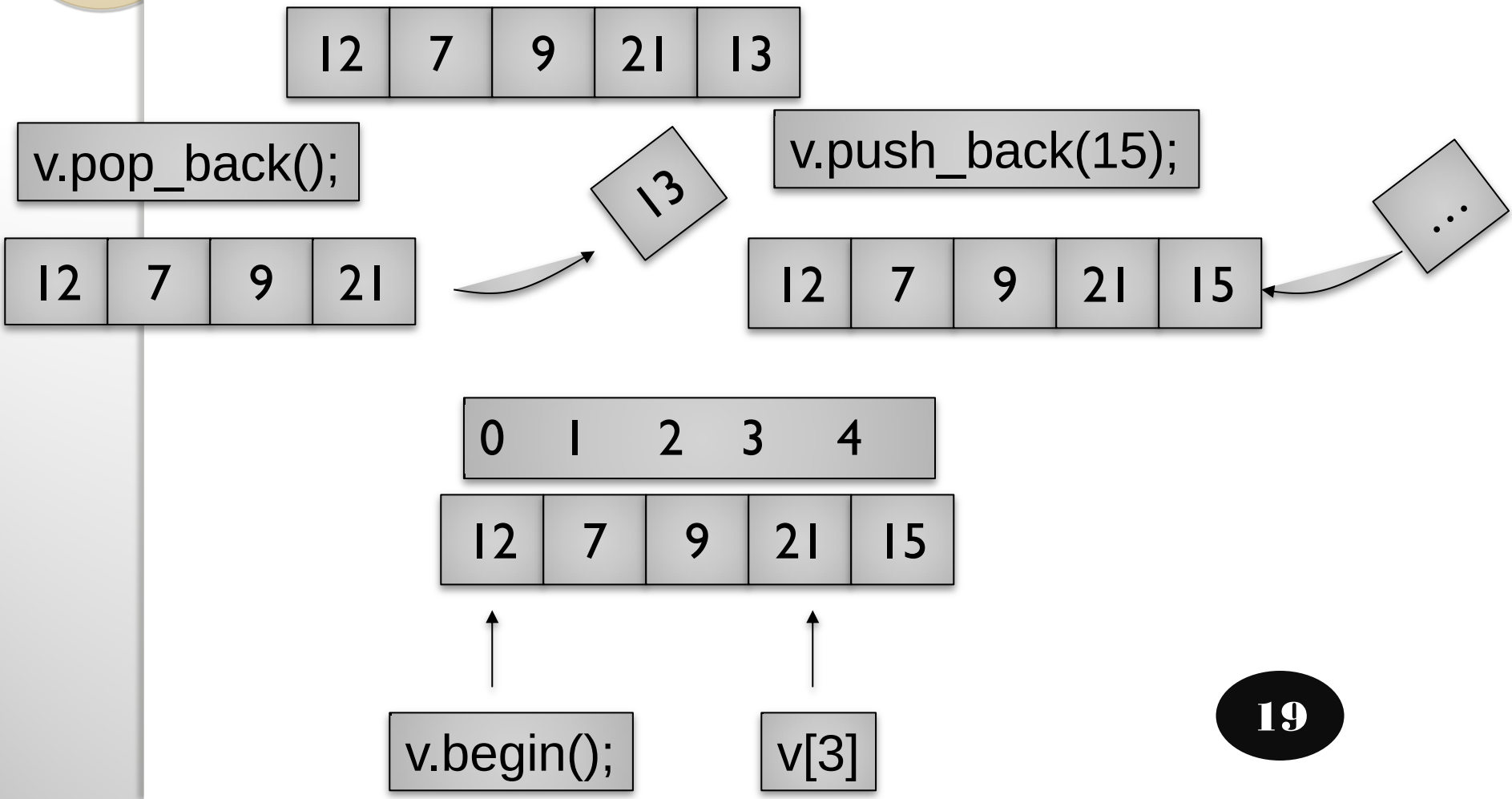
- **`vector` functions**
  - **`v.insert(`***iterator*`,  `***value*`)`**
    - Inserts *value* before location of *iterator*
  - **`v.insert(`***iterator*`,  `***array*`,  `***array*** + ***SIZE*`)`**
    - Inserts array elements (up to, but not including *array* + *SIZE)* into vector
  - **`v.erase( iterator )`**
    - Remove element from container
  - **`v.erase( iter1, iter2 )`**
    - Remove elements starting from **`iter1`** and up to (not including) **`iter2`**
  - **`v.clear()`**
    - Erases entire container

vector Sequence Container

- **`vector`** functions operations
  - **`v.front(), v.back()`**
    - Return first and last element
  - **`v.[elementNumber] = value;`**
    - Assign **`value`** to an element

Vector : Sequence Container

```
int array[5] = {12, 7, 9, 21, 13 };
vector<int> v(array,array+5);
```



Vector : Sequence Container

```
#include <vector>
#include <iostream>
using namespace std;
void main
{
int arr[] = {12, 7, 9, 21, 13 };  // standard C array
vector<int> v(arr, arr+5);  // initialize vector with C array

while ( ! v.empty()) // until vector is empty
{
   cout << v.back() << " ";   // output last element of vector
   v.pop_back();              // delete the last element
}
for(i=0; i<v.size(); ++i)
          cout<<v[i]<<' ';
 cout<<endl }
```

O/P of previous program

13 21 9 7 12

12 7 9 21 13

Vector : Using Iterator

```
#include <vector>
#include <iostream>
using namespace std;

int main()
{
    vector <int> vec1;
    vector <int>::iterator vec1_Iter;
    vector <int>::reverse_iterator vec1_rIter;

   vec1.push_back(10);
    vec1.push_back(7);
    vec1.push_back(3);
```

Vector : Using Iterator

```
cout<<"vec1 data: ";
for(int i=0; i<vec1.size(); ++i)
    cout<<vec1[i]<<' ';
cout<<endl;

cout<<"\nOperation: vec1.begin()\n";
vec1_Iter = vec1.begin();

cout<<"The first element of vec1 is "<<*vec1_Iter<<endl;

cout<<"\nOperation: vec1.rbegin()\n";
vec1_rIter = vec1.rbegin();
cout<<"The first element of the reversed vec1 is ";
cout<<*vec1_rIter<<endl;
return 0;
}
```

O/P of previous program

vec1 data: 10 7 3

Operation: vec1.begin()
The first element of vec1 is10

Operation: vec1.rbegin()
The first element of the reversed vec1 is : 3

Vector : Using Iterator

```
    cout<<"Operation: vec1.rbegin() and vec1.rend()\n";
    cout<<"vec1 data: ";

For(key = vec1.rbegin(); key != vec1.rend();
                                        key++)
   cout<<*key<<' ';
    cout<<endl;
    return 0;
}
```

O/P of previous program

Operation: vec1.begin() and vec1.rend()

vec1 data: 1 4 3 7

deque : Sequence Container

- Double ended Queue
- Insertion / Deletion of elements both ends
- Permits direct access to any element

deque : Sequence Container

Container	Header File	Iterator
deque	<deque>	Random Access

Deque

```
#include <iostream>
 #include <deque>
using namespace std;
int main ()
 {
   deque<int> mydeque;
  mydeque.push_back (100);
  mydeque.push_back (200);
  mydeque.push_back (300);

cout << "\nThe final size of mydeque is "
cout<<<< mydeque.size() << “\n”;
```

Deque

```
cout << "Popping out the elements in mydeque:";
   while (!mydeque.empty())
     {
          cout << mydeque.front();
            mydeque.pop_front();
     }


cout << "\nThe final size of mydeque is "
cout<<<< mydeque.size() << “\n”;

 return 0;

}
```

O/P of previous program

The final size of mydeque is : 3

Popping out the elements in mydeque:

100 200 300

The final size of mydeque is : 0

list : Sequence Container

Bidirectional

Insertion / Deletion of elements anywhwere

list : Sequence Container

Container	Header File	Iterator
list	<list>	Bidirectional

List

```
#include <iostream.h>
 #include <list>

void print(list <char> );

 main()
  {
        list <char> l;
        list <char>::iterator p;

          l.push_back('o');
          l.push_back('a');
          l.push_back('t');

          p=l.begin();
```

List

```
cout <<" "<< *p<<endl; // p refers to the 'o' in ('o', 'a', 't')
print(l);

l.insert(p, 'c'); // l is now ('c', 'o', 'a', 't') and p still refers to
'o‘
 cout <<" "<< *p<<endl;

print(l);

 l.erase(p);

cout <<" "<< *p<<endl; // p refers to an 'o' but it is not in l!

print(l);
```

```
 l.erase(l.begin());                    //removes front of l

 print(l);

 }

void print( list<char> a)
 {
 for(list<char>::iterator ai=a.begin(); ai!=a.end(); ++ai)
                    cout << *ai << " ";
          cout << endl;
          cout << "----------------"<<endl;
 }
```

O/P of previous program

o

o a t

o

c o a t

null

c a t

a t

Comparison of sequence containers

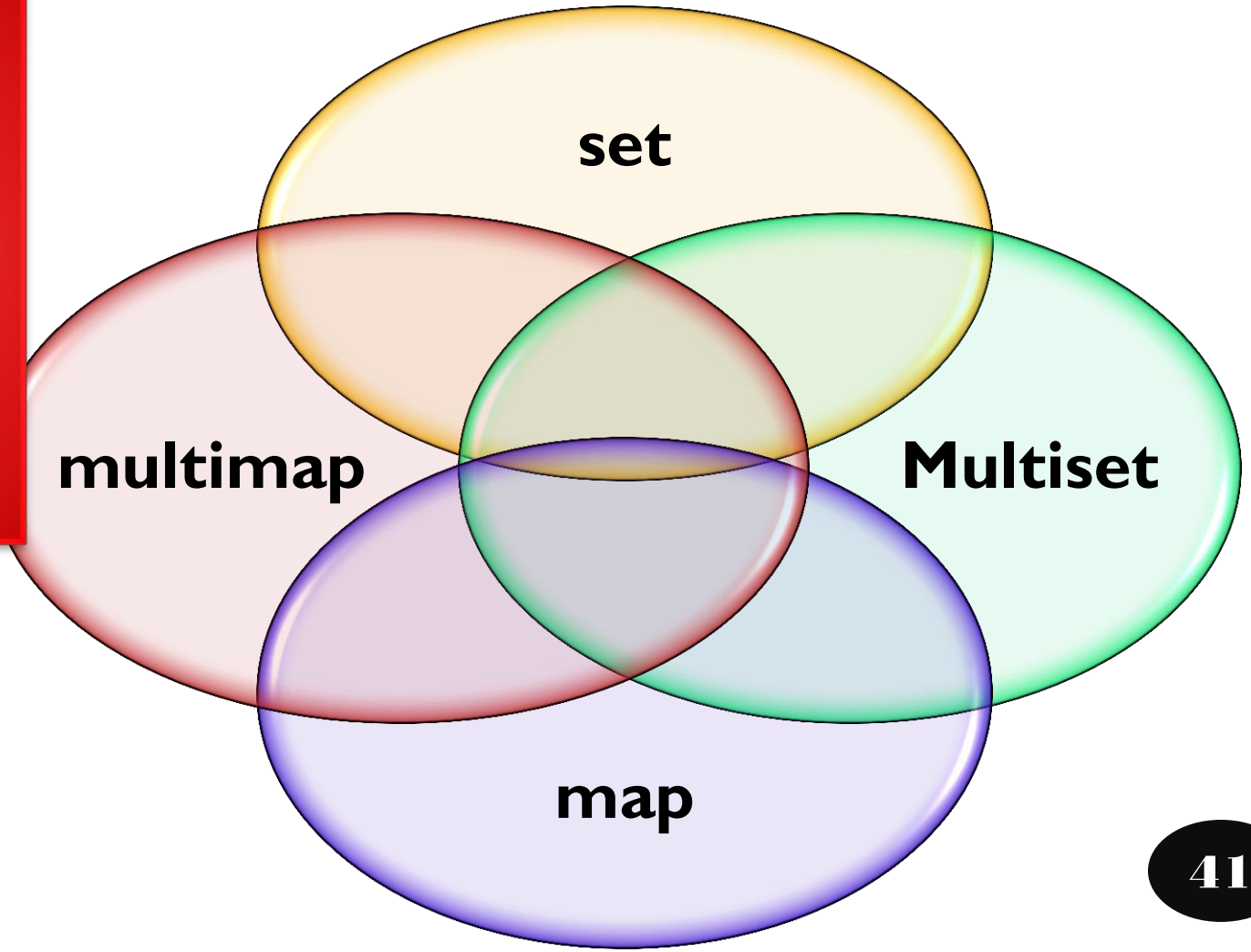
Container	Random Access	Insertion Deletion in middle	Insertion or Deletion at the ends
vector	Fast	Slow	Fast at Back
deque	Fast	Slow	Fast at both ends
list	Slow	Fast	Fast at front

ASSOCIATIVE CONTAINERS

- Non-sequential
- Supports direct access to elements using keys
- The keys are typically numbers or strings

FOUR TYPES OF ASSOCIATIVE CONTAINERS

All these store data in a structure called tree which facilitates fast searching



Set & Multiset : Associative Container

- Stores a number of items which contain key
- Elements here are referenced by keys and not their positions.
- Example : Storing the objects of **student** class which are ordered alphabetically using **names as keys**
- **Multiset** allows duplicate items while set does not

Set

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

using namespace std;

int main()
{
  string a[] = {"Alice", "Bob", "Carl", "Dick", "Eve", "Fred“} ;
  set<string> s(a, a+6);

   set<string>::iterator p = s.begin();
   while (p != s.end())
          cout << *p++ << endl;
    cout<<_____”<<endl;
```

Set

```
set<string>::size_type numberDeleted = s.erase("Bob");
 p = s.begin();
 while (p != s.end()) cout << *p++ << endl;
  cout<<_____"<<endl;

  numberDeleted = s.erase("William");
  p = s.begin();
  while (p != s.end()) cout << *p++ << endl;
   cout<<_____"<<endl;

  s.erase(s.begin());
  p = s.begin();
  while (p != s.end()) cout << *p++ << endl;
  cout<<_____"<<endl;

s.erase(s.find("Carl"), s.find("Eve"));
  p = s.begin();
  while (p != s.end()) cout << *p++ << endl;
```

Set

```
cout<<_____”<<endl;
 s.clear();
if (s.empty())
   cout << "\nThe set is now empty.";

}
```

O/P of previous program

Alice
Bob
Carl
Dick
Eve
Fred

Alice
Carl
Dick
Eve
Fred

O/P of previous program

Alice
Carl
Dick
Eve
Fred

Carl
Dick
Eve
Fred

O/P of previous program

Fred

The set is now empty.

MultiSet

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

class Book
{
   public :
   Book()
     {
          title = author = publisher = date = "";
     }
   Book(string a)
     {
         author = a;
          title = publisher = date = "";
    }
```

MultiSet

```
Book(string t, string a, string p, string d)
 {
        title = t;
        author = a;
        publisher = p;
        date = d;
}

 string Author()
   {
      return author;
    }
```

MultiSet

```
void GetInfo(string &t, string &a, string &p, string &d)
{
   t = title;
   a = author;
   p = publisher;
   d = date;
 }
private:
string author;
string title;
string publisher;
string date;
 };
```

Multiset

```
int main()
{
   multiset<Book> b;
   string a;

b.insert(Book("C++ book",“ABC",“McGraw-Hill", "1998"));
b.insert(Book(“Java ",“XYZ",“BB Publisher",“2001"));
b.insert(Book(“Let Us C",“Kanetkar",“McGraw-Hill ", "1997"));

multiset<Book>::iterator p = b.begin();
while (p != b.end())
  {
     cout<<*p++<<endl;
  }
};
```

O/P of previous program

C++ book
ABC
McGraw-Hill
1998

Java
XYZ
BB Publisher
2001

Let Us C
Kanetkar
McGraw-Hill
1997

Map & Multimap : Associative Container

- Stores **pair** of items, one called **key** and other **value**
- Manipulate the values using the keys associated with them
- Values are called as **mapped values**
- **Multimap** allows multiple keys while **map** does not

Map

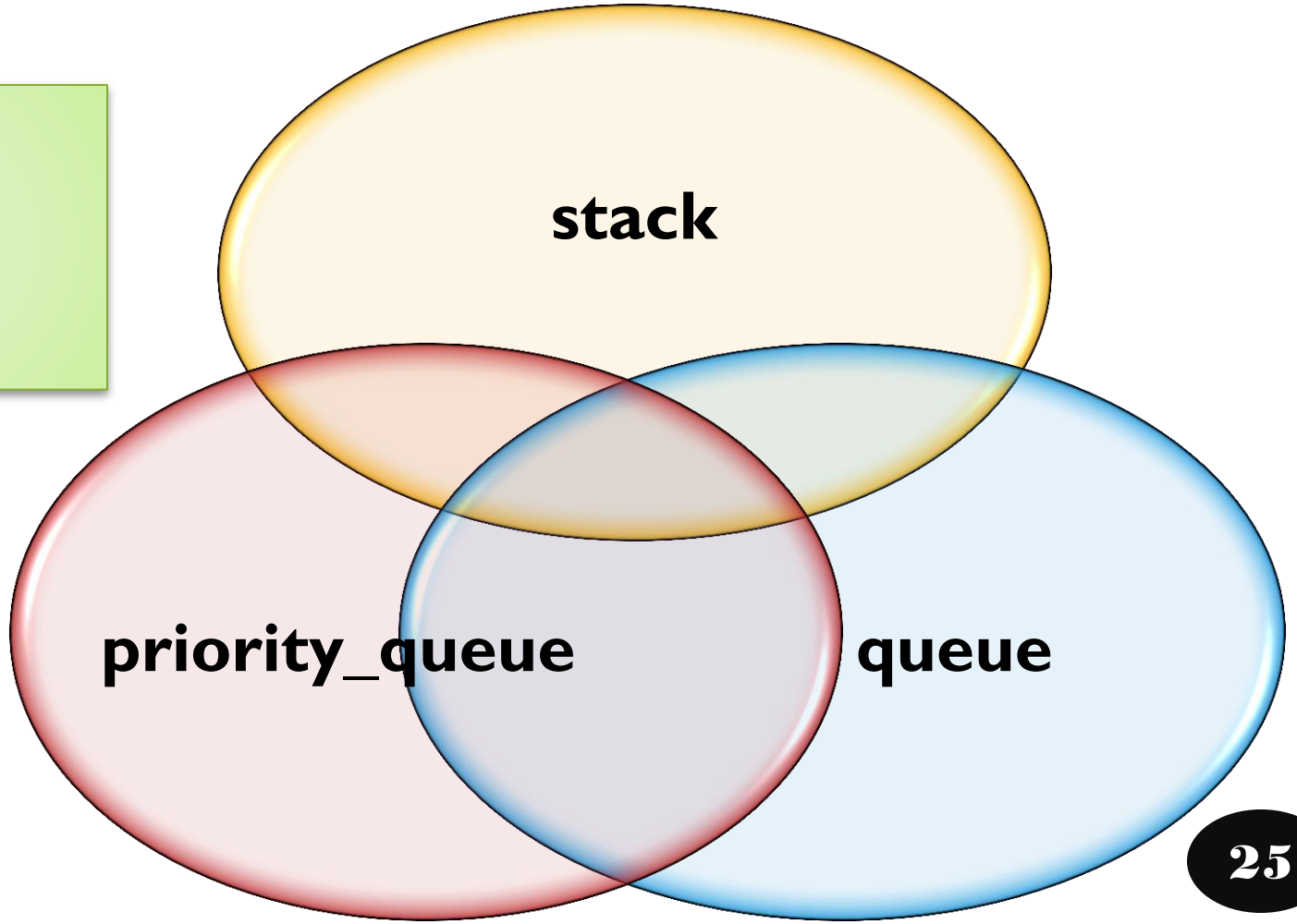
```
#include <map>
 #include <algorithm>
 #include <iostream>
 #include <string>

int main() {
 map<string,int> amap;

 amap[”First”]=1;
 amap[“Second”]=2;

 cout << "Size : " << amap.size() << endl;

 amap["Third"]=3;
 amap[”Fourth"]=4;
 cout << "Size : " << amap.size() << endl;
```

Map

```
map<string,int>::iterator it;

for ( it=amap.begin(); it!=amap.end(); it++)
  cout << "map : " << it->first << " "
                          << it->second << endl;
cout << amap.find("Third")->second << endl;
return 0;
}
```

THREE TYPES OF DERIVED CONTAINERS

These are known as container adaptors



Stack, Queue, Priority_Queue

```
#include <stack>
#include <queue>
using namespace std;

int main()
{
 // STL Stack
  stack<int, vector<int> > S;  // Changing default container

 for ( int i=0 ; i<10; i++ )
            S.push(i);

  for ( int i=0 ; i<10; i++ )
{
    cout << S.top() << " ";
    S.top() = 2 * S.top();
    cout << S.top() << endl;
    S.pop();
  }
```

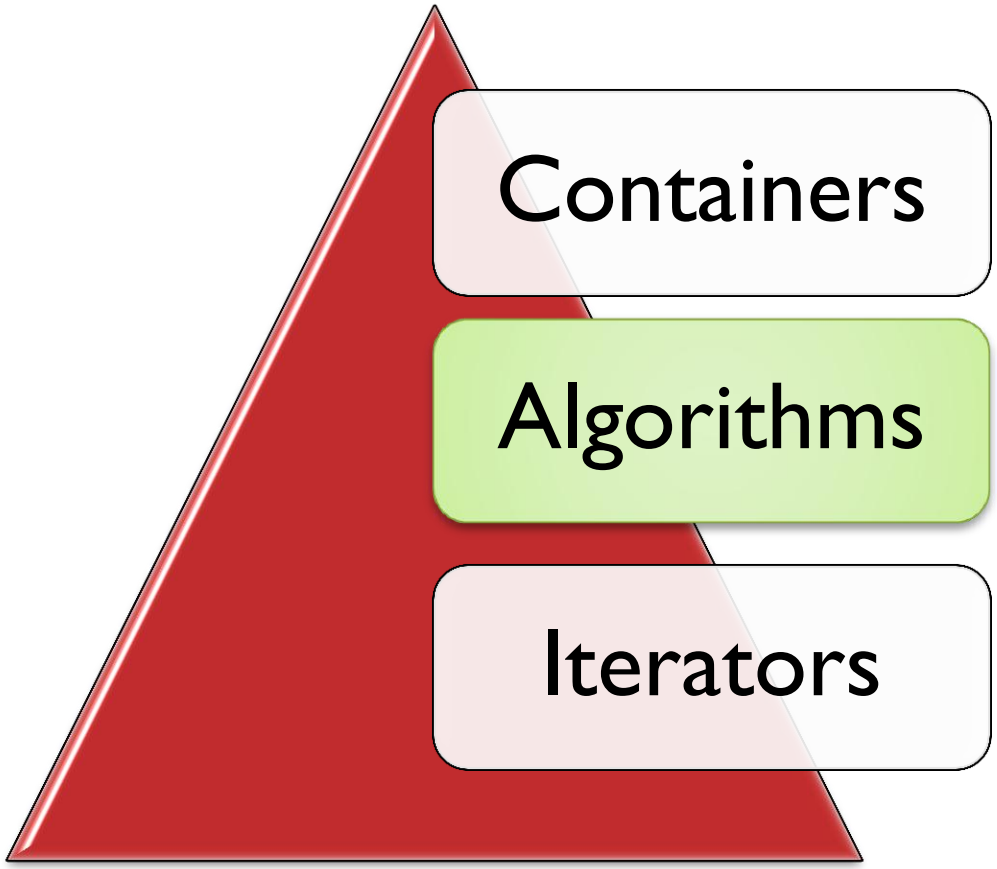
```
// STL Queue
 queue<int> Q;
 for ( int i=0 ; i<10; i++ )
        Q.push(i);
 for ( int i=0 ; i<10; i++ )
{
    cout << Q.front() << endl;
    Q.pop();
 }
```

```
// STL Priority Queue
 priority_queue<int> P;
 for ( int i=0 ; i<10; i++ )
        P.push(i);
 for ( int i=0 ; i<10; i++ )
  {
     cout << P.top() << endl;
     P.pop();
  }
}
```

Stack , Queue , Priority_Queue : Derived Containers

- Can be created from different sequence containers
- These do not support Iterators
- Therefore cannot be used for data manipulation
- Support two member functions : **push( ) and pop( )**

COMPONENTS OF STL



2. ALGORITMS

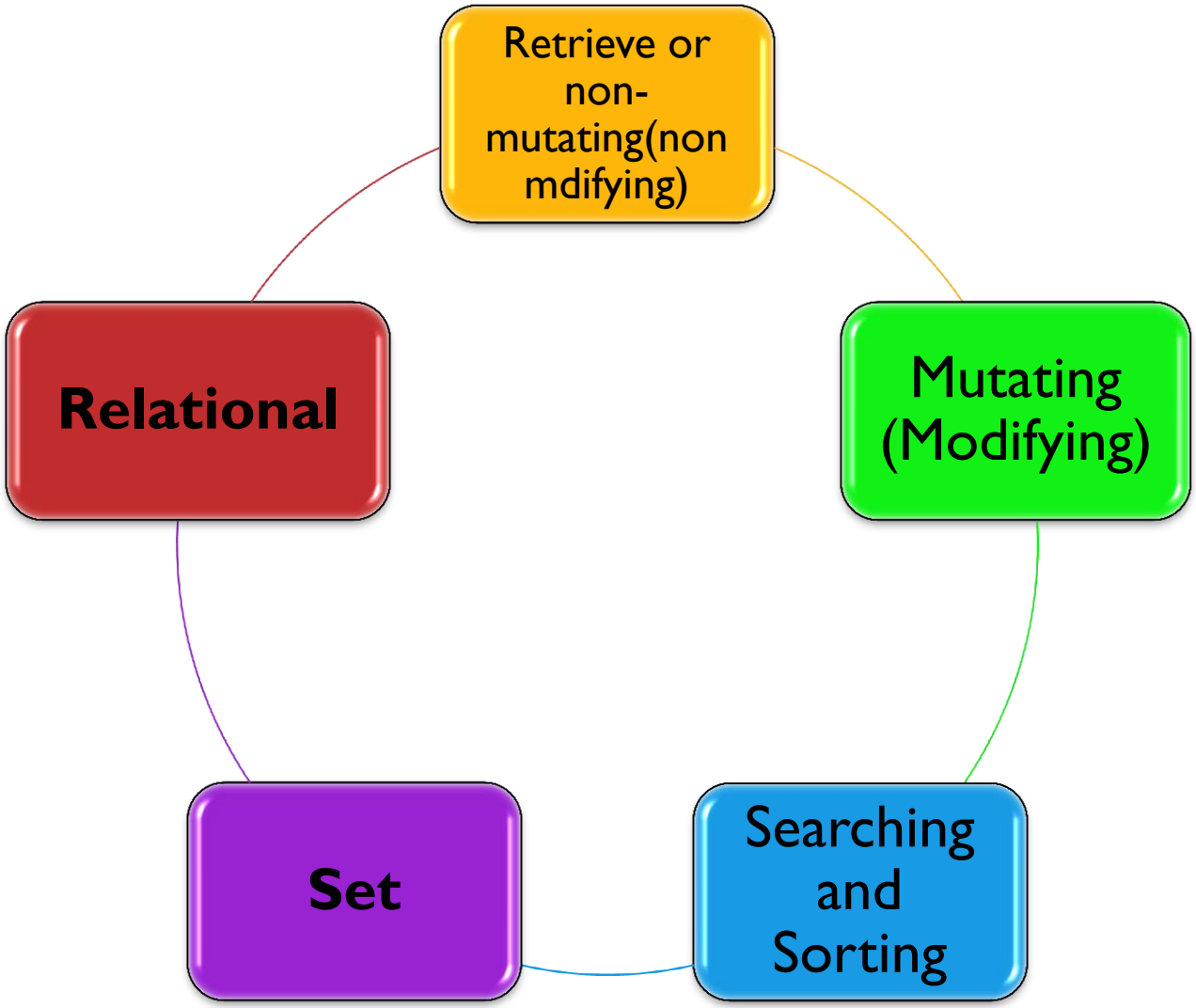
Generic functions that handle common tasks such as searching, sorting, comparing, and editing

More than 60 Algortihms exist

These are not member functions or friends of containers but are standalone template functions

To use them we include<algorithm> in the program

CATEGORY OF ALGORITHMS



Non-Mutating Algorithms

Operations	Description
search()	Searches desired element from the sequence
count()	Count appearances of value in range
count_if	Return number of elements in range satisfying condition
equal()	Test whether the elements in two ranges are equal
find()	Find position of desired element

Non-Mutating Algorithms

Operations	Description
find_end	Find last subsequence in range
find_first_of()	Find element from set in range
find_if()	Find element in range
for_each()	Apply function to range
mismatch()	Return first position where two ranges differ

Mutating Algorithms

Operations	Description
copy()	Copy sequence of elements
copy_backward( )	Copy range of elements backward
swap( )	Exchange values of two objects
fill()	Fill range with value
generate ( )	Generate values for range with function
reverse()	Reverse the given sequence
remove()	Remove value from sequence

Mutating Algorithms

Operations	Description
unique()	Remove consecutive duplicates in range
random_shuffle()	Randomly rearrange elements

Sorting Algorithms

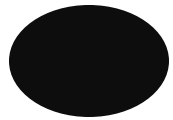
Operations	Description
sort()	Using quick sort elements are sorted
stable_sort( )	Using stable sort elements are sorted
merge()	Merging of two objects
sort_heap( )	Sort the created heap
min ( )	Finds minimum element
max()	Finds maximum element
binary_search()	Performs binary search on sorted elements

Algorithms

```
#include<algorithm>
#include<iostream>
using namespace std;

int  main()
 {
   vector<int> v;
    vector<int>p;
   v.push_back(10); v.push_back(20);    v.push_back(10);
 p.push_back(60); v.push_back(40);    v.push_back(50);

swap(v,p);
   int * ptr = find(a,a+6,20);
}
```

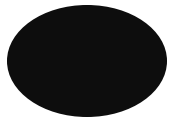
Algorithms

```
int n, value,arr[10],i;
int *Limit = arr + n;

cout<<" Enter the numbers" ;
for(i =0; i< n;++i)
 {
    cin>>value;
   arr[i] = value;
}

sort(arr, Limit);
```



Algorithms

```
cout<<“ Sorted List is” ;
for(i =0; i< n;++i)
 {
    cout<<arr[i];
    cout<<endl;
}

return 0;
}
```



Algorithm…Searching Example find , search , binary search

```
/ binary_search example
#include <iostream>     // std::cout
#include <algorithm>    // std::binary_search, std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }
int main () {
  int myints[] = {1,2,3,4,5,4,3,2,1};
   int my2ints[] = {5,4,3,2};

 vector<int> v(myints,myints+9);                    // 1 2 3 4 5 4 3 2 1
vector<int>::iterator it;
```

Algorithm…Searching Example find , search , binary search

```
// using default comparison:
  sort (v.begin(), v.end());


it = find(v.begin(), v.end(), 3);
  cout<<“Item found at position “ <<(it-v.begin());


it = search (v.begin(), v.end(), my2ints, my2ints+4);
    cout<<“Item found at position “ <<(it-v.begin());
```

Algorithm…Searching Example find , search , binary search

```
  cout << "looking for a 3... ";
  if (binary_search (v.begin(), v.end(), 3))
    cout << "found!\n"; else std::cout << "not found.\n";

  // using myfunction as comp:
  sort (v.begin(), v.end(), myfunction);

 cout << "looking for a 6... ";
  if (binary_search (v.begin(), v.end(), 6, myfunction))
    std::cout << "found!\n"; else std::cout << "not found.\n";

  return 0;
}
```

Output:

looking for a 3... found!
looking for a 6... not found.

Algorithm…Min Max Example

```
/ min max example
#include <iostream>     // std::cout
#include <algorithm>

int main ()
{
  cout<<"\n  min(20,10) = " <<min(20,10);
  cout<<"\n min('a','b) – " <<min('a','b');
  cout<<"\n max('e', 'f') = <<max('e','f');
}
```

Algorithm…Set Union Example

```
#include <iostream>     // std::cout
#include <algorithm>
#include <vector>       // std::vector
int main ()
{
  int first[] = {5,10,15,20,25};
  int second[] = {50,40,30,20,10};
  vector<int> v(10);                      // 0  0  0  0  0  0  0  0  0  0
  vector<int>::iterator it;

  std::sort (first,first+5);     //  5 10 15 20 25
  std::sort (second,second+5);   // 10 20 30 40 50
```

Algorithm…Set OperationsExample

```
it= set_union (first, first+5, second, second+5, v.begin());
                                    // 5 10 15 20 25 30 40 50  0  0
  v.resize(it-v.begin());                    // 5 10 15 20 25 30 40 50

  cout << "The union has " << (v.size()) << " elements:\n";
  for (it=v.begin(); it!=v.end(); ++it)
      cout << ' ' << *it;
   cout << '\n';

  return 0;
}
```

Output:

The union has 8 elements:
5 10 15 20 25 30 40 50

Algorithm…Set Intersection Example

```
#include <iostream>     // std::cout
#include <algorithm>
#include <vector>       // std::vector
int main ()
{
  int first[] = {5,10,15,20,25};
  int second[] = {50,40,30,20,10};
  vector<int> v(10);                    // 0  0  0  0  0  0  0  0  0  0
  vector<int>::iterator it;

  std::sort (first,first+5);     //  5 10 15 20 25
  std::sort (second,second+5);   // 10 20 30 40 50
```

Algorithm…Set Intersection Example

```
it= set_intersection (first, first+5, second, second+5, v.begin());
                                        // 5 10 15 20 25 30 40 50  0  0
 v.resize(it-v.begin());                    // 5 10 15 20 25 30 40 50

 cout << "The intersection has has " << (v.size()) << " elements:\n";
 for (it=v.begin(); it!=v.end(); ++it)
     cout << ' ' << *it;
  cout << '\n';

 return 0;
}
```

Output:

The intersection has 2 elements:
{10 20}

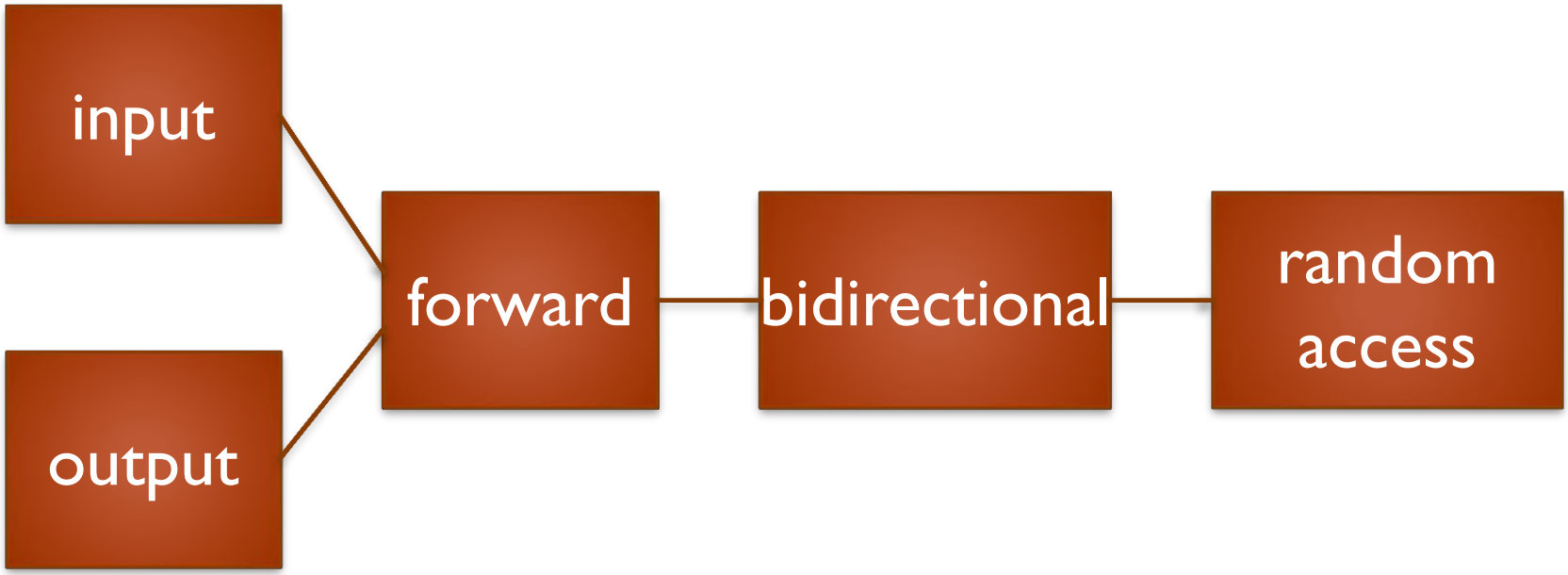
Algorithm…Set difference Example

```
it= set_difference (first, first+5, second, second+5, v.begin());
                                    // 5 10 15 20 25 30 40 50  0  0
 v.resize(it-v.begin());                    // 5 10 15 20 25 30 40 50

 cout << "The intersection has has " << (v.size()) << " elements:\n";
 for (it=v.begin(); it!=v.end(); ++it)
     cout << ' ' << *it;
  cout << '\n';

 return 0;
}
```

Output:

The difference has 3 elements:
{5 15 25}

Iterators



Input & Output Iterator

Provides least functions

Used only to traverse in a container

Forward Iterator

Supports all functions of input & output iterators

Retain its position in the container

Bi-directional Iterator

Supports all functions of forward iterators

Provides ability to move in backward direction in the container

Random – Access Iterator

Supports all functions of bi-directional iterators

Has the ability to jump to any arbirtary location

Iterators and their characteristics

Iterator	Access Method	Direction of Movement	I/O Capability
Input	Linear	Forward	Read
Output	Linear	Forward	Write
Forward	Linear	Forward	Read/Write
Bi-directional	Linear	Forward & Backward	Read/Write
Random Access	Random	Forward & Backward	Read/Write

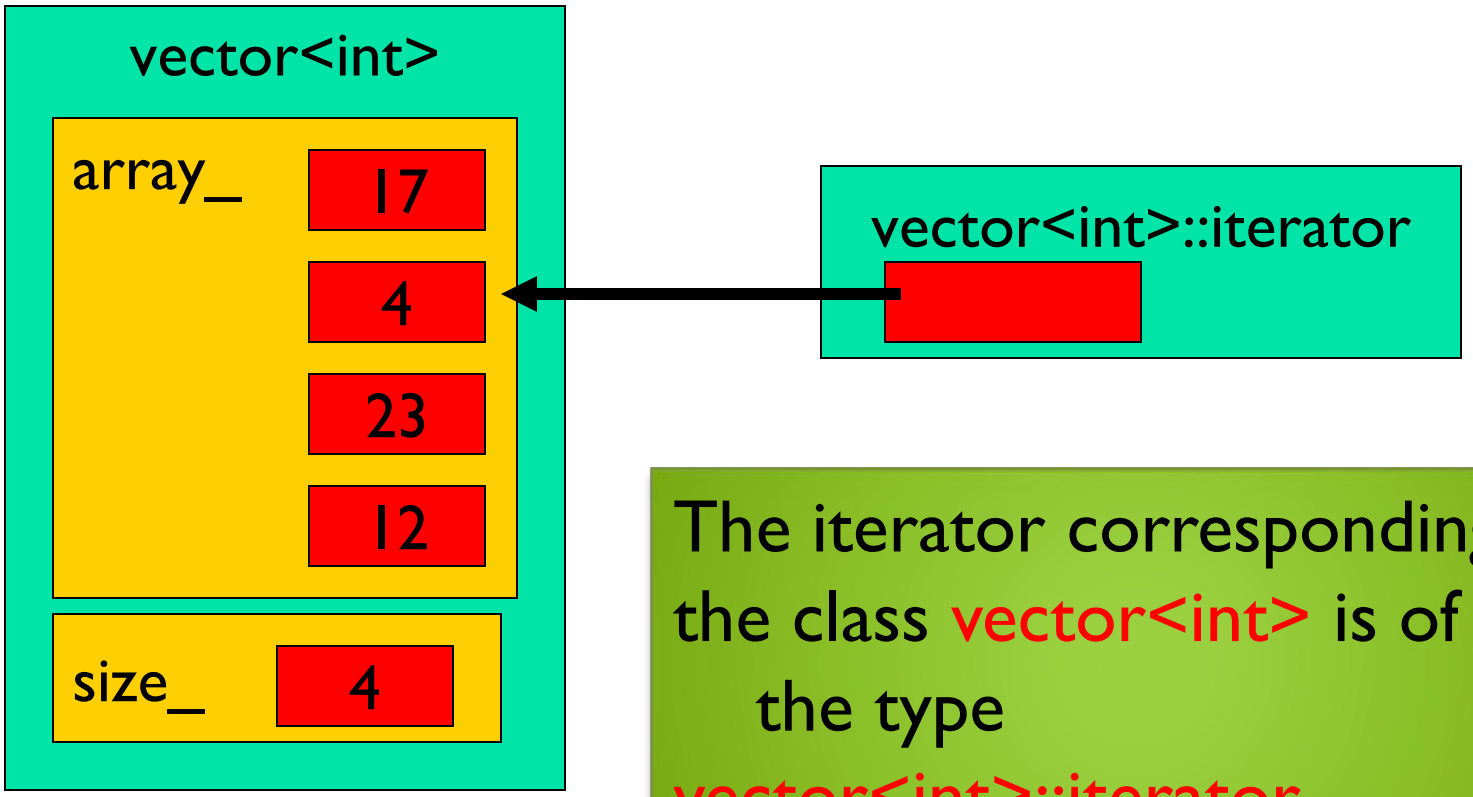
Iterators and their Providers

Iterator	Provider	Example
Input	istream	T1.cpp
Output	ostream, inserter	T1.cpp T2.cpp
Forward		
Bi-directional	list,set, multiset, map, multimap	
Random Access	vector, deque, array string	

Operations Supported by Iterators

Iterator	Element Access	Read	Write	Increment	Comparison
Input		v=*p		++	==, !=
Output			*p=v	++	
Forward		v=*p	*p=v	++	==, !=
Bi-directional		v=*p	*p=v	++,--	==, !=
Random Access	 & [ ]	v=*p	*p=v	++,--,+,-	==, !=,<,>, <=,>=

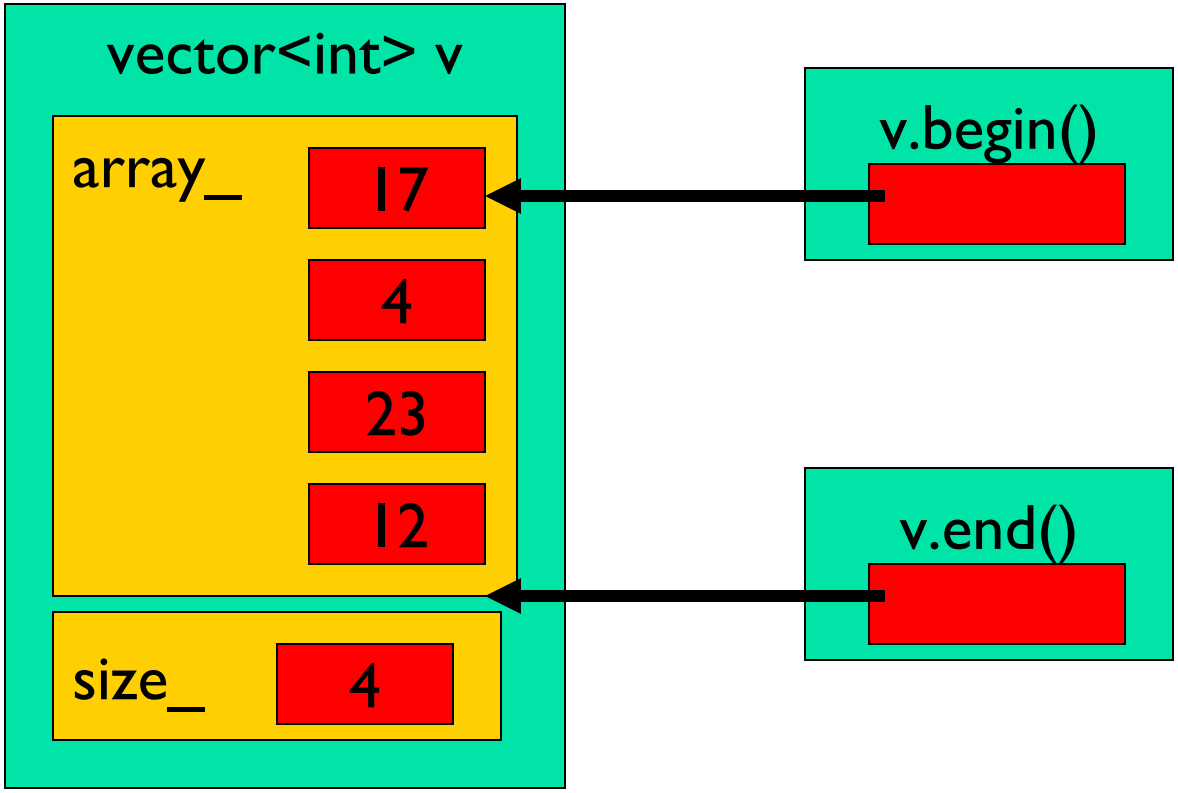
Iterator



The iterator corresponding to the class vector<int> is of the type vector<int>::iterator

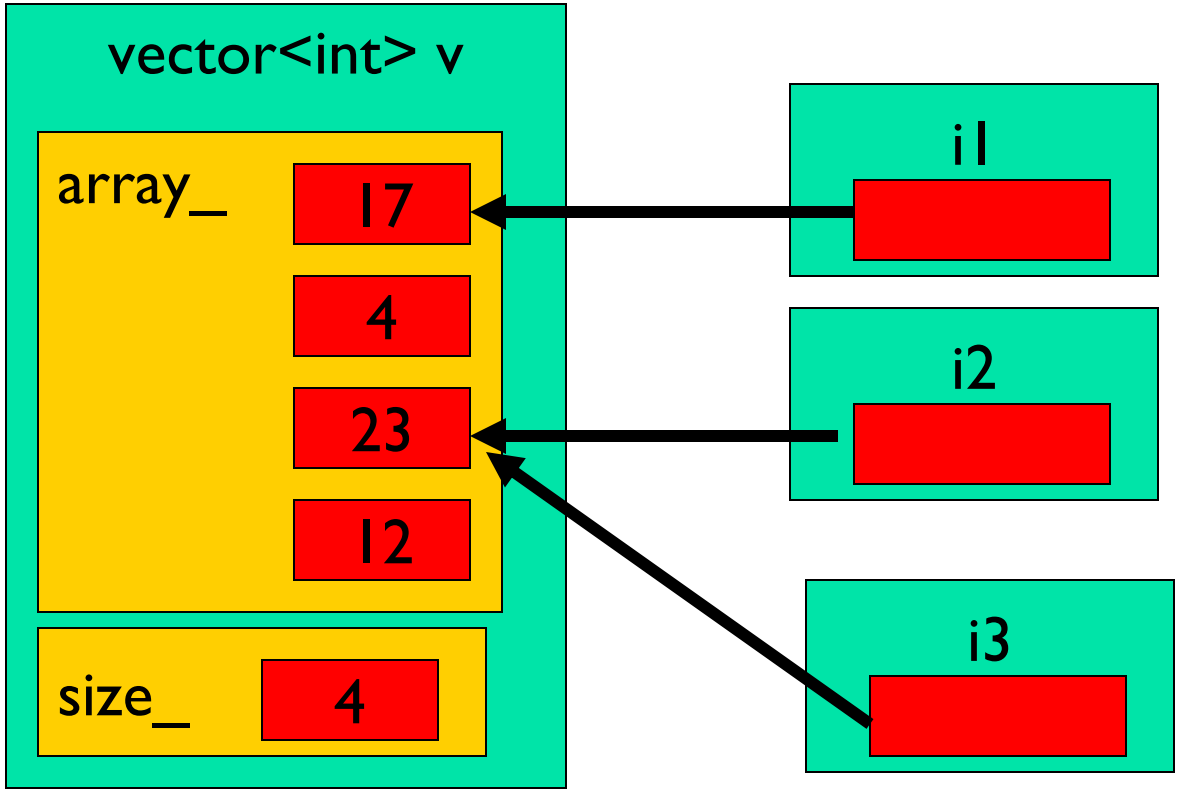
Iterator

The member functions begin() and end() return an iterator to the first and past the last element of a container



Iterator

One can have multiple iterators pointing to different or identical elements in the container



Istream & ostream iterator…Input & output iterator Example

```
// istream_iterator example
#include <iostream>     // std::cin, std::cout
#include <iterator>     // std::istream_iterator

int main () {
  double value1, value2;
  std::cout << "Please, insert two values: ";
  std::istream_iterator<double> iit (std::cin);   // stdin iterator
std::ostream_iterator<int> ot(std""cout," ");
  value1=*iit;
  ++iit;
  value2=*iit;
  std::cout << value1 << "*" << value2 << "=" << (value1*value2) <<
'\n';
  return 0;   }
```

Inserter Example

```
#include <iostream>     // std::cout
#include <iterator>     // std::front_inserter
#include <list>         // std::list
#include <algorithm>    // std::copy
int main ()
{
    list<int> l1, l2;
    for (int i=1; i<=5; i++)
      {
        l1.push_back(i);
       l2.push_back(i*10);
      }
```

Inserter Example …Continued

```
list<int>::iterator it = l1.begin();
  advance (it,3);
copy (l2.begin() , l2.end() , inserter(l1,it));

  std::cout << “l1 contains:";
  for ( it = l1.begin(); it!= l1.end(); ++it )
          cout << ' ' << *it;
  std::cout << '\n';
  return 0;
}
```

1 2 3 10 20 30 40 50 4 5 …Output