Contemporary C++: Learning Modern C++ in a Modern Way

الماس فناوری ابری پاسارگاد- آلفا

مدرس: سعید امراللهی بیوکی

Agenda 20/24

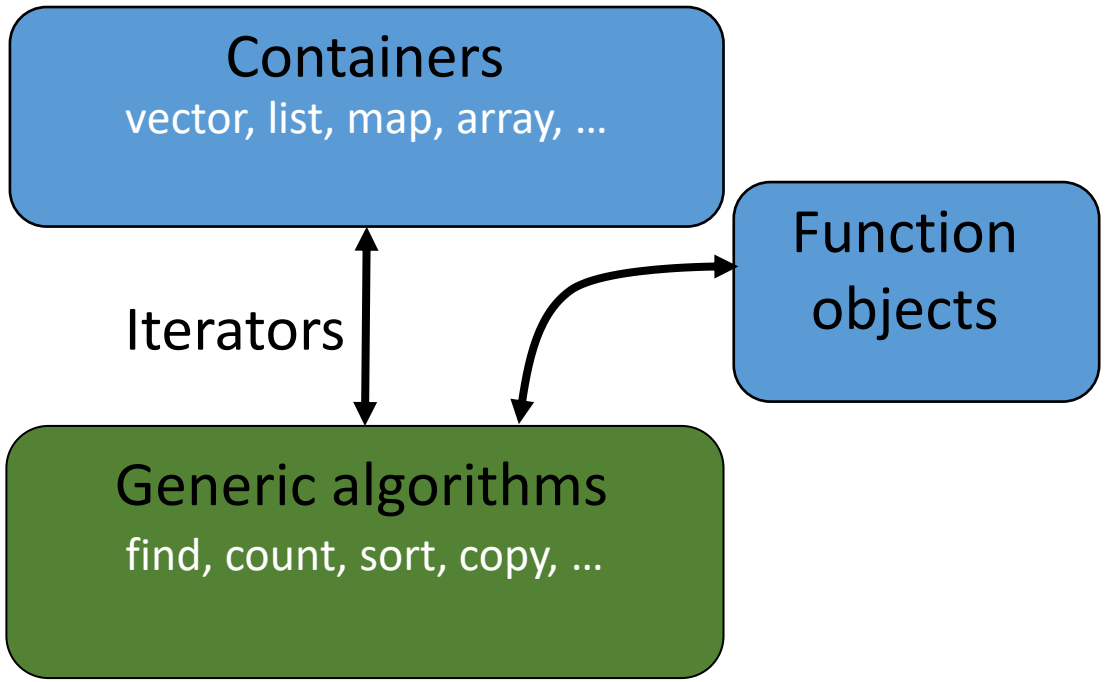
Session 20. Introduction to Standard Template Library: STL Architecture, Containers and Iterators

- The C++ Standard library
- STL architecture
- Containers classification: Vectors, Lists, Maps and Unordered maps
- Container members
- Vector as the default container
- Iterators and Iterator categories
- Q&A

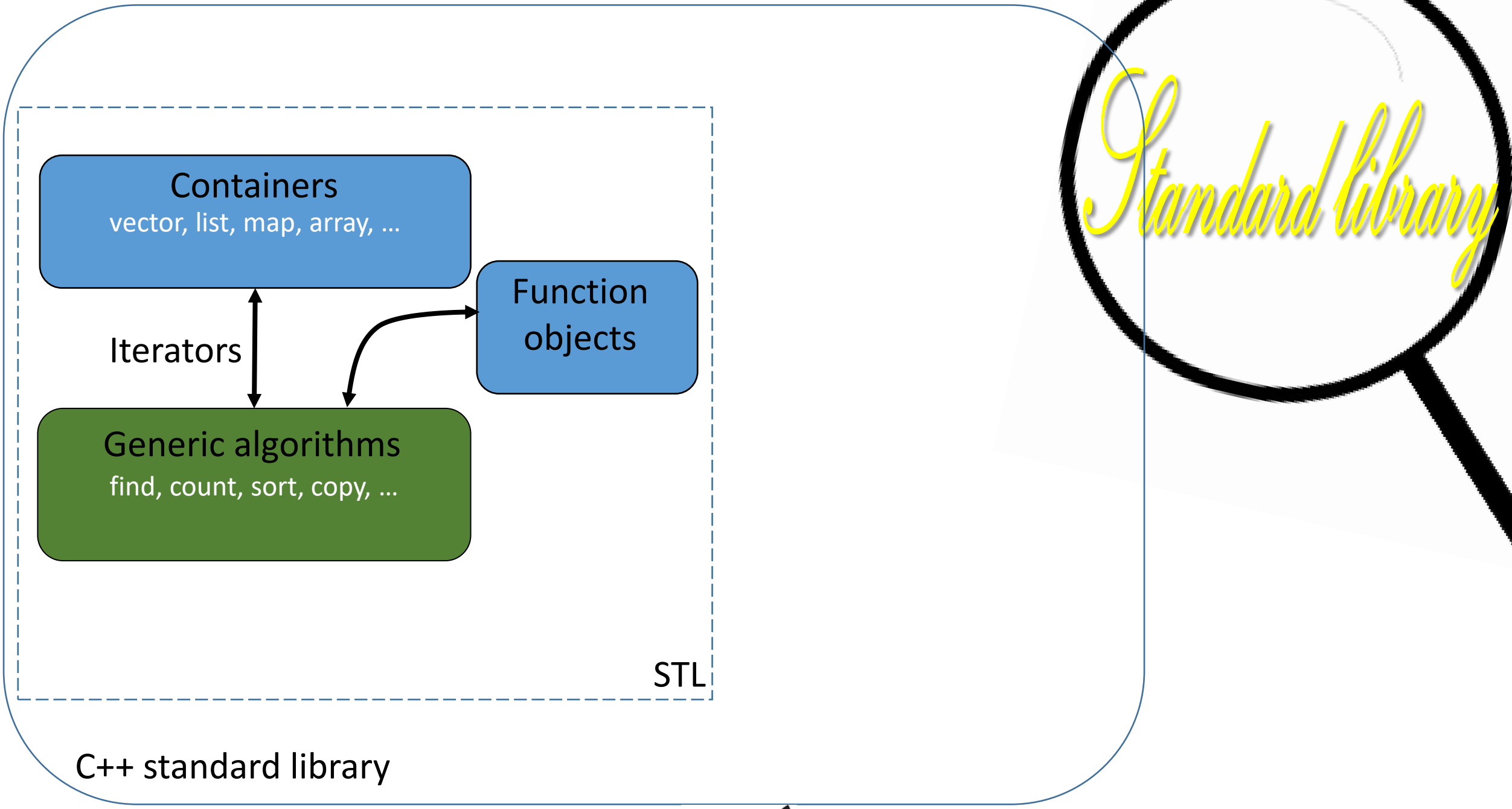
150 min (incl. Q & A)

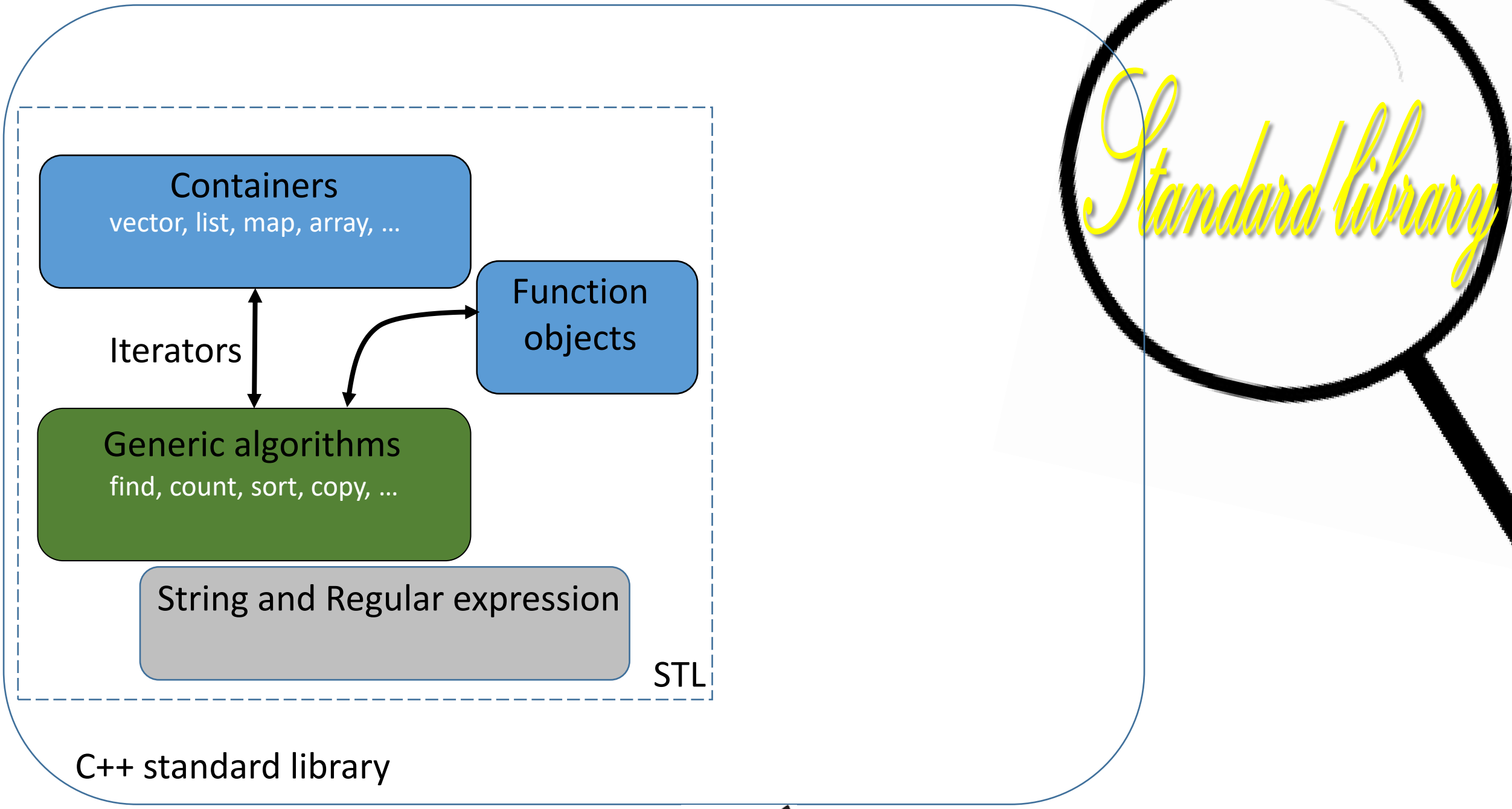

Standard library

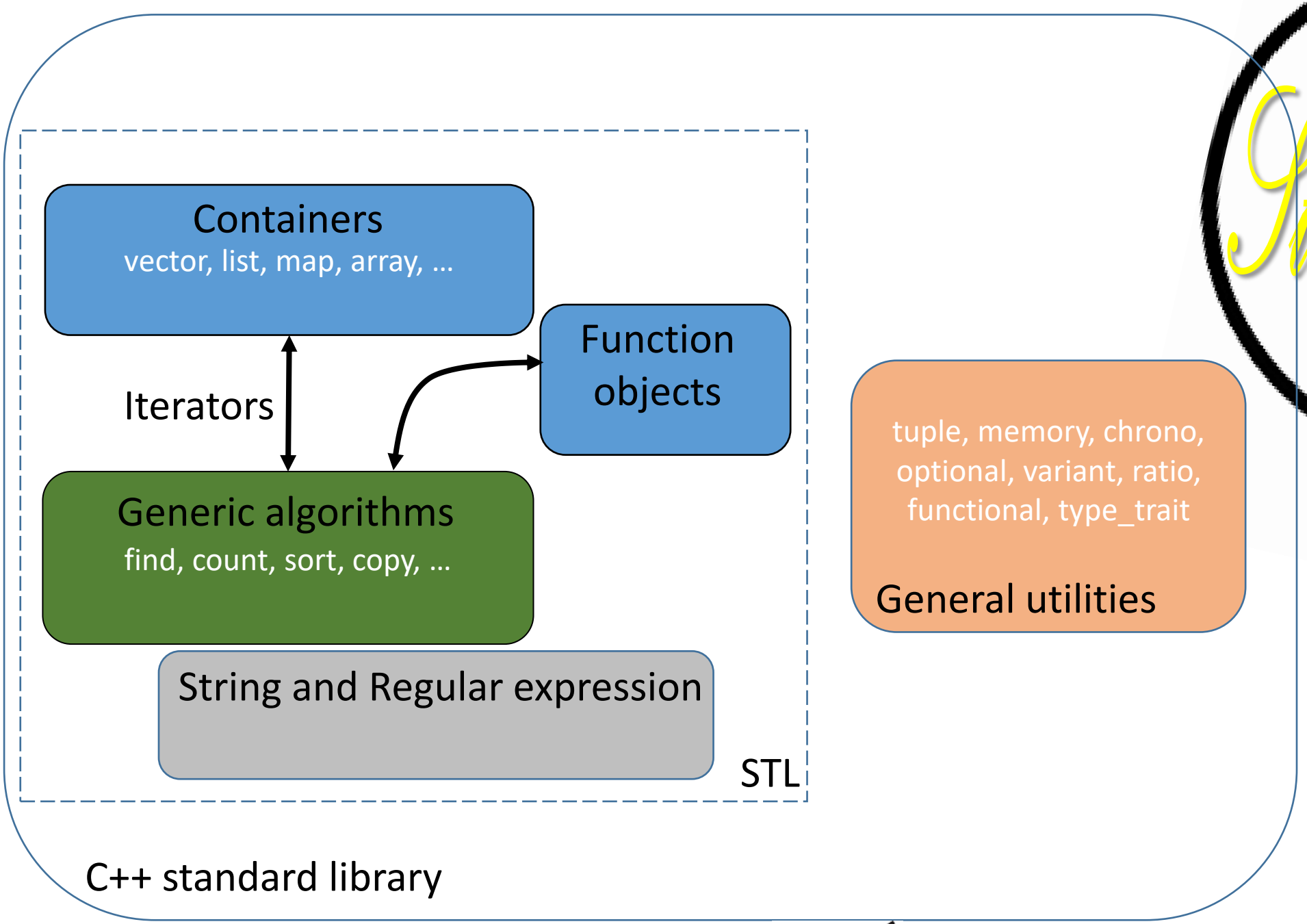
C++ standard library



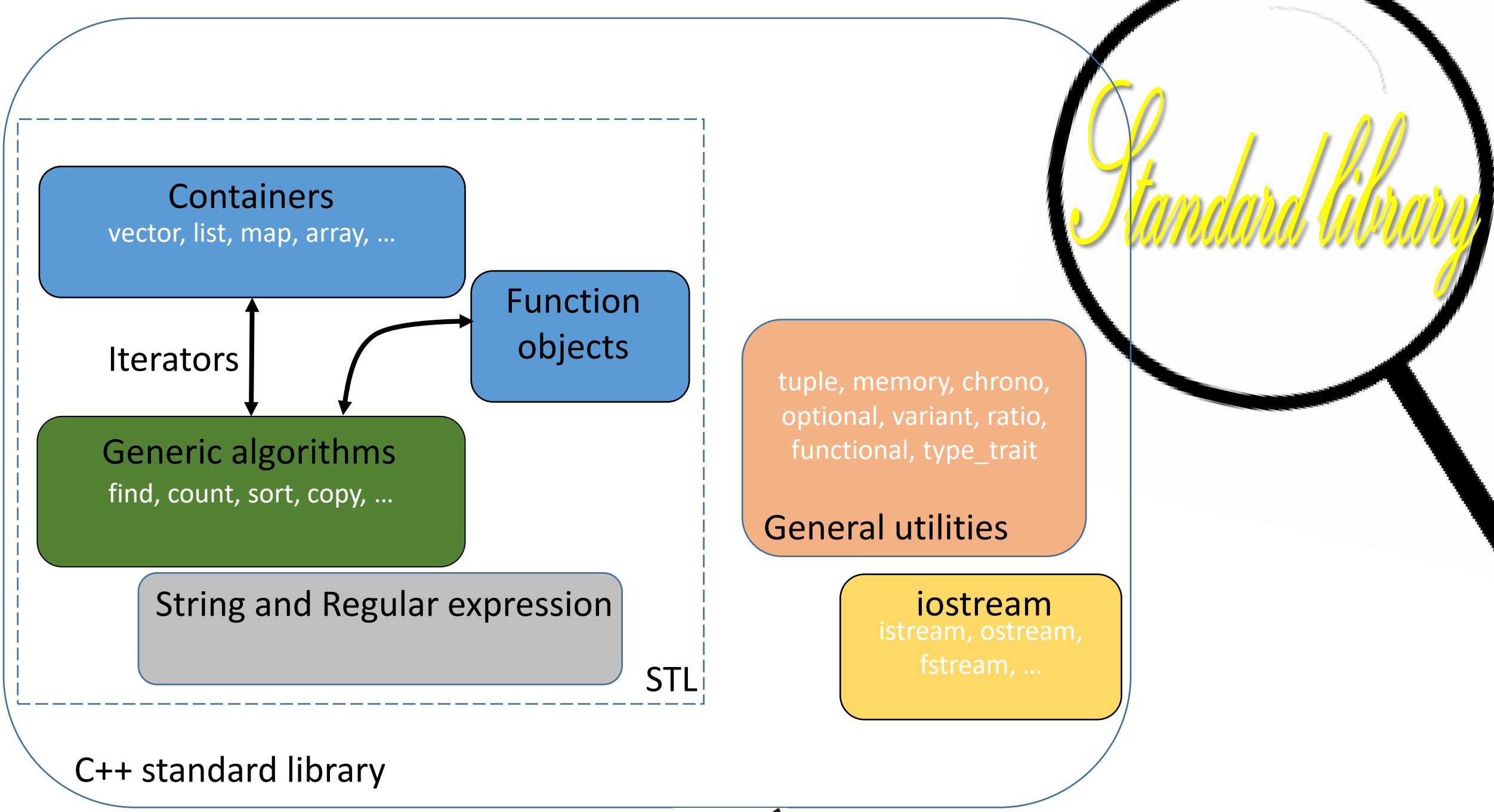
C++ standard library

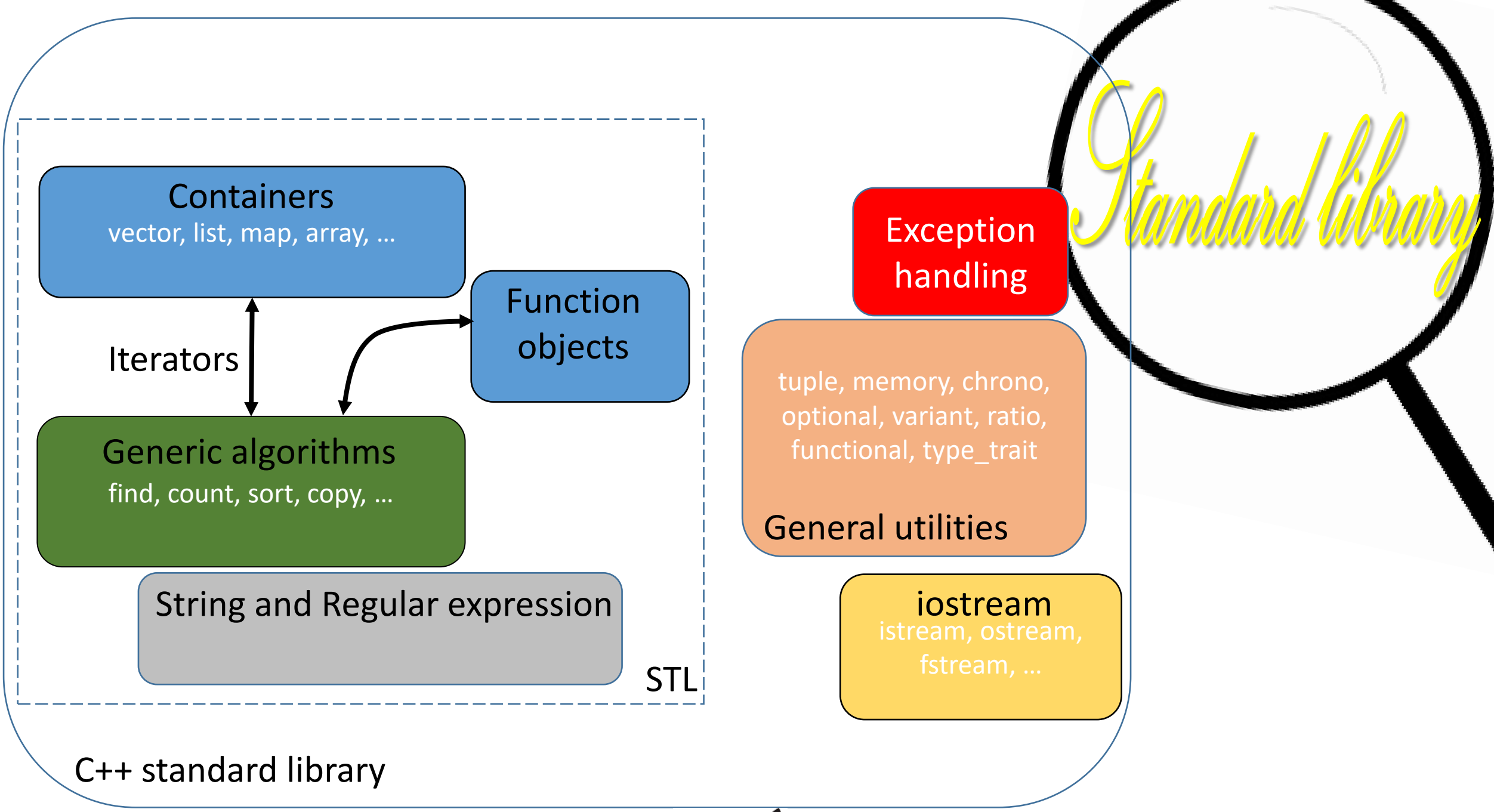

Containers
vector, list, map, array, …
Iterators
Function objects
Generic algorithms
find, count, sort, copy, …
STL
C++ standard library
Standard library

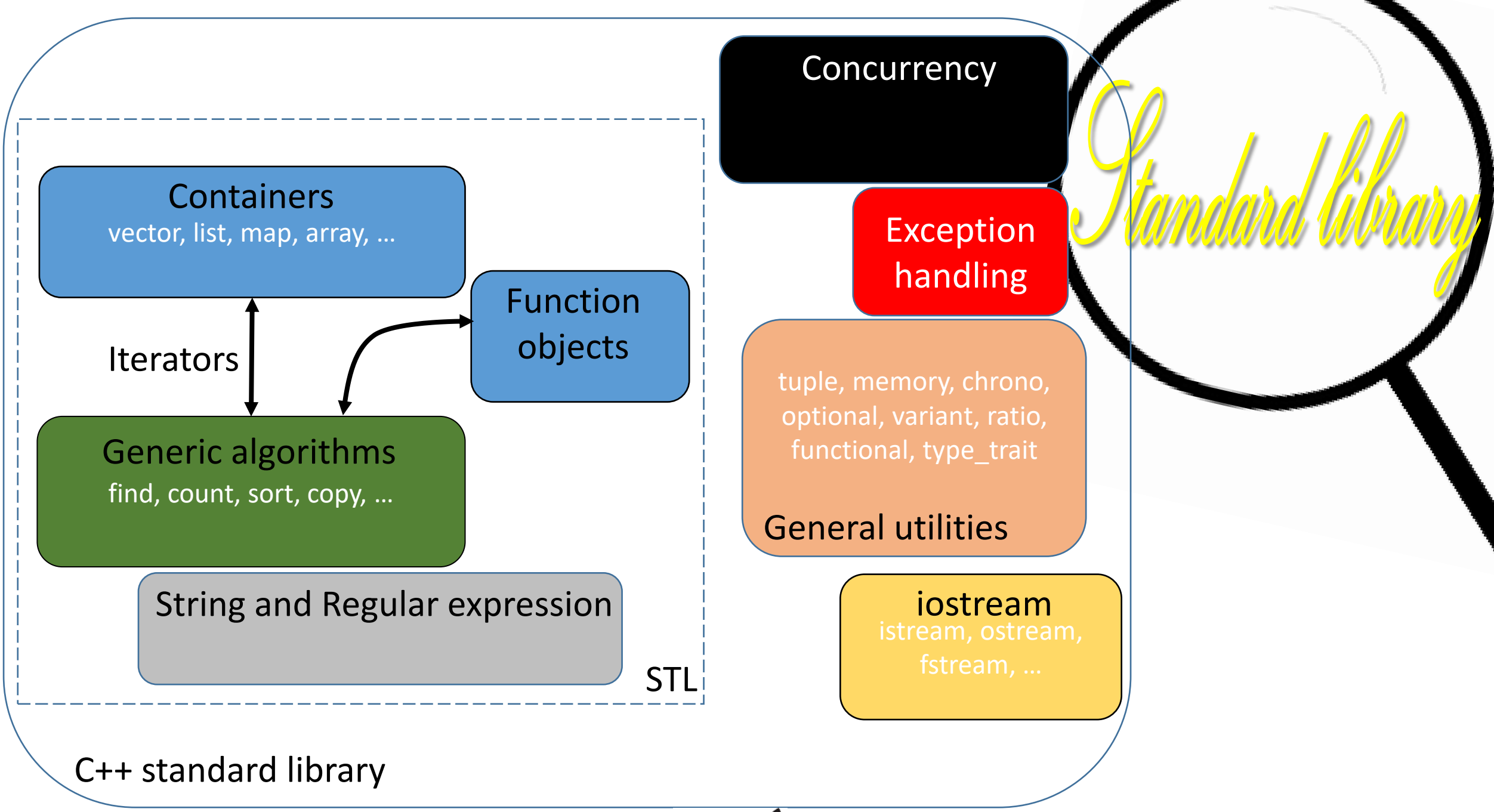

Containers
vector, list, map, array, …
Iterators
Function objects
Generic algorithms
find, count, sort, copy, …
String and Regular expression
STL
C++ standard library
Standard library

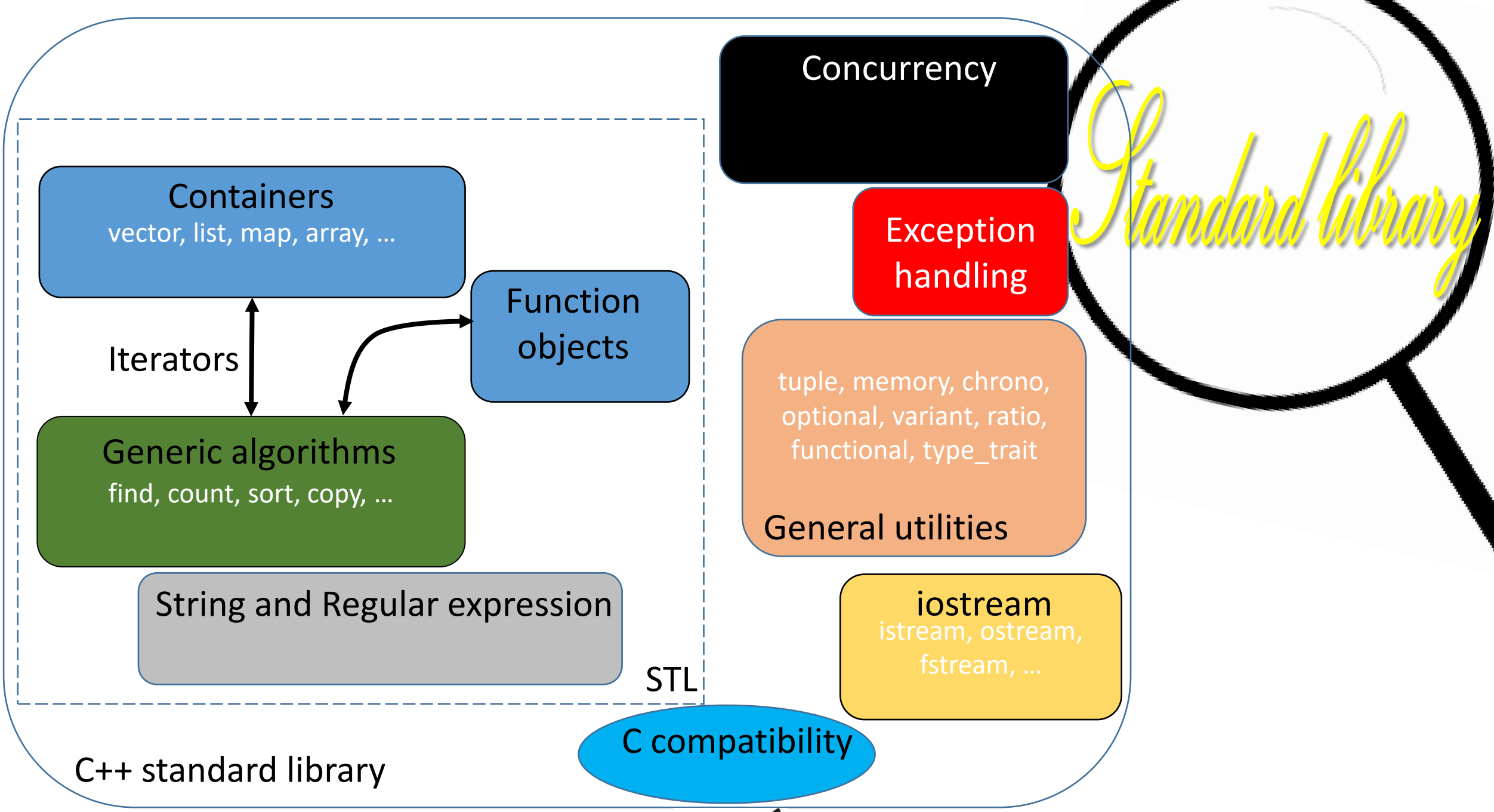

Containers
vector, list, map, array, …
Iterators
Function objects
Generic algorithms
find, count, sort, copy, …
String and Regular expression
STL
tuple, memory, chrono, optional, variant, ratio, functional, type_trait
General utilities
C++ standard library


Standard library


Containers
vector, list, map, array, …
Iterators
Function objects
Generic algorithms
find, count, sort, copy, …
String and Regular expression
STL
tuple, memory, chrono, optional, variant, ratio, functional, type_trait
General utilities
iostream
istream, ostream, fstream, …
C++ standard library
Standard library


Containers
vector, list, map, array, …
Iterators
Function objects
Generic algorithms
find, count, sort, copy, …
String and Regular expression
STL
C++ standard library
Exception handling
tuple, memory, chrono, optional, variant, ratio, functional, type_trait
General utilities
iostream
istream, ostream, fstream, …
Standard library


Concurrency
Standard library
Containers
vector, list, map, array, …
Exception handling
Function objects
Iterators
tuple, memory, chrono, optional, variant, ratio, functional, type_trait
General utilities
Generic algorithms
find, count, sort, copy, …
String and Regular expression
iostream
istream, ostream, fstream, …
STL
C++ standard library


Concurrency
Standard library
Containers
vector, list, map, array, …
Exception handling
Function objects
Iterators
tuple, memory, chrono, optional, variant, ratio, functional, type_trait
General utilities
Generic algorithms
find, count, sort, copy, …
String and Regular expression
iostream
istream, ostream, fstream, …
STL
C compatibility
C++ standard library

Standard library

Standard library

- The facilities of the standard library are defined in the std namespace and presented as a set of headers.

Standard library

- The facilities of the standard library are defined in the std namespace and presented as a set of headers.
- About 3 quarter of ISO C++ committee draft is on standard library.

Standard library

- The facilities of the standard library are defined in the std namespace and presented as a set of headers.
- About 3 quarter of ISO C++ committee draft is on standard library.
- The standard library aims to be the common foundation for other libraries.

Standard library

- The facilities of the standard library are defined in the std namespace and presented as a set of headers.
- About 3 quarter of ISO C++ committee draft is on standard library.
- The standard library aims to be the common foundation for other libraries.

C++ Standard Library

Standard library

- The facilities of the standard library are defined in the std namespace and presented as a set of headers.
- About 3 quarter of ISO C++ committee draft is on standard library.
- The standard library aims to be the common foundation for other libraries.

C++ Standard Library

Standard library

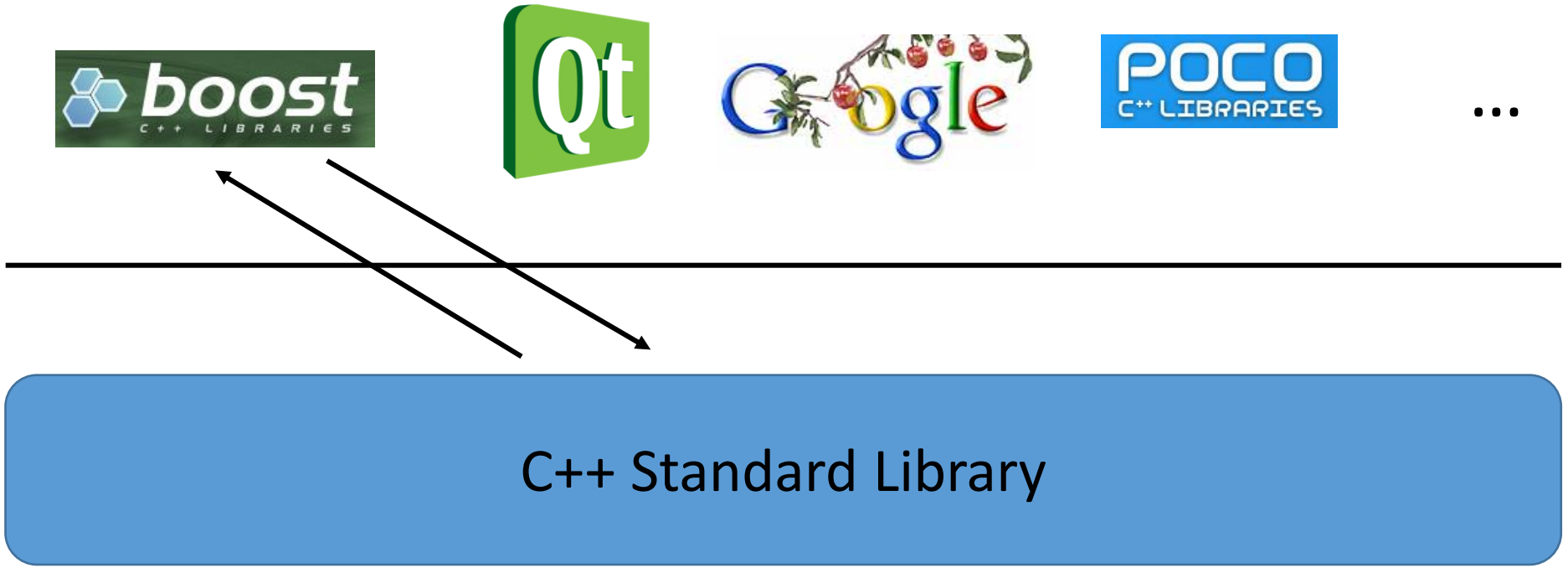
- The facilities of the standard library are defined in the std namespace and presented as a set of headers.
- About 3 quarter of ISO C++ committee draft is on standard library.
- The standard library aims to be the common foundation for other libraries.

...

C++ Standard Library

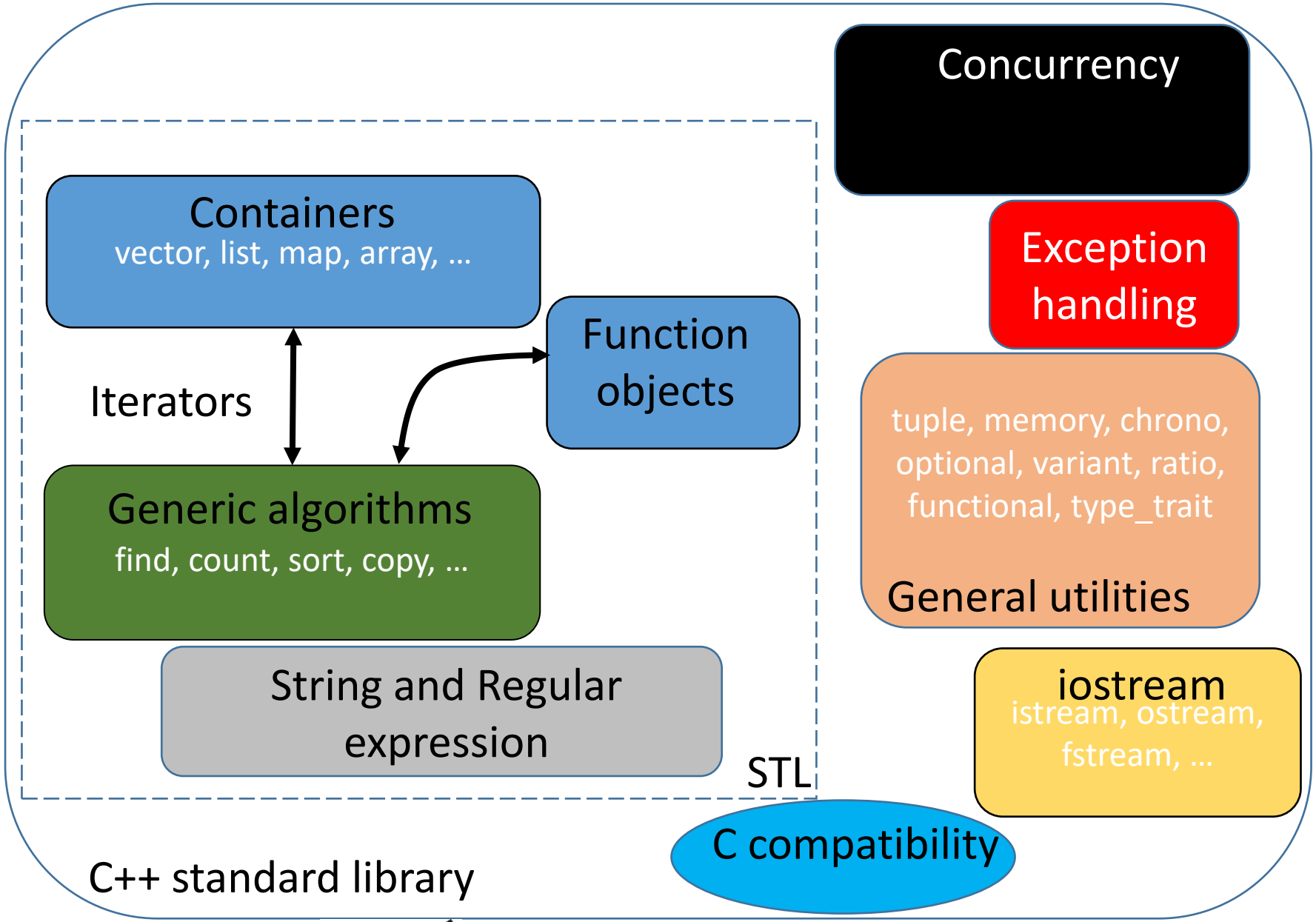
Standard library

- The facilities of the standard library are defined in the std namespace and presented as a set of headers.
- About 3 quarter of ISO C++ committee draft is on standard library.
- The standard library aims to be the common foundation for other libraries.



- Use standard-library facilities to maintain portability.
- Use standard-library facilities to minimize maintenance costs.
- Use standard-library facilities as a base for more extensive and more specialized libraries.
- Use standard-library facilities as a model for flexible, widely usable software.

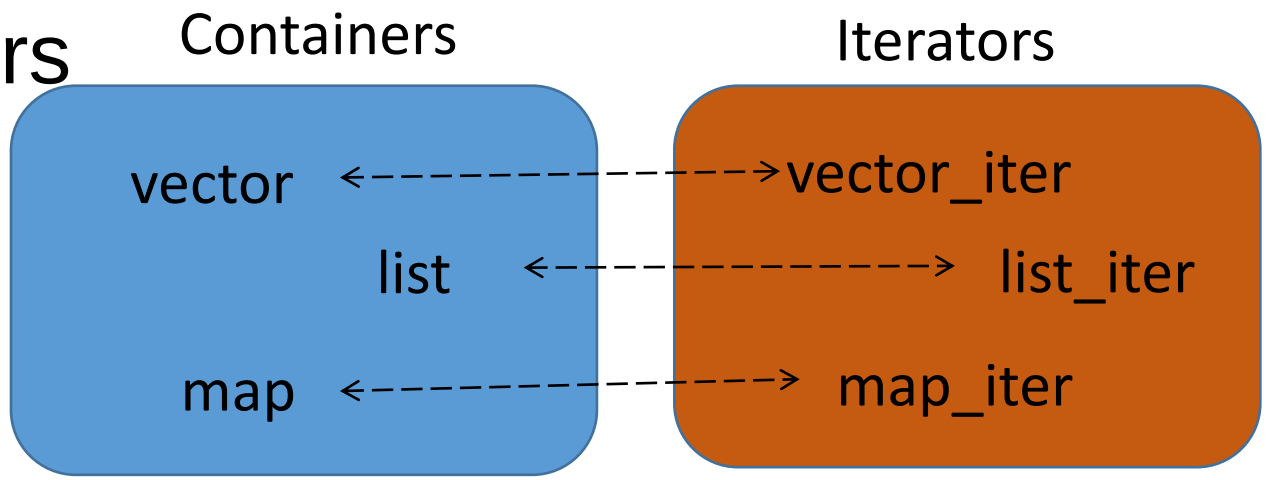
Standard library



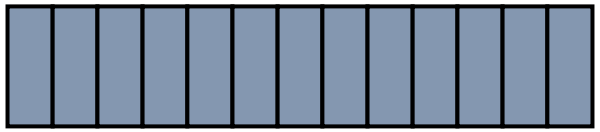
Container design- traditional approach

1. Specialized containers & Iterators
2. Based containers

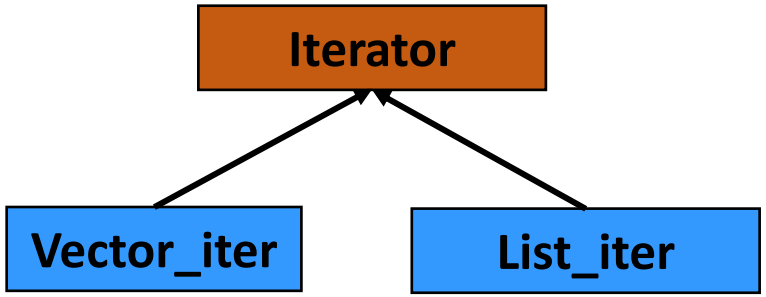
Specialized containers & Iterators




```
template<class T> class Vector {
public:
    explicit Vector(int sz);
    int size();
    bool empty();
    void push_back(const T& t);
    void pop_back();
    T& operator[](int index);
    // other member functions
private:
    // representation
};
```



```
template <class T> class Iterator { // common interface
public:
    // return 0 to indicate no-more-elements
    virtual T* first() =0;
    virtual T* next() =0;
    // other traverse operations
};
```

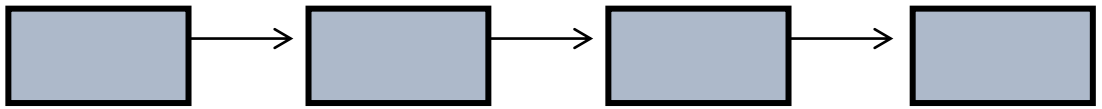



```
template <class T> class Vector_iter : public Iterator<T> {
    Vector<T>& v;
    int index; // index of current element
public:
    Vector_iter(Vector<T>& vv) : v(vv} {}
    virtual T* first() { return v.size() ? &v[size()] : 0;
    virtual T* next() { return ++index < v.size() ? &v[index] : 0;
    // other traverse operations
};
```

Specialized containers & Iterators cont.

```
template<class T> class List {
    int size();
    bool empty();
    void push_back(const T& t);
    void pop_back();
    void push_front(const T& t);
    void pop_front();
    // other member functions
private:
    // representation
};
```

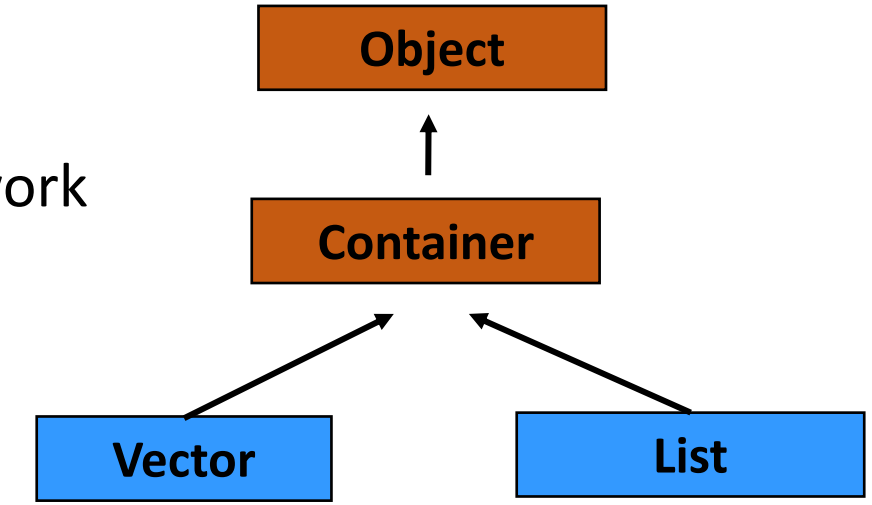
...

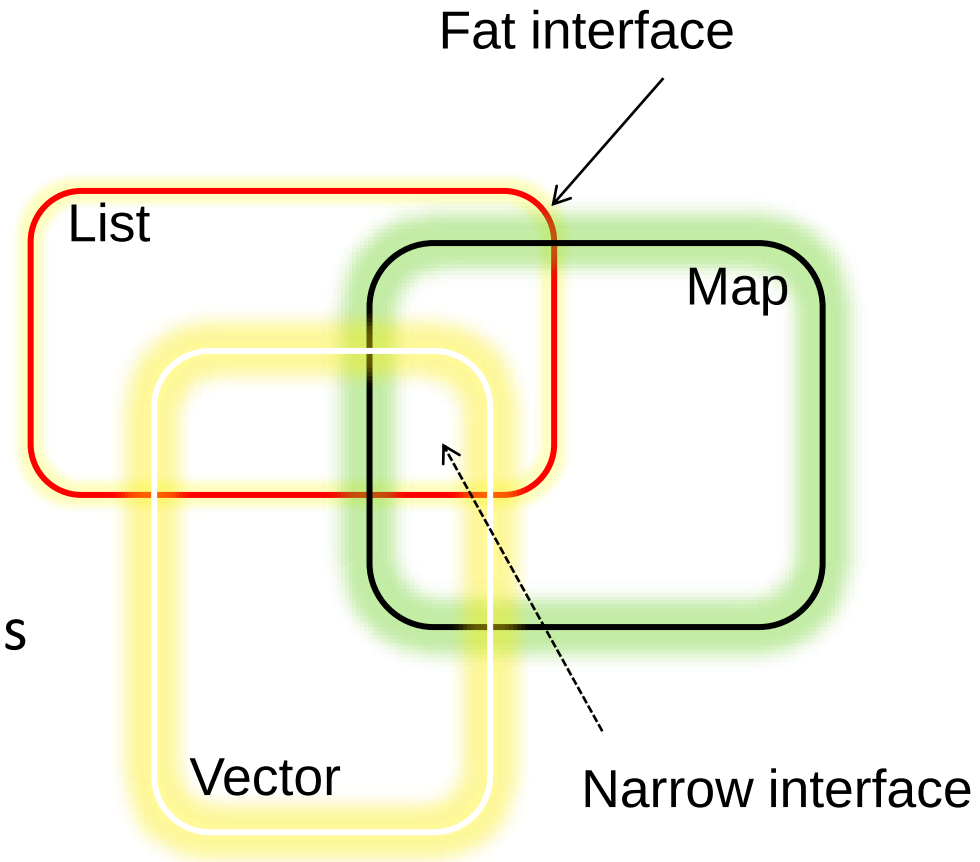


Based containers

- Intrusive container: an object has a special base class or link field to be a member of a container.

- Qt framework





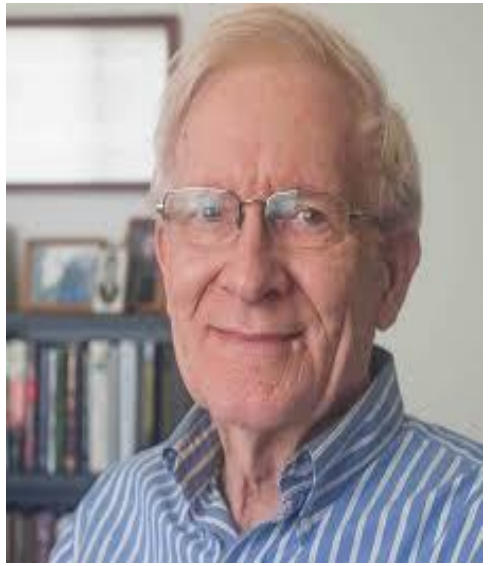
- Intersection of operations vs. Union of operations

Narrow interface

Fat interface

- C++ doesn't have a universal class Object.
  - No semantics
  - Sloppy interfaces

STL- Introduction



David Musser

Alex Stepanov

- Alex Stepanov, Meng Lee, David Musser, …
- STL: Standard Template Library
- The greatest and most important innovation in the 1998 standard was the inclusion of the STL.
- STL: A framework of algorithms and containers in the standard library
- [STL] is a phenomenal piece of work …

Sean Parent

- The C++ standard library containers were designed to meet two criteria:

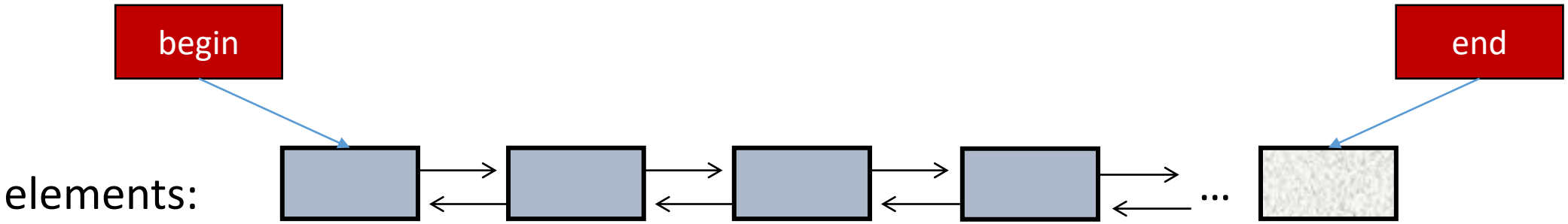
To provide the maximum freedom in the design of an individual container.

Allowing containers to present a common interface to users.

STL- basic model

- A pair of iterators define a sequence
  - –The beginning (points to the first element –if any)
  - –The end (points to the one-beyond-the-last element)
- Half-Open range: [begin, end)

Iterators:



- A *sequence* is defined by a pair of iterators defining a half-open range [begin, end)

STL- Basic model

Independent concepts should be independently represented and should be combined only when needed.

The reason that STL containers and algorithms work so well together is that they know nothing of each other.

- Alex Stepanov

Separation of concerns

- Algorithms manipulate data, but don't know about containers
- Containers store data, but don't know about algorithms
- Algorithms and containers interact through iterators

STL architecture

STL architecture

Class templates

Containers

Function templates

Generic algorithms

STL architecture

Class templates

Containers

vector, list, map, set,
multimap, multiset,
stack, queue, deque, string, …

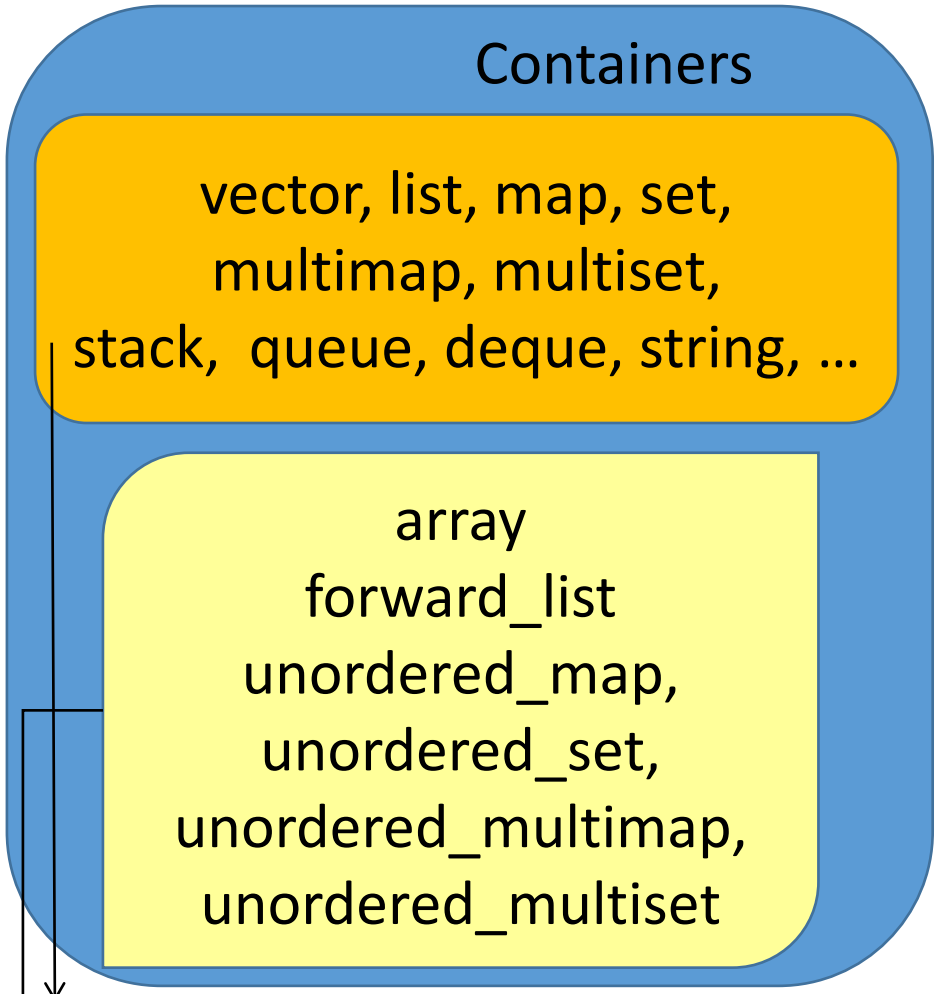
Function templates

Generic algorithms

find, find_if , for_each, copy, count,
rotate, sort, transform
unique , fill, generate, remove,
reverse, binary_search, …

STL architecture

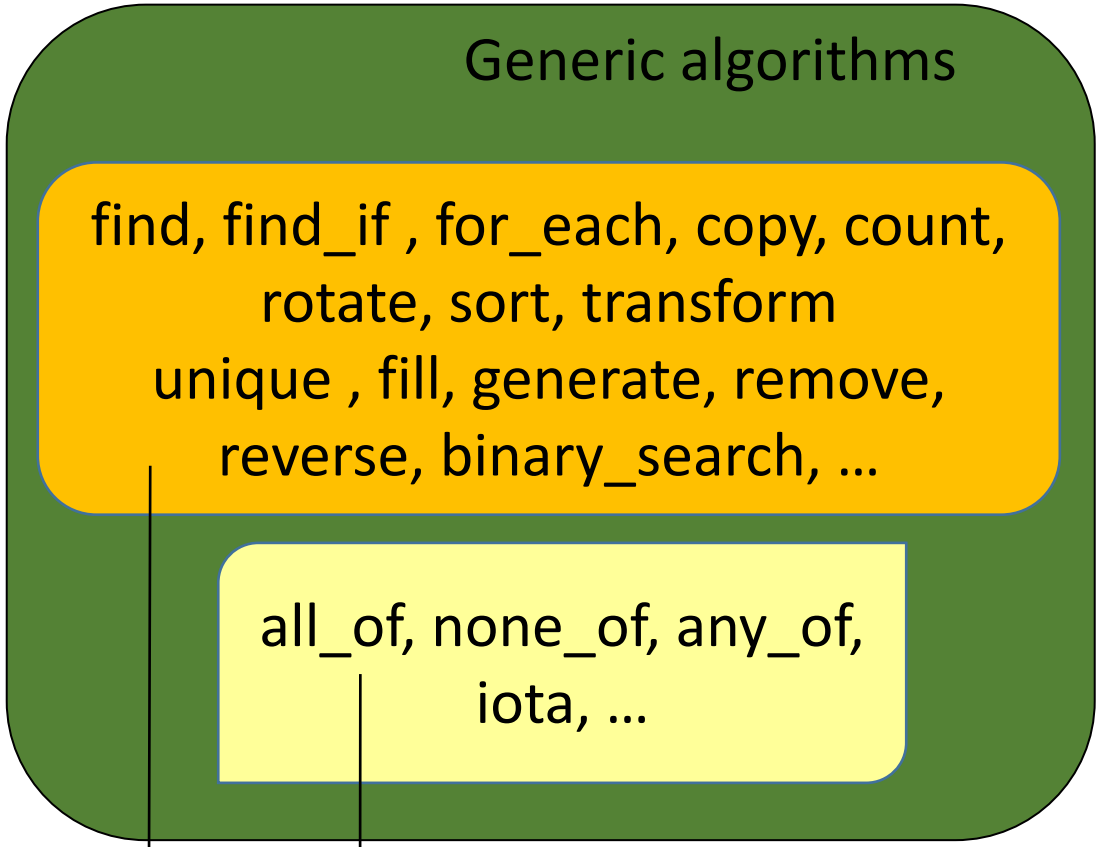
Class templates



C++98 → ~ 12 containers

C++11 → ~7 new containers

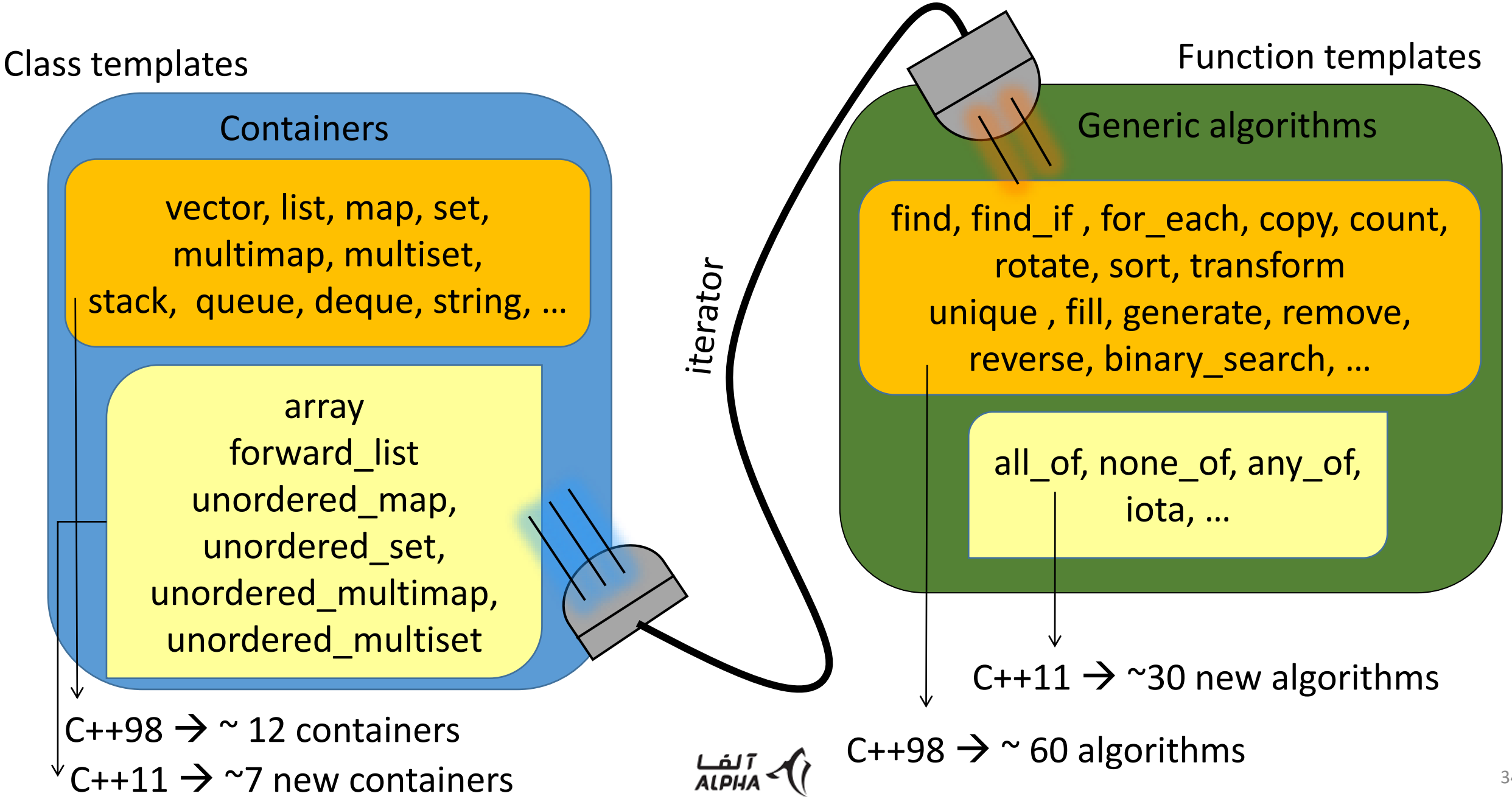
Function templates



C++11 → ~30 new algorithms

C++98 → ~ 60 algorithms

STL architecture



Container- definition

the container Classification

- A container is an object that holds other objects. from *Committee Draft*
- From internal structure point of view

Container

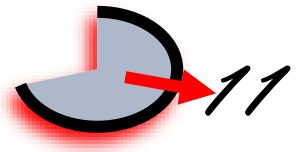
Sequence containers

vector

list

deque

forward_list



Associative containers

Ordered associative containers

map

set

multimap

multiset

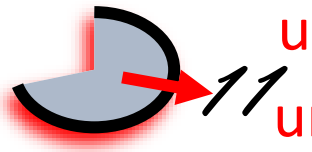
Unordered associative containers

unorederd_map

unordered_set

unordered_multimap

unordered_multiset



- Container adaptors
- Almost containers

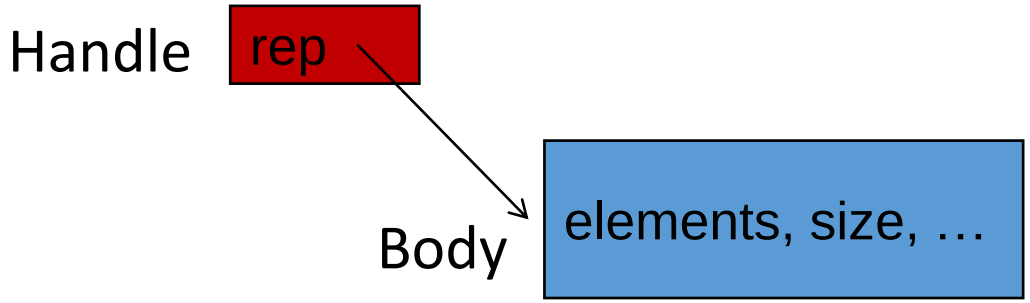
the STL Containers

- The standard containers are logically interchangeable.
- The standard doesn't prescribe a particular representation for each standard container.

Instead, the standard specifies the container interfaces and some complexity requirements.

- STL containers are resource handle → Handle-Body idiom

stack, queue, deque, priority_queue

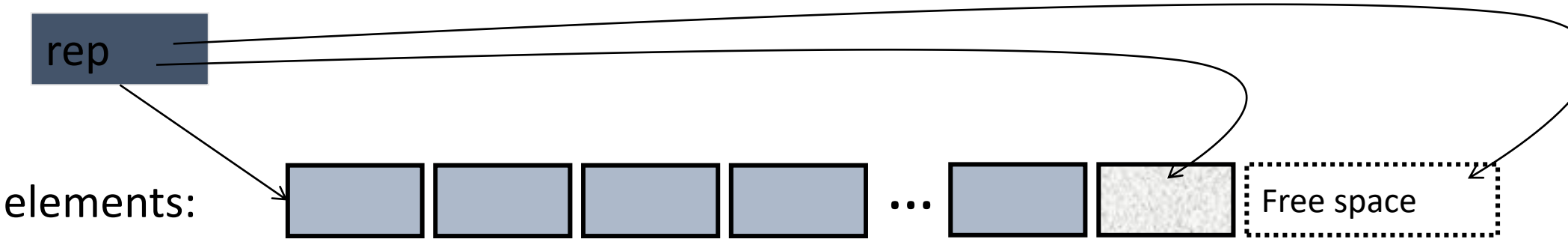


- A pair of iterators define a sequence:
  - –The beginning
  - –The end (one-beyond-the-last element) → Half-open range

- Sequence: string, array, vector, map, I/O stream, file, (doubly-linked) list, stack, queue, set, (various) hash tables, singly-linked list, …

Vector

- vector implementation: array of elements

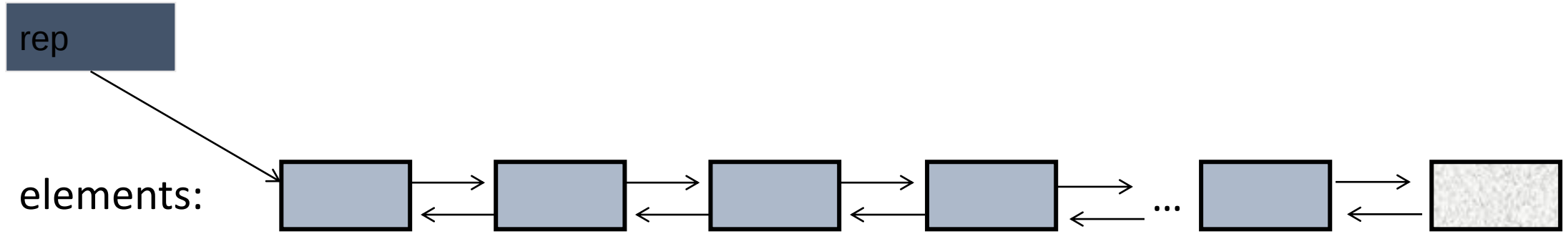


- Header file: <vector>

```
template<class T> class vector {
    // representation
    T* elem; // start of allocation
    T* space; // end of element sequence
    T* last; // end of allocated space
    // ...
};
```

List

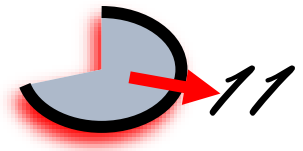
- A *list* is most likely represented by a sequence of links pointing to the elements and the number of elements.



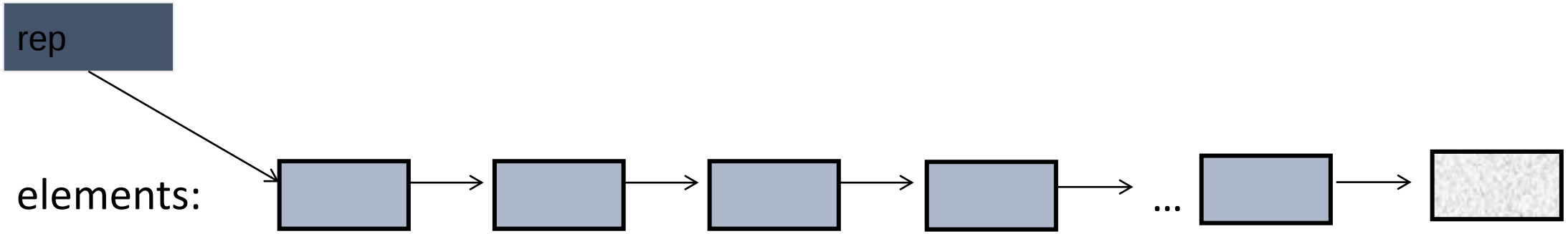
- One implementation:
- Header file: <list>

```
template<class T> class list {
private: // representation
     struct Node {
          Node* next;
          Node* prev;
          T t; // sattelite data
     };
     // ...
};
```

Forward lists



- A *forward_list* is most likely represented by a sequence of links pointing to the elements.



- One possible implementation:

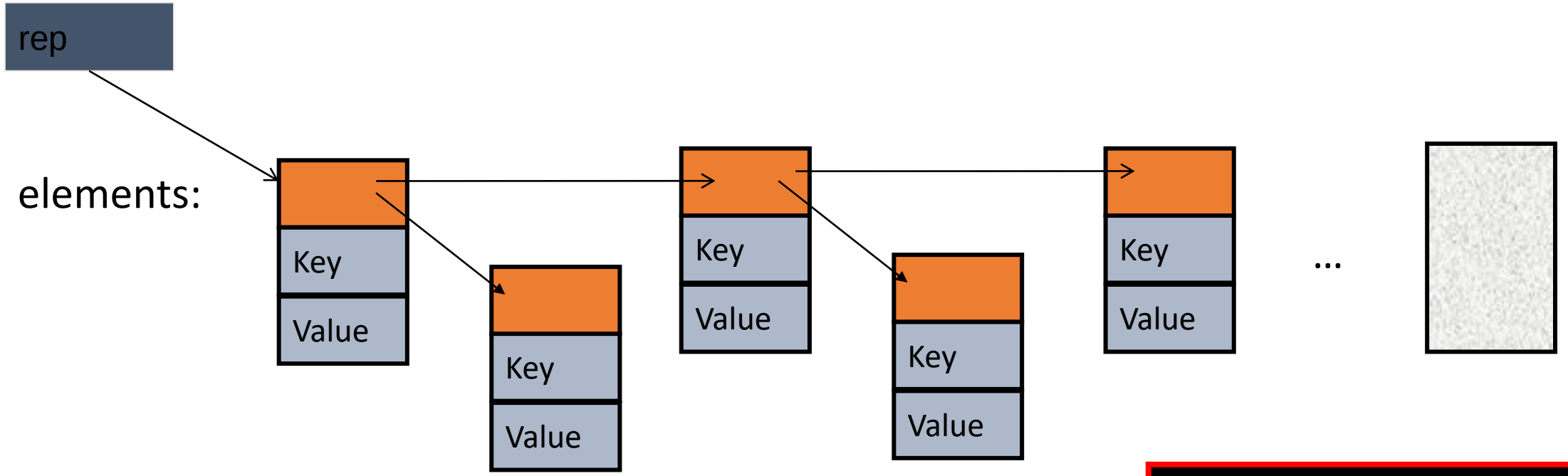
```
template<class T> class forward_list {
private: // representation
     struct Node {
          Node* next;
          T t; // satellite data
     };
     // ...
};
```

- Header file: <forward_list>

- A forward_list (a singly-linked list) is basically a list optimized for empty and very short lists.
- An empty forward_list takes up only one word.

Map

- A *map* is most likely implemented as a (balanced) tree of nodes pointing to (key, value) pairs:

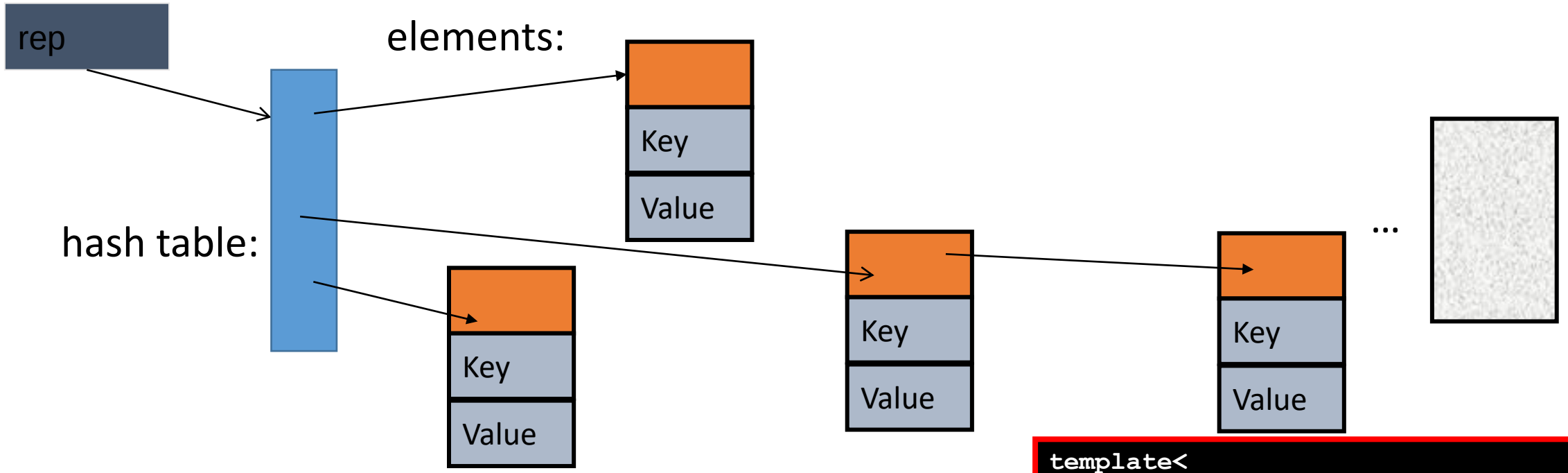


  - One possible implementation: Red/Black tree
- Header file: <map>: map and multimap containers

```
template<
    class Key,
    class T,
    class Compare = less<Key>>
class map {
    // representation
    // balanced binary tree
};
```

Unordered_map

- An unordered_*map* is most likely implemented as a hash table:

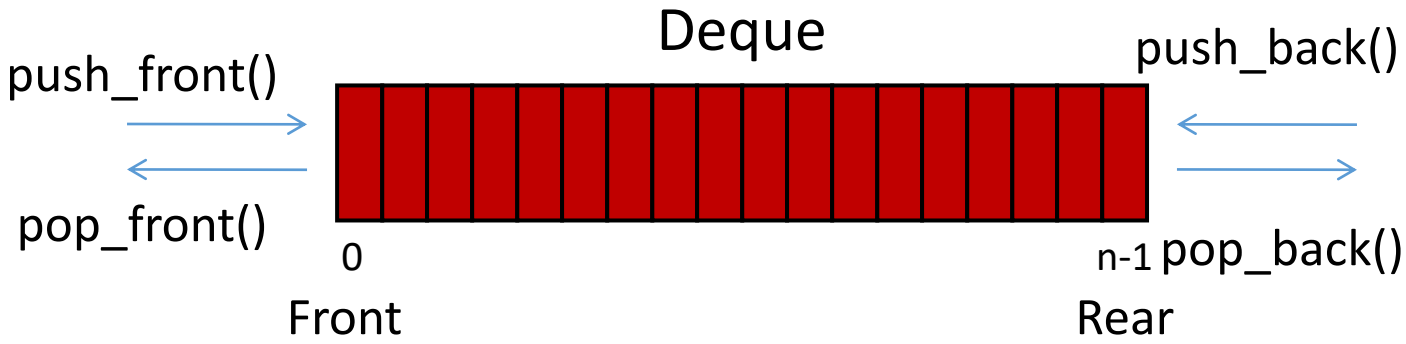


- One possible implementation: array of buckets
- Header file: <unordered_map>: unordered_map and unordered_multimap containers
- By default, an unordered_map<X> uses hash<X> for hashing and equal_to<X> to compare keys

```
template<
    class Key,
    class T,
    class Hash = hash<Key>,
    class Pred = std::equal_to<Key>>
class unordered_map {
    // representation
    // array of buckets
};
```

Deque

- Deque stands for double-ended queue.
- A *deque* is a sequence container that, like a vector, supports random access iterators. In addition, it supports constant time insert and erase operations at the beginning or the end; insert and erase in the middle take linear time. That is, a deque is especially optimized for pushing and popping elements at the beginning and end.

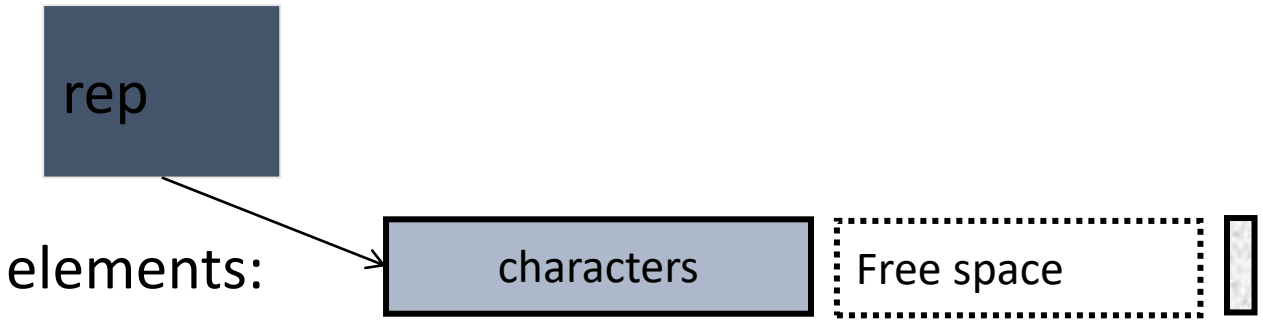


the container classification: Almost containers

- built-in arrays, string, array, bitset

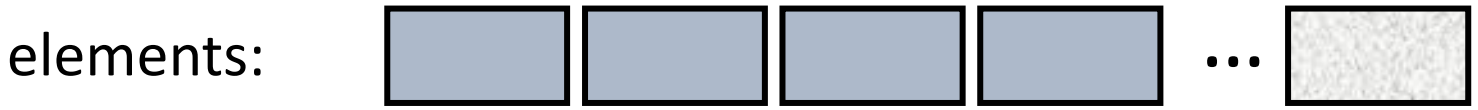
String

- A string might be implemented:
  - For short strings that characters are stored in the string handle itself, and
  - For longer string s the elements are stored contiguously on the free-store (like vector elements)



Array

- Like a built-in array, an array is simply a sequence of elements, with no handle:



- This implies that a local array does not use any free store.

- <array>

Compile-time array size

```
template <class T, size_t N> struct array;
```

Vector as the Default container

Vector as the Default container

Use vector as the default container.
- Alexander Stepanov

Vector as the Default container

Use vector as the default container.
- Alexander Stepanov

If you understand int and vector then you understand C++: the rest is details (there are a lot of details).
- Bjarne Stroustrup

Vector as the Default container

Use vector as the default container.
- Alexander Stepanov

Why?

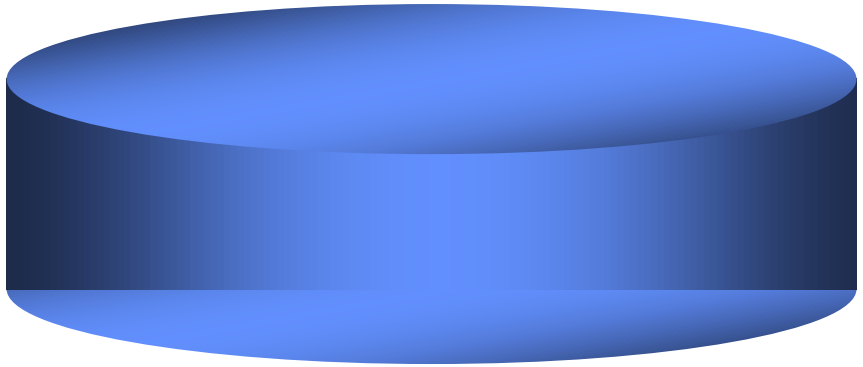
If you understand int and vector then you understand C++: the rest is details (there are a lot of details).
- Bjarne Stroustrup

Vector as a contiguous data structure

- A vector and similar contiguously allocated data structures included string and array has three major advantages compared to other data structures:
  - The elements of vector are compactly stored:
    Amount of memory for a vector of X = Size of (vector<X>) + vector size * sizeof(X)
  - Traversal of a vector is very fast:
    Modern hardware: consecutive access
    find(), copy(), and other forward iterator-related generic algorithms are close to optimal.
  - Vector supports simple and efficient random access
    sort(), binary_search() and other random-access iterator-related generic algorithms are efficient.
- Vector vs. list

Memory hierarchy- different kind of memories

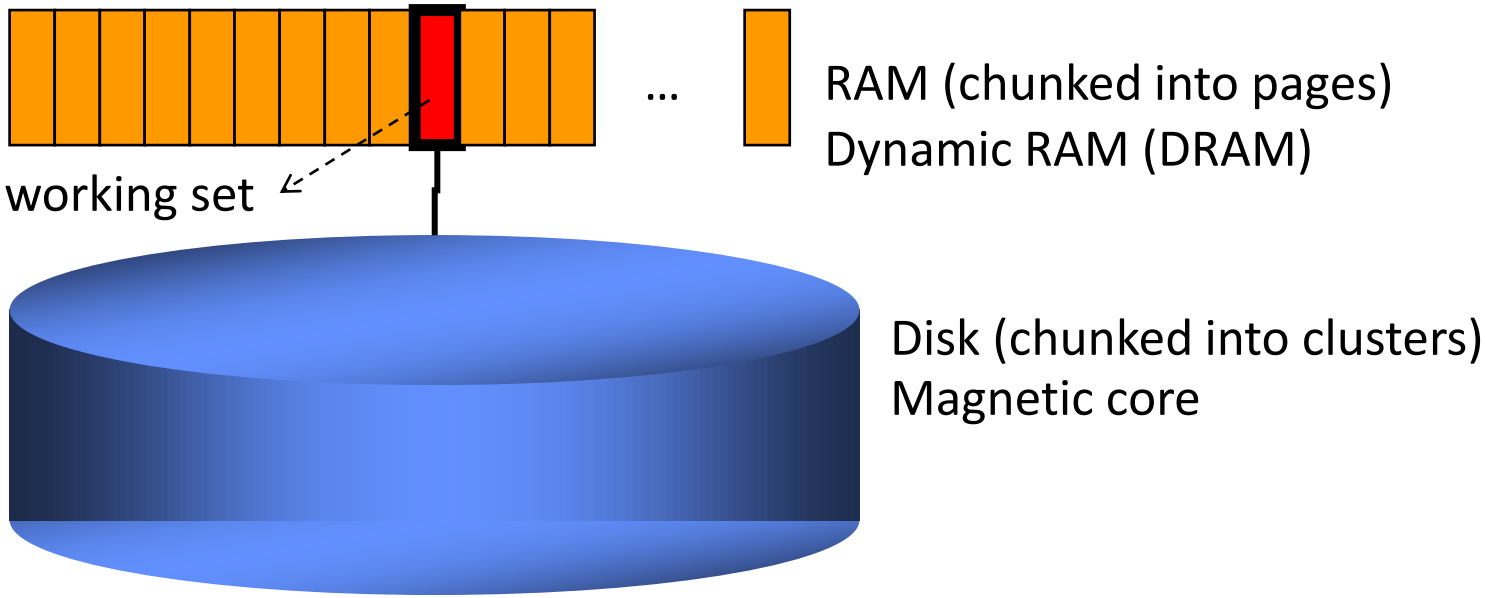
Memory hierarchy- different kind of memories



Disk (chunked into clusters)
Magnetic core

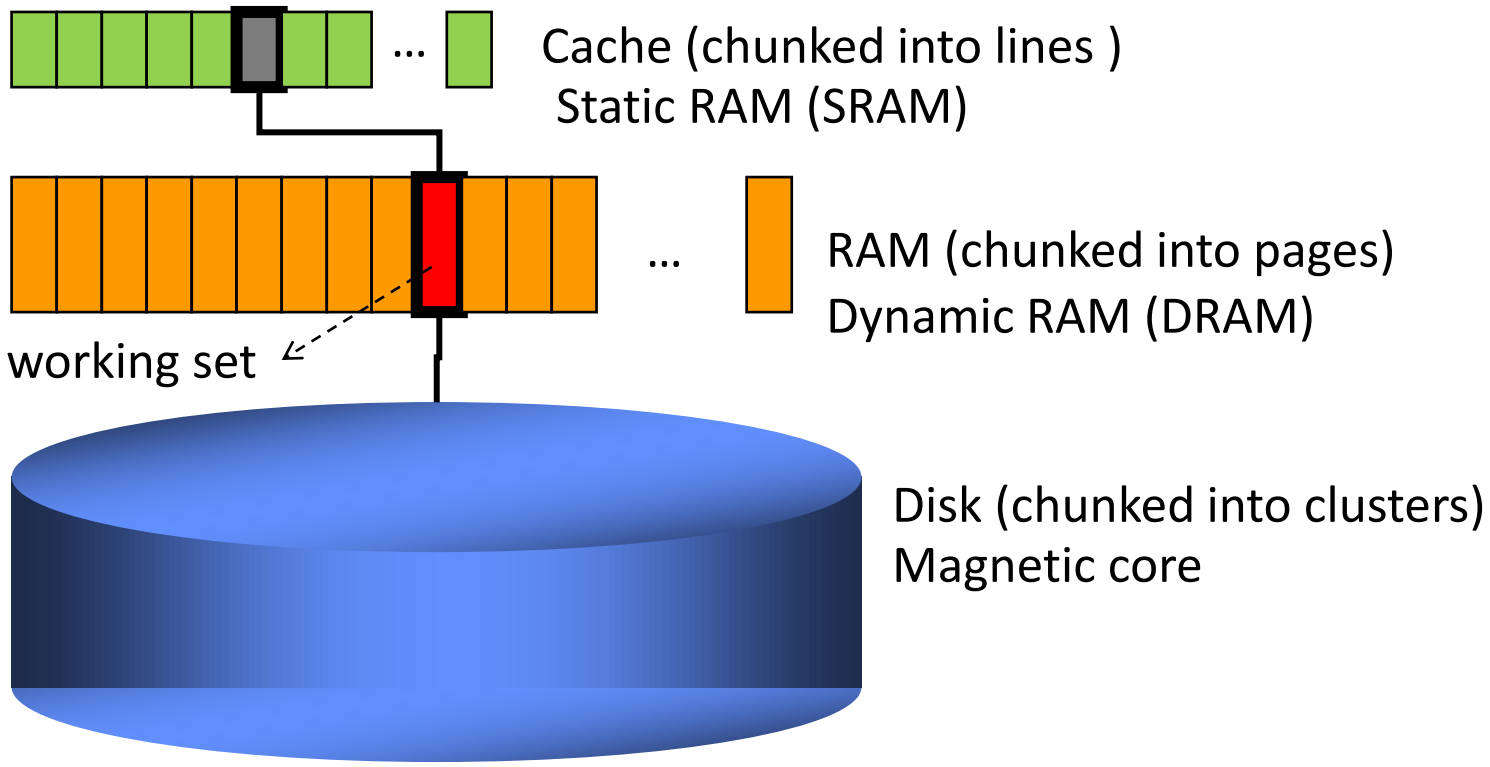
the picture adapted from Herb Sutter. Maximize Locality, Minimize Contention. Dr. Dobb's Journal, May 2008.
http://www.drdobbs.com/parallel/maximize-locality-minimize-contention/208200273

Memory hierarchy- different kind of memories



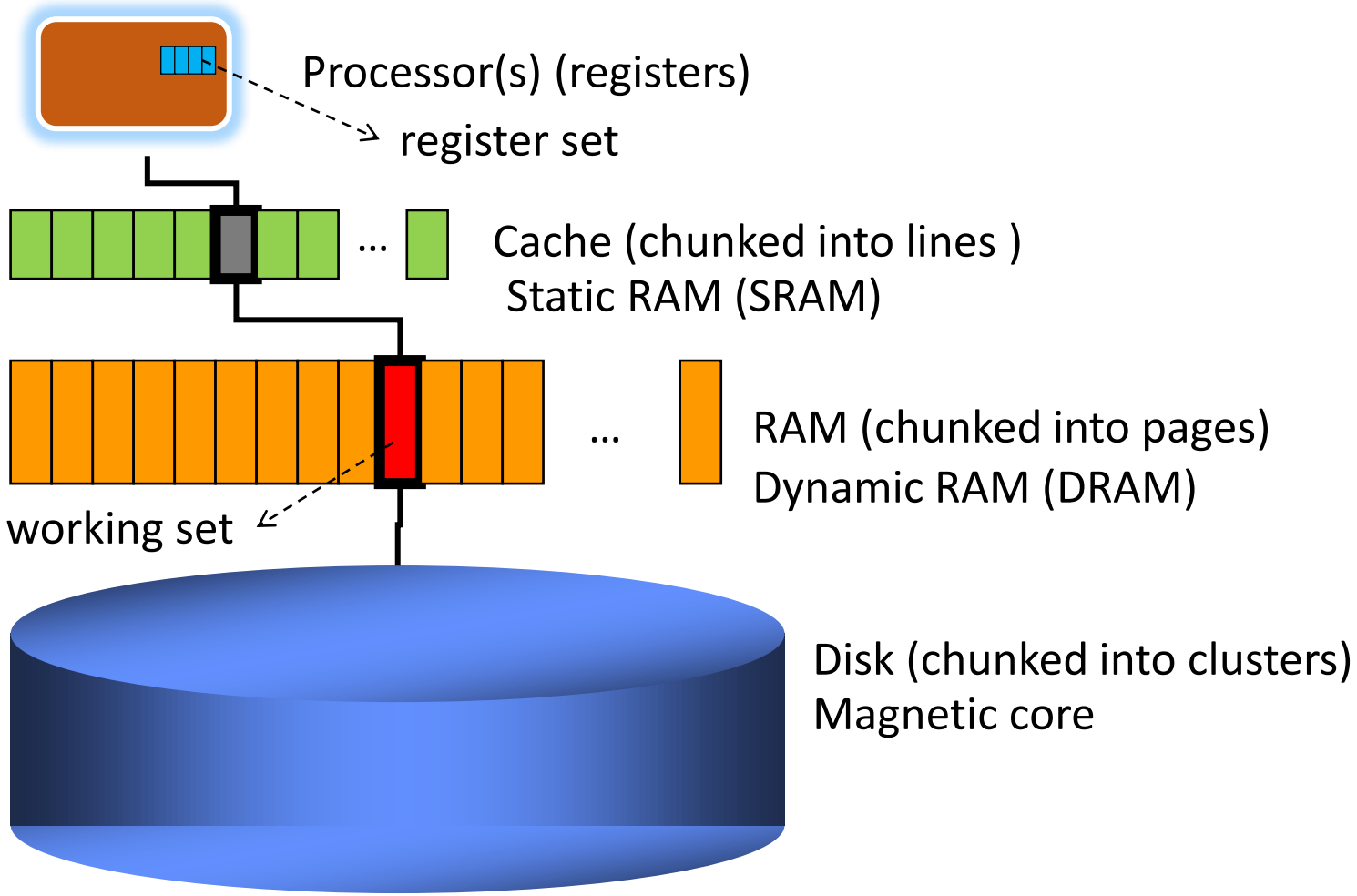
the picture adapted from Herb Sutter. Maximize Locality, Minimize Contention. Dr. Dobb's Journal, May 2008.
http://www.drdobbs.com/parallel/maximize-locality-minimize-contention/208200273

Memory hierarchy- different kind of memories



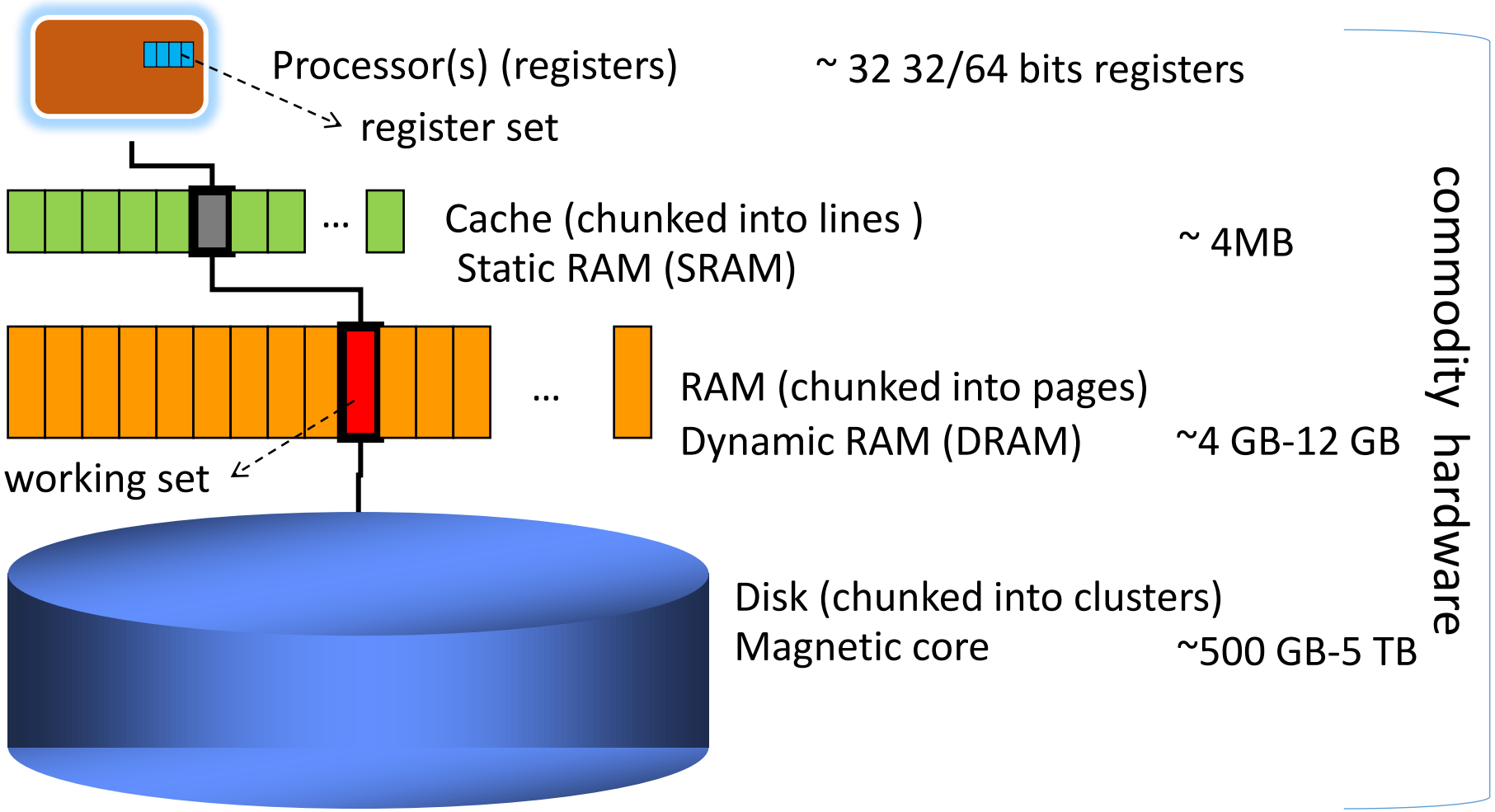
the picture adapted from Herb Sutter. Maximize Locality, Minimize Contention. Dr. Dobb's Journal, May 2008.
http://www.drdobbs.com/parallel/maximize-locality-minimize-contention/208200273

Memory hierarchy- different kind of memories



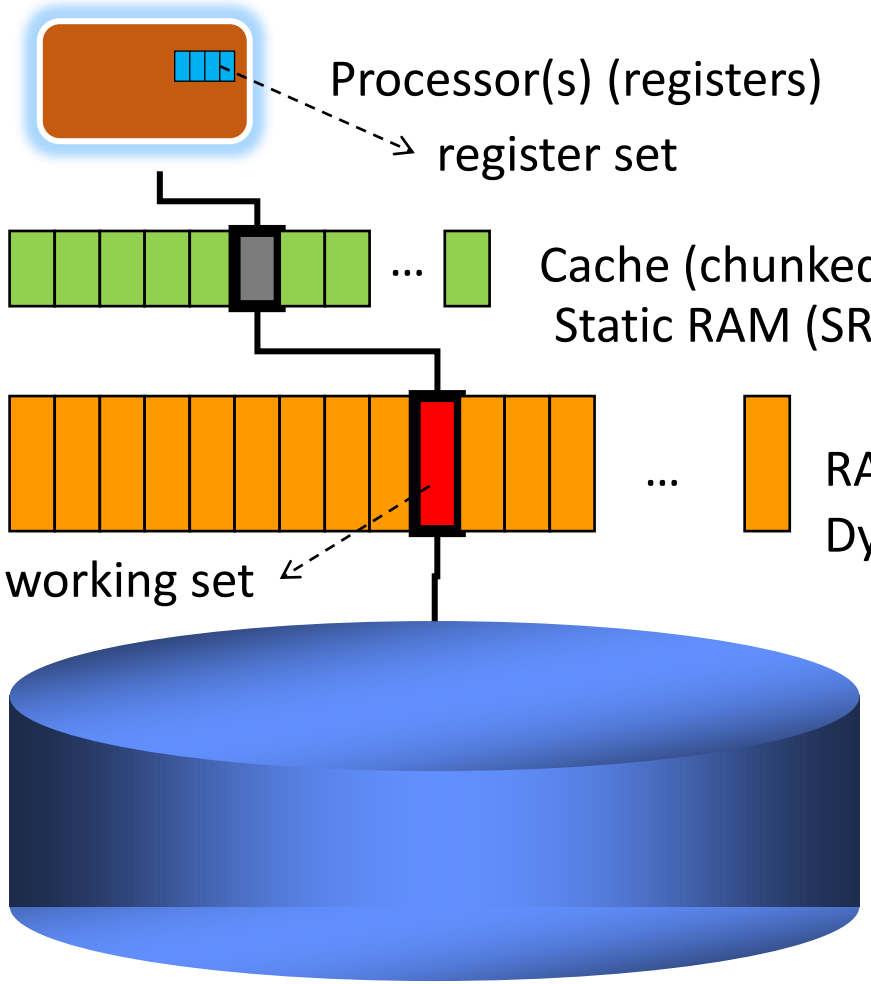
the picture adapted from Herb Sutter. Maximize Locality, Minimize Contention. Dr. Dobb's Journal, May 2008.
http://www.drdobbs.com/parallel/maximize-locality-minimize-contention/208200273

Memory hierarchy- different kind of memories



the picture adapted from Herb Sutter. Maximize Locality, Minimize Contention. Dr. Dobb's Journal, May 2008.
http://www.drdobbs.com/parallel/maximize-locality-minimize-contention/208200273

Memory hierarchy- different kind of memories



Processor(s) (registers) ~ 32 32/64 bits registers

register set

Cache (chunked into lines) ~ 4MB

Static RAM (SRAM)

RAM (chunked into pages)

Dynamic RAM (DRAM) ~4 GB-12 GB

working set

Disk (chunked into clusters)

Magnetic core ~500 GB-5 TB

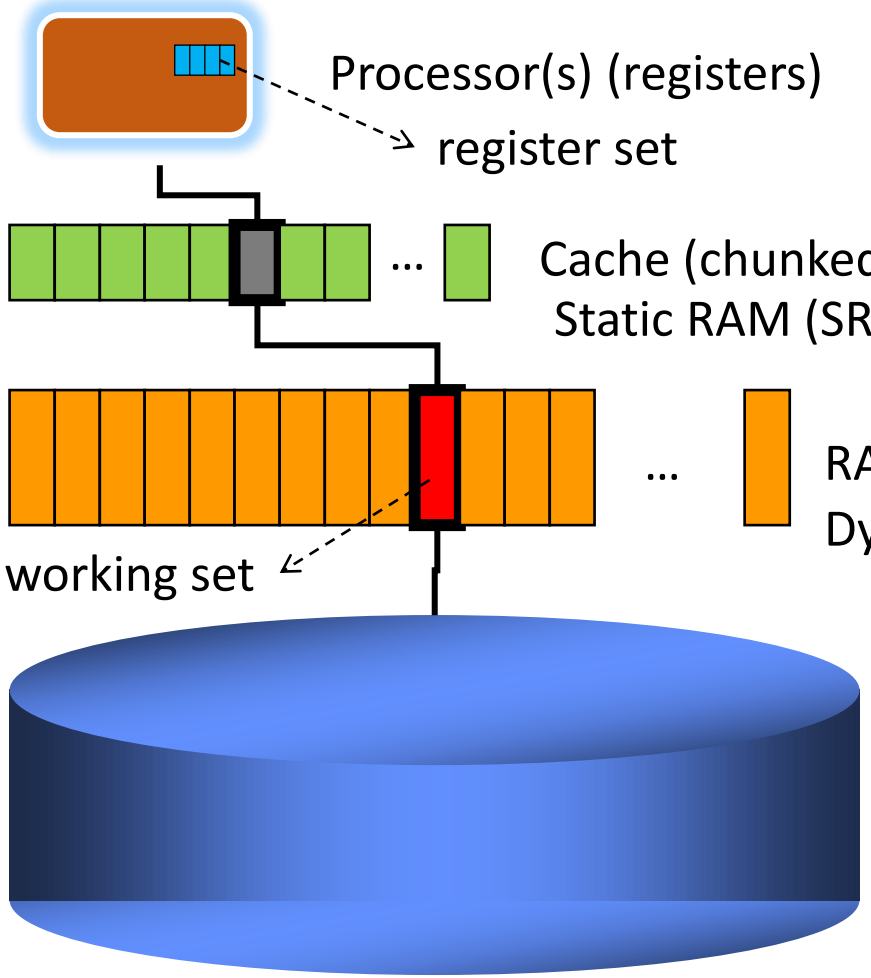
commodity hardware

- Jeff Dean (Google)

Hierarchical Memory Structure		
Register Access	0.1	ns
L1 Cache	0.5	ns
Branch Mis-preditct	5	ns
L2 Cache	7.0	ns
Mutex Lock/Unlock	25	ns
Memory	100.0	ns

the picture adapted from Herb Sutter. Maximize Locality, Minimize Contention. Dr. Dobb's Journal, May 2008.
http://www.drdobbs.com/parallel/maximize-locality-minimize-contention/208200273

Memory hierarchy- different kind of memories



~ 32 32/64 bits registers

~ 4MB

~4 GB-12 GB

~500 GB-5 TB

commodity hardware

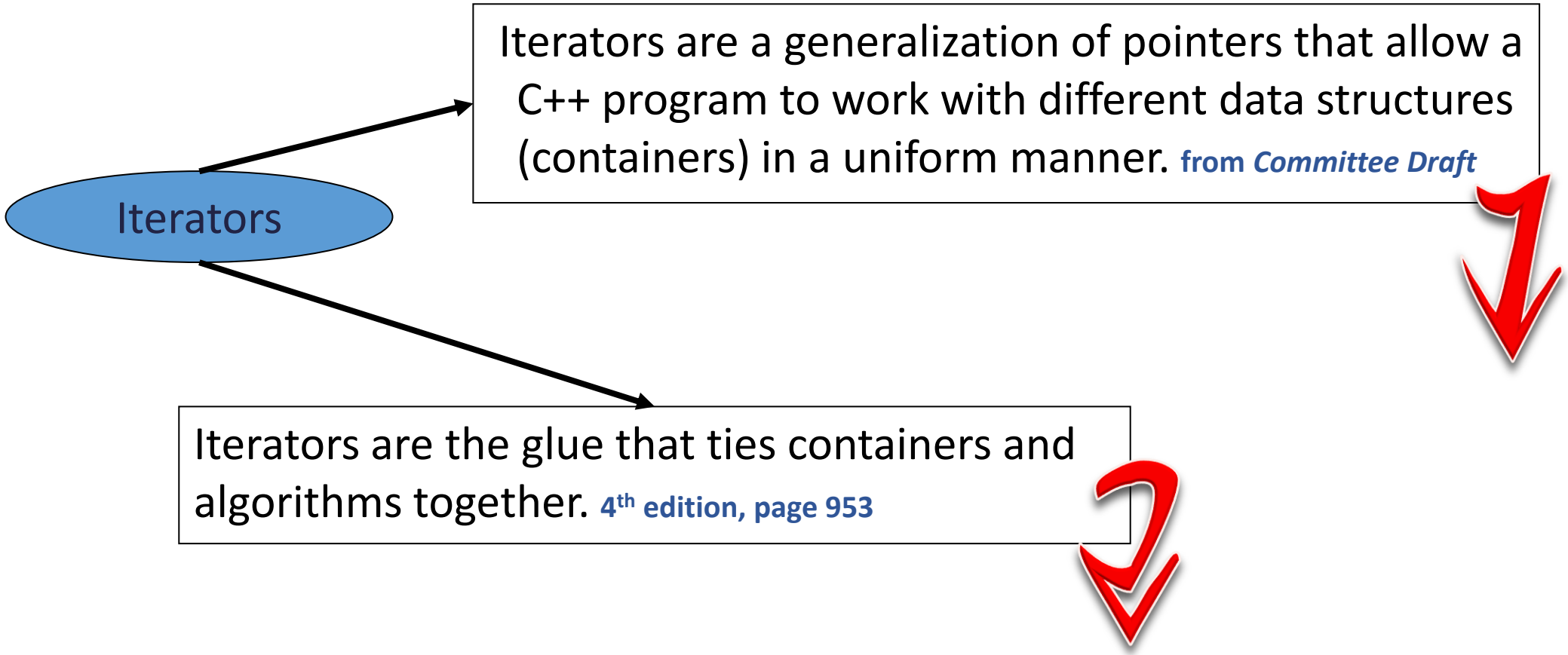
- Jeff Dean (Google)

Hierarchical Memory Structure		
Register Access	0.1	ns
L1 Cache	0.5	ns
Branch Mis-preditct	5	ns
L2 Cache	7.0	ns
Mutex Lock/Unlock	25	ns
Memory	100.0	ns

- Predictable memory access
- Cache misses

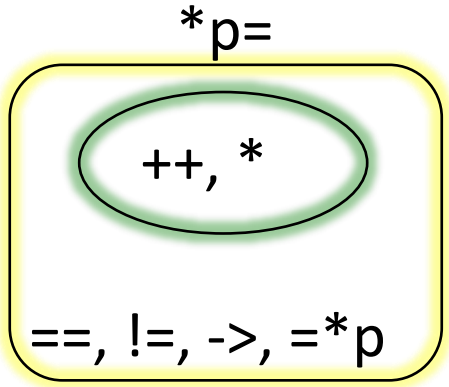
- In infrastructure software development, compactness and predictable access patterns are essential for efficiency.

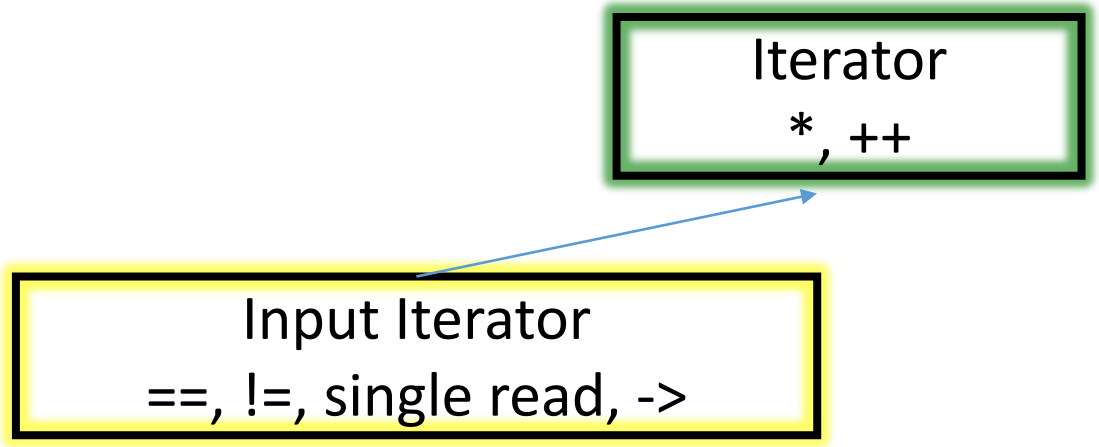
Iterators: definition



- Each kind of container provides its own iterators that support a standard set of iterator operations.

Iterator categories



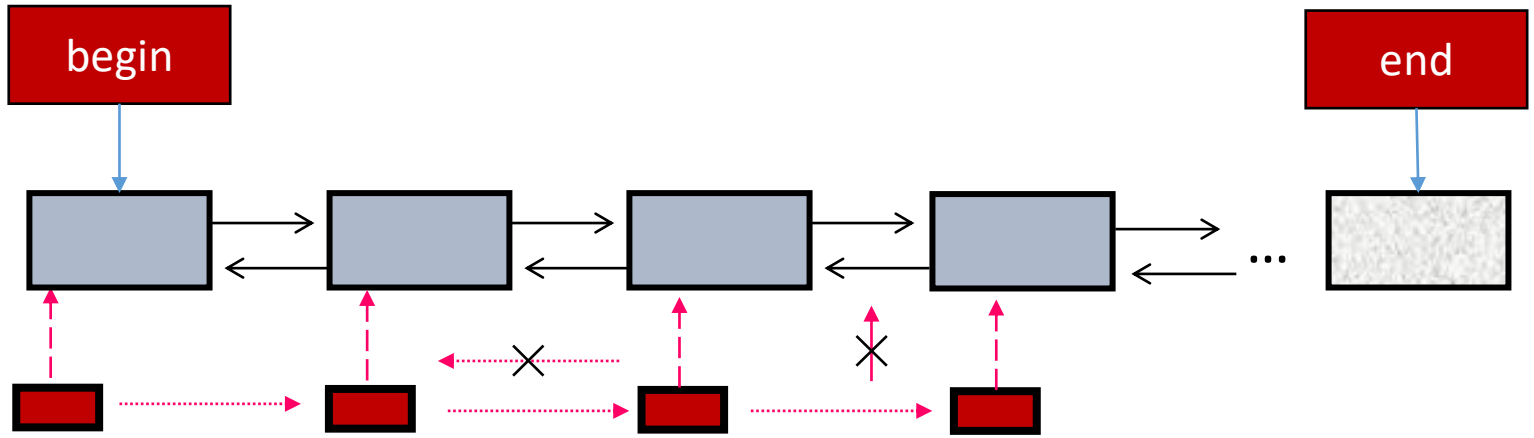


Pointer-related operators

- Single read: RHS = *p
- Single write: LHS *p =
- Access : ->

Input iterator: details

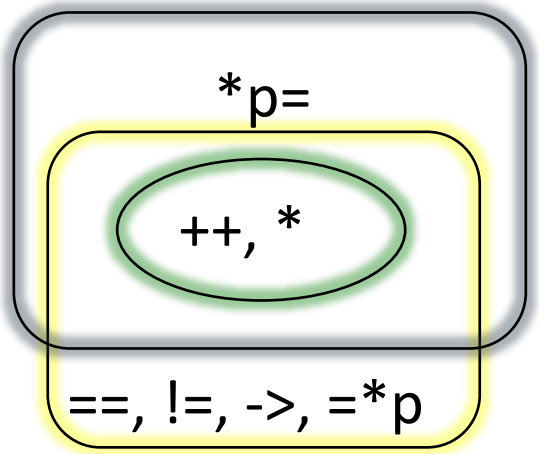
- Input iterator: read-only
  - Algorithms: find, count, equal, …
  - Container: istream

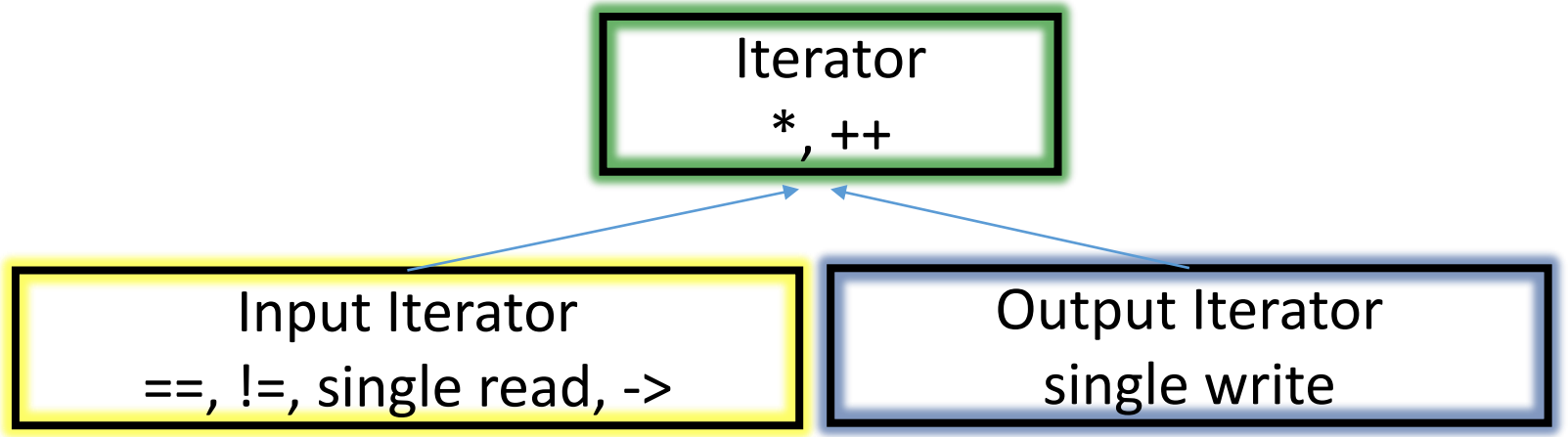



```
template<class InputIter, class T>
InputIterator find(InputIter first, InputIter last, const T& v)
{
  for (; first != last && *first != v; ++first) ;
  return first;
}

void find_user()
{
    list<int> L{ 2, 3, 10 };
    list<int>::iterator it = find(L.begin(), L.end(), 10);
    std::string a = "Make as simple as possible, but no simpler.";
    string::iterator it2 = find(a.begin(), a.end(), 'k');
    int a[10];
    int* p = find(a, a + 10, '0');
```

Iterator categories



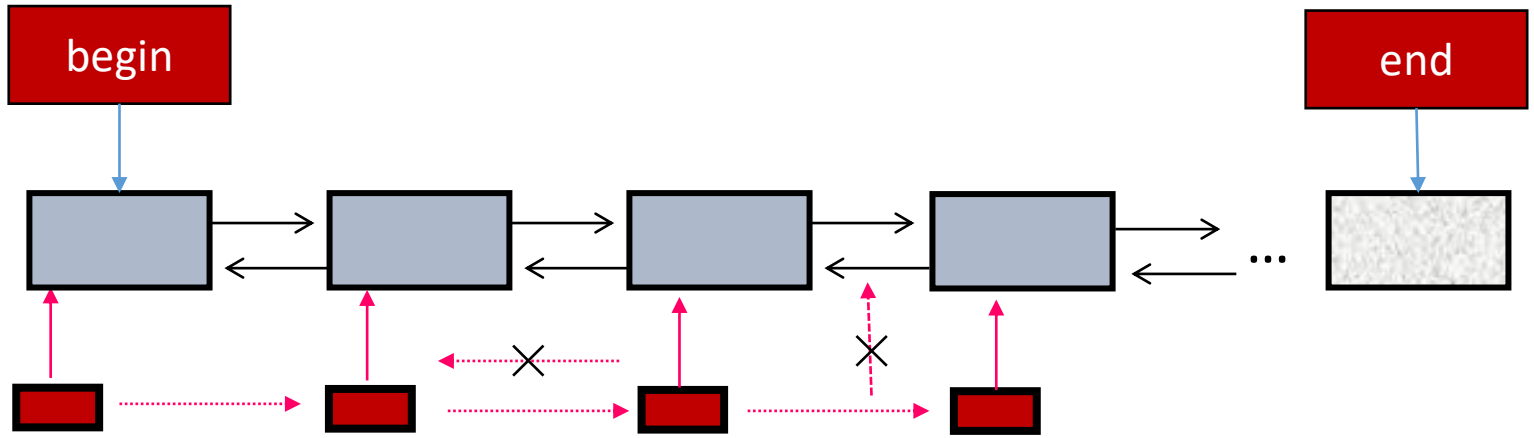


Pointer-related operators

- Single read: RHS = *p
- Single write: LHS *p =
- Access : ->

Output iterator: details

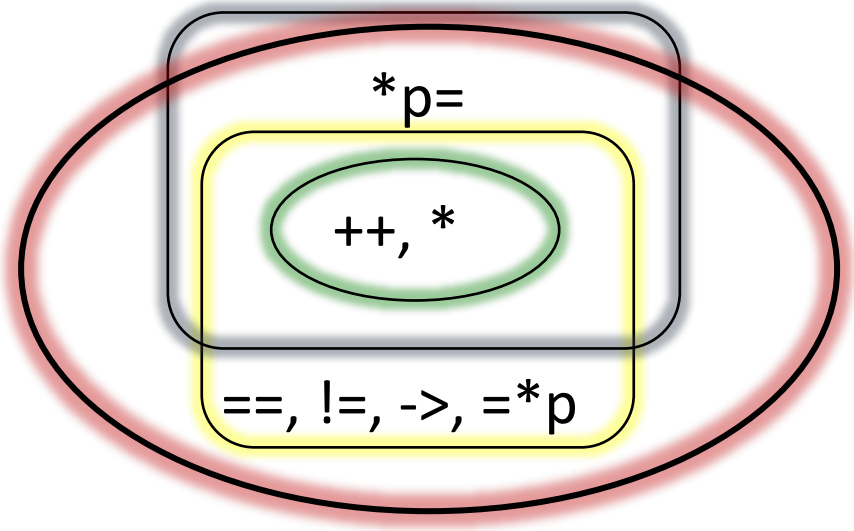
- Output iterator: write-only
  - Algorithms: copy
  - Container: ostream

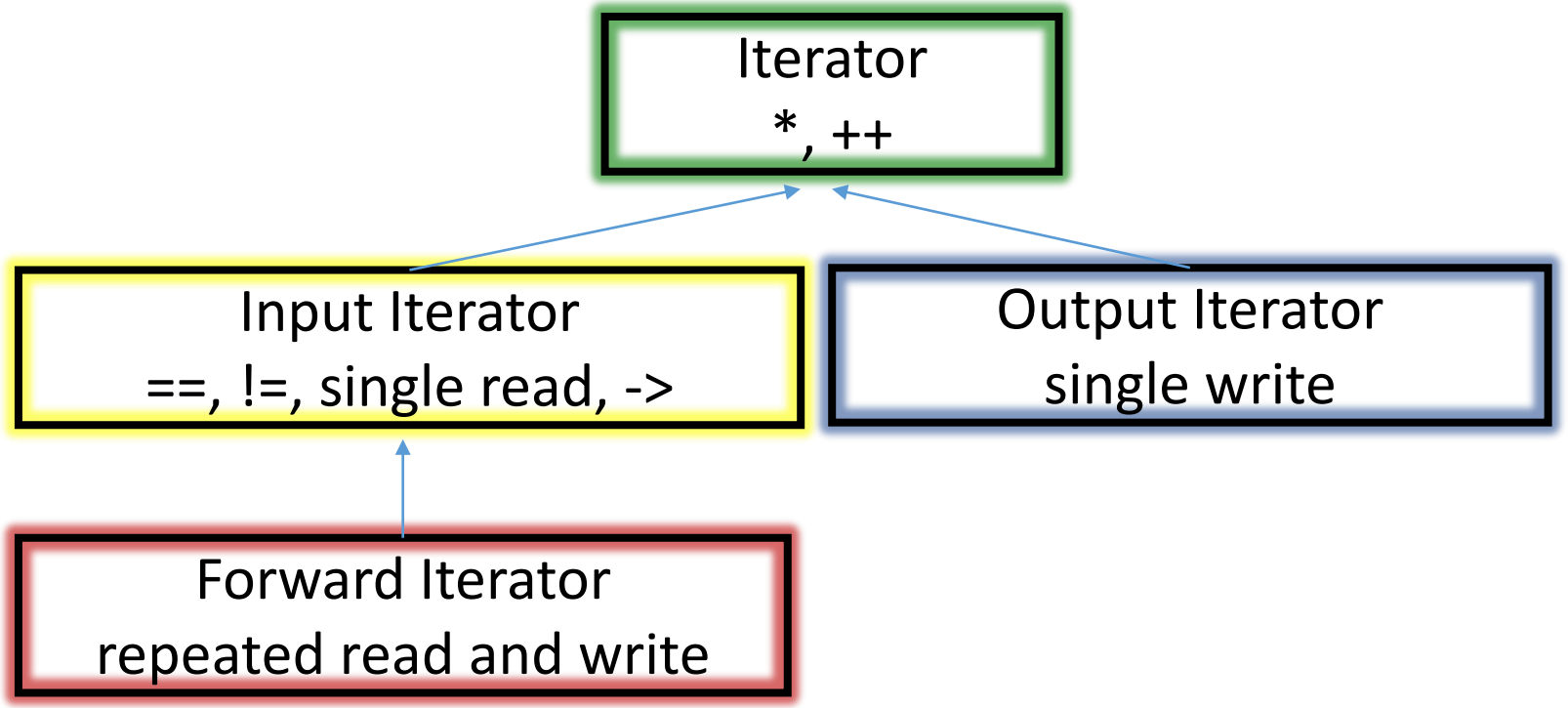



```
template<class InputIter, class OutputIter>
OutputIter copy(InputIter first, InputIter last, OutputIter result)
{
  for (; first != last; ++result, ++first)
    *result = *first
  return result;
}

void copy_user()
{
    list<int> L{ 2, 3, 10 );
    forward_list<int> FL(3);
    copy(L.begin(), L.end(), FL.begin());
}
```

Iterator categories



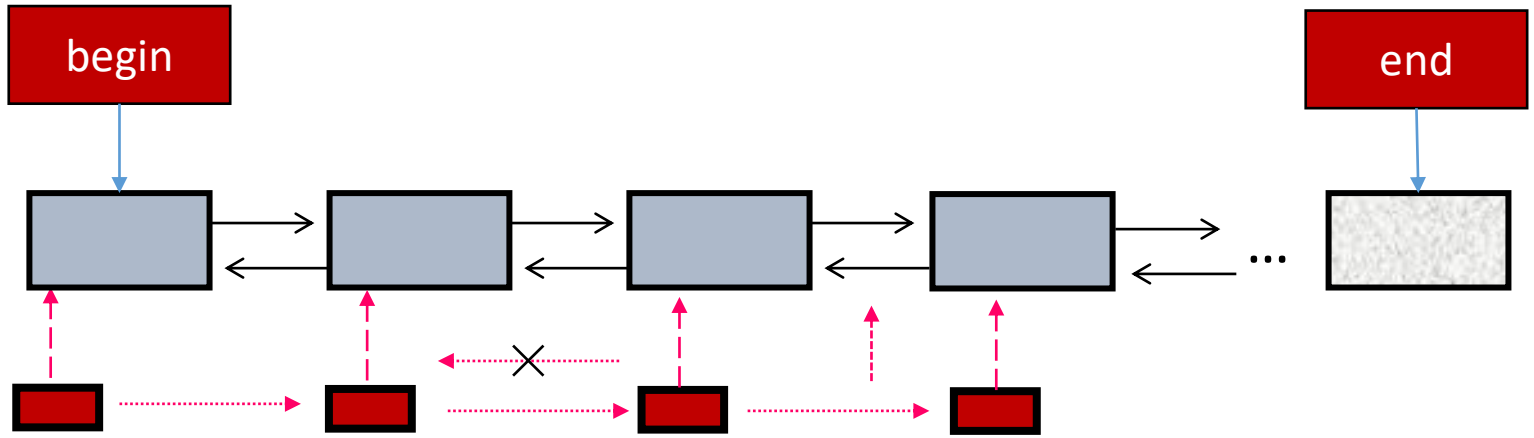


Pointer-related operators

- Single read: RHS = *p
- Single write: LHS *p =
- Access : ->

Forward iterator: details

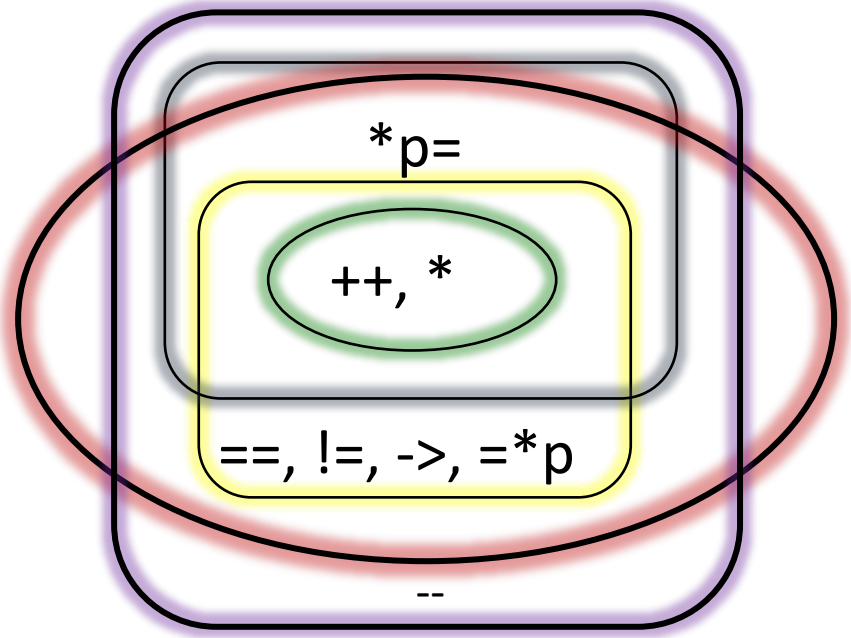
- Forward iterator: Read and Write
  - Algorithms: replace
  - Container: forward_list

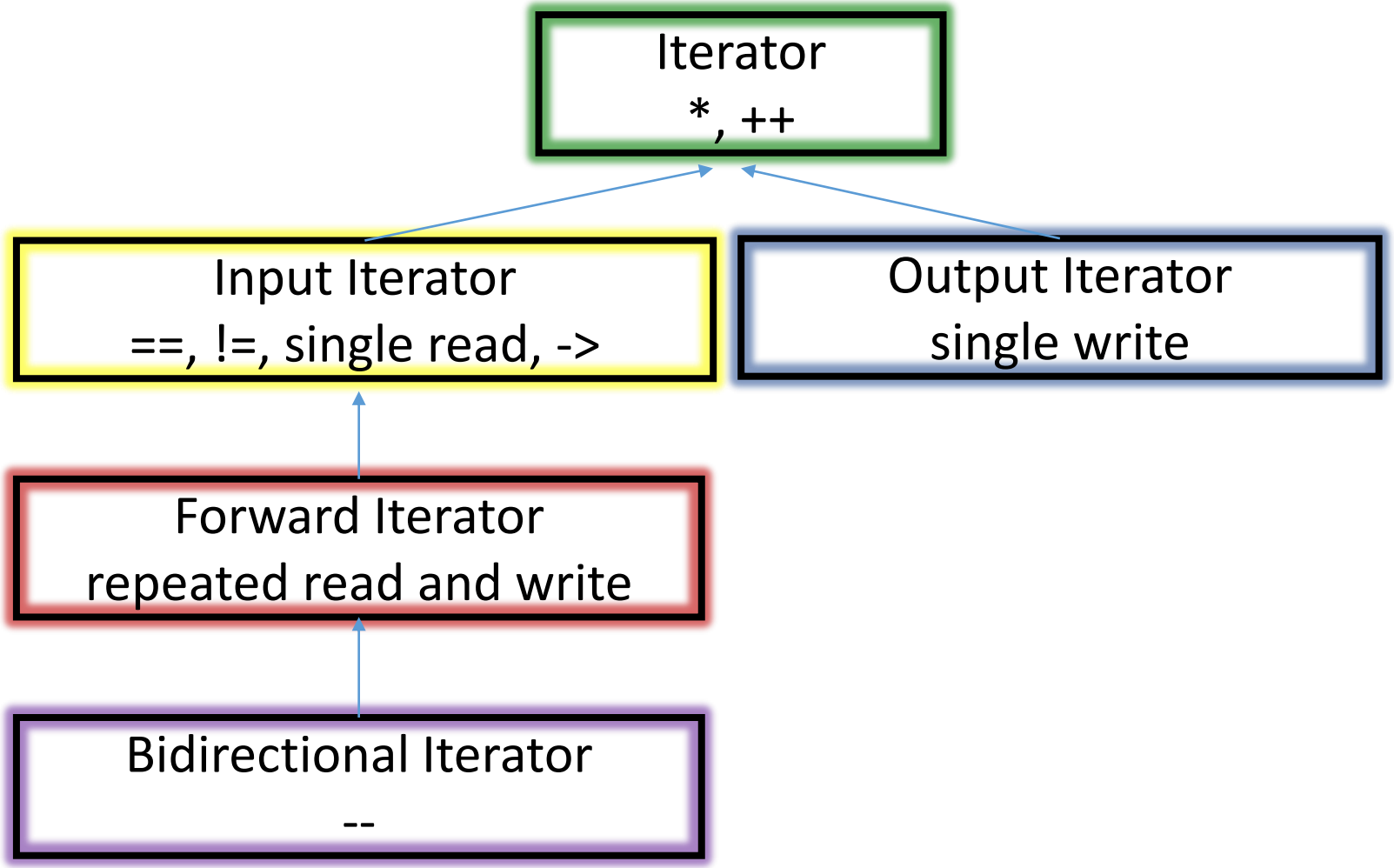



```
template<class ForwardIter, class T>
void replace(ForwardIter first, InputIter last, const T& old_value, const T& new_value)
{
  for (; first != last; ++first) {
    if (*first == old_value)
      *first = new_value;
  }
}

void replace_user()
{
    list<int> L{ 2, 3, 10 };
    vector<int> v(3);
    replace(L.begin(), L.end(), 3, 10);
}
```

Iterator categories



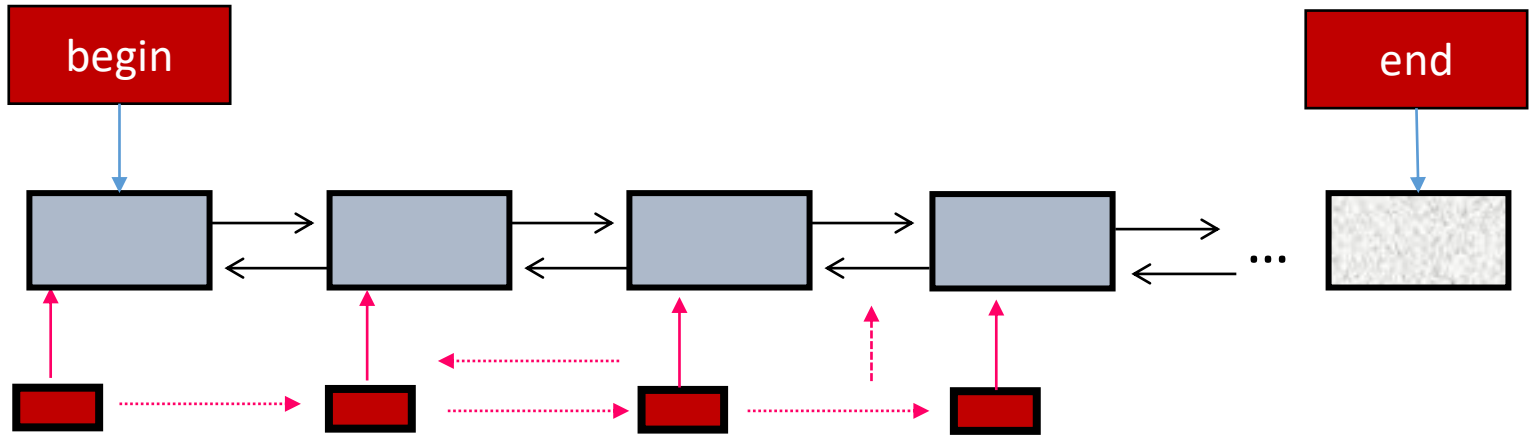


Pointer-related operators

- Single read: RHS = *p
- Single write: LHS *p =
- Access : ->

Bidirectional iterator: details

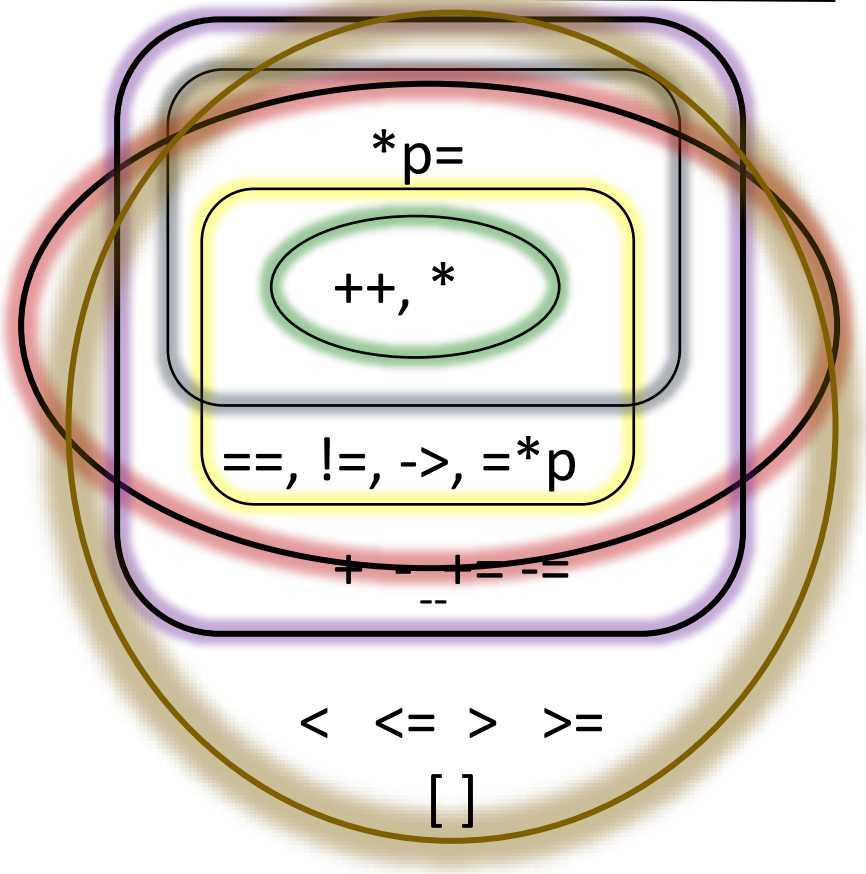
- Bidirectional iterator: Bidirectional read/write
  - Algorithms: reverse_copy
  - Container: list

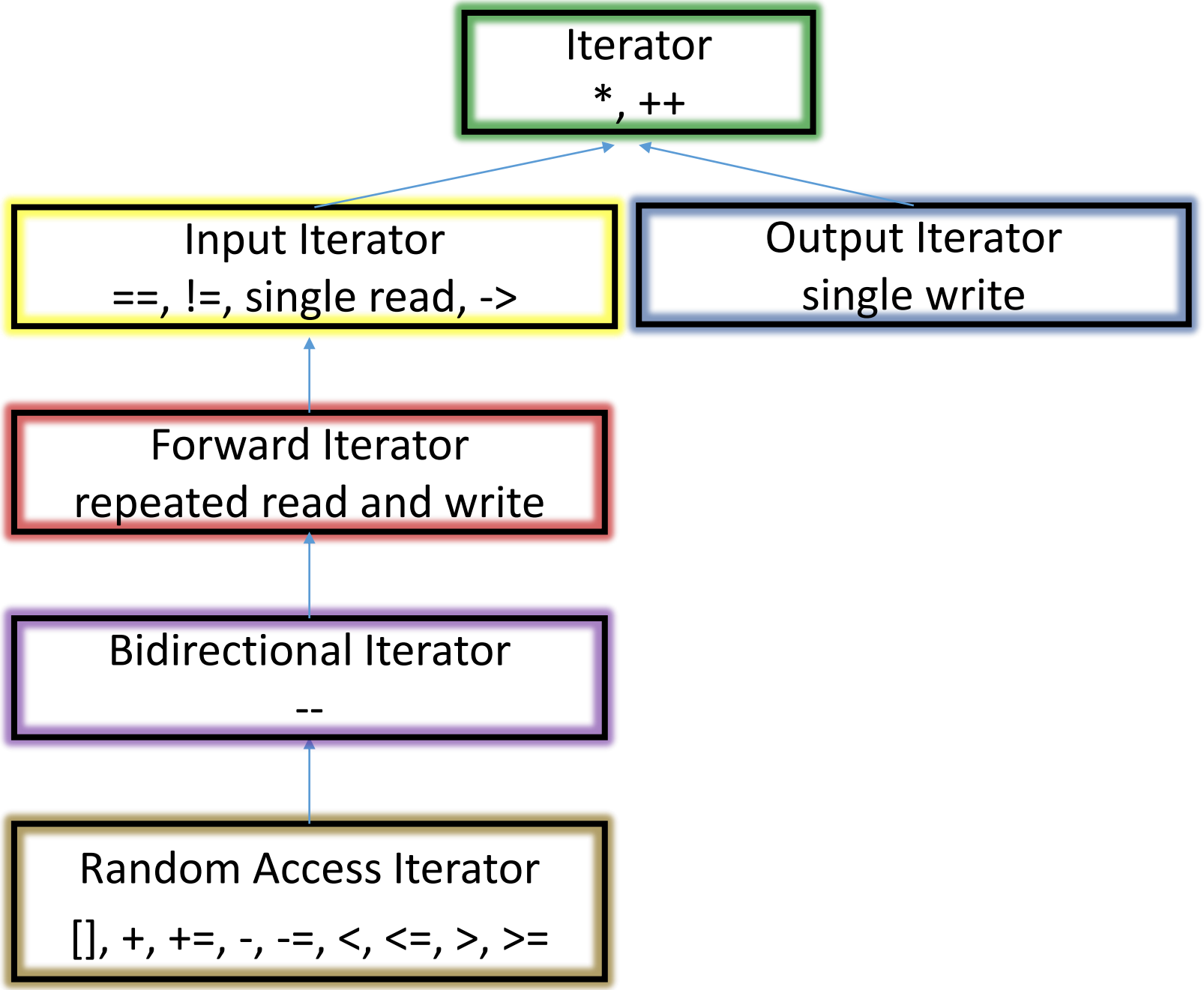



```
template<class BidirectionalIter, class OutputIter>
OutputIter reverse_copy(BidirectionalIter first, BidirectionalIter last, OutputIter result)
{
    while (first != last) {
        --last;
        *result = *first;
        ++result;
    }
    return result;
}

void reverse_copy_user()
{
    list<int> L{ 2, 3, 10 );
    vector<int> v(3);
    replace_copy(L.begin(), L.end(), v.begin());
}
```

Iterator categories



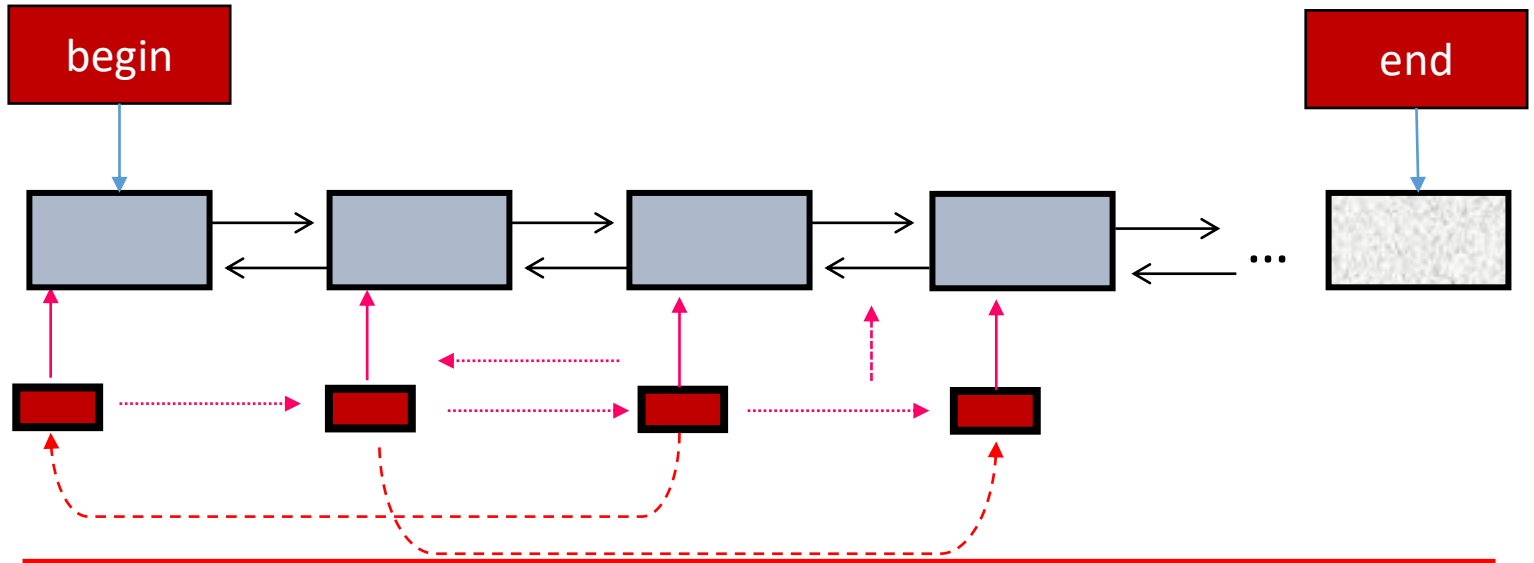


Pointer-related operators

- Single read: RHS = *p
- Single write: LHS *p =
- Access : ->

Random access iterator: details

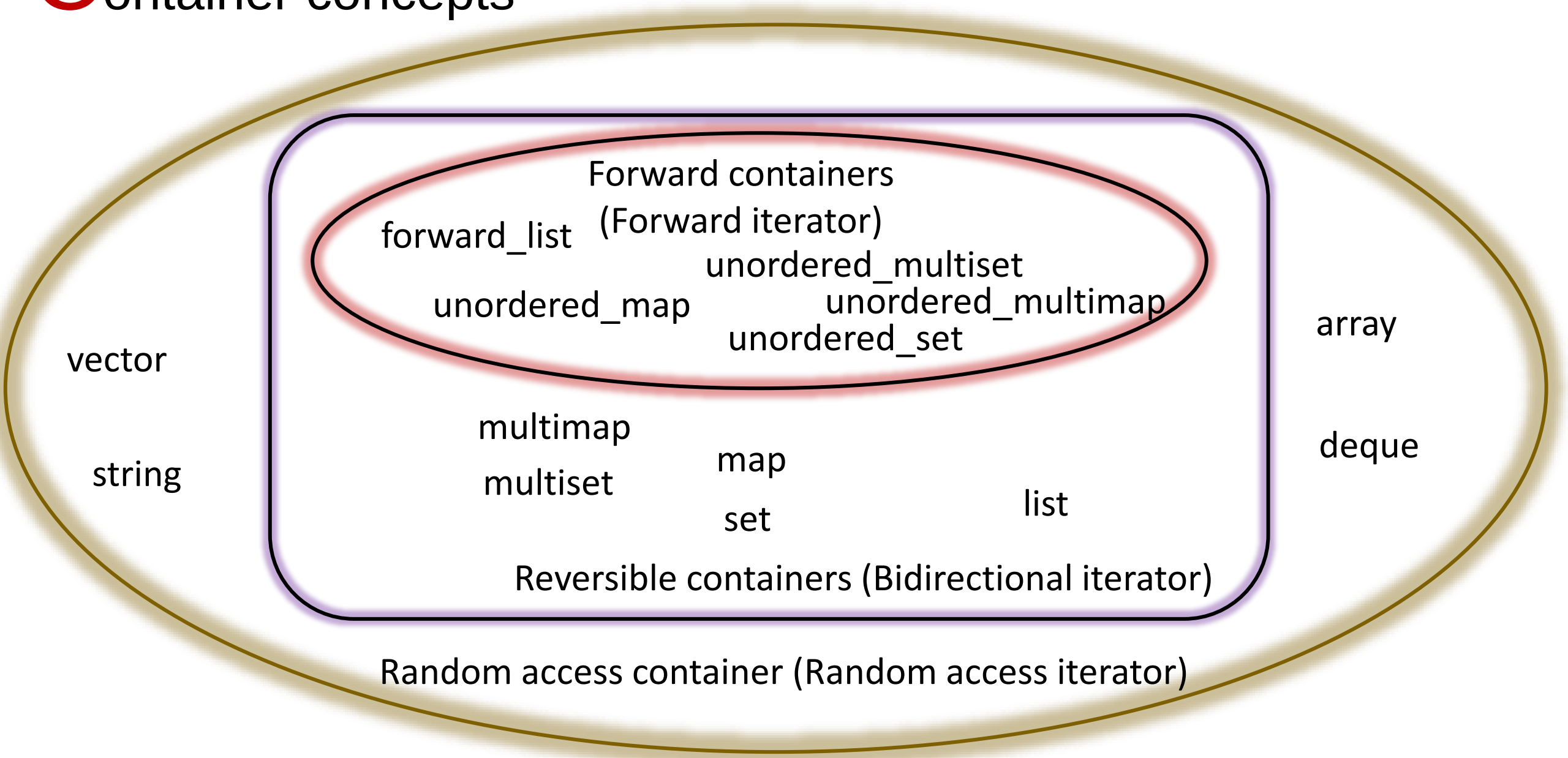
- Random access iterator: Randomly read/write
  - Algorithms: sort
  - Container: vector




```
template<class RandomAccesslIter>
void sort(RandomAccesslIter first, RandomAccesslIter last)
{
    /* the quick-sort implementation */
}

void sort_user()
{
    list<int> L{ 10, 3, 2 };
    L.sort(); // sort member function
    vector<int> v(3) = { 0, 6, 5 };
    sort(v.begin(), v.end()); // sort generic function
}
```

Container concepts



References and further readings

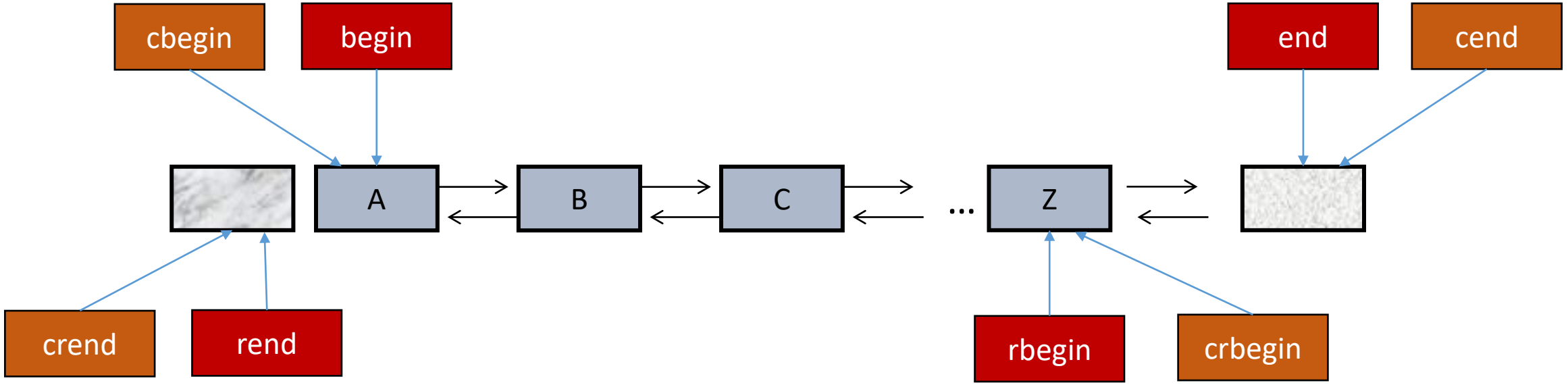
- Matthew H. Austern. Generic Programming and the STL: Using and Extending the C++ Standard Template Library. Addison-Wesley, 1999.

Containers- members

- Member types
- Special member functions: constructors, destructor, assignments
- Size and Capacity
- Iterators
- Element Access
- Stack operations
- List operations

container members: Iterators

- begin(), end(), cbegin(), cend(), rbegin(), rend(), crbegin(), crend()
- c for constant
- r for reverse iterator
- both member functions and non-member functions



containers: Interfaces

- Vector

```
template<class T> class std::vector {
  // ctors, assignment op., dtors
  // …
  iterator begin();
  iterator end();
  size_type size();
  bool empty();
  T front();
  T back();
  void push_back(const T& t);
  void pop_back();
  T operator[](int index);
  void resize(size_type sz, T val = T());
  // other member functions
};
```

- list

```
template<class T> class std::list {
  // ctors, assignment op., dtors
  // …
  iterator begin();
  iterator end();
  size_type size();
  bool empty();
  T front();
  T back();
  void push_back(const T& t);
  void pop_back();
  void push_front(const T& t);
  void pop_front();
  T splice (iterator pos, list& x);
  // other member functions
};
```

- map

```
template<class Key, class T> class std::map {
  // ctors, assignment op., dtors
  // …
  iterator begin();
  iterator end();
  size_type size();
  bool empty();
  T operator[](Key& k);
  iterator lower_bound(const key& k);
  iterator upper_bound(const key& k);
  // other member functions
};
```

ContainerManipTest Prog.

Thanks for your patience …

A man who asks a question is a fool for minute,
The man who does not ask, is a fool for a life.
- Confucius

Learning to ask the right (often hard) questions is an essential part of learning to think as a programmer.
- Bjarne Stroustrup *programming Principles and Practice Using C++, page 4.*

There is no stupid question, but there is stupid answer.
- Howard Hinnant