

+ 22

Back To Basics

Standard Library Containers

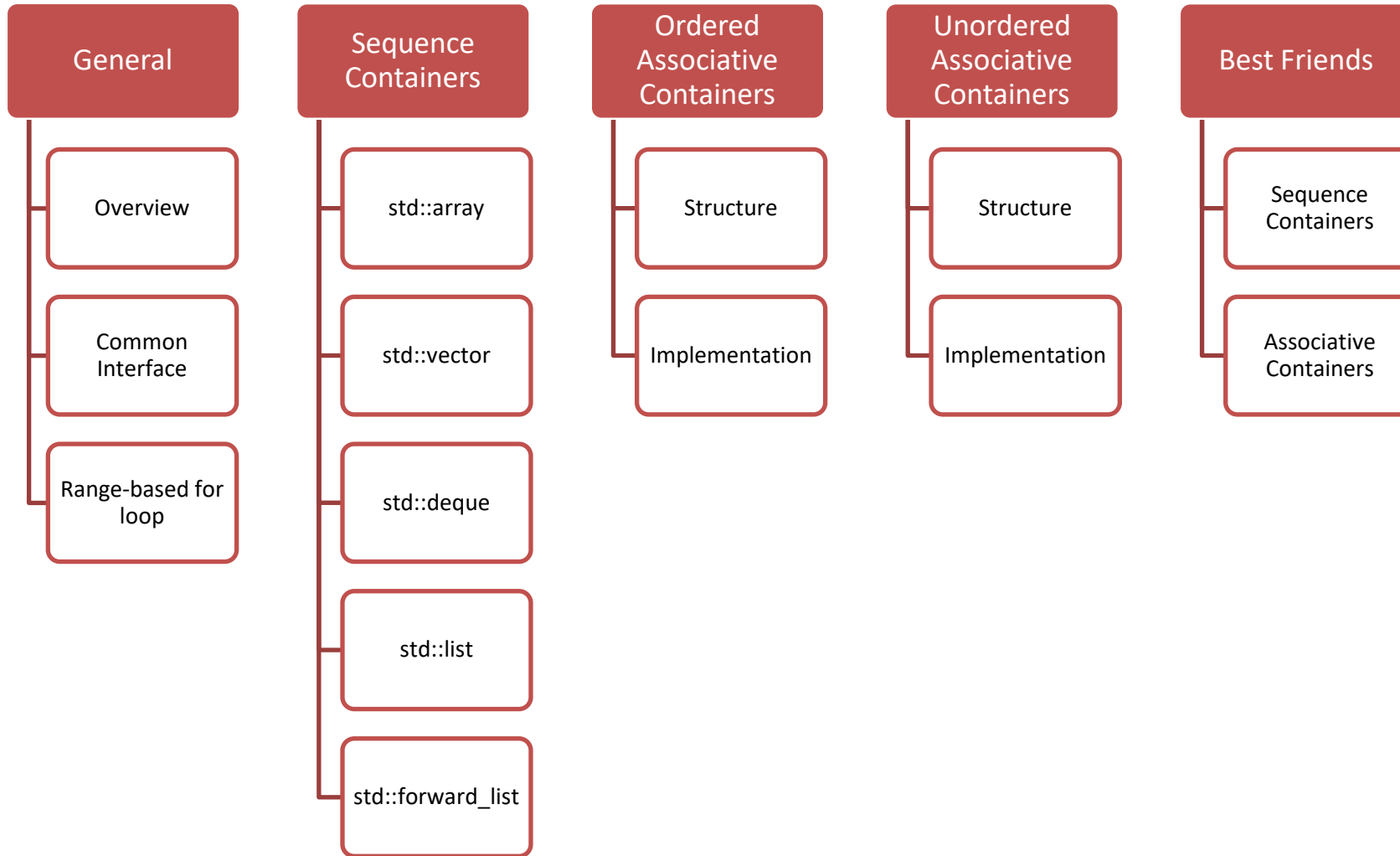
RAINER GRIMM



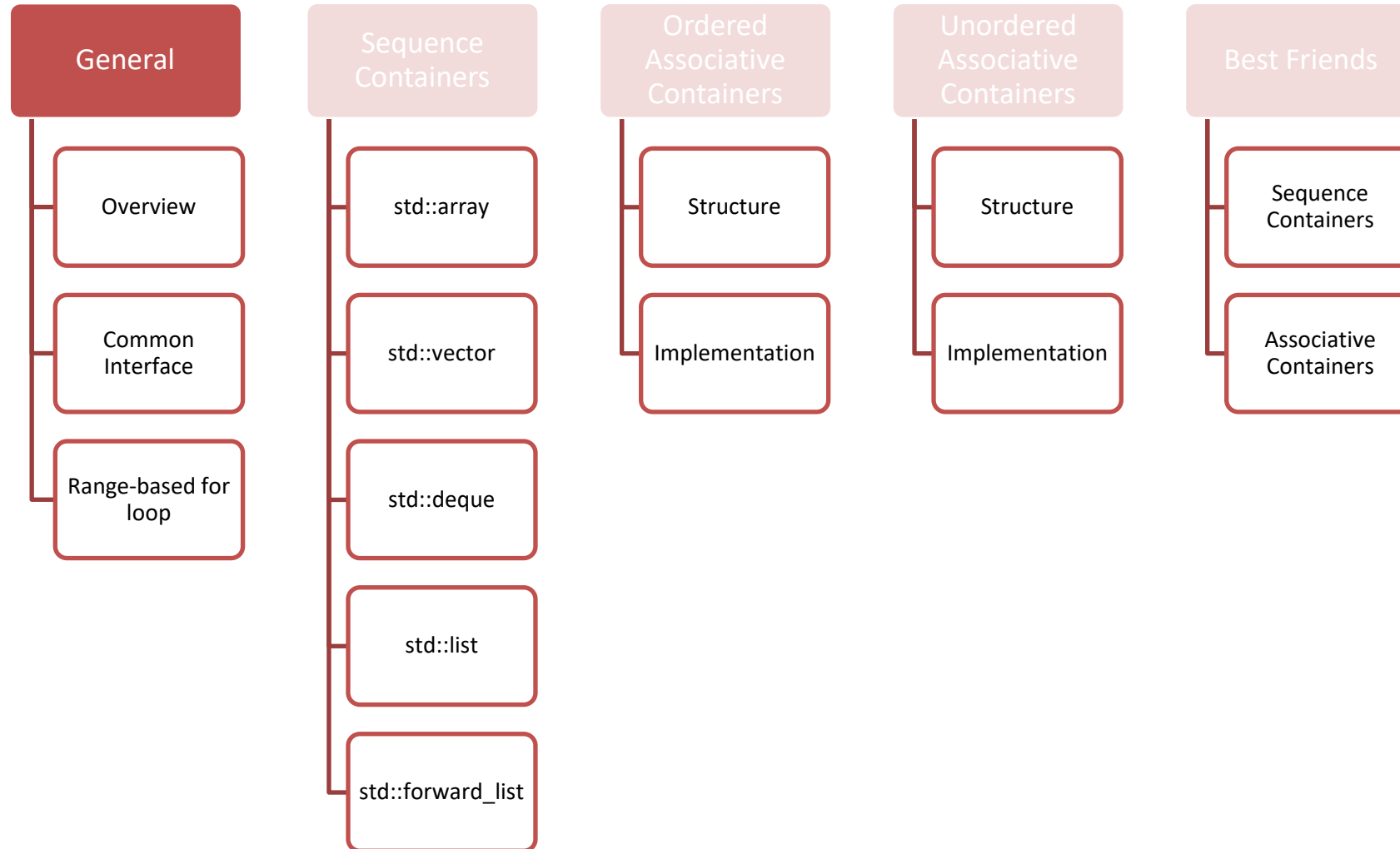
20
22



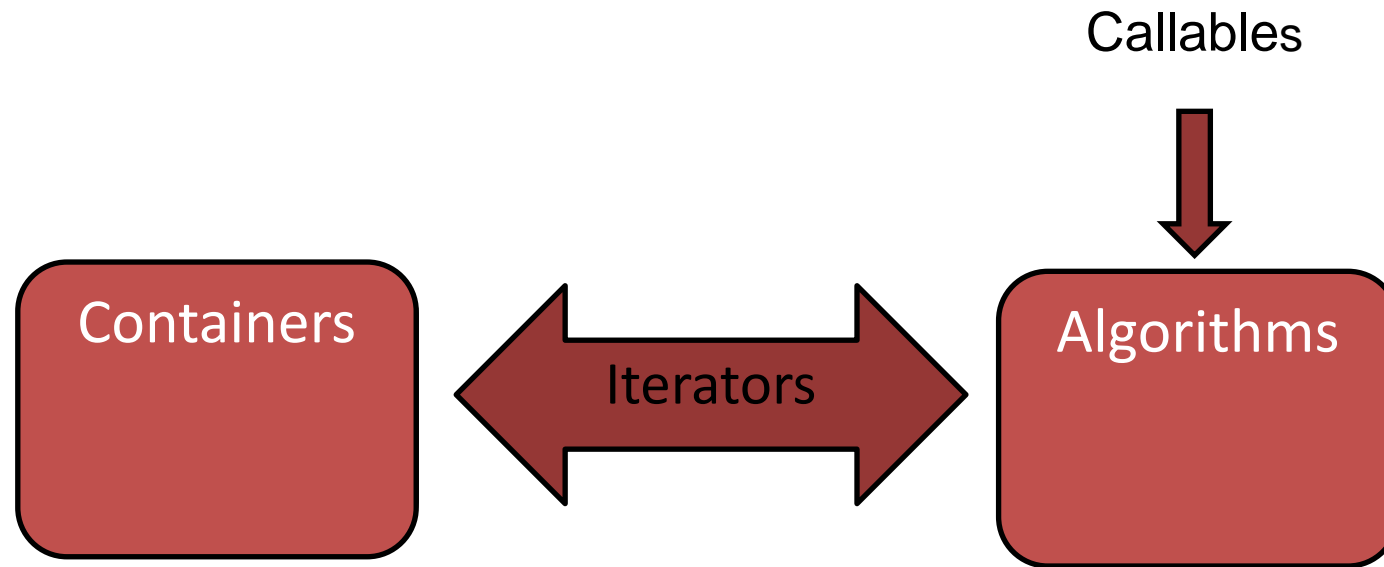
Containers of the STL



Containers of the STL



The Standard Template Library: STL



The Containers of the STL

Sequence Containers

```
std::array  
std::vector  
std::deque  
std::list  
std::forward_list
```

Ordered Associative Containers

```
std::map  
std::set  
std::multimap  
std::multiset
```

Unordered Associative Containers

```
std::unordered_map  
std::unordered_set  
std::unordered_multimap  
std::unordered_multiset
```

Interface of the Containers

The Containers

- have a type parameter(s) and an allocator.

```
template<typename T, typename Allocator = std::allocator<T>>  
class vector;
```

- support the same basic functionality.
- provide value semantics.

Exceptions

- `std::array` has a fixed size → you cannot change its size during run time
- `std::forward_list` does not know its length



`std::string` is quite similar to `std::vector<char>`.

Create and Delete

Constructors

- Default

```
std::vector<int> first;
```
- Count

```
std::vector<int> sec(5);
```
- Range

```
std::vector<int> third(sec.begin(), sec.end());
```
- Copy

```
std::vector<int> fourth(third);
```
- Move

```
std::vector<int> fifth(std::move(fourth));
```
- Sequence

```
std::vector<int> sixth{1, 2, 3, 4, 5};
```

Create and Delete

Destructor

```
std::vector<int> first;  
delete first;
```

Removing

```
std::vector<int> sixth{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
sixth.clear();
```

```
std::erase(sixth, 5);  
std::erase_if(sixth, [](auto i){ return i >= 3; });
```


Determine the Size

```
std::vector<std::string> first{"one", "two", "three", "four"};
```

Empty?

```
first.empty();
```

Numbers of elements?

```
first.size();
```

Maximal size?

```
first.max_size();
```

Assignment and Swap

```
std::vector<std::string> first{"one", "two", "three", "four"};  
std::vector<std::string> second{"five", "six"};
```

Assignment

```
second = first;  
second = std::move(first);  
second = {"seven", "eight"};
```

Swap

```
second.swap(first);  
std::swap(second, first)
```

Comparison

All Containers

== and !=

Sequence and Ordered Associative Containers

<, <=, >, and >=

Rules

- The containers must have the same type.
- Two containers are equal, if they have the same elements in the same sequence (applies to sequence containers and ordered associative containers).
- The containers are compared lexicographically.

The Range-Based for loop

Syntax:

```
for (Declaration: Sequence) { . . . }
```

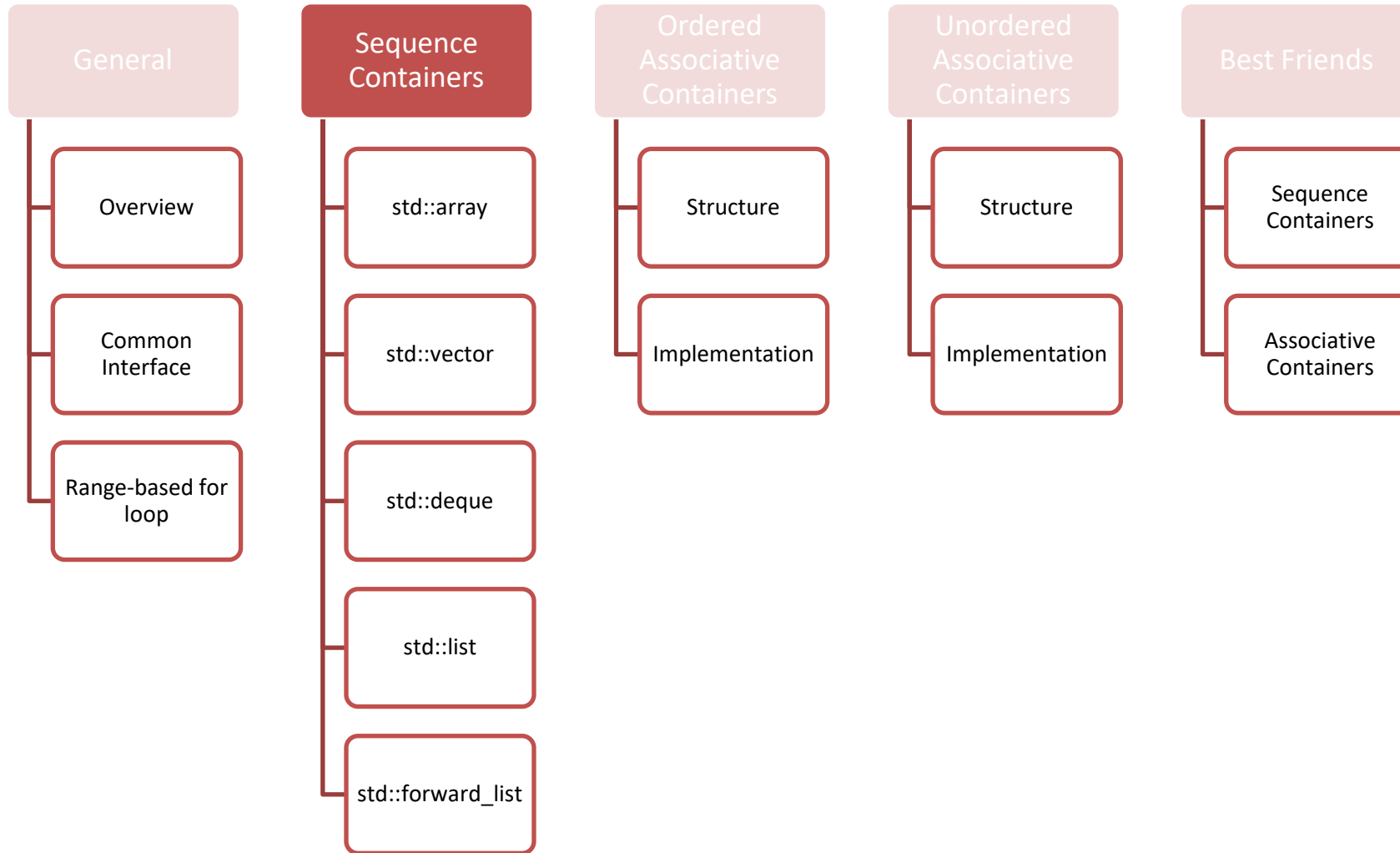
Sequence: initializer list, C array, C++ string, STL container, or range

```
std::vector<int> vec{1, 2, 3, 4, 5};  
for (auto v: vec) std::cout << v << " ";           // 1 2 3 4 5
```

You must take the arguments by reference to modify them.

```
int array[5] = {1, 2, 3, 4, 5};  
for (auto& a: array) a *=2;  
for (auto a: array) std::cout << a << " ";           // 2 4 6 8 10
```

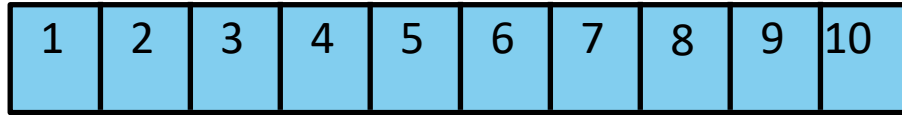
Containers of the STL



Sequence Containers: Overview

Characteristic	<code>std::array</code>	<code>std::vector</code>	<code>std::deque</code>	<code>std::list</code>	<code>std::forward_list</code>
Size	static	dynamic	dynamic	dynamic	dynamic
Implementation	static array	dynamic array	sequence of arrays	doubly linked list	singly linked list
Access	random access	random access	random access	for- and backward	forward
Optimized for		end $O(1)$	begin and end $O(1)$	begin and end $O(1)$	begin $O(1)$
Memory reservation		yes	no	no	no
Memory release		<code>shrink_to_fit()</code>	<code>shrink_to_fit()</code>	always	always
Iterator invalidation		yes	yes	no	no
Strength	<ul style="list-style-type: none"> ▪ no memory allocation ▪ minimal memory requirements 	95% solution	insert and delete at the begin and end	insert and delete at each position	<ul style="list-style-type: none"> ▪ fast insertion and deletion ▪ minimal memory requirements
Weakness	no dynamic memory allocation	insertion and deletion at arbitrary positions $O(n)$	insertion and deletion at arbitrary positions $O(n)$	no random access	no random access

std::array



```
std::array<int,10> myArr{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

std::array

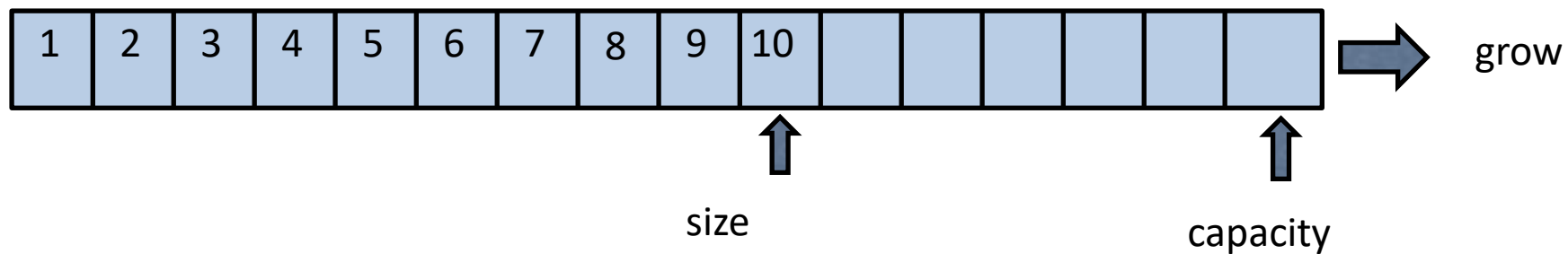
- is a homogeneous container of fixed length.
- combines the memory and performance characteristics of the C array with the interface of a C++ vector.
- knows its length.

std::array

Aggregat Initialization

- `std::array<int, 10> arr:` Elements are not initialized
- `std::array<int, 10> arr{}`: Elements are default initialized
- `std::array<int, 10> arr{1, 2, 3, 4, 5}`: Remaining elements are default initialized

std::vector



```
std::vector<int> myInt{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

std::vector

- it's a homogenous container of variable length.
- manages automatically its memory.
- stores its elements continuously. → Supports the index operator
- reserves more memory than needed. → reduces expensive memory allocation

std::vector


Special elements

`vec.front()`  first element (not checked)

`vec.back()`  last element (not checked)

Index access

`vec[n]`  vector boundaries are not checked

`vec.at(n)`  vector boundaries are checked (`std::out_of_range` exception)

Pointer arithmetic

`&vec[i] ≡ &vec[0] + i`

std::vector

Elements

- **Assign**

`vec.assign(...)`

- **Insert**

`vec.insert(pos, ...), vec.push_back(elem)`

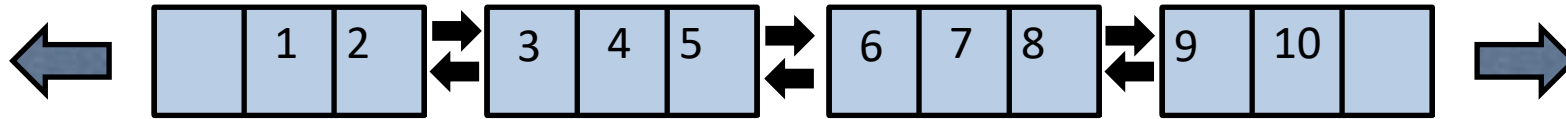
- **In-place creation**

`vec.emplace(pos, args ...), vec.emplace_back(args ...)`

- **Clear**

`vec.pop_back(), vec.erase(...), vec.clear()`

std::deque

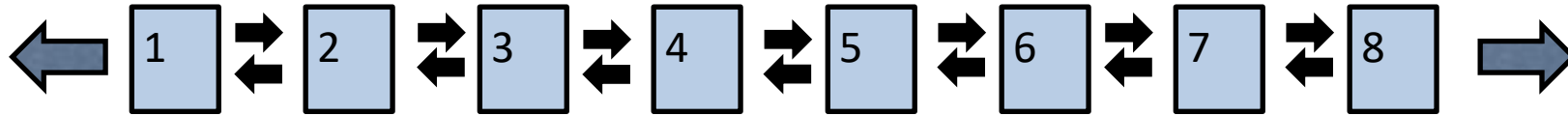


```
std::deque<int> deq{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

std::deque (**double ended queue**)

- Extends the interface of `std::vector`
 - `deq.push_front(elem)`, `deq.pop_front()`, and `deq.emplace_front(args ...)`

std::list



```
std::list<int> lis{1, 2, 3, 4, 5, 6, 7, 8};
```

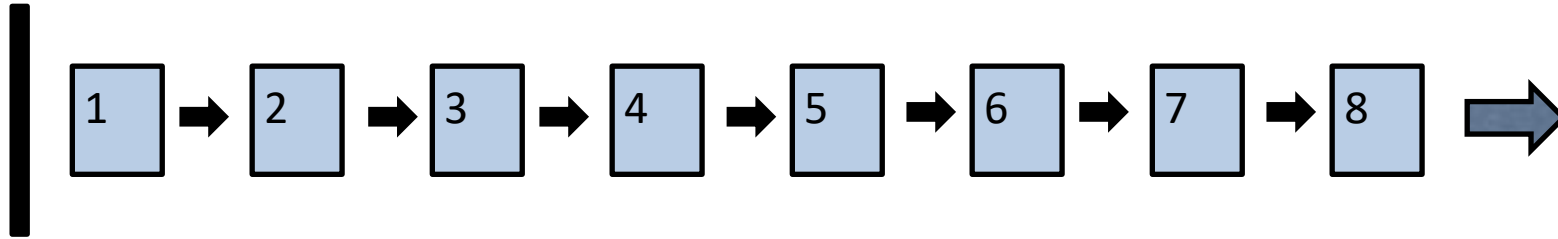
std::list

- is pretty different to `std::array`, `std::vector`, and `std::deque`.
- supports fast access at the front and end of the list.



`std::list` has special member functions optimized for pointer manipulation.

std::forward_list



```
std::forward_list<int> for{1, 2, 3, 4, 5, 6, 7, 8};
```

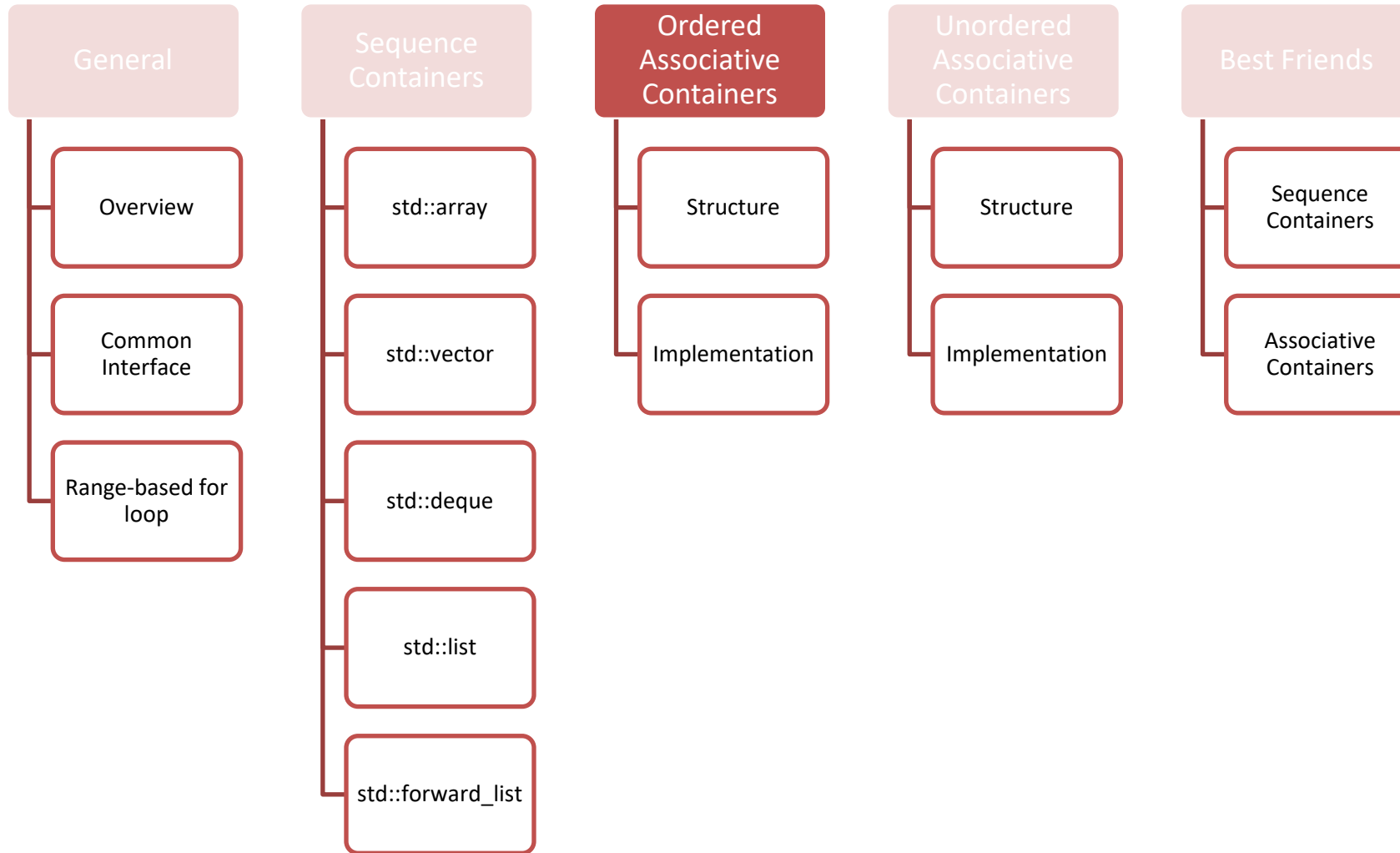
std::forward_list

- is a single linked list.
- is similar to `std::list`, but with a restricted interface.
- is optimized for minimal memory requirements.



`std::forward_list` is designed for the special use case.

Containers of the STL



Ordered Associative Containers

The ordered associative containers

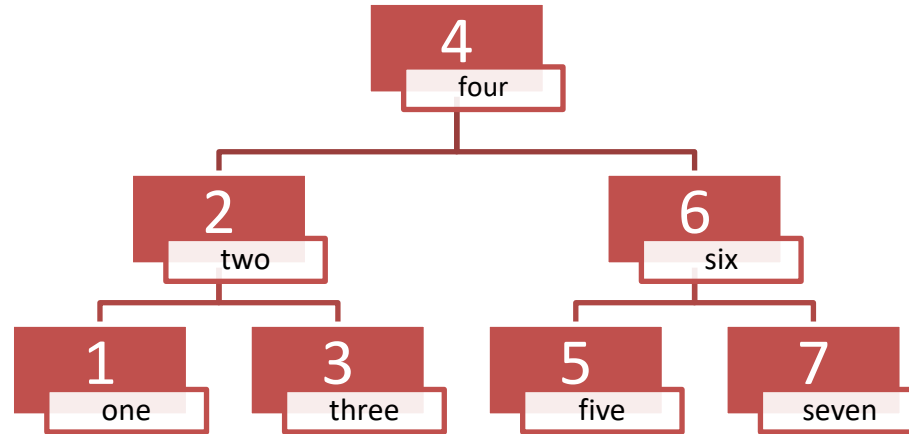
- are *analogous phone books*.

Ordered Associative Containers	Value Associated	More Identical Keys	Header
<code>std::set</code>	no	no	<code><set></code>
<code>std::multiset</code>	no	yes	<code><set></code>
<code>std::map</code>	yes	no	<code><map></code>
<code>std::multimap</code>	yes	yes	<code><map></code>

- have a type(s), a comparison function, and an allocator.

```
template <typename Key, typename Value,  
          typename Compare = std::less<Key>,  
          typename Allocator = std::allocator<std::pair<const Key, Value>>>  
class map;
```


Ordered Associative Containers



```
std::map<int, std::string> int2String{{3, "three"}, {2, "two"},  
                                     {1, "one"}, {5, "five"},  
                                     {6, "six"}, {4, "four"},  
                                     {7, "seven"} };
```

Ordered associative Containers are


- binary, balanced search trees.
- sorted in ascending order (by default).

Ordered Associative Containers


`std::map`

- is the most popular ordered associative container.
- supports the index operator [].

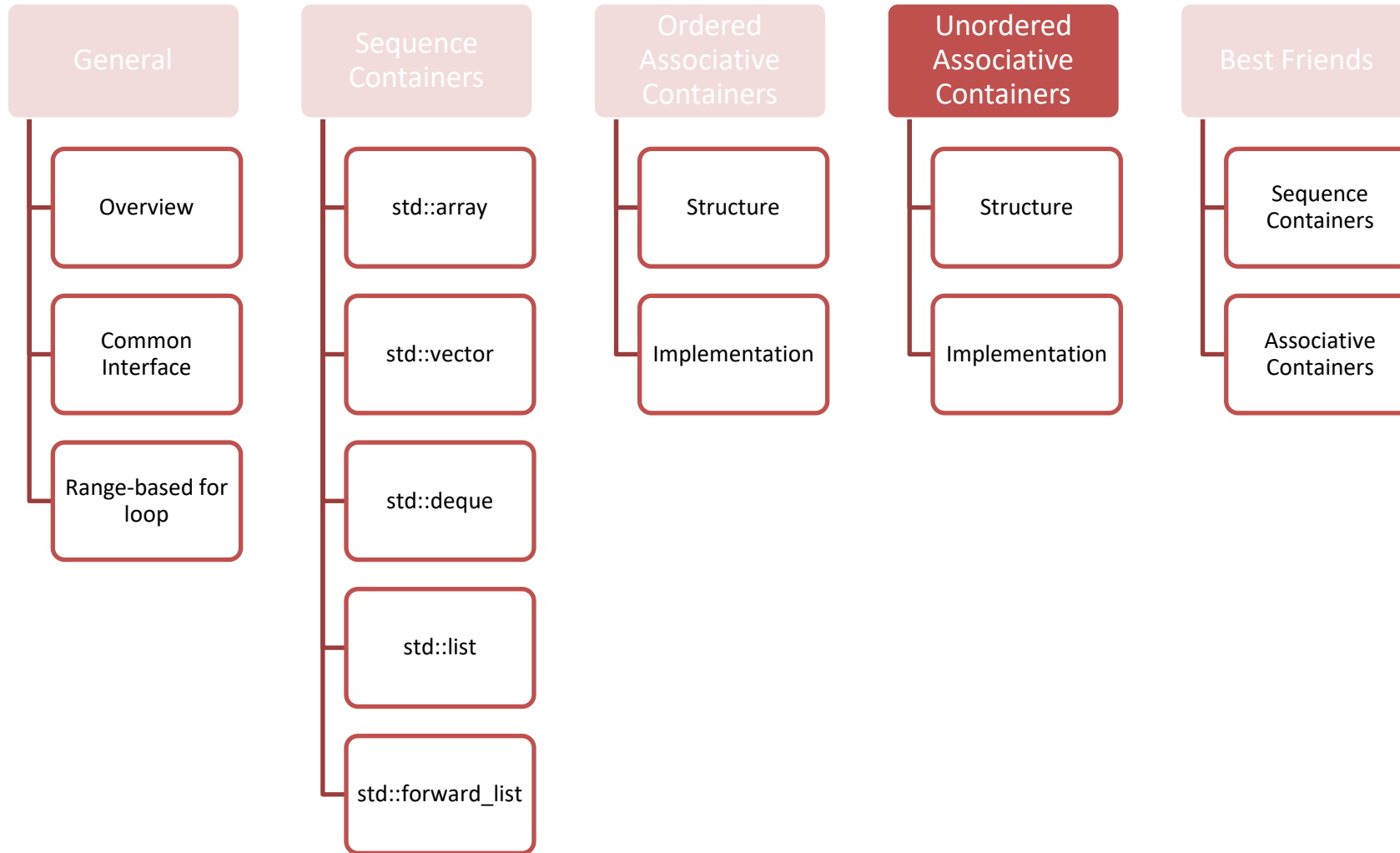
Index operator []

- enables the reading and writing access.
- Creates a new key/value pair if the key is not available,
 invokes the default constructor for the value.

at operator

- Allows to read a key without creating the value
 `std::out_of_range` exception

Containers of the STL



Unordered associative Containers

The unordered associative Containers

- are *digital phone books*.

Unordered Associative Containers	Value Available	More Identical Keys	Header
<code>std::unordered_set</code>	no	no	<code><unordered_set></code>
<code>std::unordered_multiset</code>	no	yes	<code><unordered_set></code>
<code>std::unordered_map</code>	yes	no	<code><unordered_map></code>
<code>std::unordered_multimap</code>	yes	yes	<code><unordered_map></code>

- are also known as dictionary, associative arrays, or hash tables.
- extend the interface of the ordered associative containers.

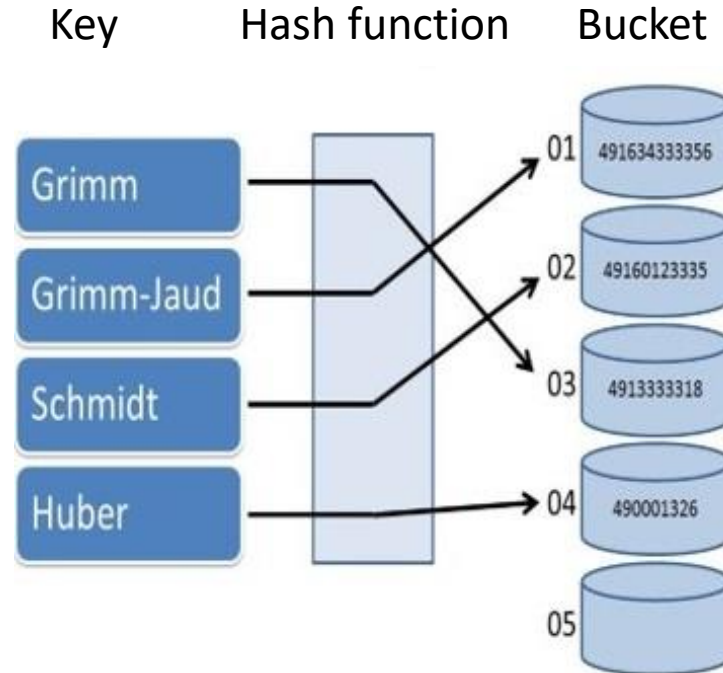
Unordered associative Containers

The unordered associative Containers

- have a data type(s), a hash function, a equal function, and an allocator.

```
template<typename Key, typename Value,  
        typename Hash = std::hash<Key>,  
        typename KeyEqual = std::equal_to<Key>,  
        typename Allocator = std::allocator<std::pair<const Key, Value>>>  
class unordered_map;
```

Unordered Associative Containers



```
std::unordered_map<std::string, int> {{"Grimm", 4916343333},  
                                     {"Grimm-Jaud", 491601233},  
                                     {"Schmidt", 49133318},  
                                     {"Huber", 4900013}};
```



The hash function maps the key in constant time to its index.

Unordered Associative Containers

Collision:

- Unordered associative containers store their keys in the buckets.
- Different keys with the same hash value can be stored in the same bucket.
- The access time of the bucket is constant, the search in the bucket is linear.

Capacity:

- Number of buckets

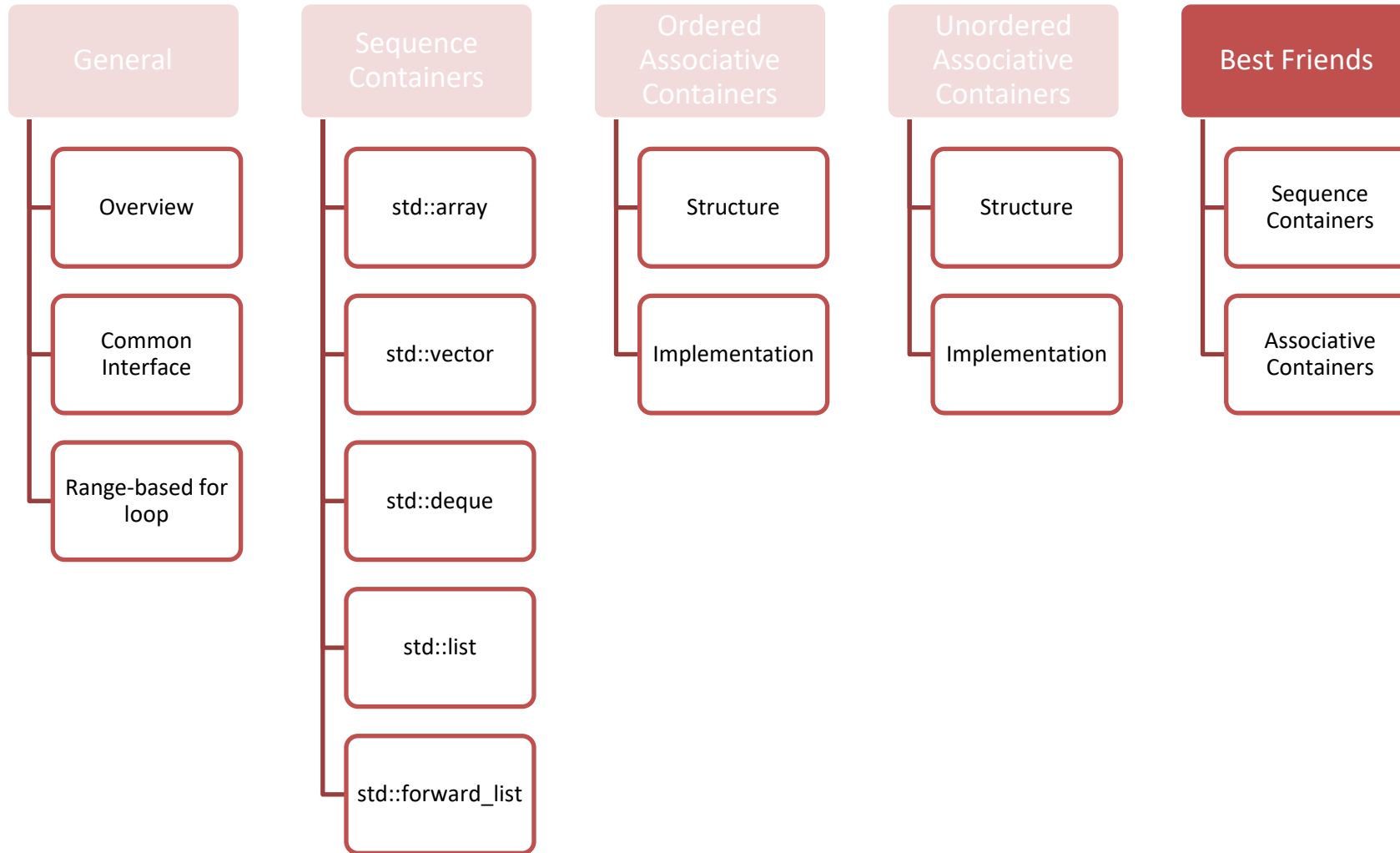
Load factor:

- Average number of elements of each bucket.

Rehashing:



- New buckets are created per default, if the load factor is bigger than 1.

Containers of the STL



`std::array` and `std::vector`

Prefer `std::array` and `std::vector` to a C-array

- The container size is known at compile time and small  `std::array`
- The container size is not known at compile time or big  `std::vector`
- `std::vector` and `std::array`
 - know its size.
 - automatically manage its memory (RAII).
 - allow the protected element access via the `at`-operator.
 - have an ideal memory layout.



`std::array` and `std::vector` should be your first choice for a sequence container.

Sequence Containers

Performance Comparison

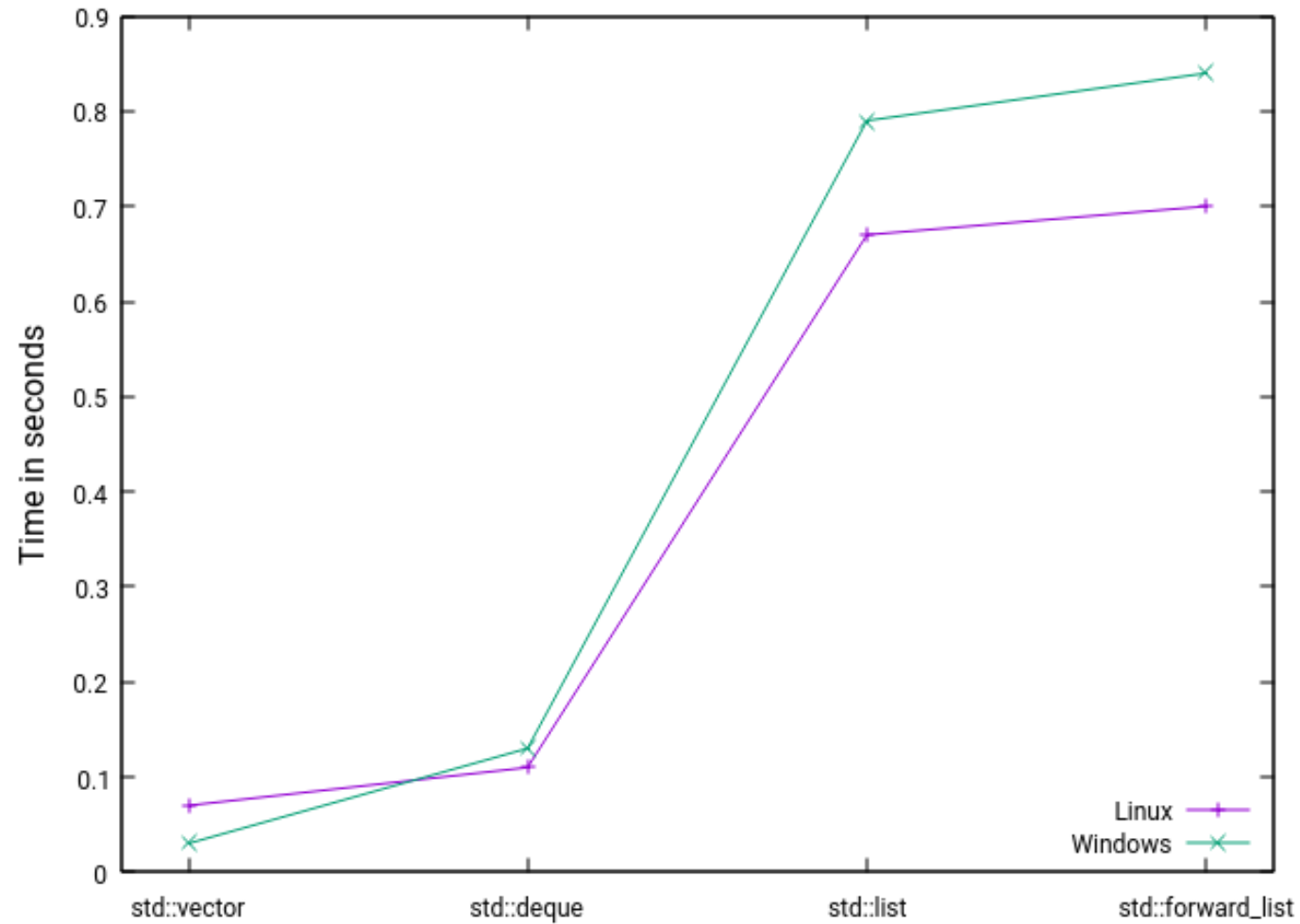
- Sum up all 100'000'000 values of various sequence containers using `std::accumulate`
- **Used sequence containers:** `std::vector`, `std::deque`, `std::list`, and `std::forward_list`

Computer architectures are optimized for the reading of contiguous memory blocks.

 memory predictability

Sequence Containers

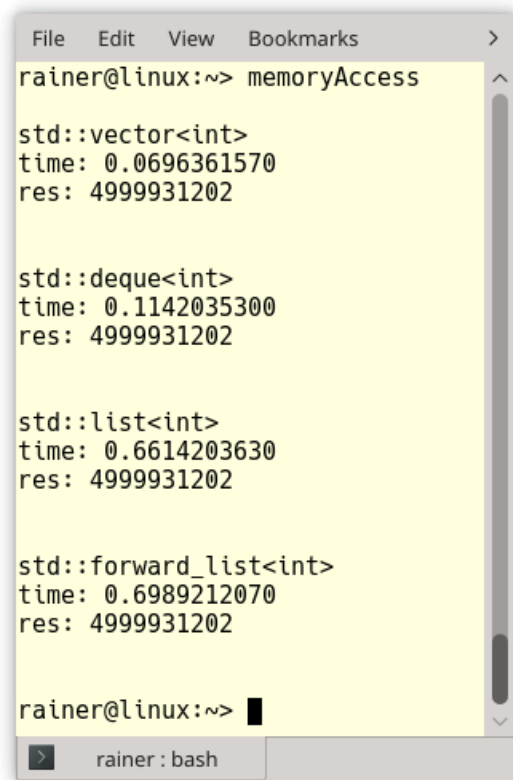
Relative Performance



Sequence Containers

Absolute Performance

Linux



```
File Edit View Bookmarks >
rainer@linux:~> memoryAccess

std::vector<int>
time: 0.0696361570
res: 4999931202

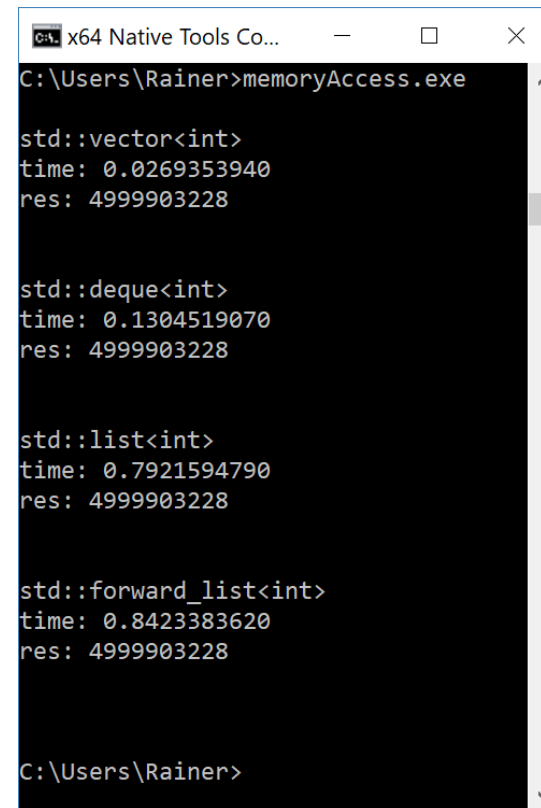
std::deque<int>
time: 0.1142035300
res: 4999931202

std::list<int>
time: 0.6614203630
res: 4999931202

std::forward_list<int>
time: 0.6989212070
res: 4999931202

rainer@linux:~> █
rainer : bash
```

Windows



```
cs x64 Native Tools Co... - □ ×
C:\Users\Rainer>memoryAccess.exe

std::vector<int>
time: 0.0269353940
res: 4999903228

std::deque<int>
time: 0.1304519070
res: 4999903228

std::list<int>
time: 0.7921594790
res: 4999903228

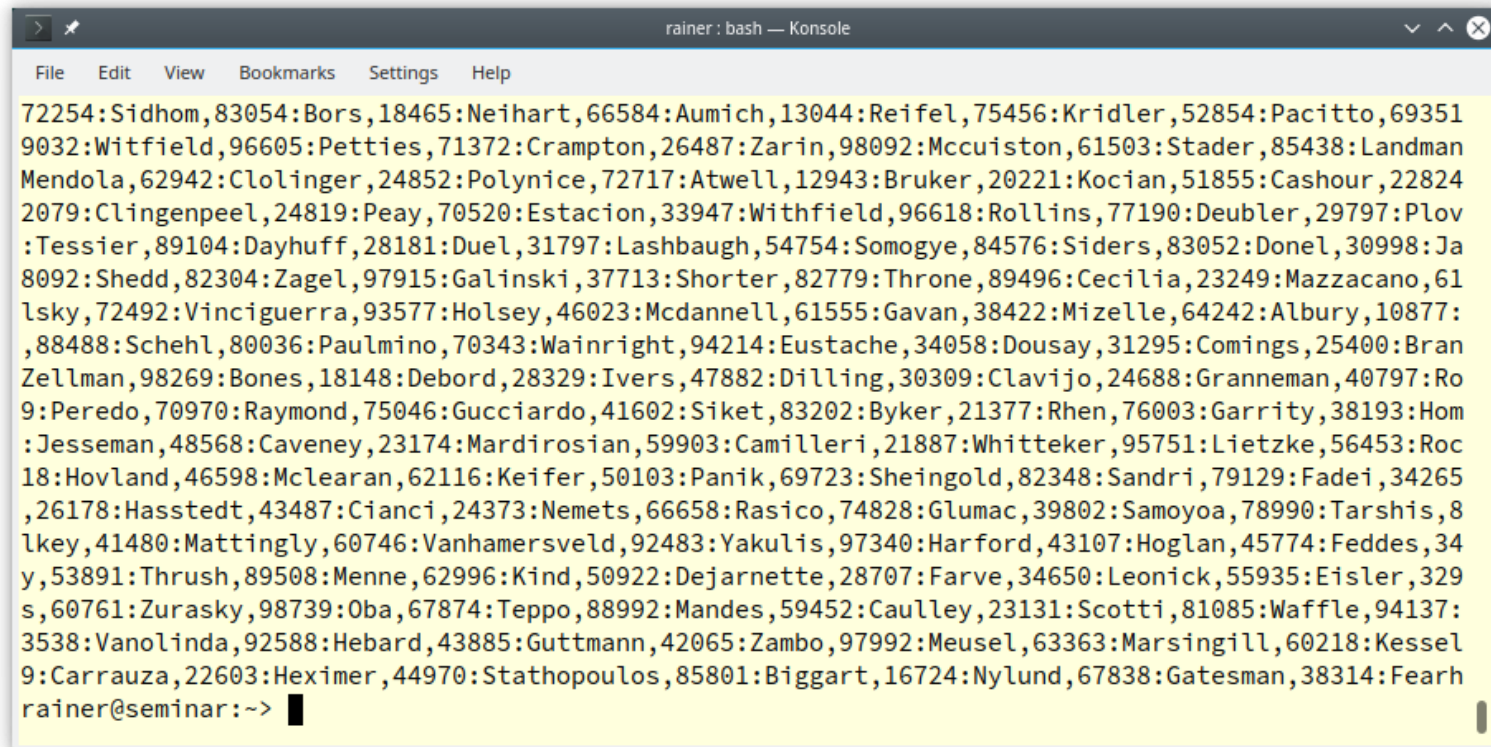
std::forward_list<int>
time: 0.8423383620
res: 4999903228

C:\Users\Rainer>
```

`std::map` or `std::unordered_map`

Performance Comparison:

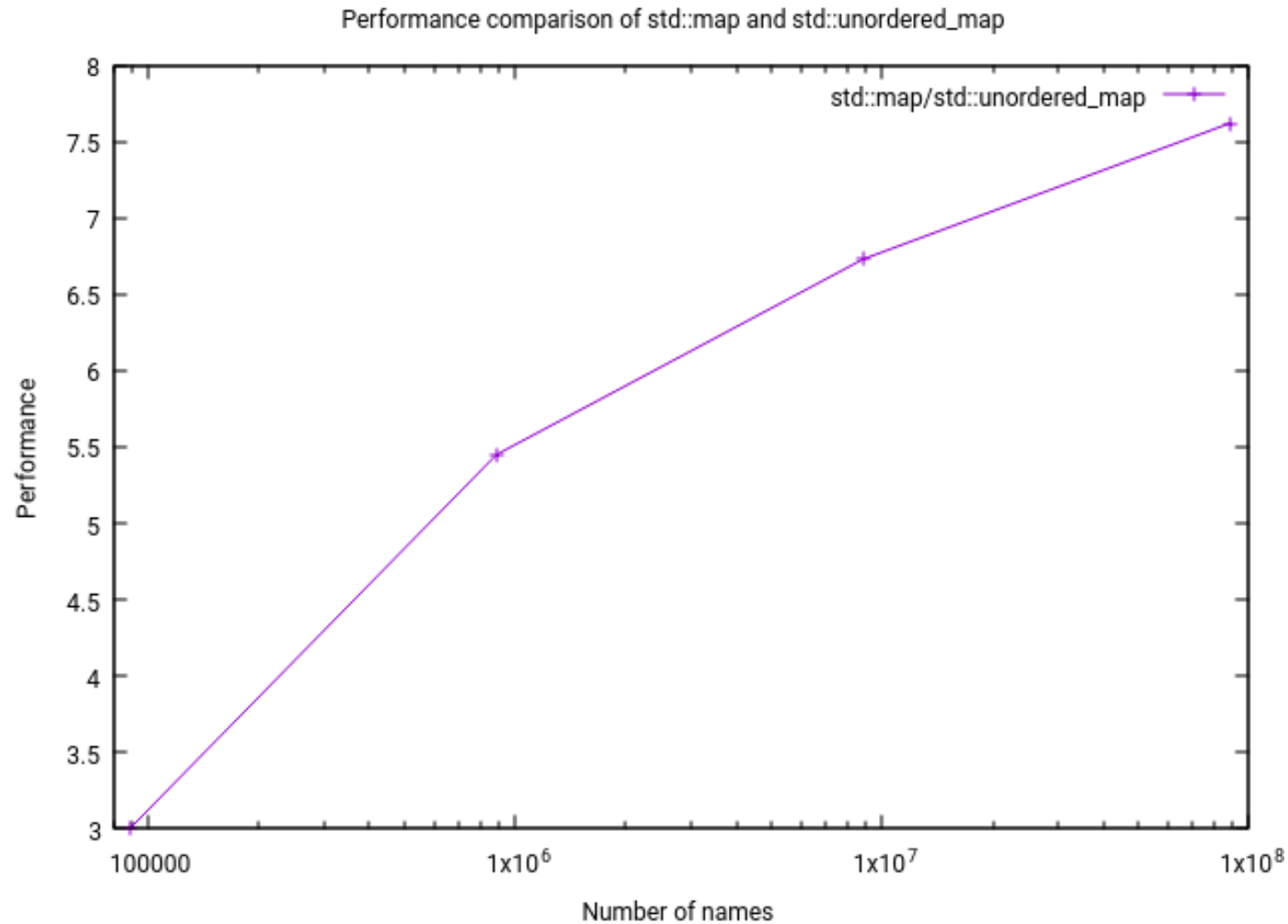
- Reading all keys of a `std::map` and a `std::unordered_map`
- Container size: Roughly 90'000 – 90'000'000 entries



```
rainer : bash — Konsole
File Edit View Bookmarks Settings Help
72254:Sidhom,83054:Bors,18465:Neihart,66584:Aumich,13044:Reifel,75456:Kridler,52854:Pacitto,69351
9032:Witfield,96605:Petties,71372:Crampton,26487:Zarin,98092:Mccuiston,61503:Stader,85438:Landman
Mendola,62942:Clolinger,24852:Polynice,72717:Atwell,12943:Bruker,20221:Kocian,51855:Cashour,22824
2079:Clingenpeel,24819:Peay,70520:Estacion,33947:Withfield,96618:Rollins,77190:Deubler,29797:Plov
:Tessier,89104:Dayhuff,28181:Duel,31797:Lashbaugh,54754:Somogye,84576:Siders,83052:Donel,30998:Ja
8092:Shedd,82304:Zagel,97915:Galinski,37713:Shorter,82779:Throne,89496:Cecilia,23249:Mazzacano,61
lsky,72492:Vinciguerra,93577:Holsey,46023:Mcdannell,61555:Gavan,38422:Mizelle,64242:Albury,10877:
,88488:Schehl,80036:Paulmino,70343:Wainright,94214:Eustache,34058:Dousay,31295:Comings,25400:Bran
Zellman,98269:Bones,18148:Debord,28329:Ivers,47882:Dilling,30309:Clavijo,24688:Granneman,40797:Ro
9:Peredo,70970:Raymond,75046:Gucciardo,41602:Siket,83202:Byker,21377:Rhen,76003:Garrity,38193:Hom
:Jesseman,48568:Caveney,23174:Mardirosian,59903:Camilleri,21887:Whitteker,95751:Lietzke,56453:Roc
18:Hovland,46598:Mclearan,62116:Keifer,50103:Panik,69723:Sheingold,82348:Sandri,79129:Fadei,34265
,26178:Hasstedt,43487:Cianci,24373:Nemets,66658:Rasico,74828:Glumac,39802:Samoyoa,78990:Tarshis,8
lkey,41480:Mattingly,60746:Vanhamersveld,92483:Yakulis,97340:Harford,43107:Hoglan,45774:Feddes,34
y,53891:Thrush,89508:Menne,62996:Kind,50922:Dejarnette,28707:Farve,34650:Leonick,55935:Eisler,329
s,60761:Zurasky,98739:Oba,67874:Teppo,88992:Mandes,59452:Caulley,23131:Scotti,81085:Waffle,94137:
3538:Vanolinda,92588:Hebard,43885:Guttman,42065:Zambo,97992:Meusel,63363:Marsingill,60218:Kessel
9:Carrauz,22603:Heximer,44970:Stathopoulos,85801:Biggart,16724:Nylund,67838:Gatesman,38314:Fearh
rainer@seminar:~>
```

`std::map` or `std::unordered_map`

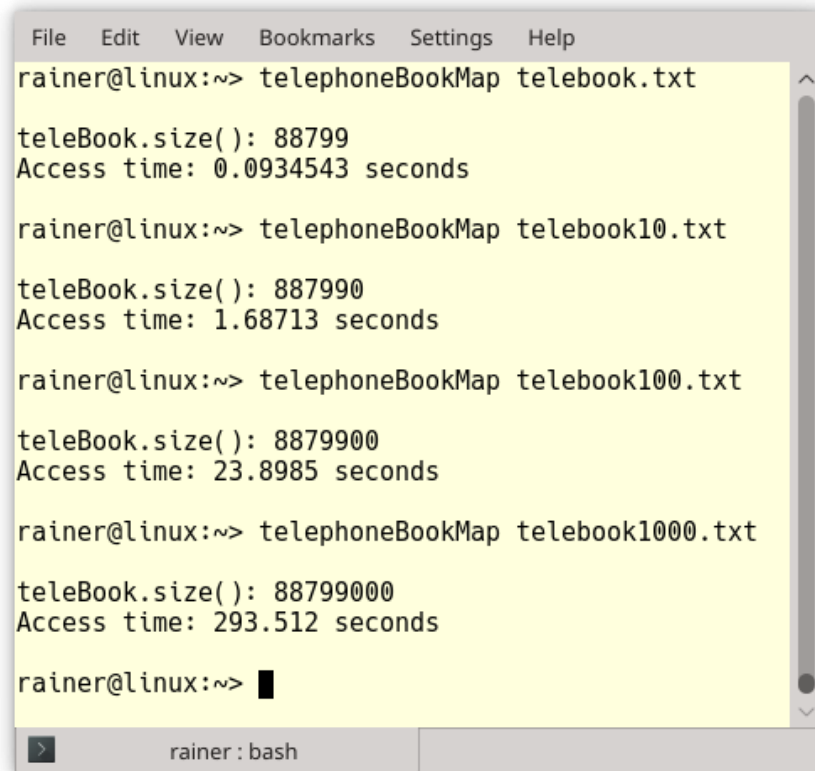
Relative Performance



std::map or std::unordered_map

Absolute Performance

std::map



```
File Edit View Bookmarks Settings Help
rainer@linux:~> telephoneBookMap telebook.txt
teleBook.size(): 88799
Access time: 0.0934543 seconds

rainer@linux:~> telephoneBookMap telebook10.txt
teleBook.size(): 887990
Access time: 1.68713 seconds

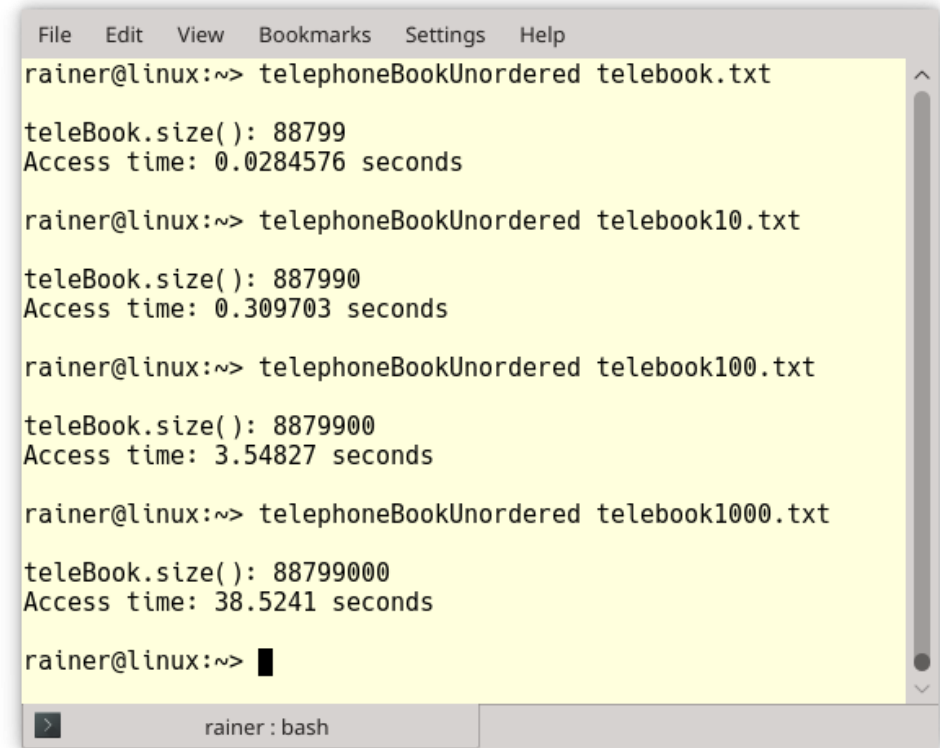
rainer@linux:~> telephoneBookMap telebook100.txt
teleBook.size(): 8879900
Access time: 23.8985 seconds

rainer@linux:~> telephoneBookMap telebook1000.txt
teleBook.size(): 88799000
Access time: 293.512 seconds

rainer@linux:~> █
```

rainer : bash

std::unordered_map



```
File Edit View Bookmarks Settings Help
rainer@linux:~> telephoneBookUnordered telebook.txt
teleBook.size(): 88799
Access time: 0.0284576 seconds

rainer@linux:~> telephoneBookUnordered telebook10.txt
teleBook.size(): 887990
Access time: 0.309703 seconds

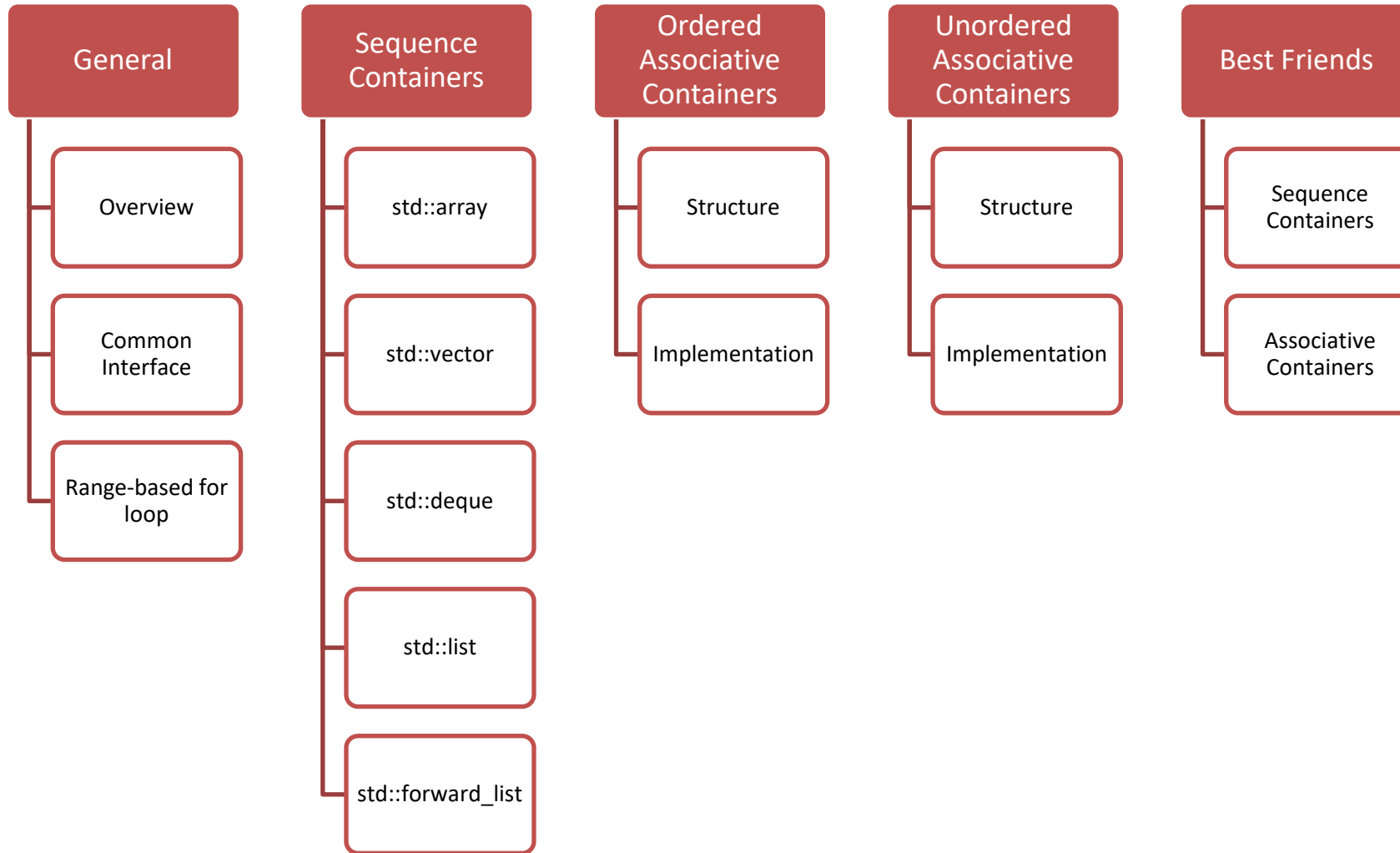
rainer@linux:~> telephoneBookUnordered telebook100.txt
teleBook.size(): 8879900
Access time: 3.54827 seconds

rainer@linux:~> telephoneBookUnordered telebook1000.txt
teleBook.size(): 88799000
Access time: 38.5241 seconds

rainer@linux:~> █
```

rainer : bash

Containers of the STL



Further Information



[The C++ Standard Library](#)



[Posts on Modernes C++](#)



[Modernes C++ Training](#)



[Modernes C++ Mentoring](#)

+ 22

Back To Basics

Standard Library Containers

RAINER GRIMM



20
22

