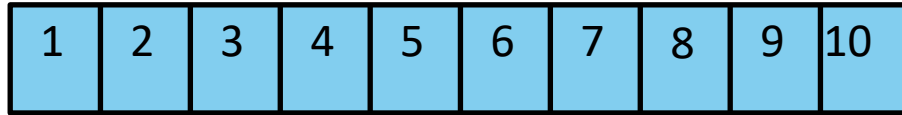


# std::array



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

- `std::array`
  - needs the header `<array>`.
  - is a homogeneous container of fixed length.
  - is similar to `std::tuple`.
  - combines the storage and runtime characteristics of the C array with the interface of a C++ vector.
  - can be used in the STL algorithm.

# std::array

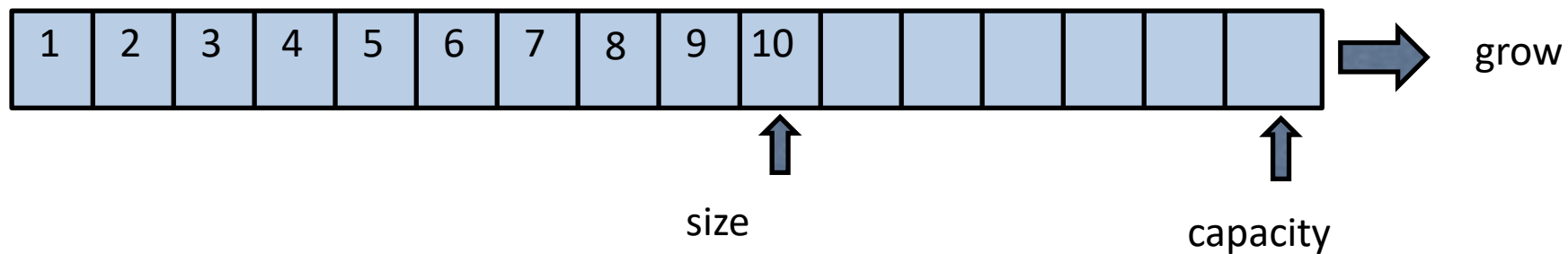
- **Specializations of the initialization**

- `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

- **Relationship with `std::tuple`**

- `arr[4] == std::get<4>(arr)`

# std::vector



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

## ■ std::vector

- Need the header `<vector>`
- Manages automatically its memory
- Stores its elements continuously → support pointer arithmetic
- 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]  $\equiv$  &vec[0] + i`

# std::vector

- **Elements**

- **Assign**

- ```
vec.assign( ... )
```

- **Insert**

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

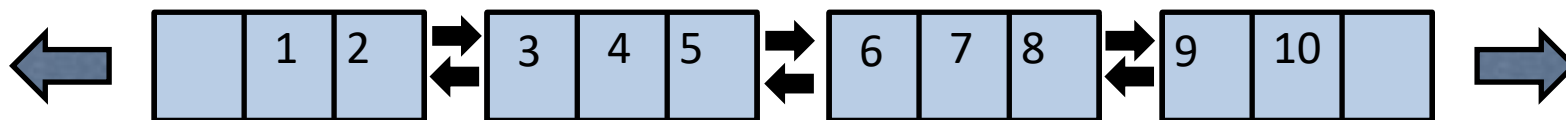
- **In-place creation**

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

- **Clear**

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

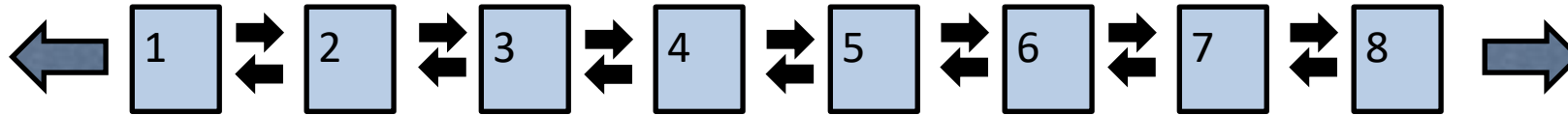
# std::deque



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

- `std::deque` (**double ended queue**)
  - needs the header `<deque>`.
- **Relation to `std::vector`**
  - `std::deque` is quite similar to `std::vector`
  - **Extended interface of `std::deque`**
    - `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};
```

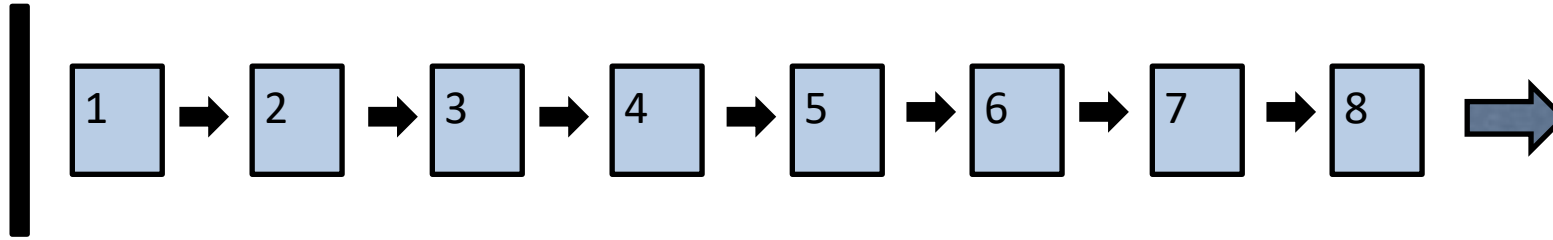
std::list

- needs the header `<list>`.
- is quite different to `std::array`, `std::vector`, and `std::deque`.
- fast access at the front and end of the list.




std::list has many 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`
  - needs the header `<forward_list>`.
  - is a single linked list.
  - similar to `std::list`, but with a restricted interface.
  - optimized for minimal memory requirements.
-  `std::forward_list` is designed for the special use case.