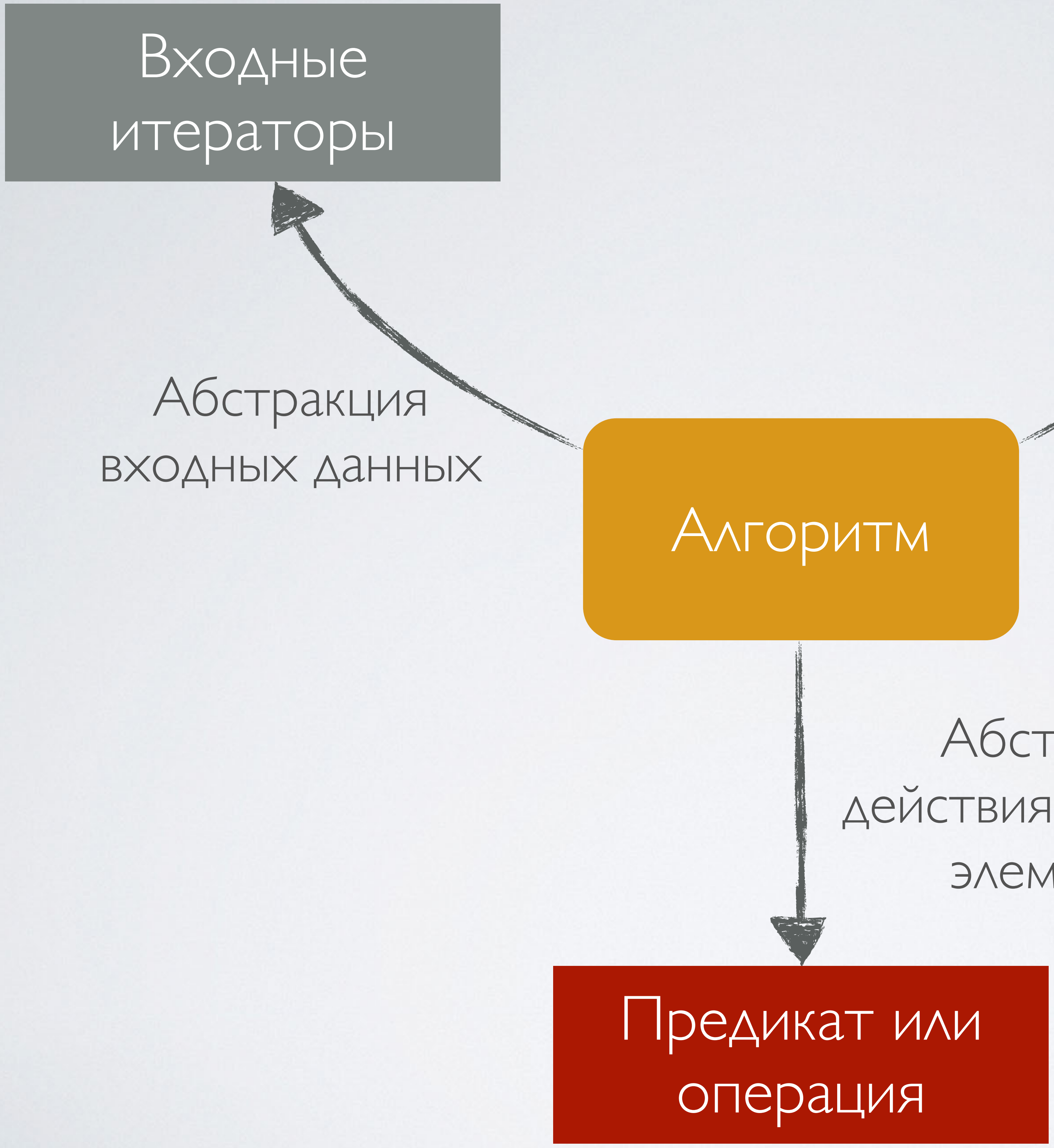
# ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

## STL ALGORITHMS

# STL ALGORITHMS

- **Легкость сопровождения.**
- **Правильность.**
- **Эффективность.**


Входные итераторы
Выходной итератор
Абстракция входных данных
Абстракция выходных данных
Алгоритм
Абстракция действия над одним элементом
Предикат или операция

## COPYING ITEMS FROM CONTAINERS TO CONTAINERS

```cpp
...

int main()
{
   std::vector<std::pair<int, std::string>> v {
         {1, "one"}, {2, "two"}, {3, "three"}, {4, "four"}};
   std::map<int, std::string> m;

   std::copy_n(v.begin(), 3, std::inserter(m, m.begin()));

   ...
}
```

## COPYING ITEMS FROM CONTAINERS TO CONTAINERS

```
...

std::ostream& operator<<(std::ostream &os,
                          const std::pair<int, std::string> &p)
{
    return os << "(" << p.first << ", " << p.second << ")";
}

int main()
{
   ...

   auto shell_it (std::ostream_iterator<
                 std::pair<int, std::string>>{std::cout, ", "});
   std::copy(m.begin(), m.end(), shell_it);

   return 0;
}
```

## STD::COPY

```
template <typename InputIterator, typename OutputIterator>
OutputIterator copy(InputIterator it, InputIterator end_it,
                    OutputIterator out_it)
{
   for (; it != end_it; ++it, ++out_it) {
      *out_it = *it;
   }

   return out_it;
}
```

# STD::COPY

```
template <typename InputIterator, typename OutputIterator>
OutputIterator copy(InputIterator it, InputIterator end_it,
                    OutputIterator out_it)
{
  const size_t num_items (std::distance(it, end_it));
  memmove(out_it, it, num_items * sizeof(*it));
  return it + num_items;
}
```

# SORTING CONTAINERS

```
...

int main()
{
   std::vector<int> v {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   std::random_device rd;
   std::mt19937 g {rd()};

   std::cout << std::is_sorted(v.begin(), v.end()) << '\n';

   std::shuffle(v.begin(), v.end(), g);

   std::cout << std::is_sorted(v.begin(), v.end()) << '\n';

   ...
}
```

# SORTING CONTAINERS

```cpp
bool isLessThanFive(int i){
   return i < 5;
}

int main()
{
   ...
   std::sort(v.begin(), v.end());

   std::cout << std::is_sorted(v.begin(), v.end()) << '\n';

   std::shuffle(v.begin(), v.end(), g);

   std::partition(v.begin(), v.end(), isLessThanFive);

}
```

# SORTING CONTAINERS

```
int main()
{
   ...
   std::shuffle(v.begin(), v.end(), g);

   auto middle (std::next(v.begin(), int(v.size()) / 2));
   std::partial_sort(v.begin(), middle, v.end());

}
```

## SORTING CONTAINERS

```
int main()
{
   ...

   std::shuffle(v.begin(), v.end(), g);

   std::sort(v.begin(), v.end(), std::greater<int>());

}
```

# TRANSFORMING ITEMS IN CONTAINERS

```
...

int square(int i){
  return i * i;
}

int main()
{
  std::vector<int> v {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

  std::transform(v.begin(), v.end(),
      std::ostream_iterator<int>{std::cout, ", "}, square);

  std::cout << '\n';
  ...
}
```

## TRANSFORMING ITEMS IN CONTAINERS

```
...

std::string int_to_string(int i){
   std::stringstream ss;
   ss << i;
   ss << "^2 = ";
   ss << i * i;
   return ss.str();
}

int main()
{
   std::vector<int> v {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   std::vector<std::string> vs;

   std::transform(v.begin(), v.end(),
        std::back_inserter(vs), int_to_string);
}
```

## FINDING ITEMS

```
struct city {
    std::string name;
    unsigned population;
};

bool operator==(const city &a, const city &b) {
    return a.name == b.name && a.population == b.population;
}

int main()
{
   const std::vector<city> c {
      {"Aachen",          246000},
      {"Berlin",         3502000},
      {"Braunschweig",    251000},
      {"Cologne",        1060000}
   };

}
```

## FINDING ITEMS

```
...

int main()
{
   ...
   auto found_cologne (std::find(c.begin(), c.end(),
                                 city{"Cologne", 1060000}));

   //std::vector<city>::iterator found_cologne (
   //   std::find(c.begin(), c.end(), city{"Cologne", 1060000}));
}
```

# FINDING ITEMS

```
...
bool isCologne(const city& item){
   return item.name == "Cologne";
}

int main()
{
   ...
   auto found_cologne (std::find_if(c.begin(), c.end(),
                                    isCologne));

   // std::vector<city>::iterator found_cologne(
   //                 std::find_if(c.begin(), c.end(),isCologne));

}
```

# FINDING ITEMS

```
...

int main()
{
   std::vector<int> v {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   bool contains_7 {std::binary_search(v.begin(), v.end(), 7)};

   auto [lower_it, upper_it] =
                          std::equal_range(v.begin(), v.end(), 7));
```

**half-open interval**
**[lower; upper)**

```
   // std::pair<std::vector<int>::iterator, std::vector<int>::iterator> range =
   //                                       std::equal_range(v.begin(), v.end(), 7));
   // std::vector<int>::iterator lower_it = range.first;
   // std::vector<int>::iterator upper_it = range.second;
}
```

## LOCATING PATTERNS IN STRINGS WITH STD::SEARCH

```cpp
int main()
{
   const std::string long_string {
                  "Lorem ipsum dolor sit amet, consetetur"
                  " sadipscing elitr, sed diam nonumy eirmod"};

   const std::string pattern {"elitr"};

   auto match (std::search(std::begin(long_string),
                           std::end(long_string),
                           std::begin(pattern),
                           std::end(pattern)));
}
```

If found substring: return iterator on substring.
If not found: return std::end(long_string)

# LOCATING PATTERNS IN STRINGS WITH STD::SEARCH

```
int main()
{
  const std::string long_string {
                "Lorem ipsum dolor sit amet, consetetur"
                " sadipscing elitr, sed diam nonumy eirmod"};

  const std::string pattern {"elitr"};

  auto match (std::search(std::begin(long_string),
                          std::end(long_string),
                          std::default_seacher{
                              std::begin(pattern),
                              std::end(pattern)}));
}
```

C++17

## LOCATING PATTERNS IN STRINGS WITH STD::SEARCH

```
int main()
{
   const std::string long_string {
                  "Lorem ipsum dolor sit amet, consetetur"
                  " sadipscing elitr, sed diam nonumy eirmod"};

   const std::string pattern {"elitr"};

   auto match (std::search(std::begin(long_string),
                           std::end(long_string),
                           std::boyer_moore_searcher{
                                std::begin(pattern),
                                std::end(pattern)}));
}
```

C++17

# LOCATING PATTERNS IN STRINGS WITH STD::SEARCH

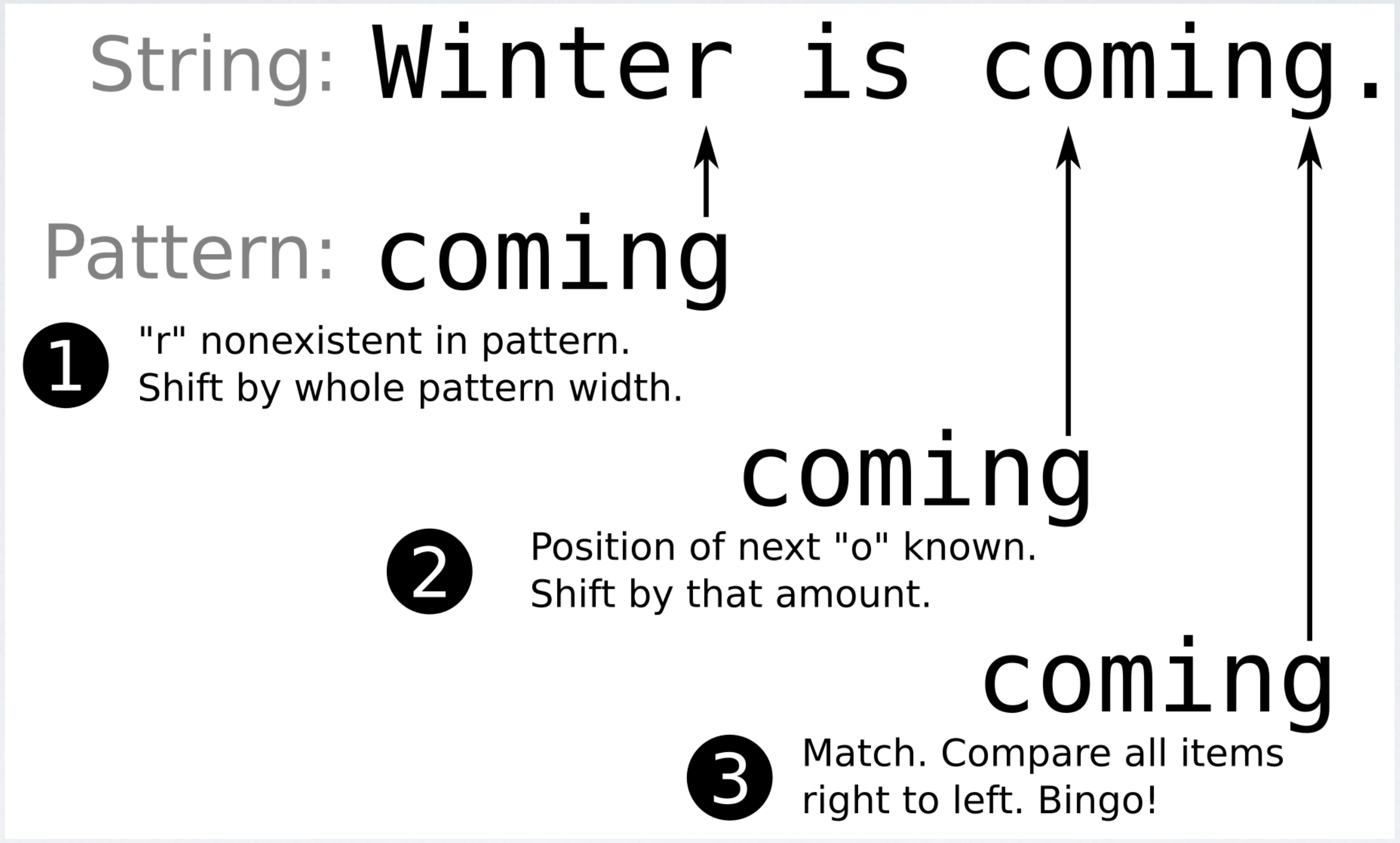
```
int main()
{
   const std::string long_string {
                  "Lorem ipsum dolor sit amet, consetetur"
                  " sadipscing elitr, sed diam nonumy eirmod"};

   const std::string pattern {"elitr"};

   auto match (std::search(std::begin(long_string),
                           std::end(long_string),
                           std::boyer_moore_horspool_searcher{
                                std::begin(pattern),
                                std::end(pattern)}));
}
```

C++17

# LOCATING PATTERNS IN STRINGS WITH STD::SEARCH


String: Winter is coming.
Pattern: coming
❶ "r" nonexistent in pattern.
Shift by whole pattern width.
coming
❷ Position of next "o" known.
Shift by that amount.
coming
❸ Match. Compare all items
right to left. Bingo!

# DICTIONARY MERGING TOOL

```
int main()
{
   std::deque<std::pair<std::string, std::string>> dict1;
   std::deque<std::pair<std::string, std::string>> dict2;
   std::deque<std::pair<std::string, std::string>> dstDict;
   //Init dict1, dict2
   ...
   //-----------------

   std::sort(std::begin(dict1), std::end(dict1));
   std::sort(std::begin(dict2), std::end(dict2));

   std::merge(std::begin(dict1), std::end(dict1),
          std::begin(dict2), std::end(dict2),
          std::back_inserter{dstDict});
}
```

## FILL CONTAINERS

```
int main()
{
   std::vector<int> v {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   std::fill(v.begin(), v.end(), -1);
}
```

# STD::GENERATE

```
int main()
{
   std::vector<int> v(5);
   std::generate(v.begin(), v.end(), std::rand);
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

inline bool isEven(int x) {
    return x % 2 == 0;
}

template <int N>
inline bool greaterThan(int x) {
    return x > N;
}

int main() {
    vector<int> x = { 1, 2, 3, 4, 5, 6, 7, 8 };

    if (!all_of(x.begin(), x.end(), isEven))
        cout << "Not all are even!" << endl;

    if (any_of(x.begin(), x.end(), isEven))
        cout << "But there is at least one even!" << endl;

    if (none_of(x.begin(), x.end(), greaterThan<10>))
        cout << "No number is > 10!" << endl;

    return 0;
}
```

## Checks by predicate

```cpp
#include <string>
#include <iostream>
#include <algorithm>
#include <cctype>

using namespace std;

int main() {
    string w("Dolly"), e("  \t\t \n   "), s("Hello Dolly!");

    if (all_of(w.begin(), w.end(), ::isalnum))
        cout << w << " is alphanumeric" << endl;

    if (all_of(e.begin(), e.end(), ::isspace))
        cout << "e is completely whitespace" << endl;

    cout << "Space in " << s << ": " <<
         count_if(s.begin(), s.end(), ::isspace) << endl;

    return 0;
}
```

**Same for strings**

```cpp
#include <iostream>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

struct less_than {
    less_than(int _value)
        : value(_value) {}

    bool operator()(int x) const {
        return x < value;
    }

    int value;
};

int main() {
    vector<int> x = { 1, 2, 3, 4 }, y;
    copy_if(x.begin(), x.end(), back_inserter(y), less_than(3));

    for (auto val: y)
        cout << val << endl;        // 1… 2

    return 0;
}
```

## Functor

## КОНЕЦ СЕДЬМОЙ ЛЕКЦИИ

```
const string str {"End of 7 lecture!"};

copy(begin(str), end(str),
        ostream_iterator<char>{cout});
```