

# Boost.Algorithm

Что, зачем и почему

Александр Зайцев

# Что попытаемся прояснить

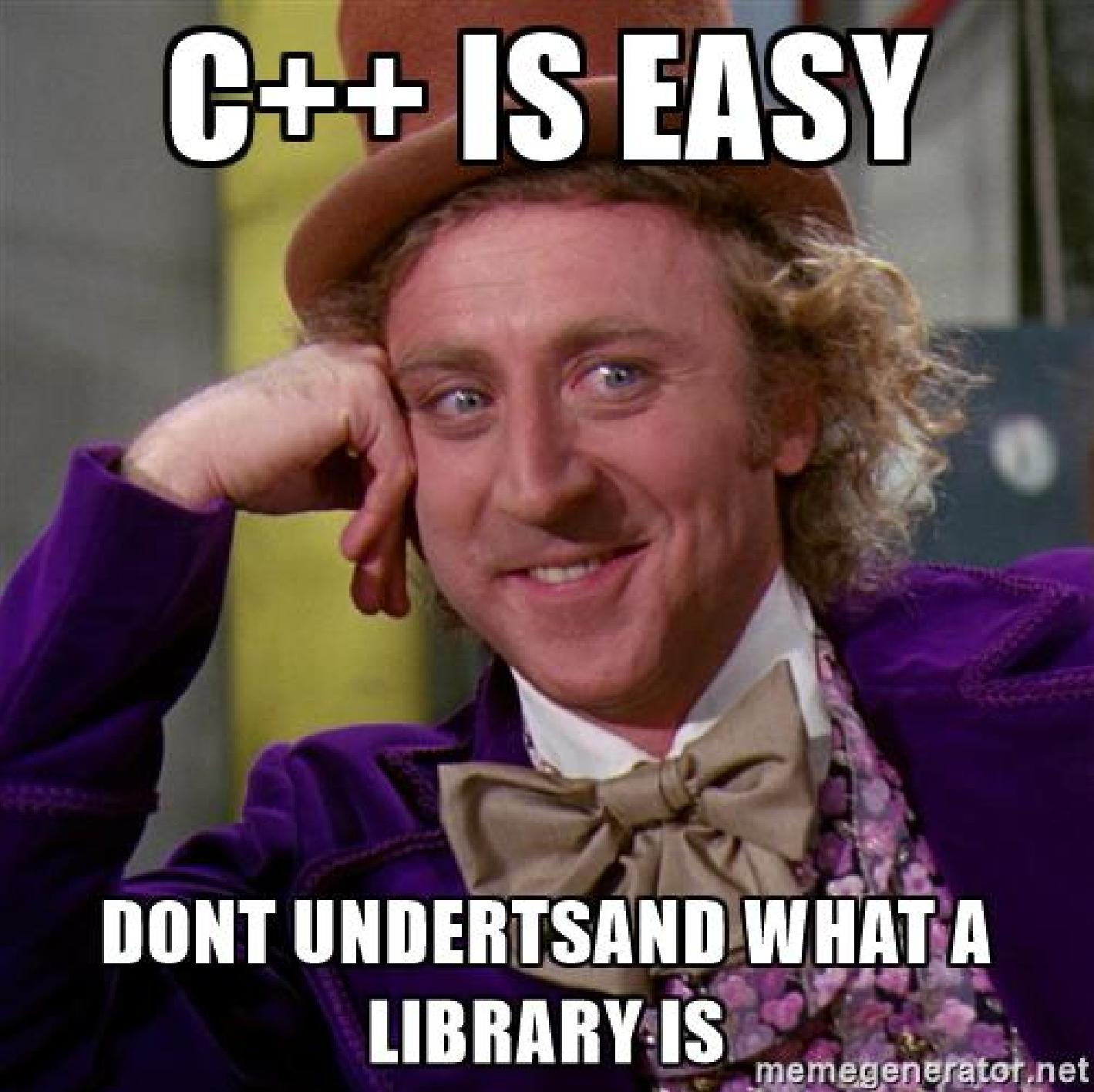
- ЧТО это такое
- ЗАЧЕМ оно нам надо (или не надо)
- ПОЧЕМУ было бы неплохо это использовать
- Также мы узнаем, что же есть и что появится интересного в этой библиотеке:)

# План

- Что есть сейчас
- А что \*возможно\* будет потом
- (Если останется время) Немного внутренней кухни

# Команда Boost.Algorithm

- Marshall Clow – автор и мэнтейнер, архитектор, председатель LWG, code owner for libc++, участник Boost Release Team, наверное что-то ещё и просто очень хороший человек.
- Александр Зайцев – какой-то студент
- Приходящие авторы



**C++ IS EASY**

**DONT UNDERSTAND WHAT A  
LIBRARY IS**

memegenerator.net

# Цель библиотеки

- Расширить функциональность <algorithm>
- Дать возможность пользователям на старых компиляторах юзать фичи, которые им пока что не завезли
- <algorithm> должен же откуда-то тянуть идеи :)

# Структура библиотеки

Алгоритмы делятся на:

- C++11
- C++14
- String algo
- Search
- Other

# Что есть из C++11?

- all\_of
- any\_of
- copy\_if
- copy\_n
- find\_if\_not
- iota
- is\_partitioned
- is\_permutation
- is\_sorted
- none\_of
- one\_of
- partition\_copy

# Что есть из C++14?

- equal
- is\_permutation
- mismatch

# Строковые алгоритмы

- `to_upper/to_lower`
- `trim/trim_left/trim_right`
- `join/join_if`
- `find_all/etc.`
- `starts_with/ends_with`
- `split`
- `replace functions`
- `erase functions`

# Split рукописный

```
template<typename Out>
void split(const std::string &s, char delim, Out result) {
    std::stringstream ss;
    ss.str(s);
    std::string item;
    while (std::getline(ss, item, delim)) {
        *(result++) = item;
    }
}

std::vector<std::string> split(const std::string &s, char delim) {
    std::vector<std::string> elems;
    split(s, delim, std::back_inserter(elems));
    return elems;
}
```

# Split приятный

```
std::vector<std::string> SplitVec;
split( SplitVec, "one:two::three",
       boost::is_any_of(":"),
       token_compress_on );
```

# Без категории

- clamp (C++17)
- gather
- hex
- is\_palindrome
- is\_partitioned\_until

# hex

```
hex( "abcdef", out ) --> "616263646566"
```

```
hex( "32", out ) --> "3332"
```

```
hex( "abcdef", wout ) --> "006100620063006400650066"
```

```
hex( "32", wout ) --> "00330032"
```

```
unhex( "616263646566", out ) --> "abcdef"
```

```
unhex( "3332", out ) --> "32"
```

```
unhex( "616263646566", wout ) --> "\6162\6364\6566" ( i.e, a 3 character string )
```

```
unhex( "3332", wout ) --> "\3233" ( U+3332, SQUARE HUARADDO )
```

```
unhex( "3", out ) --> Error - not enough input
```

```
unhex( "32", wout ) --> Error - not enough input
```

```
unhex( "ACEG", out ) --> Error - non-hex input
```

# Алгоритмы поиска

- Что уже есть:
  - Бойер-Мур (C++17)
  - Бойер-Мур-Хорспул (C++17)
  - Кнут-Моррис-Пратт (нет и не будет)

# Планы по развитию

- Оптимизация существующего поиска
- Умный boost::super\_search
- Нечёткий поиск
- Добавление constexpr
- Параллельность
- ...

# Зачем меняем поиск?

- std::search медленный (наивный алгоритм)
- Поисковые алгоритмы не стоят на месте

# Сравнение алгоритмов (тесты Маршалла)

Algorithm	At Start	Middle	At End	Not found
std::search	100.0	100.0	100.0	100.0
Boyer-Moore (procedural)	486.1	10.07	15.91	9.236
Boyer-Moore (object)	39.36	9.859	15.09	8.647
Boyer-Moore-Horspool (procedural)	154.2	9.111	12.12	8.113
Boyer-Moore-Horspool (object)	36.96	8.896	12.88	8.476

# Будущие алгоритмы поиска

- Что, возможно, будет в будущем:
  - Турбо Бойер-Мур
  - Тюнингованный Бойер-Мур
  - Быстрый поиск
  - Алгоритм Райта
  - **Мюссер-Нишанов**
  - И многое другого

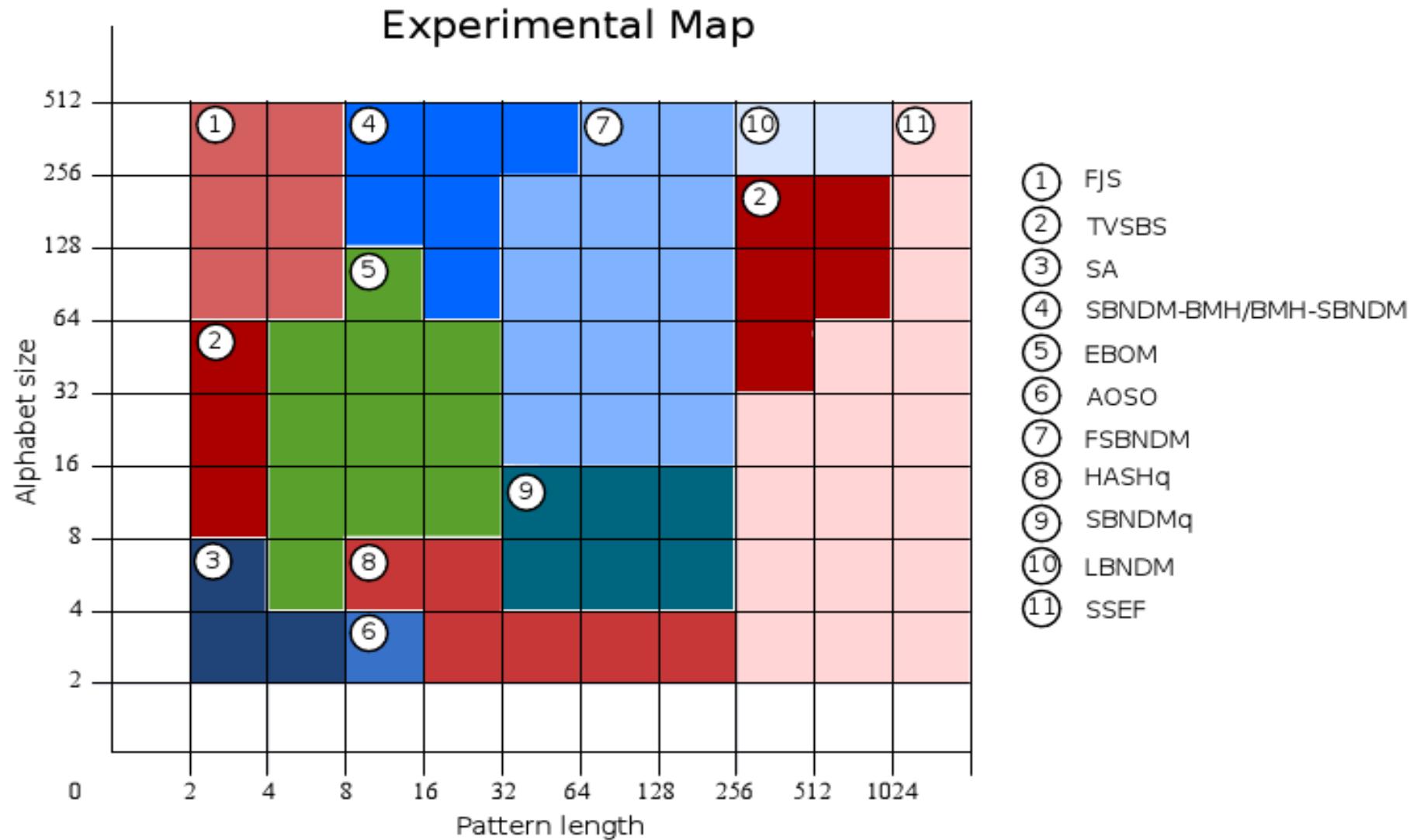
# Снова тесты

Название алгоритма	Размер паттерна	Размер алфавита
Shift-And	[1; 4]	[1; 4]
TVSBS	[1; 4], (32; 256]	[1; 4], [32; 128)
FJS	[1; 4]	[32; INF)
EBOM	(4; 32]	(128; INF)
SBNDM-BMH	(4; 32]	(128; INF)
HASHq	(4; 256]	[4; 32)
FSBNDM	(32; 256]	[32; INF)
SBNDMq	(32; 256]	[4; 32)
LBNDM	(256; INF)	(128; INF)
SSEF	(256; INF)	NA

# Снова тесты

Название алгоритма	Размер паттерна	Размер алфавита
Shift-And	[1; 4]	[1; 4]
TVSBS	[1; 4], (32; 256]	[1; 4], [32; 128)
FJS	[1; 4]	[32; INF)
EBOM	(4; 32]	(128; INF)
SBNDM-BMH	(4; 32]	(128; INF)
HASHq	(4; 256]	[4; 32)
FSBNDM	(32; 256]	[32; INF)
SBNDMq	(32; 256]	[4; 32)
LBNDM	(256; INF)	(128; INF)
SSEF	(256; INF)	NA

# Снова тесты(2)



# Мюссер-Нишанов

```
Corpus  is 2756252 entries long
----- Middle -----
Pattern is 105 entries long
          std::search 0.4848 seconds      100%
          boyer_moore_search 0.1171 seconds    24.16%
          boyer_moore_horspool_search 0.1076 seconds    22.2%
          musser_nishanov_search 0.1033 seconds    21.31%
----- End -----
Pattern is 43 entries long
          std::search 0.6206 seconds      100%
          boyer_moore_search 0.2075 seconds    33.44%
          boyer_moore_horspool_search 0.1742 seconds    28.07%
          musser_nishanov_search 0.1686 seconds    27.17%
--- Not found ---
Pattern is 91 entries long
          std::search 1.028 seconds      100%
          boyer_moore_search 0.1971 seconds    19.17%
          boyer_moore_horspool_search 0.1825 seconds    17.75%
          musser_nishanov_search 0.1702 seconds    16.56%
```

# **boost::algorithm::super\_search!**

- „Умный“ выбор алгоритма поиска
- Использование множества алгоритмов
- Конечно же будет возможность использовать алгоритмы напрямую

# Текущий статус super\_search

- Написаны реализации некоторых алгоритмов
- Проведена часть тестов по выявлению эффективности алгоритмов в различных случаях (тестирование продолжается)
- Более-менее стабилизирован интерфейс (но это не точно)

# Нечёткий поиск

- Господин Маршалл желает видеть это
- Хочет что-то вроде agrep
- Нужно проводить очень много исследований
- Статус фичи: только общие идеи, когда появится – очень скоро

# Оптимизация существующего поиска

- Попробовать сделать КМП на двунаправленных итераторах (сейчас RA). Перспективы – скорее всего не будет.
- Поиграться со структурами данных, лежащих под капотом БМ, БМХ. Перспективы – не туманны.

# constexpr

- Статус: есть pull request от Антона Полухина
- Маршаллу некогда посмотреть/смержить, поэтому Boost.Algorithm пока что без constexpr будет

# Параллельность

- Очень хотелось бы иметь параллельные алгоритмы „из коробки“
- Алгоритмы поиска сложно параллелятся
- Статус: идея в моей голове. Ведётся исследование по выполнимости и целесообразности этого в рамках Boost.Algorithm

# Полезные ссылки

- Repo: <https://github.com/boostorg/algorithm>
- My profile: <https://github.com/ZaMaZaN4iK>
- SMART:
  - <https://www.dmi.unict.it/~faro/smart/>
  - <https://github.com/smart-tool>



**std::exit (0);**

**Спасибо за внимание!**

**Вопросы?**



# Как придумываются фичи

- Команда списывается в IRC/почте
- Было бы неплохо, если бы <feature-X> появилась
- Спрашиваем у знакомых плюсовиков
- Спрашиваем в Boost mailing list (там игнор)
- Если отзывы положительные, пишем

# Проблемы

- Команда очень занята другими делами
- Люди быстро „сливаются“
- Время отклика Маршалла

# Оценка времени ожидания отклика Marshall Clow

Способ связи	Время ожидания	Перспективность способа
GitHub	Неделя - INF	Средняя
IRC	Сутки - неделя	Крайне высокая
Почта	INF	Надежда умирает последней

# Boost.Algorithm и <algorithm>

Лайфхак, как добавить что-то в <algorithm>:

- Пишем что-то годное в Boost.Algorithm
- Добиваемся, чтобы Маршалл посмотрел на это
- Вылизываем код, тесты, документацию, примеры
- Добиваемся, чтобы Маршалл посмотрел на это (2)
- Сливаемся в master ветку Boost.Algorithm
- Релизимся
- Пишем Маршаллу, что неплохо было бы это добавить в <algorithm>
- Готовим предложение в стандарт
- Добиваемся, чтобы Маршалл посмотрел на это (3)
- Маршалл соглашается и будет защищать вашу идею в Комитете

# Немного уличной магии

```
template <typename T>
struct has_find_all
{
    struct dummy { /* something */ };

    template <typename C, typename P>
    static auto test(P * p) ->
        decltype(std::declval<C>().find_all(*p, *p, *p), std::true_type{});

    template <typename, typename>
    static std::false_type test(...);

    typedef decltype(test<T, dummy>{nullptr}) type;
    static const bool value = std::is_same<std::true_type,
        decltype(test<T, dummy>{nullptr})>::value;
};
```