

Три с половиной variant'a

Потапов Павел

Архитектор ядра редакторов «МойОфис»

Компания «Новые облачные технологии»

Долгая компиляция



Зачем нужен variant

```
struct HueTransform;
```

```
struct InverseTransform;
```

```
...
```

```
using ColorTransform = boost::variant<
```

```
    HueTransform hueTransform,
```

```
    InverseTransform inverseTransform,
```

```
    ...
```

```
>;
```

```
std::vector<ColorTransform> transform;
```

- 28 типов трансформаций цветов
- Любое количество
- Любой порядок

- В массиве только один тип

- Тип-сумма

Альтернативы

- union
 - Не поддерживает типы с конструкторами и деструкторами
 - Можно хранить указатели
 - Нужно выделять и освобождать динамическую память
 - Нужно отдельно хранить тип текущего значения
- Иерархия объектов
 - Нужно выделять динамическую память
 - Нет общего базового типа

Основные реализации

Реализация	Стандарт C++
Boost.Variant	C++11 (C++98 до версии 1.83 включительно)
Std.Variant	C++17
Boost.Variant 2	C++11

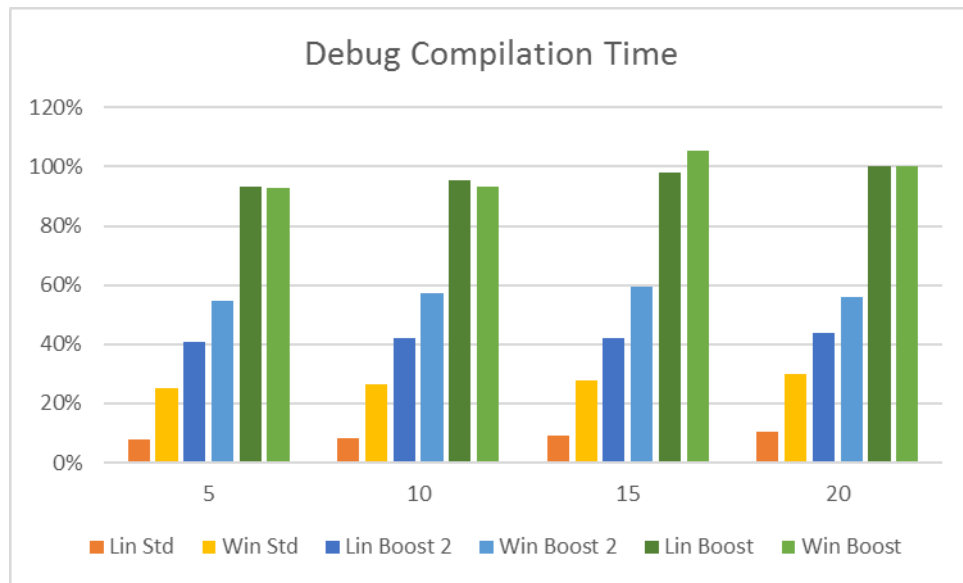
Измерение времени компиляции

- Clang 22.1.3
- Boost 1.83
- STL
 - VS 2022 (Windows)
 - libstdc++ 13.3 (Linux)

```
template <size_t N>
class A {
    double val;
};

struct VariantN20 :
    boost::variant<A<0>, A<1>, ..., A<19>>
{
};
```

Время компиляции в Debug

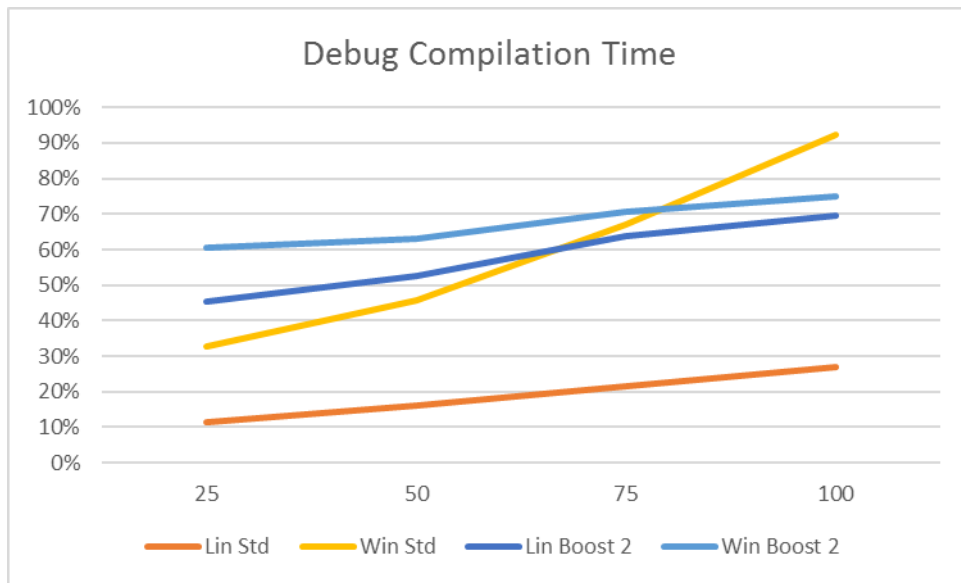


- `boost::variant<... 20 ...>` - 100%
- **Лидер**
Std.Variant
- **Аутсайдер**
Boost.Variant
- **STL**
Linux лучше Windows

Лидер vs аутсайдер

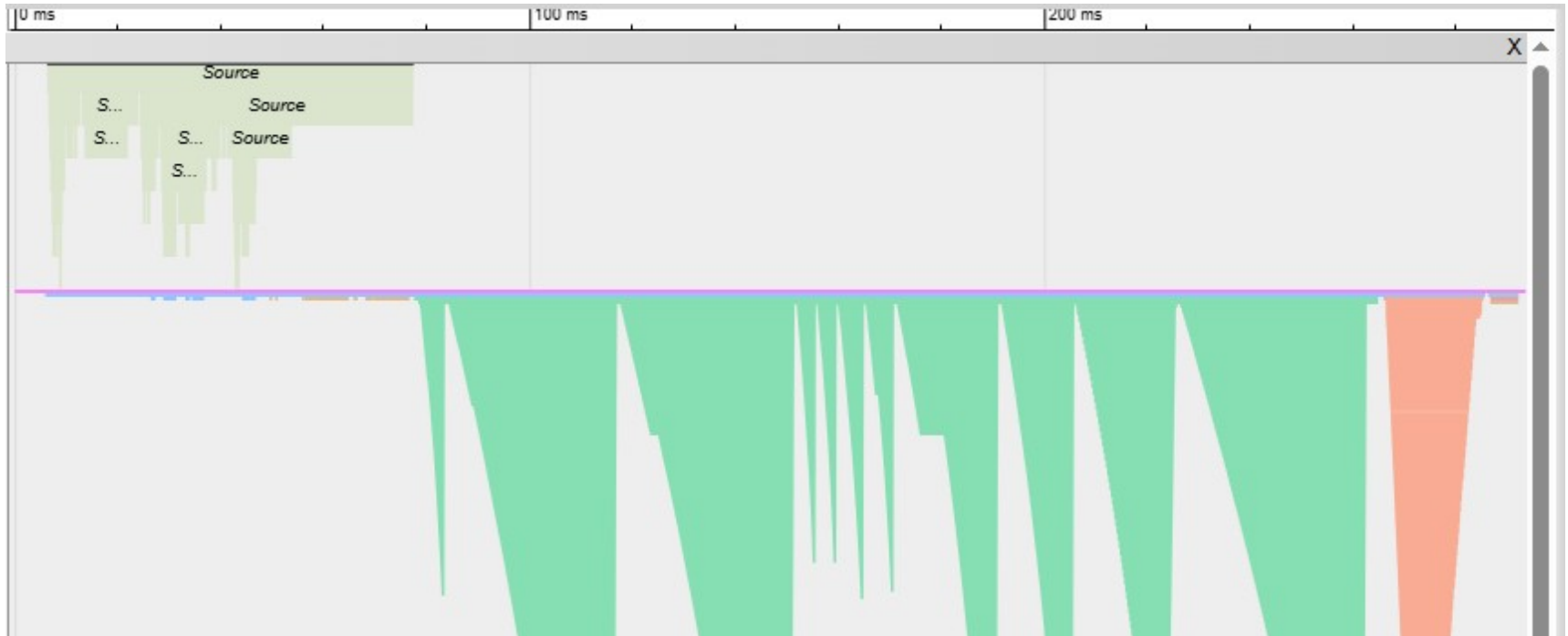


Время компиляции больших вариантов



- Файловые операции - фиксированы
- Время компиляции растёт
- Лидер роста
Std.Variant (Windows)

Std.Variant (VS 2022)



Компиляция: выводы

- Версия компилятора
- Скорость дисковой подсистемы
- Для `Std.Variant` - реализация STL

Призовые места в номинации «время компиляции»:

1. `Std.Variant`
2. `Boost.Variant 2`
3. `Boost.Variant`

Перспективный
вариант

Boost.Variant 2



Как реализованы?

```
template<class... VariantType>
class Variant {
    int currentIndex;
    std::uint8_t value[Size];
};
```

value[Size] - достаточно места для любого типа
currentIndex - индекс типа текущего значения

Контракт

- Всегда содержит значение одного из типов
Type X | Type Y | Type Z | ...
- По умолчанию создаётся со значением первого в списке типа
Может не быть конструктора по умолчанию
- Если нужно представить вариант без значения – отдельный тип
 - Boost.Variant – boost::blank
 - Std.Variant – std::monostate
 - Boost.Variant 2 – boost::variant2::monostate

В чём отличие?

Порядок операций при изменении значения на новое

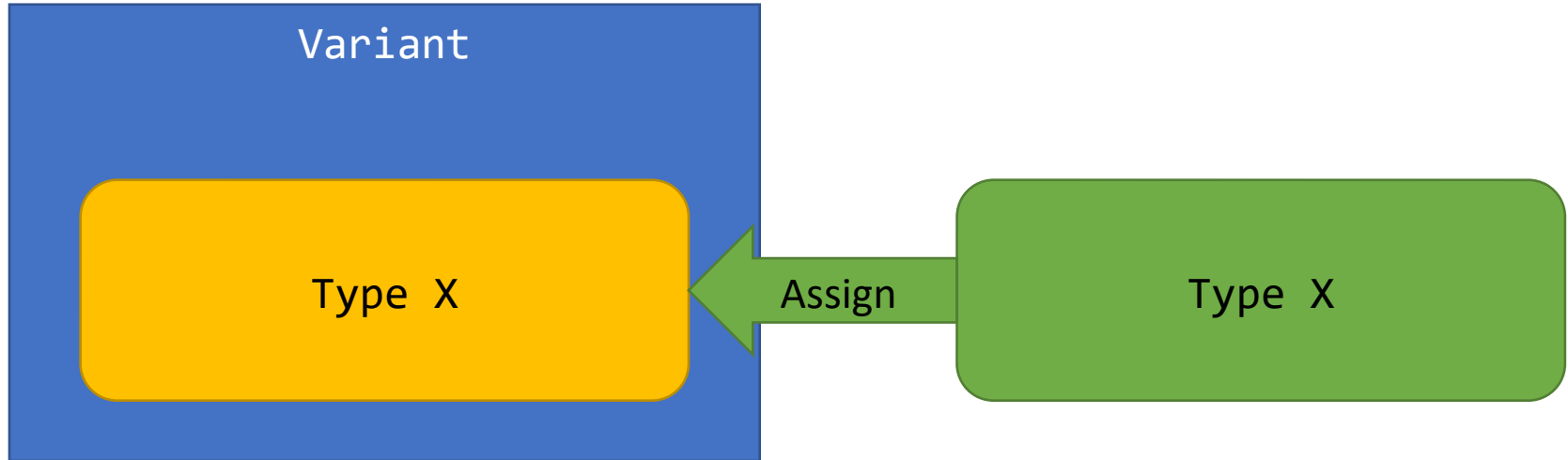
1. Разрушаем старое значение
2. Создаём новое значение

Что делать, если при создании выброшено исключение?

Boost.Variant – стратегии

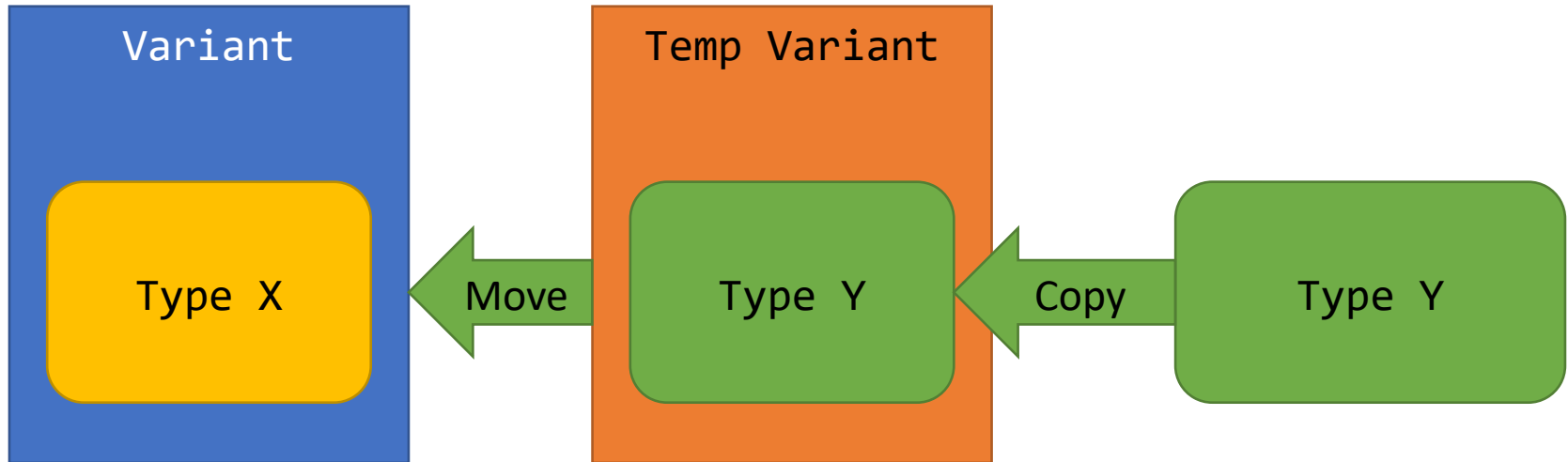
Родное присваивание

- Присваиваем тот же тип
Присваиваем текущему значению



Boost.Variant – стратегии Перемещение

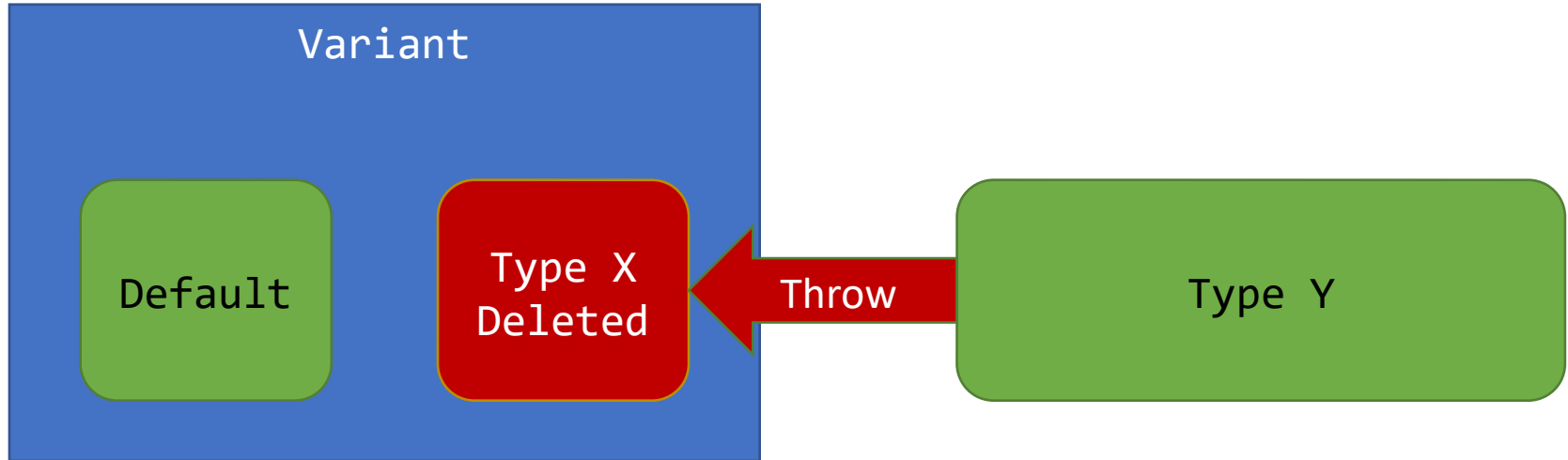
- У целевого типа конструктор перемещения `noexcept`
Создаём временный вариант и перемещаем его



Boost.Variant – стратегии

Объект по умолчанию

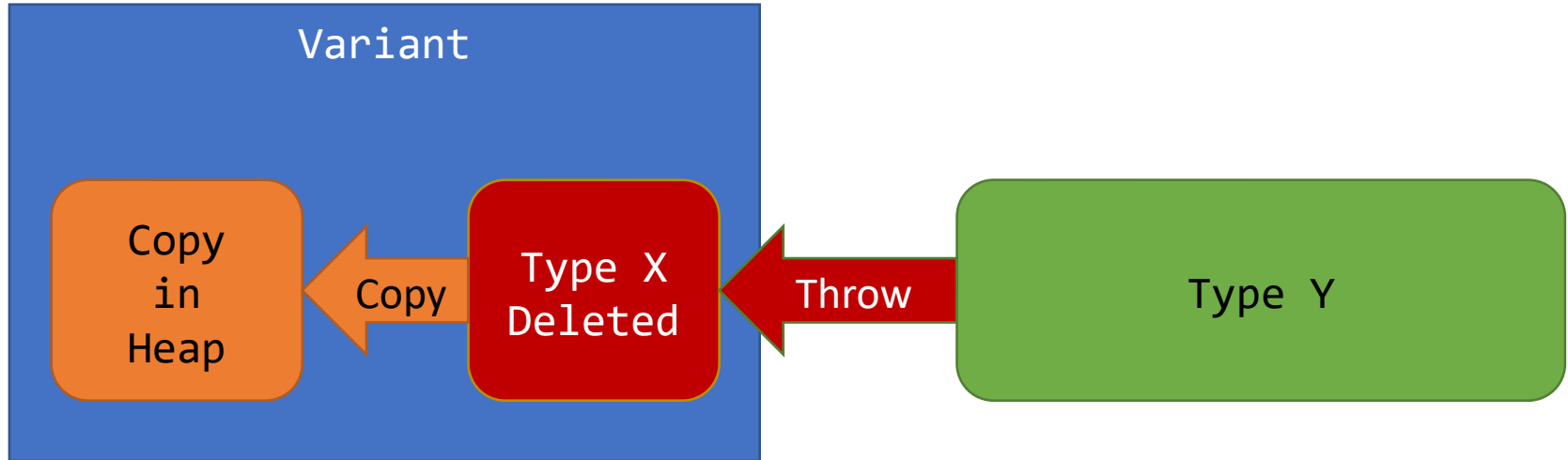
- Есть тип с конструктором по умолчанию `noexcept`?
При исключении создаём объект этого типа



Boost.Variant – стратегии

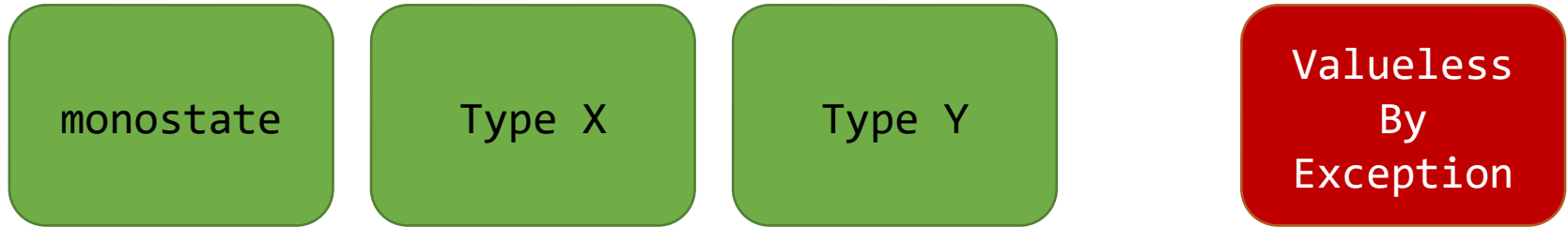
Копия в куче

- В крайнем случае
 Перед присваиванием копируем текущее значение в кучу



Std.Variant

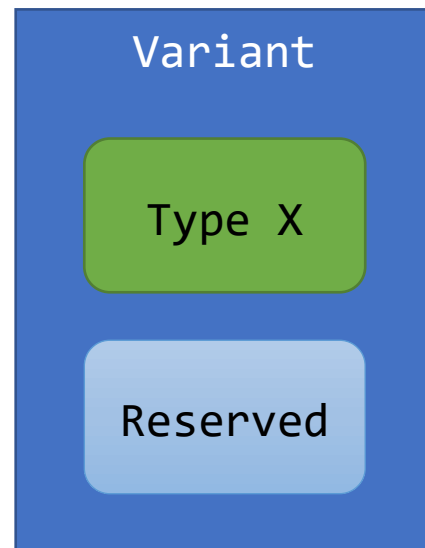
- Нельзя выделять память в куче
- Нельзя задействовать памяти больше необходимого
- Исключения при перемещении бывают редко



- “Valueless by exception” - допустимое состояние

Boost.Variant 2

- “Valueless by exception” – плохо
- Хотим “A never valueless variant type”
- Не хотим выделять память в куче
- Двойное хранилище (но только когда надо!)
- Строгая гарантия обработки исключений



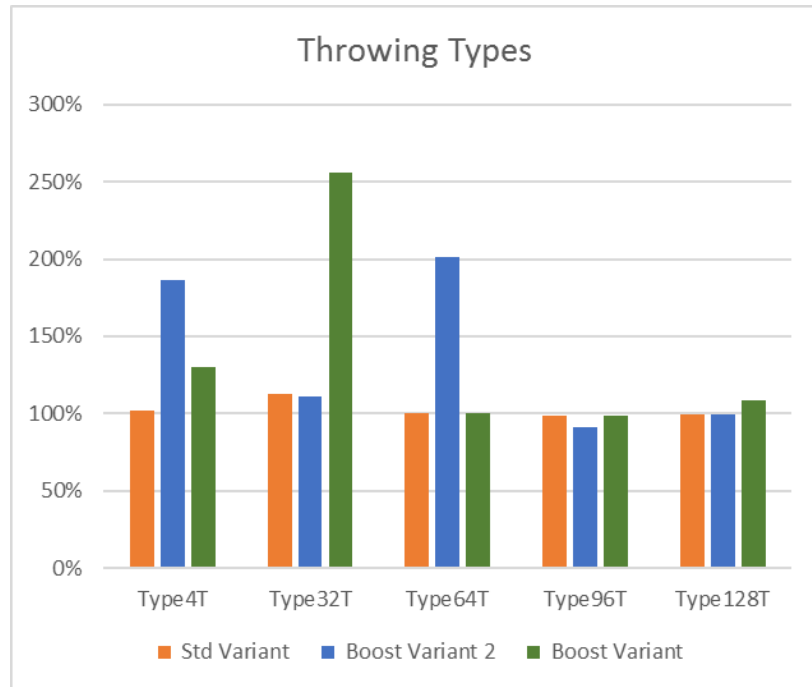
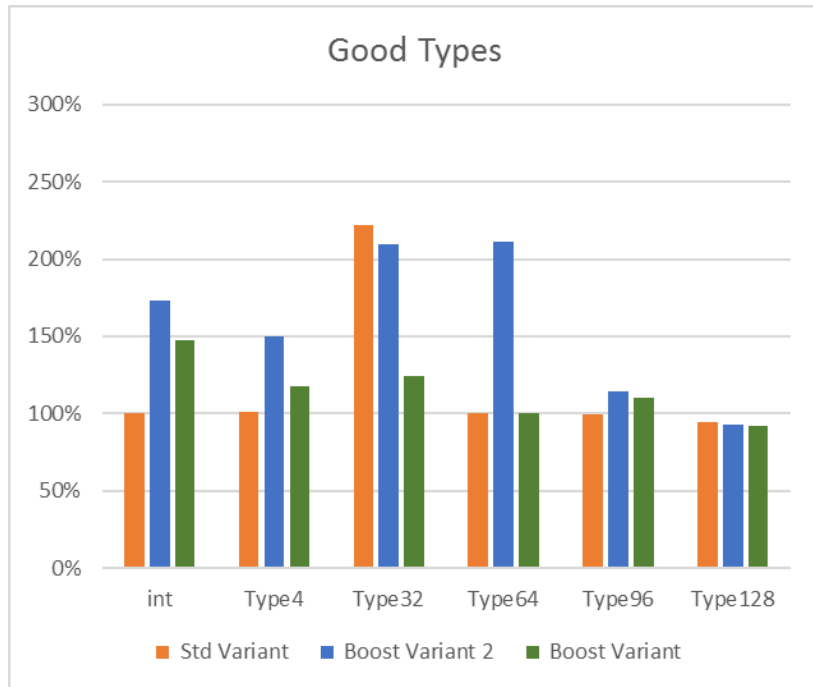
Измерения производительности



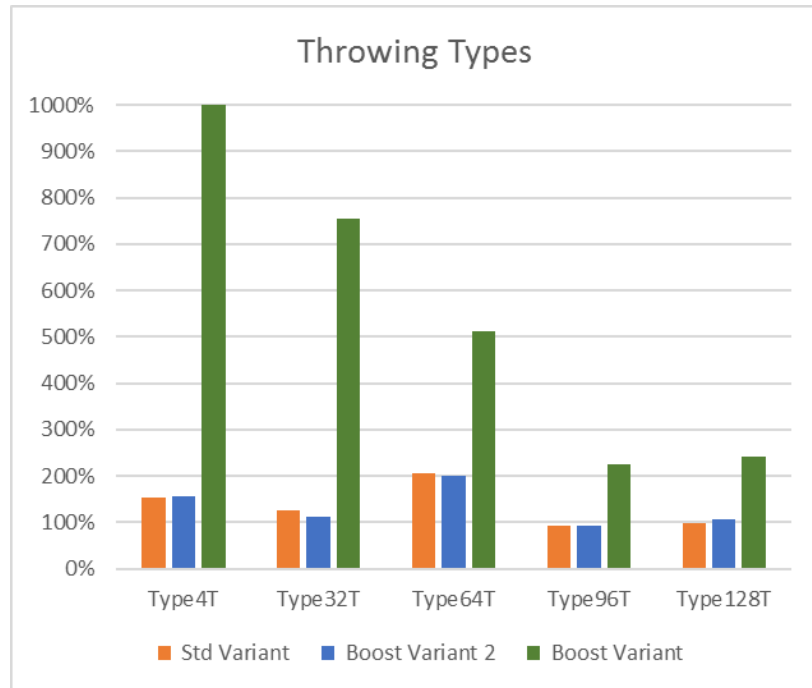
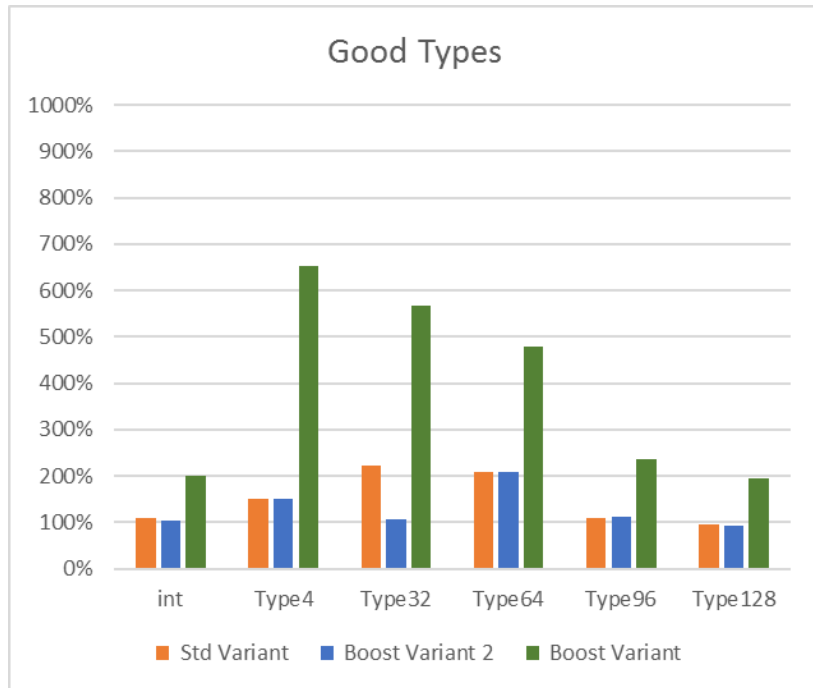
Подопытные варианты

- Вариант с «хорошими» типами
 - Boost.Variant использует стратегию перемещения
 - Boost.Variant 2 использует одинарное хранилище
- Вариант с «плохими» типами
 - Boost.Variant использует стратегию копии в куче
 - Boost.Variant 2 использует двойное хранилище
- Поведение Std.Variant всегда одинаково
- Типы разных размеров: 4 байта, 32, 64, 96, 128

Присваивание значения того же типа



Присваивание значения другого типа



Производительность: ВЫВОДЫ

- `Std.Variant` и `Boost.Variant 2` – можно брать любой
- `Boost.Variant` – если нет других вариантов

Новый вариант

`boost::variant2`



Проблемный код

```
struct Result {  
    Variant<A, B> v;  
    AnotherValue* p;  
};  
  
Result makeResult();  
  
void test() {  
    Result r = makeResult();  
  
    r.p->callMeToCrash(); // и вот здесь всё падает  
}
```

ВИНОВНИК

```
// TestType.h
template <bool>
struct TestType {
    TestType();
    TestType(TestType&&);

    std::int64_t val[4];
};

using BadVariant =
    boost::variant2::variant<
        TestType<true>,
        TestType<false>
    >;
```

```
// TestType.cpp
template <bool X>
TestType<X>::
    TestType(TestType&&) = default;

template class TestType<true>;
template class TestType<false>;

Result makeResult() {
    return Result{TestType<true>{}}, ...};
}
```

Защита на будущее

```
template <bool UseDoubleStorage, typename BaseVariant>
class VariantImpl : public BaseVariant {

    static_assert(BaseVariant::uses_double_storage() == UseDoubleStorage);

};

template <typename... Alternatives>
using Variant = VariantImpl<false, boost::variant2::variant<Alternatives...>>;

template <typename... Alternatives>
using DoubleStorageVariant = VariantImpl<true, boost::variant2::variant<Alternatives...>>;
```

Рекомендации

1. Стандарт

- C++17 и новее – Std.Variant или Boost.Variant 2
- C++11/14 – Boost.Variant 2
- до C++ 11 – Boost.Variant из Boost 1.83

2. Boost не используется – Std.Variant

3. Строгая гарантия обработки исключений – Boost.Variant 2

4. Быстрая компиляция – Std.Variant и сборка на Linux

! Boost.Variant 2 – расстановка поехсепт и (опционально) проверка размера

Благодарности

Помощь в измерениях на Линуксе

- Александр Петров
- Дмитрий Шубин

Q & A