BACK TO BASICS:
VIRTUAL DISPATCH AND ITS ALTERNATIVES

INBAL LEVI

LOST IN TRANSLATION

OVERVIEW

- Define the goal
- Understand the virtual table
- Go over (some of) the alternatives
- Benchmark!
- Draw conclusions

THE GOAL

- Understand what happens behind the scenes - use it to improve performance
- Identify the type of our object, and “do something” accordingly
- To implement a "Derived is a Base" relation

IMPROVE (RUNTIME) PERFORMANCE

- Optimize hot path (80%) most probable case first, pass by ref, etc.
- Decrease:
  - Dynamic allocation
  - System calls
  - Runtime decision making → Branching
  - Function calls (*Use inline)
  - Code size → Cache-miss

WARM UP

```
class Base {
public:
    Base() {
        cout << "Base Ctor" << endl;
    }
    ~ Base() {
        cout << "Base Dtor" << endl;
    }
    void printMe() {
        cout << "Hi, Base" << endl;
    }
};
```

```
class Derived : public Base {
public:
    Derived() {
        cout << "Derived Ctor" << endl;
    }
    ~ Derived() {
        cout << "Derived Dtor" << endl;
    }
    void printMe() {
        cout << "Hi, Derived" << endl;
    }
};
```

```
void func(Base *b)
{
    b->printMe();
}
int main()
{
    Derived d;
    func(&d);
}
```

Hi, Base

WARM UP

```
class Base {
public:
	Base() {
		cout << "Base Ctor" << endl;
	}
	~ Base() {
		cout << "Base Dtor" << endl;
	}
	void printMe() {
		cout << "Hi, Base" << endl;
	}
};
```

```
class Derived : public Base {
public:
	 Derived() {
		cout << "Derived Ctor" << endl;
	}
	~ Derived() {
		cout << "Derived Dtor" << endl;
	}
	void printMe() {
		cout << "Hi, Derived" << endl;
	}
};
```

```
int main()
{
	Base *d = new Derived;
	func(&d);
	delete d;
}
```

Base Ctor
Derived Ctor
Hi, Base
Base Dtor

WARM UP

```
class Base {
public:
        Base() {
                cout << "Base Ctor" << endl;
        }
        virtual ~Base() {
                cout << "Base Dtor" << endl;
        }
protected:
        virtual void printMe() {
                cout << "Hi, Base" << endl;
        }
};
```

```
class Derived : public Base {
public:
        Derived() {
                cout << "Derived Ctor" << endl;
        }
        ~ Derived() {
                cout << "Derived Dtor" << endl;
        }
        void printMe() {
                cout << "Hi, Derived" << endl;
        }
};
```

```
int main()
{
        Base *b = new Derived;
        delete b;
}
```

Base Ctor
Derived Ctor
Derived Dtor
Base Dtor

WARM UP

```
class Base {
public:
        Base() {
                cout << "Base Ctor" << endl;
        }
        virtual ~Base() {
                cout << "Base Dtor" << endl;
        }
        virtual void printMe() {
                cout << "Hi, Base" << endl;
        }
};

int main()
{
        Derived d;
        func(&d);
}
```

```
class Derived : public Base {
public:
        Derived() {
                cout << "Derived Ctor" << endl;
        }
        ~ Derived() {
                cout << "Derived Dtor" << endl;
        }
        void printMe() {
                cout << "Hi, Derived" << endl;
        }
};
```

Base Ctor
Derived Ctor
Hi, Derived
Derived Dtor
Base Dtor

DYNAMIC BINDING

- Implementation details are not covered by the standard
- Common implementation: vtable

THE EXAMPLE

- A Picture of: 10^8 = 10000 x 10000 pixels
- X86 Linux Debian machine
- Apply filters:
  - Filter is operating on one pixel
  - Dispatch “Activate” function

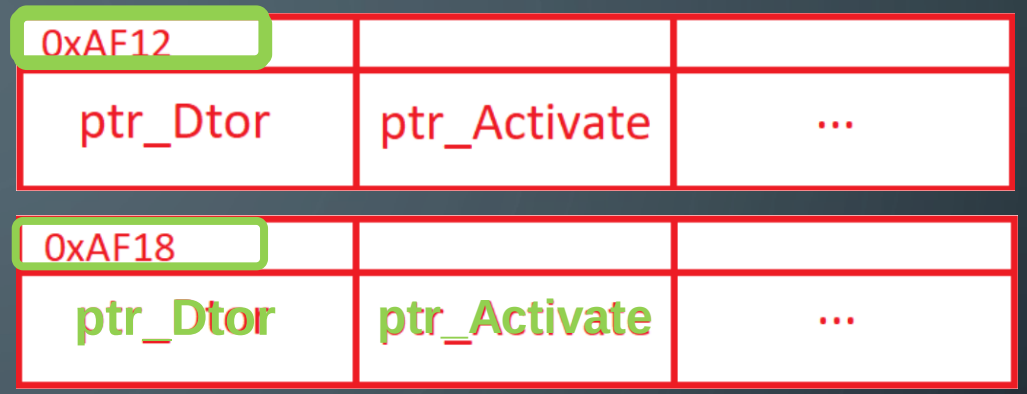
TOOLS

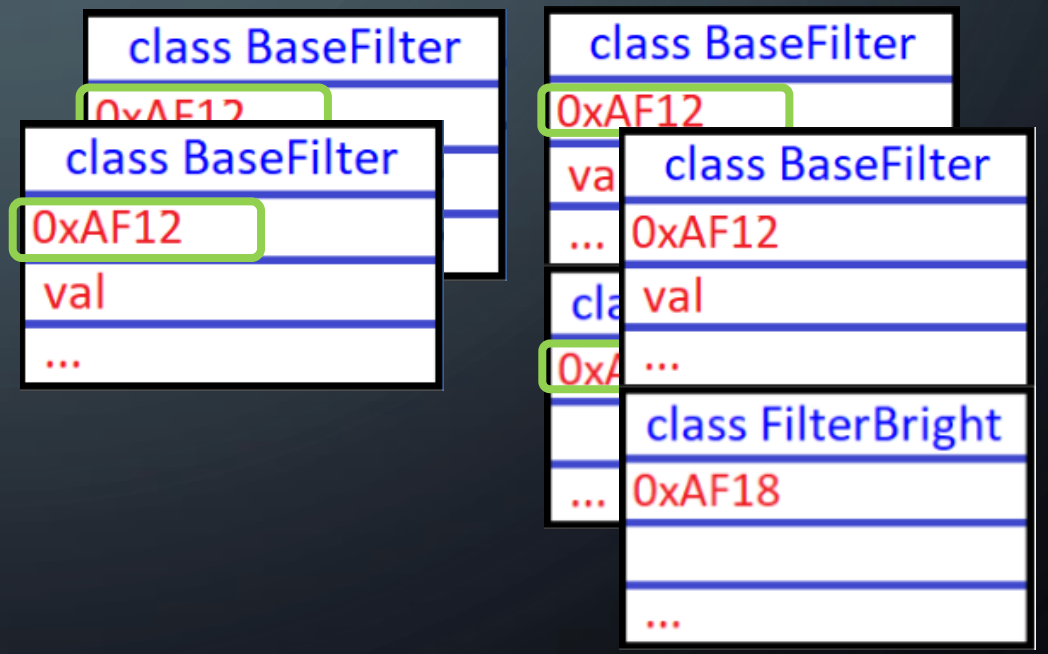
- godbolt.org
- cppinsights.io
- Quick-bench.com
- google/benchmark
- clock_t()

VTABLE - INTRODUCTION

```
class BaseFilter
{
public:
        virtual void Activate(PIXEL *pixel) const  {
                cout << "BaseFilter" <<endl;
        }
        virtual ~BaseFilter();
        ...
        char val;
};
```

```
class FilterBright : public BaseFilter
{
public:
        void Activate(PIXEL *pixel) const override {
                *pixel += val;
        }
        ~ FilterBright();
};
```





VTABLE - STRUCTURE

```
class BaseFilter
{
public:
        virtual void Activate(PIXEL *pixel) const  {
                cout << "BaseFilter" <<endl;
        }
        virtual ~BaseFilter();
        ...
        char val;
};
class FilterBright : public BaseFilter
{
public:
        void Activate(PIXEL *pixel) const override {
                *pixel += val;
        }
        ~ FilterBright();
};
```

RTTI
&
dynamic_cast
Type information

```
vtable for FilterBright:
        .quad   0
        .quad   typeinfo for FilterBright
        .quad   FilterBright::~FilterBright() [Complete Dtor]
        .quad   FilterBright::~FilterBright() [Deleting Dtor]
        .quad   FilterBright::Activate(unsigned char*)
vtable for BaseFilter:
        .quad   0
        .quad   typeinfo for BaseFilter
        .quad   BaseFilter::~BaseFilter() [Complete Dtor]
        .quad   BaseFilter::~BaseFilter() [Deleting Dtor]
        .quad   BaseFilter::Activate(unsigned char*)
typeinfo for FilterBright:
        .quad   vtable for __cxxabiv1::__si_class_type_info+16
        .quad   typeinfo name for FilterBright
        .quad   typeinfo for BaseFilter
typeinfo name for FilterBright:
        .string "12FilterBright“
typeinfo for BaseFilter:
        .quad   vtable for __cxxabiv1::__class_type_info+16
        .quad   typeinfo name for BaseFilter
typeinfo name for BaseFilter:
        .string "10BaseFilter"
```

-fno-rtti

VTABLE - IMPLEMENTED CODE

```
class BaseFilter
{
public:
    FilterBright f1;
    virtual void Activate(PIXEL *pixel) const {
    // f1->BaseFilter CTOR:
    f1.vtable = &BaseFilter_vtable_instance;
    virtual ~BaseFilter();
    // f1-> FilterBright CTOR:
    f1.vtable = &FilterBright_vtable_instance;
};  ...
class FilterBright : public BaseFilter
{     (f1.vtable->Activate)(&f1, p1);
public:
    void Activate(PIXEL *pixel) const override {
    (f1.vtable->BaseFilter_Dtor)(&f1);
}     }
    ~ FilterBright();
};
```

```
struct FilterBright_vtable
{
    void (*Dtor)(FilterBright *_this);
    void (*Activate)(FilterBright *_this, int *pixel);
};
```

```
struct FilterBright
{
    const FilterBright_vtable *vtable;
};
```

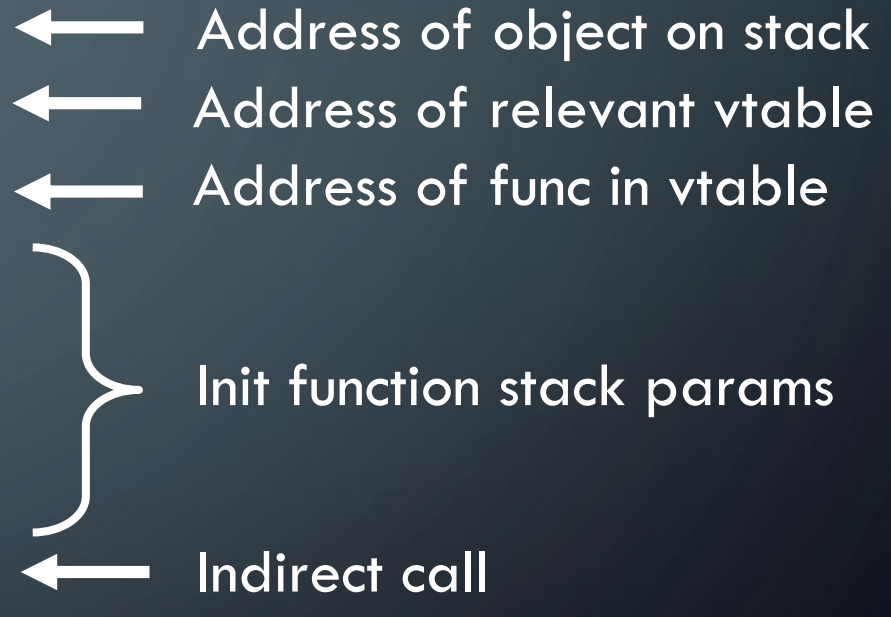
```
void FilterBright_Dtor(FilterBright *_this) {}

void FilterBright_Activate(FilterBright *_this, int *pixel)
{
    *p1+=1;
}
```

```
static const FilterBright_vtable FilterBright_vtable_instance =
{
    &FilterBright_Dtor,
    &FilterBright_Activate
};
```

VTABLE – RUN TIME

The call for "Activate" function

```
mov     QWORD PTR [rbp-24], rbx
mov     rax, QWORD PTR [rbp-24]
mov     rax, QWORD PTR [rax]
mov     rdx, QWORD PTR [rax]
mov     rax, QWORD PTR [rbp-24]
mov     esi, 4
mov     rdi, rdx
call    rax BaseFilter<FilterDerived>::Activate(int*)
```



VTABLE - OVERHEAD

- Retrieve and calculate the relative location of the function.
- Dereference the pointer(s) to the virtual table(s).
- Execute Indirect call (jmp / call).
- (optional) Adjust the value of this pointer.

Solutions:

Minimize the overhead of VT by increasing localization, decreasing dereferences, etc.

- VT alternative implementations
- Dyno by Louis Dionne

Minimize the use of VT by implementing polymorphism with alternative mechanisms.

- Template meta-programming
- Compile time decision making

VTABLE - FEATURES

- Don't Repeat Yourself DRY
- RunTime Binding RTB
- Allows Override Functions in the Derived OFD
- Multi-Level Inheritance MLI
- Add External Derived AED
- Multiple inheritance

VTABLE - THE ALTERNATIVES

- Solutions should provide a structure, with as little overhead as possible.
- Trade flexibility with performance – from **Runtime** to **Compile time**.
- Some of the solutions are only possible if we have all the types known.

THE ALTERNATIVES - SUBCLASSING

Non virtual inheritance

```
class Base
{
public:
    Base()
    {
        cout<< "Base Ctor“ <<endl;
    }
    void printMe()
    {
        cout<< "Hi, Base“ <<endl;
    }
protected:
    ~Base()
    {
        cout<< "Base Dtor“ <<endl;
    }
};
```

```
class Derived : public Base
{
public:
    Derived()
    {
        cout<< "Derived Ctor“ <<endl;
    }
    void printMe()
    {
        cout<< "Hi, Derived“ <<endl;
    }
    ~ Derived()
    {
        cout<< "Derived Ctor“ <<endl;
    }
};
```

```
int main ()
{
    Derived d;
    d.printMe();
}
```

Base Ctor
Derived Ctor
Hi, Derived
Derived Dtor
Base Dtor

THE ALTERNATIVES - SUBCLASSING

```
class Base
{
public:
        Base(int a = 1, int b = 1): A(a) , B(b) {}
        void Print()
        {
                cout << "getA: " << getA() << endl;
                cout << "getB: " << getB() << endl;
        }
        int getA()      {   return A;   }
        int getB()      {   return B;   }

        int A;
        int B;
};
```

```
class Derived : public Base
{
public:
        void PrintCaller ()
        {
                Print();
        }
        int getA()      {   return A*5;   }
        int getB()      {   return B*5;   }
};
```

```
int main()
{
        Derived d;
        d.PrintCaller();
}
```

getA: 1
getB: 1

DRY ~~RTB~~ ~~OFD~~ MLI AED

THE ALTERNATIVES - CRTP

```
template <typename D>
class BaseFilter
{
public:
        void Activate()
        {
                D& derived = static_cast<D&>(*this);
                derived.derivedActivate();
        }
        . . .
};

int main()
{
        BaseFilter<FilterBright> f1;
        f1.Activate();
        BaseFilter<FilterDark> f2;
        f2.Activate();
}
```

Activate Bright
Activate Dark

```
class FilterBright : public BaseFilter <FilterBright>
{
public:
        void derivedActivate()
        {
                cout << "Activate Bright" << endl;
        }
};

class FilterDark : public BaseFilter <FilterDark>
{
public:
        void derivedActivate()
        {
                cout << "Activate Dark" << endl;
        }
};
```

THE ALTERNATIVES - CRTP

```
class BaseSubclass {
public:
        BaseSubclass(int a = 1, int b = 1): A{a} , B{b} {}
        void Print() {
                cout << "getA: " << getA() << endl;
                cout << "getB: " << getB() << endl;
        }
        int getA()  {     return A;      }
        int getB()  {     return B;      }
        int A;
        int B;
};
class DerivedSubclass : public BaseSubclass {
public:
        void PrintCaller ()  {    Print();    }
        int getA()  {      return A*5;      }
        int getB()  {      return B*5;      }
};

int main()
{
        DerivedSubclass d;
        d.PrintCaller();
}
```

getA: 1
getB: 1

```
template <typename D> class BaseCRTP  {
public:
        BaseCRTP (int a = 1, int b = 1): A{a} , B{b}{}
        void Print() {
                cout << "getA: " << static_cast<D&>(*this).getA() << endl;
                cout << "getB: " << static_cast<D&>(*this).getB() << endl;
        }
        int getA() {     return A;      }
        int getB() {     return B;      }
        int A;
        int B;
};
class DerivedCRTP : public BaseCRTP<DerivedCRTP>  {
public:
        void PrintCaller () {    Print();     }
        int getA()  {      return A*5;      }
        int getB()  {      return B*5;      }
};

int main()
{
        DerivedCRTP d;
        d.PrintCaller();
}
```

getA: 5
getB: 5

THE ALTERNATIVES - CRTP

```
template <typename D>
class BaseFilter {
public:
    void Activate()  {
        static_cast<D&>(*this) .Activate();
    }
protected:
    ~BaseFilter() = default;
};

class FilterDerived : public MiddleFilter<FilterDerived> {
public:
    void Activate()  {
        cout << "Derived Activate" << endl;
    }
};
```

```
template <typename D = void>
class MiddleFilter : public BaseFilter<MiddleFilter<D>> {
public:
    void Activate()  {
        Activate_impl (std::is_same<D, void>{});
    }
private:
    void Activate_impl (std::true_type)  {
        cout << "Middle Activate" << endl;
    }

    void Activate_impl (std::false_type)  {
        static_cast<D&>(*this).Activate();
    }
};
```

THE ALTERNATIVES - CRTP

```
template <typename D>
class BaseFilter {
public:
        void Activate()  {
                static_cast<D&>(*this) .Activate();
        }
protected:
        ~BaseFilter() = default;
};

class FilterDerived : public MiddleFilter<FilterDerived> {
public:
        void Activate()  {
                cout << "Derived Activate" << endl;
        }
};
```

```
int main()
{
        MiddleFilter<> f;
        f.Activate();
}
```

Middle Activate

```
template <typename D = void>
class MiddleFilter : public BaseFilter<MiddleFilter<D>> {
public:
        void Activate()  {
                Activate_impl (std::is_same<D, void>{});
        }
private:
        void Activate_impl (std::true_type)  {
                cout << "Middle Activate" << endl;
        }

        void Activate_impl (std::false_type)  {
                static_cast<D&>(*this).Activate();
        }
};
```

```
int main()
{
        FilterDerived f;
        f.Activate();
}
```

Derived Activate

THE ALTERNATIVES – CRTP - CONCLUSION

```
template <class FilterCRTP>
class BaseFilterCRTP
{
public:
      void Activate(int *pixel)  {
            static_cast <FilterCRTP&>(*this).ImplementActivate(pixel);
       }
};
class FilterCRTP : public BaseFilterCRTP<FilterCRTP>
{
public:
      void ImplementActivate(int *pixel) {
            *pixel -=1;
       }
};
```

```
int main()
{
      FilterCRTP fc;
      FilterCRTP *p_fc = &fc;
      p_fc ->Activate(&pixel);
}
```

~~DRY~~ ~~RTB~~ ~~OFD~~ ~~MLI~~ ~~AED~~

THE ALTERNATIVES - CRTP

Subclassing	CRTP
Easy to read, understand & implement	Not trivial for use
Inheritance of more than one descendant is easy and clear	Inheritance of more than one descendant is hard
Derived can only HIDE Base member functions.	Derived can OVERRIDE Base member functions.

NOTICE: Both only allow to make decisions on Compile Time.

THE ALTERNATIVES - VISITOR

std::variant

```
template <class... Types>
class variant;
```

std::visit

```
template <class Visitor, class... Variants>
constexpr auto visit(Visitor&& vis, Variants&&... vars);
```

Lambda

```
[ captures ]<tparams> ( params ) -> ret { body }
```

THE ALTERNATIVES - VISITOR

```
struct FilterBright { };
struct FilterDark { };
```

```
struct VisitActivate
{
    void operator()(FilterBright& filter) { cout << "Filter Bright"; }
    void operator()(FilterDark& filter) { cout << "Filter Dark"; }
};
```

```
int main()
{
    std::variant <FilterBright, FilterDark> MyFilter { FilterBright() };
    std::visit (VisitActivate(), MyFilter);
}
```

Filter Bright

THE ALTERNATIVES - VISITOR

```
template<typename V, typename... Variants>                    // variadic template
constexpr decltype(auto) visit(V&& visitor, Variants&&... variants)
{
    if ((variants . valueless_by_exception() || ...))        // error handling
        throw_bad_variant_access("Unexpected index");

    using resultType = decltype (std::forward<V>(visitor)    // recursive variant extraction
    (std::get<0>(std::forward<Variants>(variants))...));

    constexpr auto& variantVtable = detail::variant::gen_vtable   // creation of vtable
        <resultType, V&&, Variants&& . . . >::S_vtable;

    auto funcPtr = variantVtable.M_access(variants.index()...);   // retrieve funPtr from index

    return (*funcPtr)(std::forward<V>(visitor), std::forward<Variants>(variants)...);
}
```

THE ALTERNATIVES - VISITOR

Anonymous

```
template<class... Ts> struct overload : Ts... { using Ts::operator()...; };
template<class... Ts> overload(Ts...) -> overload<Ts...>;

int main()
{
    std::variant<FilterBright, FilterDark, . . . > Filter{FilterDark()};

    std::visit (overload {
        [](FilterBright& )  { cout << "Filter Bright"; },
        [](FilterDark& )   { cout << "Filter Dark";   },
        . . .
    }, Filter);
}
```

THE ALTERNATIVES - VISITOR

Multi variant

```
std::variant<FilterBright, FilterDark> MyFilter1 {FilterDark()};
std::variant<FilterBright, FilterDark> MyFilter2 {FilterBright()};

std::visit(overload{
      [](FilterBright&, FilterBright& ) { cout << "FilterBright & FilterBright"; },
      [](FilterBright&, FilterDark& ) { cout << "FilterBright & FilterDark"; }
}, MyFilter1, MyFilter2);
```

FilterDark
&
FilterBright

THE ALTERNATIVES - VISITOR

```
std::variant<FilterBright, FilterDark> MyFilter1 {FilterDark()};
std::variant<FilterBright, FilterDark> MyFilter2 {FilterBright()};

std::visit(overload{
      [](FilterBright&, Pixel& pixel) { cout << "FilterBright"; pixel += 1; },
      [](FilterDark&, Pixel& pixel) { cout << "FilterDark"; pixel -= 1; }
}, FilterBright, Pixel&);
```

Before: 5
After: 6

THE ALTERNATIVES - COMPILE TIME VISITOR

```
std::variant<FilterBright, FilterDark>
Filter{FilterDark()};
std::variant<Pixel> pixel = 5;

for(int i = 0 ; I < 1000 ; ++i)
{
      std::visit (VisitActivate(), Filter, pixel);
      std::variant<int> var{i};
      std::visit (CTAVisitActivate(), Filter, var);
      photoPixelAfter[i] = (var);
}
```

THE ALTERNATIVES – CONSTEXPR ?

A constexpr function must satisfy the following requirements:

- It must not be **virtual.** (until C++20)
- Its return type must be a LiteralType.
- Each of its parameters must be a LiteralType.
- The function body must not contain:
  - A definition of a variable of non-literal type.

A constexpr constructor must satisfy the following requirements:

- Each of its parameters must be a LiteralType.
- The class must have no virtual base classes.

DRY RTB OFD MLI AED

BENCHMARKING

Pay attention:

- Platform
- Optimization level
- Compiler instructions: Inlining, constexpr, etc.
- Compiler version

THE CODE

```
class BaseFilterVirtual
{
public:
	virtual inline void Activate(int *pixel) const
	{
		cout << "BaseFilterVirtual Activate" <<endl;
	}
};
class FilterVirtual : public BaseFilterVirtual
{
public:
	virtual inline void Activate(int *pixel) const override
	{
		*pixel -=1;
	}
};


FilterVirtual fv;
BaseFilterVirtual *p_fv = &fv;

p_fv->Activate(&pixel);
```

```
template <class FilterCRTP> class BaseFilterCRTP
{
public:
	constexpr auto Activate(int *pixel) const noexcept
	{
		static_cast <const FilterCRTP *>(this)->
		ImplementActivate(pixel);
	}
};
class FilterCRTP : public BaseFilterCRTP<FilterCRTP>
{
public:
	constexpr auto ImplementActivate(int *pixel)
	const noexcept	{
		*pixel -=1;
	}
};

FilterCRTP fc;
FilterCRTP * p_fc = &fc;

p_fc ->Activate(&pixel);
```

THE CODE

```
struct FilterBright { };
struct FilterDark { };

struct VisitActivate
{
        constexpr void operator()(FilterBright& filter, Pixel& pixel)
        { pixel+=1; }
        constexpr void operator()(FilterDark&, Pixel& pixel)
        { pixel-=1; }
};
```

```
int main()
{
        std::variant<int> photoS[PHOTO_SIZE][PHOTO_SIZE];
        std::variant<FilterBright, FilterDark> MyFilter {
        FilterBright() };

        std::visit(VisitActivate(), MyFilter, photoS[Si][Sj]);
}
```

```
struct FilterBright {};
struct FilterDark {};

constexpr std::variant<Pixel> filter_pixel(Pixel &val)
{
        std::variant<FilterBright, FilterDark> Filter{FilterDark()};
        std::variant<Pixel> pixel = 1;

        std::visit (VisitActivate(), Filter, pixel);

        return pixel;
}
```

```
int main()
{
        std::variant<int> photoPixelAfter[PHOTO_SIZE] = {0};

        constexpr auto pixel = filter_pixel(i);
}
```

OPTIMIZATIONS

-O0 / -O1

Reduce code size and execution time, without optimizations that take a great deal of compilation time.

-O2

Reduce code size and execution time,increasing compile time. Adds loop unrolling.

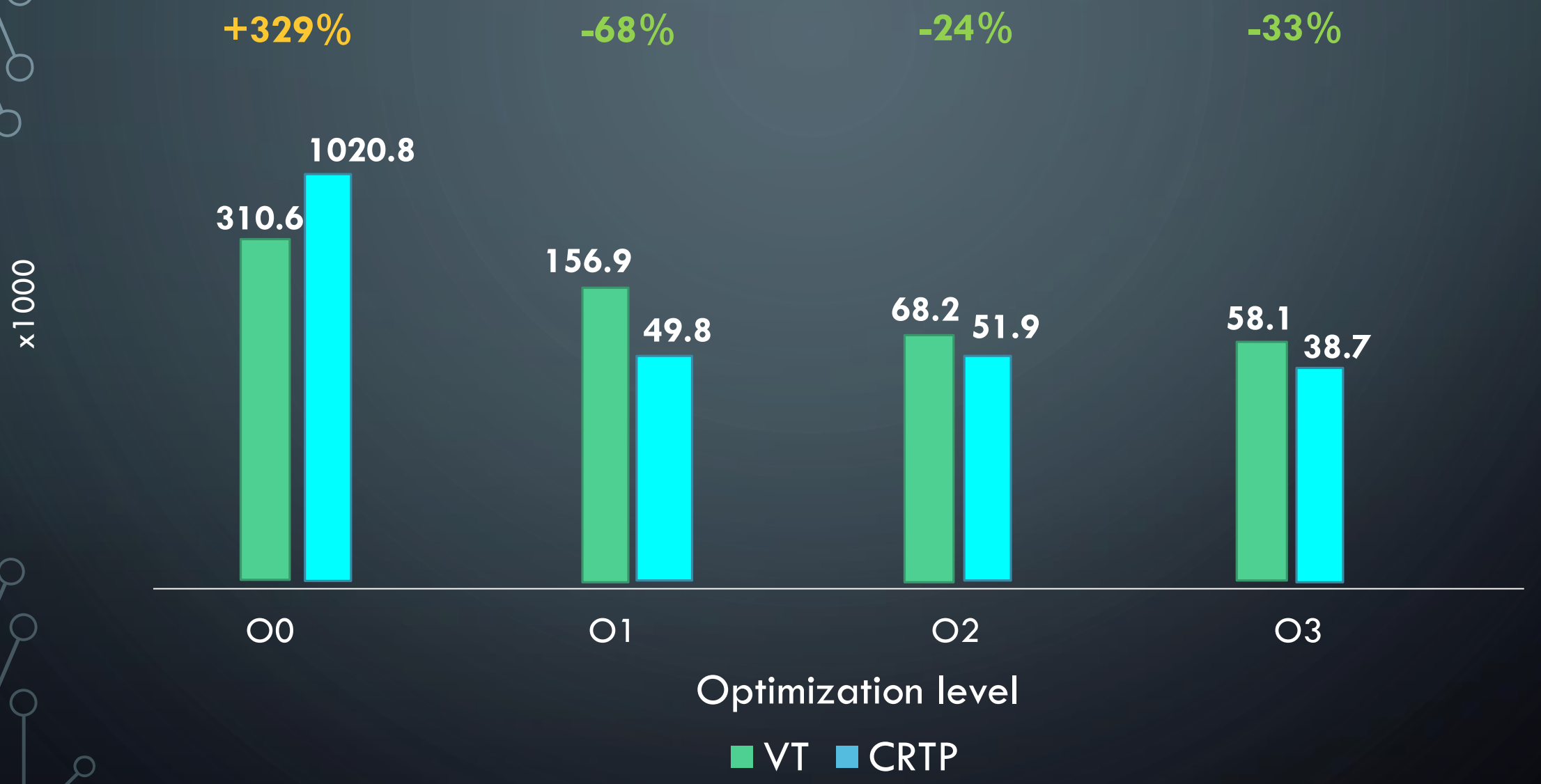
-O3

All optimizations are on, including inline.

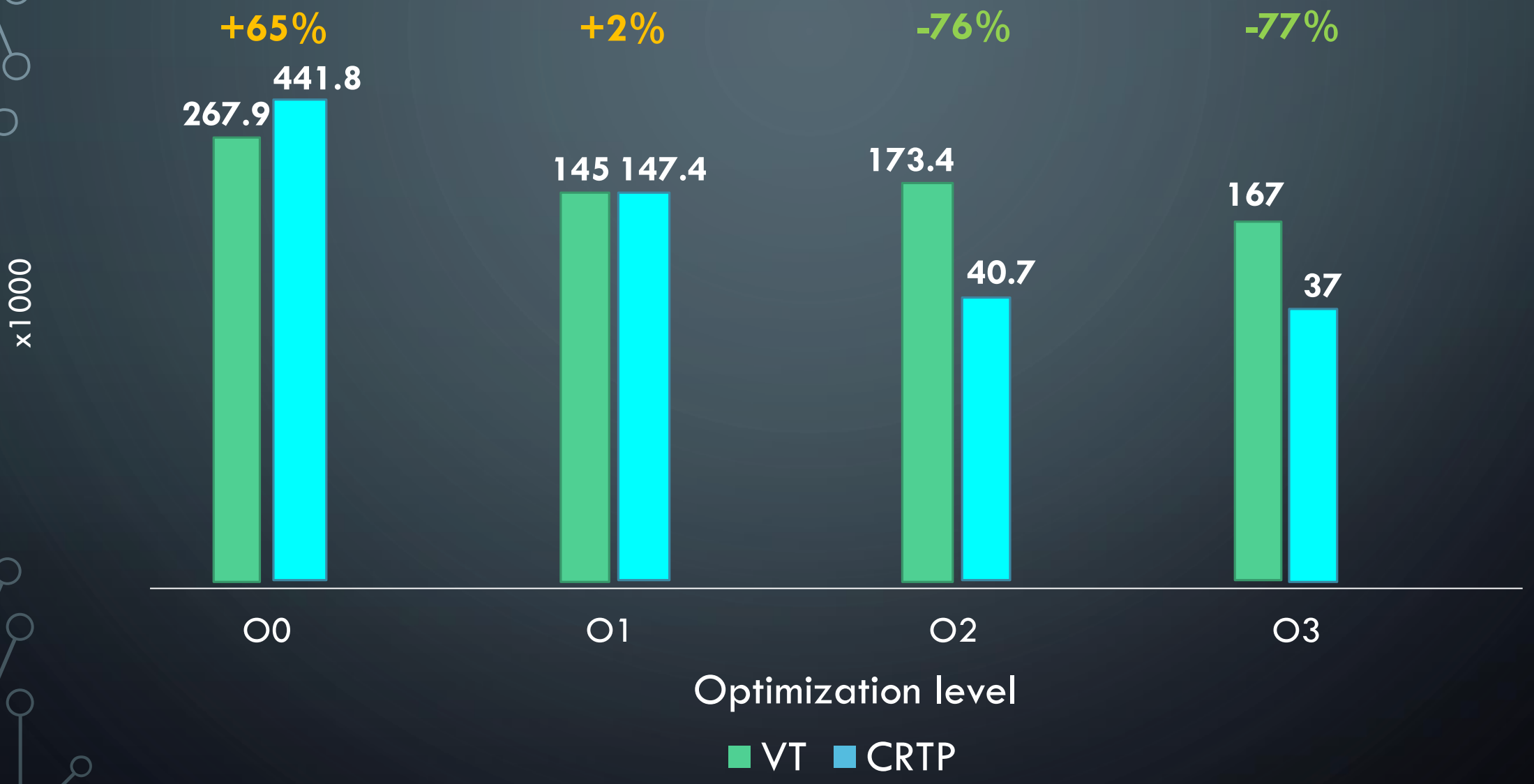
-Os

Reduce size.

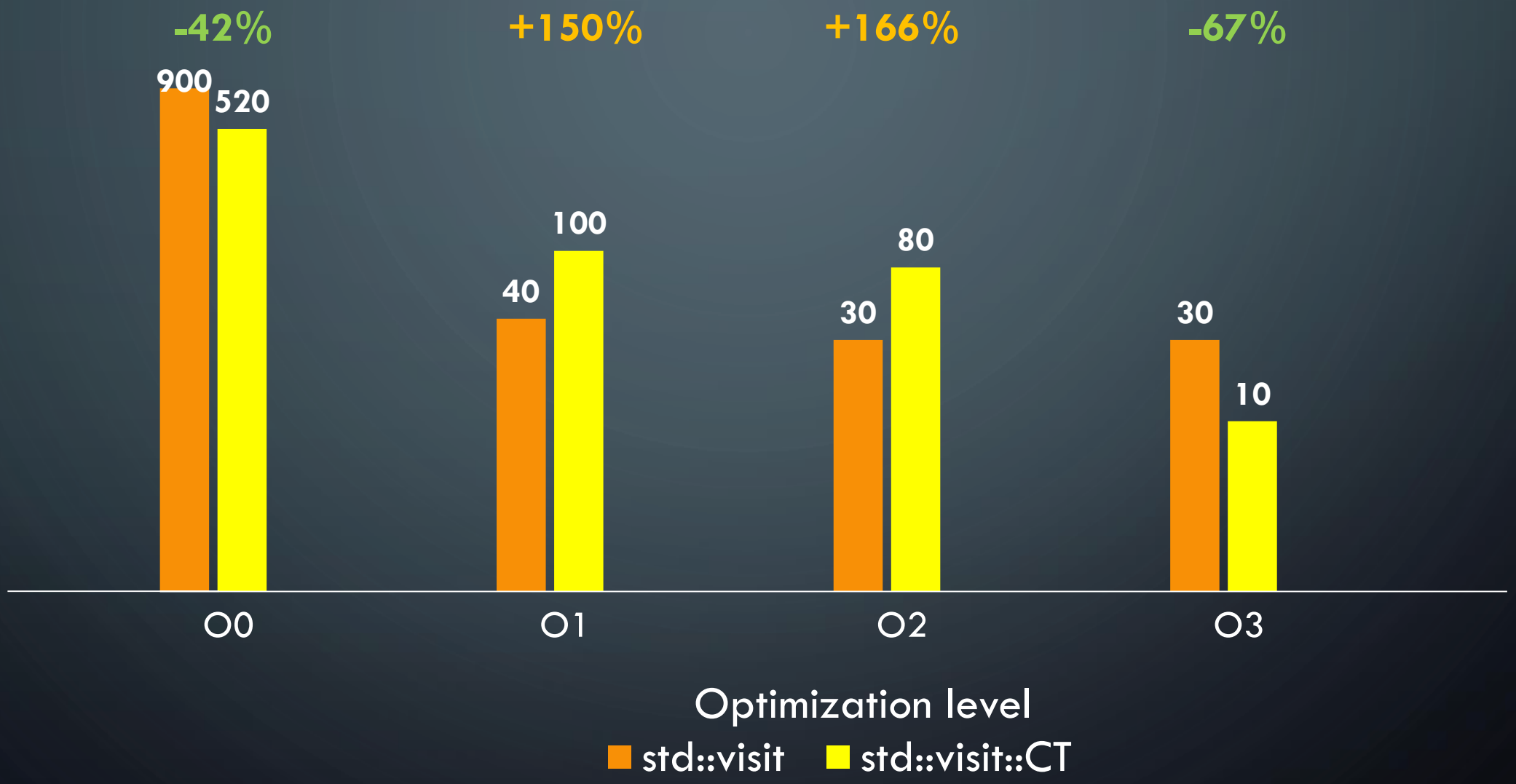
RESULTS - GCC


+329%
-68%
-24%
-33%
x1000
310.6
1020.8
156.9
49.8
68.2
51.9
58.1
38.7
O0
O1
O2
O3
Optimization level
VT
CRTP

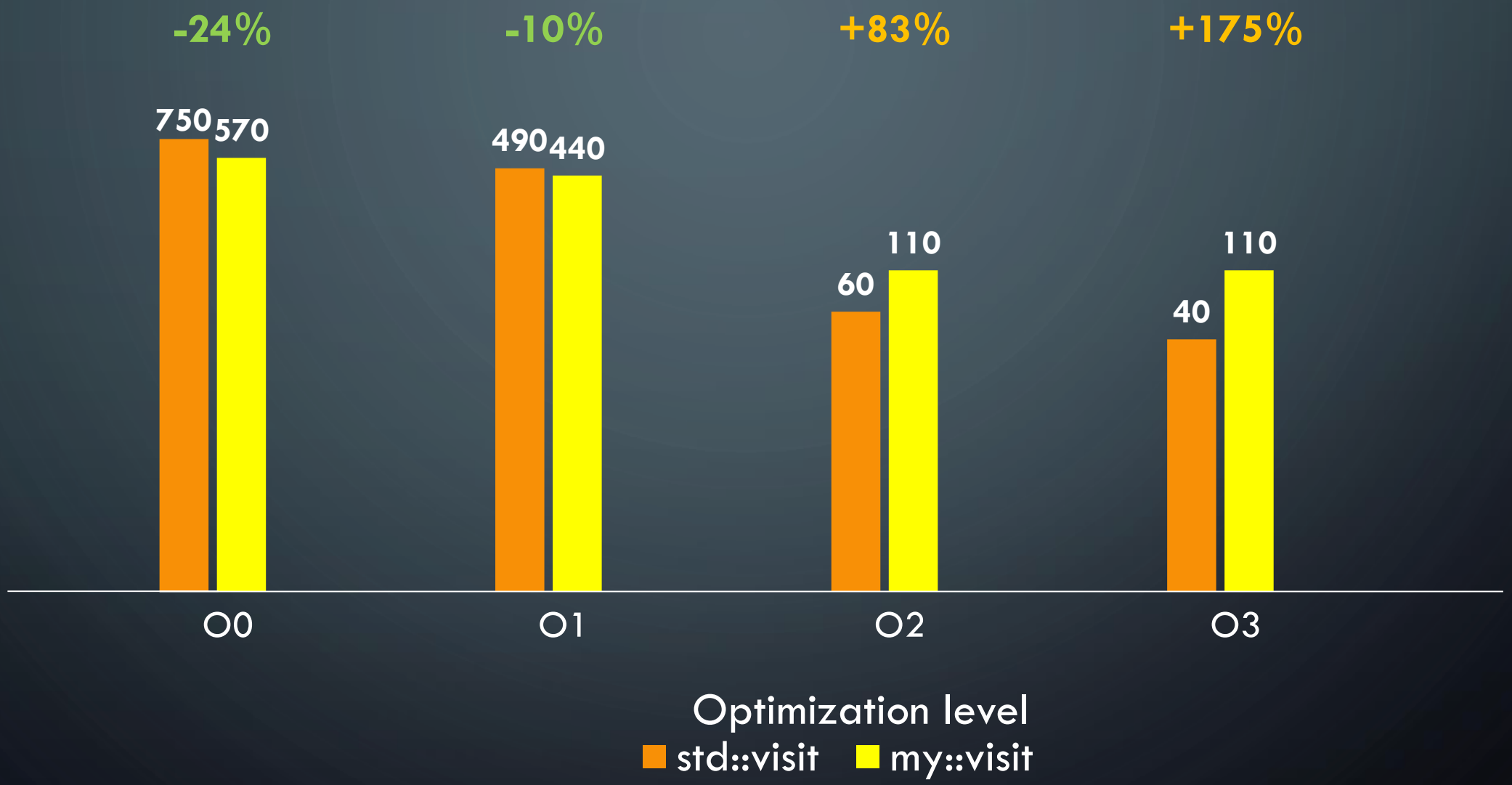
RESULTS - CLANG



RESULTS - GCC


-42%
+150%
+166%
-67%
900
520
40
100
30
80
30
10
O0
O1
O2
O3
Optimization level
std::visit
std::visit::CT

RESULTS - CLANG



CONCLUSION

- **Overhead:**
  - Memory vs. Run time
  - Design
- **Structure:**
  - Static dispatch
  - Dynamic dispatch

	VT	CRTP	Visitor	CTVisitor
DRY	✓	✓	✓	✓
RTB	✓	✗	✓	✗
OFD	✓	✓	✓	✓
MLI	✓	✓	✗	✗
AED	✓	✓	✗	✗
	Slow			

THE FUTURE…

- Expand compile time programming

  C++20's major features include:

  - Broad use of "normal" C++ for direct compile-time programming, without using template metaprogramming (the guys in the next room…)

- constexpr, consteval

- virtual constexpr? /something else?

  - 2019: P1717 Compile-time Metaprogramming in C++
  - 2018: P1064 Allowing Virtual Function Calls in Constant Expressions

- WIP ISO C++20

VIRTUAL DISPATCH AND ITS ALTERNATIVES

Thank you!

- Bryce Ielbach
- Louis Dionne
- Jason turner
- Compiler Explorer (godbolt.org) (the guy in the next room…)

The internet!

Stay In Touch!
github.com/inbal2l
linkdin.com/in/inballevi
sinbal2l@gmail.com