Press Esc to exit full screen

Virtual Tables OR The Overhead Of Magic

Inbal Levi

The Goal

Overview

The Alternatives

When in doubt, Benchmark!

Conclusion

Press Esc to exit full screen

Polymorphism

We want to be able to implement a "**Derived is a Base**" relation.

In the lecture we will try to understand what happens behind the scenes, and use it.

Warm Up

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

int main() {
    cout << "Start:" << endl;
    Base *b = new Derived;
    delete b;
}
```

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

```
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() {
    cout << "Start:" << endl;
    Base *b = new Derived;
    delete b;
}
```

```
Base Ctor
Derived Ctor
Base Dtor
```

```
class Base {
public:
    Base() {
        cout << "Base Ctor" << endl;
    }
    virtual ~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() {
    cout << "Start:" << endl;
    Base *b = new Derived;
    delete b;
}
```

```
class Base {
public:
    Base() {
        cout << "Base Ctor" << endl;
    }
    virtual ~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() {
    cout << "Start:" << endl;
    Base *b = new Derived;
    delete b;
}
```

```
Base Ctor
Derived Ctor
Derived Dtor
Base Dtor
```

```
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;
    }
    ~Derived() {
        cout << "Derived Dtor" << endl;
    }
    void printMe() {
        cout << "Hi, Derived" << endl;
    }
};
```

```
int main() {
    cout << "Start:" << endl;
Derived ~~Base~~ *b = new Derived;
    delete b;
}
```

```
Base Ctor
Derived Ctor
Derived Dtor
Base Dtor
```

Press Esc to exit full screen

Virtual Tables OR The Overhead Of Magic

Inbal Levi

The Goal

Overview

The Alternatives

When in doubt, Benchmark!

Conclusion

Dynamic Binding

vftable is used to support **dynamic dispatch** (run-time method binding).

Structure

Run time

Pitfalls

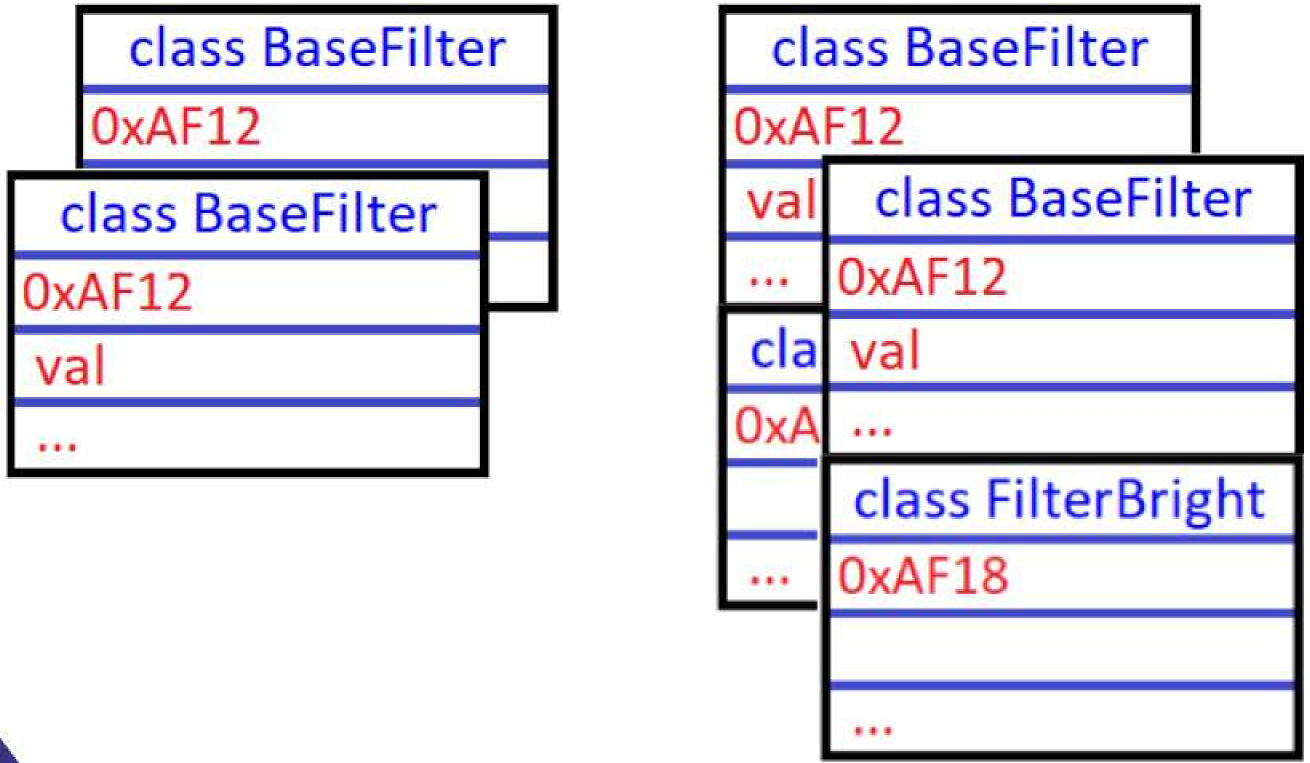
Structure

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

class FilterBright : public BaseFilter {
public:
    virtual inline void Activate(PIXEL *pixel {
        *pixel += 1;
    }
};
```

Structure

0xAF12		
class BaseFilter	ptr_Activate	...

0xAF18		
class FilterBright	ptr_Activate	...



Structure

```
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"
```

Activate Ptr

RTTI & dynamic_cast Type information

Structure

```
vtable for FilterBright:
      .quad   0

      .quad   FilterBright::~FilterBright() [Complete Dtor]
      .quad   FilterBright::~FilterBright() [Deleting Dtor]
      .quad   FilterBright::Activate(unsigned char*)
vtable for BaseFilter:
      .quad   0

      .quad   BaseFilter::~BaseFilter() [Complete Dtor]
      .quad   BaseFilter::~BaseFilter() [Deleting Dtor]
      .quad   BaseFilter::Activate(unsigned char*)
```

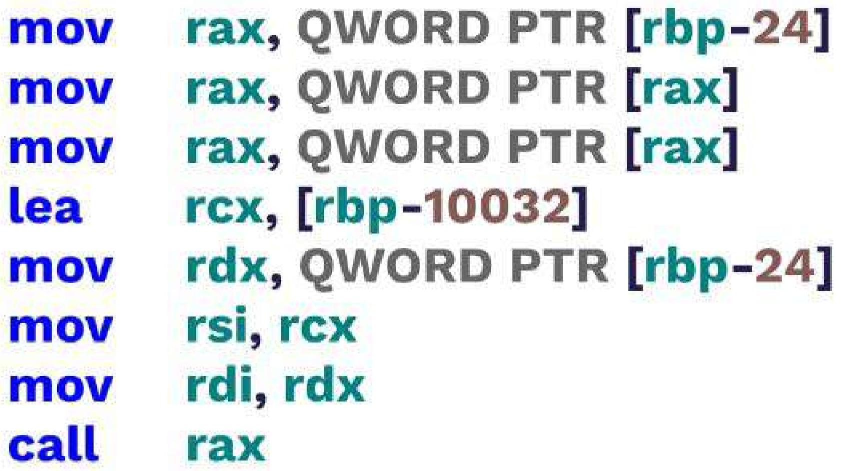
Activate Ptr

RTTI & dynamic_cast Type information

-fno-rtti

Run Time

- The call for "Activate" function



VT

```
mov     rax, QWORD PTR [rbp-24]
mov     rax, QWORD PTR [rax]
mov     rax, QWORD PTR [rax]
lea     rcx, [rbp-10032]
mov     rdx, QWORD PTR [rbp-24]
mov     rsi, rcx
mov     rdi, rdx
call    rax
```

rax	accumulator register
rbp	stack base pointer
rcx	counter register

Run Time

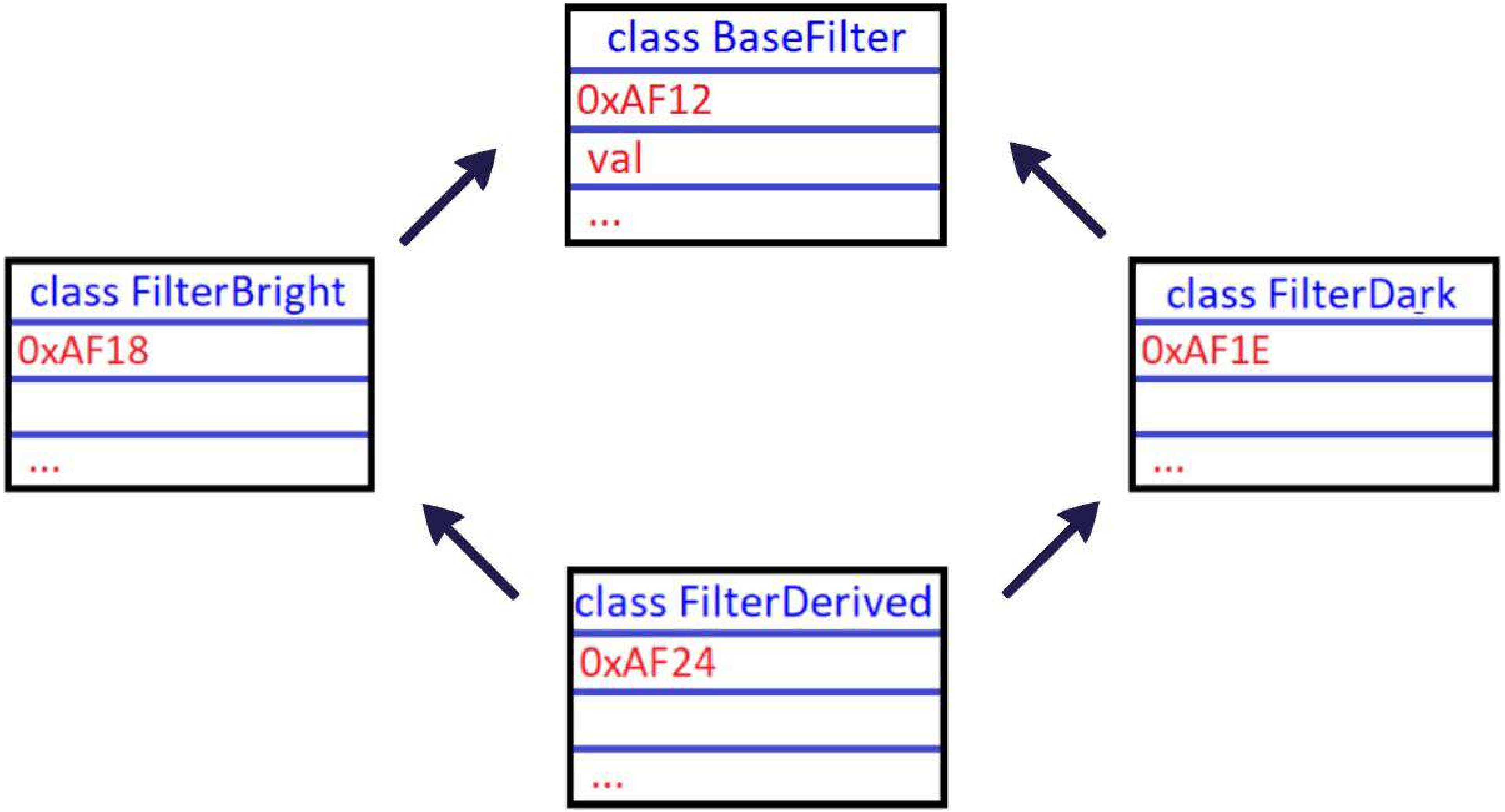
- The call for "Activate" function

```
lea     rcx, [rbp-10032]
mov     rdx, QWORD PTR [rbp-24]
mov     rsi, rcx
mov     rdi, rdx
call    ~~rax~~ FilterBright::Activate(unsigned char*)
```

rax	accumulator register
rbp	stack base pointer
rcx	counter register

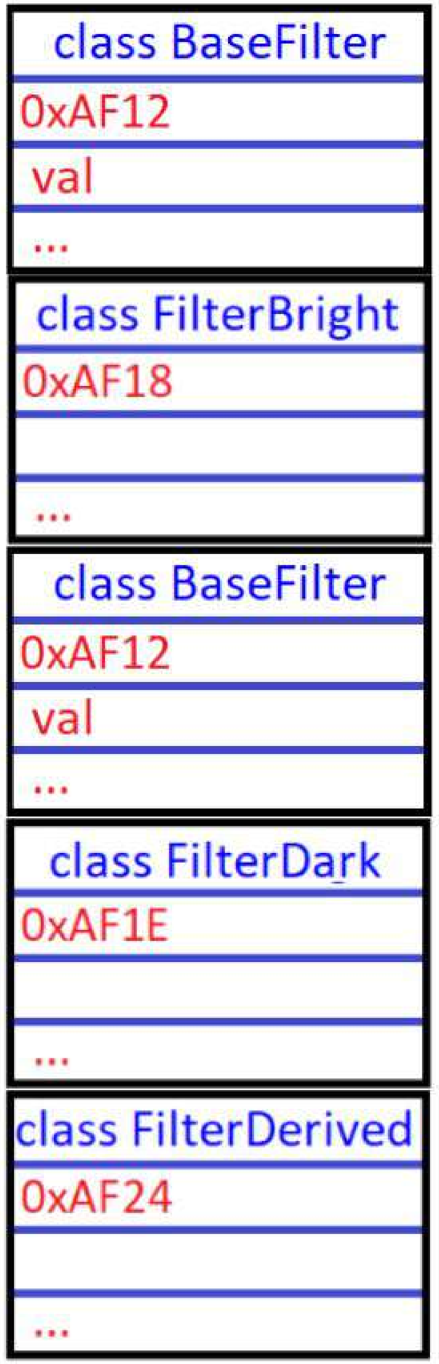
Pitfalls

- Multiple Inheritance / Diamond problem



Diamond Problem

```
class BaseFilter {
public:
     BaseFilter (char val_ = 0) : val (val_) {}
    void Activate(PIXEL *p)  {    *p += val;      }
     char val;
};

class FilterBright : public BaseFilter {
public:
    FilterBright (char val_ ) : BaseFilter(val_) {}
};

class FilterDark : public BaseFilter {
public:
     FilterDark (char val_) : BaseFilter(val_) {}
};

class FilterDerived : public FilterBright , public FilterDark {
public:
     FilterDerived (char Aval_ = 1 , char Bval_  = -1) :
                 FilterBright (Aval_) ,
                 FilterDark (Bval_) {}
};
int main()
{
    FilterDerived * myFilter = new FilterDerived;
    cout << "Val = " << myFilter->val << endl;
}
```



class BaseFilter

0xAF12

val

...

class FilterBright

0xAF18

...

class BaseFilter

0xAF12

val

...

class FilterDark

0xAF1E

...

class FilterDerived

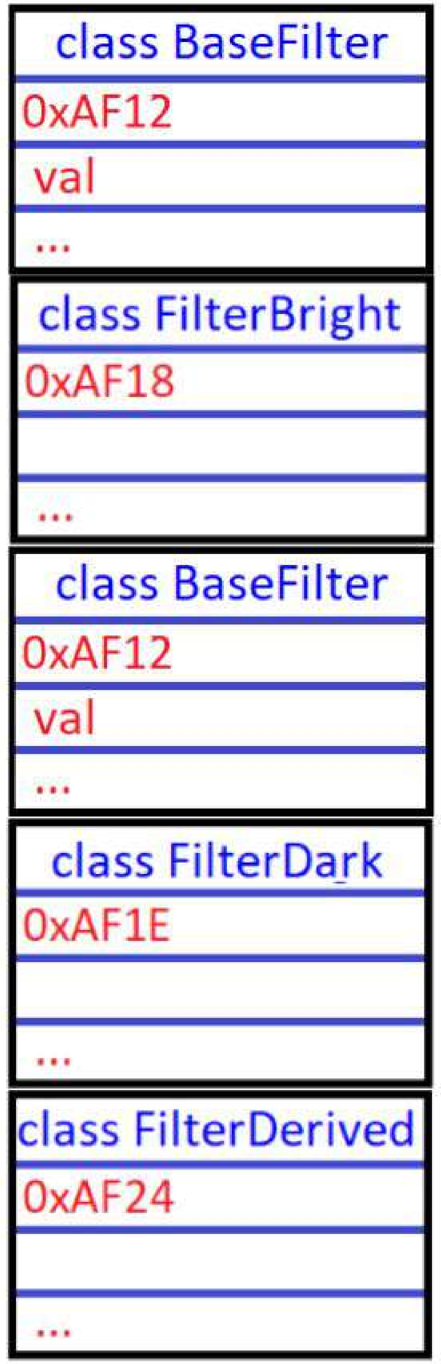
0xAF24

...

error:
reference to 'val' is ambiguous virtual inline void Activate(PIXEL *pixel)
{   *pixel-=val;   }

Diamond Problem

```
class BaseFilter {
public:
     BaseFilter (char val_ = 0) : val (val_) {}
    void Activate(PIXEL *p)  {    *p += val;      }
     char val;
};

class FilterBright : public BaseFilter {
public:
    FilterBright (char val_ ) : BaseFilter(val_) {}
};

class FilterDark : public BaseFilter {
public:
     FilterDark (char val_) : BaseFilter(val_) {}
};

class FilterDerived : public FilterBright , public FilterDark {
public:
     FilterDerived (char Aval_ = 1 , char Bval_  = -1) :
                 FilterBright (Aval_) ,
                 FilterDark (Bval_) {}
};
int main()
{
    FilterDerived * myFilter = new FilterDerived;
    cout << "Val = " << myFilter->FilterBright::val << endl;
}
```



class BaseFilter
0xAF12
val
...

class FilterBright
0xAF18

...

class BaseFilter
0xAF12
val
...

class FilterDark
0xAF1E

...

class FilterDerived
0xAF24

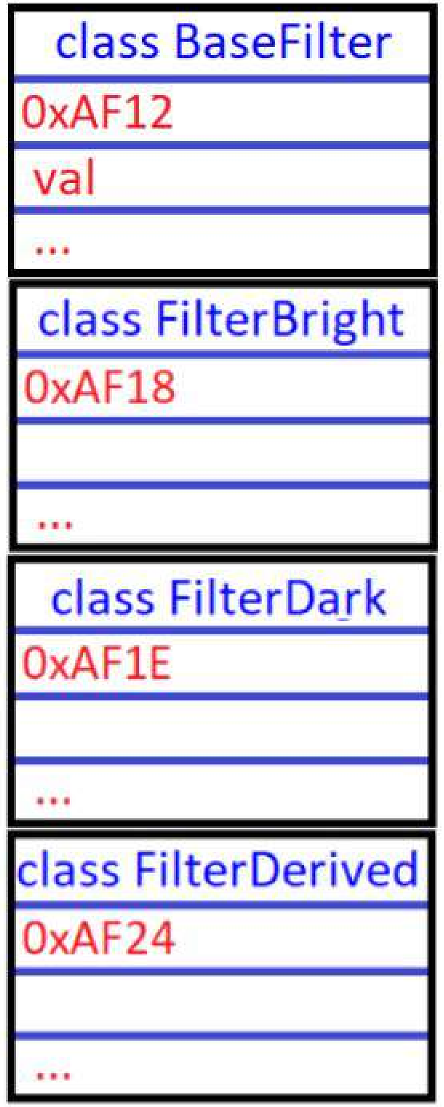
...

error:
reference to 'val' is ambiguous virtual inline void Activate(PIXEL *pixel) {   *pixel-=val;   }

Val = 1

Diamond Problem

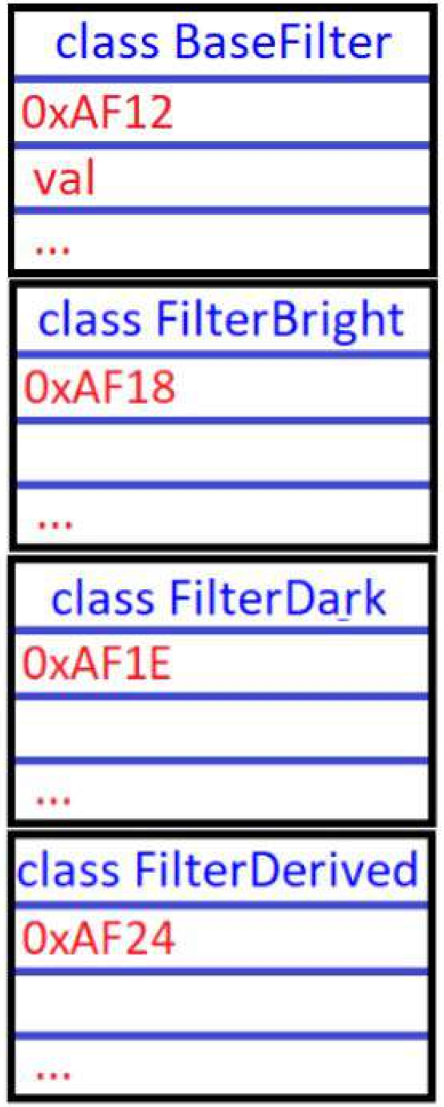
```
class BaseFilter
{
public:
    BaseFilter(int val_ = 0) : val (val_) {
        cout << "Base val: " << val  << endl;
    }
    void Activate(PIXEL *p)  {    *p += val;      }
    int val;
};

class FilterBright : virtual public BaseFilter {
public:
    FilterBright(int val_ = 1) : BaseFilter(val_) {
        cout << "Bright val: " << val  << endl;
    }
};

class FilterDark : virtual public BaseFilter {
public:
    FilterDark(int val_ = -1) : BaseFilter(val_) {
        cout << "Dark val: "<< val  << endl;
    }
};

class FilterDerived : public FilterBright , public FilterDark {
public:
     FilterDerived() : FilterBright(4) , FilterDark(5) {}
};
```



```
int main()
{
    FilterDerived * myFilter = new FilterDerived;
    cout << "Val = " << myFilter->val << endl;
}
```

Diamond Problem

```
class BaseFilter
{
public:
    BaseFilter(int val_ = 0) : val (val_) {
        cout << "Base val: " << val  << endl;
    }
    void Activate(PIXEL *p)  {   *p += val;      }
    int val;
};

class FilterBright : virtual public BaseFilter {
public:
    FilterBright(int val_ = 1) : BaseFilter(val_) {
        cout << "Bright val: " << val  << endl;
    }
};

class FilterDark : virtual public BaseFilter {
public:
    FilterDark(int val_ = -1) : BaseFilter(val_) {
        cout << "Dark val: "<< val  << endl;
    }
};

class FilterDerived : public FilterBright , public FilterDark {
public:
     FilterDerived() : FilterBright(4) , FilterDark(5) {}
};
```



class BaseFilter

0xAF12

val

...

class FilterBright

0xAF18

...

class FilterDark

0xAF1E

...

class FilterDerived

0xAF24

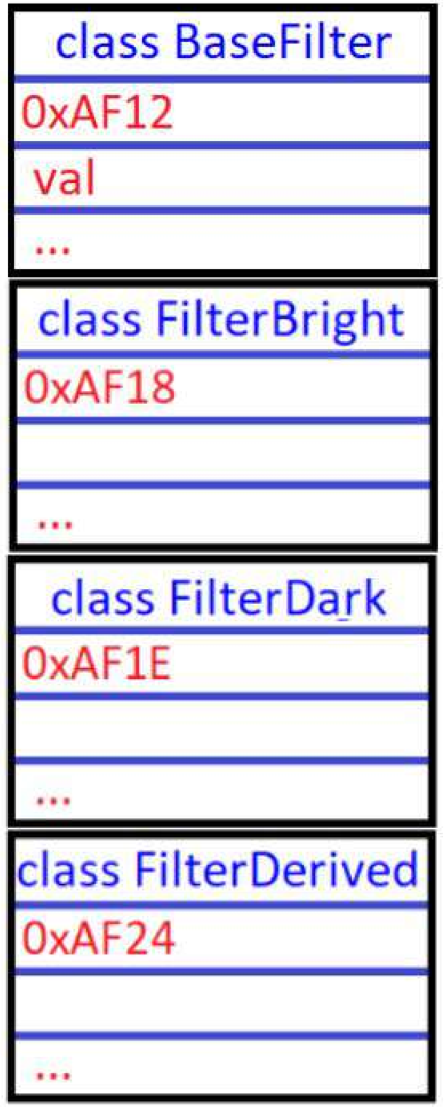
...

Val = 0

```
int main()
{
    FilterDerived * myFilter = new FilterDerived;
    cout << "Val = " << myFilter->val << endl;
}
```

Diamond Problem

```
class BaseFilter
{
public:
    BaseFilter(int val_ = 0) : val (val_) {
        cout << "Base val: " << val  << endl;
    }
    void Activate(PIXEL *p)  {    *p += val;      }
    int val;
};

class FilterBright : virtual public BaseFilter {
public:
    FilterBright(int val_ = 1) : BaseFilter(val_) {
        cout << "Bright val: " << val  << endl;
    }
};

class FilterDark : virtual public BaseFilter {
public:
    FilterDark(int val_ = -1) : BaseFilter(val_) {
        cout << "Dark val: "<< val  << endl;
    }
};

class FilterDerived : public FilterBright , public FilterDark {
public:
     FilterDerived() : FilterBright(4) , FilterDark(5) {}
};
```



class BaseFilter

0xAF12

val

...

class FilterBright

0xAF18

...

class FilterDark

0xAF1E

...

class FilterDerived

0xAF24

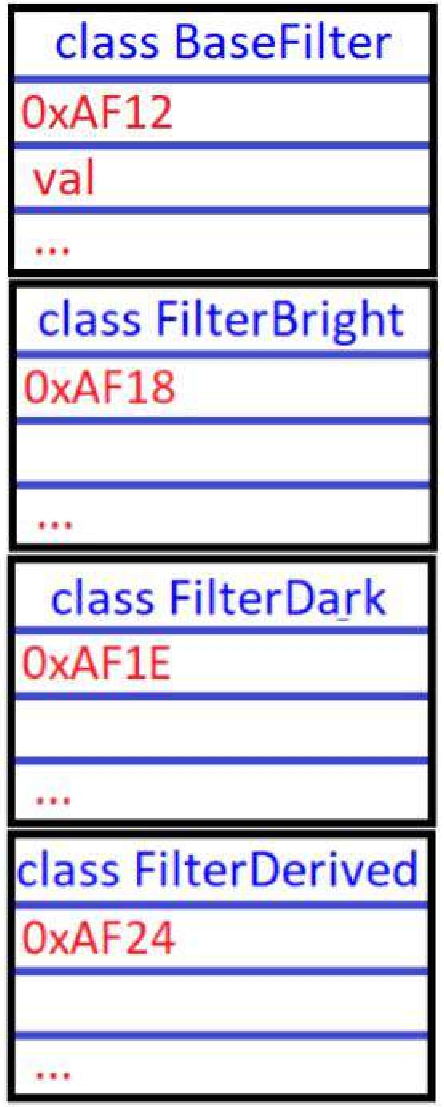
...

```
Base val: 0
Bright val: 0
Dark val: 0
myFilter val: 0
```

```
int main()
{
    FilterDerived * myFilter = new FilterDerived;
    cout << "Val = " << myFilter->val << endl;
}
```

Diamond Problem

```
class BaseFilter
{
public:
    BaseFilter(int val_ = 0) : val (val_) {
        cout << "Base val: " << val  << endl;
    }
    void Activate(PIXEL *p)  {   *p += val;      }
    int val;
};

class FilterBright : virtual public BaseFilter {
public:
    FilterBright(int val_ = 1) : BaseFilter(val_) {
        cout << "Bright val: " << val  << endl;
    }
};

class FilterDark : virtual public BaseFilter {
public:
    FilterDark(int val_ = -1) : BaseFilter(val_) {
        cout << "Dark val: "<< val  << endl;
    }
};

class FilterDerived : public FilterBright , public FilterDark {
public:
     FilterDerived() : BaseFilter (6) , FilterBright(4) , FilterDark(5) {}
};
```



class BaseFilter

0xAF12

val

...

class FilterBright

0xAF18

...

class FilterDark

0xAF1E

...

class FilterDerived

0xAF24

...

```
Base val: 0
Bright val: 0
Dark val: 0
myFilter val: 0
```

```
Base val: 6
Bright val: 6
Dark val: 6
myFilter val: 6
```

```
int main()
{
    FilterDerived * myFilter = new FilterDerived;
    cout << "Val = " << myFilter->val << endl;
}
```

Press Esc to exit full screen

Virtual Tables OR The Overhead Of Magic

Inbal Levi

The Goal

Overview

The Alternatives

When in doubt, Benchmark!

Conclusion

Virtual Table Alternatives

- Parametric Polymorphism
- Subtyping
- CRTP

Parametric Polymorphism

- We refer to objects as memory buffers, and manage types on our own.
- We can use operator new, operator delete, etc,. in order to manage the memory.

```
class Base
{
public:
    Base()
    {
        cout<<"Base Ctor"<<endl;
    }

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

```
class Derived : public Base
{
public:
    Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }

    ~Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }

    void printMe()
    {
        cout<<"Hi, Derived"<<endl;
    }
};
```

```
Derived *d = new Derived();

int Activate(int type, Derived *d)
{
  switch(type)
  {
    case BASE:
          static_cast<Base *>(d)->printMe();

    case DERIVED:
          d->printMe();
  }
}
```

Parametric Polymorphism

- **We refer to objects as memory buffers, and manage types on our own.**
- **We can use operator new, operator delete, etc,. in order to manage the memory.**

```cpp
class Base
{
public:
    Base()
    {
        cout<<"Base Ctor"<<endl;
    }

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

```cpp
class Derived : public Base
{
public:
    Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }

    ~Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }

    void printMe()
    {
        cout<<"Hi, Derived"<<endl;
    }
};
```

```cpp
Derived *d = new Derived();

int Activate(int type, Derived *d)
{
  switch(type)
  {
    case BASE:
          static_cast<Base *>(d)->printMe();

    case DERIVED:
          d->printMe();
  }
}
```

```
Base Ctor
Derived Ctor
Hi, Base
Hi, Derived
```

* constexpr inline everywhere.

Parametric Polymorphism

```
class Base
{
public:
    Base()
    {
        cout<<"Base Ctor"<<endl;
    }
    virtual ~Base()
    {
        cout<<"Base Dtor"<<endl;
    }
    void printInt()
    {
        cout << "Base No int" << endl;
    }
};
```

```
class Derived : public Base
{
public:
    Derived(): derivedInt(5)
    {
         cout<<"Derived Ctor"<<endl;
    }
    ~Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }
    void printInt()
    {
        cout<< "Derived int:" << derivedInt << endl;
    }
    int derivedInt;
};
```

```
Derived *d = new Derived();

int Activate(int type, Derived *d)
{
   switch(type) {

      case DERIVED:
              d->printInt();

      case BASE:
              static_cast<Base *>(d)->printInt();
   }
}
```

Parametric Polymorphism

```
class Base
{
public:
    Base()
    {
        cout<<"Base Ctor"<<endl;
    }
    virtual ~Base()
    {
        cout<<"Base Dtor"<<endl;
    }
    void printInt()
    {
        cout << "Base No int" << endl;
    }
};
```

```
class Derived : public Base
{
public:
    Derived(): derivedInt(5)
    {
        cout<<"Derived Ctor"<<endl;
    }
    ~Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }
    void printInt()
    {
        cout<< "Derived int:" << derivedInt << endl;
    }
    int derivedInt;
};
```

```
Derived *d = new Derived();

int Activate(int type, Derived *d)
{
   switch(type) {

      case DERIVED:
            d->printInt();

      case BASE:
            static_cast<Base *>(d)->printInt();
   }
}
```

```
Base Ctor
Derived Ctor
Derived int: 5
Base No int
```

Parametric Polymorphism

```
class Base
{
public:
    Base()
    {
        cout<<"Base Ctor"<<endl;
    }
    virtual ~Base()
    {
        cout<<"Base Dtor"<<endl;
    }
    void printInt()
    {
        cout << "Base No int" << endl;
    }
};
```

```
class Derived : public Base
{
public:
    Derived(): derivedInt(5)
    {
        cout<<"Derived Ctor"<<endl;
    }
    ~Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }
    void printInt()
    {
        cout<< "Derived int:" << derivedInt << endl;
    }
    int derivedInt;
};
```

```
Derived *d = new Derived();

int Activate(int type, Derived *d)
{
  switch(type) {

    case DERIVED:
          d->printInt();

    case BASE:
          static_cast<Base *>(d)->printInt();
  }
}
```

```
Base Ctor
Derived Ctor
Derived int: 5
Base No int
```

```
Base *d = new Base();

int Activate(int type, Base *d)
{
   switch (type) {

       case BASE:
             d->printInt();

       case DERIVED:
             static_cast<Derived *>(d)->printInt();
   }
}
```

Parametric Polymorphism

```
class Base
{
public:
    Base()
    {
        cout<<"Base Ctor"<<endl;
    }
    virtual ~Base()
    {
        cout<<"Base Dtor"<<endl;
    }
    void printInt()
    {
        cout << "Base No int" << endl;
    }
};
```

```
class Derived : public Base
{
public:
    Derived(): derivedInt(5)
    {
        cout<<"Derived Ctor"<<endl;
    }
    ~Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }
    void printInt()
    {
        cout<< "Derived int:" << derivedInt << endl;
    }
    int derivedInt;
};
```

```
Derived *d = new Derived();

int Activate(int type, Derived *d)
{
  switch(type) {

    case DERIVED:
          d->printInt();

    case BASE:
          static_cast<Base *>(d)->printInt();
  }
}
```

```
Base Ctor
Derived Ctor
Derived int: 5
Base No int
```

```
Base *d = new Base();

int Activate(int type, Base *d)
{
   switch (type) {

      case BASE:
            d->printInt();

      case DERIVED:
            static_cast<Derived *>(d)->printInt();
   }
}
```

```
Base Ctor
Base No int
Derived int: 0
```

Parametric Polymorphism

```
class Base
{
public:
    Base()
    {
        cout<<"Base Ctor"<<endl;
    }
    virtual ~Base()
    {
        cout<<"Base Dtor"<<endl;
    }
    void printInt()
    {
        cout << "Base No int" << endl;
    }
};
```

```
class Derived : public Base
{
public:
    Derived(): derivedInt(5)
    {
        cout<<"Derived Ctor"<<endl;
    }
    ~Derived()
    {
        cout<<"Derived Ctor"<<endl;
    }
    void printInt()
    {
        cout<< "Derived int:" << derivedInt << endl;
    }
    int derivedInt;
};
```

```
Derived *d = new Derived();

int Activate(int type, Derived *d)
{
  switch(type) {

    case DERIVED:
          d->printInt();

    case BASE:
          static_cast<Base *>(d)->printInt();
  }
}
```

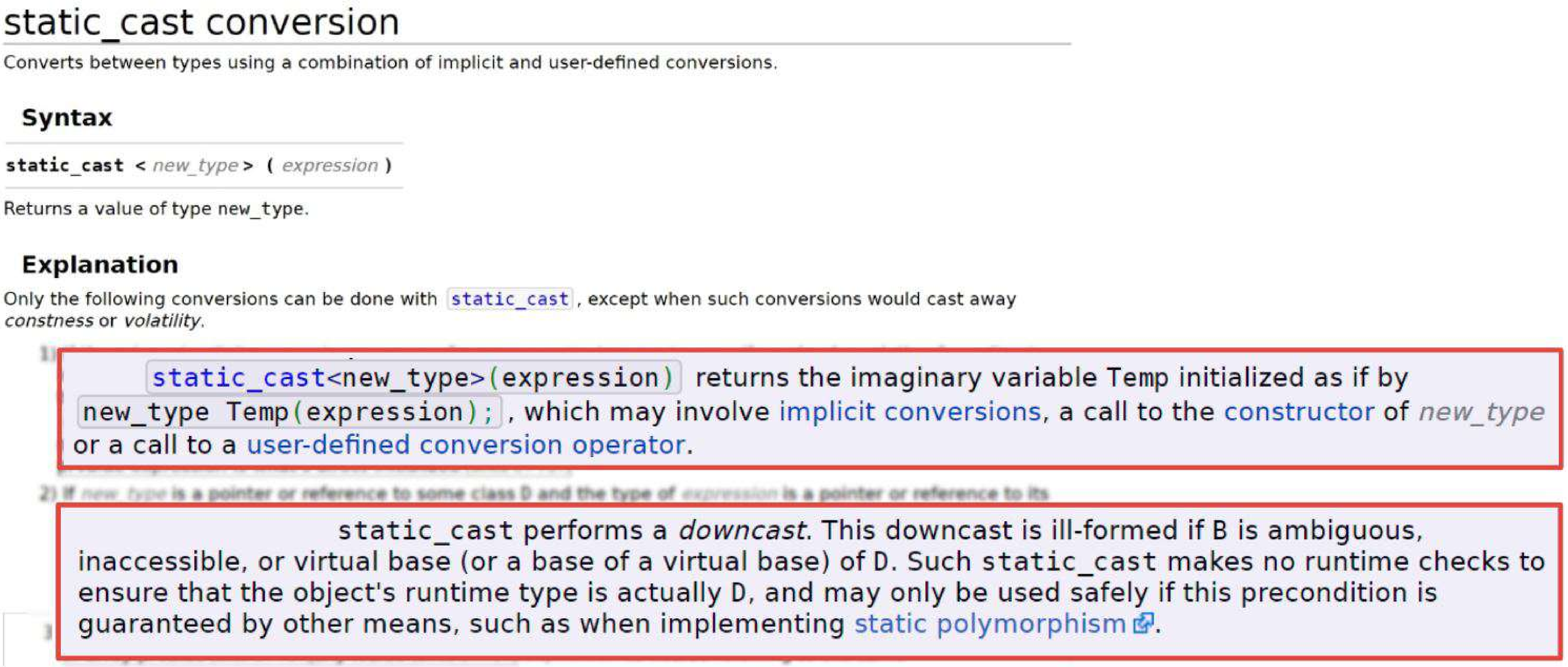
Base Ctor
Derived Ctor
Derived int: 5
Base No int

```
Base d;

int Activate(int type, Base& d)
{
  switch (type) {

    case BASE:
          d.printInt();

    case DERIVED:
          static_cast<Derived *>(&d)->printInt();
  }
}
```

Base Ctor
Base No int
Derived int: -2

Let's not even consider reinterpret_cast...

Parametric Polymorphism



static_cast conversion

Converts between types using a combination of implicit and user-defined conversions.

Syntax

`static_cast < new_type > ( expression )`

Returns a value of type new_type.

Explanation

Only the following conversions can be done with `static_cast`, except when such conversions would cast away *constness* or *volatility*.

`static_cast<new_type>(expression)` returns the imaginary variable Temp initialized as if by `new_type Temp(expression);`, which may involve implicit conversions, a call to the constructor of *new_type* or a call to a user-defined conversion operator.

2) If *new_type* is a pointer or reference to some class D and the type of *expression* is a pointer or reference to its

static_cast performs a *downcast*. This downcast is ill-formed if B is ambiguous, inaccessible, or virtual base (or a base of a virtual base) of D. Such static_cast makes no runtime checks to ensure that the object's runtime type is actually D, and may only be used safely if this precondition is guaranteed by other means, such as when implementing static polymorphism.

static_cast may call **CTOR!**

static_cast does not validate object type!

Subtyping

```
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();
}
```

* constexpr inline everywhere.

Subtyping

```
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
```

* constexpr inline everywhere.

CRTP

Library

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

```
int main()
{
    BaseFilter<FilterBright> *f1 = new FilterBright();
    f1->Activate();
    BaseFilter<FilterDark> *f2 = new FilterDark();
    f2->Activate();
}
```

User implementation

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

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

CRTP

Library

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

```
int main()
{
     BaseFilter<FilterBright> *f1 = new FilterBright();
    f1->Activate();
    BaseFilter<FilterDark> *f2 = new FilterDark();
     f2->Activate();
}
```

User implementation

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

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

```
Activate Bright
Activate Dark
```

CRTP

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

```
template <typename T> class BaseCRTP {
public:
    BaseCRTP (int a = 1, int b = 1): A(a) , B(b){}
    void Print() {
        cout << "getA: " << static_cast<T *>(this)->getA() << endl;
        cout << "getB: " << static_cast<T *>(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;      }
};
```

CRTP

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

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

```
getA: 1
getB: 1
```

```
template <typename T> class BaseCRTP {
public:
   BaseCRTP (int a = 1, int b = 1): A(a) , B(b){}
   void Print() {
      cout << "getA: " << static_cast<T *>(this)->getA() << endl;
      cout << "getB: " << static_cast<T *>(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;      }
};
```

CRTP

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

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

getA: 1
getB: 1

```
template <typename T> class BaseCRTP {
public:
   BaseCRTP (int a = 1, int b = 1): A(a) , B(b){}
   void Print() {
      cout << "getA: " << static_cast<T *>(this)->getA() << endl;
      cout << "getB: " << static_cast<T *>(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

CRTP

Subtyping	CRTP
Easy to **read**, understand and **implement**	Not intuitive for **reading**, complicates code
Inheritance of more than one descendant is easy and clear	**Inheritance of more than one descendant** is hard
Derived class implements Inherited functions, calls **Base member functions** in the derived.	**Derived class** implements inherited functions, **can call Derived** member functions.

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

CRTP

Pitfalls:

- Less intuitive.
- Multiple inheritance demands special implementation.

CRTP

```
template <typename T>
class BaseFilter
{
public:
    inline constexpr void Activate() {
        static_cast<const T*>(this)->Activate();
    }

protected:
    ~BaseFilter() = default;
};

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

```
template <typename T = void>
class MiddleFilter : public BaseFilter<MiddleFilter<T>>
{
public:
    inline void Activate()  {
        Activate_impl(std::is_same<T, void>{});
    }

private:
    inlinevoid Activate_impl (std::true_type) {
        cout << "Middle Activate" << endl;
    }
    void Activate_impl (std::false_type) {
        if (&MiddleFilter::Activate == &T::Activate)
            Activate_impl (std::true_type{});
        else
            static_cast<const T*>(this)->Activate();
    }
};
```

CRTP

```
template <typename T>
class BaseFilter
{
public:
    inline constexpr void Activate() {
        static_cast<const T*>(this)->Activate();
    }

protected:
    ~BaseFilter() = default;
};
```

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

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

Derived Activate

```
template <typename T = void>
class MiddleFilter : public BaseFilter<MiddleFilter<T>>
{
public:
    inline void Activate()  {
        Activate_impl(std::is_same<T, void>{});
    }

private:
    inlinevoid Activate_impl (std::true_type) {
        cout << "Middle Activate" << endl;
    }
    void Activate_impl (std::false_type) {
        if (&MiddleFilter::Activate == &T::Activate)
            Activate_impl (std::true_type{});
        else
            static_cast<const T*>(this)->Activate();
    }
};
```

CRTP

```
template <typename T>
class BaseFilter
{
public:
    inline constexpr void Activate() {
        static_cast<const T*>(this)->Activate();
    }

protected:
    ~BaseFilter() = default;
};

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

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

Derived Activate

```
template <typename T = void>
class MiddleFilter : public BaseFilter<MiddleFilter<T>>
{
public:
    inline void Activate()  {
        Activate_impl(std::is_same<T, void>{});
    }

private:
    inlinevoid Activate_impl (std::true_type) {
        cout << "Middle Activate" << endl;
    }
    void Activate_impl (std::false_type) {
        if (&MiddleFilter::Activate == &T::Activate)
            Activate_impl (std::true_type{});
        else
            static_cast<const T*>(this)->Activate();
    }
};

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

Middle Activate

Press Esc to exit full screen

Virtual Tables OR The Overhead Of Magic

Inbal Levi

The Goal

Overview

The Alternatives

When in doubt, Benchmark!

Conclusion

Benchmarking

Pay attention:

The Code

Optimizations

Results

Benchmarking

Pay attention:

- Platform

The Code

Optimizations

Results

Benchmarking

Pay attention:

- Platform
- Optimization level

The Code

Optimizations

Results

Benchmarking

Pay attention:

- Platform
- Optimization level
- Compiler explicit instructions

The Code

Optimizations

Results

Benchmarking

The Code

Pay attention:

- Platform
- Optimization level
- Compiler explicit instructions
- Compiler

Optimizations

Results

Benchmarking

```
class BaseFilterVirtual
{
public:
    virtual inline void Activate(int *pixel)
    {
        cout << "BaseFilterVirtual Activate" <<endl;
    }
};

class FilterVirtual : public BaseFilterVirtual
{
public:
    virtual inline void Activate(int *pixel)
    {
        *pixel-=1;
    }
};


BaseFilterVirtual *f1 = new FilterVirtual();
f1->Activate();
```

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

class FilterCRTP : public BaseFilterCRTP<FilterCRTP>
{
public:
    inline void ImplementActivate(int *pixel)
    {
        *pixel-=1;
    }
};


FilterCRTP f1;
f1.Activate();
```

+Locality

Optimizations

-O / -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.

CPU Ticks

Vtable:
CRTP:

(x 10^3, Thousands)

10^8 = 10000 x 10000 pixels

CPU Ticks

O0

Vtable:	**276.7**
CRTP:	**967.2**

+250%

(x 10^3, Thousands)

10^8 = 10000 x 10000 pixels

CPU Ticks

	O0	O1
Vtable:	**276.7**	**141.0**
CRTP:	**967.2**	**34.5**
	+250%	**-76%**

(x 10^3, Thousands)

10^8 = 10000 x 10000 pixels

CPU Ticks

	O0	O1	O2
Vtable:	**276.7**	**141.0**	**54.9**
CRTP:	**967.2**	**34.5**	**34.6**
	+250%	**-76%**	**-37%**

(x 10^3, Thousands)

10^8 = 10000 x 10000 pixels

CPU Ticks

	O0	O1	O2	O3
Vtable:	276.7	141.0	54.9	64.8
CRTP:	967.2	34.5	34.6	10.4
	+250%	-76%	-37%	-84%

(x 10^3, Thousands)

10^8 = 10000 x 10000 pixels

Press Esc to exit full screen

Virtual Tables OR The Overhead Of Magic

Inbal Levi

The Goal

Overview

The Alternatives

When in doubt, Benchmark!

Conclusion

- Structure:
    - Static dispatch
    - Dynamic dispatch

- Overhead:
    - Design
    - Run time

- Structure:
    - Static dispatch
    - Dynamic dispatch

- Overhead:
    - Design
    - Run time

	ManageMem	VT	CRTP
Readability	🟥 locally readable	🟩 readable	🟧 readable (with practice)
Run time flexibility	🟧 flexible	🟩 flexible	🟥 Not flexible
Run time performance	🟧 medium (switch/if)	🟥 slow (ptr)	🟩 fast (+locality)

Do you care about performance?

Do you care about how?

Yes!

Thanks!

Compiler Explorer: https://godbolt.org/
Benchmarking for fun: http://quick-bench.com/
Fluent C++: https://www.fluentcpp.com/2018/05/22/how-to-transform-a-hierarchy-of-virtual-methods-into-a-crtp/
Eli Bendersky's website: https://eli.thegreenplace.net/2013/12/05/the-cost-of-dynamic-virtual-calls-vs-static-crtp-dispatch-in-c

The internet!

Stay In Touch!

github.com/inbal2l
linkdin.com/in/inballevi
sinbal2l@gmail.com