

# **OPEN IS GOOD**

**YOMM2: Fast, Orthogonal Open (Multi) Methods**

Jean-Louis Leroy - [jl@leroy.nyc](mailto:jl@leroy.nyc)

**IF YOMM2 IS THE  
SOLUTION,  
WHAT IS THE  
PROBLEM?**

# THE EXPRESSION PROBLEM

*in a polymorphic system...*

- existing operations += new types?
- existing types += new operations?

*should be possible, easy*

# C++ COMPILE-TIME POLYMORPHISM

(aka templates)

- existing operations += new types? **easy**
- existing types += new operations? **easy**

to wit: the STL

# C++ RUN-TIME POLYMORPHISM

- existing operations += new types?  
**easy: virtual functions, derivation**
- existing types += new operations?  
**emmm...**

# CASE STUDY

# ABSTRACT SYNTAX TREE

```
struct Node {
    virtual ~Node() {}
    virtual int value() const = 0;
};

struct Number : Node {
    explicit Number(int value) : val(value) { }
    int value() const override { return val; }
    int val;
};

struct Plus : Node {
    Plus(const Node& left, const Node& right) : left(left), right(right) { }
    int value() const override { return left.value() + right.value(); }
    const Node& left; const Node& right;
};

struct Times : Node {
    Times(const Node& left, const Node& right) : left(left), right(right) { }
    int value() const override { return left.value() * right.value(); }
    const Node& left; const Node& right;
};
```

# AST

```
int main() {
    Number n2(2), n3(3), n4(4);
    Plus sum(n3, n4);
    Times product(n2, sum);

    const Node& expr = product;
    cout << expr.value() << "\n";

    return 0;
}
```

Output:

# ADD AN OPERATION

```
cout << to_rpn(expr) << " = " << expr->value() << "\n";  
//           ^^^^^^
```

Output:

```
2 3 4 * + = 14
```

# VIRTUAL FUNCTION?

```
struct Node {  
    // as before  
    virtual string to_rpn() const = 0;  
};  
  
struct Number : Node {  
    // as before  
    string to_rpn() const override { return to_string(val); }  
};  
  
struct Plus : Node {  
    // as before  
    string to_rpn() const override { return left.to_rpn() + " " + right.to_rpn() + " +"; }  
};  
  
struct Times : Node {  
    // as before  
    string to_rpn() const override { return left.to_rpn() + " " + right.to_rpn() + " &"; }  
};
```

*banana -> gorilla -> jungle*  
*(C) Erlang creator Joe Armstrong*

# TYPE SWITCH?

```
string to_rpn(const Node& node) {
    if (auto expr = dynamic_cast<const Number*>(&node)) {
        return to_string(expr->value());
    } else if (auto expr = dynamic_cast<const Plus*>(&node)) {
        return to_rpn(expr->left) + " " + to_rpn(expr->right) + " +";
    } else if (auto expr = dynamic_cast<const Times*>(&node)) {
        return to_rpn(expr->left) + " " + to_rpn(expr->right) + " *";
    }
    throw runtime_error("unknown node type");
}
```

- operations += types: nope

# VISITOR?

```
struct Node {  
    // as before  
    struct Visitor {  
        virtual void accept(const Number& expr) = 0;  
        virtual void accept(const Plus& expr) = 0;  
        virtual void accept(const Times& expr) = 0;  
    };  
    virtual void visit(Visitor& viz) const = 0;  
};  
  
struct Number : Node {  
    // as before  
    void visit(Visitor& viz) override { viz.accept(*this); }  
};  
  
struct Plus : Node {  
    void visit(Visitor& viz) override { viz.accept(*this); }  
};  
// etc.
```

# VISITOR...

```
struct RPNVisitor : Node::Visitor {
    void accept(const Number& expr) {
        result = to_string(expr.val);
    }
    void accept(const Plus& expr) {
        expr.left.visit(*this);
        string l = result;
        expr.right.visit(*this);
        result = l + " " + result + " +";
    }
    void accept(const Times& expr) { ... }
    string result;
};

string to_rpn(const Node& node) {
    RPNVisitor viz;
    node.visit(viz);
    return viz.result;
}
```

- a lot of work
- more visitors, or more complexity (non-const...)
- operations += types: nope

# FUNCTION TABLE?

```
unordered_map<type_index, string (*)(<b>const</b> Node&)> RPNformatters;

string <b>to_rpn</b>(<b>const</b> Node& node) {
    <b>return</b> RPNformatters[<b>typeid</b>(node)](node);
}

<b>struct</b> Init {
    Init() {
        RPNformatters[<b>typeid</b>(Number)] = [<b>]</b>(<b>const</b> Node& node) {
            <b>return</b> <b>to_string</b>(<b>static_cast</b><b>const</b> Number&>(node).val); </b>};
        RPNformatters[<b>typeid</b>(Plus)] = [<b>]</b>(<b>const</b> Node& node) {
            auto expr = <b>static_cast</b><b>const</b> Plus&>(node);
            <b>return</b> <b>to_rpn</b>(expr.left) + " " + <b>to_rpn</b>(expr.right) + " +"; </b>};
        RPNformatters[<b>typeid</b>(Times)] = [<b>]</b>(<b>const</b> Node& node) {
            auto expr = <b>static_cast</b><b>const</b> Times&>(node);
            <b>return</b> <b>to_rpn</b>(expr.left) + " " + <b>to_rpn</b>(expr.right) + " *"; </b>};
    }
} init;
```

- types += operations: yes
- operations += types: yes

# POLL

Only one vote!

1. virtual function
2. type switch
3. visitor
4. function table
5. they all stink

# **OPEN METHODS**

# OPEN METHODS

- free virtual functions
  - i.e. virtual functions that exist outside of a class
- existing types += new operations

# YOMM2 OPEN METHODS

```
struct Node {  
    virtual string to_rpn(/* const Node* */) const = 0;  
};
```

```
declare_method(string, to_rpn, (virtual_<const Node&>));
```

Common Lisp: defgeneric, Clojure: defmulti

```
struct Plus : Node {  
    string to_rpn(/* const Node* this */) const override {  
        return left.to_rpn() + " " + right.to_rpn() + " +";  
    }  
};
```

```
define_method(string, to_rpn, (const Plus& expr)) {  
    return to_rpn(expr.left) + " " + to_rpn(expr.right) + " +";  
}
```

Common Lisp, Clojure: defmethod

# AST

```
#include <yorel/yomm2/keywords.hpp>

declare_method(string, to_rpn, (virtual_<const Node&>));

define_method(string, to_rpn, (const Number& expr)) {
    return std::to_string(expr.val);
}

define_method(string, to_rpn, (const Plus& expr)) {
    return to_rpn(expr.left) + " " + to_rpn(expr.right) + " +";
}

define_method(string, to_rpn, (const Times& expr)) {
    return to_rpn(expr.left) + " " + to_rpn(expr.right) + " *";
}

register_classes(Node, Number, Plus, Times);

int main() {
    yorel::yomm2::update();
    cout << to_rpn(expr) << " = " << expr.value() << "\n";
    return 0;
}
```

# AST: WHAT ABOUT VALUE?

- value in the node hierarchy => interpreter
- AST classes should *only* represent the tree

```
declare_method(int, value, (virtual_<Node&>));  
  
define_method(int, value, (Number& expr)) {  
    return expr.val;  
}  
  
define_method(int, value, (Plus& expr)) {  
    return value(expr.left) + value(expr.right);  
}
```

# PERFORMANCE

```
mov    rax, qword ptr [rdi]
mov    rdx, qword ptr [rip+fast_perfect_hash<release>::mult]

imul   rdx, qword ptr [rax-8]
mov    cl, byte ptr [rip+fast_perfect_hash<release>::shift]

shr    rdx, cl
mov    rax, qword ptr [rip+vptr_vector<release>::vptrs]

mov    rax, qword ptr [rax+8*rdx]
mov    rcx, qword ptr [rip+method<value, int (virtual_<Node const&>)::fn+80]

jmp    qword ptr [rax+8*rcx]
```

- 15-30% slower than equivalent native virtual function call (but see `virtual_ptr`)
- Optimizing Away C++ Virtual Functions May Be Pointless - Shachar Shemesh - CppCon 2023

# MULTIPLE DISPATCH

sometimes useful

```
add(Matrix, Matrix)           -> Matrix  
                                add all elements  
add(DiagonalMatrix, DiagonalMatrix) -> DiagonalMatrix  
                                just add diagonals  
  
fight(Human, Creature, Axe)    -> not agile enough to wield  
fight(Warrior, Creature, Axe)   -> chop it into pieces  
fight(Warrior, Dragon, Axe)     -> die a honorable death  
fight(Human, Dragon, Hands)    -> congratulations! you have just  
                                vanquished a dragon with your  
                                bare hands! (unbelievable,  
                                isn't it?)
```

- works just like selecting from set of overloads (but at runtime!)
- ambiguities can arise

# POLL

## IS THIS OOP?

Only one vote!

1. Yes
2. No

# POLL

## WHAT DO YOU PREFER?

Only one vote!

1. virtual function, type switch, visitor, function table
2. open methods

# **INSIDE YOMM2**

# INSIDE YOMM2

- purely in C++17 (no extra tooling)
- constant time dispatch
- uses tables of function pointers
- object -> dispatch data?
  - perfect integer hash of `&type_info`

# A PAYROLL APPLICATION

- *Role*
  - Employee
    - Manager
  - Founder
- *Expense*
  - Cab, Jet
  - *Public*
    - Bus, Train

# THE pay UNI-METHOD

```
declare_method(double, pay, (virtual_<Employee&>));  
  
define_method(double, pay, (Employee&)) {  
    return 3000;  
}  
  
define_method(double, pay, (Manager& manager)) {  
    return next(manager) + 2000;  
}
```

# DECLARE\_METHOD

```
declare_method(double, pay, (virtual_<Employee&>));
```

```
struct YoMm2_S_pay;  
yomm2::method<  
    YoMm2_S_pay, double(virtual_<const Employee&>),  
    yomm2::default_policy>  
pay_yOMM2_selector_(  
    yomm2::detail::remove_virtual<virtual_<const Employee&>> a0);
```

# DECLARE\_METHOD

```
declare_method(double, pay, (virtual_<Employee&>));  
  
inline double  
pay(yomm2::detail::remove_virtual<virtual_<const Employee&>> a0) {  
    return yomm2::method<  
        YoMm2_S_pay, double(virtual_<const Employee&>),  
        yomm2::default_policy>::  
    fn(std::forward<  
        yomm2::detail::remove_virtual<virtual_<const Employee&>>>(a0));  
};
```

# DEFINE\_METHOD

```
define_method(double, pay, (Employee&)) { return 3000; }
```

```
namespace { namespace YoMm2_gS_10 {
template<typename T> struct _yOMM2_select;
template<typename... A> struct _yOMM2_select<void(A...)> {
    using type = decltype(pay_yOMM2_selector_(std::declval<A>()...));
};
using _yOMM2_method = _yOMM2_select<void(const Employee&)>::type;
using _yOMM2_return_t = _yOMM2_method::return_type;
_yOMM2_method::function_pointer_type next;
struct _yOMM2_spec {
    static YoMm2_gS_10::_yOMM2_method::return_type
    yOMM2_body(const Employee&);
};
_yOMM2_method::add_function<_yOMM2_spec::yOMM2_body>
    YoMm2_gS_11(&next, typeid(_yOMM2_spec).name()); } }
```

# DEFINE\_METHOD

```
define_method(double, pay, (Employee&)) { return 3000; }
```

```
YoMm2_gS_10::yOMM2_method::return_type  
YoMm2_gS_10::yOMM2_spec::yOMM2_body(const Employee&) {  
    return 3000;  
}
```

# UPDATE

uses the class and method info registered by static ctors

- build representation of class hierarchies
- calculate the hash and dispatch tables
- find a perfect (not minimal) hash function for the `type_infos`
  - $H(x) = (M * x) \gg S$

# DISPATCHING A UNI-METHOD

- pretty much like virtual member functions
- method table contains a pointer to the effective function
- only it is not at a fixed offset in the method table

# DISPATCHING A UNI-METHOD

during update

```
method<pay>::slots_strides.i = 1;

// method table for Employee
mtbls[ H(&typeid(Employee)) ] = {
    ... // used by approve,
    wrapper(pay(Employee&))
};

// method table for Manager
mtbls[ H(&typeid(Manager&)) ] = {
    ... // used by approve,
    wrapper(pay(Manager&))
};
```

# DISPATCHING A UNI-METHOD

pay(bill)

=>

```
mtbls[ H(&typeid(bill)) ]      // mtable for type
  [ method<pay>::slots_strides.i ] // pointer to fun
(bill)                           // call
```

# ASSEMBLER

```
double call_pay(Employee& e) { return pay(e); }
```

```
mov r8, qword ptr [rip + context+24]          ; hash table
mov rdx, qword ptr [rip + context+32]          ; M
mov cl, byte ptr [rip + context+40]            ; S
movsx rsi, dword ptr [rip + method<pay>::fn+96] ; slot
mov rax, qword ptr [rdi]                        ; vptr
imul rdx, qword ptr [rax - 8]                  ; M * &typeid(e)
shr rdx, cl                                     ; >> S
mov rax, qword ptr [r8 + 8*rdx]                ; method table
jmp qword ptr [rax + 8*rsi]                     ; call wrapper
```

# approve MULTI-METHOD

```
declare_method(bool, approve,
  (virtual_<Role&>, virtual_<Expense&>, double));
define_method(bool, approve, (Role& r, Expense& e, double amount)) {
  return false;
}
define_method(bool, approve, (Employee& r, Public& e, double amount)) {
  return true;
}
define_method(bool, approve, (Manager& r, Taxi& e, double amount)) {
  return true;
}
define_method(bool, approve, (Founder& r, Expense& e, double amount)) {
  return true;
}
```

# DISPATCHING A MULTI-METHOD

- it's a little more complicated
- use a multi-dimensional dispatch table
- size can grow very quickly
- the table must be "compressed", devoid of redundancies
- in fact the "uncompressed" table never exists
- work in terms of class *groups*, not classes

# DISPATCHING A MULTI-METHOD

	Expense+Jet	Public+Bus+Train	Cab
Role	R,X	R,X	R,X
Employee	R,X	E,P	R,X
Manager	R,X	E,P	M,C
Founder	F,X	F,X	F,X
(column major)			

# BUILDING THE DISPATCH TABLE

- Fast Algorithms for Compressed Multi-Method Dispatch, Eric Amiel, Eric Dujardin, Eric Simon, 1996
- Open Multi-Methods for C++11, Part 3 - Inside Yomm11: Data Structures and Algorithms, Jean-Louis Leroy, 2013

# DISPATCHING A MULTI-METHOD

```
method<approve>:::slots_strides.pw = { 0, 4, 0 };

mtbls[ H(&typeid(Employee)) ] = {
    // & of (Employee,Expense+Jet) cell
    // used by pay
};

mtbls[ H(&typeid(Manager)) ] = {
    // & of (Manager,Expense+Jet) cell
    // used by pay
};

mtbls[ H(&typeid(Expense)) ] = { 0 }; // also for Jet
mtbls[ H(&typeid(Public)) ] = { 1 }; // also for Bus, Train
mtbls[ H(&typeid(Cab)) ] = { 2 };
```

# DISPATCHING A MULTI-METHOD

```
approve(bill, ticket, 5000)
```

=>

```
word* slots_strides = method<approve>::slots_strides.pw;  
  
mtbls[ H(&typeid(bill)) ]           // method table for bill  
[ slots_strides[0].i ]               // ptr to cell in 1st column  
[ mtbls [ H(&typeid(ticket)) ] // method table for ticket  
  [ slots_strides[2].i ]           // column  
  * slots_strides[1].i            // stride  
]  
(bill, ticket, 5000)                // pointer to function  
                                     // call
```

# BENCHMARKS

		gcc6	clang6
normal inheritance			
virtual function	1-method	16%	17%
double dispatch	2-method	25%	35%
virtual inheritance			
virtual function	1-method	19%	17%
double dispatch	2-method	40%	33%

# YOMM2 VS OTHER SYSTEMS

- Pirkelbauer - Solodkyi - Stroustrup (PSS)
- yomm11
- Cmm
- Loki / Modern C++

# YOMM2 VS PSS

- Solodkyi's papers on open methods etc.:
  - Open Multi-Methods for C++
  - Design and Evaluation of C++ Open Multi-Methods
  - Simplifying the Analysis of C++ Programs
- PSS attempts harder to resolve ambiguities
- yomm2 overrides not visible as overloads, cannot specialize multiple methods
- yomm2 supports smart pointers, next

# **YOMM2 VS YOMM11**

- no need to instrument classes
- methods are ordinary functions

# **EVOLUTION OF YOMM2**

# PAST

- goals:
  - help promote adoption in the standard
    - submit to Boost
    - talk about it (CppCon 2018...)
    - 2018-2020: only bug fixes, cleanup...
- results:
  - Boost community: no interest
  - standard committee: no interest

# PRESENT

- 2020: give up on adoption in the standard
- new developments

# VIRTUAL\_PTR

```
int call_via_vptr(virtual_ptr<const Node> node) {
    return value(node);
}

mov rax, qword ptr [rip + method<value, int (virtual_ptr<Node>)::fn+80]
mov rax, qword ptr [rsi + 8*rax]
jmp rax

declare_method(int, value, (virtual_ptr<Node>));
```

# VIRTUAL\_PTR

```
auto make_node_ptr(const Node& node) {
    return virtual_ptr(node);
}
```

```
mov rax, rdi
mov rcx, qword ptr [rdi]
mov rdx, qword ptr [rcx - 8]
lea rcx, [rip + typeinfo for Node]
cmp rdx, rcx
je .LBB7_1
imul rdx, qword ptr [rip + fast_perfect_hash<release>::hash_mult]
movzx ecx, byte ptr [rip + fast_perfect_hash<release>::hash_shift]
shr rdx, cl
shl rdx, 3
add rdx, qword ptr [rip + vptr_vector<release>::vptrs]
mov rdx, qword ptr [rdx]
ret
.LBB7_1:
lea rdx, [rip + method_tables<release>::static_vptr<Node>]
mov rdx, qword ptr [rdx]
ret
```

# VIRTUAL\_PTR

```
auto make_final_node_ptr(const Plus& node) {
    return final_virtual_ptr(node);
}
```

classes need not be polymorphic

```
mov rax, rdi
mov rdx, qword ptr [rip + method_tables<release>::static_vptr<Plus>]
ret
```

# CORE API

```
use_classes<Node, Number, Plus, Times> use_ast_classes;

struct value_id;
using value = method<value_id, int(virtual_<const Node&>)};

auto result = value::fn(expr);

int number_value(const Number& node) {
    return node.val;
}
value::add_function<number_value> add_number_value;

template<class NodeClass, class Op>
struct binary_value {
    static int fn(const NodeClass& expr) {
        return Op()(value::fn(expr.left), value::fn(expr.right));
    }
};

YOMM2_STATIC(value::add_definition<binary_value<Plus, std::plus<int>>>);
YOMM2_STATIC(value::add_definition<binary_value<Times, std::multiplies<int>>>);
```

# PRESENT

- `virtual_ptr`
- core API
- template interop toolkit
- header only (+ Compiler Explorer)
- friendship
- member methods
- policies and facets
  - custom RTTI
  - custom error handling, trace, vptr placement...

# FUTURE

- goals:
  - match (beat?) virtual function speed
  - pre-calculate dispatch tables
  - malloc-free operation
  - C++20

# Q&A

GitHub:



examples are on Compiler Explorer:

<https://jll63.github.io/yomm2/ce/slides.html>  
(redirects to volatile godbolt.org short URL)