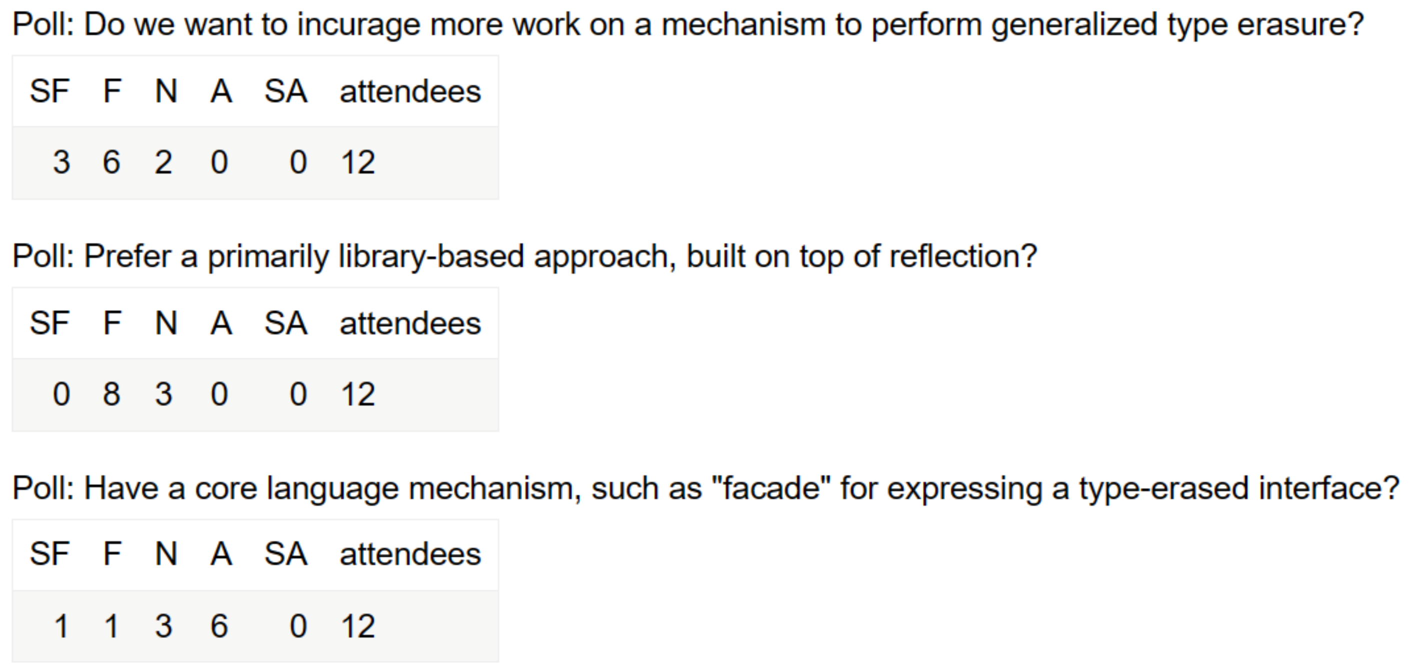
Proxy: A Pointer-Semantics-Based Polymorphism Library

Mingxin Wang <mingxwa@microsoft.com>

https://isocpp.org/files/papers/P3086R1.pdf

Introduction

- Review of P0957R3 in EWGI in Belfast (2019)
- https://wiki.edg.com/bin/view/Wg21belfast/P0957



Poll: Do we want to incurage more work on a mechanism to perform generalized type erasure?

SF	F	N	A	SA	attendees
3	6	2	0	0	12

Poll: Prefer a primarily library-based approach, built on top of reflection?

SF	F	N	A	SA	attendees
0	8	3	0	0	12

Poll: Have a core language mechanism, such as "facade" for expressing a type-erased interface?

SF	F	N	A	SA	attendees
1	1	3	6	0	12

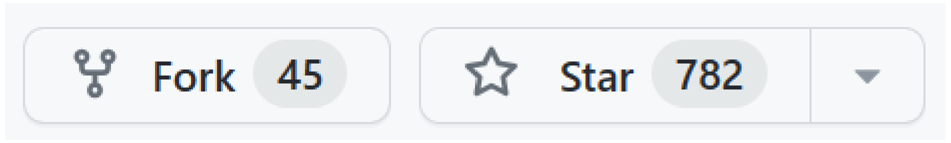
Introduction

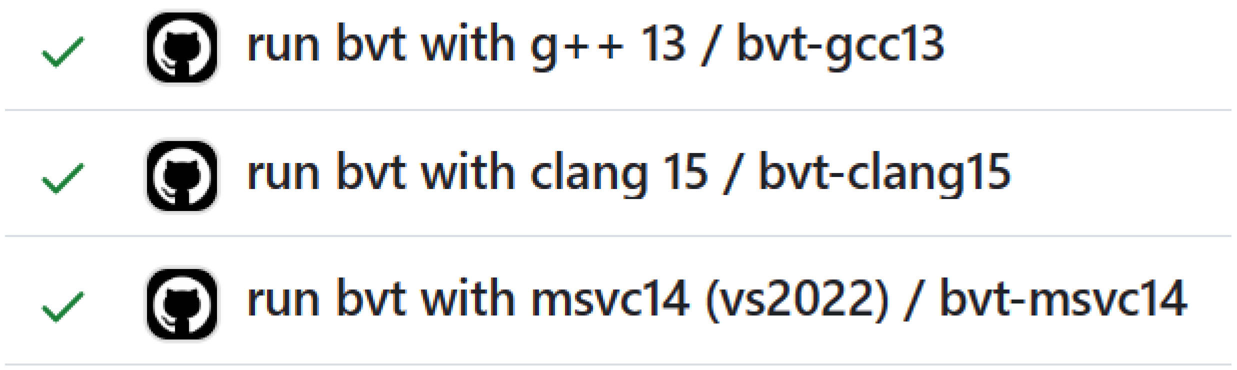
- Review of P0957R8 in LEWG mailing list (2022)
- https://lists.isocpp.org/lib-ext/2022/06/23355.php

- Library feature or language feature?
- Deployment experience and usage feedback?
- any_object, any_ref, any_unique, any_shared?
- Research and comparison to existing libraries, like Boost.TypeErasure?

Introduction

- Open-sourced: https://github.com/microsoft/proxy
- Deployed in Windows via vcpkg





✓ run bvt with g++ 13 / bvt-gcc13

✓ run bvt with clang 15 / bvt-clang15

✓ run bvt with msvc14 (vs2022) / bvt-msvc14

Overview

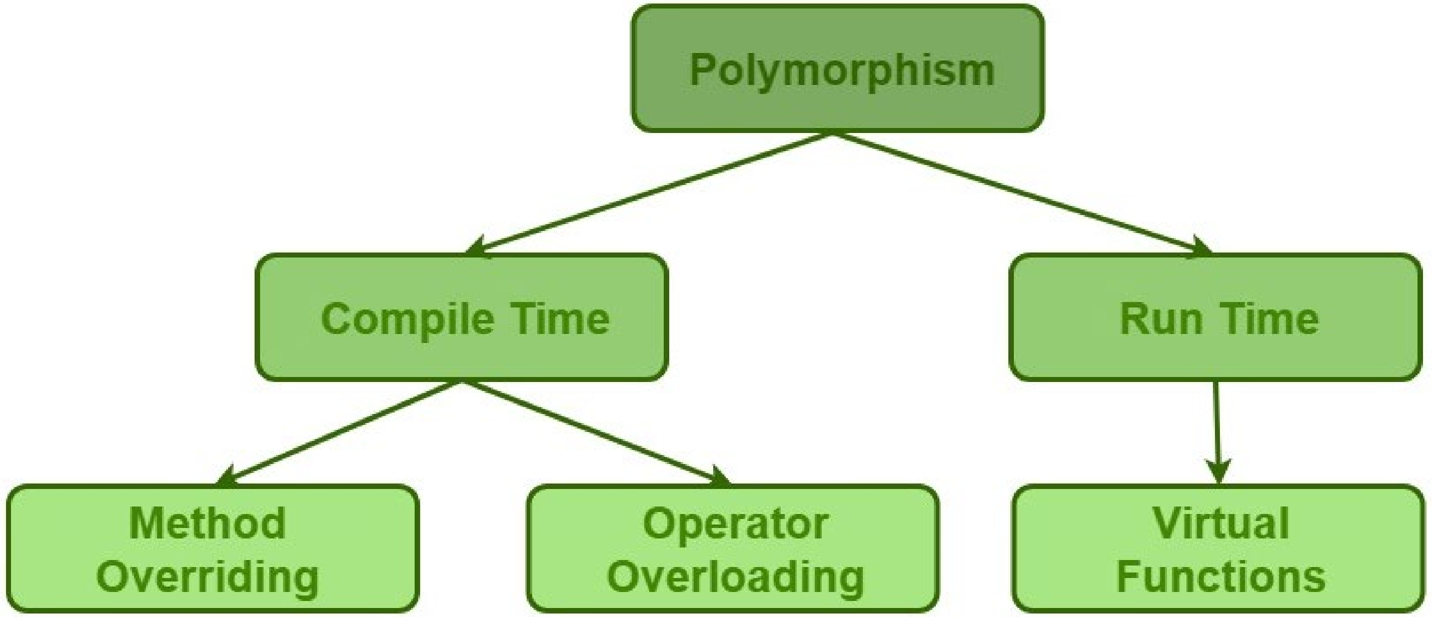
- Motivation
- Considerations
- Demo

Motivation

- Runtime polymorphism is a useful paradigm of abstraction
- Virtual functions in C++ has certain limitations
- Existing polymorphic wrappers in C++ are not sufficient
- We want to write polymorphic code as easily as in Java or C# (while keeping it C++ quality)

Motivation – Runtime polymorphism

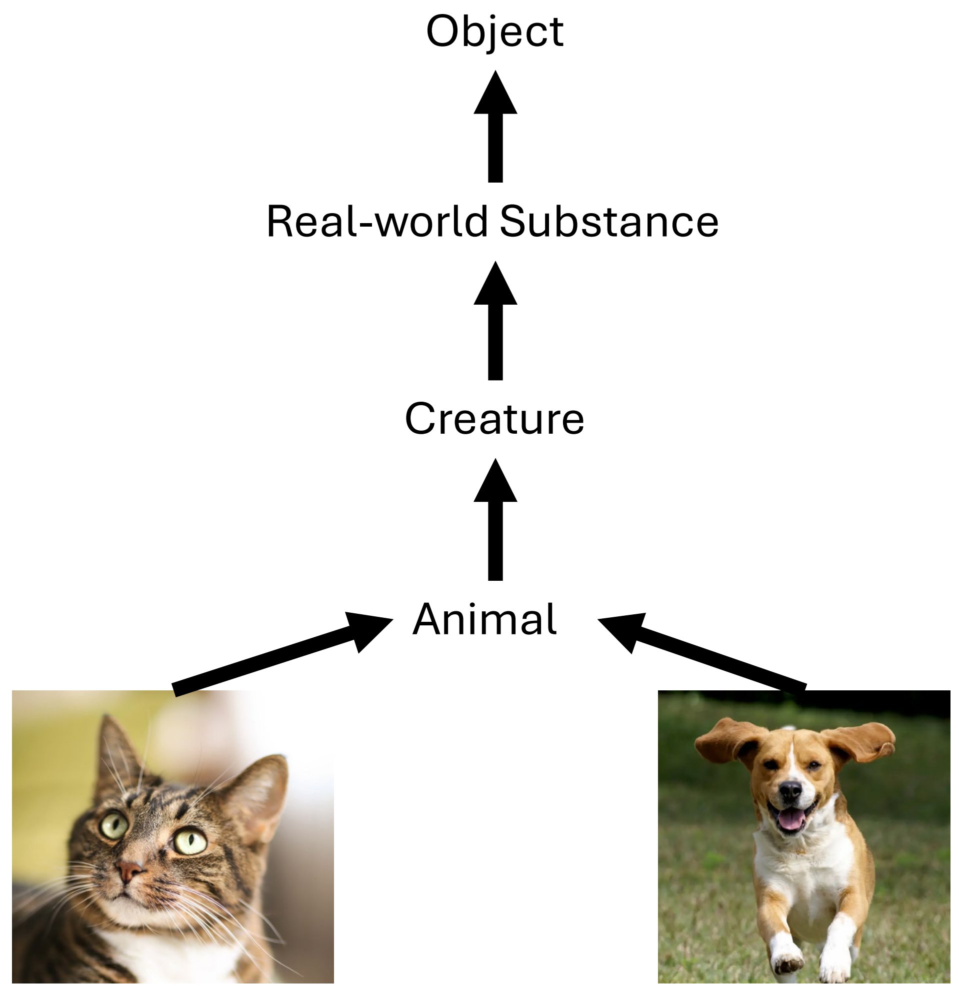
- Polymorphism is the provision of a single interface to entities of different types. – Bjarne



Motivation – Virtual functions

- Being intrusive is sometimes acceptable, but is not making everyone comfortable
  - Architecting challenge
  - Loosing potential for compile-time optimizations
- Lifetime management is missing, making people write bad code sometimes
  - Memory leak
  - Abuse of std::shared_ptr

Motivation – Virtual functions



```
void TakeCareOf(Pet);
```

Existing codebase

Your job

Motivation

- Runtime polymorphism is a useful paradigm of abstraction
- Virtual functions in C++ has certain limitations
- Existing polymorphic wrappers in C++ are not sufficient
- We want to write polymorphic code as easily as in Java or C# (while keeping it C++ quality)

Motivation – Virtual functions

```cpp
class IDrawable {
public:
    virtual void Draw() const = 0;
};

class Rectangle : public IDrawable {
public:
    void Draw() const override;
};
```

C++

```rust
trait IDrawable {
    fn draw(&self);
}

struct Rectangle;

impl IDrawable for Rectangle {
    fn draw(&self) {}
}
```

Rust

Motivation – Virtual functions

- Being intrusive is sometimes acceptable, but is not making everyone comfortable
  - Architecting challenge
  - Loosing potential for compile-time optimizations
- Lifetime management is missing, making people write bad code sometimes
  - Memory leak
  - Abuse of std::shared_ptr

Motivation – Virtual functions

```
class IDrawable {
public:
    virtual void Draw() const = 0;
};

class Rectangle : public IDrawable {
public:
    void Draw() const override;
};

Rectangle rect;
rect.Draw();
```

Motivation – Virtual functions

```
class IDrawable {
public:
    virtual void Draw() const = 0;
};

class Rectangle : public IDrawable {
public:
    void Draw() const override;
};

Rectangle rect;
rect.Draw();
```

Thank you compiler! But...

Motivation – Virtual functions

```
class Shape {
public:
    virtual ~Shape() = default;
};
class Polygon : public Shape {};
class TwoDShape : public Shape {};
class Rectangle : public Polygon, public TwoDShape {};
```

Motivation – Virtual functions

- Being intrusive is sometimes acceptable, but is not making everyone comfortable
  - Architecting challenge
  - Loosing potential for compile-time optimizations
- Lifetime management is missing, making people write bad code sometimes
  - Memory leak
  - Abuse of std::shared_ptr

Motivation – Virtual functions

```
class IDrawable {
public:
    virtual void Draw() const = 0;
};

class Rectangle : public IDrawable {
public:
    void Draw() const override;
};
```

Motivation – Virtual functions

```
class IDrawable {
public:
    virtual void Draw() const = 0;
};

class Rectangle : public IDrawable {
public:
    void Draw() const override;
};

IDrawable* drawable = new Rectangle();
delete drawable;
```

Motivation – Virtual functions

```
class IDrawable {
public:
    virtual void Draw() const = 0;
    virtual ~IDrawable() = default;
};

class Rectangle : public IDrawable {
public:
    void Draw() const override;
};

IDrawable* drawable = new Rectangle();
delete drawable;
```

Motivation – Virtual functions

- Being intrusive is sometimes acceptable, but is not making everyone comfortable
  - Architecting challenge
  - Loosing potential for compile-time optimizations
- Lifetime management is missing, making people write bad code sometimes
  - Memory leak
  - Abuse of std::shared_ptr

Motivation – Virtual functions

```
?? MakeDrawableFromCommand(const std::string& s);
```

Motivation – Virtual functions

```
?? MakeDrawableFromCommand(const std::string& s);

IDrawable* MakeDrawableFromCommand(const std::string& s) ?

std::unique_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?

std::shared_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?

some_gc_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?
```

Motivation

- Runtime polymorphism is a useful paradigm of abstraction
- Virtual functions in C++ has certain limitations
- Existing polymorphic wrappers in C++ are not sufficient
- We want to write polymorphic code as easily as in Java or C# (while keeping it C++ quality)

Motivation – Existing polymorphic wrappers

```
std::function
std::packaged_task
std::move_only_function
std::any
```

Motivation – Existing polymorphic wrappers

```
std::function
std::packaged_task
std::move_only_function
std::any
```

```
More member functions / overloads?
Reflection?
Small Buffer Optimization (SBO)?
Shared ownership?
```

Motivation

- Runtime polymorphism is a useful paradigm of abstraction
- Virtual functions in C++ has certain limitations
- Existing polymorphic wrappers in C++ are not sufficient
- We want to write polymorphic code as easily as in Java or C# (while keeping it C++ quality)

Motivation

```
?? MakeDrawableFromCommand(const std::string& s);

IDrawable* MakeDrawableFromCommand(const std::string& s) ?

std::unique_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?

std::shared_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?

some_gc_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?
```

Motivation

```
?? MakeDrawableFromCommand(const std::string& s);

~~IDrawable* MakeDrawableFromCommand(const std::string& s) ?~~

~~std::unique_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?~~

~~std::shared_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?~~

~~some_gc_ptr<IDrawable> MakeDrawableFromCommand(const std::string& s) ?~~

proxy<Drawable> MakeDrawableFromCommand(const std::string& s);
```

Overview

- Motivation
- Considerations
- Demo

Considerations

- Pointer semantics!
- Capabilities
- Library or Core?
- QoI

Considerations – Pointer semantics

- Roadmap
- Constraints
- Implementation

Considerations – Pointer semantics

- Semantics based on configuration since P0957R0
  - value semantics, reference semantics, etc.
- Direction changed since P0957R5
- C++ pointer types are good at storage and lifetime management!

Considerations – Pointer semantics

- Roadmap
- Constraints
- Implementation

Considerations – Pointer semantics

- Constraints

```
enum class constraint_level { none, nontrivial, nothrow, trivial };

struct proxiable_ptr_constraints {
  std::size_t max_size;
  std::size_t max_align;
  constraint_level copyability;
  constraint_level relocatability;
  constraint_level destructibility;
};
```

Considerations – Pointer semantics

- Default constraints

Constraints	Defaults
Maximum size	No less than the size of a pointer
Maximum alignment	No less than the alignment of a pointer
Copyability	Trivial
Relocatability	Trivial
Destructibility	Trivial

Default constraints of relocatable pointer types

Constraints	Defaults
Maximum size	No less than the size of two pointers
Maximum alignment	No less than the alignment of a pointer
Copyability	None
Relocatability	Nothrow
Destructibility	Nothrow

Default constraints of trivial pointer types

Constraints	Defaults
Maximum size	No less than the size of two pointers
Maximum alignment	No less than the alignment of a pointer
Copyability	Nontrivial
Relocatability	Nothrow
Destructibility	Nothrow

Default constraints of copyable pointer types

Considerations – Pointer semantics

- Roadmap
- Constraints
- Implementation

Considerations – Pointer semantics

- Inspired by GCC implementation of std::move_only_function

```
template<typename _Tp, typename _Self>
  static _Tp*
  _S_access(_Self* __self) noexcept
  {
    if constexpr (__stored_locally<remove_const_t<_Tp>>)
      return static_cast<_Tp*>(__self->_M_storage._M_addr());
    else
      return static_cast<_Tp*>(__self->_M_storage._M_p);
  }
```

Considerations – Pointer semantics

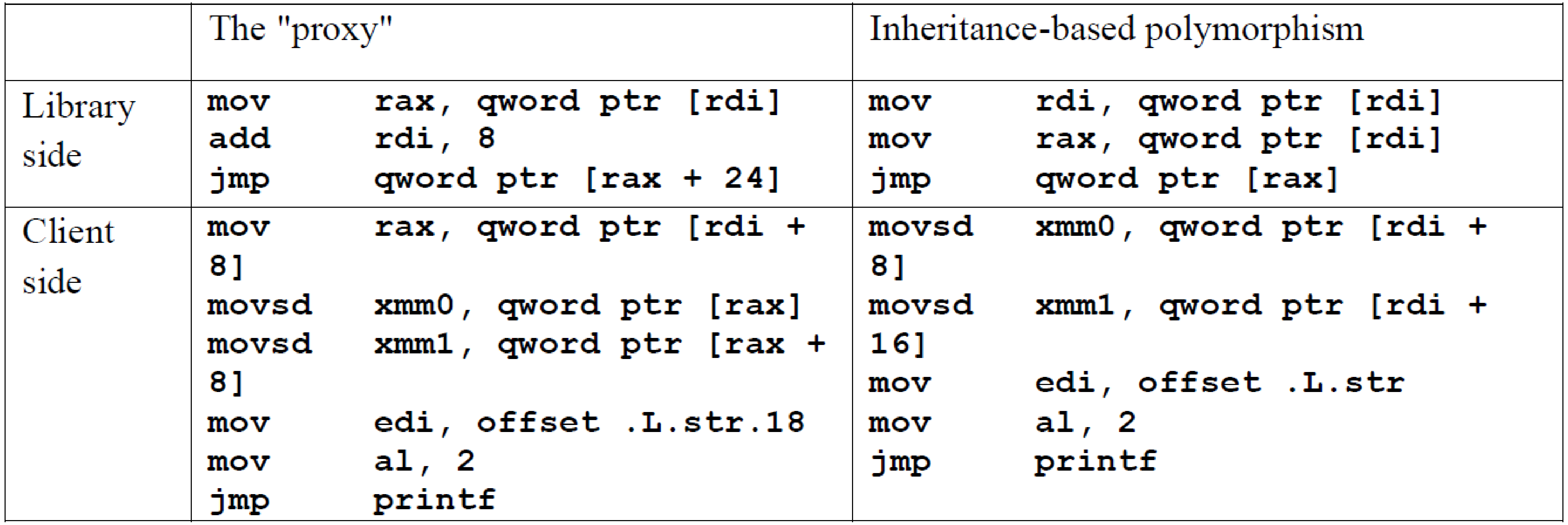
```
struct IDrawable {
  virtual void Draw() const = 0;
  virtual double Area() const = 0;
  virtual ~IDrawable() {}
};
```

```
class Rectangle : public IDrawable {
 public:
  void Draw() const override {
    printf("{Rectangle: width = %f, height = %f}", width_, height_);
  }
  double Area() const override
      { return width_ * height_; }

 private:
  double width_;
  double height_;
};
```

```
void DoSomethingWithDrawable(std::unique_ptr<IDrawable> p) {
  p->Draw();
}
```

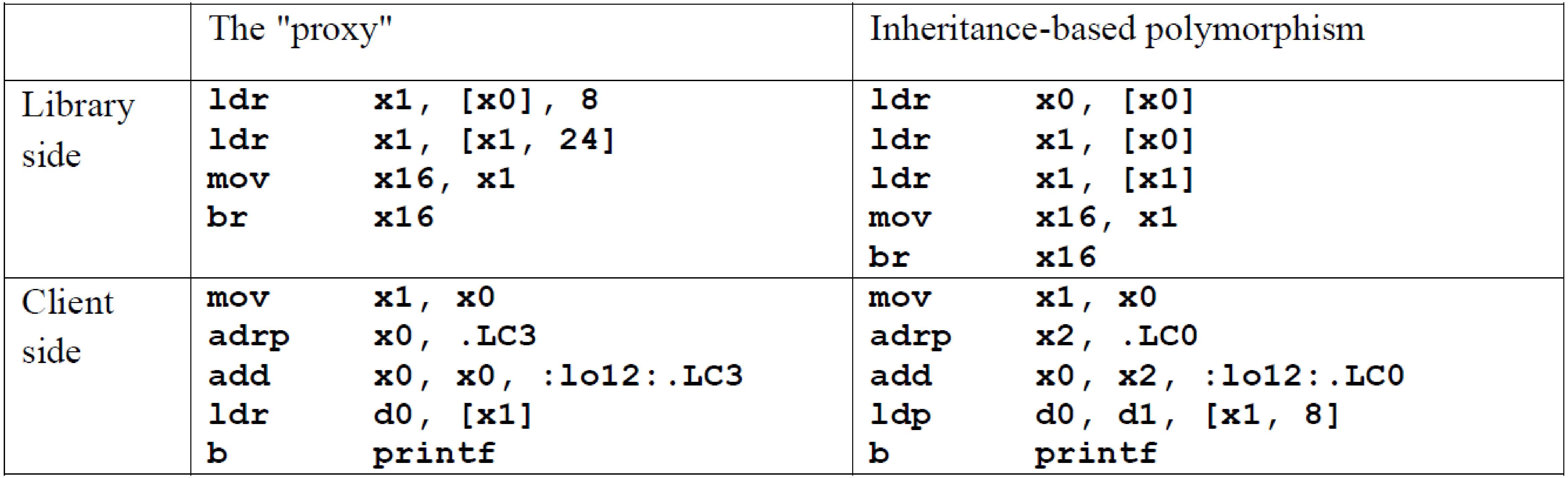
Considerations – Pointer semantics



	The "proxy"	Inheritance-based polymorphism
Library side	`mov rax, qword ptr [rdi]` `add rdi, 8` `jmp qword ptr [rax + 24]`	`mov rdi, qword ptr [rdi]` `mov rax, qword ptr [rdi]` `jmp qword ptr [rax]`
Client side	`mov rax, qword ptr [rdi + 8]` `movsd xmm0, qword ptr [rax]` `movsd xmm1, qword ptr [rax + 8]` `mov edi, offset .L.str.18` `mov al, 2` `jmp printf`	`movsd xmm0, qword ptr [rdi + 8]` `movsd xmm1, qword ptr [rdi + 16]` `mov edi, offset .L.str` `mov al, 2` `jmp printf`

Table 6 – Generated code from clang 13.0.0 (x86-64)

Considerations – Pointer semantics



	The "proxy"	Inheritance-based polymorphism

Library side	`ldr x1, [x0], 8`
	`ldr x1, [x1, 24]`
	`mov x16, x1`
	`br x16`
		`ldr x0, [x0]`
		`ldr x1, [x0]`
		`ldr x1, [x1]`
		`mov x16, x1`
		`br x16`

Client side	`mov x1, x0`
	`adrp x0, .LC3`
	`add x0, x0, :lo12:.LC3`
	`ldr d0, [x1]`
	`b printf`
		`mov x1, x0`
		`adrp x2, .LC0`
		`add x0, x2, :lo12:.LC0`
		`ldp d0, d1, [x1, 8]`
		`b printf`

Table 7 – Generated code from gcc 11.2 (ARM64)

Considerations

- Pointer semantics!
- Capabilities
- Library or Core?
- QoI

Considerations – Capabilities

- Reflection
- Overloading
- Multiple dispatches

Considerations – Capabilities

```
template <class F>
concept facade;

template <class P, class F>
concept proxiable;

template <class F>
class proxy;

template <class F, class T, class... Args>
proxy<F> make_proxy(Args&&... args);
```

Considerations

- Pointer semantics!
- Capabilities
- Library or Core?
- QoI

Considerations – Library or Core?

```
struct Draw {
  using overload_types = std::tuple<void()>;
  template <class T>
  void operator()(T& self) requires(requires{ self.Draw(); }) {
    self.Draw();
  }
};
struct Drawable {
  using dispatch_types = Draw;
  static constexpr auto constraints = std::relocatable_ptr_constraints;
  using reflection_type = void;
};
```

Considerations – Library or Core?

```
PRO_DEF_MEMBER_DISPATCH(Draw, void());
PRO_DEF_FACADE(Drawable, Draw);
```

Considerations

- Pointer semantics!
- Capabilities
- Library or Core?
- QoI

Considerations – QoI

- Constraints prototypes
- Inline
- noexcept
- is_constval()

```
template <class Expr>

consteval bool is_consteval(Expr) {
  return requires {
    typename std::bool_constant<(Expr{}(), false)>;
  };
}
```

Overview

- Motivation
- Considerations
- Demo

Demo

- https://godbolt.org/z/6EWr4G1KM
- https://godbolt.org/z/cd55hrGv3
- https://godbolt.org/z/Yno7qnGz4
- https://godbolt.org/z/voEacxT76
- https://godbolt.org/z/KTMcP7e9v