C++ Russia 2020

Applying the “Hourglass” in Library Design

Sergey Nepomnyachiy

TechAtBloomberg.com

Bloomberg

Engineering

Prior Art

- Stefanus DuToit
  "Hourglass Interfaces for C++ APIs"
  (CPPCON 2014)
- David R. Ibeas
  "Hourglass Interfaces. Stable Library API"
  (Using std::cpp 2017)

- Javier García Sogo
  "ABI compatibility is not a MAJOR problem"
  (C++ Russia 2019)

Bloomberg

Engineering

Contents

- Introduction
  - Problem(s)
  - API compatibility
  - ABI compatibility
  - Hourglass

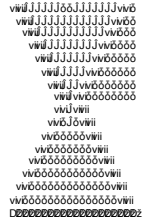
- Detailed discussion
  - Handle wrapper
  - Intrusive types
  - Shared ownership
  - Complex types
  - Type safety

Bloomberg

Engineering

Problem

Shipping a closed-source library to clients we do not control.



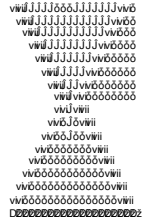
Bloomberg

Engineering

Problem

Shipping a closed-source library to clients we do not control.

- different compilers
- infrequent updates
- no source code



Bloomberg

Engineering

Forced into dll/so

We must support drop-in changes for:

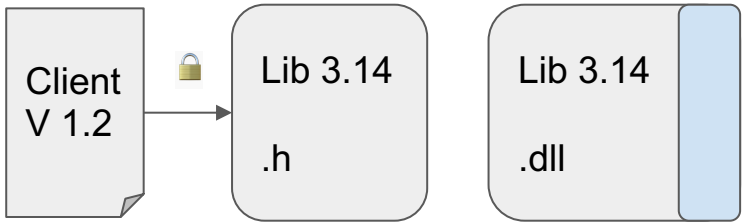
- patch
- upgrade
- reverting

Bloomberg

Engineering

Example

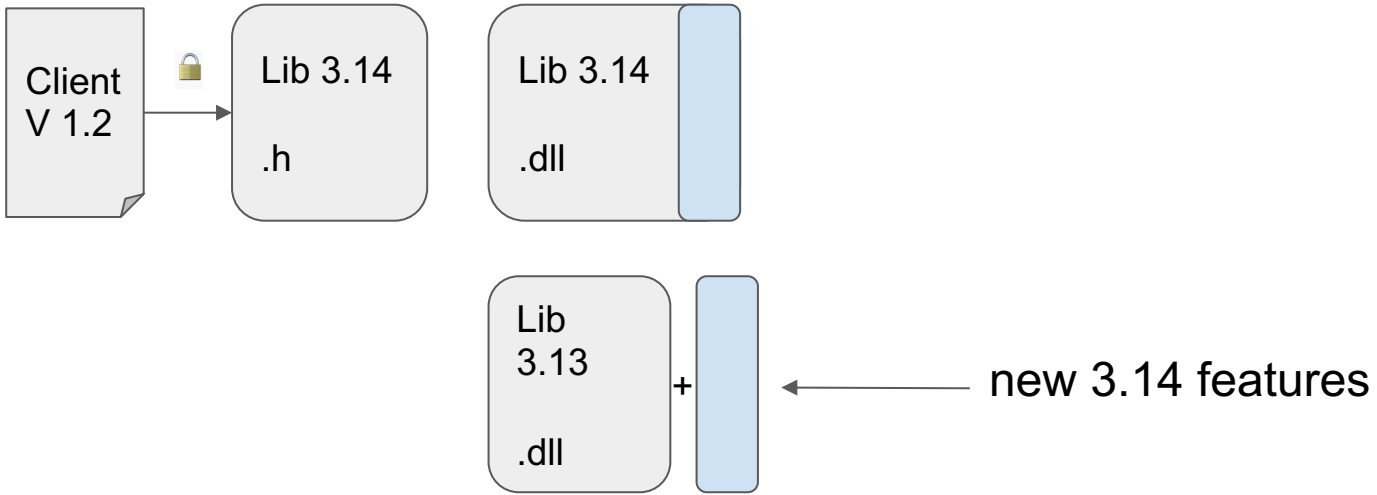
Client app built and shipped with 3.14



Bloomberg
Engineering

Example

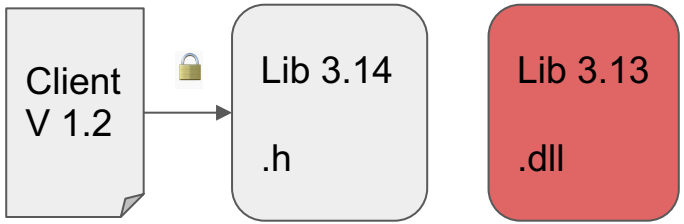
Client app built and shipped with 3.14



Bloomberg
Engineering

Revert

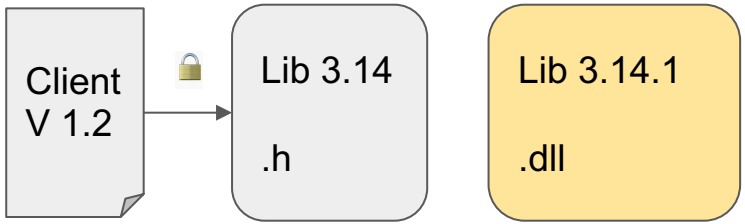
While adding features, we introduced a bug



Bloomberg
Engineering

Patch

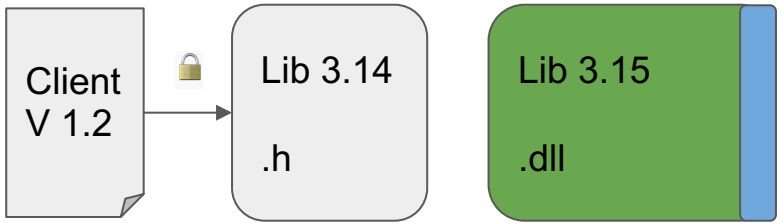
We fixed the bug and release a patched version



Bloomberg
Engineering

Update

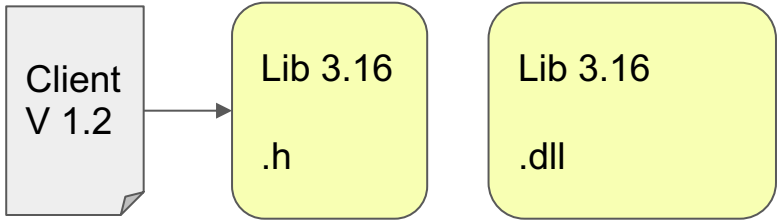
We have even more features now



Bloomberg
Engineering

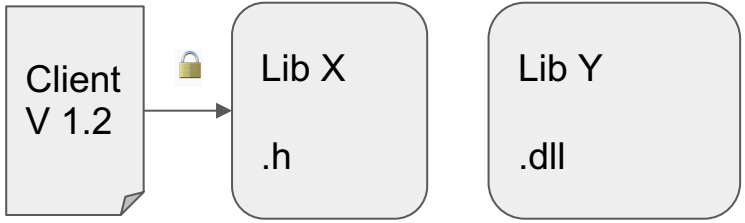
Upgrade

Public API change for new features



Bloomberg
Engineering

Strong Guarantee



- X ≤ Y: works, as we do not introduce breaking changes
- X > Y: works, as long as no new features are used

Bloomberg
Engineering

API Compatibility

(for public facing classes)

- Renaming namespaces/classes/methods
- Changing signatures
  including return types, CV-qualifiers, template args
  (overloads — fine)
- Changing exceptions thrown in code
- Changing types (e.g., global/static constants)
- Making public things private/protected
- Modifying the class hierarchy of a class

  ...

Bloomberg

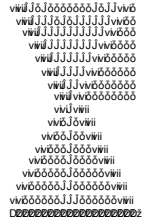
Engineering

ABI

Machine level code interface.

- calling and returning from functions (including sys. calls)
- passing arguments
- data layout
- managing the stack

  ...



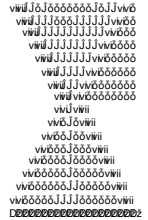
Bloomberg

Engineering

ABI

Needn’t be the same for:

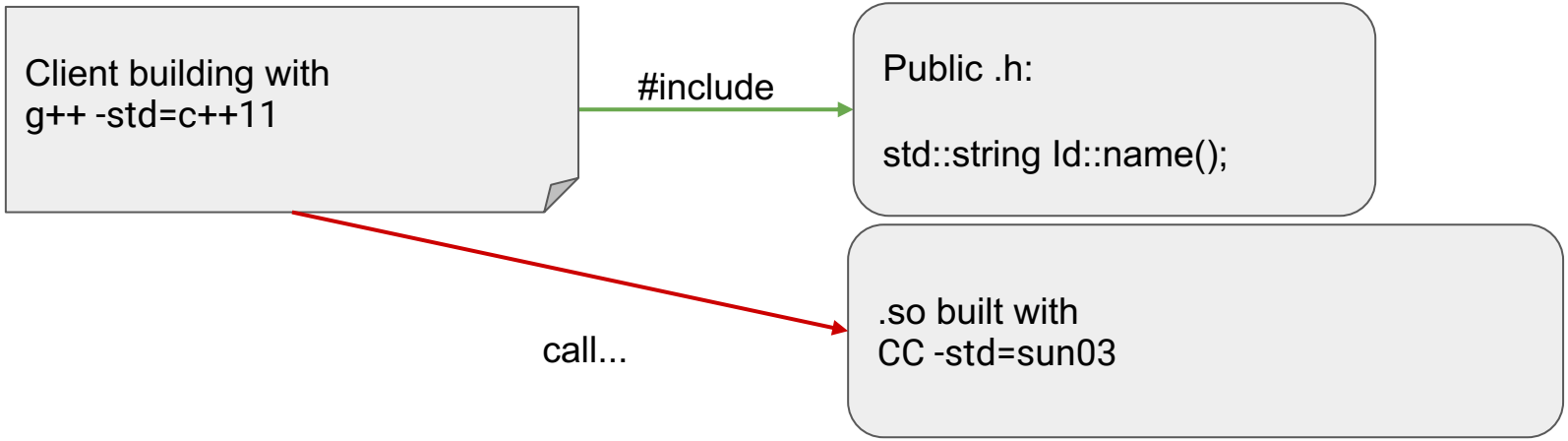
- different compilers
- different versions of the same compiler
- even same compiler with different compilation flags



Bloomberg

Engineering

Example



Bloomberg
Engineering

Breaking Changes

- Add, remove, or re-order member variables in a class
- Change the type of any member variable in a class
- Add an explicit copy constructor or destructor
- Inline a function/method (sneaky!)

Bloomberg

Engineering

Breaking Changes

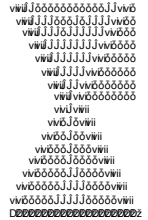
- Change a non-virtual method to be virtual, and vice versa
- Add a virtual method to non-virtual class
- Remove a virtual method
- Change the order of virtual methods.
- Override a virtual that doesn't come from a primary base

Bloomberg

Engineering

Standard and Stable ABI

- C (C89)



Bloomberg

Engineering

Standard and Stable ABI

- C (C89)

```
#ifdef __cplusplus
extern "C" {
#endif
```

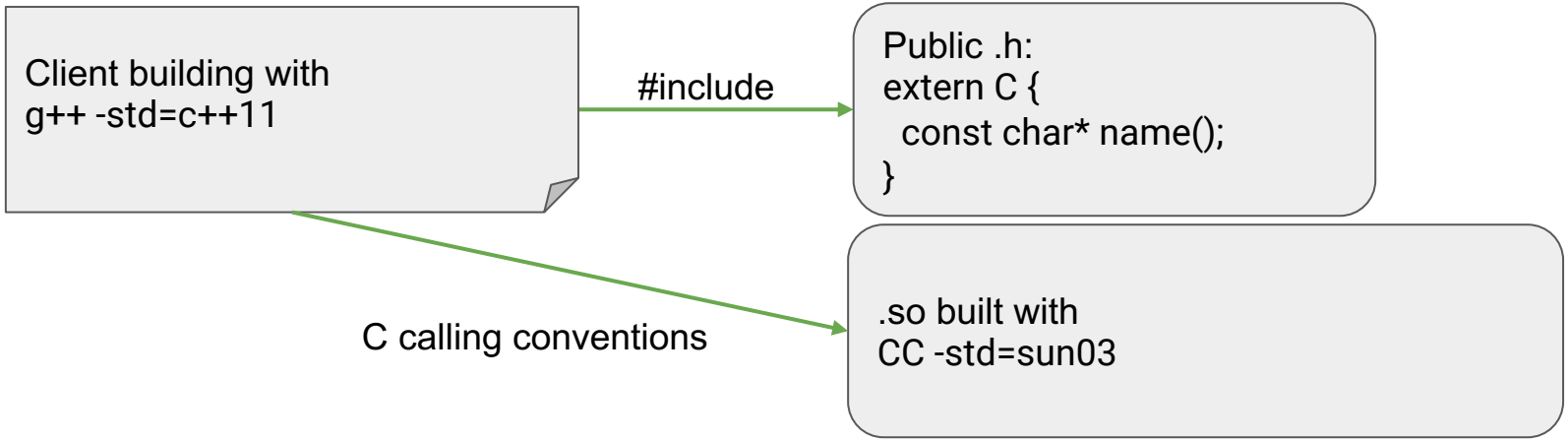
Work here!

```
#ifdef __cplusplus
}
#endif
```

Bloomberg

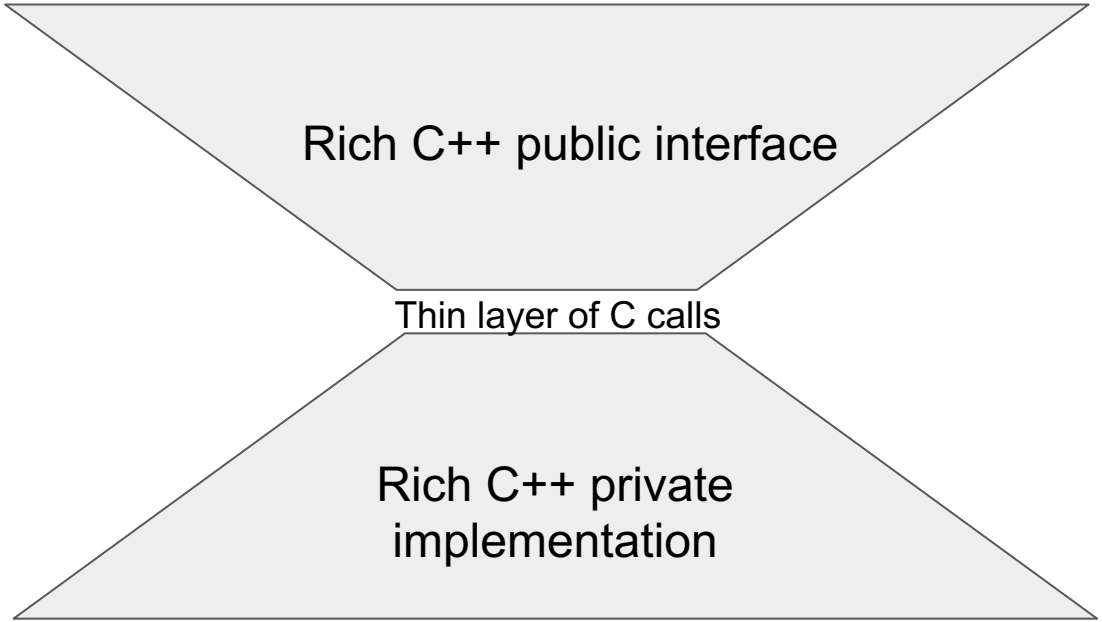
Engineering

Example



Bloomberg
Engineering

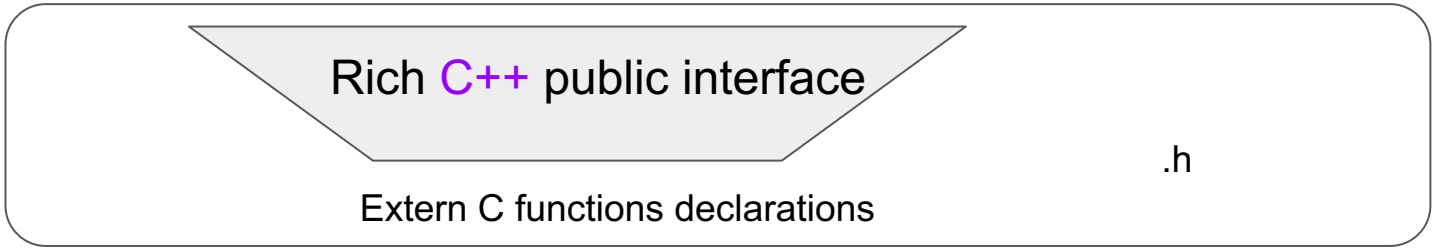
Hourglass

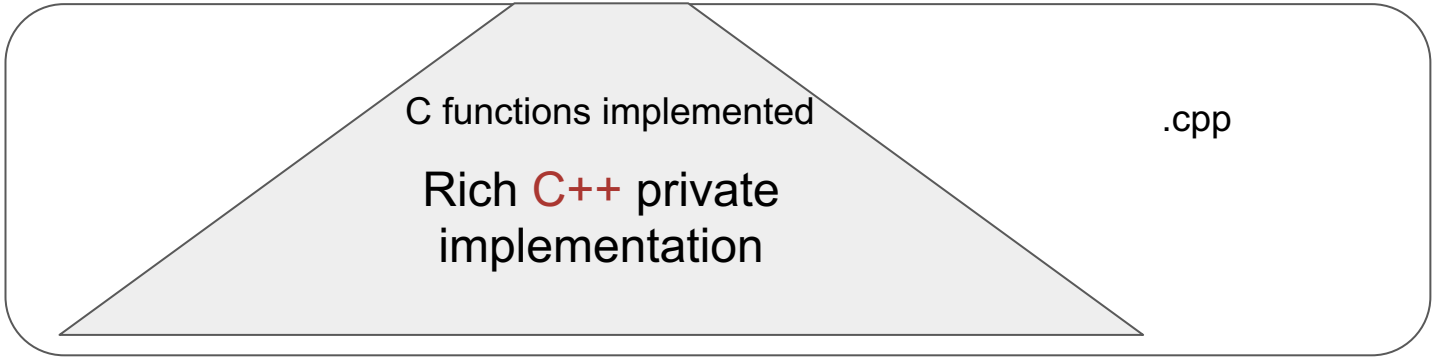


Bloomberg

Engineering

Hourglass

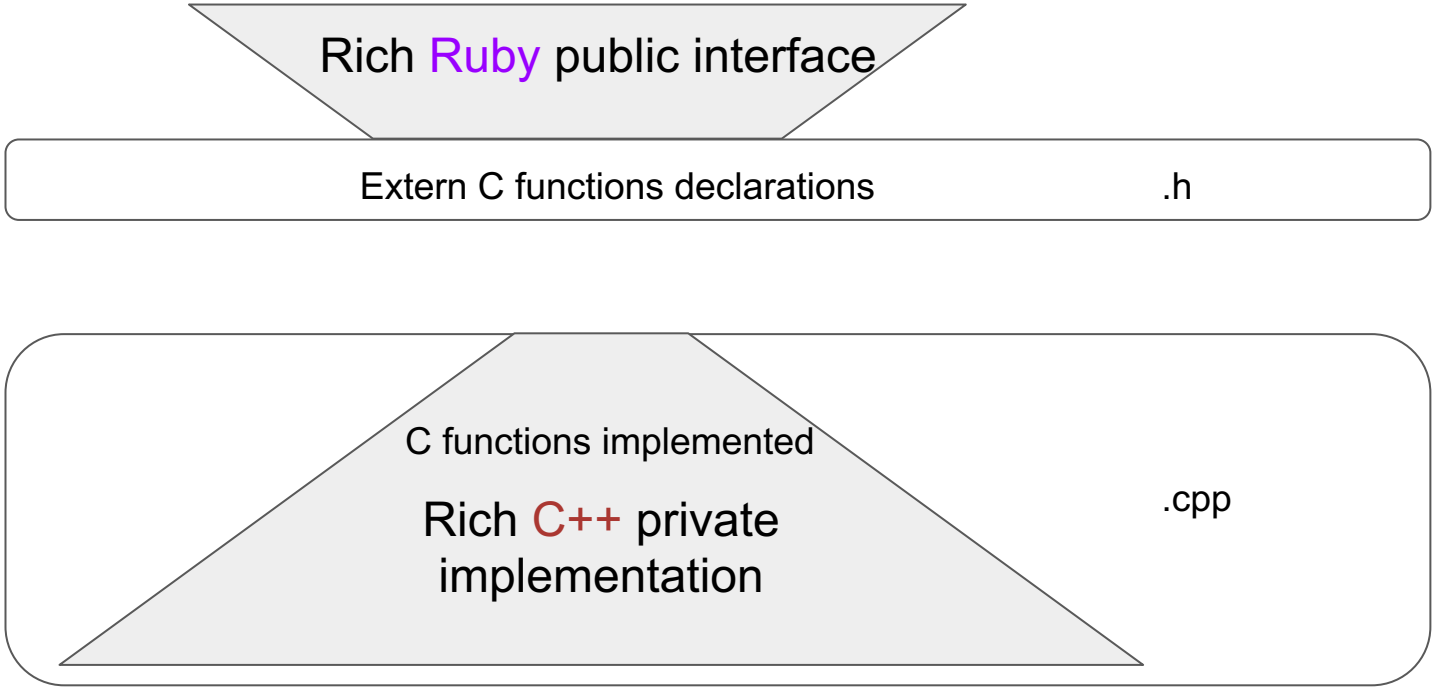




Bloomberg

Engineering

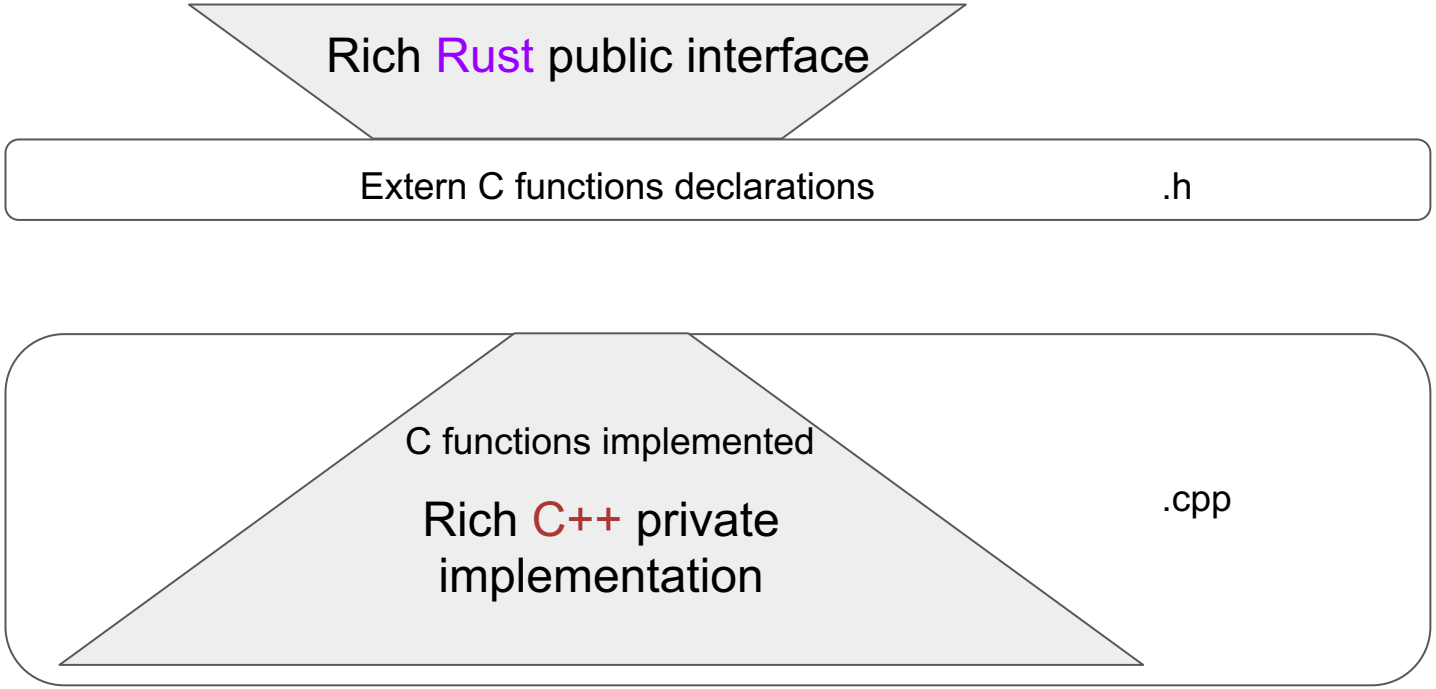
Bonus



Bloomberg

Engineering

Bonus



Bloomberg

Engineering

End-to-end Class example

How do we “hourglass” an existing C++ code?

Bloomberg

Engineering

Example: Profile

```
class ProfileImpl {
private:
   mystring name;
   int age;

public:
   ProfileImpl(mystring n, int a);
};
```

Not public

Bloomberg

Engineering

Opaque Handle

```
#ifdef __cplusplus
extern "C" {
#endif

struct Profile_t;
...
```

Bloomberg

Engineering

C Layer

...

```
int Profile_create(Profile_t **handle,
                   const char* n, int a);

int Profile_copy(const Profile_t *src,
                 Profile_t **dest);

int Profile_destroy(Profile_t *handle);
```

Bloomberg

Engineering

C Layer

```
...
// we allocate
int Profile_create(Profile_t **handle,
                   const char* n, int a);

int Profile_copy(const Profile_t *src,
                 Profile_t **dest);
// you own
int Profile_destroy(Profile_t *handle);
```

out arg

string

return code

why not void*

Bloomberg

Engineering

Public RAII Wrapper

```
class Profile {
private:
   Profile_t* handle;

public:
   Profile(std::string n, int a);
   Profile(const Profile& other);
   ~Profile();
};
// assignment, move...
```

Bloomberg

Engineering

Inline implementation

```
inline Profile::Profile(std::string n, int a){
  if(Profile_create(&handle, n.c_str(), a))
    throw std::runtime_error("Failed @ctor");
}
inline Profile::~Profile() {
  int rc = Profile_destroy(handle);
  assert(0 == rc);
}
inline Profile::Profile(const Profile& other){
  if(Profile_copy(other.handle, &handle))
    throw std::runtime_error("Failed @copy");
}
```

Bloomberg

Engineering

C Layer

```
int Profile_create(Profile_t **handle,
                   const char* n, int a) {
  *handle =
    reinterpret_cast<Profile_t*>(
      new ProfileImpl(mystring(n), a));
  return 0;
}
int Profile_destroy(Profile_t *handle) {
  auto ptr =
    reinterpret_cast<ProfileImpl*>(handle);
  delete ptr;
  return 0;
}
```

Bloomberg

Engineering

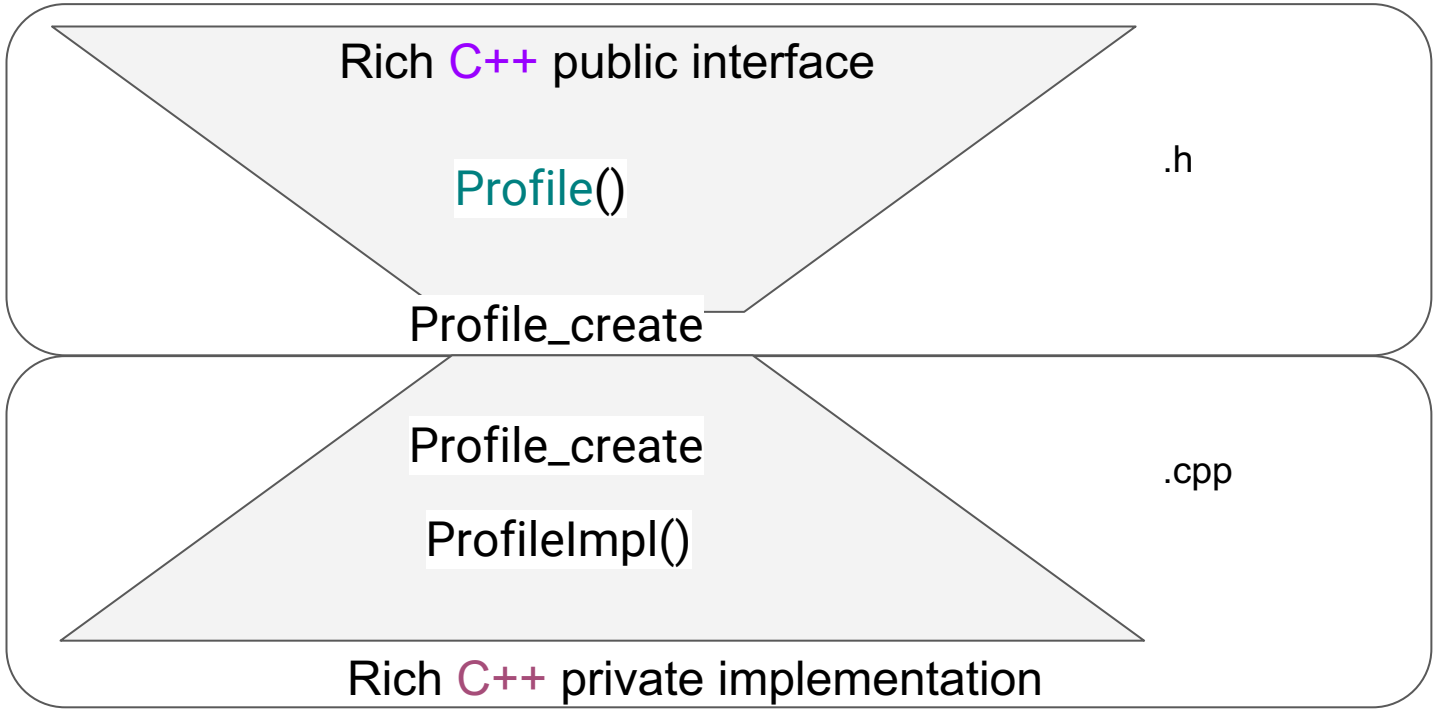
C Layer

```
int Profile_copy(const Profile_t *src,
                 Profile_t **dest) {
  auto ptr = reinterpret_cast<ProfileImpl*>(src);
  *dest = reinterpret_cast<Profile_t*>(
          new ProfileImpl(*ptr));
  return 0;
}
```

Bloomberg

Engineering

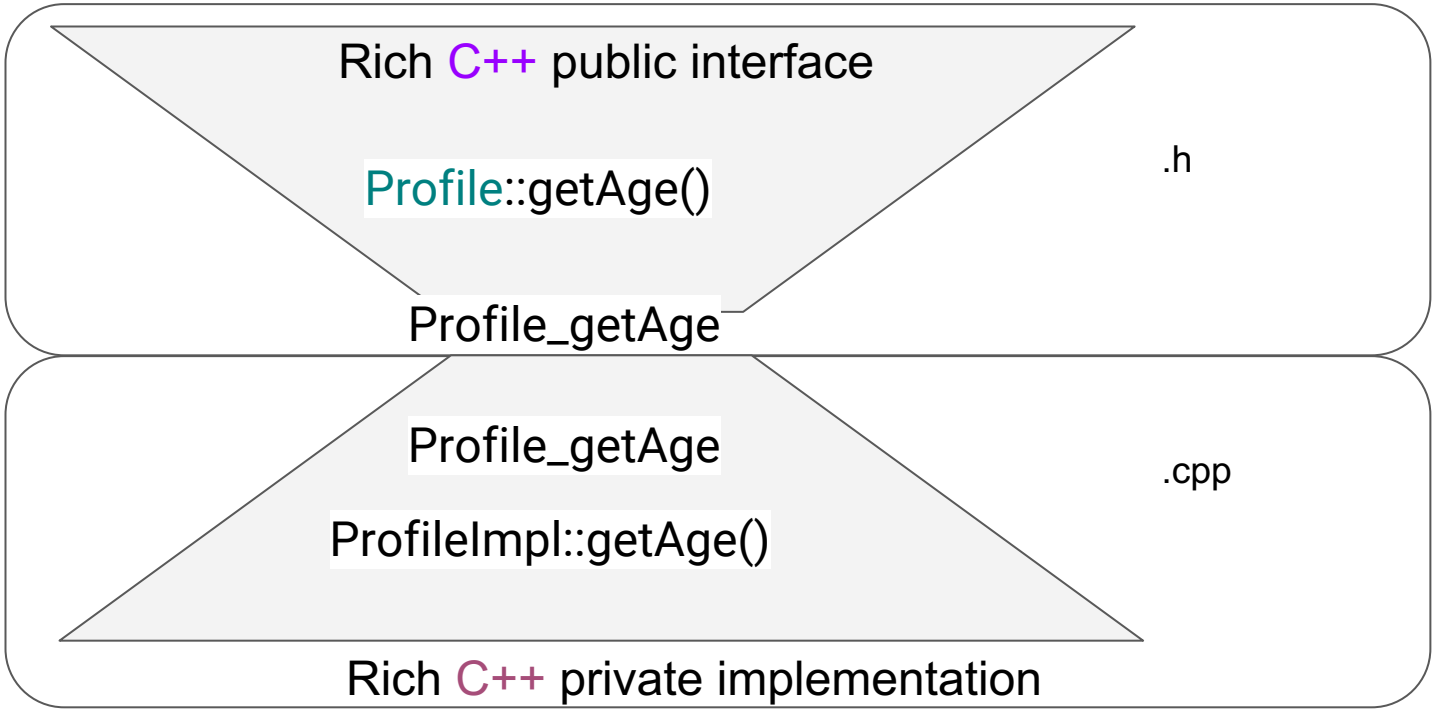
Constructor in Hourglass



Bloomberg

Engineering

Getter in Hourglass



Bloomberg

Engineering

Adding a getter: public

```
int Profile_getAge(const Profile_t *p, int *a);

inline int Profile::getAge() const {
  int age;
  if(Profile_getAge(&handle, &age))
    throw std::runtime_error("Failed @getAge");
  return age;
}
```


Bloomberg

Engineering

Adding a getter: private

```
int ProfileImpl::getAge() const {
  return age;
}
int Profile_getAge(const Profile_t *p, int *a) {
  auto ptr = reinterpret_cast<const ProfileImpl*>(p);
  *a = ptr->getAge();
  return 0;
}
```

Bloomberg

Engineering

Recap: how

- extern “C”
- Opaque handle
- All interactions through C functions
- *PImpl* on steroids

Bloomberg

Engineering

Recap: what

- Hiding C++ code behind a C layer
- Thus providing ultimate binary compatibility
- Wrapping C functions in public C++ classes

Bloomberg

Engineering

Good practices

- Always return an int
- Use wider types for arguments
- Unit test your public wrappers and private impls separately

Bloomberg

Engineering

Good practices

- Be consistent in naming:
  - Foo (public C++)
  - Foo_t (opaque)
  - FooImpl (private C++)
  - Foo_create (C layer)
  - Ideally, “namespace” with mylib_Foo*

- Exceptions can’t go through the hourglass:
  catch them and return as error codes
  (or introduce a RAII wrapper for exceptions)

Bloomberg

Engineering

Breakpoint

- Introduction
  - Problem(s)
  - API compatibility
  - ABI compatibility
  - Hourglass

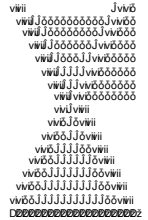
- Detailed discussion
  - Handle wrapper
  - Intrusive types
  - Shared ownership
  - Complex types
  - Type safety
  - Bonus slides?

Bloomberg

Engineering

Intrusive vs. Handle Types

```
struct Id {
  int value1;
  int value2;

  Id() {
    Id_create(&v1, &v1);
  }
  void applyLogic() {
    Id_applyLogic(&v1, &v1);
  }
};
int Id_create(int *v1, int *v2);
int Id_applyLogic(int *v1, int *v2);
```



Bloomberg

Engineering

Types Crossing the Border

- **char, int, float, double**…
- **int***, **void***, Foo_t* (opaque)
- C function pointers

- Strings
  - In: strings as **char***
  - Return: **char*** – alive as long as your object is
  - Alternative: populate a given buffer

Bloomberg

Engineering

C++ Types Crossing the Border

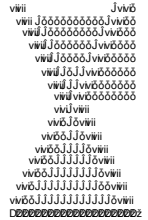
- Shared pointers?
- C++ invocables?
- Vectors or other collections?
- Streams?

Bloomberg

Engineering

Lifetime and Ownership

- Manage lifetime of objects created by private code
- Share objects between layers and threads
- Ensure resources are freed correctly



Bloomberg

Engineering

Shared Pointer from Heap

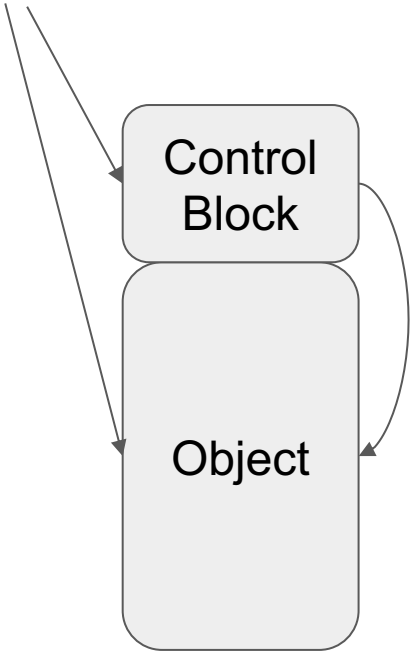
```
int Session_create(Session_t **handle) {
  shared_ptr<SessionImpl>* sptrptr
       = new shared_ptr<SessionImpl>(
            make_shared<SessionImpl>());
   *handle = reinterpret_cast<Session_t*>(sptrptr);
   return 0;
}
int Session_destroy(Session_t *handle) {
   delete reinterpret_cast<shared_ptr<SessionImpl>*>(handle);
   return 0;
}
```

Bloomberg

Engineering

Release for Shared Pointer

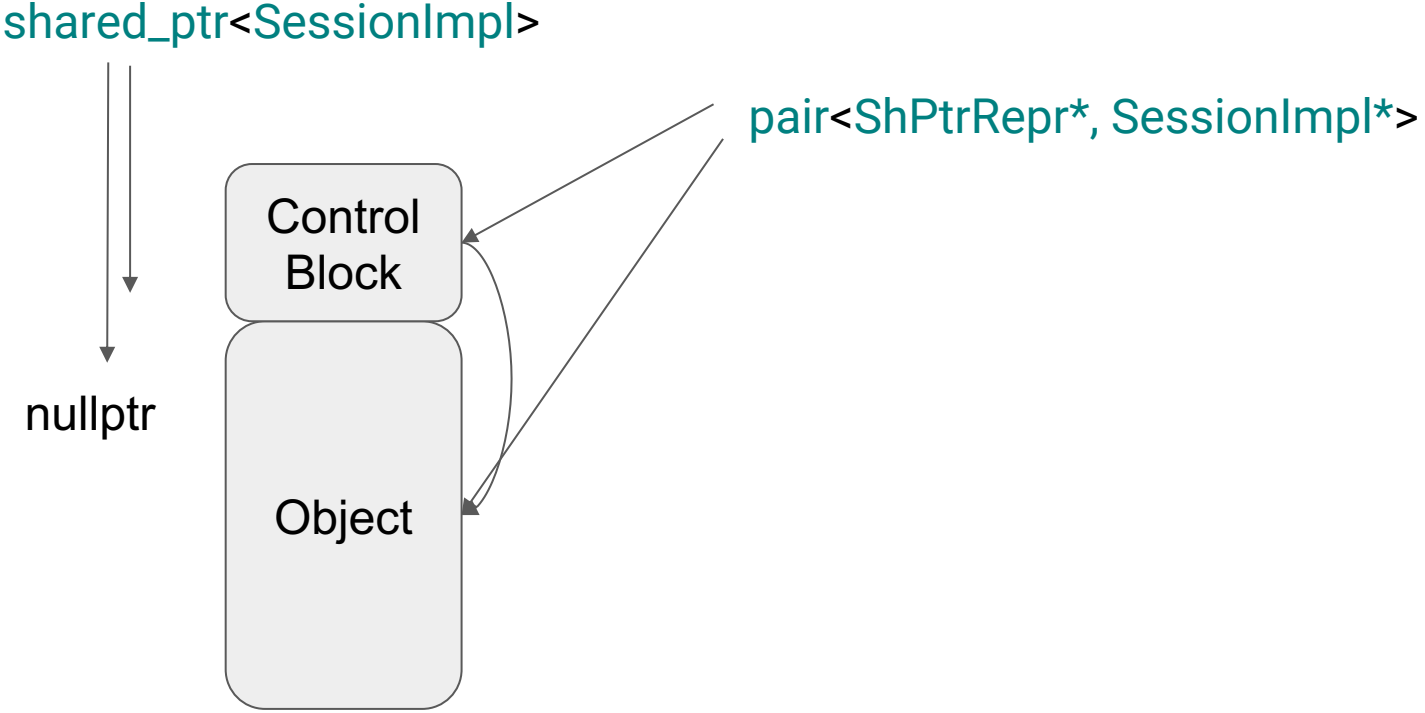
shared_ptr<SessionImpl>



Bloomberg

Engineering

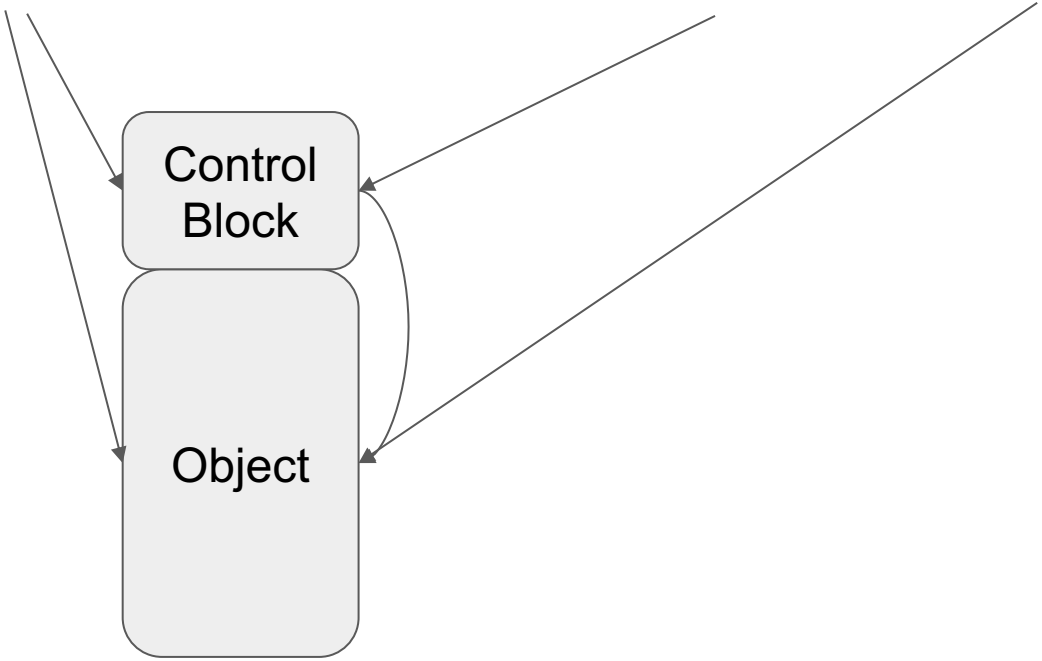
Released for Shared Pointer



Bloomberg
Engineering

Shared Pointer from Repr

shared_ptr<SessionImpl>(ShPtrRepr*, SessionImpl*)



Bloomberg

Engineering

Shared Pointer

```
int Session_create(Session_t **handle) {
  auto sp = make_shared<SessionImpl>();
  *handle = reinterpret_cast<Session_t*>(sp.release().first);
  return 0;
}
int Session_destroy(Session_t *handle) {
   reinterpret_cast<ShPtrRepr*>(handle).decrement();
   return 0;
}
```

Bloomberg

Engineering

Shared Pointer

```
int Session_create(Session_t **handle) {
  auto sp = make_shared<SessionImpl>();
  *handle = reinterpret_cast<Session_t*>(sp.release().first);
  return 0;
}
int Session_destroy(Session_t *handle) {
   reinterpret_cast<ShPtrRepr*>(handle).decrement();
   return 0;
}
```

ShPtrRepr*

Bloomberg

Engineering

C++ Types Crossing the Border

- Shared pointers? (2 options)
- C++ invocables?
- Vectors or other collections?
- Streams?

Bloomberg

Engineering

DCUWCY

C++ entity	User code

```
typedef int (*operate_fn)(opaque*, ...)
extern "C" {
  inline int myOp(opaque *) {...}
  int use(operate_fn, opaque *);
}
```

.h

```
int use(operate_fn op, opaque *v) {
  op(v, …);
}
```

.cpp

Bloomberg

Engineering

Callback helper

```
extern "C" {
  // opaque callback type
  struct Session_cb_t;

  // public side C helper, propagates an int to user callback
  typedef void (*Session_caller)(Session_cb_t*, int);

  // concrete example
  void sessionCall(Session_cb_t* cb, int value) {
    (*reinterpret_cast<
       std::function<void (int)>*>(cb))(value);
  }
```

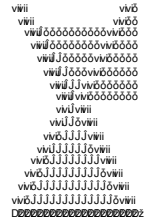
Bloomberg

Engineering

Callback Register

```
// public register
Session::reg(std::function<void (int)> f) {
  this->callback = f;
  Session_reg(this->session,
              &sessionCall,
              reinterpret_cast<Session_cb_t*>)(&this->callback));
}
// register C layer
int Session_reg(Session_t*, Session_caller,Session_cb_t*);
```

```
void SessionImpl::reg(Session_caller c,Session_cb_t* f) {
  this->caller = c;
  this->callback = f;
}
```



Bloomberg

Engineering

Calling back

```
void SessionImpl::onEvent() {
  this->caller(this->callback, 2020);
}
```



```
(*reinterpret_cast<
   std::function<void (int)>*>(cb))(value);
```



```
[](int a){ … }
```

Bloomberg

Engineering

C++ Types Crossing the Border

- Shared pointers? (2 options)
- C++ invocables?
- Vectors or other collections? Similar.
- Streams? See bonus slides.

Bloomberg

Engineering

Type Safety

- No “naked” reintrepret_cast
- Checked casts: **Foo_t * → FooImpl *** and **FooImpl * → Foo_t ***
- Homogenous access for handle types and ref. counted types
  Foo_t * → FooImpl * or **Foo_t * →** shared_ptr<**FooImpl**>*

Bloomberg

Engineering

Type Safety

- No “naked” reinterpret_cast

- Checked casts: **Foo_t *** → **FooImpl *** and **FooImpl *** → **Foo_t ***

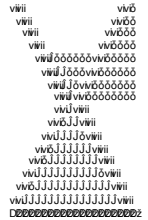
- Homogenous access for handle types and ref. counted types
  Foo_t * → **FooImpl *** or **Foo_t *** → shared_ptr<**FooImpl**>*

Bloomberg

Engineering

Bi-directional Typemaps

```
template<typename Opaque, typename Impl, typename Tag>
struct TypeMeta {
  typedef Opaque opaque;
  typedef Impl impl;
  typedef Tag tag;
};

template<typename Opaque> struct ToImpl;
template<typename Impl>   struct FromImpl;

struct Raw;      // tag for handle types
struct RefCount; // tag for ref. counted types
// a bit longer (e.g., constness traits)
```



Bloomberg

Engineering

Typemap Specialisations

```
template<>
struct ToImpl<Profile_t>
    : public TypeMeta<Profile_t, ProfileImpl, Raw> {
};

template<>
struct FromImpl<ProfileImpl>
    : public TypeMeta<Profile_t, ProfileImpl, Raw> {
};
```

Bloomberg

Engineering

Typemap Specialisations

```
template<>
struct ToImpl<Session_t>
    : public TypeMeta<Session_t, SessionImpl, RefCount> {
};

template<>
struct FromImpl<SessionImpl>
    : public TypeMeta<Session_t, SessionImpl, RefCount> {
};
```

Bloomberg

Engineering

Uniform Access →

```
template<typename Opaque>
typename ToImpl<Opaque>::impl *ptr(Opaque *p, Raw) {
  return
    reinterpret_cast<typename ToImpl<Opaque>::impl *>(p);
}
template<typename Opaque>
typename ToImpl<Opaque>::impl *ptr(Opaque *p, RefCount) {
   ShPtrRep* rep = reinterpret_cast<ShPtrRep*>(p);
   return rep->asType<typename ToImpl<Opaque>::impl>();
}
template<typename Opaque>
typename ToImpl<Opaque>::impl *ptr(Opaque *p) {
   return ptr(p, typename ToImpl<Opaque>::tag());
}
```

Bloomberg

Engineering

Uniform Access ←

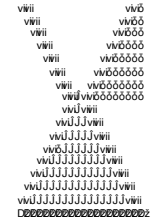
```
template<typename Impl>
typename FromImpl<Impl>::opaque *release(Impl *p) {
  return
    reinterpret_cast<typename FromImpl<Impl>::opaque *>(p);
}

template<typename Impl>
typename FromImpl<Impl>::opaque *release(ShPtr<Impl>& p) {
  return
    reinterpret_cast<typename FromImpl<Impl>::opaque *>(
      p.release().first);
}
```

Bloomberg

Engineering

Type Erasure Dangers

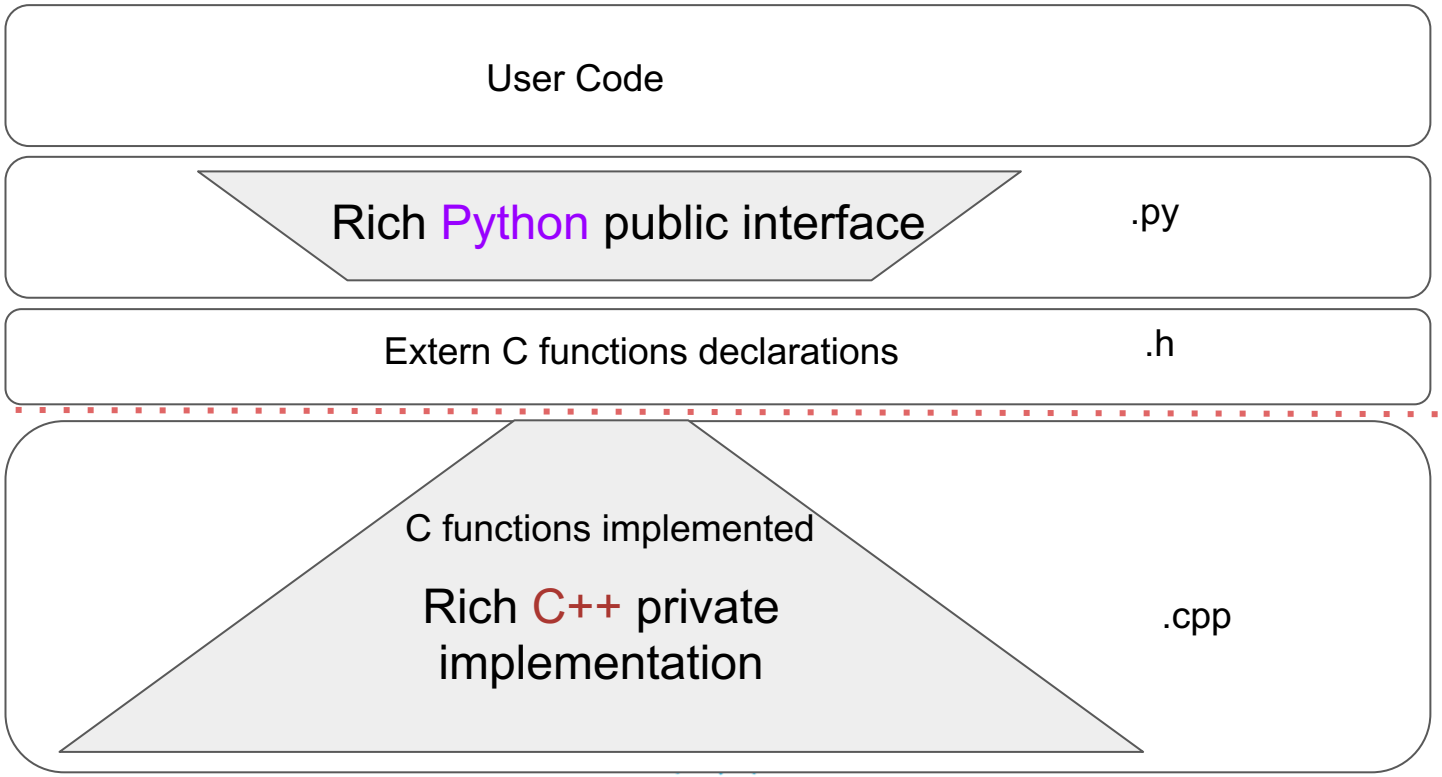
- reusing opaque types for unrelated classes
- reusing opaque types for classes related by inheritance
- using for types with challenging layout (e.g., virtual base)



Bloomberg

Engineering

Take Away



Bloomberg

Engineering

Code examples

https://github.com/nsergey82/hourglass

Bloomberg

Engineering

Bonus Slides

Bloomberg

Engineering

Stream Helper

```
typedef int (*streamcb)(const char *, int, stream_t *);

extern “C” {
  // write to type erased stream provided by opaque pointer
  int streamOut(const char *, int, stream_t*);
}

int streamOut(const char *data, int length, stream_t *stream) {
   reinterpret_cast<Stream*>(stream)->write(data, length);
   return 0;
}
```

Bloomberg

Engineering

Stream Helper

```
int Session_print(
            Session_t *p, streamcb cb, stream_t *stream) {
        auto ptr = reinterpret_cast<SessionImpl>(p);
        mystring msg = session.description();
        cb(msg.name(), msg.size(), stream);
}
```

Bloomberg

Engineering

Dispatch Table

```
typedef struct Dispatch {
  Fptr1,
  Fptr2,
  ...
} Dispatch_t;

extern Dispatch_t g_DispatchEntries;
extern size_t     g_DispatchSize;
```

Bloomberg

Engineering

Dispatch Table

```
typedef struct Dispatch {
  void (*f1)(int),
  int  (*f1_fix)(int),
  int  (*f2)(double) } Dispatch_t;

void init() {
  g_DispatchEntries.f1 = &f1;
  g_DispatchEntries.f2 = &f2;
  …
}
```

Bloomberg

Engineering

Dispatch Table

```
#define CALL(f) \
 offsetof(g_DispatchEntries, f) < g_DispatchSize ? \
 g_DispatchEntries.f : \
 unsupported_stub;

CALL(f1)(2);
```

Bloomberg

Engineering

This slide is empty