CIB

Component **I**nterface **B**inder:

ABI stable architecture for a C++ SDK

Satya Das

About me

- A developer at TomTom
- Previously I was at Adobe

ABI Stability

Binary compatibility of older client with new library

Agenda

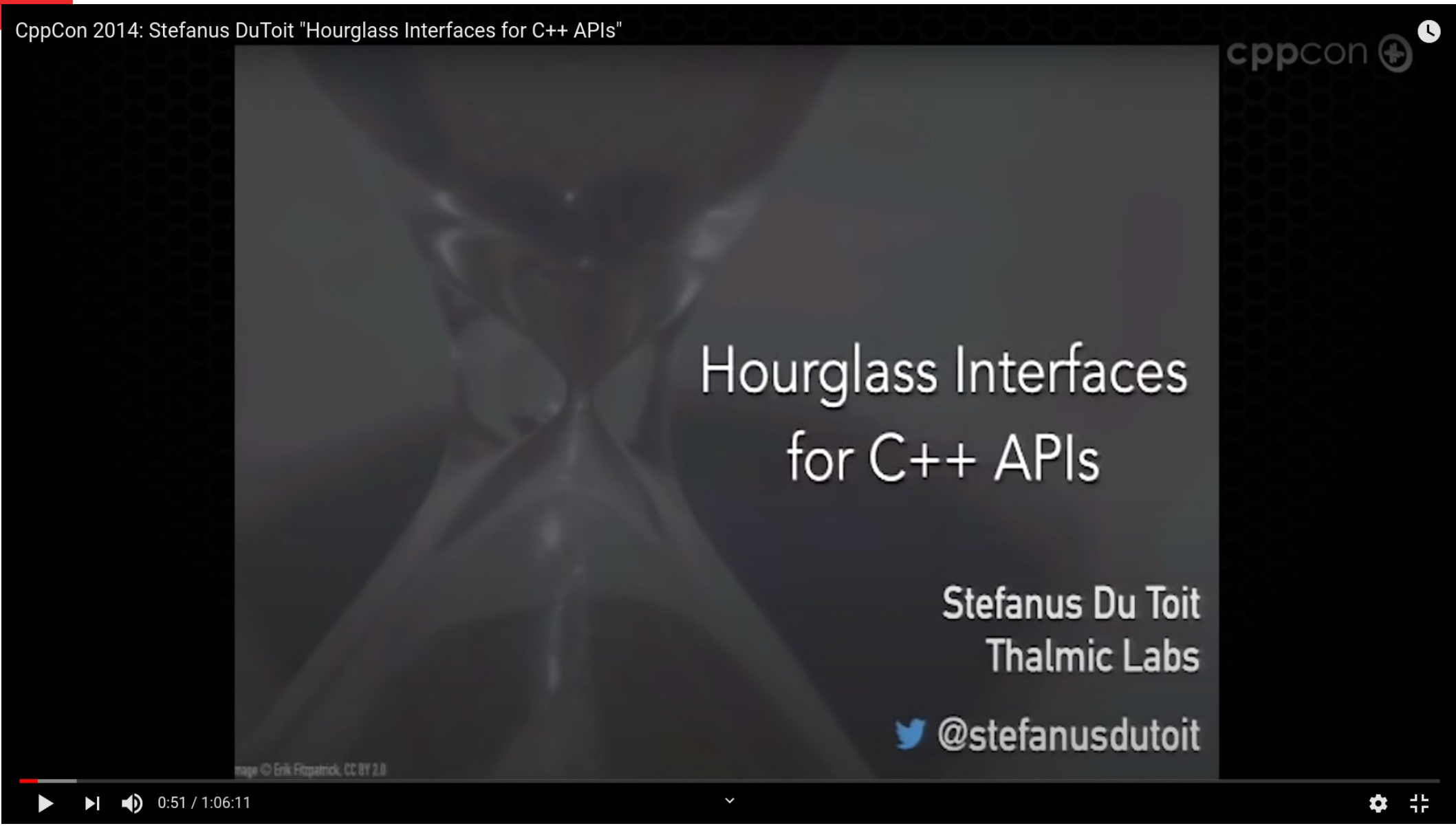
- ABI stable architecture for
  - Simple Class with demo
  - Class Hierarchy
  - Interface Class with demo
- About CIB tool
- Case Study
- What all we left

Simple class Demo

SIMPLE CLASS

```
class Circle {
public:
  Circle(float r);
  Circle(const Circle&) = default;
  virtual ~Circle() = default;

  float Radius() const;
  void SetRadius(float r);

  virtual float Area() const;

private:
  float mRadius;
};
```

Hourglass Interface



Hourglass Interface

- It was a great talk but few things it did not cover
  - Virtual function
  - Interface classes
  - Inheritance

- As per Stefanus it was more for ABI isolation

Export free functions ?

```
Circle* CreateCircle(float r) {
  return new Circle(r);
}

Circle* CopyCircle(const Circle* pCircle) {
  return new Circle(*pCircle);
}

void DeleteCircle(Circle* pCircle) {
  delete pCircle;
}
```

```
float Radius(const Circle* pCircle) {
  return pCircle->Radius();
}

void SetRadius(Circle* pCircle, float r) {
  pCircle->SetRadius(r);
}

float Area(const Circle* pCircle) {
  return pCircle->Area();
}
```

Except the constructor all functions have first parameter as pointer of the class.

Export MethodTable instead

```
class Circle;
using CircleImpl = Circle;

extern "C" struct MethodTableCircle {
  const size_t numMethods;

  CircleImpl* (*Create)     (float);
  CircleImpl* (*Copy)       (const CircleImpl*);
  void        (*Delete)     (CircleImpl*);
  float       (*Radius)     (const CircleImpl*);
  void        (*SetRadius)  (CircleImpl*, float);
  float       (*Area)       (const CircleImpl*);
};
```

Export MethodTable instead

```
class Circle;
using CircleImpl = Circle;

extern "C" struct MethodTableCircle {
  const size_t numMethods;

  CircleImpl* (*Create)     (float);
  CircleImpl* (*Copy)       (const CircleImpl*);
  void        (*Delete)     (CircleImpl*);
  float       (*Radius)     (const CircleImpl*);
  void        (*SetRadius)  (CircleImpl*, float);
  float       (*Area)       (const CircleImpl*);
};
```

Export MethodTable instead

```
class Circle;
using CircleImpl = Circle;

extern "C" struct MethodTableCircle {
  const size_t numMethods;

  CircleImpl* (*Create)     (float);
  CircleImpl* (*Copy)       (const CircleImpl*);
  void        (*Delete)     (CircleImpl*);
  float       (*Radius)     (const CircleImpl*);
  void        (*SetRadius)  (CircleImpl*, float);
  float       (*Area)       (const CircleImpl*);
};
```

```
extern "C" {
  MethodTableCircle DLLEXPORT
    gMethodTableCircle = {
    6,
    &CreateCircle,
    &CopyCircle,
    &DeleteCircle,
    &Radius,
    &SetRadius,
    &Area
  };
}
```

Client Uses Proxy Class

Library side class definition

```
class Circle {
public:
  Circle(float r);
  Circle(const Circle&) = default;
  virtual ~Circle() = default;

  float Radius() const;
  void SetRadius(float r);

  virtual float Area() const;

private:
  float mRadius;
};
```

Client side class definition

```
class CircleImpl;

class Circle {
public:
  Circle(float r);
  Circle(const Circle&);
  virtual ~Circle();

  float Radius() const;
  void SetRadius(float r);

  virtual float Area() const;

private:
  Circle(CircleImpl* pCircleImpl)
    : pImpl(pCircleImpl) {}
  CircleImpl* pImpl;
};
```

Client Imports MethodTable

```
class CircleImpl;


extern "C" struct MethodTableCircle {
  const size_t numMethods;

  CircleImpl* (*Create)     (float);
  CircleImpl* (*Copy)       (const CircleImpl*);
  void        (*Delete)     (CircleImpl*);
  float       (*Radius)     (const CircleImpl*);
  void        (*SetRadius)  (CircleImpl*, float);
  float       (*Area)       (const CircleImpl*);
};


extern "C" MethodTableCircle DLLIMPORT gMethodTableCircle;
```

Proxy Class Implementation

```
Circle::Circle(float r)
  : pImpl(gMethodTableCircle.Create(r))
{}

Circle::Circle(const Circle& a)
  : pImpl(gMethodTableCircle.Copy(a.pImpl))
{}

Circle::~Circle() {
  gMethodTableCircle.Delete(pImpl);
}
```

Proxy Class Implementation

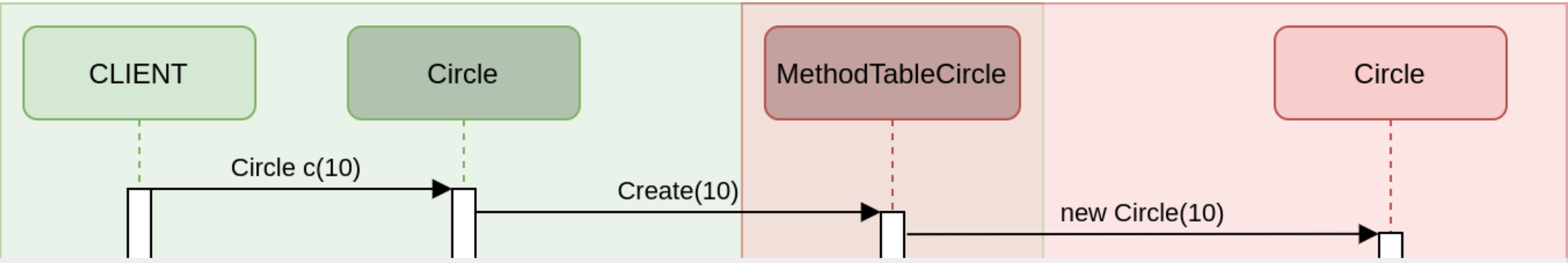
```
float Circle::Radius() const {
  return gMethodTableCircle.Radius(pImpl);
}

void Circle::SetRadius(float r) {
  return gMethodTableCircle.SetRadius(pImpl, r);
}

float Circle::Area() const {
  return gMethodTableCircle.Area(pImpl);
}
```

Proxy Class Sequence Diagram

```
Circle c(10);
c.Area();
```

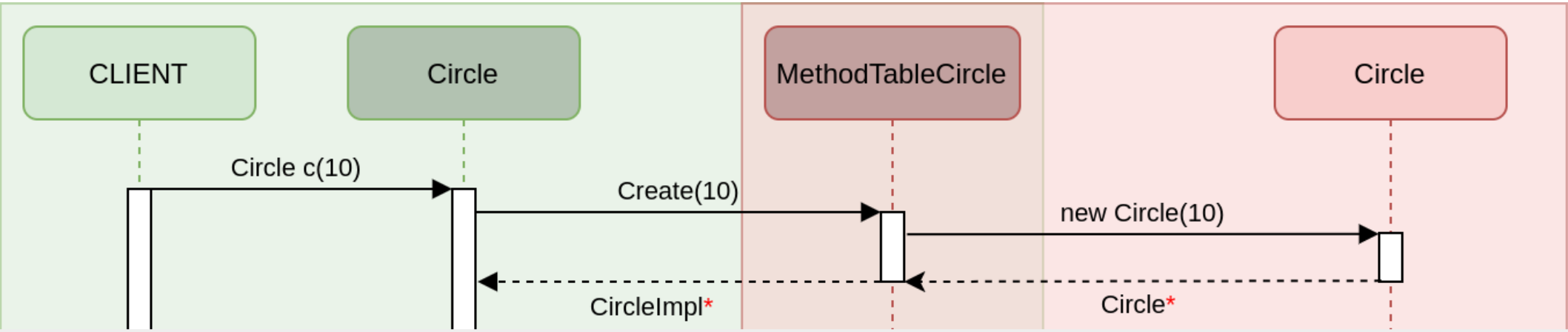
Class use sequence diagram

```
Circle c(10);
c.Area();
```



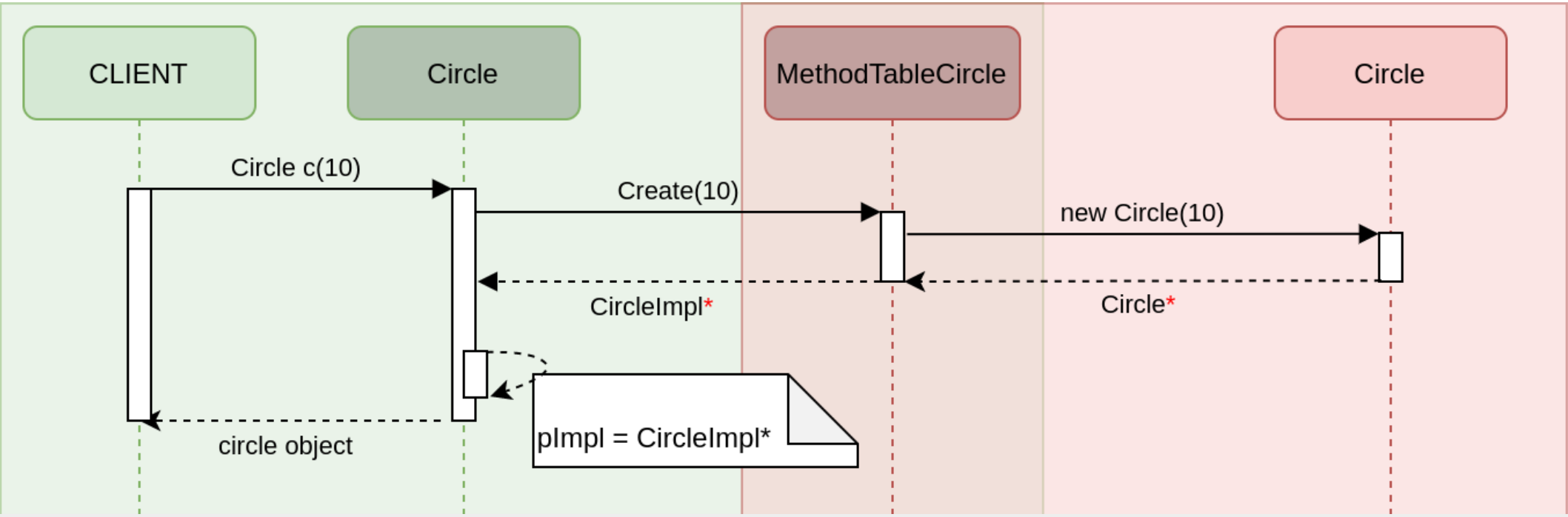
Class use sequence diagram

```
Circle c(10);
c.Area();
```



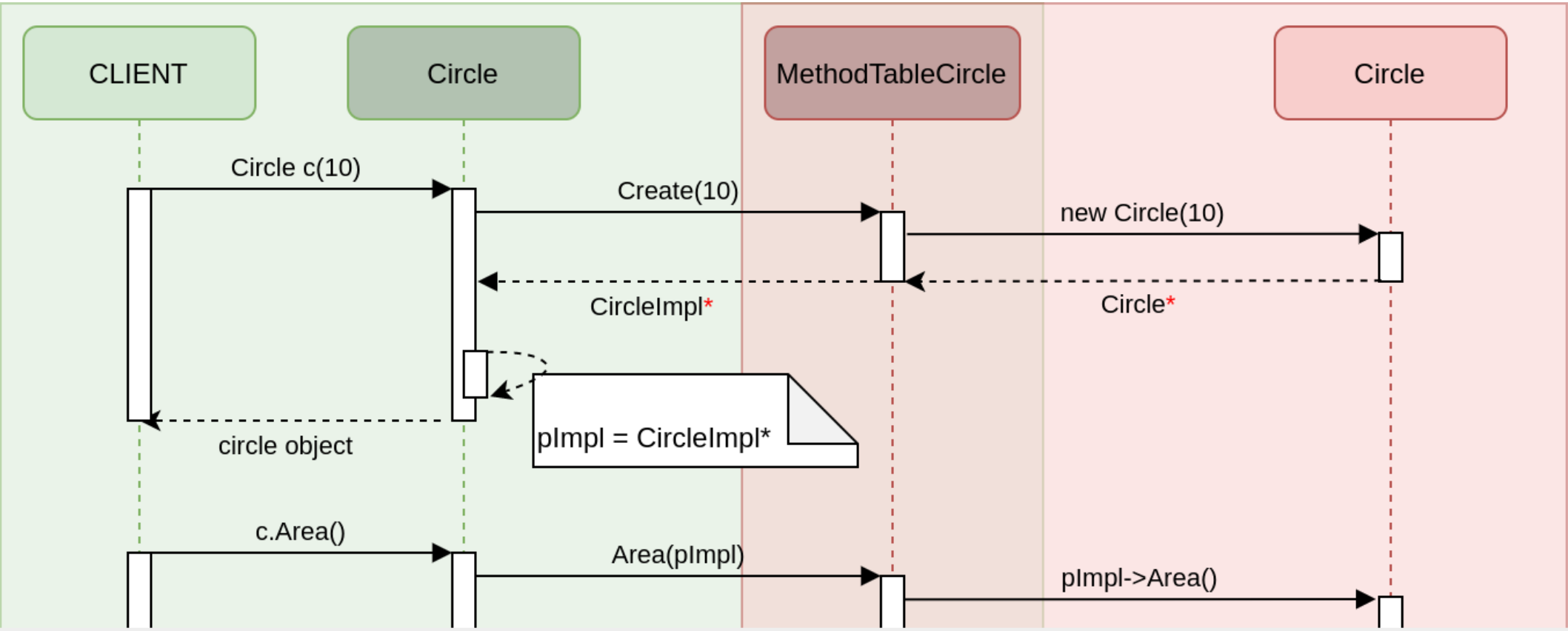
Class use sequence diagram

```
Circle c(10);
c.Area();
```



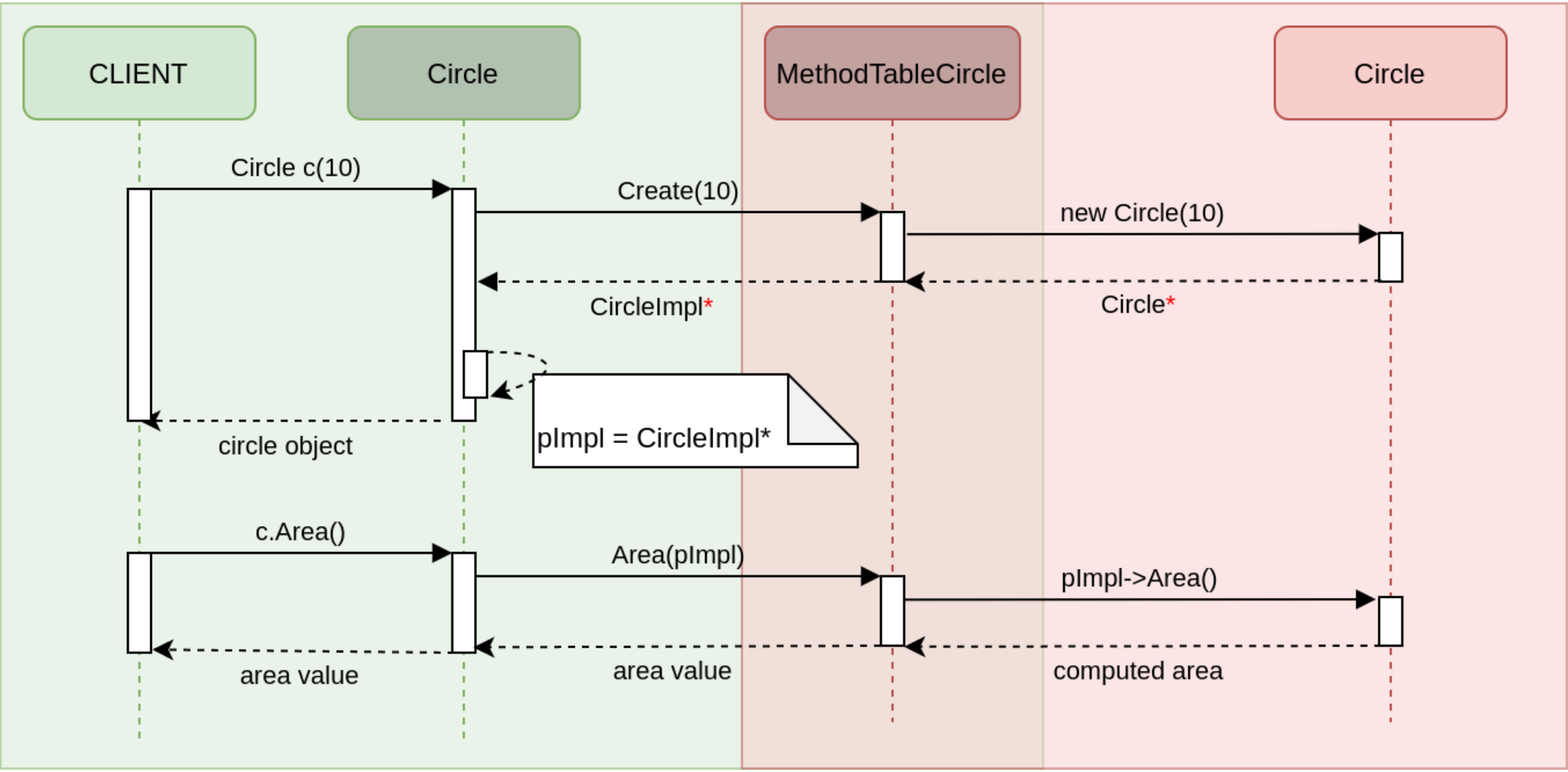
Class use sequence diagram

```
Circle c(10);
c.Area();
```



Class use sequence diagram

```
Circle c(10);
c.Area();
```



Ensuring ABI stability

```
class Circle {
public:
  Circle(float r);
  Circle(const Circle&) = default;
  virtual ~Circle() = default;

  float Radius() const;
  void SetRadius(float r);

  virtual float Perimeter() const;
  virtual float Area() const;

private:
  float mOx {0};
  float mOy {0};
  float mRadius;
};
```

Ensuring ABI stability

MethodTable is only **appended**

Method Table: Version 1

```
extern "C" {
  MethodTableCircle DLLEXPORT
    gMethodTableCircle = {
     6,
     &CreateCircle,
     &CopyCircle,
     &DeleteCircle,
     &Radius,
     &SetRadius,
     &Area
   };
}
```

Method Table: Version 2

```
extern "C" {
  MethodTableCircle DLLEXPORT
    gMethodTableCircle = {
     7,
     &CreateCircle,
     &CopyCircle,
     &DeleteCircle,
     &Radius,
     &SetRadius,
     &Area,
     &Perimeter
   };
}
```

MethodTable Vs Virtual Table

- MethodTable is NOT replacement of virtual table.

MethodTable Vs Virtual Table

- MethodTable is NOT replacement of virtual table.
- MethodTable is just a mechanism for cross component function call.

MethodTable Vs Virtual Table

- MethodTable is NOT replacement of virtual table.
- MethodTable is just a mechanism for cross component function call.
- MethodTable is shared between components.

MethodTable Vs Virtual Table

- MethodTable is NOT replacement of virtual table.
- MethodTable is just a mechanism for cross component function call.
- MethodTable is shared between components.
- Virtual table remains local.

MethodTable Vs Virtual Table

- MethodTable is NOT replacement of virtual table.
- MethodTable is just a mechanism for cross component function call.
- MethodTable is shared between components.
- Virtual table remains local.
- MethodTable is handy in supporting backward compatibility.

Client's Backward Compatiblity

```
class CircleImpl;

class Circle {
public:
  Circle(float r);
  Circle(const Circle&);
  virtual ~Circle();

  float Radius() const;
  void SetRadius(float r);

  virtual float Perimeter() const;
  virtual float Area() const;

private:
  Circle(CircleImpl* pImplA);
  CircleImpl* pImpl;
};
```

Client’s Backward Compatiblity

```
float Circle::Perimeter() const {
  if (gMethodTableCircle.numMethods < 7)
    throw std::bad_function_call();
  return gMethodTableCircle.Perimeter(pImpl);
}
```

Client’s Backward Compatiblity

```
  try {

    Circle c;
    const auto p = c.Perimeter();

  } catch(std::bad_function_call) {

    std::clog << "New client with old library" << std::endl;

  }
}
```

MethodTable Vs Virtual Table

- MethodTable is NOT replacement of virtual table.
- MethodTable is just a mechanism for cross component function call.
- MethodTable is shared between components.
- Virtual table remains local.
- MethodTable is handy in supporting backward compatibility.
- MethodTable does NOT inherit.

EXAMPLE 2

INHERITANCE

Inheritance

Library Version 1

```
struct Base {
  Base() {}
  virtual ~Base() {}

  virtual int F() { return 1; }
  virtual int G() { return 2; }
};
```

```
struct Derived : Base {
  Derived() {}
  virtual ~Derived() {};

  int G() override { return 3; }
  virtual int H()  { return 4; }
};
```

Inheritance

Library Version 1

```
struct Base {
  Base() {}
  virtual ~Base() {}

  virtual int F() { return 1; }
  virtual int G() { return 2; }
};



struct Derived : Base {
  Derived() {}
  virtual ~Derived() {};

  int G() override { return 3; }
  virtual int H()  { return 4; }
};
```

Library Version 2

```
struct Base {
  Base() {}
  virtual ~Base() {}

  virtual int F() { return 1; }
  virtual int G() { return 2; }
  virtual int E() { return 9; }
};



struct Derived : Base {
  Derived() {}
  virtual ~Derived() {};

  int G() override { return 3; }
  virtual int H()  { return 4; }
};
```

Inheritance

Library Version 1

```
struct Base {
  Base() {}
  virtual ~Base() {}

  virtual int F() { return 1; }
  virtual int G() { return 2; }
};



struct Derived : Base {
  Derived() {}
  virtual ~Derived() {};

  int G() override { return 3; }
  virtual int H()  { return 4; }
};
```

Library Version 2

```
struct Base {
  Base() {}
  virtual ~Base() {}

  virtual int F() { return 1; }
  virtual int G() { return 2; }
};


struct Base2 {...};

struct Derived : Base, Base2 {
  Derived() {}
  virtual ~Derived() {};

  int G() override { return 3; }
  virtual int H()  { return 4; }
};
```

Inheritance

MethodTable of Base Class

```
static Base* CreateBase() {
  return new Base;
}

static void DeleteBase(Base* pBase)
{
  delete pBase;
}

static int F(Base* pBase) {
  return pBase->F();
}

static int G(Base* pBase) {
  return pBase->Base::G();
}
```

```
extern "C" {
  MethodTableBase DLLEXPORT
    gMethodTableBase = {
    4,
    &CreateBase,
    &DeleteBase,
    &F,
    &G
  };
}
```

Inheritance

MethodTable of Derived Class does NOT contain entries for Base class

```
static Derived* CreateDerived() {
  return new Derived;
}

static void DeleteDerived(Derived* pDerived)
{
  delete pDerived;
}

static int G(Derived* pDerived) {
  return pDerived->G();
}

static int H(Derived* pDerived) {
  return pDerived->H();
}

static Base* CastToBase(Derived* pDerived) {
  return pDerived;
}
```

```
extern "C" {
  MethodTableDerived DLLEXPORT
    gMethodTableDerived = {
    5,
    &CreateDerived,
    &DeleteDerived,
    &CastToBase,
    &G,
    &H
  };
}
```

Inheritance

MethodTable of Derived Class does NOT contain entries for Base class

```
static Derived* CreateDerived() {
  return new Derived;
}

static void DeleteDerived(Derived* pDerived)
{
  delete pDerived;
}

static int G(Derived* pDerived) {
  return pDerived->G();
}

static int H(Derived* pDerived) {
  return pDerived->H();
}

static Base* CastToBase(Derived* pDerived) {
  return pDerived;
}
```

```
extern "C" {
  MethodTableDerived DLLEXPORT
    gMethodTableDerived = {
    5,
    &CreateDerived,
    &DeleteDerived,
    &CastToBase,
    &G,
    &H
  };
}
```

- No function pointer corresponding to base class methods

Inheritance

Base Class definition on client side

```
struct BaseImpl;

struct Base {
  Base();
  virtual ~Base();

  virtual int F();
  virtual int G();

protected:
  Base(BaseImpl*
       pBaseImpl);
  BaseImpl* pImpl;
};
```

```
extern "C" MethodTableBase DLLIMPORT
gMethodTableBase;

Base::Base(BaseImpl* pBaseImpl)
  : pImpl(pBaseImpl)
{}

Base::Base()
  : Base(gMethodTableBase.Create())
{}

Base::~Base() {
  gMethodTableBase.Delete(pImpl);
}

int Base::F() {
  return gMethodTableBase.F(pImpl);
}

int Base::G() {
  return gMethodTableBase.G(pImpl);
}
```

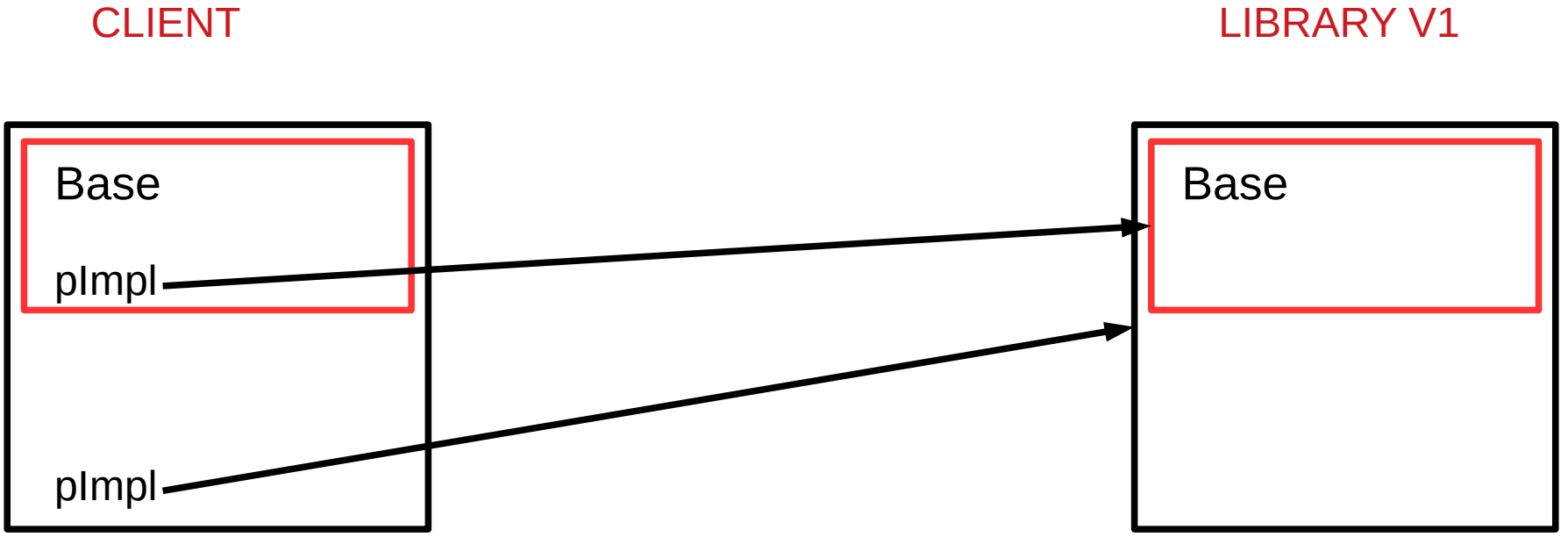
Inheritance

Derived Class definition on client side

```
struct DerivedImpl;

struct Derived : Base {
  Derived();
  virtual ~Derived();

  int G() override;
  virtual int H();

private:
  Derived(DerivedImpl*
          pDerivedImpl);
  DerivedImpl* pImpl;
};
```

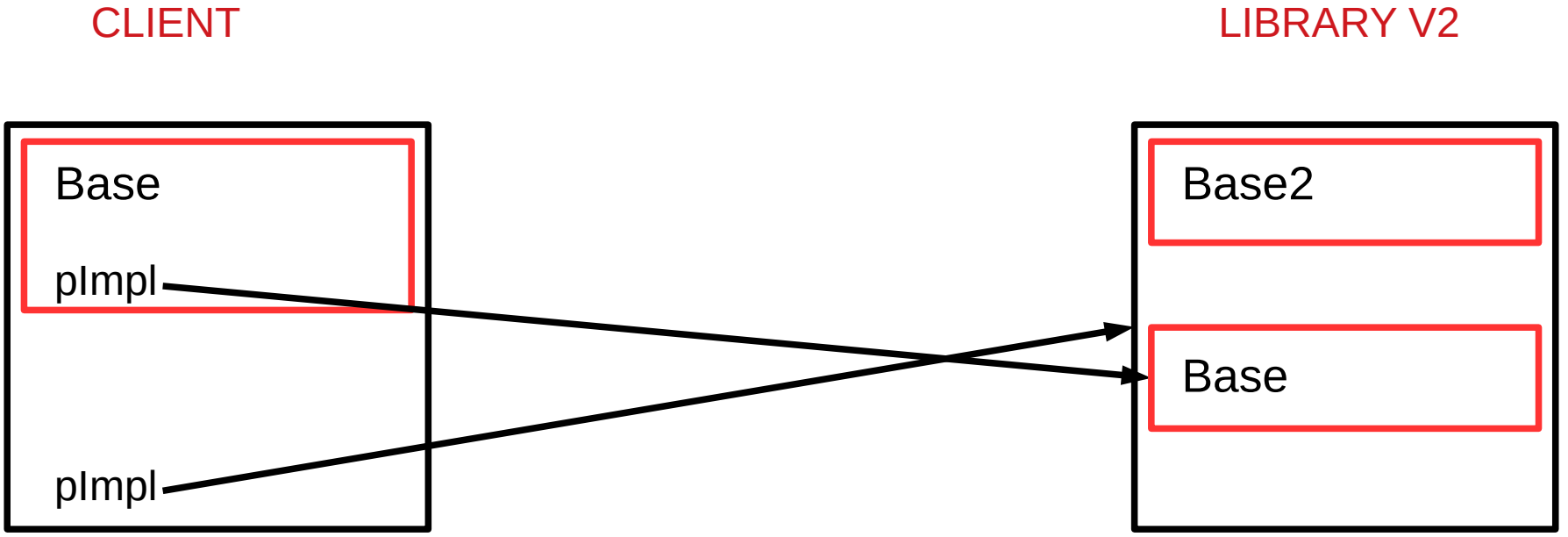
```
extern "C" MethodTableDerived DLLIMPORT
gMethodTableDerived;

Derived::Derived(DerivedImpl* pImplDerived)
  : Base(gMethodTableDerived.CastToBase(pImplDerived))
  , pImpl(pImplDerived)
{}

Derived::Derived()
  : Derived(gMethodTableDerived.Create())
{}

Derived::~Derived() {
  gMethodTableDerived.Delete(pImpl);
  Base::pImpl = nullptr;
}

int Derived::G() {
  return gMethodTableDerived.G(pImpl);
}

int Derived::H() {
  return gMethodTableDerived.H(pImpl);
}
```

Inheritance




```
Derived::Derived(DerivedImpl* pImplDerived)
  : Base(gMethodTableDerived.CastToBase(pImplDerived))
  , pImpl(pImplDerived)
{}
```

Inheritance




```
Derived::Derived(DerivedImpl* pImplDerived)
  : Base(gMethodTableDerived.CastToBase(pImplDerived))
  , pImpl(pImplDerived)
{}
```

EXAMPLE 3

INTERFACE CLASS

Interface Class Demo

Interface Classes

Library

```
class Context {
public:
  virtual ~Context() {}

  virtual void DrawCircle(float r) = 0;
};
```

```
class Circle {
public:
  Circle(float r) : mRadius(r) {}

  void Draw(Context* pCtx) const {
    pCtx->DrawCircle(mRadius);
  }

private:
  float mRadius;
};
```

Client

```
class DrawLogger : public Context {
  void DrawCircle(float r) override {
    std::cout << "DrawCircle(" << r << ")";
  }
};

int main() {
  DrawLogger logger;
  Circle c(5.0);
  c.Draw(&logger);
}
```

Client “exports” method table

```
class Context;
using ClientCtxImpl = Context;

struct MethodTableClientCtx {
  const size_t numMethods;

  void (*DrawCircle) (ClientCtxImpl*, float);
};
```

Client “exports” method table

```
class Context;
using ClientCtxImpl = Context;

struct MethodTableClientCtx {
  const size_t numMethods;

  void (*DrawCircle) (ClientCtxImpl*, float);
};
```

```
static void DrawCircle(Context* pCtx, float r) {
  return pCtx->DrawCircle(r);
}
```

Client “exports” method table

```
class Context;
using ClientCtxImpl = Context;

struct MethodTableClientCtx {
  const size_t numMethods;

  void (*DrawCircle) (ClientCtxImpl*, float);
};



static void DrawCircle(Context* pCtx, float r) {
  return pCtx->DrawCircle(r);
}

static MethodTableClientCtx gMethodTableCtx = { 1, &DrawCircle };
```

Library uses implementaion through proxy

Library glue code

```
class ClientCtxImpl;

class GenericCtx : public Context {
public:
```

```
};
```

Library uses implementaion through proxy

Library glue code

```
class ClientCtxImpl;

class GenericCtx : public Context {
public:
```

```
private:
  ClientCtxImpl*         pImpl;
  MethodTableClientCtx* pMethodTable;
};
```

Library uses implementaion through proxy

Library glue code

```
class ClientCtxImpl;

class GenericCtx : public Context {
public:
  GenericCtx(ClientCtxImpl* p,
             MethodTableClientCtx* m)
    : pImpl(p)
    , pMethodTable(m)
  {}
```

```
private:
  ClientCtxImpl*         pImpl;
  MethodTableClientCtx* pMethodTable;
};
```

Library uses implementaion through proxy

Library glue code

```
class ClientCtxImpl;

class GenericCtx : public Context {
public:
  GenericCtx(ClientCtxImpl* p,
             MethodTableClientCtx* m)
    : pImpl(p)
    , pMethodTable(m)
  {}

  void DrawCircle(float r) override {
    pMethodTable->DrawCircle(pImpl, r);
  }

private:
  ClientCtxImpl*         pImpl;
  MethodTableClientCtx* pMethodTable;
};
```

Library uses implementaion through proxy

Library glue code

```
class ClientCtxImpl;

class GenericCtx : public Context {
public:
  GenericCtx(ClientCtxImpl* p,
             MethodTableClientCtx* m)
    : pImpl(p)
    , pMethodTable(m)
  {}

  void DrawCircle(float r) override {
    pMethodTable->DrawCircle(pImpl, r);
  }

private:
  ClientCtxImpl*        pImpl;
  MethodTableClientCtx* pMethodTable;
};
```

```
static Context* CreateGenericCtx(
    ClientCtxImpl* p,
    MethodTableClientCtx* m) {
  return new GenericCtx(p, m);
}
```

Library uses implementaion through proxy

Library glue code

```
class ClientCtxImpl;

class GenericCtx : public Context {
public:
  GenericCtx(ClientCtxImpl* p,
             MethodTableClientCtx* m)
    : pImpl(p)
    , pMethodTable(m)
  {}

  void DrawCircle(float r) override {
    pMethodTable->DrawCircle(pImpl, r);
  }

private:
  ClientCtxImpl*        pImpl;
  MethodTableClientCtx* pMethodTable;
};
```

```
static Context* CreateGenericCtx(
    ClientCtxImpl* p,
    MethodTableClientCtx* m) {
  return new GenericCtx(p, m);
}

static void DeleteGenericCtx(
    Context* pGenericCtx) {
  delete pGenericCtx;
}
```

Library uses implementaion through proxy

Library glue code

```
class ClientCtxImpl;

class GenericCtx : public Context {
public:
  GenericCtx(ClientCtxImpl* p,
             MethodTableClientCtx* m)
    : pImpl(p)
    , pMethodTable(m)
  {}

  void DrawCircle(float r) override {
    pMethodTable->DrawCircle(pImpl, r);
  }

private:
  ClientCtxImpl*         pImpl;
  MethodTableClientCtx* pMethodTable;
};
```

```
static Context* CreateGenericCtx(
    ClientCtxImpl* p,
    MethodTableClientCtx* m) {
  return new GenericCtx(p, m);
}

static void DeleteGenericCtx(
    Context* pGenericCtx) {
  delete pGenericCtx;
}


extern "C" {
  MethodTableGenericCtx DLLEXPORT
    gMethodTableGenericCtx = {
    2,
    &CreateGenericCtx,
    &DeleteGenericCtx
  };
}
```

Library exports MethodTable of the circle class too

Library glue code

```
static Circle* CreateCircle(float r) {
  return new Circle(r);
}

static void DeleteCircle(Circle* pCircle) {
  delete pCircle;
}

static void Draw(const Circle* pCircle,
                 Context* pCtx) {
  pCircle->Draw(pCtx);
}
```

```
extern "C" {
  MethodTableCircle DLLEXPORT
    gMethodTableCircle = {
    3,
    &CreateCircle,
    &DeleteCircle,
    &Draw
  };
}
```

Client side interface definition

```
class Context {
public:
  virtual ~Context();

  virtual void DrawCircle(float r) = 0;




};
```

Client side interface definition

```
class ContextImpl;

class Context {
public:
  virtual ~Context();

  virtual void DrawCircle(float r) = 0;

  ContextImpl* const pImpl;


};
```

Client side interface definition

```
class ContextImpl;

class Context {
public:
  virtual ~Context();

  virtual void DrawCircle(float r) = 0;

  ContextImpl* const pImpl;

protected:
  Context();
};
```

Client side interface definition

```
extern "C" MethodTableGenericCtx DLLIMPORT gMethodTableGenericCtx;

Context::Context()
  : pImpl(gMethodTableGenericCtx.Create(this, &gMethodTableCtx))
{}
```

Client side interface definition

```
extern "C" MethodTableGenericCtx DLLIMPORT gMethodTableGenericCtx;

Context::Context()
  : pImpl(gMethodTableGenericCtx.Create(this, &gMethodTableCtx))
{}

Context::~Context() {
  gMethodTableGenericCtx.Delete(pImpl);
}
```

Implementation of proxy class

```
class CircleImpl;

class Circle {
public:
  Circle(float r);
  ~Circle();

  void Draw(Context* pCtx) const;

private:
  Circle(CircleImpl* pCircleImpl);
  CircleImpl* pImpl;
};
```

```
extern "C" MethodTableCircle DLLIMPORT
gMethodTableCircle;

Circle::Circle(CircleImpl* pCircleImpl)
  : pImpl (pCircleImpl)
{}

Circle::Circle(float r)
  : Circle(gMethodTableCircle.Create(r))
{}

Circle::~Circle() {
  gMethodTableCircle.Delete(pImpl);
}

void Circle::Draw(Context* pCtx) const {
  return gMethodTableCircle.Draw(
                        pImpl, pCtx->pImpl);
}
```

Interface Classes

Library

```
class Context {
public:
  virtual ~Context() {}

  virtual void DrawCircle(float r) = 0;
};



class Circle {
public:
  Circle(float r) : mRadius(r) {}

  void Draw(Context* pCtx) const {
    pCtx->DrawCircle(mRadius);
  }

private:
  float mRadius;
};
```

Client

```
class DrawLogger : public Context {
  void DrawCircle(float r) override {
    std::cout << "DrawCircle(" << r << ")";
  }
};

int main() {
  DrawLogger logger;
  Circle c(5.0);
  c.Draw(&logger);
}
```

Interface Classes

Library

```
class Context {
public:
  virtual ~Context() {}

  virtual void DrawCircle(float r) = 0;
};



class Circle {
public:
  Circle(float r) : mRadius(r) {}

  void Draw(Context* pCtx) const {
    pCtx->DrawCircle(mRadius);
  }

private:
  float mRadius;
};
```

Client

```
class DrawLogger : public Context {
  void DrawCircle(float r) override {
    std::cout << "DrawCircle(" << r << ")";
  }
};

int main() {
  DrawLogger logger;
  Circle c(5.0);
  c.Draw(&logger);
}
```

Interface Classes

Library

```
class Context {
public:
  virtual ~Context() {}

  virtual void DrawCircle(float r) = 0;
};
```

```
class Circle {
public:
  Circle(float r) : mRadius(r) {}

  void Draw(Context* pCtx) const {
    pCtx->DrawCircle(mRadius);
  }

private:
  float mRadius;
};
```

Client

```
class DrawLogger : public Context {
  void DrawCircle(float r) override {
    std::cout << "DrawCircle(" << r << ")";
  }
};

int main() {
  DrawLogger logger;
  Circle c(5.0);
  c.Draw(&logger);
}
```

Interface Classes

Library

```
class Context {
public:
  virtual ~Context() {}

  virtual void DrawCircle(float r) = 0;
};
```

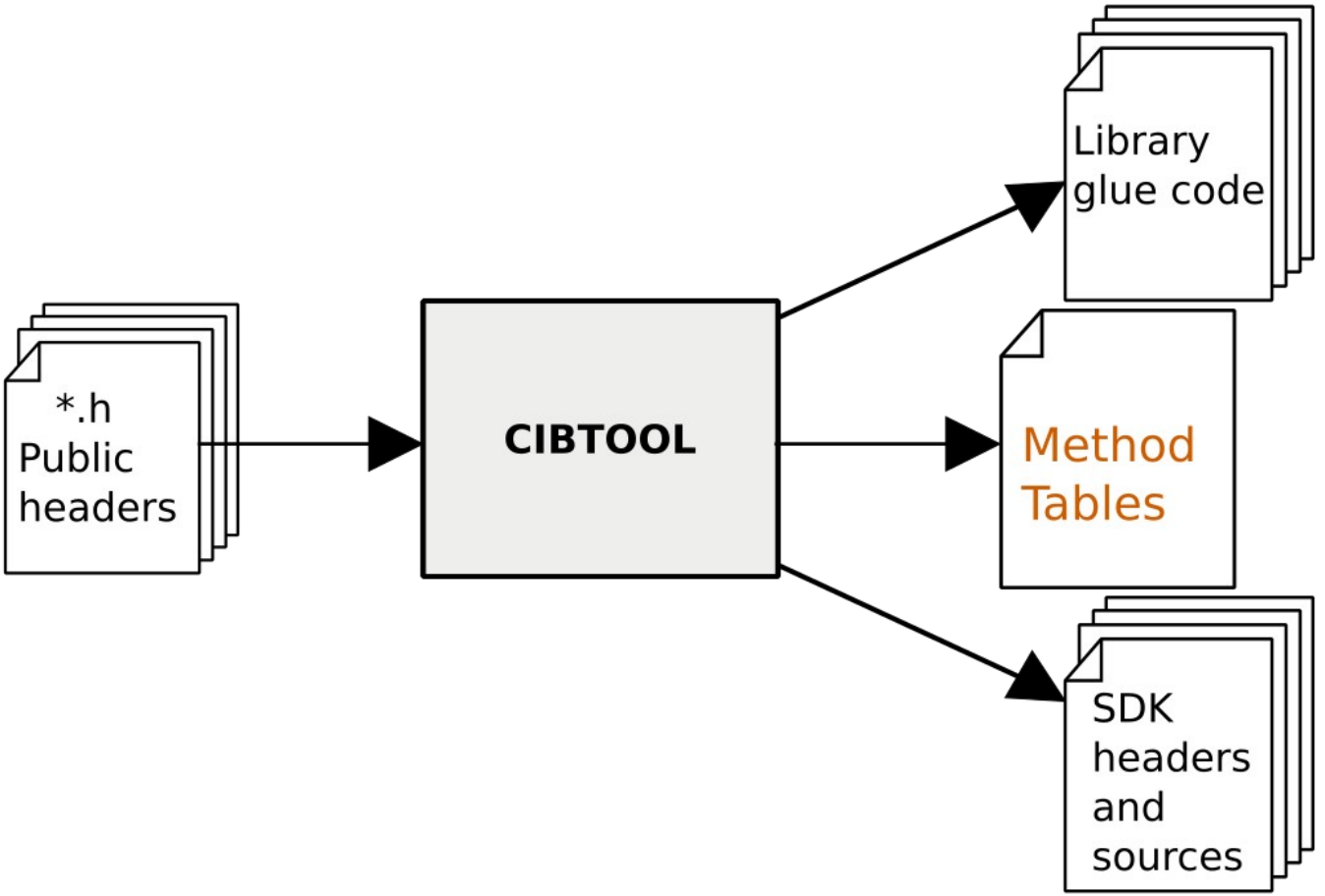
```
class Circle {
public:
  Circle(float r) : mRadius(r) {}

  void Draw(Context* pCtx) const {
    pCtx->DrawCircle(mRadius);
  }

private:
  float mRadius;
};
```

Client

```
class DrawLogger : public Context {
  void DrawCircle(float r) override {
    std::cout << "DrawCircle(" << r << ")";
  }
};

int main() {
  DrawLogger logger;
  Circle c(5.0);
  c.Draw(&logger);
}
```

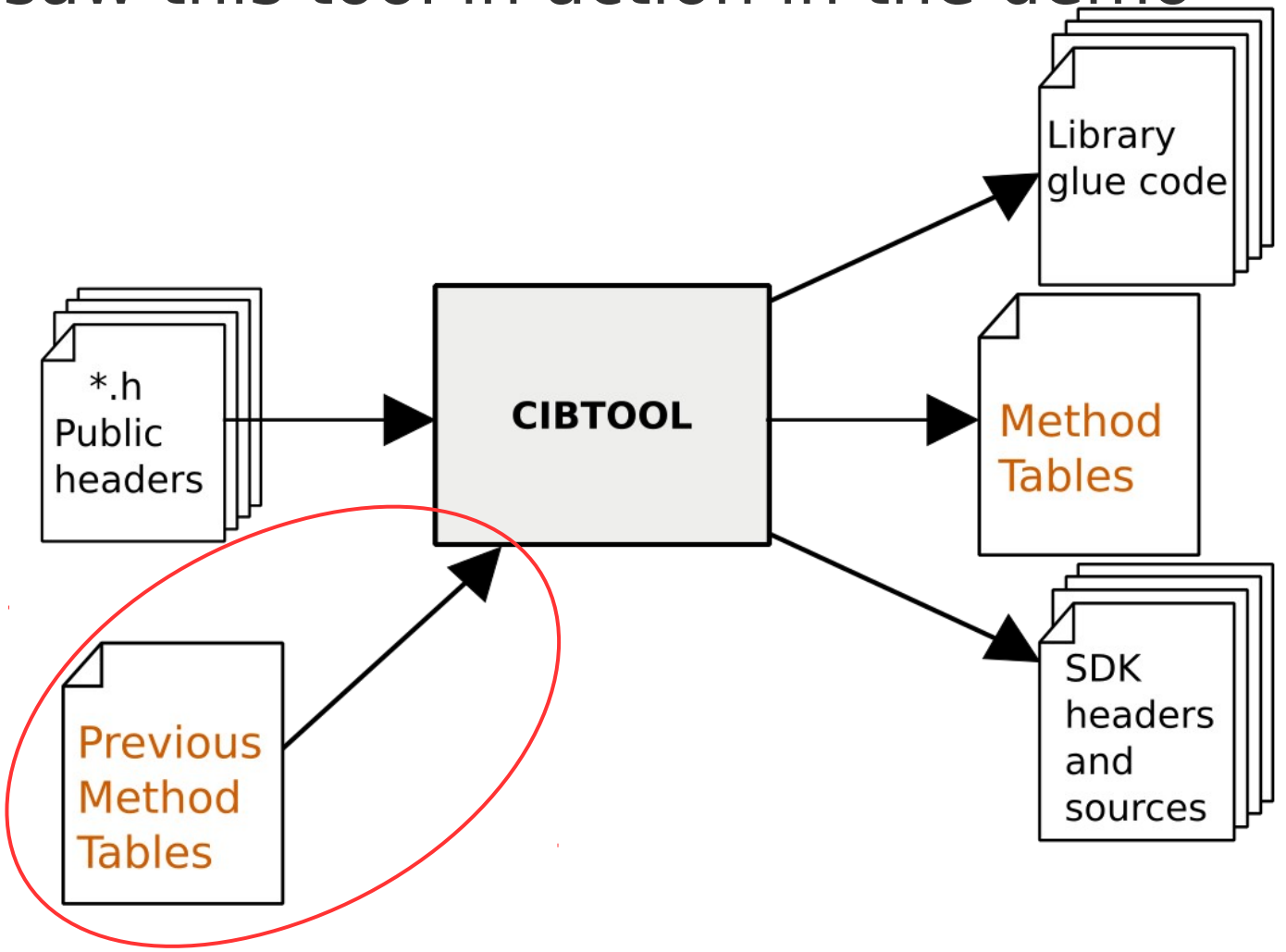
Auto generating glue code

- We can have a tool that can generate all this glue code automatically.



Glue code for next version

- Glue code generation for next version can use MethodTable of previous version.
- We saw this tool in action in the demo



Case Study (with PoDoFo library)

- PoDoFo is an open source PDF library with about 200 classes.
- Created cibified library and tested results.
- Examples and tests are run and results compared.
- To auto implement glue code I developed a tool github.com/satya-das/cib and used it for the case study.

Case Study Report

Library Binary Size		
Size Diff	**+3366 KB**	**+198%**

Case Study Report

Library Binary Size		
Size Diff	**+3366 KB**	**+198%**

Client Binary Size (Cummulative)		
Cumulative Size Diff	**+1989 KB**	**+94%**

Case Study Report

Library Binary Size		
Size Diff	+3366 KB	+198%

Client Binary Size (Cummulative)		
Cumulative Size Diff	+1989 KB	+94%

Memory Usage		
AVERAGE Memory Diff	+2,127,705 B	+4.26%

Case Study Report

Library Binary Size		
Size Diff	+3366 KB	+198%

Client Binary Size (Cummulative)		
Cumulative Size Diff	+1989 KB	+94%

Memory Usage		
AVERAGE Memory Diff	+2,127,705 B	+4.26%

Runtime Performance		
AVERAGE diff	+0.08 sec	+0.41%

What's more

- std::unique_ptr
- std::shared_ptr
- Intrusive pointers
- std::function
- lambda
- RTTI
- Templates
- Exceptions
- Object lifecycle management
- STL objects
- Global functions
- Protected methods
- Conditional APIs
- POD struct
- Value classes
- Rvalue Reference
- Etc.

Most things are possible and examples are already available at github.com/satya-das/cib

Thank You!!