bonding Perl and C++ with minimal pain

Steffen Müller

August 4, 2010

Prequel – XS

- What is XS?
- *DSL for interfacing between Perl and C*

- Myth: XS is hard!
- Truth: XS is quite simple, just the syntax is horrible

- The perl API is the difficult bit!

Prequel – XS

- What is XS?
- *DSL for interfacing between Perl and C*

- Myth: XS is hard!
- Truth: XS is quite simple, just the syntax is horrible

- The perl API is the difficult bit!

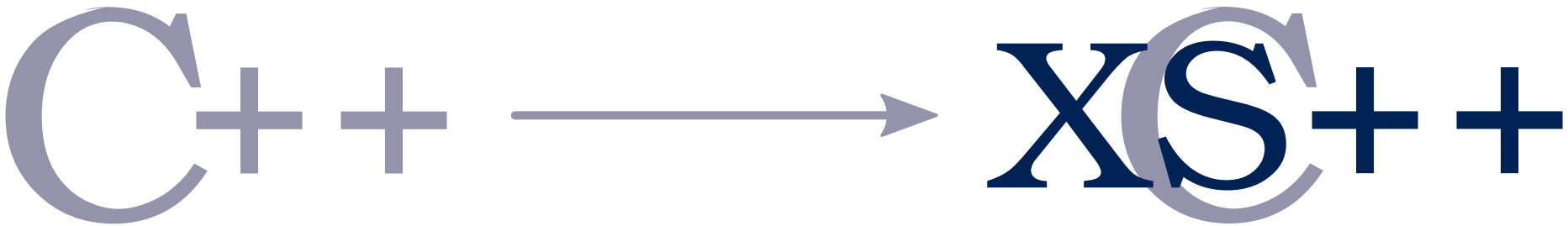
Prequel – XS

- What is XS?
- *DSL for interfacing between Perl and C*

- Myth: XS is hard!
- Truth: XS is quite simple, just the syntax is horrible

- The perl API is the difficult bit!

What is XS++?



- A DSL (in this regard like XS)
- Completely different syntax from XS!
- Looks like C(++) header with annotations
- Intended for wrapping C++
  (but can wrap plain functions, too)

A Contrived Example

```
class Dog : public Animal
{
  Dog();
  ~Dog();

  void Bark();
  const std::string& GetName();
  void SetName(const std::string& name);
};
```

A Contrived Example

```
%module{Animals};

class Dog : public Animal
{
  Dog(); // will be ->new()
  ~Dog(); // will be ->DESTROY()

  void Bark();
  const std::string& GetName();
  void SetName(const std::string& name);
};
```

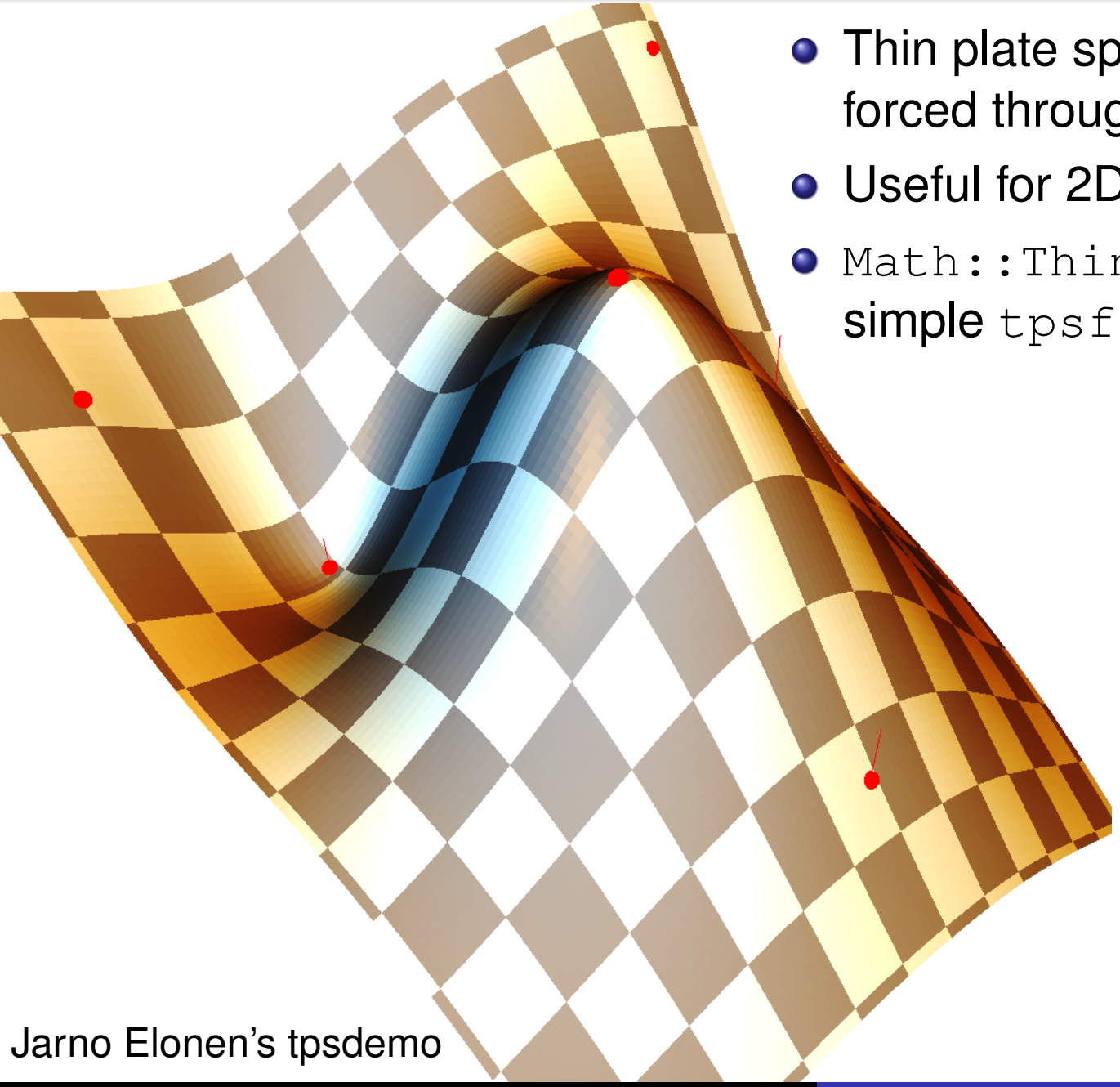
A Contrived Example

```
%module{Animals};

class Dog : public Animal
{
  Dog(); // will be ->new()
  ~Dog(); // will be ->DESTROY()

  %name{bark}
    void Bark();
  %name{get_name}
    const std::string& GetName();
  %name{set_name}
    void SetName(const std::string& name);
};
```

A Contrived Example

```
use Animals;

{
  my $dog = Dog->new;
  isa_ok($dog, "Dog");
  isa_ok($dog, "Animal");

  $dog->bark;
  $dog->set_name("skip");
  isa_ok($dog->get_name(), "skip");

} # $dog->DESTROY aka ~Dog() called here
```

A Real Example



Jarno Elonen's tpsdemo

- Thin plate spline: "thin metal plate" forced through control points
- Useful for 2D interpolation
- `Math::ThinPlateSpline` wraps simple `tpsfit` library

A Real Example

```
class ThinPlateSpline {
  ThinPlateSpline(const vector<Vec3D>& controlPoints,
                  const double regularization = 0.);

  // Alternative constructor from serialization:
  ThinPlateSpline(istream& input);
  ~ThinPlateSpline();

  double Evaluate(const double x,
                  const double y) const;
}
```

A Real Example

```
%module{Math::ThinPlateSpline};

%name{Math::ThinPlateSpline}
class ThinPlateSpline {
  ThinPlateSpline(const vector<Vec3D>& controlPoints,
                  const double regularization = 0.);
  %name{new_from_string}
  ThinPlateSpline(istream& input);
  ~ThinPlateSpline();

  %name{evaluate}
  double Evaluate(const double x,
                  const double y) const;
};
```

A Real Example

```
use Math::ThinPlateSpline;

my $spline = Math::ThinPlateSpline->new(
   [
     [0, 0, 0],     # control points
     [1, 2, 0],
     ...
   ]
);

my $z = $spline->evaluate($x, $y);
```

Type conversion!

@

\&

$

*

%

\(...)

unsigned int

const double

const SomeClass&

void (Foo::*funcptr)(float)

std::string

std::vector<int>

Typemaps

Concept

Write code for type conversion between Perl and C only once

Traditional XS typemaps

- Contain actual type conversion C code using perl API
- Material for a talk on their own!
- `perlxstut.pod` (typemap section)
- `http://p3rl.org/xs#The-Typemap`
- (In XS++), these live in `*.map` files

Typemaps

Concept

Write code for type conversion between Perl and C only once

Traditional XS typemaps

- Contain actual type conversion C code using perl API
- Material for a talk on their own!
- `perlxstut.pod` (typemap section)
- `http://p3rl.org/xs#The-Typemap`
- (In XS++), these live in `*.map` files

Typemaps

XS++ typemaps

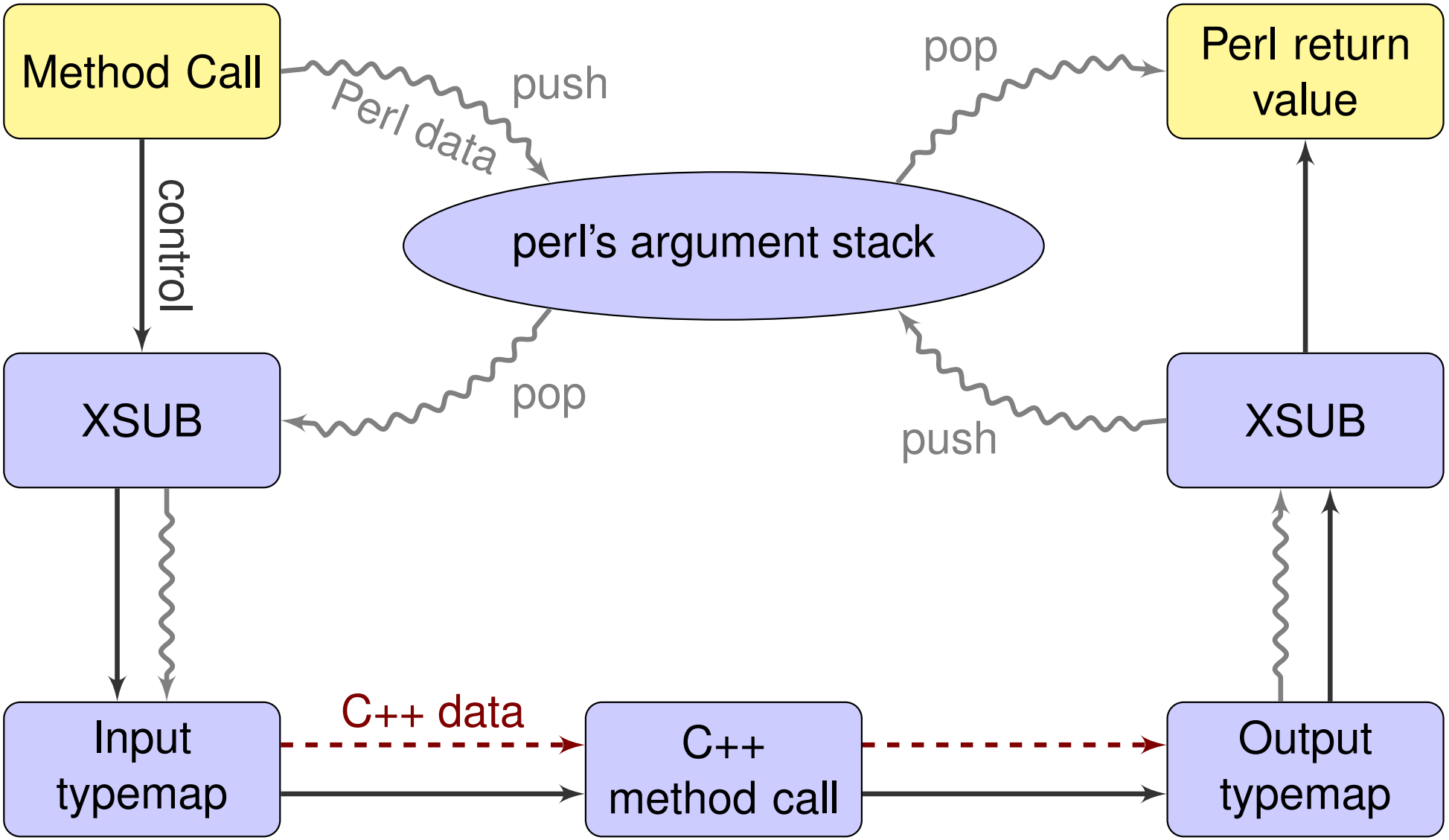
- XS++ sits on top of XS
- → XS++ typemaps sit on top of XS typemaps
- Future: Intended to replace XS typemaps
- Declare as-yet unknown types to the parser
- `%typemap{SomeType};`
  Simply declares the type (and its reference type).
- `%typemap{SomeType}{parsed}{...};`
  gives full control
- Go into `*.xspt` files for reuse

Typemaps

XS++ typemaps

- XS++ sits on top of XS
- → XS++ typemaps sit on top of XS typemaps
- Future: Intended to replace XS typemaps
- Declare as-yet unknown types to the parser
- `%typemap{SomeType};`
  Simply declares the type (and its reference type).
- `%typemap{SomeType}{parsed}{...};`
  gives full control
- Go into `*.xspt` files for reuse

What happens on `$obj->method(...)`?



XS++ Typemap Example

- Code to turn a returned array of integers into a Perl list
- `const std::vector<int>& method();`
- `@integers = $obj->method()`

```
%typemap{const std::vector<int>&}{parsed}{

  %cpp_type{%std::vector<int>%};

  %output_list{%
    for(size_t i = 0; i < RETVAL.size(); ++i) {
        mXPUSHi( RETVAL[i] );
  %};

};
```

Example CPAN distribution

```
lib/                # ordinary Perl modules and docs
    Math/ThinPlateSpline.pm
src/                # arbitrary C++ stuff
    ThinPlateSpline.h
    ThinPlateSpline.cc
    ...
    ppport.h        # ports new perlapi to older perls
xsp/                # XS++ code goes here
    ThinPlateSpline.xsp
    type.map        # XS type map
Build.PL
```

Example CPAN distribution

```
lib/                    # ordinary Perl modules and docs
    Math/ThinPlateSpline.pm
src/                    # arbitrary C++ stuff
    ThinPlateSpline.h
    ThinPlateSpline.cc
    ...
    ppport.h            # ports new perlapi to older perls
xsp/                    # XS++ code goes here
    ThinPlateSpline.xsp
    type.map            # XS type map
Build.PL
```

Example CPAN distribution

```
# Same as normal XS-based module:
package Math::ThinPlateSpline;
use 5.012;
use warnings;
our $VERSION = '0.04';

require XSLoader;
XSLoader::load('Math::ThinPlateSpline', $VERSION);

1;
```

Example CPAN distribution

```
lib/                # ordinary Perl modules and docs
    Math/ThinPlateSpline.pm
src/                # arbitrary C++ stuff
    ThinPlateSpline.h
    ThinPlateSpline.cc
    ...
    ppport.h        # ports new perlapi to older perls
xsp/                # XS++ code goes here
    ThinPlateSpline.xsp
    type.map        # XS type map
Build.PL
```

Example CPAN distribution

```
lib/                # ordinary Perl modules and docs
    Math/ThinPlateSpline.pm
src/                # arbitrary C++ stuff
    ThinPlateSpline.h
    ThinPlateSpline.cc
    ...
    ppport.h        # ports new perlapi to older perls
xsp/                # XS++ code goes here
    ThinPlateSpline.xsp
    type.map        # XS type map
Build.PL
```

Example CPAN distribution

```
lib/                # ordinary Perl modules and docs
    Math/ThinPlateSpline.pm
src/                # arbitrary C++ stuff
    ThinPlateSpline.h
    ThinPlateSpline.cc
    ...
    ppport.h        # ports new perlapi to older perls
xsp/                # XS++ code goes here
    ThinPlateSpline.xsp
    type.map        # XS type map
Build.PL
```

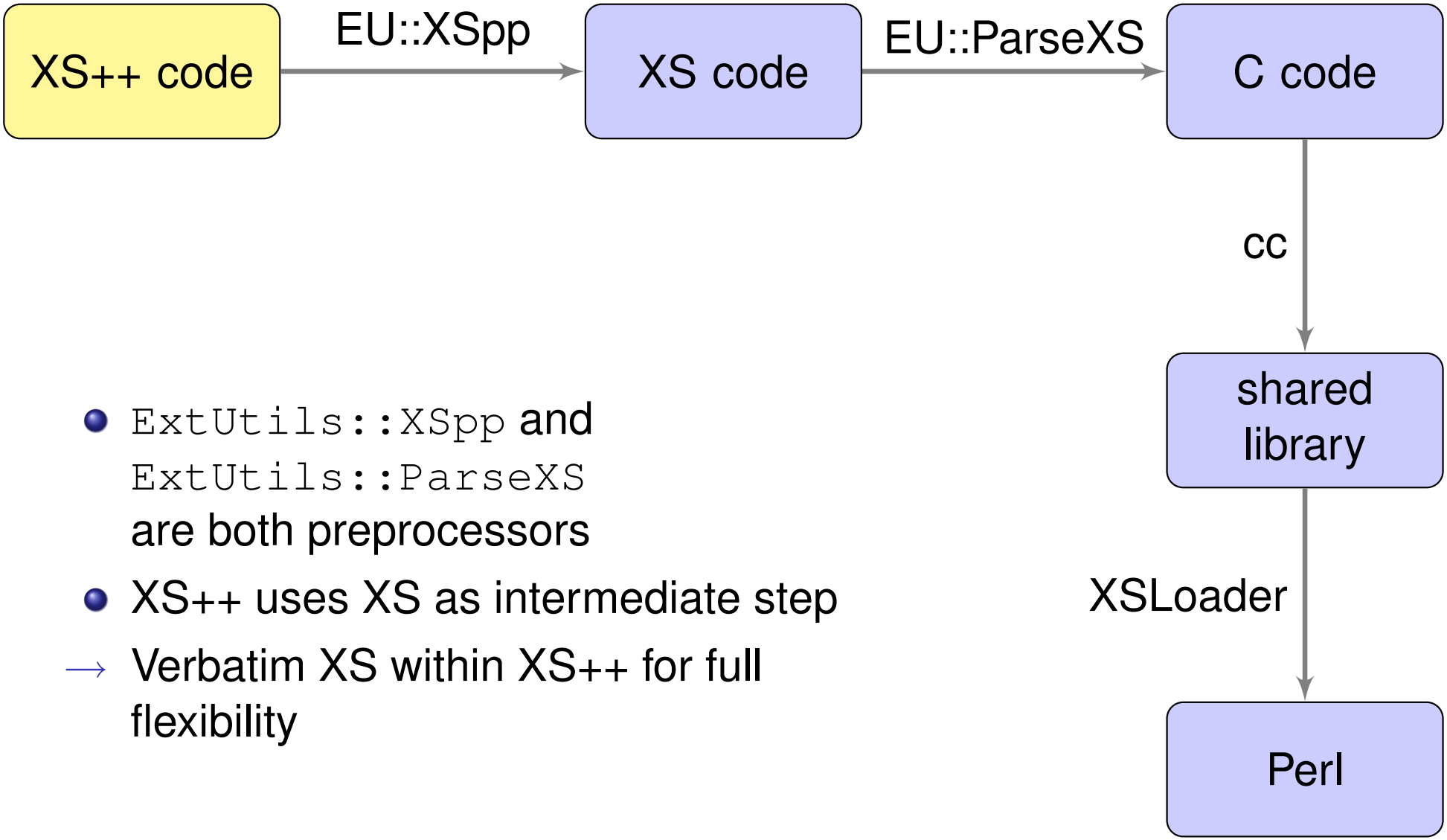
Example CPAN distribution

```
#!/usr/bin/perl -w
use 5.012;
use Module::Build::WithXSpp;

my $build = Module::Build::WithXSpp->new(
  module_name     => 'Math::ThinPlateSpline',
  license         => 'perl',
  # Provides extra C typemaps that are auto-merged
  extra_typemap_modules => {
    'ExtUtils::Typemap::Default' => '0.01',
  },
  # extra_linker_flags   => '-lMyLibrary',
  # extra_compiler_flags => ...
);

$build->create_build_script;
```

Building an extension

- `perl Build.PL`...
  - ...detects the C++ compiler and sets correct flags
- `./Build`...
  - ...compiles and links all C++ in `src/`
  - ...pulls in predefined typemaps from modules such as `ExtUtils::Typemap::STL`
  - ...and merges them with your custom ones
  - ...generates main XS file
  - ...compiles and links everything into shared library
- `./Build test`
- `./Build install`
- `./Build dist`...
  - ...automatically adds dependencies on the build system

How does it work at compile time?



- `ExtUtils::XSpp` and `ExtUtils::ParseXS` are both preprocessors
- XS++ uses XS as intermediate step

→ Verbatim XS within XS++ for full flexibility

C++ exception handling

- Automatically converted to Perl `croak(...)`'s:
  `Caught C++ exception of type '$sometype'`
- Customization with `%exception{...}{...}` declaration
- Class and method fine tuning with `%catch{...}` decorator.

```
class Dog %catch{itch} {
  void scratch() %catch{sore}; // itch, too!
};
```

C++ exception handling

- Automatically converted to Perl `croak(...)`'s:
  `Caught C++ exception of type '$sometype'`
- Customization with `%exception{...}{...}` declaration
- Class and method fine tuning with `%catch{...}` decorator.

```
%exception{itch}{CppItchException}{stdmessage};
class Dog %catch{itch} {
  void scratch() %catch{sore}; // itch, too!
};
```

C++ exception handling

- Automatically converted to Perl `croak(...)`'s:
  `Caught C++ exception of type '$sometype'`
- Customization with `%exception{...}{...}` declaration
- Class and method fine tuning with `%catch{...}` decorator.

```
%exception{itch}{CppItchException}{stdmessage};
%exception{sore}{CppSoreException}{code}{
  Your exception handling XS here!
};
class Dog %catch{itch} {
  void scratch() %catch{sore}; // itch, too!
};
```

C++ exception handling

- Automatically converted to Perl `croak(...)`'s:
  `Caught C++ exception of type '$sometype'`
- Customization with `%exception{...}{...}` declaration
- Class and method fine tuning with `%catch{...}` decorator.

```
%exception{itch}{CppItchException}{stdmessage};
%exception{sore}{CppSoreException}{code}{
  Your exception handling XS here!
};
class Dog %catch{itch} {
  void scratch() %catch{sore}; // itch, too!
  void no_thanks() %catch{nothing}; // opt out
};
```

Availability and Acknowledgment

- Code:
  - `http://search.cpan.org/dist/ExtUtils-XSpp`
  - `http://search.cpan.org/dist/Module-Build-WithXSpp`
  - `http://github.com/mbarbon/extutils-xspp`
  - `http://github.com/tsee/module-build-withxspp`
- Documentation:
  - `http://search.cpan.org/perldoc?ExtUtils::XSpp`
- Slides:
  - `http://steffen-mueller.net/#talks`
- More on XS:
  - Vincent's XS slides: `http://profvince.com/perl/xs.pdf`
  - `perlxstut`, `perlxs`, `perlguts`, `perlapi`, etc.

Thanks to Mattia Barbon for doing all the work and letting me present it!