# Object-Relational Mapping with ODB and Boost

Boris Kolpackov

Code Synthesis

v1.2, May 2011

# ORM for C++

## No

- No manual parameter binding
- No manual result set extraction
- No hand-written mapping or registration code
- Not a framework

# ORM for C++

## Yes

- Automatic generation of mapping code from C++ classes
- Handle any standard C++ code
- Object-oriented database API
- Statically-typed, C++-integrated query language
- Database portability
- Flexible and customizable

# GCC Plugin Support

- Available since GCC 4.5.0/April 2010
- Current releases are GCC 4.5.3 and 4.6.0

# GCC Plugin Support

- Available since GCC 4.5.0/April 2010
- Current releases are GCC 4.5.3 and 4.6.0

- Dynamic loading
- Hook into compilation pipeline *anywhere*
- ...starting from compiler startup
- ...ending with assembler output

# Platforms

*ODB can be used with any modern C++ compiler*

- GNU/Linux with GCC
- Mac OS X with GCC
- Solaris (x86 and SPARC) with Sun Studio C++
- Windows with GCC (MinGW) and VC++ 2008 and 2010

# Databases

*Cross-database*

| **Supported** | **Coming Soon** |
| --- | --- |
| MySQL | PostgreSQL |
| SQLite | Oracle |
| | MS SQL |

# Performance

*High-performance and low overhead*

- Prepared statements
- Caching of connections, statements, and buffers
- Low-level native database APIs
- Zero per-object memory overhead

# Performance

*High-performance and low overhead*

- Prepared statements
- Caching of connections, statements, and buffers
- Low-level native database APIs
- Zero per-object memory overhead

Loading performance

- MySQL — 21,000 objects per second — 48 μs per object
- SQLite —143,000 object per second — 7 μs per object

# License

*Dual-licensed*

- GPL + commercial license
- Can be used without restrictions within your organization

# Declaring a Persistent Class

```
enum status {open, confirmed, closed};

class bug
{
public:
  bug (const std::string& summary,
       const std::string& description);
  ...

private:
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

# Declaring a Persistent Class

```
enum status {open, confirmed, closed};

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

# Declaring a Persistent Class

```
enum status {open, confirmed, closed};

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

# Declaring a Persistent Class

```
enum status {open, confirmed, closed};

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

# Declaring a Persistent Class

```
enum status {open, confirmed, closed};

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

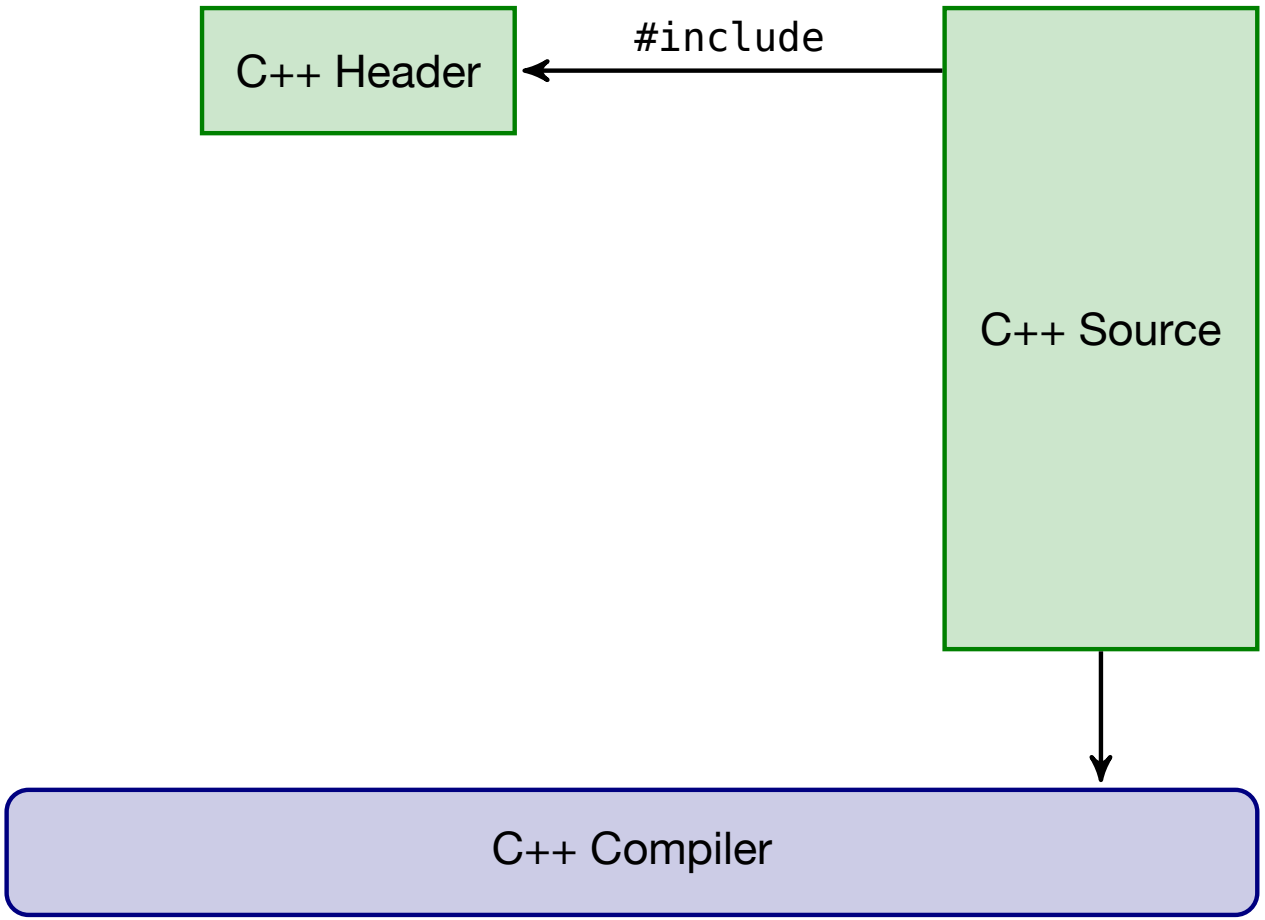
# Declaring a Persistent Class

```
enum status {open, confirmed, closed};

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

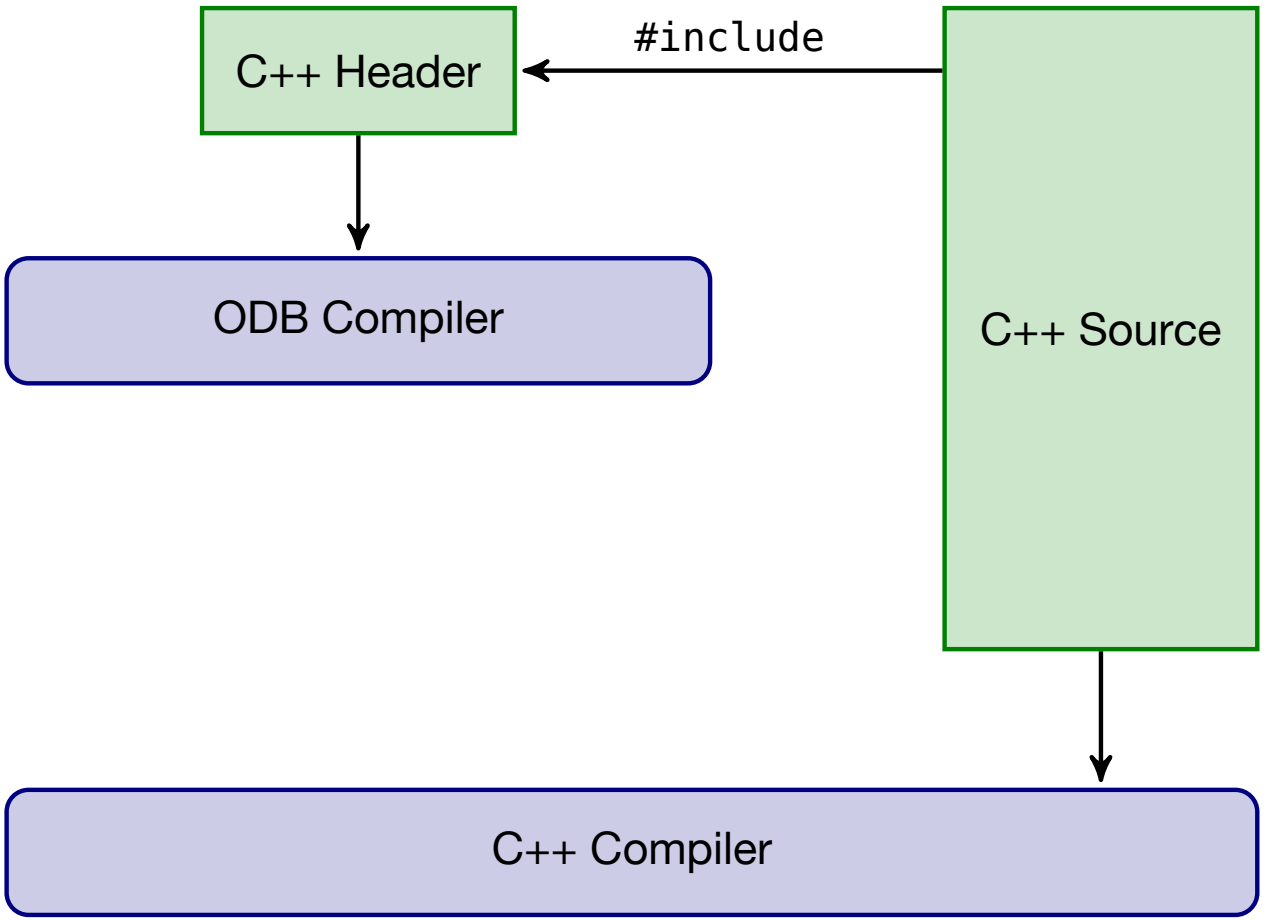
# Declaring a Persistent Class

```
enum status {open, confirmed, closed};

#pragma db object
class bug
{
  ...
private:
  friend class odb::access;
  bug () {}

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;
};
```

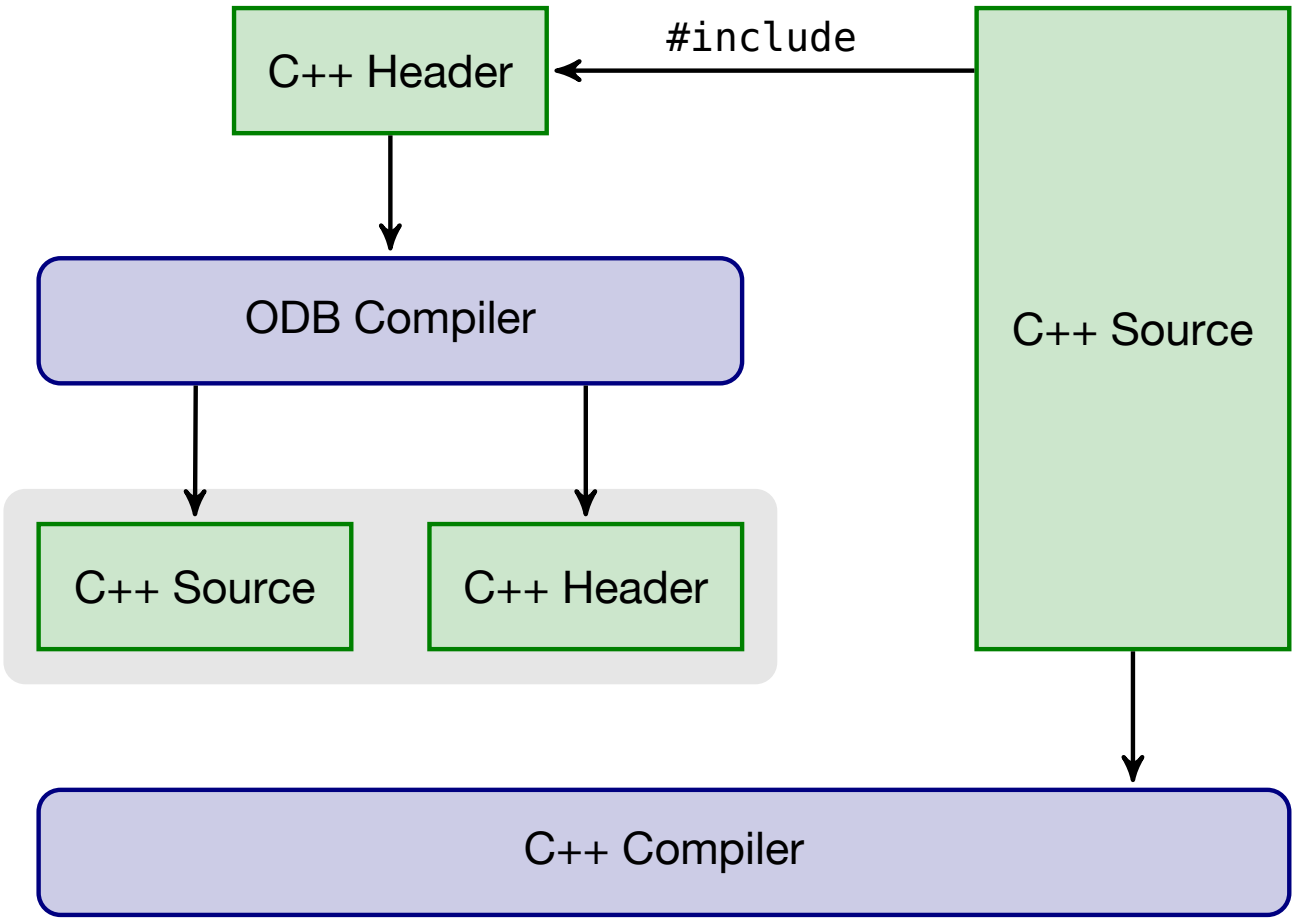
# Workflow



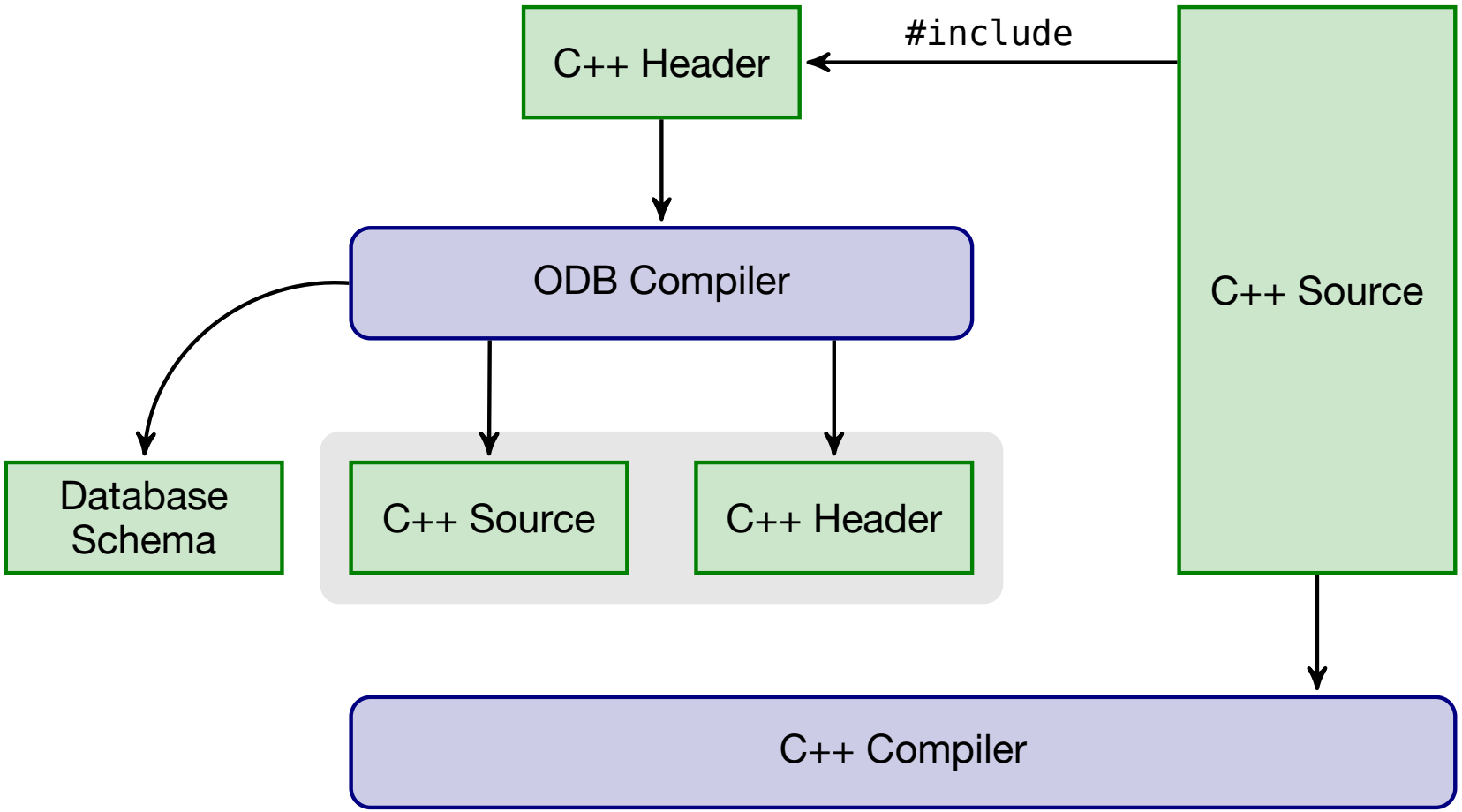
# Workflow



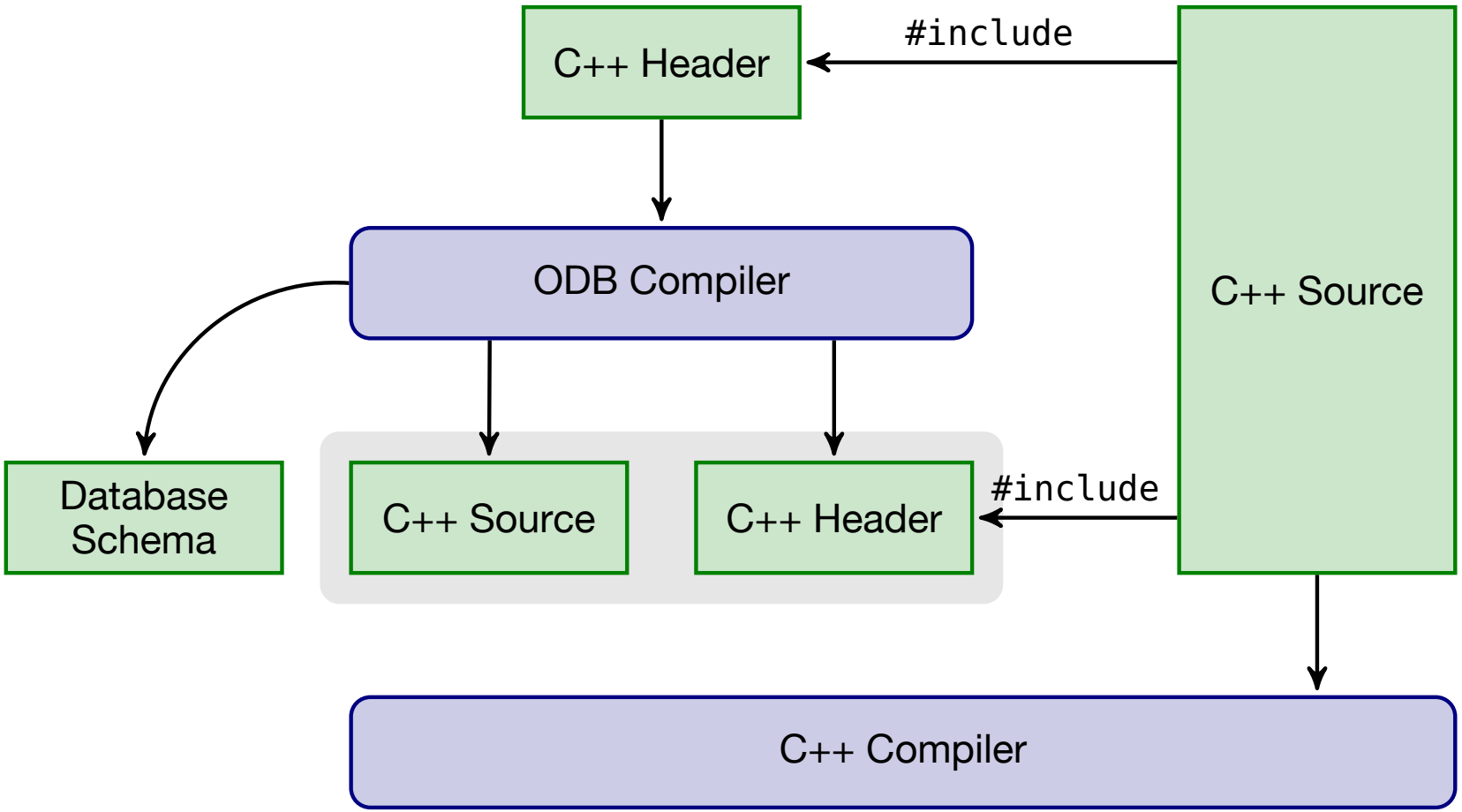
# Workflow


C++ Header
#include
C++ Source
ODB Compiler
C++ Source
C++ Header
C++ Compiler

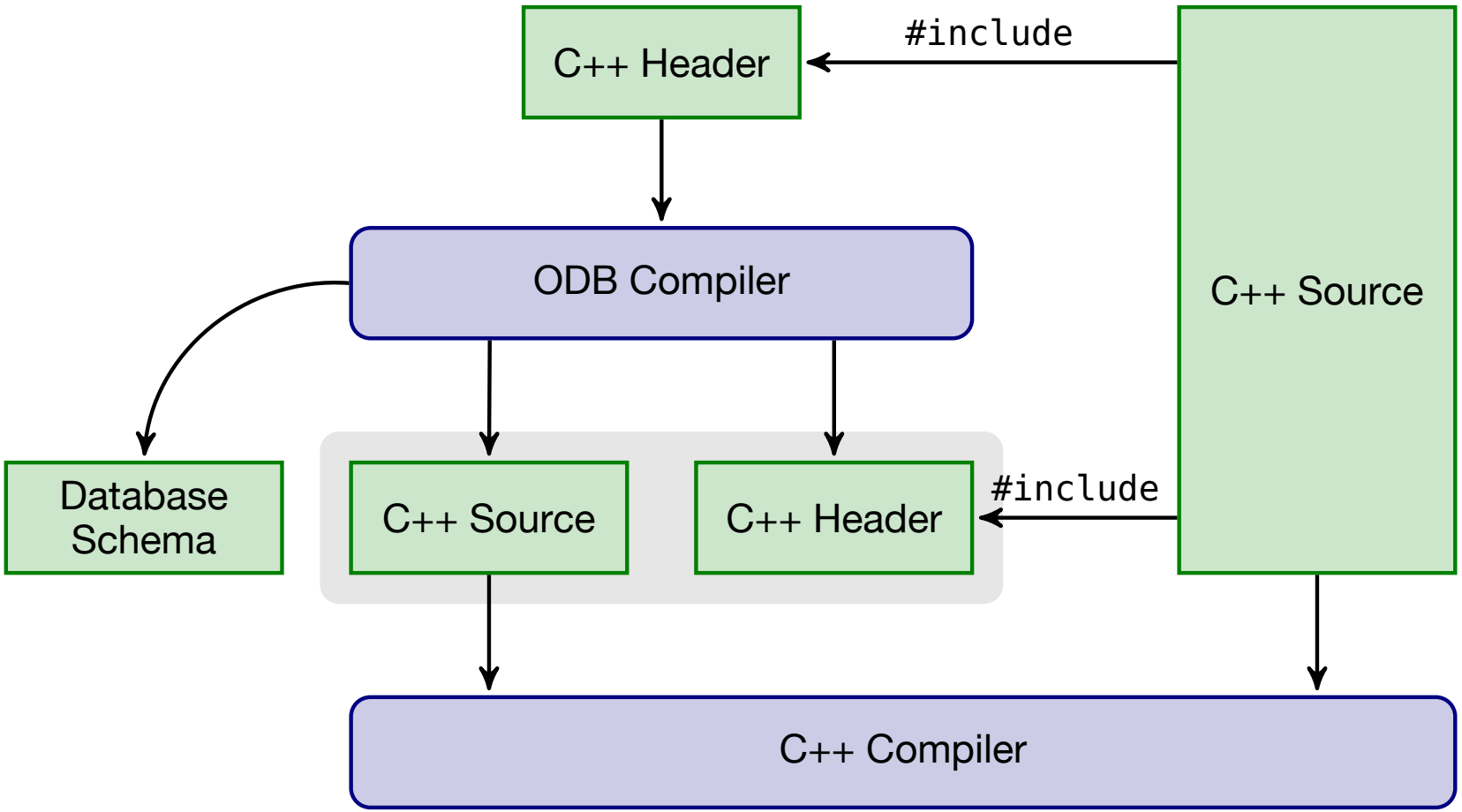
# Workflow


C++ Header
#include
C++ Source
ODB Compiler
Database Schema
C++ Source
C++ Header
C++ Compiler

# Workflow


C++ Header
#include
C++ Source
ODB Compiler
Database Schema
C++ Source
C++ Header
#include
C++ Compiler

# Workflow


C++ Header
#include
C++ Source
ODB Compiler
Database Schema
C++ Source
C++ Header
#include
C++ Compiler

# ODB Compiler

```
odb --database mysql --generate-schema bug.hxx
```

# ODB Compiler

```
odb --database mysql --generate-schema bug.hxx

odb -I/opt/boost -d mysql -s bug.hxx
```

# ODB Compiler

```
odb --database mysql --generate-schema bug.hxx

odb -I/opt/boost -d mysql -s bug.hxx
```

```
CREATE TABLE bug (
  id INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  status ENUM('open', 'confirmed', 'closed') NOT NULL,
  summary TEXT NOT NULL,
  description TEXT NOT NULL);
```

# Database

```
#include <odb/database.hxx>
#include <odb/mysql/database.hxx>

shared_ptr<odb::database> db (
  new odb::mysql::database ("bugtracker", // user
                            "secret",     // password
                            "bugs"));     // database
```

# Database

```
#include <odb/database.hxx>
#include <odb/mysql/database.hxx>

shared_ptr<odb::database> db (
  new odb::mysql::database ("bugtracker", // user
                            "secret",     // password
                            "bugs"));     // database


#include <odb/sqlite/database.hxx>

shared_ptr<odb::database> db (
  new odb::sqlite::database ("bugs.db")); // database
```

# Making an Object Persistent

```
bug b ("Support for PostgreSQL",
       "ODB does not yet support PostgreSQL.");

transaction t (db->begin ());

unsigned int id = db->persist (b);

t.commit ();
```

# Making an Object Persistent

```
bug b ("Support for PostgreSQL",
       "ODB does not yet support PostgreSQL.");

transaction t (db->begin ());

unsigned int id = db->persist (b);

t.commit ();
```

```
=> INSERT INTO bug (
     id,
     status,
     summary,
     description)
   VALUES (?, ?, ?, ?)
```

# Loading a Persistent Object

```
transaction t (db->begin ());

shared_ptr<bug> b (db->load<bug> (id));

// bug b;
// db->load (id, b);

t.commit ();
```

# Loading a Persistent Object

```
transaction t (db->begin ());

shared_ptr<bug> b (db->load<bug> (id));

// bug b;
// db->load (id, b);

t.commit ();
```

```
=> SELECT
     status,
     summary,
     description
   FROM bug WHERE id = ?
```

# Updating a Persistent Object

```
transaction t (db->begin ());

shared_ptr<bug> b (db->load<bug> (id));
b->confirm ();
db->update (b);

t.commit ();
```

# Updating a Persistent Object

```
transaction t (db->begin ());

shared_ptr<bug> b (db->load<bug> (id));
b->confirm ();
db->update (b);

t.commit ();
```

```
=> UPDATE bug SET
     status = ?,
     summary = ?,
     description = ?
   WHERE id = ?
```

# Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;

transaction t (db->begin ());

result r (db->query<bug> (query::status == open);

for (result::iterator i (r.begin ()); i != r.end (); ++i)
  cout << i->id () << " " << i->summary () << endl;

t.commit ();
```

# Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;

transaction t (db->begin ());

result r (db->query<bug> (query::status == open);

for (result::iterator i (r.begin ()); i != r.end (); ++i)
  cout << i->id () << " " << i->summary () << endl;

t.commit ();
```

# Querying the Database

```
typedef odb::query<bug> query;
typedef odb::result<bug> result;

transaction t (db->begin ());

result r (db->query<bug> (query::status == open);

for (result::iterator i (r.begin ()); i != r.end (); ++i)
  cout << i->id () << " " << i->summary () << endl;

t.commit ();
```

```
=> SELECT
     id
     status,
     summary,
     description
   FROM bug WHERE status = ?
```

# Querying the Database

```
db->query<bug> (query::status == open ||
                query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db->query<bug> (q); // status == open

s = closed;
db->query<bug> (q); // status == closed

db->query<bug> ("status = " + query::_val (open));

db->query<bug> ("stats = " + query::_val (123));
```

# Querying the Database

```
db->query<bug> (query::status == open ||
                query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db->query<bug> (q); // status == open

s = closed;
db->query<bug> (q); // status == closed

db->query<bug> ("status = " + query::_val (open));

db->query<bug> ("stats = " + query::_val (123));
```

# Querying the Database

```
db->query<bug> (query::status == open ||
                query::status == confirmed);
```

```
status s;
query q (query::status == query::_ref (s));

s = open;
db->query<bug> (q); // status == open

s = closed;
db->query<bug> (q); // status == closed
```

```
db->query<bug> ("status = " + query::_val (open));

db->query<bug> ("stats = " + query::_val (123));
```

# Querying the Database

```
db->query<bug> (query::status == open ||
                query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db->query<bug> (q); // status == open

s = closed;
db->query<bug> (q); // status == closed

db->query<bug> ("status = " + query::_val (open));

db->query<bug> ("stats = " + query::_val (123));
```

# Querying the Database

```
db->query<bug> (query::status == open ||
                query::status == confirmed);

status s;
query q (query::status == query::_ref (s));

s = open;
db->query<bug> (q); // status == open

s = closed;
db->query<bug> (q); // status == closed

db->query<bug> ("status = " + query::_val (open));

db->query<bug> ("stats = " + query::_val (123));
```

# Deleting a Persistent Object

```
transaction t (db->begin ());

db->erase<bug> (id);

t.commit ();
```

# Deleting a Persistent Object

```
transaction t (db->begin ());

db->erase<bug> (id);

t.commit ();
```

```
=> DELETE FROM bug WHERE id = ?
```

# Profiles

- Generic integration mechanism
- Covers smart pointers, containers, and value types
- ODB includes profiles for Boost and Qt
- You can write your own

```
odb -d mysql -p boost bug.hxx
```

# Boost Profile

- `shared_ptr/weak_ptr` + their lazy variants
- `unordered` containers library
- `date_time` library

# Adding Creation and Modification Dates

```
#pragma db object
class bug
{
  ...

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;

  boost::posix_time::ptime created_;
  boost::posix_time::ptime updated_;
};
```

# Containers

- Standard containers: `vector`, `list`, `set`, `map`, etc
- Profiles provide additional containers
- Easy to support custom containers

# Adding Comments and Tags

```
#pragma db object
class bug
{
  ...

  #pragma db id auto
  unsigned int id_;

  status status_;
  std::string summary_;
  std::string description_;

  boost::posix_time::ptime created_;
  boost::posix_time::ptime updated_;

  std::vector<std::string> comments_;
  boost::unordered_set<std::string> tags_;
};
```

# Composite Value Types

- Class or struct type
- Mapped to more than one database column
- Can contain composite values, containers, and pointers to objects

# Extending Comments

```
#pragma db value
class comment
{
  ...

  std::string text_;
  boost::posix_time::ptime created_;
};

#pragma db object
class bug
{
  ...

  std::vector<comment> comments_;
};
```

# Relationships

- Relationships are represented as pointers to objects
- Standard pointers: raw, `auto_ptr`, `tr1::shared_ptr`
- Profiles provide additional pointers
- Easy to support custom smart pointers

# Adding User Object

```
#pragma db object
class user
{
  ...

  #pragma db id
  std::string email_;

  std::string first_;
  std::string last_;
};
```

# Adding Reporter

```
#pragma db object
class bug
{
  ...

  shared_ptr<user> reporter_;
};
```

# Adding Reporter

```
#pragma db object
class bug
{
  ...

  shared_ptr<user> reporter_;
};
```

Example of a *unidirectional to-one* relationship.

# Adding Bug List

```
#pragma db object
class user
{
  ...

  #pragma db id
  std::string email_;

  std::string first_;
  std::string last_;

  #pragma db inverse(reporter_)
  std::vector<shared_ptr<bug> > reported_bugs_;
};
```

# Adding Bug List

```
#pragma db object
class user
{
  ...

  #pragma db id
  std::string email_;

  std::string first_;
  std::string last_;

  #pragma db inverse(reporter_)
  std::vector<shared_ptr<bug> > reported_bugs_;
};
```

Example of a *bidirectional many-to-one* relationship.

# Adding Bug List

```
#pragma db object
class user
{
  ...

  #pragma db id
  std::string email_;

  std::string first_;
  std::string last_;

  #pragma db inverse(reporter_)
  std::vector<shared_ptr<bug> > reported_bugs_;
};
```

Example of a *bidirectional many-to-one* relationship.

# Inverse Side of a Relationship

```
CREATE TABLE user (
  email VARCHAR (255) NOT NULL PRIMARY KEY,
  ...);

CREATE TABLE user_reported_bugs (
  object_id VARCHAR (255) NOT NULL,
  index BIGINT UNSIGNED NOT NULL,
  value INT UNSIGNED REFERENCES bug (id));

CREATE TABLE bug (
  id INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  ...
  reporter VARCHAR (255) REFERENCES user (email));
```

# Inverse Side of a Relationship

```
CREATE TABLE user (
  email VARCHAR (255) NOT NULL PRIMARY KEY,
  ...);

CREATE TABLE bug (
  id INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  ...
  reporter VARCHAR (255) REFERENCES user (email));
```

# We Have a Problem

```
#pragma db object
class user
{
  ...

  #pragma db inverse(reporter_)
  std::vector<shared_ptr<bug> > reported_bugs_;
};

#pragma db object
class bug
{
  ...

  shared_ptr<user> reporter_;
};
```

# We Have a Problem

Actually, we have two:

1. Ownership cycle between `user` and `bug`
2. Eager loading of the bug list in `user`

# We Have a Problem

Actually, we have two:

1. Ownership cycle between `user` and `bug`
2. Eager loading of the bug list in `user`

```
#pragma db object
class user
{
  ...

  #pragma db inverse(reporter_)
  std::vector<weak_ptr<bug> > reported_bugs_;
};
```

# Lazy Pointers

- Finer grained control over relationship loading
- Every supported pointer has a corresponding lazy version

# Lazy Pointers

- Finer grained control over relationship loading
- Every supported pointer has a corresponding lazy version

```
#pragma db object
class user
{
  ...

  #pragma db inverse(reporter_)
  std::vector<lazy_weak_ptr<bug> > reported_bugs_;
};

lazy_weak_ptr<bug> lwp = ...
shared_ptr<bug> b (lwp.load ()); // Load and lock.
```

# Generated or Custom Schema

- Database schema can be automatically generated
- Or we can map persistent classes to a custom schema

# Generated Schema

- Standalone SQL file
- Embedded into the generated C++ code

# Generated Schema

- Standalone SQL file
- Embedded into the generated C++ code

```
shared_ptr<odb::database> db (
  new odb::mysql::database (...));

transaction t (db->begin ());
schema_catalog::create_schema (*db);
t.commit ();
```

# Custom Schema

- Map classes to tables
- Map members to columns
- Map C++ types to database types

# Legacy Bug Database

```
CREATE TABLE Bugs (
  BugId INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  BugStatus INT NOT NULL,
  Summary VARCHAR (128) NOT NULL,
  Descr VARCHAR (1024) NOT NULL)
```

# Legacy Bug Database

```
#pragma db object table("Bugs")
class bug
{
  ...

  #pragma db id auto column("BugId")
  unsigned int id_;

  #pragma db column("BugStatus") type("INT")
  status status_;

  #pragma db column("Summary") type("VARCHAR (128)")
  std::string summary_;

  #pragma db column("Descr") type("VARCHAR (1024)")
  std::string description_;
};
```

# Mapping Placement

```
class bug
{
  ...

  unsigned int id_;
  status status_;
  std::string summary_;
  std::string description_;
};

#pragma db object(bug) table(”Bugs”)
#pragma db member(bug::id_) id auto column(”BugId”)
...
```

# Mapping Placement

```
// bug.hxx

class bug
{
  ...

  unsigned int id_;
  status status_;
  std::string summary_;
  std::string description_;
};

#include ”bug-mapping.hxx”

// bug-mapping.hxx

#pragma db object(bug) table(”Bugs”)
#pragma db member(bug::id_) id auto column(”BugId”)
...
```

# Mapping Placement

```
// bug.hxx

class bug
{
  ...

  unsigned int id_;
  status status_;
  std::string summary_;
  std::string description_;
};

// bug-mapping.hxx

#pragma db object(bug) table("Bugs")
#pragma db member(bug::id_) id auto column("BugId")
...

odb --odb-epilogue-file bug-mapping.hxx bug.hxx
```

# Summary

*ODB can rival Hibernate for Java and EF/NHibernate for C#.*

- Open-source
- Cross-platform
- Cross-database
- Well documented

# Future

- Support more libraries from Boost
- Support more databases
- SQL to C++ compiler
- Optimistic concurrency
- Database schema evolution and versioning

# Resources

- ODB home page
  - www.codesynthesis.com/products/odb/
- ODB manual
  - www.codesynthesis.com/products/odb/doc/manual.xhtml
- ODB mailing lists
  - www.codesynthesis.com/products/odb/mailing-lists.xhtml
- A Sense of Design
  - www.codesynthesis.com/~boris/blog/

# Questions

?