November 17-20, 2008, Santa Clara Marriott, Santa Clara, CA

cmpi-bindings
Compiler-free provider development

Klaus Kämpf
<kkaempf@suse.de>

Debunking myths

Myth #1

Writing CIM providers is hard

Debunking myths

Myth #2

You need to code in C or C++

MANAGEMENT DEVELOPERS CONFERENCE

Debunking myths

Myth #3

Half of the code is 'glue'

cmpi-bindings to the rescue !

cmpi-bindings

Use your favorite scripting language

(Look Ma, no compiler !)

cmpi-bindings

Faster development

(Edit-Run vs. Edit-Compile-Link-Run)

cmpi-bindings

Drastically reduce code size

(Process Provider: C++ ~3600 loc, Python ~800 loc
File System Provider: C++ 2900 loc, Python 440 loc)

Motivation

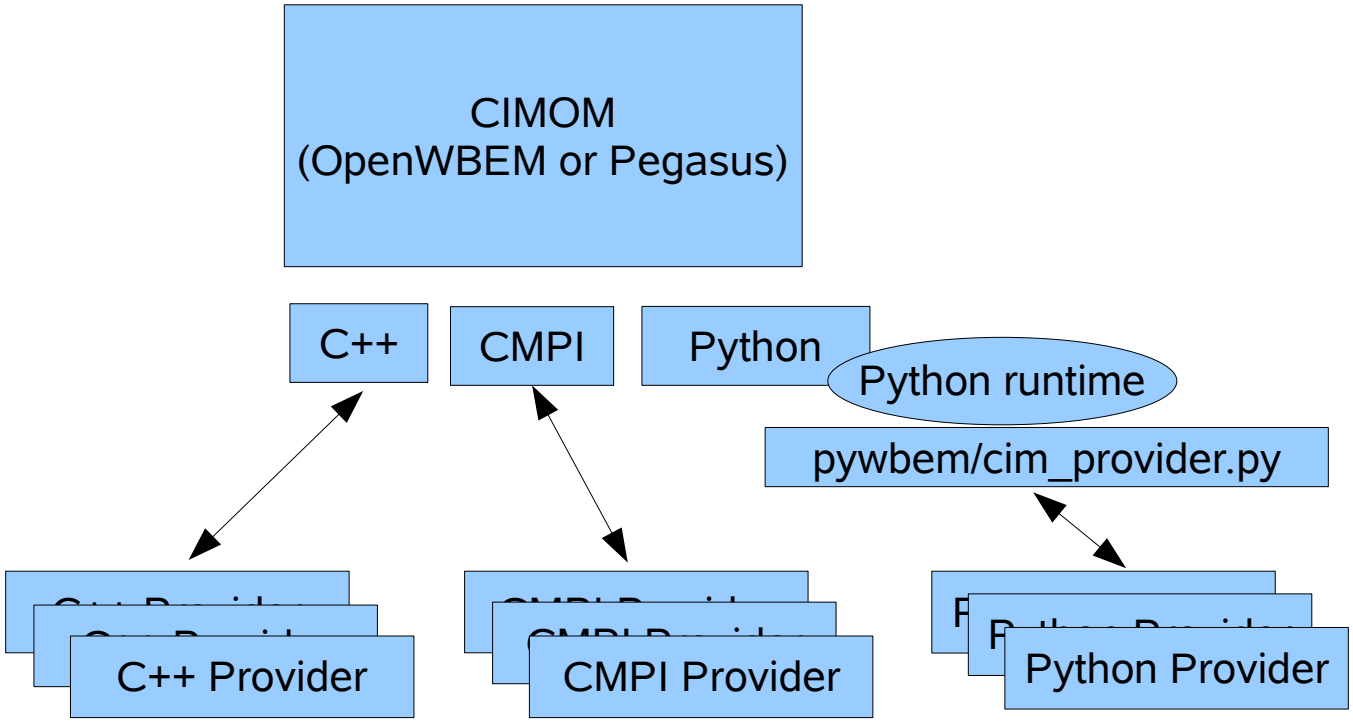
- Make a developers life easier
- Ease debugging
- Use the tools best fitted to the task
- Let developers focus on instrumentation
- Leverage dynamic scripting languages
- Portability

Earlier attempts

- cmpi-perl
  - Part of sblim (sblim.sourceforge.net)
  - Limited functionality
- pywbem
  - Part of omc-project (www.omc-project.org)
  - non-CMPI (v1)

pywbem (v1)

First attempt on scripting providers



pywbem (v1)

- Pros
  - Scripting language
  - Reduced code size
  - Leverage Python environment
- Cons
  - Binary interface to CIMOM
  - Python only
  - Limited object-orientation
  - Manually created bindings

What if ...

- CMPI interface
  - Make it truly CIMOM agnostic
- More languages
  - There's more than C, C++ and Python
- Generated bindings
  - Lesser code
  - Easy adoption
  - All languages profit

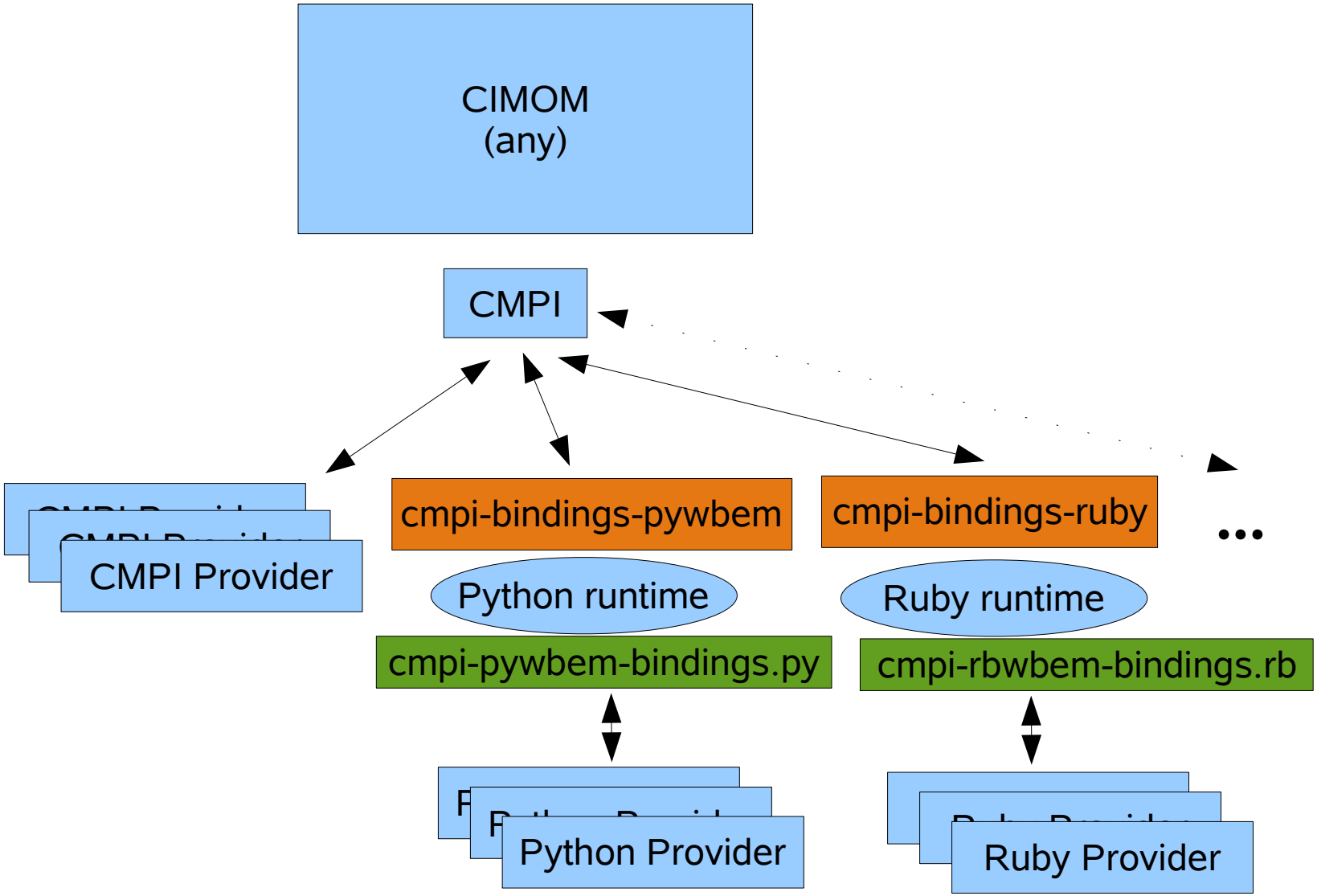
cmpi-bindings

Design goals

- CIMOM neutral
  - CMPI provider interface
- Support most popular scripting languages
  - Python, Ruby, Perl, ...
- Object orientation
  - Reduce parameters
  - Leverage exceptions
- Code similarity
  - Learn from looking at other code

MANAGEMENT DEVELOPERS CONFERENCE

cmpi-bindings



How it was done

- Use a code generator (SWIG)
- Reuse of generic code
- Similar 'look&feel' across languages
- Small language dependent layer

SWIG

Simplified Wrapper and Interface Generator

SWIG

SWIG is an **interface compiler** that **connects** programs written in **C and C++ with scripting languages** such as Perl, Python, Ruby, and more.

SWIG: Motivation

- Building more powerful C/C++ programs
- Portability
- Make C libraries 'object oriented'
- Rapid prototyping and debugging
- Systems integration
- Construction of scripting language extension modules

SWIG: About

- Homepage: http://www.swig.org
- Available for
  - Linux
  - Unix (AIX, HP-UX, Solaris, ...)
  - Macintosh OS-X/Darwin
  - Windows 95/98/NT/2000/XP/Vista

- History

  Initially started in July, 1995 at Los Alamos National Laboratory.

  First alpha release: February, 1996.

  Latest release: April 7, 2008. SWIG-1.3.35

SWIG: Languages

Allegro Common Lisp

CFFI (Common Lisp)

CLisp

Octave

Chicken (Scheme)

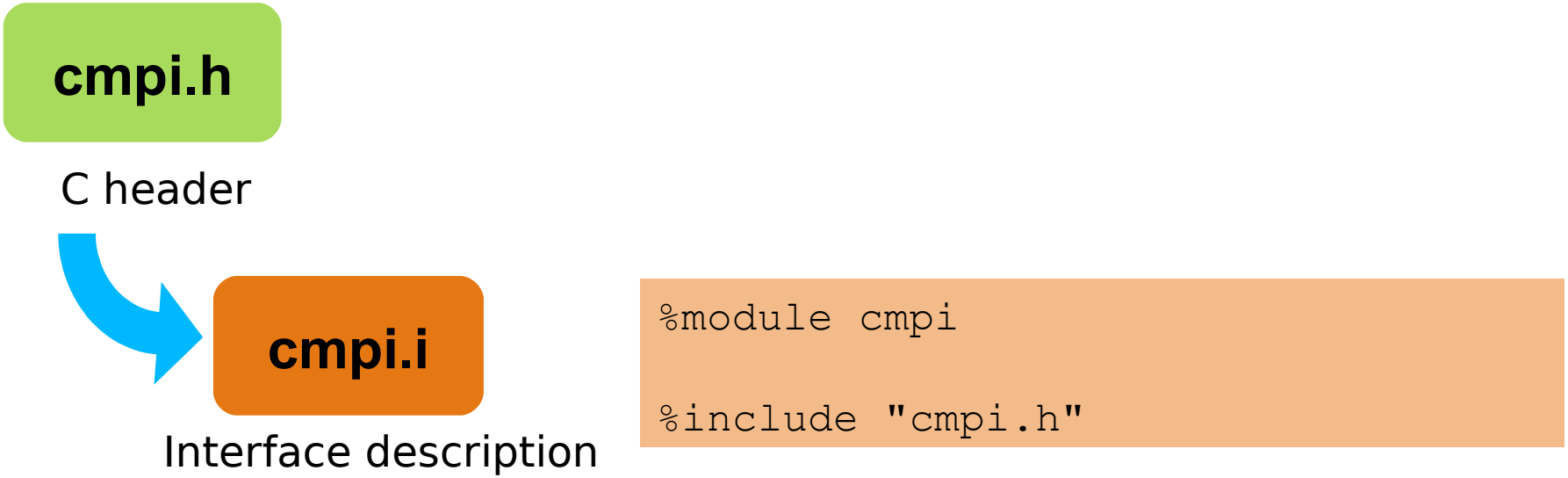
MzScheme

SWIG – How does it work ?

cmpi.h

C header

SWIG – How does it work ?

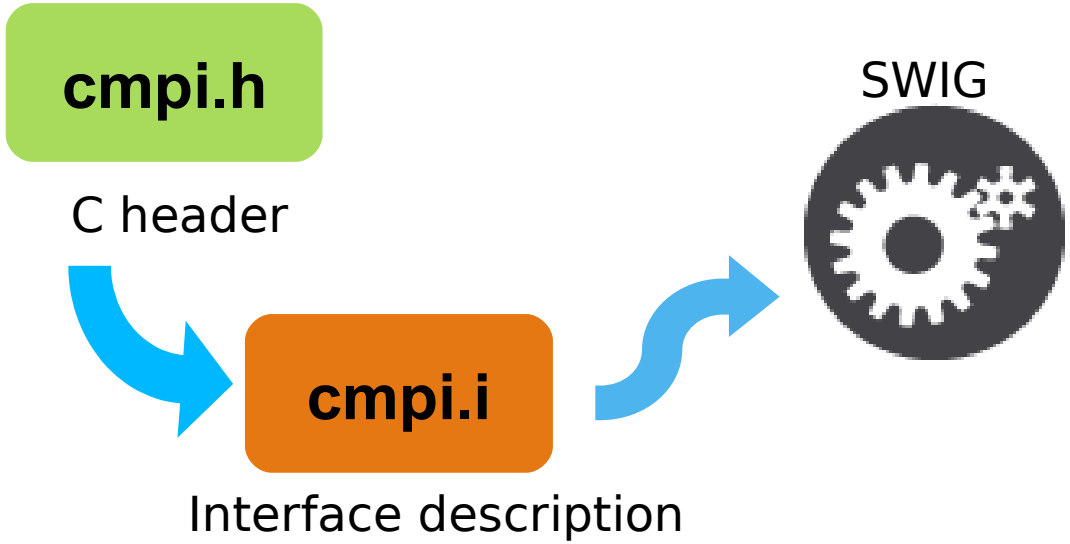



```
%module cmpi

%include "cmpi.h"
```

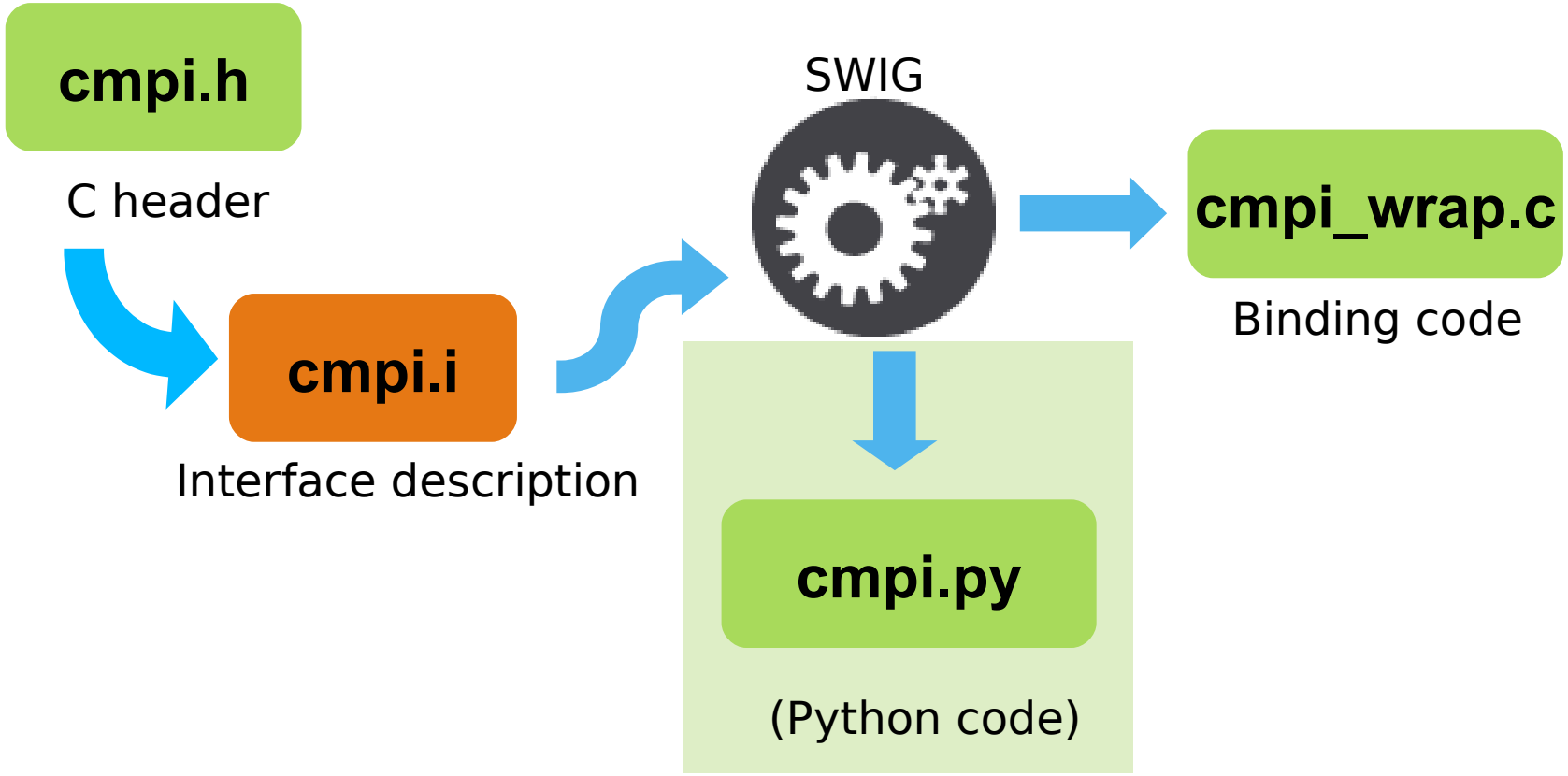
MANAGEMENT DEVELOPERS CONFERENCE

SWIG – How does it work ?



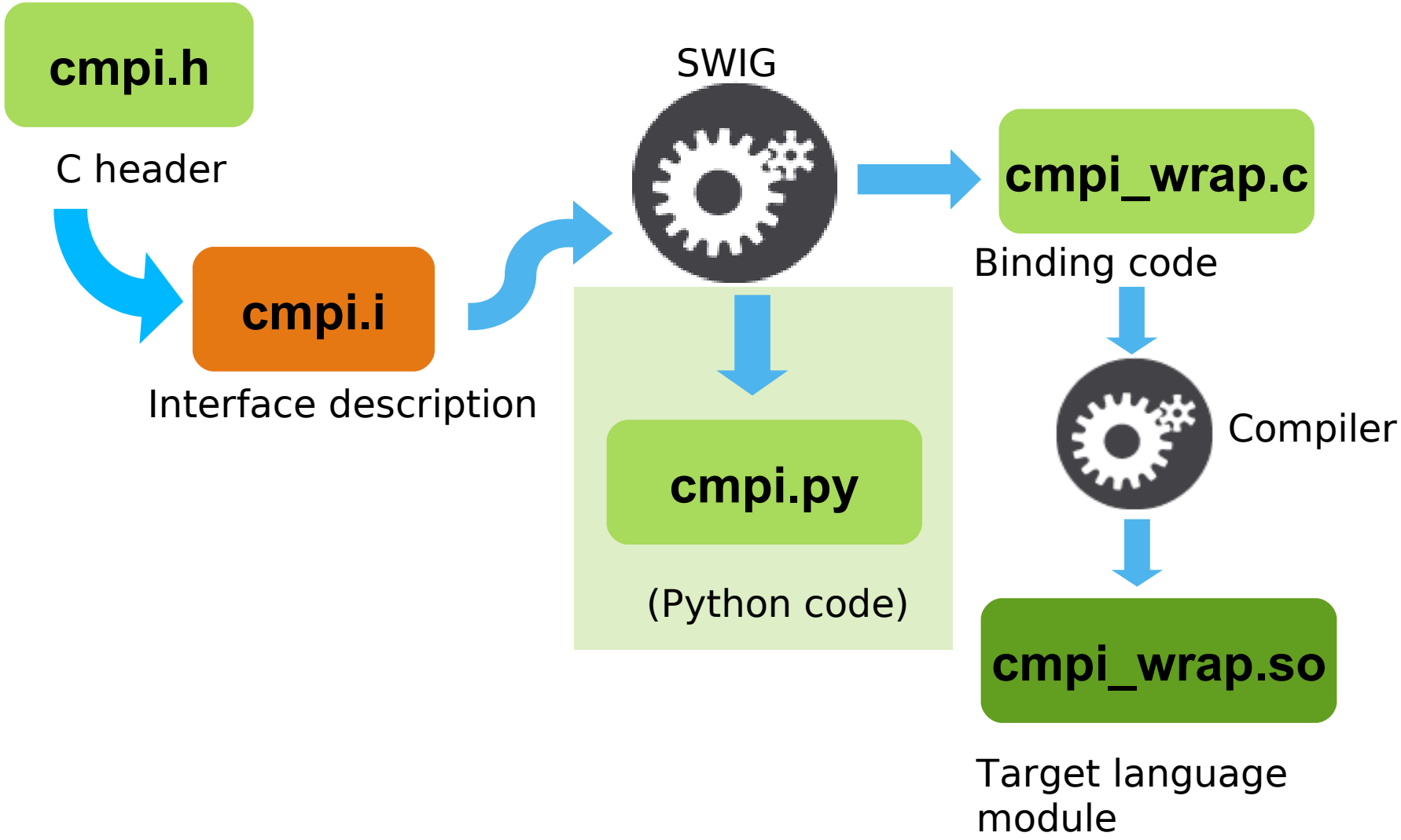
MANAGEMENT DEVELOPERS CONFERENCE

SWIG – How does it work ?



MANAGEMENT DEVELOPERS CONFERENCE

SWIG – How does it work ?



SWIG - Usage

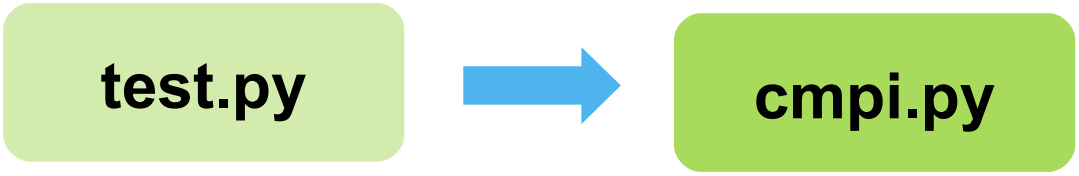
Example: Python

test.py

```
import cmpi
```

SWIG - Usage

Example: Python




```
import cmpi
```

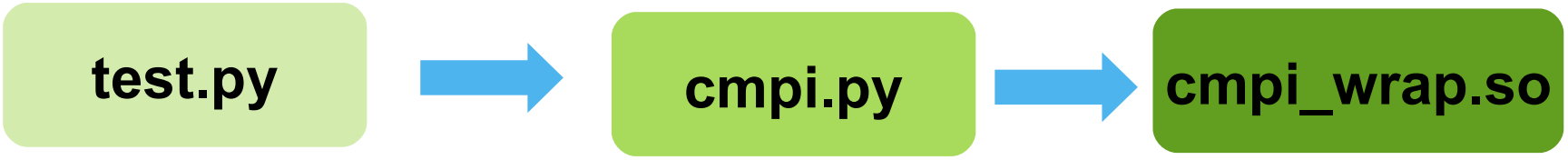
SWIG - Usage

Example: Python


```
import cmpi
```

SWIG - Usage

Example: Python




```
import cmpi
data = cmpi.CMPIData()
data.type = cmpi.CMPI_uint8
data.state = 0
data.value.uint8 = 42
```

Result

- Target language module
- Access to CMPI data structures
- Access to CMPI manipulation functions
- Data wrappers (C <-> target language)
- Thread safe

Code reduction

- Example: Property access
  - C

```
CMGetProperty(instance, "Username", &st);
```

  - Python

```
instance['Username']
```

- Object oriented CMPI programming
- Exceptions

MANAGEMENT DEVELOPERS CONFERENCE

Building bridges

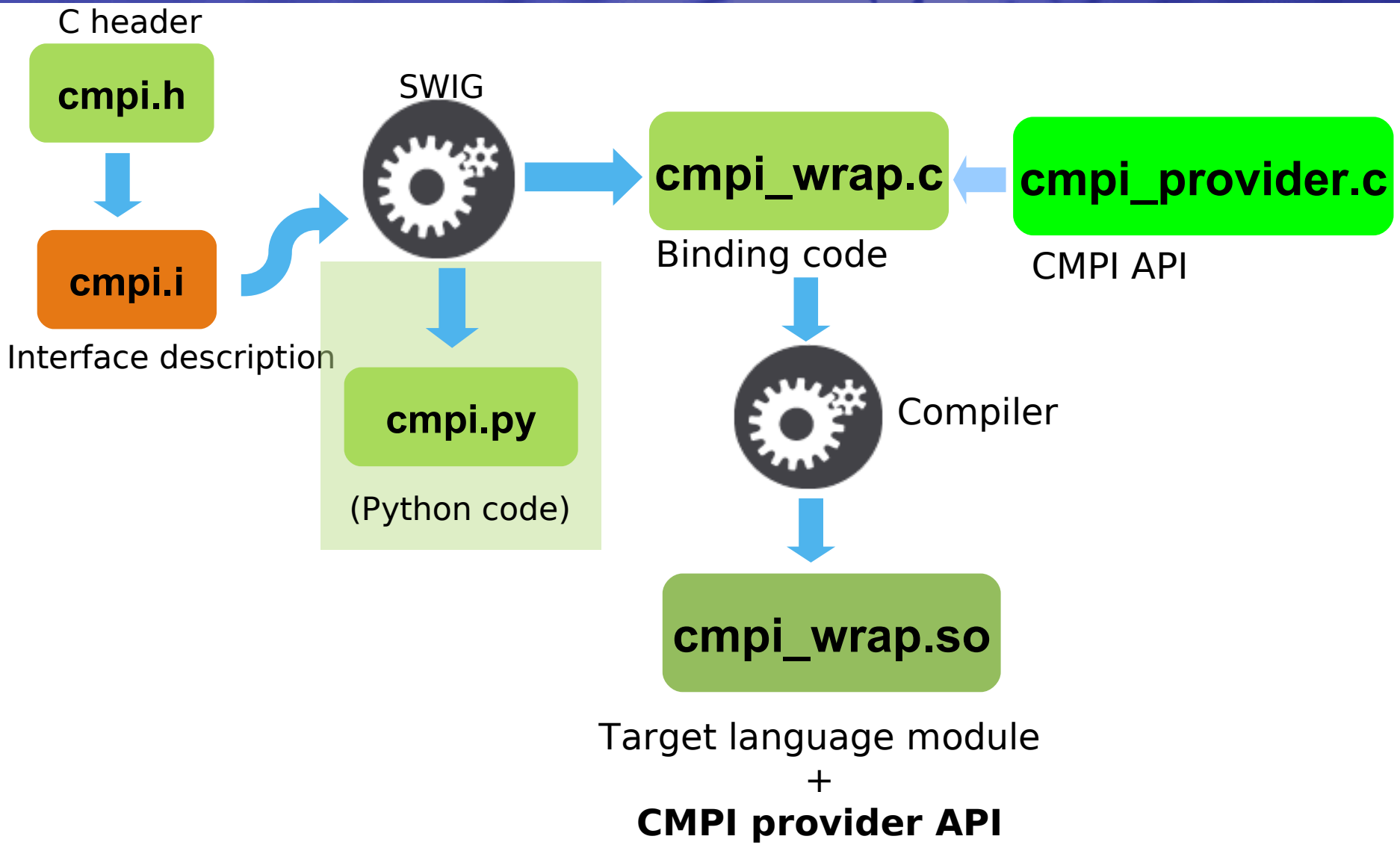
- SWIG gives access to CMPI data structures
  - Target language interface
- Missing: CMPI provider interface
  - Access to 'broker'
- Add glue code
- End result: Plugin with two interfaces
  - Target language extension
  - CMPI provider

cmpi_provider.c

- Manually crafted CMPI provider interface
- Implements the full CMPI API
  - (Instance, Method, Association, Indication)
- Target language agnostic
- Converts C data to target language
- Calls target language
- Status handling

MANAGEMENT DEVELOPERS CONFERENCE

cmpi_provider + SWIG



cmpi_provider.c: Code example

```
static CMPIStatus
EnumInstanceNames(CMPIInstanceMI * self,
        const CMPIContext * context,
        const CMPIResult * result,
        const CMPIObjectPath * reference)
{
 /* ... */
 _context = SWIG_NewPointerObj(context, SWIGTYPE_p__CMPIContext, 0);
 _result = SWIG_NewPointerObj(result, SWIGTYPE_p__CMPIResult, 0);
 _reference = SWIG_NewPointerObj(reference,SWIGTYPE_p__CMPIObjectPath,0);

 TargetCall((ProviderMIHandle*)self->hdl, &st,"enum_instance_names",
    3, _context, _result, _reference);
 return st;
}
```

target_$lang.c

- Target language specific layer
- Very thin
  - TargetInitialize(...)
  - TargetCall(...)
  - TargetCleanup(...)
- Loads/Unloads target interpreter
- Loads provider implementation
- Calls provider implementation

Code size

- cmpi_provider.c: 1225 lines
- target_python.c: 397 lines
- target_ruby.c: 290 lines

- Easy to maintain
- Easy to extend

Implementing EnumInstanceNames

- C snippet

```
static CMPIStatus
EnumInstanceNames(CMPIInstanceMI * self,
        const CMPIContext * context,
        const CMPIResult * result,
        const CMPIObjectPath * reference)
{
  char * namespace =
    CMGetCharPtr(CMGetNameSpace(reference, NULL));
```

- Ruby counterpart

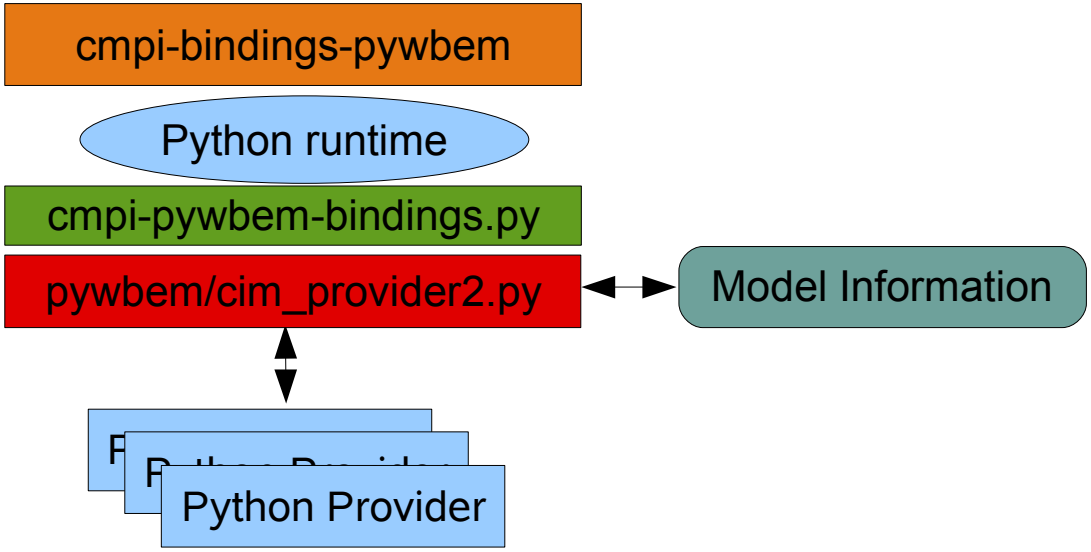
```
def enum_instance_names context, result, reference
  namespace = context.namespace
```

What goes where ?

- Provider module
  - Binary, dynamically loaded by CIMOM
  - Lives in /usr/lib/cmpi/lib<provider>.so
    - `/usr/lib/cmpi/libpyCmpiProvider.so`
- Provider name
  - Script language file
  - Lives in /usr/lib/{py,rb,pl}cim
    - `/usr/lib/pycim/Py_UnixProcessProvider.py`
- Provider class name
  - Class within script language file

pywbem(v2) + cmpi-bindings

- Python (pywbem v2) as first choice
  - provides model information
  - trivial conversion of existing pywbem providers



Status

- Production ready
  - Ships with SUSE Enterprise Server 11
- Python as first choice
  - because of pywbem
  - Ruby runner-up, then Perl
- Need other languages ?
  - Please give feedback (or code)
- www.omc-project.org

Outlook

- Verify Ruby interface
  - Implement Ruby provider
- Finish Perl interface
  - Needs help
- Provide model information
  - Language agnostic
- Generic provider registration

References

- OMC-Project
  - http://www.omc-project.org
- cmpi-bindings
  - https://omc.svn.sourceforge.net/svnroot/omc/cmpi-bindings/trunk
- pywbem
  - https://pywbem.svn.sourceforge.net/svnroot/pywbem/pywbem

MANAGEMENT DEVELOPERS CONFERENCE

Thank you !

Questions ?