



Generating language bindings for C/C++ libraries

Klaus Kämpf

<kkaempf@suse.de>

Novell.[®]

What and why ?

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

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

About SWIG

- Homepage: <http://www.swig.org>
- # `zypper` in `swig`
- 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
- Active development
 - 3-4 releases per year

Supported languages

Allegro Common Lisp



CLisp

CFFI (Common Lisp)



(guile)



Octave



Ruby

A Programmer's Best Friend

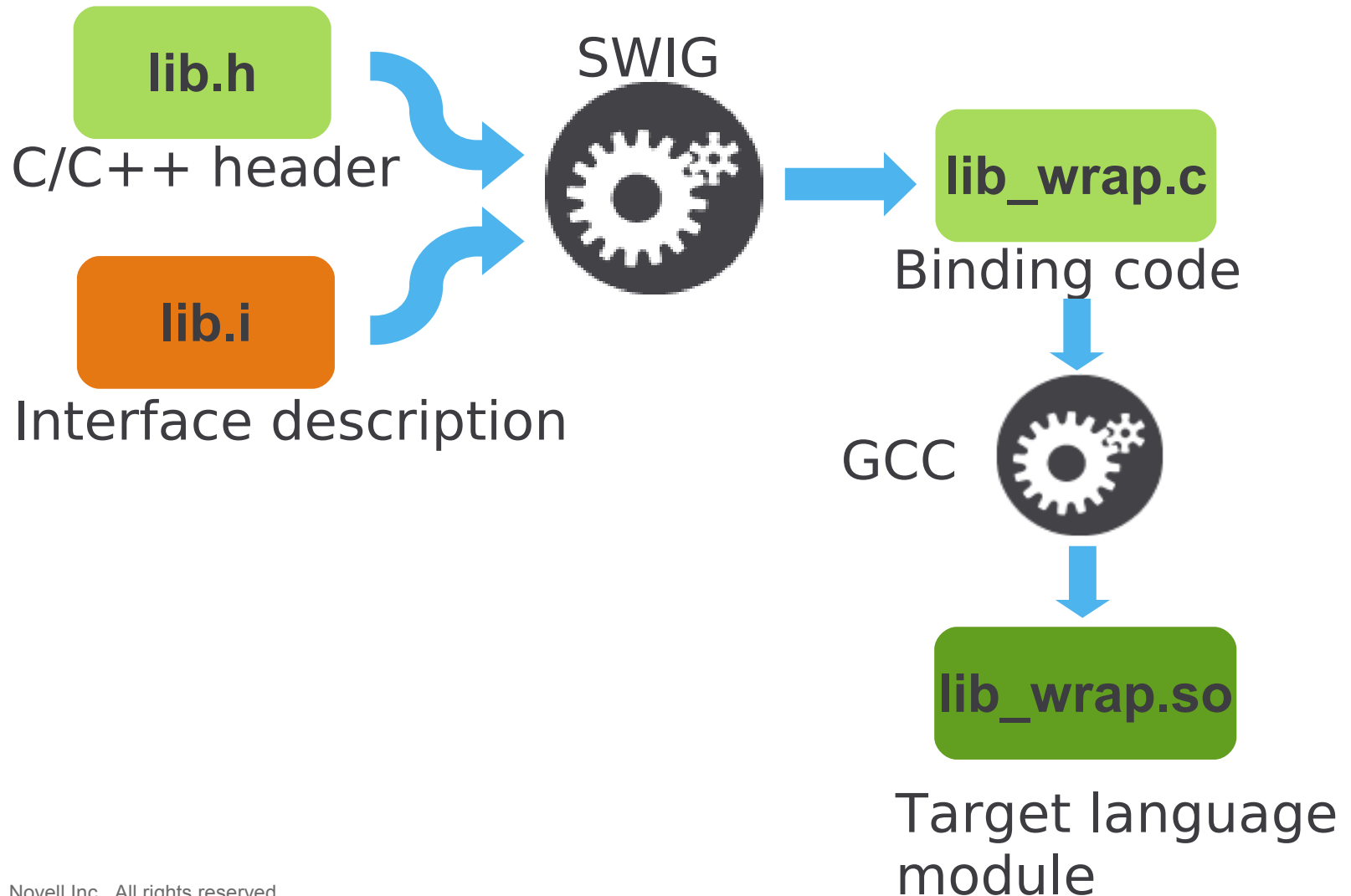


Chicken
(Scheme)

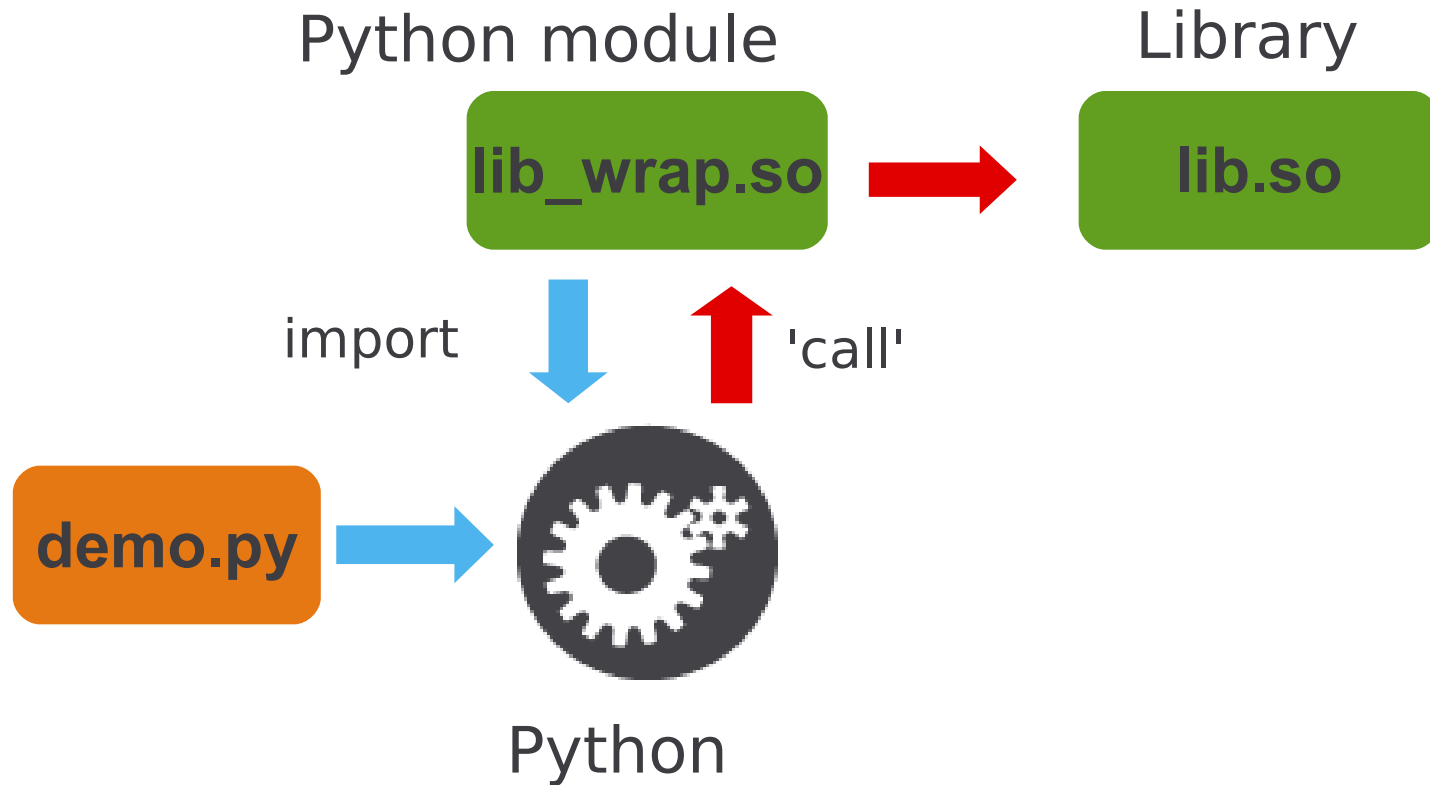
MzScheme



How SWIG works



How SWIG works (cont.)



Using SWIG

Example interface description

```
# Trivial example
%module example
%{
#include "satsolver/solver.h"
%}

#include satsolver/solver.h
```


Running SWIG

- Generating

```
swig -ruby -I/usr/include example.i
```

- Compiling

```
gcc -fPIC -I /usr/lib64/ruby/1.8/x86_64-linux -c example_wrap.c
```

- Linking

```
gcc -shared example_wrap.o -lsatsolver -o example.so
```

- Running

```
irb
```

```
irb(main):001:0> require "example"
```

```
=> true
```

```
irb(main):002:0> s = Example::Solver.new
```

```
=> #<Example::Solver:0x7ffd300d4de8>
```

Structure of interface descriptions

# Trivial example	Comment
%module example	Namespace
{	
#include "satsolver/solver.h"	C/C++ code
}	
%include satsolver/solver.h	Declarations

- C syntax, no C compiler
- Only minimal syntax checking

What does SWIG do for you ?

- Namespace
- Constants
- Type conversion
 - For simple types (int, float, char *, enum)
- Wraps complex types
 - Pointers to structs and classes
- Exposes functions
- Memory management
 - Constructors, destructors

Example (Python)

(taken from libyui-bindings)

YaST2/yui/YUI.h

```
class YUI
{
...
    static YWidgetFactory *
    widgetFactory();
...
}
```

demo.py

```
import yui
factory = yui.YUI.widgetFactory()
dialog = factory.createPopupDialog()
vbox = factory.createVBox( dialog )
factory.createLabel( vbox, "Hello,
World!" )
```

yui.i

```
%module yui
%{
#include "YaST2/yui/YUI.h"
%}
#include YUI.h
```

Now how does it look like in ...

Ruby

```
require 'yui'  
  
factory = Yui::YUI::widget_factory  
dialog = factory.create_popup_dialog  
vbox = factory.create_vbox dialog  
factory.create_label vbox, "Hello, World!"
```

Perl

```
use yui;  
  
my $factory = yui::YUI::widgetFactory;  
my $dialog = $factory->createPopupDialog;  
my $vbox = $factory->createVBox( $dialog );  
$factory->createLabel( $vbox, "Hello, World!" );
```

Things to watch out for

- Function names (target language conventions)

```
factory.create_popup_dialog
```

```
$factory->createPopupDialog;
```

- Comparing objects

SWIG *wraps* pointers to structs/classes, resulting in target languages objects (Python: `PyObject*`, Ruby: `VALUE`)

'a == b' compares `PyObject*` (resp. `VALUE`), not the wrapped C++ object pointer

- Object ownership

No explicit 'free' in e.g. Ruby and Python

Controlling the bindings

Exposure

- Swig recognizes C/C++ declarations

'struct' or 'class'

functions

- Hiding elements

```
%ignore solver::nouupdate;
```

```
%include "satsolver/solver.h"
```

- Hiding everything

```
typedef struct solver {} Solver;
```

```
%extend Solver {
```

```
...
```


Memory management

- Complex types (struct/class) as pointers
- SWIG runs constructor ('malloc (sizeof struct)')
- Might not be useful

```
%nodefault solver;
```

- Explicit constructor/destructor

```
%extend Solver {  
    Solver( Pool *pool, Repo *installed = NULL )  
    { return solver_create( pool, installed ); }  
    ~Solver()  
    { solver_free( $self ); }  
}
```

Making C object-oriented

- Swig maps function calls 1:1, Ok for C++, bad for C

```
void solver_solve(Solver *solv, Queue *job);
```

(Ruby)

```
solver = Solver.new
```

```
solver_solve solver, job # Bad
```

```
solver.solve job # Good
```

- The power of %extend

```
%extend Solver {  
  int solve( Queue *job )  
  {  
    solver_solve( $self, job);  
    return $self->problems.count == 0;  
  }  
}
```

Multiple target languages

- .i files are generic
- The target language is a SWIG runtime parameter

```
swig -ruby bindings.i
```

- Use `#if defined(SWIG<lang>)`

```
#if defined (SWIGRUBY)
```

```
...
```

```
#endif
```

Useful commands

- Renaming

```
%rename("to_s") asString();  
%rename("name=") set_name(const char *name);  
%rename("empty?") empty();
```

- Aliasing

```
%alias get "[]";
```

- Constants

```
%constant int Script = C_CONSTANT;
```

- Defines

```
%define YUILogComponent "bindings"  
%enddef  
  
%define %macro(PARAMETER)
```

...

Type conversions

- SWIG has default conversions for most types
- Look at the SWIG 'library'

```
/usr/share/swig/<version>
```

```
%include "carray.i"
```

- Typemaps

```
#if defined(SWIGRUBY)
```

```
    %typemap(in) (int bflag) {
```

```
        $1 = RTEST( $input );
```

```
    }
```

```
    %typemap(out) int problems_found
```

```
        "$result = ($1 != 0) ? Qtrue : Qfalse;";
```

```
    %rename("problems?") problems_found();
```

```
#endif
```

Target specifics

- Bypassing SWIG type conversion
- Use target-specific types

Ruby: VALUE

Python: PyObject *

- Example

```
%rename( "attr?" ) attr_exists( VALUE attrname );  
VALUE attr_exists( VALUE attrname )  
{  
    ...  
}
```

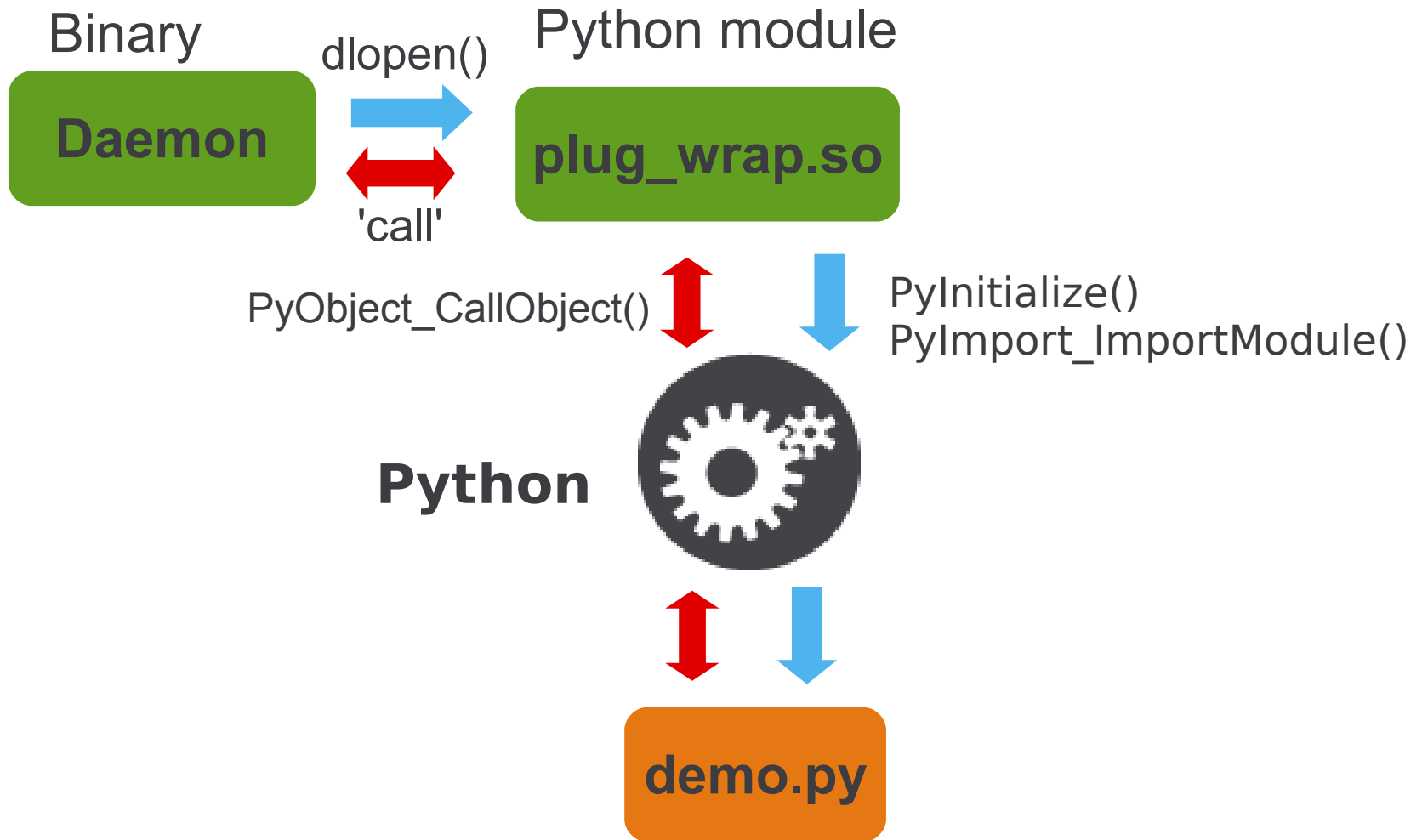
Generating Documentation

- SWIG can generate target-specific documentation
e.g. rdoc for Ruby, pydoc for Python
- Enable with `%feature("autodoc", "1");`
- Converts C-style comments in .i files
- Needs fixing ...

The background of the slide is a solid blue color with a pattern of diagonal lines in various shades of blue, creating a sense of motion or depth. The lines are more densely packed on the right side and become more sparse towards the left.

Inversion of control

Inversion of control



Wrap up / Lessons learned

- SWIG is a tool, use it wisely
- Take the (script language) programmers view
How should it look in Python/Ruby/Perl/... ?
- Tweak the bindings, not the target language
- Look at other SWIG code
- SWIG is very well documented
But not without bugs ...
- Memory ownership is tricky

Links for inspiration

- C++ Library

libyui-bindings (YaST user interface)

<http://svn.opensuse.org/svn/yast/trunk/libyui-bindings>

- C Library

Sat-solver (package dependency resolver)

<http://svn.opensuse.org/svn/zypp/trunk/sat-solver/bindings>

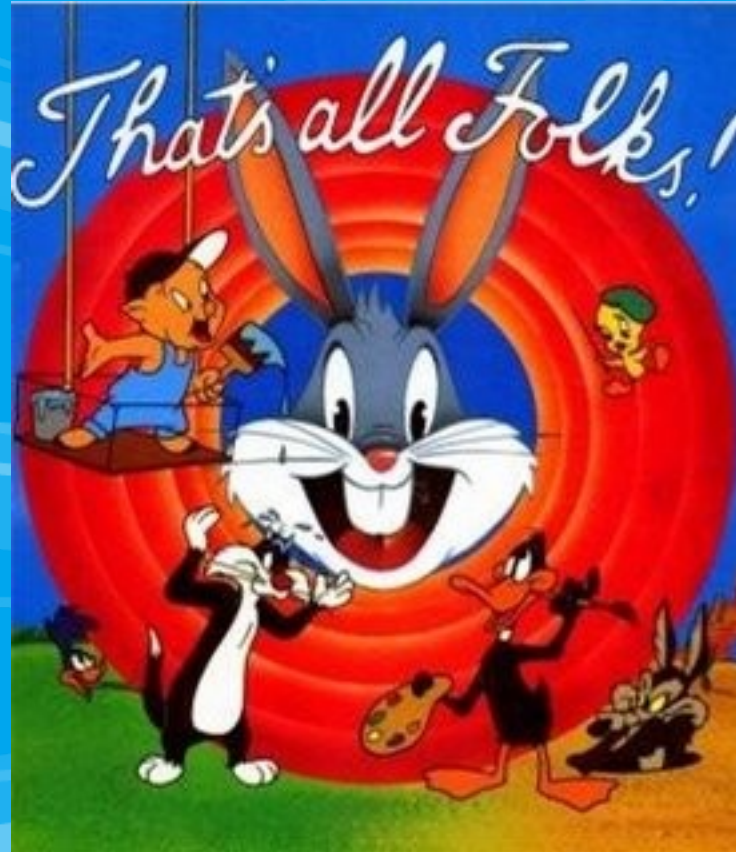
openwsman (Web Services for Management protocol)

<http://www.openwsman.org/trac/browser/openwsman/trunk/bindings>

- Inversion of control

cmpi-bindings (CIM Provider interface)

<http://omc.svn.sourceforge.net/viewvc/omc/cmpi-bindings>



Novell®

Unpublished Work of Novell, Inc. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary, and trade secret information of Novell, Inc. Access to this work is restricted to Novell employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of Novell, Inc. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Novell, Inc. makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for Novell products remains at the sole discretion of Novell. Further, Novell, Inc. reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All Novell marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.

