Enforcing type-safe linking using inter-package relationships

Mehdi Dogguy Stéphane Glondu Sylvain Le Gall Stefano Zacchiroli

{mehdi,glondu,gildor,zack}@debian.org

16/04/2010

OCaml Meeting — Paris, France

What's our (i.e. distro editor) problem?

this:

```
$ ocamlc -o main foo.cmo bar.cmo main.cmo
File "_none_", line 1, characters 0-1:
Error:  Files main.cmo and bar.cmo
make inconsistent assumptions over interface Bar
```

should *really* imply:

this is your (i.e. developer) own mistake!

(no, it ~~is~~ was not the case)

What's the big deal about OCaml(-like) linking?

With system-level language (such as C) linking:

- During the linking phase, some checks are performed upon compilation units to see if the different components fit well together: (essentially: referential integrity)

  Definition (Linkability)

  1. All symbols are resolved: $\bigcup_i \mathcal{R}(u_i) \subseteq \bigcup_i \mathcal{A}(u_i)$
  2. Avoid multiple definitions: $\forall i\, j, \quad i \neq j \rightarrow \mathcal{A}(u_i) \cap \mathcal{A}(u_j) = \emptyset$

- With some "disclipline", backward compatibility is taken for granted!

```
Package: random-ocaml-app
Depends: ..., random-lib (>= 1.2-3),...
```

With statically typed programming languages . . .

. . . such as OCaml and Haskell

Some additional link-time, type-aware checks come into play

- cross-module compatibility
- ABI cannot change between compile-time and link-time

```
$ cat foo.ml
let f x = x
$ ocamlc -c foo.ml -o foo.cmo
$ ocamlobjinfo foo.cmo
File foo.cmo
Unit name:  Foo
Interfaces imported:
88cb1505c8bdf9a4dcd2cdf3452732b4 Pervasives
9a3f59bb6c948951f5d08752689d4fb7 Foo
Uses unsafe features:  no
```

Do we still have backward compatibility?

```
$ cat main.ml
let _ = print_int (Foo.f 1)
$ ocamlc -c main.ml -o main.cmo
$ ocamlc foo.cmo main.cmo -o main
$ cat foo.ml
let f x = x
let new_f () = ()
$ ocamlc -c foo.ml -o foo.cmo
$ ocamlc foo.cmo main.cmo -o main
File "_none_", line 1, characters 0-1:
Error:  Files main.cmo and foo.cmo
make inconsistent assumptions over interface Foo
```

How can we enforce type-safe linking using inter-package relationships?

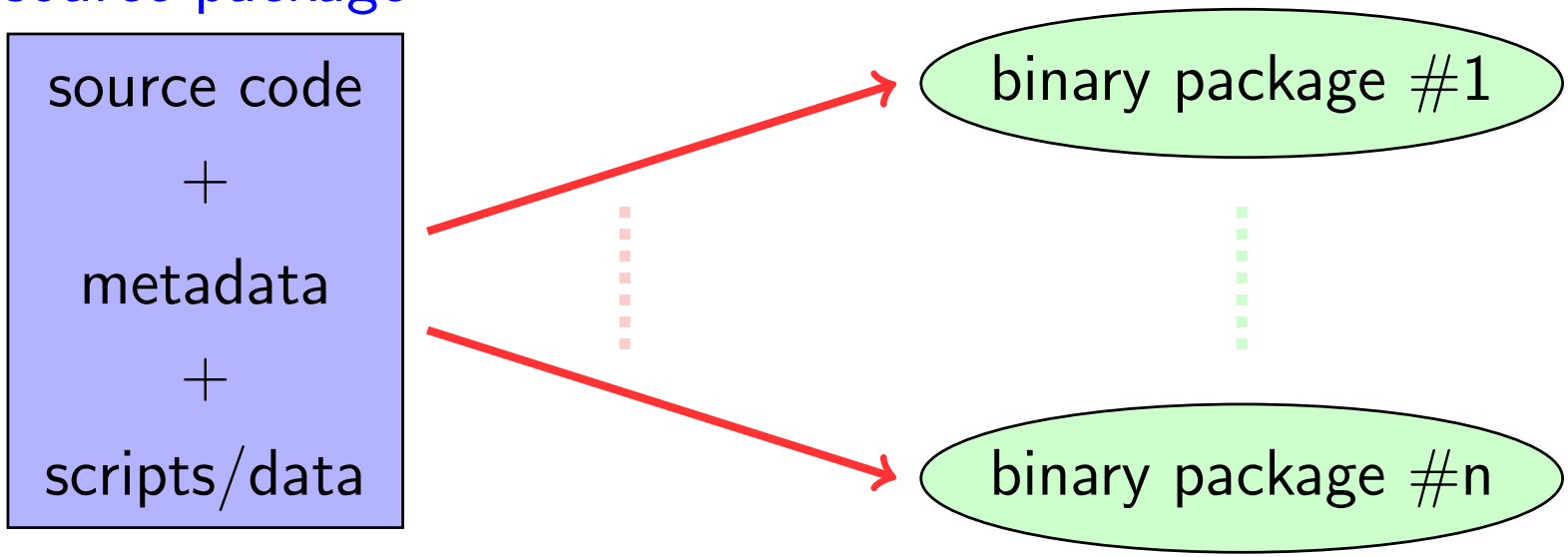
Outline

1 Packaging basics

Packages
Auto-(re)building
Dependency Inference

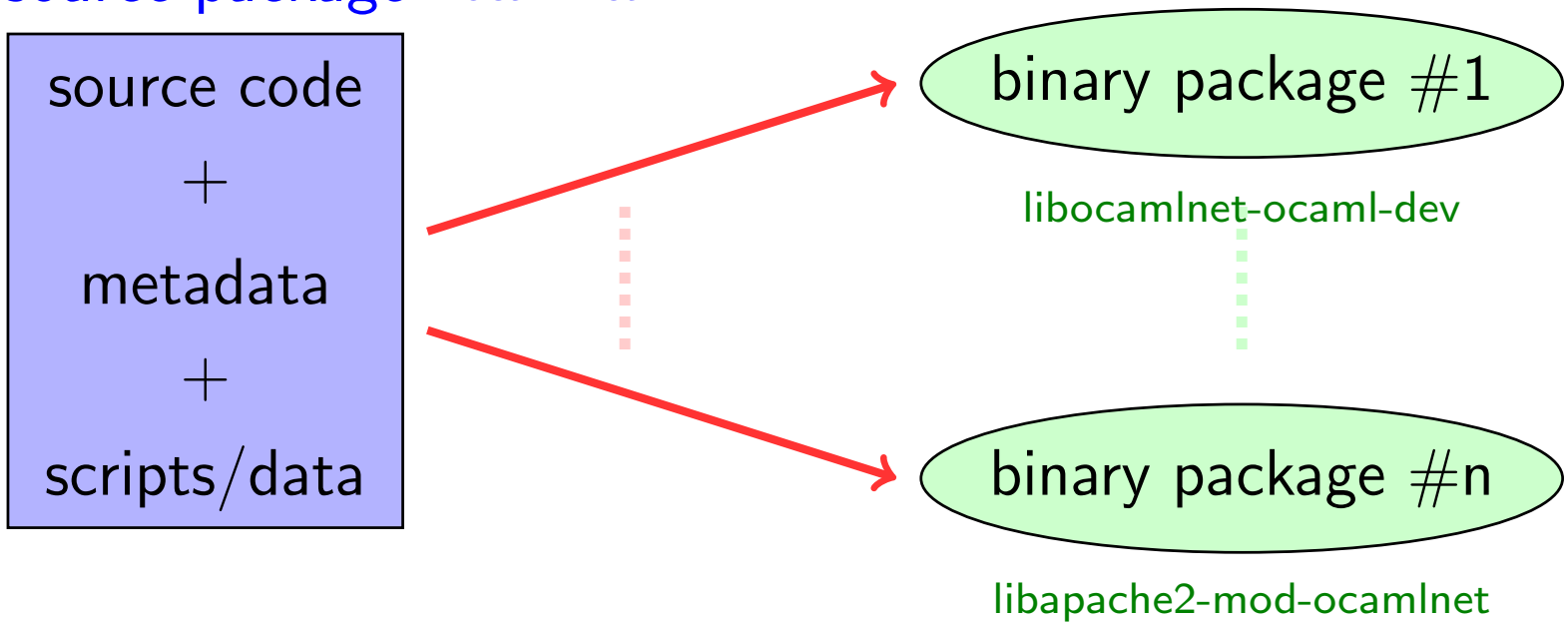
Packages

source package



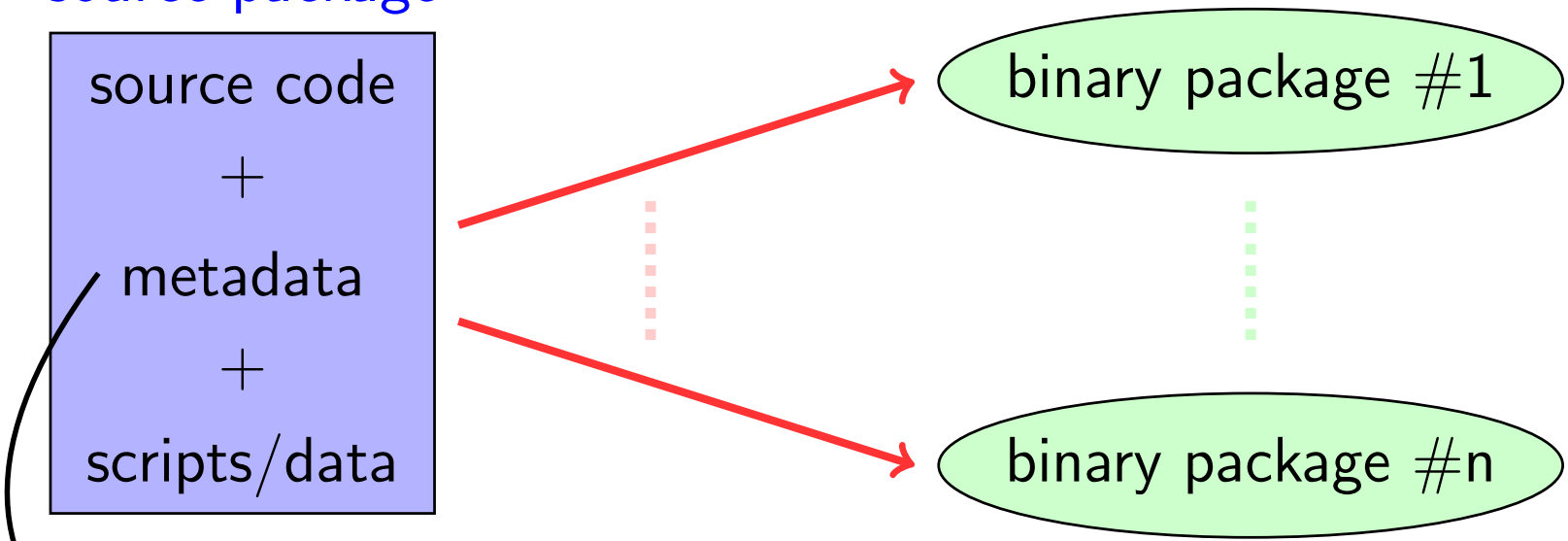
Packages

source package ocamlnet



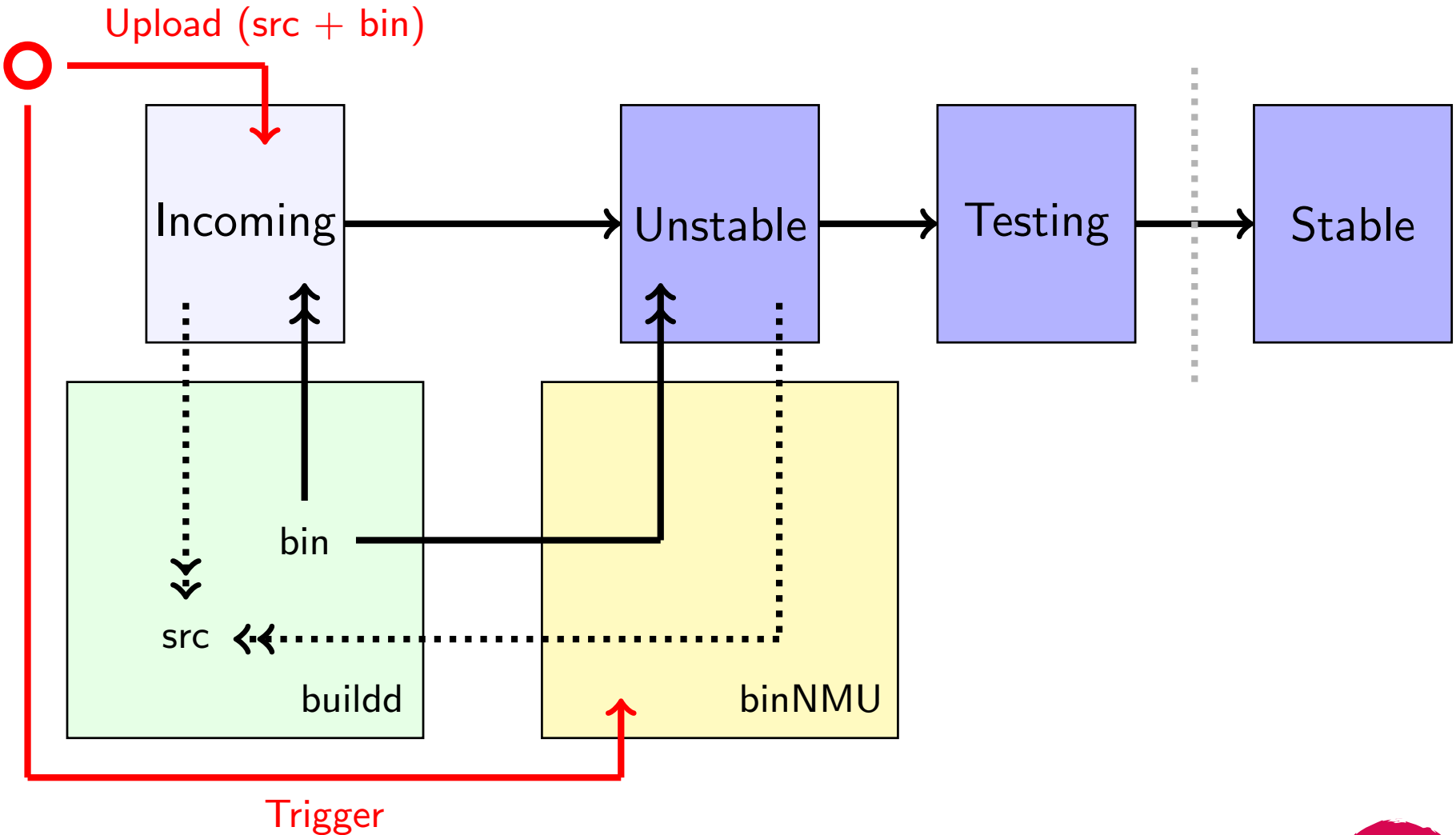
Packages

source package




```
[...]
Package: libocamlnet-ocaml-dev
Version: 2.2.9-3
Depends: ocaml-nox-3.10.2, ocaml-findlib, libocamlnet-ocaml (= 2.2.9-3),
         libpcre-ocaml-dev (≥ 5.11.1), libcryptgps-ocaml-dev (≥ 0.2.1)
Provides: libequeue-ocaml-dev, libnetclient-ocaml-dev, librpc-ocaml-dev
Conflicts: libequeue-ocaml-dev (<< 2.2.3-1),
      libnetclient-ocaml-dev (<< 2.2.3-1), librpc-ocaml-dev (<< 2.2.3-1)
Description: OCaml application-level Internet libraries - core libraries
[...]
```

Auto-building and binNMUs (in Debian)



binNMU = binary non-maintainer upload

buildd = build daemon

Dependency Inference

```
Package: mldonkey-server
Depends: adduser, mime-support, ucf,
 ${shlibs:Depends},
 ${misc:Depends}
```

```
Package: mldonkey-server
Depends: adduser, mime-support, ucf,
 libbz2-1.0, libc6 (>= 2.3.2), libjpeg62,
 libfreetype6 (>= 2.2.1), libgcc1 (>= 1:4.1.1),
 zlib1g (>= 1:1.1.4), libpng12-0 (>= 1.2.13-4),
 libstdc++6 (>= 4.2.1),
 libgd2-noxpm(>= 2.0.36~rc1~dfsg)
  | libgd2-xpm (>= 2.0.36~rc1~dfsg),
 debconf | debconf-2.0
```

2 Requirements

Requirement 1/4: dependency soundness

Requirement (Dependency soundness)

A package has sound dependencies if and only if its compilation units are linkable with all units shipped by its (transitive) dependencies in any healthy installation.

Requirement 2/4: dependency inference

Requirement (Dependency inference)

Given a source package s and its build-dependencies $B = \{p_1, \ldots, p_n\}$, the dependencies that ensure soundness on all binary packages obtained by building s, should be inferrable on the basis of B and its (transitive) dependencies.

Package: `ocaml-app`
Depends:
`random-util`,
`yet-another-random-util`,
`${shlibs:Depends}`,
`${misc:Depends}`,
`${ocaml:Depends}`

Requirement 3/4: binNMU-safety

Requirement (binNMU-safety)

If a package p has sound dependencies, performing a binNMU on it should not make its dependencies unsound.

Requirement 4/4: "light" dependencies

Requirement (Light dependencies)

All inter-package relationships needed to ensure dependency soundness should be terse, readable, human-manageable.

```
$ rpm -qRp ocaml-ocamlnet*.rpm
ocaml(Arg) = b6513be035dc9c8a458c189cd8841700
ocaml(Array) = 9c9fa5f11e2d6992c427dde4d1168489
ocaml(Bigarray) = fc2b6c88ffd318b9f111abe46ba99902
ocaml(Buffer) = 23af67395823b652b807c4ae0b581211
... snipped 67 more ocaml(*) deps
```

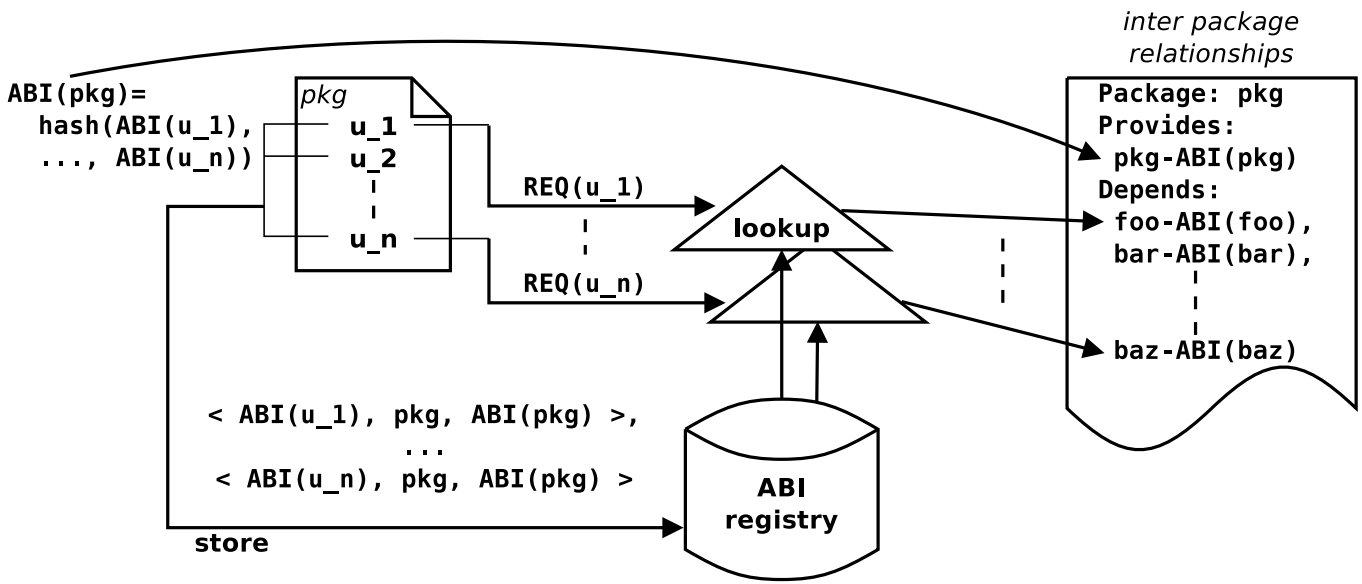
```
$ rpm --provides -qp ocaml-ocamlnet*.rpm
ocaml(Equeue) = 329e036bb2778b249d6763d22407af19
ocaml(Ftp_client) = d36822b105eacef219a2b6e0331ba34b
ocaml(Ftp_data_endpoint) = f279805dc3b7ced5d8554f92e287c889
ocaml(Generate) = 418deddda65b04bdc4d0c6e9fb918d4
... snipped 110 more ocaml(*) virtual packages
```

3 ABI approximation

ABI approximation

After building a package:

1. the hash of the ABI of all units provided by a library is computed and put in the metadata of the resulting binary packages
   - `dh_ocaml`
2. installed units are inspected in order to compute the list of dependencies[1]
   - `ocaml-md5sums` (`ocamlbyteinfo`, `ocamlplugininfo`, ...)



[1] installed packages maintain a (system-wide) registry mapping from units to packages

ABI approximation (example)

Package:
libocamlnet-ocaml-dev
Provides:
libequeue-ocaml-dev,
libnetclient-ocaml-dev,
librpc-ocaml-dev,
${ocaml:Provides}
Depends:
ocaml-findlib,
${ocaml:Depends},
${shlibs:**Depends**},
${misc:**Depends**}

Package:
libocamlnet-ocaml-dev
Provides:
libequeue-ocaml-dev,
libnetclient-ocaml-dev,
librpc-ocaml-dev,
libocamlnet-ocaml-dev-2m6d1
Depends:
ocaml-findlib,
libcryptgps-ocaml-dev-fhk10,
libocamlnet-ocaml-2m6d1,
libpcre-ocaml-dev-g7y84,
ocaml-nox-3.11.2

1% collision probability with $\approx 1'100$ versions

4 Conclusion

Conclusion

Solution	**Soundness**	**Inference**	**binNMU**	**Lightness**
past Debian *status quo*	✗	✗	✓	✓
+ old-style `dh_ocaml`	✗	✓	✓	✓
current Fedora guidelines	✓	✓	✓	✗
ABI evolution tracking	✓	✓	✗	✓
ABI approximation	✓	✓	✓	✓

Limitations

- architecture independent packages
- `ABI` of C-stubs

OCaml in Debian Squeeze

158	OCaml source packages
353	OCaml binary packages
$\approx$ 2'500	OCaml modules

OCaml in Debian Squeeze

158	OCaml source packages
353	OCaml binary packages
$\approx$ 2'500	OCaml modules
0	inconsistent assumptions

OCaml in Debian Squeeze

158	OCaml source packages
353	OCaml binary packages
$\approx$ 2'500	OCaml modules
0	inconsistent assumptions

debian-ocaml-maint@lists.debian.org
Join us !

Questions?