Evolving ASDF

More cooperation, less coordination

François-René Rideau and Robert P. Goldman
ITA and SIFT (respectively)

International Lisp Conference, 19 October 2010

Outline

In a nutshell

- What were we doing?
  - Fixing up ASDF
    - Trying not to wreck a key piece of CL community plumbing in the process
- We discovered
  - Interesting technical challenges
    - From hot-patching
    - From CL pathnames
  - Interesting social challenges
- Some principles
  - Don't wreck backward compatibility
  - Configuration
    - Let users configure based on what they know
    - Let library authors configure based on what *they* know

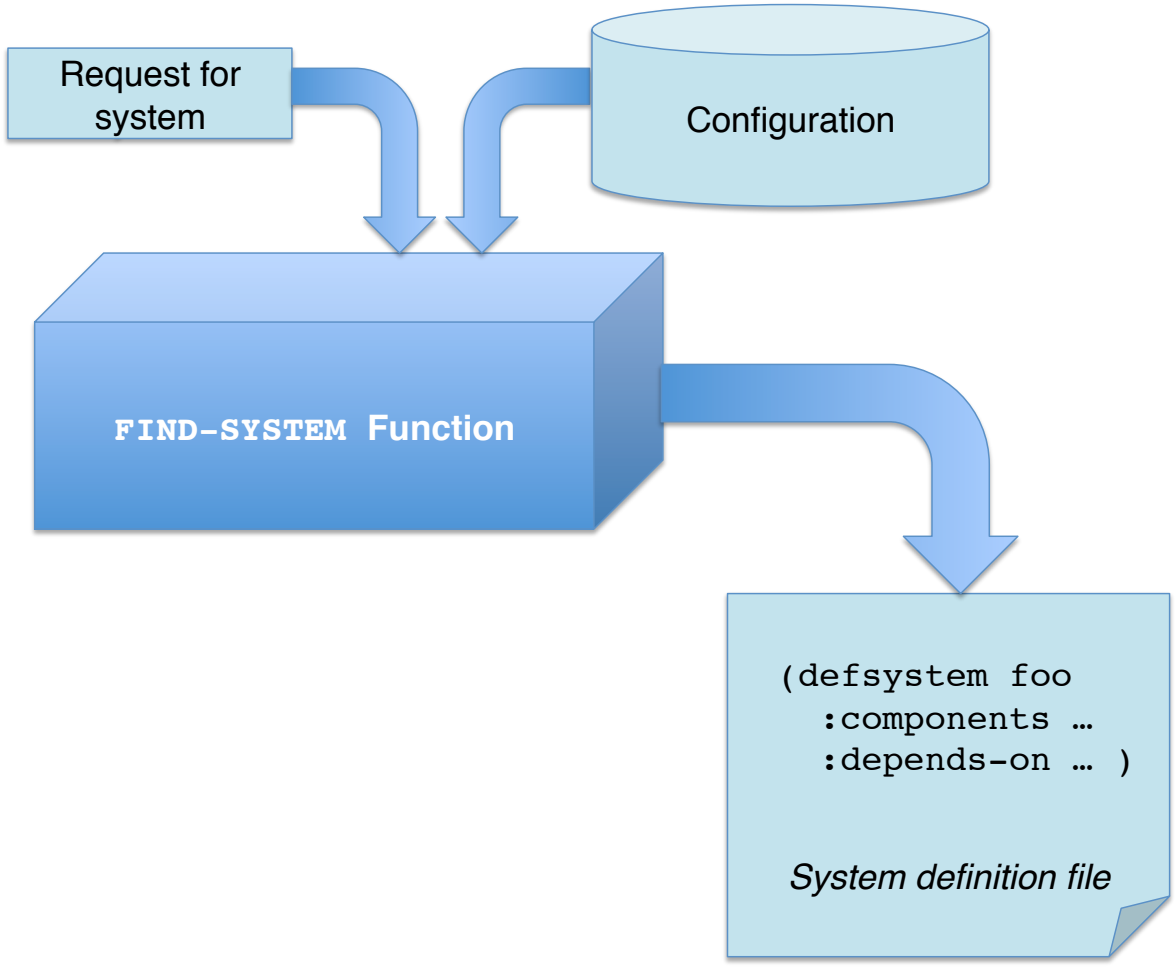
ASDF is the dominant build system for CL

- Plays a role superficially akin to `make` or `ant`
- But must do more
  - Not just build, but load
  - Maintain coherence of long-lived CL images
    - Not always going back to a clean slate like other tools
- Built on the shoulders of giants
  - `MK-DEFSYSTEM`
  - Symbolics and other proprietary `DEFSYSTEM` versions
  - `BUILD`
- Brilliant key idea establishes ASDF dominance
  - Use `*load-truename*` to find system component files
  - Suddenly, installing CL systems is easy
    - No more wrestling with logical pathnames
    - Especially since logical pathnames are insufficiently portable

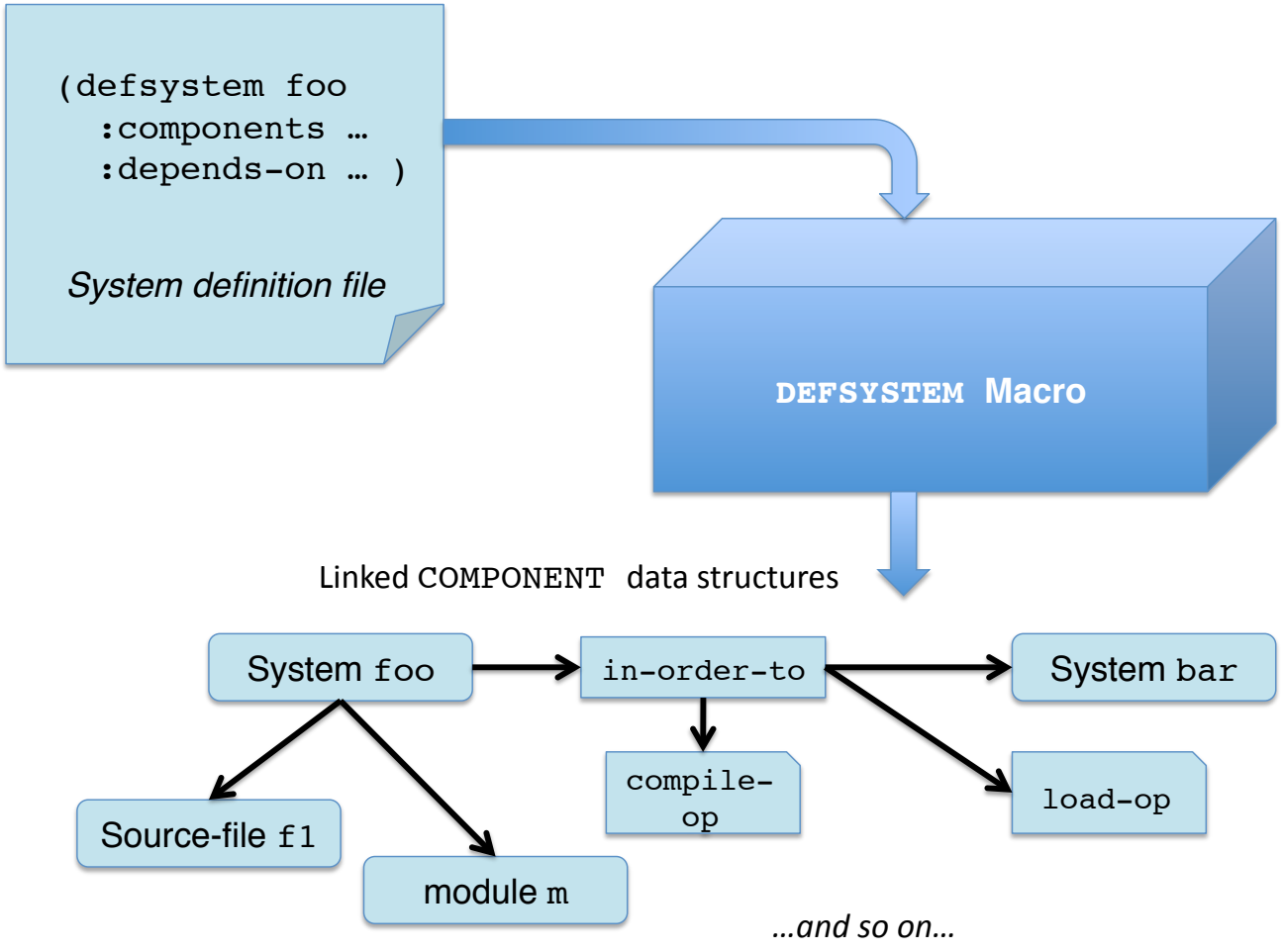
ASDF system definitions

- Contain *metadata*
- Contain *components*
- Contain *dependencies*
  - external
  - internal

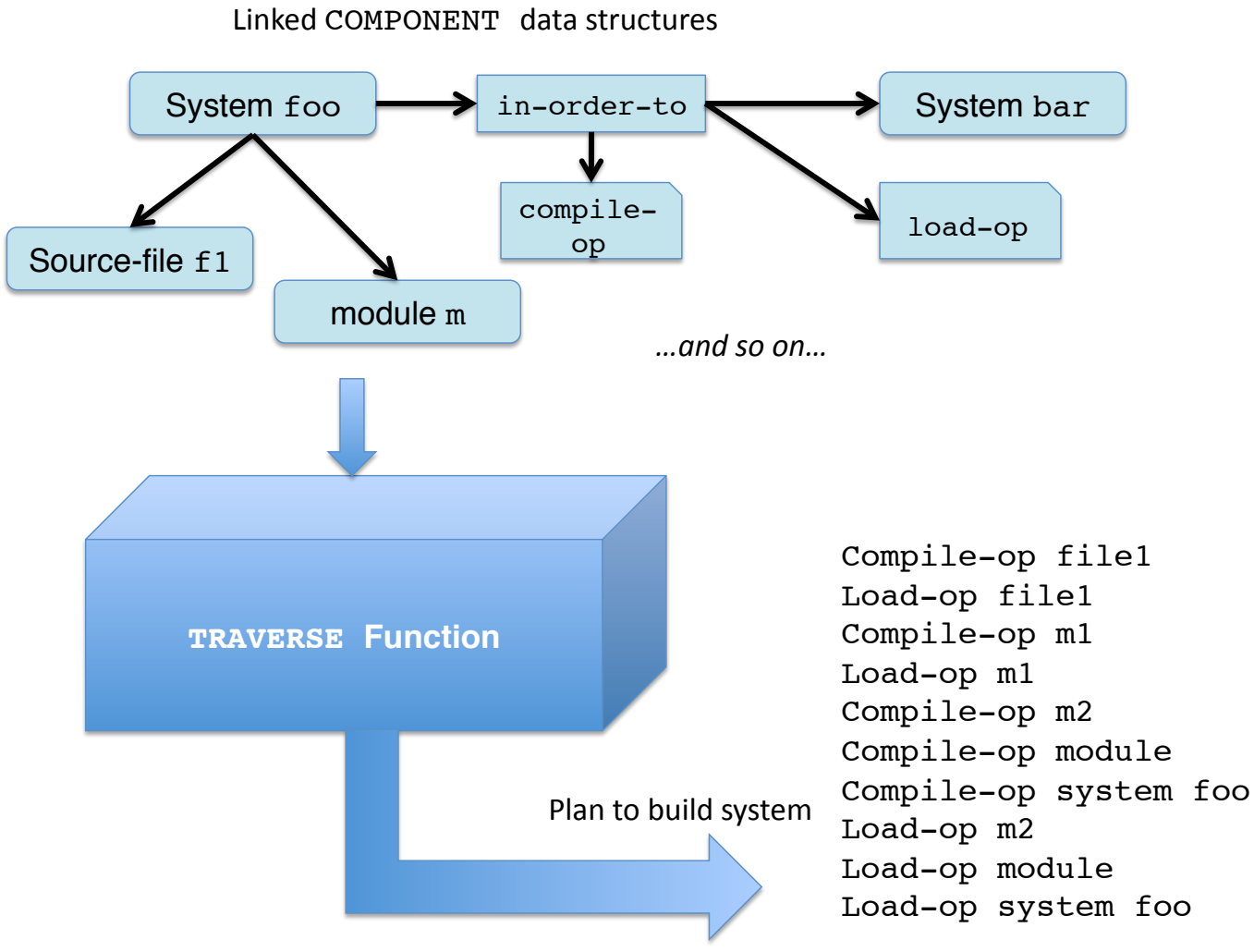
How does ASDF work?



How does ASDF work?



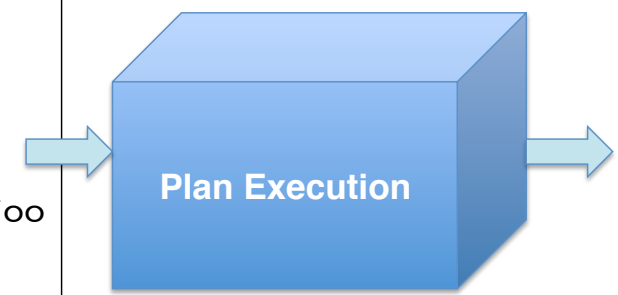
How does ASDF work?



How does ASDF work?

Plan to build system

```
Compile-op file1
Load-op file1
Compile-op m1
Load-op m1
Compile-op m2
Compile-op module
Compile-op system foo
Load-op m2
Load-op module
Load-op system foo
```



Operations on file system,
Compilation,
Loading,
Testing,
Etc.

Three things I wish you knew about ASDF

1. ASDF doesn't just build: it makes a *plan* to build, and then executes the plan
2. Operating on a module is not *wrapped around* the operations on components
   - `:around` methods don't work for customization
3. You can tell ASDF about system versions
   - Say what the version of your system is
     - $x.y.z$, please
     - Please don't skimp on the high-order numbers or ASDF won't detect incompatibilities
   - Say what library versions it depends on

Hot-patching ASDF

- Loading ASDF into a running CL image containing ASDF
- Why is this critical?
  - If people can't load a new ASDF on top of an old one, they can never write portable code using new features
  - *Unless* all the implementations get together and *simultaneously* update their packaged versions
  - Hot-patching allows ASDF to evolve
- Why is this hard?
  - We are replacing bits of ASDF *while it is running*
  - While it is running *to build and load itself*

Configuring ASDF

- Need to be able to find systems
  - We can find system components from the system definition
  - But we still need to find the system definitions!
- Need to place compiled files

  Explain the need here

Key issues are wrestling with CL pathnames

Future directions

- There is still a lot of room for improvement with ASDF
- Bugs
  - Outstanding bug: across system dependencies do not properly trigger recompilations
    - Challenging to fix
    - Some think it's not a bug
    -
- New features
- Better documentation

ASDF 2 is now available

- Download and load into your implementation of choice
- Bundled with an increasing number of implementations
  - SBCL
  - ABCL
  - ECL
  - SCL
  - CMUCL
  - CCL
- `:asdf2` in `*features*`
  - CLISP under discussion
  - Yet to come (?)
    - ACL
    - LispWorks
    - Corman

Conclusions

- ASDF 2 needs new maintainers
- But please remember — ASDF is the glue that holds the CL community together
  - Be gentle with it!