GNU Guix: Scheme as a uniform OS admin and deployment interface

Ludovic Courtès

Commercial Users of Functional Programming
24 September 2016, Nara, Japan

GuixSD

```
$ guix package -i gcc-toolchain coreutils sed grep
…

$ eval `guix package --search-paths`
…

$ guix package --manifest=my-software.scm
…
```

```
(operating-system
  (host-name "schememachine")
  (timezone "Japan")
  (locale "ja_JP.utf8")
  (bootloader (grub-configuration (device "/dev/sda")))
  (file-systems (cons (file-system
                        (device "my-root")
                        (title 'label)
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))
  (users (cons (user-account
                (name "alice")
                (group "users")
                (home-directory "/home/alice"))
               %base-user-accounts))
  (services (cons* (dhcp-client-service)
                   (lsh-service #:port-number 2222)
                   %base-services)))
```

How we got there.

“Is there a package manager for neural networks?”

— Question from the audience,
Martin Abadi’s keynote on TensorFlow,
ICFP day 1

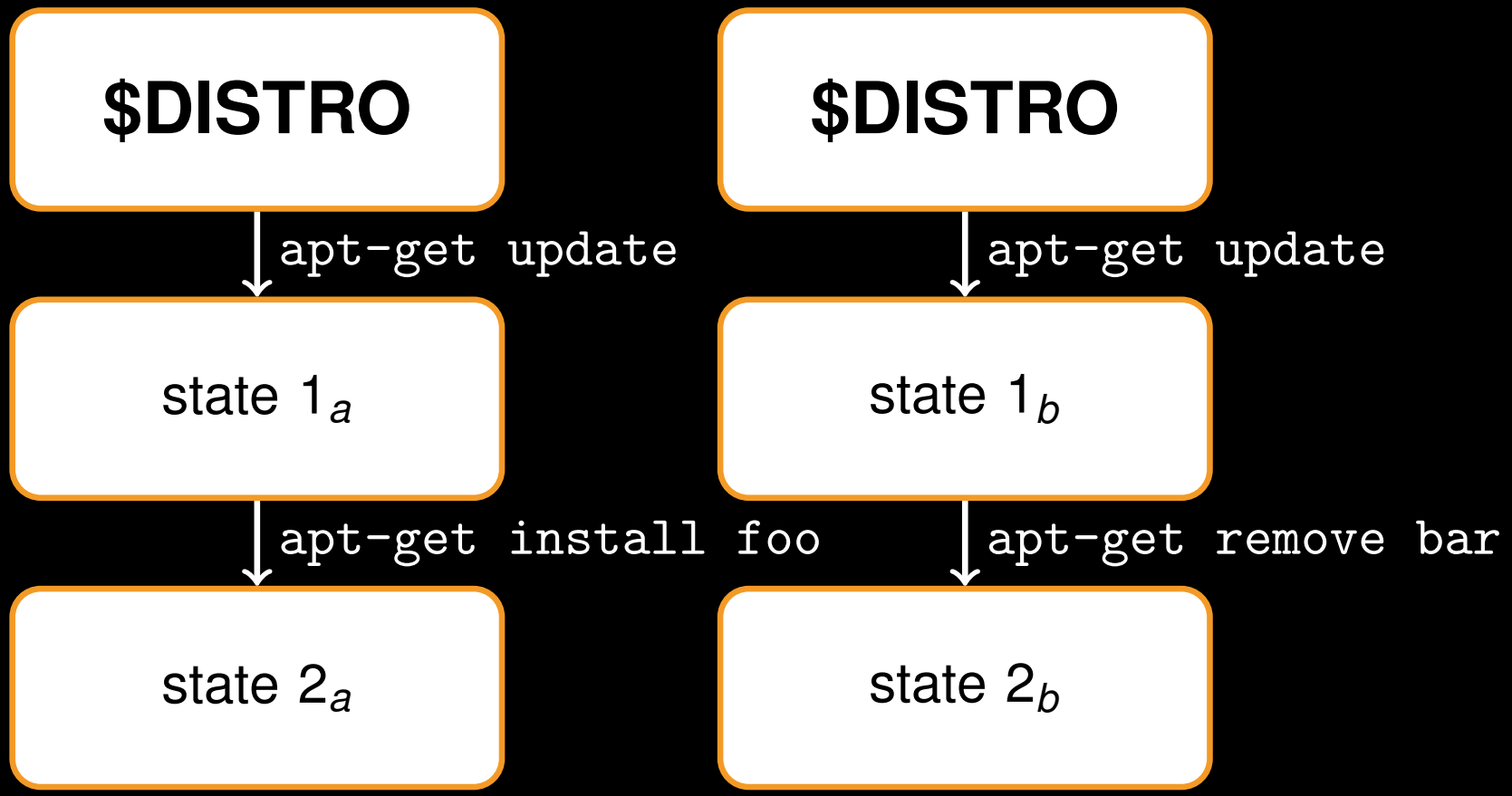

Cabal

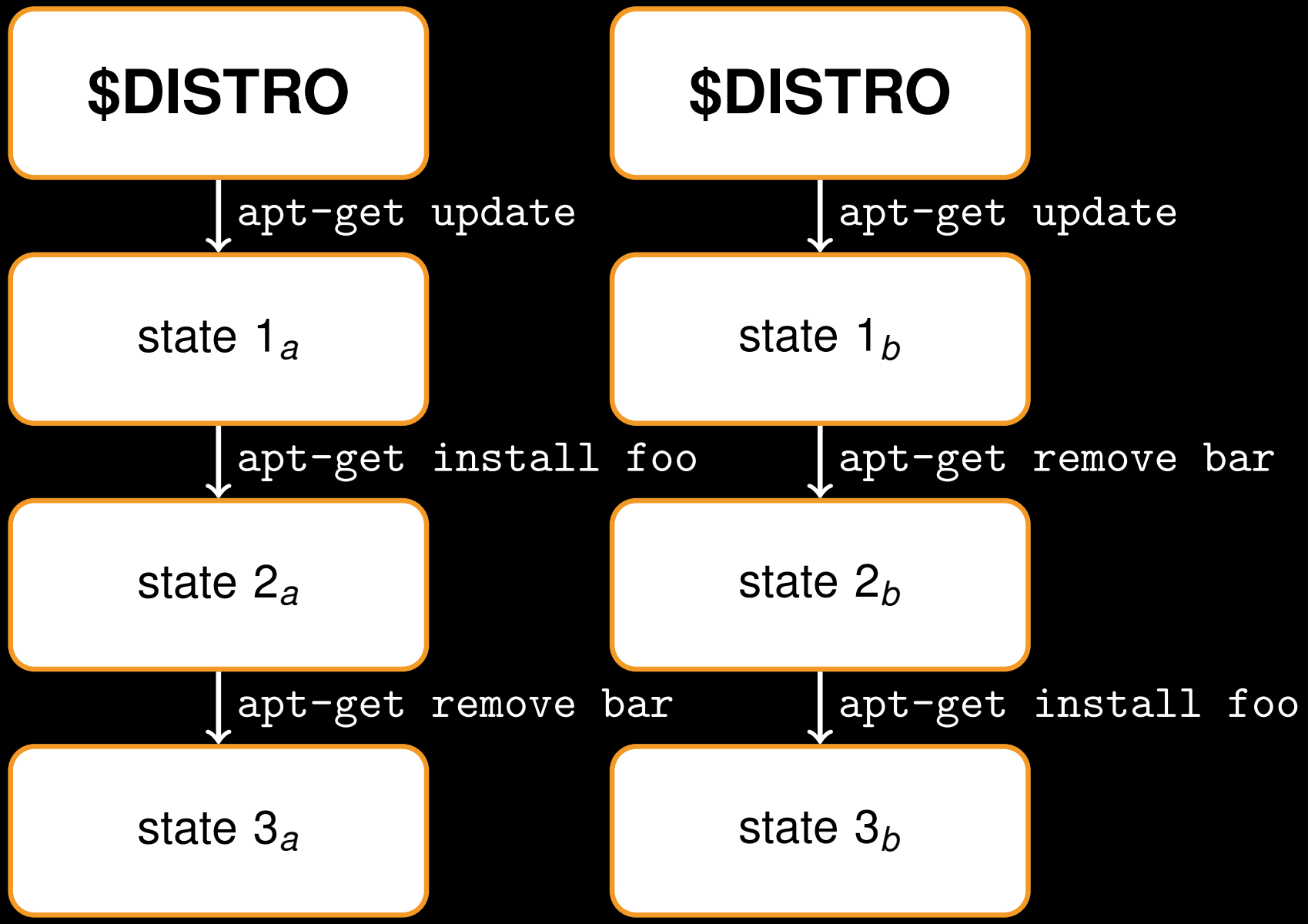

elm

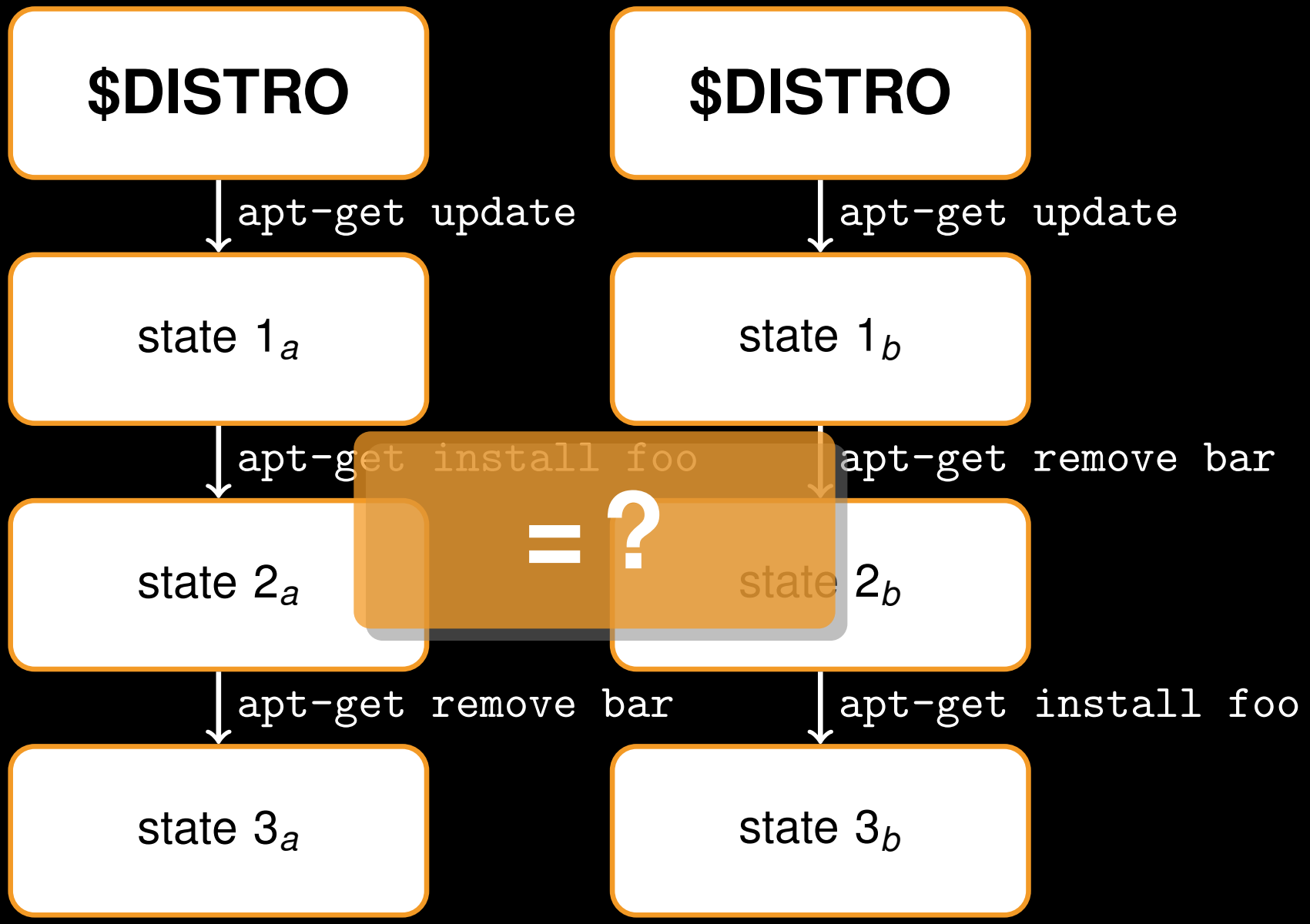
$DISTRO

$DISTRO


$DISTRO
apt-get update
state 1_a
$DISTRO
apt-get update
state 1_b


$DISTRO
apt-get update
state 1_a
apt-get install foo
state 2_a
$DISTRO
apt-get update
state 1_b
apt-get remove bar
state 2_b


$DISTRO
$DISTRO
apt-get update
apt-get update
state 1_a
state 1_b
apt-get install foo
apt-get remove bar
state 2_a
state 2_b
apt-get remove bar
apt-get install foo
state 3_a
state 3_b


$DISTRO
$DISTRO
apt-get update
apt-get update
state 1_a
state 1_b
apt-get install foo
apt-get remove bar
= ?
state 2_a
state 2_b
apt-get remove bar
apt-get install foo
state 3_a
state 3_b

Functional package management paradigm:

1. build process = **pure function**
2. built software = **persistent graph**

Imposing a Memory Management Discipline on Software Deployment, Dolstra et al., 2004 (Nix package manager)

```
$ guix build ocaml
```

```
$ guix build ocaml
/gnu/store/ h2g4sc09h4... -ocaml-4.02.3
```

hash of *all* the dependencies

The Nix language

function definition

function call

```
{  fetchurl, stdenv  } :

 stdenv . mkDerivation {
  name = "hello-2.3";
  src = fetchurl {
    url = mirror://gnu/hello/hello-2.3.tar.bz2;
    sha256 = "0c7vijq8y68...";
  };

  meta = {
    description = "Produces a friendly greeting";
    homepage = http://www.gnu.org/software/hello/;
    license = "GPLv3+";
  };
}
```

The Nix language

```
{ fetchurl, stdenv } :

 stdenv . mkDerivation {
  name = "hello-2.3";
  src = fetchurl {
    url = mirror://gnu/hello/hello-2.3.tar.bz2;
    sha256 = "0c7vijq8y68...";
  };
  preCheck = "echo 'Test suite coming up!'";
  meta = {
    description = "Produces a friendly greeting";
    homepage = http://www.gnu.org/software/hello/;
    license = "GPLv3+";
  };
}
```

Bash snippet

NixOS / nixpkgs

Code | Issues 1,215 | Pull requests 317 | Projects 0 | Pulse

lib: Make escapeShellArg more robust

Quoting various characters that the shell *may* interpret specially is a very fragile thing to do.

I've used something more robust all over the place in various Nix expression I've written just because I didn't trust escapeShellArg.

Here is a proof of concept showing that I was indeed right in distrusting escapeShellArg:

Scheme all the way down.

“Escaping DSL hell by having parentheses all the way down”

– talk by Tom Hall

`https://skillsmatter.com/skillscasts/5488-escaping-dsl-hell-by-having-parenthesis-all-the-way-down`

```
(define hello
  (package
    (name "hello")
    (version "2.8")
    (source (origin
              (method url-fetch)
              (uri (string-append
                    "http://ftp.gnu.org/…/hello-" version
                    ".tar.gz"))
              (sha256 (base32 "0wqd…dz6"))))
    (build-system gnu-build-system)
    (synopsis "An example GNU package")
    (description "Produce a friendly greeting.")
    (home-page "https://gnu.org/software/hello/")
    (license gpl3+)))

;; Yields: /gnu/store/…-hello-2.8
```

```
(operating-system
  ;; …
  (services (cons (openssh-service #:port 2222)
                  %base-services)))
```

```
(operating-system
  ;; …
  (services (remove (lambda (service)
                      (eq? ntp-service-type
                           (service-kind service)))
                    %desktop-services)))
```

```
(define %my-services
  ;; My very own list of services.
  (modify-services %desktop-services
    (mingetty-service-type config =>
                           (mingetty-configuration
                            (inherit config)
                            (motd (plain-file "motd"
                                    "Howdy Nara!"))))
    (upower-service-type config =>
                         (upower-configuration
                          (inherit config)
                          (ignore-lid? #true)
                          (percentage-critical 5.)))))
```

- Emacs and Web user interfaces
- `guix refresh` package auto-updater
- `guix lint` package checker
- `guix graph` dependency graph viewer
- `guix system extension-graph` service composition viewer
- ...

Unification
beyond the “distro”.

The Initial RAM Disk

The Initial RAM Disk

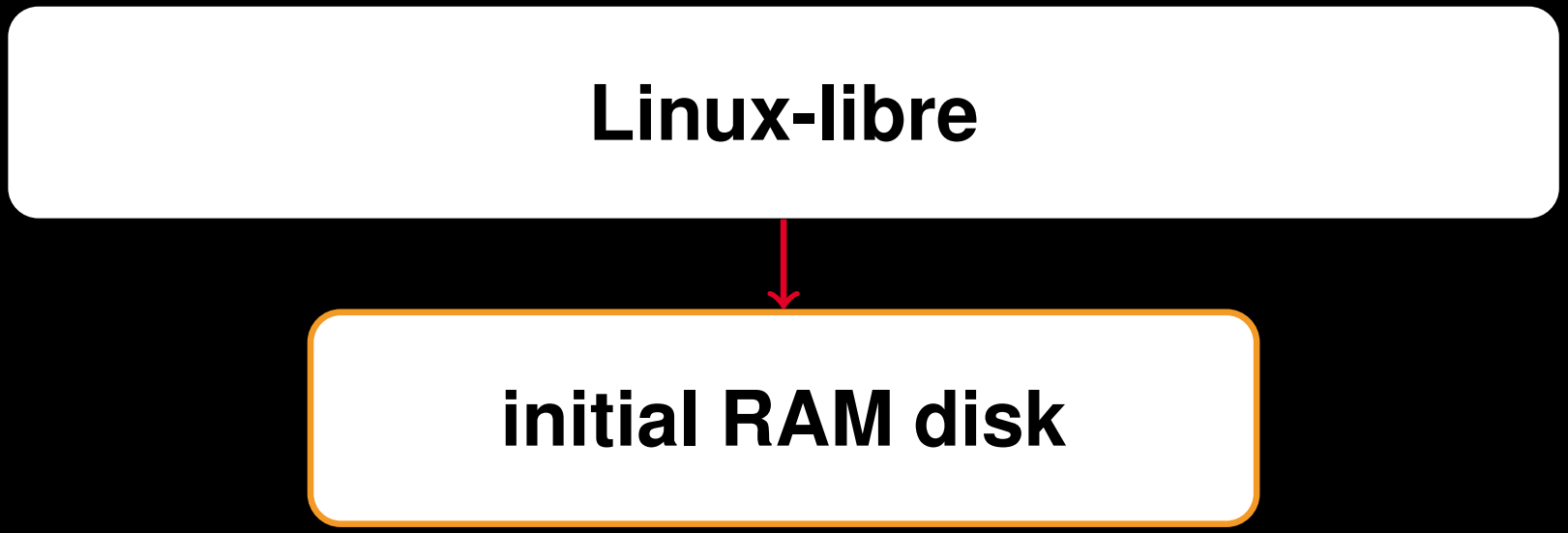
```
(expression->initrd
 (with-imported-modules (source-module-closure
                         '((gnu build linux-boot)
                           (guix build utils)))
   #~ (begin
       (use-modules (gnu build linux-boot)
                    (guix build utils))

       (boot-system #:mounts '#$file-systems
                    #:linux-modules '#$linux-modules
                    #:linux-module-directory '#$kodir)))
```

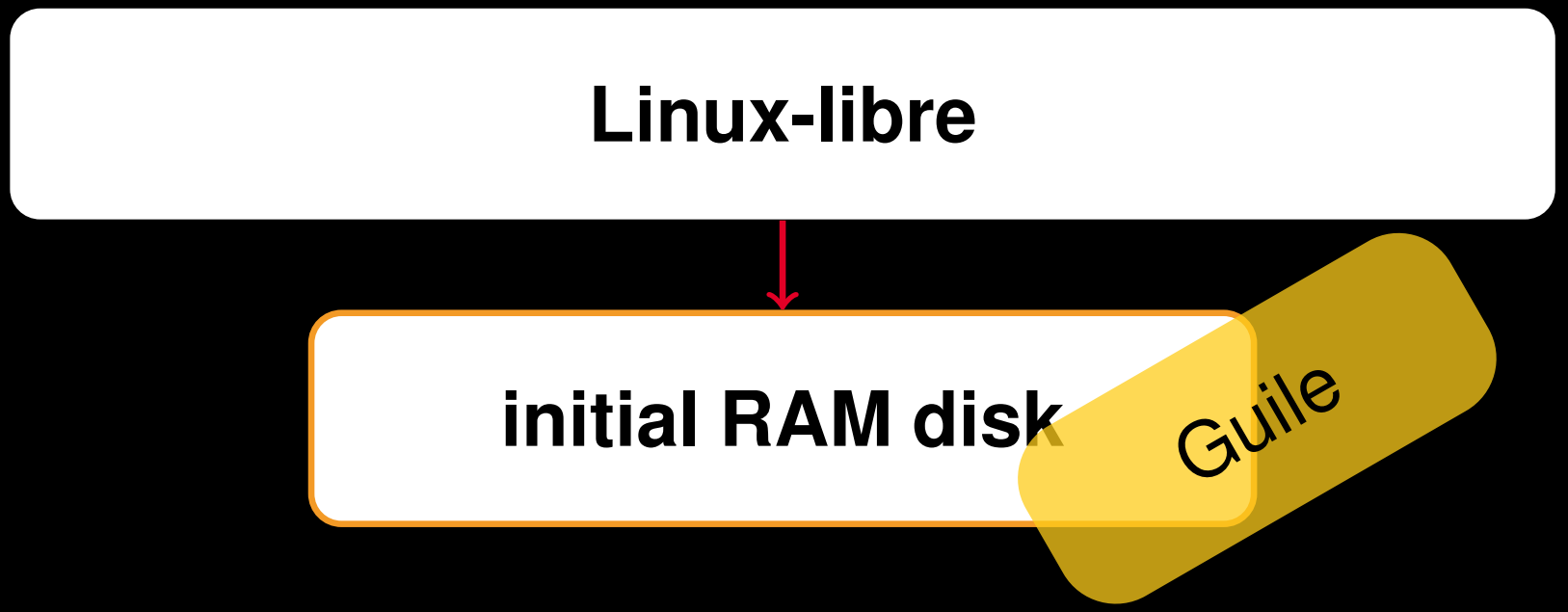
The Initial RAM Disk

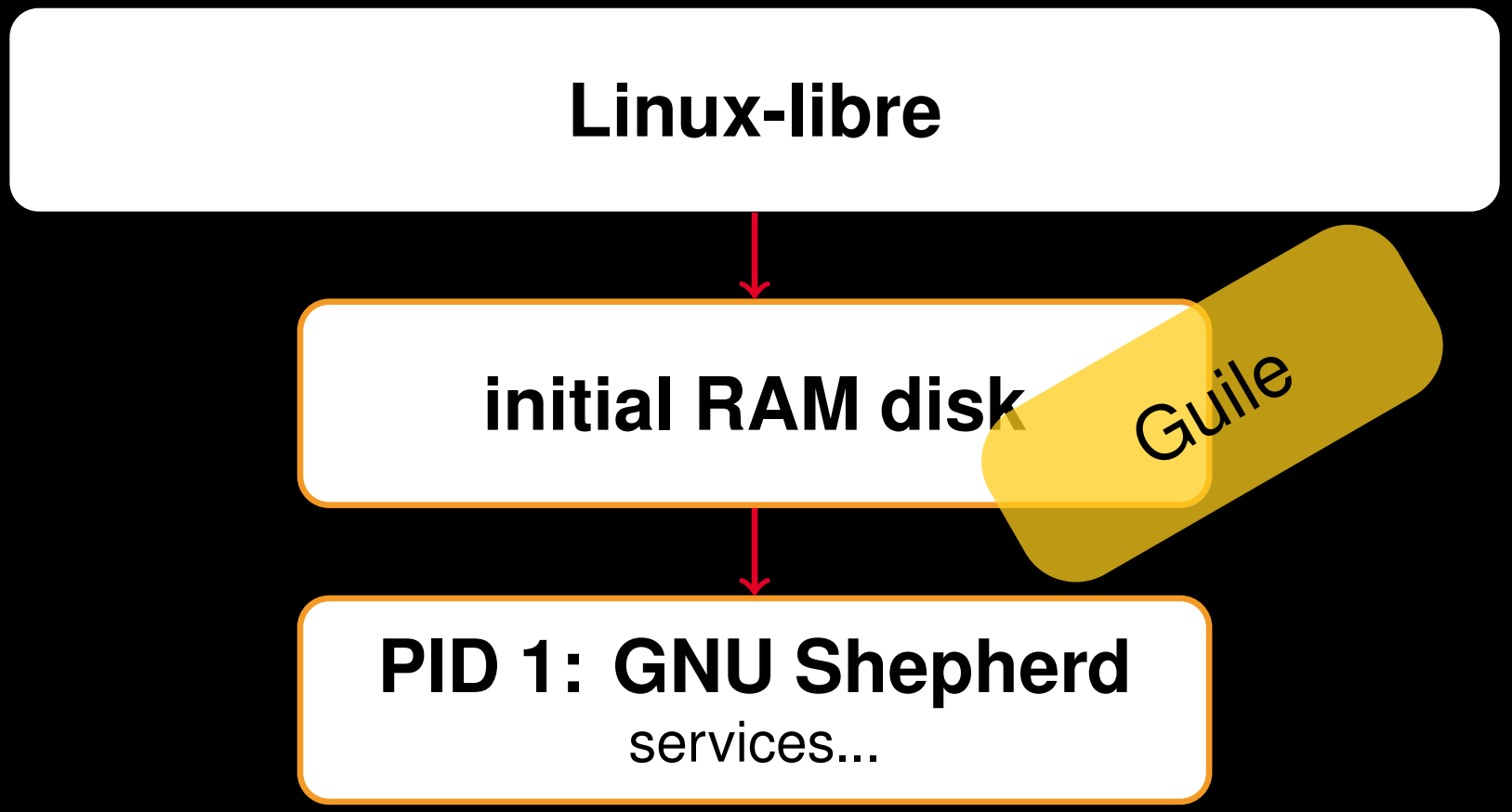
code staging

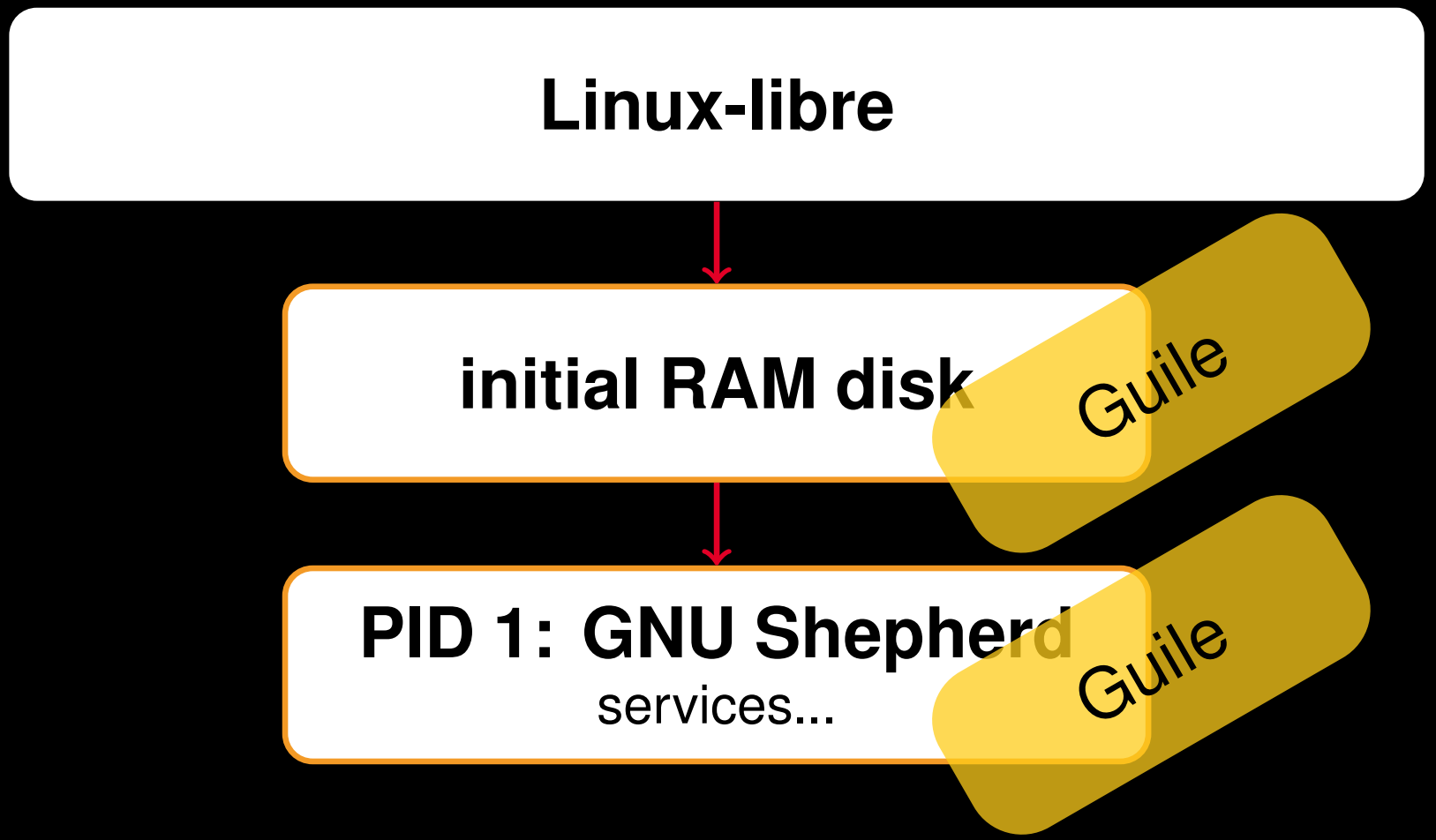
```
(expression->initrd
 (with-imported-modules (source-module-closure
                         '((gnu build linux-boot)
                           (guix build utils)))
   #~ (begin
        (use-modules (gnu build linux-boot)
                     (guix build utils))

        (boot-system #:mounts '#$file-systems
                     #:linux-modules '#$linux-modules
                     #:linux-module-directory '#$kodir)))
```

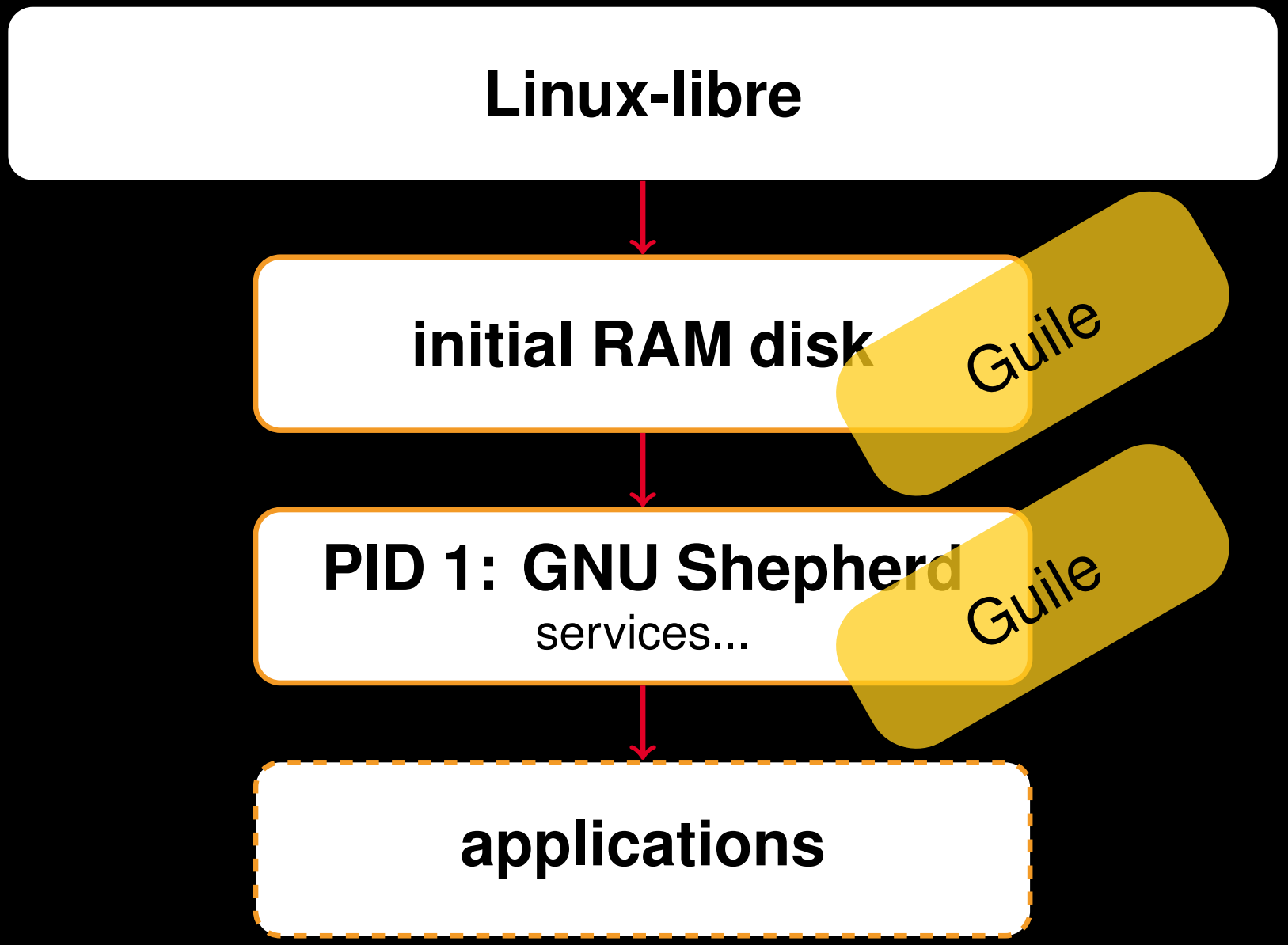
Linux-libre


Linux-libre
initial RAM disk


Linux-libre
initial RAM disk
Guile


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...
Guile


Linux-libre
initial RAM disk
Guile
PID 1: GNU Shepherd
services...
Guile
applications

System Services

```
;; Service definition for the GNU Shepherd (PID 1)
;; embedded in GuixSD.

(shepherd-service
  (provision '(mysql))
  (documentation "Run the MySQL server.")
  (start (let ((my.cnf (mysql-configuration-file config)))
           #~(make-forkexec-constructor
              (list (string-append #$mysql "/bin/mysqld")
                    (string-append "--defaults-file="
                                   #$my.cnf))
              #:user "mysql" #:group "mysql")))
  (stop #~(make-kill-destructor)))
```

Wrap-up.

Summary

- distro & tools as a **Scheme library**
- **hackability** through uniformity
- **code staging** techniques to glue it all

Join us now, share the parens!

- **install the distribution**
- **use it**, report bugs, add packages
- share your **ideas**!

GuixSD

ludo@gnu.org

https://gnu.org/software/guix/