Introduction to G-Expressions

Christopher Marusich, FOSDEM 2020

Copying

About Me

- Software developer from Seattle
- Free software supporter
- Occasional contributor to GNU Guix
- All views my own

Quick Review of Guix

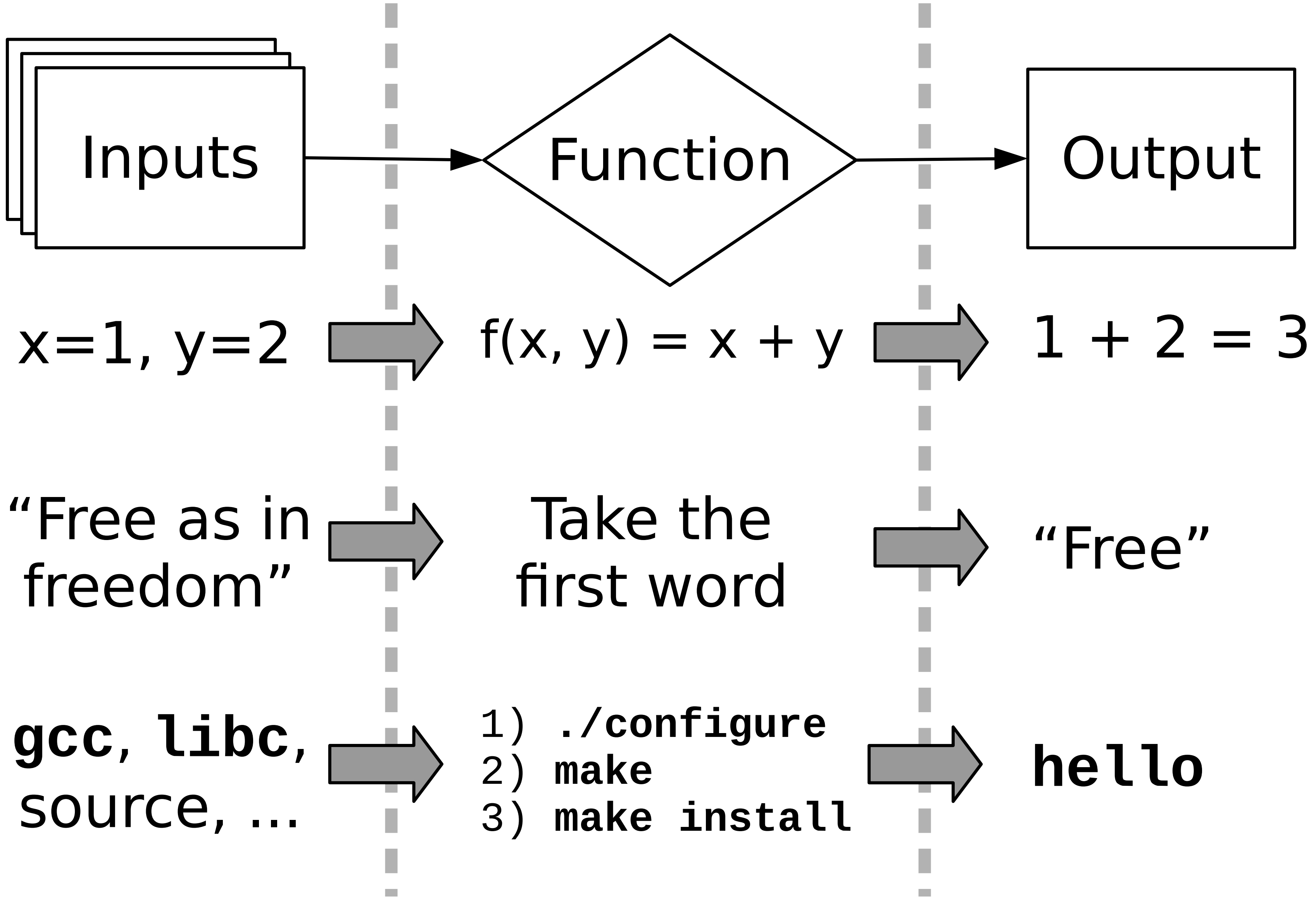
- Functional software deployment
- Build and compose software components
- A GNU/Linux distribution from scratch

```
guix package --install emacs vim
guix package --remove vim
guix package --roll-back
guix system reconfigure my-os.scm
guix system list-generations
guix system roll-back
```

```
(package
  (name "clutter-gst")
  (version "3.0.27")
  (source
   (origin
     (method url-fetch)
     (uri (string-append "mirror://gnome/sources/clutter-gst/"
                         (version-major+minor version) "/"
                         "clutter-gst-" version ".tar.xz"))
     (sha256
      (base32 "17czmpl92dzi4h3rn5rishk015yi3jwiw29zv8qan94xcmnbssgy"))))
  (build-system gnu-build-system)
  (native-inputs
   `(("glib:bin" ,glib "bin")          ; for glib-mkenums
     ("pkg-config" ,pkg-config)
     ("gobject-introspection" ,gobject-introspection)))
  (inputs
   `(("clutter" ,clutter)
     ("gstreamer" ,gstreamer)
     ("gst-plugins-base" ,gst-plugins-base)))
  (home-page "http://www.clutter-project.org")
  (synopsis "Integration library for using GStreamer with Clutter")
  (description
   "Clutter-Gst is an integration library for using GStreamer with Clutter.
It provides a GStreamer sink to upload frames to GL and an actor that
implements the ClutterGstPlayer interface using playbin.  Clutter is an
OpenGL-based interactive canvas library.")
  (license license:lgpl2.0+))
```

```
(operating-system
  (host-name "komputilo")
  (timezone "Europe/Berlin")
  (locale "en_US.utf8")
  (bootloader (bootloader-configuration
                (bootloader grub-bootloader)
                (target "/dev/sda")))
  (file-systems (cons (file-system
                        (device (file-system-label "my-root"))
                        (mount-point "/")
                        (type "ext4"))
                      %base-file-systems))
  (users (cons (user-account
                (name "alice")
                (comment "Bob's sister")
                (group "users")
                (supplementary-groups '("wheel" "audio" "video")))
               %base-user-accounts))
  (packages (cons screen %base-packages))
  (services (append (list (service dhcp-client-service-type)
                          (service openssh-service-type
                                   (openssh-configuration
                                    (port-number 2222))))
                    %base-services)))
```

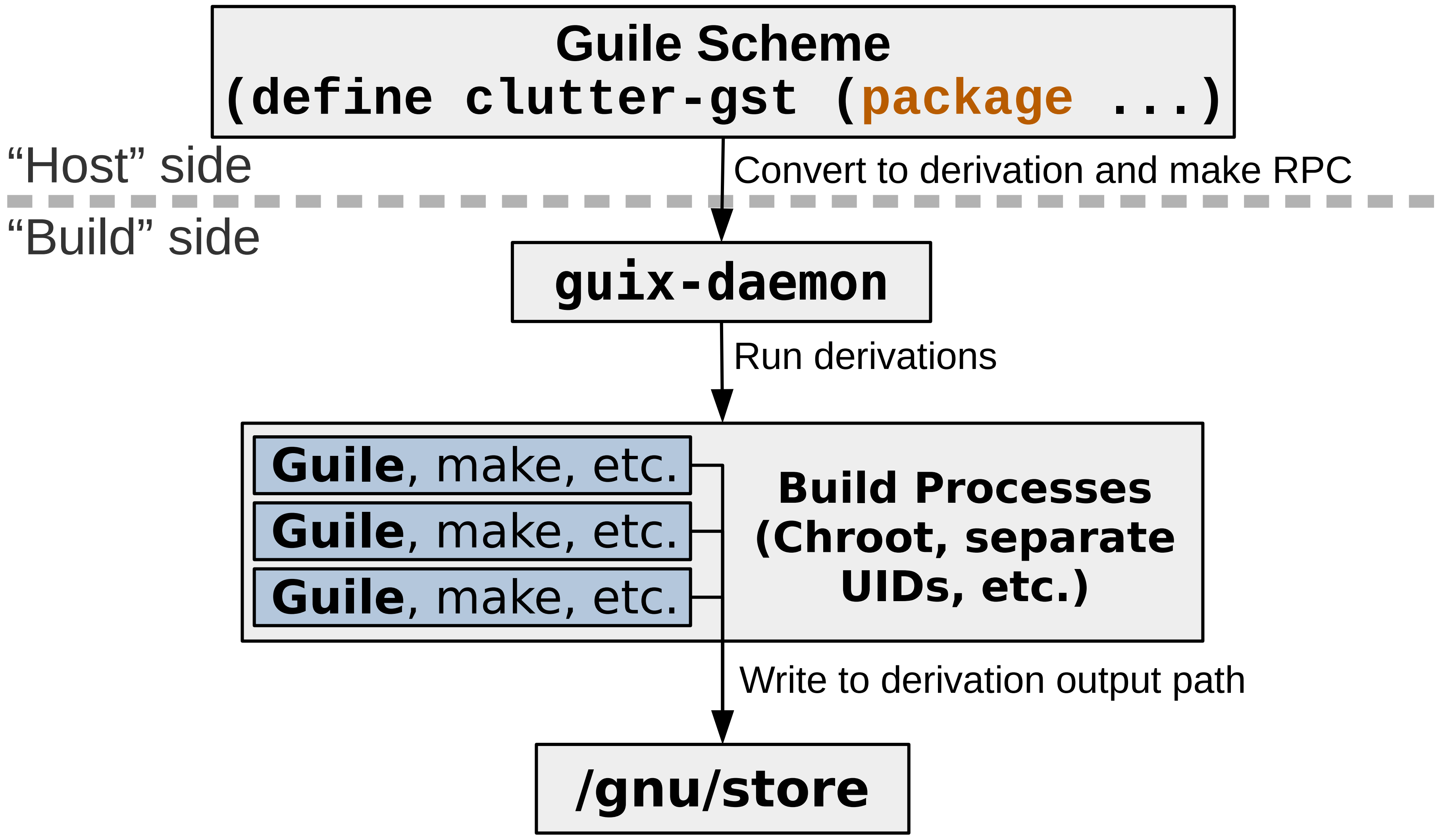
The Functional Model



The Functional Model

- Output stored in ***/gnu/store***
- Same inputs → same output
  - `/gnu/store/...zf83la7k-hello-2.10`
- Different inputs → different output
  - **/gnu/store/**$\underbrace{\texttt{...369lzbbr}}_{\text{Hash of inputs}}$**-hello-2.10**

Bird’s-Eye View of Guix


Guile Scheme
(define clutter-gst (package ...))
“Host” side
Convert to derivation and make RPC
“Build” side
guix-daemon
Run derivations
Guile, make, etc.
Guile, make, etc.
Guile, make, etc.
Build Processes (Chroot, separate UIDs, etc.)
Write to derivation output path
/gnu/store

```
;; Code staging, first attempt: S-Expressions.

(define build-exp
  ;; Build-side code.
  ‘(symlink “/gnu/store/123...-coreutils-8.25”
            “/gnu/store/abc...-result”))
```

```
;; Code staging, first attempt: S-Expressions.

(define build-exp
  ;; Build-side code.
  '(symlink (assoc-ref %build-inputs "coreutils")
            %output))

;; ... with unhygienic global variable:
;; (define %build-inputs
;; '(("coreutils" . "/gnu/store/...-coreutils-8.25")))

(define inputs
  ;; What goes into the chroot.
  `(("coreutils" ,coreutils)))

(build-expression->derivation
  store "symlink-to-coreutils" build-exp
  #:inputs inputs)
```

```scheme
;; Code staging, first attempt: S-Expressions.

(define build-exp
  ;; Build-side code.
  ‘(symlink (assoc-ref %build-inputs “coreutils”)
            %output))

;; ... with unhygienic global variable:
;; (define %build-inputs
;; ‘(()))

(define inputs
  ;; What goes into the chroot.
  ‘(()))

(build-expression->derivation
  store “symlink-to-coreutils” build-exp
  #:inputs inputs)
```

Code Staging: S-Exps

- Pros:
  - Familiar
  - Dynamic
- Cons:
  - No dependency tracking
  - More complex/verbose
  - Not composable

```
;; Code staging, second attempt: G-Expressions

(define build-exp
  ;; First-class object that carries info
  ;; about its dependencies.
  (gexp (symlink (ungexp coreutils)
                 (ungexp output))))


;; Leads to a build script like:
;; (symlink “/gnu/store/123...-coreutils-8.25”
;;          (getenv “out”))

(gexp->derivation “symlink-to-coreutils” build-exp)
```

```
;; Code staging, second attempt: G-Expressions

(define build-exp
  ;; First-class object that carries info
  ;; about its dependencies.
  #~(symlink #$coreutils #$output))



;; Leads to a build script like:
;; (symlink “/gnu/store/123...-coreutils-8.25”
;;          (getenv “out”))

(gexp->derivation “symlink-to-coreutils” build-exp)
```

```
;; Modules

(define build-exp




    #~(begin
        (use-modules (guix build utils))
        (mkdir-p (string-append #$output “/bin”))))

(gexp->derivation “empty-bin-dir” build-exp)
;; ERROR: no code for module (guix build utils)
;; Modules

(define build-exp
  ;; Compile (guix build utils) and add it
  ;; to the build environment.
  (with-imported-modules ‘((guix build utils))
    #~(begin
        (use-modules (guix build utils))
        (mkdir-p (string-append #$output “/bin”)))))

(gexp->derivation “empty-bin-dir” build-exp)
```

Code Staging: G-Exps

- Pros:
  - Dependency tracking
  - Less complex/verbose
  - Composable
- Cons:
  - Unfamiliar
  - Not dynamic

```
;; Daemon management with GNU Shepherd.

(define (openssh-shepherd-service config)
  “Return a <shepherd-service> for openssh with CONFIG.”

  (define pid-file
    (openssh-configuration-pid-file config))

  (define openssh-command
    #~(list (string-append
               #$(openssh-configuration-openssh config)
               “/sbin/sshd”)
             “-D” “-f” #$(openssh-config-file config)))

  (list (shepherd-service
         (documentation “OpenSSH server”)
         (requirement ‘(syslogd loopback))
         (provision ‘(ssh-daemon ssh sshd))
         (start #~(make-forkexec-constructor
                   #$openssh-command
                   #:pid-file #$pid-file))
         (stop #~(make-kill-destructor))
         (auto-start? (openssh-auto-start? config)))))
```

```
;; Booting an initrd.

(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))))
```

```scheme
;; Activation script that runs when system boots.

(define dhcpd-activation
  (match-lambda
    (($ <dhcpd-configuration> package config-file version
                              run-directory lease-file pid-file
                              interfaces)
     (with-imported-modules '((guix build utils))
       #~(begin
           (unless (file-exists? #$run-directory)
             (mkdir #$run-directory))
           ;; According to the DHCP manual (man
           ;; dhcpd.leases), the lease database must be
           ;; present for dhcpd to start successfully.
           (unless (file-exists? #$lease-file)
             (with-output-to-file #$lease-file
               (lambda _ (display ""))))
           ;; Validate the config.
           (invoke
            (string-append #$package "/sbin/dhcpd") "-t" "-cf"
            #$config-file))))))
```

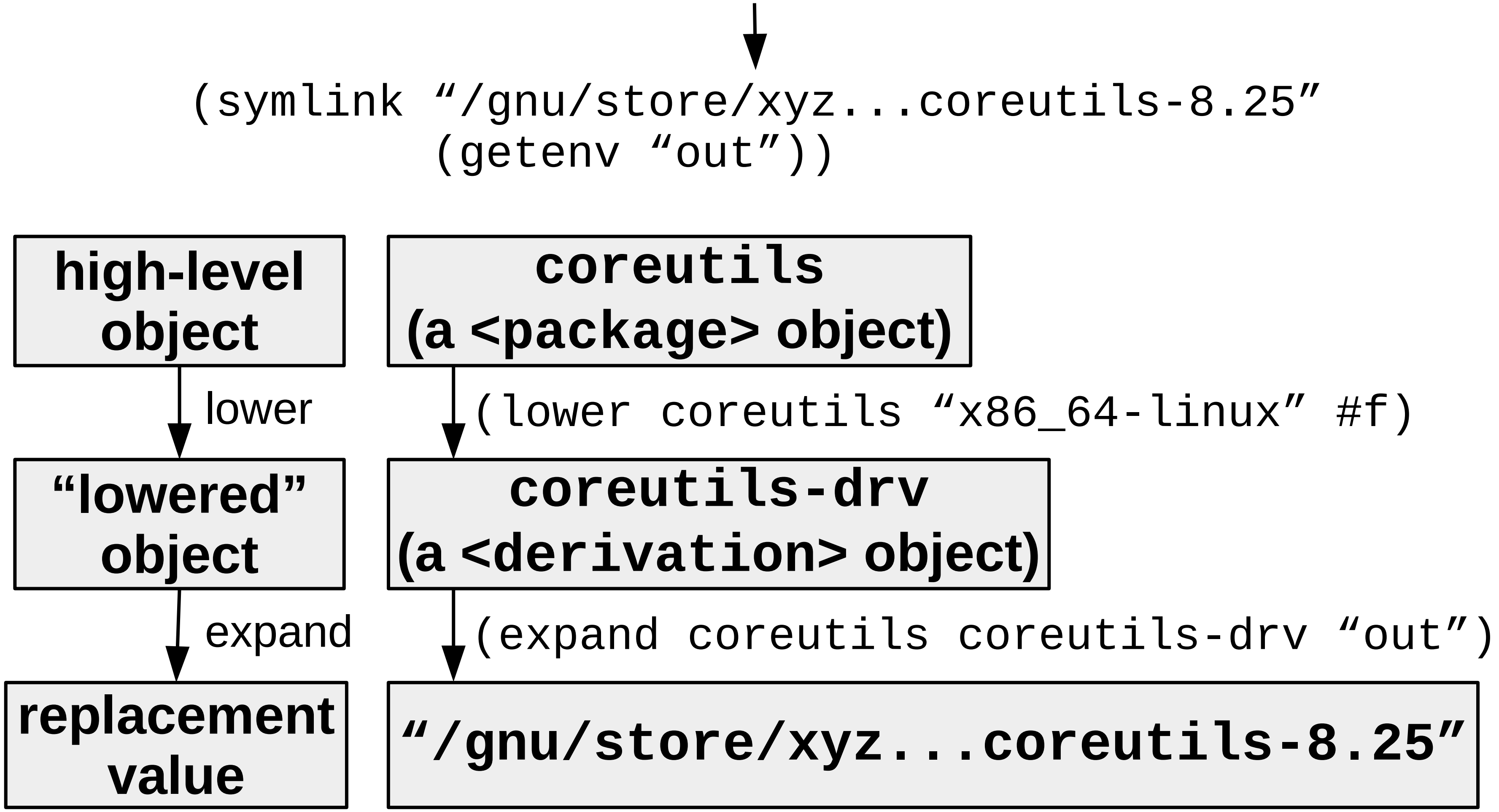
Other Uses

- IceCat liberation procedure
- System tests
- `guix pull`
- `remote-eval` (used by `guix deploy`)
- `eval/container`

G-Expression Compilers

```
#~(symlink #$coreutils #$output)
```

```
(symlink "/gnu/store/xyz...coreutils-8.25"
         (getenv "out"))
```



high-level object
lower
"lowered" object
expand
replacement value
coreutils (a <package> object)
(lower coreutils "x86_64-linux" #f)
coreutils-drv (a <derivation> object)
(expand coreutils coreutils-drv "out")
"/gnu/store/xyz...coreutils-8.25"

For details, see `gexp->sexp` in `guix/gexp.scm`

```
;; Custom high-level objects.

#~(system* (string-append #$hello “/bin/hello”))
;; Expands to:
;; (system* (string-append “/gnu/store/xyz...hello-2.10”
;;                         “/bin/hello"))

#~(system* #$(file-append hello “/bin/hello”))
;; Expands to:
;; (system* “/gnu/store/xyz...hello-2.10/bin/hello”)
;; Custom high-level objects.

#~(system* (string-append #$hello “/bin/hello”))
;; Expands to:
;; (system* (string-append “/gnu/store/xyz...hello-2.10”
;;                          “/bin/hello"))

#~(system* #$(file-append hello “/bin/hello”))
;; Expands to:
;; (system* “/gnu/store/xyz...hello-2.10/bin/hello”)

#~(system* #$(file-append tor “/bin/tor”)
           “-f” #$(local-file “/path/to/my/torrc))
;; Expands to:
;; (system* “/gnu/store/9b5...tor-0.4.2.5/bin/tor”
            “-f” “/gnu/store/c72...torrc”)
```

G-Expressions: Summary

- Tie code staging to software deployment
- Can be more ergonomic than S-Expressions
- Widely used in Guix for many purposes

More Information

- “Code Staging in GNU Guix” by Ludovic Courtès:
  - https://hal.inria.fr/hal-01580582/en
- Manual:
  - https://guix.gnu.org/manual/en/html_node/G_002dExpressions.html
- Source:
  - https://savannah.gnu.org/git/?group=guix
  - https://git.savannah.gnu.org/git/guix.git
- Project:
  - https://guix.gnu.org/

Thank You! Questions?