Reproducible and User-Controlled Package Management in HPC with GNU Guix

Ludovic Courtès (ludovic.courtes@inria.fr)
Ricardo Wurmus (ricardo.wurmus@mdc-berlin.de)

Workshop on Reproducibility in Parallel Computing (RepPar)
25 August 2015

“Reproducibility”?

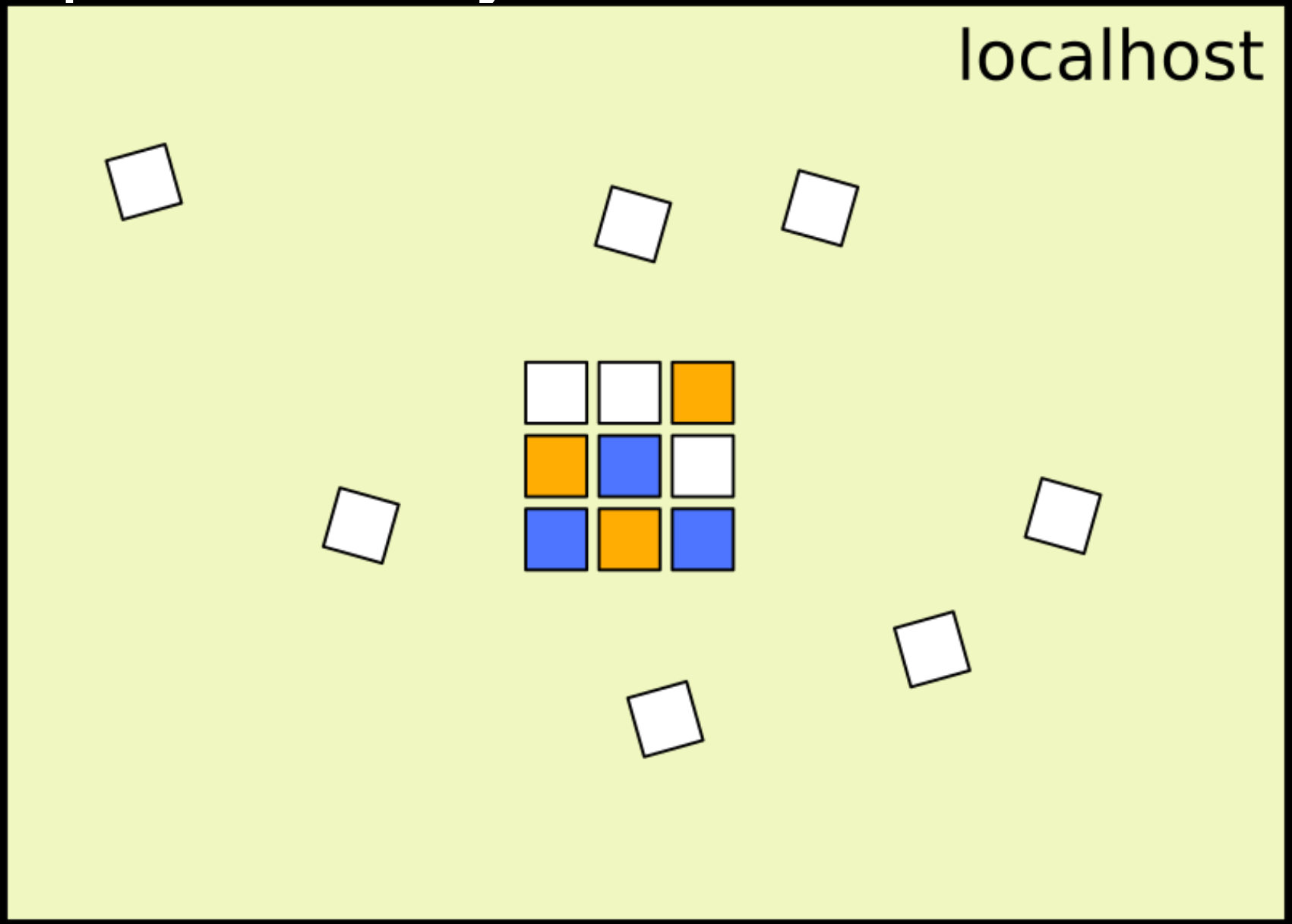
“Reproducibility”?

1. **bit-reproducible** builds

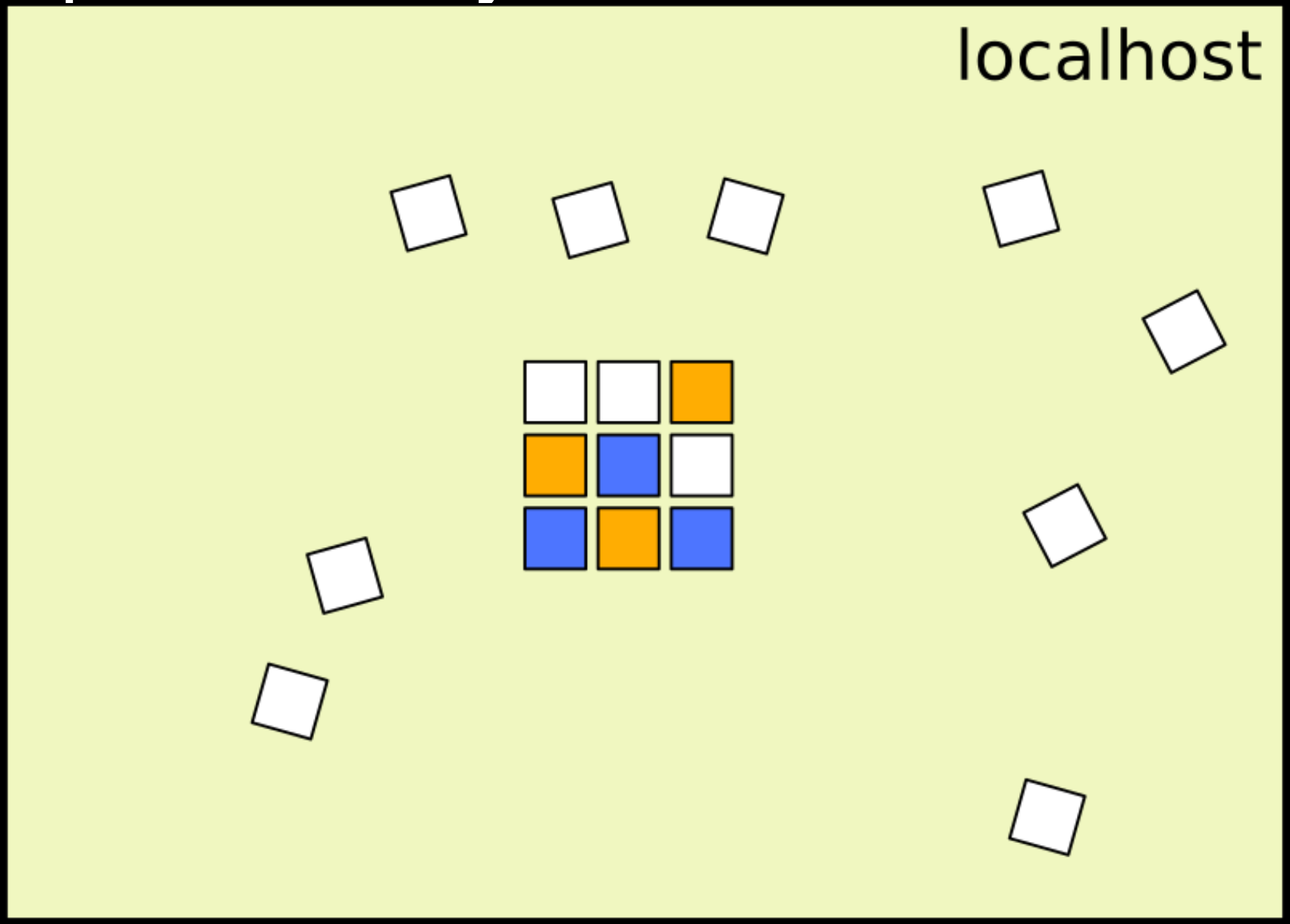
“Reproducibility”?

1. **bit-reproducible** builds
2. **isolating** a software environment from changes

“Reproducibility”?



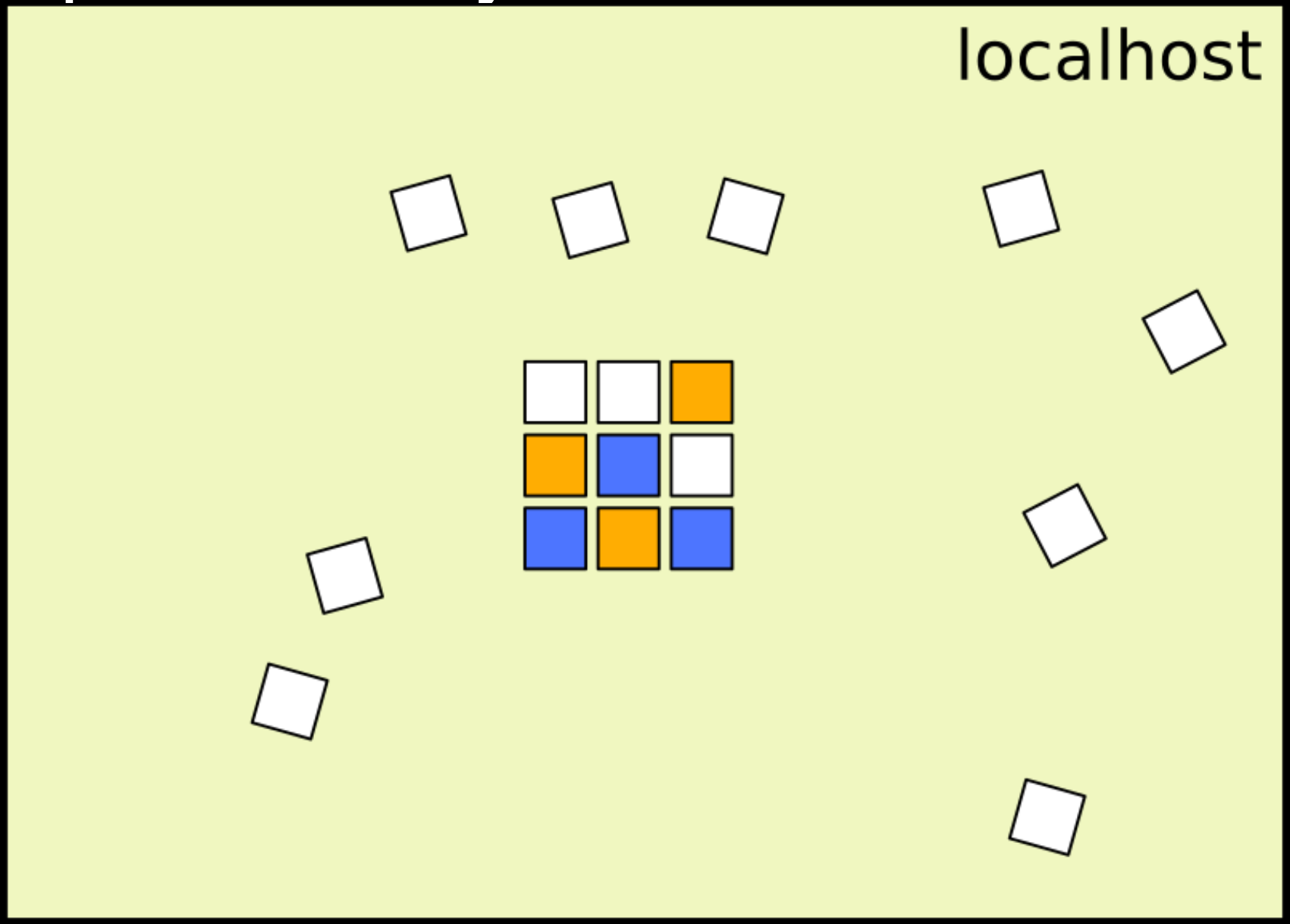
“Reproducibility”?



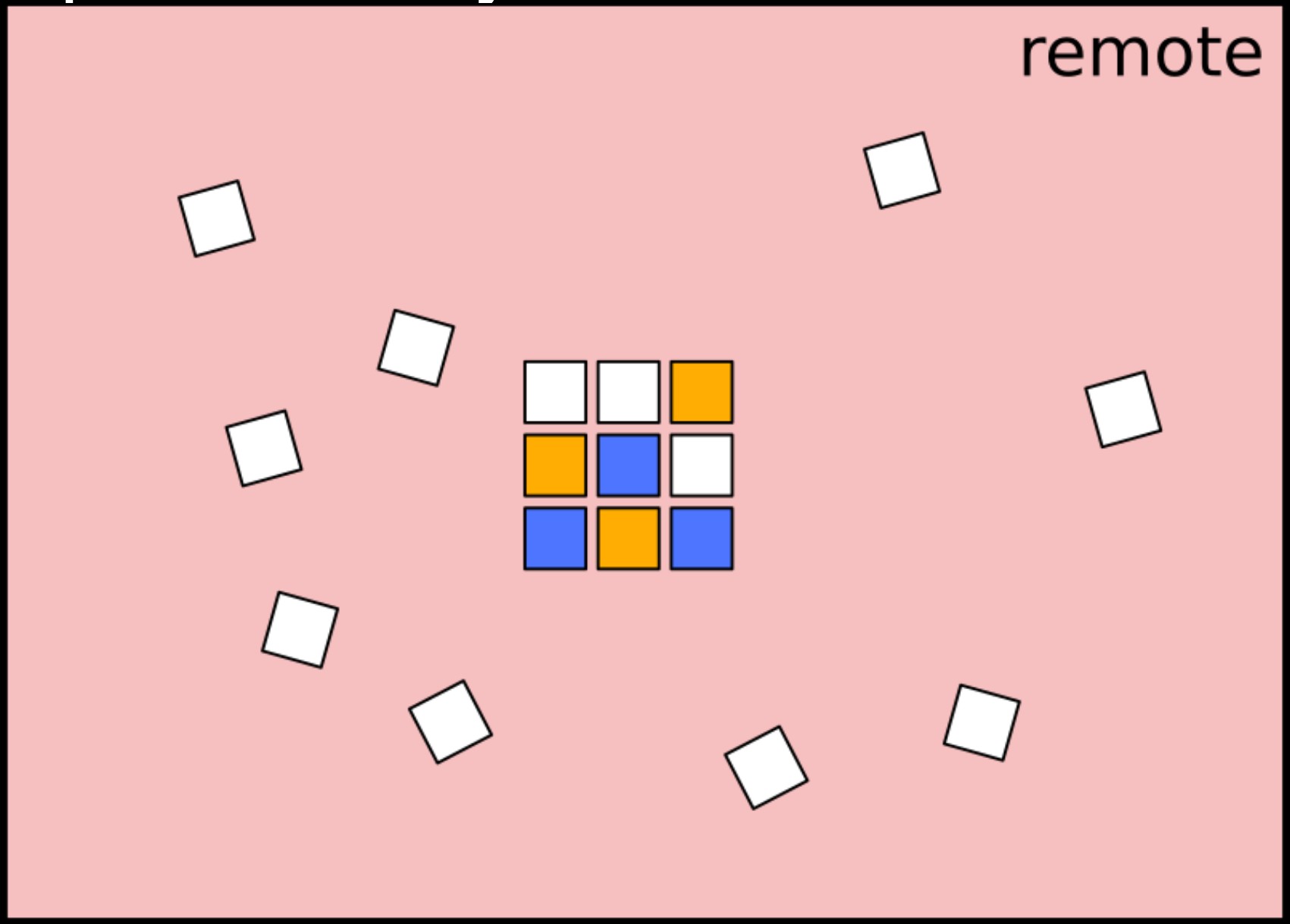
“Reproducibility”?

1. **bit-reproducible** builds
2. **isolating** a software environment from changes
3. **sharing** environments with others

“Reproducibility”?



“Reproducibility”?

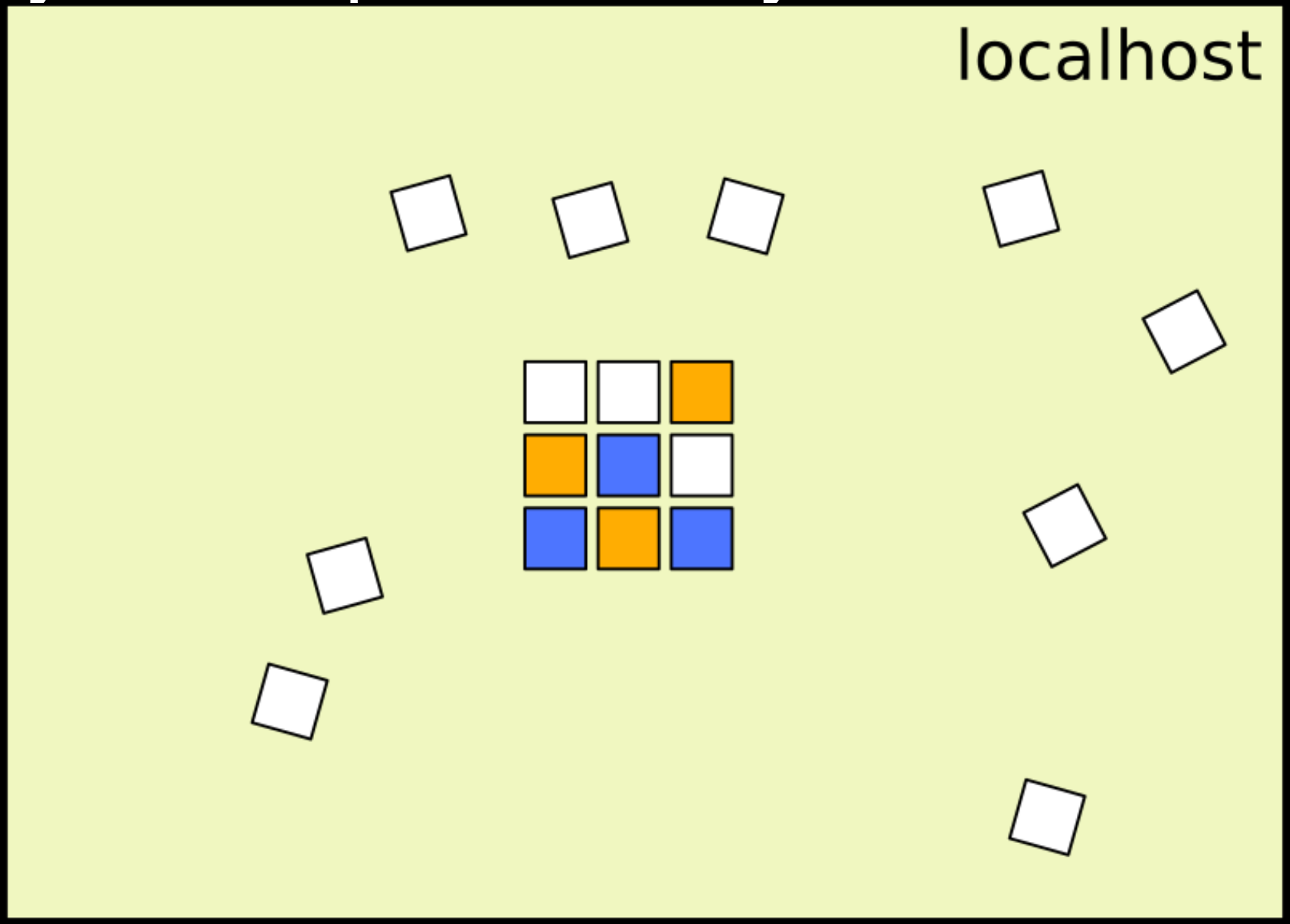


Beyond Reproducibility

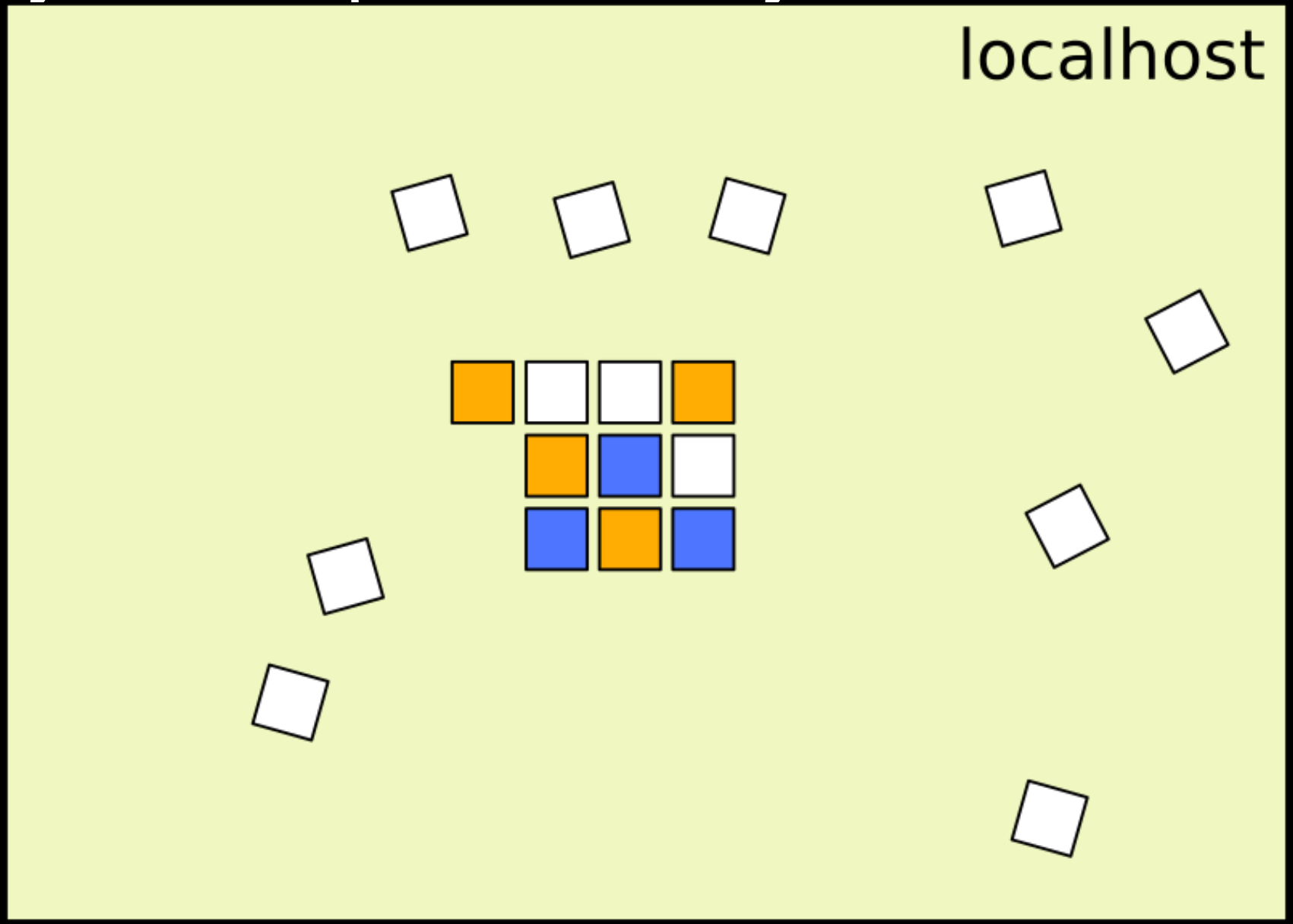
Beyond Reproducibility

- **user-controlled upgrades** and roll-backs

Beyond Reproducibility



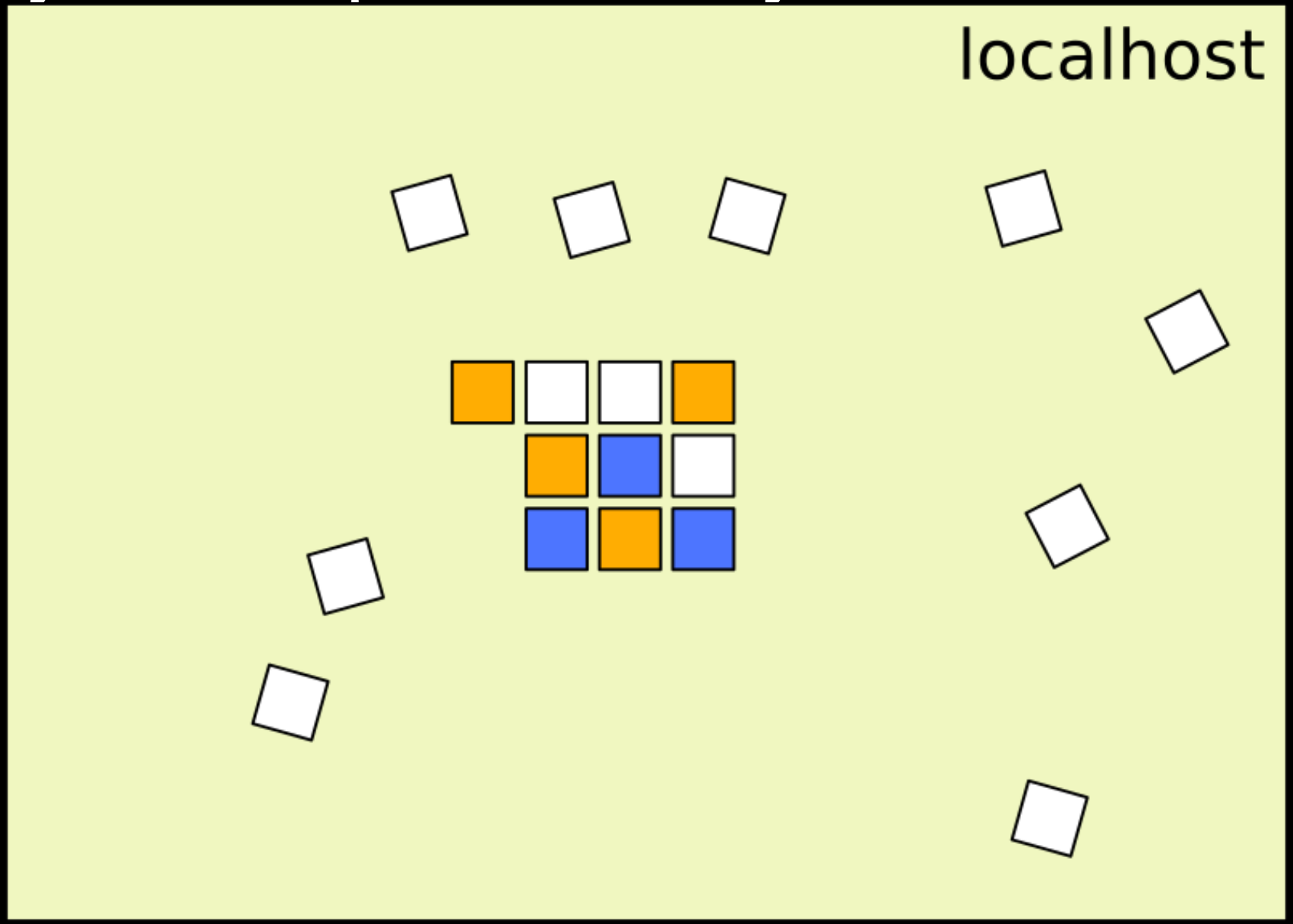
Beyond Reproducibility



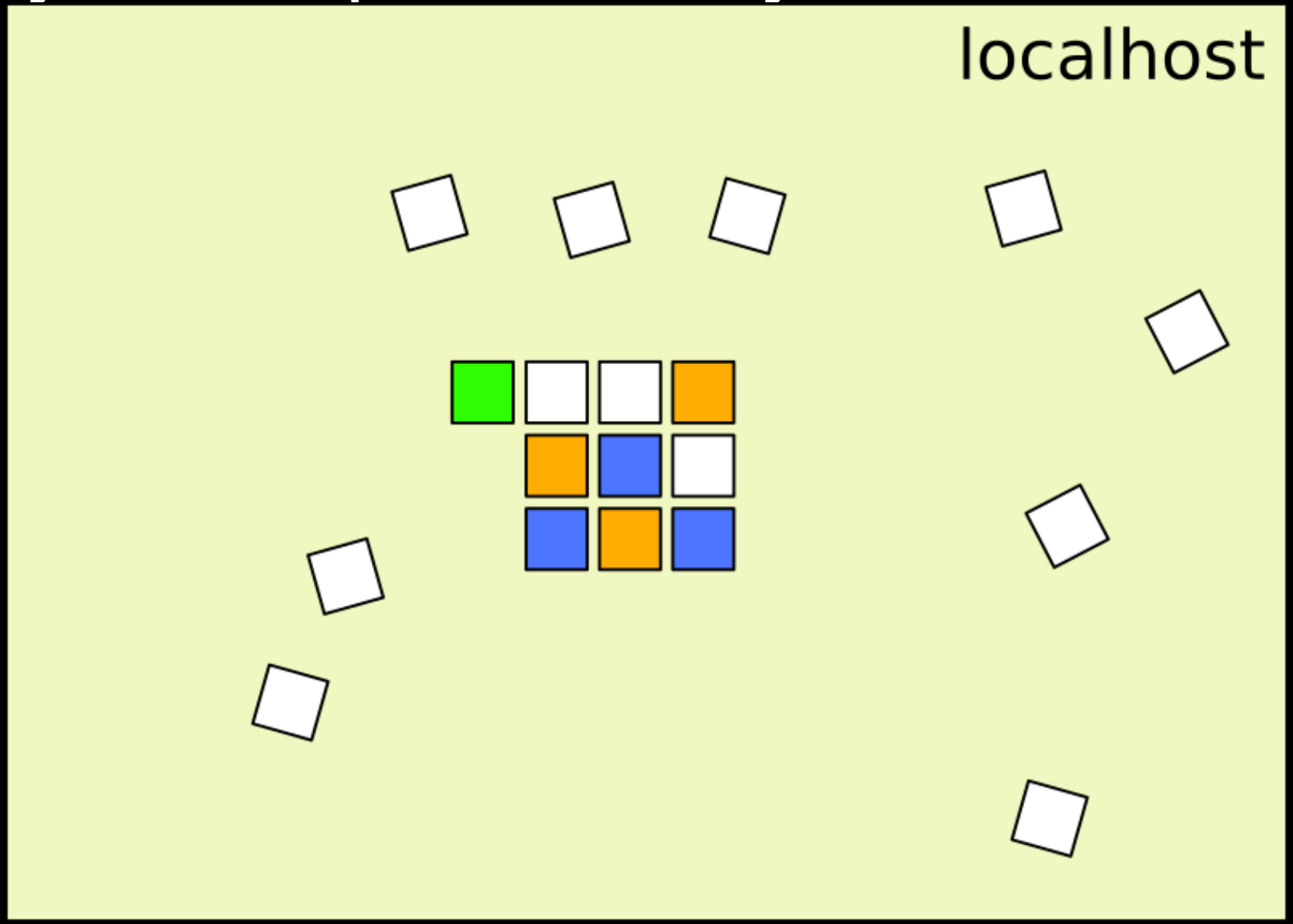
Beyond Reproducibility

- **user-controlled upgrades** and roll-backs
- change **specific parts** of the software stack

Beyond Reproducibility



Beyond Reproducibility



Beyond Reproducibility

- **user-controlled upgrades** and roll-backs
- change **specific parts** of the software stack
- **hackability**: no black boxes

VMs and Docker

VMs and Docker

Pros:

- “bit-reproducible”
- reproducible
  anywhere by anyone

VMs and Docker

Pros:

- “bit-reproducible”
- reproducible anywhere by anyone

Problems:

- VMs are heavyweight
- binary images are opaque
- not composable

Functional Package Management

Functional Package Management

Regarding the build & installation process of a package as a **pure function**.

$$\texttt{openmpi} = f(\texttt{hwloc}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$

where $f = \texttt{./configure \&\& make \&\& make install}$

$$\texttt{openmpi} = f(\texttt{hwloc}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$
$$\texttt{hwloc} = g(\texttt{pciaccess}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$

$$\texttt{openmpi} = f(\texttt{hwloc}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$
$$\texttt{hwloc} = g(\texttt{pciaccess}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$
$$\texttt{gcc} = h(\texttt{make}, \texttt{coreutils}, \texttt{gcc}_0)$$
$$\ldots$$

$$\texttt{openmpi} = f(\texttt{hwloc}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$
$$\texttt{hwloc} = g(\texttt{pciaccess}, \texttt{gcc}, \texttt{make}, \texttt{coreutils})$$
$$\texttt{gcc} = h(\texttt{make}, \texttt{coreutils}, \texttt{gcc}_0)$$
$$\ldots$$

the complete DAG is captured

References

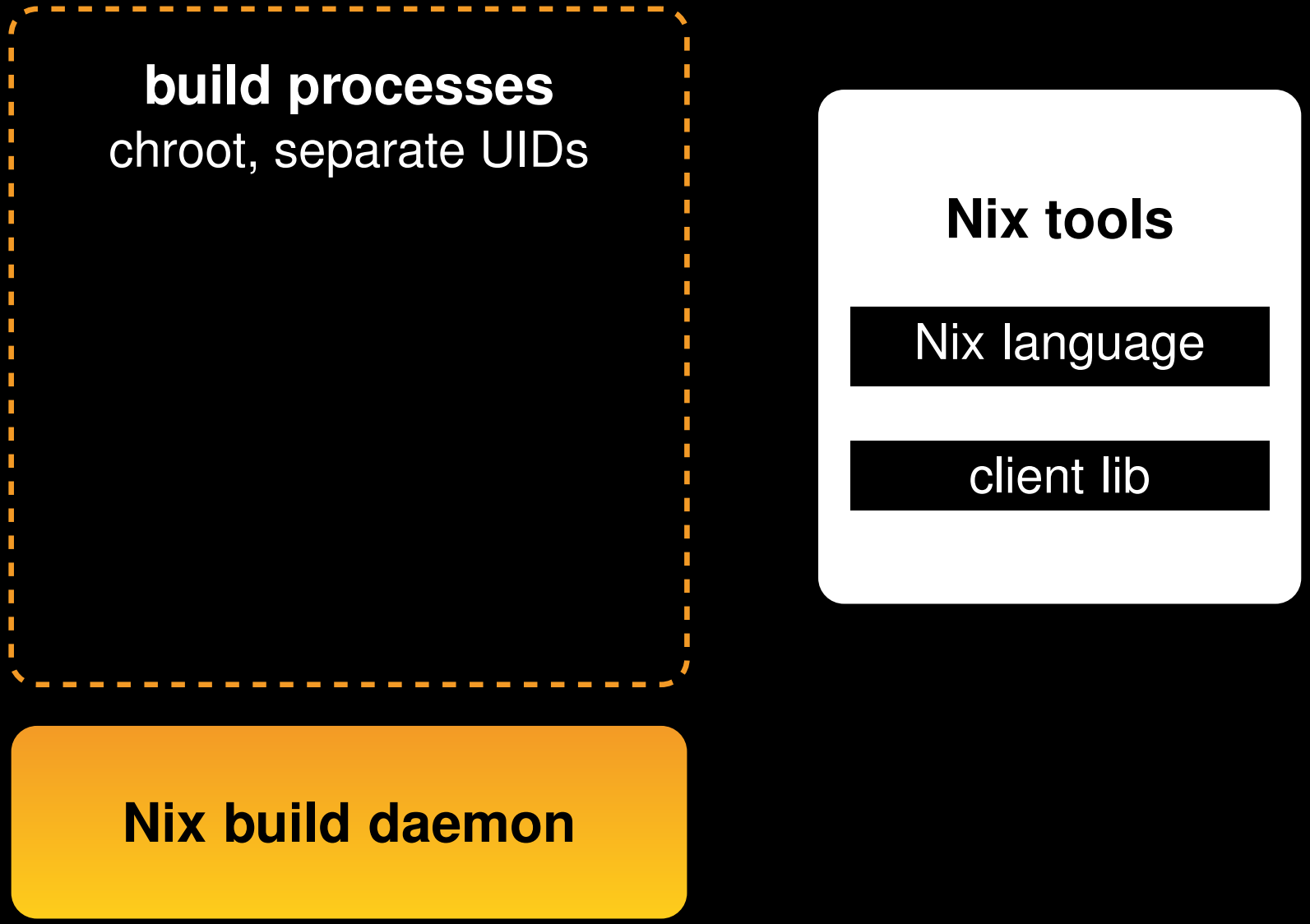
- *A Safe and Policy-Free System for Software Deployment* (“Nix”), Dolstra et al., 2003
- *Functional Package Management with Guix*, Courtès, 2013

Thesis

1. functional package management (FPM) **empowers cluster users**
2. FPM is a solid foundation for **reproducible software deployment**
3. **beyond reproducibility**: Guix is programmable, supports experimentation

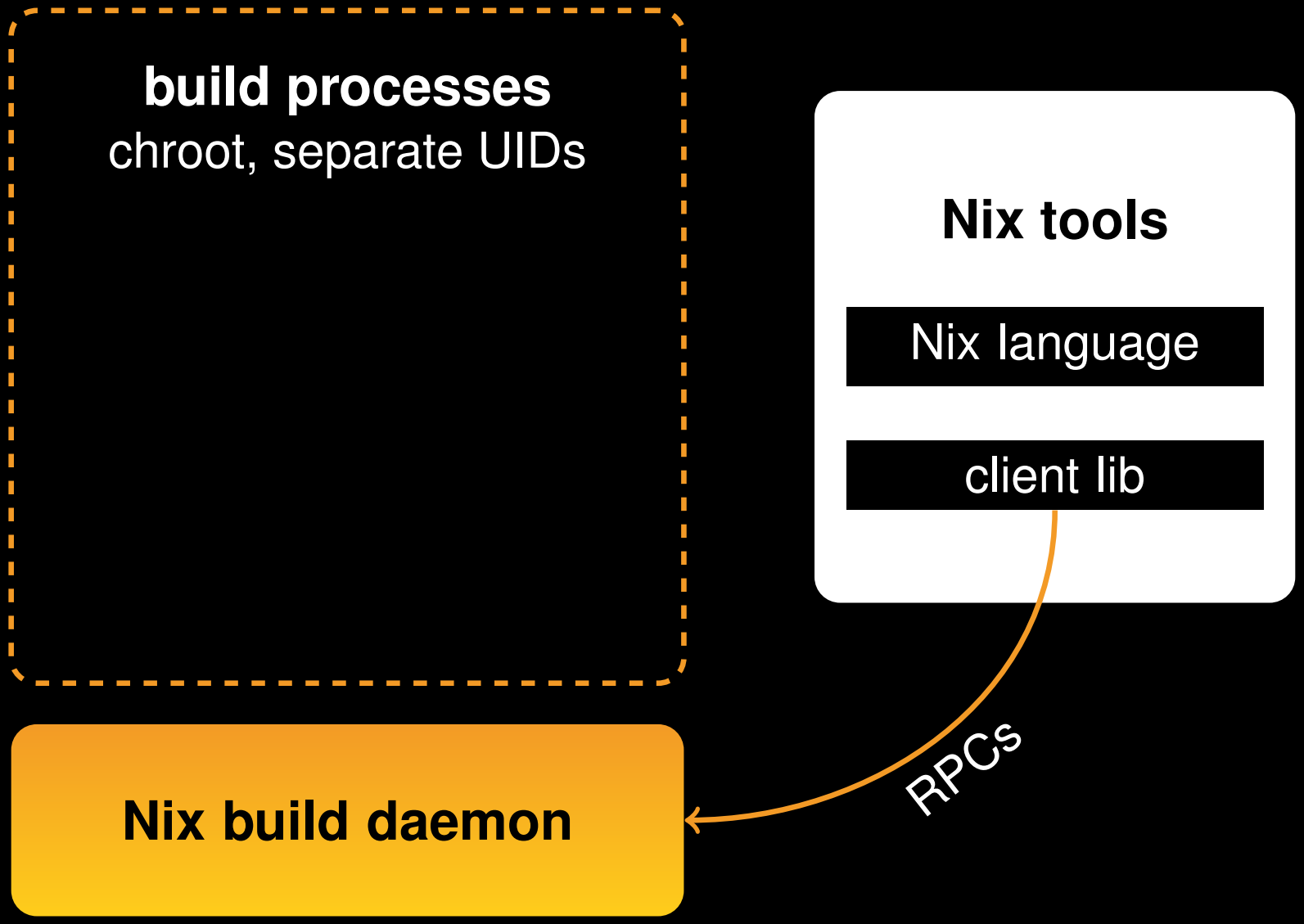
From the Architecture of Nix...

http://nixos.org/nix/



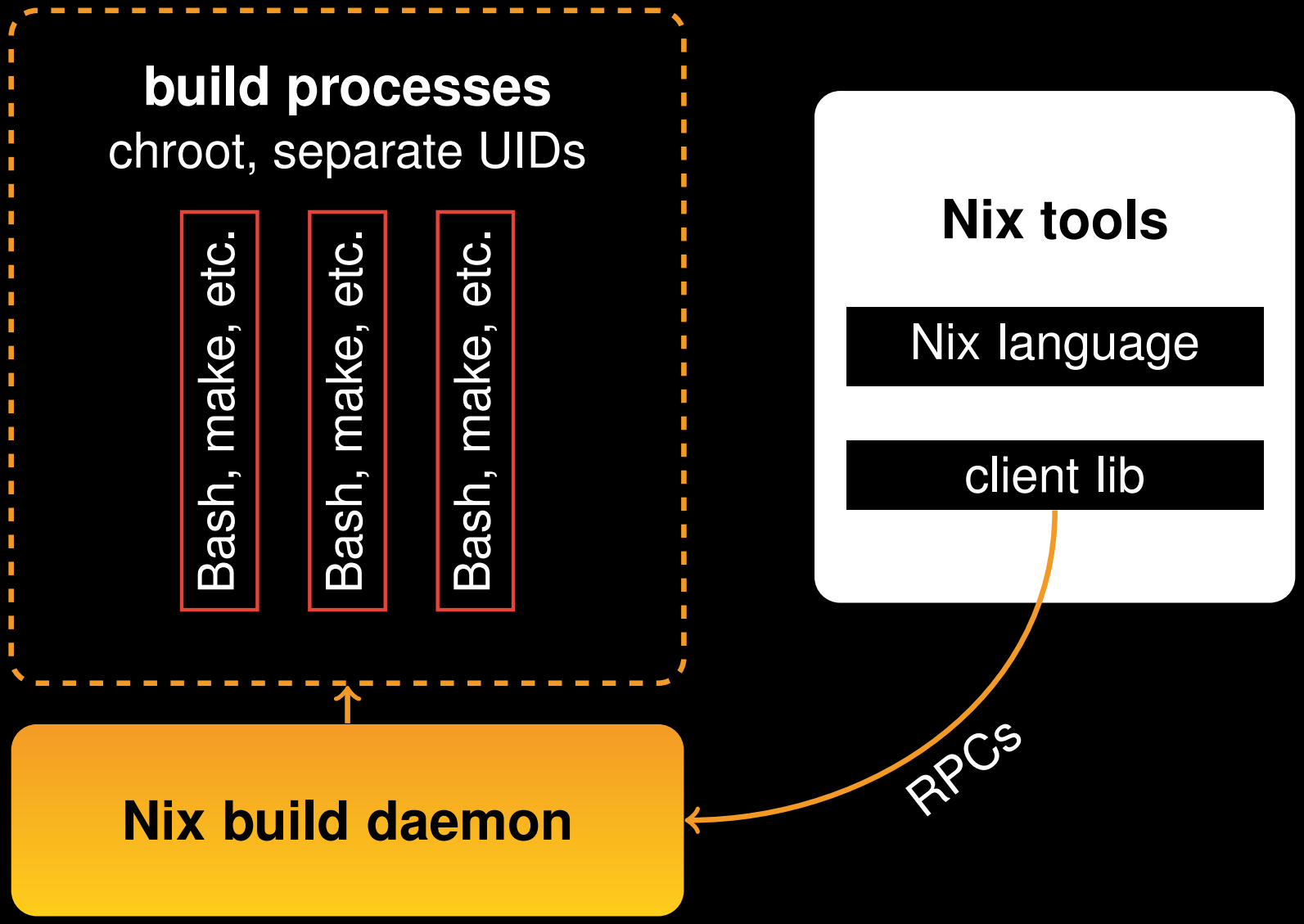
From the Architecture of Nix...

http://nixos.org/nix/



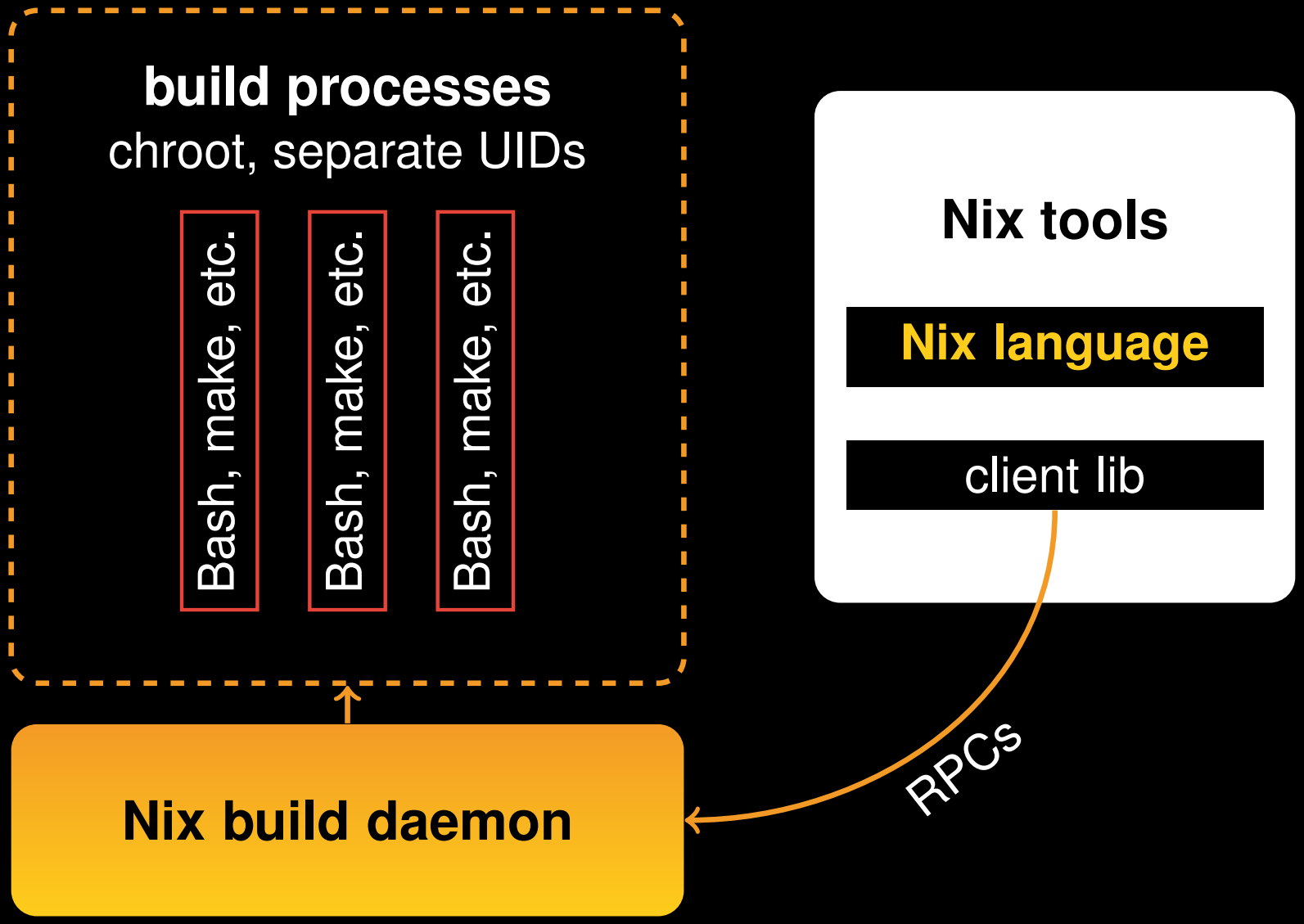
From the Architecture of Nix...

http://nixos.org/nix/



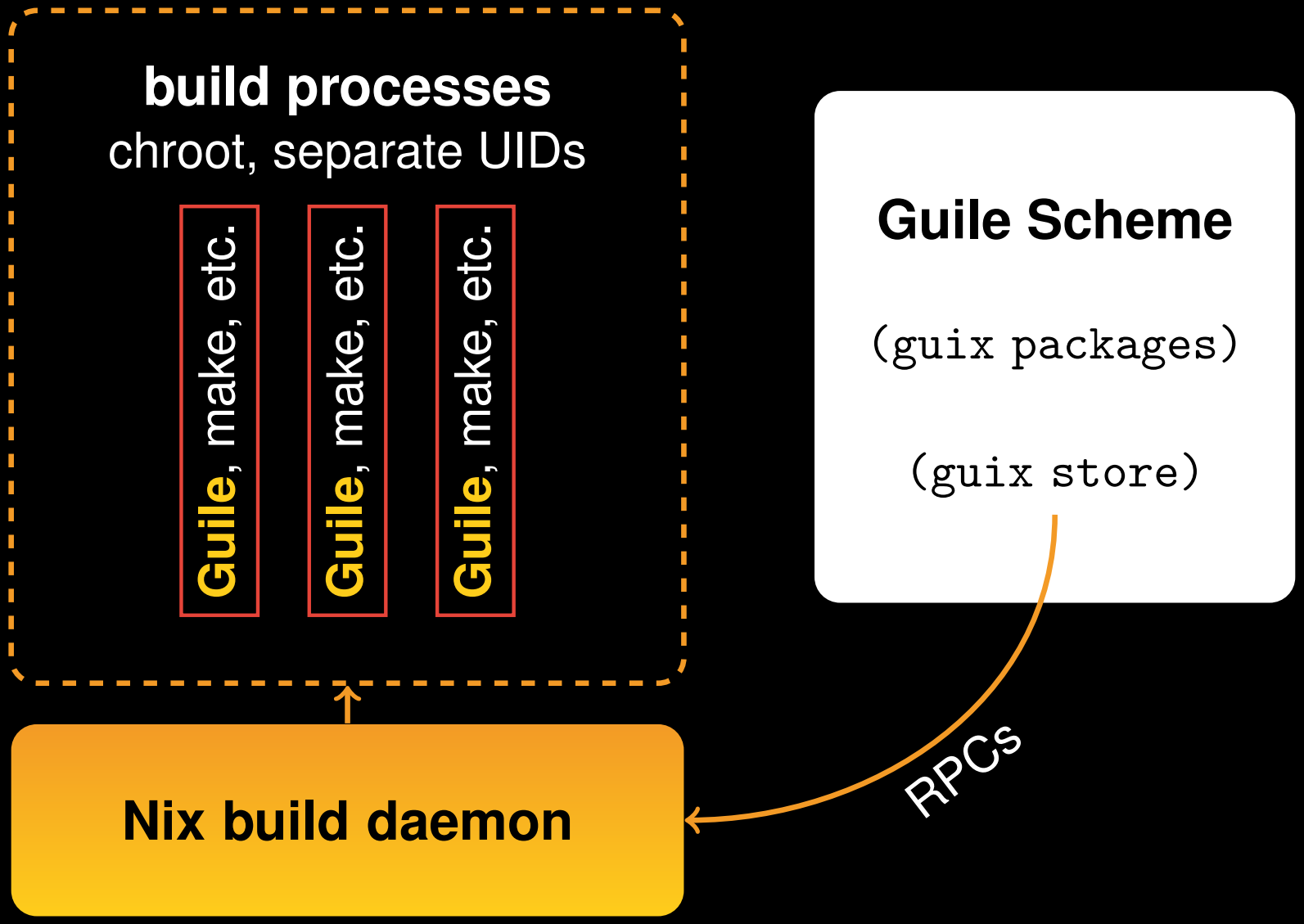
From the Architecture of Nix...

http://nixos.org/nix/



... to the Architecture of Guix

http://gnu.org/s/guix/



Why Guix?

[Courtès 2013]

1. Scheme is a “programmable programming language” $\rightarrow$ **tailored EDSLs**
2. **general-purpose language** with compiler, debugger, libraries, etc.
3. **a single language** $\rightarrow$ more code reuse, unified environment
4. **complete package programming interface**

Bit-Reproducible Builds*

* almost!

```
$ guix build petsc
```

isolated build: chroot, separate name spaces, etc.

Bit-Reproducible Builds*

* almost!

```
$ guix build petsc
/gnu/store/ h2g4sf72... -petsc-3.6.0
```

hash of **all** the dependencies

Bit-Reproducible Builds*

* almost!

```
$ guix build petsc
/gnu/store/ h2g4sf72... -petsc-3.6.0

$ guix gc --references /gnu/store/...-petsc-3.6.0
/gnu/store/...-openmpi-1.8.5
/gnu/store/...-gfortran-4.9.3-lib
/gnu/store/...-superlu-4.3
/gnu/store/...-lapack-3.5.0
/gnu/store/...-glibc-2.21
...
```

Bit-Reproducible Builds*

* almost!

```
$ guix build petsc
/gnu/store/ h2g4sf72... -petsc-3.6.0

$ guix gc --references /gnu/store/...-petsc-3.6.0
/gnu/store/...-openmpi-1.8.5
/gnu/store/...-gfortran-4.9.3-lib
/gnu/store/...-superlu-4.3
/gnu/store/...-lapack-3.5.0
/gnu/store/...-glibc-2.21
...
```

(nearly) bit-identical for everyone

Reproducible Environments

- **per-user “profiles”**
- **non-interference** among users/profiles
- **transactional upgrades & rollbacks**

Reproducible Environments

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

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

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

$ guix environment petsc
…

$ guix environment --ad-hoc python-ipython python-numpy \
    -E ipython
…
```

demo

Experience at the Max Delbrück Center, Berlin

- **Guix deployed on two clusters** (≈250 nodes) + workstations
- used by **bioinformatics** researchers
- **50+ bioinfo packages** in use (C/C++, Python, etc.)
- replaces CentOS packages + sysadmin-managed software

Experience at the Max Delbrück Center, Berlin

- **Guix deployed on two clusters** (≈250 nodes) + workstations
- used by **bioinformatics** researchers
- **50+ bioinfo packages** in use (C/C++, Python, etc.)
- replaces CentOS packages + sysadmin-managed software
- **advantages**: more user control, better resource usage, ...

Beyond Reproducibility: Supporting Experimentation

Fiddling with the HPC Stack

Example from Inria

linear algebra

run-time

PaSTiX, Chameleon (solvers)

StarPU (task scheduling)

hwloc (hardware topology)

MPI (message passing)

compiler's run-time support, C library, etc.

Requirements for an Experimentation-Capable System

1. customize + non-ambiguously **specify package DAG**
2. reliably reproduce **variants of the DAG**

```
(define starpu
  (package
    (name "starpu")
    (version "1.1.4")
    (source (origin
             (method url-fetch)
             (uri "http://...")
             (sha256 (base32 "0zmkw...))))
    (build-system  gnu-build-system )
    (native-inputs `(("pkg-config" ,pkg-config)))
    ( inputs  `(("fftw" ,fftw)
               ("hwloc" , hwloc )))
    (home-page "http://starpu.gforge.inria.fr/")
    (synopsis "Run-time system for heterogeneous computing")
    (description "Blah...")
    (license lgpl2.1+)))
```

```
(define starpu
  (package
    (name "starpu")
    (version "1.1.4")
    (source (origin
              (method url-fetch)
              (uri "http://...")
              (sha256 (base32 "0zmkw...")))))
    (build-system  gnu-build-system )
    (native-inputs `(("pkg-config" ,pkg-config)))
    ( inputs  `(("fftw" ,fftw)
               ("hwloc" , hwloc )))
    (home-page "http://starpu.gforge.inria.fr/")
    (synopsis "Run-time system for heterogeneous computing")
    (description "Blah...")
    (license lgpl2.1+)))
```

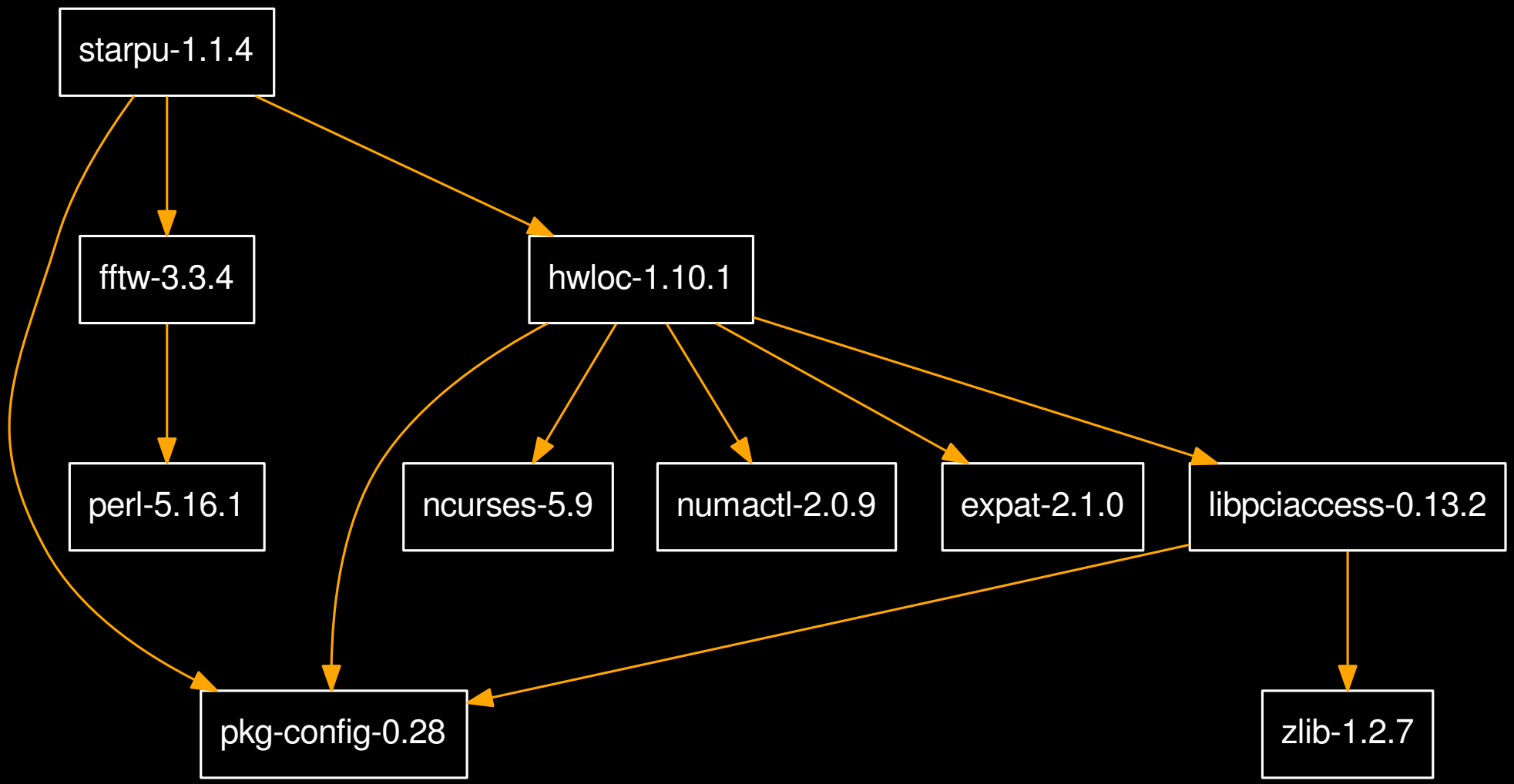
dependencies

```
(define starpu
  (package
    (name "starpu")
    (version "1.1.4")
    (source (origin
              (uri "http://...")
              (sha256 (base32 "0zmkw...")))))
    (build-system  gnu-build-system )
    (native-inputs `(("pkg-config" ,pkg-config)))
    ( inputs  `(("fftw" ,fftw)
               ("hwloc" , hwloc )))
    (home-page "http://starpu.gforge.inria.fr/")
    (synopsis "Run-time system for heterogeneous computing")
    (description "Blah...")
    (license lgpl2.1+)))
```

./configure && make install...

depends on gcc, make, bash, etc.

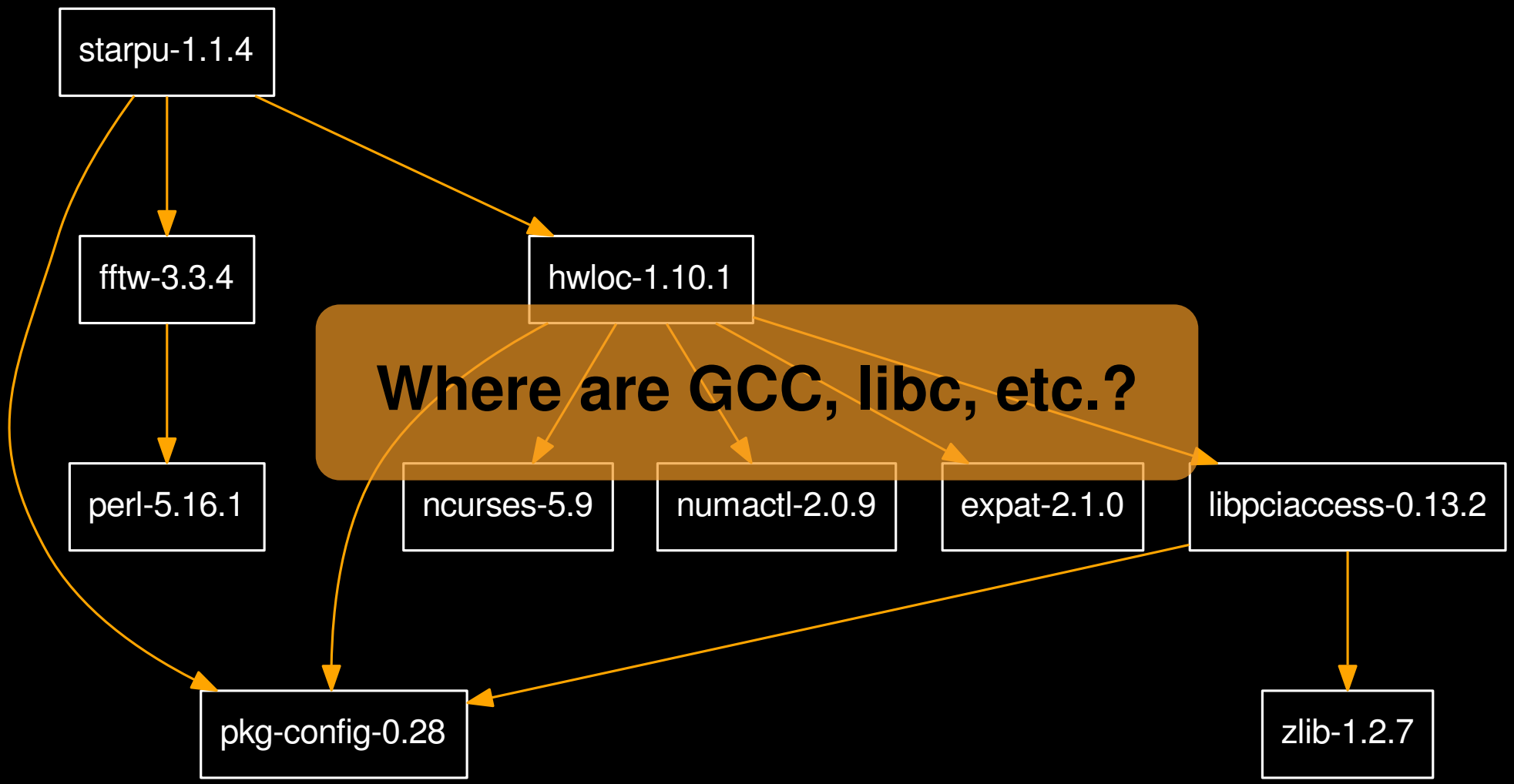
DAG of Package Objects



10 nodes

```
guix graph --type=package starpu
```

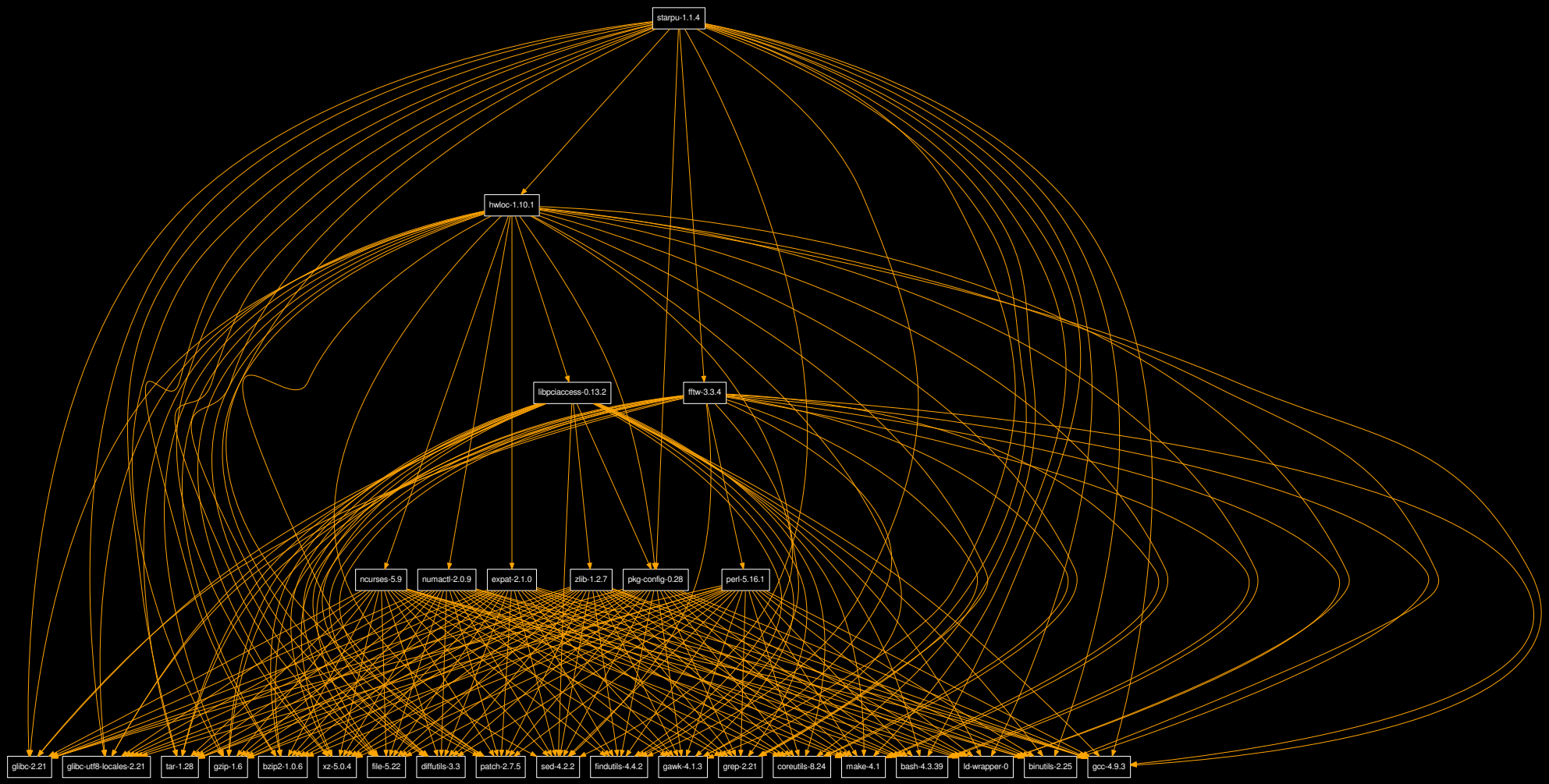
DAG of Package Objects



10 nodes

```
guix graph --type=package starpu
```

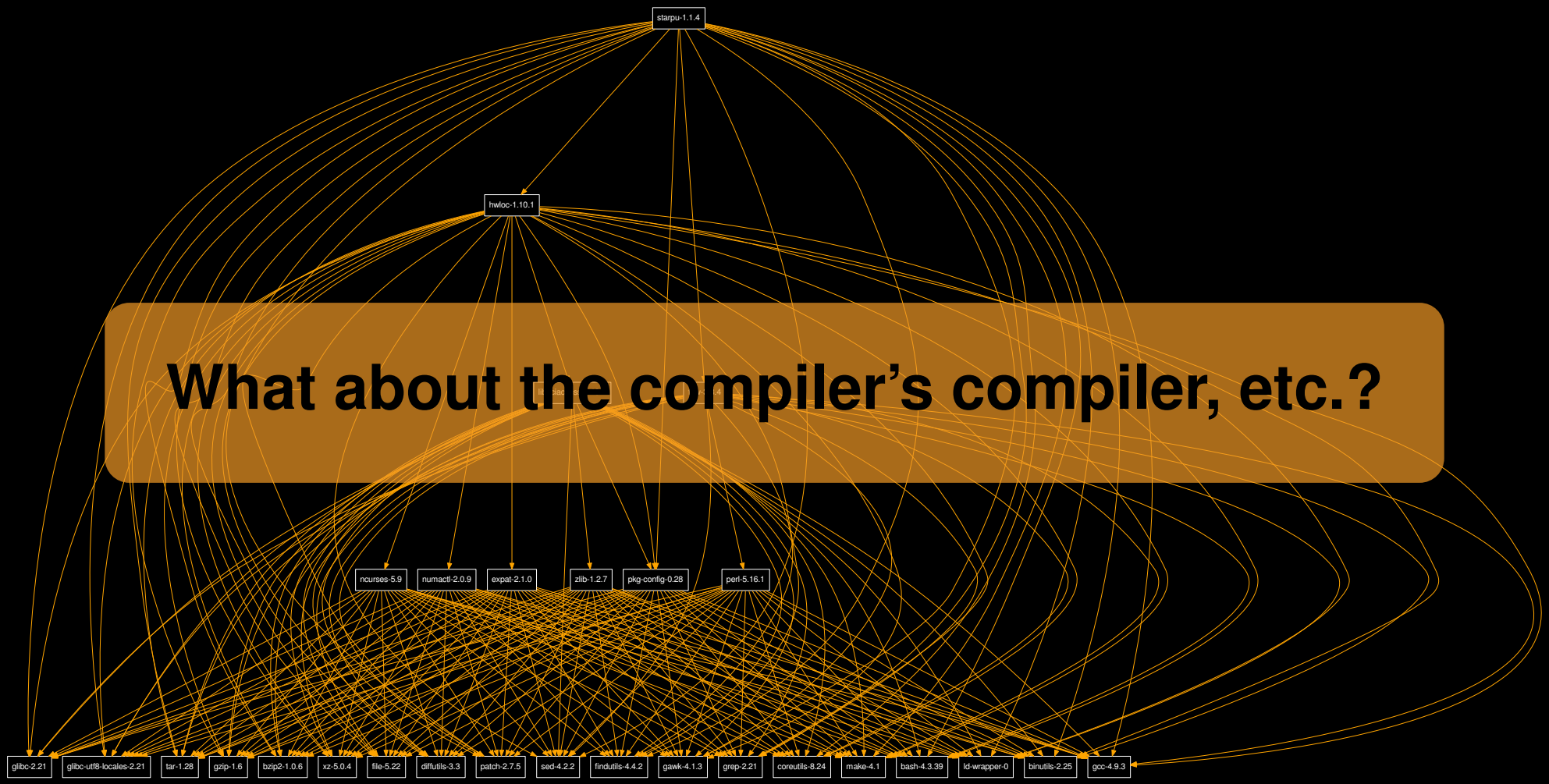
Same DAG, including implicit inputs



29 nodes

```
guix graph --type=bag-emerged starpu
```

Same DAG, including implicit inputs



What about the compiler's compiler, etc.?

29 nodes

```
guix graph --type=bag-emerged starpu
```

Full DAG, including bootstrap

(too big)

321 nodes

```
guix graph --type=bag starpu
```

Defining Package Variants

```
(define starpu-1.2rc                ;release candidate
  (package (inherit starpu)
    (version "1.2.0rc2")
    (source (origin
              (method url-fetch)
              (uri (string-append "http://…/"
                                  "starpu-" version ".tar.gz"))
              (sha256 (base32 "0qgb6y…"))))))
```

Defining Package Variants

```
(define starpu-with-simgrid
  (package (inherit starpu)
    (name "starpu-with-simgrid")

    ;; Add SimGrid, an optional dependency.
    (inputs `(("simgrid" ,simgrid)
              ,@(package-inputs starpu)))))
```

Package Functions

```
(define (make-chameleon starpu )
   ;; Return the Chameleon solver linked against
   ;; this particular variant of StarPU.
   (package
     ;; …
     (inputs `(("starpu" , starpu )
               ("blas" ,atlas)
               ("lapack" ,lapack)
               ("gfortran" ,gfortran-4.8)
               ("python" ,python-2)))))
```

Package Functions

```
(define (make-chameleon  starpu )
   ;; Return the Chameleon solver linked against
   ;; this particular variant of StarPU.
   (package
     ;; …
     (inputs `(("starpu" , starpu )
               ("blas" ,atlas)
               ("lapack" ,lapack)
               ("gfortran" ,gfortran-4.8)
               ("python" ,python-2)))))

 (define chameleon
   (make-chameleon starpu))
 (define chameleon/starpu-simgrid
   (make-chameleon starpu-with-simgrid))
```

Conclusion

Limitations

- daemon must run as root to isolate builds
- non-deterministic build systems
- non-free software unavailable in Guix

Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on **user name spaces** (Linux 3.8+)
- non-deterministic build systems
- non-free software unavailable in Guix

Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on **user name spaces** (Linux 3.8+)
- non-deterministic build systems
  - must be identified & **fixed upstream**
  - WIP thanks to `http://reproducible.debian.net`
- non-free software unavailable in Guix

Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on **user name spaces** (Linux 3.8+)
- non-deterministic build systems
  - must be identified & **fixed upstream**
  - WIP thanks to `http://reproducible.debian.net`
- non-free software unavailable in Guix
  - **would you do chemistry research out of magic potions?**

Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on **user name spaces** (Linux 3.8+)
- non-deterministic build systems
  - must be identified & **fixed upstream**
  - WIP thanks to `http://reproducible.debian.net`
- non-free software unavailable in Guix
  - would you do chemistry research out of magic potions?
  - **reproducible research demands free software**

Summary

- Guix allows **cluster users** to reproduce environments

Summary

- Guix allows **cluster users** to reproduce environments
- it provides **the source of software environments**, not just the bits

Summary

- Guix allows **cluster users** to reproduce environments
- it provides **the source of software environments**, not just the bits
- **composability, transparency, and hackability** of software stacks are key to reproducible research

Guix

http://gnu.org/software/guix/