Software deployment with Nix

NLUUG meeting, 31/03/2006

Eelco Dolstra

eelco@cs.uu.nl

Universiteit Utrecht, Faculty of Science,
Department of Information and Computing Sciences

March 31, 2006

Overview

Nix

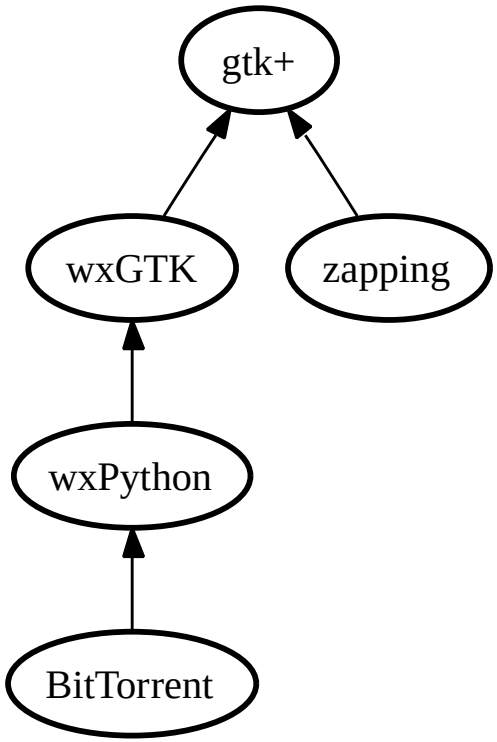
- Motivation: Existing package managers (like RPM, **apt**, ...) have lots of problems
- We set out to improve this
- Result: Nix — basically a package manager (but more!)

Deployment Problems

- Software deployment: the art of **transferring software** (packages) from one machine to another (and managing it).
- The hard part: packages should **work the same** on the target machine.
  - "DLL hell"
  - "Dependency hell"

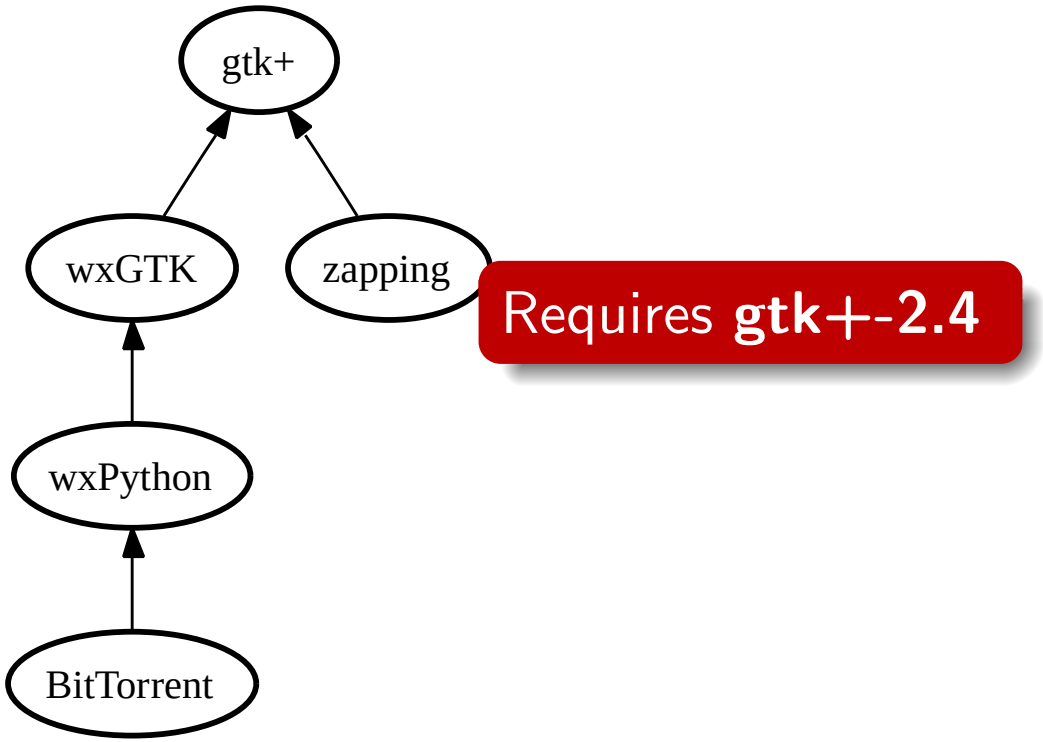
So why is this hard?

- Difficult to have multiple versions; but we want this to
  - Test upgrades
  - Deal with conflicting dependencies
  - Support different user / service requirements



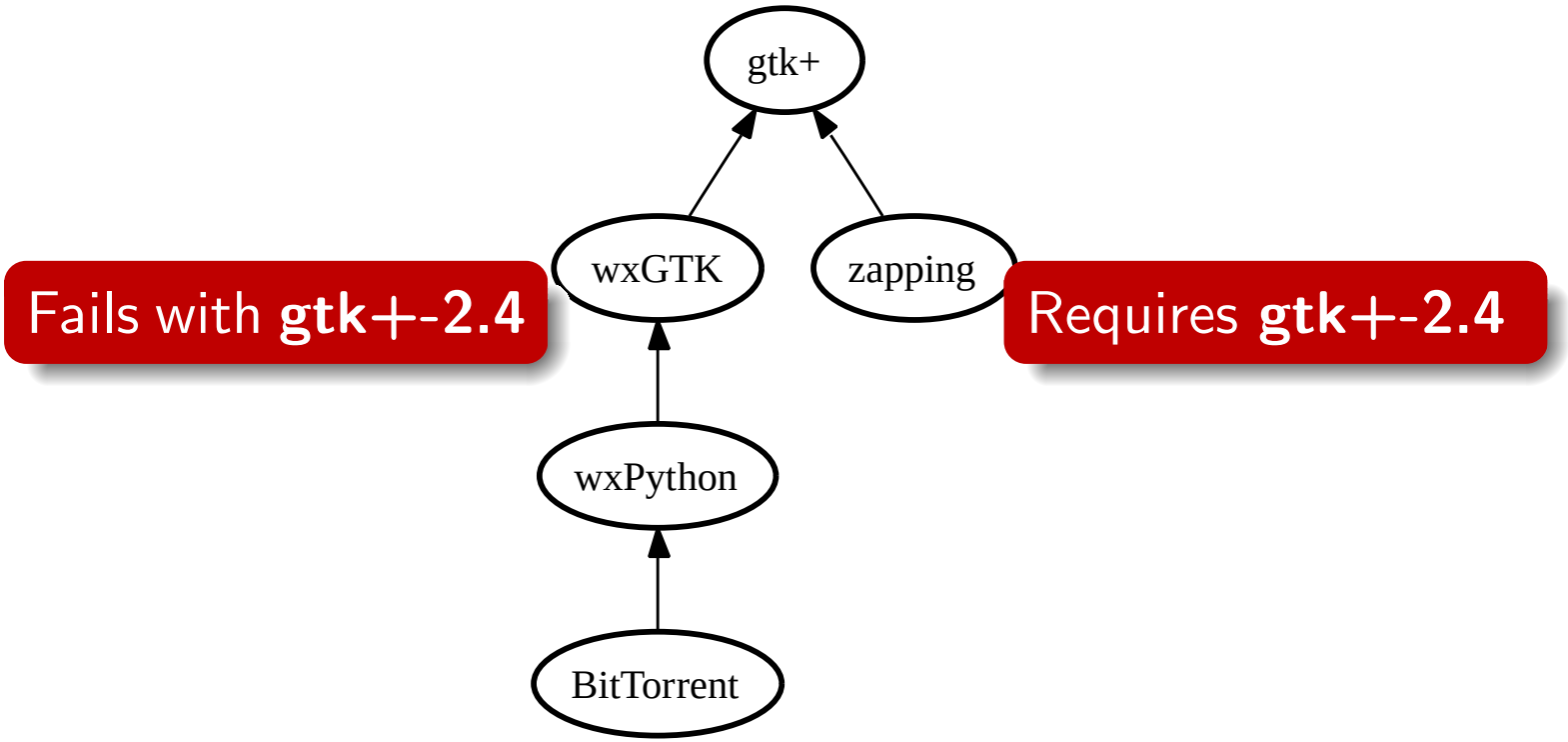
So why is this hard?

- Difficult to have multiple versions; but we want this to
  - Test upgrades
  - Deal with conflicting dependencies
  - Support different user / service requirements



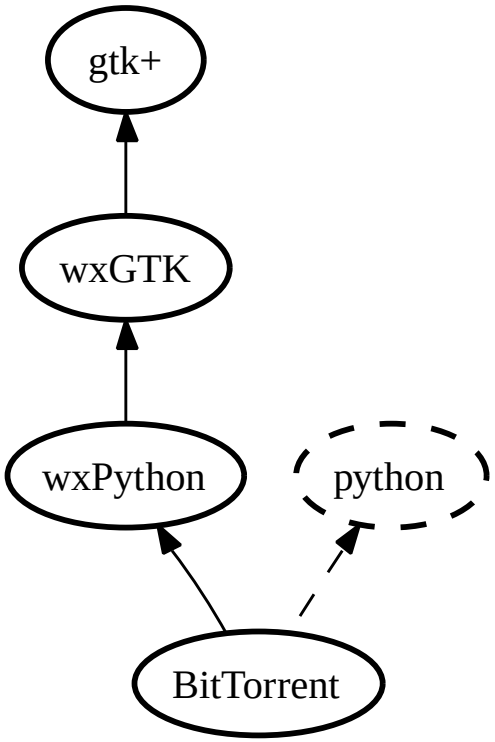
So why is this hard?

- Difficult to have multiple versions; but we want this to
  - Test upgrades
  - Deal with conflicting dependencies
  - Support different user / service requirements



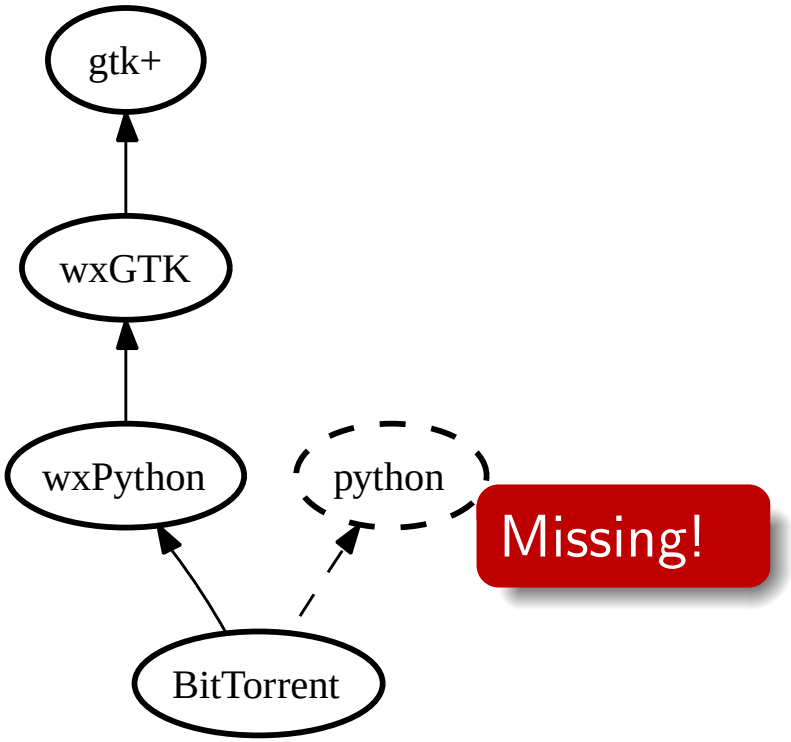
So why is this hard?

- Unreliable dependency information
  - What components are needed?
  - What versions?

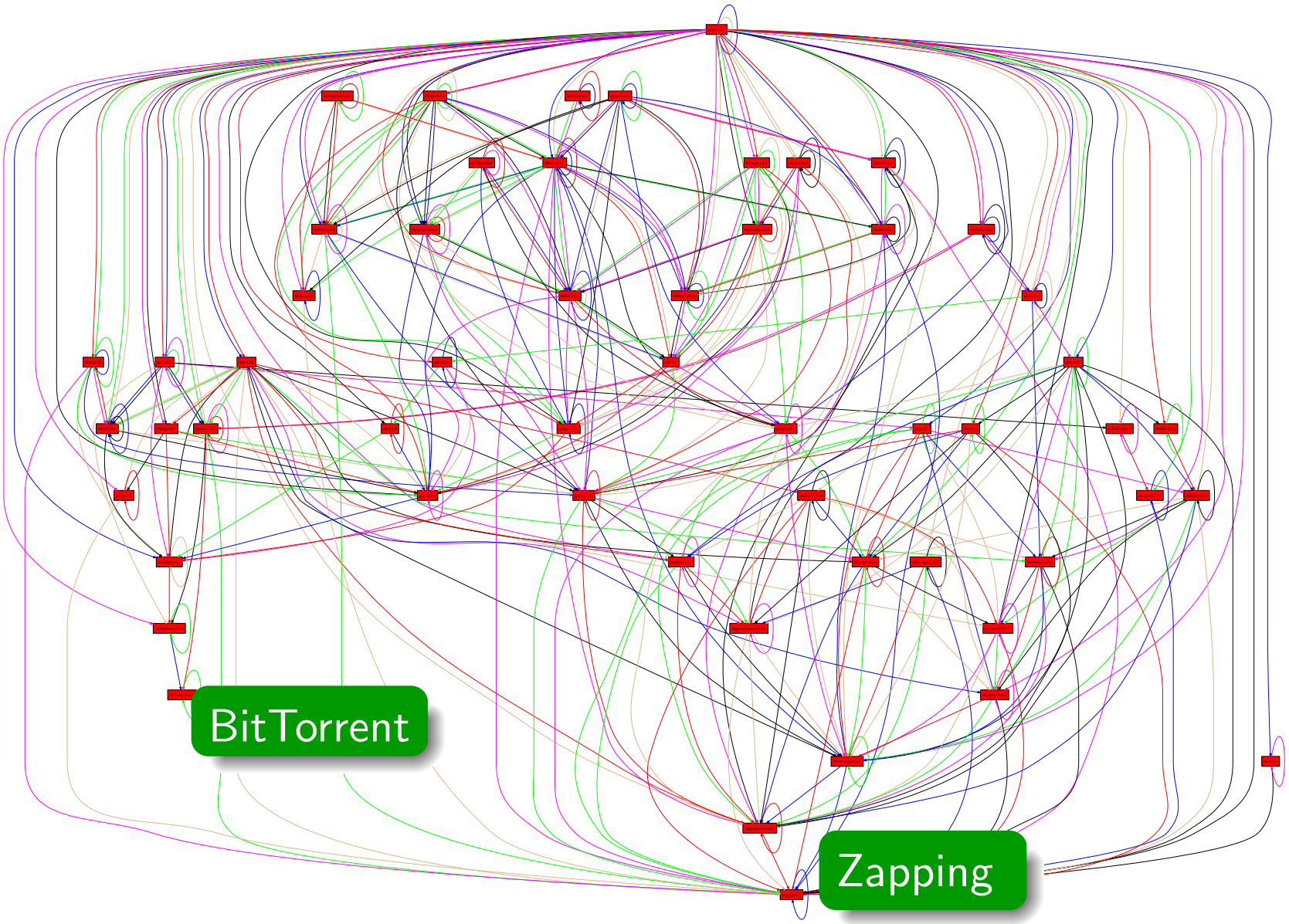


So why is this hard?

- Unreliable dependency information
  - What components are needed?
  - What versions?



So why is this hard?



Tool Support

- Deployment was (is) often done in an *ad hoc*, undisciplined fashion.
  - Files installed in global locations (**/usr/bin**, **C:/Windows/System32**).
  - "DLL Hell" — overwriting of shared components with older/newer versions.
  - "Dependency Hell" — components may have gazillions of dependencies.
  - Each application has its own (un)installer (so no unified view on the system).
  - Interactive installers ⇒ considered harmful (hard to automate).
  - Packaging = lots of work.
- Package managers manage software installations in a unified way: RPM, FreeBSD Ports/Packages, Depot, Debian apt-get/dpkg, ..., Nix.

Requirements on a Deployment System

- Support multiple versions, variants.
- Handle dependencies.
- Ensure safe upgrades / uninstalls.
- Atomic upgrades/downgrades (e.g., important in server environments).
- Allow different "views" for multiple users.
- Configuration management aspects:
  - Identification
  - Reproducibility

The Nix Deployment System

- Central idea: store all components in isolation.
- Unique paths:

```
/nix/store/jjp9pirx8b3nqs9k...-firefox
```

  which is an 160-bit **cryptographic hash** of **all** inputs used to build the component:

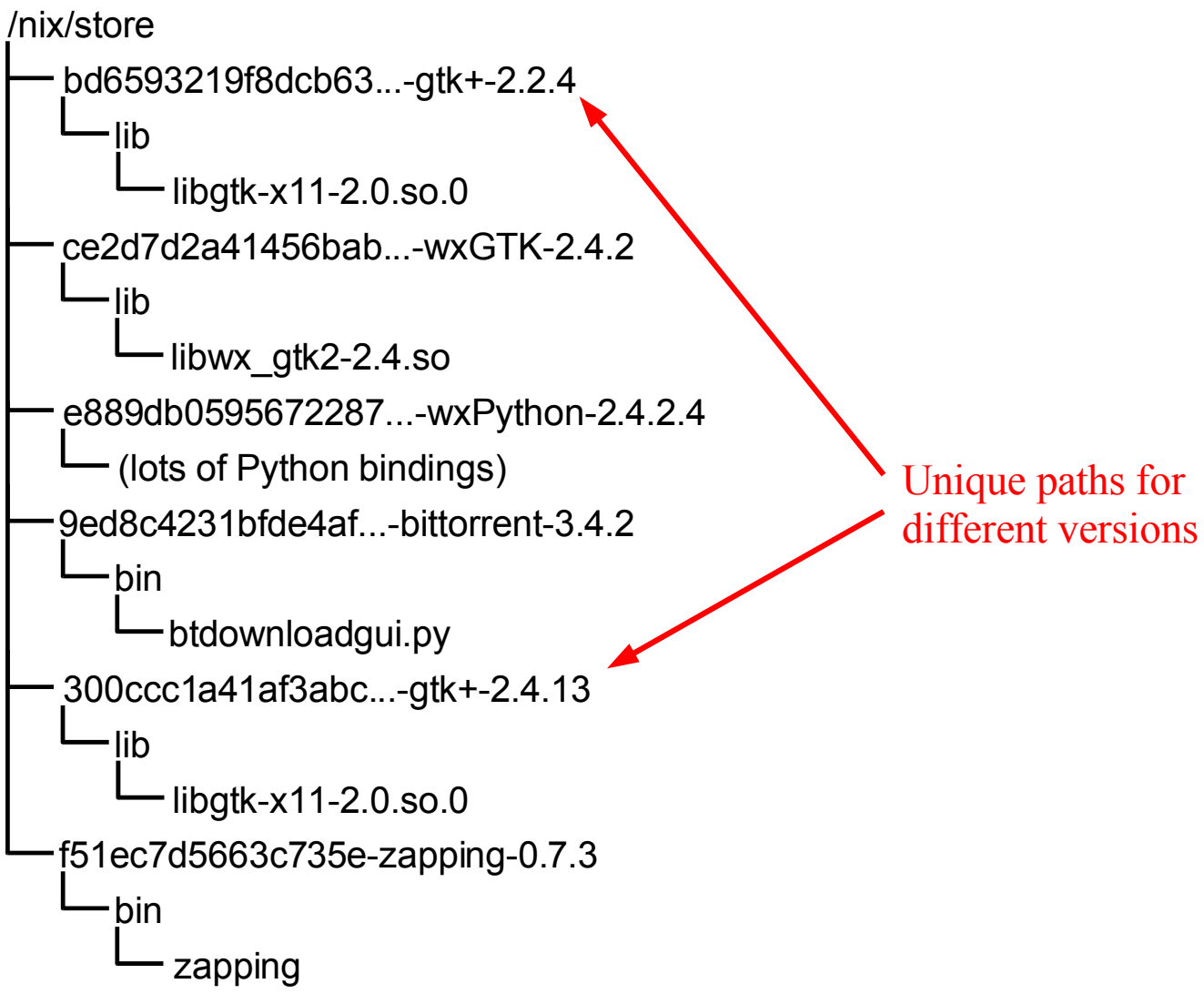
  - Sources
  - Libraries
  - Compilers
  - Build scripts
  - Build parameters
  - System type
  - ...

- **Prevent** undeclared **build time** dependencies.
- **Scan** for **runtime** dependencies.
- Deploy only **closures** under the **depends-on** relation.

Nix store

```
/nix/store
├── bd6593219f8dcb63...-gtk+-2.2.4
│   └── lib
│       └── libgtk-x11-2.0.so.0
├── ce2d7d2a41456bab...-wxGTK-2.4.2
│   └── lib
│       └── libwx_gtk2-2.4.so
├── e889db0595672287...-wxPython-2.4.2.4
│   └── (lots of Python bindings)
├── 9ed8c4231bfde4af...-bittorrent-3.4.2
│   └── bin
│       └── btdownloadgui.py
├── 300ccc1a41af3abc...-gtk+-2.4.13
│   └── lib
│       └── libgtk-x11-2.0.so.0
└── f51ec7d5663c735e-zapping-0.7.3
    └── bin
        └── zapping
```

Nix store


/nix/store
bd6593219f8dcb63...-gtk+-2.2.4
lib
libgtk-x11-2.0.so.0
ce2d7d2a41456bab...-wxGTK-2.4.2
lib
libwx_gtk2-2.4.so
e889db0595672287...-wxPython-2.4.2.4
(lots of Python bindings)
9ed8c4231bfde4af...-bittorrent-3.4.2
bin
btdownloadgui.py
300ccc1a41af3abc...-gtk+-2.4.13
lib
libgtk-x11-2.0.so.0
f51ec7d5663c735e-zapping-0.7.3
bin
zapping
Unique paths for different versions

Nix expressions

hello/default.nix

Packages are built using *Nix expressions*:

```
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url =
      ftp://ftp.gnu.org/pub/gnu/hello/hello-2.1.1.tar.gz;
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```

Nix expressions

hello/default.nix

Packages are built using *Nix expressions*:

```
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url =
      ftp://ftp.gnu.org/pub/gnu/hello/hello-2.1.1.tar.gz;
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```

Function arguments

Nix expressions

hello/default.nix

Packages are built using *Nix expressions*:

```
{stdenv, fetchurl, perl}:

stdenv.mkDerivation {
  name = "hello-2.1.1";
  builder = ./builder.sh;
  src = fetchurl {
    url =
      ftp://ftp.gnu.org/pub/gnu/hello/hello-2.1.1.tar.gz;
    md5 = "70c9ccf9fac07f762c24f2df2290784d";
  };
  inherit perl;
}
```

Function arguments

Build attributes

Nix expressions

hello/builder.sh

```
source $stdenv/setup

PATH=$perl/bin:$PATH

tar xvfz $src
cd hello-*
./configure --prefix=$out
make
make install
```

Nix expressions

hello/builder.sh

```
source $stdenv/setup

PATH=$perl/bin:$PATH

tar xvfz $src
cd hello-*
./configure --pref
make
make install
```

Environment initially empty; prevents undeclared dependencies

Nix expressions

system/all-packages.nix

```
hello = (import ../applications/misc/hello/ex-1) {
  inherit fetchurl stdenv perl;
};

perl = (import ../development/interpreters/perl) {
  inherit fetchurl stdenv;
};

fetchurl = (import ../build-support/fetchurl) {
  inherit stdenv; ...
};

stdenv = ...;
```

Nix expressions

system/all-packages.nix

```
hello = (import ../applications/misc/hello/ex-1) {
  inherit fetchurl stdenv perl;
};

perl = (import ../development/interpreters/perl) {
  inherit fetchurl stdenv;
};

fetchurl = (import ../build-support/fetchurl) {
  inherit stdenv; ...
};

stdenv = ...;
```

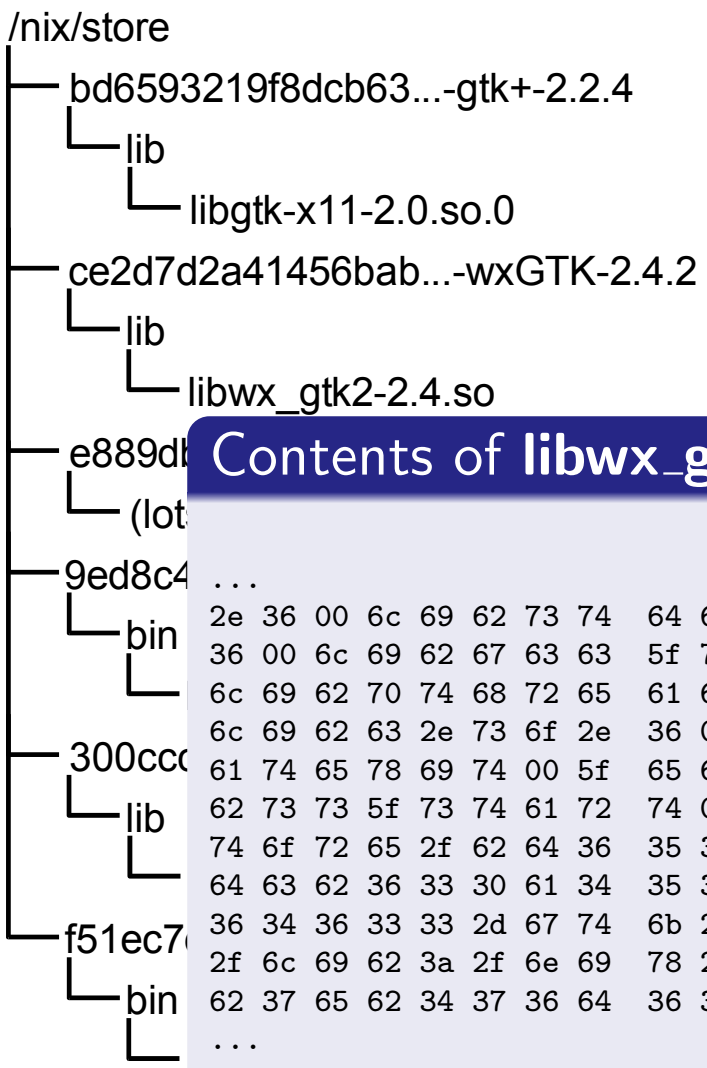
Variability

```
bittorrent = (import ../tools/networking/bittorrent) {
  inherit fetchurl stdenv wxGTK;
};

wxGTK = (import ../development/libraries/wxGTK) {
  inherit fetchurl stdenv pkgconfig;
  gtk = gtkLibs22.gtk;
};

firefox = (import ../applications/browsers/firefox) {
  inherit fetchurl stdenv pkgconfig perl zip libIDL libXi;
  gtk = gtkLibs24.gtk;
};
```

Finding runtime dependencies

```
/nix/store
├── bd6593219f8dcb63...-gtk+-2.2.4
│   └── lib
│       └── libgtk-x11-2.0.so.0
├── ce2d7d2a41456bab...-wxGTK-2.4.2
│   └── lib
│       └── libwx_gtk2-2.4.so
├── e889db0595672287...-wxPython-2.4.2.4
│   └── (lots of Python bindings)
├── 9ed8c4231bfde4af...-bittorrent-3.4.2
│   └── bin
│       └── btdownloadgui.py
├── 300ccc1a41af3abc...-gtk+-2.4.13
│   └── lib
│       └── libgtk-x11-2.0.so.0
└── f51ec7d5663c735e-zapping-0.7.3
    └── bin
        └── zapping
```

Finding runtime dependencies



Contents of **libwx_gtk2-2.4.so**

```
...
2e 36 00 6c 69 62 73 74  64 63 2b 2b 2e 73 6f 2e  |.6.libstdc++.so.|
36 00 6c 69 62 67 63 63  5f 73 2e 73 6f 2e 31 00  |6.libgcc_s.so.1.|
6c 69 62 70 74 68 72 65  61 64 2e 73 6f 2e 30 00  |libpthread.so.0.|
6c 69 62 63 2e 73 6f 2e  36 00 5f 5f 63 78 61 5f  |libc.so.6.__cxa_|
61 74 65 78 69 74 00 5f  65 64 61 74 61 00 5f 5f  |atexit._edata.__|
62 73 73 5f 73 74 61 72  74 00 2f 6e 69 78 2f 73  |bss_start./nix/s|
74 6f 72 65 2f 62 64 36  35 39 33 32 31 39 66 38  |tore/bd6593219f8|
64 63 62 36 33 30 61 34  35 35 62 31 61 35 37 66  |dcb630a455b1a57f|
36 34 36 33 33 2d 67 74  6b 2b 2d 32 2e 32 2e 34  |64633-gtk+-2.2.4|
2f 6c 69 62 3a 2f 6e 69  78 2f 73 74 6f 72 65 2f  |/lib:/nix/store/|
62 37 65 62 34 37 36 64  36 32 62 61 65 38 62 63  |b7eb476d62bae8bc|
...
```

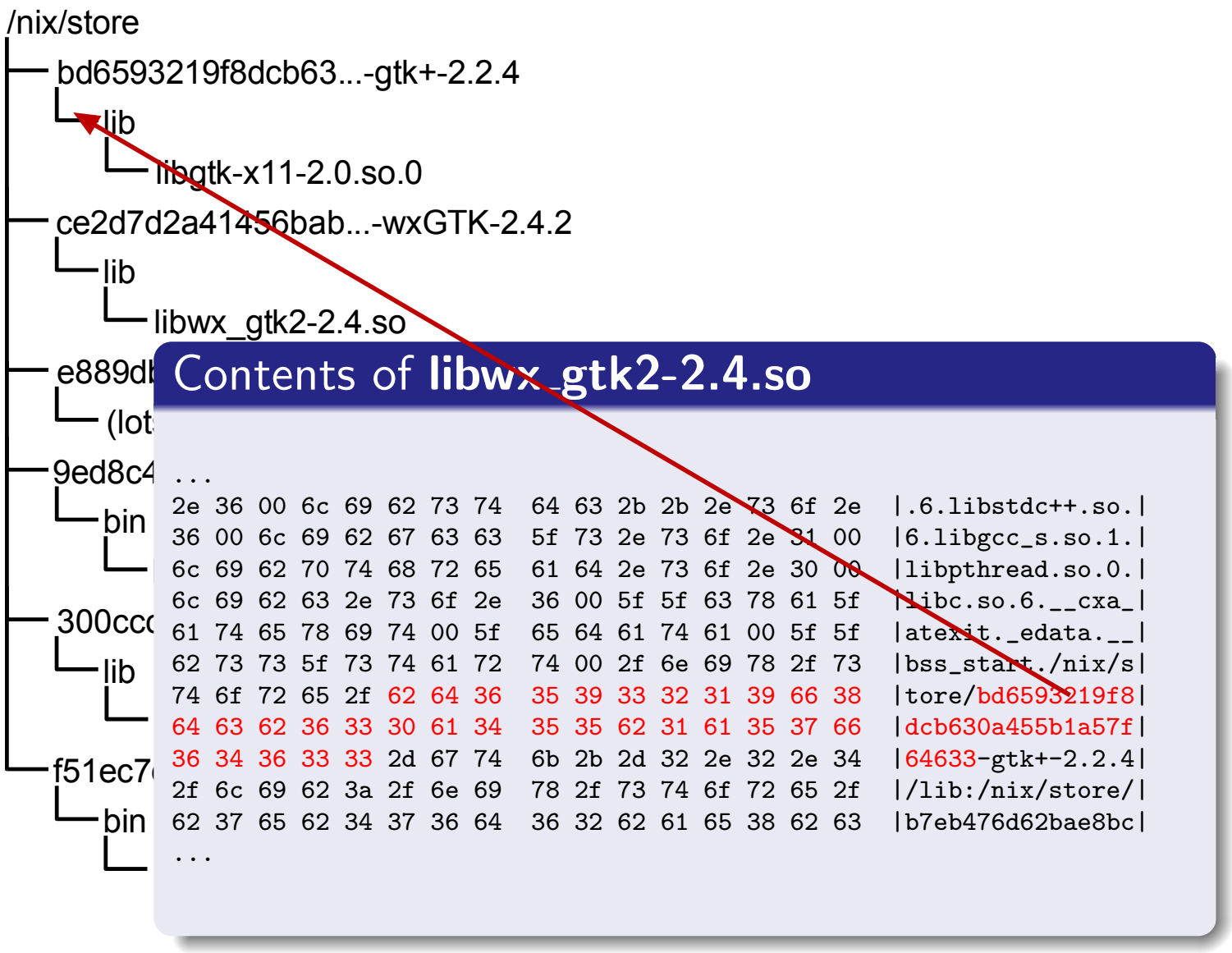
Finding runtime dependencies

```
/nix/store
├── bd6593219f8dcb63...-gtk+-2.2.4
│   └── lib
│       └── libgtk-x11-2.0.so.0
├── ce2d7d2a41456bab...-wxGTK-2.4.2
│   └── lib
│       └── libwx_gtk2-2.4.so
├── e889d
│   └── (lot
├── 9ed8c4
│   └── bin
│       └
├── 300cc
│   └── lib
│       └
└── f51ec7
    └── bin
        └
```

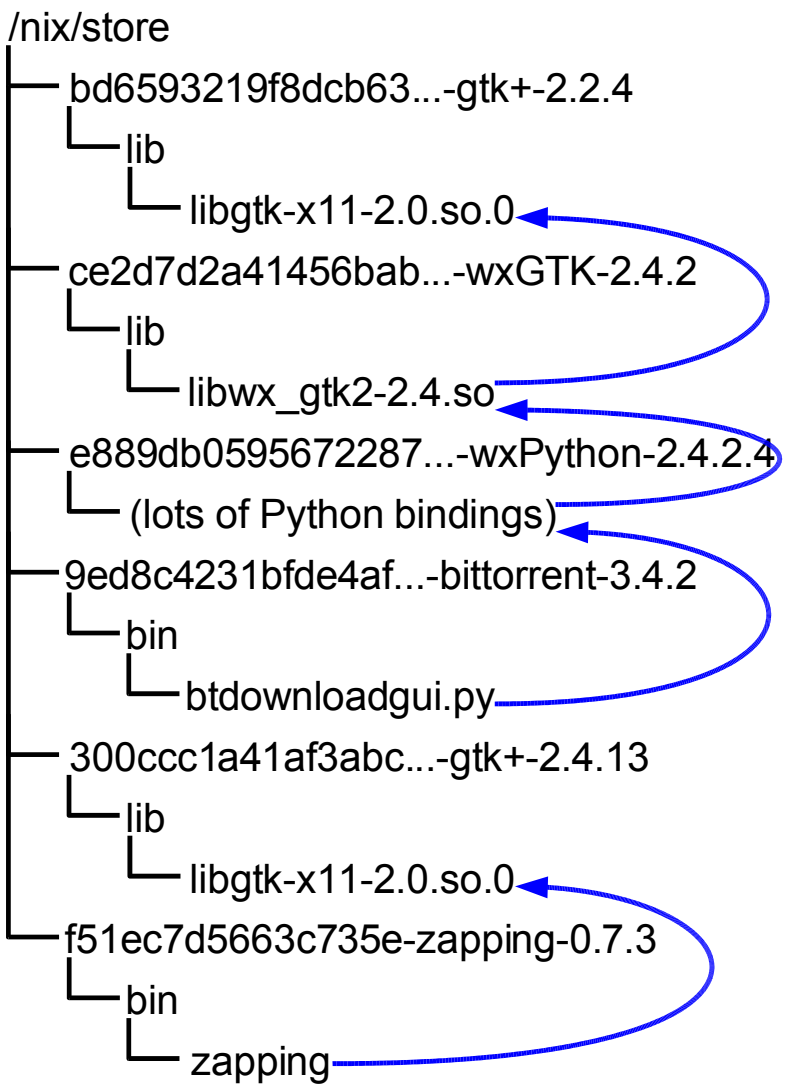
Contents of **libwx_gtk2-2.4.so**

```
...
2e 36 00 6c 69 62 73 74  64 63 2b 2b 2e 73 6f 2e  |.6.libstdc++.so.|
36 00 6c 69 62 67 63 63  5f 73 2e 73 6f 2e 31 00  |6.libgcc_s.so.1.|
6c 69 62 70 74 68 72 65  61 64 2e 73 6f 2e 30 00  |libpthread.so.0.|
6c 69 62 63 2e 73 6f 2e  36 00 5f 5f 63 78 61 5f  |libc.so.6.__cxa_|
61 74 65 78 69 74 00 5f  65 64 61 74 61 00 5f 5f  |atexit._edata.__|
62 73 73 5f 73 74 61 72  74 00 2f 6e 69 78 2f 73  |bss_start./nix/s|
74 6f 72 65 2f 62 64 36  35 39 33 32 31 39 66 38  |tore/bd6593219f8|
64 63 62 36 33 30 61 34  35 35 62 31 61 35 37 66  |dcb630a455b1a57f|
36 34 36 33 33 2d 67 74  6b 2b 2d 32 2e 32 2e 34  |64633-gtk+-2.2.4|
2f 6c 69 62 3a 2f 6e 69  78 2f 73 74 6f 72 65 2f  |/lib:/nix/store/|
62 37 65 62 34 37 36 64  36 32 62 61 65 38 62 63  |b7eb476d62bae8bc|
...
```

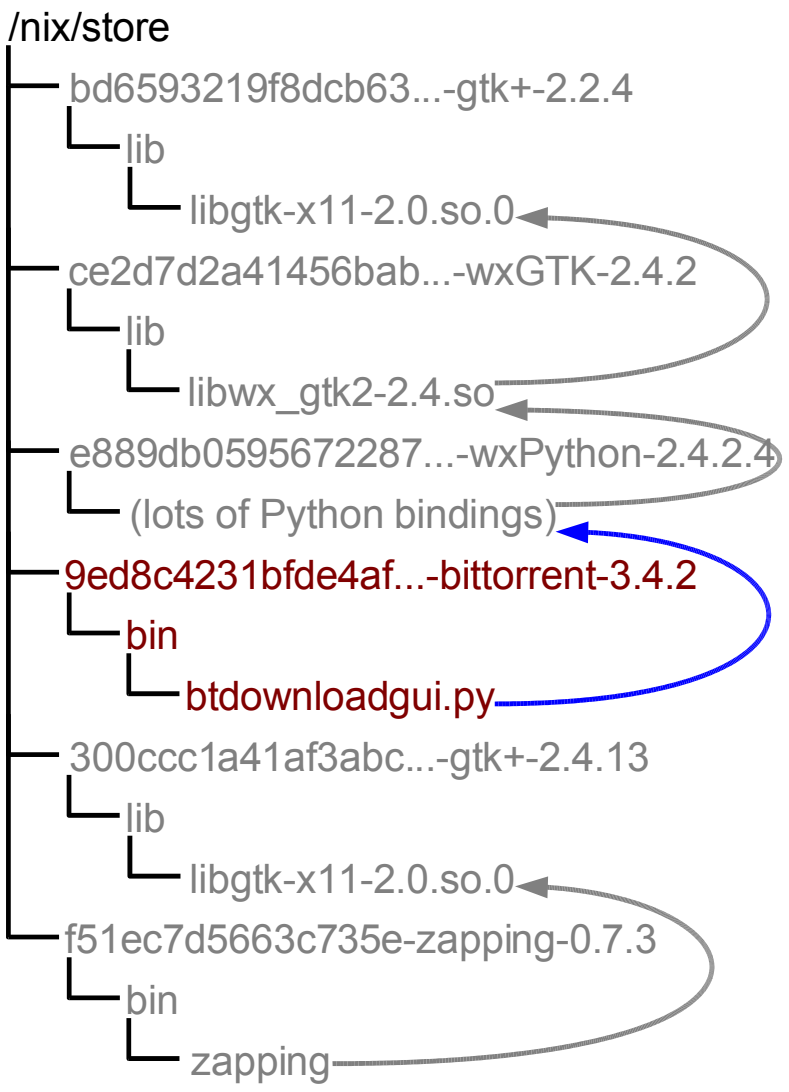
Finding runtime dependencies



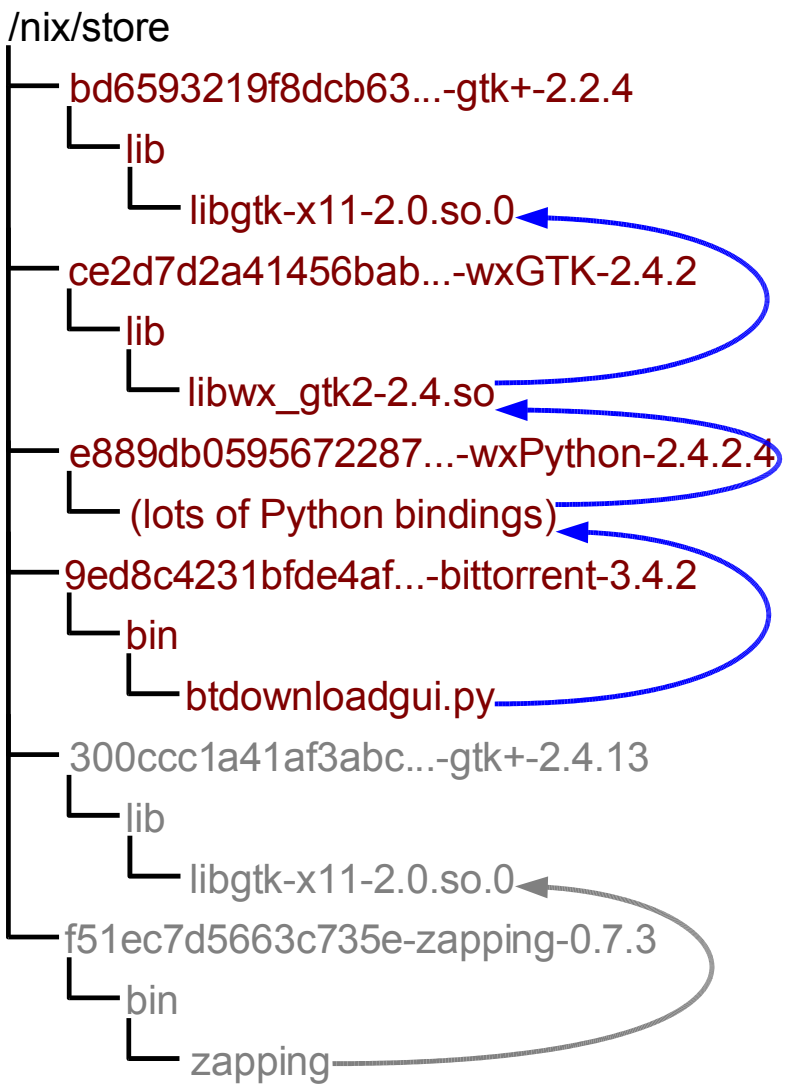
Finding runtime dependencies



Finding runtime dependencies



Finding runtime dependencies



User operations

- To build and install Hello:

```
$ nix-env -if .../all-packages.nix hello
```

- When a new version comes along:

```
$ nix-env -uf .../all-packages.nix hello
```

- If it doesn't work:

```
$ nix-env --rollback
```

- Delete unused components:

```
$ nix-collect-garbage
```

User operations

- To build and install Hello:

```
$ nix-env -if .../all-packages.nix hello
```

- When a new version comes along:

```
$ nix-env -uf .../all-packages.nix hello
```

- If it doesn't work:

```
$ nix-env --rollback
```

- Delete unused components:

```
$ nix-collect-garbage
```

User operations

- To build and install Hello:

```
$ nix-env -if .../all-packages.nix hello
```

- When a new version comes along:

```
$ nix-env -uf .../all-packages.nix hello
```

- If it doesn't work:

```
$ nix-env --rollback
```

- Delete unused components:

```
$ nix-collect-garbage
```

User operations

- To build and install Hello:

```
$ nix-env -if .../all-packages.nix hello
```

- When a new version comes along:

```
$ nix-env -uf .../all-packages.nix hello
```

- If it doesn't work:

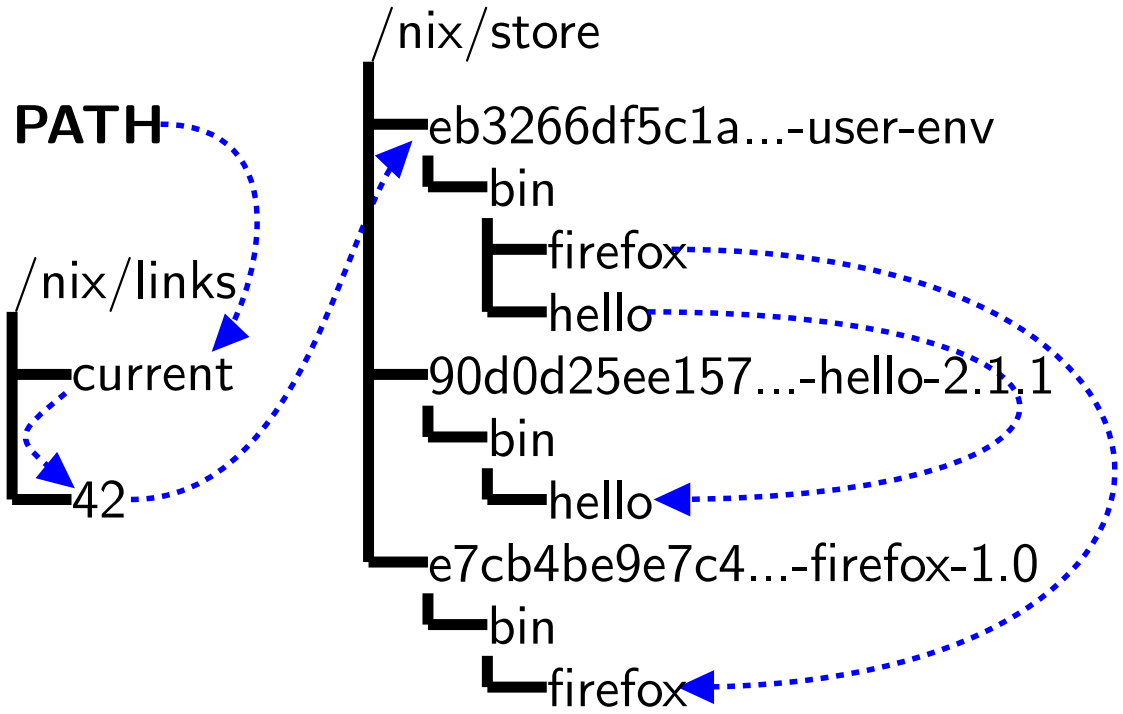
```
$ nix-env --rollback
```

- Delete unused components:

```
$ nix-collect-garbage
```

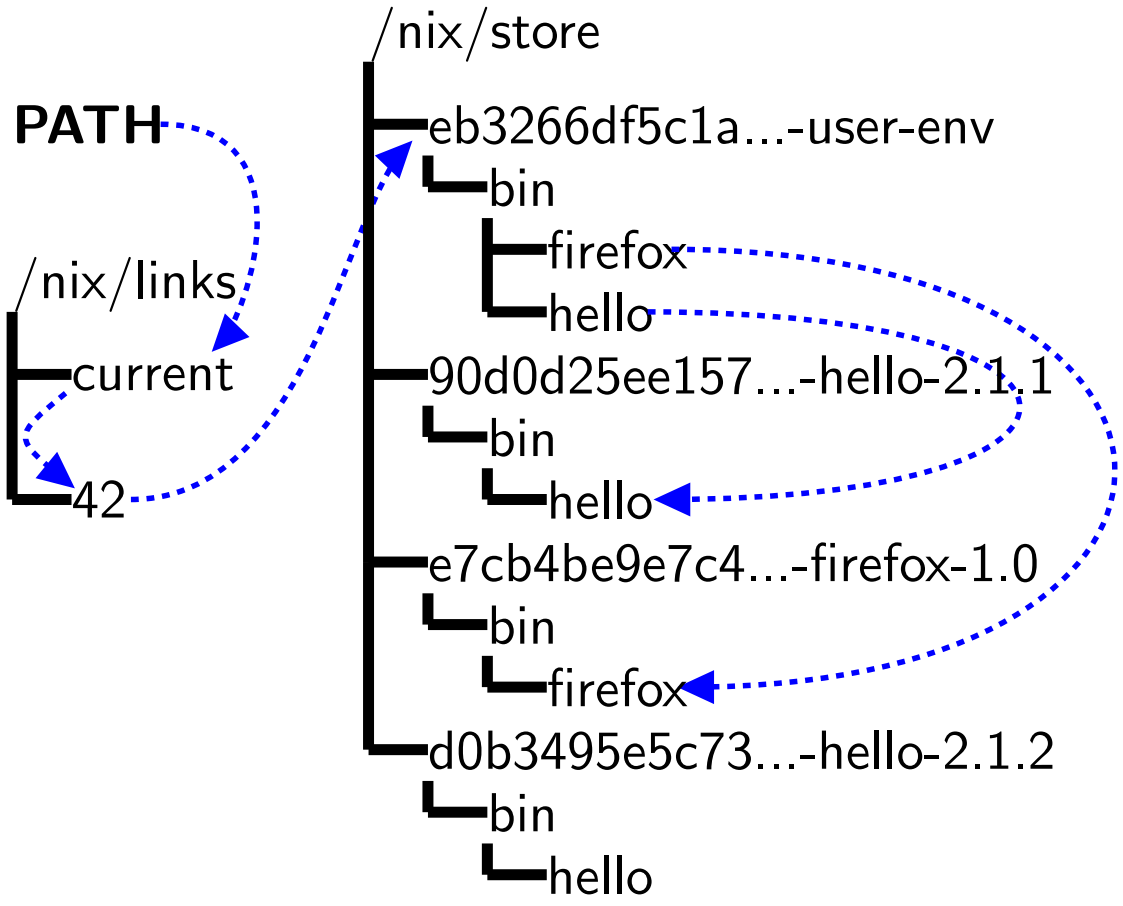
User environments

- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the **garbage collector**.



User environments

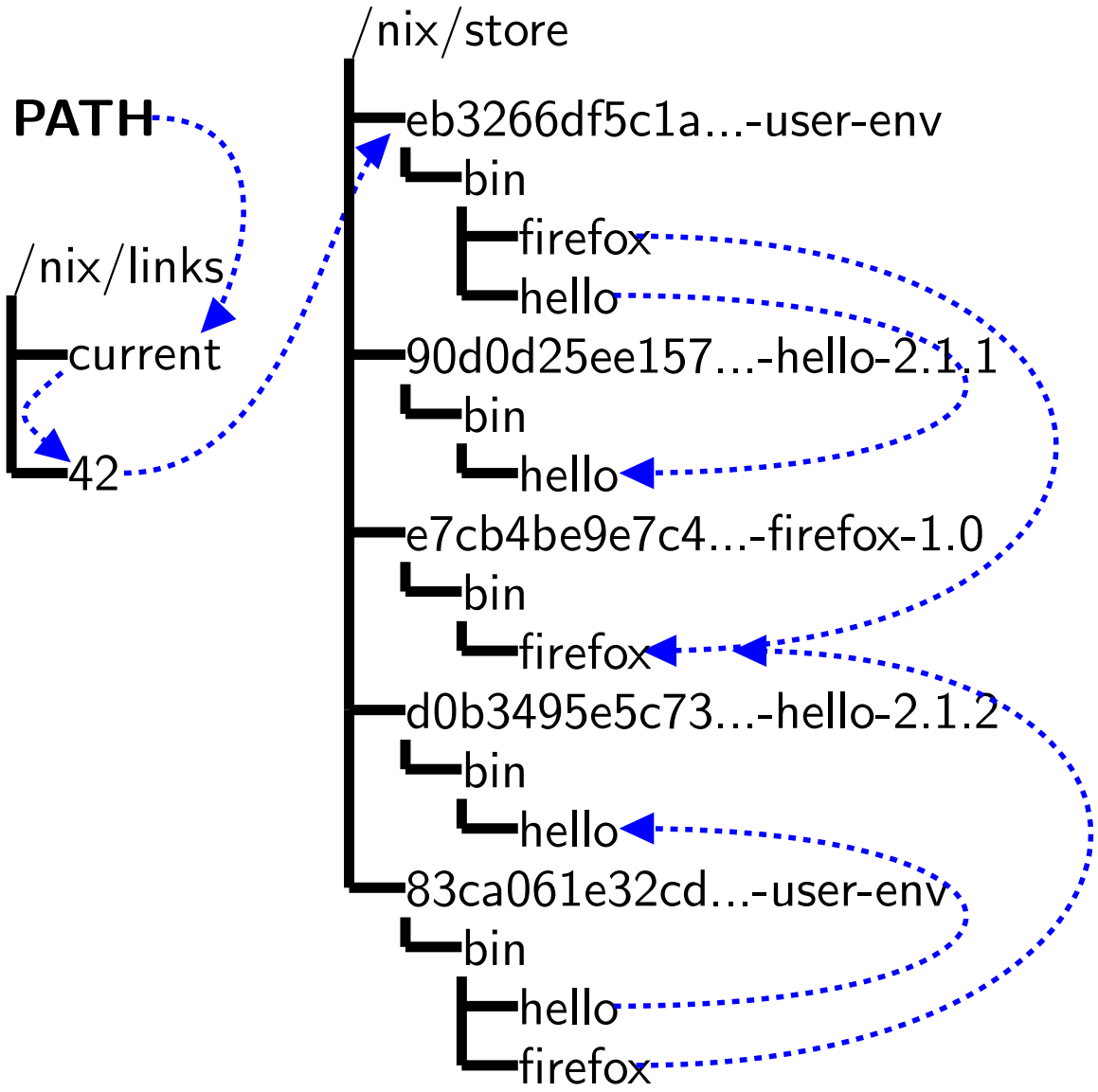
- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the **garbage collector**.



(**nix-env -u hello**)

User environments

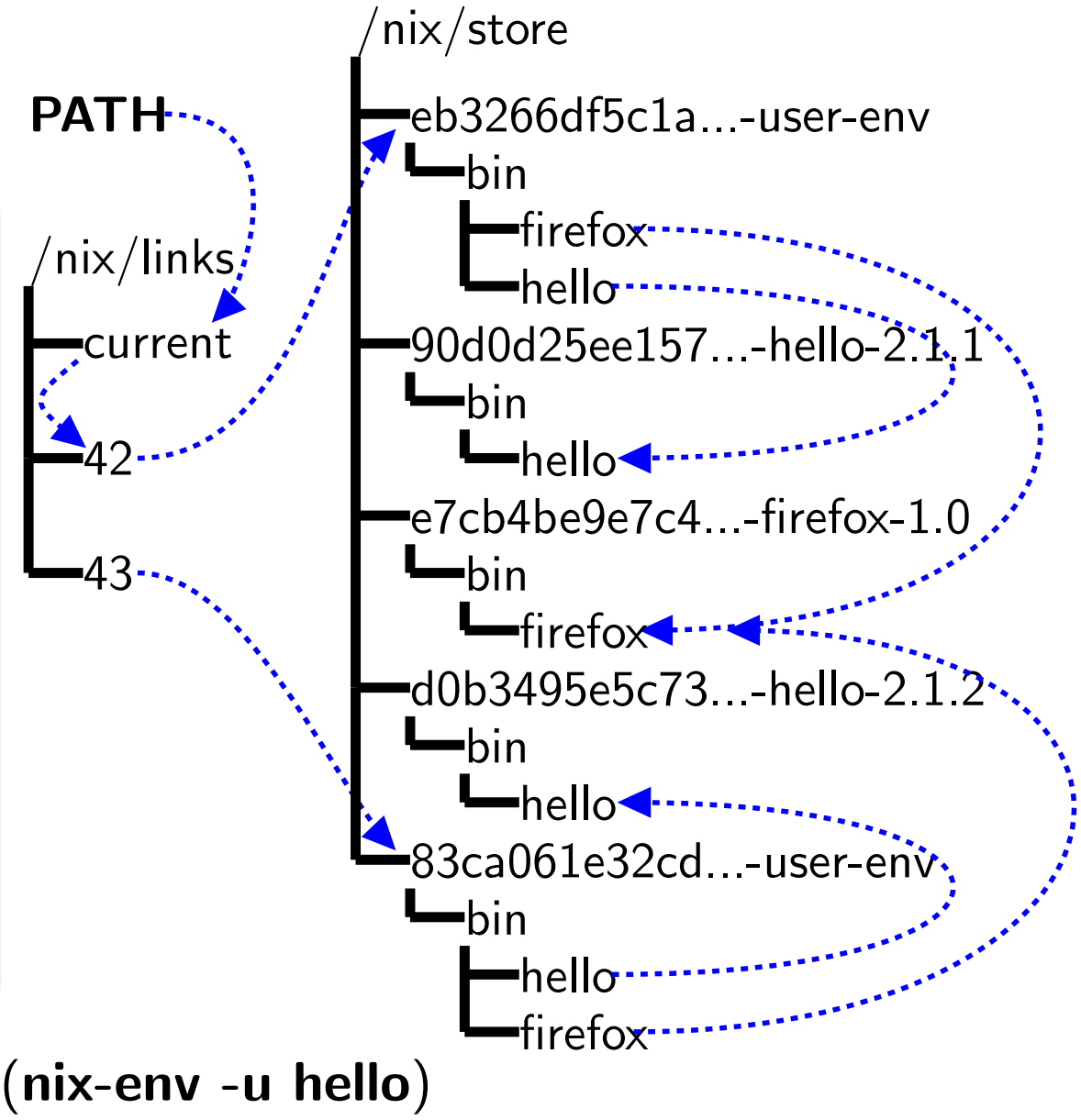
- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the **garbage collector**.



(**nix-env -u hello**)

User environments

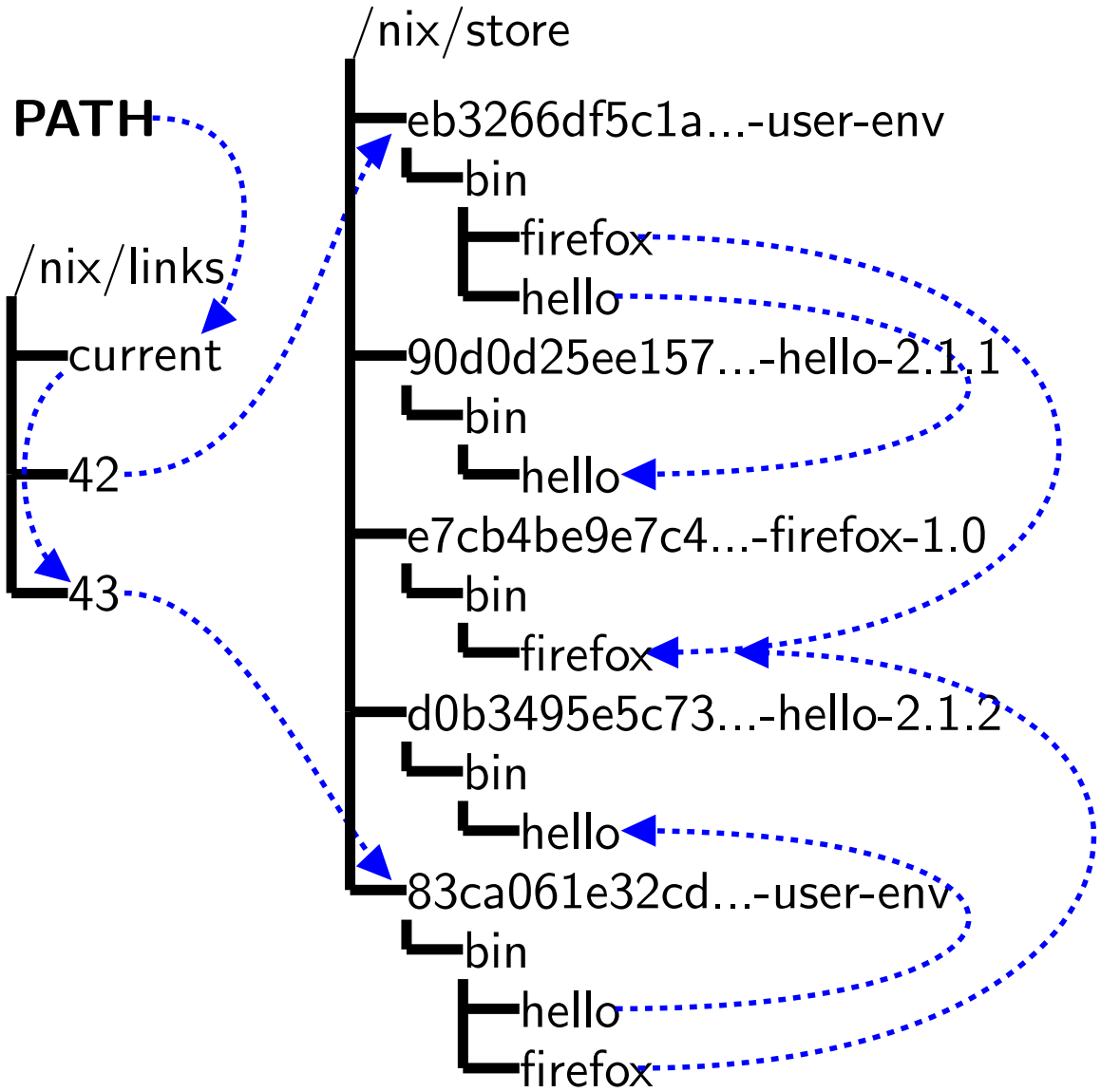
- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the **garbage collector**.



(**nix-env -u hello**)

User environments

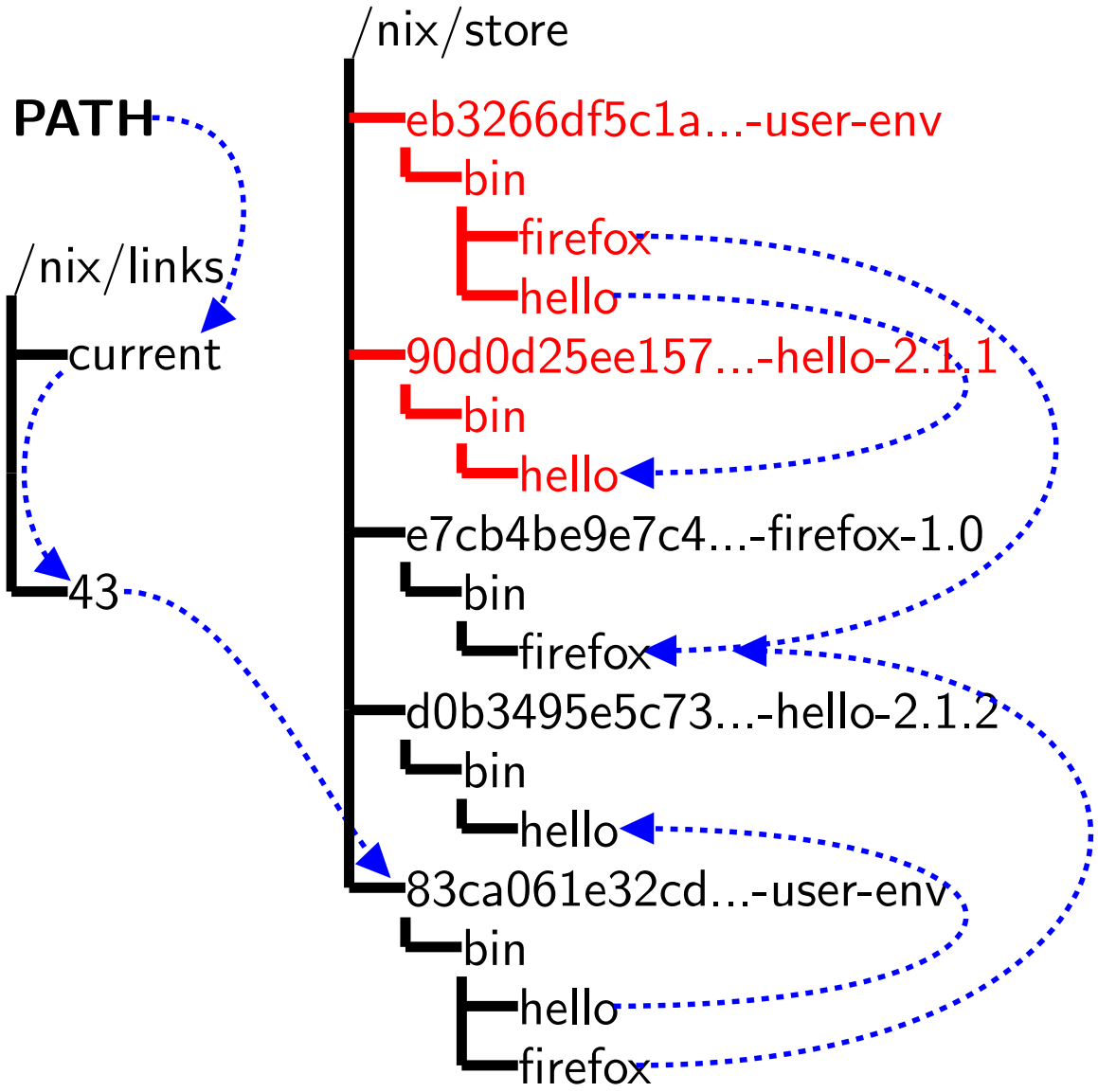
- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the **garbage collector**.



(**nix-env -u hello**)

User environments

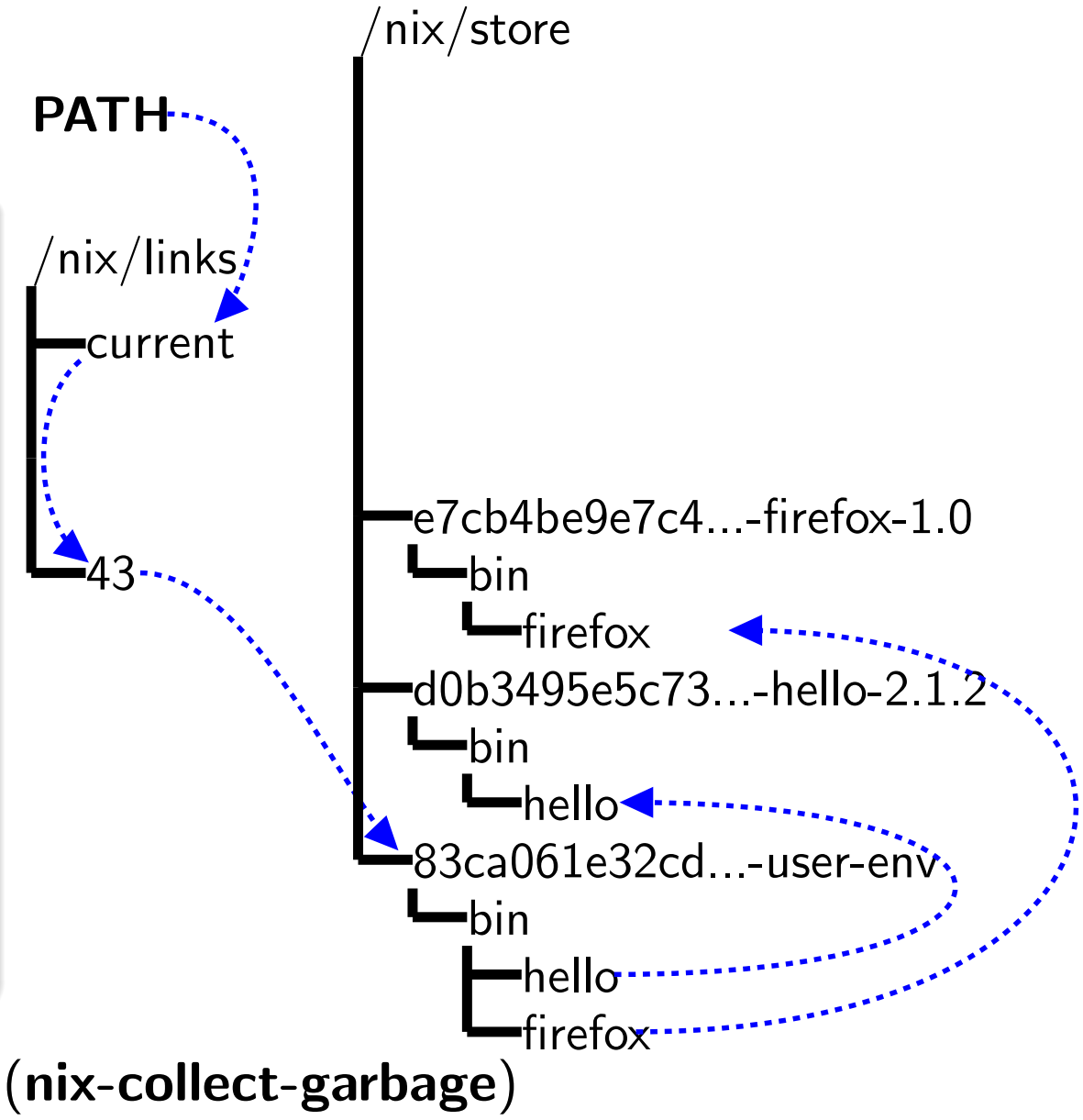
- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the **garbage collector**.



(**nix-env –remove-generations old**)

User environments

- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the **garbage collector**.



Deployment using Nix

- This is conceptually a **source deployment model**.
- We get **binary deployment** by sharing pre-built components.
- On the producer side:

```
$ nix-push $(nix-instantiate .../all-packages.nix) \
  http://server/cache
```

- On the client side:

```
$ nix-pull http://server/cache
$ nix-env -if .../all-packages.nix hello
```

- Installation will now reuse pre-built components, **iff** they are exactly the same.

Deployment using Nix

- This is conceptually a **source deployment model**.
- We get **binary deployment** by sharing pre-built components.
- On the producer side:

```
$ nix-push $(nix-instantiate .../all-packages.nix) \
  http://server/cache
```

- On the client side:

```
$ nix-pull http://server/cache
$ nix-env -if .../all-packages.nix hello
```

- Installation will now reuse pre-built components, **iff** they are exactly the same.

Deployment using Nix

- This is conceptually a **source deployment model**.
- We get **binary deployment** by sharing pre-built components.
- On the producer side:

```
$ nix-push $(nix-instantiate .../all-packages.nix) \
  http://server/cache
```

- On the client side:

```
$ nix-pull http://server/cache
$ nix-env -if .../all-packages.nix hello
```

- Installation will now reuse pre-built components, **iff** they are exactly the same.

An example deployment policy: channels

- Channels allow Nix expressions to be updated automatically.
- Subscribe to a channel:

```
$ nix-channel --add http://.../channels/nixpkgs-stable
```

- Fetch latest channel instance:

```
$ nix-channel --update
```

- Update all installed packages:

```
$ nix-env -u '*'
```

An example deployment policy: channels

- Channels allow Nix expressions to be updated automatically.
- Subscribe to a channel:

```
$ nix-channel --add http://.../channels/nixpkgs-stable
```

- Fetch latest channel instance:

```
$ nix-channel --update
```

- Update all installed packages:

```
$ nix-env -u '*'
```

An example deployment policy: channels

- Channels allow Nix expressions to be updated automatically.
- Subscribe to a channel:

```
$ nix-channel --add http://.../channels/nixpkgs-stable
```

- Fetch latest channel instance:

```
$ nix-channel --update
```

- Update all installed packages:

```
$ nix-env -u '*'
```

Nix Packages collection

Nixpkgs

- Contains Nix expressions for 716 existing Unix packages.
  - Development tools: GCC, Perl, Mono, ...
  - Libraries: Glibc, GTK, Qt, X11, ...
  - Applications: Firefox, Xine, Quake 3, ...
  - Servers: **httpd**, PostgreSQL, ...
- On Linux/x86, fully bootstrapped (no external dependencies).

Service deployment

Services: sets of running programs that provide some useful facility on a system or network.

Example: Subversion service

Subversion Repoman - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

https://svn.cs.uu.nl:12443/repoman

Go

Universiteit Utrecht

Subversion Server

Administrative tasks

- You can create a new repository.
- You can add a new user (only within the `cs.uu.nl` domain).
- You can edit your user information.

Online information

- Subversion homepage.
- Subversion: The Definitive Guide.

Repositories

The following repositories are hosted on this server:

Name	Owner	Description
3bate	kuipers	Repository van 3bate
adaptivewavelets	dijkema	Adaptive Wavelet Project
aefs	eelco	Portable cryptographic file system
afp-excercises-jenr	neus	afp
afp-exercises	bdumitri	Advanced Functional Programming exercises (Bogdan, Ze)
afp-exercises-ra	rjlwanro	AFP 2005 Exercises, Rjlwanro, Amiddelk
afp-project	bdumitriu	Advanced Functional Programming project (Ze, Bogdan, Huanwen, Jinfeng)

Done

svn.cs.uu.nl:12443

Example: Issue tracking service

[#NIXOS-16] NixOS should not wipe my hard drive - Stratego/XT JIRA - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

https://bugs.cs.uu.nl/browse/NIXOS-16

JIRA Stratego/XT JIRA

History | Log In

HOME BROWSE PROJECT FIND ISSUES CREATE NEW ISSUE QUICK SEARCH:

Issue Details [XML]

Key: NIXOS-16
Type: Bug
Status: Open
Priority: Blocker
Assignee: Armijn Hemel
Reporter: Eelco Dolstra
Votes: 0
Watchers: 0

Operations

- Clone this issue
- Comment on this issue

If you were logged in you would be able to see more operations.

NixOS

NixOS should not wipe my hard drive

Created: 2005-08-23 14:26 Updated: 2005-08-23 14:26

Return to search

Issue **14** of **25** issue(s)

<< Previous | NIXOS-16 | Next >>

Component/s: None
Affects Version/s: None
Fix Version/s: None

Original Estimate:	Unknown	Remaining Estimate:	Unknown	Time Spent:	Unknown

Description

The installer should

a) ask for confirmation before installing
b) not wipe the existing Nix store but reuse it

All Comments Work Log Change History Sort Order:

There are no comments yet on this issue.

This site is running on Atlassian JIRA with a free **Open Source Project / Non-profit License** (license details).
JIRA is an issue and bug tracking application. Evaluate JIRA for your organisation.

Done bugs.cs.uu.nl

Service deployment is hard

Service deployment involves a number of steps:

- Deploy software components (e.g., Apache, PostgreSQL, Subversion)
- Edit configuration files (e.g., **httpd.conf**, **viewcvs.conf**)
- Initialise state (e.g., logging directories, database tables)
- Start/stop processes
- ... and all of this possibly on multiple machines / platforms

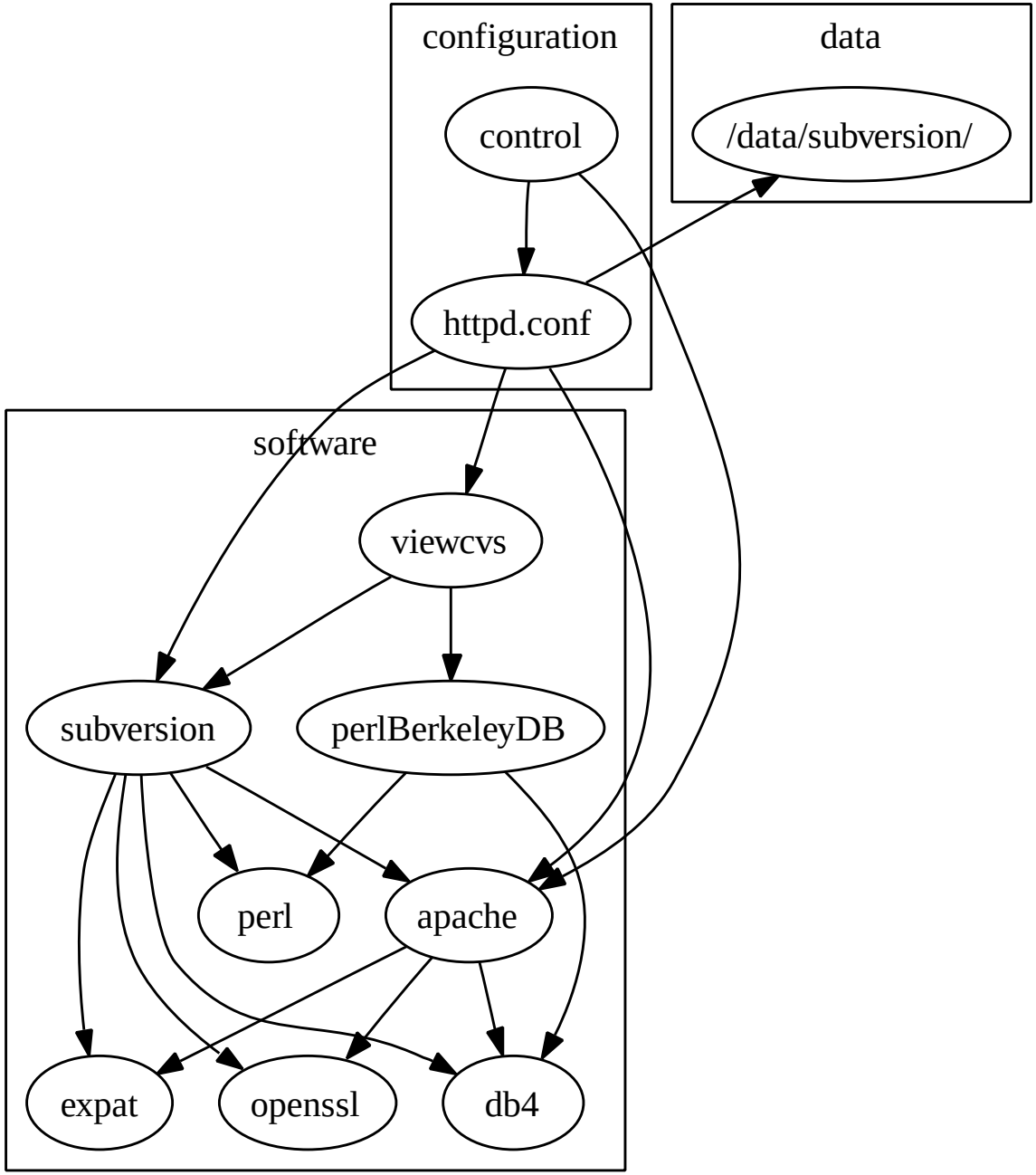
Problems

- Poor reproducibility (bad CM)
- Hard to support parallel configurations
- Cross-cutting configuration choices

Problem 1: Poor reproducibility

- Goal: it should be possible to realise a service by running a single command.
  - E.g., to move it to another machine
  - So no manual installing of missing software components, tweaking of configuration files, creating missing directories, etc.
- Why is reproducibility hard?
  - Admins often manually edit configuration files and initialise state
  - Service configuration doesn't express software component dependencies

Example


configuration
control
httpd.conf
data
/data/subversion/
software
viewcvs
subversion
perlBerkeleyDB
perl
apache
expat
openssl
db4

Gap between package management and service configuration

- Software components are typically deployed through package managers such as RPM
- Service configuration is typically kept under version management
- However, there is no good way to express the dependencies of the service on the software components

Problem 2: Parallel configurations

- It should be easy to create different instances of a service
  - Test vs. production servers (running on different ports, using different databases, etc.)
  - Instantiations for different users
  - Evolution through time (rollbacks)
- This is hard to support because there are typically lots of configuration files and control scripts that refer to lots of paths for components, state, static data files, etc.
  - **/etc/apache/httpd.conf**, **/etc/init.d/apache**, **/etc/apache/viewcvs.conf**, ...

Example

/etc/apache/httpd.conf for Subversion service (fragment)

```
ServerRoot "/var/httpd"
ServerName svn.cs.uu.nl:8080
LoadModule dav_svn_module /usr/lib/modules/mod_dav_svn.so
<Location /repos>
    AuthType Basic
    AuthDBMUserFile /data/subversion/db/svn-users
    ...
    SVNParentPath /data/subversion/repos
</Location>
ScriptAlias /viewcvs /usr/viewcvs/www/cgi/viewcvs.cgi
```

Example

/etc/init.d/httpd for Subversion service (fragment)

```
/usr/sbin/apachectl -k start -f /etc/apache/httpd.conf
```

Use cases

- Try out with a different set of repositories.
- Try out a different Apache.
- Try out a different Subversion module.

Problem 3: Cross-cutting configuration choices

- Many configuration choices are *cross-cutting*, i.e., impact many different (parts of) configuration files, scripts, etc.
- Examples:
  - Port numbers
  - Host names
  - Paths (major source of problems!)
- So a change to the configuration choices must be realised in many different places
- Lots of work
- Danger of inconsistency

Example: port number

In /etc/init.d/httpd.conf

```
ServerName www.example.org:12443
Listen 12443
<VirtualHost _default_:12443>
```

In repoman.pl

```
my $url = "https://www.example.org:12443/"
print "... <a href='$url/repos/$repoName'> ...";
```

Solution

Treat all the *static parts* of configurations as Nix components:

- Software
- Configuration files
- Control scripts
- Static data files (e.g., static web pages)

But not mutable state, e.g. databases

Advantages

- Support multiple configurations side-by-side
- Nix's functional language: can easily support multiple configurations
- Atomic upgrades / rollbacks
- SCM support:
  - Full knowledge of dependencies
  - Reproducibility (easy to move to another machine)

Example

Nix expression for **svn.cs.uu.nl**

```
{productionServer ? true}:
let {
  webServer = import ../../apache-httpd {
    hostName = "svn.cs.uu.nl";
    httpPort = if productionServer then 80 else 12080;
    httpsPort = if productionServer then 443 else 12443;
    adminAddr = "eelco@cs.uu.nl";
    subServices = [ subversionService ];
    ...
  };

  subversionService = import ../../subversion {
    reposDir = rootDir + "/repos"; ...
  }
```

Use

Building the test instance

$ upgrade-server.sh svn ./default.nix test

- Produces (say) **/nix/store/98wl...-apache-httpd-service**.
- Generated configuration files (e.g., **/nix/store/98wl...-apache-httpd-service/conf/httpd.conf** tell server to run on port 12080/12443.
- Old test server is stopped, new server is started.

Building the production instance

$ upgrade-server.sh svn ./default.nix production

- Produces (say) **/nix/store/6sc6...-apache-httpd-service**.
- Generated configuration files (e.g., **/nix/store/6sc6...-apache-httpd-service/conf/httpd.conf** tell server to run on port 80/443.
- Old production server is stopped, new server is started.

Use

Building the test instance

$ upgrade-server.sh svn ./default.nix test

- Produces (say) **/nix/store/98wl...-apache-httpd-service**.
- Generated configuration files (e.g., **/nix/store/98wl...-apache-httpd-service/conf/httpd.conf** tell server to run on port 12080/12443.
- Old test server is stopped, new server is started.

Building the production instance

$ upgrade-server.sh svn ./default.nix production

- Produces (say) **/nix/store/6sc6...-apache-httpd-service**.
- Generated configuration files (e.g., **/nix/store/6sc6...-apache-httpd-service/conf/httpd.conf** tell server to run on port 80/443.
- Old production server is stopped, new server is started.

NixOS

- Nix usually used on existing Unix systems (e.g., SuSE Linux, Fedora Core, Mac OS X, FreeBSD)
- Taking it all the way: NixOS (Armijn Hemel)
- All packages installed through Nix
- Also hope to improve system configuration
  - Stuff in **/etc** become Nix service components, so we get rollbacks etc.

Conclusions

- Contributions:
  - Safe, automatic coexistance of versions/variants.
  - Reliable dependencies.
  - Multiple concurrent configurations.
  - Atomic upgrades/rollbacks.
  - Safe garbage collection.
  - Binary deployment is automatic.
  - Can accomodate many deployment policies.
  - Useful for service deployment.
  - Integrated continuous integration / release management.
- Available at `http://www.cs.uu.nl/groups/ST/Trace/Nix`.

Further reading

ICSE'04 E. Dolstra, E. Visser, and M. de Jonge, *Imposing a Memory Management Discipline on Software Deployment*

LISA'04 E. Dolstra, M. de Jonge, and E. Visser, *Nix: A Safe and Policy-Free System for Software Deployment*

CBSE'05 E. Dolstra, *Efficient Upgrading in a Purely Functional Component Deployment Model*

ASE'05 E. Dolstra, *Secure Sharing Between Untrusted Users in a Transparent Source/Binary Deployment Model*

PhD thesis E. Dolstra, *The Purely Functional Software Deployment Model*