Software deployment with Nix

IPA Lentedagen 2005

Eelco Dolstra
eelco@cs.uu.nl

Institute of Information & Computing Sciences
Utrecht University, The Netherlands

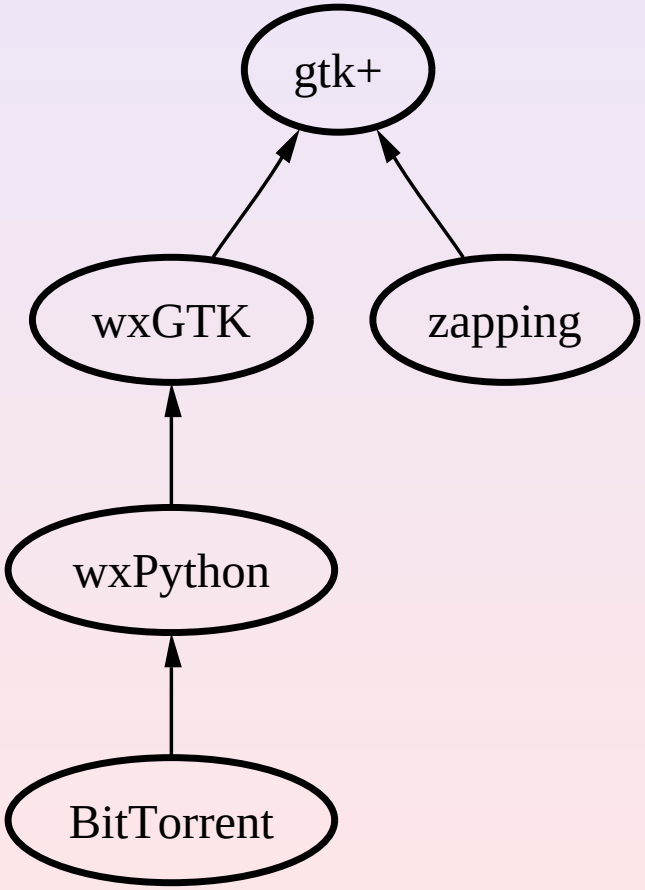
March 31, 2005

Software deployment

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

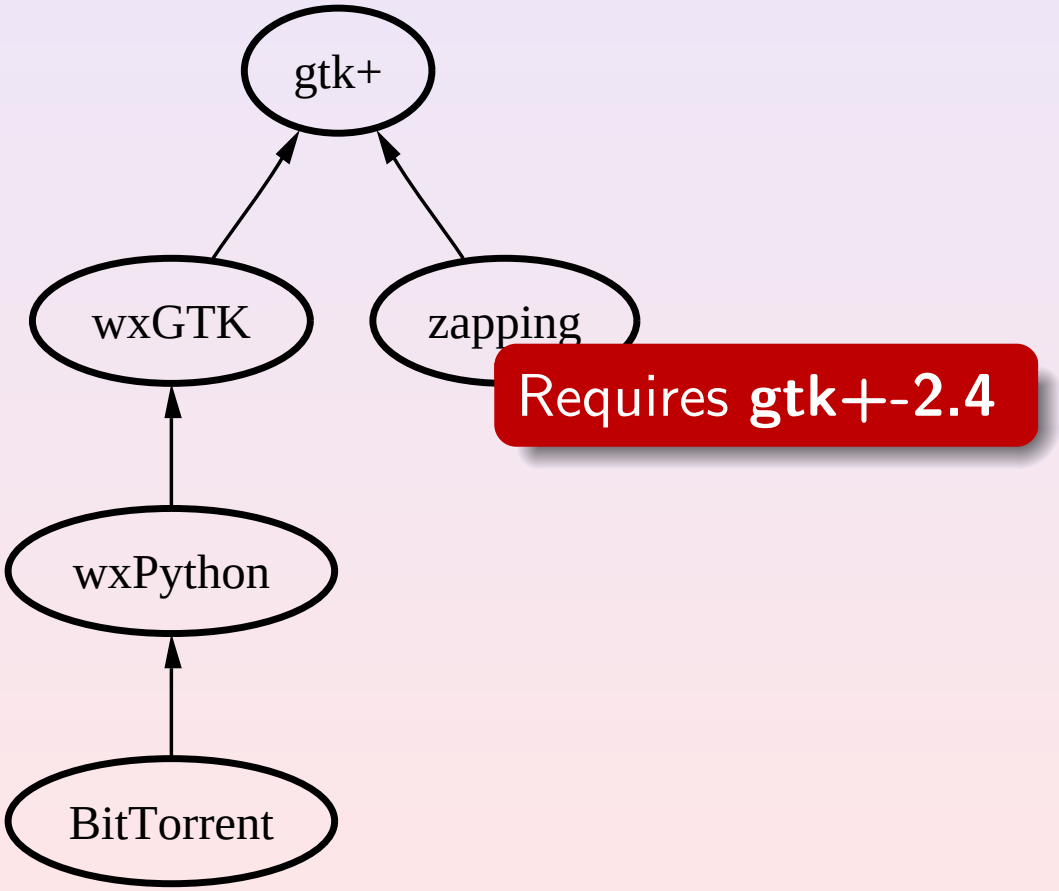
So why is this hard?

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



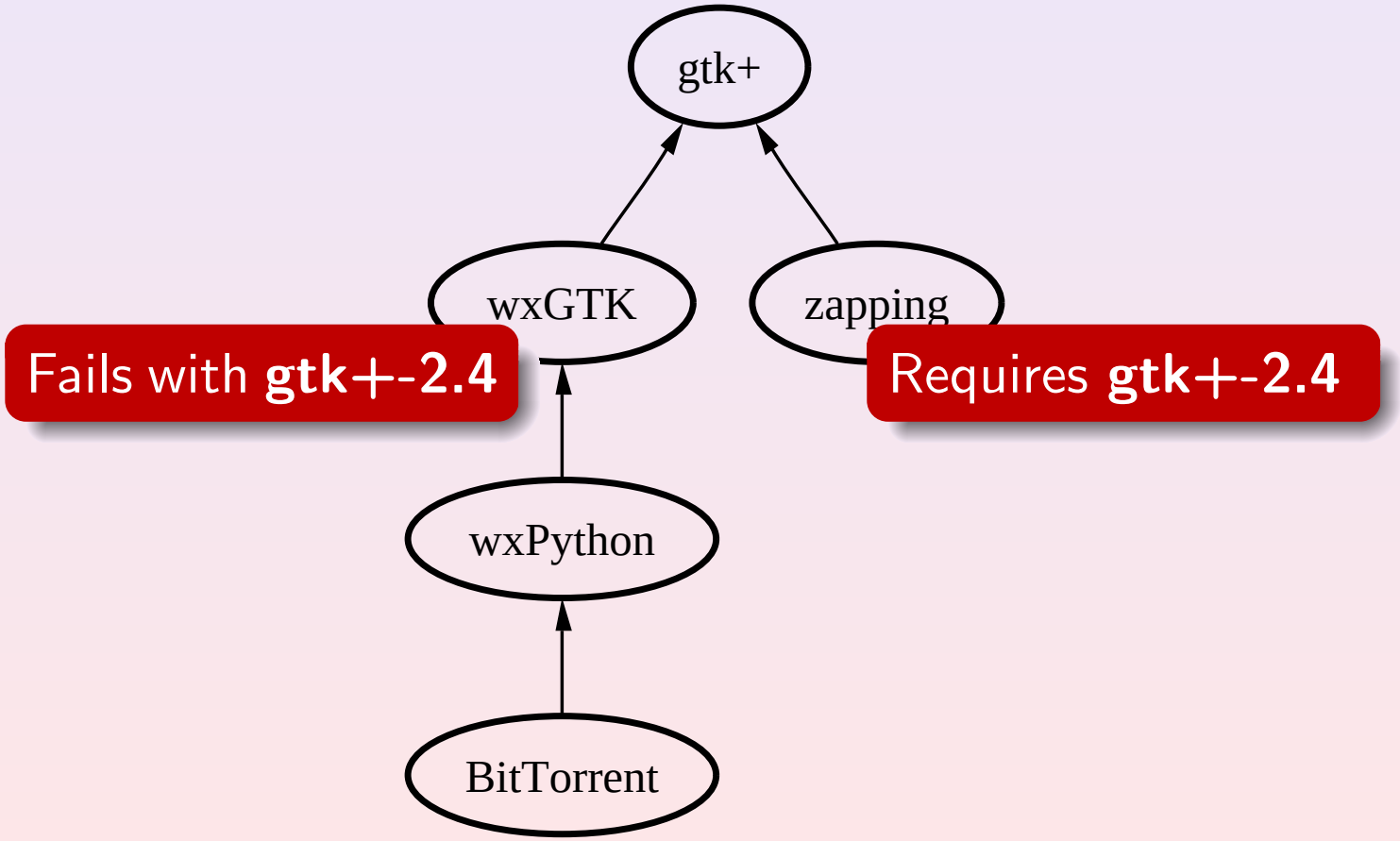
So why is this hard?

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



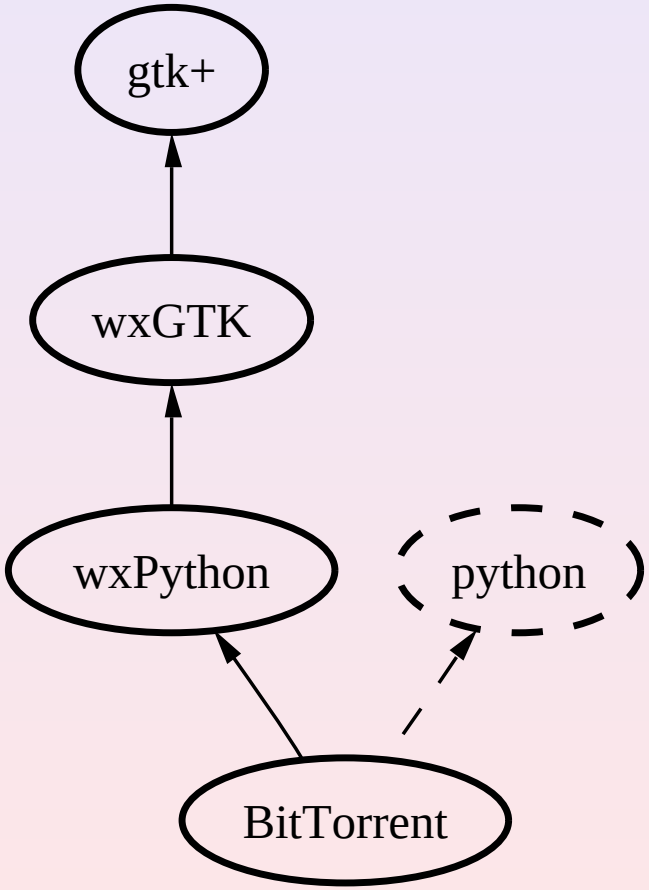
So why is this hard?

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



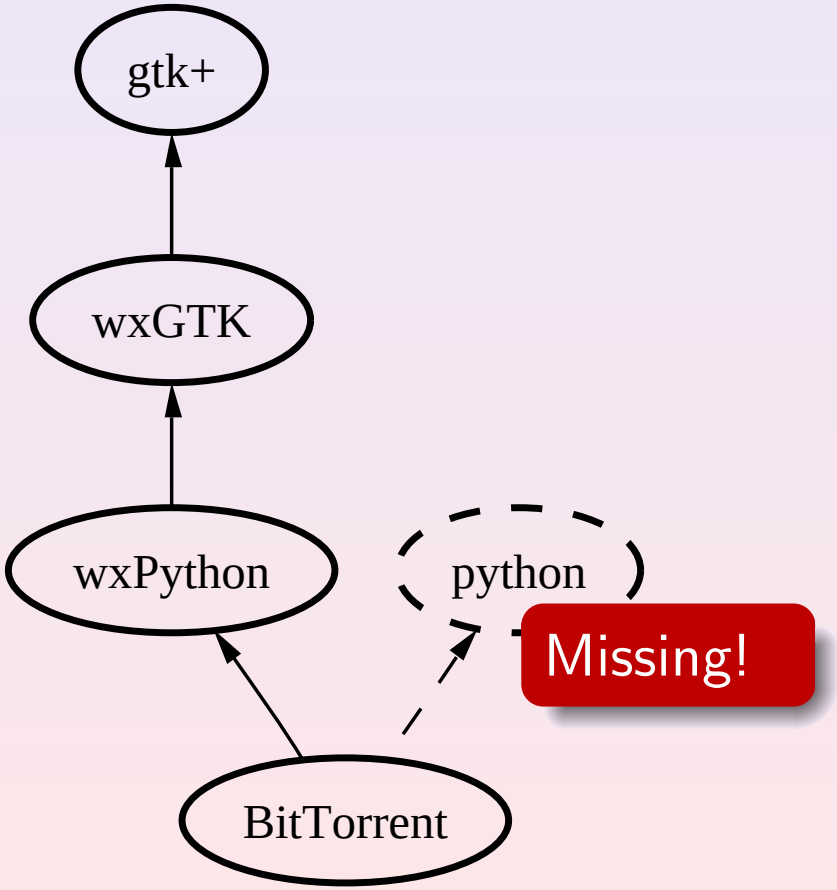
So why is this hard?

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

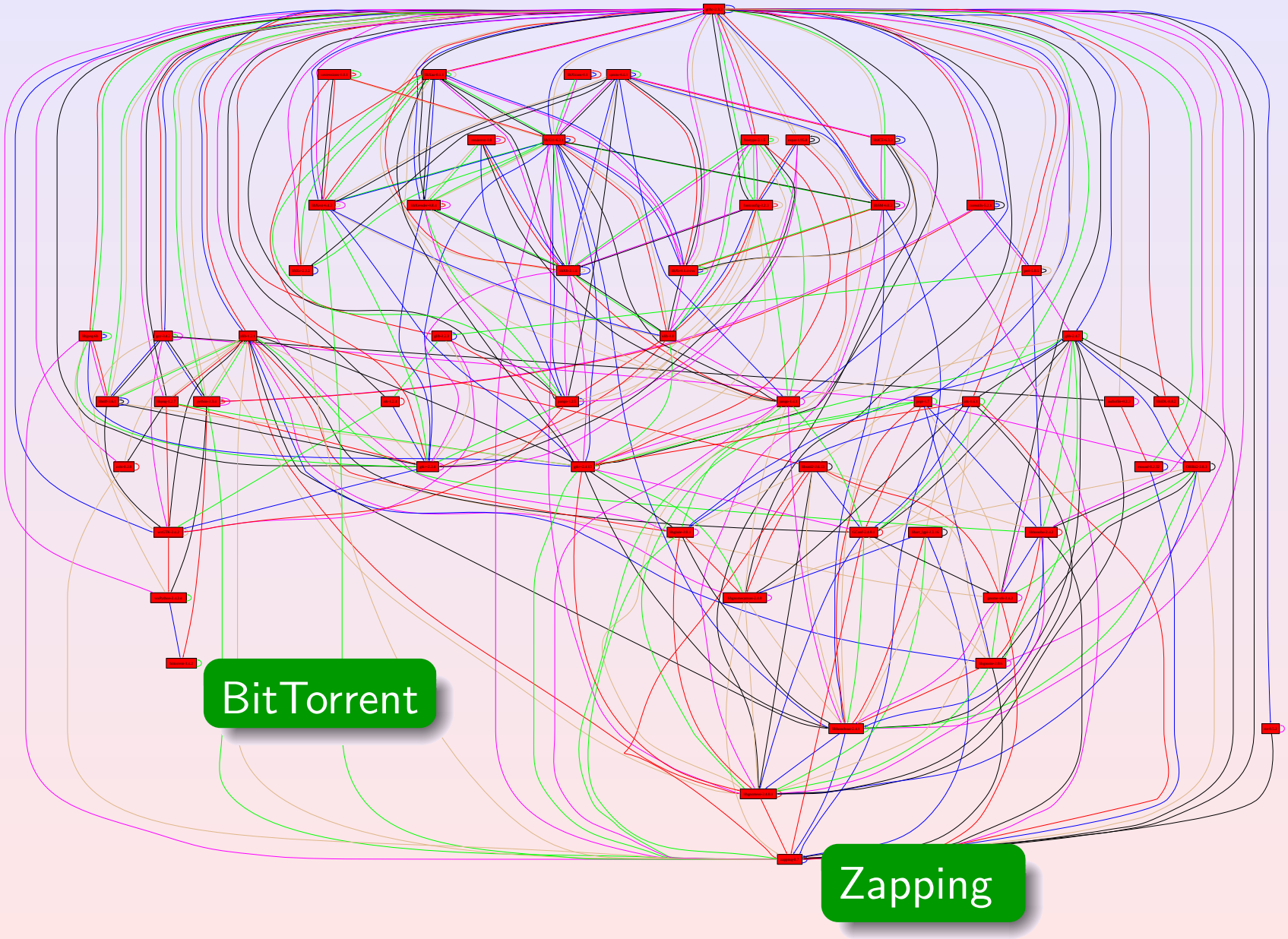


So why is this hard?

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



So why is this hard?



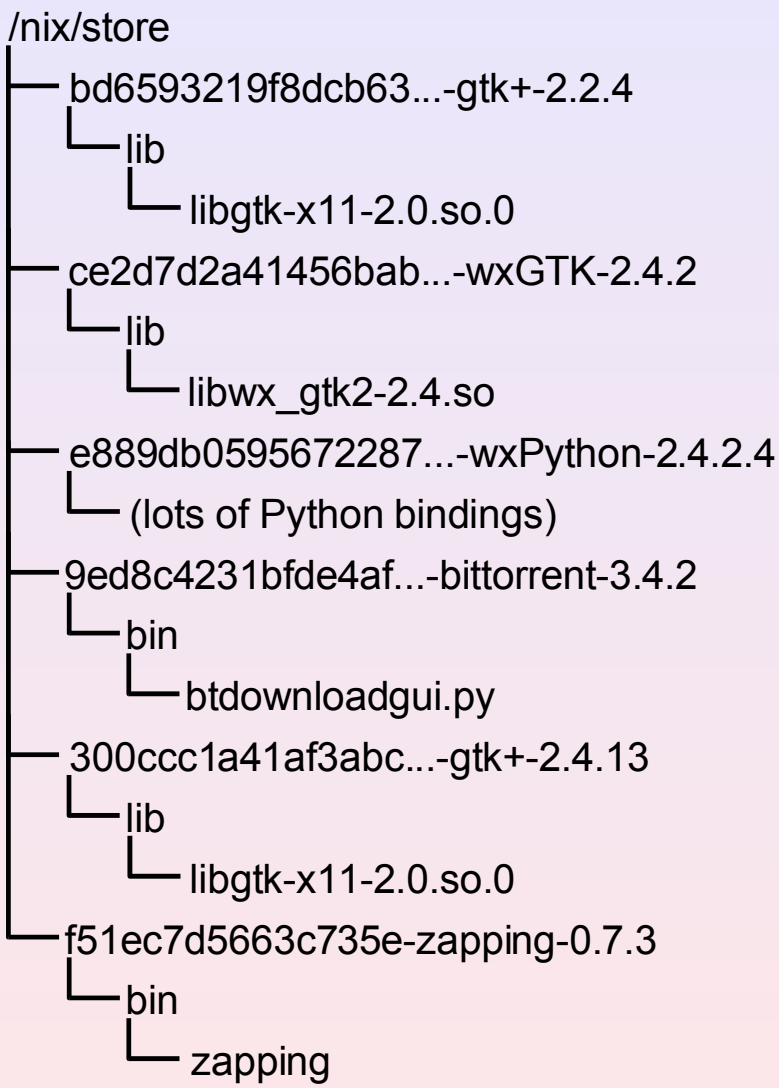
The Nix Deployment System

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

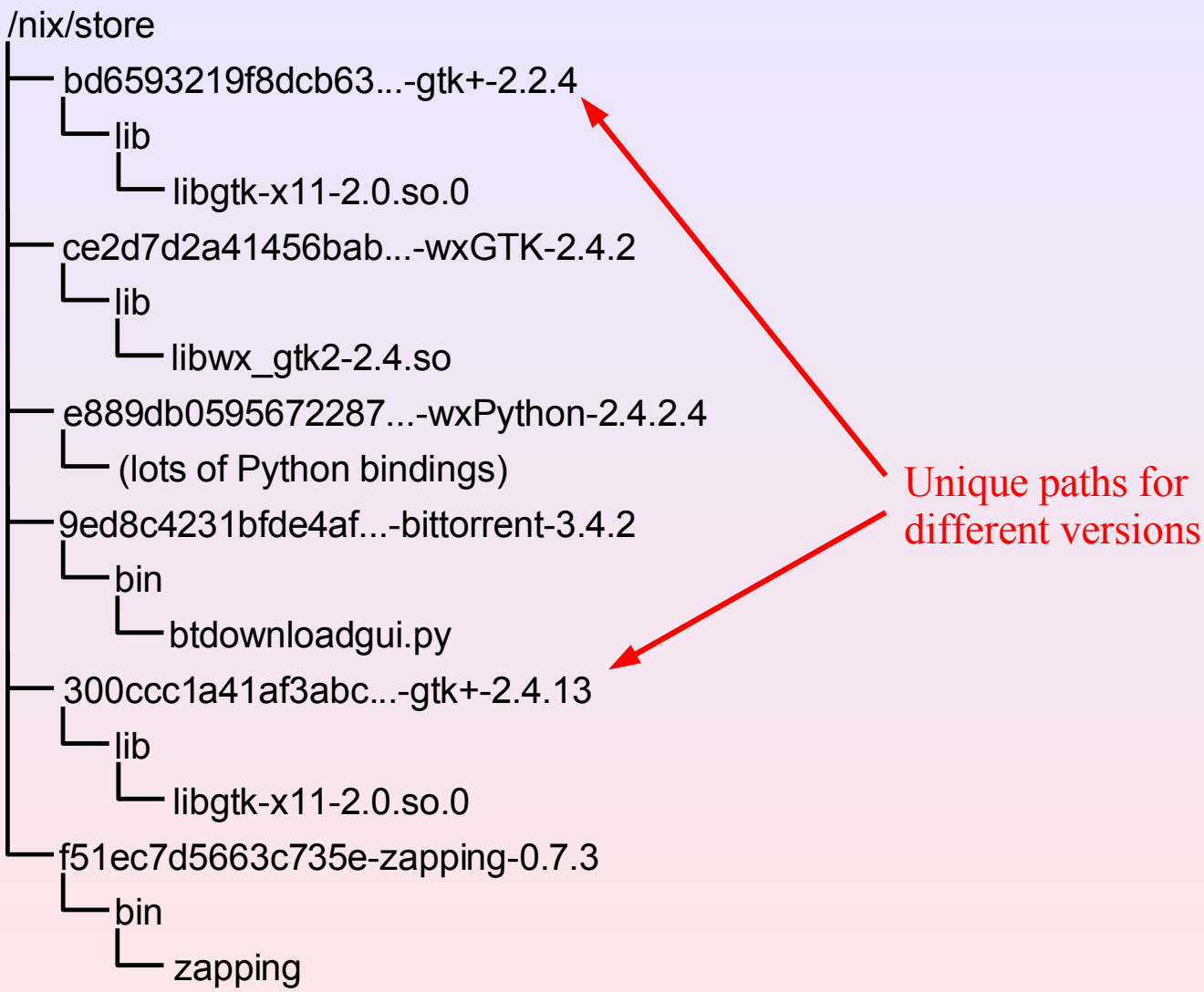
```
/nix/store/605332199533e73b...-gtk+-2.2.4
```

  which is an SHA-256 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

```
{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;
}
```

hello/default.nix

```
{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

```
{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

```
. $stdenv/setup

PATH=$perl/bin:$PATH

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

Nix expressions

hello/builder.sh

```
. $stdenv/setup

PATH=$perl/bin:$PATH

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

Environment initially empty; prevents undeclared dependencies

Nix expressions

system/all-packages-generic.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-generic.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;
};
```

Variability

```
{ localServer, stdenv, fetchurl
, openssl ? null, db4 ? null, ... }:

assert localServer -> db4 != null;
assert sslSupport
  -> openssl != null &&
  && (httpServer -> httpd.openssl == openssl);

stdenv.mkDerivation {
  name = "subversion-1.1.3";
  builder = ./builder.sh;
  src = fetchurl {url=...};
  ...
}
```

Deployment as memory management

How to discover retained dependencies?

memory	$\Leftrightarrow$	disk
objects (values)	$\Leftrightarrow$	components
addresses	$\Leftrightarrow$	path names
pointer dereference	$\Leftrightarrow$	I/O
pointer arithmetic	$\Leftrightarrow$	string operations
dangling pointer	$\Leftrightarrow$	reference to absent component

Deployment as memory management

How to discover retained dependencies?

memory	$\Leftrightarrow$	disk
objects (values)	$\Leftrightarrow$	components
addresses	$\Leftrightarrow$	path names
pointer dereference	$\Leftrightarrow$	I/O
pointer arithmetic	$\Leftrightarrow$	string operations
dangling pointer	$\Leftrightarrow$	reference to absent component

Deployment as memory management

How to discover retained dependencies?

memory	$\Leftrightarrow$	disk
objects (values)	$\Leftrightarrow$	components
addresses	$\Leftrightarrow$	path names
pointer dereference	$\Leftrightarrow$	I/O
pointer arithmetic	$\Leftrightarrow$	string operations
dangling pointer	$\Leftrightarrow$	reference to absent component

Deployment as memory management

How to discover retained dependencies?

memory	$\Leftrightarrow$	disk
objects (values)	$\Leftrightarrow$	components
addresses	$\Leftrightarrow$	path names
pointer dereference	$\Leftrightarrow$	I/O
pointer arithmetic	$\Leftrightarrow$	string operations
dangling pointer	$\Leftrightarrow$	reference to absent component

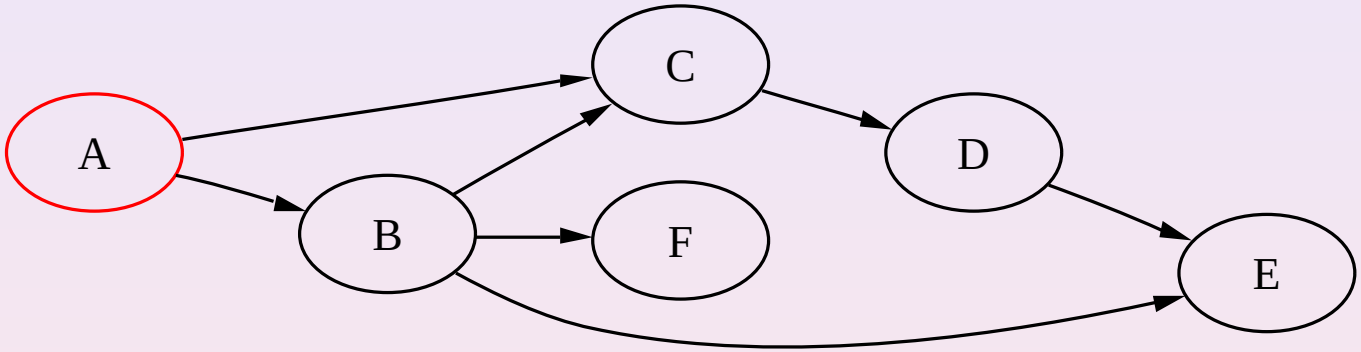
Deployment as memory management

How to discover retained dependencies?

memory	$\Leftrightarrow$	disk
objects (values)	$\Leftrightarrow$	components
addresses	$\Leftrightarrow$	path names
pointer dereference	$\Leftrightarrow$	I/O
pointer arithmetic	$\Leftrightarrow$	string operations
dangling pointer	$\Leftrightarrow$	reference to absent component

Deployment requires closures

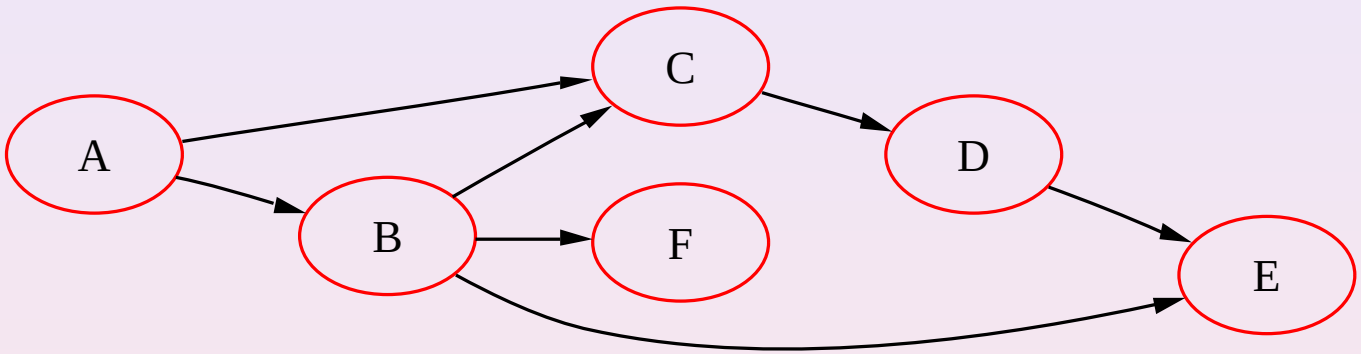
- Correct deployment of component c requires distributing the smallest set of components C containing c closed under the "has-a-pointer-to" relation.



- So we have to discover the *pointer graph*.
- This is exactly what garbage collectors for programming languages have to do.

Deployment requires closures

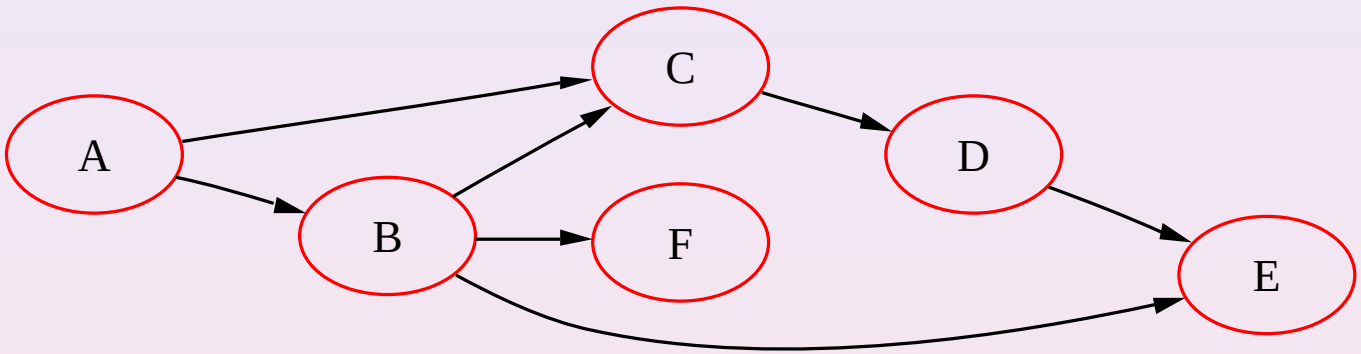
- Correct deployment of component c requires distributing the smallest set of components C containing c closed under the "has-a-pointer-to" relation.



- So we have to discover the *pointer graph*.
- This is exactly what garbage collectors for programming languages have to do.

Deployment requires closures

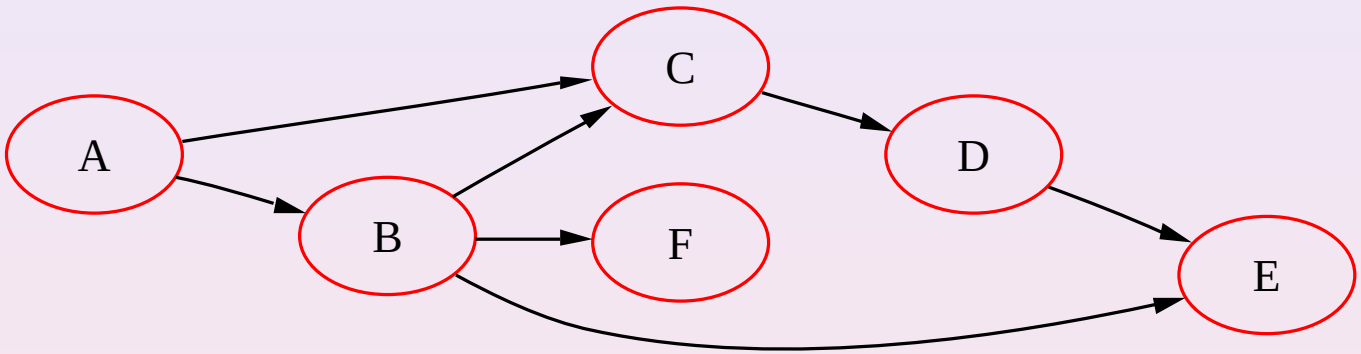
- Correct deployment of component c requires distributing the smallest set of components C containing c closed under the "has-a-pointer-to" relation.



- So we have to discover the *pointer graph*.
- This is exactly what garbage collectors for programming languages have to do.

Deployment requires closures

- Correct deployment of component c requires distributing the smallest set of components C containing c closed under the "has-a-pointer-to" relation.

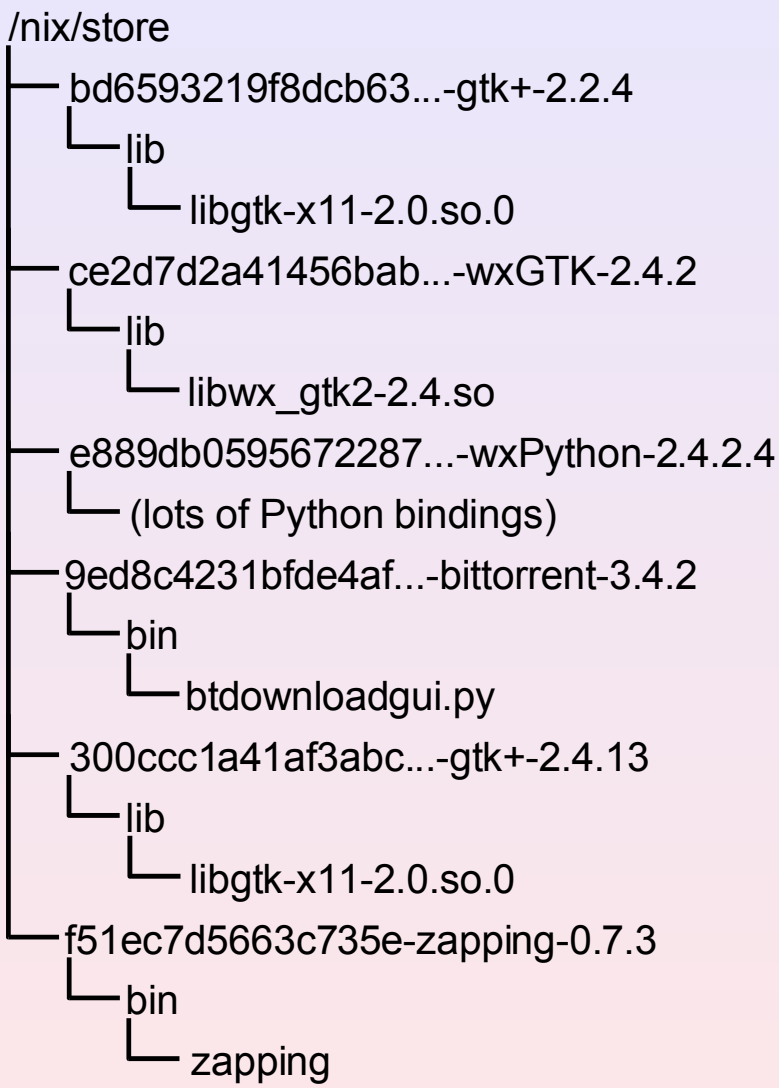


- So we have to discover the *pointer graph*.
- This is exactly what garbage collectors for programming languages have to do.

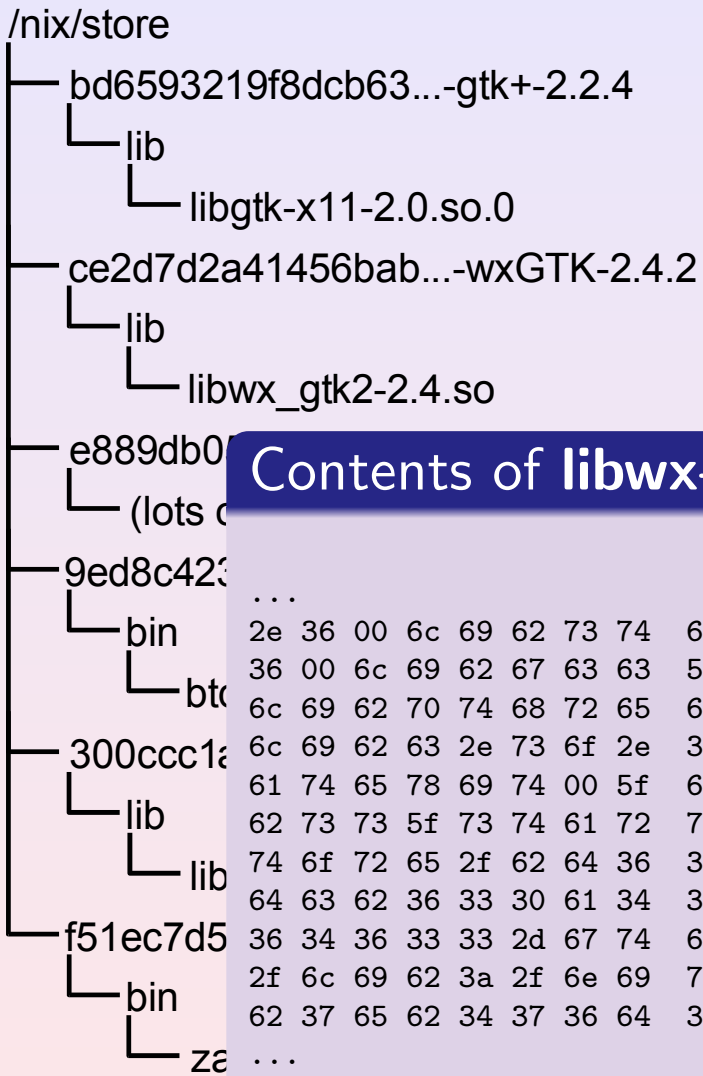
Pointer discipline in PLs

- GC requires a *pointer discipline*:
  - Ideally, entire memory layout is known, and no arbitrary pointer formation (e.g., integer $\Leftrightarrow$ pointer casts).
  - But even C/C++ has rules: pointer arithmetic is not allowed to move a pointer out of the object it points to.
  - This is why *conservative GC* works: assume that everything that looks like a pointer *is* a pointer.
- But software components do not have any pointer discipline.
  - Any string can be a pointer.
  - Pointer arithmetic and dereferencing directories can produce pointers to any object in the file system.

Finding runtime dependencies



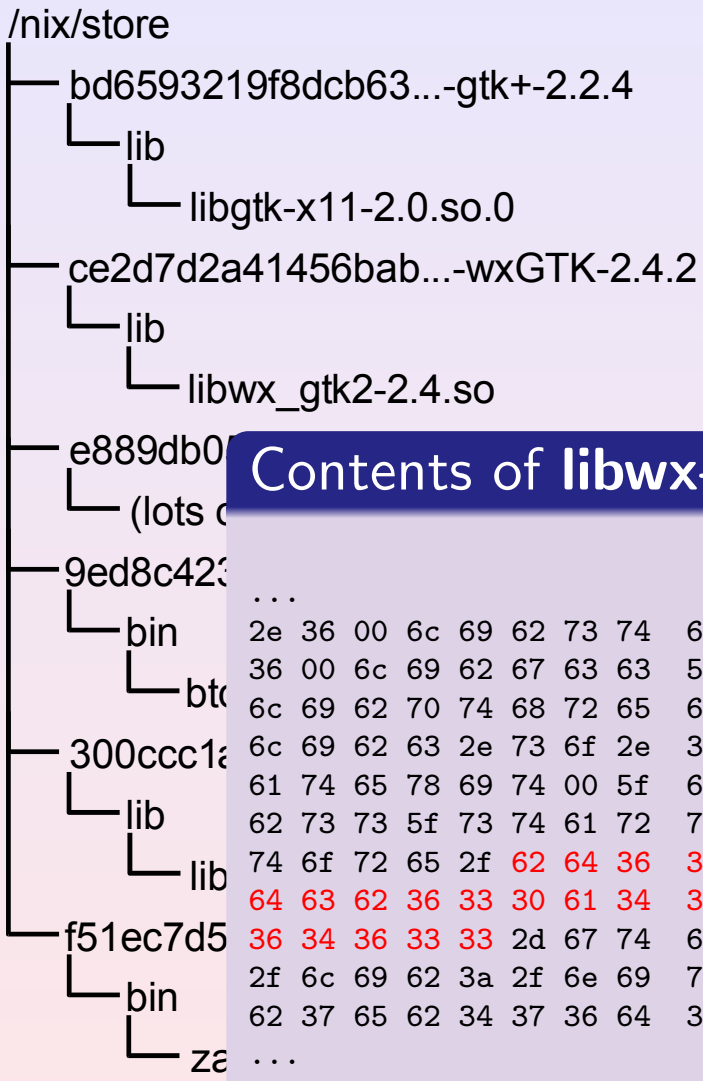
Finding runtime dependencies



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|
...
```

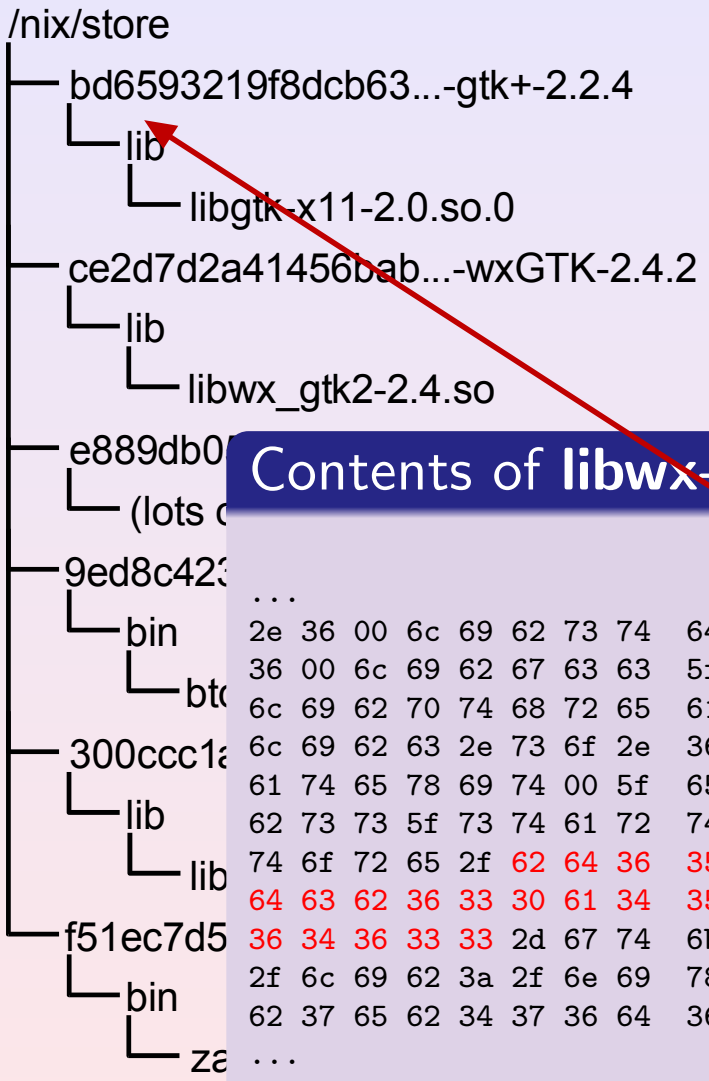
Finding runtime dependencies



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|
...
```

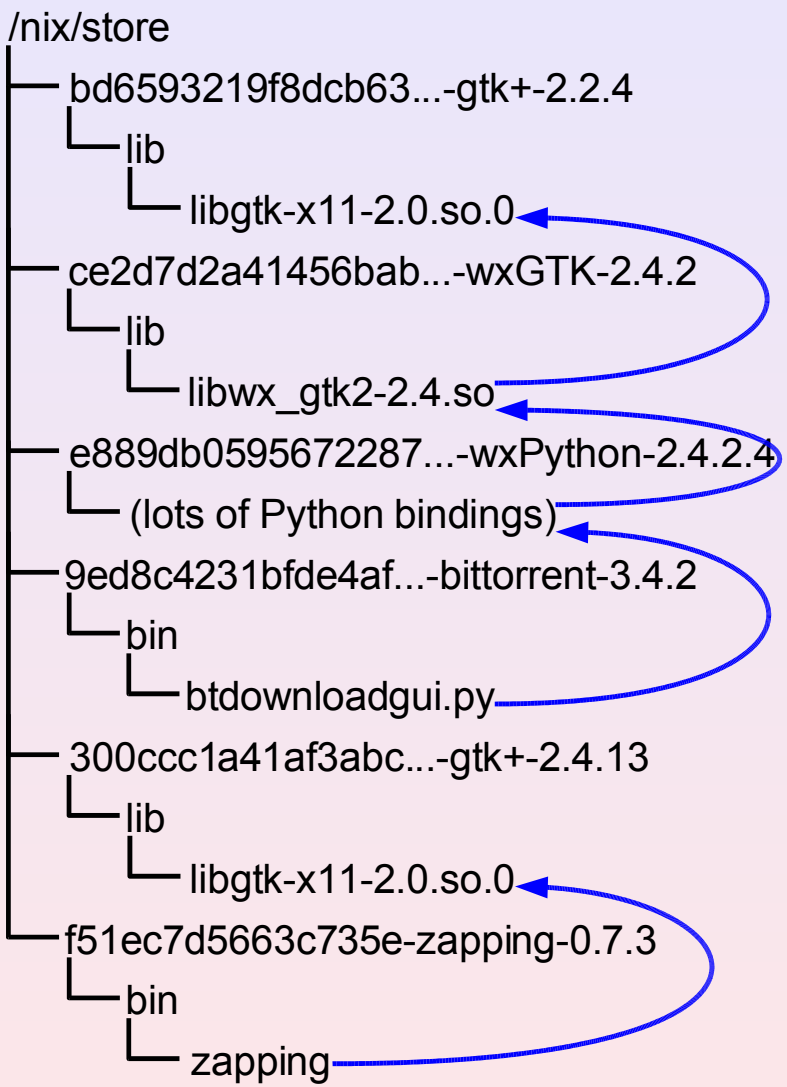
Finding runtime dependencies



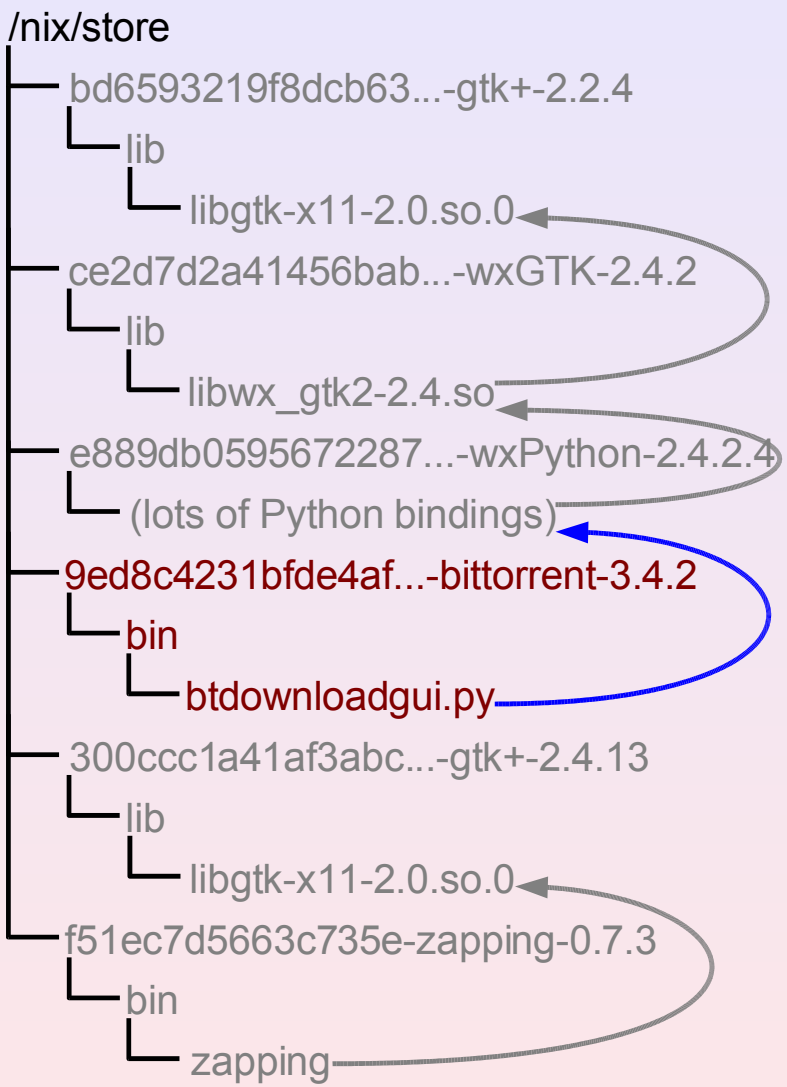
Contents of **libwx-gtk2-2.4.so**

```
...
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|
...
```

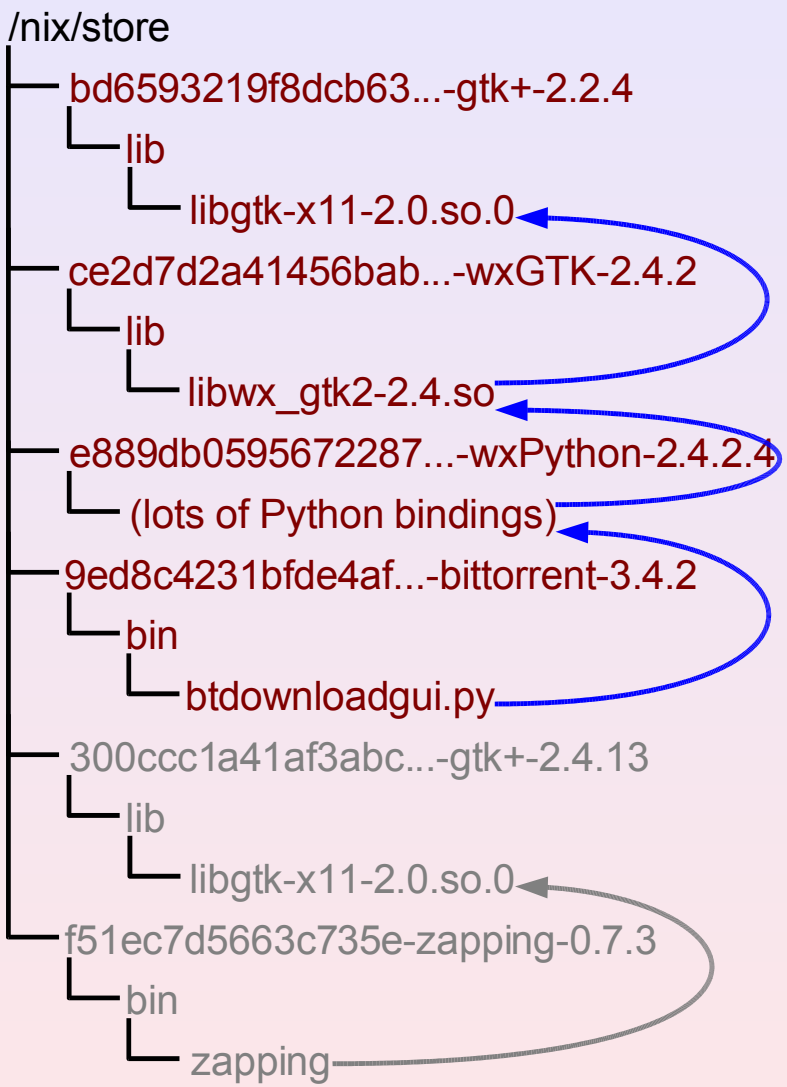
Finding runtime dependencies



Finding runtime dependencies



Finding runtime dependencies



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:

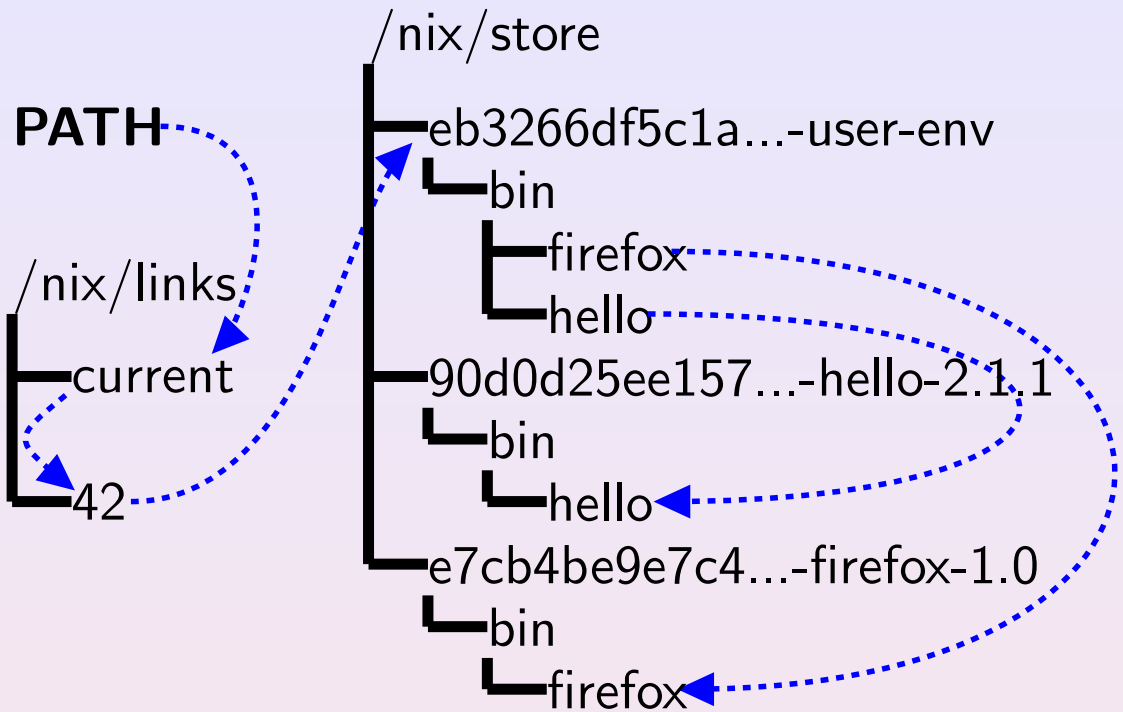
```
$ nix-env --rollback
```

- Delete unused components:

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

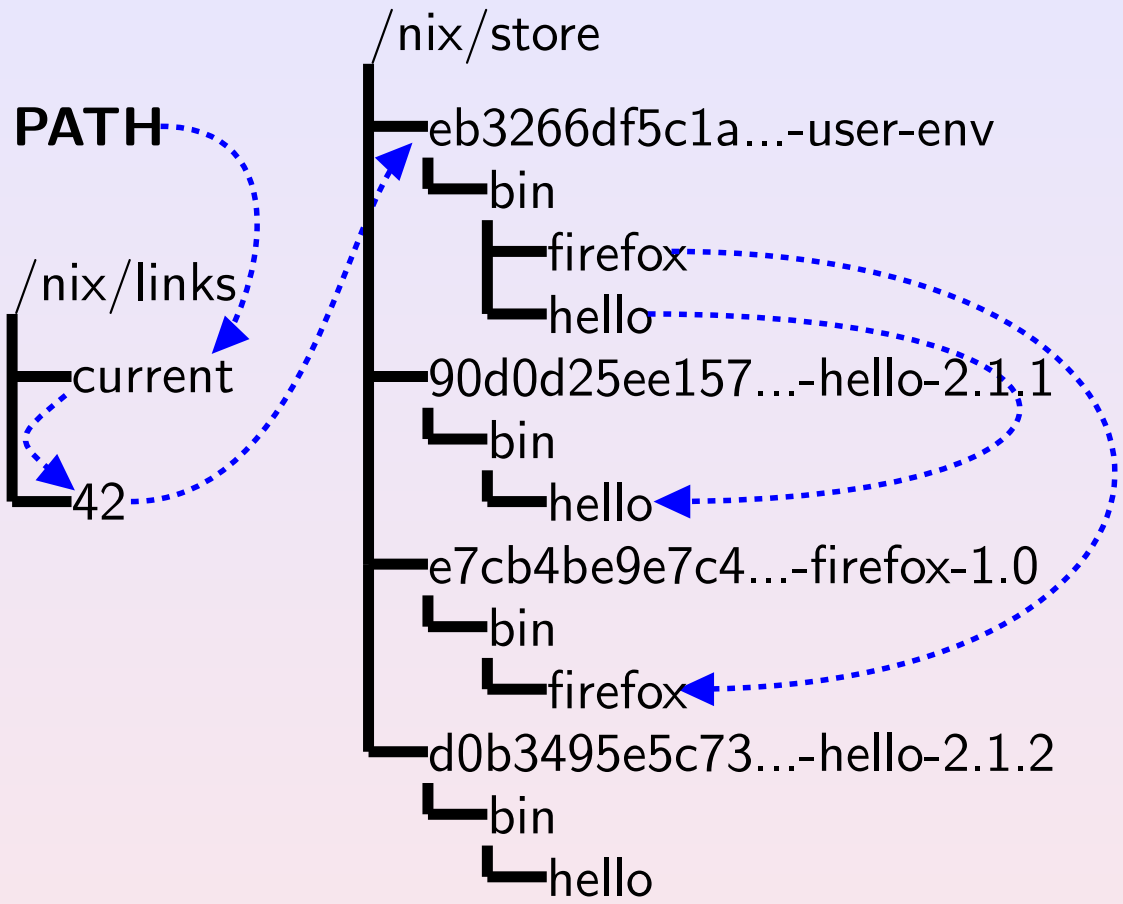
User environments

- 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

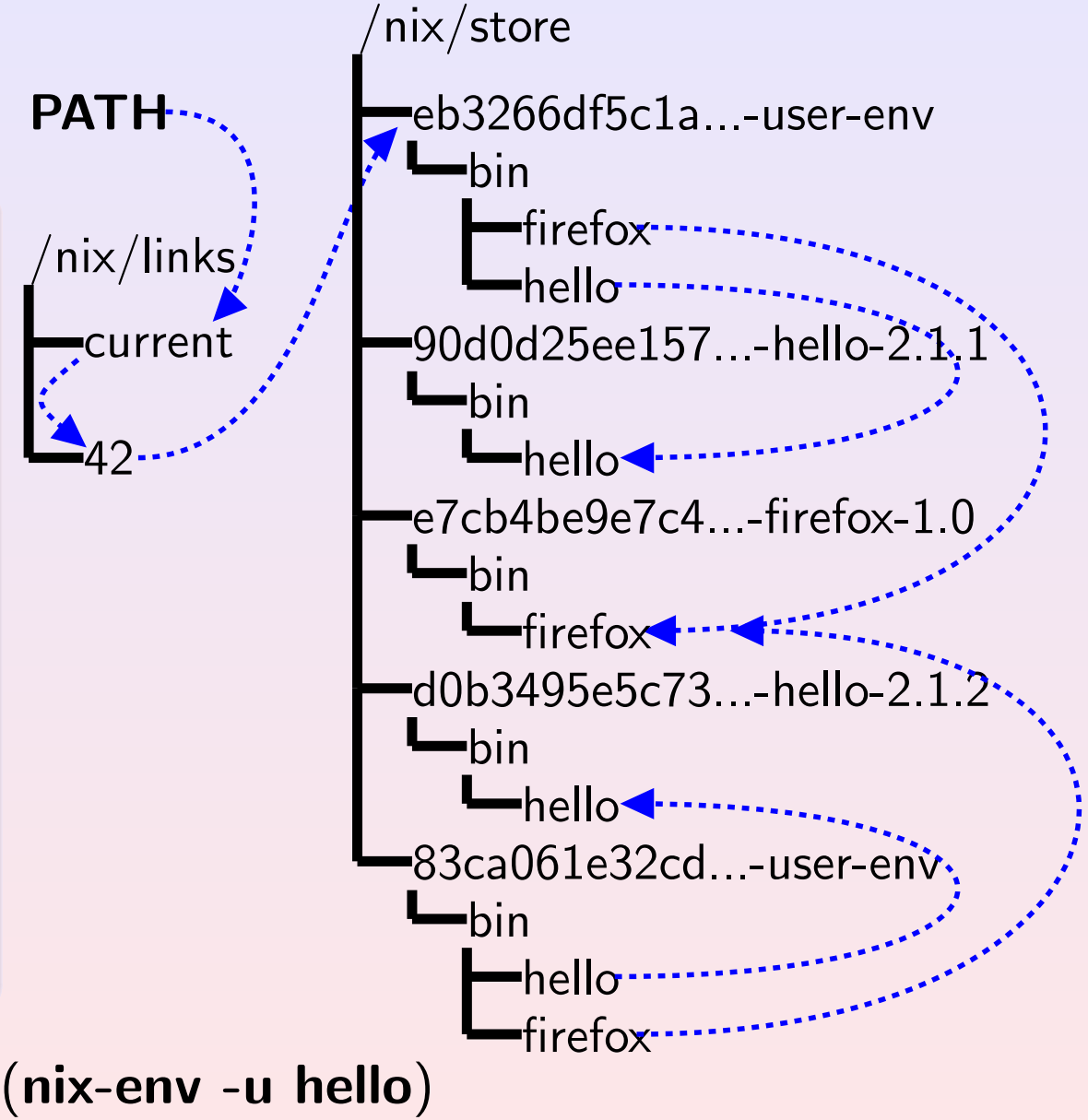
- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the garbage collector.



(nix-env -u hello)

User environments

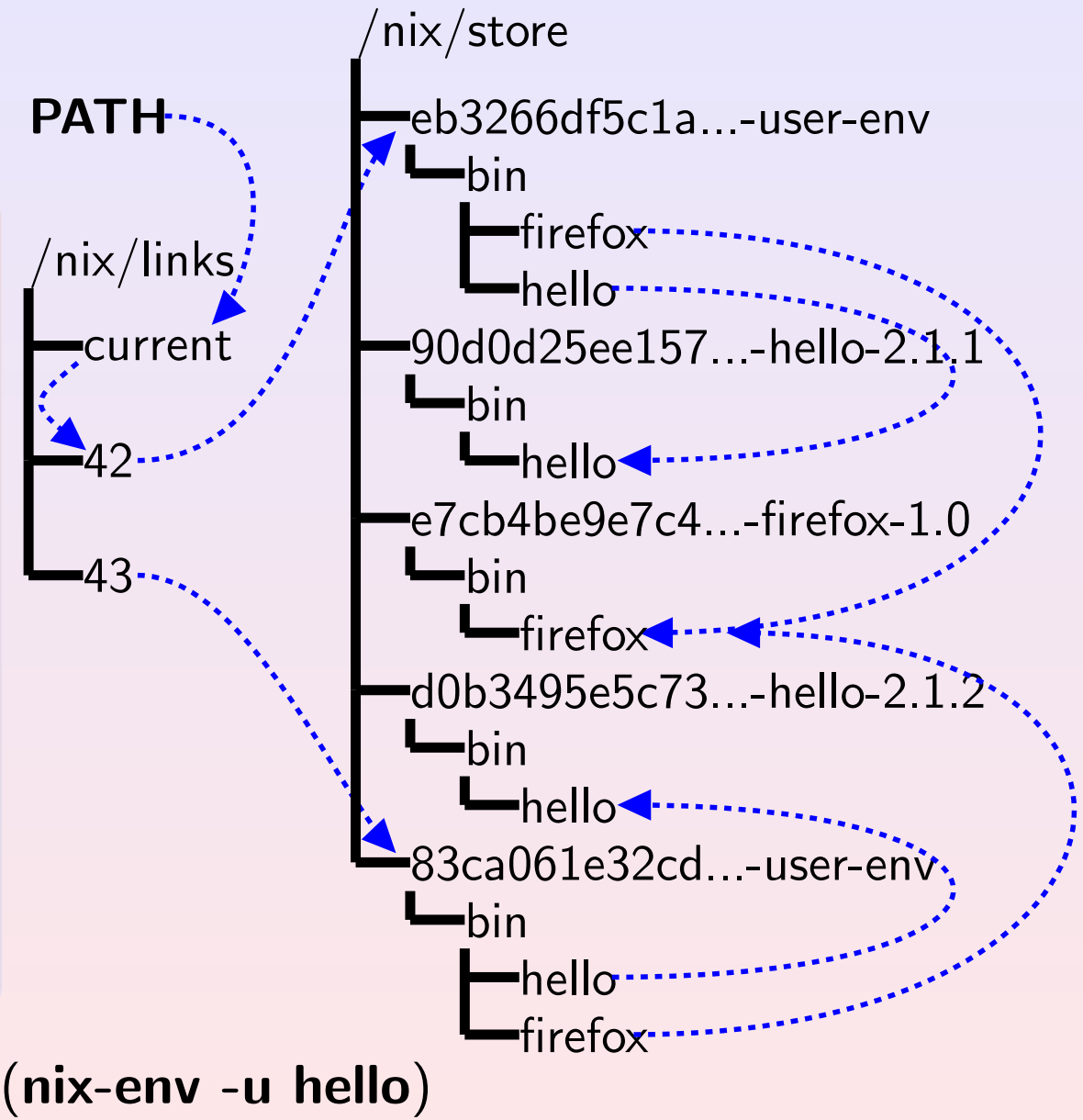
- Users can have different sets of installed applications.
- **nix-env** operations create new **user environments** in the store.
- We can atomically switch between them.
- These are roots of the garbage collector.



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

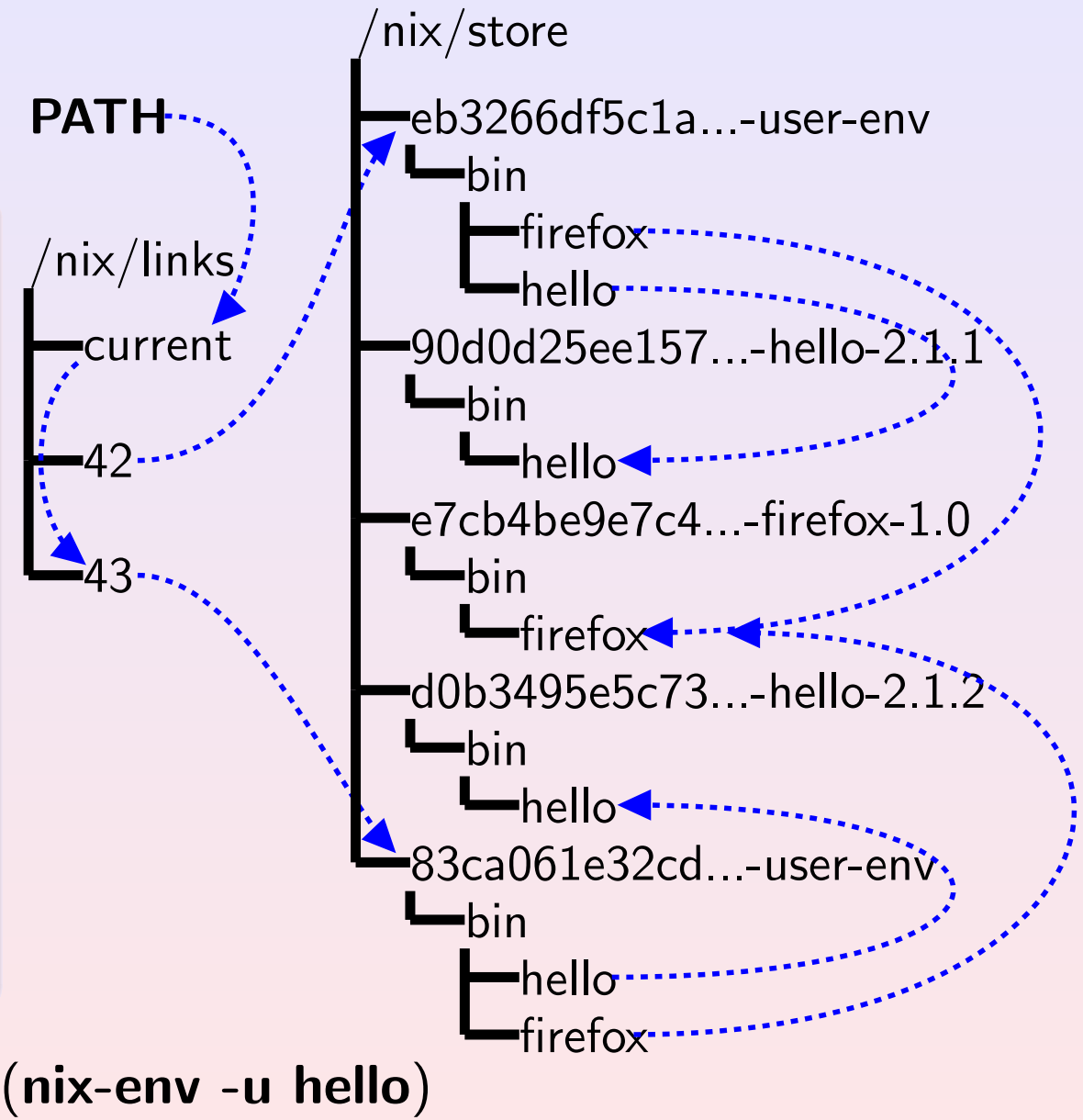
User environments

- 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.



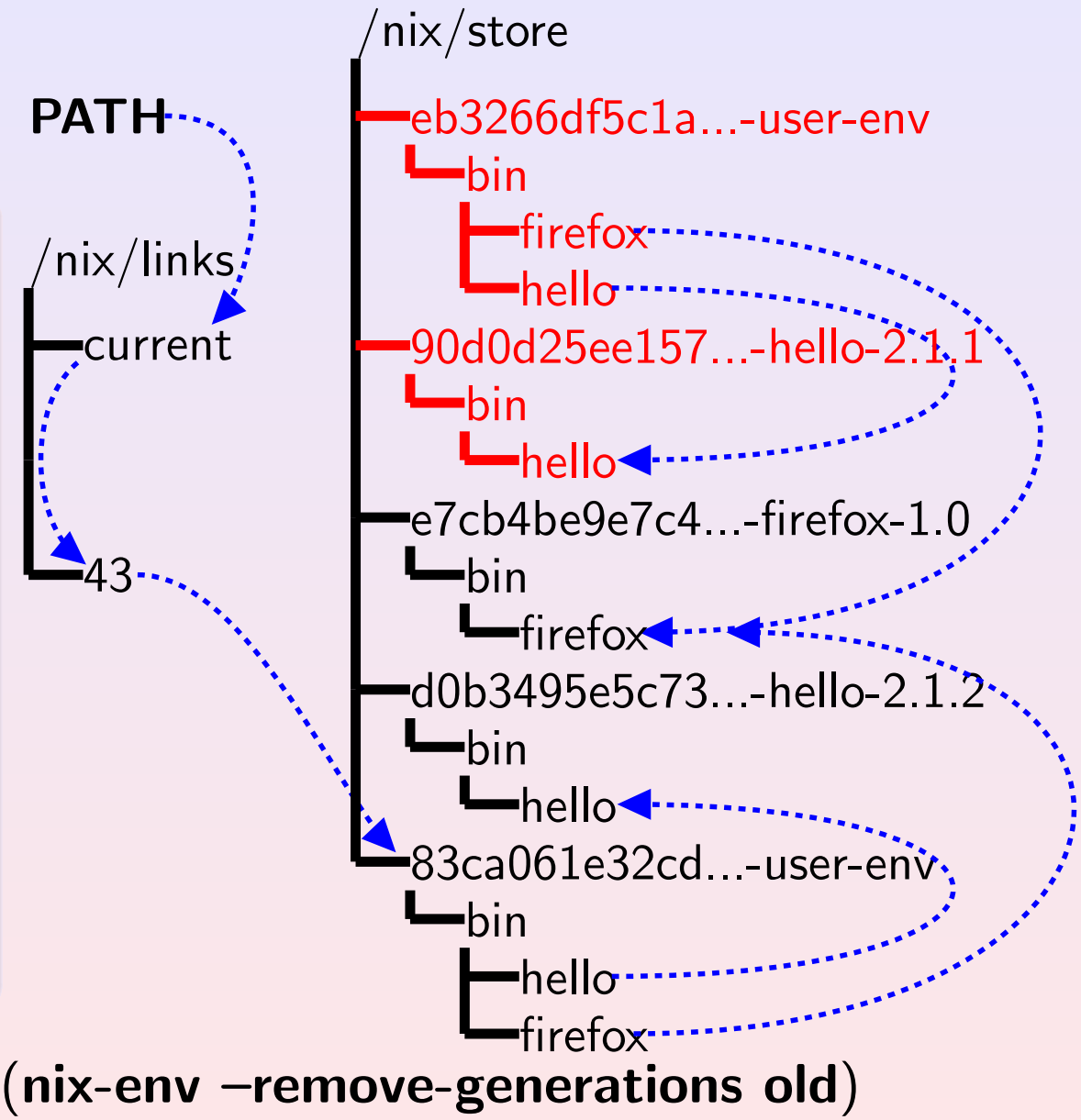
User environments

- 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.



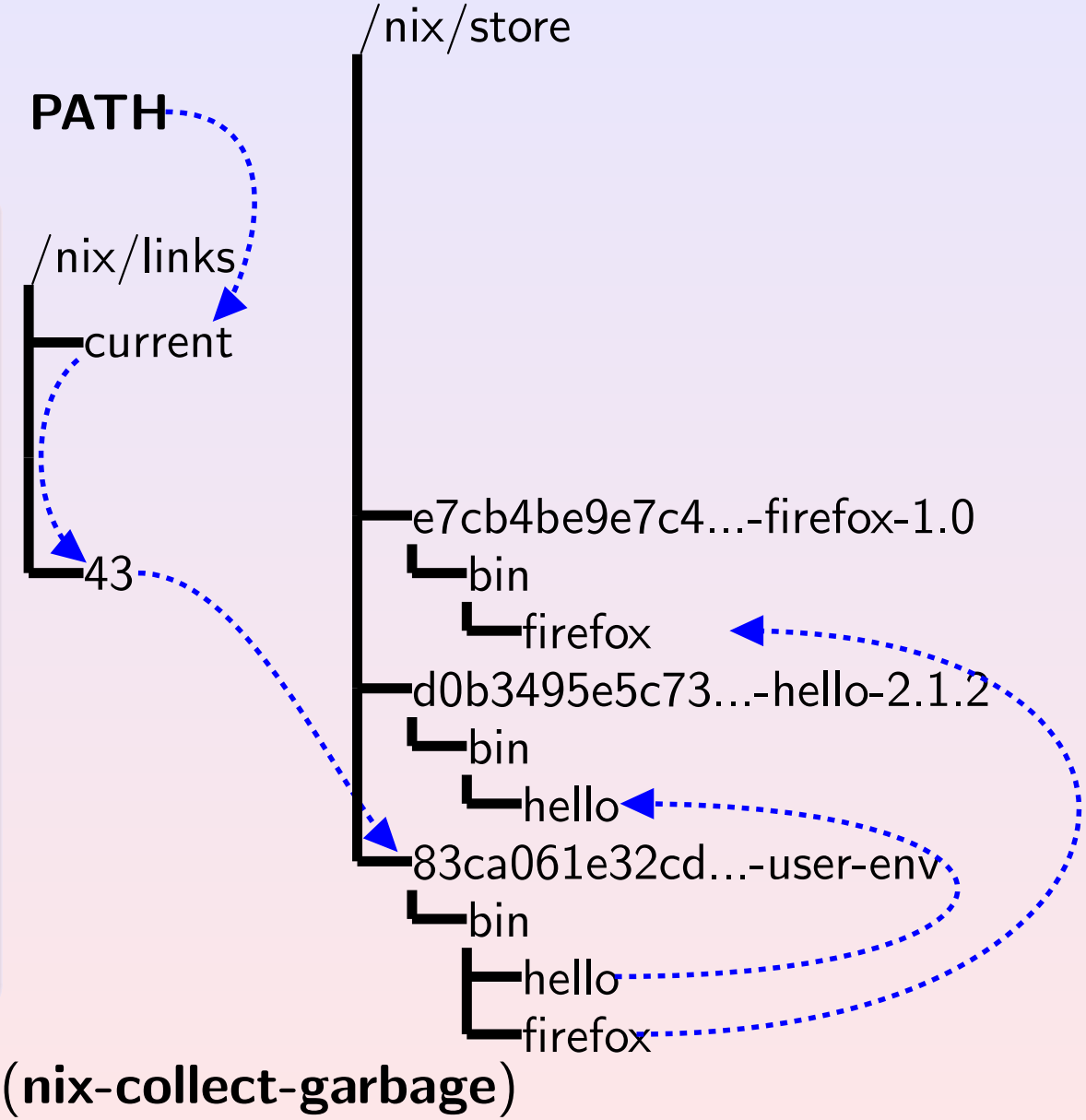
User environments

- 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**.



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-unstable
```

- 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-unstable
```

- 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-unstable
```

- Fetch latest channel instance:

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

- Update all installed packages:

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

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

SCM'03 E. Dolstra, *Integrating Software Construction and Software Deployment*

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*