WEHI
brighter together

BioNix

Nix

- Nix is a purely functional package manager, where packages are...
  - Treated like values
  - Built by functions with no side-effects
  - Never change after being built

- Nix expression language is pure, lazy and functional

https://nixos.wiki/wiki/Overview_of_the_Nix_Language

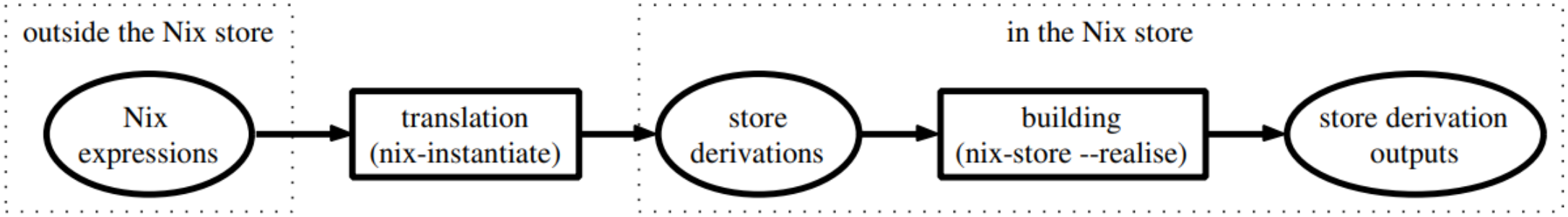
Nix Store

- Packages are stored in Nix store (under /nix/store/), each with a unique identifier that captures all dependencies
  - Each identifier is the *cyptographic hash* of the corresponding package's build dependency graph
- Subdirectories of each package look like this: /nix/store/<id>
  - id (address) has the structure of: a cryptographic hash + package name + package version
  - e.g. /nix/store/b6gvzjyb2pg0kjfwrjmg1vfhh54ad73z-firefox-33.1/

https://nixos.org/manual/nix/stable/language/derivations.html

Derivation

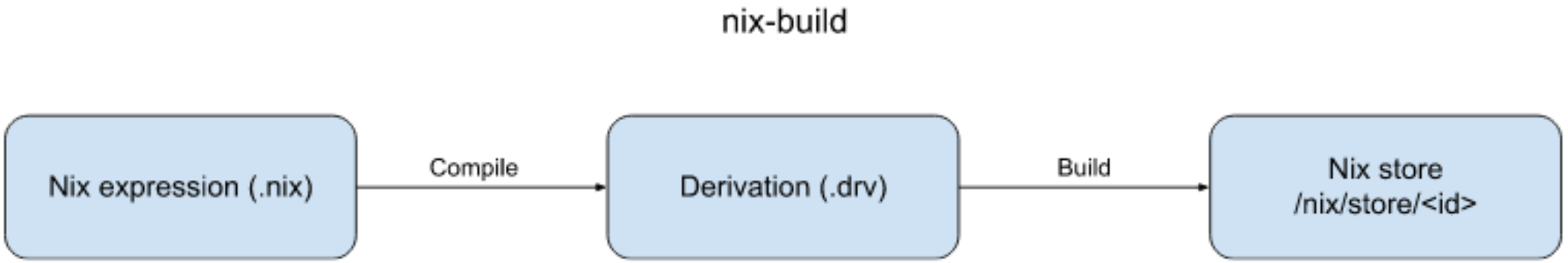
- Derivation describes a single build task.
- Derivation is usually specified in Nix expression.
- Derivation results in a store object.



https://edolstra.github.io/pubs/phd-thesis.pdf

Derivation

- Nix expressions are high-level descriptions of software packages.
- Nix expressions are translated to derivations before they can be stored in Nix store, the derivations can then be built



Flakes

- Flake is a feature of Nix and is the key to high reproducibility
- Flake is a filesystem tree that allows users to specify their code dependencies inside a file called $flake.nix
- Flake is enabled by entering the following line to $/etc/nix/nix.conf
  - `experimental-features = nix-command flakes`

https://www.tweag.io/blog/2020-05-25-flakes/
https://nixos.org/manual/nix/stable/command-ref/new-cli/nix3-flake.html
https://nixos.wiki/wiki/Flakes

flake.nix

- Code dependencies are specified inside $flake.nix

A typical flake.nix looks like this:

```
{
  description = "A flake for building Hello World";

  inputs.nixpkgs.url = github:NixOS/nixpkgs/nixos-20.03;

  outputs = { self, nixpkgs }: {

    packages.x86_64-linux.default =
      # Notice the reference to nixpkgs here.
      with import nixpkgs { system = "x86_64-linux"; };
      stdenv.mkDerivation {
        name = "hello";
        src = self;
        buildPhase = "gcc -o hello ./hello.c";
        installPhase = "mkdir -p $out/bin; install -t $out/bin hello";
      };

  };
}
```

https://nixos.org/manual/nix/stable/command-ref/new-cli/nix3-flake.html

A typical flake.nix in **BioNix** looks like this:

```
{
  description = "Flake for yourSoftware";

  inputs = {
    bionix.url = "github:papenfusslab/bionix";
    nixpkgs.url = "github:nixos/nixpkgs";
    flake-utils.url = "github:numtide/flake-utils";
  };

  outputs = { self, nixpkgs, bionix, flake-utils }:
    flake-utils.lib.eachDefaultSystem
      (system: with bionix.lib
        {
          overlays = [ (self: super: { example = self.callBionix ./example.nix { }; }) ];
          nixpkgs = import nixpkgs { inherit system; };
        };
      {
        defaultPackage = callBionix ./. { };
      }
      );
}
```

https://github.com/victorwkb/BioNix-Doc

flake.lock

- A $flake.lock file will be generated automatically in the flake's directory after executing the build for the first time (via $nix build)

- The version of dependencies will be stored in this lock file, telling Nix to use the exact same versions when re-running the build afterwards.

- There exists commands to update the versions stored in flake.lock:
  - $nix build --recreate-lock-file
  - $nix flake update; $nix build

https://nixos.org/manual/nix/stable/command-ref/new-cli/nix3-flake-update.html