

Gachix

A Binary Cache for Nix over Git

Ephraim Siegfried

16.01.2026

University of Basel

Outline

1. Motivation	2
2. Background	4
3. Design	15
4. Results	38

1. Motivation

1.1. Motivation

Motivation

- Nix and Git share many commonalities
- Most notably, both operate on a directed acyclic graph
- Implementing Nix functionality with Git gives nice features “for free”:
 - Efficient peer-to-peer replication
 - Efficient storage

2. Background

Nix Package Manager

- Declarative and **functional package manager**
- Enforces **reproducibility** in package builds:
 - All dependencies must be specified
 - Building a Nix expression twice yields same result

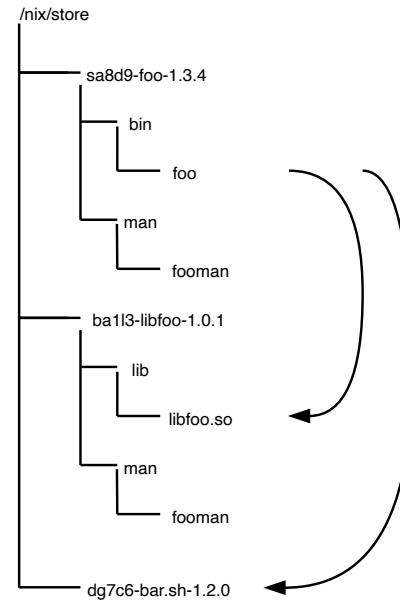


2.1. Nix

Nix Store

- **Collection of packages** and other data
- Each entry is **immutable**
- Uniquely **identified by hash** of dependency graph
- Entries are of the form:

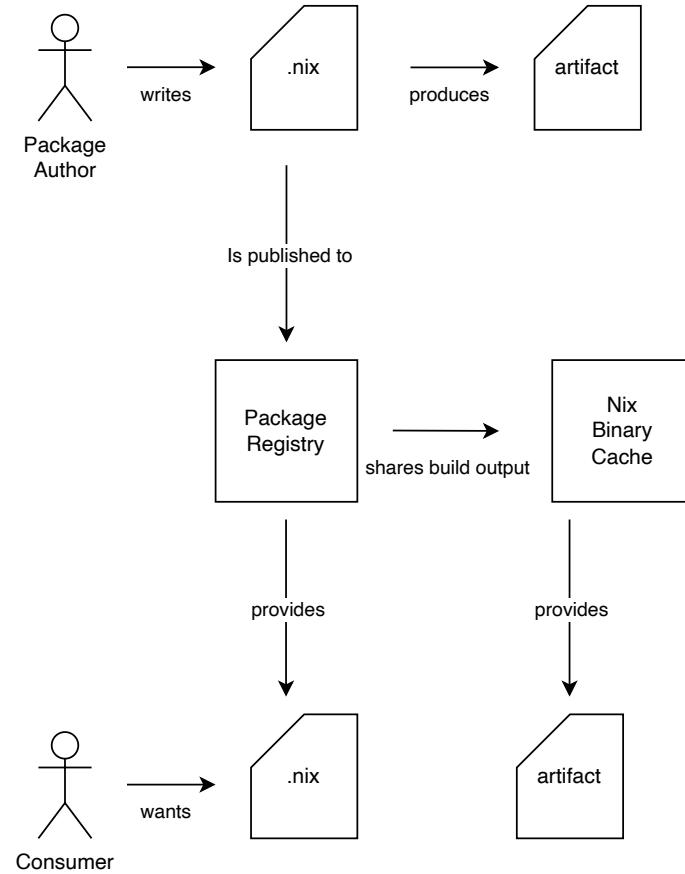
`/nix/store/<hash>-<name>-<version>`



2.1. Nix

Deployment Pipeline

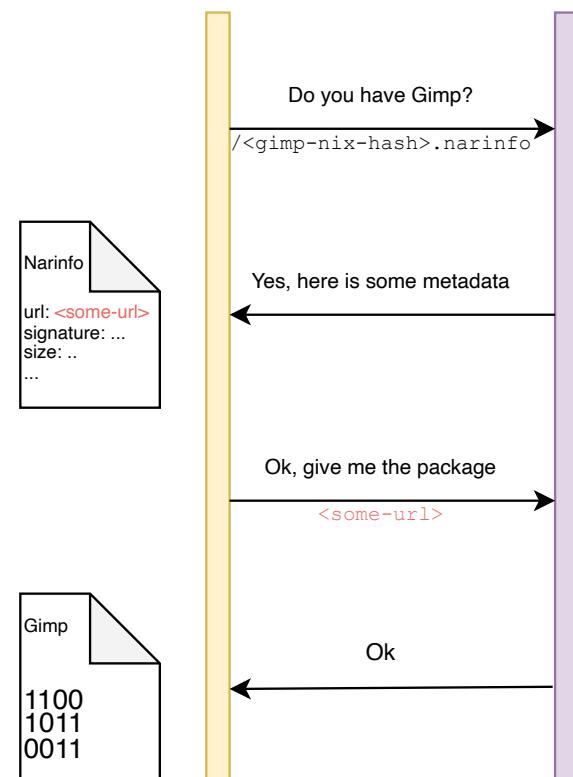
- Nix produces packages from Nix files
- A Nix file is a build recipe for a package
- Building a package can take a long time
→ Use binary caches to speed up builds



2.1. Nix

Binary Cache Protocol

- Occurs when executing:
`nix build <some-package>`
- Protocol over HTTPS
- User asks for **Narinfo**, which contains URL to package contents
- Receives package in the **Nix Archive** (NAR) format

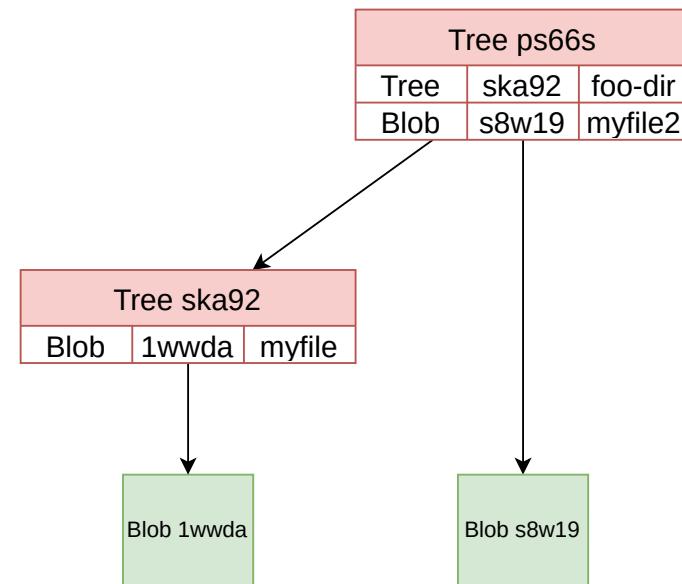


Git

- Advertised as distributed version control system
- Instead a tool for:
 - Manipulation of a **directed acyclic graph** (DAG)
 - **Content-addressable objects**
 - **Replication** of these objects across repositories

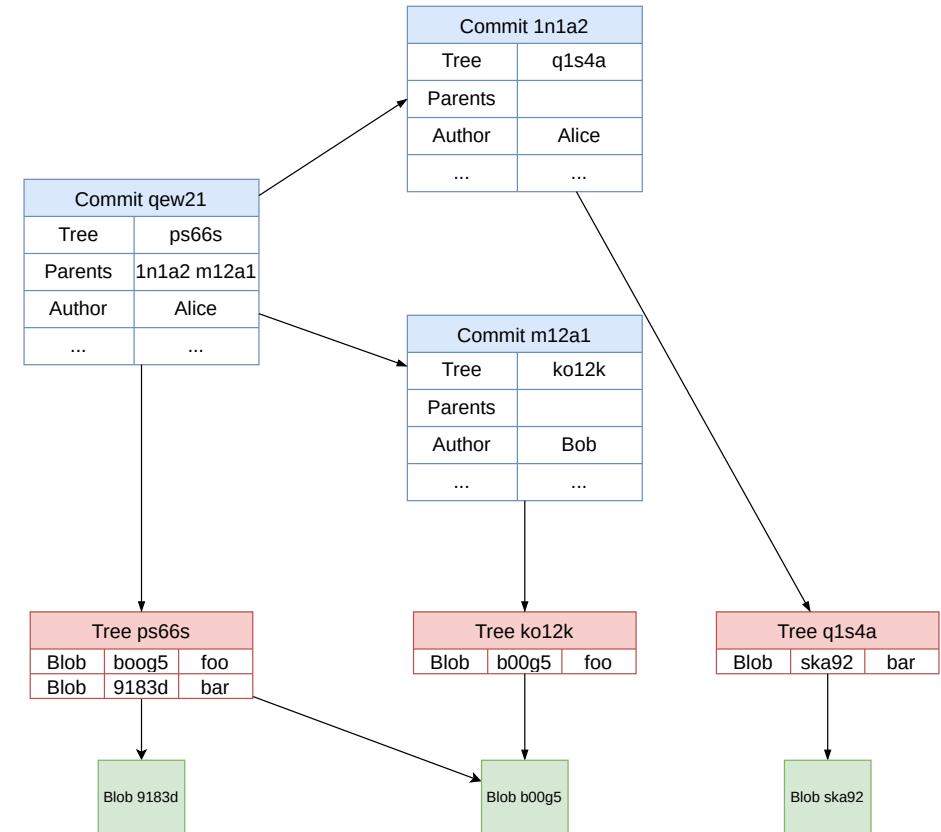
Git Objects: Blob and Tree

- **Blob**: Sequence of bytes, usually stores files
- **Tree**: Collection of pointers to blobs or trees.



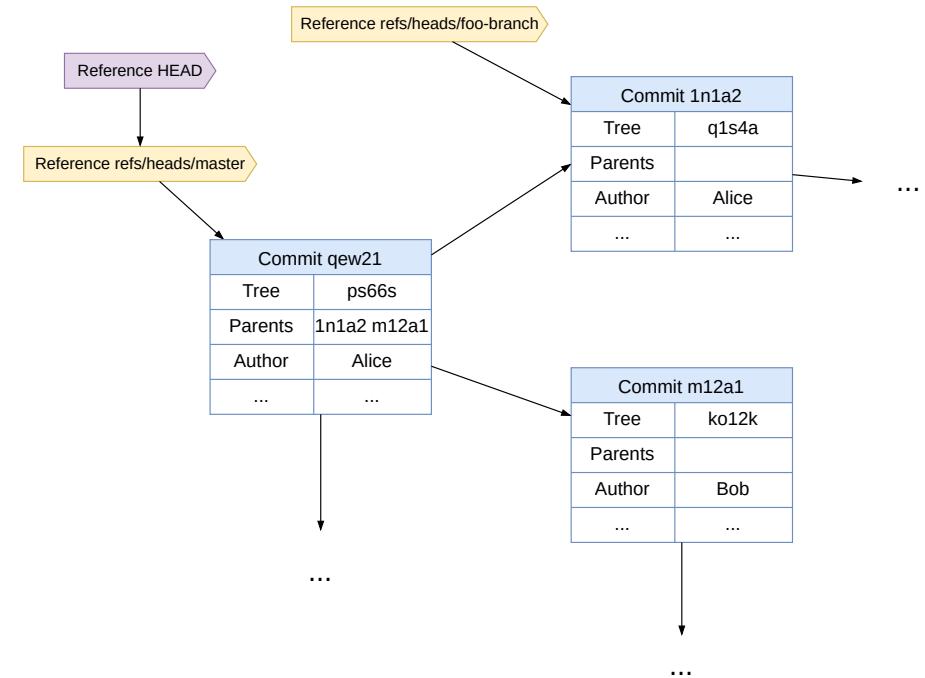
Git Objects: Commit

- **Commit** contains:
 - ▶ Pointer to **exactly one tree**
 - ▶ **Parent Pointers**: Pointers to other commits
 - ▶ Author (name and mail)
 - ▶ Timestamp
 - ▶ Message



Git Objects: Reference

- **References:** Pointer to Git objects
- **Direct Reference:** Points to blobs, trees, commits (e.g. branches and tags)
- **Symbolic Reference:** Point to direct references (e.g. HEAD)



Git Objects

- Blobs, trees and commits are **immutable**
- Blobs, trees and commits are **content-addressed** (stored in `.git/objects`)
- References are mutable and identified by a given name (stored in `.git/refs`)

Replication

- Synchronize objects with `git fetch <refspec>`
- Specify objects with **refsspecs**:
 - Constructed as `<remote_references>:<local_references>`
 - Copies the specified remote references
 - **Downloads all objects reachable** from the specified references
 - E.g. the command `git fetch refs/foo:refs/foo` copies `refs/foo` and downloads all object reachable from it

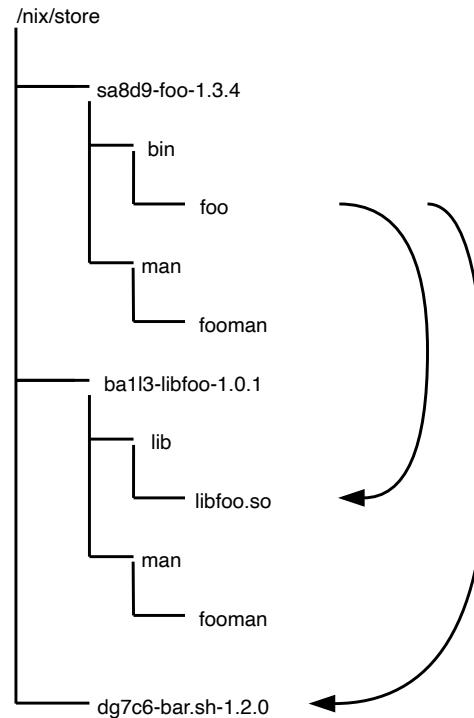
3. Design

3.1. Mapping Nix to Git

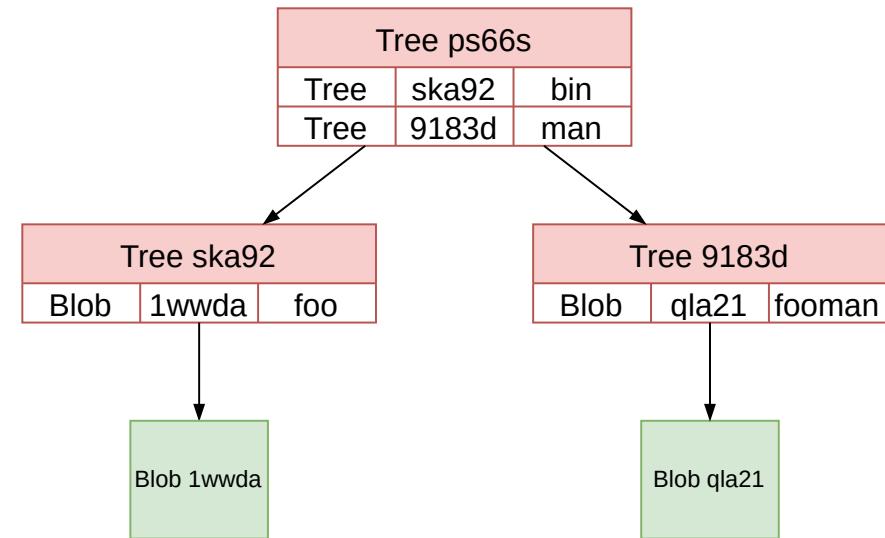
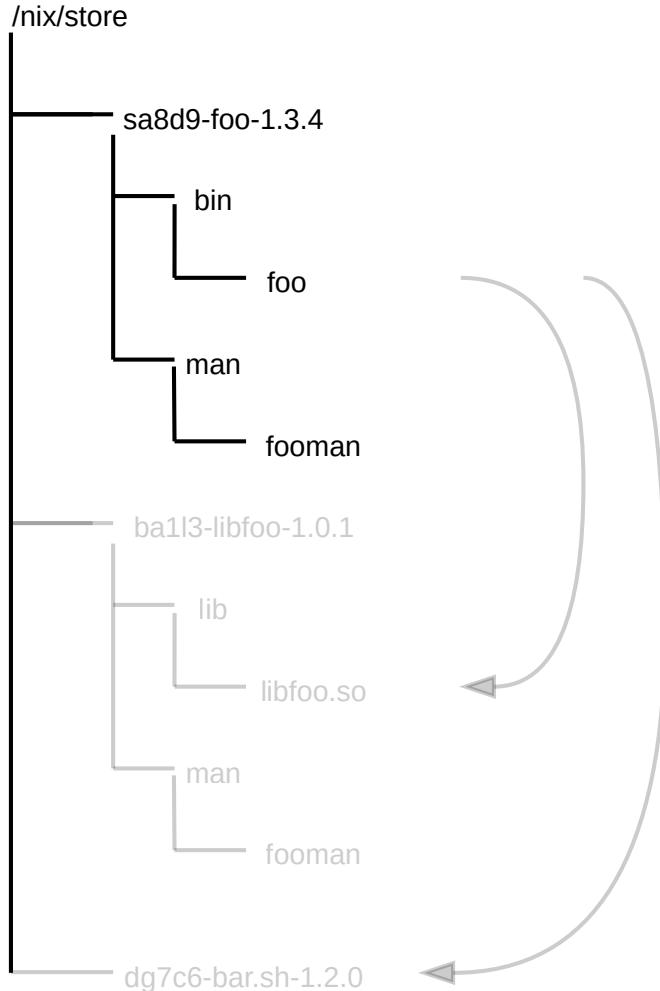
Nix to Git

- Goal: Store Nix packages in a Git database.
- Map files to blobs
- Map directories to trees
- Model dependencies between packages using commit pointers

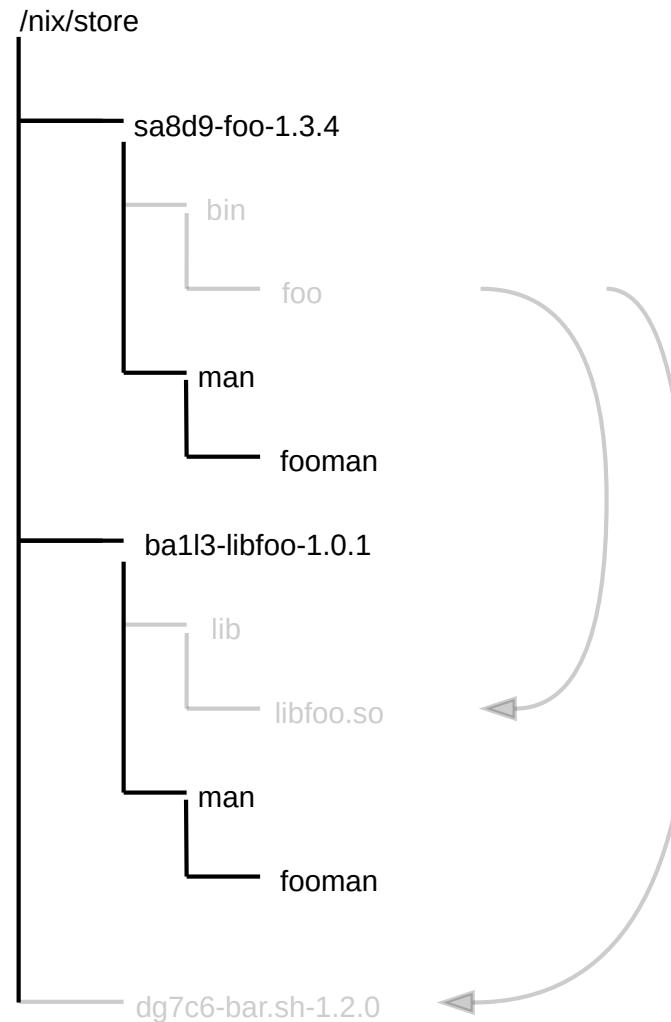
3.1. Mapping Nix to Git



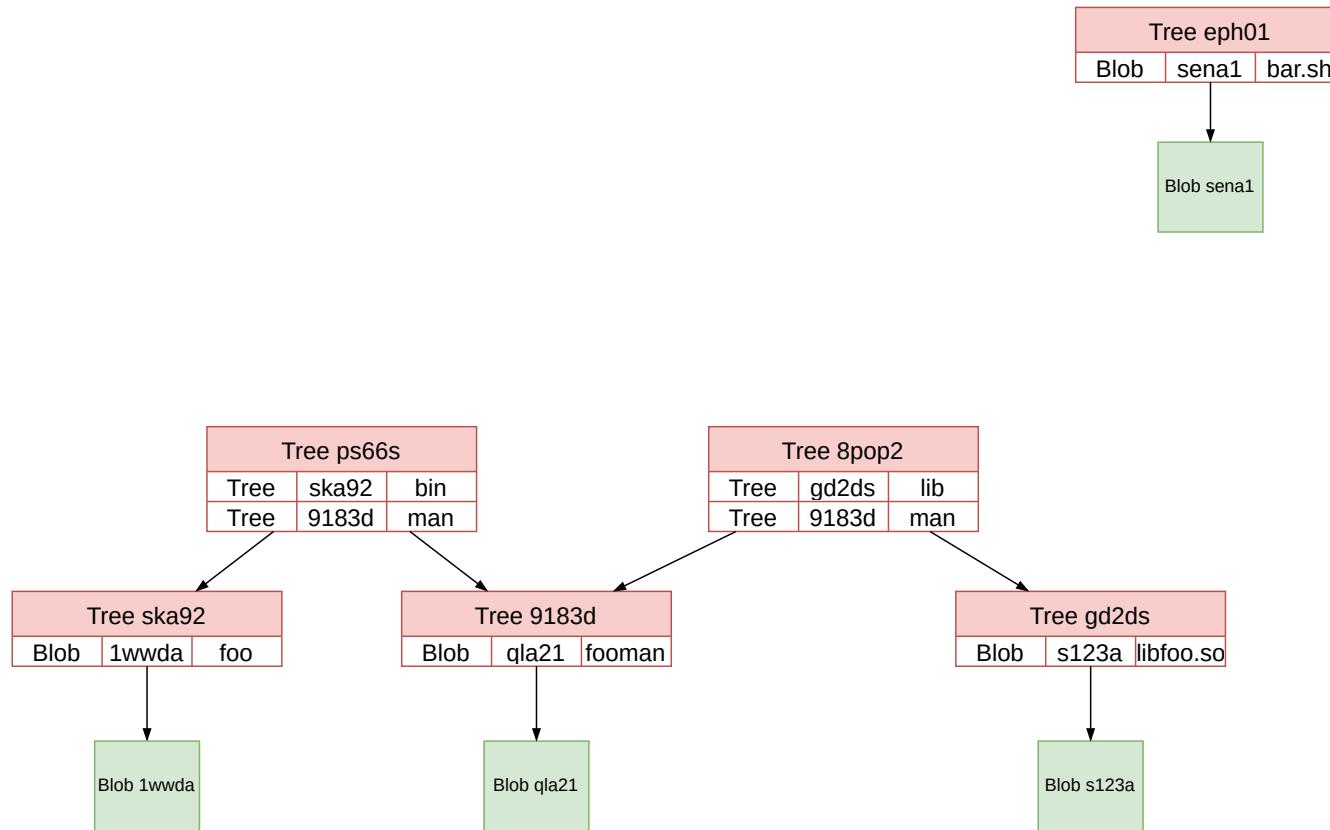
3.1. Mapping Nix to Git



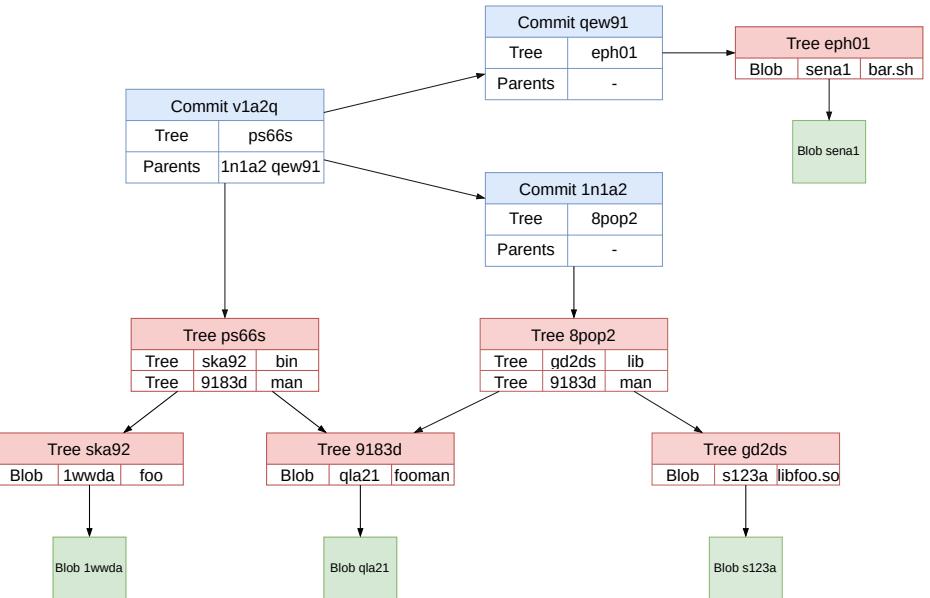
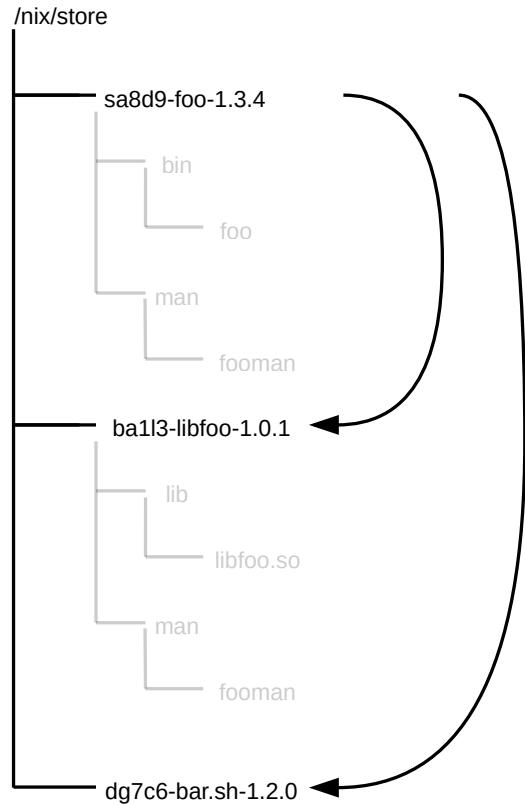
3.1. Mapping Nix to Git



3.1. Mapping Nix to Git



3.1. Mapping Nix to Git

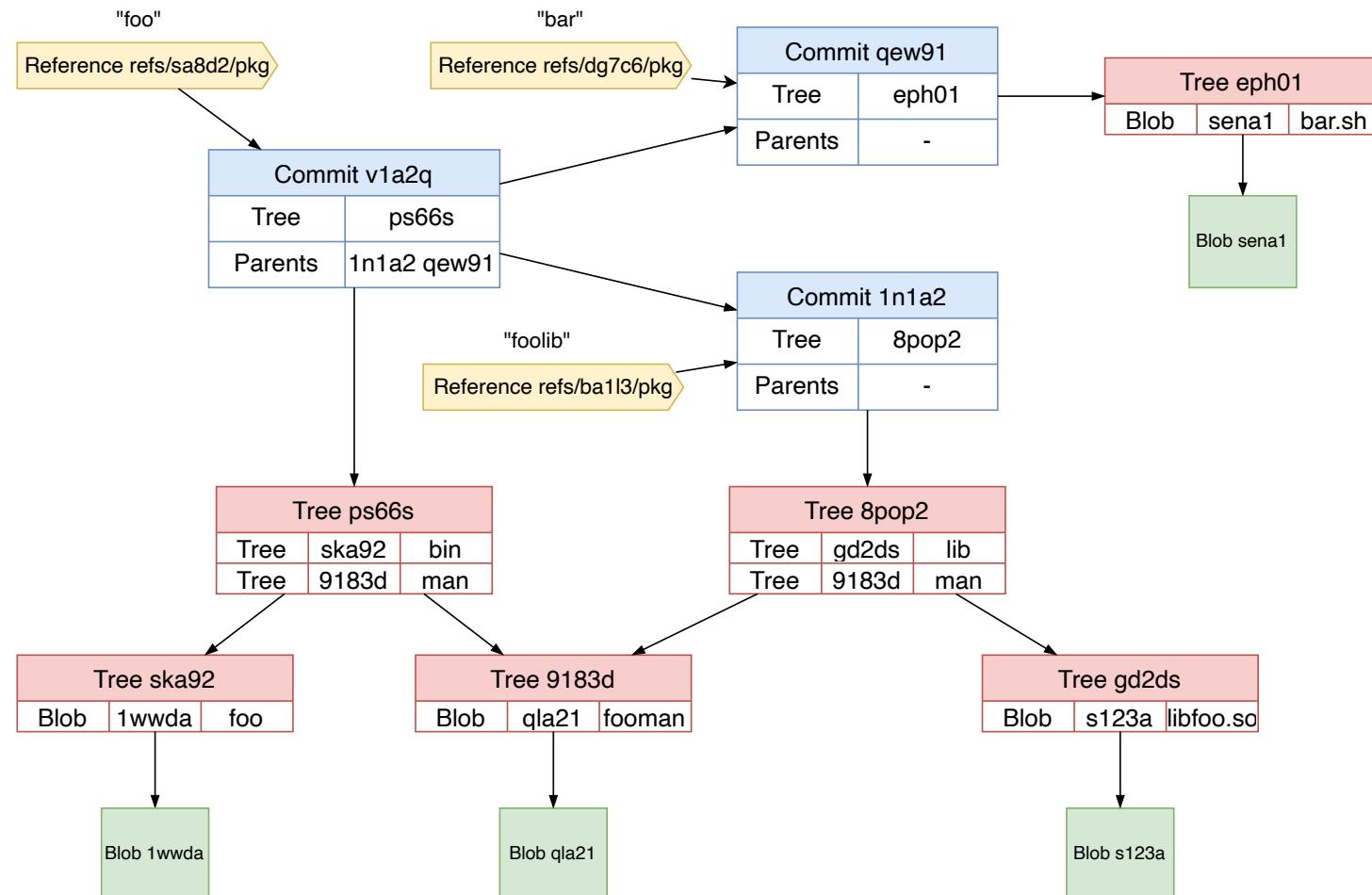


3.2. Dependency Management

Dependency Management

- Every package is associated with exactly one commit object
- Parents of the commit are dependencies of the package
- To ensure a global bijective mapping between commit hash and package:
Set commit message, timestamp, and author field to constant values
- Maintain mapping between Nix hashes and commit hashes using references in the form:
`refs/<nix-hash>/pkg`

3.2. Dependency Management

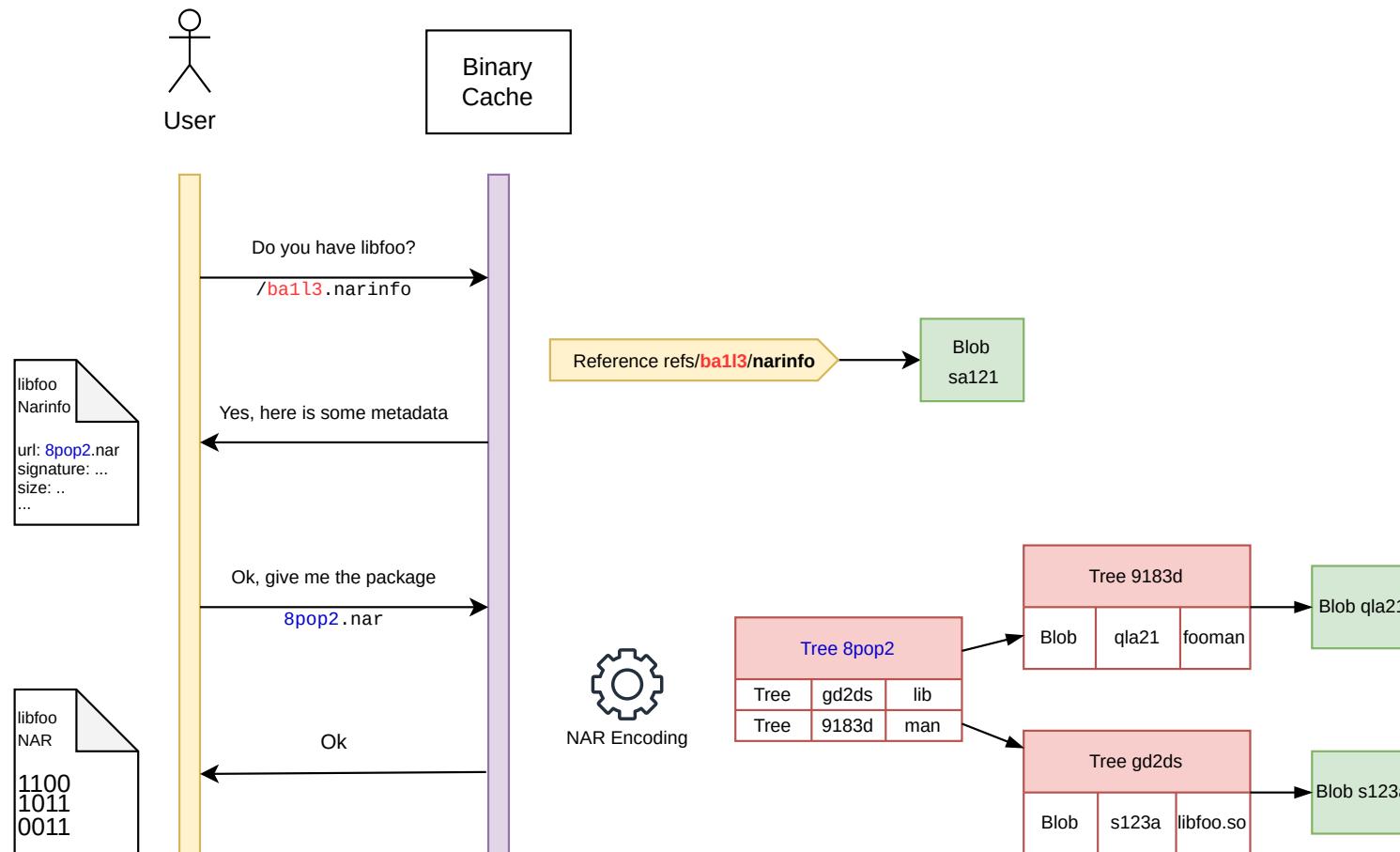


3.3. Binary Cache Protocol

Binary Cache Protocol

- Everytime a package is added to the Git database, the Narinfo is constructed and the reference `/refs/<nix-hash>/narinfo` points to it
- The server transforms package into Nix Archive (NAR) and streams it to the user

3.3. Binary Cache Protocol



3.4. Replication

Replication

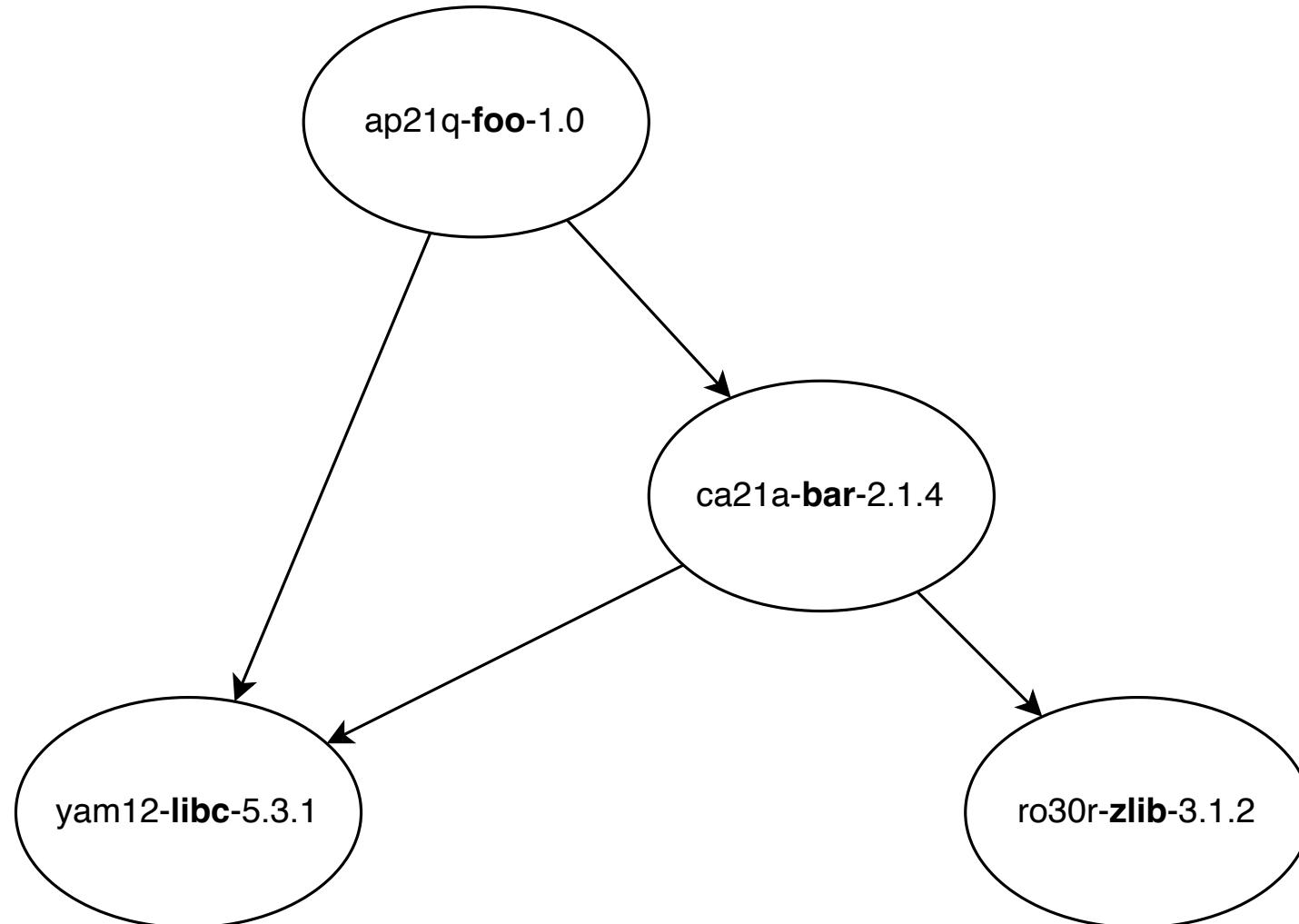
- There is no policy (yet) on when a package is added to the cache
- Packages are either added by **fetching commits from peers** or **constructed using Nix interface**

3.4. Replication

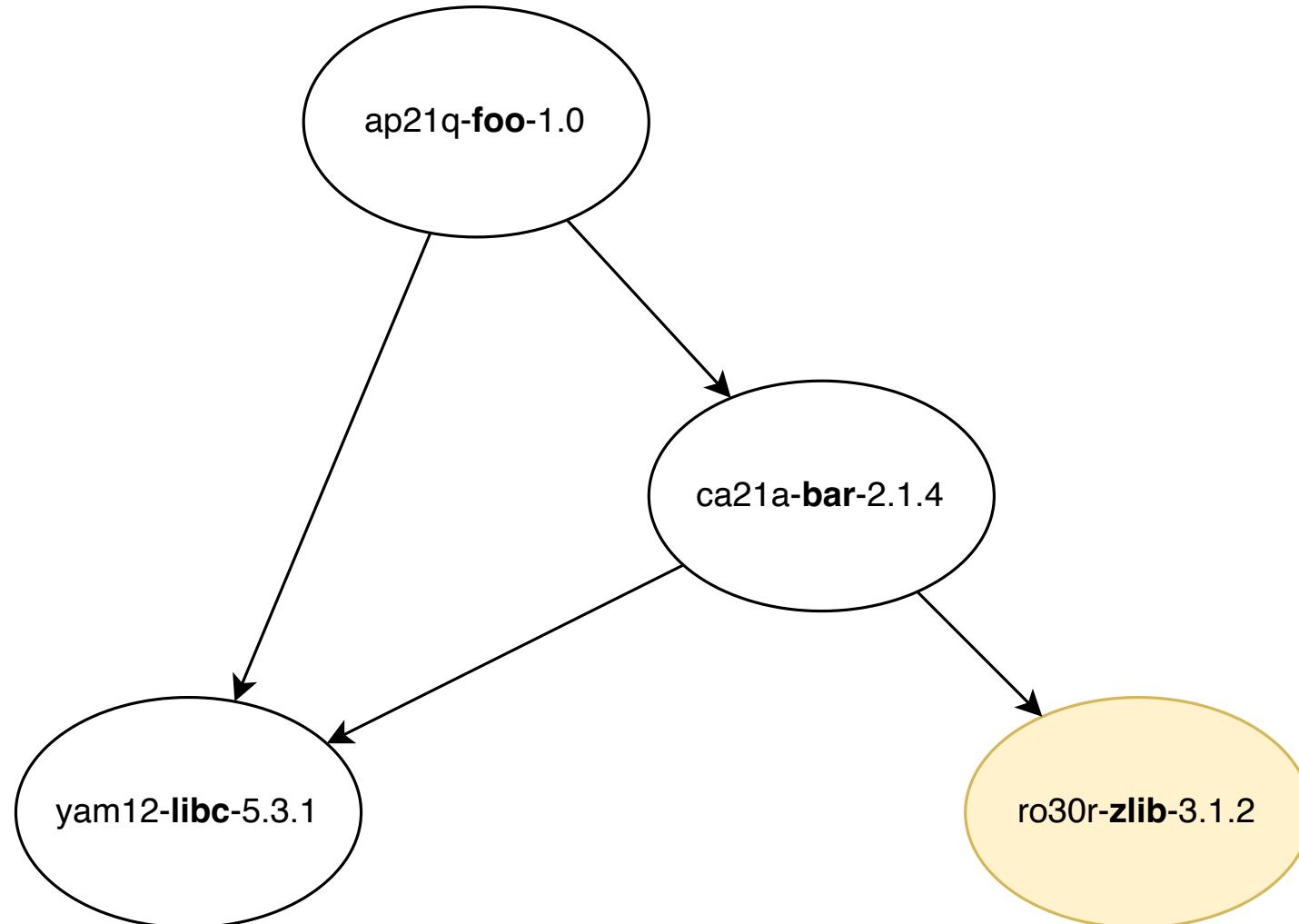
Replication with Nix interface

- Explore the dependency graph of a package in a **depth first search** manner:
 - Iterate through dependencies
 - Fully *process* one dependency, including all its dependencies, before moving to the next dependency in the list
 - *Processing* includes:
 - Fetch NAR using Nix interface and decode it to Git objects
 - Build the Narinfo
 - Create the package commit and set references

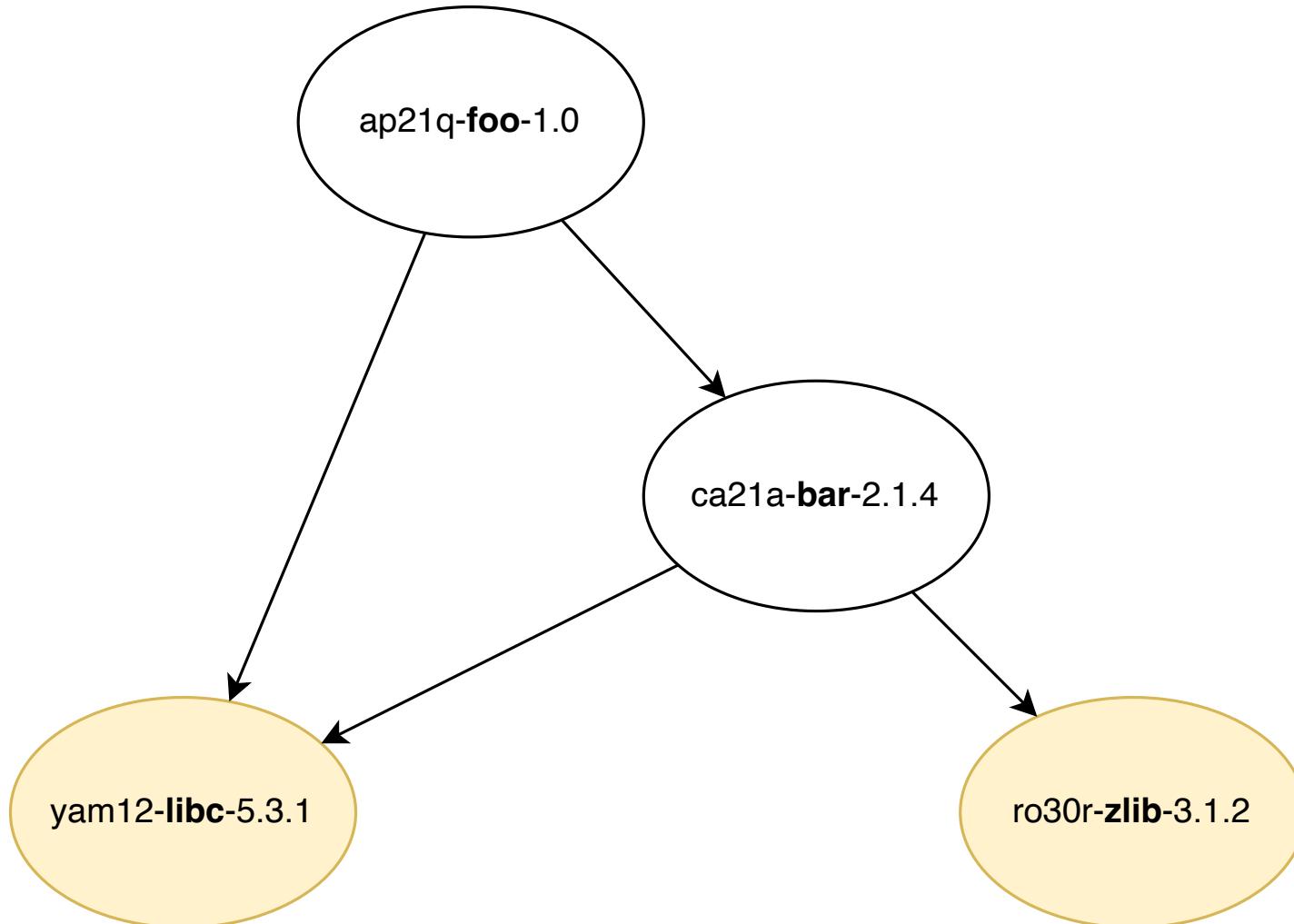
3.4. Replication



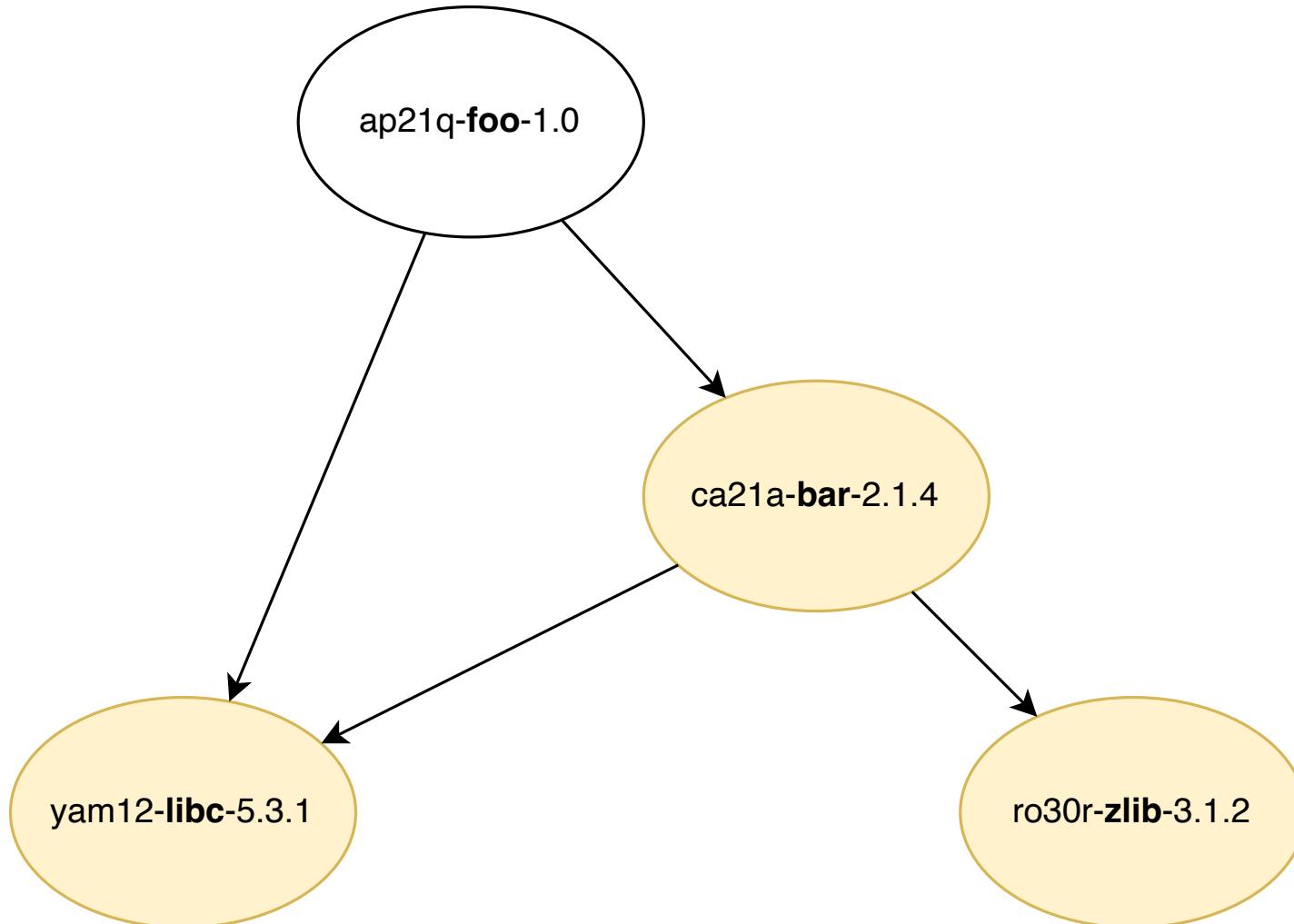
3.4. Replication



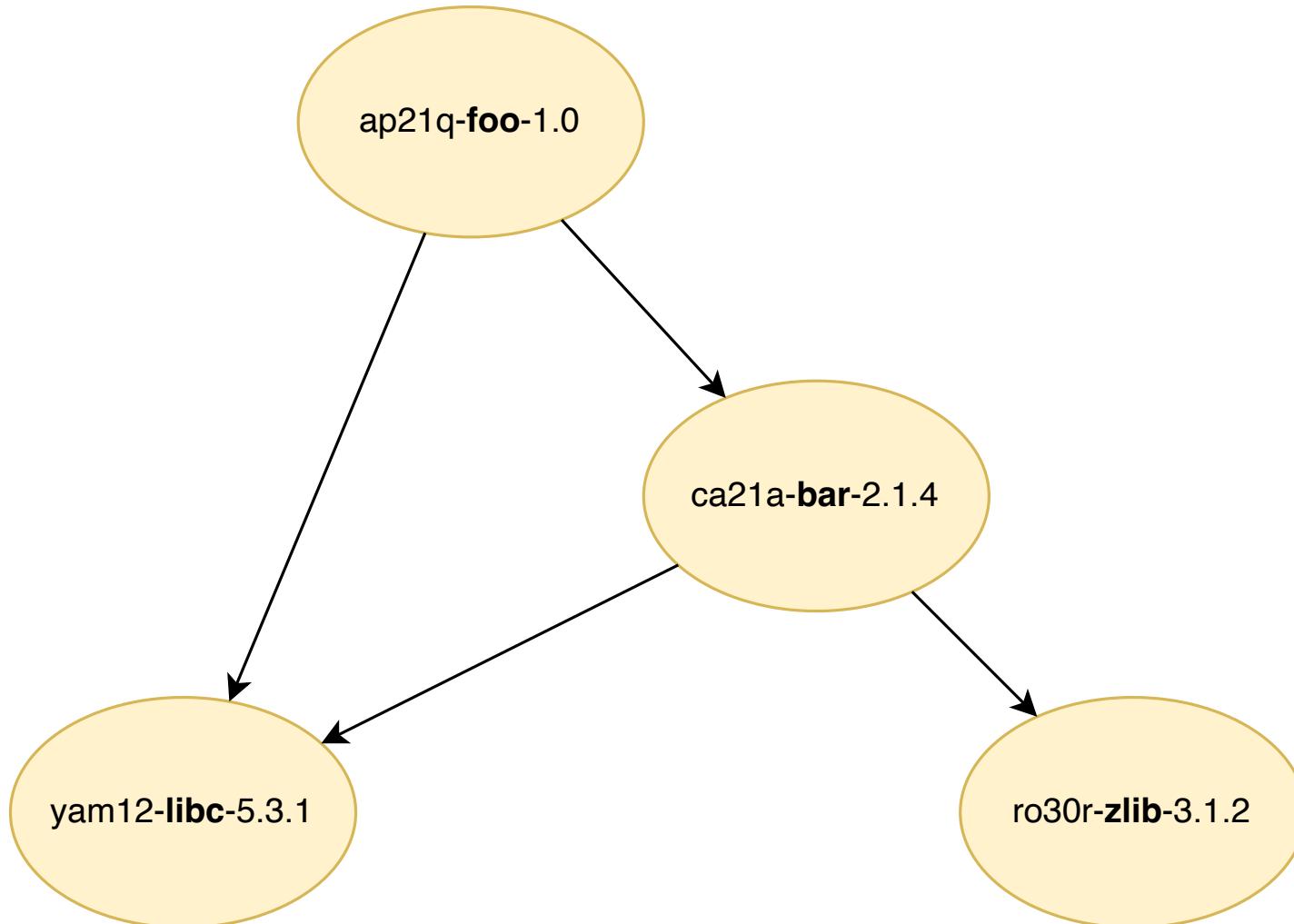
3.4. Replication



3.4. Replication



3.4. Replication



3.4. Replication

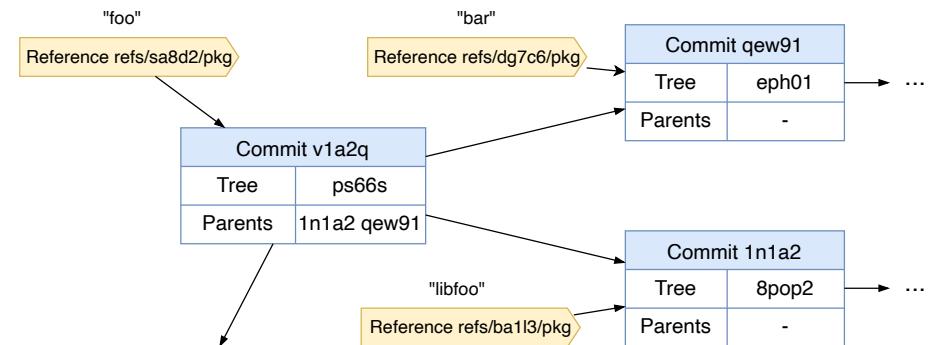
Replication with Gachix peers

- Do `git fetch refs/<h>/*:refs/<h>/*` where `<h>` is the hash of the requested package
→ fetches the reference `refs/<h>` and all objects reachable from it
- For every dependency fetch the missing reference object
- This can be done in a breadth first search manner

3.4. Replication

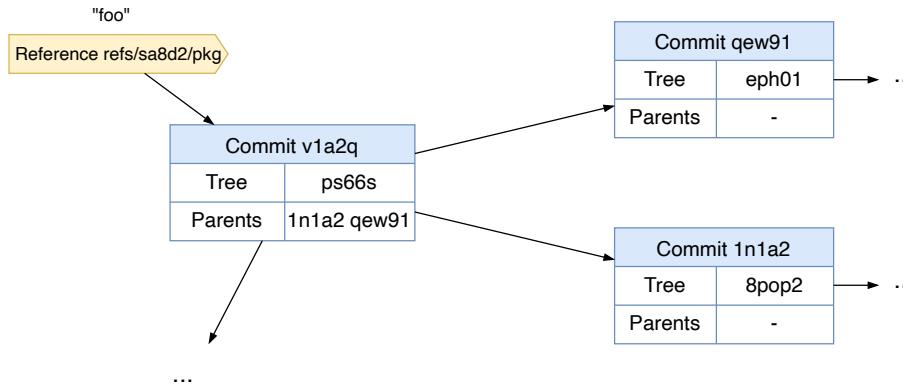
Bob Repository

Alice Repository

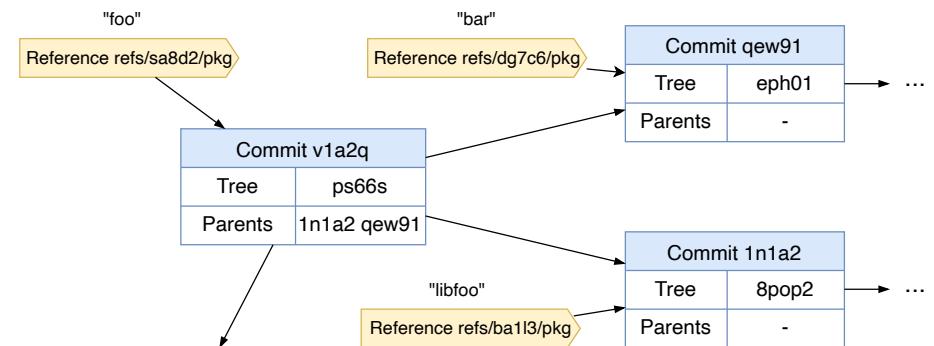


3.4. Replication

Bob Repository

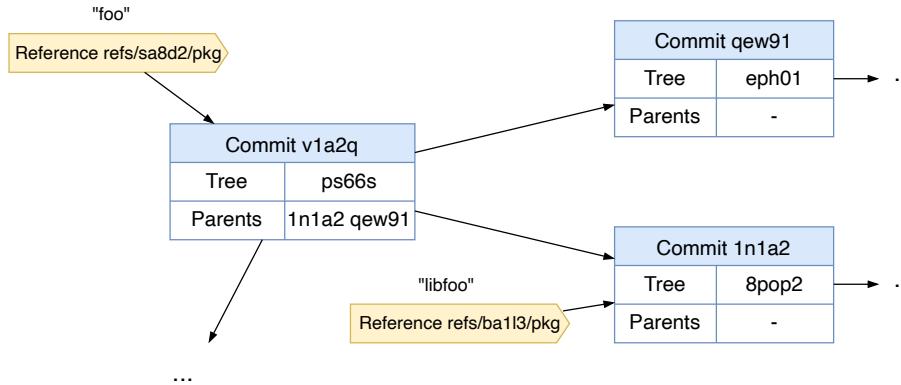


Alice Repository

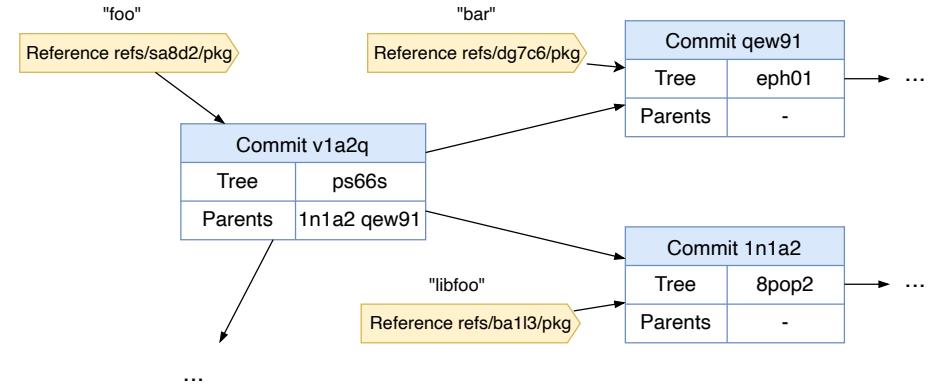


3.4. Replication

Bob Repository

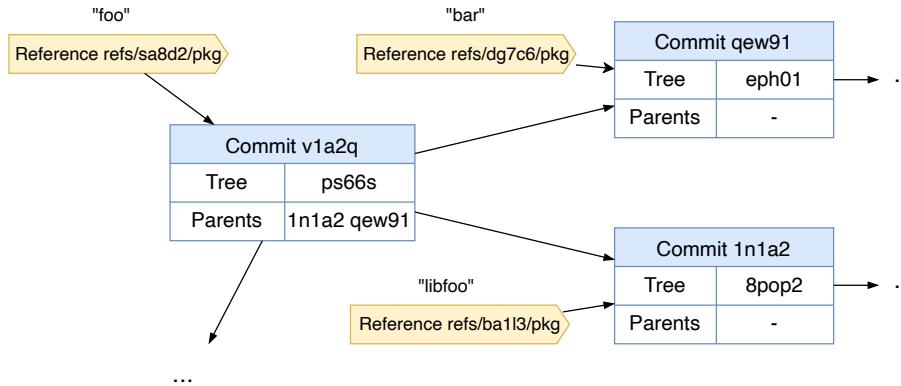


Alice Repository

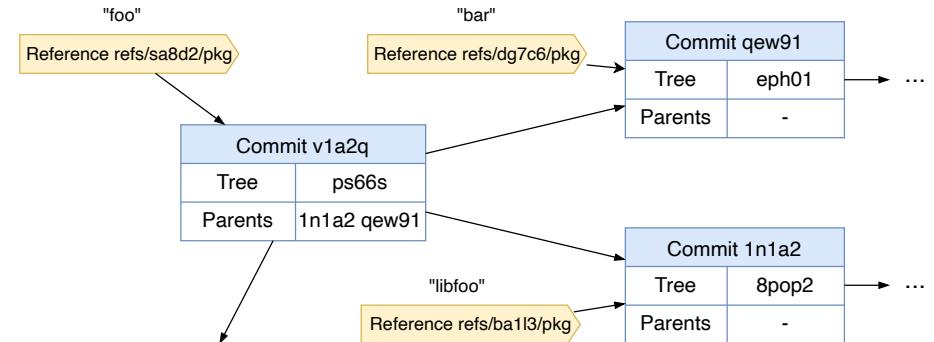


3.4. Replication

Bob Repository



Alice Repository



4. Results

4.1. Results Overview

Results

1. Gachix achieves the **lowest median latency** but shows **slower average** performance.
2. Gachix is **more storage efficient** than other cache services.
3. Gachix can be **deployed on any Unix machine**, including on systems without Nix installed.
4. Gachix is **transparent**: It can be used with the Nix interface.

4.2. Package Retrieval Latency

Methodology

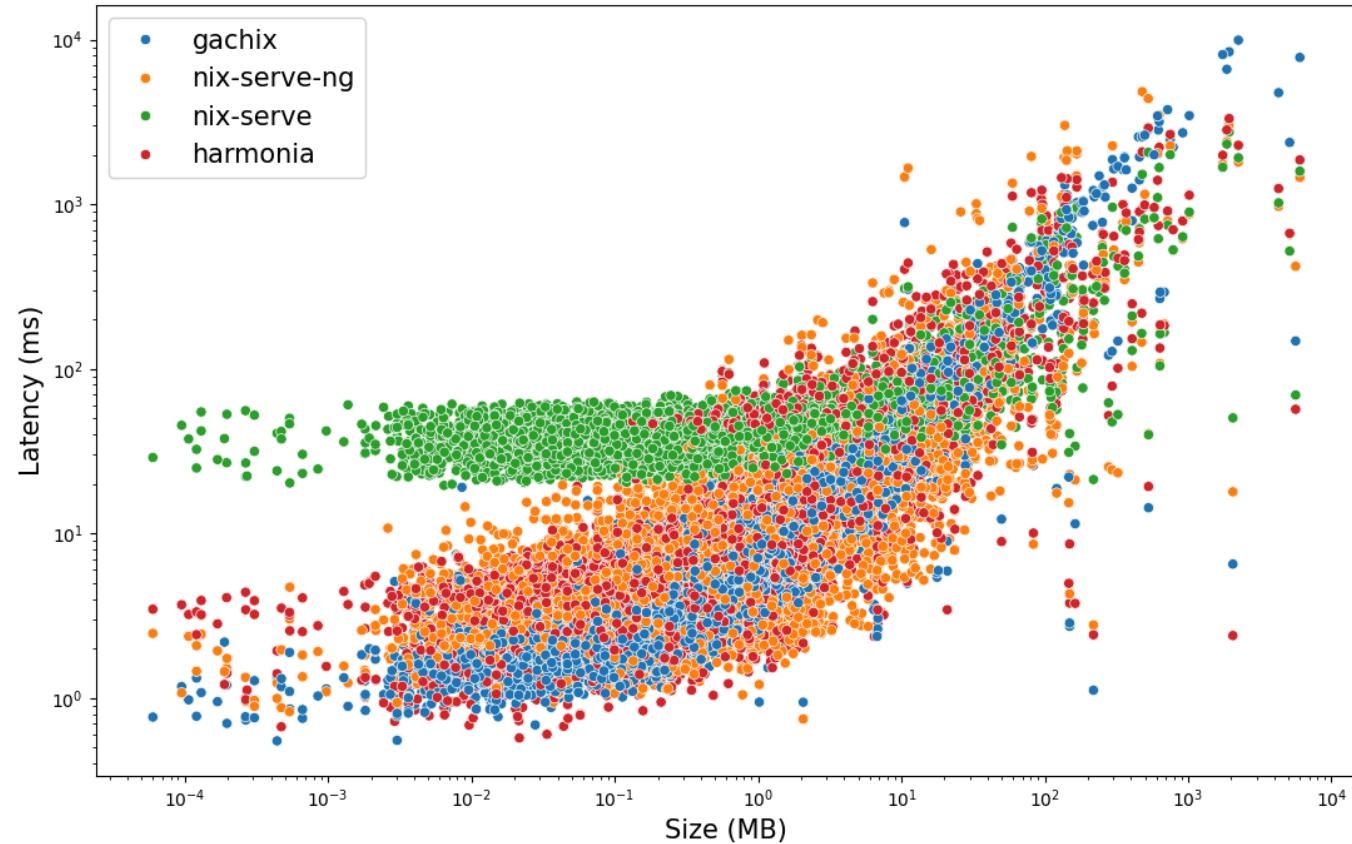
1. Randomly select 650 packages from the Nixpkgs registry (from two releases)
2. Install them along with every required dependency → **5123 packages** in total
3. Add them to the Gachix cache
4. Start each cache service and **measure end-to-end latency** of NAR retrieval of each package

4.2. Package Retrieval Latency

Results

Cache Service	Median	p95	p99	Max	Mean	Std
gachix	4.812	142.129	840.199	9931.217	49.347	327.198
harmonia	8.530	119.514	604.240	3316.529	41.912	146.475
nix-serve	42.063	101.205	447.337	2749.474	57.757	107.114
nix-serve-ng	7.689	105.879	616.550	4832.431	37.989	182.101

4.2. Package Retrieval Latency



4.3. Package Storage

Methodology

- Same as in the previous benchmark
- Add 5123 packages to the Nix store and to Gachix
- Measure size of .git and sum of the size of all packages in Nix store

4.3. Package Storage

Result

- Sum of package sizes in the Nix store is 77.87 GB
- Size of .git repository is 13.45 GB
- **Size reduction of 82.72%**
- 21.84% of Git objects had an indegree > 1

Thank you for your attention!
