FOSDEM (February 1, 2020)

NTT

Lazy distribution of container images

Current implementation status of containerd remote snapshotter

Akihiro Suda

Credit to Kohei Tokunaga (NTT) for containerd impl. & benchmark scripts

Summary

- Run containers before completion of downloading the images
- Lots of alternative image formats are proposed to support this
- **stargz** is getting wide adoption (containerd & Podman)

Demo:
Lazy distribution of docker.io/library/python:3.7

The problems of the current Docker / OCI format

Current Docker / OCI format

- **Open Containers Initiative (OCI)** defines the standard specifications for containers
  - Docker/Moby, Podman, Kubernetes (containerd, CRI-O, ...), Singularity...
- **OCI Image Spec**: defines the **tar ball** structure and the JSON metadata format
  - Based on Docker Image Manifest V2 Schema 2
- **OCI Distribution Spec**: defines the API for distributing images via HTTP
  - Based on Docker Registry HTTP API
- Focuses on legacy rather than on innovation ☹️

TAR: Tape ARchiver

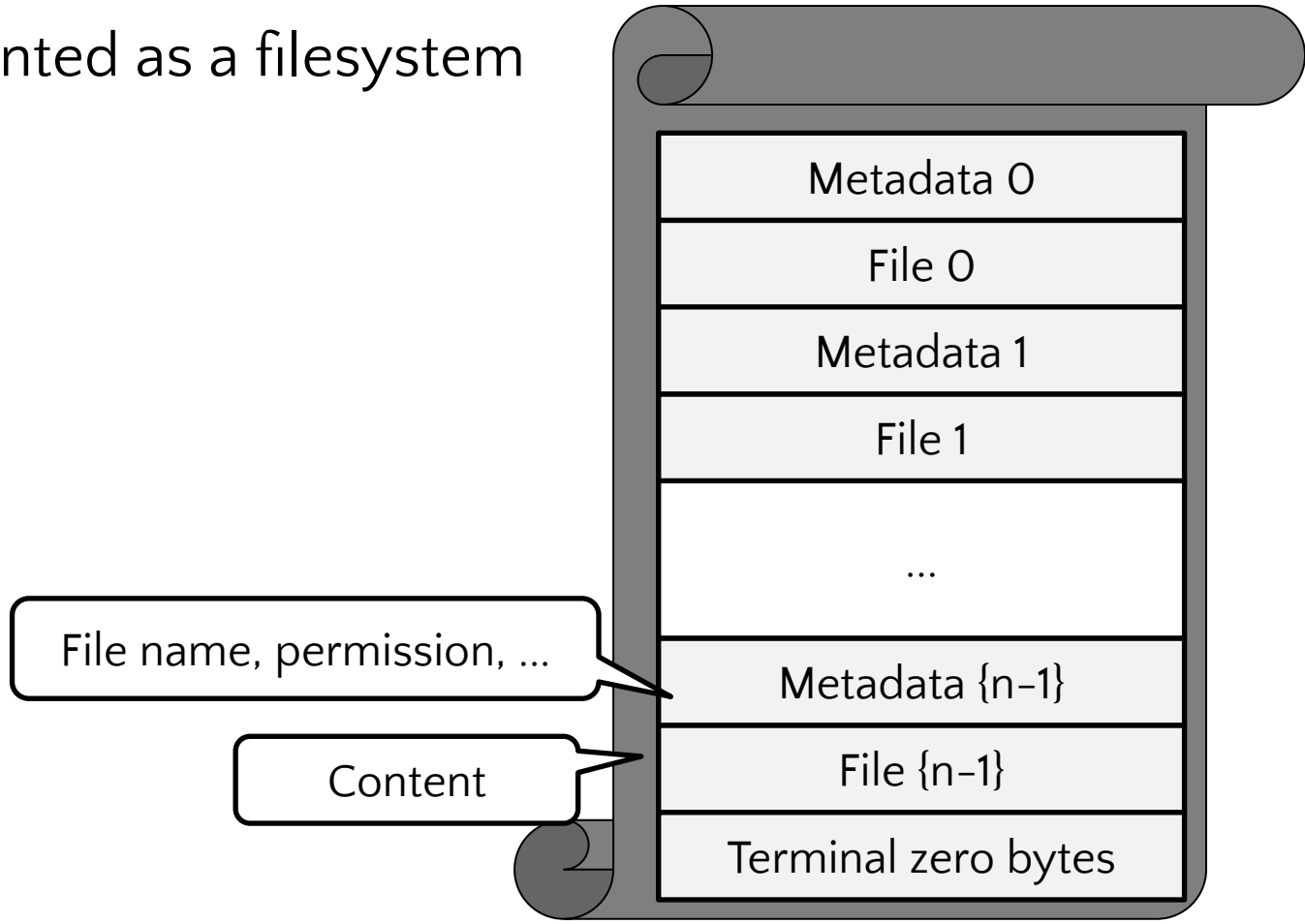
- Appeared in 1970s
- Originally designed for magnetic tapes
- **No random access**

https://en.wikipedia.org/wiki/PDP-11

Problem 1: Requires scanning the whole "tape"

- Without scanning the whole "tape“, file metadata cannot be listed up

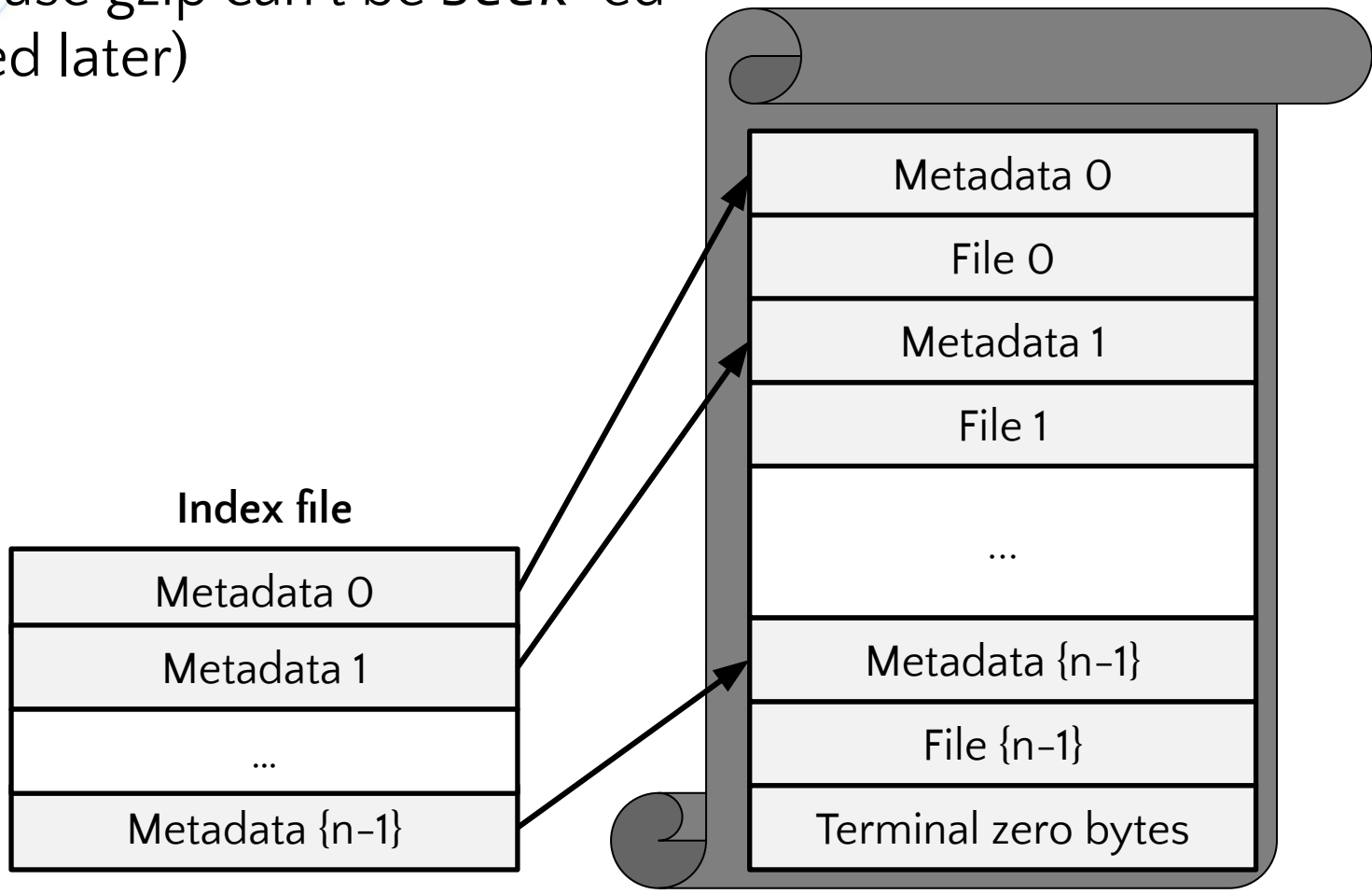
→ Can't be mounted as a filesystem



Problem 1: Requires scanning the whole "tape"

- Having an external index file can solve the problem?

→ No, because gzip can't be *seek*-ed (discussed later)



Problem 2: No deduplication

- A registry might contain very similar images
  - Different versions
  - Different architectures
  - Different configuration files

- Tar balls of these images are likely to waste the storage for identical/similar files

- But not a serious issue when you have enough budget for the cloud storage

Problems of Docker / OCI image format

1. Requires scanning the whole "tape"

2. No deduplication

The main focus towards lazy distribution

https://en.wikipedia.org/wiki/Magnetic_tape

Why do we want lazy distribution?

- *“pulling packages accounts for 76% of container start time, but only 6.4% of that data is read.”*
  - Harter, Tyler, et al. "Slacker: Fast Distribution with Lazy Docker Containers." FAST 2016

Expected use-cases

- **“dev stage” images of multi-stage Dockerfiles**
  - No need to consider tolerance against remote registry failures (because `RUN apt-get install` instructions are already flaky anyway)

```
FROM        example.com/heavy-dev-env:lazy AS dev
RUN         apt-get update && \
            apt-get install -y some-additional-libs
COPY        src .
RUN         ./configure && \
            make static && \
            cp bin/foo /foo

# the stage switches here

FROM        scratch
COPY        --from=dev /foo /foo
ENTRYPOINT  /foo
```

Expected use-cases

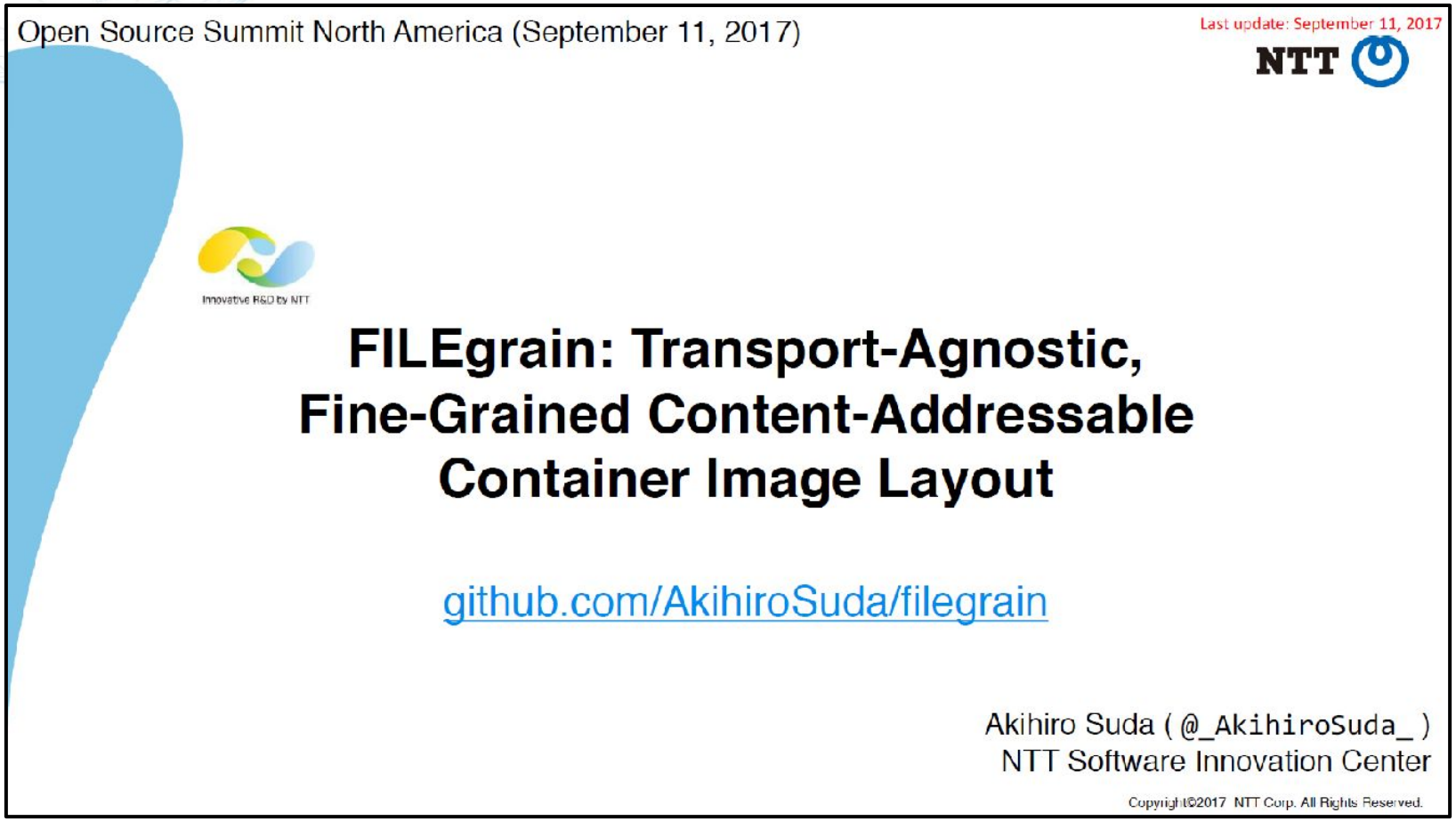
- Other use-cases are also valid, but **mind fault tolerance** (until the image gets 100% cached locally)
  - Kubernetes `readinessProbe`
- **FaaS**
- **Web apps** with huge number of HTML files and graphic files
- **Jupyter Notebooks** with big data samples included
- **Full GNOME/KDE desktop**
  - Will 2020 be the year of the *containerized* Linux desktop?

Our first attempt (2017)

Our first attempt (2017)
… and post-mortem

Our first attempt : FILEgrain (2017)

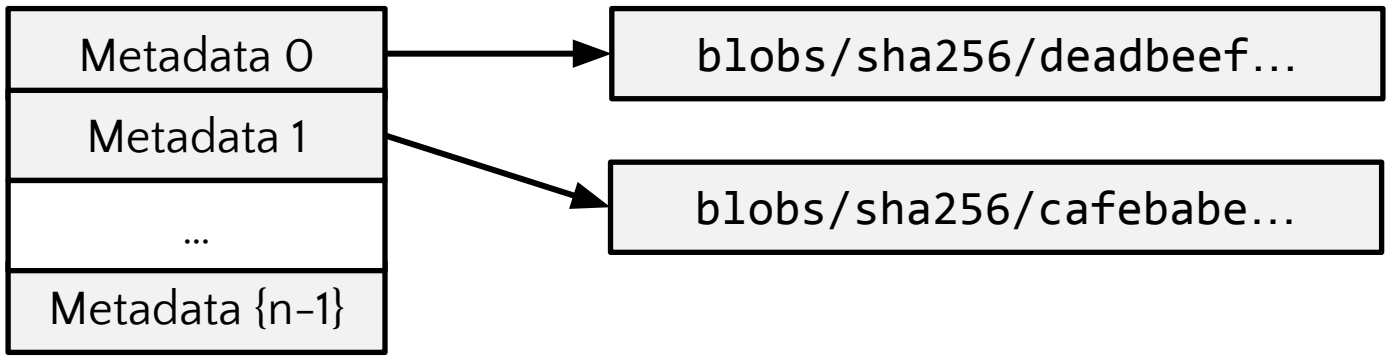
- No tar balls
- Composed of a protobuf index file (continuity manifest) + content-addressable blob files



Our first attempt : FILEgrain (2017)

- No tar balls
- Composed of a protobuf index file (continuity manifest) + content-addressable blob files

```
message Metadata {
    repeated string path;
    int64 uid;
    int64 gid;
    uint32 mode;
    uint64 size;
    repeated string sha256Digest;
    ...
}
```



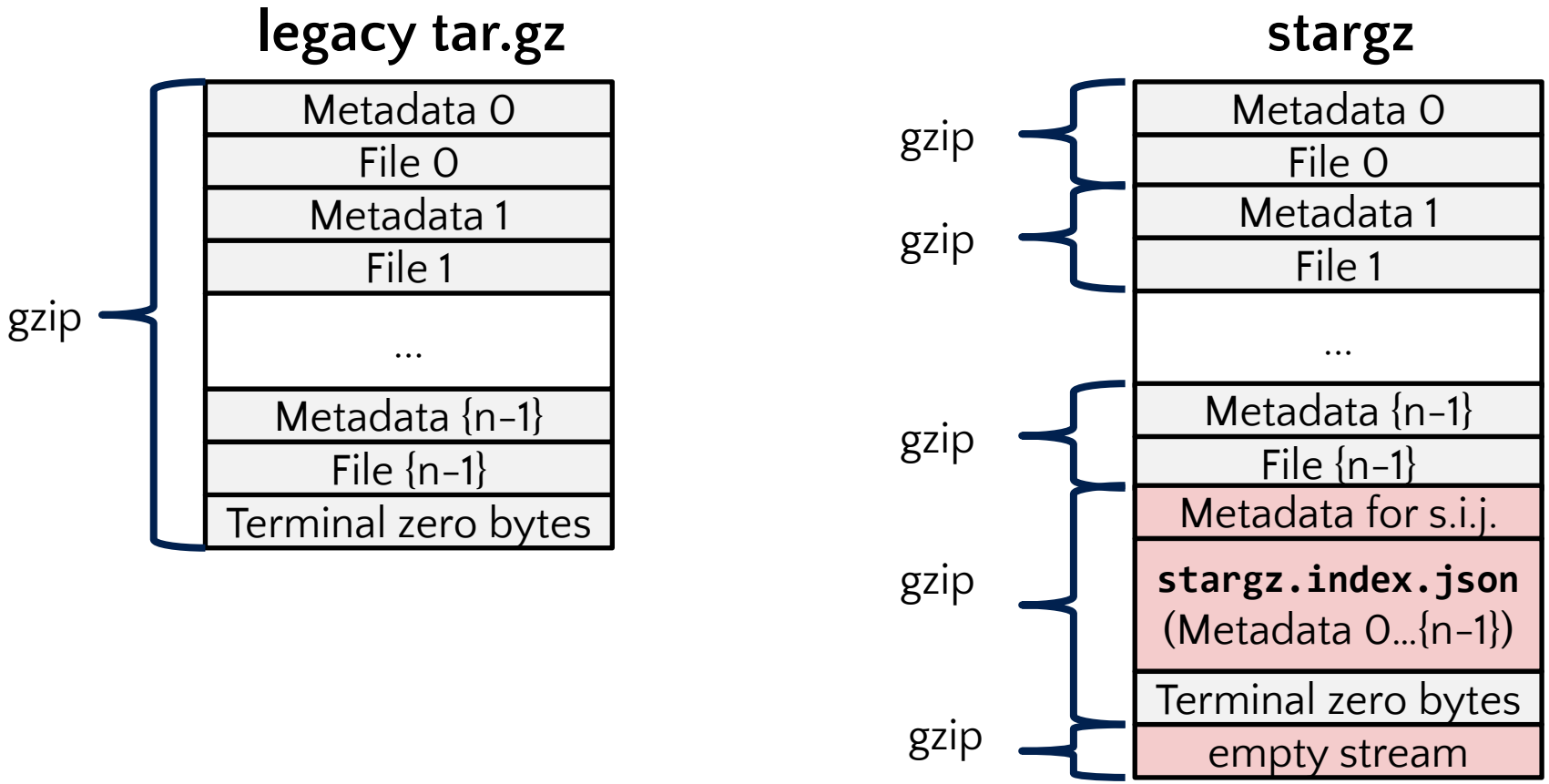
FILEgrain post-mortem

- Incompatibility with legacy tar balls
- Chicken-and-egg: hard to finalize the spec when no implementation exists; hard to promote implementation when the spec is not finalized
- Use-cases were unclear; didn't need to focus on deduplication
- Performance overhead due to huge numbers of HTTP requests for reading small files

The solution in 2020: stargz

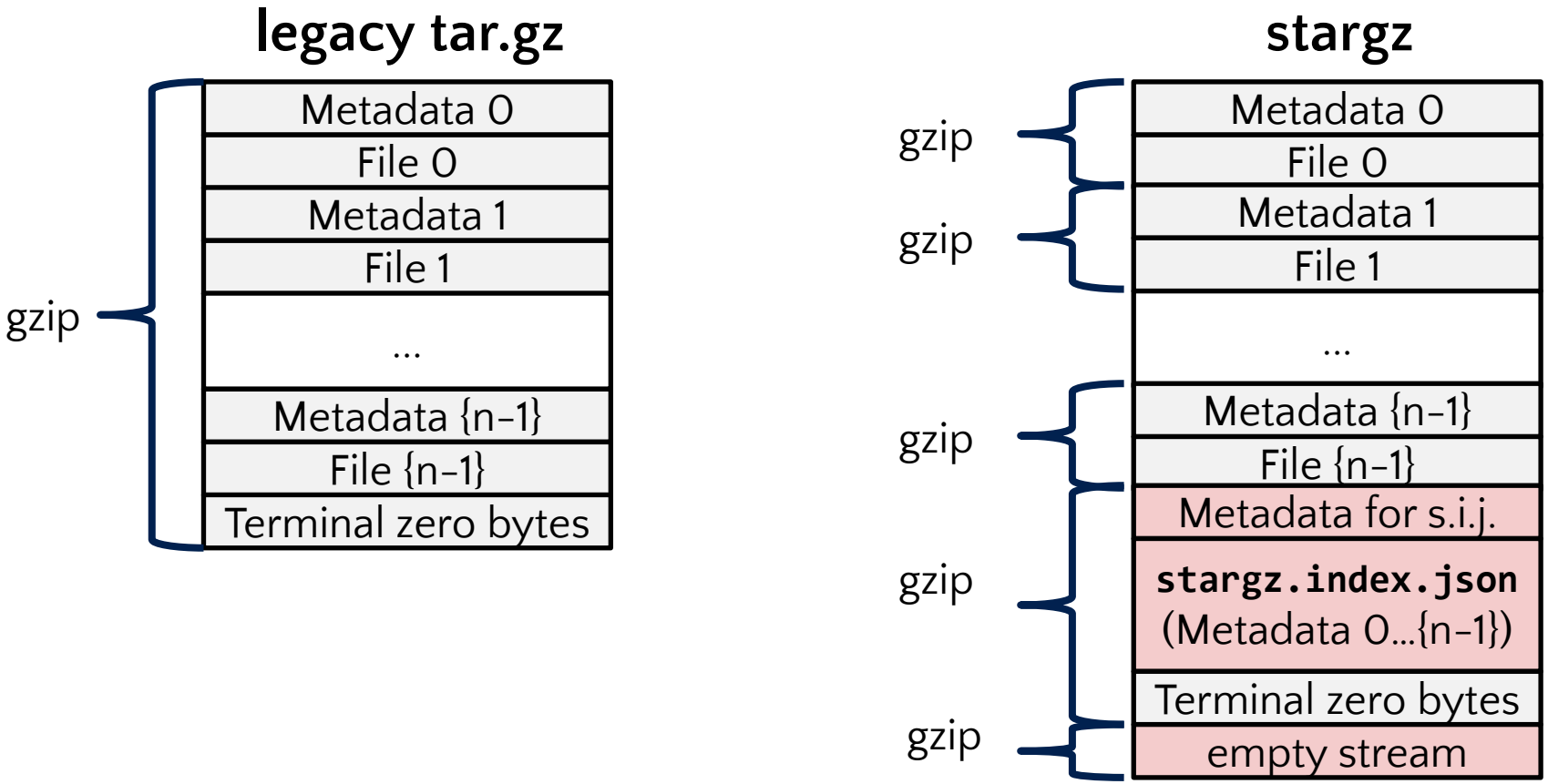
stargz: seekable tar.gz

- Proposed by **Brad Fitzpatrick (Google, at that time)** for accelerating the CI of the Go language project
- No focus on data deduplication



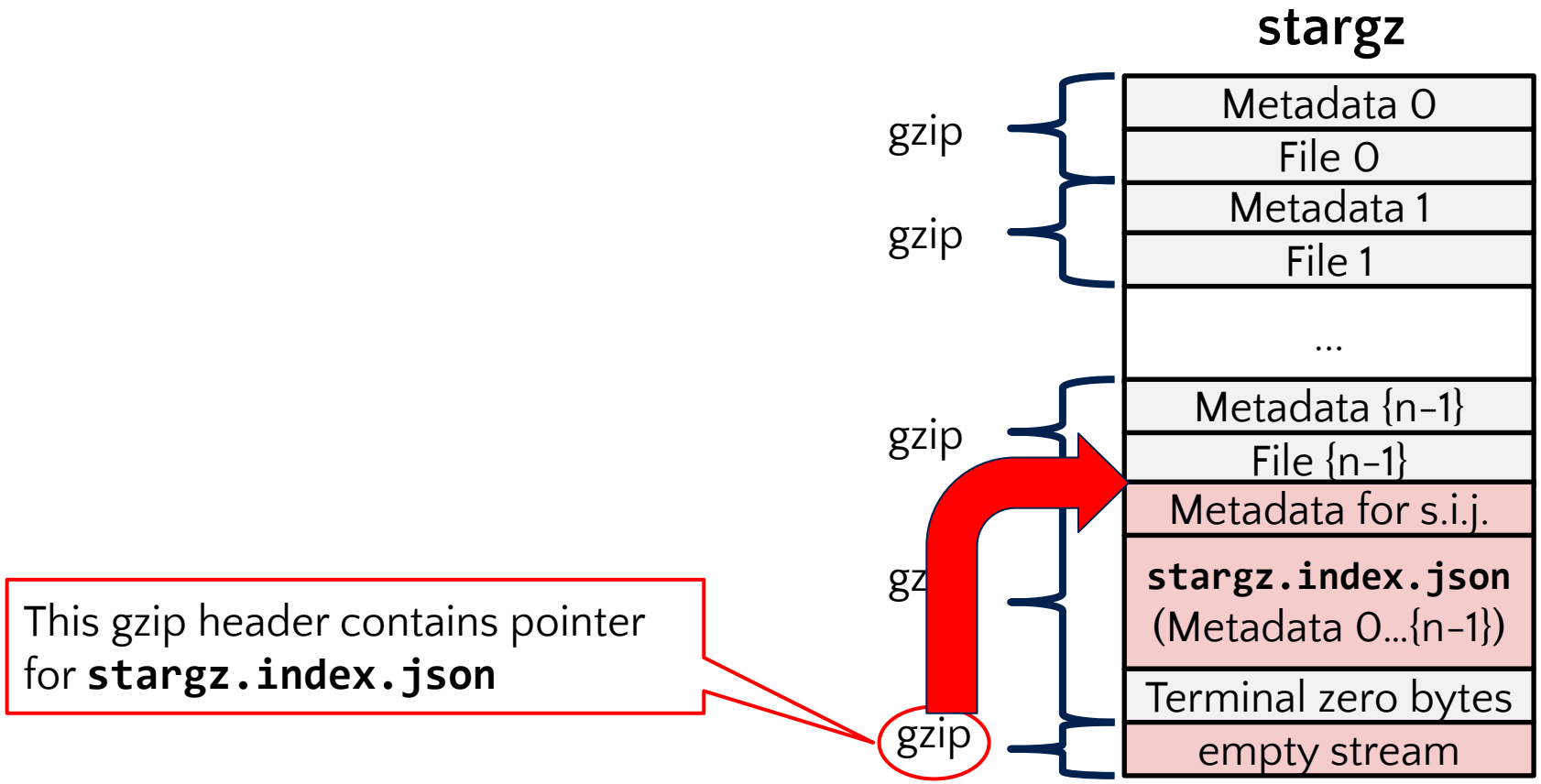
stargz: seekable tar.gz

- **Fully compatible** with legacy tar.gz
- But contains extra **“`stargz.index.json`”** entry



stargz: seekable tar.gz

- Only **`stargz.index.json`** is required for mounting the image
- Actual files in the archive can be fetched on demand (when HTTP Range Requests are supported)



stargz adoption in the ecosystem

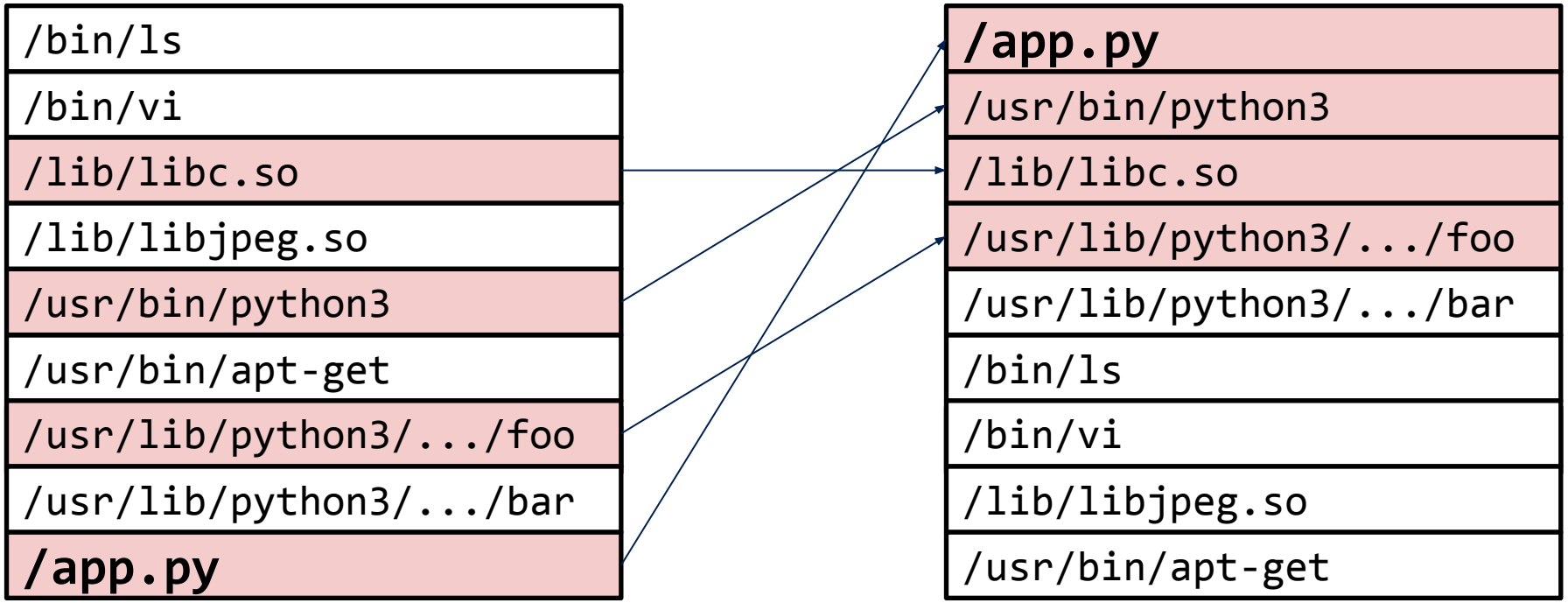
NTT

- **containerd**: https://github.com/ktock/stargz-snapshotter
  - By **Kohei Tokunaga (NTT)**
  - Implemented as a containerd snapshotter plugin
  - stargz archives are mounted as read-only FUSE filesystems
  - OverlayFS is used for supporting writing
  - **Supports more aggressive optimization** (discussed later)
- **Podman**: https://github.com/giuseppe/crfs-plugin
  - By **Giuseppe Scrivano (Red Hat)**
  - Implemented as a fuse-overlayfs plugin

stargz optimizer for containerd

- Profiles actual file access patterns by running an equivalent of `docker run`
  - Future: static analysis using `ldd`(-ish) ? Machine learning?

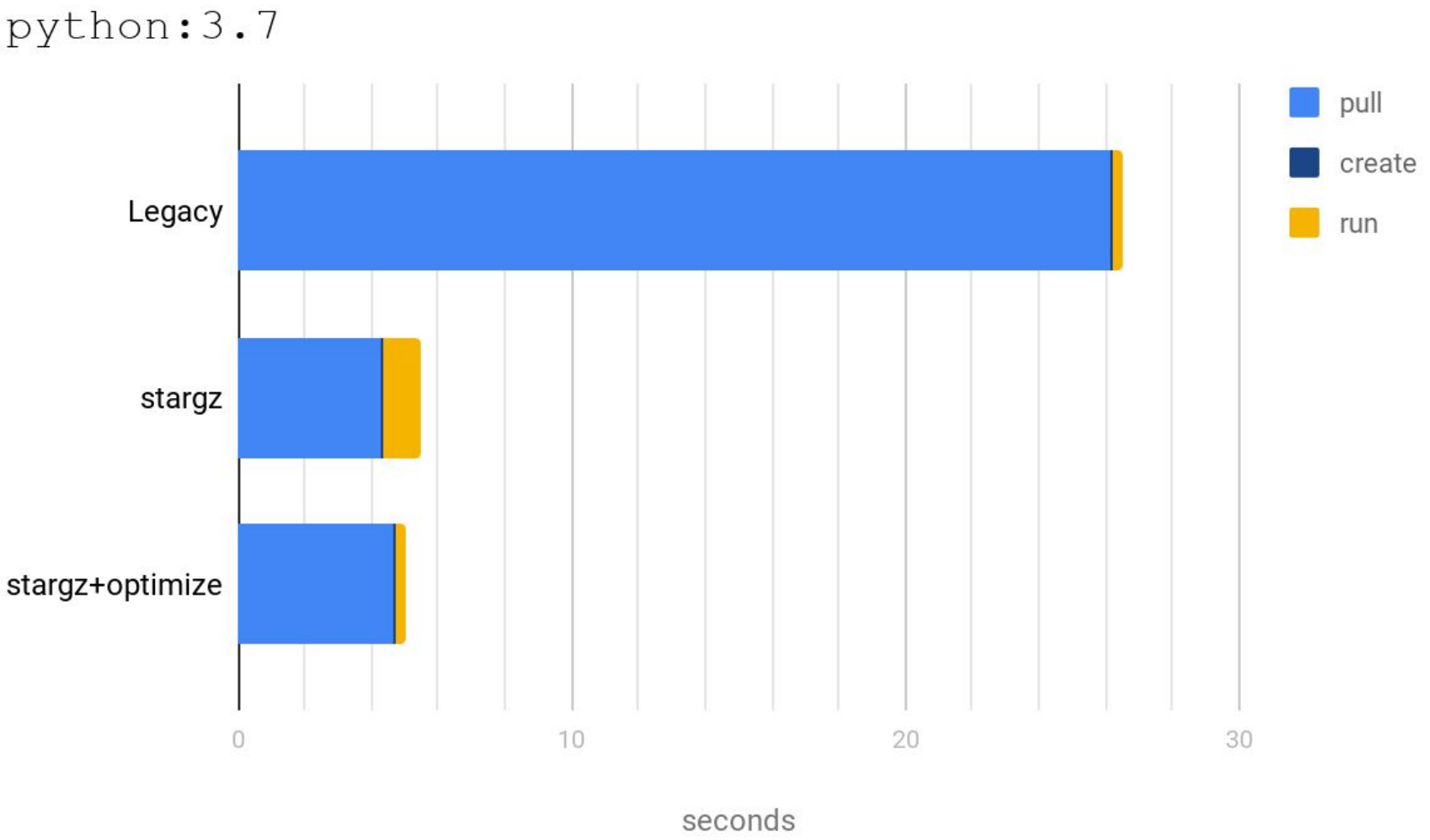
- Reorders file entries in the archive so that relevant files can be **prefetched in a single HTTP request**



Benchmark results

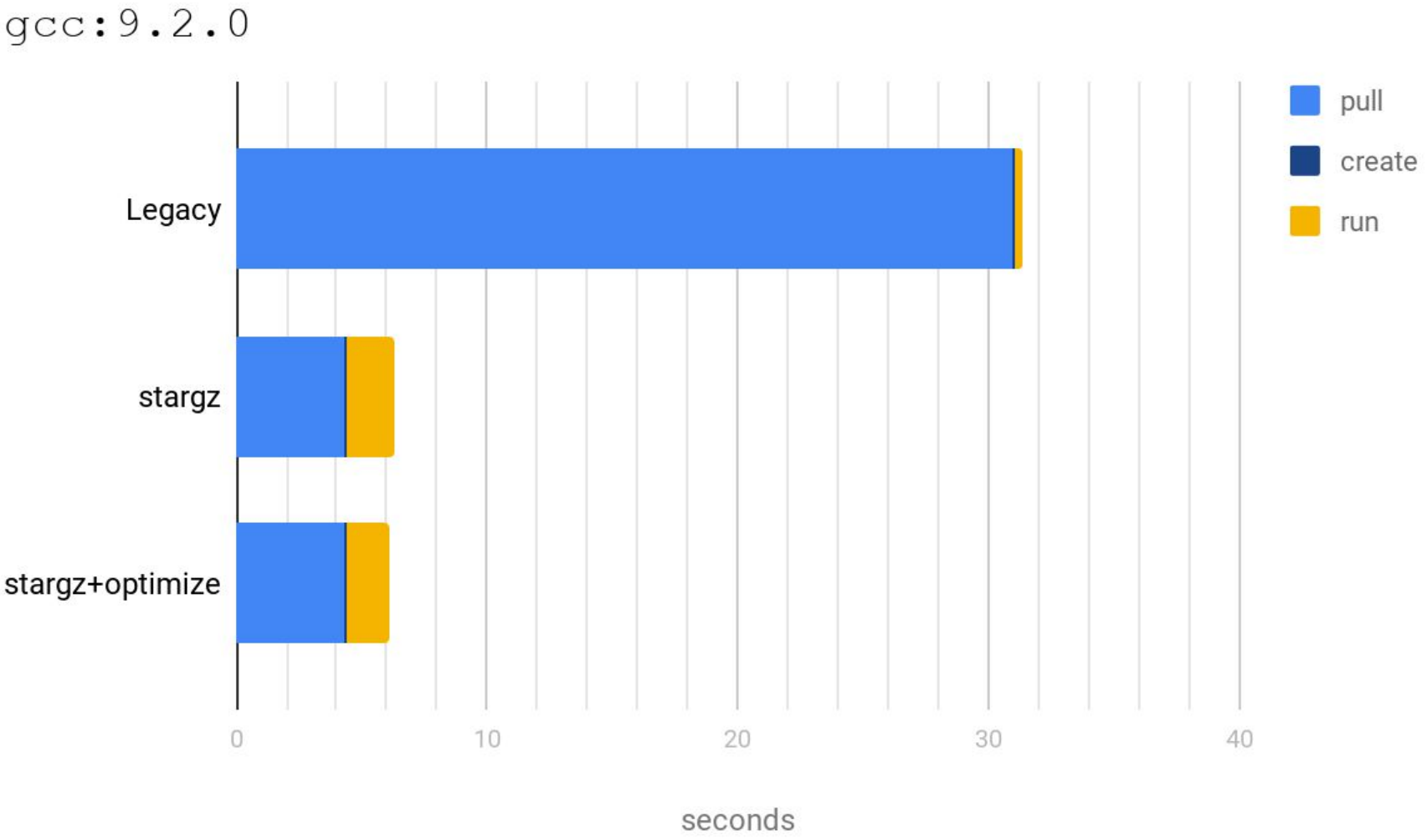
- **Registry**: Docker Hub (`docker.io`)
- **containerd host location**: EC2 Oregon
- **Benchmark**: execute typical base images with "compile hello world" command

Benchmark results



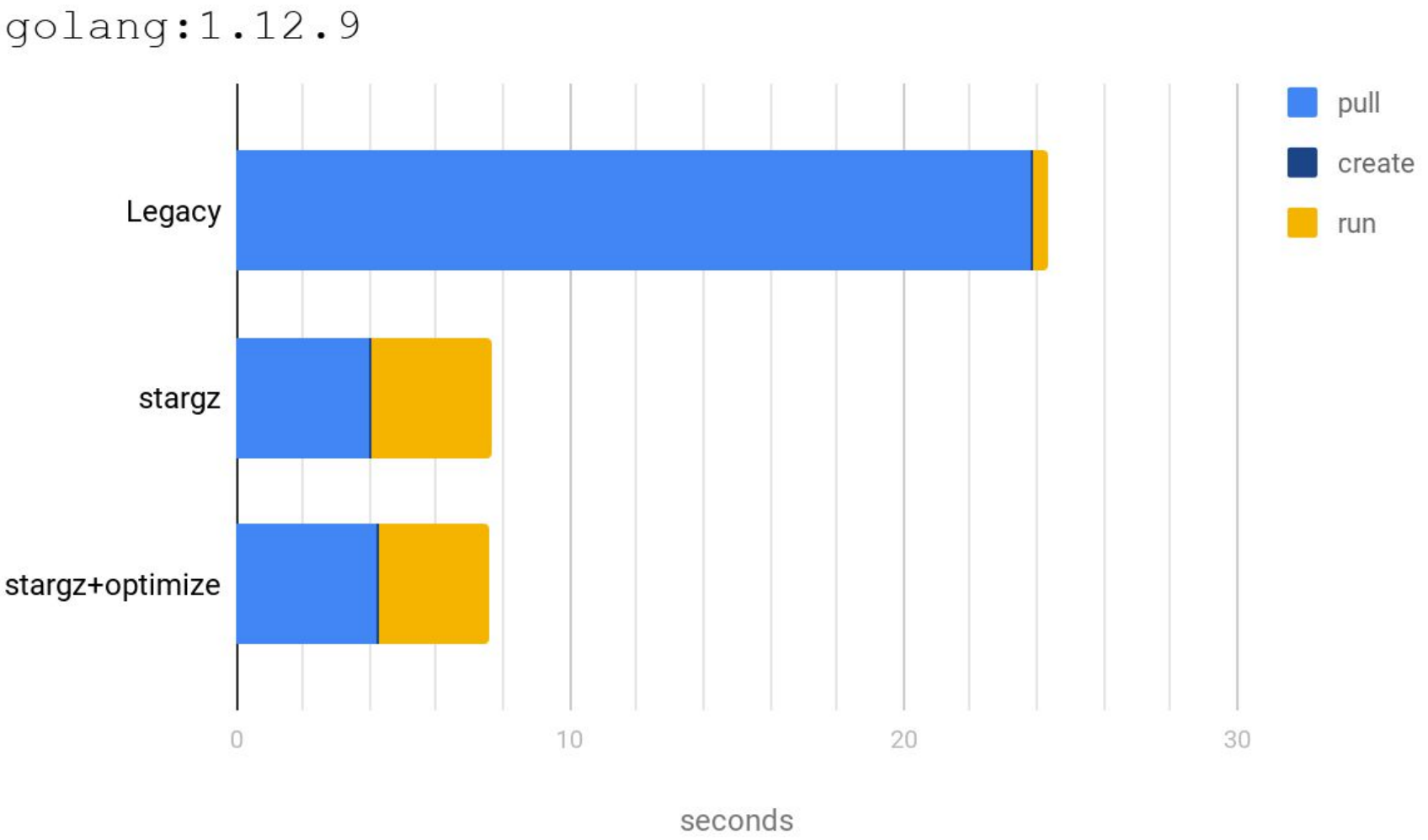
Credit to Kohei Tokunaga (NTT) for containerd impl. & benchmark scripts

Benchmark results



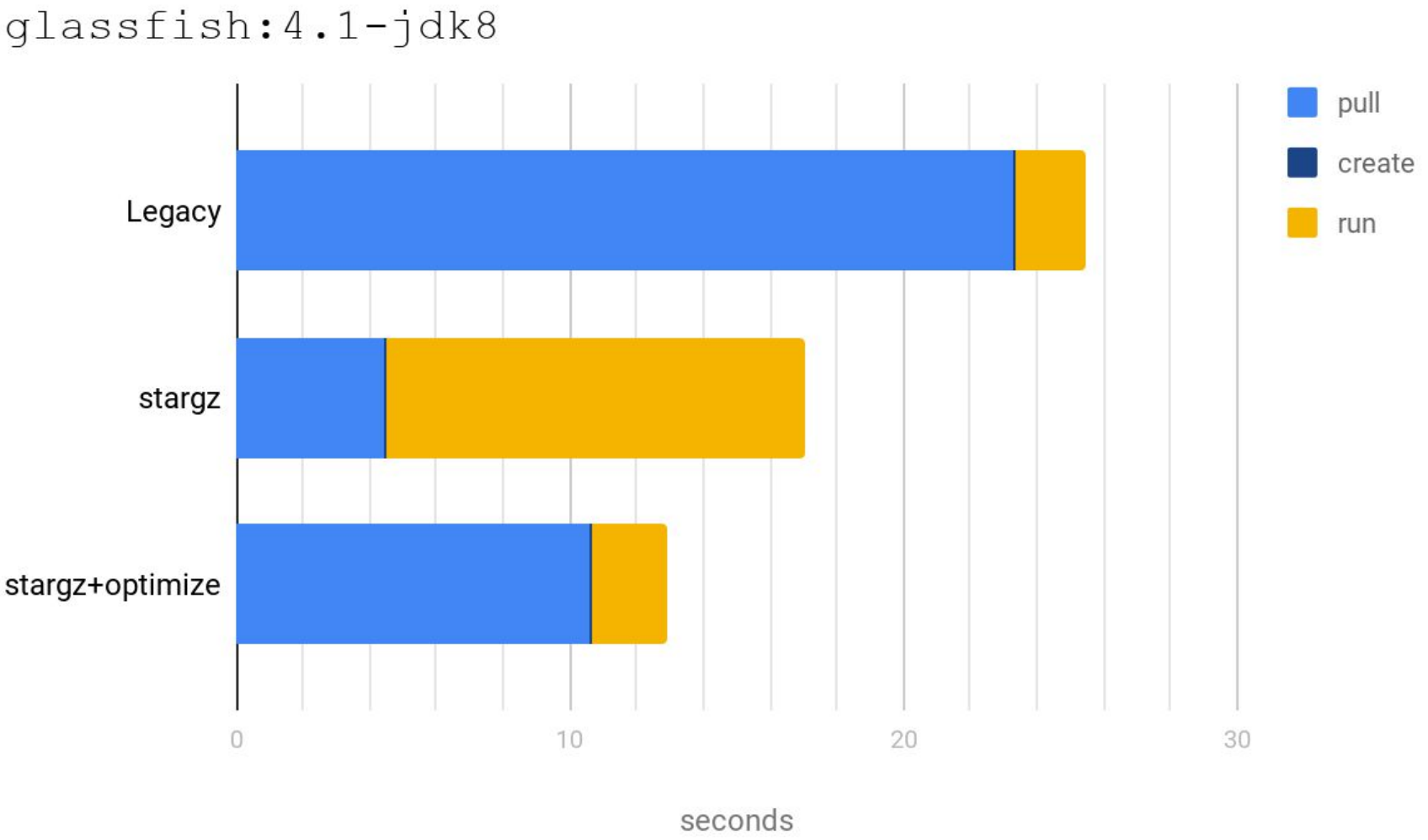
Credit to Kohei Tokunaga (NTT) for containerd impl. & benchmark scripts

Benchmark results



Credit to Kohei Tokunaga (NTT) for containerd impl. & benchmark scripts

Benchmark results



Credit to Kohei Tokunaga (NTT) for containerd impl. & benchmark scripts

More optimizations are to come

- **Impl:** Parallelize HTTP operations across image layers
  - https://github.com/ktock/stargz-snapshotter/issues/37

- **Spec**: Use zstd instead of gzip (“starzstd”?)
  - Proposed by Giuseppe https://github.com/golang/go/issues/30829#issuecomment-541532402
  - Suitable for images with many small files
  - Not compatible with OCI Image Spec v1.0.1
  - Compatible with OCI Image Spec v.Next

stargz integration for BuildKit

- **BuildKit**: modern OCI image builder
  - Concurrent execution
  - Efficient caching
  - Rootless
  - (pseudo-)daemonless
  - Clustering on Kubernetes
  - And a lot of innovative features

- stargz support is on our plan, stay tuned!
  - Producing stargz images
  - Consuming stargz images as base images

Other post-OCI formats

- **CernVM-FS**
  - Not compatible with OCI tar balls
  - Has been already widely deployed in CERN and their friends
  - Implementation available for containerd: https://github.com/ktock/remote-snapshotter/pull/27

- **Unofficial "OCI v2"**
  - Proposed by Aleksa Sarai (SUSE)
  - Not compatible with OCI v1 tarballs
  - Focuses on deduplication, using Restic algorithm
  - WIP implementation available for umoci (image manipulation tool): https://github.com/openSUSE/umoci/tree/experimental/ociv2
  - No runtime implementation seems to exist

Other post-OCI formats

- **IPCS**
  - Proposed by Edgar Lee (Netflix)
  - Built on IPFS (P2P CAS) protocol
  - Not compatible with OCI tar balls
  - Implementation available for containerd: https://github.com/hinshun/ipcs

- **Azure Container Registry "Project Teleport"**
  - Built on SMB protocol and VHD images
  - Not FLOSS

Recap

- Lots of alternative image formats are proposed for lazy distribution, but compatibility matters

- **stargz** is getting wide adoption (containerd & Podman)

- containerd supports sort+prefetch optimization for stargz
  https://github.com/ktock/stargz-snapshotter

Request for comments

- Valid & invalid use cases?
- More efficient optimization techniques?
- Issues/PRs are welcome at
  https://github.com/ktock/stargz-snapshotter
  (Expected to be moved under github.com/containerd soon)