Remote CMake Builds

For Safer C++ builds

Tipi.build @ Zürich C++ Meetup

tipi.build team

Damien
Co-founder

Yannic
Co-founder

Luc
Developer

Valentina
CCO

Stefan
Developer

We are Hiring ! Talk to us if you like C++, ASTs and build systems !

C++ cares about Safety

- **Keynote: All the Safeties**, *Sean Parent* - C++ on Sea 2023 : youtu.be/BaUv9sgLCP
- **C++ and Safety**, *Timur Doumler* - C++ on Sea 2023 : youtu.be/imtpoc9jtOE

- **Safety and Security: The Future of C++**, *JF Bastien* - CppNow 2023 : youtu.be/Gh79wcGJdTg

What is Safety ?

In Computer Science

- Type Safety
- Memory Safety
    - Lifetime safety
    - Bounds safety
    - Initialization safety
- Object access safety

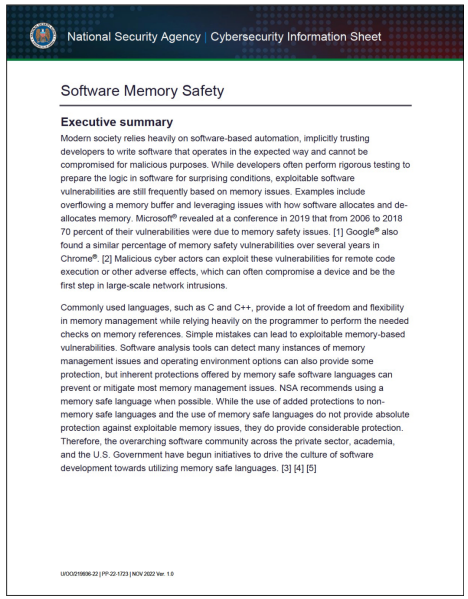
- Thread Safety
- Arithmetic safety
- Definition safety

Software Supply Chain Safety ?

Reproducibility, Traceability

Why Safety ?

Because Cybersecurity



National Security Agency | Cybersecurity Information Sheet

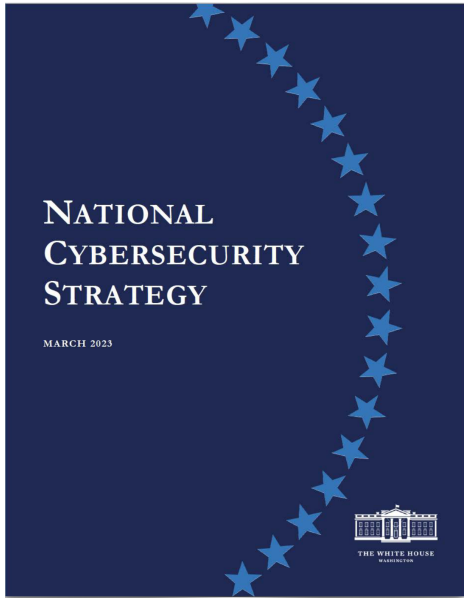
Software Memory Safety

Executive summary

Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues. Examples include overflowing a memory buffer and leveraging issues with how software allocates and de-allocates memory. Microsoft® revealed at a conference in 2019 that from 2006 to 2018 70 percent of their vulnerabilities were due to memory safety issues. [1] Google® also found a similar percentage of memory safety vulnerabilities over several years in Chrome®. [2] Malicious cyber actors can exploit these vulnerabilities for remote code execution or other adverse effects, which can often compromise a device and be the first step in large-scale network intrusions.

Commonly used languages, such as C and C++, provide a lot of freedom and flexibility in memory management while relying heavily on the programmer to perform the needed checks on memory references. Simple mistakes can lead to exploitable memory-based vulnerabilities. Software analysis tools can detect many instances of memory management issues and operating environment options can also provide some protection, but inherent protections offered by memory safe software languages can prevent or mitigate most memory management issues. NSA recommends using a memory safe language when possible. While the use of added protections to non-memory safe languages and the use of memory safe languages do not provide absolute protection against exploitable memory issues, they do provide considerable protection. Therefore, the overarching software community across the private sector, academia, and the U.S. Government have begun initiatives to drive the culture of software development towards utilizing memory safe languages. [3] [4] [5]

U/OO/219936-22 | PP-22-1723 | NOV 2022 Ver. 1.0



NATIONAL CYBERSECURITY STRATEGY

MARCH 2023

THE WHITE HOUSE
WASHINGTON

EUROPEAN COMMISSION

Brussels, 15.9.2022
COM(2022) 454 final

2022/0272 (COD)

Proposal for a

REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL

on horizontal cybersecurity requirements for products with digital elements and amending Regulation (EU) 2019/1020

(Text with EEA relevance)

{SEC(2022) 321 final} - {SWD(2022) 282 final} - {SWD(2022) 283 final}

EN **EN**

Why Cybersecurity ?

Because Functional Safety

How to achieve safety in C++ ?

Achieving safety in C++ today

with CMake

Focus for today

- Memory Safety
- Arithmetic Safety
- Software Supply Chain Safety

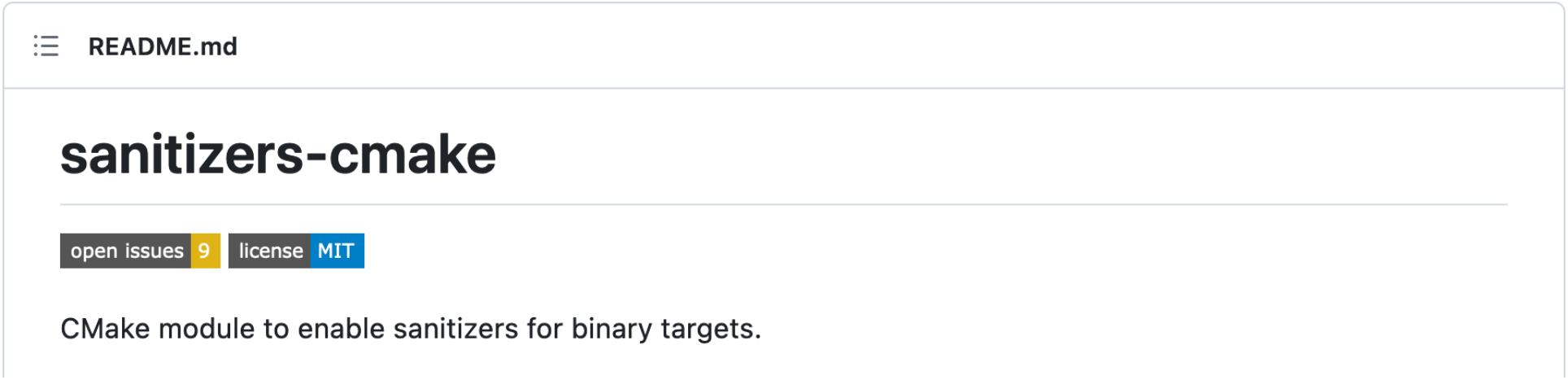
Focus for today

- Memory Safety
- Arithmetic Safety
- Software Supply Chain Safety

Yes CMake can help you with that !

Sanitizers for C++ builds with CMake

github.com/arsenm/sanitizers-cmake



Sanitizers for C++ builds with

CMake

- AddressSanitizer (ASan)
- UndefinedBehaviorSanitizer (UBSsan)
- ThreadSanitizer (TSan)
- LeakSanitizer (LSan)
- MemorySanitizer (MSan)

Using LLVM Sanitizers in your CMake

```
project(safepp)

find_package(Sanitizers MODULE REQUIRED)

add_executable(use-after-free test/use-after-free.cpp)
```

```
# enable sanitizers
add_sanitizers(use-after-free)
```

So we done ?

ASan, TSan, UBSan are incompatible with each others

Separate builds and test runs

Separate CMake toolchain files

sanitize-ub.cmake :

```
set(SANITIZE_UNDEFINED TRUE CACHE BOOL "" FORCE)
```

sanitize-address.cmake :

```
set(SANITIZE_ADDRESS TRUE CACHE BOOL "" FORCE)
```

sanitize-memory.cmake :

```
set(SANITIZE_MEMORY TRUE CACHE BOOL "" FORCE)
```

sanitize-thread.cmake :

```
set(SANITIZE_THREAD TRUE CACHE BOOL "" FORCE)
```

Memory Safety with Asan

Arithmetic Safety with UBSan

Software Supply Chain Safety

Software Supply Chain Safety

Reproducible Builds

- Deterministic Builds
- Hermetic Builds
- Traceable Build Caching

Deterministic Builds

Problems

- File order feeding to the compiler + linker
- __DATE__ and __TIME__ macros
- Compiler randomness when using -flto
- __FILE__ macro and path prefix in debug symbols

__DATE__ and __TIME__ macros

Ensure it's constant

MSVC :

```
add_link_options("/Brepro")
```

GCC, Clang in **environment** :

```
# GCC
SOURCE_DATE_EPOCH=628727383
# Clang
ZERO_AR_DATE=0
```

Compiler randomness when using -flto

Only required for GCC

```
set(APP_SOURCES
    app.cpp
    lib.cpp)

foreach(_file ${APP_SOURCES})
    file(SHA1 ${_file} sha1sum)
    string(SUBSTRING ${sha1sum} 0 8 sha1sum)
    set_property(SOURCE ${_file} APPEND_STRING PROPERTY COMPILE_FLAGS "-frandom-seed=0x${sha1sum}")
endforeach()
```

__FILE__ macro and path prefix

- MSVC: no options
- Clang: -fdebug-prefix-map but no solution for __FILE__
- GCC: -ffile-prefix-map=OLD=NEW

__FILE__ macro and path prefix

- Just build in the same environment
- At the same path

Reproducible and Traceable builds

Or let just teach Ninja to use Git

Let's merge Git and Ninja !

Because we can use Git :

- Content Addressable Filesystem ✅
- to clone sources and build trees in invariant paths ✅

- to **track changes** ✅

We just need to add binary cache packs connected to Git

comprehensive cache packs

- Store and deduplicate binary trees
- Track each build snapshot by Git Commit
- Determine the provenance of a built object by content

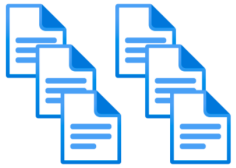
Git + CMake + cache packs + ☁️

builds ✅

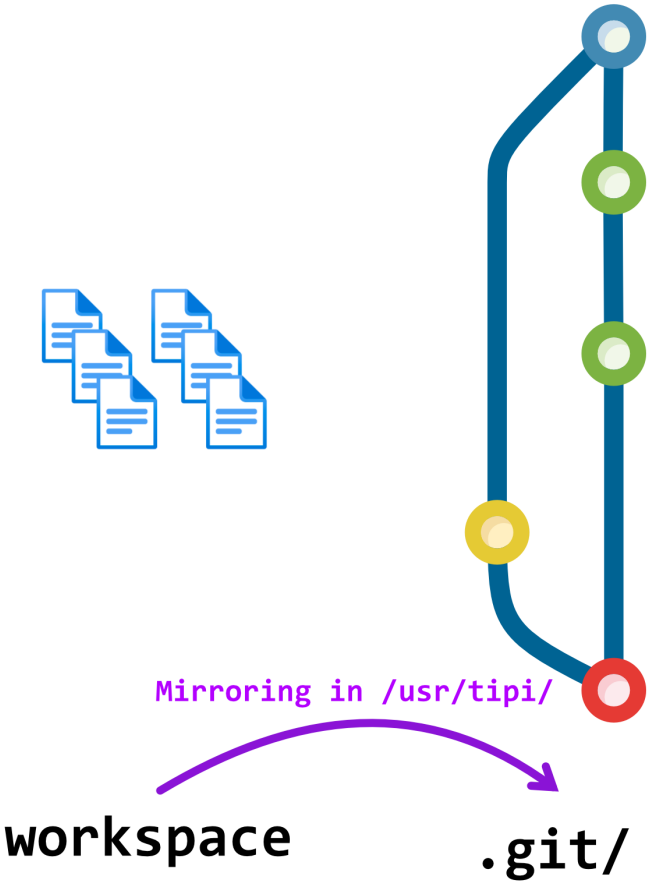
Git + Ninja + cache packs + ☁️ builds ✅

__FILE__ problem solved 👊

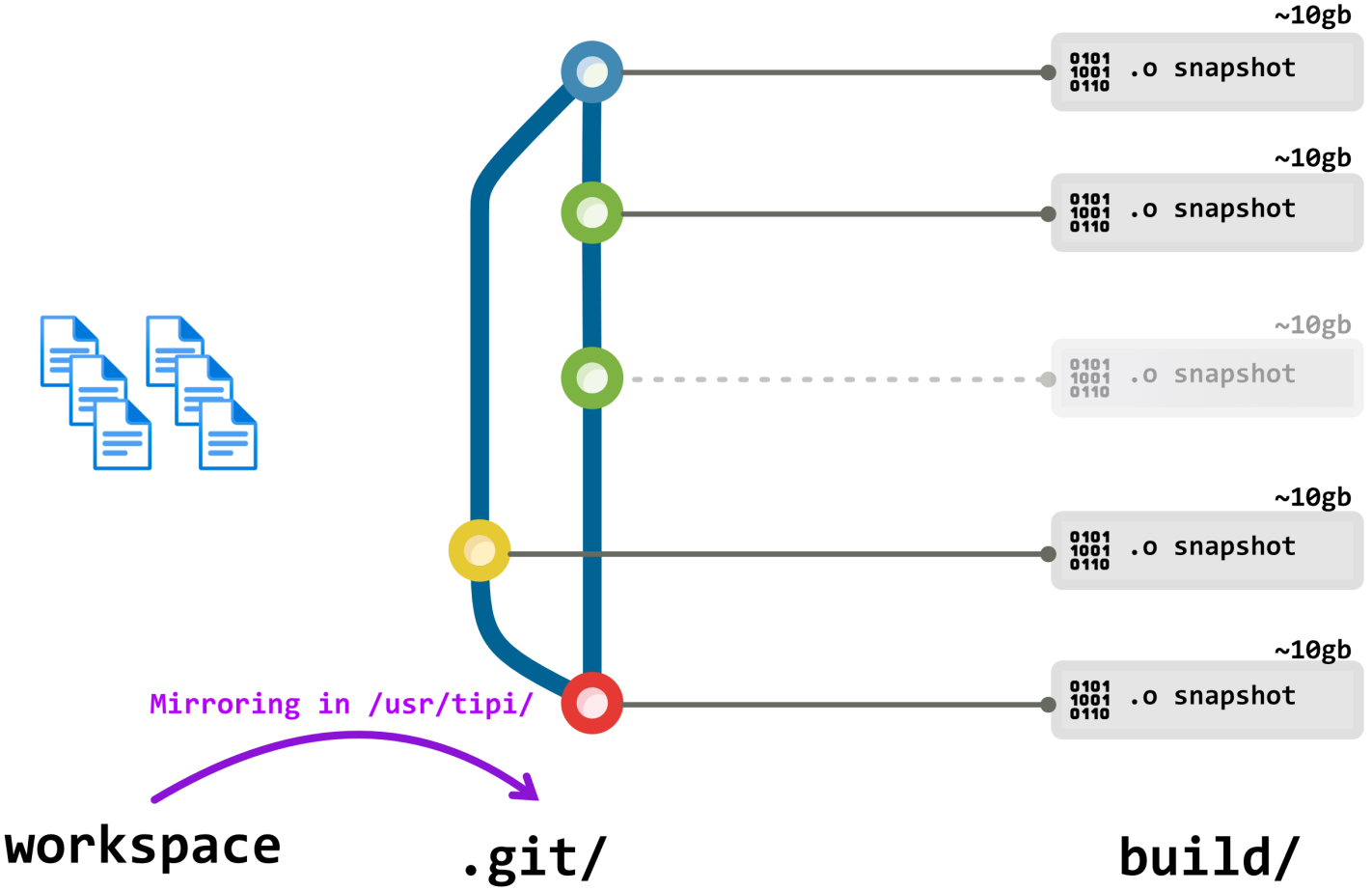
Stores code in an invariant path

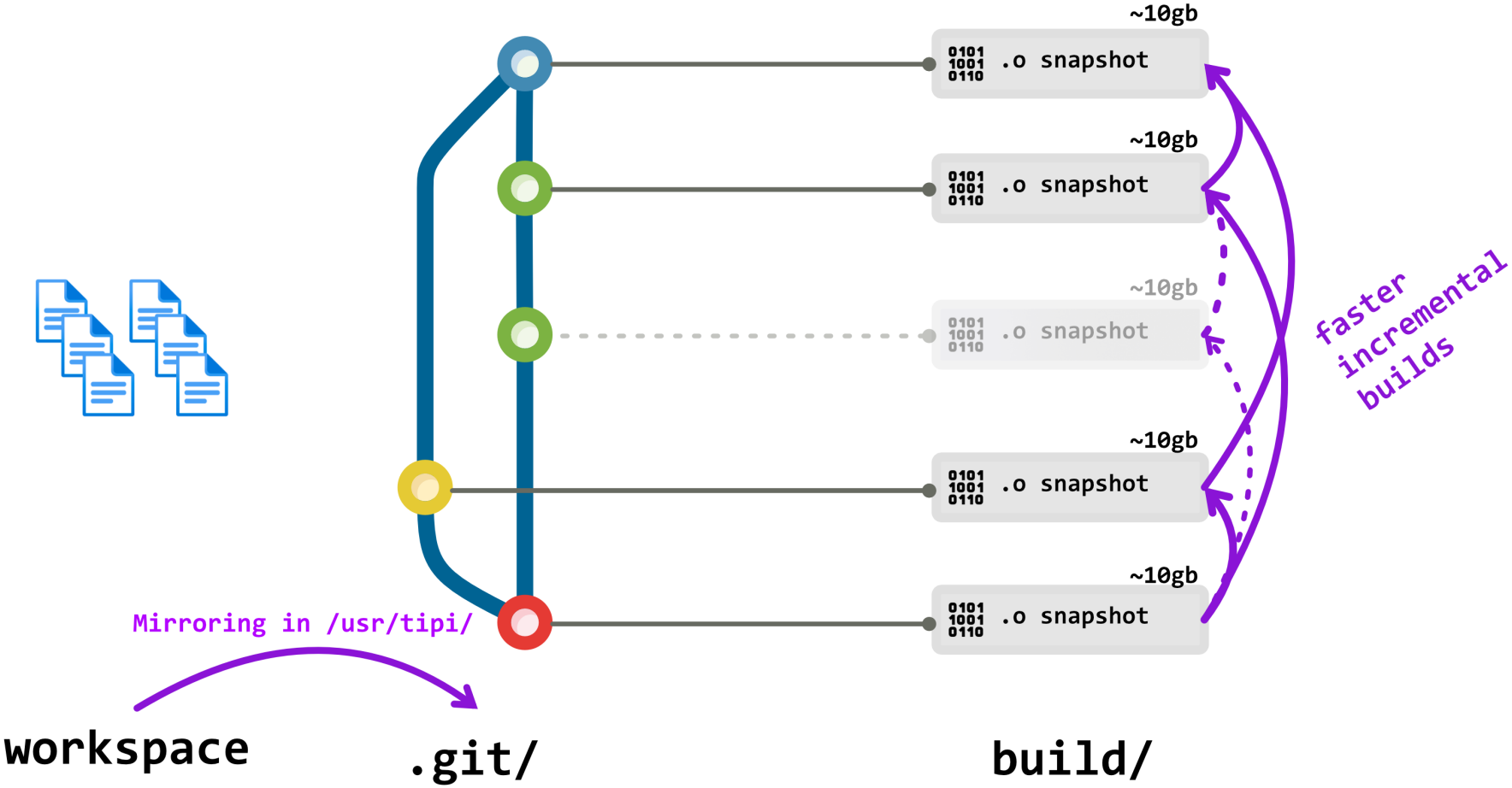


workspace

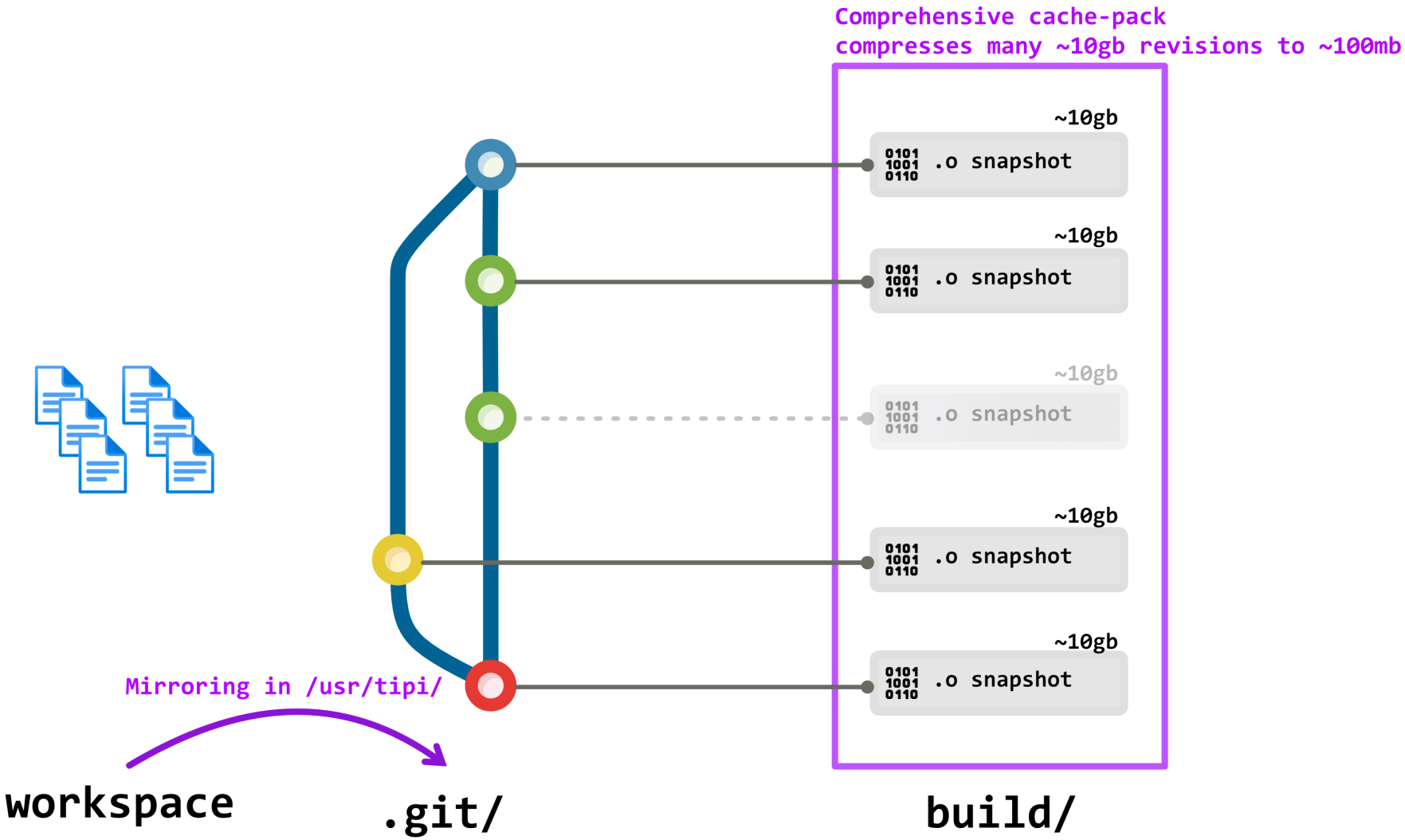


Cache id based on the repository origin


~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
Mirroring in /usr/tipi/
workspace
.git/
build/



Comprehensive packs


Comprehensive cache-pack
compresses many ~10gb revisions to ~100mb
~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
~10gb
0101 1001 0110 .o snapshot
Mirroring in /usr/tipi/
workspace
.git/
build/

tipi.build build cache

Immediate builds from source

- Comprehensive cache packs
  - Contains all branch tips, tags and recent revisions
  - Repacked on each new build
- Guaranteed reproducibility & isolation
  - .git/ mirroring
  - VM or Container environments

Safety checks immediate feedback

Remote CMake Builds

UBSan, ASan, TSan

at your fingertips

Full build isolation and hermeticity

Remote CMake Builds

The Tipi team thanks you for your attention and awaits you on www.tipi.build feel free to grab our **C++ Stickers** !

Let's share some 🍕 and 🍻!

Open source

Community support

Impress

Data Privacy

General Terms & Conditions

tipi technologies Ltd.
Josefstrasse 219, 8005 Zürich, Switzerland
UID: CHE-358.978.640