SWHID

Introduction and use cases

Agustín Benito Bethencourt

Toscalix

April 2026

Agenda

- About the speaker
- Intrinsic identifier
- SWHID description
- Use cases
- SWHID and traceability
- Takeaways

About the speaker

Agustín Benito Bethencourt

@toscalix

http://www.toscalix.com

- SwH Ambassador since 2023
- Freelance and independent consultant helping companies to increase organizational performance by:
  - Becoming good open source citizens
  - Applying BI to promote data-supported decision-making and continuous improvement processes
- Involvement:
  - Current: KDE eV, KDE España, STF, SwH and SPDX
  - Past: Eclipse Foundation, Linux Foundation (CIP, AGL...), Linaro, SUSE/openSUSE, MBition/Mercedes-Benz, ASOLIF, COVESA, Codethink, entrepreneur...
- Blog - About - Talks - Contact

Blog post series about SWHID

I am writing a blog post series about SWHID:

- Article 1: intro to intrinsic identifiers and SWHID
- Article 2: about SWHID syntax and comparing software artifacts with SWHID
- Article 3: SWHID is an open standard under open governance model
- Article 4: differences and similarities of SWHID and pURL
- Article 5: summary of the use cases provided on this slide deck

Check my blog: http://www.toscalix.com

Intrinsic Identifier

An **intrinsic identifier** is a unique marker derived directly from the inherent, natural properties of an entity. It is something the object *is* or *possesses* by its very nature.

Because it is inseparable from the entity, an intrinsic identifier exists whether or not an external system is there to record it.

Intrinsic ID definition

Examples

- Elements of the chemical table
- DNA
- Grow rings of a tree
- Iris pattern
- checksum hash / Omniborld

Intrinsic vs extrinsic ID

An extrinsic identifier is a unique label or code **assigned** to an entity by an **external authority or system**. Unlike an intrinsic identifier, it is not a natural or physical property of the object itself; rather, it is an "artificial" tag used to track, categorize, or manage the entity within a specific context or database.

Examples:

- IP addresses
- ISBN and Barcodes
- DOI, UUID, OID, GUID...
- pURL and SWID (not SWHID)
- CPE and CVE

Intrinsic ID classification

Intrinsic identifiers can be classified based on different criteria. Some are relevant in our context:

- Their physical or digital nature
- Their “persistence” or “stability” over time
- Intrinsic characteristics the id is based upon
- Standardised or not

SWHID

Description

SWHID: definition

SWHID definition

SWHID (SoftWare Hash IDentifier) is a **digital**, **content-based**, **persistent**, and **standardised** intrinsic identifier for software artifacts

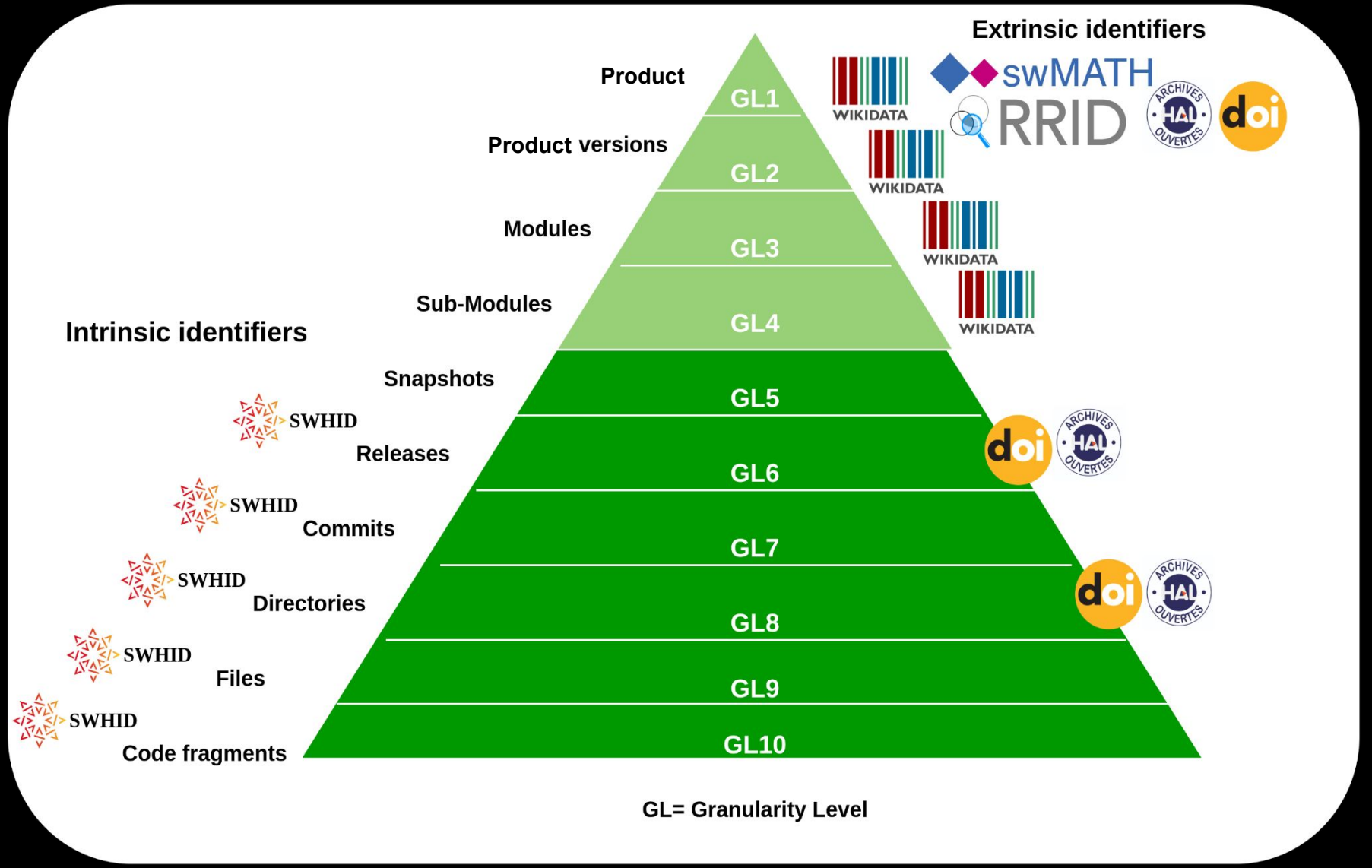
Why SWHID?

These are, at high level, some of the SWHID benefits:

- Cryptographic **integrity**
- **Decentralized** by design
- **Comprehensive**
- Designed for **provenance**
- **Fast** execution

Valid for any software artifact

Software artifacts/objects SWHID is designed for



Syntax: introduction

A SWHID consists of two separate parts:

1. a mandatory *core_identifier* that can identify any software artifact
2. an optional list of *qualifiers* that allows specification of the context where…
   a. the object is meant to be seen
   b. points to a subpart of the object itself.

<identifier> ::= <core_identifier> [<qualifiers>]

Syntax: core_identifier

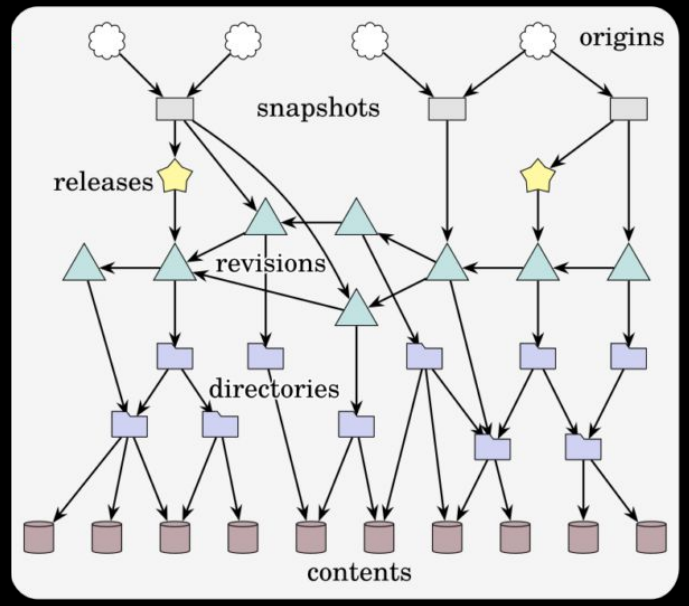
A core identifier is composed of four fields, each separated by a colon ":":

1. Type of the identifier (**prefix**)
   a. It is defined to be **swh**
2. Version of the identifier scheme (**scheme_version**)
   a. For this version of the specification, it is defined to be 1
3. A tag corresponding to the type of object identified **(object_type)**:
   a. cnt for contents (source code files, blobs…)
   b. dir for directories
   c. rev for revisions (commits)
   d. rel for releases (tags)
   e. snp for snapshots
4. The intrinsic identifier of the object (**object_id**)
   a. This is a hex-encoded (using lowercase ASCII characters) hash value computed from the content and relevant metadata of the object.

Merkle DAG

The computation of the SWHID object_id is based on Merkle DAG:

- Merkle data tree structure
- **D**irected
- **A**cyclic
- **G**raph



A qualified SWHID is composed of a core SWHID identifier, and a sequence of **qualifiers**:

- **fragment qualifiers** identify subparts of a software artifact (snippet).
  a. lines qualifier designates a line range inside a content
  b. bytes allows designation of a byte range inside a content
- **context qualifiers** provide additional context on the software artifact.
  a. origin allows declaration of the *software origin* where the object has been found or observed, as a URL
  b. visit adds the SWHID of the the *snapshot* of the repository where the object has been found or observed.
  c. path declares the *absolute file path*, from the *root directory* associated to the *anchor node*, to the object designated by the *core_identifier*.
  d. anchor identifies a node in the Merkle DAG relative to which a *path to the object* is specified, as the *core_identifier* (except a cnt). Used together with **path**

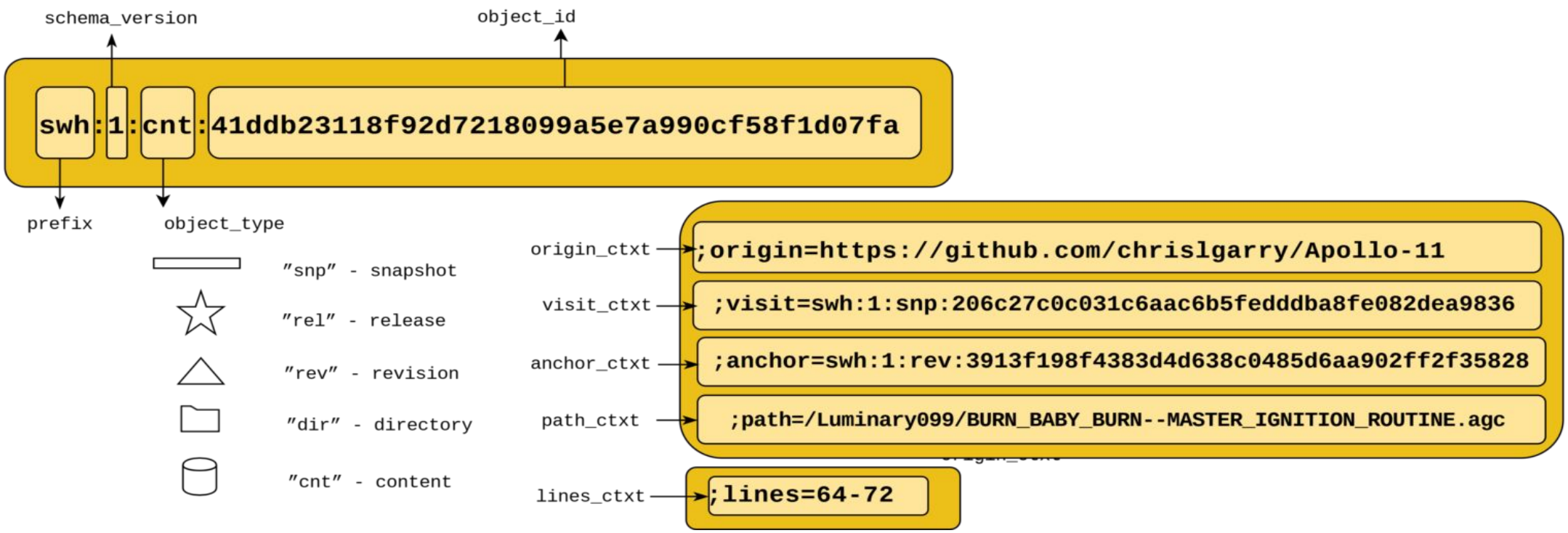
Syntax: qualifiers

Some rules apply to qualifiers:

- Each qualifier is specified as a key-value pair, using an **=** character as a separator.
- Qualifiers are separated from the core identifier and from each other by using a semicolon “**;**”.
- Some qualifliers only apply to specific content_type
  - path is invalid for *cnt*
  - Fragment qualifiers are valid for cnt *object_type* only.
- There are restrictions on the validity of some qualifiers
  - Any qualifier shall appear at most once
  - The validity of some qualifiers depends on the presence of other qualifiers.
    - visit is only valid in the presence of origin
    - anchor is only valid in the presence path
- Qualifiers canonical order (good practice): *origin*, *visit*, *anchor*, *path*, *lines* or *bytes*.

Syntax: qualifiers

SWHID syntax summary



Check the syntax description at the SWHID specifications

Comparing software

- SWHID is univocal:
  - Two software artifacts are identical (bit by bit) if their SWHID *core_identifier* are equal.
  - Two different SWHID *core_identifier correspond to two different software artifacts*
- In addition, two SWHIDs represent the same software artifact (or fragment thereof) if:
  - They both have the same core_identifier
  - They both have the same set of qualifiers
  - The values of these qualifiers are identical
    - For comparison purposes the order of the qualifiers does not matter.

SWHID:

open standard

- ISO/IEC 18670:2025
- Public and free of charge specification, published under Community-Spec-1.0 license
- SWHID is run under an open governance model
  - Feel free to contribute and contact the community
  - Check the FAQ and swhid.org for further information
- There is a reference implementation (*swhid-rs*) and a test suite for conformance
  - There are additional implementations in your language of choice

SWHID is an open standard

SWHID in action

swhid-rs

- SWHID standard reference implementation tool
- Maintained by SWHID WG
- Tool written in Rust, published on GH under MIT license
- swhid-rs implementation guide and user guide
- Installation:
  - Packages (new)
  - Rust: *cargo install swhid* (add *--features git* for VCS commands).

swhid-rs commands examples

Commands

Content Command

```
swhid content --file README.md
echo "Hello" | swhid content
```

Computes content SWHID from file or stdin.

Directory Command

```
swhid dir /path/to/directory
swhid dir --exclude-suffix .tmp --exclude-suffix .log .
```

Computes directory SWHID with optional exclusions.

Parse Command

```
swhid parse 'swh:1:cnt:...;origin=https://github.com/user/repo'
```

Parses and validates SWHID, printing in canonical format.

Verify Command

```
swhid verify --file README.md --expected 'swh:1:cnt:...'
```

Verifies file content matches expected SWHID.

Git Commands (with `--features git`)

```
swhid git revision --repo /path/to/repo
swhid git release --repo /path/to/repo --tag v1.0.0
swhid git snapshot --repo /path/to/repo
swhid git tags --repo /path/to/repo
```

Getting started

Developers, compliance experts & auditors

- Read the specification v1.2
- Check out implementations for your programming language (more coming)
  - swhid-rs

Researchers

- Browse publications for academic papers and research
- Explore use cases in software preservation

Onboarding Material

- SWHID kick-off meeting
- Wikipedia article
- Interoperable/OSOR article
- Presentation: Meet the SWHID
- Join the mailing list

Use cases

Use cases

- Who is using SWHID and how
- SWHID and pURL
- SWHID and CRA
- SWHID and SBOMs
- SWHID and Bidirectional Traceability

Use case: SWHID and Software Heritage

Why SwH?

El Mundo
2024-05-20

Pew Research Center. Article

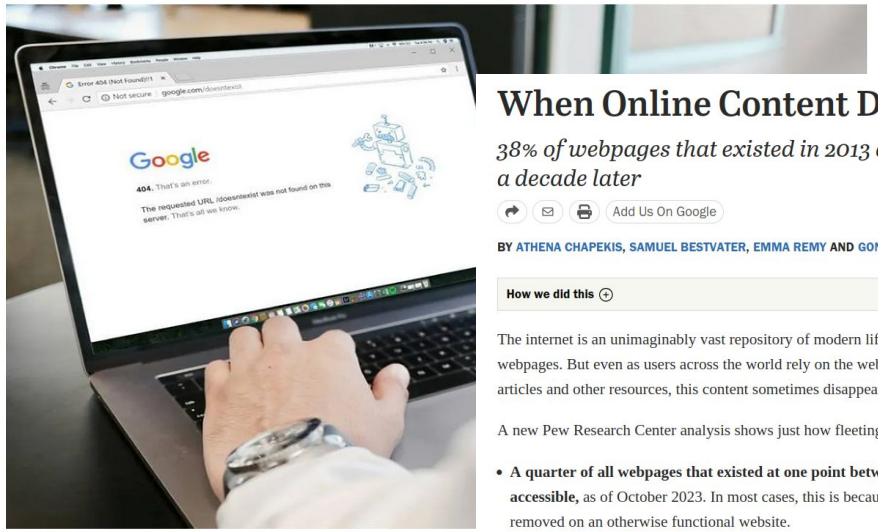
PIXEL TECNOLOGÍA

INTERNET

La web efímera: el 38% de las páginas que existían en 2013 se ha perdido

Enlaces rotos, información suprimida y problemas de configuración en los servidores web suelen estar detrás de este creciente problema

Comentar



Un ordenador en la pestaña de Google

Ángel Jiménez de Luis
EEUU
Actualizado Lunes, 20 mayo 2024 - 07:29

Un consejo habitual para quienes empiezan suele ser el de no enviar a nadie nada que no Internet nunca olvida.

La realidad, sin embargo, es mucho más com que lo hace a un ritmo mucho más acelerado del que pensamos. Un reciente estudio del Centro de Investigaciones Pew lo resume en una simple cifra: **el 38%**

When Online Content Disappears

38% of webpages that existed in 2013 are no longer accessible a decade later

Add Us On Google

BY ATHENA CHAPEKIS, SAMUEL BESTVATER, EMMA REMY AND GONZALO RIVERO

How we did this

The internet is an unimaginably vast repository of modern life, with hundreds of billions of indexed webpages. But even as users across the world rely on the web to access books, images, news articles and other resources, this content sometimes disappears from view.

A new Pew Research Center analysis shows just how fleeting online content actually is:

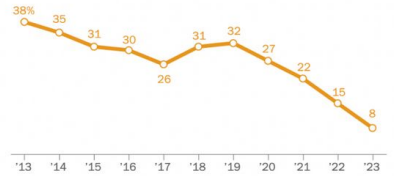
- **A quarter of all webpages that existed at one point between 2013 and 2023 are no longer accessible,** as of October 2023. In most cases, this is because an individual page was deleted or removed on an otherwise functional website.

- **For older content, this trend is even starker.** Some 38% of webpages that existed in 2013 are not available today, compared with 8% of pages that existed in 2023.

This "digital decay" occurs in many different online spaces. We examined the links that appear on

38% of webpages from 2013 are no longer accessible

% of links from each year that are no longer accessible as of October 2023



Source: Pew Research Center analysis of a random selection of URLs collected by the Common Crawl web repository (n=999,989) and checked using page and DNS response codes. Web pages defined as inaccessible if they returned a status code of 204, 400, 404, 410, 500, 501, 502, 503, 523 or did not return a valid status code.

We need a **global**, **long term** effort to build an **universal** archive of all software source code.

Make it **resilient** and make it **sustainable**.

What

SwH Mission

“Cultural heritage is the legacy of physical artifacts and intangible attributes of a group or society that are inherited from past generations, maintained in the present and bestowed for the benefit of future generations.

Software in source code form is produced by humans and is understandable by them; as such it is an important part of our heritage that we should not lose. Software is furthermore a key enabler for preserving other parts of our cultural heritage that we would *de facto* lose if we lose the software needed to access them. **Preserving software is essential for preserving our cultural heritage**.”

United Nations
Educational, Scientific and
Cultural Organization

Paris Call

«Software source code represents unique knowledge of humanity's recent history.

It is therefore crucial to work together collectively so that the knowledge embedded in software source code is properly preserved, valued and shared with all.

This lies at the core of UNESCO's cooperation with Inria to support the creation of Software Heritage, the global archive of software source code»

SwH is a...

1. Universal archive: preserve and share all software source code
2. **Reference Catalogue**: find and reference all software source code
3. Research infrastructure: enable analysis of all software source code

Nov'16

Jan'26

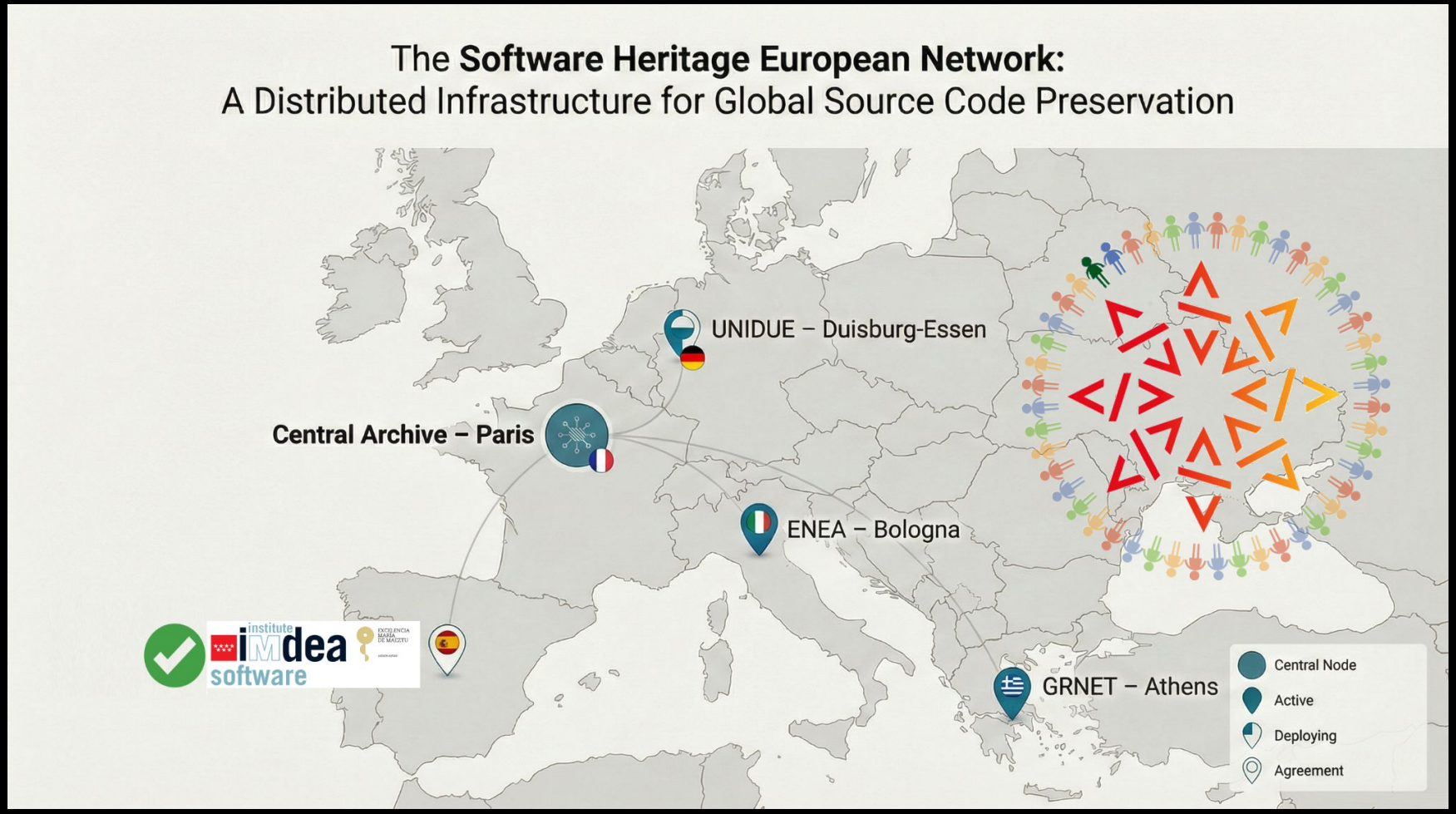
~ 3 PB

Universal archive

“ . . . let us save what remains: not by vaults and locks which fence them from the public eye and use in consigning them to the waste of time, but by such a multiplication of copies, as shall place them beyond the reach of accident.”

Thomas Jefferson, February 18, 1791

SwH mirrors


The Software Heritage European Network:
A Distributed Infrastructure for Global Source Code Preservation
UNIDUE – Duisburg-Essen
Central Archive – Paris
ENEA – Bologna
GRNET – Athens
Central Node
Active
Deploying
Agreement

Show me the archive

Search: https://archive.softwareheritage.org/browse

Example: Apollo 11 code

Download: https://archive.softwareheritage.org/vault/

Save (plugin): https://www.softwareheritage.org/browser-extensions

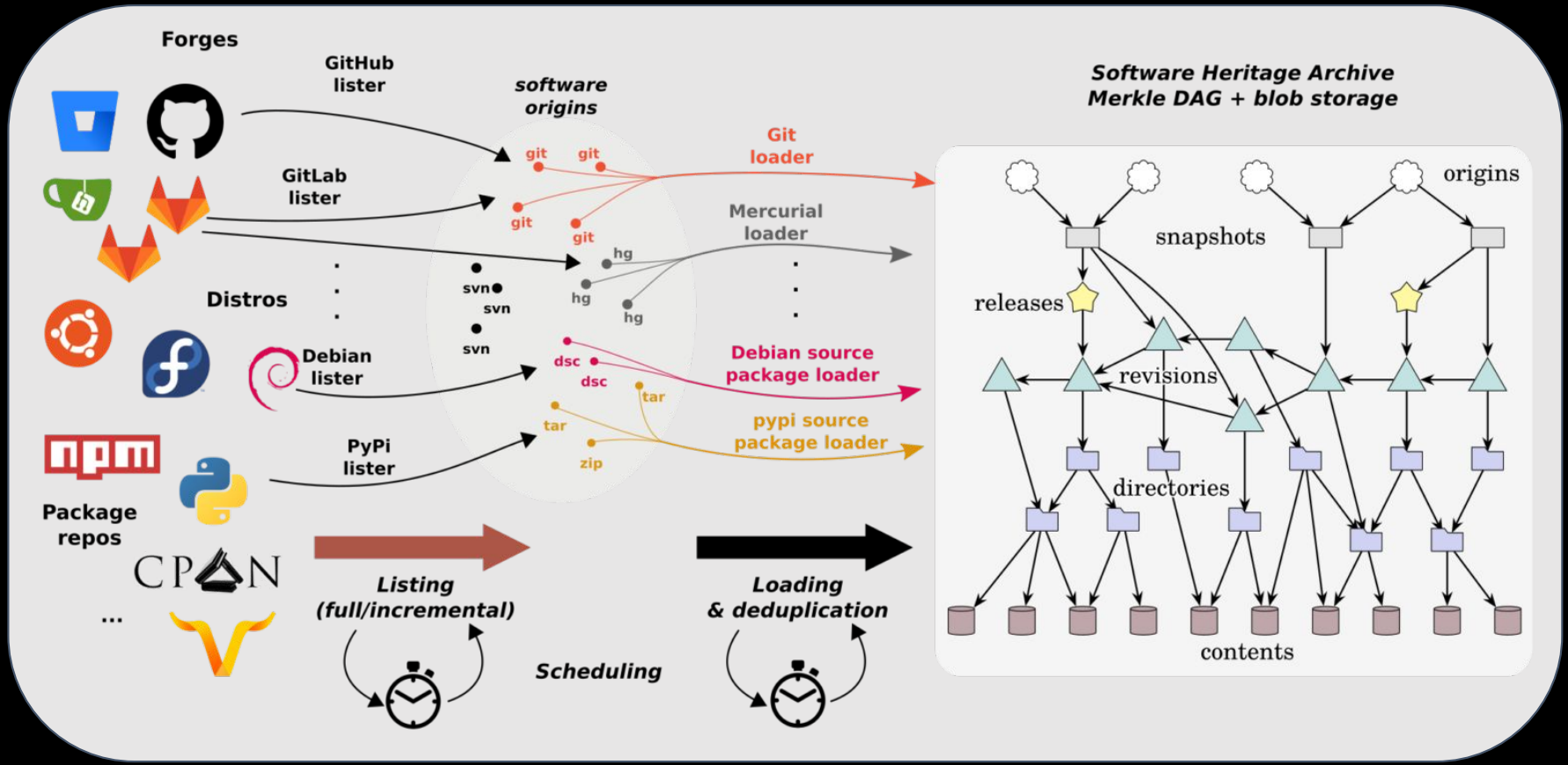
Save (web): https://archive.softwareheritage.org/save/

SWHID and SwH

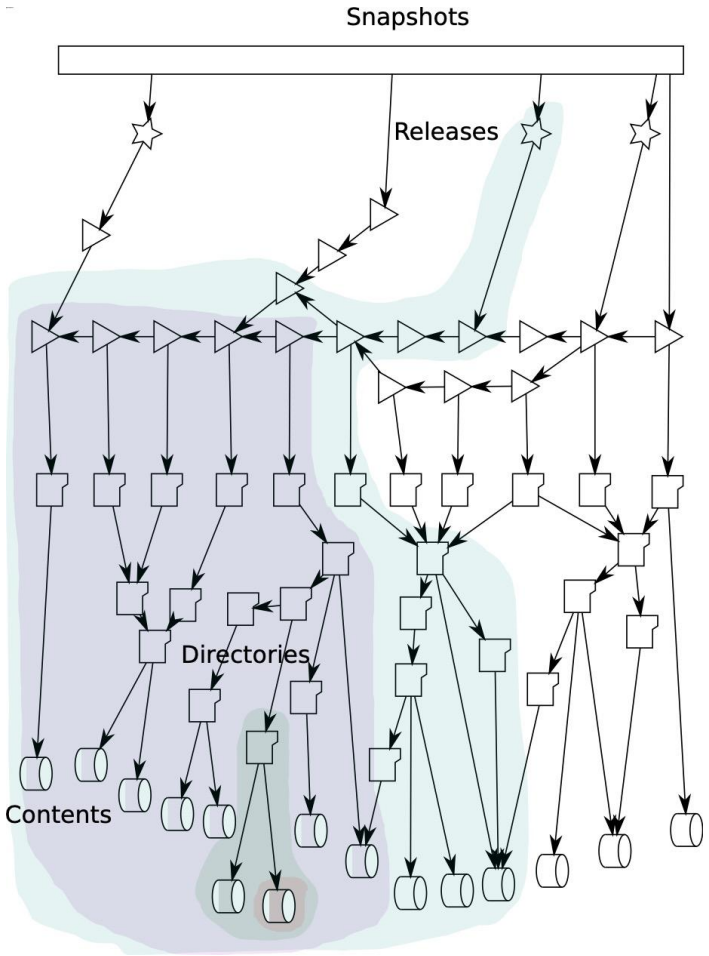
SwH is

- The inventor of SWHID's precursor, Software Heritage Identifier
- The first, and still today, largest SWHID implementation
- The main organization supporting the SWHID WG

Built for the purpose



- SwH uniform data model: Merkle DAG
- Supports deduplication



SwH is building a full graph of the software development evolution

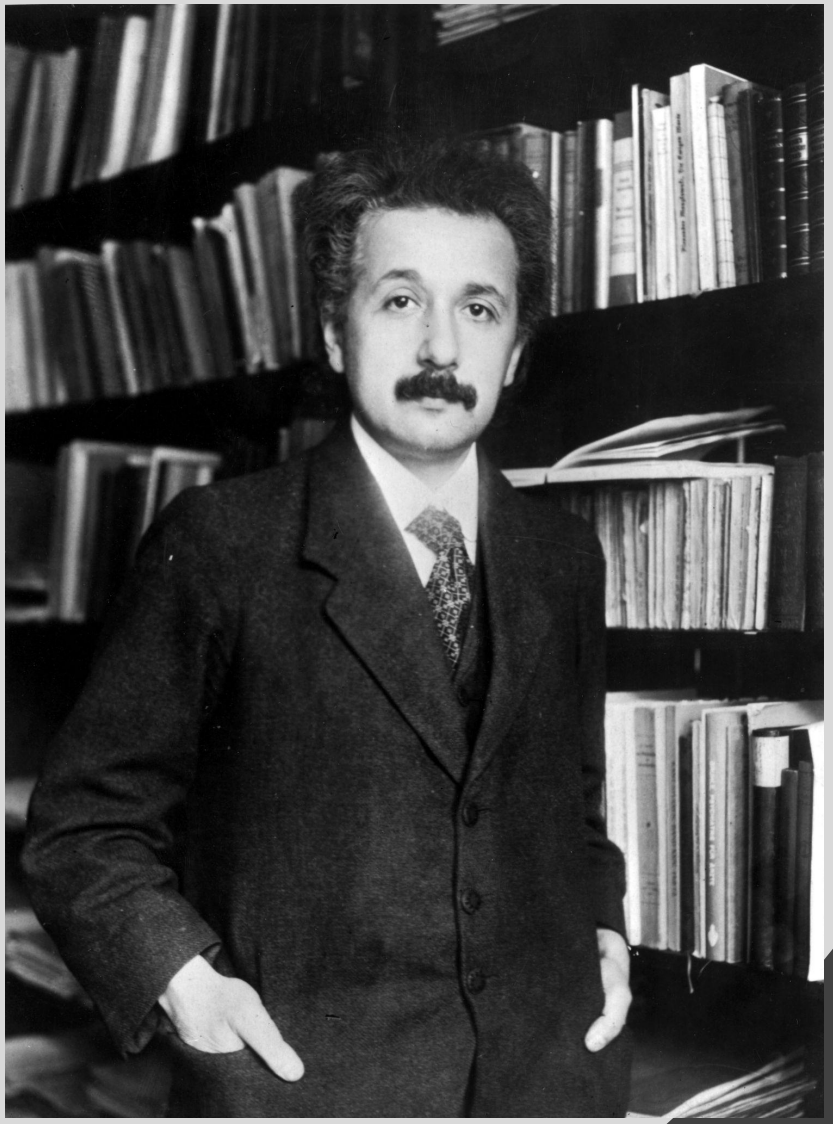
SwH as research infrastructure

- SwH Publications
- CodeCommons:
  - Archive more data, faster
  - Collect more metadata: license, language detection, version detection, dependencies detection, interaction with forges …
  - Detect similar code
- SwHSec:
  - Collect CVE
  - Link CVE to files in the history graph
  - Extend the graph with vulnerabilities
- the-stack-v2:
  - The most popular data set for training AI models

Software Heritage Statement on Large Language Models for Code

1. [...] The resulting *machine learning models* must be made available under a suitable open license, together with the documentation and toolings needed to use them.
2. The *initial training data extracted from the Software Heritage archive* must be fully and precisely identified by, for example, publishing the corresponding SWHID identifiers.
3. Mechanisms should be established, where possible, for authors to exclude their archived code from the training inputs before model training begins.

Principles summary. Check the full version



Downloaded from Getty Images

The only thing that you absolutely have to know, is the location of ~~the~~ ~~library~~".

SwH

-. *Albert Einstein*

archive.softwareheritage.org

Use case: OSS Licenses Fulfilment Requirements at Intel and SWHID

Intel

License compliance: OSS distribution obligations

1. The plan and PoC (2018–2019)
   a. By Alexios Zavras @Intel
   b. Today he is Core Team Member @ SWHID WG
2. Intel had 4 use cases to address: provide to customers…
   a. Unmodified OSS
   b. Modified OSS
   c. Intel's own OSS
   d. Modified OSS by Intel, with Intel's OSS
3. SwH + SWHID cover all four use cases. Deduplication matters
4. Intel provides SBOMs including SWHIDs where qualifiers (visit) point at SwH archive
   a. For direct push to archive(c.&d.), Intel became SwH Member
   b. Solution implemented in 2019

Use case:

SWHID and pURL

OSS: curl 8.5.0

- pURL: pkg:github/curl/curl@curl-8_5_0
- SWHID:
  - Software package: swh:1:rel:55b5fafb094ebe07ca8a5d4f79813c8b40670795
  - Software package + origin = pURL swh:1:rel:55b5fafb094ebe07ca8a5d4f79813c8b40670795;origin=https://github.com/curl/curl
  - Software package + origin + ref to SwH archive = pURL + ref to SwH archive: swh:1:rel:55b5fafb094ebe07ca8a5d4f79813c8b40670795;origin=https://github.com/curl/curl;visit=swh:1:snp:f1f7135b69cd85c18355f0d7a1d29ec94e8f4dc7

SWHID and pURL

SWHID can complement pURL

SWHID is content-based, therefor it is

- License agnostic, so SWHID identifies both, OSS and proprietary software
- Verifiable, as explained in the Comparing Software section
- Resilient to link rots for OSS, when combined with SwH

Use case: SWHID in the context of CRA

SWHID summary

Software Hash IDentifier essentials:

1. Intrinsic ID for any software artifact
2. Content-based
3. Persistent
4. Merkle DAG structure
5. ISO/IEC 18670

SWHID and CRA

Software Hash IDentifier essentials for CRA, and several other legislations:

1. **Univocal and verifiable ID:** no naming ambiguity
2. **License independent:** valid for open source and proprietary SW
3. **Technology independent:** tooling agnostic (generators, verifiers, managers…). open source reference implementations are available
4. **Platform independent**: valid regardless of where code is hosted
5. **Single-bit tamper detection**: any single-bit change—whether in a source code file or a compiled binary—will alter the SWHID
6. **Trusted**: open standard and open governance

SWHID + SwH and CRA

SWHID + SwH essentials for CRA:

1. **OSS Provenance**: optional qualifiers (origin, path, anchor, lines) provide the contextual metadata needed to "ensure and attest" to the provenance of open source components
2. **Transparent OSS Traceability**: bit-level cryptographic proof of component provenance and integrity by linking SBOM entries to intrinsic SWHIDs that include context qualifiers pointing at SwH archive
3. **OSS Persistent resolvability**: ensure artifacts remain retrievable for the full CRA-mandated 10–year period, surviving forge shutdowns, repository renames, service discontinuations or supplier closure

SWHID and SBOMs

SWHID in SPDX and CycloneDX

SWHID recognised as software artifacts ID at:

- SPDX:
  - *ExternalIdentifierType* vocabulary — *swhid*
  - *ContentIdentifierType* vocabulary — *swhid*, specifically for use in the *contentIdentifier* property of *SoftwareArtifact* elements
- CycloneDX: *evidence.identity* → field: "*swhid*"

SWHID and SBOM generation

SBOMs generation benefits from SWHID in several ways:

- Simplify declarative, machine-readable SBOMs
- Identify proprietary SW artifacts just like OSS, at both sides of the firewall
- Resolve software naming ambiguity: no name collision
- Maximise software identification accuracy: single-bit tamper precision
- Standardised software identification at custom granularity: snippets, firmware, BSPs, patches, blobs, branches…

SWHID and SPDX


```
        },
        {
            "name": "pkg:invent.kde.org/graphics/okular",
            "SPDXID": "SPDXRef-39b07aa9280defee605e9a5c76fbf099",
            "versionInfo": "19.07.80",
            "downloadLocation":
"https://invent.kde.org/graphics/okular/archive/v19.07.80.tar.gz",
            "filesAnalyzed": false,
            "supplier": "Organization: KDE",
            "homepage": "https://invent.kde.org/graphics/okular",
            "licenseDeclared": "BSD-2-Clause AND BSD-3-Clause AND DOC AND
GFDL-1.2-only AND GPL-2.0-only AND GPL-2.0-or-later AND GPL-3.0-only AND
LGPL-2.0-only AND LGPL-2.0-or-later AND MIT AND X11 AND Linux-man-pages-1-para
AND LicenseRef-scancode-kde-accepted-gpl AND LicenseRef-gpl-2.0-only-with-linux-
syscall-note",
            "licenseConcluded": "NOASSERTION",
            "copyrightText": "NOASSERTION",
            "externalRefs": [
                {
                    "referenceCategory": "PACKAGE_MANAGER",
                    "referenceLocator":
"pkg:https://invent.kde.org/graphics/okular",
                    "referenceType": "purl"
                }
            ],
            "checksums": [
                {
                    "algorithm": "MD5",
                    "checksumValue": "5f39918badc1ae31c09b401c1822509c07c6eb23”
                }
            ]
```

SWHID and CycloneDX

SWHID and CycloneDX


```
    {
      "bom-ref": "BomRef.kjcai72ah08.kuskquffg6o",
      "id": "CVE-2018-1000801",
      "source": {
        "name": "NVD",
        "url": "https://nvd.nist.gov/vuln/detail/CVE-2018-1000801"
      },
      "ratings": [
        {
          "severity": "medium"
        }
      ],
      "description": "okular version 18.08 and earlier contains a Directory
Traversal vulnerability in function \"unpackDoc...",
      "published": "2018-09-06T00:00:00.000Z",
      "updated": "2024-11-21T00:00:00.000Z",
      "affects": [
        {
          "ref": "pkg:deb/debian/okular",
          "versions": [
            {
              "version": "17.12.2.orig"
            },
            {
              "version": "4:22.12.0-1"
            }
          ]
        },
        {
          "ref": "pkg:invent.kde.org/graphics/okular",
          "versions": [
            {
              "version": "16.11.80"
            },
            {
              "version": "17.11.80"
            },
            {
              "version": "17.11.90"
            },
```


Match Case

SWHID and SBOMs verification

SWHID enhances SBOM content verifiability and auditability. The verification workflow:

1. Take the actual binary/source you received from the vendor
2. Compute its SWHID using swhid-rs (or any conformant tool), locally or in pipelines
3. Compare the computed SWHID against the one declared in the SBOM:
    a. If they match: you have cryptographic proof the artifact is exactly what the vendor declared, bit for bit
    b. If they differ: the artifact has been modified, intentionally or not — you know immediately, without trusting the vendor or any external registry

USE CASE — TELCO

SWHID in a 5G multi-vendor supply chain

A CNF delivery from three vendors, across time

THE SCENARIO

WHAT YOU RECEIVE

A 5G Core Network Function (CNF) container image delivered by a vendor. Inside: components from three different origins.

- Unmodified OSS libraries — OSS
- Vendor-modified OSS components — MODIFIED
- Proprietary firmware blobs — PROPRIETARY

ALL THREE GET A SWHID IN THE SBOM

License-agnostic. Content-based. The same mechanism works regardless of whether the component is open or closed source.

WHAT HAPPENS OVER TIME

T_0 — DAY OF DELIVERY

Instant tamper verification

Recompute SWHID on each received component. Compare against the SBOM. **No external service needed.** The math is the verification.

T+6m — SIX MONTHS LATER

Vendor's forge goes offline

OSS components remain resolvable via **SwH archive**. Proprietary blobs are still identifiable by their SWHID. No broken references.

T+2y — TWO YEARS LATER

CVE disclosed in a dependency

Query all SBOMs by SWHID. Know in **minutes** exactly which deployed products contain the affected component — down to the file level.

T+10y — TEN YEARS LATER

Audit or regulatory review

Every component is still **cryptographically traceable** to its exact origin. Supply chain history is intact and verifiable.

Key insight: SWHID treats OSS, modified OSS, and proprietary blobs identically — one mechanism for the entire stack

OSS · Modified · Proprietary

Bidirectional Traceability With SWHID In Automotive

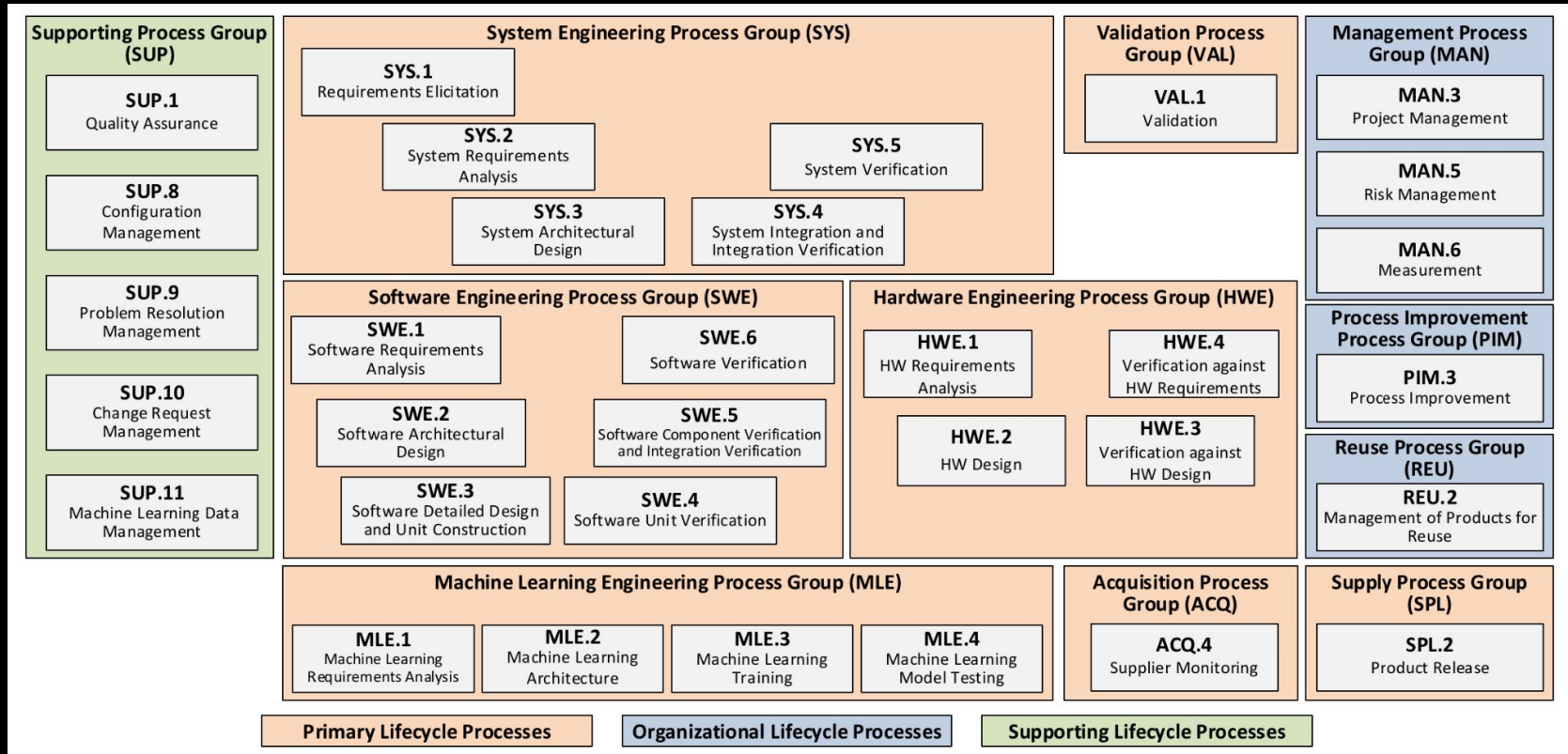


Figure 2 — Automotive SPICE process reference model - Overview

- Let's focus on BP4: ensure consistency and **establish bidirectional traceability**
- Let's start at SW Unit level: SWE.3<->SWE.4
- The use case is simple enough to work for J-Spice, ASQMS or even Lean Manufacturing applied to SW dev

SWHID and ASPICE

Bidirectional traceability

Bidirectional traceability in ASPICE: every work product (a requirement, design element, code unit, test case, test result, etc.) is connected to the what implements it and to the evidence that proves it — in both directions.

This implies:

- From requirement → see all designs, code, tests, and results that fulfill it (forward).
- From any test, code module, or document → see which requirement and regulation it exists for (backward).

Traceability and consistency are coupled. Traceability is used to support coverage checks, consistency checks, impact analysis of changes, and to demonstrate verification coverage during assessments, not just to show that links “exist”.

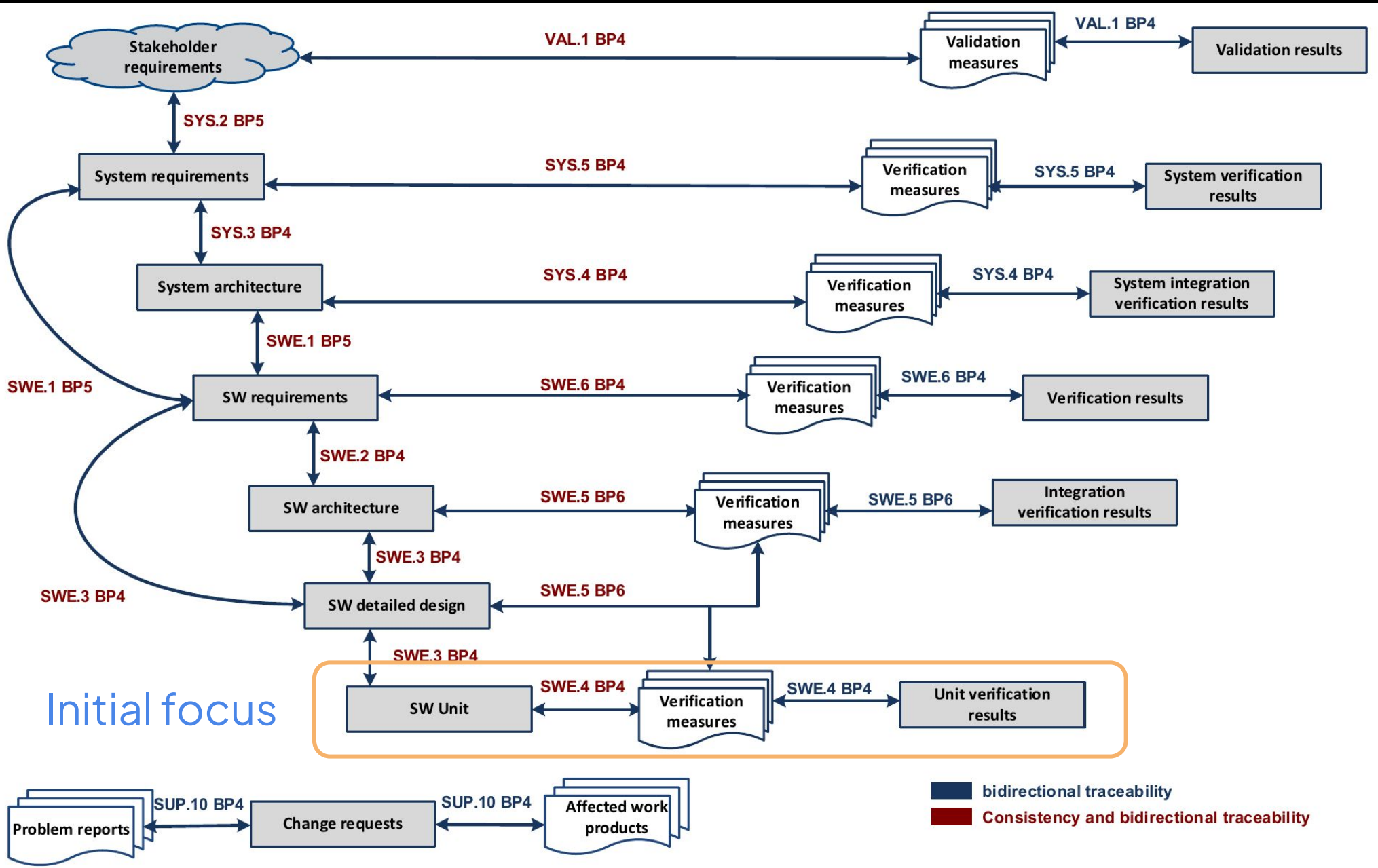


Figure C.2 — Consistency and traceability between system and software work products

STEP 1 - CREATE SWHIDs

Code
TaaC
RaaC
} XaaC

swh:1:cnt...

pswh:1:cnt...

Test as a Code
TaaC

Results as a Code
RaaC

swh:1:cnt ...

STEP 2 : RELATE SWHIDs

CODE	TEST	RESULT
swh:1:cnt...	swh:1:cnt...	swh:1:cnt...
swh:1:cnt...	swh:1:cnt...	swh:1:cnt...

Relational DD.BB.

a.k.a.

Traceability matrix

"Extrinsic" approach

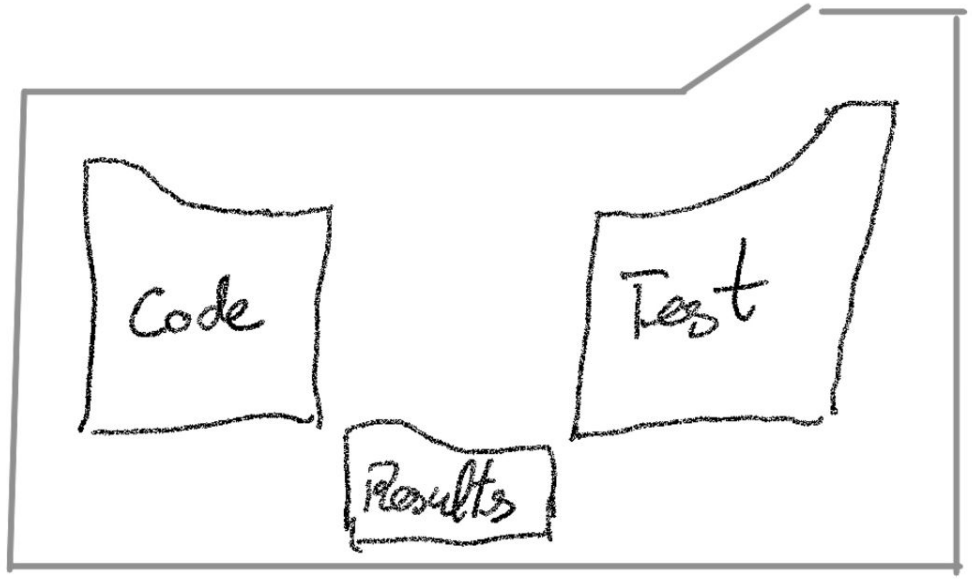
Scalability challenges...

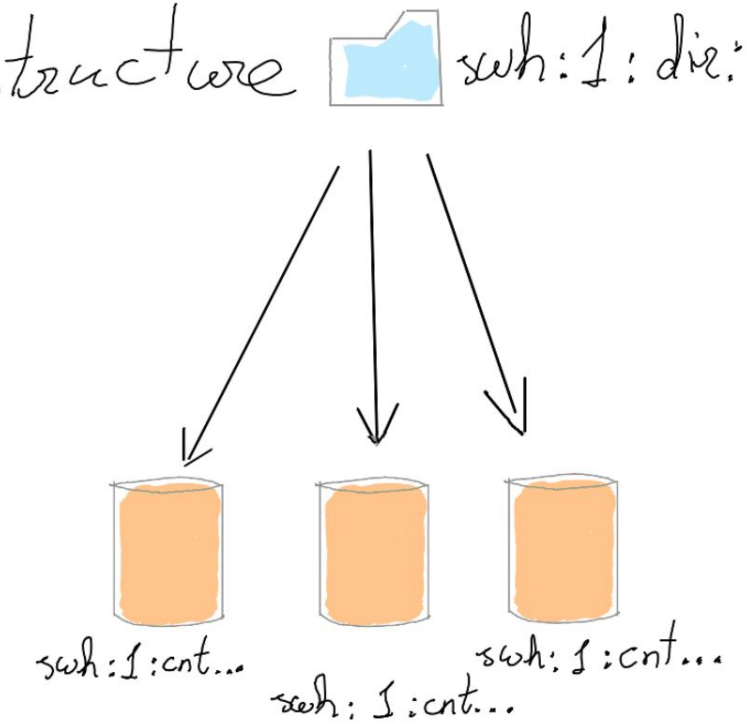
...just like SBOMs.

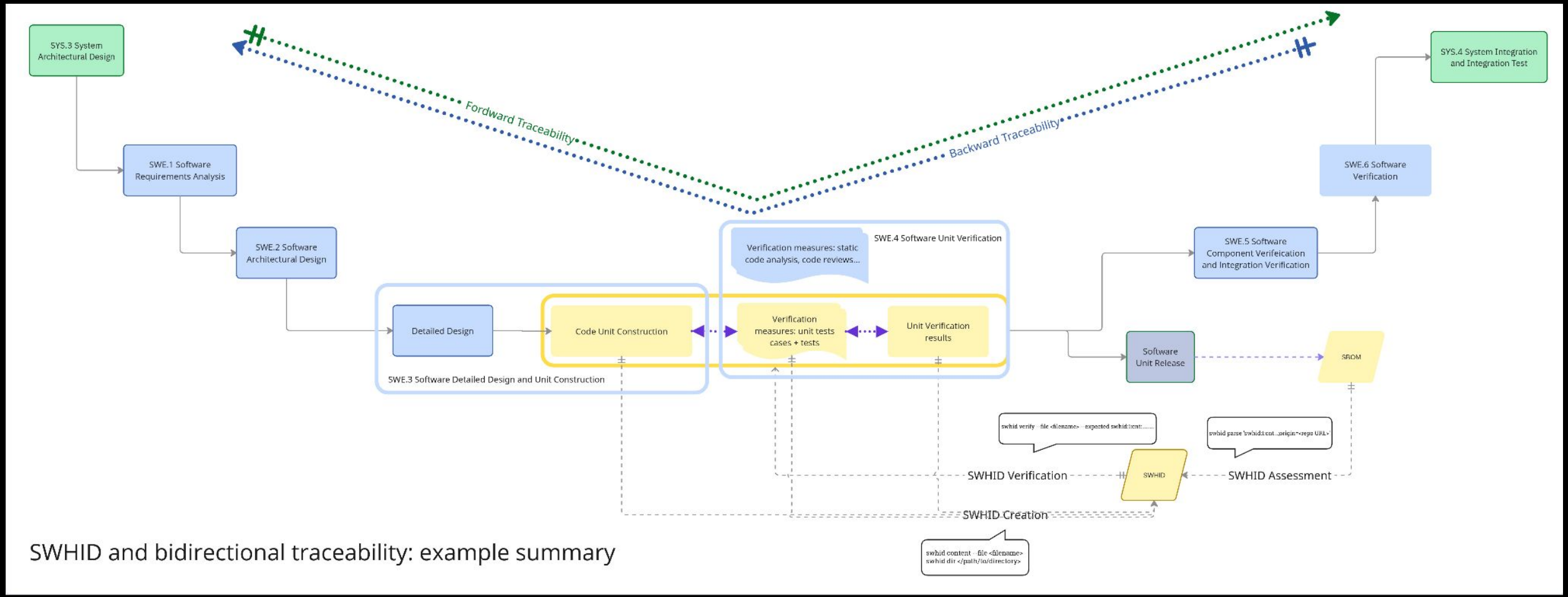
STEP 2 : RELATE SWHIDs

Reminder. SWHID is ...

- A content-based identifier
- Intrinsic
- Assumes a Merkle DAG structure







SWHID and bidirectional traceability: example summary

A single identifier across the entire software unit development process, with all the advantages that SWHID brings: intrinsic, content-based, standardised, platform and license independent, granularity, source code and binaries, etc.

This simple idea of

XaaC + linking code, tests and results

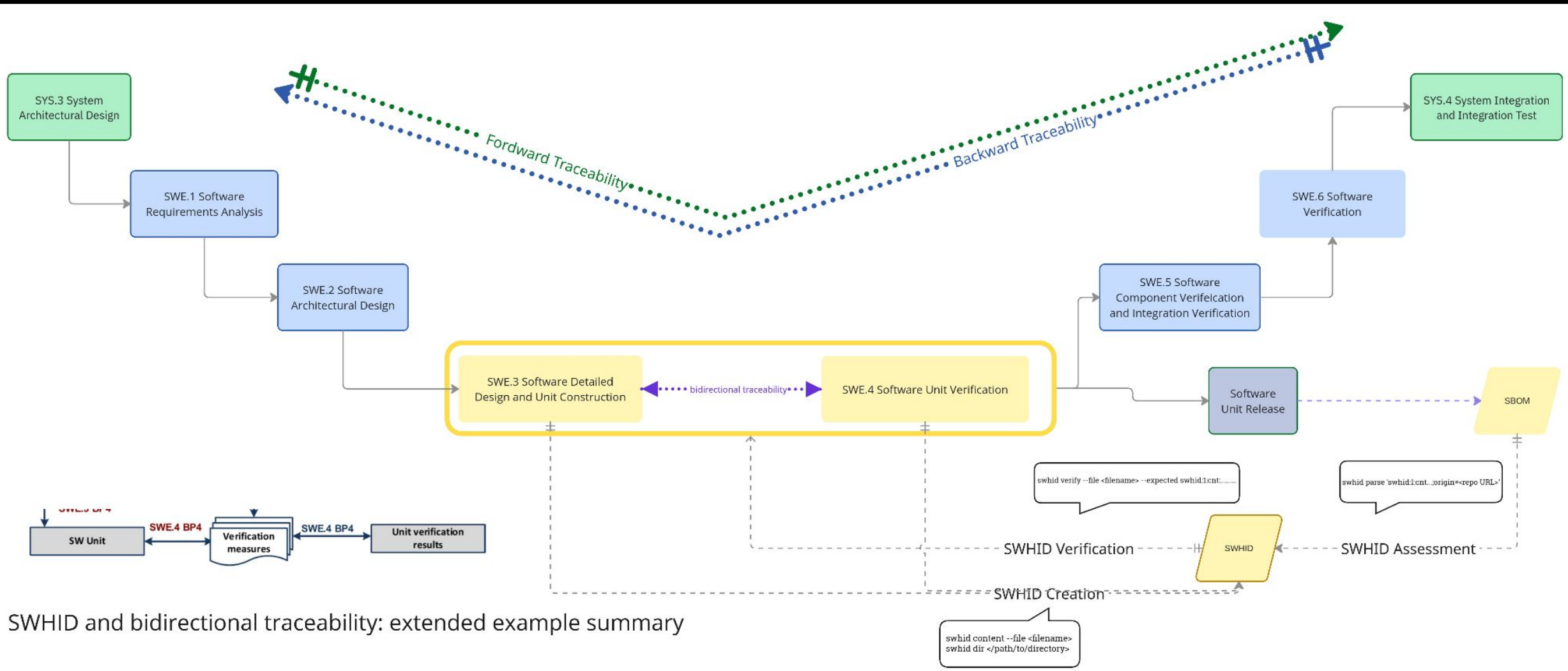
through SWHID

can be extended to:

- Test cases
- Code revisions
- Code designs: PlantUML , Mermaid, UML, SysML…

SWHID revolutionise the concept of evidence.

SWHID



SWHID and bidirectional traceability: extended example summary

Scalability:

- From traceability matrices to SWHID based (data modeled) services (i.e SwH archive)
- SBOMs as evidence tracker and communication tool

SWHID

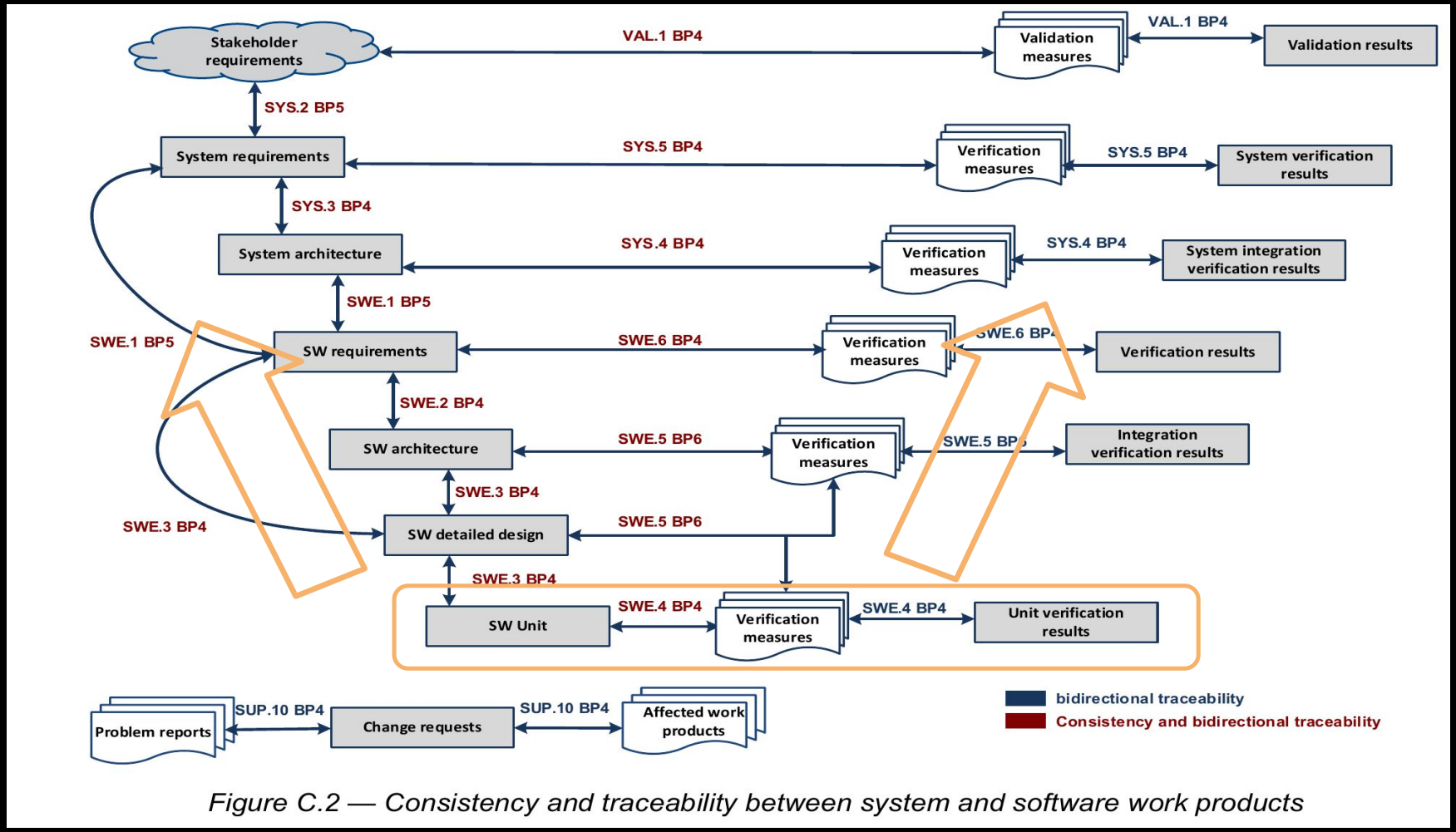


Figure C.2 — Consistency and traceability between system and software work products

This approach, based on a standardised software identifier can be extended to machine learning and system work products

Takeaways

1.- SWHID is lock-in and “single point of failure” resistant.

2.- SWHID adoption is unavoidable for identifying and managing software at scale in the AI era

3.- SWHID turns SBOMs into something “more manageable” across complex, multi-vendor supply chain environments — and satisfies OpenChain, CRA, and other compliance requirements as a consequence

4.- SWHID takes bidirectional traceability to another level, providing SBOMs an essential role across the development life cycle

5.- SWHID + SwH brings value to open source communities, academia, public administrations and industry, including automotive

Glossary and References

Glossary and References

Glossary

- SWHID: Software Hash IDentifier
- pURL: package URL
- Merkle DAG: Merkle Direct Acyclic Graph
- CRA: Cyber Resilience Act
- SwH: Software Heritage
- PoC: Proof of Concept
- ASPICE: Automotive SPICE®

References:

- OpenChain SBOM Telco Guide v1,1
- swhid.org

Images

- Page 4: Agustin Benito Bethencourt gravatar is registered by Agustin Benito Bethencourt.
- Page 32: attribution and reference in the images
- Page 35: UNESCO®
- Page 46: attribution and reference in the image
- Pages 59, 60 and 61: created by Agustín Benito Bethencourt using sbom-workbench and Spectacle
- Pages 65, 67 and 74: created by VDA Working Group 13 for Automotive SPICE® Process Assessment / Reference Model Version 4.0
- Pages 68, 69, 70, 71 and 73, created by Agustin Benito Bethencourt under the slide license

All other images created by SwH or SWHID.

SWHID

Introduction and use cases

Agustín Benito Bethencourt

Toscalix

April 2026