A complete compliance toolchain for Yocto projects

Alberto Pianon, Carlo Piana – **Array**

Fosdem2023 - SBOM DevRoom - Feb 5, 2023

Context

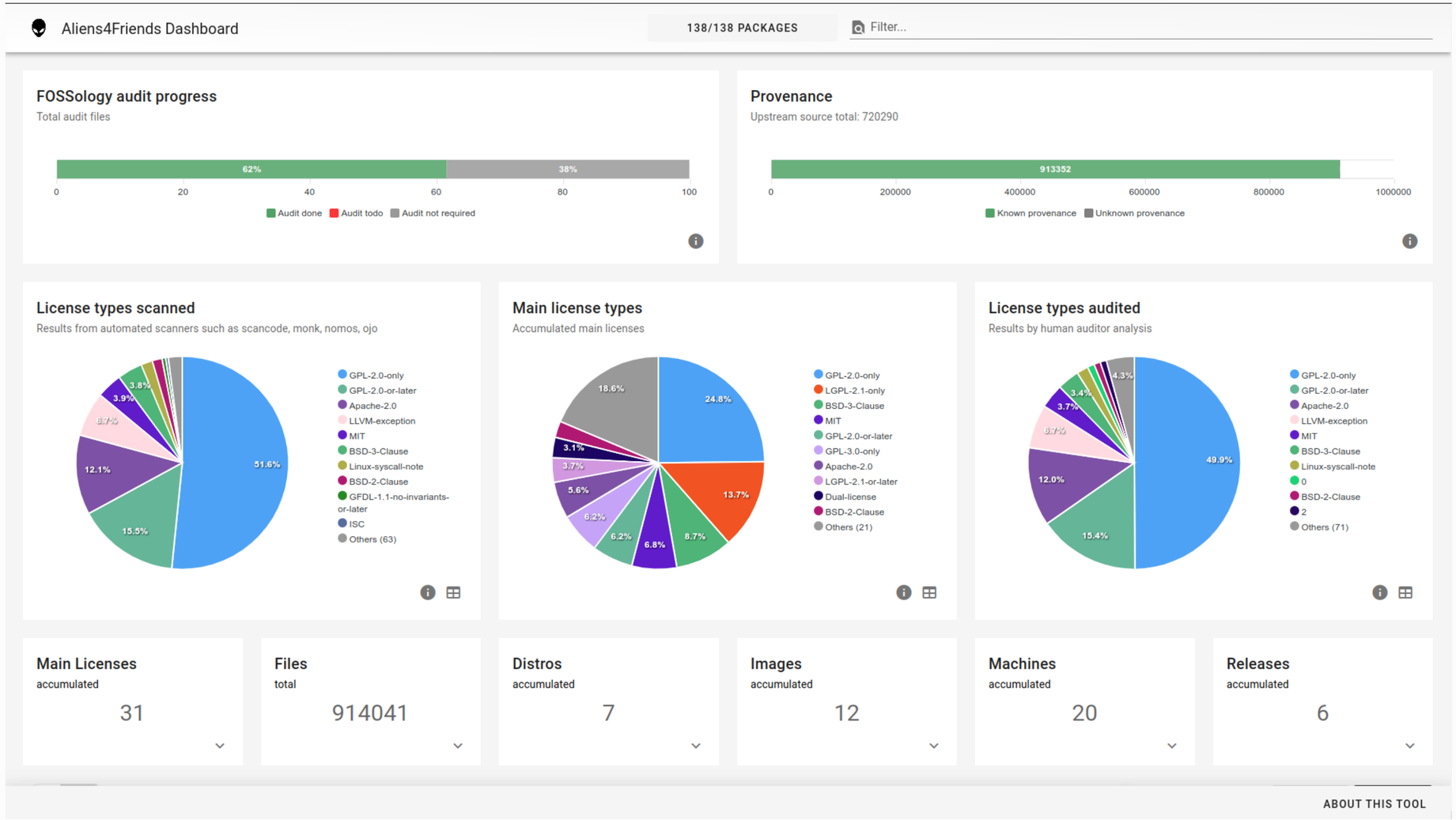
- Eclipse Foundation project
- a Yocto-based all-scenario OS platform project…
- …supporting 12 target machines, 2 build toolchains, 2 different kernels (**tens of build targets, all CI'd**)…
- …with an integrated Continuous Compliance process…
- …managed through a dedicated toolchain

A Toolchain For Compliance

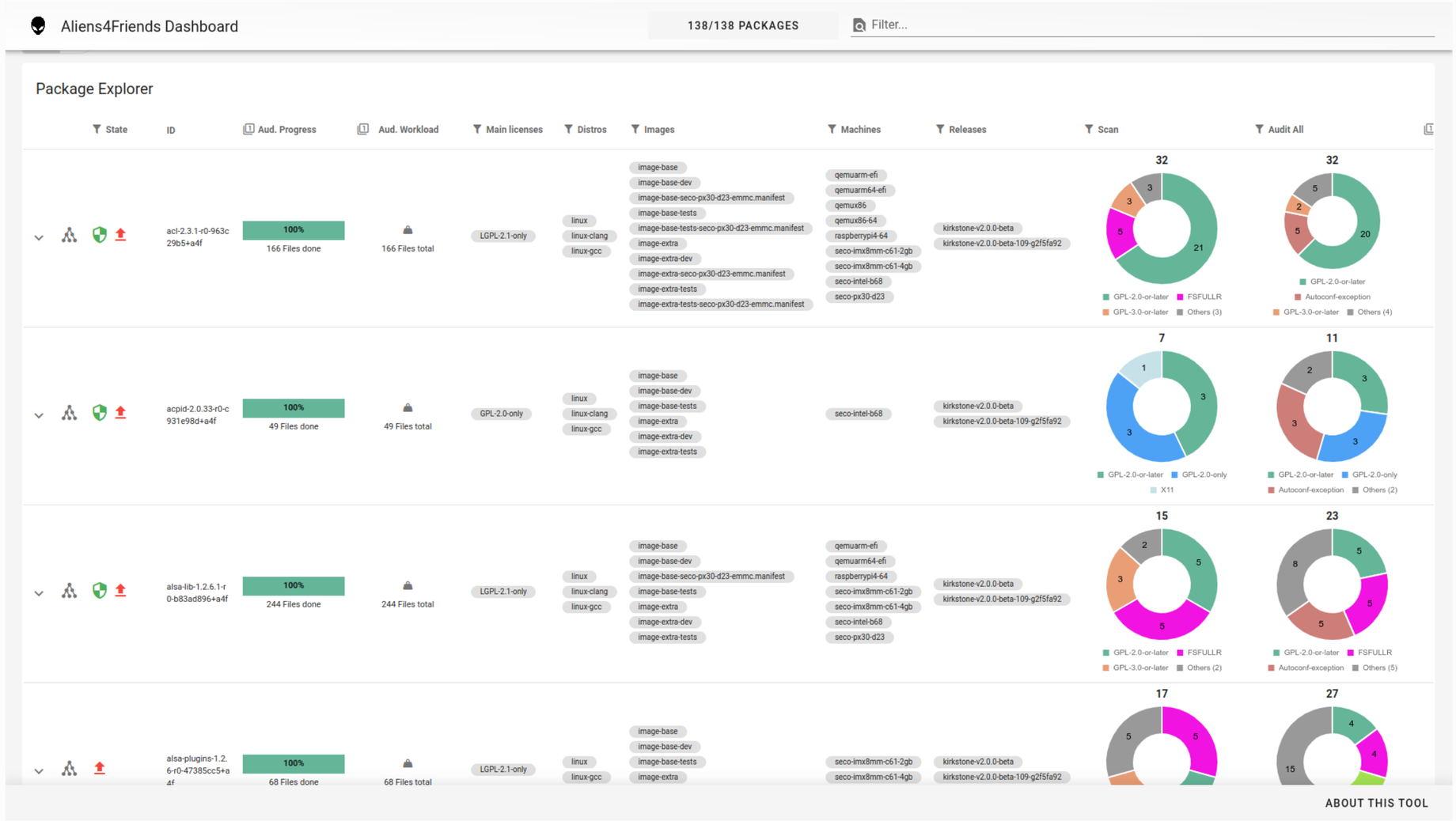
NOI Techpark	https://noi.bz.it/en
Array	https://array.eu

- an Eclipse Foundation project, too
- based on existing OSS tools (Fossology, Scancode)
- + a set of custom tools (aliens4friends, tinfoilhat, a4f dashboard)
- it can be implemented in any other Yocto-based project
- with latest Oniro release, we reached **100% coverage** on source components
  - also by automatically reusing Debian community work

sca.software.bz.it



sca.software.bz.it



ARRAY

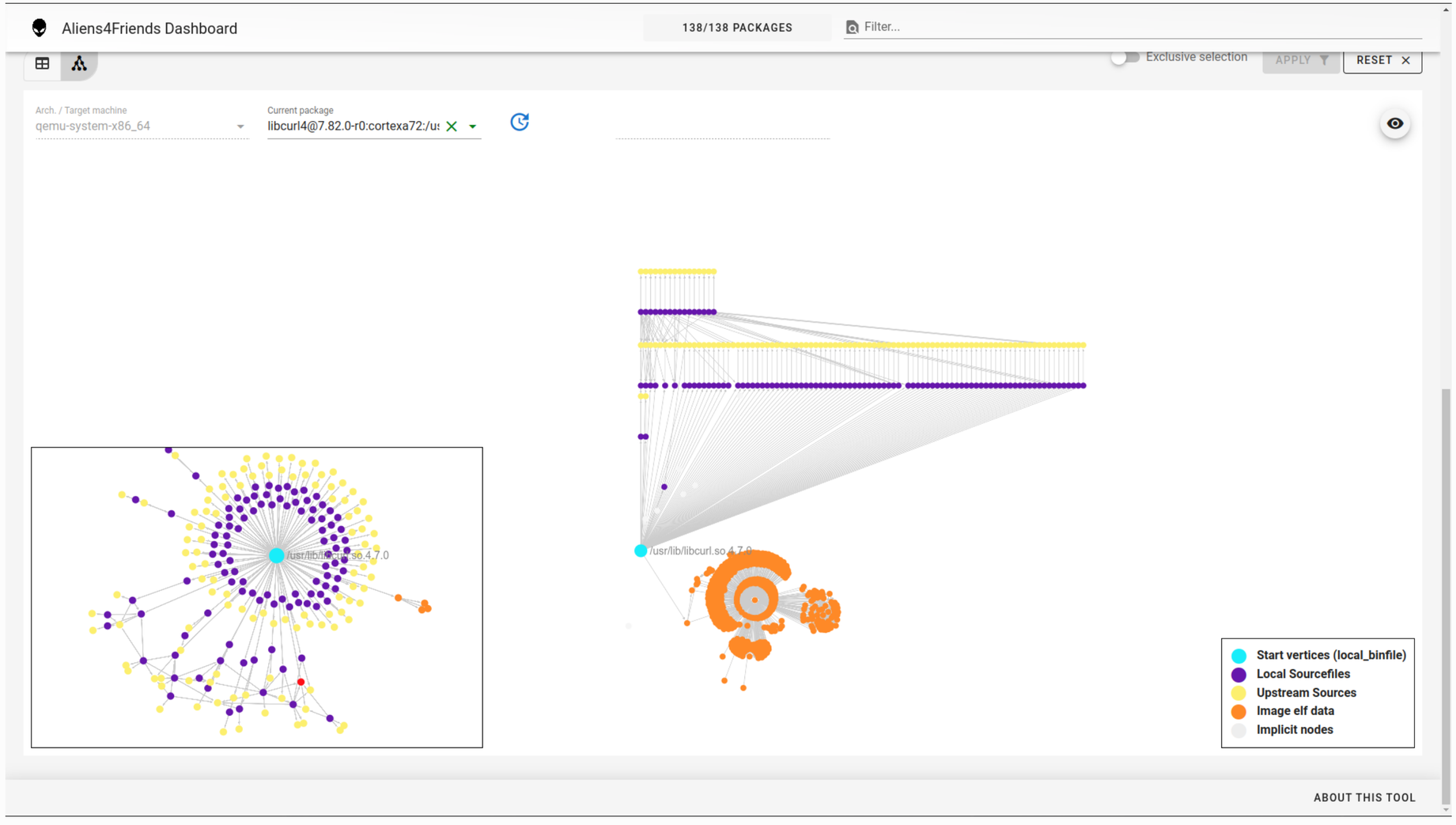
oniro

but now we want more!

R&D for Oniro Compliance Toolchain v2.0

- A **graph** database
- Software composition analysis, dependency analysis, automated license incompatibility checks
- To do that, we need to:
  - map all **license metadata*** on upstream source files down to binary files (**file-level mapping**)
    - (*) coming from our Audit Team, Debian community, possibly ClearlyDefined and other shared data sources
  - find a way to automatically combine **different inbound-outbound licenses** and check their **compatibility**

sca-staging.software.bz.it

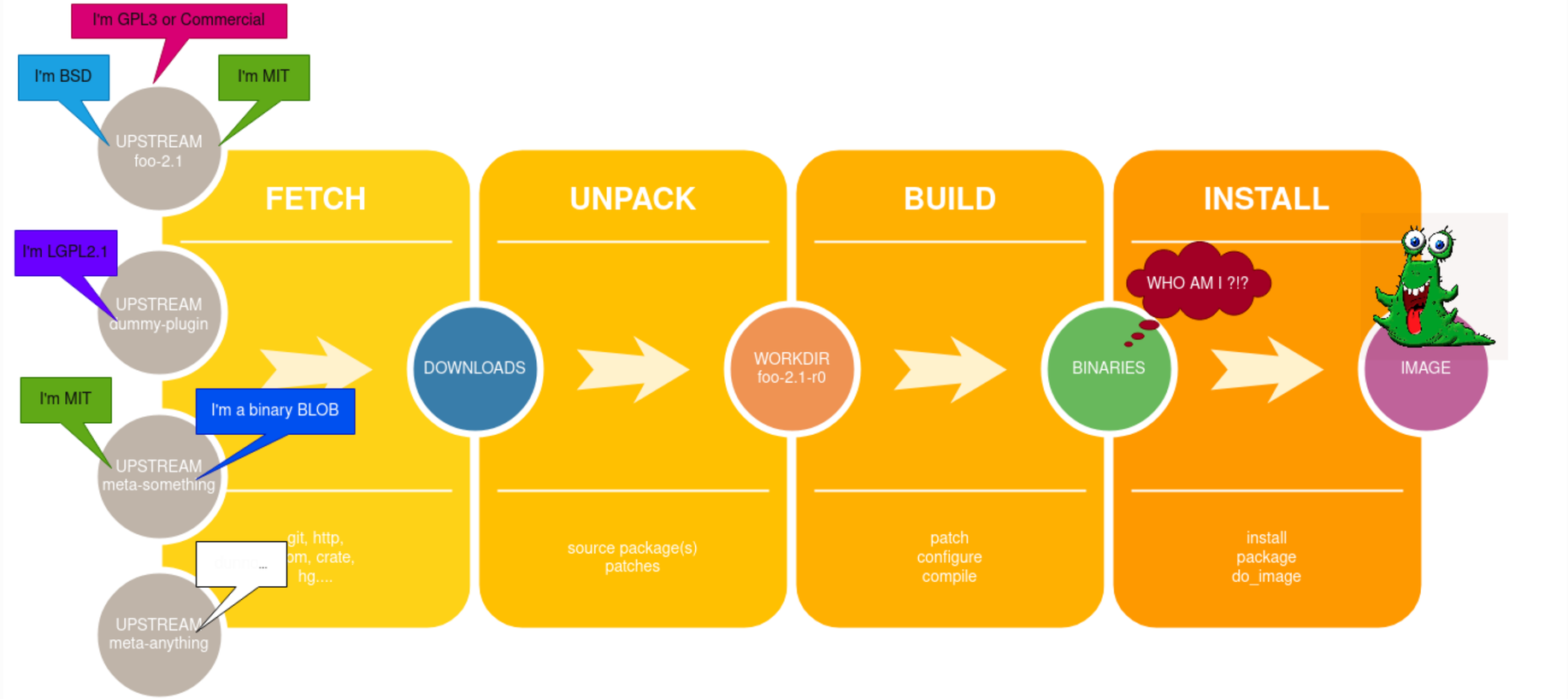


ARRAY

oniro

Why do we need this?

Yocto workflow (simplified)

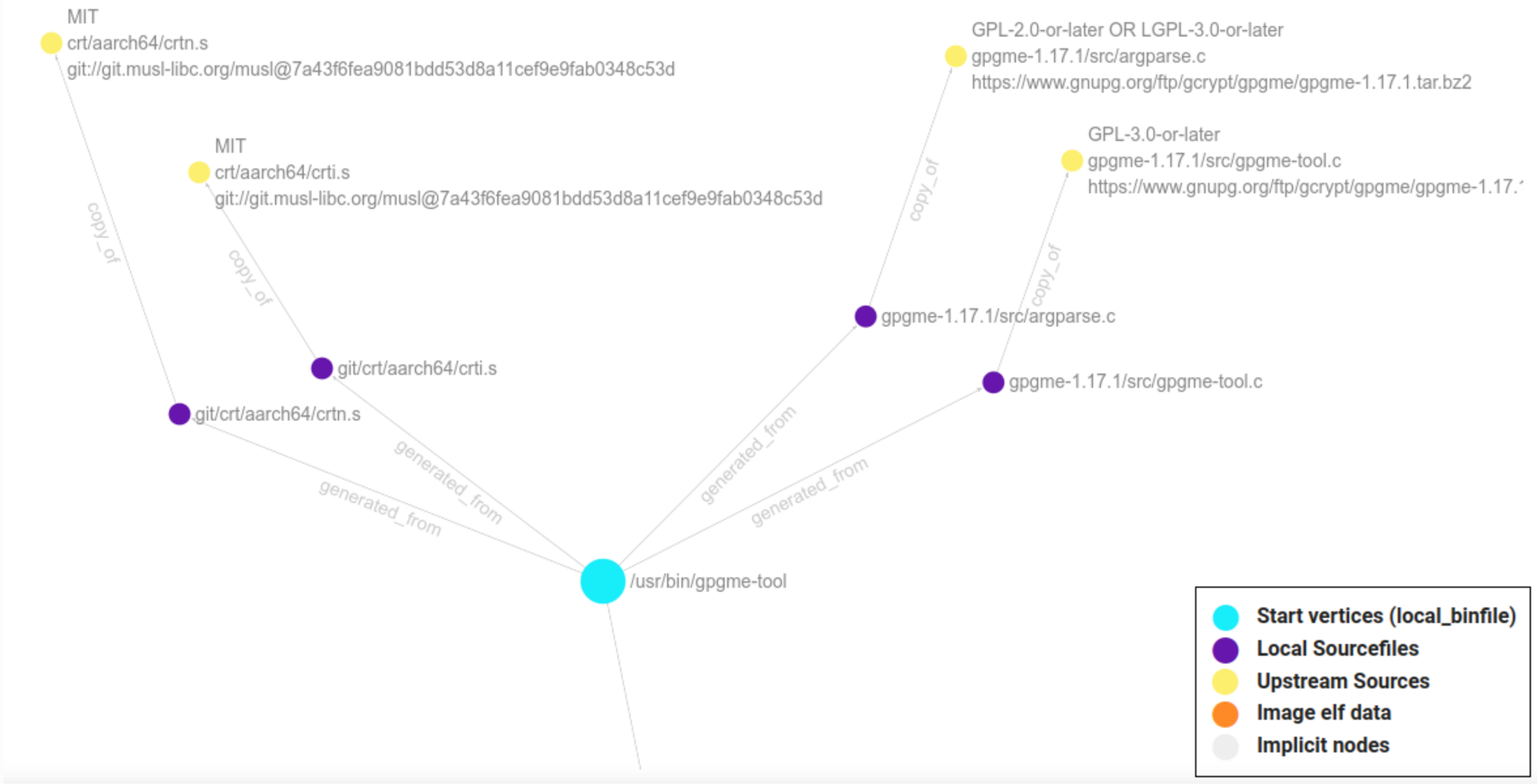


Logic Steps

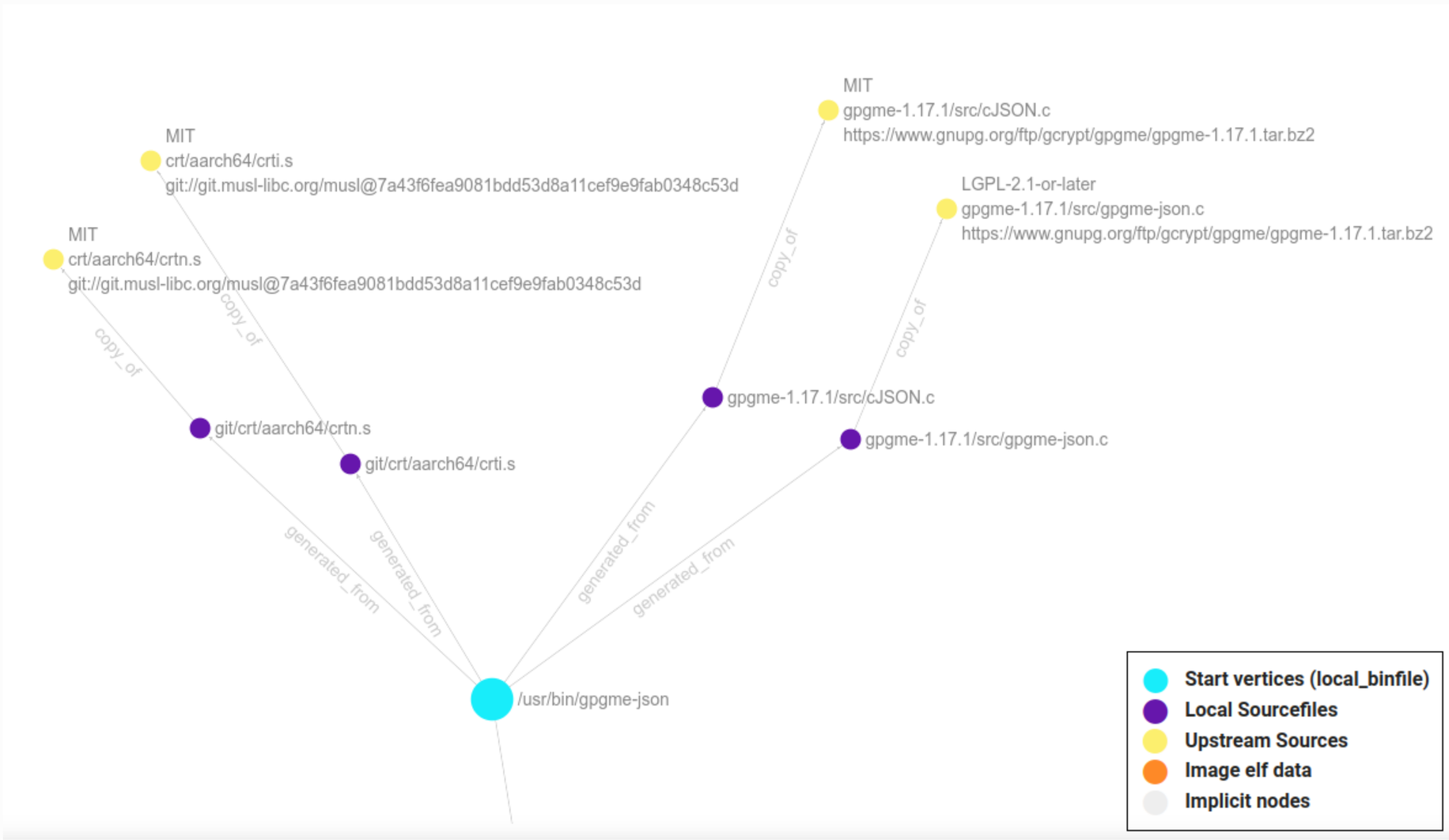
1. Find out the relationship with third party, upstream code (inbound)
2. Find out under which license(s) the inbound upstream software is, therefore the inbound Licenses
3. Find out if there is a possible combination of them
4. Match this combination with the outbound license(s).

For **each** artifact (**file!**)

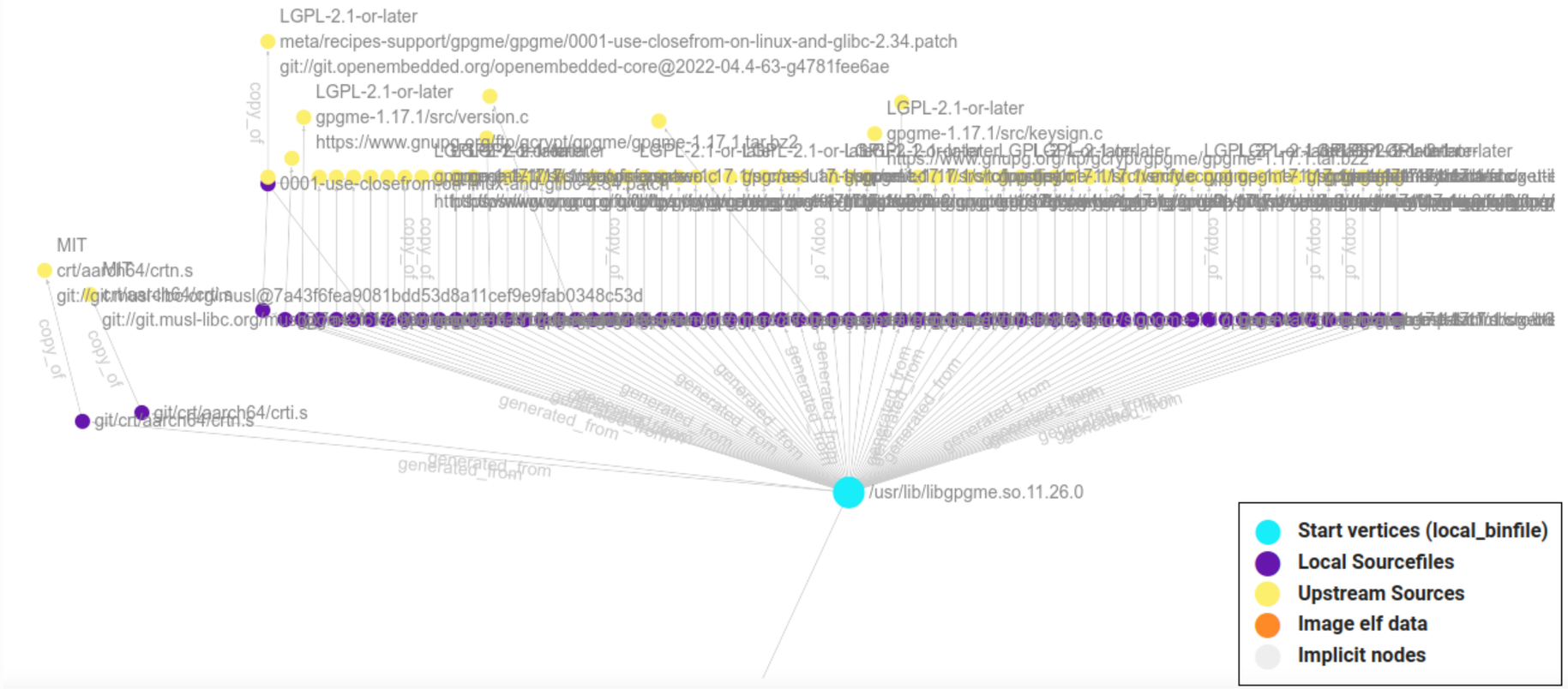
A simple example: GPGME (1)



A simple example: GPGME (2)



A simple (?) example: GPGME (3)



How can we handle that?

ARRAY

oniro

A battle beween License Cards!

Proposed syntax for license battle rules (tentative)

```
GPL-3.0-or-later vs LGPL-3.0-or-later, battlefield: strong, authority: FSF, result: GPL-3.0-or-later

LGPL-3.0-only vs GPL-2.0-or-later, battlefield: strong, authority: FSF, result: GPL-3.0-only

LGPL-3.0-only vs GPL-2.0-only, battlefield: any, authority: FSF, result: INVALID

GPL-3.0-only vs Apache-2.0, battlefield: any, authority: FSF, result: GPL-3.0-only

GPL-2.0-only vs Apache-2.0, battlefield: any, authority: FSF, result: INVALID

GPL-2.0-or-later vs Apache-2.0, battlefield: any, authority: FSF, result: GPL-3.0-or-later

LGPL-3.0-only vs MPL-2.0, battlefield: weak, authority: Mozilla, result: LGPL-3.0-only OR MPL-2.0

GPL-3.0-only vs MPL-2.0, battlefield: any, authority: Mozilla, result: GPL-3.0-only
```

Rules in action (1)


```
[
  {
    "inbound_licenses": [
      "MIT",
      "GPL-2.0-or-later OR LGPL-3.0-or-later",
      "GPL-3.0-or-later"
    ],
    "outbound_license": "GPL-3.0-or-later",
    "unhandled_licenses": [],
    "processed_license_options": [
      {
        "inbound_licenses": [
          "MIT",
          "GPL-2.0-or-later",
          "GPL-3.0-or-later"
        ],
        "results": {
          "prevailing_licenses": [
            "GPL-3.0-or-later"
          ],
          "decisions": [
            "GPL-2.0-or-later vs MIT, result: GPL-2.0-or-later",
            "GPL-3.0-or-later vs MIT, result: GPL-3.0-or-later",
            "GPL-3.0-or-later vs GPL-2.0-or-later, result: GPL-3.0-or-later"
          ],
          "unhandled_licenses": []
```

Rules in action (2)


```
{
  "processed_license_options": [
    {
      "inbound_licenses": ["MIT", "LGPL-3.0-or-later", "GPL-3.0-or-later"],
      "results": {
        "prevailing_licenses": ["GPL-3.0-or-later"],
        "decisions": [
          "GPL-3.0-or-later vs LGPL-3.0-or-later, result: GPL-3.0-or-later",
          "GPL-3.0-or-later vs MIT, result: GPL-3.0-or-later",
          "LGPL-3.0-or-later vs MIT, result: LGPL-3.0-or-later"
        ],
        "unhandled_licenses": []
      }
    }
  ]
}
```

How we collect required data on Yocto's side

- map upstream source files to local workdir source files to binary files
  - consume metadata coming from Yocto
  - fetch upstream source packages (including Yocto layers with pacthes etc.) separately from each other, and then map them to local workdir source files
- in our PoC we do that with an external, post-mortem script using `bb.tinfoil`
- it may be integrated in Yocto **`create-spdx.bbclass`**

Q&A: we need your feedback!

Thank you for your attention

https://array.eu
https://projects.eclipse.org/projects/oniro.oniro-compliancetoolchain
https://gitlab.eclipse.org/eclipse/oniro-compliancetoolchain/toolchain

---

Array Alberto Pianon Carlo Piana