Package URL (purl) and Version Range specs (vers)

The Mostly Universal Package URL and Version Range Identifiers for Dependencies and Vulnerabilities

Hritik Vijay

- Passionate about open source, Linux and infosec
- Co-maintainer at VulnerableCode and univers
- Application Security engineer at CRED
- Google summer of code mentor and a participant
- Supporter of suckless philosophy
- hritik.elf@gmail.com, @MrHritik

Philippe Ombredanne

- ▷ ScanCode and AboutCode projects lead and maintainer
- ▷ Creator of **Package URL**, co-founder of **SPDX**, ClearlyDefined
- ▷ FOSS veteran, long time **Google Summer of Code** mentor
- ▷ Co-founder and CTO of nexB Inc., makes of DejaCode
- ▷ Weird facts and claims to fame
  - Signed off on the **largest deletion of lines of code** in the **Linux kernel** (but these were only comments)
  - Unrepentant **code hoarder**. Had 60,000+ GH forks now down only to 20K forks
- ▷ pombredanne@gmail.com irc:pombreda

Agenda

- The problem: more FOSS dependencies, more bugs
  - But better code, faster and cheaper
- Solution elements
  - Identifying packages is hard
  - Understanding dependency resolution is hard
  - Understanding vulnerable ranges is hard
  - Versioning is hard
  - What if?
- Solution: purl and vers

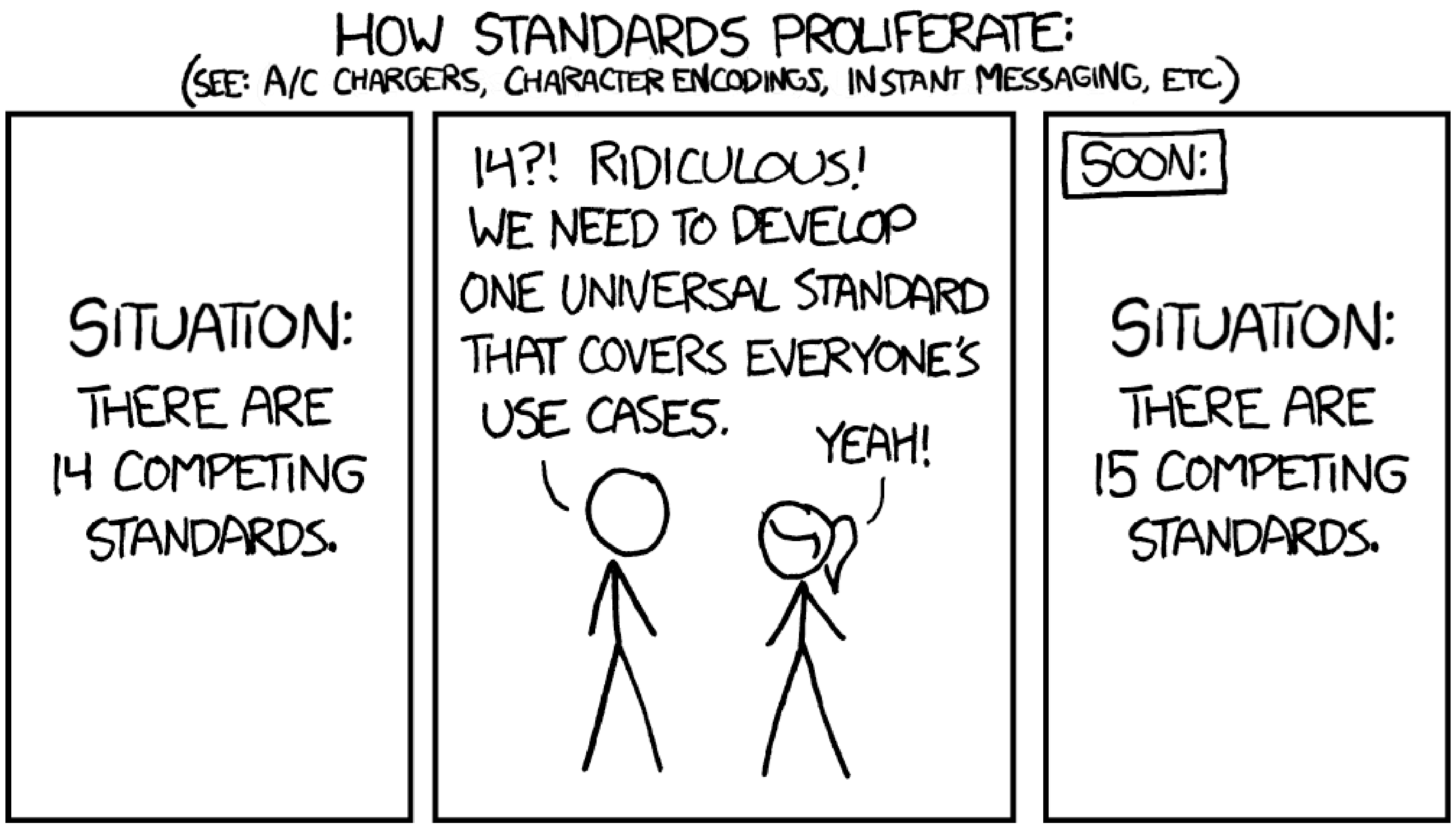
Things are getting more complex!!

- ▷ Ever **more FOSS packages** are reused. Reused like LEGO®
  - *10x to 100x more than a few years ago*
- ▷ Complex stacks with **multiple technologies and languages**
  - Deep dependency trees on both application and system packages
- ▷ **Unstated dependencies** across
  - package **ecosystem** boundaries and **system** and **application** boundaries
- ▷ **More bugs and vulnerabilities!**
- ▷ **Difficulty to automate dependencies and vulnerabilities**

How can we deal with this complexity?

- Hand crafted, good old **README** with installation instructions
- Replace all **package managers** with one to rule them all with massive **mono repo** and **"hermetic"** build system (Bazel, BUCK)
- ... **general purpose** package managers (Spack, Conda)
- ... **"functional"** package managers (Nix or Guix)
- ... and containers (to hide the details in a bigger tarball)

We need one standard to accommodate them all!


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Credits: https://xkcd.com/927/

Meet Package URL (purl)

- **Problem**: Each ecosystem has its own conventions to identify, locate and provision software packages
- **Solution**: An expressive **package-url string**, minimalist yet obvious
- Identify & locate software packages reliably across tools and languages.

```
pkg:npm/foobar@12.3.1
pkg:pypi/django@1.11.1
```

- Started with ScanCode and VulnerableCode and now adopted in many places
- Now a **de-facto standard** used in LF ACT ORT, LF ACT Tren, LF OSSF OSV, CycloneDX, LF SPDX, Sonatype OSSIndex, OASIS CSAF, Anchore Syft and Grype, Microsoft OSSGadget, GitHub and many other places.
- Libraries in Java, PHP, Go, Python, JavaScript, Ruby, Swift, Rust, .Net, etc.
- Recommended by the US NTIA as an SBOM package identifier

Package URL vs. other approaches

CPE	purl
cpe:2.3:a:apache:log4j:2.4:*:*:*:*:*:*:*	**pkg:maven/org.apache.logging.log4j/log4j-core@2.4**
cpe:2.3:a:mark_pilgrim:feedparser:4.1:*:*:*:*:*:*:*	**pkg:pypi/feedparser@4.1**

- Improved readability and obviousness: **you can find a purl origin!**
- No arbitrary vendor assignment
- NVD is adding "ecosystem" support in CVE API v5
- OSSF OSV is supporting both purl and an "ecosystem"

Package URL in the news

*"Component verification and vulnerability reporting are supported by some **SBOM data formats today.** Globally unique identifiers is a work in process supported by **the leading data formats for package URLs (PURLs)."***

https://linuxfoundation.org/wp-content/uploads/LFResearch_SBOM_Report_final.pdf

Software Bill of Materials (SBOM) and Cybersecurity Readiness

January 2022

Stephen Hendrick, VP Research, The Linux Foundation

Now we can name a package... version anyone?

- ▷ Resolving package dependencies
  - "I require package <ABC>, version 2.0 or later versions"
- ▷ Affected vulnerable versions
  - "vulnerability CVE-2021-1 affects <XYZ>, version 3.1 and version 4.2 but not version 5"
- ▷ **Version numbers should be boring**
- ▷ Yet each ecosystem has its own way for version and range!

Debian, RPM, npm, PyPI, Ruby, etc. have their different notations using comparators **>, < or =**, or **tilde** ~ or **caret** ^ or **star** *

- ▷ Each resolves a dependency version in a range differently

What if ....

- ▷ We could express a dependency in **a mostly universal** way?
- ▷ And not replace all package managers BUT rather rule them all

- ▷ Use **Package URL** "**purl**" to name a package across ecosystem
- ▷ Add new "**vers" Version Range Spec** for ranges
  - For any ecosystem, building on Package URL "package type" and namespace
  - Simplified comparators set: **>, <, =, !=, <=, >=**
  - Ecosystem-specific version comparison
- ▷ Designed for **dependency** ranges **AND vulnerability** ranges

Version Range Spec (vers)

- Problem: Each ecosystem has its own convention to specify version ranges
- Solution: An expressive **version range string**, minimalist yet obvious, a purl adjunct
- Specify ranges reliably across tools and languages for deps **and** vulnerabilities. A version range specifier ("vers") is a URI with this syntax:

```
vers:npm/1.2.3|>=2.0.0|<5.0.0

vers:pypi/0.0.1|0.0.2|0.0.3|1.0|2.0pre1
```

- Started with VulnerableCode with "univers" library and now used in CycloneDX and ScanCode.
- Can pave the way to universal dependency resolution and better vulnerabilities processing

Alternative to vers

- **Single syntax for everything** Such as CPEs, everything is SEMVER
- **Use node-semver for everything** GitHub advisories
- **Use each ecosystem specs, plus new defined specs** Gitlab advisories
- **Enumerate all the versions in a range** OSSF OSV (but does ranges too)
- **Use Maven mathematical notation** Snyk

Putting it all together

A mostly universal package name (**purl**) with a mostly universal version range (**vers**) opens up many possibilities:

- **Store vulnerable version ranges and evaluate later if a version is vulnerable**
- **Build a multi package installer for many ecosystems**
- **Write mostly universal dependency declarations**
- **Write a mostly universal dependency resolver?**

An incremental approach instead of replacing everything

Who is using it?

- **CycloneDX: purl and vers**
- **SPDX: purl**
- **OSSF OSV: purl**
- **OASIS CSAF: purl**
- **VulnerableCode, Univers and ScanCode**
  - Where purl and vers started
  - Some univers code is also used in Google OSV
    - https://github.com/nexb/vulnerablecode
    - https://github.com/nexB/univers
- And many more tools and specs

Conclusion:
The Naming of Cats is a difficult matter
But mostly solved!

- **Software package** names
  - Mostly solved with **Package URL** emerging as a de-facto standard
- **Version range** notation for dependencies and vulnerable ranges
  - New mini spec for **Version Range Specifiers**
- **Vulnerability** identifiers
  - Mostly solved with NVD's **CVE** and ever growing aliases list
- **Version control system** references
  - Likely solved with **VCS URLs** adapted from Python pip, now in SPDX

Credit https://www.severnedgevets.co.uk/pets/advice/advice-new-kitten-owners
Credit: T.S. Eliot, "Old Possum's Book of Practical Cats"

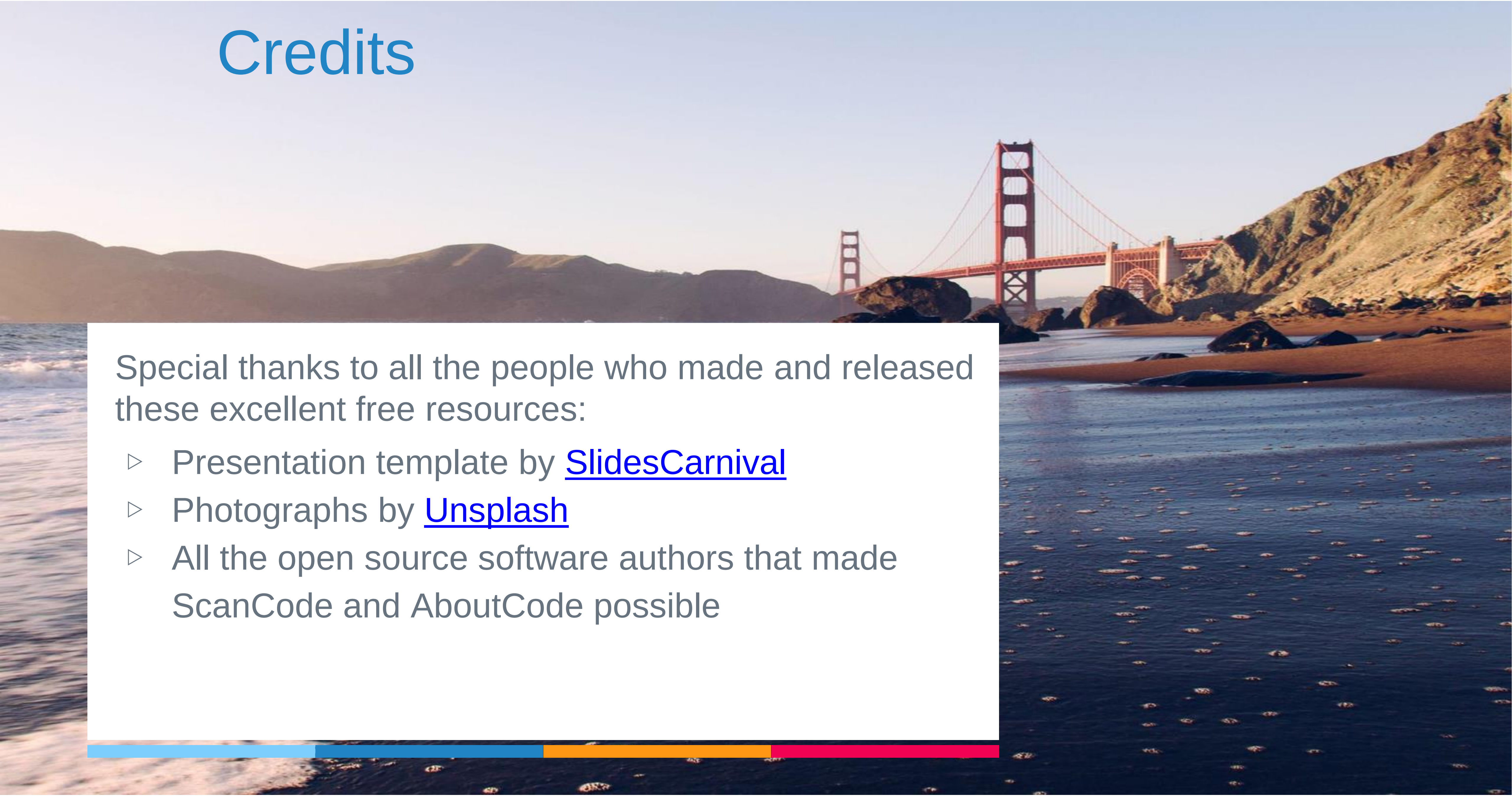
If you want to help

You can contribute code, time, docs (or cash?)

- Use these fine FOSS tools and specs
  - https://github.com/package-url
  - https://www.aboutcode.org/
  - https://github.com/nexB/
- Join the conversation at
  - https://gitter.im/aboutcode-org
- Donate at
  - https://opencollective.com/aboutcode

If you want to learn more about our projects

- On Package URL and Version Ranges (purl and vers):
  - Visit Purl https://github.com/package-url for the specs and libraries
  - Vers is at https://github.com/package-url/purl-spec/blob/version-range-spec/VERSION-RANGE-SPEC.rst
- On VulnerableCode usage of purl and vers
  - Register for our upcoming webinar on using purl with VulnerableCode on July 21 at https://nexb.com/0721-vulnerablecode
  - Read our latest blog post at https://nexb.com/vulnerablecode-public-release
  - Visit https://nexb.com/vulnerablecode for more information



Credits

Special thanks to all the people who made and released these excellent free resources:

- Presentation template by SlidesCarnival
- Photographs by Unsplash
- All the open source software authors that made ScanCode and AboutCode possible

Summary

No tech stack is an island running

- No single programming language
- No Single single package ecosystem

We need a way to talk about packages and their versions across ecosystems.

- purl and vers are an attempt to solve this problem and express package dependencies and vulnerabilities using a common language among multiple tools, SBOM formats and tech stacks.

We will present Package URL, a way to reference packages across ecosystems which is emerging as a de-facto standard identifier for open source software packages. And we will introduce a new universal notation for package version ranges, such as used when resolving package dependencies as in "I require package foo, version 2.0 or later versions" and referencing affected vulnerable package versions as in "vulnerability CVE-123 affects package bar, version 3.1 and version 4.2 but not version 5".

These two mini standards pave the way towards (mostly) universal FOSS package naming and versioning for dependency resolution and vulnerability ranges references; and are emerging as essential to reliably process vulnerability data in the software supply chain.

What is dependency resolution?

- ▷ Start with **package** you directly depend on
  - name & version (or version range)
- ▷ Access some **package** index or metadata source
  - collect all known versions of the package
- ▷ Select one **version** that matches the range you need
- ▷ For this version, collect the packages it **depends** on
  - names and versions (or version range)
- ▷ **Repeat recursively**
- ▷ Update selected **names and versions** as needed until consistent
  - Can be involved algos using sat solvers, backtracking, etc.
- ▷ ***Finally install these packages (not today's scope)***