
# Integrating Knowledge Graphs into the Debian Ecosystem

## KG-powered dependency, vulnerability & license insights

Alexander Belikov
GrowGraph

DebConf2025, Brest, France 2025-07-15

# Plan

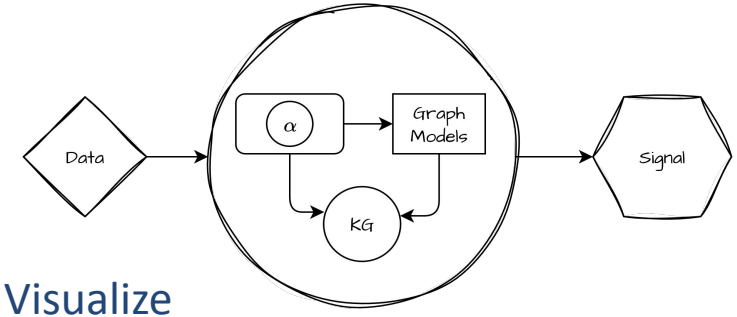
Knowledge Graphs: Motivation

Building DebKG

DebKG: it is already useful

Next Steps

# Why do we need DebKG?




**Problem**

Debian's ecosystem is a sprawling network of

- 70 000+ packages
- 50 000+ open bugs
- Thousands of maintainers

- Visualize
- Evaluate
- Predict

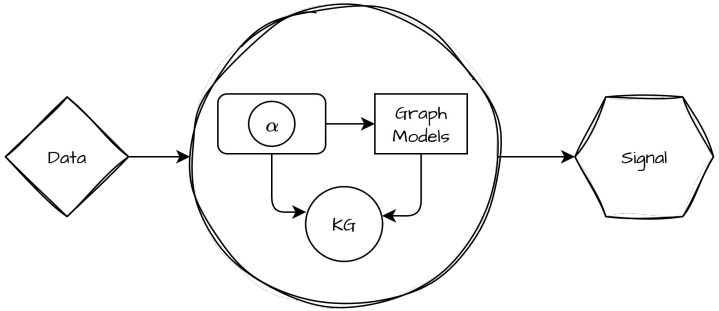
All stored in disparate files and trackers—making it hard to answer questions like "Which packages are impacted by CVE-2025-1234?" or "Which maintainers handle GPL-licensed libraries?"

**Why Knowledge Graphs?**

Knowledge graphs model entities (packages, bugs, maintainers) and their relationships (depends_on, affects, authored_by) as a single, unified graph.

**Intuitive**: Mirror the natural structure of Debian concepts
**Powerful**: Express complex queries (e.g., multi-hop dependency chains) in one line of Cypher
**Extensible**: Easily add new data sources (security feeds, license info, bug metadata)

# Why do we need DebKG?




**How DebKG Delivers**

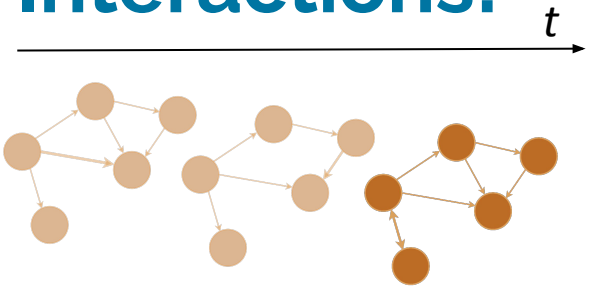
**Ingest**: Crawl Packages.gz, Debian Security Tracker, maintainers, copyright files
**Map**: Apply a Debian-specific ontology (classes, properties, constraints)
**Load**: Publish into a property-graph database (e.g., Neo4j or Dgraph)
**Demo**: Trace CVE impact across package trees in seconds

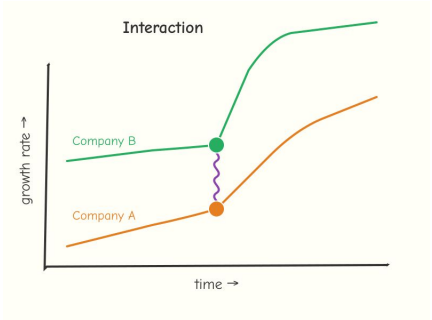
**Where We Go Next**

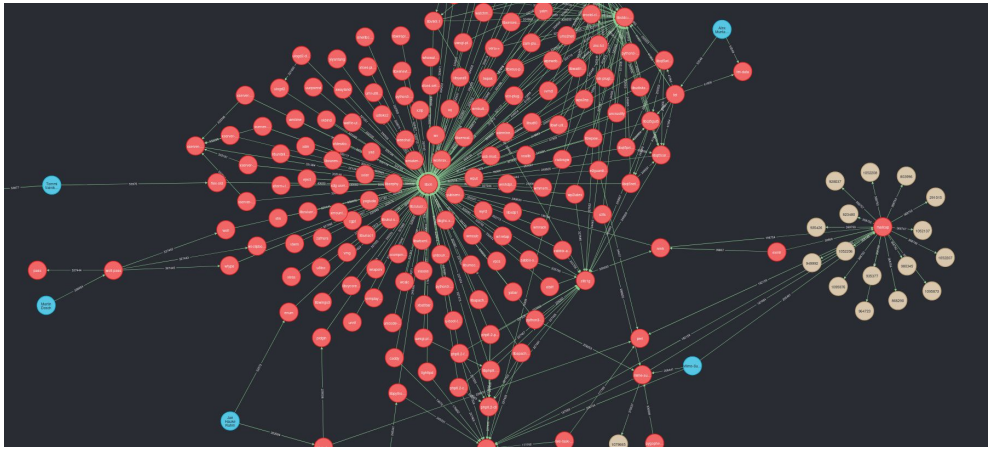
**Expand**: Add suite-based snapshots (bookworm, trixie, etc.)
**Enrich**: Incorporate external feeds (OSS license databases, vulnerability timelines)
**Build**: Community-driven Web UI and CLI toolkits for packagers, QA, and security teams
**Contribute**: Join us on https://github.com/alexander-belikov/deb-kg (soon on salsa) - help refine the ontology, improve ingest pipelines, or add new visualizations

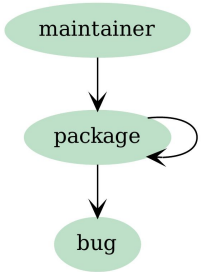
# Why do we need graphs?

## Interactions!





# What is a Knowledge Graph?

A **Knowledge Graph (KG)** is a **graph-based data structure** (*network*) where entities (nodes) and their relationships (edges) are explicitly represented.

It integrates **structured and unstructured data** into a unified, queryable format.

**Core Components**

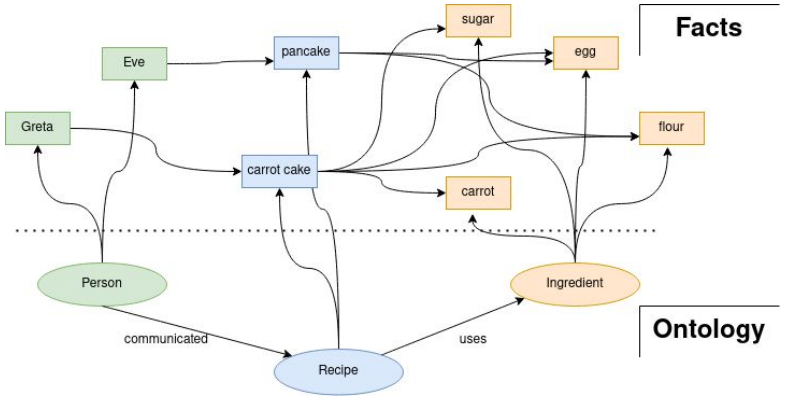
- **Entities**: e.g., *package*, *contributor*, *bug*
- **Relations**: e.g., *maintains*, *depends_on*, *reported_in*
- **Attributes**: properties of nodes/edges (e.g., text, severity, created)
- **Semantics**: often enhanced with ontologies or schemas for interpretability

**Why It Matters**

- Handles **heterogeneous, relational data** natively (e.g., linking maintainers, different types of relations between packages etc)
- Enables **non-tabular** (e.g., Graph Neural Networks, community detection)
- Captures **contextual and temporal dependencies** (which might be causal)

# Ontology: the backbone of KG

- Ontology: set of concepts and categories in a subject area or domain that shows their properties and the relations between them
  (something that experts agree upon)
  [classes, properties, axioms, constraints]
- It matters for precision: models get confused without an ontology (context)
- Allow for reification and semantic reasoning [validation]



# Benefits of KG

**Natural Fit for Software Development / Project Management**

- Packages, tools, maintainer, bugs  and **their relationships** are inherently graph-structured
- Easily integrates **heterogeneous sources** (Debian API, bug reports, etc)
- Flexible wrt to extension

**Augments Retrieval & Explainability**

- Powers **semantic search** and **entity-centric queries** (e.g., "Which maintainer would be best to help?")
- **Human-interpretable reasoning chains** support transparency & auditability

**Enhances Modeling**

- Improves **prediction tasks** (e.g., fraud detection, legal risk etc) via relational inductive bias
- Enables **Graph ML**: GNNs, link prediction, anomaly detection
- **Few-shot generalization** using structure (e.g., new firms with similar graph neighborhoods)

# Property Graphs vs Triple Stores

**Data Model**

*Property Graph*: Nodes and edges with key-value properties

*Triple Store*: RDF triples (subject–predicate–object)

**Schema Flexibility**

*Property Graph*: Flexible, schema-optional

*Triple Store*: Based on RDF/OWL; more structured

**Query Language**

*Property Graph*: Gremlin, Cypher

*Triple Store*: SPARQL

**Semantics & Reasoning**

*Property Graph*: Limited or custom logic, community detection modules

*Triple Store*: Built-in reasoning (RDFS/OWL)

**Use Cases**

*Property Graph*: Operational, traversal-heavy apps (e.g., fraud detection)

*Triple Store*: Knowledge representation, semantic web

**Standards Compliance**

*Property Graph*: No formal standards

*Triple Store*: W3C standards (RDF, SPARQL)

**Performance**

*Property Graph*: Optimized for deep traversals

*Triple Store*: Optimized for pattern matching and inferencing

```
@prefix sdo: <https://schema.org/>.

<https://example.com/person/123> a sdo:Person ;
    sdo:name "John Doe" ;
    sdo:birthDate "1980-01-01" ;
    sdo:email "johndoe@example.com" ;
```

# Graph Databases

| Database | License | Query | Horizontal Scaling | Comment |
|---|---|---|---|---|
| JanusGraph | Apache 2.0 | Gremlin | Native | Best for massive scales |
| Memgraph | BSL | Cypher | Enterprise only | In-memory focus |
| Neo4j | GPLv3 | Cypher | Enterprise only | Mature Ecosystem |
| ArangoDB | BSL | AQL | Enterprise only | Close to noSQL |

# Triple Stores

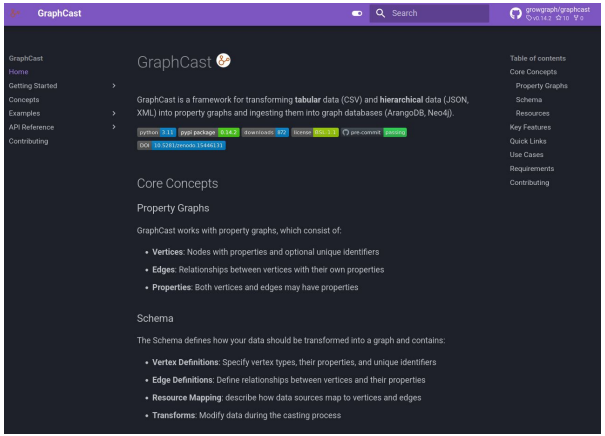
| Database | License | Query Language | Horizontal Scaling |
|---|---|---|---|
| Jena Fuseki | Apache 2.0 | SPARQL | Limited |
| Virtuoso | GPL/Commercial | SPARQL | Limited for Community |
| GraphDB | Commercial | SPARQL | Only for Enterprise |

https://growgraph.github.io/graphcast

https://growgraph.github.io/ontocast

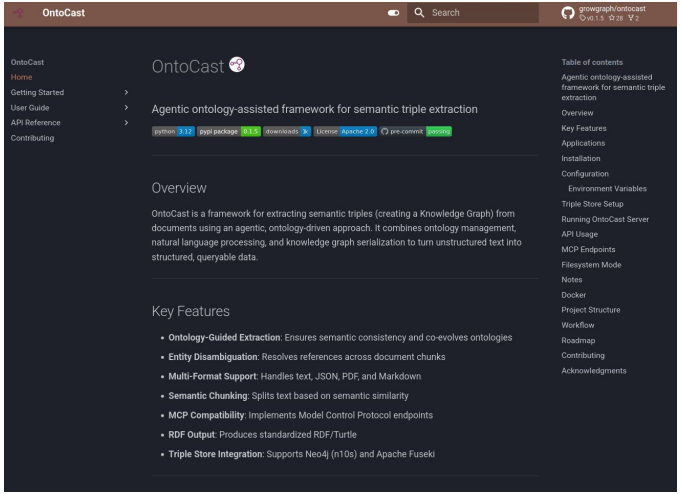
# Graphcast

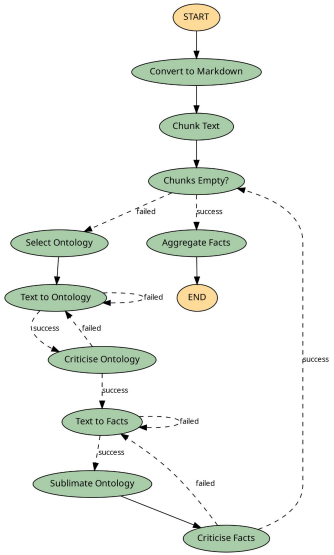
## Declarative Transformations of data to Property graphs




# Ontocast

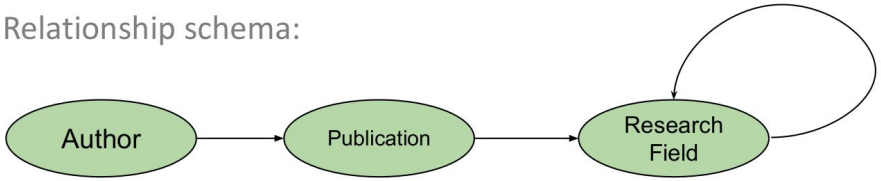
## Ontology Assisted Agentic Framework for Semantic Triple Extraction



# [Example] KG in Publishing

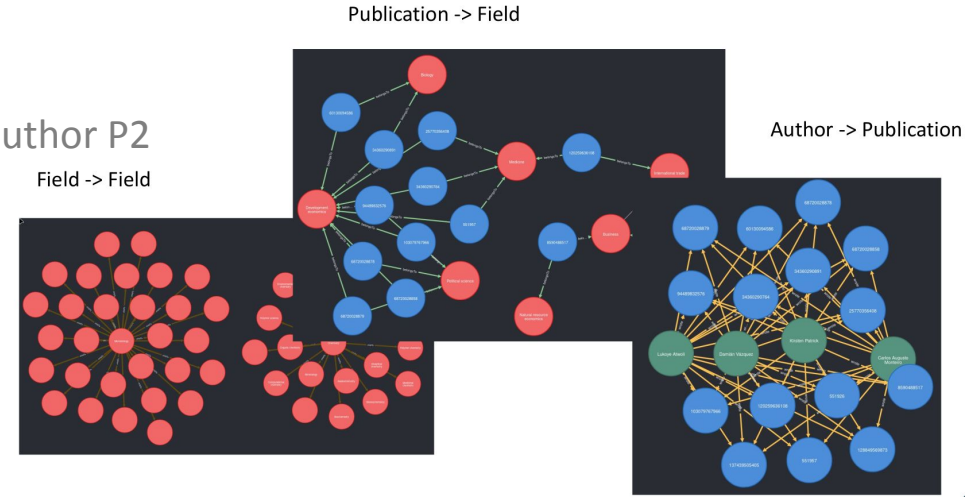
Relationship schema:




Explicit vs implicit interactions:
X authors paper P
X and Y: X and Z co-author P1, Y and Z co-author P2

Problems

1. Author disambiguation
2. Reviewer recommendation
   a. identify reviewers that publish in related fields
   b. remove such reviewers that belong the same collaboratives communities

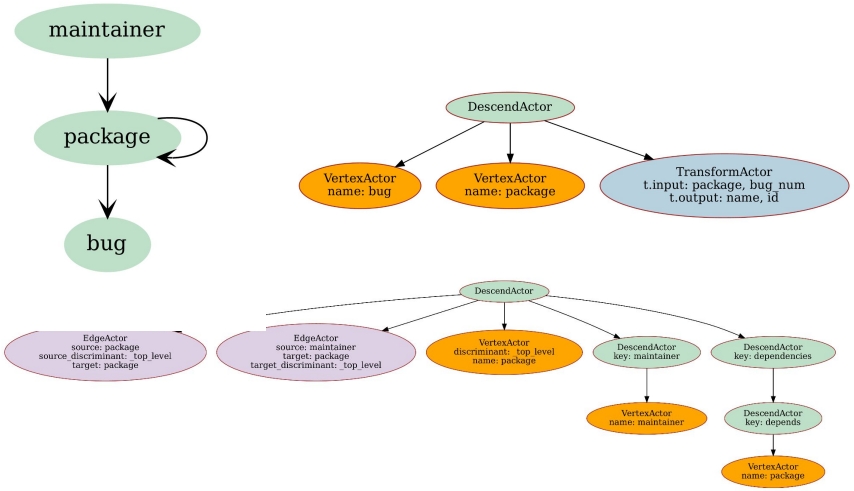


# DebKG (0.0.1)

```yaml
general:
    name: debian-eco
    version: 0.0.1
resources:
-   resource_name: package
    apply:
    -   vertex: package
        discriminant: _top_level
    -   key: dependencies
        apply:
        -   key: depends
            apply:
            -   vertex: package
    -   source: maintainer
        target: package
        target_discriminant: _top_level
    -   source: package
        target: package
        source_discriminant: _top_level
    -   key: maintainer
        apply:
        -   vertex: maintainer
-   resource_name: bug
    apply:
        - map:
            package : name
            bug_num: id
        -   vertex: package
        -   vertex: bug
vertex_config:
    vertices:
    -   name: package
        fields:
        -   name
        -   version
        indexes:
        -   fields:
            -   name
    -   name: maintainer
        fields:
        -   name
        -   email
        indexes:
        -   fields:
            -   email
    -   name: bug
        fields:
        -   id
        -   subject
        -   severity
        -   date
        indexes:
            - fields:
                - id
edge_config:
    edges:
    -   source: package
        target: package
    -   source: maintainer
        target: package
    -   source: package
        target: bug
```

Sources:
http://deb.debian.org/debian/dists/bookworm/main/binary-amd64/Packages.gz
https://www.debian.org/bugs/

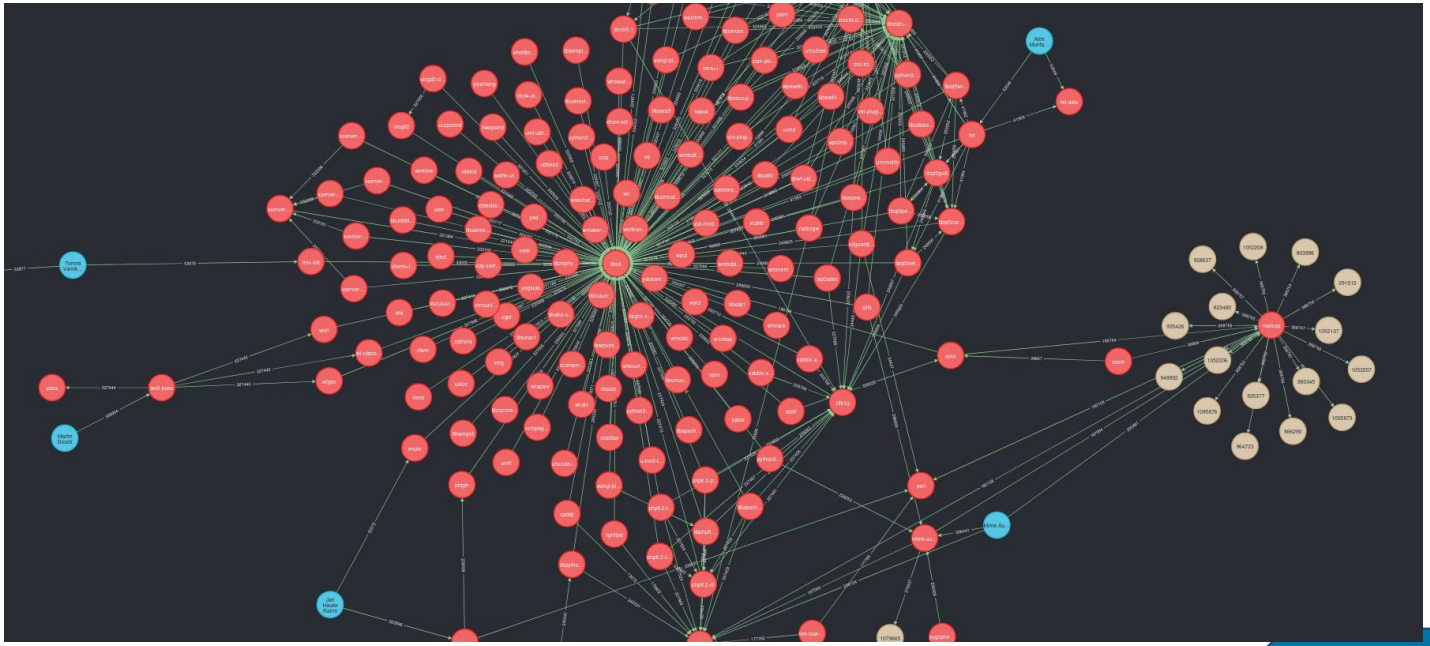
Raw Data ➤ GraphCast ➤ Neo4j



# DebKG

Snapshot: bookworm

packages: 67967
maintainers: 2122
bugs: 51690



# Use cases

- Downstream vulnerability: (P) ➢ (P), (P) ➢ (B)
  Evaluate global effect of vulnerabilities for prioritization
- License audit: (P) ➢ (P), (P) ➢ (L)
  Evaluate compliance / compatibility risks
- Maintainer expertise: (P) ➢ (P), (M) ➢ (P)
  Evaluate maintainer rescue potential
- Connect external data sources (customer feedback) to drive Debian innovation

What do we want to achieve? Better coverage, faster delivery, resilience?

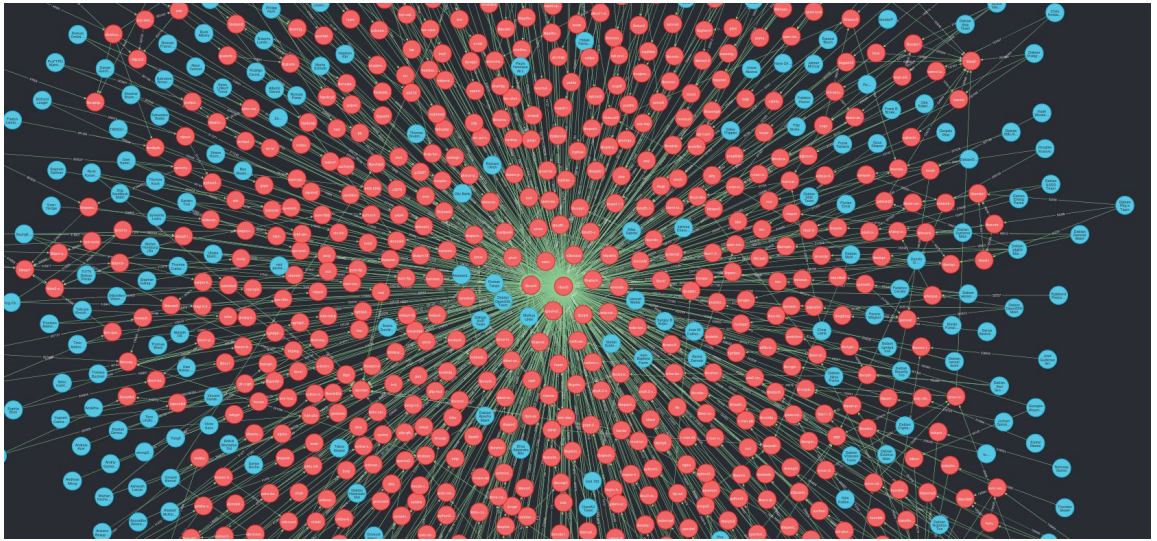
# Vulnerability Propagation

**How to measure the global effect of a bug?**

Suppose **libssl3** has a bug…

How does it affect packages/teams downstream? 340 maintainers, 1500+ packages (!)

| contributor | Packages Affected |
|---|---|
| Mattias Ellert | 51 |
| Debian Security Tools | 27 |
| Debian QA Group | 23 |
| Debian Perl Group | 14 |
| Debian VoIP Team | 13 |
| Debian MySQL Maintainers | 13 |
| Debian OpenStack | 12 |
| Debian HPC Team | 12 |
| Laszlo Boszormenyi (GCS) | 10 |
| Debian Cyrus Team | 10 |
| Debian Med Packaging Team | 9 |
| Sam Hartman | 9 |
| Debian Authentication Maintai | 8 |



# License Tracking / Alignment

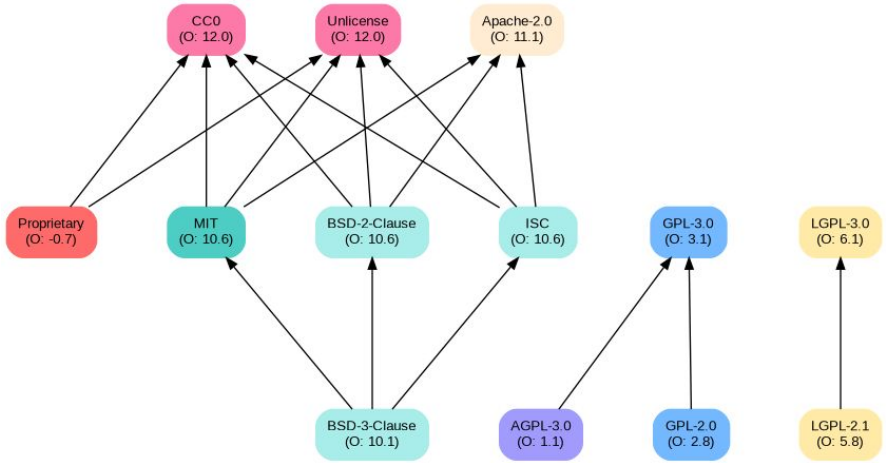
***Proposed Pipeline***

Legal obligations, compatibility issues, redistribution risks etc (copyleft spread / infection)…

Degrees of freedom:
Freedom to use, to modify etc
Endorsement prohibition,
Trademark protection

Represent as Hasse diagram (lattice)
Incorporate smaller licenses

Implementation: copyright parsing, e.g.
https://sources.debian.org/src/curl/8.15.0~rc3-1~exp1/debian/copyright/



# Next Steps

- More metadata
  - Treat package versions (automated version ordering)
  - Fine-grained maintainers
  - Impact weighed by importance (downloads)
- Semantic similarity analysis: similar function, similar implementations
- Evolution
  - Suite-based time serie of KG
  - History of updates
- External data: to steer development effort
- Online KG enabled intelligence platform [resource allocation recommendations]

# Conclusion

- KG graphs are flexible
- KG easily unlock a multitude of resource allocation insights
- Graphs reveal long-range (multi-hop) interactions and are indispensable for detecting collective modes, deviations from equilibrium
- DebKG can already be used for evaluation of vulnerability impact

**Thank you!**