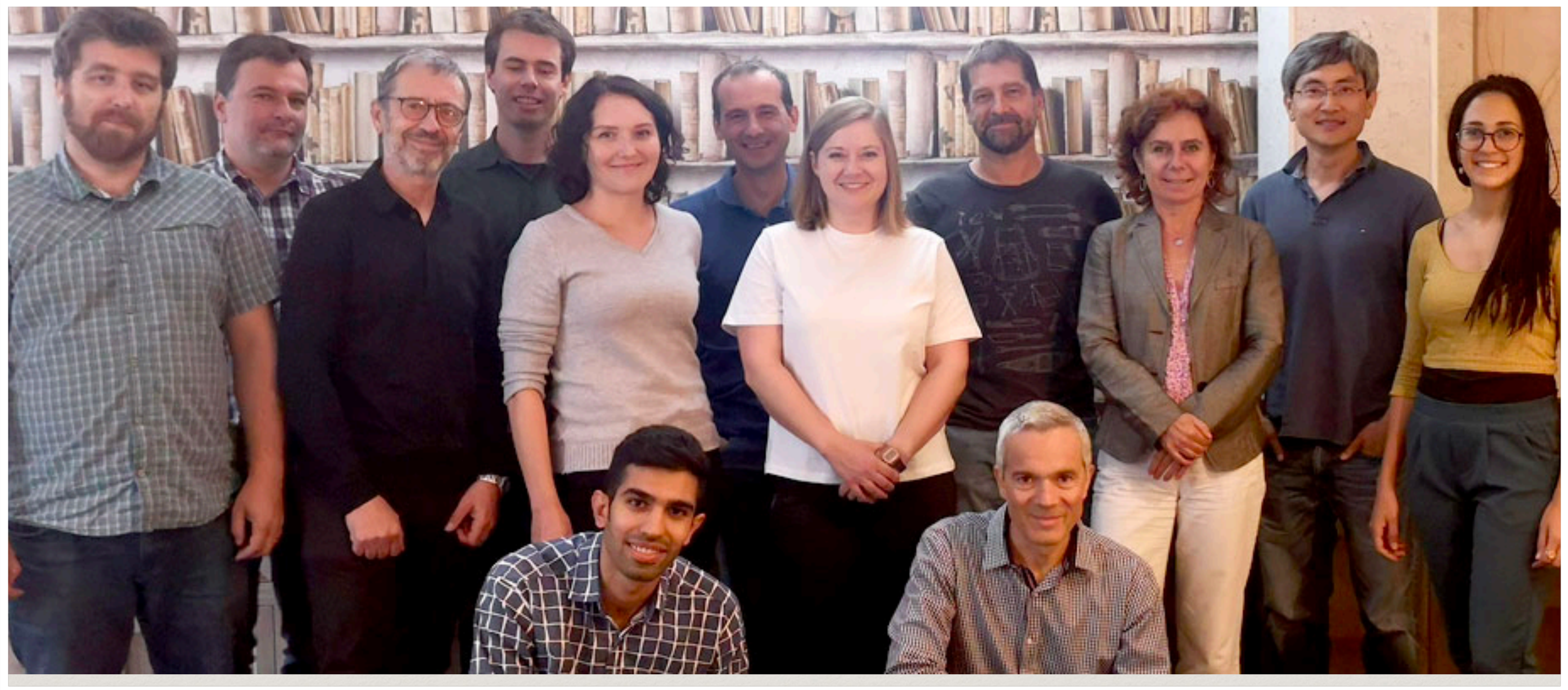


Software Ecosystems as Networks

Advances on the FASTEN project

Paolo Boldi
Università degli Studi di Milano
Italy

The FASTEN Project

- **F**ine-Grained **A**nalysis of **S**of**T**ware **E**cosystems as **N**etworks
- Part of the EU H2020-ICT-2018-2020 Program
- Consortium

Why FASTEN?

Sharing through software libraries

Sharing through software libraries

- Internet made the **dream** of collaborative development a **reality**, by means of libraries that are made available:

Sharing through software libraries

- Internet made the **dream** of collaborative development a **reality**, by means of libraries that are made available:
  - on *repositories* (SourceForge, GitHub, BitBucket, …)

Sharing through software libraries

- Internet made the **dream** of collaborative development a **reality**, by means of libraries that are made available:
  - on *repositories* (SourceForge, GitHub, BitBucket, …)
  - or *forges* (Maven, PyPi, CPAN, …)

Industrial revolution

at the harbour of software development

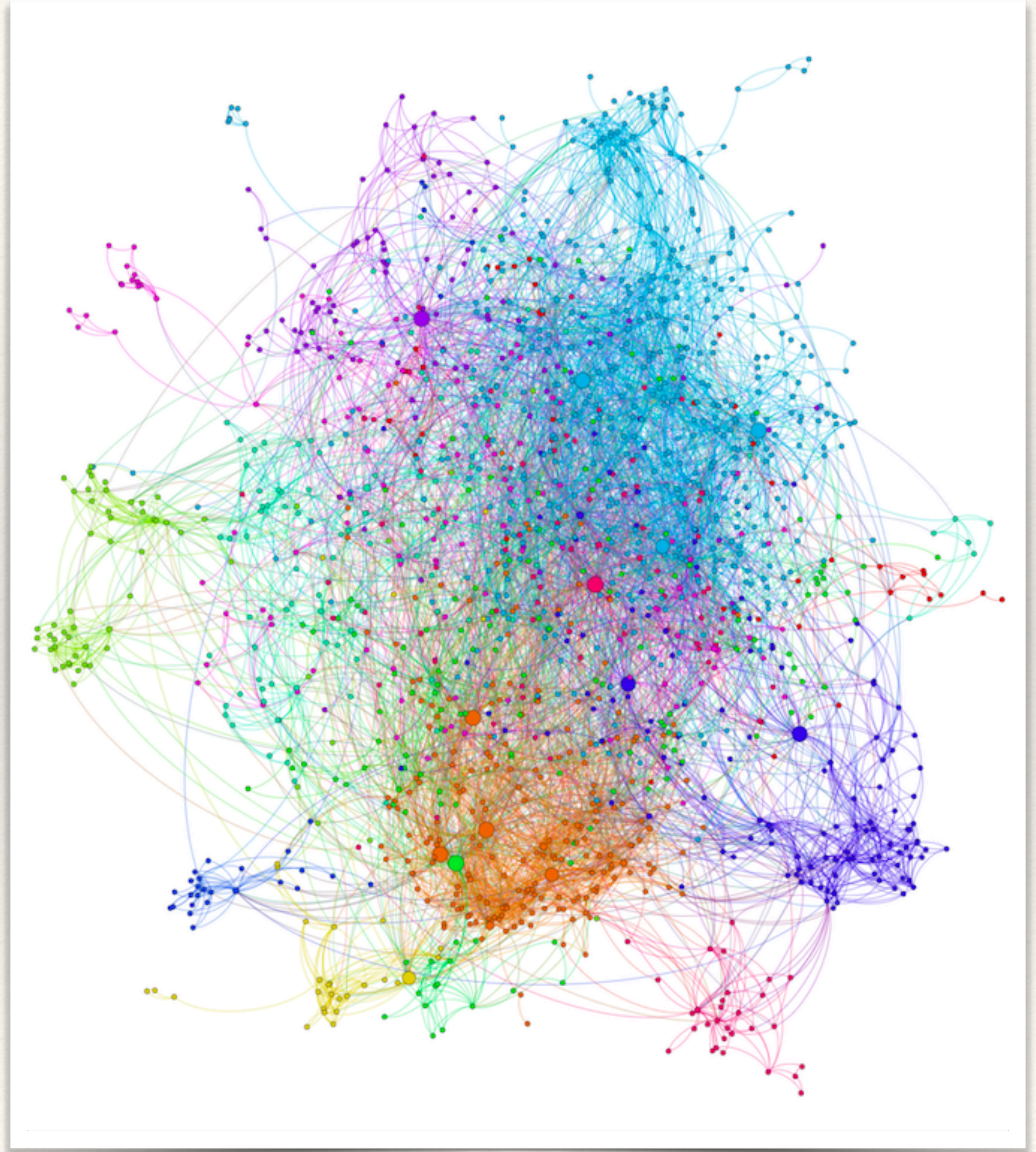
Industrial revolution

at the harbour of software development

- All trades, arts, and handiworks have gained by **division of labour**, namely, when, instead of one man doing everything, each confines himself to a certain kind of work distinct from others in the treatment it requires, so as to be able to perform it with greater facility and in the greatest perfection. Where the different kinds of work are not distinguished and divided, where everyone is a jack-of-all-trades, there manufactures remain still in the greatest barbarism.

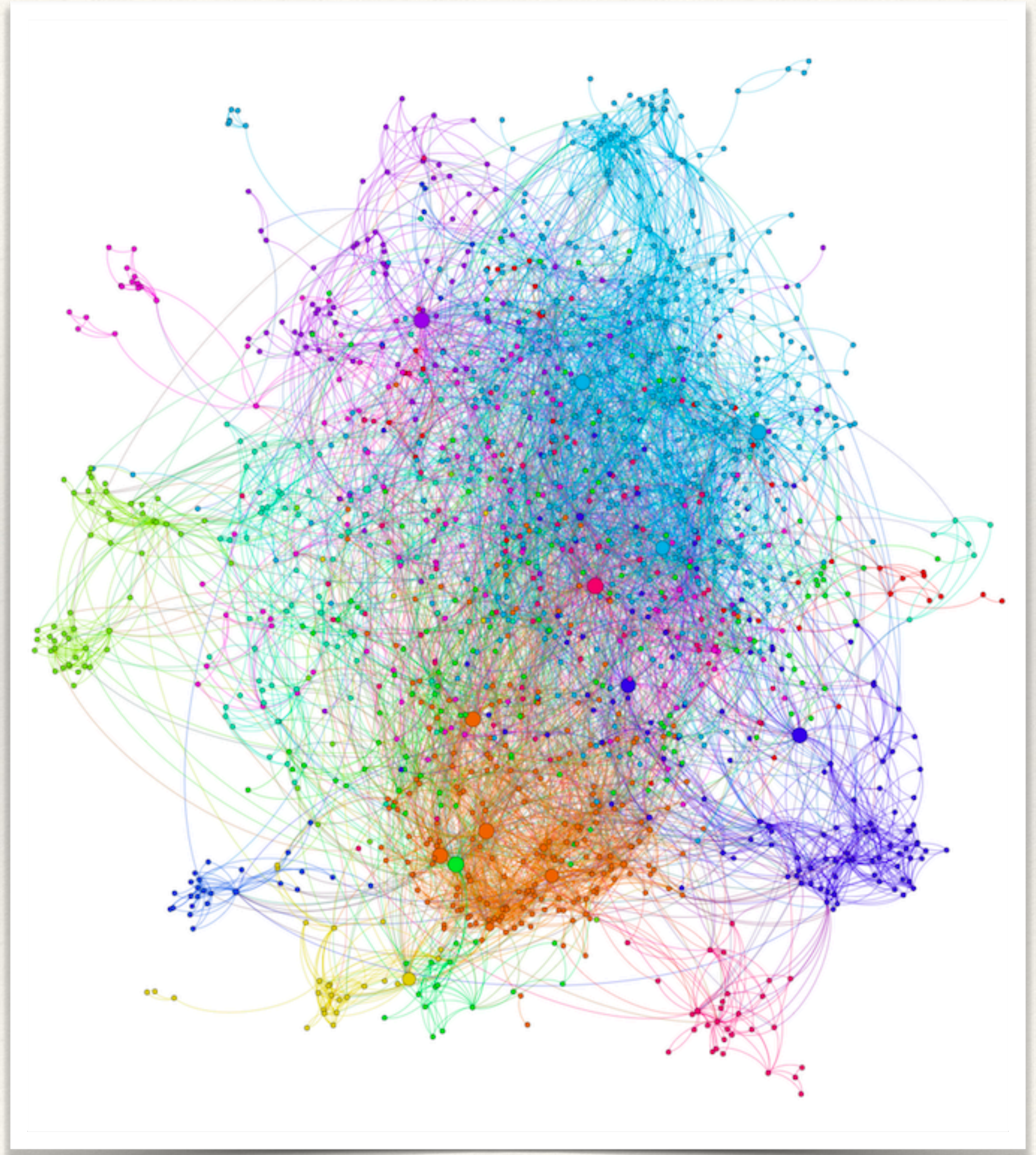
Immanuel Kant
Groundwork for the Metaphysics of Morals (1785)

Dependency graphs



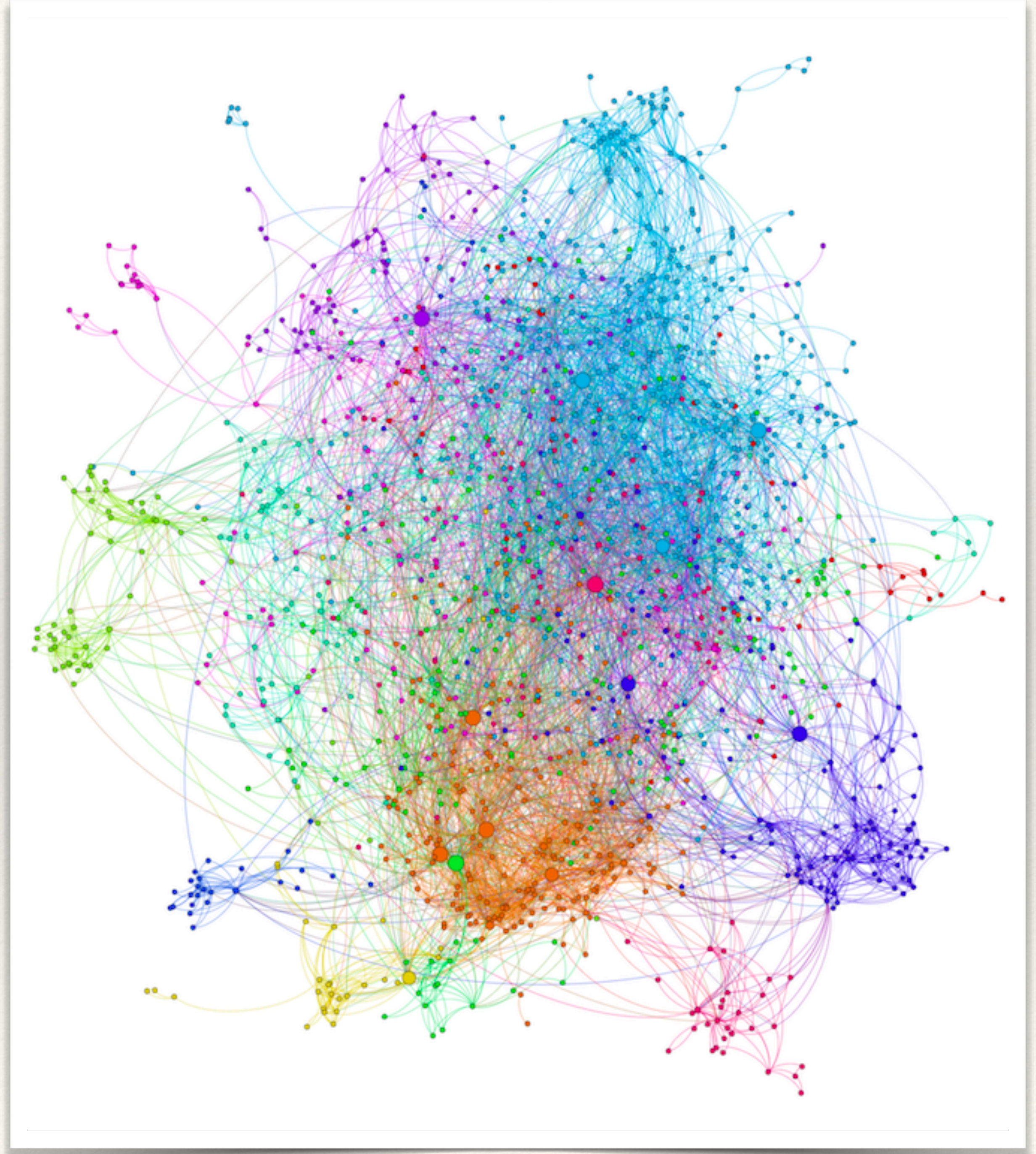
Dependency graphs

- Library+versions and their dependencies form (complex, huge) dependency networks



Dependency graphs

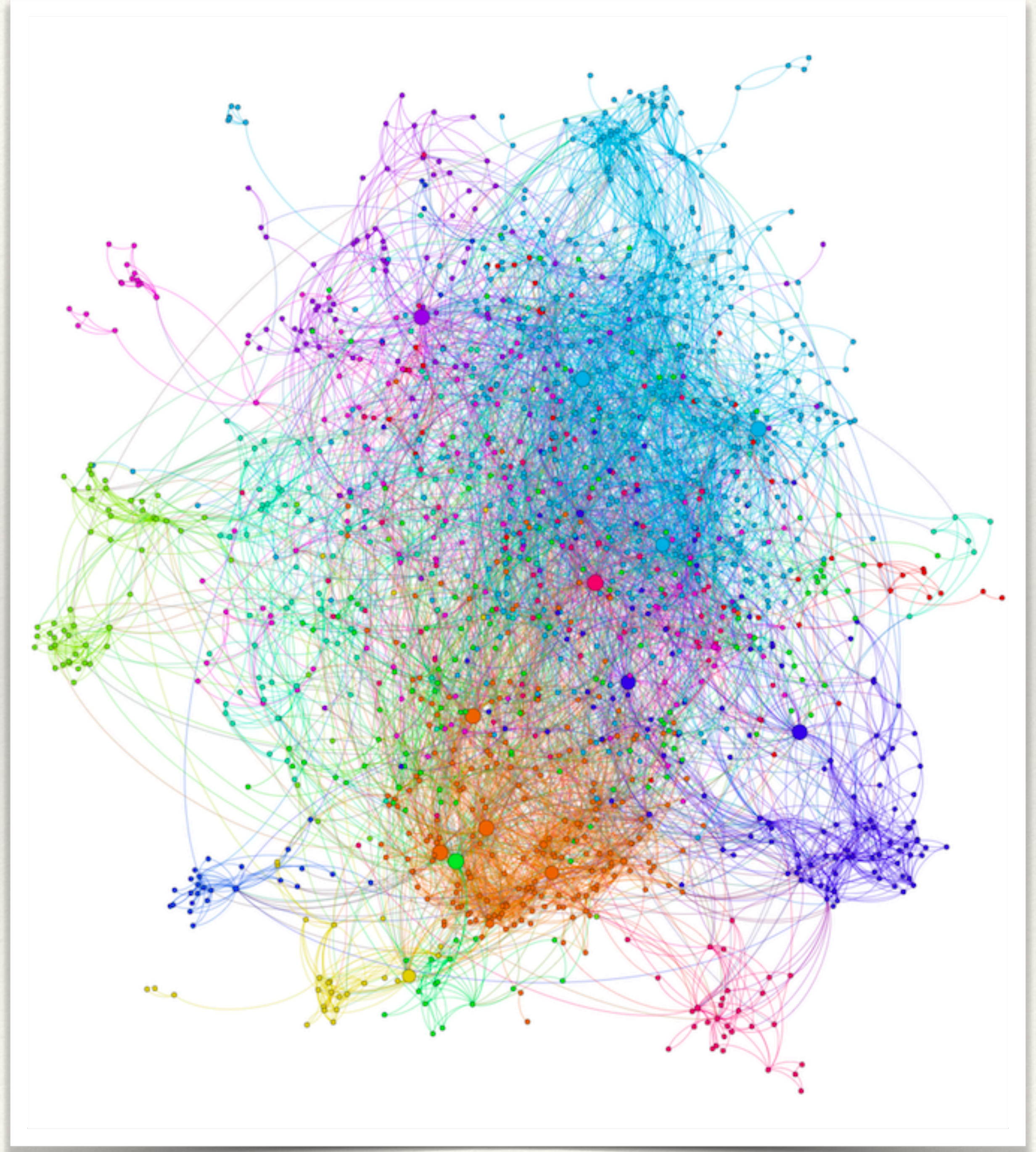
- Library+versions and their dependencies form (complex, huge) dependency networks
- Version constraints make these networks more complicated than simple *graphs*



Dependency graphs

FASTEN

- Library+versions and their dependencies form (complex, huge) dependency networks
- Version constraints make these networks more complicated than simple *graphs*
- Package manager will finally determine which version is chosen for each library



The dependency heaven

FASTEN

The dependency heaven

- Relying on an ecosystem of easy-to-use well written libraries made the dream of code reuse a reality

The dependency hell

The dependency hell

- A bug or security breach or legal issue concerning one single piece…
- …can make the whole tower fall!

Recent dependency nightmares

Recent dependency nightmares

- The `leftpad` incident (2016): millions of websites affected

Recent dependency nightmares

- The `leftpad` incident (2016): millions of websites affected
- The Equifax breach (2017): costed 4B$

Ecosystems

Ecosystems

- Ecosystems **grow** at mind boggling speed

Ecosystems

- Ecosystems **grow** at mind boggling speed
  - JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies

Ecosystems

- Ecosystems **grow** at mind boggling speed
  - JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
  - 50% of dependencies change in a 6-month time (Hejderup et al., 2019)

Ecosystems

- Ecosystems **grow** at mind boggling speed
  - JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
  - 50% of dependencies change in a 6-month time (Hejderup et al., 2019)
- And **deteriorate** almost as rapidly

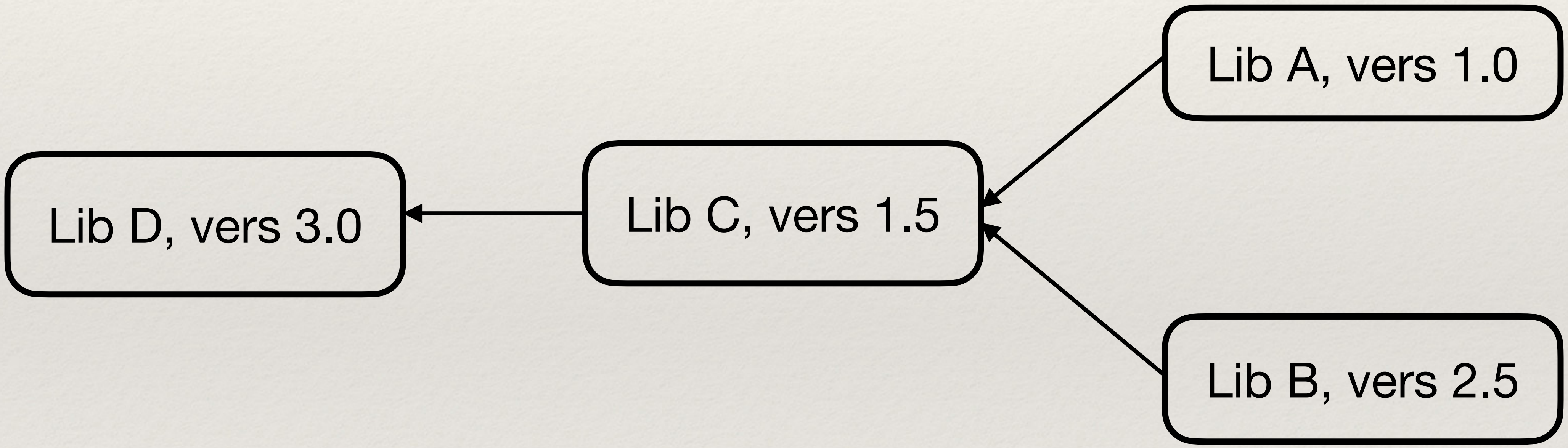
Ecosystems

- Ecosystems **grow** at mind boggling speed
  - JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
  - 50% of dependencies change in a 6-month time (Hejderup et al., 2019)
- And **deteriorate** almost as rapidly
  - Existence of package bottlenecks (the removal on one single package can bring down almost 40% of the system)

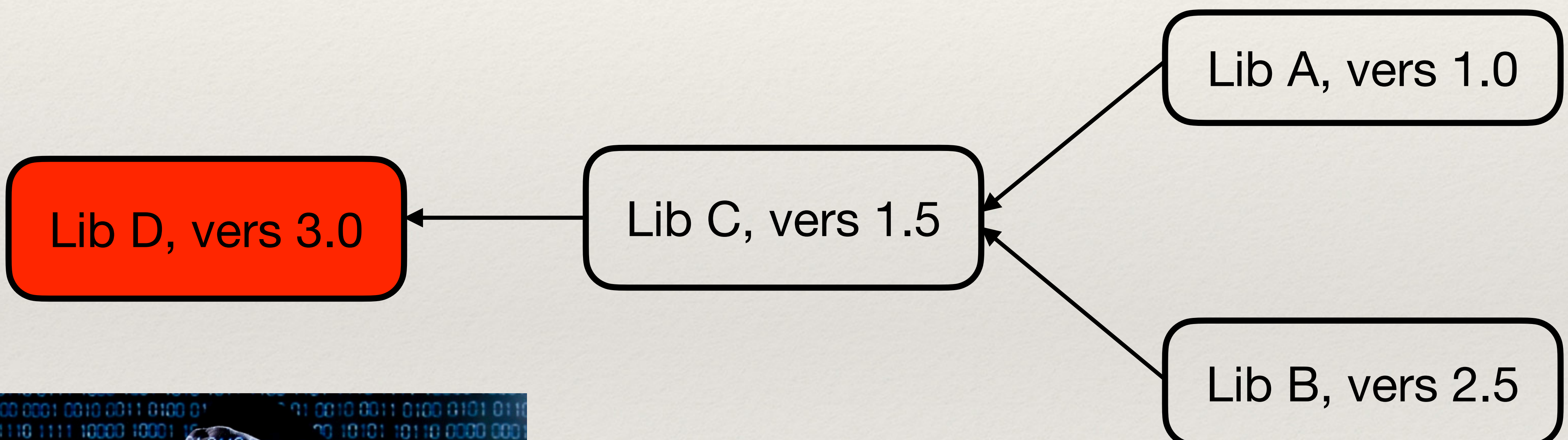
Ecosystems

- Ecosystems **grow** at mind boggling speed
  - JavaScript projects have an average of 80 (Zimmerman et al., 2019) transitive dependencies
  - 50% of dependencies change in a 6-month time (Hejderup et al., 2019)
- And **deteriorate** almost as rapidly
  - Existence of package bottlenecks (the removal on one single package can bring down almost 40% of the system)
  - Rich get richer: few maintainers dominate most packages

Epidemics in dependency graphs

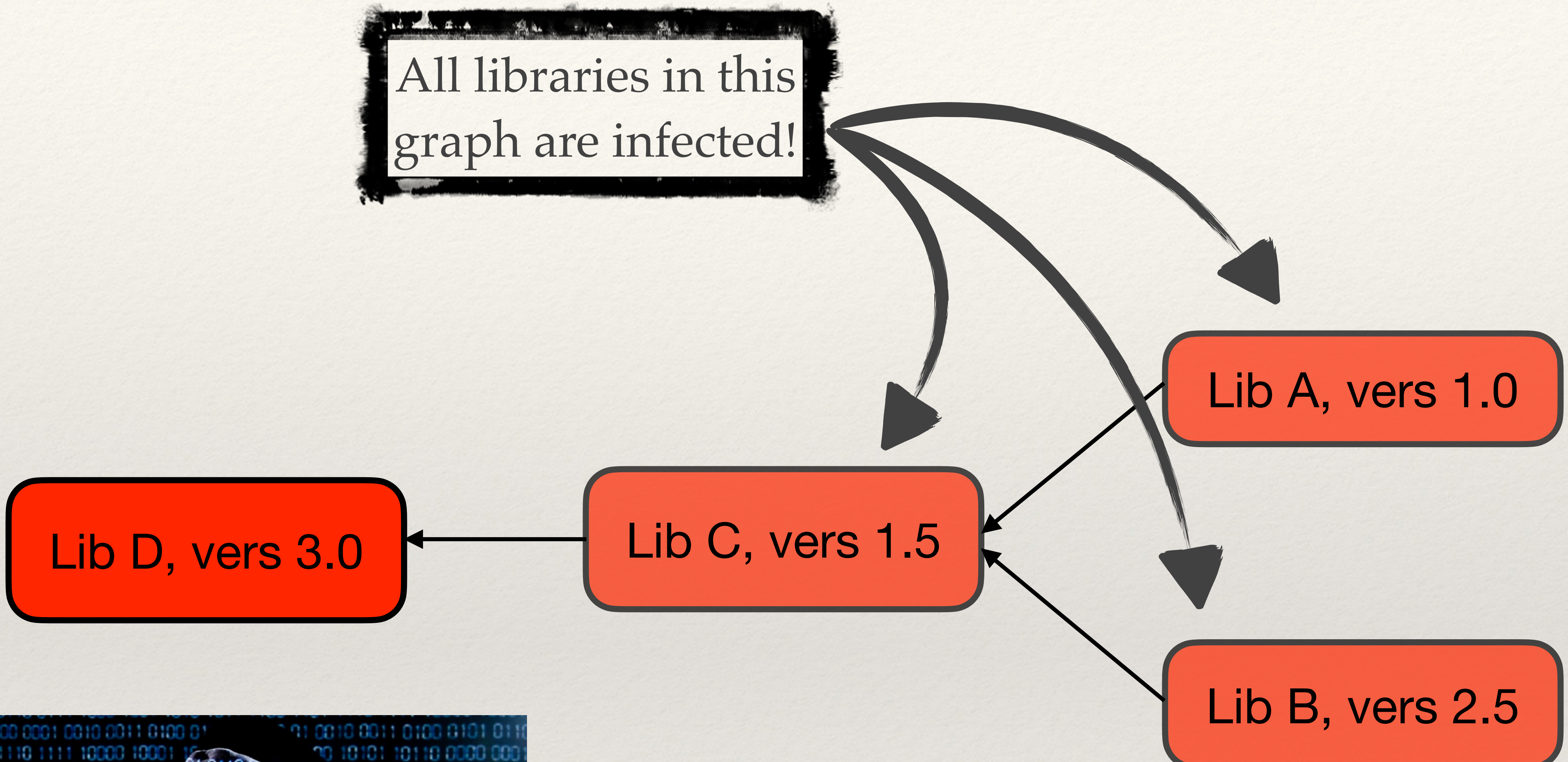


Epidemics in dependency graphs



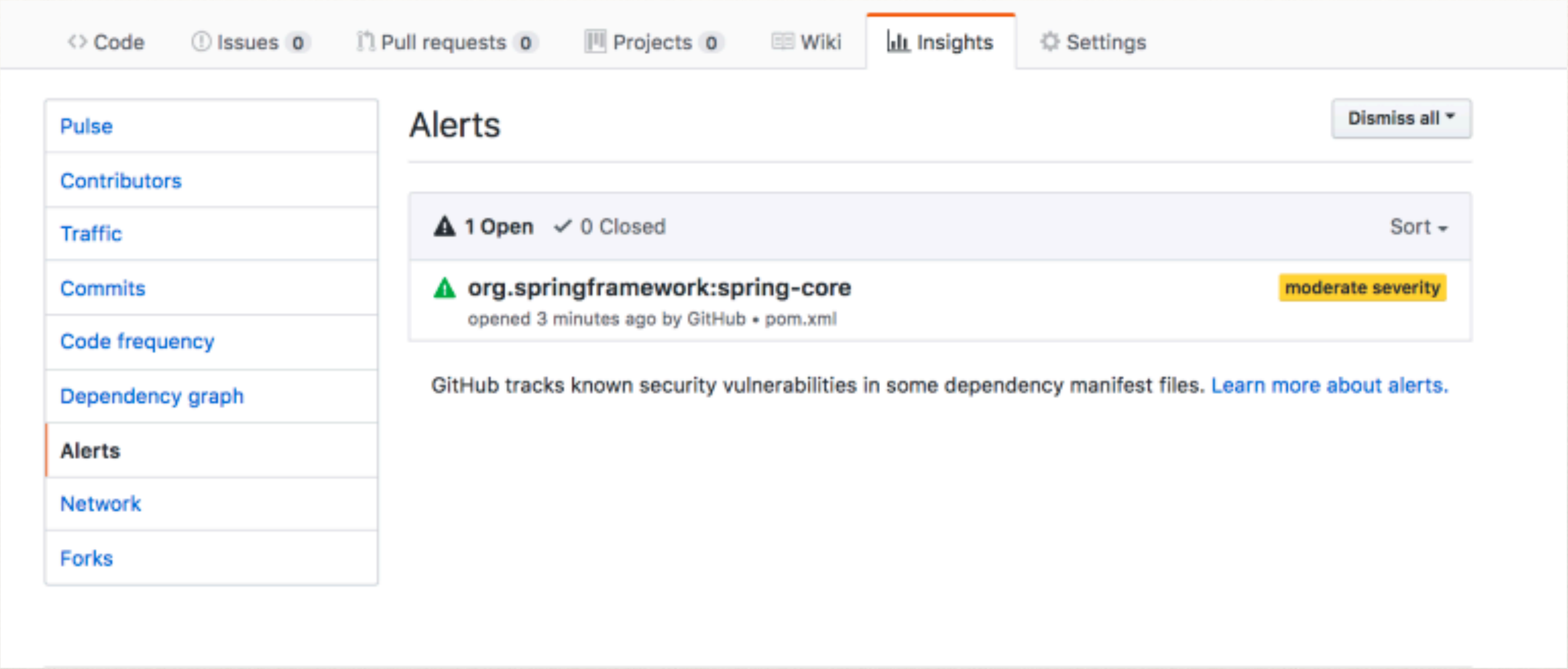
A vulnerability alert
is issued
about Lib D, vers 3.0

Epidemics in dependency graphs



A vulnerability alert
is issued
about Lib D, vers 3.0

GitHub security alerts



But is this enough?

Isn't this kind of tool enough?

Isn't this kind of tool enough?

- In theory. But in practice:

Isn't this kind of tool enough?

- In theory. But in practice:
    - Developers don't update

Isn’t this kind of tool enough?

- In theory. But in practice:
  - Developers don’t update
  - → Vulnerabilities proliferate

Isn't this kind of tool enough?

- In theory. But in practice:
  - Developers don't update
  - → Vulnerabilities proliferate
- Why?

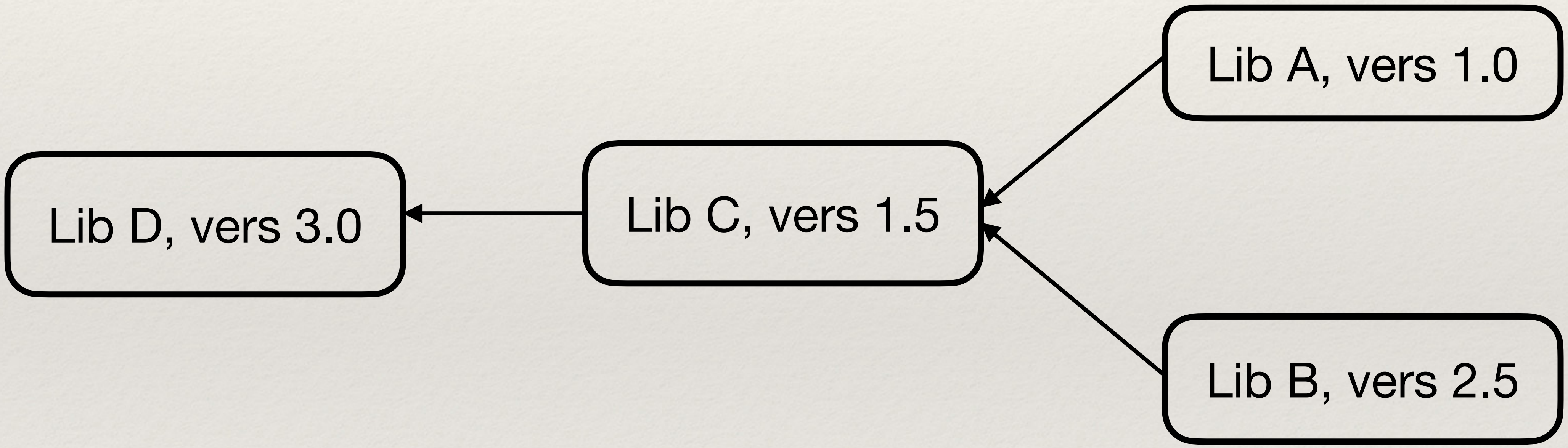
Isn't this kind of tool enough?

- In theory. But in practice:
  - Developers don't update
  - → Vulnerabilities proliferate
- Why?
  - Our tools are not sharp enough for what we want

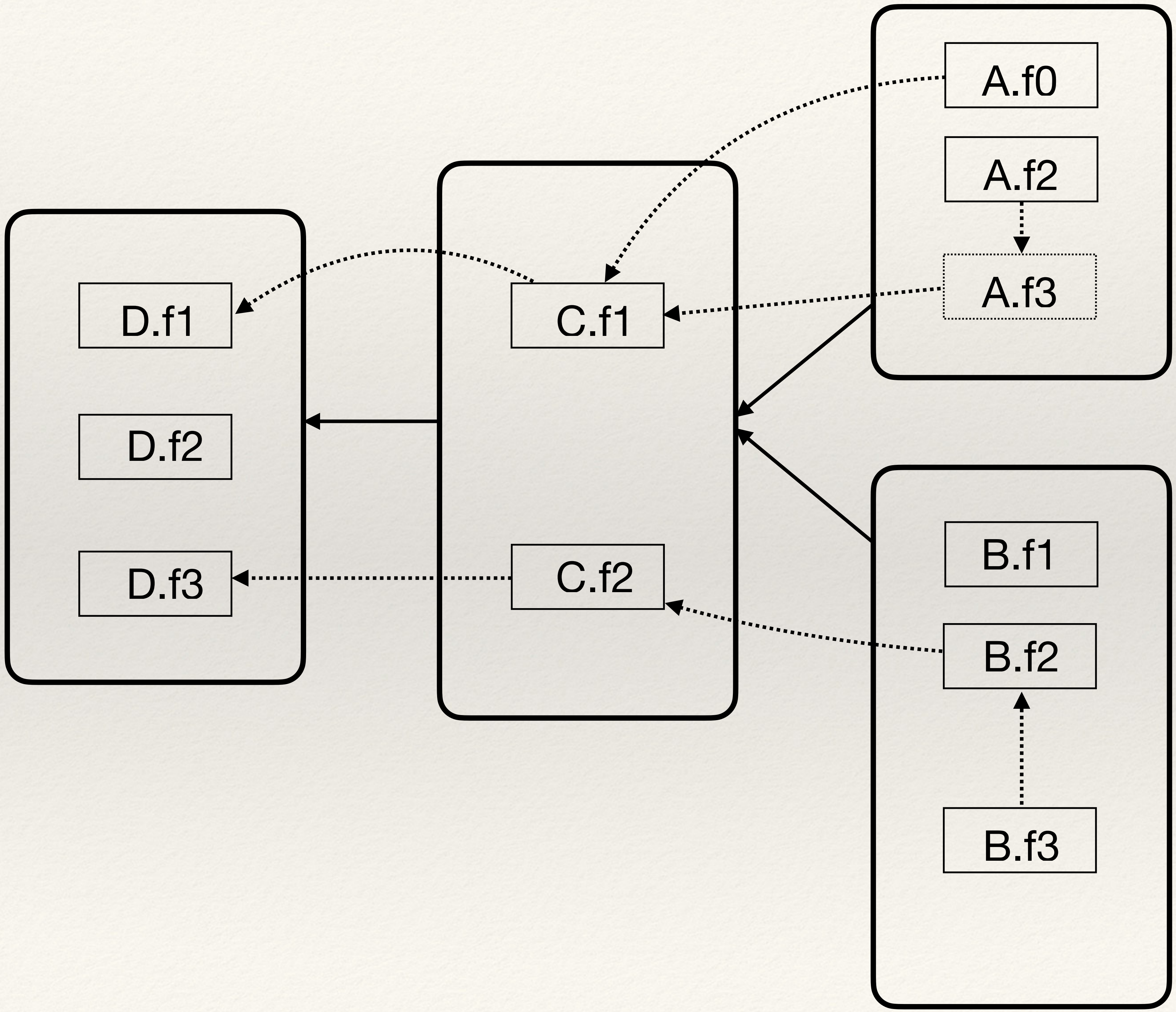
Examples of what people want

	Developers	Maintainers
Update	Does this outdated dependency *really* break my code?	How do I update *without breaking* too many of my important clients?
Violations	Am I violating anyone's *copyright*?	How do I spot instances of my code being distributed *without permission*?

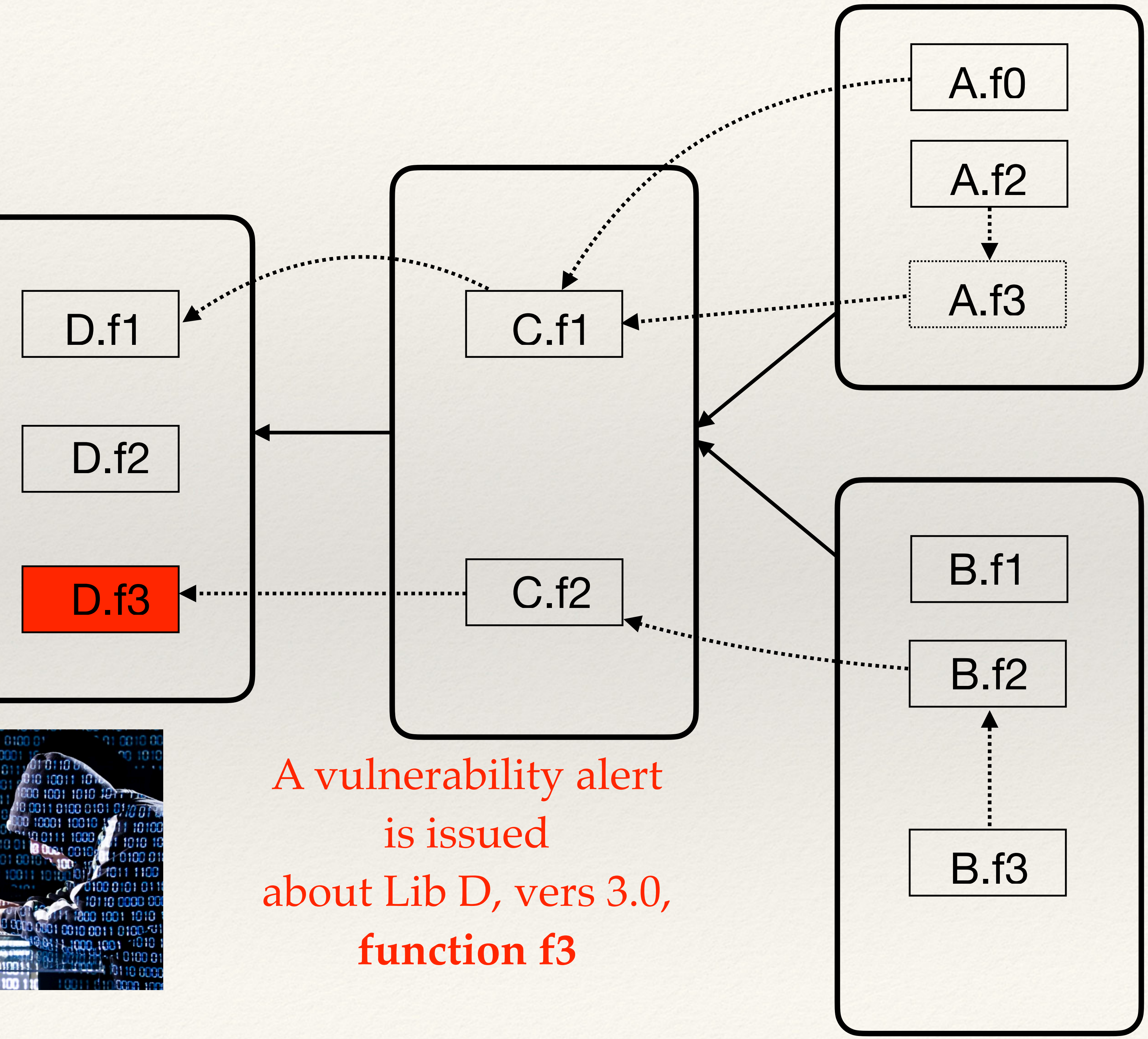
Epidemics in dependency graphs



Epidemics in dependency graphs

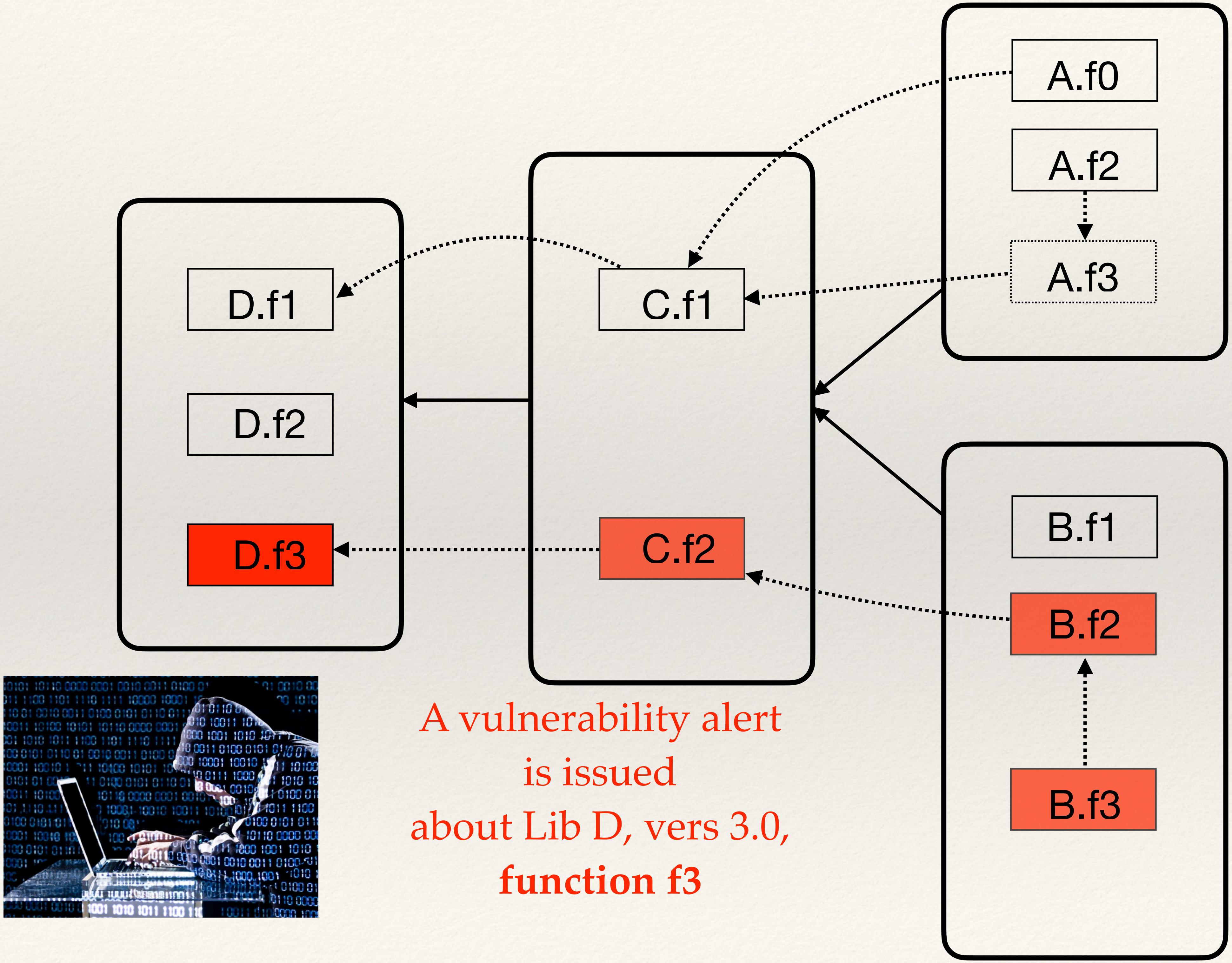


Epidemics in dependency graphs



A vulnerability alert
is issued
about Lib D, vers 3.0,
function f3

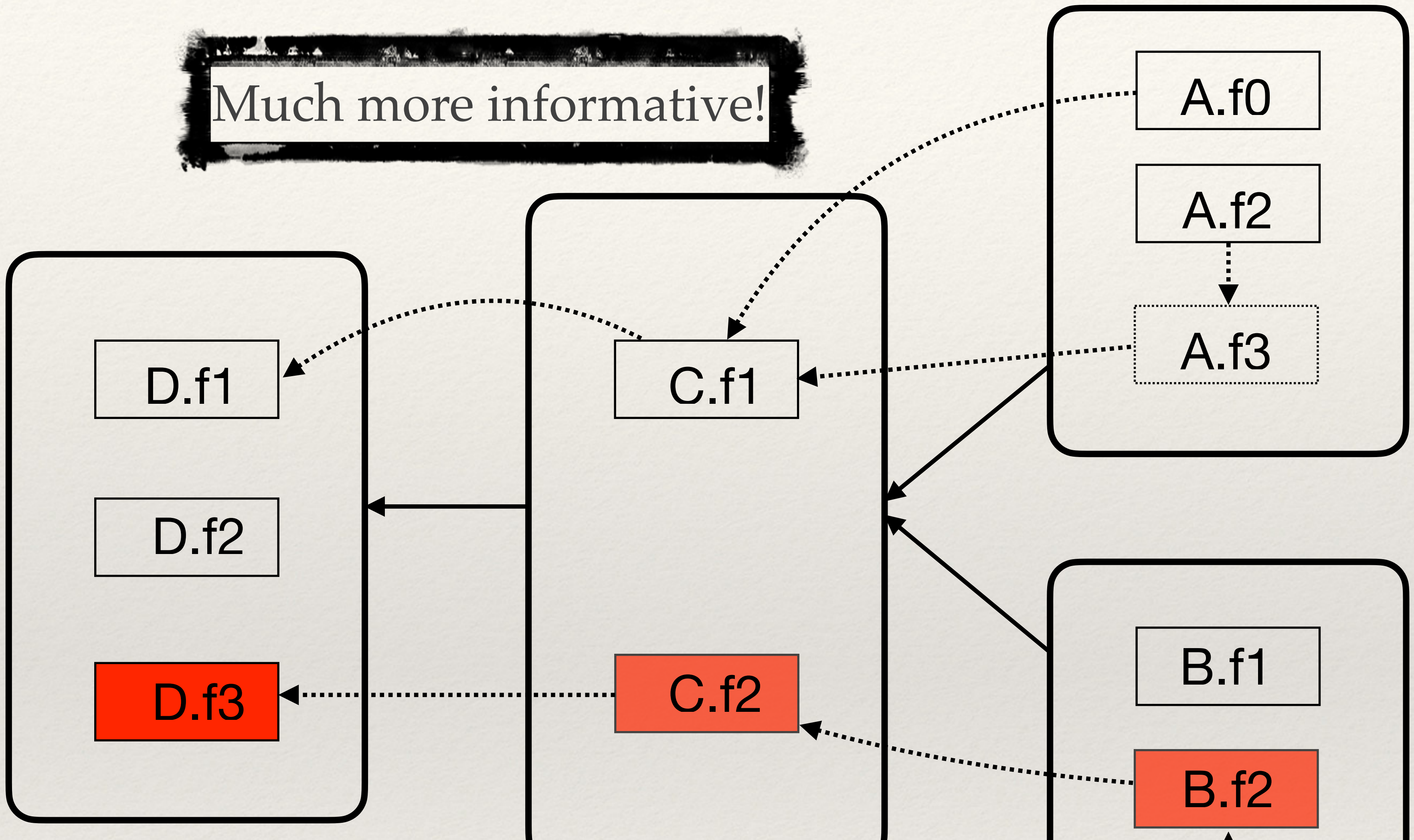
Epidemics in dependency graphs



A vulnerability alert
is issued
about Lib D, vers 3.0,
function f3

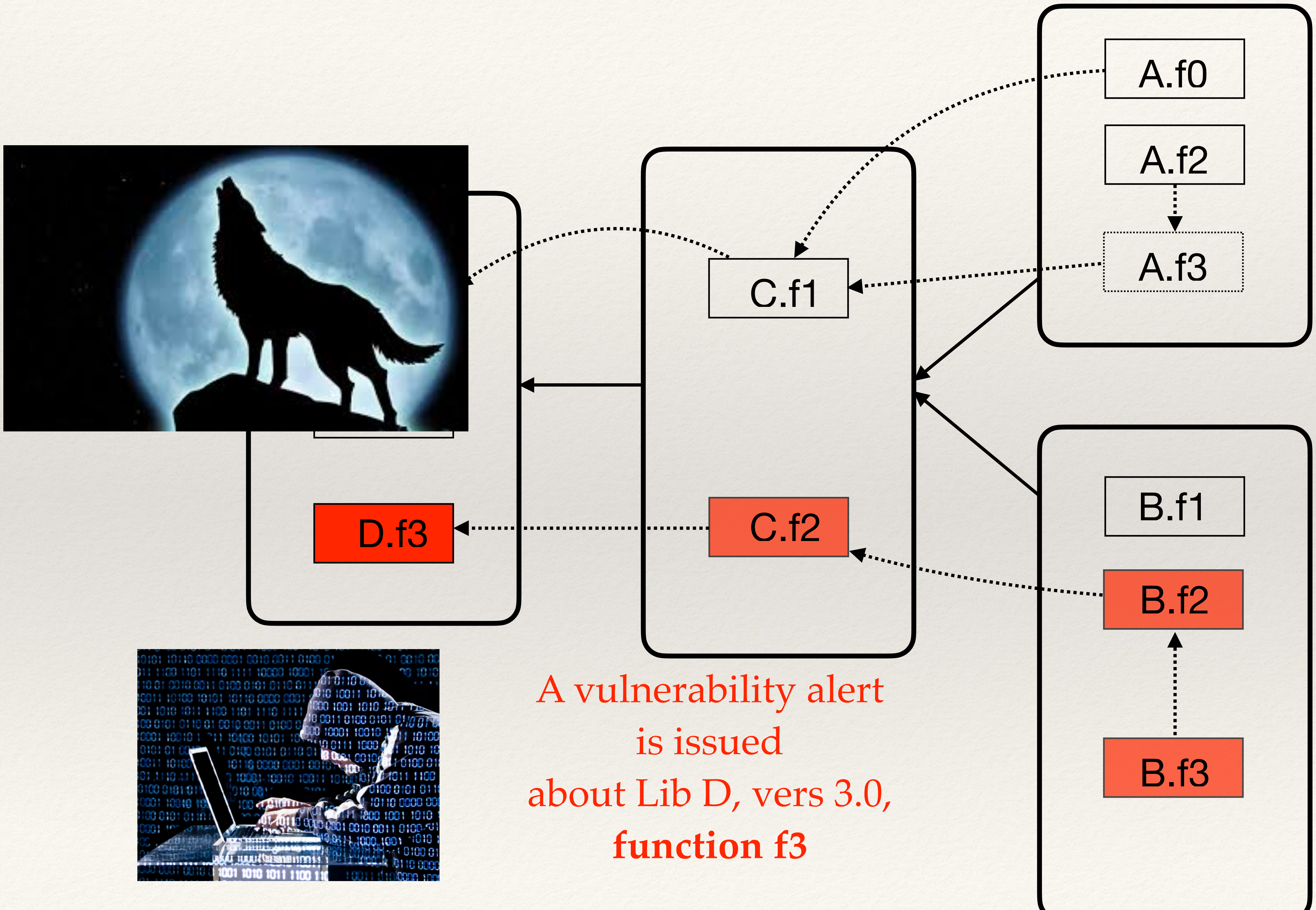
Epidemics in dependency graphs

Much more informative!



A vulnerability alert
is issued
about Lib D, vers 3.0,
function f3

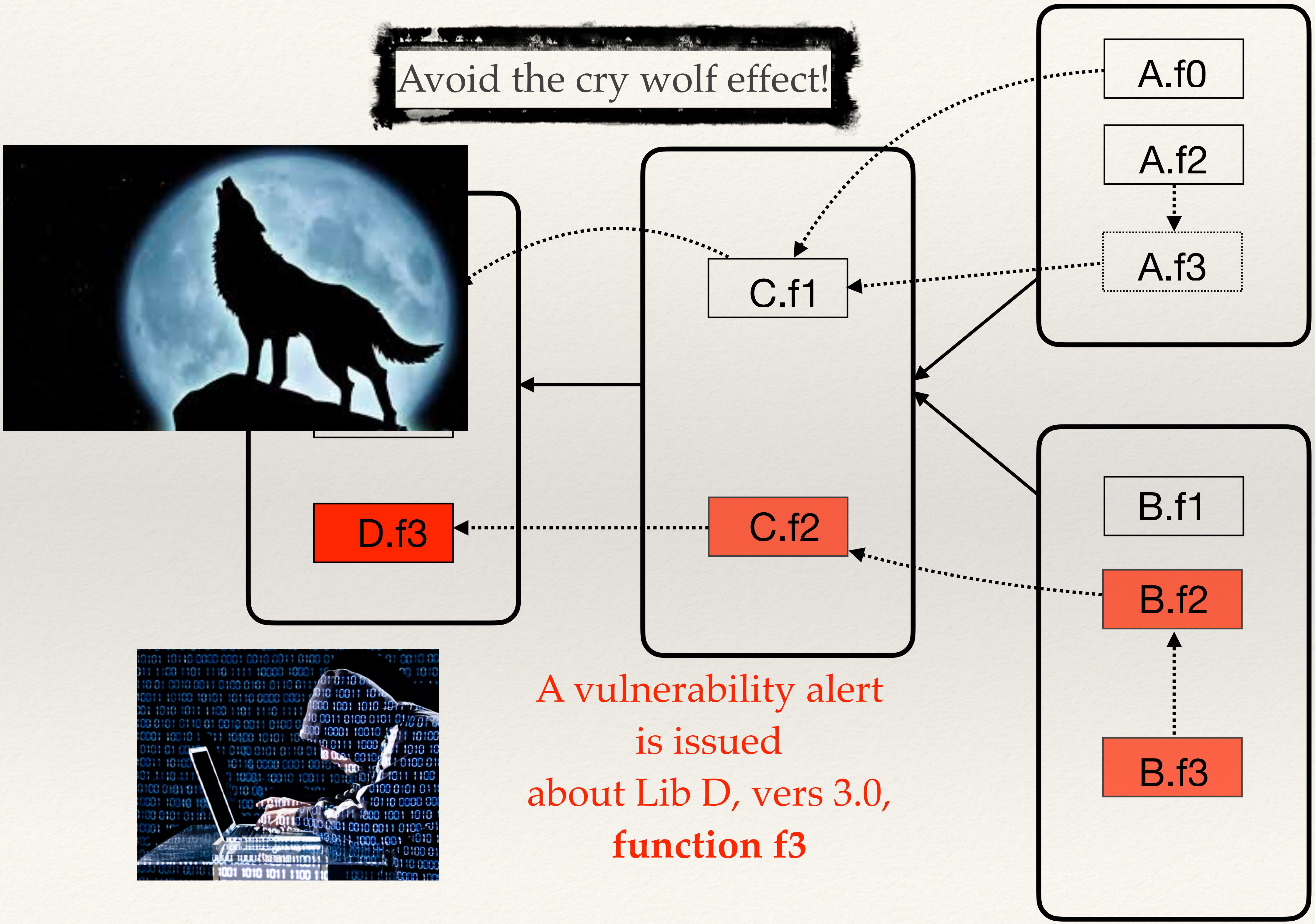
Epidemics in dependency graphs



A vulnerability alert
is issued
about Lib D, vers 3.0,
function f3

Epidemics in dependency graphs

Avoid the cry wolf effect!



A vulnerability alert
is issued
about Lib D, vers 3.0,
function f3

Examples

Examples

- Fully precise change impact analysis: *"How many libraries are affected if I remove/modify a certain method/interface?"*

Examples

- Fully precise change impact analysis: *"How many libraries are affected if I remove/modify a certain method/interface?"*
- Fully precise license compliance: *"Is my library compliant with the licenses of the libraries that I depend from (directly or indirectly)? (e.g., am I linking any GPL code?)"*

Examples

- Fully precise change impact analysis: *"How many libraries are affected if I remove/modify a certain method/interface?"*
- Fully precise license compliance: *"Is my library compliant with the licenses of the libraries that I depend from (directly or indirectly)? (e.g., am I linking any GPL code?)"*
- Fully precise risk profiling: *"Does this vulnerability affect my code?"*

Examples

- Fully precise change impact analysis: *"How many libraries are affected if I remove/modify a certain method/interface?"*
- Fully precise license compliance: *"Is my library compliant with the licenses of the libraries that I depend from (directly or indirectly)? (e.g., am I linking any GPL code?)"*
- Fully precise risk profiling: *"Does this vulnerability affect my code?"*
- Centrality analysis: *"What methods/functions are more central within a given ecosystem? are there bottlenecks? critical points?"*

The FASTEN toolchain

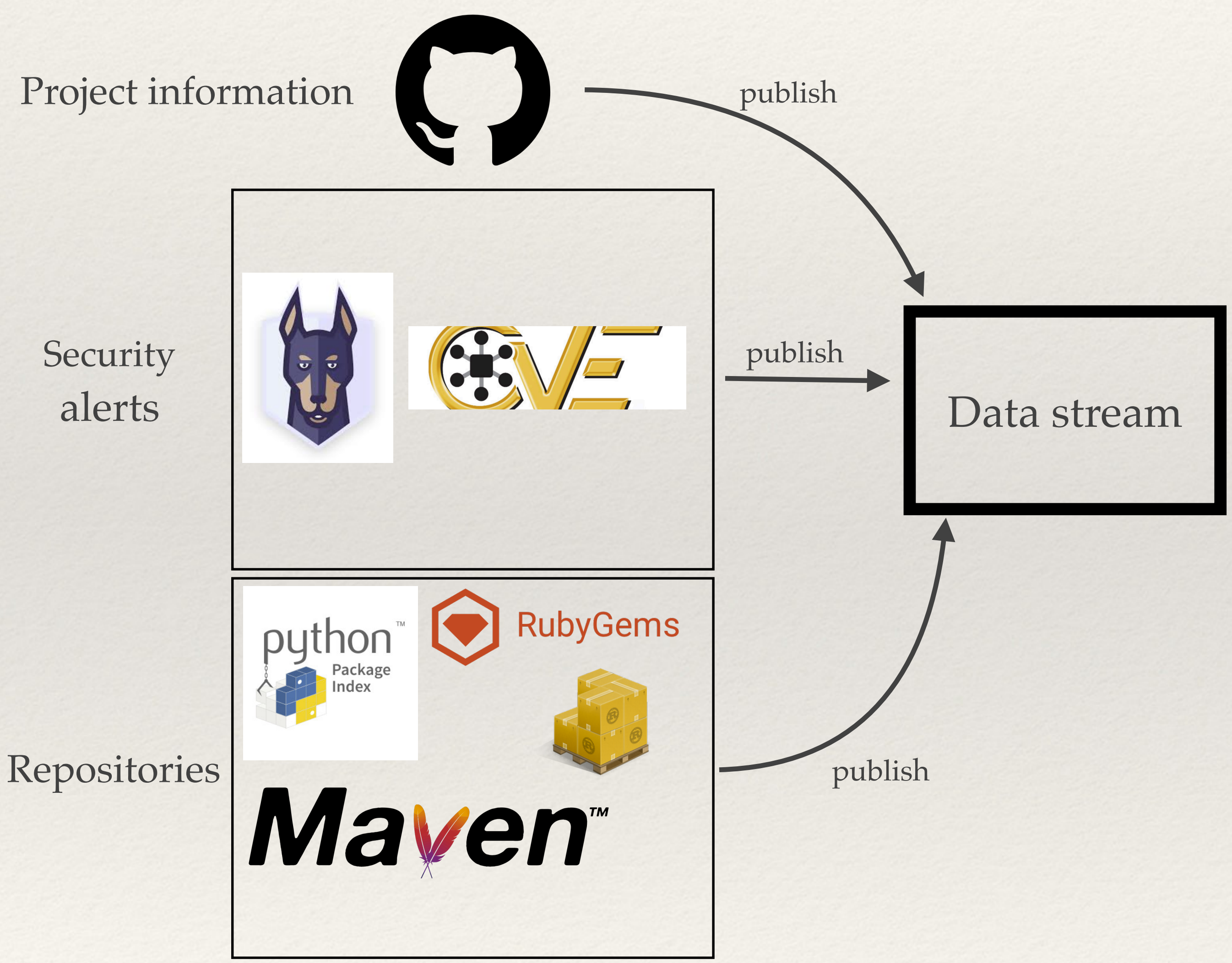
The FASTEN toolchain

Project information

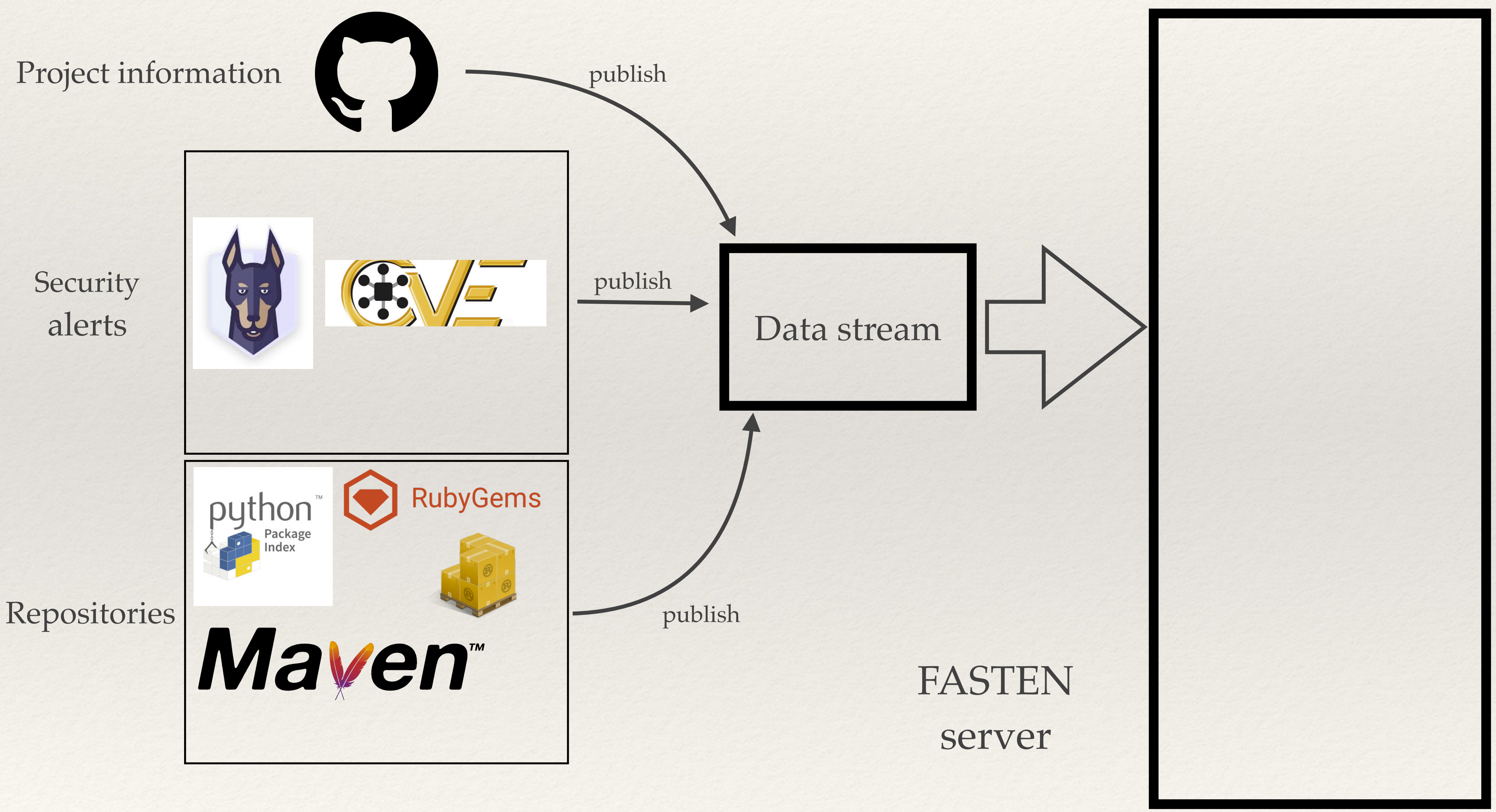
Security alerts

Repositories

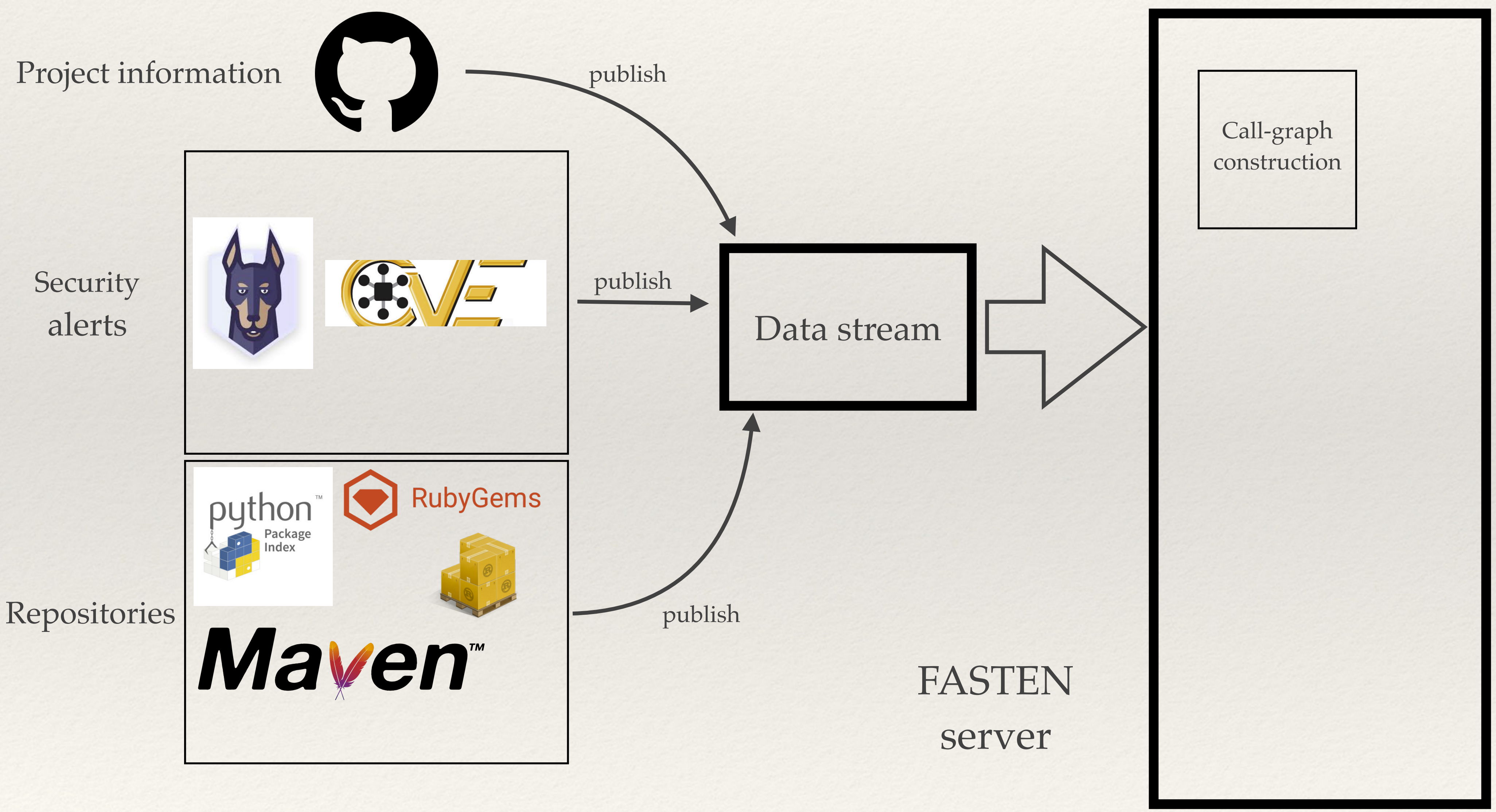
The FASTEN toolchain



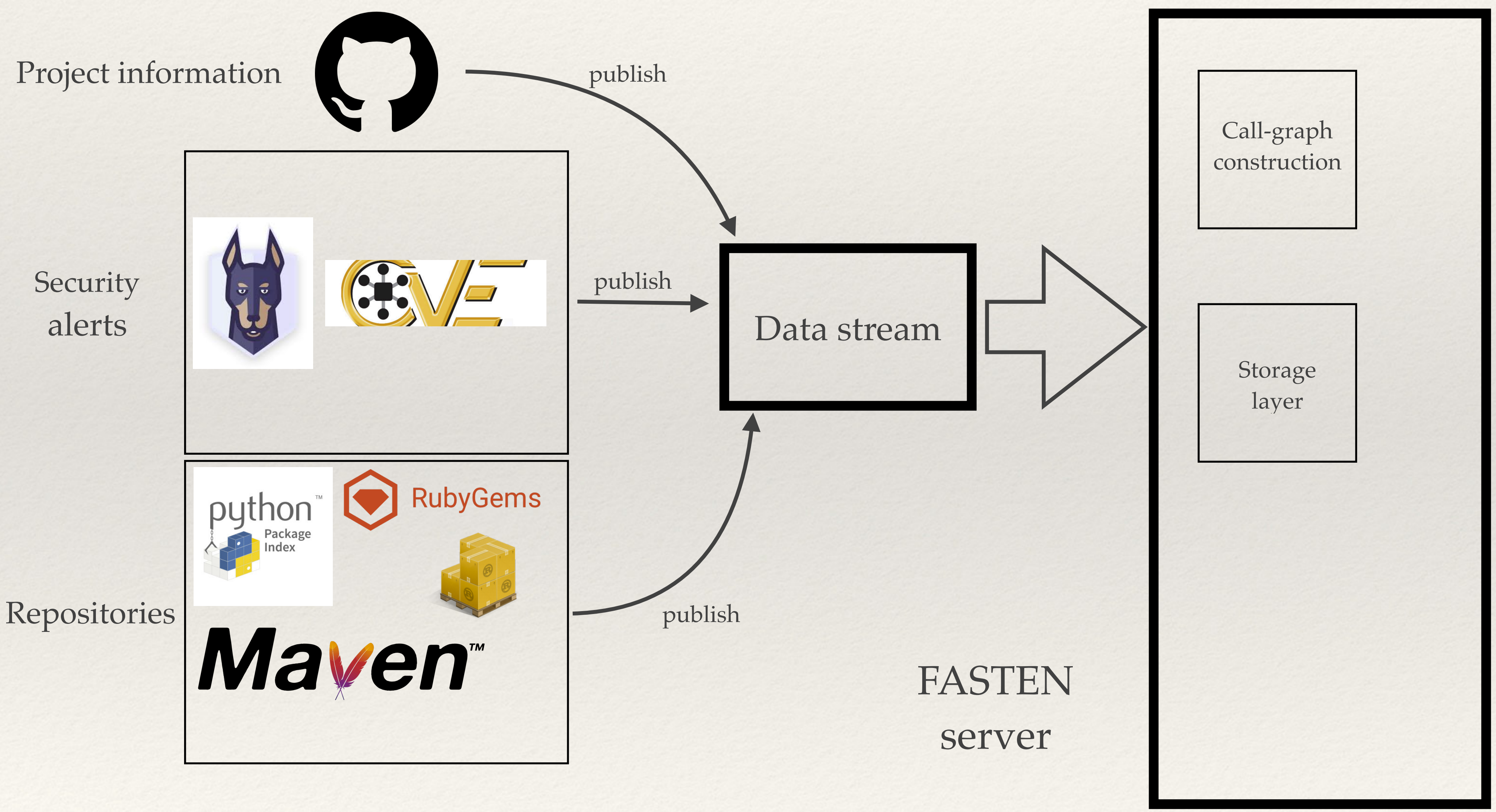
The FASTEN toolchain



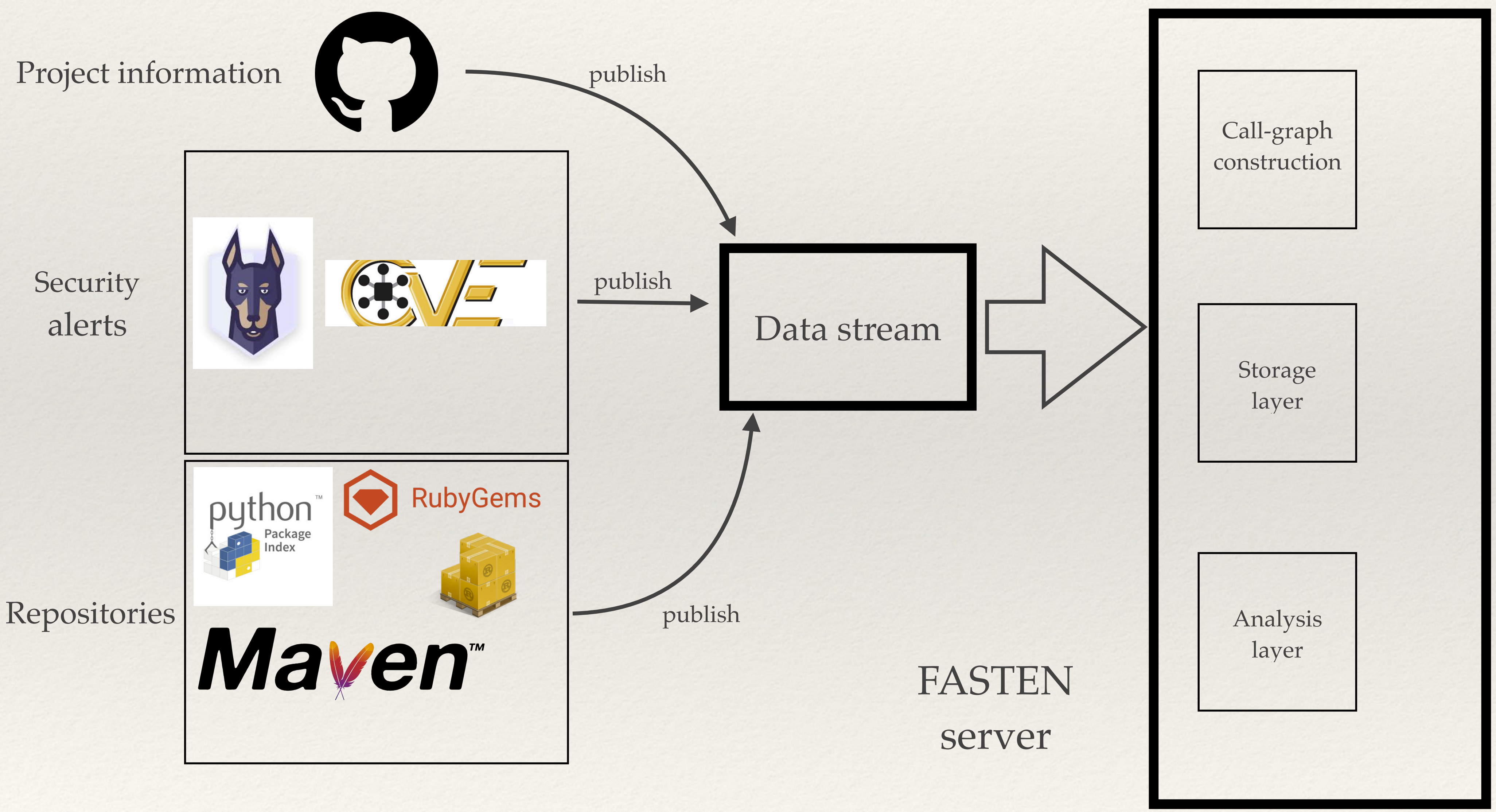
The FASTEN toolchain



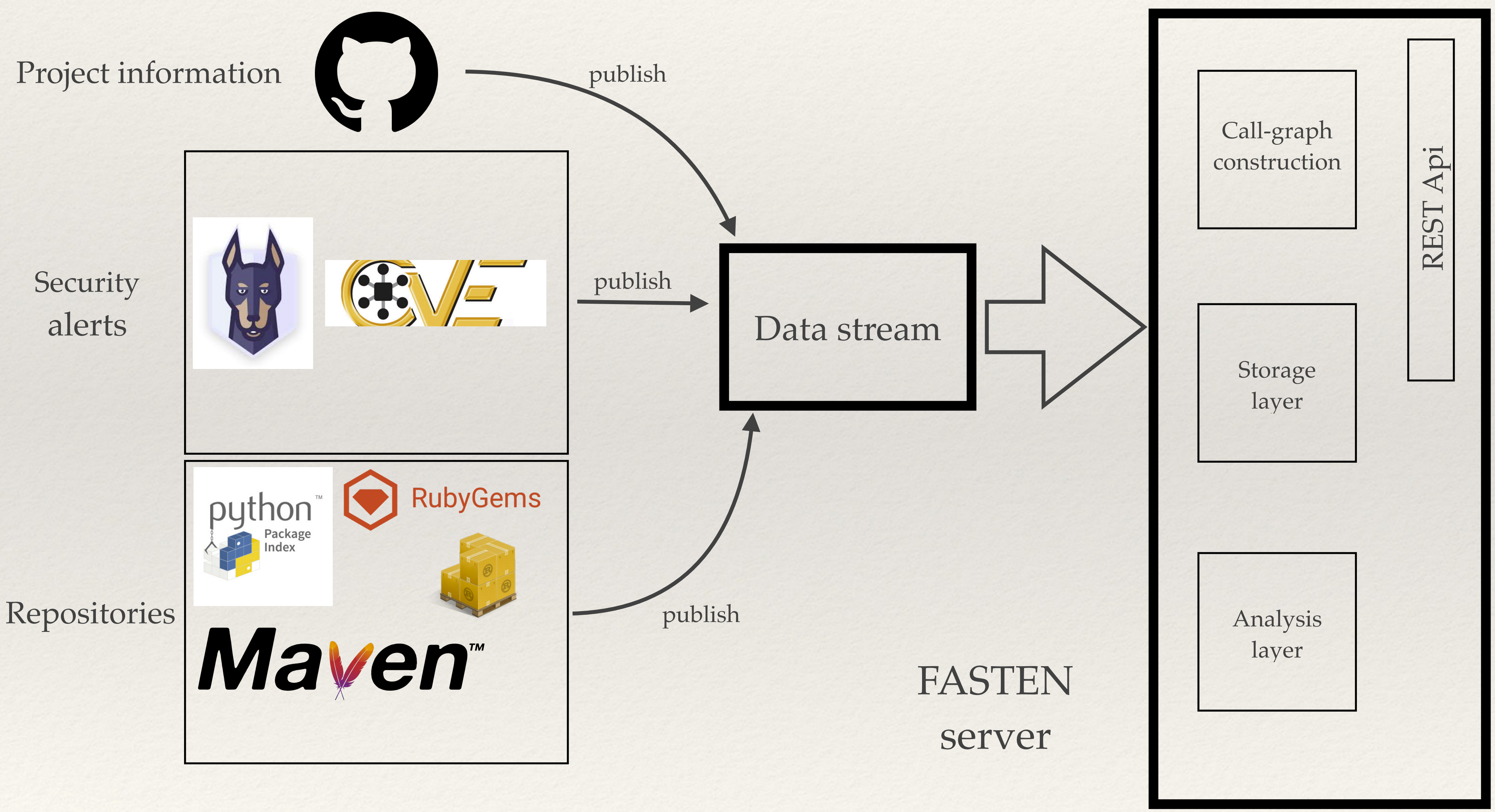
The FASTEN toolchain



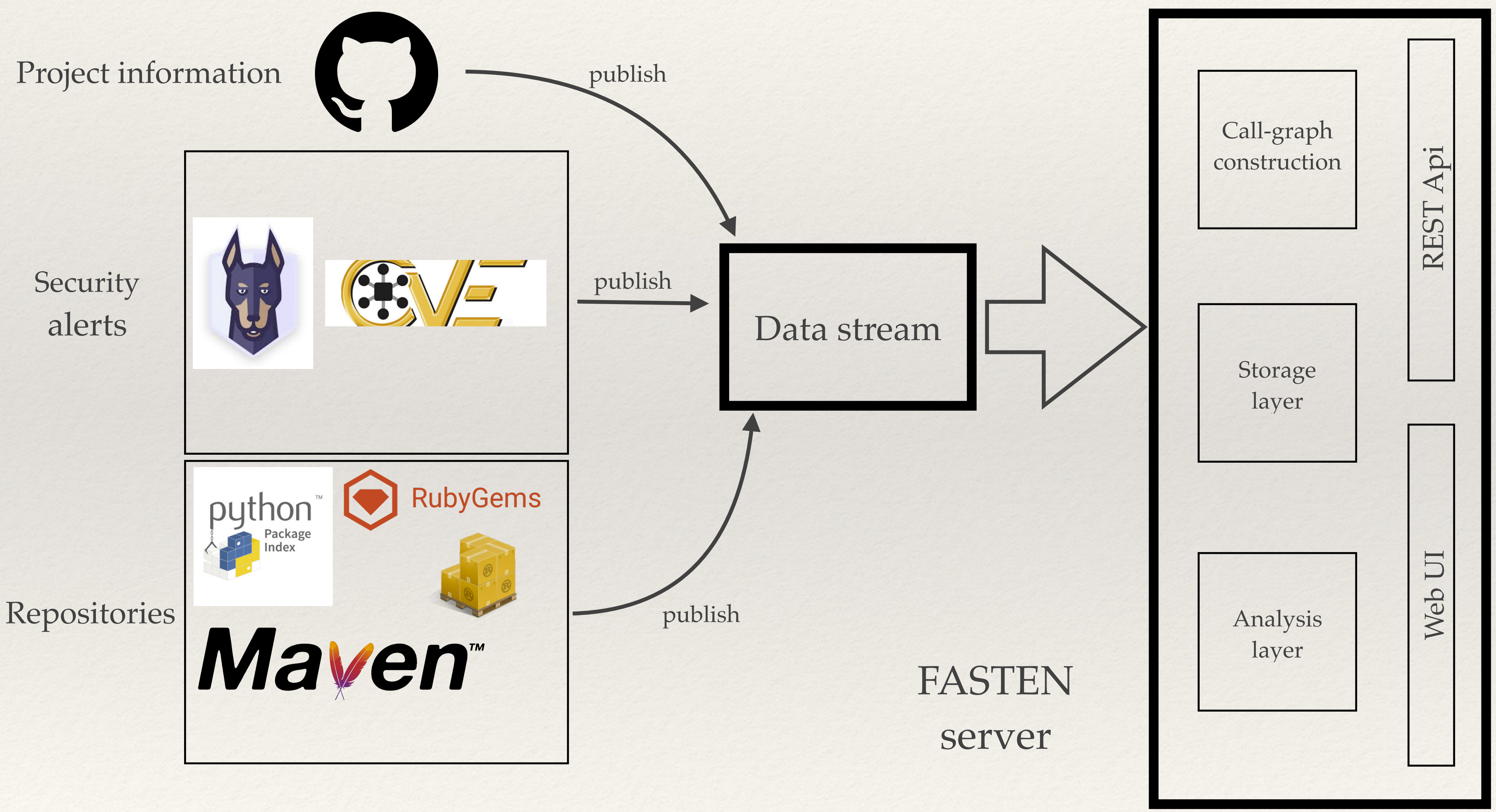
The FASTEN toolchain



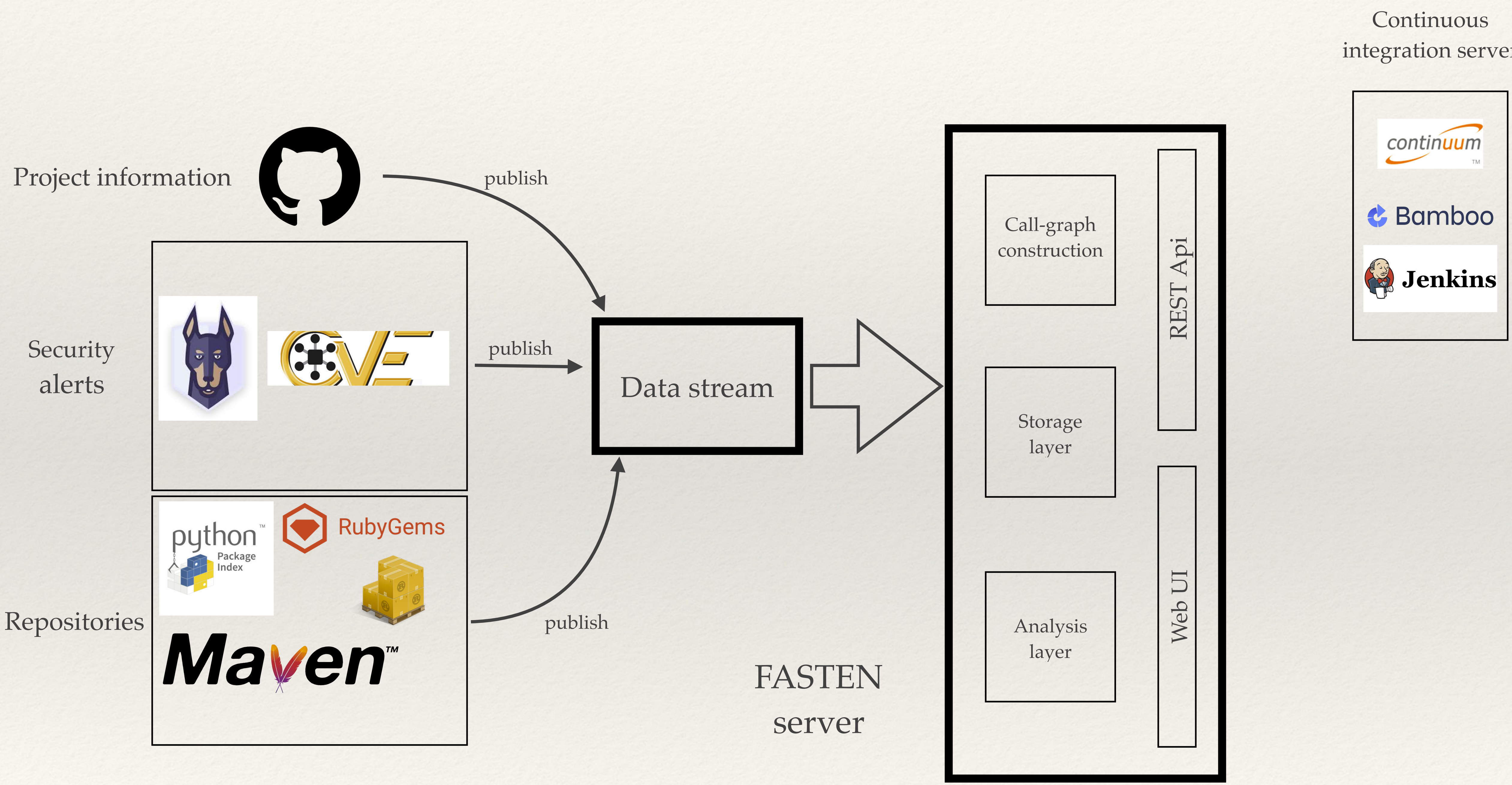
The FASTEN toolchain



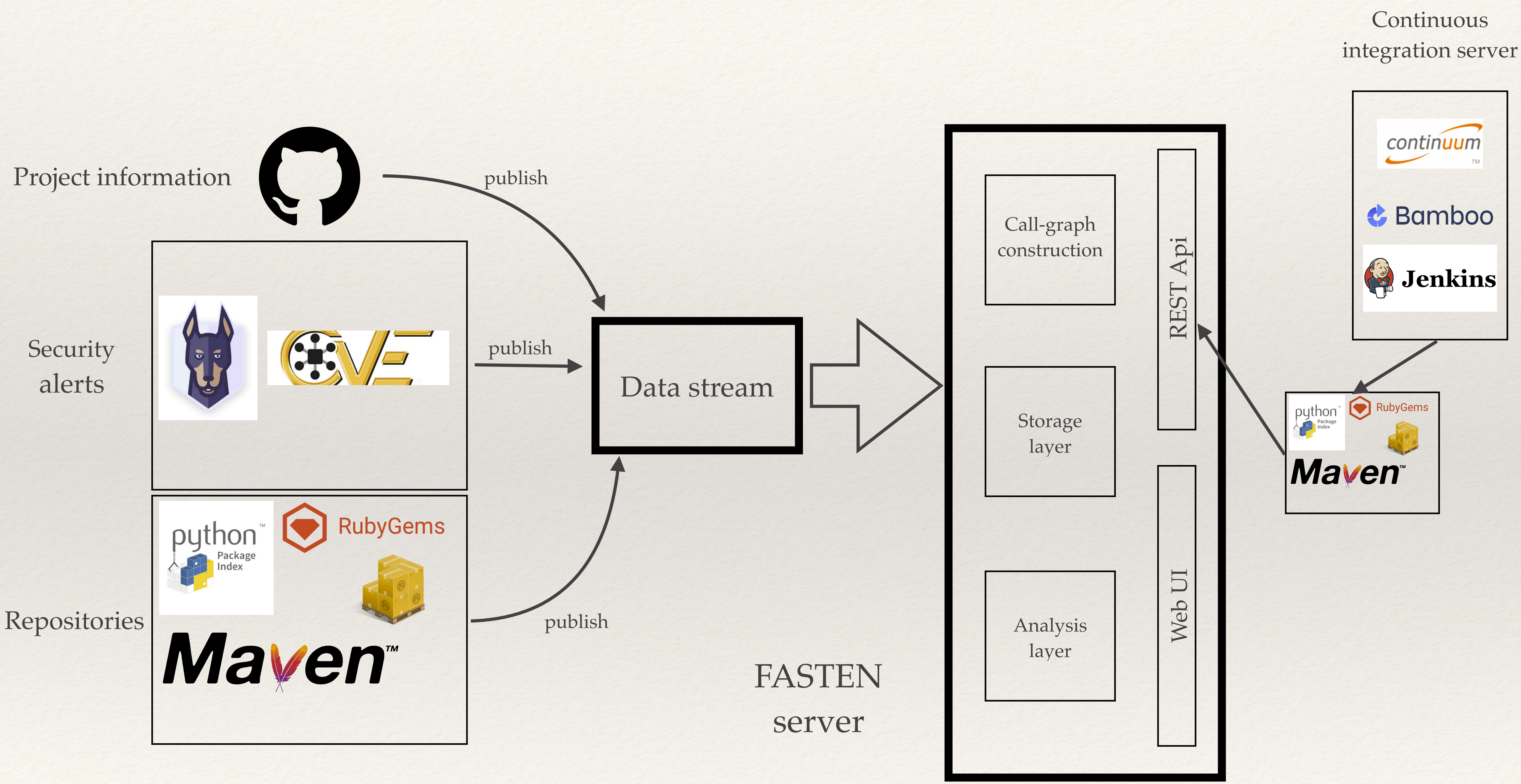
The FASTEN toolchain



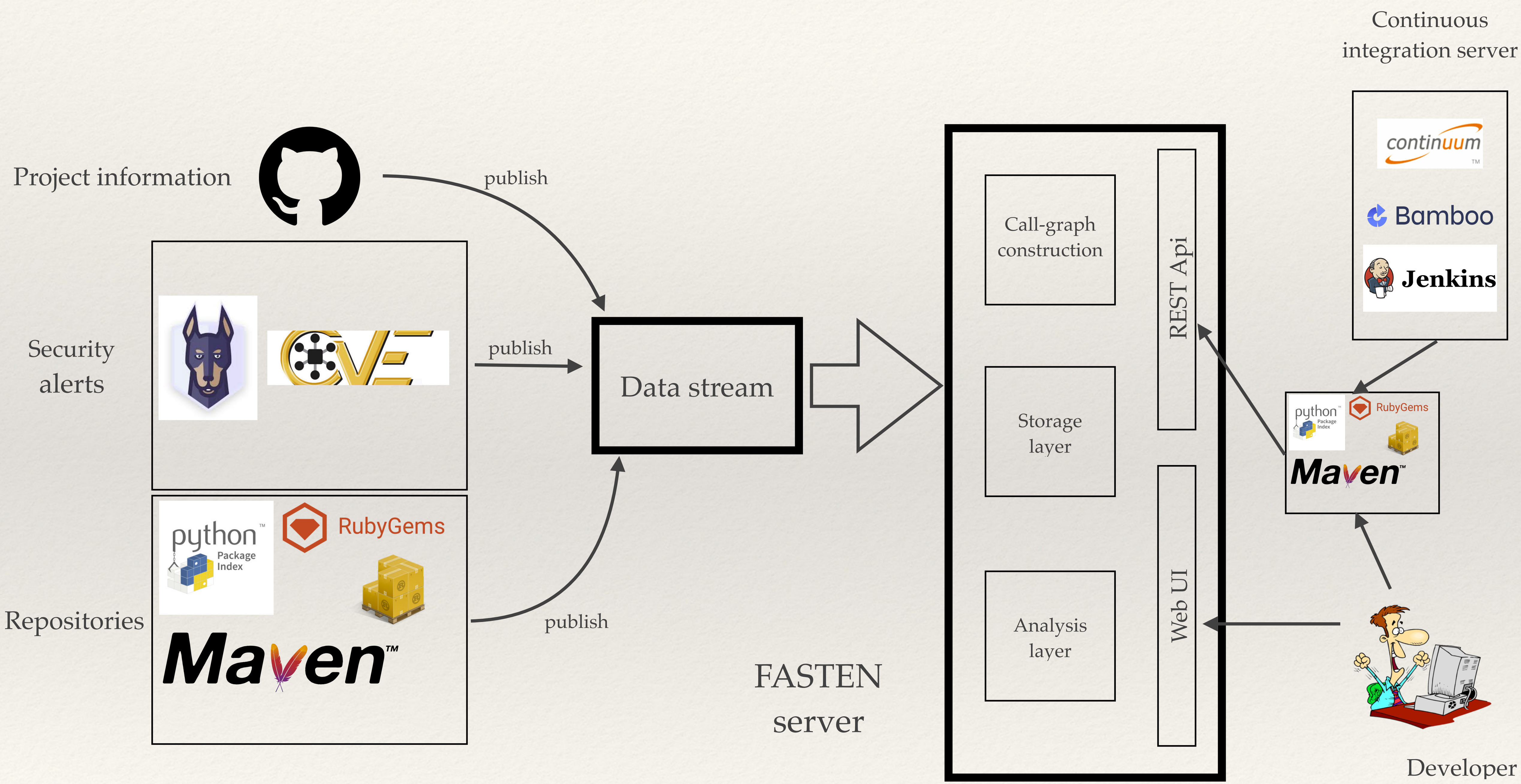
The FASTEN toolchain



The FASTEN toolchain



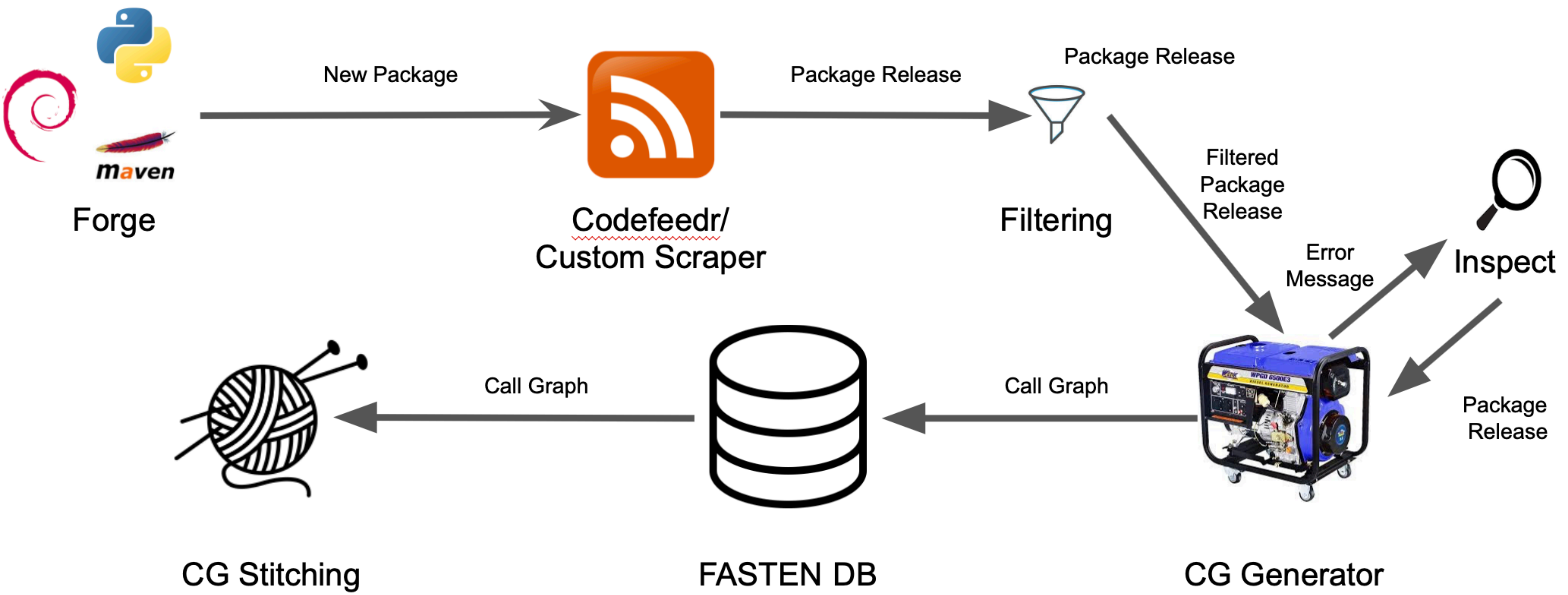
The FASTEN toolchain



Preliminary results

Server-side highlights

Dataflow example: CG generation



Universal function identifiers

How to uniquely reference a function in a global namespace?

`scheme`	`fasten://`
`forge`	`/mvn`
`artifact`	`/org.slf4j.slf4j-api`
`version`	`/1.2.3`
`namespace`	`/org.slf4j.helpers`
`function`	`/BasicMarkerFactory.getDetachedMarker`
`argument(s)`	`(%2Fjava.lang%2FString)`
`return type`	`%2Forg.slf4j%2FMarker`

Done

Universal function identifiers

How to uniquely reference a function in a global namespace?

`scheme`	`fasten://`
`forge`	`/mvn`
`artifact`	`/org.slf4j.slf4j-api`
`version`	`/1.2.3`
`namespace`	`/org.slf4j.helpers`
`function`	`/BasicMarkerFactory.getDetachedMarker`
`argument(s)`	`(%2Fjava.lang%2FString)`
`return type`	`%2Forg.slf4j%2FMarker`

Generic format +
Java
Python
C

Done

Call graph transport

```json
{
  "product": "foo",
  "forge": "mvn",
 "depset": [
    [
      { "product": "a", "forge": "mvn", "constraints": ["[1.2..1.5]", "[2.3..]"] },
      { "product": "b", "forge": "mvn", "constraints": ["[2.0.1]"] }
    ]
  ],
  "version": "3.10.0.7",
  "cha": {
    "/name.space/A": {
      "methods": {
        "0": "/name.space/A.A()%2Fjava.lang%2FVoidType",
        "1": "/name.space/A.g(%2Fjava.lang%2FString)%2Fjava.lang%2FInteger"
      },
      "superInterfaces": [ "/java.lang/Serializable" ],
      "sourceFile": "filename.java",
      "superClasses": [ "/java.lang/Object" ]
    }
  },
  "graph": {
    "internalCalls": [
      [ 0, 1 ]
    ],
    "externalCalls": [
      [ "1", "///their.package/TheirClass.method()Response", { "invokeinterface": "1" } ]
    ]
  },
  "timestamp": 123
}
```

Done

Call graph transport

Generic format +
Java
Python
C

```
{
  "product": "foo",
  "forge": "mvn",
 "depset": [
    [
      { "product": "a", "forge": "mvn", "constraints": ["[1.2..1.5]", "[2.3..]"] },
      { "product": "b", "forge": "mvn", "constraints": ["[2.0.1]"] }
    ]
  ],
  "version": "3.10.0.7",
  "cha": {
    "/name.space/A": {
      "methods": {
        "0": "/name.space/A.A()%2Fjava.lang%2FVoidType",
        "1": "/name.space/A.g(%2Fjava.lang%2FString)%2Fjava.lang%2FInteger"
      },
      "superInterfaces": [ "/java.lang/Serializable" ],
      "sourceFile": "filename.java",
      "superClasses": [ "/java.lang/Object" ]
    }
  },
  "graph": {
    "internalCalls": [
      [ 0, 1 ]
    ],
    "externalCalls": [
      [ "1", "///their.package/TheirClass.method()Response", { "invokeinterface": "1" } ]
    ]
  },
  "timestamp": 123
}
```

Language-dependent call graph generation

Done

Language-dependent call graph generation

- **Java**: Based on tools from the OPAL project (stg-tud/opal)
- **Python**: New static analysis tool: *PyCG (Submitted ICSE 2020)*
- **C**: CScout for static call graphs; gprof, callgrind for dynamic calls

In progress

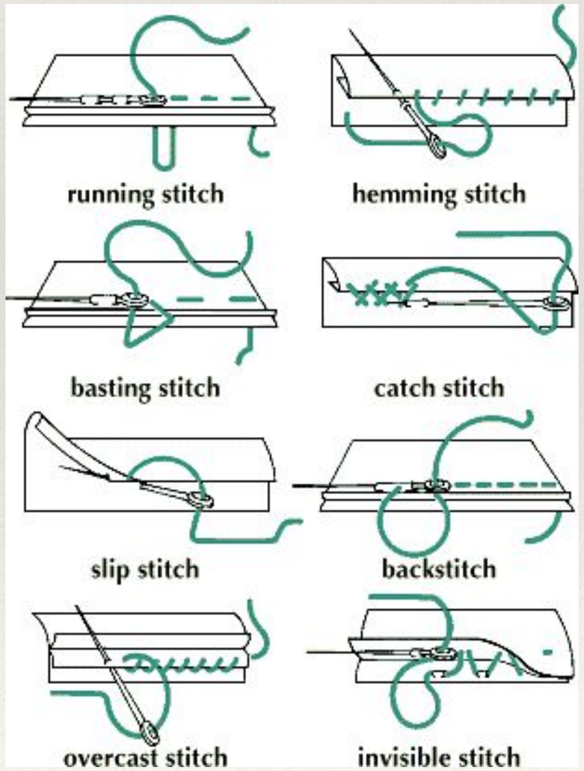
Current CG results

Language / Ecosystem	Total Packages	Results			
		Packages	Nodes	Edges	Success Rate
C / Debian Buster	7.380 (757 analyzed) *	531	491.721	579.253	70%
Java / Maven	2.7M artifacts	2.4M	~5B+	~56B+	89.13%
Python / PyPI	~740 K	~520K	~211M	~310M	70%

* Technical issues prohibited us from downloading the rest of the packages.

Call graph stitching

How to scale call graph processing to 10^6 package versions?



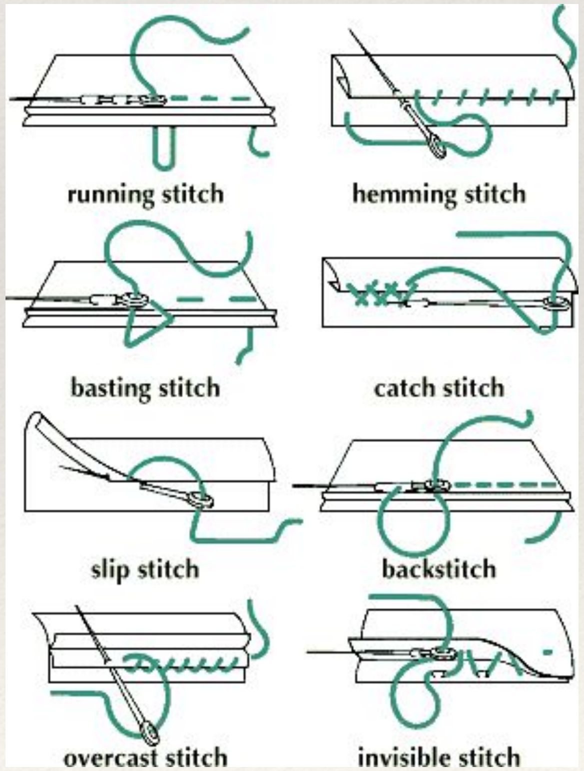
In progress

In progress

Call graph stitching

How to scale call graph processing to 10^6 package versions?

- **Idea:** Decouple package resolution from call graph generation

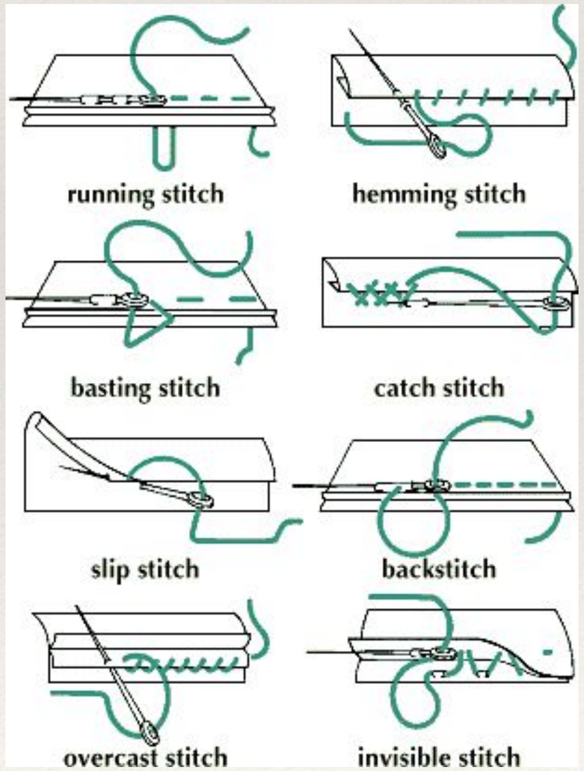


In progress

Call graph stitching

How to scale call graph processing to 10^6 package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:

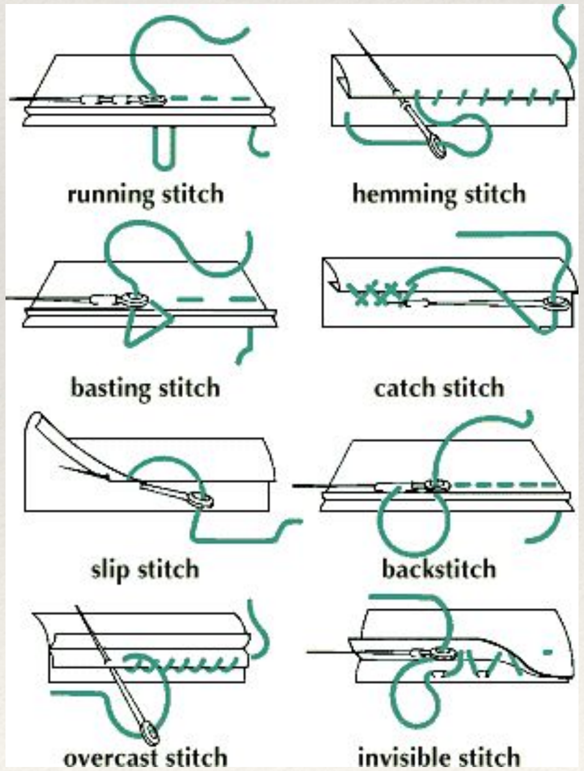


In progress

Call graph stitching

How to scale call graph processing to 10^6 package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:
  - unresolved calls

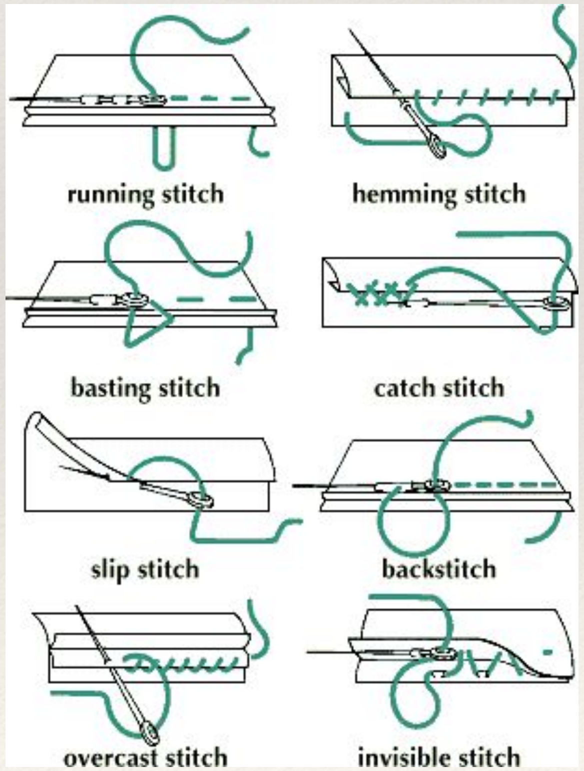


In progress

Call graph stitching

How to scale call graph processing to 10^6 package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:
  - unresolved calls
  - class hierarchies (Java, Python)

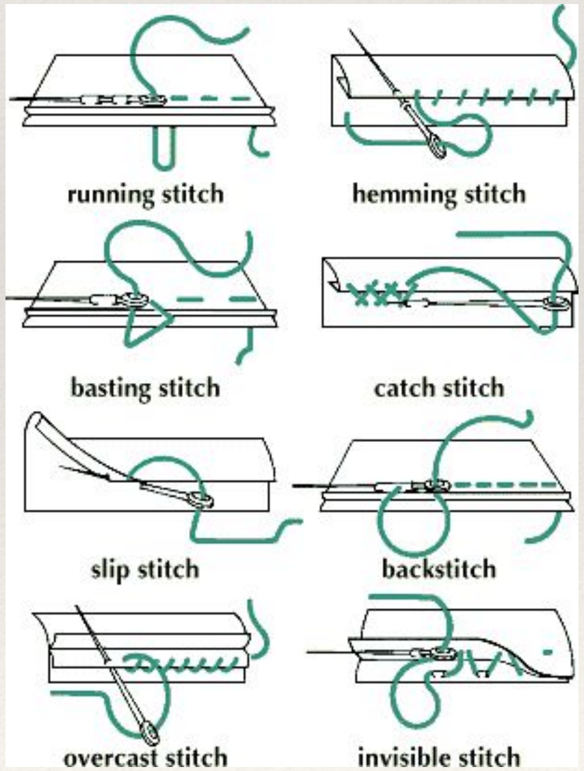


In progress

Call graph stitching

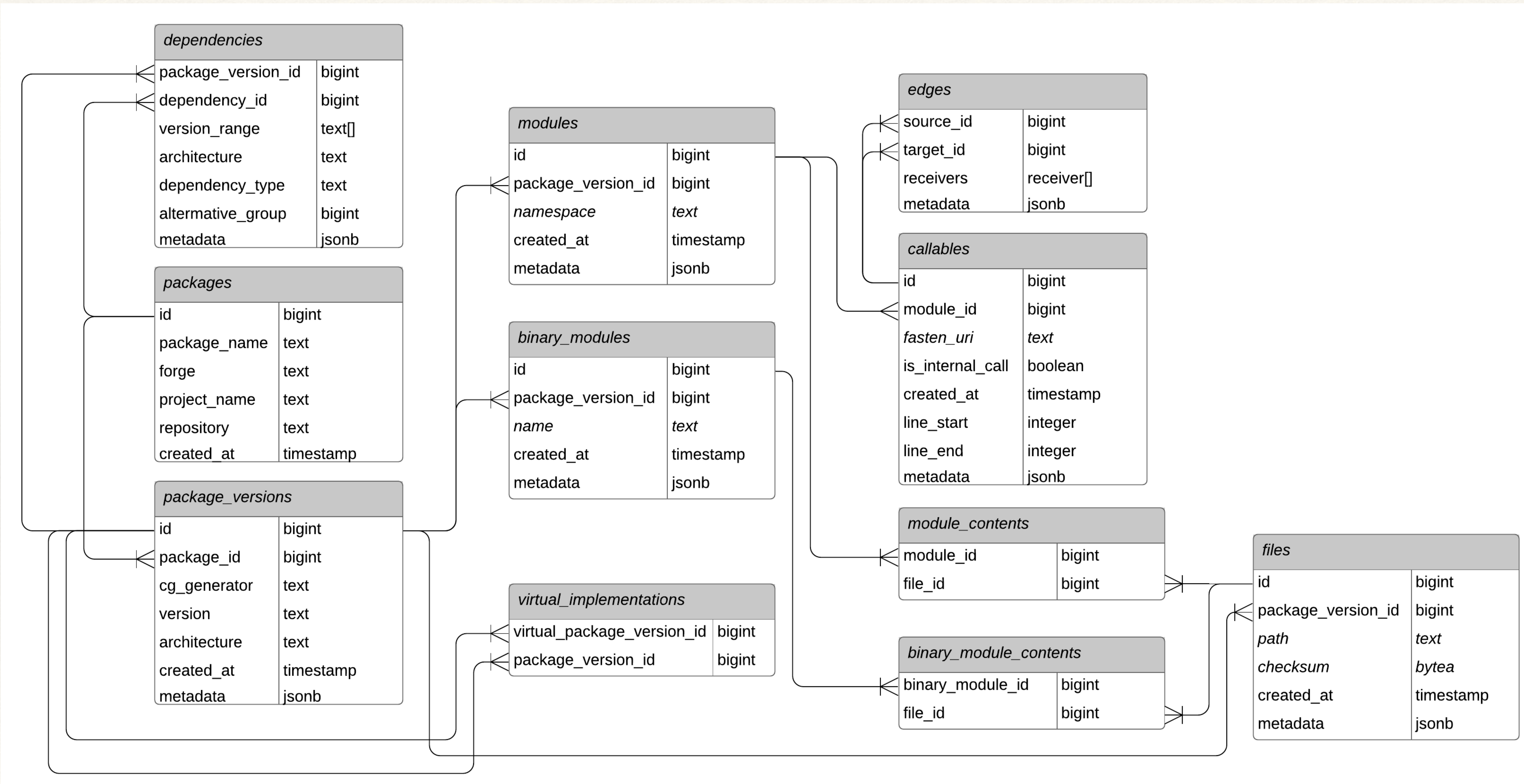
How to scale call graph processing to 10^6 package versions?

- **Idea:** Decouple package resolution from call graph generation
- Build and store call graphs per package version, incl.:
  - unresolved calls
  - class hierarchies (Java, Python)
- **Call graph stitching**: Resolve unresolved calls given a dependency tree



Done

The database schema



Examples of queries:
largest packages (# of functions)

```sql
select p.package_name, pv.version, count(*)
from package_versions pv
  join packages p on pv.package_id = p.id
  join modules m on m.package_version_id = pv.id
  join callables c on c.module_id = m.id
group by p.package_name, pv.version
order by count(*) desc
limit 10;
```

package_name	version	count
org.bouncycastle:bcprov-jdk15on	1.54	16912
com.google.guava:guava	20.0	13956
xalan:xalan	2.7.2	13058
org.apache.pdfbox:pdfbox	2.0.8	6727
external_callables_library	0.0.1	5457
org.apache.santuario:xmlsec	2.0.9	4783
org.apache.santuario:xmlsec	2.0.8	4780
org.apache.commons:commons-collections4	4.1	4607
org.apache.commons:commons-lang3	3.6	3432
org.apache.httpcomponents:httpclient	4.5.3	3024

(10 rows)

Examples of queries:
Packages depending on vulnerable package

```sql
SELECT package_version_id, p.package_name, pv.version
FROM dependencies d
JOIN package_versions pv  ON pv.id = d.package_version_id
JOIN packages p ON p.id = pv.package_id
WHERE d.dependency_id =
  (SELECT id
   FROM packages
   WHERE package_name = 'com.google.guava:guava')
AND '20.0' = ANY(d.version_range);
```

package_version_id	package_name	version
16	org.digidoc4j.dss:dss-utils-google-guava	5.0.d4j.5
41	org.digidoc4j.dss:dss-utils-google-guava	5.0.d4j.4
81	org.digidoc4j:digidoc4j	1.0.8.beta.2
107	org.digidoc4j:digidoc4j	1.0.7.beta.2
119	org.digidoc4j:digidoc4j	1.0.7.2
133	org.digidoc4j:digidoc4j	1.0.7.1
156	org.digidoc4j.dss:dss-utils-google-guava	5.1.d4j.5
142	org.digidoc4j.dss:dss-utils-google-guava	5.0.d4j.3

In progress

Graph analytics
(results shown refer to Java CG's)

In progress

Graph analytics
(results shown refer to Java CG's)

- Graph stored using WebGraph (UMIL)

In progress

Graph analytics
(results shown refer to Java CG's)

- Graph stored using WebGraph (UMIL)
- For 1.1M graphs (2.3B nodes, 18B edges):

In progress

Graph analytics
(results shown refer to Java CG's)

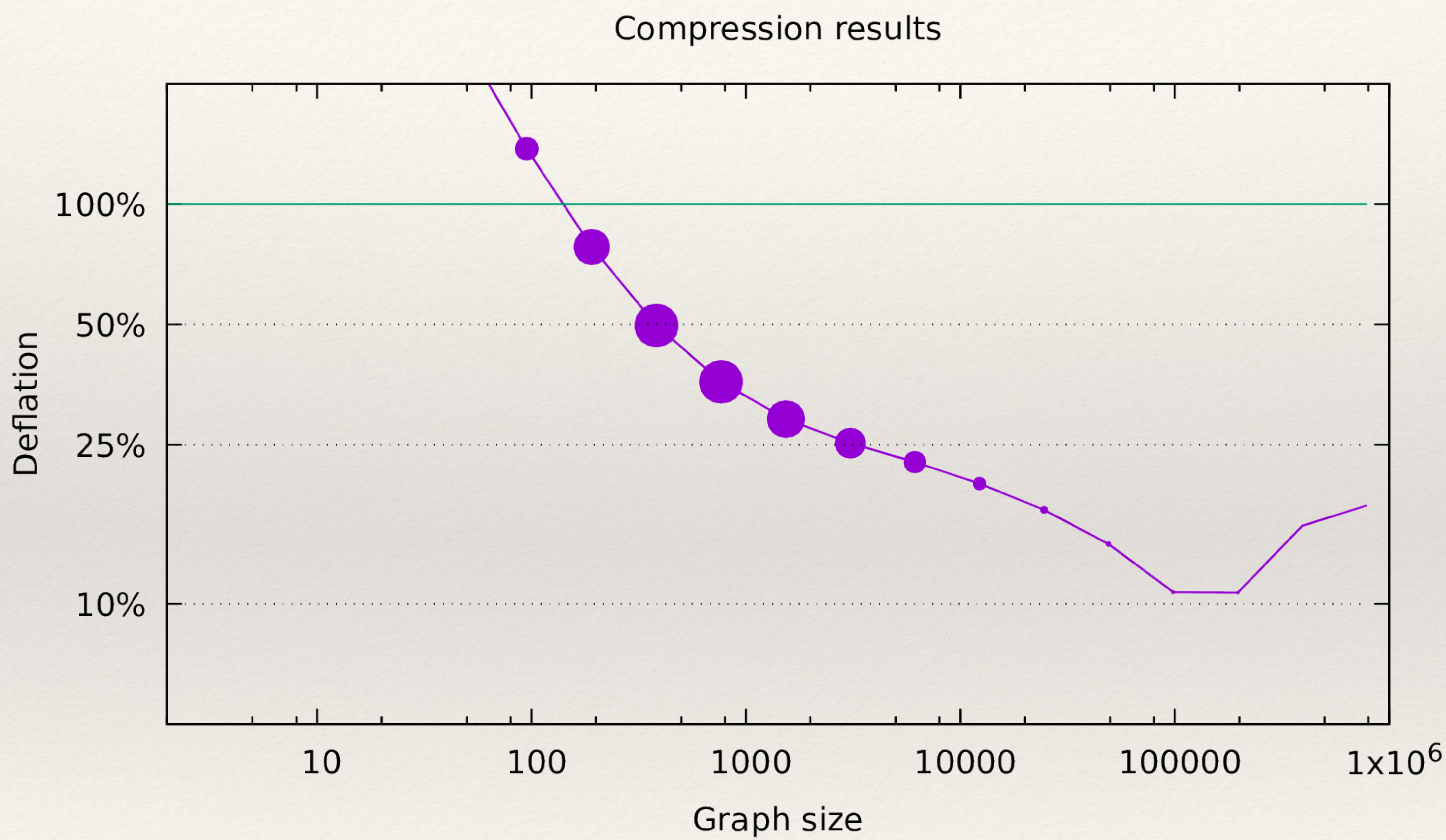
- Graph stored using WebGraph (UMIL)
- For 1.1M graphs (2.3B nodes, 18B edges):
  - 3.6 bits per edge, plus global ID storage for each node (9.0 bits per edge overall)

Graph analytics
(results shown refer to Java CG's)

- Graph stored using WebGraph (UMIL)
- For 1.1M graphs (2.3B nodes, 18B edges):
  - 3.6 bits per edge, plus global ID storage for each node (9.0 bits per edge overall)
  - DB size: 38GB → we can fit the whole of Maven in RAM

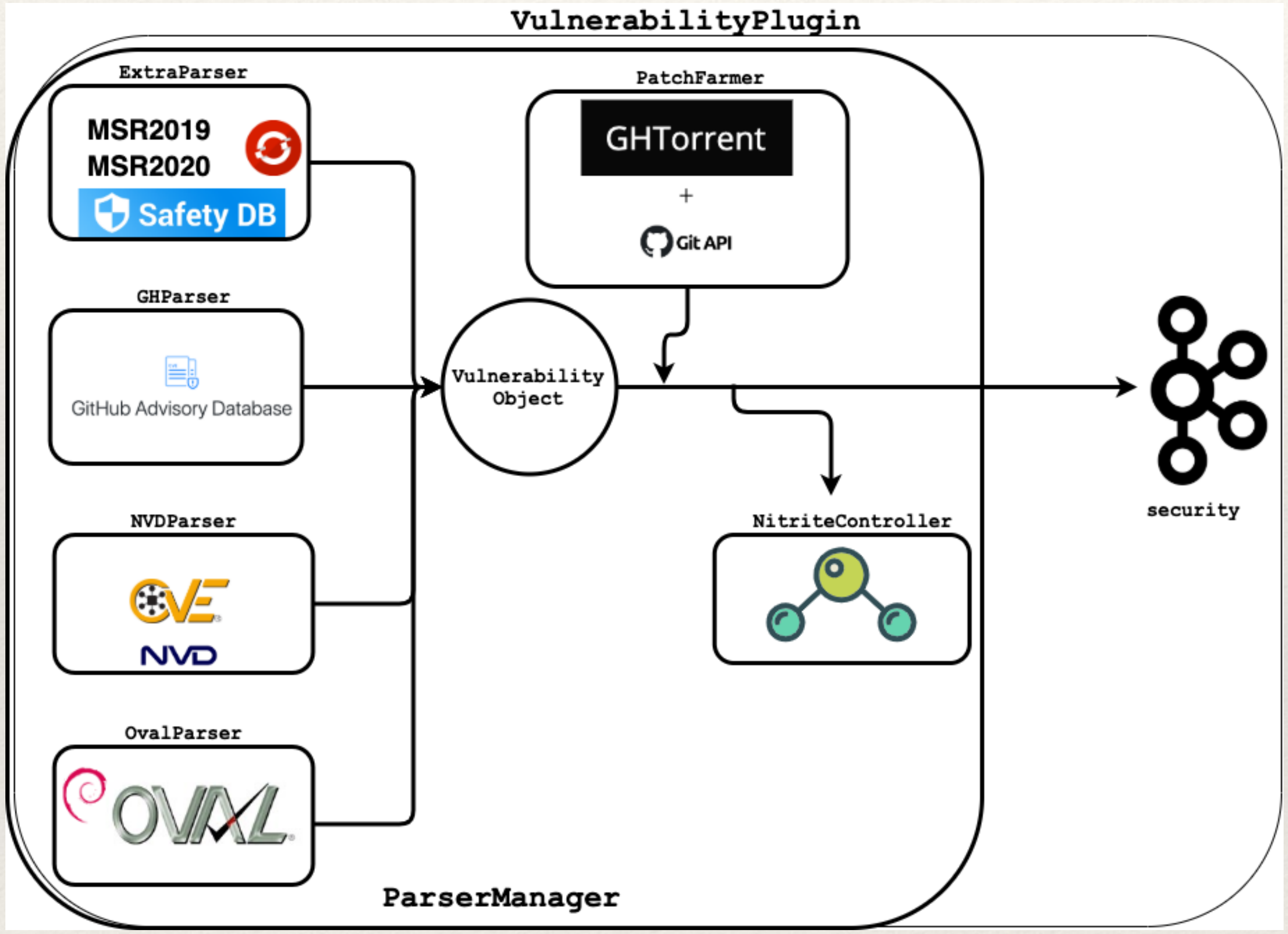
In progress

Graph storage



In progress

Vulnerability Plugin



- Gathering vulnerability information (at `package` and `callable` level)
- A normalized `Vulnerability` Object definition is injected in the metadata database
- Normalization is needed to smooth out the different sources of information
- The plugin continuously pulls updates for new information and keeps storing the results

In progress

Analysis plug-ins

RAPID: Risk Analysis and Propagation Inspection for Security and Maintainability risks

- On the server side (to enrich the metadata DB):
  - Plugin for code *maintainability analysis*:
    V1 deployed, processed 126K Maven coordinates to date
  - Plugin for *security vulnerability propagation*
- On the client side:
  - A user application to model and present risks

In progress

License and Compliance analysis

- QMSTR Plugin consists of 3 steps:
    1. Use the CG generator to gather information about all the generated artifacts that will be distributed together with the source code
    2. Execution of static analysis tools that augment the build graph with license and compliance metadata
    3. Generation of a report with package's relevant license and authorship metadata that is finally distributed

Client-side highlights

In progress

REST API

- Implementation of endpoints to expose canned queries from the metadata database
- In development:
    - Full DB entity support
    - Custom extension points

In progress

Use cases

- Endocode
  - Endocode developed a license-compliance solution, called *Quartermaster*
  - They are integrating FASTEN to improve the precision of their compliance offering
- SIG
  - Integration of FASTEN in *BetterCodeHub*, their GitHub-connected code quality monitoring product
- XWiki
  - Risk validation in the dependencies at Maven build time
  - Risk validation in the installed extensions of an XWiki instance
  - Filter out available compatible extensions for an XWiki instance
  - Discoverability of XWiki components in available extensions

Future timeline

The future

End 2020 — REST API, first full version of knowledge base, CG enrichment, build graph integration, first public announcement

Q1 2021 — Impact analysis, integration with MVN / PyPI; first external user

Q2 2021 — Industrial use cases integrated; first external adoption

Q3 2021 — Licensing and security fully integrated; Data-driven API evolution

Q4 2021 — Project finished; external integrations

Q1 2022 — FASTEN 2?

Network analysis will be the next
step for the future of
software development

Network analysis will be the next step for the future of software development

Questions?

Paolo Boldi
Università degli Studi di Milano
Italy
paolo.boldi@unimi.it