CHAOSS

THE LINUX FOUNDATION PROJECTS

Dr. Tom Mens
Dr. Ahmed Zerouali

Software Engineering Lab
University of Mons

tom.mens@umons.ac.be

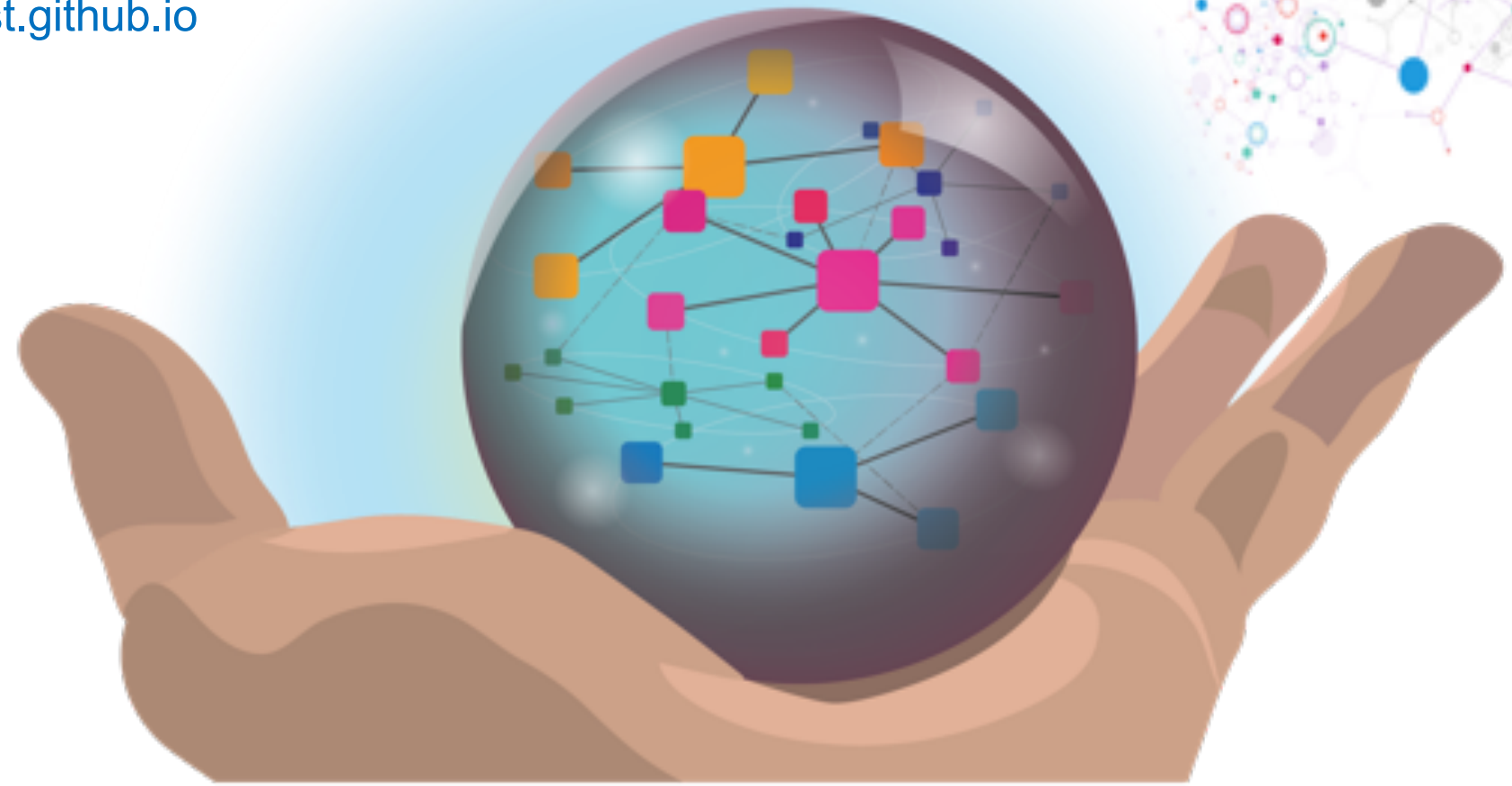
@tom_mens

UMONS
Université de Mons

chaoss.community

@secoassist

secoassist.github.io



fwo

fnrs

LA LIBERTÉ DE CHERCHER

UMONS

Université de Mons

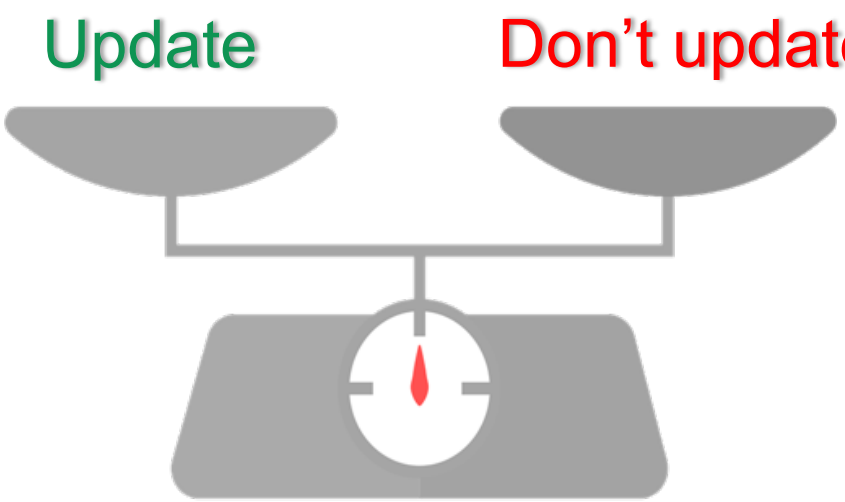
SECO-Assist

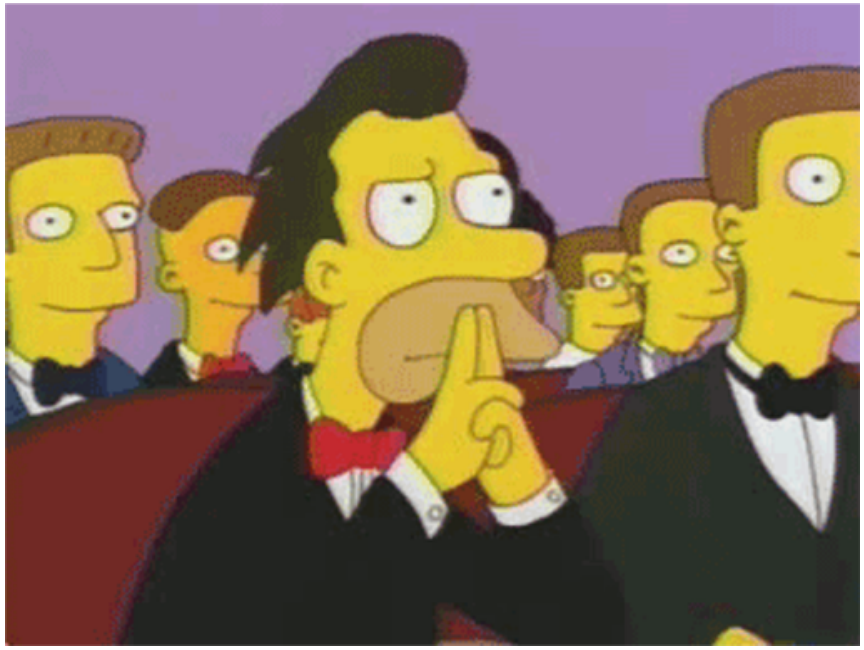
"Excellence of Science" Research Project

chaoss.community

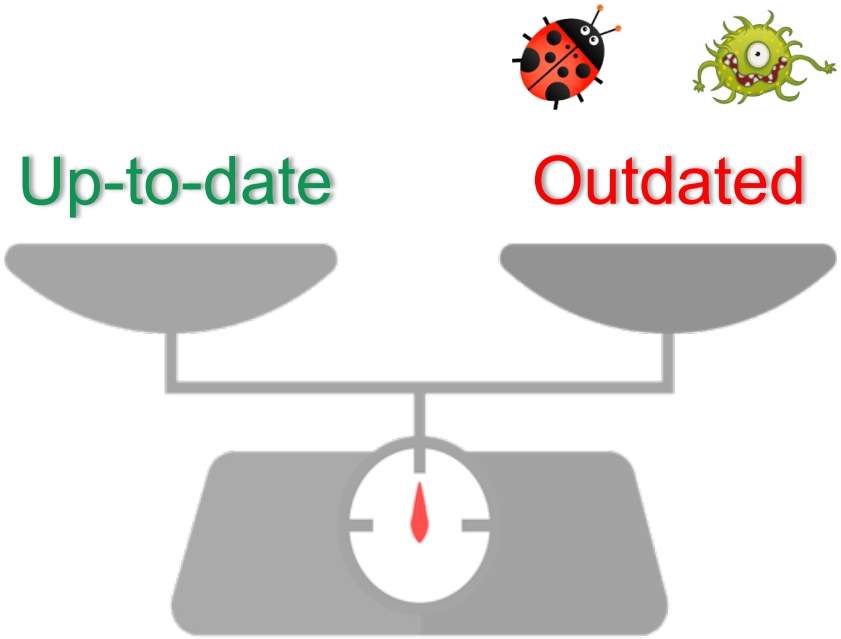
Focus

*Which **measures** can help software developers and deployers to decide **when** and **why** they should update?*





Focus



Online survey

What would be the most appropriate (i.e., ideal) version of a software library to depend on?

- *17 respondents*

  Highly educated with an average of 3 years of development experience

- *Responses:*
  - ★ Most stable (14)
  - ★ Latest available (9)
  - ★ Most documented (7)
  - ★ Most secure (5)

Idea: Technical Lag

"The increasing **difference** between deployed software packages and the **ideal** available upstream packages."

Ideal

- stability, security, functionality, recency, etc.

Difference

- time, version updates, bugs, vulnerabilities, …

J. Gonzalez-Barahona, P. Sherwood, G. Robles, D. Izquierdo (2017) "Technical lag in software compilations: Measuring how outdated a software deployment is." *IFIP International Conference on Open Source Systems*. Springer

Measuring Technical Lag

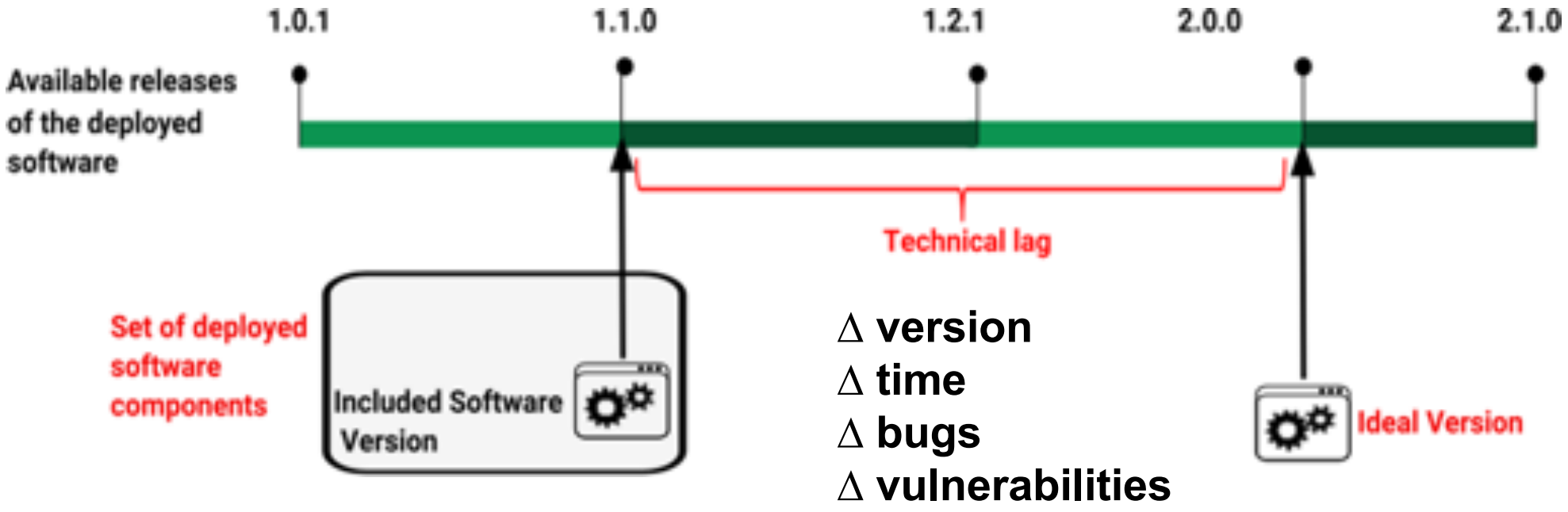
Semi-structured interviews:

FOSDEM 2019

5 highly educated software practitioners with an average of 10 years of experience

Technical Lag is important, especially if we mix between the benefits of updating and the effort required to do that.

Measuring Technical Lag



∆ version
∆ time
∆ bugs
∆ vulnerabilities

chaoss.community

Measuring Technical Lag

A **technical lag framework** *F* is a tuple *(C, L, ideal, delta, agg)* with

- *C* a set of component releases
- *L* a set of possible lag values
- *ideal*: $C \rightarrow C$ computes the "ideal" component release
- *delta*: $C \times C \rightarrow L$ computes the difference between two component releases
- *agg*: $2^L \rightarrow L$ aggregates the results of a set of lags

A formal framework for measuring technical lag in component repositories – and its application to npm. A. Zerouali, T. Mens, J. Gonzalez-Barahona, A. Decan, E. Constantinou, G. Robles. Wiley Journal on Software Evolution and Process, 2019

Measuring Technical Lag

Given a **technical lag framework** *F*, *we define*

$techlag_F(c) = delta(c, ideal(c))$

for any deployed component *c*

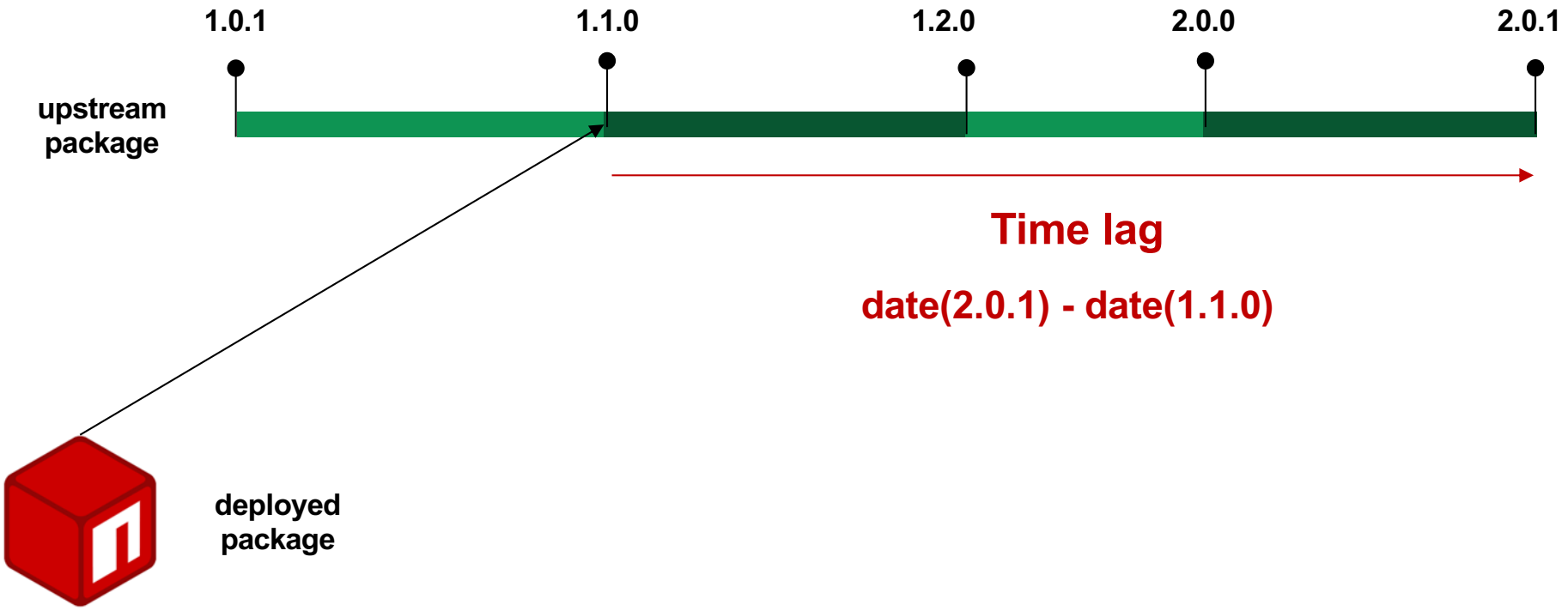
$aggLag_F(D) = agg(\ \{techlag_F(c) \mid c \text{ in } D\}\ )$

for any set of deployed components *D*

A formal framework for measuring technical lag in component repositories – and its application to npm. A. Zerouali, T. Mens, J. Gonzalez-Barahona, A. Decan, E. Constantinou, G. Robles. Wiley Journal on Software Evolution and Process, 2019

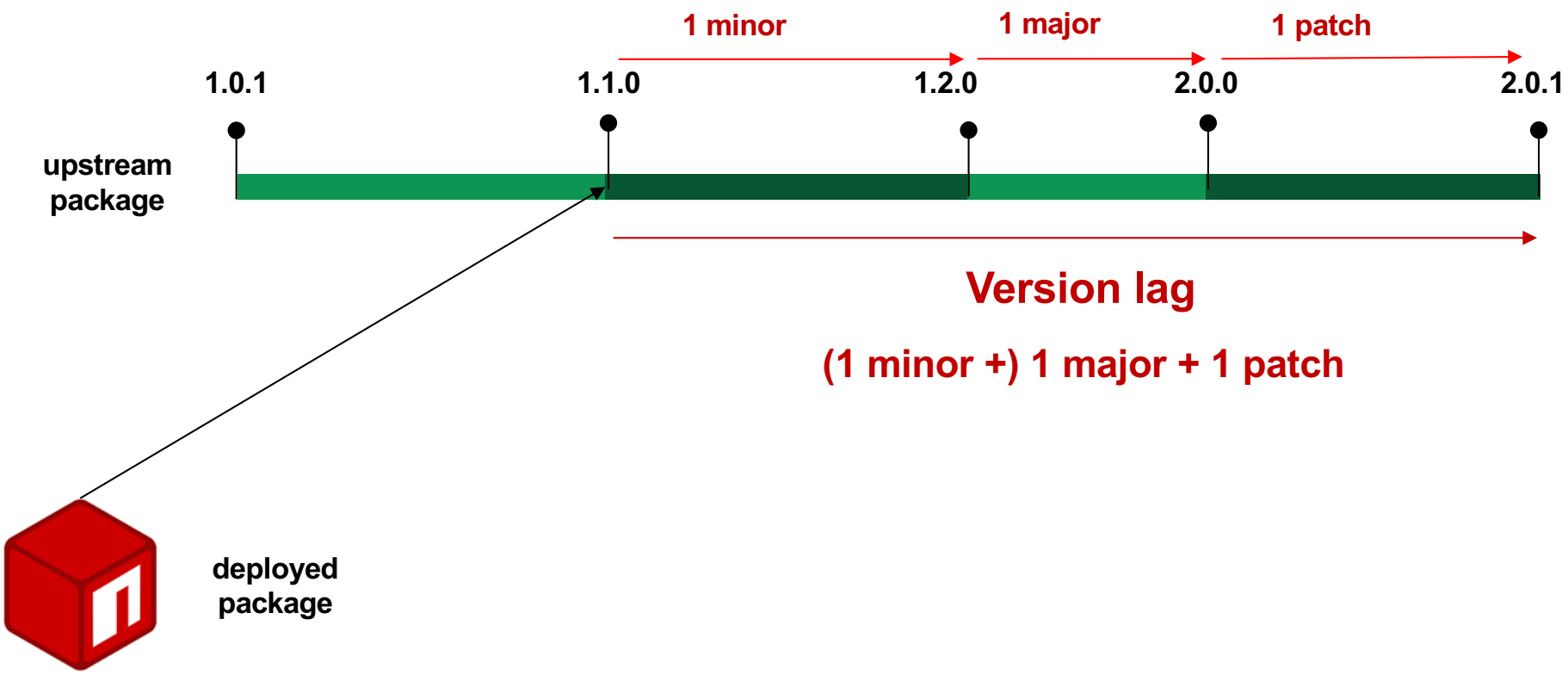
Technical Lag - Example

***Time*-based measurement of technical lag**



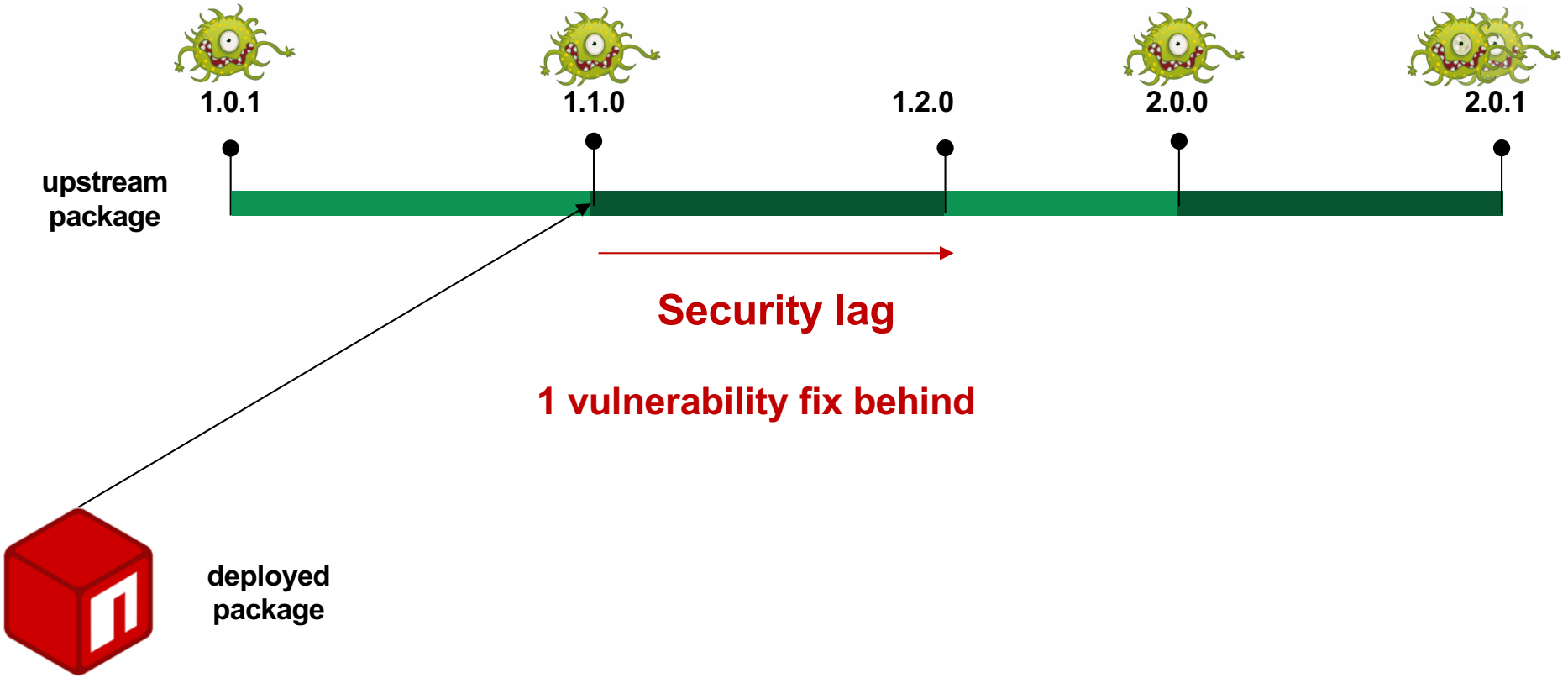
Technical Lag - Example

Version-based measurement of technical lag



Technical Lag - Example

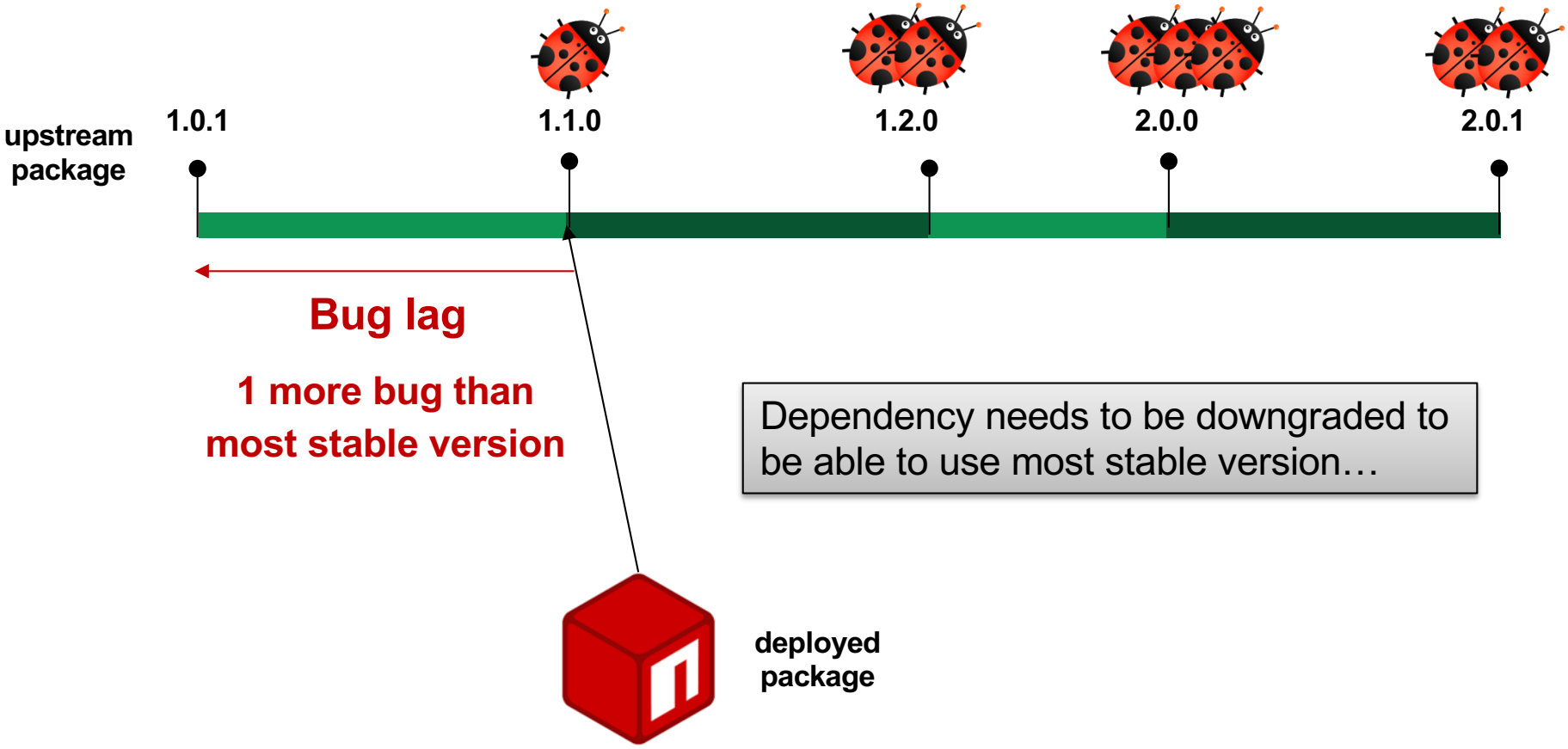
***Vulnerability*-based measurement of technical lag**



chaoss.community

Technical Lag - Example

***Bug*-based measurement of technical lag**



Case study 1: Technical lag in npm distribution of JavaScript packages

+20M dependencies

Credits: https://exploring-data.com/vis/npm-packages-dependencies/

A. Decan, T. Mens, E. Constantinou (2018) *On the evolution of technical lag in the npm package dependency network.* IEEE Int'l Conf. Software Maintenance and Evolution

Technical Lag – Example

youtube-player

5.5.2 • Public • Published 4 months ago

Readme | 3 Dependencies

Dependencies (3)

debug load-script sister

Dev Dependencies (15)

ava babel-cli babel-plugin-add-module-exports

babel-plugin-transform-flow-strip-types babel-plugin-transform-obj

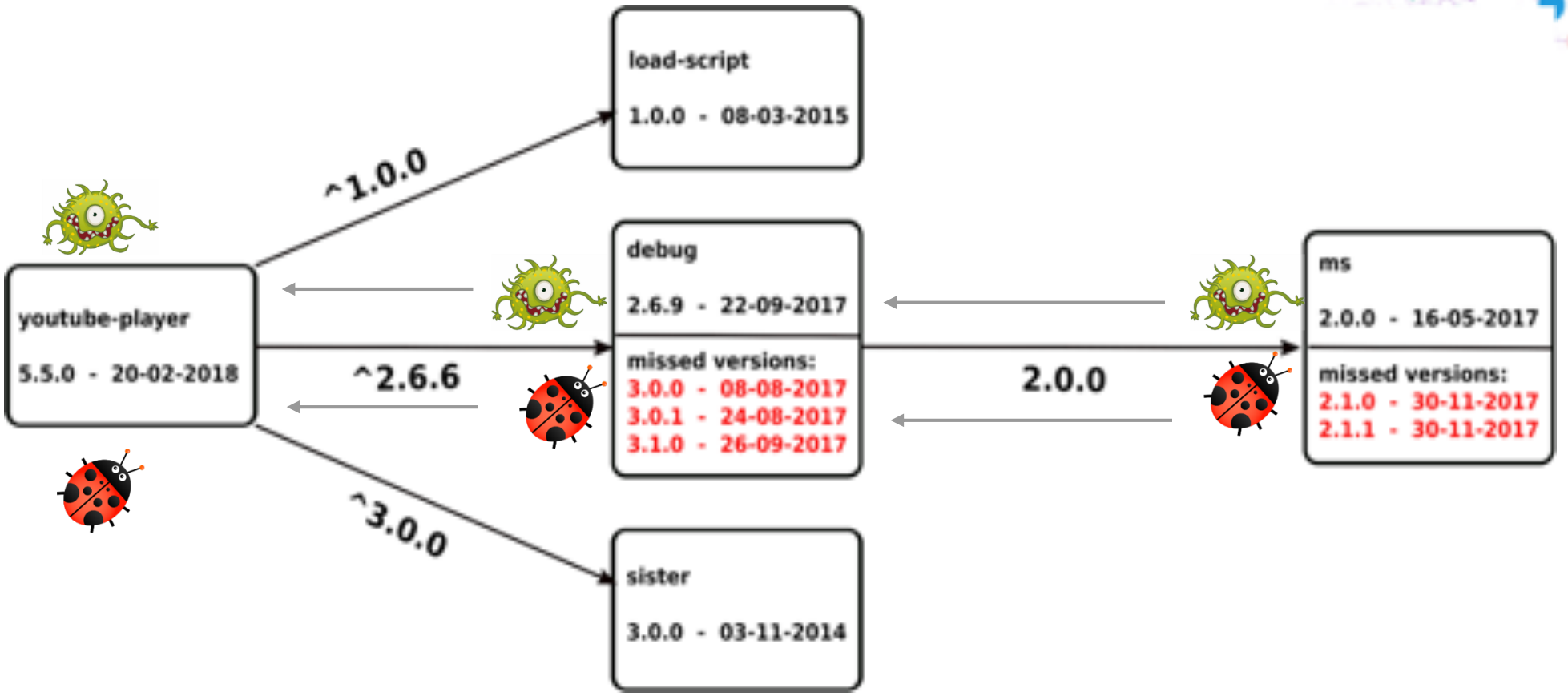
babel-preset-env babel-register chai eslint eslint-config-canonic

flow-copy-source husky npm-watch semantic-release

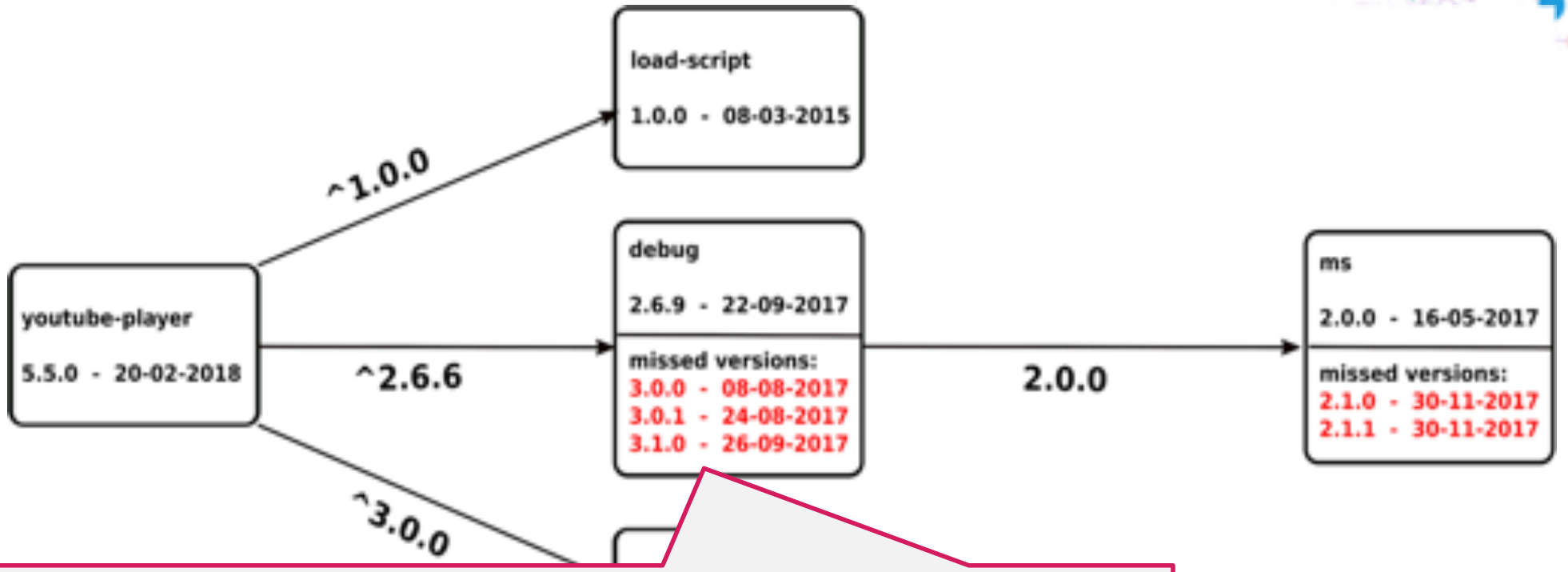
package.json

```
13  "dependencies": {
14    "debug": "^2.6.6",
15    "load-script": "^1.0.0",
16    "sister": "^3.0.0"
17  },
18  "description": "YouTube IFrame Player API abstraction.",
19  "devDependencies": {
20    "ava": "^0.19.1",
21    "babel-cli": "^6.24.1",
22    "babel-plugin-add-module-exports": "^0.2.1",
23    "babel-plugin-transform-flow-strip-types": "^6.22.0",
24    "babel-plugin-transform-object-rest-spread": "^6.23.0",
25    "babel-preset-env": "1.4.0",
26    "babel-register": "^6.24.1",
27    "chai": "^3.5.0",
28    "eslint": "^3.19.0",
29    "eslint-config-canonical": "^8.2.0",
30    "flow-bin": "^0.45.0",
31    "flow-copy-source": "^1.1.0",
32    "husky": "^0.13.3",
33    "npm-watch": "^0.1.9",
34    "semantic-release": "^6.3.2"
35  },
36  "keywords": [
```

Technical Lag – Example



Technical Lag – Example



Time-based technical lag for deployed release **debug 2.6.9**

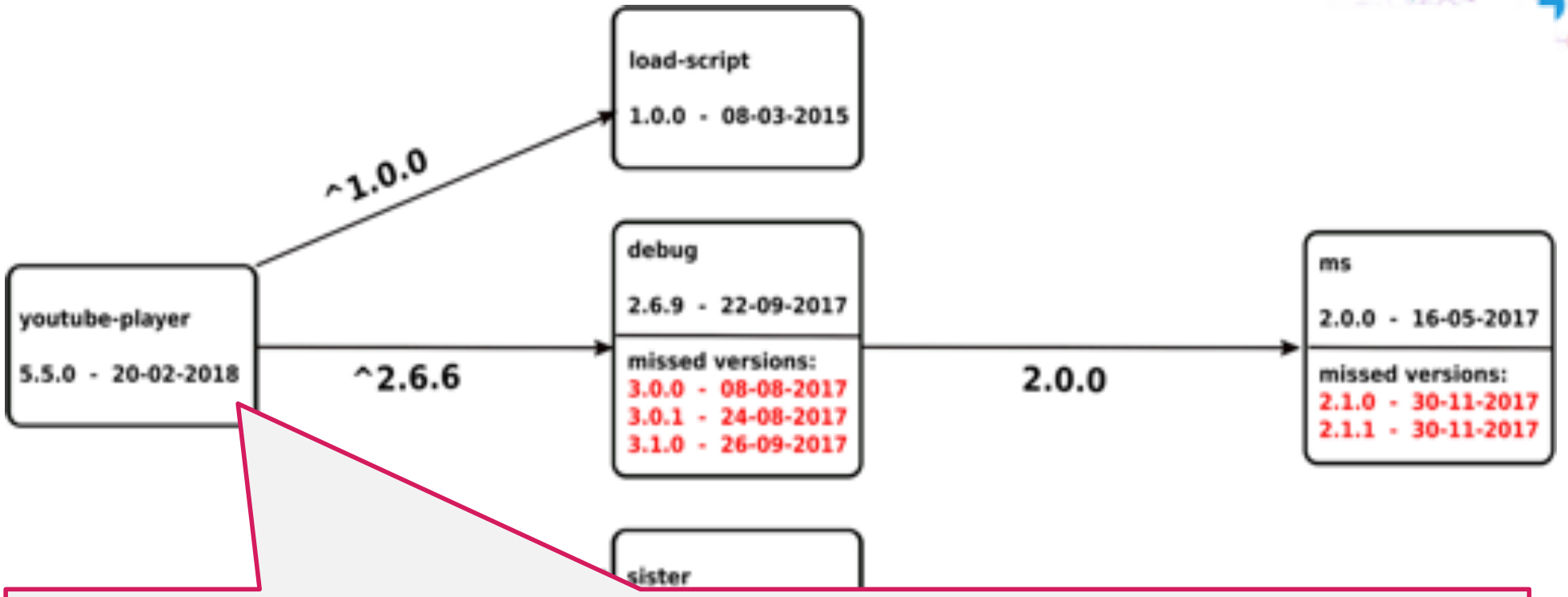
ideal (debug 2.6.9) = debug 3.1.0
timeLag(debug 2.6.9) = 26-09-2017 - 22-09-2017 = 4 days
versionLag(debug 2.6.9) = 1 major + 1 minor + 1 patch

Technical Lag – Example

Time-based technical lag for deployed release **ms 2.0.0**

ideal (ms 2.0.0) = ms 2.1.1
timeLag(ms 2.0.0) = 30-11-2017 - 16-05-2017 = 198 days
versionLag(ms 2.0.0) = 1 minor + 1 patch

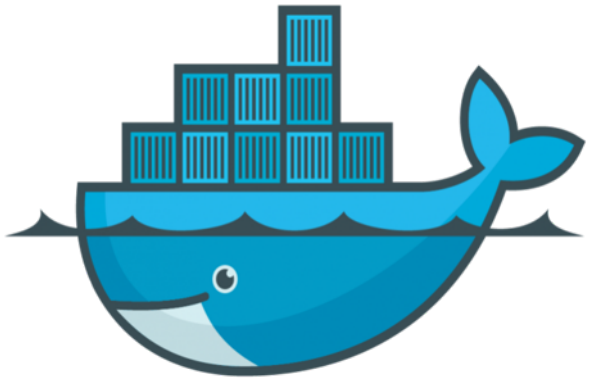
Technical Lag – Example



Aggregated transitive time lag for deployed release **youtube-player 5.5.0**

agglag({**debug 2.6.9, ms 2.0.0**}) = ***max***(**4 days, 198 days**) **= 198 days**

Case study 2: Technical lag in Debian-based Docker containers



docker

debian

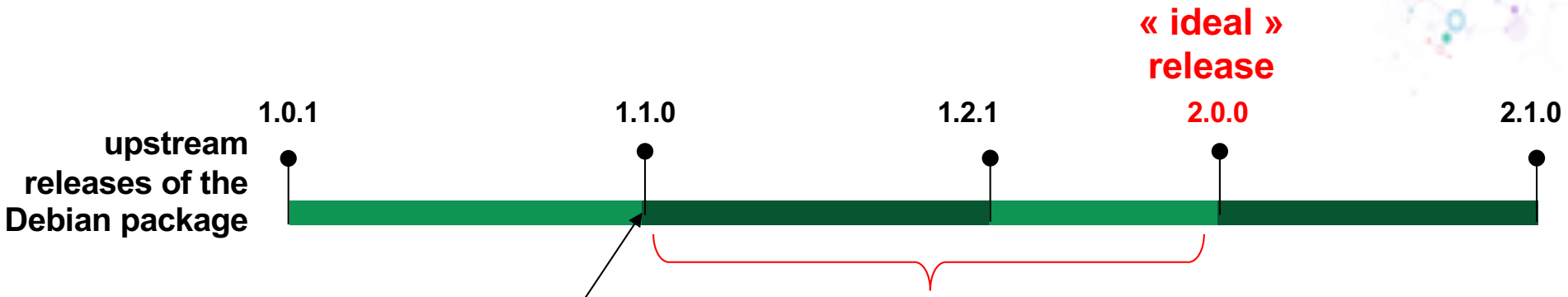
A. Zerouali, T. Mens, G. Robles, J. Gonzalez-Barahona (2019). On the relation between outdated Docker containers, security vulnerabilities, and bugs. IEEE In'tl Conf. SANER

Case study 2: Technical lag in Debian-based Docker containers

Important issues faced when deploying Docker containers:

- Security vulnerabilities
- Dependence on external software packages
- Presence of bugs in third-party software
- Outdated third-party software

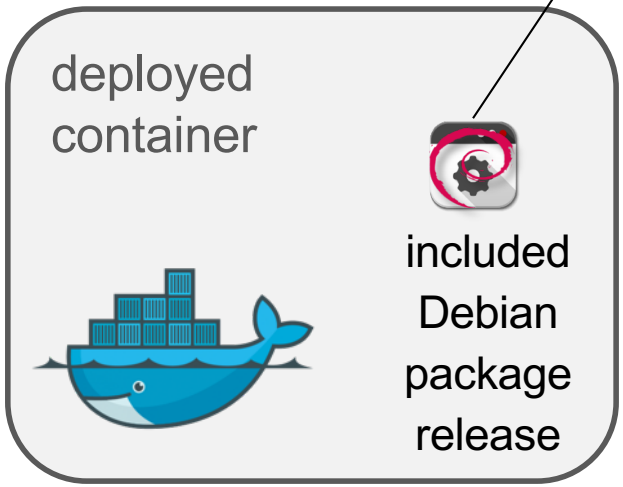
Technical Lag in Debian-based Docker containers



Δ versions (freshness)

Δ vulnerabilities (security)

Δ bugs (stability)



Summary

> If you can't **measure** it
> you can't **manage** it
>
> Peter Drucker

Technical Lag is a very useful generic measure for assessing to which extent deployed software is outdated w.r.t. upstream releases.

- Different ways to measure (time, version, bugs, vulnerabilities, …) and aggregate (max, sum, …) technical lag
- It can be operationalized in different contexts (package dependency management, container deployment, …)

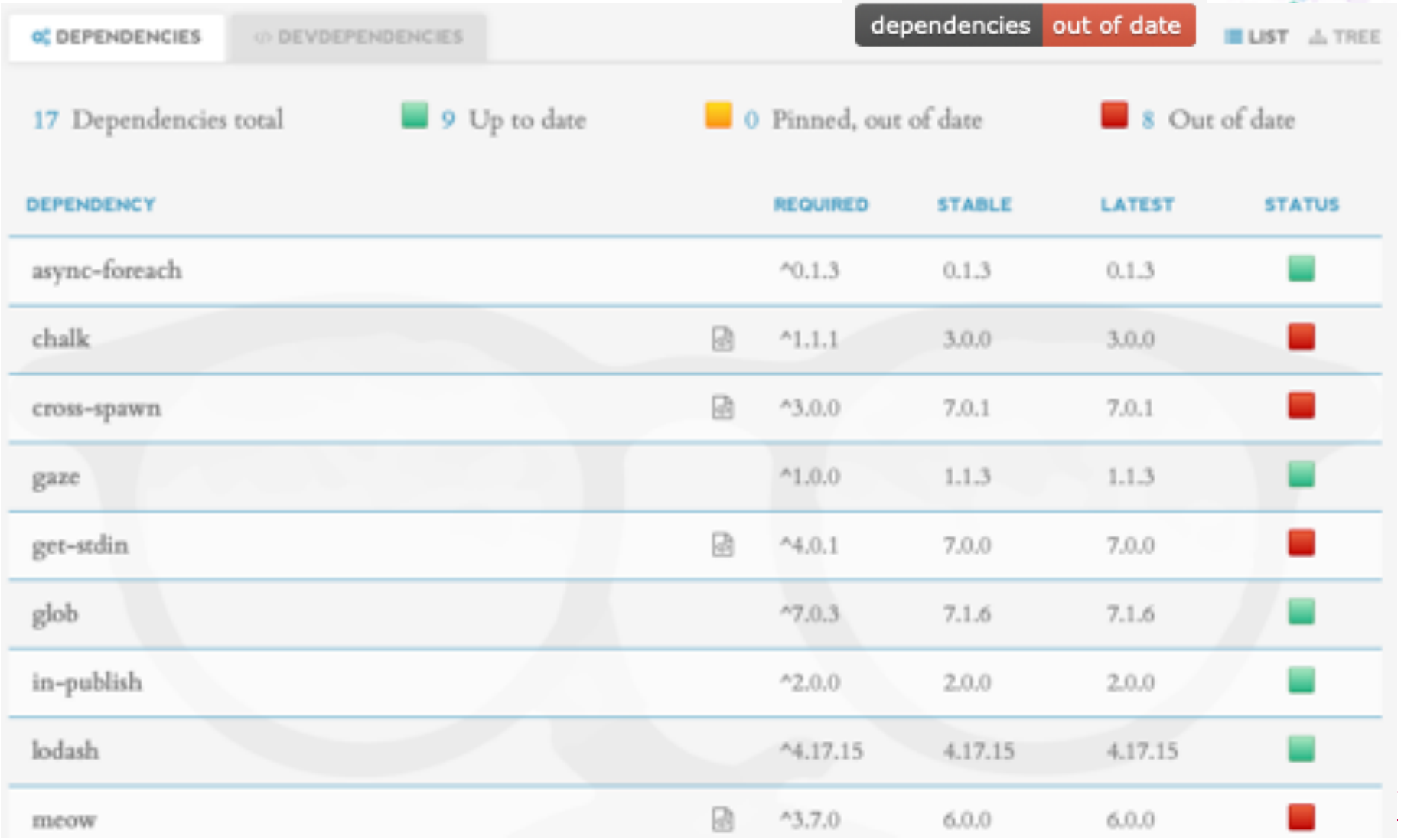
Suggestion:

- Include this measure as part of the CHAOSS Metrics and Tooling

Open Challenges:

- How to measure effort required to update?
- How to combine multiple dimensions of technical lag?
- How to assess whether updates do not cause breaking changes?

Tool support
Example: david-dm.org



New proposed CHAOSS project metrics

- **Dependencies**
  - Number of / List of; Direct or transitive
- **Dependency depth**
- **Outdated dependencies**
  - List of / Number of / Ratio of
- **Vulnerable dependencies**
  - List of / Number of / Ratio of
- **Dependents** (i.e. reverse dependencies)
  - Number of / List of; Direct or transitive
- **Dependency lag**
  - aggregated dependency-based technical lag of a project

chaoss.community