Comparing dependency issues across software package distributions

Tom Mens

Software Engineering Lab

Faculty of Sciences

UMONS
Université de Mons

FOSDEM'20
Brussels / 1 & 2 February 2020

@tom_mens

tom.mens@umons.ac.be

UMONS
Université de Mons

"Excellence of Science" Research Project
2018-2021

VUB
VRIJE UNIVERSITEIT BRUSSEL

UNIVERSITÉ DE NAMUR

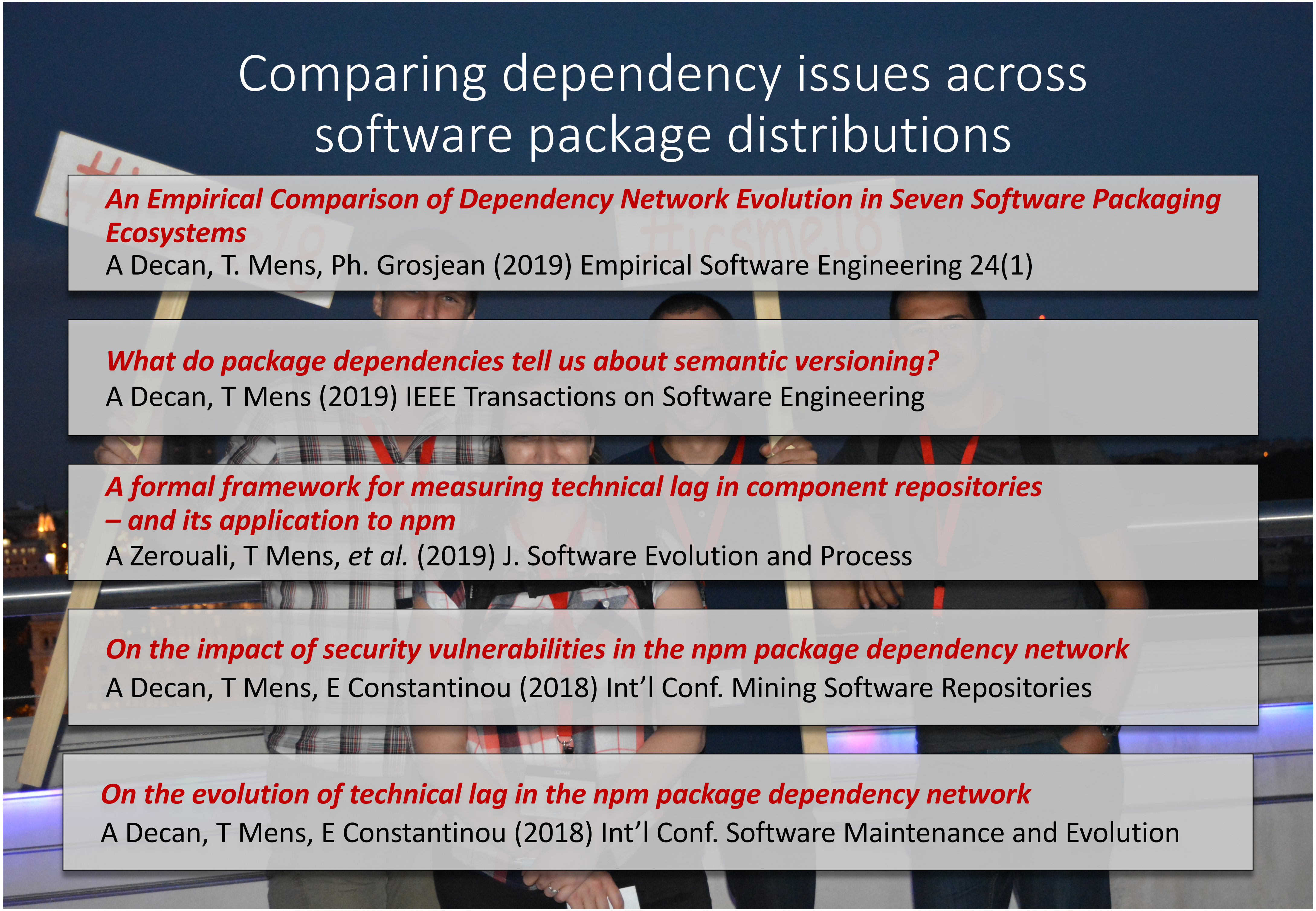
@secoassist

secoassist.github.io

Universiteit Antwerpen

fnrs
LA LIBERTÉ DE CHERCHER

fwo



Comparing dependency issues across software package distributions

An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems
A Decan, T. Mens, Ph. Grosjean (2019) Empirical Software Engineering 24(1)

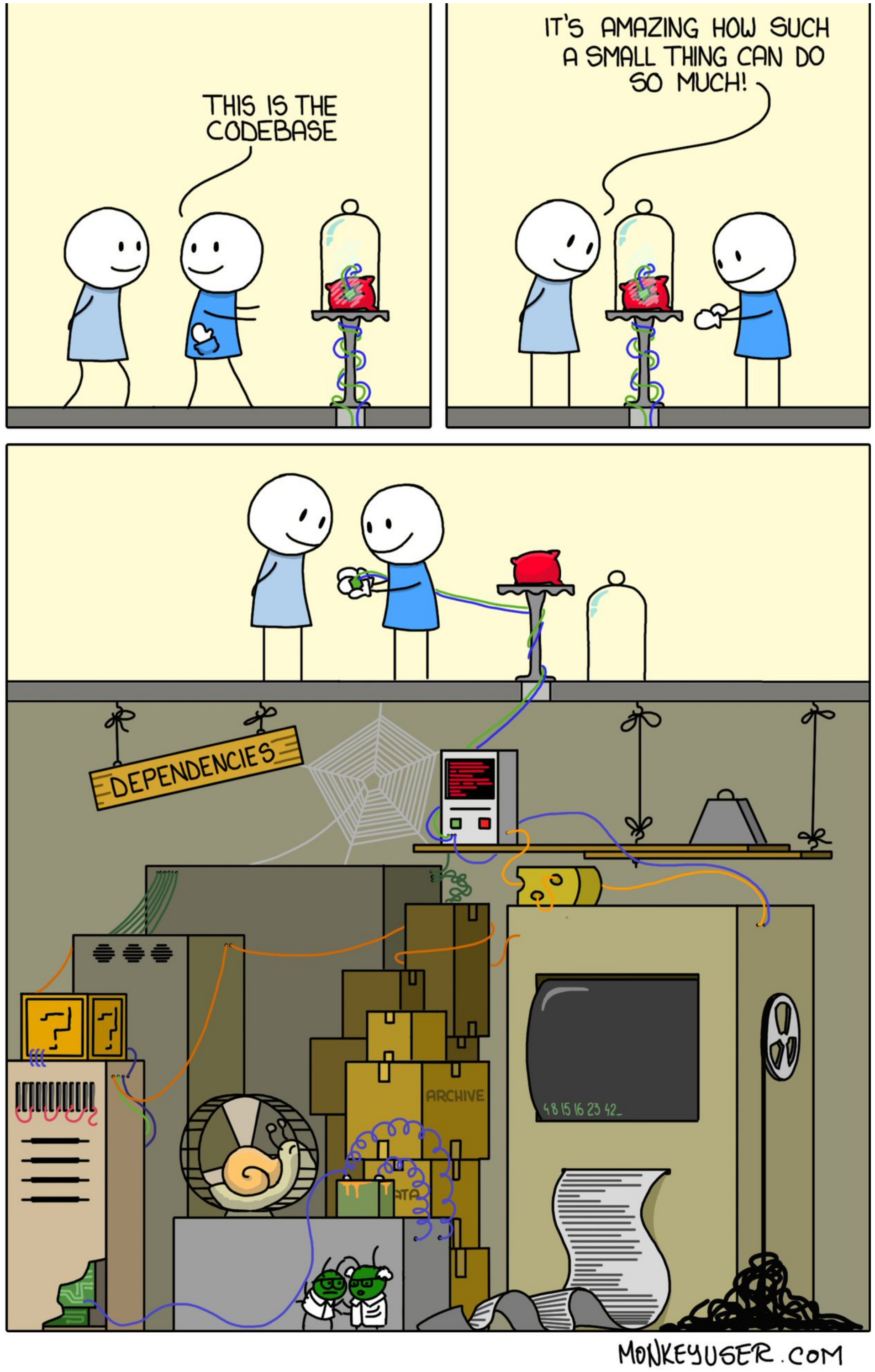
What do package dependencies tell us about semantic versioning?
A Decan, T Mens (2019) IEEE Transactions on Software Engineering

A formal framework for measuring technical lag in component repositories – and its application to npm
A Zerouali, T Mens, *et al.* (2019) J. Software Evolution and Process

On the impact of security vulnerabilities in the npm package dependency network
A Decan, T Mens, E Constantinou (2018) Int’l Conf. Mining Software Repositories

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

Dependency issues



Dependency issues

"Technical lag" due to outdated dependencies

Missed opportunities to benefit from new functionality, or fixes of known bugs and security vulnerabilities

"Dependency hell"

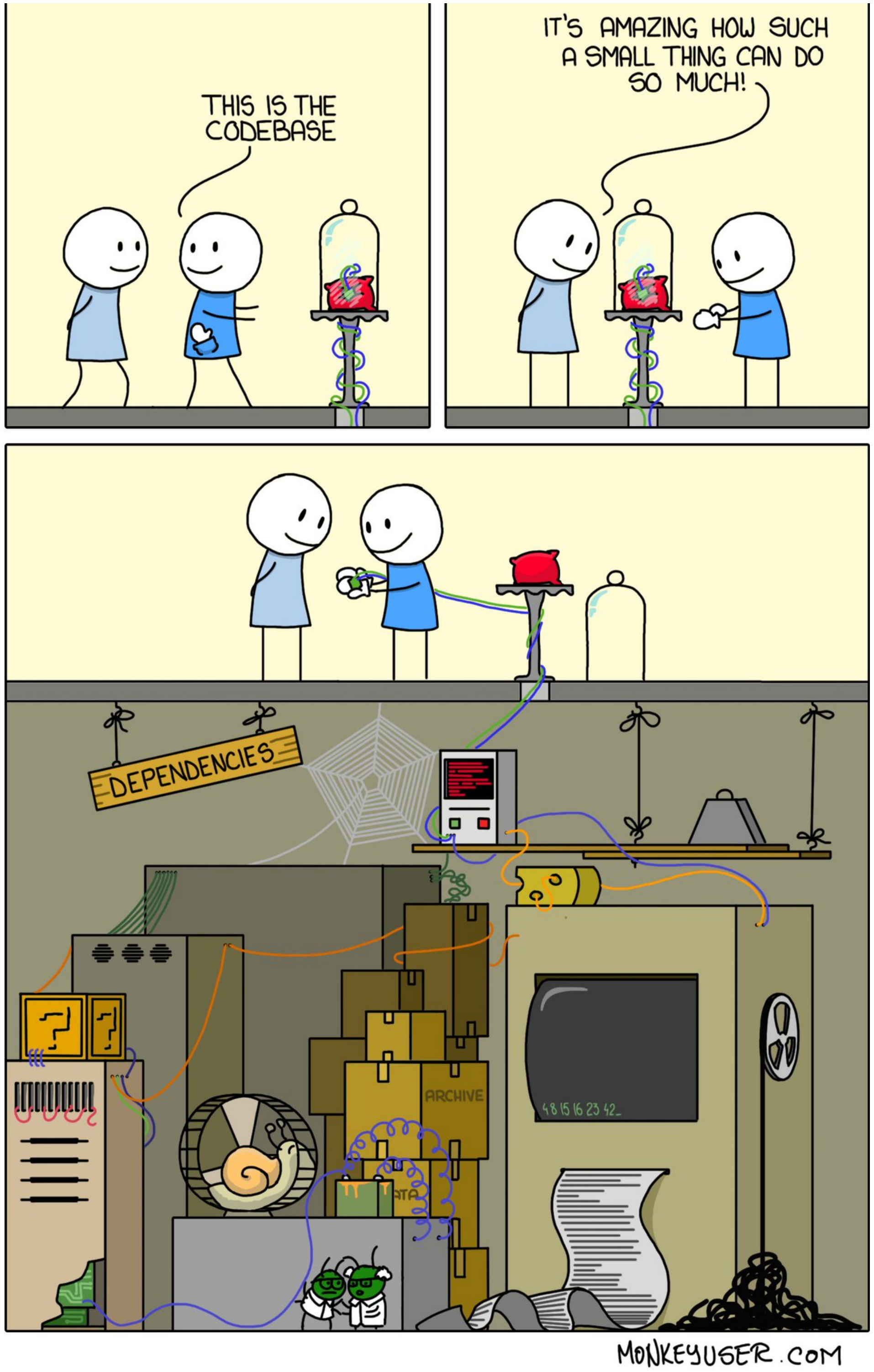
- Too many direct and transitive dependencies
- Broken dependencies due to backward incompatibilities
- Co-installability problems

Unmaintained packages

due to departure of maintainers

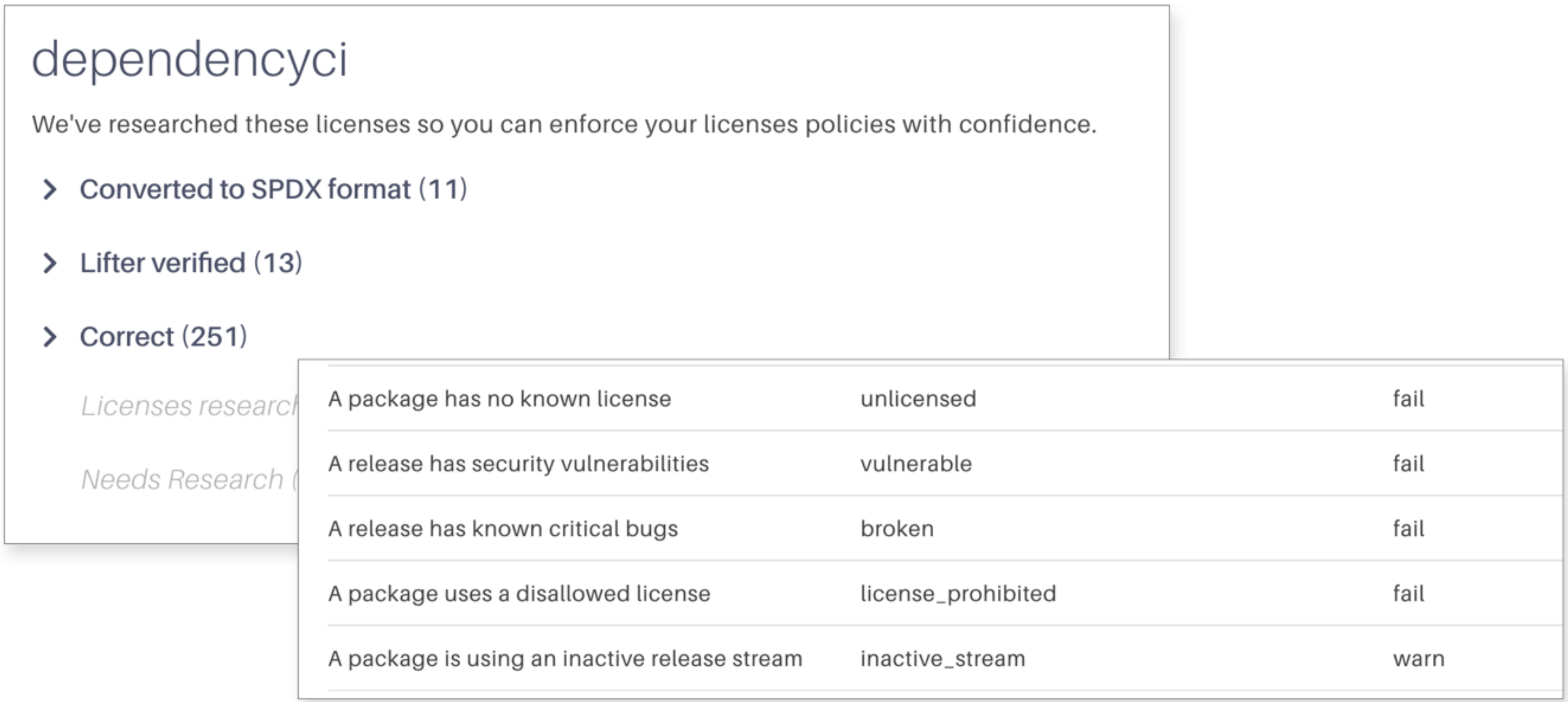
Nontransparent update policies

Incompatible or prohibited licenses



Incompatible licenses

https://tidelift.com



Fragility due to transitive dependencies

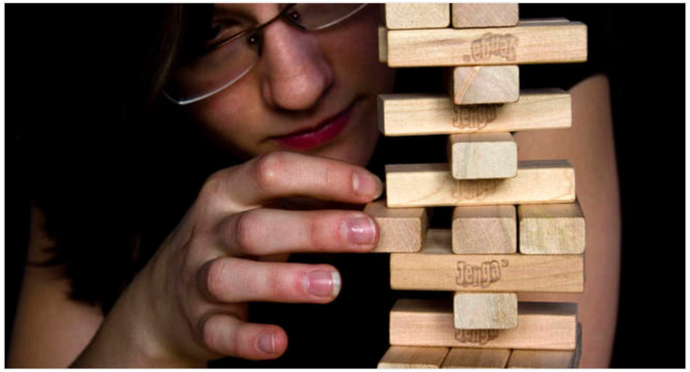
March 2016

Unexpected removal of **left-pad** caused > 2% of all packages to become uninstallable (> 5,400 packages)

Software ▸ Developer

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using



Careful, careful ... Don't fumble this like the JS world (Credit: Claus Rebler)

23 Mar 2016 at 01:24, Chris Williams 1320

November 2010

Release 0.5.0 of **i18n** broke dependent package **ActiveRecord** that was transitively required by >5% of all packages

Libraries.io monitors **6,901,989** open source packages across **37** different package managers

Go 1.82M Packages	npm 1.27M Packages	Packagist 312K Packages	PyPI 231K Packages
NuGet 199K Packages	Maven 185K Packages	Rubygems 161K Packages	Bower 69.7K Packages
CocoaPods 67.8K Packages	WordPress 65.5K Packages	CPAN 37.5K Packages	Cargo 35.4K Packages
Clojars 24.3K Packages	CRAN 16.7K Packages	Hackage 14.5K Packages	Meteor 13.4K Packages
Atom 12.8K Packages	Pub 9.97K Packages	Hex 9.44K Packages	PlatformIO 6.67K Packages
Puppet 6.43K Packages	Emacs 4.87K Packages	Homebrew 4.7K Packages	SwiftPM 4.21K Packages
Carthage 3.87K Packages	Julia 3.05K Packages	Sublime 2.01K Packages	Dub 1.9K Packages
conda 1.85K Packages	Racket 1.67K Packages	Elm 1.51K Packages	Haxelib 1.42K Packages
Nimble 1.19K Packages	Jam 772 Packages	Alcatraz 464 Packages	PureScript 384 Packages
Inqlude 224 Packages	Shards 33 Packages		

https://libraries.io (7 January 2020)

Characterising the evolution of package dependency networks

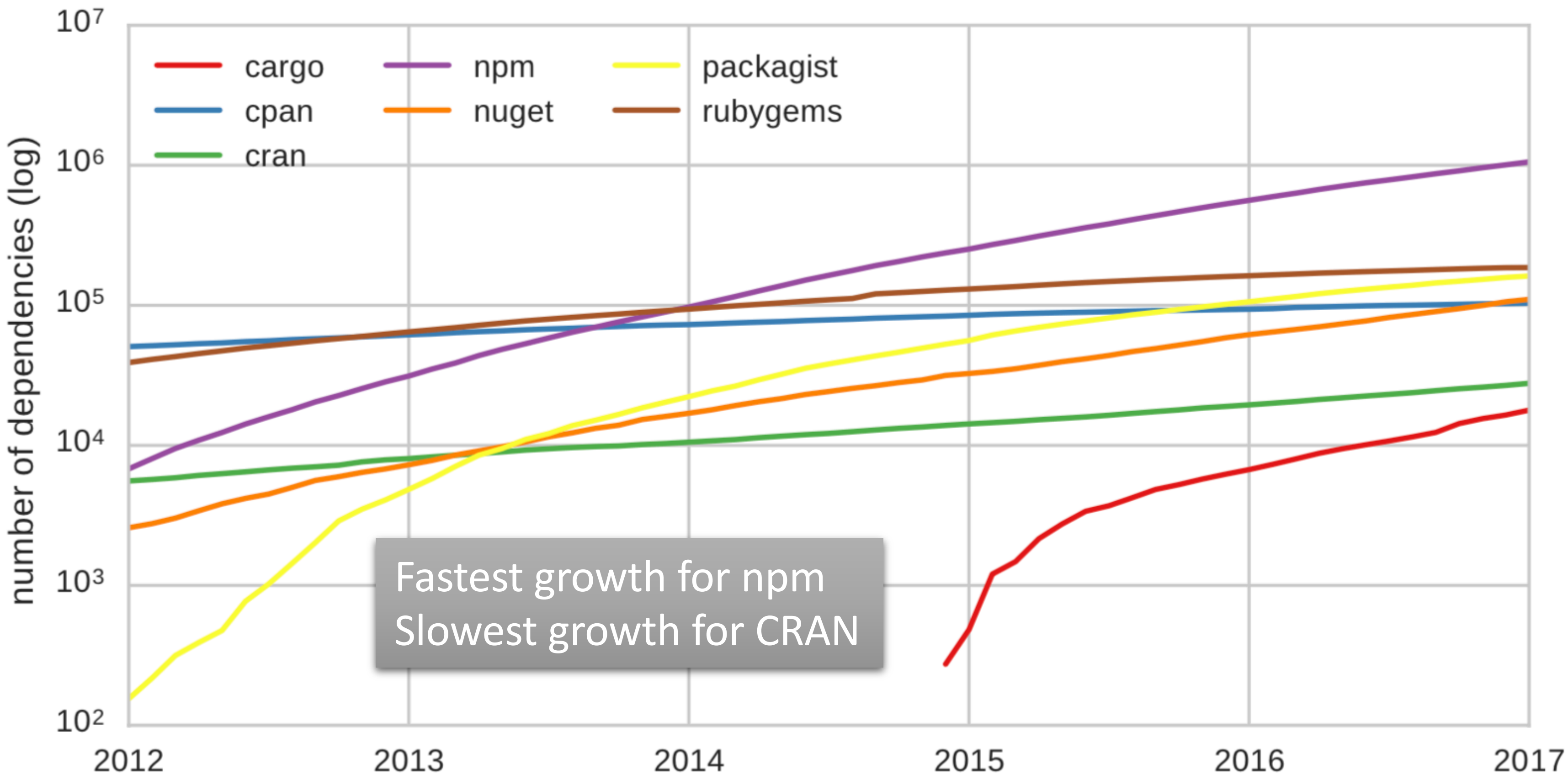
830K packages – 5.8M package versions – 20.5M dependencies (*April 2017*)

Manager	Creation	Lang.	Pkg.	Rel.	Deps.
Cargo	2014	Rust	9k	48k	150k
CPAN	1995	Perl	34k	259k	1,078k
CRAN	1997	R	12k	67k	164k
npm	2010	JavaScript	462k	3,038k	13,611k
NuGet	2010	.NET	84k	936k	1,665k
Packagist	2012	PHP	97k	669k	1,863k
RubyGems	2004	Ruby	132k	795k	1,894k

Decan & Mens (2019) *An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems*. Empirical Software Engineering Journal

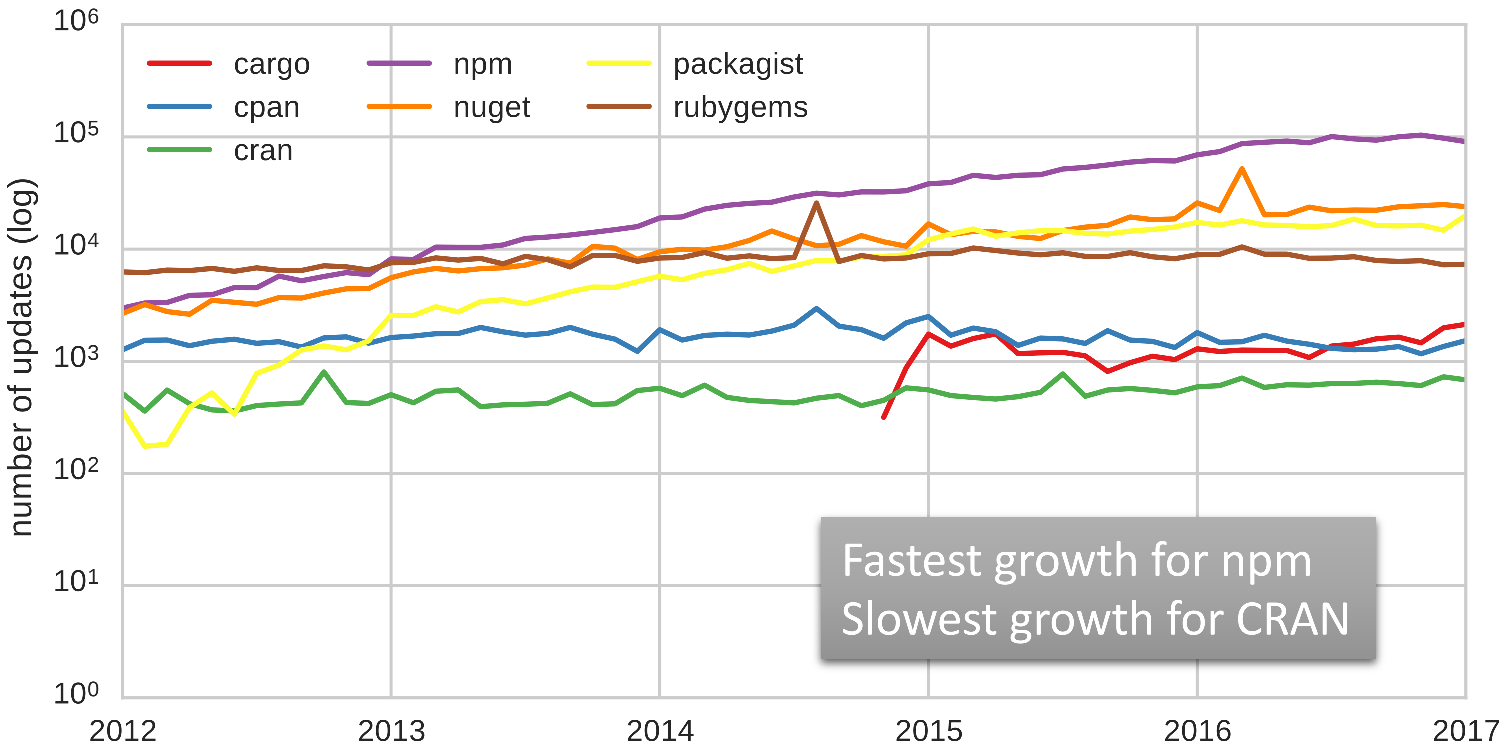
Continuing Growth

Package dependency networks grow **exponentially** in terms of number of packages and/or dependencies



Continuing Change

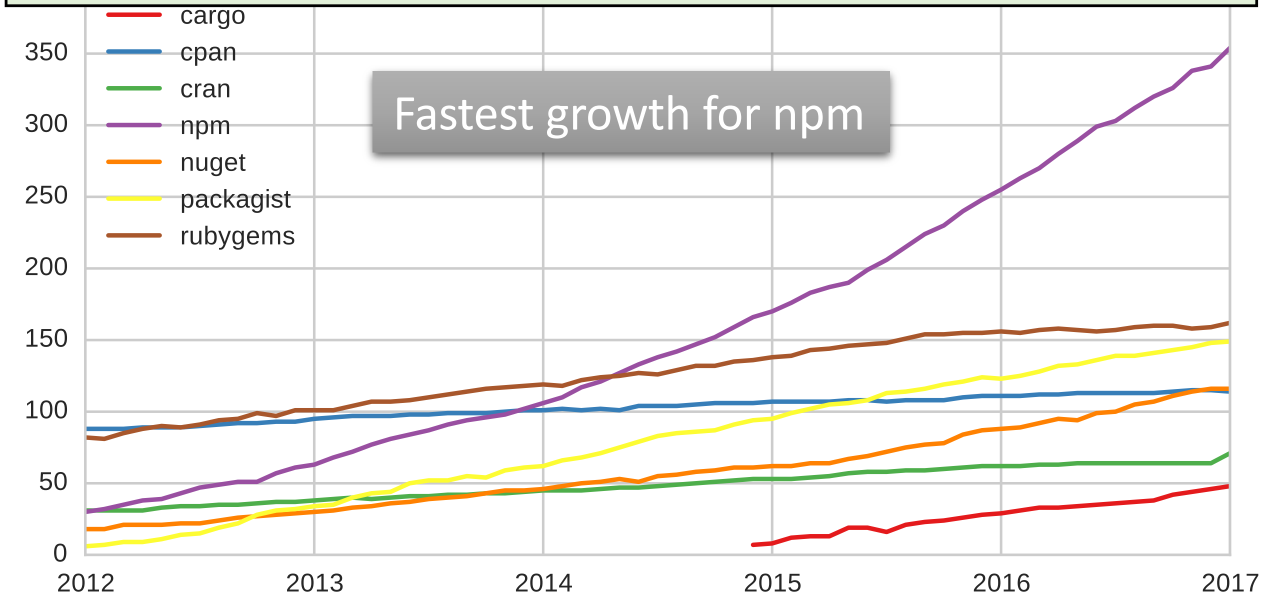
- Number of package updates grows over time
- >50% of package releases are *updated within 2 months*
- *Required* and *young* packages are updated more frequently



Increasing level of reuse

- Highly connected network, containing 60% to 80% of all packages
- Power law behavior: A stable minority (20%) of required packages collect over 80% of all reverse dependencies

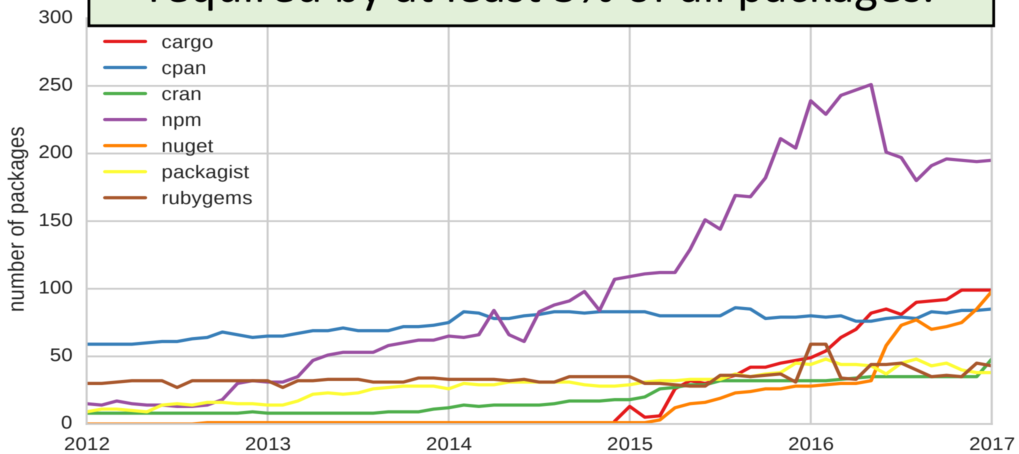
Reusability index: Maximal value n such that there exist n required packages having at least n dependent packages.



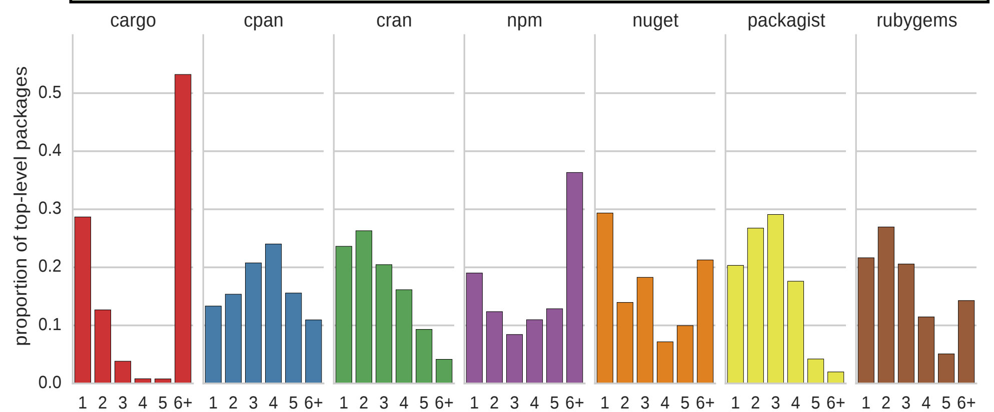
High number of deep transitive dependencies

- Fragile packages may have a very high transitive impact
- Over 50% of top-level packages have a *deep dependency graph*

Number of packages that are ***transitively*** required by at least 5% of all packages.



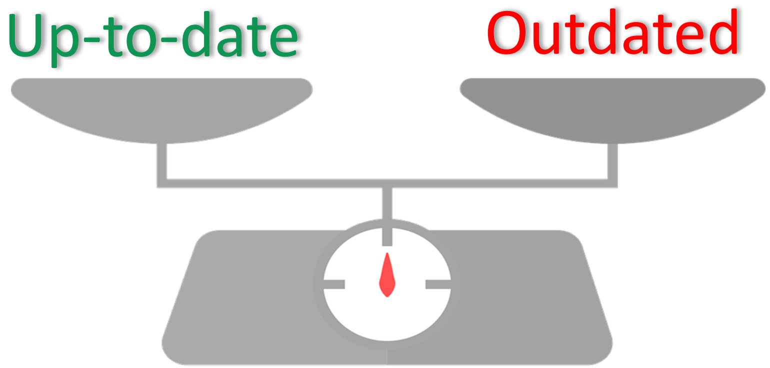
Transitive dependency depth of top-level packages



Outdated Dependencies

Should package maintainers upgrade their dependencies to more recent versions?

- Upgrades benefit from bug and security fixes
- Upgrading allows to use new features
- Upgrading requires effort
- Upgrading may introduce breaking changes

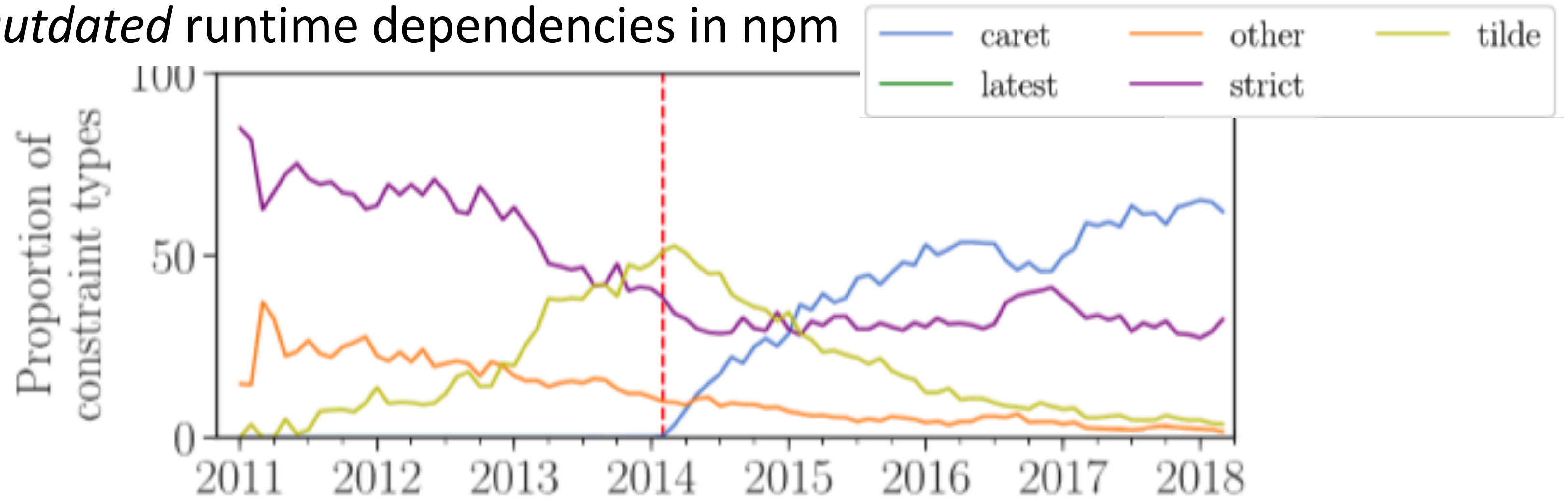


Outdated Dependencies

Outdatedness is related to the type of dependency constraint being used

Strict (i.e. pinned) constraints represent
about 33% of all outdated dependencies

Outdated runtime dependencies in npm



Technical Lag

Technical lag measures how outdated a package or dependency is w.r.t. the "ideal" situation

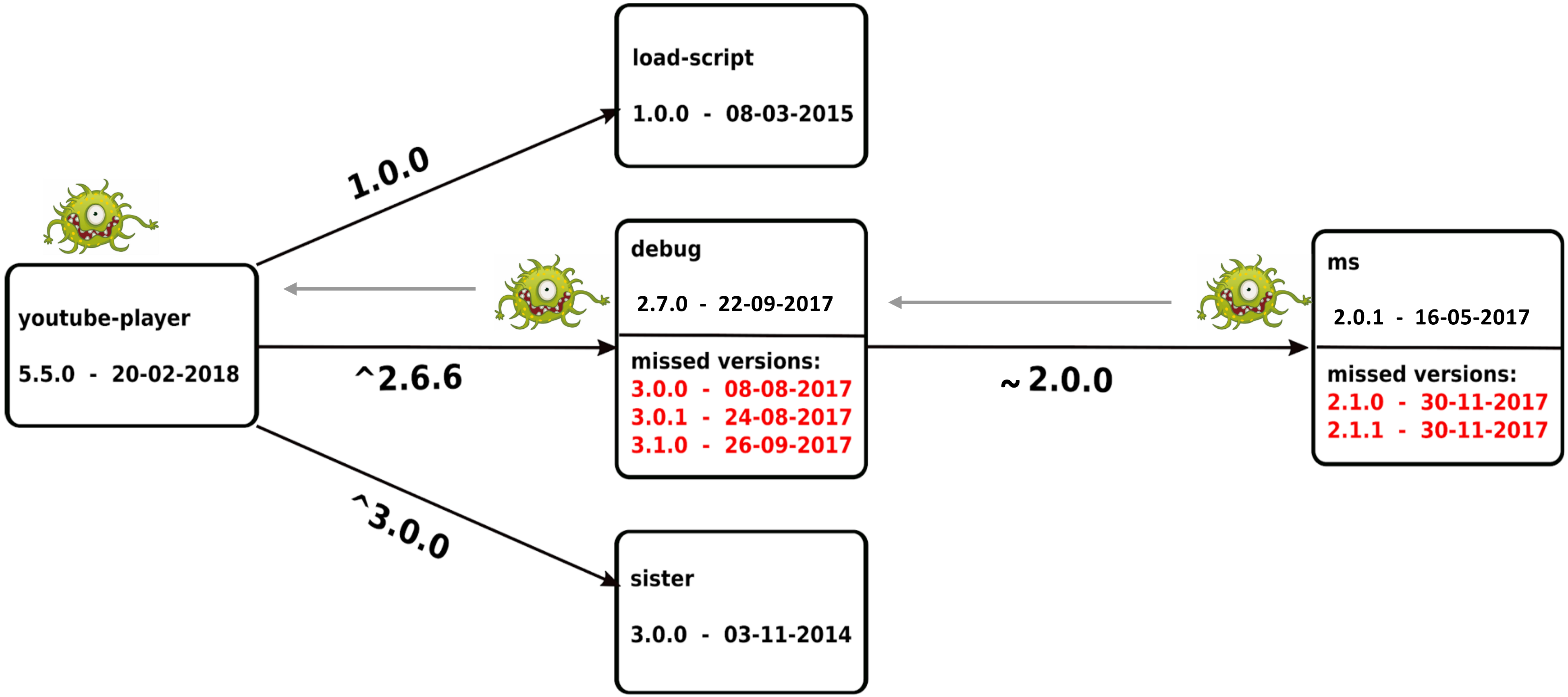
where "ideal" = "most recent"; "most secure"; "least bugs"; "most compatible"; …

A Zerouali *et al* (Feb. 2019) *A formal framework for measuring technical lag in component repositories – and its application to npm.* Wiley Journal on Software Evolution and Process

Technical Lag

Technical lag measures how outdated a package or dependency is w.r.t. the "ideal" situation

where "ideal" = "most recent"; "most secure"; "least bugs"; "most compatible"; …



Need for dependency monitoring tools

Example: David Dependency Manager for npm projects

My npm Project 4.13.1 https://david-dm.org dependencies out of date

Wrapper around libsass

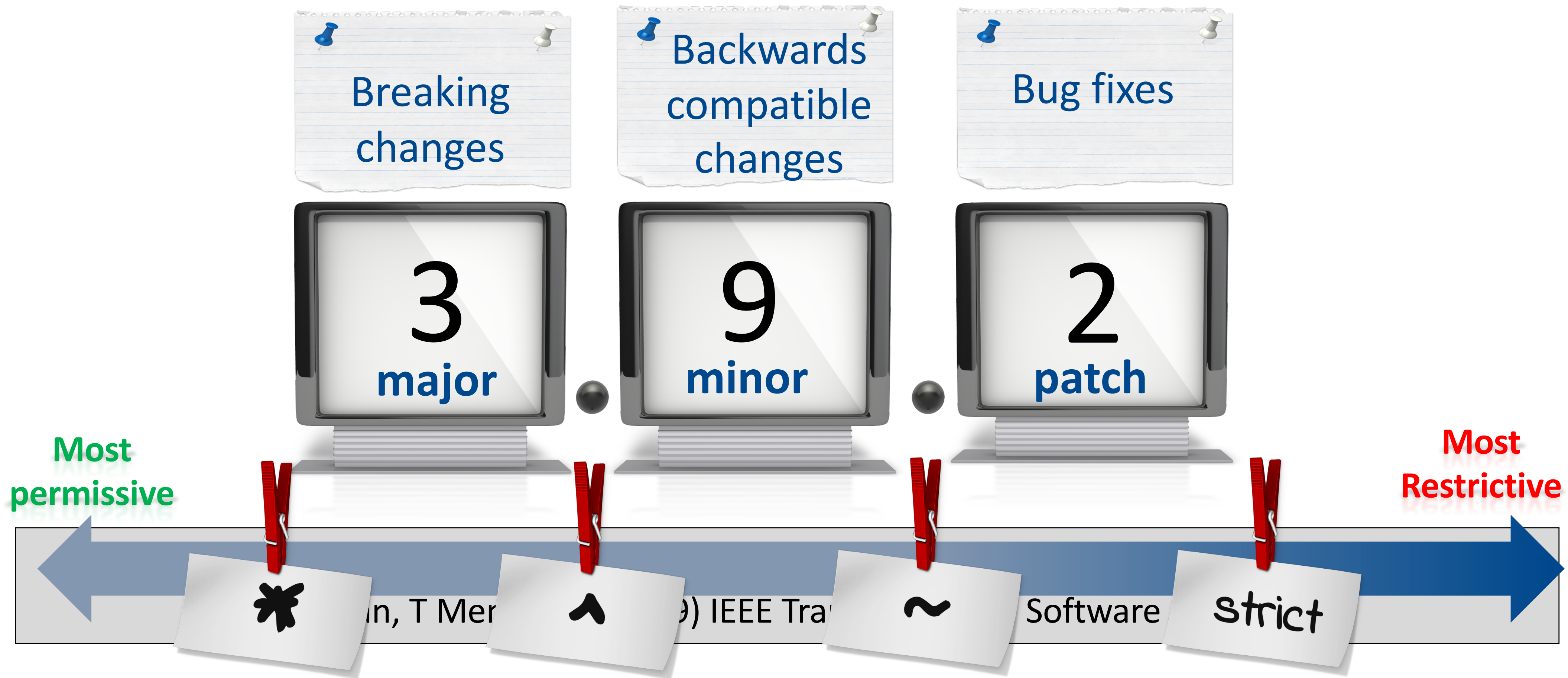
DEPENDENCIES DEVDEPENDENCIES LIST TREE

17 Dependencies total | 9 Up to date | 0 Pinned, out of date | 8 Out of date

DEPENDENCY	REQUIRED	STABLE	LATEST	STATUS
async-foreach	^0.1.3	0.1.3	0.1.3	
chalk	^1.1.1	3.0.0	3.0.0	
cross-spawn	^3.0.0	7.0.1	7.0.1	
gaze	^1.0.0	1.1.3	1.1.3	
get-stdin	^4.0.1	7.0.0	7.0.0	
glob	^7.0.3	7.1.6	7.1.6	

Avoiding breaking changes through Semantic Versioning

Is semantic versioning respected by software package distributions?



Semantic versioning

Different package managers interpret version constraints in different ways:

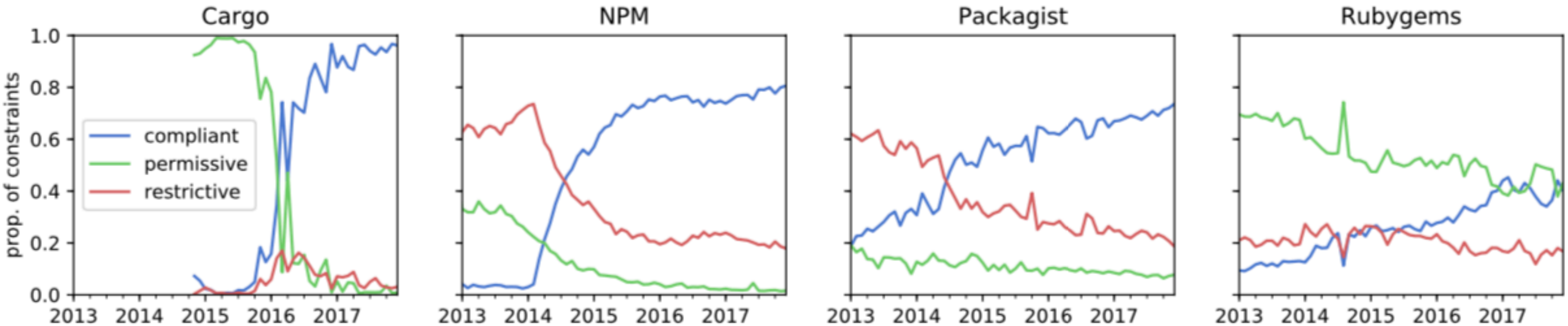
Constr.	Cargo	npm	Packagist	Rubygems
=1.0.0	[1.0.0]	[1.0.0]	[1.0.0]	[1.0.0]
1.0.0	[1.0.0, 2.0.0[	[1.0.0]	[illegible]	[illegible]
1.0	[1.0.0, 2.0.0[	[1.0.0, 1.1.0[	[1.0.0]	[1.0.0]
1	[1.0.0, 2.0.0[	[1.0.0, 2.0.0[	[1.0.0]	[1.0.0]
~1.2.3	[1.2.3, 1.3.0[	[1.2.3, 1.3.0[	[1.2.3, 1.3.0[	[1.2.3, 1.3.0[
~1.2	[1.2.0, 1.3.0[	[1.2.0, 1.3.0[	[1.2.0, 2.0.0[	[1.2.0, 2.0.0[
~1	[1.0.0, 2.0.0[	[1.0.0, 2.0.0[	[1.0.0, 2.0.0[	N/A
^1.2.3	[1.2.3, 2.0.0[	[1.2.3, 2.0.0[	[1.2.3, 2.0.0[	N/A
>1.2.3	]1.2.3, +∞[	]1.2.3, +∞[	]1.2.3, +∞[	]1.2.3, +∞[
~0.1.2	[0.1.2, 0.[illegible]	[illegible]	[illegible]2.0[	[0.1.2, 0.2.0[
^0.1.2	[0.1.2, 0.2.0[	[0.1.2, 0.2.0[	[0.1.2, 0.2.0[	N/A

More ***restrictive*** than semver

More ***permissive*** than semver

Semantic versioning

- Cargo, npm and Packagist are mostly semver-compliant.
  All three are more permissive than semver for 0.y.z versions
- All considered ecosystems become more compliant over time.
- >16% of restrictive constraints in npm, Packagist and Rubygems
  ➔ prevents adoption of backward compatible upgrades



Security vulnerabilities

OWASP Foundation Top 10 Application Security Risks

A9 - Using Components with Known Vulnerabilities

You are likely vulnerable:

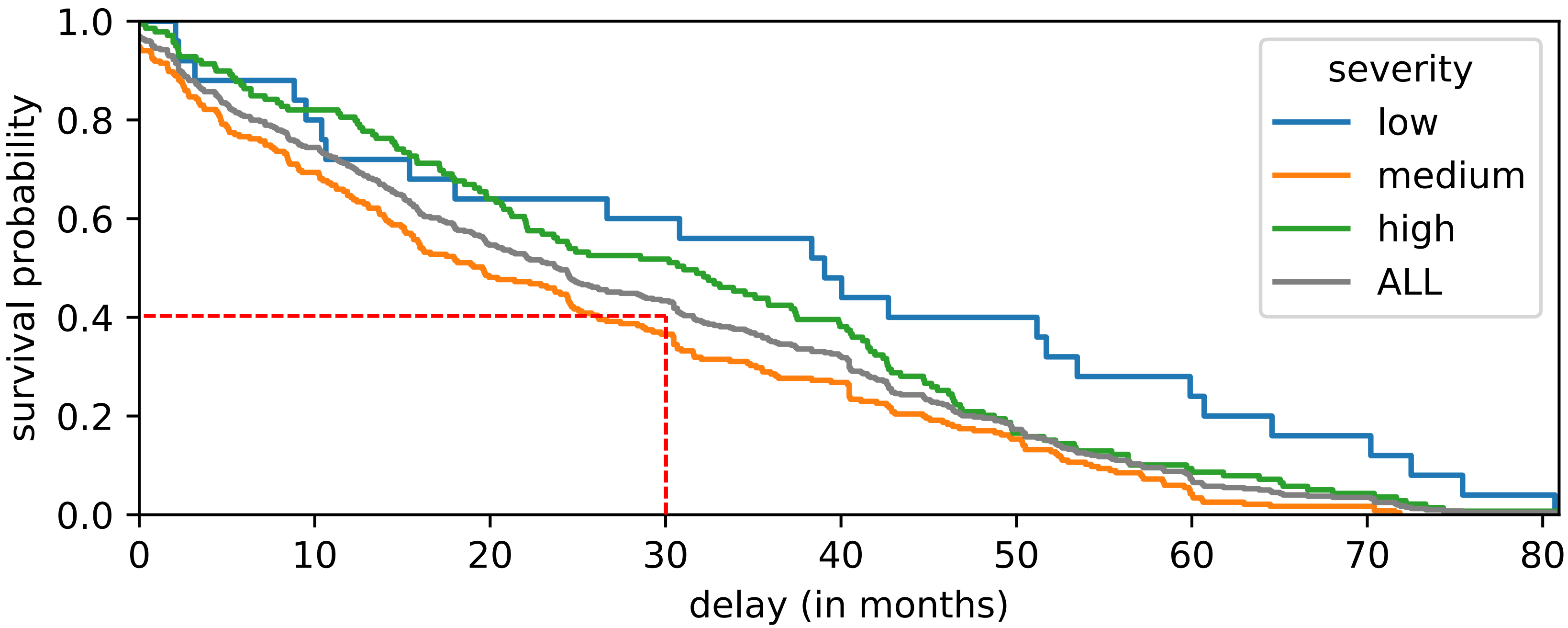
- If you do not know the versions of all components you use … This includes components you directly use as well as nested dependencies.
- If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, which leaves organizations open to many days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.

Security vulnerabilities in npm

Vulnerable packages	
# **vulnerabilities**	**399**
# vulnerable **packages**	**269**
# releases of vulnerable packages	14,931
# vulnerable **releases**	**6,752 (45%)**
# dependent packages	133,602
# **dependent packages** affected by the vulnerable packages	**72,470 (54%)**

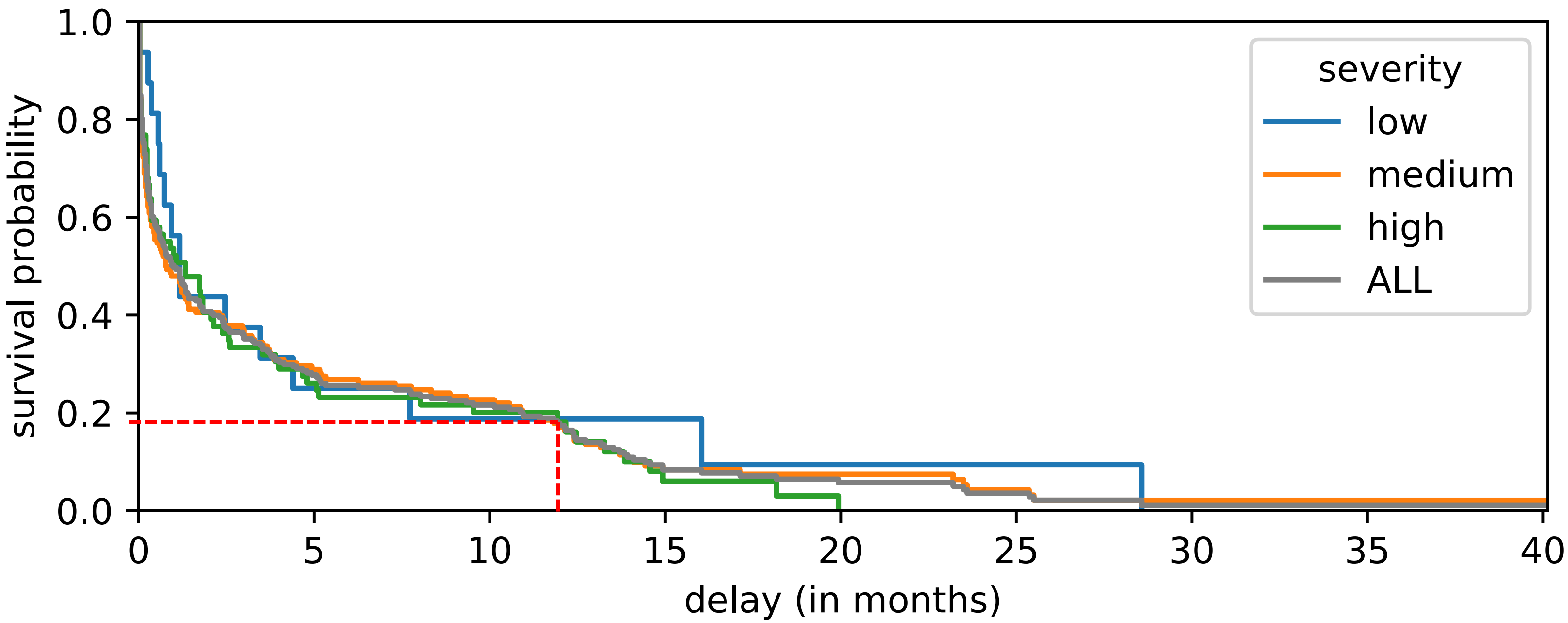
On the impact of security vulnerabilities in the npm package dependency network.
A Decan, T Mens, E Constantinou (2018) Int'l Conf. Mining Software Repositories

Security vulnerabilities in npm
When are vulnerabilities discovered?



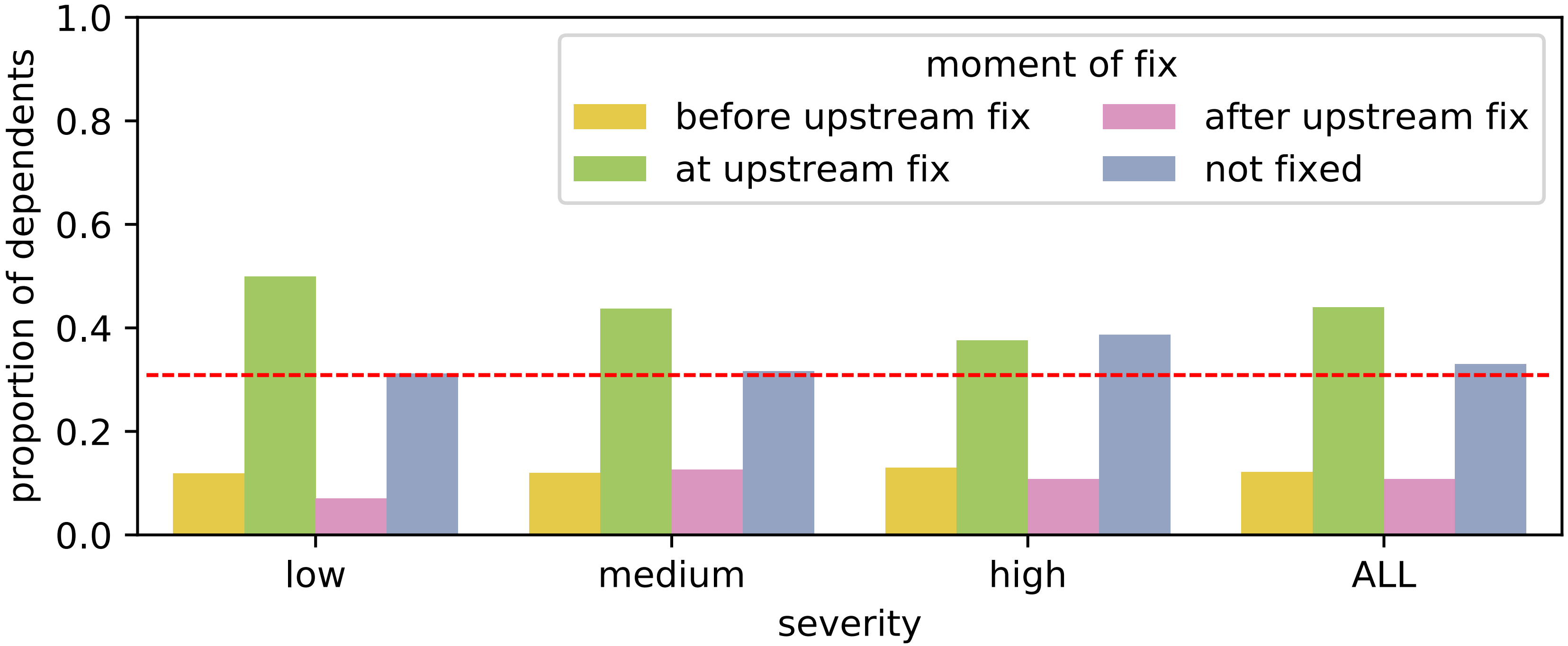
>40% of all vulnerabilities are not ***discovered*** even 2.5 years after their introduction, regardless of their severity.

Security vulnerabilities in npm
When are vulnerabilities fixed?



~20% of vulnerabilities take **more than 1 year** to be fixed.

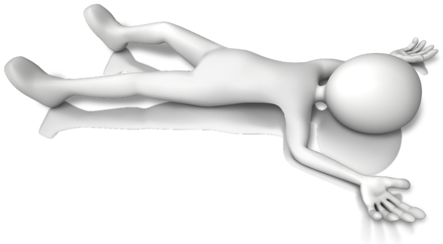
Security vulnerabilities in npm
When are vulnerabilities fixed in dependent packages?



>33% of all affected dependents are not (yet) fixed!

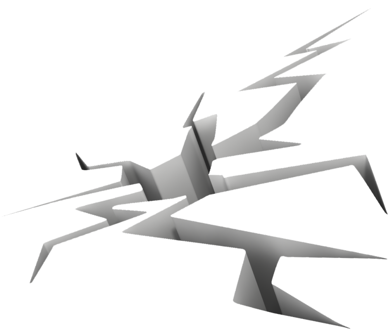
Security vulnerabilities in npm
Why do vulnerabilities remain unfixed in dependent packages?

Improper or too restrictive use of *dependency constraints*



Package is no longer actively maintained

Maintainers are unaware of the vulnerability or how to fix it



Fixed version of the dependency contains incompatible changes

Tool support: Monitor and update vulnerable dependencies

GitHub

Automated security alerts and updates
https://help.github.com/en/github/managing-security-vulnerabilities

Snyk

Continuously find and fix known vulnerabilities in a package's dependencies
https://snyk.io

Retire.js No known vulnerabilities

Scans for the use of JavaScript libraries with known vulnerabilities
http://retirejs.github.io/retire.js/

OWASP Dependency-Check

Detects publicly disclosed vulnerabilities contained within a project's dependencies.
https://github.com/jeremylong/DependencyCheck

Eclipse Steady

Detects known vulnerabilitis in dependencies to open source Java and Python components through combination of static and dynamic analysis techniques
https://eclipse.github.io/steady/

Conclusion

- Package dependency networks are affected my multiple dependency issues
  - Many and deep transitive dependencies
  - Outdated dependencies
  - Breaking changes
  - Vulnerable dependencies
- Automated tools and policies can help mitigating these issues
  - Measuring, monitoring and updating outdated and vulnerable dependencies
  - Supporting semantic versioning
  - Supporting transitive dependencies
  - Detecting vulnerabilities that matter
    (avoid false positives/negatives)