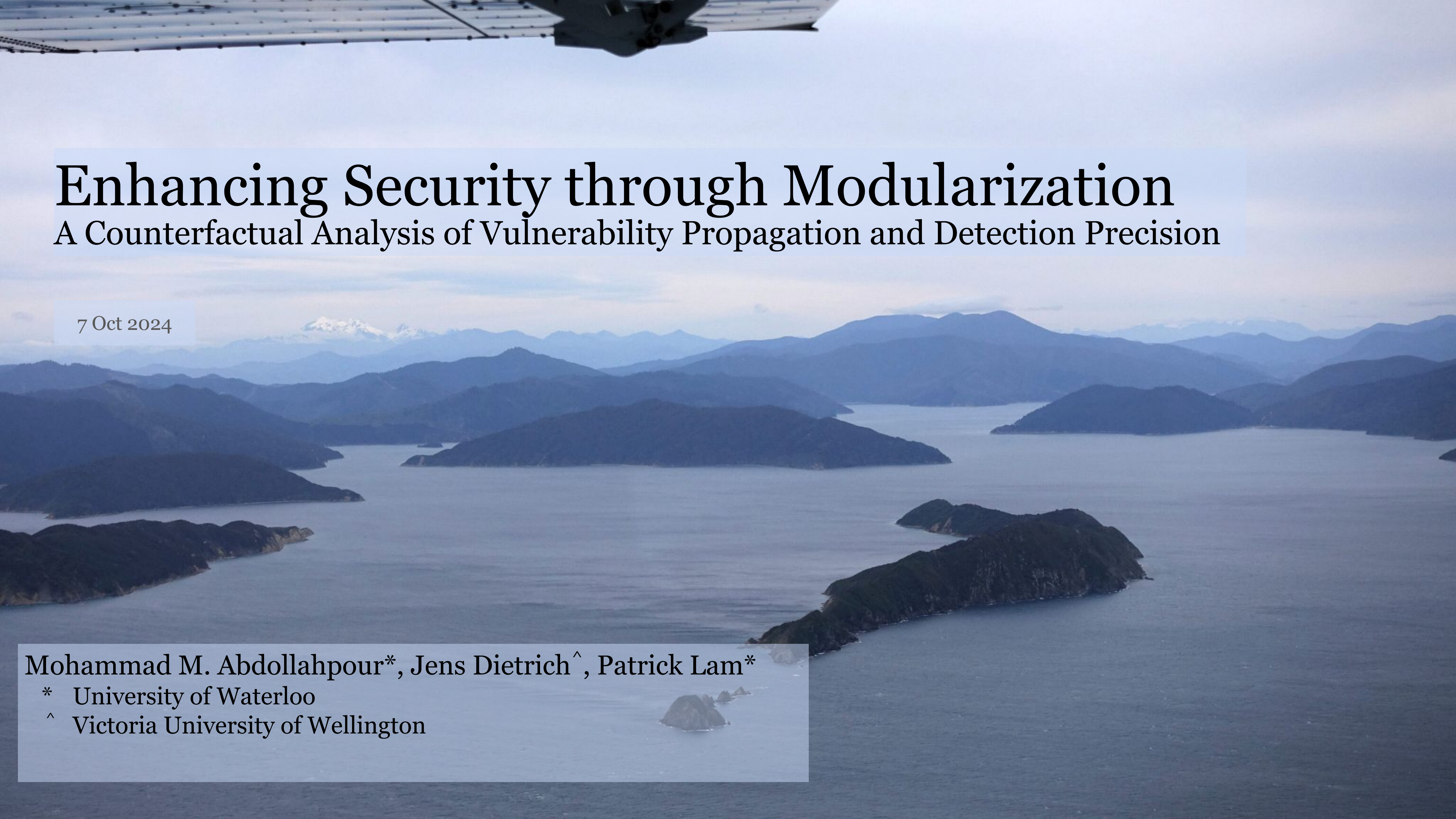


Enhancing Security through Modularization

A Counterfactual Analysis of Vulnerability Propagation and Detection Precision

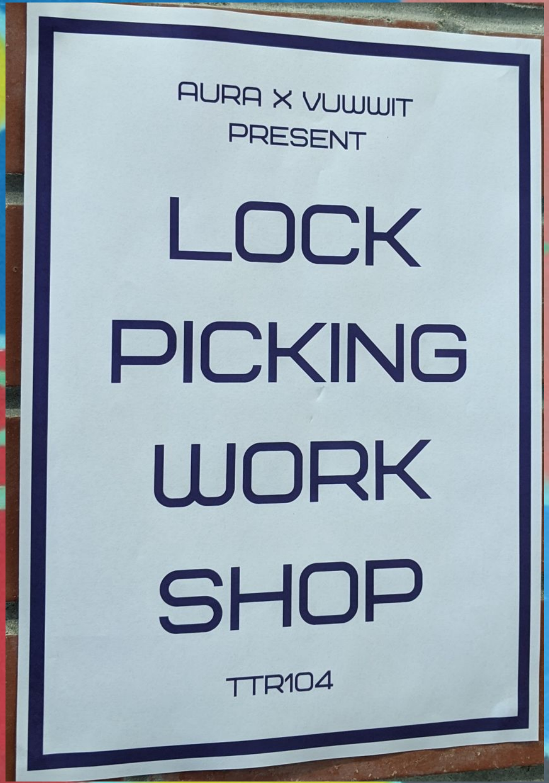
7 Oct 2024

Mohammad M. Abdollahpour*, Jens Dietrich^, Patrick Lam*

* University of Waterloo

^ Victoria University of Wellington

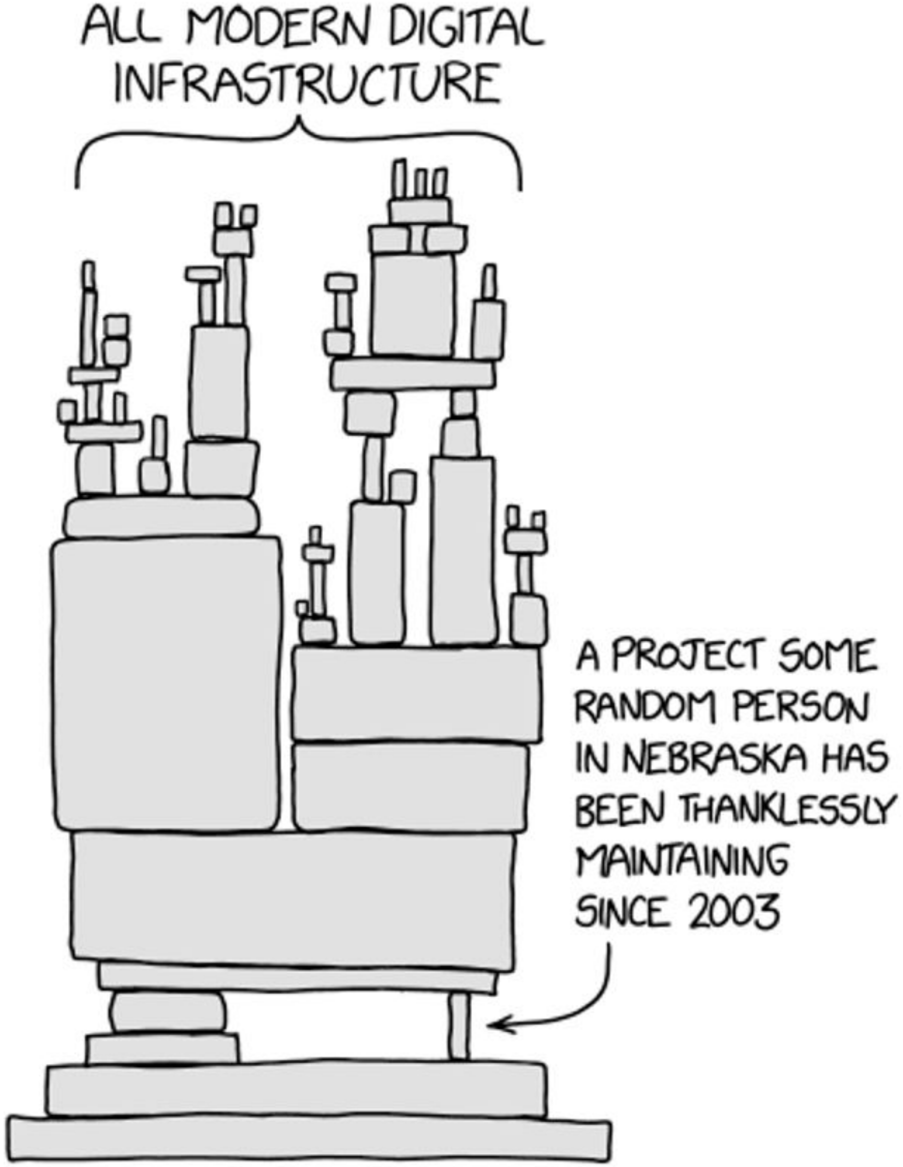
tl;dl: more modular libraries can lead to more secure software



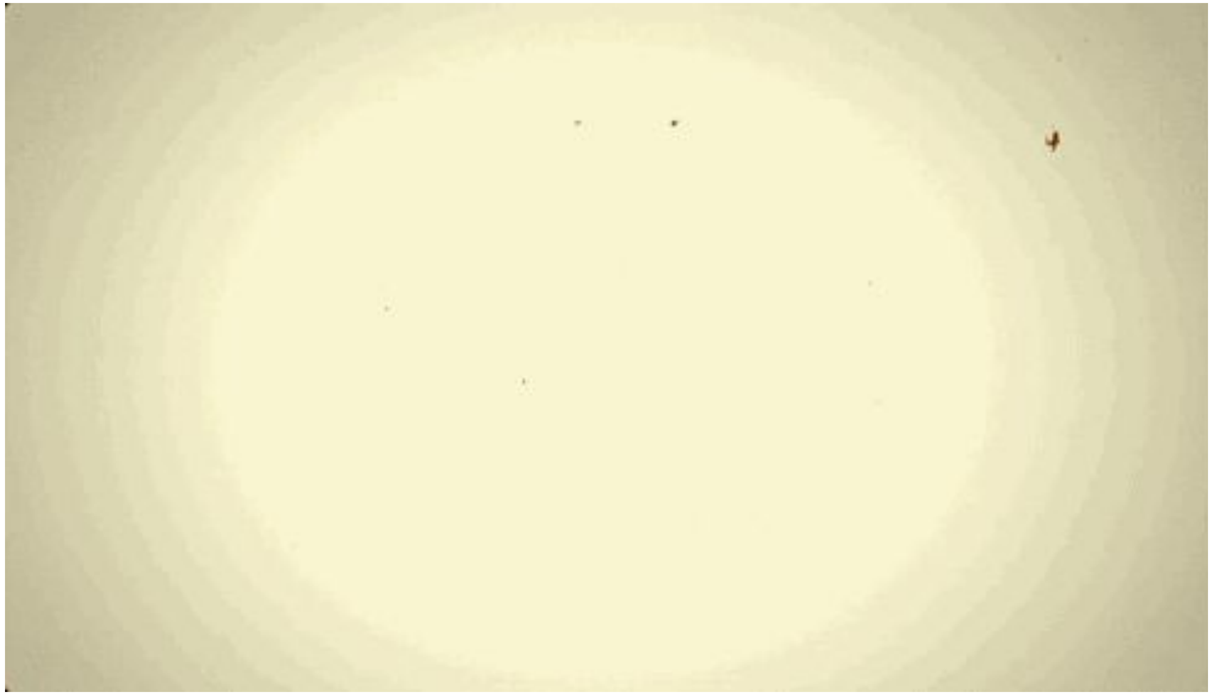
AURA X VUWWIT
PRESENT

LOCK
PICKING
WORK
SHOP

TTR104


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

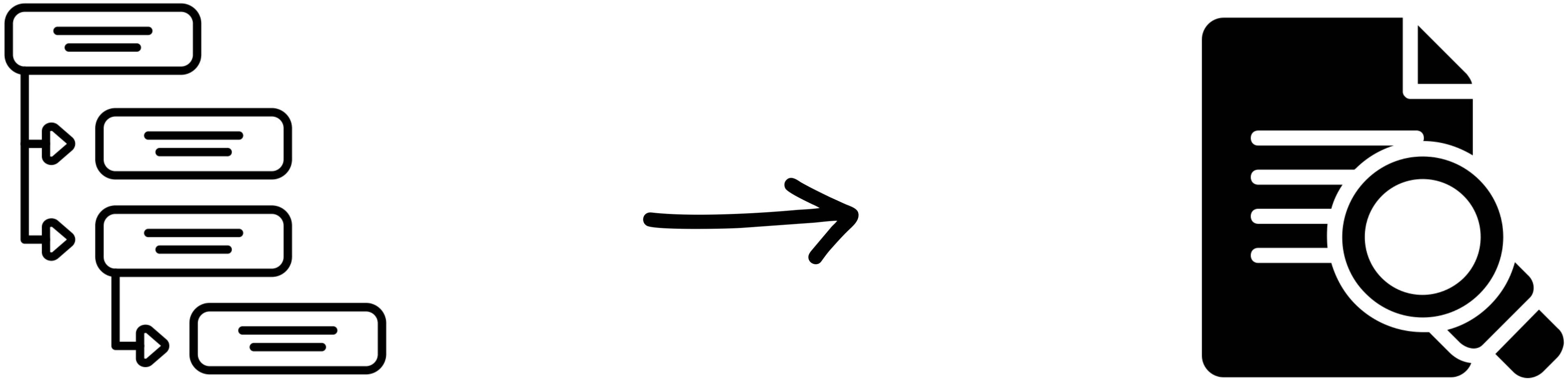
3rd-party libraries are awesome!



But they come at a cost:
Security vulnerabilities!

Software Composition Analysis (SCA) to the rescue

Software Composition Analysis (SCA) to the rescue



Dependency Tree

CVE Database

Software Composition Analysis (SCA) ~~to the rescue~~

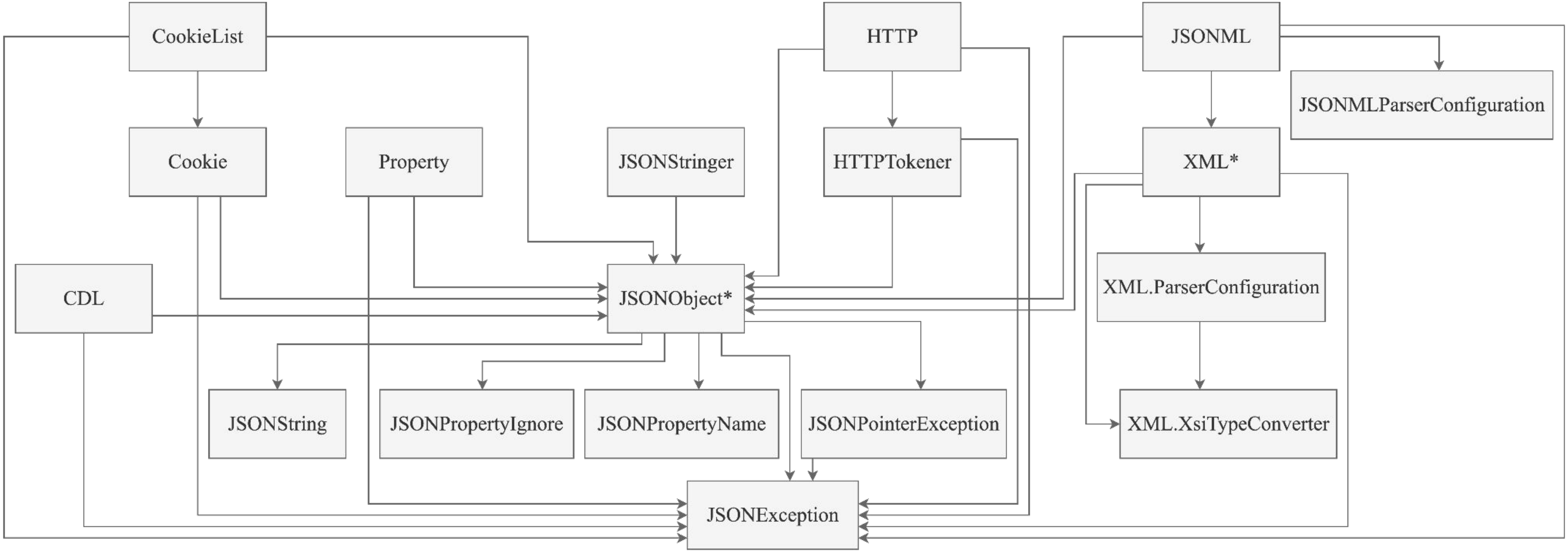
Software Composition Analysis (SCA) ~~to the rescue~~

Too many
false positives

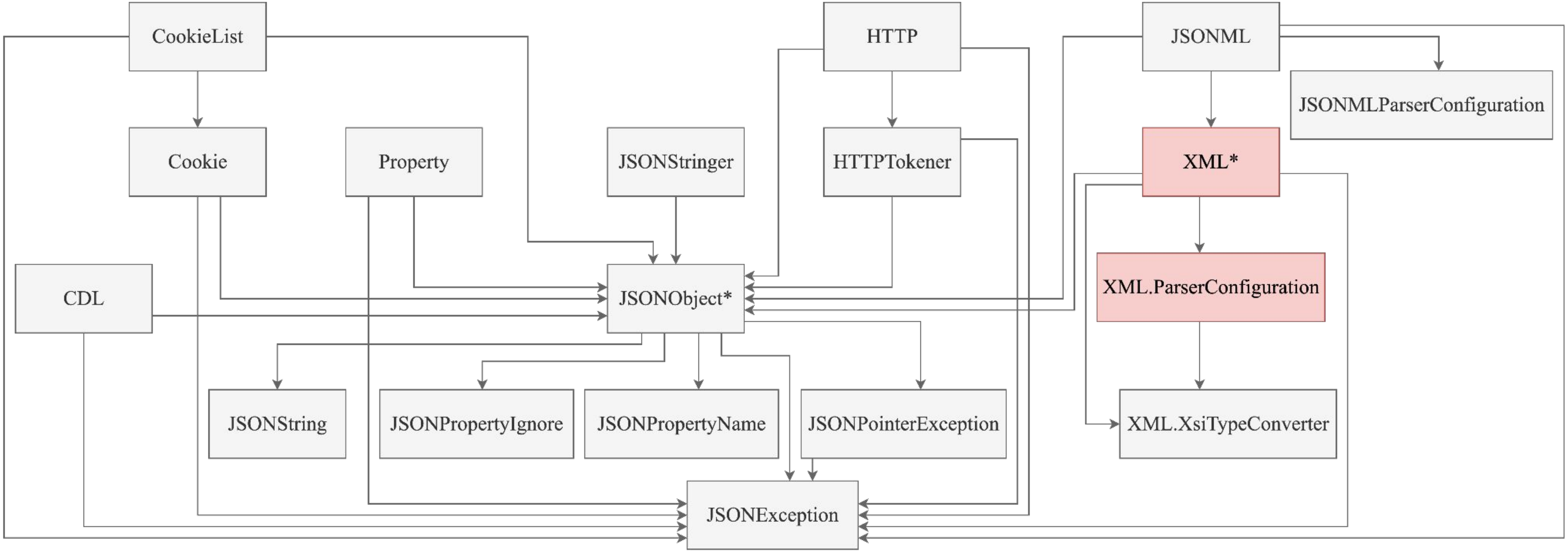
Call graph
analysis is
expensive

Rely on
reported CVEs

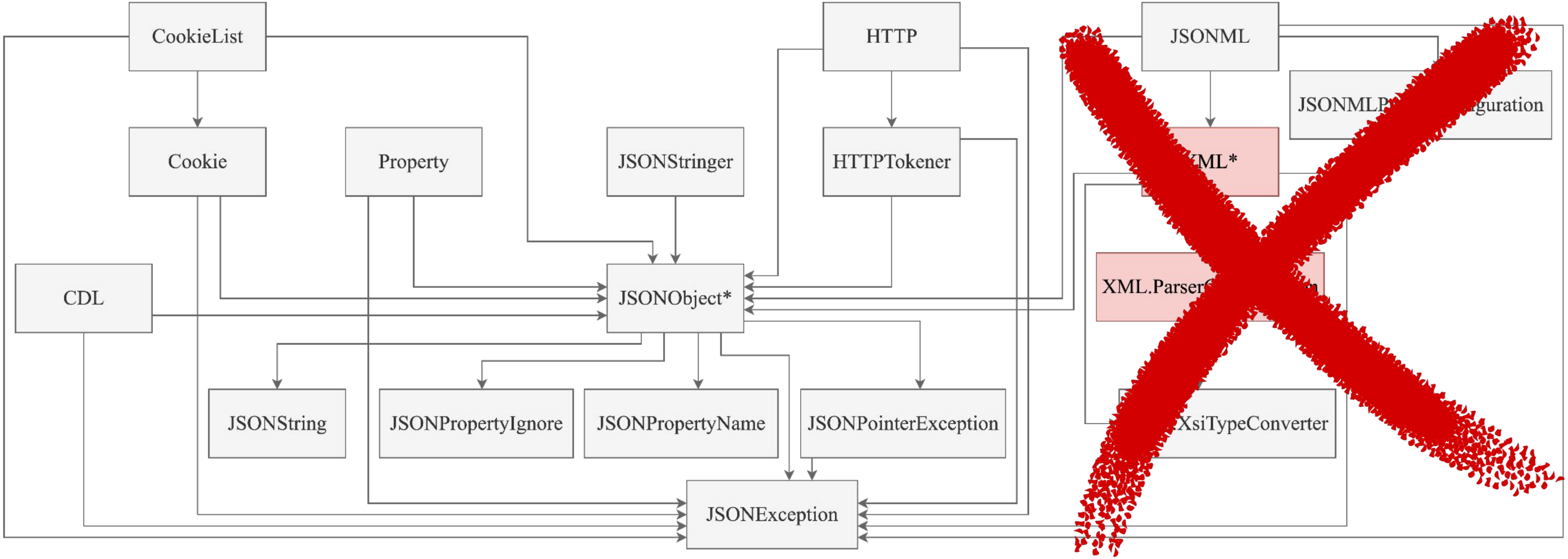
pkg:mvn/org.json/json used by >1k other libraries



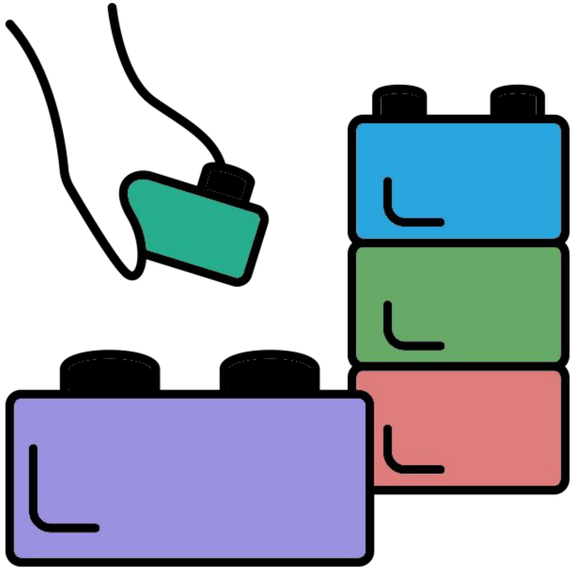
CVE-2022-45688: vulnerability in the XML transformer



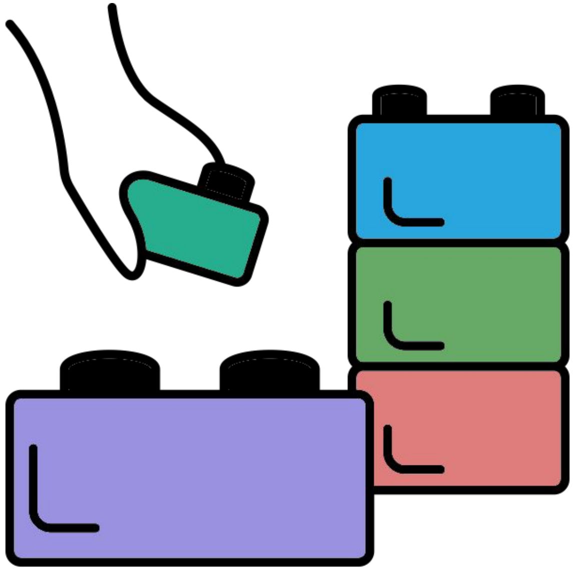
But I don’t need the XML stuff!



What if clients *could* reference only what they need?



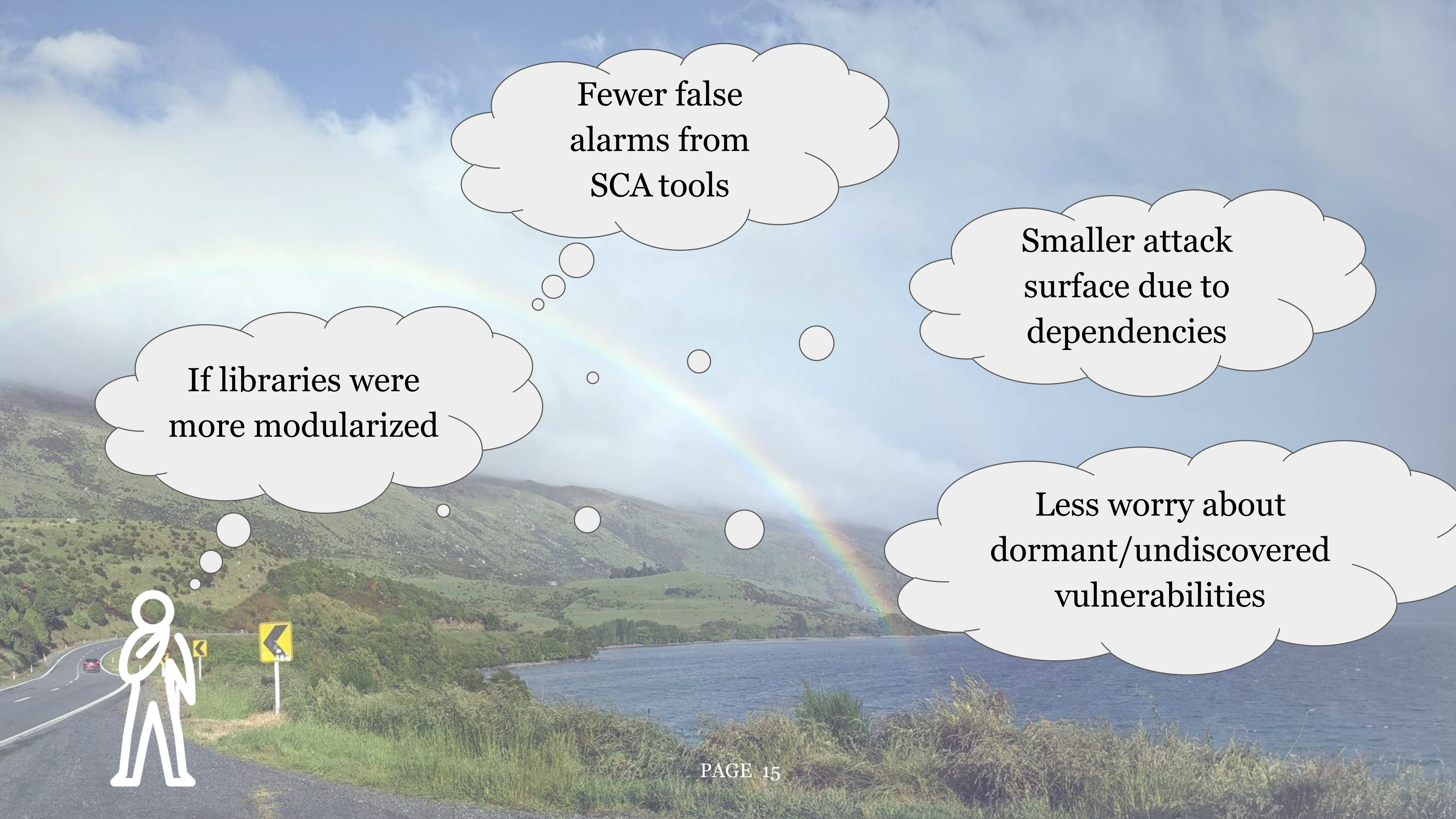
What if clients *could* reference only what they need?



= What if libraries were more modularized?

Actually most clients do not need the whole library!

Only **6%** of clients use functionalities from all modules



If libraries were more modularized

Fewer false alarms from SCA tools

Smaller attack surface due to dependencies

Less worry about dormant/undiscovered vulnerabilities

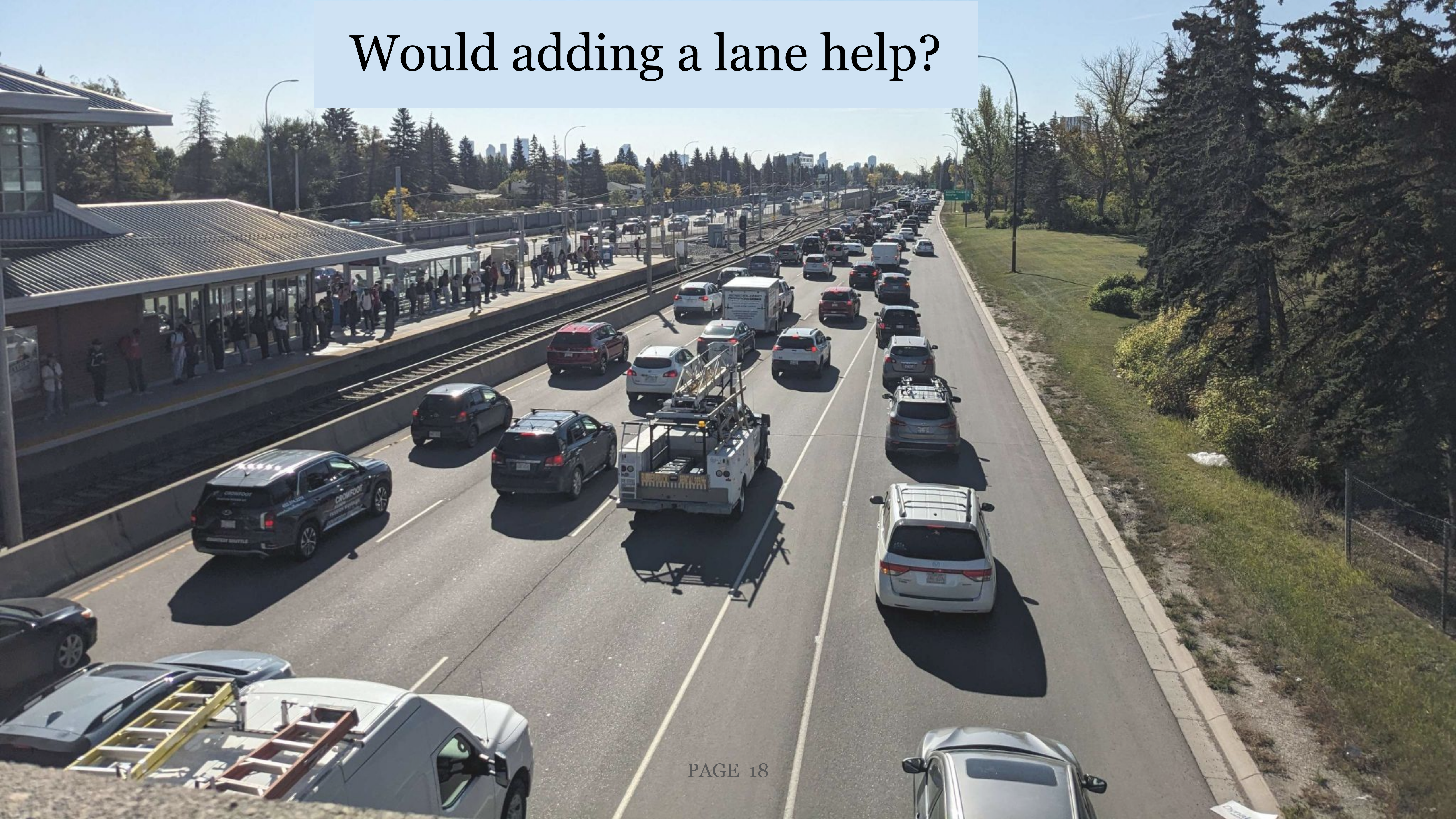
We need study subjects

We need study subjects, but …

Hard to find a large number of libraries *transitioned* from monolith to modular

Hard to control the confounding factors

Would adding a lane help?



We opted for a simulation-based counterfactual analysis

We *simulate* library **modularization**

Measure security metrics **before and after** modularization

Modularization can substantially increase the effectiveness of metadata-based SCA tools

SCA precision reached **71%** after modularization (before: 35%)

94.5% of safe* clients would not receive false security alerts

More than half of the modules (51%) *become safe** after modularization

Modularization can substantially increase the effectiveness of metadata-based SCA tools

SCA precision reached **71%** after modularization (before: 35%)

94.5% of safe* clients would not receive false security alerts

More than half of the modules (51%) *become safe** after modularization

* Refers to transitive constant pool reference to any vulnerability in the class-level dependency graph

Modularization has great potential to isolate the vulnerabilities

SCA precision reached **71%** after modularization (before: 35%)

94.5% of safe* clients would not receive false security alerts

More than half of the modules (51%) *become safe** after modularization

* Refers to transitive constant pool reference to any vulnerability in the class-level dependency graph

Modularization can greatly enhance security of client deployments

78.26% of statically safe clients are **no longer susceptible** to attacks targeting *inactive vulnerabilities*

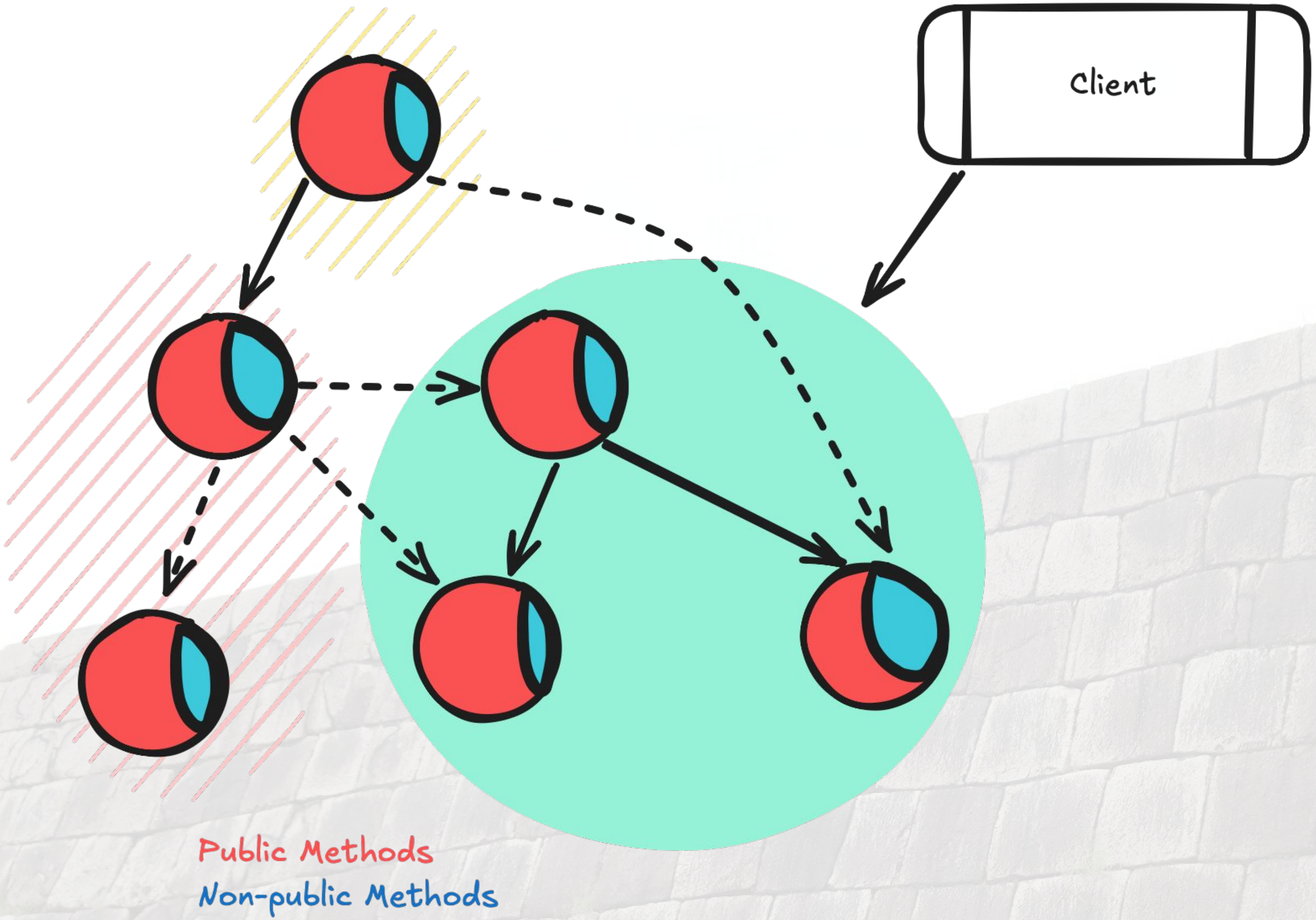
Public attack surface shrinks by **64%** after modularization

Gadget Chains: Attacks Targeting Inactive Vulnerabilities

Modularization can greatly enhance security of client deployments

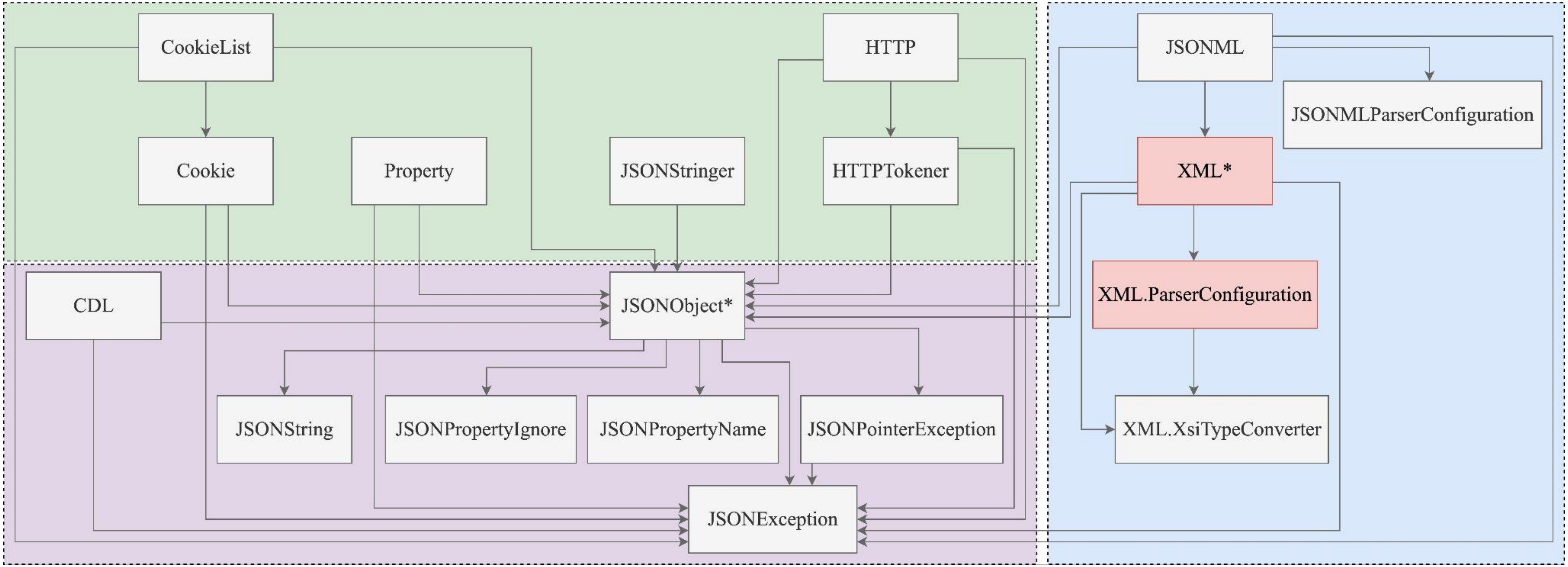
78.26% of statically safe clients are **no longer susceptible** to attacks targeting *inactive vulnerabilities*

Public attack surface shrinks by **64%** after modularization

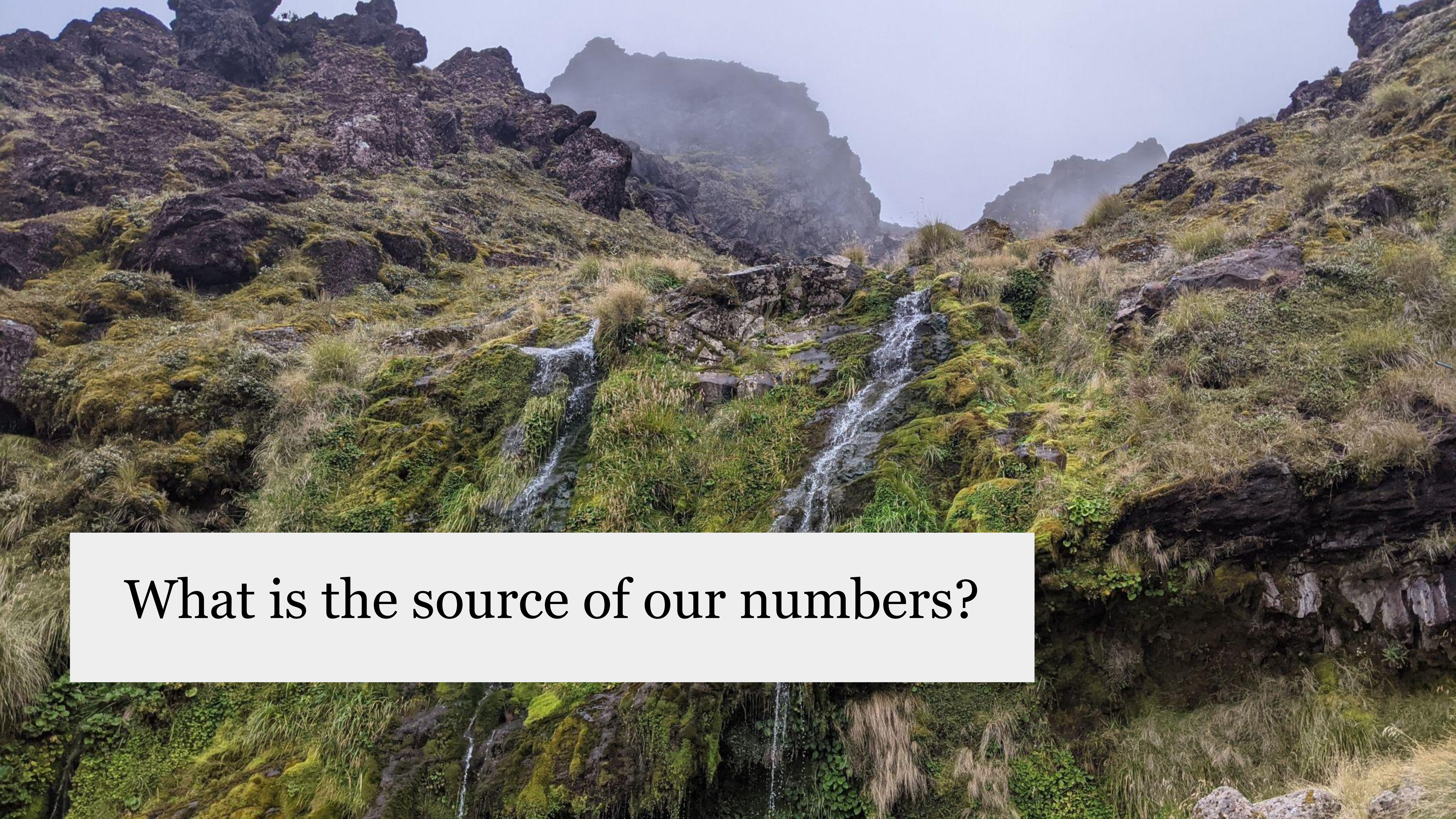


Public API surface shrinks by 64% after modularization

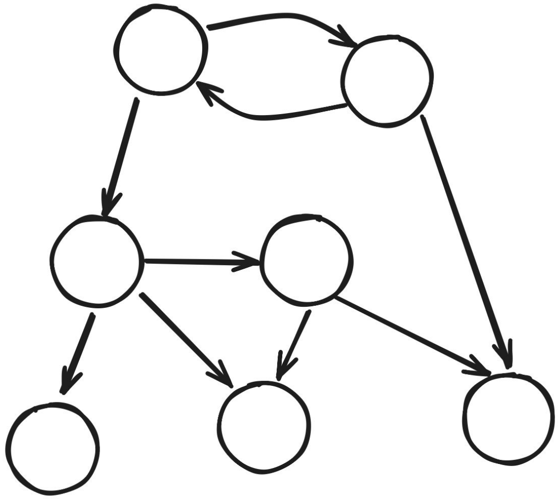
Our modularization can save *org.json*'s clients from the XML vulnerability



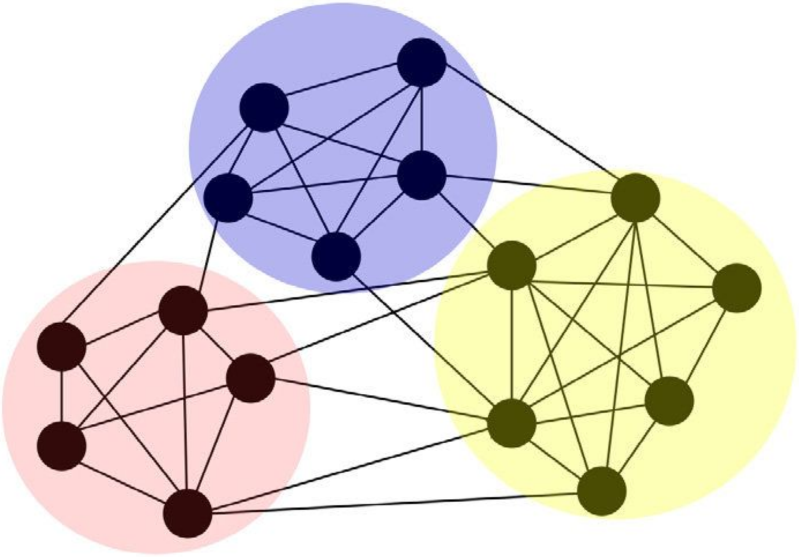
What is the source of our numbers?



What modularization technique do we use?



We need a notion of
dependency graph

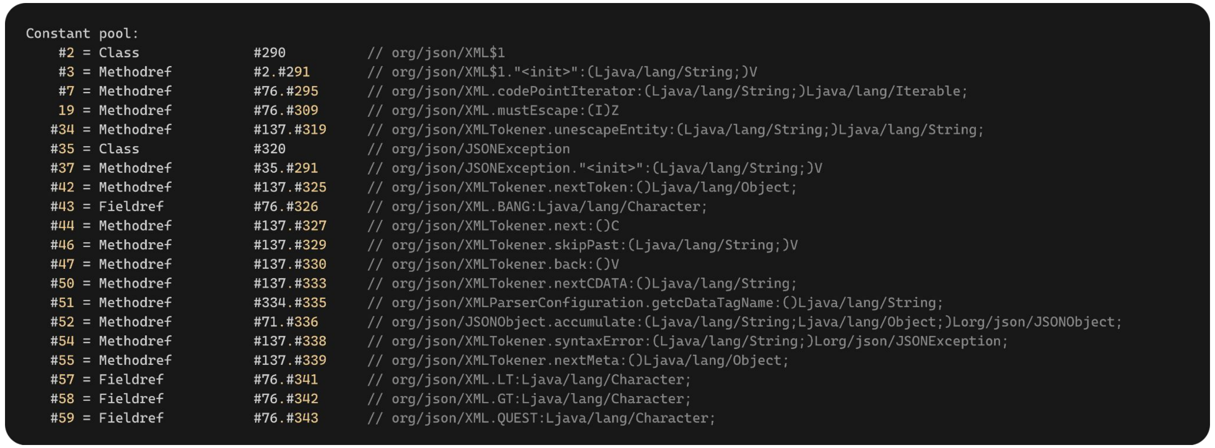


We need a graph
partitioning algorithm

We use ***constant pool*** references to construct *dependency graphs*

```
Constant pool:
    #2 = Class              #290          // org/json/XML$1
    #3 = Methodref          #2.#291       // org/json/XML$1."<init>":(Ljava/lang/String;)V
    #7 = Methodref          #76.#295      // org/json/XML.codePointIterator:(Ljava/lang/String;)Ljava/lang/Iterable;
    19 = Methodref          #76.#309      // org/json/XML.mustEscape:(I)Z
   #34 = Methodref          #137.#319     // org/json/XMLTokener.unescapeEntity:(Ljava/lang/String;)Ljava/lang/String;
   #35 = Class              #320          // org/json/JSONException
   #37 = Methodref          #35.#291      // org/json/JSONException."<init>":(Ljava/lang/String;)V
   #42 = Methodref          #137.#325     // org/json/XMLTokener.nextToken:()Ljava/lang/Object;
   #43 = Fieldref           #76.#326      // org/json/XML.BANG:Ljava/lang/Character;
   #44 = Methodref          #137.#327     // org/json/XMLTokener.next:()C
   #46 = Methodref          #137.#329     // org/json/XMLTokener.skipPast:(Ljava/lang/String;)V
   #47 = Methodref          #137.#330     // org/json/XMLTokener.back:()V
   #50 = Methodref          #137.#333     // org/json/XMLTokener.nextCDATA:()Ljava/lang/String;
   #51 = Methodref          #334.#335     // org/json/XMLParserConfiguration.getcDataTagName:()Ljava/lang/String;
   #52 = Methodref          #71.#336      // org/json/JSONObject.accumulate:(Ljava/lang/String;Ljava/lang/Object;)Lorg/json/JSONObject;
   #54 = Methodref          #137.#338     // org/json/XMLTokener.syntaxError:(Ljava/lang/String;)Lorg/json/JSONException;
   #55 = Methodref          #137.#339     // org/json/XMLTokener.nextMeta:()Ljava/lang/Object;
   #57 = Fieldref           #76.#341      // org/json/XML.LT:Ljava/lang/Character;
   #58 = Fieldref           #76.#342      // org/json/XML.GT:Ljava/lang/Character;
   #59 = Fieldref           #76.#343      // org/json/XML.QUEST:Ljava/lang/Character;
```

We use ***constant pool*** references to construct *dependency graphs*



```
Constant pool:
    #2 = Class              #290          // org/json/XML$1
    #3 = Methodref          #2.#291       // org/json/XML$1."<init>":(Ljava/lang/String;)V
    #7 = Methodref          #76.#295      // org/json/XML.codePointIterator:(Ljava/lang/String;)Ljava/lang/Iterable;
    19 = Methodref          #76.#309      // org/json/XML.mustEscape:(I)Z
   #34 = Methodref          #137.#319     // org/json/XMLTokener.unescapeEntity:(Ljava/lang/String;)Ljava/lang/String;
   #35 = Class              #320          // org/json/JSONException
   #37 = Methodref          #35.#291      // org/json/JSONException."<init>":(Ljava/lang/String;)V
   #42 = Methodref          #137.#325     // org/json/XMLTokener.nextToken:()Ljava/lang/Object;
   #43 = Fieldref           #76.#326      // org/json/XML.BANG:Ljava/lang/Character;
   #44 = Methodref          #137.#327     // org/json/XMLTokener.next:()C
   #46 = Methodref          #137.#329     // org/json/XMLTokener.skipPast:(Ljava/lang/String;)V
   #47 = Methodref          #137.#330     // org/json/XMLTokener.back:()V
   #50 = Methodref          #137.#333     // org/json/XMLTokener.nextCDATA:()Ljava/lang/String;
   #51 = Methodref          #334.#335     // org/json/XMLParserConfiguration.getcDataTagName:()Ljava/lang/String;
   #52 = Methodref          #71.#336      // org/json/JSONObject.accumulate:(Ljava/lang/String;Ljava/lang/Object;)Lorg/json/JSONObject;
   #54 = Methodref          #137.#338     // org/json/XMLTokener.syntaxError:(Ljava/lang/String;)Lorg/json/JSONException;
   #55 = Methodref          #137.#339     // org/json/XMLTokener.nextMeta:()Ljava/lang/Object;
   #57 = Fieldref           #76.#341      // org/json/XML.LT:Ljava/lang/Character;
   #58 = Fieldref           #76.#342      // org/json/XML.GT:Ljava/lang/Character;
   #59 = Fieldref           #76.#343      // org/json/XML.QUEST:Ljava/lang/Character;
```

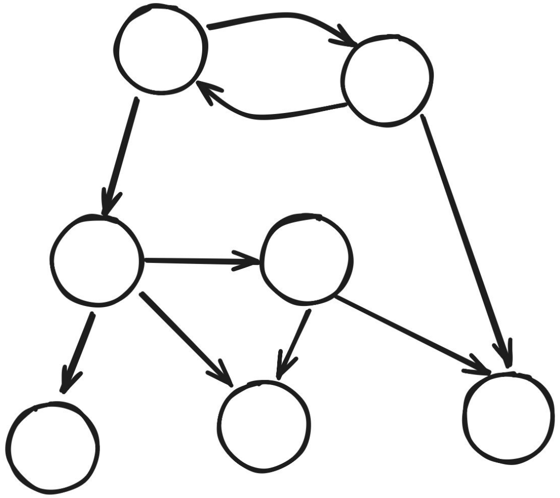
Includes all* sorts of **dependencies**

Super **fast**!

Preserve **compilability**

* Excluding dynamic dependencies

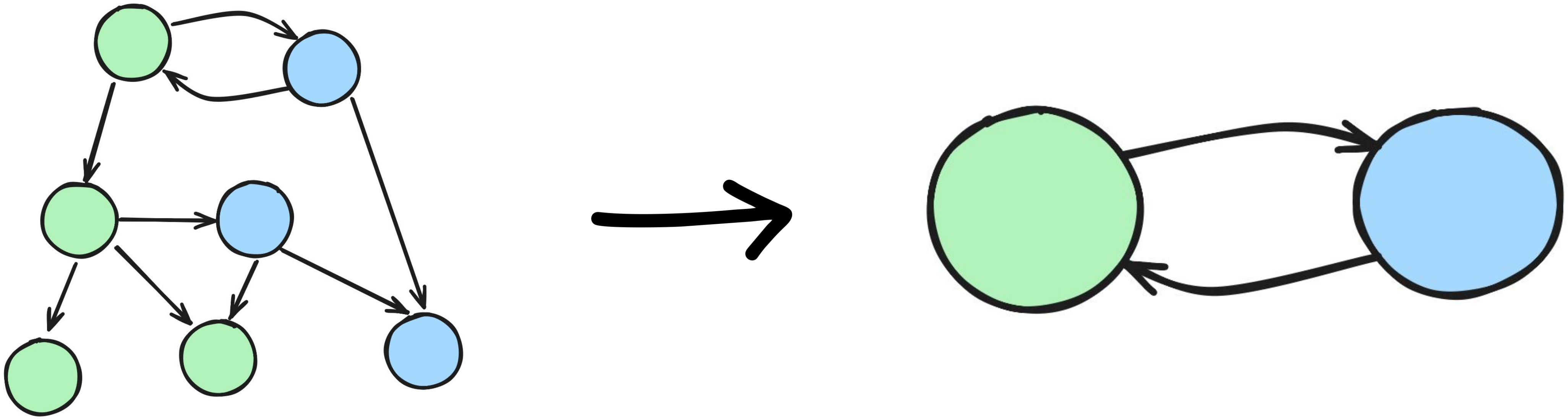
We need a proper modularization technique for reliable results



We use **constant pool** references to construct the ***dependency graphs***

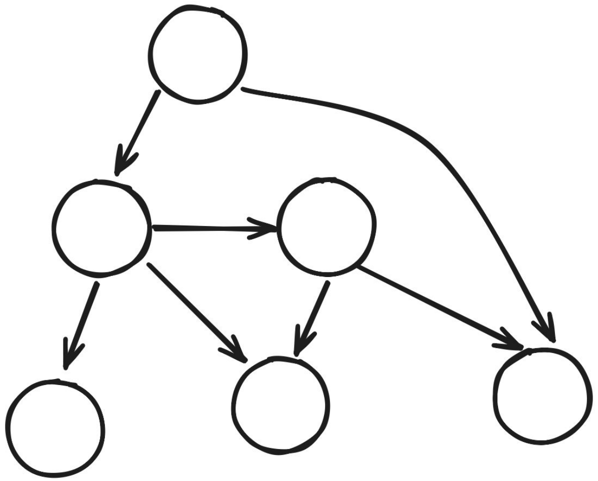
We need a graph ***partitioning algorithm***

The resulting modules should not have dependency cycles

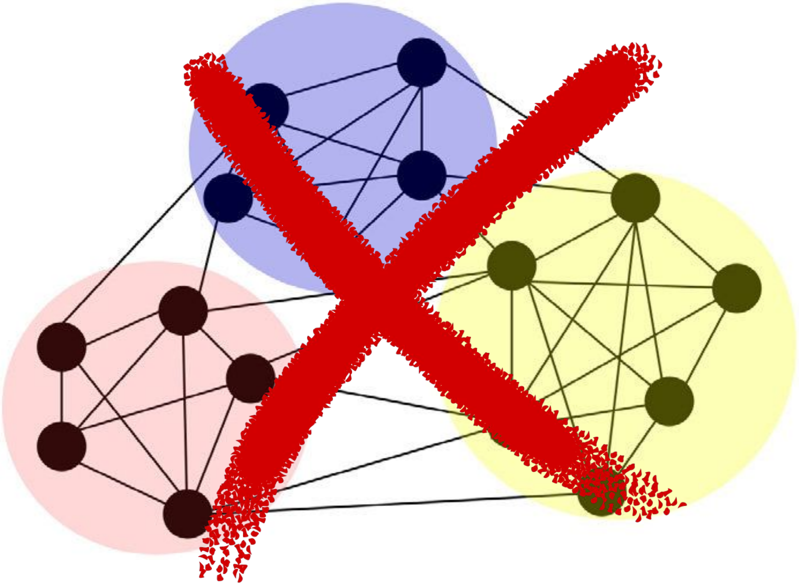


Convert the dependency graph to a DAG, and partition that

We need a proper modularization technique for reliable results

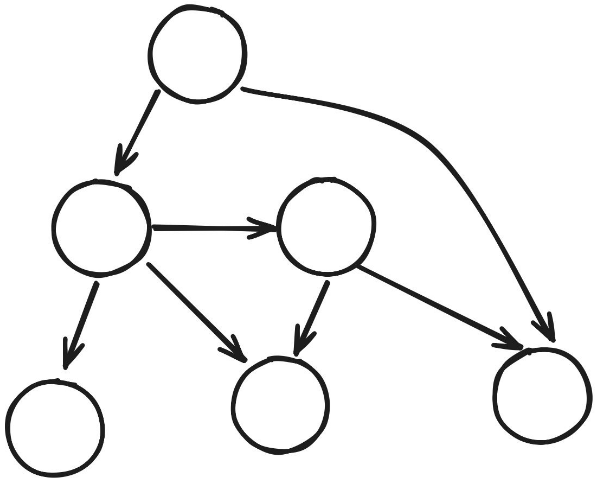


We use *constant pool* references to construct the ***dependency <u>DAGs</u>***

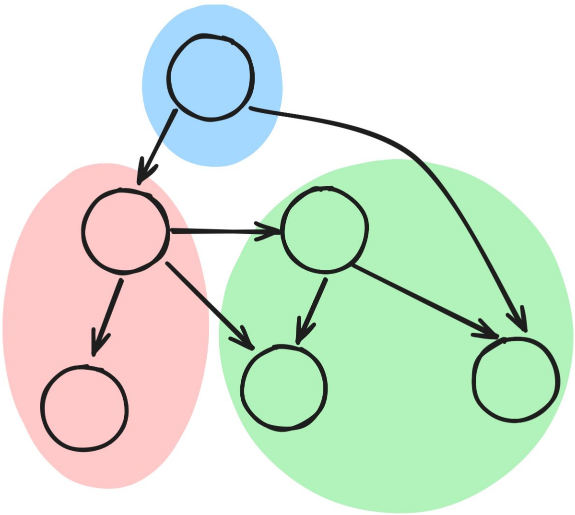


We need a **DAG** *partitioning algorithm*

We need a proper modularization technique for reliable results



We use *constant pool* references to construct the ***dependency DAGs***



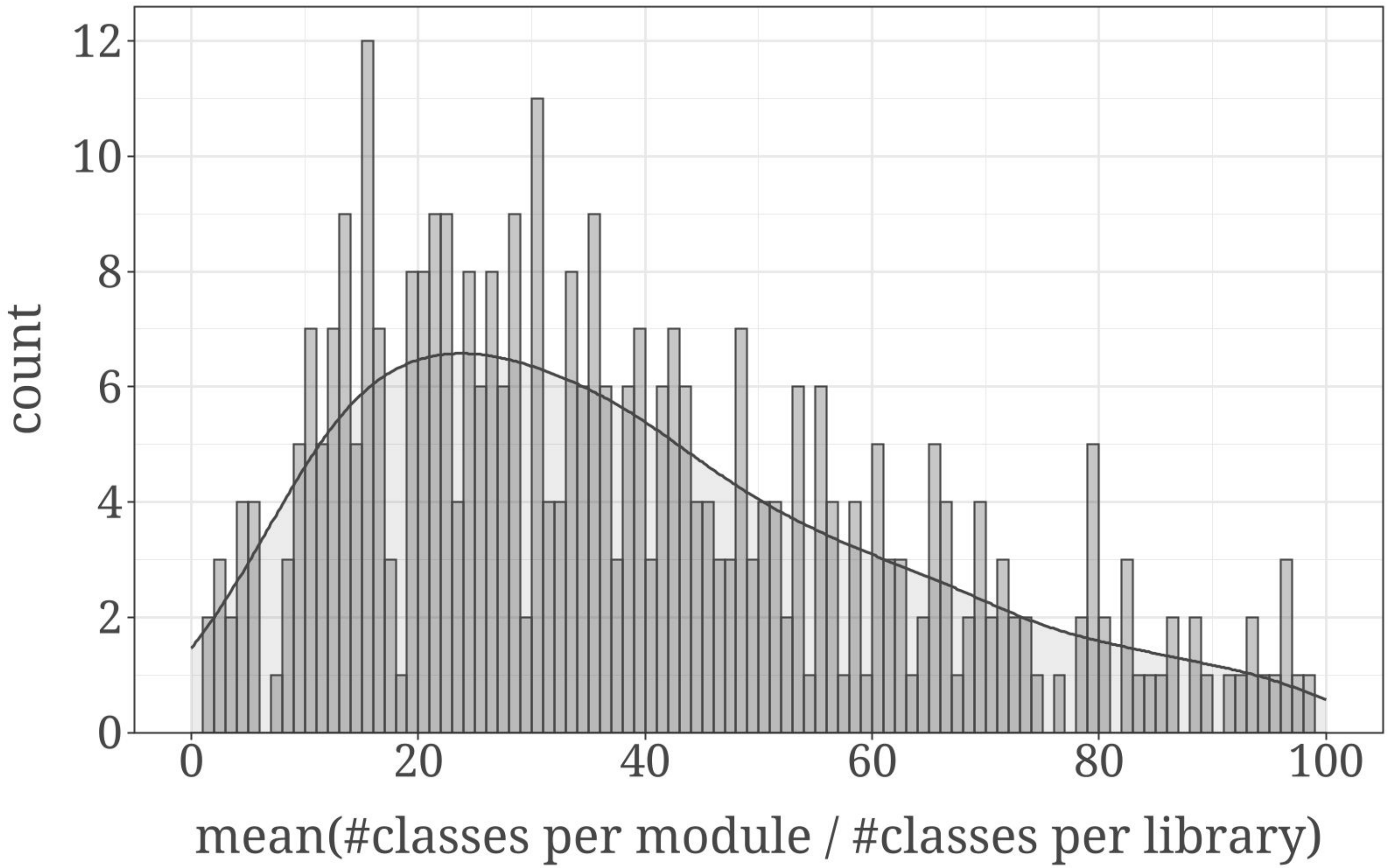
Minimizes the edge cut!

We use **dagP***
to partition dependency graphs *without introducing cycles*

* J. Herrmann, J. Kho, B. Uçar, K. Kaya and Ü. V. Çatalyürek, "Acyclic Partitioning of Large Directed Acyclic Graphs," 2017

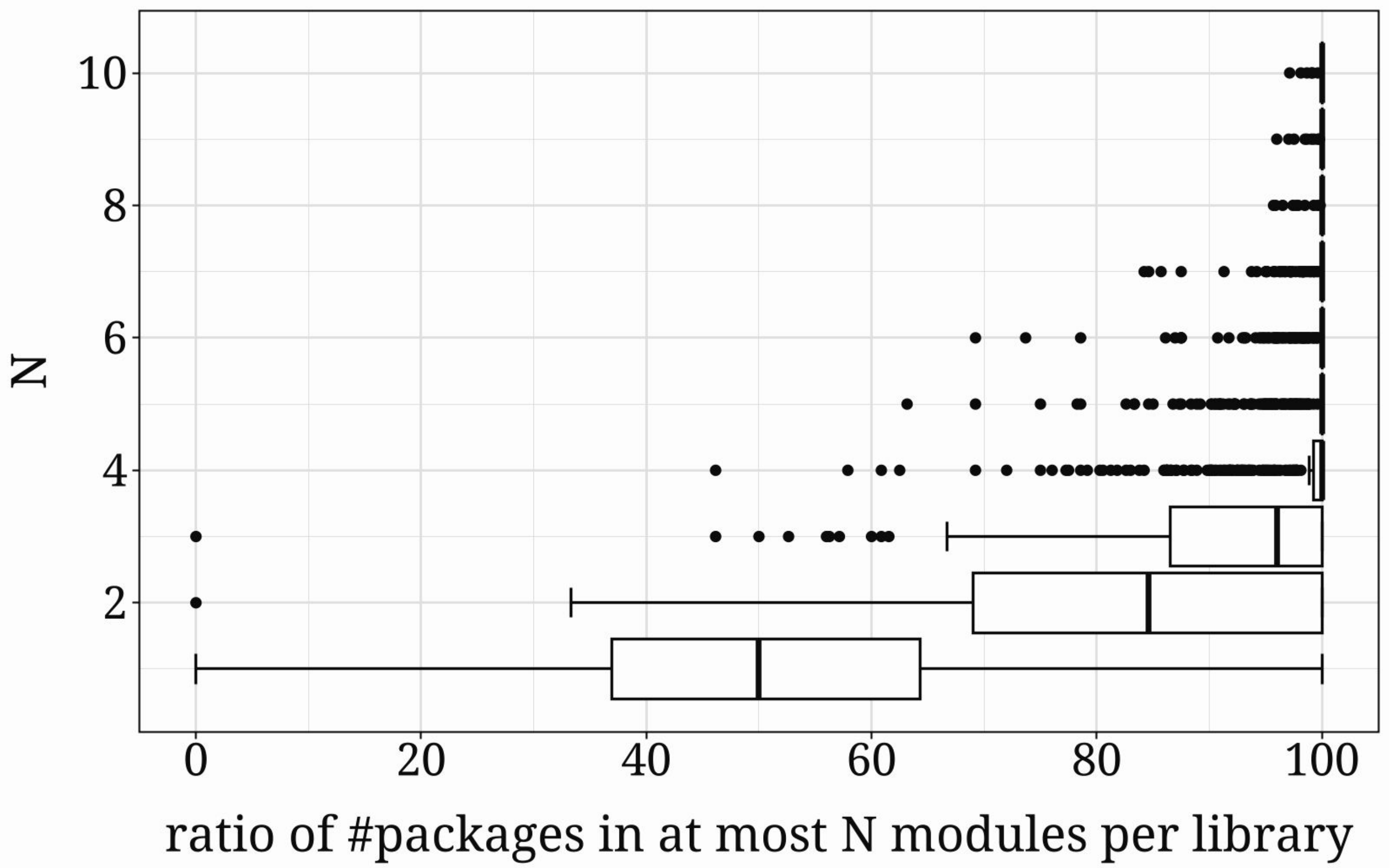
Are the created modules balanced?

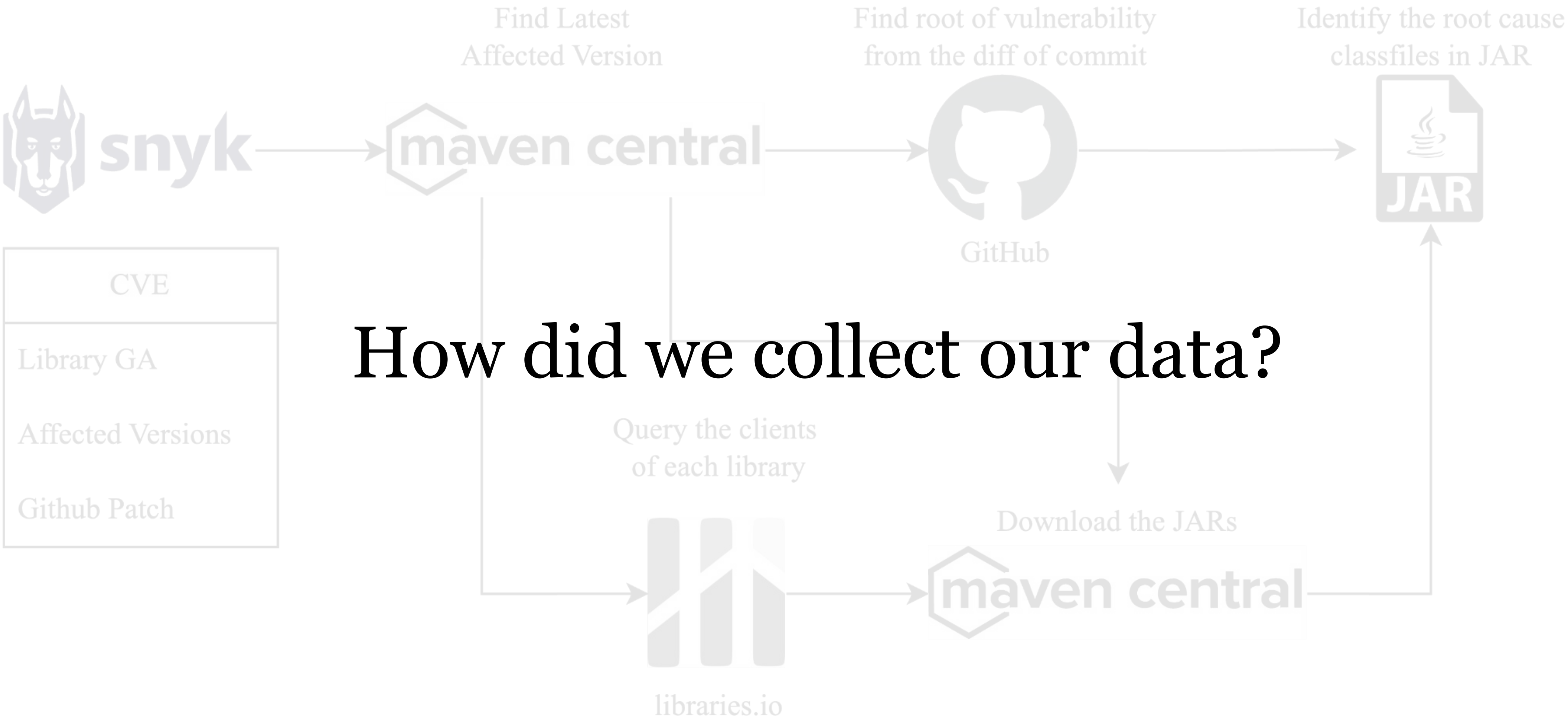
Yes: module sizes resulting from dagP are reasonably balanced



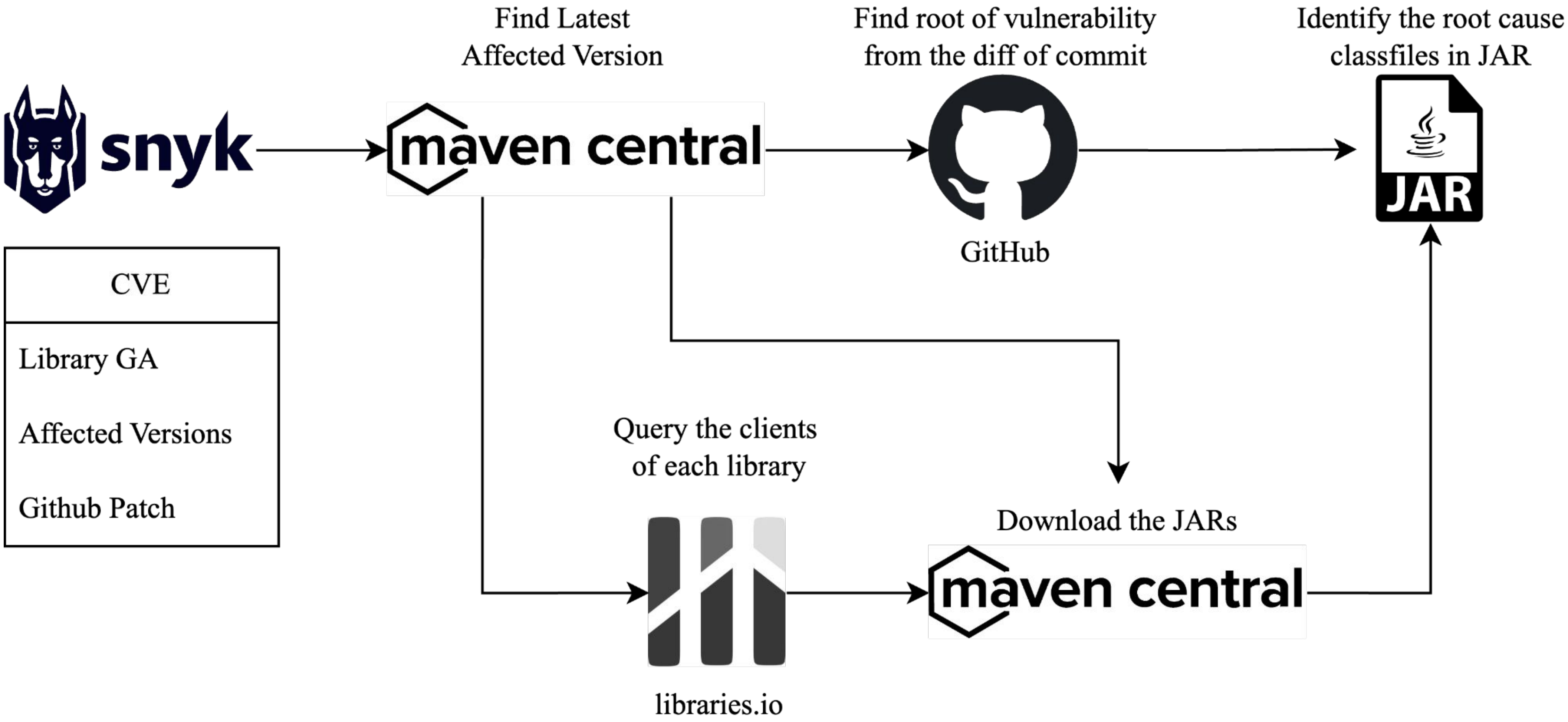
Do the created modules align with existing hierarchies?

Yes: modules are well-aligned with the current library hierarchies.





How did we collect our data?



CVE

Library GA

Affected Versions

Github Patch

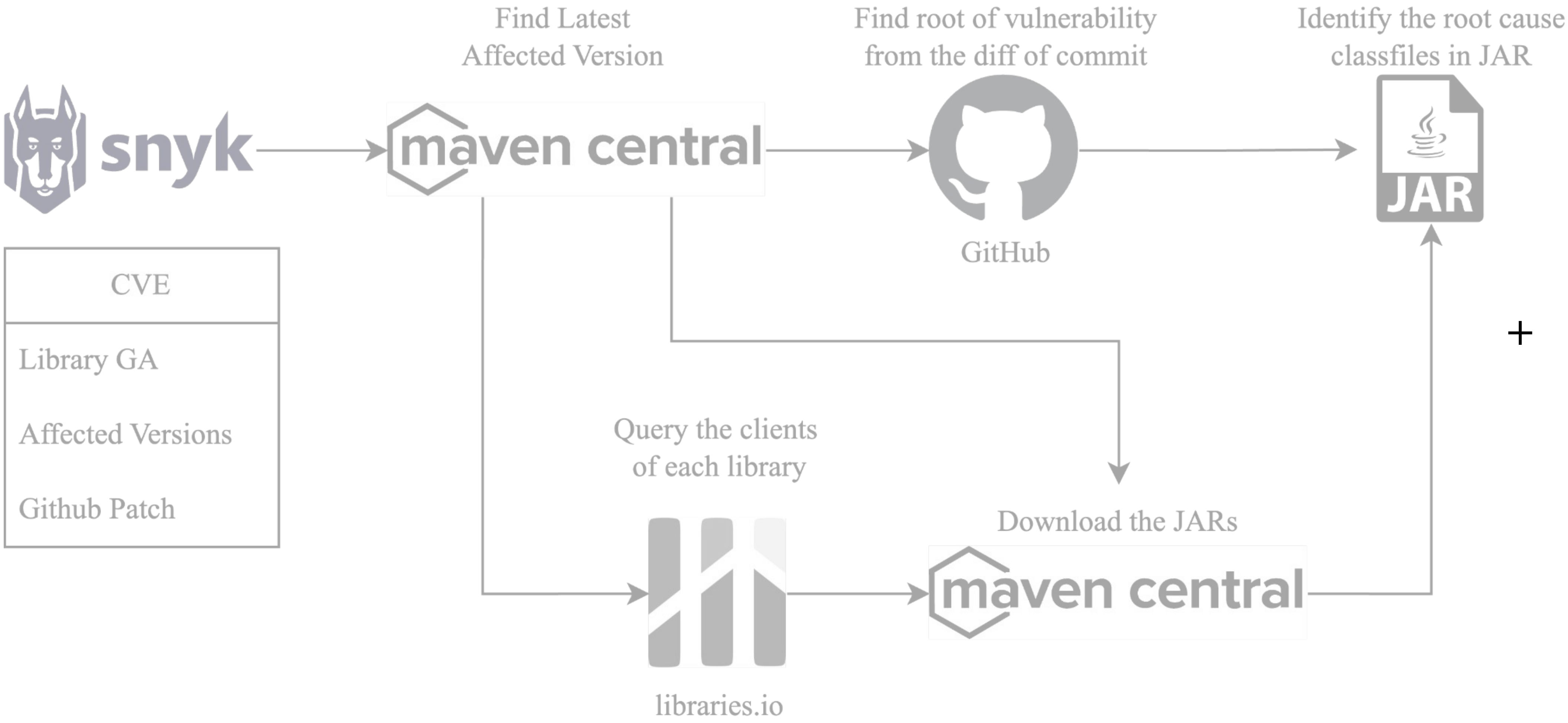
Find Latest Affected Version

Find root of vulnerability from the diff of commit

Identify the root cause classfiles in JAR

Query the clients of each library

Download the JARs



CVE

Library GA

Affected Versions

Github Patch

Find Latest Affected Version

Find root of vulnerability from the diff of commit

Identify the root cause classfiles in JAR

Query the clients of each library

Download the JARs

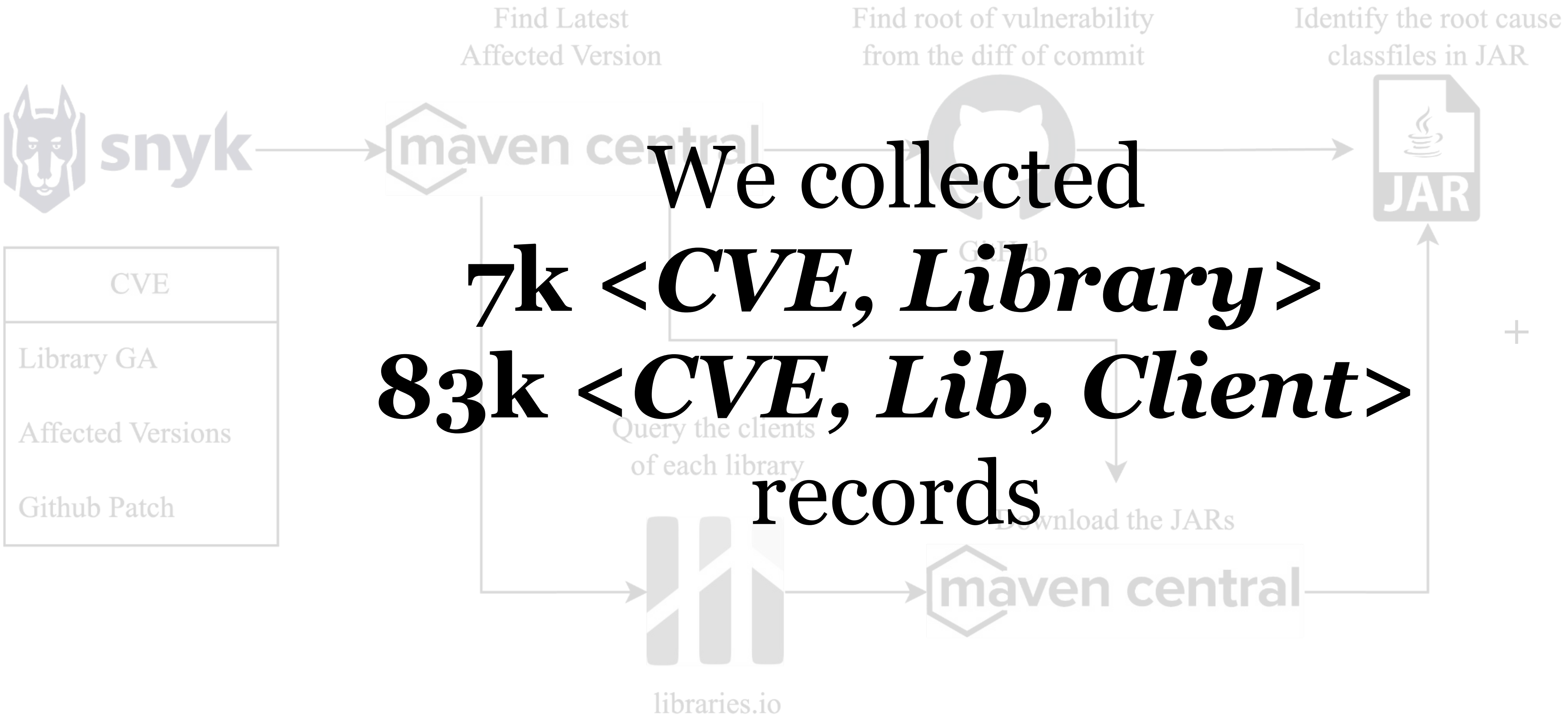
+ data from previous research

We collected

7k <*CVE, Library*>

83k <*CVE, Lib, Client*>

records



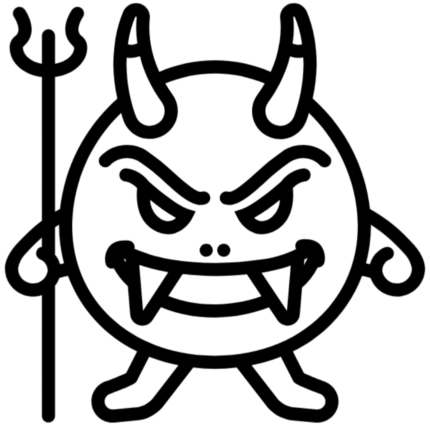
CVE
Library GA
Affected Versions
Github Patch

+ data from previous research

Exploiting vulnerabilities is often more challenging than it initially appears

Only **one class** is *causing* the *vulnerability* (median)

Have to go through **two classes** *to hit a vulnerability* (median)



95% of classes are *public*

Are you a library developer?

Try to release smaller coherent artifacts and let people decide what they need

You can use our modularization approach as a starting point

Do you use large third-party libraries?

You also can use our technique to break large artifacts

Use smaller artifacts (sometimes from the same project!) when possible

Clipart attributions: all CC-BY 3.0 from Noun Project

- Vector Points
- Sam Designs
- Cosmin Petriser
- Solikin
- Amethyst Studio
- Gofficon
- choirun niswah
- ramacae
- tulpan
- Meko
- SeeMoo
- Suharsono
- Imam Kurniadi
- canvas dazzle
- Ifanicon
- Olena Panasovska

Photos from Patrick Lam collection

Hot Takes

1. Many libraries out there are too big.
2. Humans shouldn’t have to do grunt work to modularize libraries.
3. dotnet is better than Java (in terms of clients not including extra libraries).