# Nullable Method Detection

Don’t Repeat The Mistakes Others Have Already Fixed

Manuel Leuenberger

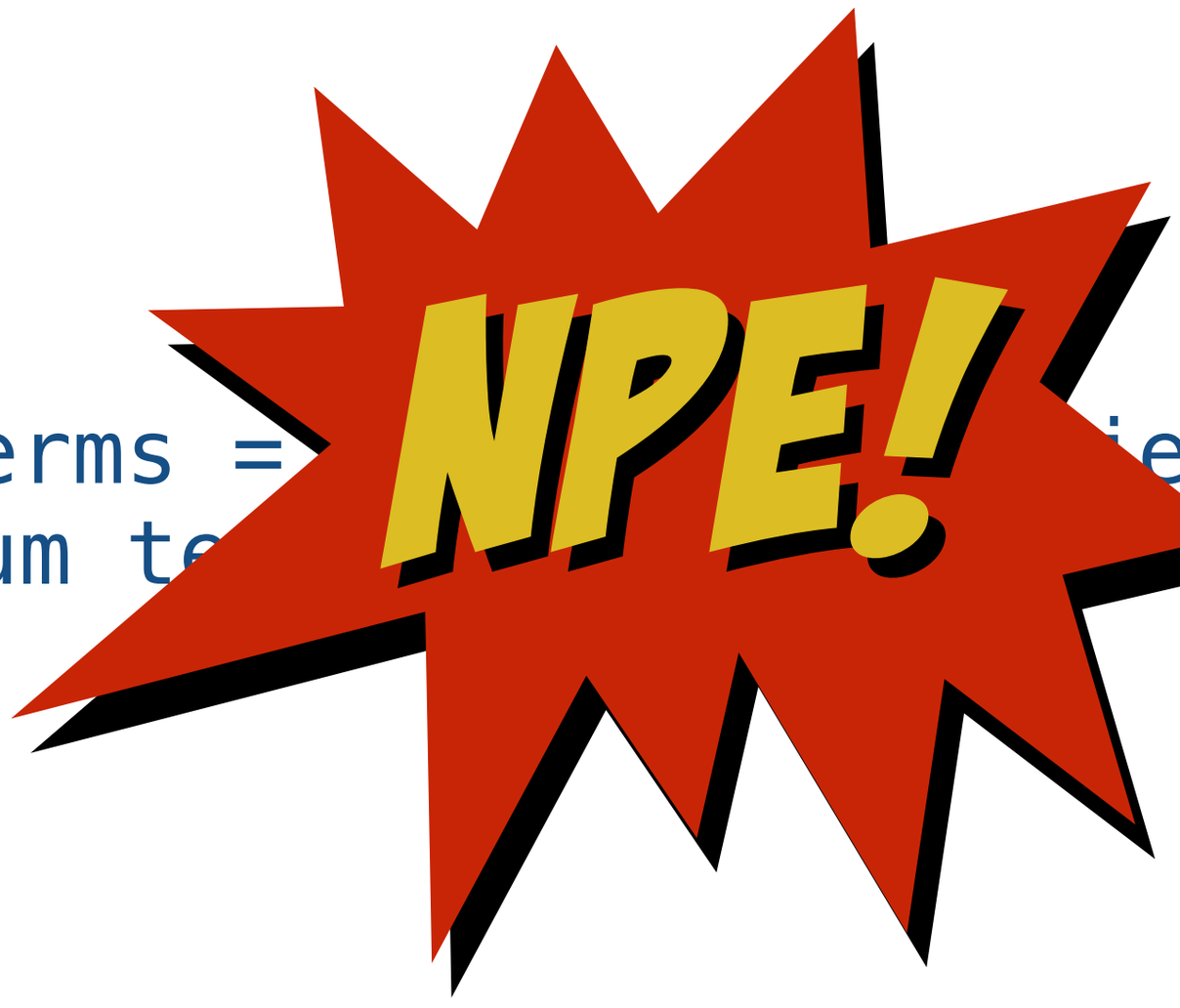
Master thesis

13.12.2016

# Problem

```
Terms terms = fields.terms(field);
TermsEnum termsEnum = terms.iterator();
```

# Problem

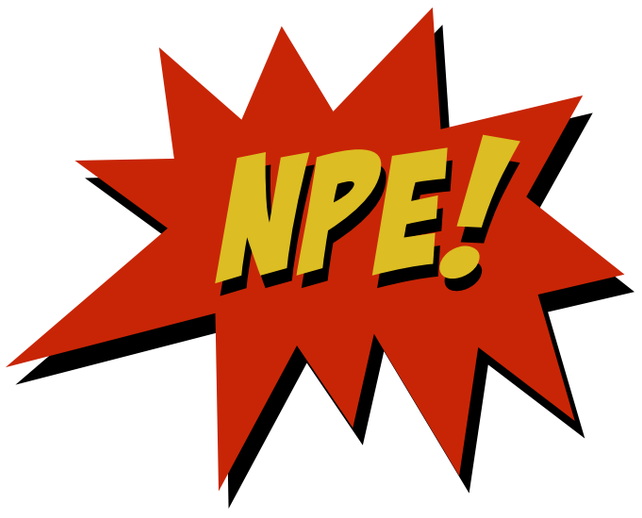



```
Terms terms =                    ield);
TermsEnum te                     erator();
```

# Problem

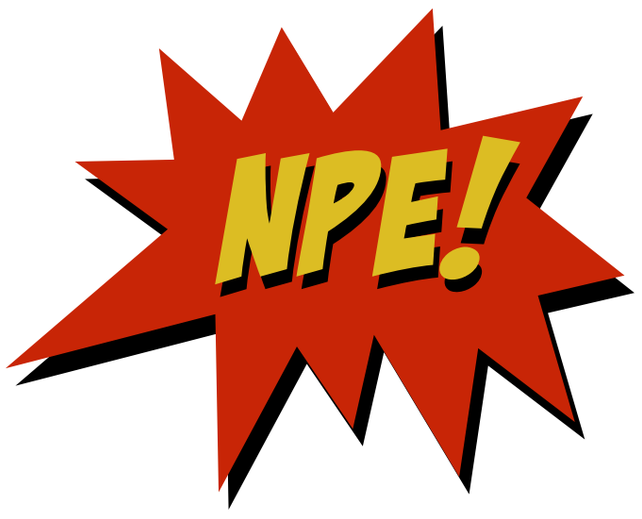
```
Terms terms = fields.terms(field);
TermsEnum termsEnum = terms.iterator();
```

# Related Work



# Related Work

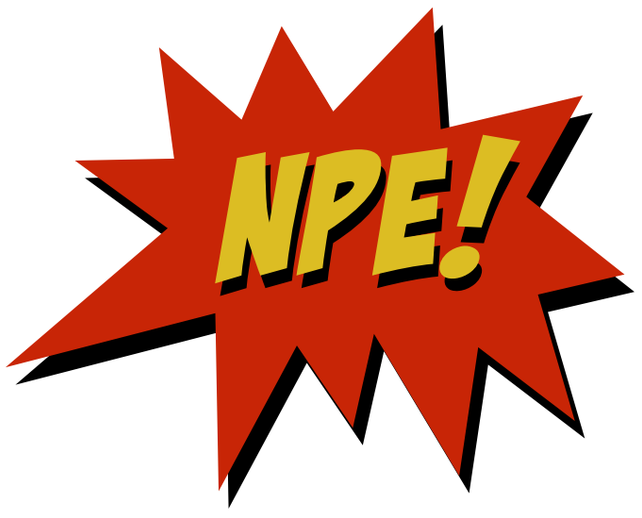
null check is most frequent bug-fix pattern



# Related Work

null check is most frequent bug-fix pattern

checked values are mostly method return values

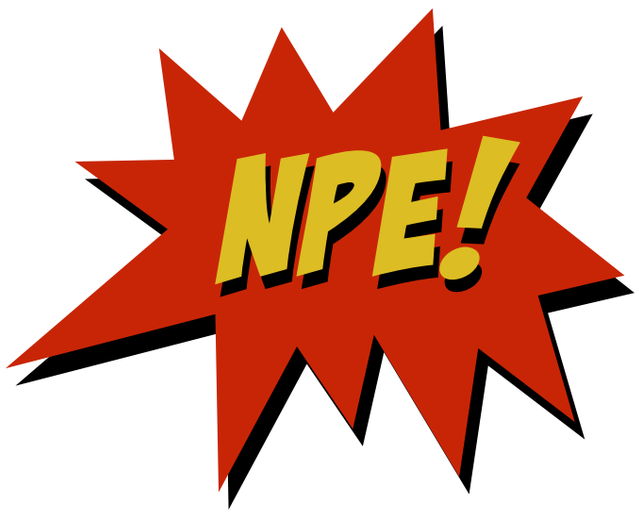


# Related Work

null check is most frequent bug-fix pattern

checked values are mostly method return values

dereference verification by static analysis



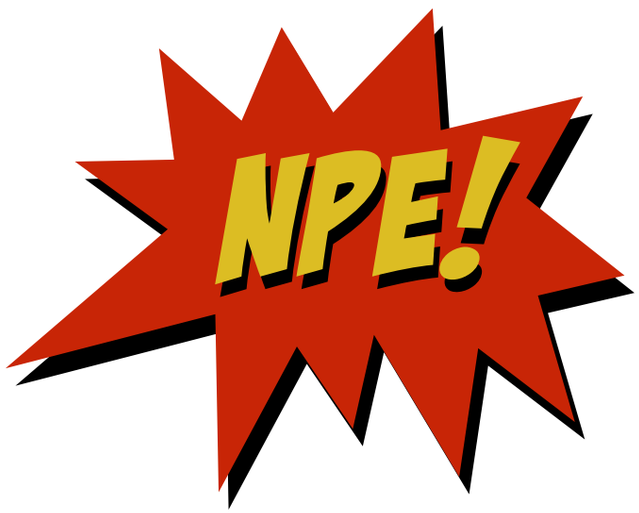
# Related Work

null check is most frequent bug-fix pattern

checked values are mostly method return values

dereference verification by static analysis

false-positives due to infeasible paths



# Idea

**Don’t Repeat The Mistakes Others Have Already Fixed**

empirical evidence instead of verification

# Goal

```
Terms terms = fields.terms(field);
TermsEnum termsEnum = terms.iterator();
```

# Goal

**77% (535/699) checked for null**

```
Terms terms = fields.terms(field);
TermsEnum termsEnum = terms.iterator();
```

# Method UUID

- lookup requires method UUID
- method contracts may evolve
- (class, method) not universally unique

→ (**Maven artifact**, class, method) is unique

```
    groupId: org.apache.lucene
 artifactId: lucene-core
    version: 6.2.0
      class: o.a.l.i.Fields
     method: o.a.l.i.Terms terms(j.l.String)
```

# Nullable Method Detection

*nullable method* := “callers check return value for null”

# Nullable Method Detection

*nullable method* := “callers check return value for null”

```
Terms terms = fields.terms(field);
if (terms == null) {
  // this can happen[…]
  continue;
}
TermsEnum termsEnum = terms.iterator();
```

# Nullable Method Detection

*nullable method* := “callers check return value for null”

```
Terms terms = fields.terms(field);
if (terms == null) {
  // this can happen[…]
  continue;
}
TermsEnum termsEnum = terms.iterator();
```

# Nullable Method Detection

*nullable method* := “callers check return value for null”

```
Terms terms = fields.terms(field);
if (terms == null) {
  // this can happen[…]
  continue;
}
TermsEnum termsEnum = terms.iterator();
```

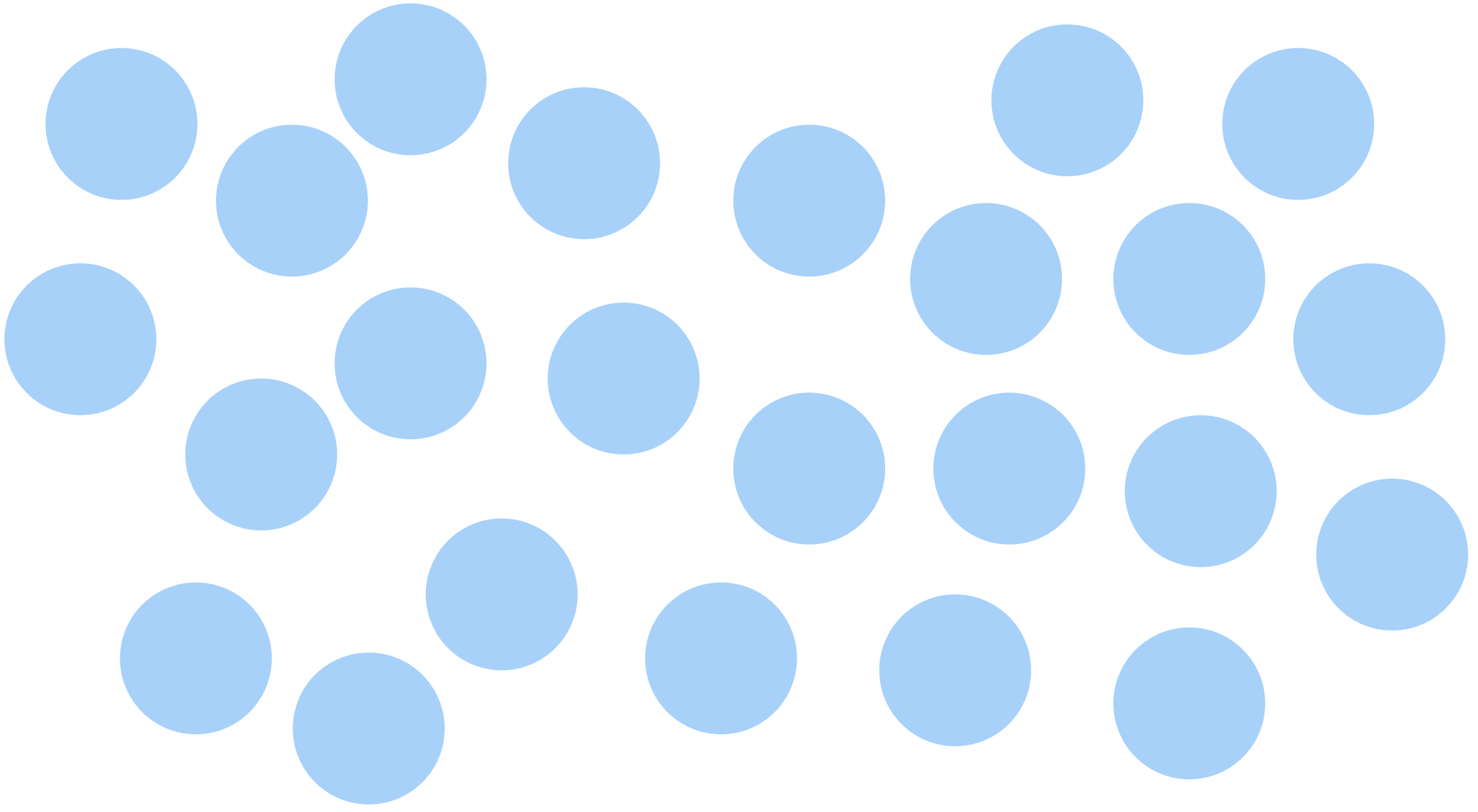
# Nullable Method Detection

*nullable method* := “callers check return value for null”

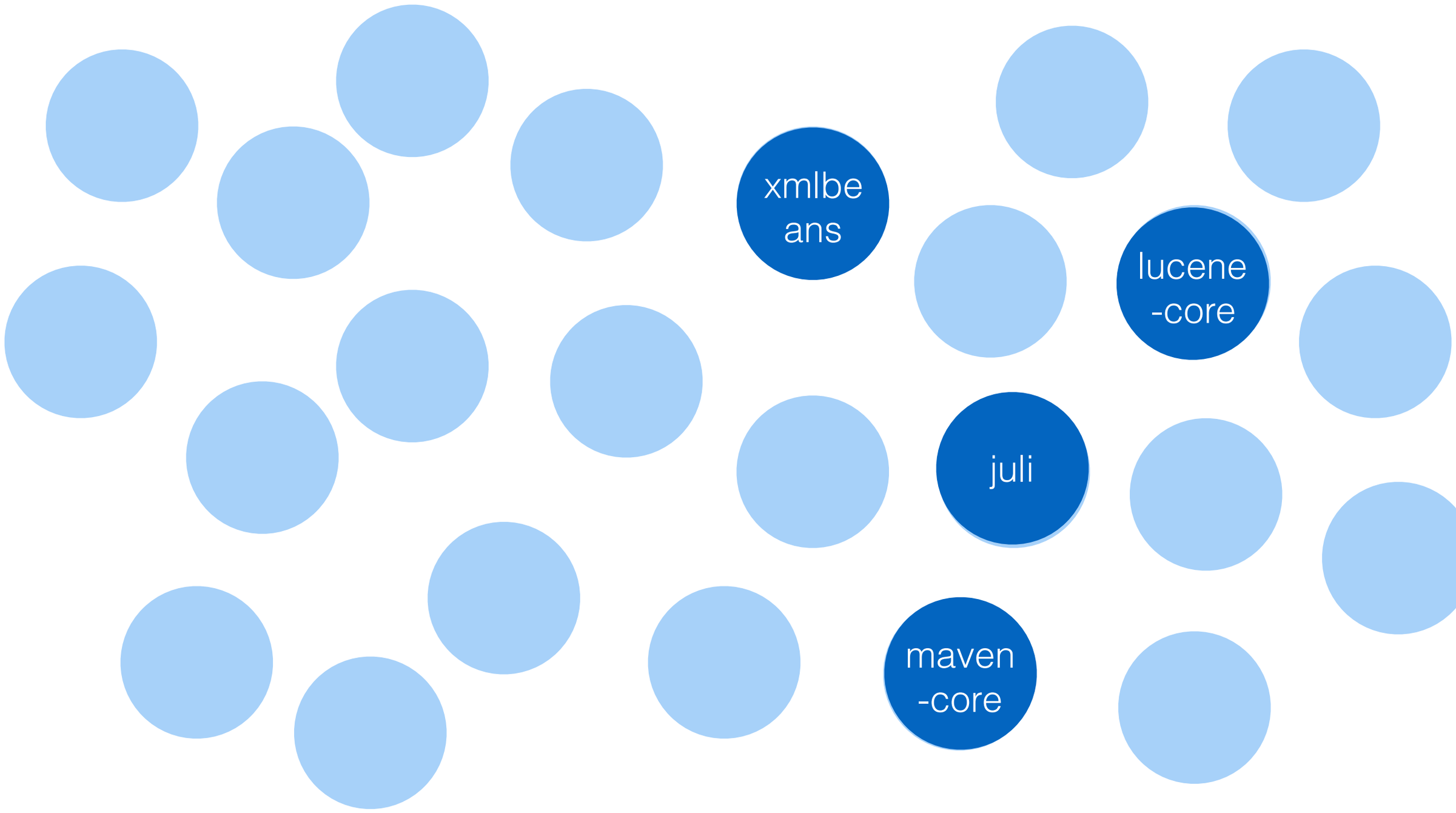
```
org.apache.lucene.index.Fields
org.apache.lucene.index.Terms terms(java.lang.String)
```

```
Terms terms = fields.terms(field);
if (terms == null) {
  // this can happen[…]
  continue;
}
TermsEnum termsEnum = terms.iterator();
```

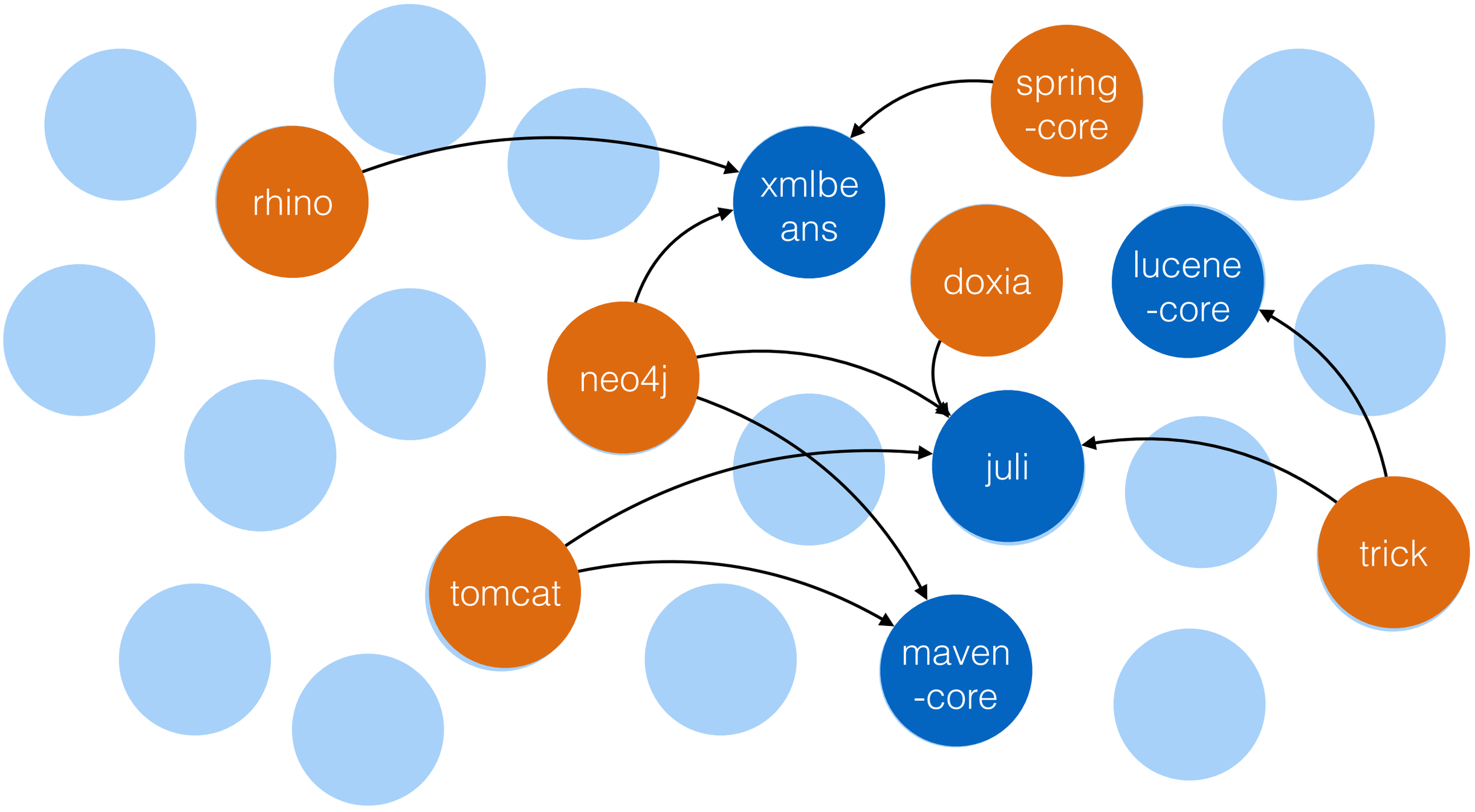
# Usage of **Apache** Artifacts



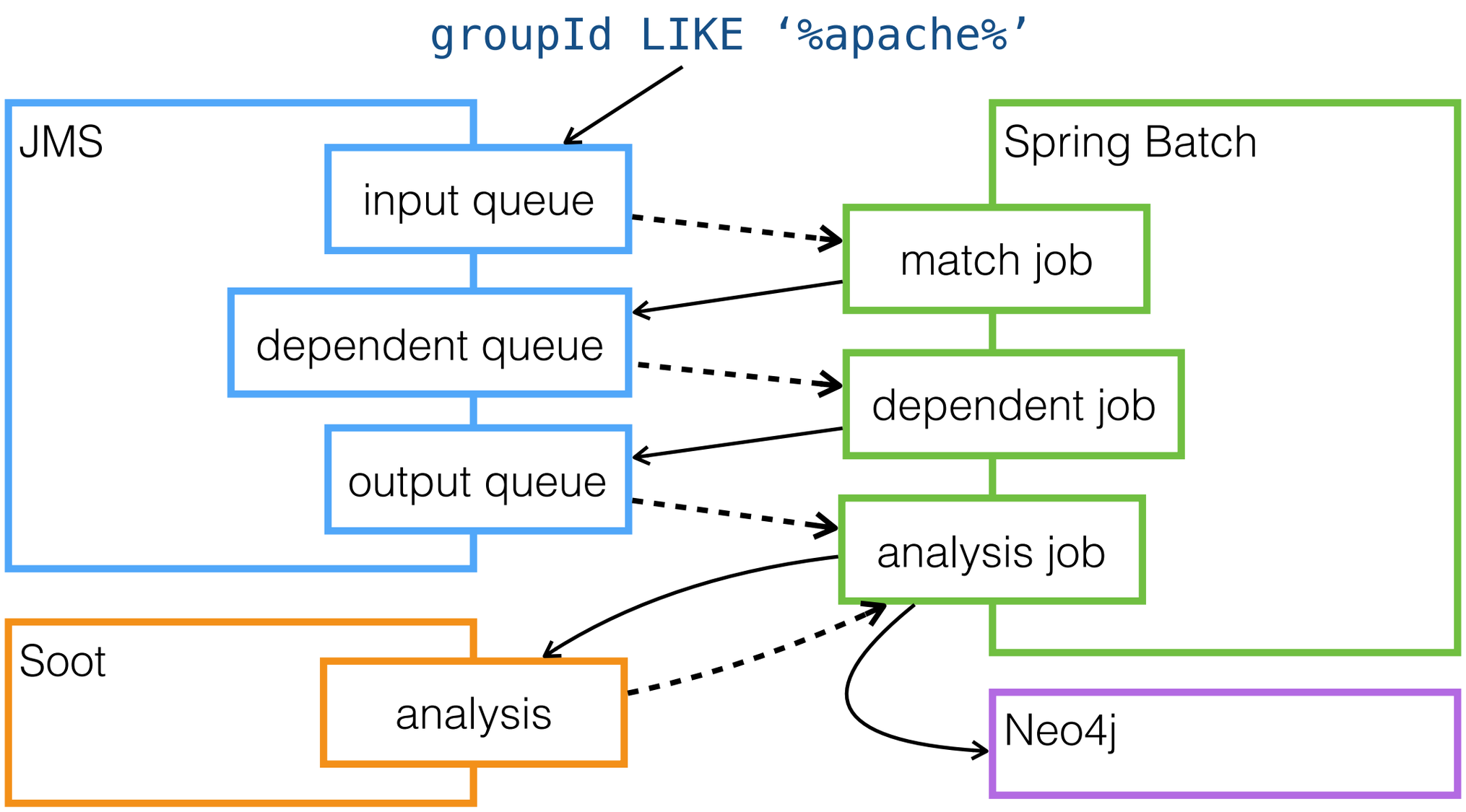
# Usage of **Apache** Artifacts



# Usage of **Apache** Artifacts

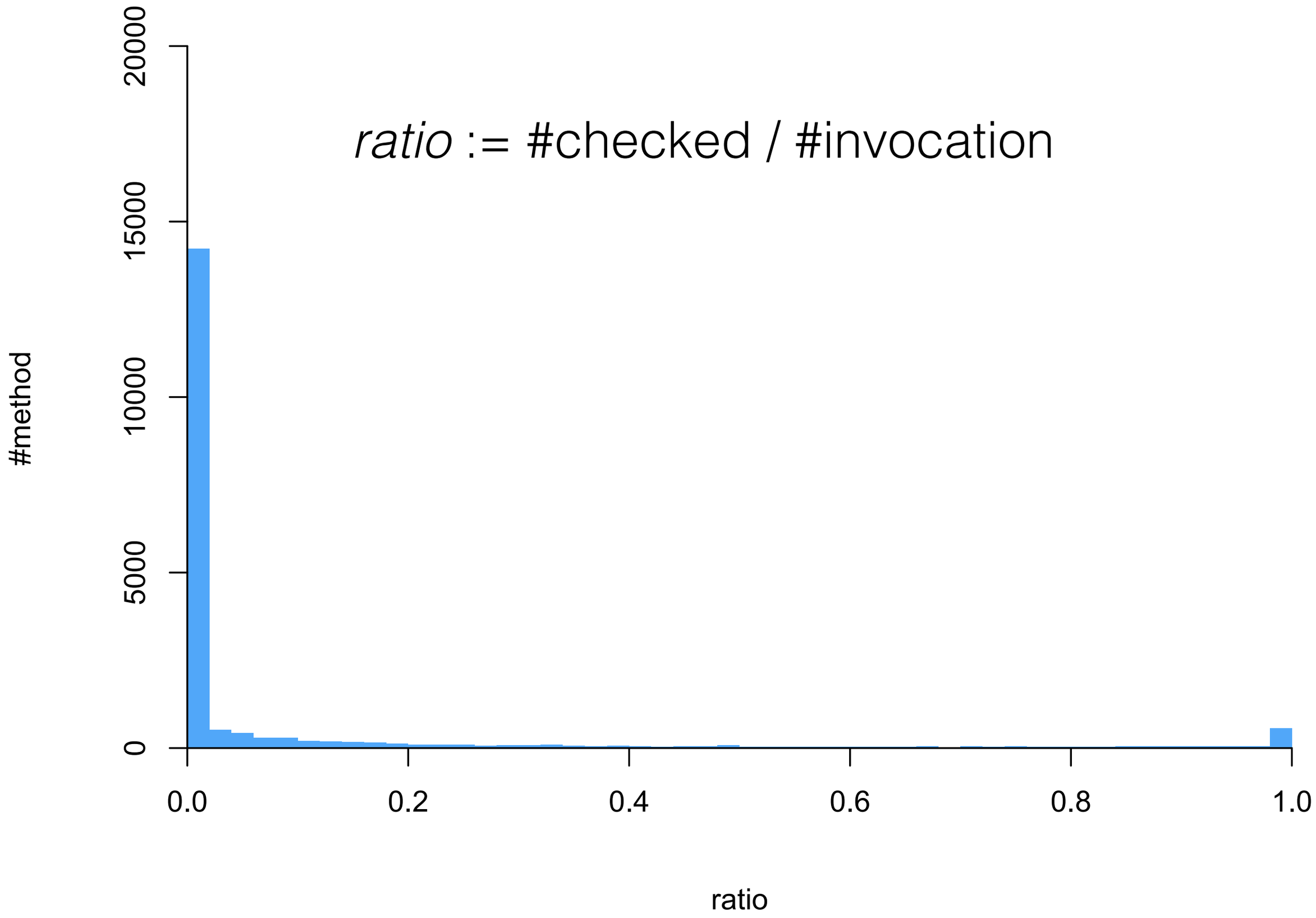


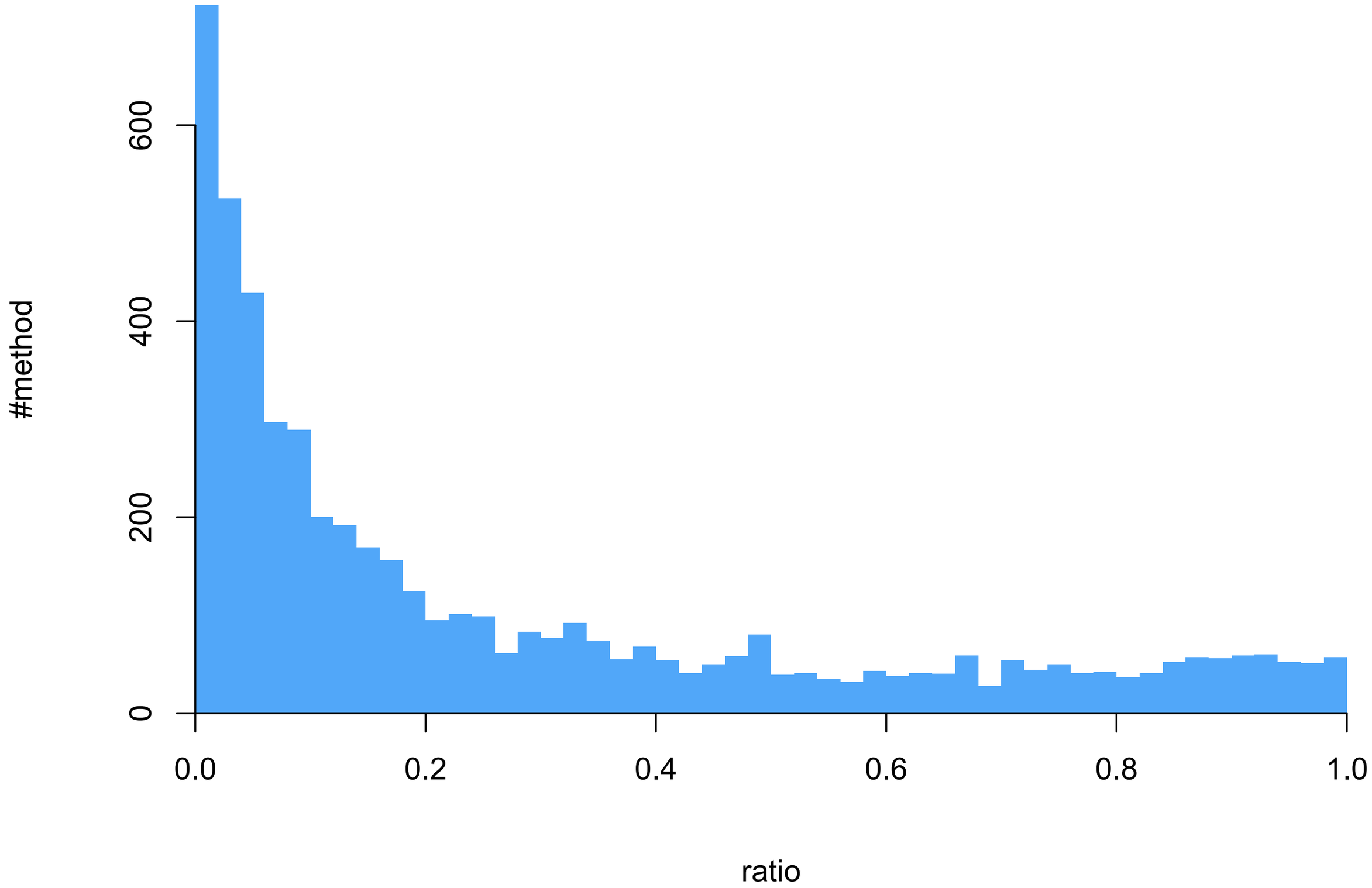
# Architecture



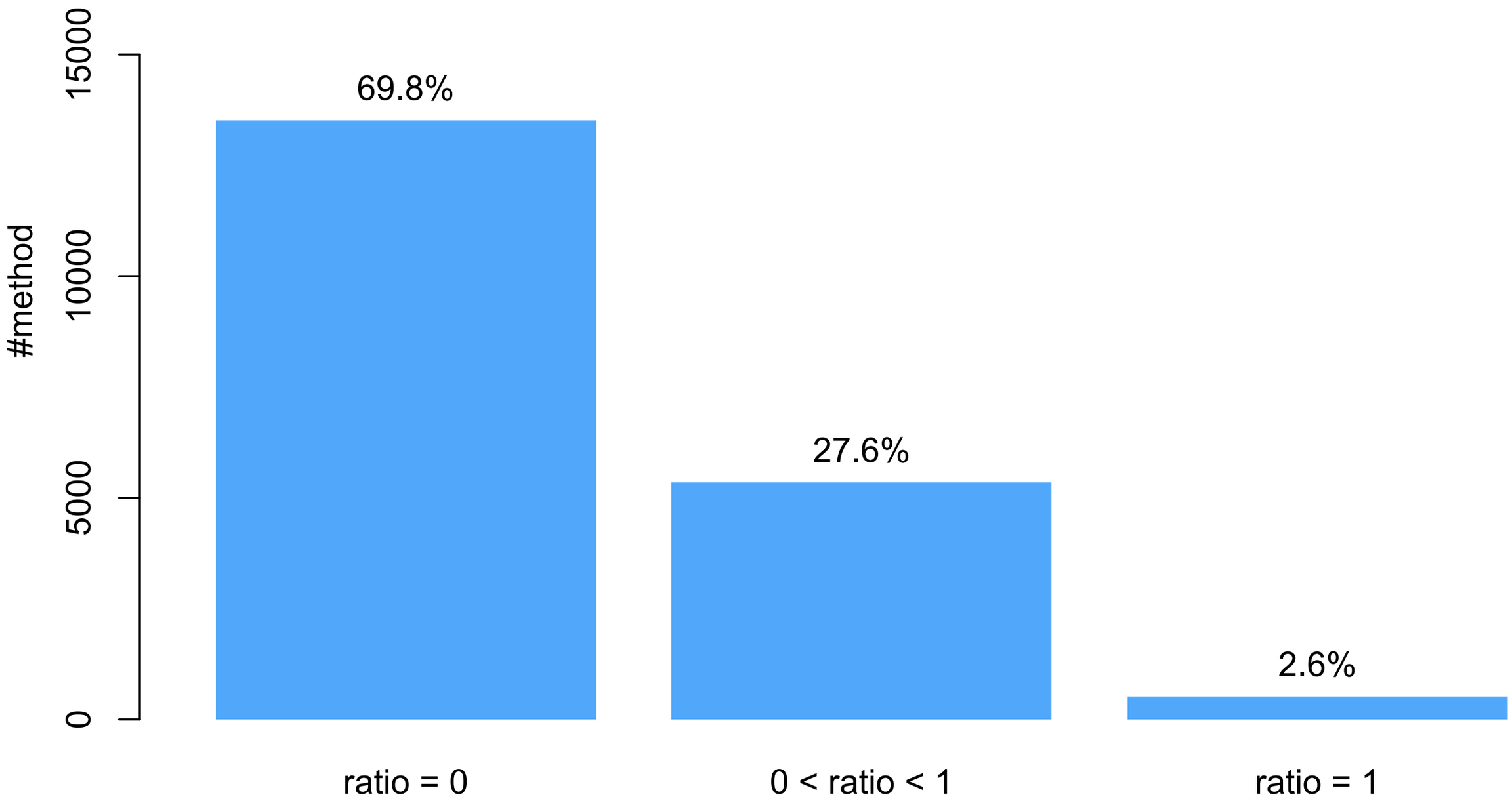
# Results

- analyzed
  - ~45’000 artifacts
  - ~5’000’000 classes
  - ~37’000’000 methods
- found
  - ~150’000’000 invocations
  - ~10’000’000 null checks


nullable method distribution, #method = 19363, #invocation >= 100
ratio := #checked / #invocation
#method
0
5000
10000
15000
20000
0.0
0.2
0.4
0.6
0.8
1.0
ratio


nullable method distribution, #method = 5342, #invocation >= 100, 0 < ratio < 1
#method
0
200
400
600
0.0
0.2
0.4
0.6
0.8
1.0
ratio

# nullable method distribution, #method = 19363, #invocation >= 100



# Limitations

- only ~20’000 methods with >100 invocations
- analysis only detects single expression checked for null in if statements

```
// not detected
Objects.notNull(value)
if (value instanceof  String) {
if (value1 > 0 && value2 != null) {
```

# Conclusion

- Focussed API usage analysis
  - build thousands of class-paths per hour
  - analyze hundreds of artifacts per hour
- 70% of methods are never checked
- 27% of methods are sometimes checked
- 3% of methods are always checked

→ **only invocations of 27% of methods need further analysis**

# Future Work

- IDE plugin that looks up invocations on nullability
- analyze whole Maven repositories
- intra-procedural: detect bugs
- inter-procedural: prune call-graphs with knowledge about nullable methods

# Lessons Learned

- analyzing binaries with Soot is easy, and you get precise types for free
- JMS works well as stream buffers
- Neo4j scales, writing with kernel is nice, querying with cypher is ok, query planner is not always clever, transactions have isolation issues
- Spring is still not a close friend
- still love Java 8 streams
- crunching numbers and plotting with R is surprisingly easy

# References


Haidar Osman, Mircea Lungu, and Oscar Nierstrasz. Mining frequent bug-fix code changes. In Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on, pages 343–347. IEEE, 2014.

Haidar Osman, Manuel Leuenberger, Mircea Lungu, and Oscar Nierstrasz. Tracking null checks in open-source java systems. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), volume 1, pages 304–313. IEEE, 2016.

David Hovemeyer and William Pugh. Finding bugs is easy. ACM Sigplan Notices, 39(12):92–106, 2004.

David Hovemeyer and William Pugh. Finding more null pointer bugs, but not too many. In Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering, pages 9–14. ACM, 2007.

David Hovemeyer, Jaime Spacco, and William Pugh. Evaluating and tuning a static analysis to find null pointer bugs. In Proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, PASTE '05, pages 13–19, New York, NY, USA, 2005. ACM.

Mangala Gowri Nanda and Saurabh Sinha. Accurate interprocedural null-dereference analysis for java. In Proceedings of the 31st International Conference on Software Engineering, pages 133–143. IEEE Computer Society, 2009.

Alexey Loginov, Eran Yahav, Satish Chandra, Stephen Fink, Noam Rinetzky, and Mangala Nanda. Verifying dereference safety via expanding-scope analysis. In Proceedings of the 2008 international symposium on Software testing and analysis, pages 213–224. ACM, 2008.

Ravichandhran Madhavan and Raghavan Komondoor. Null dereference verification via over- approximated weakest pre-conditions analysis. ACM Sigplan Notices, 46(10):1033–1052, 2011.

Raja Valle´e-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot-a java bytecode optimization framework. In Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research, page 13. IBM Press, 1999.