Università
della
Svizzera
italiana

**Faculty
of Informatics**

AutoBench

Finding Workloads That You Need Using Pluggable Hybrid Analysis

Andrea Rosà
PhD candidate
Dynamic Analysis Group
Università della Svizzera Italiana (USI)
Lugano, Switzerland

University of Kyoto
March 22nd, 2016

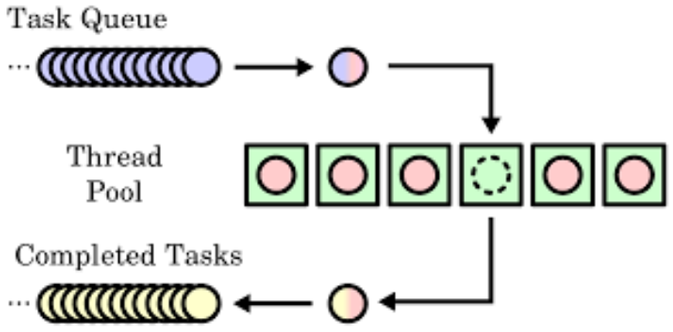
What is it?

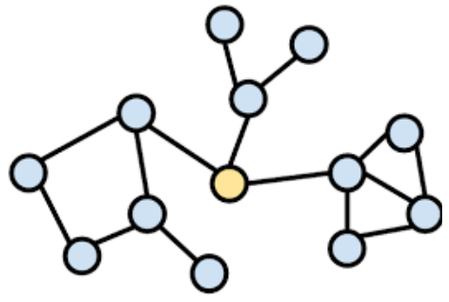
- A methodology and toolchain to **find workloads** and synthesize **benchmark suites** from open-source projects

- Definitions:
  - **Workload**: the execution profile of a running application
  - **Benchmark**: workload considered as reference for a given domain

What can be used for?

- **Find** workloads utilizing specific libraries/features





```
i = j + k;                    1      ILOAD j    // i = j + k      Byte Offset
if (i == 3)                   2      ILOAD k                    0    0x15 0x02
    k = 0;                    3      IADD                       2    0x15 0x03
else                          4      ISTORE i                   4    0x60
    j = j – 1;                5      ILOAD i    // if (i < 3)     5    0x36 0x01
                              6      BIPUSH 3                   7    0x15 0x01
                              7      IF_ICMPEQ L1               9    0x10 0x03
                              8      ILOAD j    // j = j – 1      11   0x9F 0x00 0x0D
                              9      BIPUSH 1                   14   0x15 0x02
                             10      ISUB                       16   0x10 0x01
                             11      ISTORE j                   18   0x64
                             12      GOTO L2                    19   0x36 0x02
                             13  L1: BIPUSH 0    // k = 0       21   0xA7 0x00 0x07
                             14      ISTORE k                   24   0x10 0x00
                             15  L2:                            26   0x36 0x03
                                                                28
```

- **Synthesize** benchmarks utilizing specific libraries/ features

- **Apply** analyses on workloads

- **Validate** optimizations and tools

What cannot be used for?

- Compare performance of similar applications
- Find workloads functionally equivalent to a given one
  - Find same algorithm/application written in different programming languages

When should we use it?

- When benchmarks for **specific needs** are needed
- Typical benchmark suite:

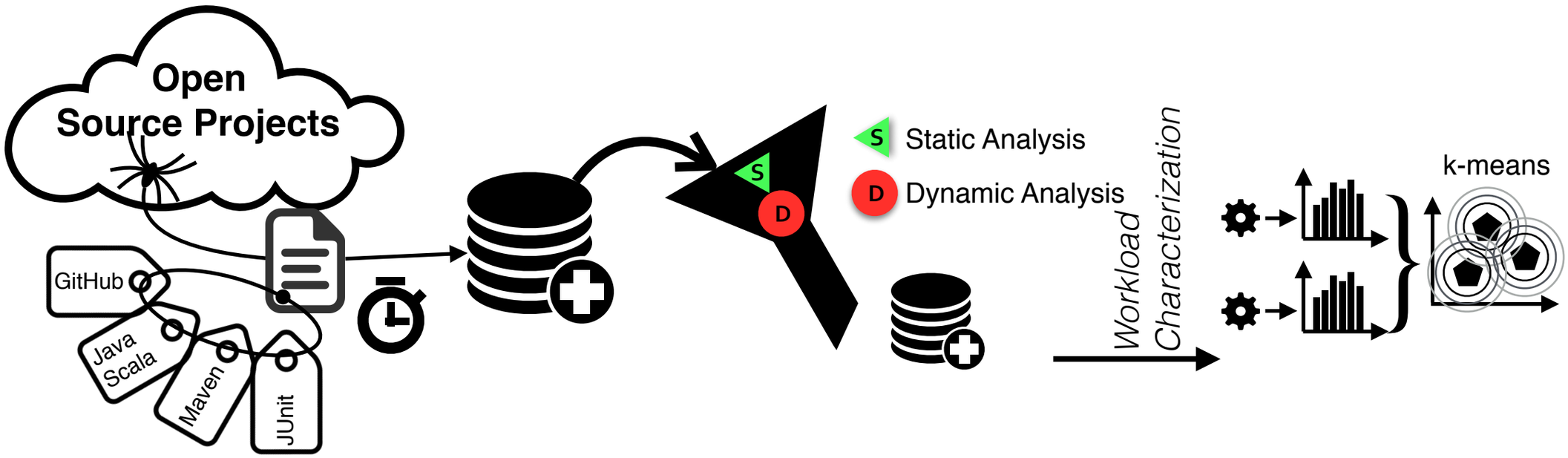
- **Representative**
- **Reproducible results**
- **Diverse**
- but… **general-purpose**
  - AutoBench synthesizes **specific** benchmarks

AutoBench

- Ease the process of constructing benchmark suites for specific needs
- Benchmarks synthesized with AutoBench:
  - **Representative** → Benchmarks taken from **real-world** open-source projects
  - **Reproducible results** → Benchmarks taken from **unit tests**
    - **Automatic** process
  - **Diverse** → Workload **characterization** to choose diverse benchmarks
  - **Specific** → Workload **identification** via static and dynamic analysis

AutoBench

A methodology



A toolchain

```
classToFilter =  Array("java.util.concurrent.Execut
                       "java.util.concurrent.E
                       "java.util.concurrent.A
                       "java.util.concurrent.T
                       "java.util.concurrent.F
mode = FilterMode.Imports
```

Instrumentation.java

```
@Before(mark
static void
     Profil
}

@After(marke
static void
     Profil
}
```

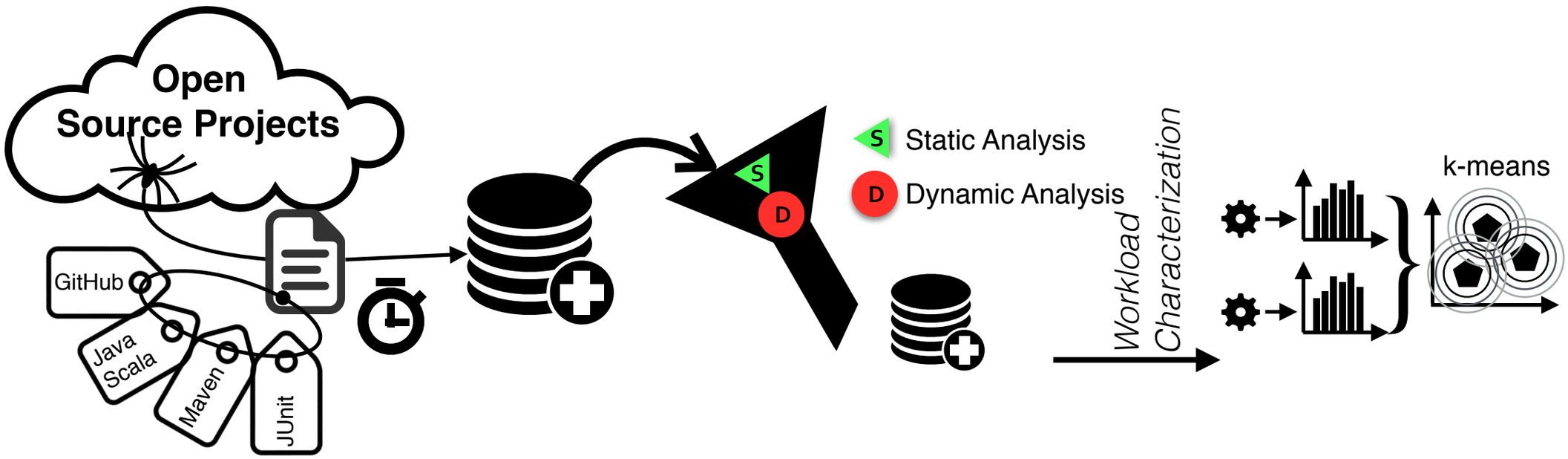
GuardThreadPool.java

```
static Set <Type> typeToCheck = Arrays.asList (
            Executor.class, ExecutorService.class,
            AbstractExecutorService.class, ThreadPoolExecutor.class,
            ForkJoinPool.class)
            .stream().map(Type::getType).collect(Collectors.toSet());

@GuardMethod
     public static boolean isApplicable (final ReflectionStaticContext rsc) {
          return checkType (rsc.thisClass(), typeToCheck);
     }
```

Research question

AutoBench



- Can a **fully automated** process find **open-source workloads** that are suitable as **benchmarks** for **specific evaluation needs**?
- First steps towards *automatic benchmark synthesis*

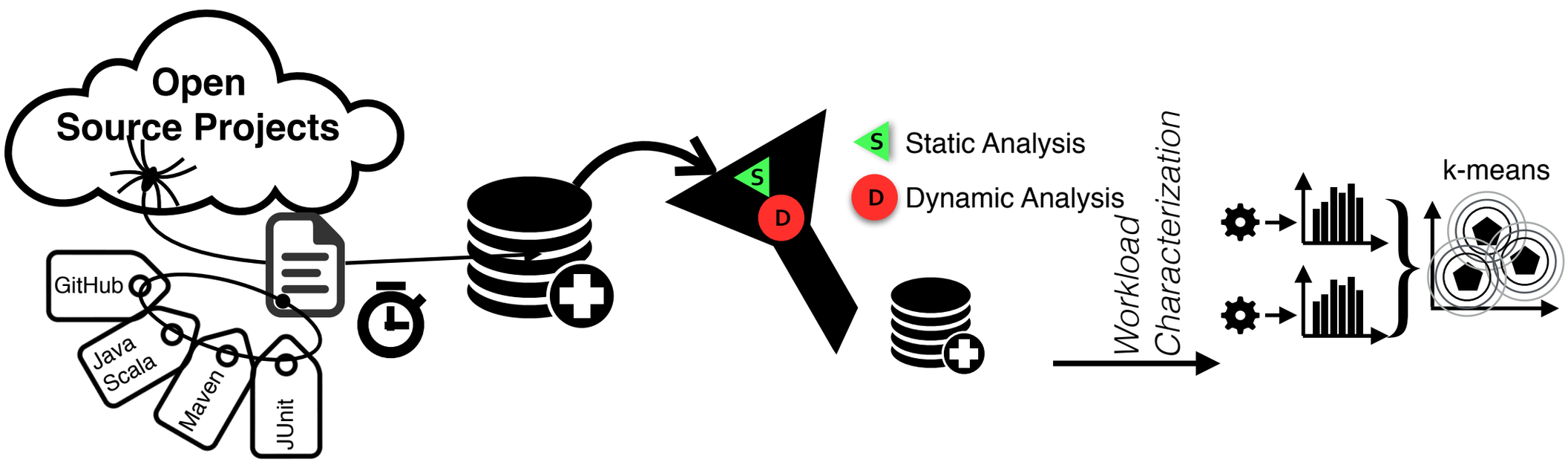
Outline

Can a **fully automated** process find **open-source workloads** that are suitable as **benchmarks** for **specific evaluation needs**?

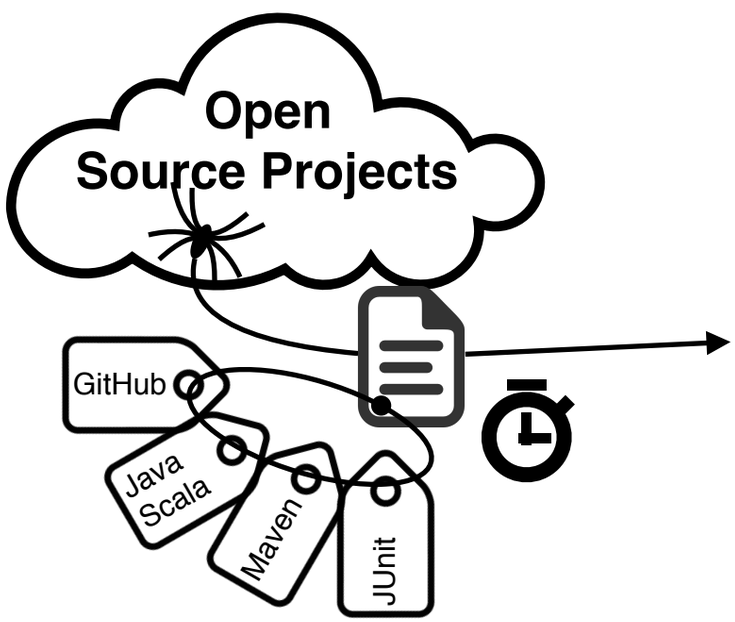
1. AutoBench Methodology
2. Use Cases
3. Preliminary Evaluation Results

AutoBench Methodology

AutoBench


Open
Source Projects
GitHub
Java
Scala
Maven
JUnit
S
D
Static Analysis
Dynamic Analysis
Workload
Characterization
k-means

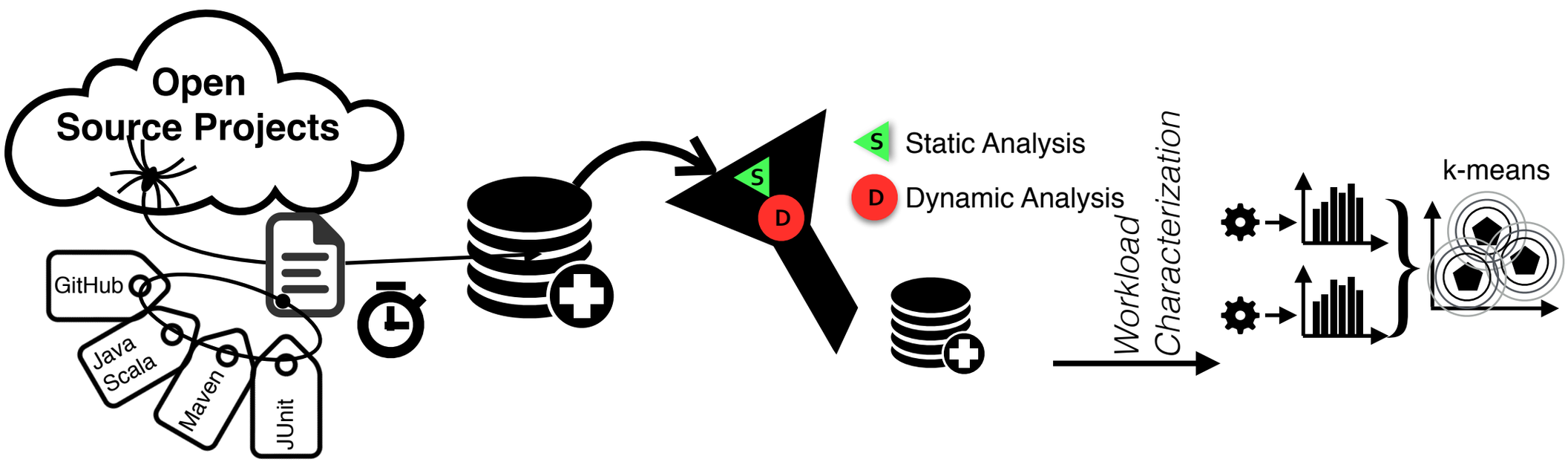
AutoBench



Finding

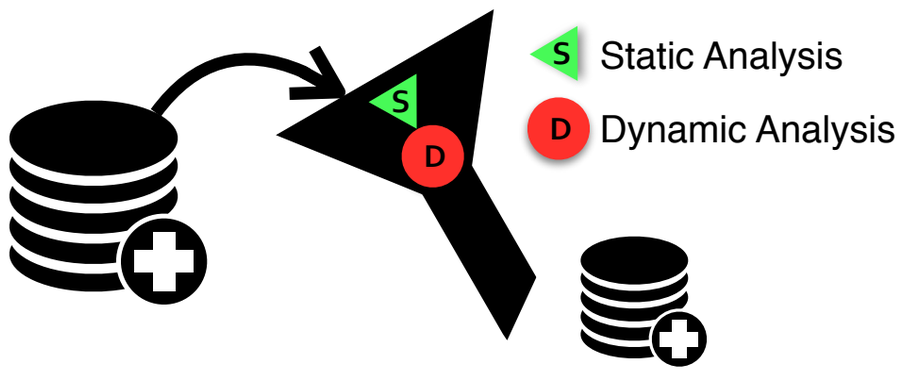
- Find a base of **executable** and **significative** workloads from **open-source** projects

AutoBench



Finding

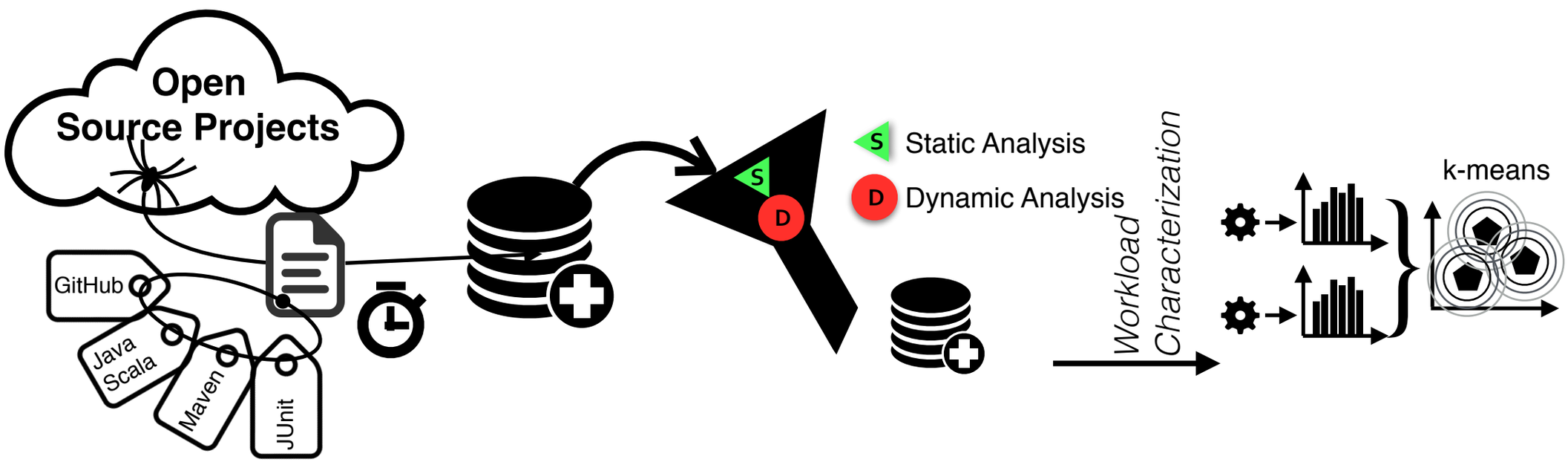
AutoBench



Finding **Filtering**

- Preserve only workloads matching **specific needs**

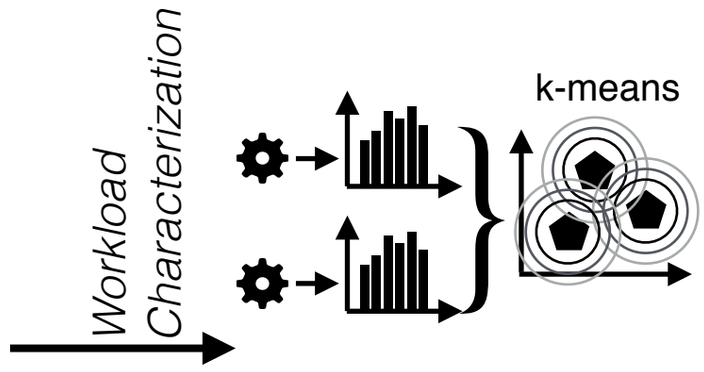
AutoBench



Finding

Filtering

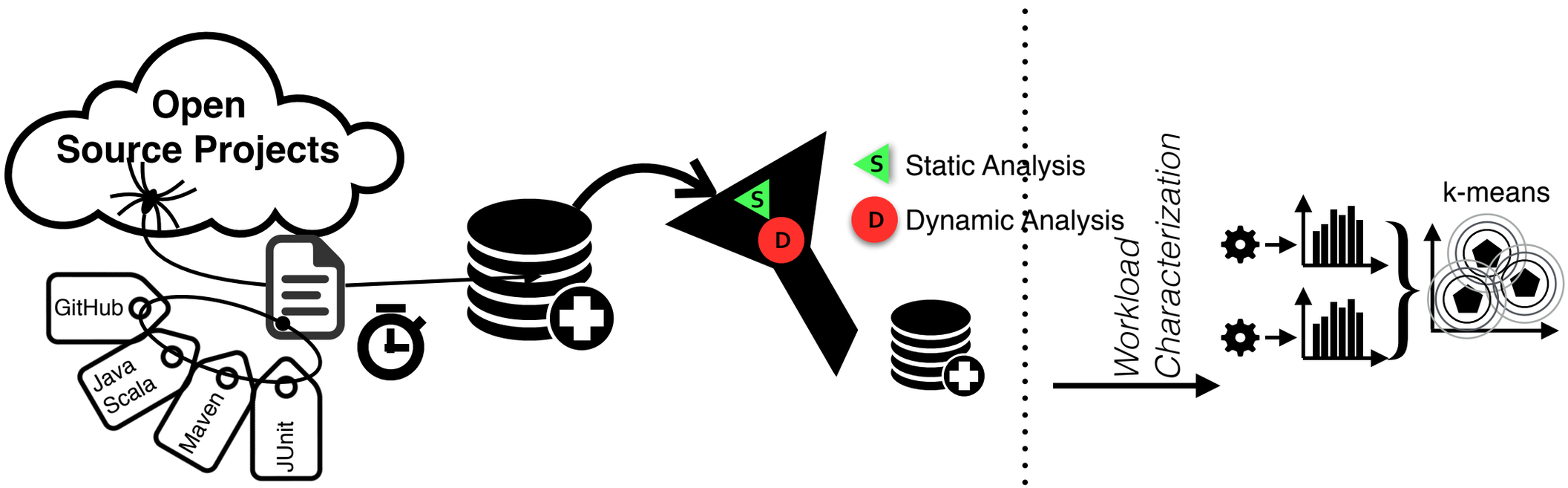
AutoBench



Finding **Filtering** **Characterization**

- Workload characterization identifies a **diverse** set of benchmarks

AutoBench



Finding

Workload
Discovery

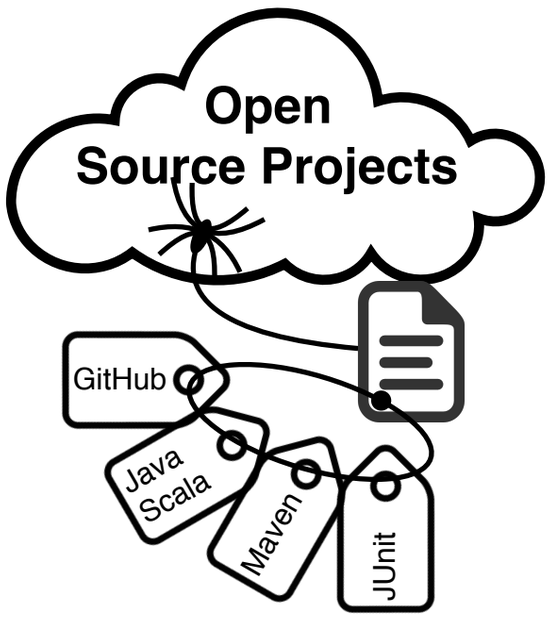
Filtering

Characterization

Benchmark
Synthesis

Optional

Finding



Crawler

Github

- Most popular code hosting site
- 31M projects
- 12M users
- Offers APIs for conditional repository search

Java/Scala

Maven

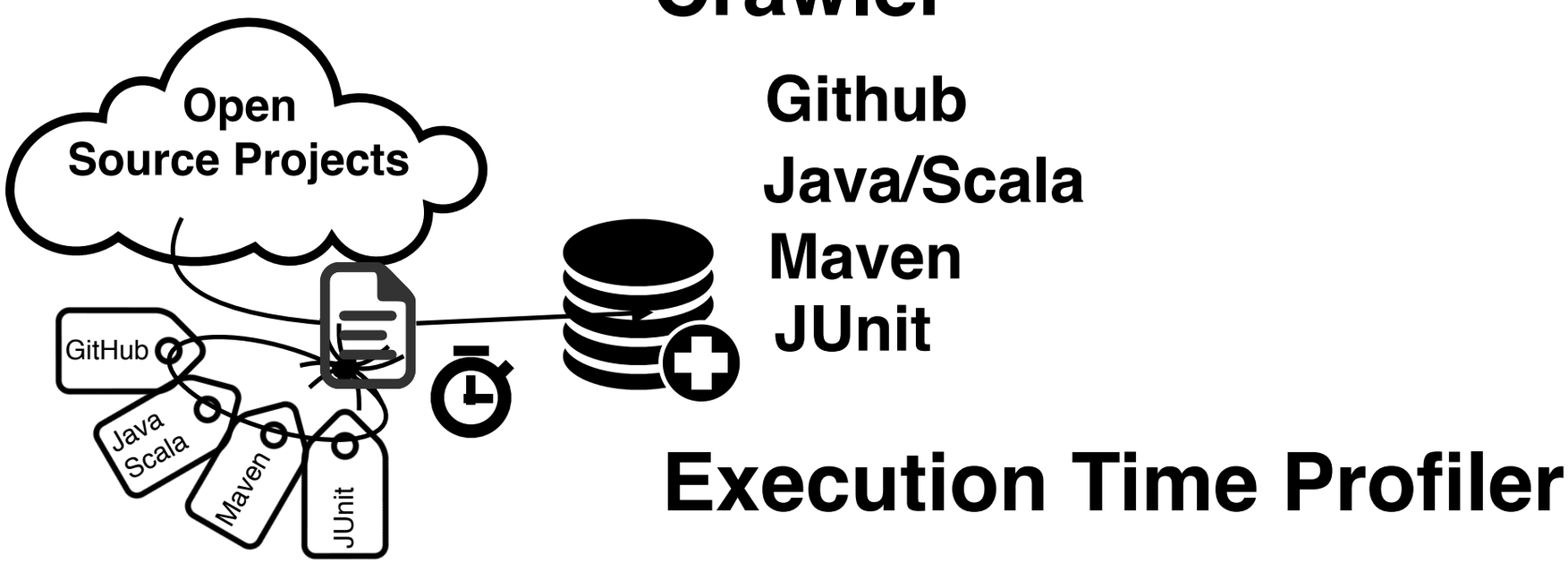
- Automatic process
  Build must be automatic

JUnit

- We target unit tests
  Execution must be automatic

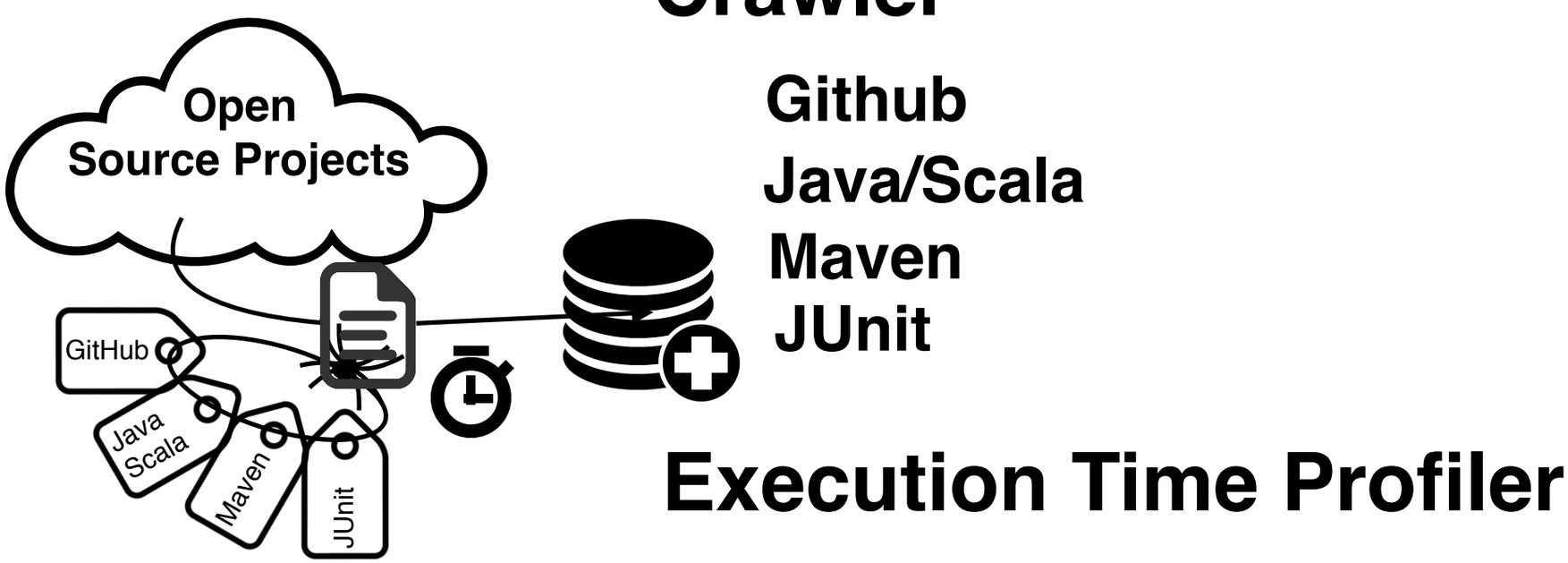
Easily extendible

Finding



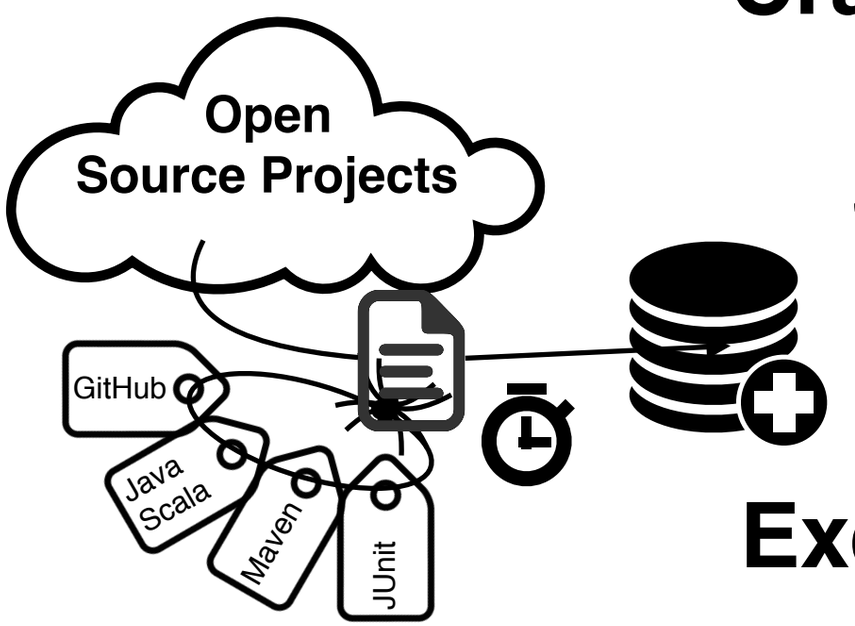
- Idea: significative workload should execute for some time
- Profile execution time of each test
- Filter out too short-running workloads

Finding



- How do we set **time threshold**?
  - Idea: use *existing* and *established* benchmark suites to obtain **baseline** times.
  - Measure **minimum execution time** of benchmarks

Finding



Crawler

Github
Java/Scala
Maven
JUnit

Execution Time Profiler

T = 1s

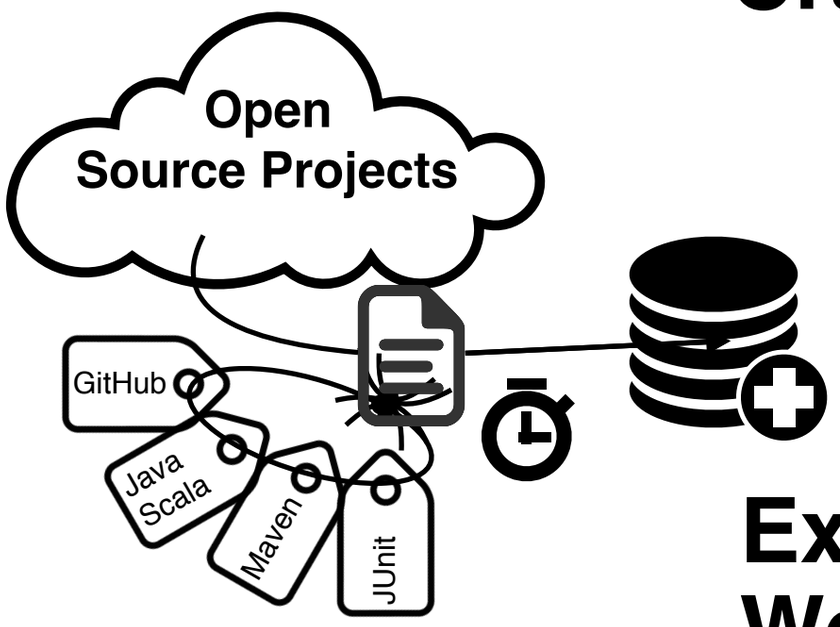
Suite		Min. Exec. Time
DaCapo-9.12	[1]	1.1s
ScalaBench	[2]	0.8s
SPECjvm2008	[3]	0.8s

[1] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur et al., “The DaCapo Benchmarks: Java Benchmarking Development and Analysis,” in ACM OOPSLA, 2006, pp. 169–190.

[2] A. Sewe, M. Mezini, A. Sarimbekov, and W. Binder, “Da Capo con Scala: Design and Analysis of a Scala Benchmark Suite for the Java Virtual Machine,” in ACM OOPSLA, 2011, pp. 657–676.

[3] https://www.spec.org/jvm2008/

Finding



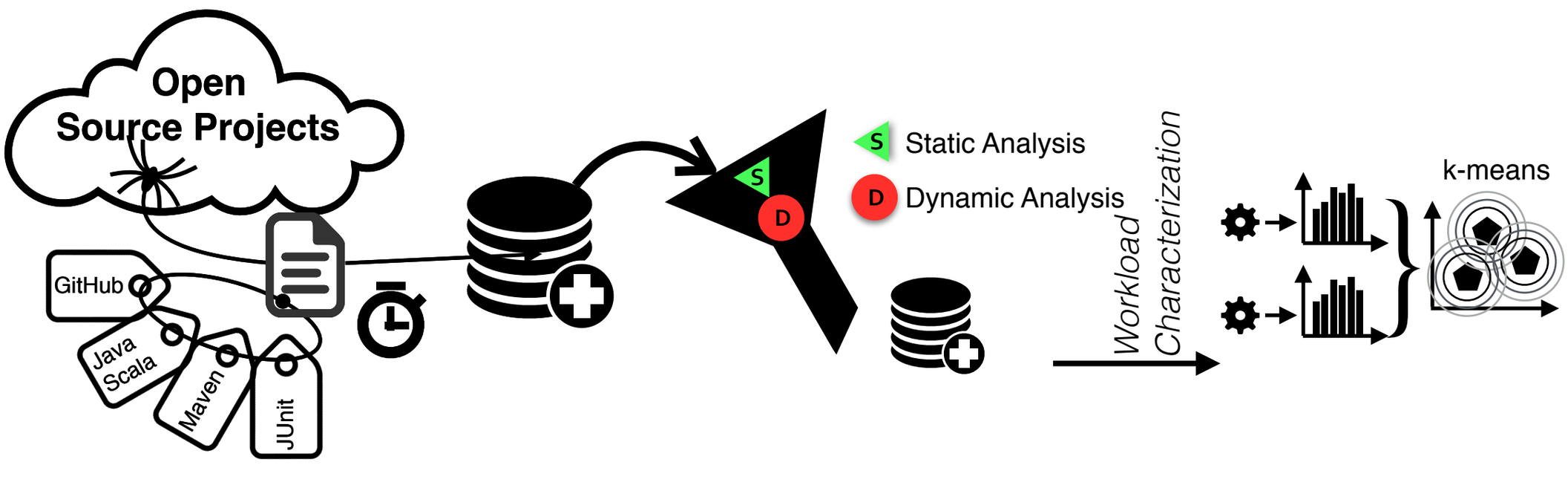
Crawler

Github
Java/Scala
Maven
JUnit

Execution Time Profiler
Workload Database

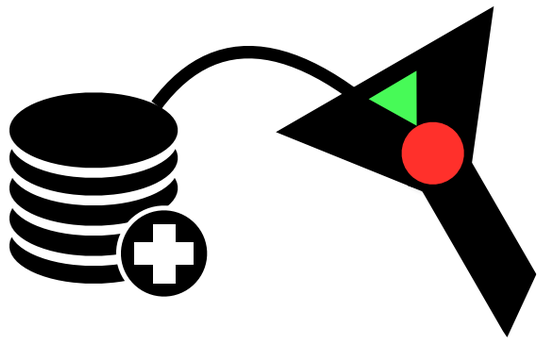
- A base of **significant** and **executable** workloads taken from **real-world open source** projects

AutoBench



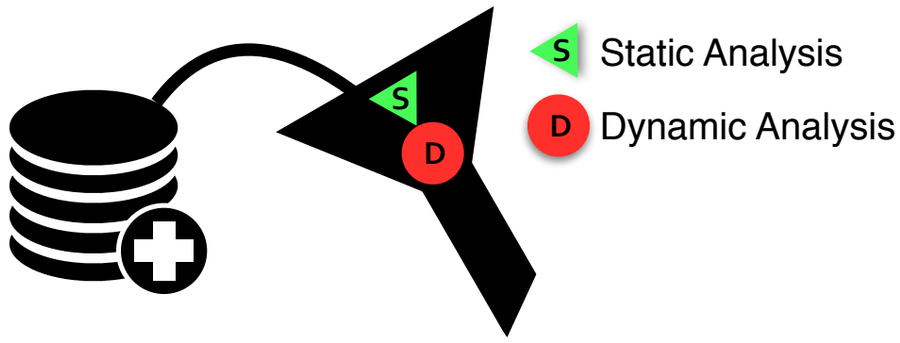
Finding **Filtering** **Characterization**

Filtering



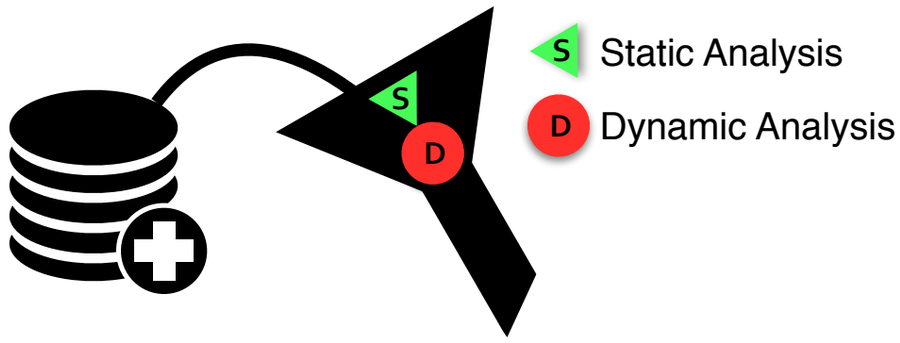
- Preserve only workloads matching specific needs
- **Filter pipeline**

Filtering



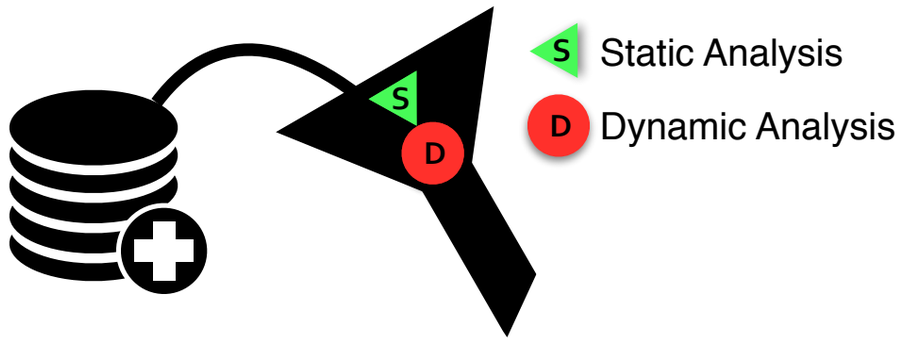
- User-specific needs
  - User-specific filters
- **Static/dynamic** analysis performed on test methods
  - Users write their own analysis and requirements

Filtering



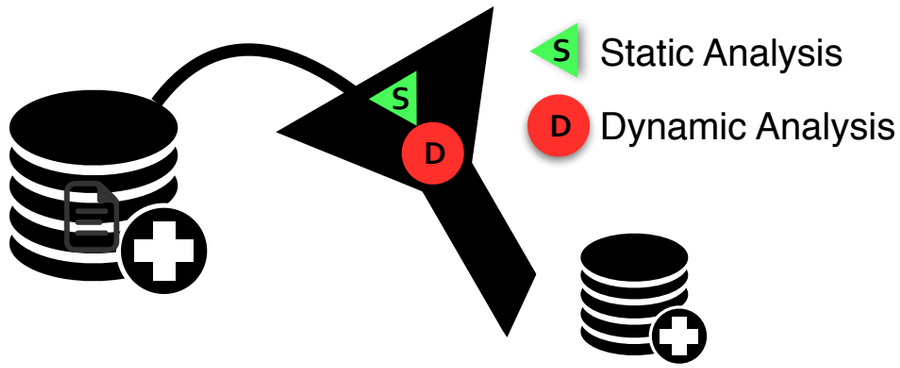
- Examples:
  - Parsing of imported libraries
  - Source code analysis
  - Bytecode parsing
  - …

Filtering

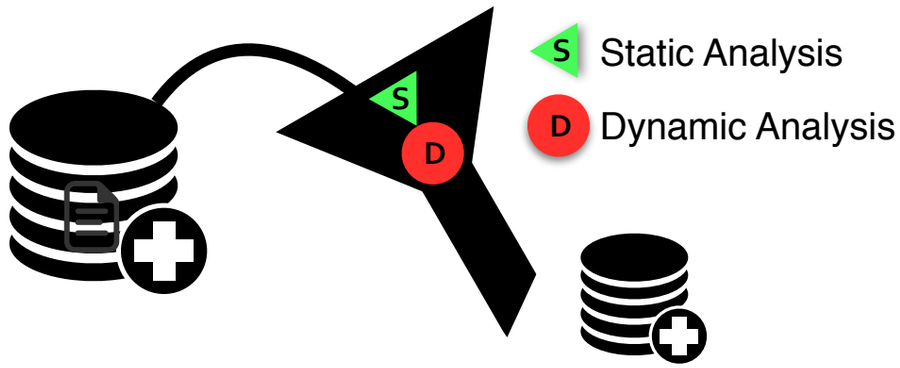


- Examples:
  - Runtime validation of class usage
  - Detection of called APIs
  - Execution time profiling
  - N. of allocated objects
  - …
- AutoBench integrates **DiSL**
  - User can easily write dynamic analyses

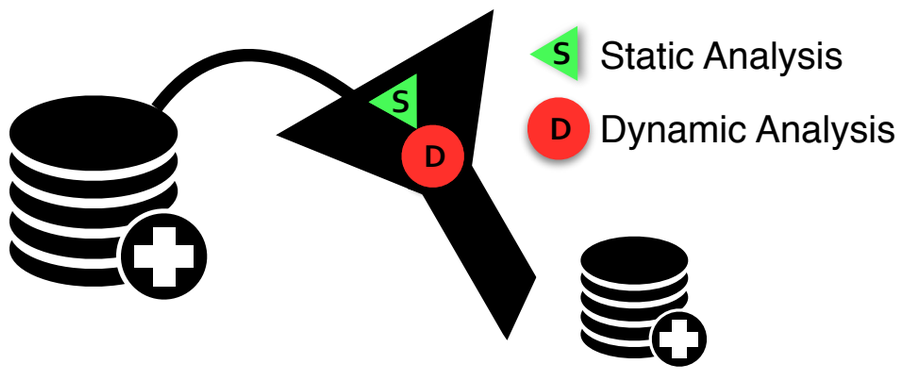
Filtering



Filtering

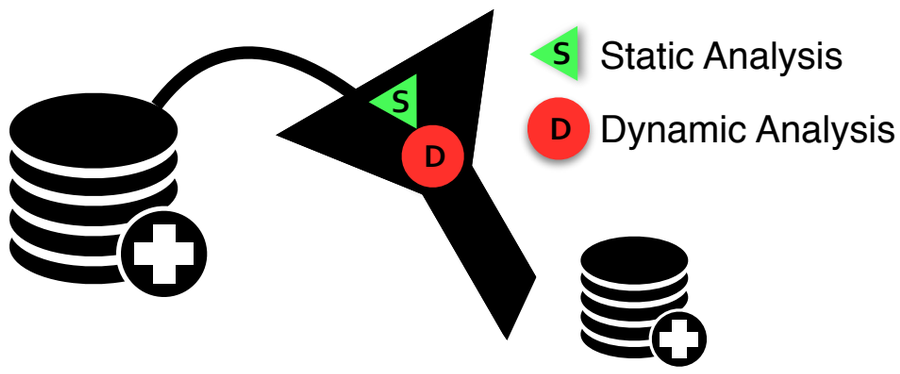


Filtering



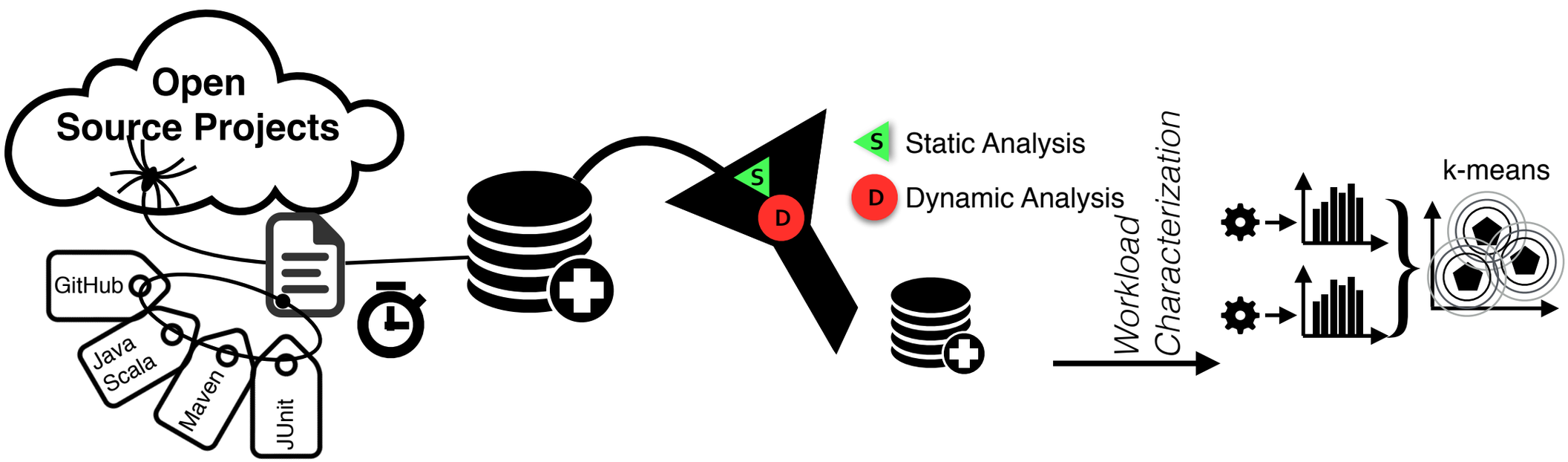
- Filters can be arranged in any order

Filtering



- Final database contains only **significant** and **executable** workloads matching **specific needs**

AutoBench



Finding **Filtering** **Characterization**

Characterization

- Suitable unit tests can be **numerous**
- Workload should be **diverse**

Characterization

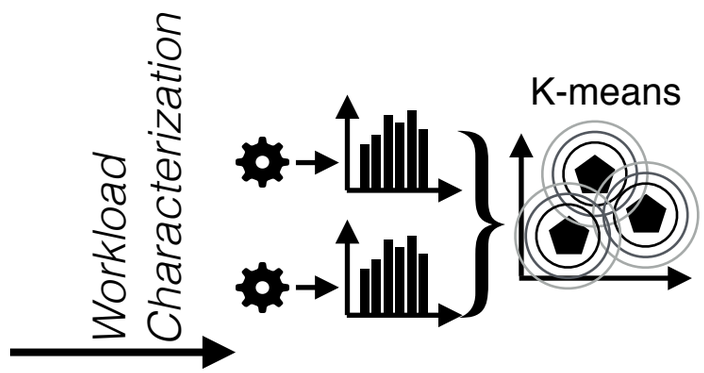
- Suitable unit tests can be **numerous**
- Workload should be **diverse**
- **Characterization** helps selecting diversifies workloads

- **Diversity**
  - User-defined requirements

Characterization

- **Metrics**

Object allocations
Method invocations
Execution time



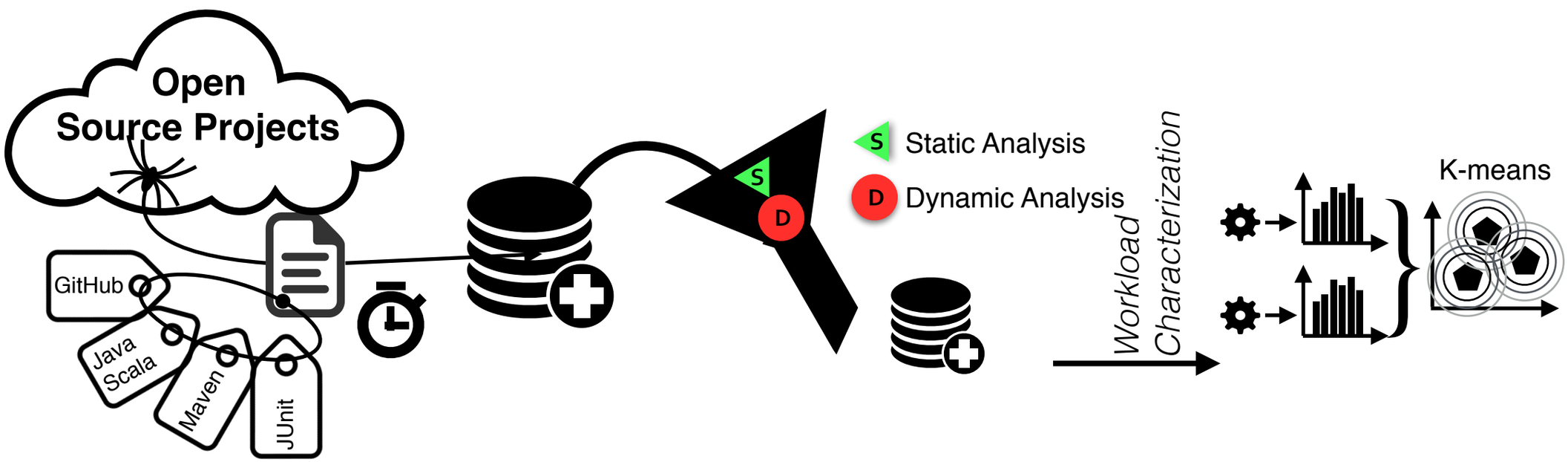
- **Measurement**

Dynamic analysis with DiSL

- **Identification**

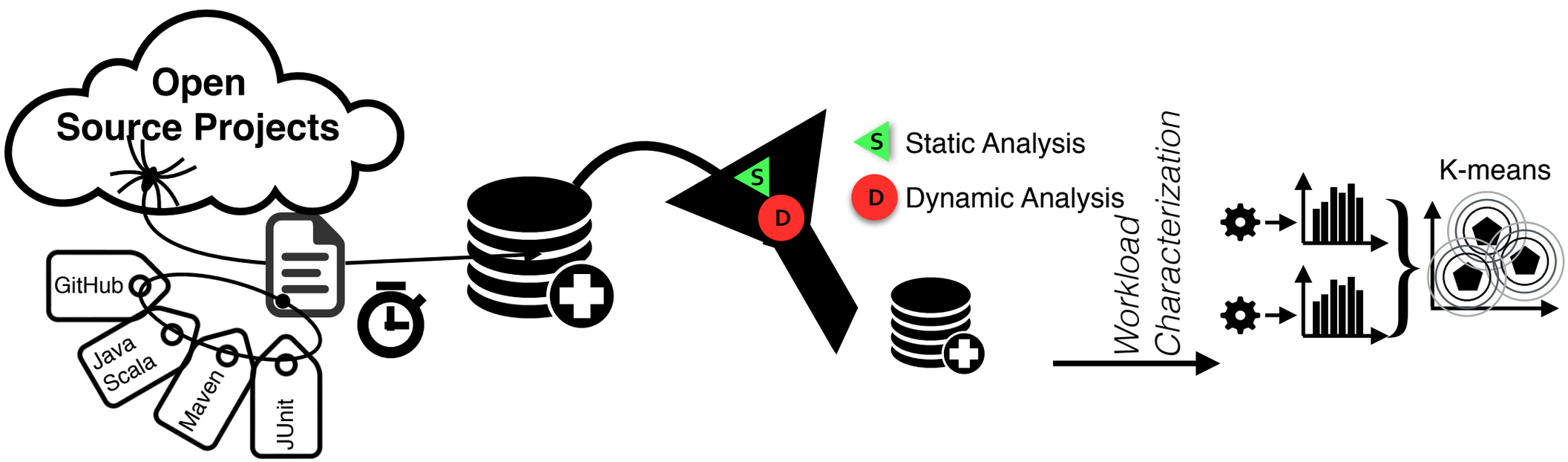
K-means

Characterization



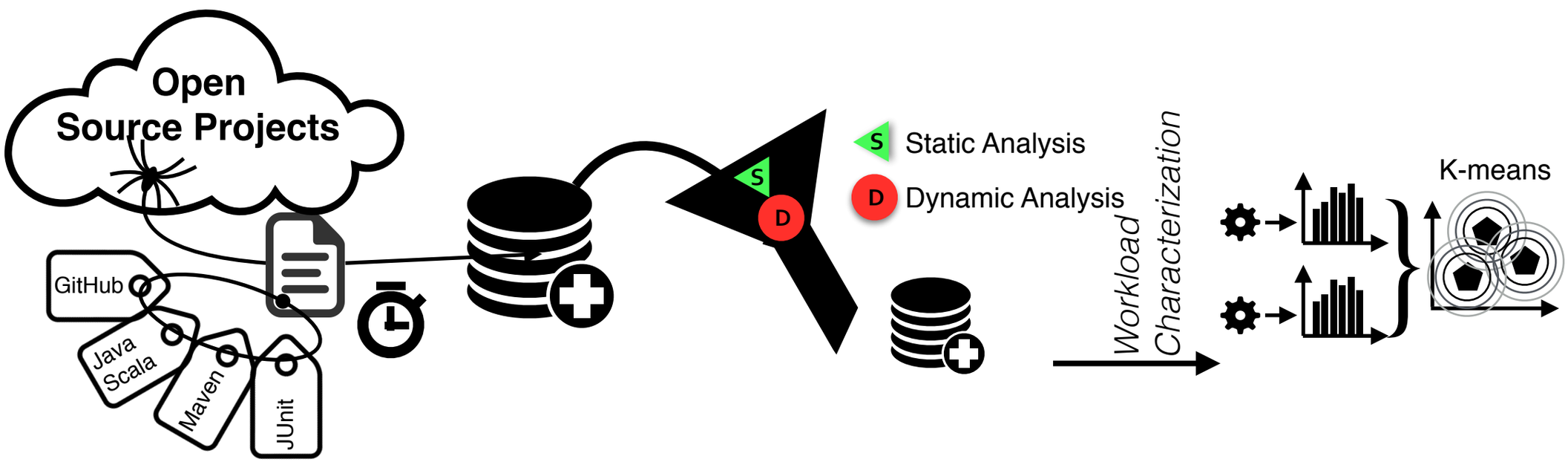
- Optional
- K-means can be replaced with other characterizations
- Output:
  - Set of **diverse**, **suitable** and **executable** workloads
  - **Reduced** set of unit tests (**K**)

Characterization



- How to choose K?
  - Again: use *existing* and *established* suites to obtain minimum number of benchmarks

Characterization



Suite		N. Benchmarks
DaCapo-9.12	[1]	14
ScalaBench	[2]	12
SPECjvm2008	[3]	16

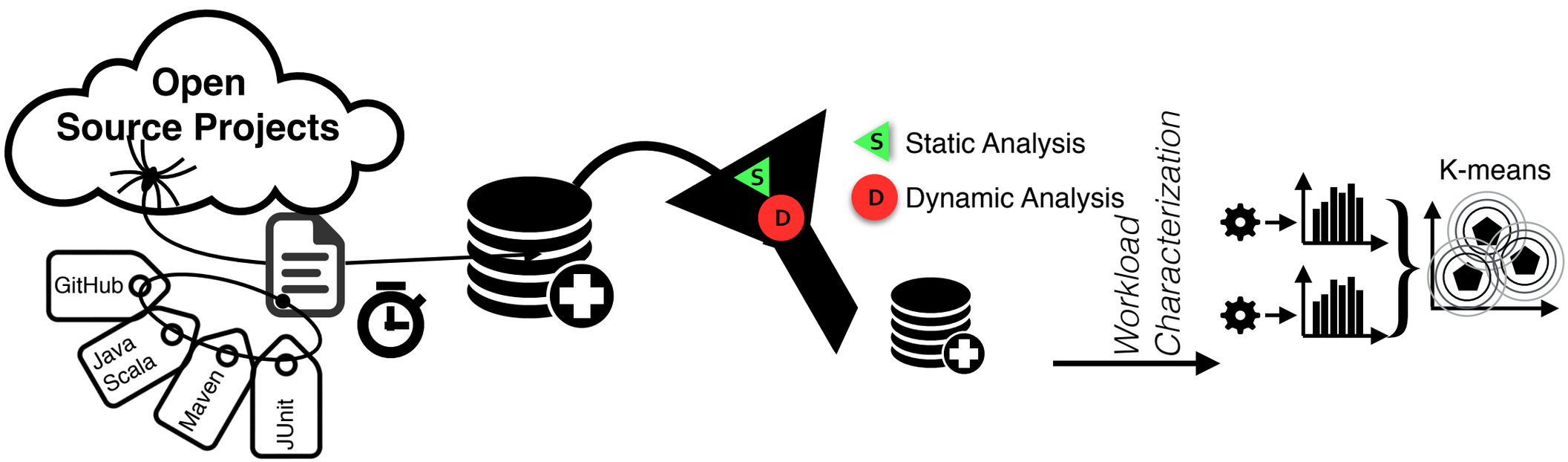
K = 12

[1] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur et al., “The DaCapo Benchmarks: Java Benchmarking Development and Analysis,” in ACM OOPSLA, 2006, pp. 169–190.

[2] A. Sewe, M. Mezini, A. Sarimbekov, and W. Binder, “Da Capo con Scala: Design and Analysis of a Scala Benchmark Suite for the Java Virtual Machine,” in ACM OOPSLA, 2011, pp. 657–676.

[3] https://www.spec.org/jvm2008/

AutoBench



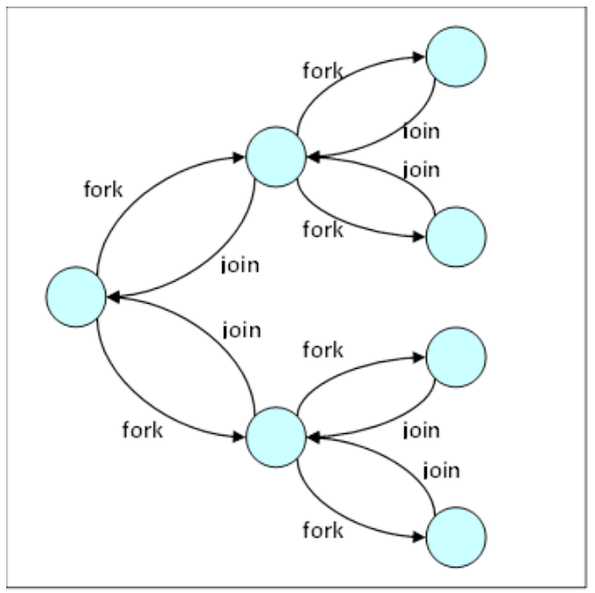
Finding **Filtering** **Characterization**

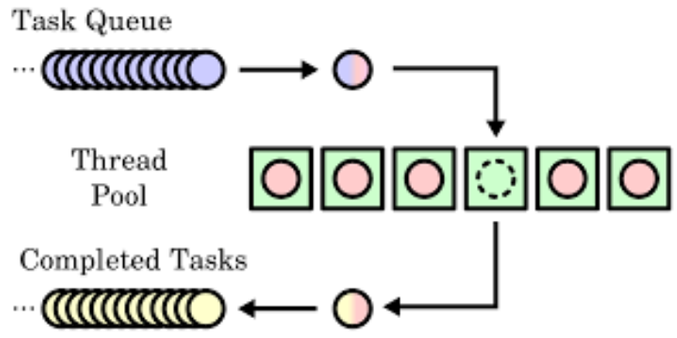
Use Cases

Executor

- A researcher is investigating the usage of task execution frameworks in Java applications

- His goal:
  - performance modeling
  - investigation of scalability issues
  - validation of new optimizations

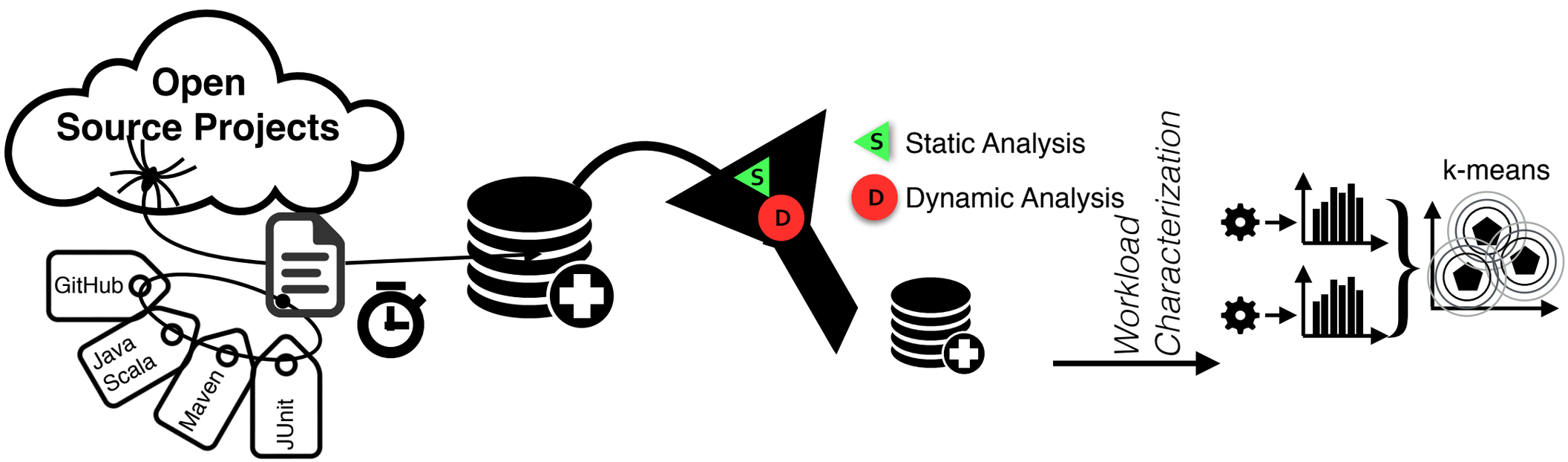




Executor

- Suites may be **old**
  - More recent workload is needed
- **Custom** task execution frameworks could be used
  - Should not focus only on standard Java task execution frameworks(e.g., ThreadPoolExecutor, ForkJoinPool)
- Need of **real-world** workloads
- New benchmarks are needed

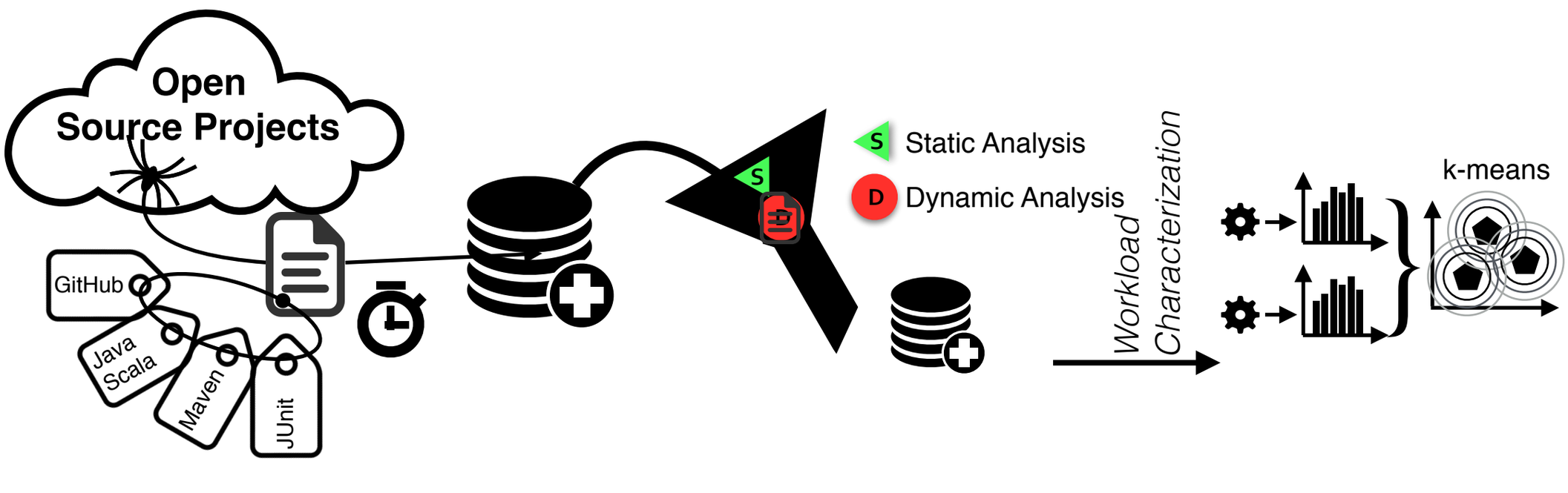
Executor



- Parsing of imported classes

```
classToFilter =  Array("java.util.concurrent.Executor",
                       "java.util.concurrent.ExecutorService",
                       "java.util.concurrent.AbstractExecutorService",
                       "java.util.concurrent.ThreadPoolExecutor",
                       "java.util.concurrent.ForkJoinPool");
mode = FilterMode.Imports
```

Executor



- Verify that a task execution framework is used at runtime

`invokedynamic`

- A JVM developer is tasked with optimizing the execution of a new bytecode instruction

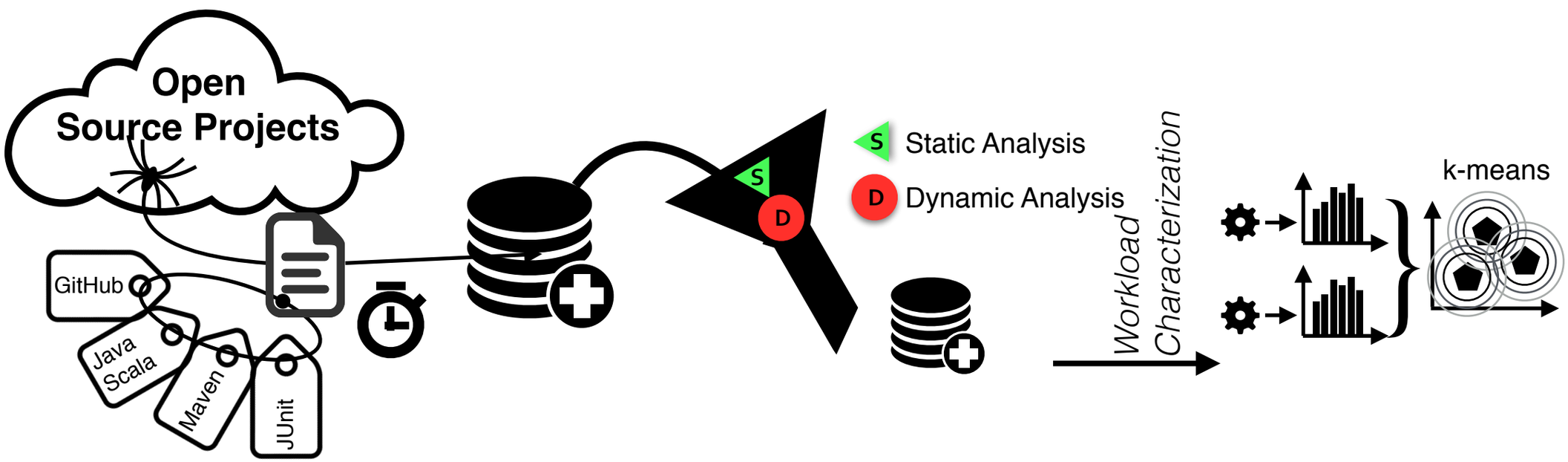
```
                                                  Byte Offset
i = j + k;          1     ILOAD j    // i = j + k     0    0x15 0x02
if (i == 3)         2     ILOAD k                     2    0x15 0x03
    k = 0;          3     IADD                        4    0x60
else                4     ISTORE i                    5    0x36 0x01
   j = j – 1;       5     ILOAD i    // if (i < 3)     7    0x15 0x01
                    6     BIPUSH 3                    9    0x10 0x03
                    7     IF_ICMPEQ L1               11    0x9F 0x00 0x0D
                    8     ILOAD j    // j = j – 1     14    0x15 0x02
                    9     BIPUSH 1                   16    0x10 0x01
                   10     ISUB                       18    0x64
                   11     ISTORE j                   19    0x36 0x02
                   12     GOTO L2                    21    0xA7 0x00 0x07
                   13 L1: BIPUSH 0     // k = 0      24    0x10 0x00
                   14     ISTORE k                   26    0x36 0x03
                   15 L2:                            28
```

- In this example: `invokedynamic`

- His goal:
  - investigate related performance improvements
  - evaluate usage in real-world workloads

- Suites may be **too old**
  - More recent workload is needed
- User need is **very specific**
  - General-purpose suites may simply not use `invokedynamic`
- Need of **real-world** workloads
- New benchmarks are needed

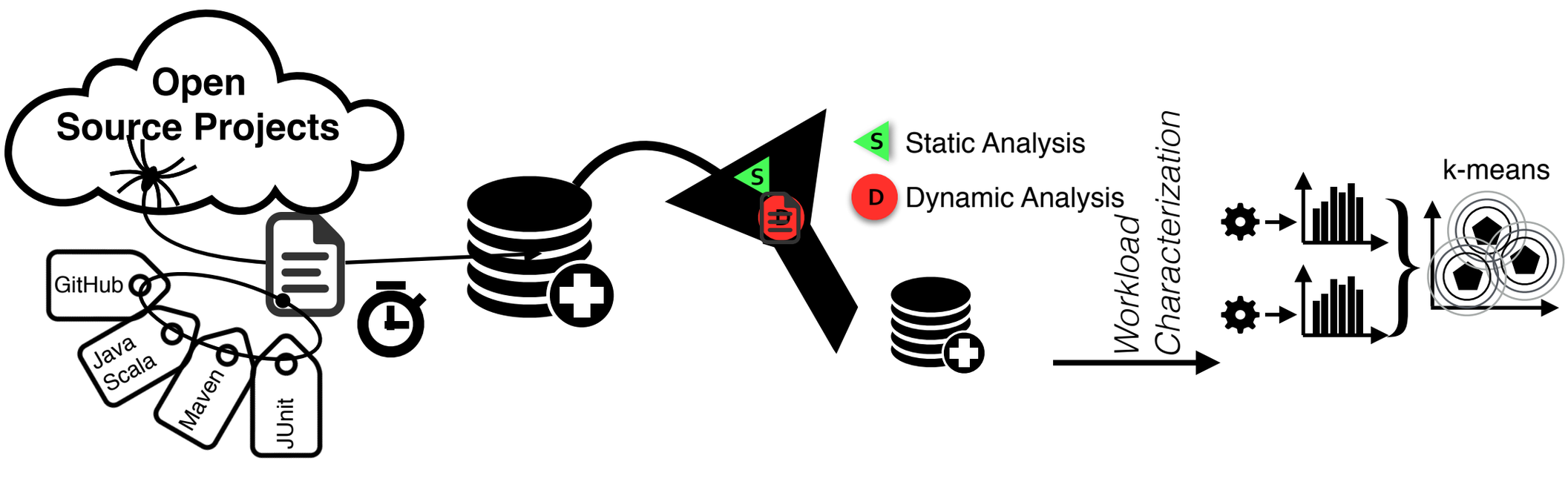
invokedynamic



- Look for usage of Lambda expressions
- Library import parsing (java.lang.invoke)
- Bytecode parsing

```
regex = "*->*"
classToFilter =  "java.lang.invoke.*")
bytecode = "invokedynamic"
mode = FilterMode.Combined
```

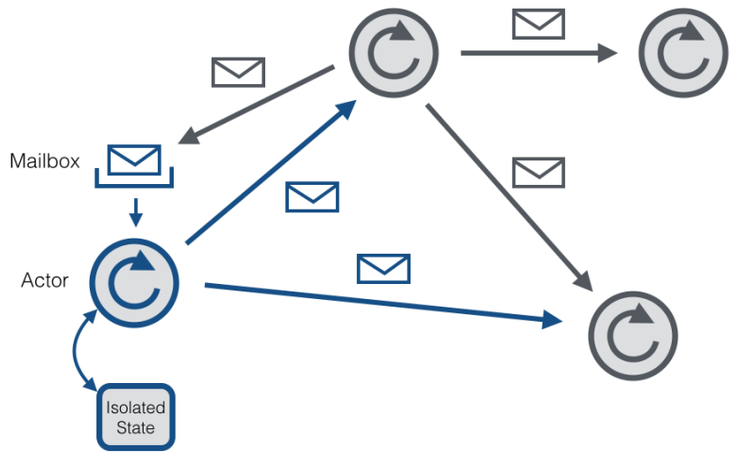
invokedynamic



- Verify that an `invokedynamic` bytecode is effectively executed at runtime

Actors

- A developer is looking for realistic workloads utilizing actor libraries



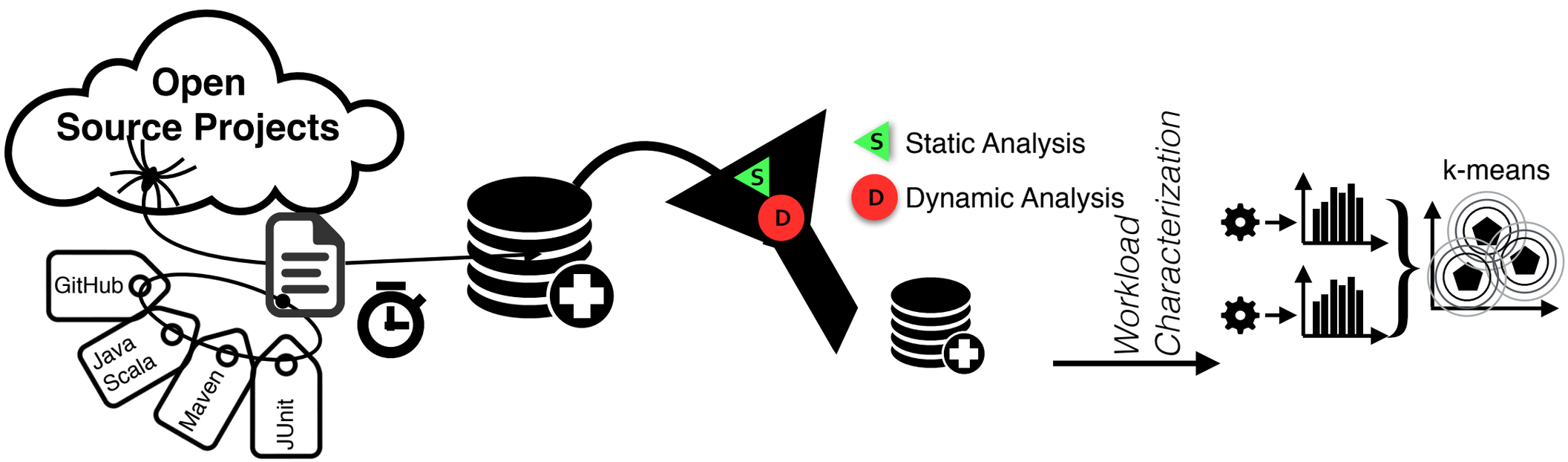
- His goal:
  - discover optimizations related to actor libraries
  - reuse discovered optimizations and tricks in his own actor-based applications

Actors

- Suites may be **unrelated**
  - They just do not use actor libraries
- Some benchmarks for actor applications exist, but are **not suitable**
  - E.g., Savina benchmark suite [1]
  - However, focus on inter-library comparison rather than real workloads
- Need of **real-world** workloads
- New benchmarks are needed

[1] S. M. Imam and V. Sarkar, “Savina - An Actor Benchmark Suite: Enabling Empirical Evaluation of Actor Libraries,” in AGERE!, 2014, pp. 67–80.

Actors

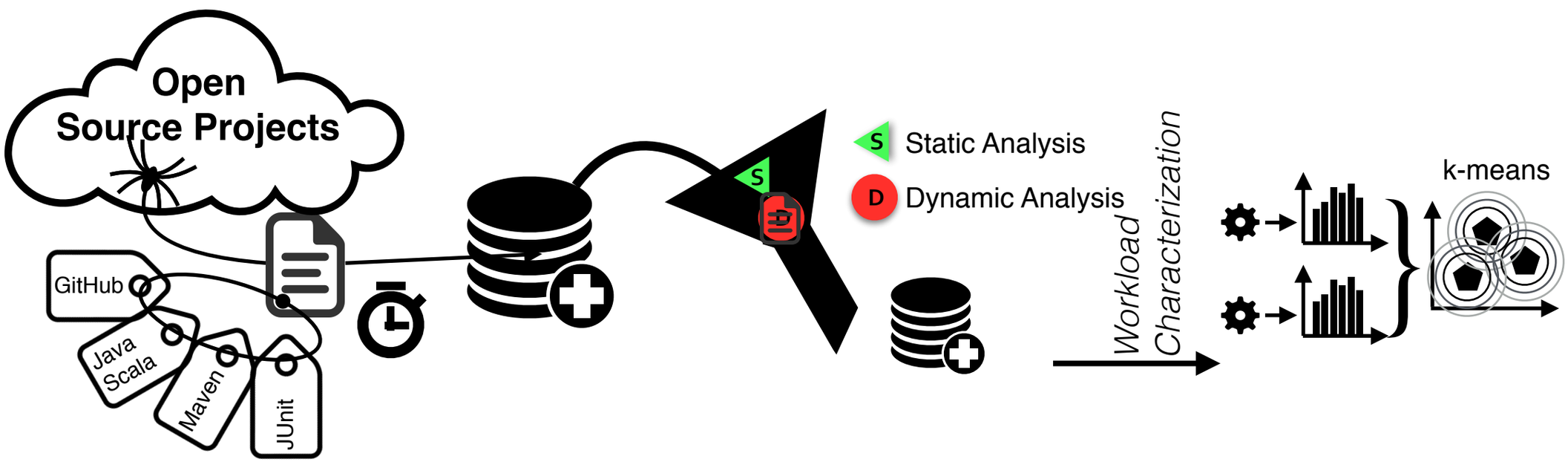


- Parsing of imported classes/interfaces

```
classToFilter = Array("akka.actor.Actor", "akka.actor.UntypedActor",
            "scala.actors.Actor","fj.control.parallel.Actor",
            "groovyx.gpars.actor.Actor",  "edu.rice.hj.api.HjActor",
            "fi.jumi.actors.Actors",
            "net.liftweb.actor.LiftActor","scalaz.consurrent.Actor");

mode = FilterMode.Imports
```

Actors



- Verify that at least an actor is created/executed at runtime

Preliminary Evaluation Results

Evaluation

Can a fully automated process find open-source workloads that are suitable as benchmarks for specific evaluation needs?

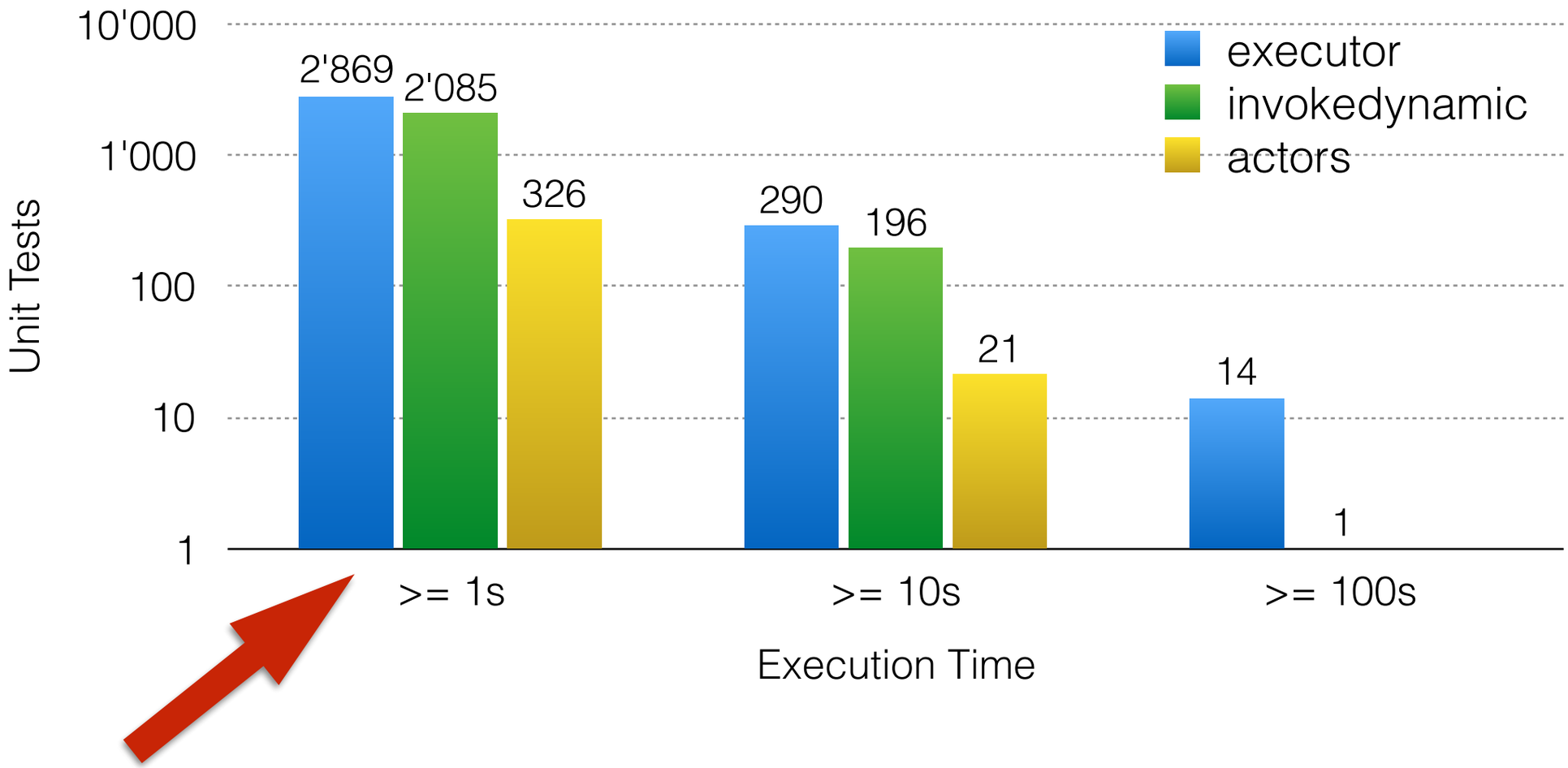
1. Can we expect to find tests that are **long enough** to serve as benchmark workload?

2. Can dynamic analysis **improve identification** of workloads matching a specific evaluation context?

3. Can we find **diversity** in the workloads that would allow synthesizing a benchmark suite?

Evaluation

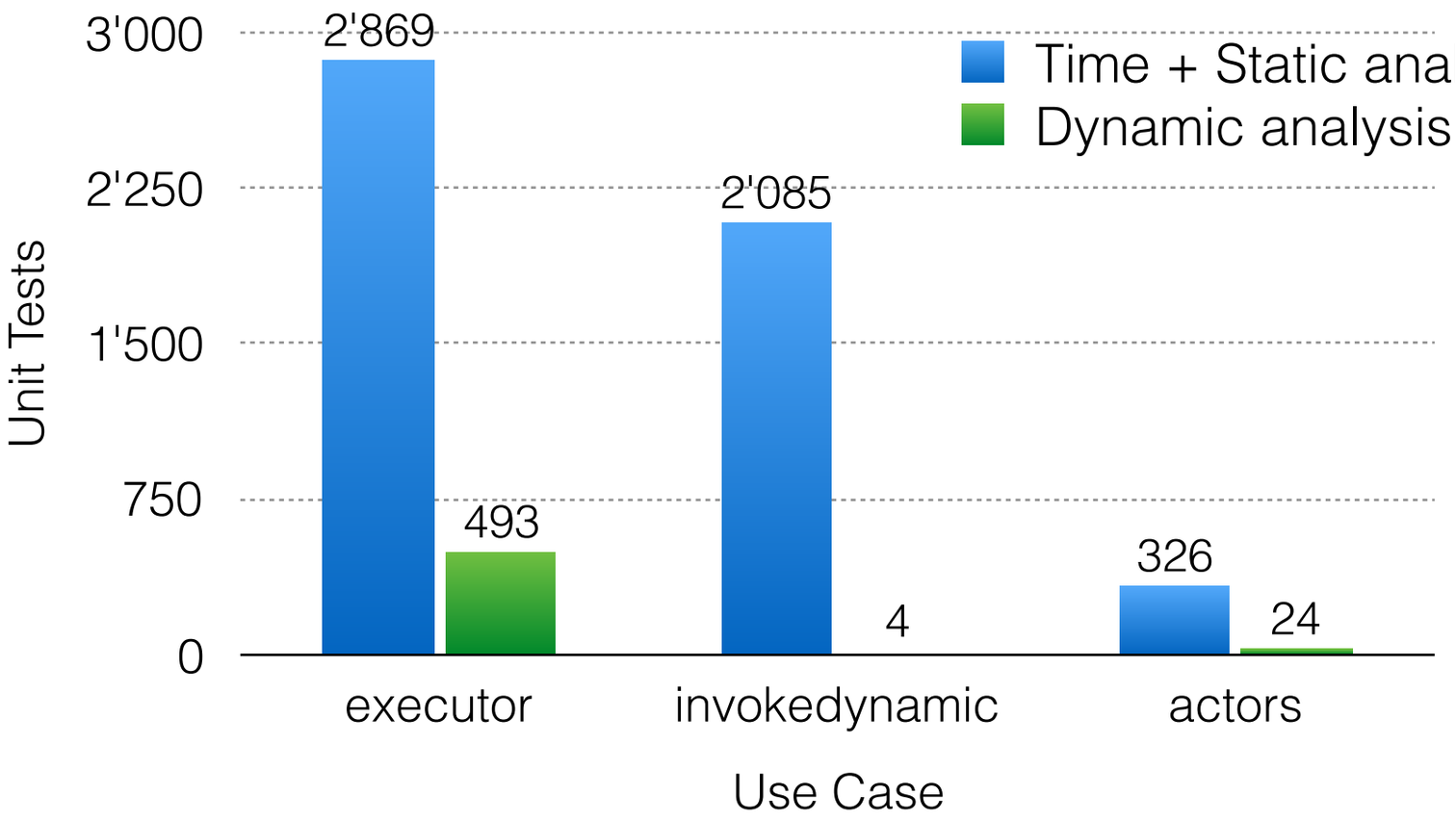
1. Can we expect to find tests that are long enough to serve as benchmark workload?



☑ In all use cases, a significant number of tests run for more than 1s

Evaluation

2. Can dynamic analysis improve identification of workloads matching a specific evaluation context?



Stage	%
executor	17.18%
invokedynamic	0.19%
actors	7.36%

☑ Dynamic analysis helps selecting only suitable workloads

Evaluation

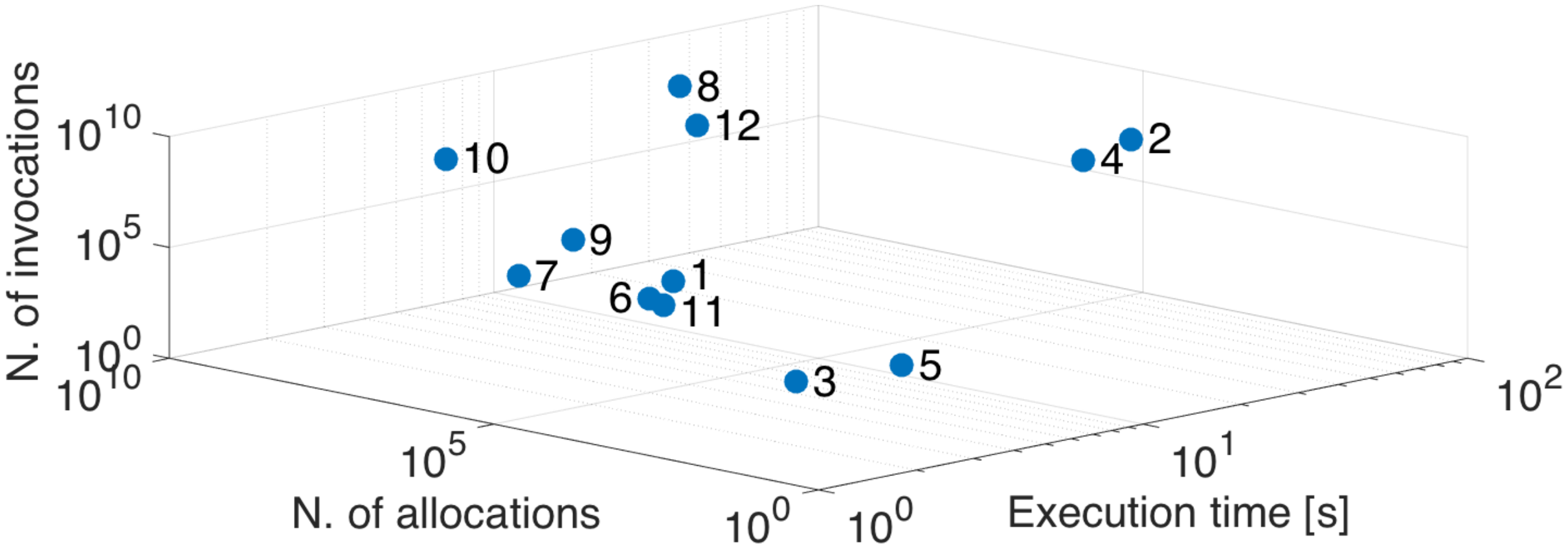
3. Can we find diversity in the workloads that would allow synthesizing a benchmark suite?

Executor: 493 unit tests after dynamic filtering

#	Project	Unit Test	Time [s]	Allocations	Invocations
1	prova	test/ws/prova/test2/ProvaWorkflowsTest.predicate_join	2.523	17938	469320
2	datacube	com/urbanairship/datacube/HBaseBackfillerTest.testMutationsWhileBackfilling	96.806	127961	7628592
3	jdeferred	org/jdeferred/impl/FilteredPromiseTest.testNoOpFilter	2.003	72	755
4	datacube	com/urbanairship/datacube/BackfillExampleTest.test	62.371	78728	4367224
5	cmb	com/comcast/cmb/test/unit/CassandraTest.testCassandraCounters	4.007	54	636
6	svarut	no/kommune/bergen/soa/svarut/ServiceContextTest.init	1.983	12510	205187
7	antlr4	org/antlr/v4/test/runtime/java/TestPerformance.testExpressionGrammar_1	1.212	109350	2595015
8	prova	test/ws/prova/test2/ProvaMessagingTest.ring_parallel	17.961	251119179	3144752537
9	pangool	com/datasalt/pangool/tuplemr/mapred/lib/output/TestMultipleOutputs.test	2.088	241759	12943082
10	prova	test/ws/prova/test2/ProvaFunctionalProgrammingTest.func_reactive_unfoldr_iteration_perf_large	2.690	78180455	928571592
11	graphdb	org/neo4j/backup/TestBackup.fullThenIncremental	2.159	11568	88334
12	prova	test/ws/prova/test2/ProvaMetadataTest.cep005	13.267	30393192	443299688

Evaluation

3. Can we find diversity in the workloads that would allow synthesizing a benchmark suite?



☑ From 493 workload candidates to 12 representative and diverse ones

Evaluation

- Can a **fully automated** process find **open-source workloads** that are suitable as **benchmarks** for **specific evaluation needs**?

☑ Positive evaluation

☑ Encouraging results

☑ Further research is promising

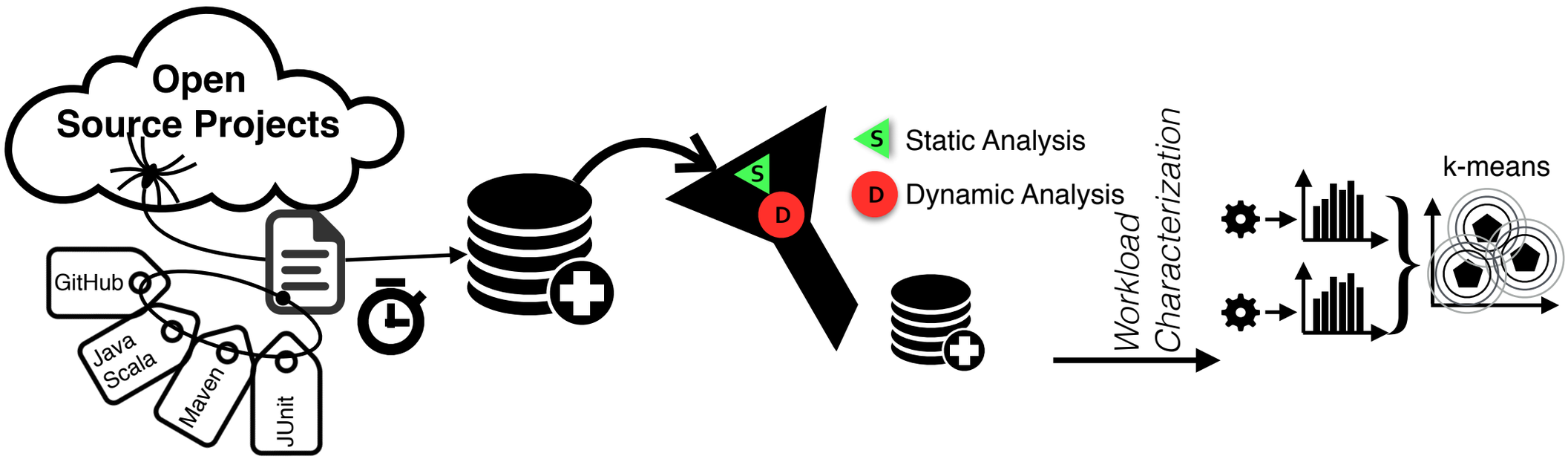
Limitations

- Representativeness of unit tests wrt. real-world workloads
- Potential inclusion of “garbage" projects
- Non-deterministic execution with deterministic output
  - E.g.: randomized algorithms
- Execution time as metric for significance
  - Too architecture-specific
- Potential security issues
- Target single hosting site, build system, testing framework

Conclusions

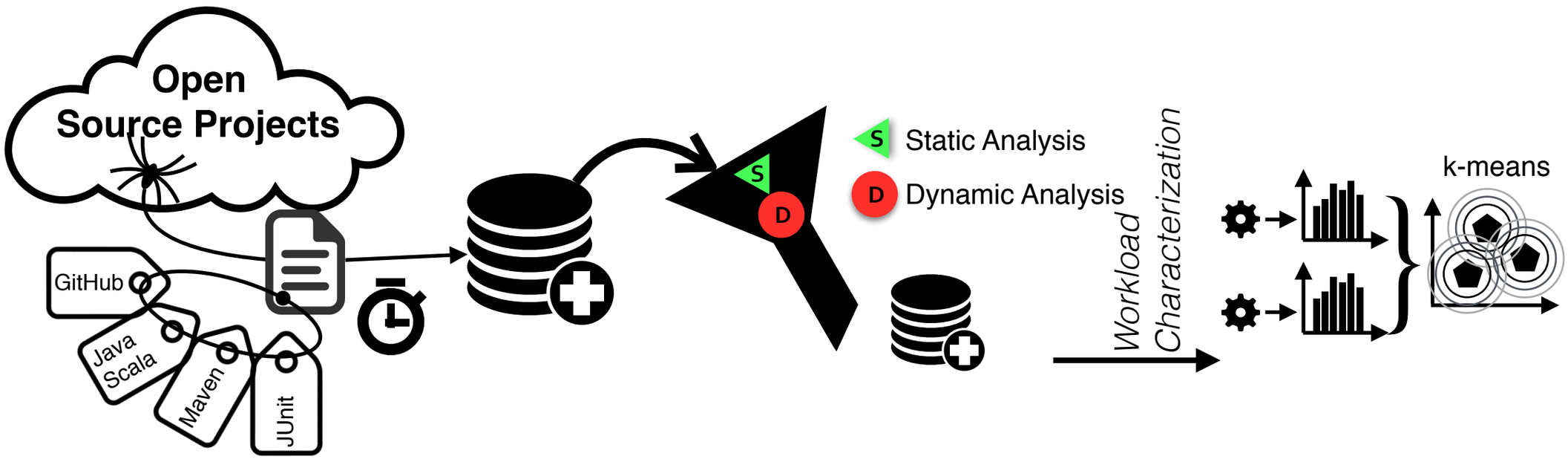
- We investigated the feasibility of using **unit tests as workloads** in custom benchmarks
- **AutoBench**: a methodology and toolchain to *find*, *filter*, and *classify* workloads from open-source projects
- Encouraging results towards **automatic generation of benchmark suites** for specific needs.
- Future work:
  - DSL for expressing workload properties and optimization goals
  - Ranking of similar workloads instead of filtering
  - Incremental filtering reusing results of previous analyses

AutoBench



- More information in:
  - Yudi Zheng, Andrea Rosà, Luca Salucci, Yao Li, Haiyang Sun, Omar Javed, Lubomír Bulej, Lydia Y. Chen, Zhengwei Qi and Walter Binder. “AutoBench: Finding Workloads That You Need Using Pluggable Hybrid Analyses”. In IEEE SANER 2016 (ERA paper).
  - Slides: `http://inf.usi.ch/phd/rosaa/autobench_kyoto_2016_03_22.pdf`
- Contact detail:
  - Andrea Rosà
    andrea.rosa@usi.ch
    http://www.inf.usi.ch/phd/rosaa

Questions?



- More information in:
  - Yudi Zheng, Andrea Rosà, Luca Salucci, Yao Li, Haiyang Sun, Omar Javed, Lubomír Bulej, Lydia Y. Chen, Zhengwei Qi and Walter Binder. “AutoBench: Finding Workloads That You Need Using Pluggable Hybrid Analyses”. In IEEE SANER 2016 (ERA paper).
  - Slides: `http://inf.usi.ch/phd/rosaa/autobench_kyoto_2016_03_22.pdf`
- Contact detail:
  - Andrea Rosà
    andrea.rosa@usi.ch
    http://www.inf.usi.ch/phd/rosaa