Bringing Ultra-Large-Scale Software Repository Mining to the Masses with Boa

Robert Dyer

November 8, 2013

Department of Computer Science

Iowa State University

The research and educational activities described in this talk were supported in part by the US National Science Foundation (NSF) under grants CCF-13-49153, CCF-13-20578, TWC-12-23828, CCF-11-17937, CCF-10-17334, and CCF-10-18600.

Research Overview

Dynamic Aspect Virtual Machine Support - Nu

[AOSD'08] [TOSEM]

Language Evaluation - Ptolemy

[AOSD'12] [TAOSD]

Easing Ultra-large-scale Software Mining - Boa

[ICSE'13] [GPCE'13] [SPLASH'13 SRC]

In submission: [ICSE'14] Planned: [PLDI'14]

What is actually practiced

Keep doing what works

To find better designs

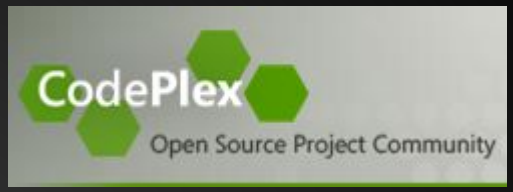
Empirical validation

Spot (anti-)patterns

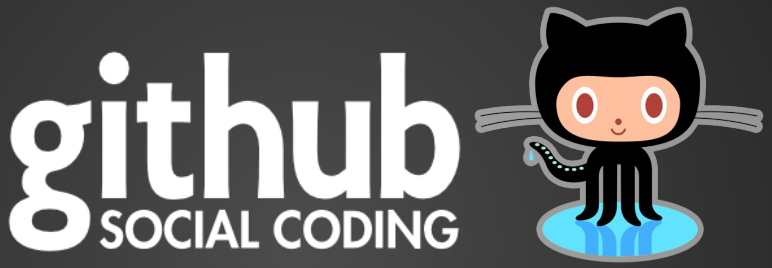
Why mine software repositories?

Learn from the past → **Inform the future**


Google code


CodePlex
Open Source Project Community


Apache


github
SOCIAL CODING


Java.net


SOURCEFORGE.NET®


Atlassian
bitbucket


launchpad

1,000,000+ projects

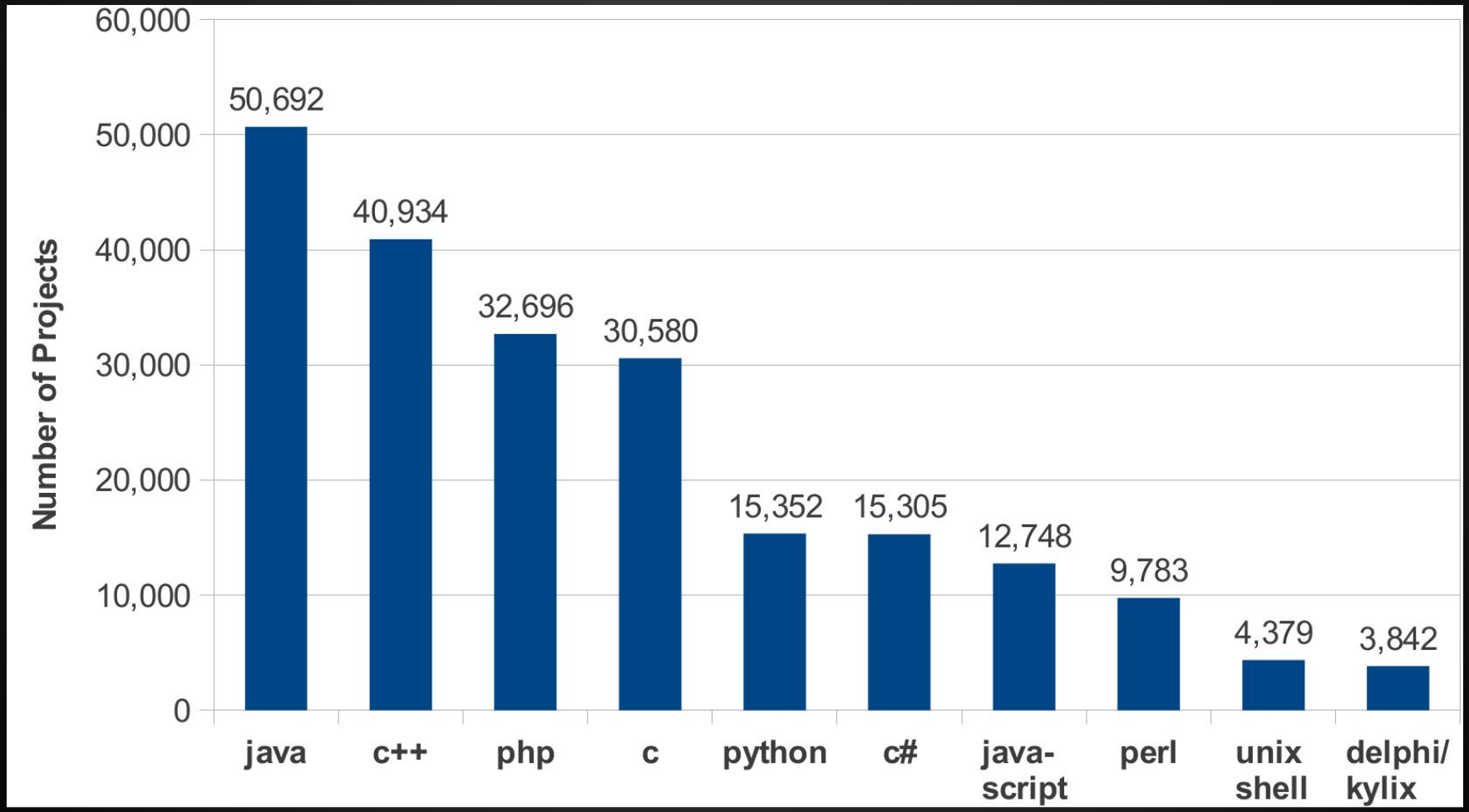
1,000,000,000+ lines of code

10,000,000+ revisions

3,000,000+ issue reports

1,000,000+ projects

What is the most used PL



1,000,000,000+ lines of code

How many methods are named "test"?

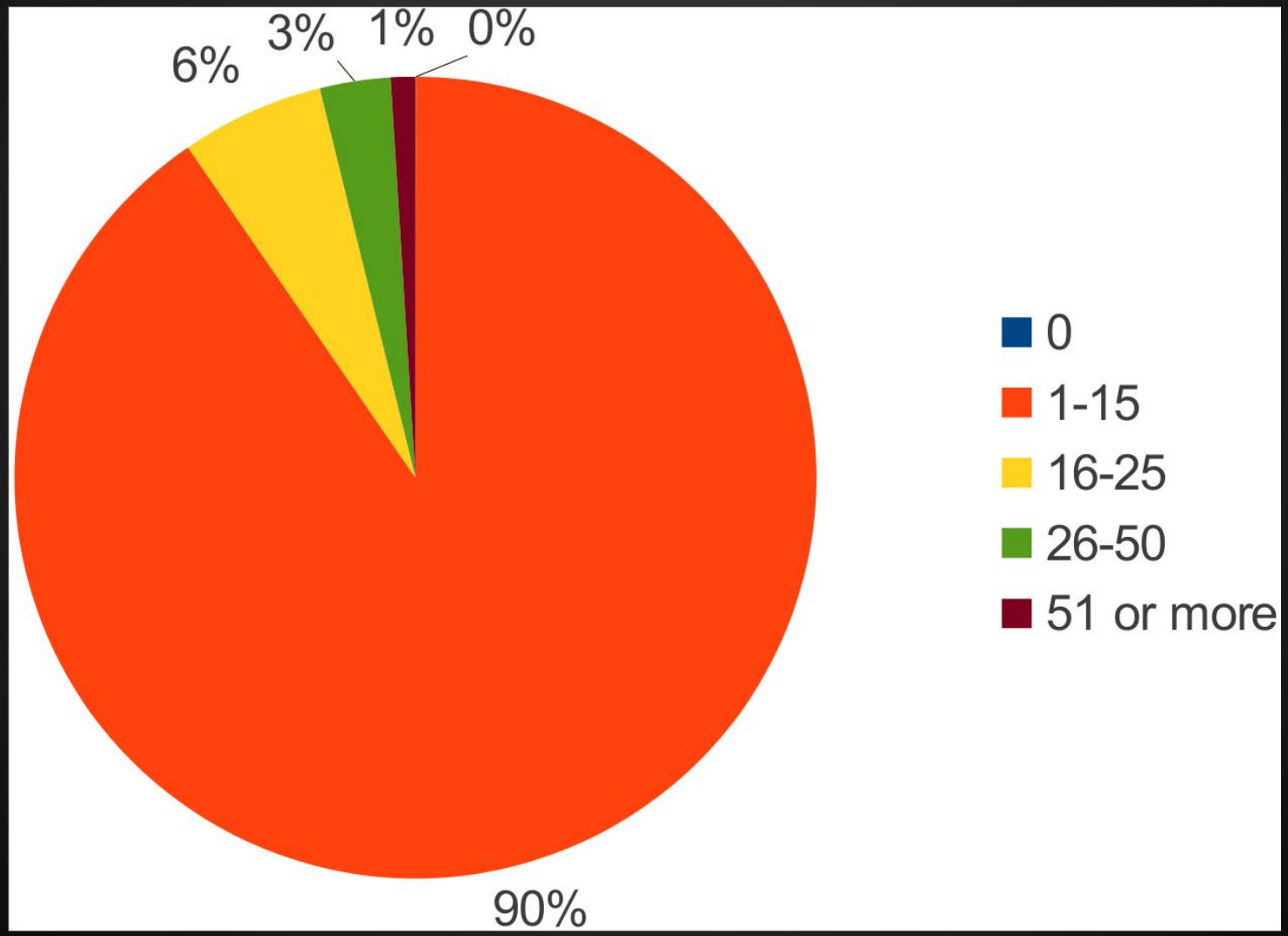
32,203

How many methods use JUnit's @Test annotation?

870,181
in **4,578** projects

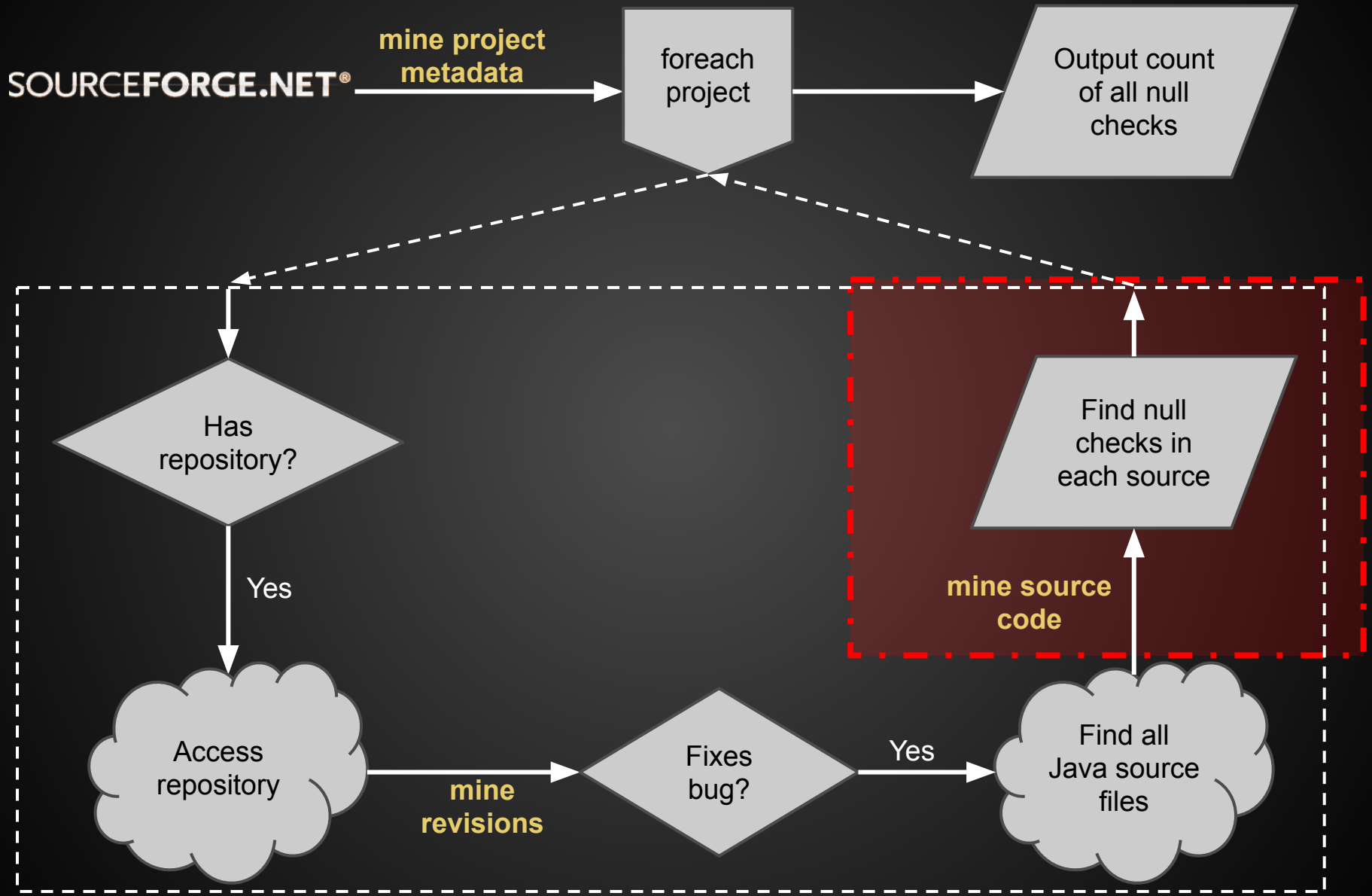
10,000,000+ revisions

How many words are in log messages?



Running example task

"*How many bug fixes add checks for null?*"


SOURCEFORGE.NET®
mine project metadata
foreach project
Output count of all null checks
Has repository?
Yes
Find null checks in each source
mine source code
Access repository
mine revisions
Fixes bug?
Yes
Find all Java source files

A solution in Java...

```
class AddNullCheck {
    static void main(String[] args) {
        ... /* create and submit a Hadoop job */
    }
    static class AddNullCheckMapper extends Mapper<Text, BytesWritable, Text, LongWritable> {
        static class DefaultVisitor {
            ... /* define default tree traversal */
        }
        void map(Text key, BytesWritable value, Context context) {
            final Project p = ... /* read from input */
            new DefaultVisitor() {
                boolean preVisit(Expression e) {
                    if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
                        for (Expression exp : e.expressions)
                            if (exp.kind == ExpressionKind.LITERAL && exp.literal.equals("null")) {
                                context.write(new Text("count"), new LongWritable(1));
                                break;
                            }
                    }
                }.visit(p);
            }
        }
        static class AddNullCheckReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
            void reduce(Text key, Iterable<LongWritable> vals, Context context) {
                int sum = 0;
                for (LongWritable value : vals)
                    sum += value.get();
                context.write(key, new LongWritable(sum));
            }
        }
```

Too much code!
Do not read!

Full program
over 140 lines of code

Uses ***JSON, SVN, and Eclipse JDT libraries***

Uses ***Hadoop framework***

Explicit/manual
parallelization

The Boa language and data-intensive infrastructure

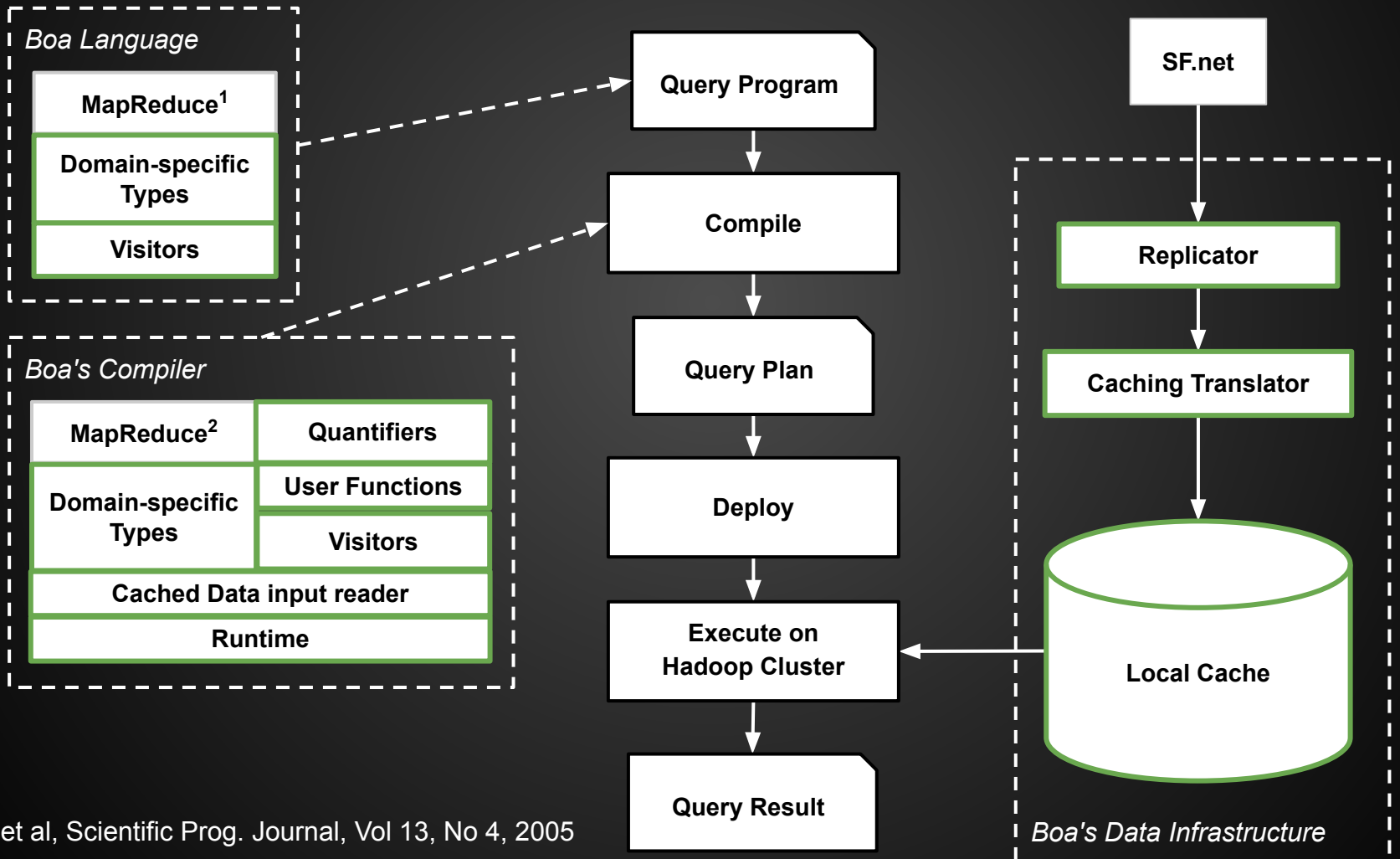
http://boa.cs.iastate.edu/

[ICSE'13]

Challenges and Design goals

- Easy to use
- Scalable and efficient
- Reproducible research results

Boa architecture



[1] Pike et al, Scientific Prog. Journal, Vol 13, No 4, 2005

[2] Anthony Urso, http://github.com/anthonyu/Sizzle

Recall: A solution in Java...

```
class AddNullCheck {
    static void main(String[] args) {
        ... /* create and submit a Hadoop job */
    }
    static class AddNullCheckMapper extends Mapper<Text, BytesWritable, Text, LongWritable> {
        static class DefaultVisitor {
            ... /* define default tree traversal */
        }
        void map(Text key, BytesWritable value, Context context) {
            final Project p = ... /* read from input */
            new DefaultVisitor() {
                boolean preVisit(Expression e) {
                    if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
                        for (Expression exp : e.expressions)
                            if (exp.kind == ExpressionKind.LITERAL && exp.literal.equals("null")) {
                                context.write(new Text("count"), new LongWritable(1));
                                break;
                            }
                    return true;
                }
            }.visit(p);
        }
    }
    static class AddNullCheckReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
        void reduce(Text key, Iterable<LongWritable> vals, Context context) {
            int sum = 0;
            for (LongWritable value : vals)
                sum += value.get();
            context.write(key, new LongWritable(sum));
        }
    }
}
```

Too much code!
Do not read!

Full program
over 140 lines of code

Uses ***JSON, SVN, and Eclipse JDT libraries***

Uses ***Hadoop framework***

Explicit/manual
parallelization

A better solution...

```
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
                count << 1;
});
```

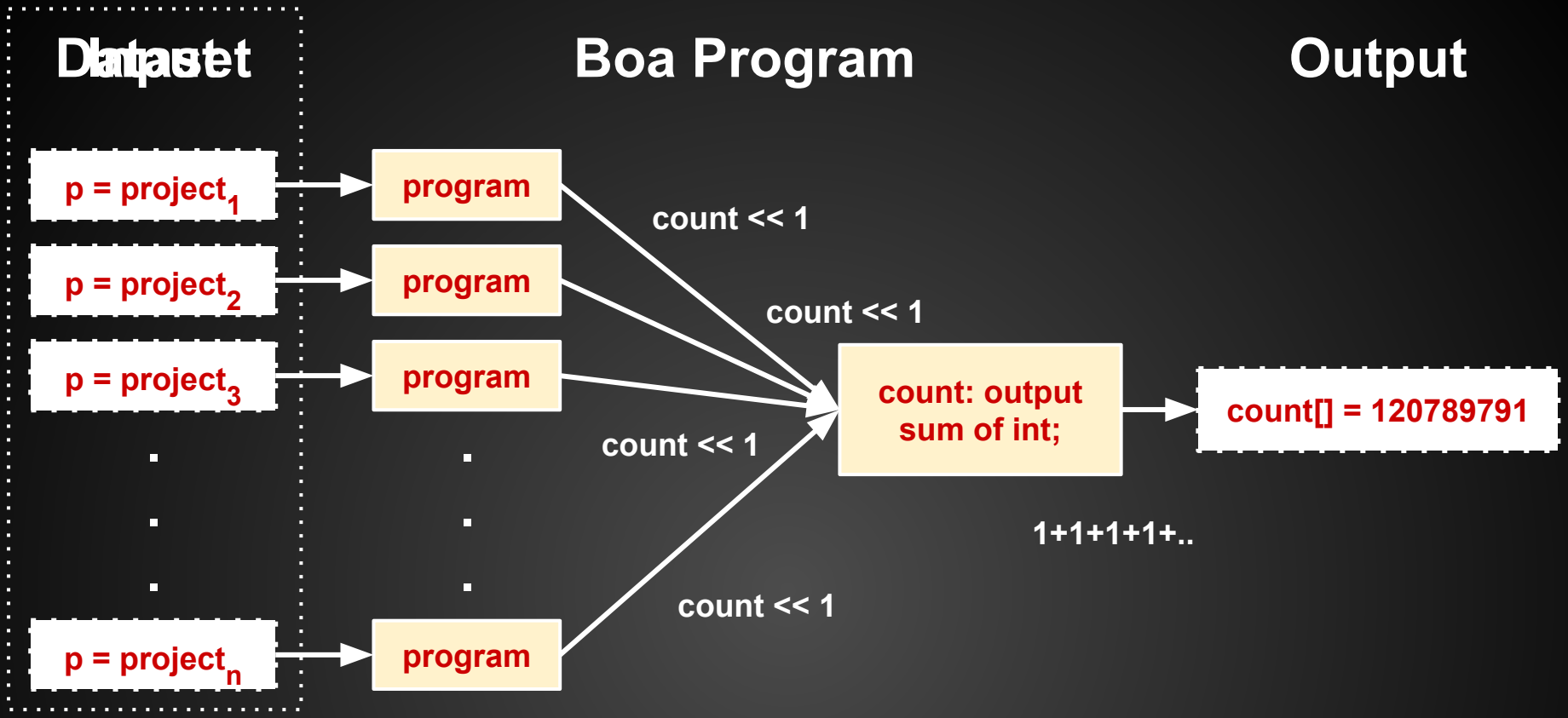
Full program **8 lines of code**!

Automatically parallelized!

No external libraries needed!

Analyzes **28.8 million** source files in about **15 minutes**!

(only 32 ***micro***seconds each!)




```
p: Project = input;
count: output sum of int;

visit(p, visitor {
     before e: Expression ->
          if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
               exists (i: int; isliteral(e.expressions[i], "null"))
                    count << 1;
});
```

Challenges and Design goals

Easy to use

Scalable and efficient

Reproducible research results

Let's see it in action!

http://boa.cs.iastate.edu/boa/

Why are we waiting for results?

Program is analyzing...

699,331 projects

494,158 repositories

15,063,073 revisions

69,863,970 files

18,651,043,238 AST nodes

Let's check the results!

<<demo>>

Domain-specific types

http://boa.cs.iastate.edu/docs/dsl-types.php

```
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null")))
                count << 1;
});
```

Abstracts details of *how* to mine software repositories

User-defined functions

http://boa.cs.iastate.edu/docs/user-functions.php

```
id := function (a1: t1, ..., an: tn) [: ret] {
    ... # body
    [return ...;]
};
```

Return type is optional

- Allows for complex algorithms and code re-use
- Users can provide their own mining algorithms

Quantifiers

http://boa.cs.iastate.edu/docs/quantifiers.php

```
foreach (i: int; condition...) body;
```

For each value of **i** where **condition** holds, run **body**

Quantifiers

http://boa.cs.iastate.edu/docs/quantifiers.php

```
foreach (i: int; condition...) body;
```

For each value of i where condition holds, run body

```
exists (i: int; condition...) body;
```

If there **exists** a value of **i** where **condition** holds, run **body**

Quantifiers

http://boa.cs.iastate.edu/docs/quantifiers.php

```
foreach (i: int; condition...) body;
```

For each value of i where condition holds, run body

```
exists (i: int; condition...) body;
```

If there exists a value of i where condition holds, run body

```
ifall (i: int; condition...) body;
```

If for **all** values of **i** **condition** holds, run **body**

Output and aggregation

http://boa.cs.iastate.edu/docs/aggregators.php

```
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
                count << 1;
});
```

- Output defined in terms of predefined data aggregators
  - sum, set, mean, maximum, minimum, etc
- Values sent to output aggregation variables

What about source code?

```
p: Project = input;
count: output sum of int;

visit(p, visitor {
    before e: Expression ->
        if (e.kind == ExpressionKind.EQ || e.kind == ExpressionKind.NEQ)
            exists (i: int; isliteral(e.expressions[i], "null"))
                count << 1;
});
```

Declarative Visitors in Boa

[GPCE'13]

Basic Syntax

```
id := visitor {
   before id:T -> statement
   after  id:T -> statement
   ...
};
visit(startNode, id);
```

Execute **`statement`** either **`before`** or **`after`** visiting the children of a node of type **`T`**

Type Lists and Wildcards

```
visitor {
   before id:T     -> statement
   after T2,T3,T4  -> statement
   after _         -> statement
}
```

Matching single type (with identifier)

Attributes of the node available via identifier

Type Lists and Wildcards

```
visitor {
   before id:T      -> statement
   after T2,T3,T4   -> statement
   after _          -> statement
}
```

Type list (no identifier)

Executes **`statement`** when visiting nodes of type **`T2`**, **`T3`**, or **`T4`**

Type Lists and Wildcards

```
visitor {
   before id:T     -> statement
   after T2,T3,T4  -> statement
   after _         -> statement
}
```

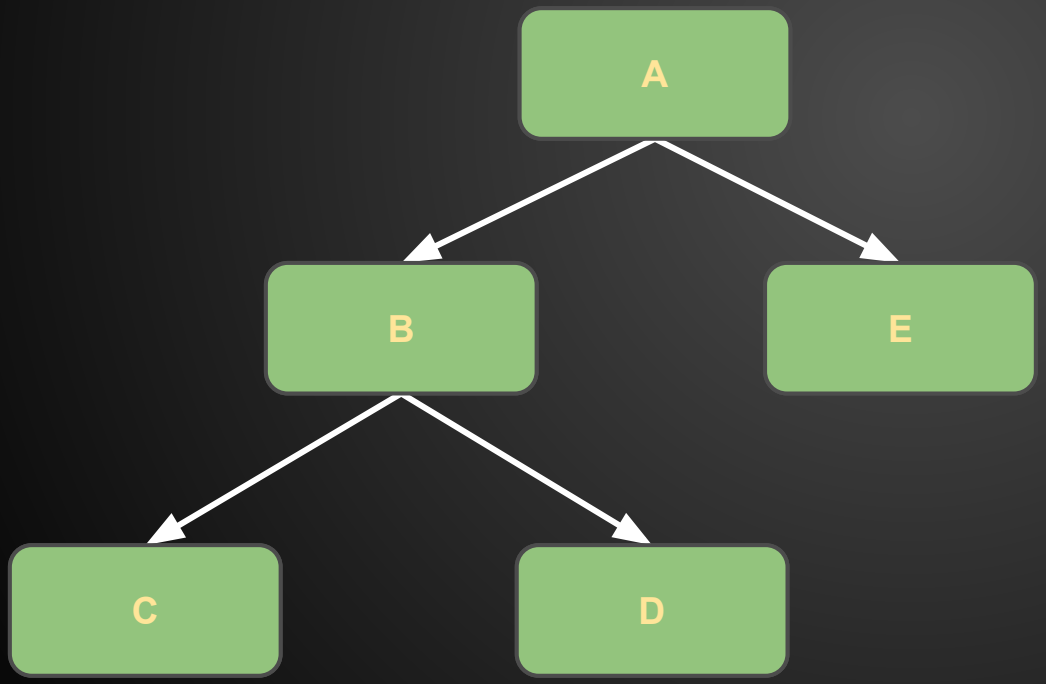
Wildcard (no identifier)

Executes **`statement`** for any node not already listed in another similar clause (e.g., T but not T2/T3/T4)

Provides ***default*** behavior

Custom Traversals

A -> E -> B -> C -> D




```
before n: A -> {
   visit(n.E);
   visit(n.B);
   stop;
}
```

That’s the language…

what can we do with it?

Expressiveness

Treasure study reproduction [Grechanik10]

⇒ 22 tasks

[GPCE'13]

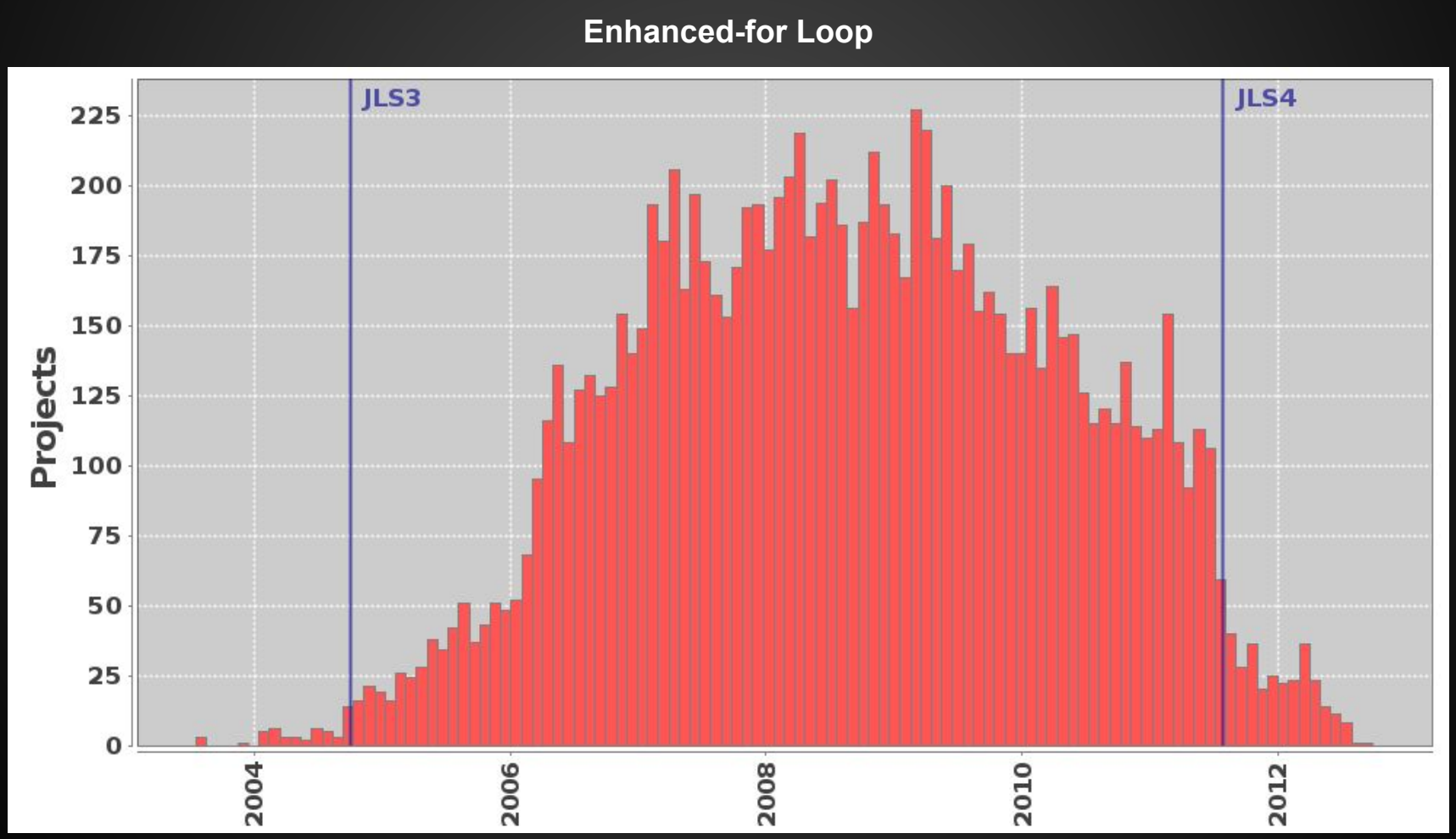
Java language feature adoption

⇒ 18 tasks

[in submission ICSE'14]

Several additional tasks (on Boa website)

How do projects adopt features?



Most features see low use

Research question: are there missed opportunities to use language features?

e.g., Underscore literals

- 22M occurrences before feature release
- 2M occurrences after feature release

Is old code refactored to use new features?

Yes!

e.g., diamond pattern (JDK7)

- **8.5k refactorings detected**
- 3.8k files
- 72 projects

Challenges and Design goals

Easy to use

Scalable and efficient

Reproducible research results

Source Code Comprehension [1/3]

- Controlled Experiment
  - Subjects shown 5 source code mining tasks in Boa
  - Asked to describe (in own words) each task
  - Same tasks shown again (random order)
    - Multiple choice this time
  - Experiment repeated 6 months later in Hadoop
    - Same tasks
    - Same wording for multiple choice answers

Source Code Comprehension [3/3]

Boa Programs				
Q1	Q2	Q3	Q4	Q5
N	Y	Y	Y	Y
-Y	Y	Y	Y	Y
?	Y	Y	Y	Y
-Y	Y	Y	Y	Y
?	+N	Y	Y	N
N	Y	Y	Y	-Y
N	-Y	Y	Y	Y
N	+N	-Y	-Y	Y

Hadoop Programs				
Q1	Q2	Q3	Q4	Q5
-Y	-Y	N	-Y	-Y
?	-Y	-Y	-Y	N
-Y	Y	+N	Y	-Y
N	Y	N	-Y	N
N	-Y	N	N	N
-Y	Y	Y	Y	Y
N	N	Y	-Y	-Y
-Y	+N	Y	N	Y

Source Code Comprehension [3/3]

Grading: Use Multiple Choice

Boa Programs					
Q1	Q2	Q3	Q4	Q5	Total
N	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	100%
?	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	100%
?	+N	Y	Y	N	40%
N	Y	Y	Y	-Y	80%
N	-Y	Y	Y	Y	80%
N	+N	-Y	-Y	Y	60%

77.5%

Hadoop Programs					
Q1	Q2	Q3	Q4	Q5	Total
-Y	-Y	N	-Y	-Y	80%
?	-Y	-Y	-Y	N	60%
-Y	Y	+N	Y	-Y	80%
N	Y	Y	Y	N	40%
N	-Y	N	N	N	20%
-Y	Y	Y	Y	Y	100%
N	N	Y	-Y	-Y	60%
-Y	+N	Y	N	Y	60%

62.5%

Source Code Comprehension [3/3]

Grading: Use Free-form

Boa Programs

Q1	Q2	Q3	Q4	Q5	Total
N	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	80%
?	Y	Y	Y	Y	80%
-Y	Y	Y	Y	Y	80%
?	+N	Y	Y	N	60%
N	Y	Y	Y	-Y	60%
N	-Y	Y	Y	Y	60%
N	+N	-Y	-Y	Y	40%

67.5%

Hadoop Programs

Q1	Q2	Q3	Q4	Q5	Total
-Y	-Y	N	-Y	-Y	0%
?	-Y	-Y	-Y	N	0%
-Y	Y	+N	Y	-Y	60%
N	Y	N	-Y	N	20%
N	-Y	N	N	N	0%
-Y	Y	Y	Y	Y	80%
N	N	Y	-Y	-Y	20%
-Y	+N	Y	N	Y	60%

30%

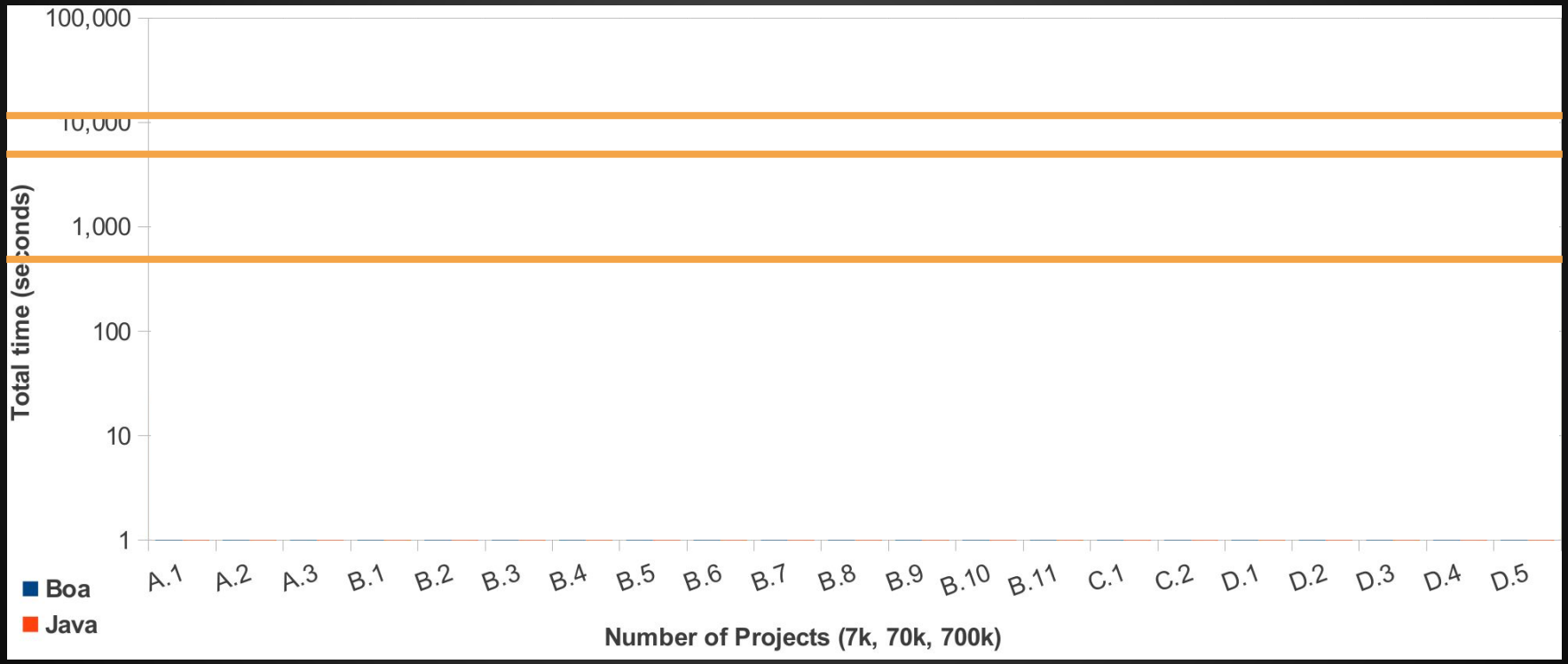
Challenges and Design goals

Easy to use

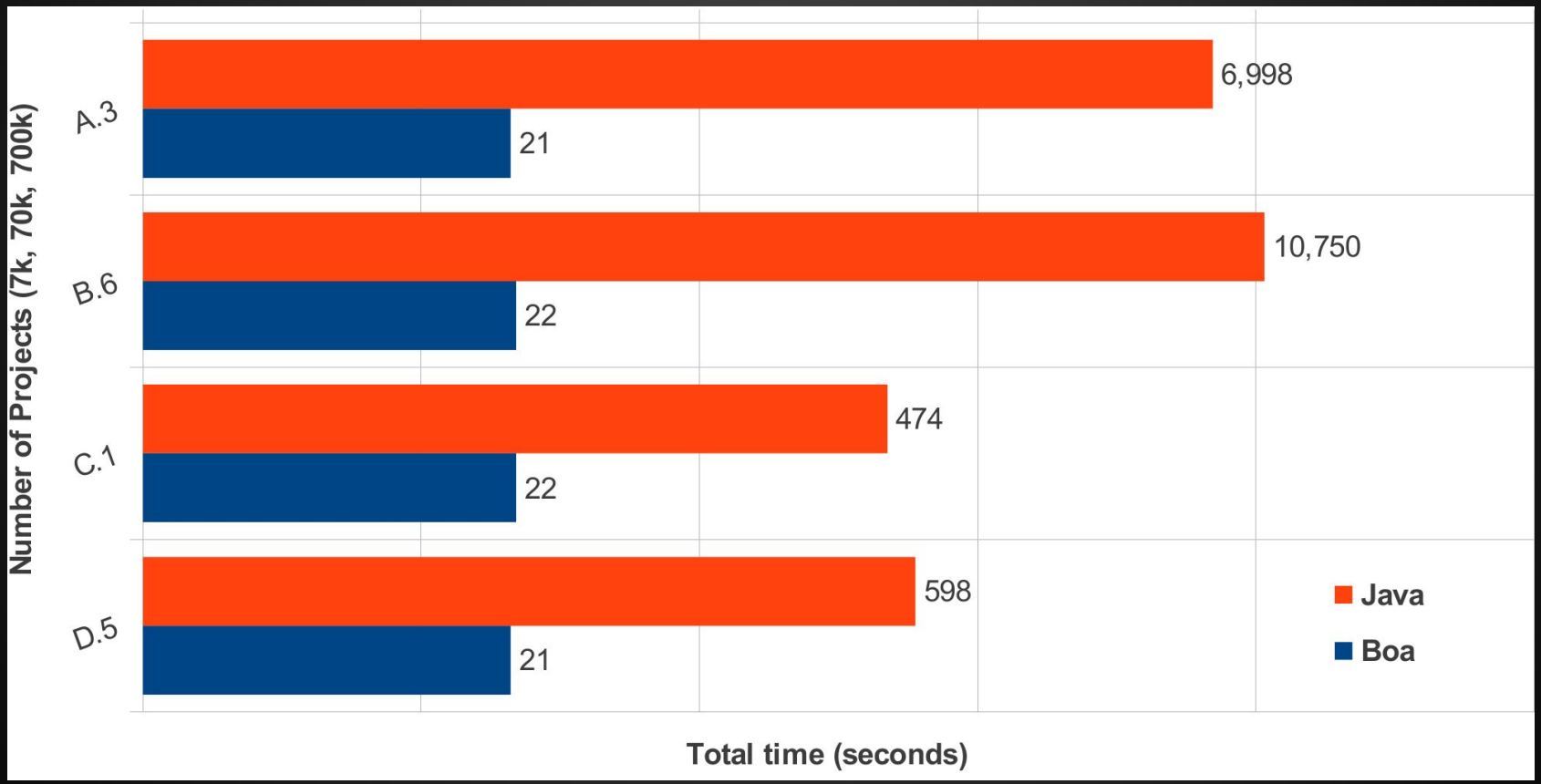
Scalable and efficient

Reproducible research results

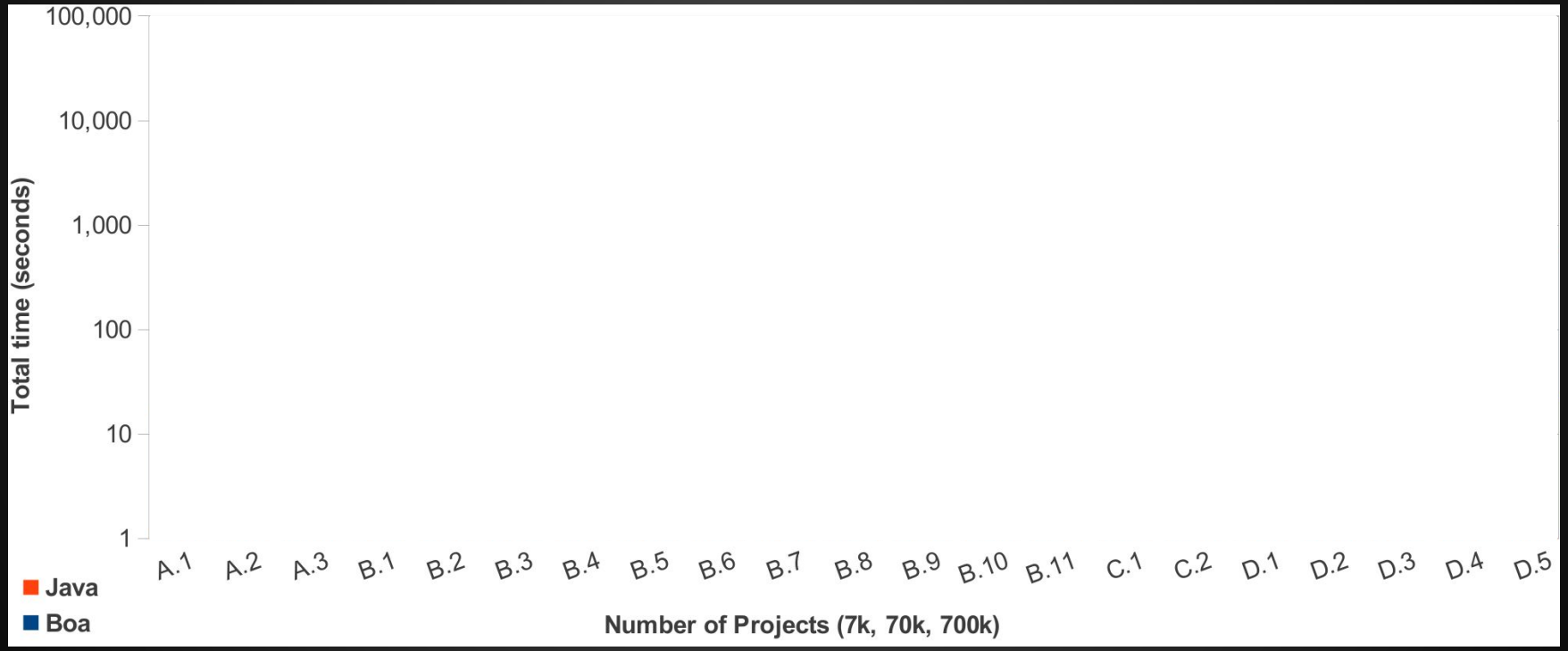
Efficient execution



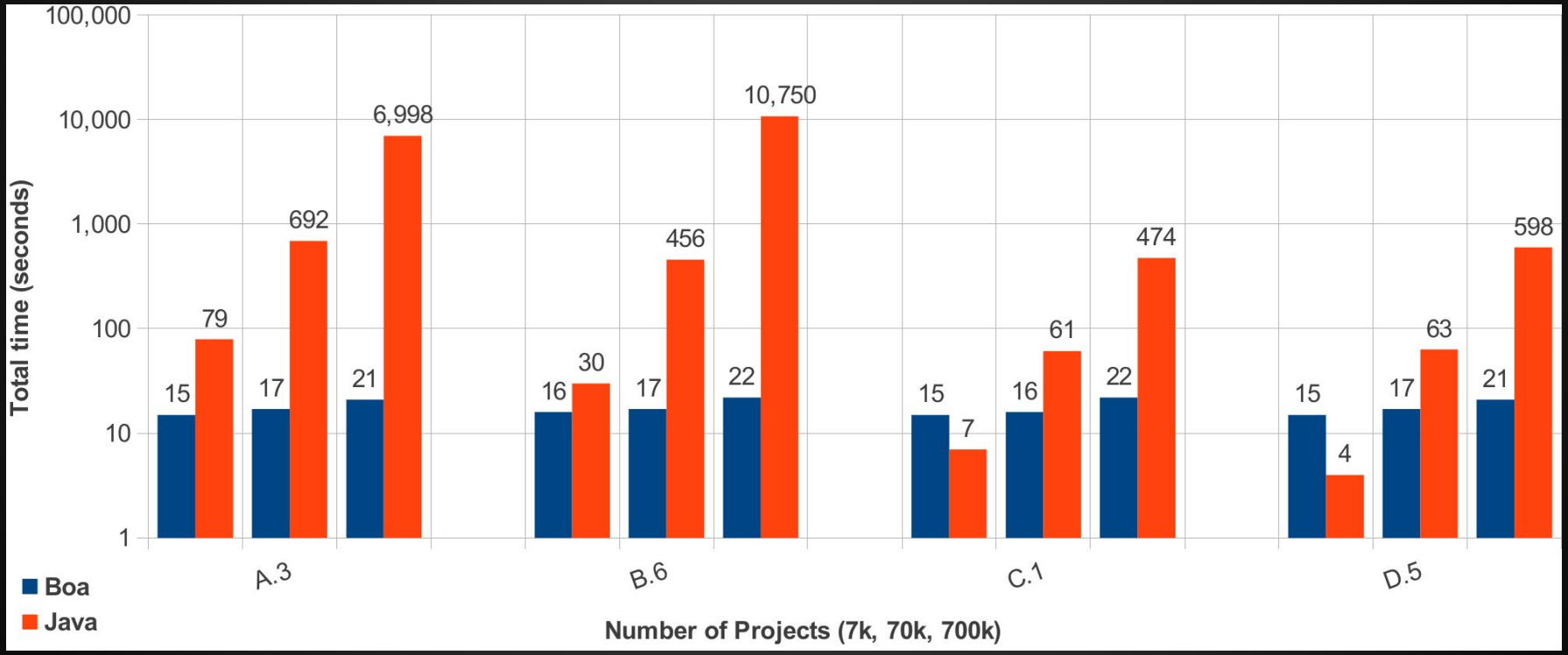
Efficient execution



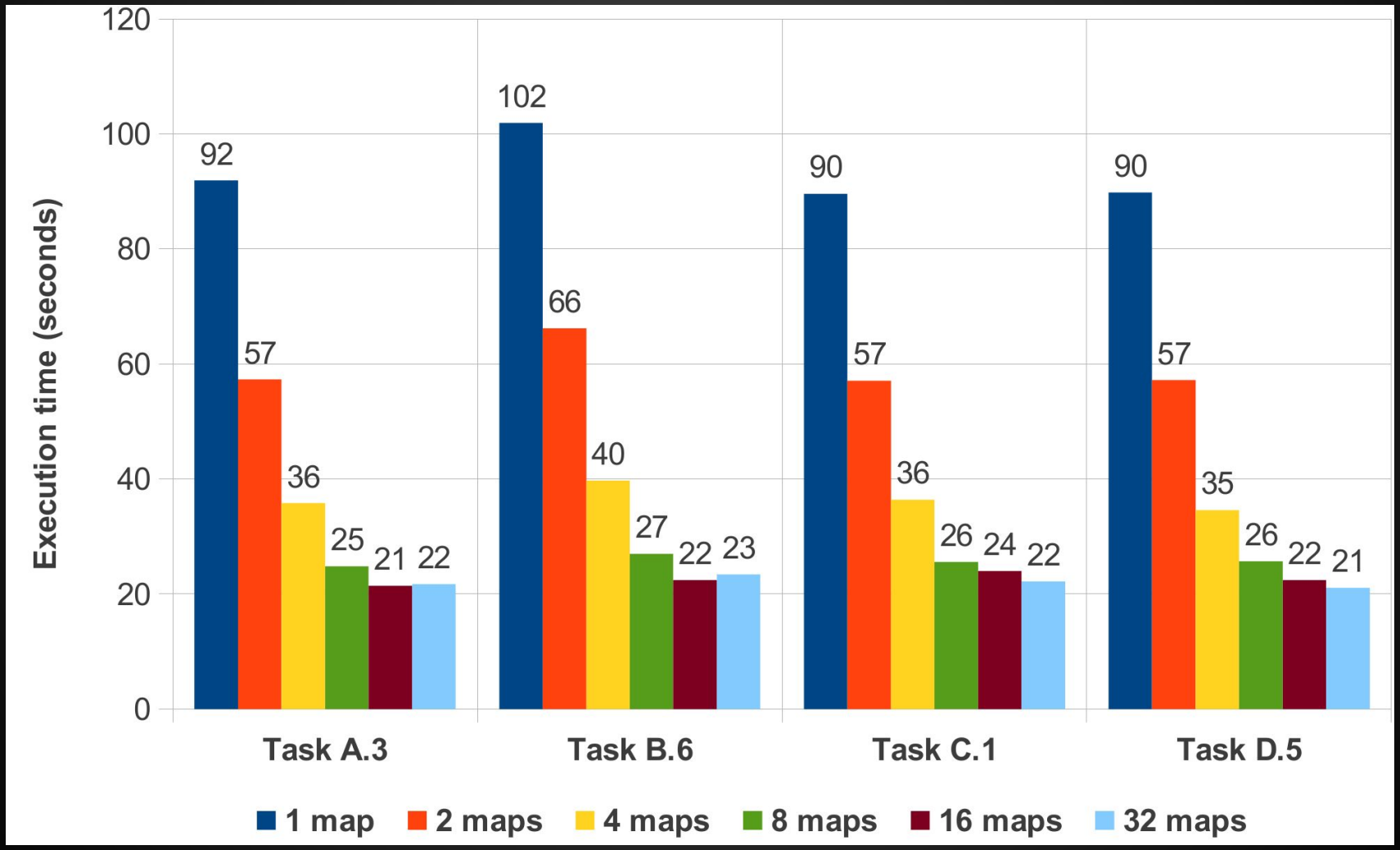
Scalability of input size



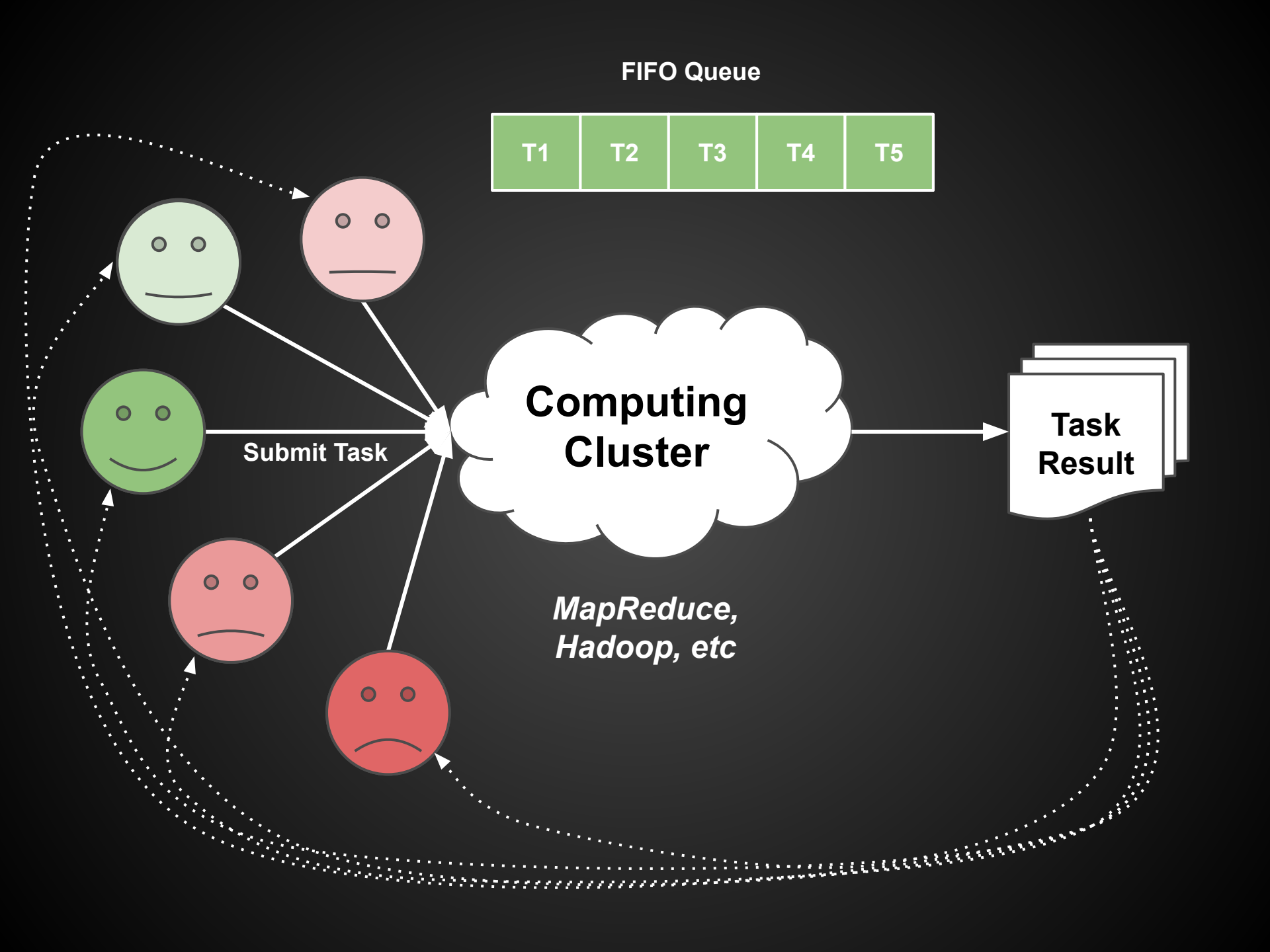
Scalability of input size

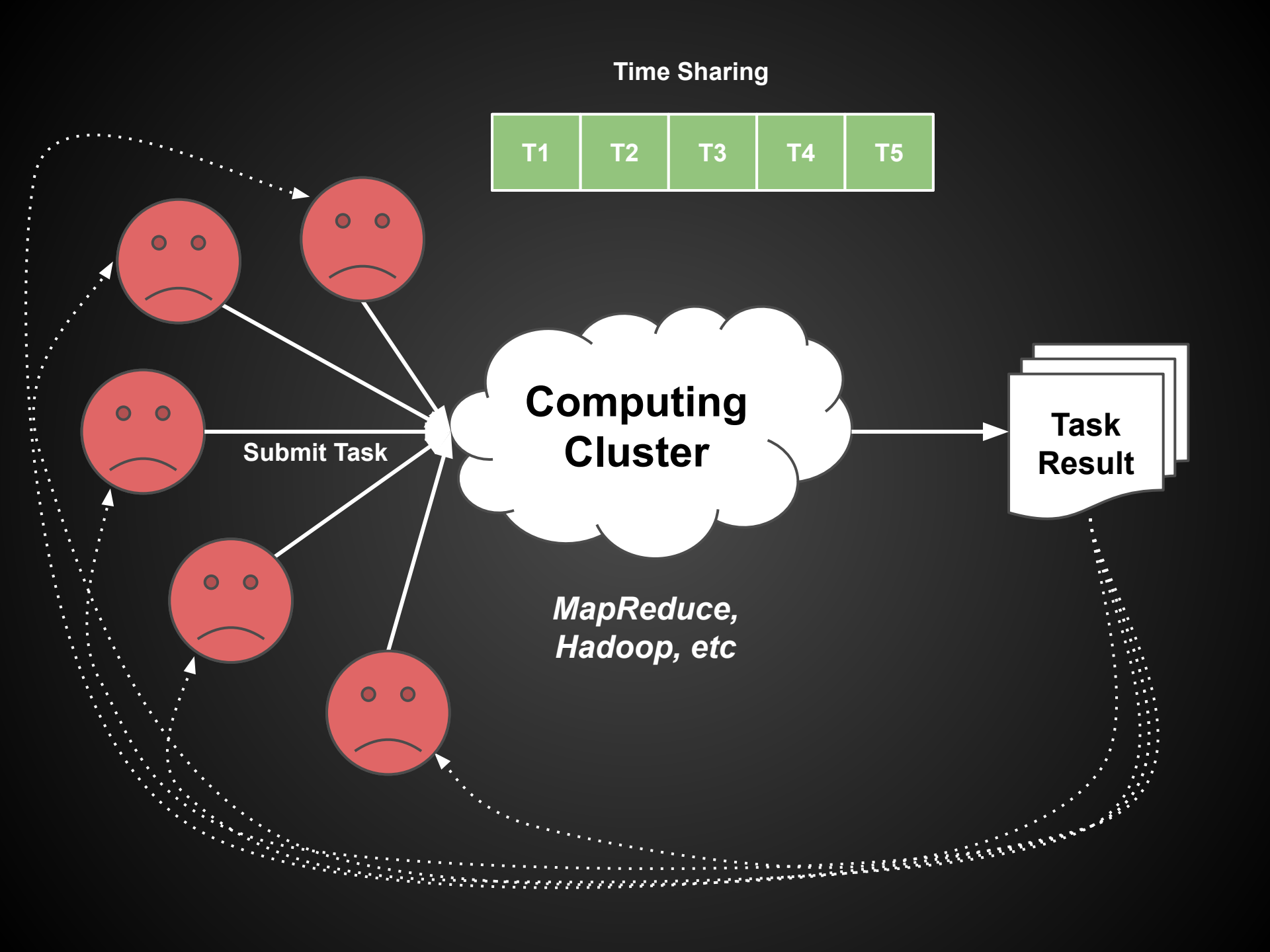


Scales to more cores



Optimizations


FIFO Queue
T1
T2
T3
T4
T5
Submit Task
Computing Cluster
Task Result
MapReduce,
Hadoop, etc


Time Sharing
T1
T2
T3
T4
T5
Submit Task
Computing Cluster
Task Result
MapReduce,
Hadoop, etc

Solutions?

- Scale the hardware
  - Expensive
  - Not always feasible (small businesses, MOOCs, researchers, etc)
- Optimize the software
  - Optimize individual tasks
    - standard program optimizations
    - chain folding [MinerShook12], sibling/MSCR fusion [Chambers10]
  - Optimize multiple tasks
    - manual job merging [MinerShook12]

[Chambers10] Craig Chambers et al., “FlumeJava”, PLDI 2010
[MinerShook12] Donald Miner and Adam Shook, “MapReduce Design Patterns”, O’Reilly, 2012

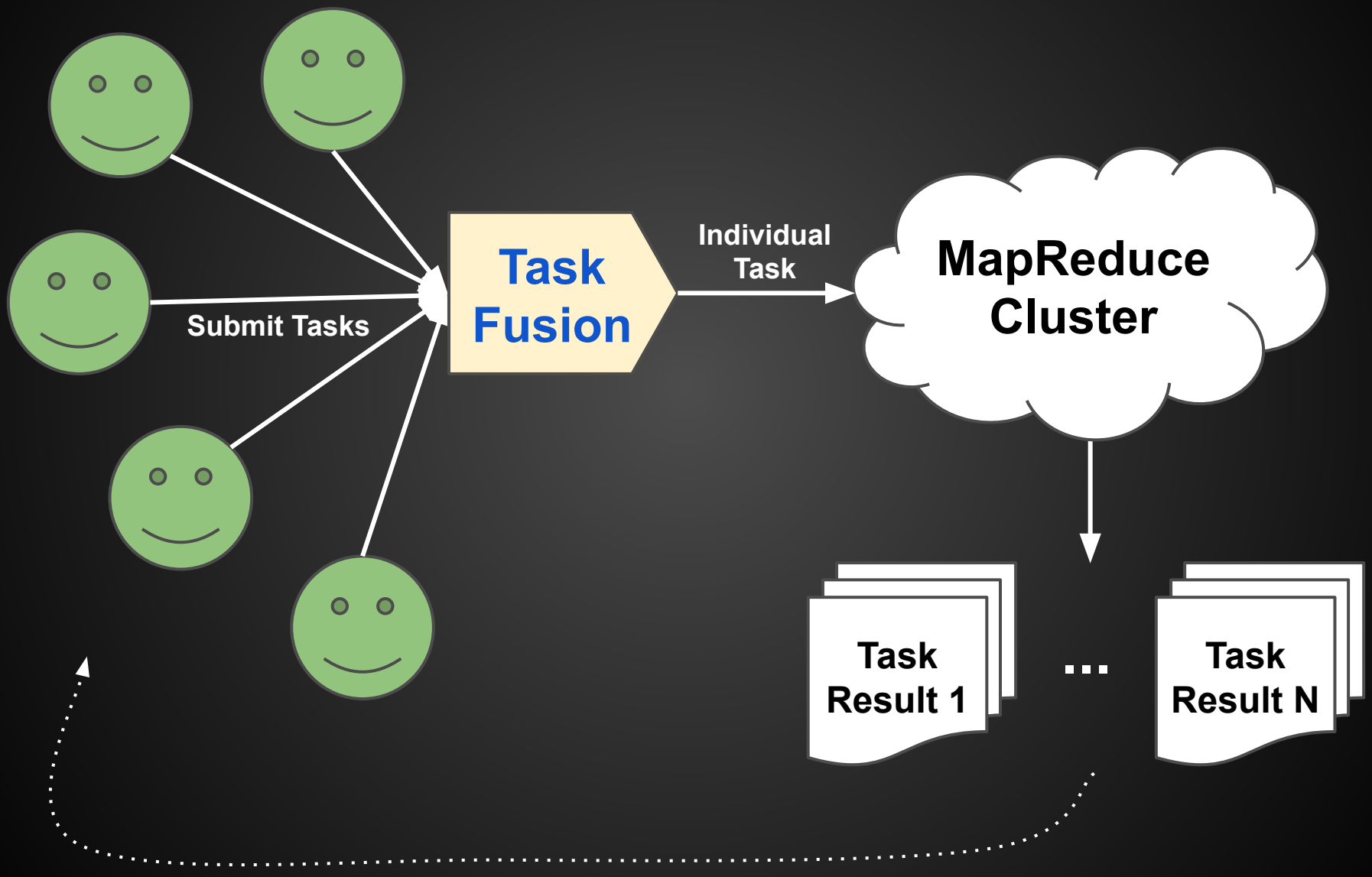
Research Questions

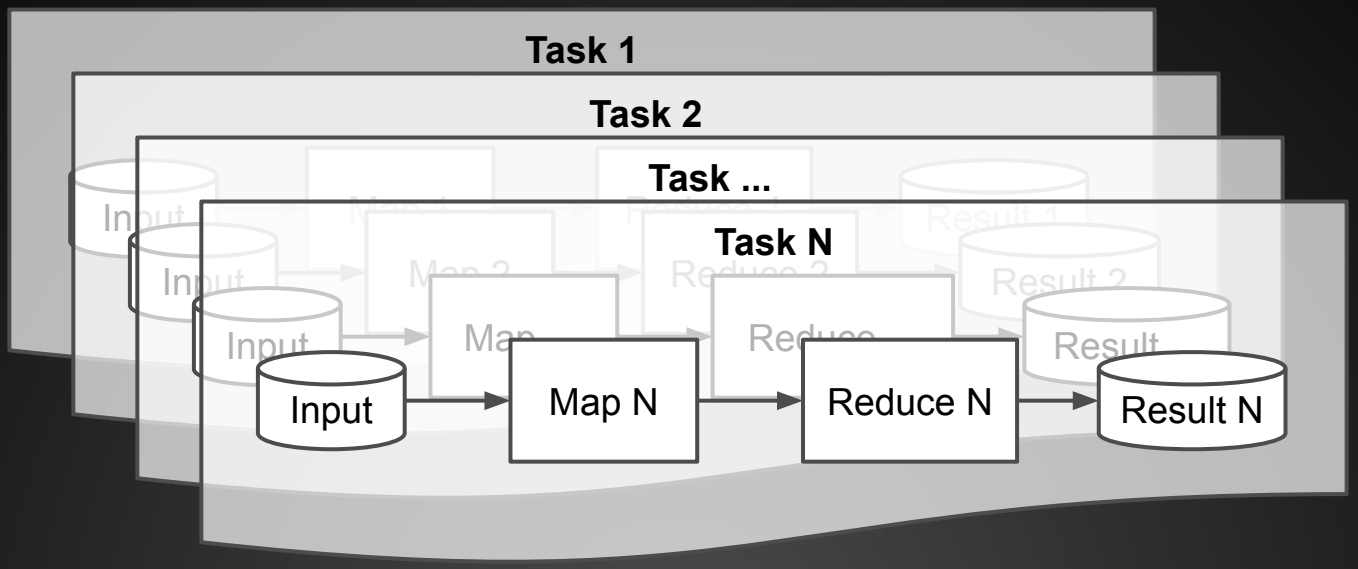
1. Can we **automatically** merge related tasks from **different users**?

   Answer: **Task Fusion**

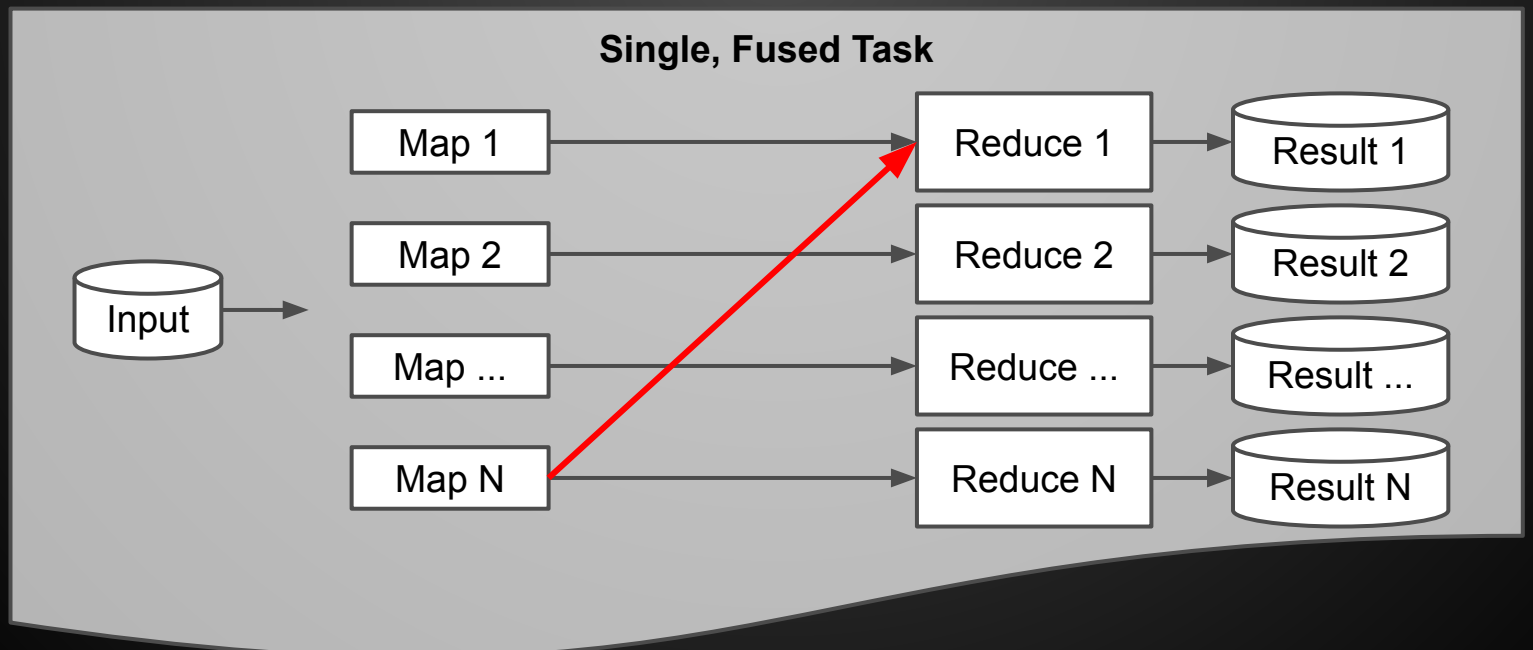
   [SPLASH'13 SRC]

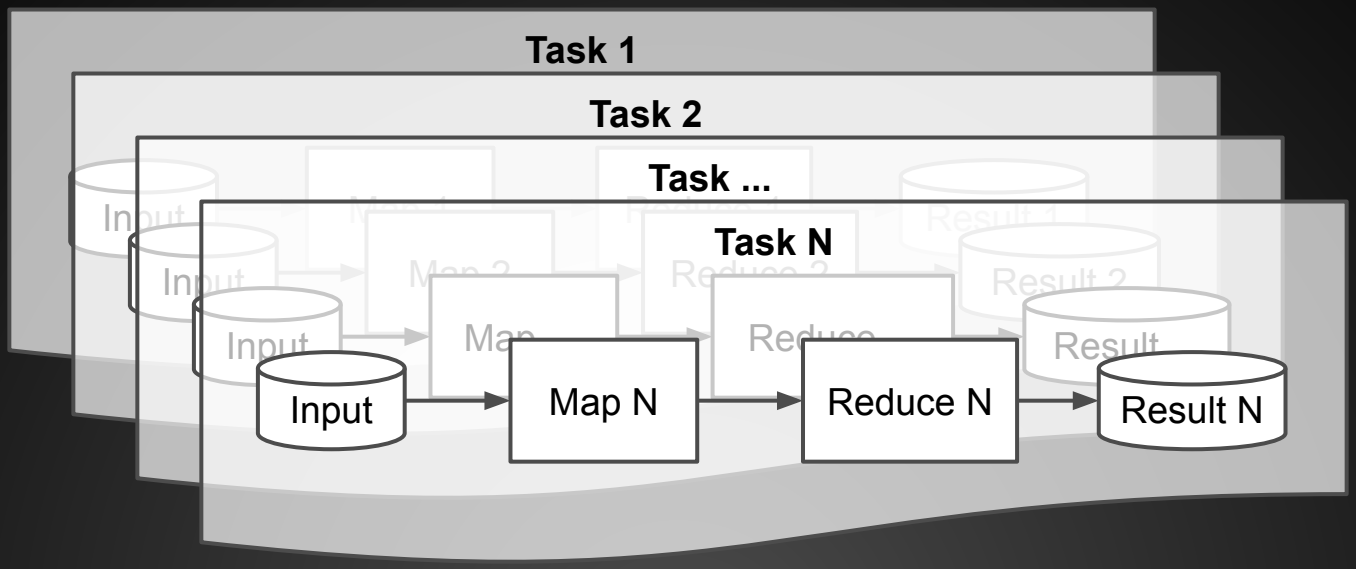
2. Does *Task Fusion* decrease user wait times in shared computing clusters?


Submit Tasks
Task Fusion
Individual Task
MapReduce Cluster
Task Result 1
...
Task Result N



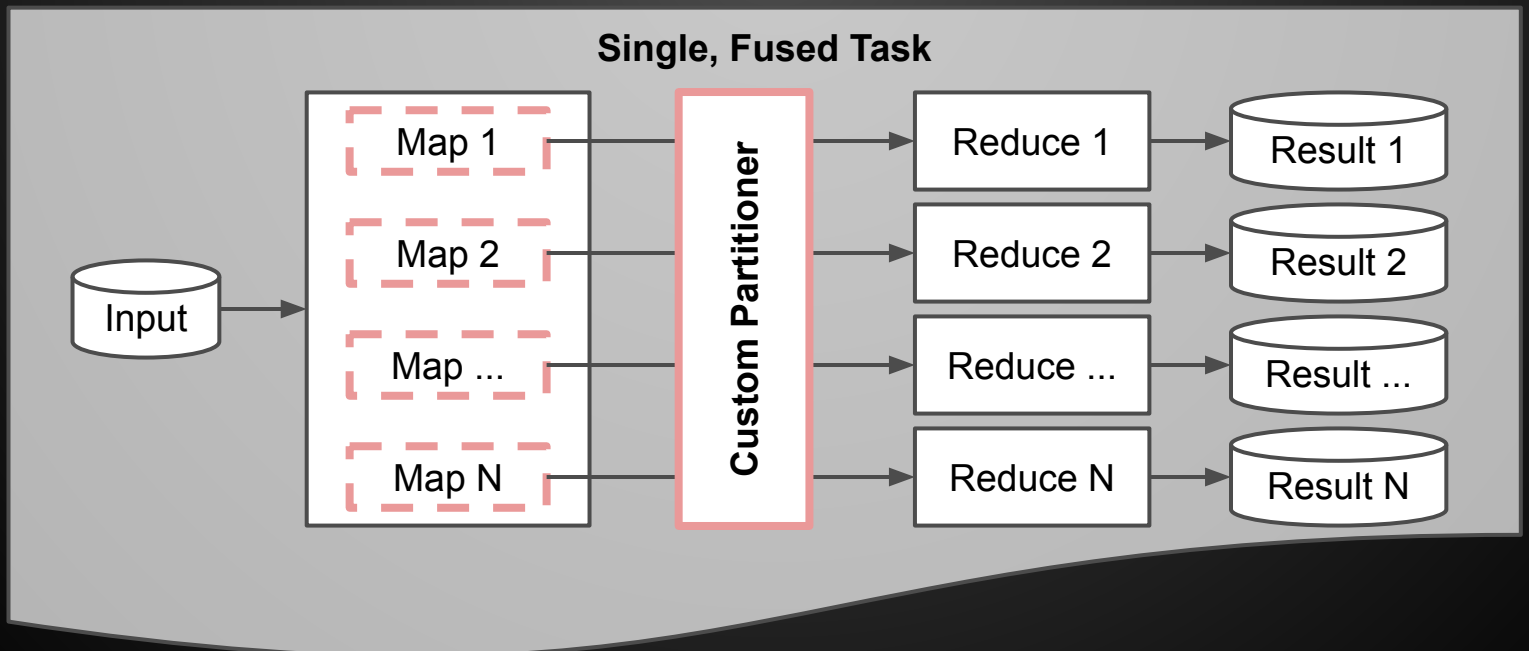
Technical Challenge:
map output == side effect





Solution: **modify maps to output composite keys**

Custom partitioner ensures proper routing

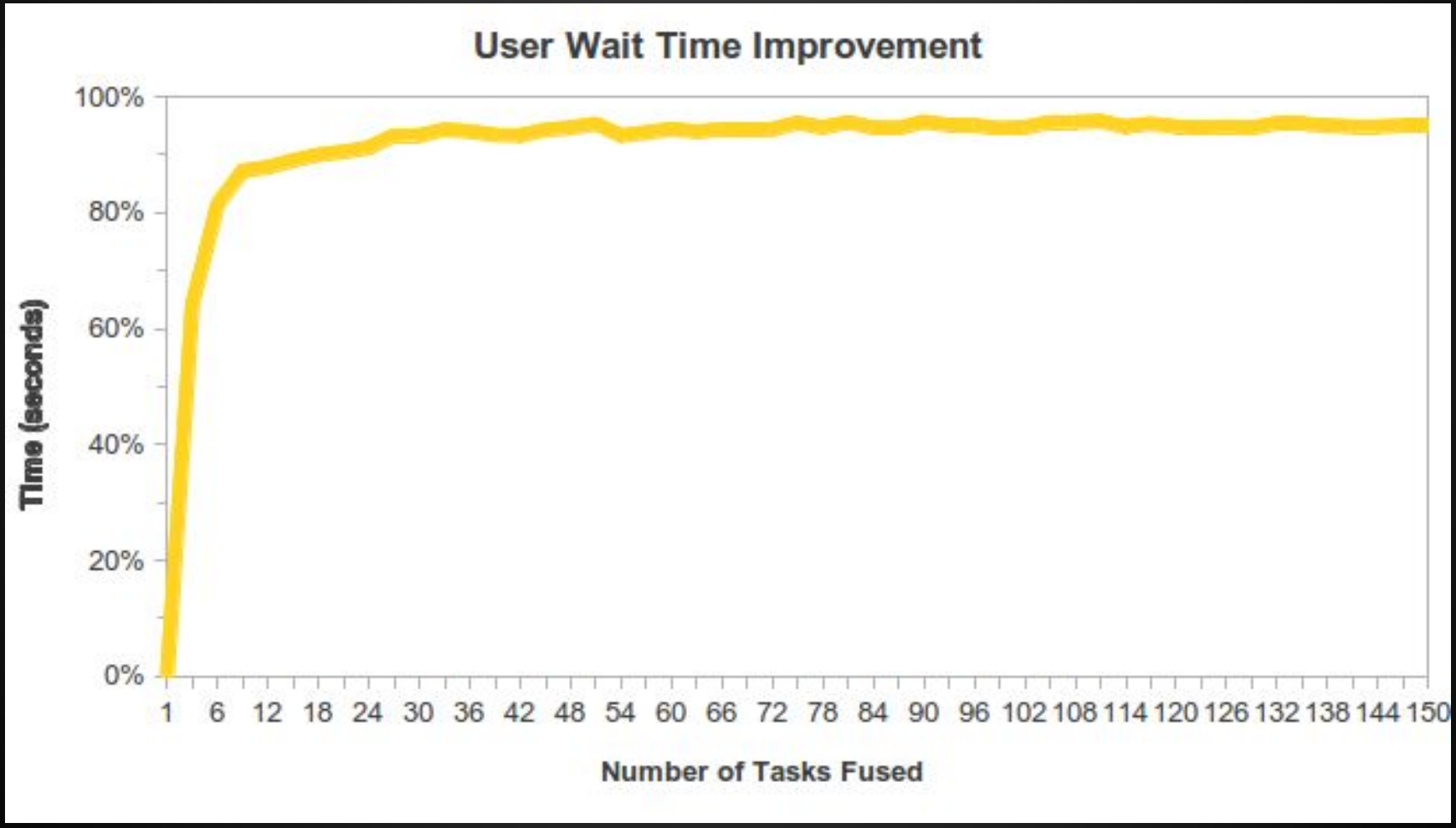


Results

Task Size	# of Tasks	Times		Speedup
		No Task Fusion	Task Fusion	
Small[1]	21	8.1m	0.8m	**10.8X**
Medium[2]	22	2.3h	1.8h	**1.3X**
Large[2]	18	4.6h	3.9h	**1.2X**
Mixed[3]	9	1.3h	0.9h	**1.4X**

[1] queries on project and revision metadata only
[2] queries on metadata and millions of source files
[3] 3 small, 3 medium, 3 large

Results


User Wait Time Improvement
100%
80%
60%
40%
20%
0%
Time (seconds)
1 6 12 18 24 30 36 42 48 54 60 66 72 78 84 90 96 102 108 114 120 126 132 138 144 150
Number of Tasks Fused

Can we do better?

Yes! ⇒ Visitor fusion

[to submit PLDI'14]

```
visit(p, visitor {
    before T1 -> s1;
    before T2 -> s3;
});
```

```
visit(p, visitor {
    before T1 -> s2;
    after T3 -> s4;
});
```

```
visit(p, visitor {
    before T1 -> {
        s1;
        s2;
    }
    before T2 -> s3;
    after T3 -> s4;
});
```

Results

Task Size	# of Tasks	Times		Speedup
		Task Fusion	Visitor Fusion	
Medium	22	1.8h	1.8h	**0X**
Large	18	3.9h	0.5h	**7.4X**
Mixed	9	0.9h	0.6h	**1.5X**

Combined Results

Task Size	# of Tasks	Times		Speedup
		No Fusion	Task+Visitor Fusion	
Small[1]	21	8.1m	0.8m	**10.8X**
Medium[2]	22	2.3h	1.8h	**1.3X**
Large[2]	18	4.6h	0.5h	**9.2X**
Mixed[3]	9	1.3h	0.6h	**2.2X**

[1] queries on project and revision metadata only
[2] queries on metadata and millions of source files
[3] 3 small, 3 medium, 3 large

Challenges and Design goals

Easy to use

Scalable and efficient

Reproducible research results

Reproducing MSR results

Replicating MSR:
A study of the potential replicability of papers published in the Mining Software Repositories Proceedings

Gregorio Robles
GSyC/LibreSoft
Universidad Rey Juan Carlos
Madrid, Spain
Email: grex@gsyc.urjc.es

***Abstract*—This paper is the result of reviewing all papers published in the proceedings of the former International Workshop on Mining Software Repositories (MSR) (2004-2006) and now Working Conference on MSR (2007-2009). We have analyzed the papers that contained any experimental analysis of software projects for their potentiality of being replicated. In this regard, three main issues have been addressed: i) the public availability of the data used as case study, ii) the public availability of the processed dataset used by researchers and iii) the public availability of the tools and scripts. A total number of 171 papers have been analyzed from the six workshops/working conferences up to date. Results show that MSR authors use in general publicly available data sources, mainly from free software repositories, but that the amount of publicly available processed datasets is very low. Regarding tools and scripts, for a majority of papers we have not been able to find any tool, even for papers where the authors explicitly state that they have built one. Lessons learned from the experience of reviewing the whole MSR literature and some potential solutions to lower the barriers of replicability are finally presented and discussed.**

***Keywords*-replication, tools, public datasets, mining software repositories**

Replication package: http://gsyc.urjc.es/~grex/msr2010.

I. Introduction

Mining software repositories (MSR) has become a fundamental area of research for the Software Engineering community, and of vital importance in the case of empirical studies. Software repositories contain a large amount of valuable information that includes source control systems storing all the history of the source code, defect tracking systems that host defects, enhancements and other issues, and other communication means such as mailing lists or forums. As a result of the possibilities that mining software repositories offer, an annual workshop first, then working conference on this topic has been organized with an extraordinary success in participation and research output.

Being mainly focused on empirical research, we wanted to evaluate how much of the research presented at the MSR can be potentially replicated. Replication is a fundamental task in empirical sciences and one of the main threats to validity that empirical software engineering may suffer [1]. Among these threats, we may encounter: lack of independent validation of the presented results; changes in practices, tools or methodologies; or generalization of knowledge although a limited amount of case studies have been performed.

A simple taxonomy of replication studies provides us with two main groups: exact replications and conceptual replications. The former ones are those in "which the procedures of an experiment are followed as closely as possible to determine whether the same results can be obtained", while the latter ones are those "one in which the same research question or hypothesis is evaluated by using a different experimental procedure, i.e. many or all of the variables described above are changed." [2]. In this paper, we will target exact replications as the requirements that have to be met to perform an exact replication are more severe, and in general make a conceptual replication feasible.

We are focusing in this paper on potential replication as we have actually not replicated any of the studies presented in the papers under review. Our aim in this sense is more humble: we want to check if the necessary conditions that make a replication possible are met.

The rest of the paper is structured as follows: in the next section, the method used for this study is presented. Then some general remarks on the MSR conference are given, to give the reader a sense of the type of papers that are published in the MSR proceedings. Results will be presented in section IV: first, the replication-friendliness of the papers will be shown and then each of the individual characteristics that we have defined will be studied independently. MSR has a special track called the "Mining Challenge", a section is devoted to analyze it with the aim of finding if results differ from those for the rest of papers. Then, other non-quantitative facts from the review are enumerated. Section VII discusses the findings of the paper and hints at possible solutions. Then, conclusions are drawn. In a final section, the replicability of this paper is considered.

II. Method

The method that has been used to perform this study is a complete literature review of the papers published in

978-1-4244-6803-4/10/$26.00 © 2010 IEEE

Robles, MSR'10

2/154 experimental papers "replication friendly."

48 due to lack of published data

Prior research results are difficult
(or impossible) to reproduce.

Boa makes this easier!

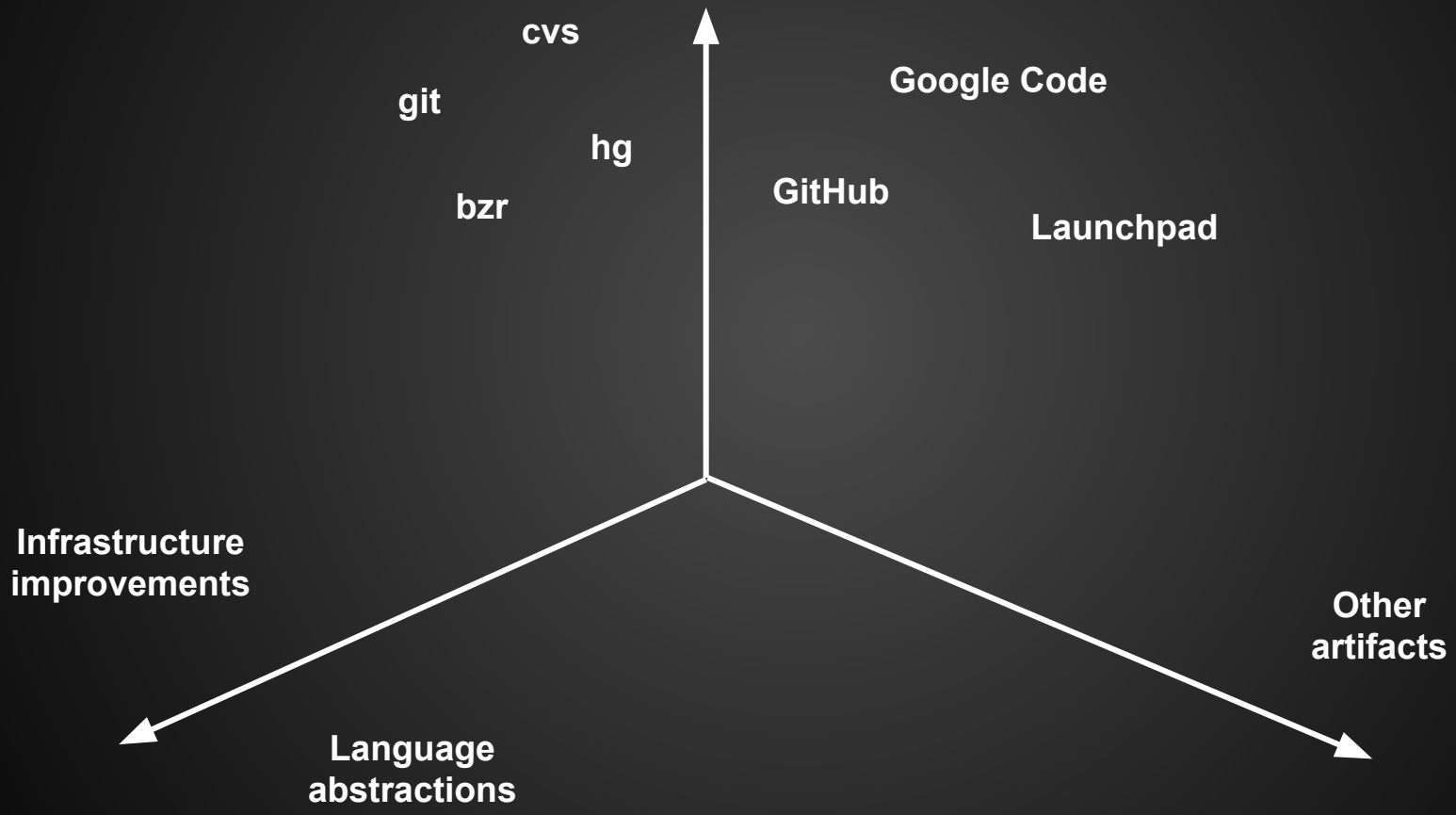
Controlled Experiment

- Published artifacts (Boa website):
  - Boa source code
  - Dataset used (timestamp of data)
  - Results

		Intro	Task 1		Task 2		Task 3	
Expert	**Education**	**Time**	**Task**	**Time**	**Task**	**Time**	**Task**	**Time**
Yes	Post-doc	6	B.1	1	B.6	4	B.9	3
Yes	PhD	5	A.1	3	B.6	2	B.7	6
No	PhD	4	B.6	1	B.10	4	B.9	4
No	PhD	4	A.2	2	B.6	2	D.5	4
No	MS	4	A.1	4	B.6	1	D.3	2
No	MS	3	B.6	2	C.1	2	D.4	10
No	MS	6	A.1	2	B.7	3	B.10	3
No	BS	2	A.2	2	D.1	2	D.3	2

Fig. 16. Study results. All times given in minutes.

Ongoing work



Boa

http://boa.cs.iastate.edu/

- Domain-specific language and infrastructure for software repository mining that is:
  - Easy to use
  - Efficient and scalable
  - Amenable to reproducing prior results

Related - MSR

Sourcerer

- Linstead et.al. 2009

PROMISE

- Menzies et.al. 2009

Kenyon

- Bevan et.al 2005

Related - Data-Parallel

MapReduce

Dean and Ghemawat 2004

Hadoop

Dryad

Isard et.al. 2007

Related - Data-Parallel

Sawzall

Pike et.al. 2005

PigLatin

Olston et.al. 2008

FlumeJava

Chambers et.al. 2010

Related - Visitors

GOF Visitor pattern 1994

DemeterJ/DJ

Orleans and Lieberherr 2001

Recursive Traversals

Ovlinger and Wand 1999

Related - Studies

Java generics

- Parnin et.al. 2011

Treasure

- Grechanik et.al 2010

Language adoption

- Meyerovich and Rabkin 2013

Related - Optimizations

Chain folding, job merging

Miner and Shook 2012

sibling fusion, MSCR fusion (FlumeJava)

Chambers et.al. 2012

ChainMapper/ChainReducer (Hadoop)

Domain-specific functions

http://boa.cs.iastate.edu/docs/dsl-functions.php

```
hasfiletype := function (rev: Revision, ext: string) : bool {
    exists (i: int; match(format(`\.%s$`, ext), rev.files[i].name))
        return true;
    return false;
};
```

Mines a revision to see if it contains any files of the type specified.

Domain-specific functions

http://boa.cs.iastate.edu/docs/dsl-functions.php

```
isfixingrevision := function (log: string) : bool {
    if (match(`\bfix(s|es|ing|ed)?\b`, log))    return true;
    if (match(`\b(error|bug|issue)(s)\b`, log)) return true;
    return false;
};
```

Mines a revision log to see if it fixed a bug.