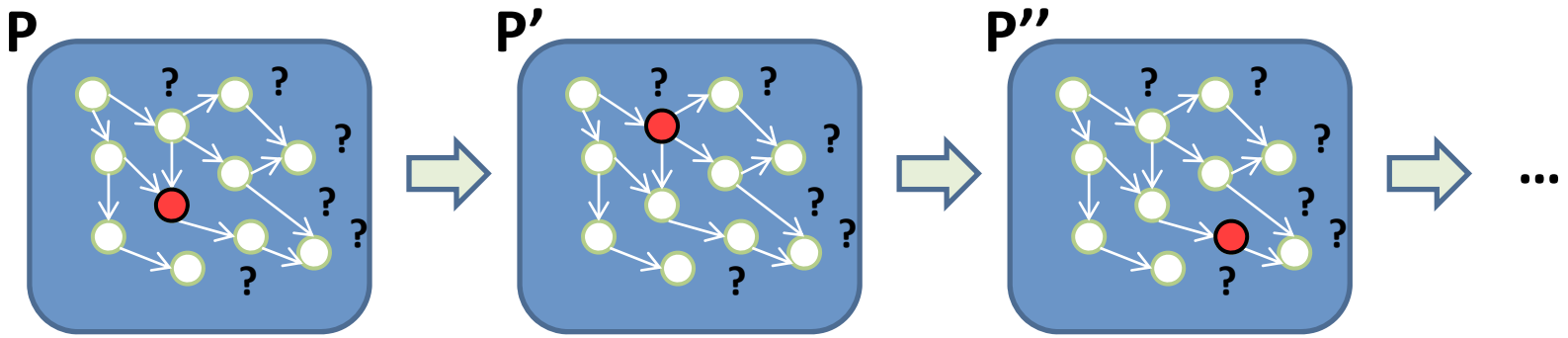
Supported by ONR award N000141410037

DIVER: Precise Dynamic Impact Analysis Using Dependence-based Trace Pruning

Haipeng Cai and **Raul Santelices**

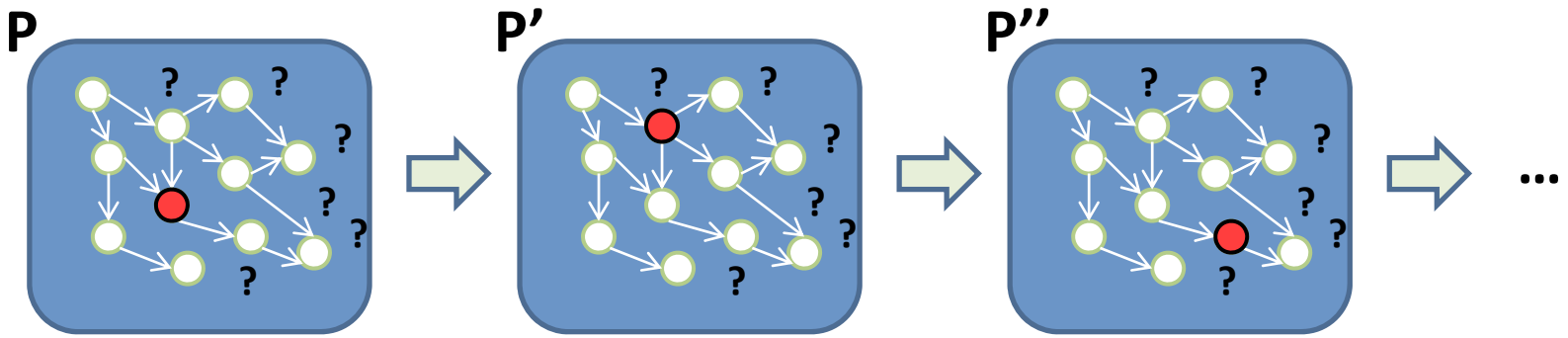
U. of Notre Dame, U.S.A.

Problem



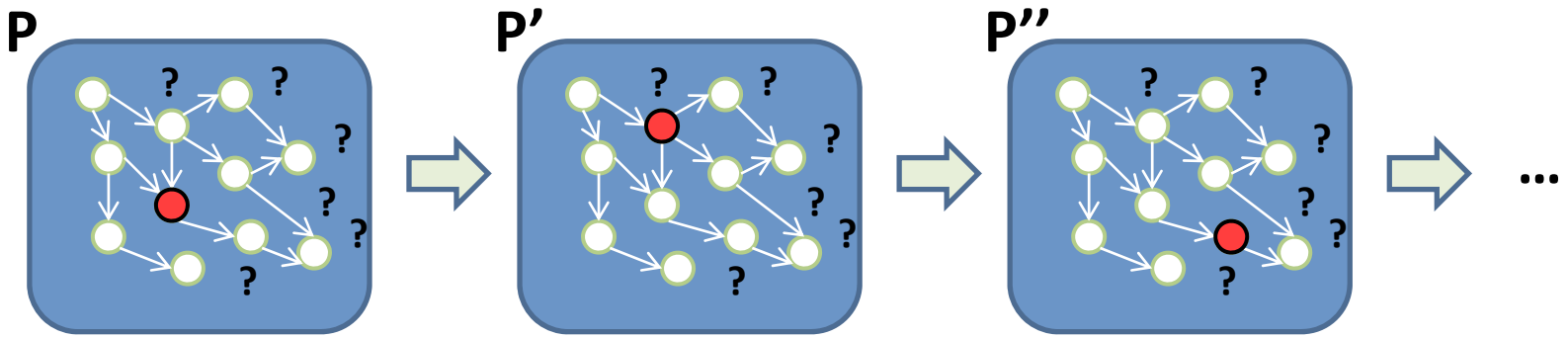
- Change-impact analysis (or, simply, impact analysis)
- Many types of impact analyses
  - Static, dynamic, hybrid, repository-based, information retrieval
  - Granularity: files, methods, statements
- Dynamic and method-level: scalable and representative of actual behavior

Dynamic Impact Analysis



- Forward dynamic slicing [Korel-Laski '88]
  - Statement level ➔ expensive but precise
  - Would need to analyze all statements in method(s)
- Coverage based with static reachability [Orso et al. '03]
  - Cheap but imprecise [Orso et al. '04]

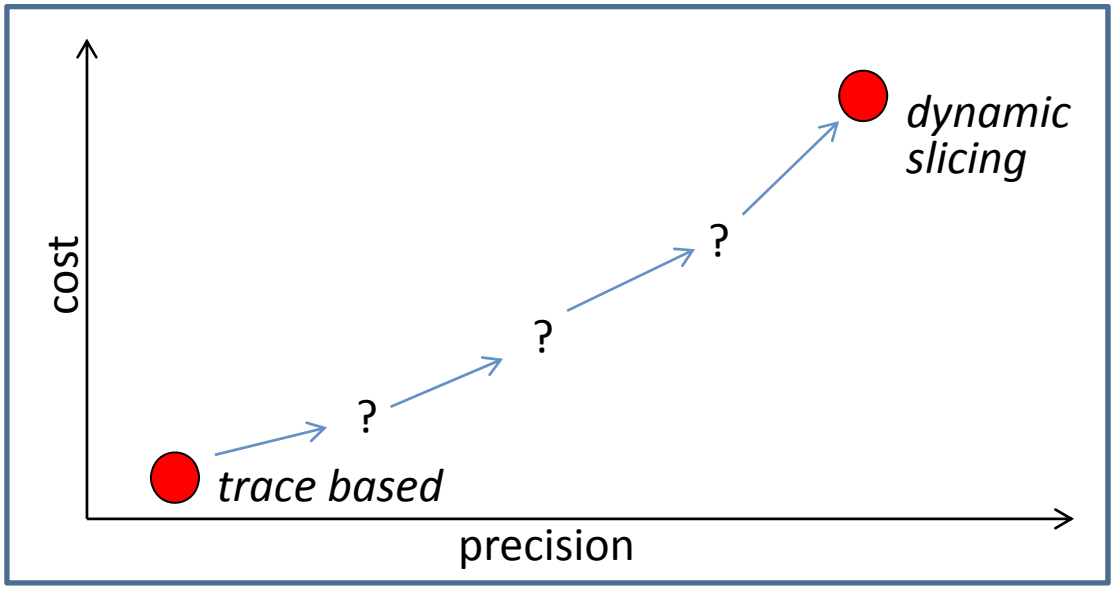
Dynamic Impact Analysis



- Trace based [Law-Rothermel '03], control flow [Ren et al. '04]
  - More precise than coverage based [Orso et al. '04]
  - A bit more expensive after optimization [Apiwattanapong et al. '05]
- Trace based with influence mechanisms [Breech et al. '06]
  - Only marginally better, more expensive

Dynamic Impact Analysis

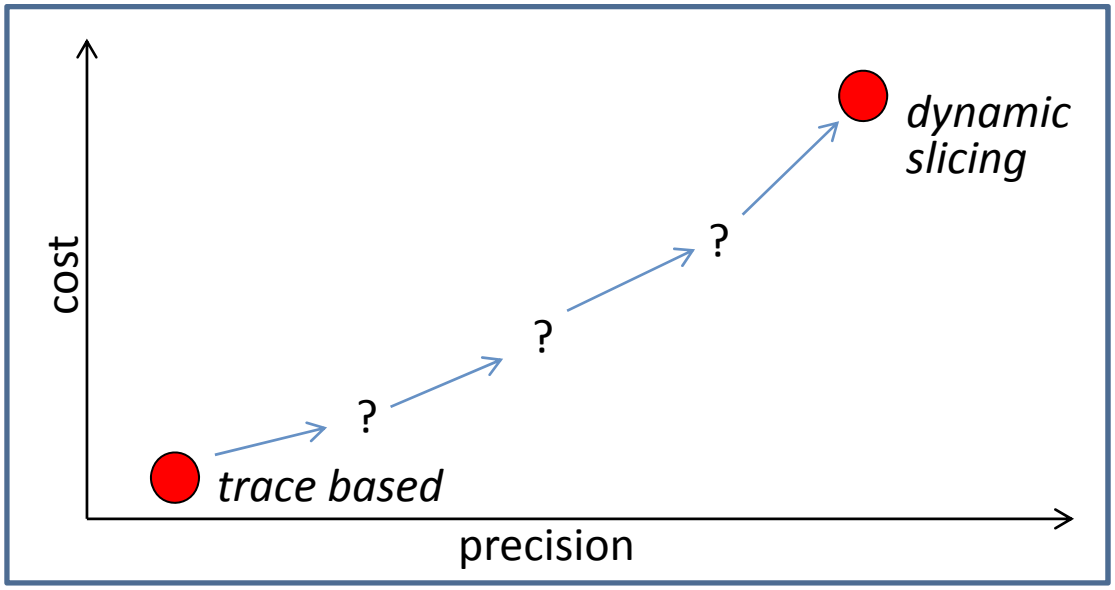
- Problem: trace-based technique is imprecise! [Cai et al. '14]
  - Large fraction of "impacted" methods not really impacted



- Huge gap with dynamic slicing [Jiang et al. '14]
  - There is considerable room for intermediate solutions

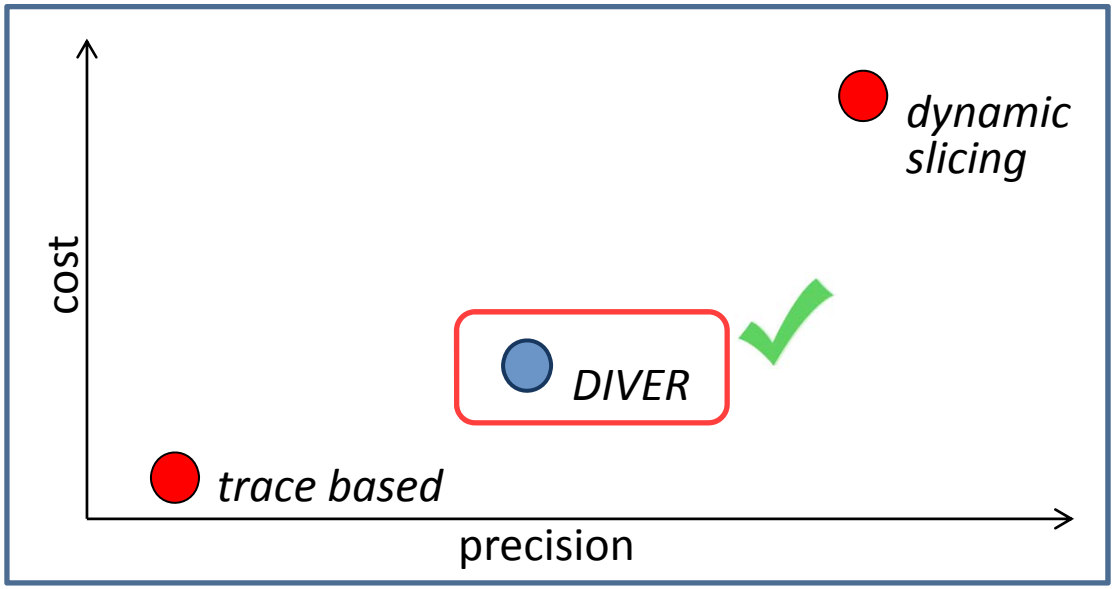
Dynamic Impact Analysis

- What is missing from trace-based?
  - Data & control dependencies not considered (only control flow)
  - Cost is a concern ➔ need method-level dependencies



Dynamic Impact Analysis

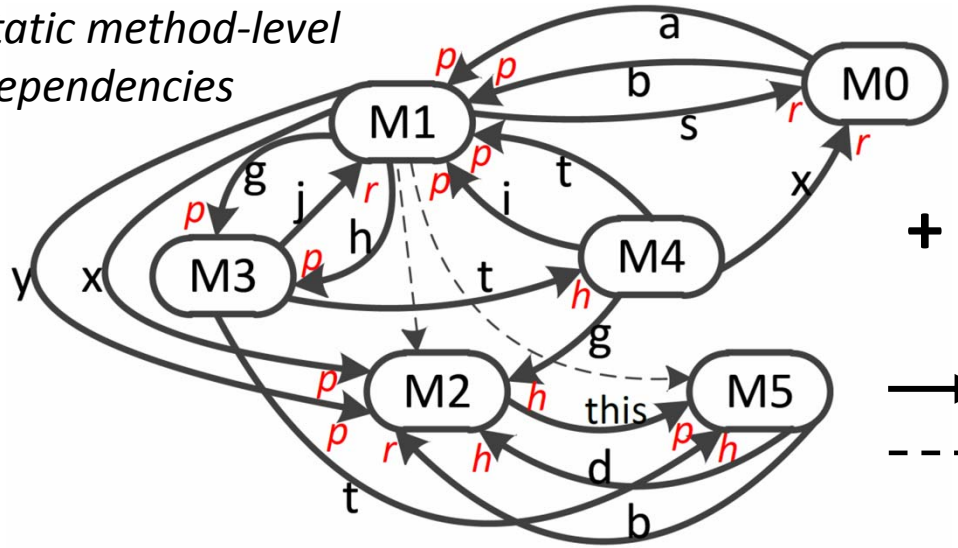
- What is missing from trace-based?
  - Data & control dependencies not considered (only control flow)
  - Cost is a concern ➔ need method-level dependencies



- Solution: one-time static dependence analysis to prune method traces ➔ DIVER

DIVER

static method-level dependencies



+ map method → entry dependence → exit dependence

data dependence

control dependence

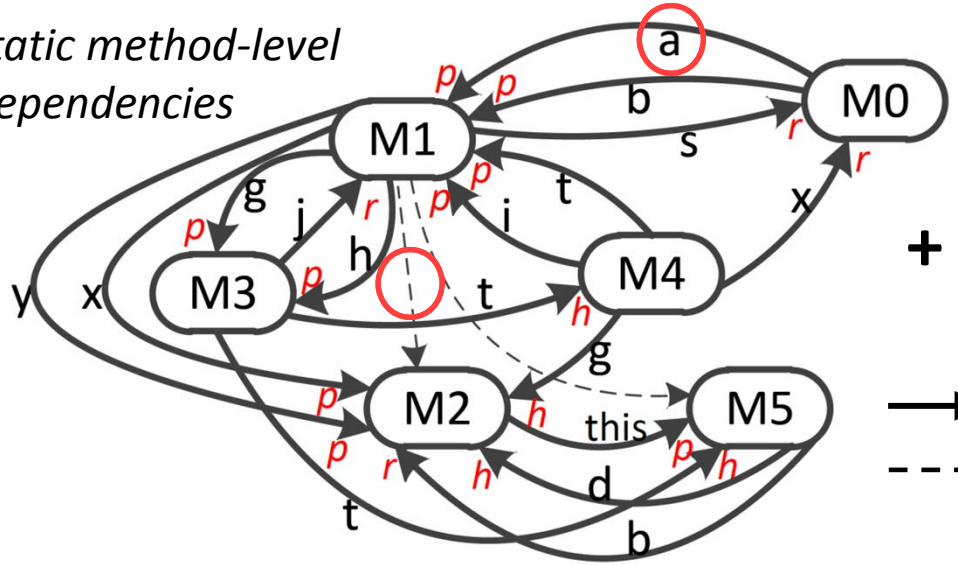
p: parameter
r: return value
h: heap variable

- Example trace: M0 M1 M2 M5 r_{M5} r_{M2} M3 r_{M3} r_{M1} M4 r_{M4} r_{M0}
- Trace-based impact set of M2: {M0, M1, M2, M3, M4, M5}
  - All methods called or returned into after M2
- DIVER impact set of M2: {M2, M5} (just **two** methods)
  - Down from **six** methods when using just traces (control flow)

definitely not dependent!

DIVER

static method-level dependencies



+ map method → entry dependence → exit dependence

→ data dependence

- - → control dependence

p: parameter
r: return value
h: heap variable

- Step 1: statically identify escaping variables and conditional call sites
- Step 2: collect compressed method trace(s)
- Step 3: traverse trace(s) using rules to prune non-dependent methods
  - Ex: M2 can impact only M5
  - Ex: M0 impacts M1 only if M1 occurs immediately after
  - Also: keep track of which dependencies carry an impact

Evaluation (latest!)

- 7 Java applications
  - Up from 4 in paper
- Open-source toolset*
- All executed methods
  - Impact set for each using trace-based and Diver

Subject	KLOC	Methods	Tests
schedule	0.3	20	2,650
nanoxml	3.5	172	214
ant	18.8	607	112
xml-sec.	22.4	632	92
jmeter	35.5	732	79
jaba	37.9	1,129	70
argouml	102.4	1,098	211

* http://nd.edu/~hcai/diver and http://nd.edu/~rsanteli/duaf [Santelices et al. '13]

Results (latest!)

Average impact set sizes

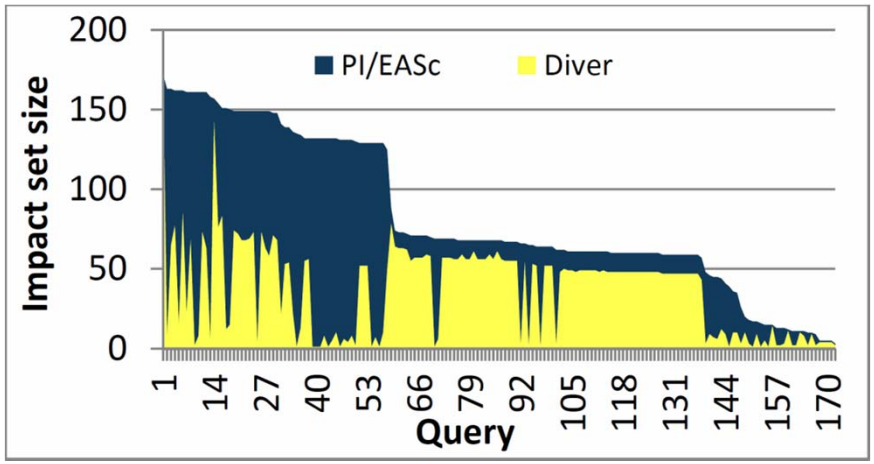
Average size ratios

		average impact-set size		
Subject	**Methods**	**Trace based**	**DIVER**	**Ratio**
schedule	20	18.0	12.8	71.3%
nanoxml	172	82.6	37.1	51.7%
ant	607	159.5	17.9	25.7%
xml-sec.	632	199.8	45.1	28.8%
jmeter	732	149.6	12.3	18.8%
jaba	1,129	677.0	471.9	66.9%
argouml	1,098	151.0	27.6	31.5%
	average:	***291.4***	***141.4***	***38.3%***

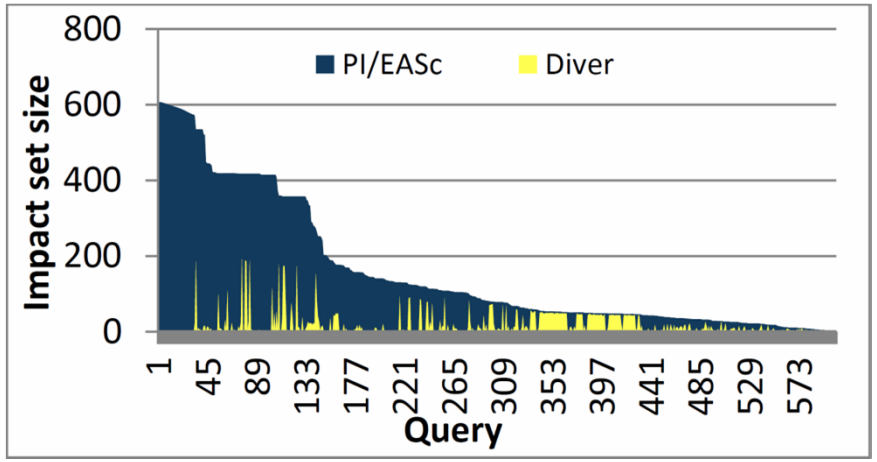
Results (latest!)

PI/EASc = trace based

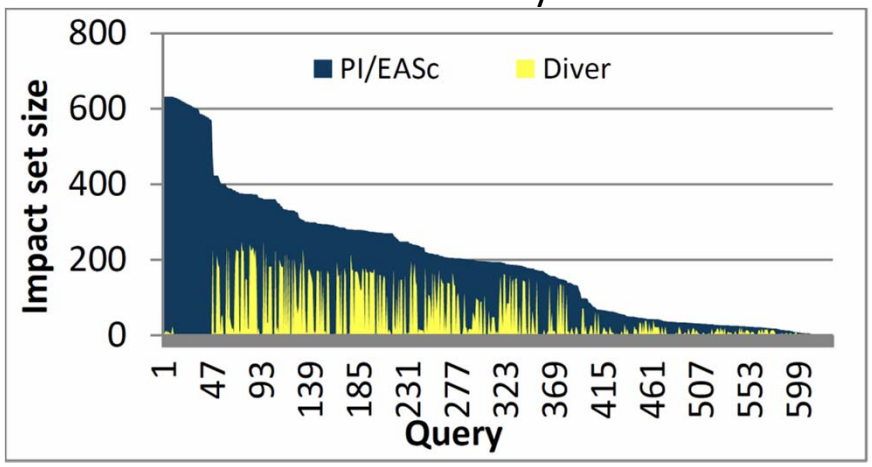
NanoXML



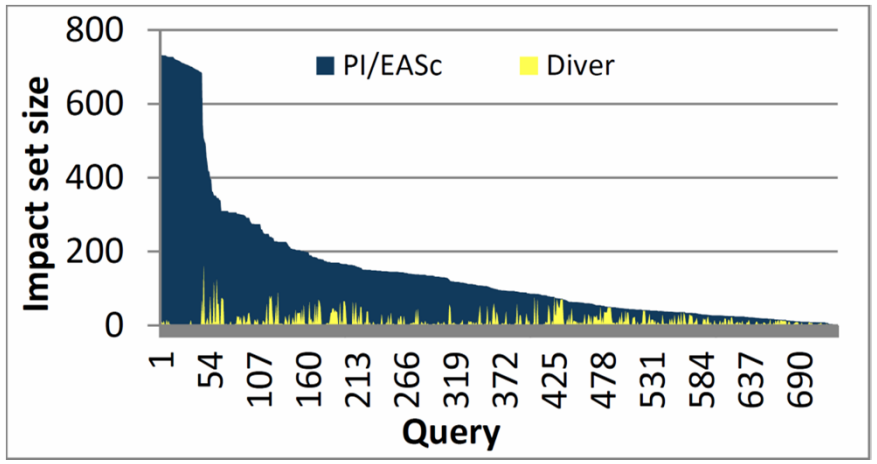
Ant



XML-Security



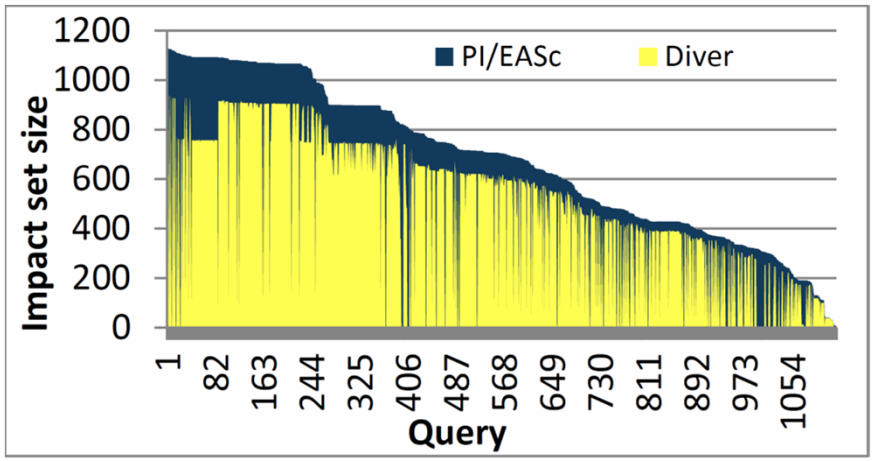
JMeter



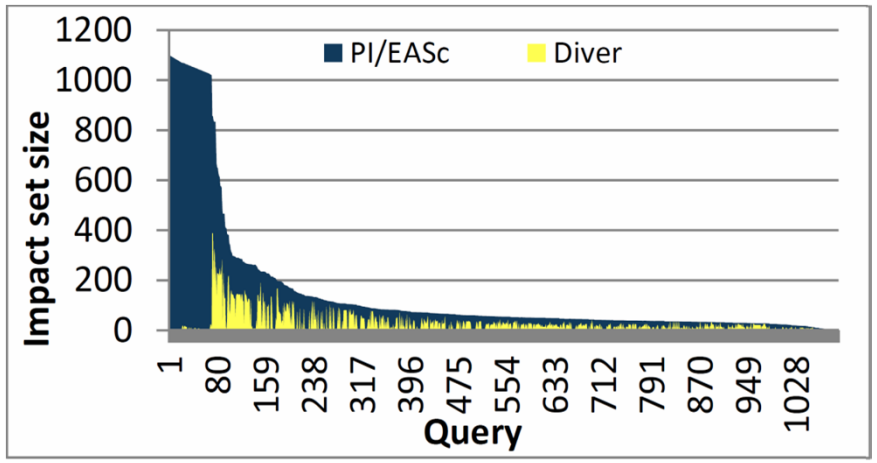
Results (latest!)

PI/EASc = trace based

Jaba



ArgoUML



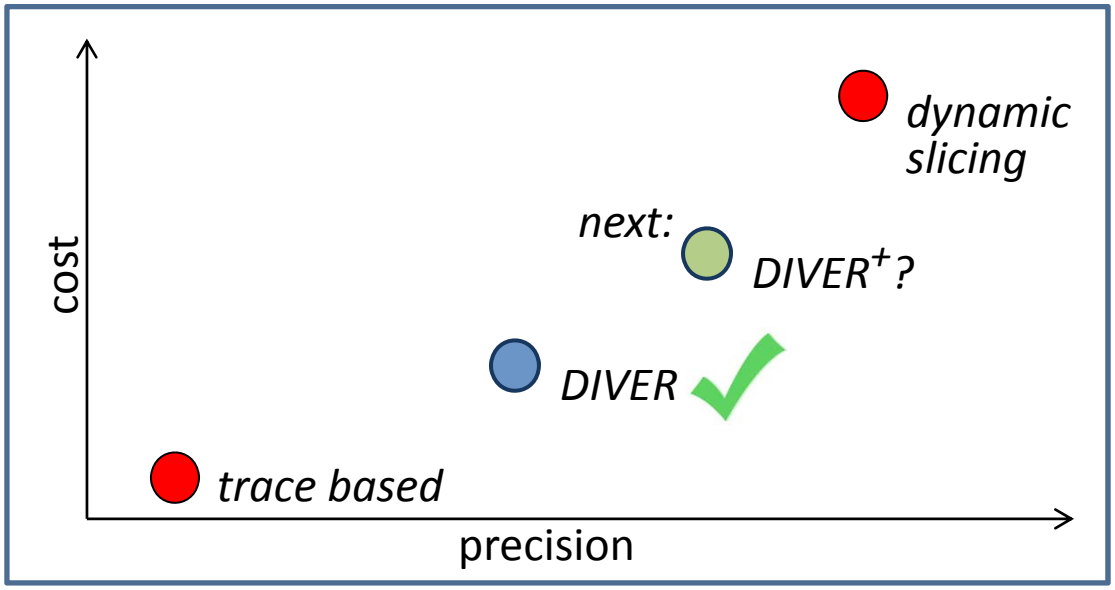
- Costs of DIVER
  - Step 1: average 2K seconds per subject (one-time analysis)
    - ≤ 41 MB dependence information
  - Step 2: average 11.6 seconds (vs 8.6 sec. trace based)
    - ≤ 15 MB compressed traces
  - Step 3: average 26.4 sec/query (vs 0.1 sec/query trace based)

parallelizable!

Conclusion

Questions?

- Huge cost-precision gap in previous techniques
  - New idea: method-level dependencies (DIVER)



UNIVERSITY OF NOTRE DAME

Supported by ONR award N000141410037