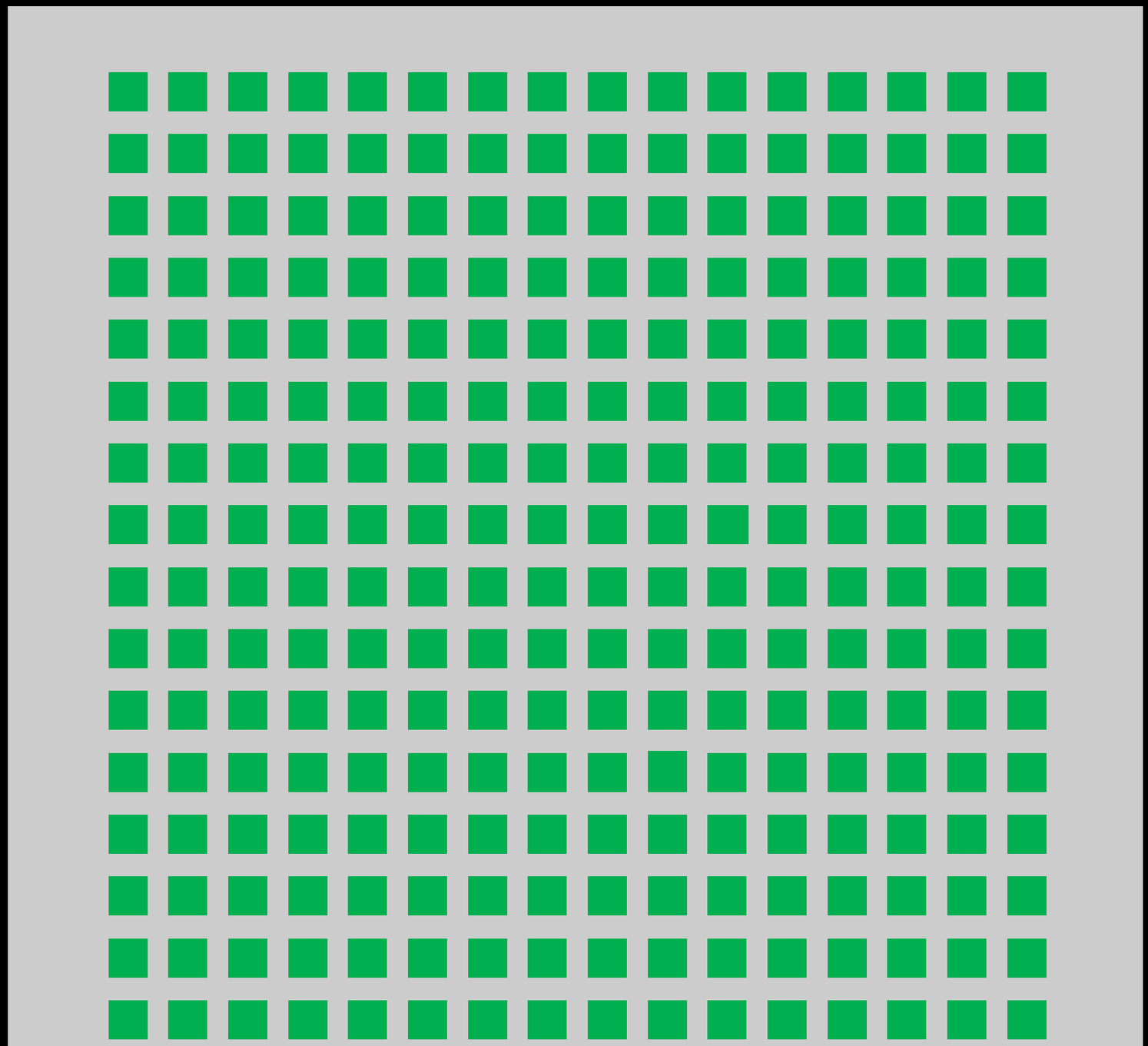
BugCache

Predicting Defects

Sung Kim • MIT
Tom Zimmermann • U. of Calgary
Jim Whitehead • UC Santa Cruz
Andreas Zeller • Saarland University

Dream

All modules are bug-free!



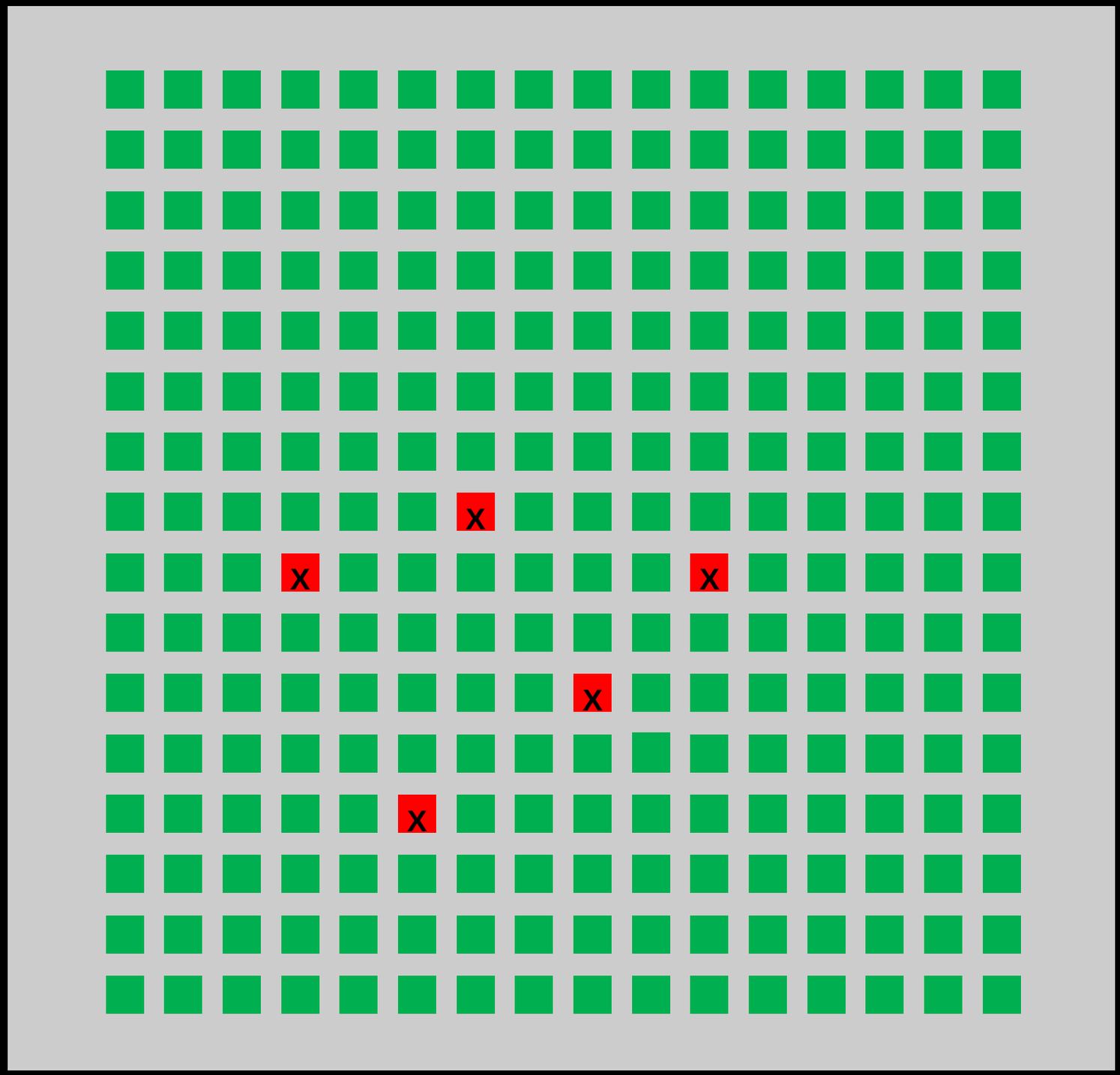
Bug-free module

Some are Buggy

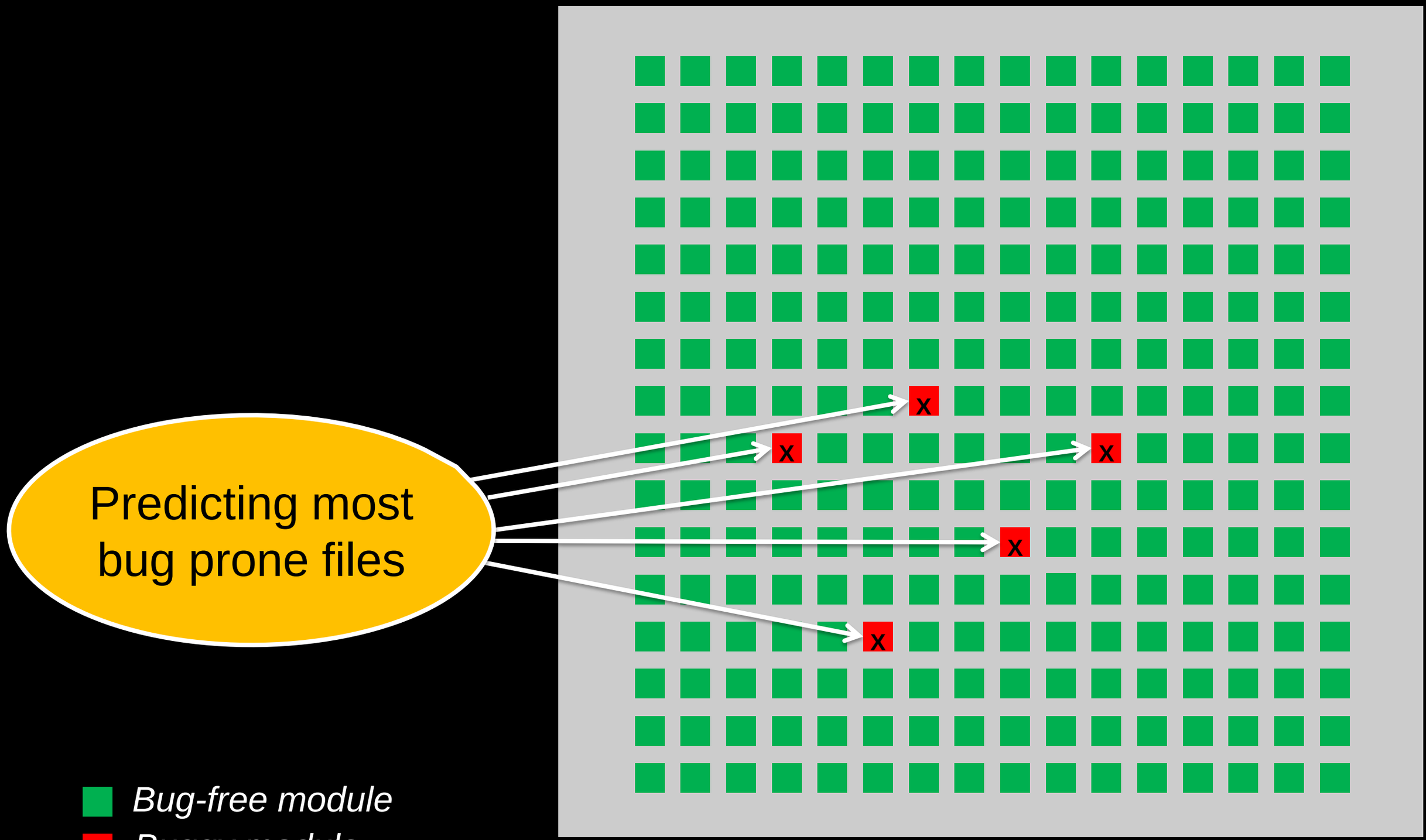
Some modules are buggy!

Bug-free module

x *Buggy module*



Predicting Buggy Files



Motivation

Which files should we focus on?

Where are bugs?

In new files!
[Graves et al.]

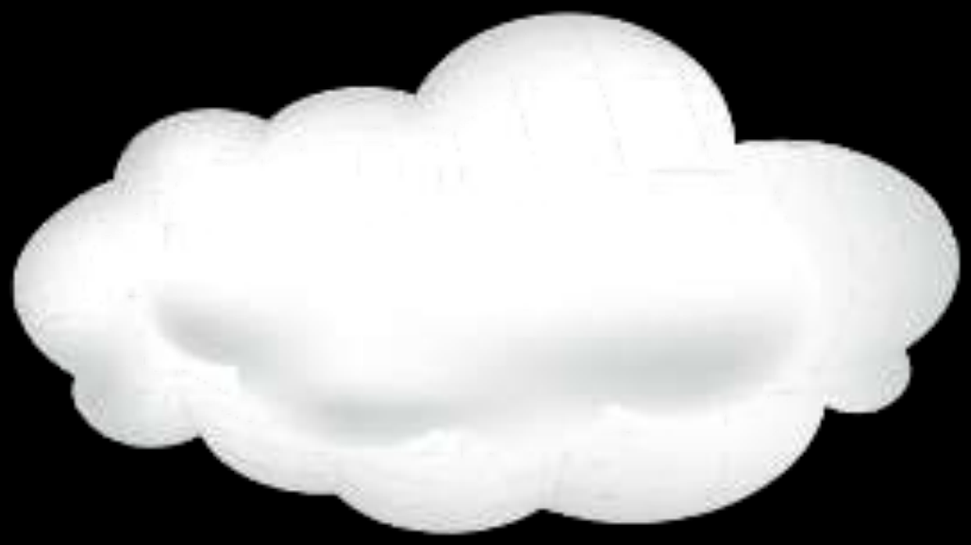
In modified files!
[Nagappan et al.]

Spatial locality:
In nearby other bugs!
[Zimmermann et al.]

Temporal locality:
Defected files are
likely to have more soon.
[Ostrand, Weyuker]

Our Solution

- List of most bug-prone files
- Dynamically adaptive and intuitive
- Combine bug prediction models

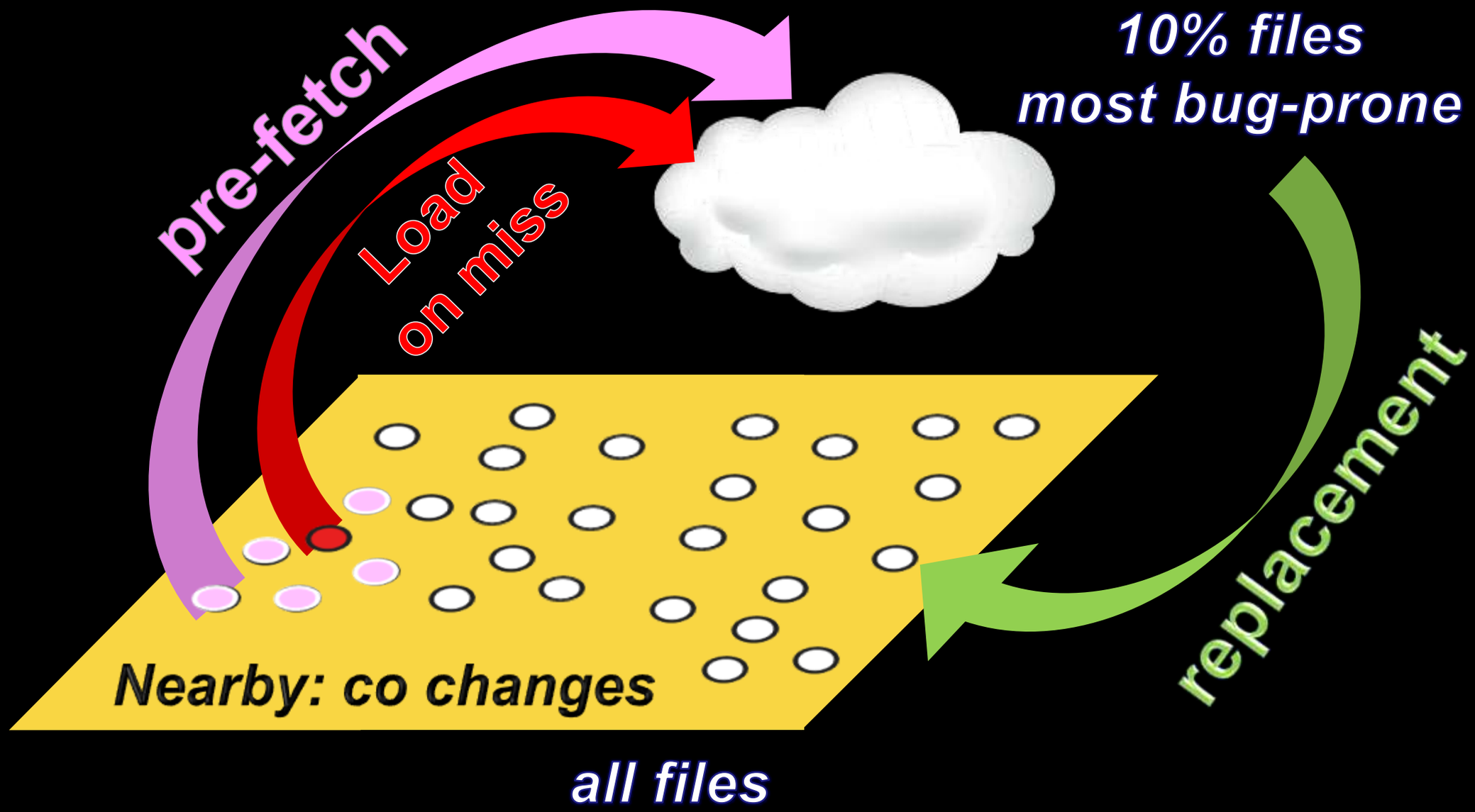


10% BugCache predicts 73~95% of bugs

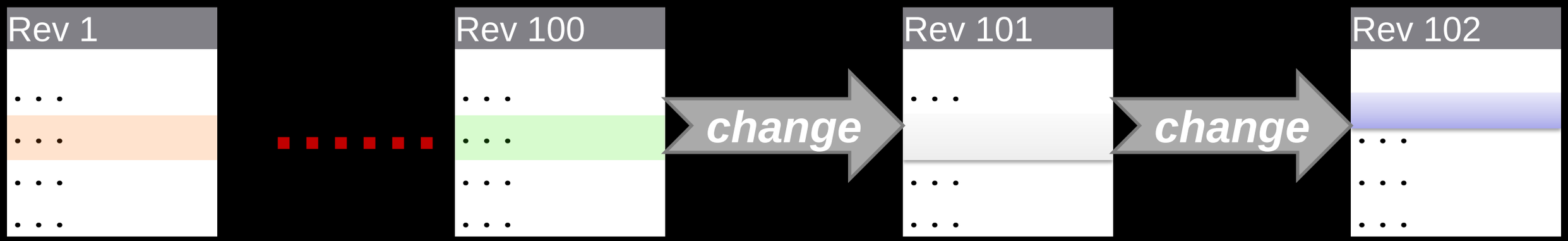
Outline

- Bug Cache Model and Operation
- Evaluation
  - Seven open source projects
- Related Work
- Applications
- Summary

Bug Cache Model

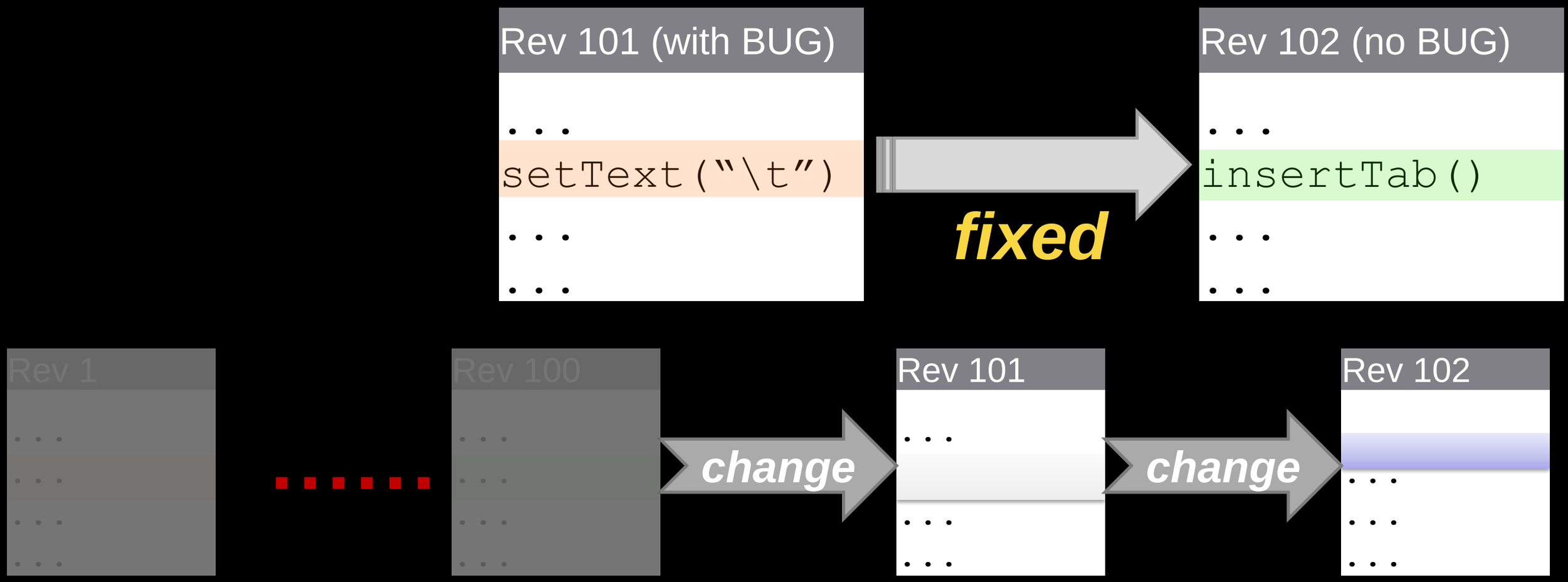


Identifying Bugs



Development history of *JEditTextArea.java*

Identifying Bugs



Development history of *JEditTextArea.java*

Identifying Bugs

Change message: “fix for bug 28434”

Rev 101 (with BUG)

```
...
setText(“\t”)
...
...
```

fixed

Rev 102 (no BUG)

```
...
insertTab()
...
...
```

Rev 1

Rev 100

change

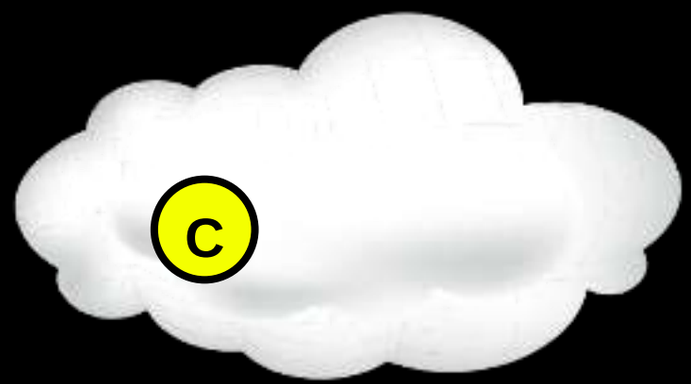
Rev 101

change

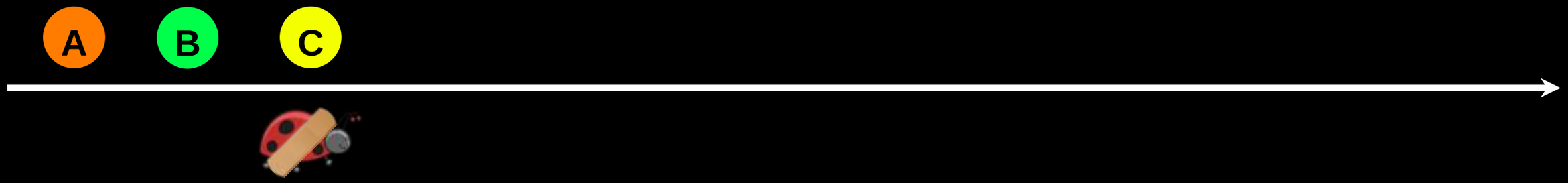
Rev 102

Development history of *JEditTextArea.java*

Cache Operation



Cache size: 2



Miss

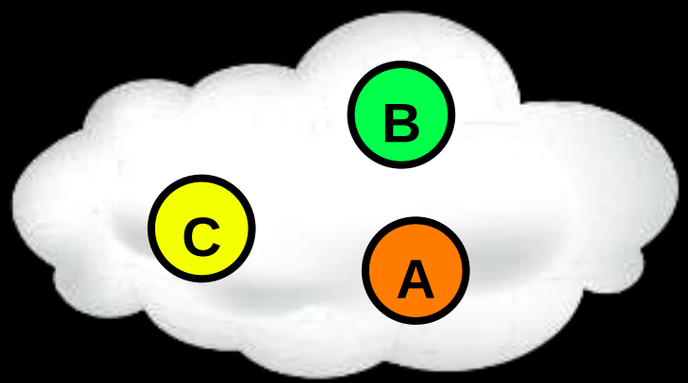
Cache Update

- Load missed files
- Load nearby files (spatial locality)

File	Number of common changes with C.
A	1
B	4
D	0

Parameter: Block size (neighborhood size)

Cache Operation



Cache size: 2
Block size: 2

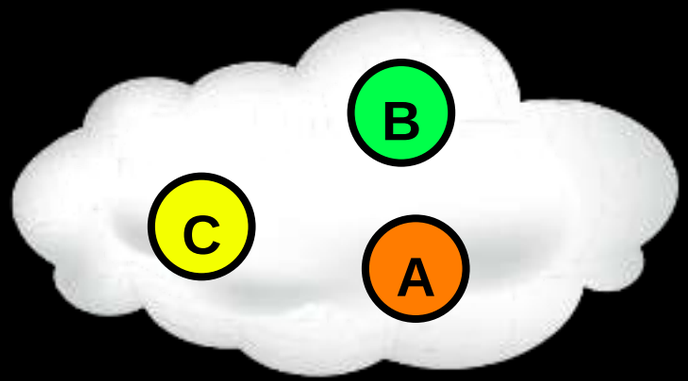
Which one should be replaced?

Replacement Policies

- Least recently used (LRU)
  Unload the files that have the least recently found defect.
- Least frequently changed (CHANGE)
  Unload the files that have the fewest changes.
- Least frequent defects (BUG)
  Unload the files that have the fewest defects.

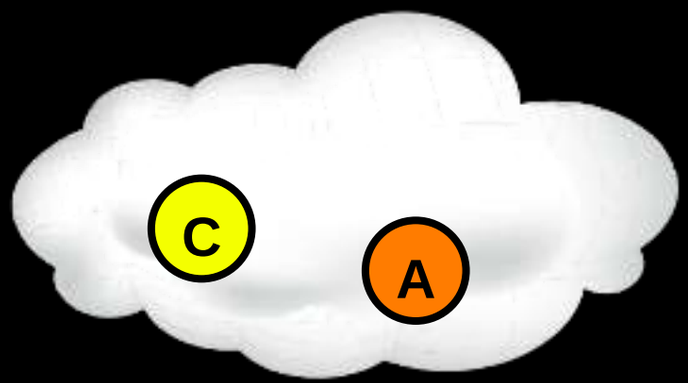
Parameter: Replacement Policy

File	LRU	CHANGE	BUG
C	-5	2	2
B	-3	3	1 (replace)



Cache size: 2
Block size: 2
Replacement: BUG

Cache Evaluation



Cache size: 2
Block size: 2
Replacement: BUG

Hit rate = #Hits / #Defects = 50%

Subject Programs

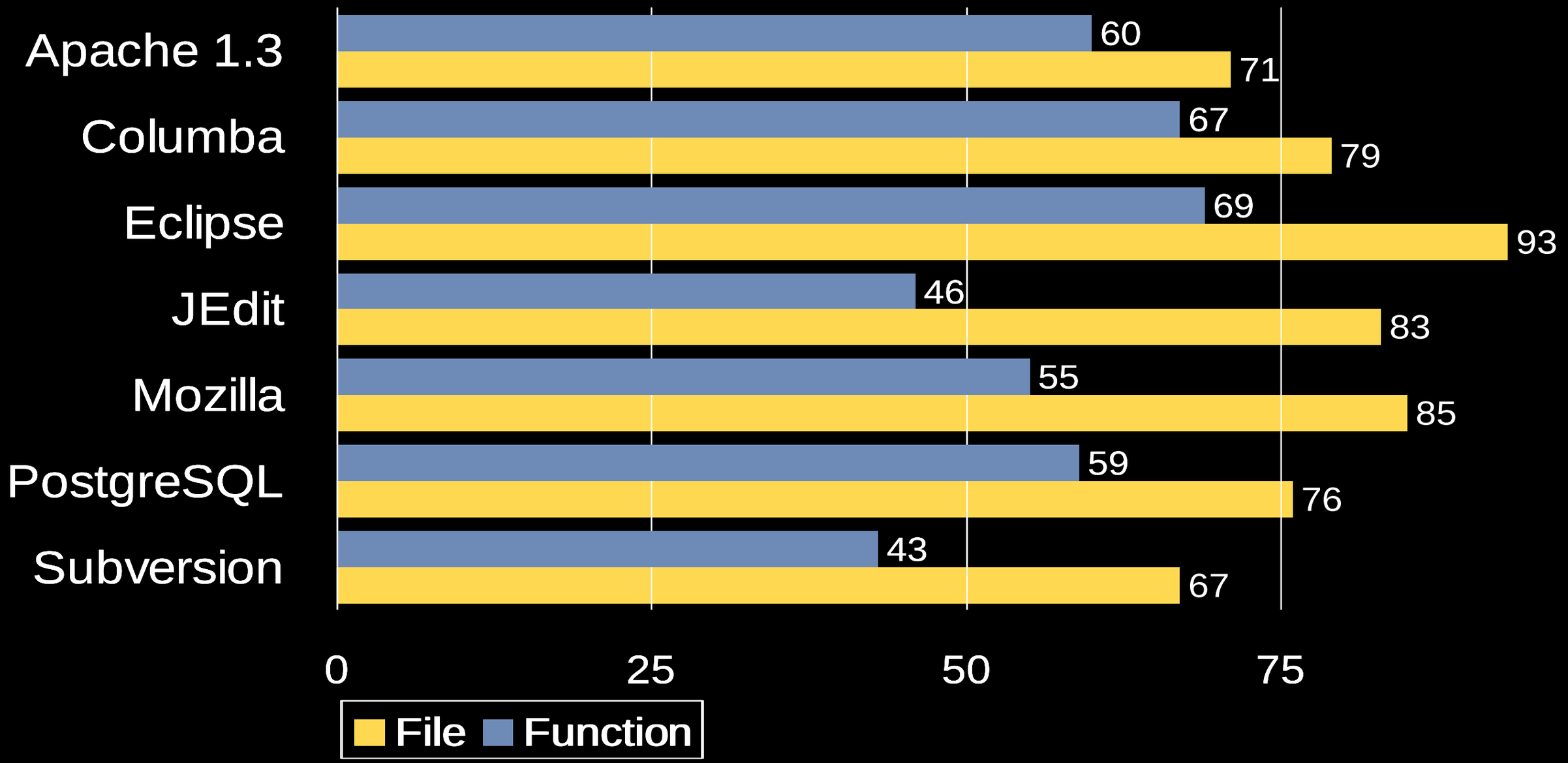
Mozilla

jEdit

PostgreSQL

Columba

Hit Rates

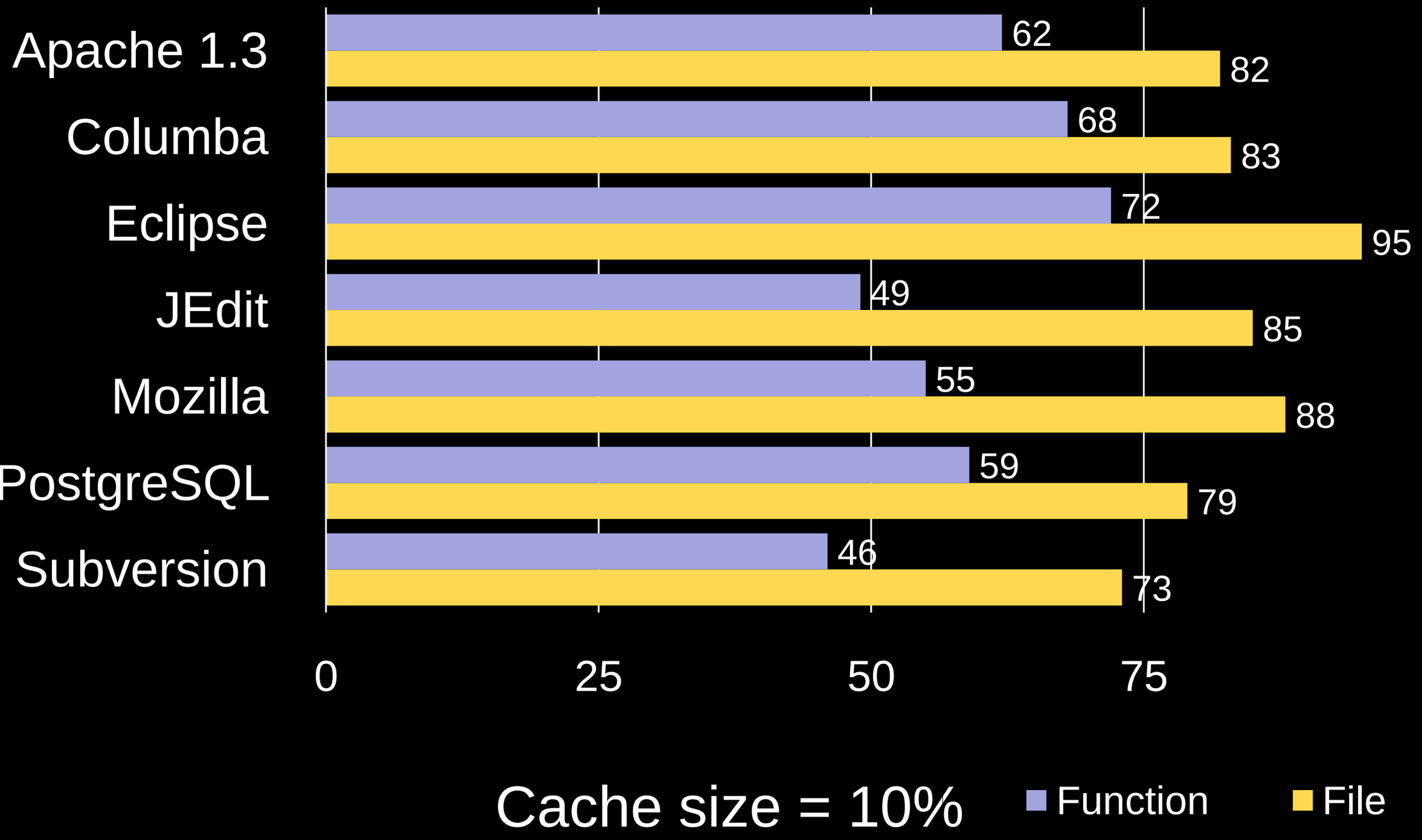


Cache size = 10%
Block size = 50% of the cache size
Replacement policy = LRU

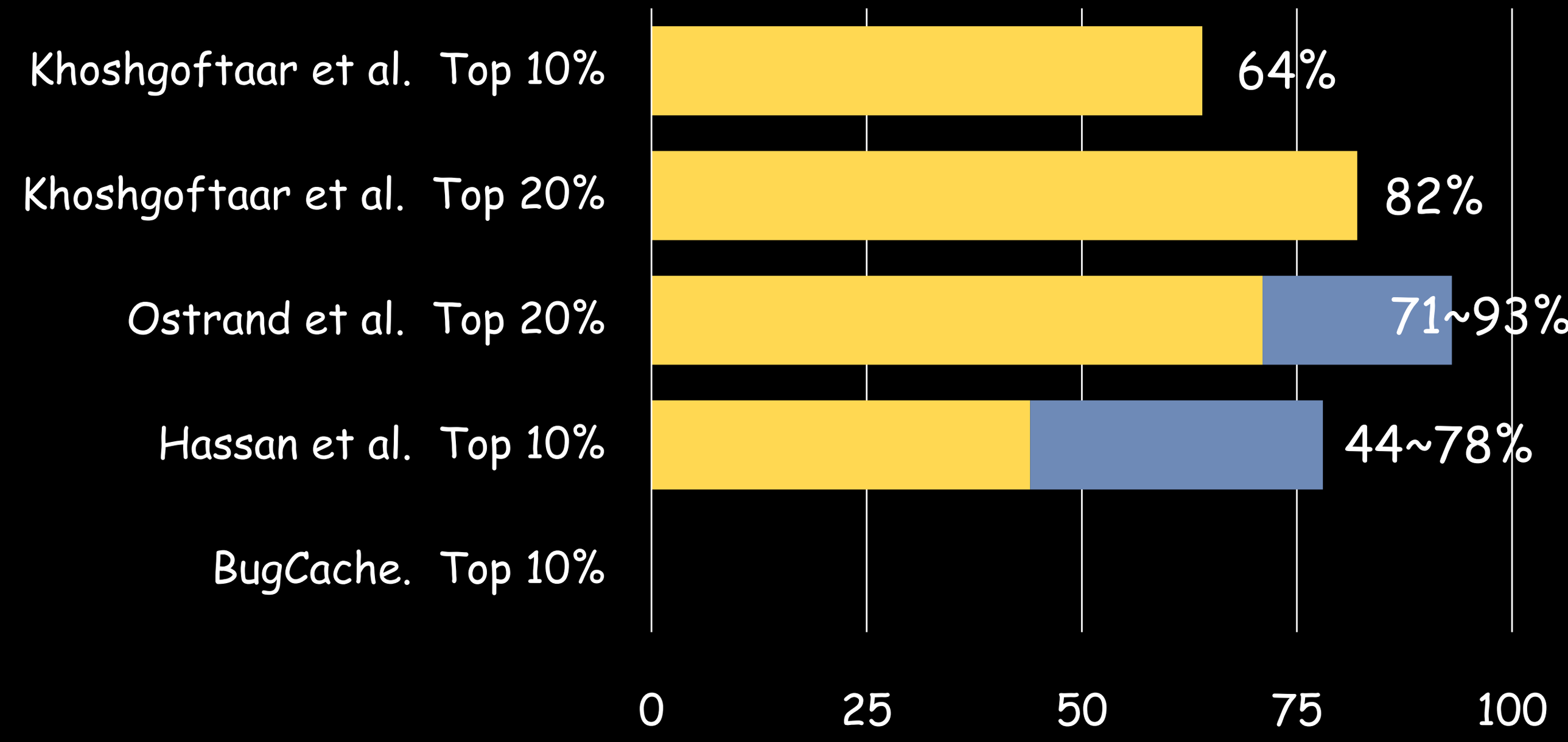
Exhaustive Evaluation

- Cache size: fixed to 10%
- Vary block size:
  0% to 100% of cache size
- Vary replacement: LRU, CHANGE, BUG

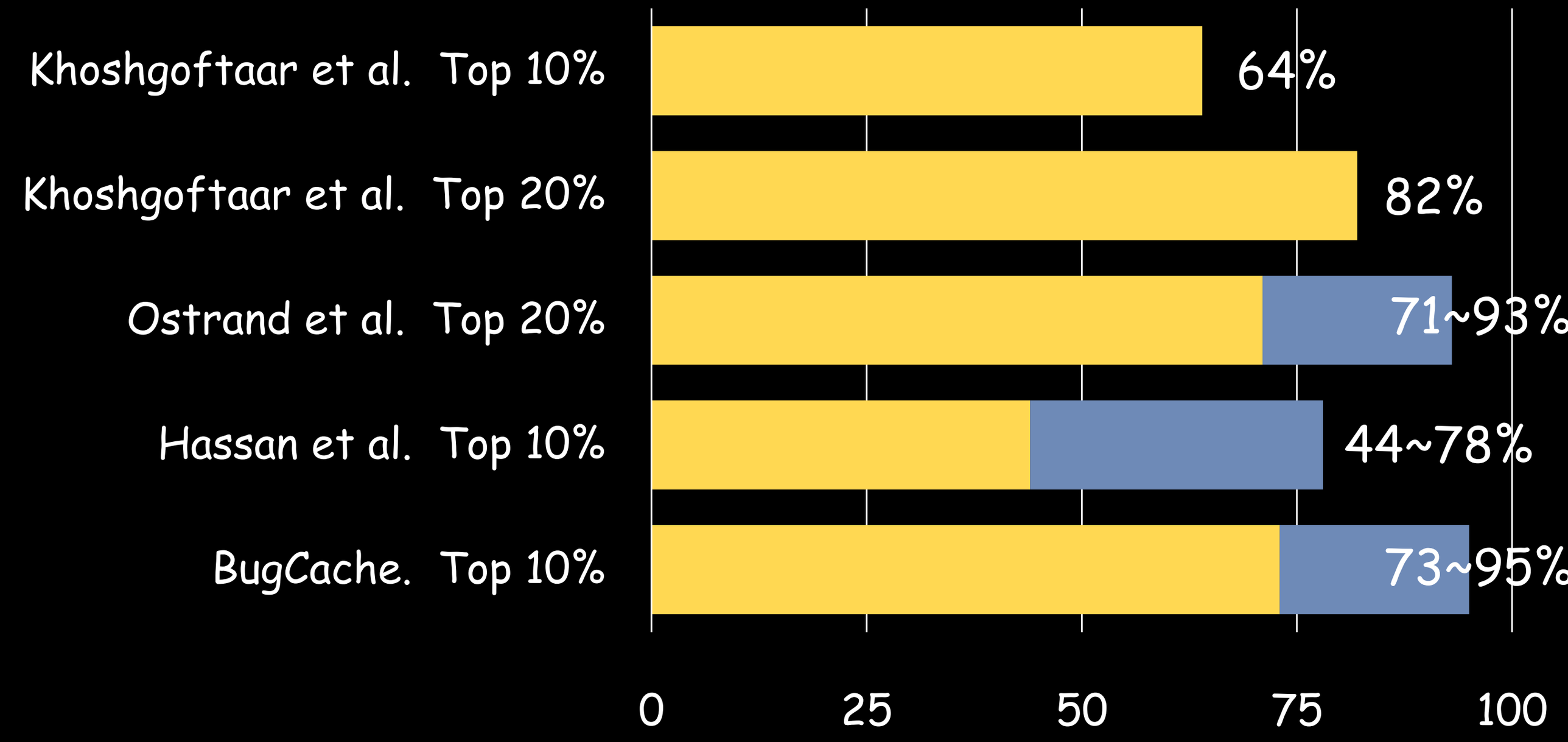
Optimal Hit Rates



Related Work



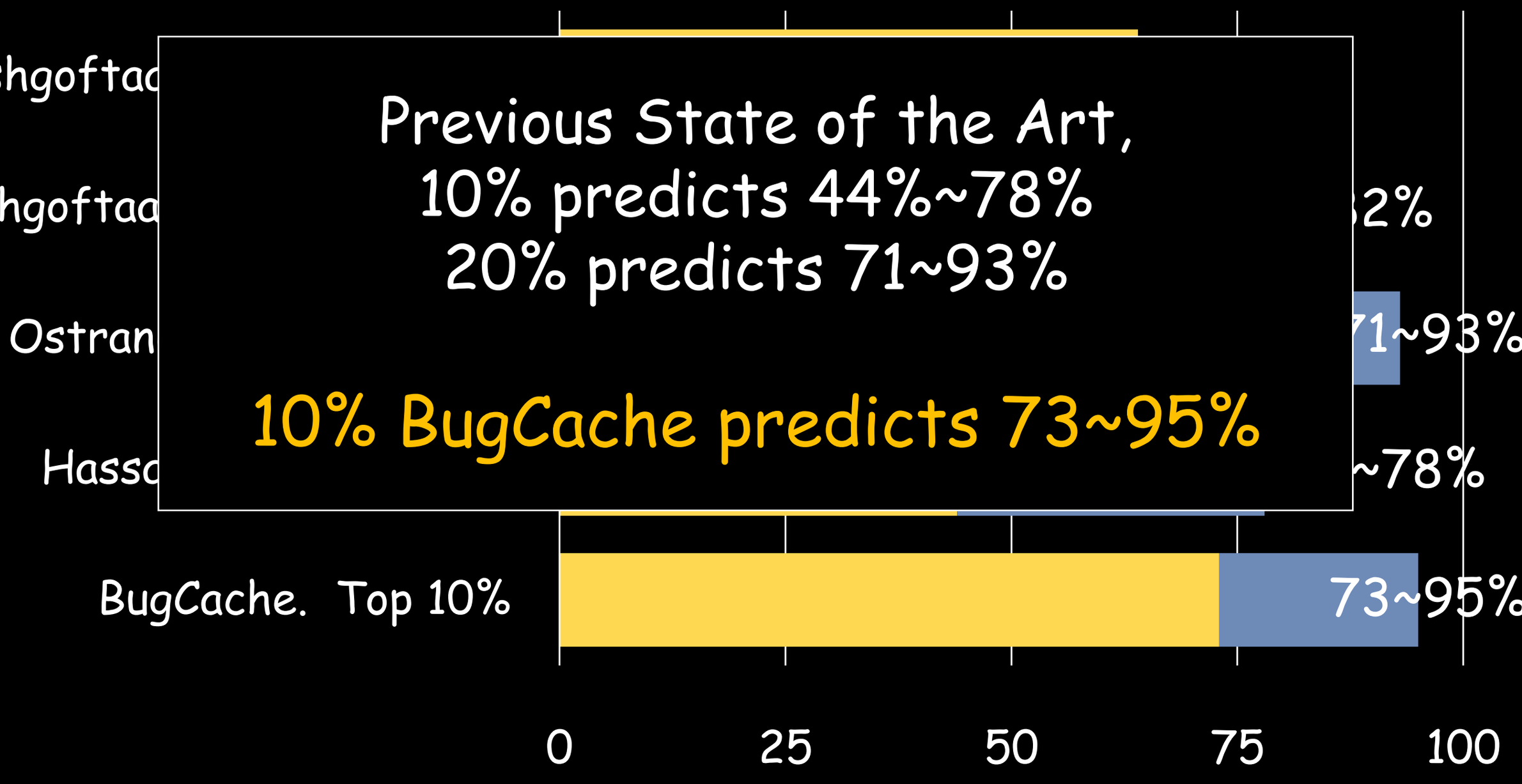
Related Work



Related Work

Previous State of the Art,
10% predicts 44%~78%
20% predicts 71~93%

10% BugCache predicts 73~95%



Applications

- Additional QA efforts
- Selective assertion/error checking
- Dynamic and static (bug cache) analysis combination

Static and Dynamic Analysis

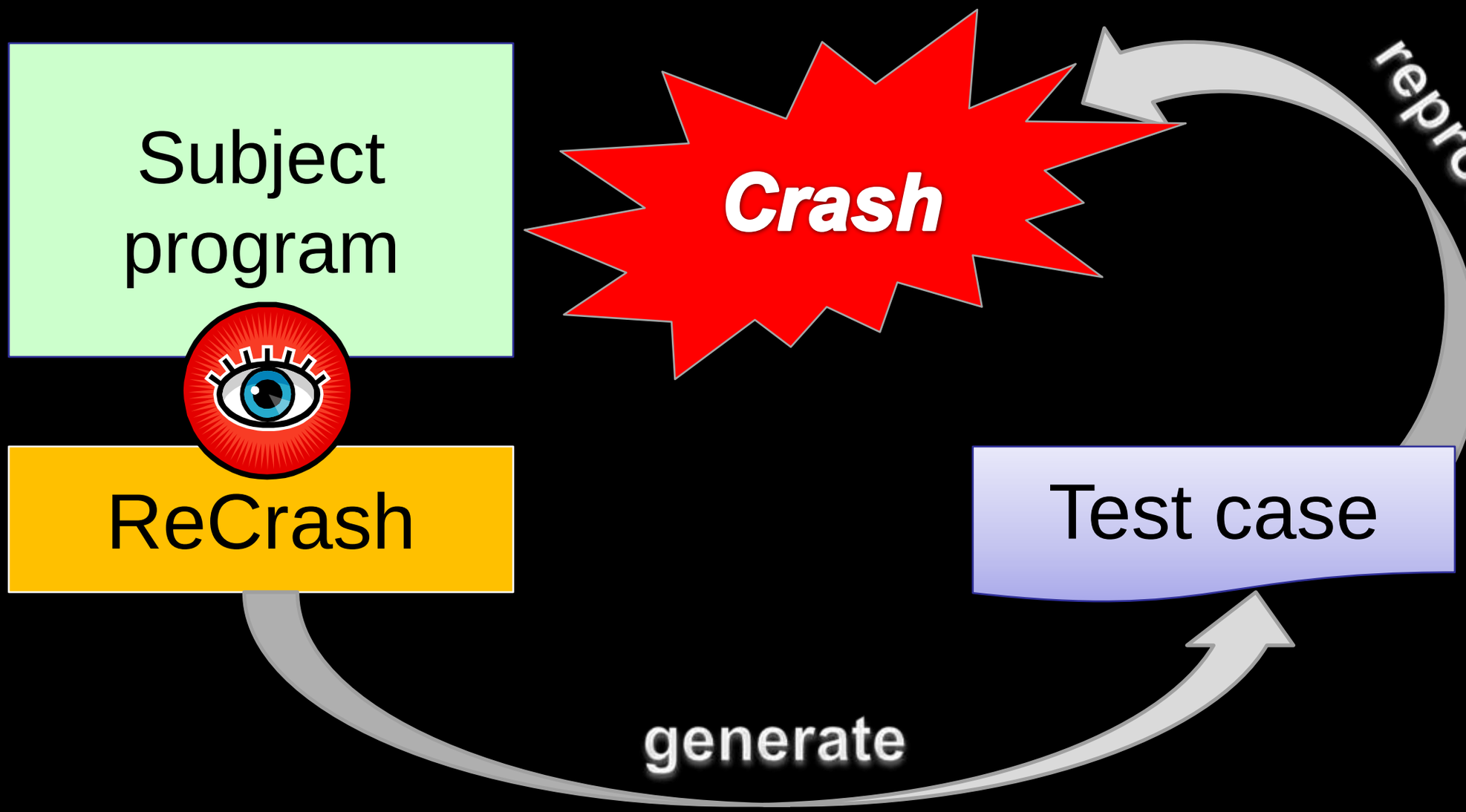
	Overhead	False positives
Static	Low	High
Dynamic	High	Low

- Combine static and dynamic analysis
  - *BugCache + Dynamic Analysis*

Reproducing Crashes

- Reproducing crashes (faults) is hard!
  - Require the exact configuration of crash (in field)
  - Crashes usually involve nondeterministic facts
- Must be able to reproduce crashes to fix bugs and validate fixes

ReCrash

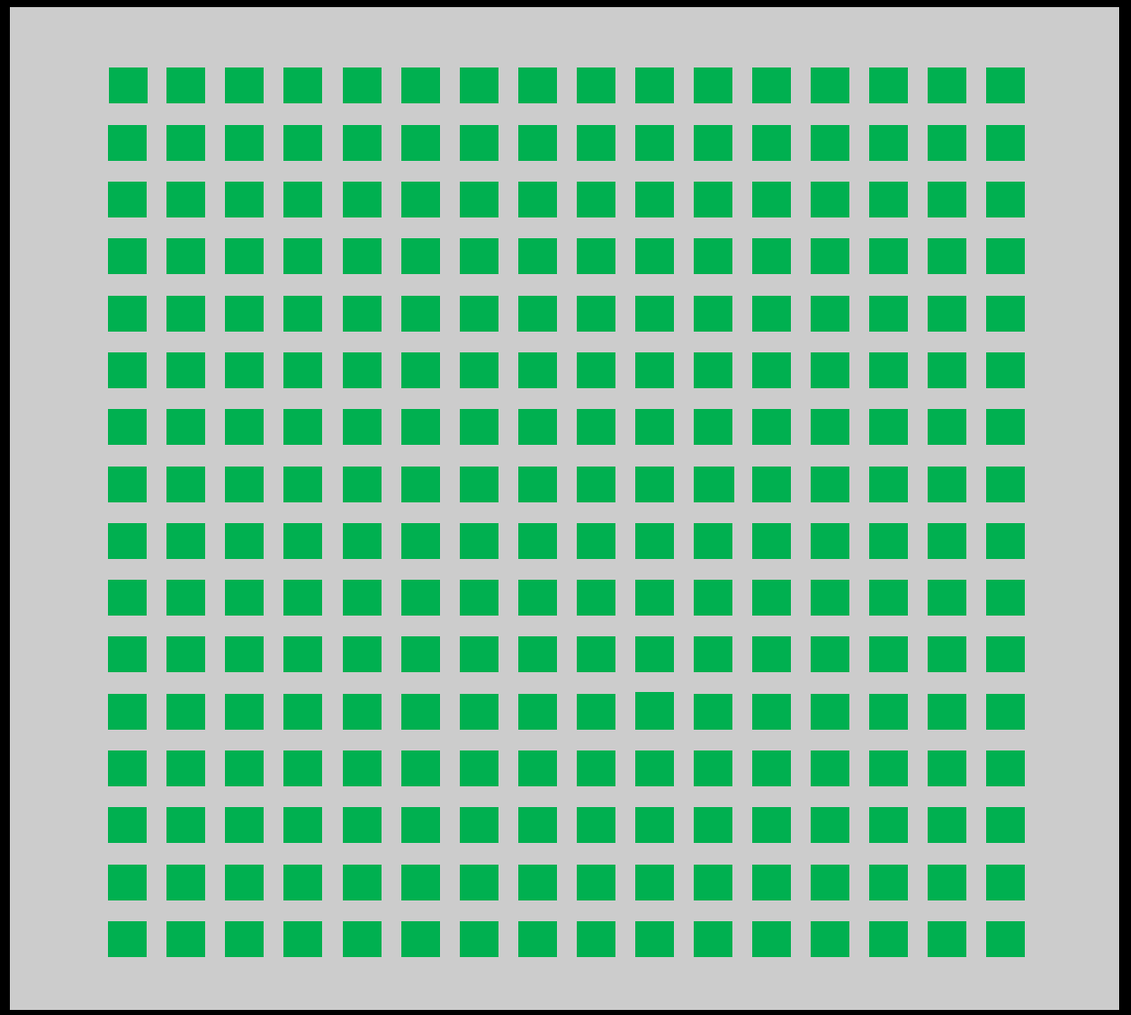


13-64% performance overhead

ReCrash + BugCache

ReCrash

- Monitoring all modules



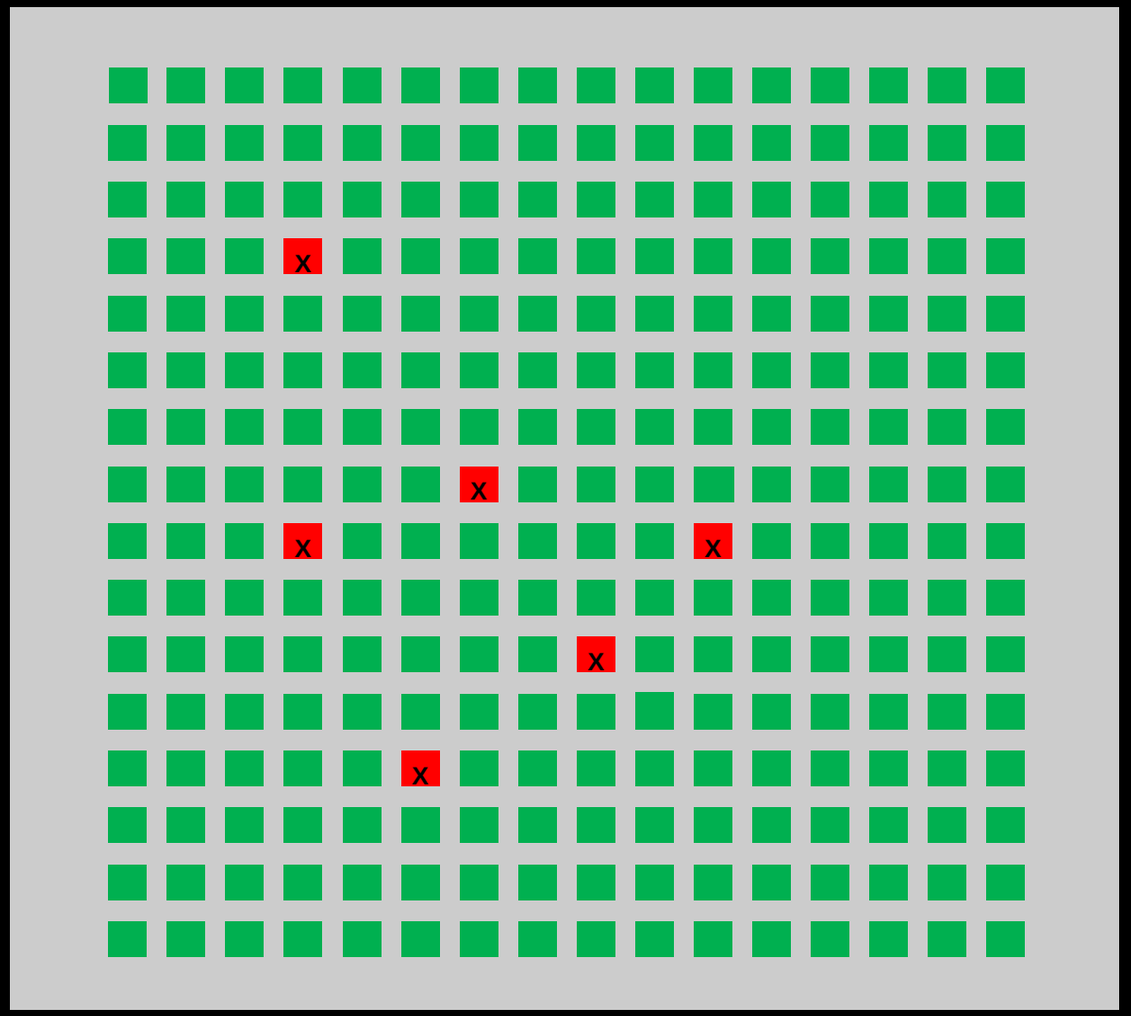
ReCrash + BugCache

ReCrash

- Monitoring all modules

BugCache

- Identify crash-able modules



ReCrash + BugCache

ReCrash

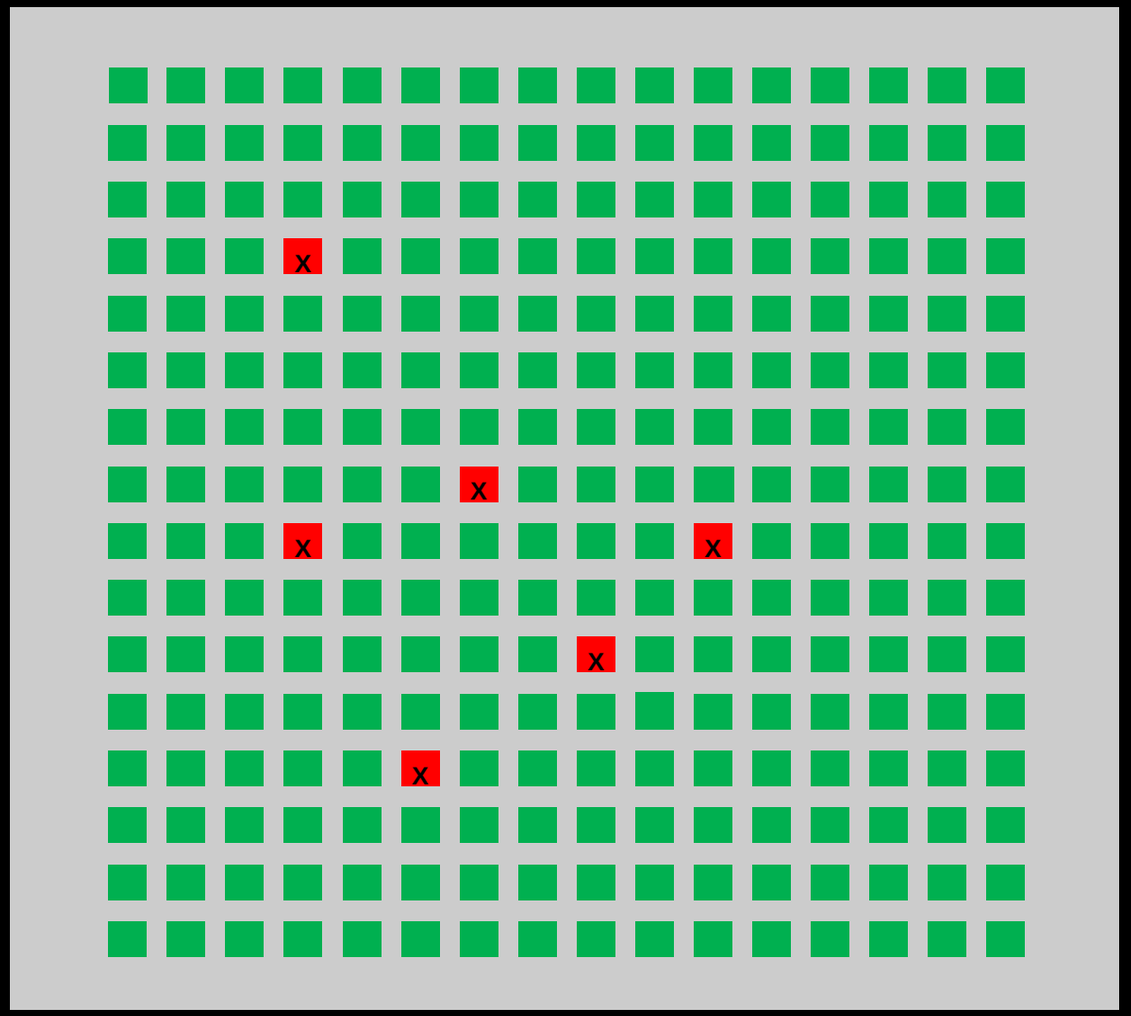
- Monitoring all modules

BugCache

- Identify crash-able modules

ReCrash + BugCache

- Monitoring only identified modules



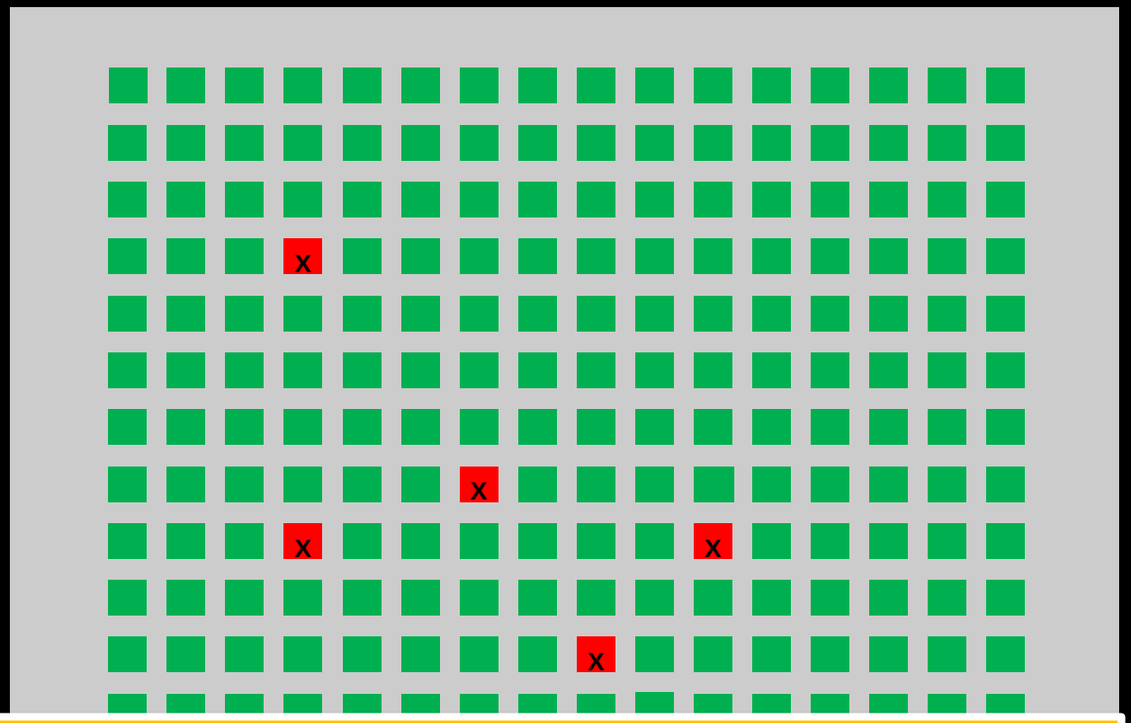
ReCrash + BugCache

ReCrash

- Monitoring all modules

BugCache

- Identify crash-able modules



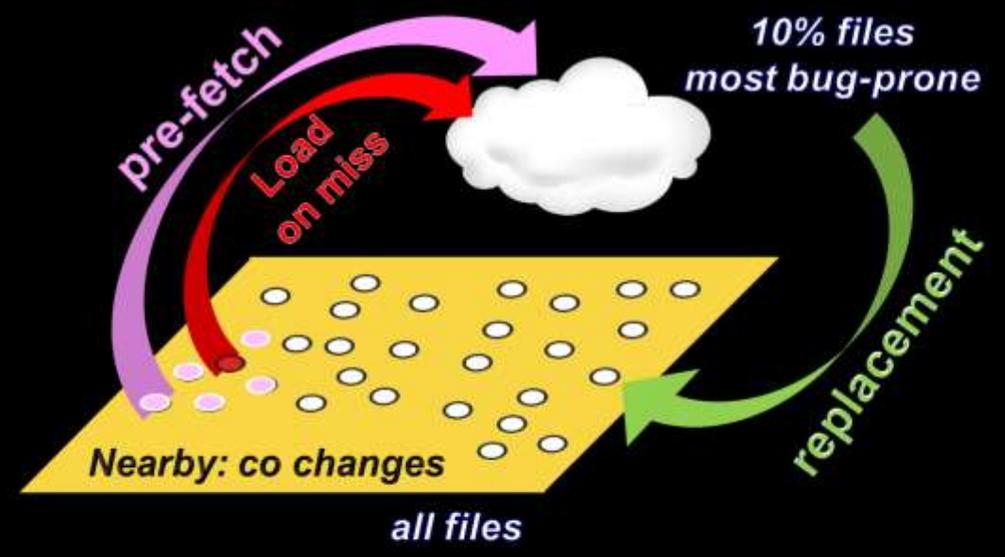
ReCrash + BugCache

- 10% of modules account for 70% of crashes
- ReCrash + BugCache can possibly
  - run with **1~6%** overhead
  - reproduce **70%** of crashes

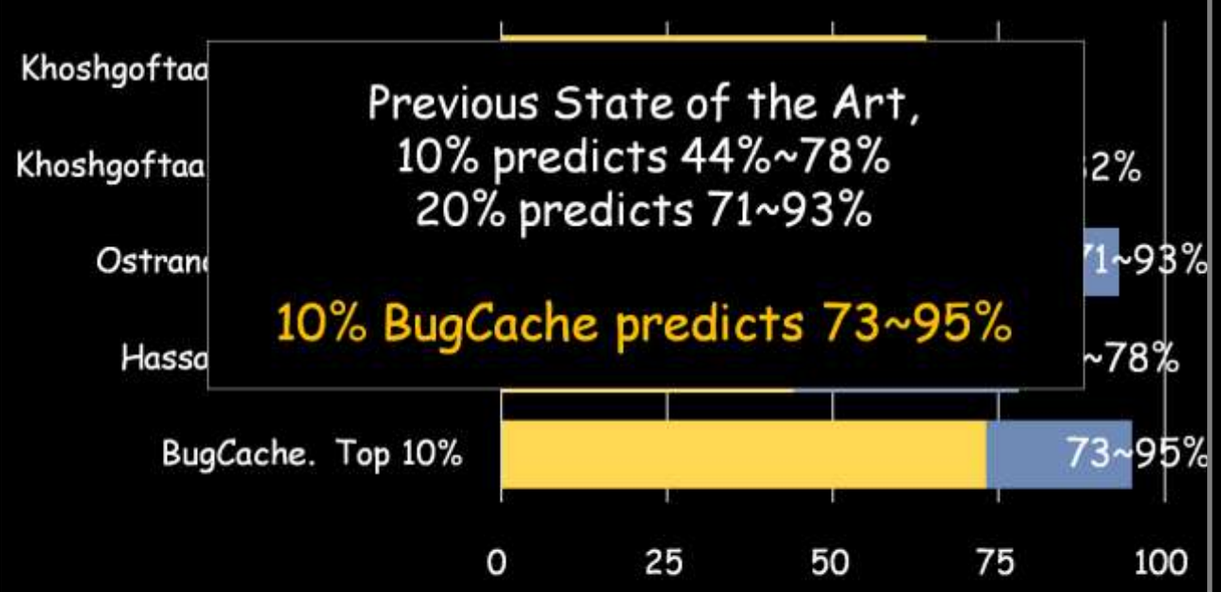
Summary

The Problem

Bug Cache Model



Related Work



ReCrash + BugCache

- ReCrash: Monitoring all modules
- BugCache: Identify crash-able modules
- ReCrash + BugCache:
  - 10% of modules account for 70% of crashes
  - ReCrash + BugCache can possibly
    - run with 1~6% overhead
    - reproduce 70% of crashes