Spencer

Mining for Safety using Interactive Trace Analysis

Stephan Brandauer, Tobias Wrigstad
http://stbr.me/spencer
sbrandauer

This Project

- Dynamic, offline analysis of executions of Java programs
- Look for safety properties of objects, classes, fields
  - immutability, uniqueness, stack boundedness
- Develop a Spencer: tool to facilitate this, and similar studies
  - “rapid prototyping” of dynamic analyses
  - you can use it!

Spencer

- We’re building http://spencer-t.racing
- Openly accessible web service
- For analysing pre-recorded program traces
- 680GB of data, 4.5E9 events, 9 program runs
- Open source (github.com/kaeluka/spencer-all)

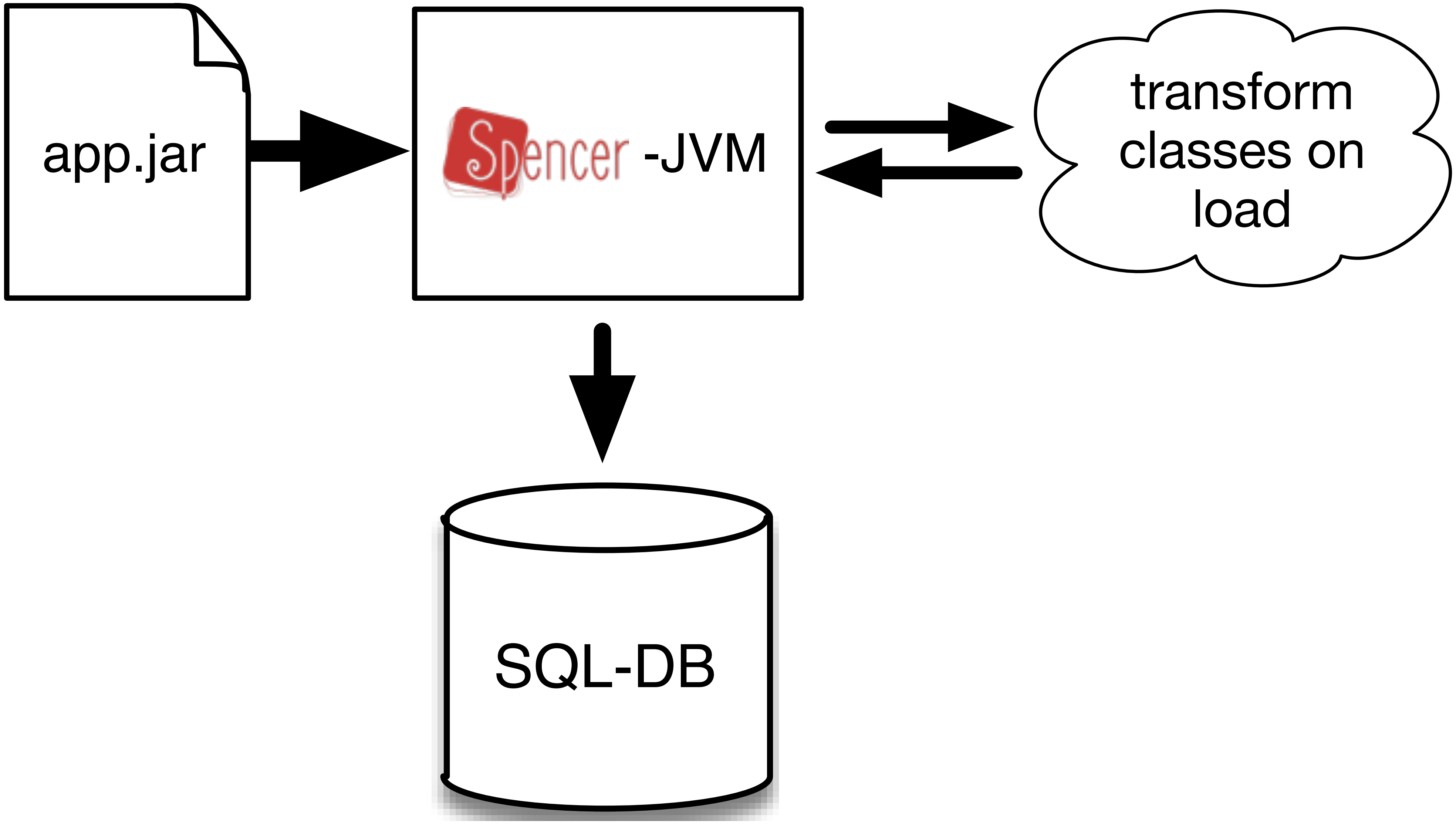
Spencer — Use Cases

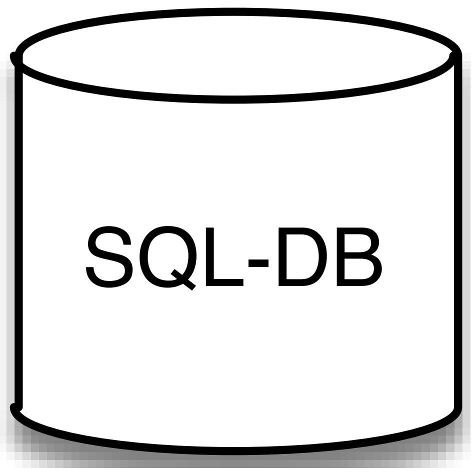
- “I’m inventing a language abstraction and want to find cases that it can’t handle well.”

- “I’m about to implement a new garbage collector and want to know whether the heaps it optimises for are common (and which programs could be problematic).”

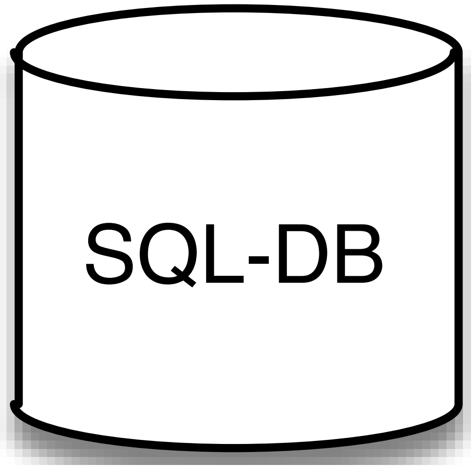
- We’re currently using the data to evaluate hypothetical computer architecture changes.

Workflow



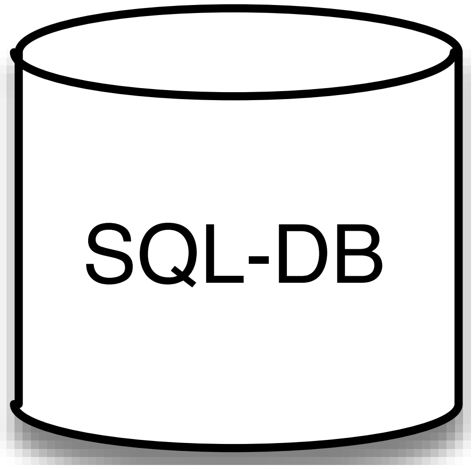


SQL-DB



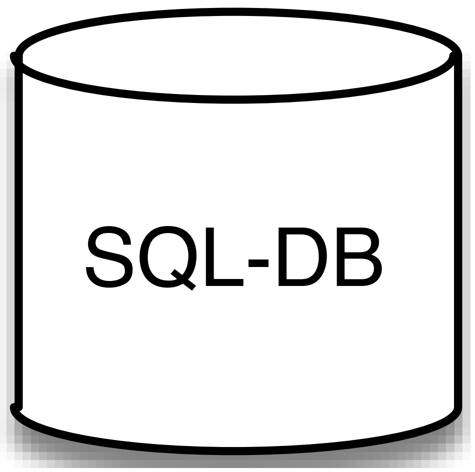
calls ✓

```
# SELECT * FROM calls WHERE callstart = 511073 ;
 caller | callee |    name    | callstart | callend |  callsitefile   | callsiteline | thread
--------+--------+------------+-----------+---------+-----------------+--------------+--------
  10530 |  10247 | startsWith |    511073 |  511091 | MetaIndex.java |          242 | main
```



calls ✓
uses ✓

```
# SELECT * FROM calls WHERE callstart = 511073 ;
 caller | callee |    name     | callstart | callend |  callsitefile   | callsiteline | thread
--------+--------+------------+-----------+---------+----------------+--------------+--------
  10530 |  10247 | startsWith |    511073 |  511091 | MetaIndex.java |          242 | main


# SELECT * FROM uses WHERE idx ⩾ 511073 AND idx ⩽ 511091 ;
 caller | callee | name  |   method   |    kind    |  idx   | thread
--------+--------+-------+------------+------------+--------+--------
  10247 |  10247 | var_1 | startsWith | varstore   | 511074 | main
  10247 |  10247 | var_1 | startsWith | varload    | 511075 | main
  … snip …
  10247 |  10247 | var_5 | startsWith | varload    | 511088 | main
  10247 |  10453 | _0    | startsWith | read       | 511089 | main
```



calls ✓
uses ✓
refs ✓

```
# SELECT * FROM calls WHERE callstart = 511073 ;
 caller | callee |    name    | callstart | callend |  callsitefile  | callsiteline | thread
--------+--------+------------+-----------+---------+----------------+--------------+--------
  10530 |  10247 | startsWith |    511073 |  511091 | MetaIndex.java |          242 | main


# SELECT * FROM uses WHERE idx ⩾ 511073 AND idx ⩽ 511091 ;
 caller | callee | name  |   method   |    kind    |  idx   | thread
--------+--------+-------+------------+------------+--------+--------
  10247 |  10247 | var_1 | startsWith | varstore   | 511074 | main
  10247 |  10247 | var_1 | startsWith | varload    | 511075 | main
  … snip …
  10247 |  10247 | var_5 | startsWith | varload    | 511088 | main
  10247 |  10453 | _0    | startsWith | read       | 511089 | main


# SELECT * FROM refs WHERE caller = 10247 AND kind = 'field' ;
 caller | callee | kind  | name  | refstart | refend | thread
--------+--------+-------+-------+----------+--------+--------
  10247 |  10248 | field | value |   421877 |        | main
```

Spencer DSL

- Object selections are single expressions
- Compiled to SQL queries
- Simplicity for Expressivity Tradeoff

Spencer DSL

ImmutableObj() ↔

```
SELECT id FROM objects WHERE id > 4
EXCEPT
  (SELECT DISTINCT callee AS id
   FROM uses_cstore
   WHERE callee > 4
   AND   NOT(caller = callee AND method = ‘<init>')
   AND   (kind = 'fieldstore' OR kind = 'modify'))
```

Spencer DSL

StackBoundObj()

```
SELECT id
FROM   objects
WHERE  id > 4
AND    NOT EXISTS (
         SELECT 1
         FROM   refs
         WHERE  refs.callee = objects.id
         AND    refs.kind = 'field'
       )
```

Spencer DSL

HeapUniqueObj() ↔

```
SELECT callee AS id FROM
(SELECT callee, time, SUM(delta) OVER(PARTITION BY
callee ORDER BY time) AS sum_at_time
 FROM (
   (SELECT
      callee, refstart AS time, 1 AS delta
    FROM refs
    WHERE callee > 4 AND kind = 'field') UNION ALL
(SELECT
      callee, refend AS time, -1 AS delta
    FROM refs
    WHERE callee > 4 AND kind = 'field')
 ) AS steps) AS integrated_steps
GROUP BY callee
HAVING MAX(sum_at_time) = 1
```

Spencer DSL: Composing Queries

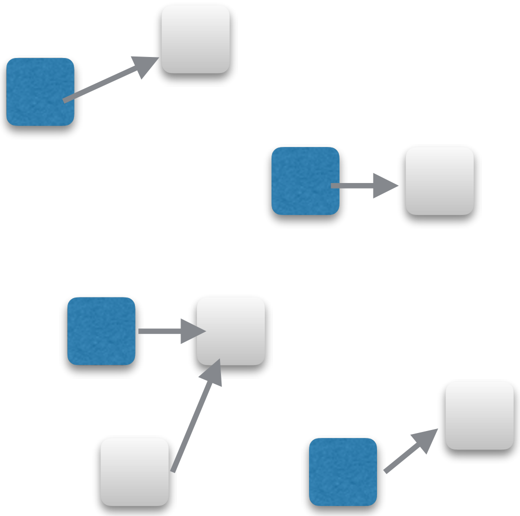
InstanceOf(java.lang.String)

```
SELECT id FROM objects WHERE klass = 'java.lang.String'
```

Spencer DSL: Composing Queries

InstanceOf(java.lang.String) ↔

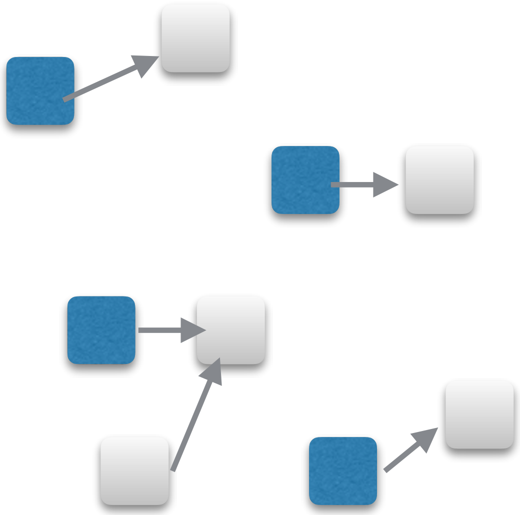
```
SELECT id FROM objects WHERE klass = 'java.lang.String'
```



Spencer DSL

```
HeapReferredFrom(
  InstanceOf(java.lang.String))
```

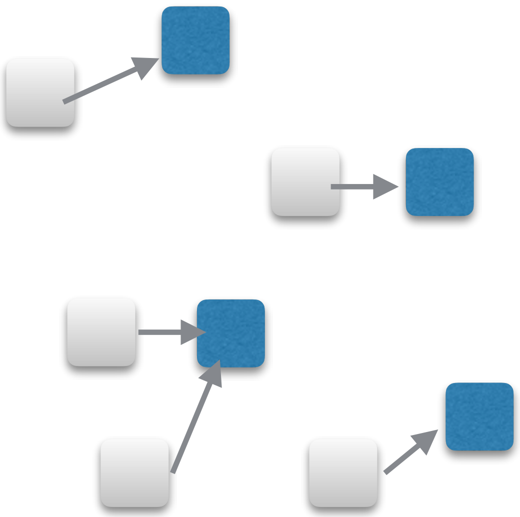
```
SELECT callee AS id
FROM   refs
WHERE  kind = 'field'
AND    caller IN (
  SELECT id FROM objects WHERE klass = 'java.lang.String'
)
```



Spencer DSL

HeapReferredFrom(
  InstanceOf(java.lang.String))

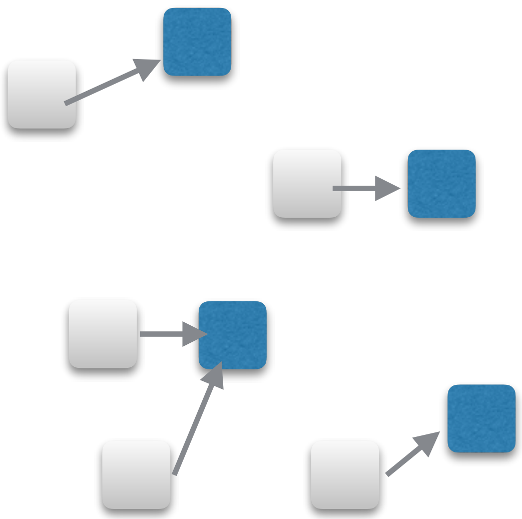
```
SELECT callee AS id
FROM   refs
WHERE  kind = 'field'
AND    caller IN (
  SELECT id FROM objects WHERE klass = 'java.lang.String'
)
```



Spencer DSL

```
And(
 HeapReferredFrom(
  InstanceOf(java.lang.String))
 ?)
```

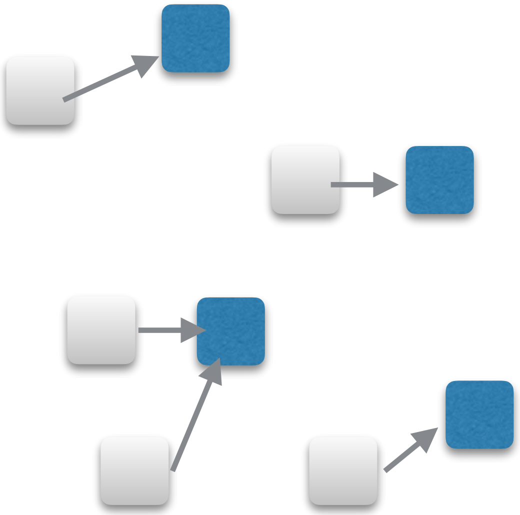
```
 (
  SELECT callee AS id
  FROM   refs
  WHERE  kind = 'field'
  AND    caller IN (
   SELECT id FROM objects WHERE klass = 'java.lang.String'
  )
 ) INTERSECT (


   ?




 )
)
```



Spencer DSL

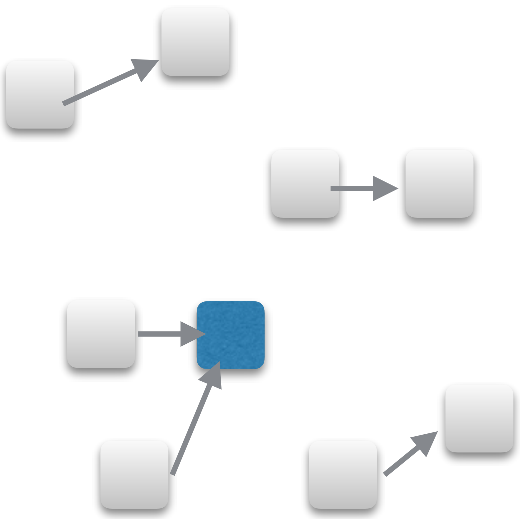
```
And(
  HeapReferredFrom(
    InstanceOf(java.lang.String))
  Not(HeapUniqueObj()))
```

```
 (
  SELECT callee AS id
  FROM   refs
  WHERE  kind = 'field'
  AND    caller IN (
   SELECT id FROM objects WHERE klass = 'java.lang.String'
  )
 ) INTERSECT (
  SELECT id FROM objects WHERE id > 4
  EXCEPT
   (SELECT callee AS id FROM
    (SELECT callee, time, SUM(delta) OVER(PARTITION BY callee ORDER BY time) AS sum_at_time
     FROM (
      (SELECT
        callee, refstart AS time, 1 AS delta
       FROM refs
       WHERE callee > 4 AND kind = 'field') UNION ALL (SELECT
        callee, refend AS time, -1 AS delta
       FROM refs
       WHERE callee > 4 AND kind = 'field')
     ) AS steps) AS integrated_steps
    GROUP BY callee
    HAVING MAX(sum_at_time) = 1)

 )
)
```



Spencer DSL

```
And(
  HeapReferredFrom(
    InstanceOf(java.lang.String))
  Not(HeapUniqueObj()))
```

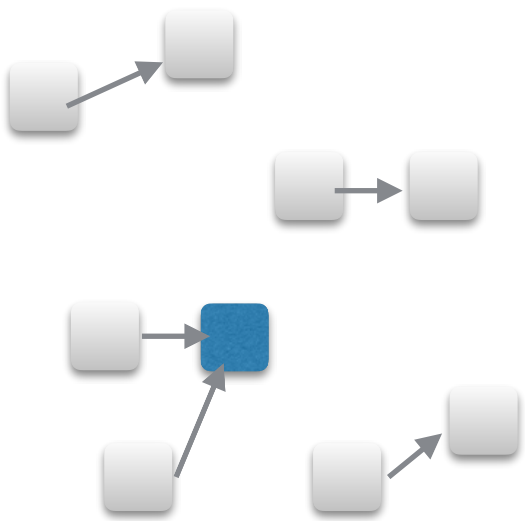


```
 (
  SELECT callee AS id
  FROM   refs
  WHERE  kind = 'field'
  AND    caller IN (
   SELECT id FROM objects WHERE klass = 'java.lang.String'
  )
 ) INTERSECT (
  SELECT id FROM objects WHERE id > 4
  EXCEPT
   (SELECT callee AS id FROM
    (SELECT callee, time, SUM(delta) OVER(PARTITION BY callee ORDER BY time) AS sum_at_time
     FROM (
      (SELECT
        callee, refstart AS time, 1 AS delta
       FROM refs
       WHERE callee > 4 AND kind = 'field') UNION ALL (SELECT
        callee, refend AS time, -1 AS delta
       FROM refs
       WHERE callee > 4 AND kind = 'field')
     ) AS steps) AS integrated_steps
    GROUP BY callee
    HAVING MAX(sum_at_time) = 1)

 )
)
```

Spencer DSL

```
HeapRefersTo(
  And(
    HeapReferredFrom(
      InstanceOf(java.lang.String))
    Not(HeapUniqueObj())))
```

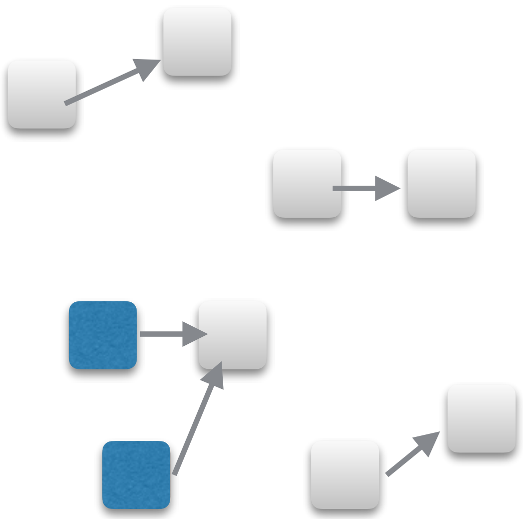
```
SELECT caller AS id
FROM   refs
WHERE  kind = 'field'
AND    callee IN (
  (
    SELECT callee AS id
    FROM   refs
    WHERE  kind = 'field'
    AND    caller IN (
      SELECT id FROM objects WHERE klass = 'java.lang.String'
    )
  ) INTERSECT (
    SELECT id FROM objects WHERE id > 4
    EXCEPT
      (SELECT callee AS id FROM
       (SELECT callee, time, SUM(delta) OVER(PARTITION BY callee ORDER BY time) AS sum_at_time
        FROM (
          (SELECT
             callee, refstart AS time, 1 AS delta
           FROM refs
           WHERE callee > 4 AND kind = 'field') UNION ALL (SELECT
             callee, refend AS time, -1 AS delta
           FROM refs
           WHERE callee > 4 AND kind = 'field')
        ) AS steps) AS integrated_steps
       GROUP BY callee
       HAVING MAX(sum_at_time) = 1)

  )
)
```



Spencer DSL

```
HeapRefersTo(
  And(
    HeapReferredFrom(
      InstanceOf(java.lang.String))
    Not(HeapUniqueObj())))
```

```
SELECT caller AS id
FROM   refs
WHERE  kind = 'field'
AND    callee IN (
  (
    SELECT callee AS id
    FROM   refs
    WHERE  kind = 'field'
    AND    caller IN (
      SELECT id FROM objects WHERE klass = 'java.lang.String'
    )
  ) INTERSECT (
    SELECT id FROM objects WHERE id > 4
    EXCEPT
      (SELECT callee AS id FROM
       (SELECT callee, time, SUM(delta) OVER(PARTITION BY callee ORDER BY time) AS sum_at_time
        FROM (
          (SELECT
             callee, refstart AS time, 1 AS delta
           FROM refs
           WHERE callee > 4 AND kind = 'field') UNION ALL (SELECT
             callee, refend AS time, -1 AS delta
           FROM refs
           WHERE callee > 4 AND kind = 'field')
        ) AS steps) AS integrated_steps
       GROUP BY callee
       HAVING MAX(sum_at_time) = 1)

  )
)
```



Spencer DSL

```
HeapRefersTo(
 And(
  HeapReferredFrom(
   InstanceOf(java.lang.String))
  Not(HeapUniqueObj())))
```

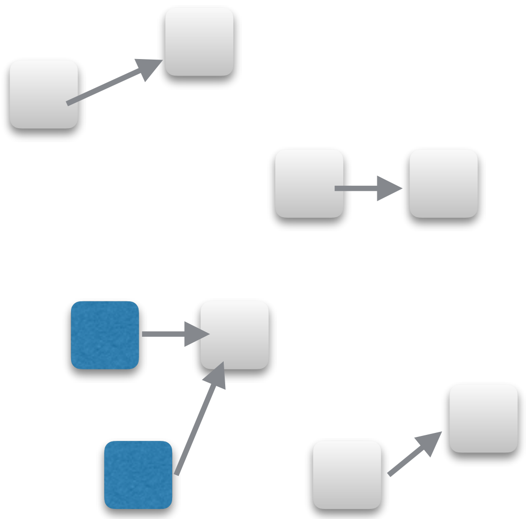
```
SELECT caller AS id
FROM   refs
WHERE  kind = 'field'
AND    callee IN (
* (
    SELECT callee AS id
    FROM   refs
    WHERE  kind = 'field'
    AND    caller IN (
      SELECT id FROM objects WHERE klass = 'java.lang.String'
    )
  ) INTERSECT (
    SELECT id FROM objects WHERE id > 4
    EXCEPT
      (SELECT callee AS id FROM
       (SELECT callee, time, SUM(delta) OVER(PARTITION BY callee ORDER BY time) AS sum_at_time
        FROM (
          (SELECT
             callee, refstart AS time, 1 AS delta
           FROM refs
           WHERE callee > 4 AND kind = 'field') UNION ALL (SELECT
             callee, refend AS time, -1 AS delta
           FROM refs
           WHERE callee > 4 AND kind = 'field')
        ) AS steps) AS integrated_steps
       GROUP BY callee
       HAVING MAX(sum_at_time) = 1)

  )
)
```

* and caching of subexpressions

Spencer DSL

```
HeapRefersTo(
 And(
  HeapReferredFrom(
   InstanceOf(java.lang.String))
  Not(HeapUniqueObj())))
```

```
SELECT caller AS id
FROM   refs
WHERE  kind = 'field'
AND    callee IN (
*(
  SELECT callee AS id
  FROM   refs
  WHERE  kind = 'field'
  AND    caller IN (
   SELECT id FROM objects WHERE klass = 'java.lang.String'
  )
 ) INTERSECT (
  SELECT id FROM objects WHERE id > 4
  EXCEPT
   (SELECT callee AS id FROM
    (SELECT callee, time, SUM(delta) OVER(PARTITION BY callee ORDER BY time) AS sum_at_time
     FROM (
      (SELECT
         callee, refstart AS time, 1 AS delta
       FROM refs
       WHERE callee > 4 AND kind = 'field') UNION ALL (SELECT
         callee, refend AS time, -1 AS delta
       FROM refs
       WHERE callee > 4 AND kind = 'field')
     ) AS steps) AS integrated_steps
    GROUP BY callee
    HAVING MAX(sum_at_time) = 1)

 )
)
```



* and caching of subexpressions

Dynamic Analysis	Static Analysis
false positives (“upper bound”)	false negatives (“lower bound”)
often-used code weighed stronger	all code weighed equally
easily deals with runtime code generation, dynamic code loading	easily can produce sound claims

The Study

“Safety”

unique	*at most one variable/field refers to object at a time*
stack bound	*no field ever refers to the object*
heap-unique	*at most one field refers to object at a time*
deeply immutable	*shallow immutable + can only reach (via fields) other shallow immutable objects*
shallow immutable	*object never changed outside of constructor*
safe	*at least one of the above*

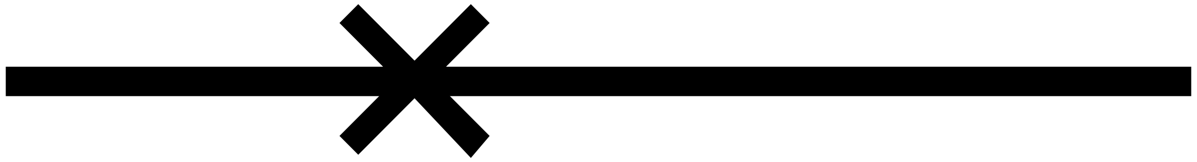
Dynamic Analysis

Static Analysis

“What proportion of objects are safe?”

Dynamic Analysis

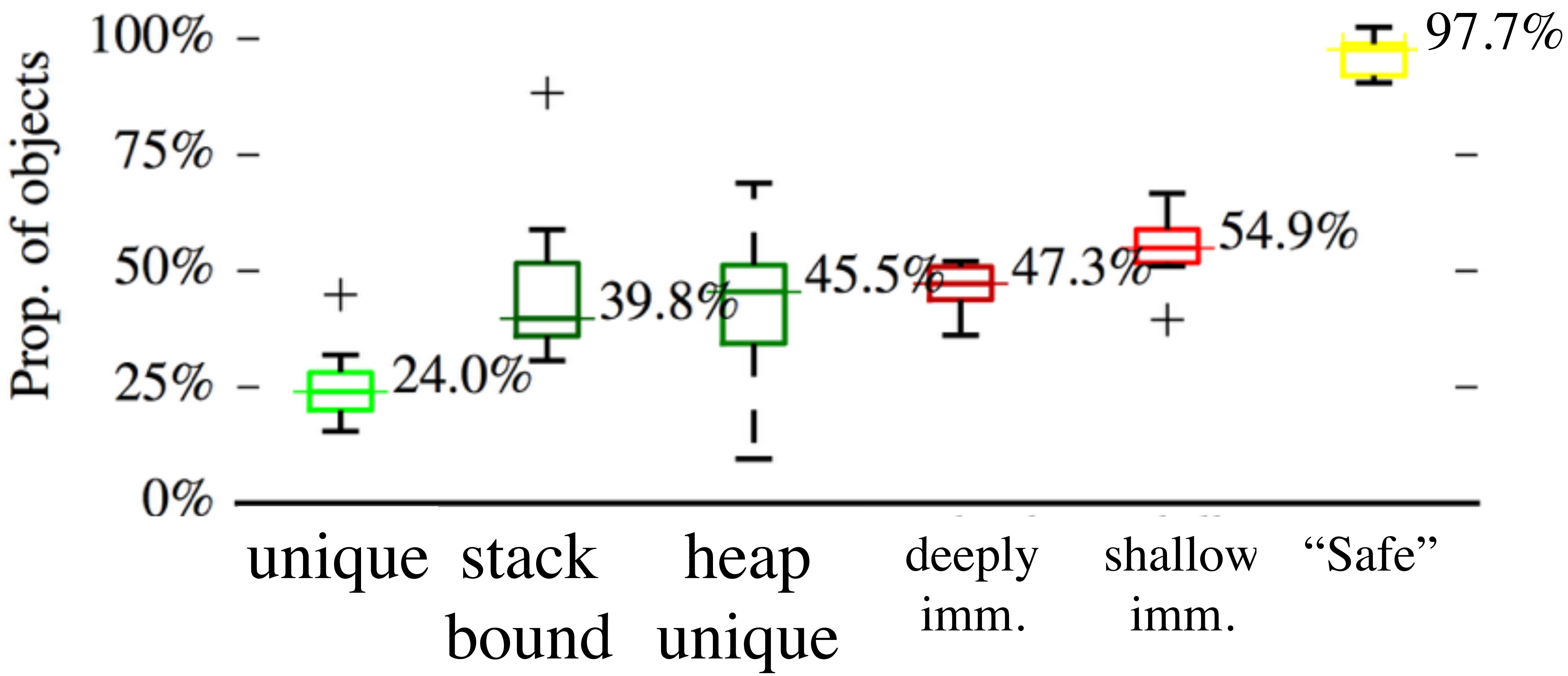
Static Analysis

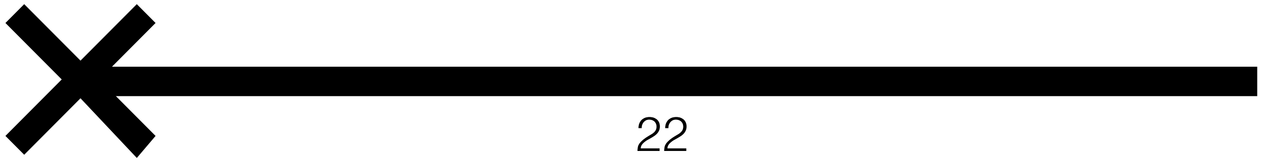


“What proportion of classes only produce safe instances?”

“What proportion of fields only contain safe instances?”

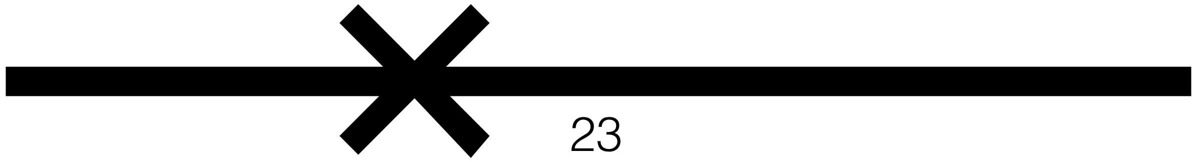
Per Object Analysis





Per Class Analysis

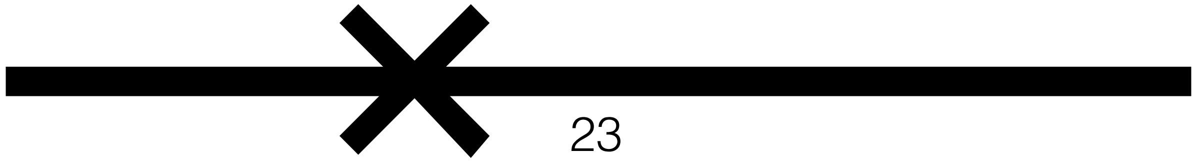
Out of all classes with more than 10 instances, how many classes…



Per Class Analysis

Out of all classes with more than 10 instances,
how many classes…

1) … have ONLY instances that fulfil a safety property?



Per Class Analysis

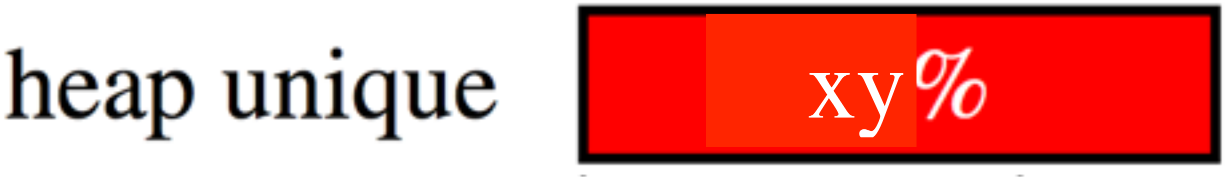
Out of all classes with more than 10 instances,
how many classes…

1) … have ONLY instances that fulfil a safety property?

2) … have NO instances that fulfil a safety property?

Per Class Analysis

Per Class Analysis



Classes with NO
heap-unique instances

Per Class Analysis

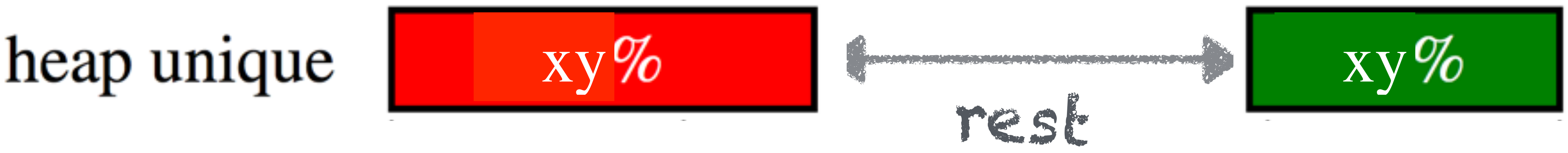
Classes with ONLY
heap-unique instances

heap unique

Classes with NO
heap-unique instances

Per Class Analysis

Classes with ONLY heap-unique instances



heap unique

xy%

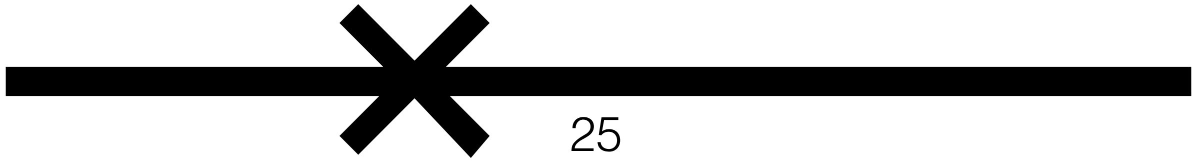
rest

xy%

Classes with NO heap-unique instances

Per Class Analysis

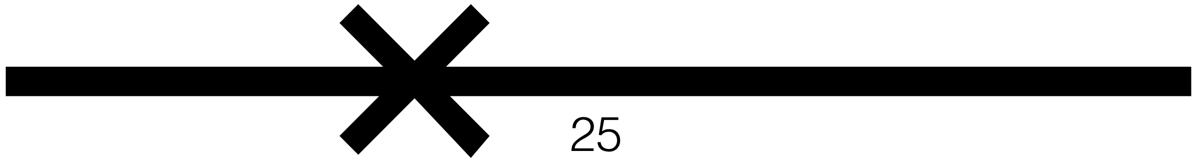
heap unique



Per Class Analysis

heap unique
xy%
xy%

Hypothesis: could annotate class with "heap-shared" keyword



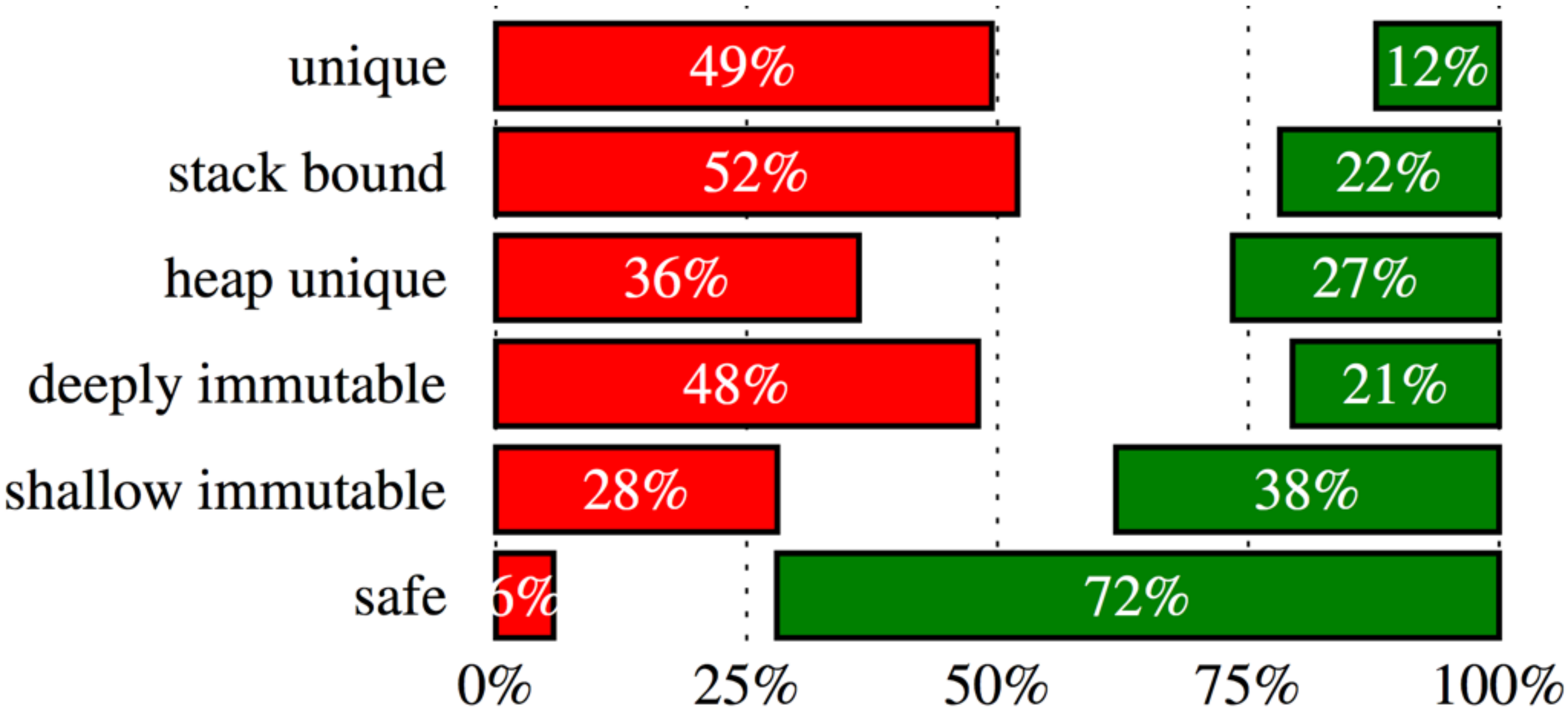
Per Class Analysis

Hypothesis: could annotate class with "heap-unique" keyword

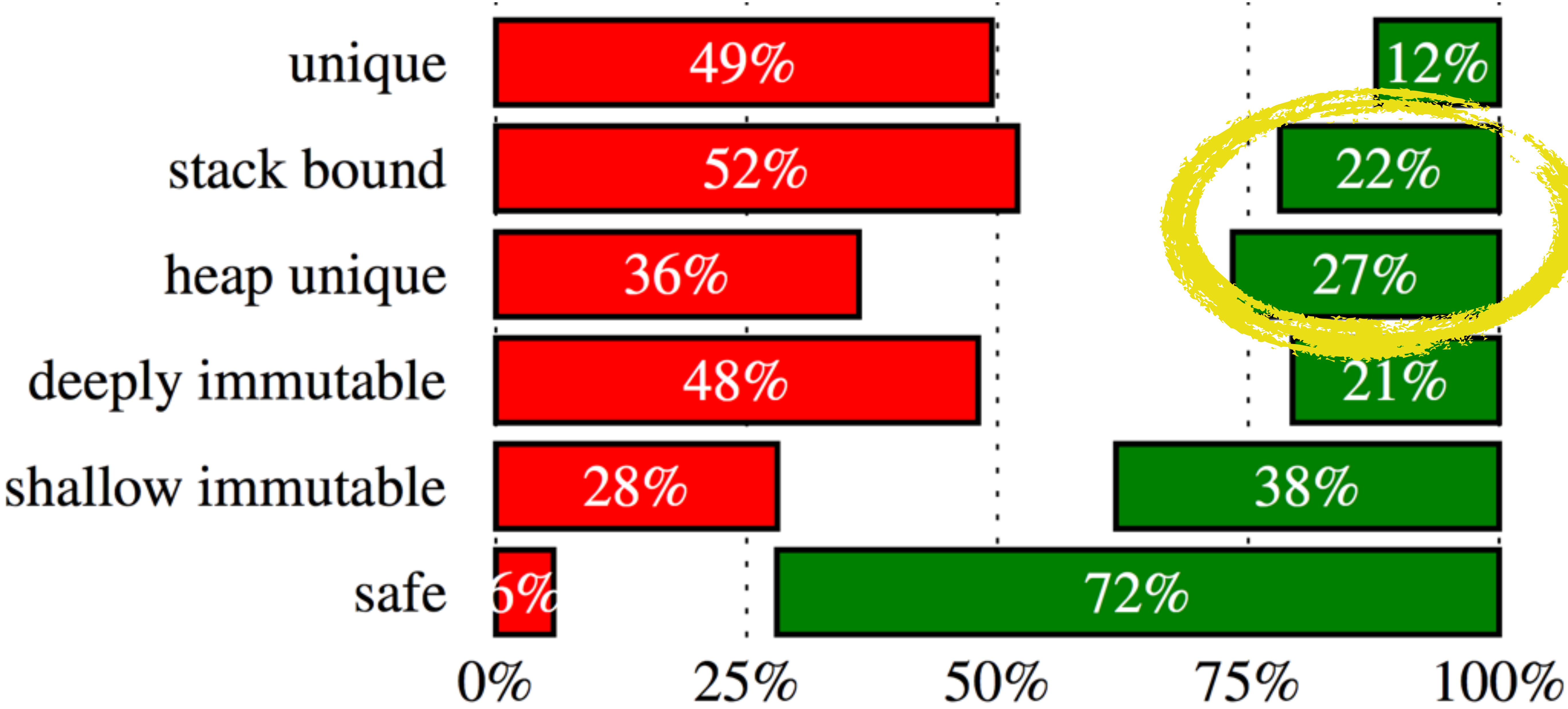
heap unique
xy%
xy%

Hypothesis: could annotate class with "heap-shared" keyword

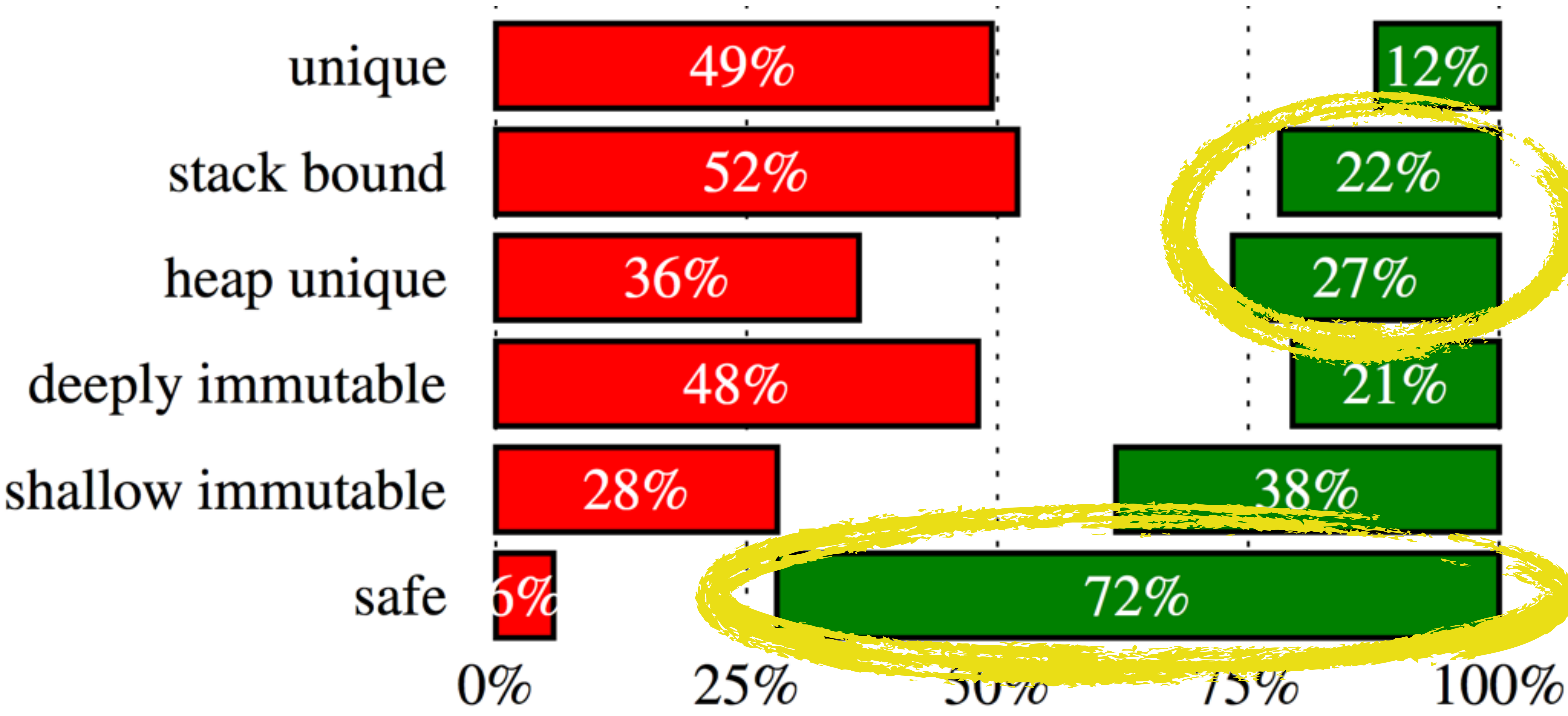
Per Class Analysis



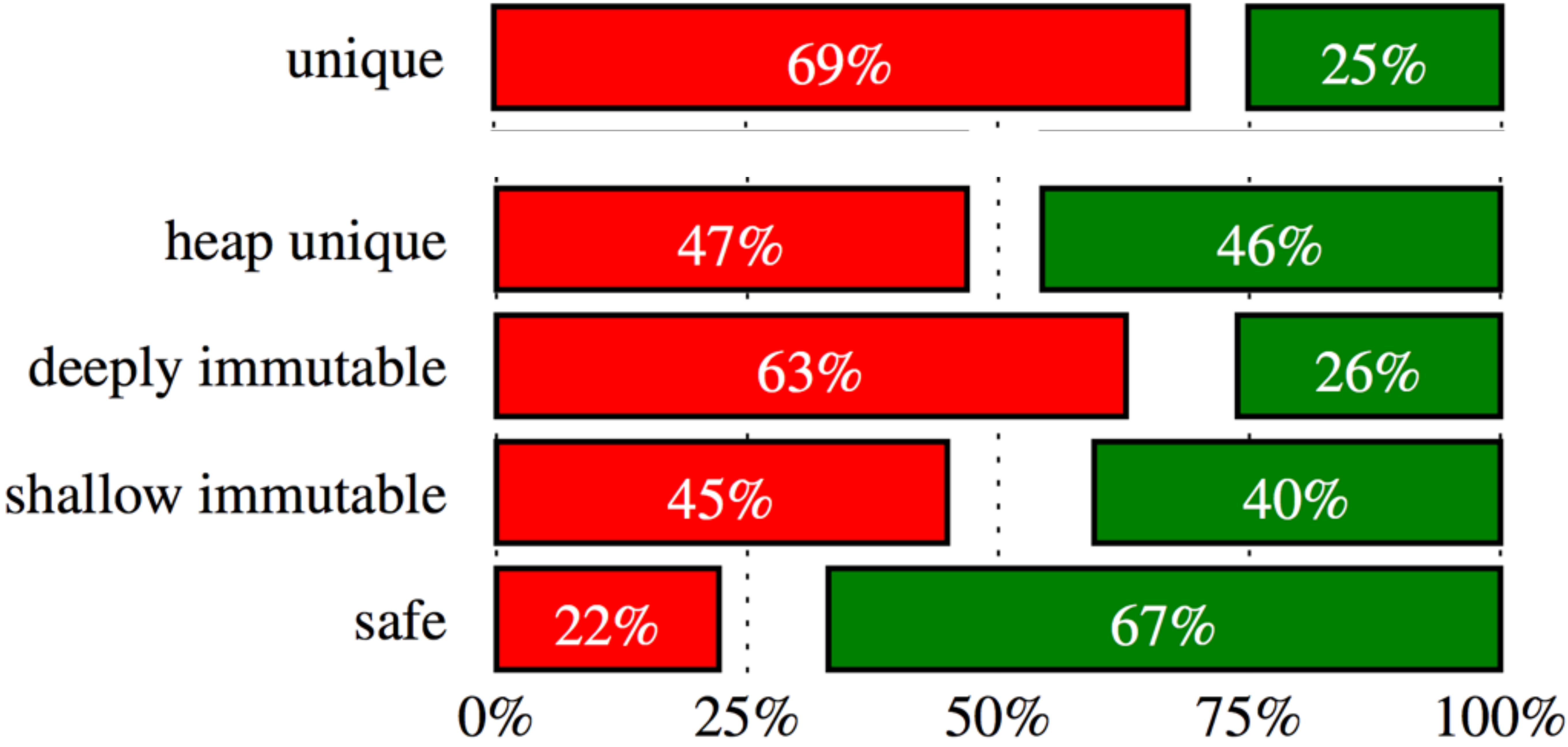
Per Class Analysis



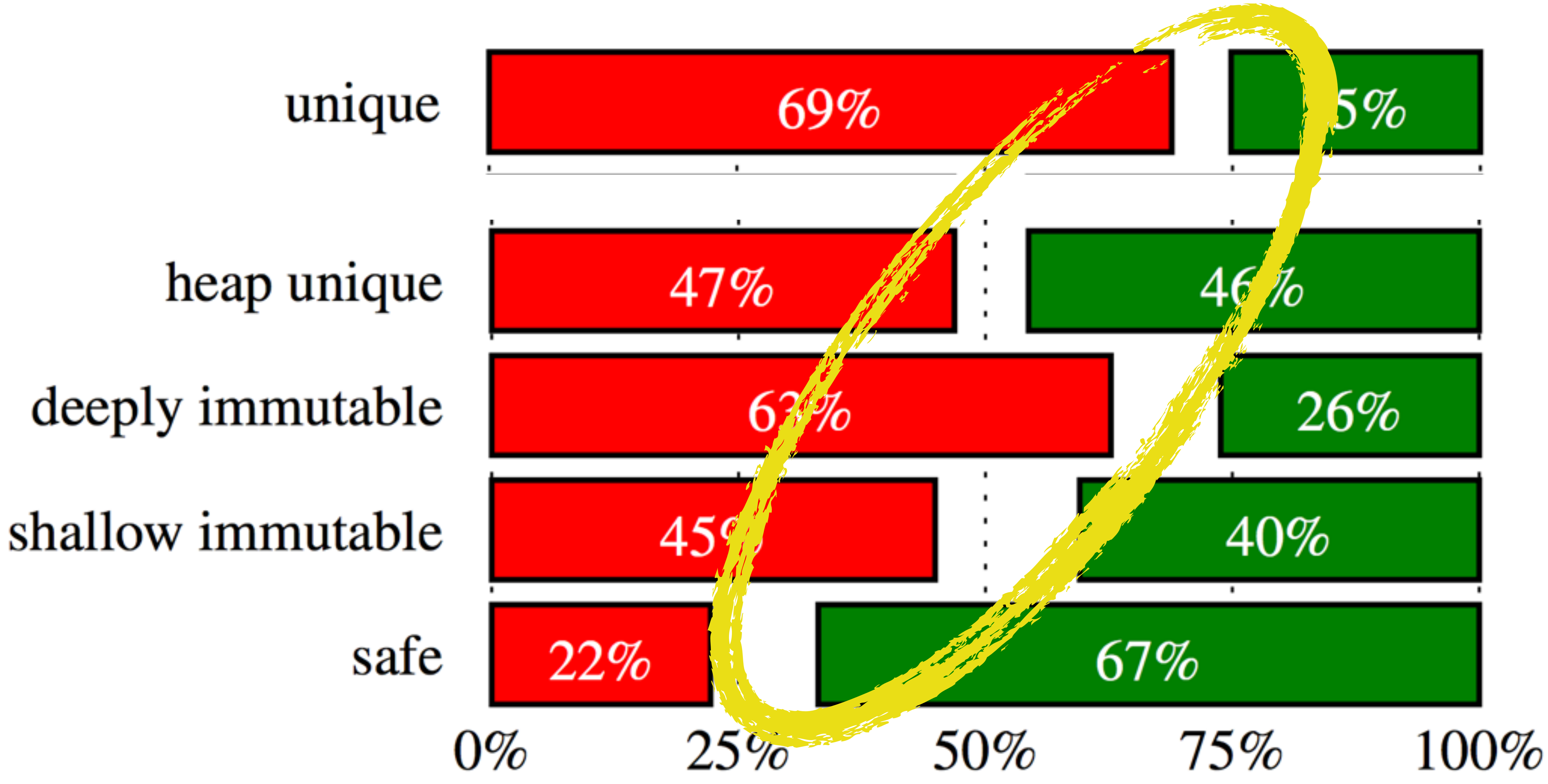
Per Class Analysis



Per Field Analysis



Per Field Analysis



Further Work/Research

- Put the data to good use!
    - What are the classes/fields in the white gaps? Do they provide different invariants?
- Trace more programs!
    - Also in different languages!

Spencer

Stephan Brandauer, Tobias Wrigstad

http://stbr.me/spencer

🐦sbrandauer