Performance Diagnosis for Inefficient Loops

Linhai Song[1] and Shan Lu[2]

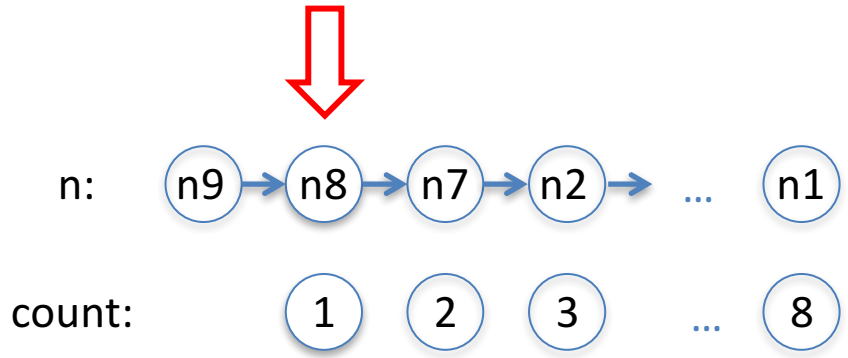
[1]FireEye, Inc.

[2]University of Chicago

What are Performance Problems?

- Definition of Performance Problems (PPs):
  - Implementation mistakes causing inefficiency
- An example

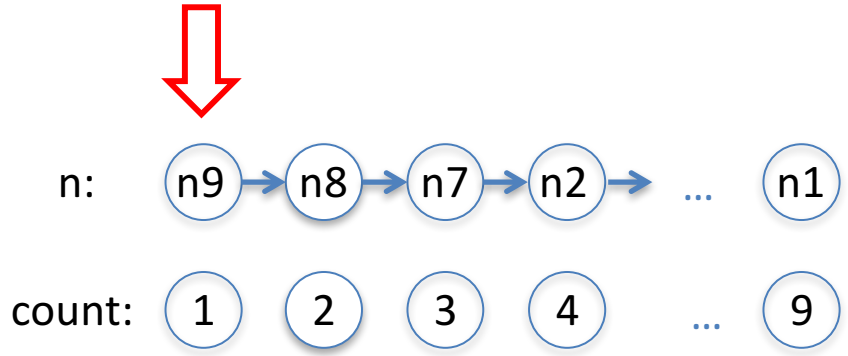
```
char * ssh_xph_generate(node_t * aNode)
{
   int count = 0;
   for (n = aNode; n; n = aNode->prev)
      if(n->name == aNode->name)
         count ++;
   return count;
} //Mozilla#477564
```



What are Performance Problems?

- Definition of Performance Problems (PPs):
  - Implementation mistakes causing inefficiency
- An example

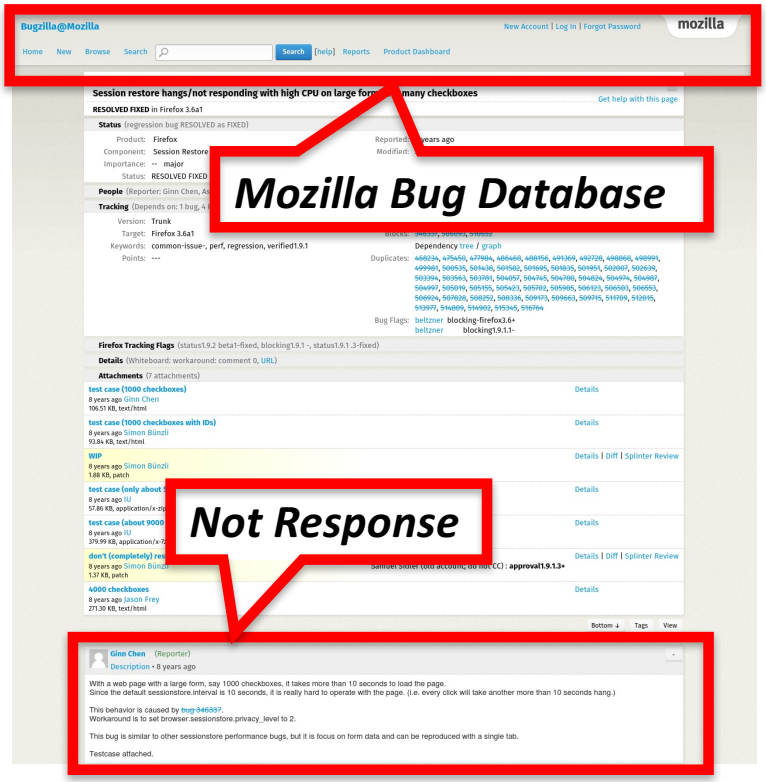
```
char * ssh_xph_generate(node_t * aNode)
{
   int count = 0;
   for (n = aNode; n; n = aNode->prev)
      if(n->name == aNode->name)
         count ++;
   return count;
} //Mozilla#477564
```



What are Performance Problems?

- Definition of Performance Problems (PPs):
  - Implementation mistakes causing inefficiency
- An example

```
char * ssh_xph_generate(node_t * aNode)
{
    int count = 0;
    for (n = aNode; n; n = aNode->prev)
        if(n->name == aNode->name)
            count ++;
    return count;
} //Mozilla#477564
```

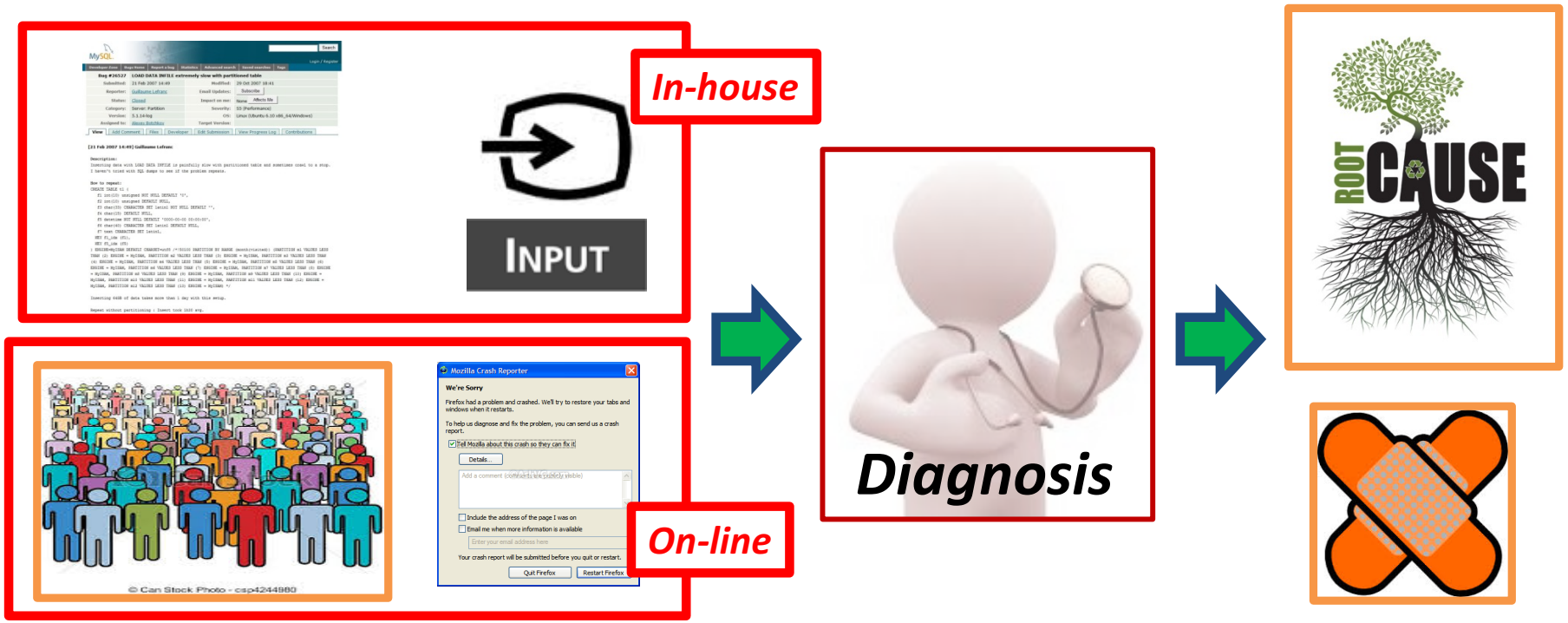


Fighting PPs is Important

- PPs widely exist in production run software
  - 5 to 60 Mozilla PPs are fixed each month
- PPs are getting more important
  - Hardware is not getting faster (per-core)
  - Software is getting more complex
  - Energy saving is getting more urgent

Performance Diagnosis

- Identifying the causes of inefficient execution
  - Different from testing & bug detection
  - In-house and on-line diagnosis

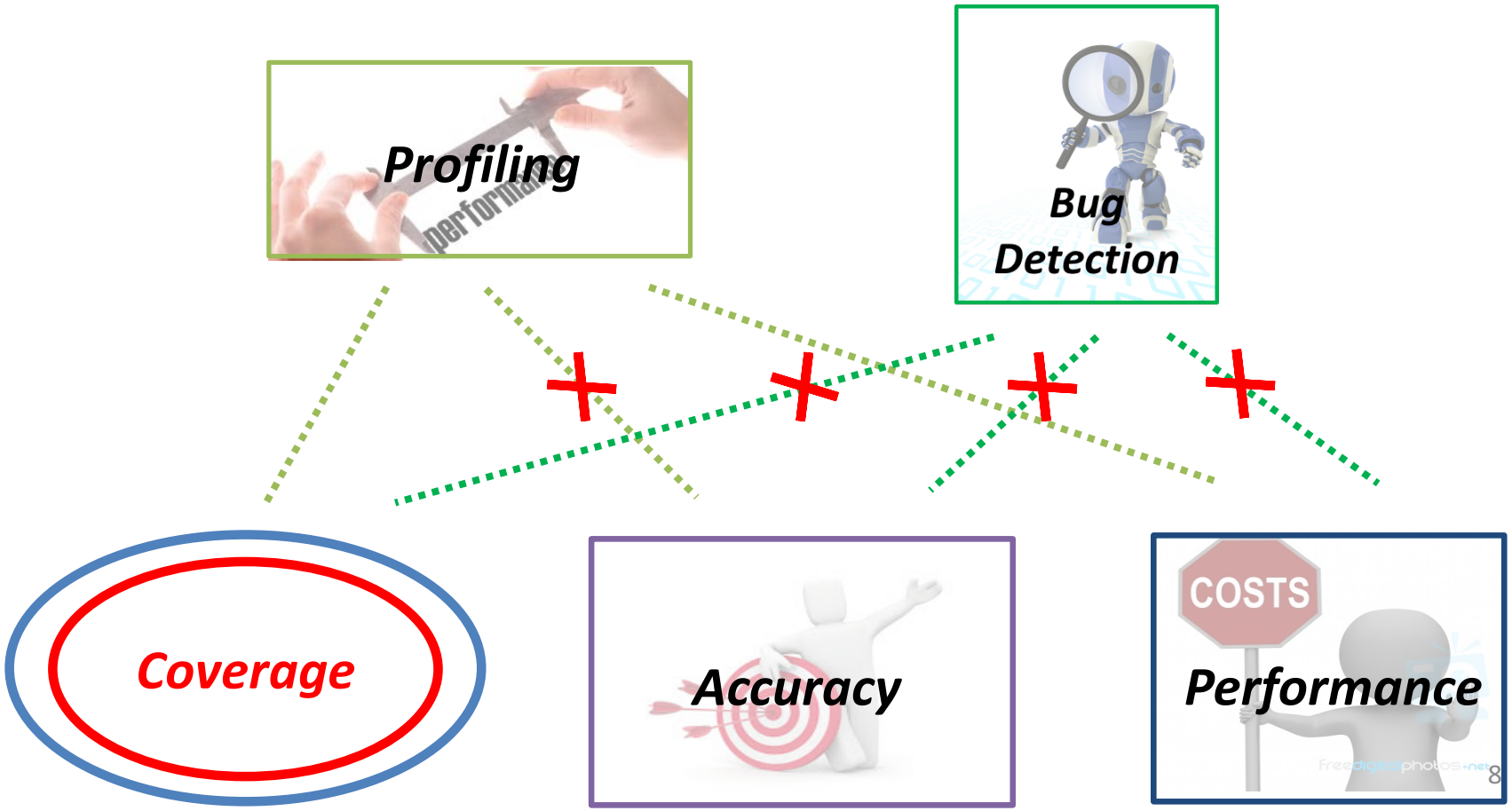


Diagnosing PPs is Challenging

- Three criteria for diagnosis tools
  - Coverage: handle a good portion of PPs
  - Accuracy: accurately tell
    - Which code regions are inefficient
    - Why they are inefficient
  - Performance: low run-time overhead

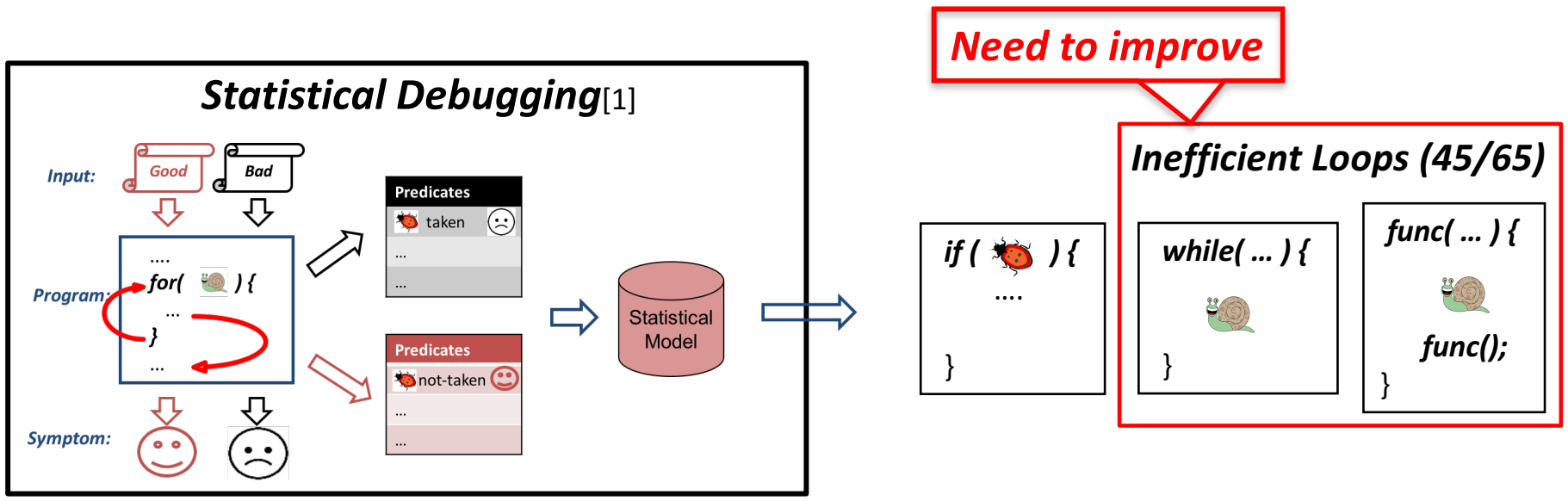
State of the Art

- No existing tools can satisfy the three criteria



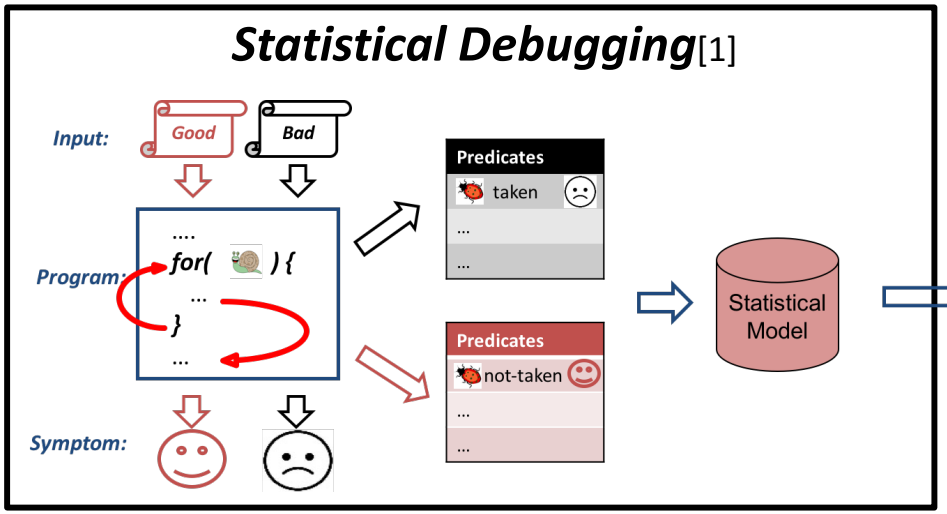
Statistical Debugging

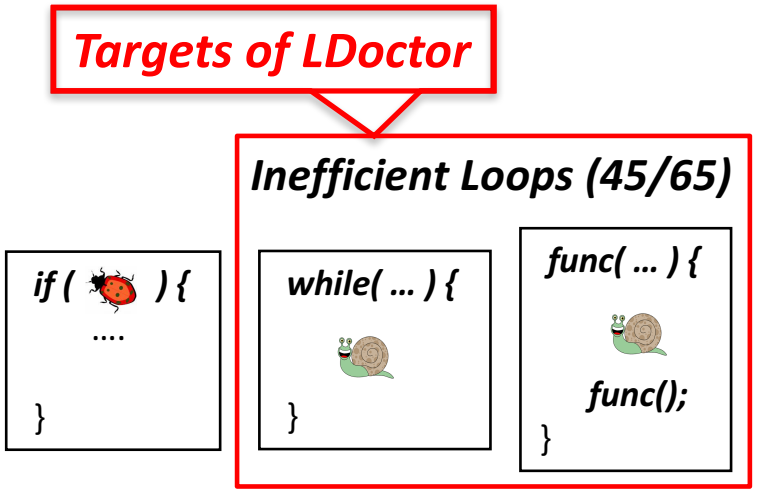
[1] Linhai Song, Shan Lu. Statistical Debugging for Real-World Performance Problems. In OOPSLA'2014.



LDoctor

- A tool(kit) targets inefficient loops





Contributions

- A root-cause taxonomy for inefficient loops
  - Guide the design of automated diagnosis tools
  - Resultless vs. redundant
- LDoctor: a toolkit to diagnose inefficient loops
  - Hybridize static and dynamic analysis
  - Identify inefficiency root causes
    - With high coverage, high accuracy, and low overhead

Outline

- Overview
- Root-cause Taxonomy
- LDoctor Design
- Evaluation of LDoctor
- Conclusion

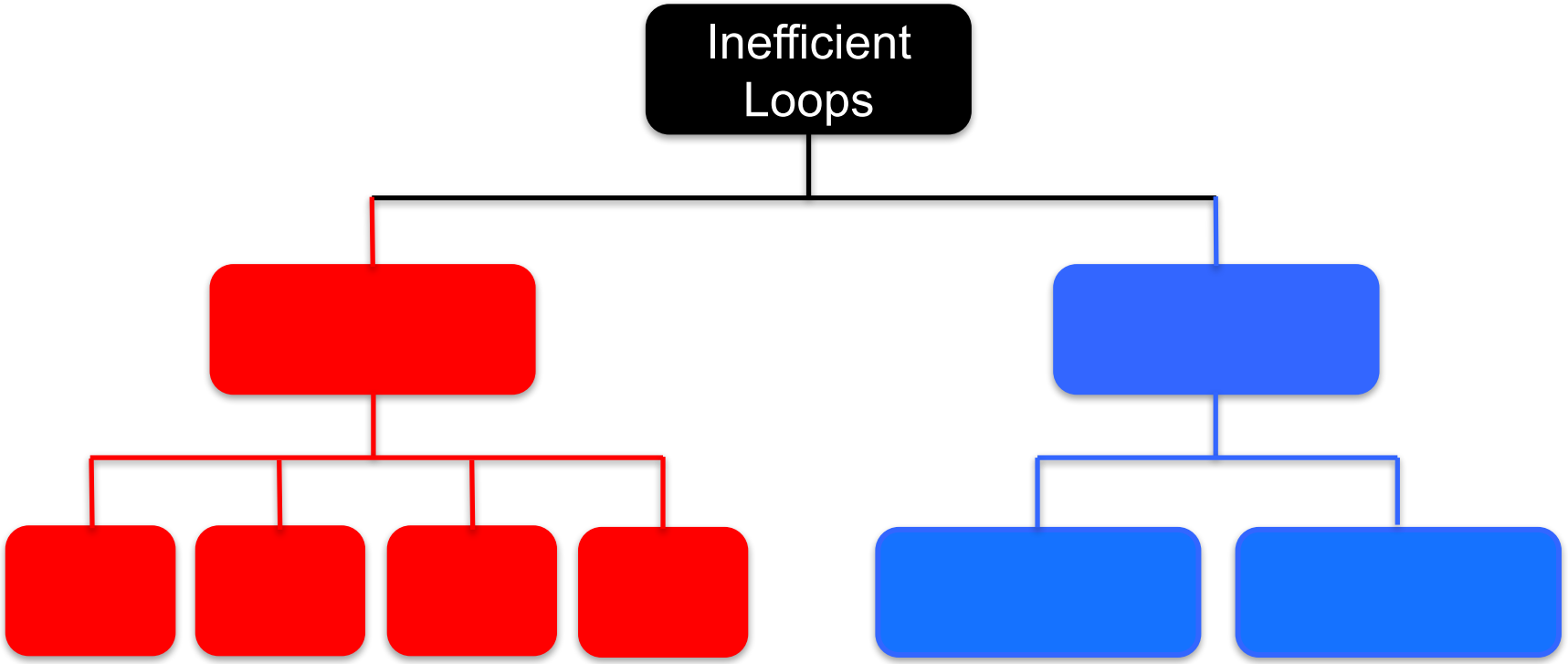
Outline

- Overview
- Root-cause Taxonomy
- LDoctor Design
- Evaluation of LDoctor
- Conclusion

What Type of Taxonomy We Need?

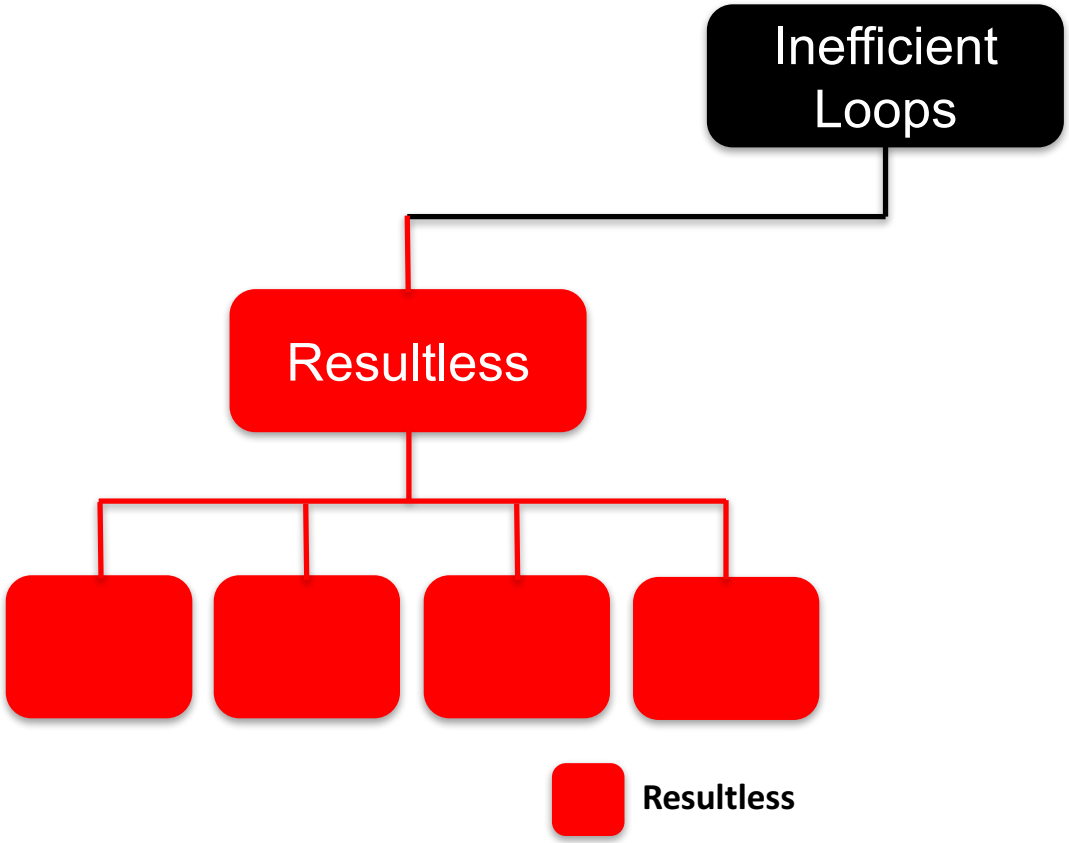
- Inclusive: covering most/all inefficient loops
- General: Not application specific
- Actionable: helpful to design fix

Root-Cause Taxonomy



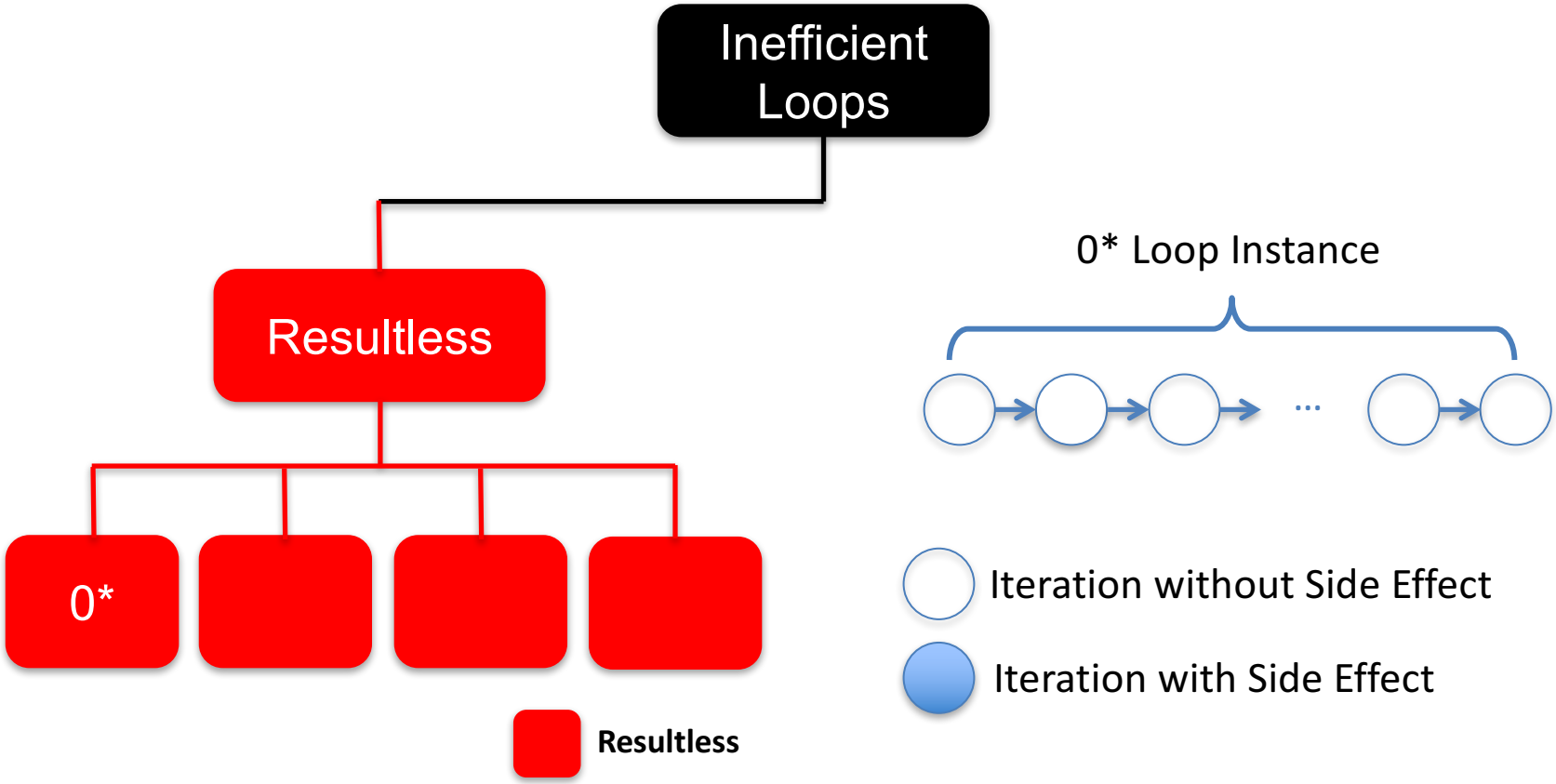
Resultless Loops

- Produce no side effects



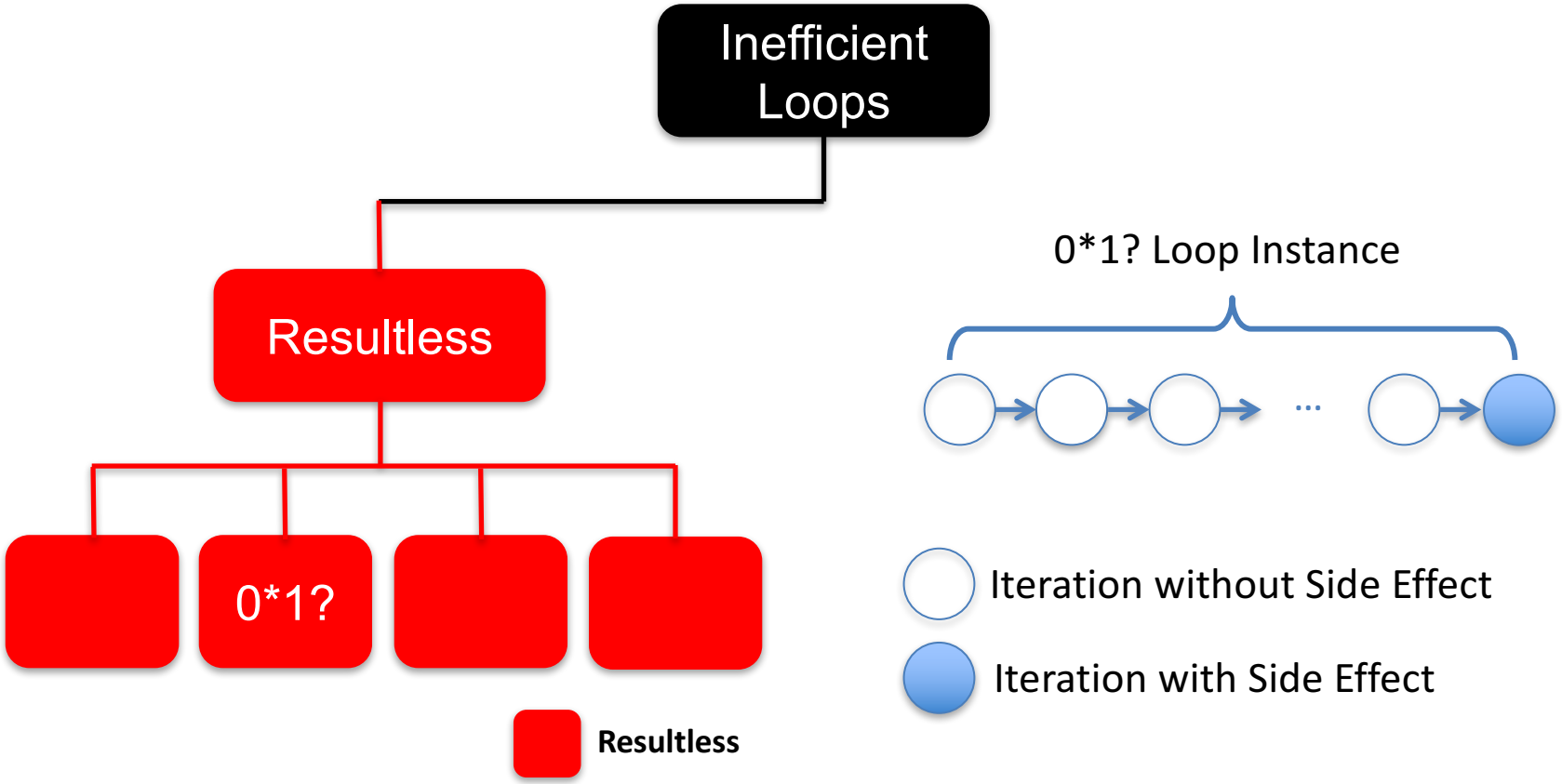
Resultless 0*

- Never produce results in any iteration



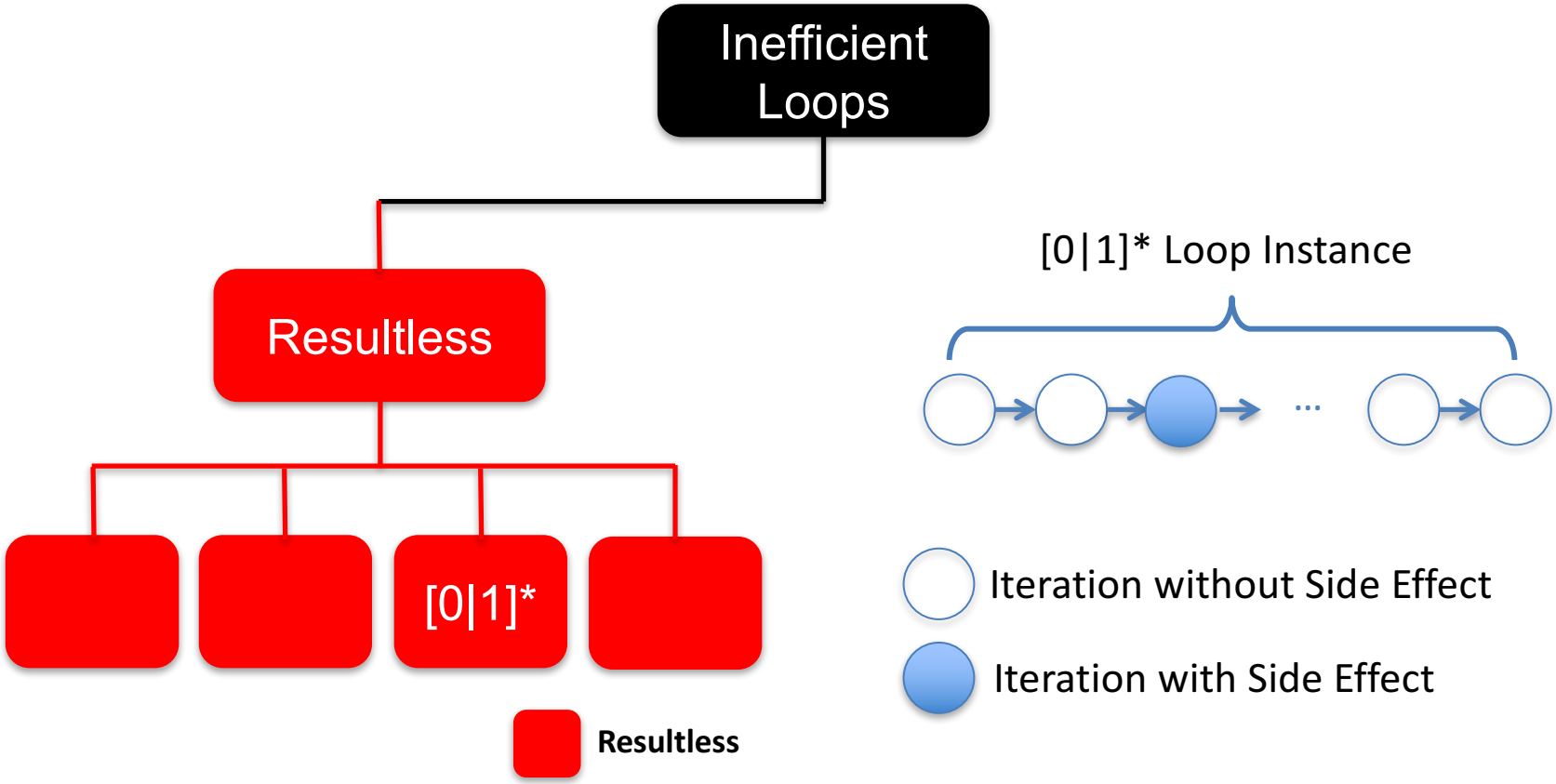
Resultless 0*1?

- Only produce results in the last iteration



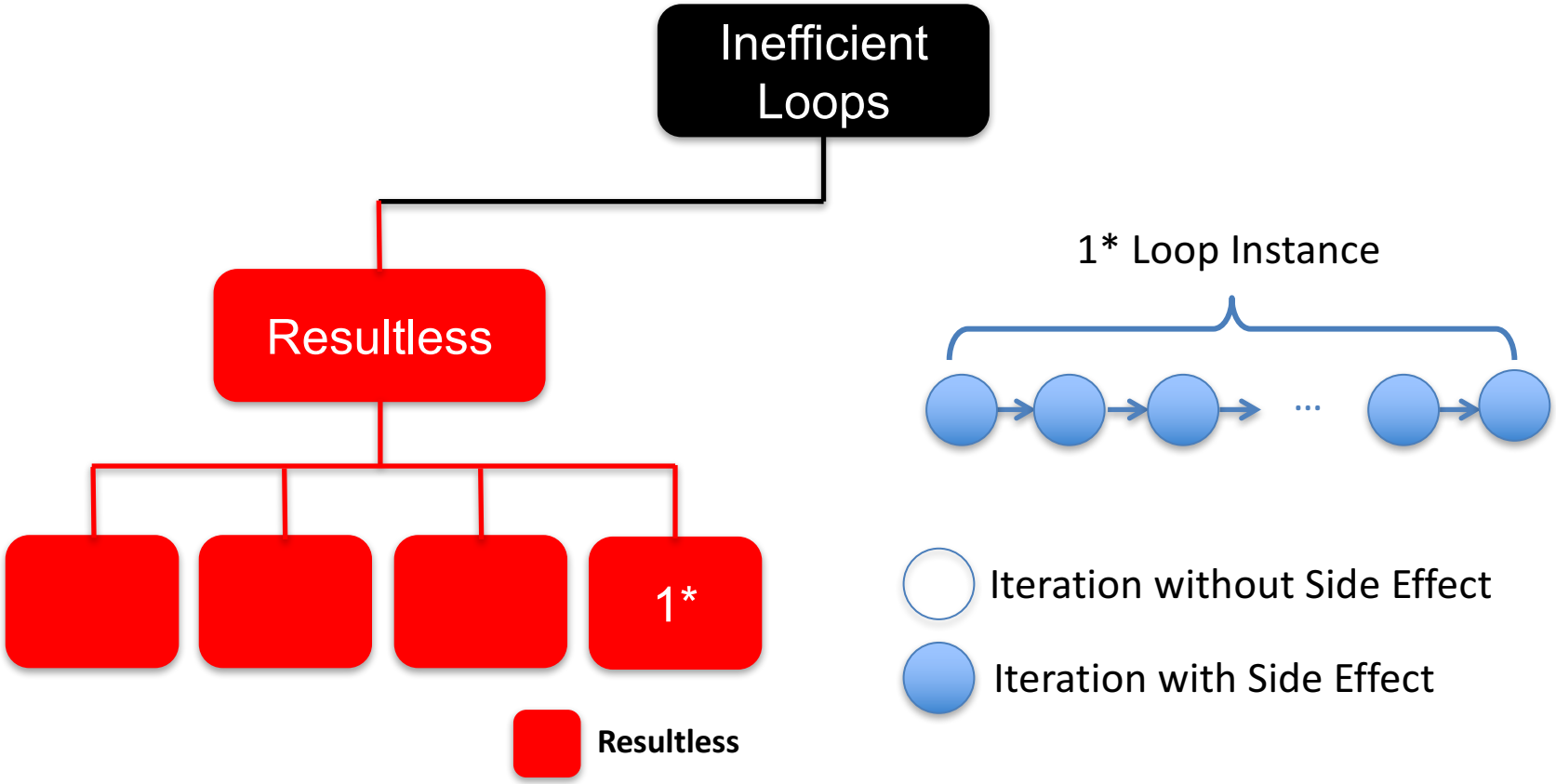
Resultless [0|1]*

- May produce results in each iteration



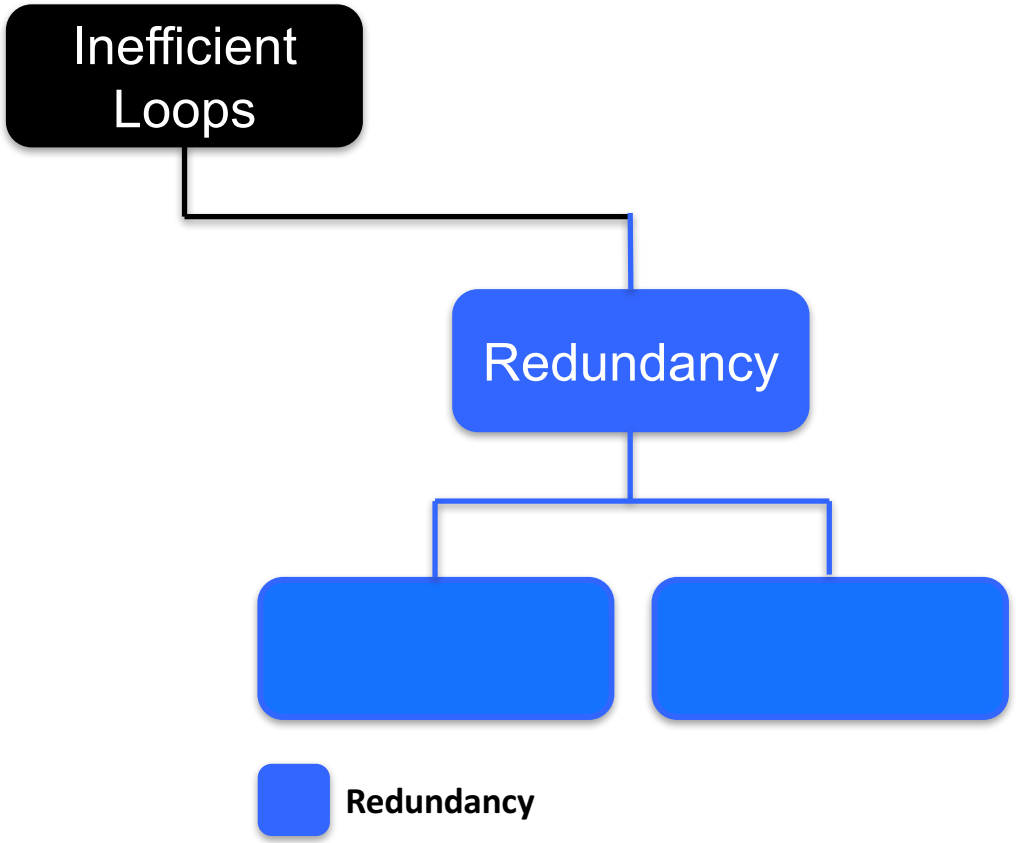
Resultless 1*

- Produce results in almost all iterations

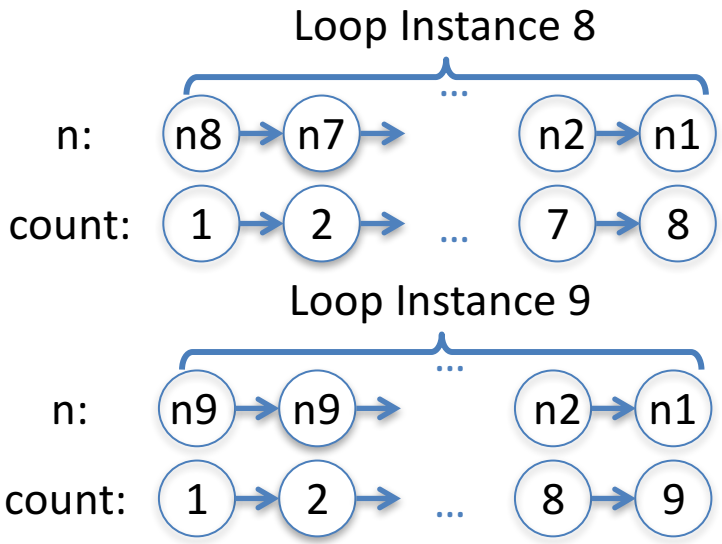


Redundant Loops

- Produce already-available results

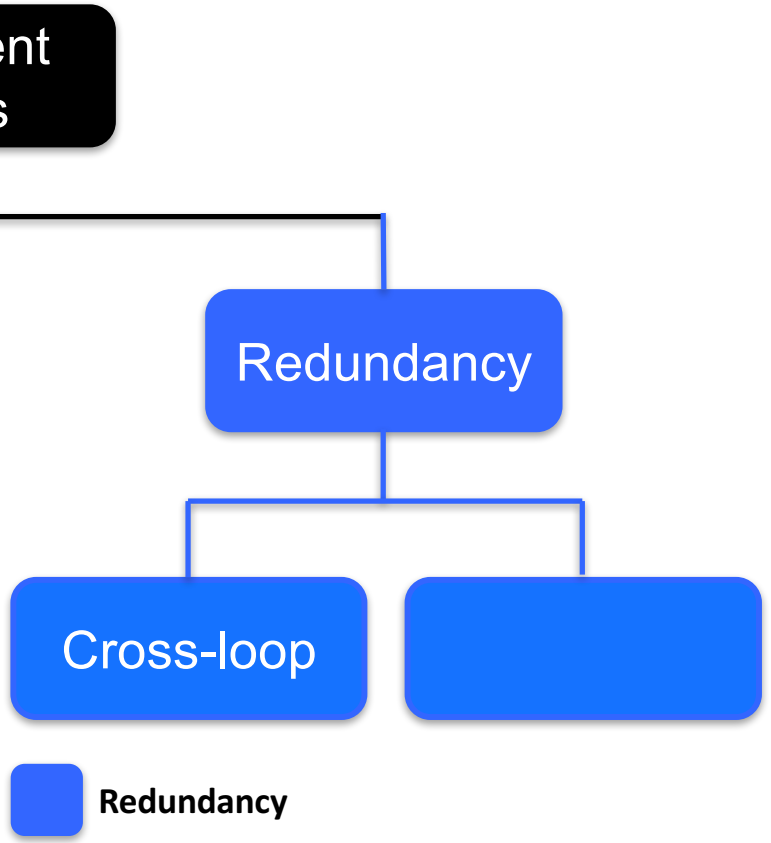


Cross-loop Redundancy

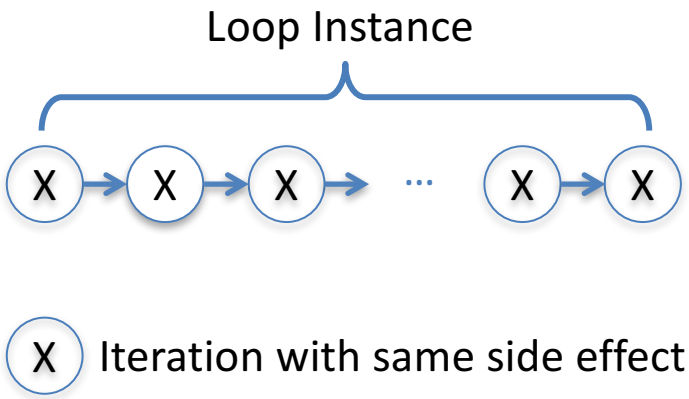


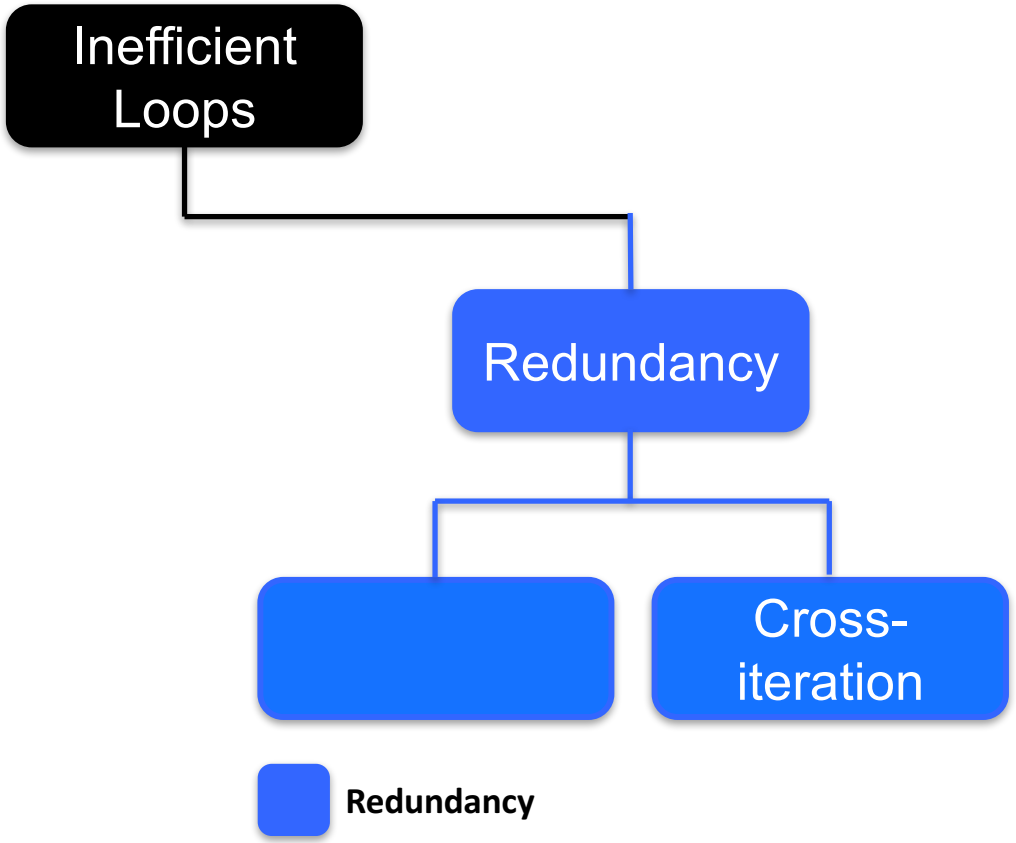

```
char * ssh_xph_generate(node_t * aNode)
{
   int count = 0;
   for (n = aNode; n; n = aNode->prev)
      if(n->name == aNode->name)
         count ++;
   return count;
} //Mozilla#477564
```

Buggy Loop

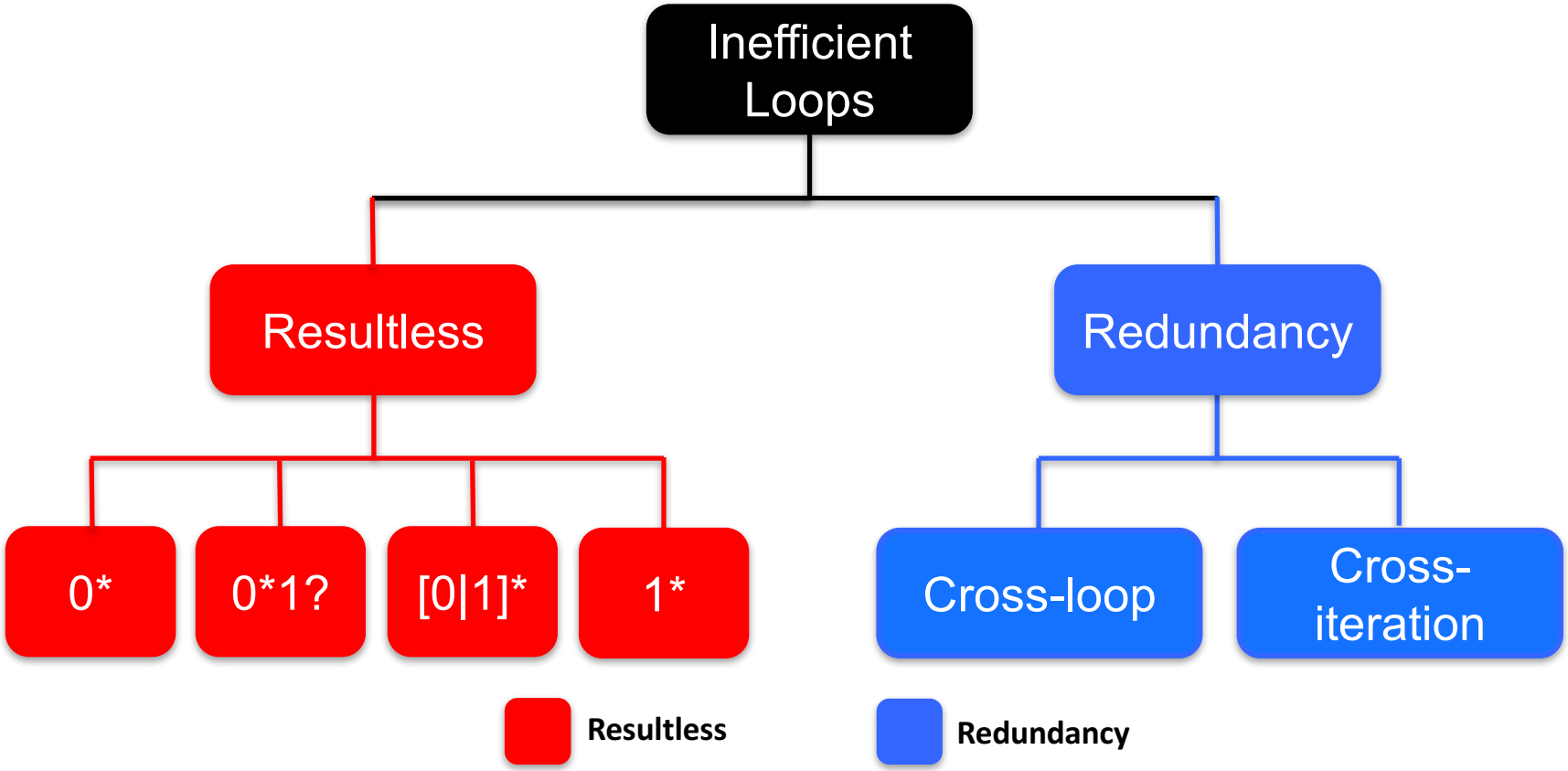


Cross-iteration Redundancy





Root-Cause Taxonomy

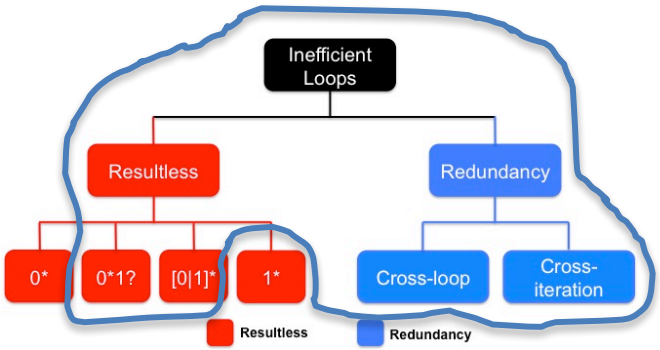


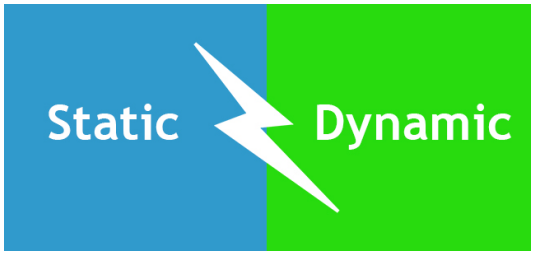
Outline

- Overview
- Root-cause Taxonomy
- LDoctor Design
- Evaluation of LDoctor
- Conclusion

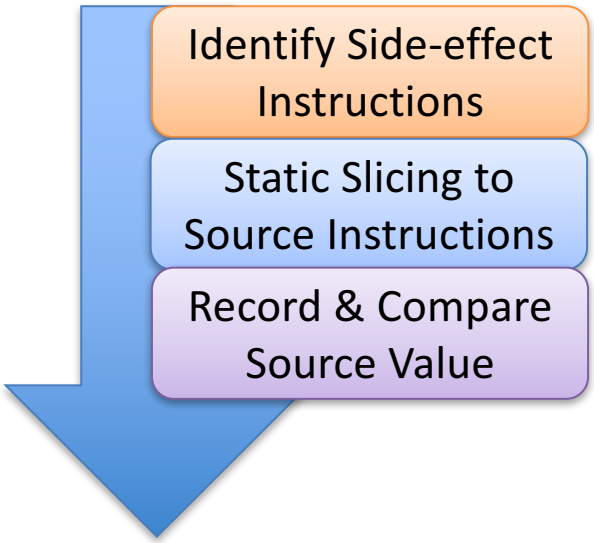
Design Principles

- Taxonomy guided design
- Focused checking
- Static-dynamic hybrid analysis





C.-L. Redundancy Analysis (I)



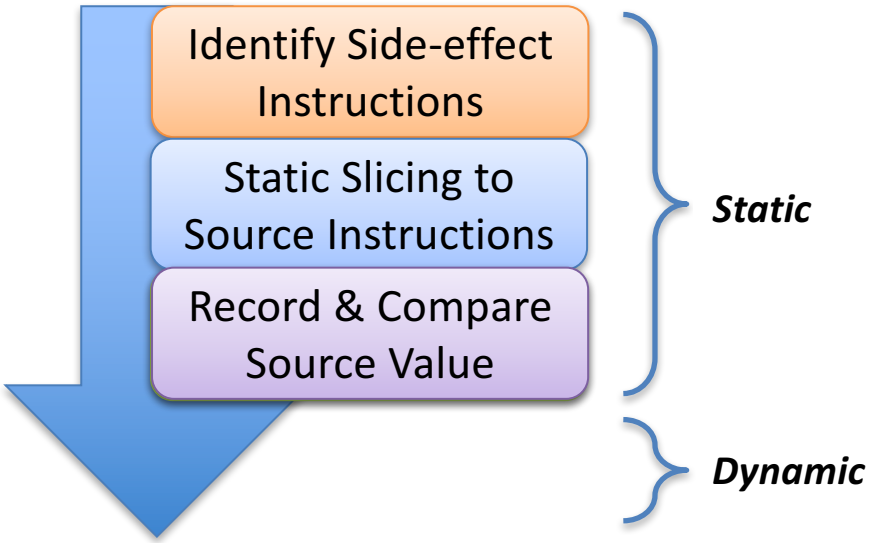

```
char * ssh_xph_generate(node_t * aNode)
{
   int count = 0;
   for (n = aNode; n; n = aNode->prev)
      if(n->name == aNode->name)
         count ++;
   …
}// called for ev
```

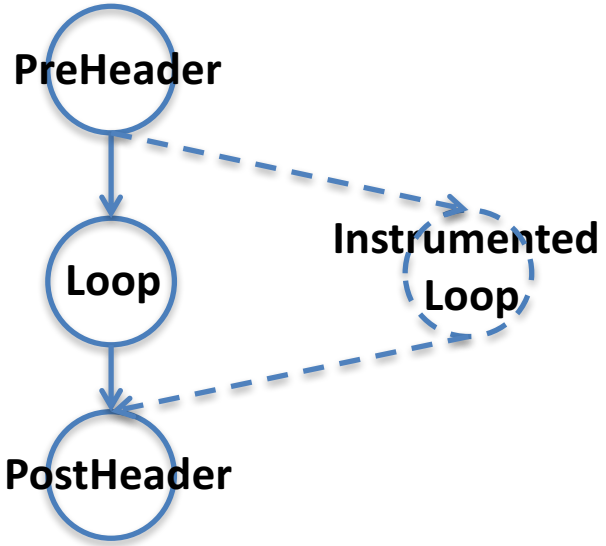
Source Instruction

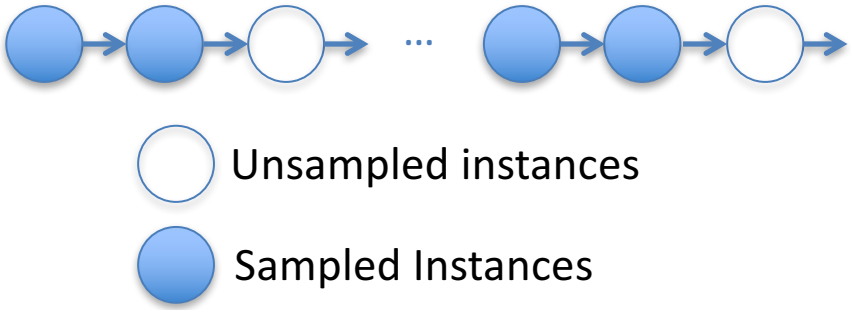
Side-effect Instruction

Mozilla#477564

C.-L. Redundancy Analysis (II)







Outline

- Overview
- Root-cause Taxonomy
- LDoctor Design
- Evaluation of LDoctor
- Conclusion

Implementation & Evaluation

- Implementation using LLVM-3.4.2
- Experimental setting
  - Evaluating 39 bugs from two benchmark suite
  - Applying on look rank list from SD
  - Sampling rate: 1/100 (C.L.) & 1/1000 (C.I.)
- Evaluation metrics

Coverage & Accuracy

BugID	Reported Root Cause	Fix Suggestion	False Positive
Mozilla347306	√	√	-
[illegible]	√	√	-
Mozilla490742	√	√	-
Mozilla35294	√	√	-
Mozilla477564	√	√	-
MySQL27287	√	✗	01
MySQL15811	√	√	-
Apache32546	√	√	-
Apache37184	√	√	-
Apache29742	√	√	-
Apache34464	√	√	-
Apache47223	√	√	-
…	…	…	…

Coverage

Coverage & Accuracy

BugID	Reported Root Cause	Fix Suggestion	False Positive
Mozilla347306	√	√	-
Mozilla416628	√	√	-
Mozilla490742	√	√	-
Mozilla35294	√	√	-
Mozilla477564	√	√	-
MySQL27287	√	✗	01
MySQL15811	√	√	-
Apache32546	√	√	-
Apache37184	√	√	-
Apache29742	√	√	-
Apache34464	√	√	-
Apache47223	√	√	-
…	…	…	…

Accuracy

Performance

BugID	LDoctor			w/o sampling	
	Resultless	C-L R.	C-I R.	C-L R.	C-I R.
Mozilla347306	1.00	1.18	1.18	356	662
Mozilla416628	1.00	1.04	1.02	73.5	113
MySQL27287	1.05	1.00	-	264	881
MySQL15811	-	1.00	-	414	1088
GCC46401	1.00	1.00	1.00	33.8	60.0
GCC1687	-	/	1.01	/	224
GCC27733	1.00	/	1.00	/	19.4
GCC8805	-	1.00	1.00	3.89	4.36
GCC21430	-	1.04	1.02	165	255
GCC12322	-	1.04	1.00	26.5	25.8

Conclusions

- A root-cause taxonomy for inefficient loops

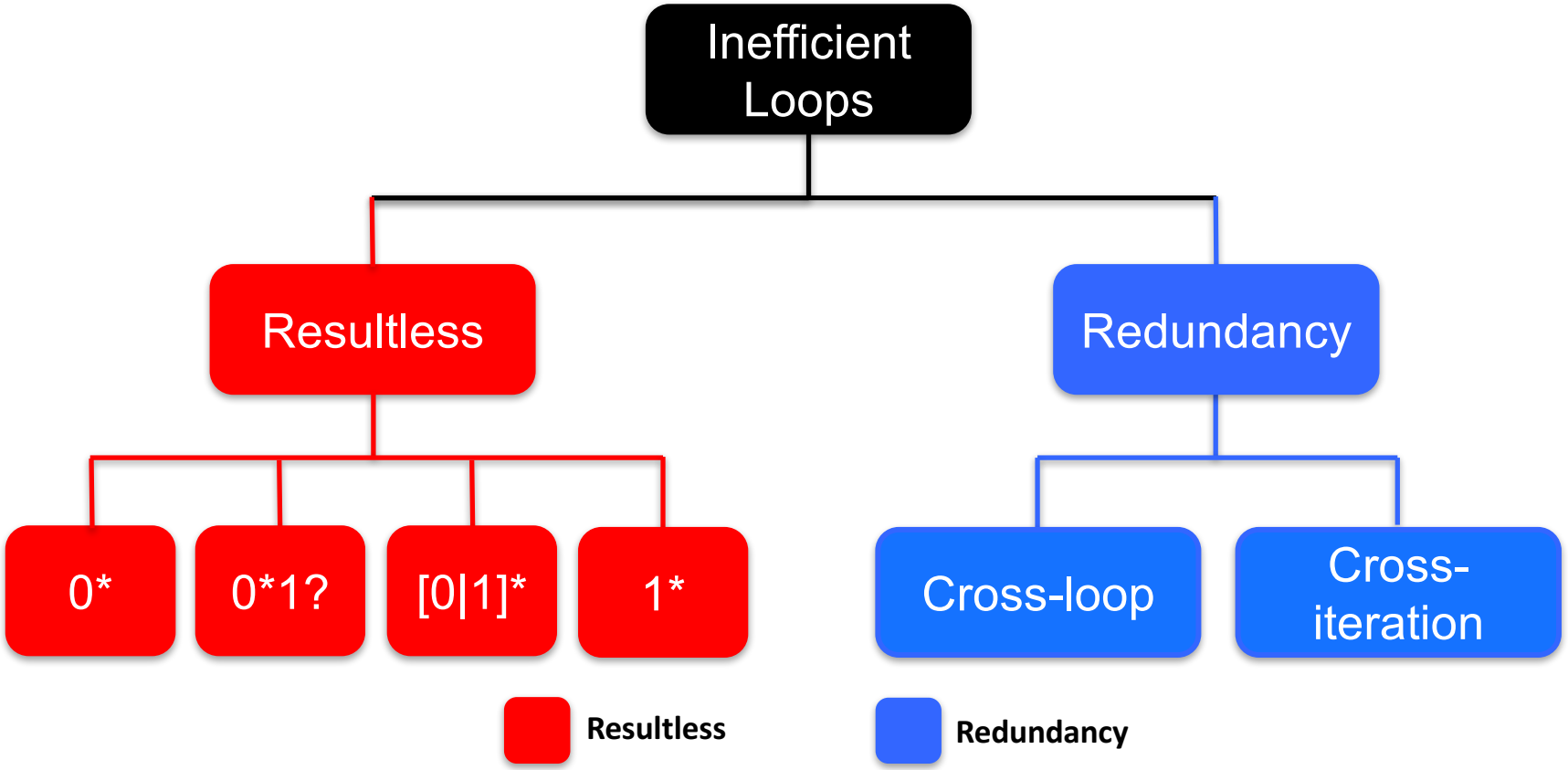
  Our taxonomy are general and specific!

- A series of static-dynamic hybrid analysis

  Achieve good coverage, accuracy and performance

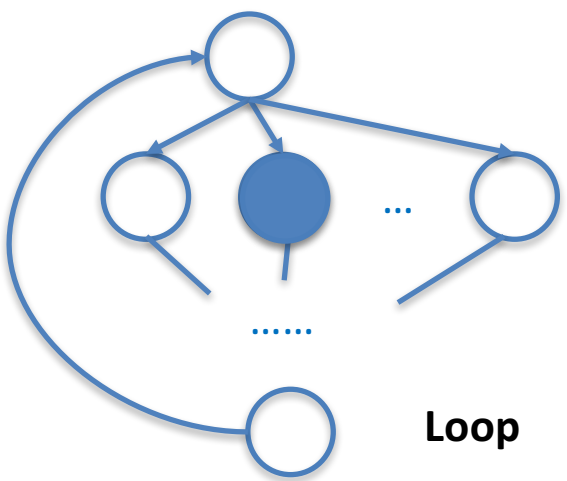
Thanks a lot!

Question?


Inefficient Loops
Resultless
Redundancy
0*
0*1?
[0|1]*
1*
Cross-loop
Cross-iteration
Resultless
Redundancy


BACKUP

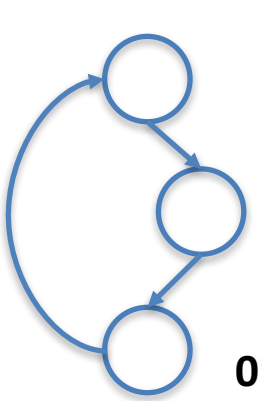
Resultless Analysis



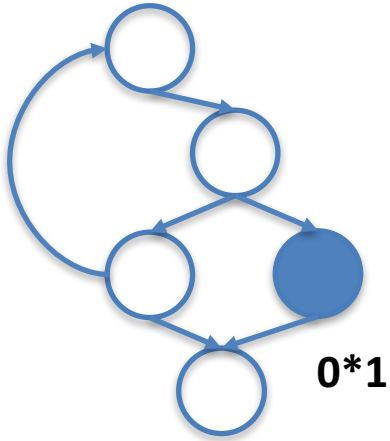
Loop

with side effect

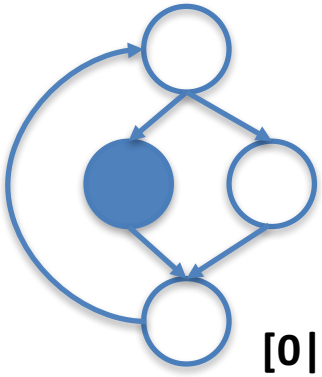
without side effect



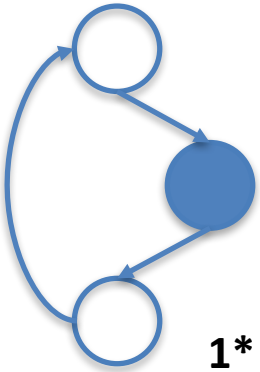
0*



0*1?



[0|1]*



1*