# SanRazor: Reducing Redundant Sanitizer Checks in C/C++ Programs

Jiang Zhang[1], Shuai Wang[2], Manuel Rigger[3], Pinjia He[3], Zhendong Su[3]

1: *University of Southern California*

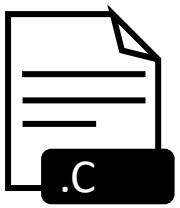
2: *HKUST*

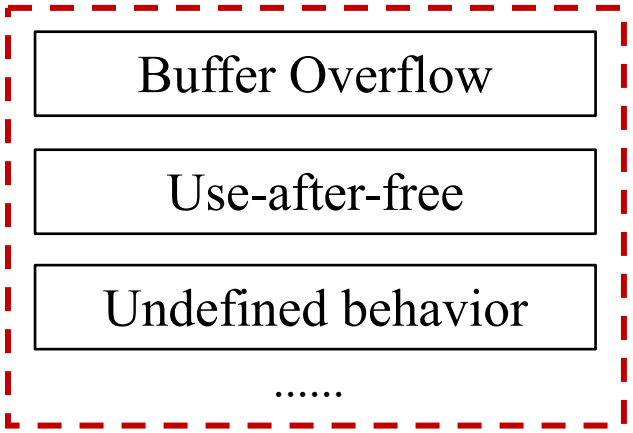
3: *ETH Zurich*



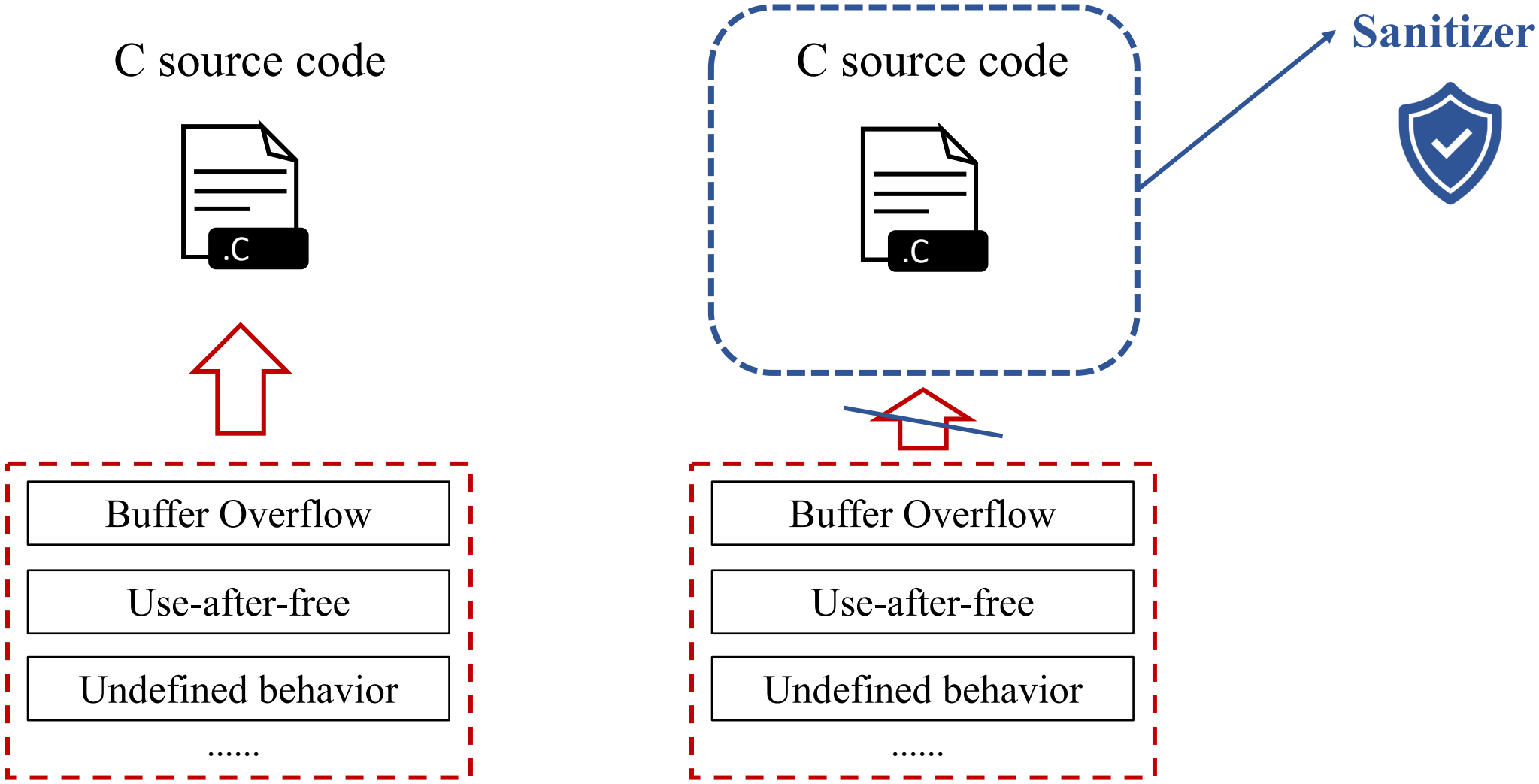
# C/C++ programs are unsafe

C source code

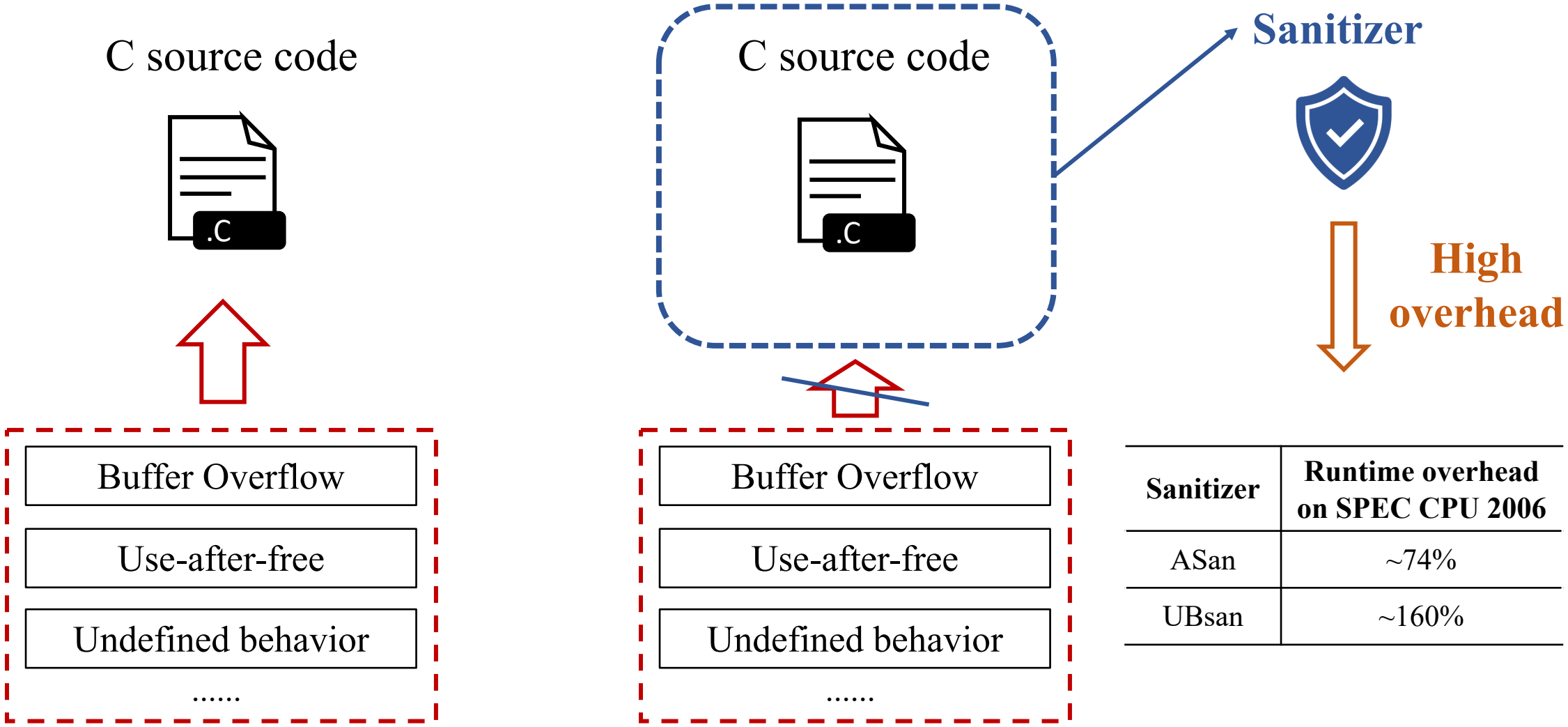


# Sanitizers are designed to detect software bugs/vulnerabilities

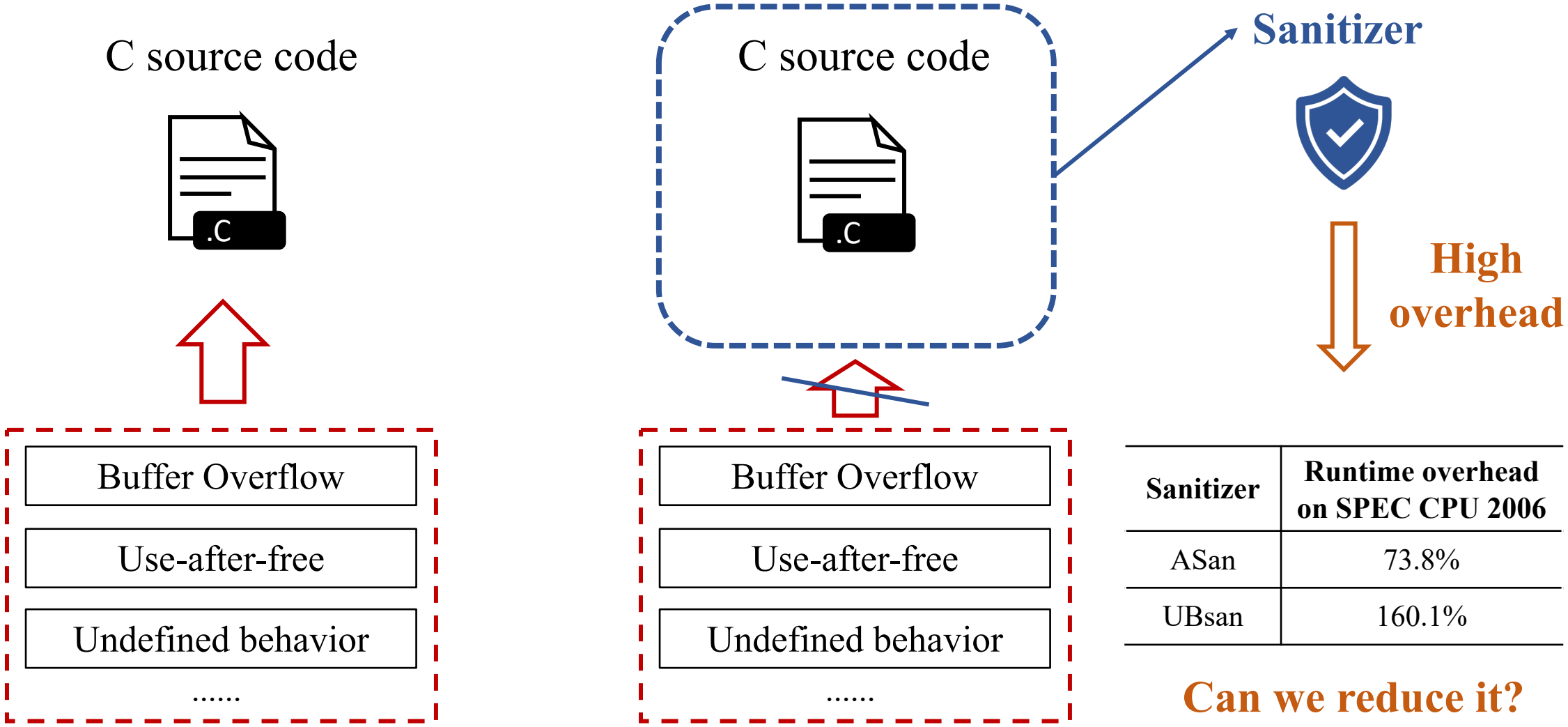


# However, sanitizers have high runtime overhead




| Sanitizer | Runtime overhead on SPEC CPU 2006 |
|---|---|
| ASan | ~74% |
| UBsan | ~160% |

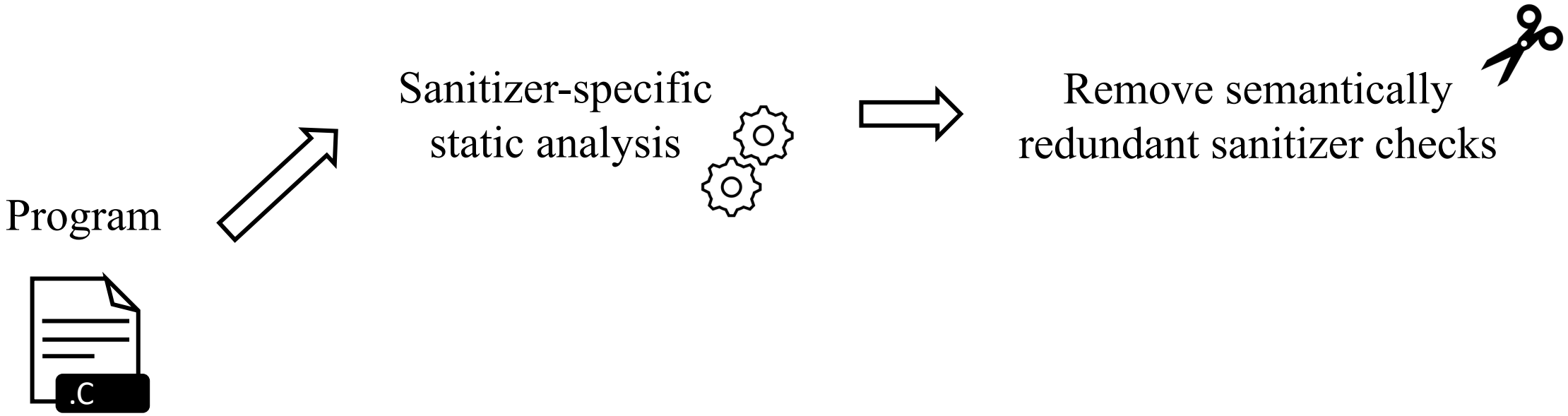
# However, sanitizers have high runtime overhead



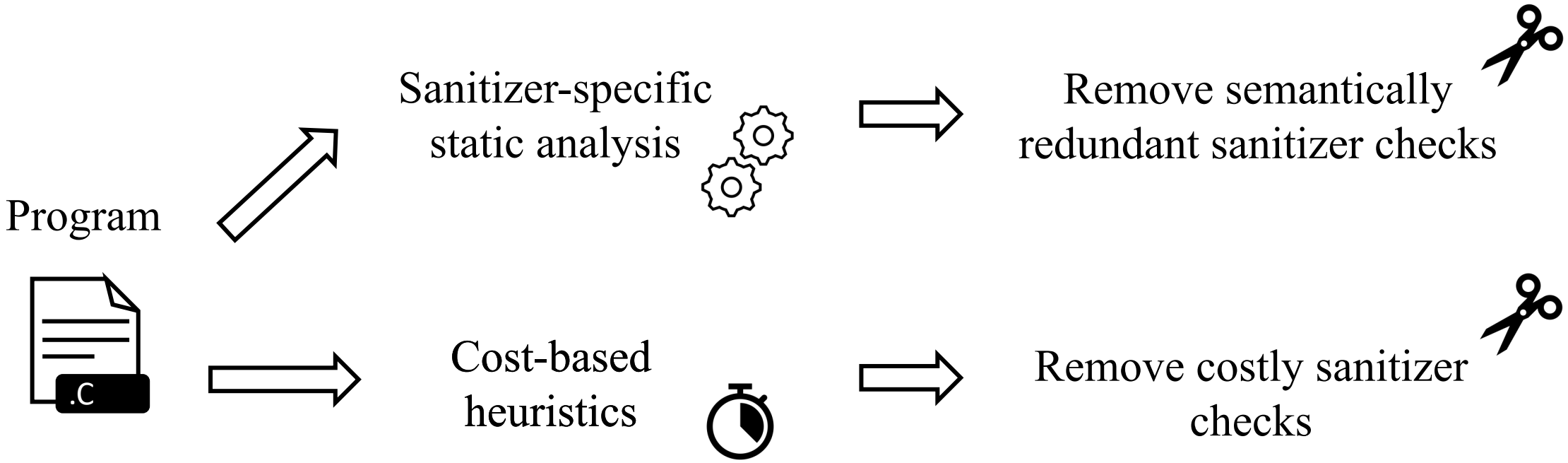

| Sanitizer | Runtime overhead on SPEC CPU 2006 |
|---|---|
| ASan | 73.8% |
| UBsan | 160.1% |

**Can we reduce it?**

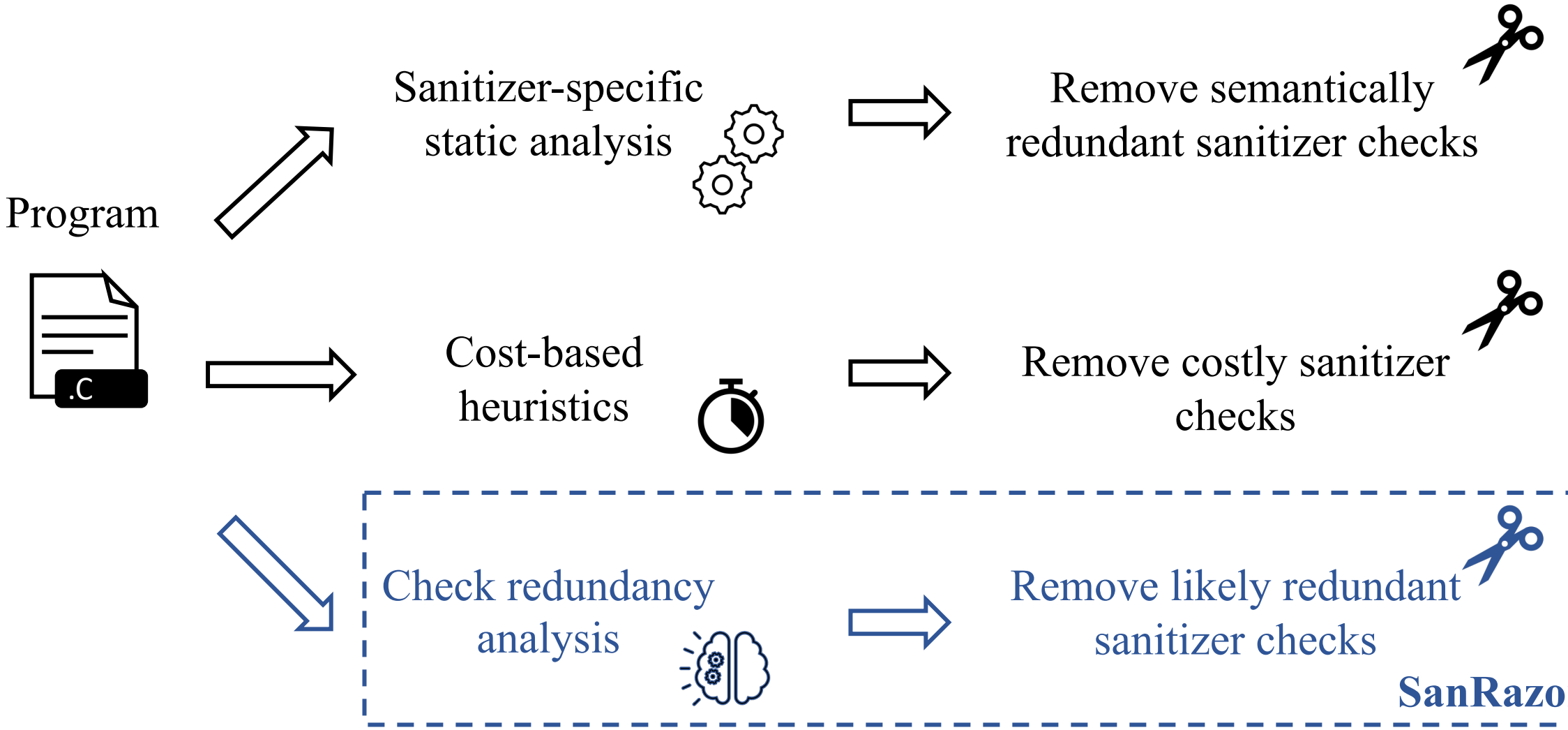
# Prior approaches



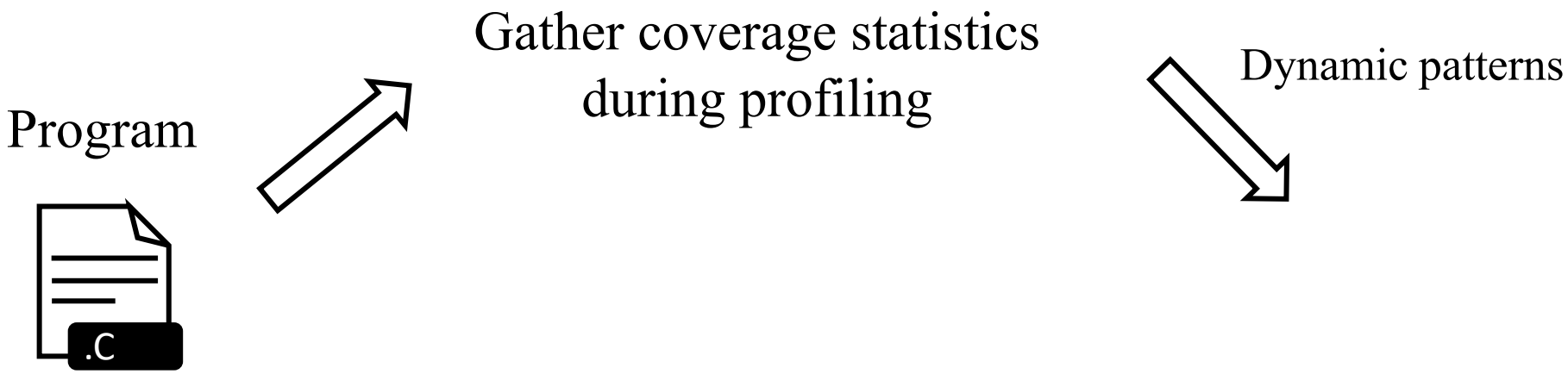
# Prior approaches



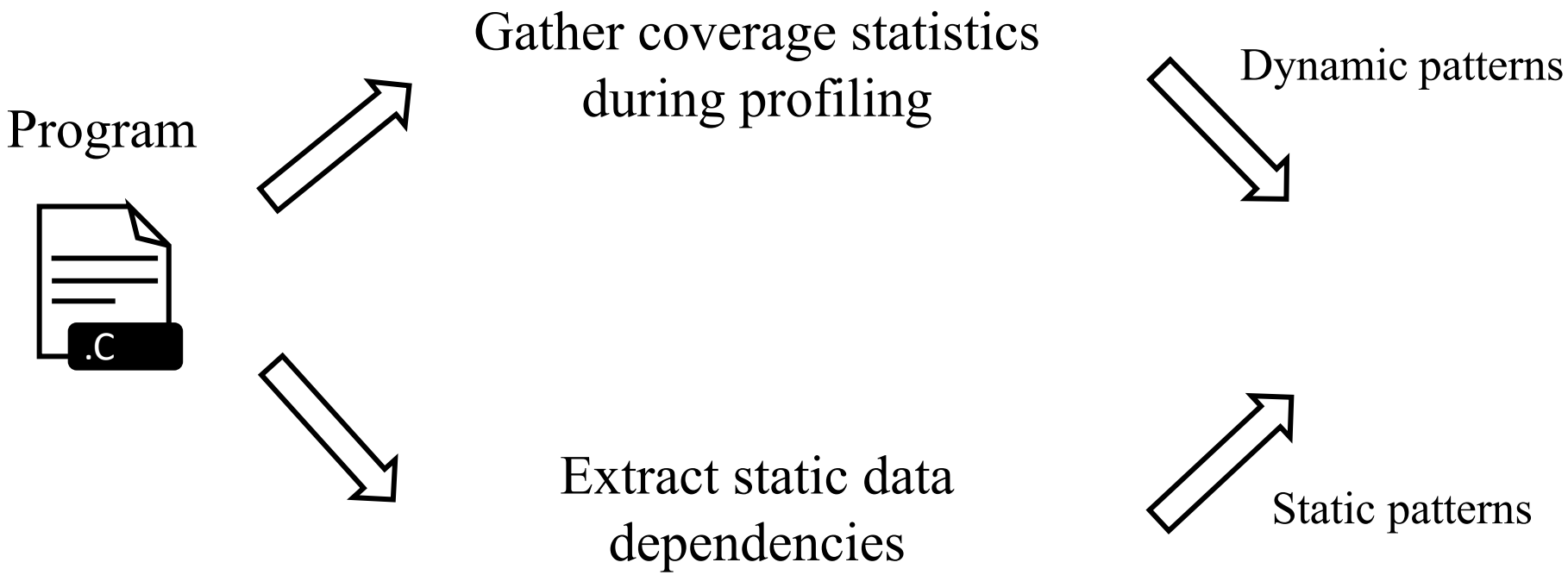
# Our novel design



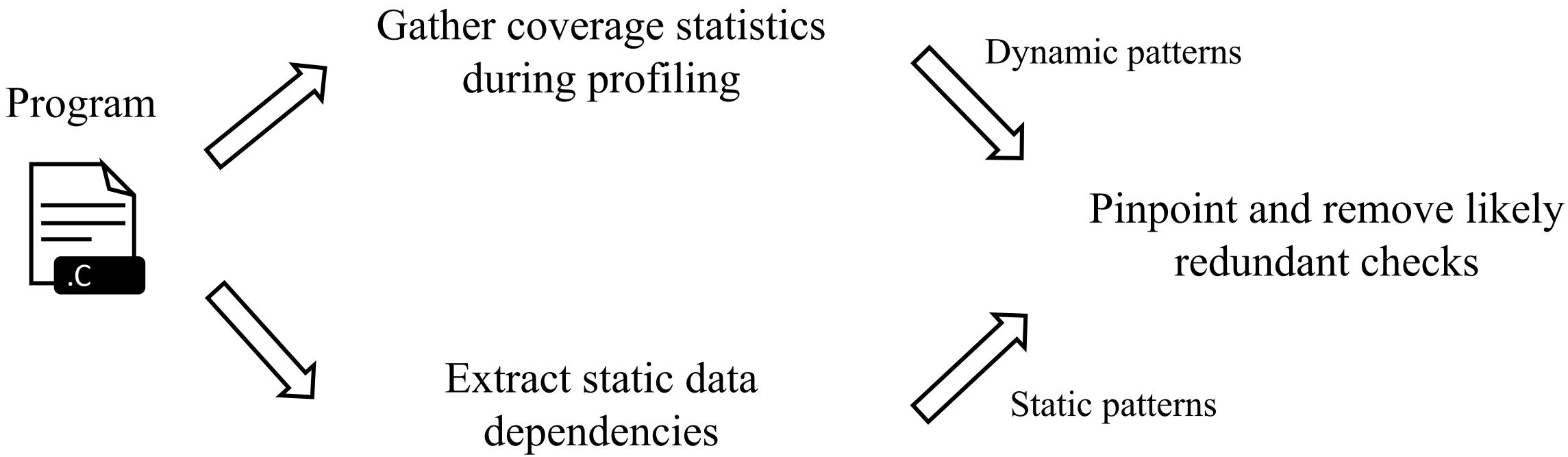
# Design: overall workflow



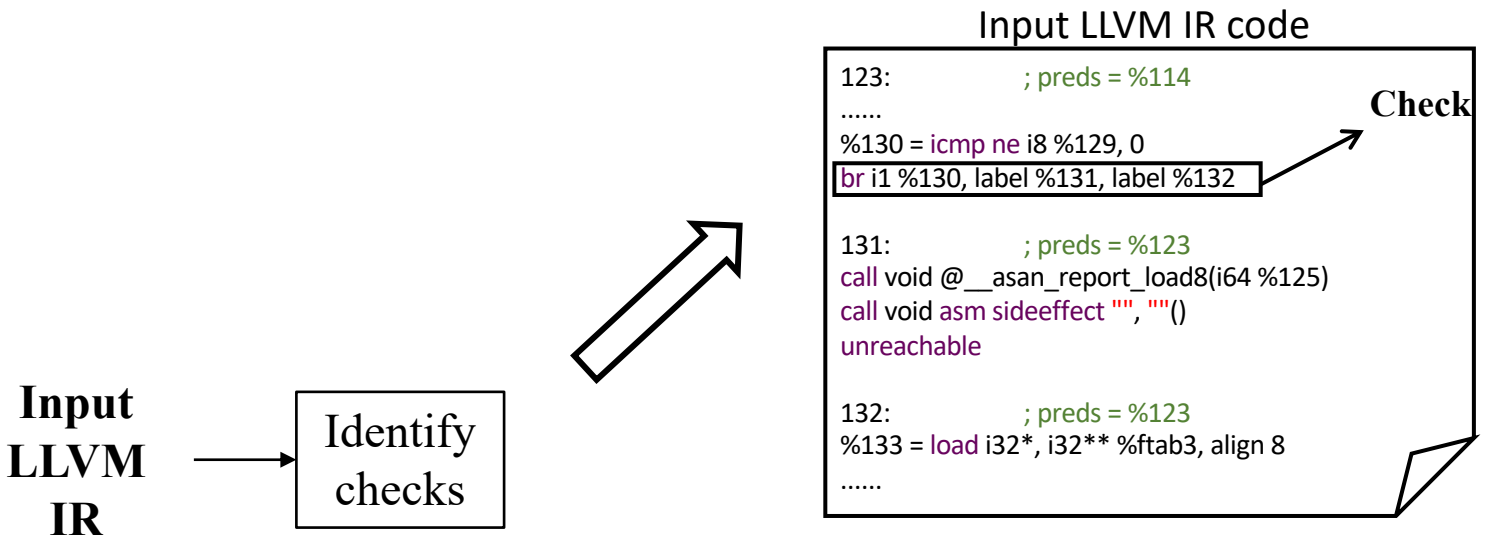
# Design: overall workflow



# Design: overall workflow



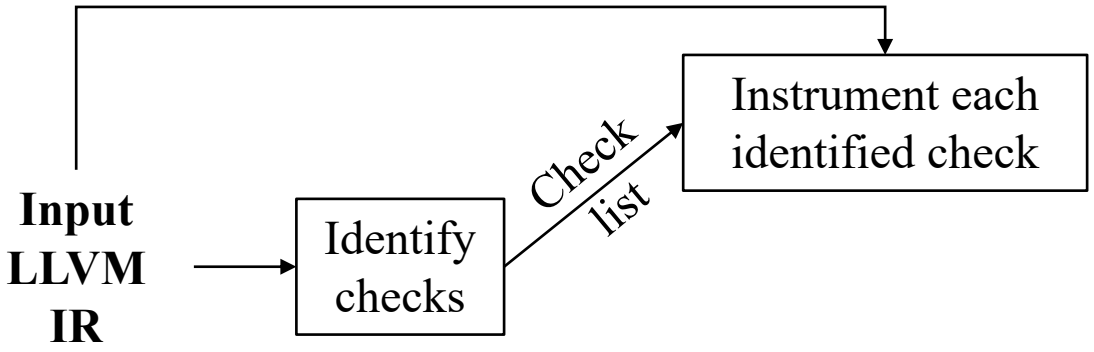
# Design: check identification




Check in LLVM IR:

```
%o = icmp cond %a, %b
br i1 %o, label %bb1, label %bb2
```

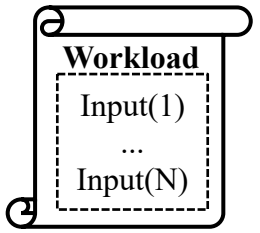
# Design: check identification

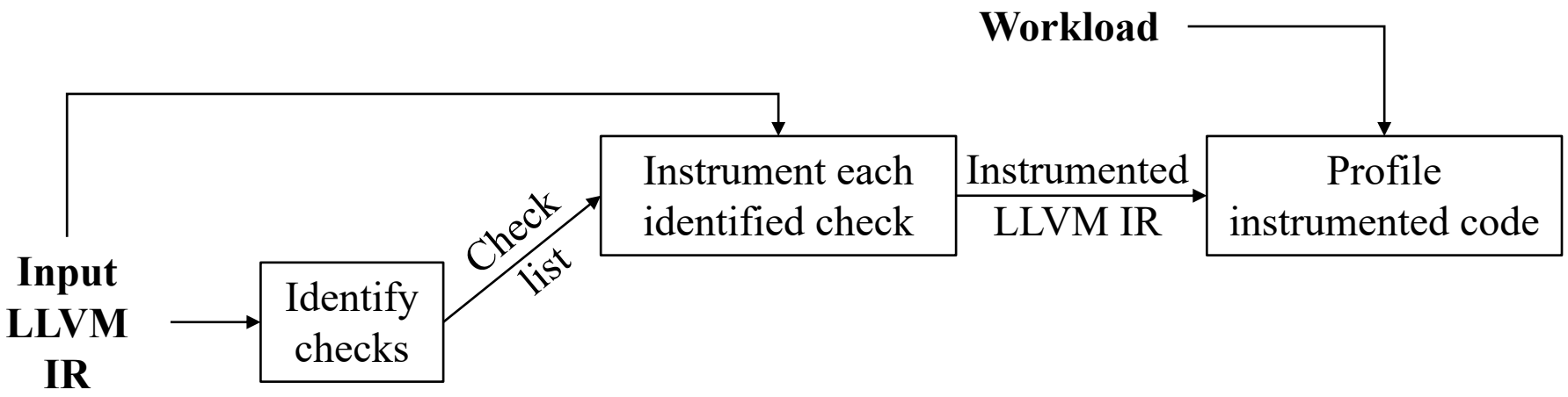



Instrumented LLVM IR code

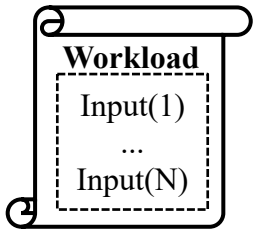
```
123:                ; preds = %114
......
%130 = icmp ne i8 %129, 0
call void @COUNTER_calledSCbzip2(i64 32, i1 %130)
br i1 %130, label %131, label %132

131:                ; preds = %123
call void @__asan_report_load8(i64 %125)
call void asm sideeffect "", ""()
unreachable

132:                ; preds = %123
%133 = load i32*, i32** %ftab3, align 8
......
```

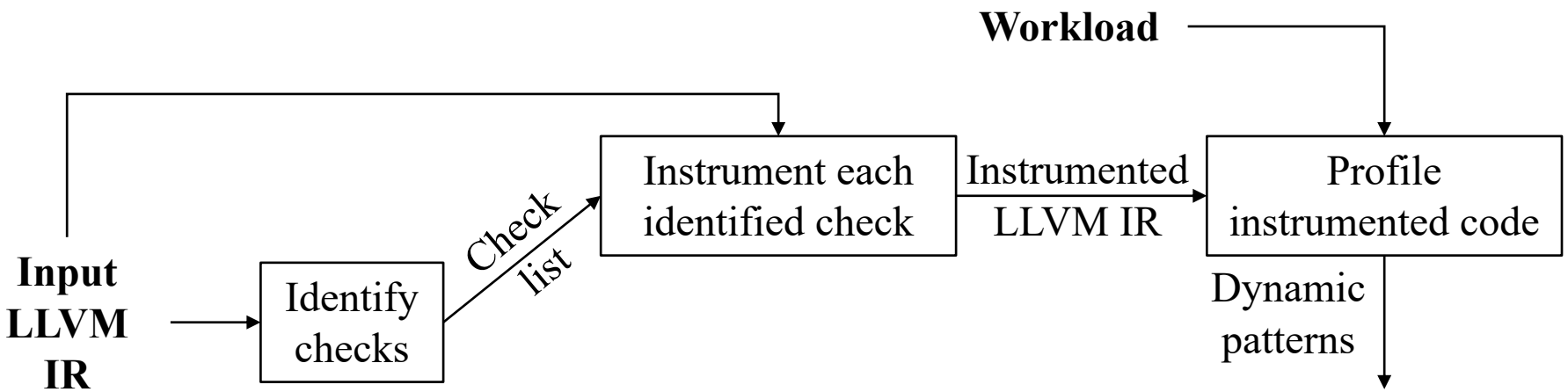
# Design: dynamic pattern capturing



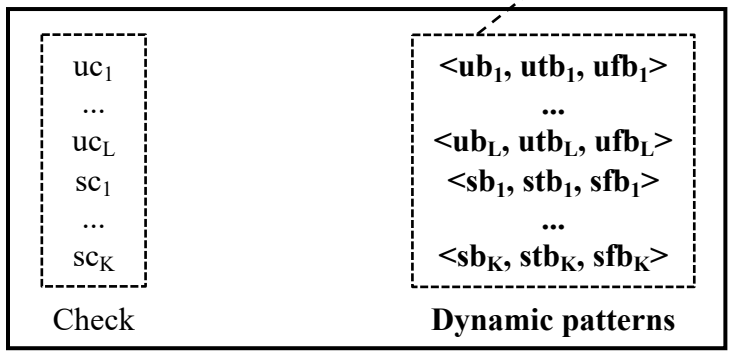

# Design: dynamic pattern capturing



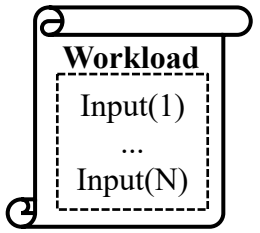



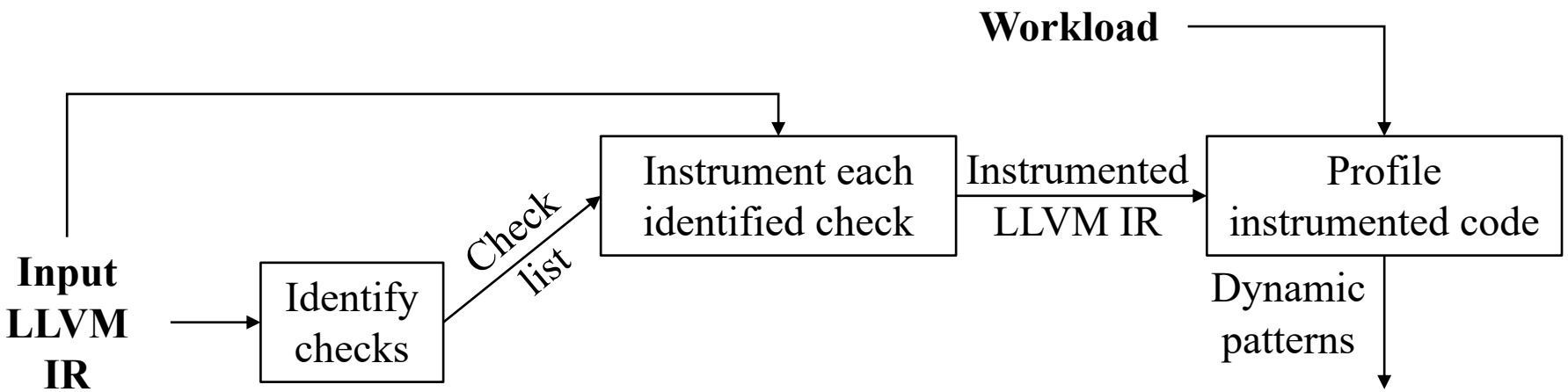

```
%o = icmp cond %a, %b
br i1 %o, label %bb1, label %bb2
```

Total covered times

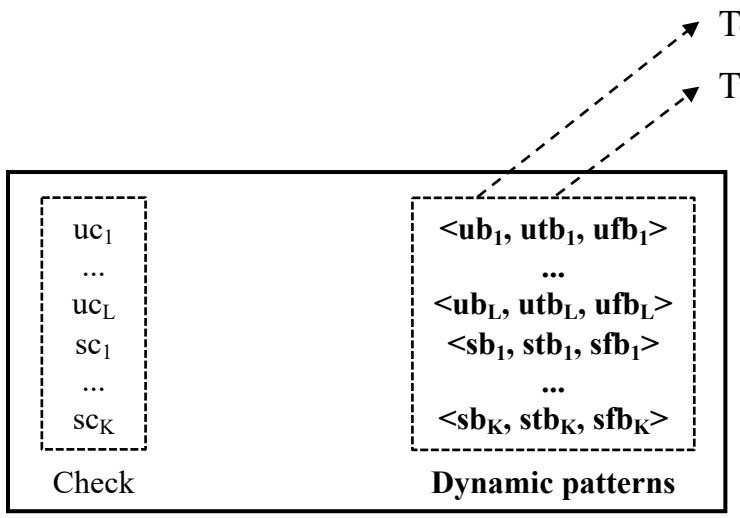


# Design: dynamic pattern capturing

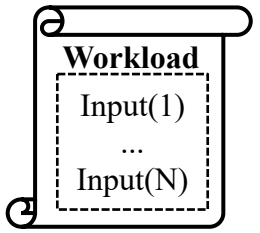


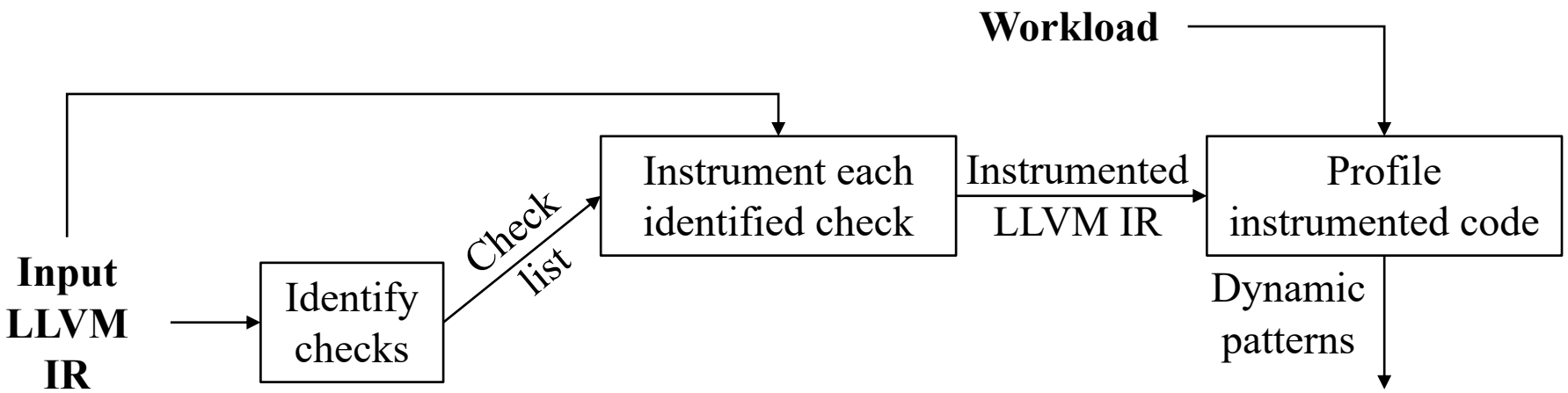




```
%o = icmp cond %a, %b
br i1 %o, label %bb1, label %bb2
```

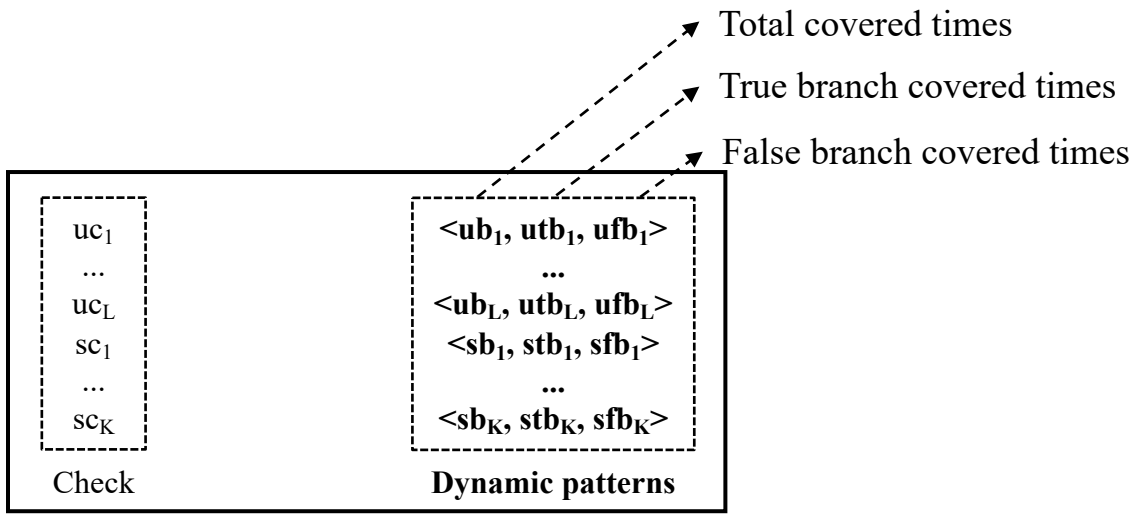


# Design: dynamic pattern capturing

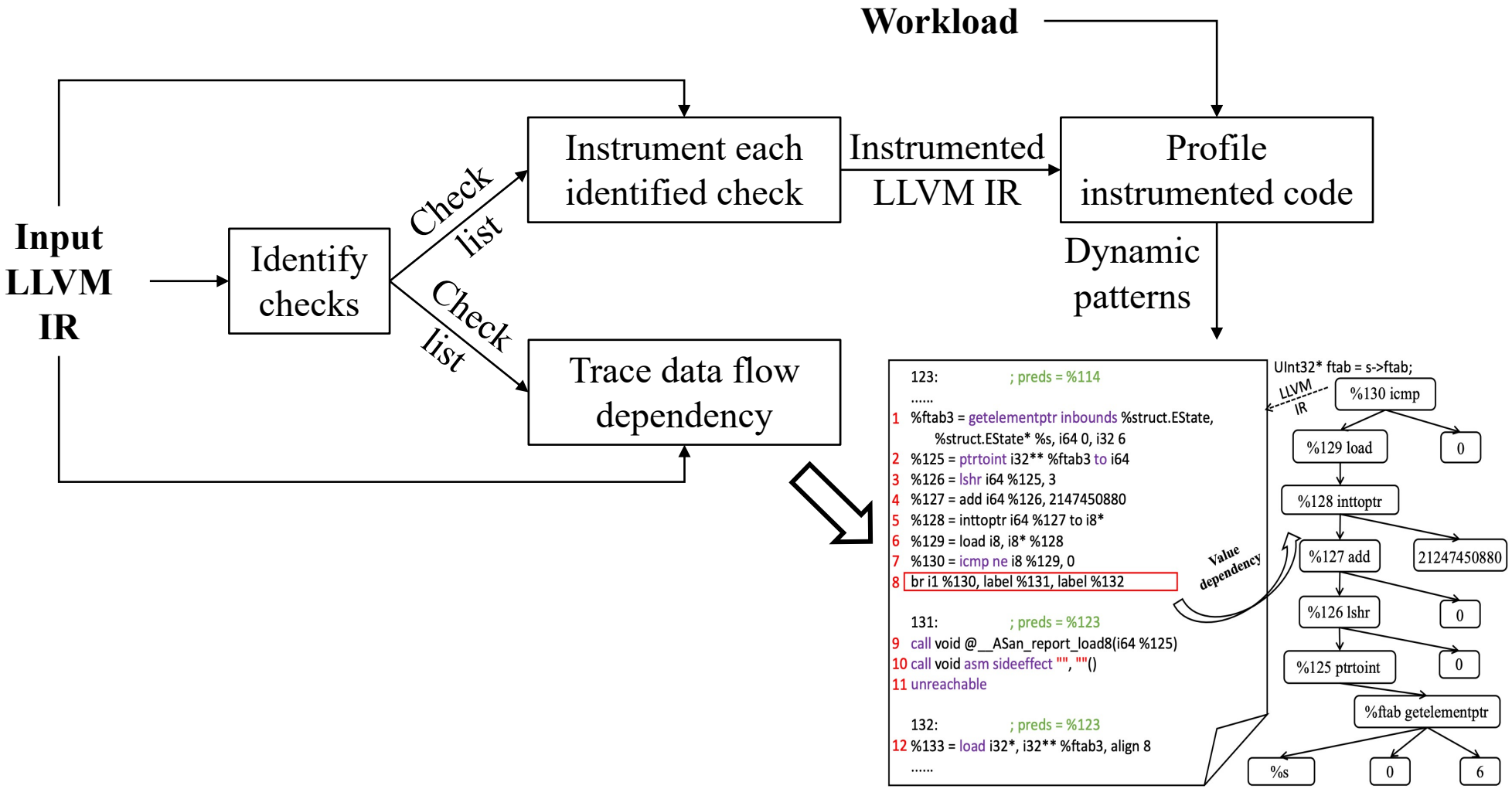


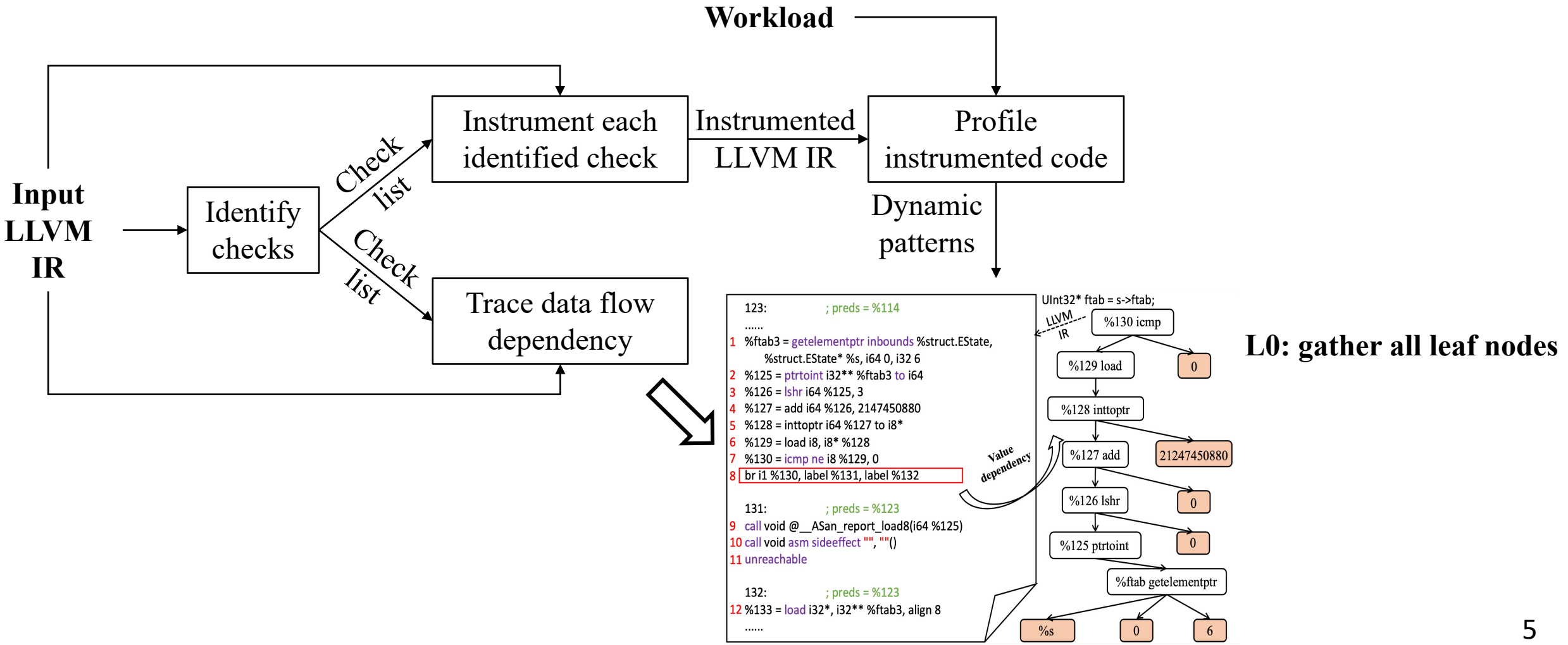

```
%o = icmp cond %a, %b
br i1 %o, label %bb1, label %bb2
```



# Design: static pattern capturing

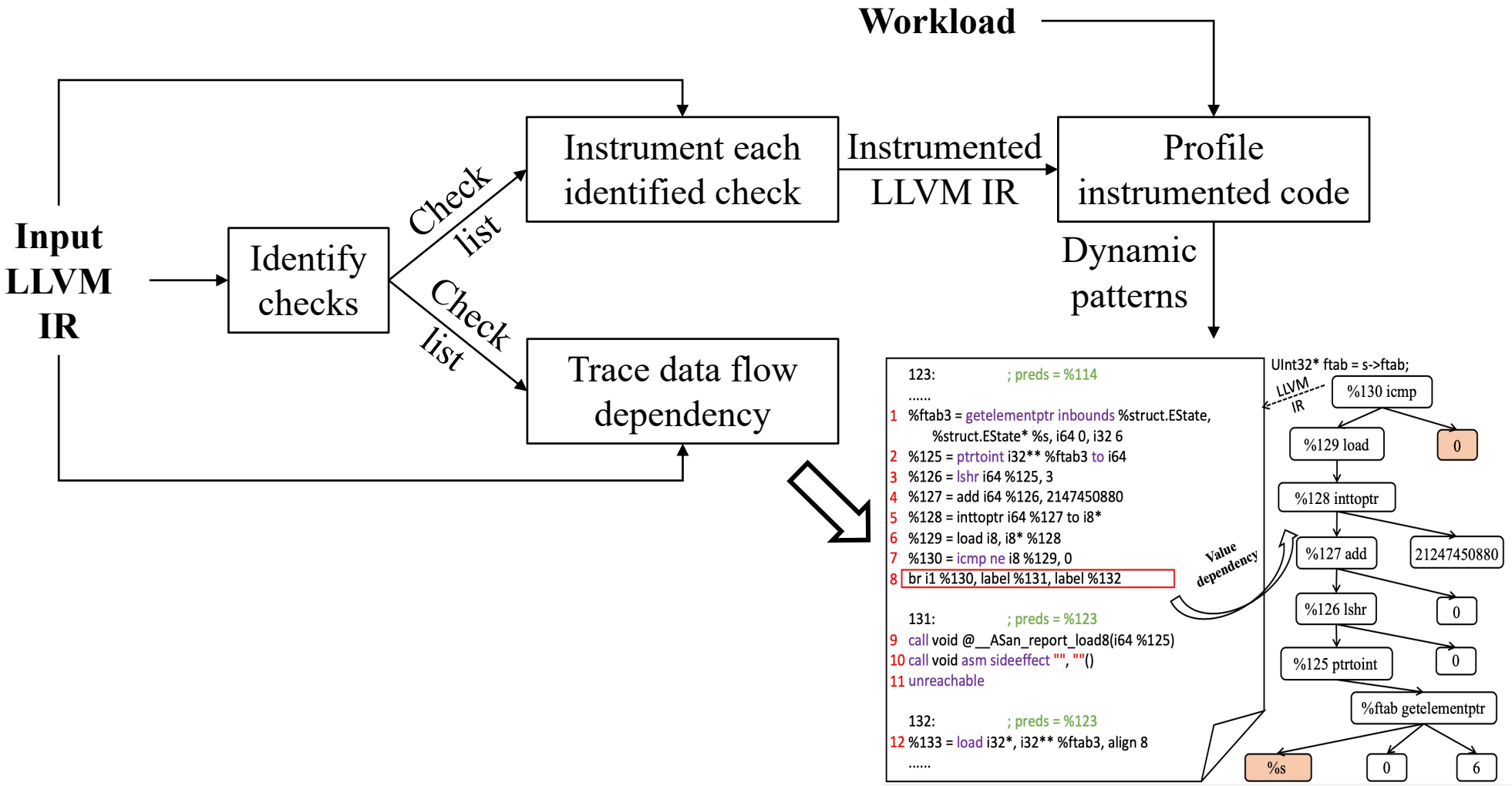


# Design: static pattern capturing




**L0: gather all leaf nodes**

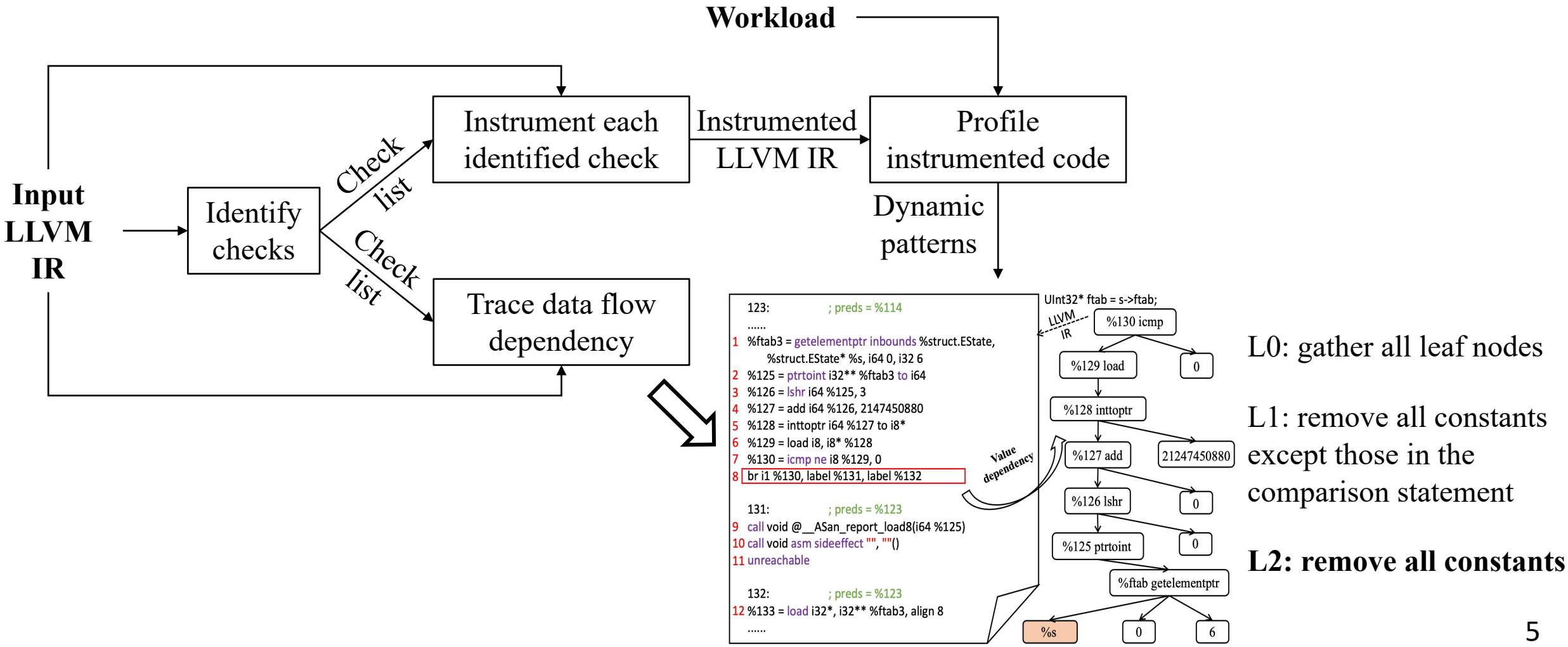
# Design: static pattern capturing




L0: gather all leaf nodes

**L1: remove all constants except those in the comparison statement**

# Design: static pattern capturing

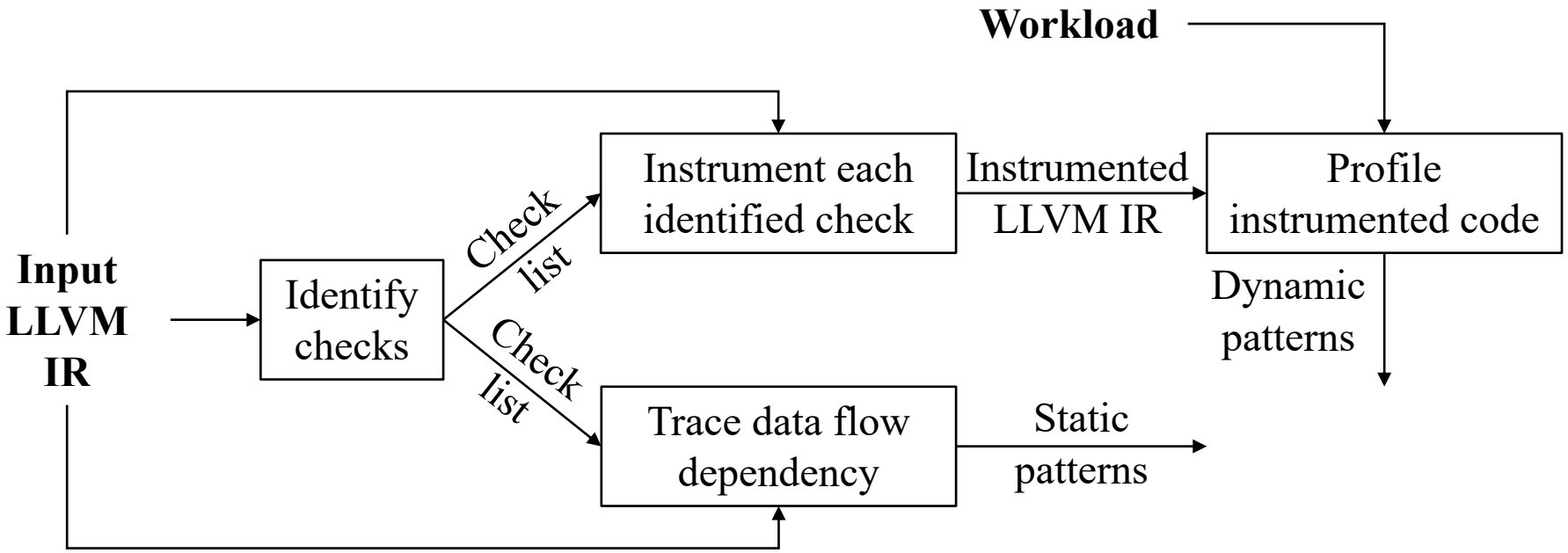



L0: gather all leaf nodes

L1: remove all constants except those in the comparison statement

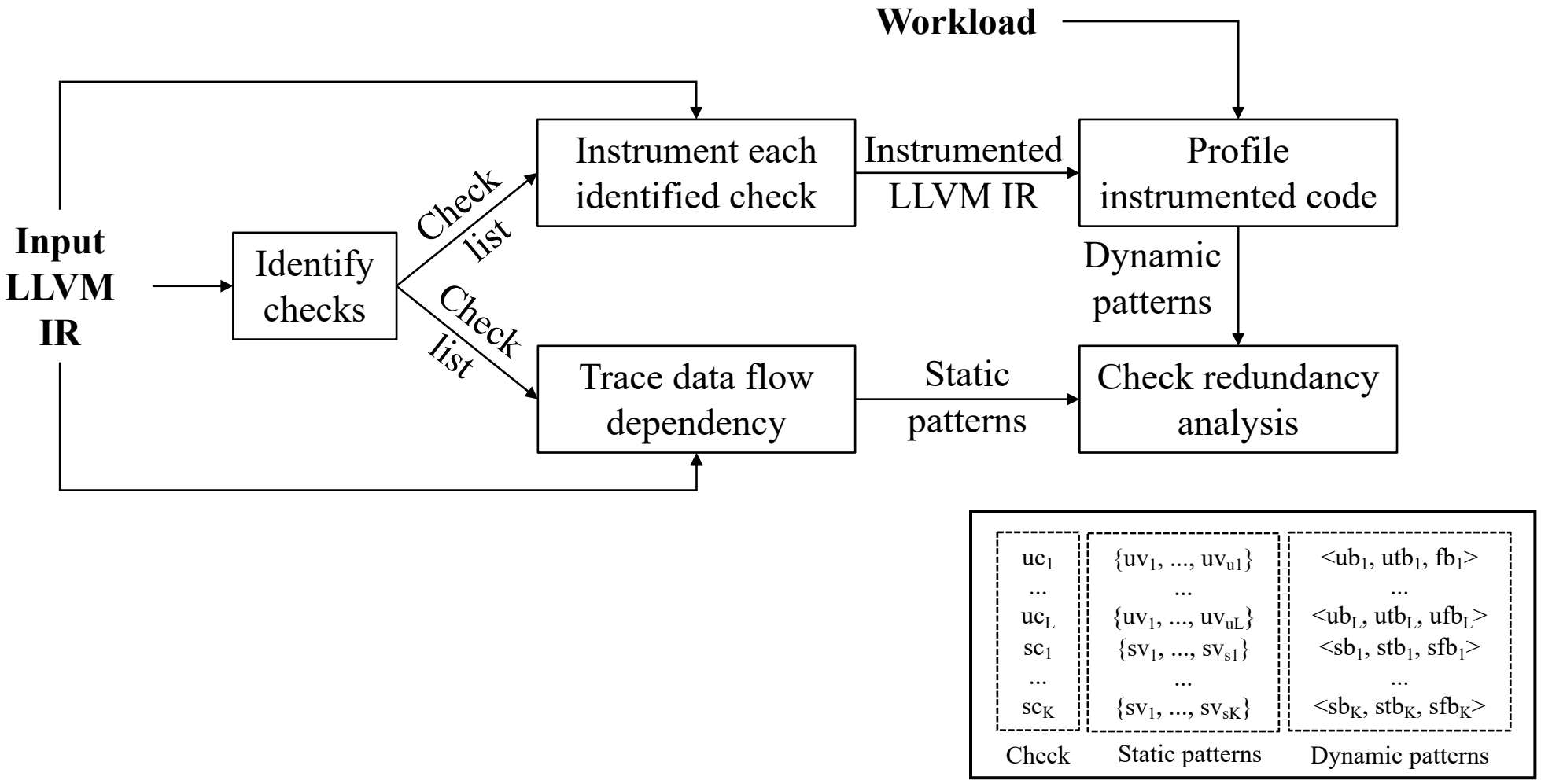
**L2: remove all constants**

# Design: sanitizer check reduction

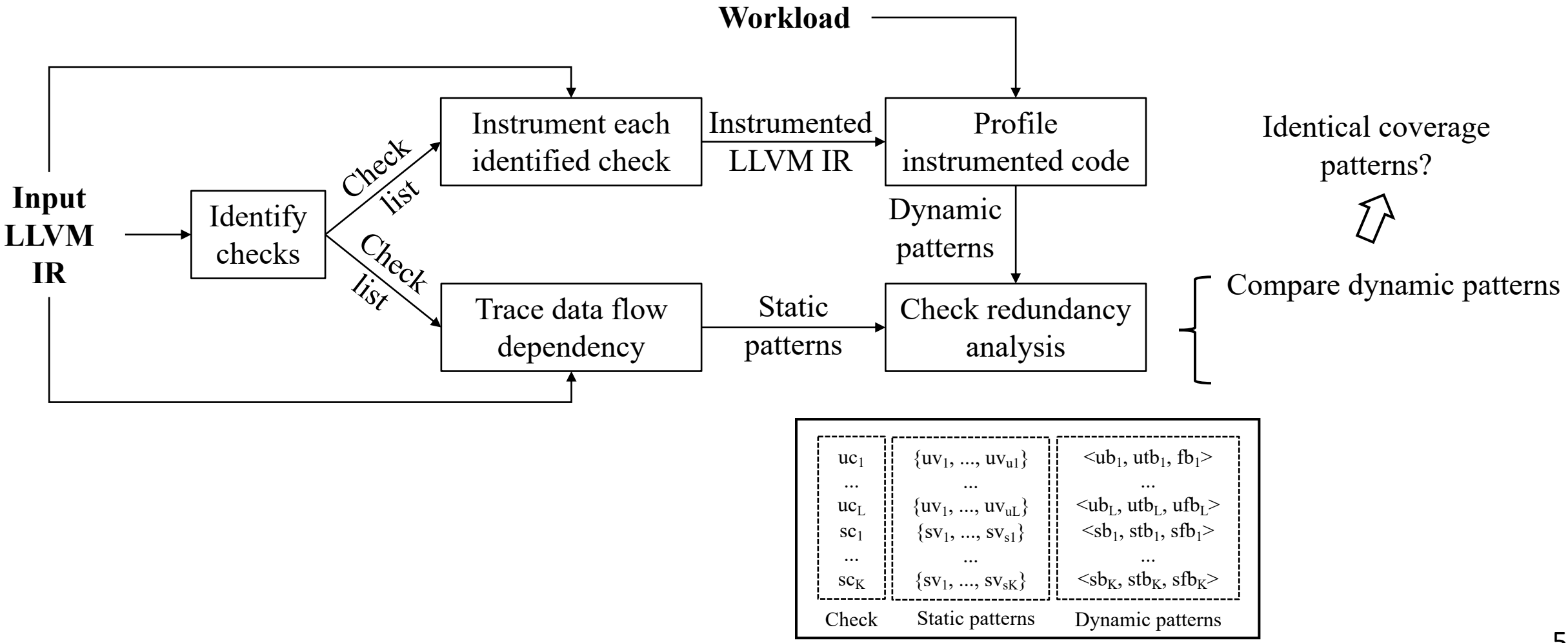



| Check | Static patterns | Dynamic patterns |
|---|---|---|
| $uc_1$ | $\{uv_1, ..., uv_{u1}\}$ | $<ub_1, utb_1, fb_1>$ |
| ... | ... | ... |
| $uc_L$ | $\{uv_1, ..., uv_{uL}\}$ | $<ub_L, utb_L, ufb_L>$ |
| $sc_1$ | $\{sv_1, ..., sv_{s1}\}$ | $<sb_1, stb_1, sfb_1>$ |
| ... | ... | ... |
| $sc_K$ | $\{sv_1, ..., sv_{sK}\}$ | $<sb_K, stb_K, sfb_K>$ |

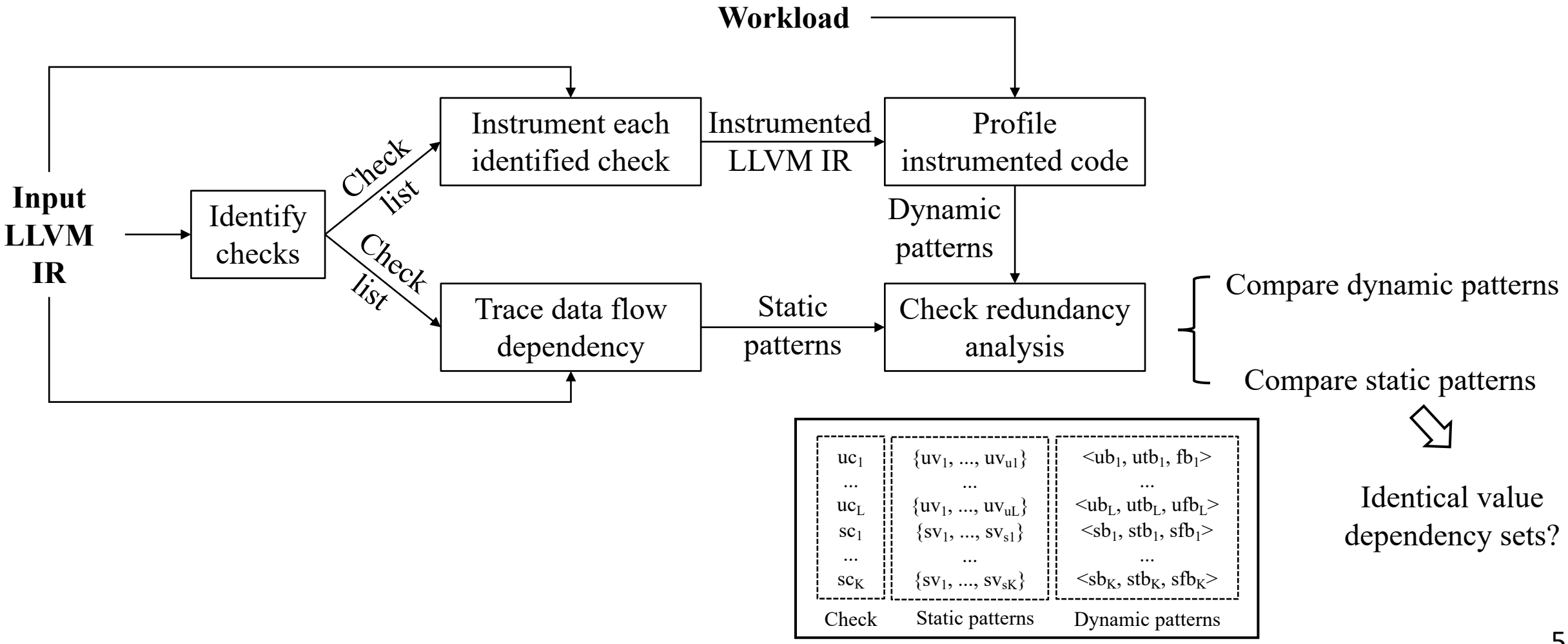
# Design: sanitizer check reduction



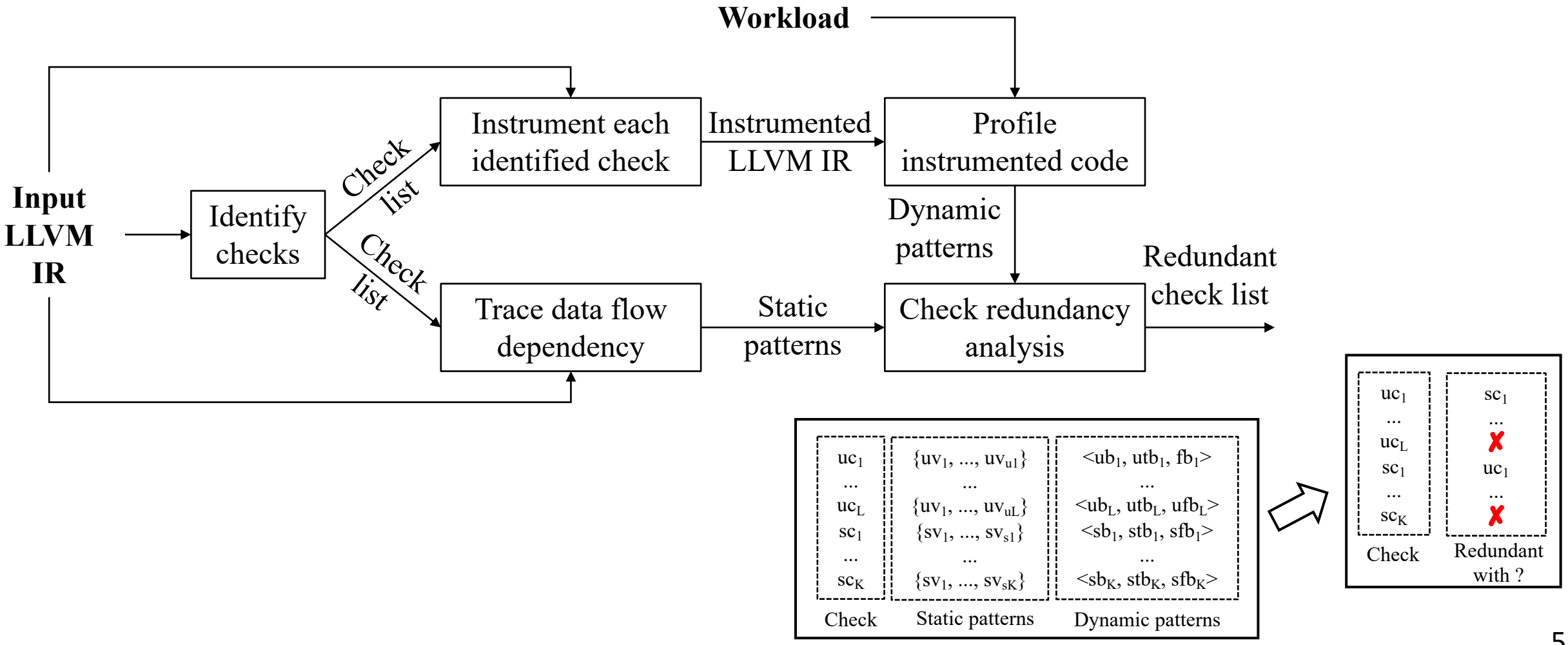
# Design: sanitizer check reduction



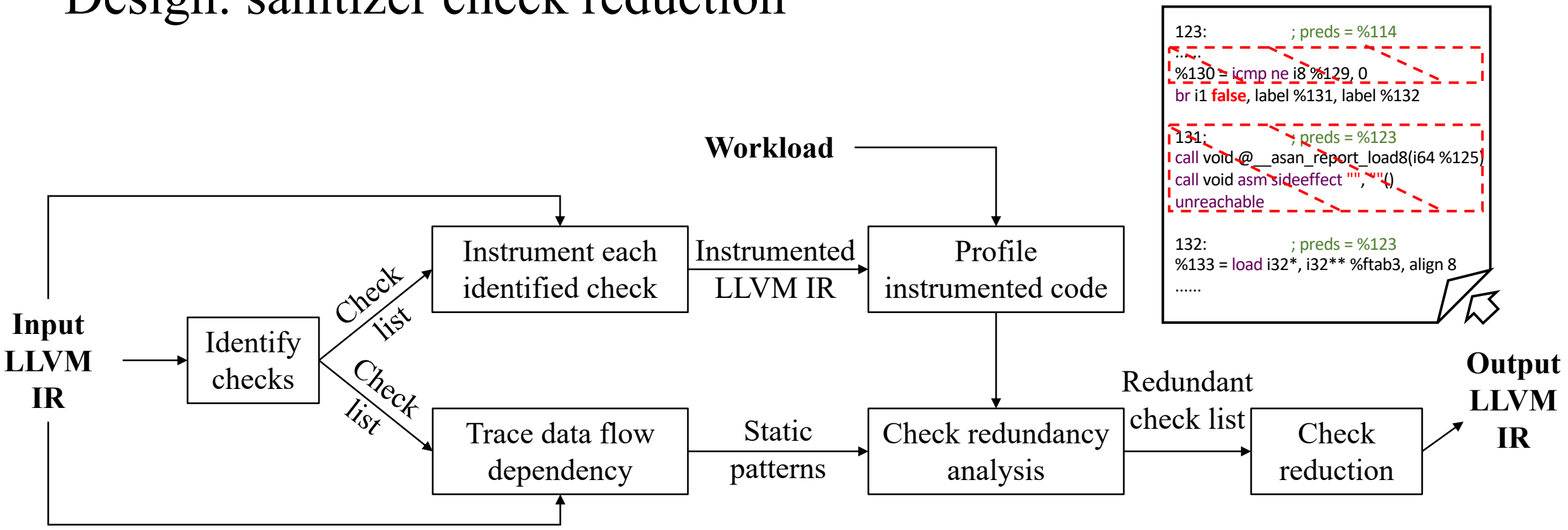
# Design: sanitizer check reduction



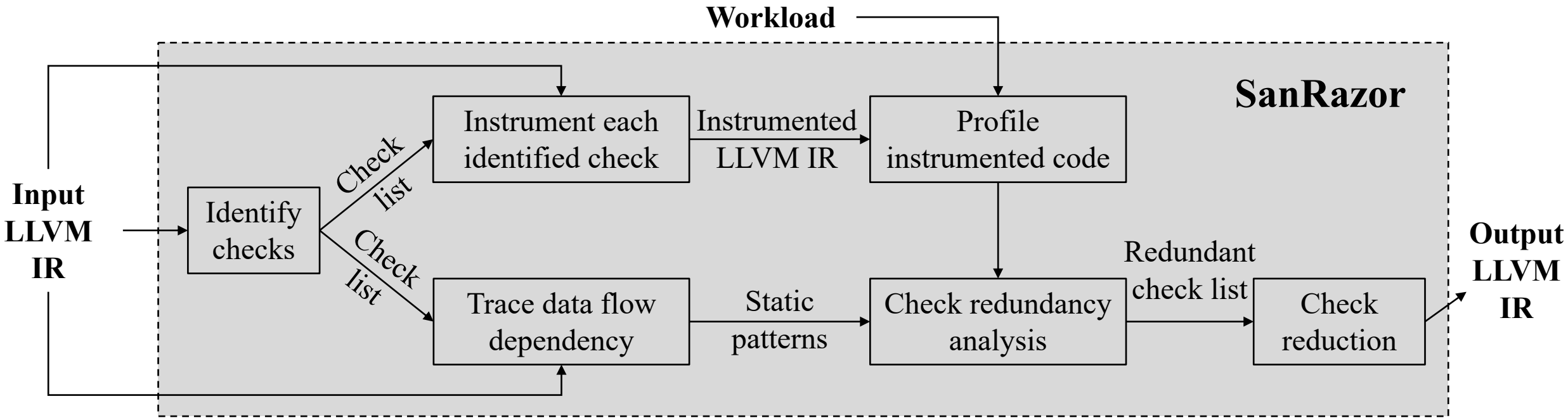
# Design: sanitizer check reduction



# Design: sanitizer check reduction



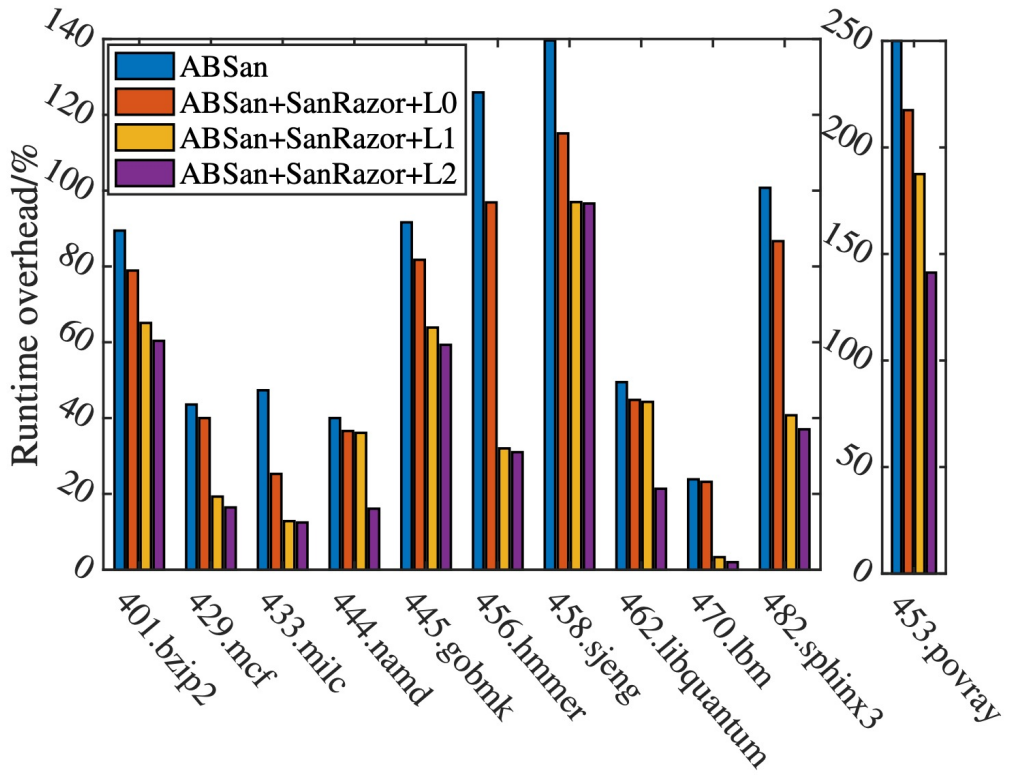
# Design and implementation

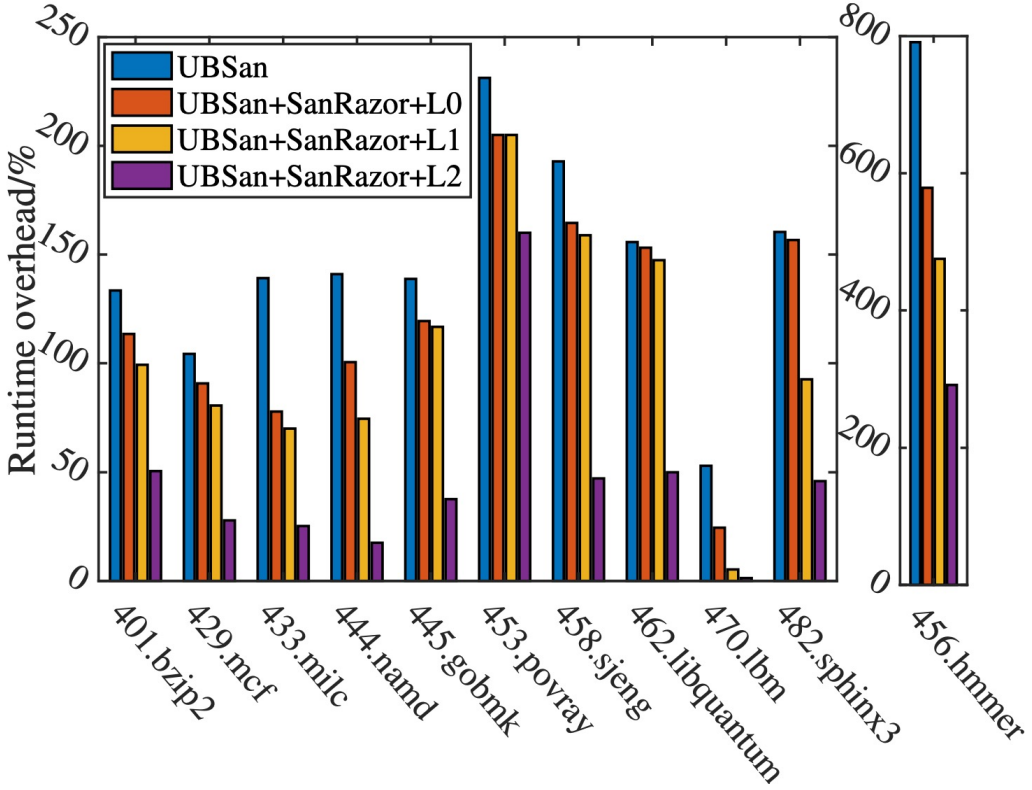



A general framework for effectively removing likely redundant sanitizer checks

Integrated into the LLVM framework → **SanRazor-clang**

# Evaluation: cost study




Runtime overhead (geo-mean)

74% ⟶ **28%** - 62%

160% ⟶ **37%** - 124%

# Evaluation: cost study

M1: number of removed sanitizer checks
M2: saved CPU cycles by reducing sanitizer checks

| Benchmark | ASan-$M_1$ | | | ASan-$M_2$ | | | UBSan-$M_1$ | | | UBSan-$M_2$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *L0* | *L1* | *L2* | *L0* | *L1* | *L2* | *L0* | *L1* | *L2* | *L0* | *L1* | *L2* |
| 401.bzip2 | 22.4% | 54.4% | 58.1% | 4.3% | 30.3% | 34.2% | 38.7% | 54.8% | 66.0% | 27.3% | 37.9% | 68.1% |
| 429.mcf | 10.2% | 53.0% | 60.9% | 3.0% | 46.6% | 60.1% | 35.0% | 51.8% | 76.2% | 37.8% | 47.6% | 86.0% |
| 445.gobmk | 5.2% | 23.4% | 26.6% | 7.2% | 33.7% | 41.0% | 12.6% | 21.6% | 51.3% | 21.4% | 23.3% | 73.9% |
| 456.hmmer | 5.9% | 11.7% | 13.1% | 14.4% | 70.3% | 70.4% | 8.2% | 11.0% | 14.8% | 49.2% | 60.7% | 78.3% |
| 458.sjeng | 5.9% | 12.6% | 13.4% | 4.4% | 34.4% | 36.7% | 12.1% | 18.3% | 51.0% | 20.7% | 25.2% | 79.2% |
| 462.libquantum | 7.4% | 16.3% | 22.6% | 0.8% | 1.4% | 2.4% | 12.7% | 15.6% | 26.9% | 0.8% | 0.8% | 58.8% |
| 433.milc | 23.5% | 32.5% | 33.5% | 35.8% | 80.9% | 82.7% | 27.6% | 42.2% | 54.6% | 51.0% | 60.6% | 83.6% |
| 444.namd | 6.4% | 18.9% | 24.0% | 10.2% | 29.8% | 57.7% | 8.7% | 16.0% | 26.2% | 40.4% | 54.1% | 84.8% |
| 470.lbm | 1.6% | 68.5% | 72.1% | 0.0% | 88.7% | 92.5% | 17.7% | 48.2% | 51.3% | 46.0% | 92.5% | 97.6% |
| 482.sphinx3 | 10.7% | 27.1% | 32.5% | 2.5% | 56.9% | 58.3% | 18.2% | 23.7% | 40.0% | 11.9% | 45.3% | 67.2% |
| 453.povray | 7.2% | 9.5% | 21.2% | 2.3% | 12.1% | 69.1% | 11.1% | 11.9% | 22.6% | 22.6% | 24.0% | 75.5% |

SanRazor eliminates up to 30% ASan checks and save 41% CPU cycles

SanRazor eliminates up to 39% UBSan checks and save 77% CPU cycles

# Evaluation: vulnerability detectability study

Select 38 CVEs from 10 commonly-used programs

| Software | CVE | | | SanRazor | | | ASAP | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Type | Sanitizer | N | *L0* | *L1* | *L2* | $Budget_0$ | $Budget_1$ | $Budget_2$ | $Budget_3$ |
| autotrace | signed integer overflow | UBSan | 8 | 8 | 8 | 6 | 6 | 8 | 8 | 8 |
| | left shift of 128 by 24 | UBSan | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | heap buffer overflow | ASan | 10 | 10 | 10 | 10 | 0 | 8 | 2 | 2 |
| imageworsener | divide-by-zero | UBSan | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| | index out of bounds | UBSan | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| lame | divide-by-zero | UBSan | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | heap buffer overflow | ASan | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| zziplib | heap buffer overflow | ASan | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 0 |
| libzip | user after free | ASan | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| graphicsmagick | heap use after free | ASan | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| libtiff | heap buffer overflow | ASan | 2 | 2 | 2 | 2 | 0 | 2 | 2 | 2 |
| | stack buffer overflow | ASan | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | divide-by-zero | UBSan | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| jasper | left shift of negative value | UBSan | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| potrace | heap buffer overflow | ASan | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| mp3gain | stack buffer overflow | ASan | 2 | 2 | 2 | 2 | 0 | 2 | 0 | 0 |
| | global buffer overflow | ASan | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| | null pointer dereference | ASan | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| **In total** | | | 38 | 38 | 37 | 33 | 15 | 33 | 24 | 23 |

SanRazor detects at least **33 out of the 38 CVEs**

# Application scenario

- Accelerate sanitization enabled programs in production usage
  - Keep the most useful checks in terms of discovering unique problems

# Application scenario

- Accelerate sanitization enabled programs in production usage
  - Keep the most useful checks in terms of discovering unique problems

- Combined with complementary approaches to further reduce overhead
  - SanRazor are generally orthogonal to other sanitizer tools
  - E.g. combining SanRazor with ASAP reduces the runtime cost to only 7% with a reasonable tradeoff of security

# Summary

- Present SanRazor, a novel and practical tool for sanitizer check reduction
- SanRazor is designed as a hybrid approach to remove likely redundant checks
- Evaluation shows that SanRazor effectively lowers the overhead caused by sanitizer, while still retaining high vulnerability detection capability

Thanks for listening!

Contact: jiangzha@usc.edu

Github repository: https://github.com/SanRazor-repo/SanRazor