SYMBIOTIC 4

BEYOND REACHABILITY

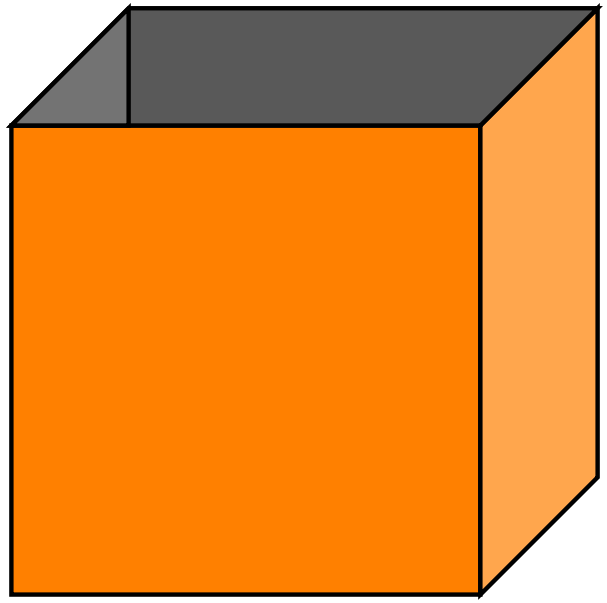
Marek Chalupa, Martin Jonáš, Jiri Slaby,
Jan Strejček, and Martina Vitovská

Masaryk University, Brno

PROGRAM SLICING

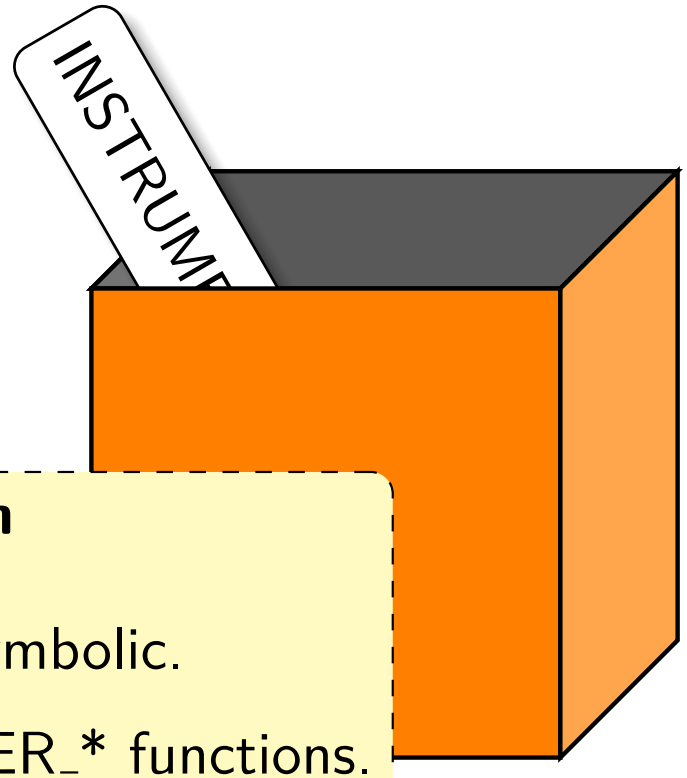
INSTRUMENTATION

SYMBOLIC EXECUTION



PROGRAM SLICING

SYMBOLIC EXECUTION

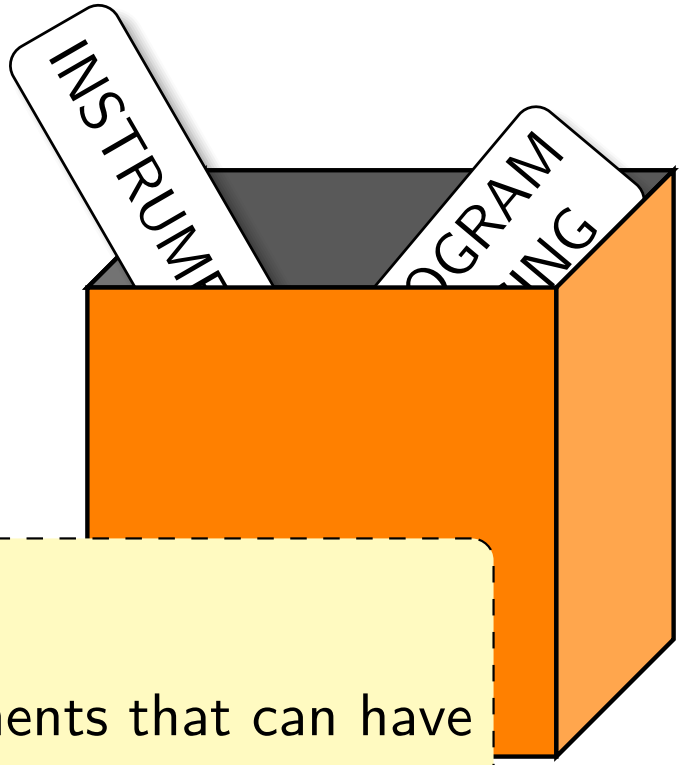


Instrumentation

Make memory symbolic.

Define __VERIFIER_* functions.

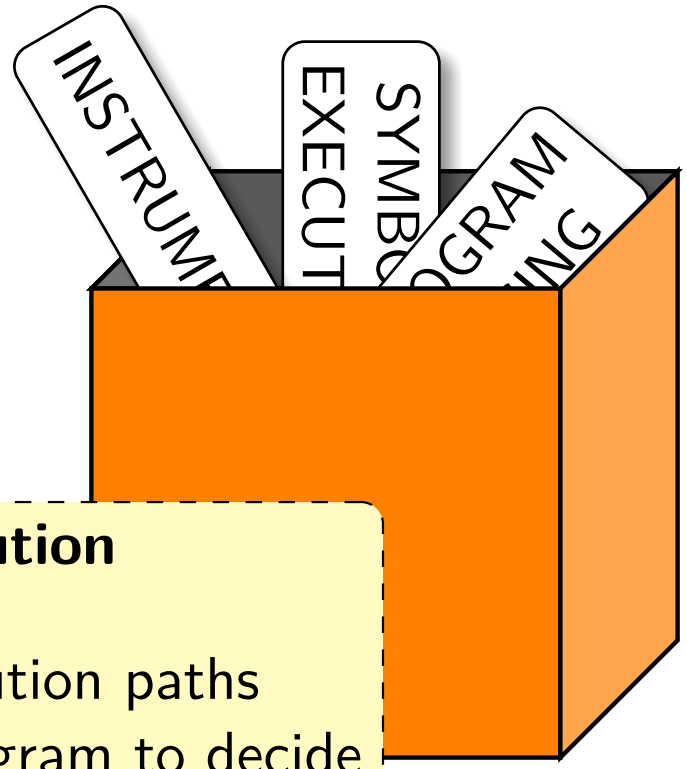
Insert runtime error checks.



SYMBOLIC EXECUTION

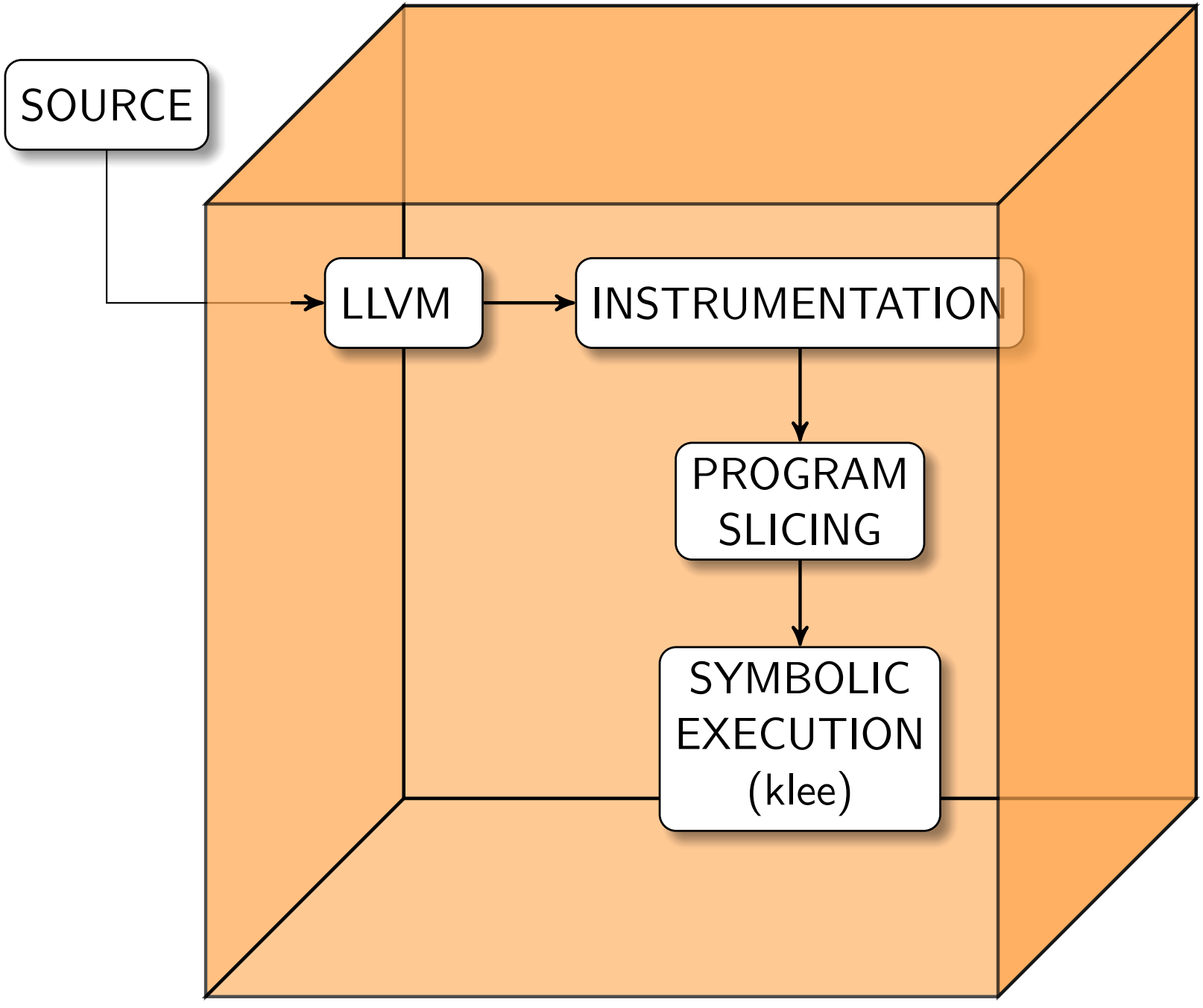
Program Slicing

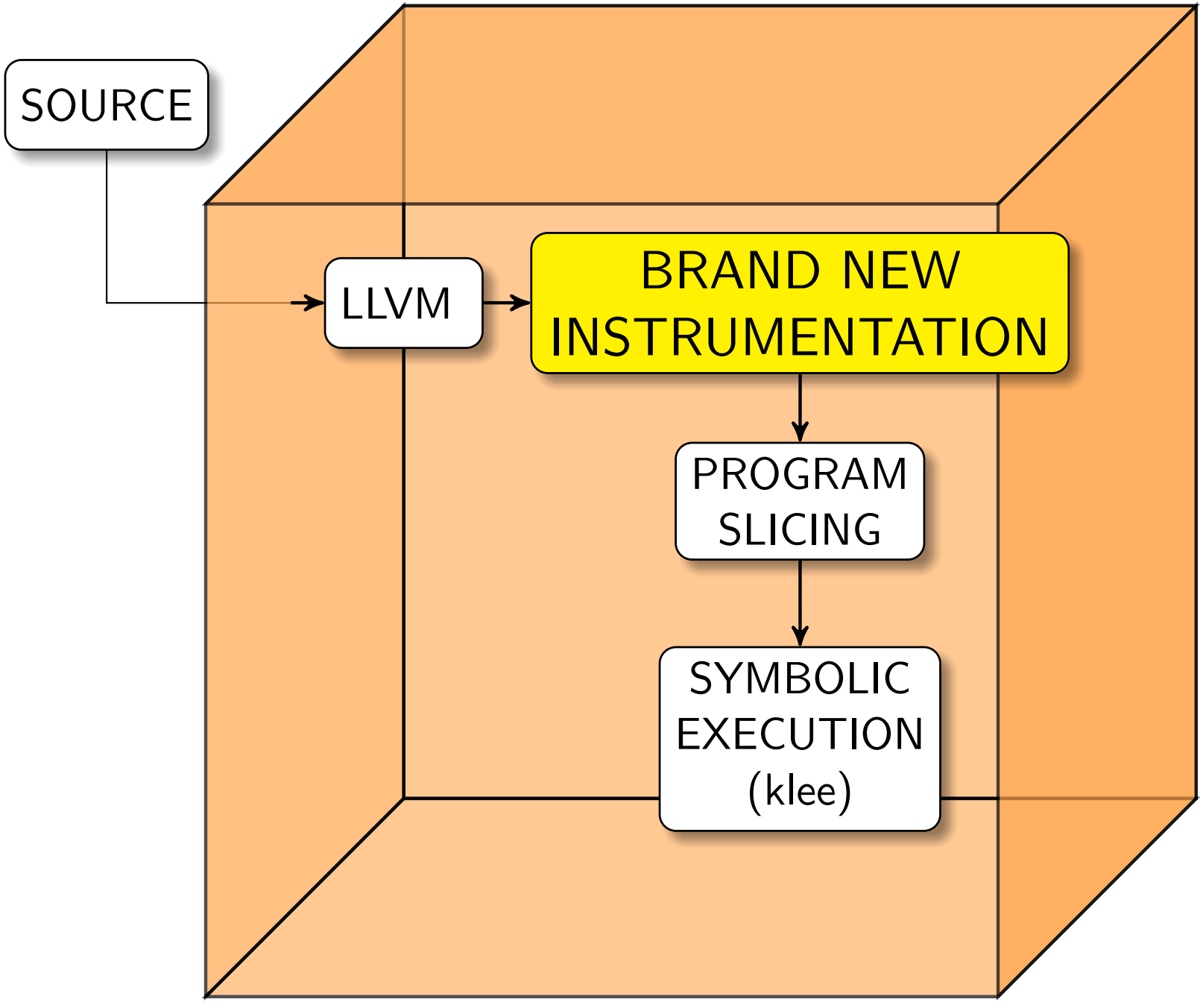
Keep only the statements that can have some effect on the inserted checks.



Symbolic execution

Explore all execution paths
in the sliced program to decide
whether an error is reachable.


SOURCE
LLVM
INSTRUMENTATION
PROGRAM
SLICING
SYMBOLIC
EXECUTION
(klee)


SOURCE
LLVM
BRAND NEW
INSTRUMENTATION
PROGRAM
SLICING
SYMBOLIC
EXECUTION
(klee)

New Instrumentation

- flexible – instrumentation rules in JSON
- a rule consists of
  - a pattern
  - a code to insert
  - a condition – checked by a query to a static analysis

```
"find": [
            {
              "instruction": "store",
              "operands": ["<x>", "<p>"]
            }
        ],
"insert": ["__INSTR_check_pointer", "<p>"],
"where": "before",
"condition": ["!isValidPointer", "<p>"],
```

New Instrumentation – Example

```
int array[10];

        ⋮

array[3] = 1;

        ⋮

n = input();

array[n] = n;
```

```
"find": [
            {
              "instruction": "store",
              "operands": ["<x>", "<p>"]
            }
        ],
"insert": ["__INSTR_check_pointer", "<p>"],
"where": "before",
"condition": ["!isValidPointer", "<p>"],
```

New Instrumentation – Example

```
int array[10];

        ⋮

array[3] = 1;

        ⋮

n = input();
__INSTR_check_pointer(&array[n]);
array[n] = n;
```

```
"find": [
          {
            "instruction": "store",
            "operands": ["<x>", "<p>"]
          }
        ],
"insert": ["__INSTR_check_pointer", "<p>"],
"where": "before",
"condition": ["!isValidPointer", "<p>"],
```

query

no check needed

pointer analysis

New Instrumentation

`__INSTR_check_*` functions are

- implemented in C
- compiled to LLVM and linked to the program

SV-COMP 2017 Results - MemSafety

	tool	score	CPU time [h]	solved tasks
1	PredatorHP	319	0.82	219
2	UAutomizer	308	1.9	145
3	Symbiotic	304	**0.08**	**233**

Conclusion

- Combining instrumentation (runtime checks) with static analysis is easy but powerful technique
- Program slicing can further improve the performance
- This approach is competitive with state-of-the-art tools for checking memory safety

Conclusion

- Combining instrumentation (runtime checks) with static analysis is easy but powerful technique
- Program slicing can further improve the performance
- This approach is competitive with state-of-the-art tools for checking memory safety

https://github.com/staticafi/symbiotic