Binary Decompilation to LLVM IR

Sandeep Dasgupta, Joshua Cranmer, Edward Schwartz (CERT) and Vikram Adve

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Motivation and Goals

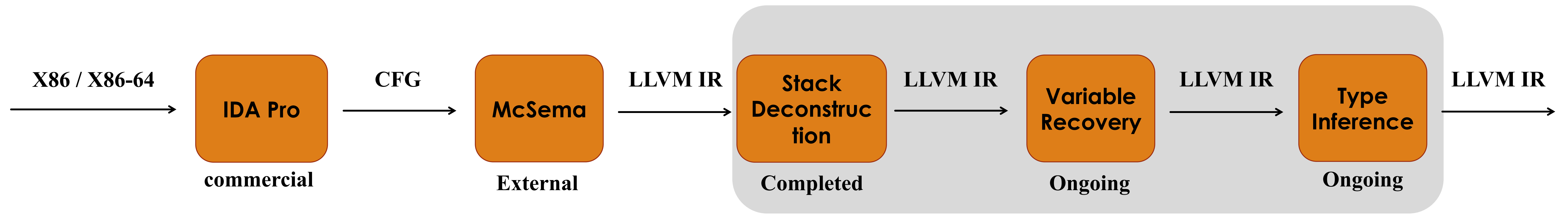
ALLVM: Represent *all* software components as LLVM IR, which provides new opportunities for analysis and optimization.

What about the software components which are in binary?

Goals

- Translate binary to *"rich"* LLVM IR where richness includes
  - Variable and type recovery
  - Per procedure stack frame recovery
  - Function signature recovery

Our Approach



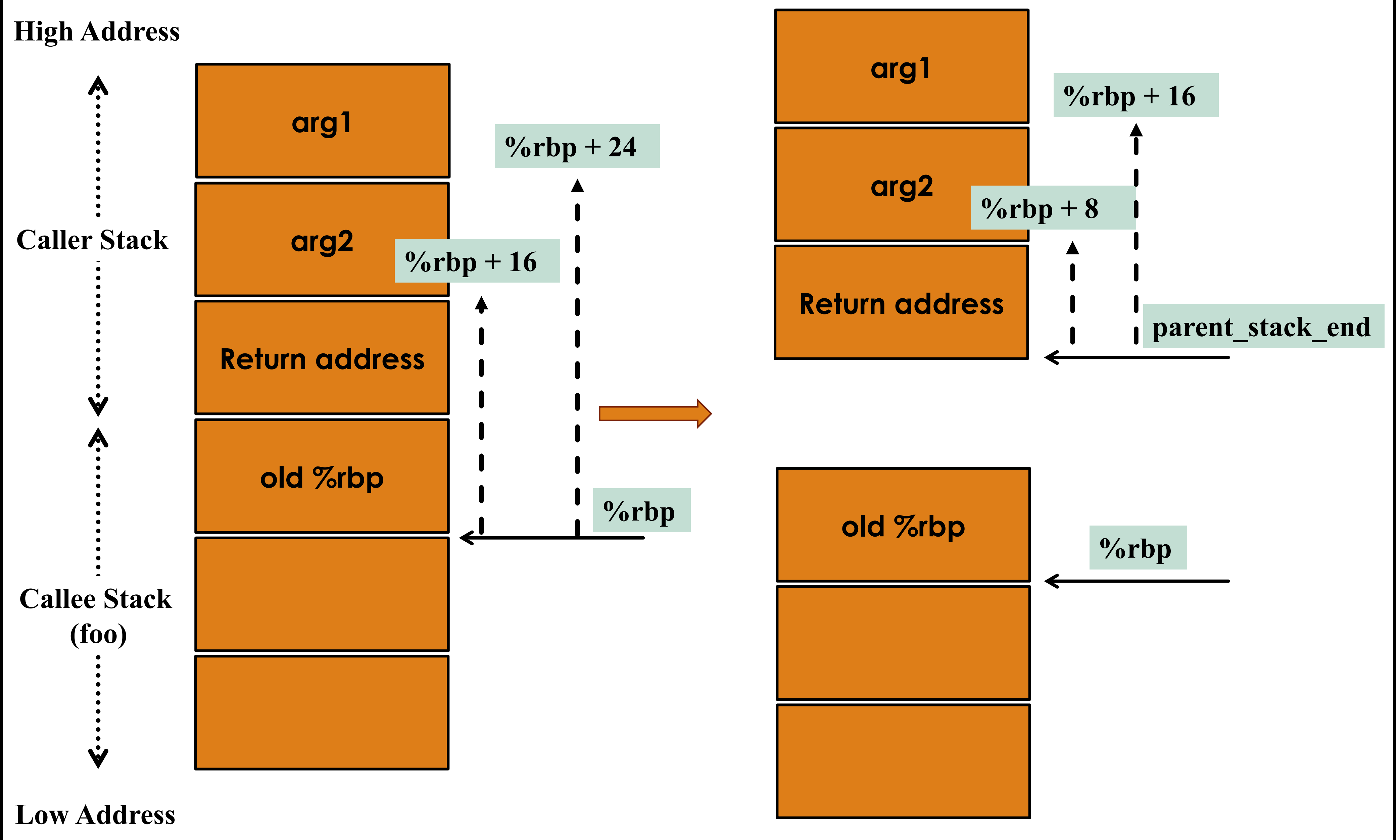
McSema Features

- Open Sourced
- Fully functional Output IR
- Plug & play CFG recovery

McSema Limitations

- Hardware registers not distinguishable from memory
- Process stack represented as flat array
- Variables not identified.
- No usable type information

Stack Deconstruction



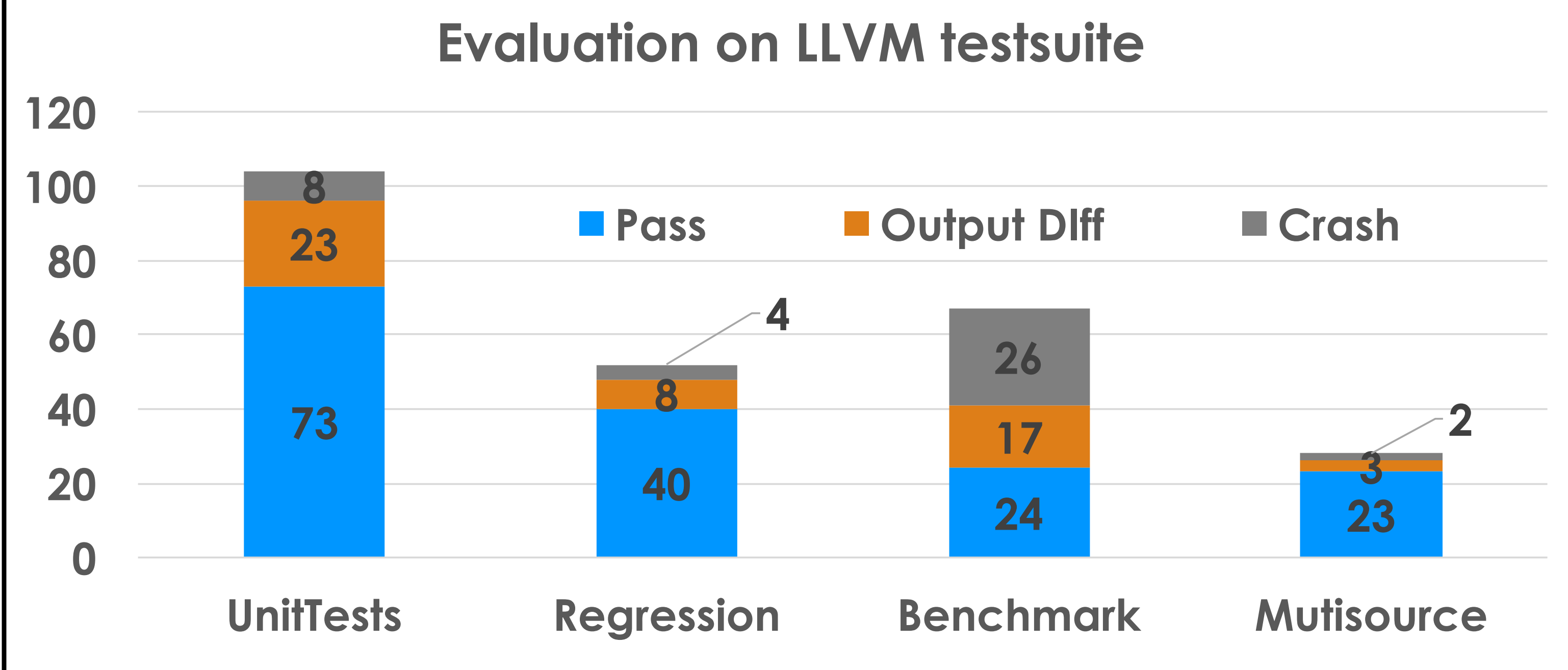
```
foo:
   push %rbp
   mov %rsp, %rbp
   mov 16(%rbp), %rax
   mov 24(%rbp), %r10
```

Before

```
foo (parent_rbp, parent_stack_end):
   push %parent_rbp
   mov %rsp, %rbp
   mov 8(%parent_stack_end), %rax
   mov 16(%parent_stack_end), %r10
```

After

Evaluation



- Some cases, not shown here, are unsupported by McSema.
- For unsupported cases, planning to include the inline assembly in the IR.

Lessons

- McSema generated IR is not amenable to pointer analysis due to `ptrtoint` or `inttoptr` casts and conservativeness of the available pointer analysis.
- Transformed the IR based on type information and using available analysis into more succinct form which can assist pointer analysis to some degree.

Related Work

	Devine	Retypd	Hex-Rays	BAP	TIE	BitBlaze	Second-Write	RevGen	Rev.ng	McSema	Allin
LLVM IR	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
Open Sourced	✗	✗	✗	✓	✗	✓	✗	✓	✓	✓	✓
Rich	✓	✓	✓	✗	✓	✗	✓	?	✗	✗	~

~ : Under progress   ? : Not quite sure