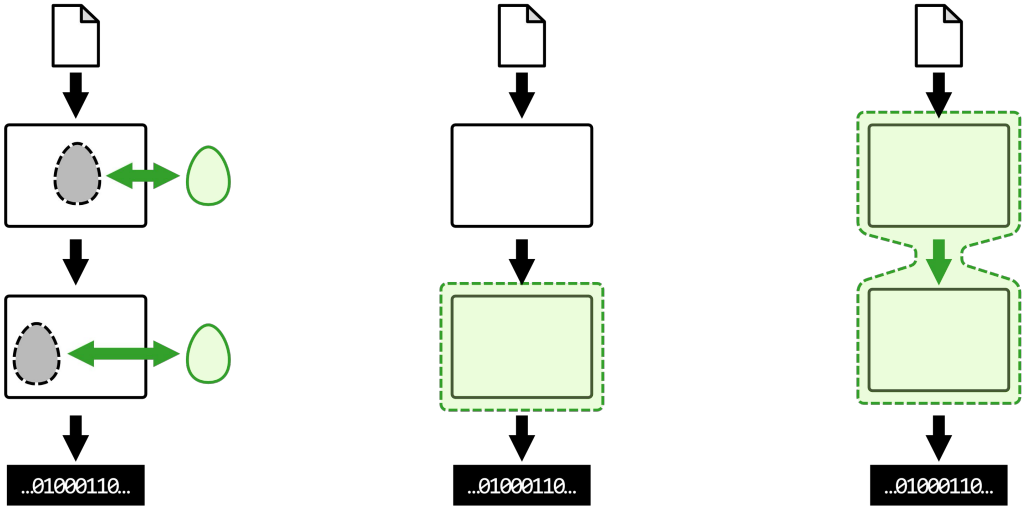


eqsat

E-Graphs as a Persistent Compiler Abstraction

Jules Merckx
Ghent University

Alexandre Lopoukhine
University of Cambridge

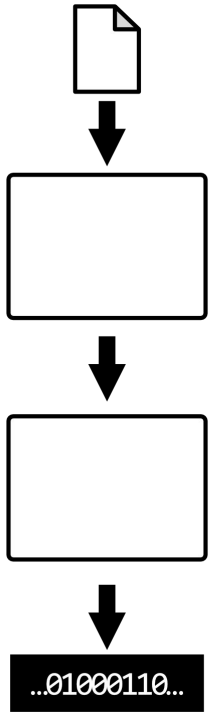
Jianyi Cheng
University of Edinburgh

Sam Coward
University College London

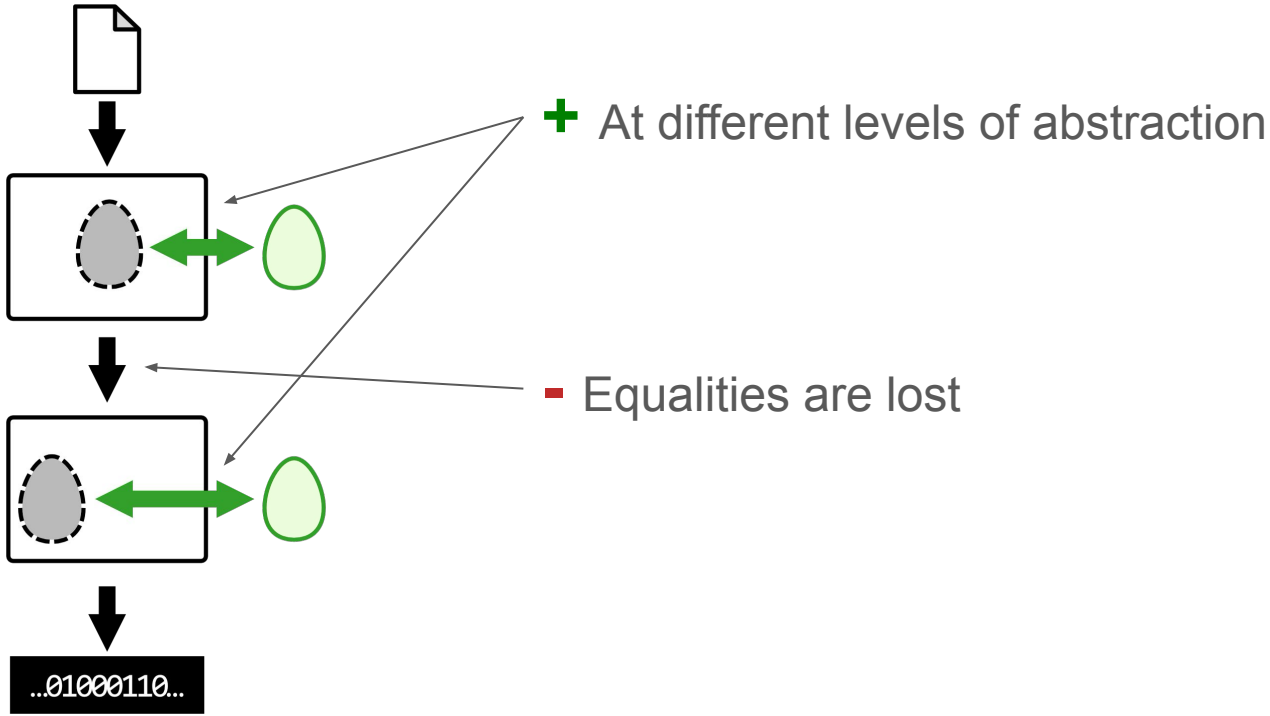
Bjorn De Sutter
Ghent University

Tobias Grosser
University of Cambridge

Code is transformed at different levels of abstraction

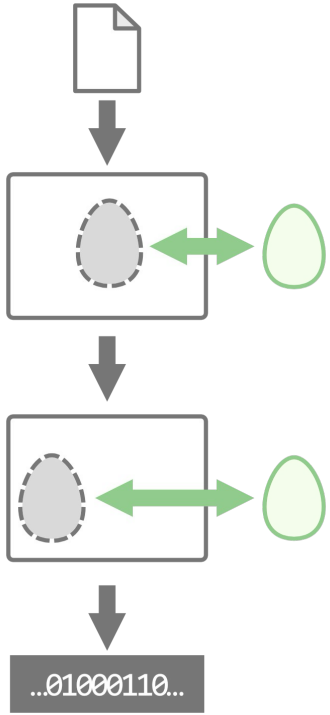


Egg is extensible but not integrated

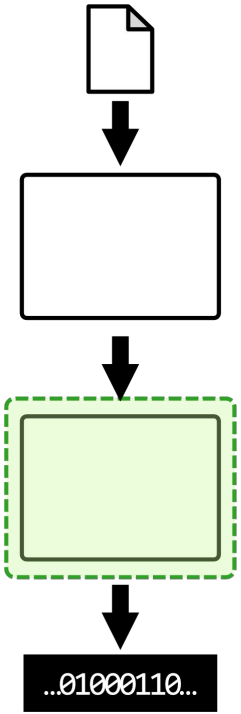


Egg

Cranelift offers tight compiler integration



Egg

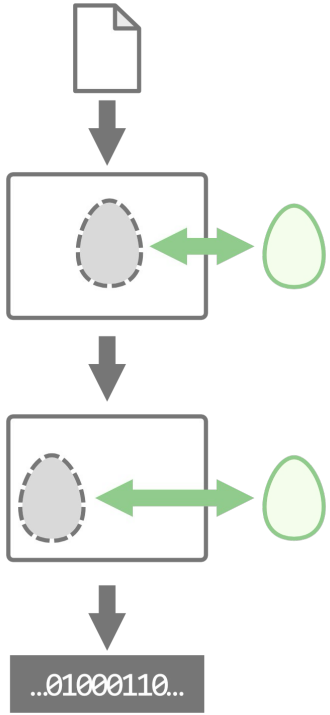


Cranelift

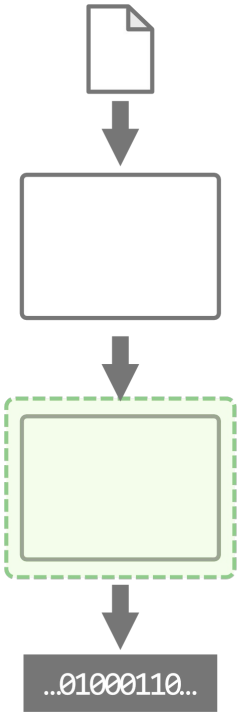
- Not at other levels of abstraction

+ Equality saturation is integrated

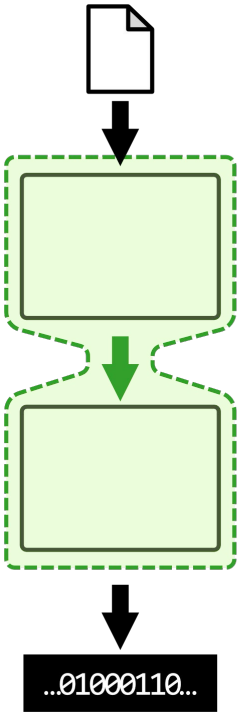
We aim to combine extensibility and integration



Egg

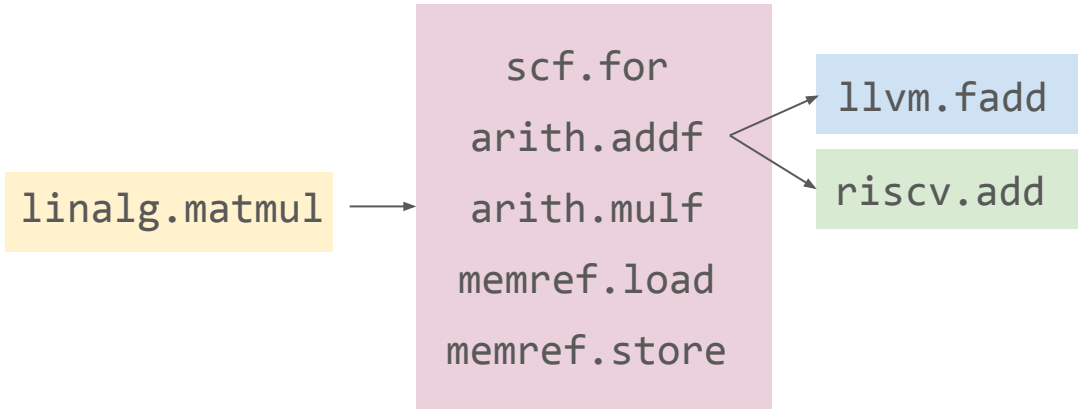


Cranelift



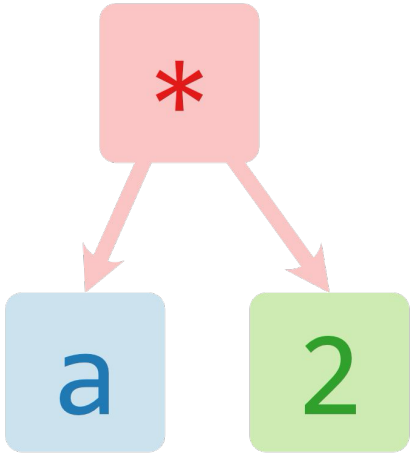
Our Work

MLIR: A User-Extensible Compiler Framework



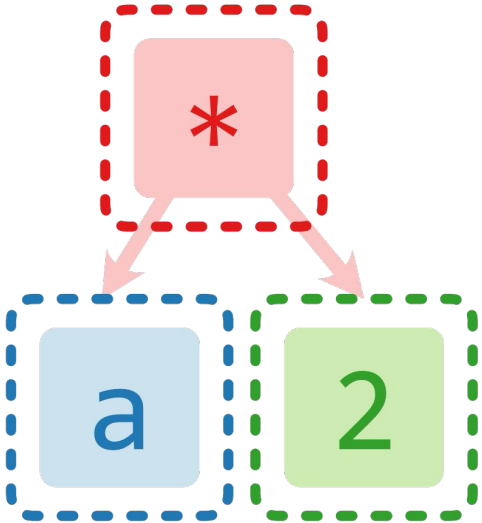
Start with arbitrary MLIR IR

```
func.func @f(%a : i64) -> i64 {
    %two = arith.constant 2 : i64

    %res = arith.muli %a   , %two    : i64

    func.return %res    : i64
}
```



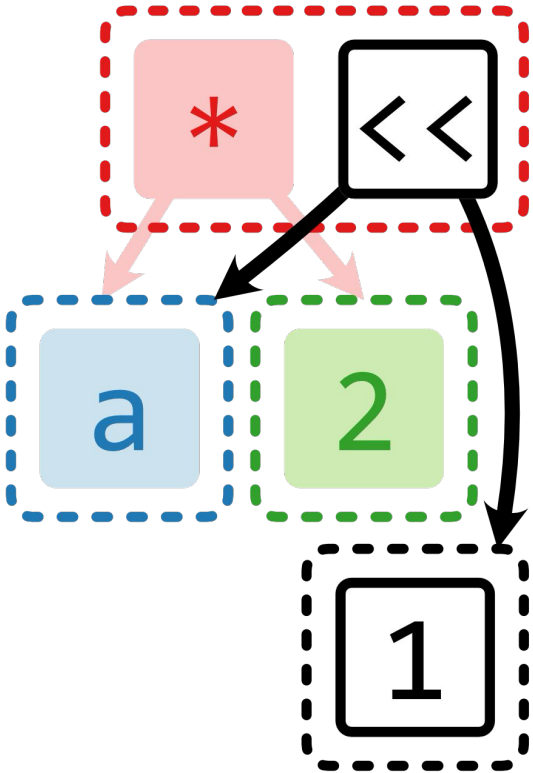
Embed e-classes as MLIR operations

```
func.func @f(%a : i64) -> i64 {
    %two = arith.constant 2 : i64


    %c_two = eqsat.eclass %two : i64
    %c_a = eqsat.eclass %a : i64
    %res = arith.muli %c_a, %c_two : i64

    %c_res = eqsat.eclass %res         : i64
    func.return %c_res : i64
}
```



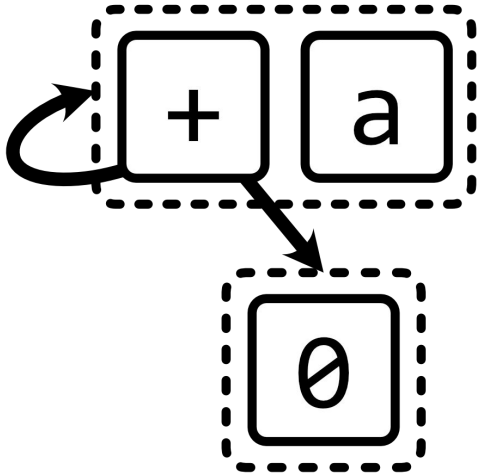
New e-nodes are introduced as operands to e-classes

```
func.func @f(%a : i64) -> i64 {
    %two = arith.constant 2 : i64
    %one = arith.constant 1
    %c_one = eqsat.eclass %one
    %c_two = eqsat.eclass %two : i64
    %c_a = eqsat.eclass %a : i64
    %res = arith.muli %c_a, %c_two : i64
    %res1 = arith.shli %c_a, %c_one
    %c_res = eqsat.eclass %res, %res1 : i64
    func.return %c_res : i64
}
```

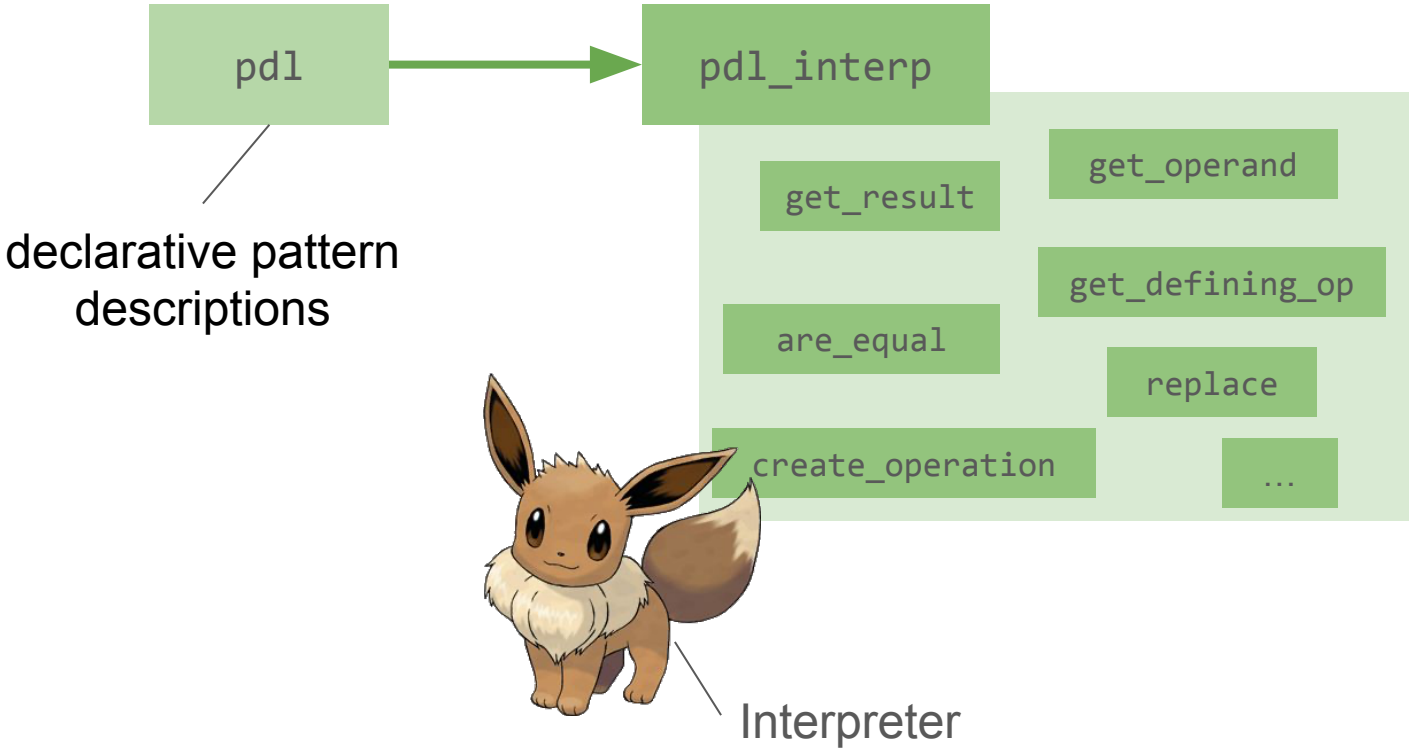


Cycles in graph regions

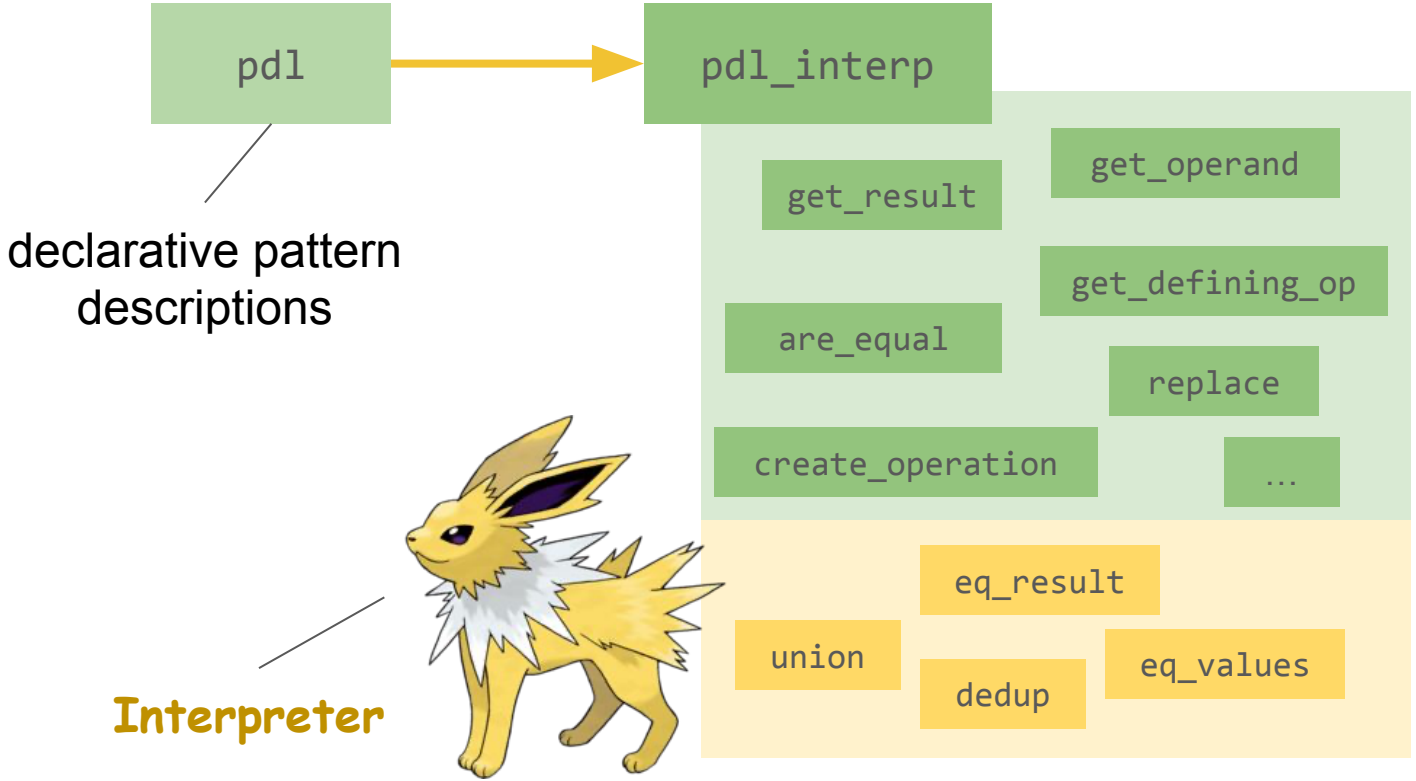
```
eqsat.egraph {
    %c_zero = eqsat.eclass %zero
    %sum = arith.addi %c_a, %c_zero
    %c_a = eqsat.eclass %sum, %a
}
```



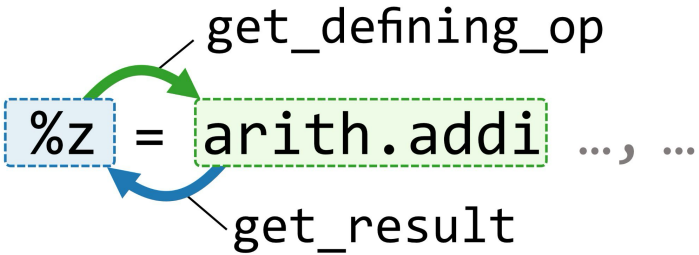
PDL: MLIR's declarative rewrite representation



We implement e-matching with small modifications



We have to take into account the use-def indirection



get_result

get_defining_op

We have to take into account the use-def indirection

introduce backtracking

get_result
eq_result

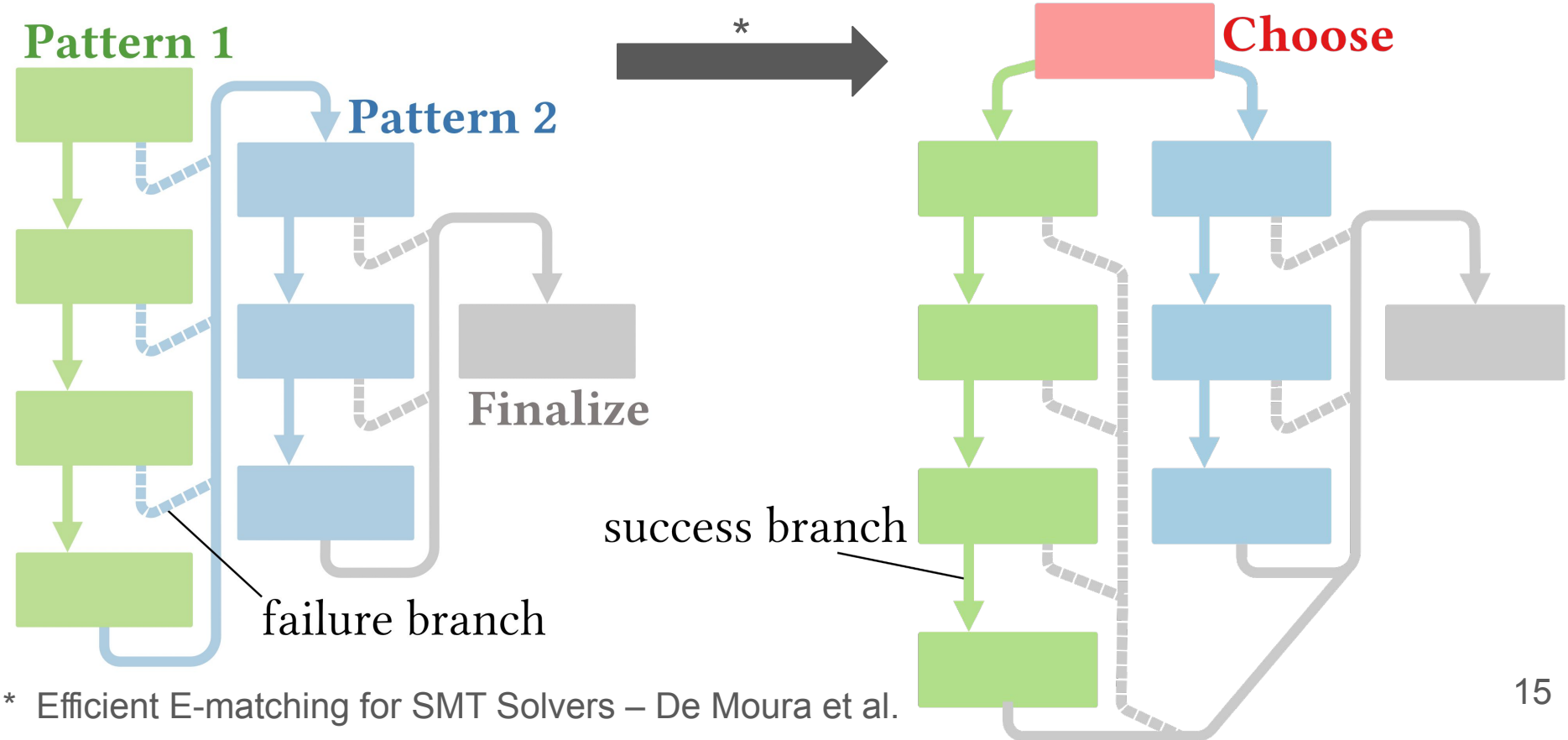
eq_values
get_defining_op

~~replace~~
union

create_operation
dedup

maintain hashcons

Naive `pdl_interp` can be exponentially slow



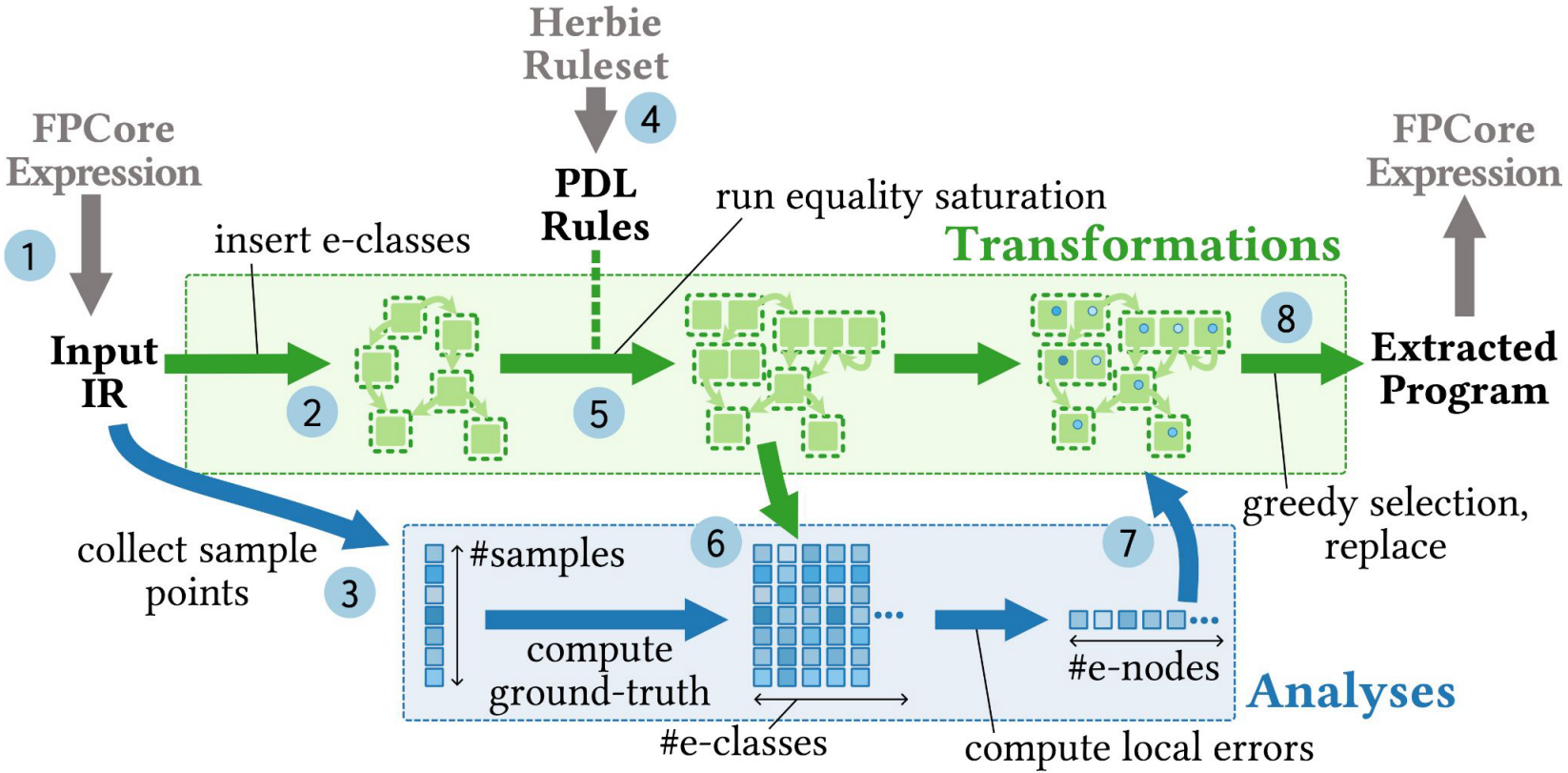
* Efficient E-matching for SMT Solvers – De Moura et al.

Reuse lattices for e-class analyses

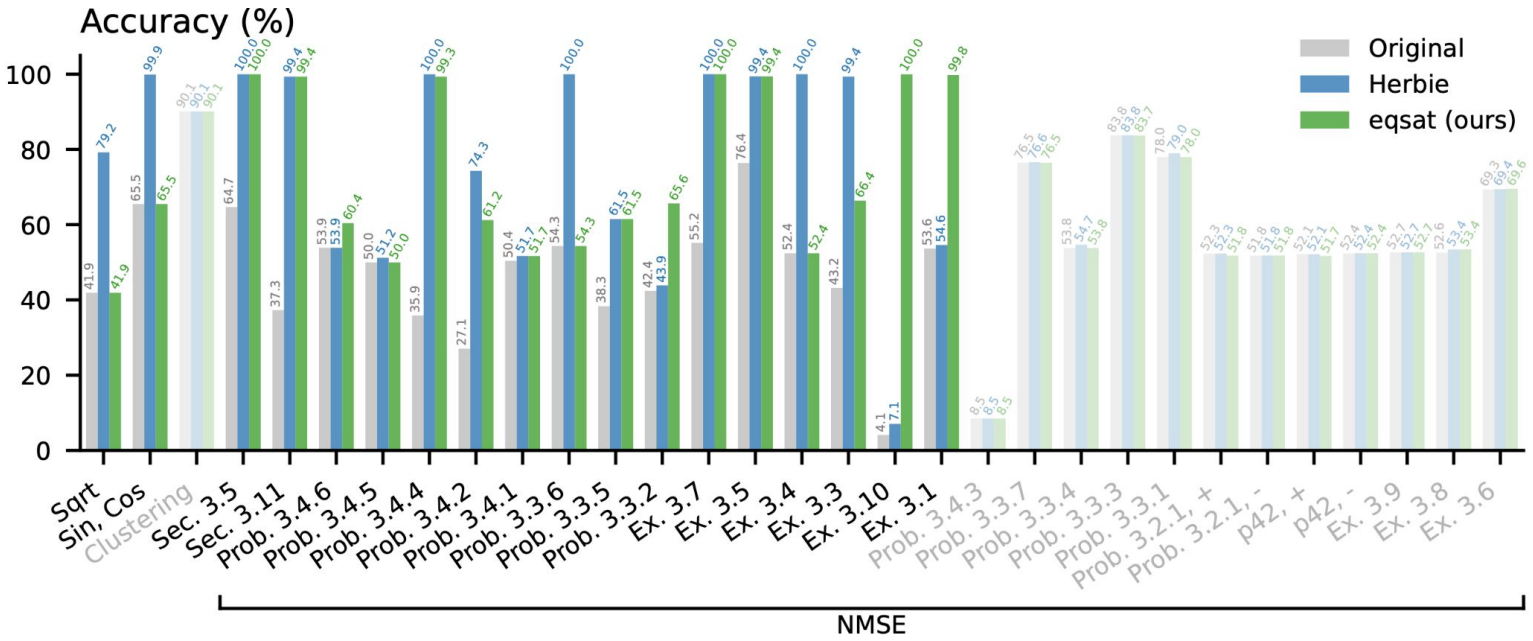
meet

In forward dataflow: join of eclass states is the meet of its operands

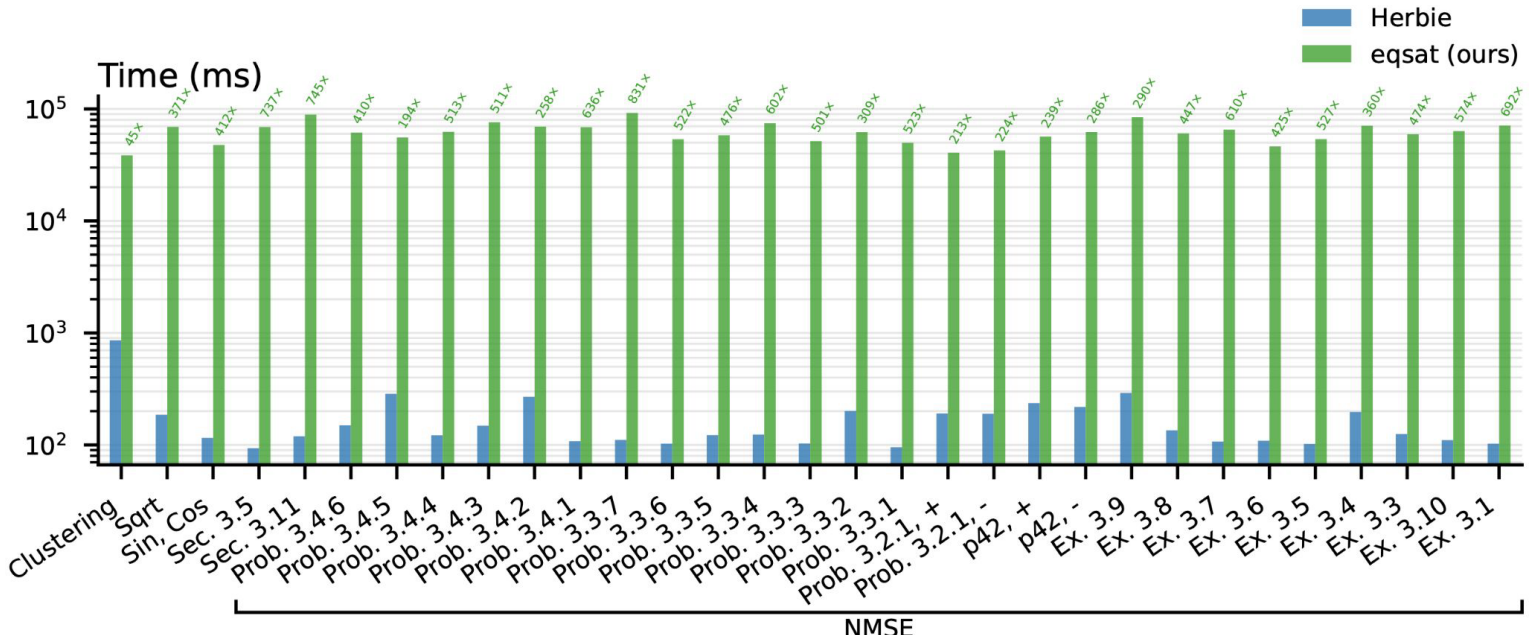
Proof of Concept: Herbie in MLIR



We match Herbie accuracy-wise



… But Much Slower*

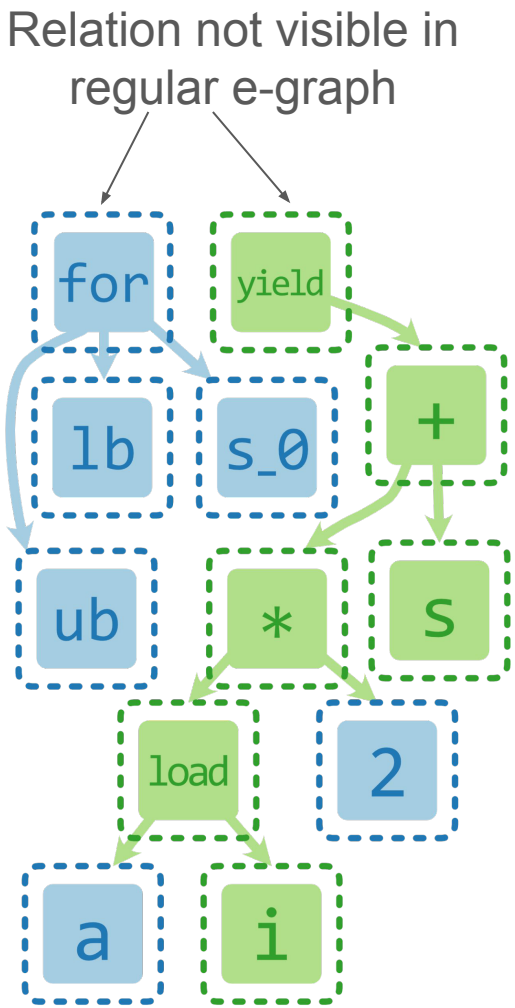


*Timing includes very inefficient arbitrary precision interval analysis

Control flow is kept intact

```
%s = scf.for %i = %lb to %ub
     iter_args(%s = %s_0) {
  %x = memref.load %a[%i]
  %term = arith.mulf %x, %two
  %s_new = arith.addf %s, %term
  scf.yield %s_new
}
```

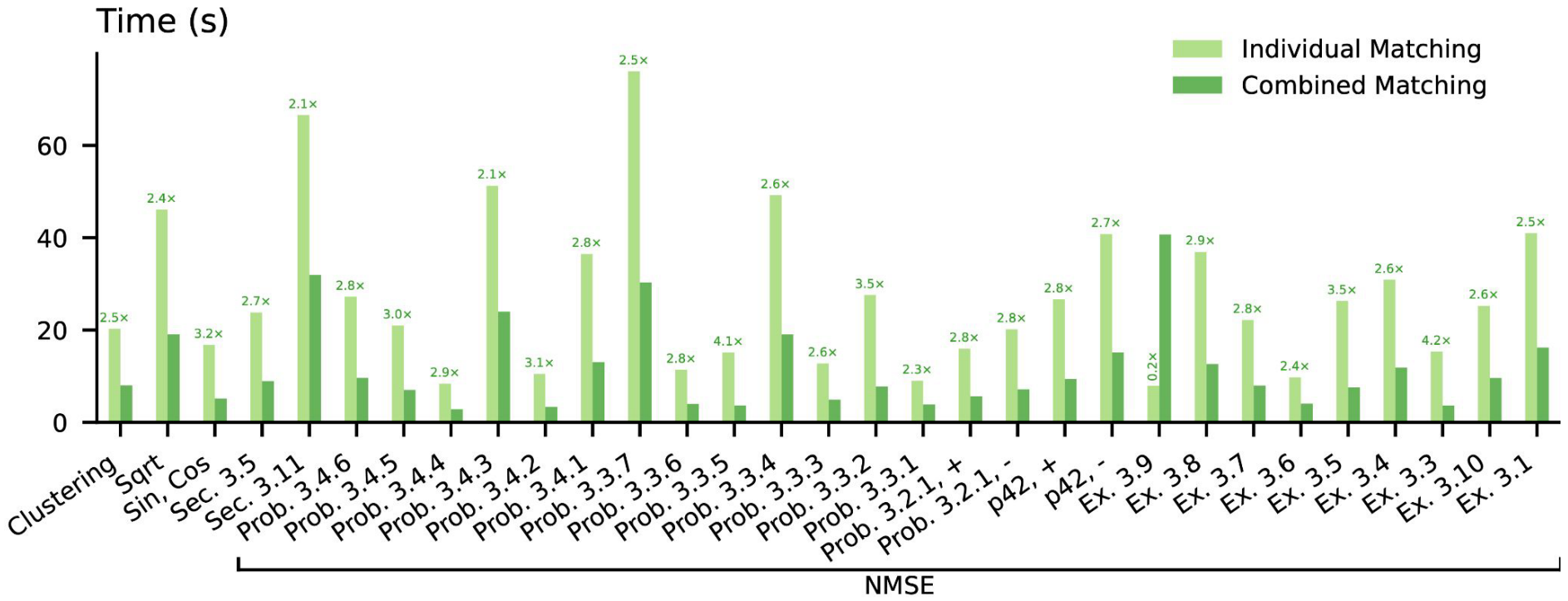
scoped hashcons?



Caveat: `pdl_interp` cannot yet match regions

Things we want to work on

- Cranelift-style acyclic e-graphs for instruction selection
- Large scale tensor-program optimizations
- Control-flow rewrites
- “Partial” e-graphs (only put some terms in an e-class)
  - Automatic e-graph insertion
- E-matching optimizations


Time (s)
Individual Matching
Combined Matching
0
20
40
60
2.5×
2.4×
3.2×
2.7×
2.1×
2.8×
3.0×
2.9×
2.1×
3.1×
2.8×
2.5×
2.8×
4.1×
2.6×
2.6×
3.5×
2.3×
2.8×
2.8×
2.8×
2.7×
0.2×
2.9×
2.8×
2.4×
3.5×
2.6×
4.2×
2.6×
2.5×
Clustering
Sqrt
Sin, Cos
Sec. 3.5
Sec. 3.11
Prob. 3.4.6
Prob. 3.4.5
Prob. 3.4.4
Prob. 3.4.3
Prob. 3.4.2
Prob. 3.4.1
Prob. 3.3.7
Prob. 3.3.6
Prob. 3.3.5
Prob. 3.3.4
Prob. 3.3.3
Prob. 3.3.2
Prob. 3.3.1
Prob. 3.2.1, +
Prob. 3.2.1, -
p42, +
p42, -
Ex. 3.9
Ex. 3.8
Ex. 3.7
Ex. 3.6
Ex. 3.5
Ex. 3.4
Ex. 3.3
Ex. 3.10
Ex. 3.1
NMSE

? Extending `pdl_interp` for region matching

? How to handle operations without results

```
memref.store %val, %A[%a, %b] : memref<8x?xi32, #layout>
```

? Which compiler passes benefit most

- Function call inlining
- Multi-level analyses
- …

? Applications

- Herbie-like floating point optimization
- Multi-level, end-to-end compilation
- …