MARCH 23, 2026 · ASPLOS 2026 · PITTSBURGH, PA

Architecture 2.0

Workshop on AI for Computing Systems Design

# EggMind: LLM-Driven Two-Dimensional Intelligence for Scalable Equality Saturation

**Youwei Xiao***, Chenyun Yin, Yun Liang

Peking University

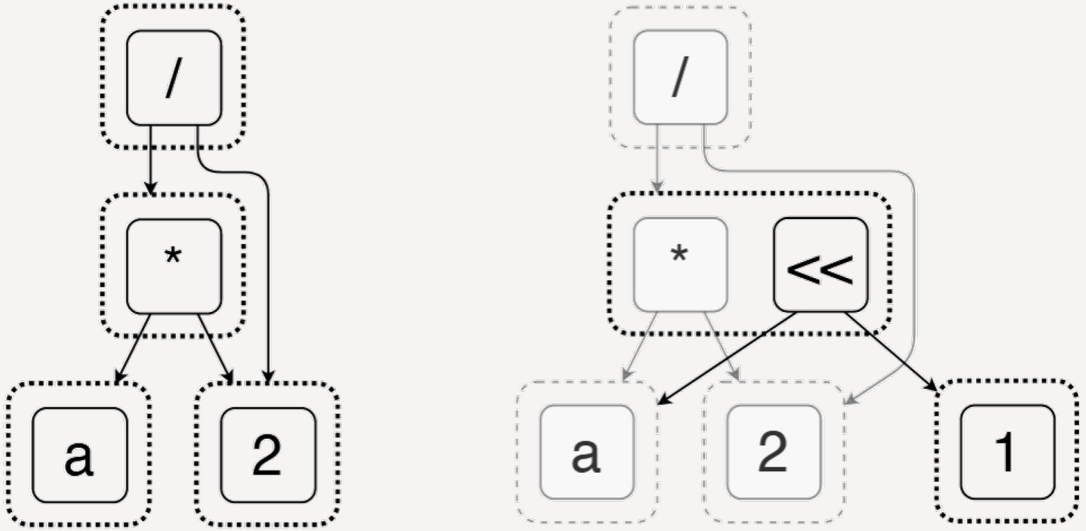
* shallwe@pku.edu.cn

# E-graph

- E-graphs compactly represent equivalent expressions
  - Each e-class includes equivalent e-nodes

E-node

E-class

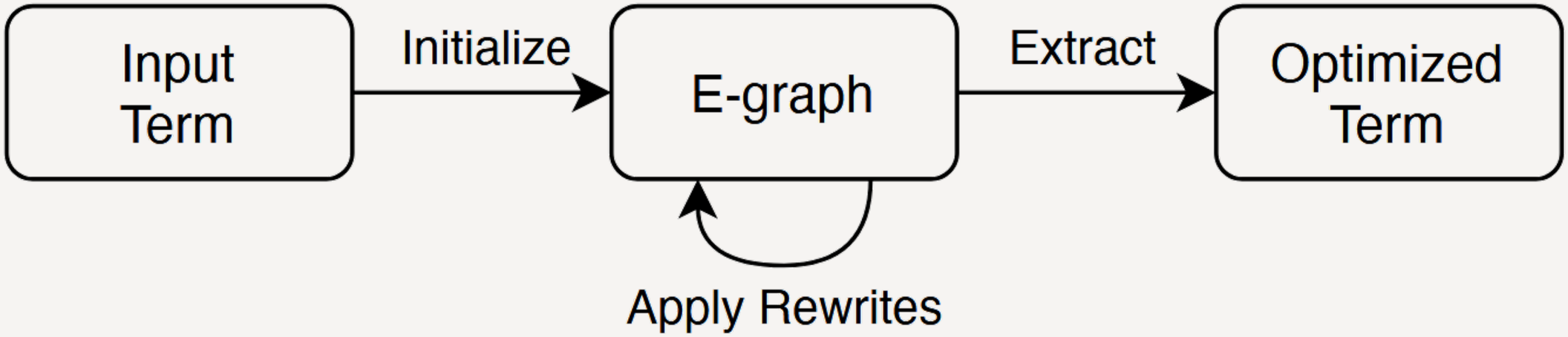



(a) Initial e-graph contains $(a \times 2)/2$.

(b) After applying rewrite $x \times 2 \rightarrow x \ll 1$.

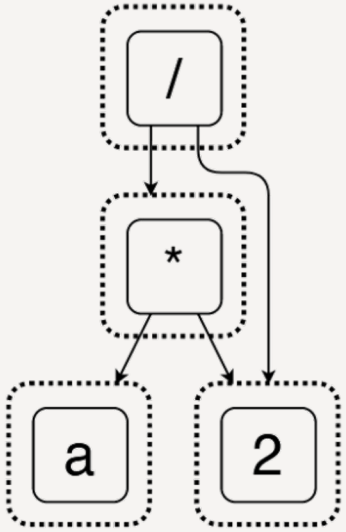
# Equality Saturation (EqSat)

- EqSat is the process of running a set of rewrite rules until a fixpoint or resource/time limit, and finally selects the optimal represented term.

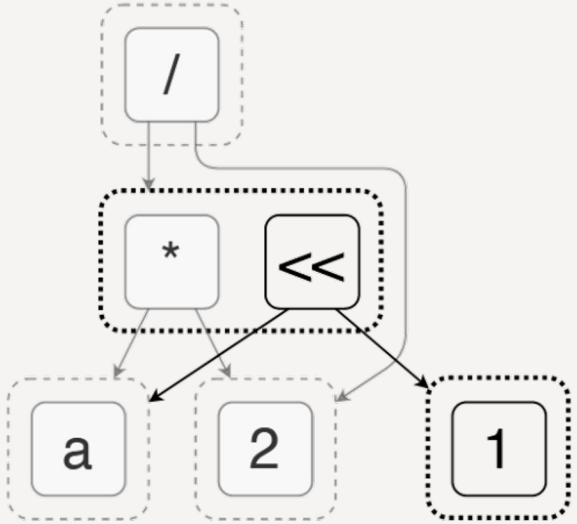



- This solves the *phase-ordering* problem in the compiler community.
  - Example: $(a \times 2)/2 \rightarrow (a \ll 1)/2$ prevents canceling out $2/2$.
  - $(a \times 2)/2 \rightarrow a \times (2/2) \rightarrow a \times 1 \rightarrow a$

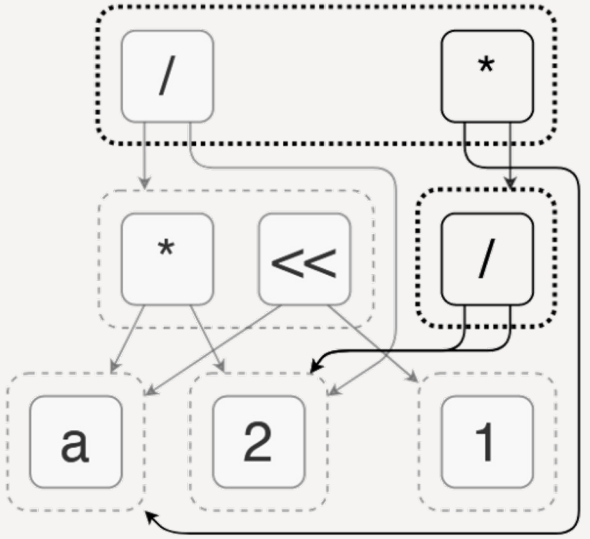
# Example: $(a \times 2)/2$



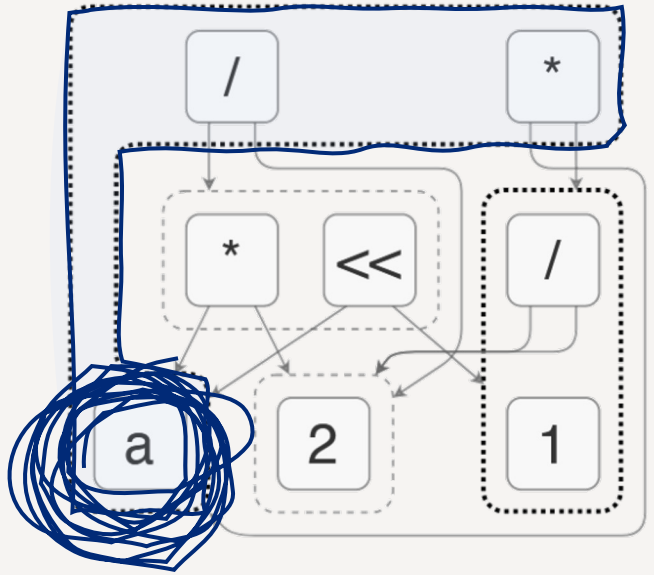

(a) Initial e-graph contains $(a \times 2)/2$.




(b) After applying rewrite $x \times 2 \rightarrow x \ll 1$.




(c) After applying rewrite $(x \times y)/z \rightarrow x \times (y/z)$.




(d) After applying rewrites $x/x \rightarrow 1$ and ~~$1 \times x$~~ $x \times 1 \rightarrow x$.

# Egg, Egglog

[1] Max Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, and P. Panchekha, "egg: Fast and extensible equality saturation," POPL 2021, doi: 10.1145/3434304.

[2] Y. Zhang *et al.*, "Better Together: Unifying Datalog and Equality Saturation," PLDI 2023, doi: https://doi.org/10.1145/3591239

- Egg[1] is an EqSat engine in Rust
- Egglog[2] is a logic programming language, so called "Unifying Datalog and Equality Saturation". Language + Engine.

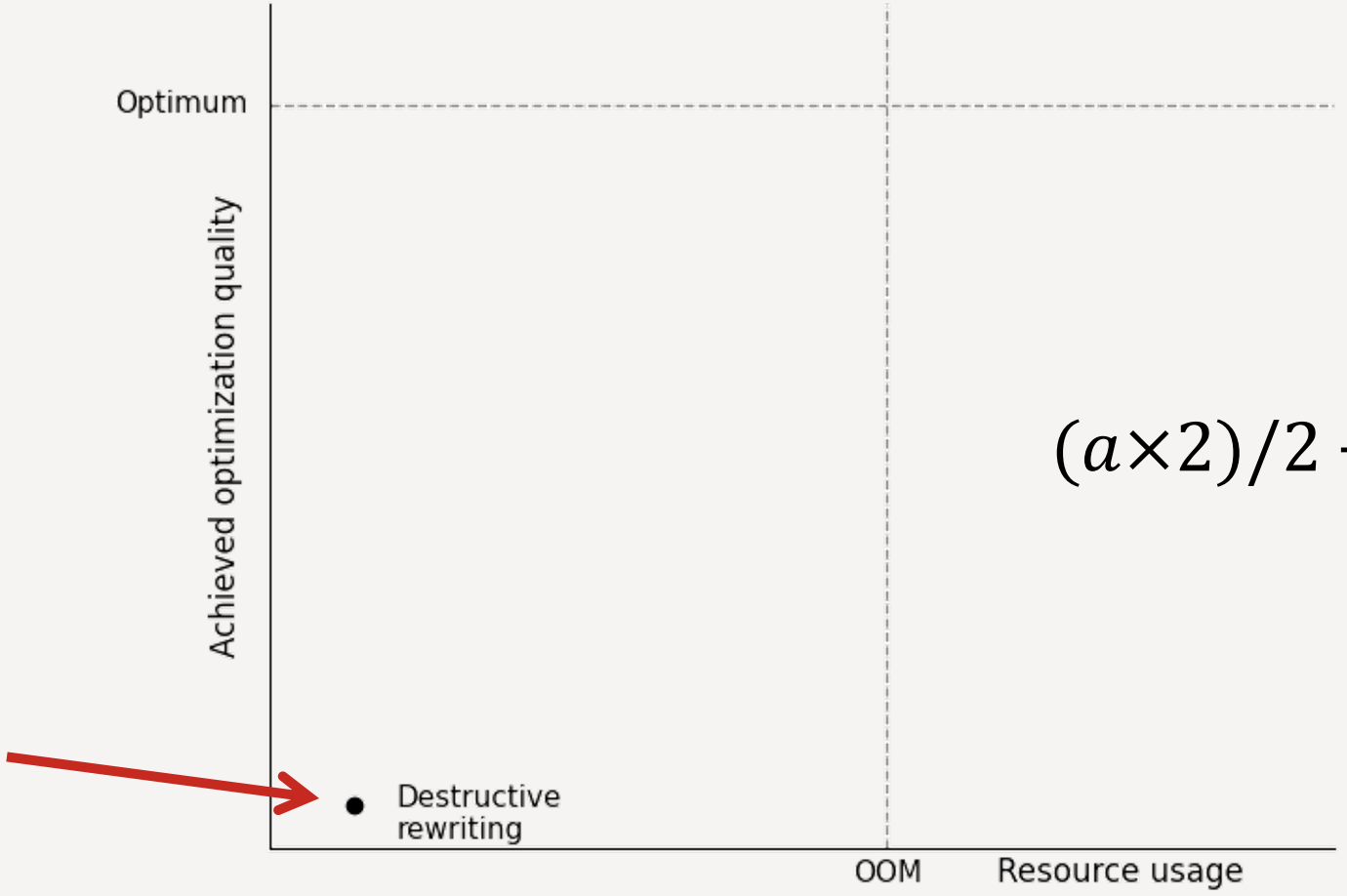
```
(datatype Expr
  (Num i64)
  (Var String)
  (Add Expr Expr)
  (Mul Expr Expr)
  (Div Expr Expr)
  (Shl Expr Expr))
```

```
(let expr (Div (Mul (Var "x") (Num 2)) (Num 2)))
(rewrite (Mul x (Num 2)) (Shl x (Num 1)))
(rewrite (Div (Mul x y) z) (Mul x (Div y z)))
(rewrite (Div x x) (Num 1))
(rewrite (Mul x (Num 1)) x)
(run 3)
(extract expr)
```

Output: `(Var "x")`

# Spectrum of Rewriting Techniques

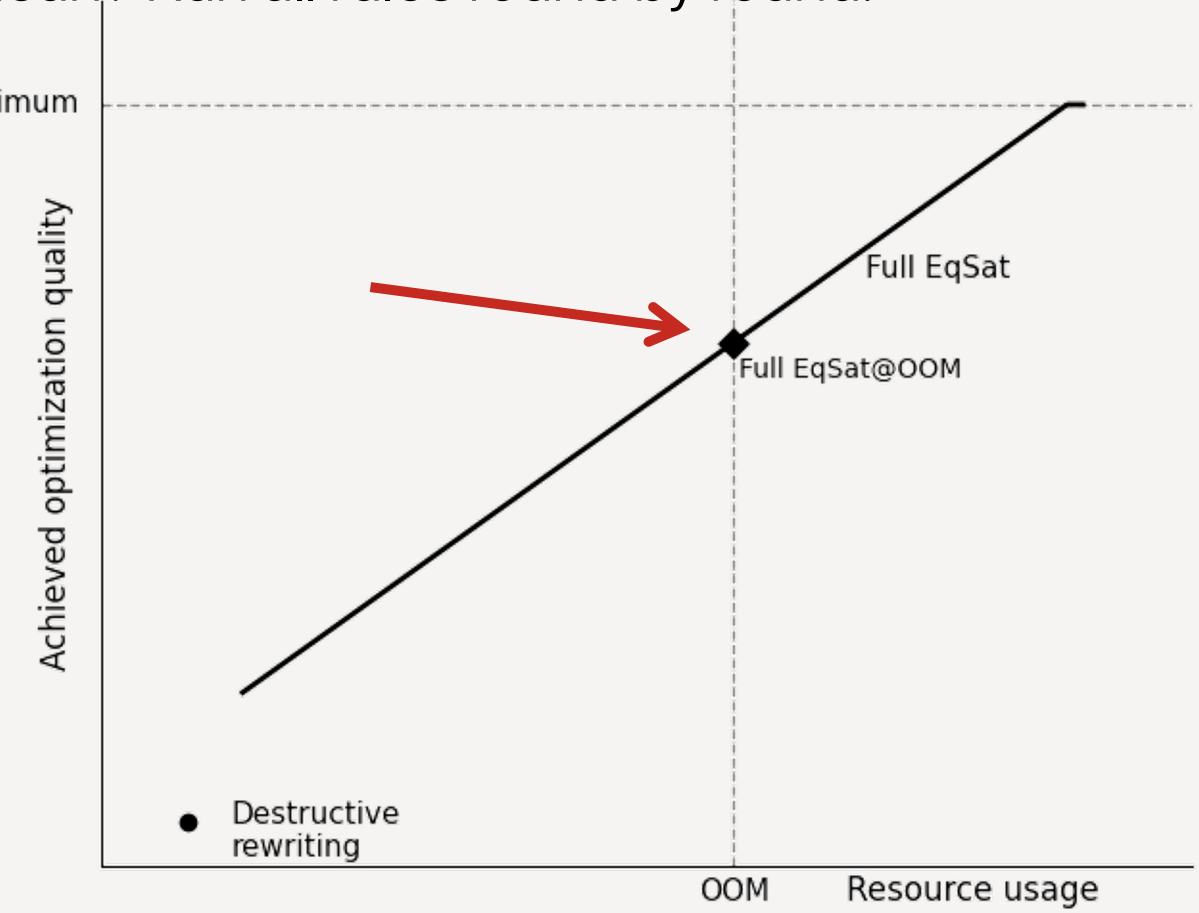
- Destructive rewriting faces phase-ordering problem




$$(a \times 2)/2 \rightarrow (a \ll 1)/2 \rightarrow ???$$

# Spectrum of Rewriting Techniques

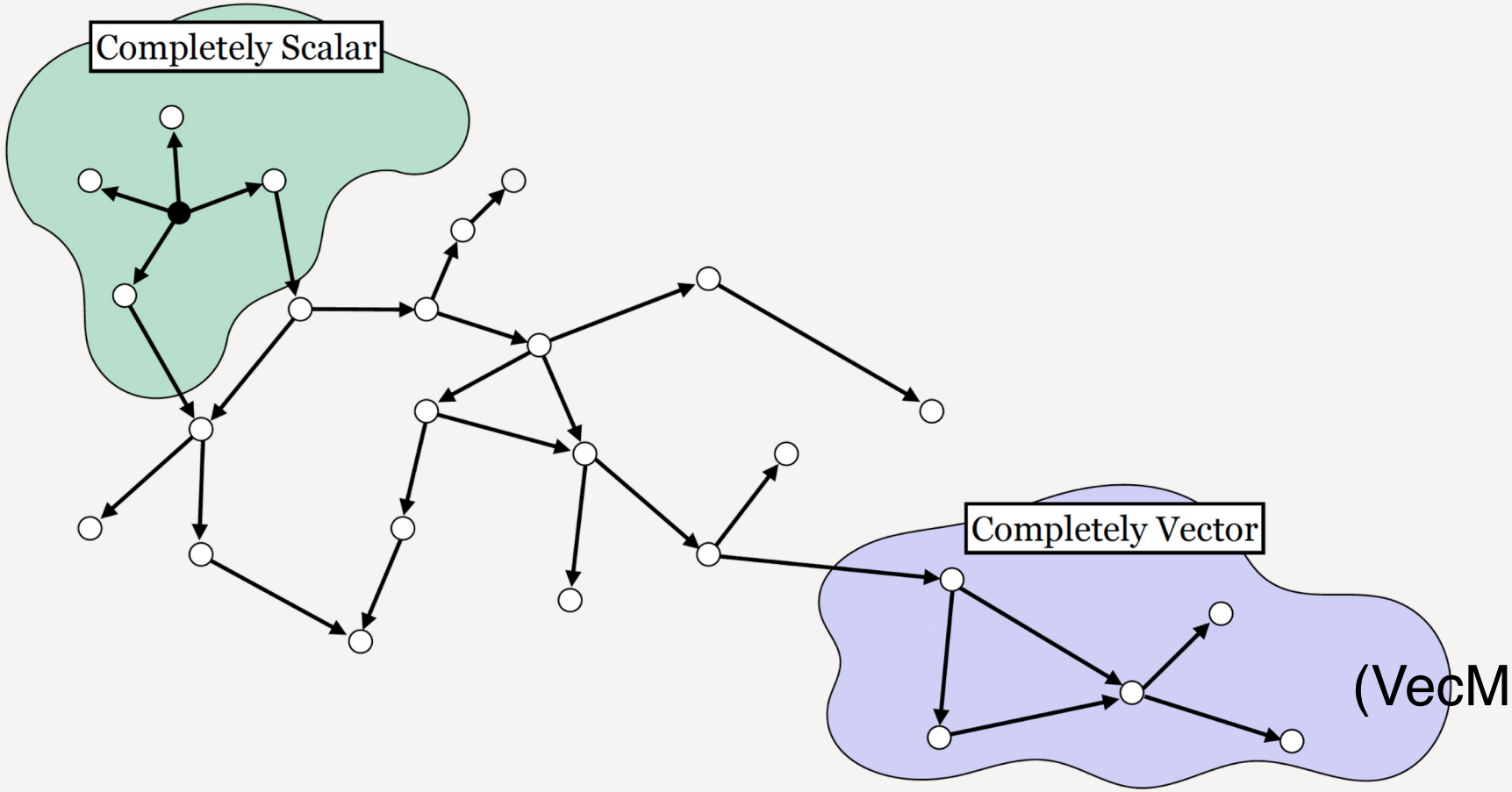
- Full EqSat causes OOM (Out of Memory) before reaching optimum
  - What does “full” mean? Run all rules round by round.



# Example: Isaria[1]

[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

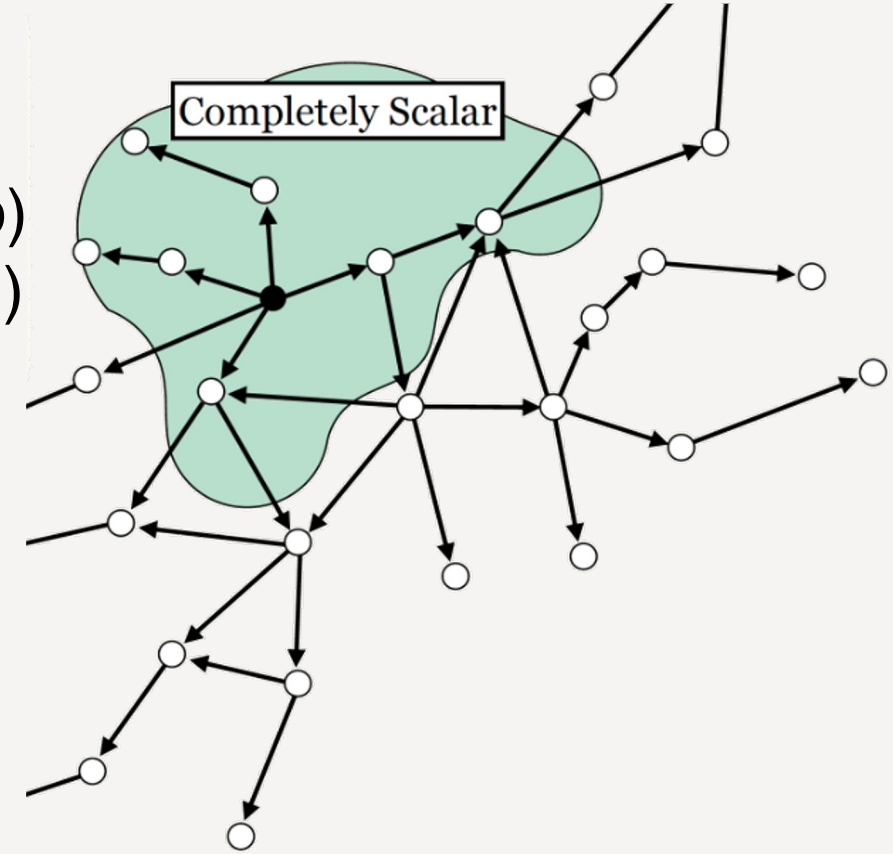
```
(let x (+ a b)
(let y (+ c d)
(Vec x y)))
```

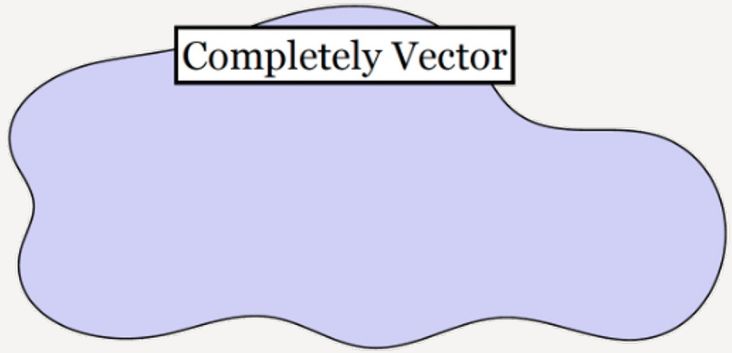



(VecMAC a b c)

# Example: Isaria[1]

(let x (+ a b)
(let y (+ c d)
(Vec x y)))



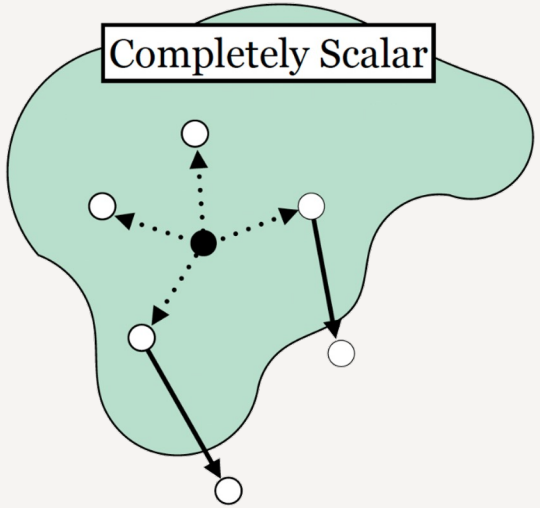



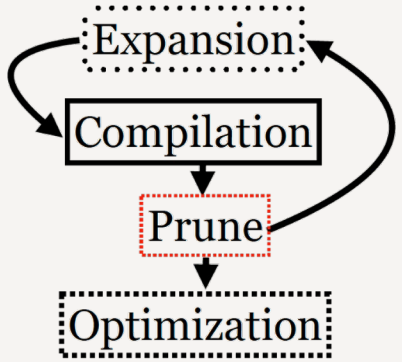

[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

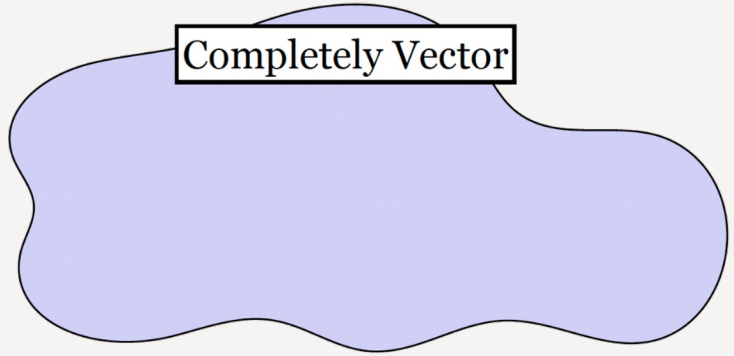
# Example: Isaria[1]

[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

```
(let x (+ a b)
(let y (+ c d)
(Vec x y)))
```



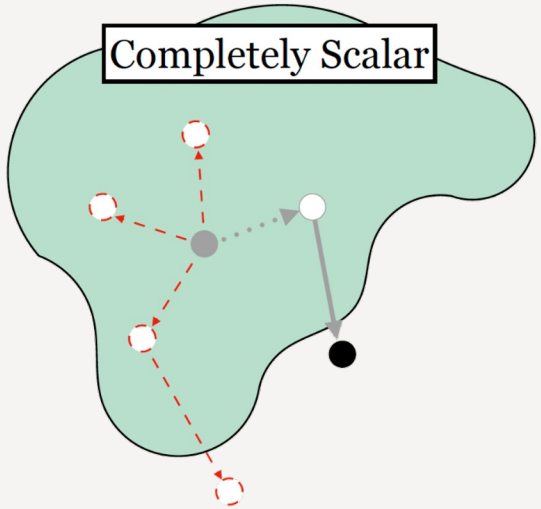



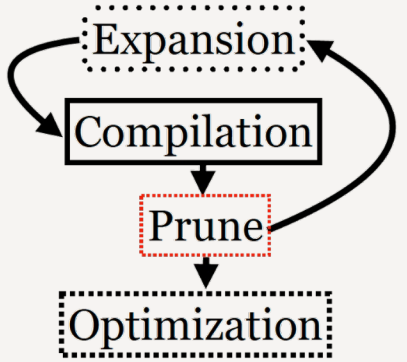



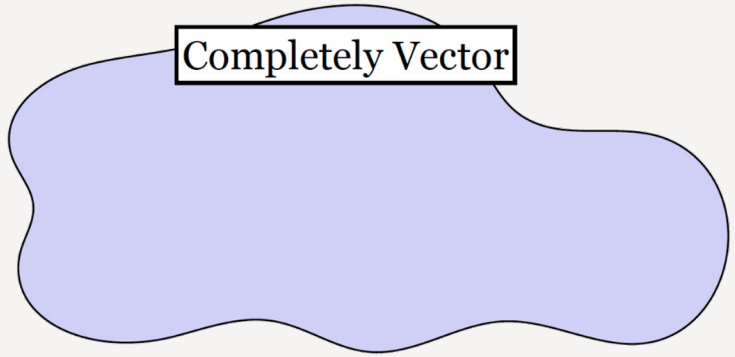
# Example: Isaria[1]

[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

```
(let x (+ a b)
(let y (+ c d)
(Vec x y)))
```



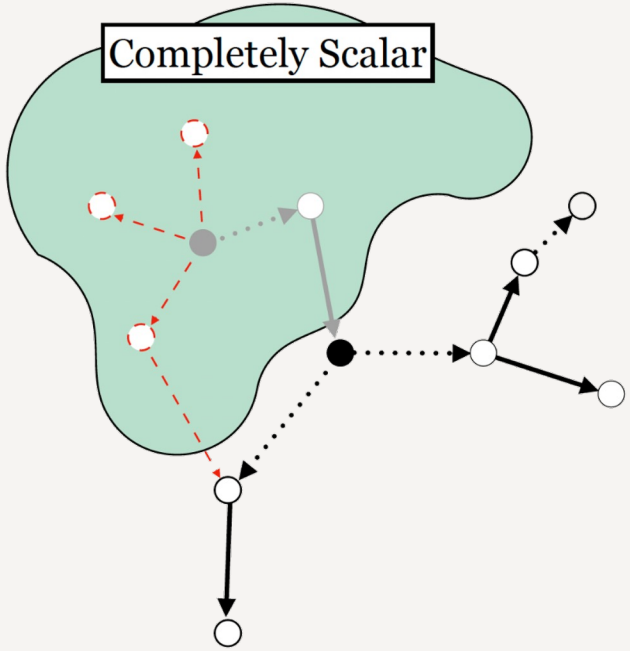



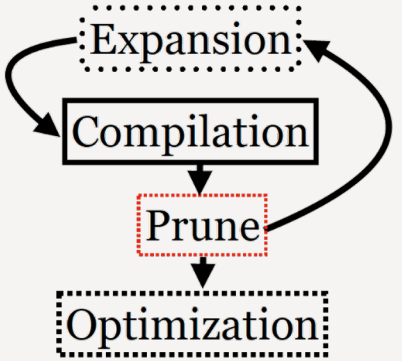



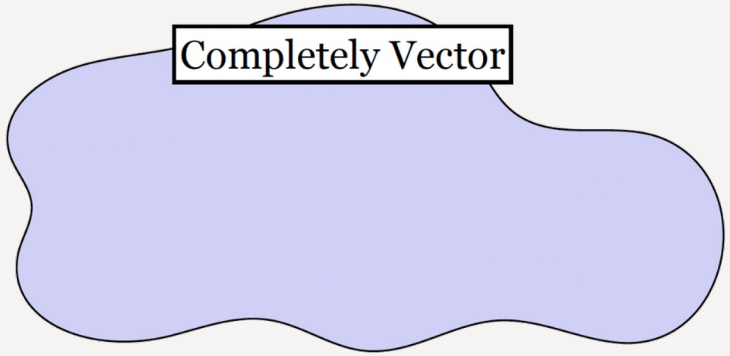
# Example: Isaria[1]

[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

```
(let x (+ a b)
(let y (+ c d)
(Vec x y)))
```



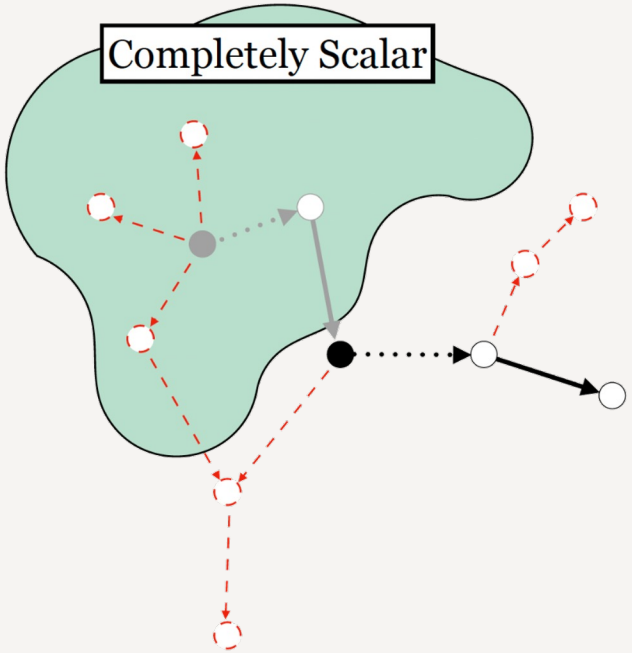



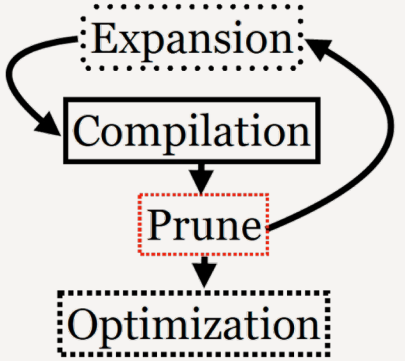



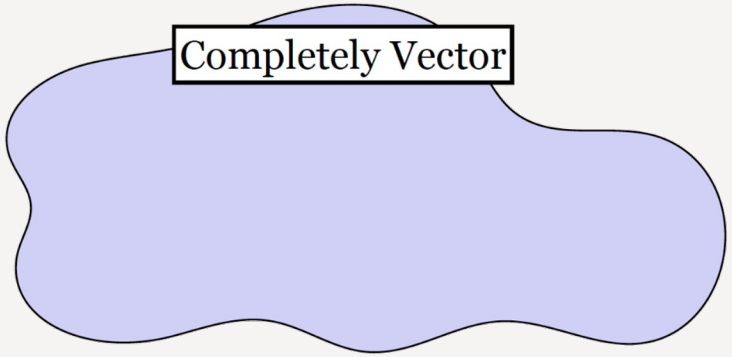
# Example: Isaria[1]

[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

```
(let x (+ a b)
(let y (+ c d)
(Vec x y)))
```

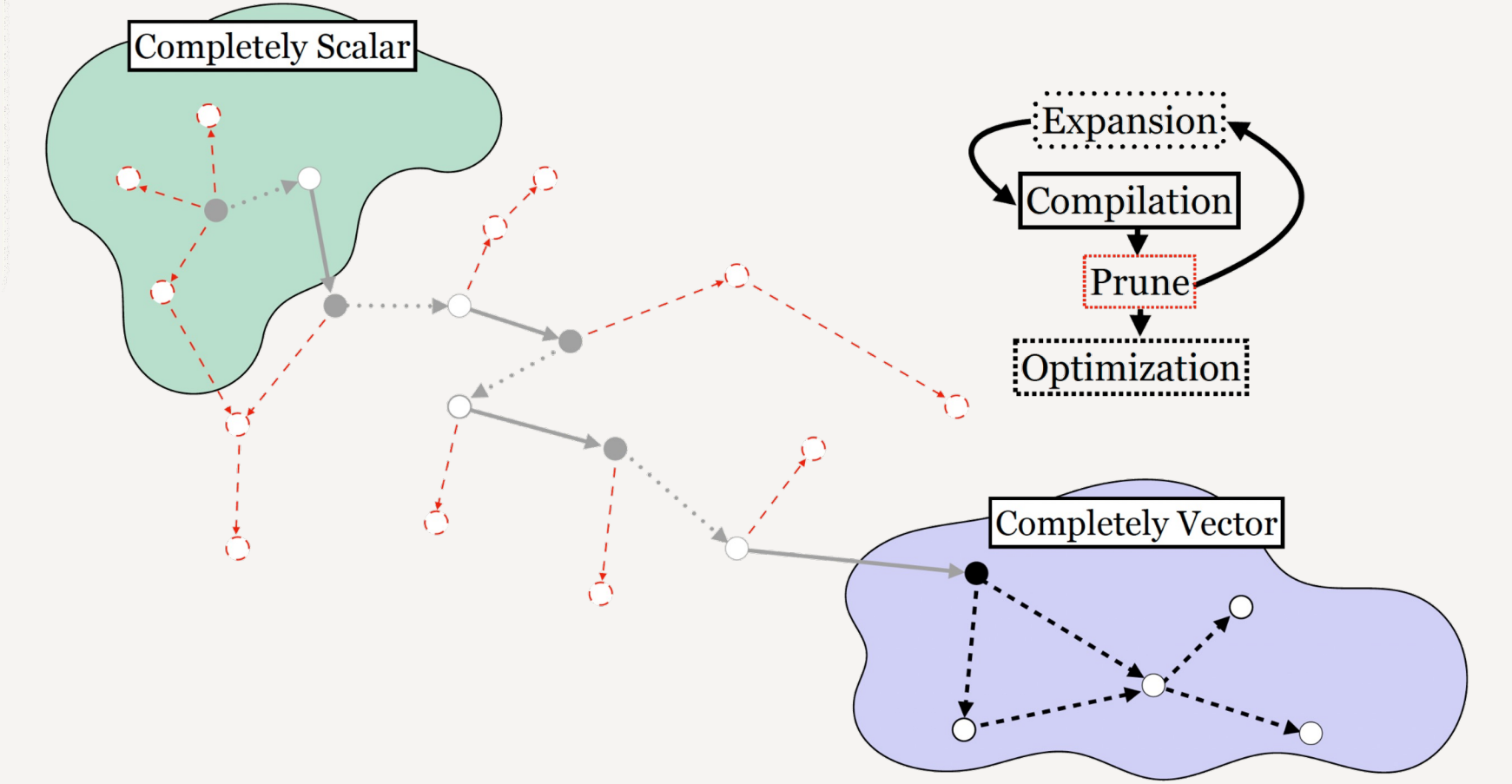


# Example: Isaria[1]

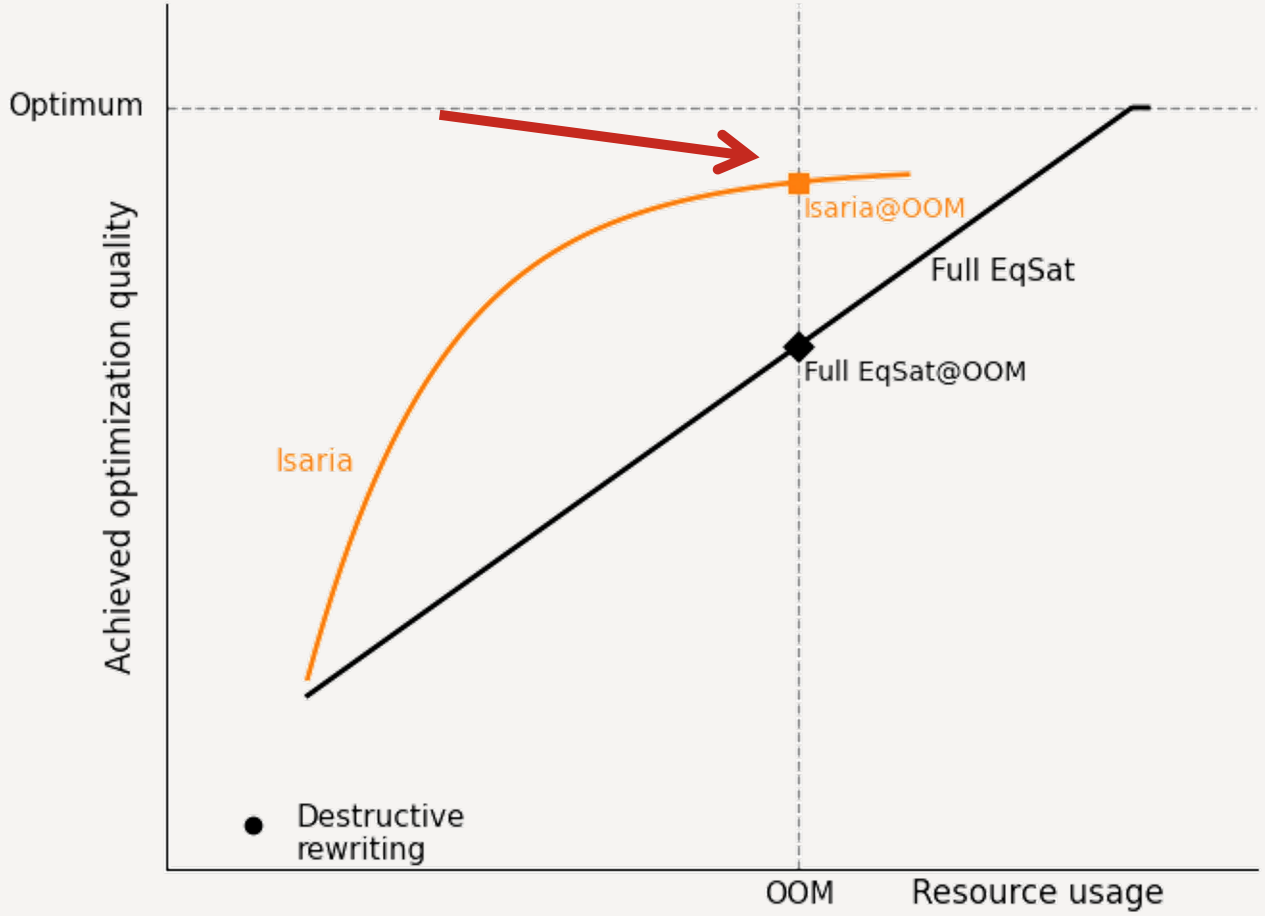
(let x (+ a b)
(let y (+ c d)
(Vec x y)))




[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

# Spectrum of Rewriting Techniques

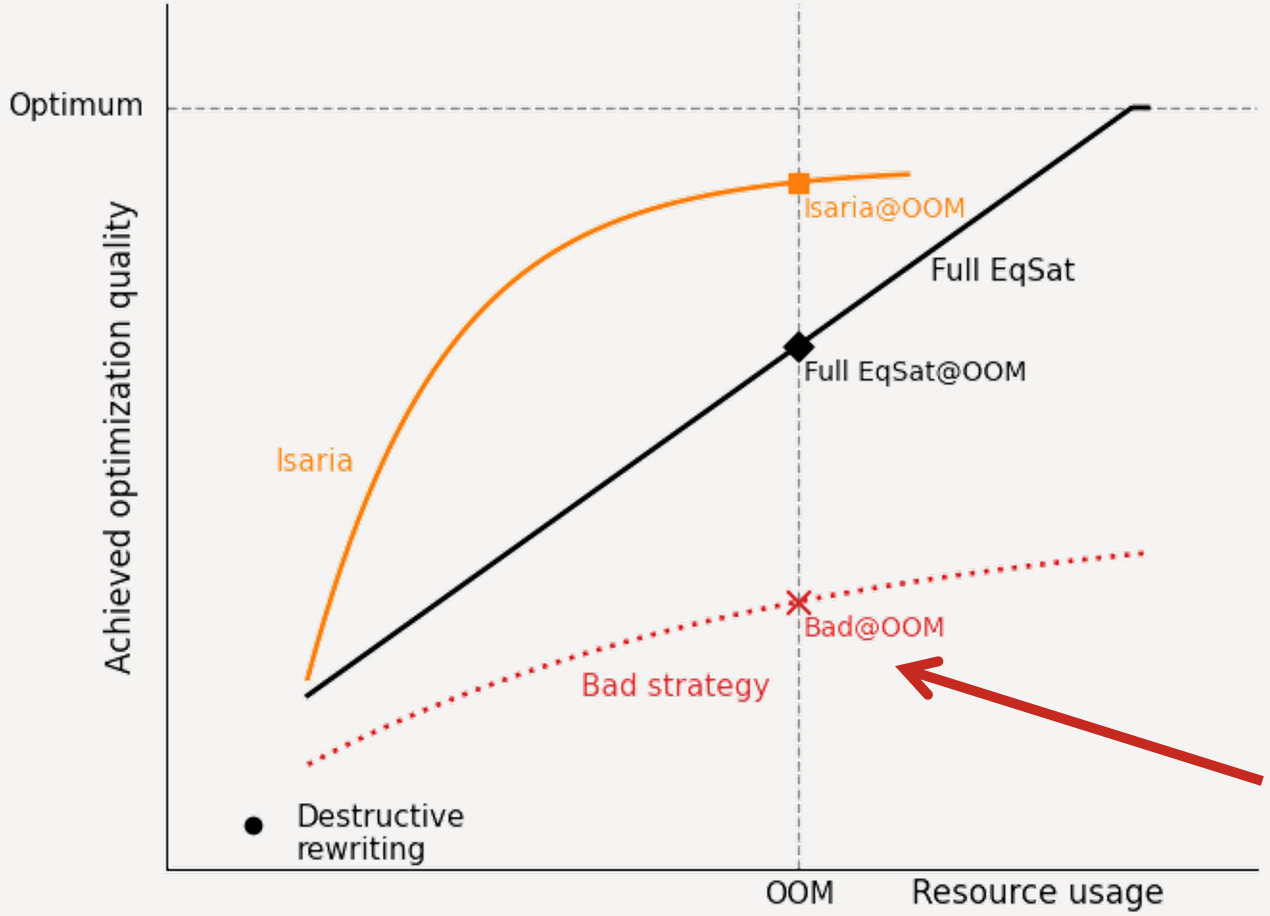
- Good strategy (like Isaria[1]) yields better solution




[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

# Spectrum of Rewriting Techniques

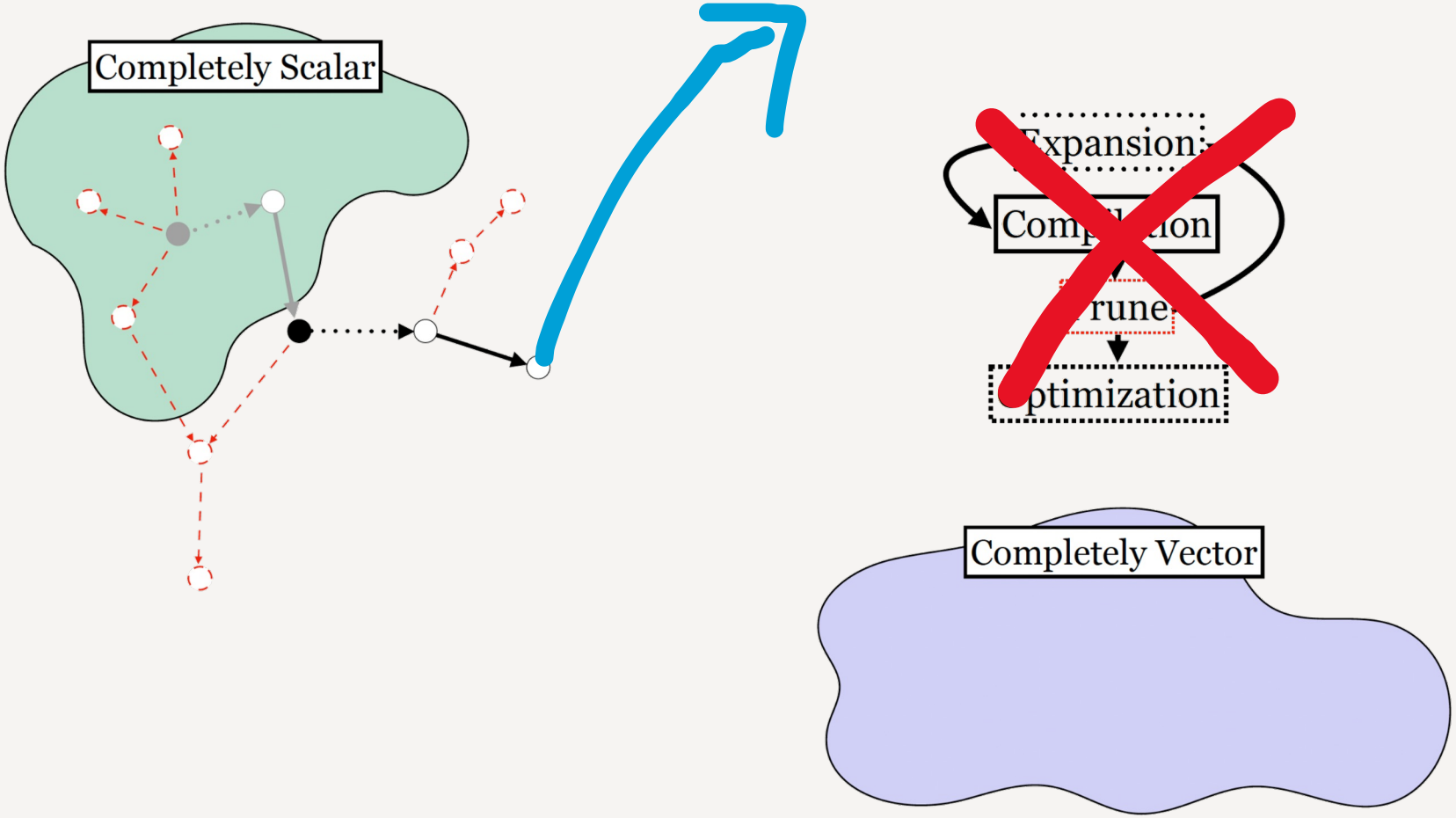
- But we might write a bad strategy also 😭



# Example: Isaria[1]

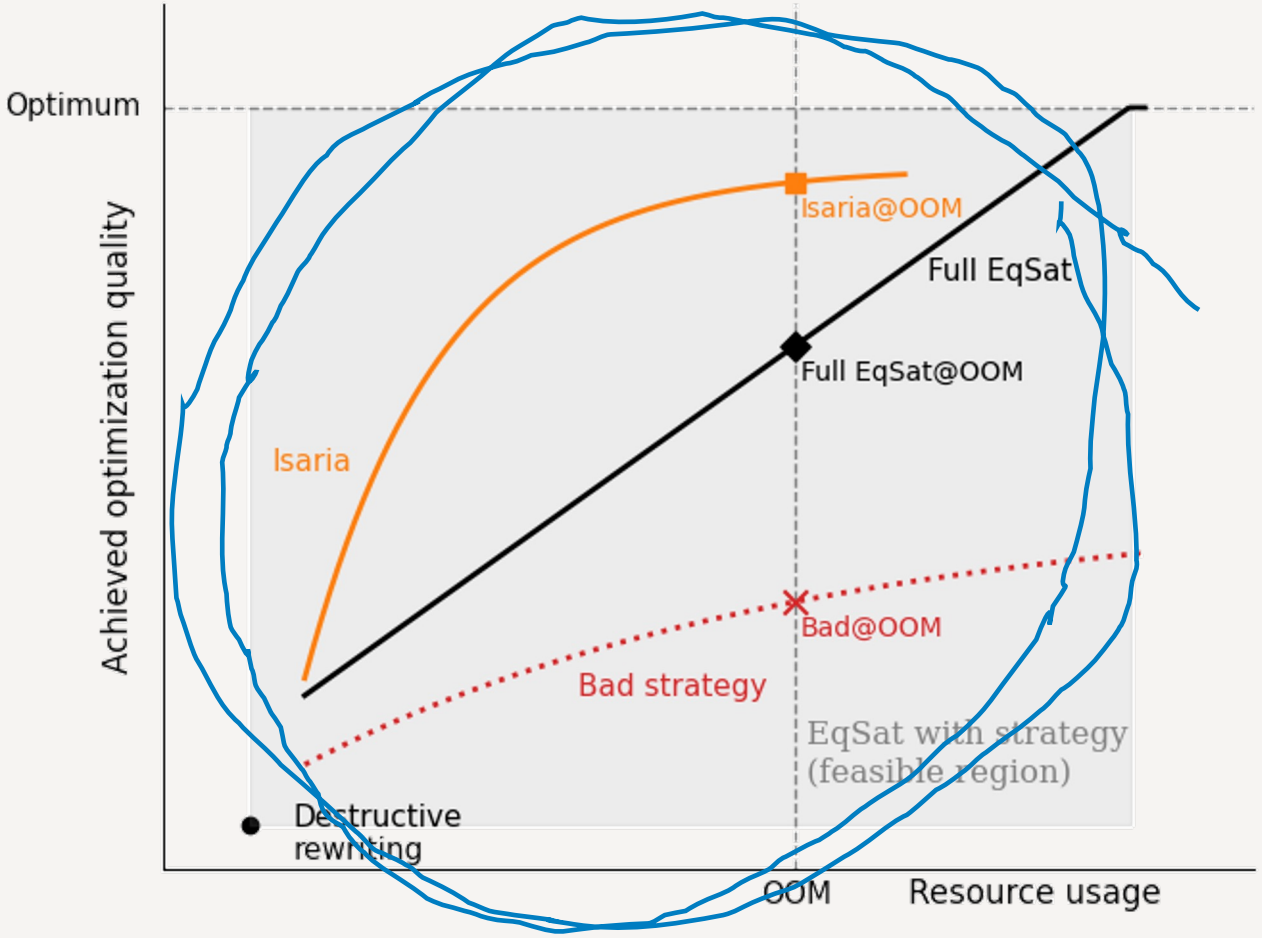
[1] Samuel Thomas and James Bornholt. 2024. Automatic Generation of Vectorizing Compilers for Customizable Digital Signal Processors. ASPLOS '24. https://doi.org/10.1145/3617232.3624873

```
(let x (+ a b)
(let y (+ c d)
(Vec x y)))
```



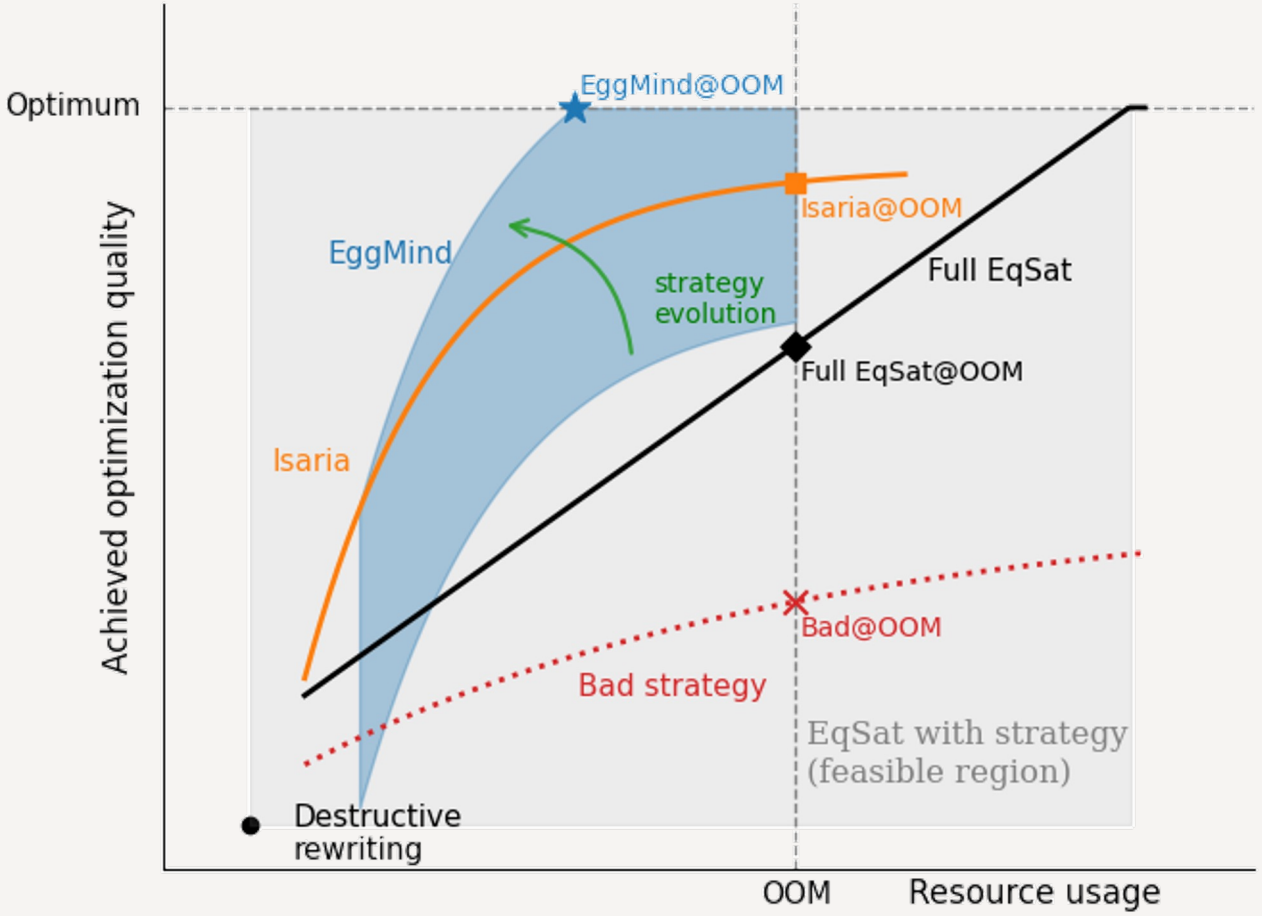
# Spectrum of Rewriting Techniques

- We need to design a good strategy for a given problem!

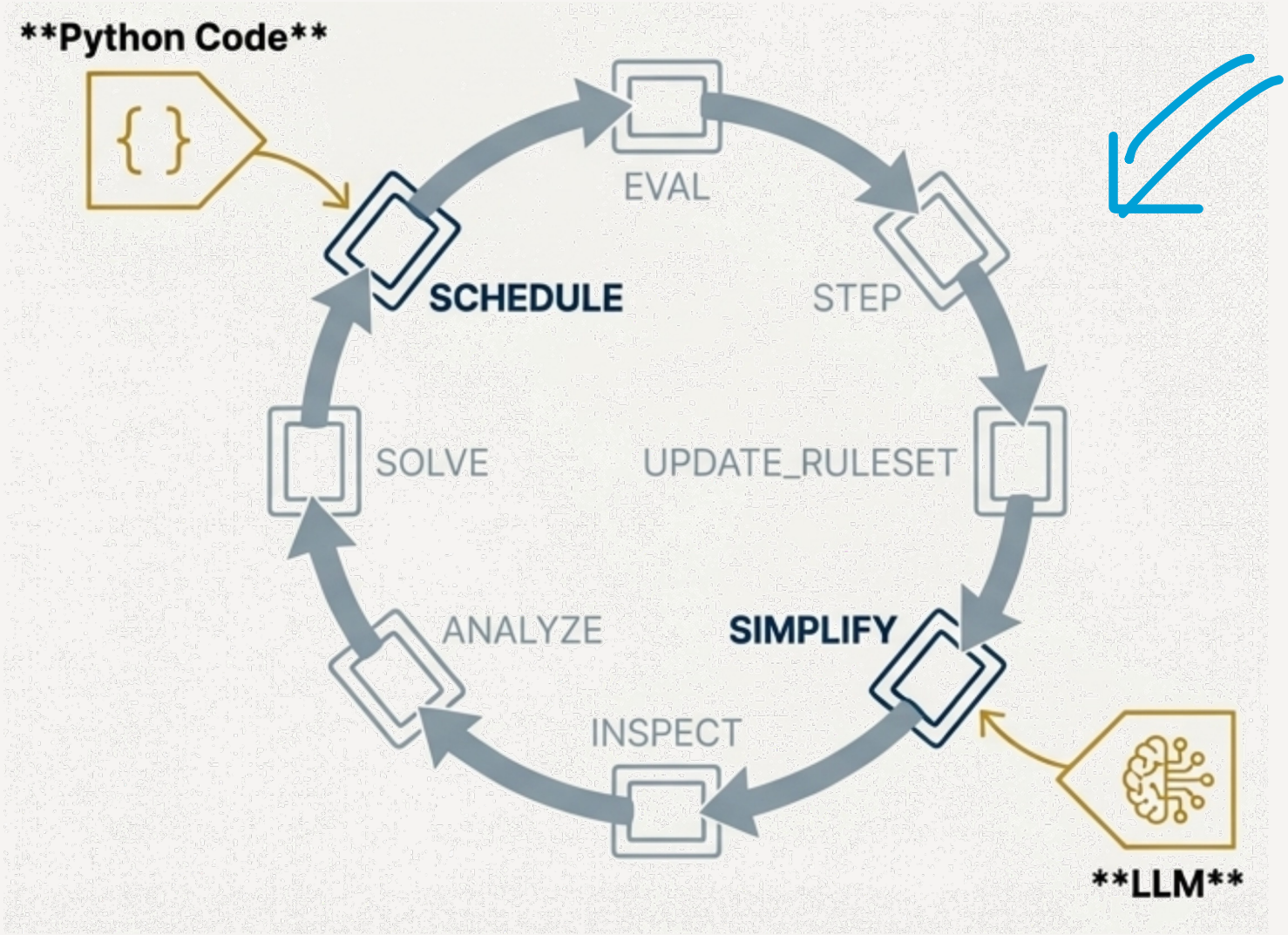


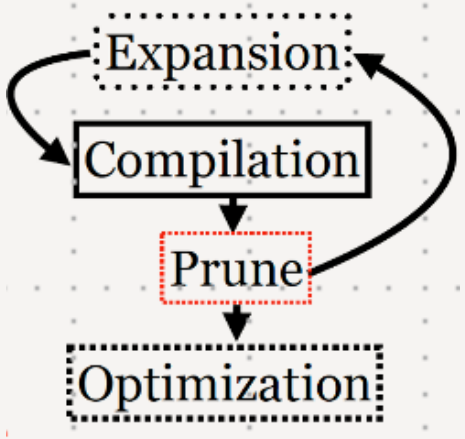
# Spectrum of Rewriting Techniques

- EggMind uses **LLM** to evolve a good EqSat strategy



# Strategy = Functor Loop



# DSL for Strategy

- What does LLM generate?

**Layer 1: Units (Optimization Atoms)**

| | | |
|---|---|---|
| $?x \in$ | PatVars | (Pattern variables, Holes) |
| $p ::=$ | $\text{term}(?x, \dots)$ | (Term templates) |
| $r ::=$ | $p_1 \rightarrow p_2\ [g]$ | (Rewrite rules with optional guard) |
| $rs ::=$ | $\{r_1, r_2, \dots\}$ | (Named rulesets) |
| $m ::=$ | $\text{cost\_fn}(e)$ | (Extraction metrics) |

**Layer 2: Commands (Rewrite Algebra)**

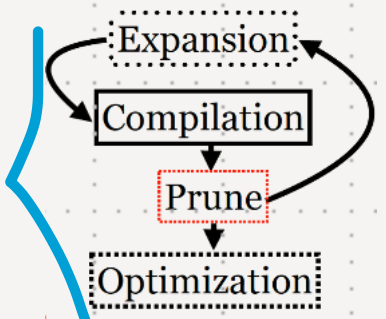
| | | |
|---|---|---|
| $c ::=$ | $\texttt{Saturate}(rs)$ | (Apply until fixpoint) |
| | $\mid \texttt{Repeat}(n, c)$ | (Bounded iteration) |
| | $\mid \texttt{Seq}(c_1, \dots)$ | (Sequential composition) |
| | $\mid \texttt{Union}(c_1, \dots)$ | (Parallel merge) |

**Layer 3: Plans (Orchestrated Workflows)**

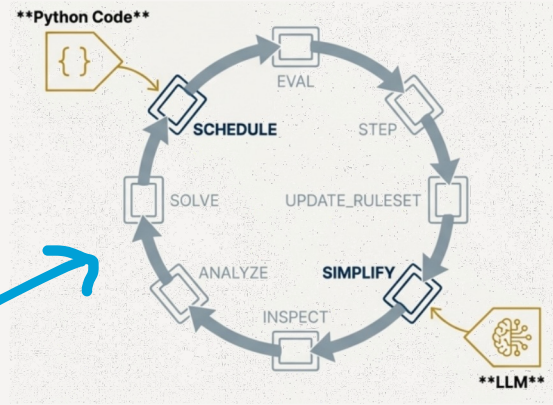
| | | |
|---|---|---|
| $ph ::=$ | $\text{Phase}(id, c, n)$ | (Named stage with iteration limit) |
| $pl ::=$ | $\text{Plan}(ph, \dots)$ | (Concatenated phases) |
| | $\mid \text{Plan}(\dots, H)$ | (With init/cleanup hooks) |

```
(datatype Expr
   (Num BigRat)
   ... ))
(let zero (bigrat (bigint 0) (bigint 1)))
(let one (bigrat (bigint 1) (bigint 1)))
(let two (bigrat (bigint 2) (bigint 1)))
(ruleset optimizations)
(ruleset analysis)
(birewrite (Add x (Add y z)) (Add (Add x y) z)
:ruleset optimizations)
(birewrite (Mul x (Mul y z)) (Mul (Mul x y) z)
:ruleset optimizations)
...
(run-schedule
  (repeat 2
    (saturate (run analysis))
    (run optimizations)))
```

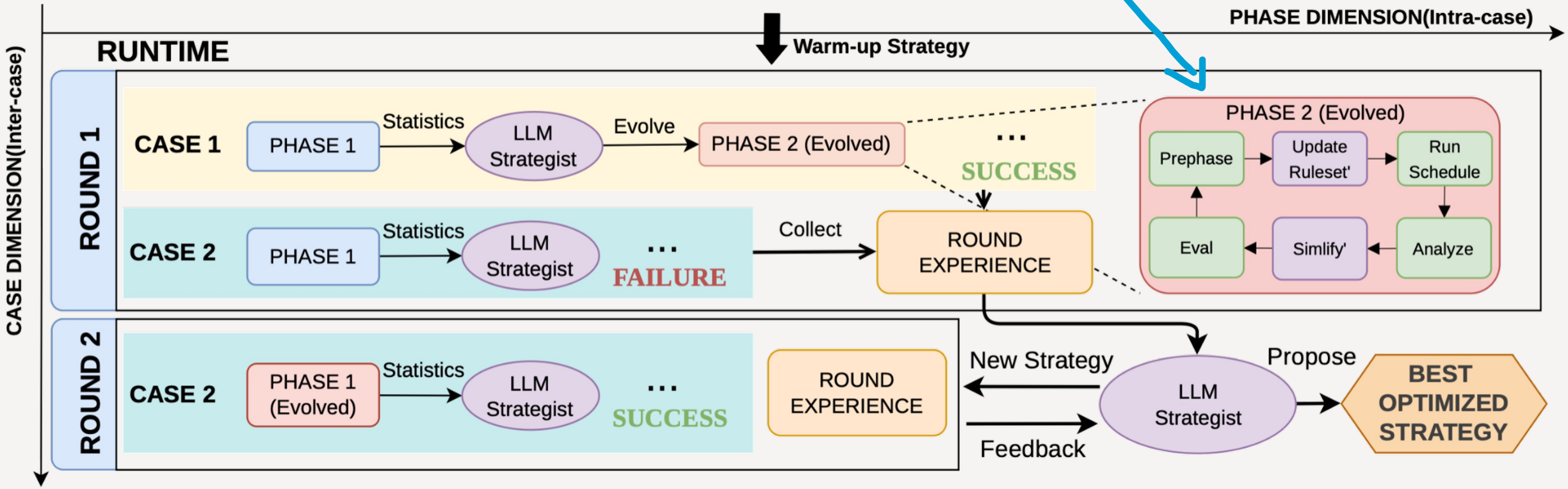
Egglog



# Two Axes: Phase & Case




Phase = a loop iteration

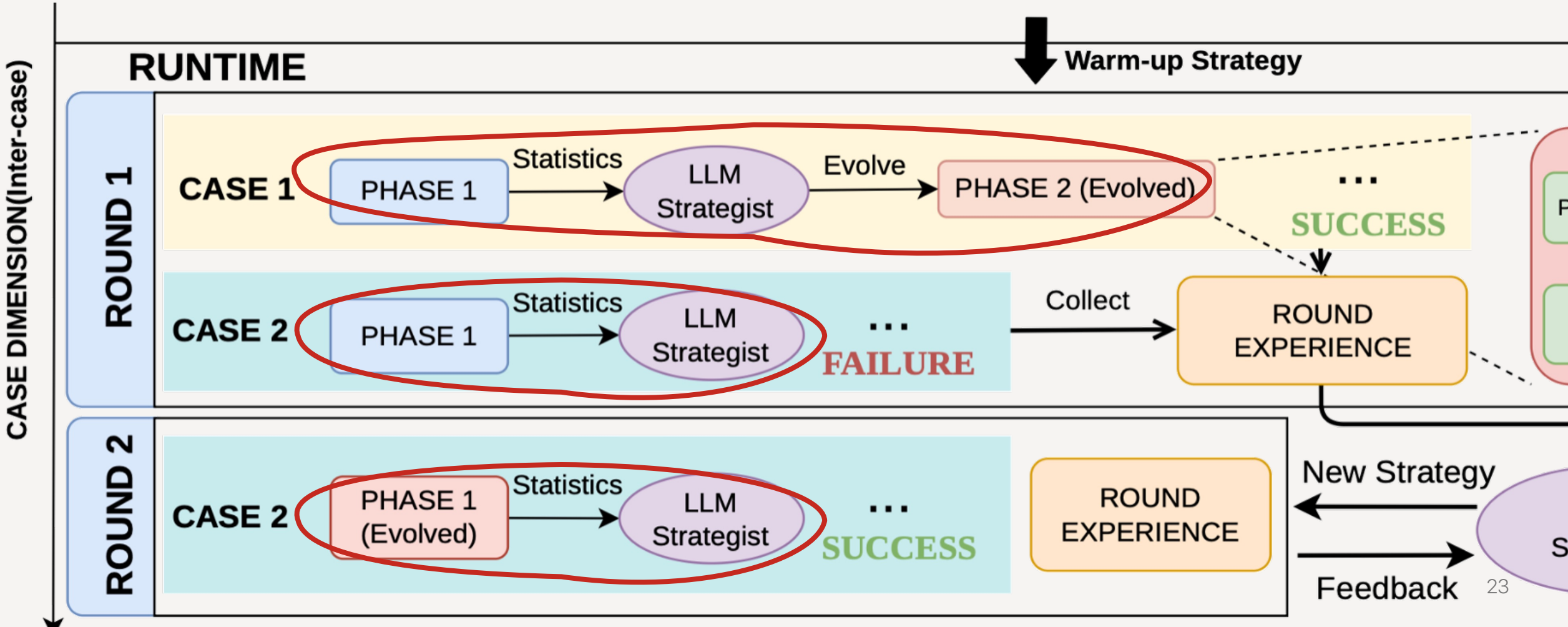



Case = a problem instance

Like vectorizing one given program.

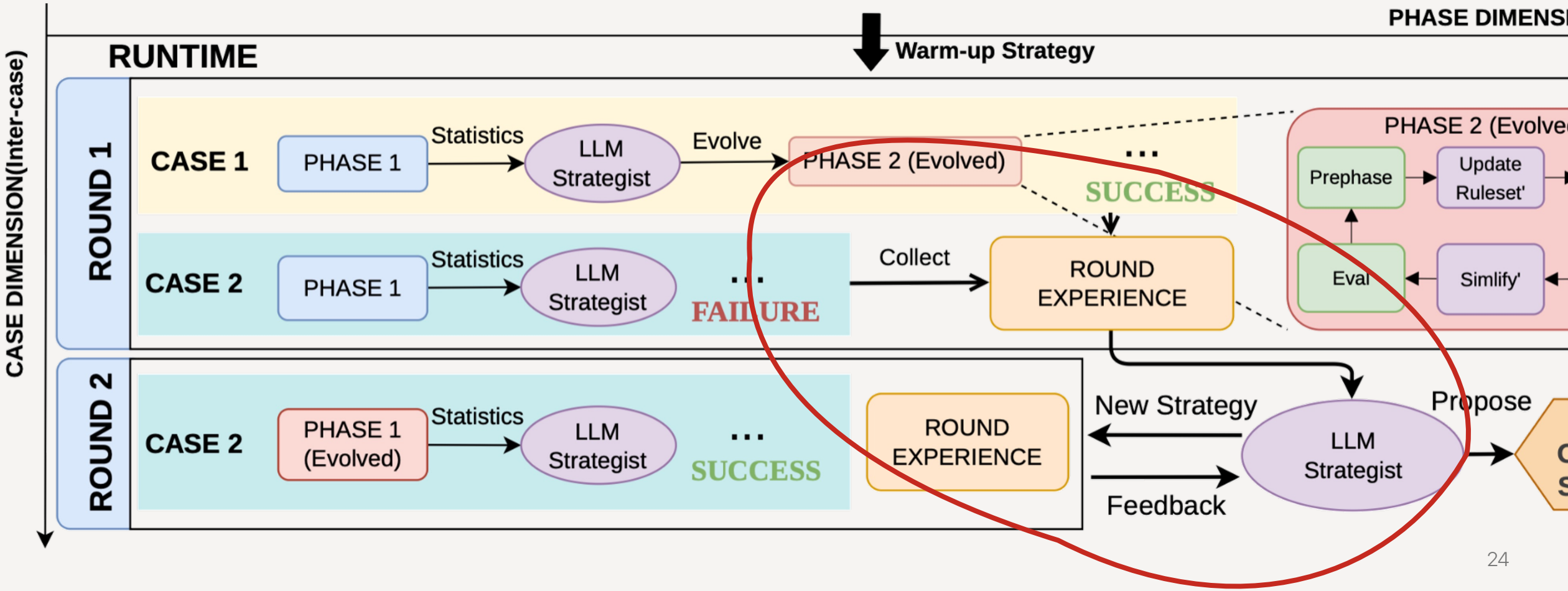
# Phase Intelligence

- Evolving across phases (runtime adaption)



# Case Intelligence

- Evolving across cases (learn from success and failure)



# Still In Progress

- Many problems being explored

- What are the statistics/experiences/... fed into LLM?
- Can LLM prompts be reused across problems?
  - Vectorization, tensor compilation, logic synthesis, ...
- How to prevent hanging/stuck/crashed from bad strategy/action?
- ...

- Implementation and evaluation

# Thanks

**EggMind: An Intelligent Engine for Scalable Equality Saturation**