

# **Michel Steuwer**

**University of Edinburgh**



# It's all about Choices!

- I couldn't agree more with Sven-Bodo about this statement
- I (of course) couldn't disagree more with Sven-Bodo about all the choices made in SaC 

**Shouldn't the consequence be not to *make* choices but to *offer* them?**

**RISE**+ **ELEVATE**

# : Expose Optimization Choices

[\[ICFP 2020\]](#)[\[CGO 2021\]](#)**RISE**

## A Pattern-Based Intermediate Languages

```
def highLevelProgram =
  depFun((n: Nat, m: Nat, o: Nat) =>
    fun(A: n.o.f32 => fun(B: m.o.f32 =>
      A ▷ map(fun(rowOfA =>
        B ▷ map(fun(rowOfB =>
          zip(rowOfA)(rowOfB) ▷
            map(fun(x => fst(x) * snd(x))) ▷
              reduce(add)(0.0f) ))) ))) )
```

Successor  
to LIFT**ELEVATE**

## A Programming Language for describing Optimization Strategies

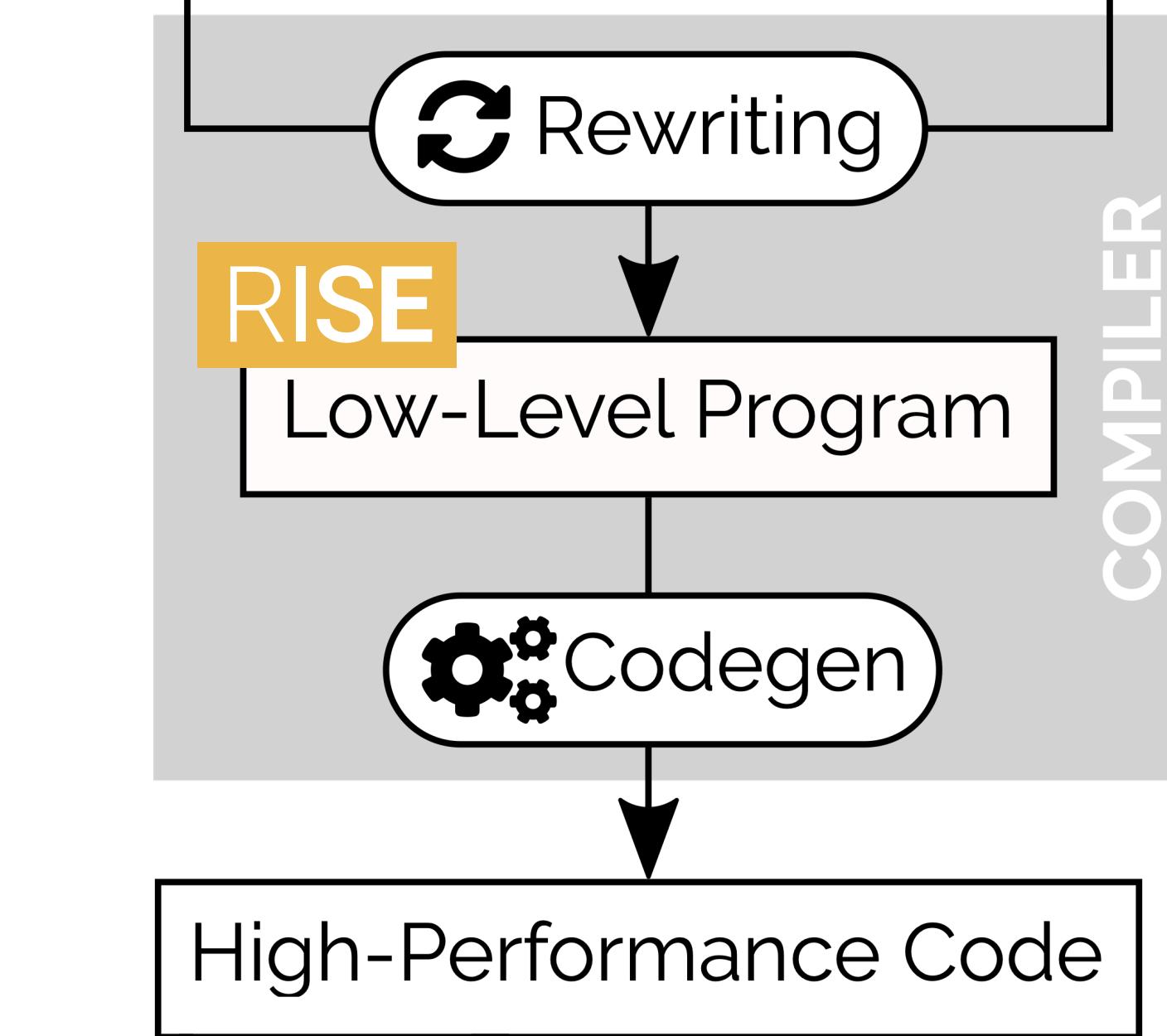
```
def optimizationStrategy =
  (`map ↳ mapPar`      `@` outermost(isMap)) `;` ;
  (`map ↳ mapSeq`     `@` outermost(isMap)) `;` ;
  (`reduce ↳ reduceSeq` `@` everywhere)
```

**RISE**

High-Level Program

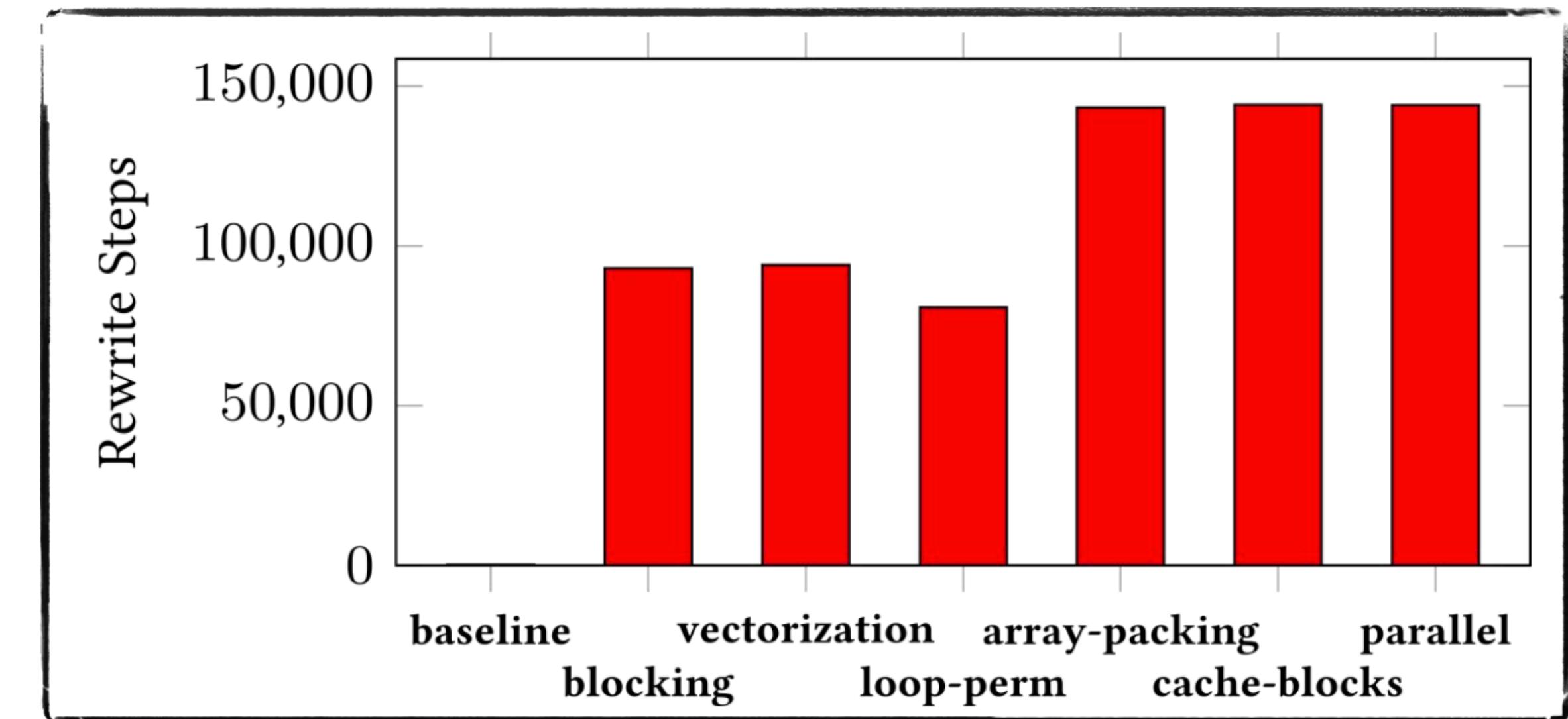
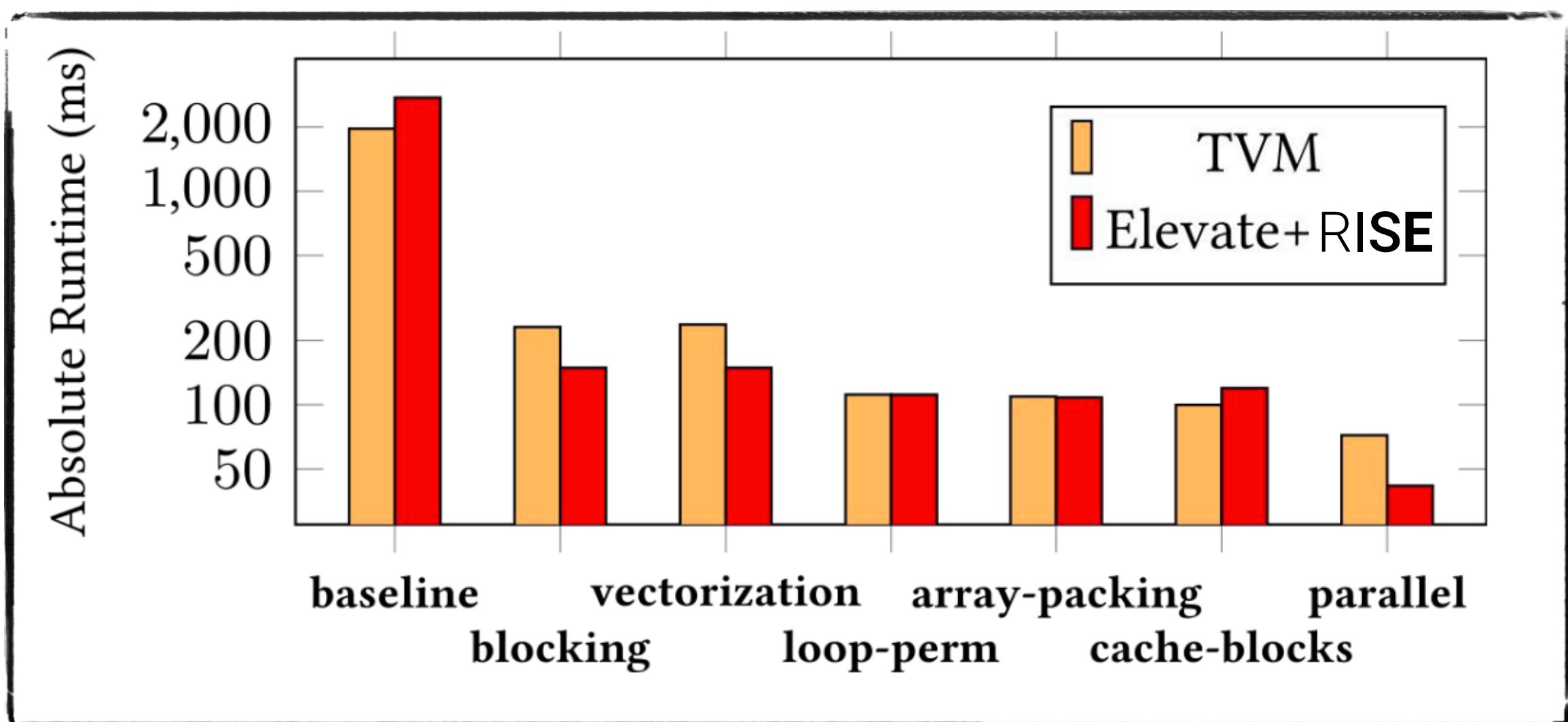
**ELEVATE**

Optimization Strategy

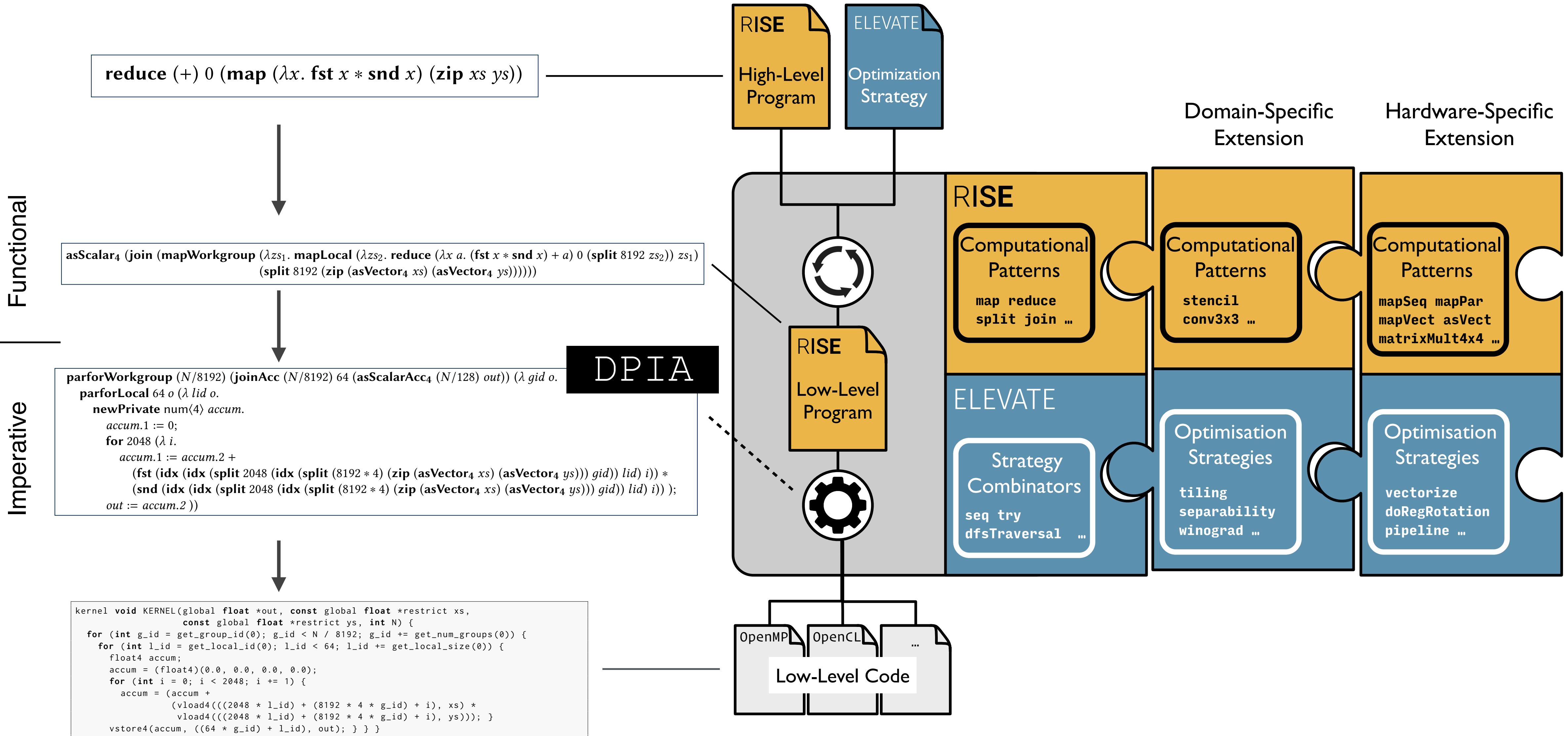
<https://rise-lang.org><https://elevate-lang.org>

**RISE**+ **ELEVATE**

# : Results for Matrix Multiplication

**ELEVATE****allows to implement state-of-the art scheduling APIs from first principle**

# Compilation via multiple intermediate languages





# MLIR: Offer integration choices

- Focus on compiler intermediate languages rather than user facing languages
  - Avoid flame war over functional programming
  - Type systems (e.g. dependent types) can be complex to carry rich information
  - Easy(er) to build fully integrated systems

The screenshot shows a PDF viewer window with the title bar 'CC-2021.pdf' and 'Page 1 of 11'. The main content area displays three circular ACM certification seals: 'Artifacts Available V1.1' (green), 'Artifacts Evaluated Functional V1.1' (red), and 'Results Reproduced V1.1' (blue). Below the seals, the title 'Integrating a Functional Pattern-Based IR into MLIR' is centered. On the left, there is a document icon and the text '[cc 2021]'. At the bottom, author information is listed for three individuals: Martin Lücke, Michel Steuwer, and Aaron Smith.

[cc 2021]

## Integrating a Functional Pattern-Based IR into MLIR

Martin Lücke  
University of Edinburgh  
United Kingdom  
[martin.luecke@ed.ac.uk](mailto:martin.luecke@ed.ac.uk)

Michel Steuwer  
University of Edinburgh  
United Kingdom  
[michel.steuwer@ed.ac.uk](mailto:michel.steuwer@ed.ac.uk)

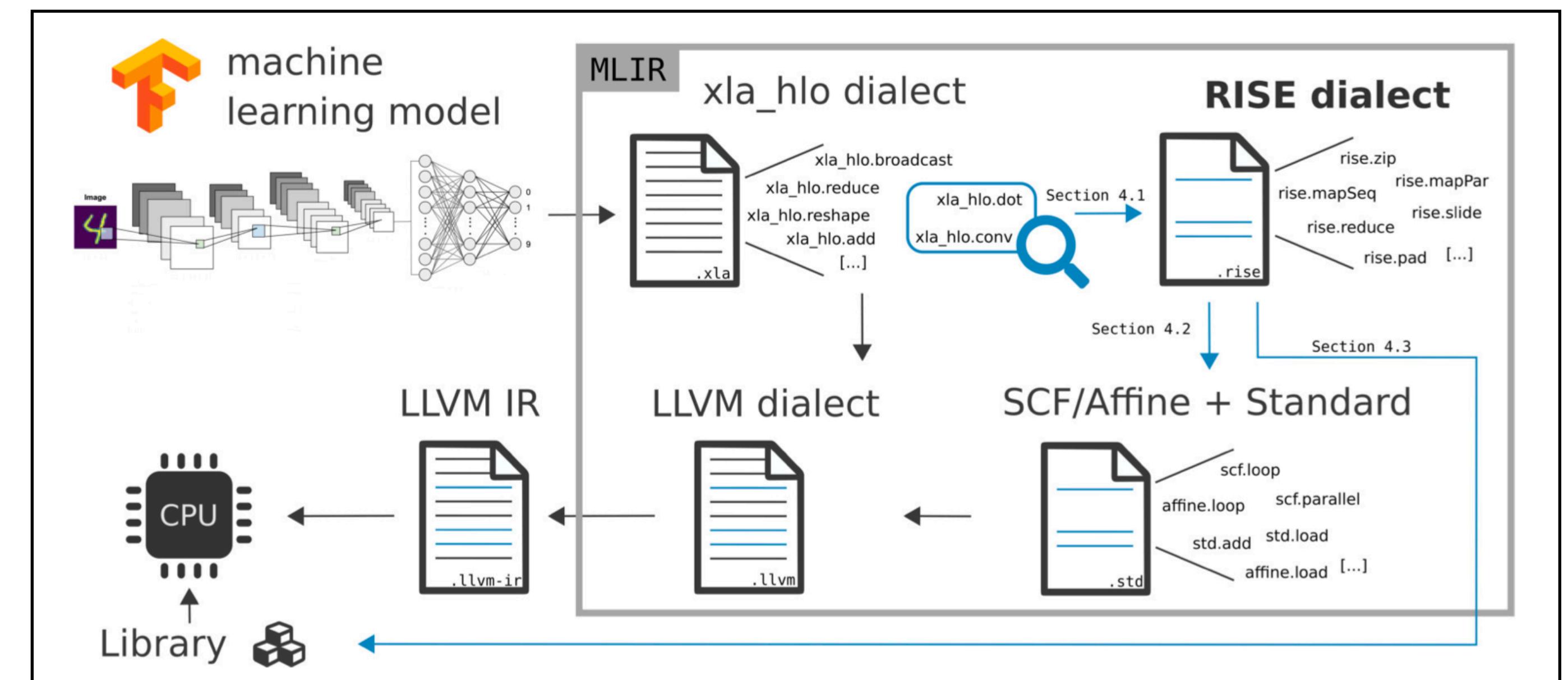
Aaron Smith  
University of Edinburgh / Microsoft  
USA  
[aaron.smith@microsoft.com](mailto:aaron.smith@microsoft.com)



# MLIR: Offer integration choices

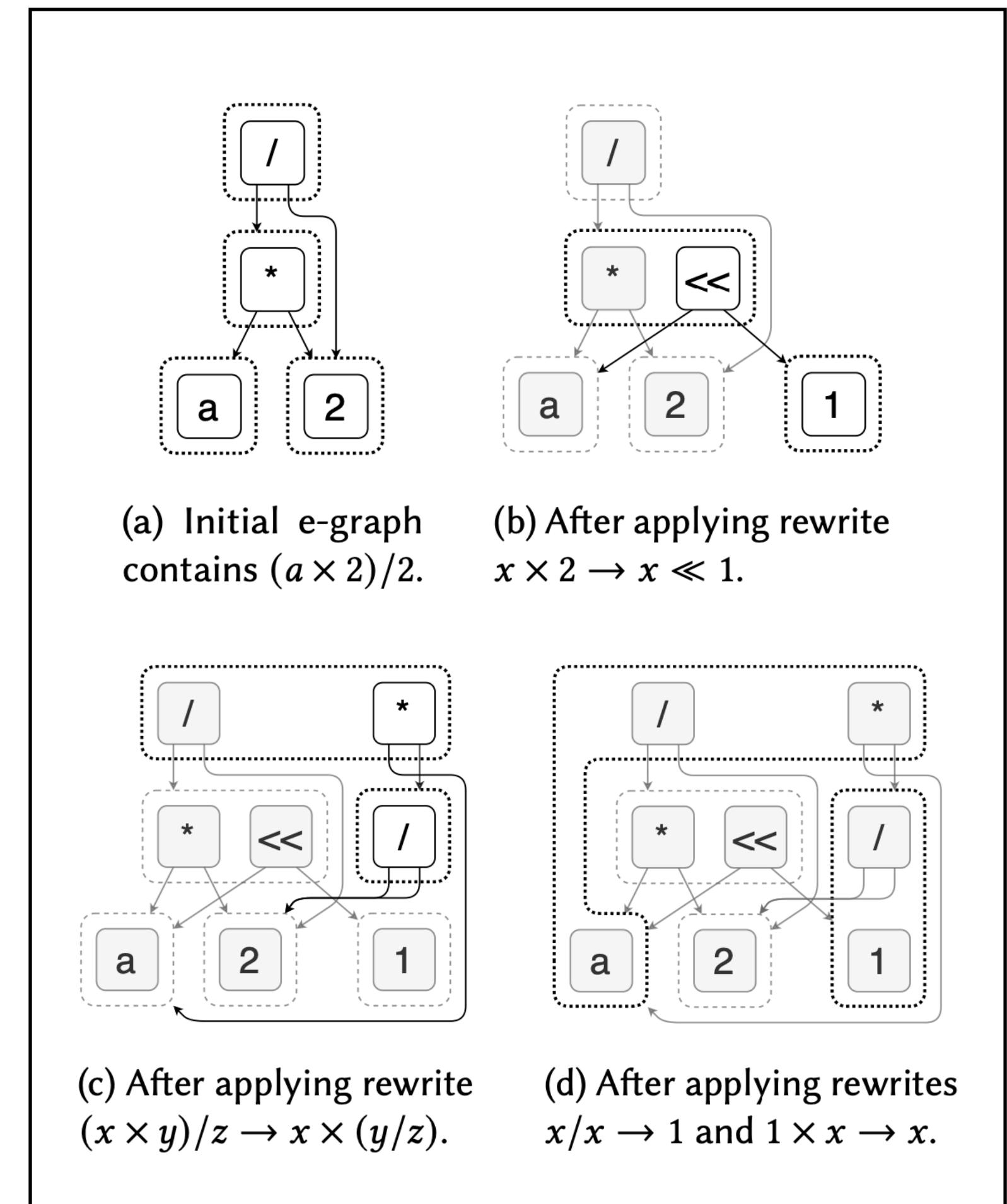
```
func @mm_fused(%outArg, %inA, %inB) {
  %A = in %inA
  %B = in %inB
  %t = rise.transpose #rise.nat<2048>
    #rise.nat<2048> #rise.scalar<f32>
  %B_t = rise.apply %t, %B
  %m1fun = lambda (%arow) -> array<2048, scalar<f32>> {
    %m2fun = lambda (%bcol) -> scalar<f32> {
      %zipFun = zip #nat<2048> #scalar<f32> #scalar<f32>
      %zippedArrays = rise.apply %zipFun, %arow, %bcol
      %reduceLambda = lambda(%tuple, %acc)->scalar<f32> {
        %fstFun = rise.fst #scalar<f32> #scalar<f32>
        %sndFun = rise.snd #scalar<f32> #scalar<f32>
        %first = rise.apply %fstFun, %tuple
        %second = rise.apply %sndFun, %tuple
        %result = rise.embed(%first, %second, %acc) {
          %product = mulf %first, %second : f32
          %result = addf %product, %acc : f32
          return %result : f32
        }
        return %result : scalar<f32>
      }
      %init = rise.literal #lit<0.0>
      %reduceFun = reduceSeq #nat<2048> #tuple
      %result = rise.apply %reduceFun, %reduceLambda,
        %init, %zippedArrays
      return %result : scalar<f32>
    }
    %m2 = mapSeq #nat<2048> #array<2048, scalar<f32>>
      #scalar<f32>
    %result = rise.apply %m2, %m2fun, %B_t
    return %result : array<2048, array<2048, scalar<f32>>>
  }
  %m1 = mapSeq #nat<2048> #array<2048, scalar<f32>>
    #array<2048, scalar<f32>>
  %result = rise.apply %m1, %m1fun, %A
  out %outArg <- %result
  return
}
```

A |> map(fun(arrow, B |> transpose |> reduce(fun(ab, acc), (ab<sub>1</sub>xab<sub>2</sub>)+acc), o)))  
zip(arrow, bcol) |> reduce(fun(ab, acc), (ab<sub>1</sub>xab<sub>2</sub>)+acc), o))



# How to make choices?

- Fully manual via **ELEVATE**
- Fully automated via:
  - Stochastic methods **[ICFP 2015]**
  - Equality Saturation & E-graphs:  
 Search “Optimizing Functional Programs with Equality Saturation” on YouTube
  - Reinforcement Learning & other machine learning methods
- Big open question: *How can we mix both modes conveniently?*



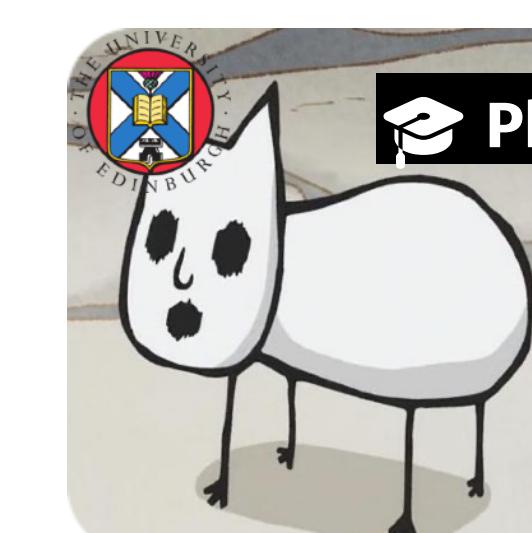
# Team



Martin  
Lücke



Federico  
Pizzuti



Xueying Qin



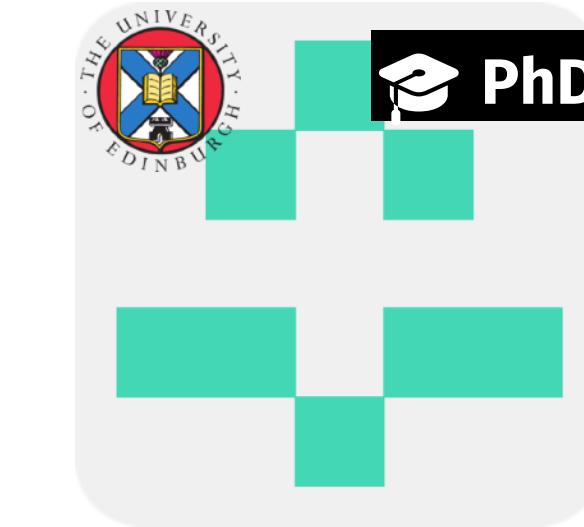
Johannes  
Lenfers



Thomas  
Køehler



Bastian  
Köpcke



Rongxiao  
Fu



Compilers

Programming Languages