Domain-specific Optimisation for the High-level Synthesis of CellML-based Simulation Accelerators

Julian Oppermann
Andreas Koch

Ting Yu
Oliver Sinnen

Embedded Systems & Applications
TECHNISCHE UNIVERSITÄT DARMSTADT

PARALLEL AND RECONFIGURABLE COMPUTING LAB
THE UNIVERSITY OF AUCKLAND
NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau

CellML

- Standard to model biomedical problems
- Differential equations describe interaction between components

CellML

- Standard to model biomedical problems
- Differential equations describe interaction between components

$$alpha\_m = \frac{-(V+47)}{e^{-\frac{V+47}{10}} - 1}$$

$$beta\_m = 40 \times e^{-0.056 \times (V+72)}$$

$$\frac{dm}{dt} = alpha\_m \times (1-m) - (beta\_m \times m)$$

CellML

- Standard to model biomedical problems
- Differential equations describe interaction between components

$$alpha\_m = \frac{-(V+47)}{e^{-\frac{V+47}{10}} - 1}$$

$$beta\_m = 40 \times e^{-0.056 \times (V+72)}$$

$$\frac{dm}{dt} = alpha\_m \times (1-m) - (beta\_m \times m)$$

C Code Generation Service

```
ALGEBRAIC[1] = -1.0 * (STATES[0] + 47.0)
               / (exp(-0.1 * (STATES[0] + 47.0)) - 1.0);
ALGEBRAIC[8] = 40.0 * exp(-0.056 * (STATES[0] + 72.0));
RATES[1]     = ALGEBRAIC[1] * (1.0 - STATES[1])
               - ALGEBRAIC[8] * STATES[1];
```

Hardware-accelerated cell simulation

- Numerical integration
- Cells can be treated independently for some time
- ODoST (Yu et al., 2015): fully-spatial, fully-pipelined FPGA accelerators from a model's equation system
- Instantiate as many pipelines as fit on the FPGA

Hardware-accelerated cell simulation

- Numerical integration
- Cells can be treated independently for some time
- ODoST (Yu et al., 2015): fully-spatial, fully-pipelined FPGA accelerators from a model's equation system
- Instantiate as many pipelines as fit on the FPGA

Fewer resources per pipeline → **More throughput**

Approach

- Fully-spatial computation
  = every SW instruction becomes HW operator

- SW compiler's architecture independent optimisations

  - eliminate redundant operations, or

  - replace "expensive" ops by "cheaper" ones

- Try unsafe floating-point transformations

Cost model

- Estimation of resource demand → guide opts
- Based on relative, per operator ALM and DSP usage on Stratix IV

$$c(op) = \left\| \begin{pmatrix} \frac{n_{\mathrm{ALM}}(op)}{212480} \\ \frac{n_{\mathrm{DSP}}(op)}{1024} \end{pmatrix} \right\|$$

- Allows transformation with a Pareto improvement
- Resulting order of operation costs
  Add < Exp < Mul < Div < Log < Pow

Adding LLVM to the mix

- Sequential computation in C generated from CellML equations → idiomatic DSL-like structure
- Use clang/LLVM as frontend
- Optimise on LLVM-IR (existing and custom opts)
- Reconstruct C code for ODoST input

Identifying redundancies

- Array accesses, function calls hinder optimisation

```
… = -0.1*(STATES[0]+50.0) / (exp(-(STATES[0]+50.0)/10.0) - 1.0);
```

same value?

Identifying redundancies

- Array accesses, function calls hinder optimisation

```
… = -0.1*(STATES[0]+50.0) / (exp(-(STATES[0]+50.0)/10.0) - 1.0);
```

same value?

- But we know:
  - Input arrays do not alias or overlap
  - Function calls are mathematical operators, side effect-free

Identifying redundancies

- Augment the IR with this domain knowledge to help alias analysis
    - Mark input pointers as `noalias`
    - Map function calls to LLVM intrinsics
- LLVM's global value numbering can now identify expressions across the whole equation system
- Equation system ≅ Dataflow graph

Existing optimisation patterns in LLVM

- -instcombine pass
  - Constant folding & algebraic identities
  - Add < Mul < Div in software compiler as well

Existing optimisation patterns in LLVM

- -instcombine pass
  - Constant folding & algebraic identities
  - Add < Mul < Div in software compiler as well
  - Some transformations only if unsafe FP transformations are allowed
    e.g. $x/c = x \cdot 1/c$ only safe if reciprocal is exact

Domain-specific optimisations

Higher-order powers

- Equations contain x^p with an integer constant
- $8 \cdot c(\text{Mul}) < 1 \cdot c(\text{Pow})$
- Use Knuth’s binary exponentiation method

Op	*ALM*	*DSP*	*c(•)*
Mul	132	4	0.39
Pow	2058	31	3.18

  - lower generic power operator to short sequence of multiplications
  - Example: $x^6 = (\underbrace{(x \cdot x)}_{:= y} \cdot y) \cdot y$

A closer look at the exponential function

$$e^{x + \underline{c}} \cdot \underline{d}$$

Common pattern!

constants underlined

Add < Exp < Mul < …

A closer look at the exponential function

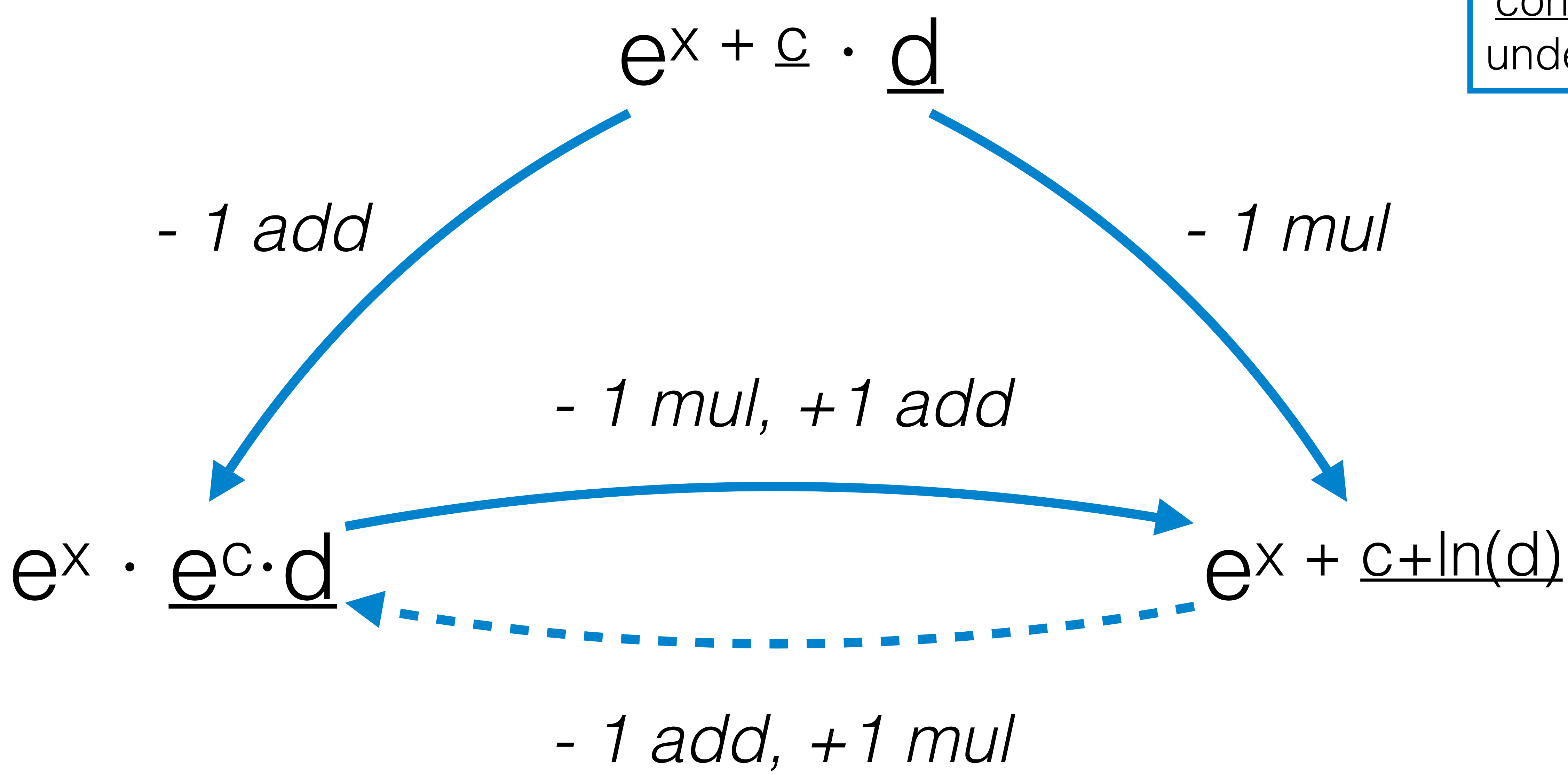
$$e^{x + \underline{c}} \cdot \underline{d}$$

<u>constants</u>
underlined

Add < Exp < Mul < …

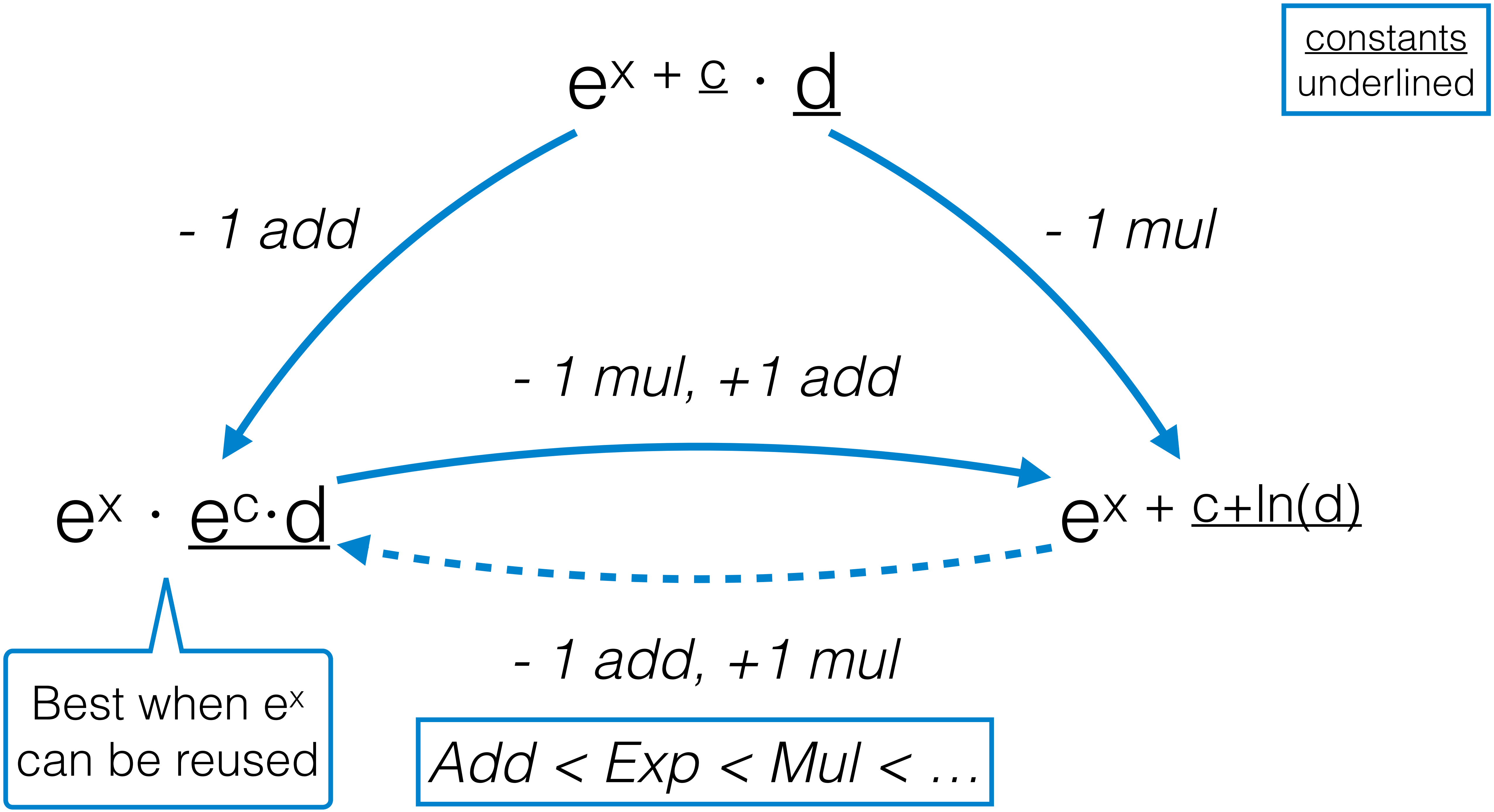
A closer look at the exponential function

constants underlined

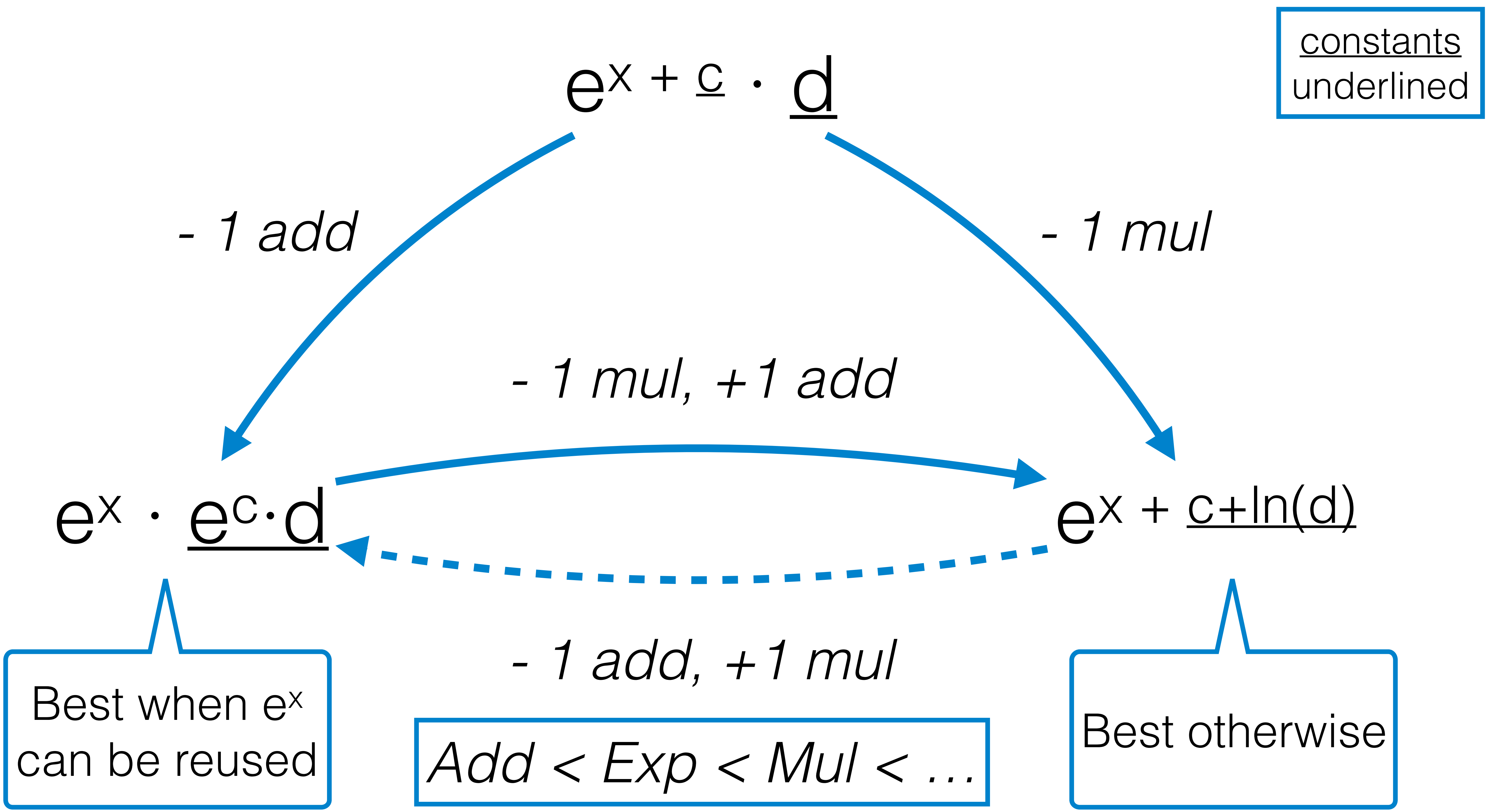


Add < Exp < Mul < …

A closer look at the exponential function



A closer look at the exponential function



Multiple Constant Multiplication

- x is multiplied with a set of constants c_i
- Can trade 1 multiplication for 1 addition if:
  - $c_2 = 2 \cdot c_1 \quad \rightarrow \quad x \cdot c_2 = (x \cdot c_1) + (x \cdot c_1)$
  - $c_4 = c_3 + 1 \quad \rightarrow \quad x \cdot c_4 = (x \cdot c_3) + x$
- Handle factors in ascending order of absolute values
  - Works also for chains of constants, e.g. 2, 3, 4, 6

Results

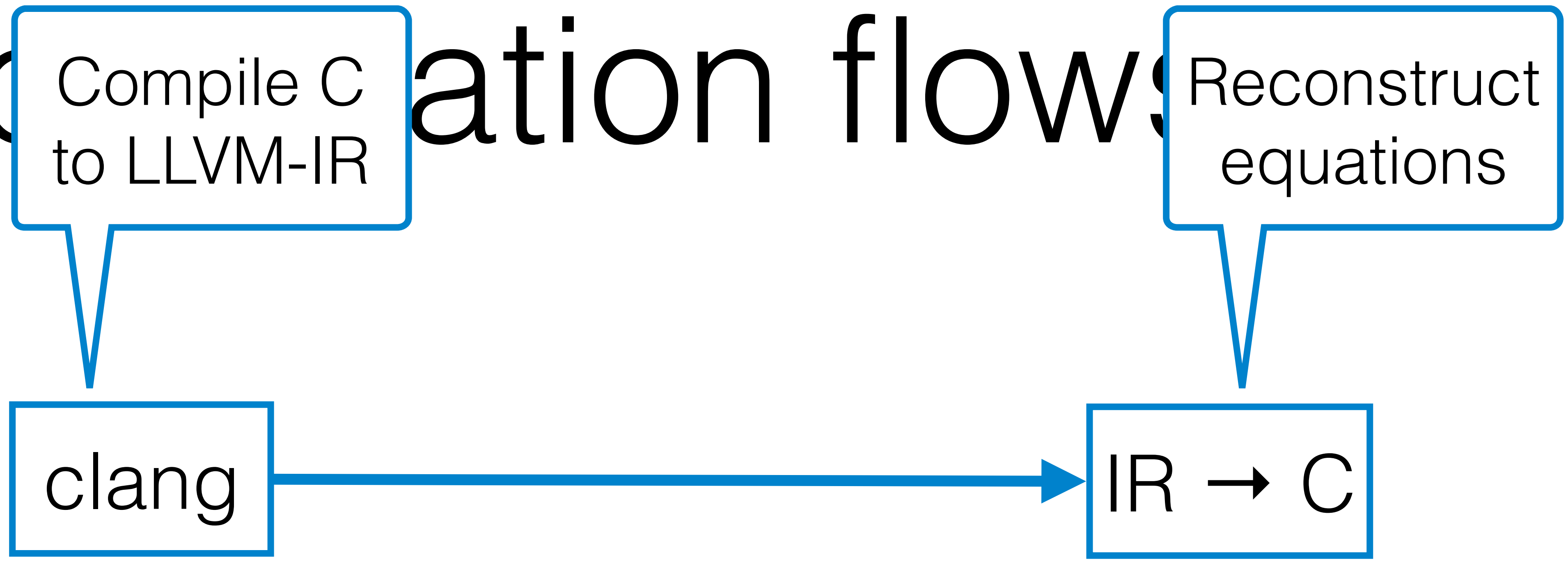
Compilation flows

Compilation flows

NoOpt:

Co ation flows

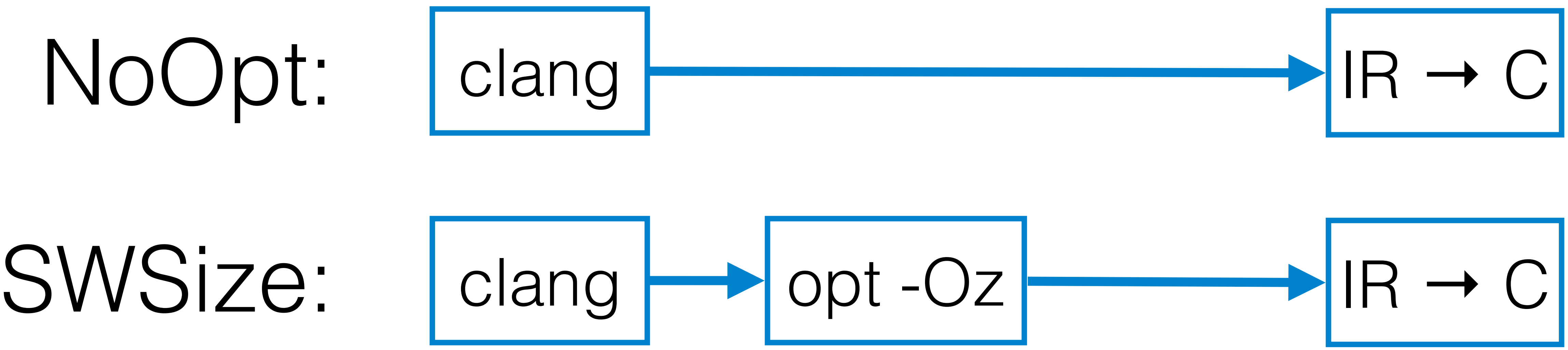
NoOpt:



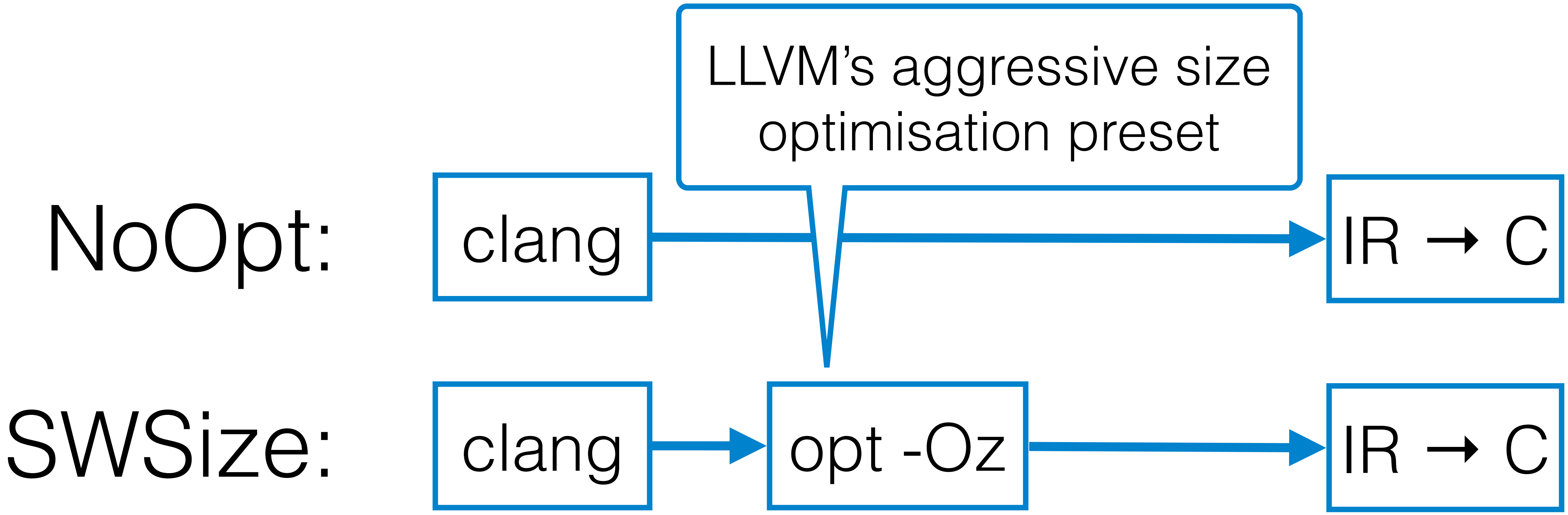
Compilation flows

NoOpt:

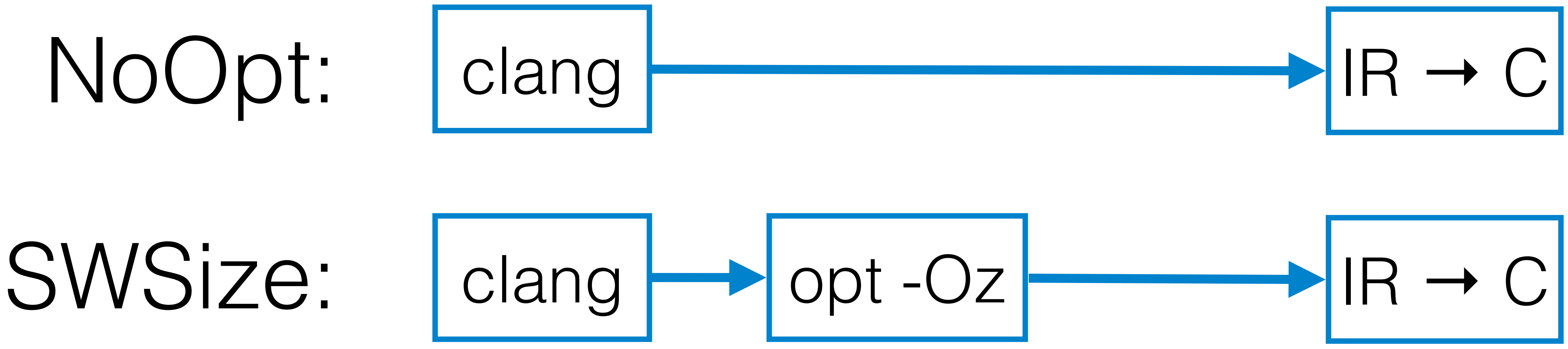
Compilation flows



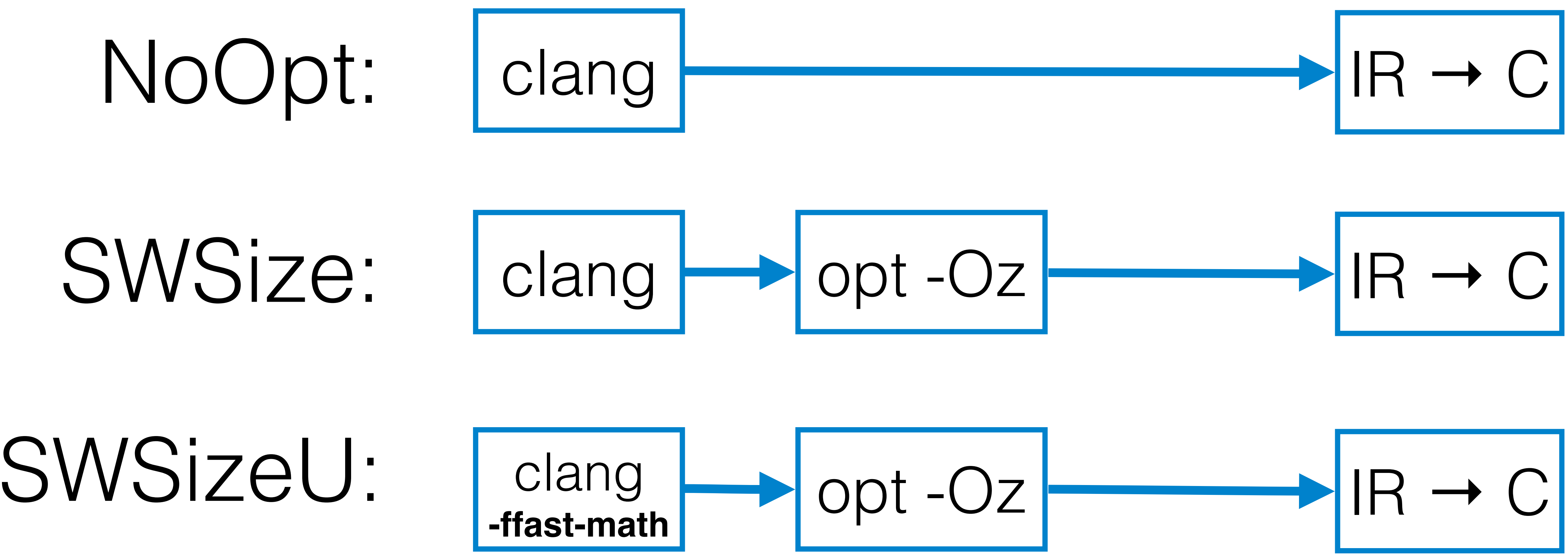
Compilation flows



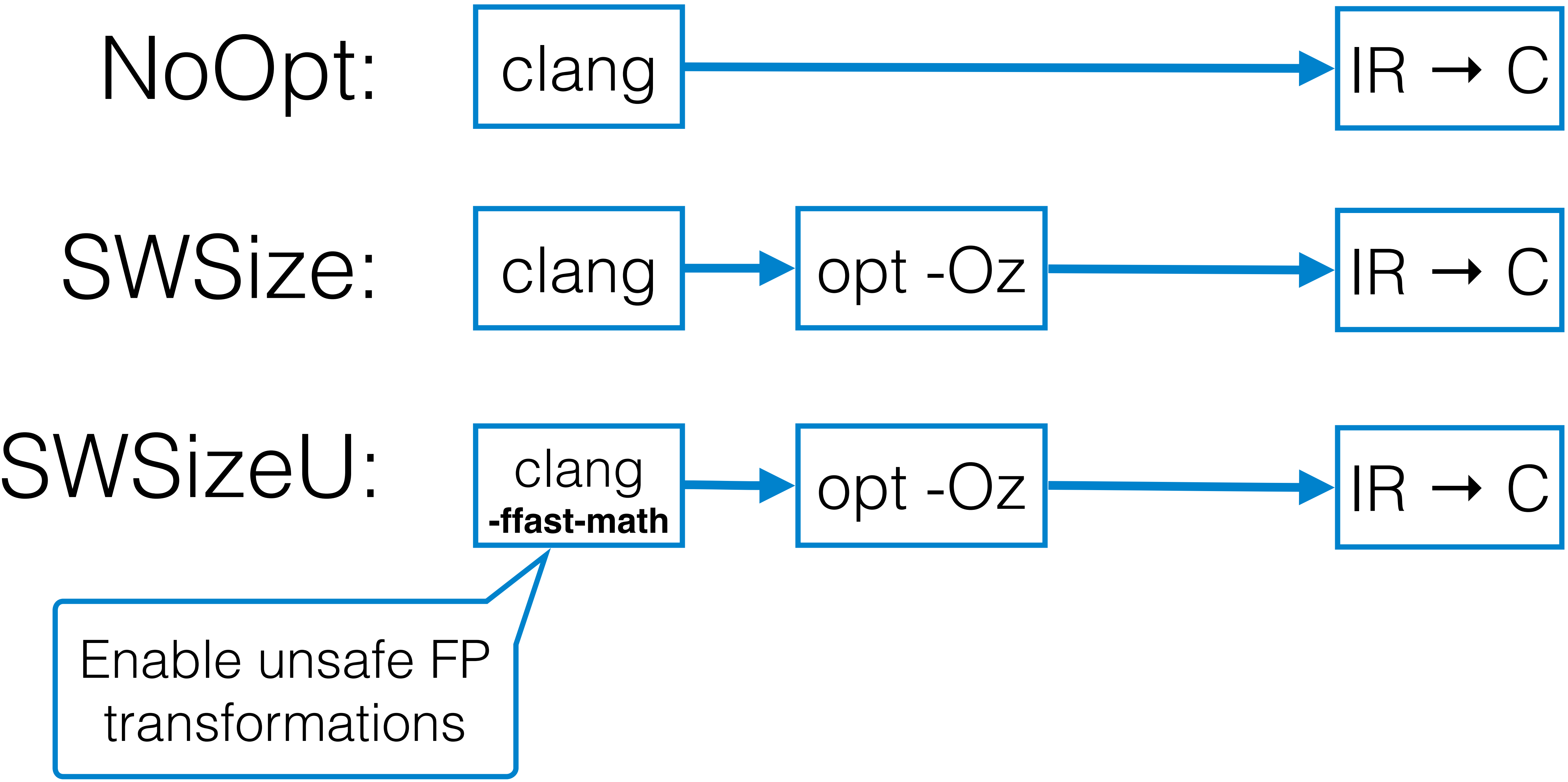
Compilation flows



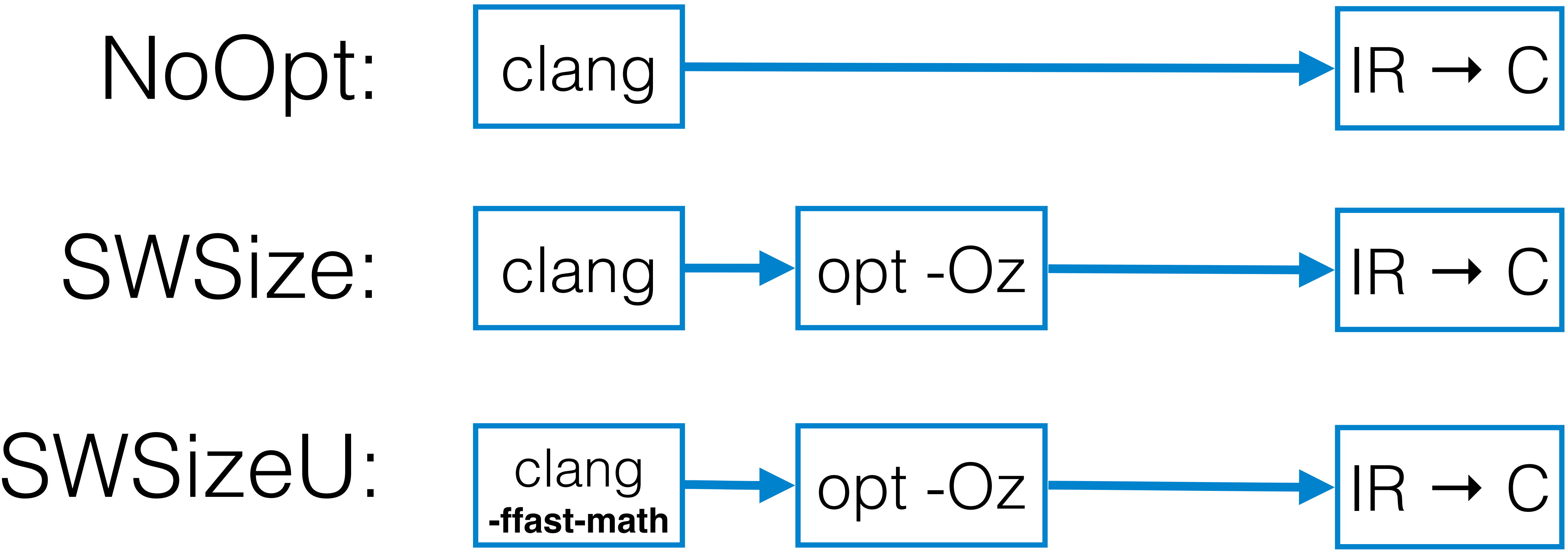
Compilation flows



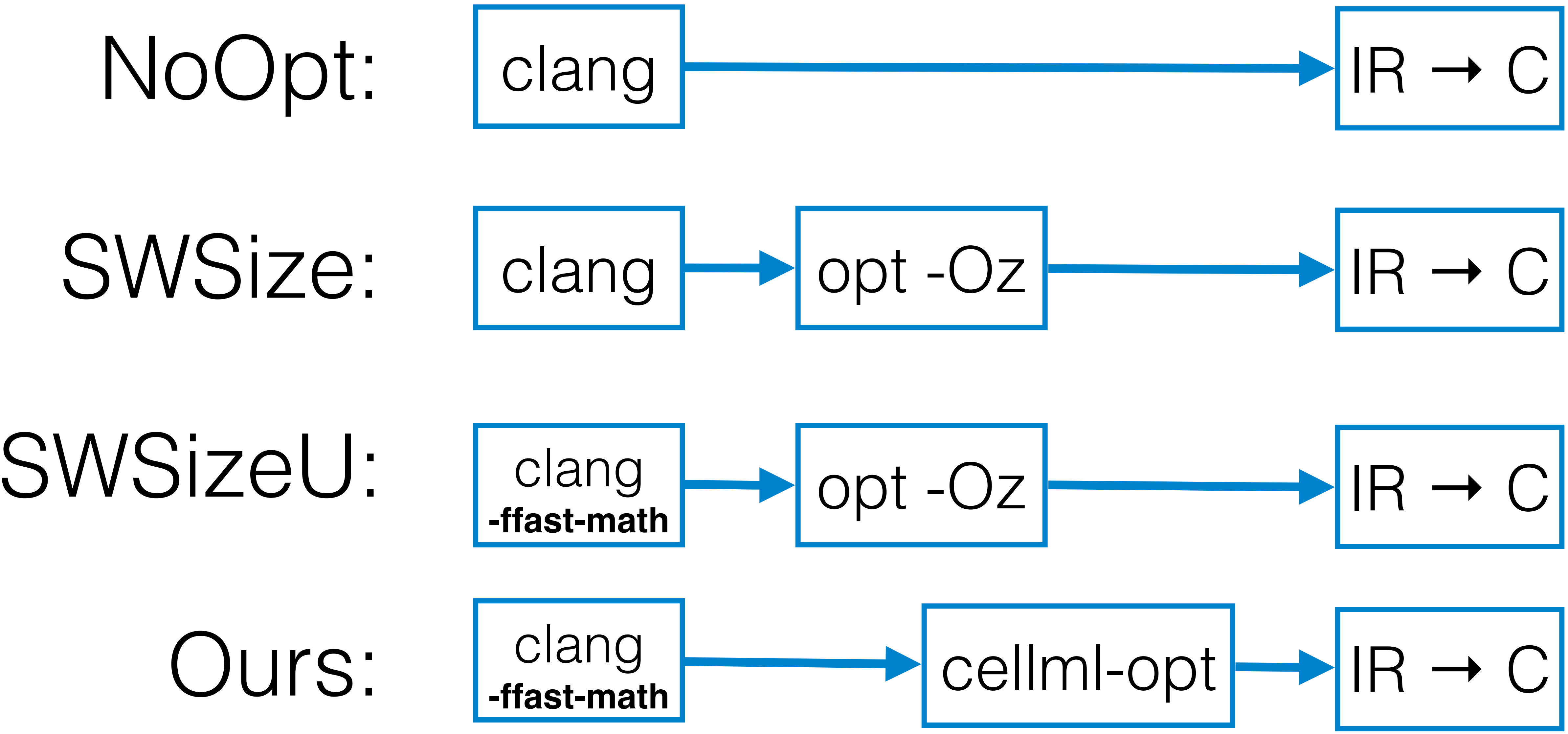
Compilation flows



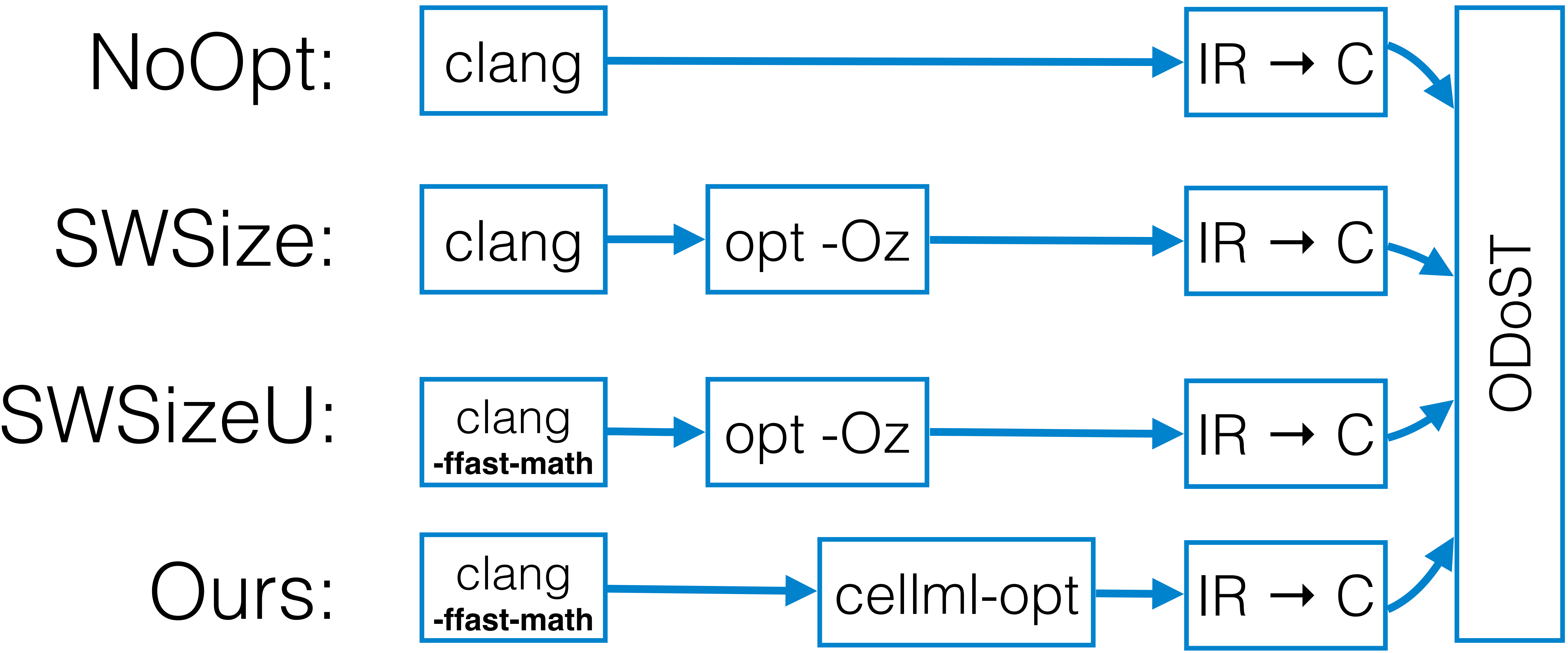
Compilation flows



Compilation flows



Compilation flows

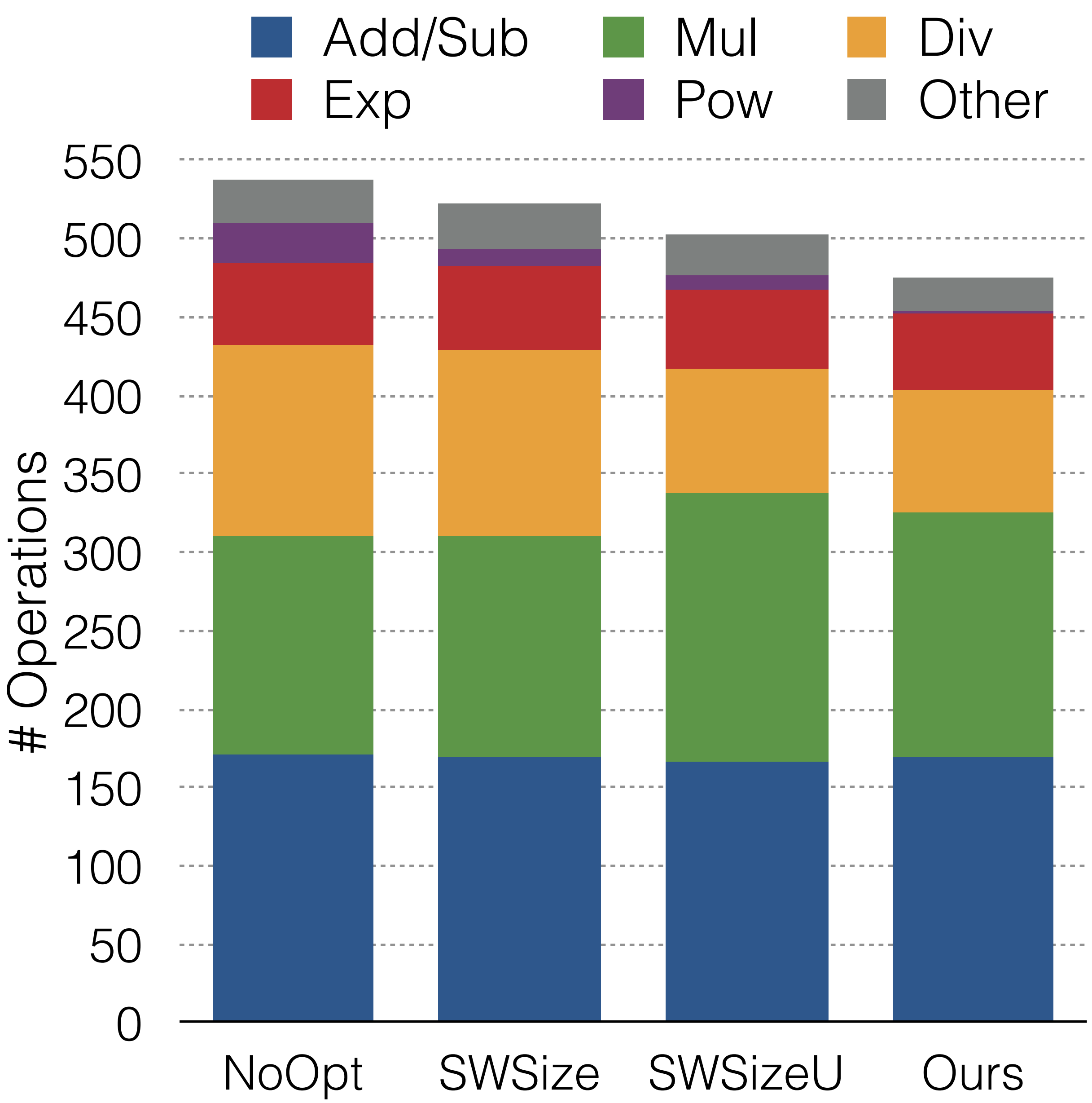


Error measurement

- Generic driver, 1000 integration steps of 1 μs each, starting at t = 1.0 s
- Compare computed values **before / after optimisation**, calculate relative error
- Certain, model specific deviation is acceptable
  - e.g. precision of “wet biology experiments” ~ 0.01 %

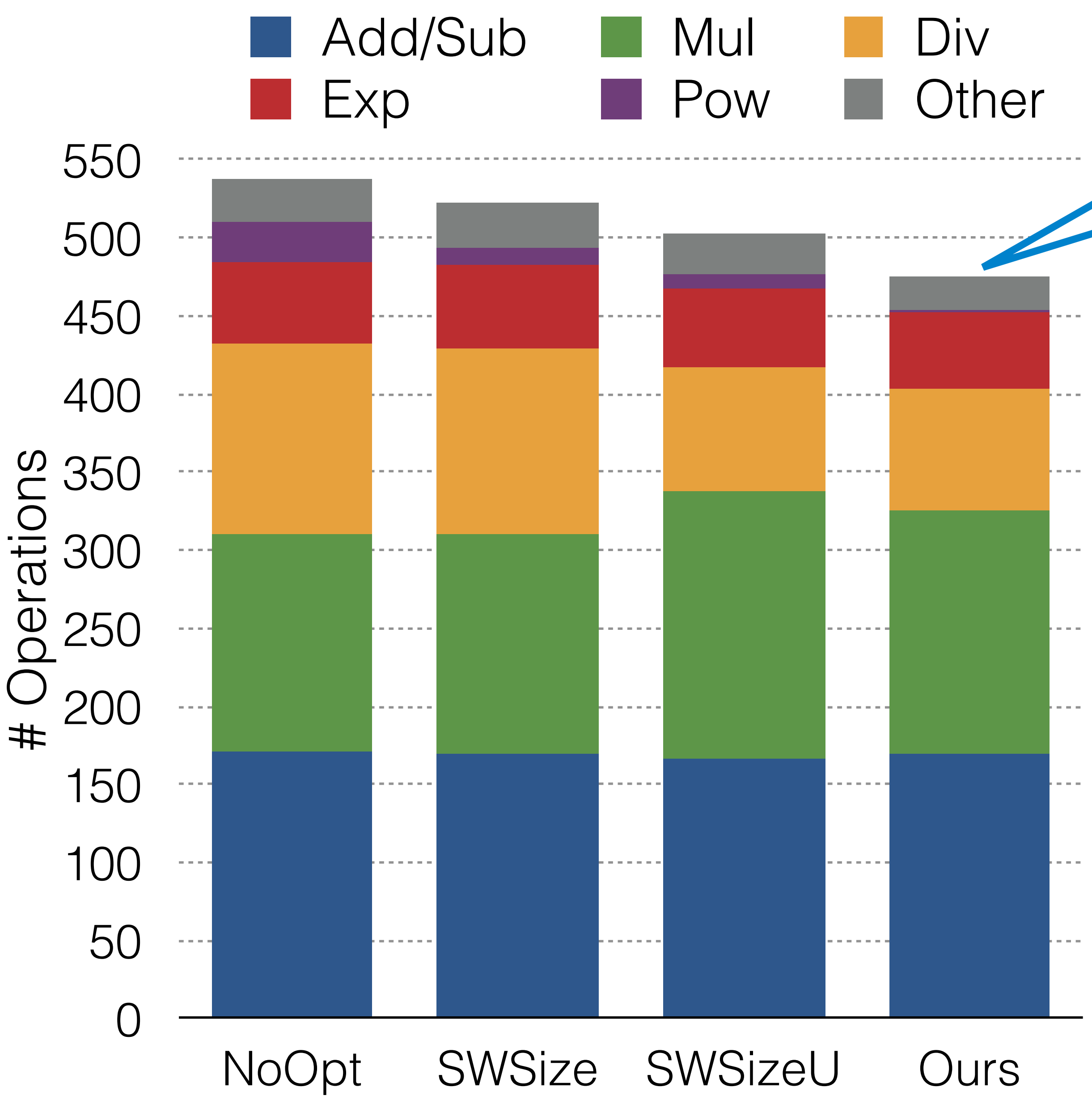
Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

Example model



Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

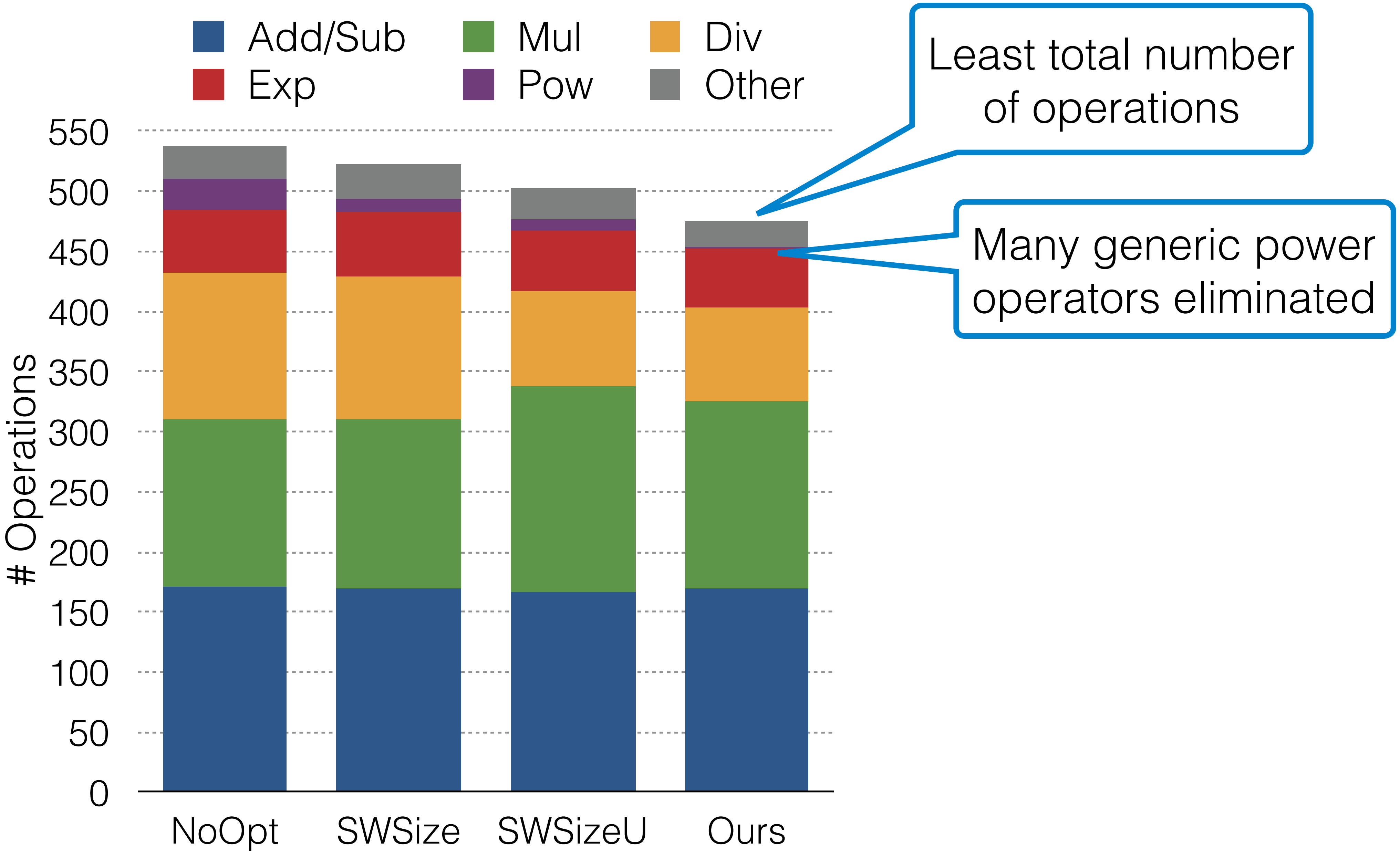
Example model



Least total number of operations

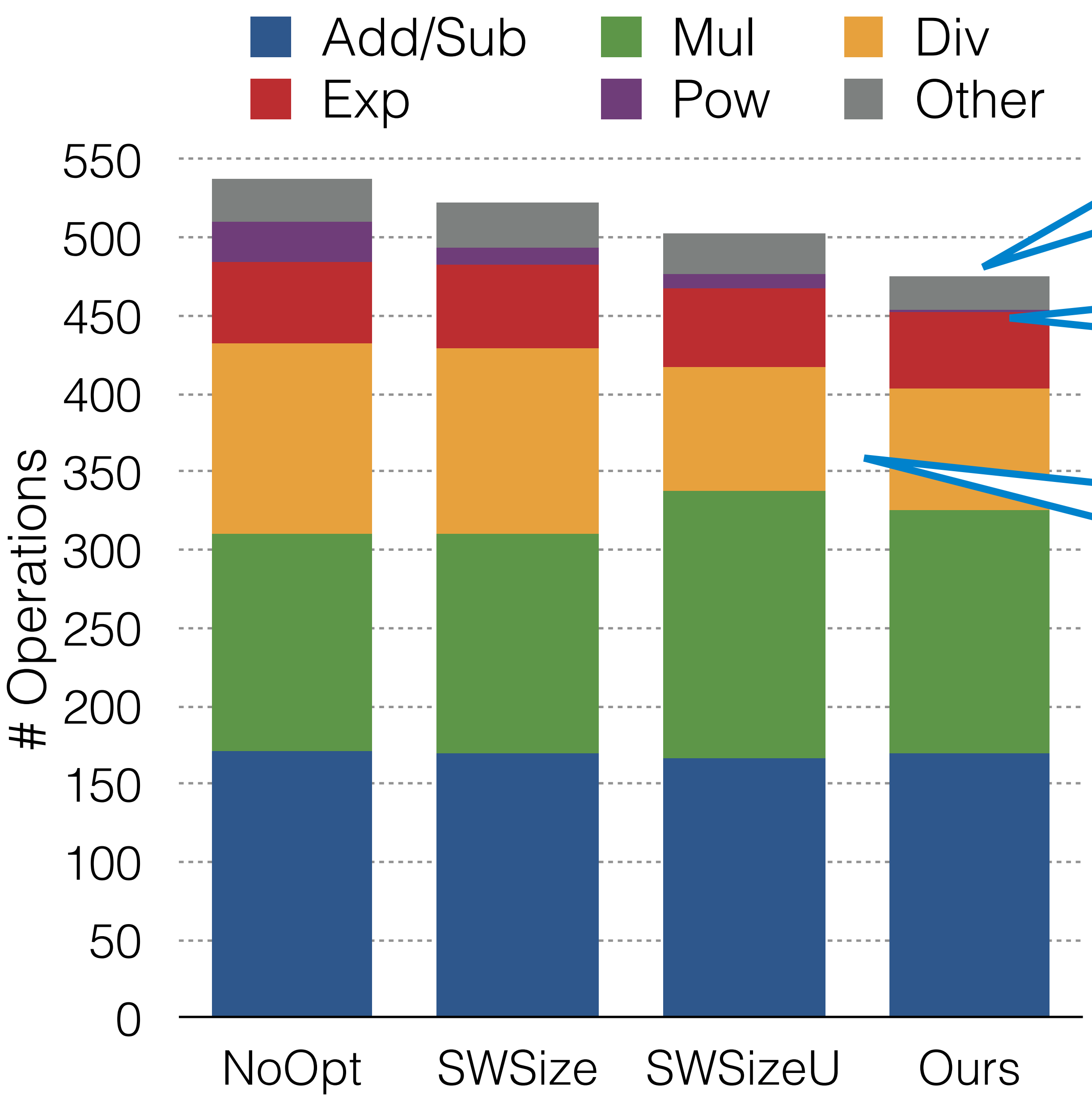
Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

Example model



Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

Example model



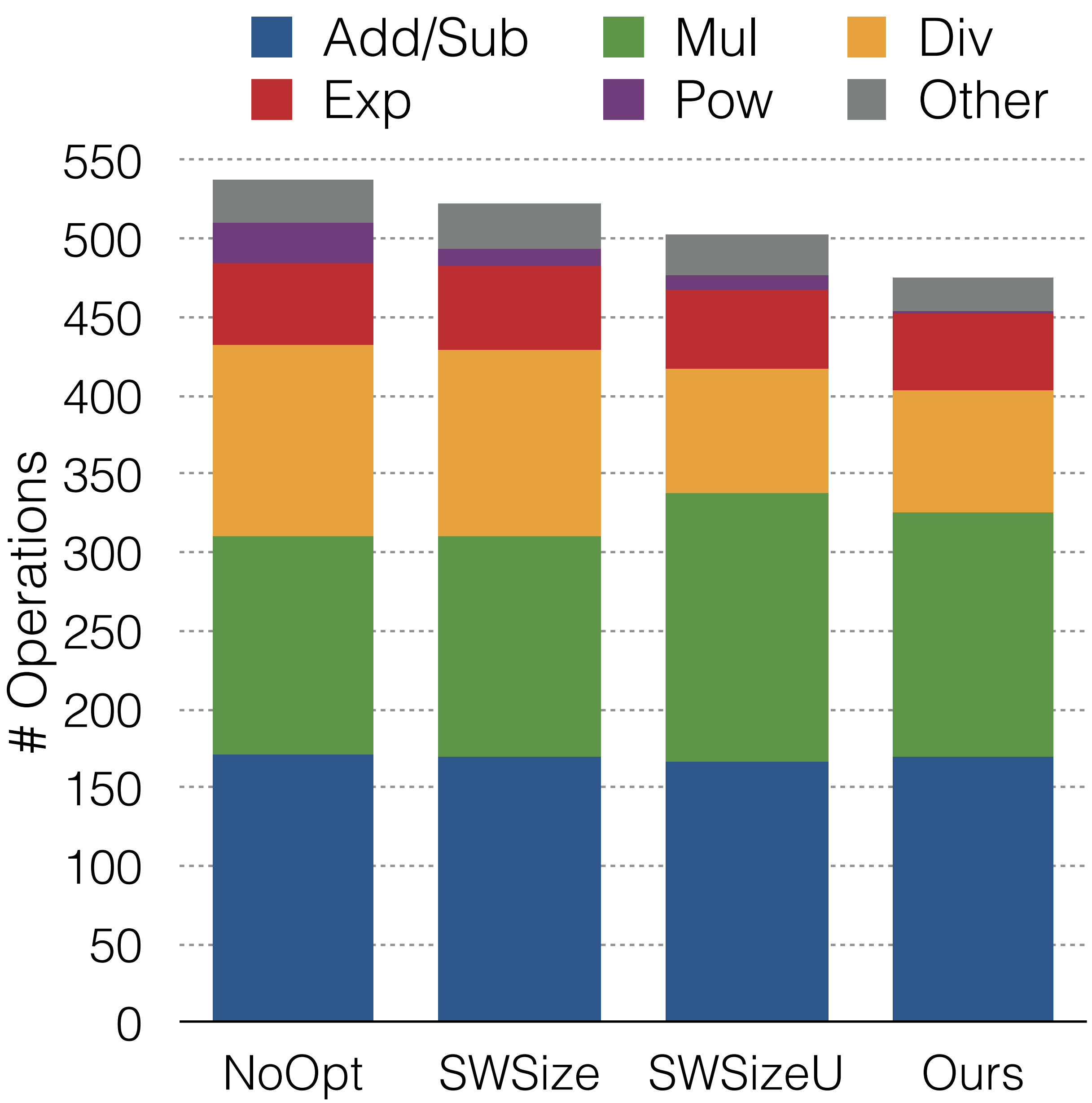
Least total number of operations

Many generic power operators eliminated

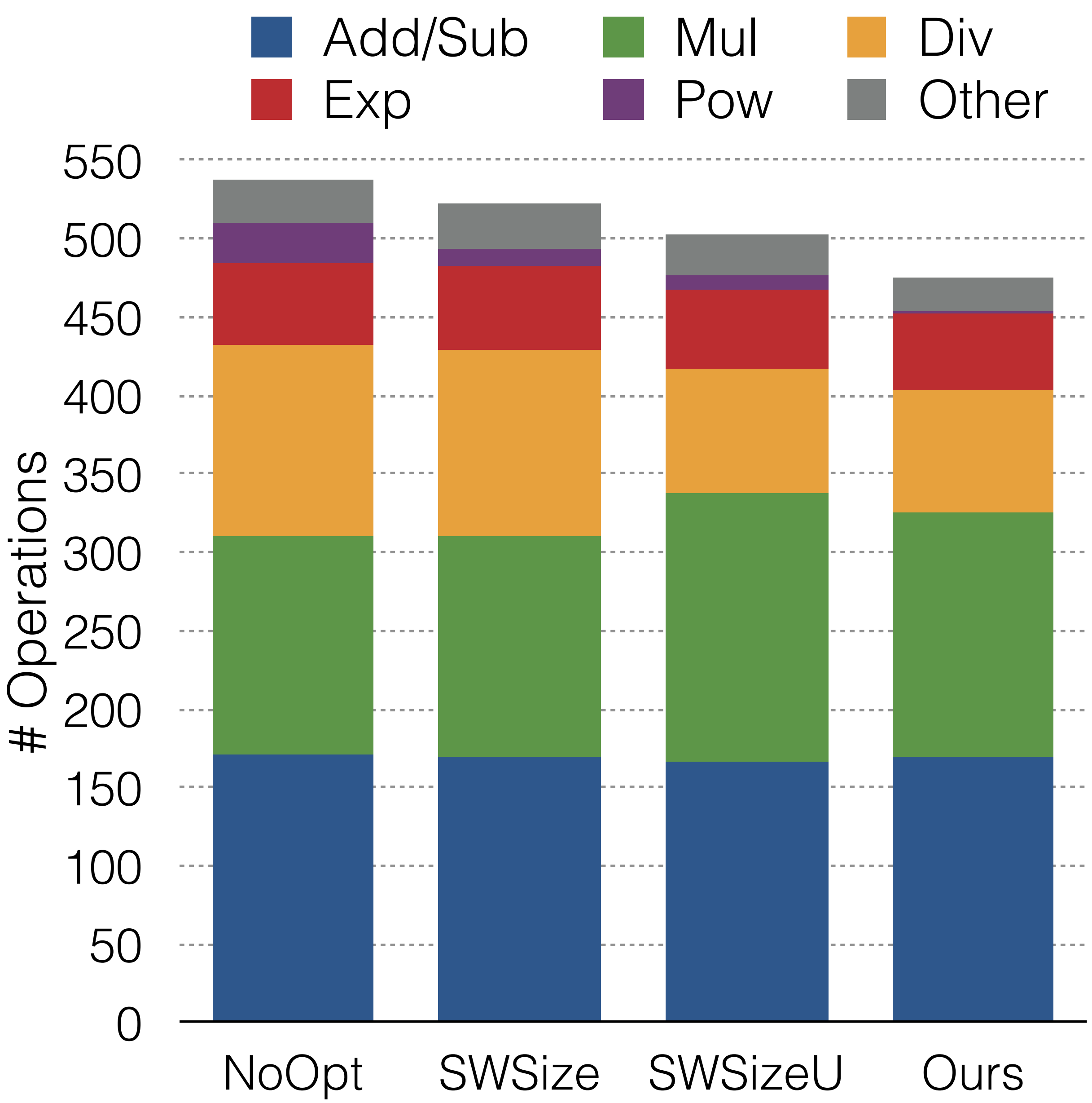
Transform many div to mul when unsafe FP is allowed

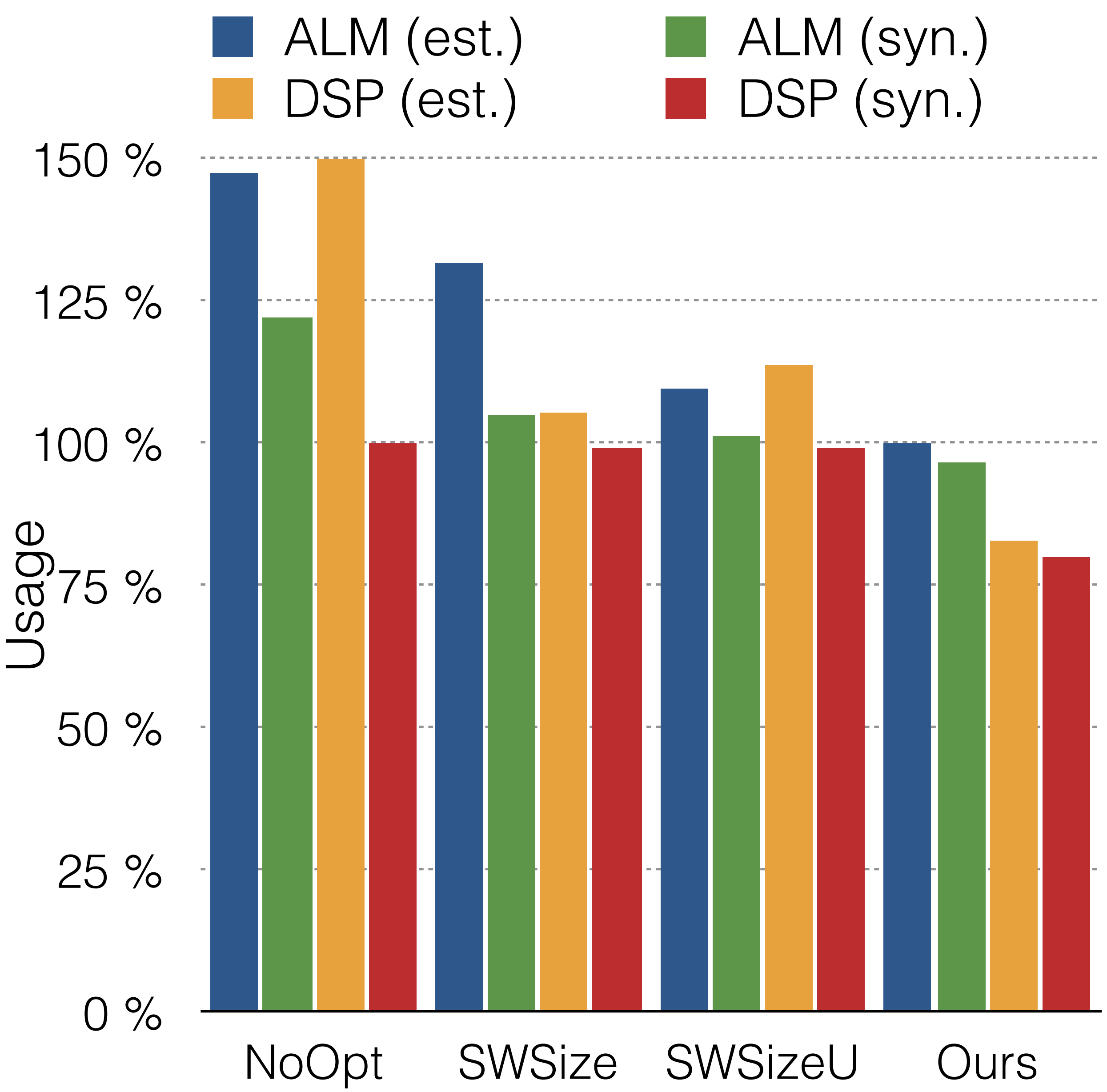
Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

Example model



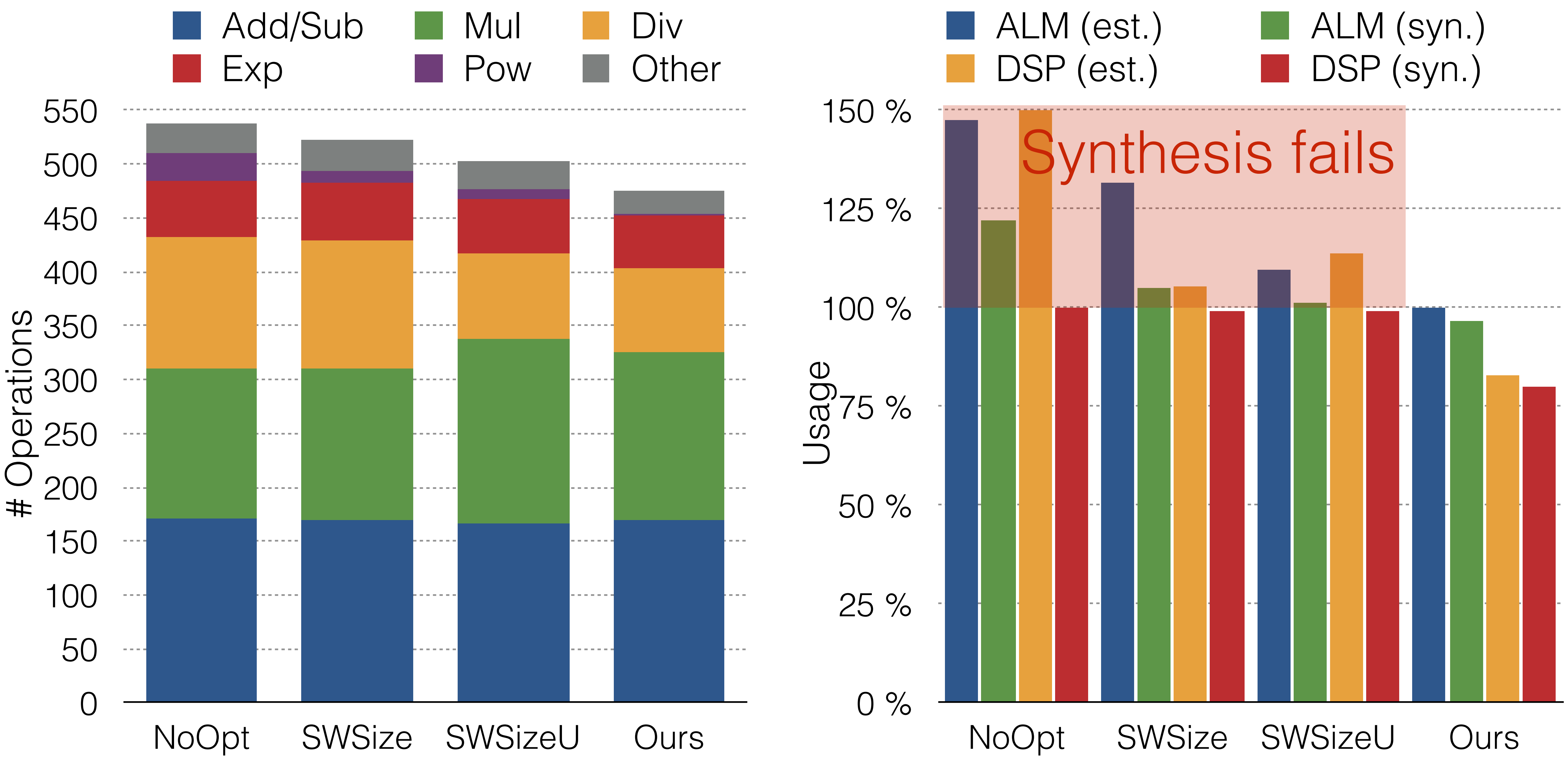
Example model





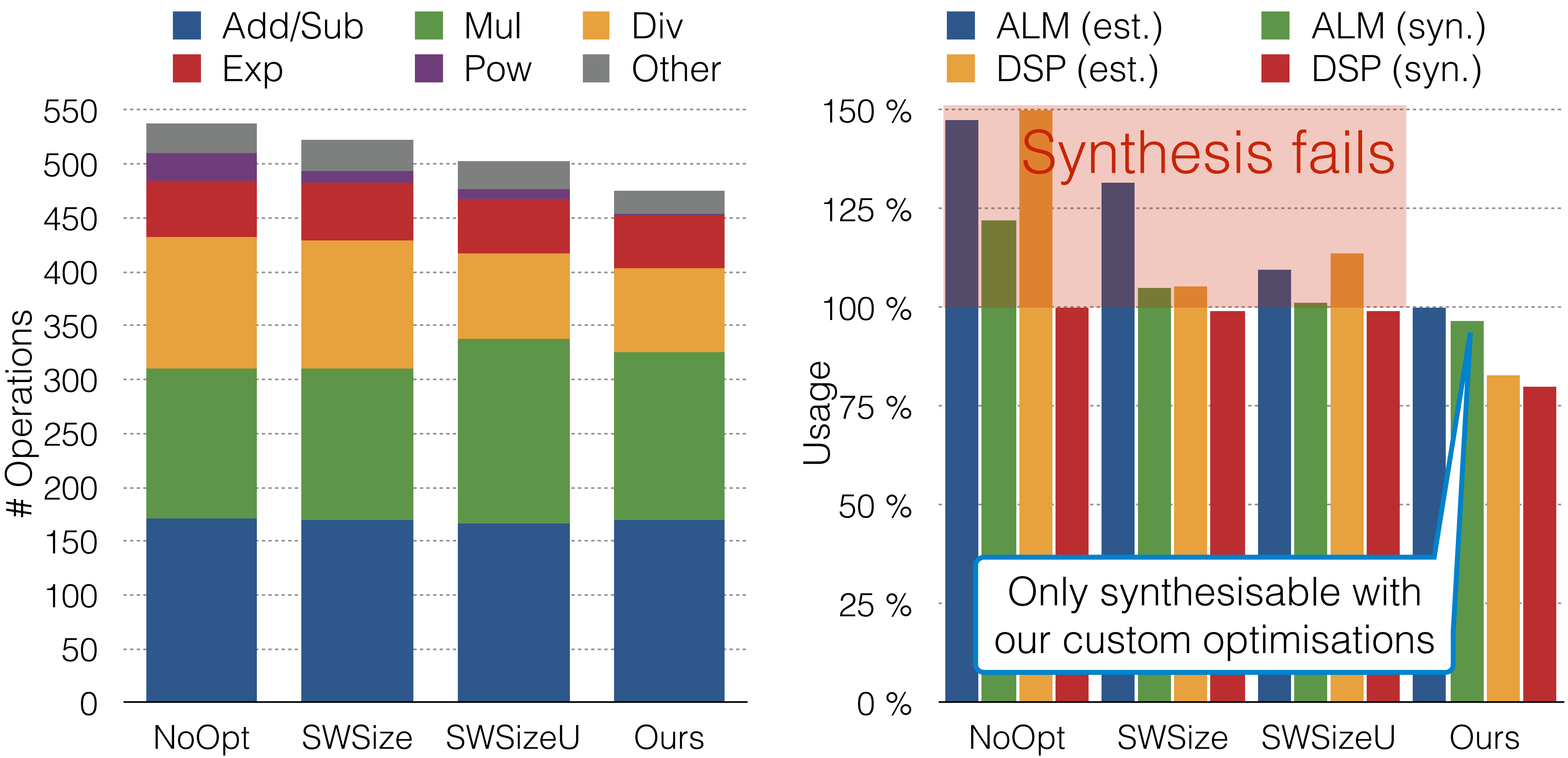
Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

Example model



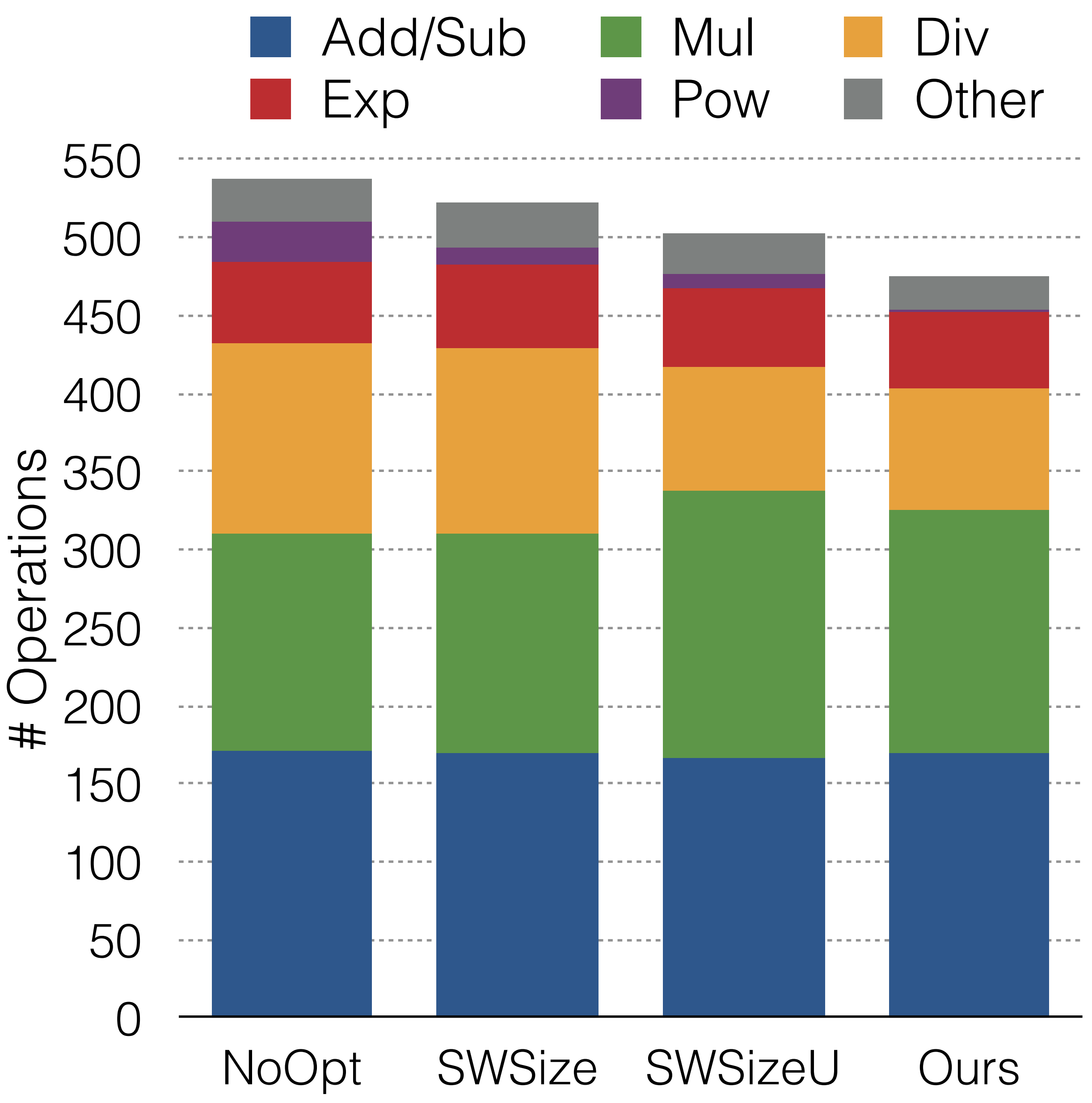
Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

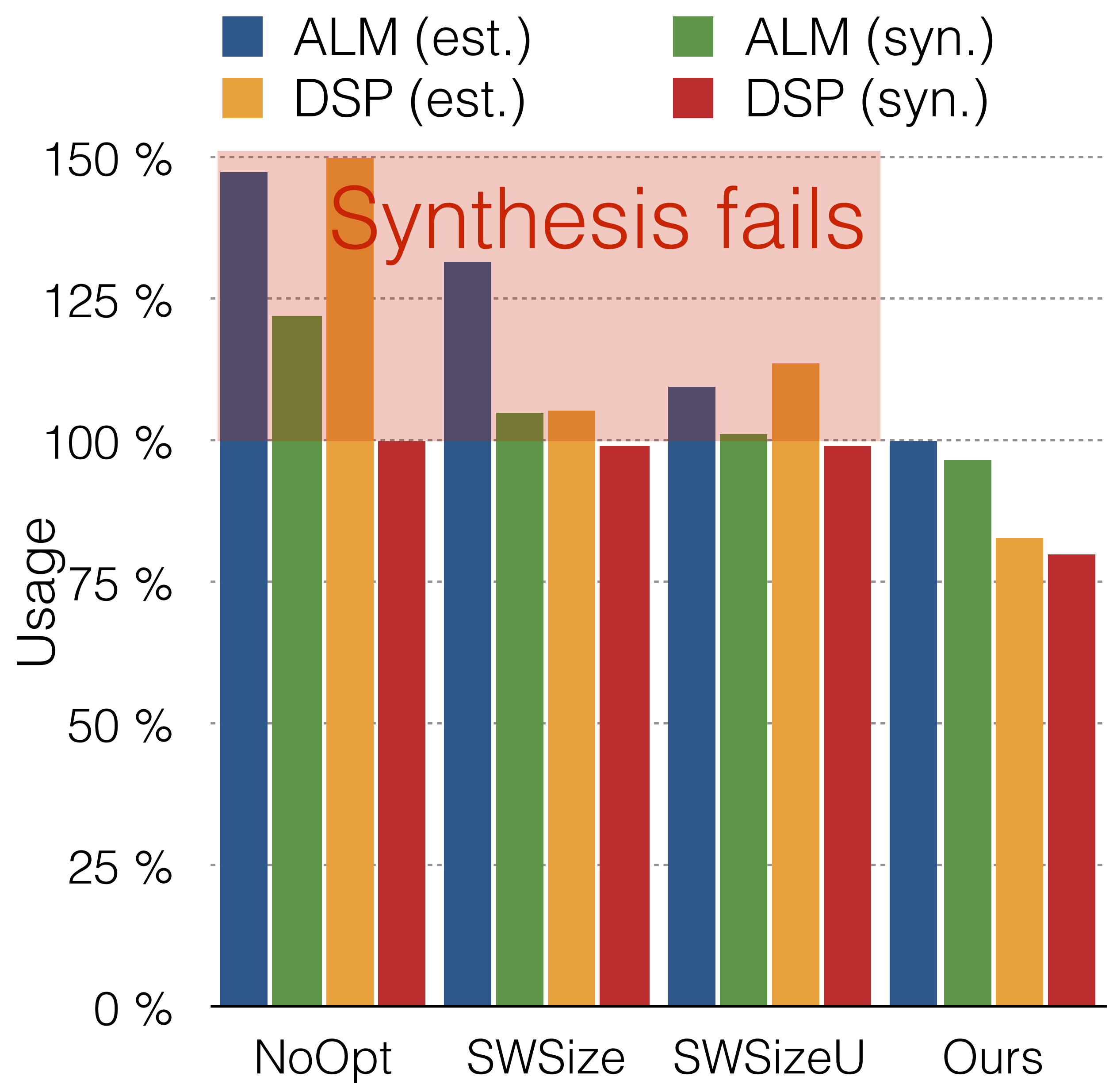
Example model



Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

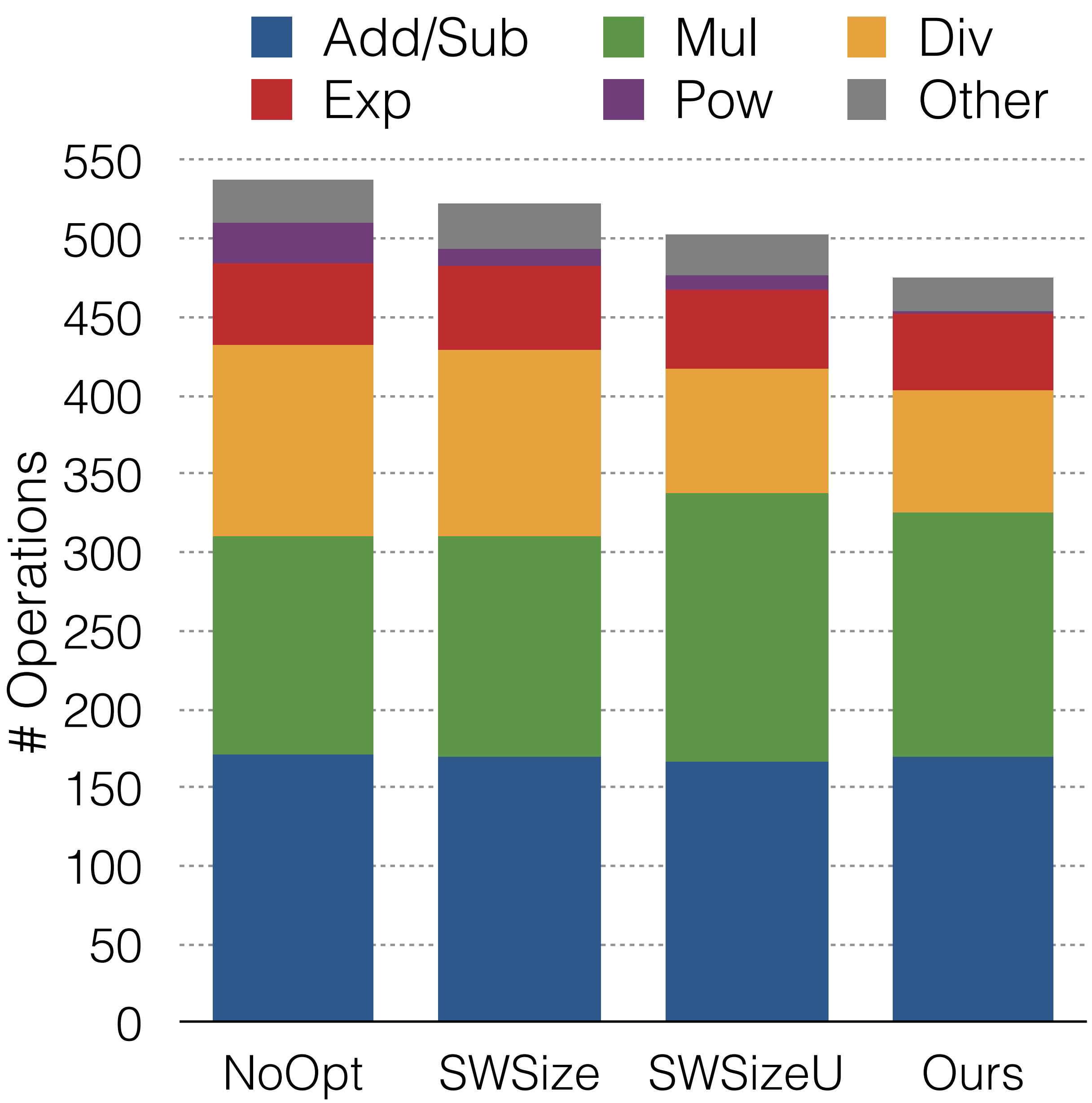
Example model

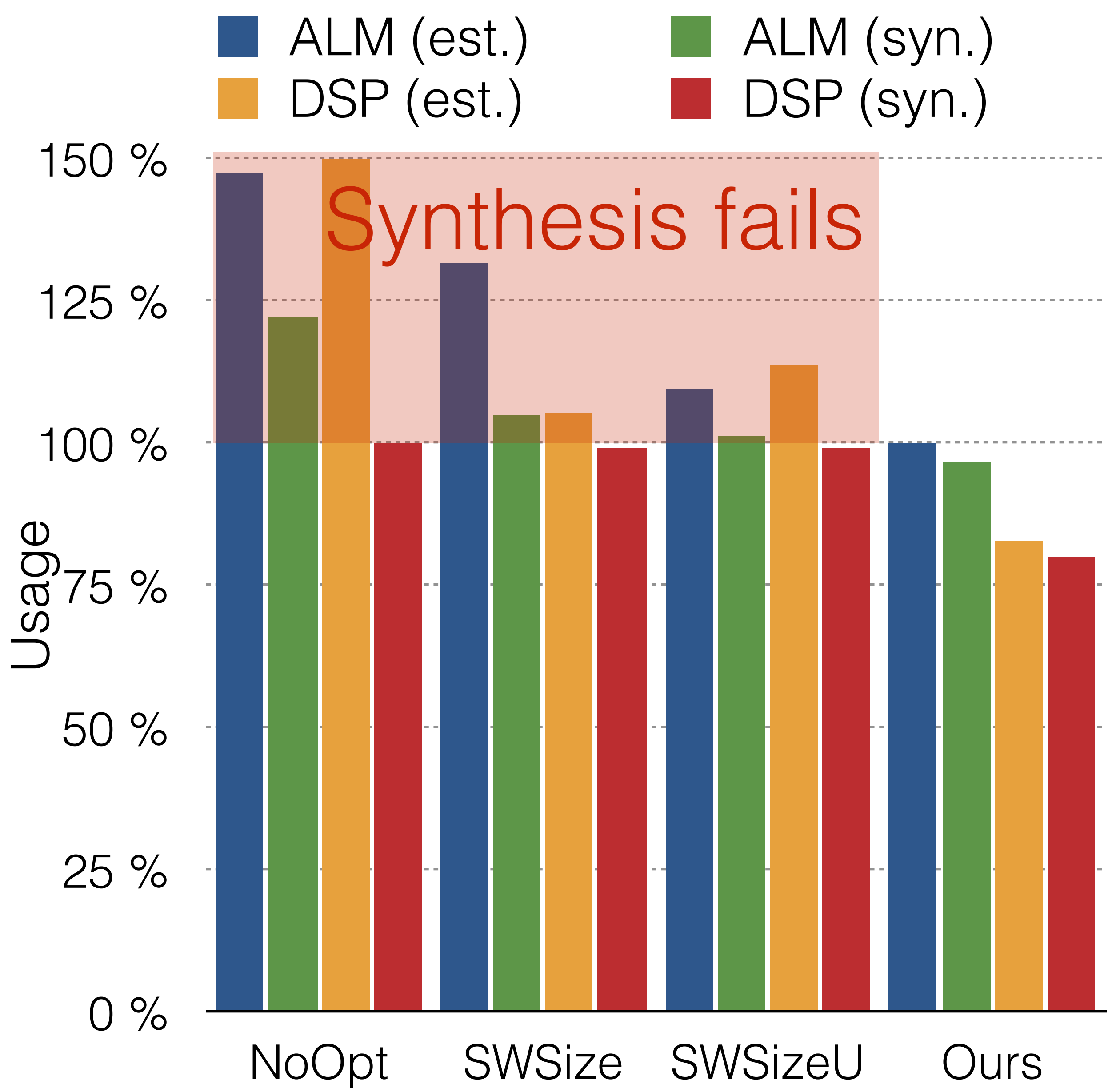




Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

Example model

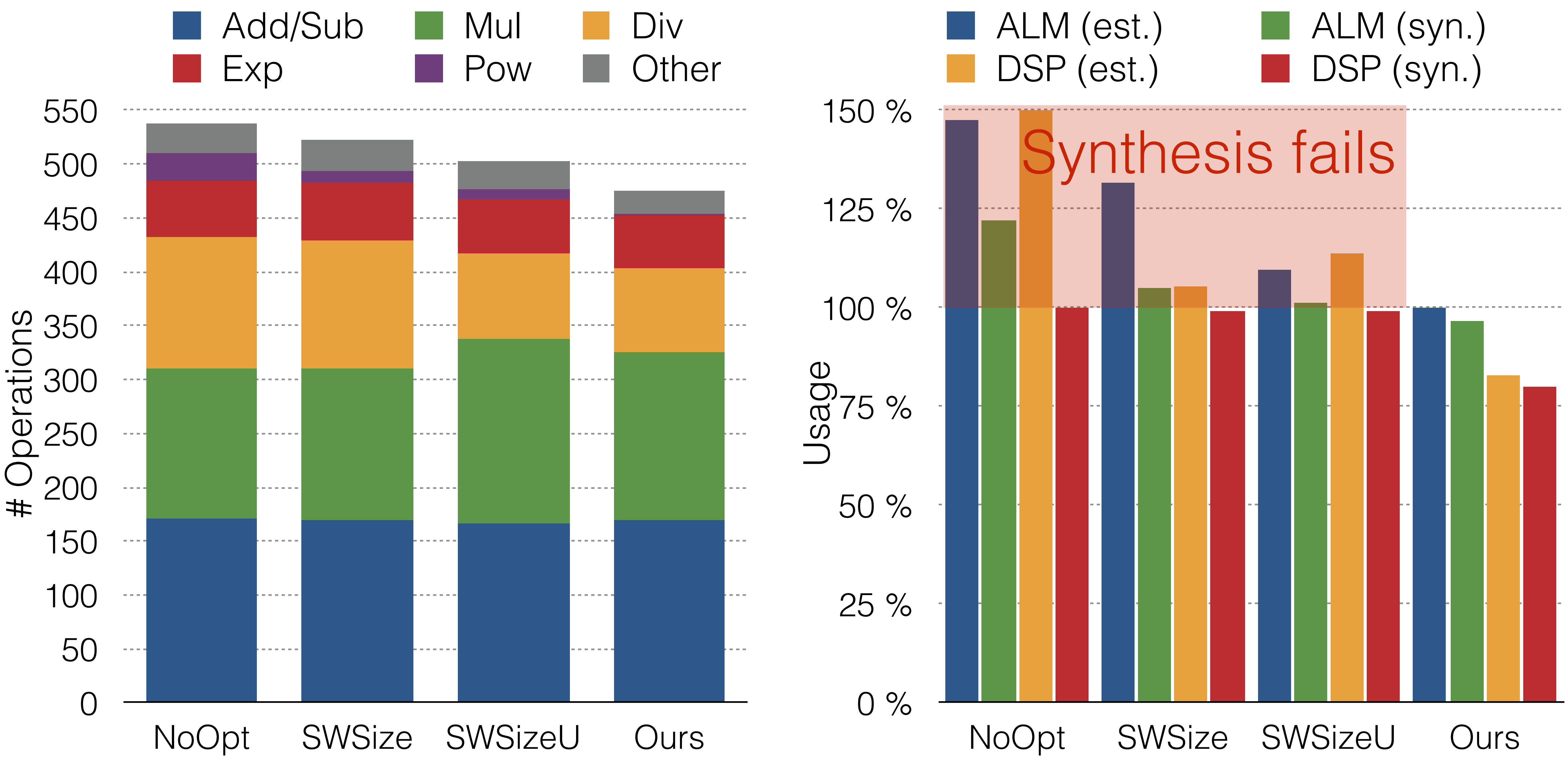




	SWSizeU	Ours
Rel. Err [%]	0.00054	0.0012
F_{max} [MHz]	-	111

Ten Tusscher, K. H. W. J., Noble, D., Noble, P. J., & Panfilov, A. V. (2004). A model for human ventricular tissue.

Example model



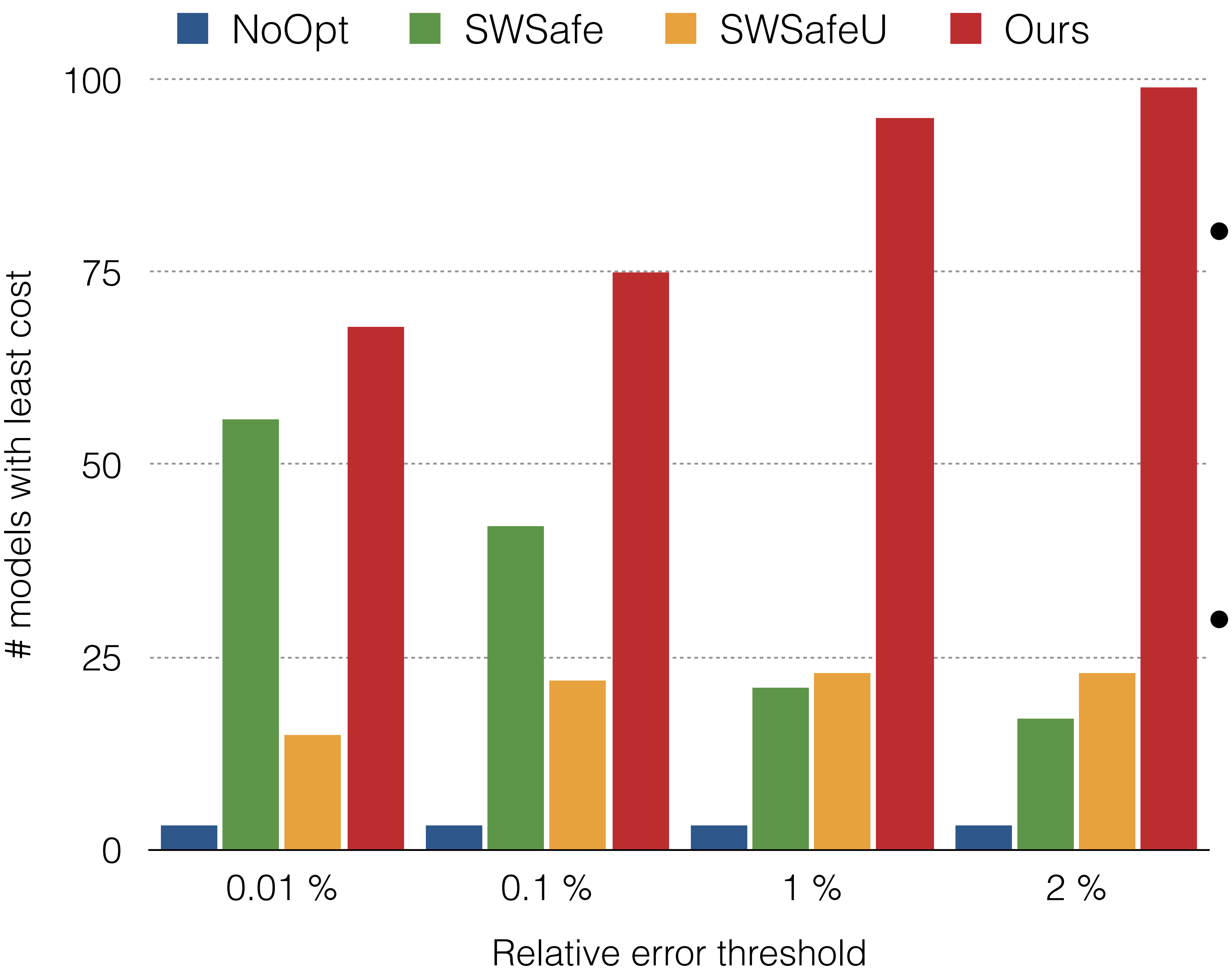
	SWSizeU	Ours
Rel. Err [%]	0.00054	0.0012
F_{max} [MHz]	-	111

Slightly larger error

General applicability

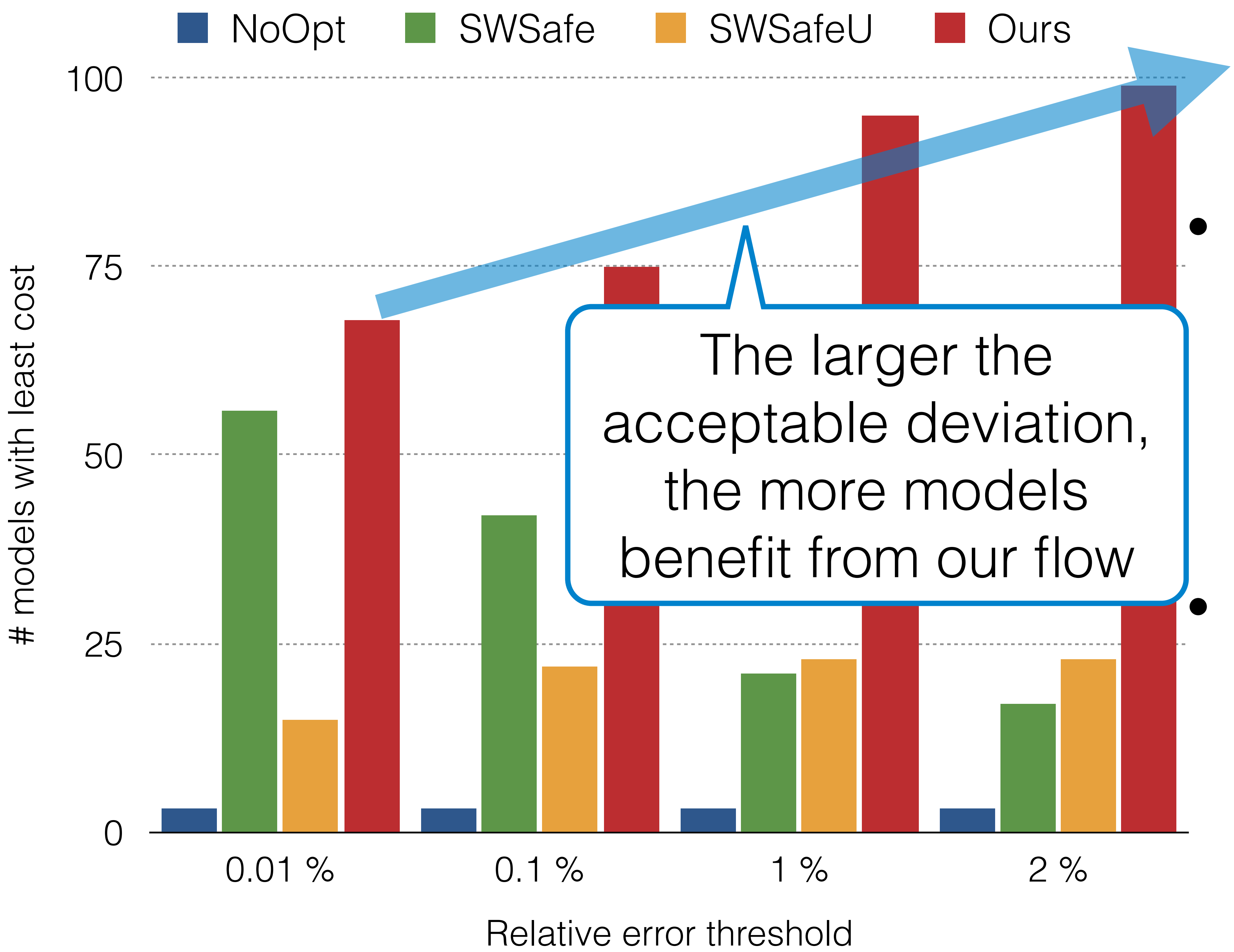
- 146 models from the CellML repository
  (> 20 equations, operators available as intrinsics, converge in input interval, 2+ curation stars)
- 4 thresholds for maximum relative error per model
- Use the cost model to estimate impact of transformations

Least cost per flow



- Count models with least cost after optimisation with a given flow
- If error > threshold, fall-back to SWSafe

Least cost per flow



- Count models with least cost after optimisation with a given flow
- If error > threshold, fall-back to SWSafe

Summary

- Size reduction after synthesis in 4 example models
  - Our recipe: up to 25 % less ALM, 20 % less DSP
  - Never worse than unoptimised (c.f. other flows)
- Broad applicability for domain-specific optimisations across 146 models

Future work

- Cost model served us well as quantitive instrument
  - Estimation of DSP usage ok
  - More accurate estimation of ALM demand needed
- A priori error analysis instead of empirical study

Thank you!