Code Generators for Mathematical Functions

N. Brunie [1], F. de Dinechin [2], **O. Kupriianova** [3], Ch. Lauter [3]

[1]**Kalray**, Grenoble, France
[2]Université de Lyon, INRIA, **INSA-Lyon, CITI**, F-69621 Villeurbanne, France
[3]Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, **LIP6**, F-75005 Paris, France

22nd IEEE Symposium on Computer Arithmetic
Lyon, France, 22-24 June 2015

Mathematical libraries

Standard libraries (libms)

- elementary functions ($\exp, \log, \sin, \sinh$)
- special functions (x^y, Γ, Bessel)
- standard precisions (single, double, quad)

Existing implementations

- Intel's MKL
- AMD's libm
- ARM's mathlib
- libmcr by Sun
- ...
- glibc libm
- CRLibm by ENS Lyon
- newlib
- OpenLibm for Julia
- Yeppp!

One size does not fit all

Current offer

- Several performance options (latency vs throughput)
- Several accuracy options
  ("quick and dirty", faithful, correctly-rounded)
- Several portability options (generic vs AVX512)

One size does not fit all

Current offer

- Several performance options (latency vs throughput)
- Several accuracy options
  ("quick and dirty", faithful, correctly-rounded)
- Several portability options (generic vs AVX512)

Some people are still not happy

- More performance, less compliance
  - degraded accuracy
  - reduced domain
- Functions not from standard libm

One size does not fit all

Current offer

- Several performance options (latency vs throughput)
- Several accuracy options
  ("quick and dirty", faithful, correctly-rounded)
- Several portability options (generic vs AVX512)

Some people are still not happy

- More performance, less compliance
  - degraded accuracy
  - reduced domain
- Functions not from standard libm

Who is going to write all these variants?

Solution

Metalibm

Write tools to produce code for math functions

Analogy

assembly $\rightarrow$ compilers

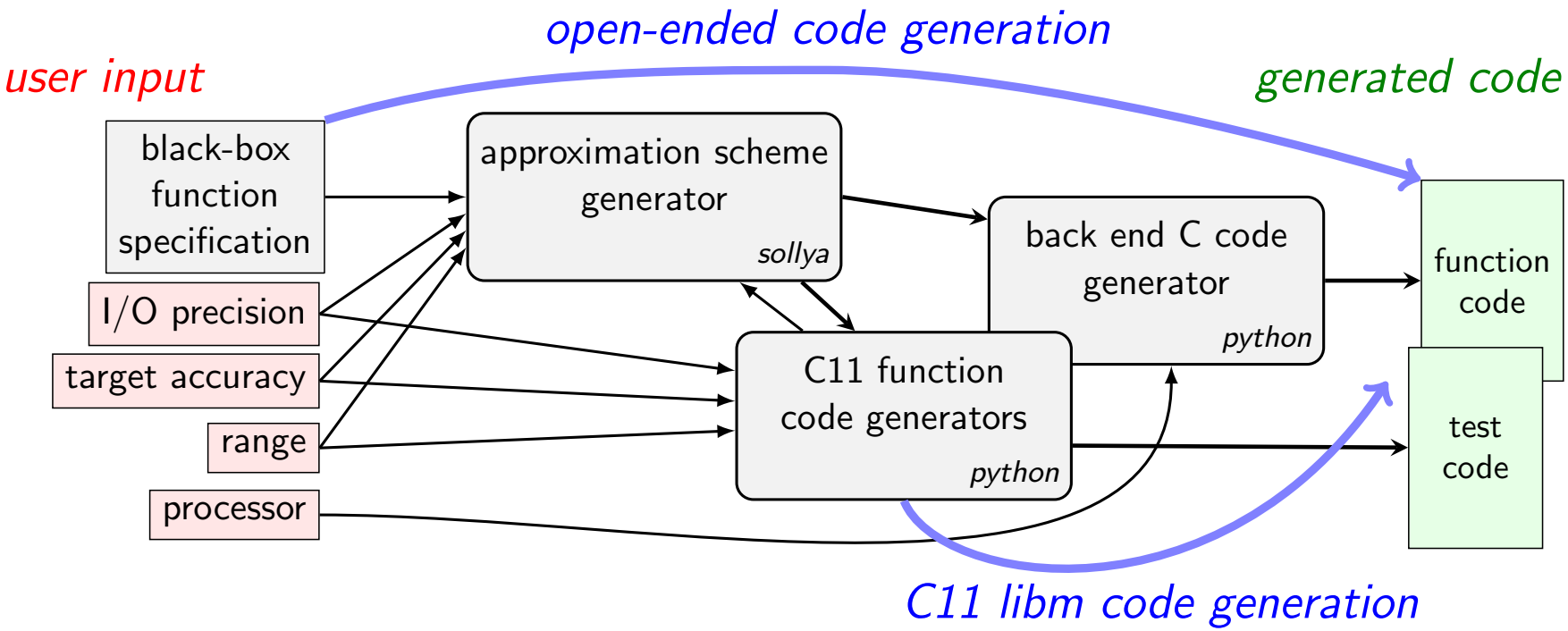
code $\rightarrow$ code generators

```
.cfi_startproc
subq        $8, %rsp
.cfi_def_cfa_offset 16
movl        $52, %r8d
movl        $37, %ecx
movl        $15, %edx
movl        $.LC0, %esi
movl        $1, %edi
xorl        %eax, %eax
call        __printf_chk
xorl        %eax, %eax
addq        $8, %rsp
.cfi_def_cfa_offset 8
ret
.cfi_endproc
```

```
#include <stdio.h>

int main() {
  int a, b, sum;
  a = 15;
  b = 37;
  sum = a + b;
  printf("%d + %d = %d\n", a, b,
         sum);
  return 0;
}
```

Metalibm: generator use-cases



Outline

How to implement a function manually

The task: from f on $[a, b]$ get an implementation `fun`: $\left|\frac{\texttt{fun} - f}{f}\right| \leq \bar{\varepsilon}$

How to implement a function manually

The task: from f on $[a, b]$ get an implementation `fun`: $\left|\frac{\texttt{fun} - f}{f}\right| \leq \bar{\varepsilon}$

1. Eliminating special cases: zeros, infinities, NaNs, etc.
2. Argument reduction: transform $[a, b]$ to $[\alpha, \beta]$, a small interval
3. Polynomial approximation:
   minimax approximation, polynomial of low degree, Remez algorithm
4. Reconstruction

How to implement a function manually

The task: from f on $[a, b]$ get an implementation `fun`: $\left|\frac{\texttt{fun} - f}{f}\right| \leq \bar{\varepsilon}$

1. Eliminating special cases: zeros, infinities, NaNs, etc.
2. Argument reduction: transform $[a, b]$ to $[\alpha, \beta]$, a small interval
3. Polynomial approximation:
   minimax approximation, polynomial of low degree, Remez algorithm
4. Reconstruction

Example

implement $f(x) = e^x$

$e^x = 2^{\frac{x}{\log 2}} = 2^{\left\lfloor \frac{x}{\log 2} \right\rceil} \cdot 2^{\frac{x}{\log 2} - \left\lfloor \frac{x}{\log 2} \right\rceil} = 2^E \cdot e^{x - E \log 2} = 2^E \cdot e^r$

Argument reduction

Based on mathematical properties:

$n^{a+b} = n^a \cdot n^b$, $\sin(x + 2\pi) = \sin(x)$, $\log(a \cdot b) = \log(a) + \log(b), \ldots$

Argument reduction

Based on mathematical properties:

$n^{a+b} = n^a \cdot n^b$, $\sin(x + 2\pi) = \sin(x)$, $\log(a \cdot b) = \log(a) + \log(b), \ldots$

What properties do we know for erf, J_0
or an open-ended function (purely defined by a differential equation)?

Argument reduction

Based on mathematical properties:

$n^{a+b} = n^a \cdot n^b$, $\sin(x + 2\pi) = \sin(x)$, $\log(a \cdot b) = \log(a) + \log(b), \ldots$

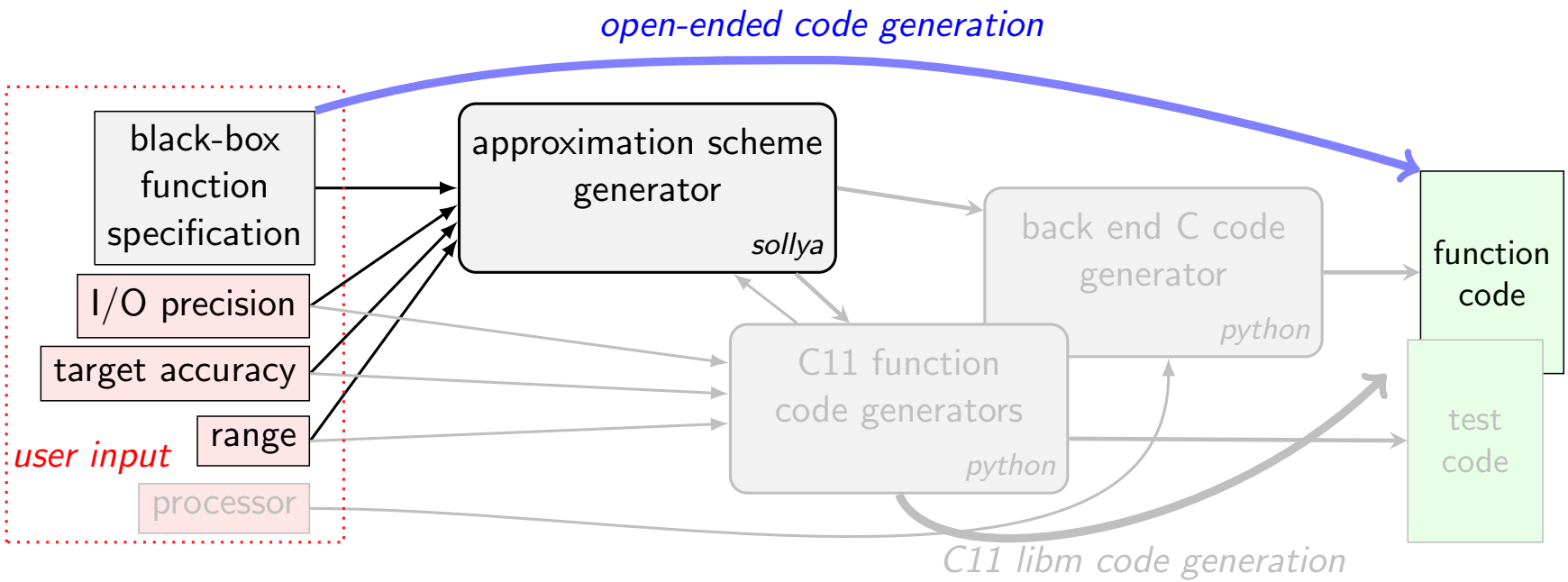
What properties do we know for erf, J_0
or an open-ended function (purely defined by a differential equation)?

When argument reduction does not work

Piecewise-polynomial approximation

Outline

Metalibm: generator use-cases



Philosophy

Objective #1: push-button tool to implement functions $f : \mathbb{R} \to \mathbb{R}$

Similar to yesterday's talk by D. Thomas, but in software

- automatic argument reduction
- automatic polynomial approximation
- automatic domain splitting

with user specified accuracy

Philosophy

Objective #1: push-button tool to implement functions $f : \mathbb{R} \to \mathbb{R}$

Similar to yesterday's talk by D. Thomas, but in software

- automatic argument reduction
- automatic polynomial approximation
- automatic domain splitting

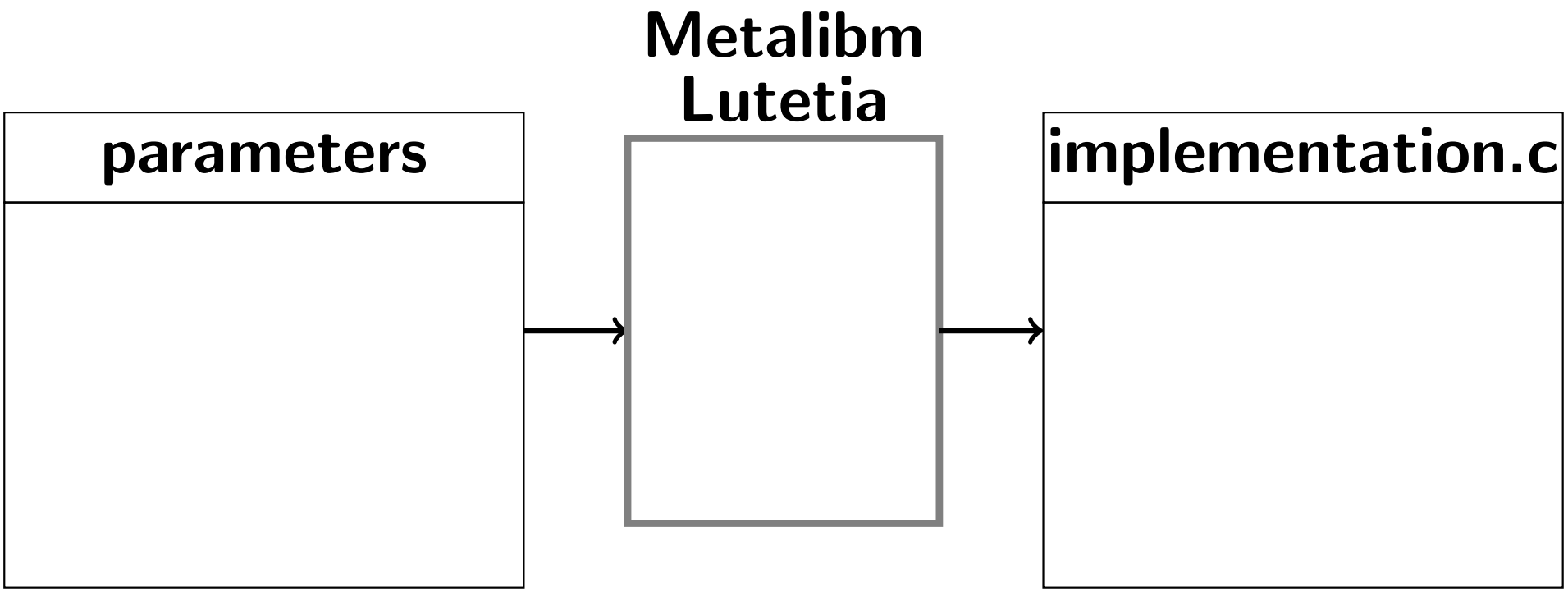
with user specified accuracy

Objective #2: black-box functions

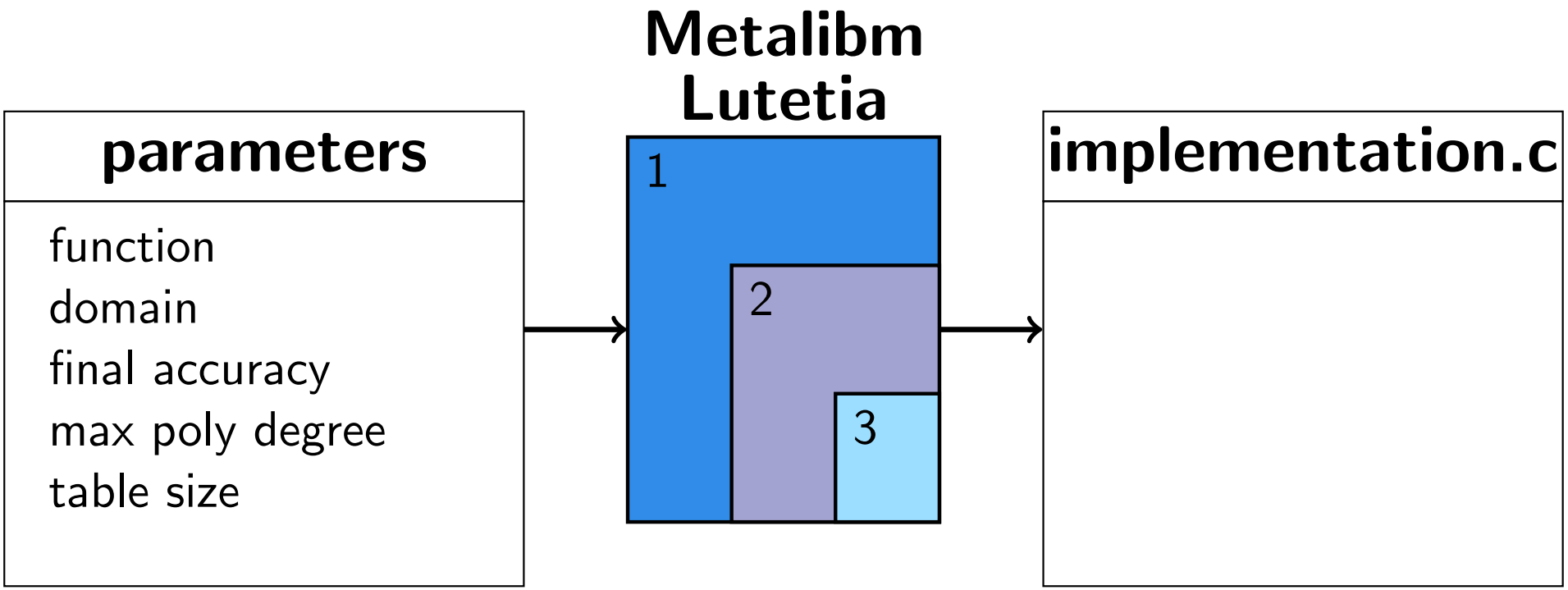
Open-ended means: no function dictionaries

- specify the function by an expression (composite functions)
- but not only:
  - all we need is code that evaluates the function and its first derivatives with arbitrary accuracy

Black-box function generator



Black-box function generator



1 - Properties detection

2 - Domain splitting

3 - Approximation

Exponential function detection

Generation hypothesis

$f(x)$ is of type β^x, unknown β

Exponential function detection

Generation hypothesis

$f(x)$ is of type β^x, unknown β

Finding the base

$\beta = \exp\left(\frac{\ln(f(\xi))}{\xi}\right)$, for some $\xi \in [a, b]$

Exponential function detection

Generation hypothesis

$f(x)$ is of type β^x, unknown β

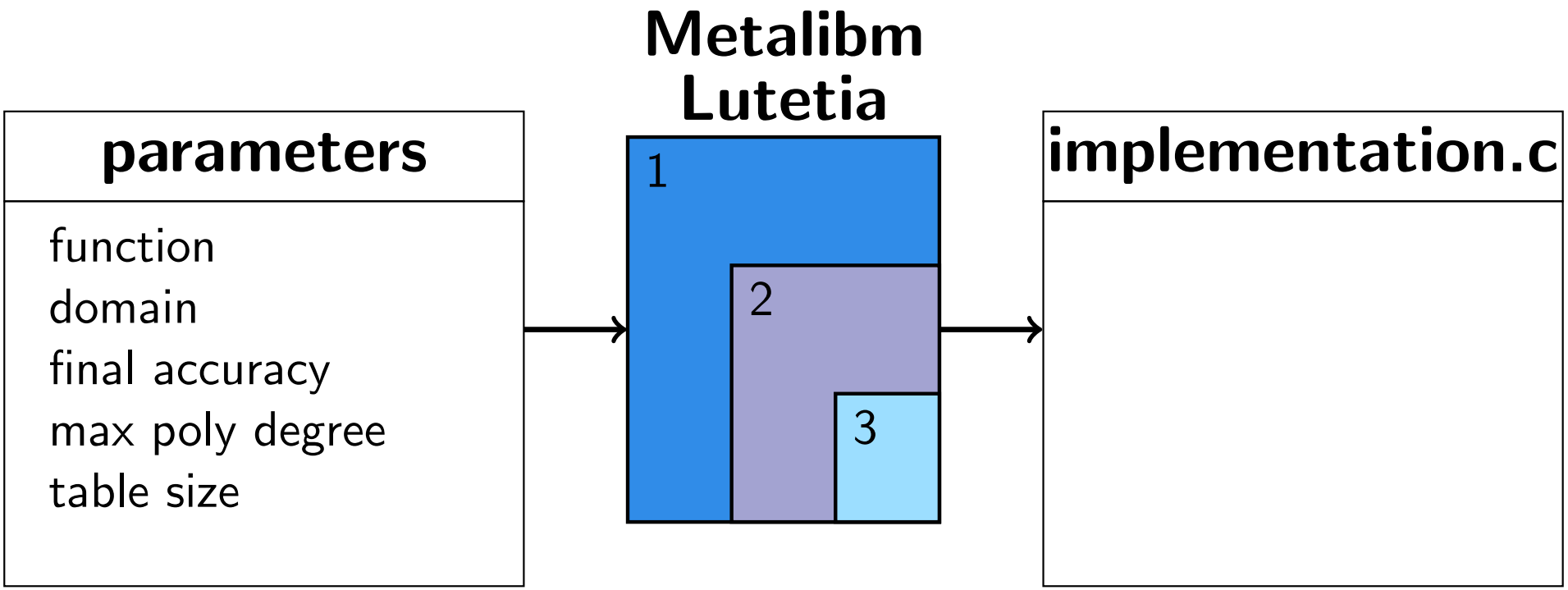
Finding the base

$\beta = \exp\left(\frac{\ln(f(\xi))}{\xi}\right)$, for some $\xi \in [a, b]$

Decision of acceptance

$\tilde{\varepsilon} = \left\| \frac{\beta^x}{f(x)} - 1 \right\|_{\infty}^{[a,b]}$ should be small

Black-box function generator



1 - Properties detection

2 - Domain splitting

3 - Approximation

Domain splitting hints

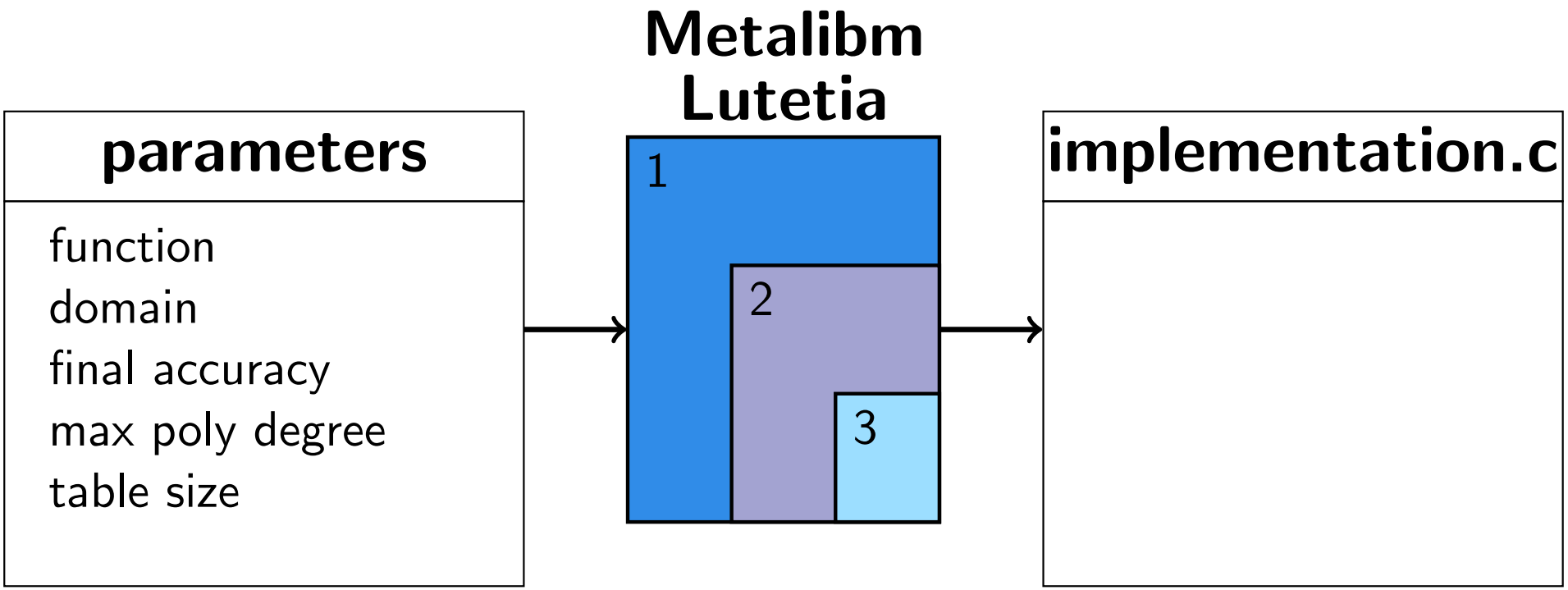
General procedures

- Naive: choose some large k
- Hierarchical: split into 2^k subdomains
- Successive powers of two

Function-adapted procedures

- Bisection
- Optimized bisection

Black-box function generator

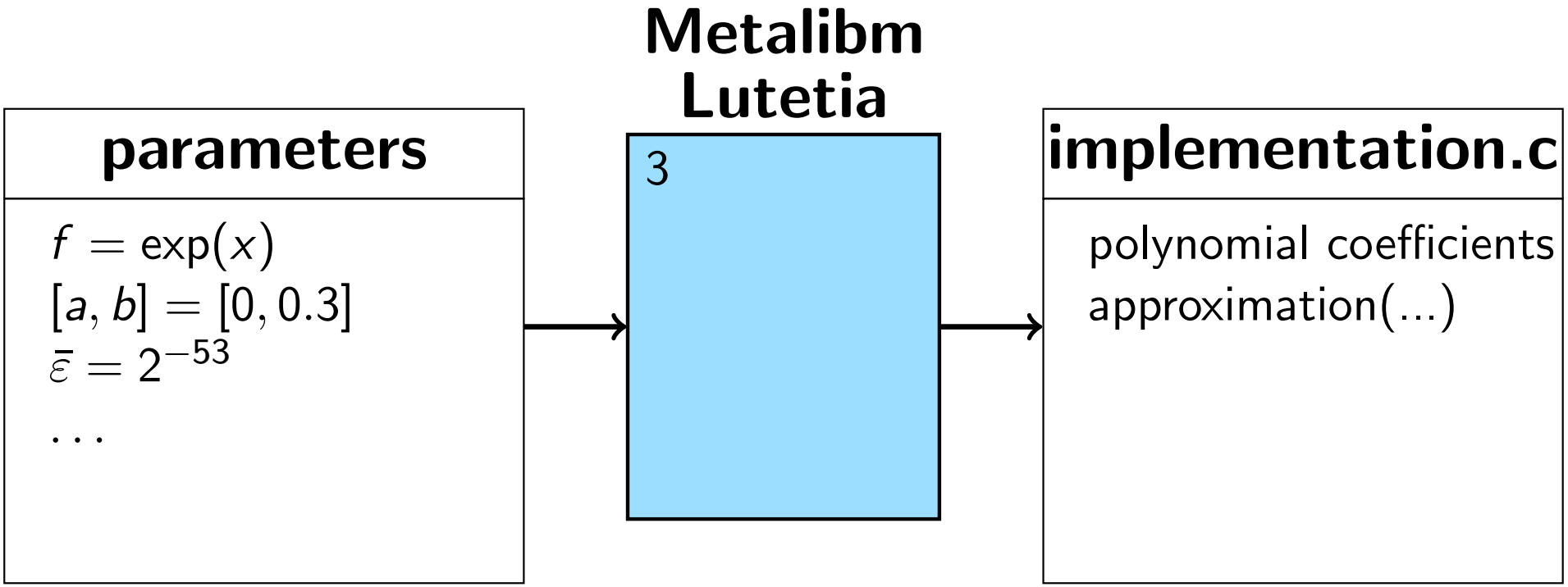


1 - Properties detection

2 - Domain splitting

3 - Approximation

Black-box function generator

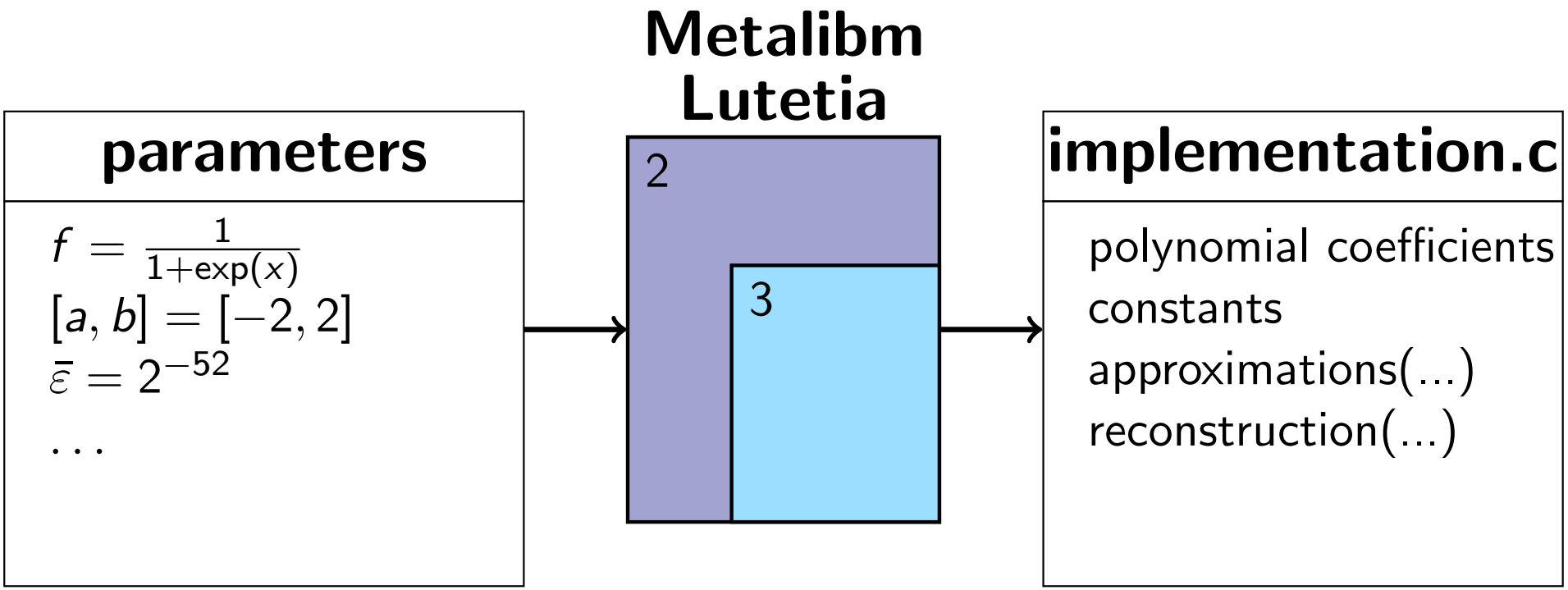


1 - Properties detection

2 - Domain splitting

3 - Approximation

Black-box function generator

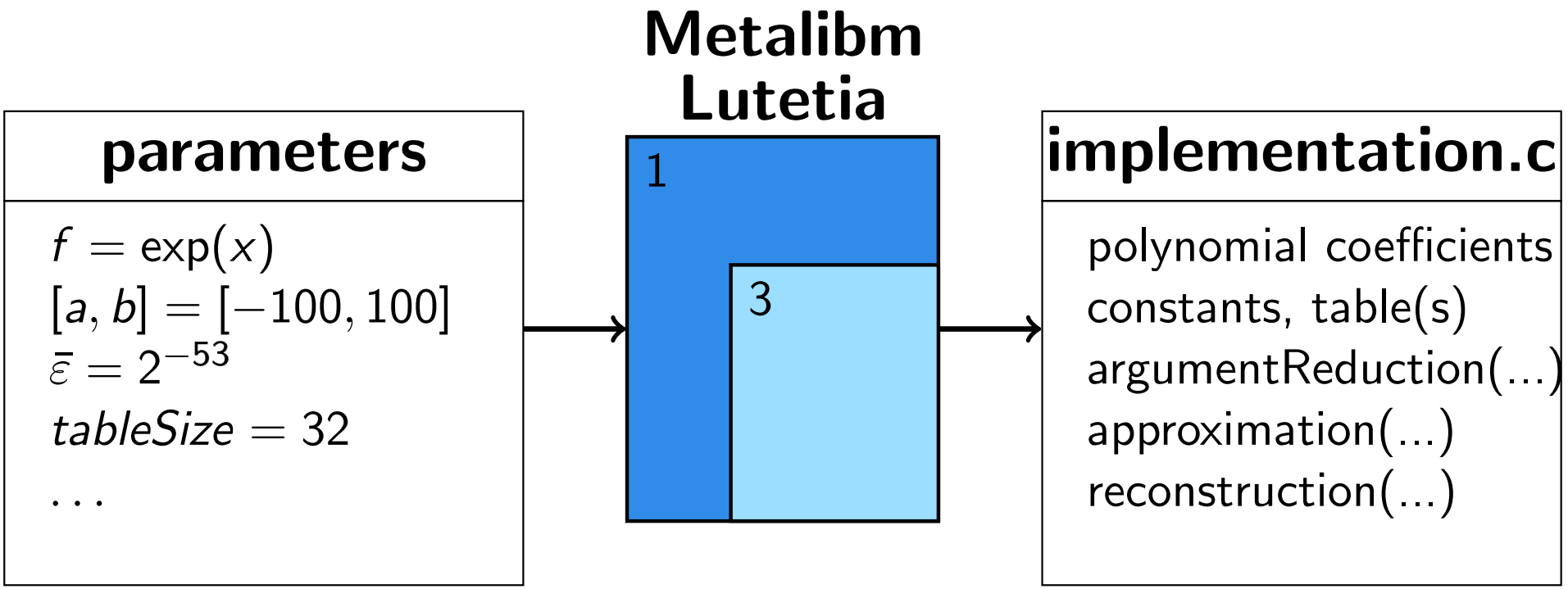


1 - Properties detection

2 - Domain splitting

3 - Approximation

Black-box function generator



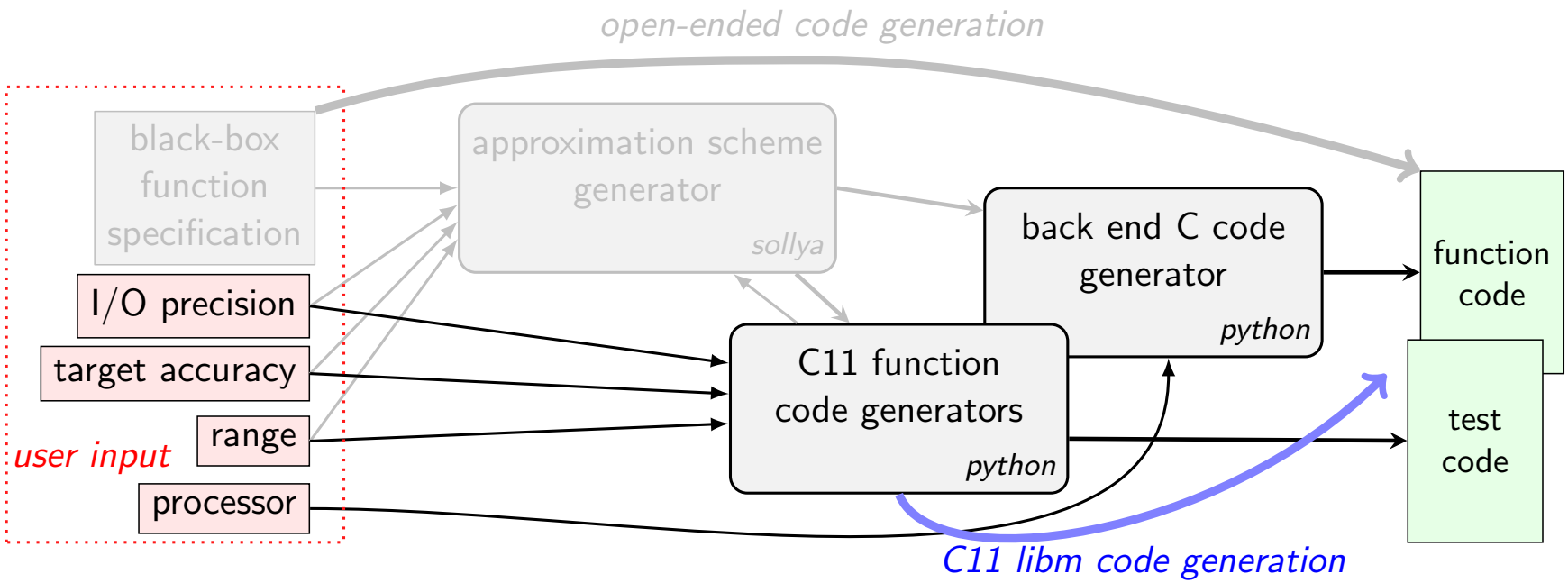
1 - Properties detection

2 - Domain splitting

3 - Approximation

Outline

Metalibm: generator use-cases



Philosophy

Objective #1: enhance the productivity of seasoned libm developer

- capture many code varieties in a single, high-level source
- capture function-specific tricks, floating-point tricks, ...
- enable design space exploration
- obtain better code in less time

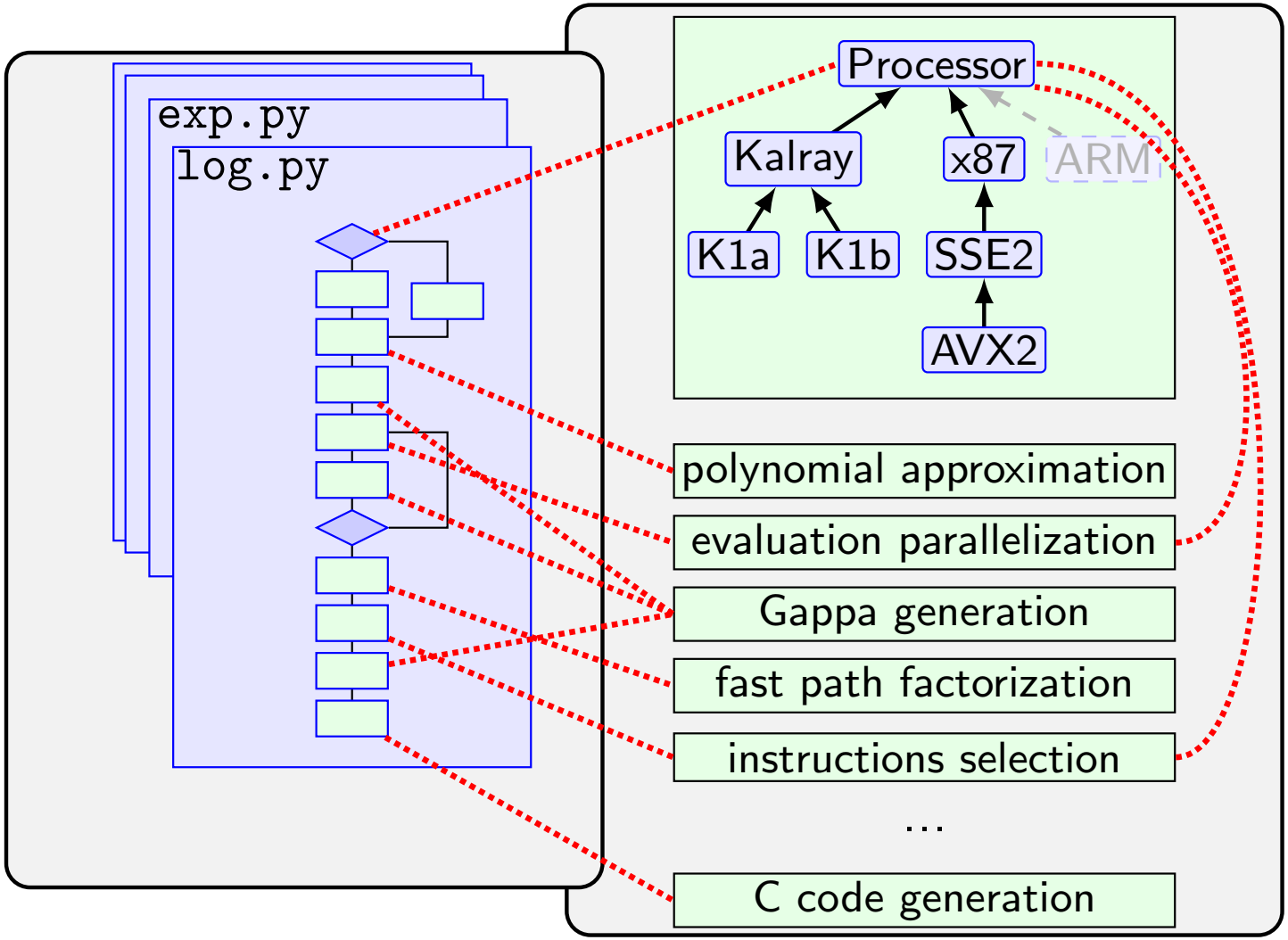
Not a push-button tool like Lutetia!

Objective #2: a back-end for Metalibm-lutetia

The Lutetia people are seasoned libm developers...

- manage processor-specific optimizations
- manage performance options (vector versus latency, ...)

Overview of code generation back-end



Technical choices

Give the libm developer full control over the generated code:

- embed code generation in Python scripts
  - describe the evaluation scheme in Python syntax
  - Sollya embedded in Python
  - use Python scripting for design-space exploration, etc
- a framework that provides all sorts of useful services
  - from high-level (Lutetia-based polynomial approximation)
  - to low-level (code transformations for vectorization)
- describe a Gappa proof also in Python

Speedups obtained with respect to default libm

Table obtained out of `exp.py` and `log.py`.

processor	function	speedup	default libm
Kalray K1a	expf (binary32)	4.0	newlib
	logf (binary32)	2.7	
	exp (binary64)	5.8	
	log (binary64)	5.8	
core i7, SSE2	expf (binary32)	1.7	glibc
	logf (binary32)	1.02	
	exp (binary64)	1.7	
	log (binary64)	1.6	
core i7, AVX2	expf (binary32)	1.1	
	logf (binary32)	0.96	
	exp (binary64)	1.9	
	log (binary64)	1.6	

(all C11-compliant and optimized for latency.)

An example

Python generic code

```
k = NearestInteger(unround_k, precision = self.precision)
```

`NearestInteger` is a method of the generic `Processor` class.

Code generated for binary32 on Kalray

```
k = rintf(unround_k);
```

Code generated for binary64 on X87/SSE2

```
t = _mm_set_sd(unround_k);
t1 = _mm_round_sd(t, t, _MM_FROUND_TO_NEAREST_INT);
k = _mm_cvtsd_f64(t1);
```

Challenge: find the right balance between Metalibm and the compiler.

Outline

Conclusion

- Two tools for libm developers available at http://metalibm.org
  - Automated generation of evaluation schemes
  - Libm development framework
- Reduced cost to get alternative function code
- Comparable or better performance
- Next goal: unify the two approaches
- Addition of argument reduction procedures and new processor classes
- Offline demos in a coffee-break

Q/A

Thank you for your attention!
Questions?

Meanwhile, research on elementary functions goes on

Preliminary results on correctly rounded logarithm
using the 64-bit integer arithmetic of modern processors:

source	**system**	**crlibm**	**crlibm-de**	**fixed-point**
avg time	94	107	65	70
max time	13K	889	573	165