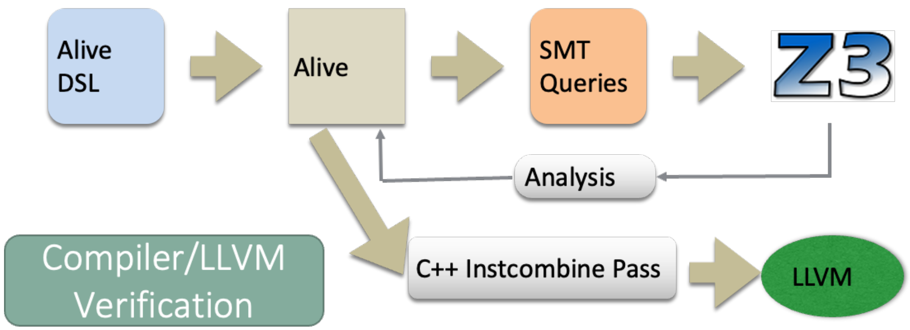
A Case for Correctly Rounded Elementary Functions

Santosh Nagarakatte @ NJPLS 2022
Rutgers University

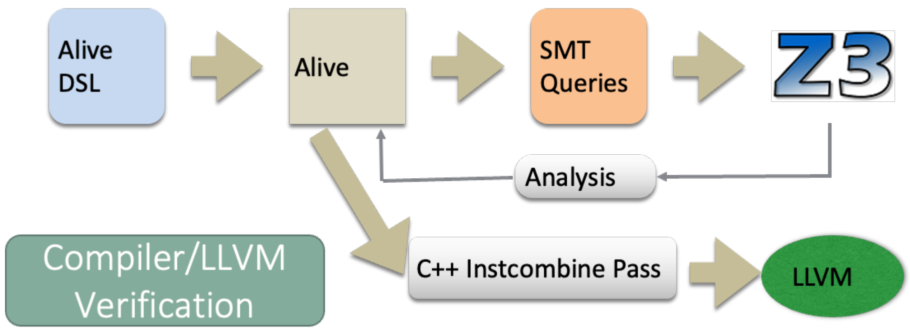
Collaborators: Jay Lim, Mridul Aanjaneya, John Gustafson, and Sehyeok Park

From Compiler Verification to Elementary Functions



POPL 2012, PLDI 2013, PLDI 2015,
SAS 2016, PLDI 2017, CACM-RH 2018

From Compiler Verification to Elementary Functions



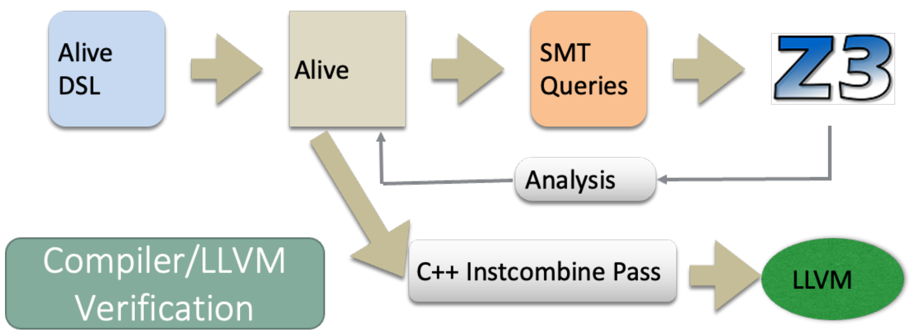
POPL 2012, PLDI 2013, PLDI 2015,
SAS 2016, PLDI 2017, CACM-RH 2018

Floating point Optimizations are wrong!

SAS 2016

Fast math optimizations

From Compiler Verification to Elementary Functions



POPL 2012, PLDI 2013, PLDI 2015,
SAS 2016, PLDI 2017, CACM-RH 2018

Floating point Optimizations are wrong!

SAS 2016

Fast math optimizations

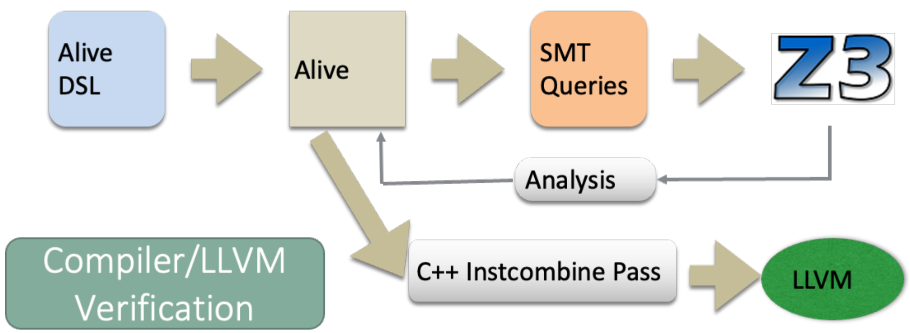
Shadow execution with a high-precision oracle

PLDI 2020,
FSE 2021

$e^x, log_2 x, ...$

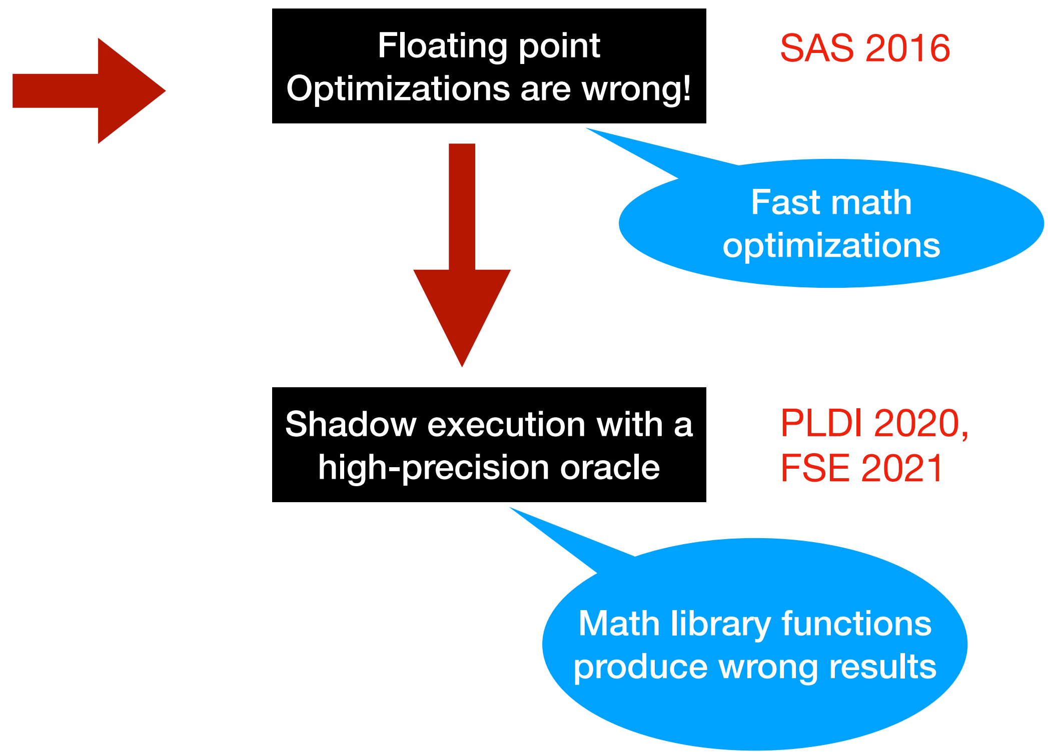
Math library functions produce wrong results

From Compiler Verification to Elementary Functions

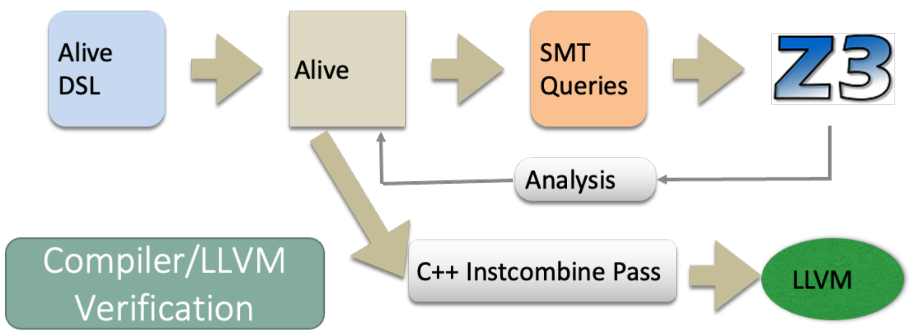


POPL 2012, PLDI 2013, PLDI 2015,
SAS 2016, PLDI 2017, CACM-RH 2018

Who cares?



From Compiler Verification to Elementary Functions



POPL 2012, PLDI 2013, PLDI 2015,
SAS 2016, PLDI 2017, CACM-RH 2018

Floating point Optimizations are wrong!

SAS 2016

Fast math optimizations

Shadow execution with a high-precision oracle

PLDI 2020, FSE 2021

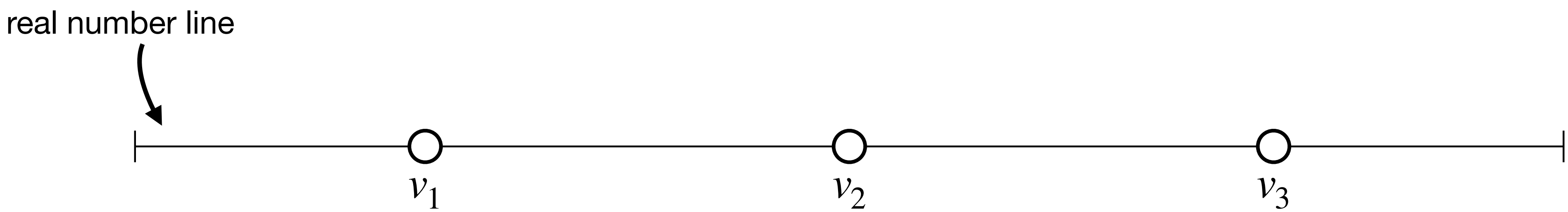
Math library functions produce wrong results

Who cares?

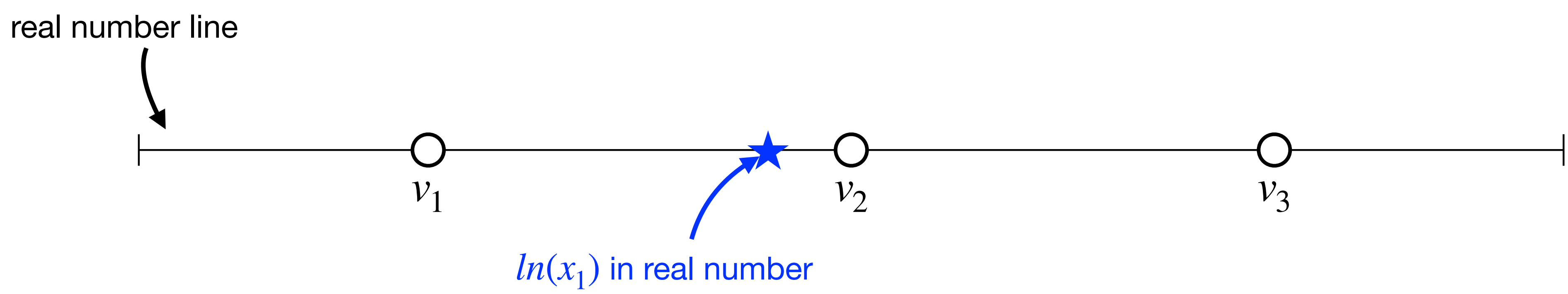
Exact same program on different machines => not reproducible executions!

Program crashes, wrong application results!

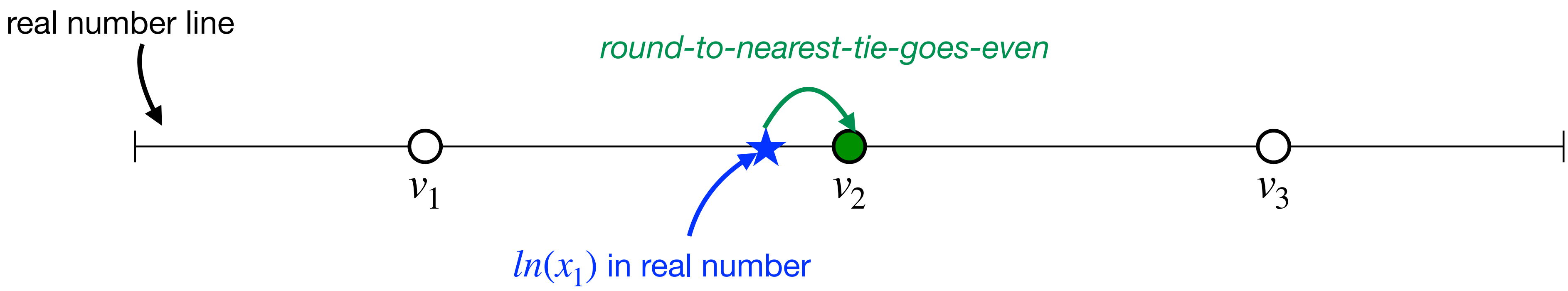
What is a Correctly Rounded Result?



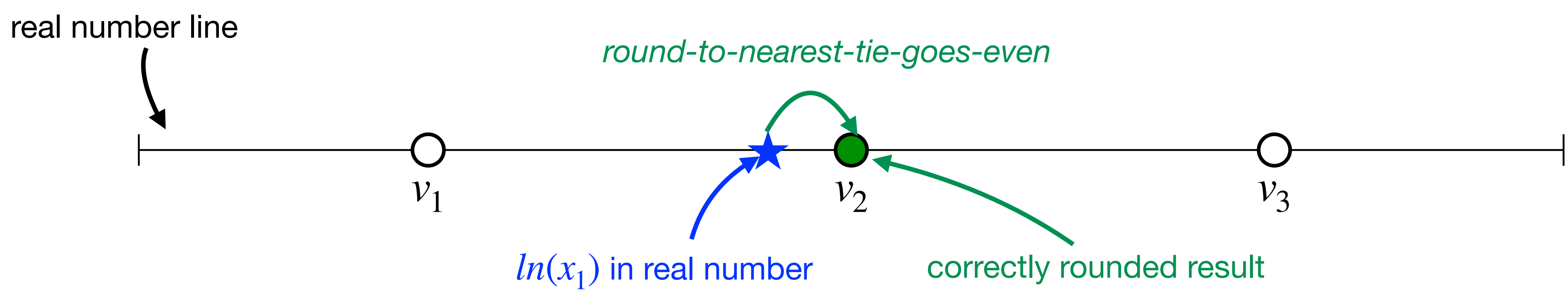
What is a Correctly Rounded Result?



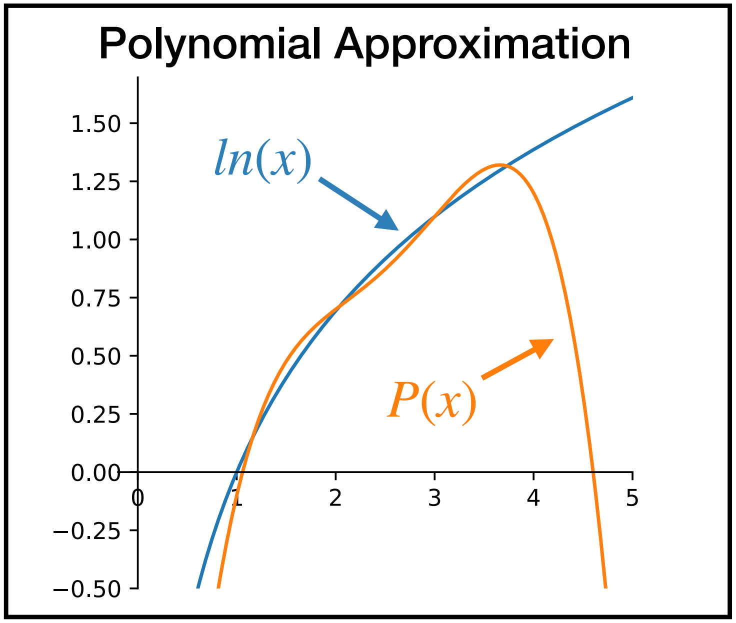
What is a Correctly Rounded Result?



What is a Correctly Rounded Result?



How do Prior Techniques Approximate Elementary Functions?



1. Approximate the REAL value of ln(x)

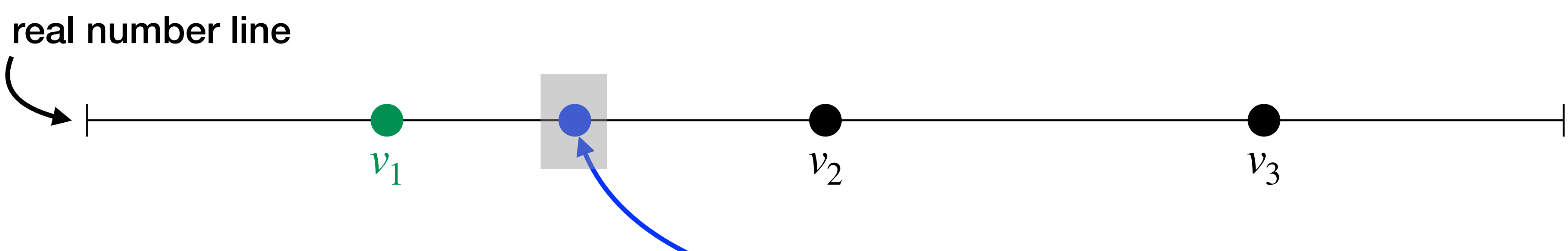
2. Feasible with small domains:

Range reduction to transform the input to a small domain

3. Mini-Max Approximation:

Polynomial Approximation that minimizes the maximum error for all points

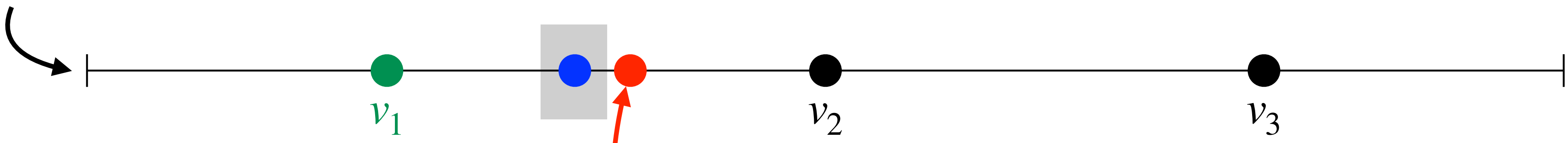
What’s the issue with Mini-Max Approximations?



The real value of $ln(x_1)$ can be extremely close to the decision boundary

Need to approximate $f(x)$ extremely accurately

What’s the issue with Mini-Max Approximations?

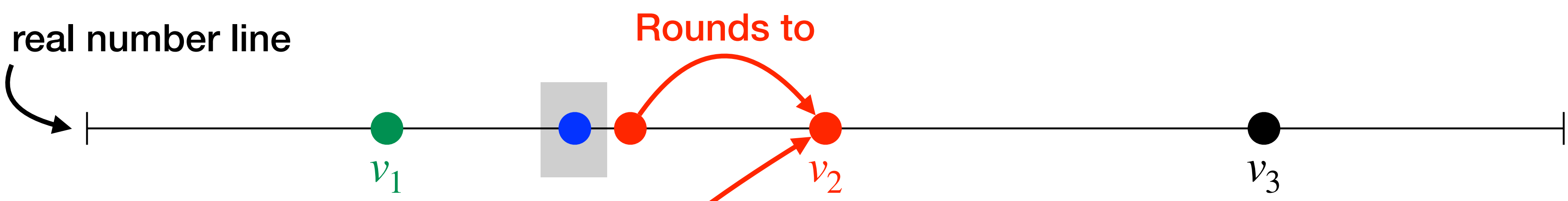


The real value of $ln(x_1)$ can be extremely close to the decision boundary

Need to approximate $f(x)$ extremely accurately

Any approximation error or a numerical error can change the rounding decision

What’s the issue with Mini-Max Approximations?



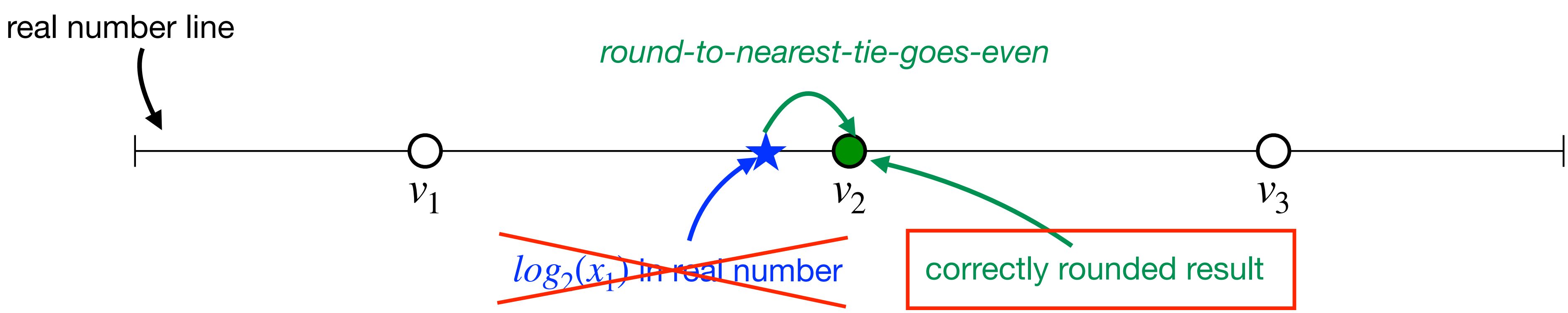
The real value of $ln(x_1)$ can be extremely close to the decision boundary

Need to approximate $f(x)$ extremely accurately

Any approximation error or a numerical error can change the rounding decision

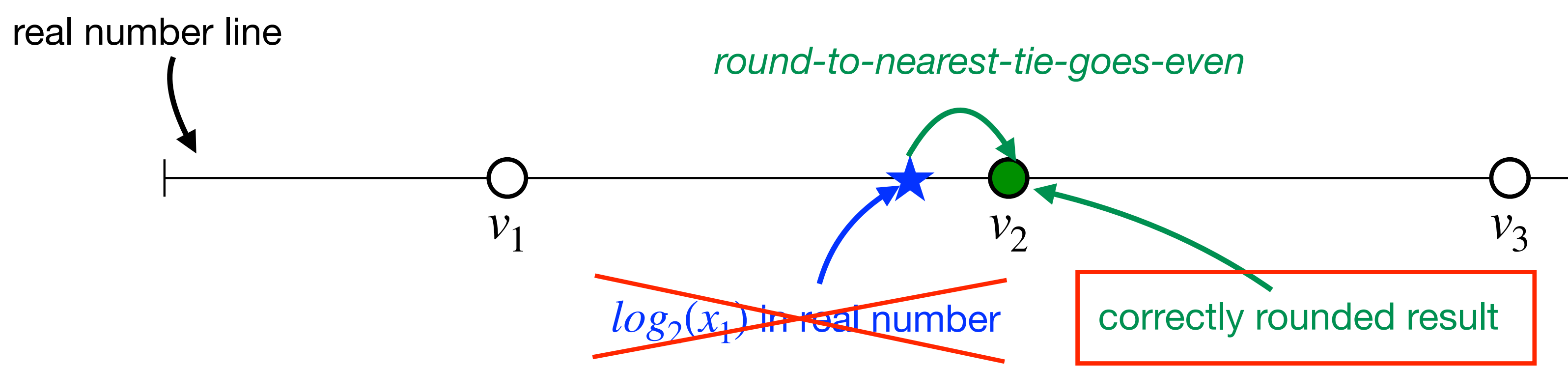
Wrong result due to Incorrect rounding!

Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]



Our RLIBM project makes a case for approximating the correctly rounded result

Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]



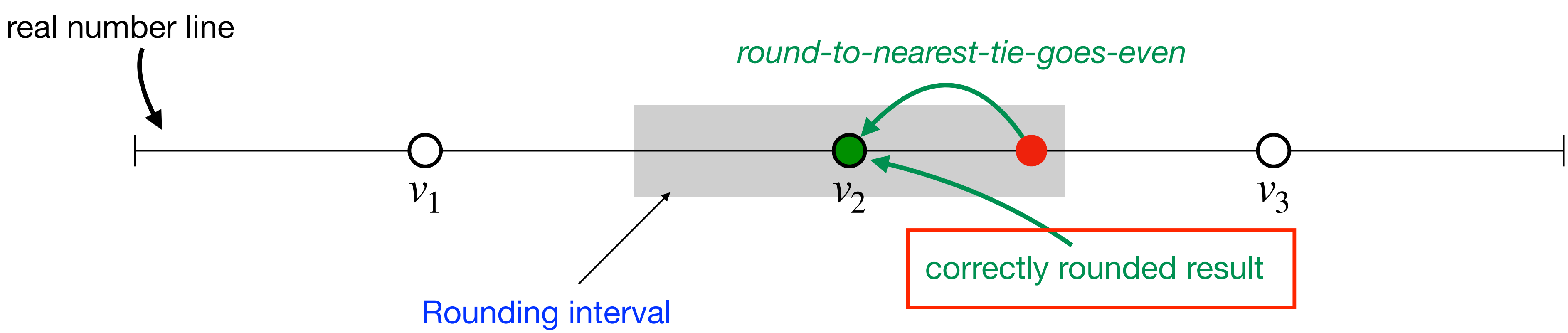
Our RLIBM project makes a case for approximating the correctly rounded result

Focus of the RLIBM project

What is the oracle correctly rounded result?

How to build an efficient implementation that produces the oracle correctly rounded result?

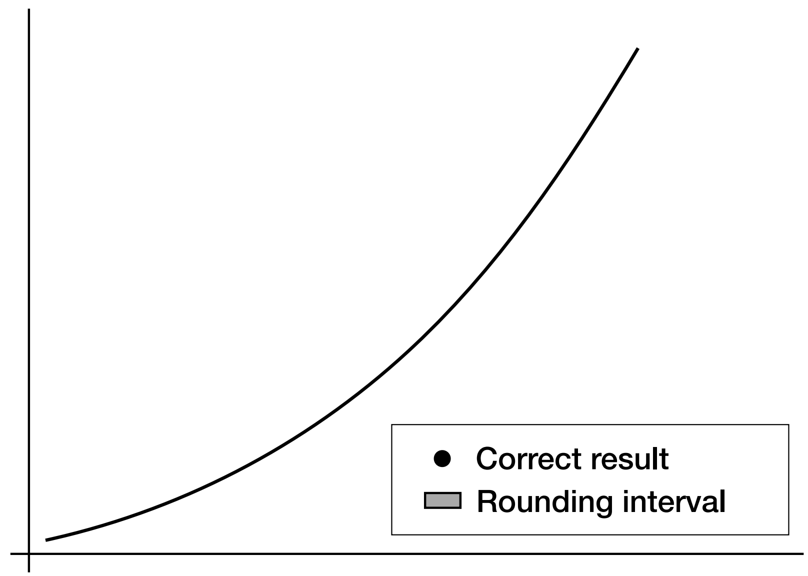
Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]



Our RLIBM project makes a case for approximating the correctly rounded result

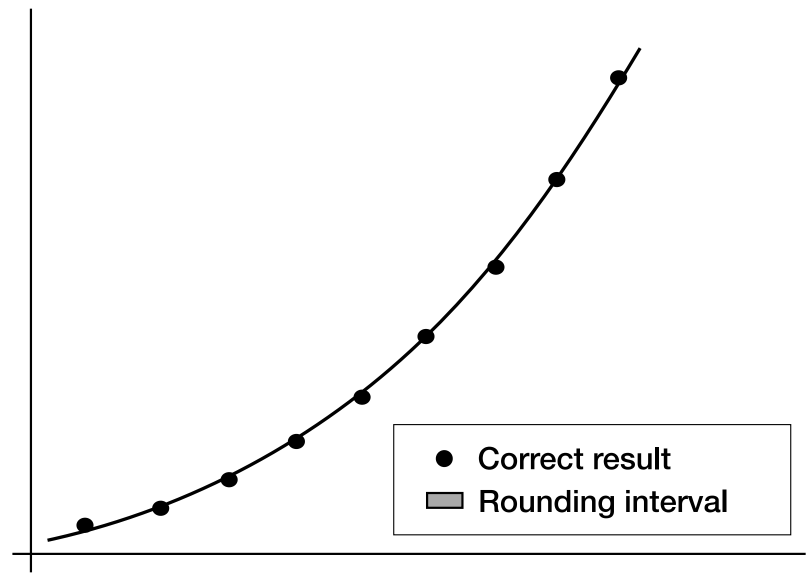
Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

- Given $f(x)$, a representation, and a rounding mode



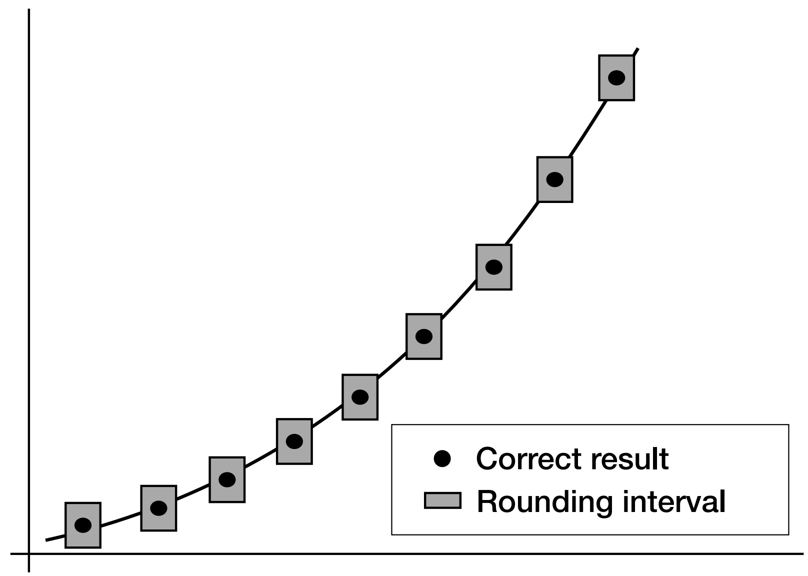
Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

- Given $f(x)$, a representation, and a rounding mode
  1. Compute the correctly rounded result of $f(x)$



Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

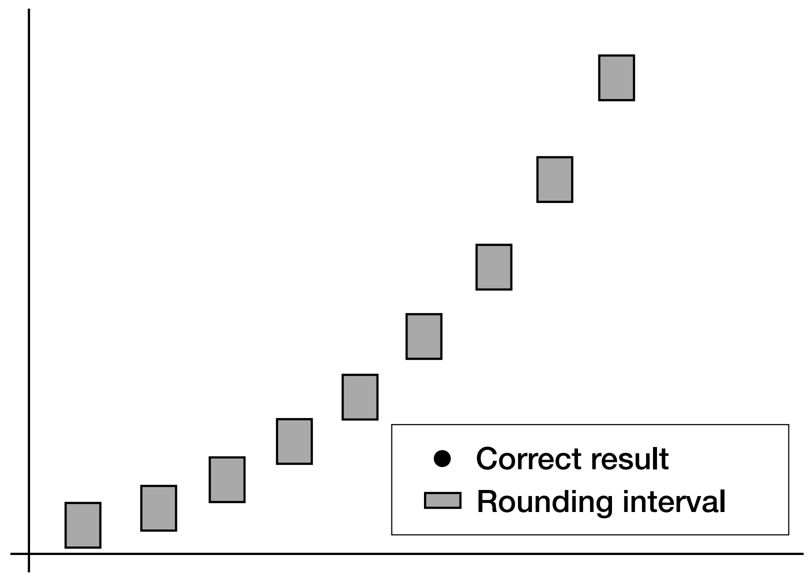
- Given $f(x)$, a representation, and a rounding mode
  1. Compute the correctly rounded result of $f(x)$
  2. Identify rounding interval for each input



Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

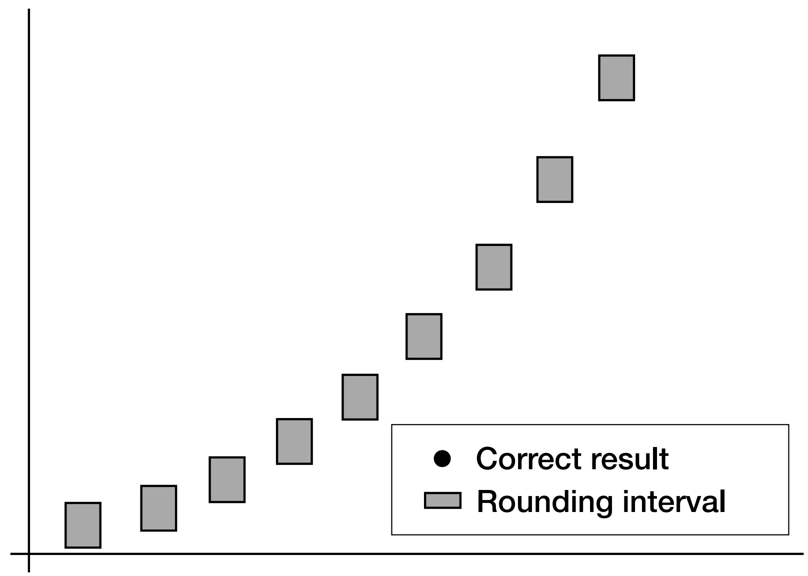
- Given $f(x)$, a representation, and a rounding mode
  1. Compute the correctly rounded result of $f(x)$
  2. Identify rounding interval for each input
     - A linear constraint on the output of the polynomial

$$
\begin{gathered}
l_1 \leq P(x_1) \leq h_1 \\
l_2 \leq P(x_2) \leq h_2 \\
l_3 \leq P(x_3) \leq h_3 \\
l_4 \leq P(x_4) \leq h_4 \\
\dots
\end{gathered}
$$



Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

- Given $f(x)$, a representation, and a rounding mode
  1. Compute the correctly rounded result of $f(x)$
  2. Identify rounding interval for each input
     - A linear constraint on the output of the polynomial
  3. Encode constraint into system of linear inequalities



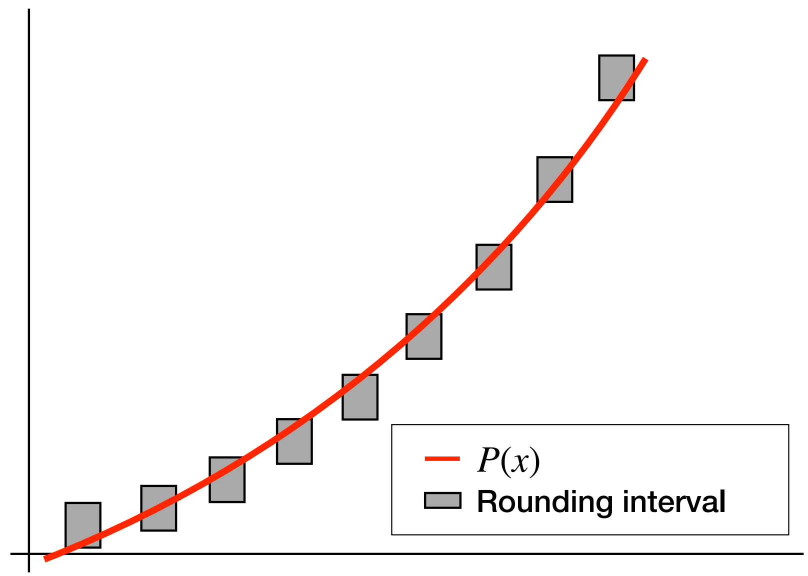
$$\begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \\ \dots \end{bmatrix} \leq \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ 1 & x_3 & x_3^2 & \dots & x_3^d \\ 1 & x_4 & x_4^2 & \dots & x_4^d \\ \dots \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \dots \\ c_d \end{bmatrix} \leq \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ \dots \end{bmatrix}$$

Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

- Given $f(x)$, a representation, and a rounding mode

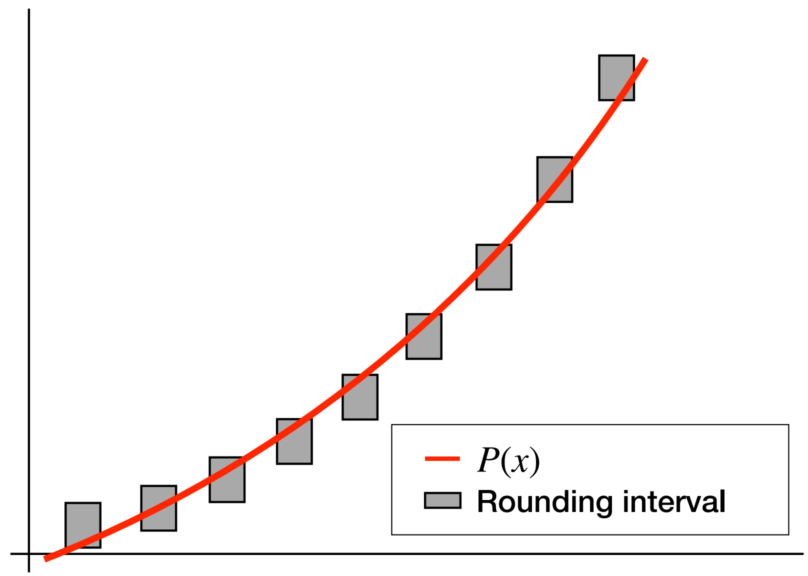
  1. Compute the correctly rounded result of $f(x)$
  2. Identify rounding interval for each input
     - A linear constraint on the output of the polynomial
  3. Encode constraint into system of linear inequalities
  4. Use a Linear Programming solver to solve for $P(x)$

$$\begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \\ \dots \end{bmatrix} \leq \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ 1 & x_3 & x_3^2 & \dots & x_3^d \\ 1 & x_4 & x_4^2 & \dots & x_4^d \\ \dots \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \dots \\ c_d \end{bmatrix} \leq \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ \dots \end{bmatrix}$$



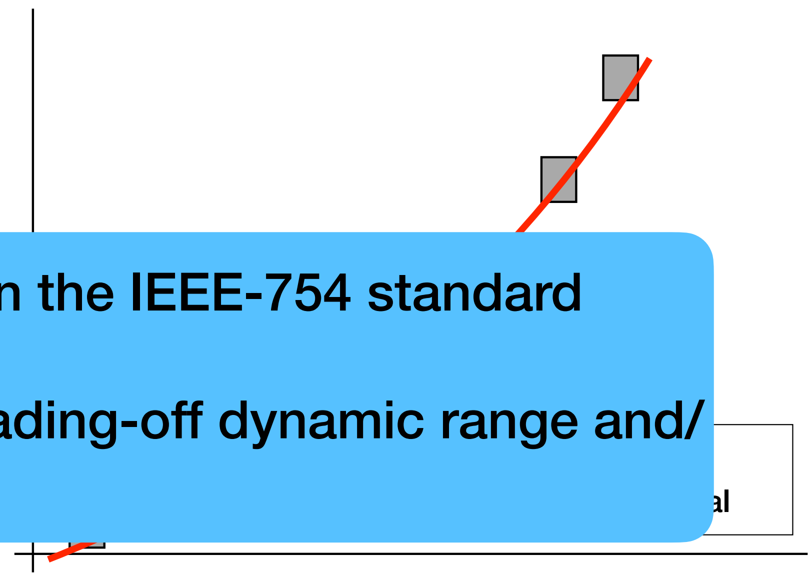
Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

- Given $f(x)$, a representation, and a rounding mode
    1. Compute the correctly rounded result of $f(x)$
    2. Identify rounding interval for each input
        - A linear constraint on the output of the polynomial
    3. Encode constraint into system of linear inequalities
    4. Use Linear Programming solver to solve for $P(x)$
- Resulting polynomial produces correct results for the chosen representation and rounding mode



Our RLIBM Project [POPL 2021, PLDI 2021, POPL 2022, PLDI 2022]

- Given $f(x)$, a representation, and a rounding mode
  1. Compute the correctly rounded result of $f(x)$
  2. Identify rounding interval for each input
     -
  3.
  4.
- Resulting polynomial produces correct results for the chosen representation and rounding mode



There are 5 different rounding modes in the IEEE-754 standard

Everyone designing new representations trading-off dynamic range and/or precision

How do we produce a single polynomial approximation that produces correctly rounded results for multiple representations and rounding modes?

A Naive Solution

Let's say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

A Naive Solution

Let's say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

$\boxed{log_2(x) \text{ for } \text{64-bit double type}}$

A Naive Solution

Let's say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

input
x
in float

$\boxed{log_2(x) \text{ for 64-bit double type}}$

A Naive Solution

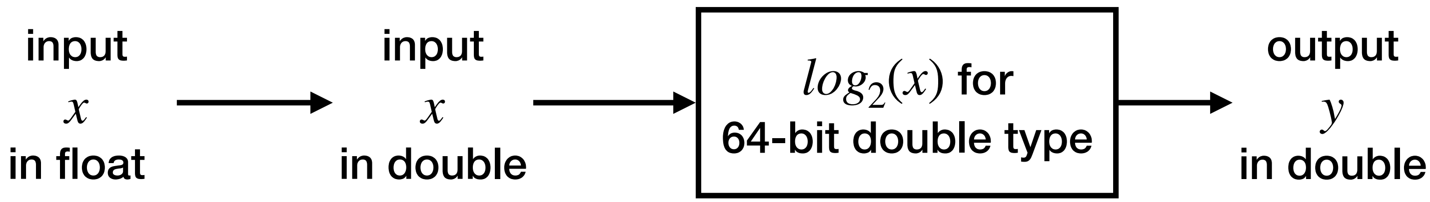
Let's say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

input x in float $\longrightarrow$ input x in double

$log_2(x)$ for 64-bit double type

A Naive Solution

Let's say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float



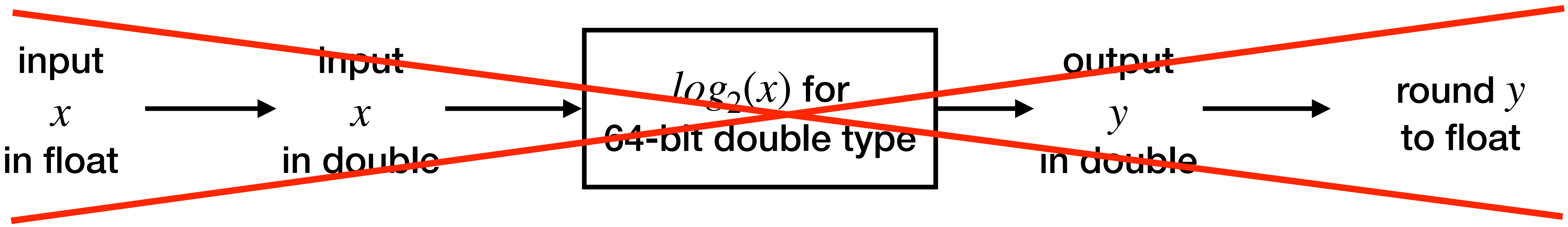
A Naive Solution

Let’s say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

input x in float → input x in double → $\boxed{log_2(x) \text{ for 64-bit double type}}$ → output y in double → round y to float

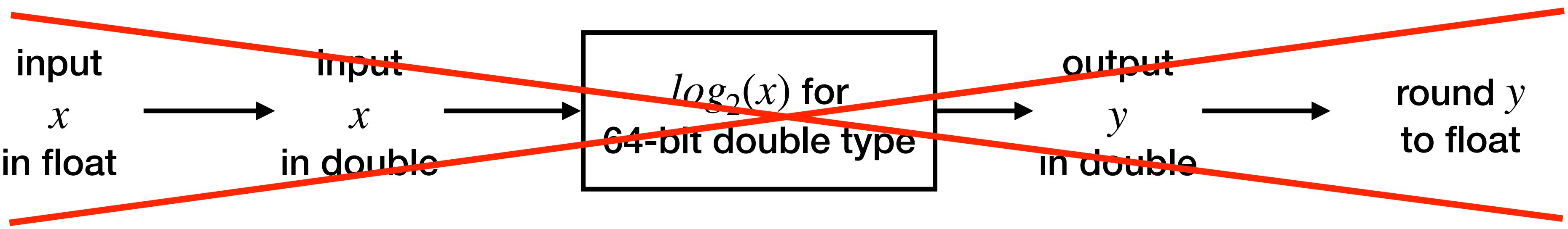
A Naive Solution

Let’s say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float



A Naive Solution

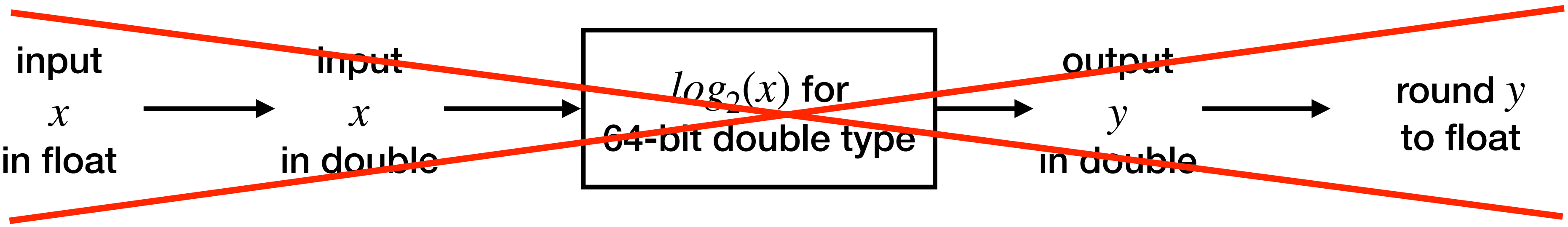
Let’s say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float



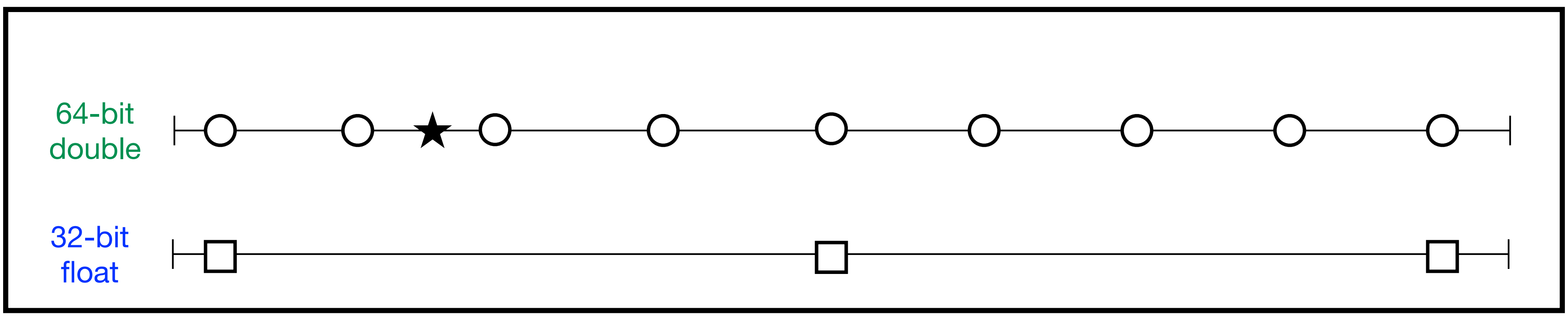
Double rounding error!

Double Rounding Is The Enemy

Let's say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

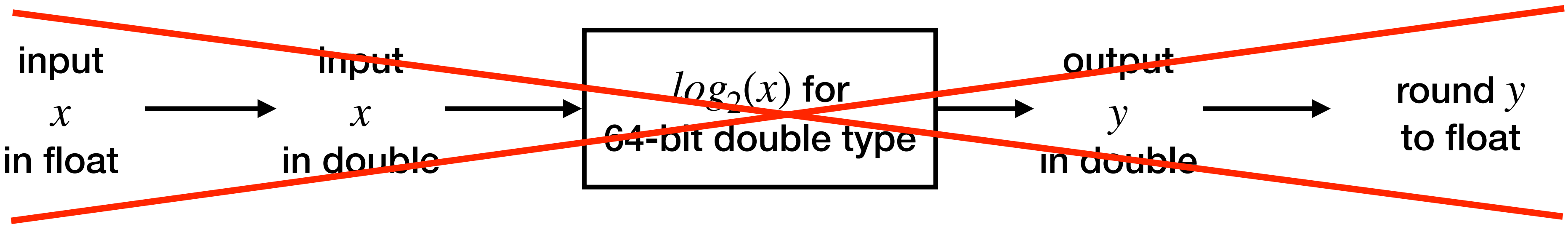


Double rounding error!

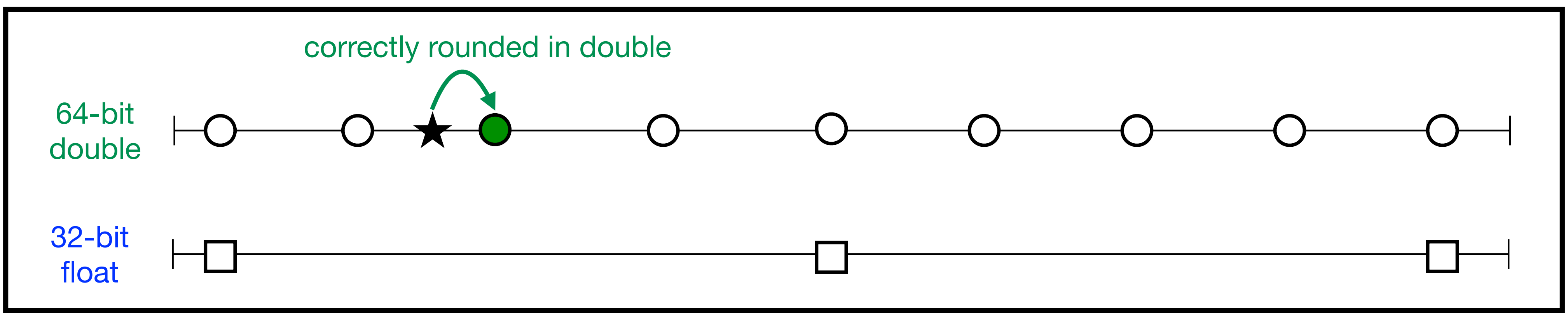


Double Rounding Is The Enemy

Let's say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

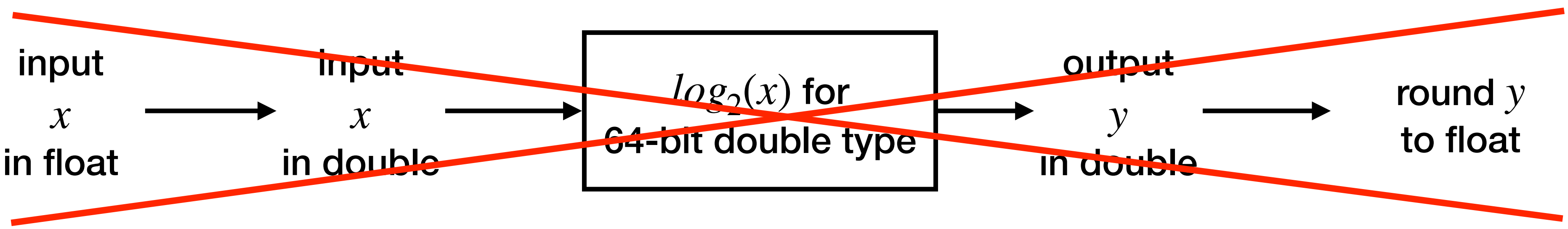


Double rounding error!

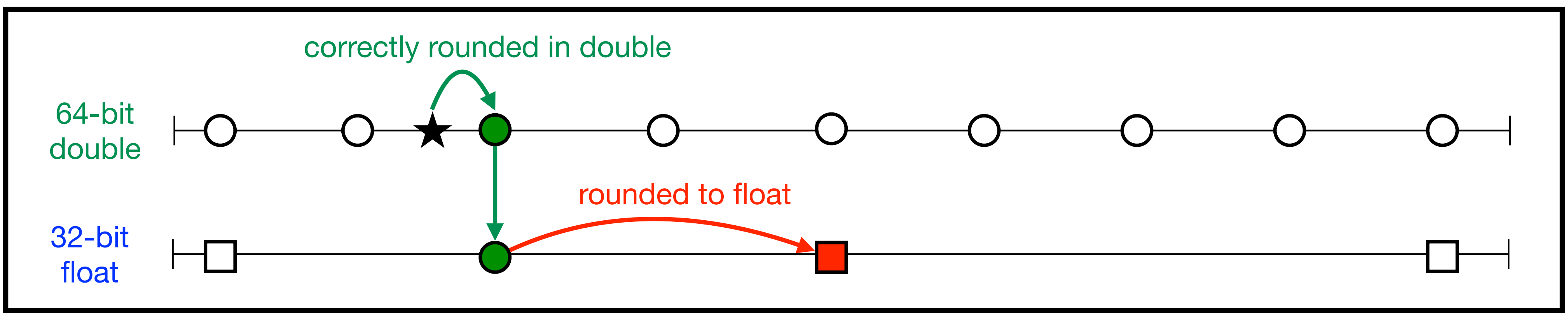


Double Rounding Is The Enemy

Let’s say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float

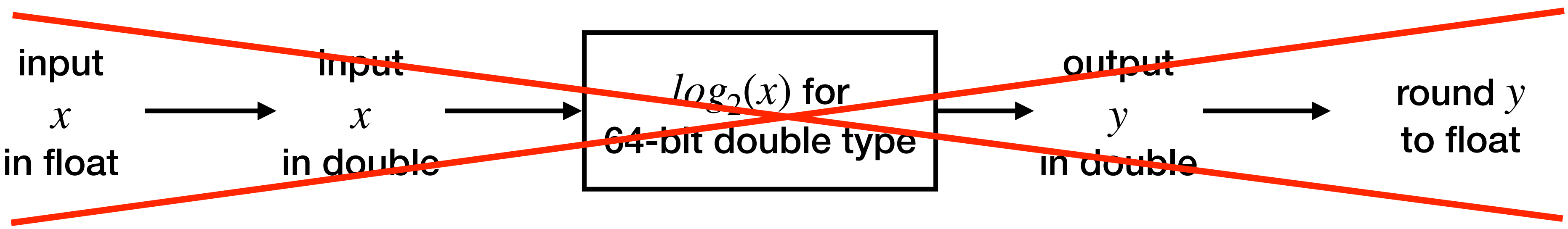


Double rounding error!

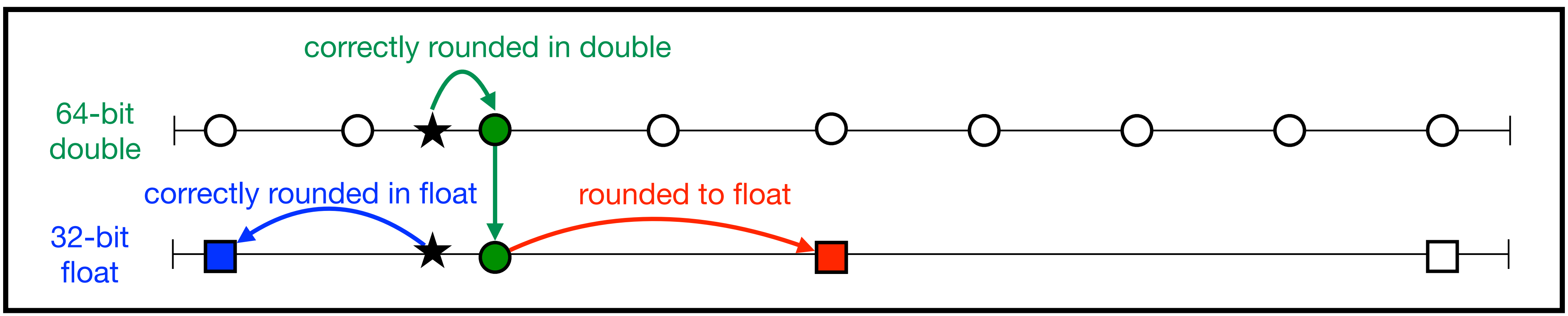


Double Rounding Is The Enemy

Let’s say we want to produce correctly rounded result of $log_2(x)$ for a 32-bit float



Double rounding error!



Our RLIBM Approach for Multiple Representations

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representations
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

Round-to-Odd Rounding Mode

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Round-to-odd:
  - Used for rounding from a decimal to a binary fraction *[Goldberg 1991]*
  - Used for primitive operations in extended precision *[Boldo et al. 2005]*

Round-to-Odd Rounding Mode

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Round-to-odd:
  - Used for rounding from a decimal to a binary fraction *[Goldberg 1991]*
  - Used for primitive operations in extended precision *[Boldo et al. 2005]*
- How do we make it work for elementary functions?
  - Extremely challenging using prior approaches (e.g., Remez Algorithm)
    - How to perform error analysis when round-to-odd mode is involved?
  - Our RLIBM approach allows straight-forward integration with round-to-odd!

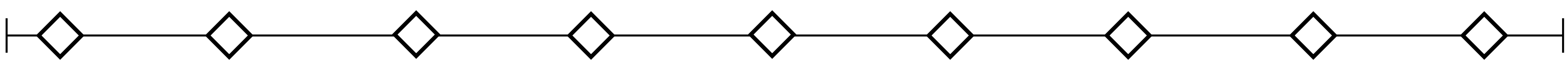
How Does Round-to-Odd Work?

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Round-to-odd:

34-bit FP

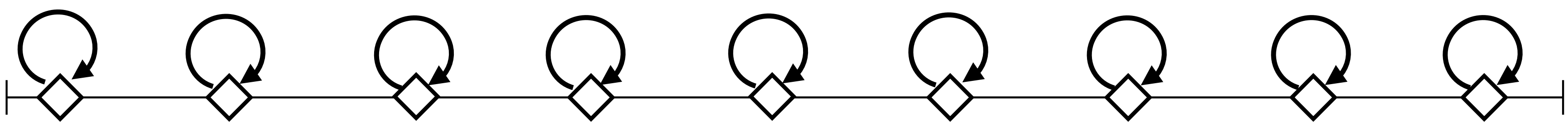


How Does Round-to-Odd Work?

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode
- Round-to-odd:
  - If exactly representable, then it is represented with the value

Generalize: (n + 2) bit floating point representation

34-bit FP

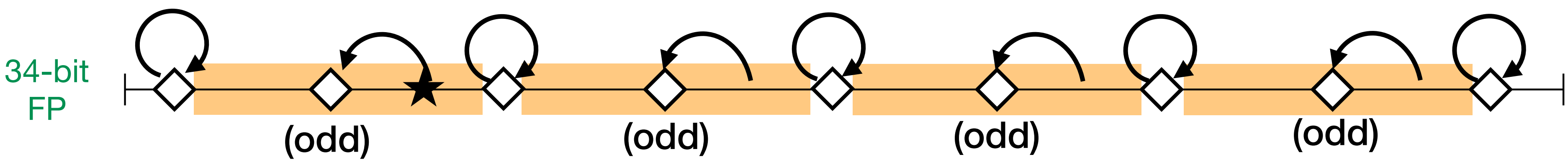


How Does Round-to-Odd Work?

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
    - Generate a polynomial for the 34-bit FP
    - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Round-to-odd:
    - If exactly representable, then it is represented with the value
    - Otherwise, rounds to the adjacent odd value

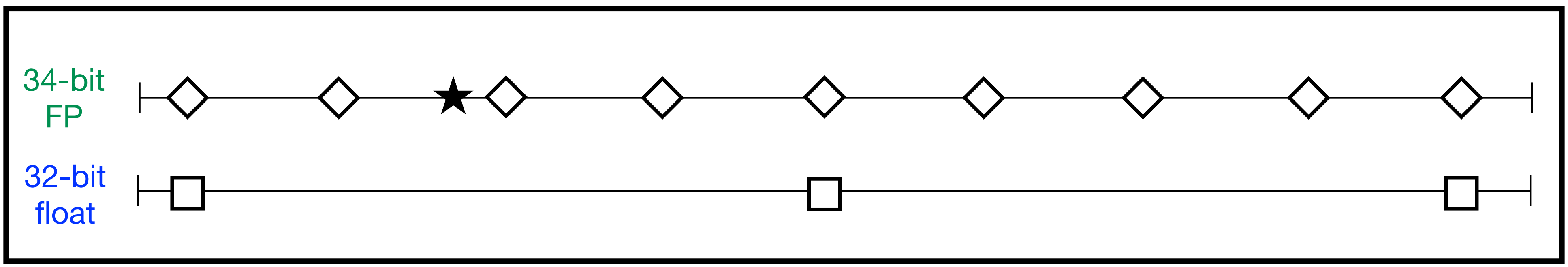


Correctly Rounded Results with Round-to-Odd

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Producing correctly rounded results for a FP type with n-bits or less than n-bit:

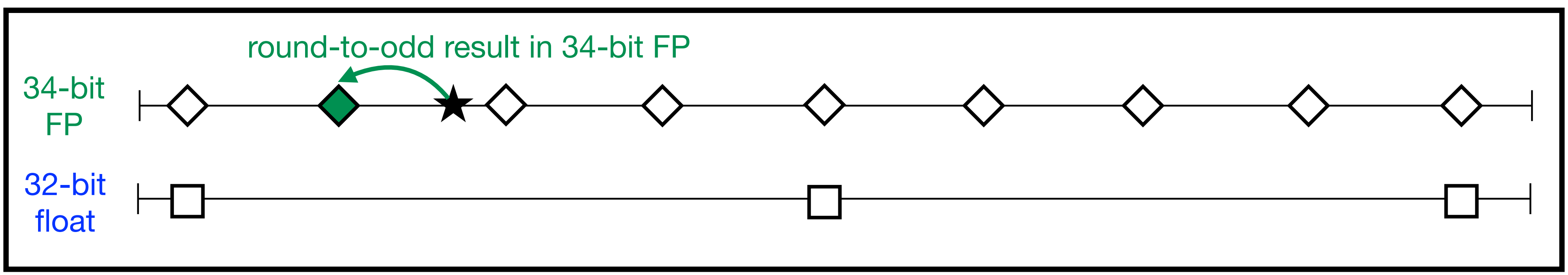


Correctly Rounded Results with Round-to-Odd

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Producing correctly rounded results for a FP type with n-bits or less than n-bit:
  - Produce correctly rounded results for (n+2)-bit FP in the round-to-odd mode

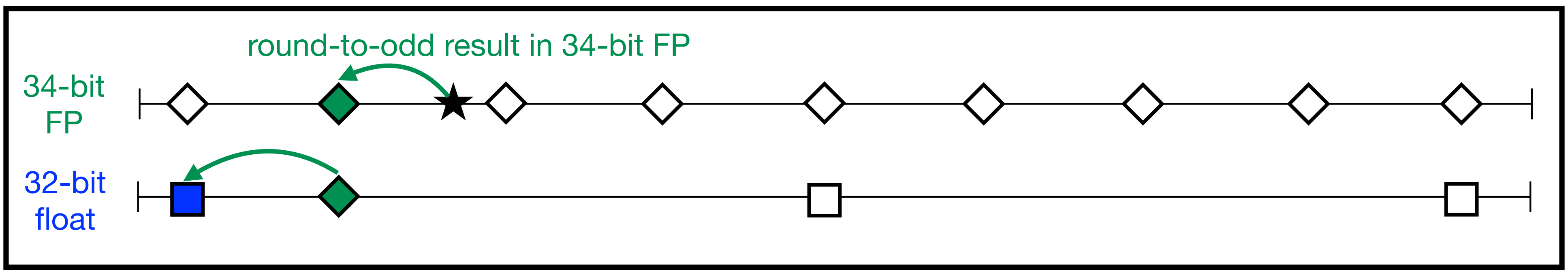


Correctly Rounded Results with Round-to-Odd

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Producing correctly rounded results for a FP type with n-bits or less than n-bit:
  - Produce correctly rounded results for (n+2)-bit FP in the round-to-odd mode
  - Round the result to FP type with n-bits or less than n-bits using any IEEE-754 rounding mode

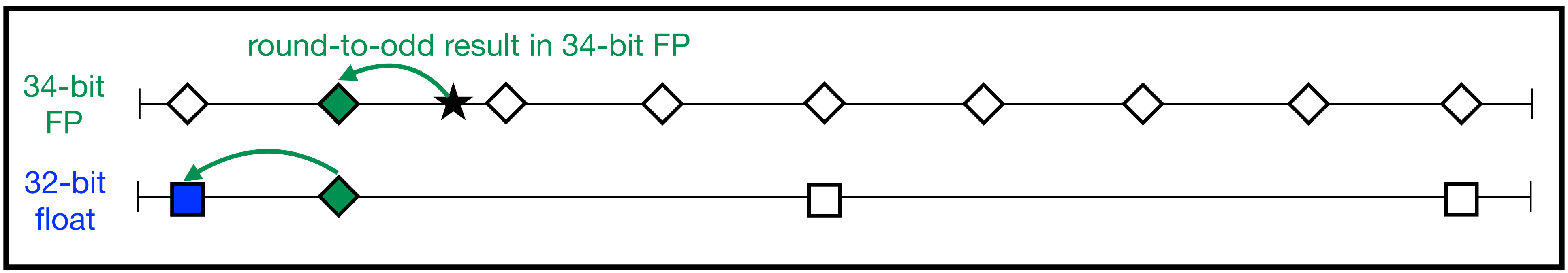


Correctly Rounded Results with Round-to-Odd

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

Generalize: (n + 2) bit floating point representation

- Producing correctly rounded results for a FP type with n-bits or less than n-bit:
  - Produce correctly rounded results for (n+2)-bit FP in the round-to-odd mode
  - Round the result to FP type with n-bits or less than n-bits using any IEEE-754 rounding mode
  - Guaranteed to produce correctly rounded results!

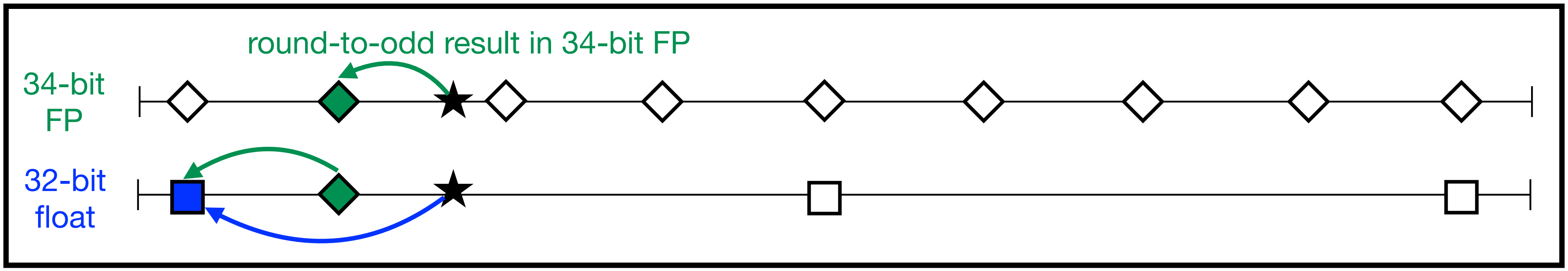


Correctly Rounded Results with Round-to-Odd

- Insight: Retain enough information about the real value even when double rounding
- How to generate 1 polynomial for 10 to 32-bit FP representation:
  - Generate a polynomial for the 34-bit FP
  - Using the round-to-odd rounding mode

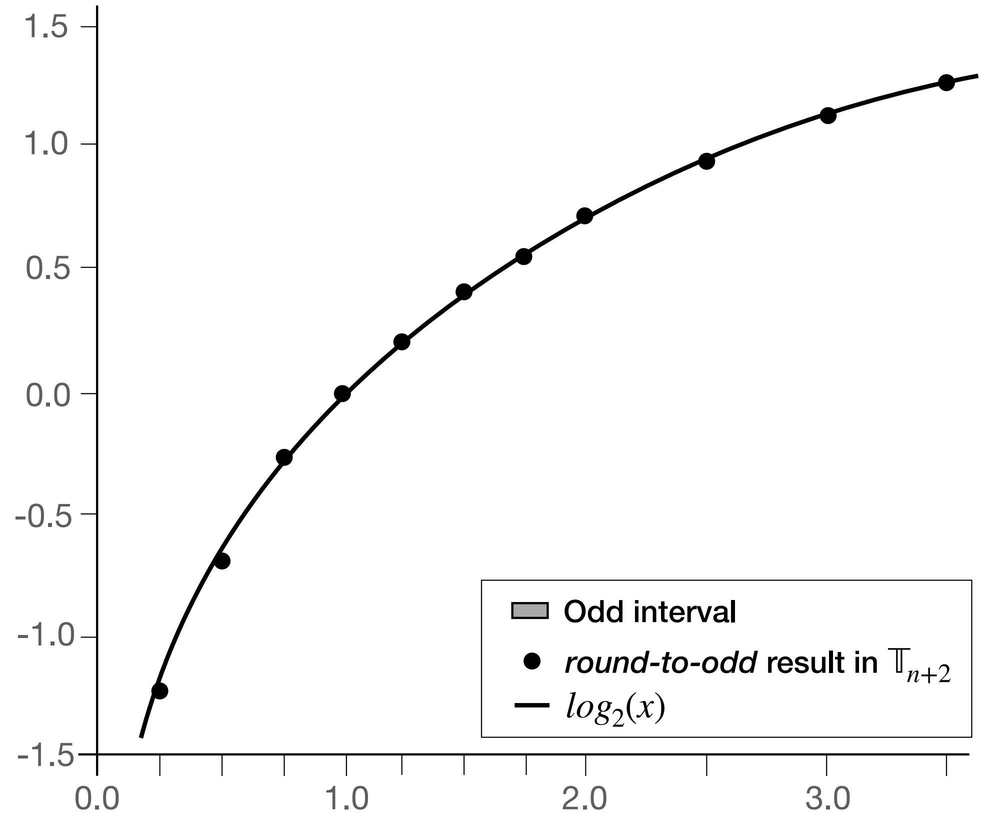
Generalize: (n + 2) bit floating point representation

- Producing correctly rounded results for a FP type with n-bits or less than n-bit:
  - Produce correctly rounded results for (n+2)-bit FP in the round-to-odd mode
  - Round the result to FP type with n-bits or less than n-bits using any IEEE-754 rounding mode
  - Guaranteed to produce correctly rounded results!



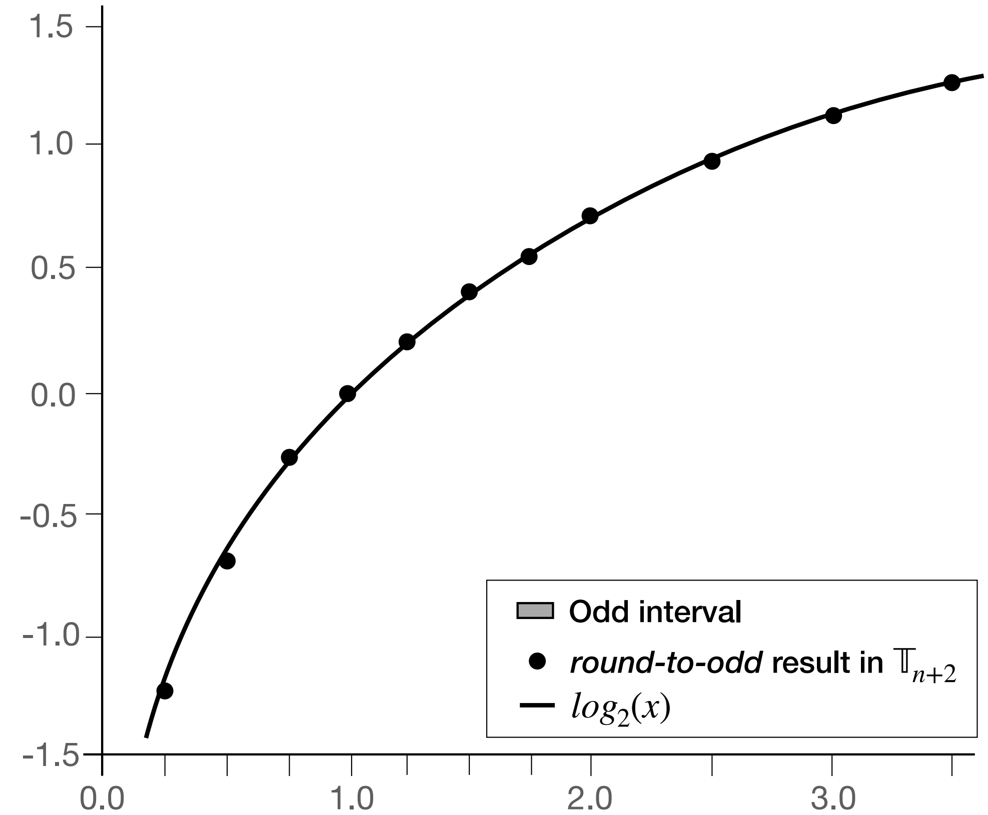
Our RLIBM Approach for Multiple Representations

- For each float input, compute



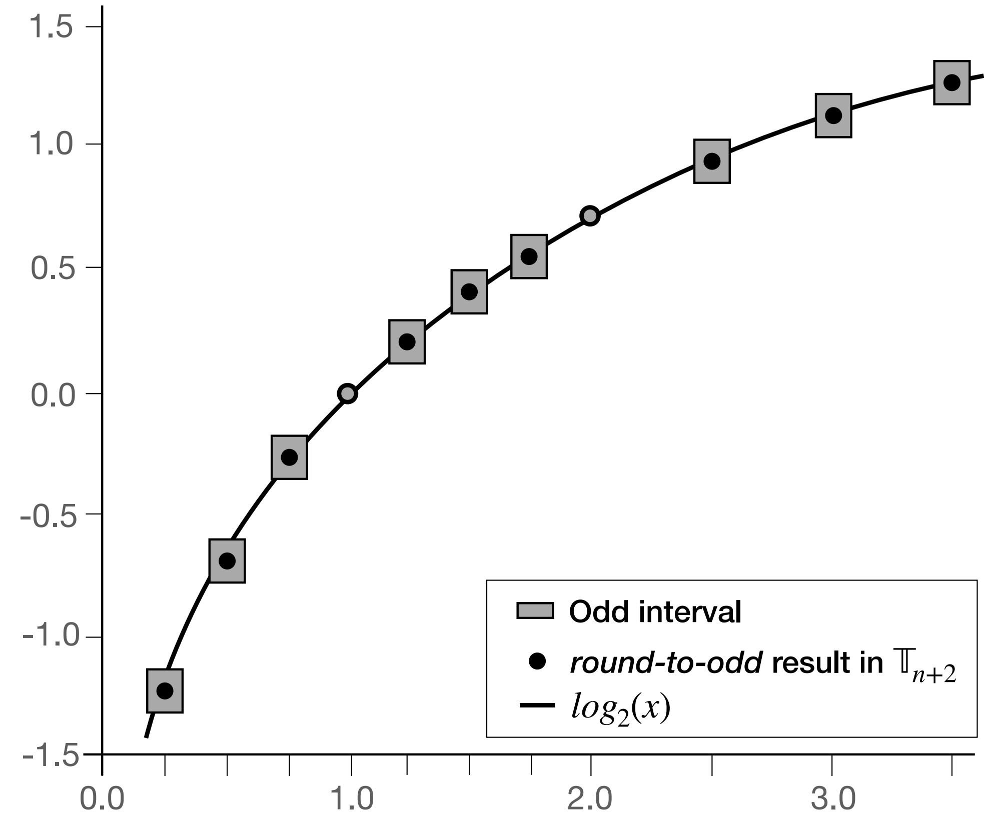
Our RLIBM Approach for Multiple Representations

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34



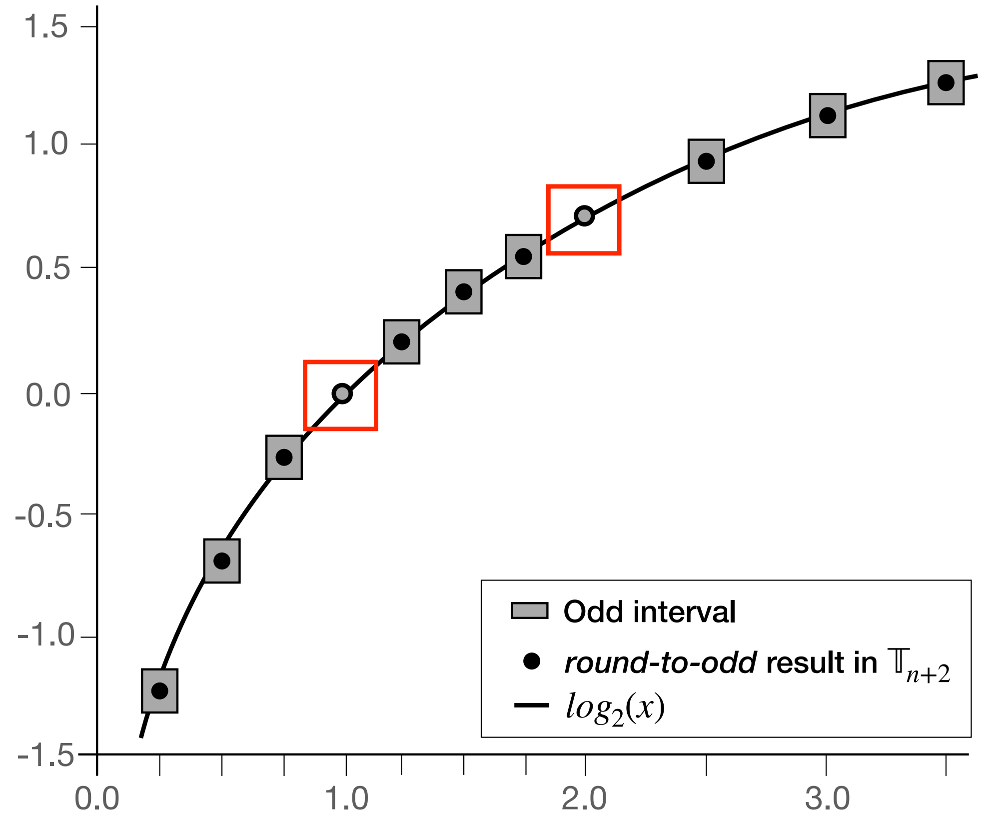
Our RLIBM Approach for Multiple Representations

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode



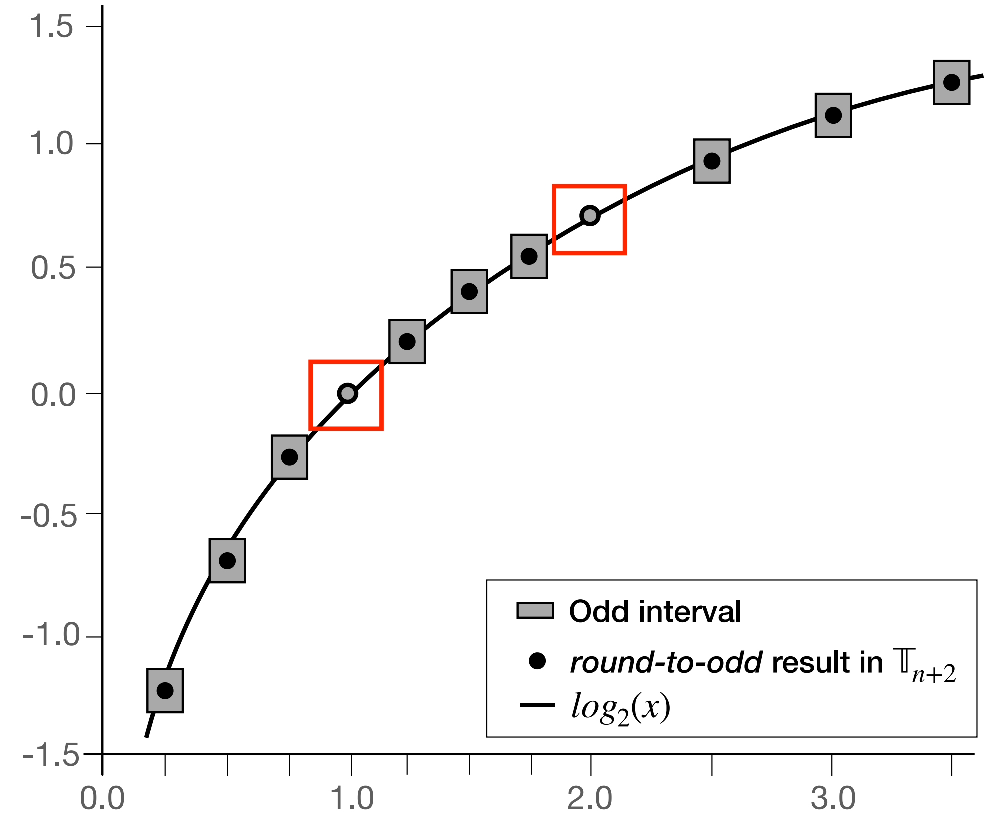
Handling Singleton Intervals

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?



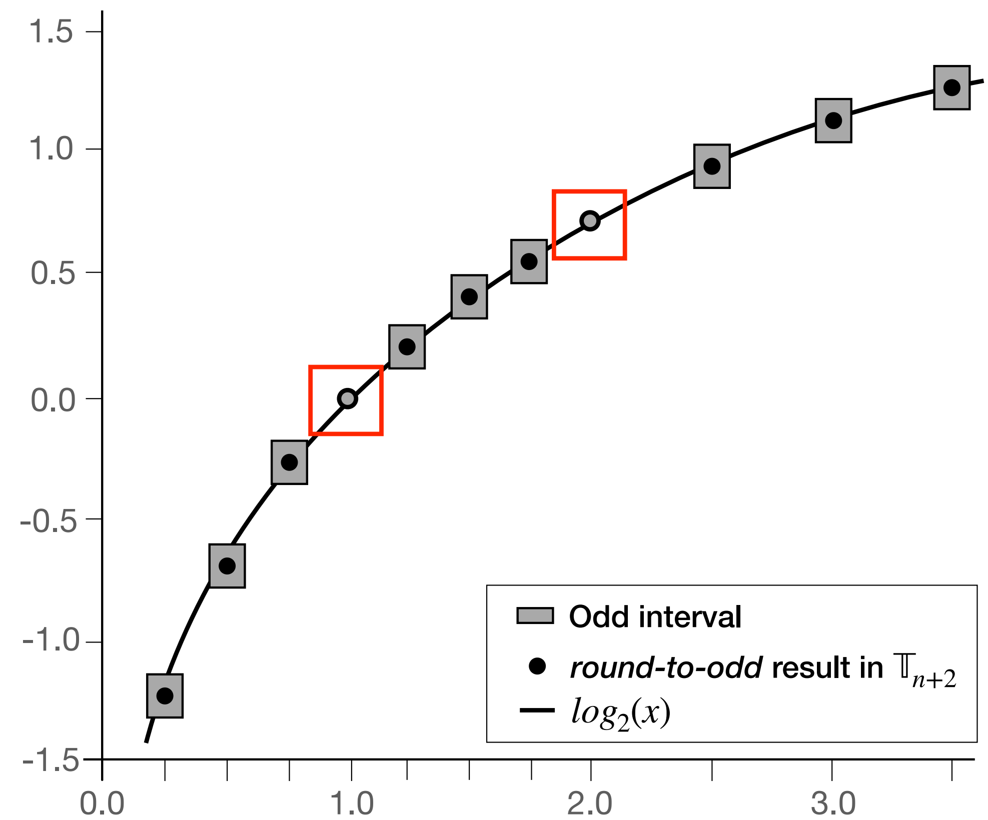
Handling Singleton Intervals

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?



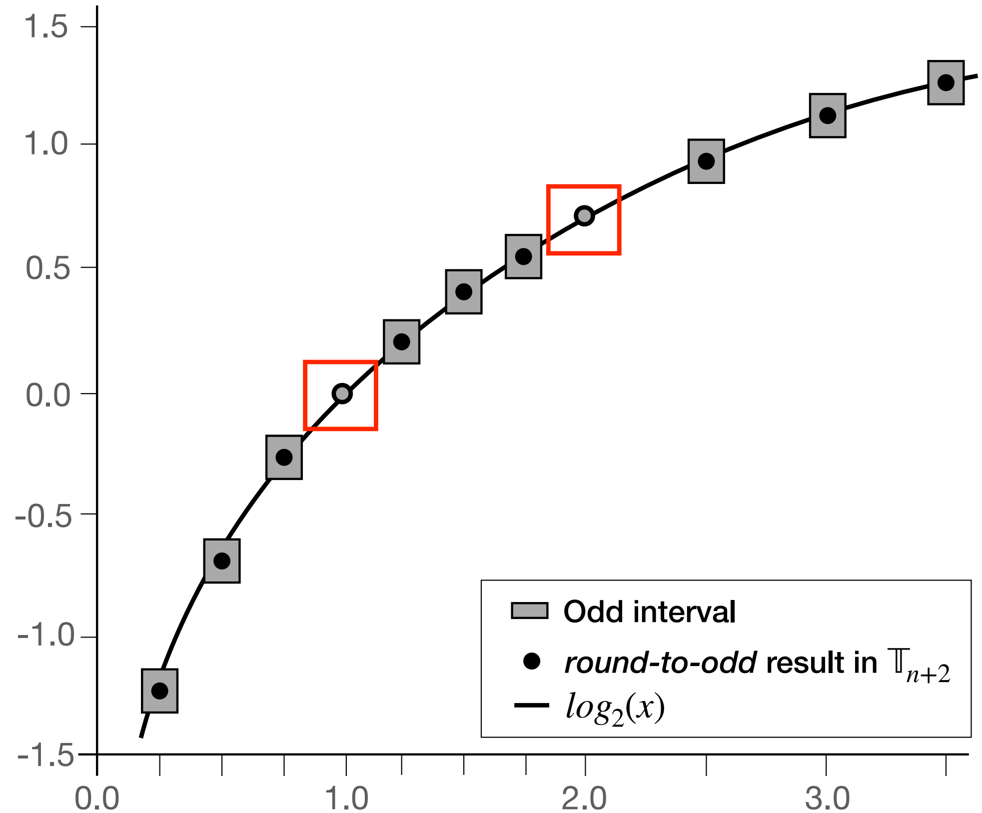
Handling Singleton Intervals

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is $\boxed{\text{exactly representable}}$ in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output



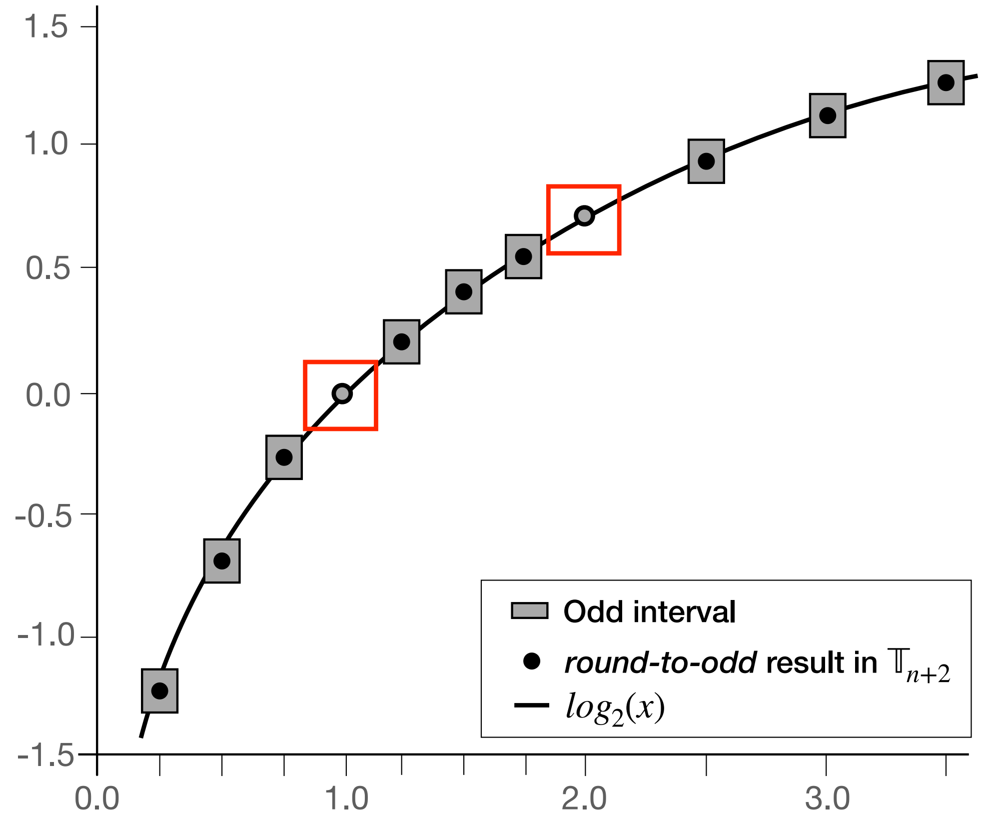
Handling Singleton Intervals

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output
    - When is $log_2(x)$ rational for rational input?



Handling Singleton Intervals

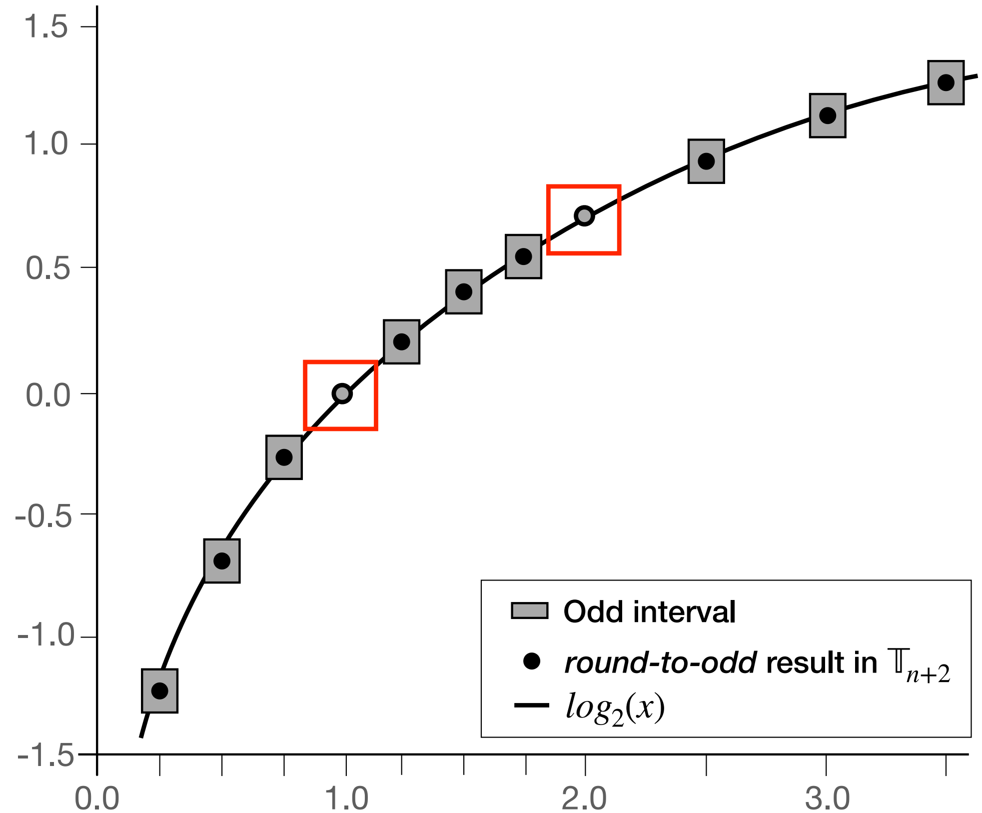
- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output
    - When is $log_2(x)$ rational for rational input?



Several results regarding rationality of elementary functions!

Handling Singleton Intervals

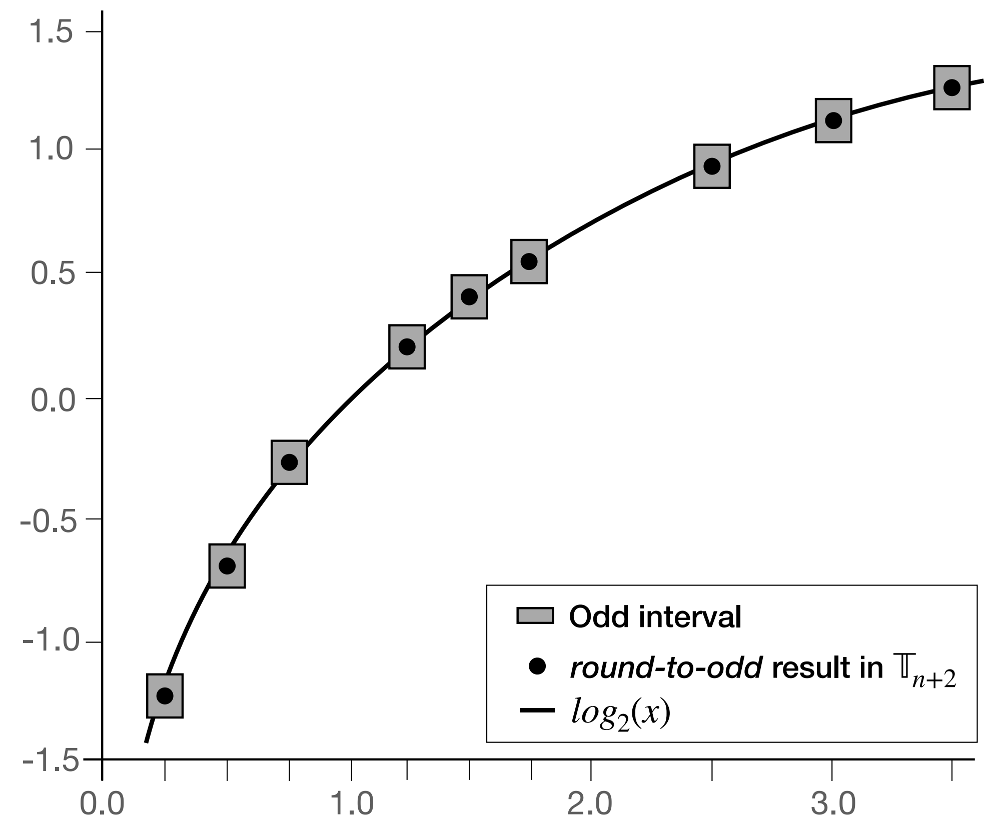
- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output
    - When is $log_2(x)$ rational for rational input?
    - When $x = 2^i$ for integer i



Several results regarding rationality of elementary functions!

Handling Singleton Intervals

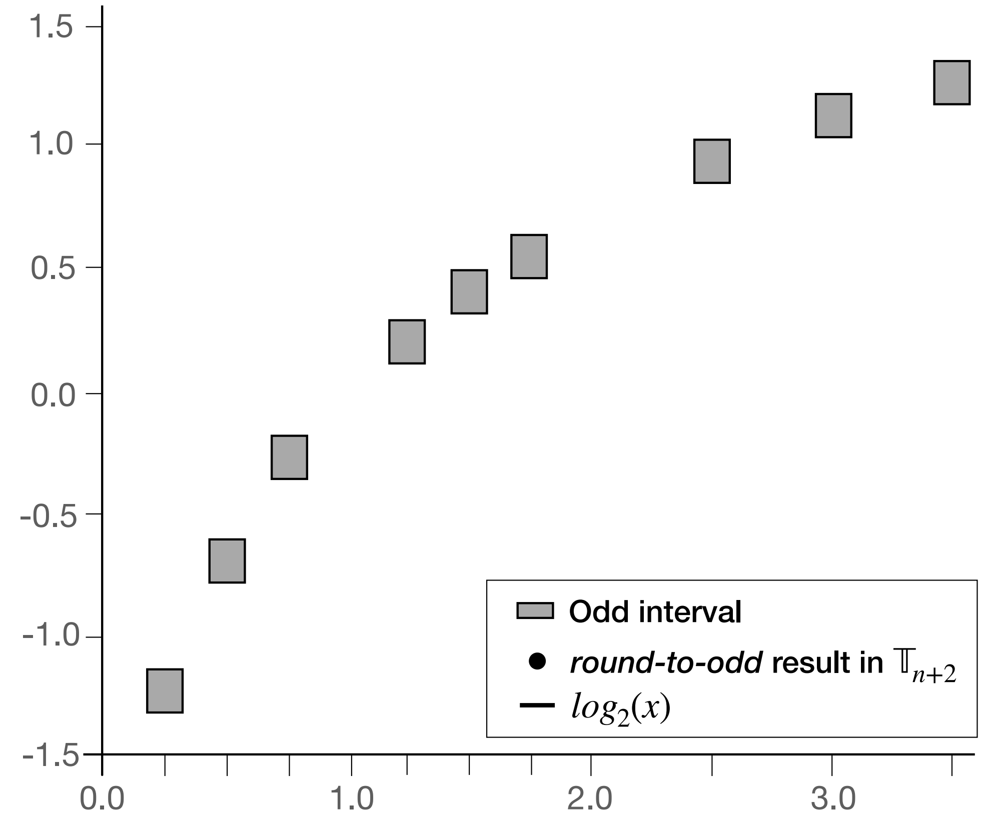
- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is $\boxed{\text{exactly representable}}$ in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output
    - When is $log_2(x)$ rational for rational input?
    - When $x = 2^i$ for integer i



Several results regarding rationality of elementary functions!

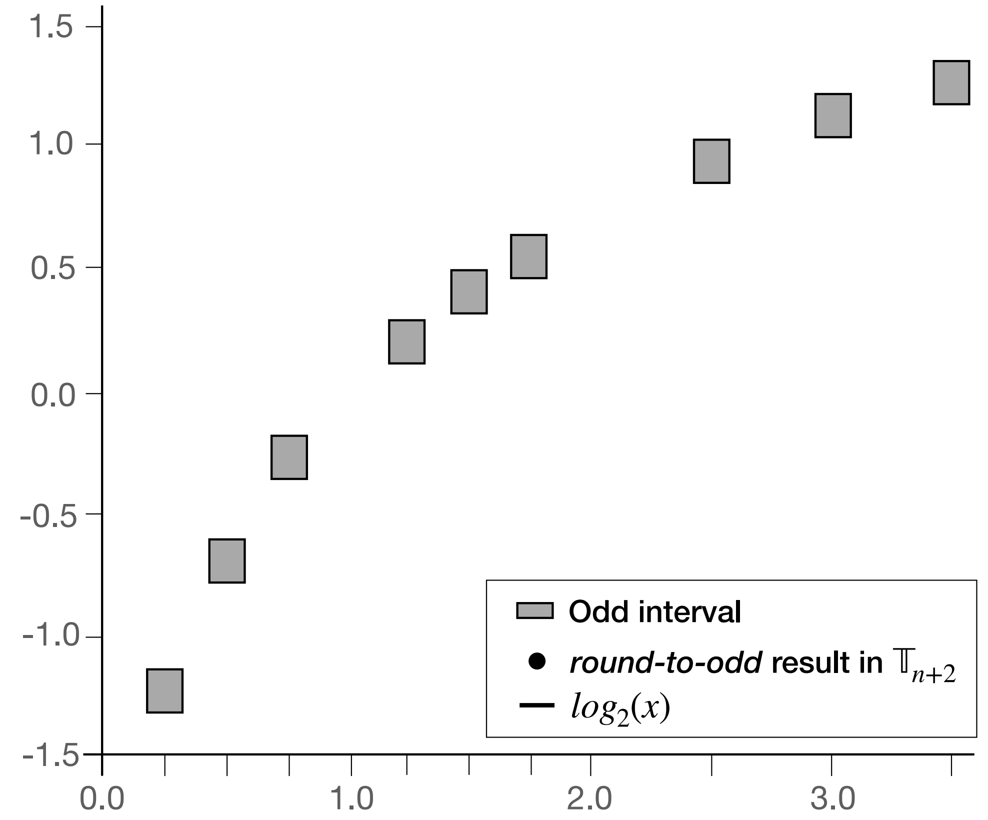
Handling Singleton Intervals

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output
    - When is $log_2(x)$ rational for rational input?
    - When $x = 2^i$ for integer i



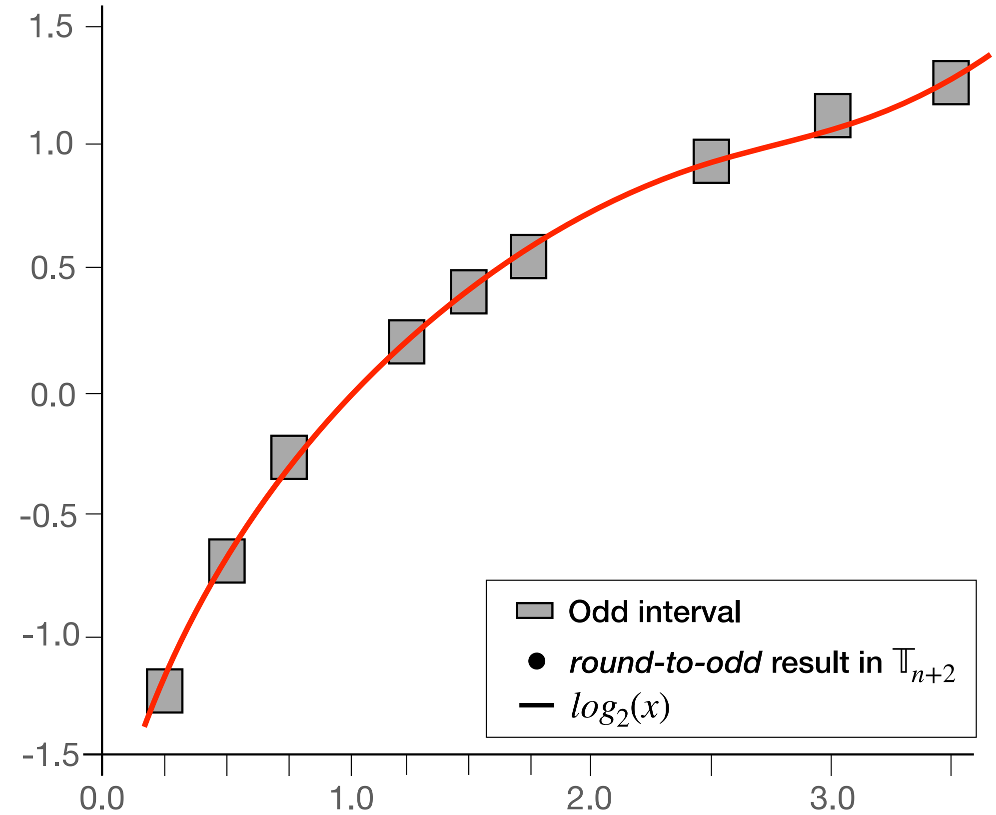
Handling Singleton Intervals

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output
    - When is $log_2(x)$ rational for rational input?
    - When $x = 2^i$ for integer i
- Use the LP formulation to generate a polynomial



Handling Singleton Intervals

- For each float input, compute
  - the *round-to-odd* result of $log_2(x)$ in FP34
  - Rounding interval with *round-to-odd* mode
- Singleton intervals (special case):
  - When $log_2(x)$ is exactly representable in FP34
  - And value is even
  - How to quickly identify them?
    - Rational input with rational output
    - When is $log_2(x)$ rational for rational input?
    - When $x = 2^i$ for integer i
- Use the LP formulation to generate a polynomial



Does it Work?

Our RLIBM Functions Are Correctly Rounded

float functions	Using RLIBM-ALL
ln(x)	✔
log2(x)	✔
log10(x)	✔
exp(x)	✔
exp2(x)	✔
exp10(x)	✔
sinh(x)	✔
cosh(x)	✔
sinpi(x)	✔
cospi(x)	✔

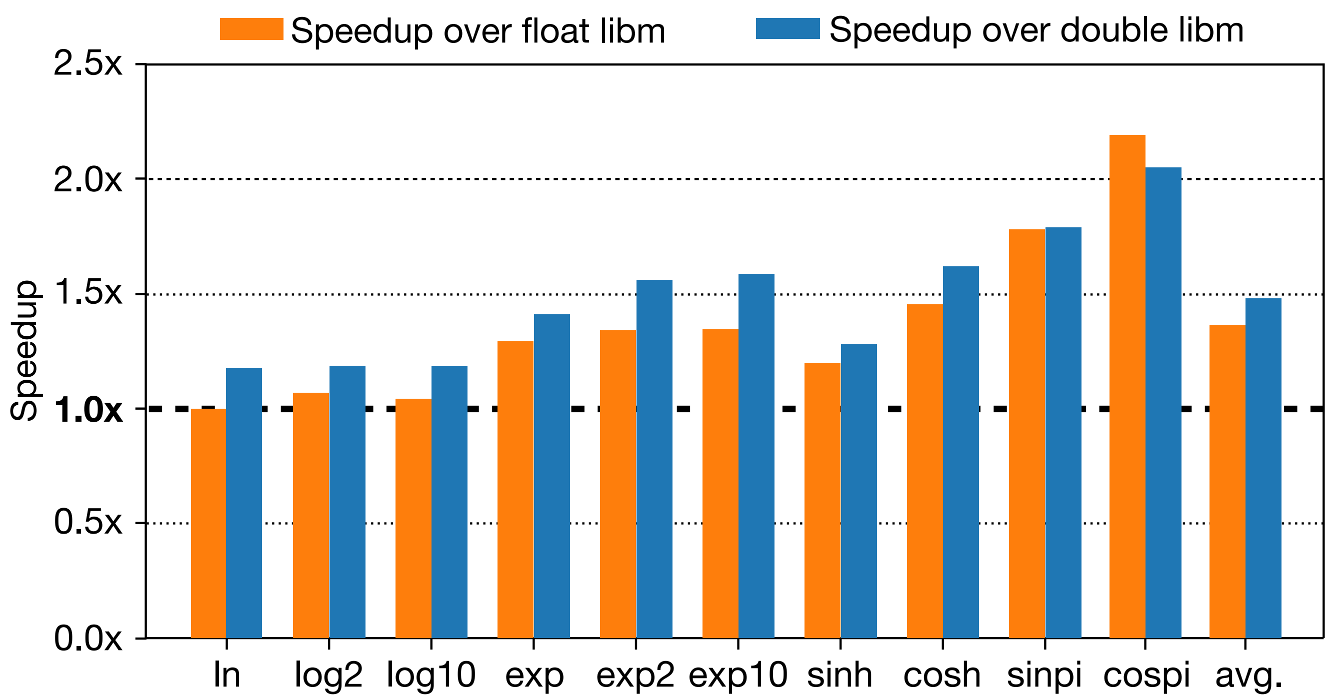
- Produces correctly rounded results for multiple representations
  - ≤ 8 exponent bits (Same or less than 32-bit float)
  - ≤ 23 mantissa bits (same or less than 32-bit float)
  - 161 different configurations
  - Includes float, bfloat16, Tensorfloat32, half, etc
- Supports all five standard rounding modes
- $161 \times 5 = 805$ combinations of configurations and rounding modes

Our RLIBM Functions Are Correctly Rounded

Ability to produce correctly rounded float value with all standard rounding modes for all inputs

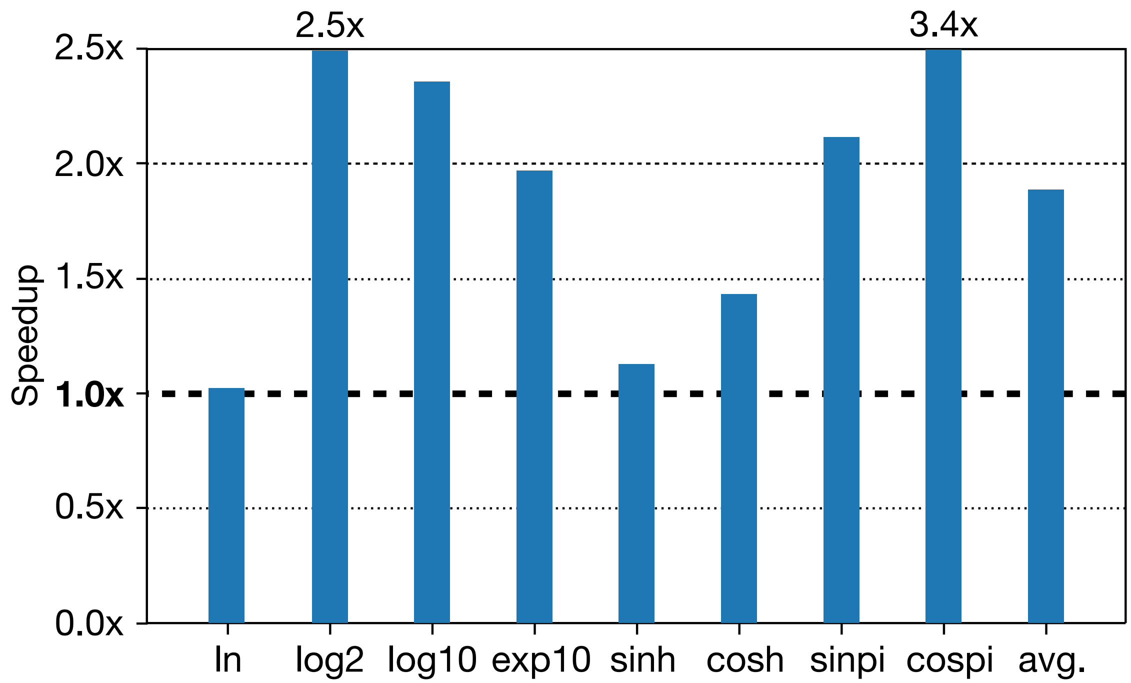
float functions	Using RLIBM-ALL	Using glibc (float)	Using glibc (double)	Using Intel (float)	Using Intel (double)	Using CRLibm (double)
ln(x)	✔	✗	✗	✗	✗	✗
log2(x)	✔	✗	✔	✗	✔	✔
log10(x)	✔	✗	✗	✗	✗	✗
exp(x)	✔	✗	✗	✗	✗	✔
exp2(x)	✔	✗	✗	✗	✗	N/A
exp10(x)	✔	✗	✗	✗	✗	N/A
sinh(x)	✔	✗	✗	✗	✗	✗
cosh(x)	✔	✗	✗	✗	✗	✔
sinpi(x)	✔	N/A	N/A	✗	✗	✔
cospi(x)	✔	N/A	N/A	✗	✗	✔

RLIBM-ALL Functions Are Fast



Speedup over using Intel math library

RLIBM-ALL Functions Are Fast



Speedup over using CR-LIBM library

Transition to Practice

5 functions in LLVM's libc is built using RLIBM's implementations!

[libc] Implement correctly rounded logf based on RLIBM library.

Browse files

`Implement correctly rounded logf based on RLIBM library: https://people.cs.rutgers.edu/~sn349/rlibm/.`

[libc] Implement correctly rounded log2f based on RLIBM library.

Browse files

`Implement log2f based on RLIBM library correctly rounded for all rounding modes.`

× **[libc] Implement log10f correctly rounded for all rounding modes.**

Browse files

`Based on RLIBM implementation similar to logf and log2f.  Most of the exceptional inputs are the exact powers of 10.`

[libc] Improve the performance of expf.

Browse files

`Reduce the polynomial's degree from 7 down to 4.`

[libc] Improve the performance of exp2f.

Browse files

```
Reduce the range-reduction table size from 128 entries down to 64 entries, and
reduce the polynomial's degree from 6 down to 4.

Currently we use a degree-6 minimax polynomial on an interval of length 2^-7
around 0 to compute exp2f.  Based on the suggestion of @santoshn and the RLIBM
project (https://github.com/rutgers-apl/rlibm-prog/blob/main/libm/float/exp2.c)
it is possible to have a good polynomial of degree-4 on a subinterval of length
2^(-6) to approximate 2^x.

We did try to either reduce the degree of the polynomial down to 3 or increase
the interval size to 2^(-5), but in both cases the number of exceptional values
exploded.  So we settle with using a degree-4 polynomial of the interval of
size 2^(-6) around 0.

Reviewed By: michaelrj, sivachandra, zimmermann6, santoshn

Differential Revision: https://reviews.llvm.org/D122346
```

main

109acea92e1acb661c404fa62b9

Conclusion

- Approximate the correctly rounded result rather than the real value
  - Linear programming formulation based on the interval around the correct result
- How to generate a generic polynomial for all representations up to n-bits?
  - Generate polynomials for (n+2)-bit representation
  - That produces correct results with round-to-odd mode
  - Quickly identify singleton intervals: when $f(x)$ is rational for rational input
- Our RLIBM prototype produces correctly rounded results for all inputs in
  - 161 different FP configurations
  - All 5 IEEE-754 standard rounding modes
- Faster than mainstream math libraries

Make correctly rounded results mandatory rather than a recommendation by the standards

Open Source

Visit the RLIBM page for papers & prototypes
https://www.cs.rutgers.edu/~santosh.nagarakatte/rlibm/

Thanks National Science Foundation for the support