Sound Probabilistic Numerical Error Analysis

How do we compute the distribution of numerical errors at the output?

Debasmita Lohar, Milos Prokop, Eva Darulova

MAX PLANCK INSTITUTE FOR SOFTWARE SYSTEMS

UNIVERSITÄT DES SAARLANDES

THE UNIVERSITY OF EDINBURGH

iFM 2019

Programming with Numerical Errors

```
def func(x:Real, y:Real, z:Real): Real = {
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

- **Reals** are implemented in **Floating point**/**Fixed point** data type

Programming with Numerical Errors

```
        (x:Float32, y:Float32, z:Float32): Float32
def func(x:Real, y:Real, z:Real): Real = {
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

- **Reals** are implemented in **Floating point**/**Fixed point** data type
- Introduces **round-off error**

Programming with Numerical Errors

```
        (x:Float32, y:Float32, z:Float32): Float32
def func(x:Real, y:Real, z:Real): Real = {
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}
```

We need to bound the round-off error

- **Reals** are implemented in **Floating point/Fixed point** data type
- Introduces **round-off error**

State-of-the-art: Worst Case Error Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res
}
```

Daisy FLUCTUAT

Gappa rosa FPTaylor

....

State-of-the-art: Worst Case Error Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

Daisy FLUCTUAT

Gappa rosa FPTaylor

....

Worst case error: **0.002**

Computes **absolute** round-off error

State-of-the-art: Worst Case Error Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

Daisy FLUCTUAT

Gappa rosa FPTaylor

....

Worst case error: **0.002**

Occurs only with probability 0.002 !

Error Resilient Applications

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

Applications may tolerate large infrequent errors

A controller system can tolerate big errors while stabilizing

Error Resilient Applications

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res +/- error
} ensuring (error <= 0.00199, 0.85)
```

Application tolerates big errors occurring with **<=** **0.15** probability

Applications may tolerate large infrequent errors

Worst Case Analysis = poor resource utilization

```
        (x:Float64, y:Float64, z:Float64): Float64
def func(x:Float32, y:Float32, z:Float32): Float32 = {
require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res +/- error
} ensuring (error <= 0.00199, 0.85)
```

Application tolerates big errors occurring with **<=** **0.15** probability

Applications may tolerate large infrequent errors

With only **Worst case Analysis**, we need to change **precision**

Worst Case Analysis = poor resource utilization

```
            (x:Float64, y:Float64, z:Float64): Float64
def func(~~x:Float32, y:Float32, z:Float32): Float32~~ = {
require (0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0)
   val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res +/- error
} ensuring (error <= 0.00199, 0.85)
```

Application tolerates big errors occurring with **<=** **0.15** probability

Applications may tolerate large infrequent errors

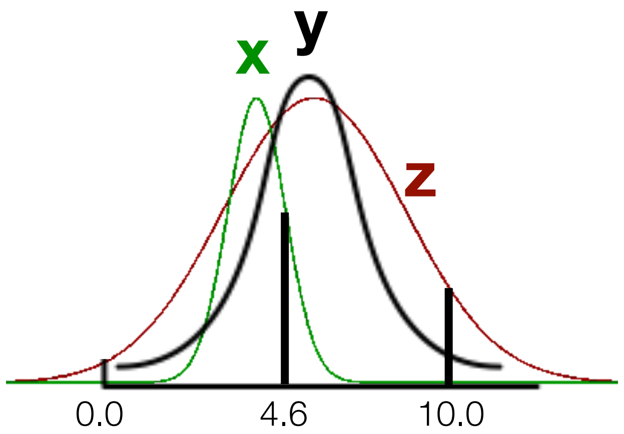
With only **Worst case Analysis**, we need to change **precision**

Need to consider the **probability distributions** of **inputs**

Our Goal: Probabilistic Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {

    x:= gaussian(0.0, 4.6)
    y:= gaussian(0.0, 10.0)
    z:= gaussian(0.0, 10.0)
```




```
    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
} ensuring (error <= 0.00199, 0.85)
```

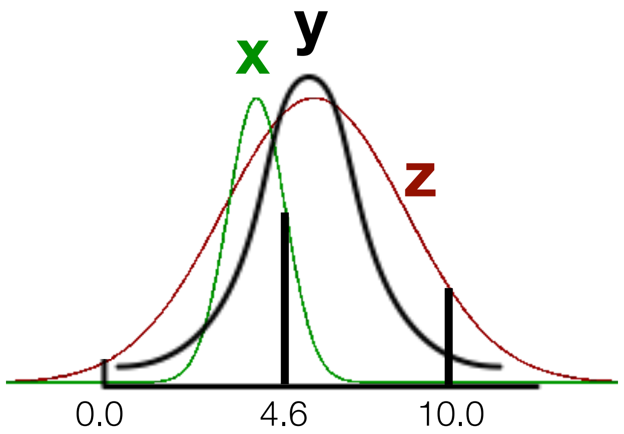
We consider **probability distributions** of **inputs**

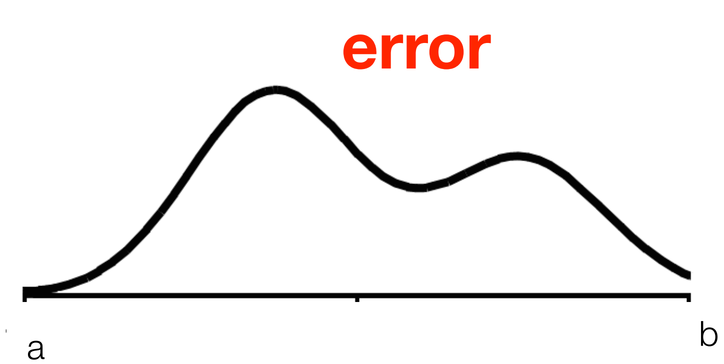
Our Goal: Probabilistic Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {

    x:= gaussian(0.0, 4.6)
    y:= gaussian(0.0, 10.0)
    z:= gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
} ensuring (error <= 0.00199, 0.85)
```





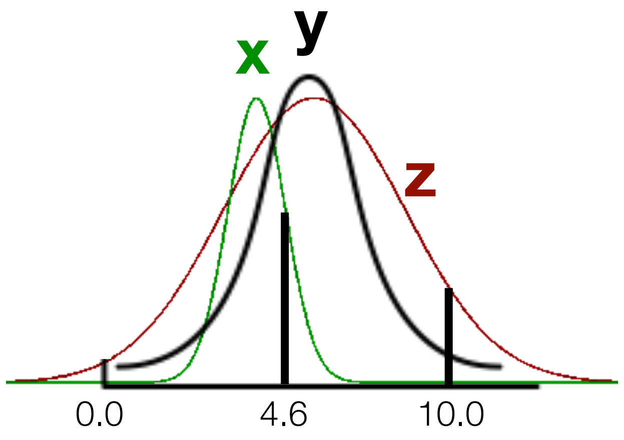
- Compute **probability distribution** of **error**

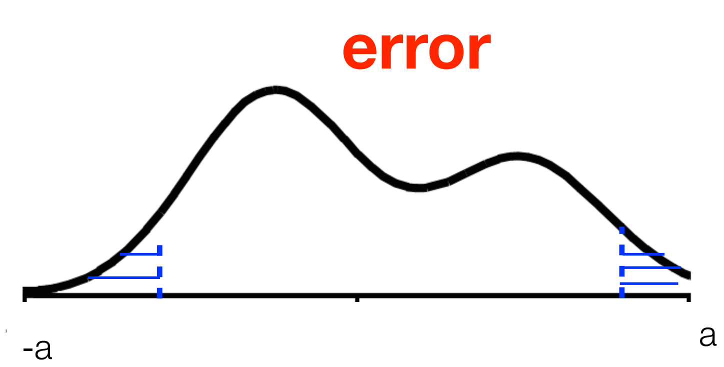
Our Goal: Probabilistic Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {

    x:= gaussian(0.0, 4.6)
    y:= gaussian(0.0, 10.0)
    z:= gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
} ensuring (error <= 0.00199, 0.85)
```





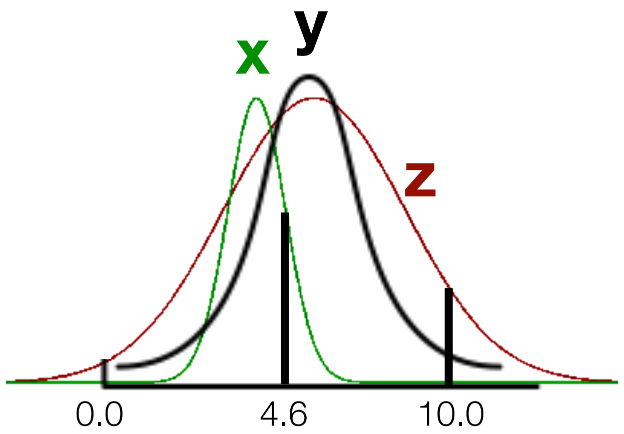
- Compute **probability distribution** of **error**
- Compute a **smaller error** given a **threshold**

Approximate Hardware

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {

    x:= gaussian(0.0, 4.6)
    y:= gaussian(0.0, 10.0)
    z:= gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
} ensuring (error <= 0.00199, 0.85)
```



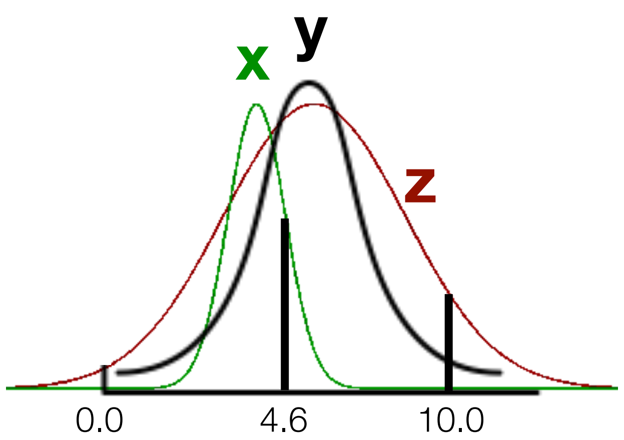
What happens if we have **Approximate Hardware** with **Probabilistic Error Specification**?

Probabilistic Analysis for Approximate Hardware

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {

    x:= gaussian(0.0, 4.6)
    y:= gaussian(0.0, 10.0)
    z:= gaussian(0.0, 10.0)

    val res = -3.79*x - 5.44*y + 9.73*z + 4.52
    return res +/- error
} ensuring (error <= 0.00199, 0.85)
```



Error Specification: **<0.00199, 0.9>, <0.00499, 0.1>**

Can we compute a **smaller** error given 0.85 as **threshold**?

Contributions

- Sound analysis of probabilistic numerical errors
  - considers probability distribution of inputs and computes error distribution

- Application on Approximate Hardware
  - considers probability distribution of error specification

- Prototype implementation on top of Daisy
  https://github.com/malyzajko/daisy/tree/probabilistic

Overview: Sound Analysis

Finite Precision Program with Probabilistic Inputs

```
def func(..) {
 x := gaussian(0.0, 4.6)
 y := gaussian(0.0, 10.0)
 z := gaussian(0.0, 10.0)
 res = -3.79*x - 5.44*y + 9.73*z + 4.52
 return res
}
```

Probabilistic Round-off Error Analysis

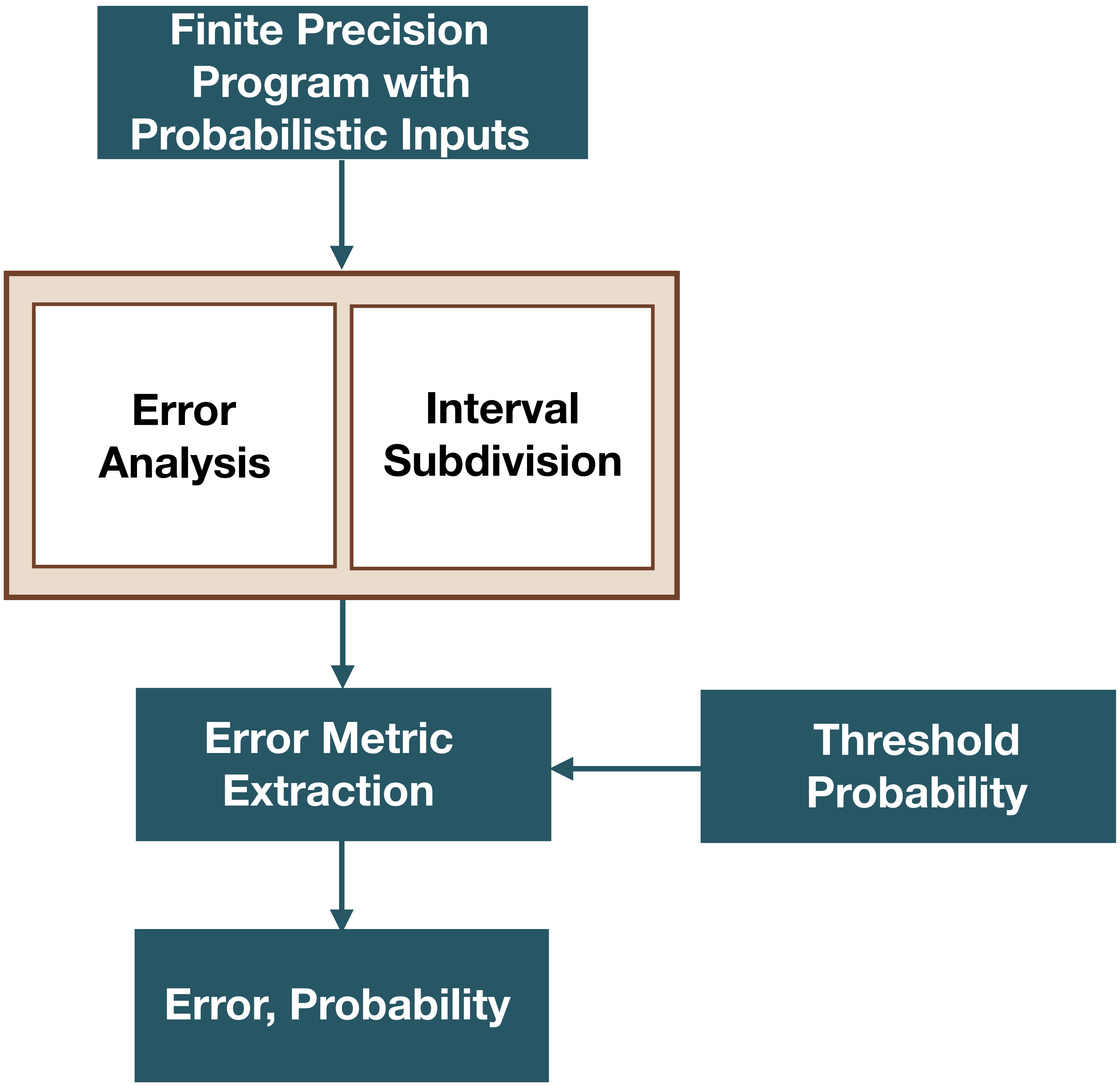
Error Metric Extraction

Threshold Probability

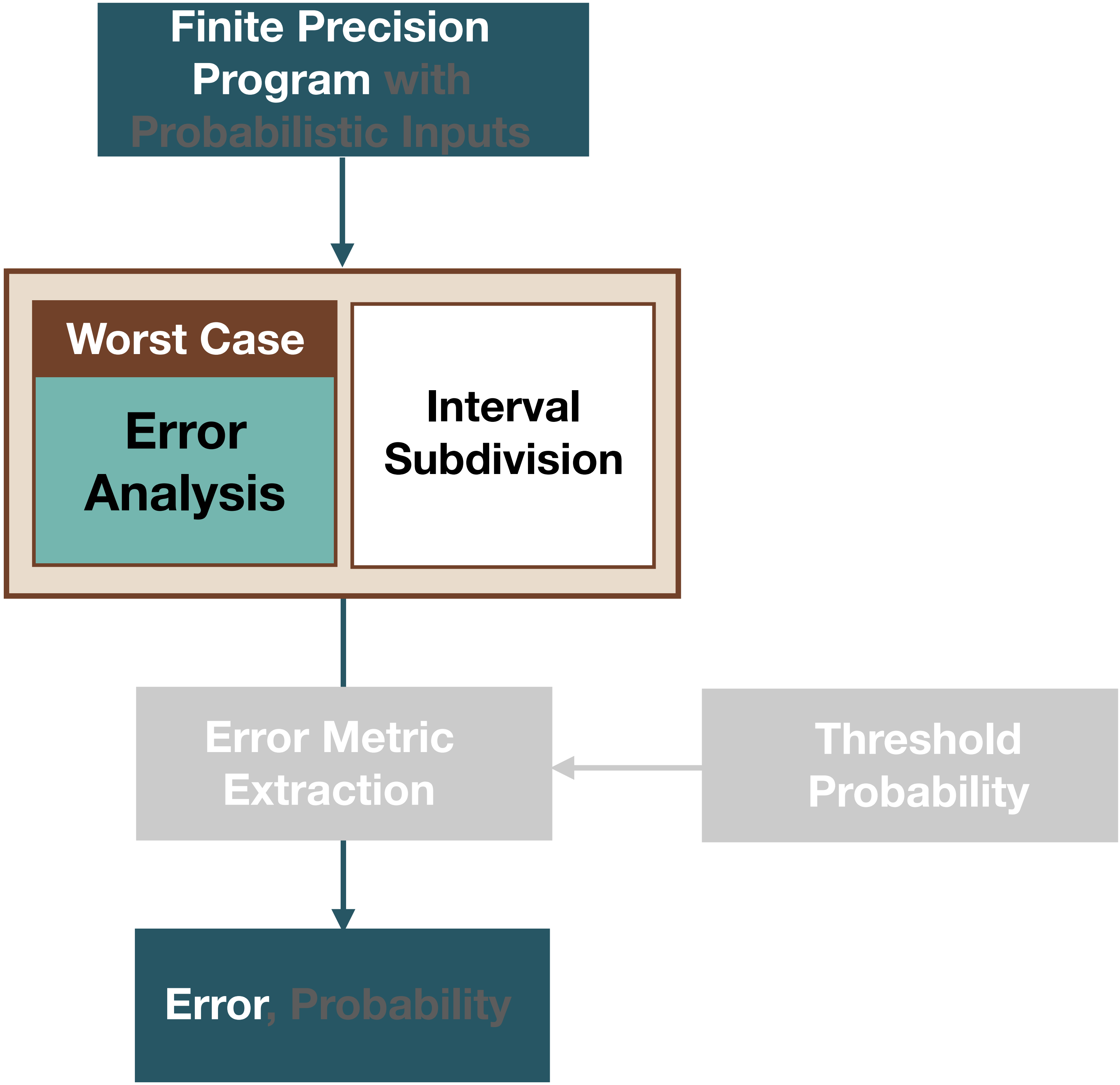
0.85

Error, Probability

Overview: Sound Analysis

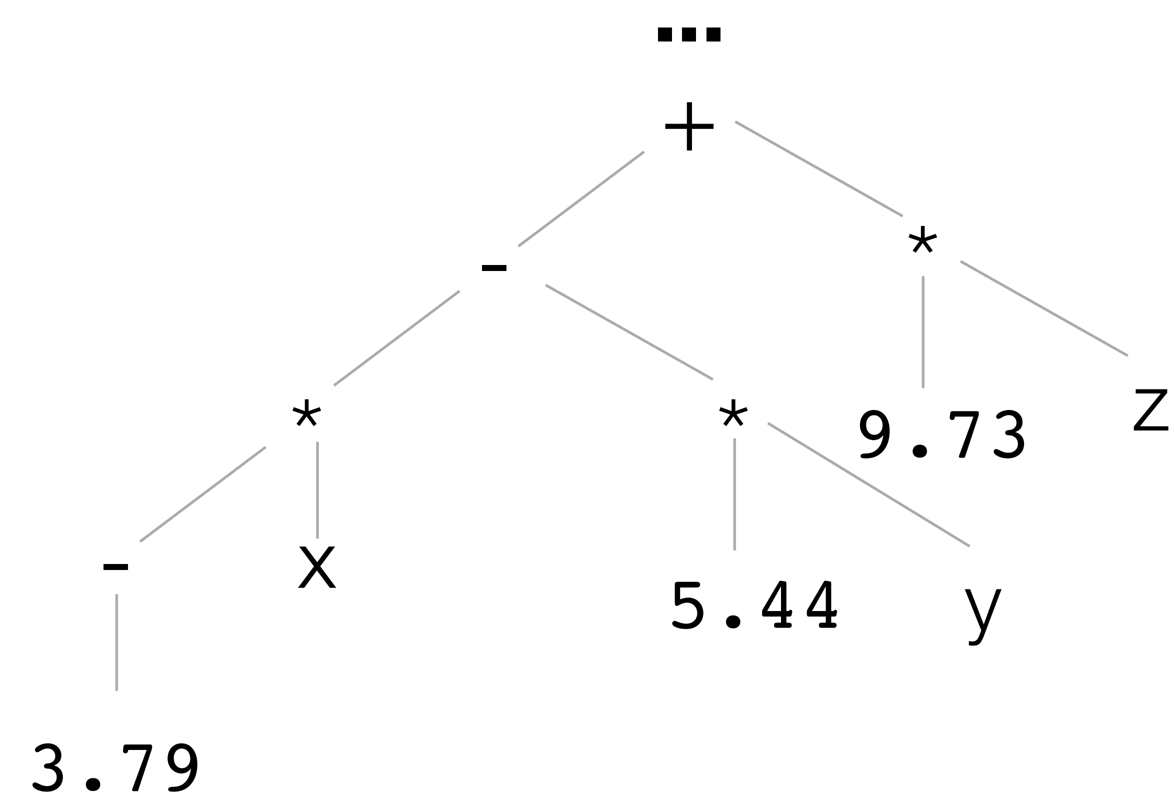


Before going into Probabilistic Analysis...



Background: Worst Case Error Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
  val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

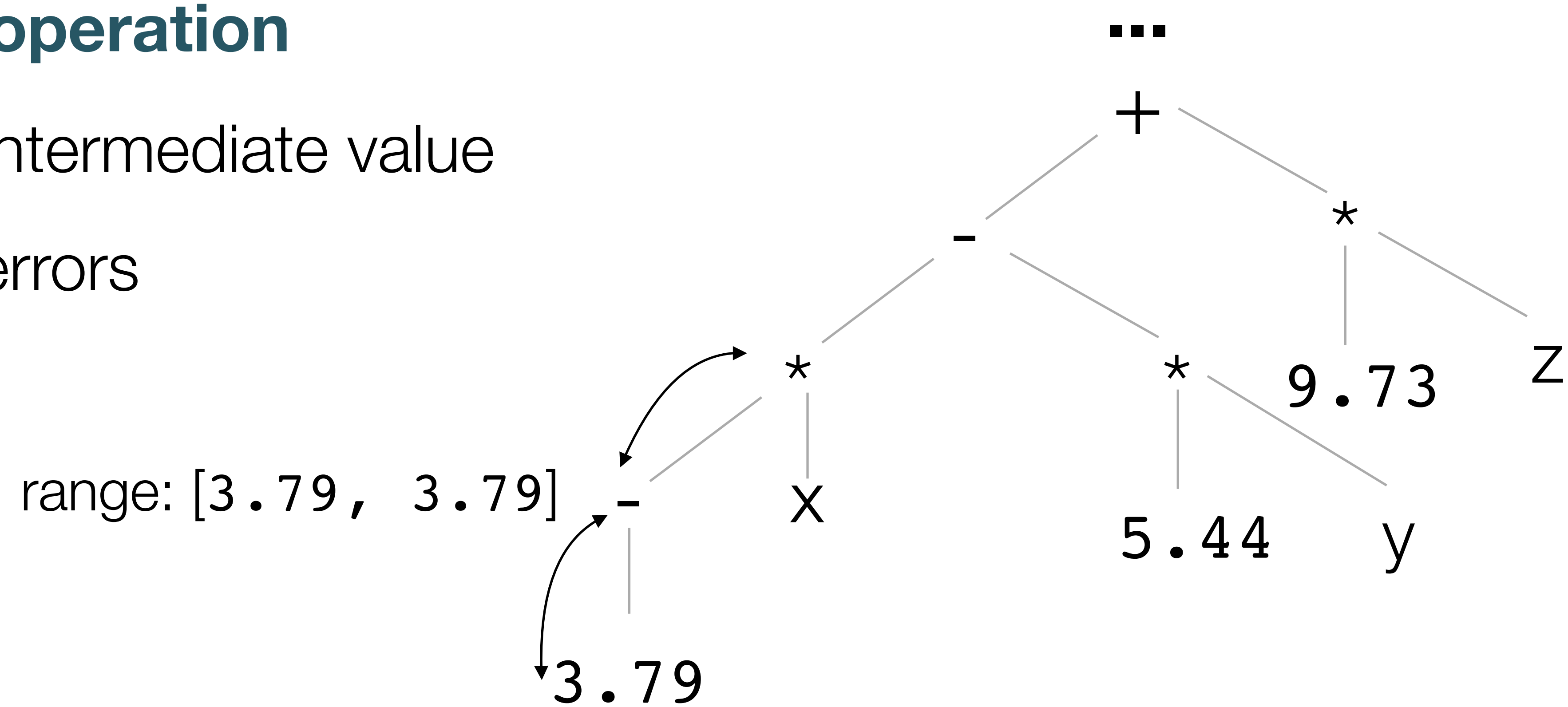


Background: Worst Case Error Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
  val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

For **each arithmetic operation**

- compute range for intermediate value
- propagate existing errors



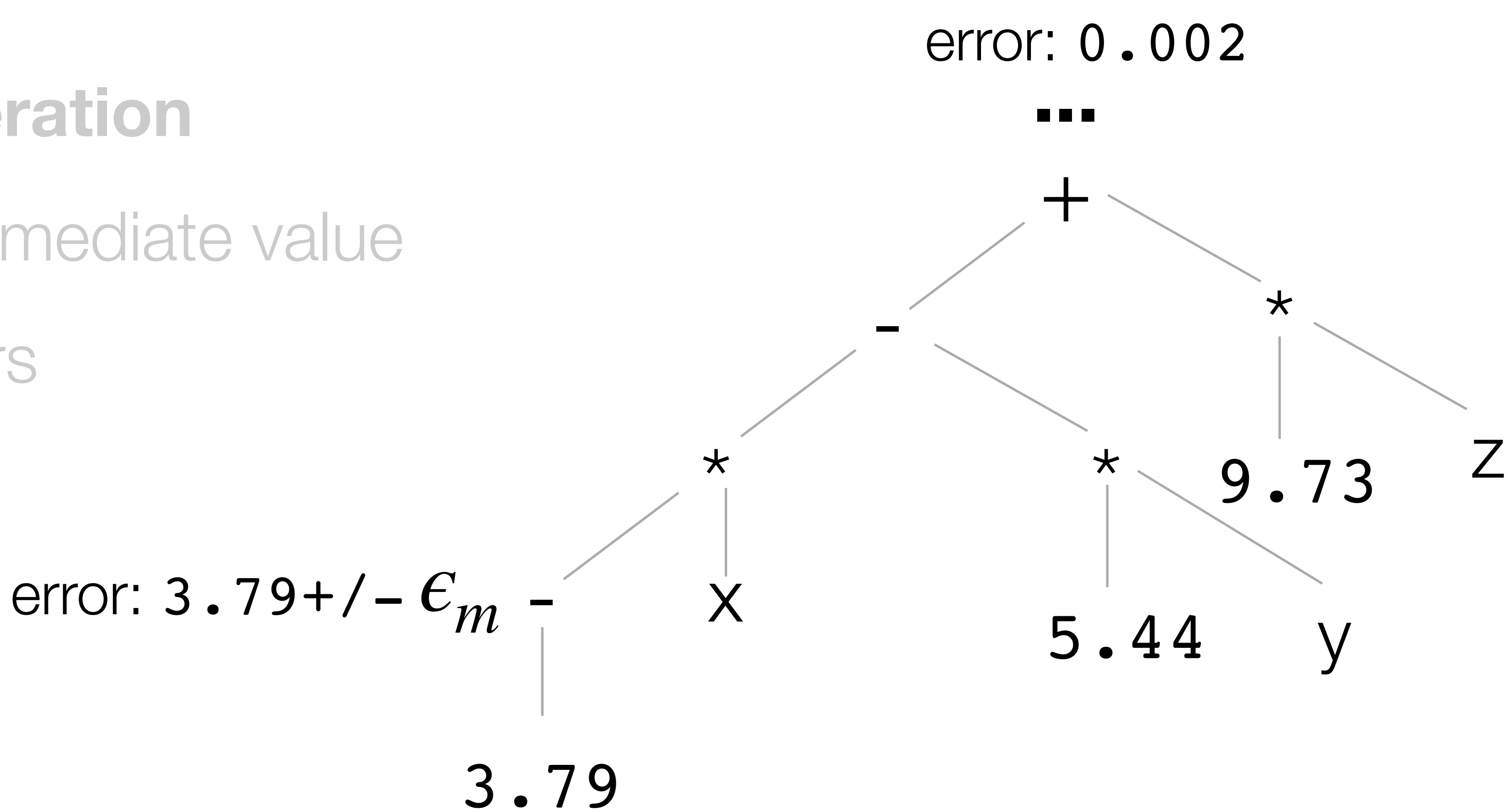
Uses **Interval** / **Affine Arithmetic**

Background: Worst Case Error Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
  val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

For **each arithmetic operation**

- compute range for intermediate value
- propagate existing errors
- compute new errors



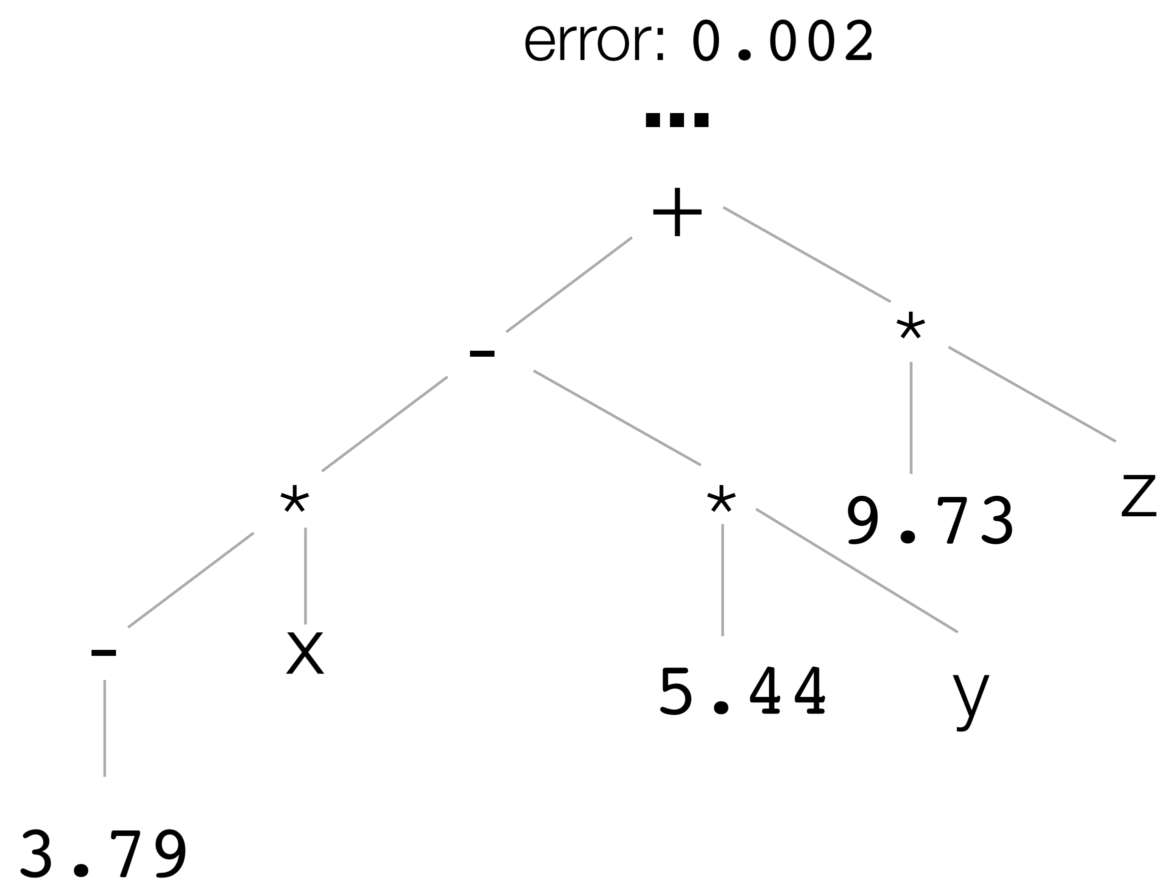
Uses **Interval** / **Affine Arithmetic**

Background: Worst Case Error Analysis

```
def func(x:Float32, y:Float32, z:Float32): Float32 = {
  val res = -3.79*x - 5.44*y + 9.73*z + 4.52
   return res
}
```

For **each arithmetic operation**

- compute range for intermediate value
- propagate existing errors
- compute new errors

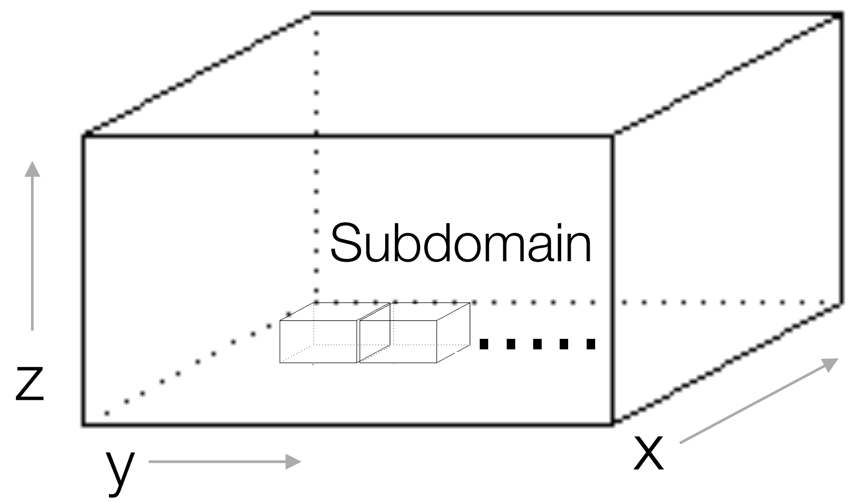


To compute **precise** error, **subdivide** the intervals

Background: Interval Subdivision

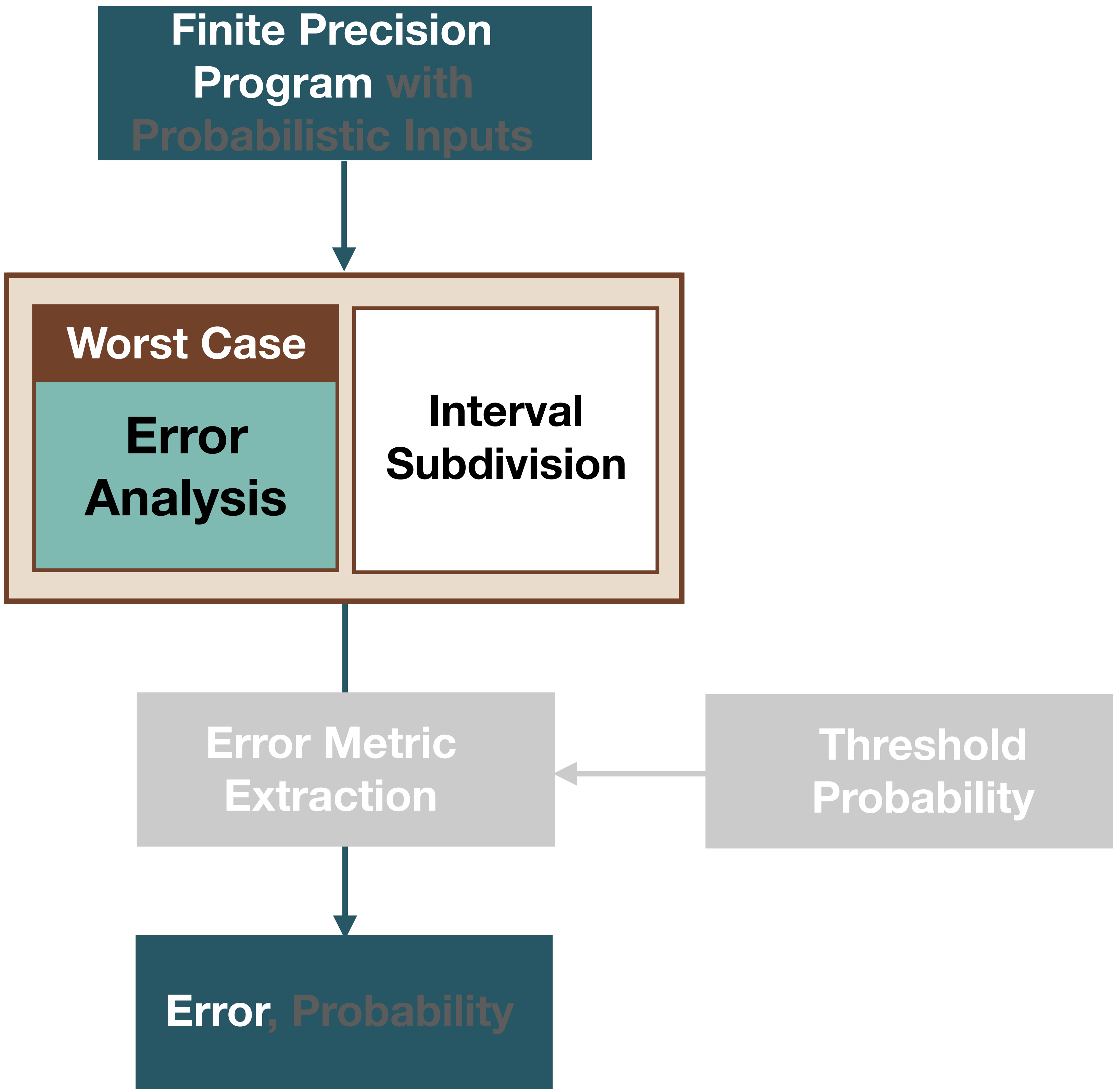
Input Space:

`0.0 <= x <= 4.6 && 0.0 <= y, z <= 10.0`

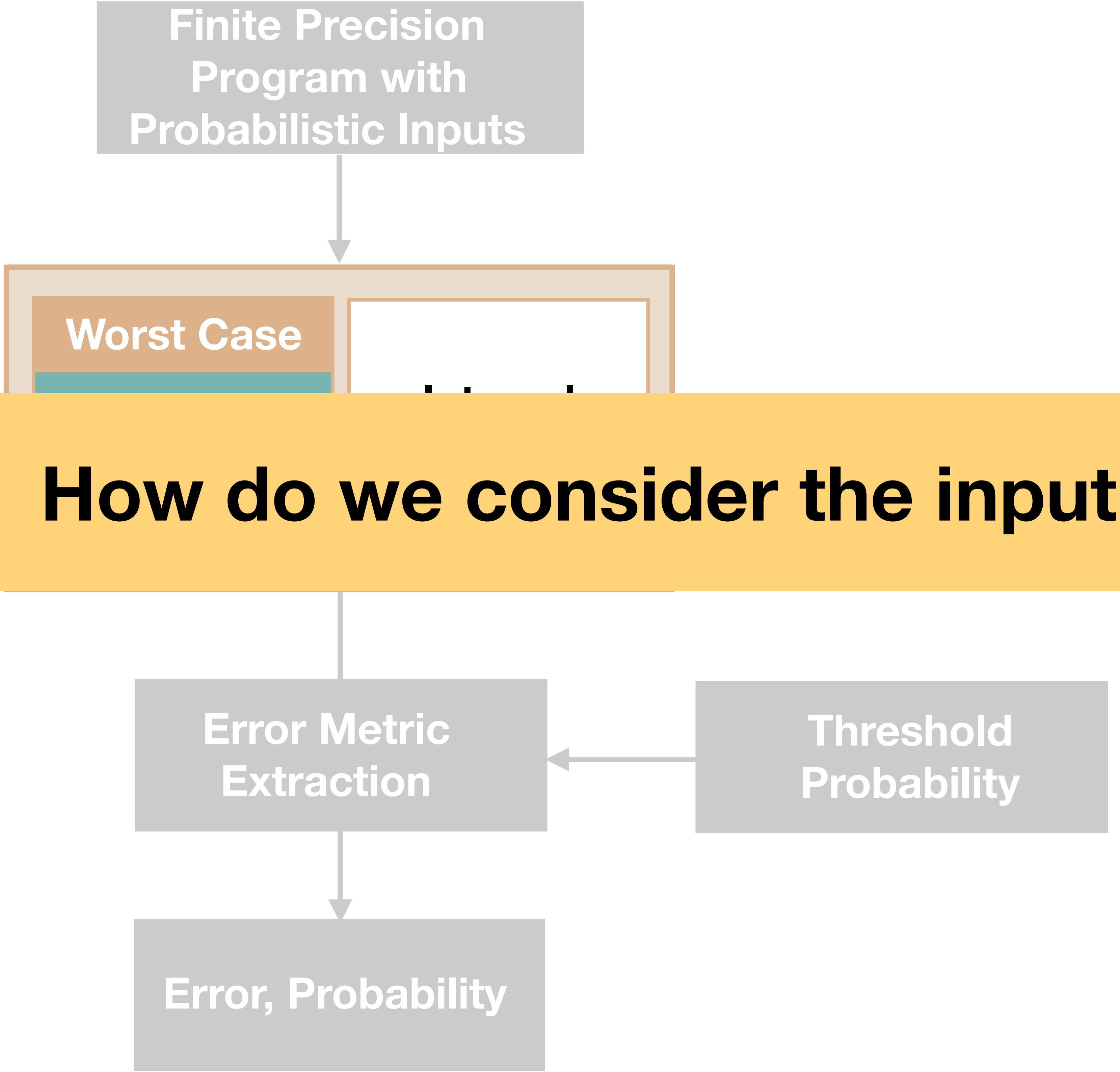


- Generate subdomains
- For each subdomain, compute the **worst case error**
- Return the max abs error

Background: Worst Case Analysis with Subdivision

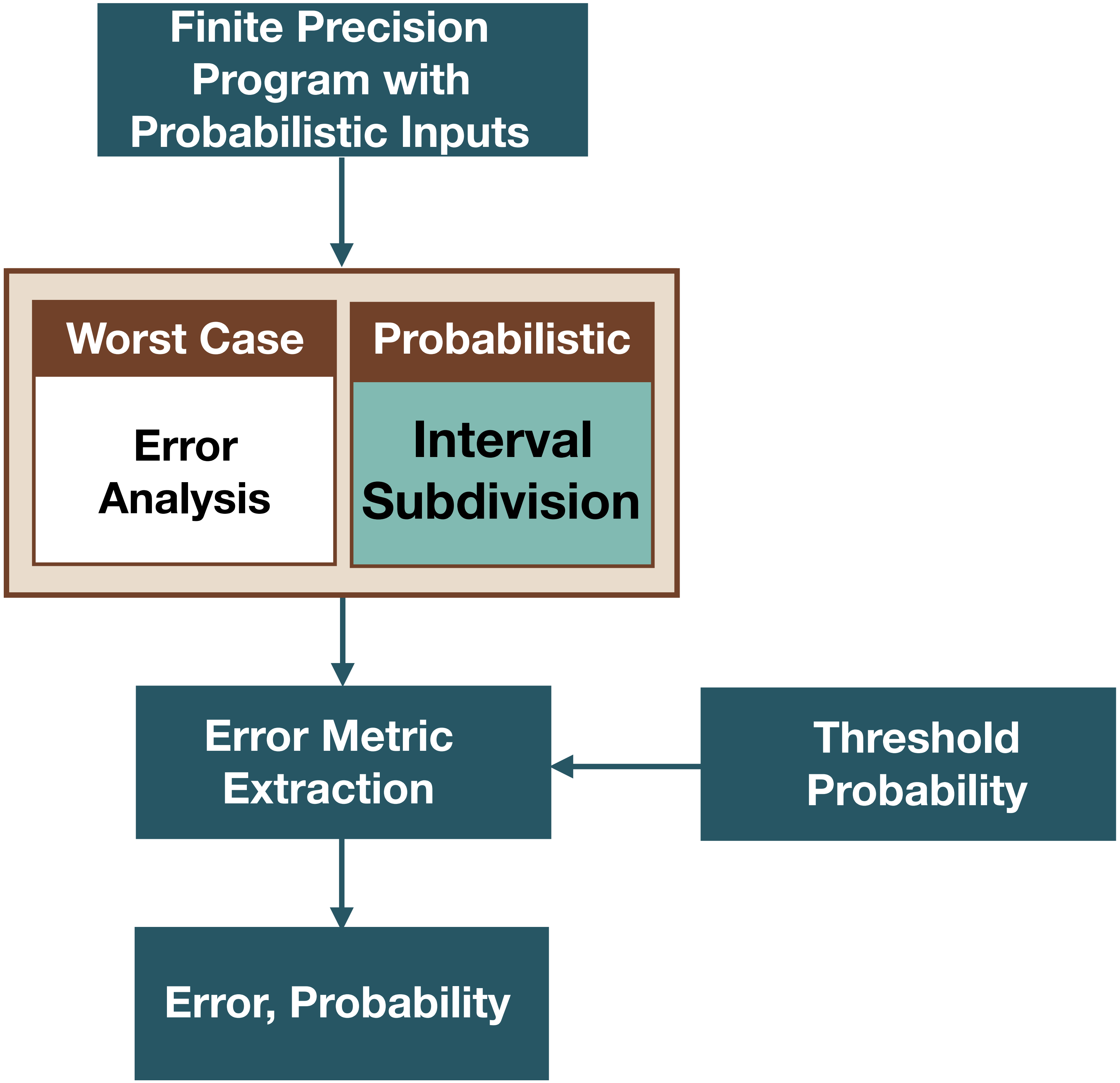


Background: Worst Case Analysis with Subdivision



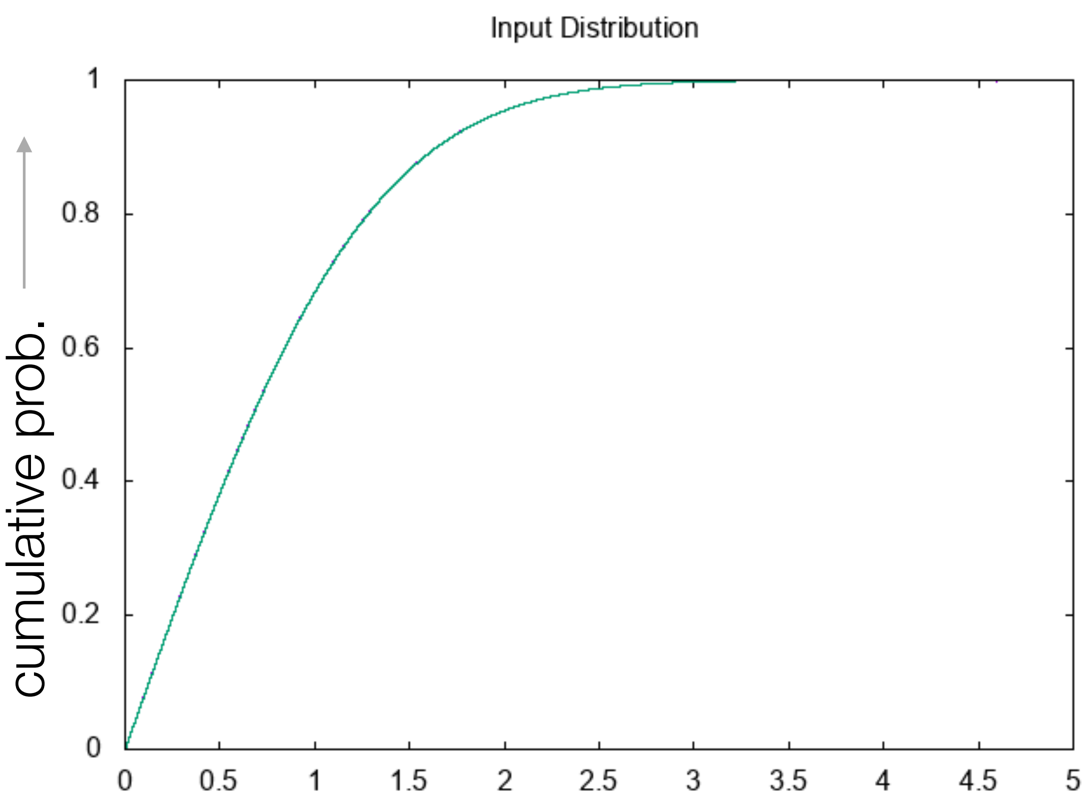
How do we consider the input distributions?

Overview: Sound Probabilistic Analysis



Probabilistic Interval Subdivision

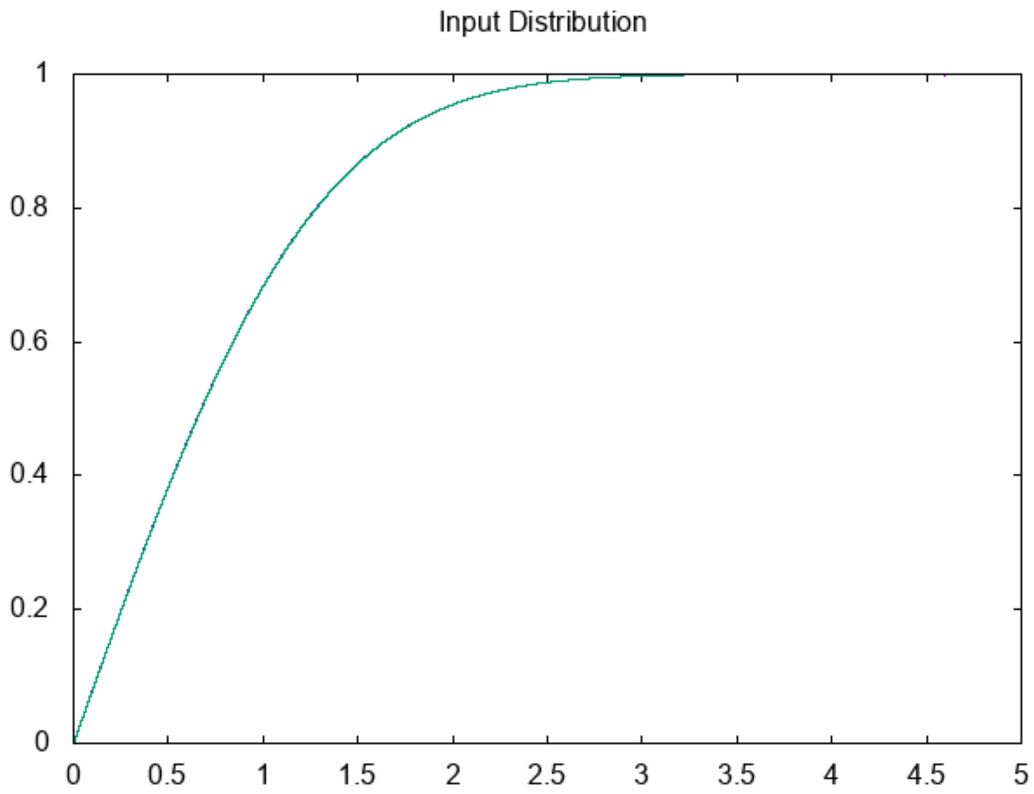
```
x := gaussian(0.0, 4.6)
```



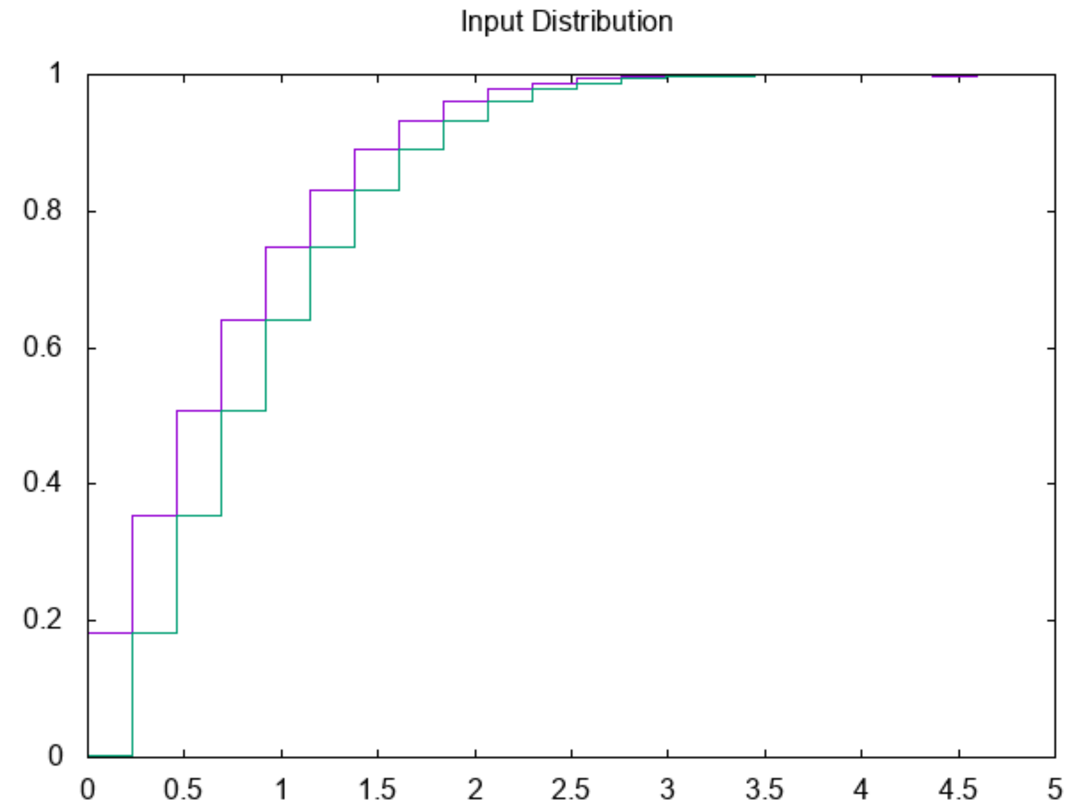

```
def func(..) {
 x := gaussian(0.0, 4.6)
 y := gaussian(0.0, 10.0)
 z := gaussian(0.0, 10.0)
 res = -3.79*x - 5.44*y + 9.73*z + 4.52
 return res
}
```

Probabilistic Interval Subdivision

```
x := gaussian(0.0, 4.6)
```



Discretize

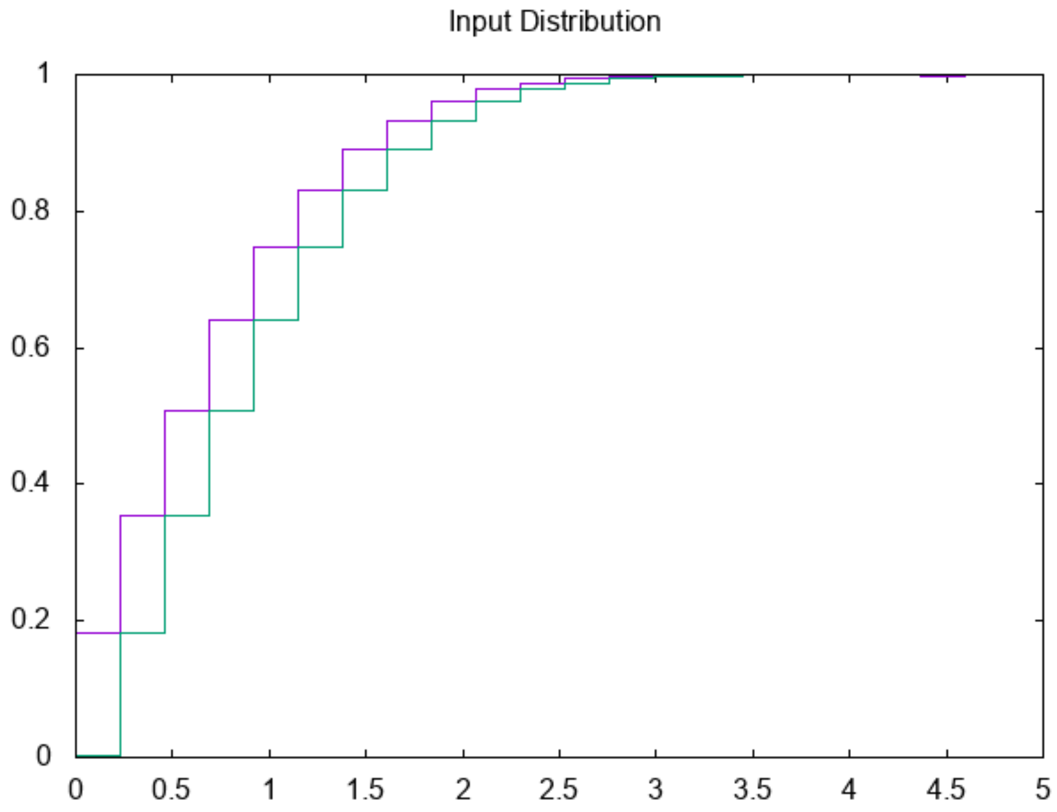


Discretize the continuous distribution into a **set** of **intervals** and **probabilities**

Probabilistic Interval Subdivision

```
x := gaussian(0.0, 4.6)
```

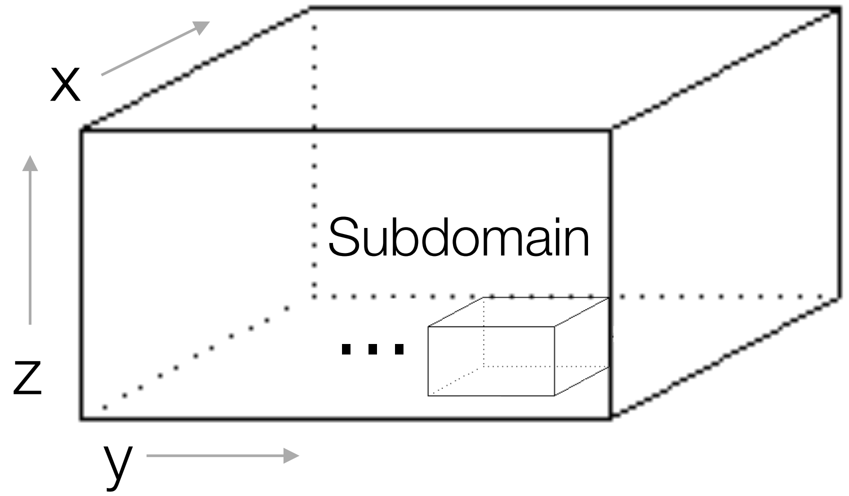
```
<[2.30, 4.60], 0.02>
          .
          .
          .
<[0.20, 0.45], 0.17>
<[0.00, 0.20], 0.18>
```



Discretize the continuous distribution into a **set** of **intervals** and **probabilities**

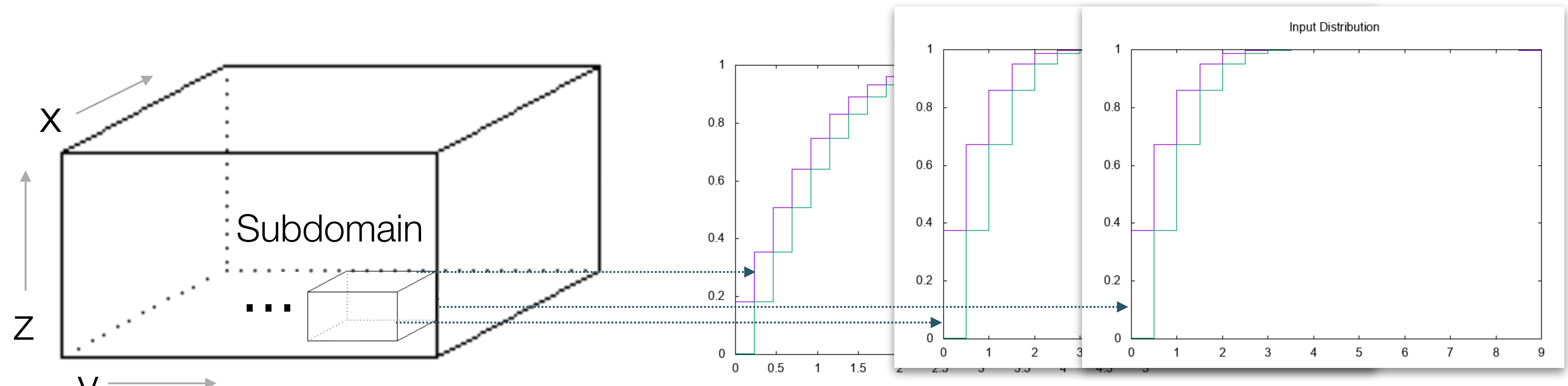
Probabilistic Interval Subdivision

Input Space:



Probabilistic Interval Subdivision

Input Space:



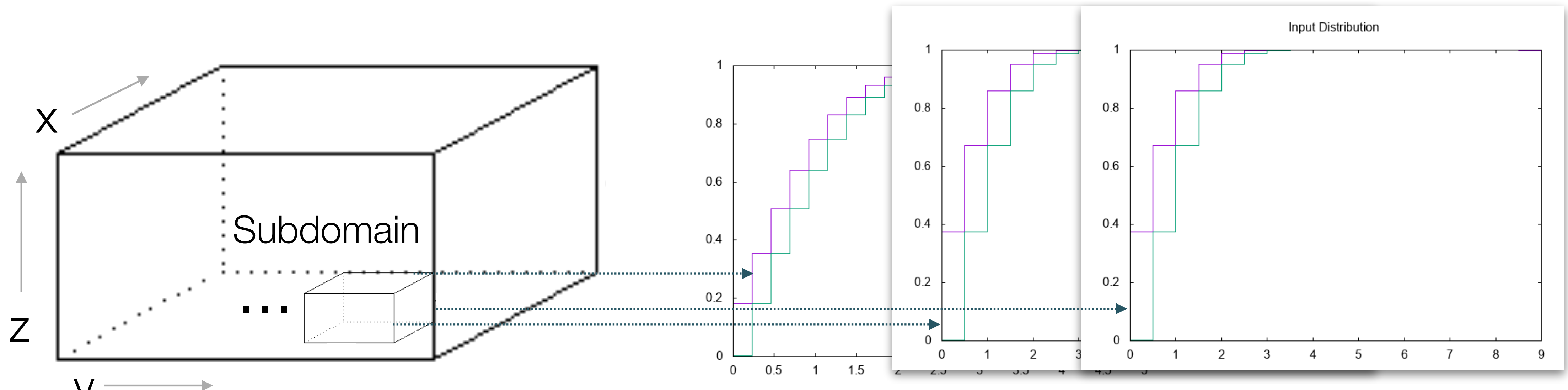
`x:=gaussian(0.0, 4.6)` `y:=gaussian(0.0, 10.0)` `z:=gaussian(0.0, 10.0)`

- A set of subdomains with probabilities by taking **Cartesian product**

$$\forall i \in x, \forall j \in y, \forall k \in z, s_{ijk} = x_i \times y_j \times z_k$$

Probabilistic Interval Subdivision

Input Space:



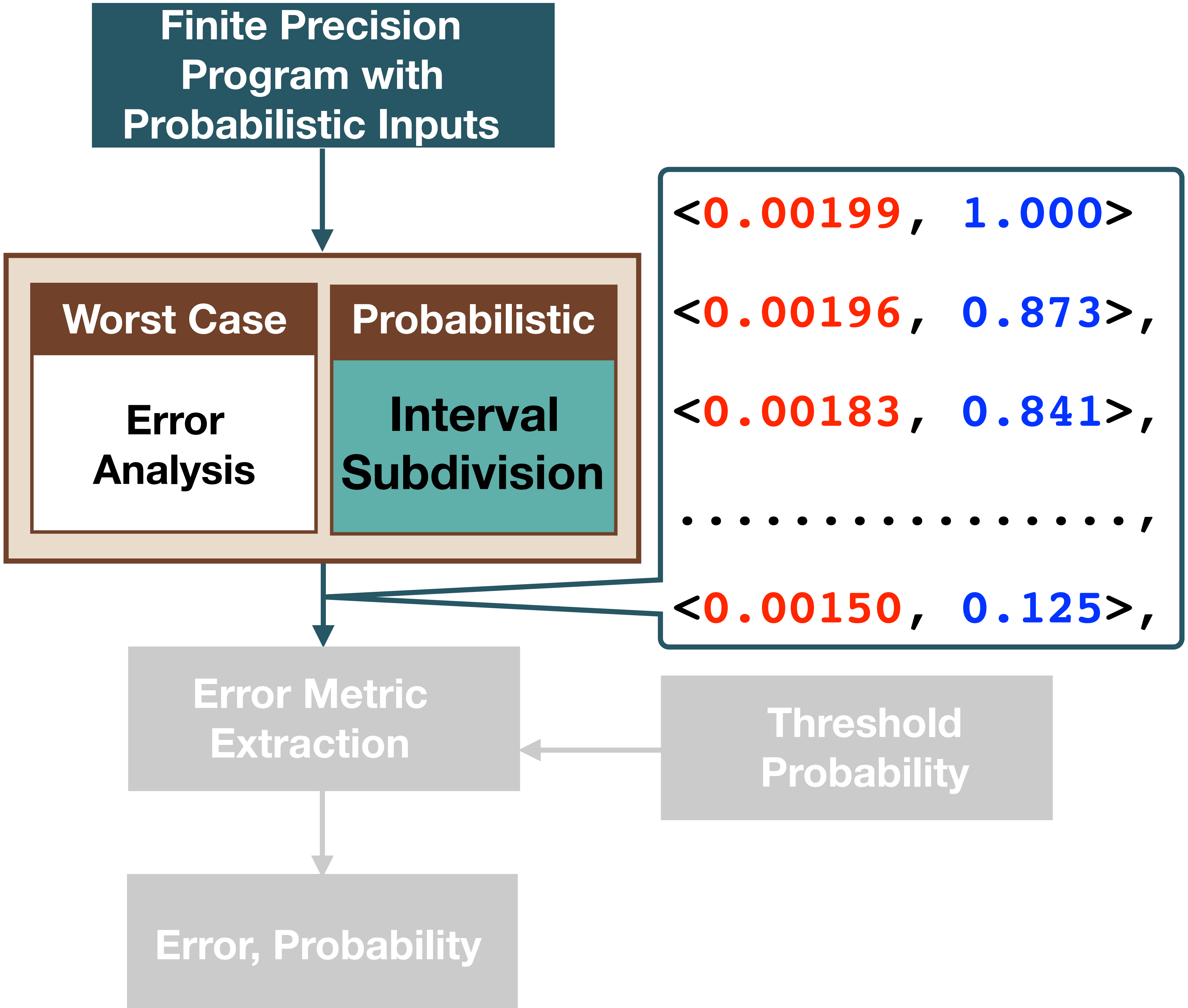
`x:=gaussian(0.0, 4.6)` `y:=gaussian(0.0, 10.0)` `z:=gaussian(0.0, 10.0)`

- A set of subdomains with probabilities by taking **Cartesian product**

$$\forall i \in x, \forall j \in y, \forall k \in z, s_{ijk} = x_i \times y_j \times z_k$$

- **Worst Case Error Analysis** for each subdomain

Computed errors and probabilities



Overview: Sound Analysis

Finite Precision Program with Probabilistic Inputs

Worst Case — Error Analysis

Probabilistic — Interval Subdivision

Error Metric Extraction

Threshold Probability

0.85

Error, Probability

largest error

<0.00199, 1.000>

<0.00196, 0.873>,

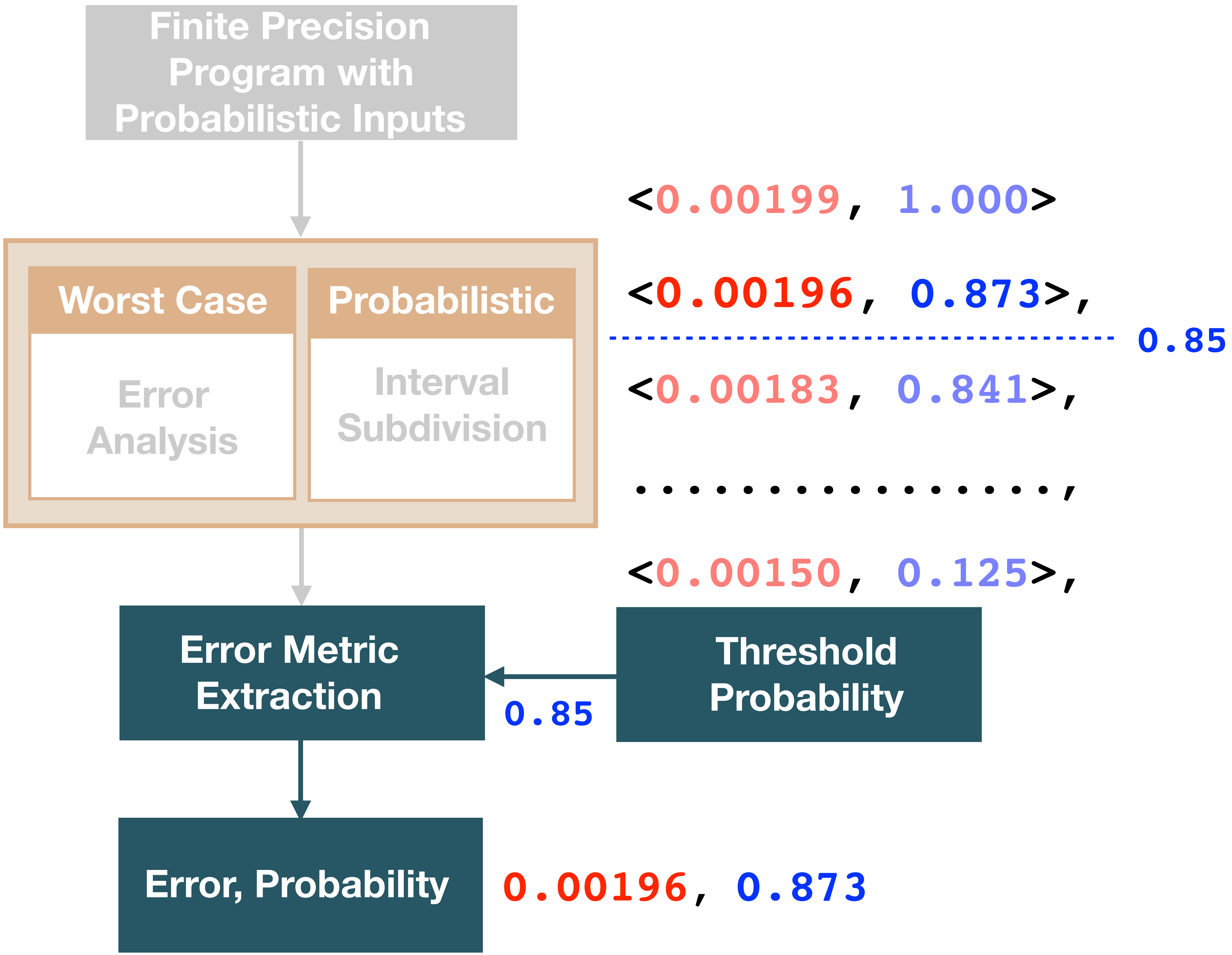
0.85

<0.00183, 0.841>,

.................,

<0.00150, 0.125>,

Overview: Sound Analysis



Results: Probabilistic Interval Subdivision

Benchmarks	**Worst Case** (state-of-the-art)
sineOrder3	4.62E-07
sqrt	1.50E-04
bspline1	2.09E-07
rigidbody2	1.94E-02
traincar2	1.37E-03
filter4	6.51E-06
cubic	1.83E-05
classIDX0	8.77E-06
polyIDX1	6.81E-04
neuron	3.22E-05

Worst case errors for 32 bit floating-point and gaussian input distributions

Results: Probabilistic Interval Subdivision

Benchmarks	**Worst Case** (state-of-the-art)	**Prob. Subdivision**
sineOrder3	4.62E-07	**2.97E-07**
sqrt	1.50E-04	**8.38E-05**
bspline1	2.09E-07	**1.96E-07**
rigidbody2	1.94E-02	**1.06E-02**
traincar2	1.37E-03	**1.32E-03**
filter4	6.51E-06	**6.09E-06**
cubic	1.83E-05	**1.73E-05**
classIDX0	8.77E-06	**7.95E-06**
polyIDX1	6.81E-04	**4.51E-04**
neuron	3.22E-05	**3.20E-05**

Reduction w.r.t. the worst case with 0.85 threshold probability for 32 bit floating-point and gaussian input distributions

Reduction using Probabilistic Interval Subdivision

Benchmarks	Worst Case (state-of-the-art)	Prob. Subdivision (% reduction)
sineOrder3	4.62E-07	**-35.7**
sqrt	1.50E-04	**-44.1**
bspline1	2.09E-07	**-6.2**
rigidbody2	1.94E-02	**-45.4**
traincar2	1.37E-03	**-3.6**
filter4	6.51E-06	**-6.5**
cubic	1.83E-05	**-5.5**
classIDX0	8.77E-06	**-9.4**
polyIDX1	6.81E-04	**-33.8**
neuron	3.22E-05	**-0.6**

Reduction w.r.t. the worst case with 0.85 threshold probability for 32 bit floating-point and gaussian input distributions

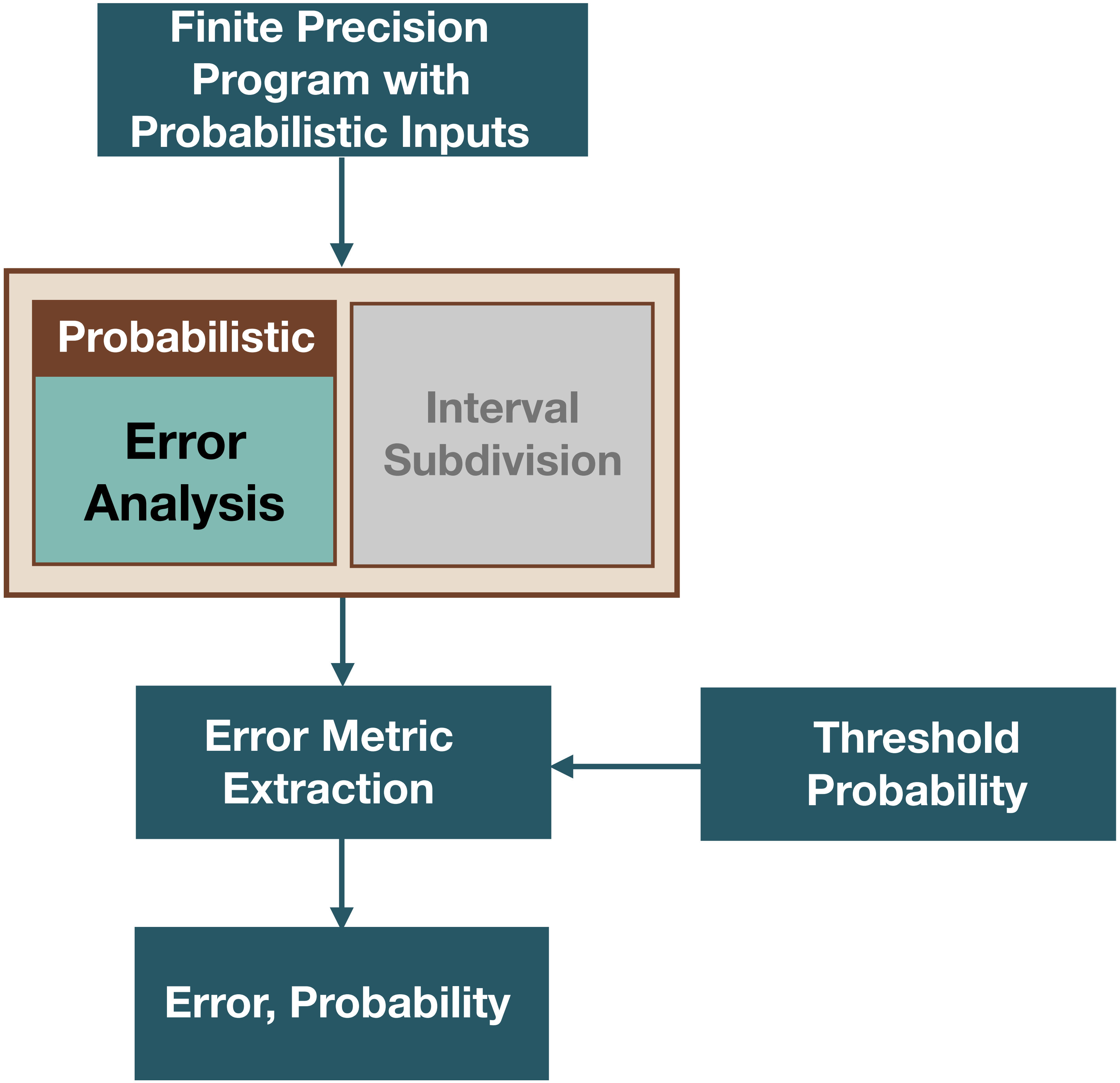
Reduction using Probabilistic Interval Subdivision

Benchmarks	Worst Case (state-of-the-art)	Prob. Subdivision (% reduction)
sineOrder3	4.62E-07	**-35.7**
sqrt	1.50E-04	**-44.1**
bspline1	2.09E-07	**-6.2**
cubic	1.83E-05	**-5.5**
classIDX0	8.77E-06	**-9.4**
polyIDX1	6.81E-04	**-33.8**
neuron	3.22E-05	**-0.6**

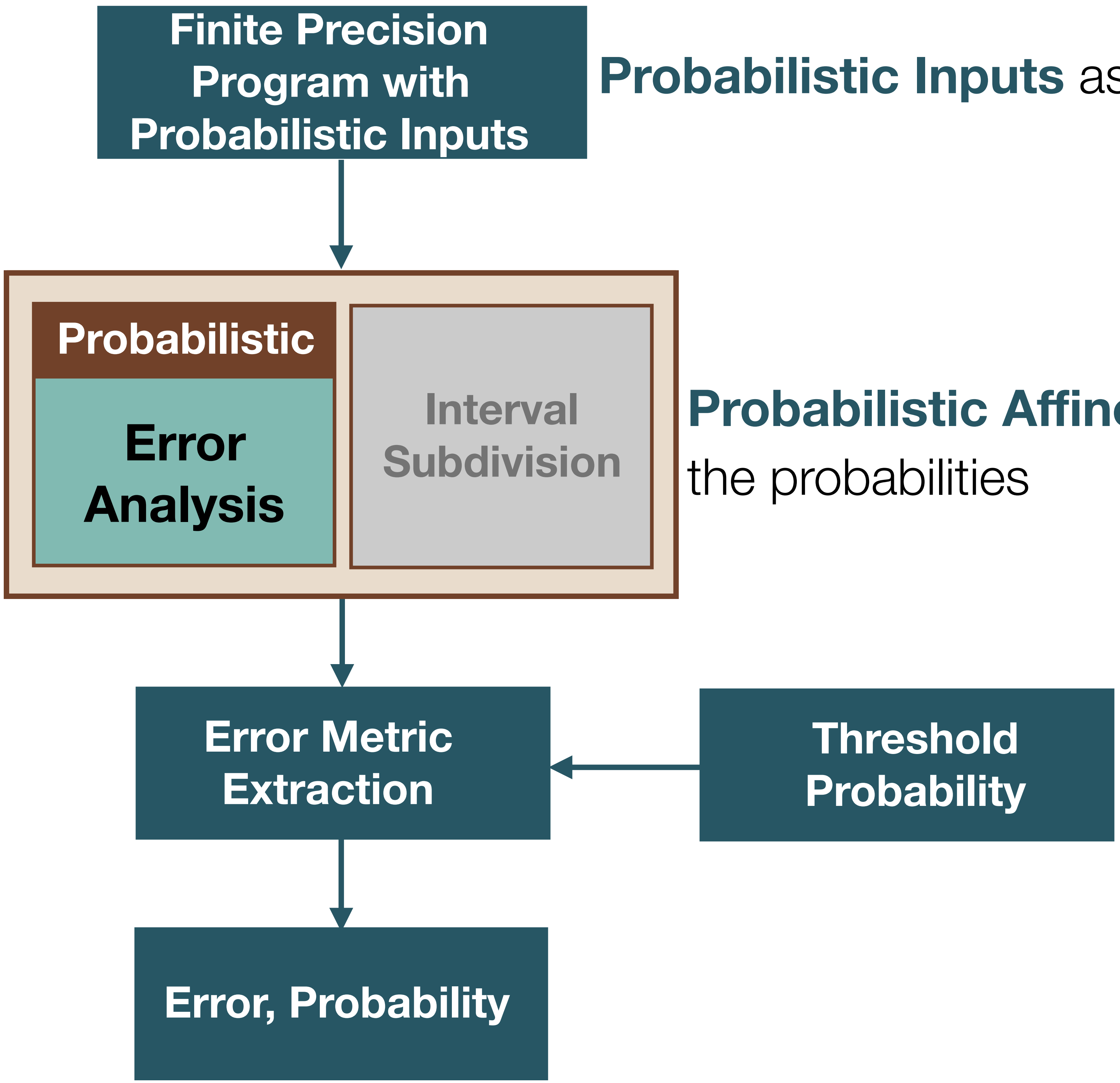
Reduction w.r.t. the worst case with 0.85 threshold probability for 32 bit floating-point and gaussian input distributions

Still computes the worst case error!

Overview: Sound Probabilistic Error Analysis



Overview: Sound Probabilistic Error Analysis



Probabilistic Inputs as uncertain probabilities

Probabilistic Affine Arithmetic propagates the probabilities

Background: Probabilistic Affine Arithmetic

- Affine Arithmetic propagates linear relations between variables
- Dependencies are tracked using shared noise symbol

$$\hat{x} := x_0 + \sum_{i=1}^{p} x_i \epsilon_i, \ \epsilon_i \in [-1,1]$$

Noise Symbol

"Static Analysis of Programs with Imprecise Probabilistic Inputs", A. Adje, O. Bouissou, J. Goubault-Larrecq, E. Goubault, S. Putot, VSTTE 2013

Background: Probabilistic Affine Arithmetic

- Affine Arithmetic propagates linear relations between variables
- Dependencies are tracked using shared noise symbol
- Keeps the probabilities while tracking dependencies

$$< [a_1, b_1], w_1 > , \cdots < [a_n, b_n], w_n >$$

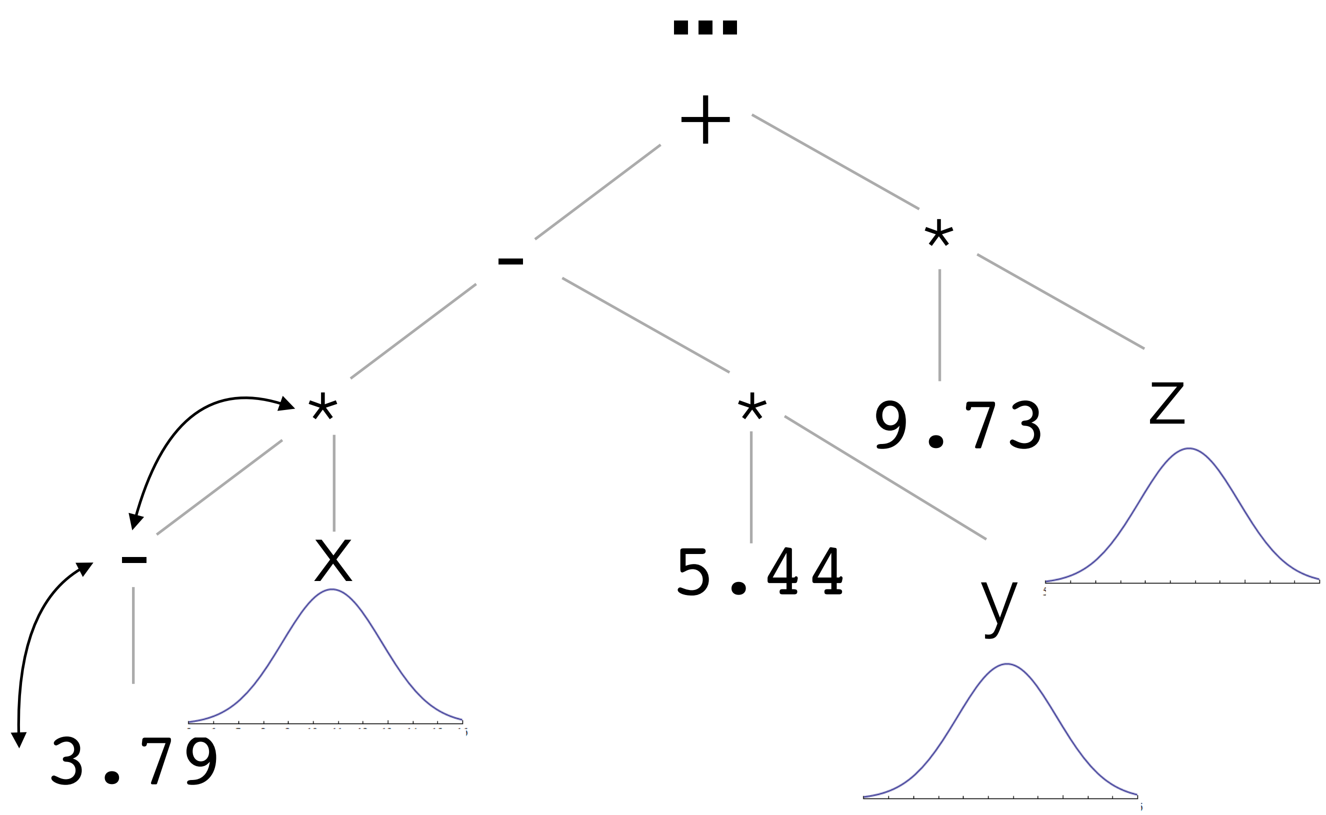
$$\hat{x} := x_0 + \sum_{i=1}^{p} x_i \epsilon_i,\ \epsilon_i \in [-1,1]$$

Noise Symbol

- Arithmetic operations are computed term wise

"Static Analysis of Programs with Imprecise Probabilistic Inputs", A. Adje, O. Bouissou, J. Goubault-Larrecq, E. Goubault, S. Putot, VSTTE 2013

Probabilistic Range Computation

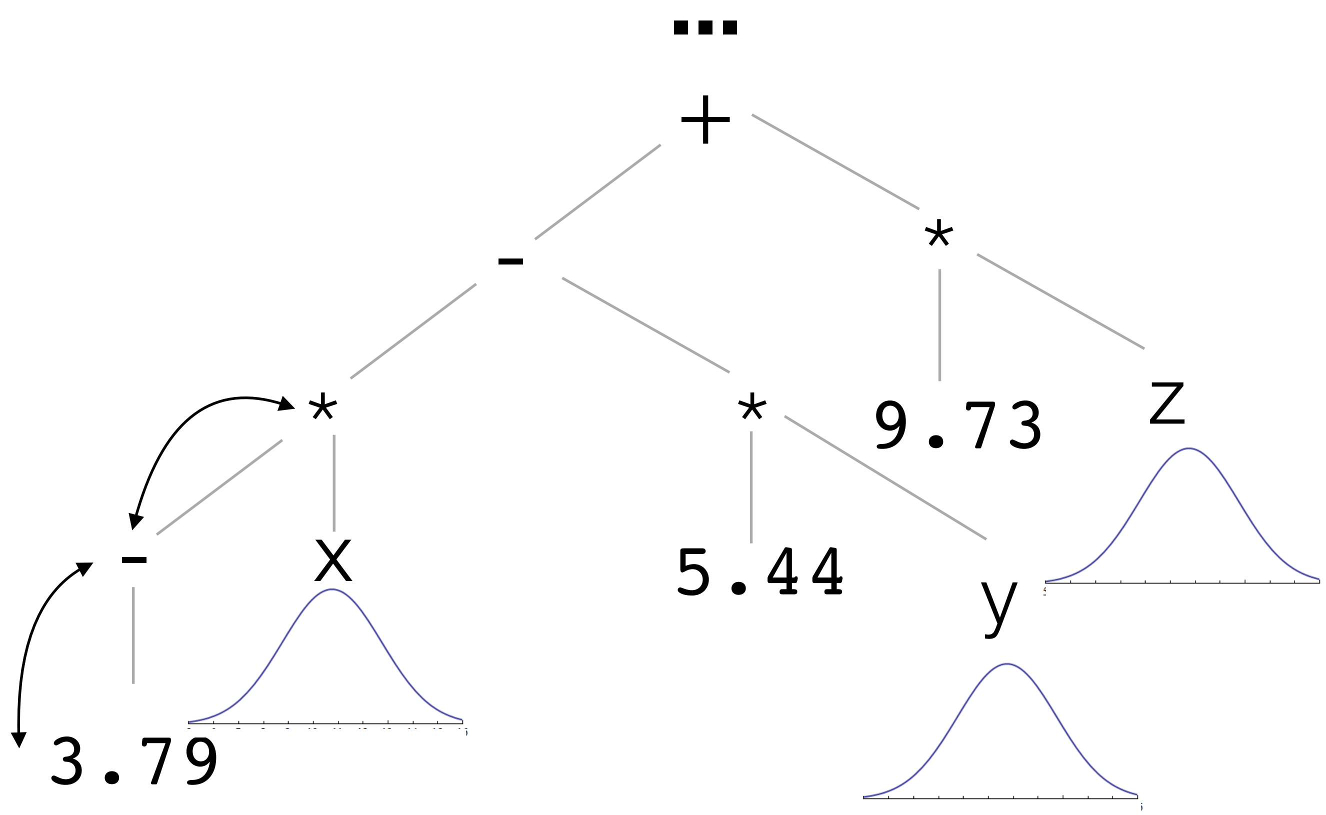


For **each arithmetic operation**

- compute range for intermediate value starting with initial distributions

"Discrete Choice in the Presence of Numerical Uncertainties", D. Lohar, E. Darulova, S. Putot, E. Goubault, EMSOFT 2018

Our Contribution: Probabilistic Error Computation



For **each arithmetic operation**

- compute range for intermediate value starting with initial distributions
- propagate existing errors
  - uses probabilistic affine arithmetic
- compute new errors
  - errors are added as fresh noise terms

Computed Error Distribution

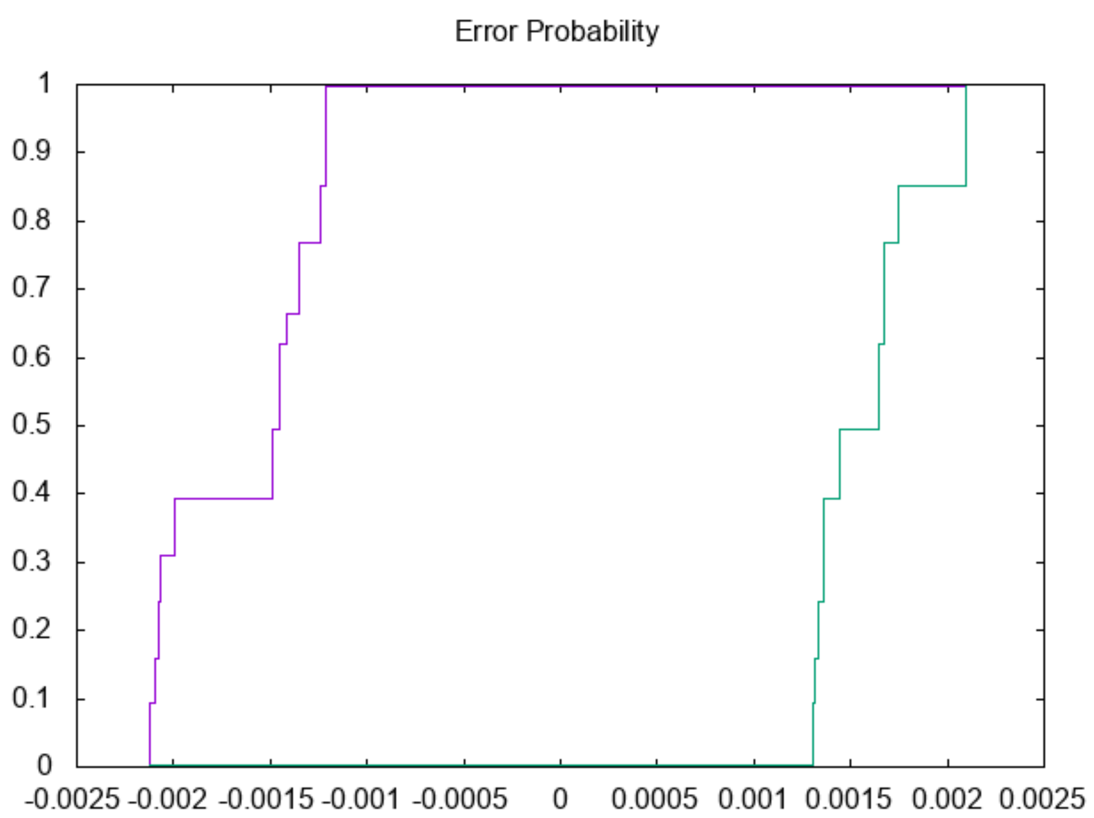
<[-0.0022, -0.0018], 1.0>,

<[-0.0018, 0.0012], 0.96>,

....

....

<[-0.0011, 0.0021], 0.14>



Generate a **set** of error **intervals** and probabilities

Computed Error Distribution

```
<[-0.0022, -0.0018], 1.0>,

<[-0.0018, 0.0012], 0.96>,
---------------------------------------------------------
.........................        Threshold probability = 0.85

.........................

<[-0.0011, 0.0021], 0.14>
```

- Repeat: remove <error, probability> if the total probability >= threshold

Computed Error Distribution

```
<[-0.0022, -0.0018], 1.0>,

<[abs(-0.0018), abs(0.0012)], 0.96>,
------------------------------------------------------------
........................

........................

<[abs(-0.0011), abs(0.0021)], 0.14>
```

Threshold probability = **0.85**

Error, Probability: **0.0021, 0.96**

- Repeat: remove <error, probability> if the total probability >= threshold
- Return the maximum error with probability

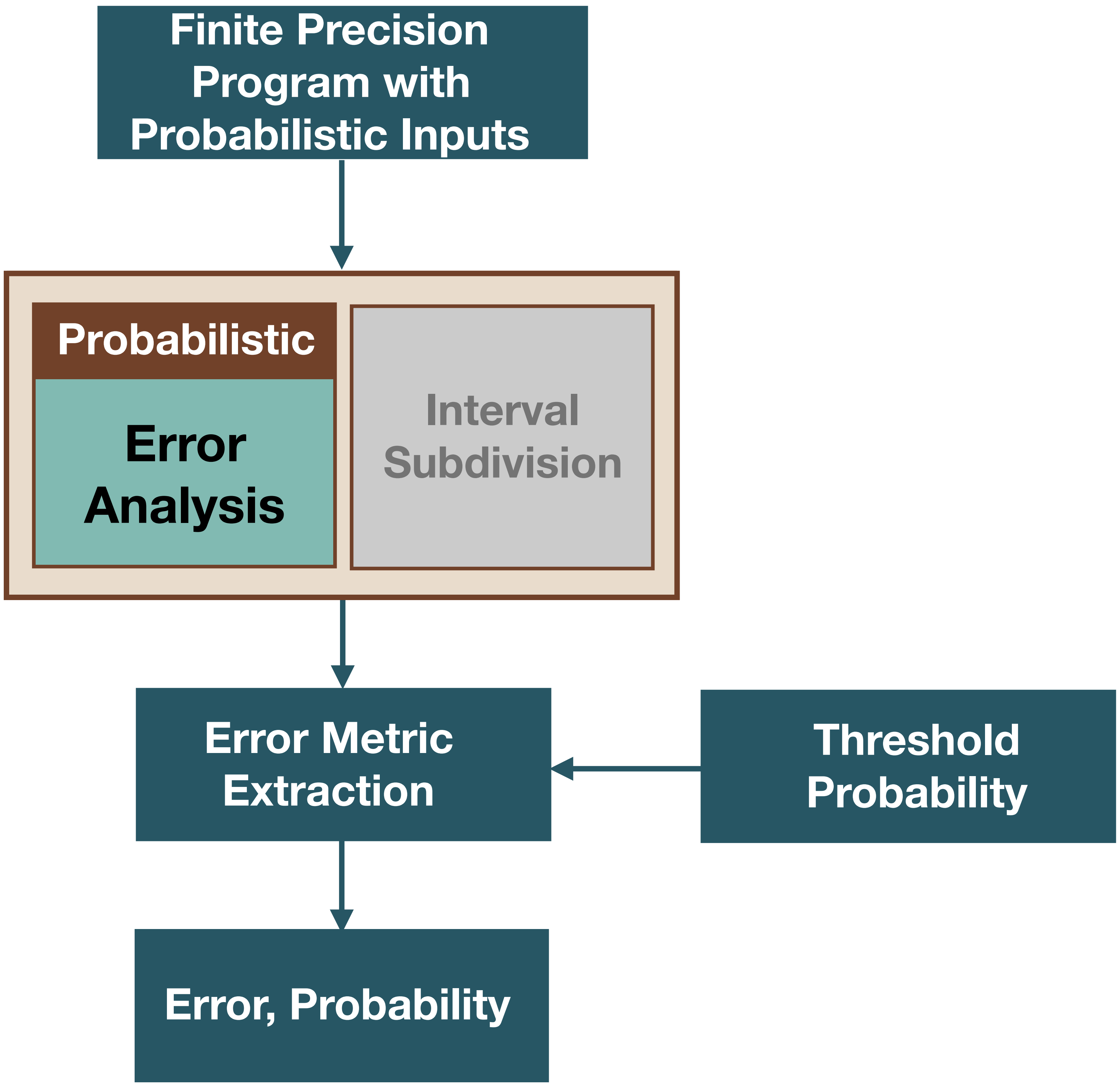
Results: Probabilistic Error Analysis

Benchmarks	Worst Case (state-of-the-art)	Prob. Subdivision (% Reduction)	Prob. Error (% Reduction)
sineOrder3	4.62E-07	-35.7	**42.2**
sqrt	1.50E-04	-44.1	**10.6**
bspline1	2.09E-07	-6.2	**84.2**
rigidbody2	1.94E-02	-45.4	**-50.0**
traincar2	1.37E-03	-3.6	**4.4**
filter4	6.51E-06	-6.5	**11.9**
cubic	1.83E-05	-5.5	**12.6**
classIDX0	8.77E-06	-9.4	**8.0**
polyIDX1	6.81E-04	-33.8	**3.9**
neuron	3.22E-05	-0.6	**>100**

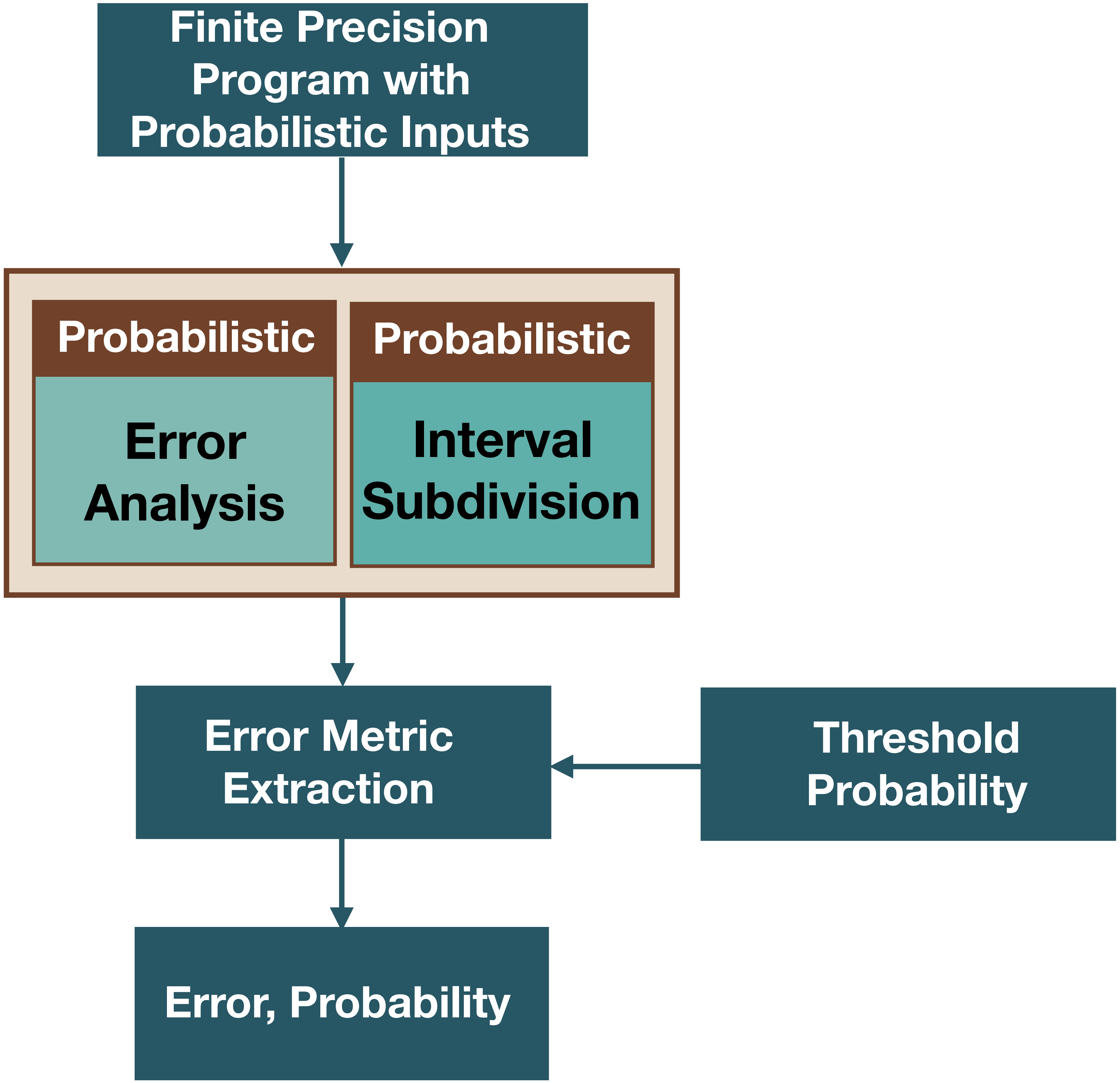
Reduction % with 0.85 threshold probability for 32 bit floating-point and gaussian input distributions

High over-approximation!

Overview: Sound Probabilistic Error Analysis



Overview: Sound Probabilistic Error Analysis



Overview: Sound Probabilistic Error Analysis

Finite Precision Program with Probabilistic Inputs

Probabilistic Error Analysis

Probabilistic Interval Subdivision

For each subdivision

- generate error distribution
- normalize with the probability of the subdomain

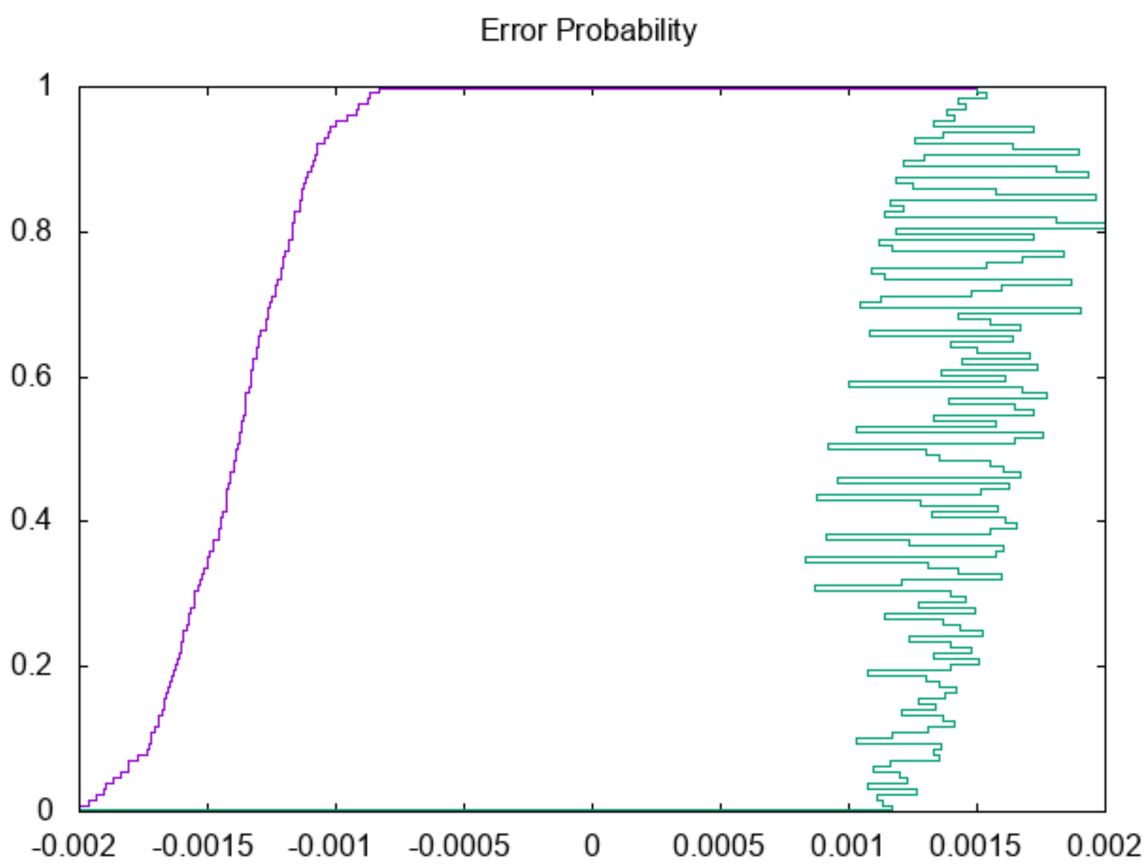
Error Metric Extraction

Threshold Probability

Error, Probability

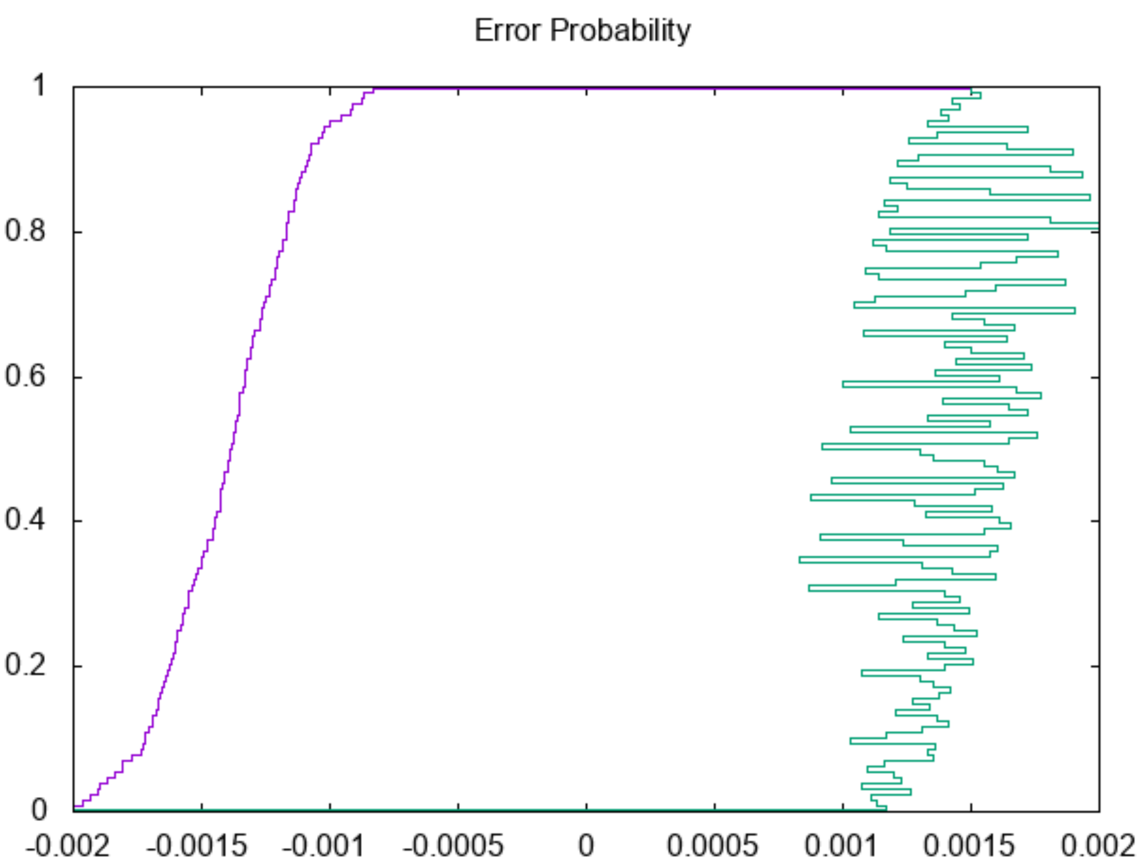
Computed Error Distribution

```
def func(..) {
 x := gaussian(0.0, 4.6)
 y := gaussian(0.0, 10.0)
 z := gaussian(0.0, 10.0)
 res = -3.79*x - 5.44*y + 9.73*z + 4.52
 return res
}
```



Error Metric Extraction

```
def func(..) {
 x := gaussian(0.0, 4.6)
 y := gaussian(0.0, 10.0)
 z := gaussian(0.0, 10.0)
 res = -3.79*x - 5.44*y + 9.73*z + 4.52
 return res
}
```



Error, Probability: **0.00176, 0.86**

Results: Prob. Error Analysis + Prob. Subdivision

Benchmarks	Worst case (state-of-the-art)	Prob. Subdivision (% Reduction)	Prob. Subdiv + Prob. Error (% Reduction)
sineOrder3	4.62E-07	-35.7	**-42.2**
sqrt	1.50E-04	**-44.1**	**-40.7**
bspline1	2.09E-07	**-6.2**	**-5.7**
rigidbody2	1.94E-02	-45.4	**-56.2**
traincar2	1.37E-03	-3.6	**-13.1**
filter4	6.51E-06	-6.5	**-23.8**
cubic	1.83E-05	**-5.5**	**3.8**
classIDX0	8.77E-06	-9.4	**-9.7**
polyIDX1	6.81E-04	**-33.8**	**-3.4**
neuron	3.22E-05	**-0.6**	**63.0**

Reduction % with 0.85 threshold probability for 32 bit floating-point and gaussian input distributions

Comparison

Benchmarks	Worst case (state-of-the-art)	Prob. Subdivision (% Reduction)	Prob. Subdiv + Prob. Error (% Reduction)
sineOrder3	4.62E-07	-35.7	**-42.2**
sqrt	1.50E-04	**-44.1**	-40.7
bspline1	2.09E-07	**-6.2**	-5.7
rigidbody2	1.94E-02	-45.4	**-56.2**
traincar2	1.37E-03	-3.6	**-13.1**
filter4	6.51E-06	-6.5	**-23.8**
cubic	1.83E-05	**-5.5**	**3.8**
classIDX0	8.77E-06	-9.4	**-9.7**
polyIDX1	6.81E-04	**-33.8**	-3.4
neuron	3.22E-05	**-0.6**	**63.0**

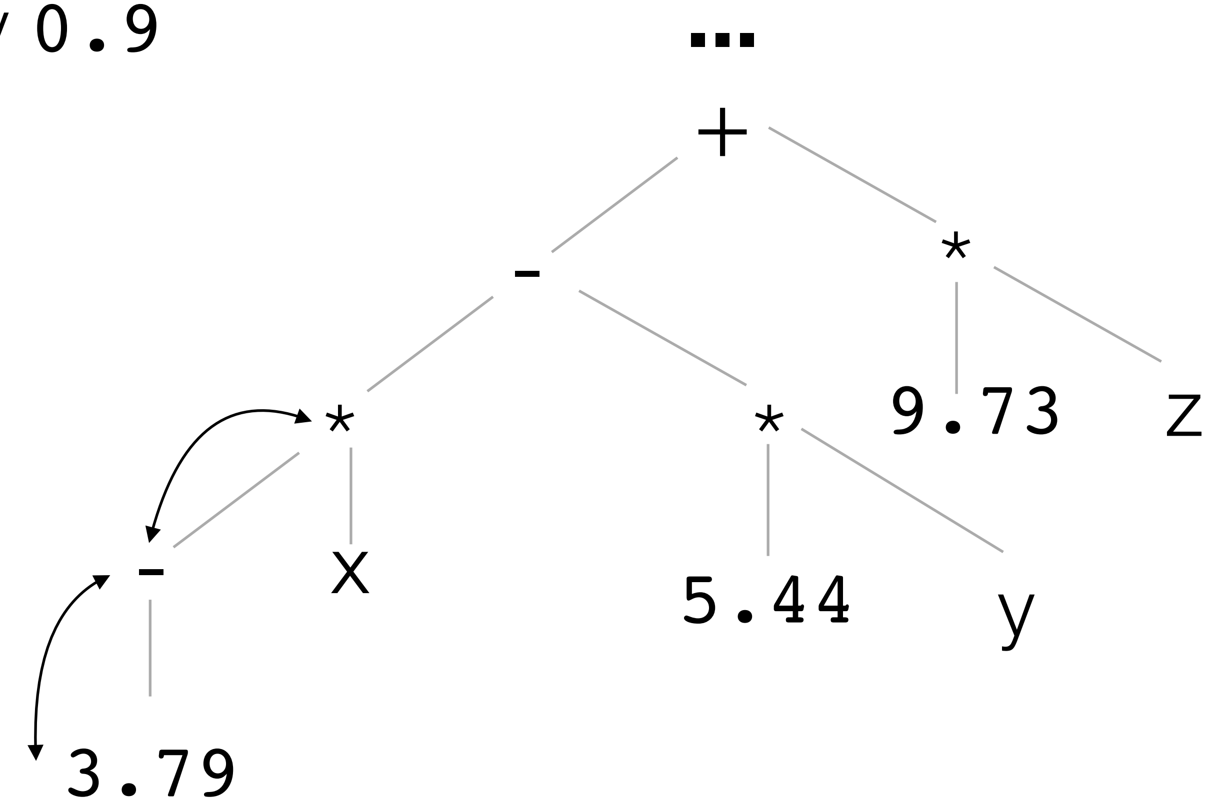
Reduction % with 0.85 threshold probability for 32 bit floating-point and gaussian input distributions

Performance depends on the application

What if we have **Approximate Hardware** with **Probabilistic Error Specifications?**

Probabilistic Error Specification

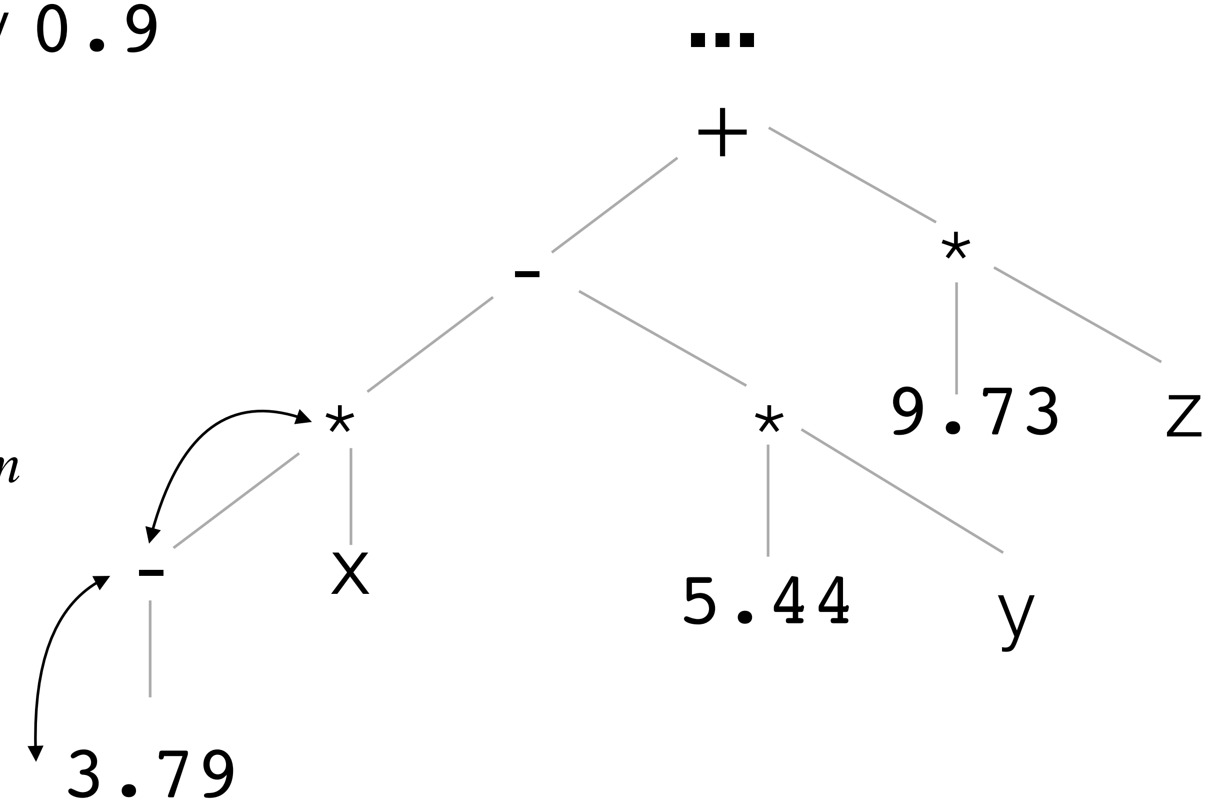
- Error: $4 \times \epsilon_m$, probability: `0.1`
- Error: $1 \times \epsilon_m$ with probability `0.9`



Worst Case Error Analysis

- Error: $4 \times \epsilon_m$, probability: `0.1`
- Error: $1 \times \epsilon_m$ with probability `0.9`

Worst-case: `3.79 +/-` $4 \times \epsilon_m$



- Worst Case Error Analysis can't utilize the probabilistic specification
- Not resource efficient

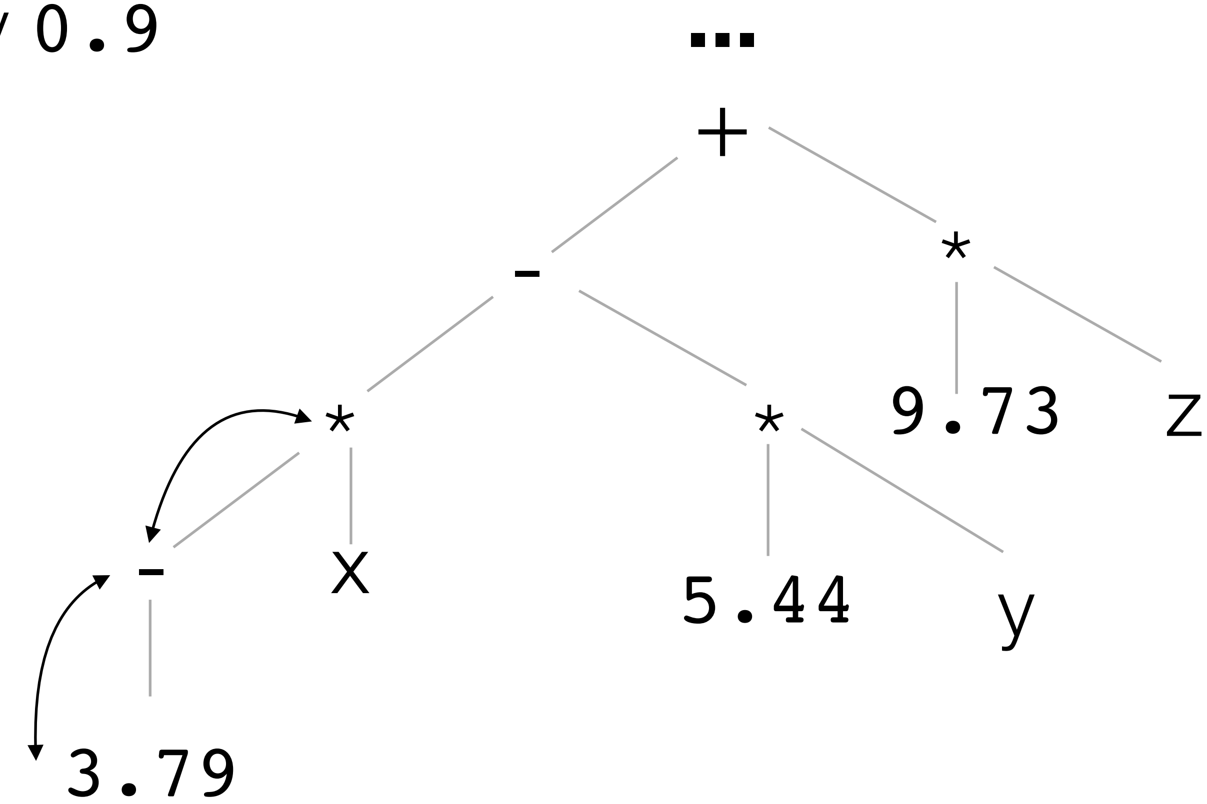
Probabilistic Error Analysis

- Error: $4 \times \epsilon_m$, probability: `0.1`
- Error: $1 \times \epsilon_m$ with probability `0.9`

Probabilistic error:

<[3.79 +/- ϵ_m], 0.9>,

<[3.79 +/- $4 \times \epsilon_m$], 0.1>



- Utilizes probabilistic error spec with probabilistic analysis
- Compute multiple errors for each operation

Results: Probabilistic Error Specification

Benchmarks	Prob Analysis + Prob Subdiv (100 subdivisions)
sineOrder3	-52.9%
sqrt	-56.6%
bspline1	-40.2%
rigidbody2	-13.5%
traincar2	-13.6%
filter4	-47.5%
cubic	-41.9%
classIDX0	-18.7%
polyIDX1	-10.6%
neuron	-41.7%

Reduction % with 0.85 threshold probability for 32 bit floating-point errors, gaussian input distributions considering $4 \times \epsilon_m$ error happens with 0.1 probability

Results: Probabilistic Error Specification

Benchmarks	Prob Analysis + Prob Subdiv (100 subdivisions)	Worst Case + Prob Subdiv (200 subdivisions)
sineOrder3	**-52.9%**	-34.0%
sqrt	**-56.6%**	-45.8%
bspline1	**-40.2%**	-9.7%
rigidbody2	-13.5%	**-49.2%**
traincar2	**-13.6%**	-1.9%
filter4	**-47.5%**	-10.4%
cubic	**-41.9%**	-9.3%
classIDX0	**-18.7%**	-13.6%
polyIDX1	-10.6%	**-37.3%**
neuron	**-41.7%**	-13.9%

Reduction % with 0.85 threshold probability for 32 bit floating-point errors, gaussian input distributions considering $4 \times \epsilon_m$ error happens with 0.1 probability

More in the paper

- Technical details of the probabilistic method
- Alternative approach to compute the error metric
- Case studies from embedded systems and machine learning
- More experiments with
  - uniform distribution of inputs
  - different error specifications

"Sound Probabilistic Numerical Error Analysis"

D. Lohar, M. Prokop, and E. Darulova

https://github.com/malyzajko/daisy/tree/probabilistic

Conclusion

- The first **Sound Analysis** of **Probabilistic Errors**
- **Interpretation** of the **error distribution** usable in real world
- Usage in applications with **Probabilistic Error Specification**