A Dynamic Program Analysis to find Floating-Point Accuracy Problems

Florian Benz
fbenz@stud.uni-saarland.de

Andreas Hildebrandt
andreas.hildebrandt@uni-mainz.de

Sebastian Hack
hack@cs.uni-saarland.de

PLDI 2012, Beijing
June 13, 2012

UNIVERSITÄT
DES
SAARLANDES

VISUAL
COMPUTING
INSTITUTE

Introduction

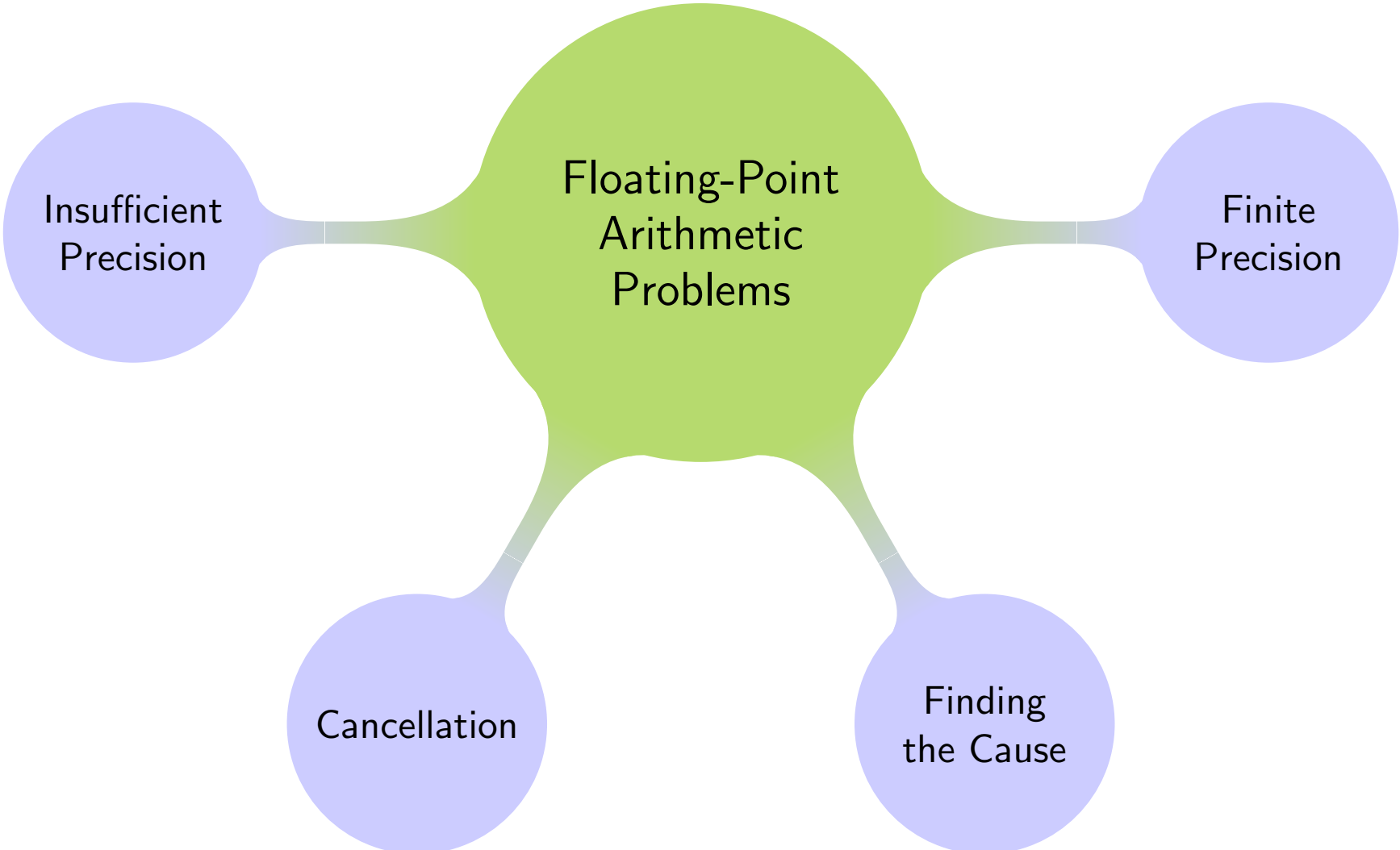
- Floating-point arithmetic is ubiquitous
  - Almost every language has a floating-point data type
  - Most PCs and supercomputers have floating-point accelerators
- Not well understood by most developers

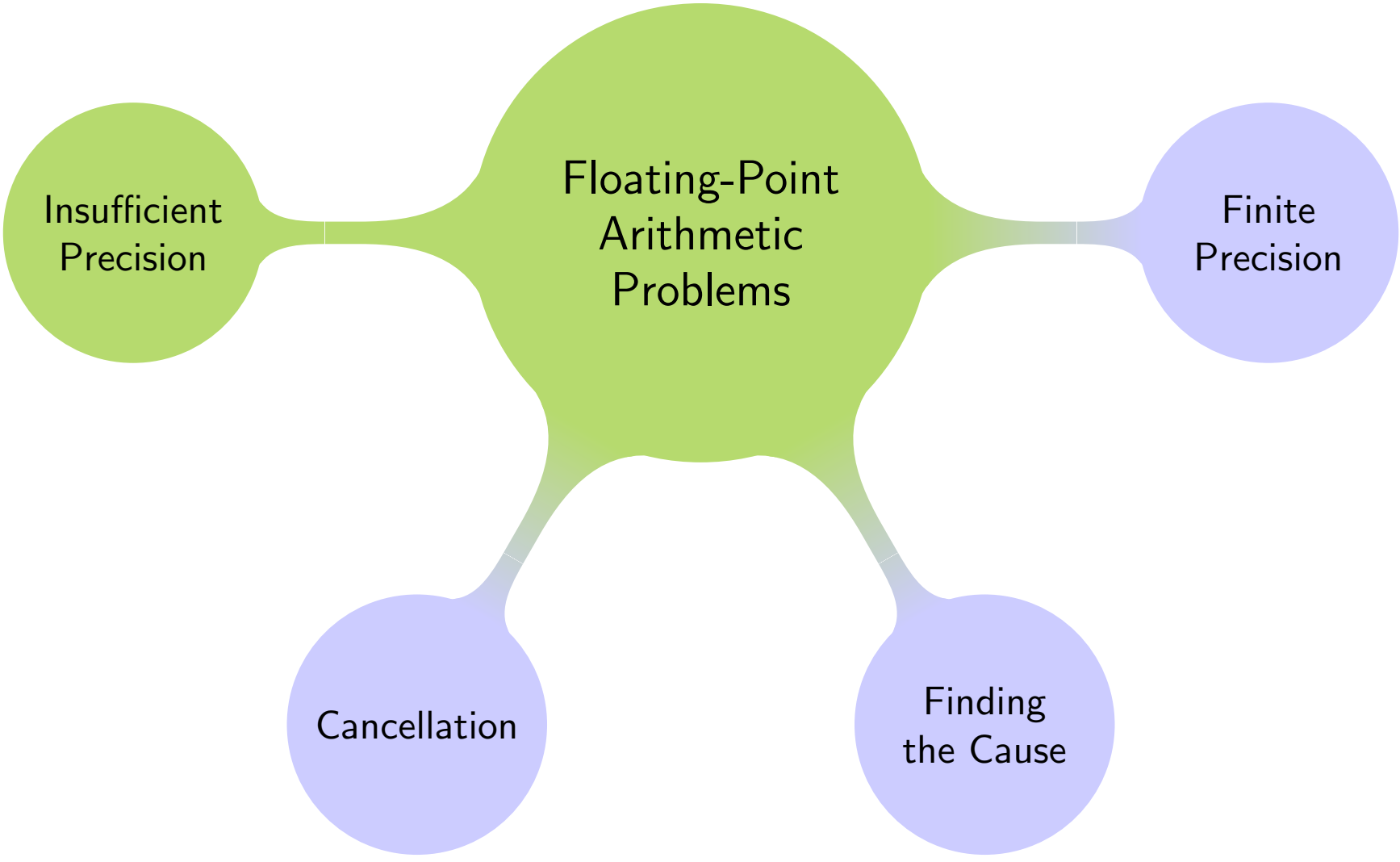
Introduction

- Floating-point arithmetic is ubiquitous
  - Almost every language has a floating-point data type
  - Most PCs and supercomputers have floating-point accelerators
- Not well understood by most developers

Our Contribution

A dynamic program analysis that assists developers in understanding and tracking down floating-point arithmetic issues in real-world programs.


Floating-Point
Arithmetic
Problems
Insufficient
Precision
Finite
Precision
Cancellation
Finding
the Cause


Floating-Point
Arithmetic
Problems
Insufficient
Precision
Finite
Precision
Cancellation
Finding
the Cause

Problem: Insufficient Precision

```
float e = 0.00000005f;
float sum = 1.0f;
int i;
for (i = 0; i < 5; i++) {
    sum += e;
}
```

- Finally $\texttt{sum} = 1.0$

Problem: Insufficient Precision

```
float e = 0.00000005f;
float sum = 1.0f;
int i;
for (i = 0; i < 5; i++) {
    sum += e;
}
```

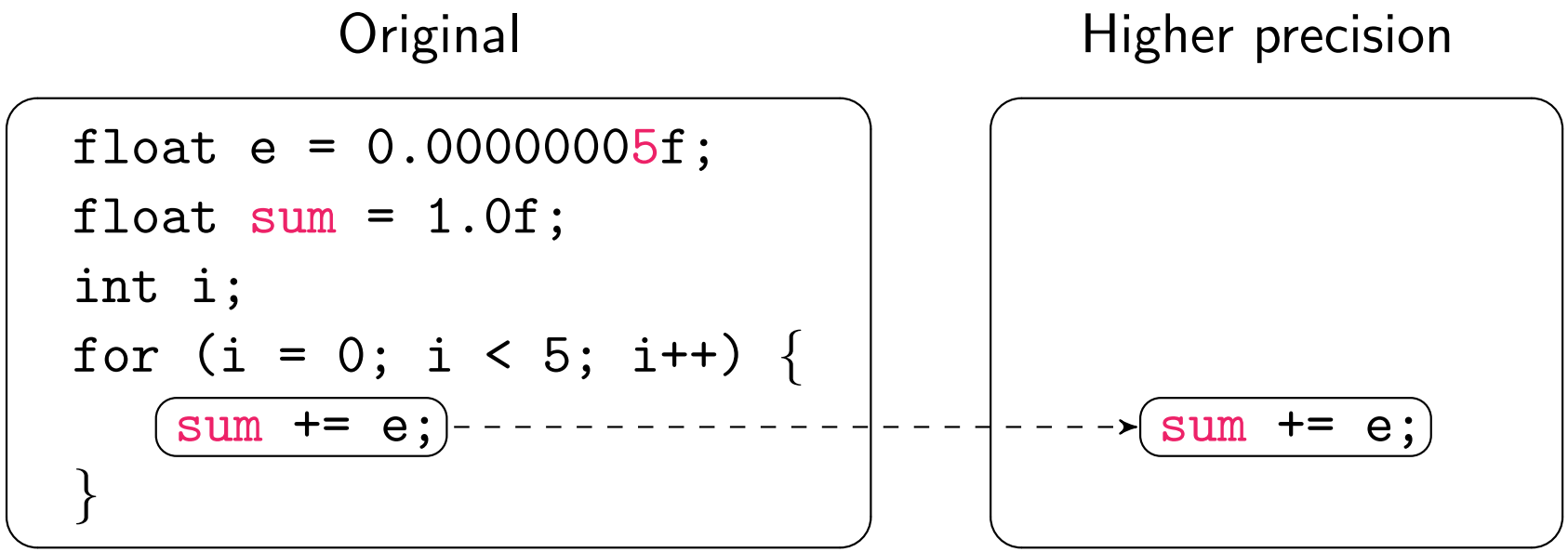
- Finally `sum` $= 1.0$
- Higher precision yields `sum` $= 1.00000025$
- Single precision machine epsilon: `0.00000006f`

Solution: Side by Side in Higher Precision

Original

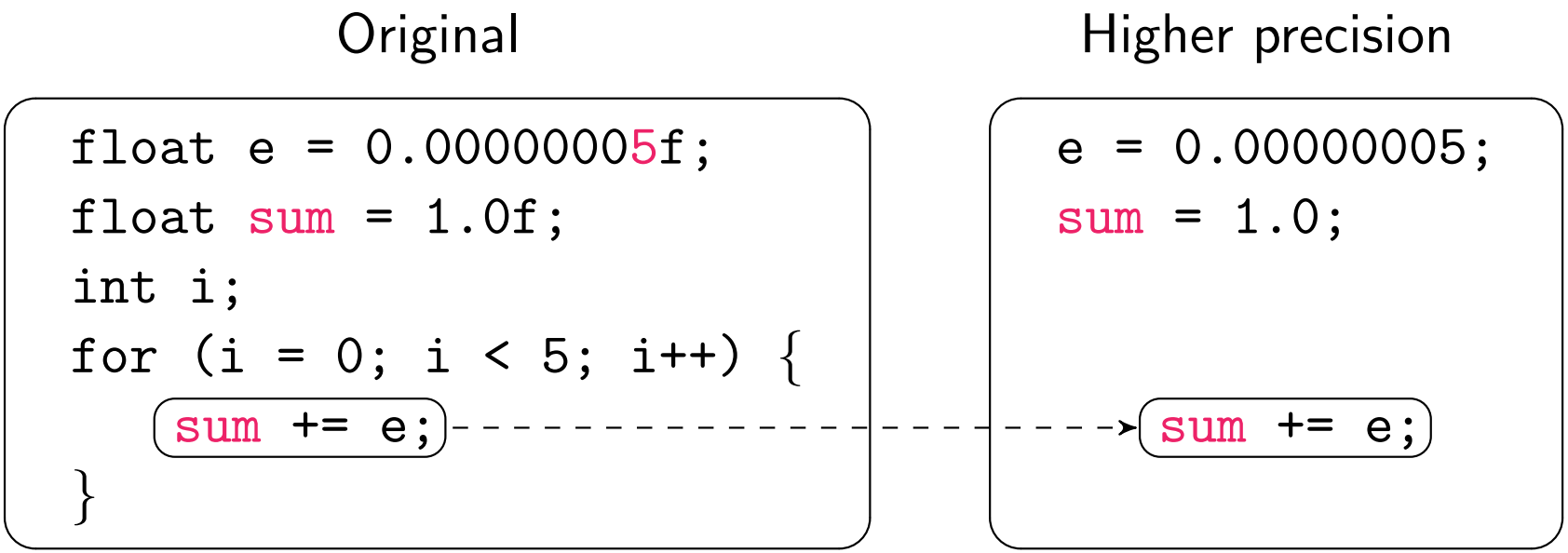
```
float e = 0.00000005f;
float sum = 1.0f;
int i;
for (i = 0; i < 5; i++) {
    sum += e;
}
```

Solution: Side by Side in Higher Precision



- Side-by-side computation in higher precision

Solution: Side by Side in Higher Precision



- Side-by-side computation in higher precision
- Shadowing every floating-point value

Solution: Side by Side in Higher Precision

Original

```
float e = 0.00000005f;
float sum = 1.0f;
int i;
for (i = 0; i < 5; i++) {
    sum += e;
}
```

Higher precision

```
e = 0.00000005;
sum = 1.0;

sum += e;
```

Iteration	Single precision	Higher precision
0	1.0	1.00000000
1	1.0	1.00000005
2	1.0	1.00000010
3	1.0	1.00000015
4	1.0	1.00000020
5	1.0	1.00000025

Error Measurement

$$\text{relative error} = \left| \frac{\text{exact value} - \text{approximate value}}{\text{exact value}} \right|$$

Error Measurement

$$\text{relative error} = \left| \frac{\text{exact value} - \text{approximate value}}{\text{exact value}} \right|$$

- Approximate exact value with higher precision value

Error Measurement

$$\text{relative error} = \left| \frac{\text{exact value} - \text{approximate value}}{\text{exact value}} \right|$$

- Approximate exact value with higher precision value

$$\left| \frac{1.00000025 - 1.0}{1.00000025} \right| = 2.5 \times 10^{-7}$$

Error Measurement

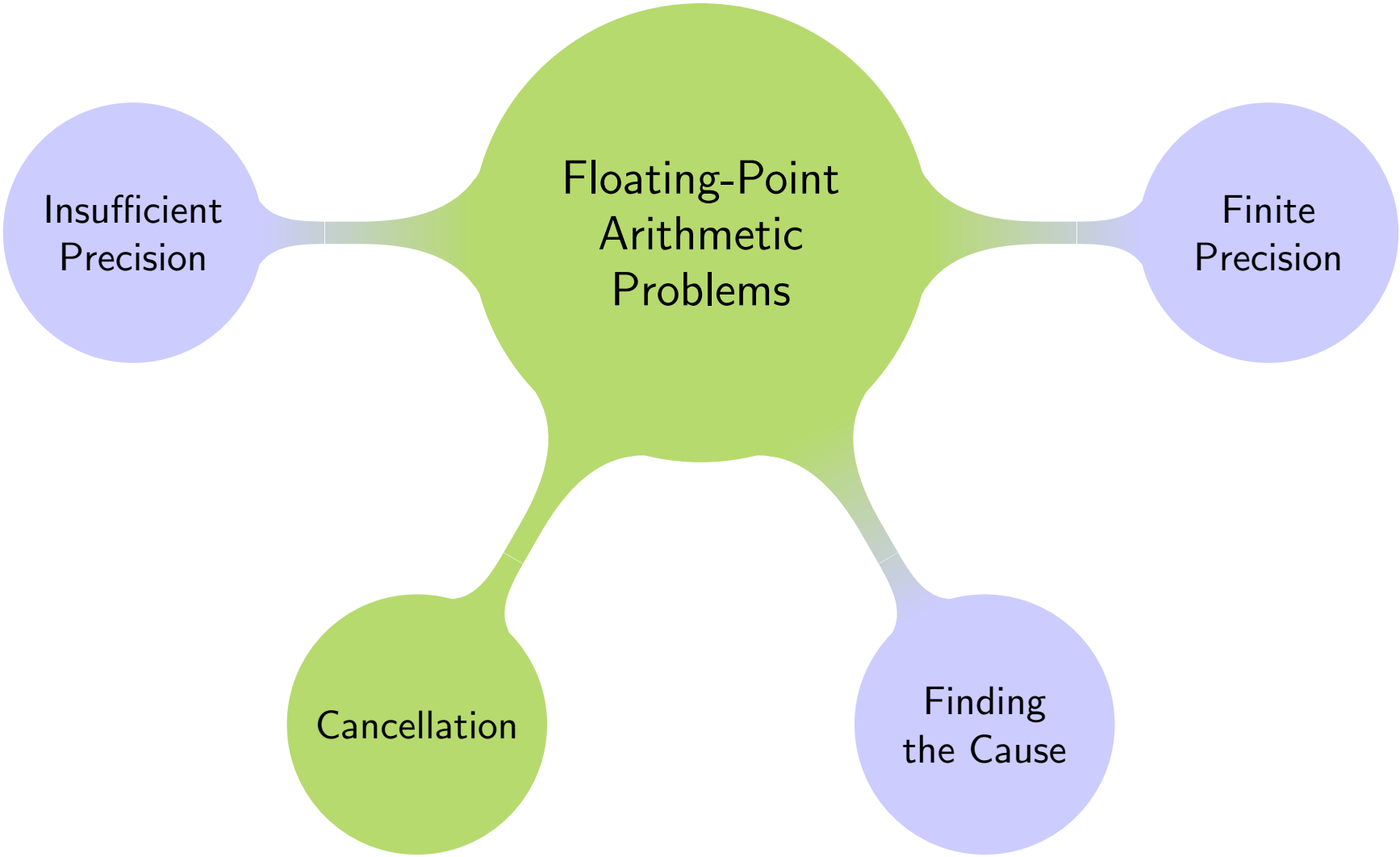
$$\text{relative error} = \left| \frac{\text{exact value} - \text{approximate value}}{\text{exact value}} \right|$$

- Approximate exact value with higher precision value

$$\left| \frac{1.00000025 - 1.0}{1.00000025} \right| = 2.5 \times 10^{-7}$$

- Relative errors smaller than machine epsilon are unavoidable

"float"	$2^{-24} \approx 5.96 \times 10^{-8}$
"double"	$2^{-53} \approx 1.11 \times 10^{-16}$


Floating-Point
Arithmetic
Problems
Insufficient
Precision
Finite
Precision
Cancellation
Finding
the Cause

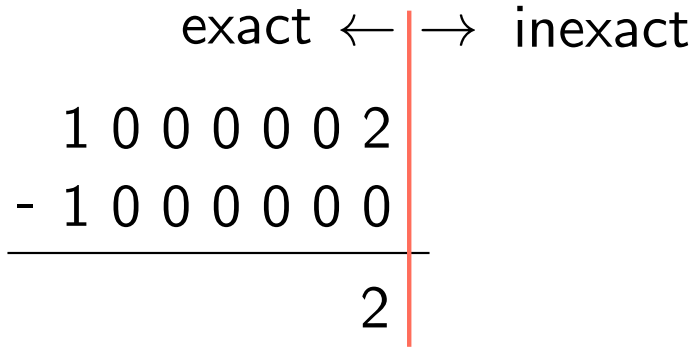
Problem: Cancellation

- Benign

$$
\begin{array}{r}
1\,0\,0\,0\,0\,0\,2 \\
-\,1\,0\,0\,0\,0\,0\,0 \\
\hline
2
\end{array}
$$

Problem: Cancellation

- Benign



Problem: Cancellation

- Benign

exact $\leftarrow$	$\rightarrow$ inexact	relative error
1 0 0 0 0 0 2		0
- 1 0 0 0 0 0 0		0
2		0

Problem: Cancellation

- Benign

exact ←	→ inexact	relative error
1 0 0 0 0 0 2		0
- 1 0 0 0 0 0 0		0
2		0

- Catastrophic

exact ←	→ inexact
1 0 0 0 0 0	3
- 1 0 0 0 0 0	0
	3

Problem: Cancellation

- Benign

exact ←	→ inexact	relative error
1 0 0 0 0 0 2		0
- 1 0 0 0 0 0 0		0
2		0

- Catastrophic

exact ←	→ inexact	relative error
1 0 0 0 0 0 3		10^{-6}
- 1 0 0 0 0 0 0		0
3		0.5

Solution: Cancellation Badness

- Benign

exact ←	→ inexact	relative error
1 0 0 0 0 0 2		0
- 1 0 0 0 0 0 0		0
2		0

$$6\ (\text{canceled})\ -7\ (\text{exact}) + 1 = 0$$

Solution: Cancellation Badness

- Benign

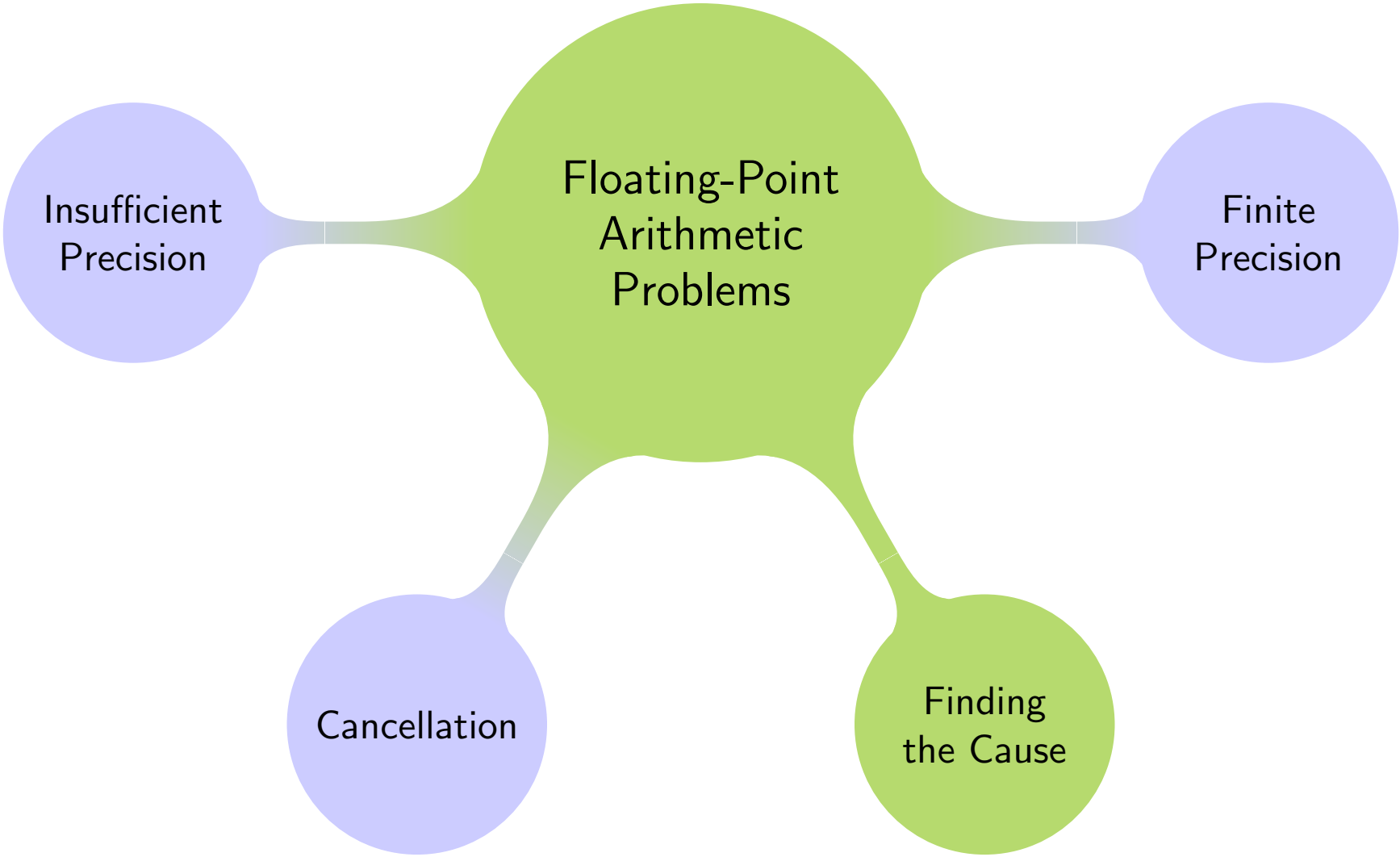
exact ←	→ inexact	relative error
1 0 0 0 0 0 2		0
- 1 0 0 0 0 0 0		0
2		0

$$6\ (\text{canceled})\ -7\ (\text{exact}) + 1 = 0$$

- Catastrophic

exact ←	→ inexact	relative error
1 0 0 0 0 0 3		10^{-6}
- 1 0 0 0 0 0 0		0
3		0.5

$$6\ (\text{canceled})\ -6\ (\text{exact}) + 1 = 1$$


Floating-Point
Arithmetic
Problems
Insufficient
Precision
Finite
Precision
Cancellation
Finding
the Cause

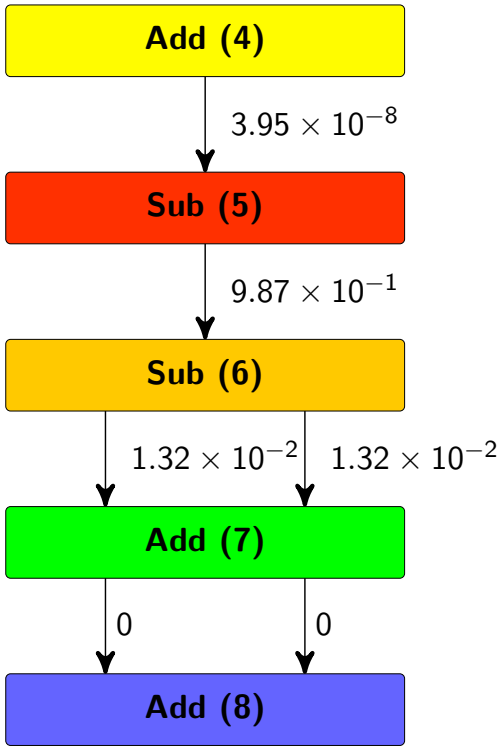
Problem: Finding the Cause

```
1  float e = 0.00000006f;
2  float x = 0.5f;
3  float y = 1.0f + x;
4  float more = y + e;
5  float diff_e = more - y;
6  float diff_0 = diff_e - e;
7  float zero = diff_0 + diff_0;
8  float result = 2 * zero;
```

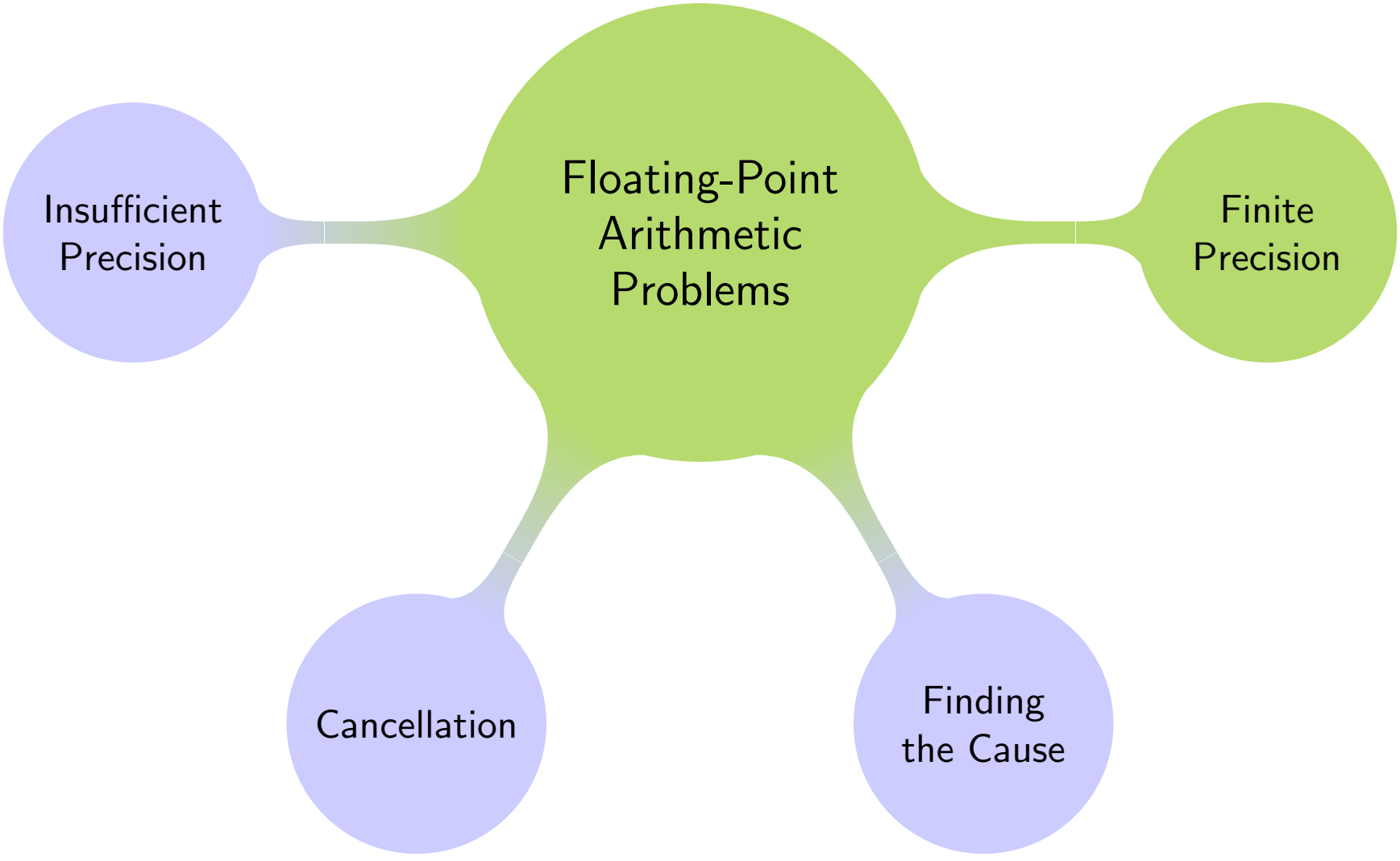
- `result` $= 2.37 \times 10^{-7}$
- Higher precision yields `result` $= 0$

Solution: Light-Weight Slicing

```
1  float e = 0.00000006f;
2  float x = 0.5f;
3  float y = 1.0f + x;
4  float more = y + e;
5  float diff_e = more - y;
6  float diff_0 = diff_e - e;
7  float zero = diff_0 + diff_0;
8  float result = 2 * zero;
```



- `result` $= 2.37 \times 10^{-7}$
- Higher precision yields `result` $= 0$


Floating-Point
Arithmetic
Problems
Insufficient
Precision
Finite
Precision
Cancellation
Finding
the Cause

Problem: Finite Precision[1]

$$u_n = \begin{cases} 2 & \text{if } n = 0, \\ -4 & \text{if } n = 1, \\ 111 - \frac{1130}{u_{n-1}} + \frac{3000}{u_{n-1}u_{n-2}} & \text{if } n > 1. \end{cases}$$

- Mathematically correct

$$\lim_{n \to \infty} u_n = 6$$

[1] Muller et al.: "Handbook of Floating-Point Arithmetic", Birkhäuser, 2010

Problem: Finite Precision[1]

$$u_n = \begin{cases} 2 & \text{if } n = 0, \\ -4 & \text{if } n = 1, \\ 111 - \frac{1130}{u_{n-1}} + \frac{3000}{u_{n-1}u_{n-2}} & \text{if } n > 1. \end{cases}$$

- Mathematically correct

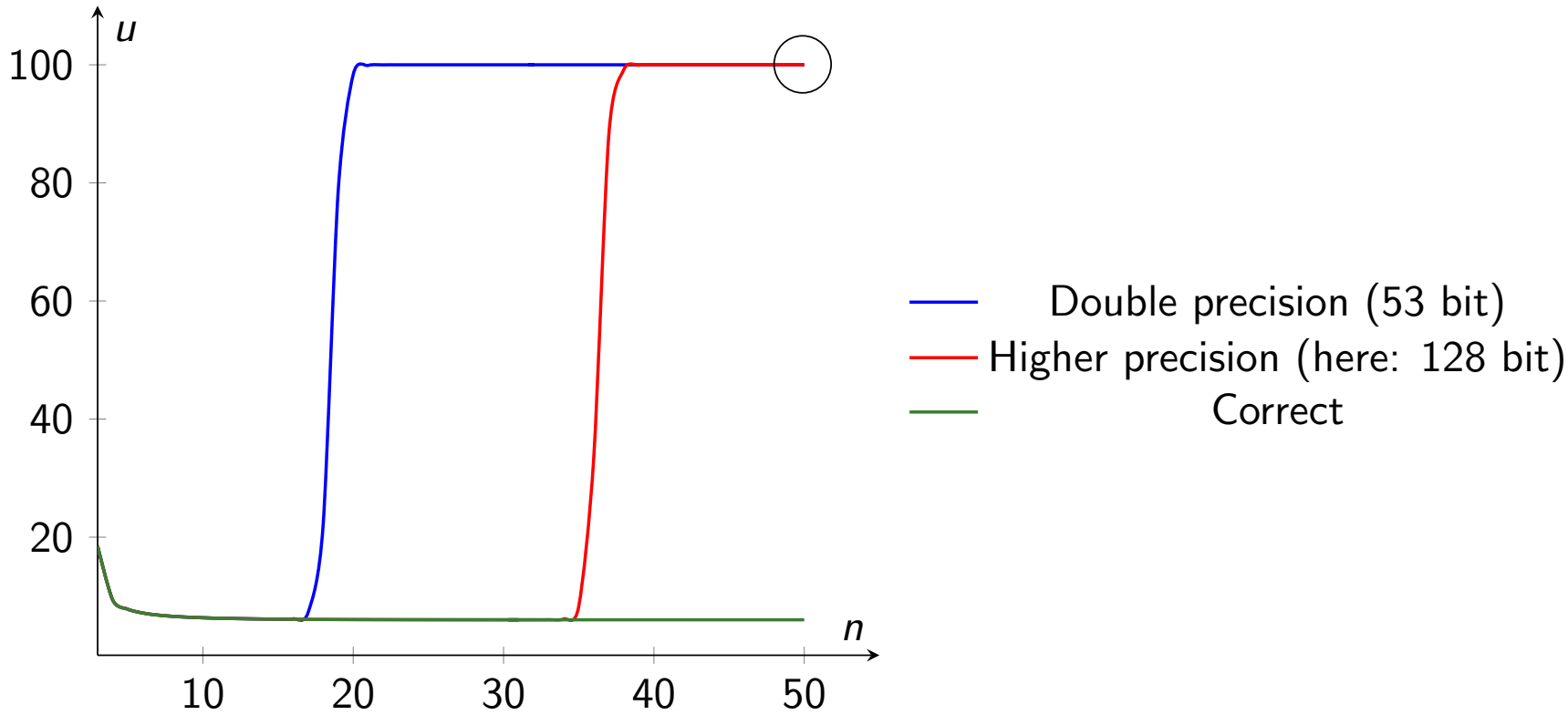
$$\lim_{n\to\infty} u_n = 6$$

- For all finite precisions

$$\lim_{n\to\infty} u_n = 100$$

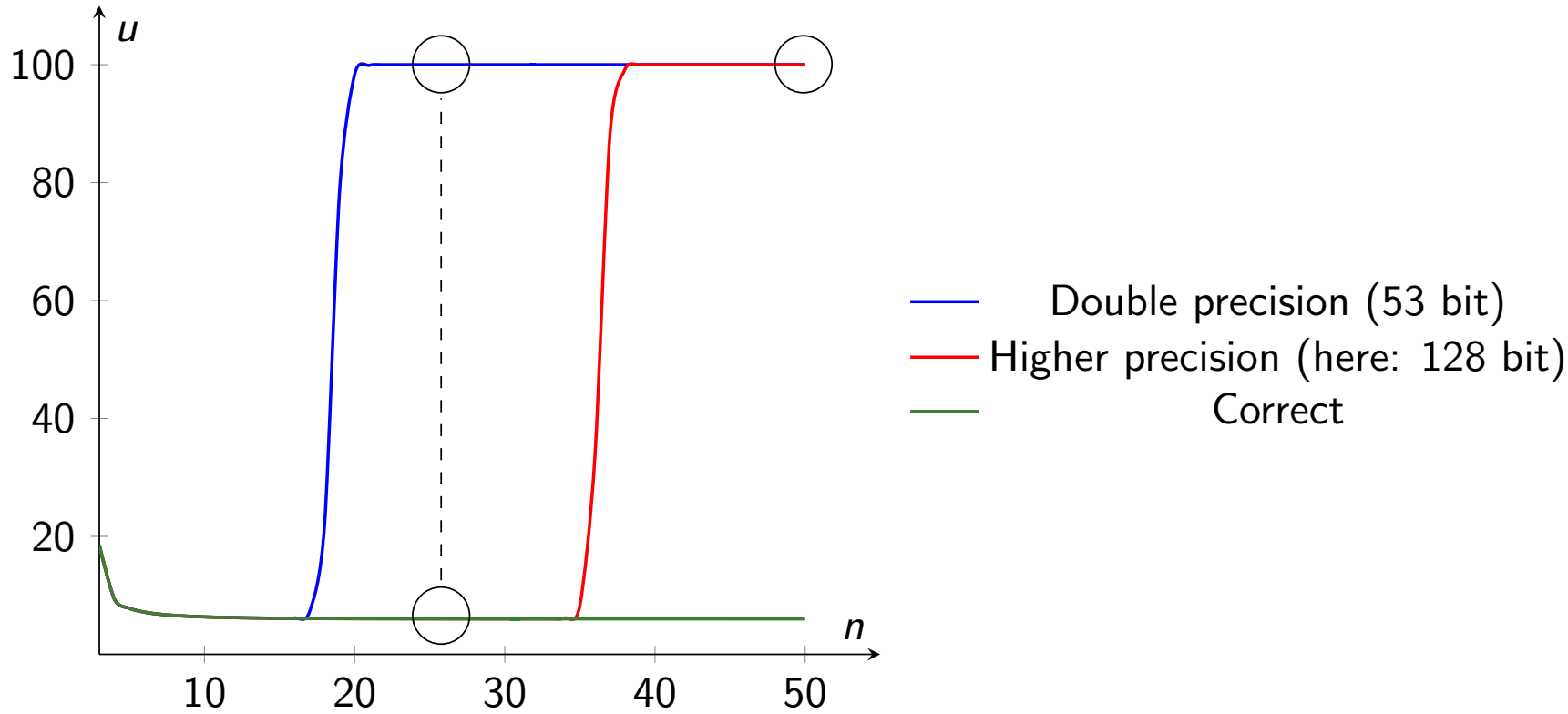
[1] Muller et al.: "Handbook of Floating-Point Arithmetic", Birkhäuser, 2010

Analysis of the Problem



- Fully automatic analysis detects no error

Analysis of the Problem



- Fully automatic analysis detects no error
- Can only be discovered with intermediate results

Solution: Stages

```
int i;
double u, v, w;

u = 2;
v = -4;

for (i = 3; i <= 50; i++) {


    w = 111.  - 1130./v + 3000./(v*u);
    u = v;
    v = w;


}
```

Solution: Stages

```
int i;
double u, v, w;

u = 2;
v = -4;

for (i = 3; i <= 50; i++) {
    FPDEBUG_BEGIN_STAGE(0);

    w = 111.  - 1130./v + 3000./(v*u);
    u = v;
    v = w;

    FPDEBUG_END_STAGE(0);
}
```

Case Study: Biochemical Algorithms Library (BALL)

- $> 400,000$ lines of code

Case Study: Biochemical Algorithms Library (BALL)

- $> 400,000$ lines of code

```
double StretchComponent::updateEnergy() {
    ...
    double distance = atom1->getDistance(*atom2);
    energy_ += stretch_[i].values.k
          * (distance - stretch_[i].values.r0)
          * (distance - stretch_[i].values.r0);
    ...
}
```

Case Study: Biochemical Algorithms Library (BALL)

- $> 400,000$ lines of code

```
double StretchComponent::updateEnergy() {
    ...
    double distance = atom1->getDistance(*atom2);
    energy_ += stretch_[i].values.k
          * (distance - stretch_[i].values.r0)
          * (distance - stretch_[i].values.r0);
    ...
}
```

- Catastrophic cancellation
- At most 24 bits canceled (double: 53 bit precision)

Case Study: Biochemical Algorithms Library (BALL)

- $> 400,000$ lines of code

```
double StretchComponent::updateEnergy() {
    ...
    double distance = atom1->getDistance(*atom2);
    energy_ += stretch_[i].values.k
          * (distance - stretch_[i].values.r0)
          * (distance - stretch_[i].values.r0);
    ...
}
```

- Catastrophic cancellation
- At most 24 bits canceled (double: 53 bit precision)

```
float Atom::getDistance(const Atom& a) const
```

Case Study: GNU Linear Programming Kit (GLPK)[2]

- $> 100,000$ lines of code

$$
\begin{aligned}
&\min\ -x_{20} \\
&\text{s.t. } (s+1)\,x_1 - x_2 \geqslant s-1, \\
&\qquad -s x_{i-1} + (s+1)\,x_i - x_{i+1} \geqslant (-1)^i\,(s+1) \ \text{ for } i = 2:19, \\
&\qquad -s x_{18} - (3s-1)\,x_{19} + 3x_{20} \geqslant -(5s-7)\,, \\
&\qquad 0 \leqslant x_i \leqslant 10 \text{ for } i = 1:13, \\
&\qquad 0 \leqslant x_i \leqslant \boxed{\text{B}} \ \text{ for } i = 14:20, \\
&\qquad \text{all } x_i \text{ integers},
\end{aligned}
$$

[2] Neumaier and Shcherbina: "Safe bounds in linear and mixed-integer linear programming", Mathematical Programming, 2004

Case Study: GNU Linear Programming Kit (GLPK)[2]

- $> 100,000$ lines of code

$$
\begin{aligned}
&\min\ -x_{20}\\
&\text{s.t. } (s+1)\,x_1 - x_2 \geqslant s-1,\\
&\qquad -sx_{i-1} + (s+1)\,x_i - x_{i+1} \geqslant (-1)^i\,(s+1) \text{ for } i = 2:19,\\
&\qquad -sx_{18} - (3s-1)\,x_{19} + 3x_{20} \geqslant -(5s-7),\\
&\qquad 0 \leqslant x_i \leqslant 10 \text{ for } i = 1:13,\\
&\qquad 0 \leqslant x_i \leqslant \boxed{\text{B}} \text{ for } i = 14:20,\\
&\qquad \text{all } x_i \text{ integers},
\end{aligned}
$$

- Unique solution if $\text{B} \geq 2$

$$x = (1, 2, 1, 2, \ldots, 1, 2)^T$$

[2] Neumaier and Shcherbina: "Safe bounds in linear and mixed-integer linear programming", Mathematical Programming, 2004

Case Study: GNU Linear Programming Kit (GLPK)

- Binary search
- B = 21871

$$x = (1, 2, 1, 2, \ldots, 1, 2)^T$$

- B = 21872

```
Problem has no integer feasible solution
```

Case Study: GNU Linear Programming Kit (GLPK)

- Binary search
- B = 21871

$$x = (1, 2, 1, 2, \ldots, 1, 2)^T$$

- B = 21872

```
Problem has no integer feasible solution
```

- Compared the runs
- Found variable that differs

Bound	Shadow	Original
21871	-21871	-21871
21872	-21872	0

Case Study: Calculix (SPEC CFP2006)

```
double DVdot (int size, double y[], double x[]) {


    double sum = 0.0;
    int i;
    for (i = 0; i < size; i++) {
       sum += y[i] * x[i];
    }


    return sum;
}
```

Case Study: Calculix (SPEC CFP2006)

```
double DVdot (int size, double y[], double x[]) {
    FPDEBUG_BEGIN();

    double sum = 0.0;
    int i;
    for (i = 0; i < size; i++) {
       sum += y[i] * x[i];
    }

    FPDEBUG_END();
    return sum;
}
```

Case Study: Calculix (SPEC CFP2006)

```
double DVdot (int size, double y[], double x[]) {
    FPDEBUG_BEGIN();

    double sum = 0.0;
    int i;
    for (i = 0; i < size; i++) {
       sum += y[i] * x[i];
    }

    FPDEBUG_INSERT_SHADOW(&sum);
    FPDEBUG_END();
    return sum;
}
```

Case Study: Calculix (SPEC CFP2006)

```
double DVdot (int size, double y[], double x[]) {
    FPDEBUG_BEGIN();

    double sum = 0.0;
    int i;
    for (i = 0; i < size; i++) {
       sum += y[i] * x[i];
    }

    double errorBound = 1e-2;
    if (FPDEBUG_ERROR_GREATER(&sum, &errorBound)) {
       /* Print arrays x, and y */
    }
    FPDEBUG_INSERT_SHADOW(&sum);
    FPDEBUG_END();
    return sum;
}
```

Performance: SPEC CFP2006

Benchmark	Original	Analyzed	Slowdown
bwaves	47.50 s	7920.00 s	167 x
gamess	0.70 s	381.00 s	544 x
milc	30.90 s	7860.00 s	224 x
gromacs	2.10 s	991.00 s	472 x
cactusADM	4.70 s	4777.00 s	1016 x
leslie3d	59.80 s	17467.00 s	292 x
namd	19.80 s	18952.00 s	957 x
soplex	0.03 s	5.00 s	185 x
povray	0.90 s	400.00 s	444 x
calculix	0.07 s	17.10 s	244 x
GemsFDTD	5.50 s	1146.00 s	208 x
tonto	1.26 s	404.00 s	321 x
lbm	9.55 s	2893.00 s	303 x
wrf	7.68 s	2623.00 s	342 x
sphinx3	4.41 s	938.00 s	213 x

Conclusion

Dynamic program analysis

- Detects floating-point accuracy problems
- Detects catastrophic cancellations
- Works on large-scale programs
- Finds real-world problems
- Is open source: github.com/fbenz/fpdebug

Conclusion

Dynamic program analysis

- Detects floating-point accuracy problems
- Detects catastrophic cancellations
- Works on large-scale programs
- Finds real-world problems
- Is open source: github.com/fbenz/fpdebug

Thank You!

Questions?