

Validating floating point implementations

Pierre Ciadoux

NIST CAVP / PQC

MPTS 2026

01/27/2026

NIST

NATIONAL INSTITUTE OF
STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

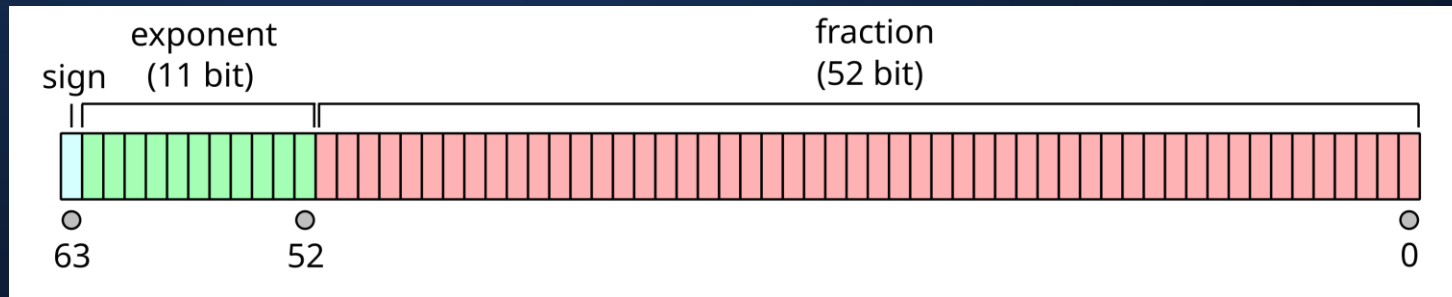
Content (12 min presentation)

- Definition of floating points and fixed points
- Inaccuracy and Variation examples
- Real world impact

Floating points (as specified in IEEE754)

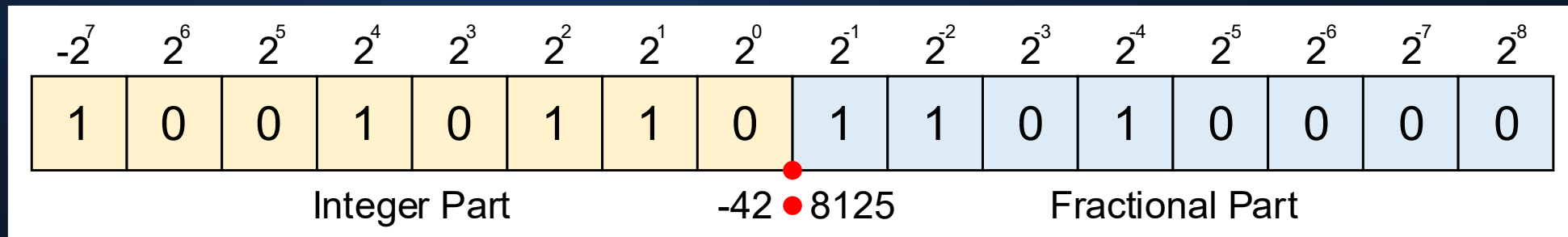
$$(-1)^{\text{sign}} \times \text{mantissa} \times 2^{\text{exponent}}$$

- Most processors use floating points based on standard IEEE754
- Useful for very large or very small numbers
- Efficient using hardware computations (FPU)



Fixed points

- Fixed number of bits to represent the integer and fractional parts
- With 64 bits: 32 bits for integral and fractional part (Q32.32)
- No Floating Point Unit (FPU) needed



16-bit fixed point example (Q8.8)

Inaccuracy

```
int main(){
    float a = 2;
    for (int i = 0; i < 6; i++)
        a += 0.1;
    if (a == 2.6)
        printf("The value is correct");
    else
        printf("a should be 2.6 but is : %f\n", a);
}
```

Inaccuracy

```
int main(){
    float a = 2;
    for (int i = 0; i < 6; i++)
        a += 0.1;
    if (a == 2.6)
        printf("The value is correct");
    else
        printf("a should be 2.6 but is : %f\n", a);
}
```

a should be 2.6 but is : 2.599999

**GCC: warning: floating-point comparison is always false;
constant cannot be represented exactly in type 'float' [-Wliteral-range]**

Variations

- Implementation differences
- Example: $(a + b) + c \neq a + (b + c)$

```
int main(){
    float a = 1.0000001f, b = 2.0000002f, c = 3.0000003f;
    float abc = a + b + c;
    float bca = b + c + a;
    printf("%.7f\n%.7f\n", abc, bca);
}
// 6.0000010
// 6.0000005
```

Variations

- Compiler induced
- Fused-instructions
- -ffp-contract=**on**/off

```
int main(){
    double a = 33370374411415;
    double b = -185547331688695;

    double ab = a * b;        // FMUL (+rounding)
    double abMab = ab - ab;   // FSUB (+rounding)
    printf("2 lines: %f\n", abMab); // ab - ab == 0

    // FMUL (+rounding)
    // FNMSUB (multiply and subtract +rounding)
    abMab = (a * b) - (a * b);
    printf("1 line : %f\n", abMab); // ab - ab != 0
}

// 2 lines: 0.000000
// 1 line: 271906978383.000000
```

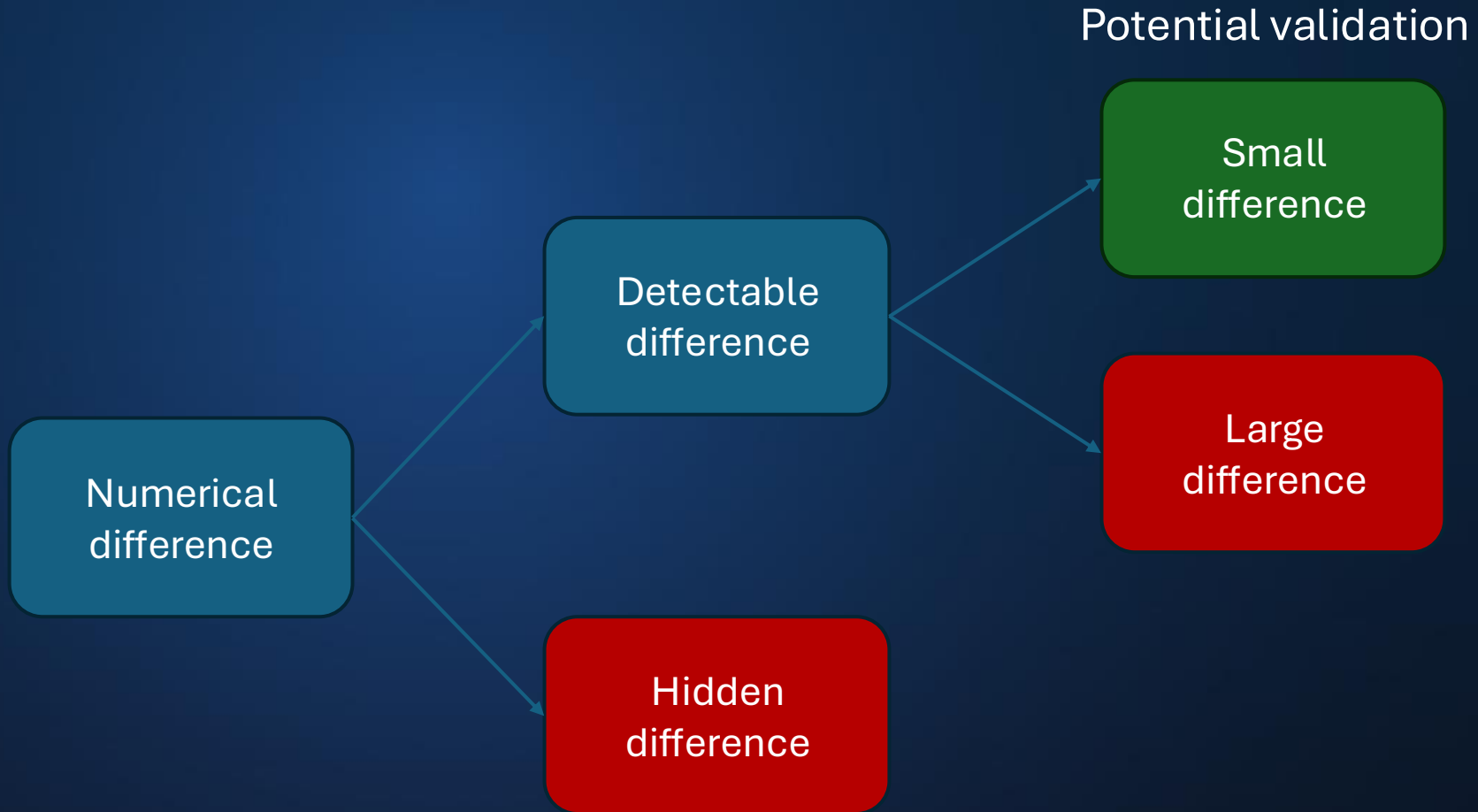
Real world impact

A variation can impact execution path
and/or numerical values

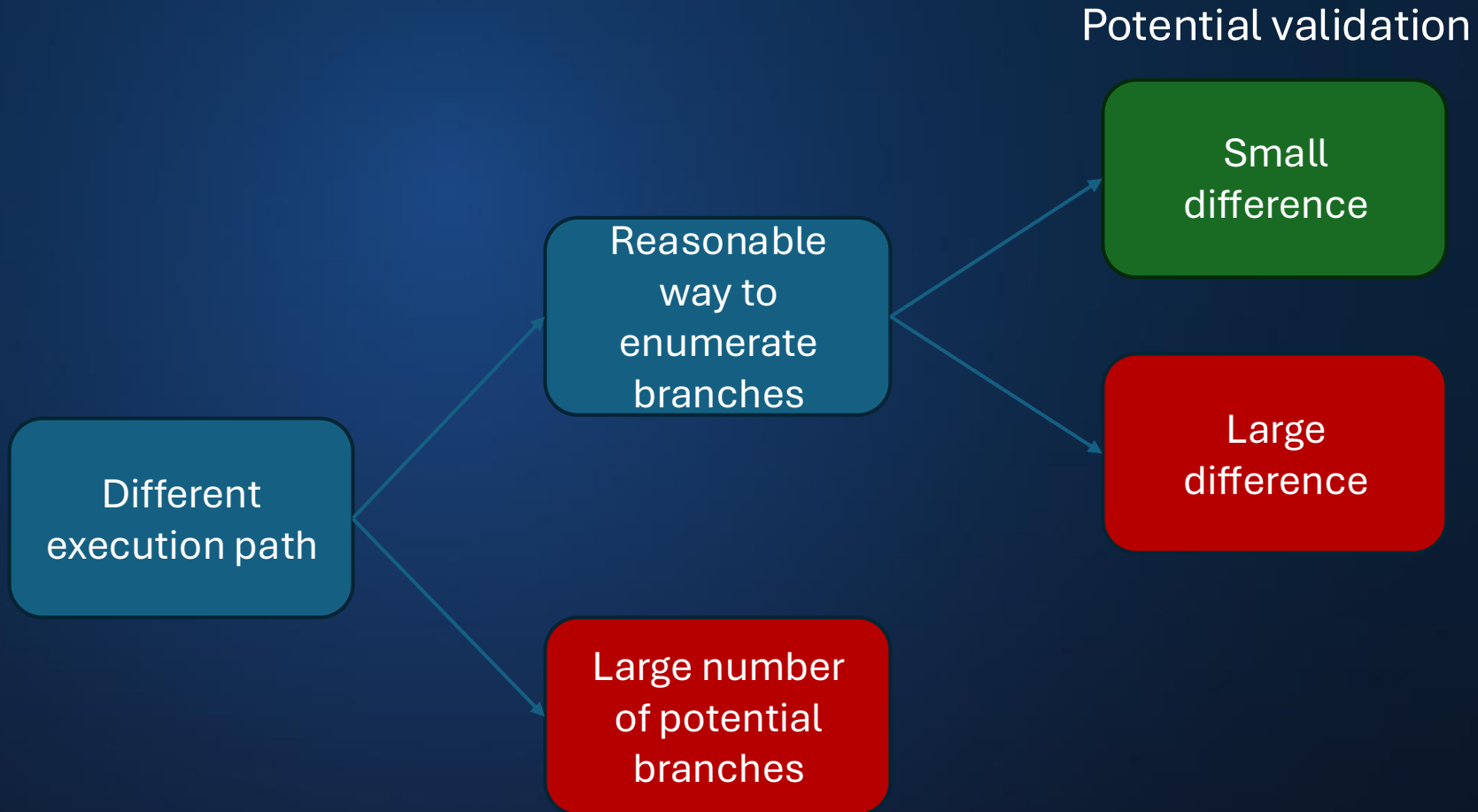
```
int ComputeValue(float a) {  
  
    if (a > 5.52f) { // tiny FP difference -> different branch  
        return -1;  
    }  
  
    return (int)roundf(a); // tiny FP difference -> numerical difference only  
}
```

ComputeValue(5.48)	ComputeValue(5.51)	ComputeValue(5.53)
5	6	-1 (error)

Validating different numerical values



Validating different execution path



Conclusion

- Validation can decide to allow for some variation in floating (and fixed) point computations
 - Depends on the scheme used
- Using floating points can impact values but will only have a real impact on a small percentage of computations
- It is likely that a validator computes the same values and doesn't notice variations
 - -> Need for more test vectors than deterministic

Questions?

Upcoming article on Validating Falcon with:

- Maxime Bros
- Chris Celi
- Ray Perlner

Contact:

pierre.ciadoux@nist.gov

cavp@nist.gov