CS101 Introduction to computing

Floating Point Numbers

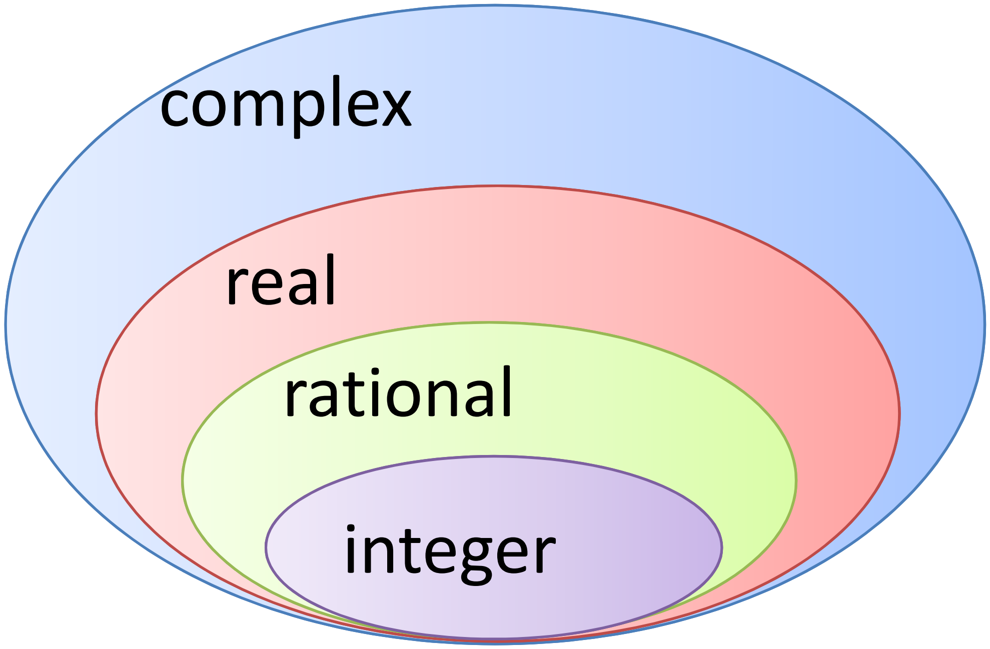
A. Sahu and S. V .Rao
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

Outline

- Need to floating point number
- Number representation : IEEE 754
- Floating point range
- Floating point density
  - Accuracy
- Arithmetic and Logical Operation on FP
- Conversions and type casting in C

Need to go beyond integers

- integer 7
- rational 5/8
- real $\sqrt{3}$
- complex 2 - 3 i



Extremely large and small values:

- distance pluto - sun = 5.9 10^{12} m
- mass of electron = 9.1 x 10^{-28} gm

Representing fractions

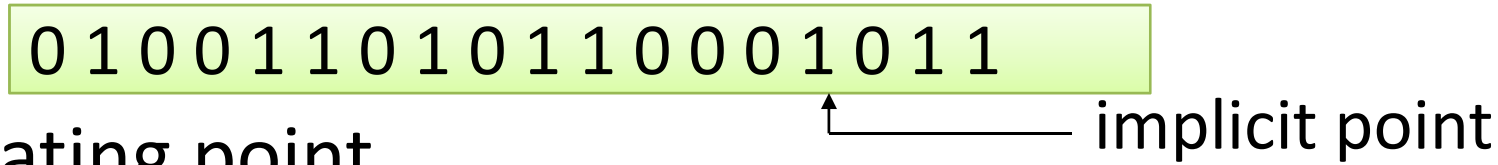
- Integer pairs (for rational numbers)

5	8	= 5/8

- Strings with explicit decimal point

-	2	4	7	.	0	9

- Implicit point at a fixed position



0 1 0 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1

implicit point

- Floating point

fraction x base $^{\textbf{power}}$

Numbers with binary point

$101.11 = 1x2^2 + 0x2^1 + 1x2^0 + \; . \; + 1x2^{-1} + 1x2^{-2}$

$= 4 + 1 + . + \; 0.5 + 0.25 = 5.75_{10}$

0.6 = 0.10011001100110011001.....

.6 x 2 = 1 + .2

.2 x 2 = 0 + .4

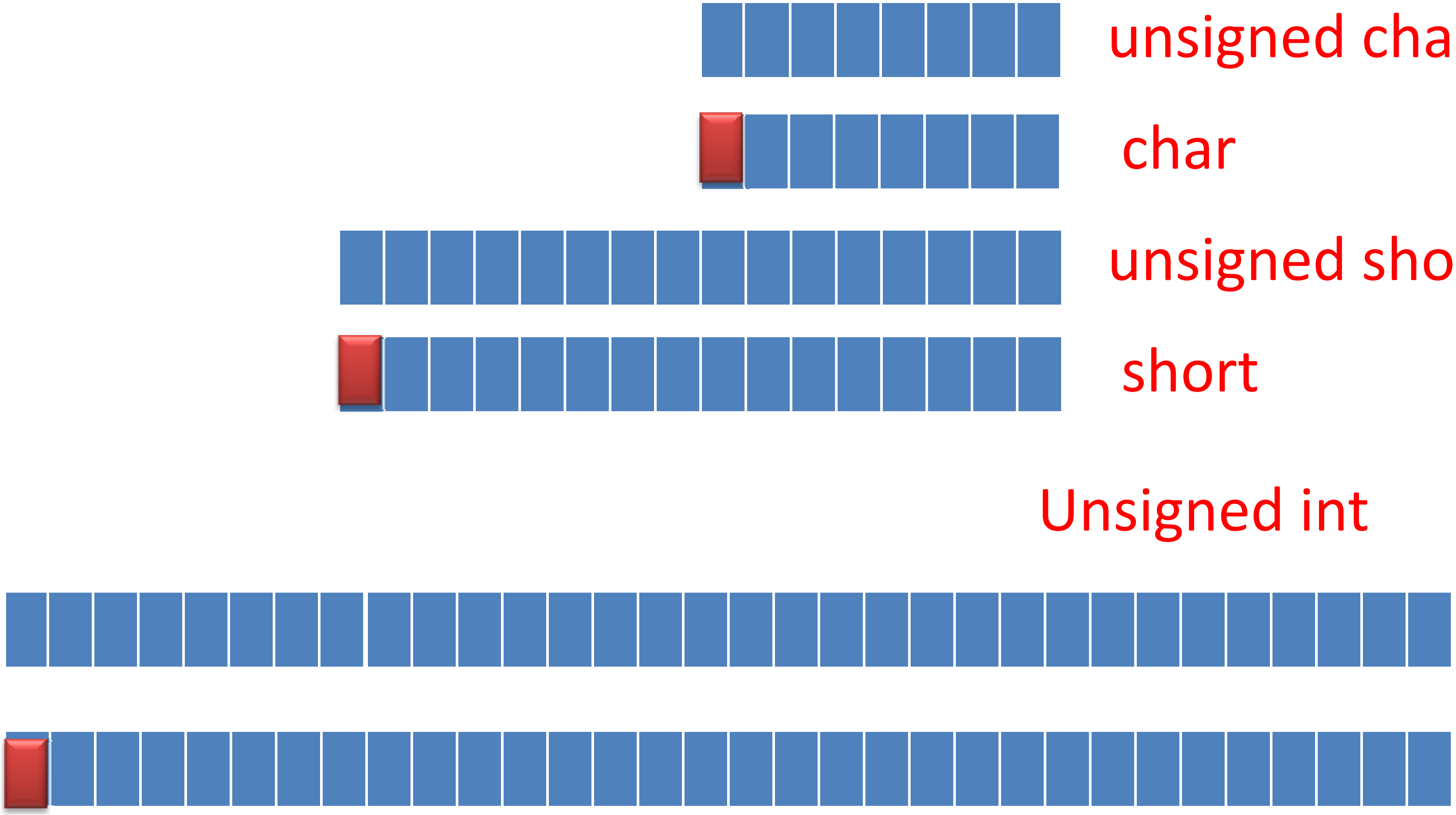
.4 x 2 = 0 + .8

.8 x 2 = 1 + .6

Numeric Data Type

- **char, short, int, long int**
  - char : 8 bit number (1 byte=1B)
  - short: 16 bit number (2 byte)
  - int : 32 bit number (4B)
  - long int : 64 bit number (8B)
- **float, double, long double**
  - float : 32 bit number (4B)
  - double : 64 bit number (8B)
  - long double : 128 bit number (16B)

Numeric Data Type



unsigned char

char

unsigned short

short

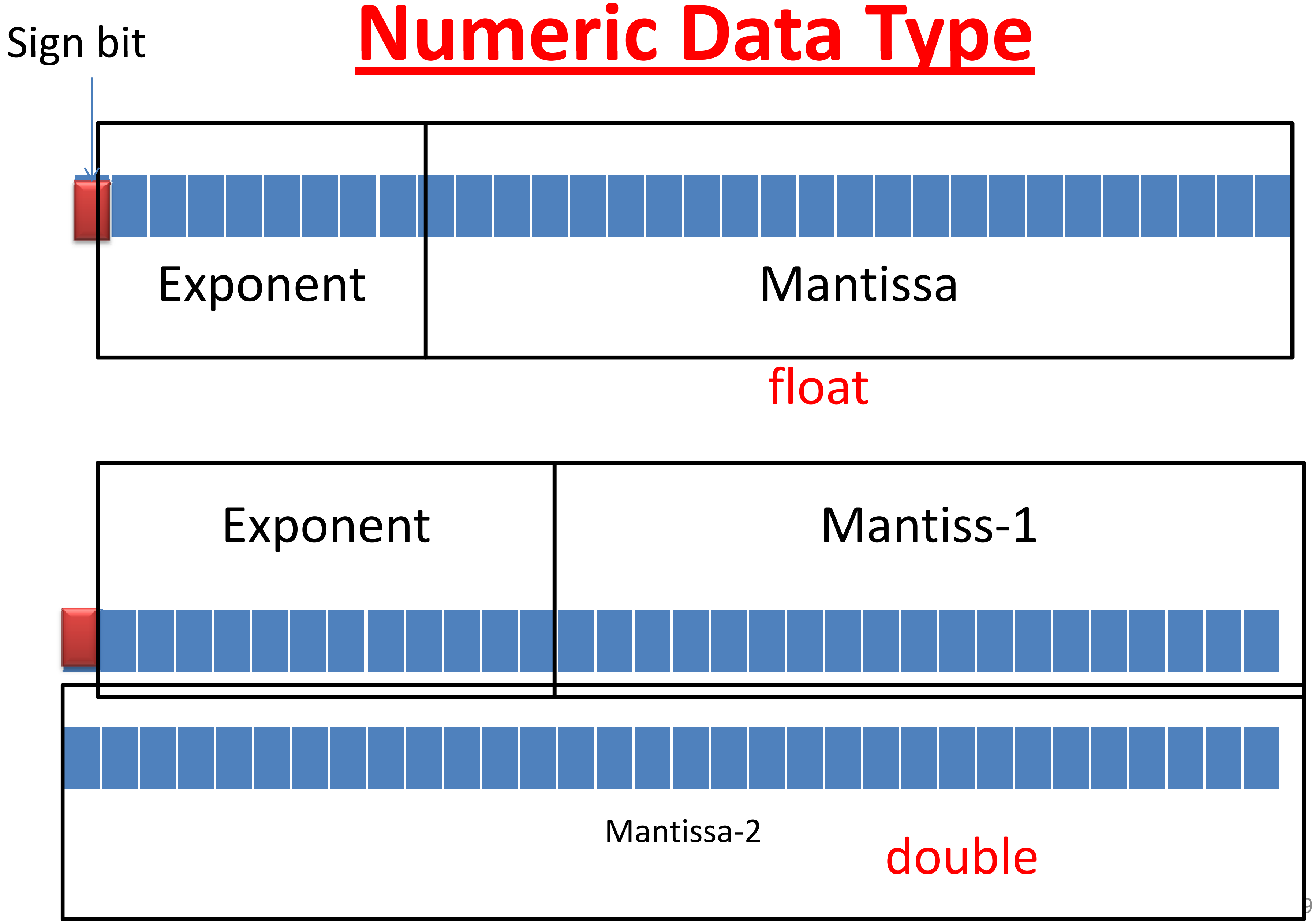
Unsigned int

int

Numeric Data Type

- **char, short, int, long int**
  - We have : Signed and unsigned version
  - char (8 bit)
    - char : -128 to 127, we have +0 and -0 ☺ ☺ Fun
    - unsigned char: 0 to 255
  - int : -2^{31} to $2^{31}-1$
  - unsigned int : 0 to $2^{32}-1$
- **float, double, long double**
  - For fractional, real number data
  - All these numbered are signed and get stored in different format

Numeric Data Type



FP numbers with base = 10

$$(-1)^{S} \times F \times 10^{E}$$

S = Sign

F = Fraction (fixed point number)

usually called **Mantissa** or **Significand**

E = Exponent (positive or negative integer)

- Example $\mathbf{5}.9\text{x}10^{12}$, $-\mathbf{2}.6\text{x}10^{3}$ $\mathbf{9}.1 \text{ x } 10^{-28}$
- **Only one non-zero digit left to the point**

FP numbers with base = 2

$$(-1)^S \times F \times 2^E$$

S = Sign

F = Fraction (fixed point number)
usually called **Mantissa** or **Significand**

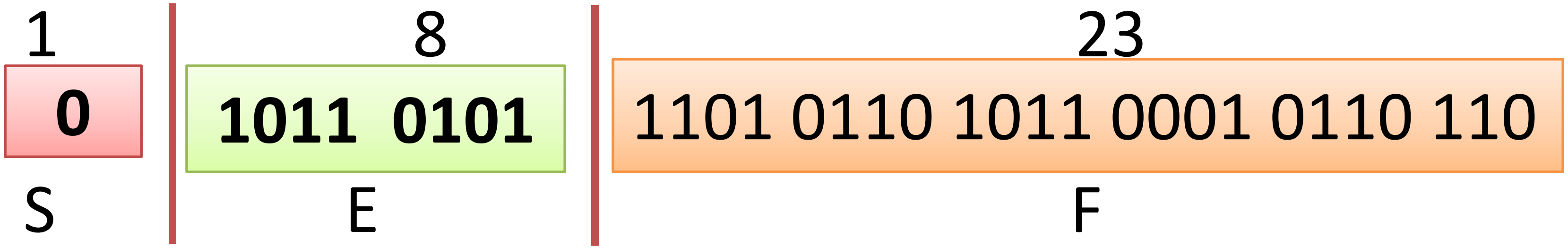
E = Exponent (positive or negative integer)

- How to divide a word into S, F and E?
- How to represent S, F and E?

- Example $1.0101\text{x}2^{12}$, $-1.11012\text{x}10^{3}$ 1.101×2^{-18}
- **Only one non-zero digit left to the point: default it will be 1 incase of binary**
  - **So no need to store this**

IEEE 754 standard

- Single precision numbers

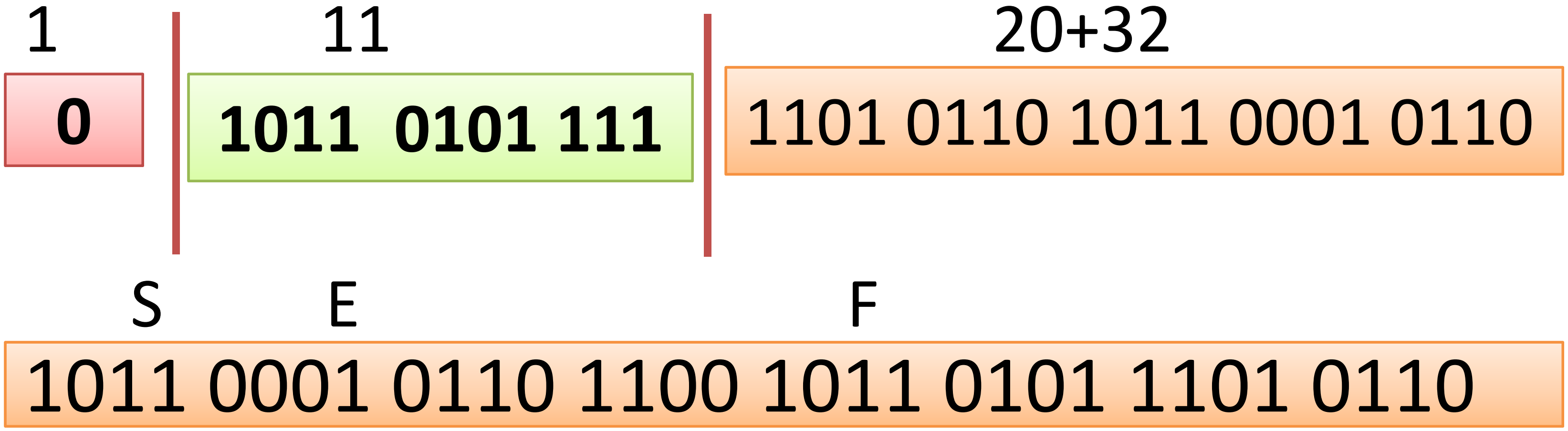


1 8 23

0 | 1011 0101 | 1101 0110 1011 0001 0110 110

S E F

- Double precision numbers



1 11 20+32

0 | 1011 0101 111 | 1101 0110 1011 0001 0110

S E F

1011 0001 0110 1100 1011 0101 1101 0110

Representing F in IEEE 754

- Single precision numbers

23

1. 110101101011000101101101

F

- Double precision numbers

20+32

1. 1011101011000101101101

F

101100010110110010110101110101101

Only one non-zero digit left to the point: default it will be 1 incase of binary. So no need to store this bit

Value Range for F

- Single precision numbers

$1 \leq F \leq 2 - 2^{-23}$ or $1 \leq F < 2$

- Double precision numbers

$1 \leq F \leq 2 - 2^{-52}$ or $1 \leq F < 2$

These are “normalized”.

Representing E in IEEE 754

- Single precision numbers

8

10110101

E bias 127

- Double precision numbers

11

10110101110

E bias 1023

Floating point values

- $E=E'-127$, $V=(-1)^s \times 1.M \times 2^{E'-127}$
- $V= 1.1101\ldots \times 2^{(40-127)} = 1.1101.. \times 2^{-87}$

- Single precision numbers

1	8	23
0	**0010 1000**	1101 0110 1011 0001 0110 110
S	E’	F

Floating point values

- $E=E'-127$, $V=(-1)^{s} \times 1.M \times 2^{E'-127}$
- $V=-1.1 \times 2^{(126-127)}=-1.1 \times 2^{-1}=-0.11 \times 2^{0}$

$$=-0.11=-11/2^{2}{}_{10}=-3/4_{10}=-0.75_{10}$$

- Single precision numbers

1	8	23
1	**0111 1110**	1000 0000 0000 0000 0000 000
S	E’	F

Value Range for E

- Single precision numbers

  $-126 \leq E \leq 127$

  (all 0's and all 1's have special meanings)

- Double precision numbers

  $-1022 \leq E \leq 1023$

  (all 0's and all 1's have special meanings)

Floating point demo applet on the web

- https://www.h-schmidt.net/FloatConverter/IEEE754.html
- Google “Float applet” to get the above link

Overflow and underflow

largest positive/negative number (SP) =

$\pm(2 - 2^{-23}) \times 2^{127} \cong \pm 2 \times 10^{38}$

smallest positive/negative number (SP) =

$\pm 1 \times 2^{-126} \cong \pm 2 \times 10^{-38}$

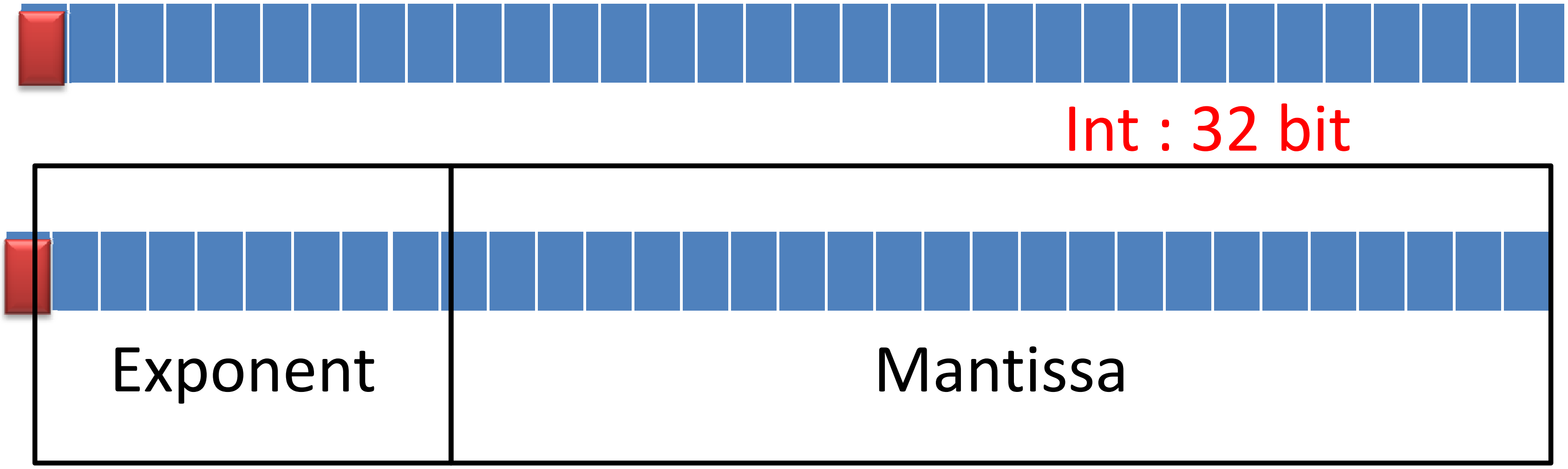
Largest positive/negative number (DP) =

$\pm(2 - 2^{-52}) \times 2^{1023} \cong \pm 2 \times 10^{308}$

Smallest positive/negative number (DP) =

$\pm 1 \times 2^{-1022} \cong \pm 2 \times 10^{-308}$

Density of int vs float



Int : 32 bit

Exponent

Mantissa

Float : 32 bit

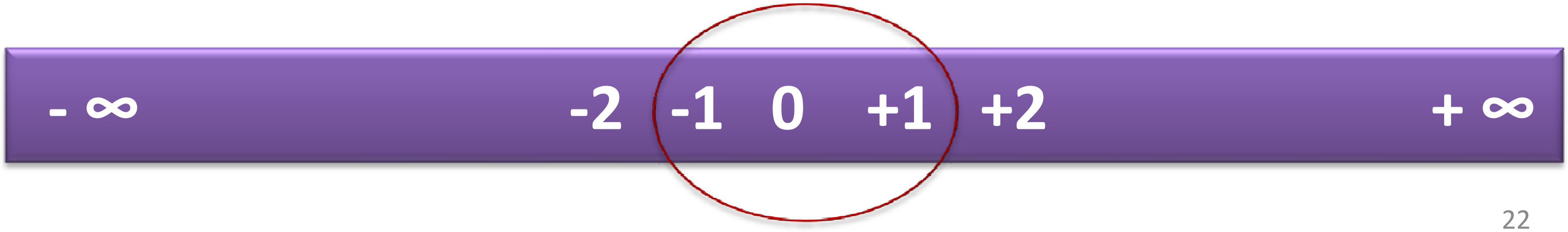
- Number of number can be represented
  - Both the cases (float, int) : 2^{32}
- Range
  - int (-2^{31} to $2^{31}-1$)
  - float Large $\pm(2 - 2^{-23}) \times 2^{127}$ **Small**$\pm 1 \times 2^{-126}$
- 50% of float numbers are **Small** **(**less then $\pm$**1**)

Density of Floating Points

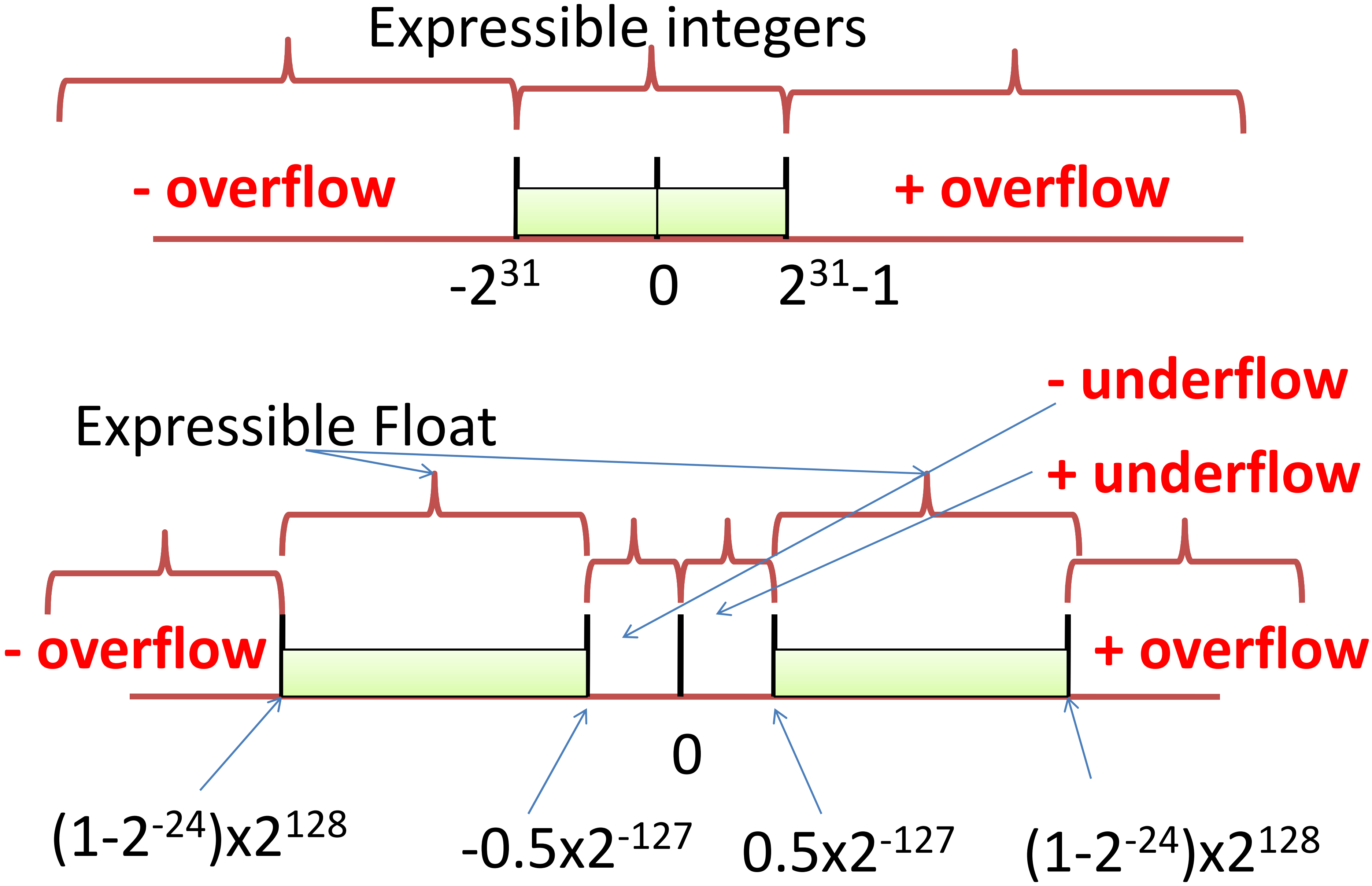
- 256 Persons in Room of Capacity 256 (Range)

  8 bit integer : 256/256 = 1

- 256 person in Room of Capacity 200000 (Range)
  - 1st Row should be filled with 128 person
  - 50% number with negative power are -1 < N > +1
- Density of Floating point number is
  - Dense towards 0



Expressible Numbers(int and float)



Expressible integers

- overflow

+ overflow

-2^{31}

0

$2^{31}-1$

- underflow

Expressible Float

+ underflow

- overflow

+ overflow

0

$(1-2^{-24})x2^{128}$

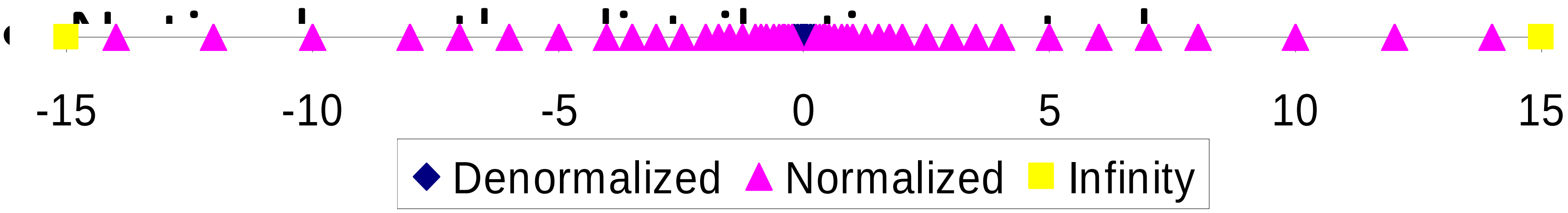
$-0.5x2^{-127}$

$0.5x2^{-127}$

$(1-2^{-24})x2^{128}$

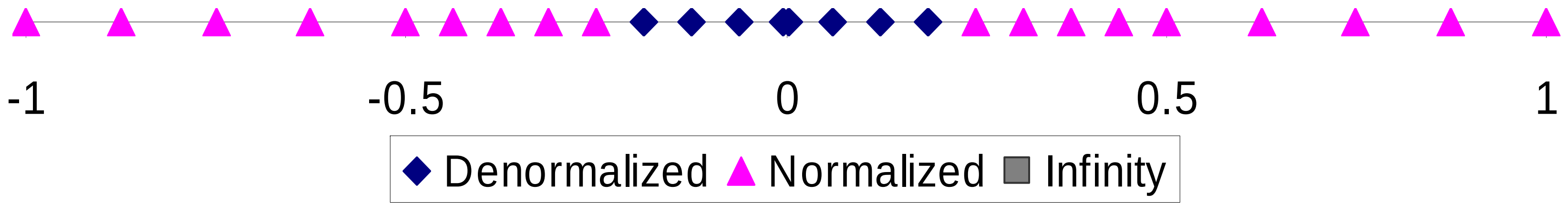
Distribution of Values

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 3



Distribution of Values (close-up view)

- 6-bit IEEE-like format
  - e = 3 exponent bits
  - f = 2 fraction bits
  - Bias is 3



Density of 32 bit float SP

- Fraction/mantissa is 23 bit
- Number of different number can be stored for particular value of exponent
  - Assume for exp=1, 2^{23}=8x1024x1024 ≈8x10^6
  - Between 1-2 we can store 8x10^6 numbers
- Similarly
  - for exp=2, between 2-4, 8x10^6 number of number can be stored
  - for exp=3, between 4-8, 8x10^6 number of number can be stored
  - for exp=4, between 8-16, 8x10^6 number of number can be stored

Density of 32 bit float SP

- Similarly
  - for exp=23, between 2^{22}-2^{23}, 8x10^{6} number of number can be stored
  - **for exp=24, between 2^{23}-2^{24}, 8x10^{6} number of number can be stored**

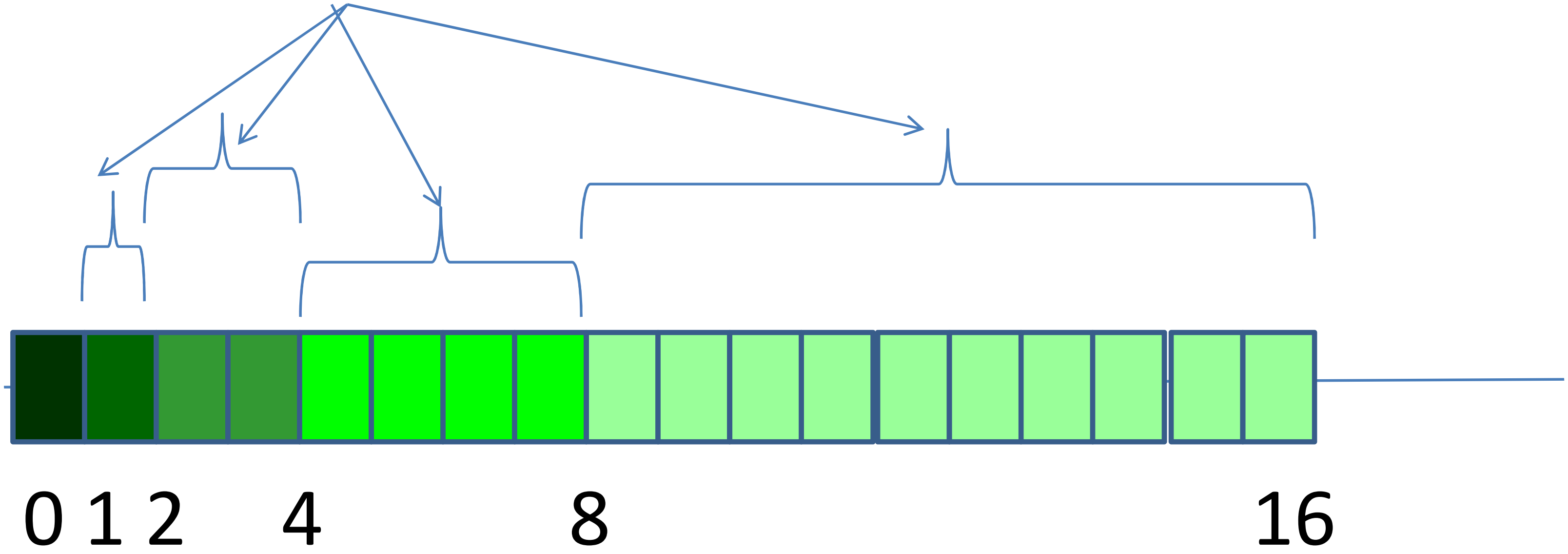
    OK
  - **for exp=25, between 2^{24}-2^{25}, 8x10^{6} number of number can be stored**
    - **2^{24}-2^{25} >8 x10^{6}**

    BAD
  - …
  - for exp=127, between 2^{126}-2^{127}, 8x10^{6} number of number can be stored

    WROST

Density of 32 bit float SP

- 2^{23}=8x1024x1024 ≈$8x10^6$



Numbers in float format

- largest positive/negative number (SP) =

  $\pm(2 - 2^{-23}) \times 2^{127} \cong \pm 2 \times 10^{38}$

Second largest number :

$\pm(2 - 2^{-22}) \times 2^{127}$

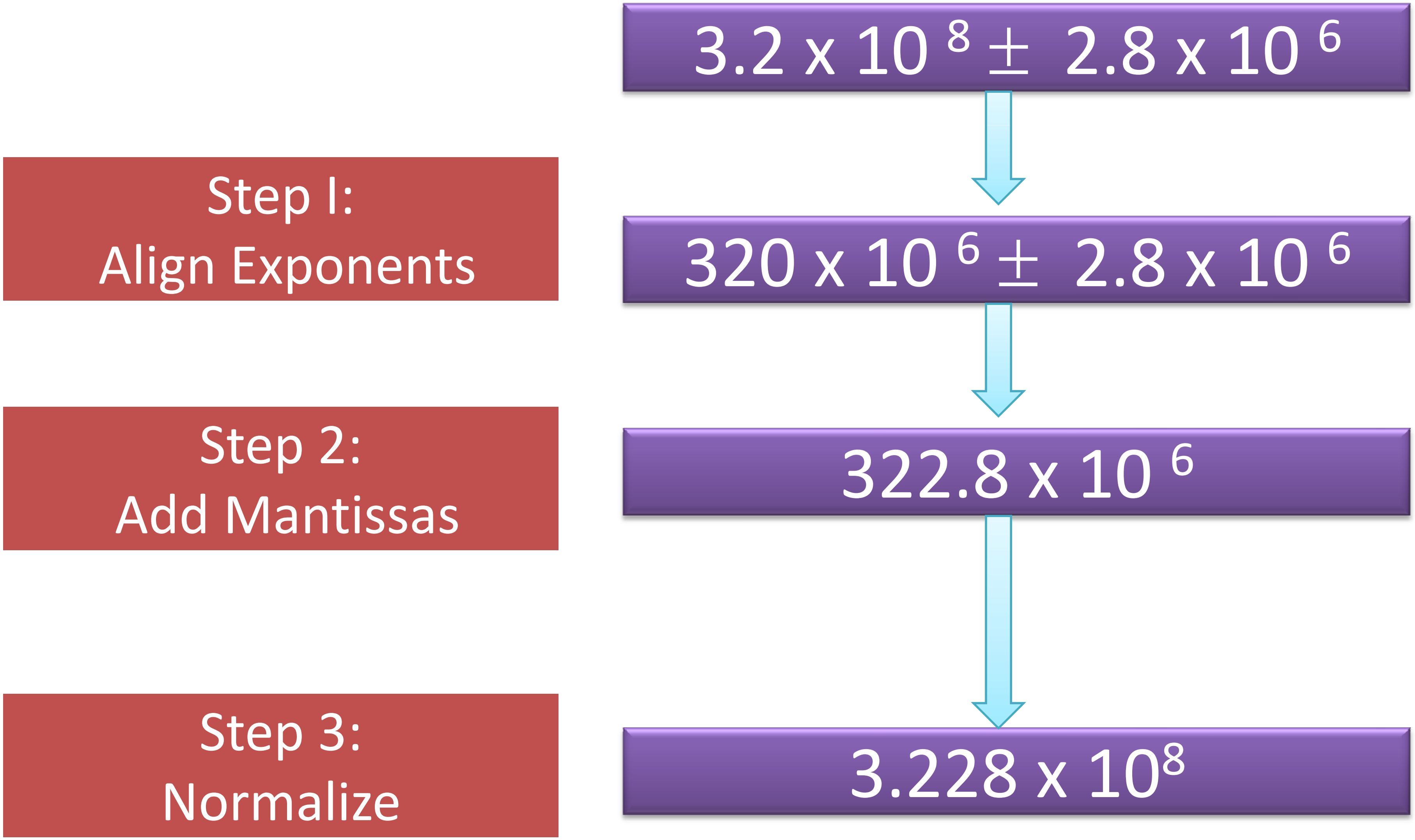
Difference Largest FP - 2nd largest FP

$= (2^{-23}-2^{-22})x2^{127}=2x2^{105}=2x10^{32}$

Smallest positive/negative number (SP) =

$\pm 1 \times 2^{-126} \cong \pm 2 \times 10^{-38}$

Addition/Sub of Floating Point



$3.2 \times 10^{8} \pm 2.8 \times 10^{6}$

Step I:
Align Exponents

$320 \times 10^{6} \pm 2.8 \times 10^{6}$

Step 2:
Add Mantissas

322.8×10^{6}

Step 3:
Normalize

3.228×10^{8}

Floating point operations: ADD

- Add/subtract $A = A1 \pm A2$

$[(-1)^{S1} \text{ x } F1 \text{ x } 2^{E1}] \pm [(-1)^{S2} \text{ x } F2 \text{ x } 2^{E2}]$

suppose E1 > E2, then we can write it as

$[(-1)^{S1} \text{ x } F1 \text{ x } 2^{E1}] \pm [(-1)^{S2} \text{ x } F2' \text{ x } 2^{E1}]$

where $F2' = F2 / 2^{E1-E2}$,

The result is

$(-1)^{S1} \text{ x } (F1 \pm F2') \text{ x } 2^{E1}$

It may need to be normalized

$3.2 \text{ x } 10^{8} \pm 2.8 \text{ x } 10^{6}$

$320 \text{ x } 10^{6} \pm 2.8 \text{ x } 10^{6}$

$322.8 \text{ x } 10^{6}$

$3.228 \text{ x } 10^{8}$

Testing Associatively with FP

- $X = -1.5\text{x}10^{38}$, $Y=1.5\text{x}10^{38}$, $z=1000.0$
- $X+(Y+Z) = -1.5\text{x}10^{38} + (1.5\text{x}10^{38} + 1000.0)$

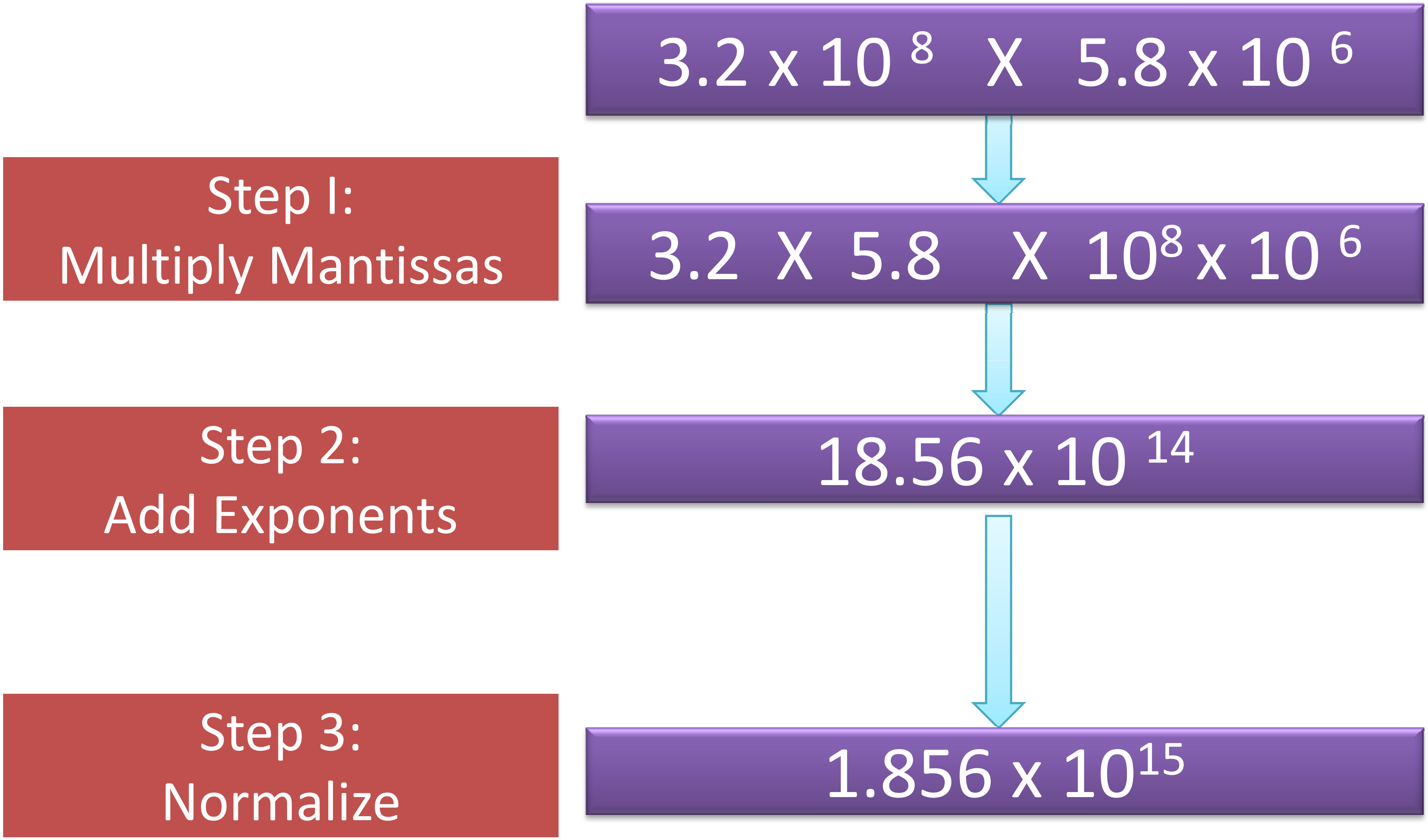
  $= -1.5\text{x}10^{38} + 1.5\text{x}10^{38}$

  $=0$
- $(X+Y)+Z = (-1.5\text{x}10^{38} + 1.5\text{x}10^{38}) + 1000.0$

  $= 0.0 + 1000.0$

  $=1000$

Multiply Floating Point



$3.2 \times 10^{8} \quad X \quad 5.8 \times 10^{6}$

Step I:
Multiply Mantissas

$3.2 \;X\; 5.8 \quad X \;\; 10^{8} \times 10^{6}$

Step 2:
Add Exponents

18.56×10^{14}

Step 3:
Normalize

1.856×10^{15}

For 32 bit SP Float : one 23 bit multiplication and 8 bit addition
For 32 bit int: one 32 bit multiplication
Above example: 3.2x5.8 is simpler, also 6+8 is also simpler as compared to 32 bit multiplication

Floating point operations

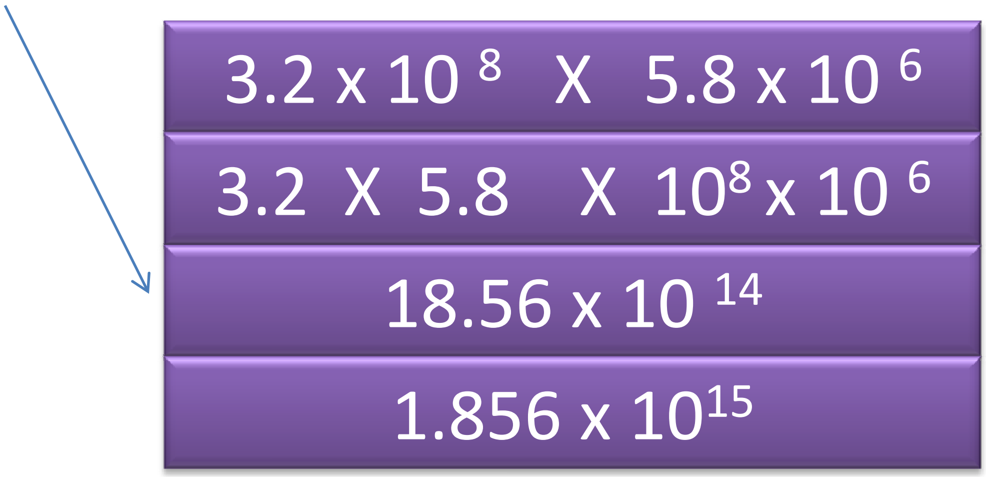
- Multiply

$[(-1)^{S1} \times F1 \times 2^{E1}]\ \mathbf{x}\ [(-1)^{S2} \times F2 \times 2^{E2}]$

$= (-1)^{S1 \oplus S2} \times (\mathbf{F1xF2}) \times 2^{E1+E2}$

Since $1 \leq (F1xF2) < 4$,

the result may need to be normalized



$3.2 \times 10^{8} \quad X \quad 5.8 \times 10^{6}$

$3.2 \ X \ 5.8 \quad X \ 10^{8} \times 10^{6}$

18.56×10^{14}

1.856×10^{15}

Floating point operations

- Divide

$[(-1)^{S1} \text{ x } F1 \text{ x } 2^{E1}] \div [(-1)^{S2} \text{ x } F2 \text{ x } 2^{E2}]$

$= (-1)^{S1 \oplus S2} \text{ x } (\mathbf{F1 \div F2}) \text{ x } 2^{E1-E2}$

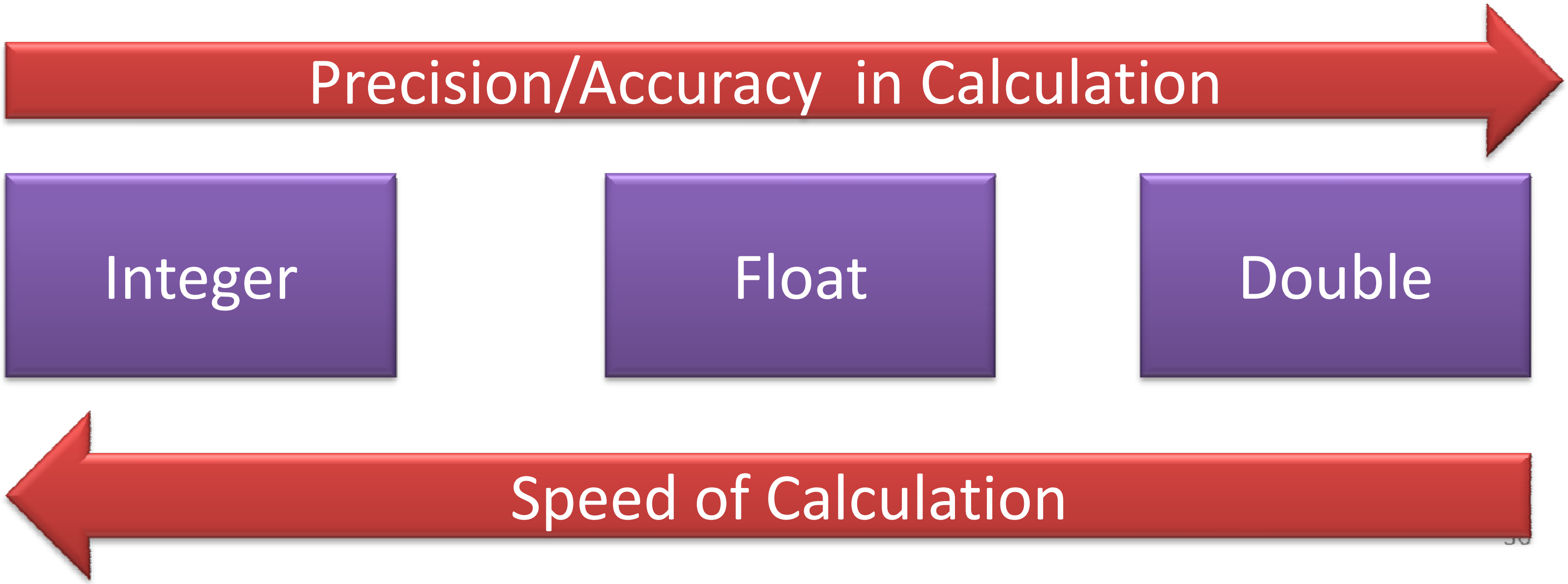
Since $.5 < (F1 \div F2) < 2$,

the result may need to be normalized

(assume $F2 \neq 0$)

Float and double

- Float : single precision floating point
- Double : Double precision floating point
- Floating points operation are slower
    - **But not in newer PC ☺ ☺**
- Double operation are even slower



Precision/Accuracy in Calculation

Integer

Float

Double

Speed of Calculation

Floating point Comparison

- Three phases
- Phase I: Compare sign (give result)
- Phase II: If (sign of both numbers are same)
  - Compare exponents and give result
  - **90% of case it fall in this categories**
  - **Faster as compare to integer comparison : Require only 8 bit comparison for float and 11 bit for double (Example : sorting of float numbers)**
- Phase III: If (both sign and exponents are same)
  - compare fraction/mantissa

Storing and Printing Floating Point

```
float x=145.0,y;
y=sqrt(sqrt((x)));
x=(y*y)*(y*y);
printf("\nx=%f",x);
```

Many Round off cause loss of accuracy

x=145.000015

```
float x=1.0/3.0;
if ( x==1.0/3.0)
   printf(“YES”);
else
   printf(“NO”);
```

Value stored in x is not exactly same as 1.0/3.0

One is before round of and other (stored x) is after round of

NO

Storing and Printing Floating Point

```
float a=34359243.5366233;
float  b=3.5366233;
float  c=0.00000212363;
printf("\na=%8.6f, b=%8.6f
    c=%8.12f\n", a, b, c );
```

```
a=34359243.000000
b=3.5366233
c=0.000002123630
```

Big number with small fraction can not combined

Storing and Printing Floating Point

```
//15 S digits to store
float a=34359243.5366233;
//8 S digits to store
float  b=3.5366233;
//6 S digits to store
float  c=0.00000212363;
```

Thumb rule: 8 to 9 significant digits of a number can be stored in a 32 bit number

Thanks