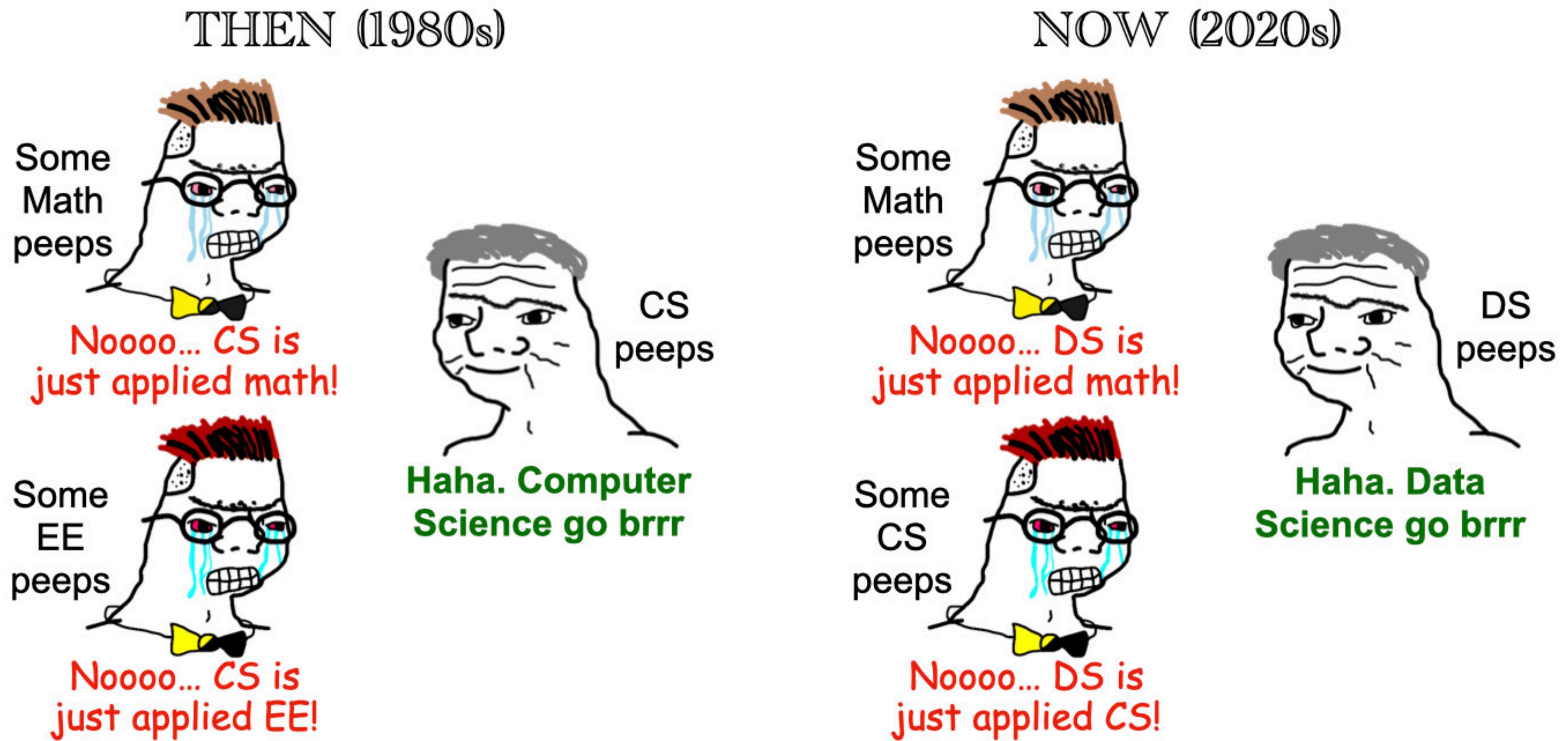


# DSC 102

## Systems for Scalable Analytics

- Haojian Jin



Meme idea credit: <https://datasystemsfun.tumblr.com/>

# Where are we in the class?

Week	Topic	Systems Principles
1-2	Basics of Machine Resources: Computer Organization	
2-3	Basics of Machine Resources: Operating Systems	
4	Basics of Cloud Computing	
4-5	Parallel and Scalable Data Processing: Parallelism Basics	
6	Midterm Exam on TBD	
6-7	Parallel and Scalable Data Processing: Scalable Data Access	
7-8	Parallel and Scalable Data Processing: Data Parallelism	
9	Dataflow Systems	
10	ML Model Building Systems	
11	Final Exam on Dec 15	

# Where are we in the class?

- ❖ Basics of Computer Organization
  - ❖ **Digital Representation of Data**
    - ❖ **Binary, integer, float, tradeoffs**
  - ❖ Processors and Memory Hierarchy
- ❖ Basics of Operating Systems
  - ❖ Process Management: Virtualization; Concurrency
  - ❖ Filesystem and Data Files
  - ❖ Main Memory Management
- ❖ Persistent Data Storage

# What is data?

データとは、情報の集合です。情報は、物事の特徴や状況などを示す情報であり、データは、その情報を記述する文字や数字などの組合せです。

データには、以下のような種類があります。

- 構造化データ：データが明確なルールで組織されているもの。たとえば、CSVファイルやJSON形式のデータなど。
- 非構造化データ：データが明確なルールで組織されていないもの。たとえば、テキストファイルや画像、音楽などのデータなど。
- 時間系列データ：データが時間の流れとともに収集されたもの。たとえば、気温や降雨量などのデータなど。
- 地理空間データ：データが位置情報を含むもの。たとえば、地図やGPSデータなど。
- 音声データ：データが音波や音楽などの形で表現されるもの。たとえば、音楽や会話などのデータなど。
- 映像データ：データが映像や動画などの形で表現されるもの。たとえば、映画やテレビ番組などのデータなど。

データを用いて、AIや機械学習などの技術によって、さまざまな分析や予測が行われます。

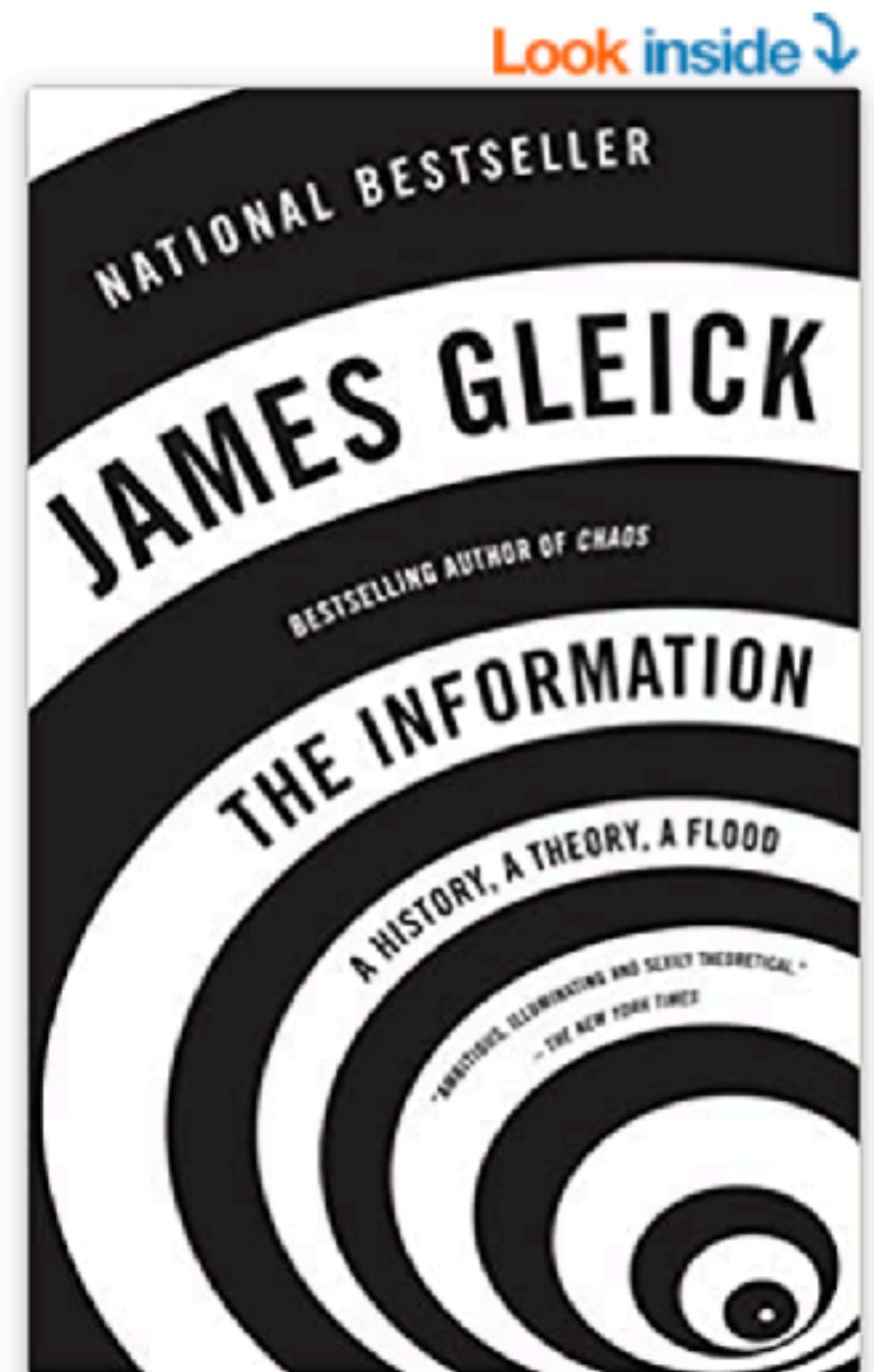
# Digital Representation of Data

- **Bits:** 01010101
- **Data type:** Boolean, Byte, Integer, “floating point” number (Float), Character, and String
- **Data structure:** Array, Linked list, Tuple, Graph, etc.

Programmer

Machine?

# Why bits?



Listen



[See this image](#)

## The Information: A History, A Theory, A Flood Paperback – Illustrated, March 6, 2012

by [James Gleick](#) (Author)

1,339 ratings

[Editors' pick](#) / [Best Science Fiction & Fantasy](#)

Searching •••

[See all formats and editions](#)

Kindle

\$9.99

[Read with Our Free App](#)

Audiobook

\$0.00

[Free with your Audible trial](#)

Hardcover

\$13.25 - \$39.94

[Other new, used and collectible from \\$2.39](#)

Paperback

**\$12.23**

[Other new and used from \\$2.45](#)

Audio CD

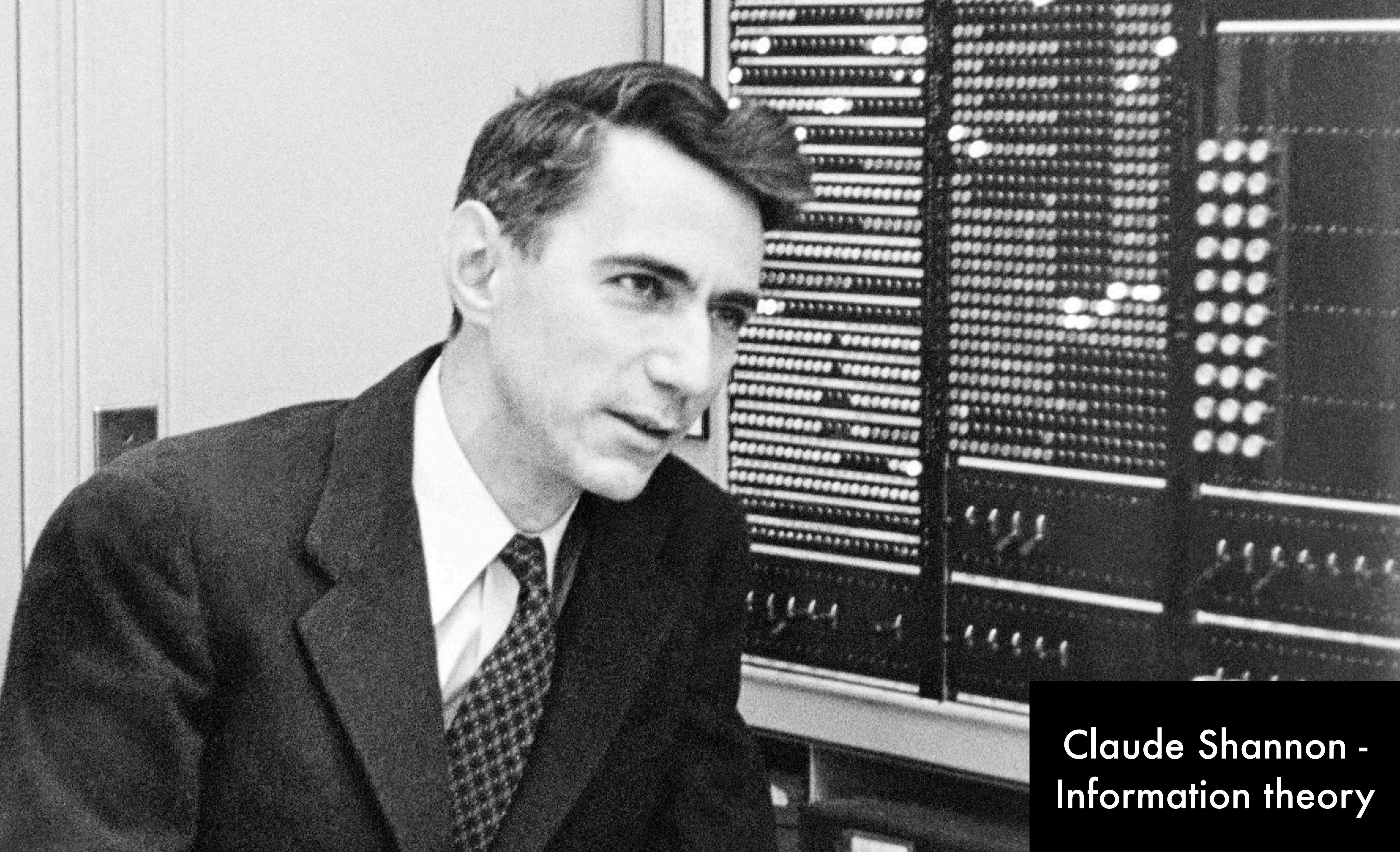
\$25.04

[8 Used from \\$21.06](#)

From the bestselling author of the acclaimed *Chaos* and *Genius* comes a thoughtful and provocative exploration of the big ideas of the modern era: Information, communication, and information theory.

Acclaimed science writer James Gleick presents an eye-opening vision of how our relationship to information has transformed the very nature of human consciousness. A fascinating intellectual journey through the history of communication and information, from the language of Africa's talking drums to the invention of written alphabets; from the electronic transmission of code to the

[▼ Read more](#)



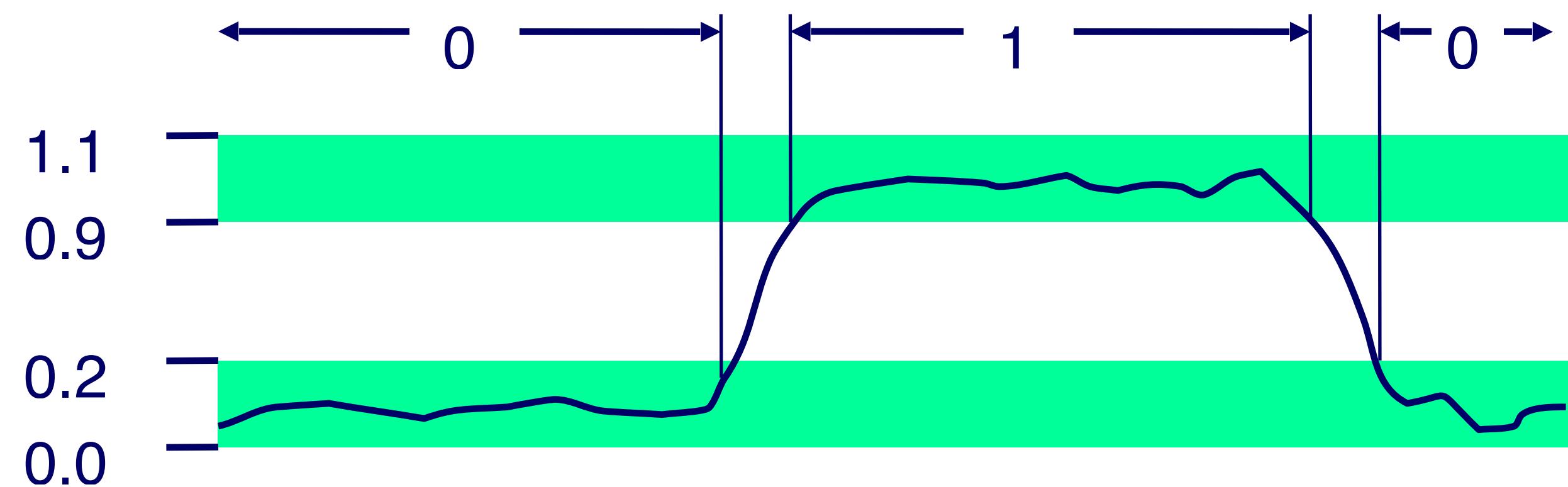
Claude Shannon -  
Information theory

# Before bits

- By 1948 more than 125 million conversations passed daily through the Bell System's 138 million miles of cable and 31 million telephone sets.
- Count words, characters, electricity consumption, minutes.

# Why bits?

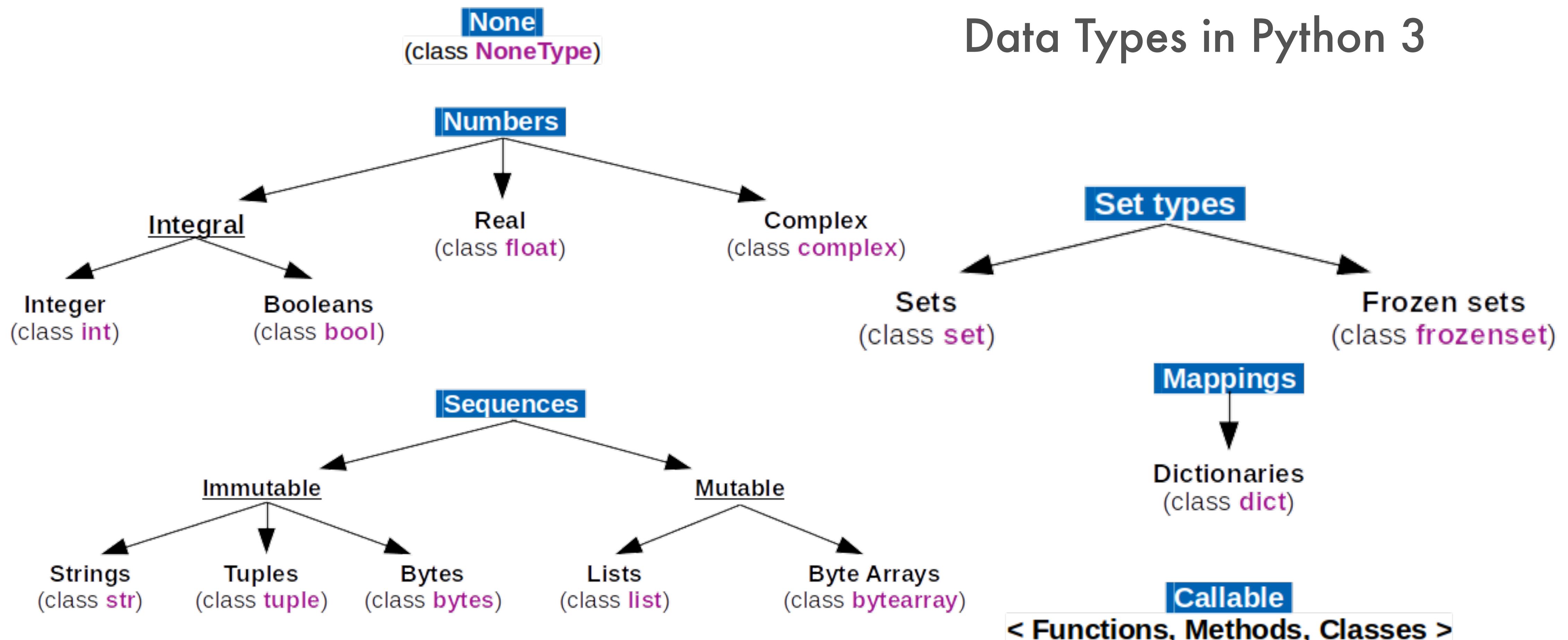
- Physics! Electronic Implementation
  - Easy to store with bistable elements.
    - e.g., high-low/off-on electromagnetism on disk.
  - Reliably transmitted on noisy and inaccurate wires
- Very expressive and efficient.
- Quantum computers?



# Digital Representation of Data

- **Bits:** All digital data are sequences of 0 & 1 (binary digits)
- **Data type:** First layer of abstraction to interpret a bit sequence with a human-understandable category of information; interpretation fixed by the PL
  - Example common datatypes: Boolean, Byte, Integer, “floating point” number (Float), Character, and String
- **Data structure:** A second layer of abstraction to organize multiple instances of same or varied data types as a more complex object with specified properties
  - Examples: Array, Linked list, Tuple, Graph, etc.

# Digital Representation of Data



# Count everything in binary

- **Base 2 Number Representation**
  - 0, 1, 10, 11, 100, 101, ...
  - Represent  $15213_{10}$  as 0011 1011 0110 1101<sub>2</sub>
  - Represent  $1.20_{10}$  as 1.0011 0011 0011 0011 [0011]...<sub>2</sub>
  - Represent  $(1.5213 \times 10^4)_{10}$  as  $(1.1101\ 1011\ 0110\ 1 \times 2^{13})_2$
- **Represent negative numbers as ...?**
  - (we'll come back to this)

# Encoding Byte Values

- **Byte = 8 bits. Why?**

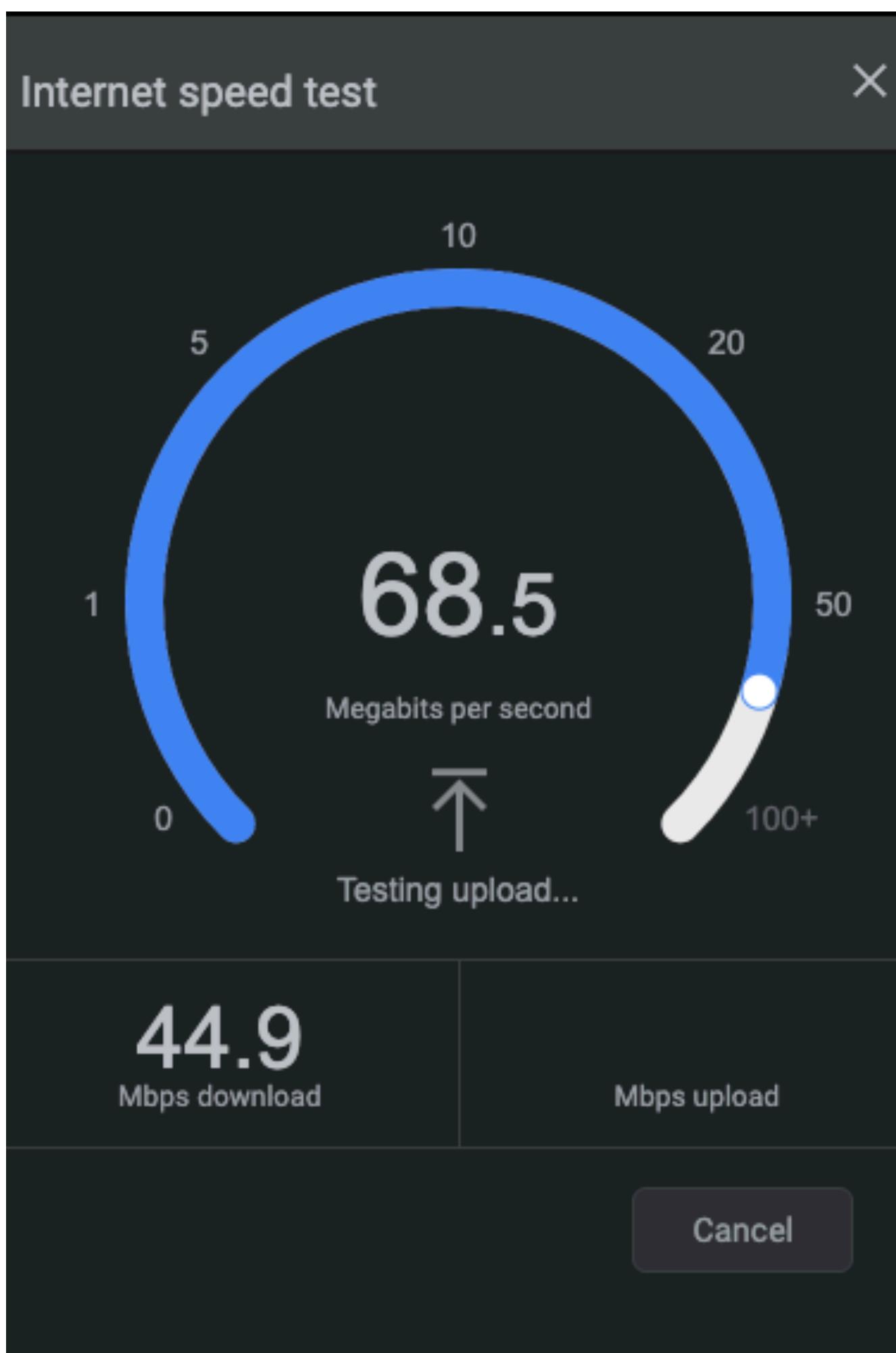
- Decimal:  $0_{10}$  to  $255_{10}$ 
  - $255 = 2^8 - 1$
- Binary:  $0000\ 0000_2$  to  $1111\ 1111_2$
- *Hexadecimal*:  $00_{16}$  to  $FF_{16}$ .
  - Why? Example: #FF5733, RGB 255, 87, 51
  - Base 16 number representation
  - Use characters '0' to '9' and 'A' to 'F'
  - Write in C with leading '0x', either case

- $0101\ 1010_2 = 0x5a = 0x5A = 0X5a$

15213: 0011 1011 0110 1101  
3 B 6 D

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

# Mbps v.s KB



Name	Size	Kind
HB50 cupcakes.JPG	2 MB	JPEG image
Roller Skating.JPG	1.3 MB	JPEG image
50HBJukebox2.jpg	720 KB	JPEG image
Facebook.tiff	399 KB	TIFF image
7_days_to_enrol.png	173 KB	PNG image
JoggingShoes.jpg	71 KB	JPEG image

# Digital Representation of Data

- The size and *interpretation* of a data type depends on PL
- A **Byte** (B; 8 bits) is typically the basic unit of data types
  - CPU can't address anything smaller than a byte.
- **Boolean:**
  - Examples in data sci.: Y/N or T/F responses
  - Just 1 bit needed but actual size is almost always 1B, i.e., 7 bits are wasted! (**Q:** Why?)
- **Integer:**
  - Examples in data science: #friends, age, #likes
  - Typically 4 bytes; many variants (short, unsigned, etc.)
  - Java *int* can represent  $-2^{31}$  to  $(2^{31} - 1)$ ; C *unsigned int* can represent 0 to  $(2^{32} - 1)$ ;  
Python3 *int* is effectively unlimited length (PL magic!)

# Combine bytes to make *scalar data types*

C Data Type	Size (# of bytes)	
	Typical 32-bit	Typical 64-bit
<b>char</b>	1	1
<b>short</b>	2	2
<b>int</b>	4	4
<b>long</b>	4	8
<b>float</b>	4	4
<b>double</b>	8	8
<b>pointer</b>	4	8

“ILP32”                    “LP64”

Estimate the object size in  
your memory, network,  
storage!

# Design considerations

- Human
- Machine
  - Storage
  - Computation

# Digital Representation of Data

**Q:** How many unique data items can be represented by 3 bytes?

- Given  $k$  bits, we can represent  $2^k$  unique data items
- 3 bytes = 24 bits  $\Rightarrow 2^{24}$  items, i.e., 16,777,216 items
- Common approximation:  $2^{10}$  (i.e., 1024)  $\sim 10^3$  (i.e., 1000); recall kibibyte (KiB) vs kilobyte (KB) and so on

**Q:** How many bits are needed to distinguish 97 data items?

- For  $k$  unique items, invert the exponent to get  $\log_2(k)$
- But #bits is an integer! So, we only need  $\lceil \log_2(k) \rceil$
- So, we only need the next higher power of 2
- $97 \rightarrow 128 = 2^7$ ; so, 7 bits

# Digital Representation of Data

**Q:** How to convert from decimal to binary representation?

- Given decimal n, if power of 2 (say,  $2^k$ ), put 1 at bit position k; if  $k=0$ , stop; else pad with trailing 0s till position 0
- If n is not power of 2, identify the power of 2 just below n (say,  $2^k$ ); #bits is then k; put 1 at position k
- Reset n as  $n - 2^k$ ; return to Steps 1-2
- Fill remaining positions in between with 0s

	7	6	5	4	3	2	1	0	Position/Exponent of 2
Decimal	128	64	32	16	8	4	2	1	Power of 2
$5_{10}$						1	0	1	
$47_{10}$				1	0	1	1	1	
$163_{10}$	1	0	1	0	0	0	1	1	
$16_{10}$				1	0	0	0	0	

**Q:** Binary to decimal?

# Digital Representation of Data

- *Hexadecimal representation is a common stand-in for binary representation; more succinct and readable*
  - Base 16 instead of base 2 cuts display length by  $\sim 4x$
  - Digits are 0, 1, ... 9, A ( $10_{10}$ ), B, ... F ( $15_{10}$ )
  - From binary: combine 4 bits at a time from lowest

Decimal	Binary	Hexadecimal	
$5_{10}$	$101_2$	$5_{16}$	Alternative notations
$47_{10}$	$10\ 1111_2$	$2F_{16}$	
$163_{10}$	$1010\ 0011_2$	$A3_{16}$	$0xA3$ or $A3_H$
$16_{10}$	$1\ 0000_2$	$10_{16}$	

# Encoding Integers

## Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

## Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

Sign Bit

```
short int x = 15213;  
short int y = -15213;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
y	-15213	C4 93	11000100 10010011

## Two-complement: Simple Example

$$\begin{array}{rccccc} & -16 & 8 & 4 & 2 & 1 \\ 10 = & 0 & 1 & 0 & 1 & 0 \end{array} \quad 8+2 = 10$$

$$\begin{array}{rccccc} & -16 & 8 & 4 & 2 & 1 \\ -10 = & 1 & 0 & 1 & 1 & 0 \end{array} \quad -16+4+2 = -10$$

# Two-complement Encoding Example (Cont.)

<b>x =</b>	15213:	00111011	01101101
<b>y =</b>	-15213:	11000100	10010011

Weight	15213	-15213	Sum	15213	-15213
1	1	1		1	1
2	0	0		1	2
4	1	4		0	0
8	1	8		0	0
16	0	0		1	16
32	1	32		0	0
64	1	64		0	0
128	0	0		1	128
256	1	256		0	0
512	1	512		0	0
1024	0	0		1	1024
2048	1	2048		0	0
4096	1	4096		0	0
8192	1	8192		0	0
16384	0	0		1	16384
-32768	0	0		1	-32768

# Numeric Ranges

- Unsigned Values

- $UMin = 0$   
• 000...0
- $UMax = 2^w - 1$   
• 111...1

- Two's Complement Values

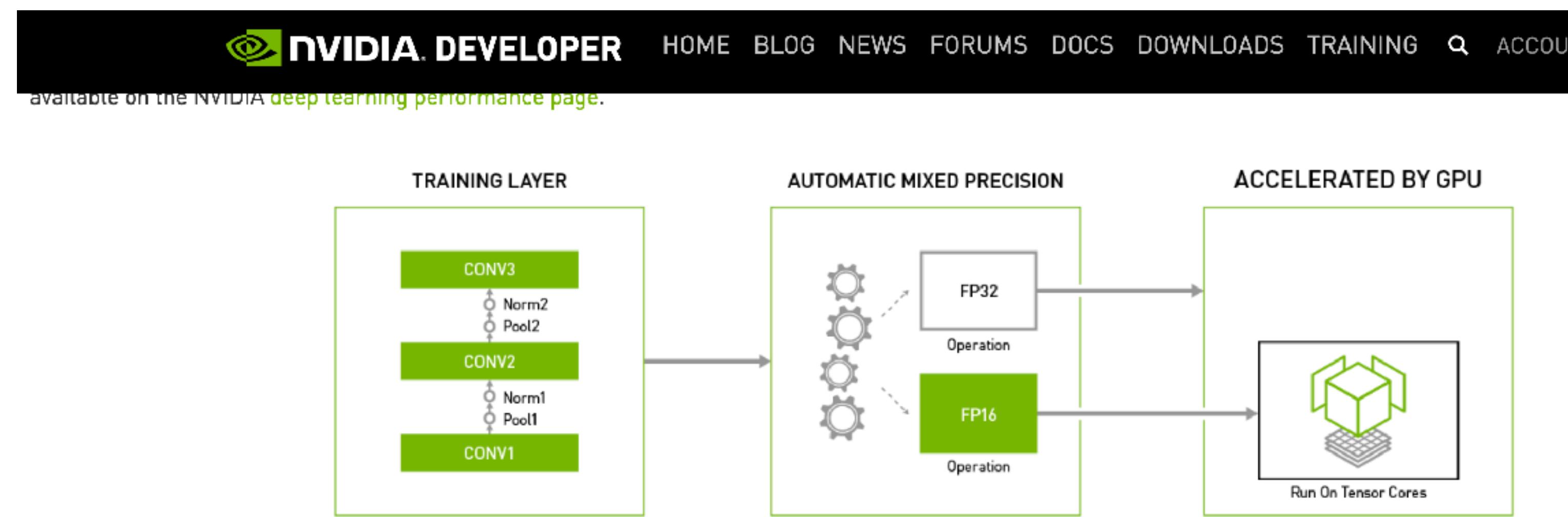
- $TMin = -2^{w-1}$   
100...0
- $TMax = 2^{w-1} - 1$   
011...1
- Minus 1  
111...1

## Values for $W = 16$

	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

# Digital Representation of Data

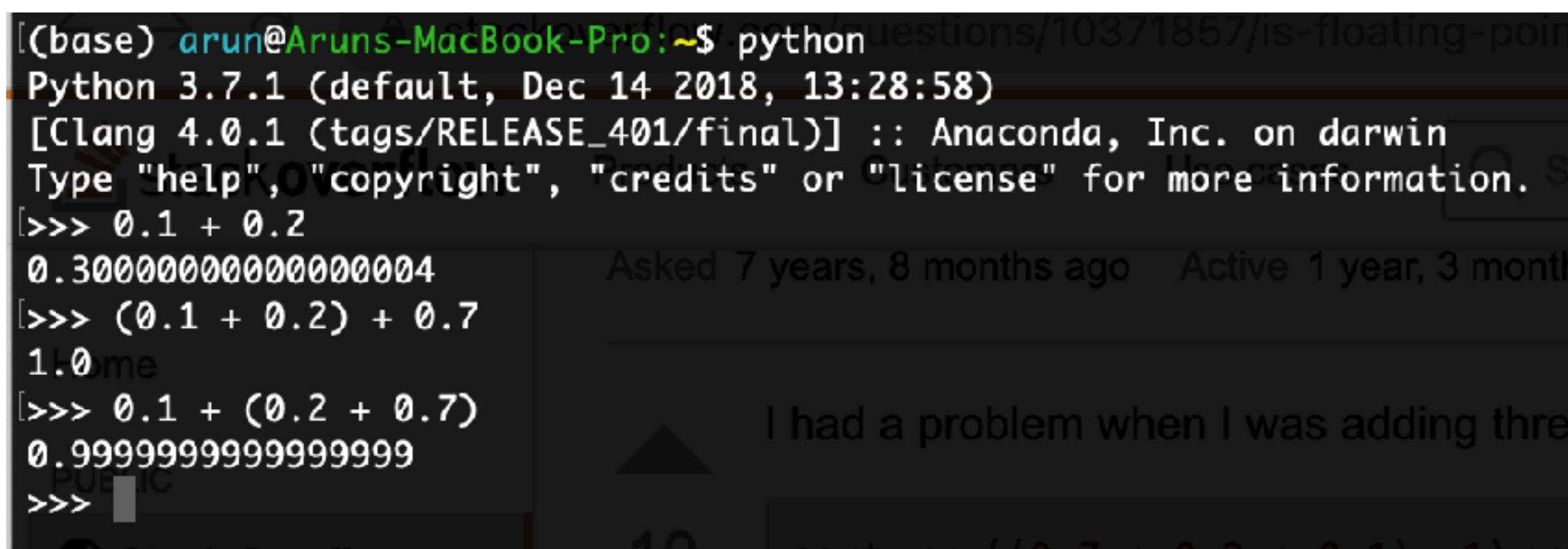
- **Float:**
  - Examples in data sci.: salary, scores, model weights
  - IEEE-754 single-precision format is 4B long; double-precision format is 8B long
  - Java and C *float* is single; Python *float* is double!



Using Automatic Mixed Precision for Major Deep Learning Frameworks

# Digital Representation of Data

- Due to representation imprecision issues, floating point arithmetic (addition and multiplication) is not associative!



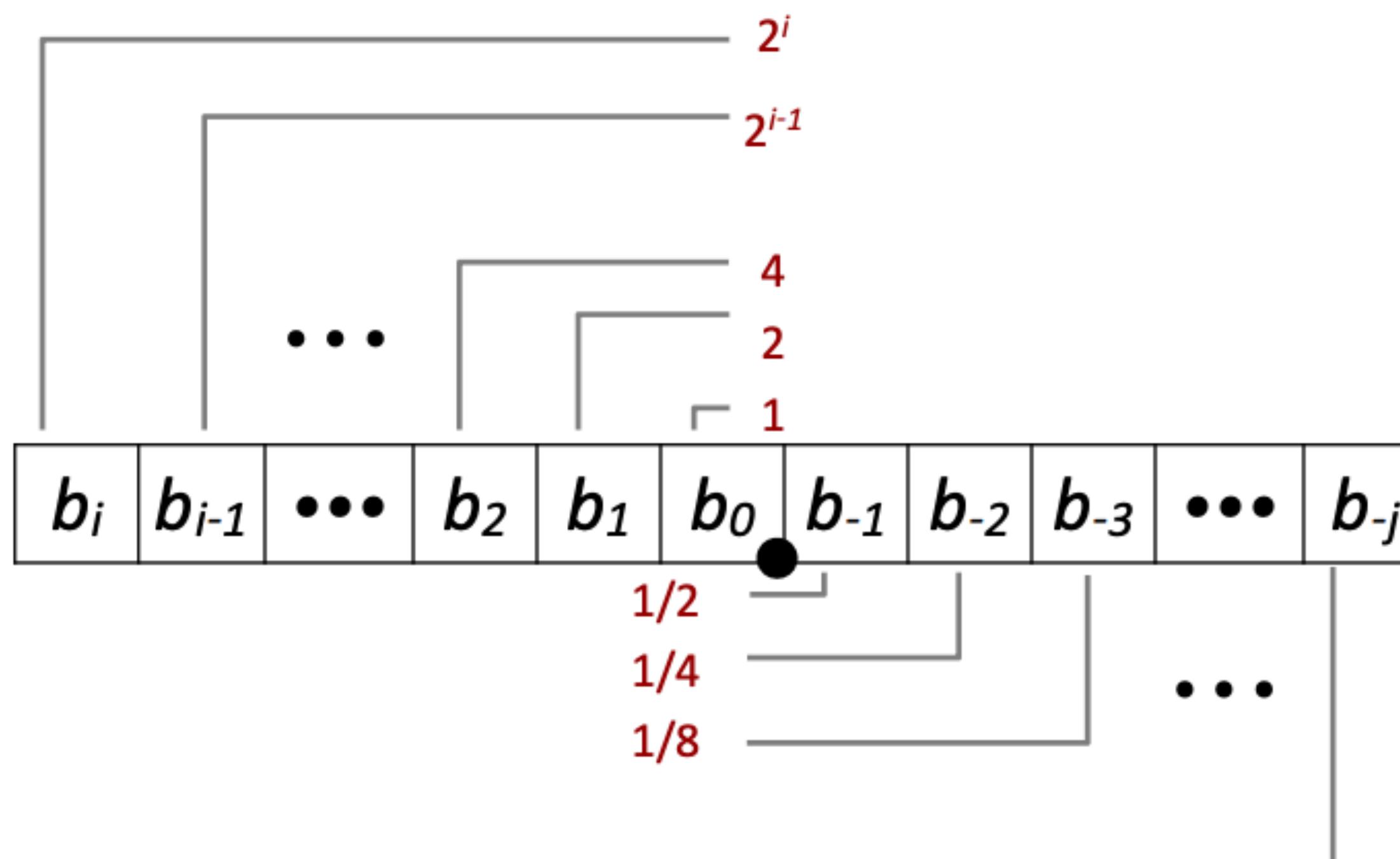
A screenshot of a Stack Overflow question titled "Is floating-point arithmetic associative?". The question was asked 7 years, 8 months ago and has been active for 1 year, 3 months. The user, "Arunkumar", describes a problem with adding three floating-point numbers. The Python code in the question shows:

```
[base] arun@Aruns-MacBook-Pro:~$ python
Python 3.7.1 (default, Dec 14 2018, 13:28:58)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> 0.1 + 0.2
0.3000000000000004
[>>> (0.1 + 0.2) + 0.7
1.0
[>>> 0.1 + (0.2 + 0.7)
0.9999999999999999
>>>
```

- In binary32, special encodings recognized:
  - Exponent 0xFF and fraction 0 is +/- “Infinity”
  - Exponent 0xFF and fraction <> 0 is “NaN”
  - Max is ~  $3.4 \times 10^{38}$ ; min +ve is ~  $1.4 \times 10^{-45}$

# A strawman solution

## Fractional Binary Numbers



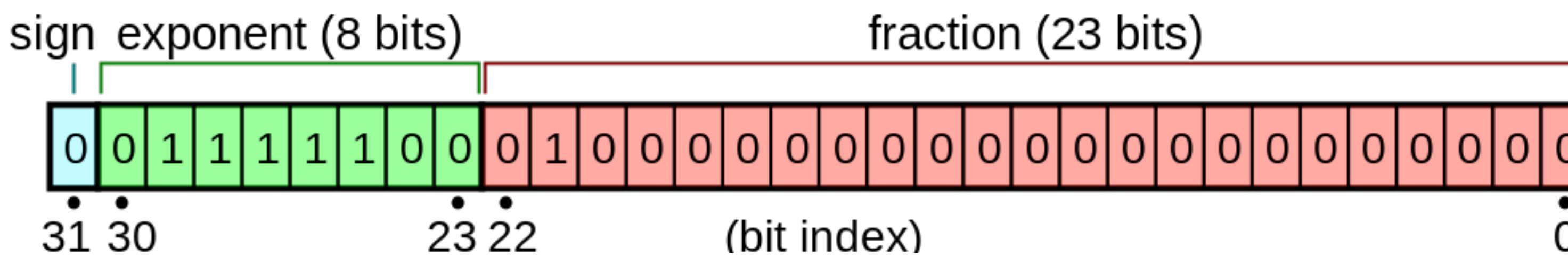
### ■ Representation

- Bits to right of “binary point” represent fractional powers of 2
- Represents rational number:

$$\sum_{k=-j}^i b_k \times 2^k$$

# The commonly adopted solution

- **Float:**
  - Standard IEEE format for single (aka binary32):



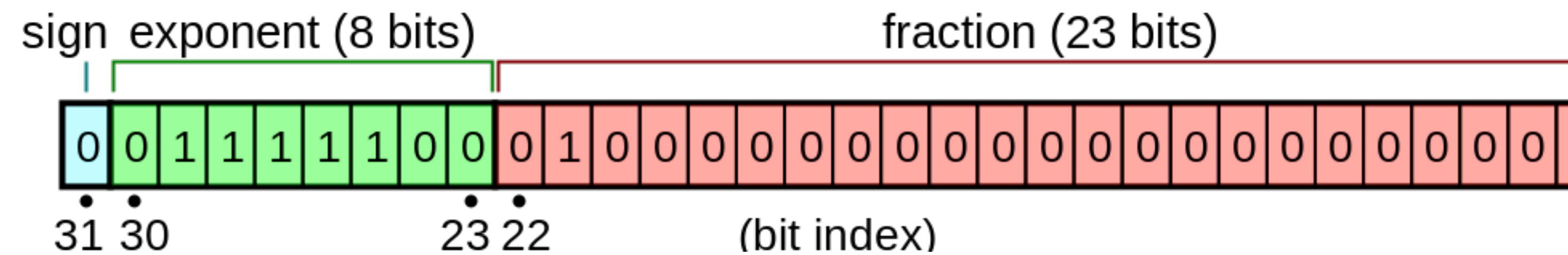
$$(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i} 2^{-i})$$

(NB: Converting decimal reals/fractions to float is NOT in syllabus!)

# High-level intuition.

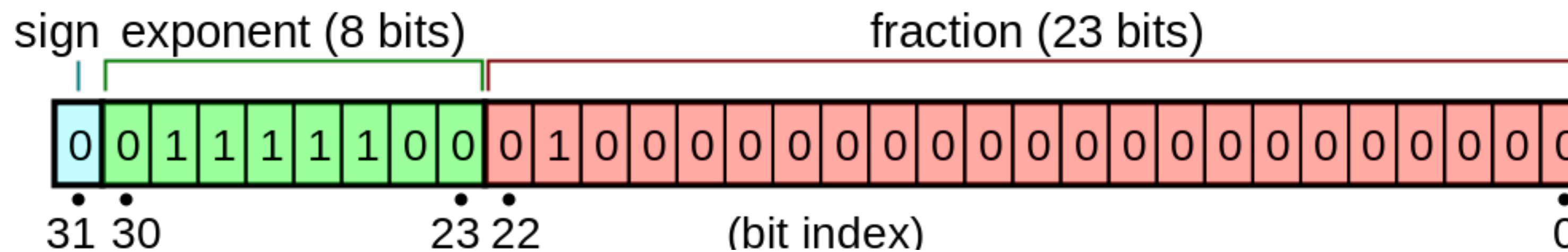
- Two extremes:
  - 0.000000000123455666
  - 99999999999999.0
- Key idea 1: turn them into scientific notion.
  - 1.23455666\*10<sup>-10</sup>
  - 9.9999999999999\*10<sup>13</sup>
- Key idea 2: Represent these two components ( $10^X$ ) and ( $Y.Z$ ) separately.

Many concepts in data science is an invention rather than a discovery.



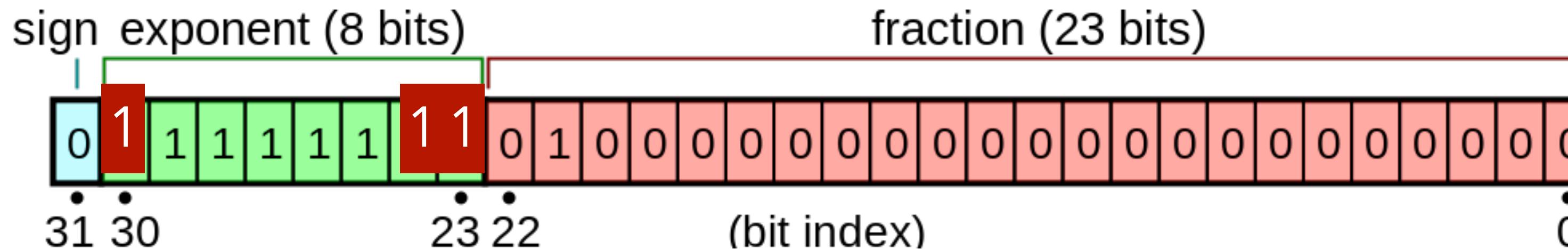
# How can we represent a number in float?

- Key idea 1: turn them into scientific notion.
  - $1.23455666 \times 10^{-10}$
  - $9.99999999999999 \times 10^{13}$
- Key idea 2: Represent these two components ( $10^X$ ) and ( $Y.Z$ ) separately.



- Key idea 3: Everything is binary.  $10^X \Rightarrow 2^W$

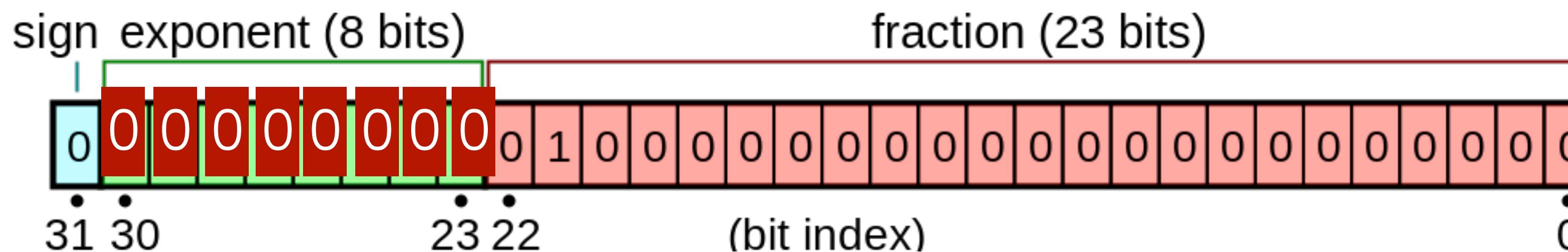
# Examples



$$(-1)^{sign} \times 2^{exponent-127}$$

- Exponent =  $(11111111)_2 = 255$
  - $2^{(255-127)} = 2^{128}$

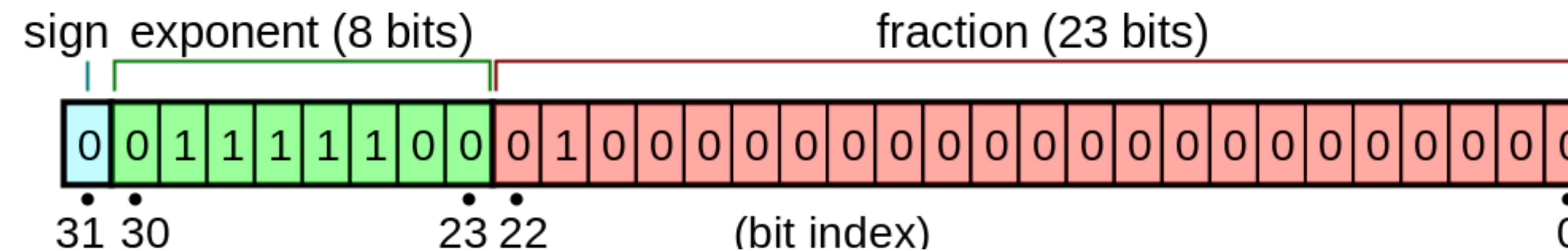
# Examples



$$(-1)^{sign} \times 2^{exponent-127}$$

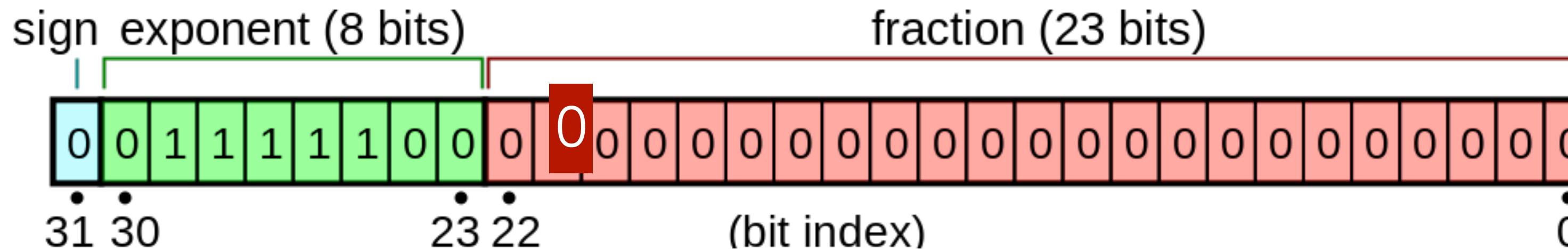
- **Exponent = (0000000)<sub>2</sub> = 0**
- $2^{(0-127)} = 2^{-127}$
- \* This is a simplified version.

# Examples



- $1 < \text{Mantissa} < 2$
- Why 23? What is i?
- $1 + 1/2 + 1/4$

# Examples

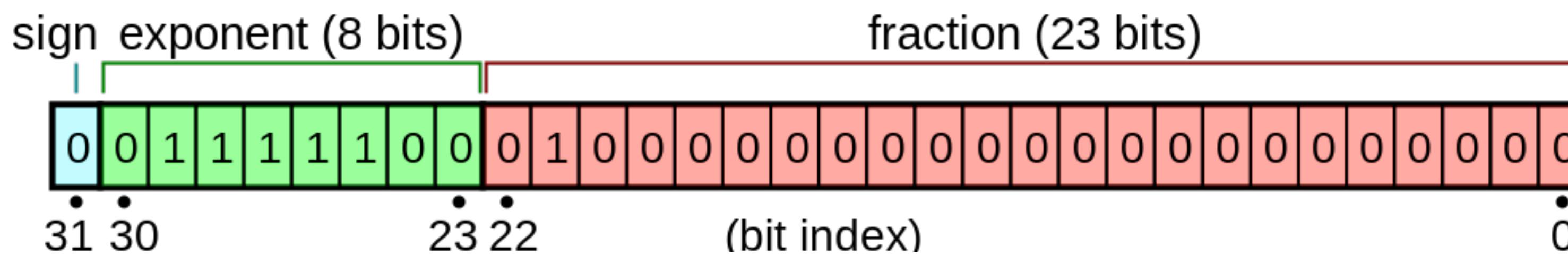


$$(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i} 2^{-i})$$

- $1.0 \leq \text{Mantissa} < 2$
  - Why 23? What is i?
  - How about the biggest number?

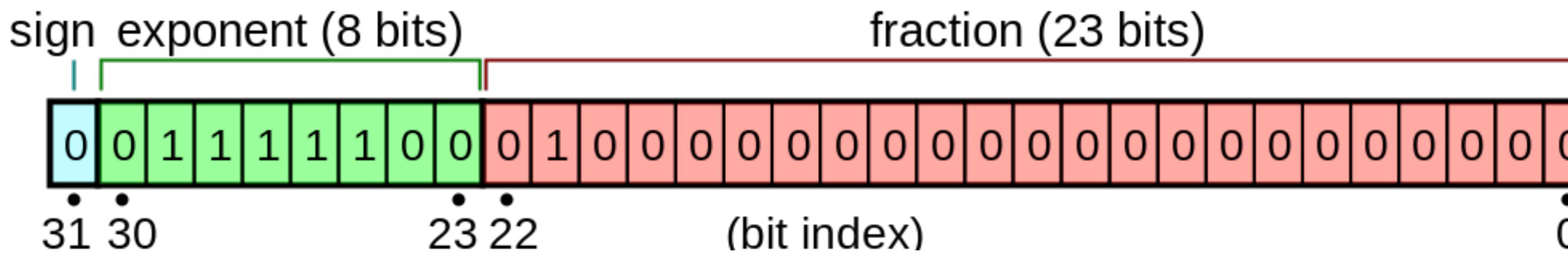
# Digital Representation of Data

- **Float:**
  - Standard IEEE format for single (aka binary32):



$$(-1)^{sign} \times 2^{exponent-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

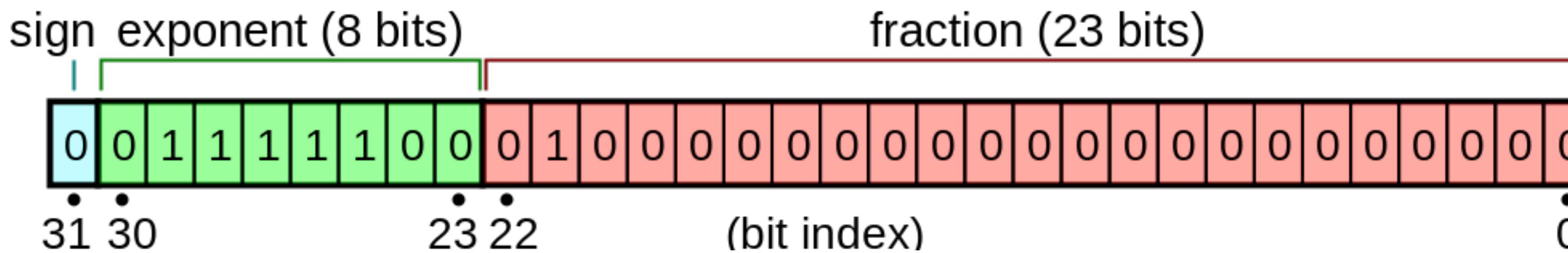
# Digital Representation of Data.



$$(-1)^{sign} \times 2^{exponent-127} \times (1 + \sum_{i=1}^{23} b_{23-i} 2^{-i})$$

- Special encodings recognized:
  - Exponent 0x00 and fraction 0 is “+/- 0”.
    - 0 00000000 0000000000000000000000 = +0
  - Exponent 0xFF and fraction 0 is +/- “Infinity”
    - 0 1111111 0000000000000000000000 = +INF
  - Not a number/Undefined (Dividing 0 by 0)
    - 0 1111111 1000000000000000000000 = NaN

# Digital Representation of Data. Why?

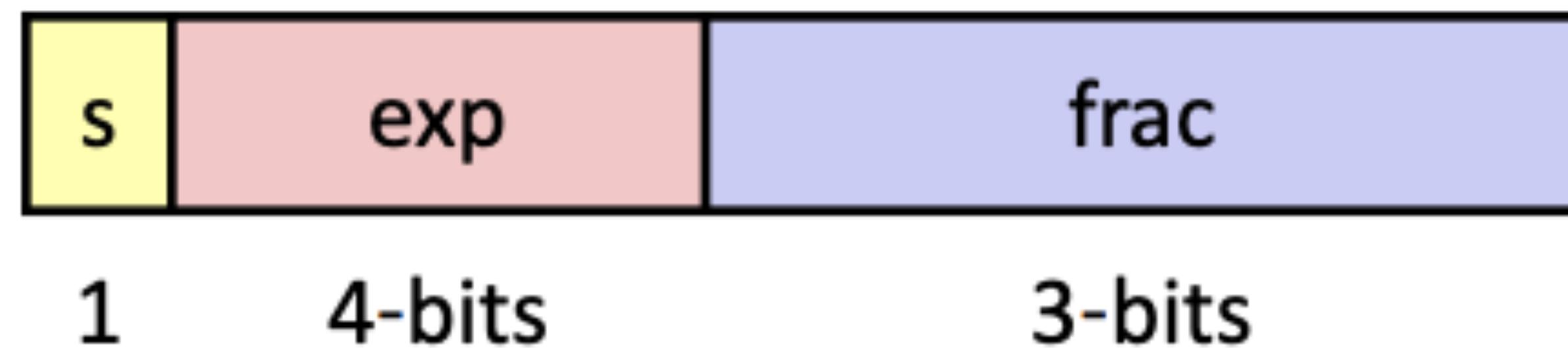


- Exponent for Range, Fraction (significant) for Accuracy.
  - Max: 0 1111110 11111111111111111111111 =>
  - $2^{(254-127)} * (1.111111\dots)_2 = 2^{(254-127)} * \left(1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{8388608}\right)$
  - Max =  $2 * 2^{127} = 2^{128} = 3.4 \times 10^{38}$
  - Why not 0 1111111 11111111111111111111111?
- Scientific notation.
- Dynamic accuracy.

# Review

- Wide range of values
- Precision
- Compatibility
- Easy to use
- Complexity
- Rounding errors
- Speed.
- Limitate precisions

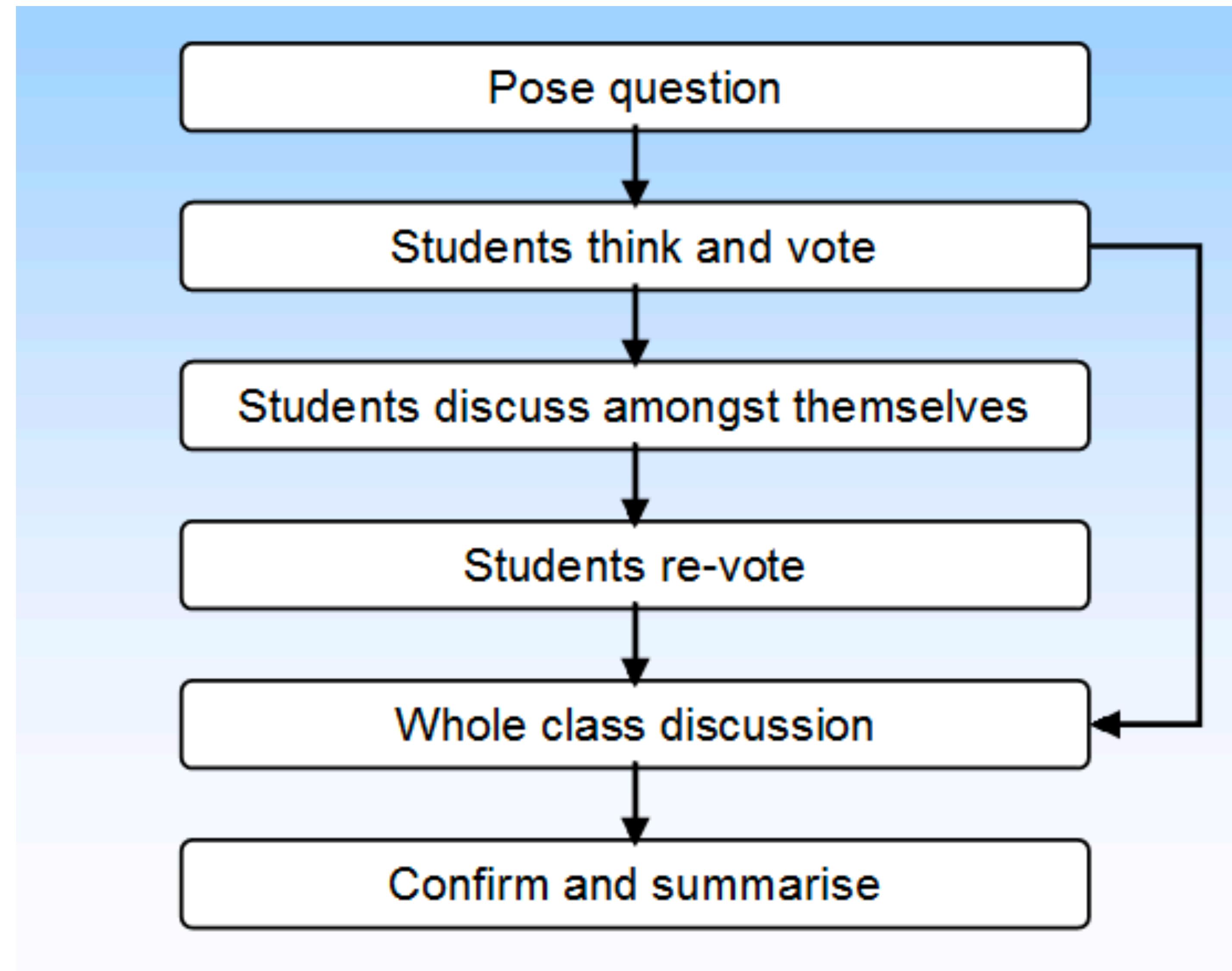
# Examples in the final exam?



# Digital Representation of Data

- More float standards: double-precision (float64; 8B) and half-precision (float16; 2B); different #bits for exponent, fraction
- Float16 is now common for deep *learning* parameters:
  - Native support in PyTorch, TensorFlow, etc.; APIs also exist for weight quantization/rounding post training
  - NVIDIA Deep Learning SDK support mixed-precision training; 2-3x speedup with similar accuracy!
  - New processor hardware (FPGAs, ASICs, etc.) enable arbitrary precision, even 1-bit (!), but accuracy is lower

# Peer instruction activity



## Peer Instruction Activity

Rough time tip: About 1 min per 1 pt.

Use your UCSD email

# Digital Representation in Python

```
import sys

i = sys.maxsize
print(i)
# 9223372036854775807
print(i == i + 1)
# False
i += 1
print(i)
# 9223372036854775808

f = sys.float_info.max
print(f)
# 1.7976931348623157e+308
print(f == f + 1)
# True
f += 1
print(f)
# 1.7976931348623157e+308
```

```
import numpy as np

a = np.array([3095693933], dtype=int)
s = np.sum(a)
print(s)
# 3095693933
s * s
# -8863423146896543127
print(type(s))
# numpy.int64
py_s = int(s)
py_s * py_s
# 9583320926813008489
```

In Python, Integers are implemented as “long”.  
In NumPy, Integers are implemented as np.int64.

# Digital Representation in Python

```
# using array-scalar type
import numpy as np
dt = np.dtype(np.int32)
print dt
```

Sr.No.	Data Types & Description
1	<b>bool_</b> Boolean (True or False) stored as a byte
2	<b>int_</b> Default integer type (same as C long; normally either int64 or int32)
3	<b>intc</b> Identical to C int (normally int32 or int64)
4	<b>intp</b> Integer used for indexing (same as C ssize_t; normally either int32 or int64)
5	<b>int8</b> Byte (-128 to 127)
6	<b>int16</b> Integer (-32768 to 32767)

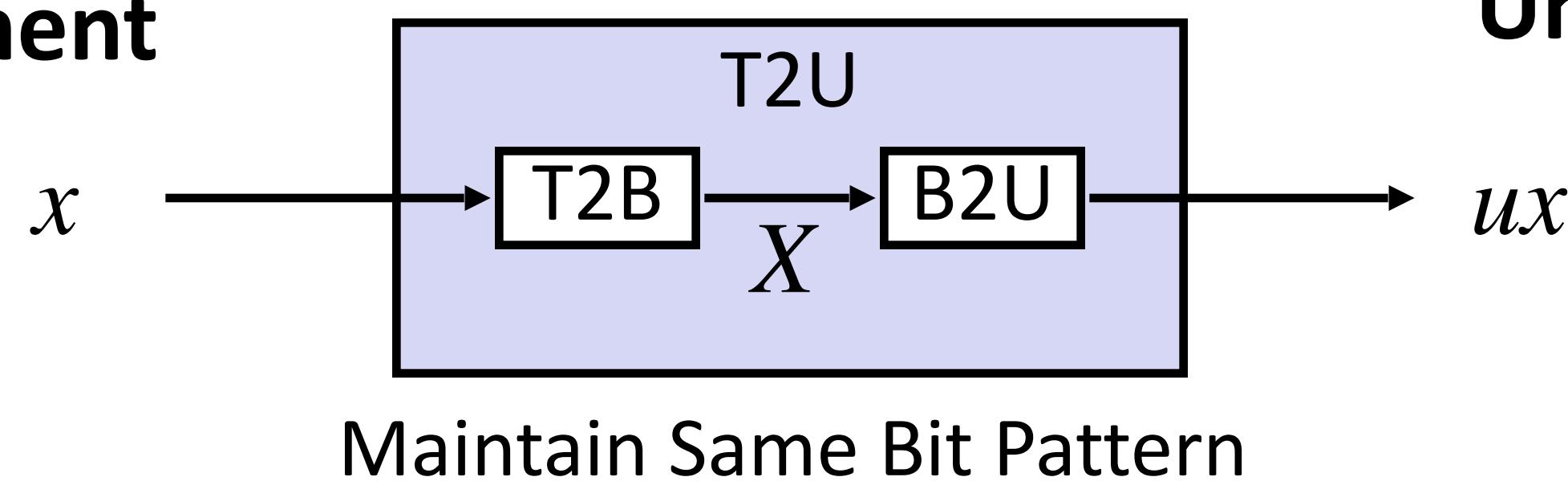
# Unsigned & Signed Numeric Values

- **Equivalence**
  - Same encodings for nonnegative values
- **Uniqueness**
  - Every bit pattern represents unique integer value
  - Each representable integer has unique bit encoding
- $\Rightarrow$  **Can Invert Mappings**
  - $U2B(x) = B2U^{-1}(x)$ 
    - Bit pattern for unsigned integer
  - $T2B(x) = B2T^{-1}(x)$ 
    - Bit pattern for two's comp integer

$X$	$B2U(X)$	$B2T(X)$
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

# Mapping Between Signed & Unsigned

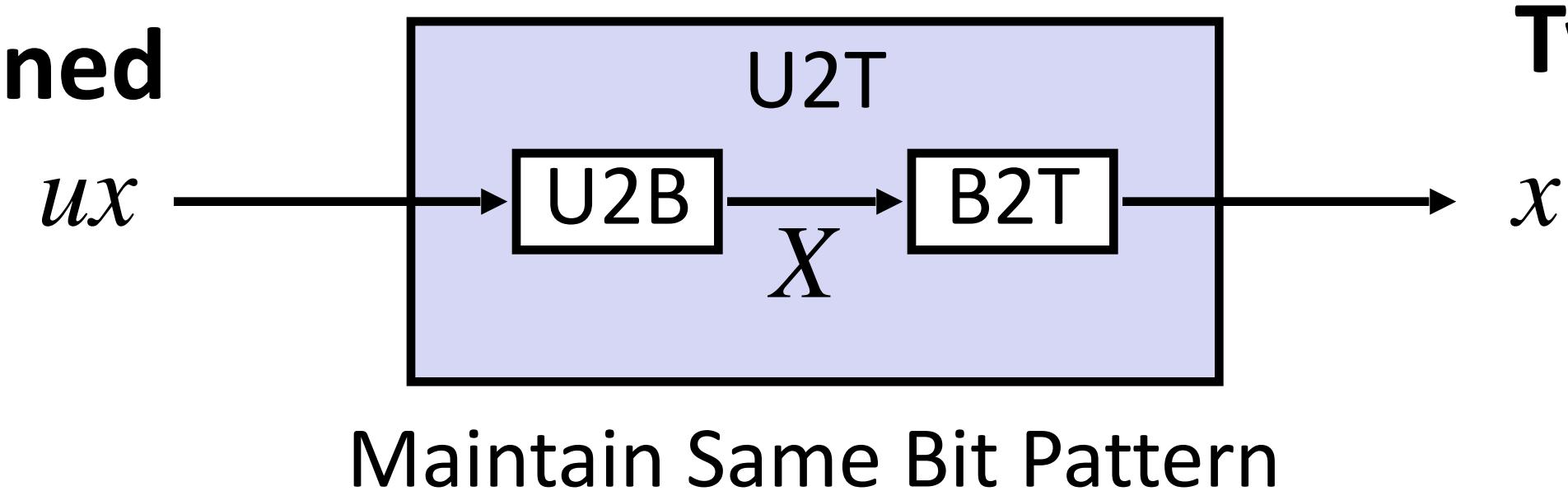
**Two's Complement**



**Unsigned**

Maintain Same Bit Pattern

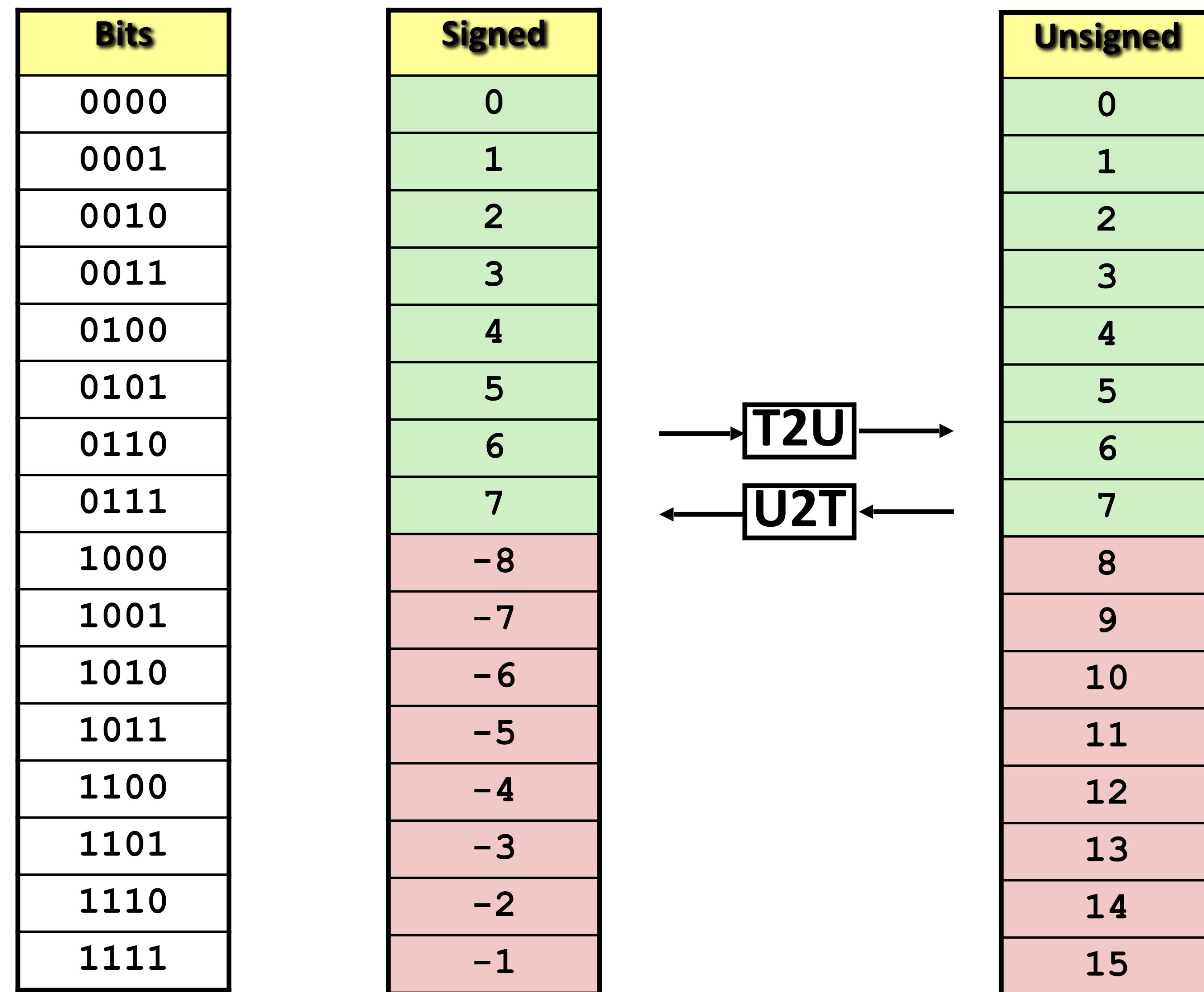
**Unsigned**



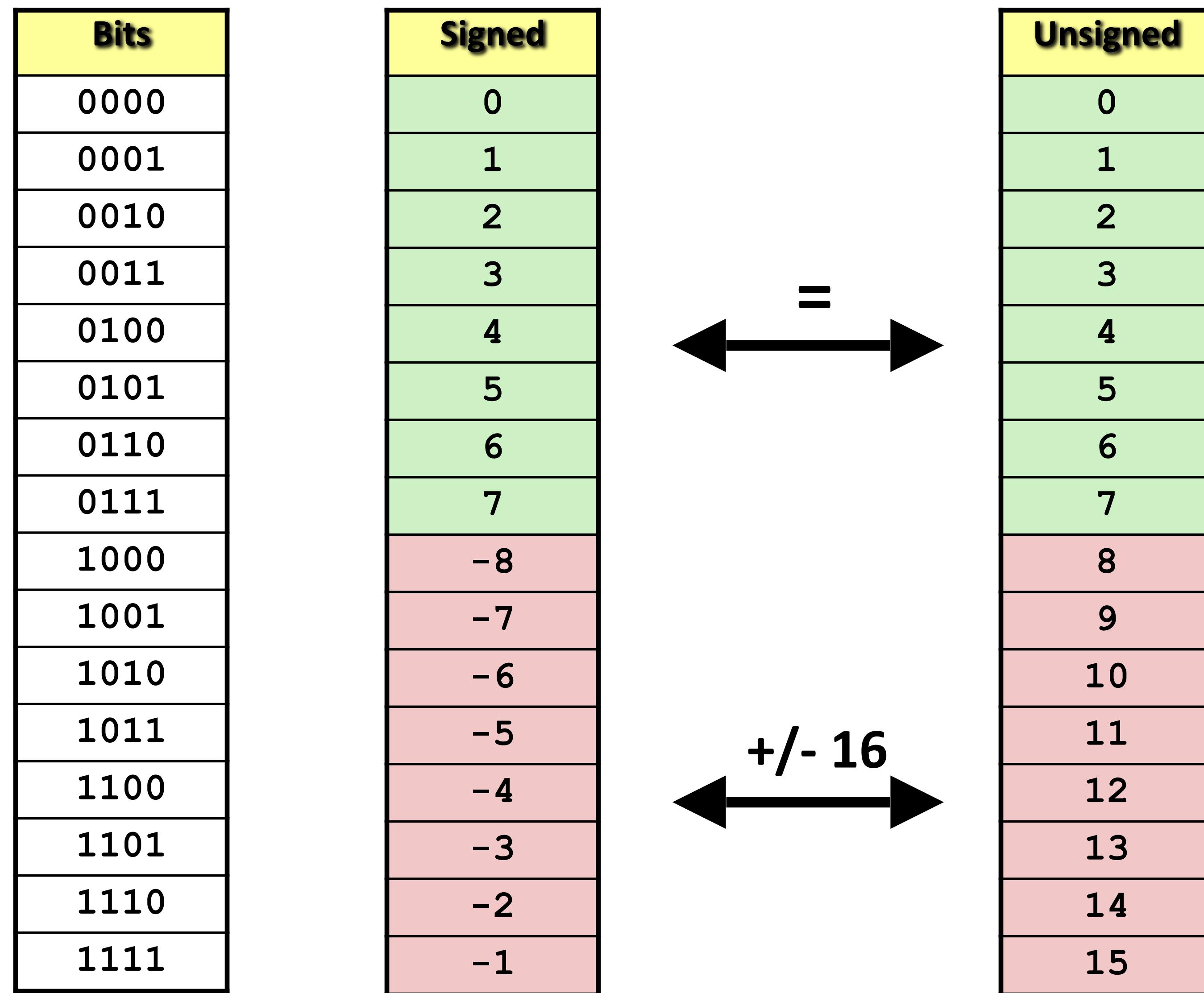
**Two's Complement**

Maintain Same Bit Pattern

# Mapping Signed ↔ Unsigned

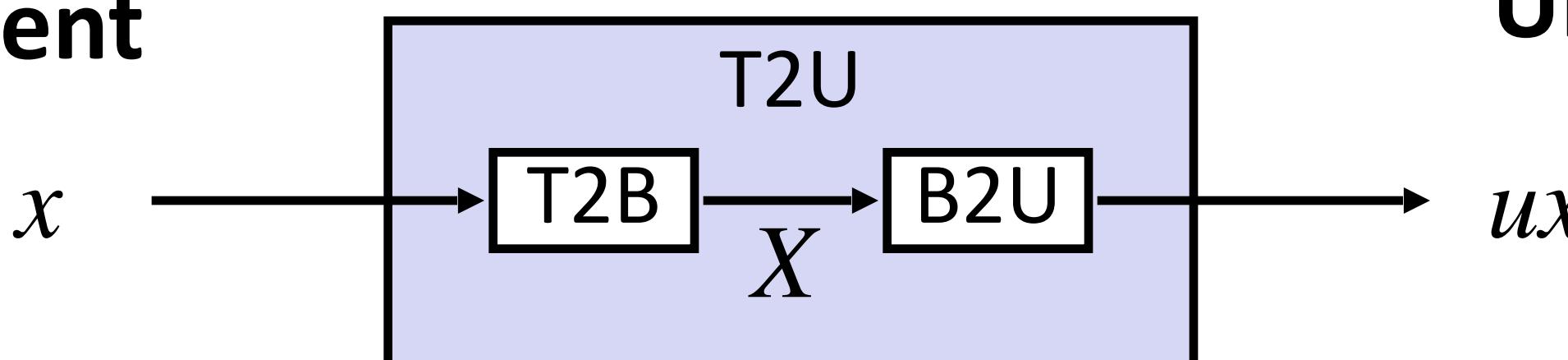


# Mapping Signed $\leftrightarrow$ Unsigned



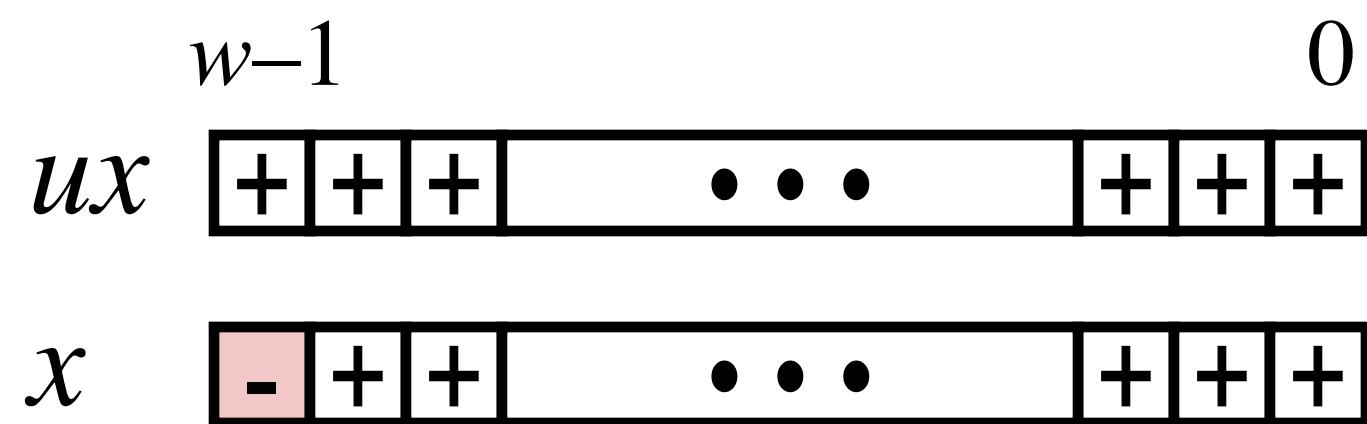
# Relation between Signed & Unsigned

**Two's Complement**



**Unsigned**

Maintain Same Bit Pattern



**Large negative weight  
becomes  
Large positive weight**

# Review the Examples

<b>x =</b>	15213:	00111011	01101101
<b>y =</b>	-15213:	11000100	10010011

**x+y = ?**

Weight	15213	-15213	Sum	15213	-15213
1	1	1	1	1	1
2	0	0	0	1	2
4	1	4	0	0	0
8	1	8	0	0	0
16	0	0	1	1	16
32	1	32	0	0	0
64	1	64	0	0	0
128	0	0	1	1	128
256	1	256	0	0	0
512	1	512	0	0	0
1024	0	0	1	1	1024
2048	1	2048	0	0	0
4096	1	4096	0	0	0
8192	1	8192	0	0	0
16384	0	0	1	1	16384
-32768	0	0	1	1	-32768

# Digital Representation of Data

```
void show_squares()
{
    int x;
    for (x = 5; x <= 5000000; x*=10)
        printf("x = %d x^2 = %d\n", x, x*x);
}
```

x = 5 x<sup>2</sup> = 25

x = 50 x<sup>2</sup> = 2500

x = 500 x<sup>2</sup> = 250000

x = 5000 x<sup>2</sup> = 25000000

x = 50000 x<sup>2</sup> = -1794967296

x = 500000 x<sup>2</sup> = 891896832

x = 5000000 x<sup>2</sup> = -1004630016

In C, int uses a 32-bit

# Character and String

- Representing **Character (char)** and **String**:
  - Letters, numerals, punctuations, etc.
  - A string is typically just a variable-sized array of char
  - C *char* is 1B; Java *char* is 2B; Python does not have a char type (use *str* or *bytes*)
  - American Standard Code for Information Interchange (ASCII) for encoding characters; initially 7-bit; later extended to 8-bit
    - Examples: ‘A’ is 61, ‘a’ is 97, ‘@’ is 64, ‘!’ is 33, etc.
  - *Unicode UTF-8* is now common, subsumes ASCII; 4B for ~1.1 million “code points” incl. many other language scripts, math symbols, emojis, etc. ☺

# Other type casting

```
string = "56"
number = 44

# Converting the string into an integer number.
string_number = int(string)

sum_of_numbers = number + string_number
print("The Sum of both the numbers is: ", sum_of_numbers)
```

```
The Sum of both the numbers is: 100
```

# Other type casting

```
string = "01101"
floating_number = 45.20

# Converting the string as well as integer number into floating point number.
new_str = int(string)
new_number = int(floating_number)
print("String converted into integer: ", new_str)
print("The floating-point number converted into integer: ", new_number)
```

```
String converted into integer: 1101
The floating-point number converted into integer: 45
```

# Data Type conversion examples

int(10)	=> 10
int(10.1)	=> 10
int('10')	=> 10
int('10.1')	=> Error
int('kitten')	=> Error

str(10)	=> '10'
str(10.1)	=> '10.1'
str('kitten')	=> 'kitten'

float(10)	=> 10.0
float(10.1)	=> 10.1
float('10')	=> 10.0
float('10.1')	=> 10.1
float('kitten')	=> Error

**serialization**

# Digital Representation of Data

- All digital objects are *collections* of basic data types (bytes, integers, floats, and characters)
  - SQL dates/timestamp: string (w/ known format)
  - ML feature vector: *array* of floats (w/ known length)
  - Neural network weights: *set* of multi-dimensional *arrays* (matrices or tensors) of floats (w/ known dimensions)
  - Graph: an *abstract data type* (ADT) with *set* of vertices (say, integers) and *set* of edges (*pair* of integers)
  - Program in PL, SQL query: string (w/ grammar)
  - Other data structures or digital objects?

# Review Questions

- Why do computers use binary digits?
- How many integers can you represent with 5 bits?
- How many bits do you need to represent 5 integers?
- What is the hexadecimal for  $20_{10}$ ?
- Why do we need a float standard?
- Why should a data scientist know about float formats?
- What does “lower precision” mean for a float weight in DL?
- Why is serialization needed?