Create Your First Program & a lot more

Special class

Love Babbar • Jan 28, 2023

Create your First Program

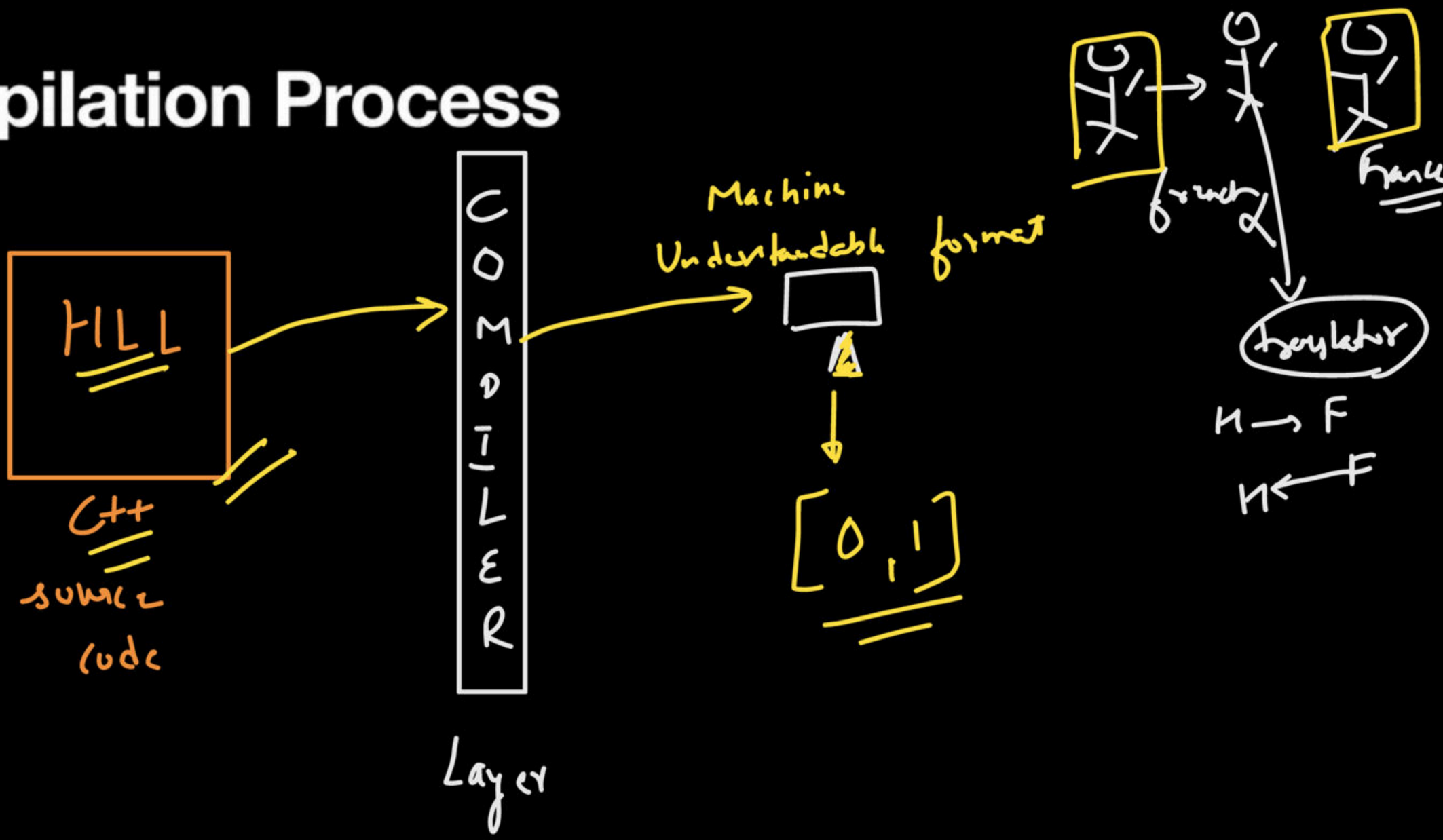
Instructor: Love Babbar

Programming Languages

Why we need it ?

- A Language using which, we can instruct the computer to carry out real life tasks and computations is called a programming language. It acts as a language in which we could easily express our thoughts to the machine.
- Like natural languages, programming language has a fixed set of rules according to which programs could be written in it. These programs are then converted into a language which machines can understand. This task is carried out by a special software called compiler.
- Every language has its own compiler/Interpreter.
- Once a program is compiled and linked, its executable is created and the computer can run our program now.

Compilation Process



IDE:

setup

↳ Replit

Code Editors

or

IDE's

↳ Code Blocks

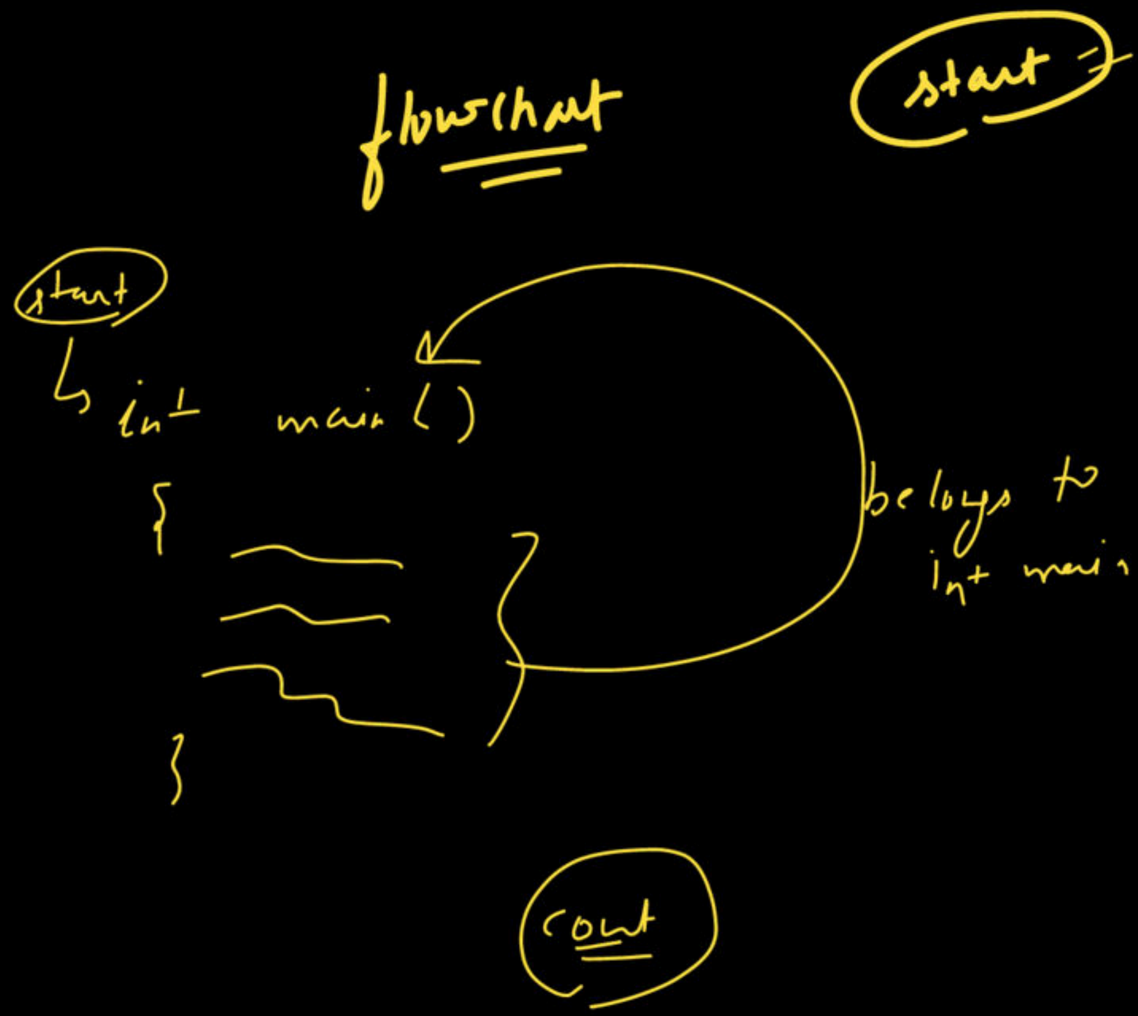
↳ VS Code

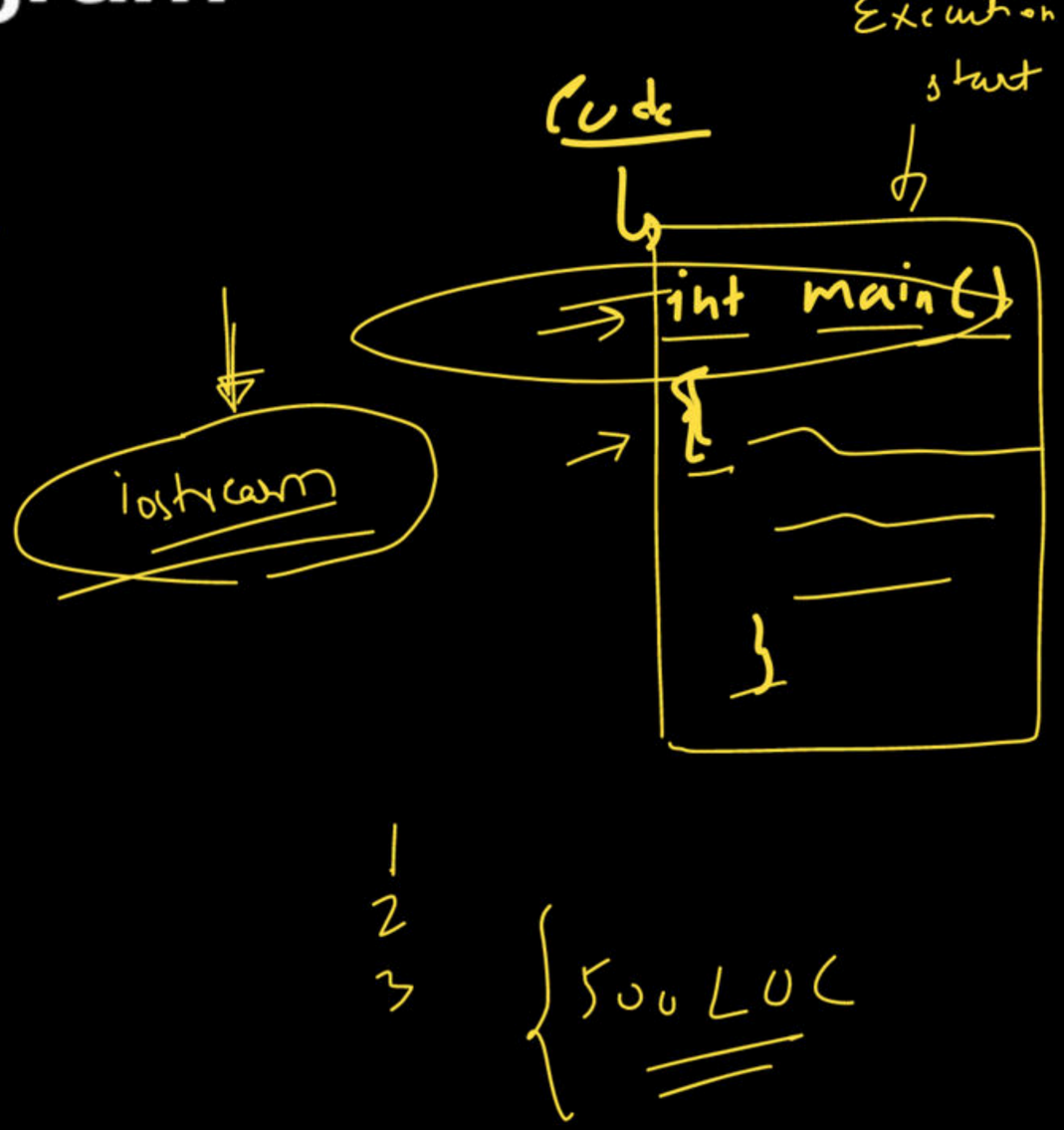
↳ Y code

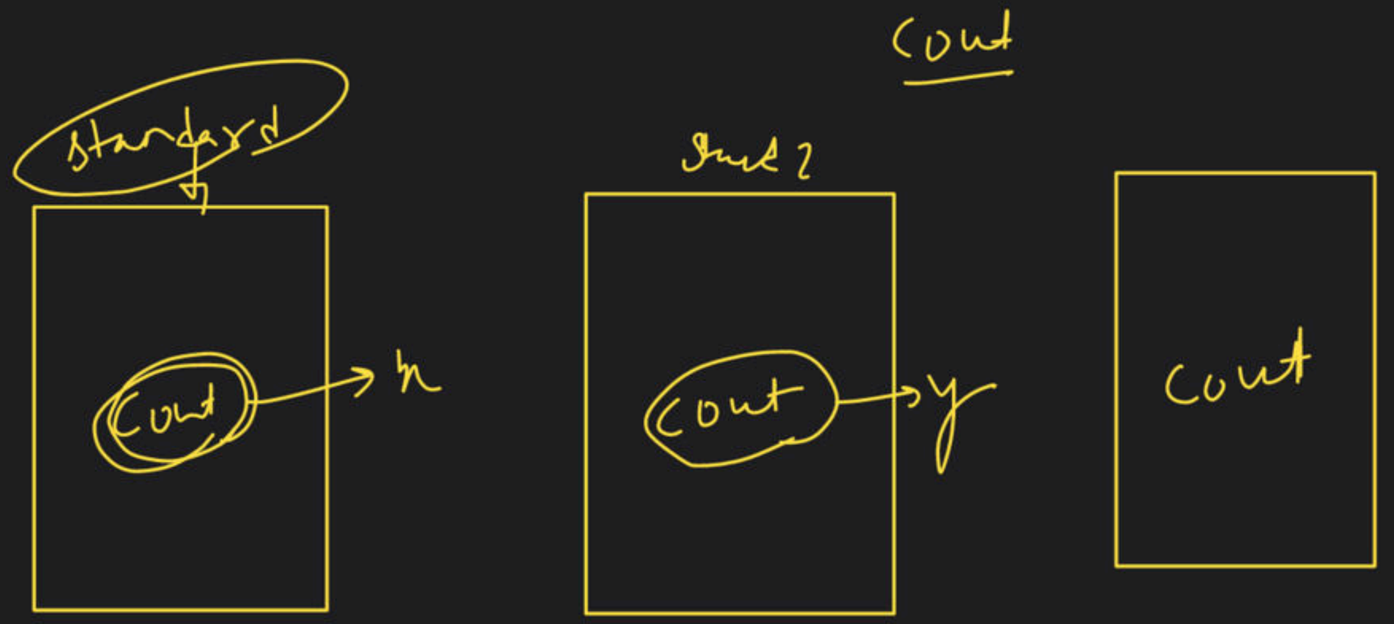
↳ Sublime

Let's Code your 1st Program

"Namaste Bharat"







SectionB.cpp

std → standard namespace

cout

Sec A::Rahul

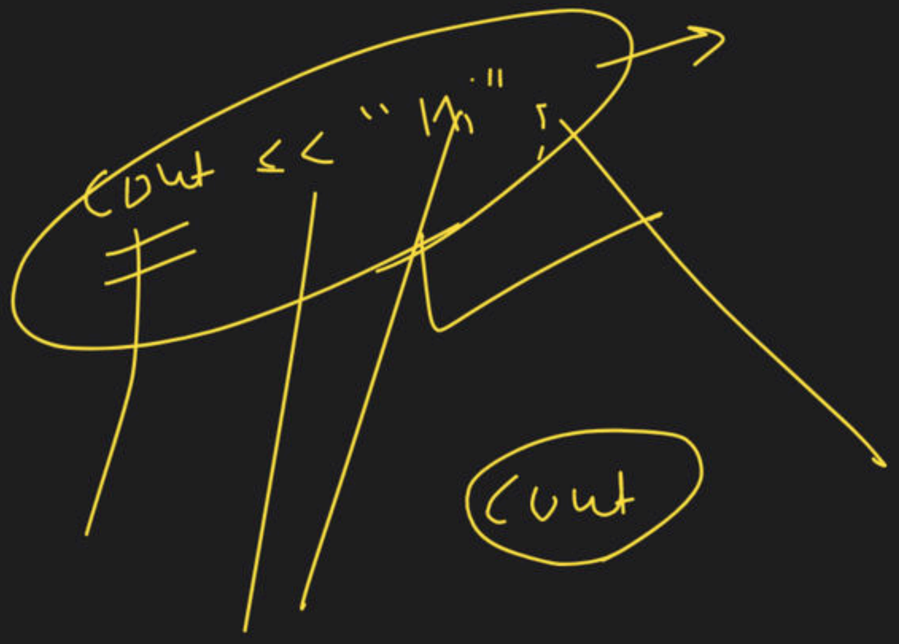
(Class SecA

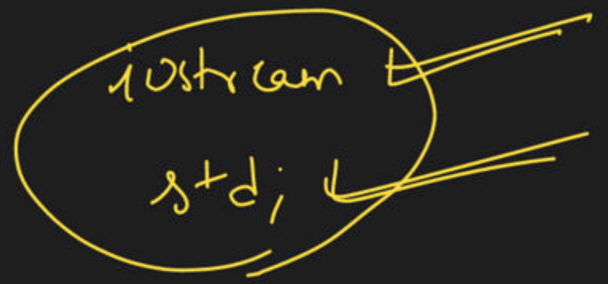
Rahul

SectionB

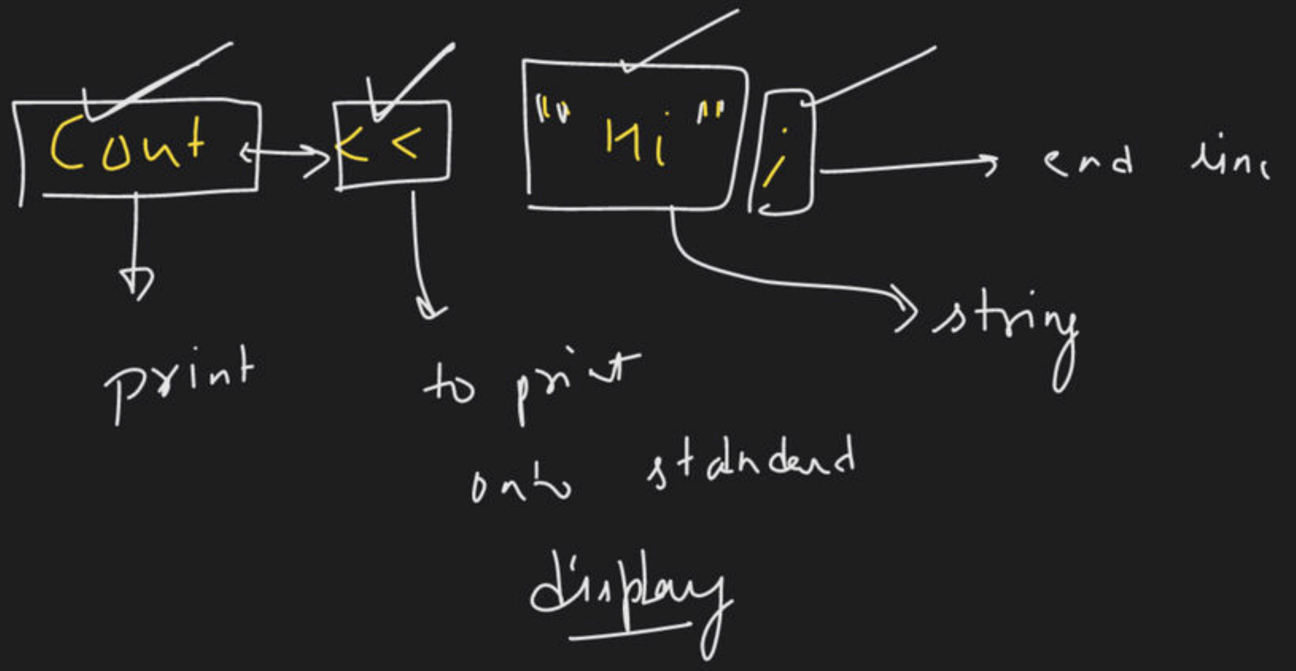
Rahul

what is namespace?


cout << "Hi" ;
cout


iostream
std;


"<<"



cout << 2;

cout << "2";

cout <<

Taking Input in C++

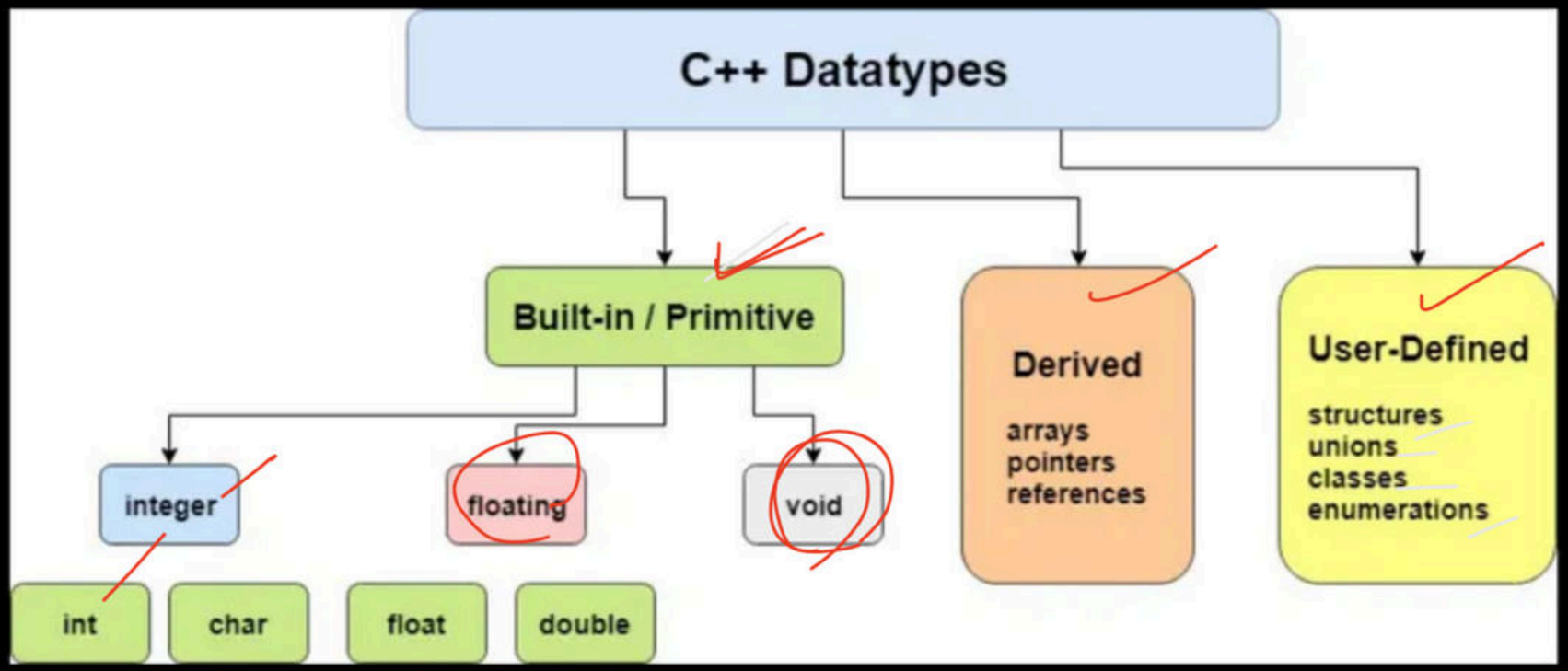
print -> cout

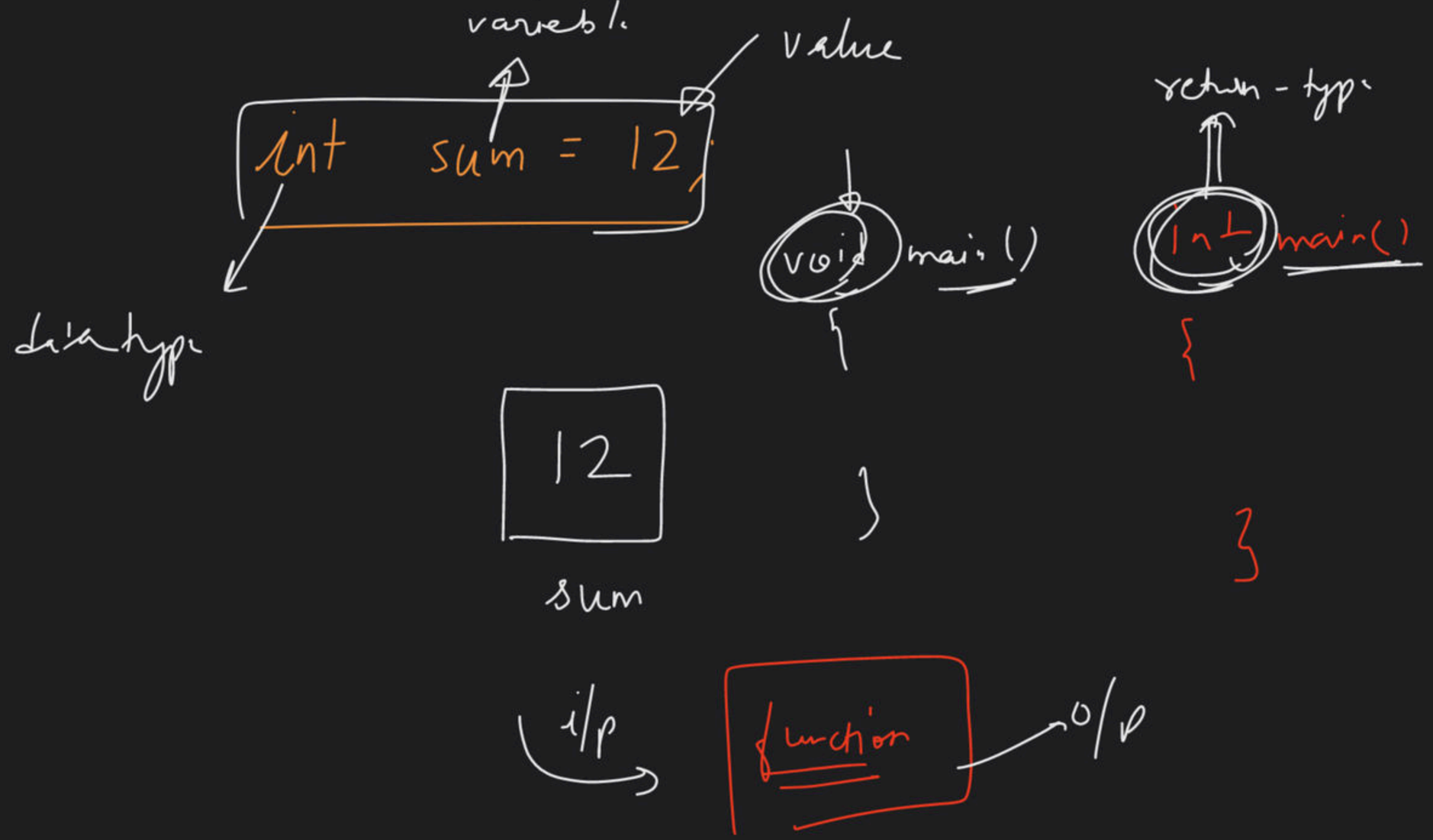
input -> cin

Skip

Datatypes & Variables

A variable in C is a memory location associated with some name in order to store some form of data and retrieve it when required & data types are used to tell the variables the type of data they can store.




variable
value
int sum = 12;
datatype
void main()
return-type
int main()
{
}
12
sum
}
i/p
function
o/p

DataTypes:

which type of data u will store

explore

32 bit CPU vs 64 bit CPU

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	-9,223,372,036,854,775,808-9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808-9,223,372,036,854,775,807	8	9,223,372,036,854,775,808-9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

```
void printName()
{
    cout << "u";
}
```

long long a = 1743;

int → 4 → 32

long long → 8 → $2^{63}-1$

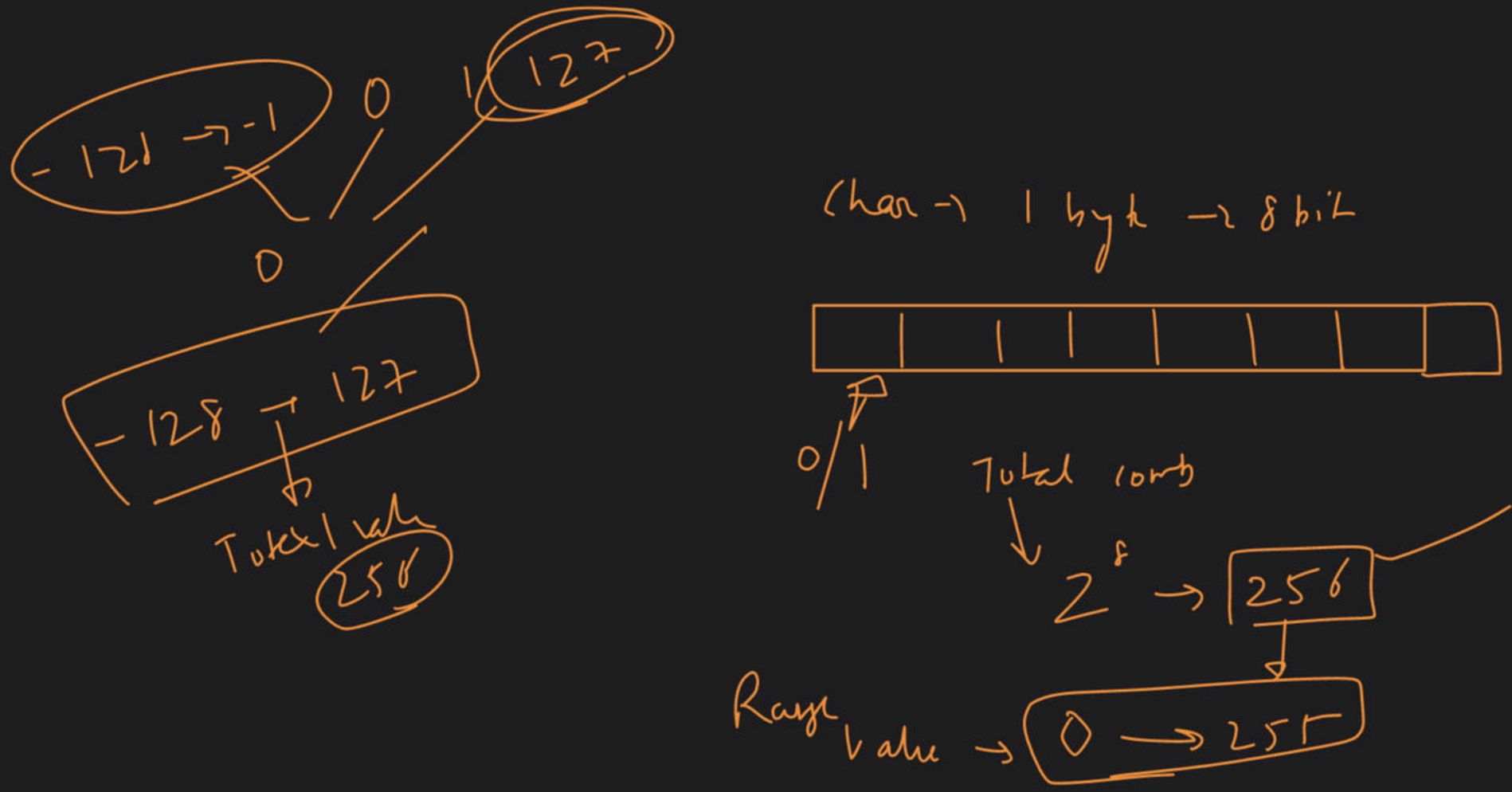
int -> 4 byte -> 32 bit

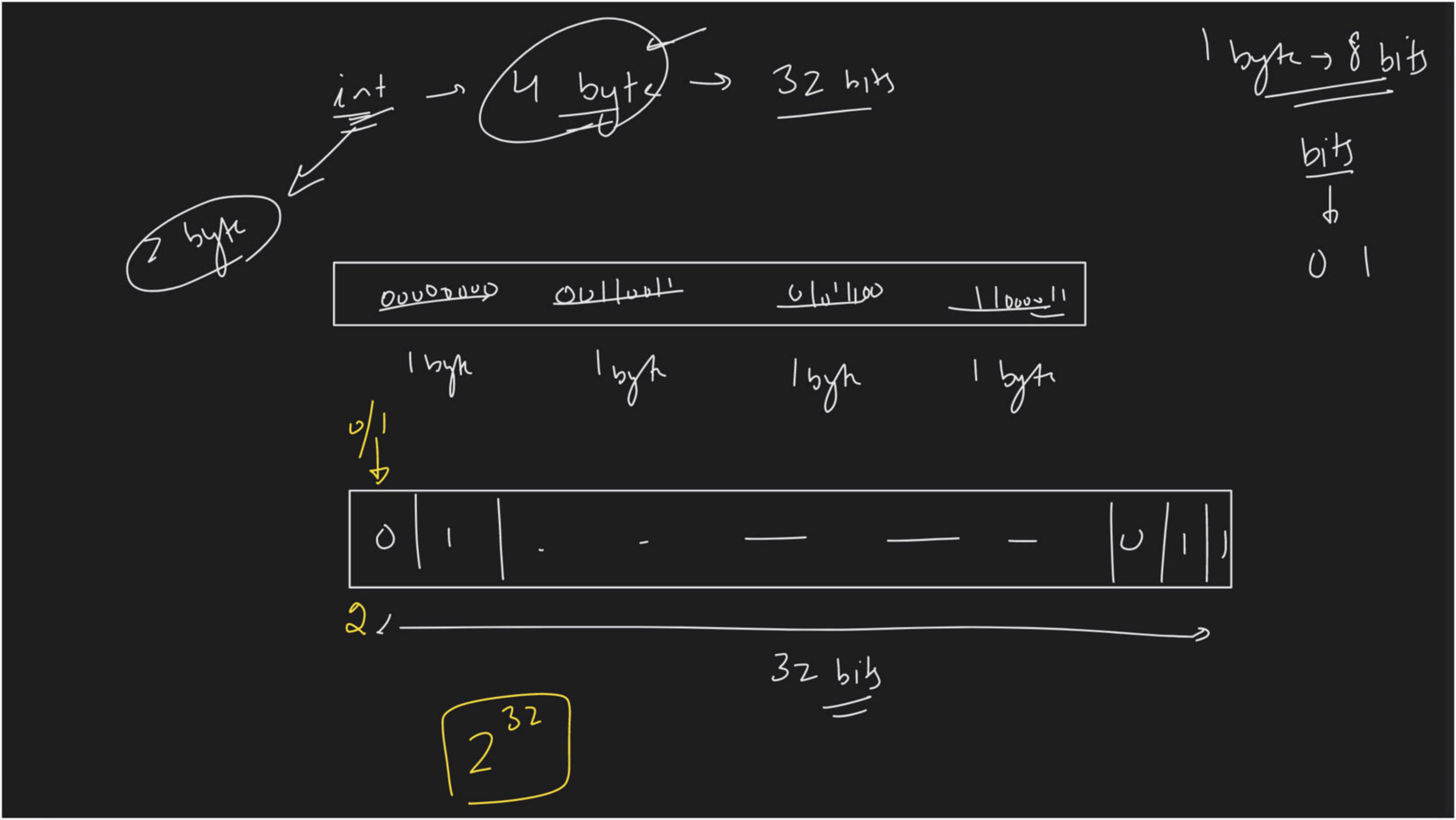
Total ca ->

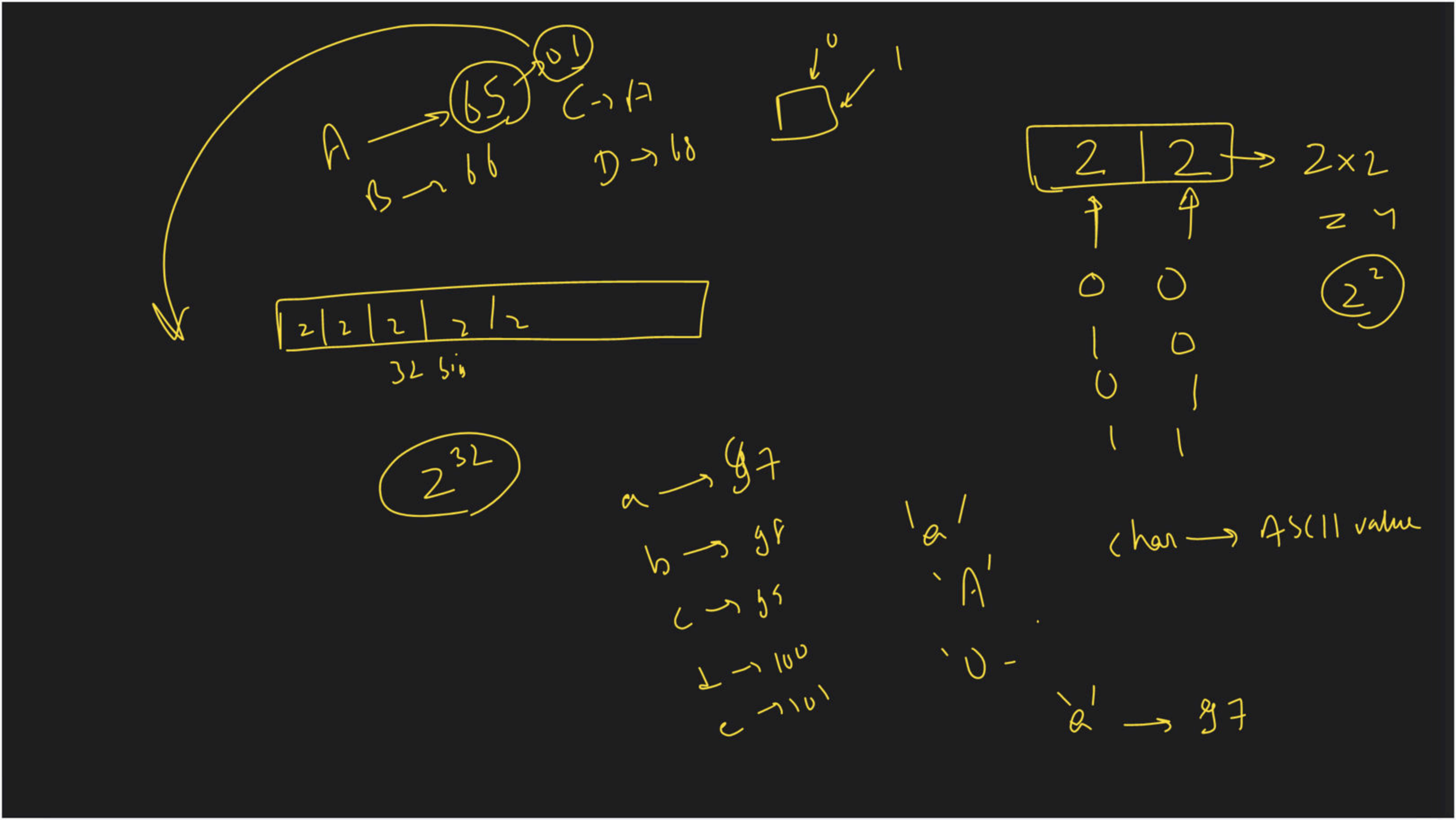
2^{32}

max val ->

$2^{32} - 1$


-128 → 127
Total value
256
0
1
-128 → -1
127
0
char → 1 byte → 8 bit
0/1
Total comb
2^8 → 256
Range value → 0 → 255


int → 4 byte → 32 bits
2 byte
1 byte → 8 bits
bits
0 1
1 byte
1 byte
1 byte
1 byte
0/1
0 1 . . — — — 0 1 1
2^
32 bits
2^32

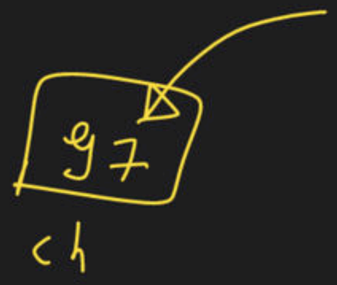

A
65
01
C → 67
B → 66
D → 68
0
1
2 | 2 | 2 | 2 | 2 | 2
32 bits
2^{32}
2 | 2 → 2×2 = 4
0 0
1 0
0 1
1 1
2^2
a → 97
b → 98
c → 99
d → 100
e → 101
'a'
'A'
'0'
char → ASCII value
'a' → 97

char ch = 256;

Explore

→ ASCII table

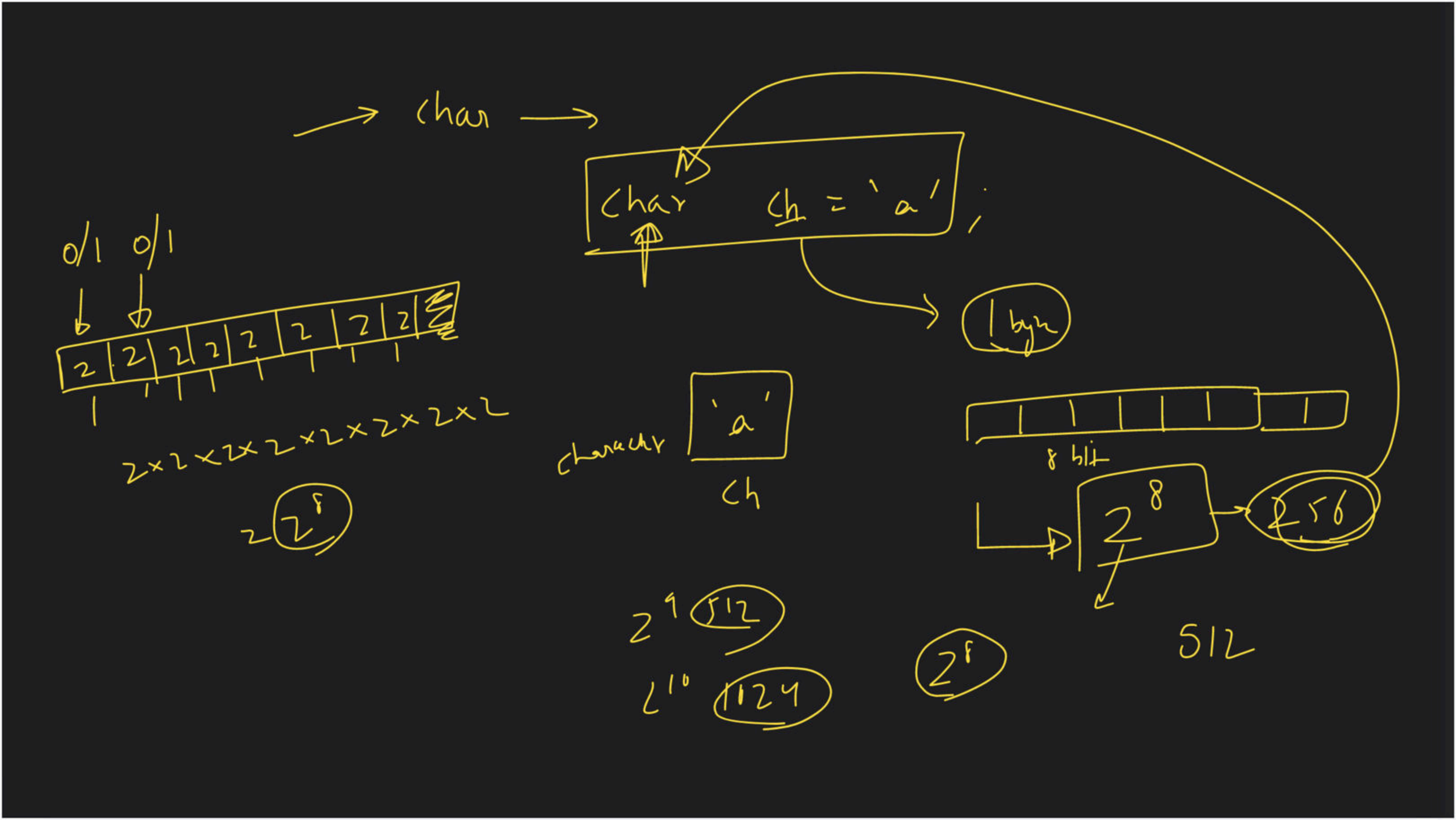
char ch = 'a';

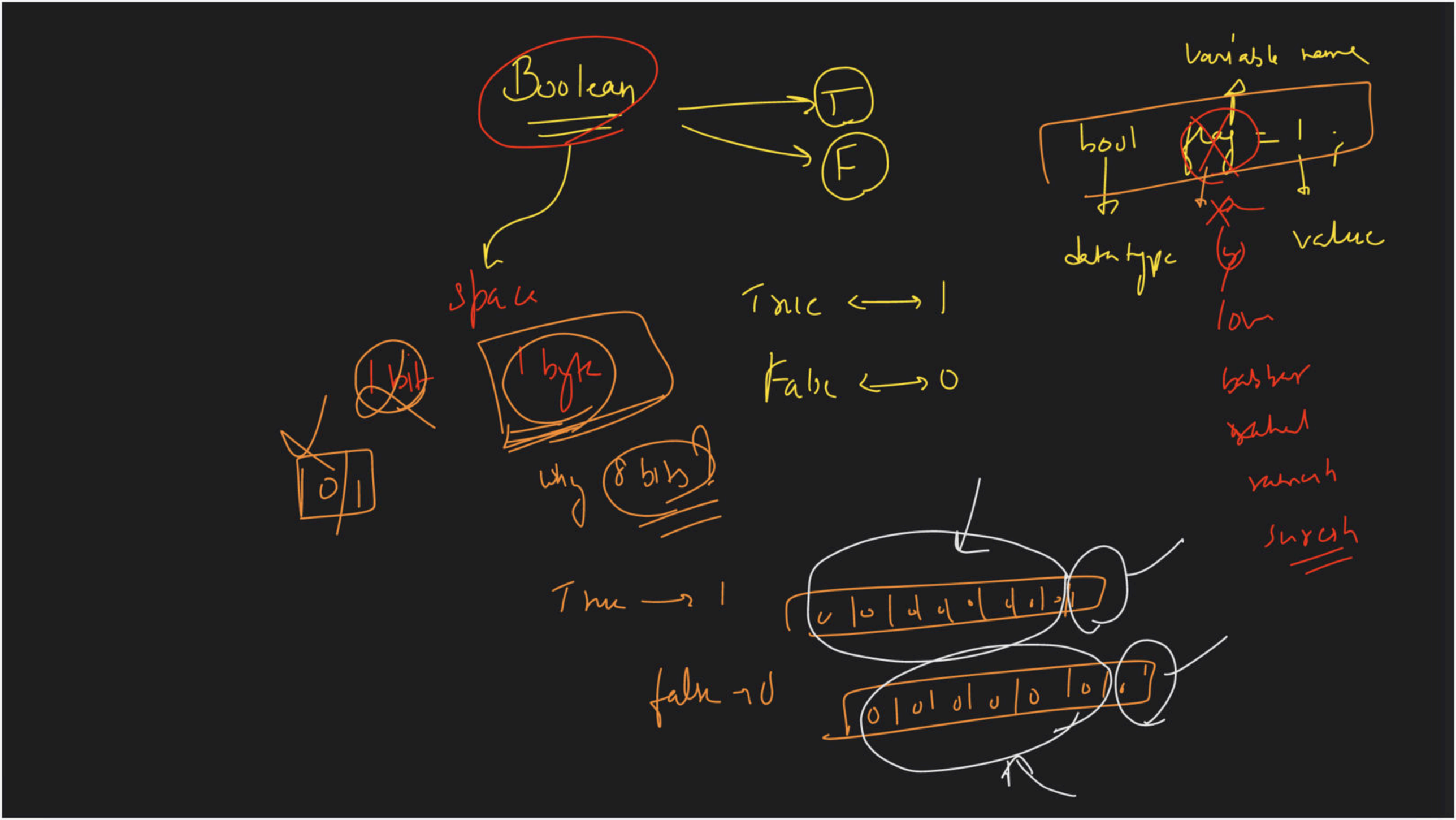


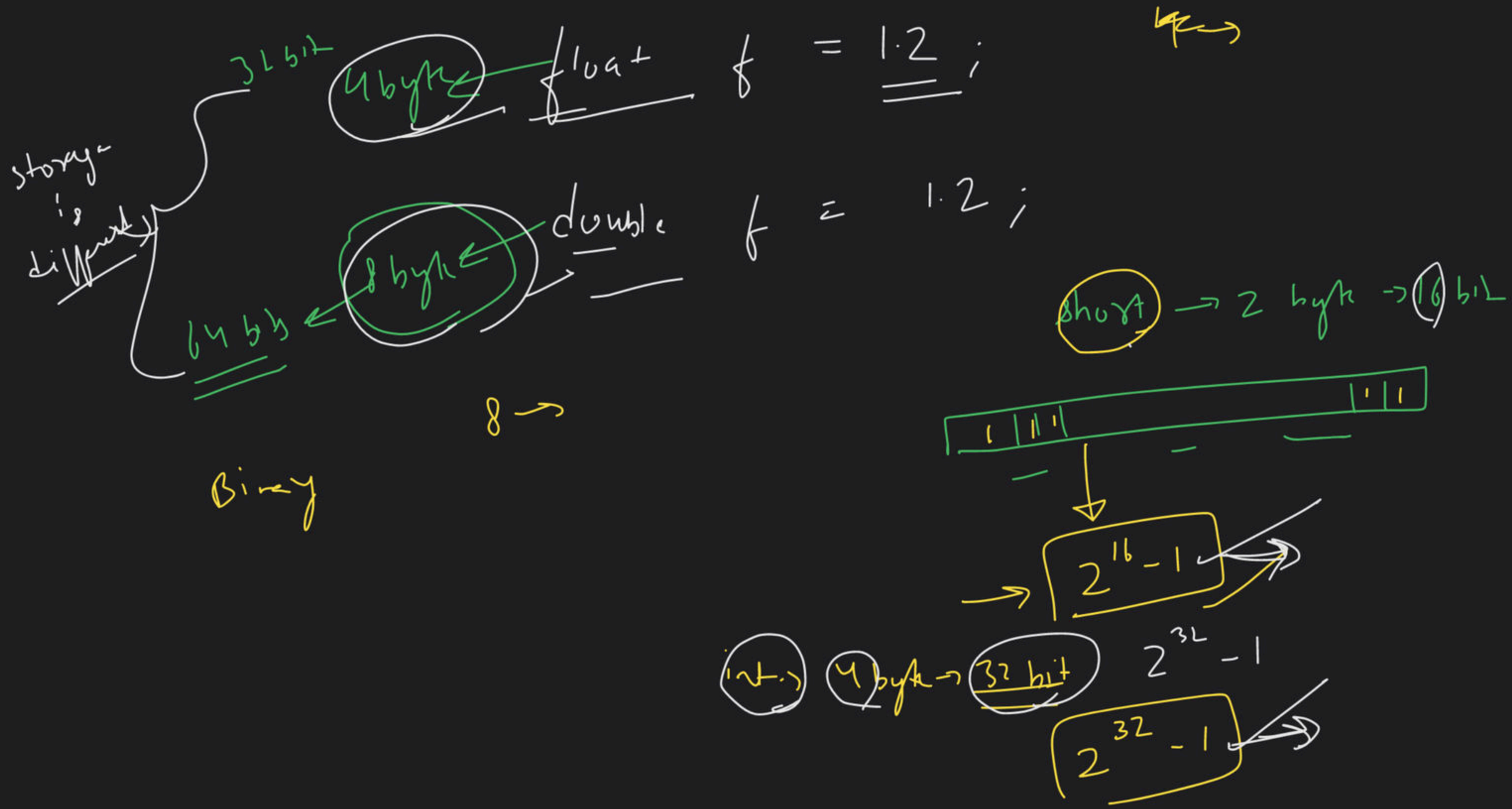
char

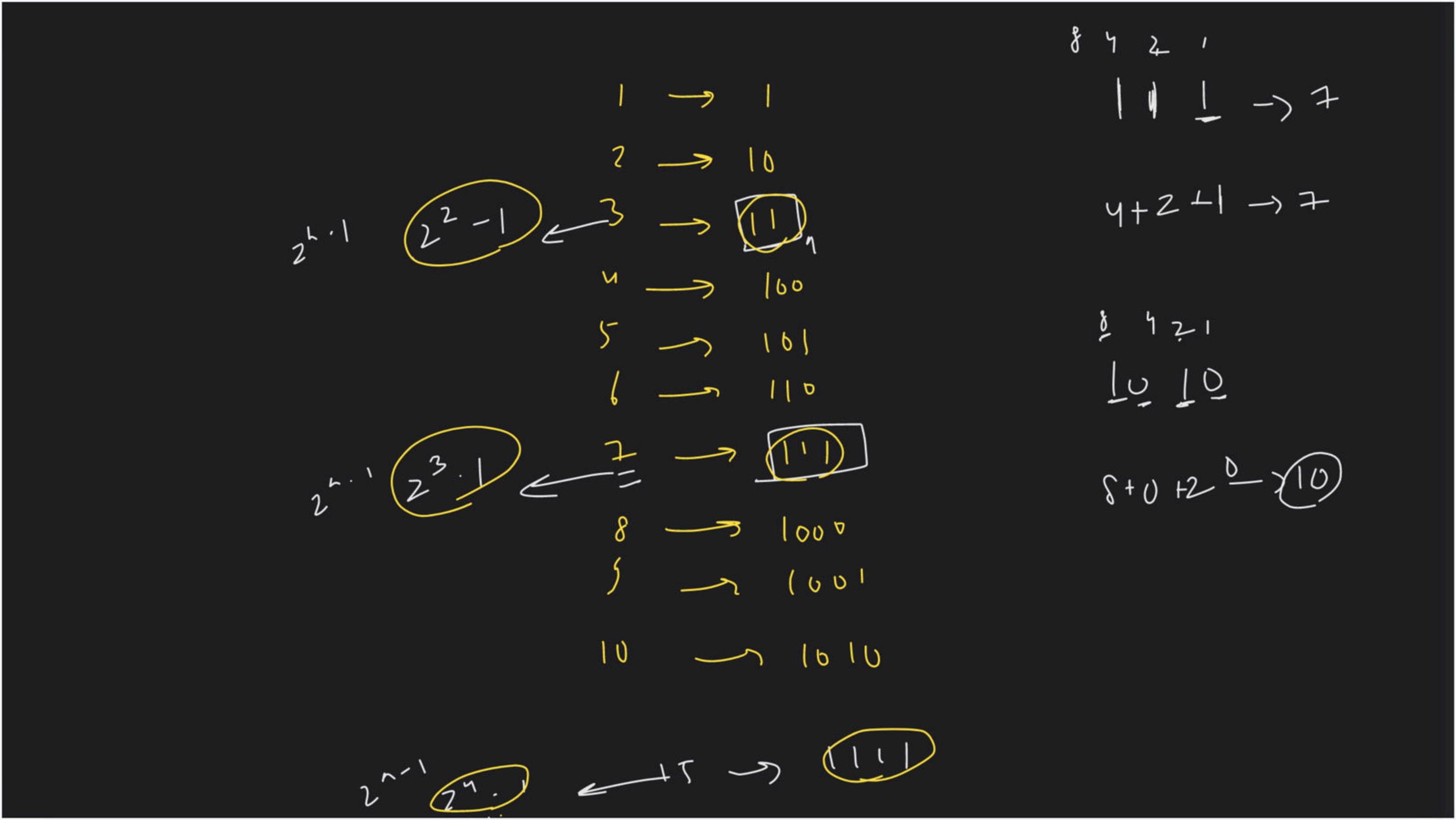
$2^8 - 1 = 255$

char → 256

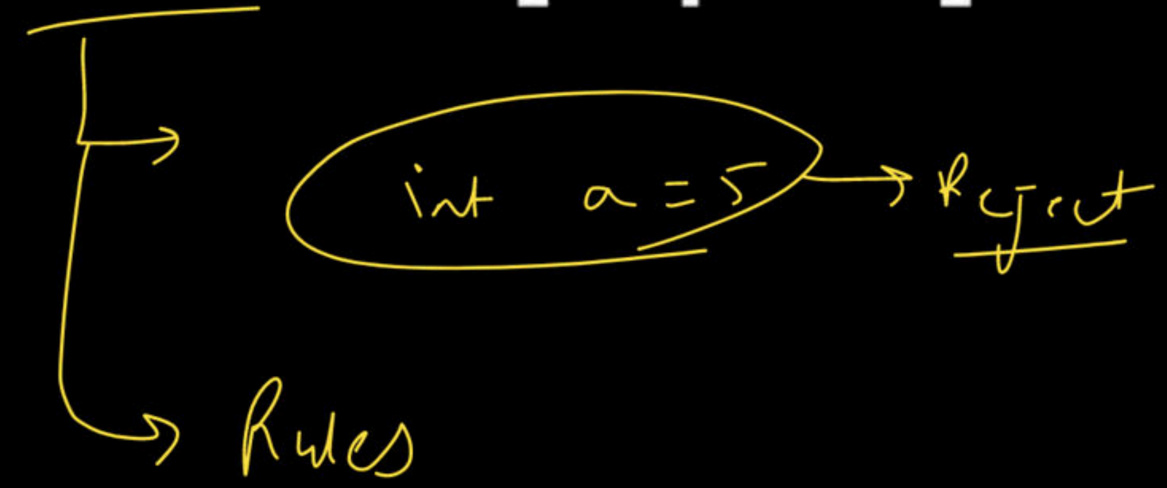

char
char ch = 'a';
0/1 0/1
2 2 2 2 2 2 2 2
2×2×2×2×2×2×2×2
2 2^8
character 'a'
ch
1 byte
8 bit
2^8
256
2^9 512
2^10 1024
2^8
512


Boolean
T
F
space
1 bit
1 byte
0/1
why 8 bits
True ⟷ 1
False ⟷ 0
True → 1
false → 0
variable name
bool
flag = 1 ;
data type
value
low
boolean
optimal
search
search


4 byte
float f = 1.2;
32 bit
storage is different
8 byte
double f = 1.2;
64 bit
8 →
Binary
short → 2 byte → 16 bit
1
$2^{16}-1$
int → 4 byte → 32 bit
$2^{32}-1$
$2^{32}-1$


1 → 1
2 → 10
3 → 11
4 → 100
5 → 101
6 → 110
7 → 111
8 → 1000
9 → 1001
10 → 1010
2^n-1
2^2 - 1
2^3 - 1
2^4 - 1
15 → 1111
8 4 2 1
1 1 1 → 7
4+2+1 → 7
8 4 2 1
1 0 1 0
8+0+2+0 → 10

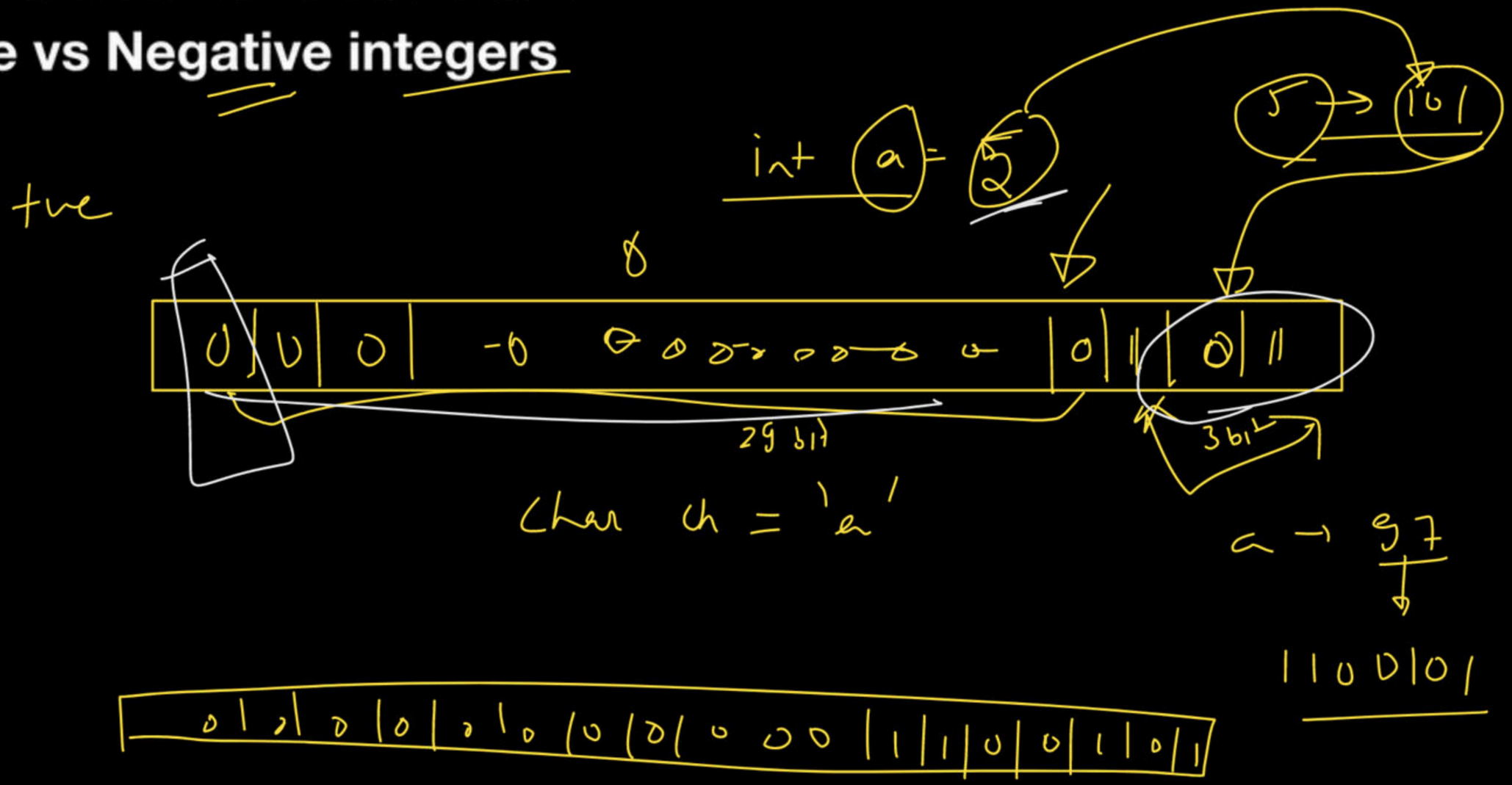
Variable Naming Conventions [Explore]

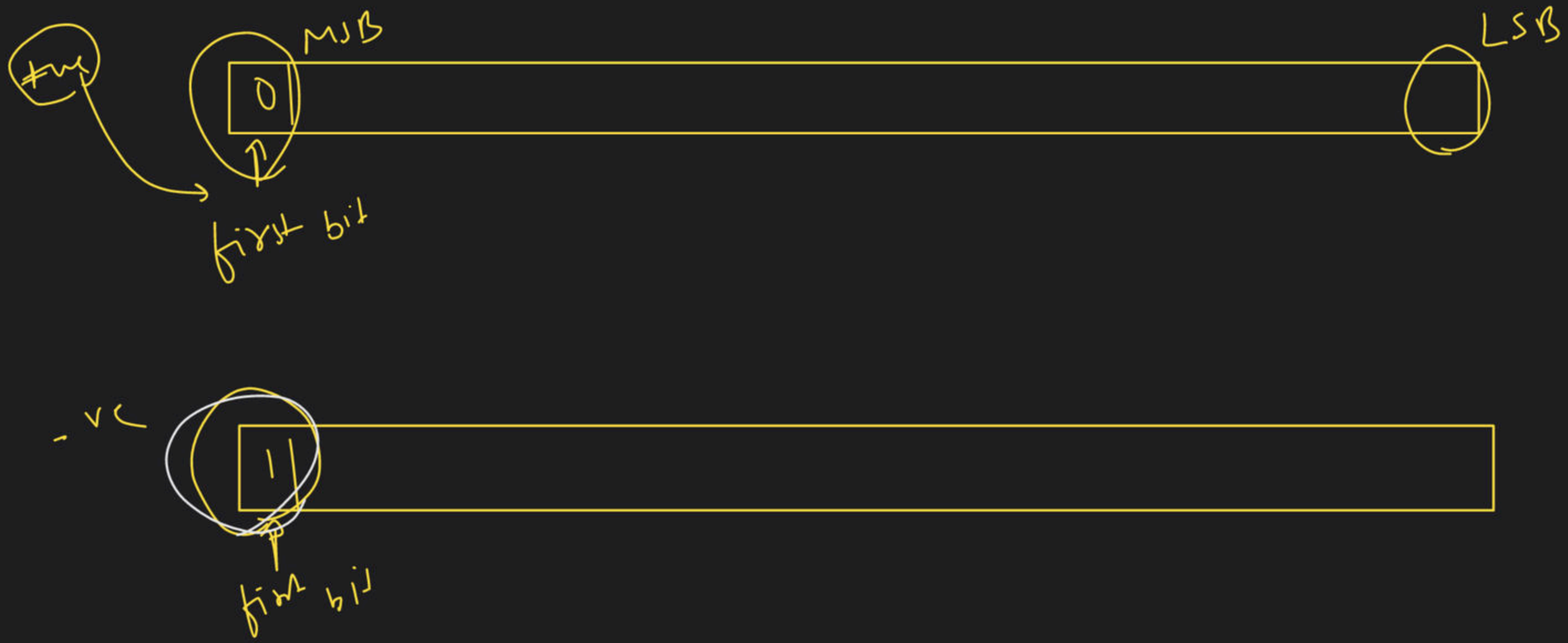


int studentcount = 5;

How data is stored ?

Positive vs Negative integers




+ve
MSB
LSB
0
first bit
-ve
1
first bit

How -ve no are stored in memory?

-ve → 2's complement → ?

1's complement + 1

1's complement

00101011

1's complement

+1

2's complement 11010101

1's complement -> flip bits

0 → 1

1 → 0

7 → 111

7 → 00000111

1's → 11111000

+1

2's 11111001

MSB or first bit

0 → true

1 → -ve

int a = -5

(1) ignore -ve sign

5

(2) Binary equivalent

0000 _ _ _ 00101

(3) 2's complement

decimal

1 → 10

2 → 10

Binary

1
1
10

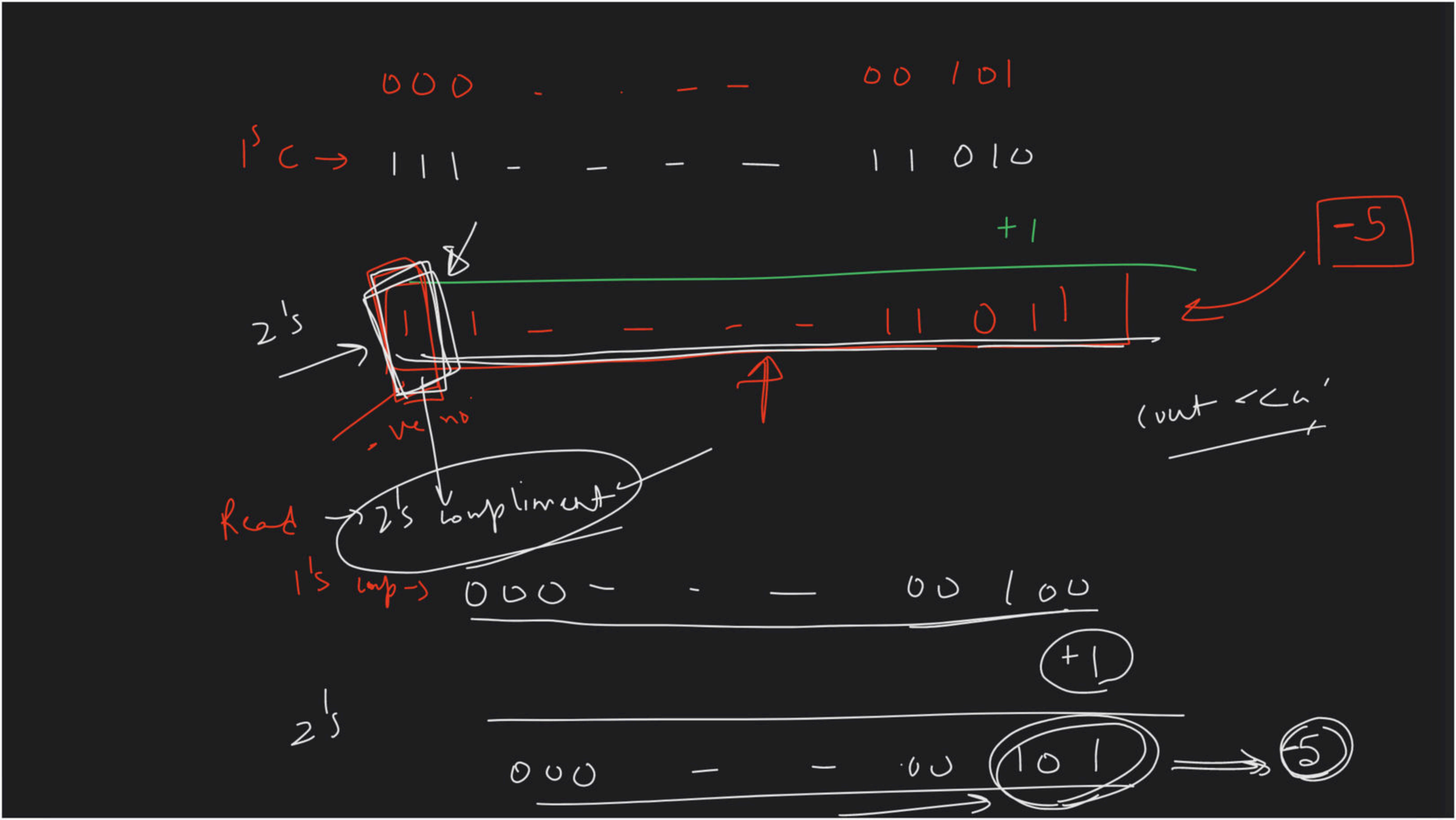
0 0 0 0 0 1 0 1

1 1 1 1

+1

10

0 0 0 0 1 0 1 1

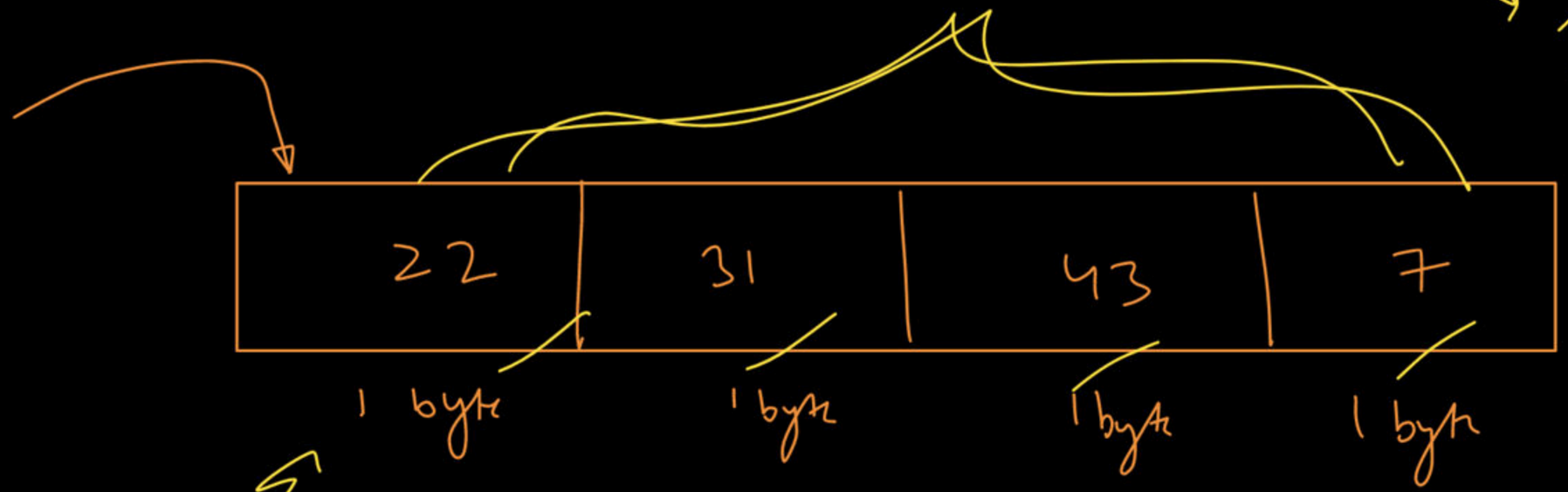

000 . . . 00 101
1's c → 111 - - - - 11 010
+1
-5
2's
1 1 - - - - 11 011
-ve no.
Read → 2's complement
(out << 'n'
1's comp → 000 - - - 00 100
+1
2's
000 - - 00 101
5

Interesting Problem:

Easy

datatype → type of data

datatype → space



datatype

4 byte

int a2;

int

char

1 int

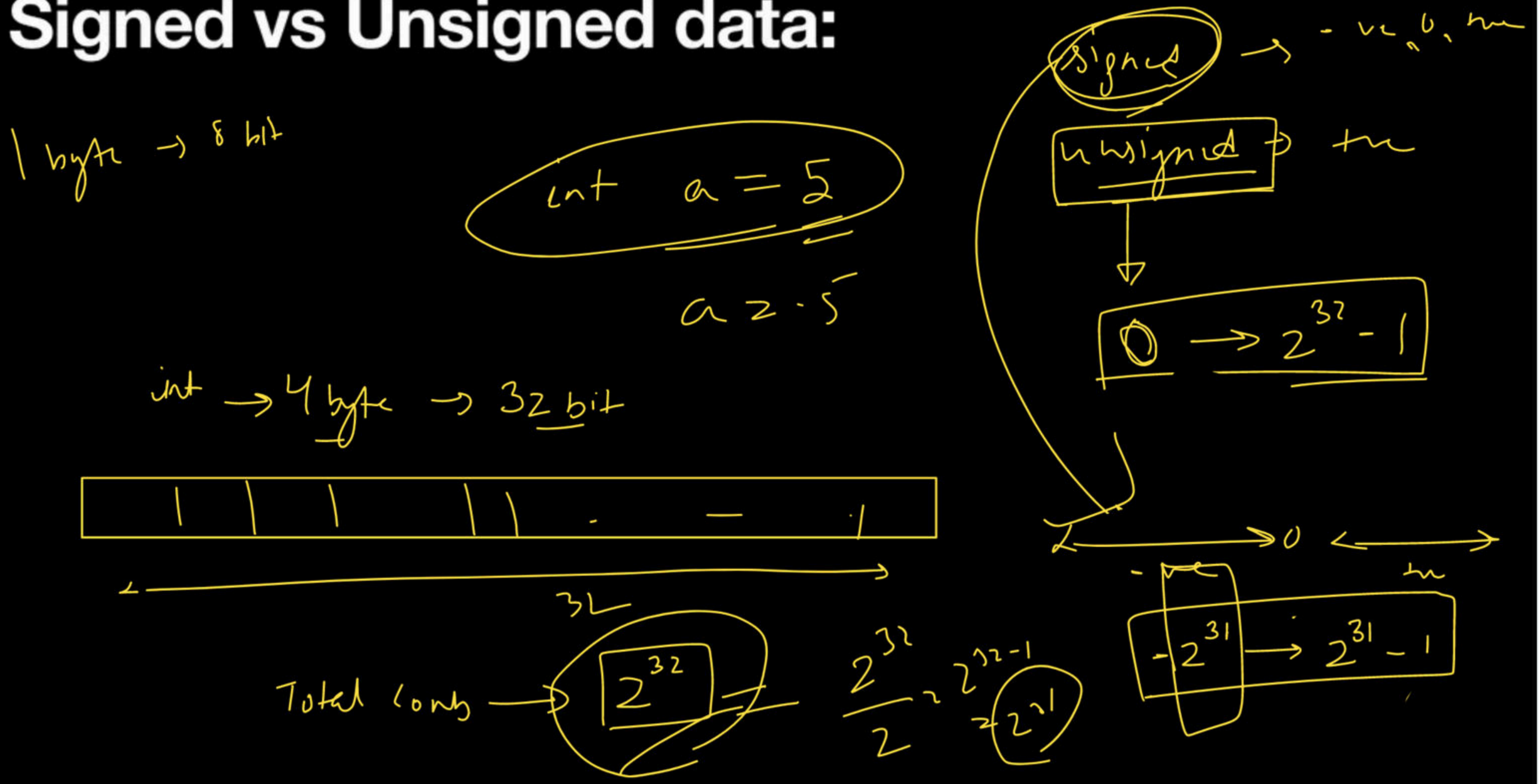
4 char

char → 1 byte

double

8 byte

Signed vs Unsigned data:



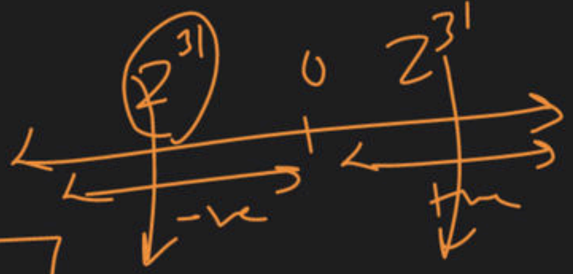
int → 4 byte → 32 bit

Total → 2^{32}

Total combination

$\frac{2^{32}}{2} = \frac{2^{32}}{2^1} = 2^{32-1} = 2^{31}$

$\frac{2^a}{2^b} = 2^{a-b}$



int → signed → -ve, 0, +ve

int → unsigned → 0, +ve → $0 \rightarrow 2^{32}-1$

$\frac{2^{32}}{2} \not< 2^{16}$

$2^{\frac{32}{2}} = 2^{16}$

$-2^{31} \rightarrow 0 \rightarrow 2^{31}-1$

$\frac{2^a}{2^b} = 2^{a-b}$

$\frac{2^{32}}{2^1} = 2^{32-1} = 2^{31}$

~~int~~ short → 2 byte → 16 bit

Total Comb → 2^{16}

$\frac{2^{16}}{2} = 2^{15}$

unsigned → ~~0 → 2^16 - 1~~ → $0 \to 2^{16} - 1$

signed → $-2^{15} \to 2^{15} - 1$

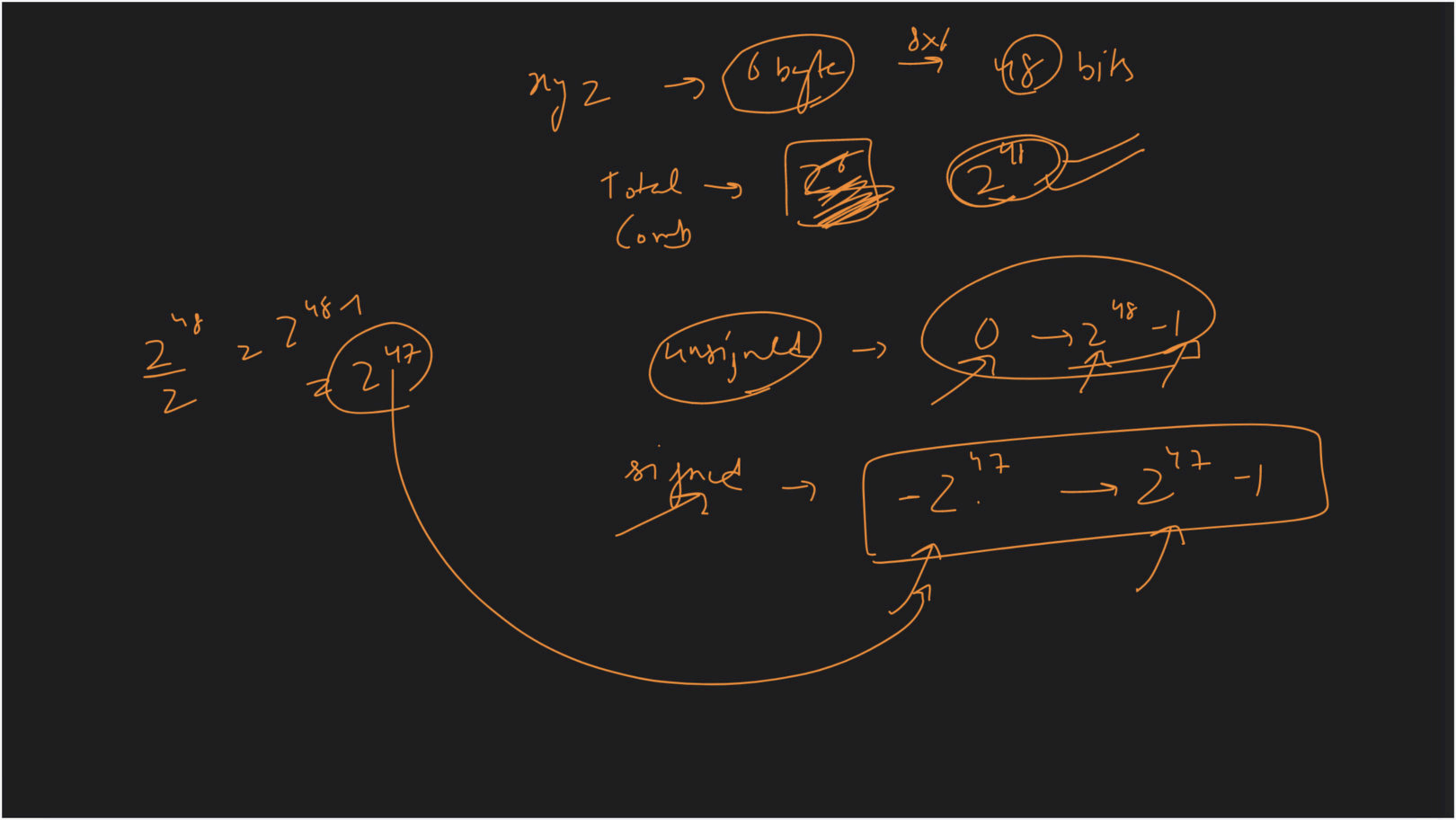
char → 1 byte → 8 bit

Total Comb → $2^8 = 256$

$\frac{2^8}{2} = 2^7$

unsigned → $0 \to 2^8 - 1$

signed → $-2^7 \to 2^7 - 1$


xyz
6 byte
8x6
48 bits
Total Comb
2^{48}
$\frac{2^{48}}{2} = 2^{48-1}$
2^{47}
unsigned
0 → $2^{48}-1$
signed
-2^{47} → $2^{47}-1$

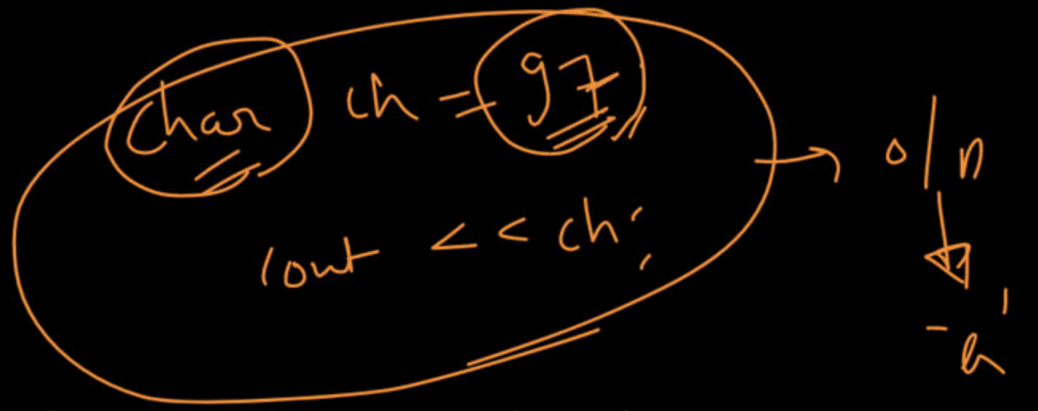
n bits

signed → $(-2^{n-1} \rightarrow 2^{n-1} - 1)$

unsigned → $0 \rightarrow 2^n - 1$

TypeCasting

Implicit vs Explicit



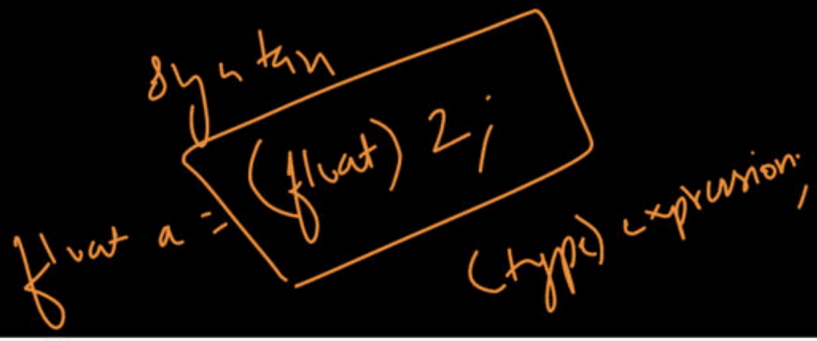
- Type casting refers to the conversion of one data type to another in a program. Typecasting can be done in two ways: automatically by the compiler and manually by the programmer or user. Type Casting is also known as Type Conversion.

Implicit Type Casting or Implicit Type Conversion

- It is known as the automatic type casting.
- It automatically converted from one data type to another without any external intervention such as programmer or user. It means the compiler automatically converts one data type to another.
- All data type is automatically upgraded to the largest type without losing any information.
- It can only apply in a program if both variables are compatible with each other.

automatically

by implicity type conversion

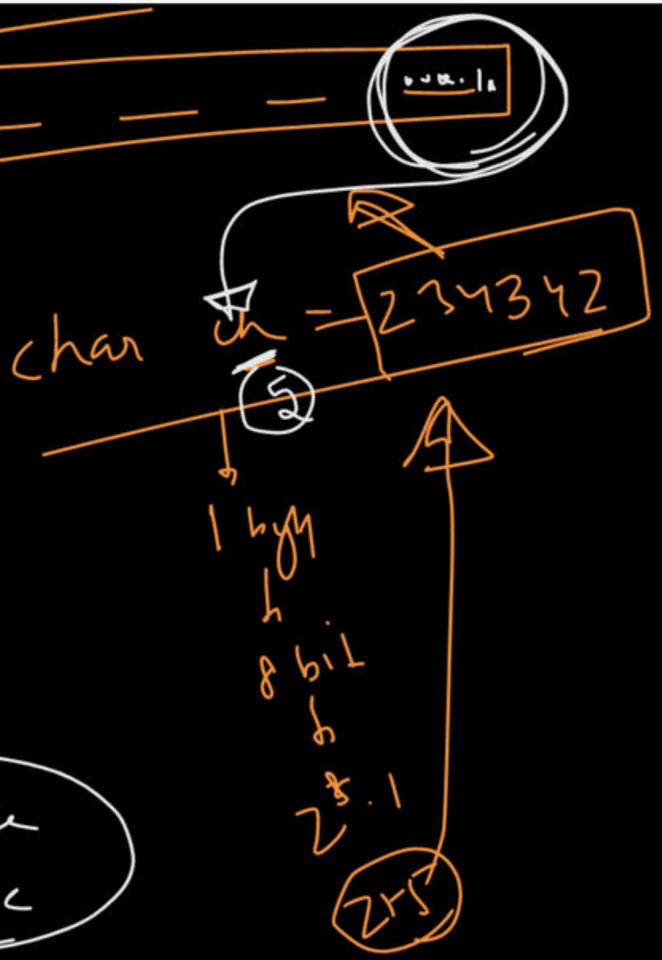


Explicit Type Casting or Explicit Type Conversion

- It is also known as the manual type casting in a program.
- It is manually cast by the programmer or user to change from one data type to another type in a program. It means a user can easily cast one data to another according to the requirement in a program.
- It does not require checking the compatibility of the variables.
- In this casting, we can upgrade or downgrade the data type of one variable to another in a program.
- It uses the cast () operator to change the type of a variable.

Operators:

- Arithmetic → % , + , - , * ,
- Relational → >, <, >=, <=, !=, ==
- Assignment → int a = 5;
- Logical
- Bitwise → 2 classes Based

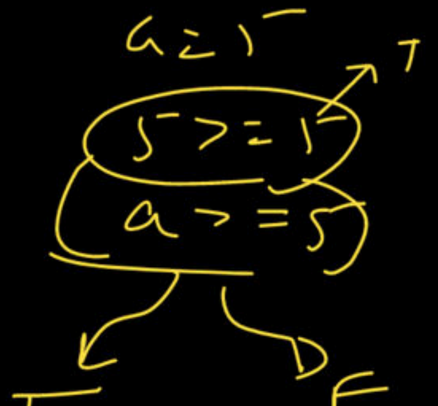


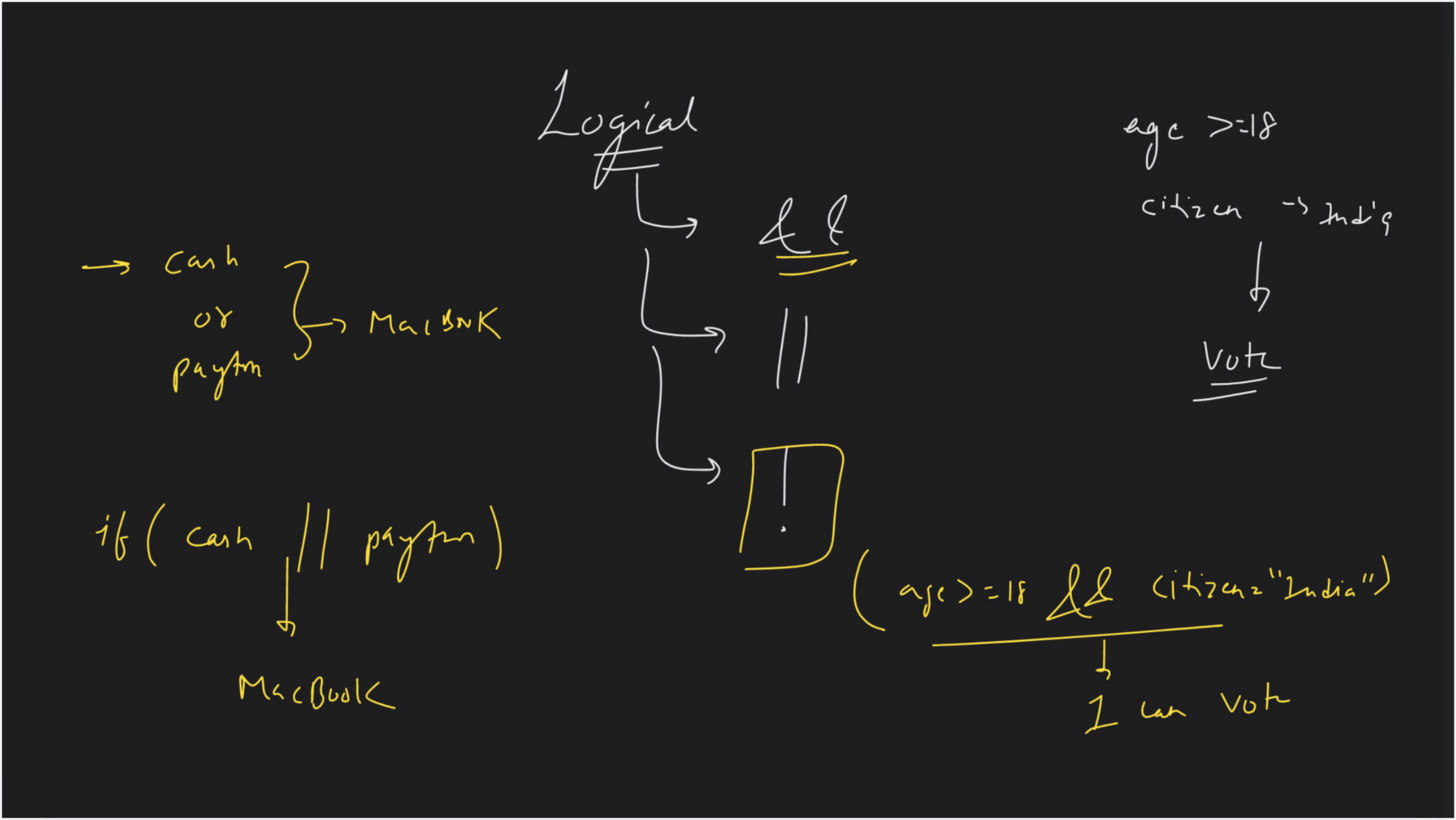
Precedence table

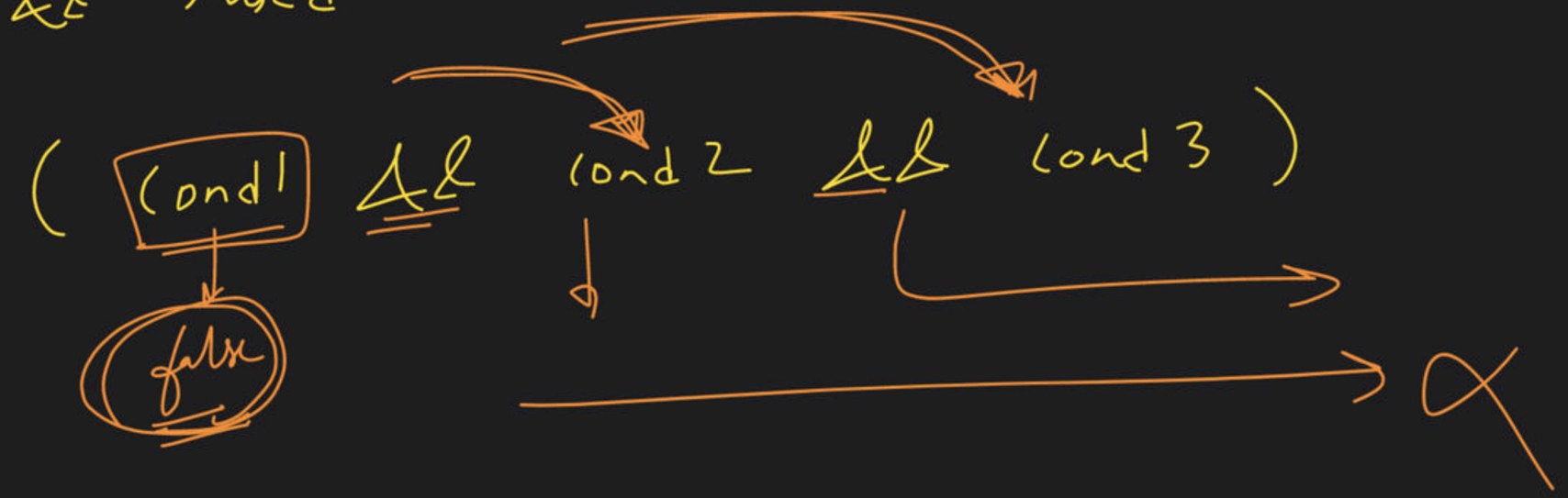
> , < , >= , <= , == , !=

!(false) → true

!(a >= 5)
!(true) → false




Logical
&&
||
!
→ Cash or paytm → MacBook
if (cash || paytm)
MacBook
age >=18
citizen -> India
Vote
(age >=18 && citizen "India")
I can Vote



int 4

int 2

→ ~~left~~ right → DSA

→ chownen - money → record

→ raat → subah → news

→ DSA

B

(! Student)

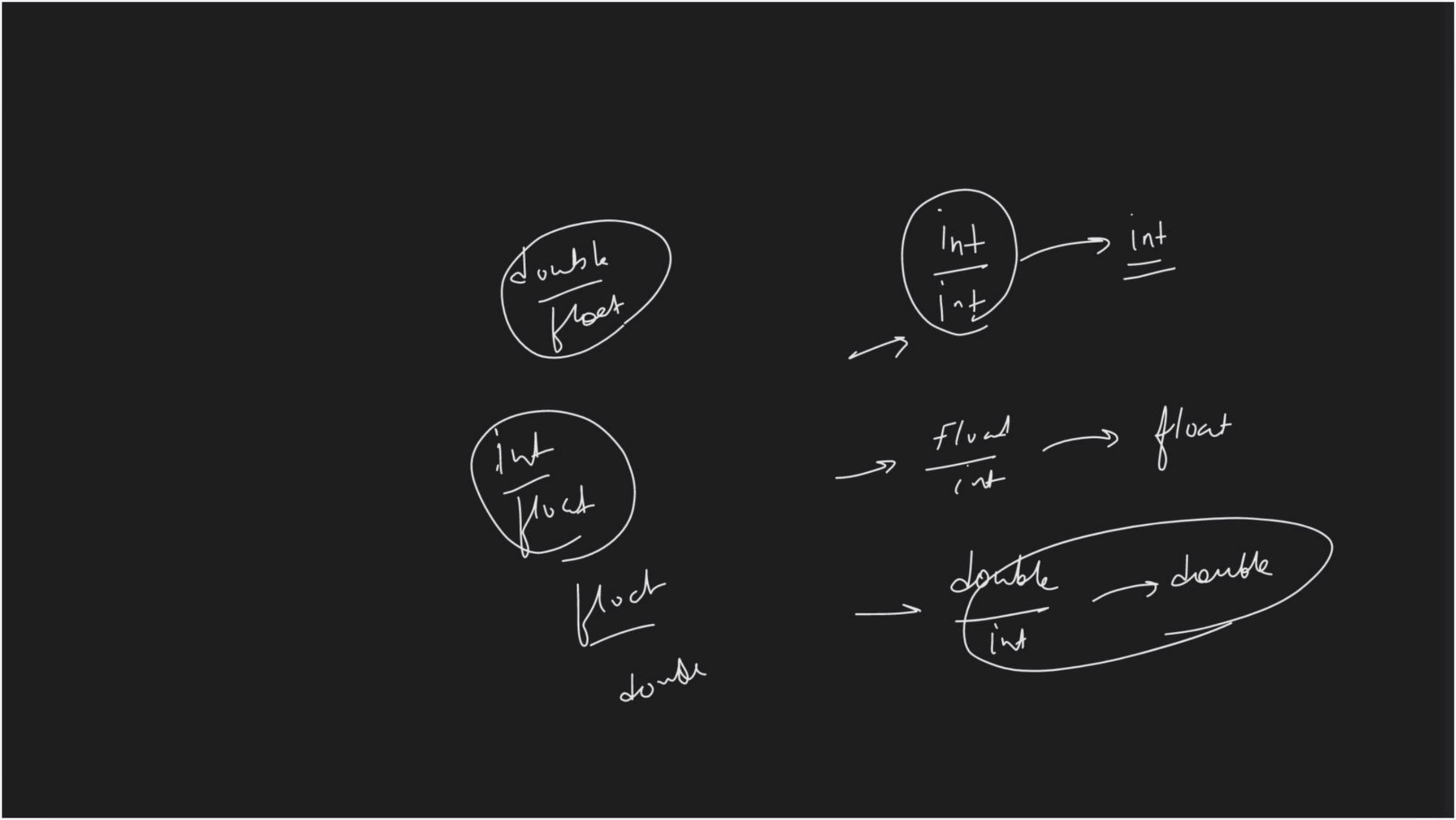
↓

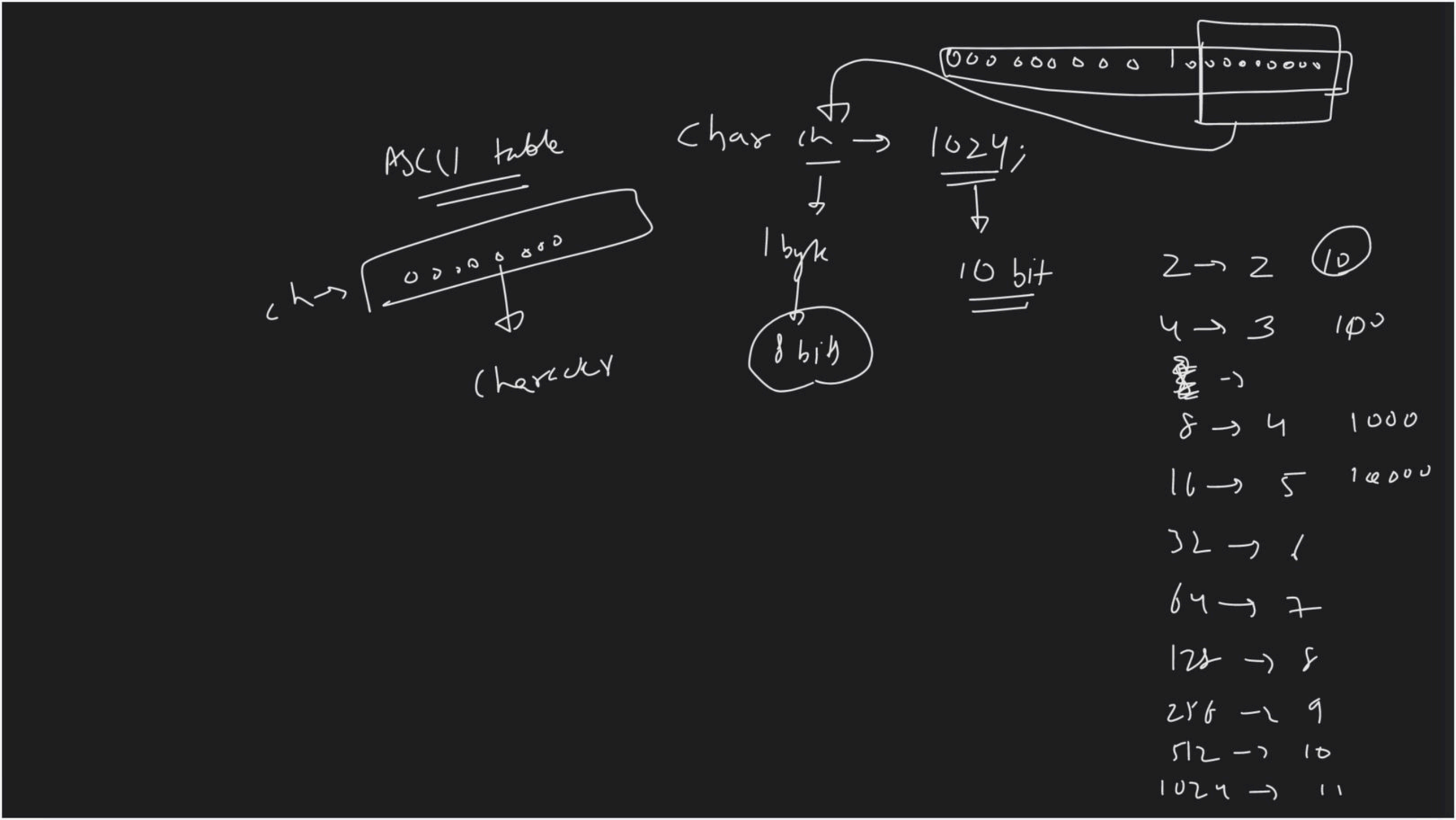
working Professional

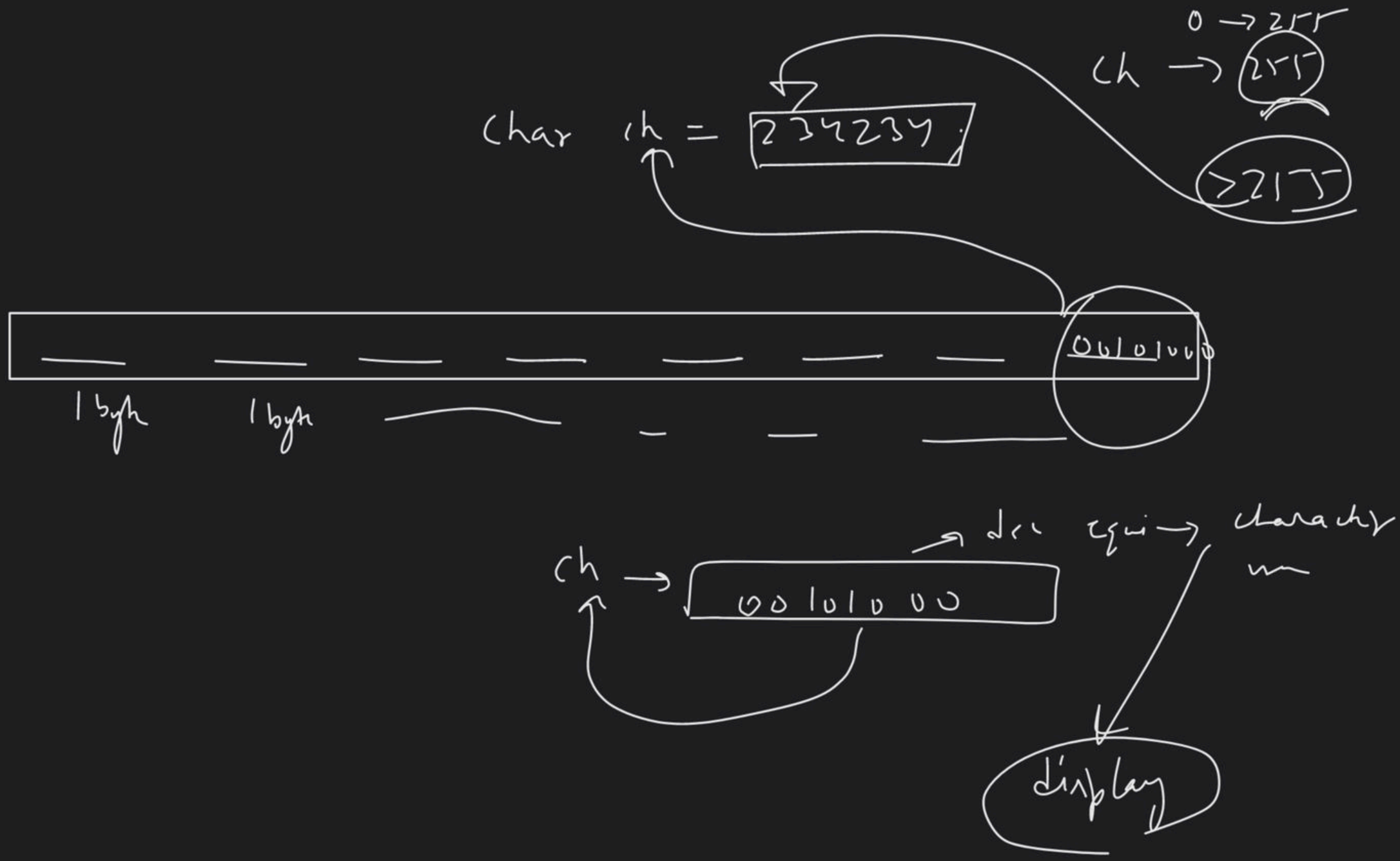
! WP

↓

Student

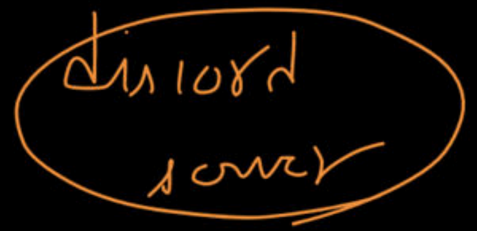

double
float
int
int
int
int
float
float
double
int
double

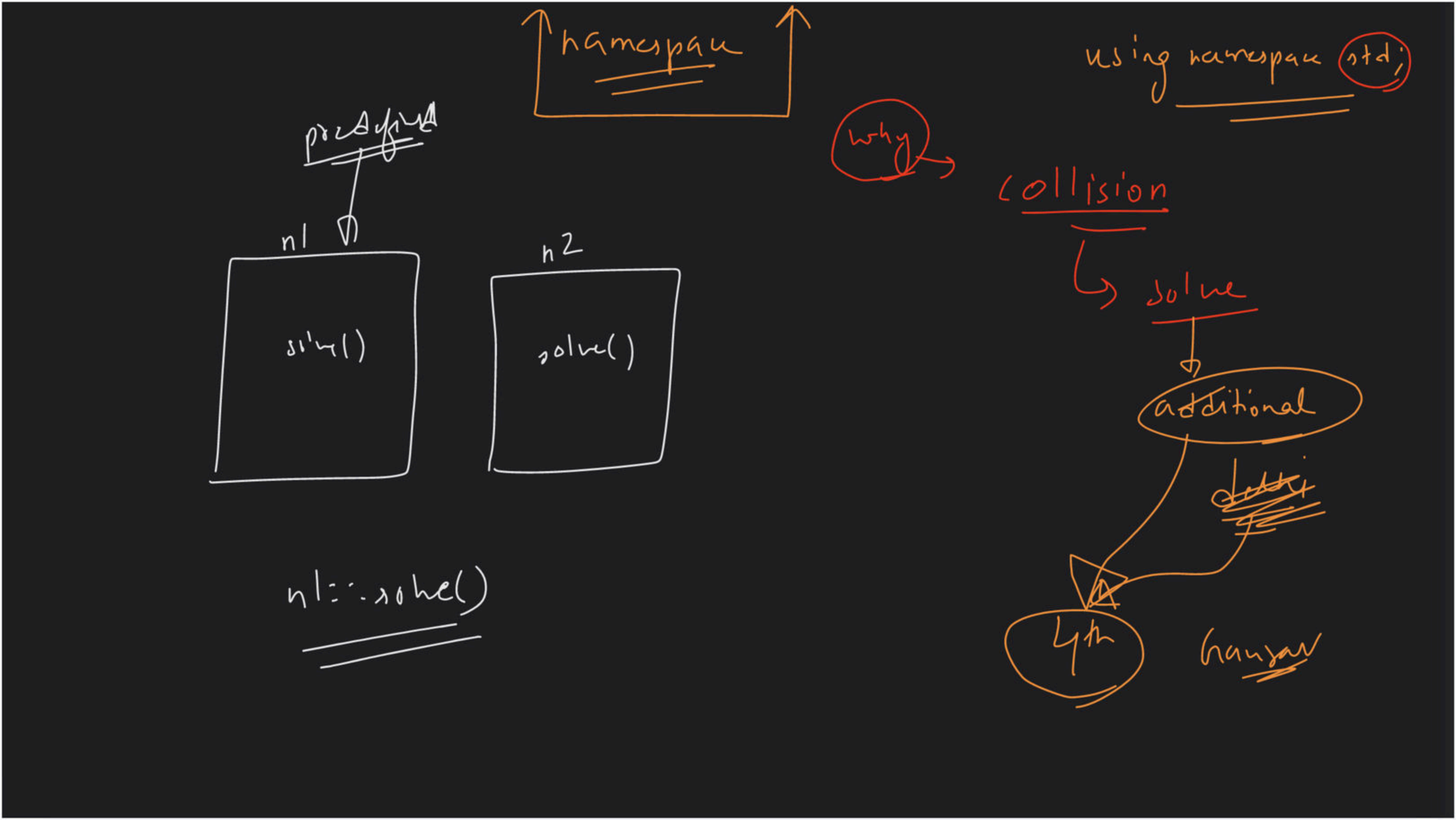

ASCII table
ch →
Character
Char ch → 1024;
1 byte
8 bit
10 bit
2 → 2 10
4 → 3 100
8 → 4 1000
16 → 5 10000
32 → 6
64 → 7
128 → 8
256 → 9
512 → 10
1024 → 11


0 → 255
ch → 255
>255
char ch = 234234;
1 byte
1 byte
ch →
display

Conditionals

Loops:




namespace
using namespace std;
why
collision
solve
additional
n1
n2
solve()
solve()
n1::solve()

custom namespace create

ABCD