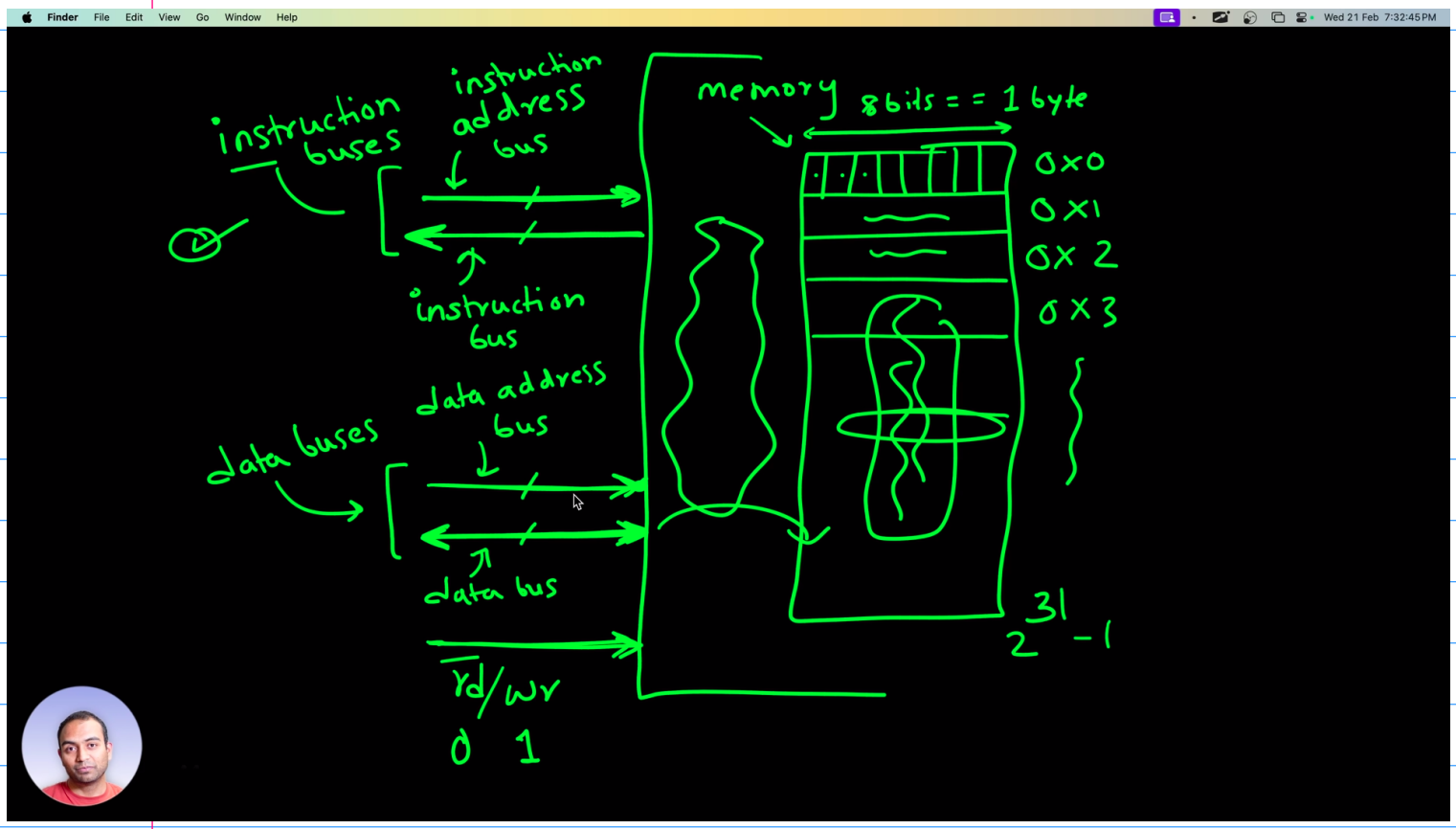
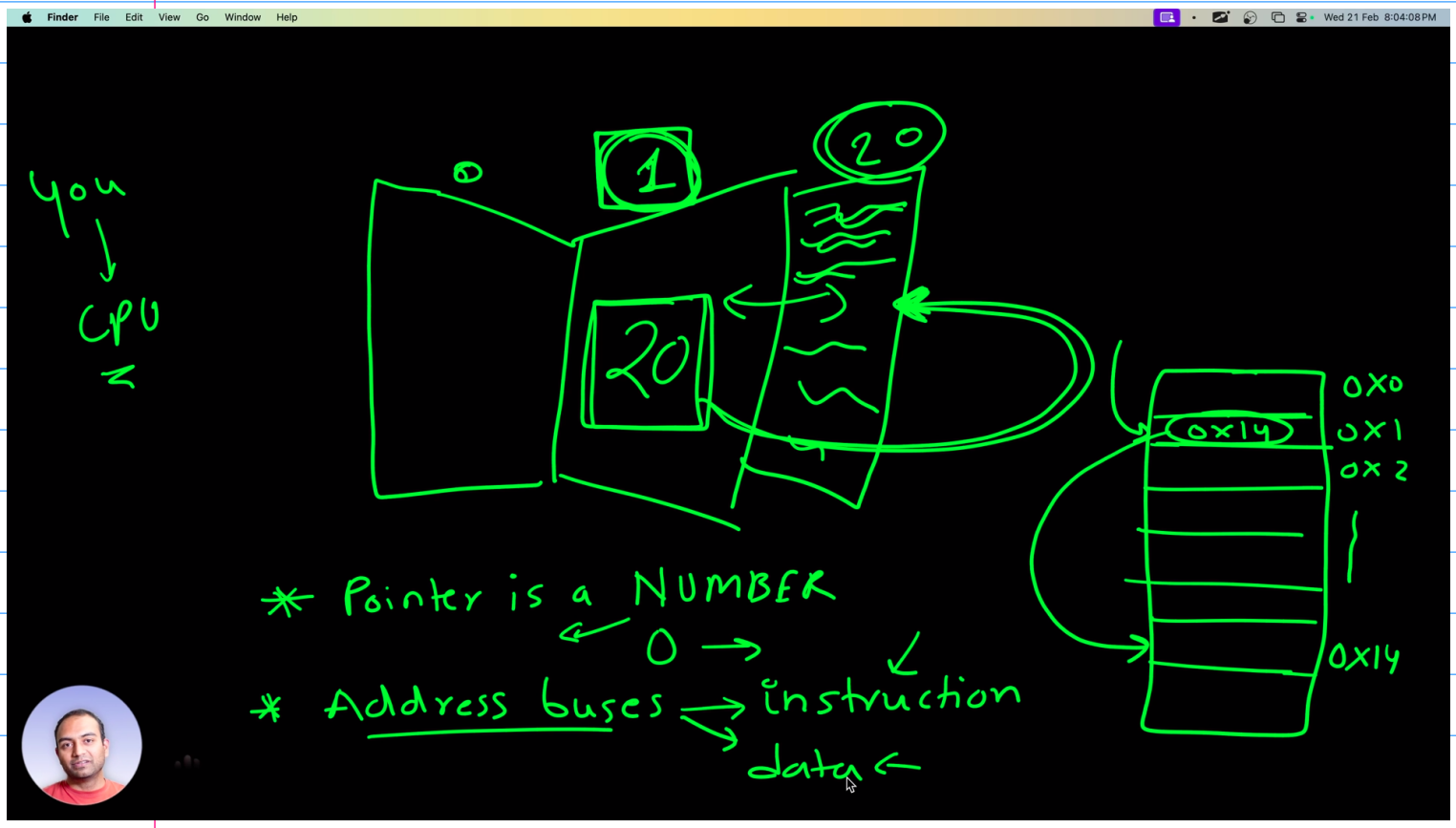
C Pointers: Secrets every Embedded Engineer MUST know!

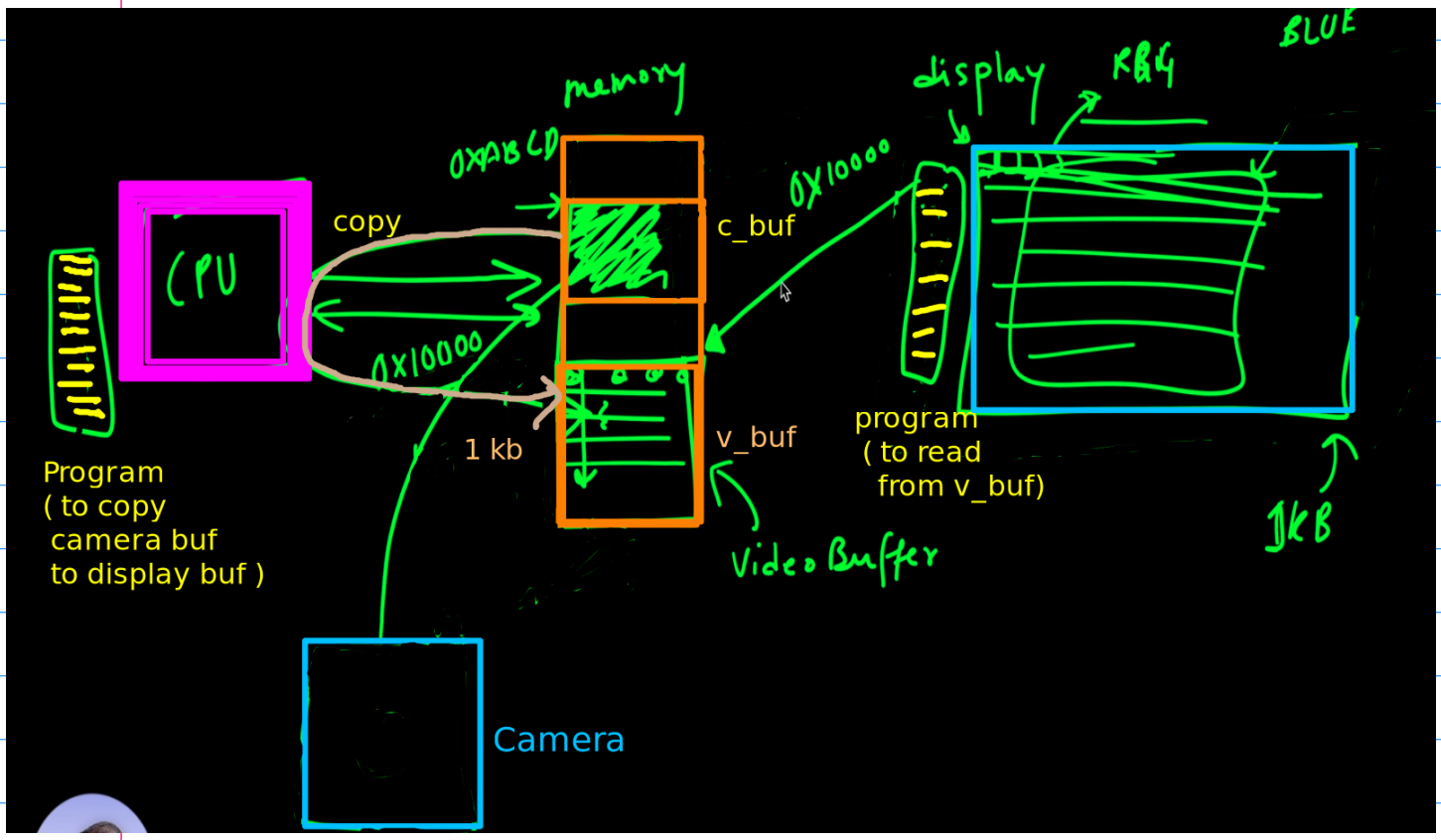
- Computer System with Data Bus and Address Bus



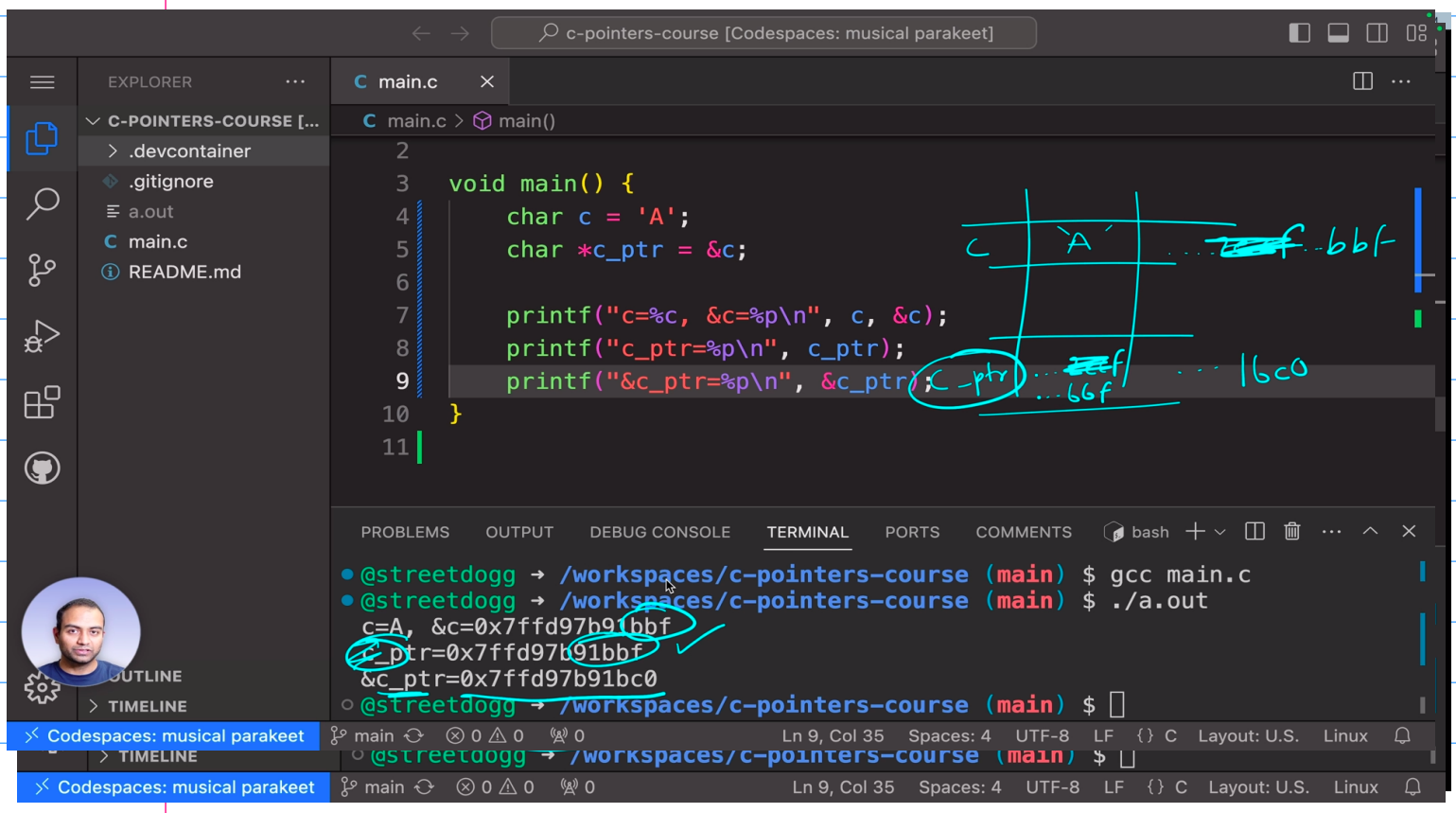
- What is pointer in computer system?



- How pointers and addresses are related to computer system?



- Basic of Pointers in Embedded C



- Some tricky concept in pointers

main.c > main()

```c
#include <stdio.h>
void main() {
    int i=0xFF;
    char c='A';
    char *cptr;
    int x;
    cptr=&c;
    x= i & cptr; // error
    x= i & & c;  // error
    x= i && c;  //  ok - logical AND
    x= i & (int)&c; // ok
    x = i * cptr; // error
    x = i ** cptr; //ok
    printf("x= i & cptr; --> %x\n",x);
    printf("cptr=%p\n",cptr);
    printf("cptr=0x%llx\n",(long long unsigned int)cptr);
}
```

```
@PranabNandy → /workspaces/c-pointers-course (main) $ gcc main.c
main.c: In function ‘main’:
main.c:8:10: error: invalid operands to binary & (have ‘int’ and ‘char *’)
    8 |     x= i & cptr; // error
      |          ^
main.c:9:10: error: invalid operands to binary & (have ‘int’ and ‘char *’)
    9 |     x= i & & c;  // error
      |          ^ ~~~
      |            |
      |            char *
main.c:11:12: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
   11 |     x= i & (int)&c; // ok
      |            ^
main.c:12:11: error: invalid operands to binary * (have ‘int’ and ‘char *’)
   12 |     x = i * cptr; // error
      |           ^
@PranabNandy → /workspaces/c-pointers-course (main) $
```

es: cuddly space robot main* ⊗ 3 ⚠ 0 0 Ln 13, Col 24 Spaces: 4 UTF-8 LF { } C Layout: us Linux

- Multi-level Pointers

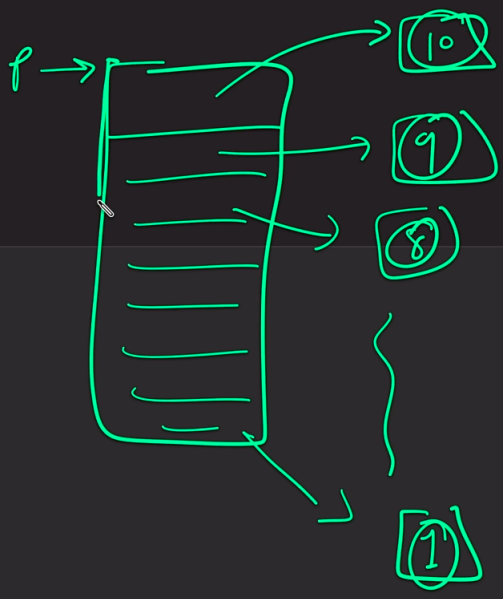
c-pointers-course [Codespaces: musical parakeet]

EXPLORER C main.c M

C-POINTERS-COURSE [...]
.devcontainer
.gitignore
a.out
main.c M
README.md

C main.c > main()

```c
#include <malloc.h>

void main() {
    char *p[10];

    printf("p: %p\n", p);
    for (int i=0; i<10; i++) {
        p[i] = (char *)malloc(1);
        *p[i] = 10 - i;
        printf("p[%d]: %p\n", i, p[i]);
    }

    for (int i=0; i<10; i++) {
```



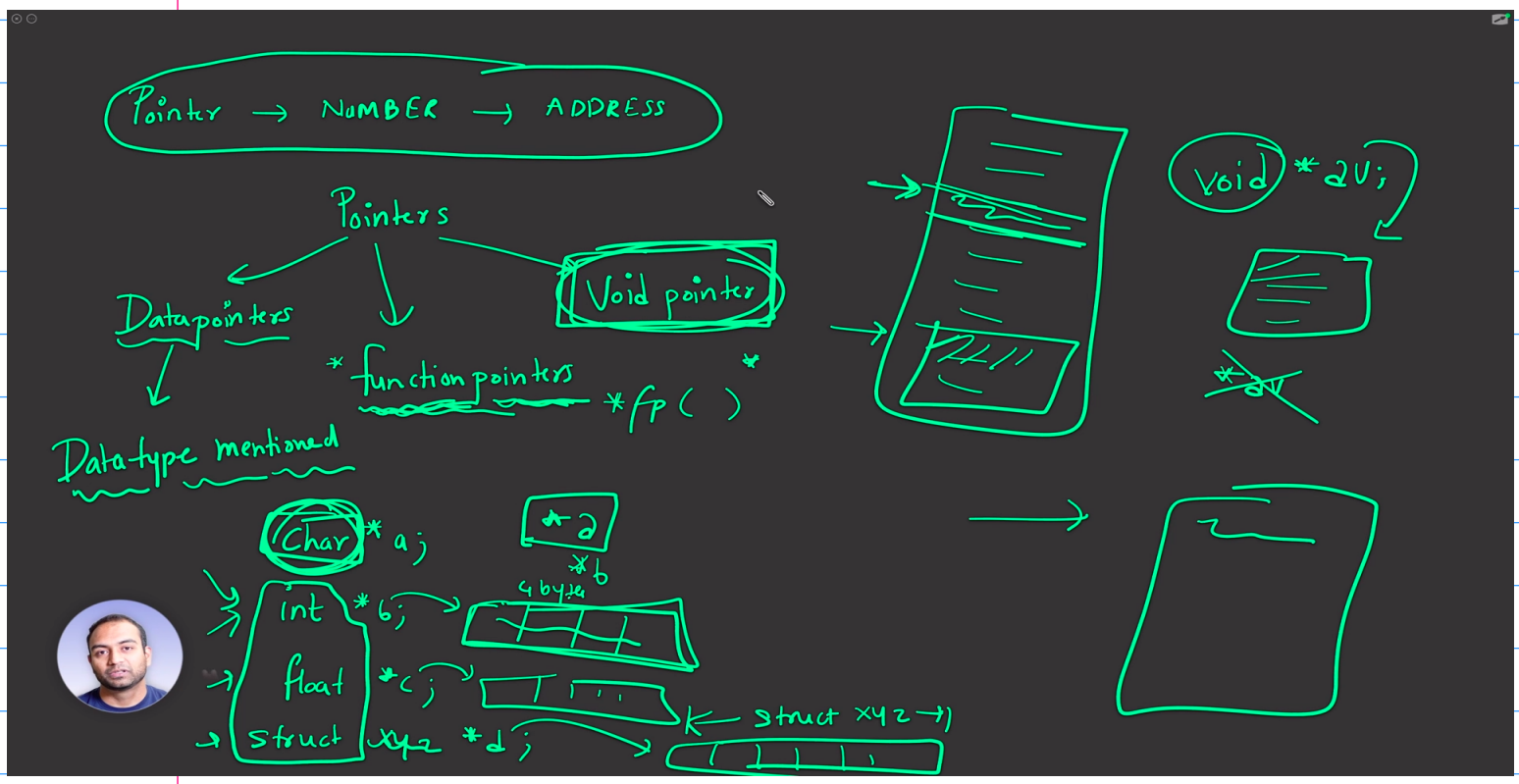
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS bash

```
@streetdogg → /workspaces/c-pointers-course (main) $ gcc main.c
@streetdogg → /workspaces/c-pointers-course (main) $ ./a.out
p: 0x7ffc47150a20
p[0]: 0x55b98cb106b0
p[1]: 0x55b98cb106d0
p[2]: 0x55b98cb106f0
p[3]: 0x55b98cb10710
p[4]: 0x55b98cb10730
p[5]: 0x55b98cb10750
p[6]: 0x55b98cb10770
p[7]: 0x55b98cb10790
p[8]: 0x55b98cb107b0
p[9]: 0x55b98cb107d0
*p[0]: 10
*p[1]: 9
*p[2]: 8
*p[3]: 7
*p[4]: 6
*p[5]: 5
*p[6]: 4
*p[7]: 3
*p[8]: 2
*p[9]: 1
sizeof(p): 80
@streetdogg → /workspaces/c-pointers-course (main) $
```

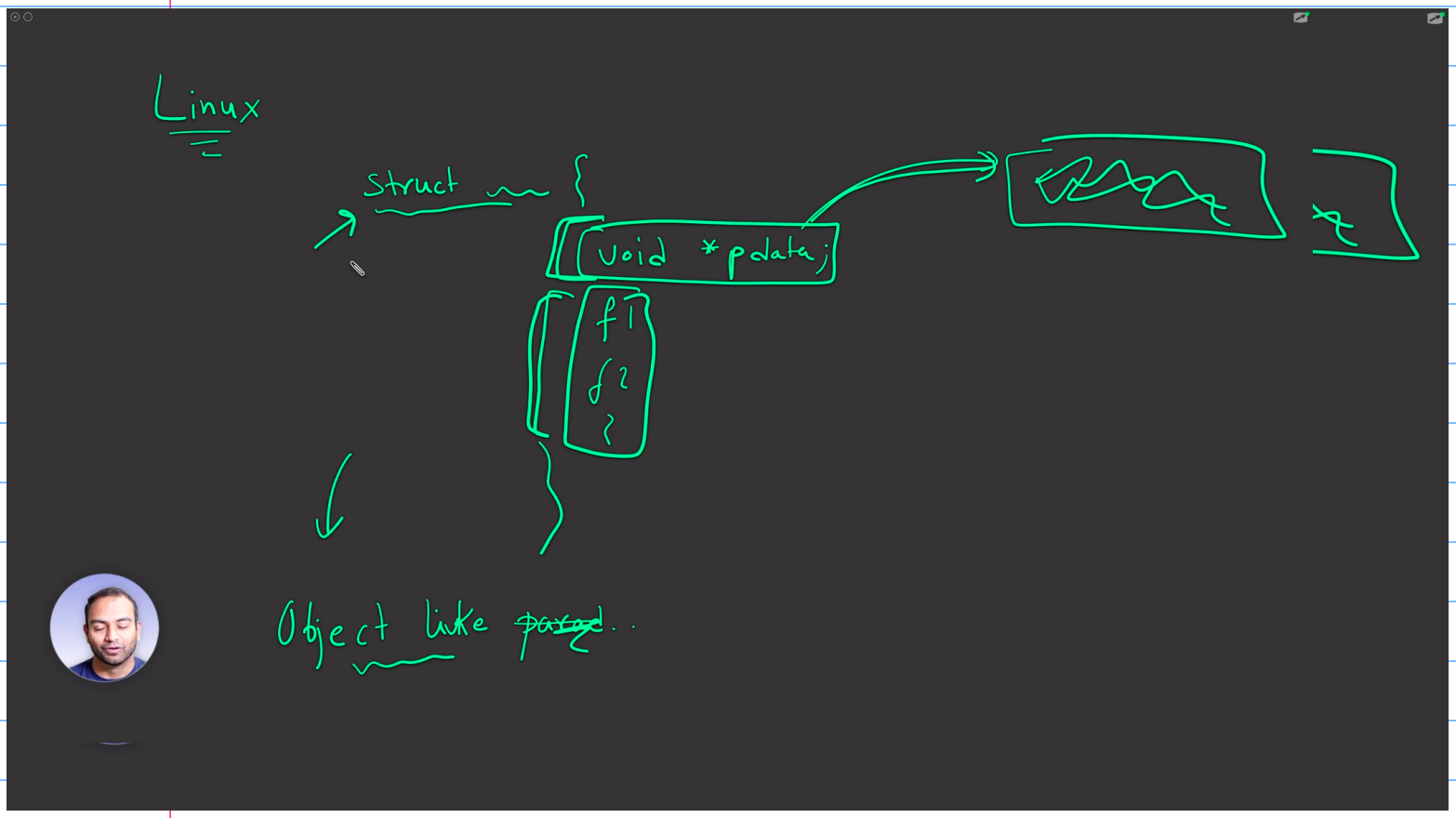
OUTLINE
TIMELINE

Codespaces: musical parakeet main* ⊗ 0 ⚠ 0 0 Ln 18, Col 26 Spaces: 4 UTF-8 LF { } C Layout: U.S. Linux

- Different Types of Pointers



- Linux is trying to follow object Oriented Paradigm

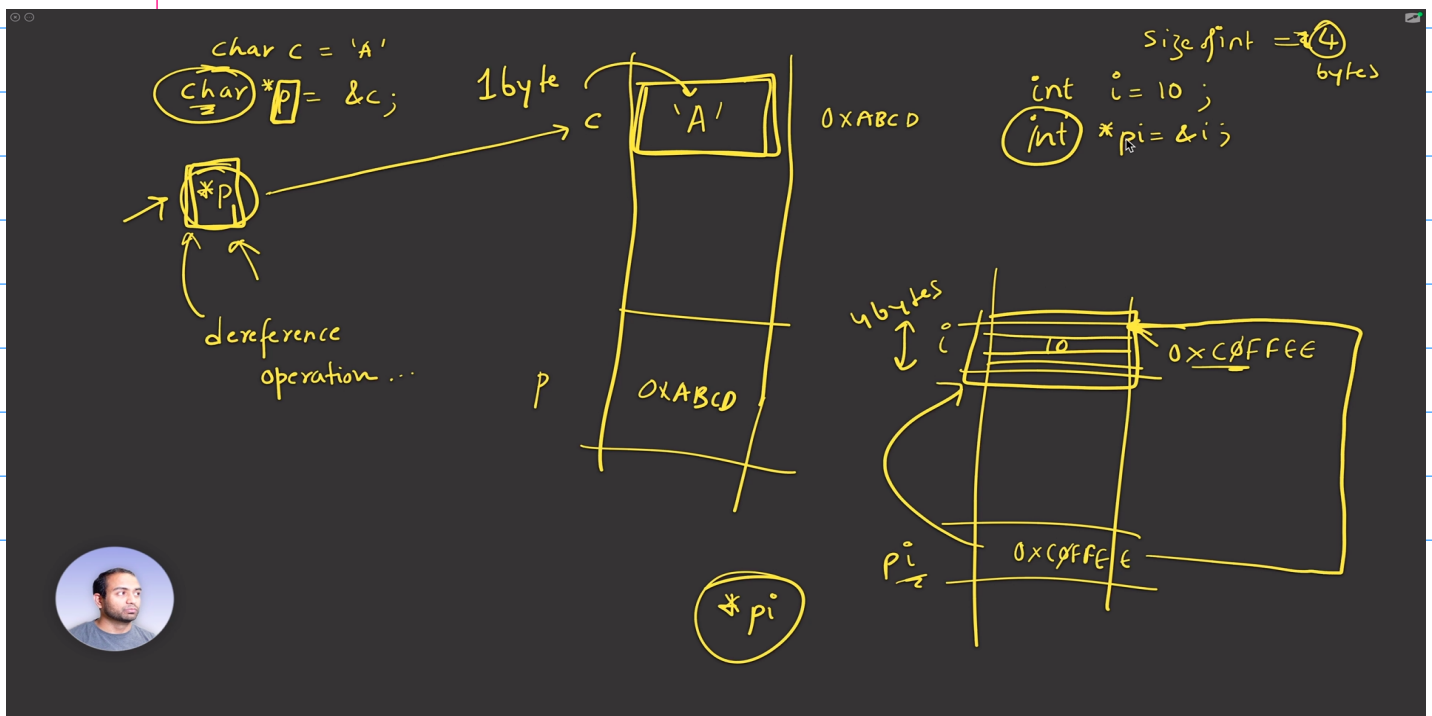


- Different types of Data Pointers

```c
#include <stdio.h>

void main() {
    char          a = 'A';
    int           b = 1024;
    float         c = 1.0;
    long long int d = 12345;

    char          *pa = &a;
    int           *pb = &b;
    float         *pc = &c;
    long long int *pd = &d;

    printf(" sizeof(*pa): %lu\n sizeof(*pb): %lu\n sizeof(*pc): %lu\n sizeof(*pd): %lu\n\n",
           sizeof(*pa),      sizeof(*pb),      sizeof(*pc),      sizeof(*pd));

    *pa = 'B';
    *pb = *pb + 1;
    *pc = *pc + 1;
    *pd = *pd + 1;

    printf(" a: %c\n b: %d\n c: %f\n d: %lld\n\n", *pa, *pb, *pc, *pd);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

a: B, b: 1025, c: 2.000000, d: 12346
@streetdogg → /workspaces/c-pointers-course (main) $ gcc main.c
@streetdogg → /workspaces/c-pointers-course (main) $ ./a.out
 sizeof(*pa): 1
 sizeof(*pb): 4
 sizeof(*pc): 4
 sizeof(*pd): 8

 a: B
 b: 1025
 c: 2.000000
 d: 12346
```

- How deferencing data pointer makes sence with respect to sizeof(data type)



- Pointer to charecter array

main.c > main()

```c
#include <stdio.h>
void main() {
    char *cptr="PRANAB-NANDY";
    char c_arr[]="pranab-nandy";
    printf("sizeof(cptr):%lu  sizeof(c_arr):%lu\n",
                                sizeof(cptr),sizeof(c_arr));

    int arr[]={11,12,13,44,55,66};
    int *ptr;
    ptr=arr;
    printf("sizeof(ptr):%lu  sizeof(arr):%lu\n",
                                sizeof(ptr),sizeof(arr));

    arr=ptr; // error
    cptr=c_arr; // ok
    c_arr=cptr; // error
    cptr[10]='$'; //ok
    printf("cptr=%s cptr=%p *cptr=%c \n",cptr,cptr,*(cptr+10));

    *&*&*&cptr[10]='@'; //ok

    int i=10,*pi,**ppi,***pppi;
    pi=&i;  ppi=&pi;  pppi=&ppi;
}
```

```
@PranabNandy → /workspaces/c-pointers-course (main) $ htop
@PranabNandy → /workspaces/c-pointers-course (main) $ htop
@PranabNandy → /workspaces/c-pointers-course (main) $ gcc ma
in.c
main.c: In function ‘main’:
main.c:14:8: error: assignment to expression with array type
   14 |     arr=ptr; // error
      |        ^
main.c:16:10: error: assignment to expression with array typ
e
   16 |     c_arr=cptr; // error
      |          ^
@PranabNandy → /workspaces/c-pointers-course (main) $
```

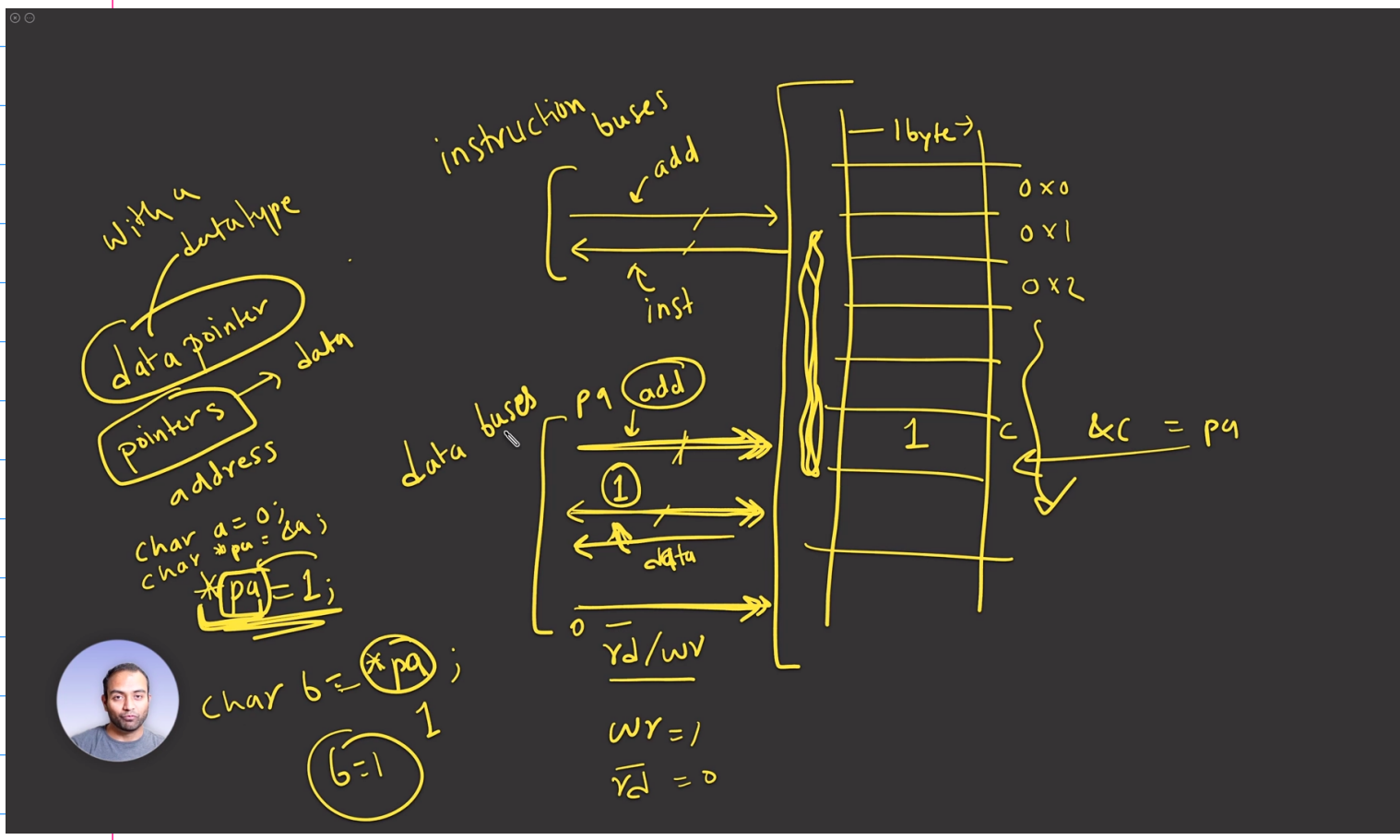
- Pointers to structure
- Pointer Arithmetic

main.c > main()

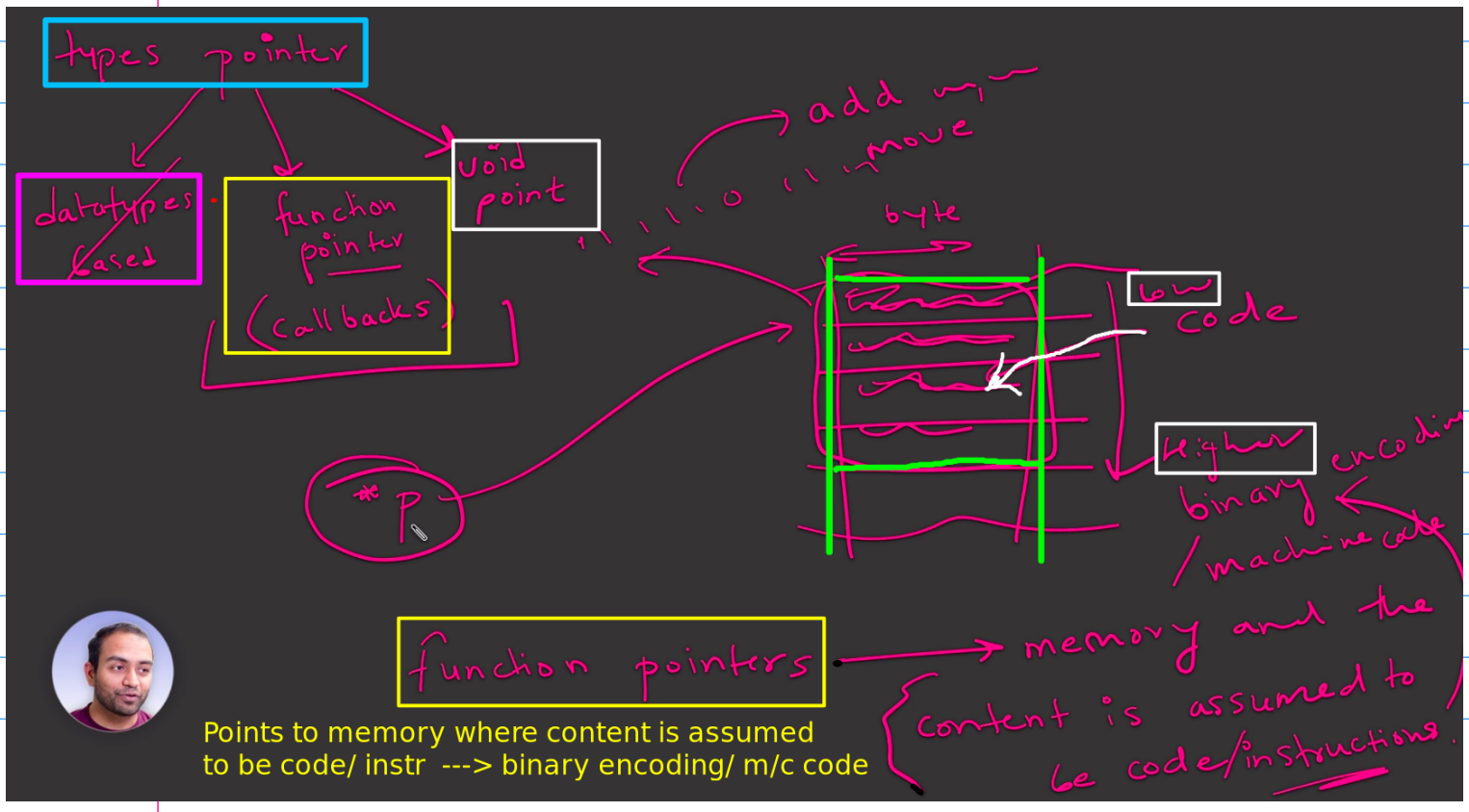
```c
#include <stdio.h>
struct main
{
    long long int a;
    float b;
    char c;
    struct main *next;
};
void main() {
    struct main var;
    struct main *p_var;
    p_var=&var;
    /* There are 3 different ways */
    var.a=100;
    (*p_var).b=200.0;
    p_var->c='P';
    printf("No Error ==> 'struct main' and 'main()' \n");
    /* ---------- Pointer Arithmetic ---------- */
    p_var=p_var - 5; / p_var - 5*sizeof( datatype)

}
```

```
@PranabNandy → /workspaces/c-pointers-course (main) $ gcc m
ain.c
@PranabNandy → /workspaces/c-pointers-course (main) $ ./a.o
ut
No Error ==> 'struct main' and 'main()'
@PranabNandy → /workspaces/c-pointers-course (main) $
```

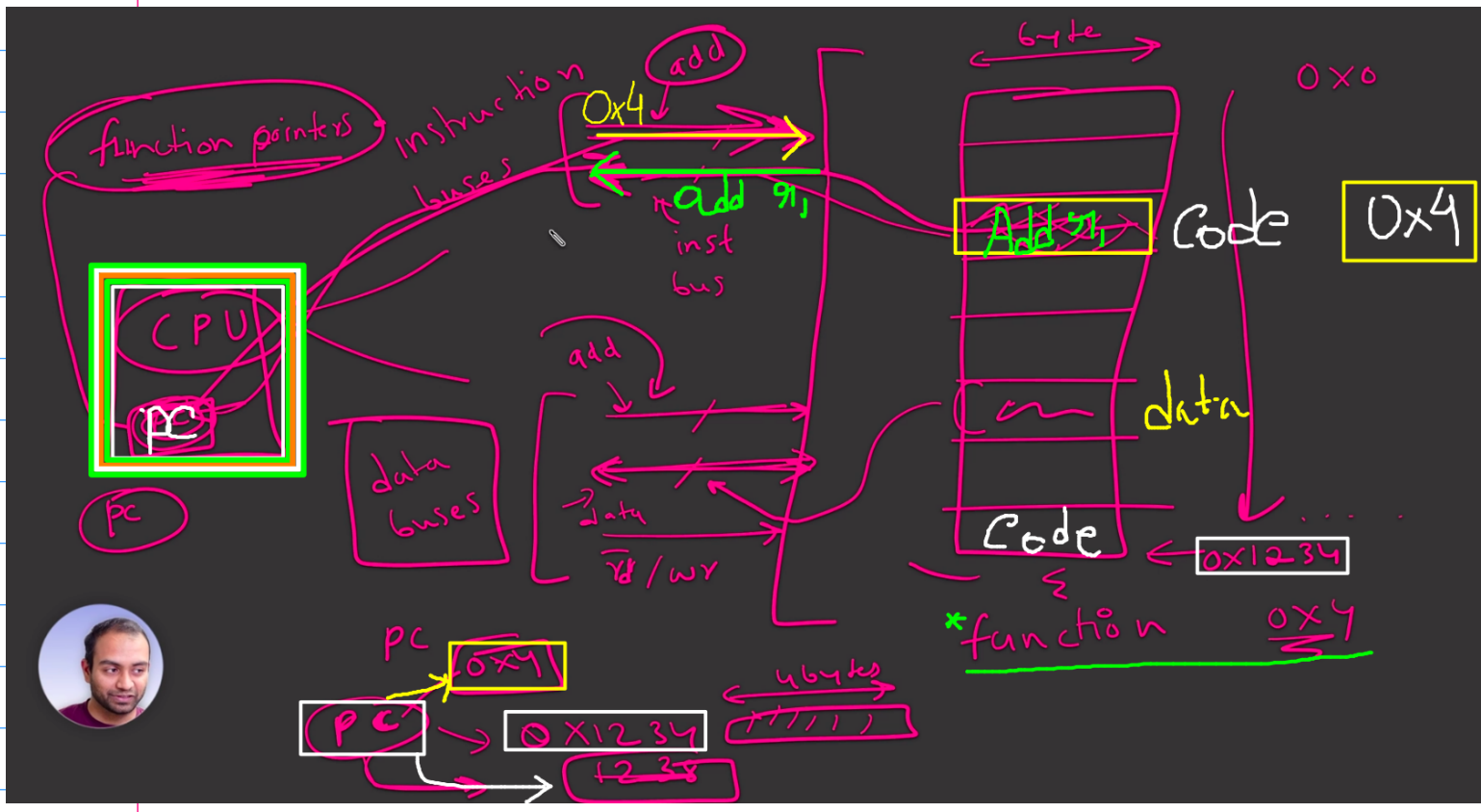
- Align data pointer concept with computer model



- function pointer (also called callback)



- function Ponters cond..




```c
#include <stdio.h>
void fun1(int a){
    printf("fun1 : a = %d\n",a);
}
void fun2(int b){
    printf("fun2 : b = %d\n",b);
}
// defines a new function type fun_ptr
typedef void (*fun_ptr)(int);
int main() {
    fun_ptr fptr;
    fptr=fun1;
    fptr(10);

    fptr=&fun2;
    fptr(22);
    return 0;
}
```

```
@PranabNandy → /workspaces/c-pointers-course (main) $ ./a.out
fun1 : a = 10
fun2 : b = 22
@PranabNandy → /workspaces/c-pointers-course (main) $
```

- greatest example of Function Pointer

main.c M

```c
#include "math_lib.h"

int main() {
    execute_operation(ADD,10,
    execute_operation(DIV,10,
}
```

math_lib.h

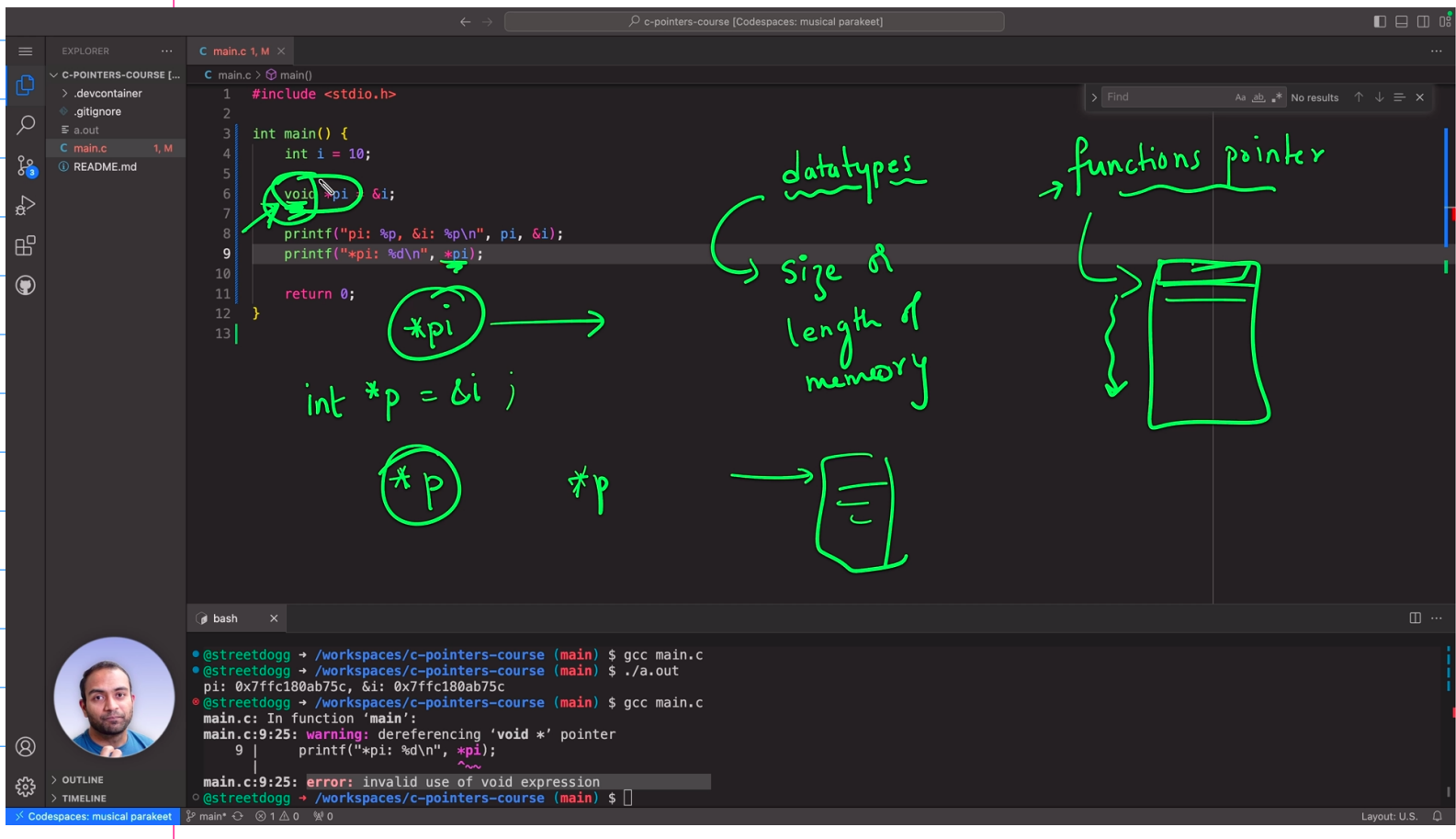
```c


#define ADD 0
#define DIV 1

void execute_operation(
    int operation,int op1,int op2)
```

math_lib.c

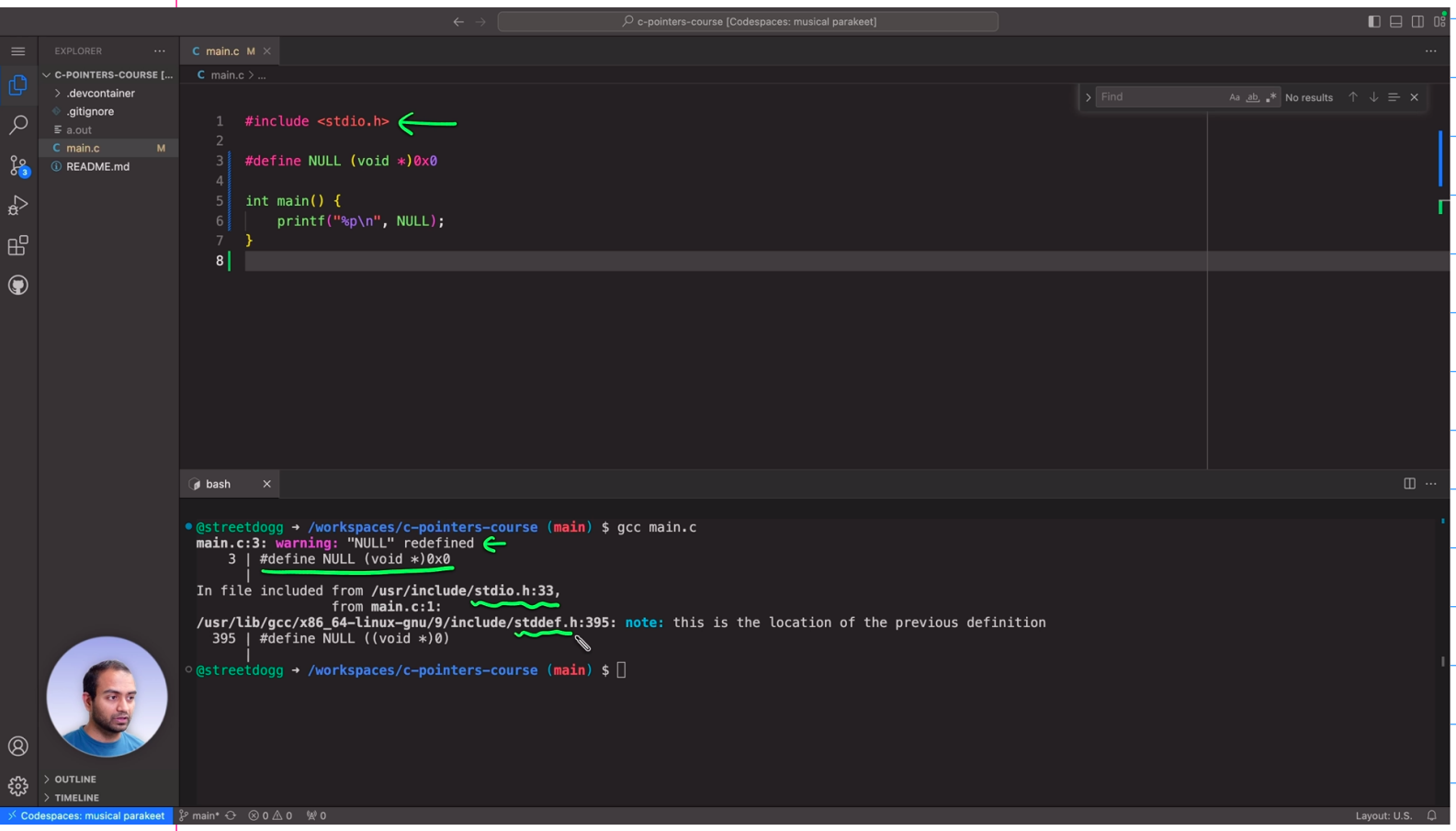
```c
#include<stdio.h>
#include "math_lib.h"

void add(int op1,int op2){
    printf("addition is : %d\n",op1+op2);
}
void div(int op1,int op2){
    printf("division is : %d\n",op1/op2);
}

typedef void (*fun_ptr)(int ,int);
fun_ptr fptr[2]={add,div};

void execute_operation(int operation,int op1,int
    fptr[operation](op1,op2);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

@PranabNandy → /workspaces/c-pointers-course (main) $ gcc main.c math_lib.c
@PranabNandy → /workspaces/c-pointers-course (main) $ ./a.out
addition is : 12
division is : 5
@PranabNandy → /workspaces/c-pointers-course (main) $
```

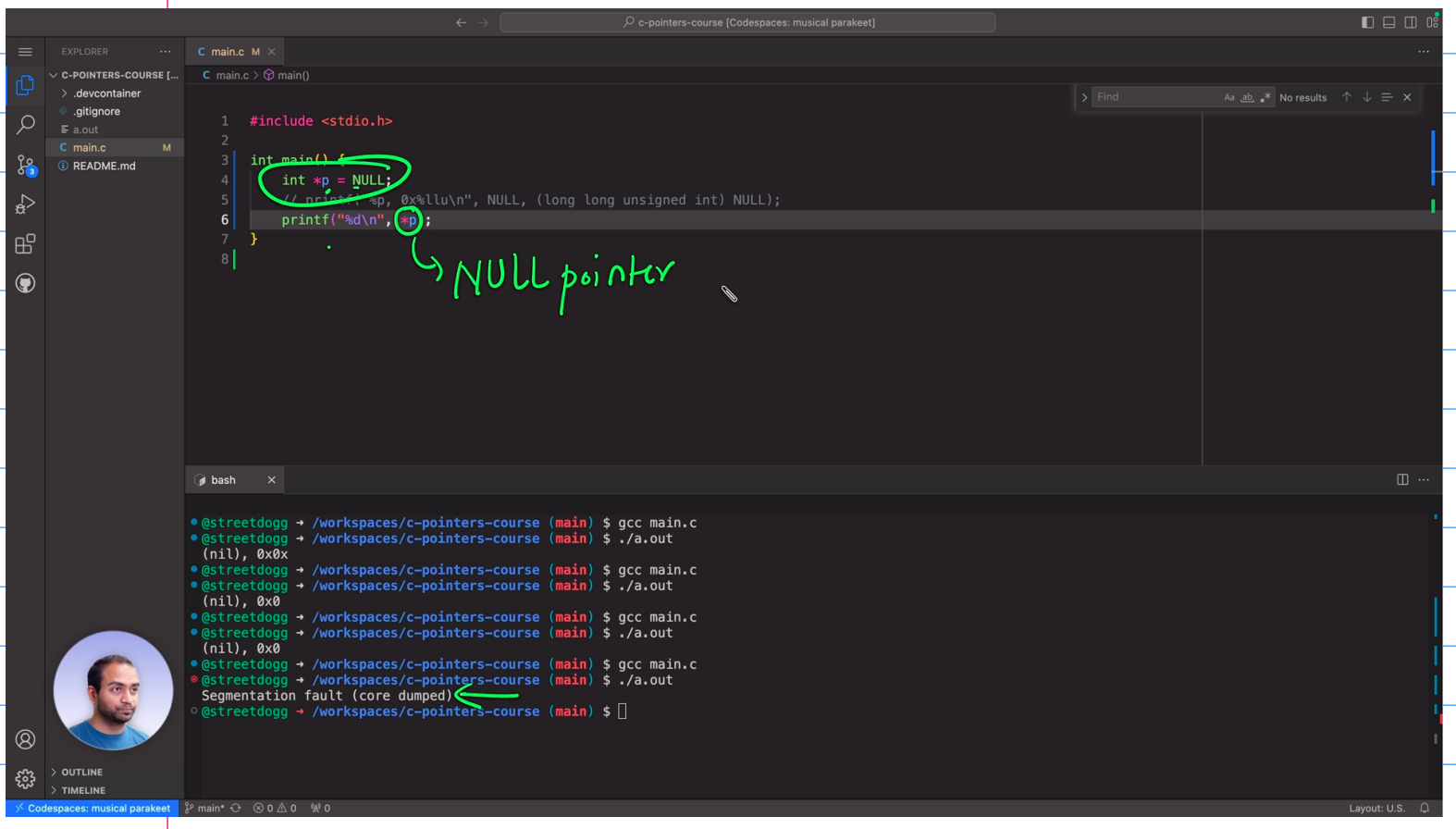
- Example of void pointer



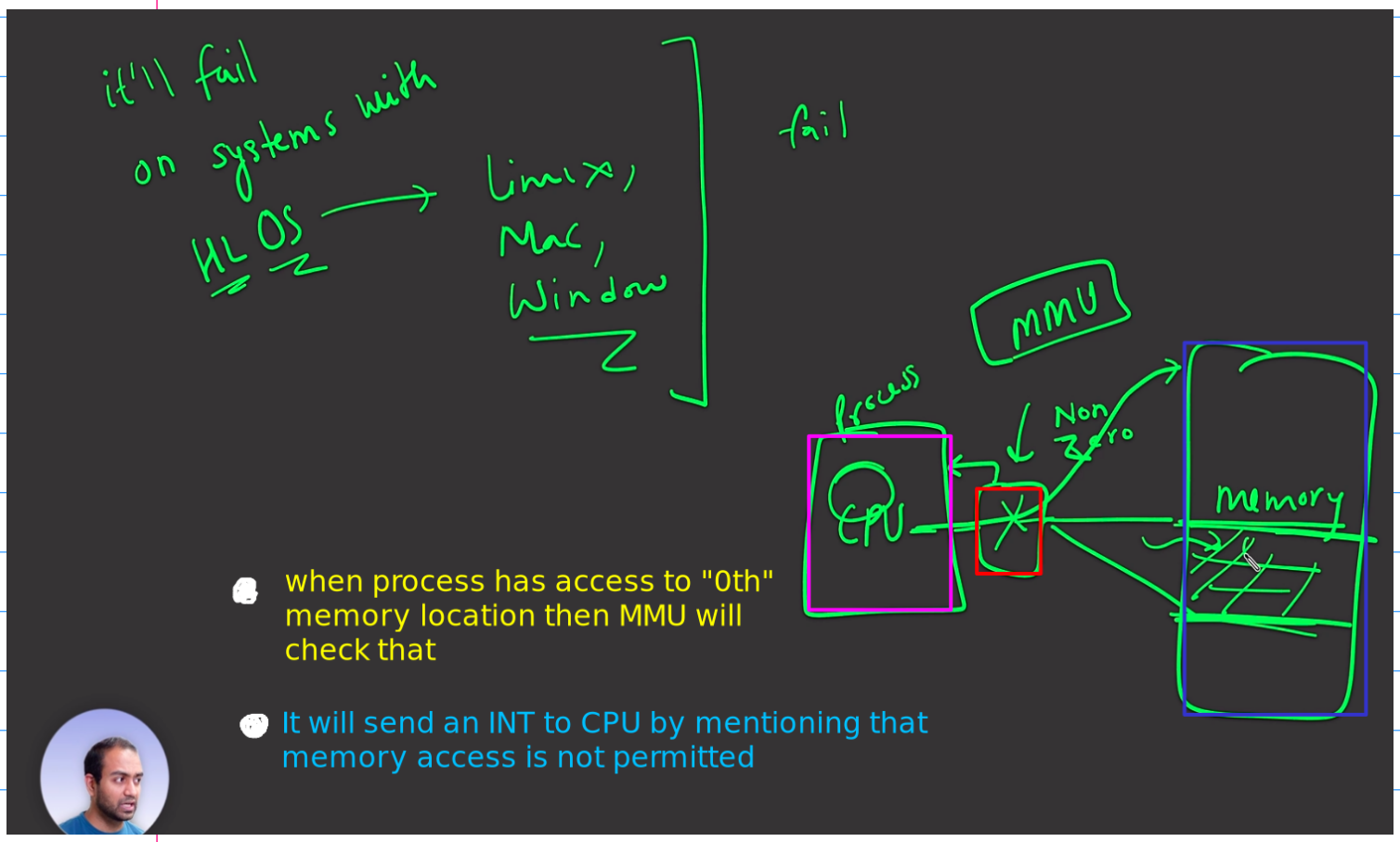
- #deifine NULL (void*)0x0

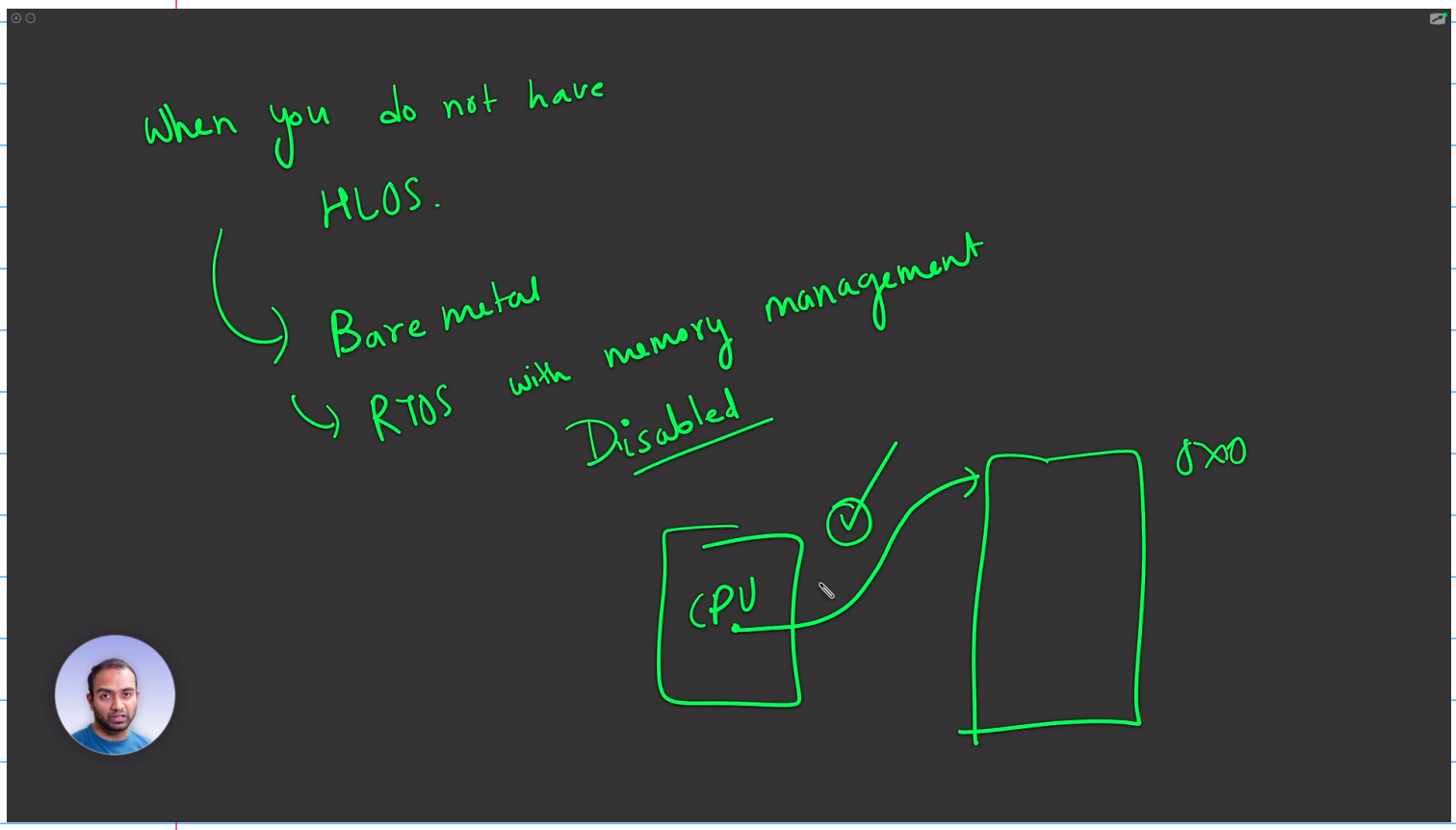


- we can not access NULL pointer

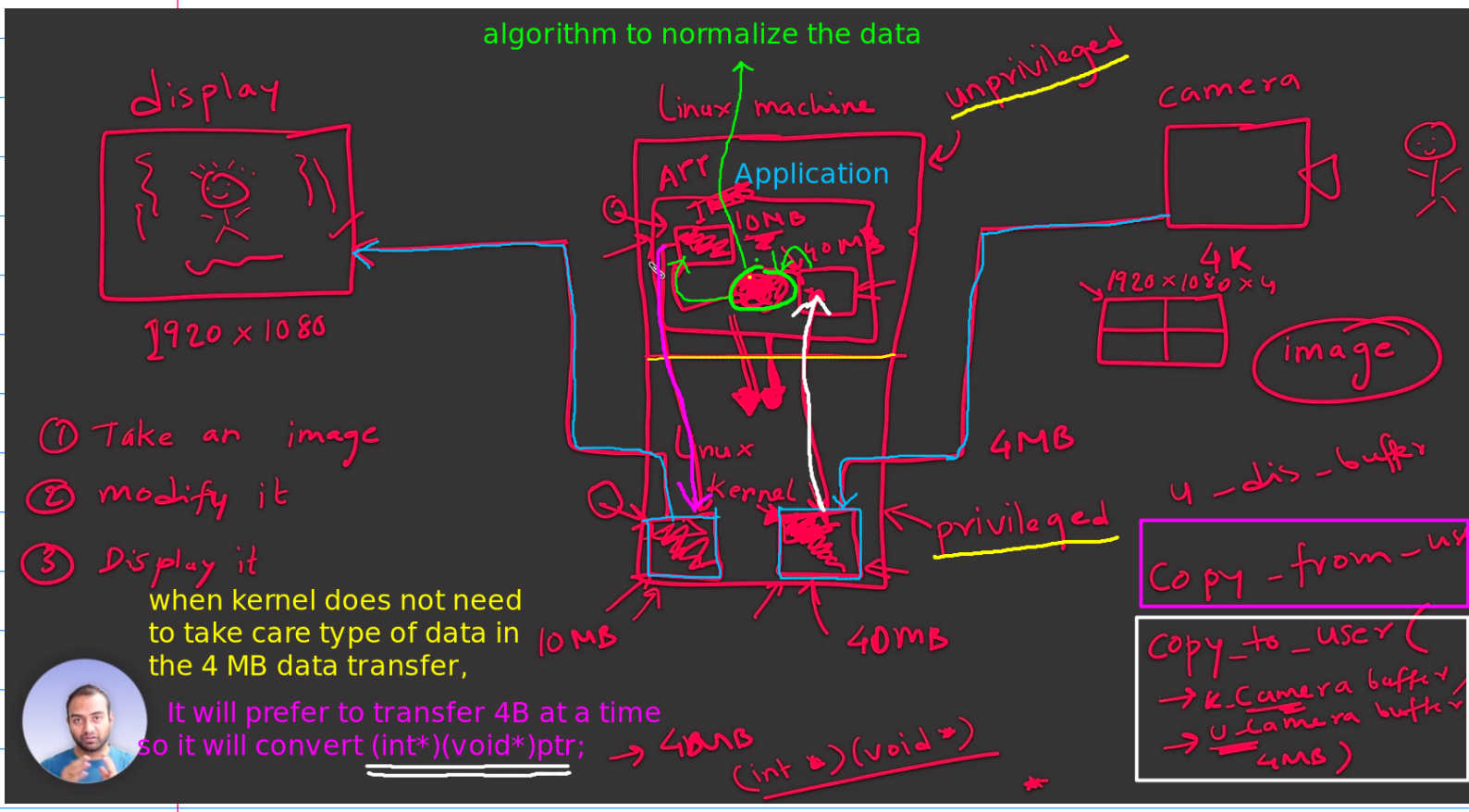


- When we can acess 0th location and when we can't ?





- Super use case example of Void Pointer



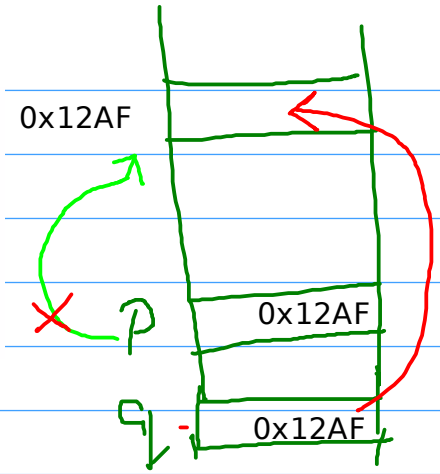
- **Heap:** Operating System

  Linux, Windows, Mac etc.

  RTOS

SYNOPSIS top

```
#include <stdlib.h>

void *malloc(size_t size);
void free(void *_Nullable ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *_Nullable ptr, size_t size);
void *reallocarray(void *_Nullable ptr, size_t nmemb, size_t size);
```



- **Dangling Pointer:**

```
int *p=(int*)malloc(4);
free(p);   q=(int*) malloc(4);
*p=4000;
```

here q got the same address

corrupting the actual data (i.e data of q)

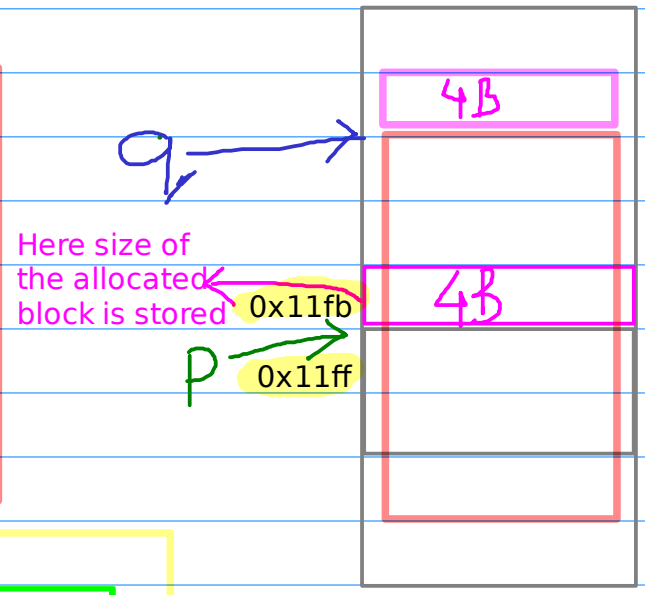
- It is better to use ,

```
free(p);
p=NULL;
```

- How free() works? ----> It will go back to 4B then check the value of that address & free up the space

```
p=(int*)malloc(128);
free(p);
q=(int*)malloc(16*1024);
[ now *(0x11fb)=16KB stored by
some piece code as it was using
with respect to "q" block]
free(p);
```

It will free the allocated block of "q" also



- size_t ---> unsigned int / int