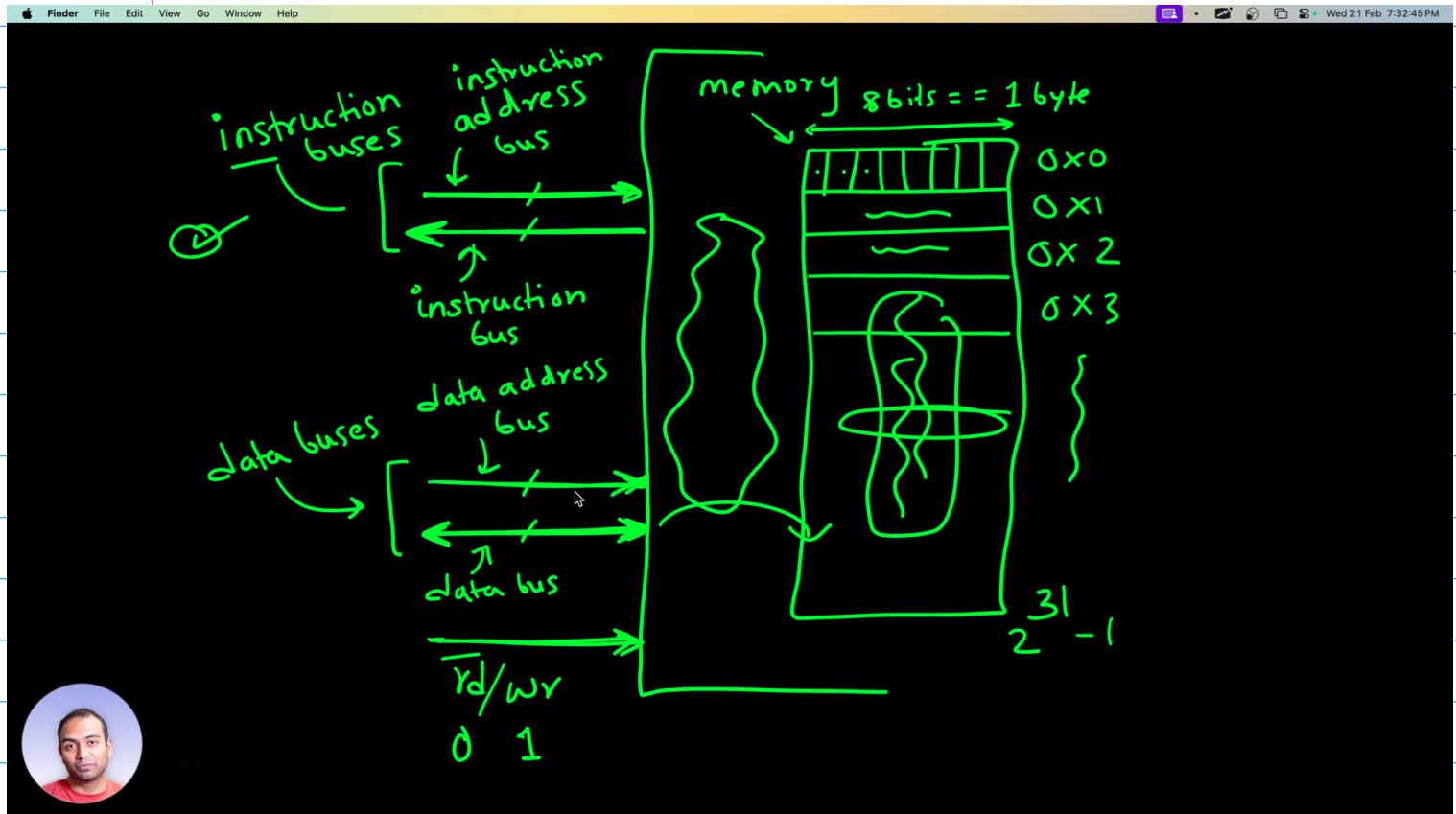
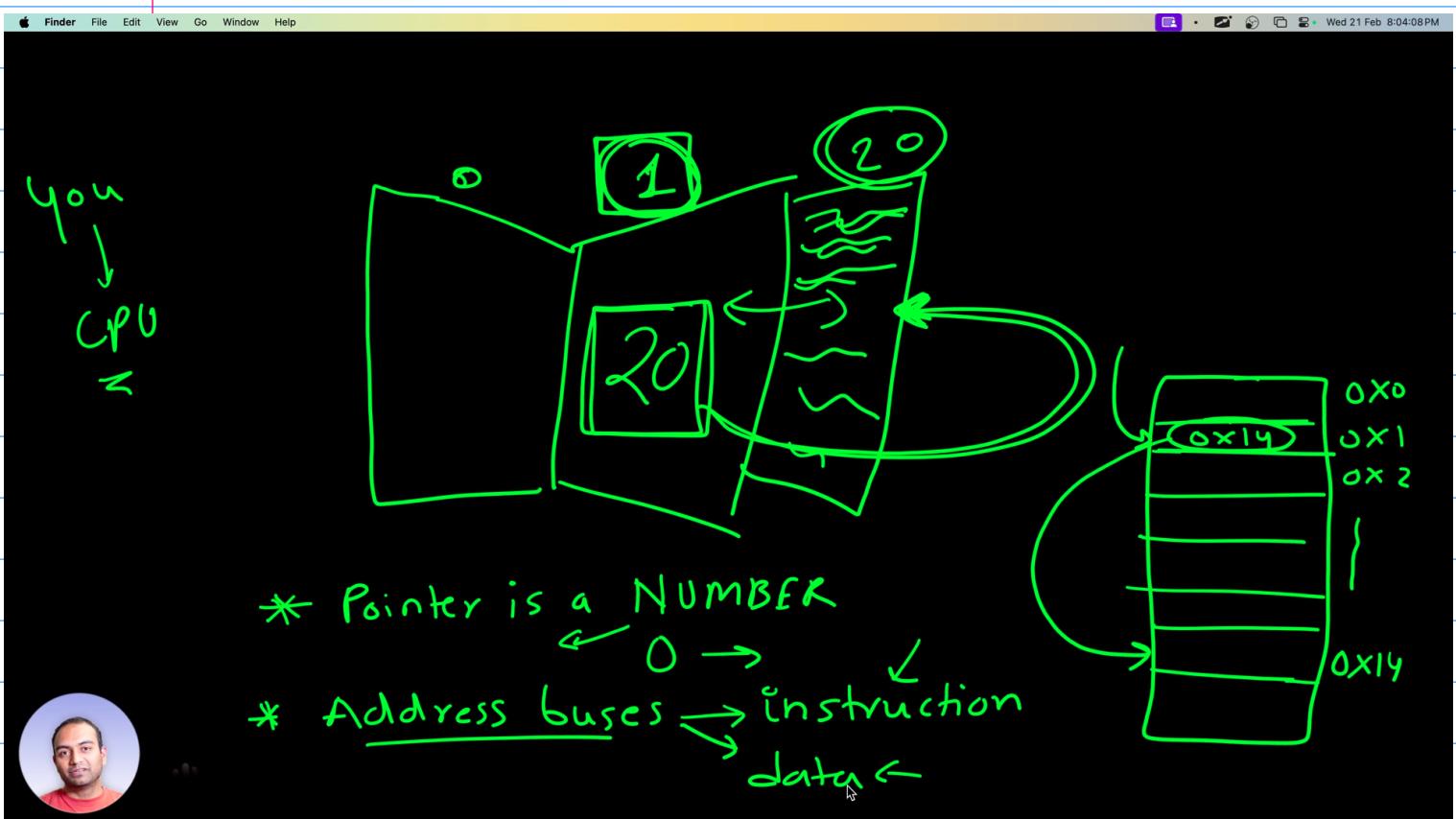


- C Pointers:
- Secrets every
- Embedded
- Engineer
- MUST know!

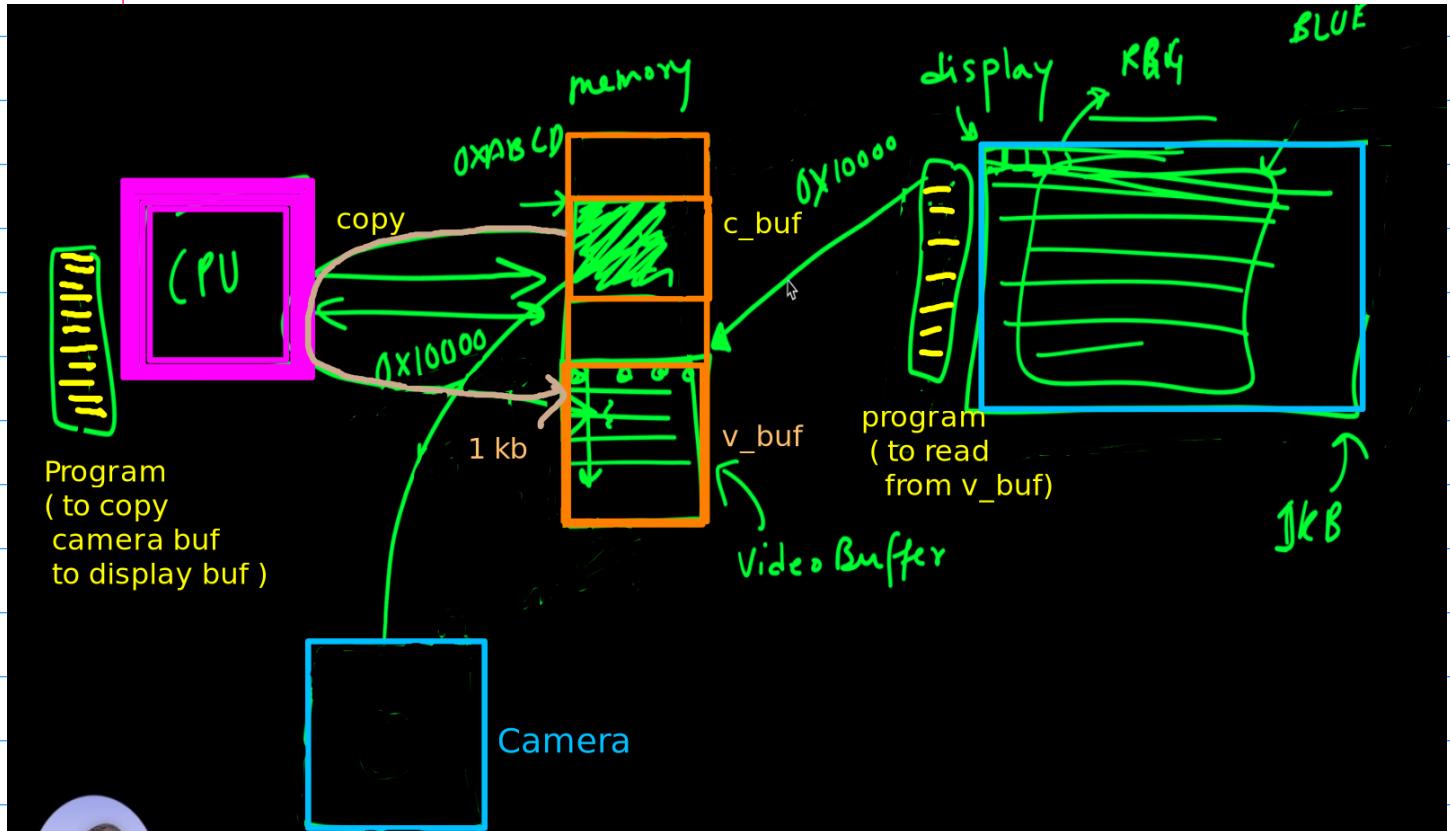
- Computer System with Data Bus and Address Bus



- What is pointer in computer system?



● How pointers and addresses are related to computer system?



● Basic of Pointers in Embedded C

```

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

main.c
2
3 void main() {
4     char c = 'A';
5     char *c_ptr = &c;
6
7     printf("c=%c, &c=%p\n", c, &c);
8     printf("c_ptr=%p\n", c_ptr);
9     printf("&c_ptr=%p\n", &c_ptr); c_ptr
10 }
11

```

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
- c=A, &c=0x7ffd97b91bbf
- c_ptr=0x7ffd97b91bbf
- &c_ptr=0x7ffd97b91bc0

● Some tricky concept in pointers

The terminal shows the following C code:

```
main.c > main()
1 #include <stdio.h>
2 void main() {
3     int i=0xFF;
4     char c='A';
5     char *cptr;
6     int x;
7     cptr=&c;
8     x= i & cptr; // error
9     x= i & & c; // error
10    x= i && c; // ok - logical AND
11    x= i & (int)&c; // ok
12    x = i * cptr; // error
13    x = i ** cptr; // ok
14    printf("x= i & cptr; --> %x\n",x);
15    printf("cptr=%p\n",cptr);
16    printf("cptr=0x%llx\n", (long long unsigned int)cptr);
17 }
```

Compilation errors:

- Line 8: `x= i & cptr;` // error
- Line 9: `x= i & & c;` // error
- Line 10: `x= i && c;` // ok - logical AND
- Line 11: `x= i & (int)&c;` // ok
- Line 12: `x = i * cptr;` // error
- Line 13: `x = i ** cptr;` // ok
- Line 14: `printf("x= i & cptr; --> %x\n",x);`
- Line 15: `printf("cptr=%p\n",cptr);`
- Line 16: `printf("cptr=0x%llx\n", (long long unsigned int)cptr);`
- Line 17: `}`

Warnings:

- Line 11: `warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]`
- Line 12: `error: invalid operands to binary * (have 'int' and 'char *)'`

Terminal status:

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

● Multi-level Pointers

The terminal shows the following C code:

```
main.c > main()
1 #include <malloc.h>
2
3 void main() {
4     char *p[10];
5
6     printf("p: %p\n", p);
7     for (int i=0; i<10; i++) {
8         p[i] = (char *)malloc(1);
9         *p[i] = 10 - i;
10        printf("p[%d]: %p\n", i, p[i]);
11    }
12
13    for (int i=0; i<10; i++) {
14        for (int j=0; j<10; j++) {
15            if (i==j) {
16                printf("p[%d][%d]: %c\n", i, j, *p[i]);
17            }
18        }
19    }
20 }
```

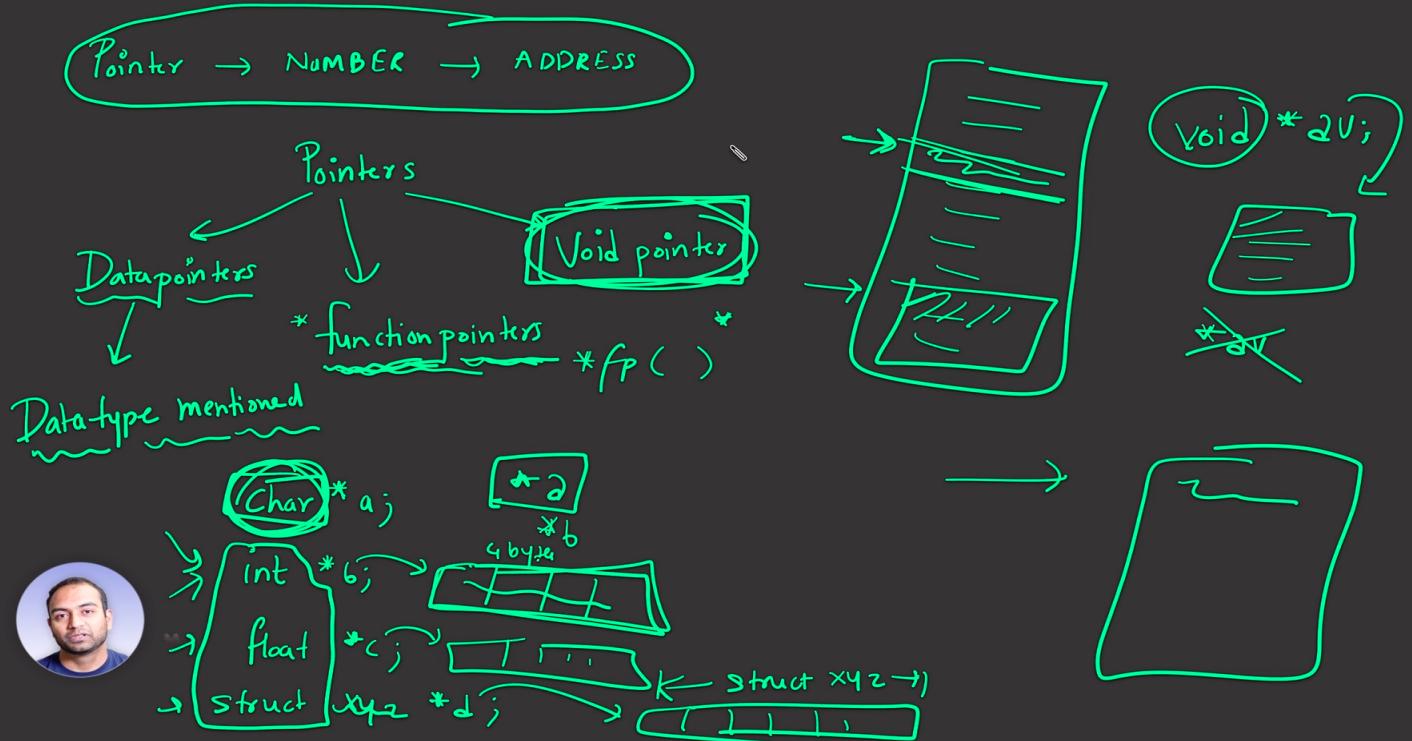
Output:

```
@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) $
```

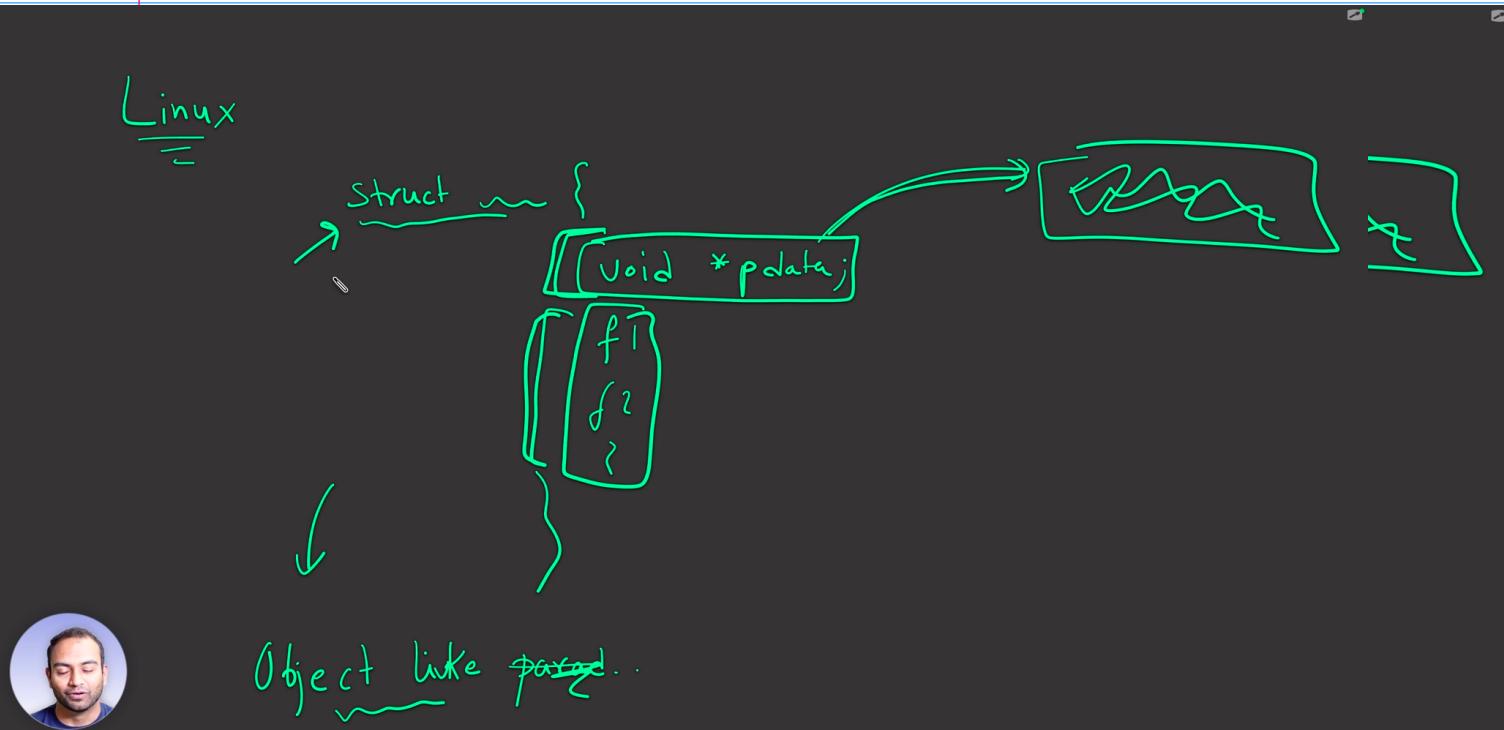
Diagram illustrating multi-level pointers:

A vertical stack of 10 memory locations, each containing a character. The top location contains '0', the second contains '1', the third contains '2', the fourth contains '3', the fifth contains '4', the sixth contains '5', the seventh contains '6', the eighth contains '7', the ninth contains '8', and the bottom one contains '9'. A pointer variable `p` at the top of the stack points to the first character. Hand-drawn arrows show the pointer `p` pointing to the first character, and subsequent characters being printed.

- Different Types of Pointers



- Linux is trying to follow object Oriented Paradigm



● Different types of Data Pointers

```

C main.c > main()
1 #include <stdio.h>
2
3 void main() {
4     char           ✓ a = 'A';
5     int            ✓ b = 1024;
6     float          ✓ c = 1.0;
7     long long int ✓ d = 12345;
8
9     char           *pa = &a;
10    int             *pb = &b;
11    float           *pc = &c;
12    long long int  *pd = &d;
13
14    printf(" sizeof(*pa): %lu\n sizeof(*pb): %lu\n sizeof(*pc): %lu\n sizeof(*pd): %lu\n\n",
15           sizeof(*pa),      sizeof(*pb),      sizeof(*pc),      sizeof(*pd));
16
17    *pa = 'B';
18    *pb = *pb + 1;
19    *pc = *pc + 1;
20    *pd = *pd + 1;
21
22    printf(" a: %c\n b: %d\n c: %f\n d: %lld\n\n", *pa, *pb, *pc, *pd);
23 }

```

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

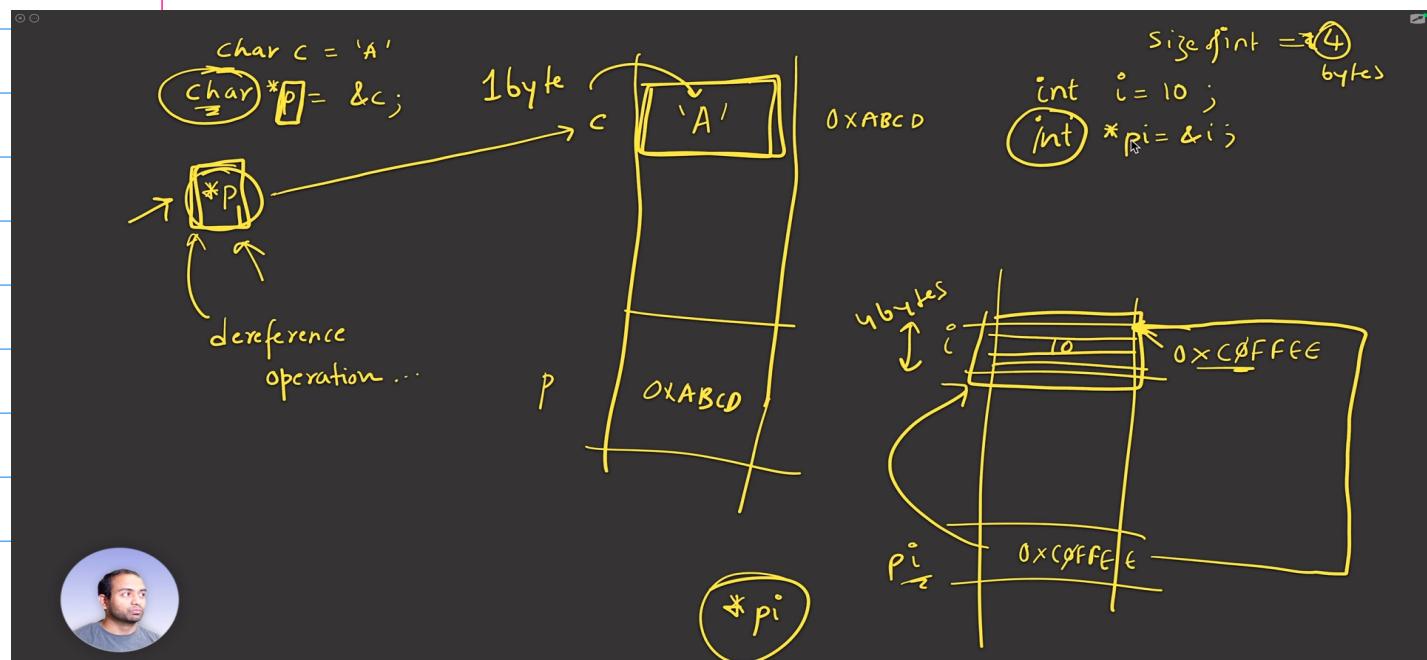
```

```

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

```

● How referencing data pointer makes sense with respect to sizeof(data type)



● Pointer to character array

```
main.c > main()
1 #include <stdio.h>
2 void main() {
3     char *cptr="PRANAB-NANDY";
4     char c_arr[]="pranab-nandy";
5     printf("sizeof(cptr):%lu sizeof(c_arr):%lu\n",
6            | | | | | sizeof(cptr),sizeof(c_arr));
7
8     int arr[]={11,12,13,44,55,66};
9     int *ptr;
10    ptr=arr;
11    printf("sizeof(ptr):%lu sizeof(arr):%lu\n",
12           | | | | | sizeof(ptr),sizeof(arr));
13
14    arr=ptr; // error
15    cptr=c_arr; // ok
16    c_arr=cptr; // error
17    cptr[10]='$'; //ok
18    printf("cptr=%s cptr=%p *cptr=%c \n",cptr,cptr,*cptr+10);
19
20    *cptr[10]='@'; //ok
21
22    int i=10,*pi,**ppi,***pppi;
23    pi=&i; ppi=&pi; pppi=&ppi;
24 }
```

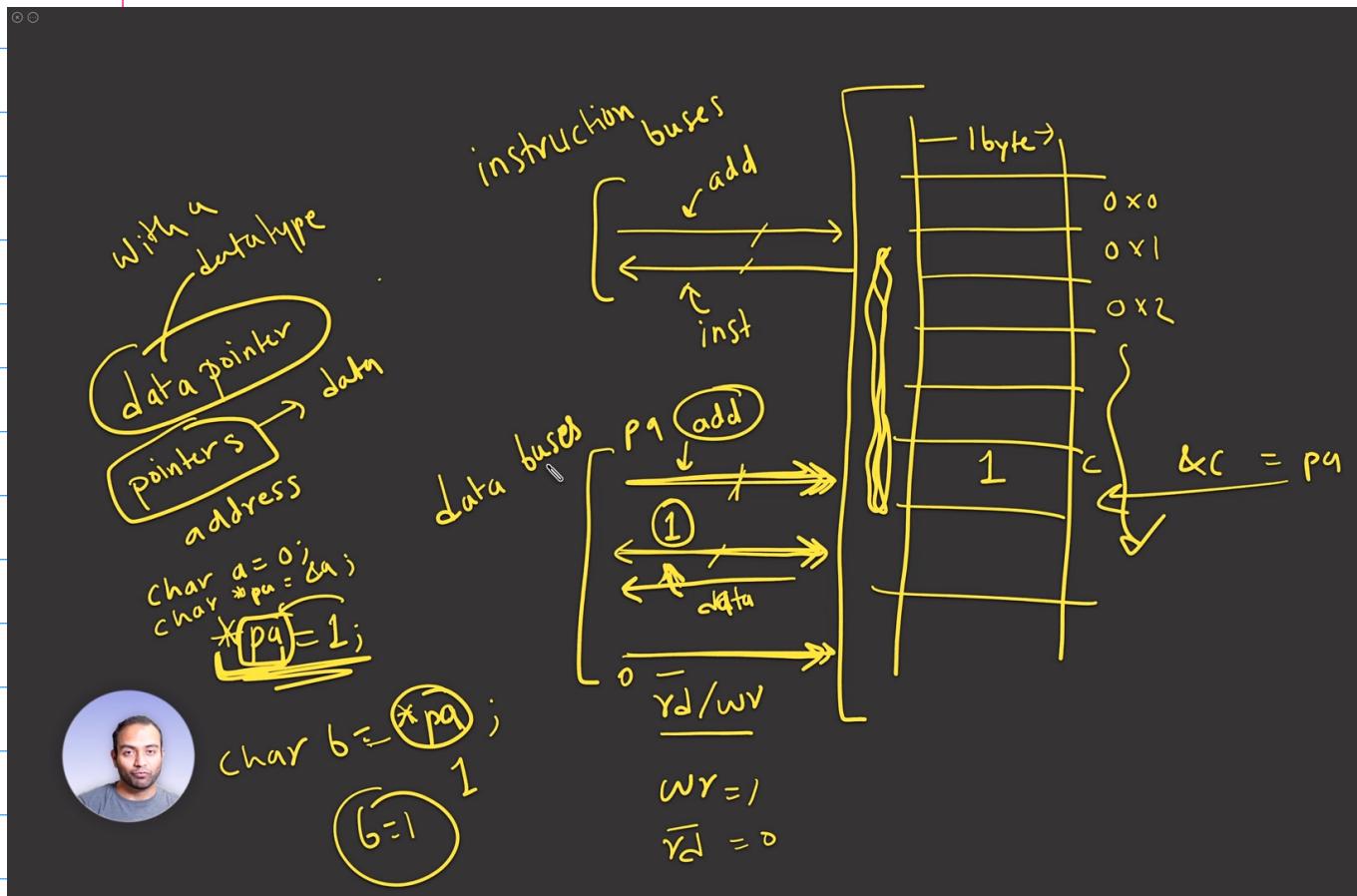
```
@PranabNandy → /workspaces/c-pointers-course (main) $ htop
@PranabNandy → /workspaces/c-pointers-course (main) $ htop
@PranabNandy → /workspaces/c-pointers-course (main) $ gcc main.c
main.c: In function 'main':
main.c:14:8: error: assignment to expression with array type
      14 |     arr=ptr; //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

- Pointers to structure
- Pointer Arithmetic

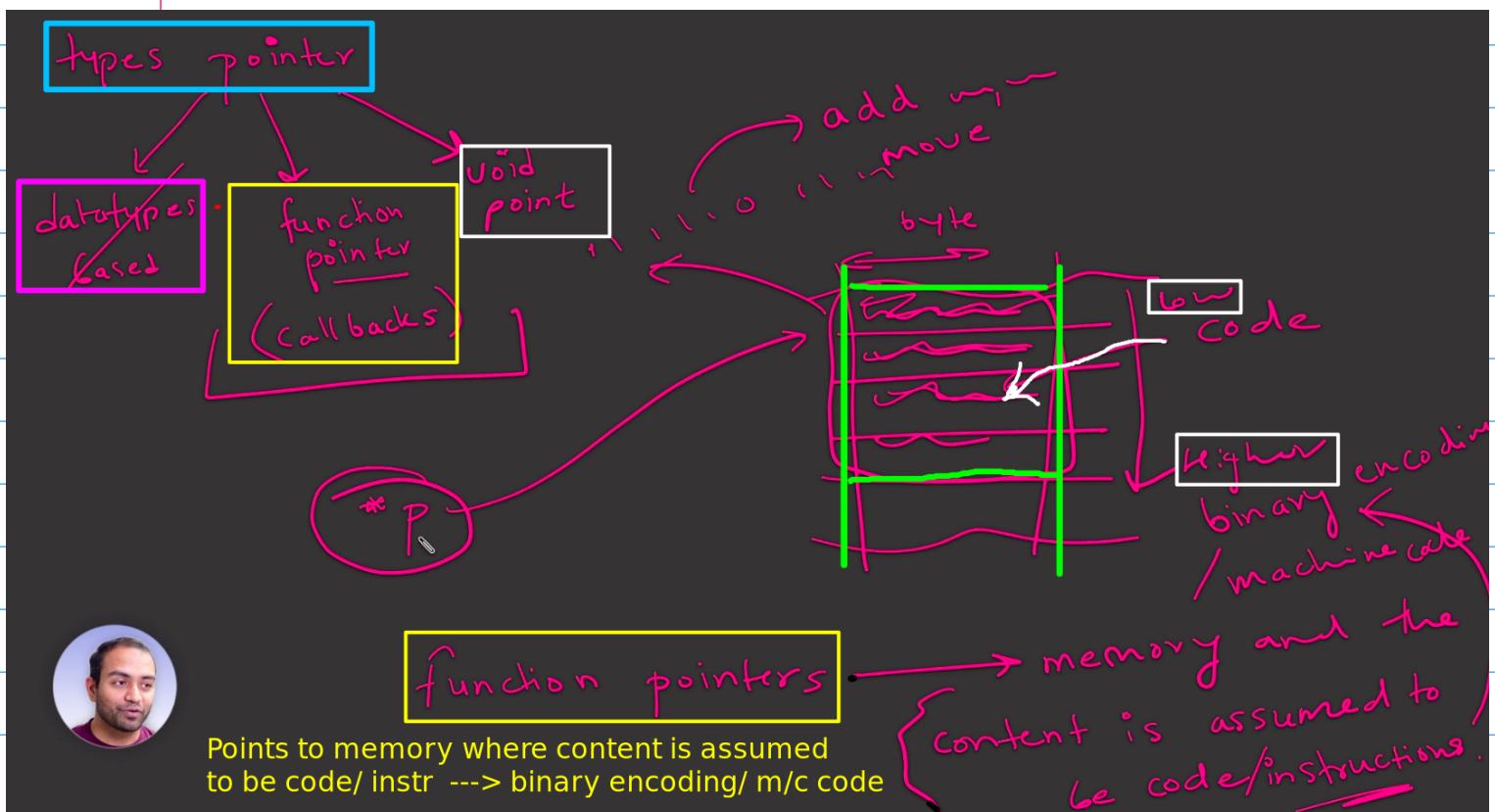
```
C main.c > main()
1 #include <stdio.h>
2 struct main
3 {
4     long long int a;
5     float b;
6     char c;
7     struct main *next;
8 };
9 void main() {
10    struct main var;
11    struct main *p_var;
12    p_var=&var;
13    /* There are 3 different ways */
14    var.a=100;
15    (*p_var).b=200.0;
16    p_var->c='P';
17    printf("No Error ==> 'struct main' and 'main()' \n");
18    /* ----- Pointer Arithmetic ----- */
19    p_var=p_var - 5; // p_var - 5*sizeof( datatype)
20 }
```

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

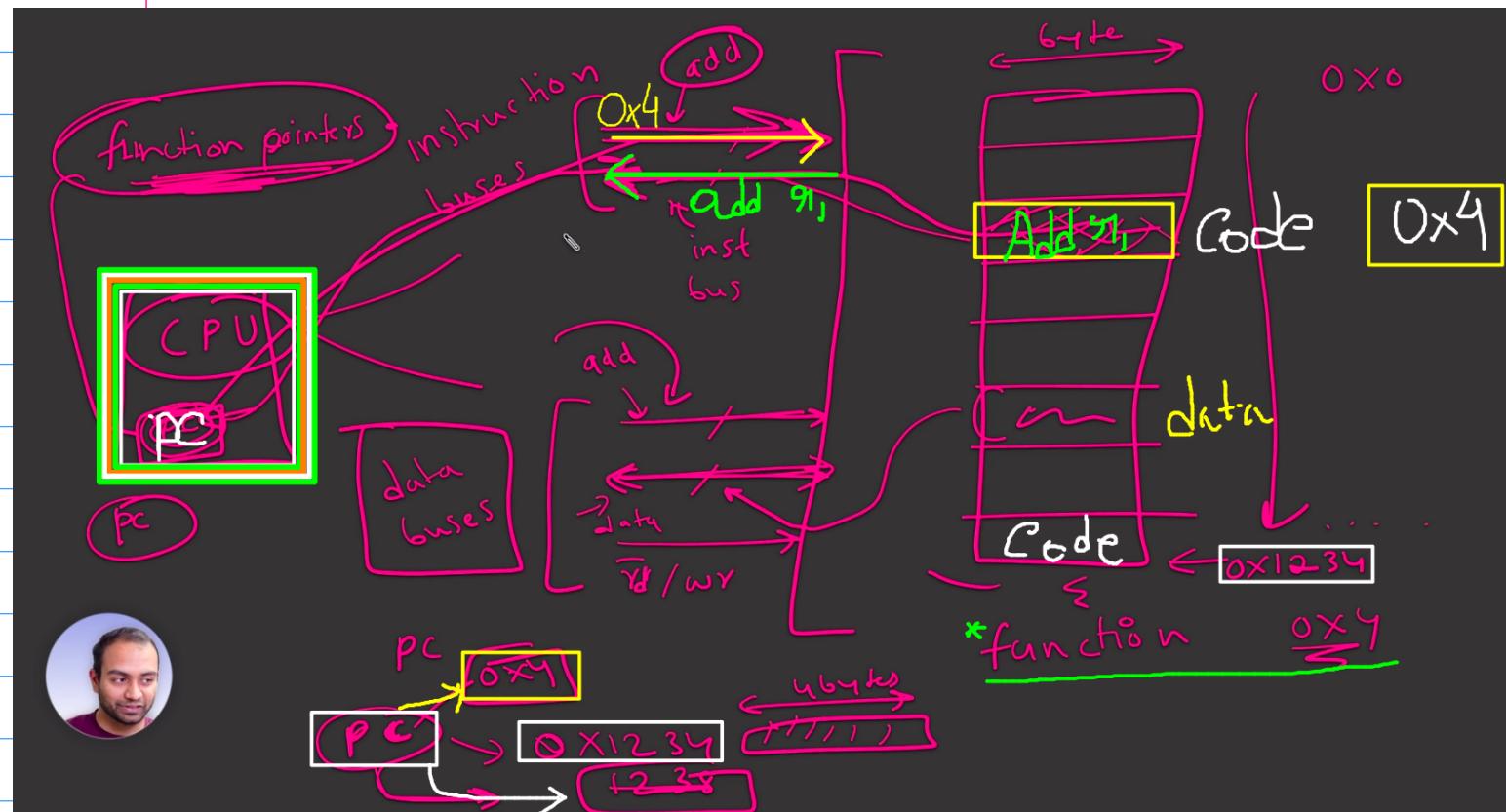
● Align data pointer concept with computer model



● function pointer (also called callback)



● function Pointers cond..



```
c main.c > ...
1 #include <stdio.h>
2 void fun1(int a){
3     printf("fun1 : a = %d\n",a);
4 }
5 void fun2(int b){
6     printf("fun2 : b = %d\n",b);
7 }
8 // defines a new function type fun_ptr
9 typedef void (*fun_ptr)(int);
10 int main() {
11     fun_ptr fptr;
12     fptr=fun1;
13     fptr(10);
14
15     fptr=&fun2;
16     fptr(22);
17     return 0;
18 }
```

```
• @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

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

EXPLORER C main.c 1, M

```
1 #include <stdio.h>
2
3 int main() {
4     int i = 10;
5     void *pi;
6     printf("pi: %p, &i: %p\n", pi, &i);
7     printf("*pi: %d\n", *pi);
8
9     return 0;
10}
11
12int *p = &i;
```

datatype
size &
length of
memory

functions
pointer

* p *p → []

bash

```
• @streetdogg → /workspaces/c-pointers-course (main) $ gcc main.c
• @streetdogg → /workspaces/c-pointers-course (main) $ ./a.out
pi: 0x7ffc180ab75c, &i: 0x7ffc180ab75c
○ @streetdogg → /workspaces/c-pointers-course (main) $ gcc main.c
main.c: In function 'main':
main.c:9:25: warning: dereferencing 'void *' pointer
  9 |         printf("*pi: %d\n", *pi);
    |             ^
main.c:9:25: error: invalid use of void expression
○ @streetdogg → /workspaces/c-pointers-course (main) $ 
```

● #define NULL (void*)0x0

A screenshot of the Visual Studio Code interface. The Explorer sidebar shows a project named 'C-POINTERS-COURSE' with files like '.devcontainer', '.gitignore', 'a.out', 'main.c', and 'README.md'. The main editor window displays 'main.c' with the following code:

```
1 #include <stdio.h>
2
3 #define NULL (void *)0x0
4
5 int main() {
6     printf("%p\n", NULL);
7 }
8
```

The line '#define NULL (void *)0x0' is highlighted with a green arrow pointing to it. The terminal window below shows the output of running 'gcc main.c' and then './a.out':

```
●@streetdogg ~ /workspaces/c-pointers-course (main) $ gcc main.c
main.c:3: warning: "NULL" redefined
  3 | #define NULL (void *)0x0
  |
In file included from /usr/include/stdio.h:33,
                 from main.c:1:
/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h:395: note: this is the location of the previous definition
  395 | #define NULL ((void *)0)
  |
●@streetdogg ~ /workspaces/c-pointers-course (main) $
```

● we can not access NULL pointer

A screenshot of the Visual Studio Code interface. The Explorer sidebar shows a project named 'C-POINTERS-COURSE' with files like '.devcontainer', '.gitignore', 'a.out', 'main.c', and 'README.md'. The main editor window displays 'main.c' with the following code:

```
1 #include <stdio.h>
2
3 int main() {
4     int *p = NULL;
5     // printf("%p, %llu\n", NULL, (long long unsigned int) NULL);
6     printf("%d\n", *p);
7 }
8
```

The line 'printf("%d\n", *p);' is highlighted with a green arrow pointing to it. A handwritten note 'NULL pointer' is written next to the variable 'p'. The terminal window below shows the output of running 'gcc main.c' and then './a.out':

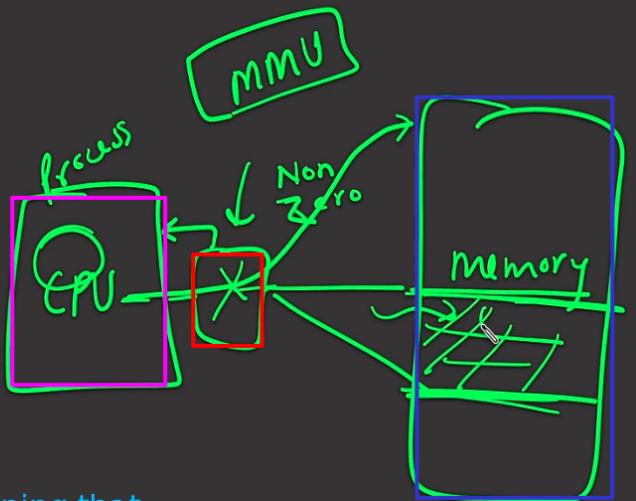
```
●@streetdogg ~ /workspaces/c-pointers-course (main) $ gcc main.c
●@streetdogg ~ /workspaces/c-pointers-course (main) $ ./a.out
(nil), 0x0
●@streetdogg ~ /workspaces/c-pointers-course (main) $ gcc main.c
●@streetdogg ~ /workspaces/c-pointers-course (main) $ ./a.out
(nil), 0x0
●@streetdogg ~ /workspaces/c-pointers-course (main) $ gcc main.c
●@streetdogg ~ /workspaces/c-pointers-course (main) $ ./a.out
(nil), 0x0
●@streetdogg ~ /workspaces/c-pointers-course (main) $ gcc main.c
●@streetdogg ~ /workspaces/c-pointers-course (main) $ ./a.out
Segmentation fault (core dumped)
●@streetdogg ~ /workspaces/c-pointers-course (main) $
```

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

it'll fail
on systems with
HLOS

Linux,
Mac,
Windows

fail

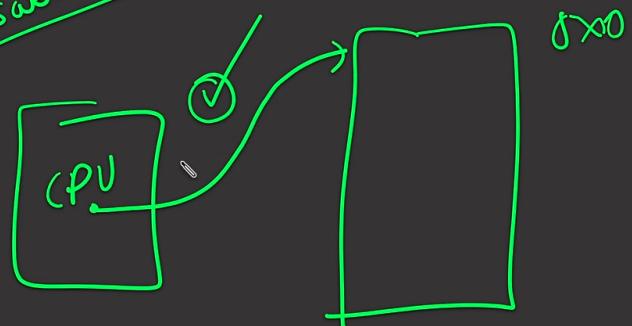


- when process has access to "0th" memory location then MMU will check that
- It will send an INT to CPU by mentioning that memory access is not permitted

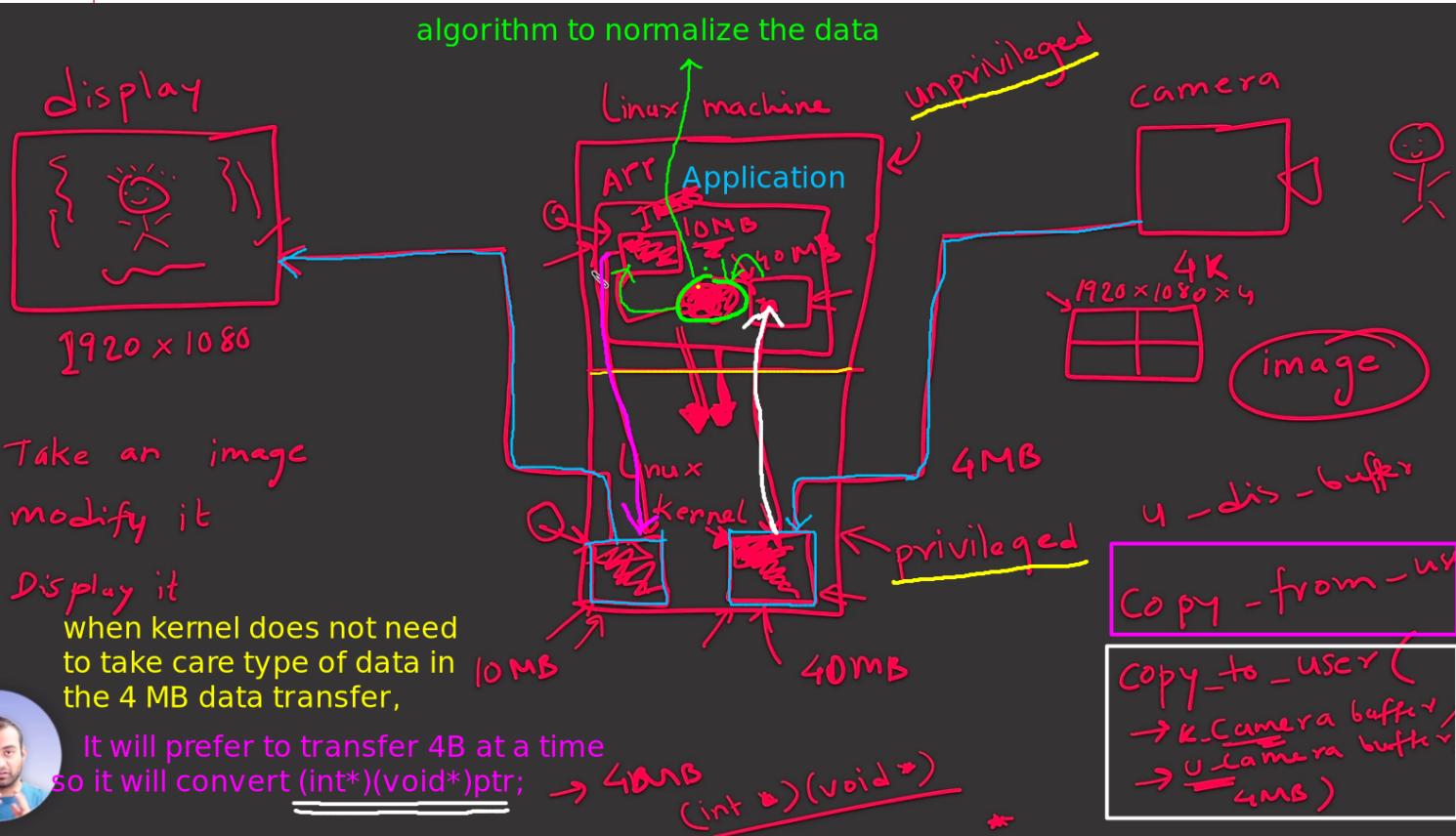


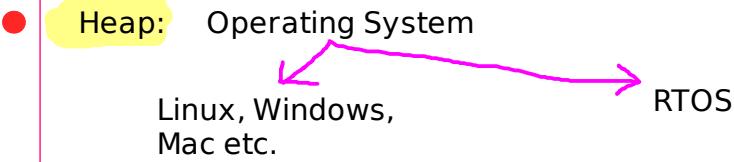
When you do not have
HLOS.

Bare metal
RTOS with memory management
Disabled



- Super use case example of Void Pointer



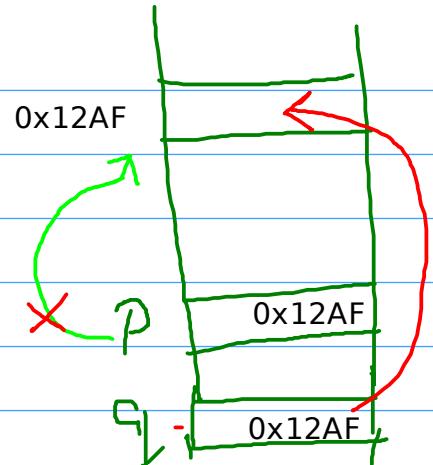


SYNOPSIS

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; 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

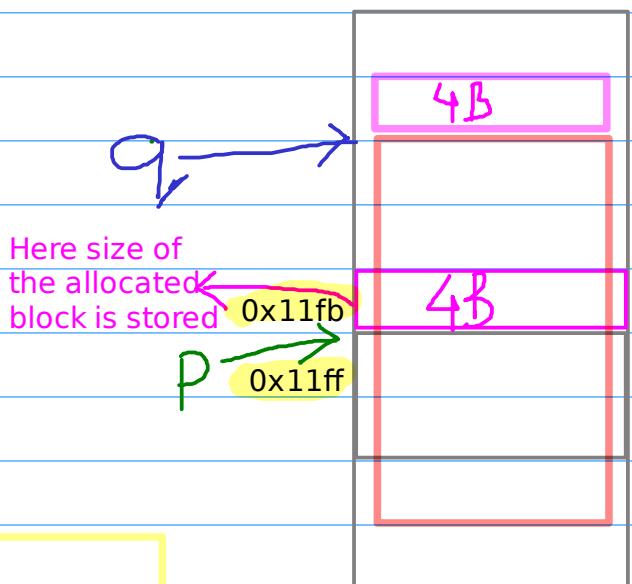
p=(int*)malloc(128);

free(p);

q=(int*)malloc(16*1024);

[now *(0x11fb)=16KB stored by some piece of 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

4 B 512 B

P= malloc(512)

