

* Auto Sar *

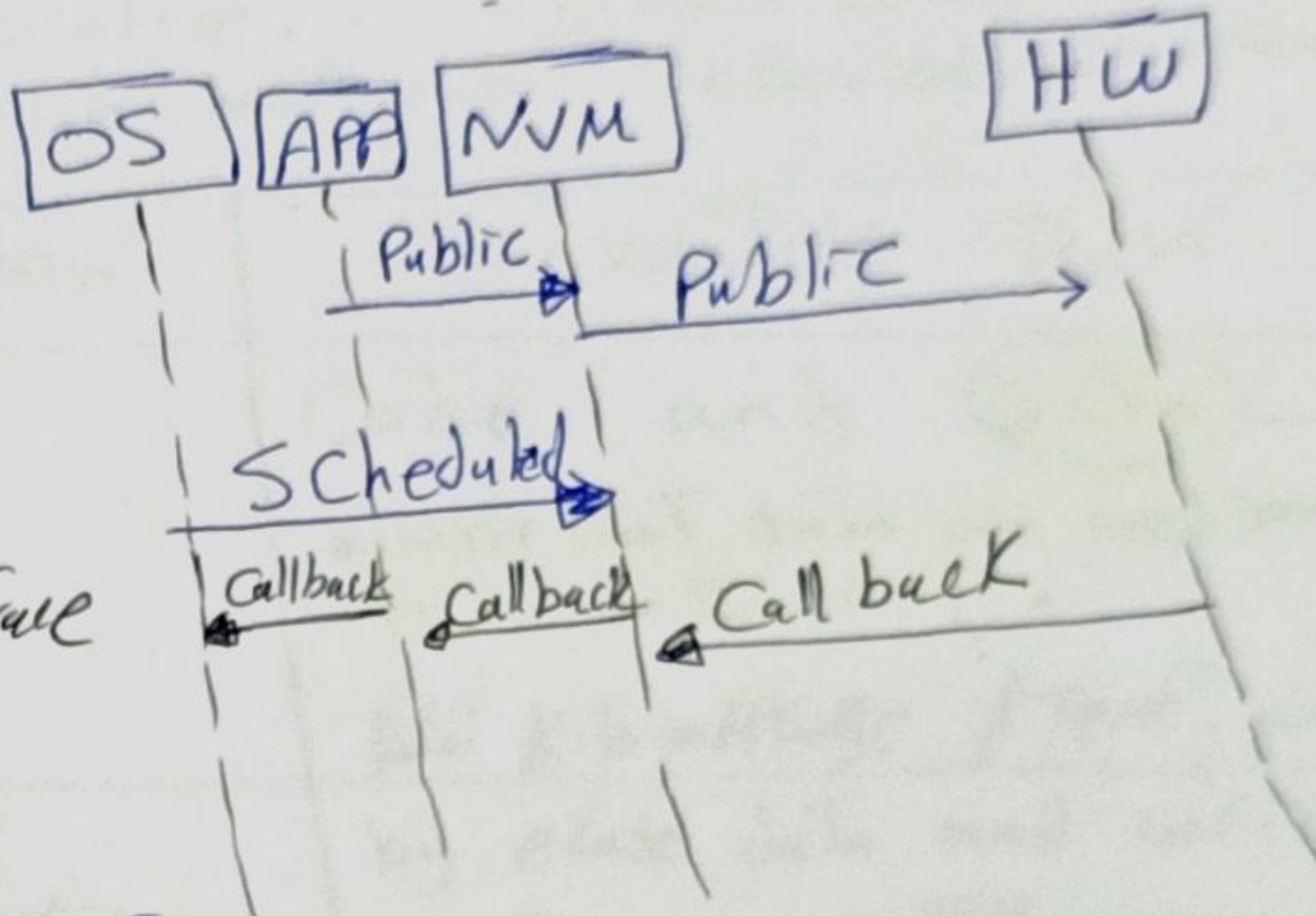
➔ Sequence diagram and interface's Types "Functions"

1) Public interface:-

Calling by upper-layer



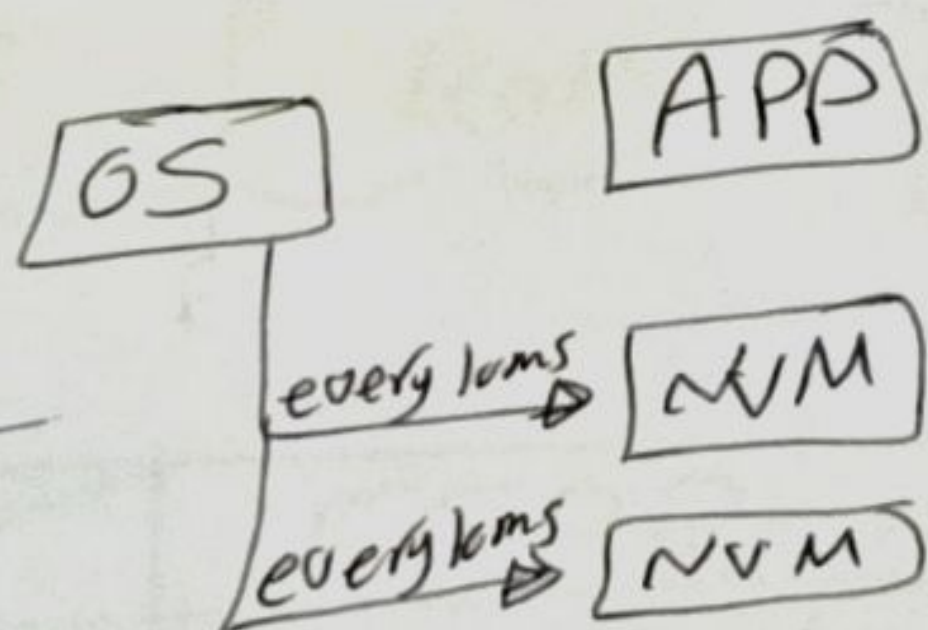
Ex:-
APP Will Calling any interface in NVM.



2) Scheduled interface:-

Calling by OS for Periodic Time To Call Function.

Ex:-

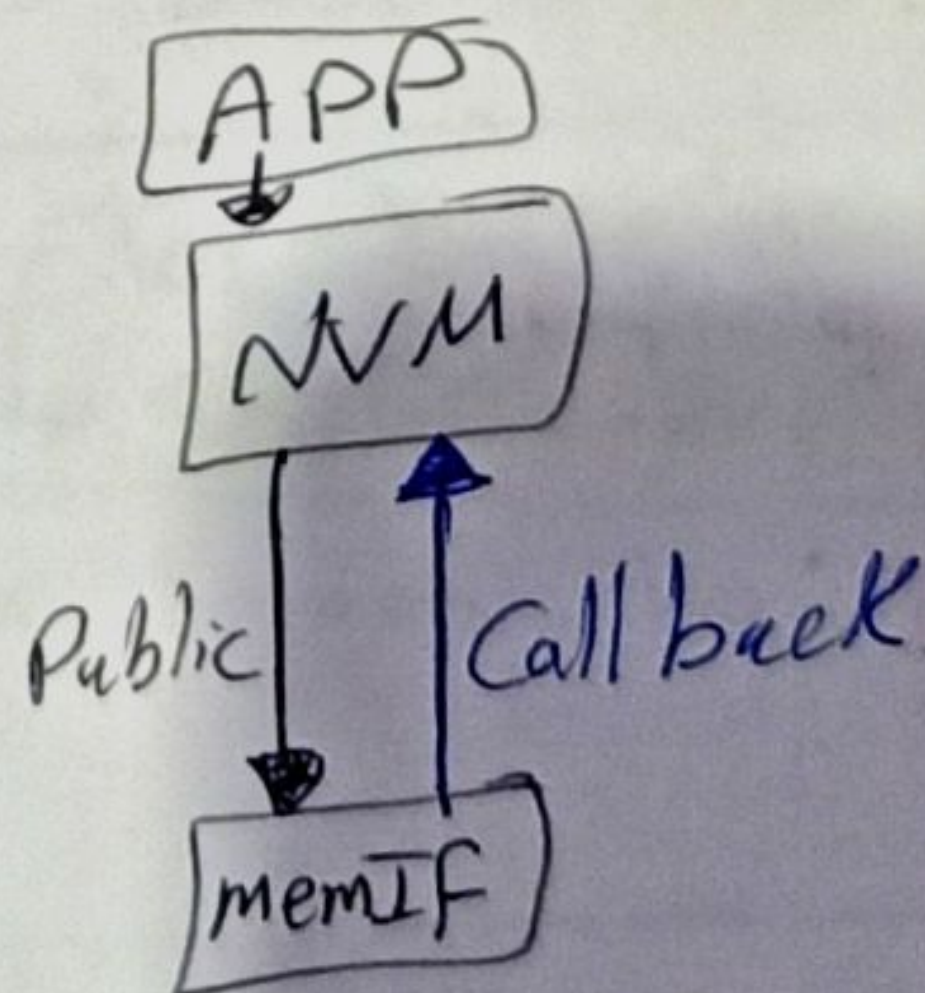


OS Can Call any interfaces in NVM every Periodic Time.

3) Call back interface:-

Calling by lower layers

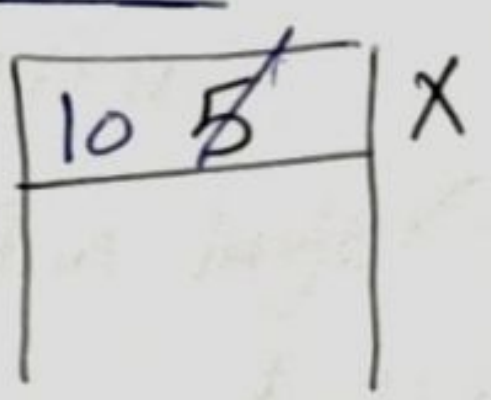
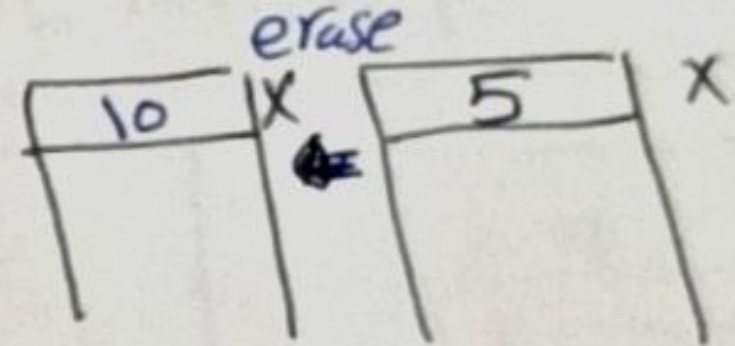
using in notification Function
To detect Function is done or not



* Memory Stack:-

→ before write about Memory Stack we need write about Memory.

* Types of memory:- ① Volatile. ② Non-Volatile.
 ↳ loss data when Power off ↳ Save data when Power off

	Volatile "RAM"	non-Volatile "ROM"
* Storage	Variables	Code and <u>data</u> ↳ important data we need save it when Power off. Ex: Kilometrage / Temp. airCondition
* update data	by <u>overwrite</u> x=5; x=10; 	by erase data and write again x=5; x=10; 

* Types of Rom:- ① EEPROM
 ↳ electrically erasable Programmable Rom
 ② Flash
 ↳ Type of EEPROM

	EEPROM	Flash
write & Read access	byte based access	Page base → Page = number of bytes
erase access	byte based erase	Sector base erase ↳ Sector = number of Page.
write erase maximum number "life"	From 100,000 To 1,000,000	From 10,000 To 100,000
COST	High COST	Low COST
Size	Large	Small
Power Consumption	High	Low

* Memory Stack:- after write about Types of memory
memory Stack care about "Non-Volatile memory"
and interested only data in Rom

→ we need store important data in Rom To need save him
and need store in specific locations To Read it any time.

→ Stack Content:-

1] NVM:- Non-Volatile manager
↳ Service layer

2] MemIF:- memory Interface
↳ ECU Abstraction layer

MemIF → IF To internal HW or
External HW

3] Fee:- flash eeprom emulation
↳ ECU ~~AL~~ AL

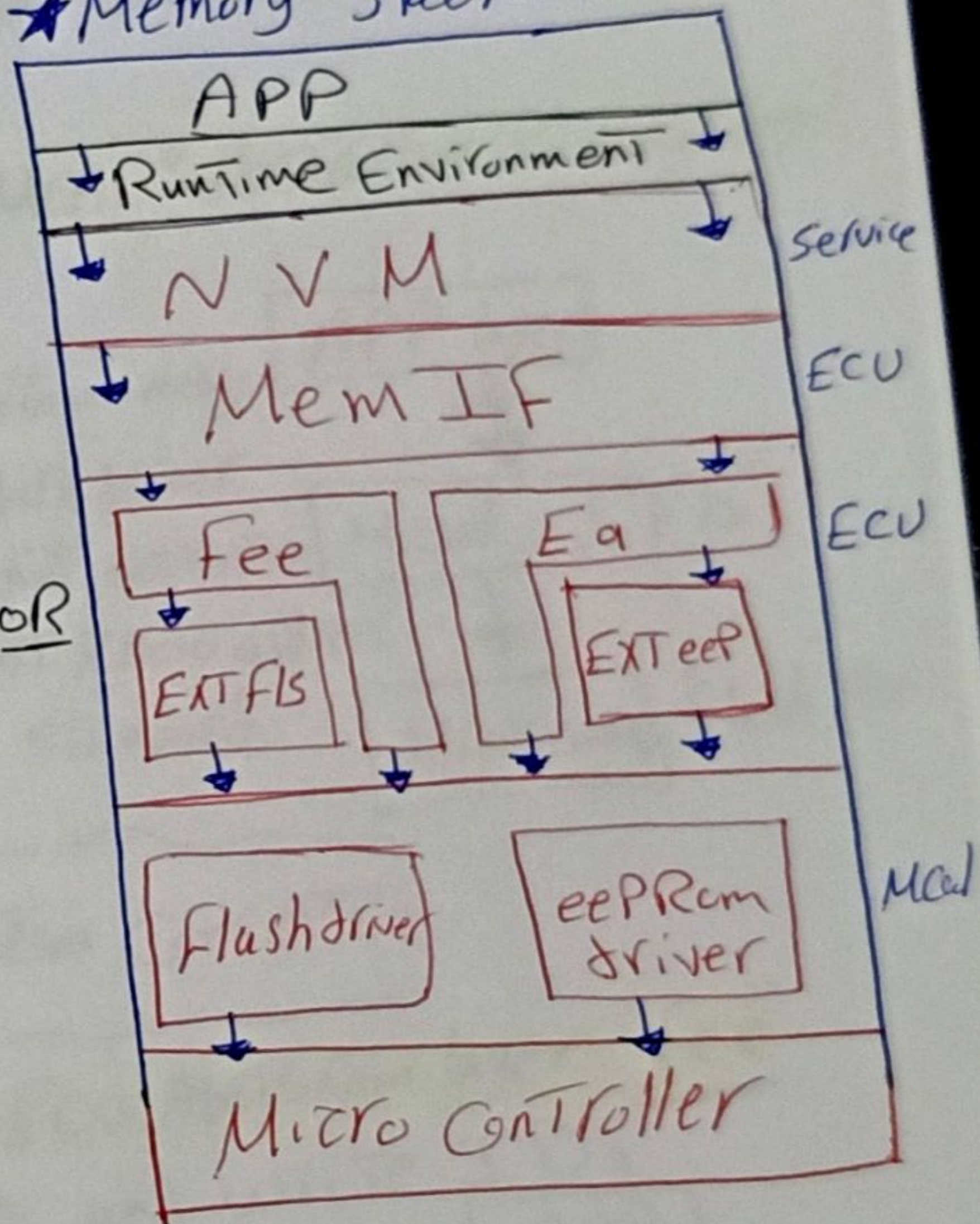
4] Ea:- eeprom abstraction
↳ ECU AL

5] EXT FIS:- External Flash
↳ ECU AL

6] EXT eep:- External eeprom
↳ ECU AL

7, 8] Flash & eeprom driver.
↳ MCAL AL

* Memory Stack*



* How To choose between external or internal Flash & eeprom

→ using Config Tool.

*before Autosar and abstraction Concept:-

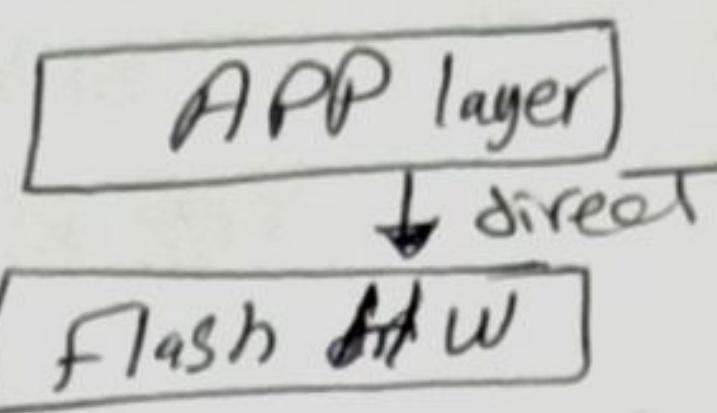
1) With no abstraction layer.

Ex:-

APP

$*(\text{u8}^*(0x30)) = \text{Value}$

→ write data on memory and must know addresses in APP
it is not standard



2) Add an abstraction layer → MCAL "abstraction Physical layer"

APP

FIS

FIS_write(Address, &data, length);

write data on memory using Flash driver but it's not abstraction because must know address it's not standard

APP layer

MCAL "FIS"

Microcontroller "Flash HW"

we need abstraction

*after Autosar and abstraction Concept:-

3) Add an abstraction layer → ECU Abstraction layer "fee"

we divide addresses To groups and write IDS

APP

fee

FIS

fee_write(ID, &data);

IDS	data	address
1	Temp	0x3B
2	Speed	0x5B

APP

fee

FIS

Microcontroller

* Target is more abstraction in layer ~~to not~~
 ↳ To more hidden Info from APP → More Time To SW dev for the APP algorithm.

* more Complexity → more abstraction

* Memory Stack → write data in NV.

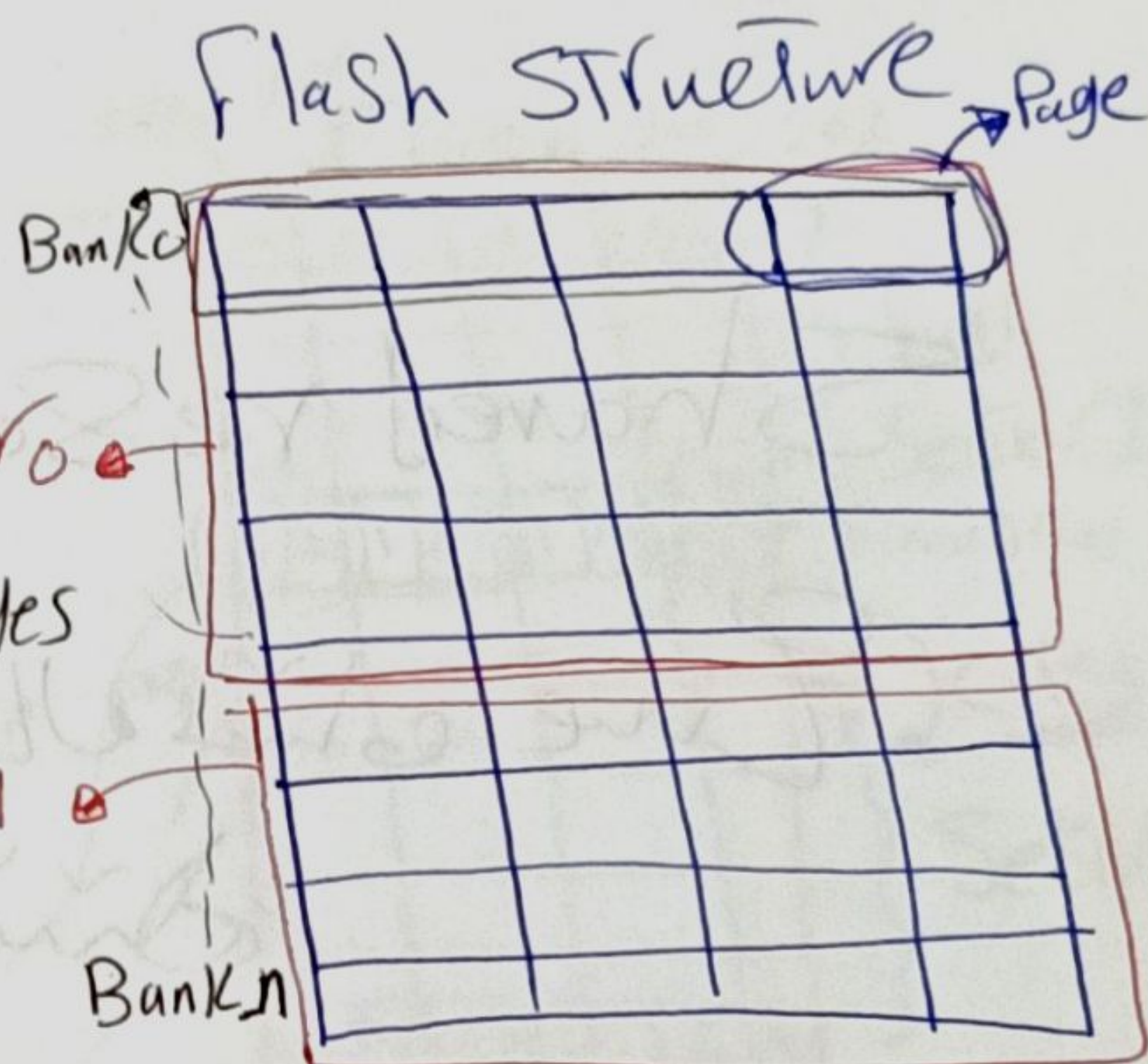
Flash Structure:

1) Byte :- no. of bits

2) Pages :- no. of bytes

3) Banks :- no. of Pages

4) Sectors :- no. of Banks or no. of Pages



Note:-

- write on Flash Access by Page
- Read from Flash access by Page
- erase by Sector

* Write Data in Flash & RAM

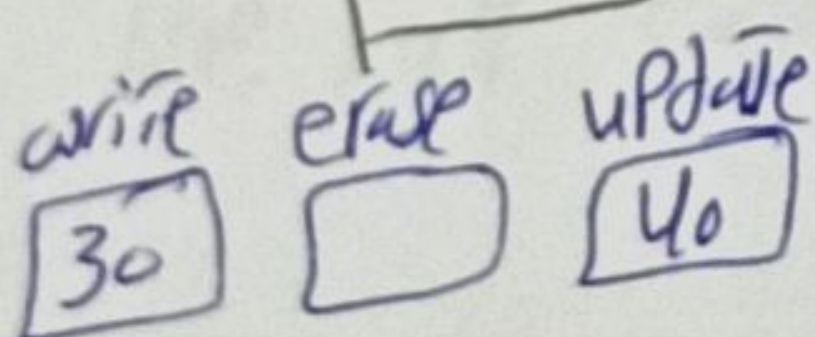
* Flash:-

write-fls(30);

write-fls(40); ~~X~~ not happen
must erase first wrong

Right

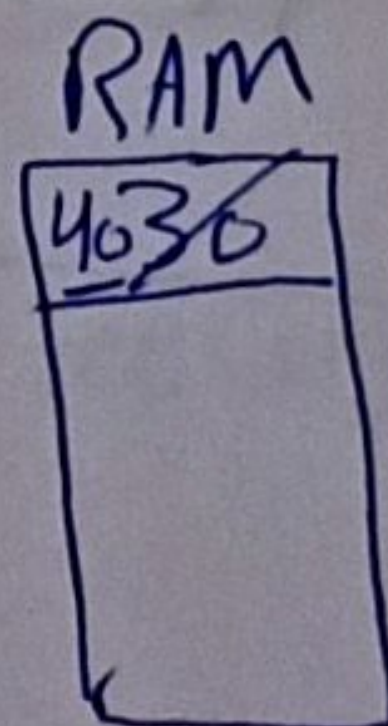
write-fls(30);
 erase-fls(30);
 write-fls(40);



* RAM:-

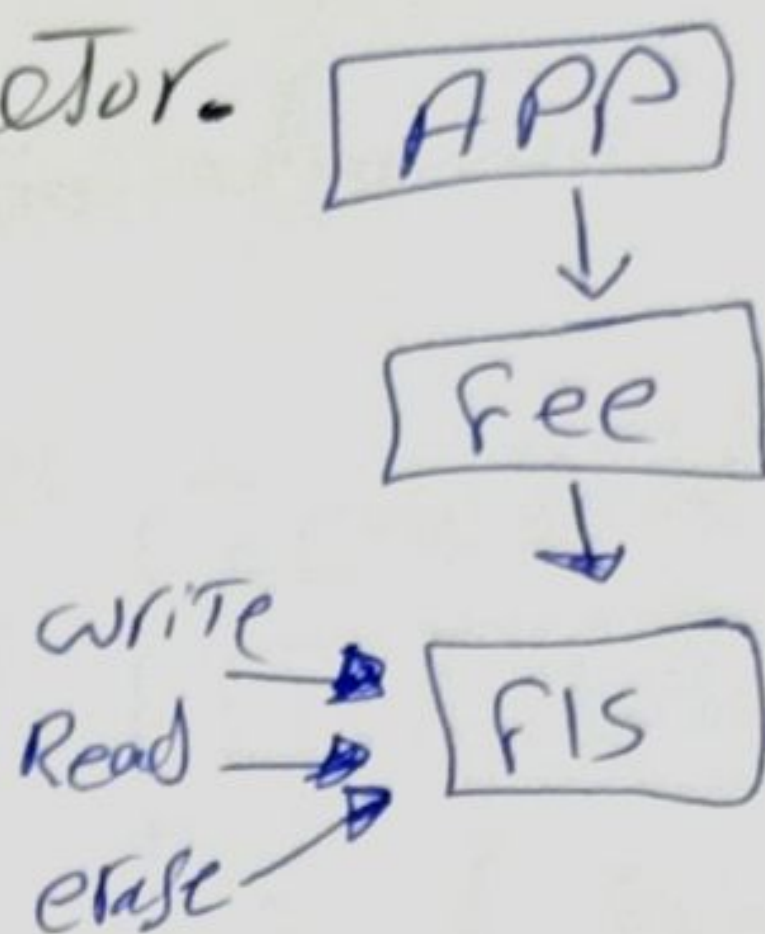
write-RAM(30);

~~write-RAM(40);~~
 write-RAM(40);



by overwrite in RAM location.

→ Fee :- Flash emulated eeprom
 ↳ Solve problem erase by sector.



* Fee Algorithm:- VIP

assume:- Page = 4 byte

→ write Data of Temp
 ↳ ID=1

Steps in Fee:-

1) Fee → write in first empty Page

2) Fee → update BMI

BMI:- Block management Info

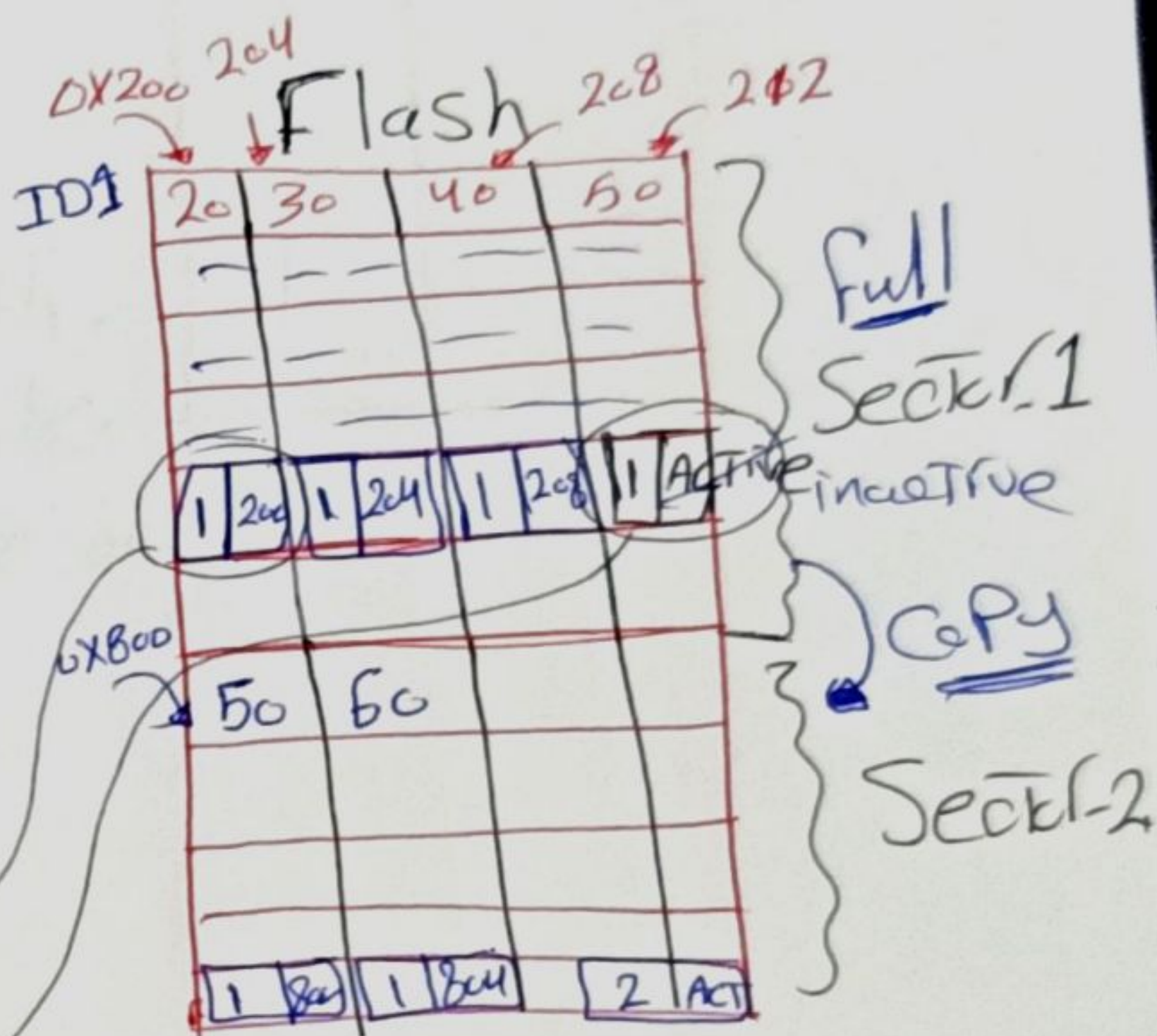
ID | Address

SMI:- Sector mang-Info

Sector number | State

Active → when write in Sector

Inactive → erase Sector



3) Fee → once The current Active Sector is Full, Fee Copy "Last Valid data" into The next Sector.

4) Fee → erase The full Sector and make it as Inactive.

* What if I need Read data?!

↳ using free_init "initialization function"

* free_init:- in start code what happen?!

1) Do the linear search for last valid data for each ID.

2) Create "Cache Table"

3) Use Cache Table for Data

Accessing → write/update → Flash.
→ Cache.

→ Read → Cache Table only.

Cache Table "RAM"

IDS	data	last data
1	50	↙

* Adv of free:

1) abstractions in layers.

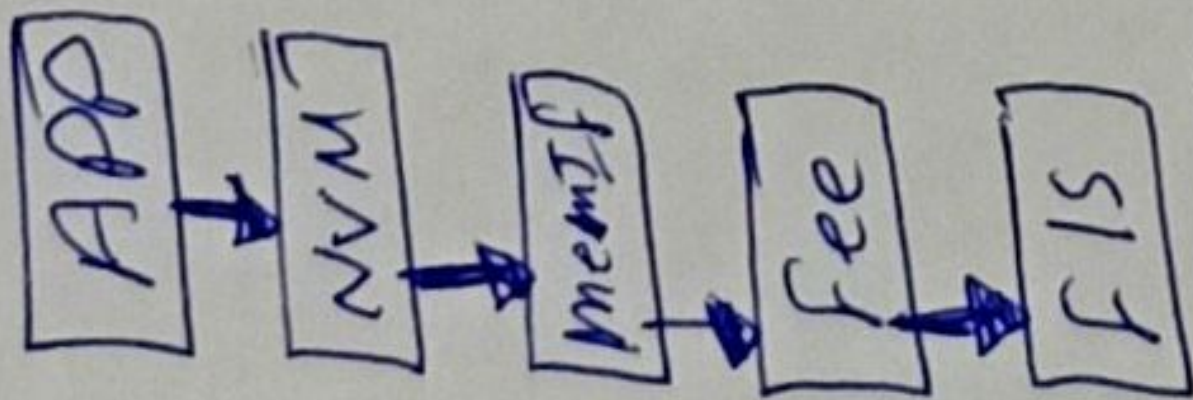
2) Solve Problem Flash accessing → write.
→ read.
→ erase.

3) extend of lifeTime Flash.

* Sequence diagram ! *

I want write data

Temp = 25, ID = 1



* Call back function using
as notification function.

* Check status -> Call back

* Check hw operation flag:-

- 1) with interrupt
 - 2) without interrupt
- "Call back direct"

