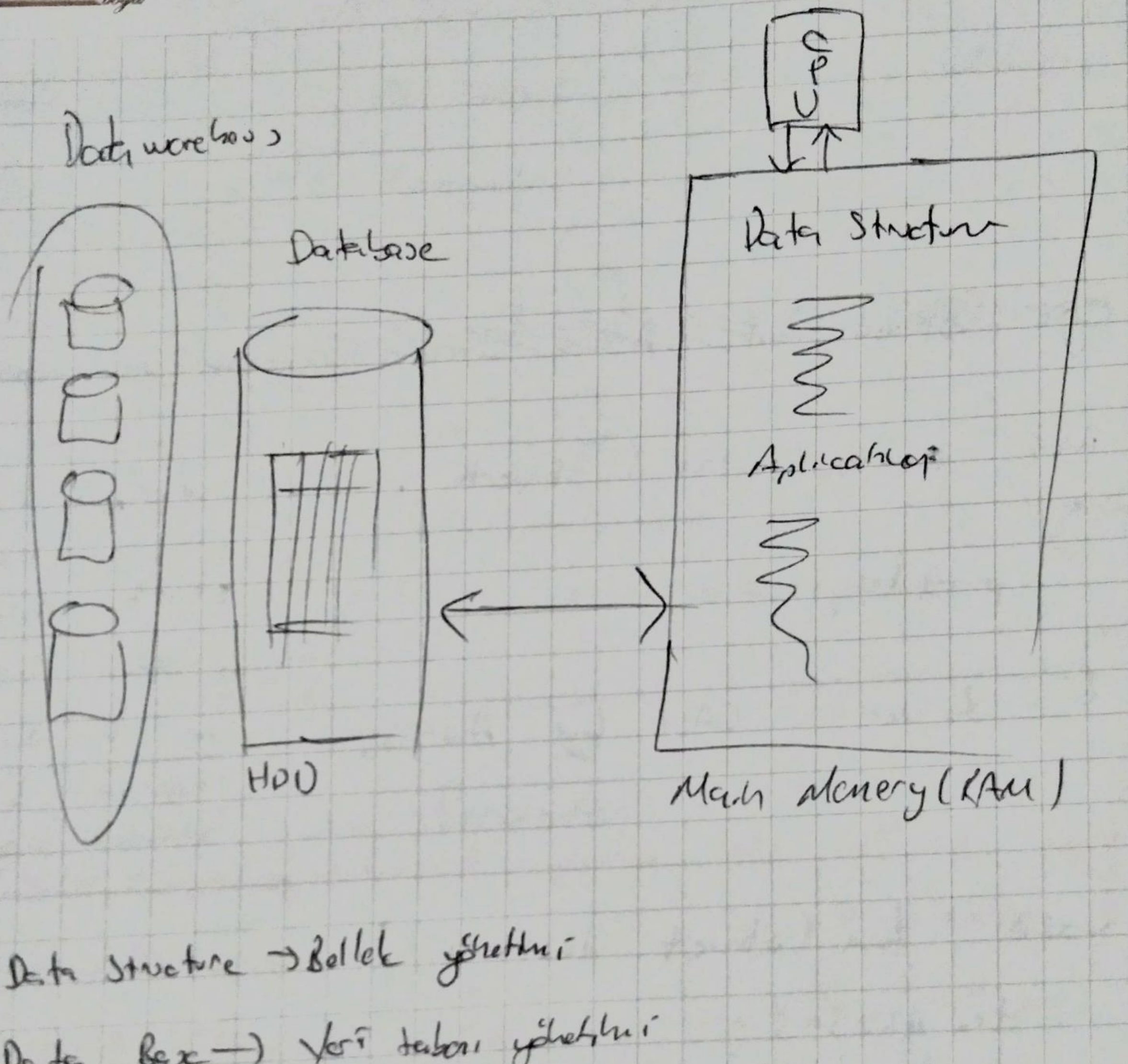
DATA STRUCTURE



Data Structure → Bellek yönetimi

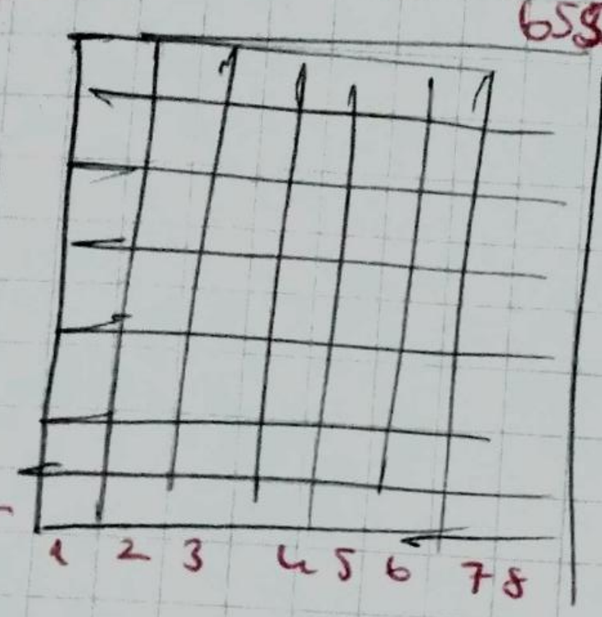
Data Base → Veri tabanı yönetimi

Data warehouse → Verilerin tutulduğu veya işlem gördüğü sistemler

Big Data → Büyük veriler üzerinde işlem yapma

Static vs Dynamic Memory Allocation

PERA

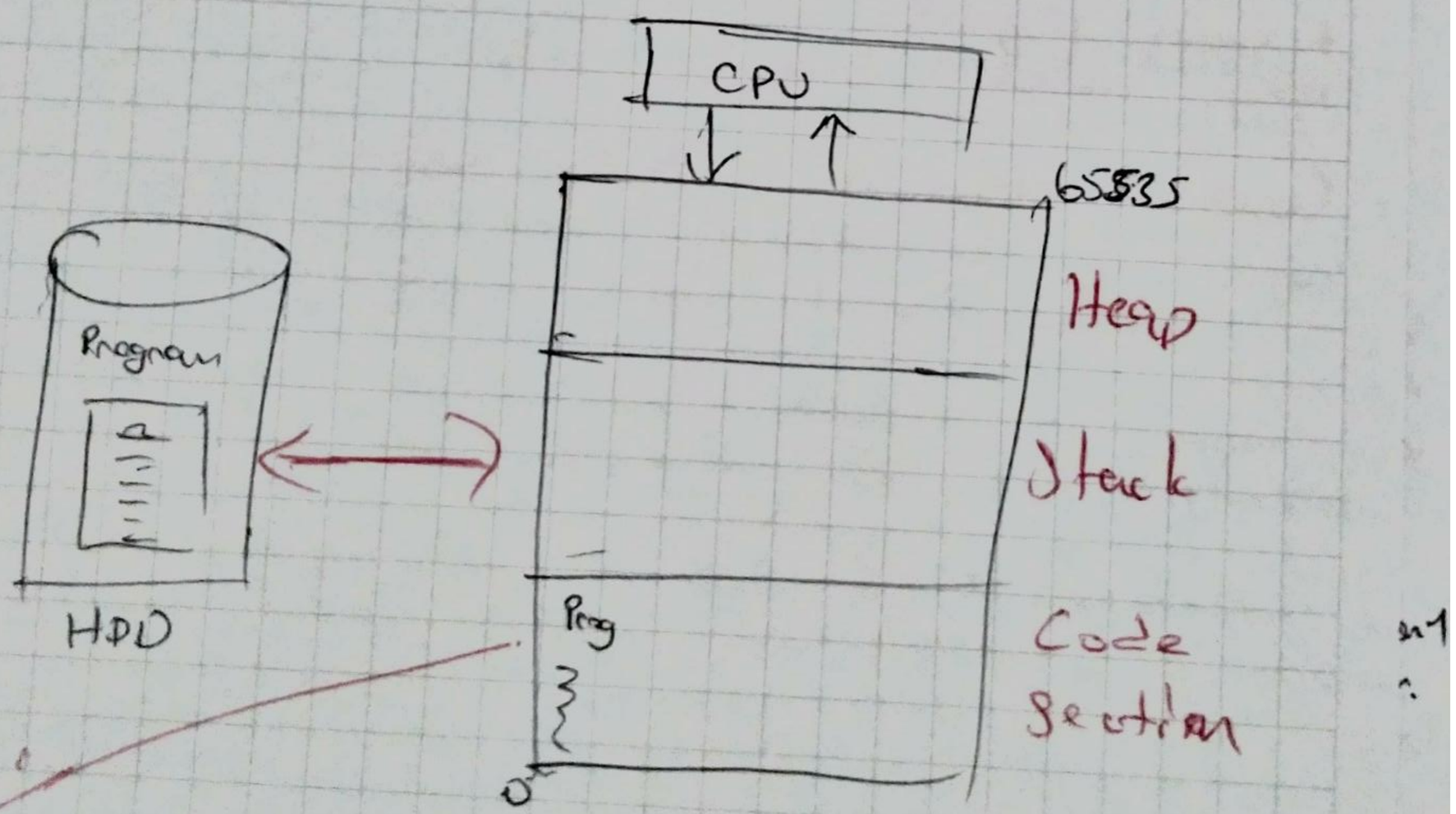


→ Bellek(Ram)

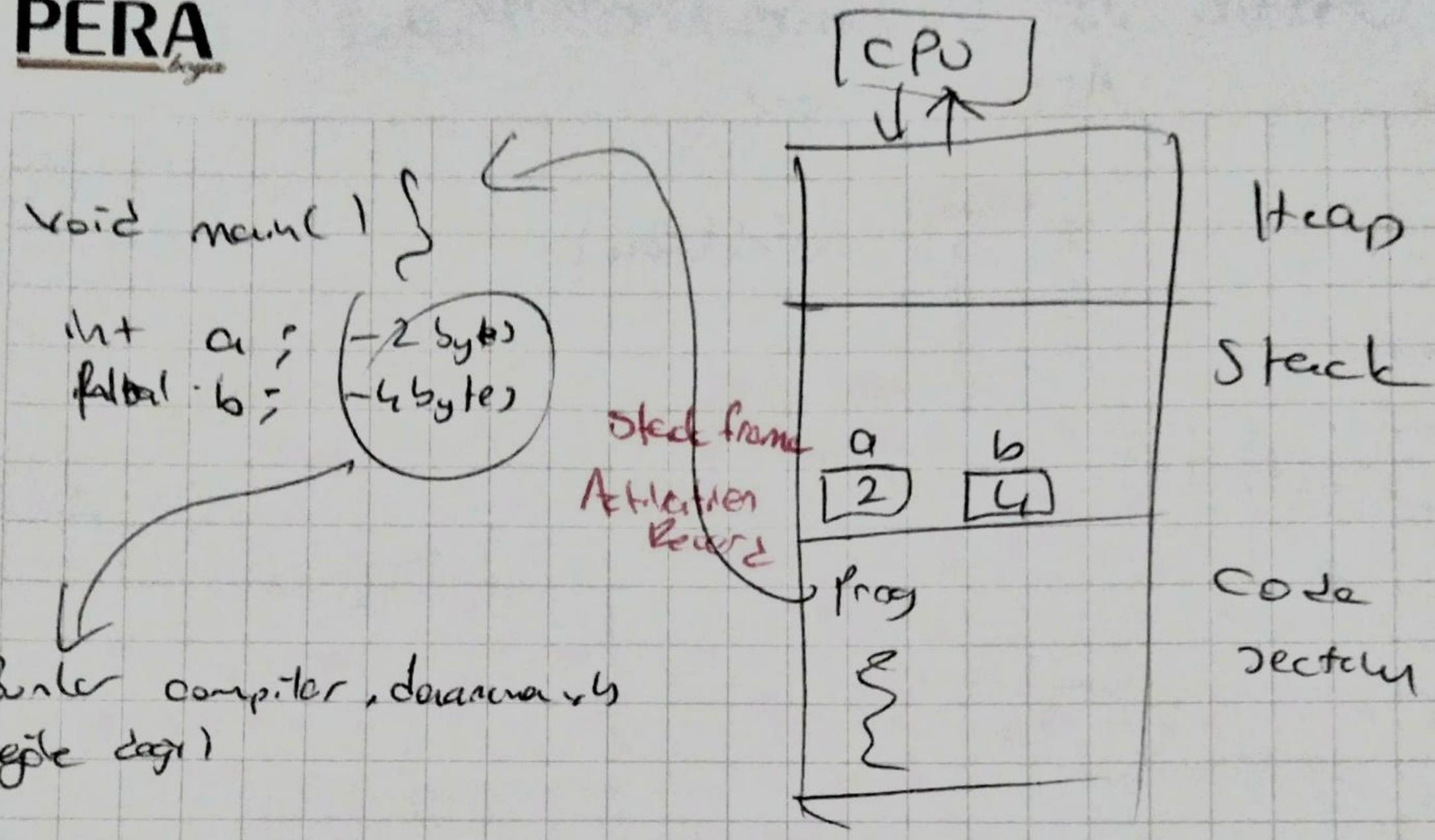
→ Bellek byte lara bölünür
→ Her byte bir adresi tutar

0 - 65535 = 65536 = 64 x 1024 = 65536 → 64KB

64 KB LIK KAM OLSUN



...ye yerlesir. daha sonra cpu program calistir.

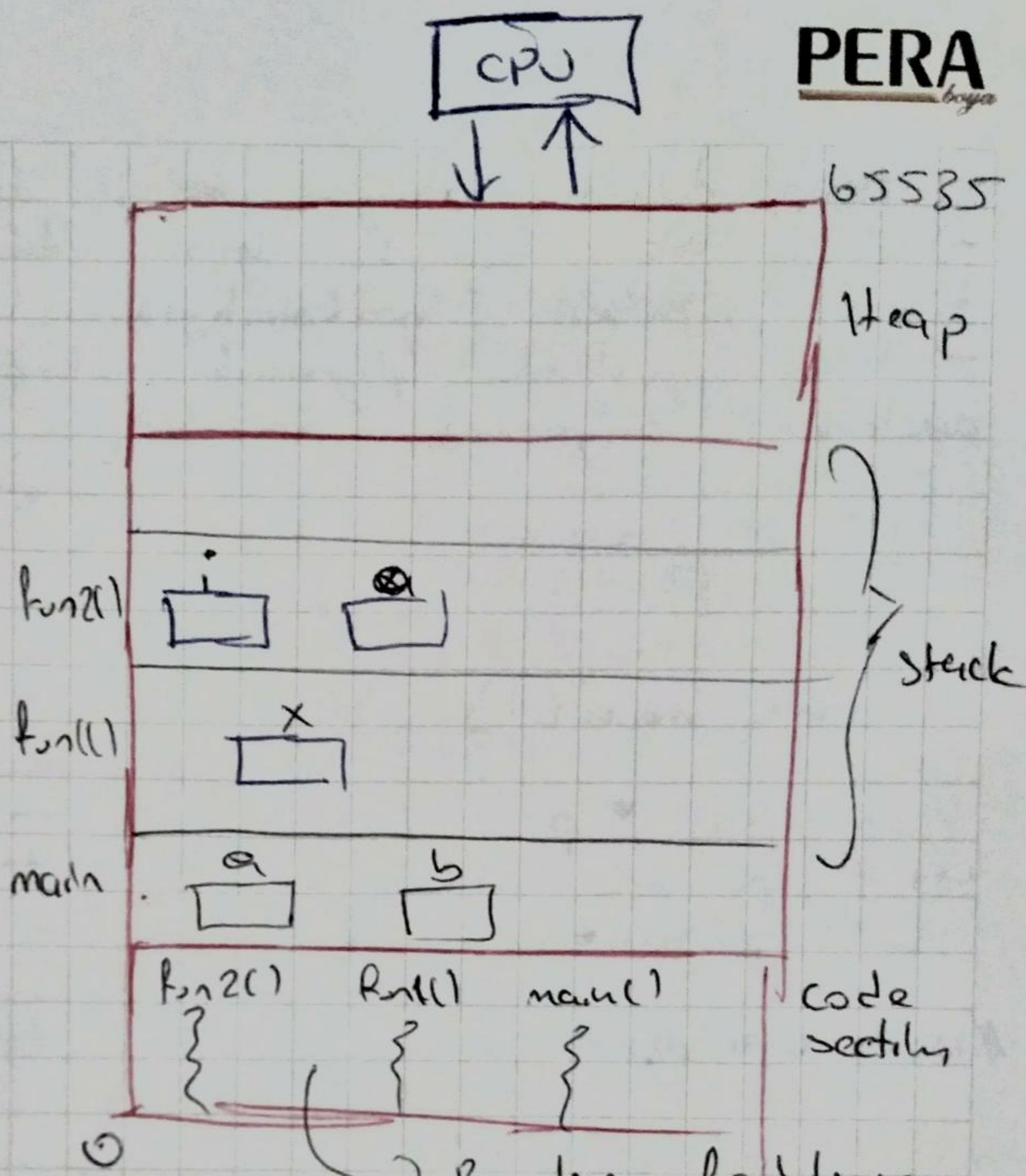


Stack code section'ın ihtiyacı duyduğu bellek alanı tutar ve kaydeder

Kısacası programın çalışabilmesi için ne kadar alana gerekli olduğunu tutar

Compile çalışmadan öğrenilir çağırır

```
void fun2(int i)
{
int q
}

void fun1(){
  int x;
  fun2(x);
}

int main(){
  int a;
  float b;
  fun1();;
}
```



→) ~~Stack~~ Bir program bellekte yer açma ve yer silme işlemi yapmaz bir kompiler yapar

→) Program derlendikten stackte ilk main ardından fun1 ve sonrada fun 2 duracaktır.

→ Program kapatıldıktan ilk önce fun 2 ardından fun1 en sonda main silinir ve program sonlandırılır.

Heap

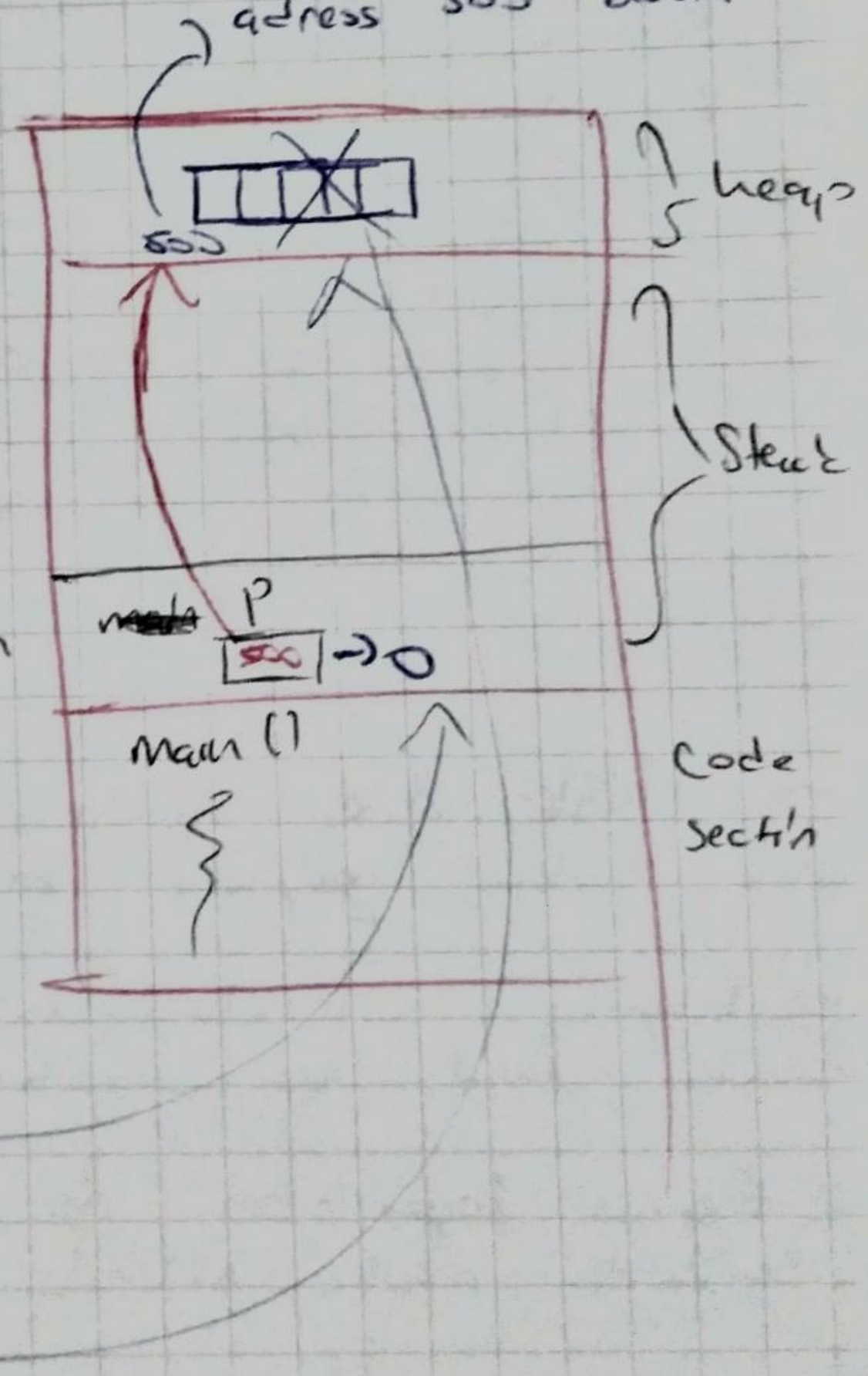
PERA boya

→ Heap düzenli dizayn edilmiş bellek alanıdır.
→ Heap bir kaynak gibi düşünülmeli ve kullanıldıktan sonra serbest bırakılmalıdır
→ Heaple dinamik programlar yazılabilir. Ancak pointerlar sayesinde erişilebilir

```
int main(){

    int *p
int  p= new int[5];
int* p= (int*) malloc(2*5);

// kullanıldıktan sonra silinir

delete[] p;
p = NULL;
```

Burada heapte ayırdığımız alanı sildik

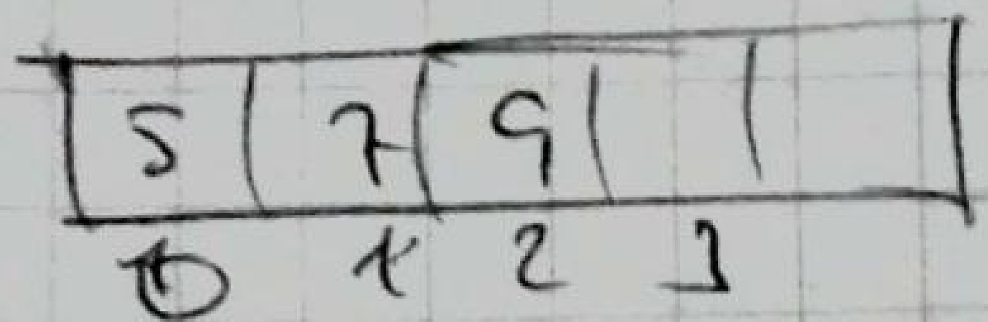
burada da pointerı hiç bir şeye göstermesin dedik



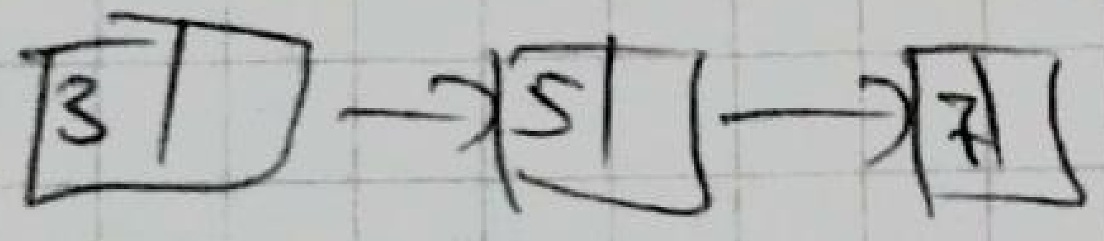
Physical vs Logical Data Structures

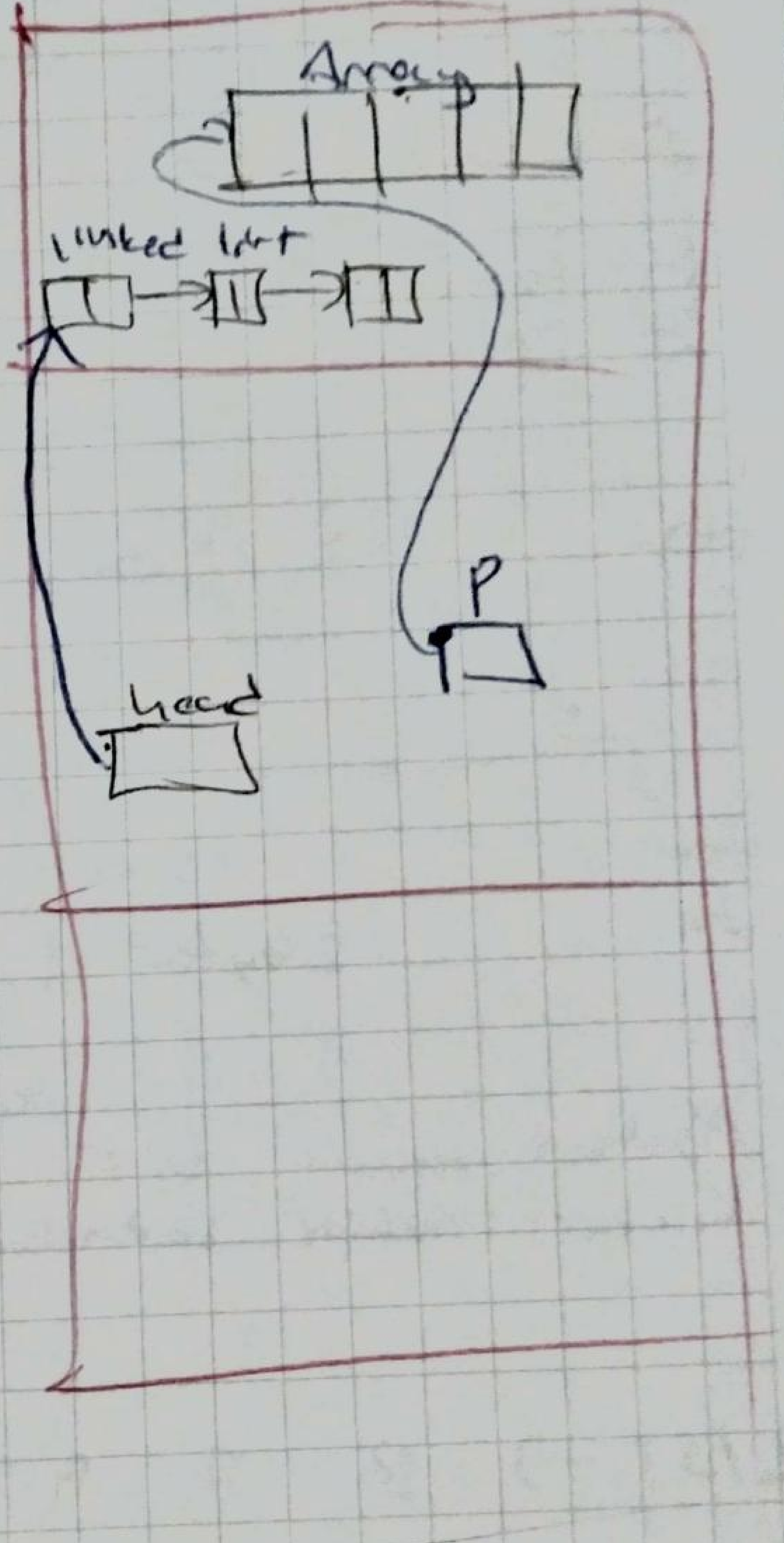
→ Physical

1. Array



2- LinkedList





→ Bellekte ne kadar yer ayarlayacağına karar verir

→ Array stack veya Heapte oluşturulabilir

→ Linked list sadece Heapte oluşturulur

→ Physical daha çok bellekle alakalı

→ Logical

- Linear [
  - 1- Stack → LIFO
  - 2- Queues → FIFO
- Non-linear [
  - 3- Trees } Hiyerarşi
  - 4- Graph
- Tabular [
  - 5- Hash Table

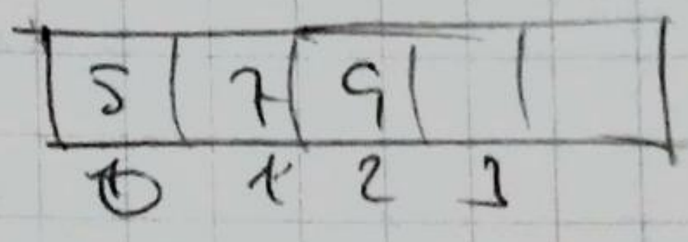
Anladığım kadar Logical Datalar Physical Datalarla işlem yapar özelliği ile sıralama silme gibi algoritmik işlemler alanıdır.

Physical vs Logical Data Structures

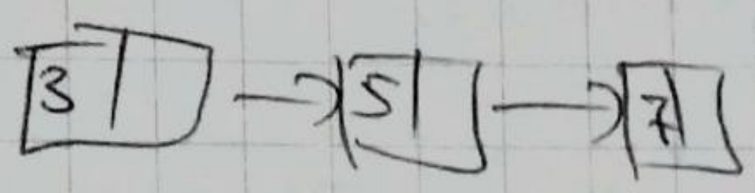
PERA

→ Physical

1. Array



2. Linked List



→ Bellekte ne kadar yer ayarlayacağına bakar karar

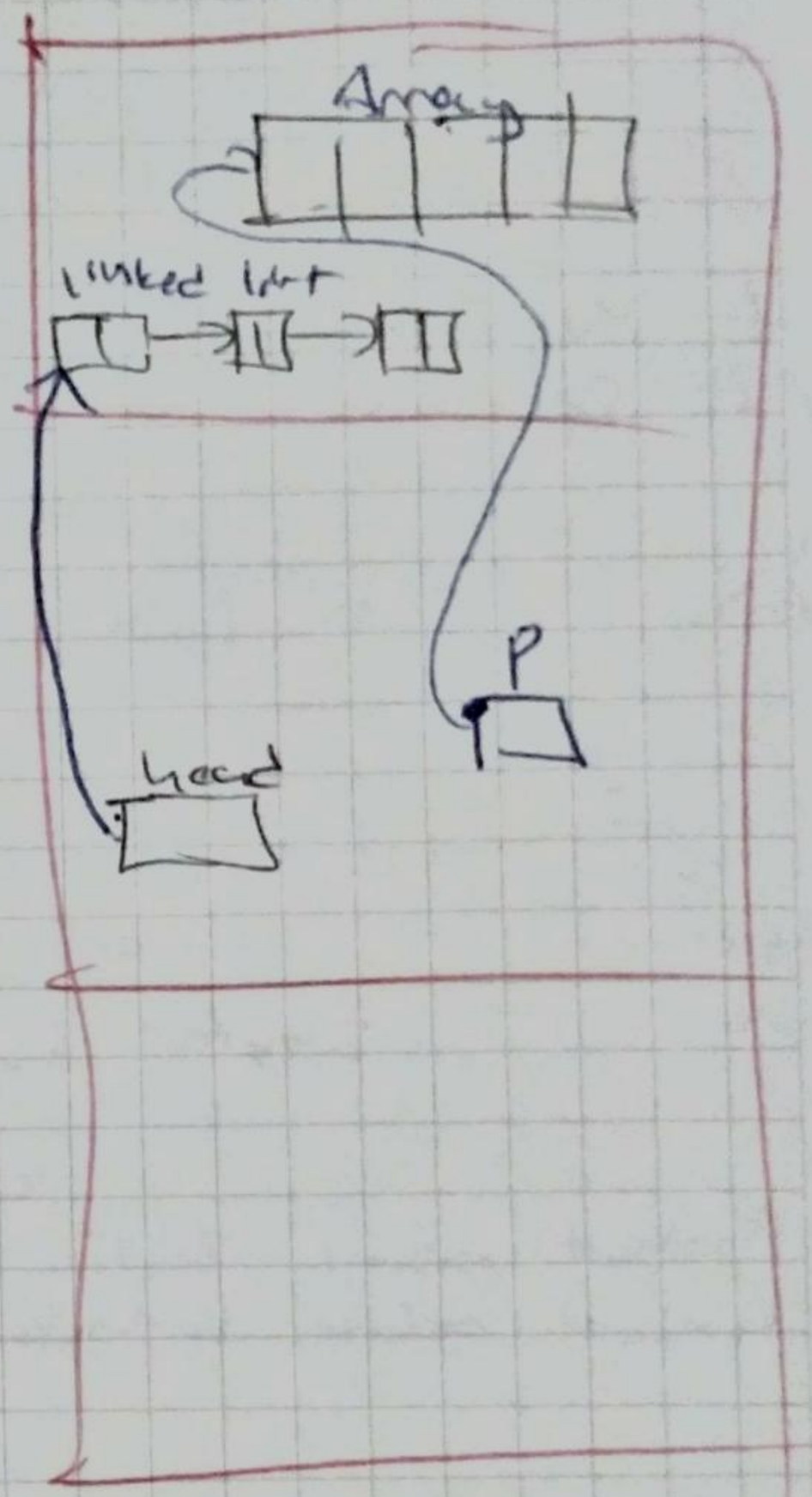
→ Array stack veya Heapte oluşturulabilir

→ Linked list sadece Heapte oluşturulur

→ Physical daha çok bellekle alakalı

→ Logical

- Linear
  - 1- Stack → LIFO
  - 2- Queue → FIFO
- non-linear
  - 3- Trees } Hiyerarşi
  - 4- Graph }
- Tables
  - 5- Hash Table



Anladığım kadar

Logical Datalar Physical Datalardan işlem yapare Ekleme ise Sıralama silme gibi algoritmik işlemler olmuştur.

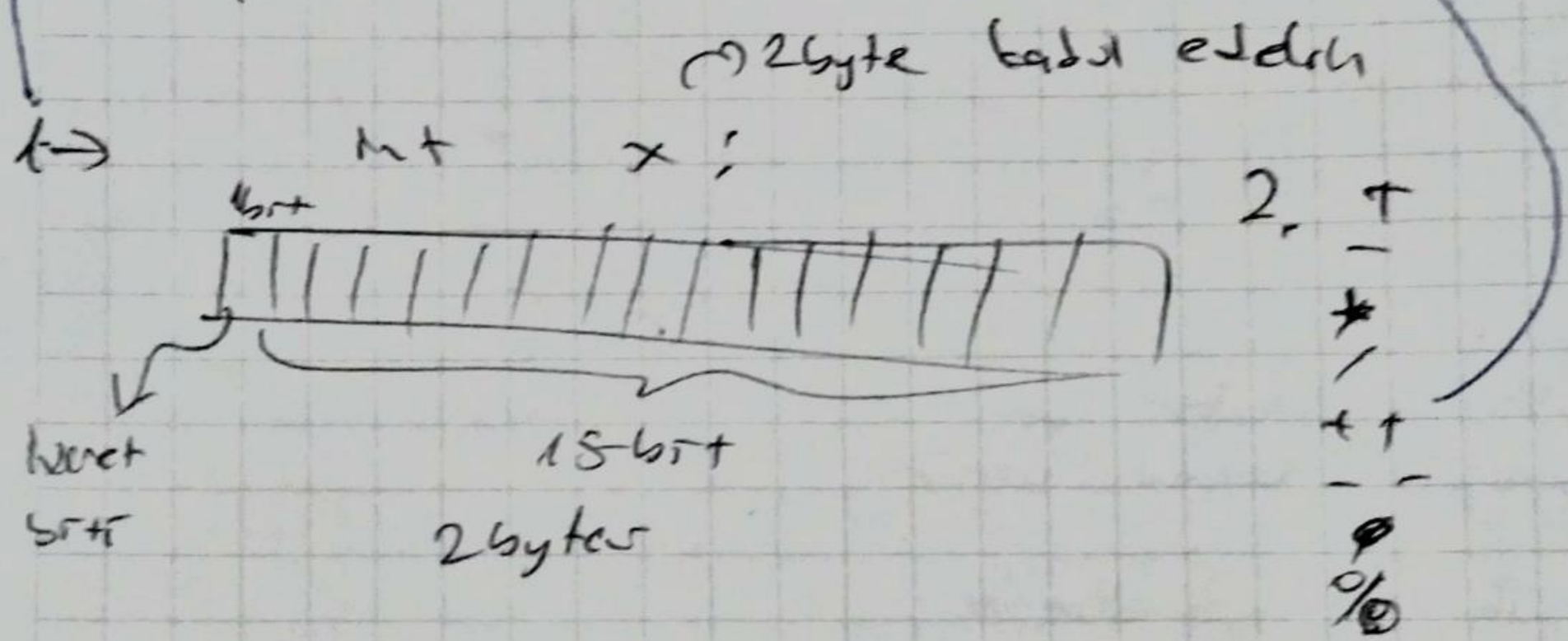
PERA

ADT (Abstract DATA TYPE)

1- What is ADT
2- CIST ADT

Abstract Data type

1- Representation of Data
2- Operation on Data



Abstract anlamı bilgi detaylardan konur
kullanıcı nasıl kodlardan kurar (anlaşılan kadar)

LIST → 3 7 9 12 15 17 21
0 1 2 3 4 5 6

Nasıl bellekte ifade eder
Data ?

1- space for ordering element
2- capacity
3- size

Bu 3 kural ile ifade edilir

Operations:

add()
remove()
search()

Bunda operasyonlar evrilerek yapılacak

PERA

List	8	3	9	4	6	10	12
	0	1	2	3	4	5	6

add (element) (append (ele))
↳ sona ekleme

add (element, index) (insert (index ek))
↳ ortaya ekleme veya basma

remove (index)

set (index, ek) (Replace (ind x, ele)
↳ indexteki datayı değiştirme

get (index)
↳ indexteki elemanı söyle

search (key) (contain (key)
↳ bir element i aratkes

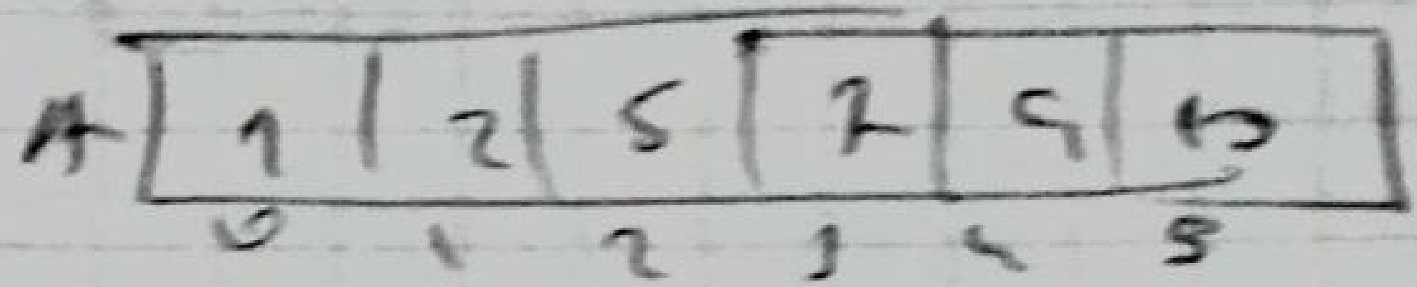
sort()

PERA

Time and Space Complexity

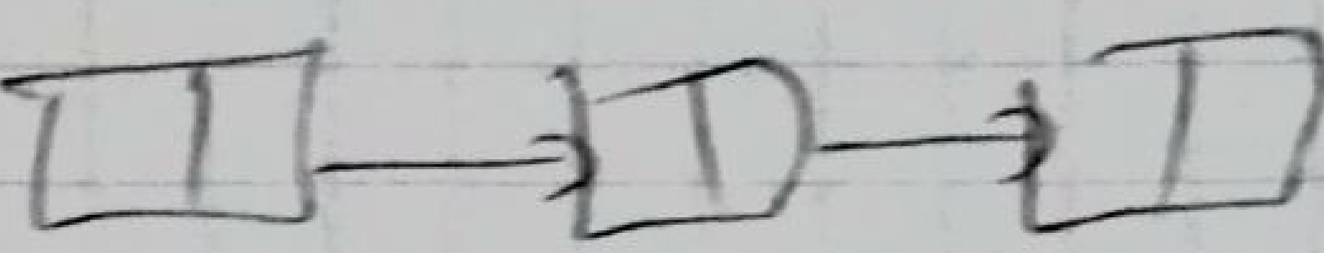
O(n) notasyonu ile gösterilir

bir dizi n tane eleman içerir ve her bir elemanı gezmek için




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

}
```

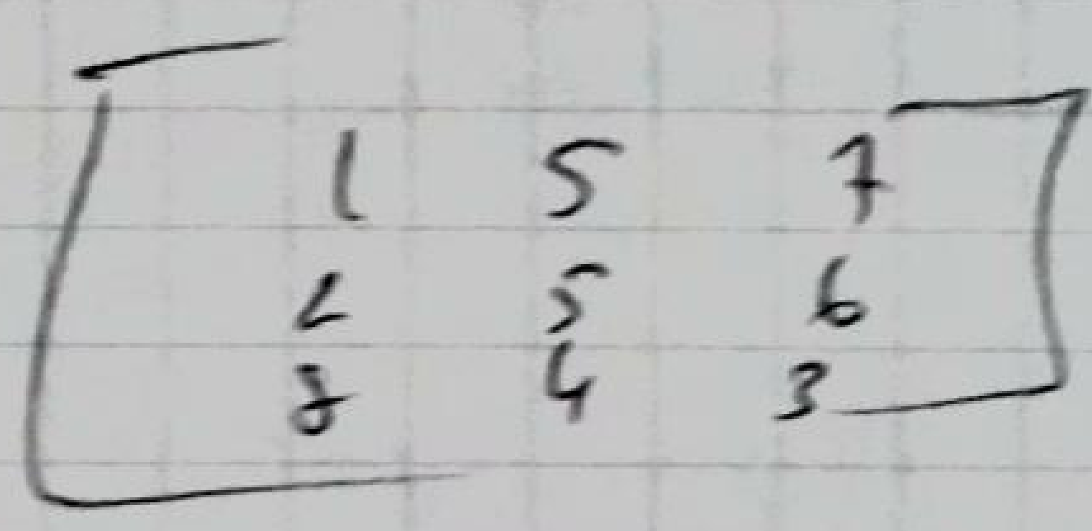
O(n) dur



bir dizi bir yerde karşılaştırılır

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

}
```

$\frac{n \cdot (n-1)}{2}$ $O(n^2)$



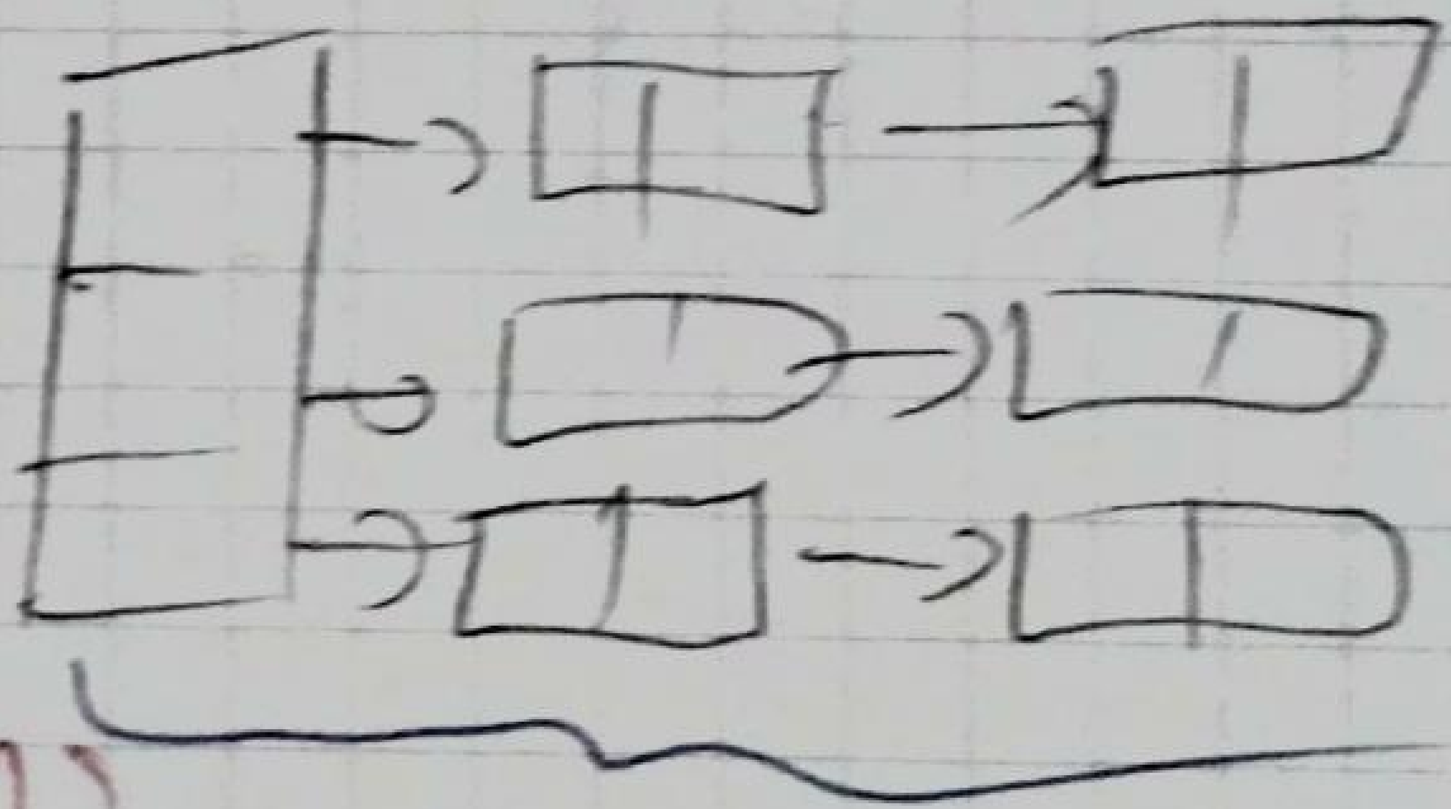
matrisi gösteren satır ver

$O(n^2)$

bir dizide aralığı daraltır ~~[illegible]~~

```
for (int i=0; i>1; i/2){

}
```

$O(\log_2 n)$



Dizi ile bir linked list karşılaştırılır.

bir dizi için birden fazla döngü olabilir

```
for (int i=0; i<n; i++){
    for (int j=0; j<n; j++){
    }
}
```

$O(n^2)$

* Not kodun karmaşıklığına bakmak, ~~faktör~~, döngülerin ~~aralarına~~ kaçıllarına bakarak karar ver

~~int~~

void swapp (x,y) {

int t;

t=x; ⟶ 1

x=y; ⟶ 1

y=t; ⟶ 1

---

f(n)=3n^0 ⟶ 1

Θ(1)

Θ(n^0) = Θ(1)

void Add (int n) {

int i, j;

for (i=0; i<n, i++) ⟶ n+1

for (j=0, j<n, j++) ⟶ (n+1).n

c[i][j] = A[i][j] + B[i][j]; ⟶ n^2

---

f(n) 2n^2+2n+1

Θ(n^2)

}

int sum(int A[], int n) {

int s, i;

s=0 ⟶ 1

for (i=0, i<n, i++) ⟶ n+1

s = s + A[i]; ⟶ 1

return s

f(n) 2n+3

Θ(n)

}

PERA

func1();

func2(); ⟵

func2() {

for (i=0; i<n; i++) {

}

}

Scanned by TapScanner