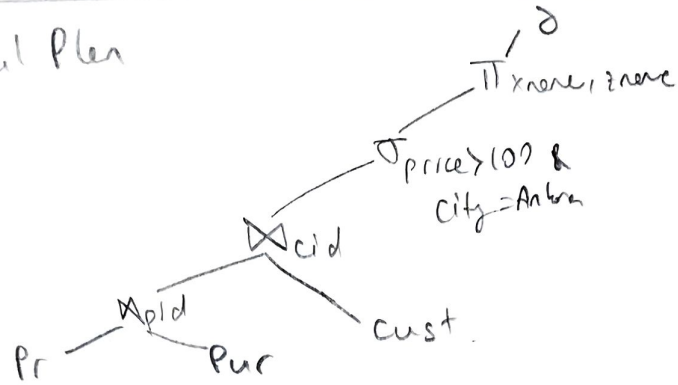
week 11

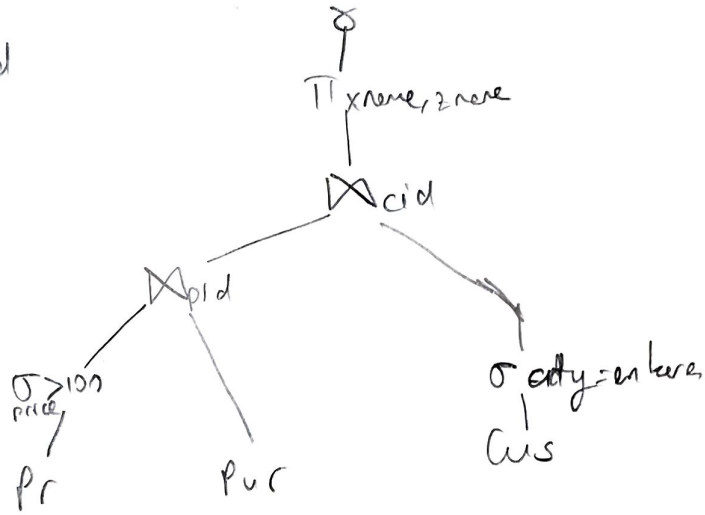
Query

S. D. xname, zname
F Pr x, Pur y, Cus z
W xpid = ypid & ycid = zcid
& xprice > 100 & zcity = Ankara

1st Logical Plan



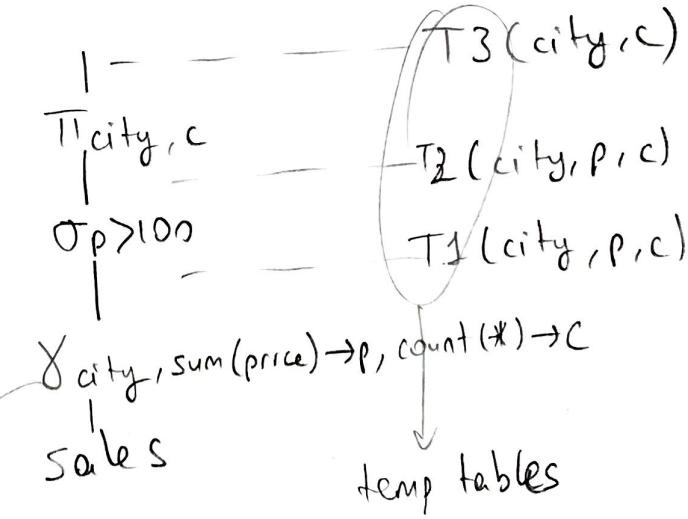
2nd, optimized



---

ex2:

S city, count(*)
F Sales
Group By city
Having Sum(price) > 100



ex: Supplier (sno, sname, scity, district)
Part (pno, pname, psize, pcolor)
Supply (sno, pno, price)

```
C.V. NearbySupp as
S   sno, sname
F   Sup
W   city = ankara and
    district = Cankaya
```

→

```
        |
    Π sno, sname
        |
    σ city = ankara & district = cankaya
        |
    Supplier
```

Suppose dbms gets:

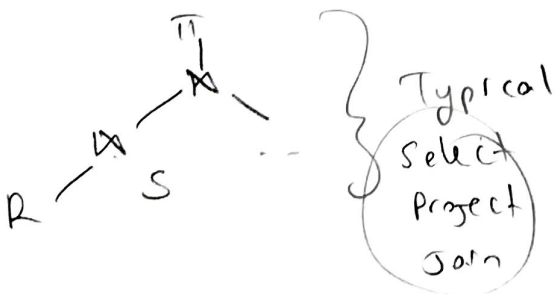
```
S   sname
F   NearbySup
W   sno in ( S   sno
             F   Supplies
             W   pno = 2 )
```

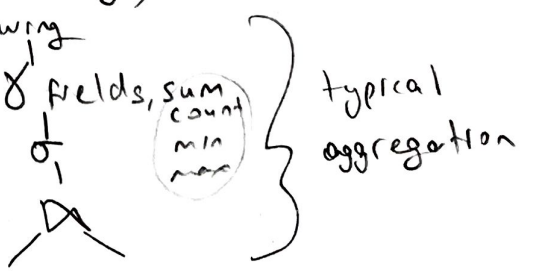
rewritten ⟹

```
S   sname
F   Suplier S, Supplier U
W   S.city = Ankara &
    S.district = Cankaya &
    S.sno = U.no &
    U.pno = 2
```

Find logical plan, generate physical plan.
- → B+ tree
- → File scan
- which algorithm is used (will come to this later)

We need to work on 1 query (no nesting)





ex: S Q.sno
F Supplier Q
W Q.dis = Gachibowli
and not exist (Se. *
F Supply P
W Psno = Q.sno
P.price > 100)

→ Correlation

⇓

DBMS works block by block. Decorrelate (NOT IN)
If we dont write, DBMS will write →

⇓ decorrelation

S Q.sno
F Supplier Q
W Q.dist = G.
Q.sno not in (S Psno
F Supply P
W P.price > 100)

still nested block

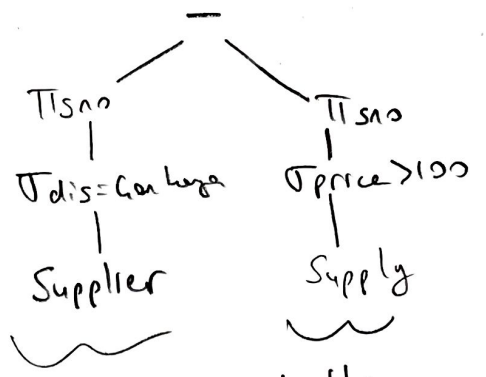
⇓ unnest (using EXCEPT = set difference)

S Q.sno
F Supplier Q
W Q.d = G.

EXCEPT

S P.sno
F Supply P
W P.price > 100

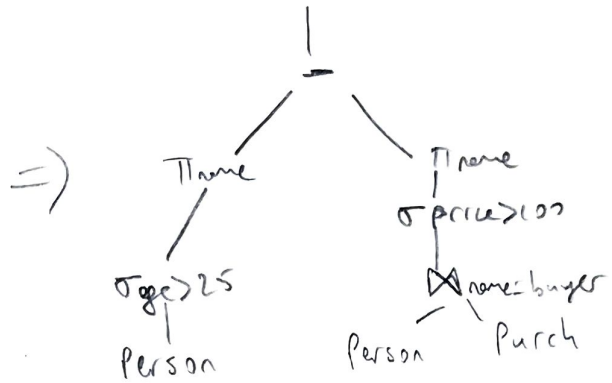
⇒



independently accessed / executed ✓

ex:

S Q.name
F Person Q
W Q.age > 25
and not exist
(S *
F Purchase P
W P.buyer = Q.name
and P.price > 100)



For each SQL query → many logical plans

For each logical plan → many physical plans.

Fast physical plan?

DBMs will generate physical plans w.o. executing. It guesses the cost.
Generating plans are costly. But spending time on a query generation is better than executing the wrong query.

---

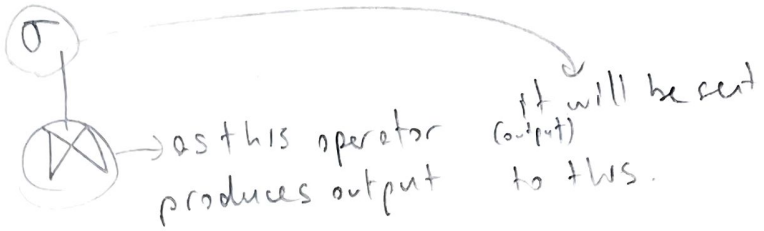
2 execution type
- → Pipelined execution
- → Intermediate result (Keeping temp Tables)
  materialization

Each operator (π, σ, γ, ⋈) implements an interface.

open() → init & point to file, set parameters
getNext() → get next item in the stream
close() → close

Pipelined Execution

- No operator sync
- No write to disk (intermediate tables)
- No read intermediate table from disk



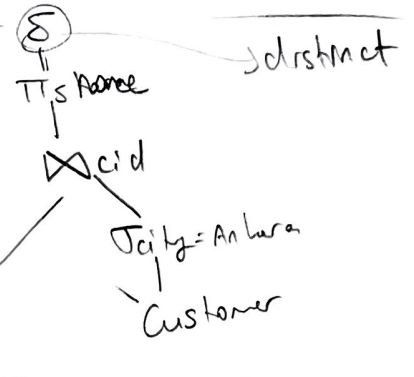
Intermediate Tuple Materialization

- No direct benefit
- Necessary for some op.
- When the op needs to examine some tuples multiple times

ex:

```
S. D. x.store
F Purch x, Cust y
W xcid=ycid
  & ycity=Ankara
```

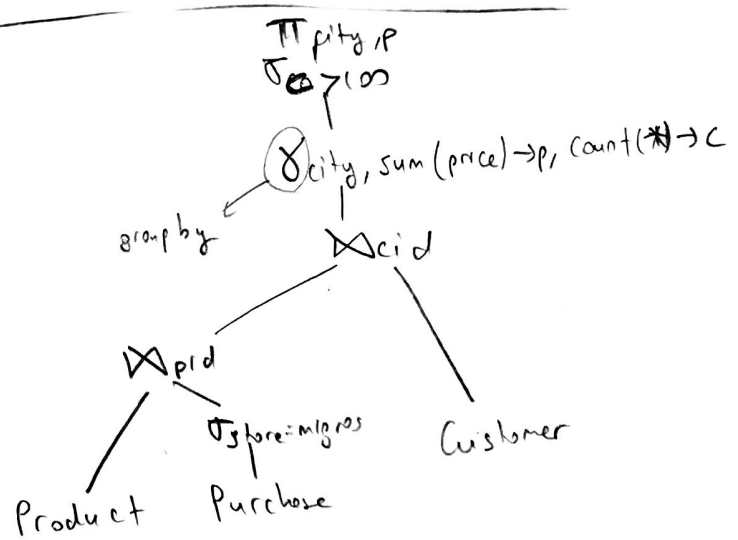
→



ex2:

```
S. z.city, sum(x.price)
F Product x, Purchase y, Cust z
W x.prid=y.prid & y.cid=z.cid
  & y.store = Migros
Group By z.city
Having count(*)>100
```

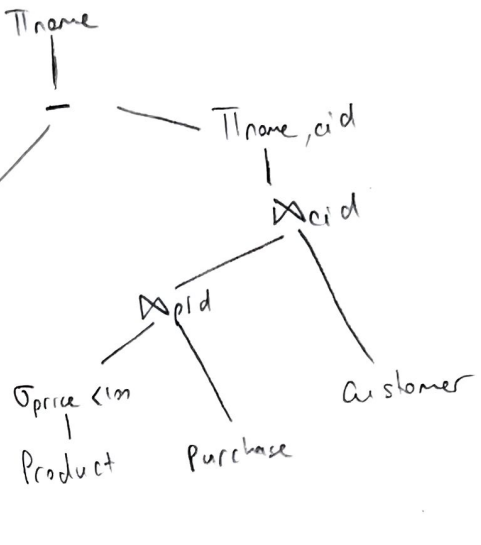
→



ex3:

```
S  z.name
F  Cust z
W  z.city='Ankara' and
NOT EXIST ( S  *
            F  Product X,
               Purchase Y,
            W  x.pid=y.pid &
               y.cid=z.cid &
               x.price<100 )
```

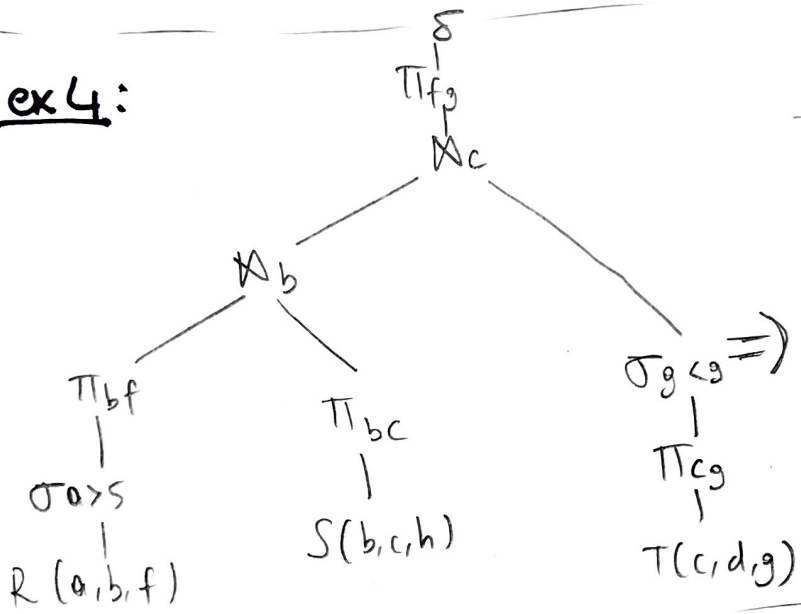
⇒



cid olması aynı 2 isimli (2 müşteri aynı farklı cid) elemesini engelliyor.

---

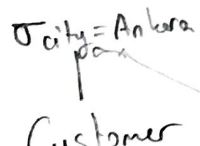
ex4:



⇒

```
Select Distinct f,g
F  R, S, T
W  R.b = S.b &
   T.c = S.c &
   R.a > 5 &
   T.g < g
```

Selection (σ)



If customer does not have an index on city

↳ File scan. Cost $B(R)$ → size of R in blocks (page)

If rows are sorted on attribute AND operation is >= <= (range)

Use binary search $(\log_2 B(R))$ → locate first data page.

+

scan linearly → there could be more than 1 page containing that data

$$\text{Cost} = \log_2 B(R) + \text{cost of scan}$$

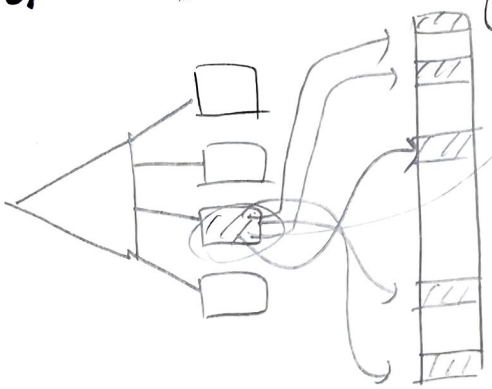
B+tree on attribute

a) Clustered

Start from root. Each level is a disk access.

Cost = depth of tree + # of pages occupied

b) Unclustered



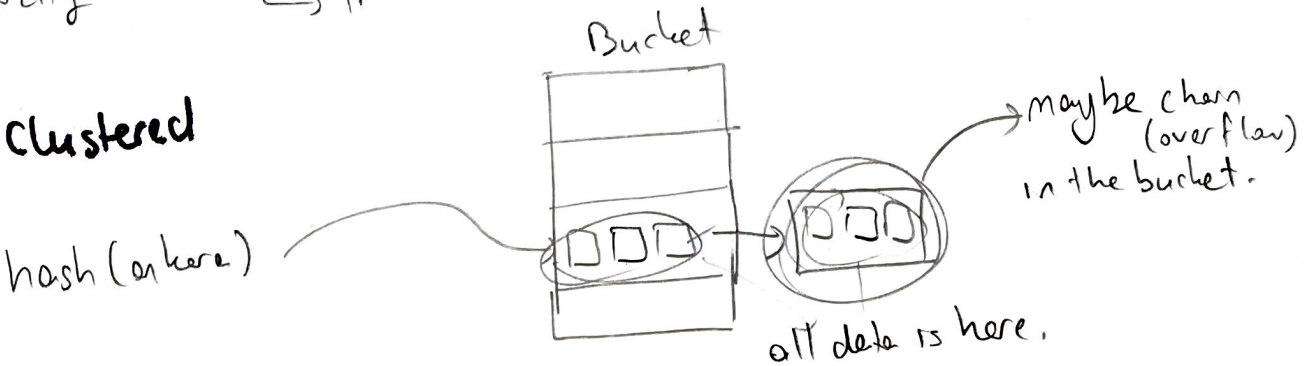
Cost = depth of tree
+
in the worst case, each entry points to a different page (1 disk access for each of them)

Hash Index

Cost ≅ 1.2

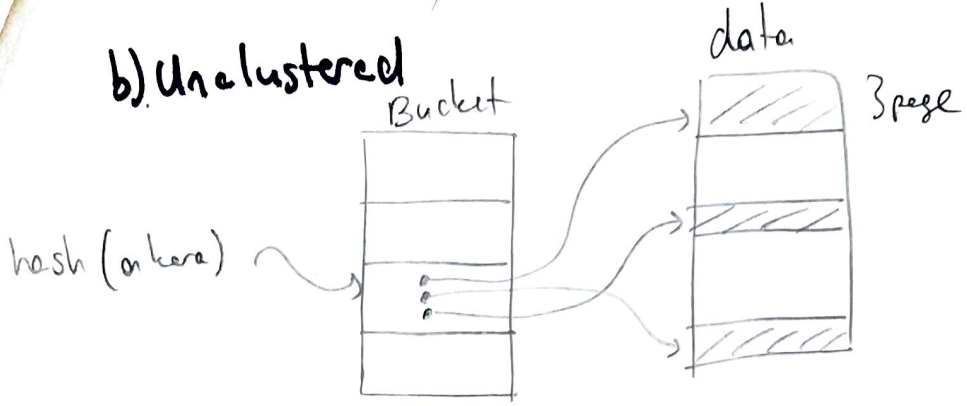
∇city = Ankara → hash ankara
→ find correct bucket, which contains hash of Ankara

a) Clustered



cost = # of pages in the bucket.

b) Unclustered



Cost = # of rows in bucket.

Farkları: Clustered olunca Bucketta page şeklinde data duruyor. hash ile doğru bucketı bulunca Bucketteki pageı almanız → datayı almanız demek.

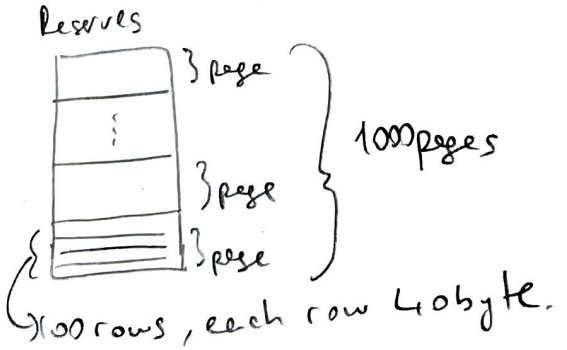
Unclustered olunca Bucketta bir sürü row oluyor. Bu rowlar disketeki yeri point ediyor. Her row için bir disk access yapmanız gerekebilir.

---

ex: Sailors (sid, sname, rating, age)

Reserves (sid, bid, day, rname)

Reserves → each tuple 40byte
→ 100 tuple per page
→ 1000 pages



Sailors → each tuple 50byte
→ 80 tuples per page
→ 500 pages

Questions

S*
F Reserves R
W R.rname = Joe

1) With no index, unsorted

↳ file scan ⇒ B(R) = 1000 → 1000 page I/O

2) With no index, sorted.

↳ $\log_2 B(R) \simeq$ 10 I/O. to find first Joe

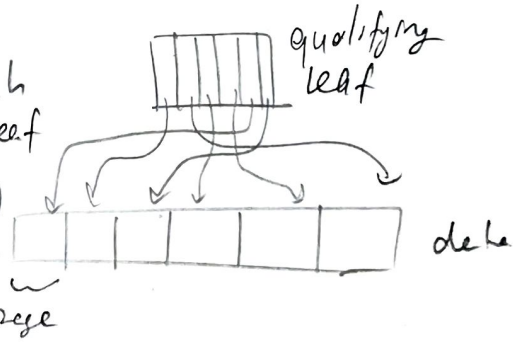
3) B+tree, index on rname, clustered

cost of finding this = depth of tree

= 2 or 3 I/O ?

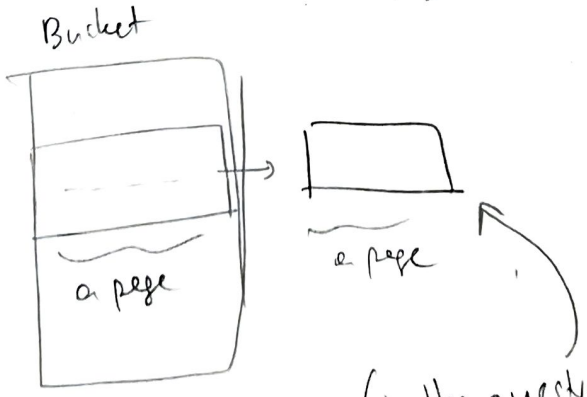
4) B+tree, index on rname, unclustered.

2 or 3 I/O for finding Joe → then for each* entry in the leaf (1 page access)



5) Hash index

1 or 2 I/O for retrieving index



If # of reservation of Joe ≥ 100 (in the question 100 tuple per page)

If clustered 1 + maybe 1
unclustered for each entry a page access

Range query example

S *
F Reserves R
W R.name < 'C%'

- Assume we have 10% of Reserves in the result (total is 1000 * 100)
  ↳ 10,000 tuples → 100 pages

Clustered index → 1 or 2 disk access for index + 100 I/O

Unclustered index → each tuple can point different page. → 10,000 I/O. (Might be cheaper to use file scan 1000 I/O)

Projection (π)

- Remove unwanted attr.
- Eliminate duplicates

2 approaches → Sorting
↳ hashing

$\pi_{sid,bid}$ | Reserves → S sid, bid F Reserves

Sorting

Reserves → each tuple 40byte, 1000 page.

S Dist sid, bid
F Reserves

Assume we have 20 buffer page.
- // each small tuple (sid, bid) = 10 byte

Bizde 40byteten 10'a düşüyor.
M page in memory → each run will have 4M.

→ Pass 0.
read 1000 page. out smaller tuples.
If M pages in memory, runs of 2M pages can be produced (size dependent *)
(40byteten 20'ye düştüğü için böyle)

Hoca derste çok hızlı geçti.
Nasıl yapıldığını bilmiyorum

Projection / Sorting

S Dist. Rsid, Rbid
F Reserves

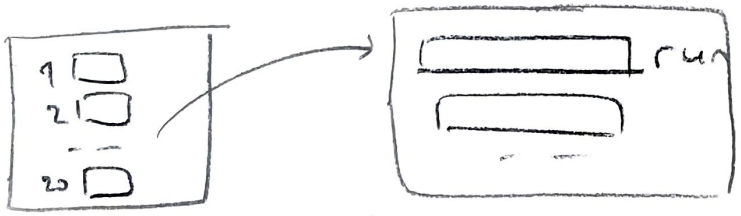
Reserves, each tuple 40byte, 1000 page.
- Assume each smaller tuple (sid, bid) = 10byte.
- // we have 20 buffer pages.

1) Read 1000 pages.

Normalde her pagede 40byte var, ama projection yaparken 10byte oluyor. Bu yüzden toplamda 250 page yaz.

- Slidedeki 7 runs 40 page muhabbeti:

↳normalde



each run will be 20 pages long.

Ama farklı bir algoritma kullanıldığı zaman 2 katına çıkarabiliyor.

★ Önemsiz bilgi ★

250 page yazdım

2) Read 250 page & merge

Total cost 1000 + 250 + 250 = 1500

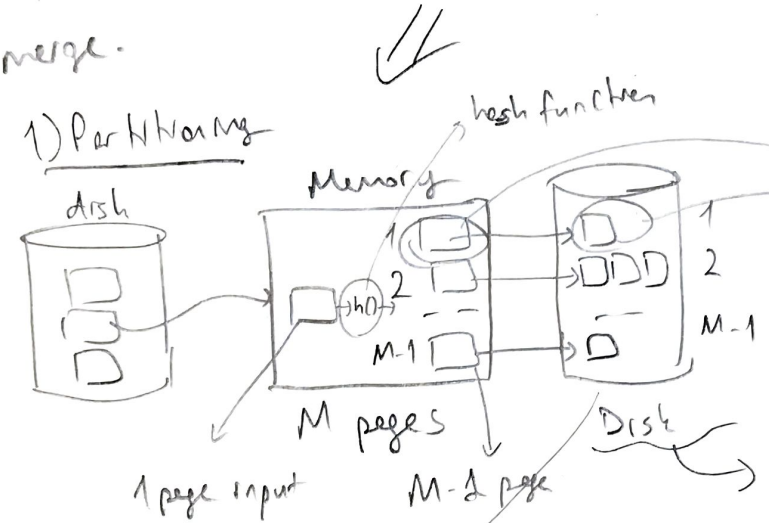
Hashing

as I read from disk, build hash table in memory. (Suppose it fits in mem.)

each (sid, bid) will be hashed and it will not be put, if it already exist (Distinct).

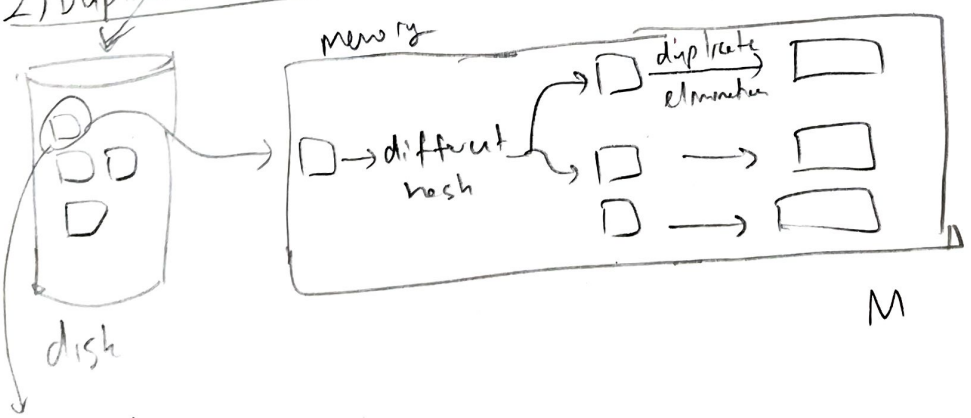
If it does not fit in the memory, send it to disk. Then 2-way merge.

1) Partitioning



It is only hashed. (101, 500) → may not be in the same bucket.
(102, 600)

2) Duplicate Elimination



since it is a small file it will fit in memory