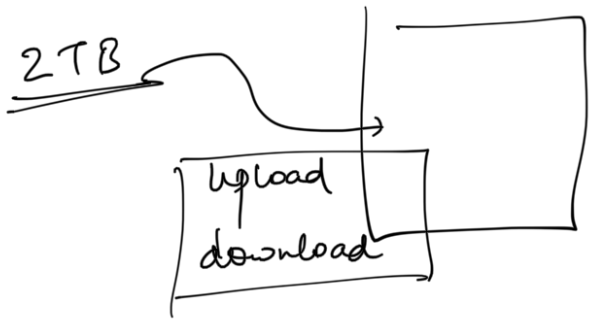
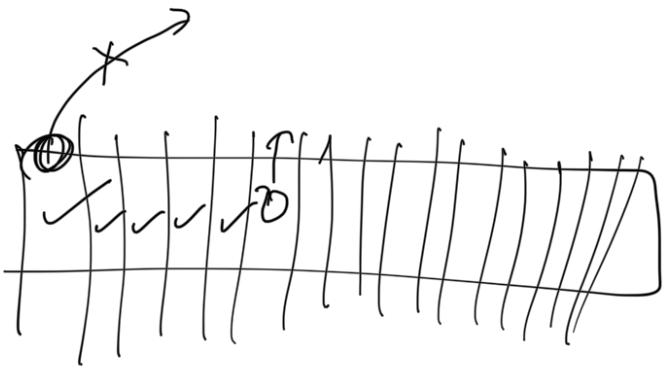
S3 / QUAD TREES

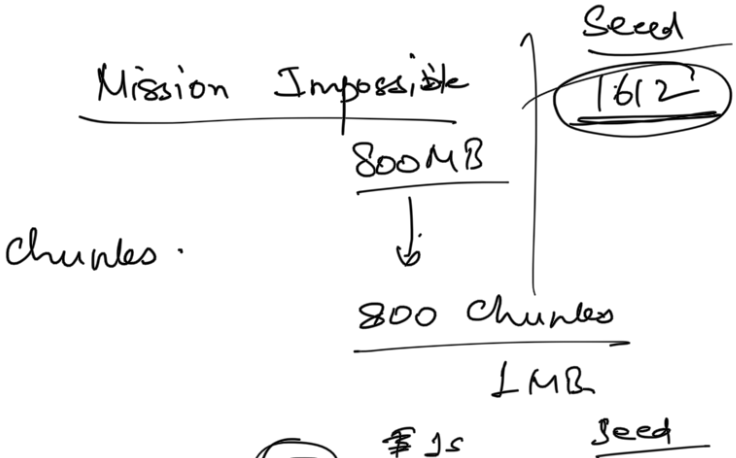


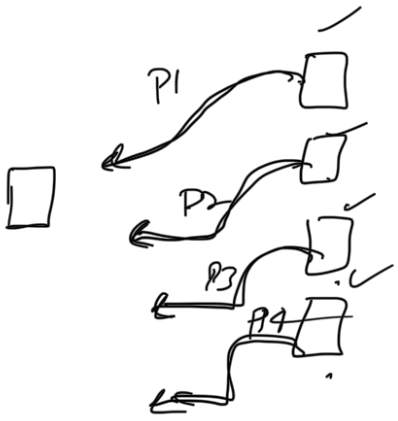
HDFS

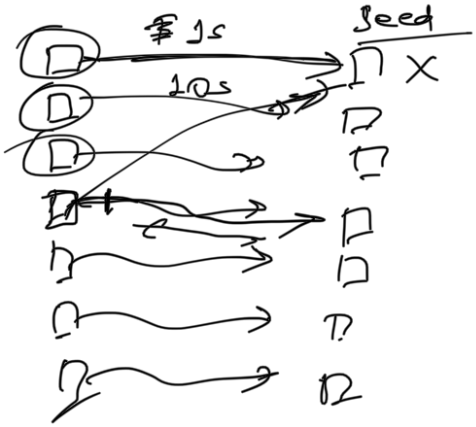
1. Storage should be able to store large files
2. Reliable & durable
3. Retry / Network failure → gracefully
   Work repeated Should be minimized
4. Download the uploaded file



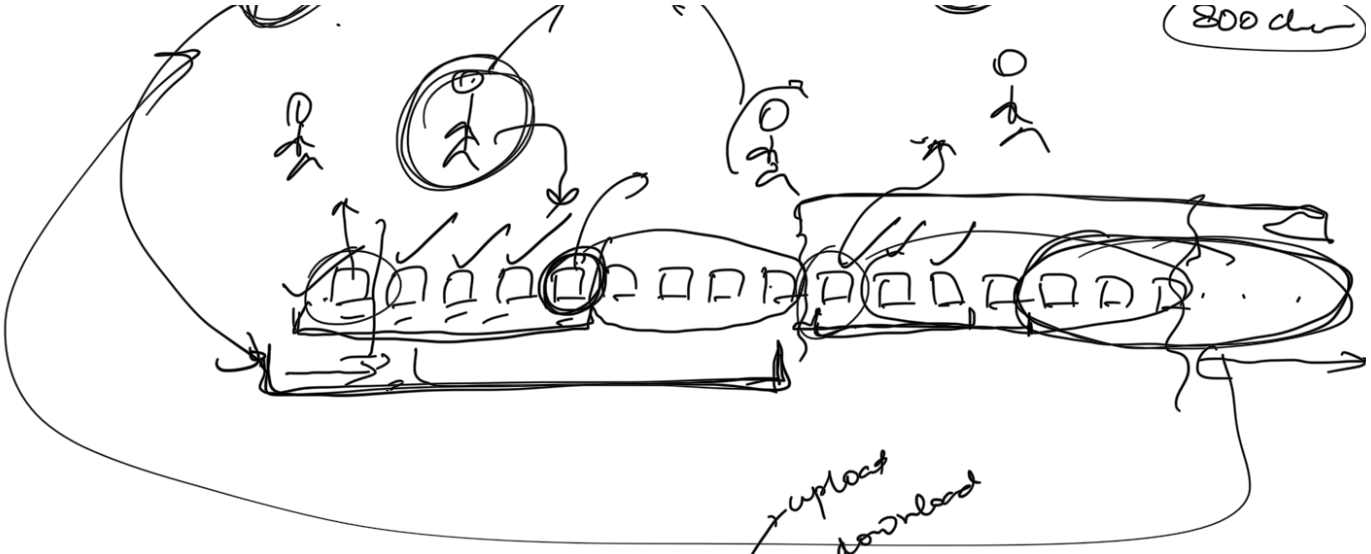
Download upload ←



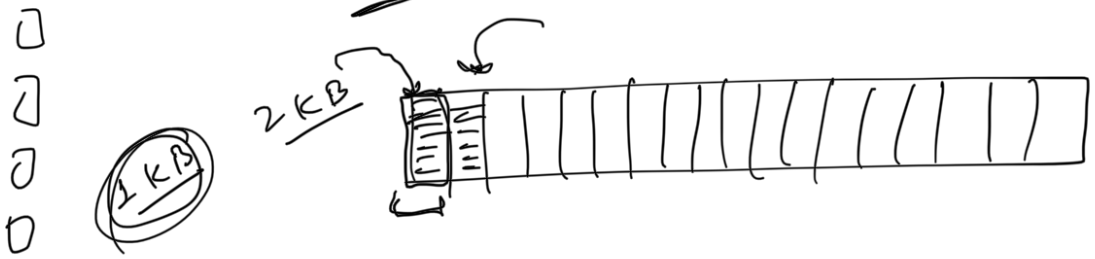




mi1.avi

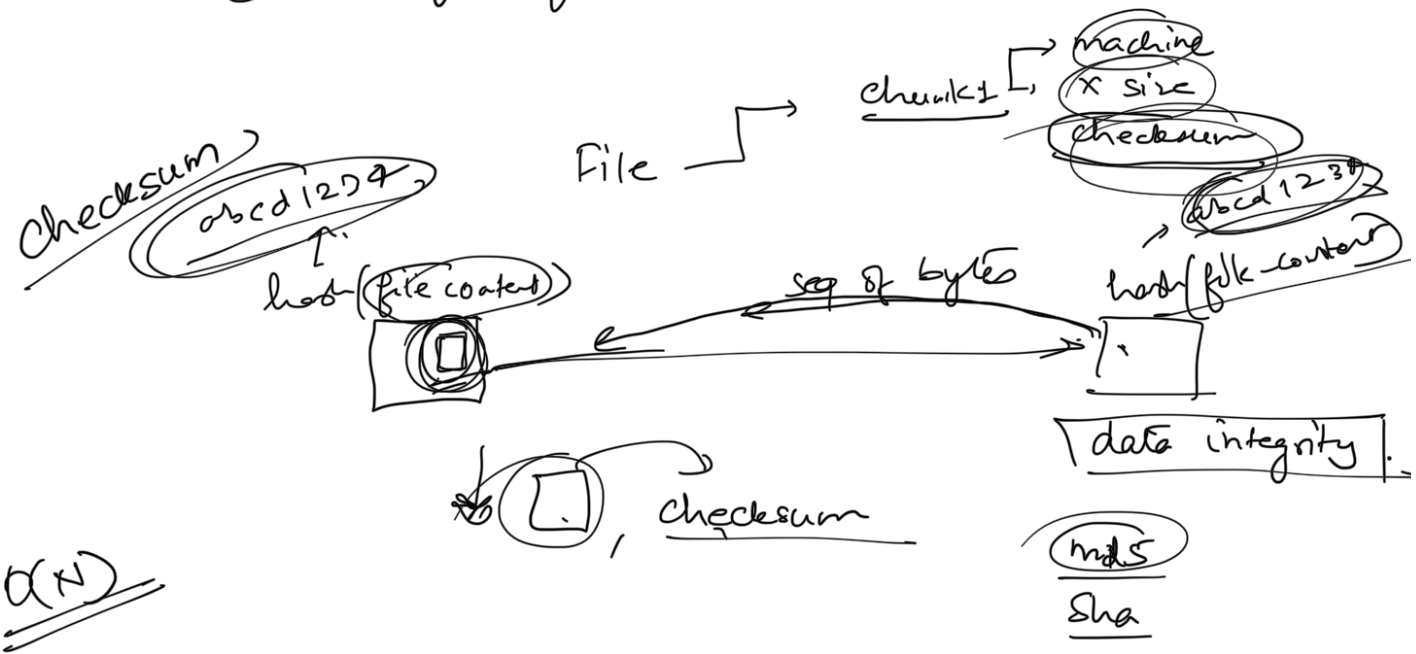


800 chunks Multipart → upload / download X seed users.



---

(1) Large file → break into chunks.

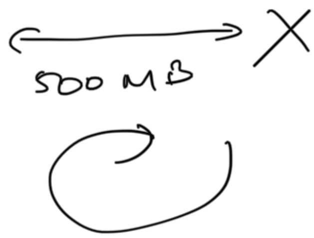


O(N)

cksum <filename>

1 GB

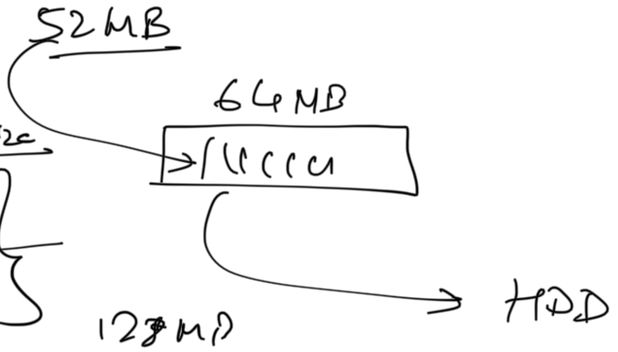
Total size

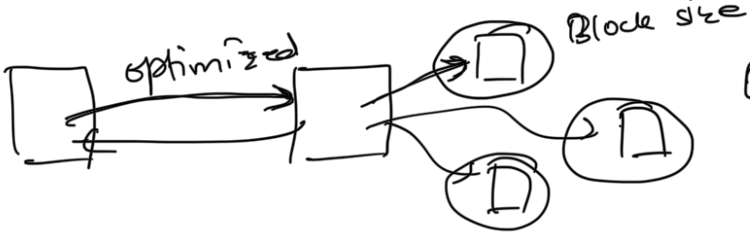


① 4 MB → default HTTP req limit

64 MB

	Chunk size
HDFS 1.0 →	64 MB
2.0 →	128 MB



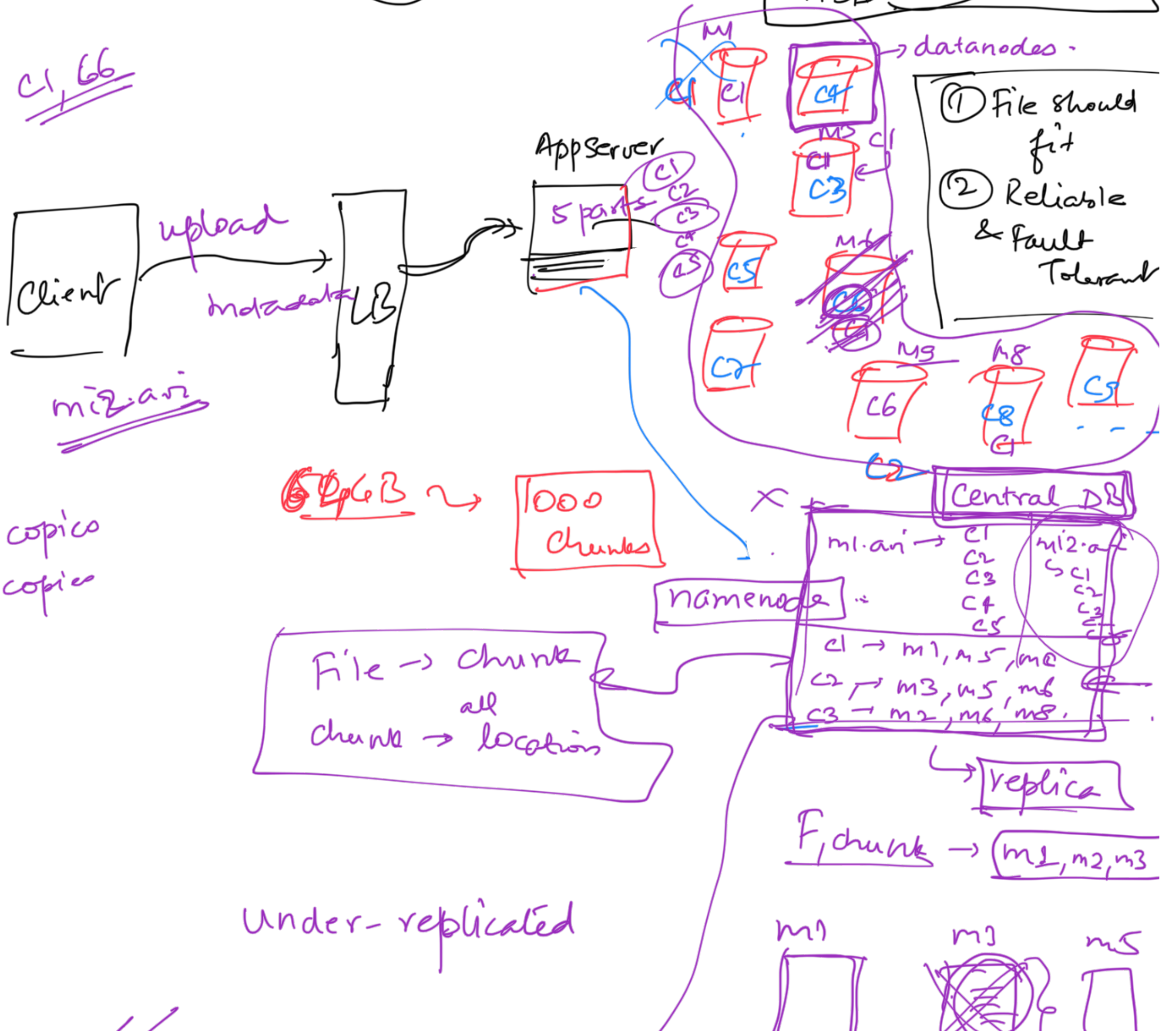


OS → Memory management

RAM	Paging
HDD	Blocks

c1, 66

mi2.ari



① File should fit
② Reliable & fault Tolerant

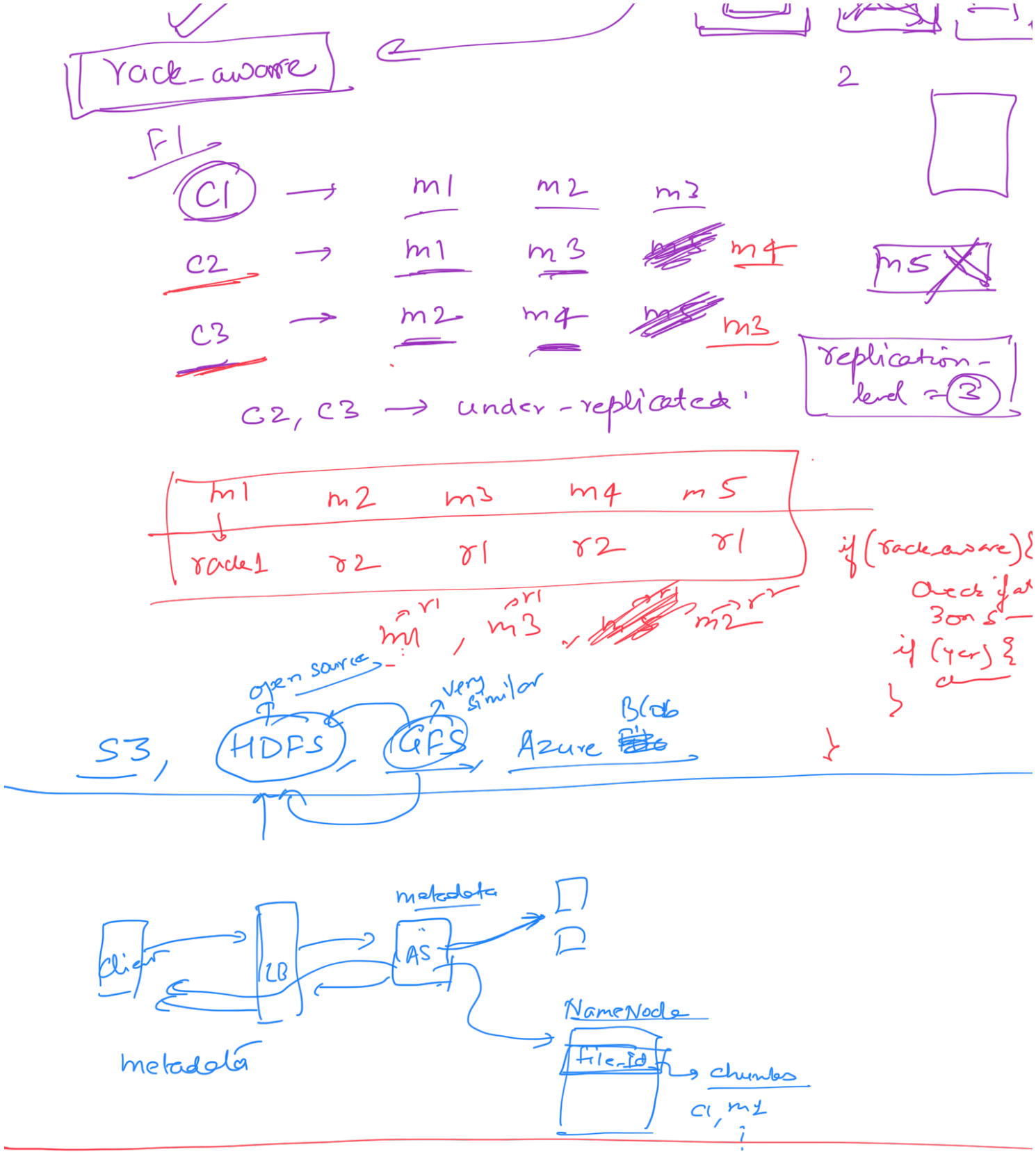
64GB → 1000 chunks

2 copies
2 copies

File → chunk
chunk → all location

F, chunk → m1, m2, m3

Under-replicated



PROBLEM #2:

Nearest neighbor problem

Brute Force

get_closest_X(lat, long) : O(N)

for rest in all-rests :

dist = $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

if dist of maxheap.top() is greater than dist :

maxheap.pop

maxheap.insert(rest)

Very very slow

SQL

rest_id	name	X	Y

index (X, Y)

(lat, long) K → SLOW → SELECT * FROM restaurants

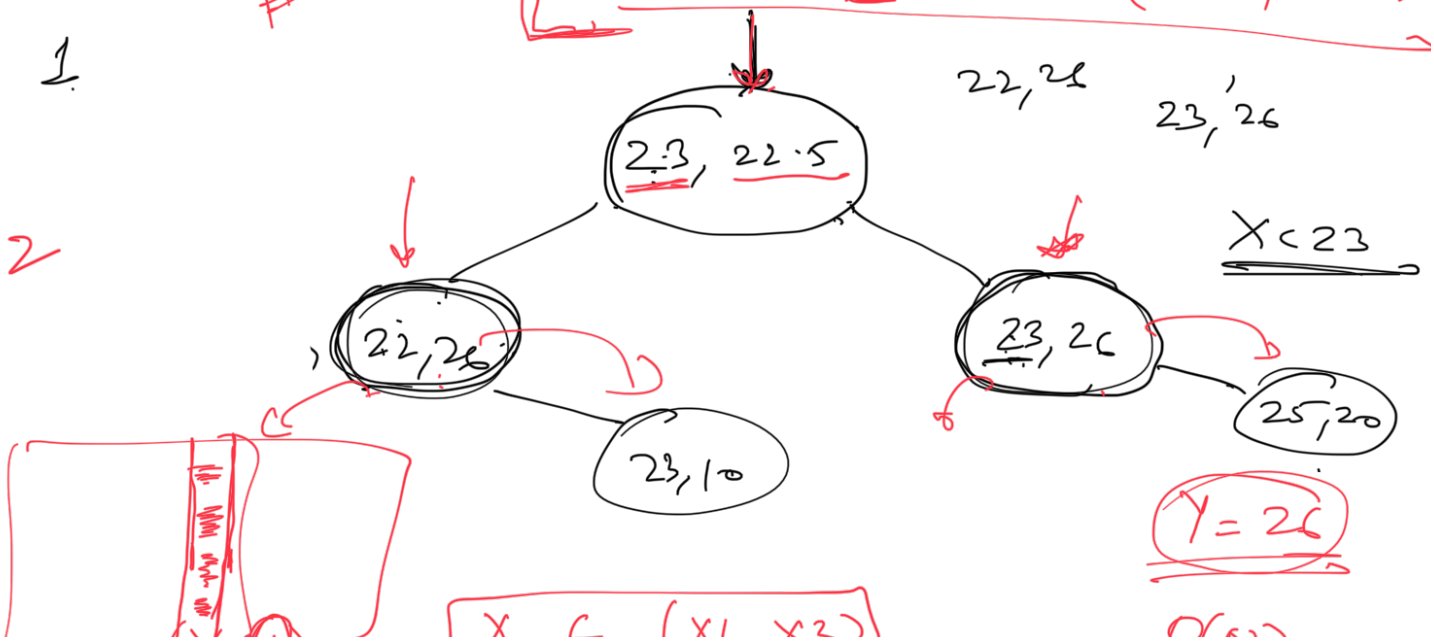
WHERE

→ X between (lat+k, lat-k)

AND Y between (lon+k, lon-k)

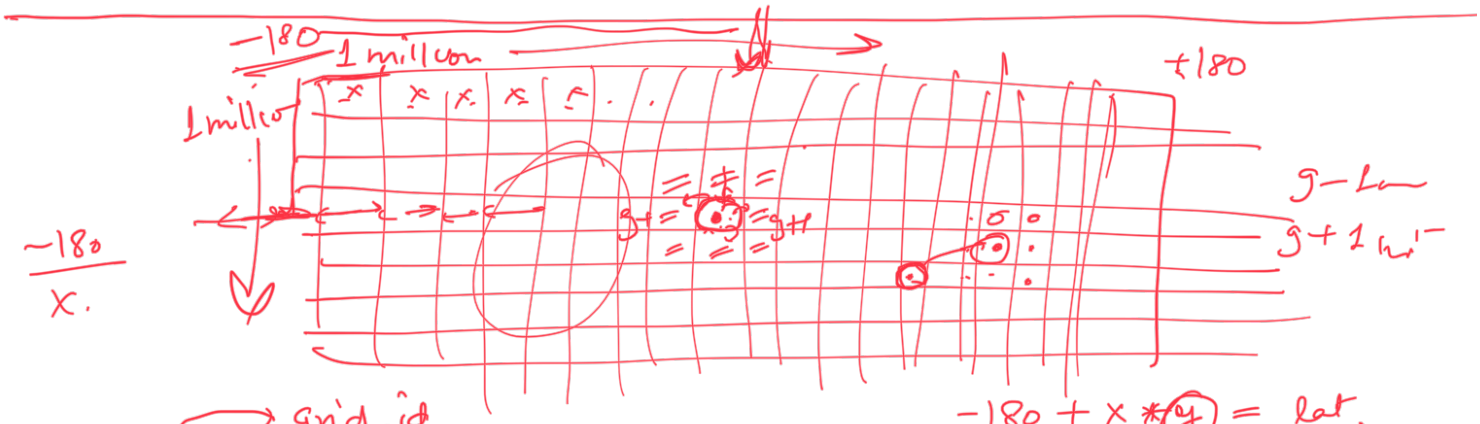
Approach 2

1.



2

→ Y ∈ (Y1, Y2) OK



(lat, long) → grid_id

rest → grid

$-180 + x * y = lat.$

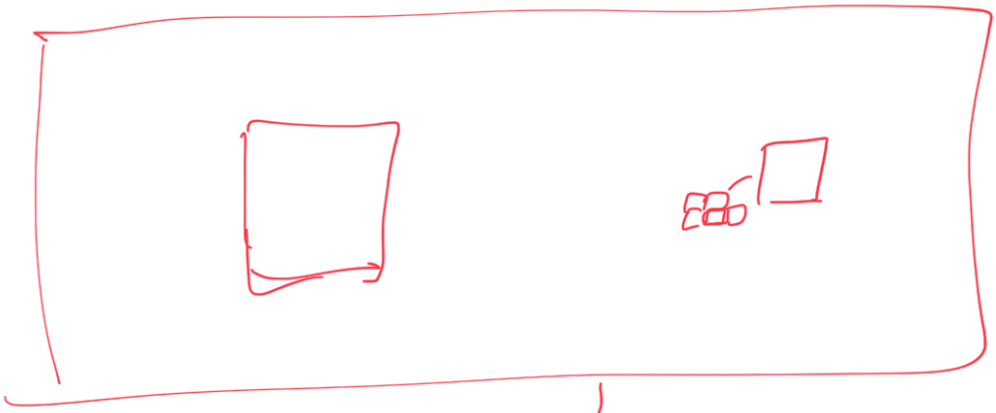
$y = \frac{lat + 180}{x}$ indexed

rest_id	name	lat	long	grid_id

grid_id → neighboring grid_ids

FAST ↑ enough results.

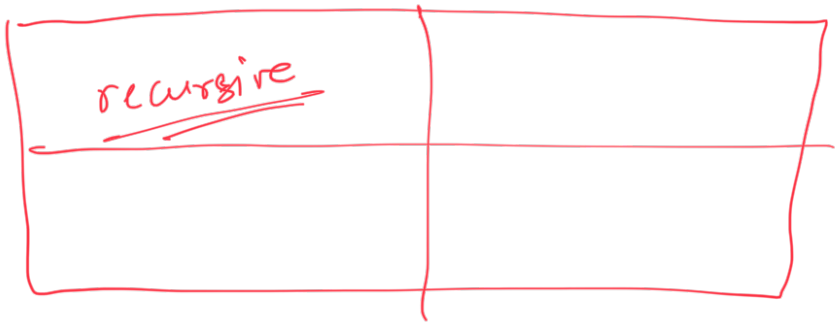
```
SELECT * FROM restaurants
WHERE grid_id IN (g1, g2, g3, ...)
```



① variable size grids

② (x, y) → grid → neighboring grids.

> 50 rest



① CSD

Quad Tree

k-d tree