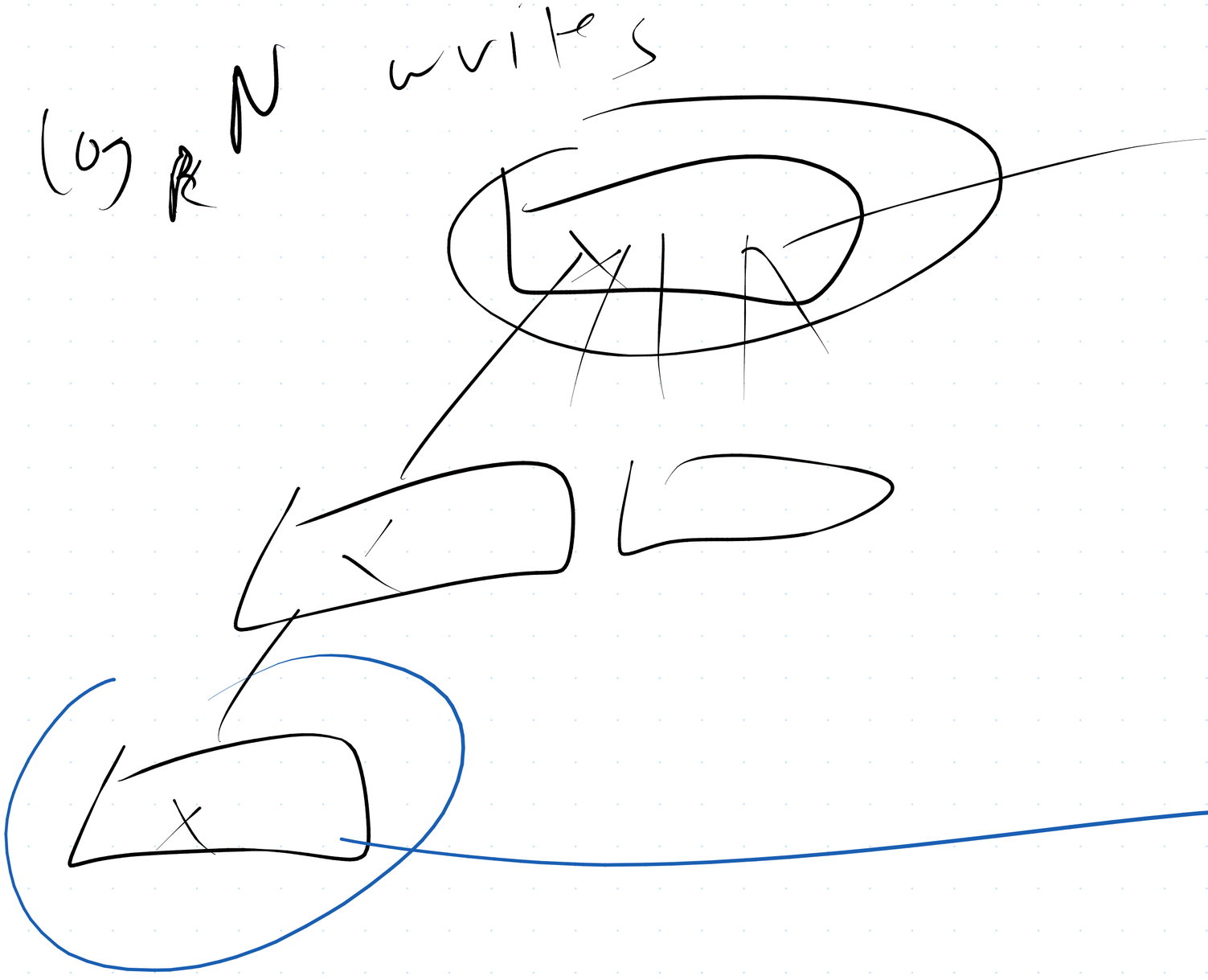
CSE 410 – Advanced Data Structures
Topic 07: Write-Optimized Structures

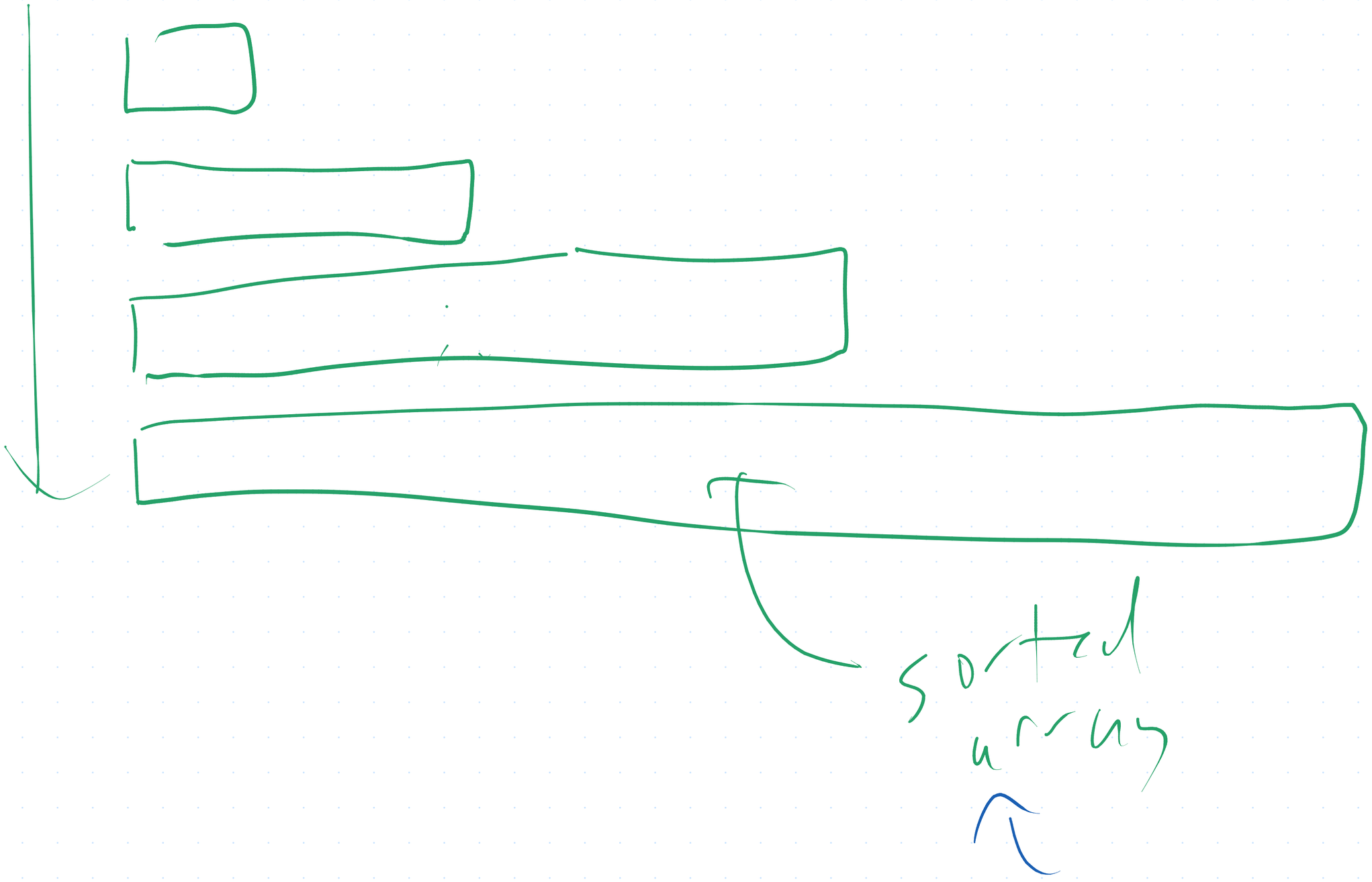
Oliver Kennedy

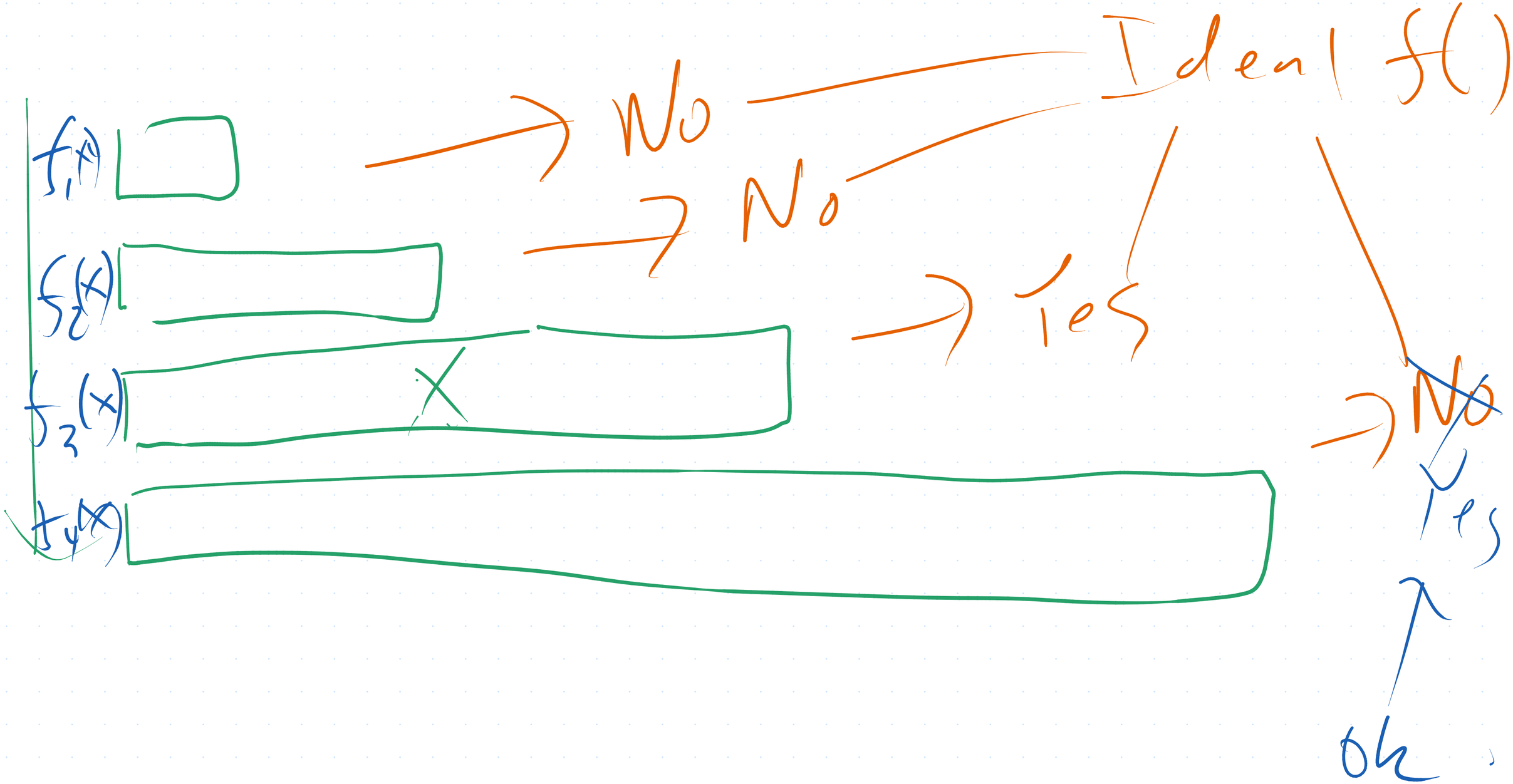
$\log_R N$ writes



High contention

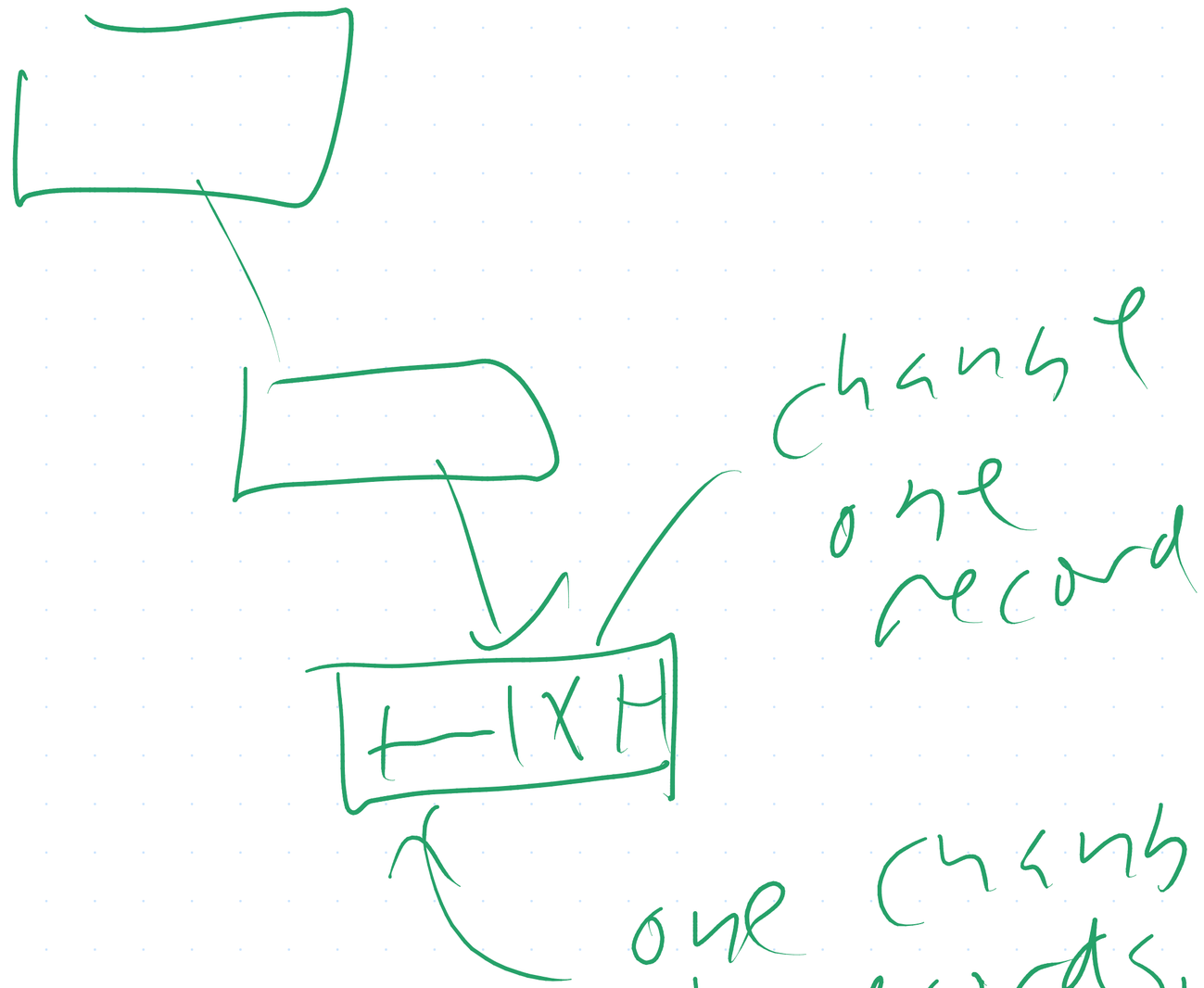
Coordination required


sorted
array

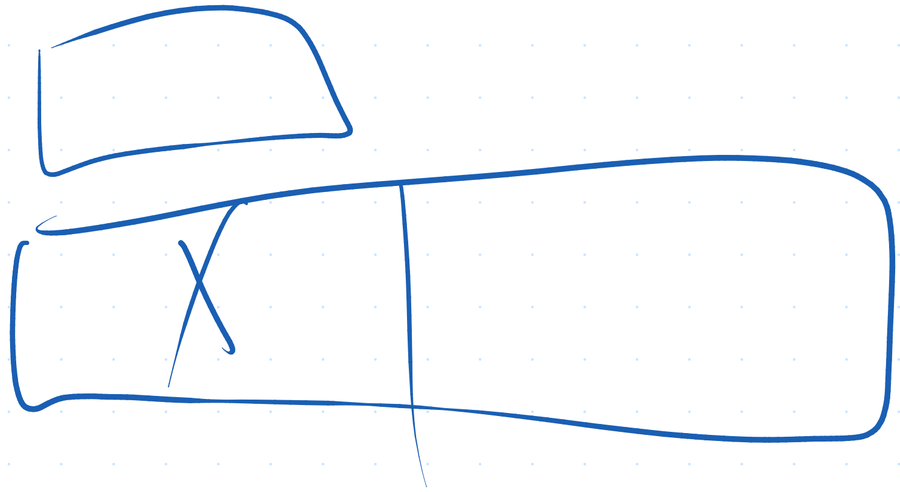

Iden(f( )
No
No
Yes
No
Yes
ok
f1(x)
f2(x)
f3(x)
f4(x)

Bloom filter

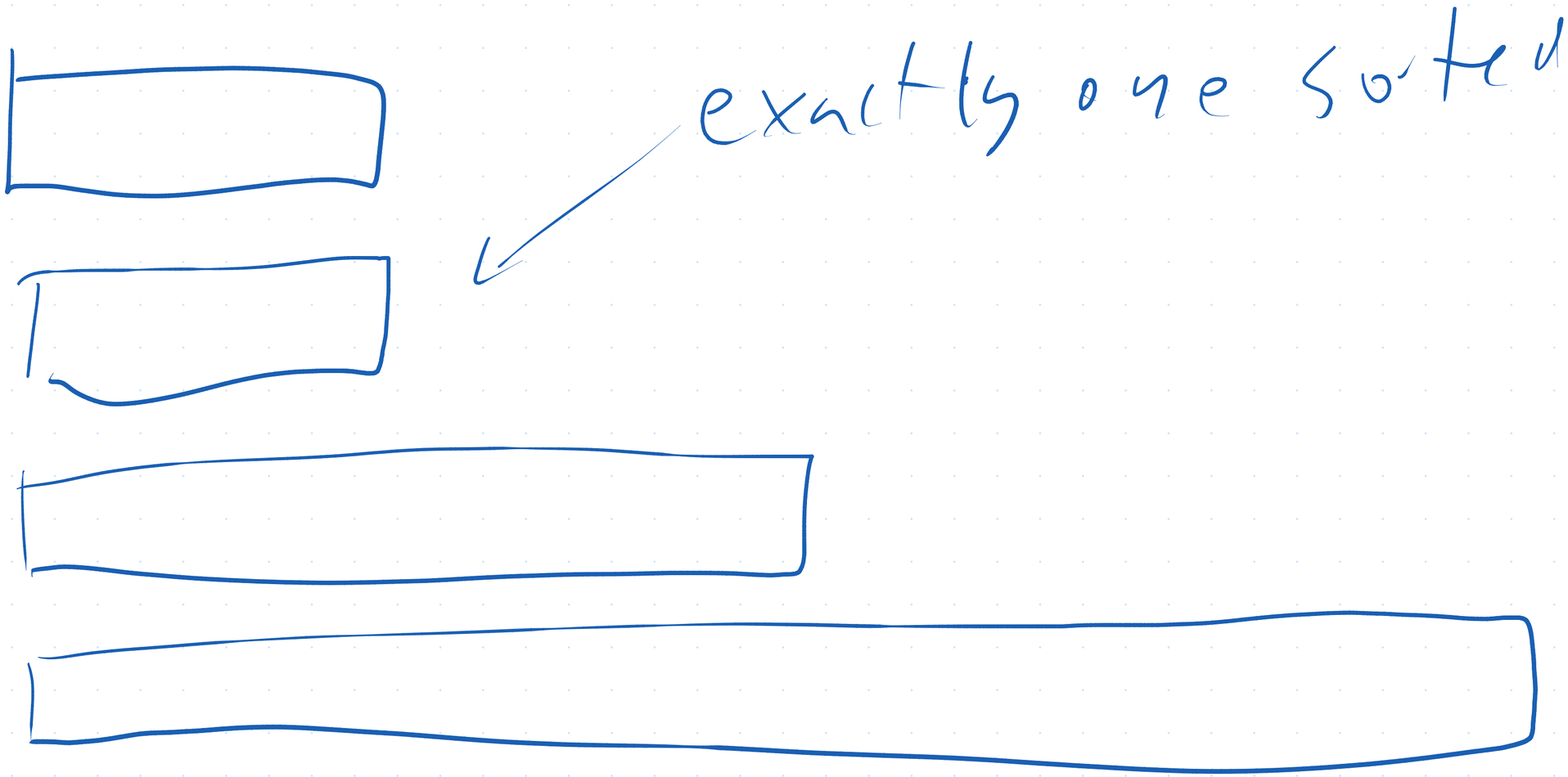
Bitwise OR


change one record
one change all records to be re-written out

Deletions & updates



Deletion = update to Null



Levels

O(N)

For all inserts

How is a sorted run stored?

- Array (sorted)
- ISAM Index / B+tree (original)
- OF-Based Index
  - or Interpolation Search
- Fence pointer table

Idea 1 : store min, max

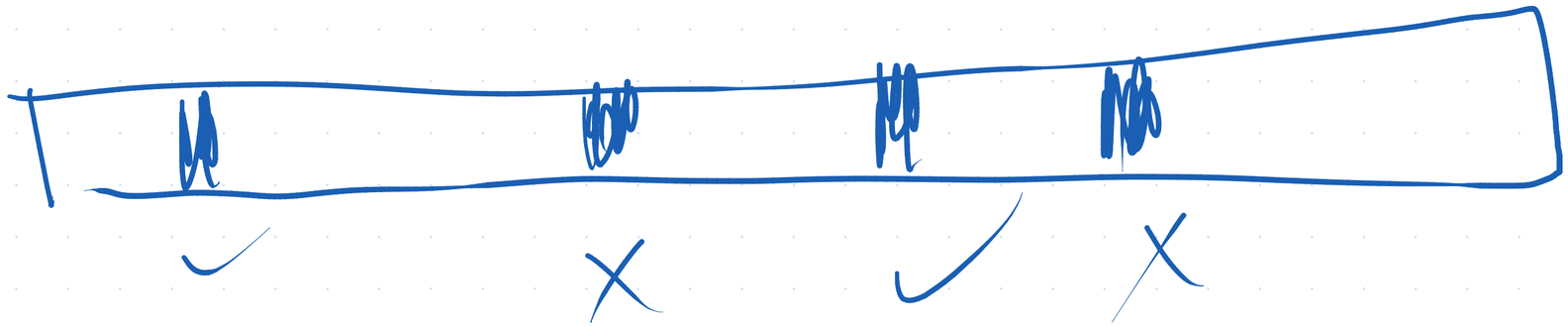
↳ discard requests that are not in range

Req: min, max

↳ Fence Pointer table gives us min max

↳ No overhead!

z →



Bloom filter parameters

- size of the bit vector
  - # of bits that each element sets
  - size of bit vector & number of elements

SAMEL

$\log_B N$ reads IO cost

B+ trees

LSM tree

unsorted array

read optimized

write optimized

goal

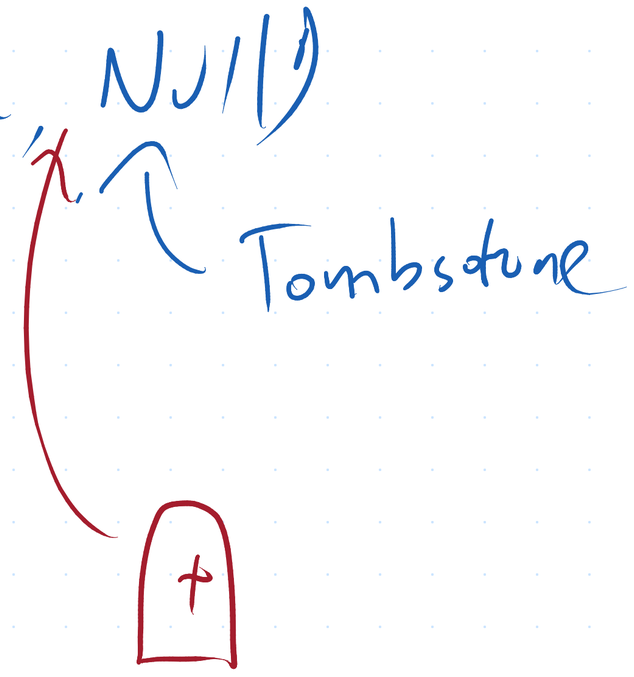
same read complexity, as B+ tree

$\log_B^2(N)$

$\frac{1}{B}$ read IO cost

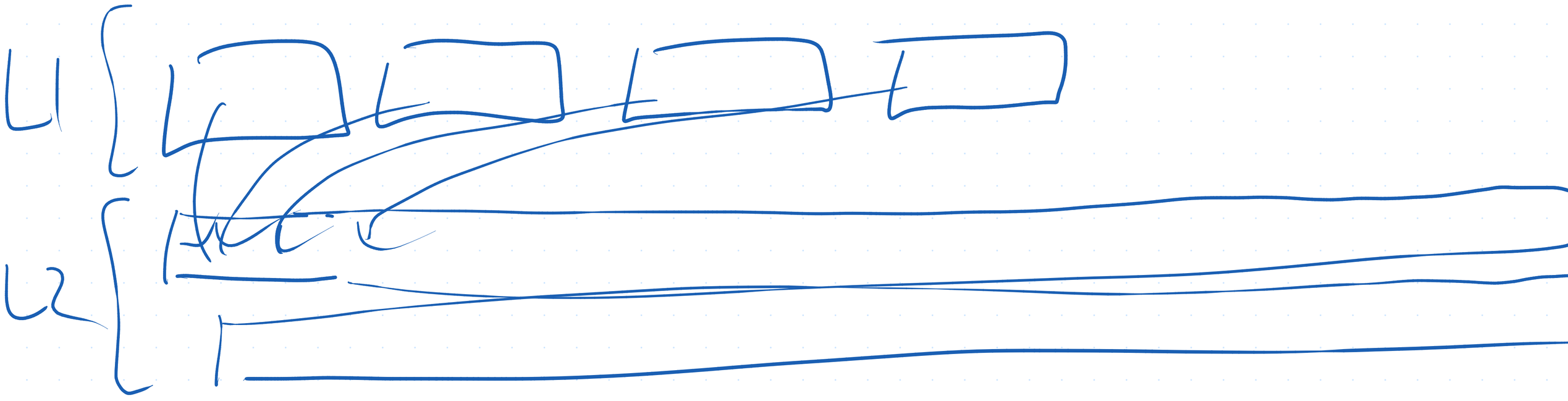
better

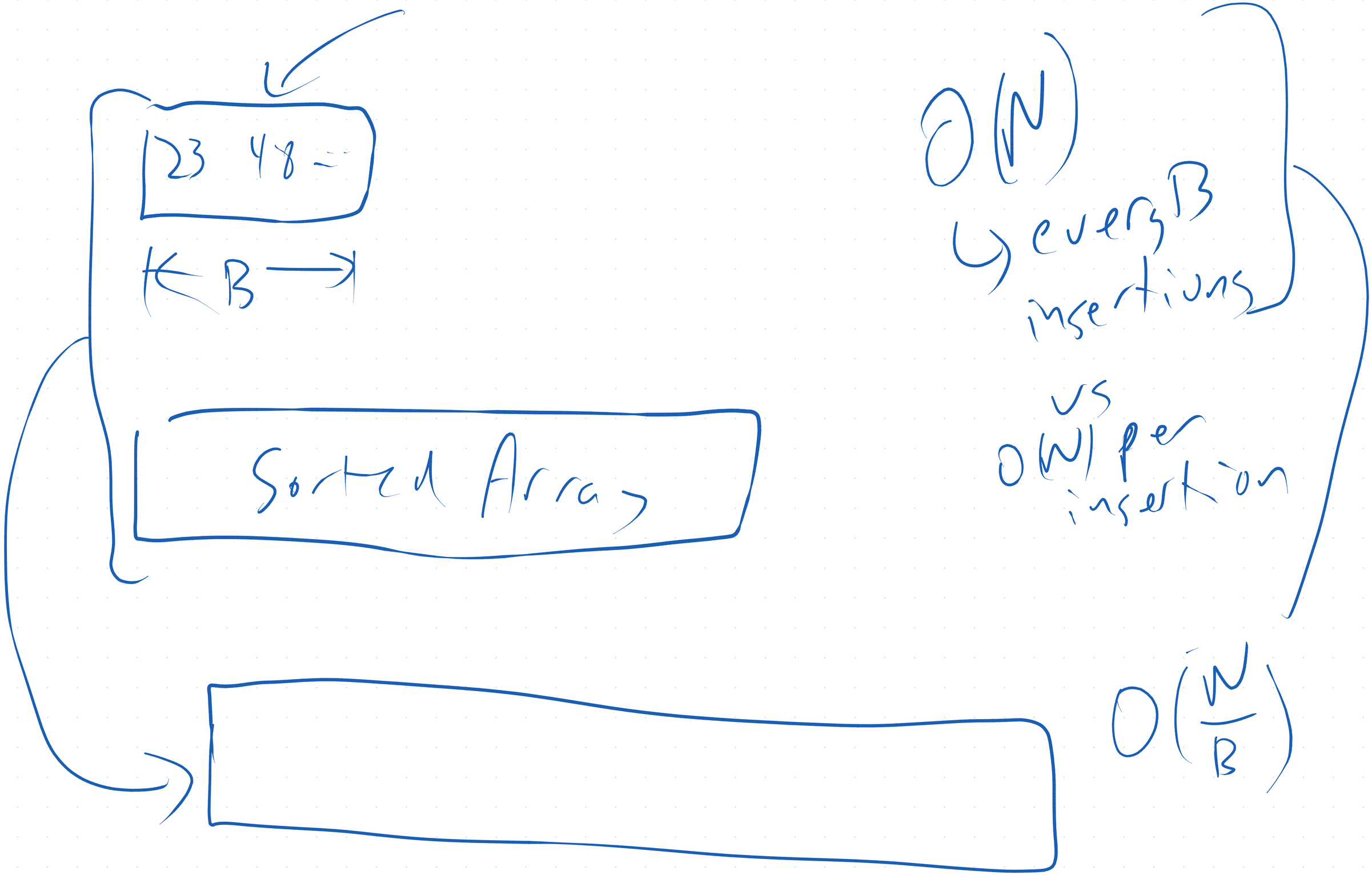
Delete(k) = Update(k, NULL)

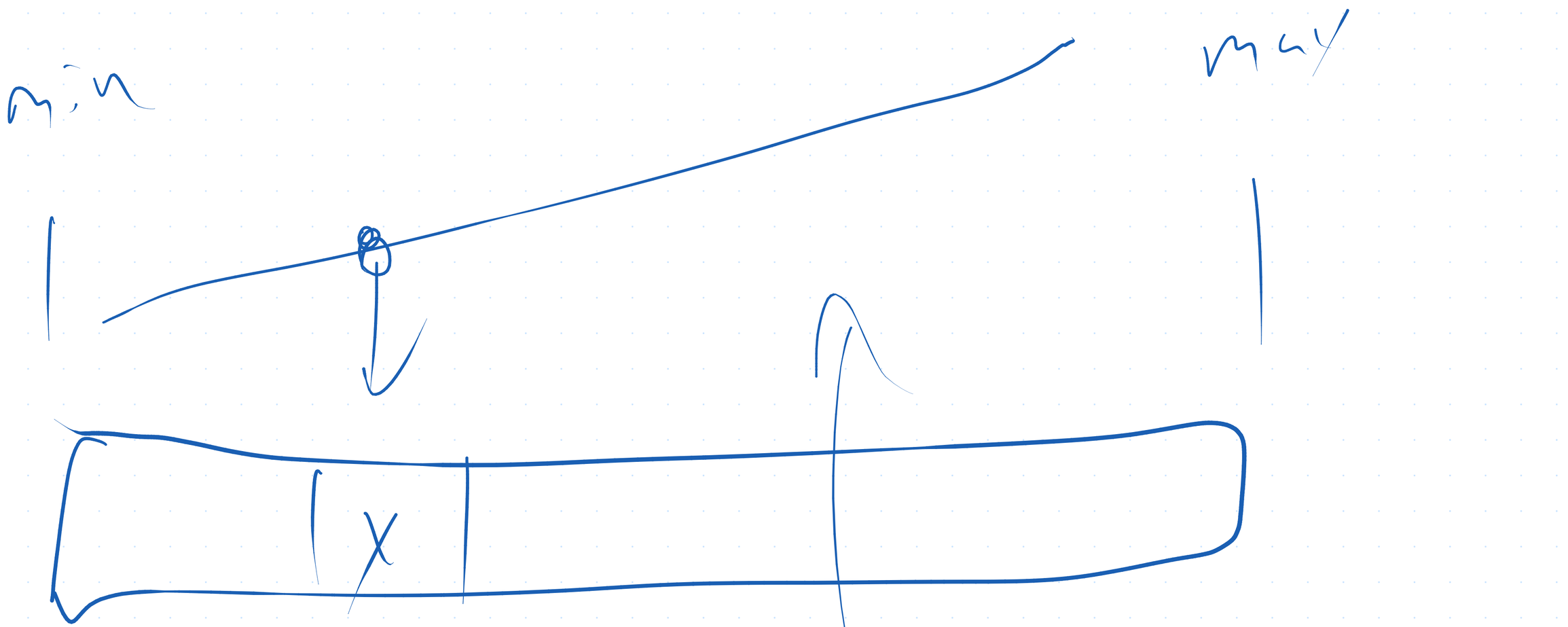


So how do we implement

Update(k, 42)


L1
L2


23 48 ...
B
Sorted Array
O(N)
every B insertions
vs
O(N) per insertion
O(N/B)



CDF-Indexes
Learned Indexes

$f_i(x) \Rightarrow$ { True : if x is a key on that level
false otherwise

Ideal

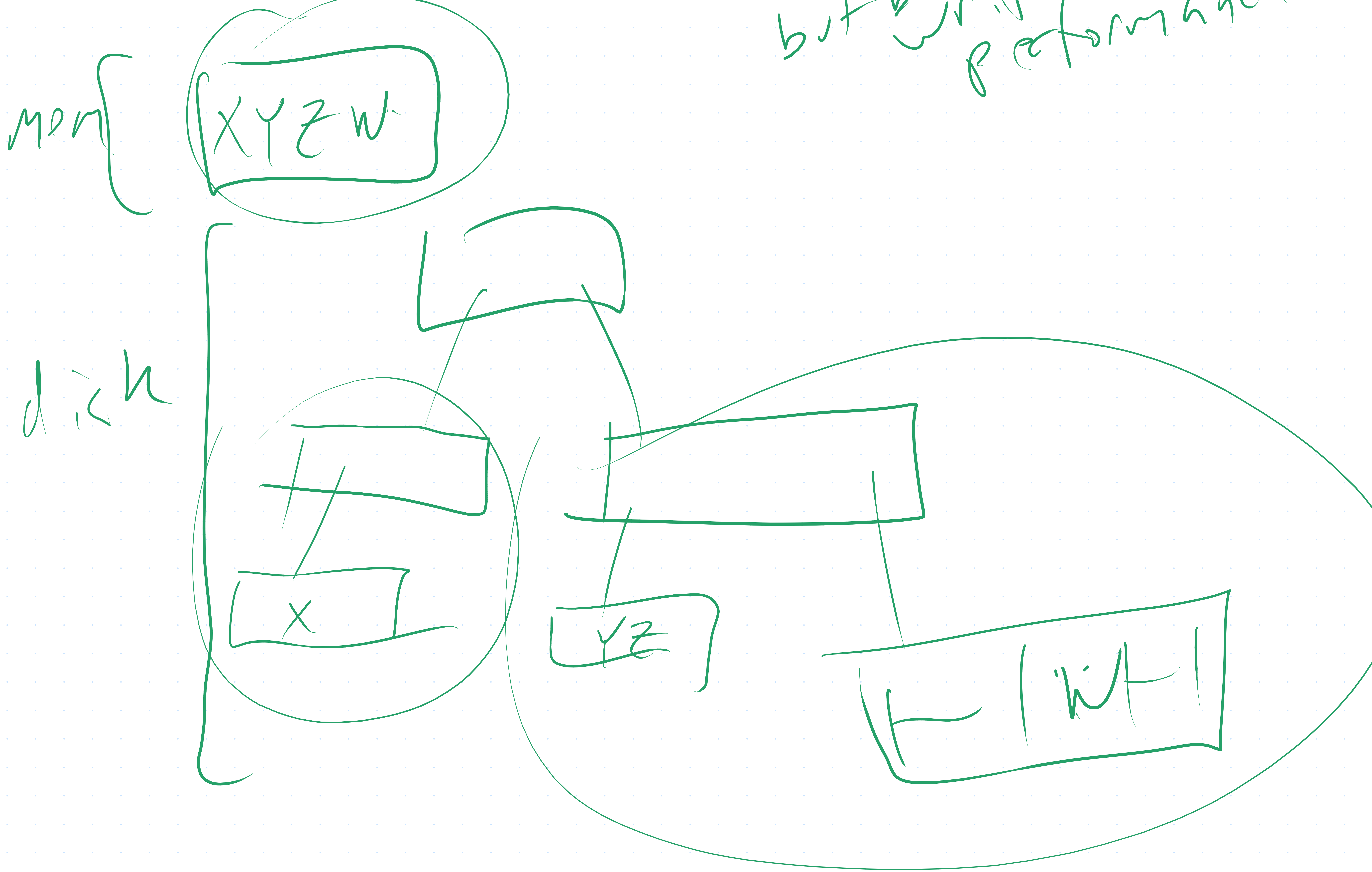
on

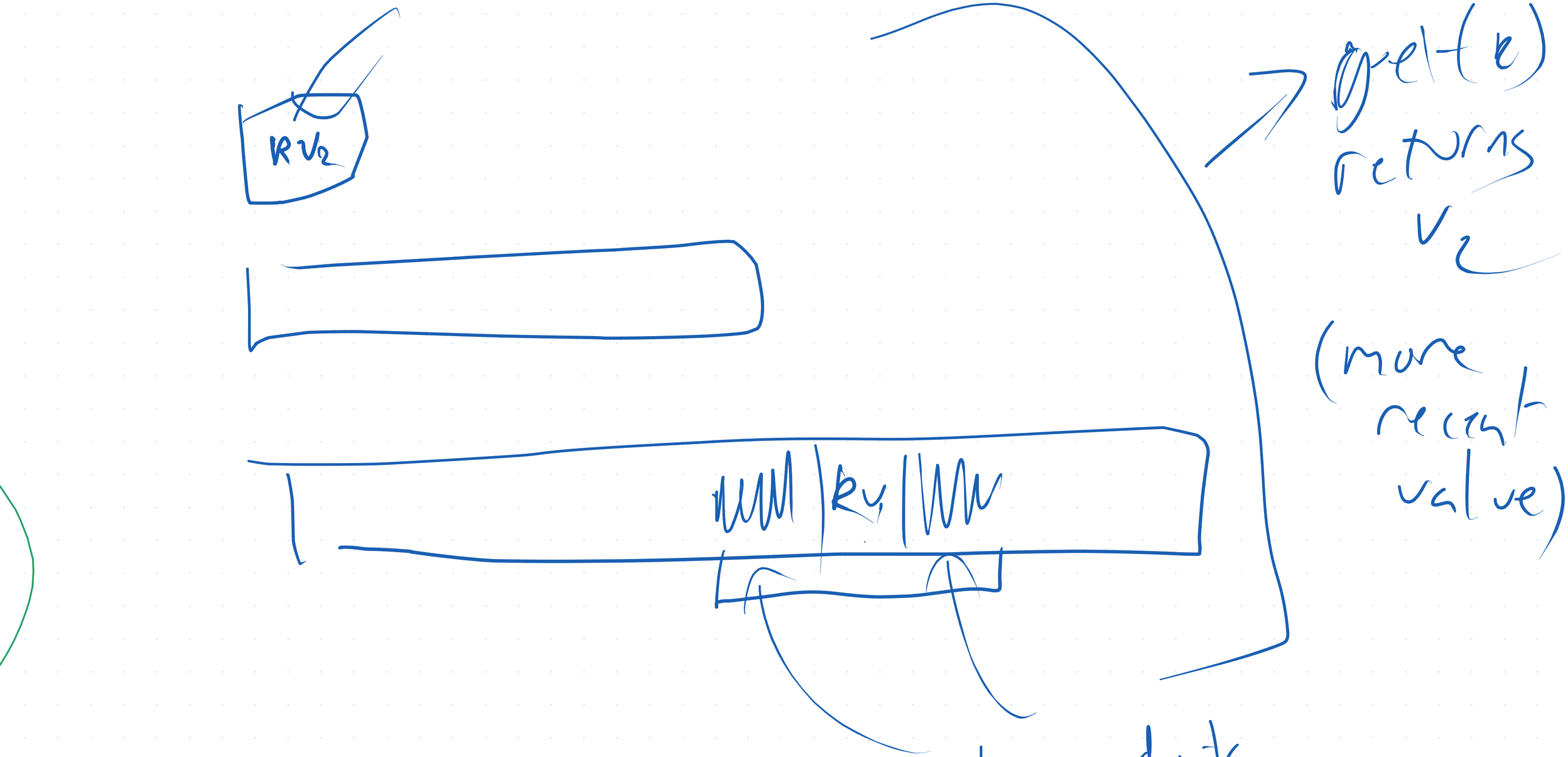
$f_i(x) =$ { false if $x \notin$ set of keys
true if x can be in the set of keys

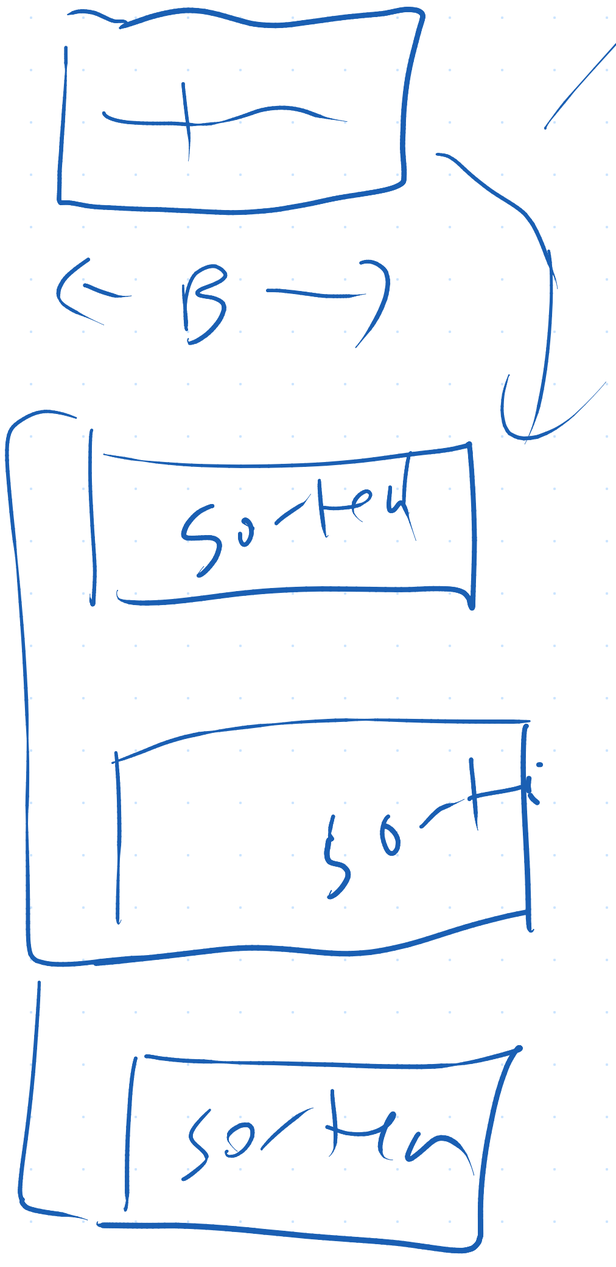
More bits per element

Lyon more chances to try collisions on test

Lyon more bits get set, so more collisions


mem
XYZW
disk
X
YZ


RV2
get(k)
returns
V2
(more
recent
value)
other data
on same page



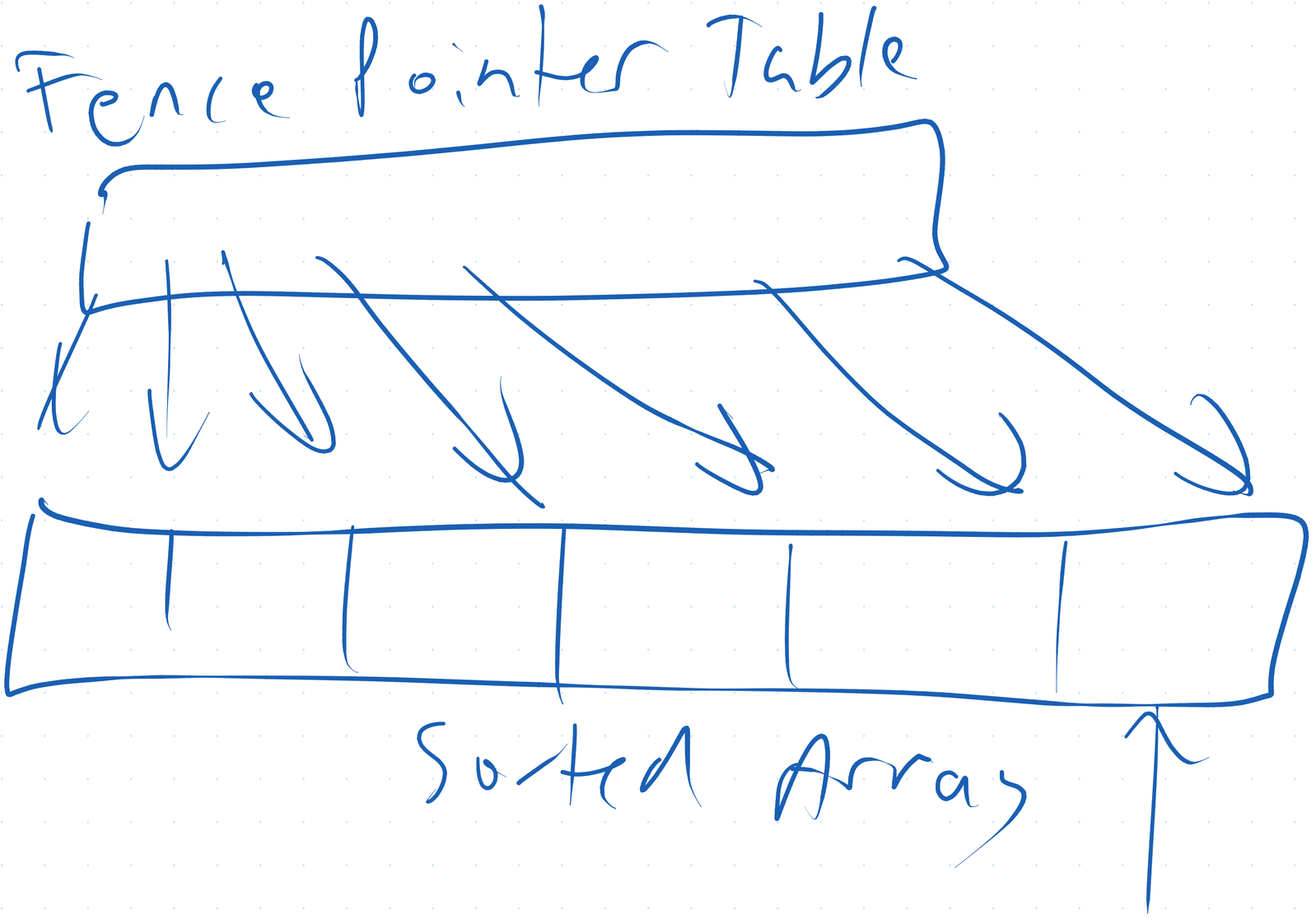
write sorted Buffer

$O(B)$ per B insertion

$O(1)$ amortized write

$O(\frac{N}{B})$ sorted arrays

↳ $O(\frac{N}{B})$ IOs per read


Fence Pointer Table
Sorted Array

Approximate Set

$$f_{\bar{0}}(x) = true \Big\} \text{ technically correct}$$

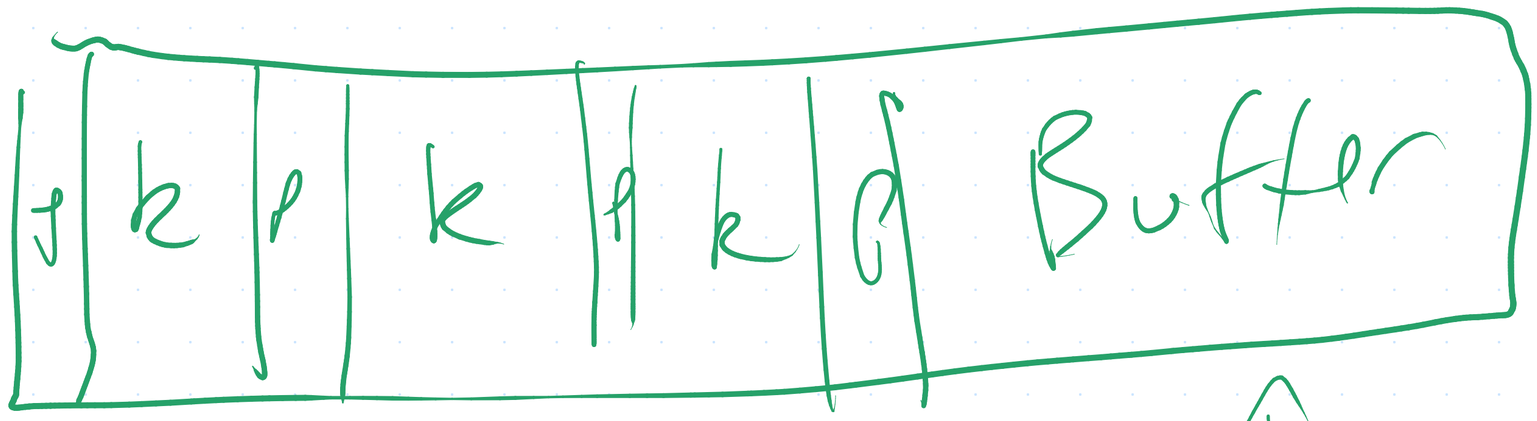
Not enough tests

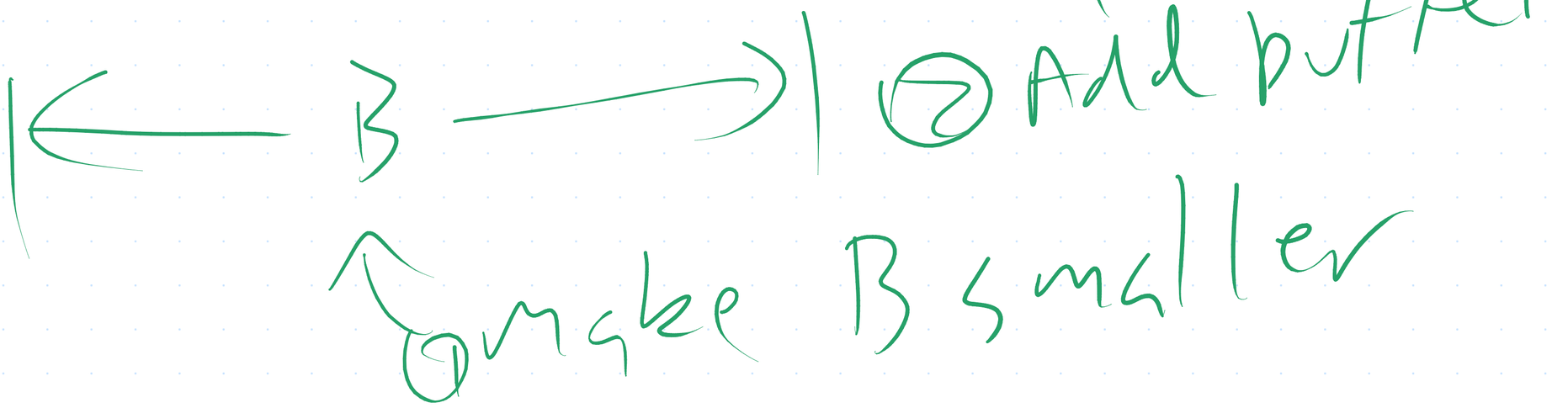
Bit vector filters

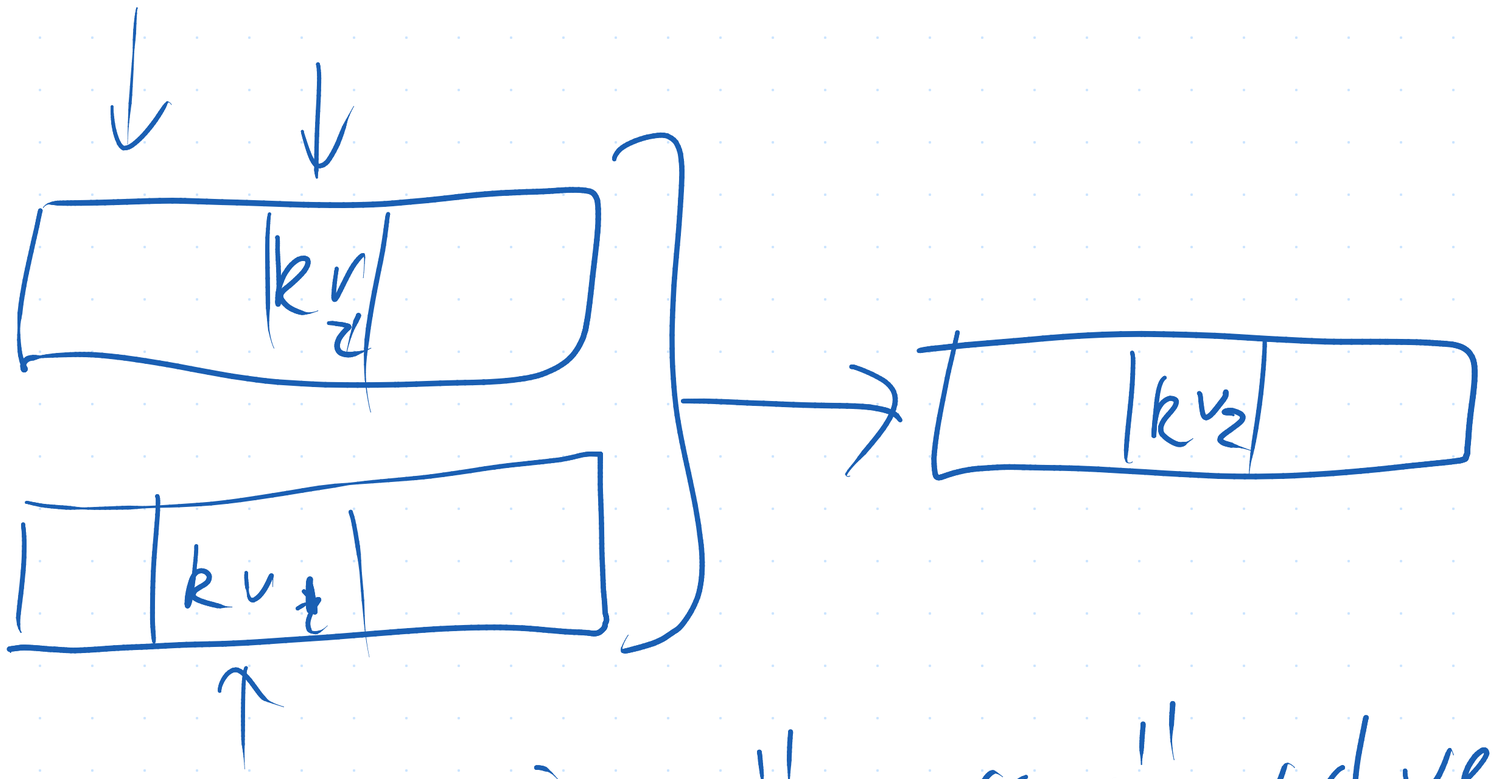
False positive of file

# of hits per object

Directory page







merging "merges" values for keys

later

$V_1 \circ V_2 \rightarrow V_2$ wins

$V_1 \circ \text{NULL} \rightarrow$ key skipped

Variant 1

write
fill buffer
sort buffer
merge with sorted array on disk] O(N) IOs

read
Access 1 sorted array] O(log N)

Variant 2

write
fill buffer
sort buffer
write buffer to disk] O(B)

read
- Access $\frac{N}{B}$ sorted arrays
- $\rightarrow O\left(\frac{N}{B} \log_B B\right)$

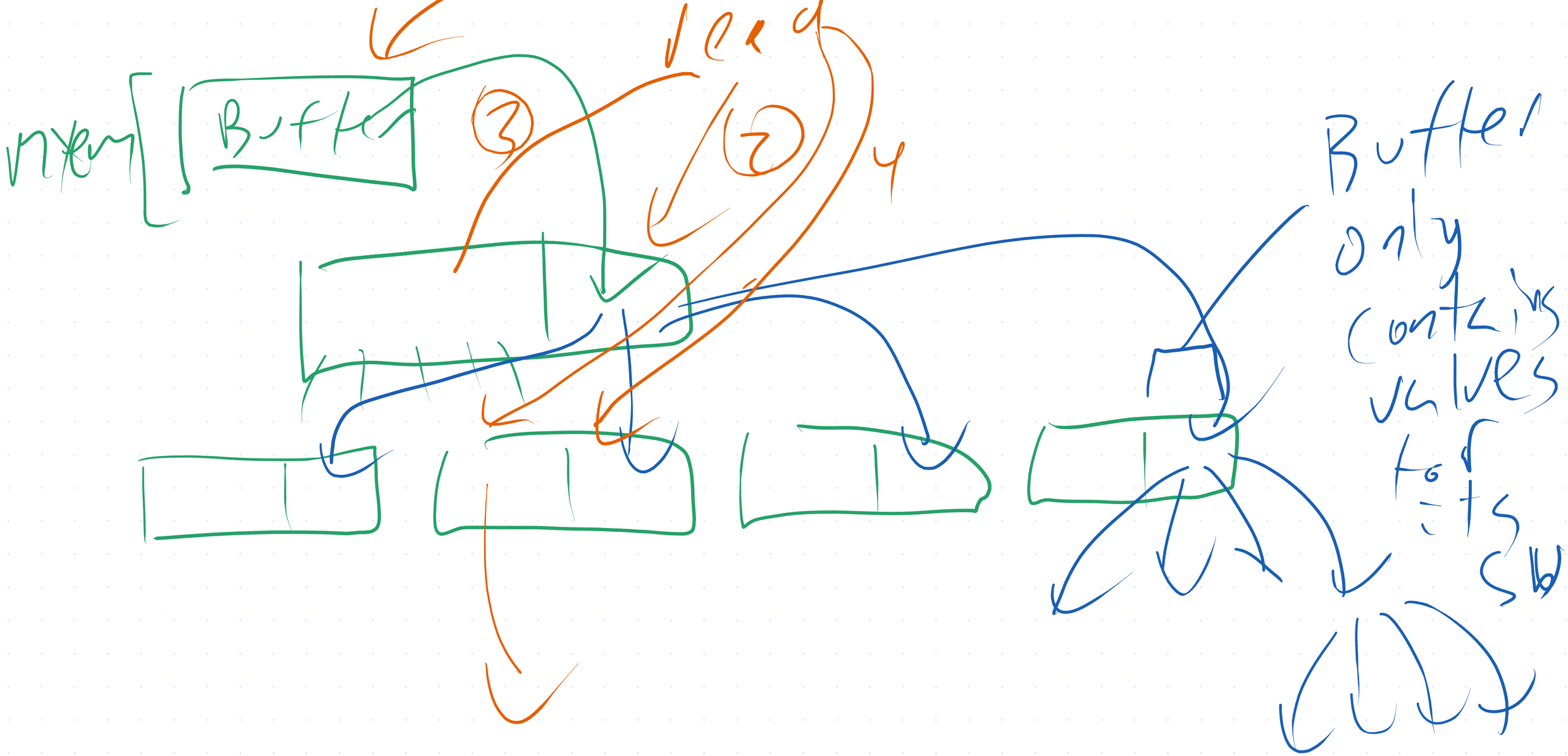
```
{
    bool a;
    bool b;
}

on insert(x)
    if h(x) mod 2 = 0
        a = true
    else
        b = true

on lookup(x)
    if h(x) mod 2 = 0
        return a
    else
        return b
```

N objects

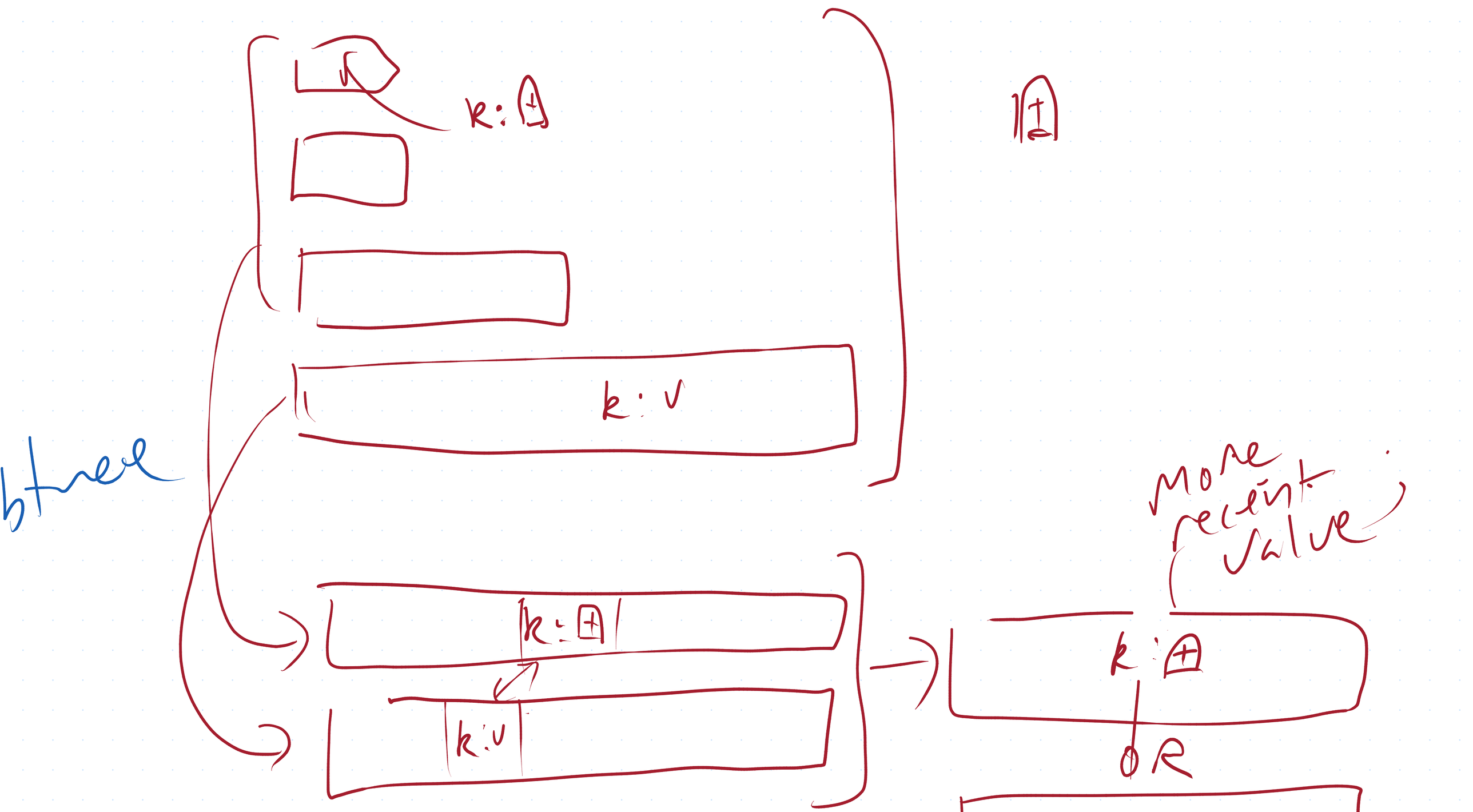
B size of bit vector

optimal # of h, h per object

$f\left(\frac{N}{B}\right)$

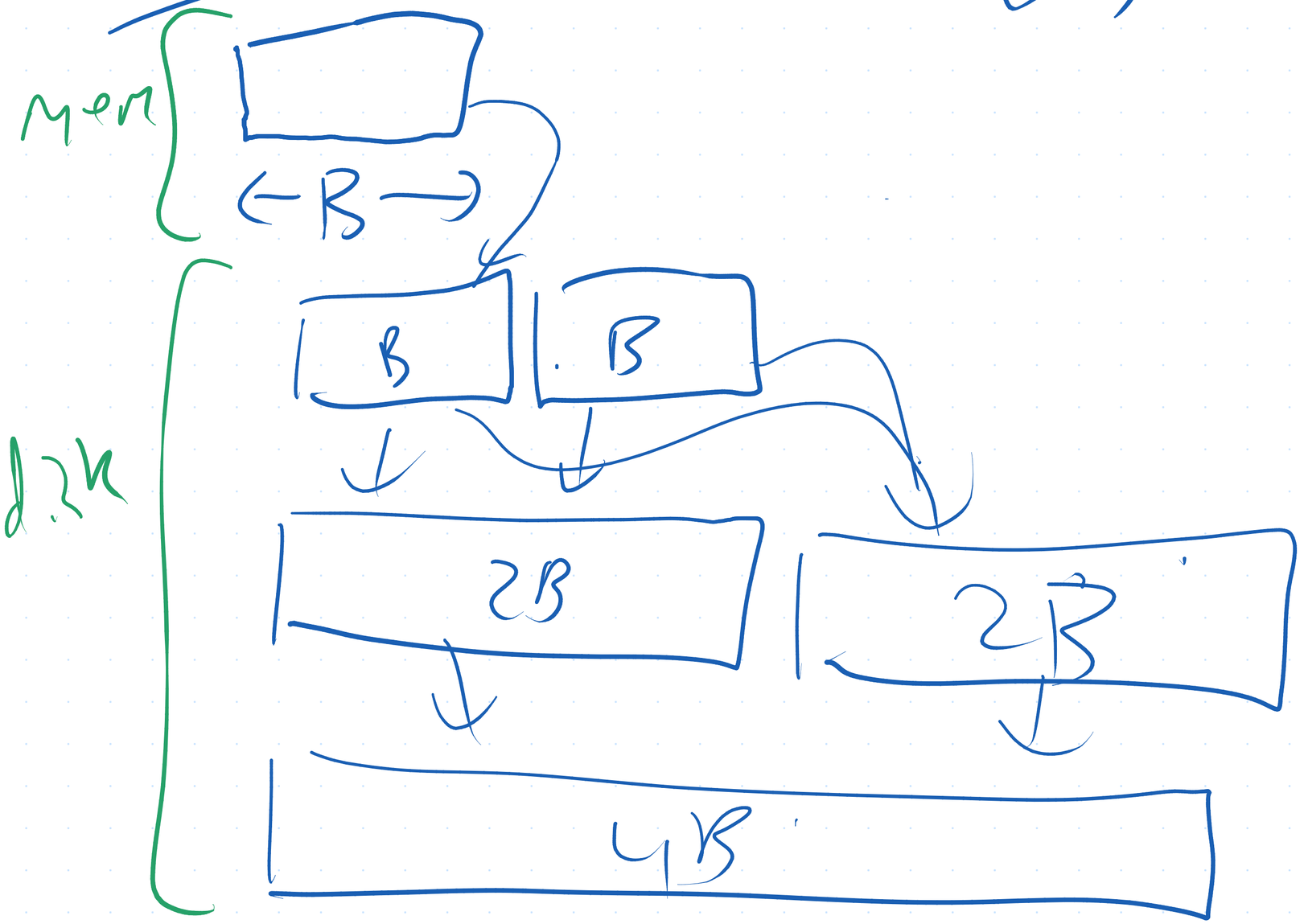


Beta-Epsilon Tree


k: t
k: v
btree
k: ⊞
k: v
k: ⊞
OR
more recent value
No key if last layer

Log Structured Merge Tree Index

Variant 3



Layer 1

layer 2

layer 3

{

bool bits[B] = [false]

}

on insert(x)

bits[h(x) % B] = true

on lookup(x)

return bits[h(x) % B]

→ insert(42)

h(42) = 3

↑ B.t 3 must be T after insert(42)

Size of bitvector

10 bits per object

→ 1% fp rate

O(N) bits?

↳ but very small constant

Analysis

Layer 1	B
Layer 2	2B
Layer 3	4B
Layer 4	8B
Layer i	$2^{i-1}B$

$$\sum_{i=1}^{k} 2^{i-1}B$$

$$= O(2^k B)$$

$$N = 2^k B$$

$$\frac{N}{B} = 2^k$$

$$k = \log\left(\frac{N}{B}\right)$$

to store N records $\log\left(\frac{N}{B}\right)$ layers

{

bits[R][B] = [false]

}

$h_i(x)$ for $i \in [0, R)$

on insert(x)

for $i \in [0, R)$

bits[i][$h_i(x)$ % B]

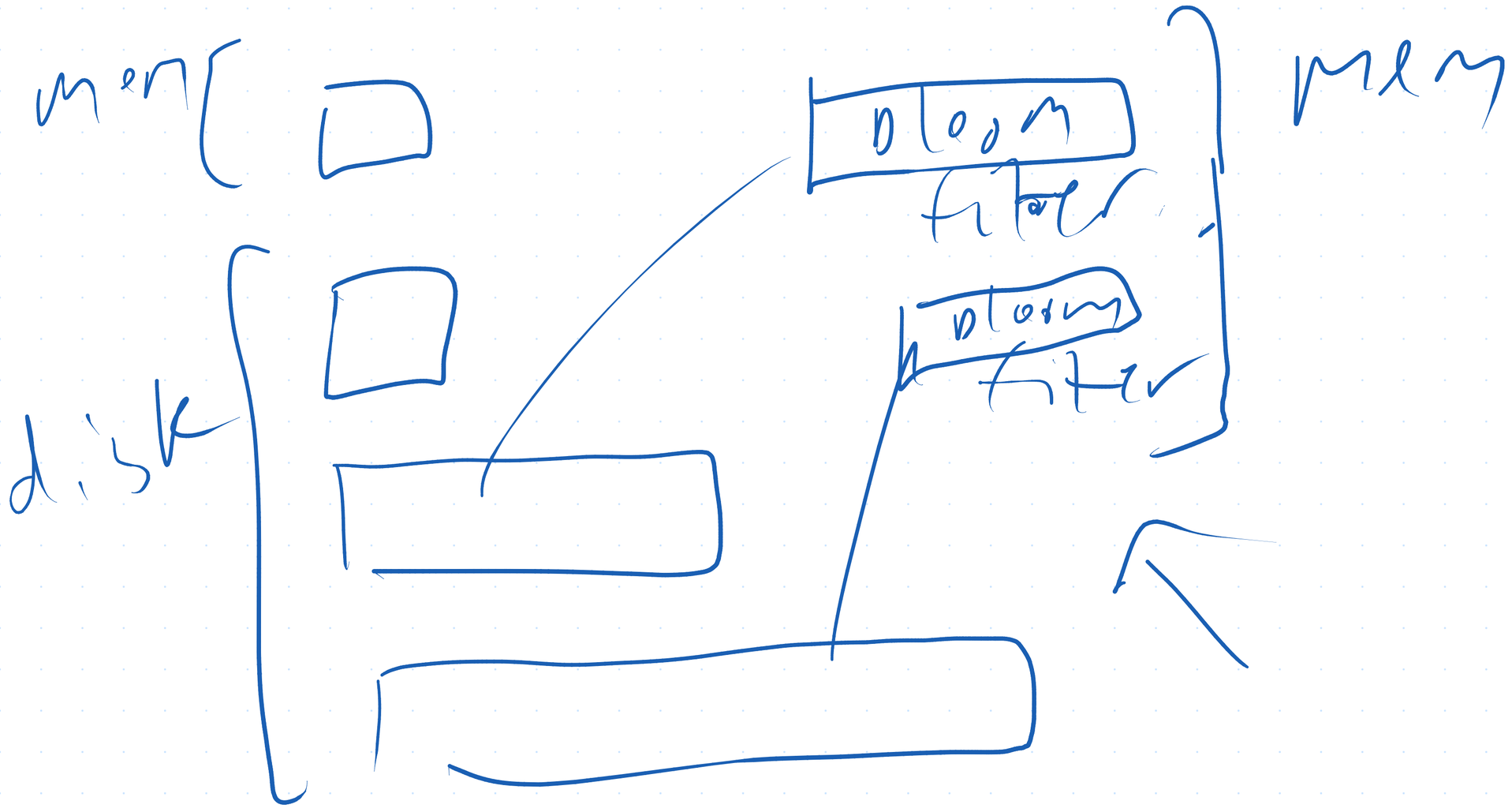
on lookup(x)

for i in [0, R)

if !bits[i][h(x)]

return false

return true


mem
bloom
filter
bloom
filter
disk

read

$$O\left(\log\left(\frac{N}{B}\right)\right) \cdot O(\log N) = O\left(\log^2(N)\right) \quad \text{cost per read}$$

$$= \left(\log(N) - \log(B)\right) \cdot \log(N)$$

$$\left(\log^2(N) - \log(B)\log(N)\right)$$

↑ read amplification

write

Observation: Each record is copied k times (at most)

$\uparrow \log\left(\frac{N}{B}\right)$ ← write amplification

{

bits [B]

}

Bloom Filter

on insert(x)

for i in (0, k)

bits[h(x)] = true

Bloom [0, n)

X

A, X

B, X

C, ∞

F, 1

(B2, C2, D2, F2)

Y

B2

C2

D2

F2

Bloom(B, C, DF)

⟨A, B, C⟩

⟨B1, B2⟩

⟨C1, C2⟩

⟨B1, B2⟩

⟨C1, C2⟩

N records, each record copied at most $\log_{\frac{M}{B}}\left(\frac{N}{B}\right)$ times

↳ $O\left(N \log_{\frac{M}{B}}\left(\frac{N}{B}\right)\right)$, so per insertion cost is $\log_{\frac{M}{B}}\left(\frac{N}{B}\right)$ writes (amortized)

Bloom filter

add(x) → Inserts x into the filter

get(x) → If x was added return true always
otherwise return false with high prob

{ buffer: (lock; B)
  data: Vec(File)
}

on insert(datum)
  add to buffer
  if buffer full
    write out to disk
    while there are 2 arrays at a level
      merge into next level down