Dynamic Indexing with Logarithmic Merging

Alfan F. Wicaksono
Fakultas Ilmu Komputer, Universitas Indonesia

Dynamic Indexing using Logarithmic Merging

We can do better than $O(T^2/n)$ by introducing $\log_2(T/n)$ indexes $I_0, I_1, I_2, \ldots$ of size $2^0 \times n, 2^1 \times n, 2^2 \times n, \ldots$

Postings percolate up this sequence of indexes and are processed only once on each level.

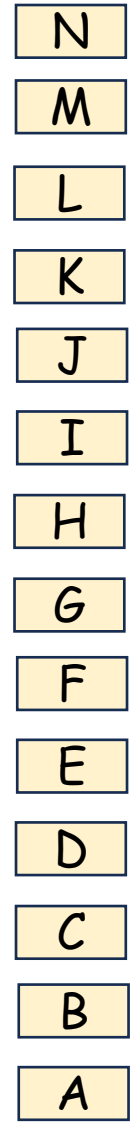
LMERGEADDTOKEN(indexes, Z_0, token):

```
Z_0 ← MERGE(Z_0, {token})
if | Z_0 | = n then
    for i ← 0 to ∞ do
        if I_i ∈ indexes then
            Z_{i+1} ← MERGE(I_i, Z_i)        (Z_{i+1} is a temporary index on disk)
            indexes ← indexes − {I_i}
        else
            I_i ← Z_i                        (Z_i becomes the permanent index I_i)
            indexes ← indexes ∪ {I_i}
            BREAK
    Z_0 ← ∅
```

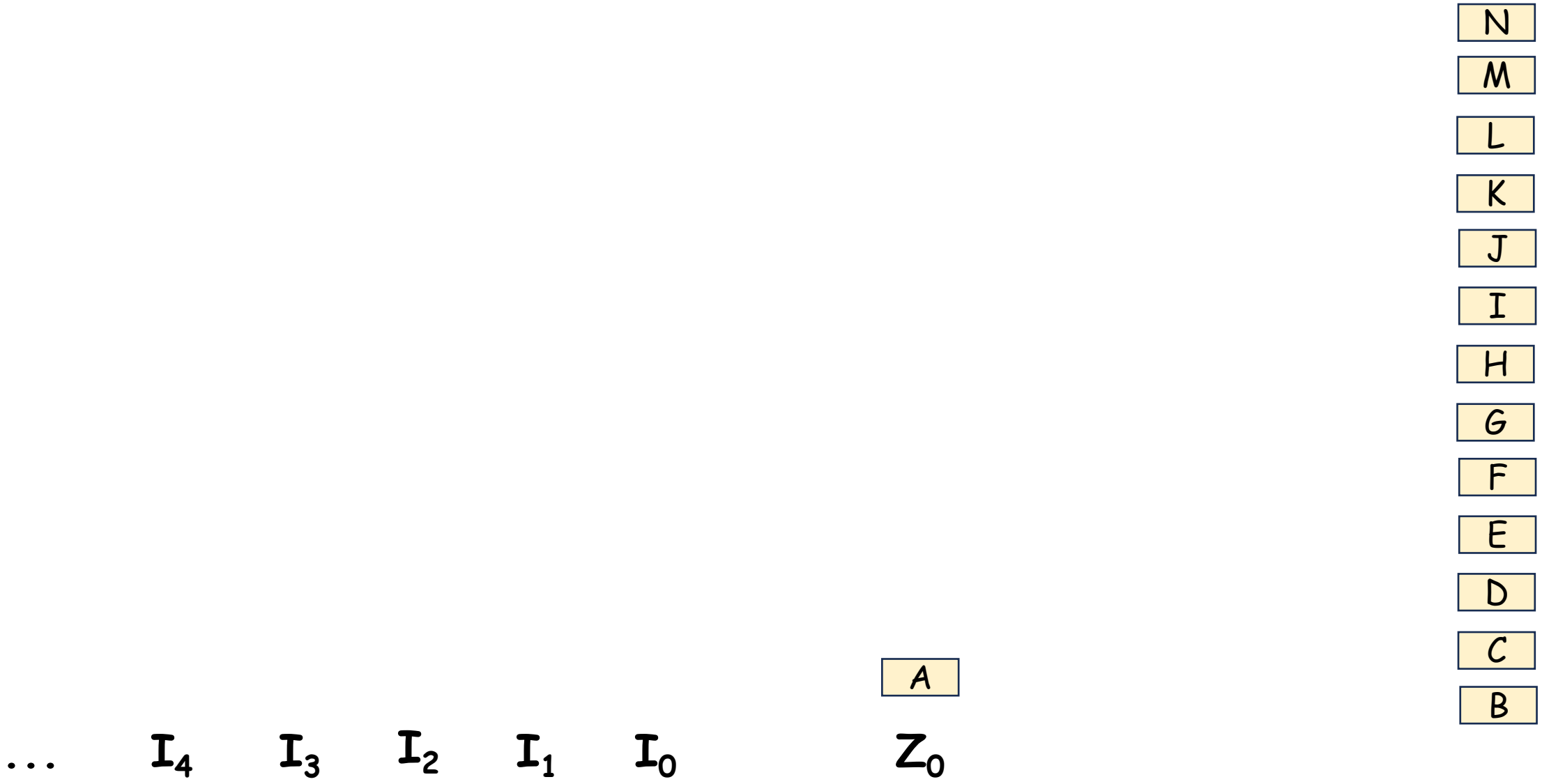
LOGARITHMICMERGE():

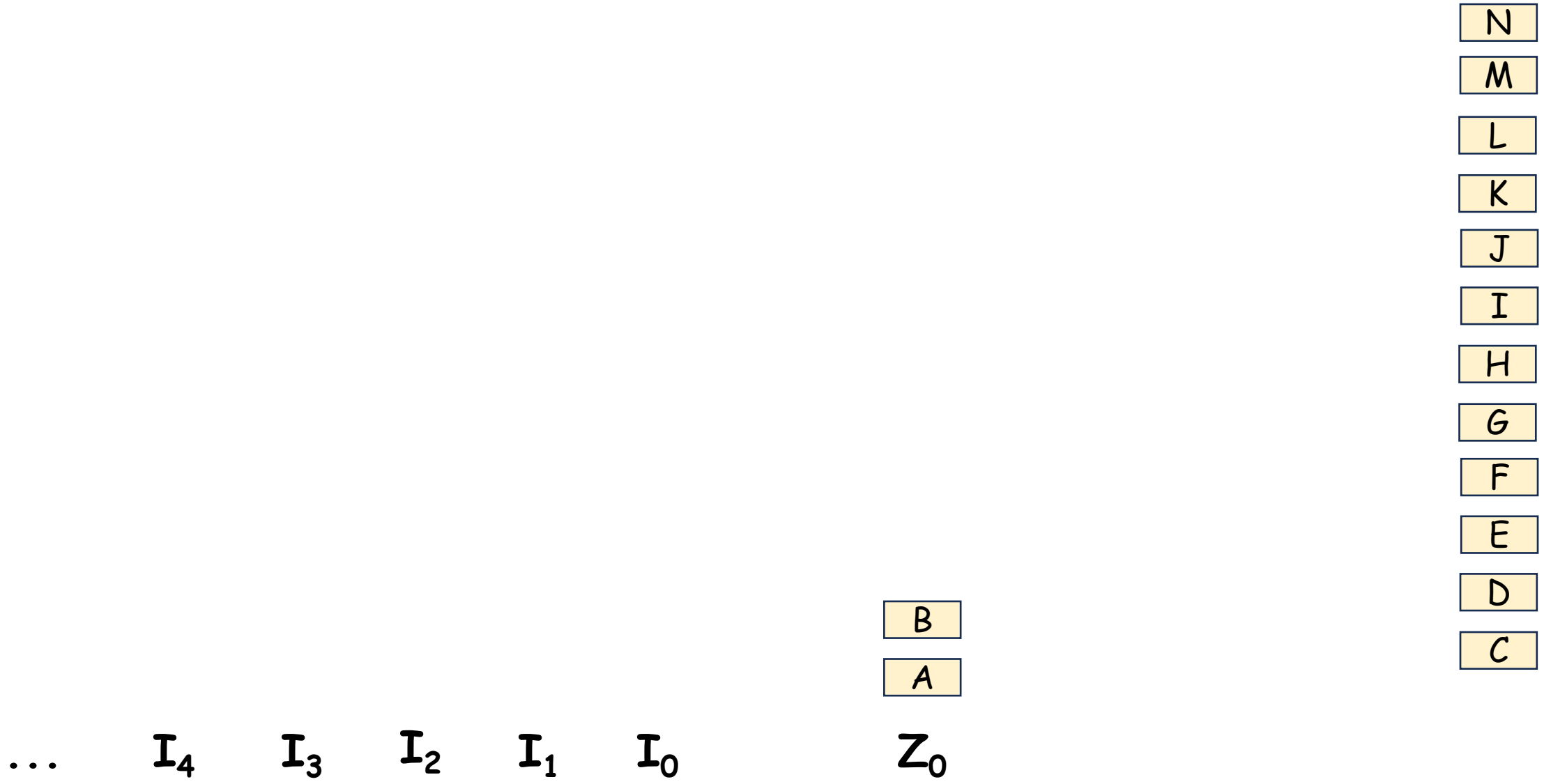
```
Z_0 ← ∅                                      (Z_0 is the in-memory index.)   Ini aux index)
indexes ← ∅
while true do
    LMERGEADDTOKEN(indexes, Z_0, GETNEXTTOKEN())
```

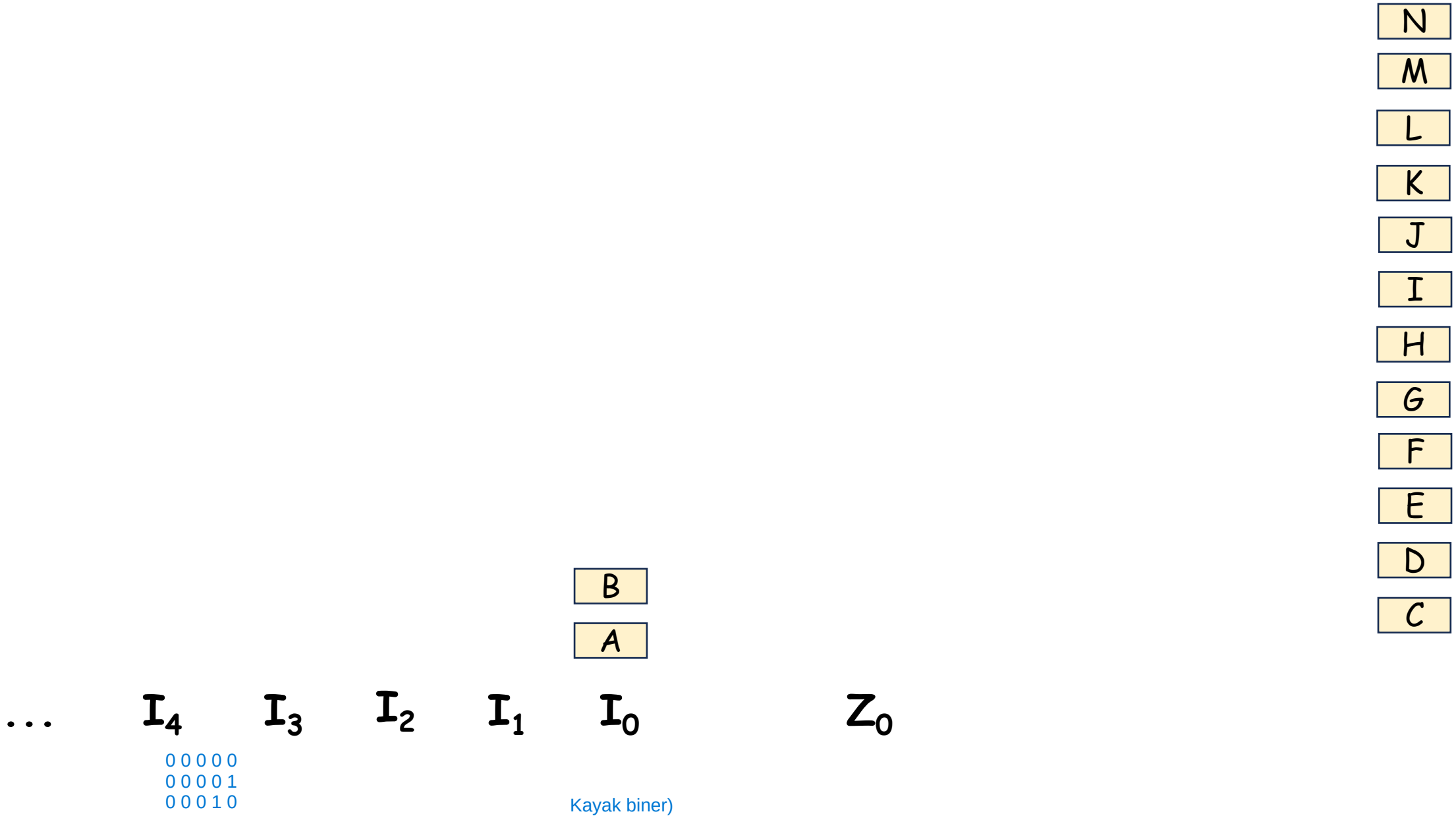
Ukuran Aux Index (n) = 2

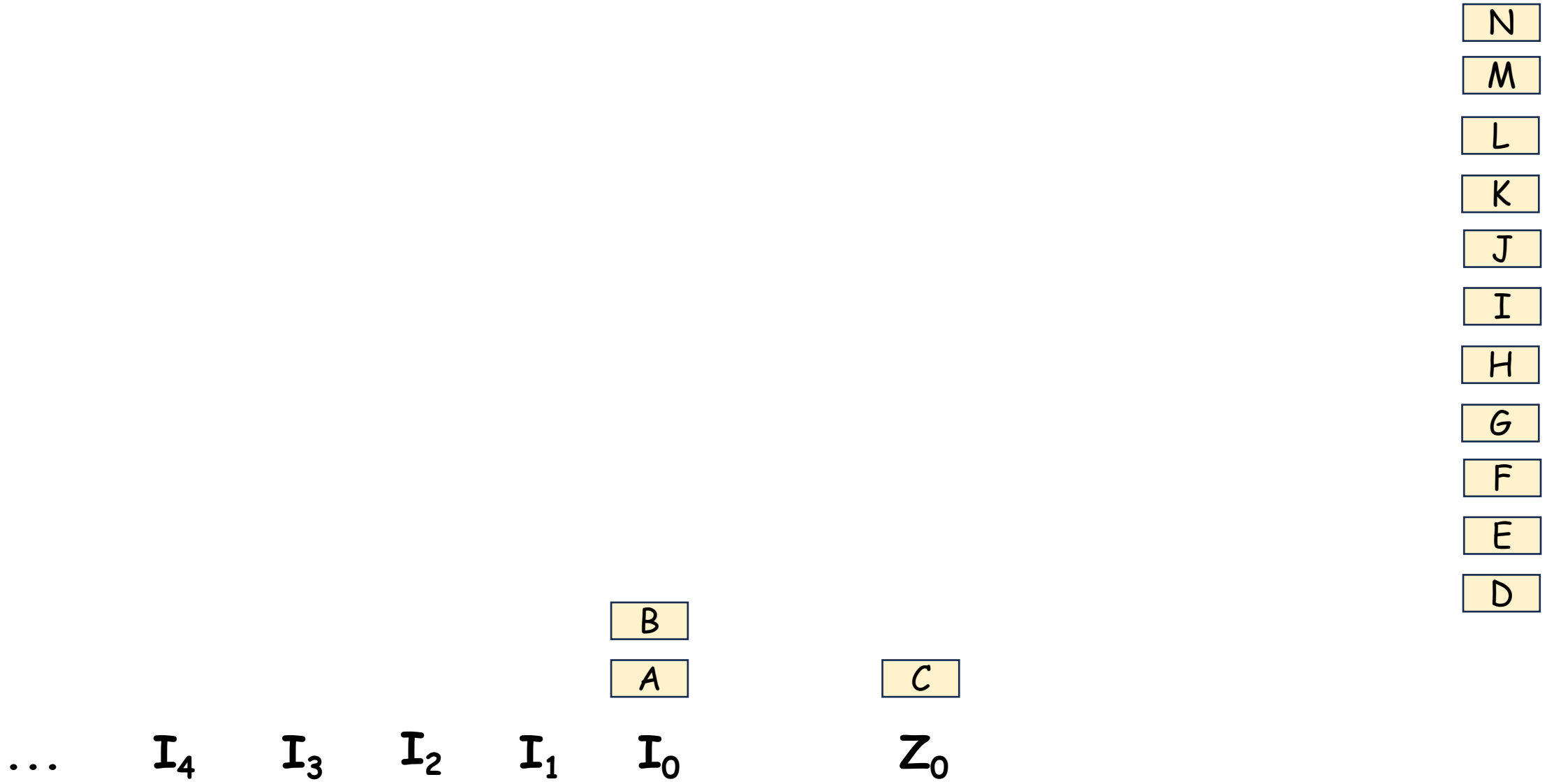


... I_4 I_3 I_2 I_1 I_0 Z_0

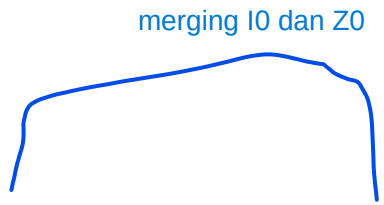





N
M
L
K
J
I
H
G
F
E
D
C
B
A
... I_4 I_3 I_2 I_1 I_0 Z_0
0 0 0 0 0
0 0 0 0 1
0 0 0 1 0
Kayak biner)


N
M
L
K
J
I
H
G
F
E
D
B
A
C
...
I_4
I_3
I_2
I_1
I_0
Z_0

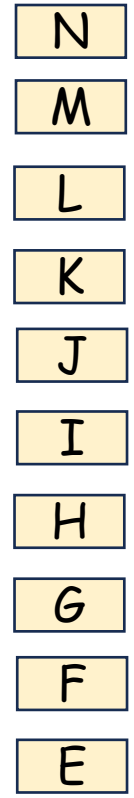
N
M
L
K
J
I
H
G
F
E

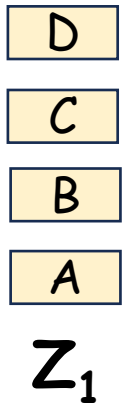


B
A

D
C

... I_4 I_3 I_2 I_1 I_0 Z_0





Merge

... I_4 I_3 I_2 I_1 I_0 Z_0

(2^4) * n

2 ^ 3 * n

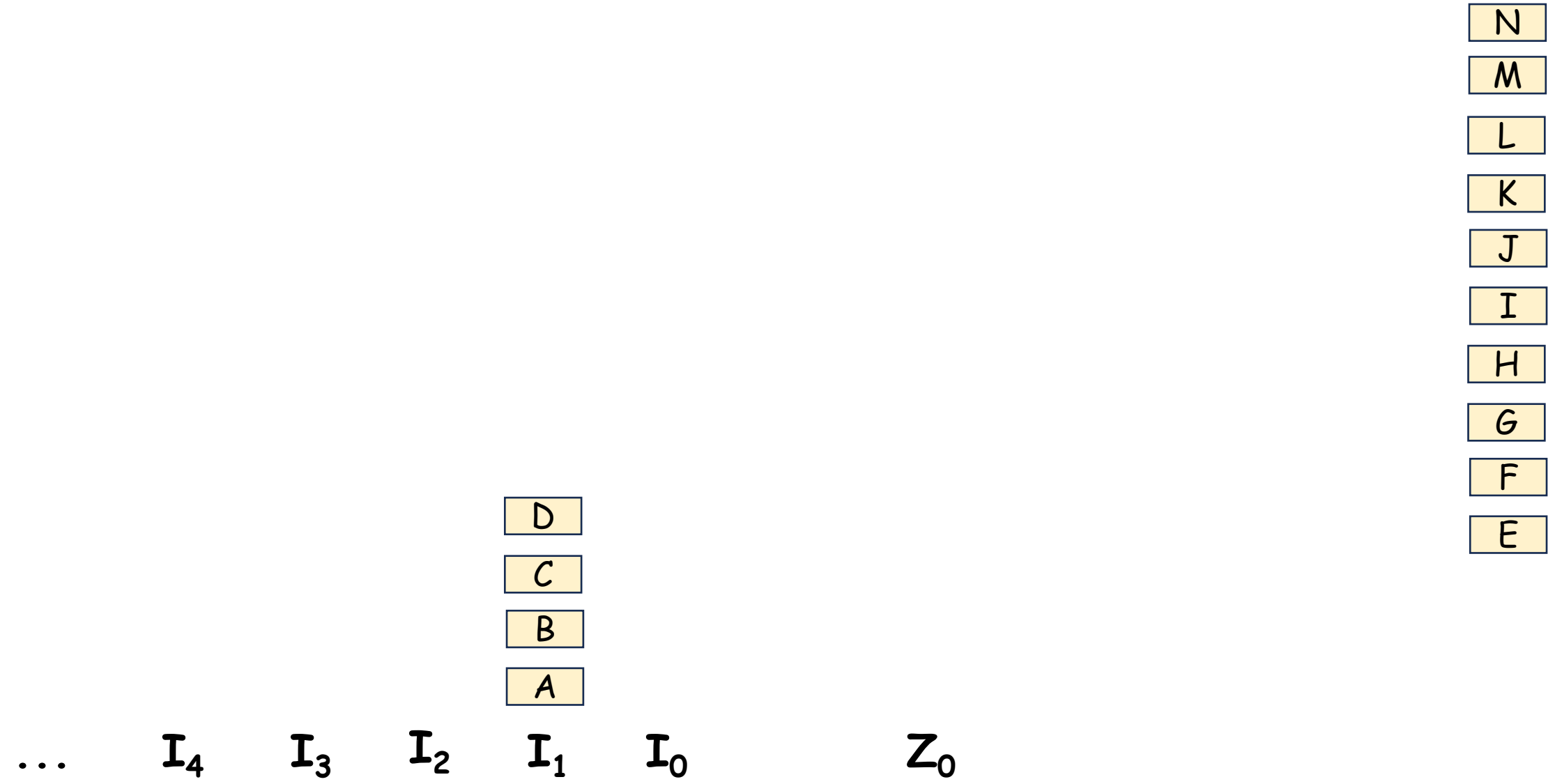
2^2 * n

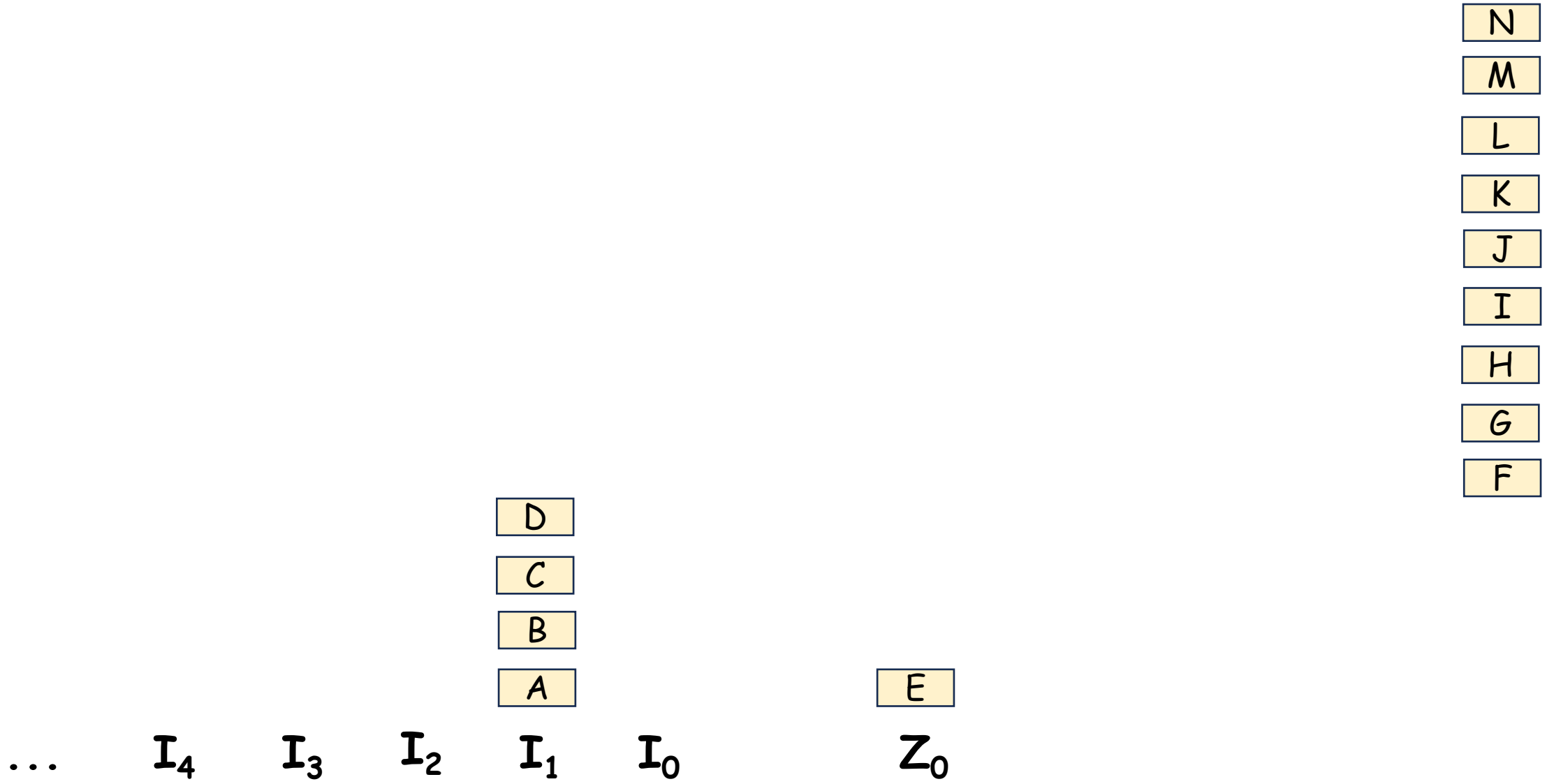
2^1 * n

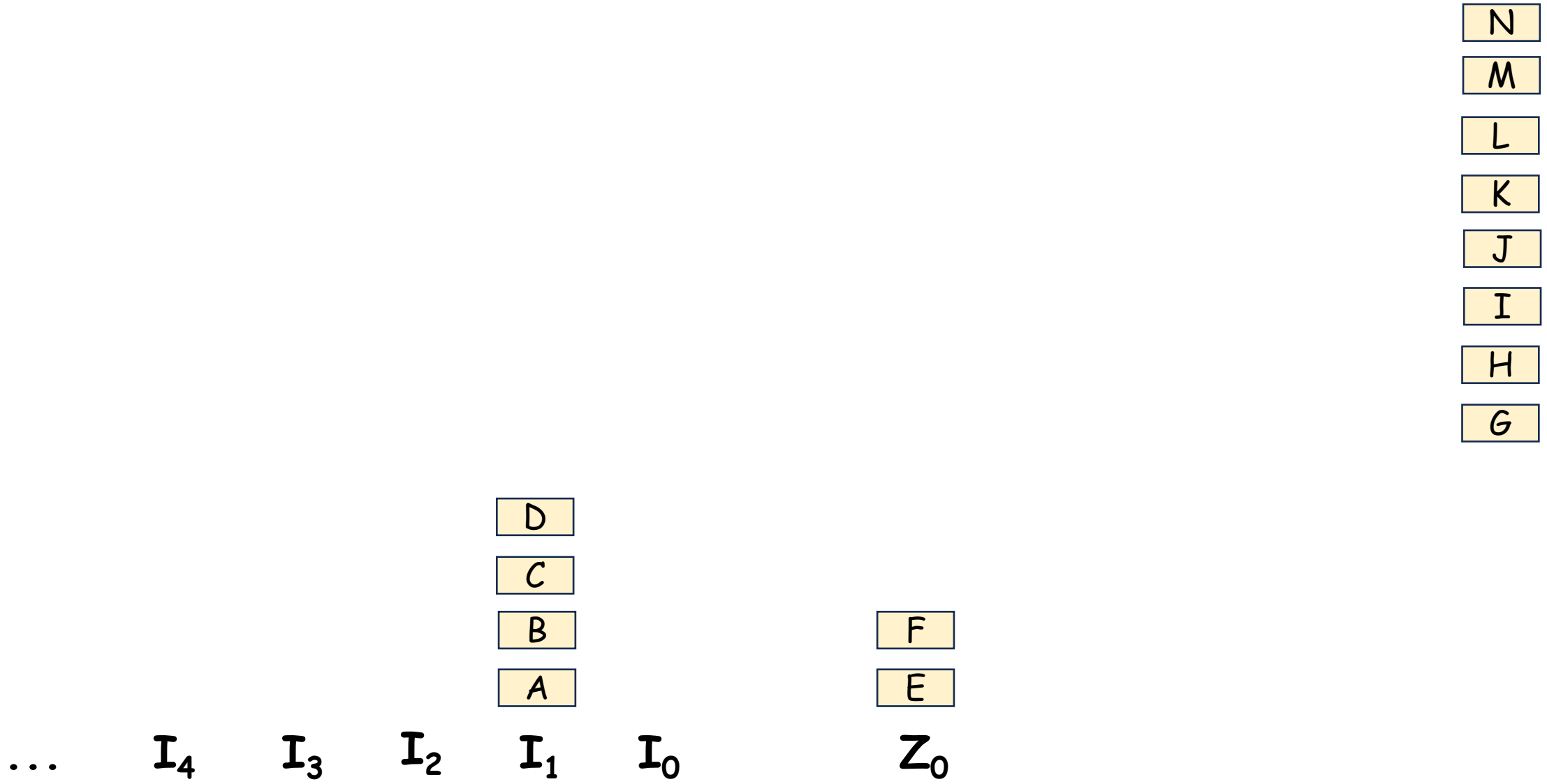
Maksimal

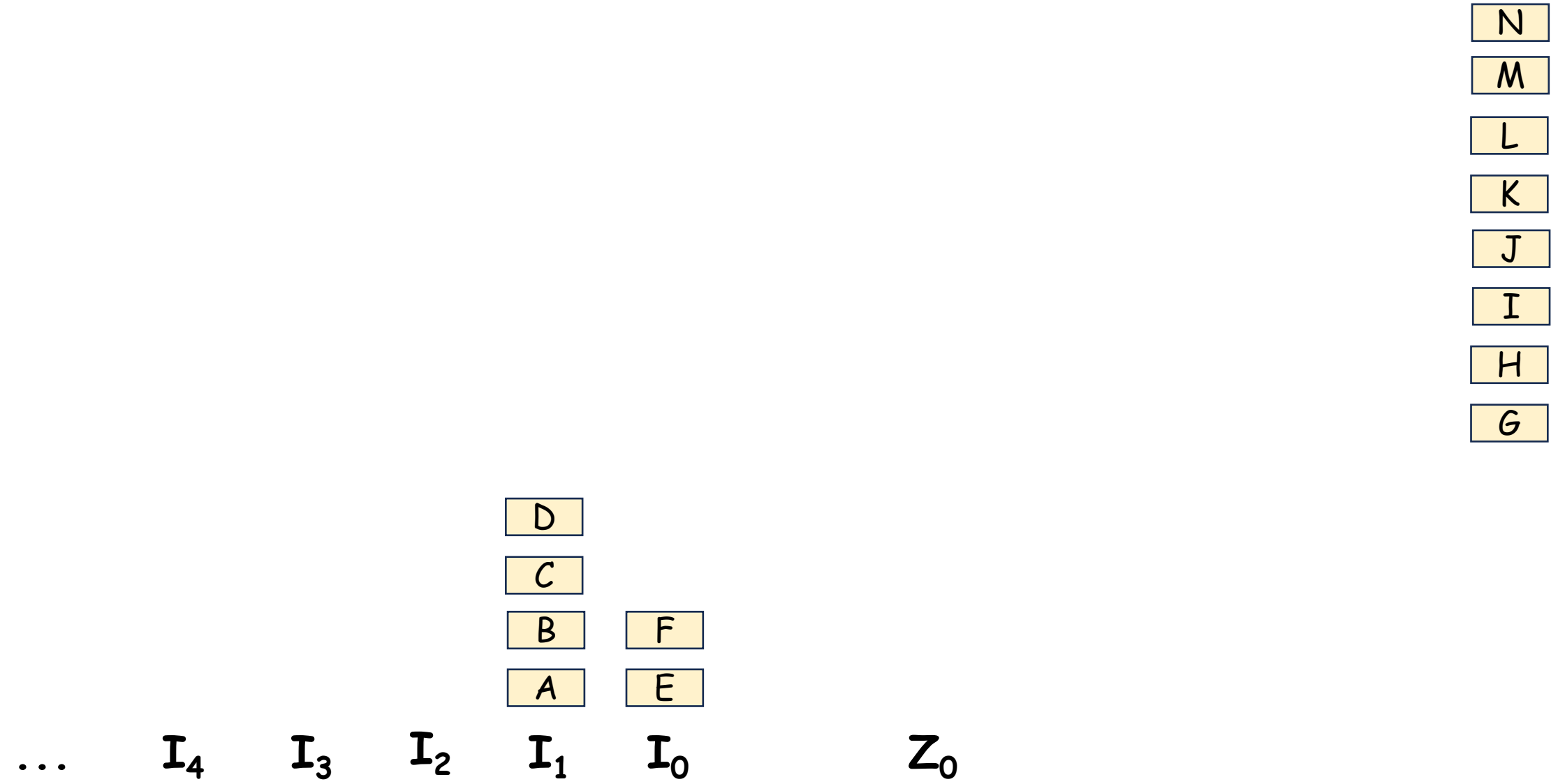
2^0 *n

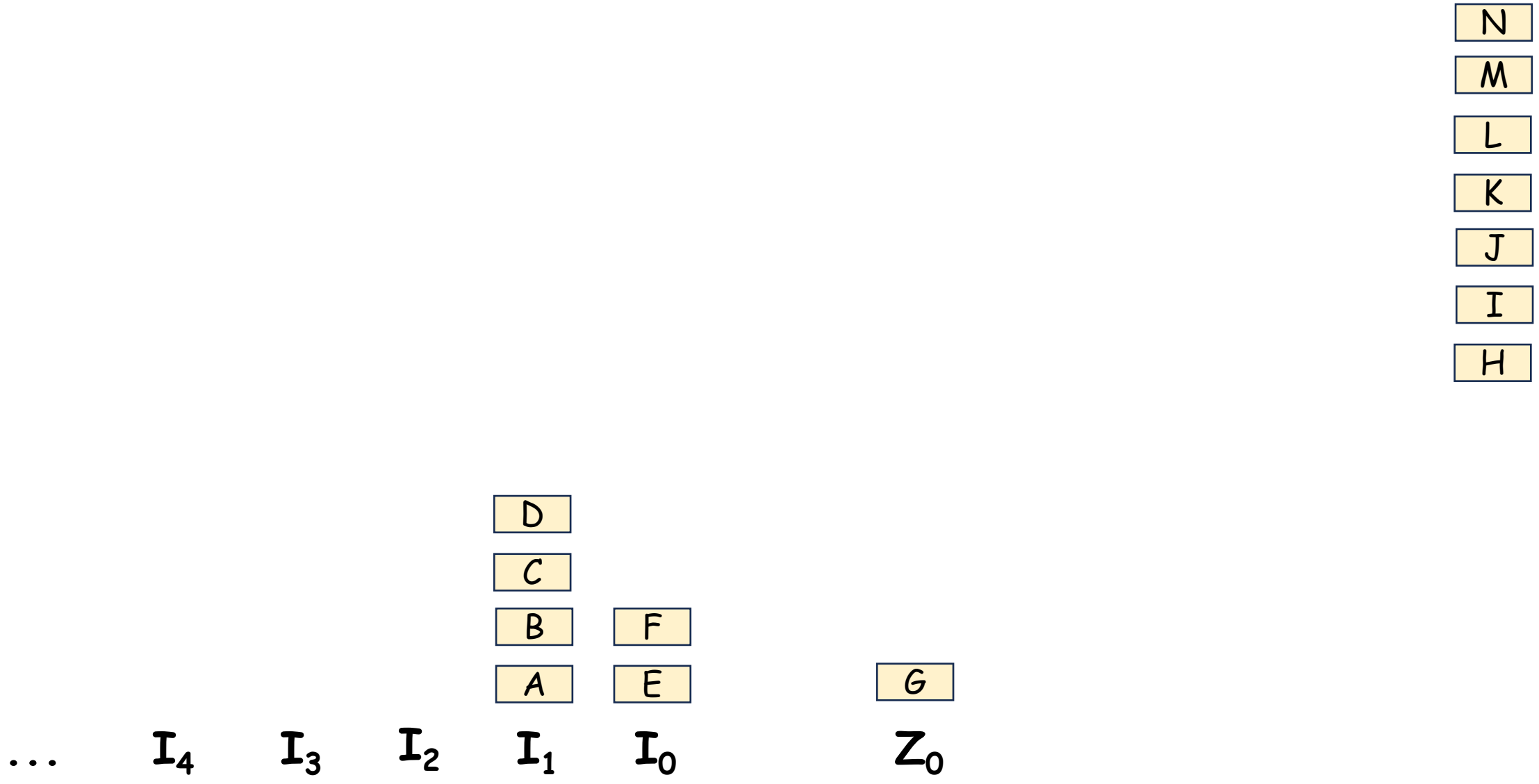
Maksimal n = 2

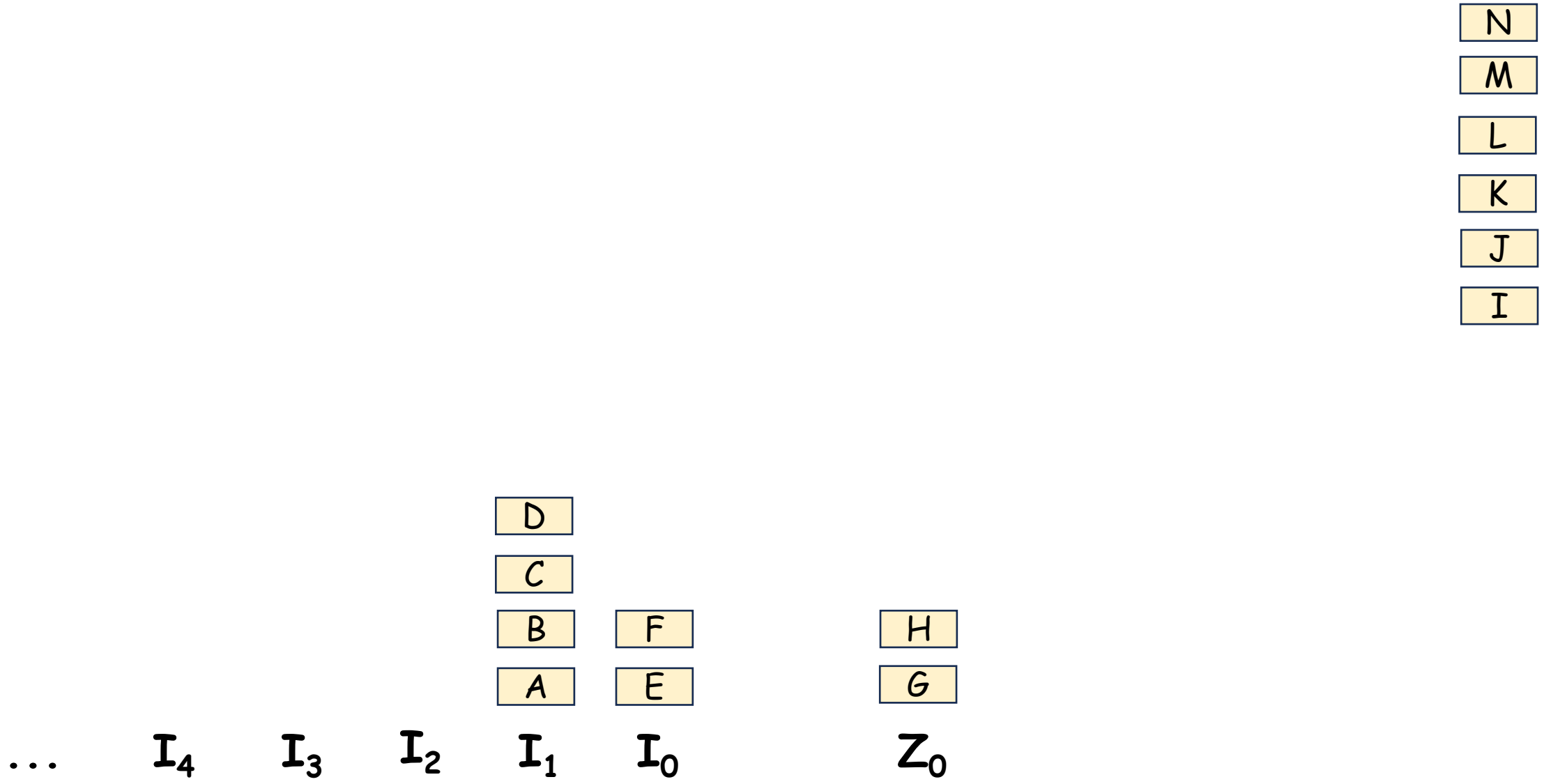


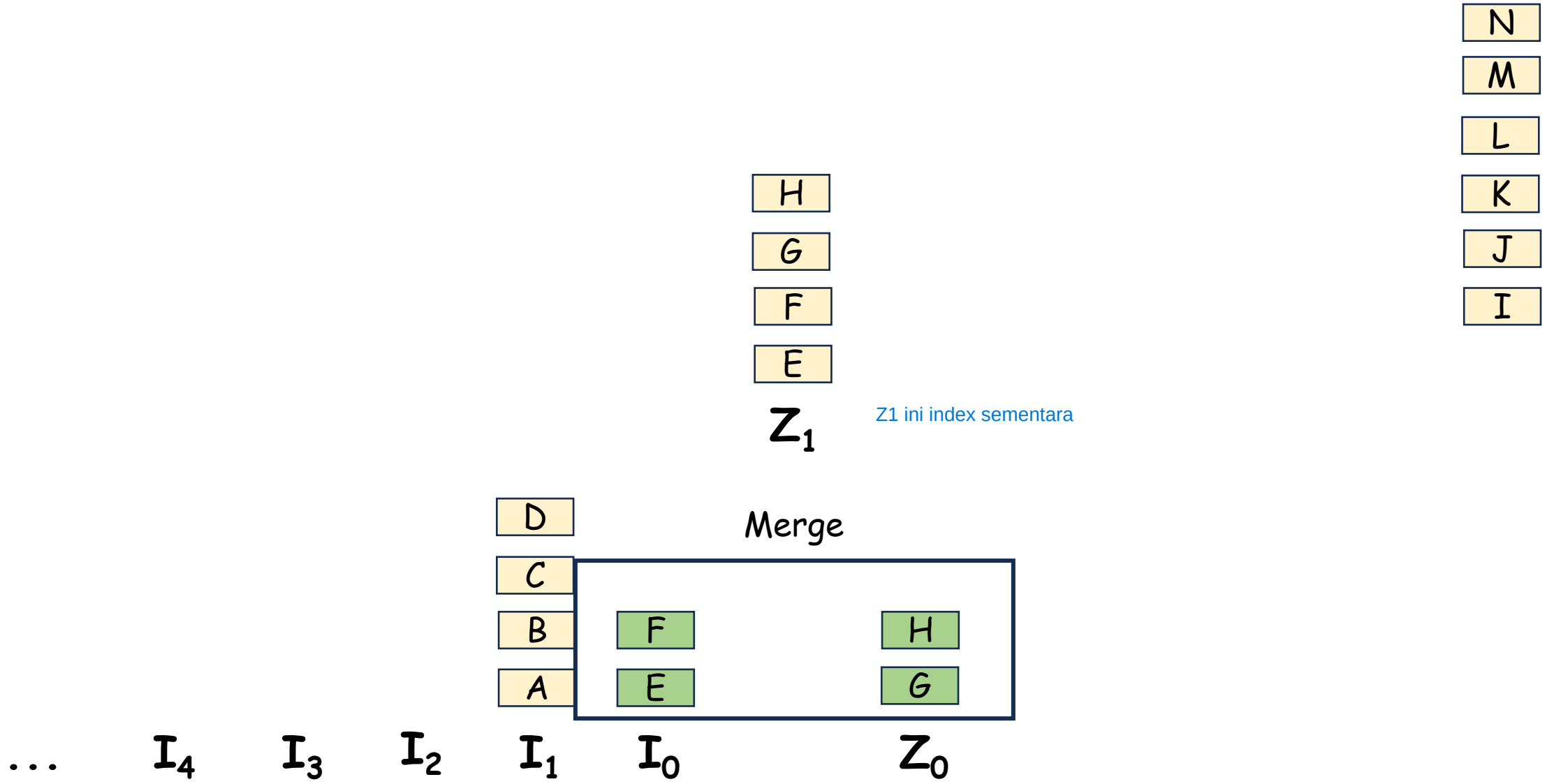


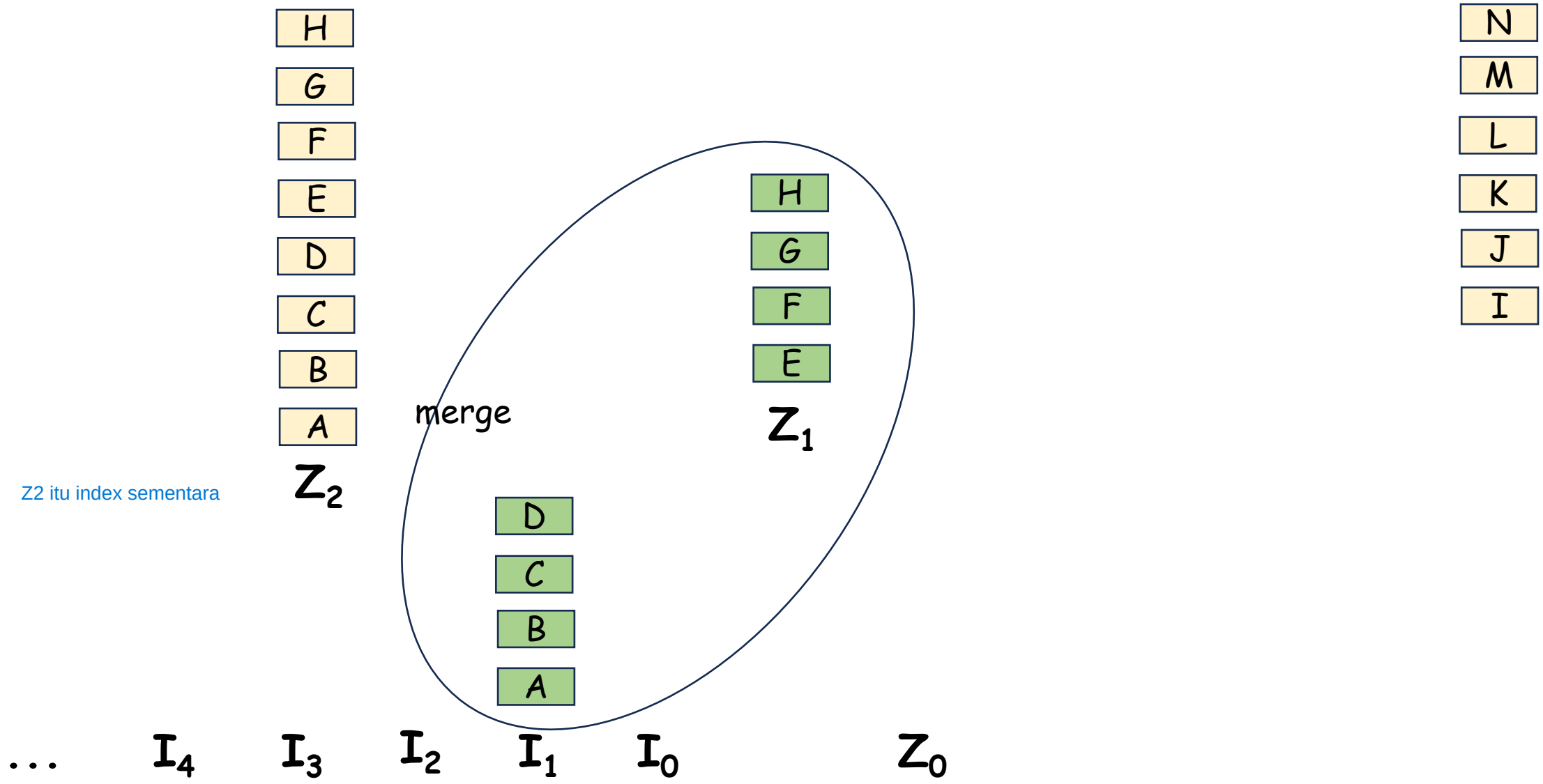


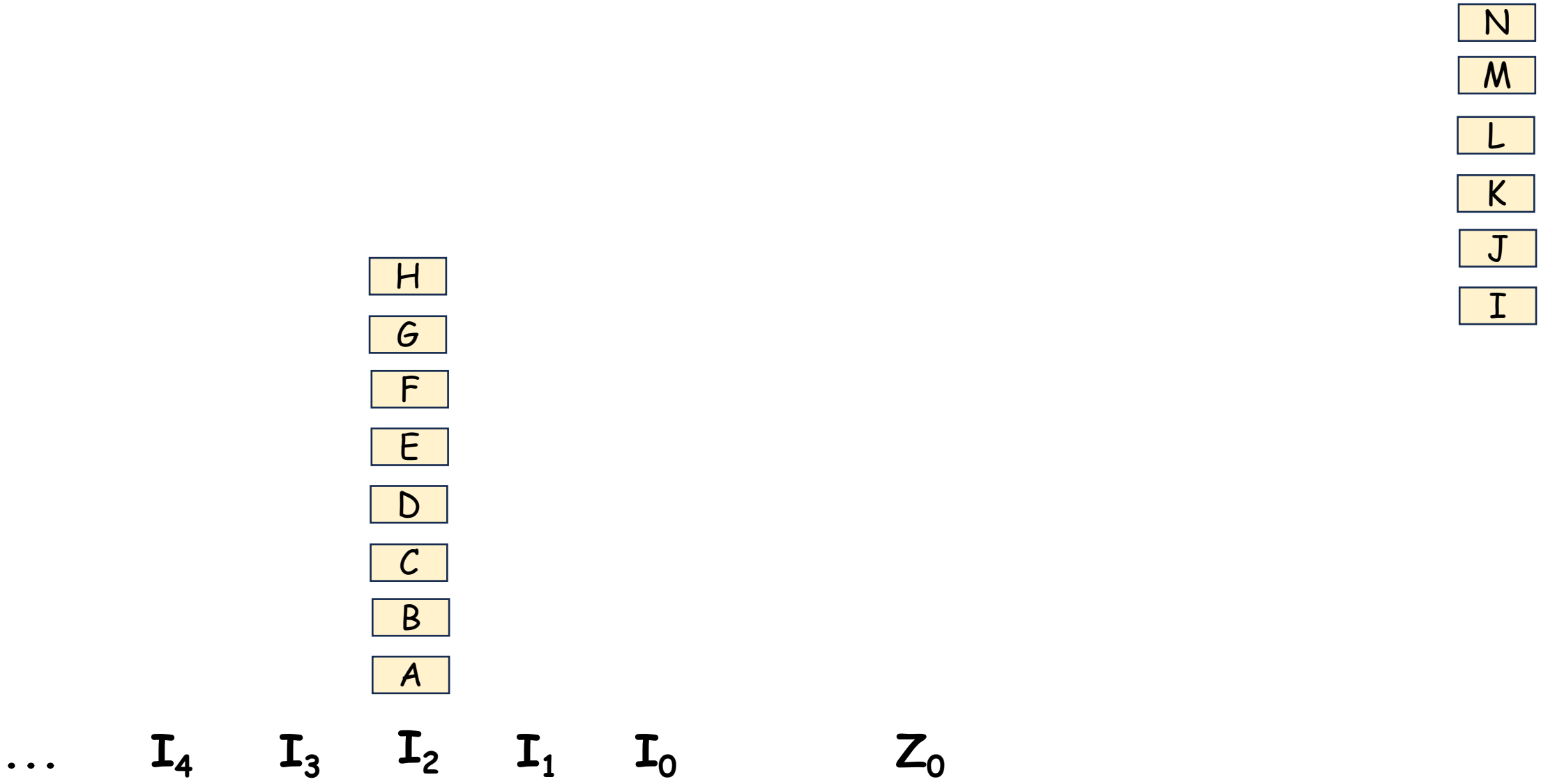


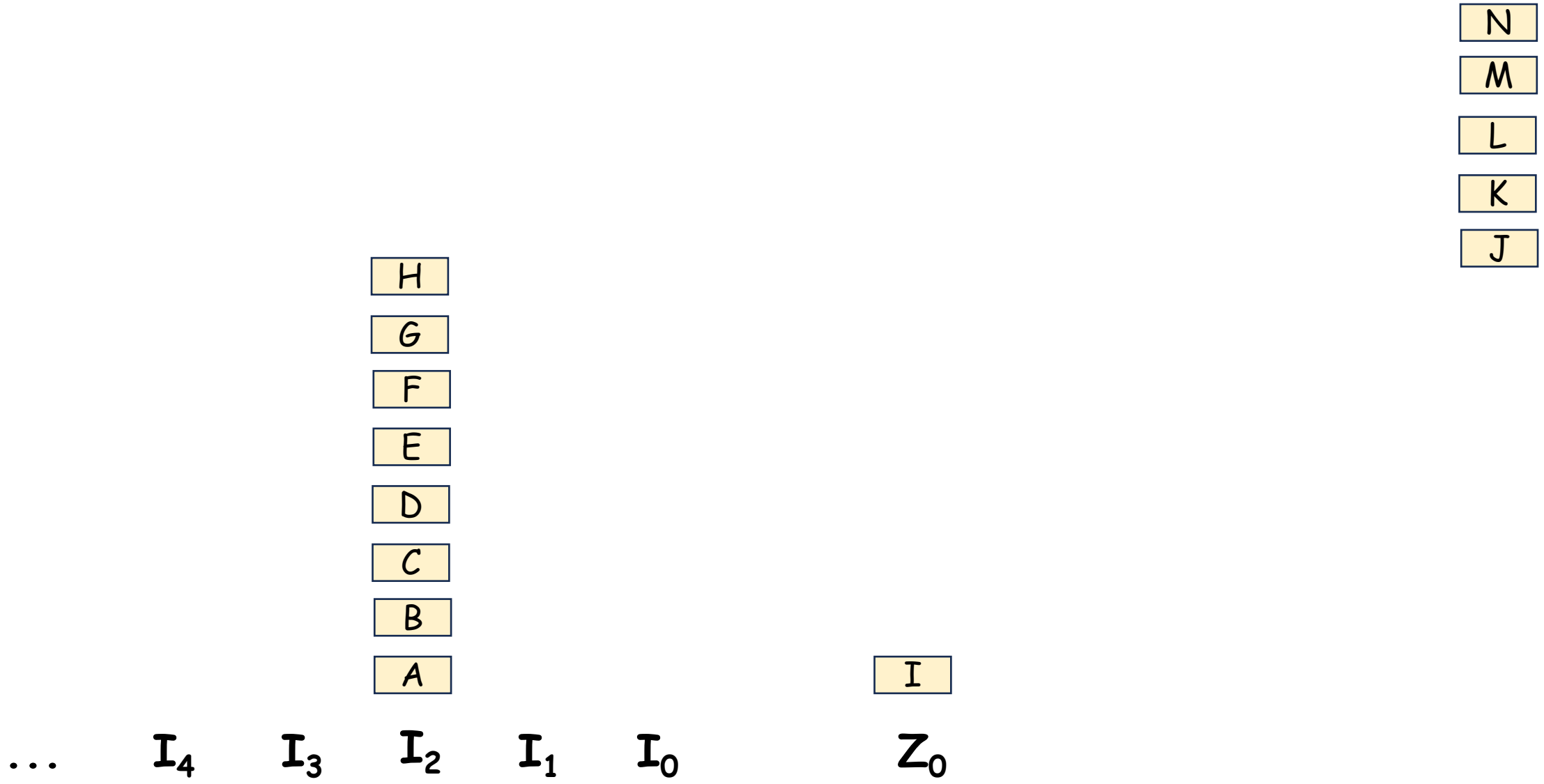


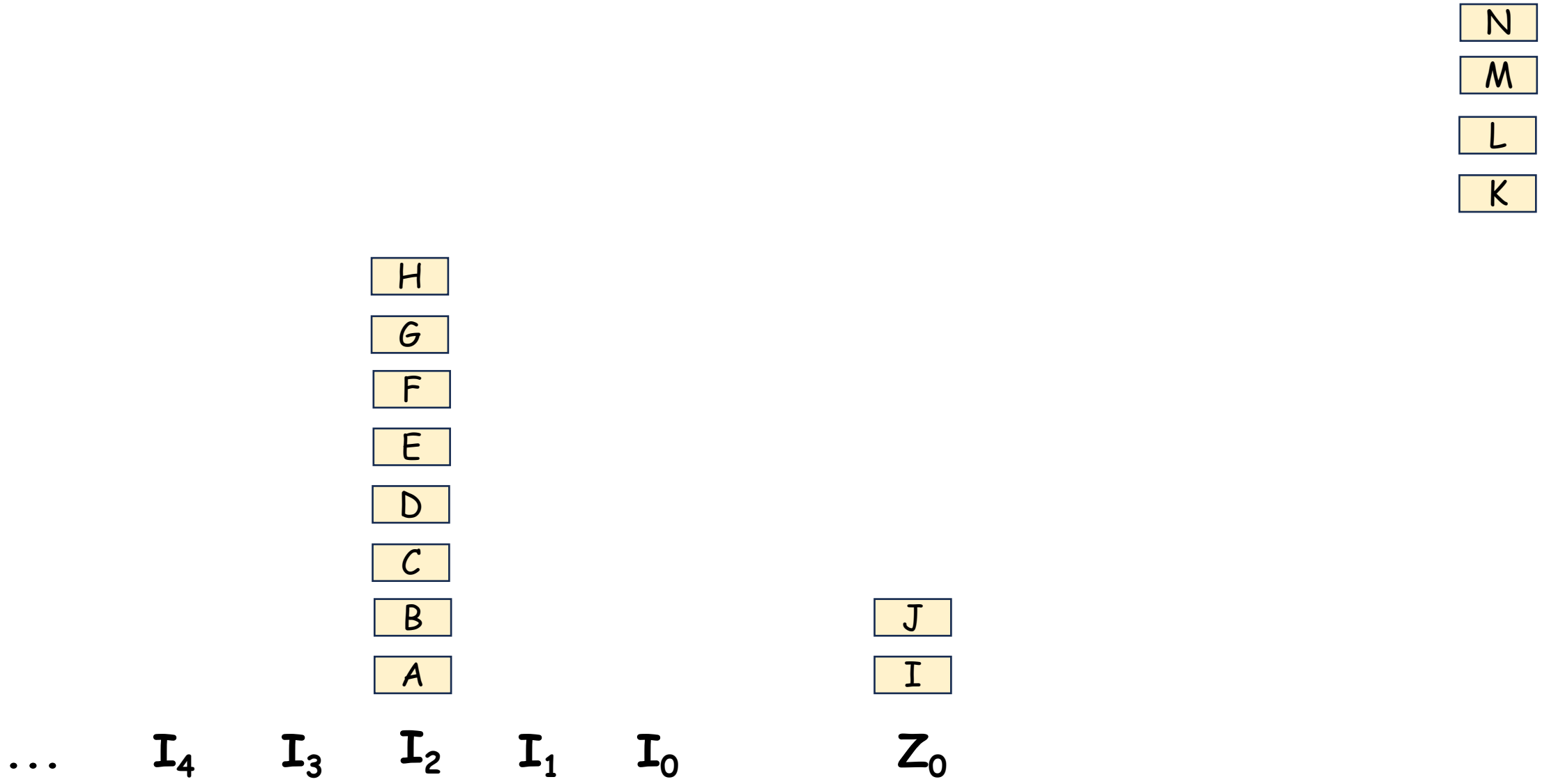


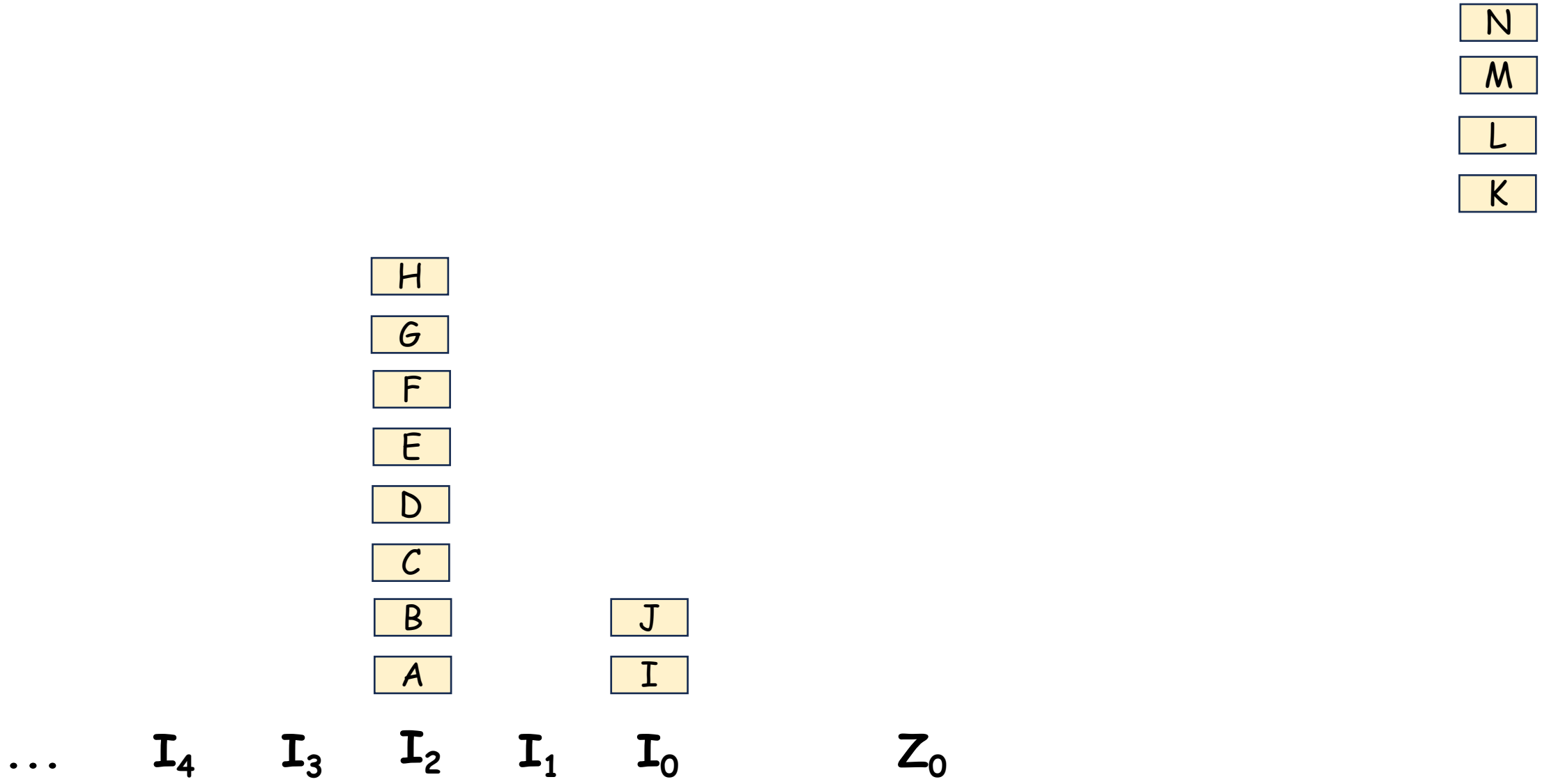

N
M
L
K
J
I
H
G
F
E
Z1 ini index sementara
D
C
B
A
Merge
F
E
H
G
...
I4
I3
I2
I1
I0
Z0

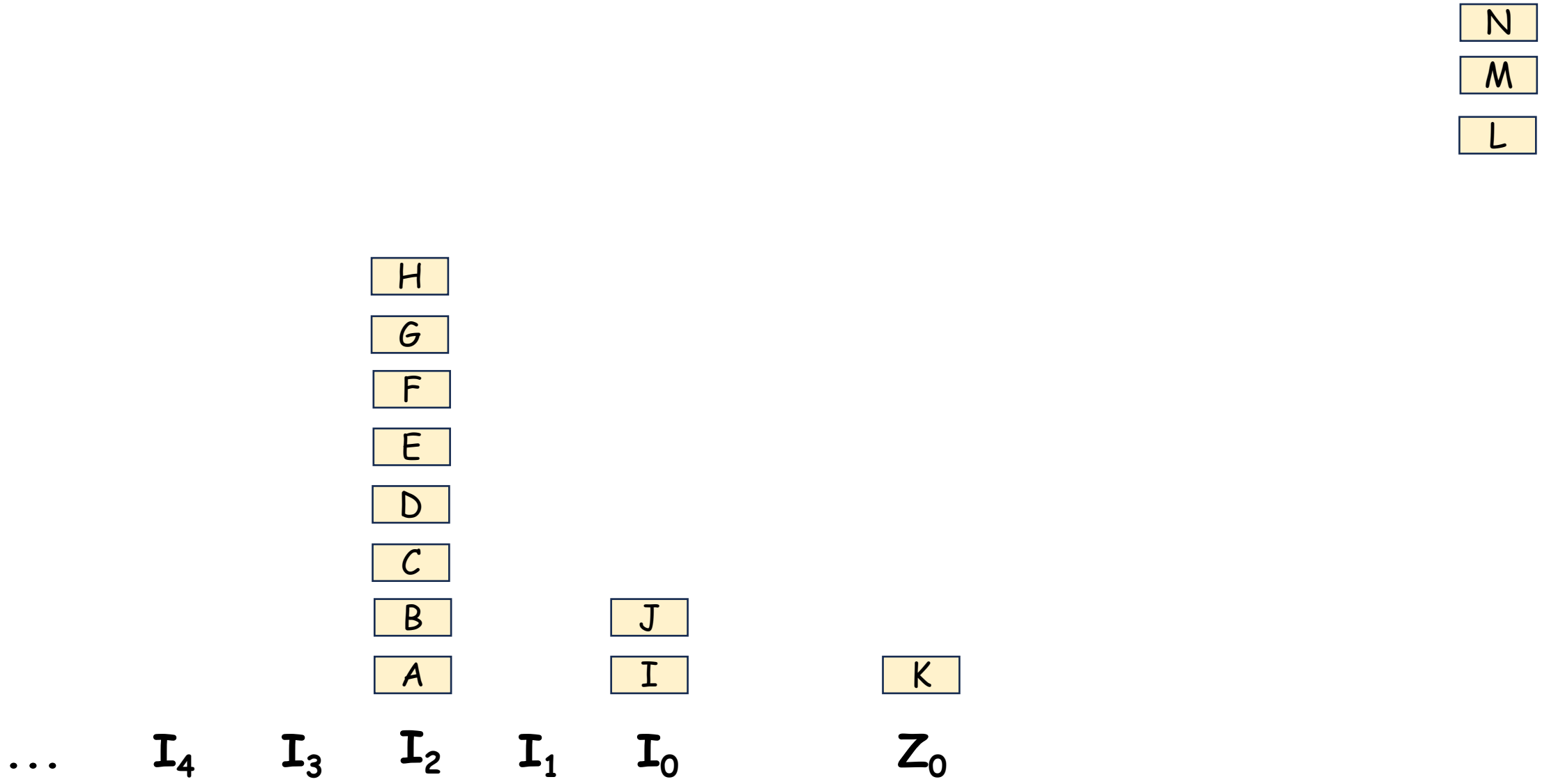

H
G
F
E
D
C
B
A
Z_2
Z2 itu index sementara
merge
H
G
F
E
Z_1
D
C
B
A
N
M
L
K
J
I
...
I_4
I_3
I_2
I_1
I_0
Z_0

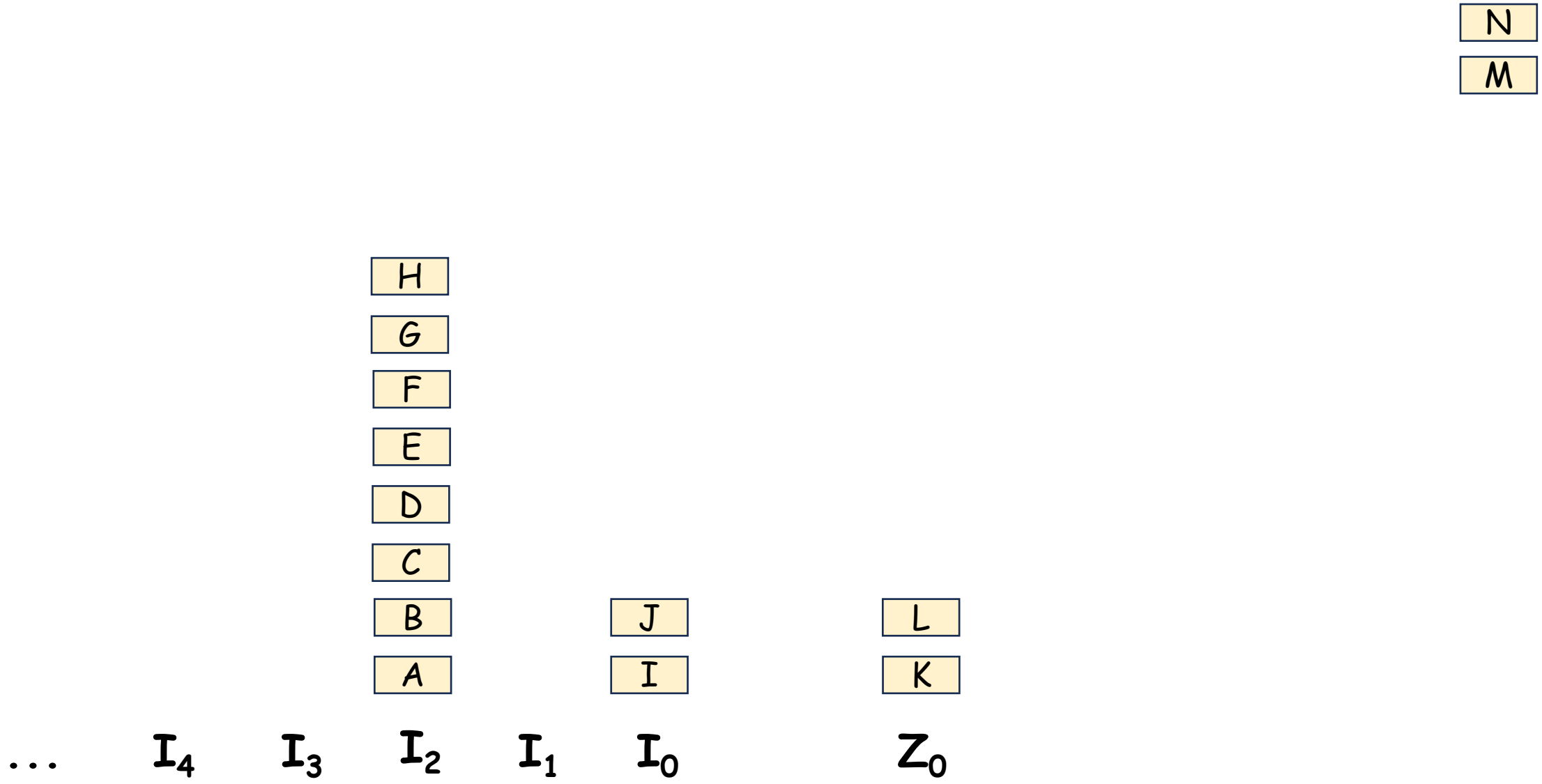


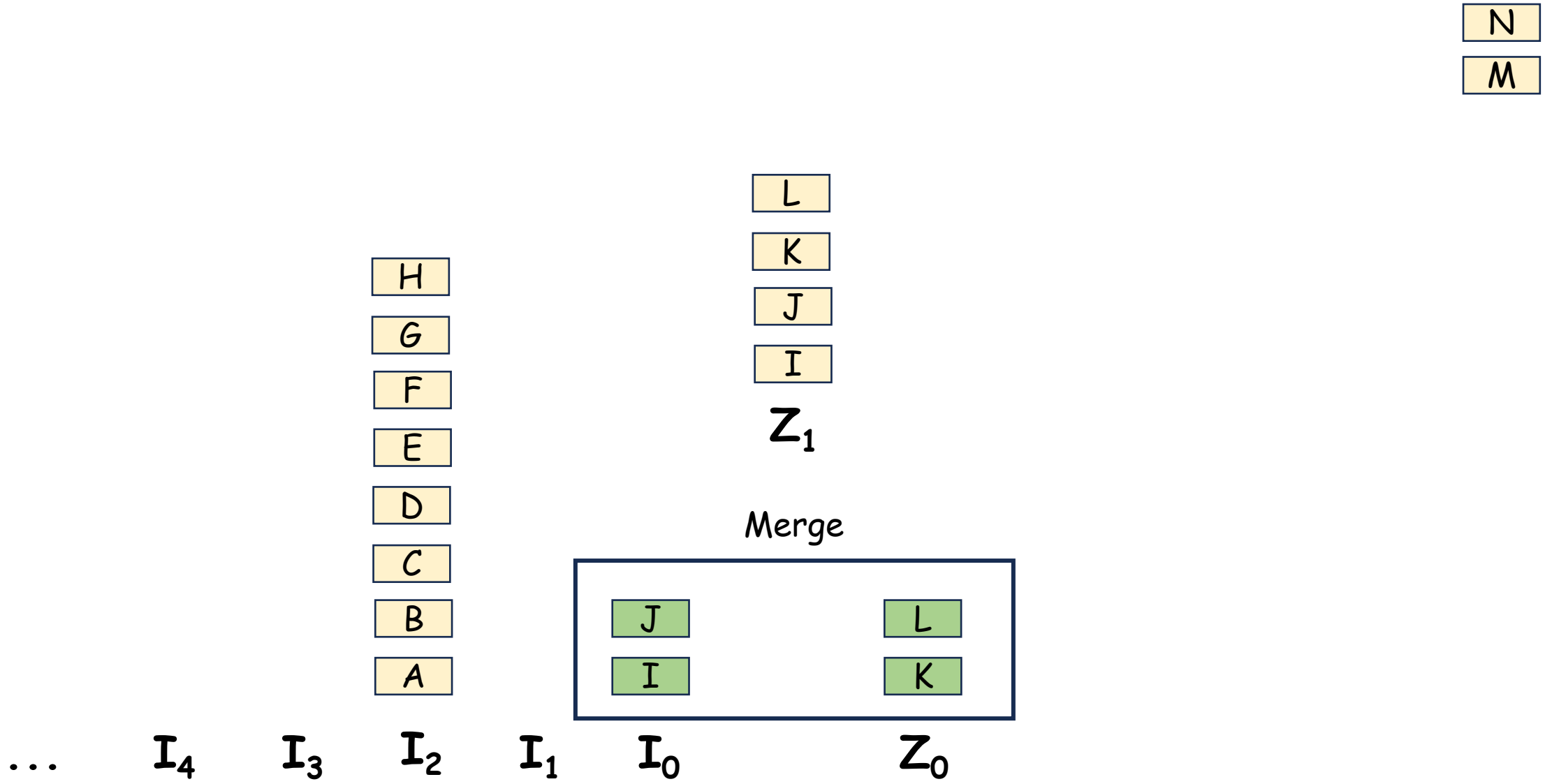






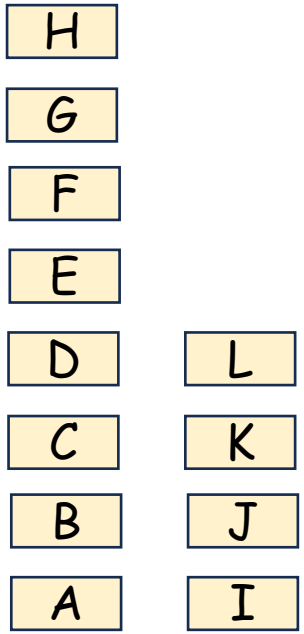





N
M
L
K
J
I
Z_1
H
G
F
E
D
C
B
A
Merge
J
I
L
K
...
I_4
I_3
I_2
I_1
I_0
Z_0

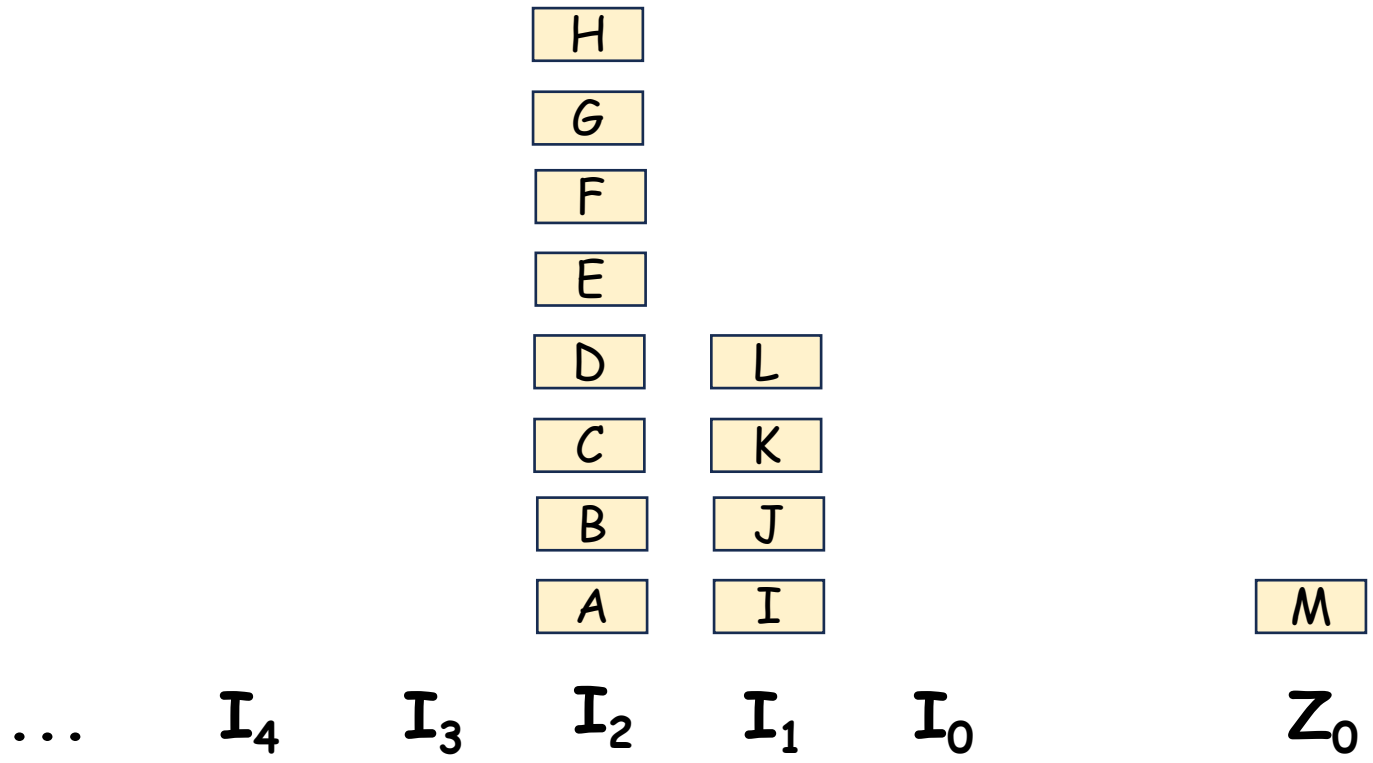
N

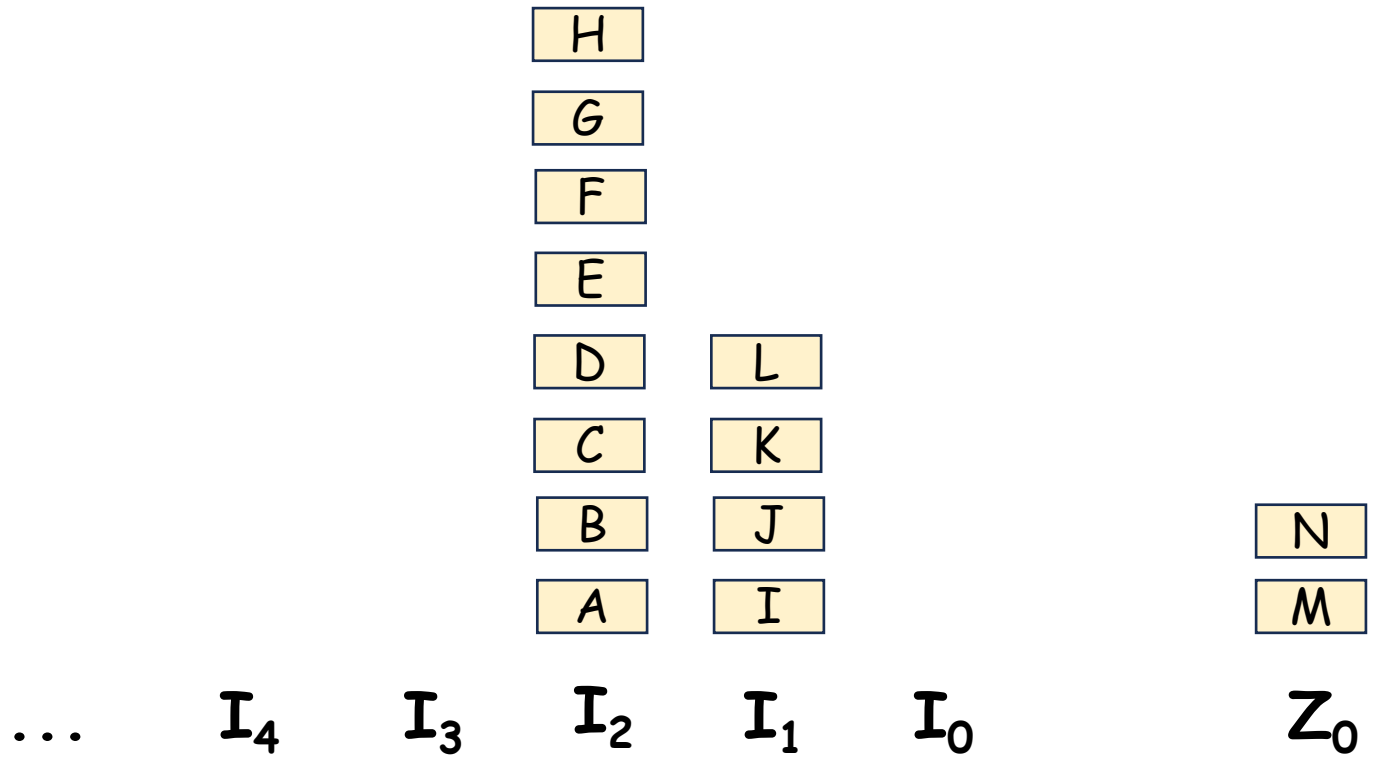
M



... I_4 I_3 I_2 I_1 I_0 Z_0

N





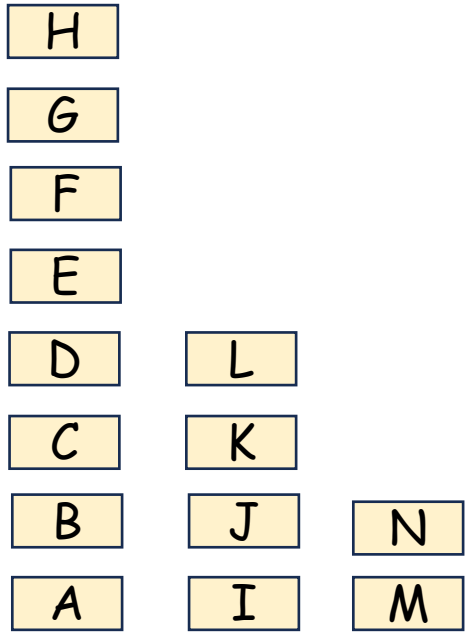
There are $\log_2(T/n)$ indexes.

O T log T/n) , karena untuk setiap posting cuman di proses sekali setiap log2 (T/n)

Ada T term, masing-masing t (posting) hanya di proses sekali setial log2 T/n

Overall index construction time is $O(T \log_2(T/n))$ because each posting is processed only once on each of $\log_2(T/n)$ the levels.

Kayak A cuman di pindah-pindah/dievaluasi sekali aja setiap per level).

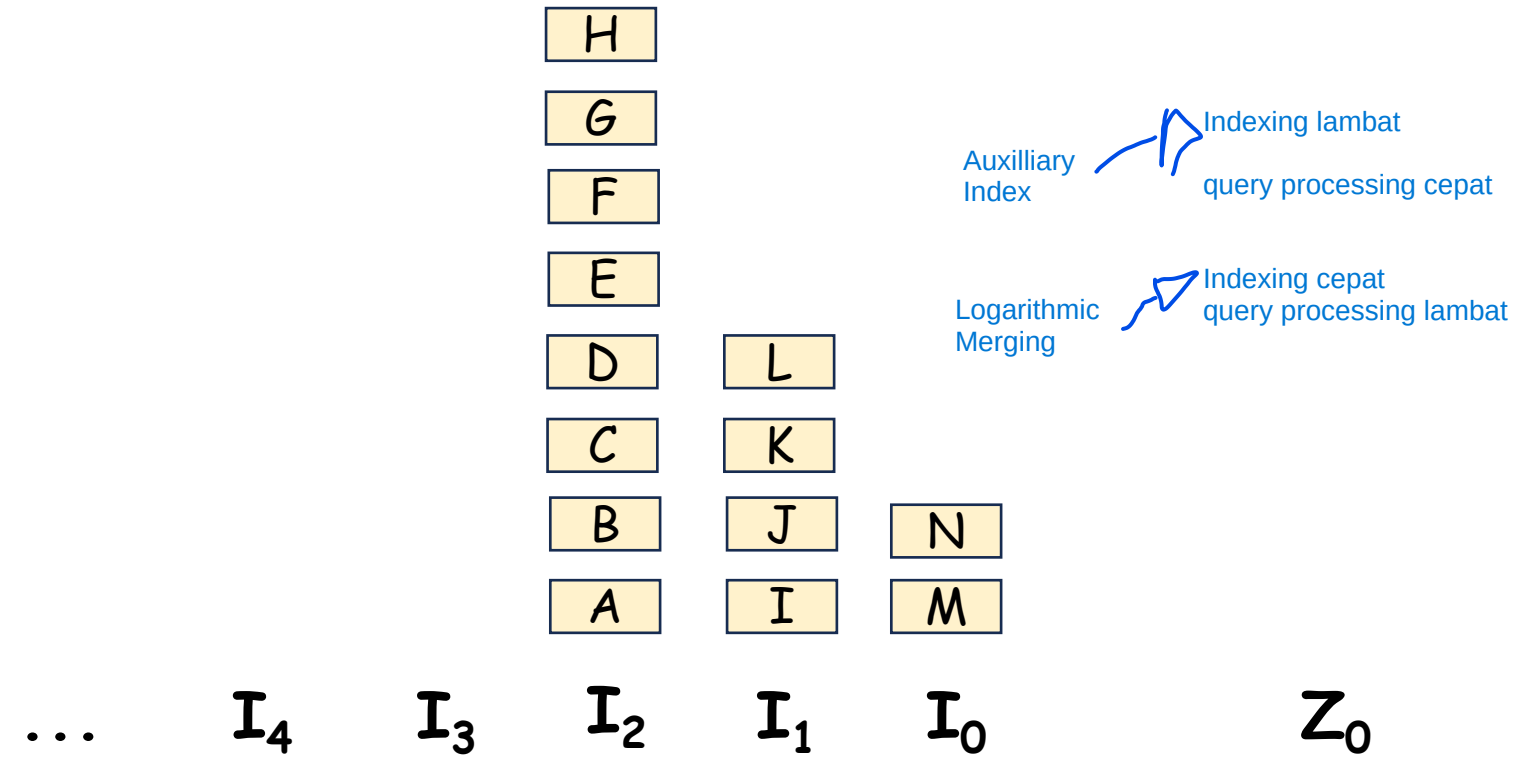


... I_4 I_3 I_2 I_1 I_0 Z_0

Dan seterusnya ...

However, We trade this efficiency gain for a slow down of query processing. We now need to merge results from $\log_2(T/n)$ indexes as opposed to **just two**.

Kita harus merge hasil dari Log2 (T/n) indexes, dibandingkan hanyalah 2 (main index dan aux index)



Auxilliary Index → Indexing lambat, query processing cepat

Logarithmic Merging → Indexing cepat, query processing lambat

Kalau menghapus akan di kasih flag (lazy deletion, jadi gak dihapus total)

Dan seterusnya ...