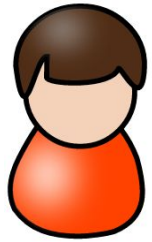
Searchable Symmetric Encryption (SSE)

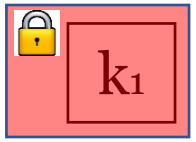
Encrypted index

deterministic!

token

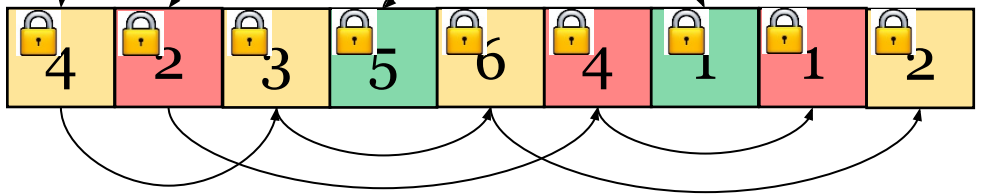
client





server

F1 F2 F4

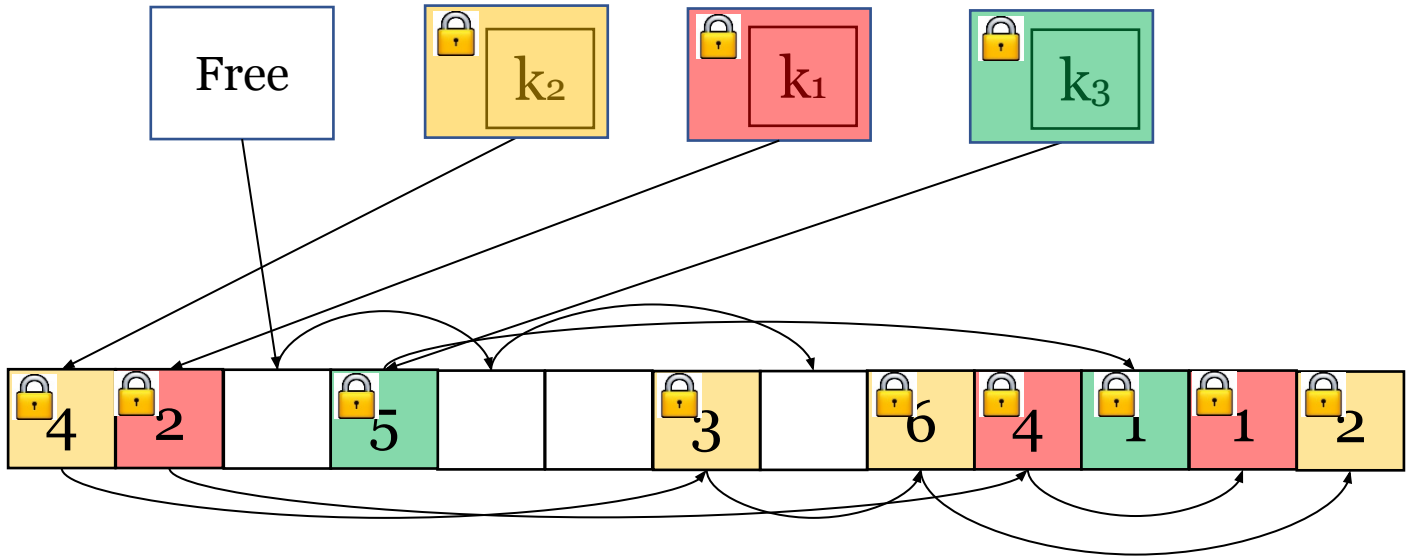


F1 F2 F3 F4 F5 F6

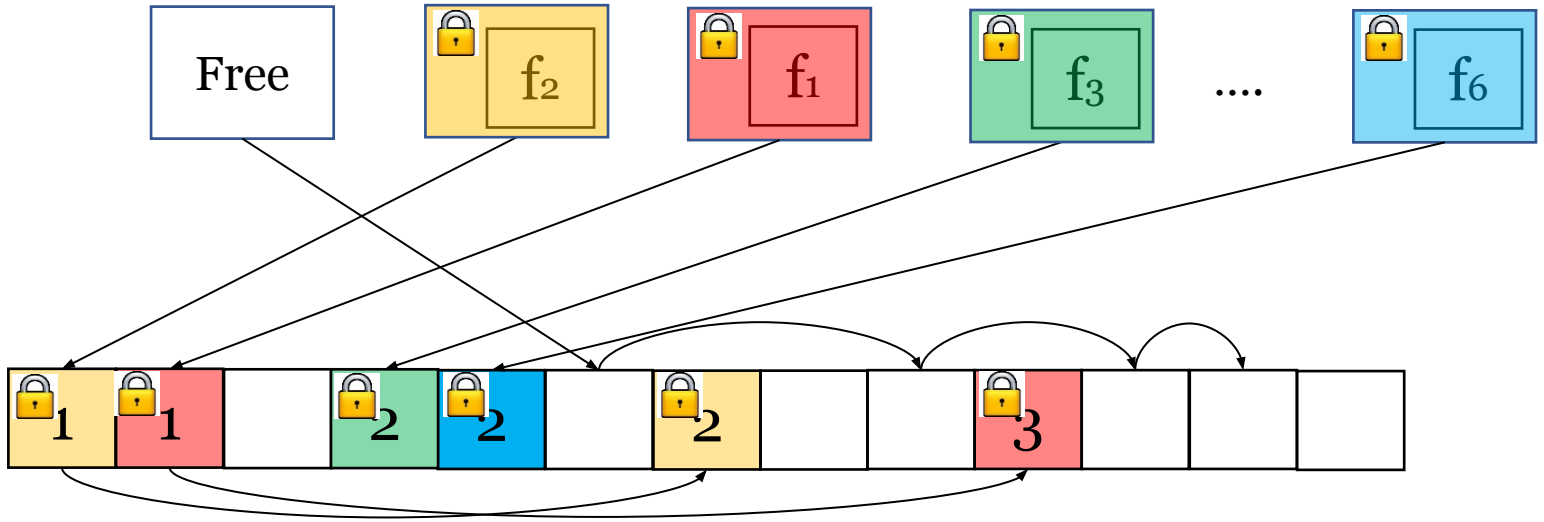
file access patterns!

Dual Index for Deletion

Search index:



Deletion index:



Leakage

- Access pattern
- Search pattern

- Add: if keyword w appears in any other file

- Delete: pointer of previous and next element

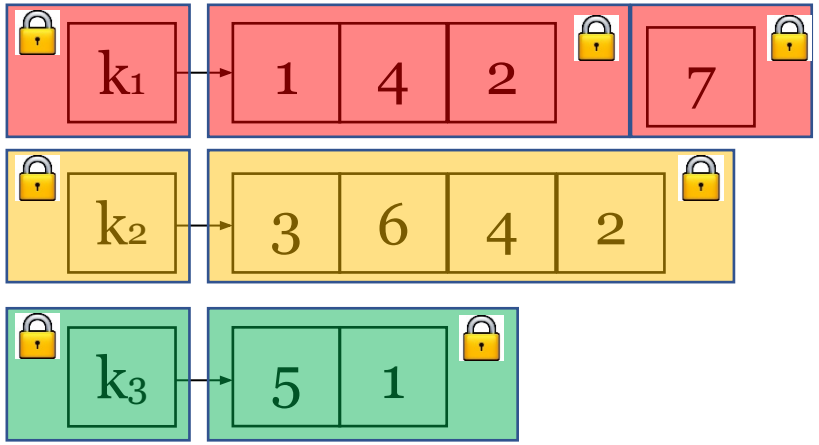
Forward and backward security

- Forward privacy: server cannot search on new files using old tokens

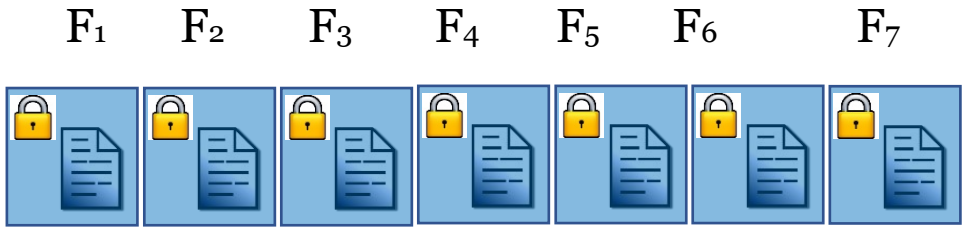
- Backward privacy: server cannot search on deleted files using new tokens

Encrypted index: no forward privacy

Pseudo random function:



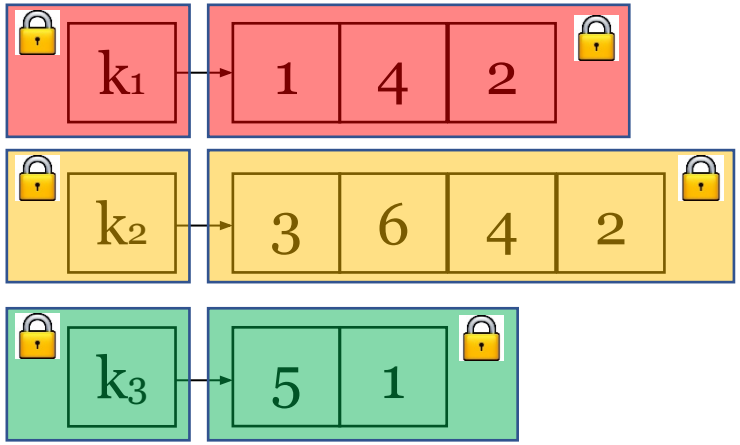
Encryption:



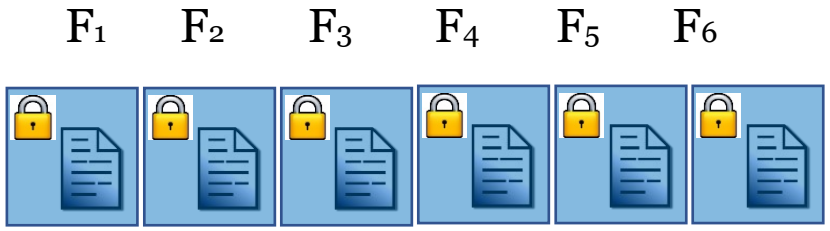
- $\mathsf{SrchToken}(K, w)$: compute and output $\tau_s := \left(F_{K_1}(w), G_{K_2}(w), P_{K_3}(w)\right)$

Encrypted index: no backward privacy

Pseudo random function:



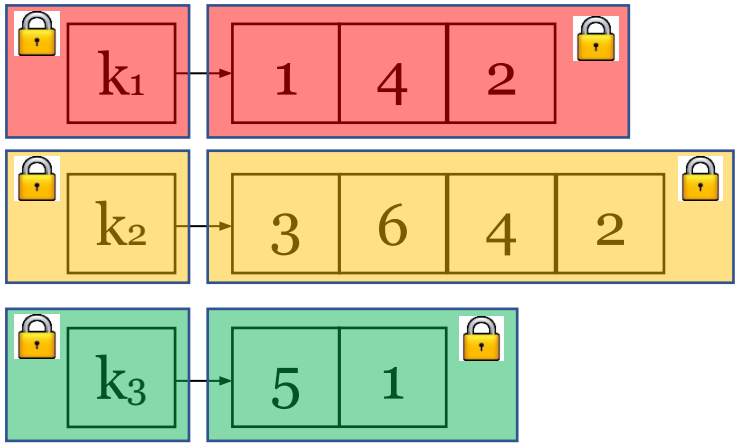
Encryption:



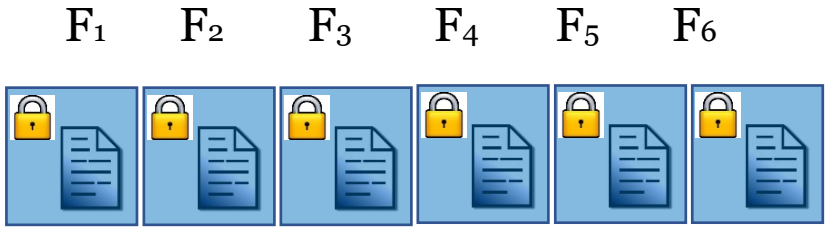
Dynamic SSE with forward privacy

Idea 1: insertion is easier than deletion

Insert (w, id, ADD) and (w, id, DELETE) instead of insert and delete (w, id)

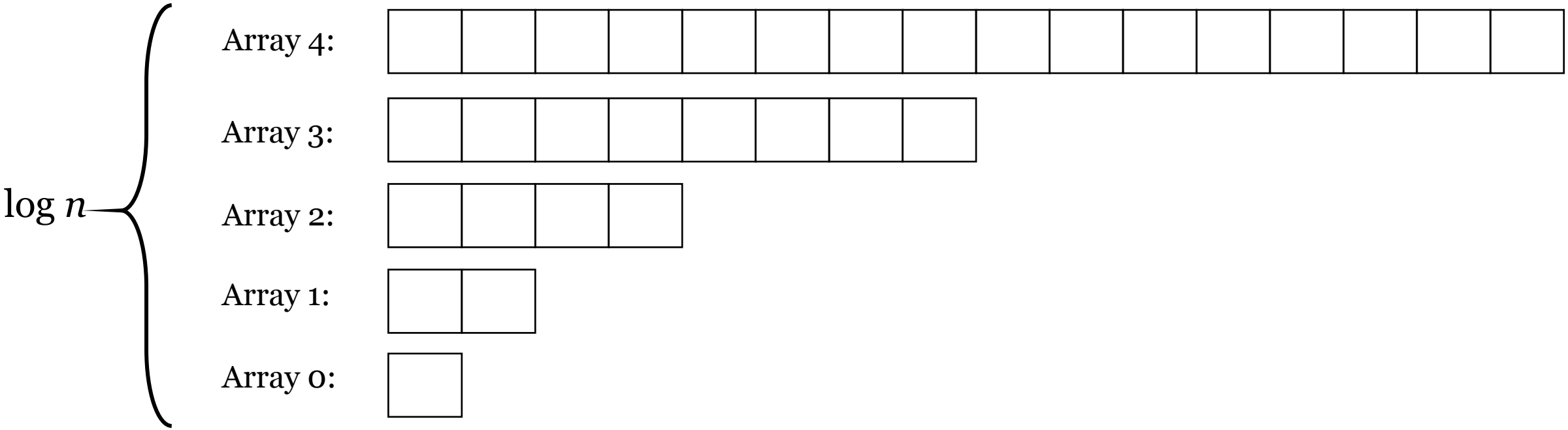


Problem: search time is not optimal



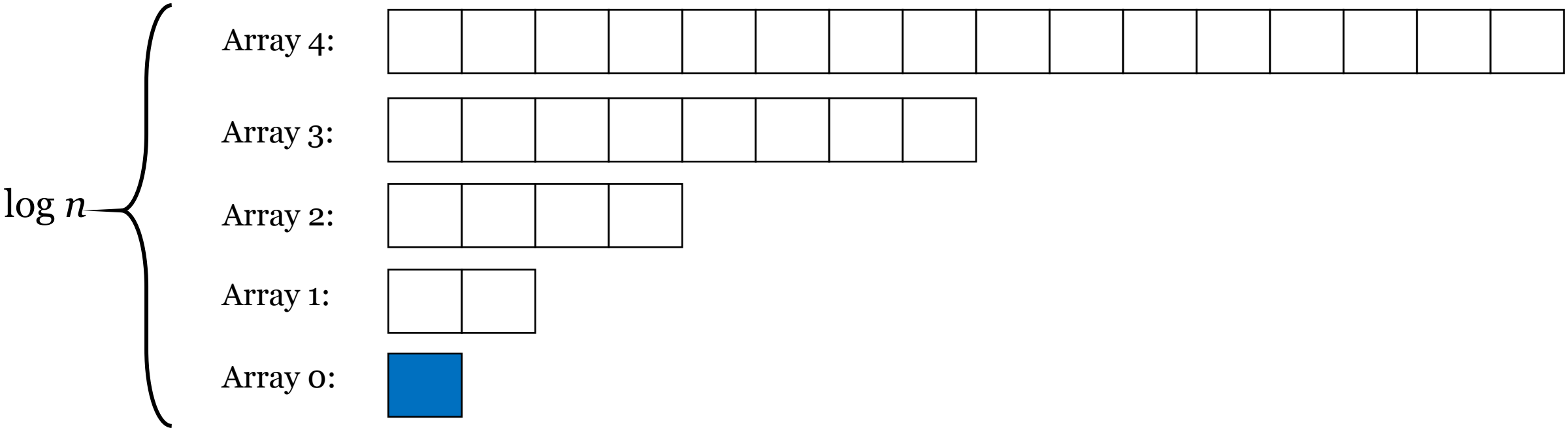
Dynamic SSE with forward privacy

- Idea 2: a new data structure

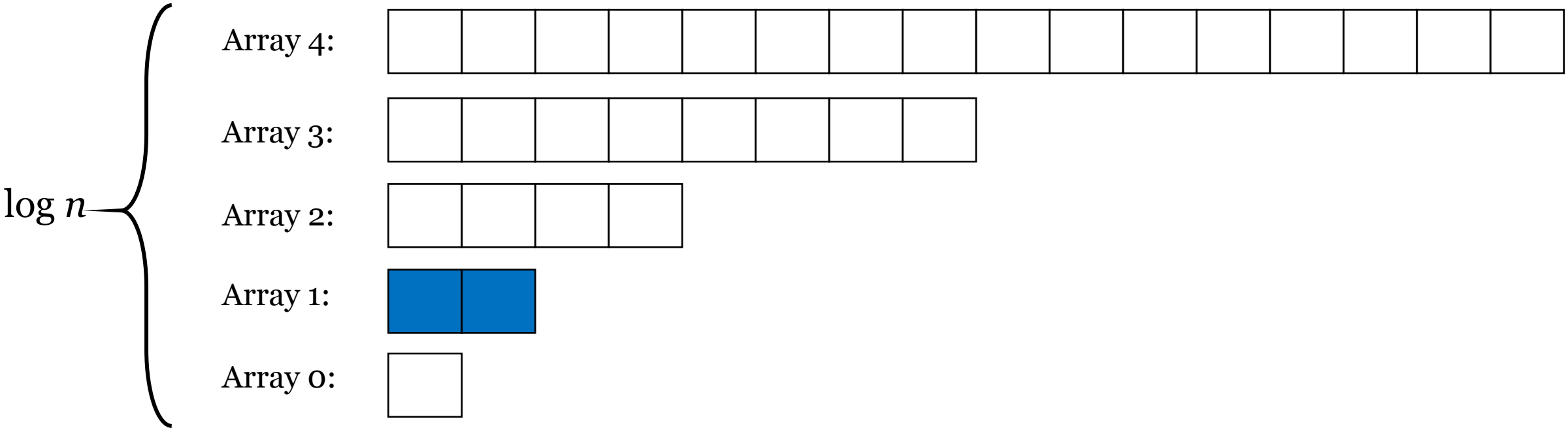


log n arrays, each with 2^i elements
O(n) space in total

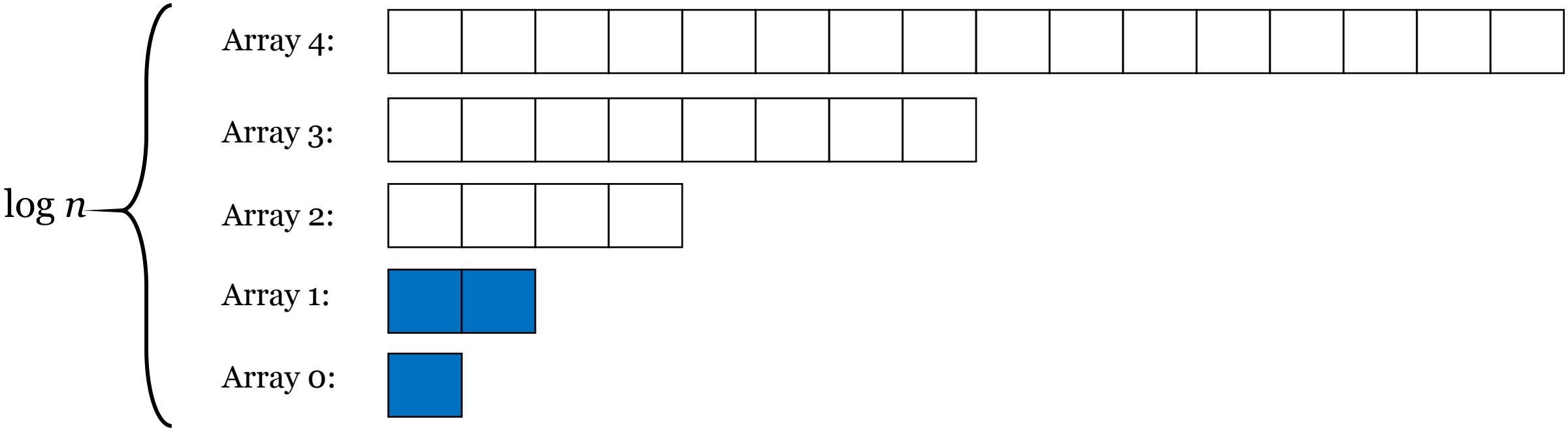
Insertion



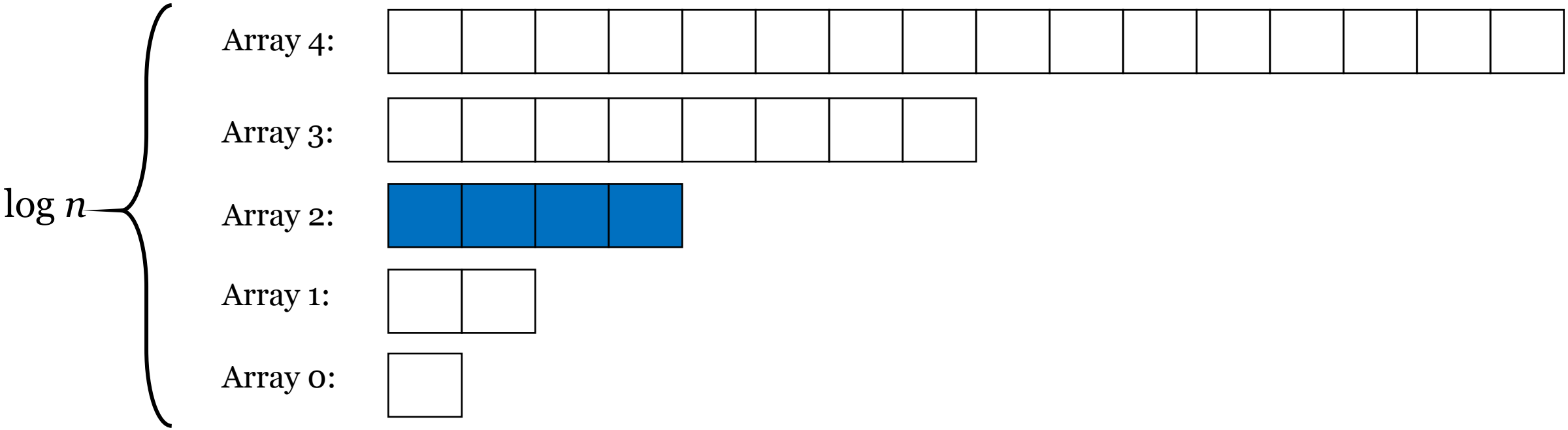
Insertion



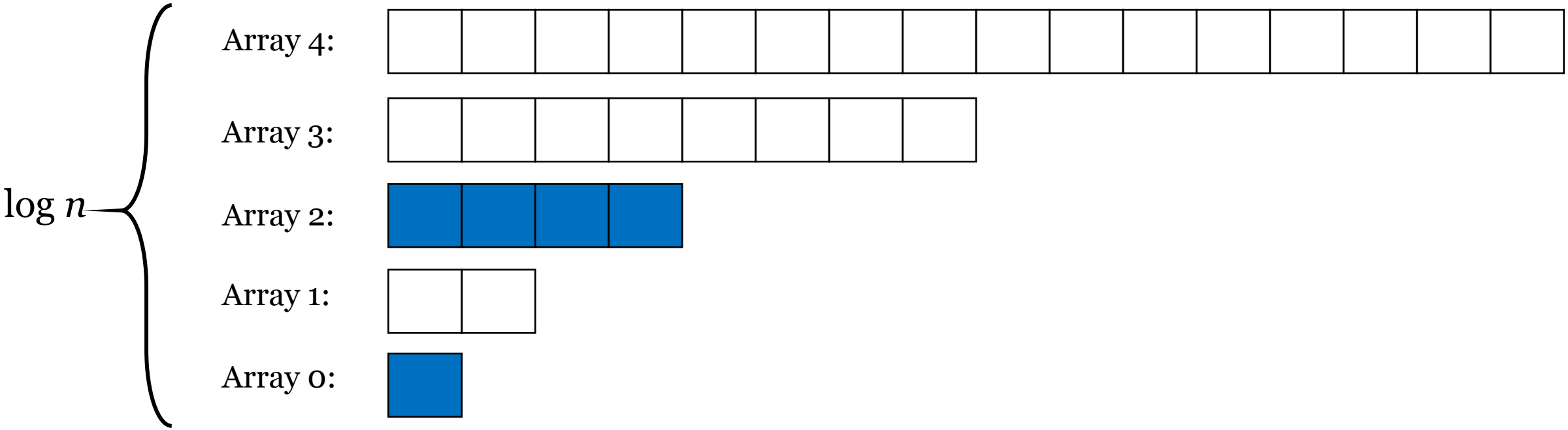
Insertion



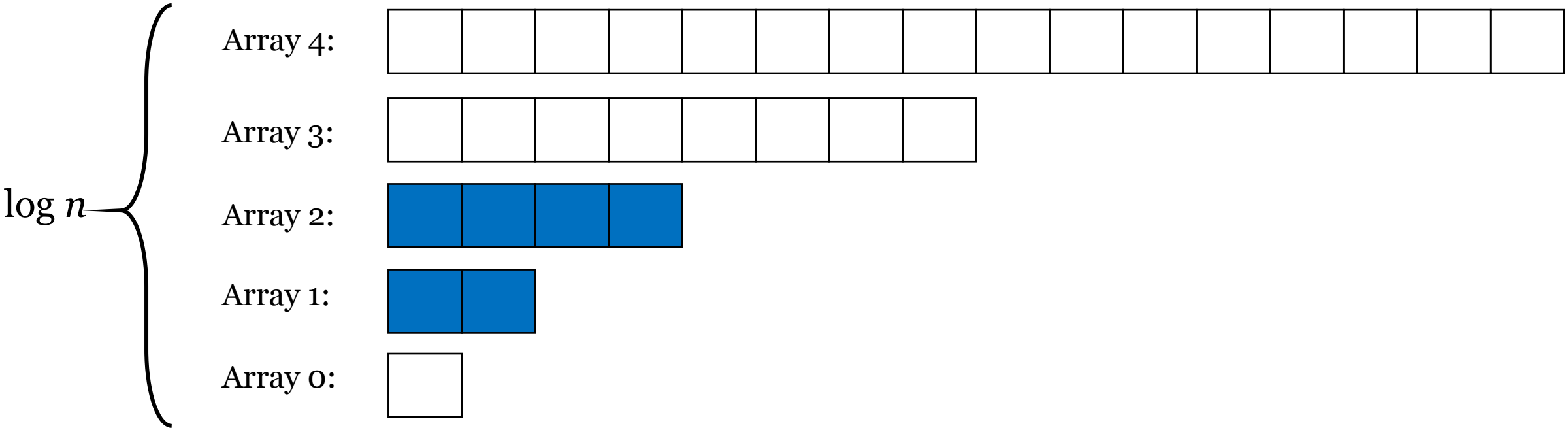
Insertion



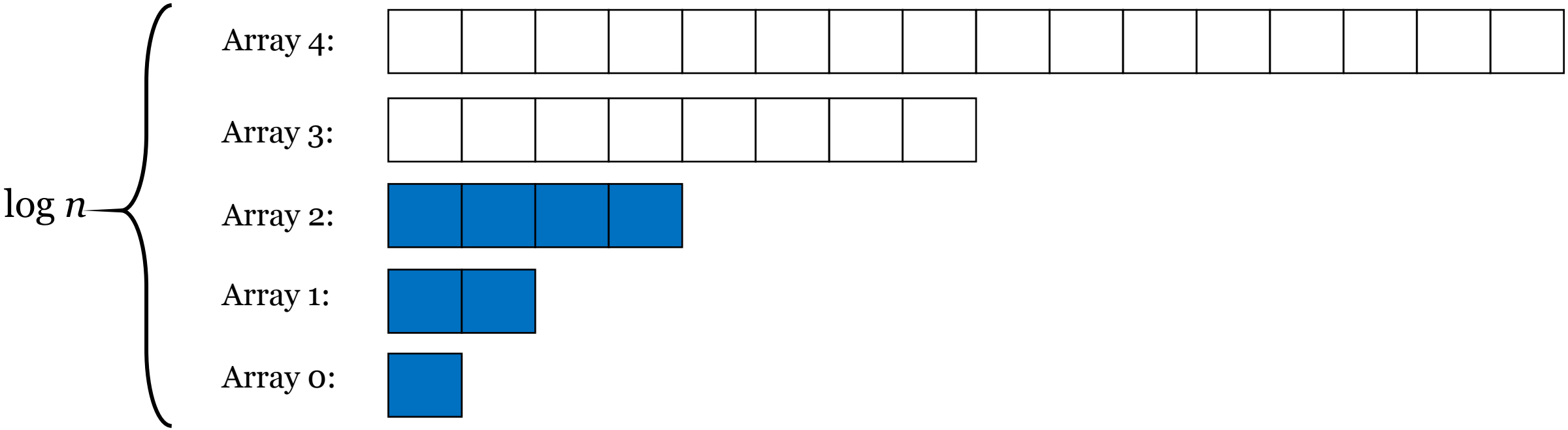
Insertion



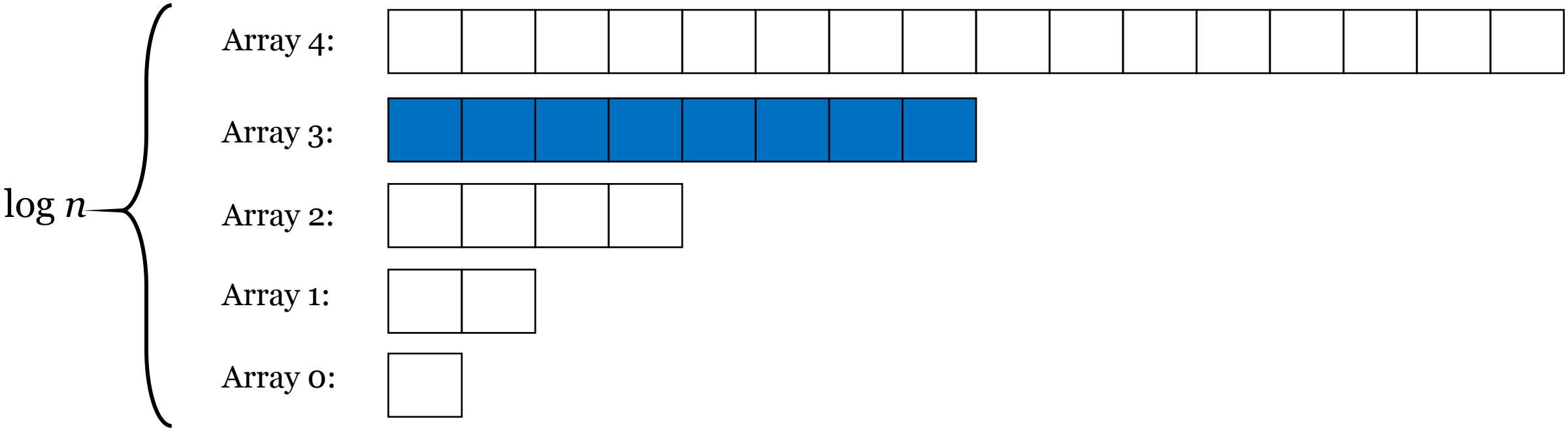
Insertion



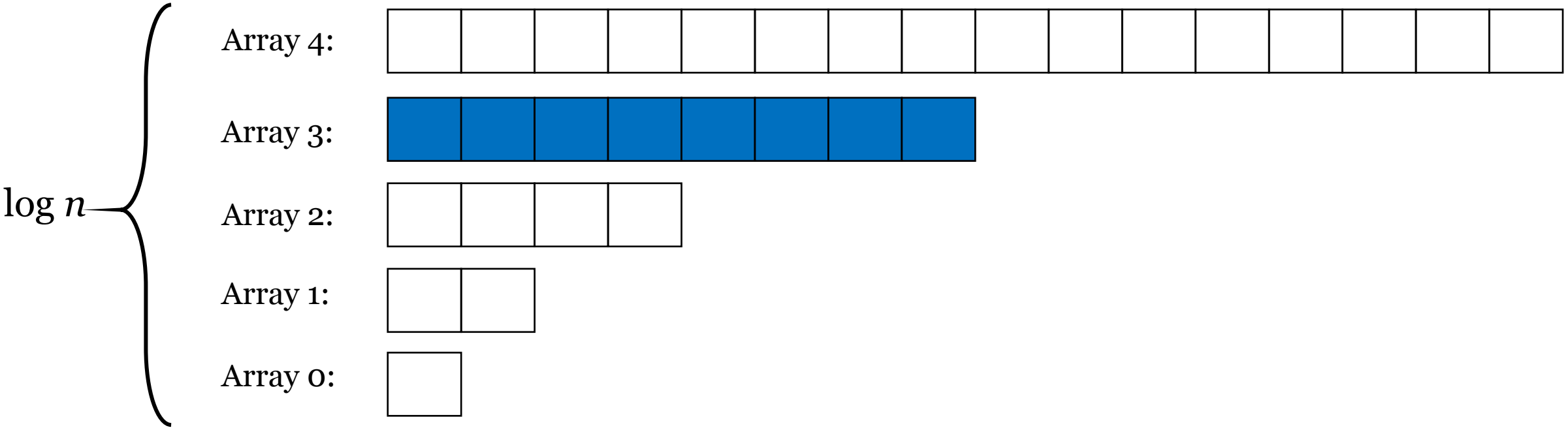
Insertion



Insertion



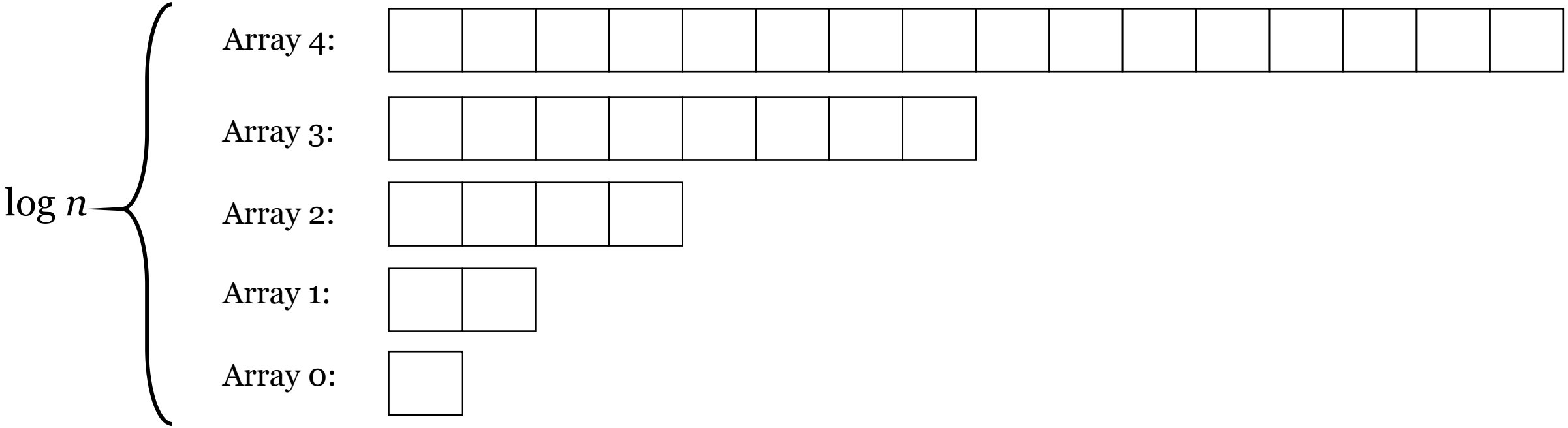
Complexity of insertion



Array 0: size 1, rebuilt n/2 times
Array 1: size 2, rebuilt n/4 times
Array 2: size 4, rebuilt n/8 times
……
Array i: size 2^i, rebuilt $n/2^{i+1}$ times

Total Complexity: O(n log$n$)

Dynamic SSE using the data structure

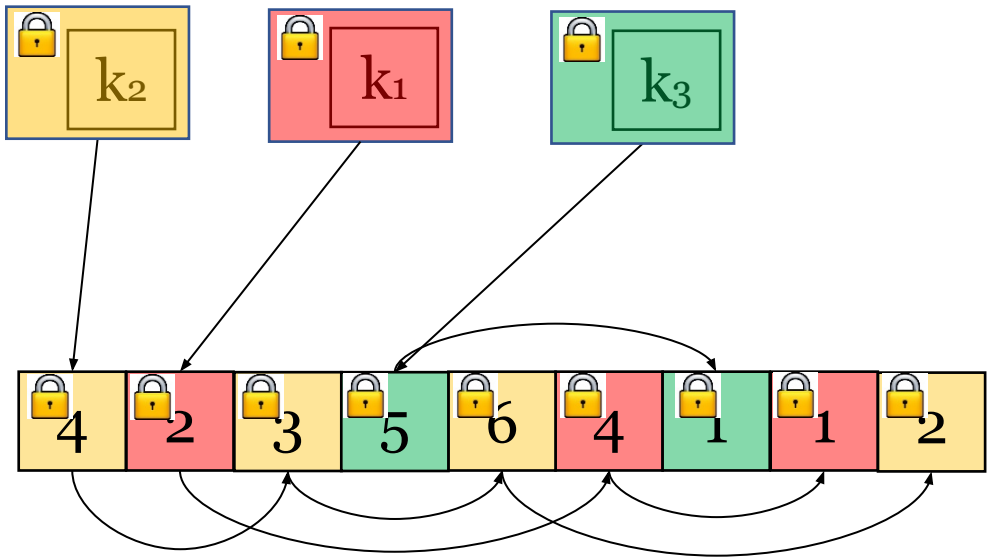


1. Insert (w, id, ADD) and (w, id, DELETE) *
2. If array 0 is empty, put it into array 0; else find the first empty level l, rebuild by downloading all arrays <= l

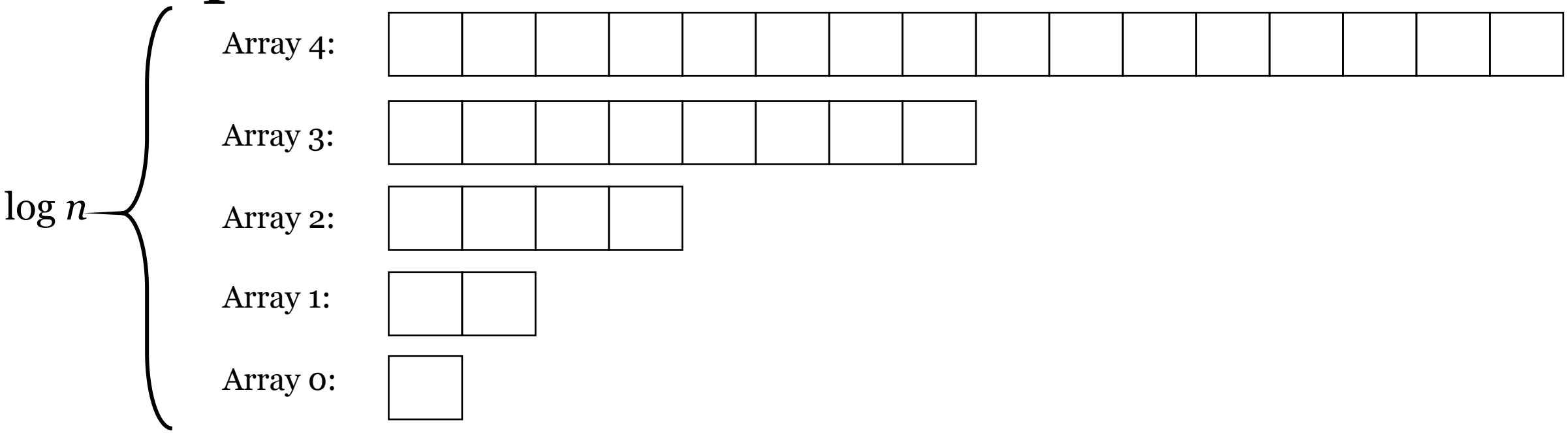
   • Cancel ADD and DELETE with same (w, id)
   • Put everything in level l and re-encrypt with new keys *

Inside each array

- No updates! Only build and destroy

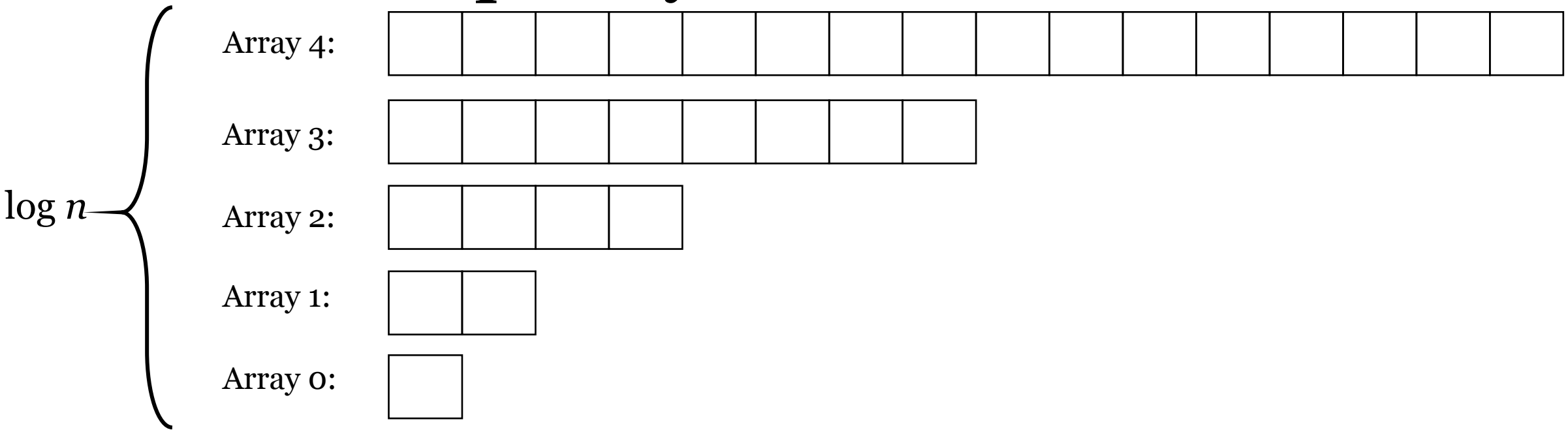


Properties



- Each update is in a level with new key
- Search: issue one token for each layer separately. Forward privacy !!!

Search complexity

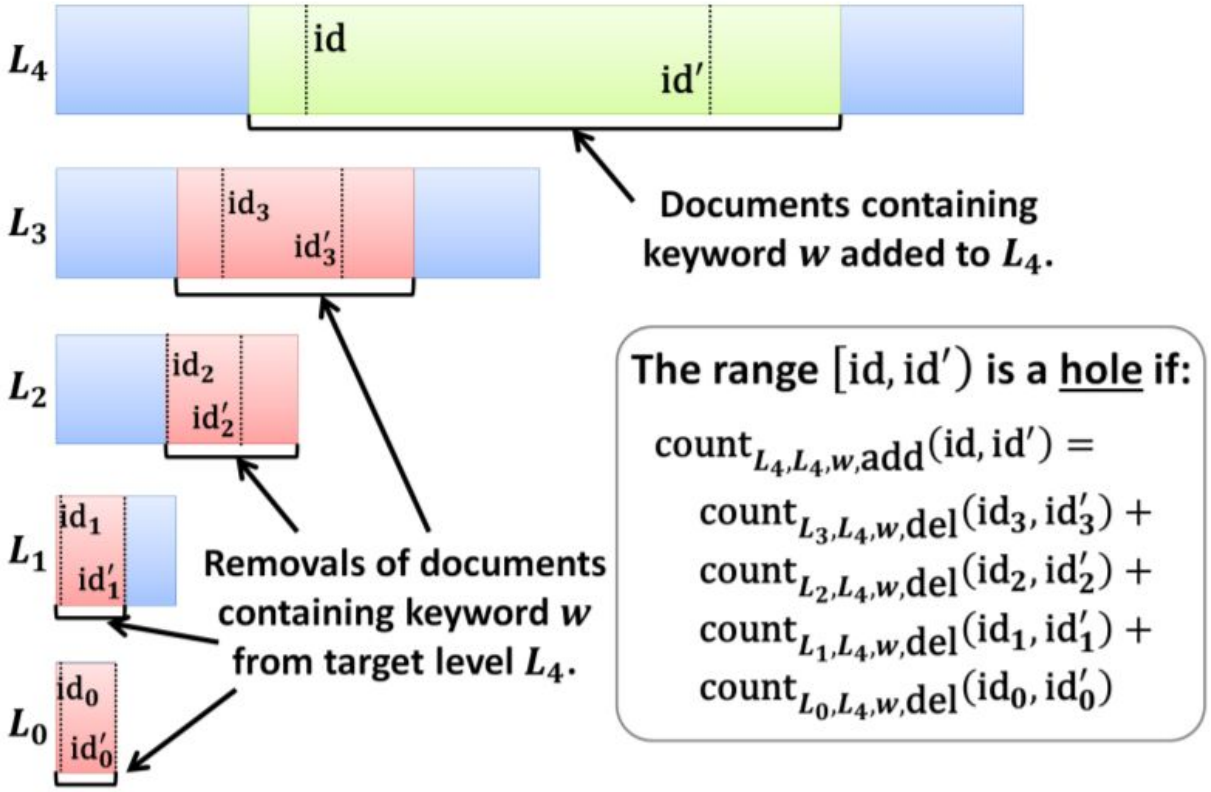


- For the same (w, id), at most 1 ADD or DELETE per level

- But search time may not be optimal in some cases: all ADDs in Array 4 and all DELETEs in Array 3

Advanced scheme

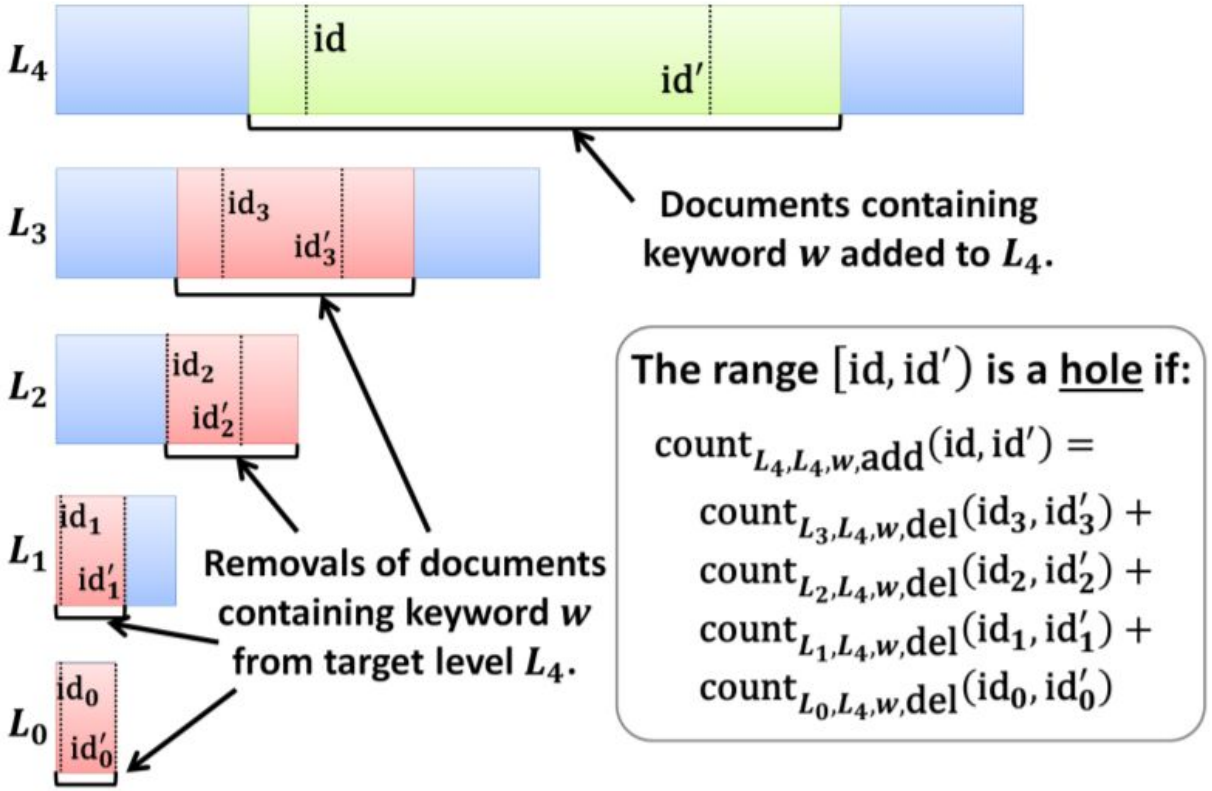
- Extra field (w, id, ADD/DELETE, l^*)
  - If ADD, l^* is the same as current level l
  - If DELETE, l^* is the corresponding level of ADD for same (w,id). $l^* > l$
  - Sort by order l^*, w, id, OP



Sublinear search time

- Double binary search

Sort by order l^*, w, id, OP



Sublinear search time

- Double binary search $O(m \log^3 n)$