Evolution of ORAMs

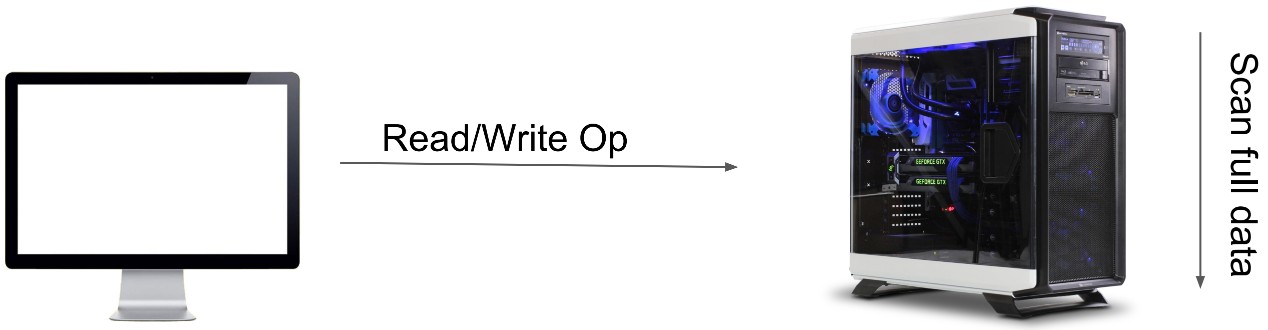
A. Satapathy C. Soni

18th July, 2017

Under the guidance of
Dr. Tzi-cker Chiueh

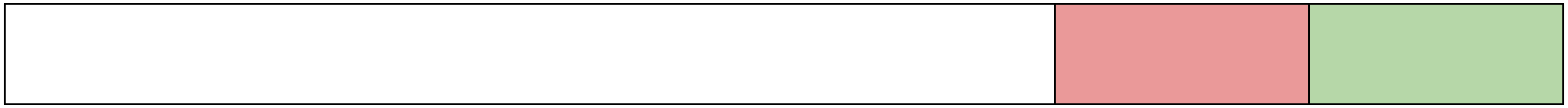
Summer Research Interns - ICL
Industrial Technology Research Institute

ORAM : Trivial Solution



- Scan full memory for each read/write operation.
- For *n* operations → $O(n^2)$
- $O(n)$ overhead per read/write operation.

Square Root ORAM



m Words

$m^{1/2}$ Dummy Words

$m^{1/2}$ Sheltered Words

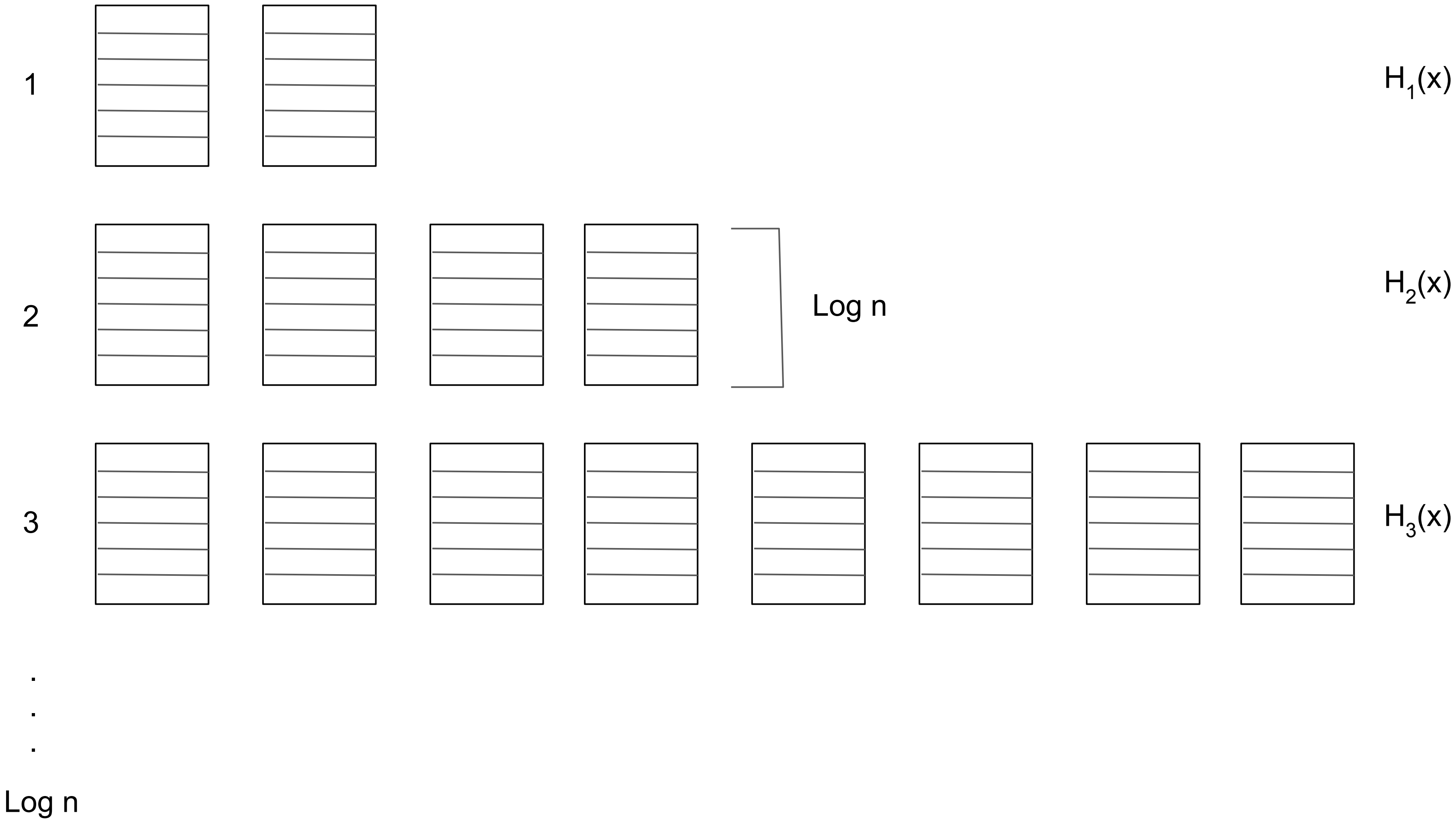
Improvements :

- Need to scan $O(m^{1/2})$ memory per operation.
- For $m^{1/2}$ operations : O(m).

Limitations:

- Shuffling is costly : New permutation function and Sorting network to remove old elements.
- Complexity : $O((m+m^{1/2})\log_2(m+m^{1/2}))$

Hierarchical ORAM



Hierarchical ORAM

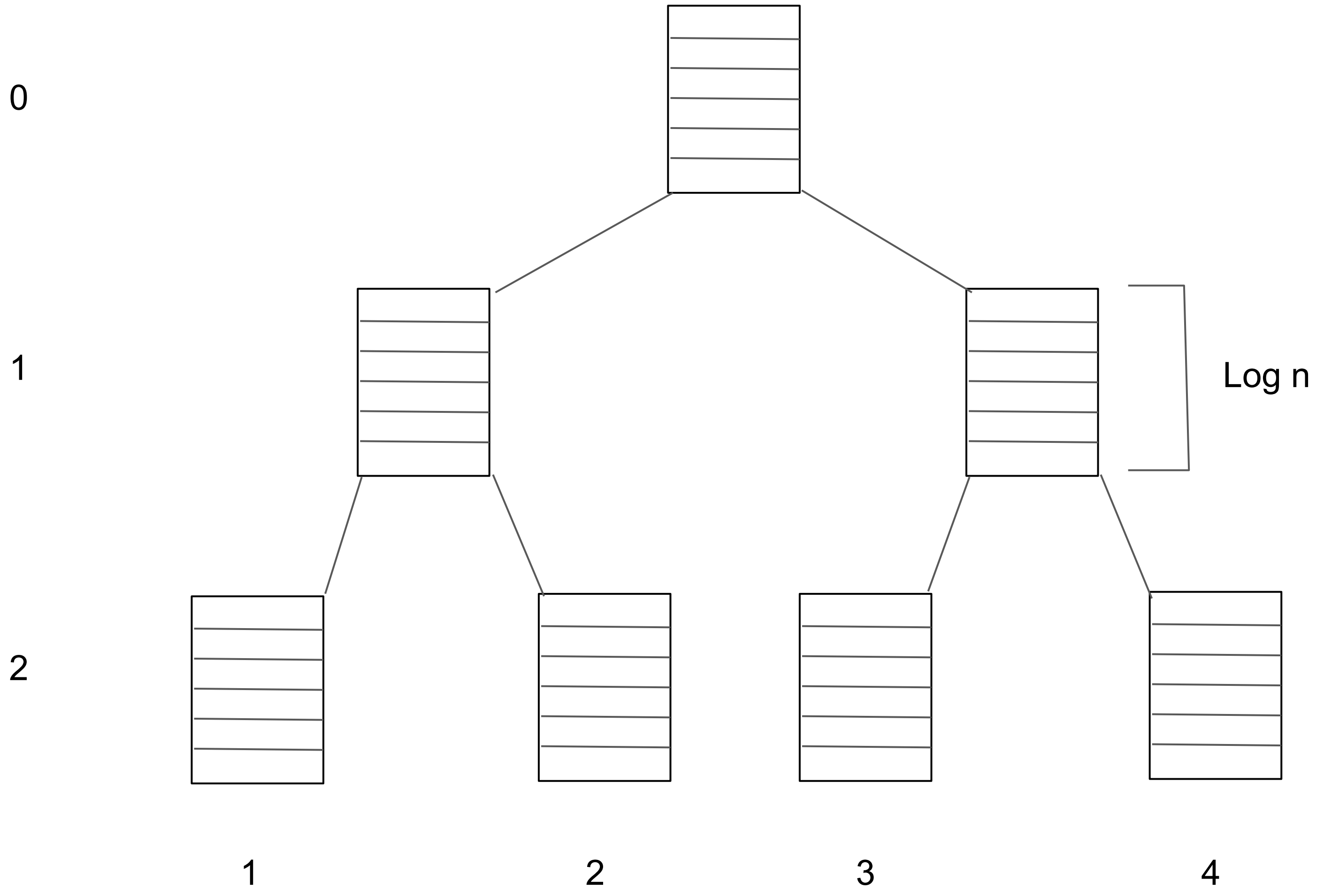
Improvements:

- Sorting needs to be done on only one level’s data instead of the full memory.
- Shuffling for level *i* needs to be done after every 2^i operations.
- As level increases, frequency of shuffling decreases.

Limitations:

- Shuffling and sorting is still expensive.
- Access Cost : $O(Log^3 n)$

Binary ORAM



n leaves

Item	Leaf
0	3
1	5
2	1
...	...
7	4

Binary ORAM

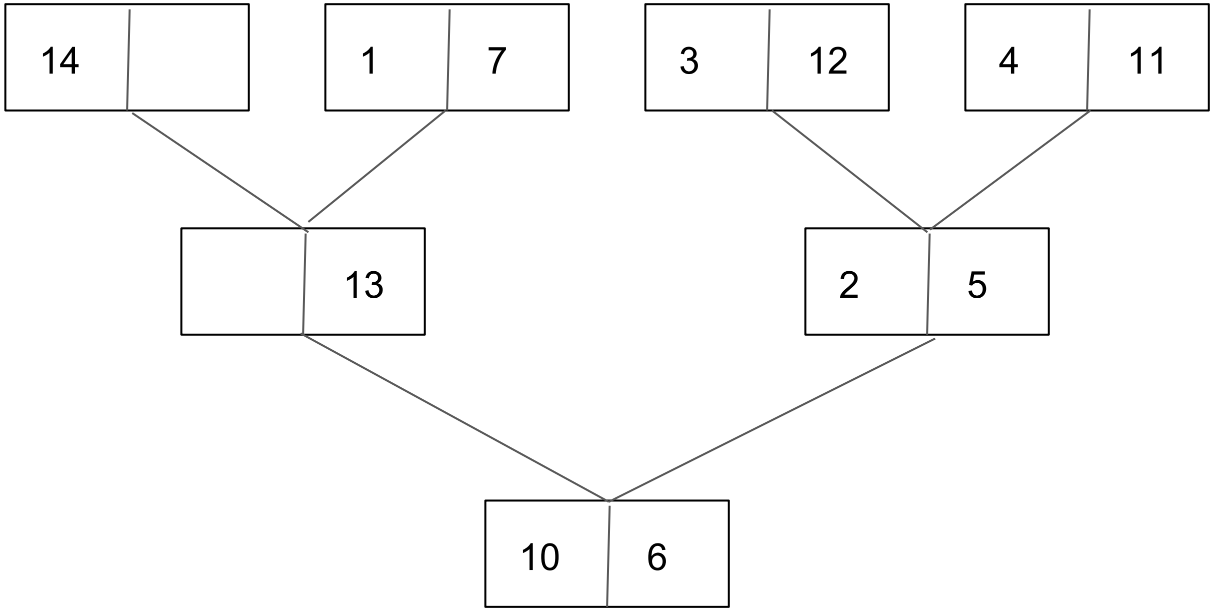
Improvements:

- No need of sorting.
- After every read operation, slot will be delocated.
- No need of Hash functions.
- Access Cost: O(Log^{2}n)

Limitations:

- Dummy values needed.
- Complex eviction procedure.

Path ORAM



Item	Leaf
1	2
2	3
3	3
...	...
7	2

8	9	1	7	13	10	6					Stash

Path ORAM

Improvements:

- Instead of root, evicted element is placed at common ancestor.
- Constant bucket size.
- No dummy values needed.
- Access Cost: O(Log n)

Limitations:

- Whole path have to be read into stash located at client side.
- All elements along the path are rearranged.
- In practical implementations, rearrangement overhead is $O(\log^2 n)$.

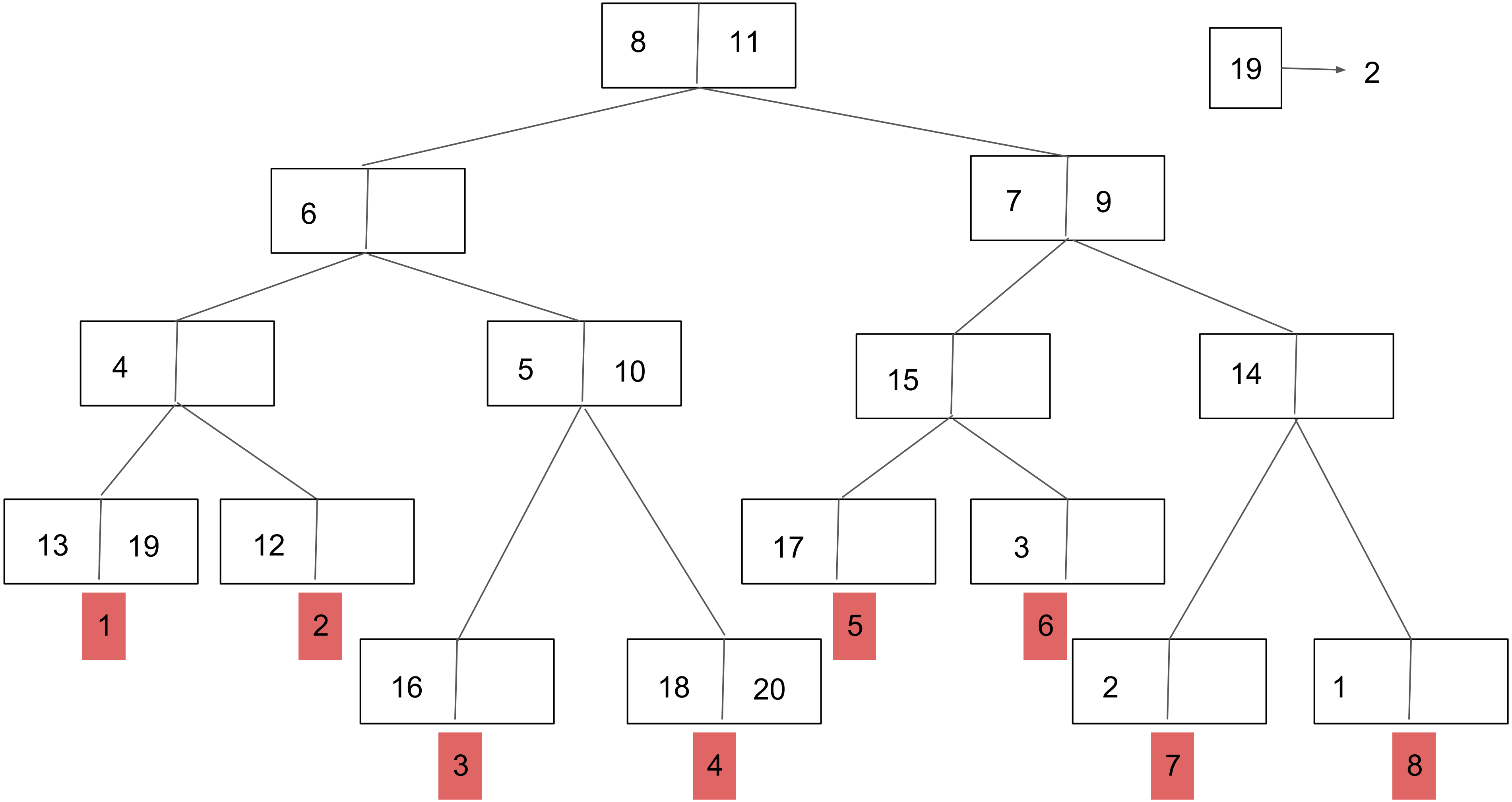
Circuit ORAM

Aim : To find an eviction circuit that is less complex than that of Path ORAM’s, and yet preserves the effectiveness of eviction.

Key idea is to complete the eviction algorithm within a single block scan of the current eviction path (while evicting as aggressively as we can).

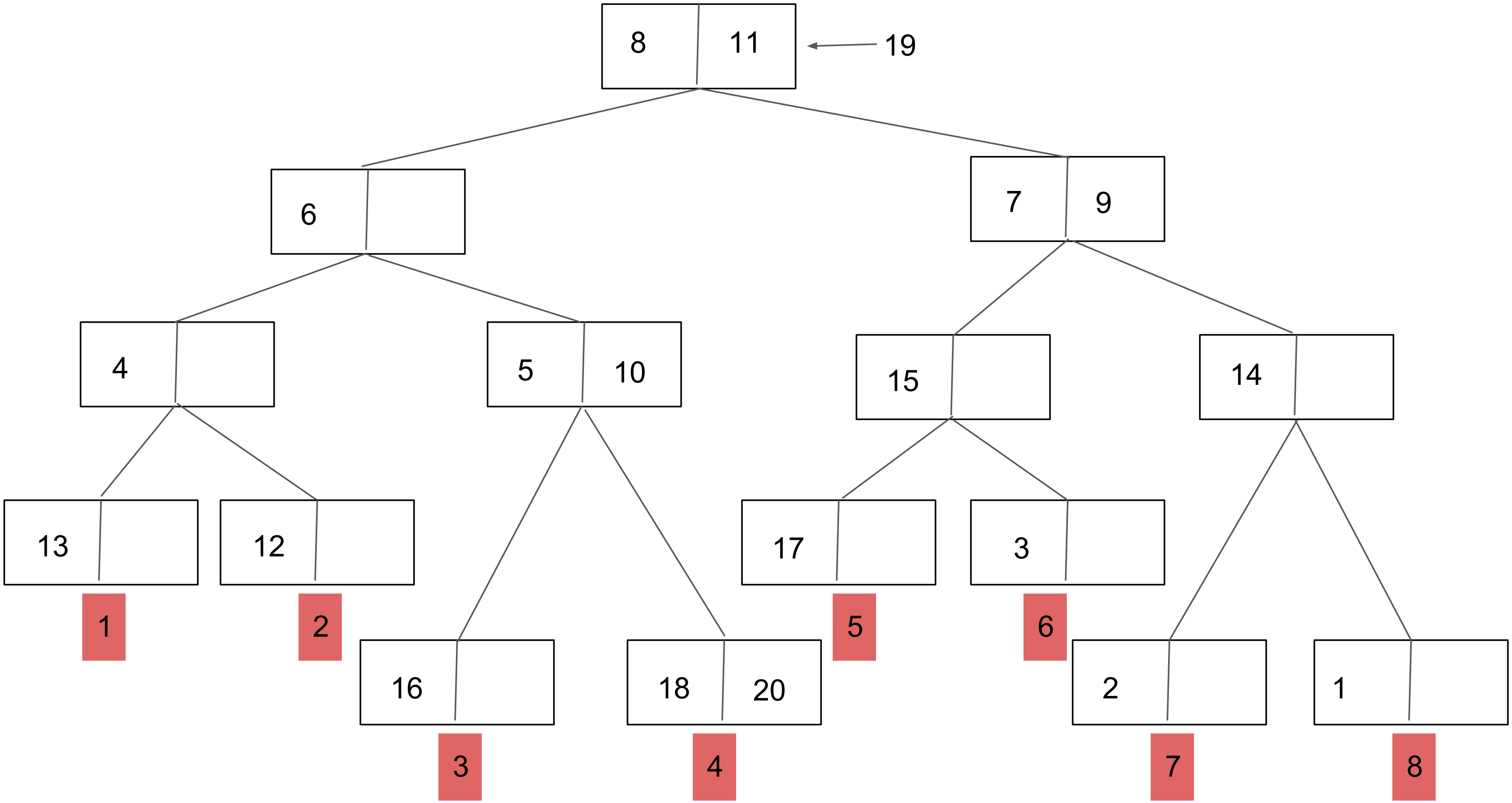
“Lack-of-foresight” : Two additional metadata scans to precompute the foresight required.

Circuit ORAM



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8	7	6	1	4	2	5	1	8	3	6	2	1	2	6	3	5	4	1	4

Circuit ORAM

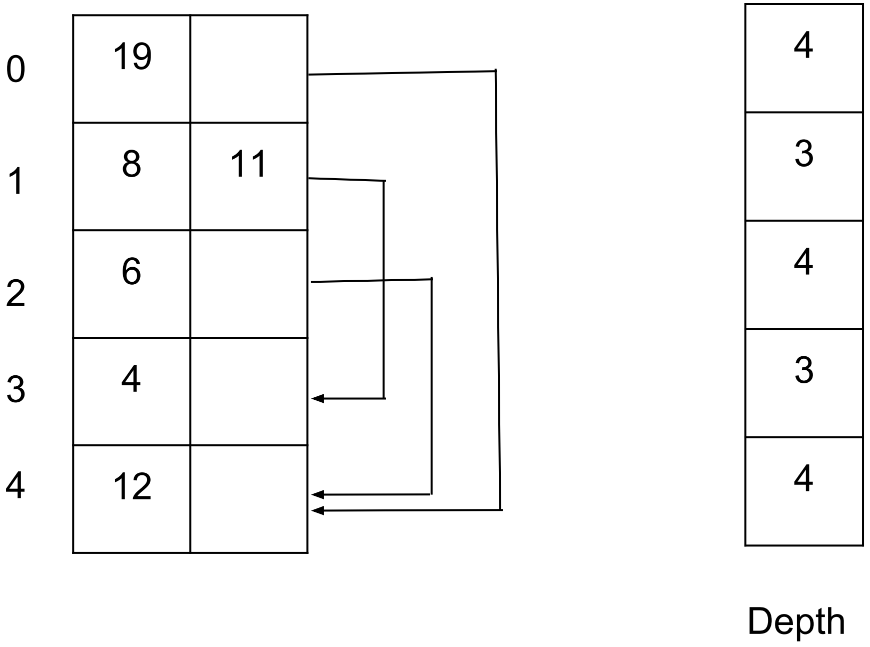


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8	7	6	1	4	2	5	1	8	3	6	2	1	2	6	3	5	4	1	4

Circuit ORAM

Find depth:

Find the deepest element in a bucket and find its depth. (Top -> Bottom).



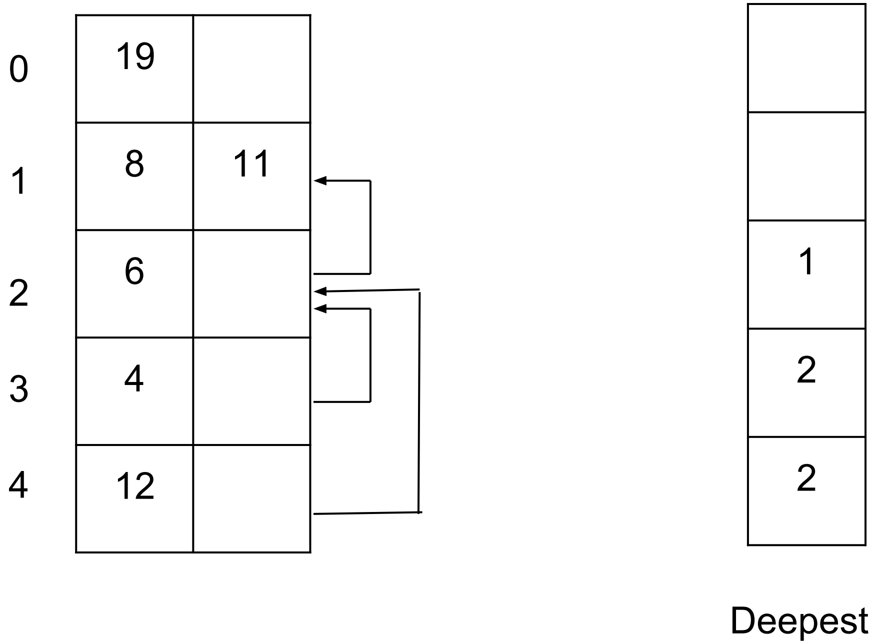
0	19	
1	8	11
2	6	
3	4	
4	12	

Depth
4
3
4
3
4

Circuit ORAM

Prepare deepest:

Assign memory locations to nearest deepest element (Bottom -> Up).



0	19	
1	8	11
2	6	
3	4	
4	12	

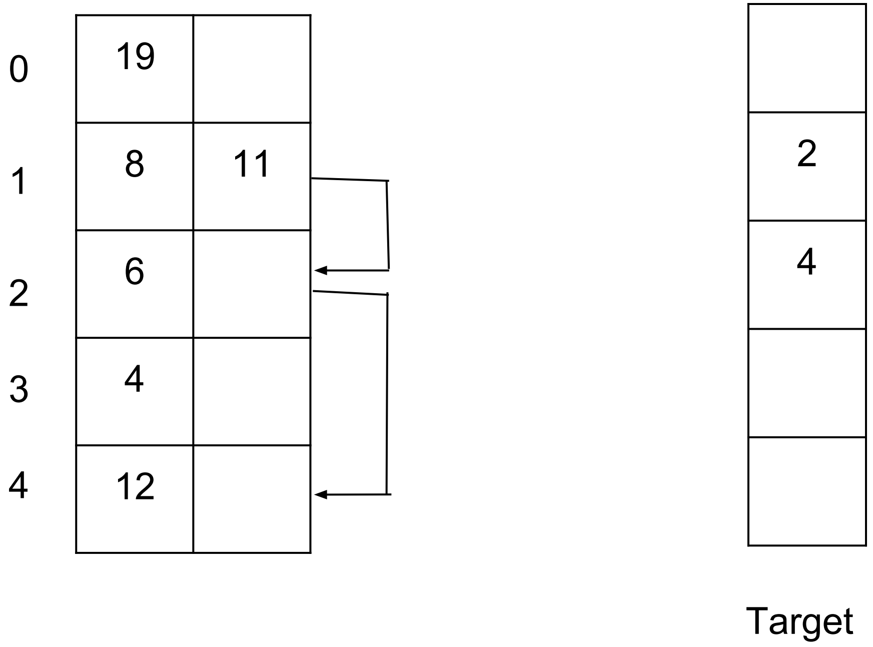
1
2
2

Deepest

Circuit ORAM

Prepare target:

(Top -> Bottom).



0	19	
1	8	11
2	6	
3	4	
4	12	

Target
2
4

Circuit ORAM

Eviction:

1	19	11
2	8	
3	4	
4	12	6

Circuit ORAM

Improvements:

- Rearrangement cost is reduced to O(Log n).
- Access Cost: O(Log n)
- Circuit ORAM achieves 58.4X improvement in terms of number of AND gates over Path ORAM, 31X improvement over the binary-tree ORAM. (over database of 4GB and with a 2^{-80} security failure probability.)