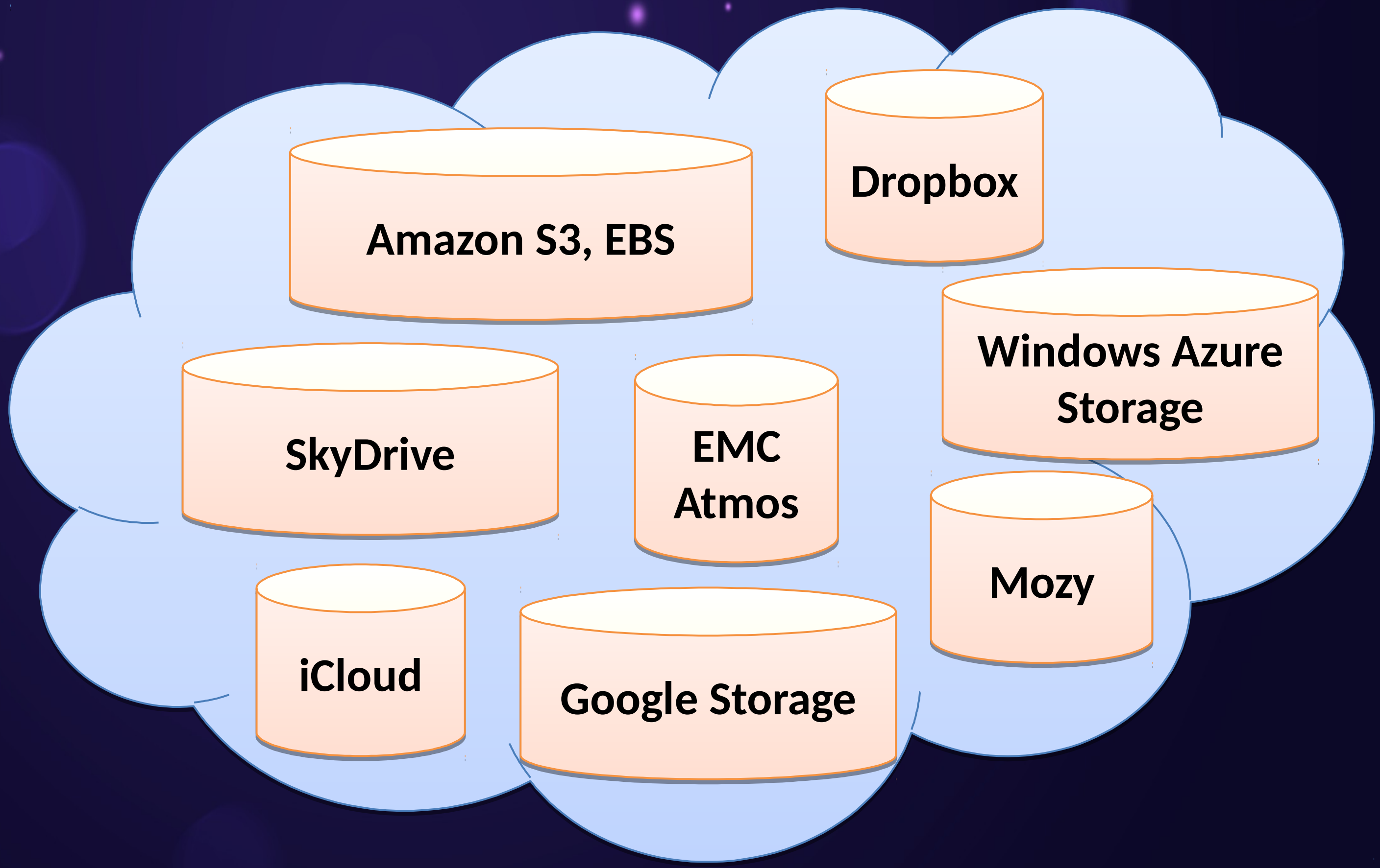
Oblivious RAM (ORAM)

more precisely: Path ORAM

Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., & Devadas, S. (2013, November). Path ORAM: an extremely simple oblivious RAM protocol. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 299-310). ACM.

Cloud storage



Cloud storage

Sensitive documents should be **encrypted**

But: the *metadata* is leaked

The *access pattern* leaks:

- Which data is being accessed
- When the data was last accessed
- Whether the same data is being accessed
- Access randomly or sequentially
- Whether the access is a read or write

Path ORAM

- Server storage

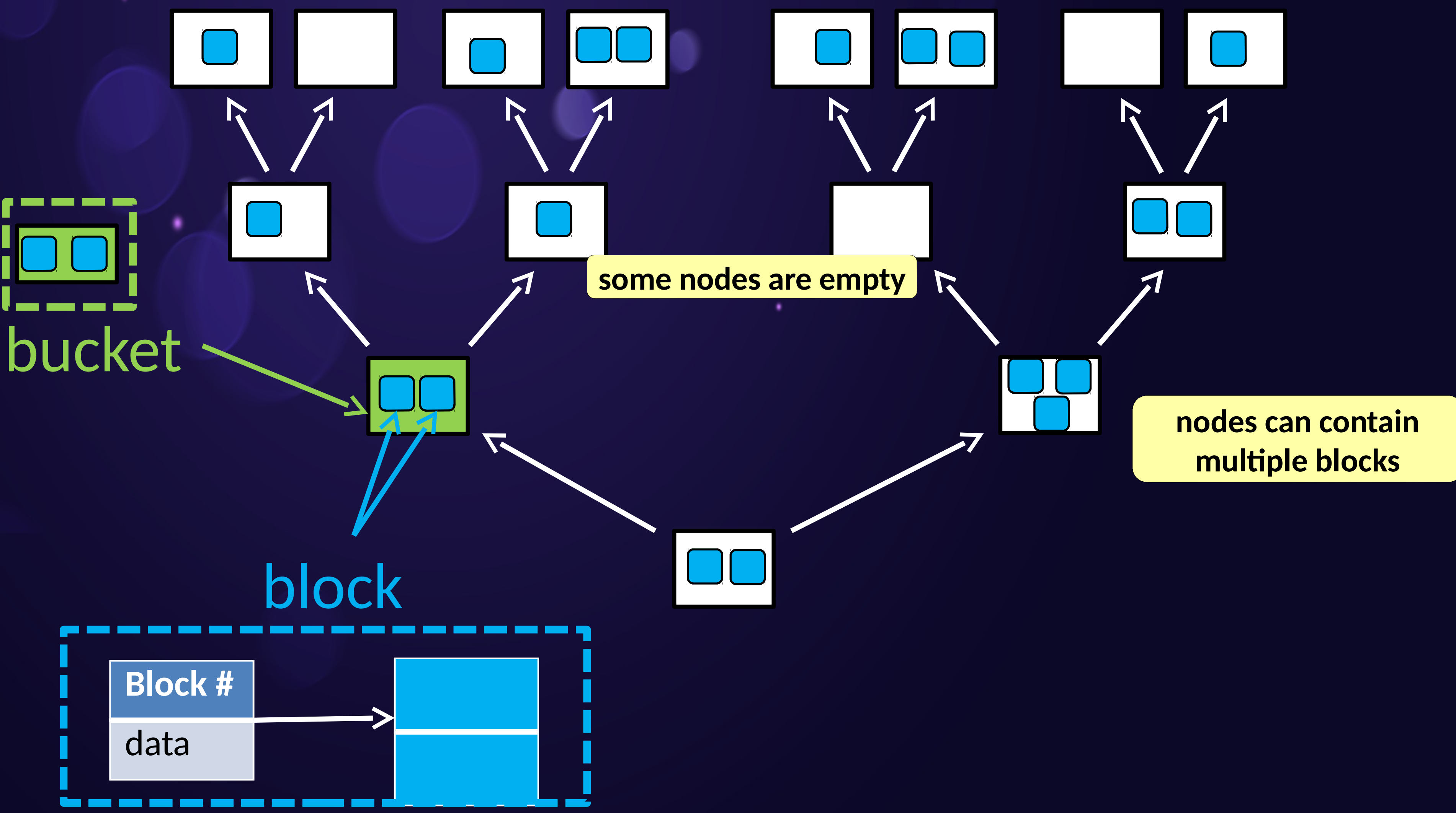
  Blocks of data

- Client storage

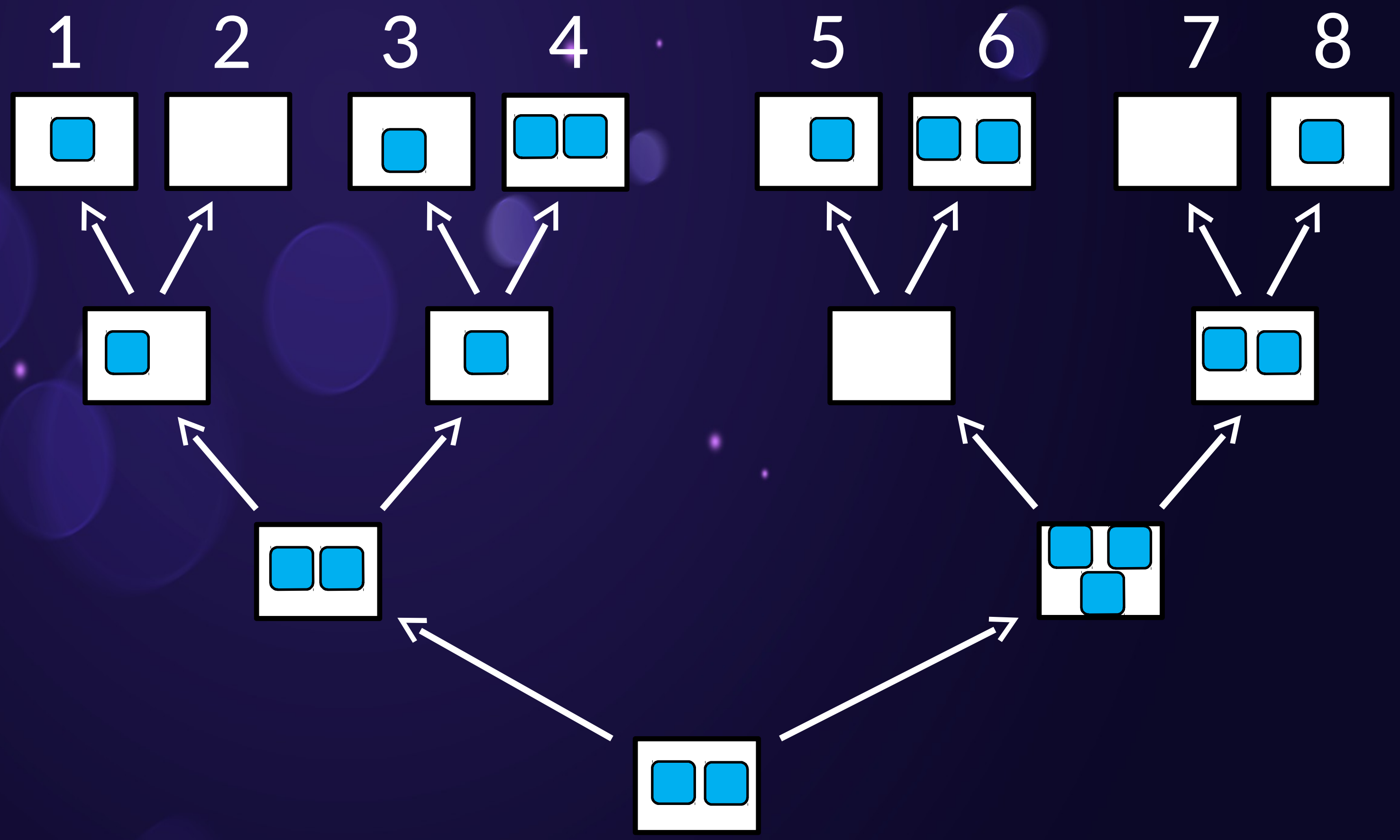
  position map, stash

Path ORAM

Server: data blocks stored in nodes of a tree

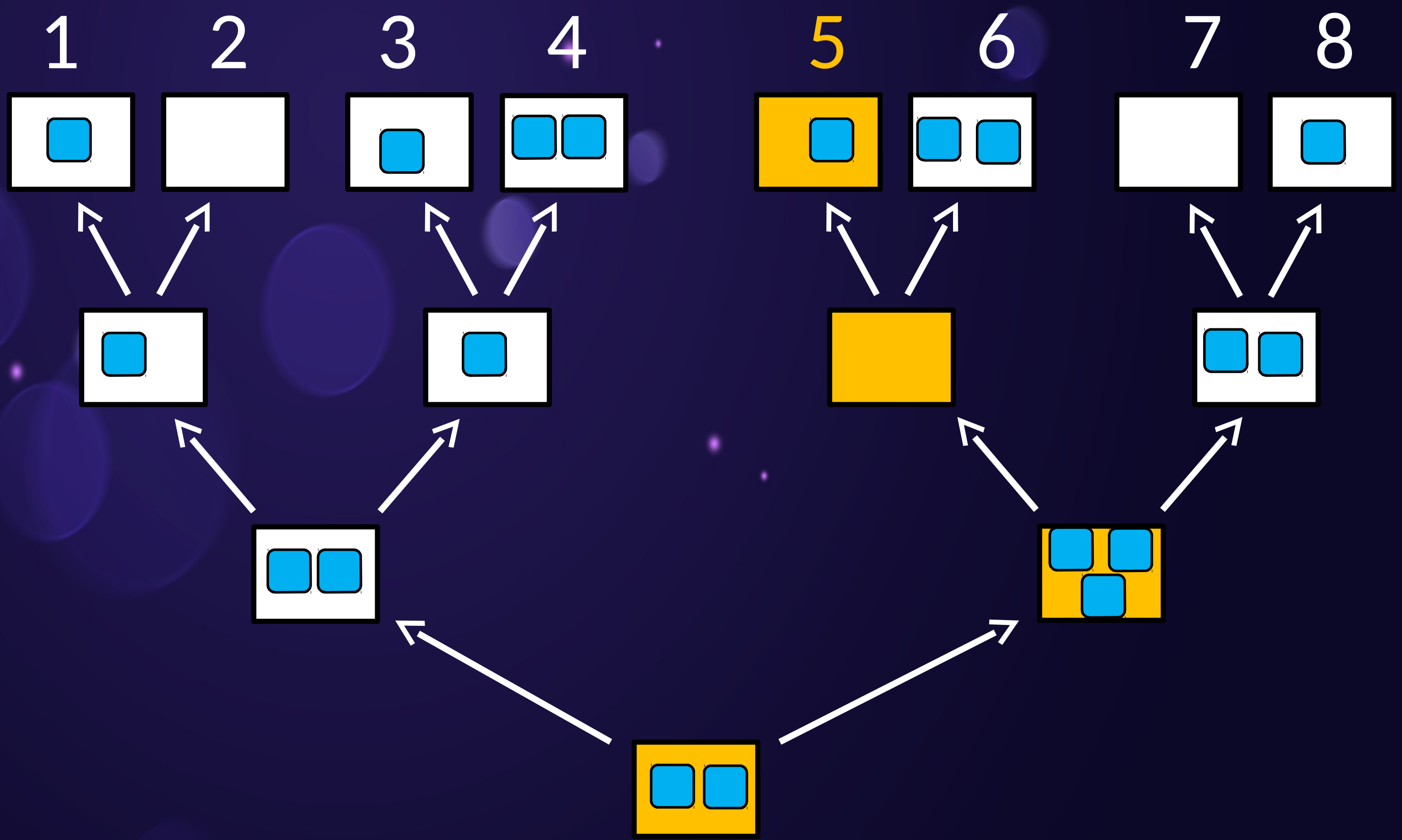


Path ORAM



"Position" of a block is the leaf sequence number.

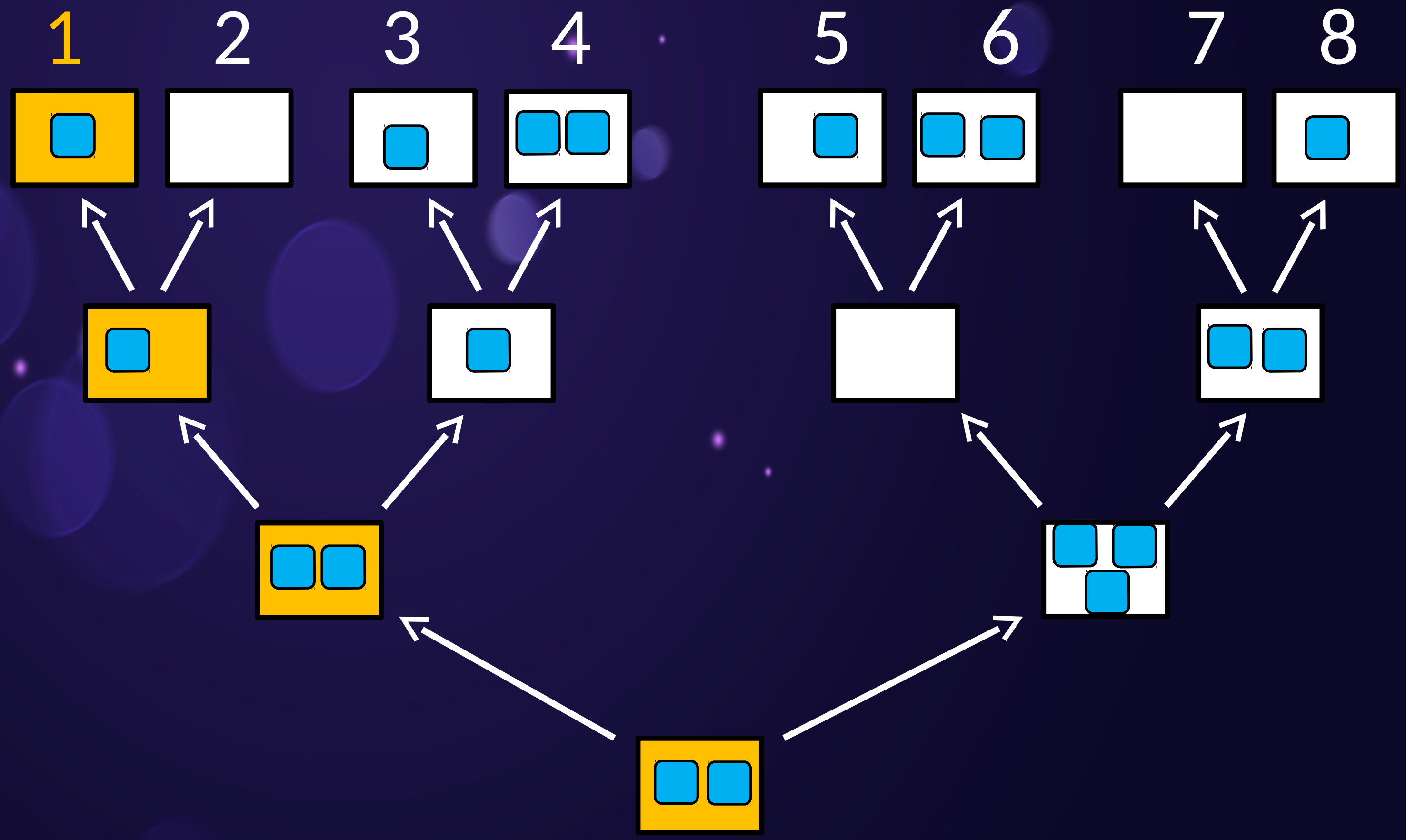
Path ORAM



A block must be placed on the path leading to its "position"

If a BLOCK's position is 5, it can only be in the this path

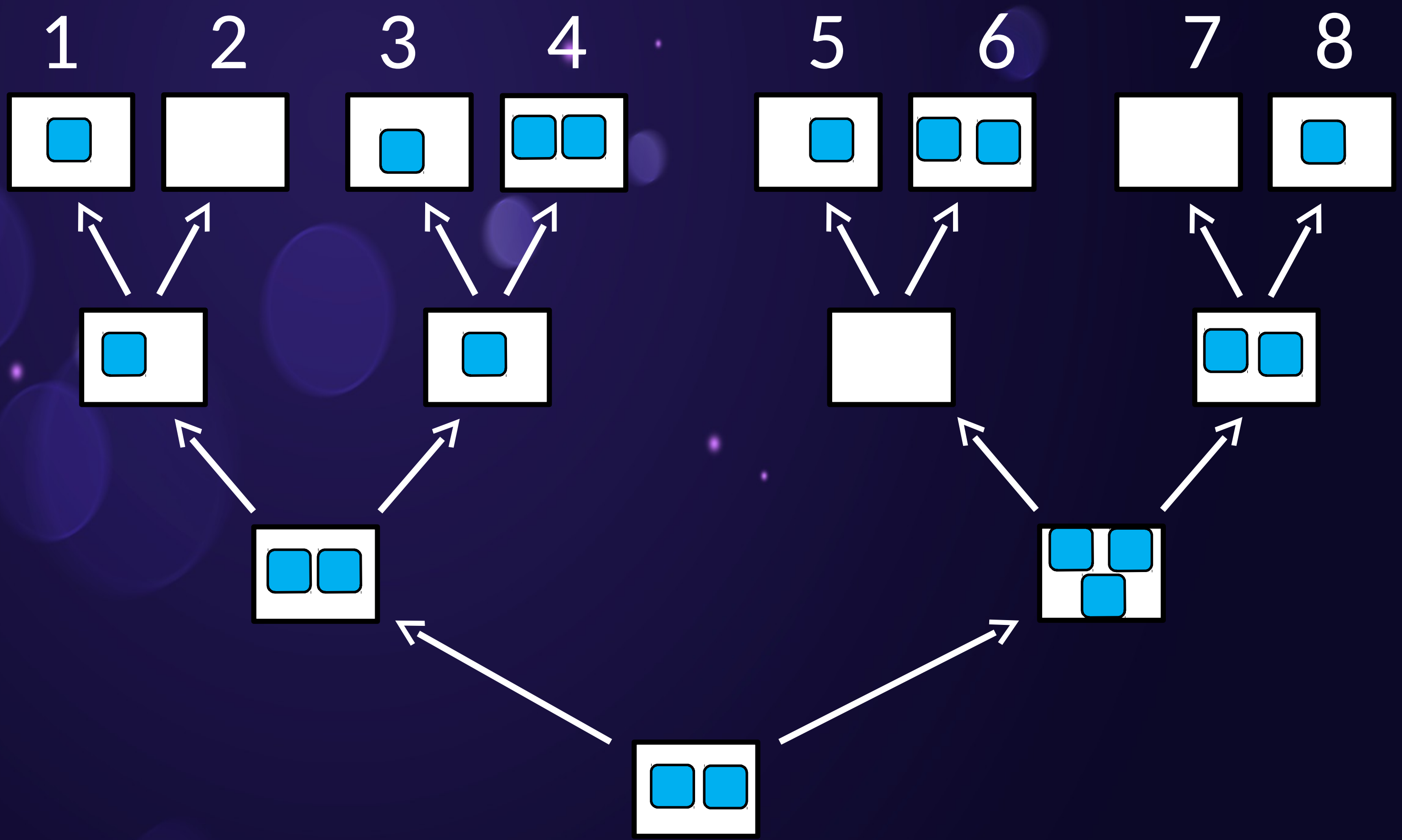
Path ORAM



A block must be placed on the path to its “position”

If a BLOCK’s position is 1, it can only be in the this path

Path ORAM



A block's position is chosen randomly from 1 to 2^L(number of leaves).

Some nodes(buckets) have multiple blocks while some are empty.

Path ORAM

Client storage: position map & stash

position map: to store each block's "position" (assume there are N blocks in the tree)

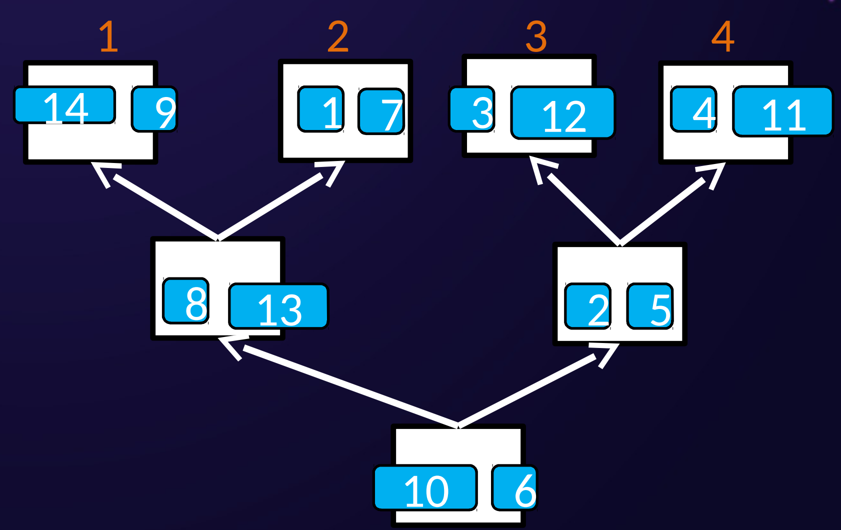
Block #	1	2	… … …. ….	N-1	N
Block's "position"	5	4	… … …. ….	1	7

Path ORAM

Client storage: position map & stash

position map: to store each block's "position"
(assume there are N blocks in the tree)

Block #	1	2	… … …. ….	13	14
Block's "position"	2	4		1	1

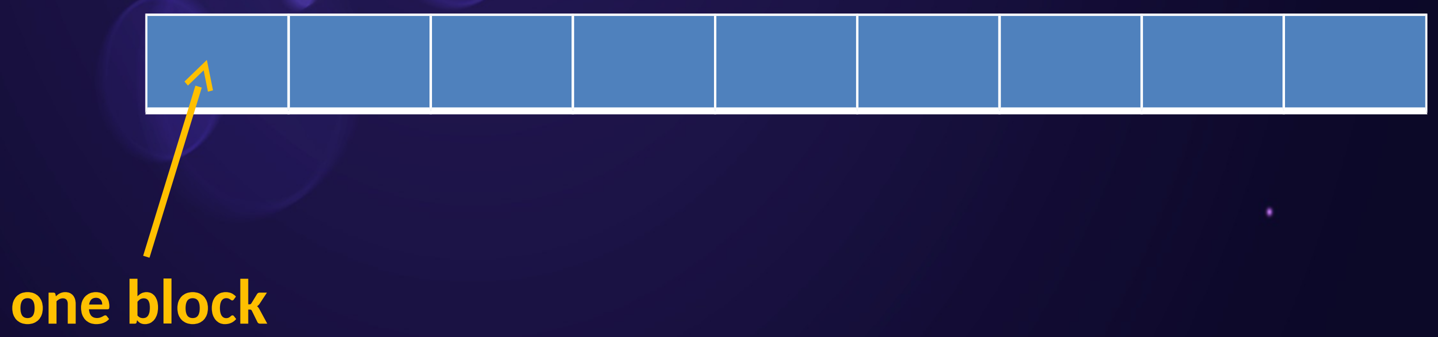


Path ORAM

Client storage: position map & stash

Stash: 1. block cache

2. stores blocks “overflowed” from the server



Path ORAM

16 lines of pseudocode

Access(op, a, data*):

```
x ← position[a]
position[a] ← UniformRandom(0 … 2^L − 1)
for ℓ ∈ {0, 1, …, L} do
    S ← S ∪ ReadBucket(𝒫(x, ℓ))
end for
data ← Read block a from S
if op = write then
    S ← (S − {(a, data)}) ∪ {(a, data*)}
end if
for ℓ ∈ {L, L − 1, …, 0} do
    S′ ← {(a′, data′) ∈ S : 𝒫(x, ℓ) = 𝒫(position[a′], ℓ)}
    S′ ← Select min(|S′|, Z) blocks from S′.
    S ← S − S′
    WriteBucket(𝒫(x, ℓ), S′)
end for
return data
```

Path ORAM

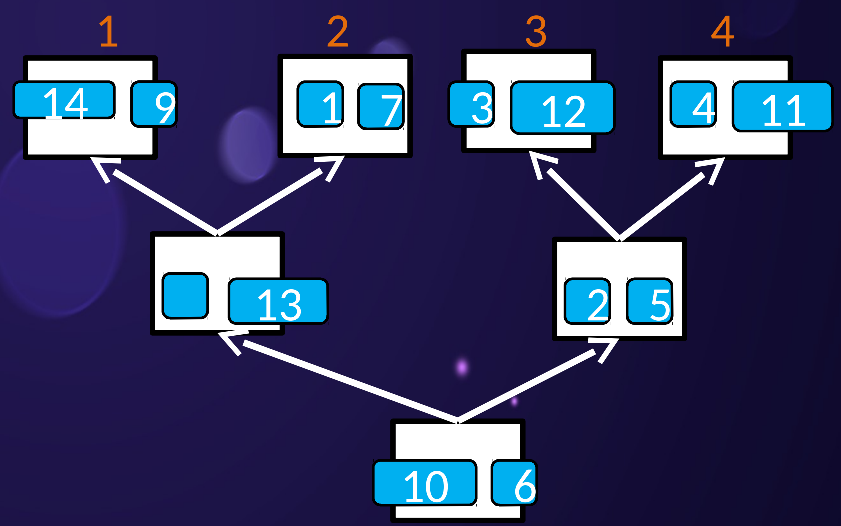
An example for access (op, a, data)

Example:

A user wants to modify block "7"

Path ORAM

Example: A user wants to modify block “7”



Server

Client

position map

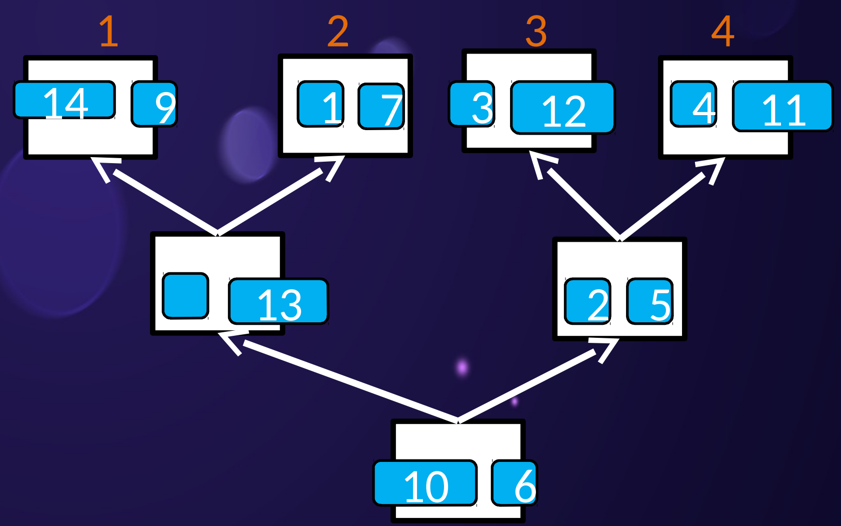
1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	2	2	1	1	4	3	2	1

stash

8	9							

Path ORAM

Example: A user wants to modify block "7"



Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	2	2	1	1	4	3	2	1

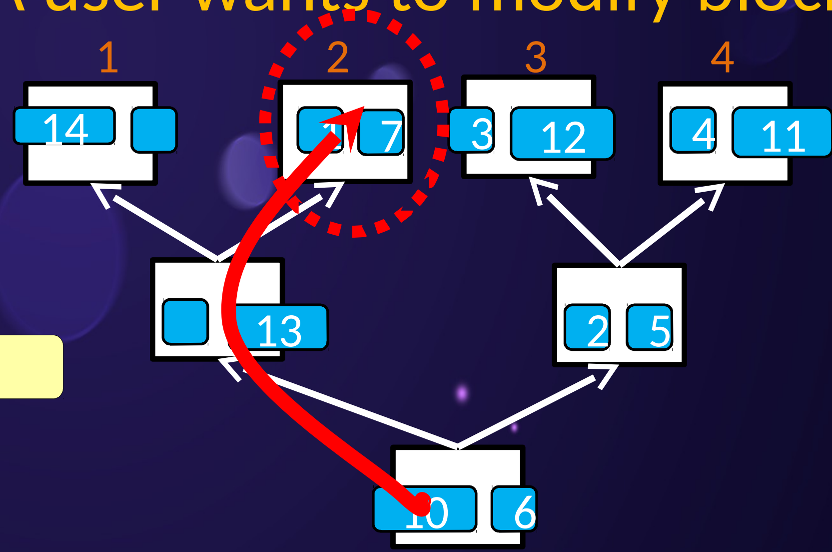
1. Lookup block's position

stash

8	9							

Path ORAM

Example: A user wants to modify block “7”



2. Read the entire path

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	2	2	1	1	4	3	2	1

1 7 13 10 6

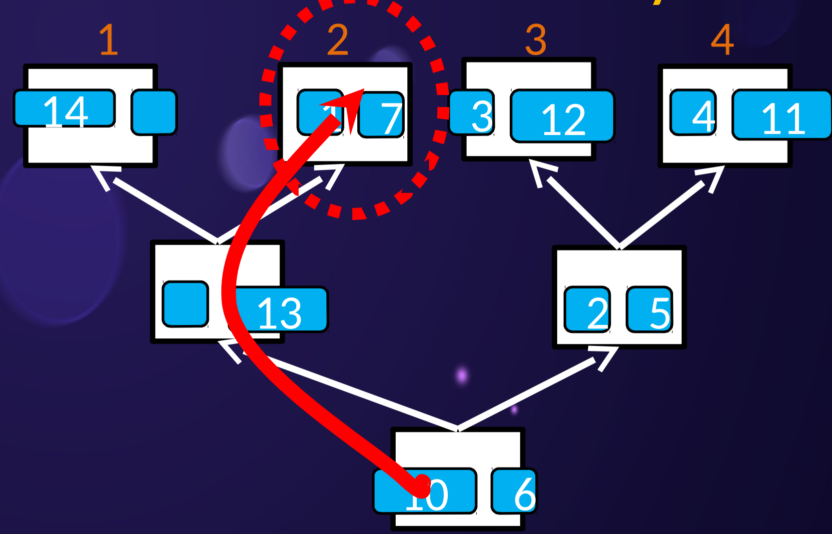
decrypt ↓

stash

8	9	1	7	13	10	6		

Path ORAM

Example: A user wants to modify block "7"



Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	2	2	1	1	4	3	2	1

1 7 13 10 6

3. Client can now read/modify data in block

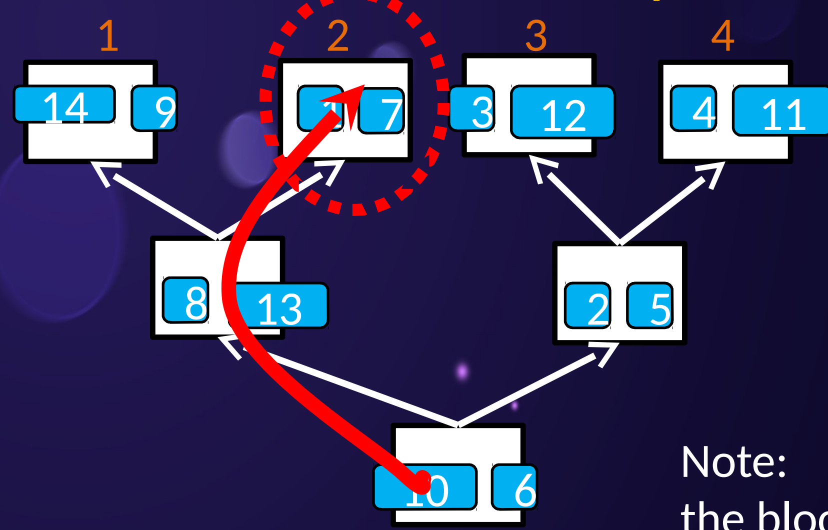
decrypt ↓

stash

8	9	1	7	13	10	6		

Path ORAM

Example: A user wants to modify block “7”



Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	2	2	1	1	4	3	2	1

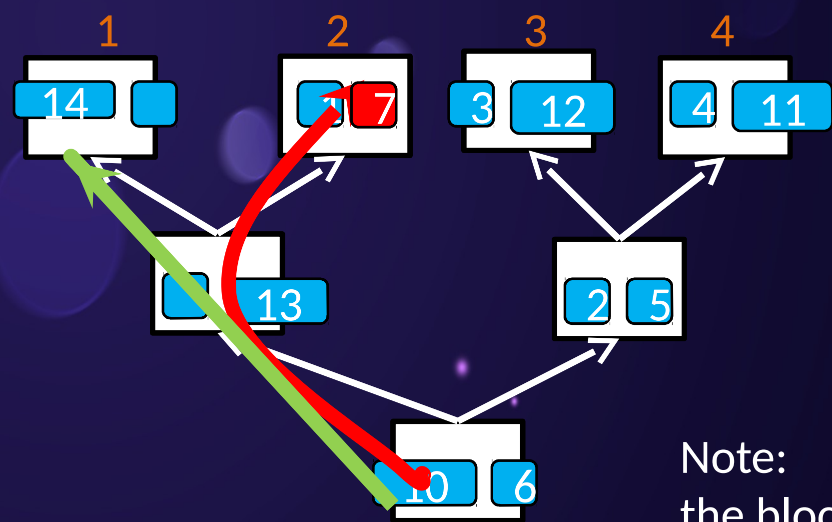
4. Assign a new random position.

Position[block 7] = UniformRandom(1, 2, 3, 4)

here = 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

Path ORAM



Note:
the blocks on the tree are encrypted

Server

Client

5. Write path back

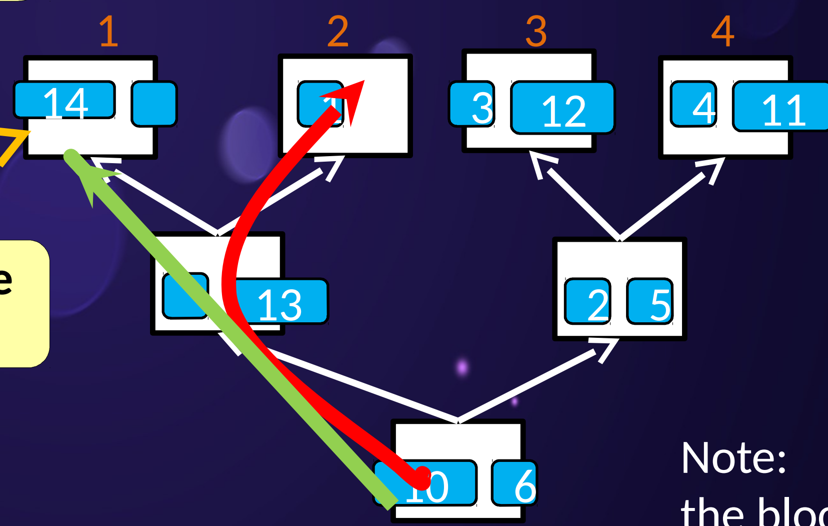
Principles:

A. Blocks should be pushed down as deep as possible
B. Blocks should always be on the path to its position

Path ORAM

**Principle A:
Push blocks as deep as possible**

Try the deepest possible node first



Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

5. Write path back.

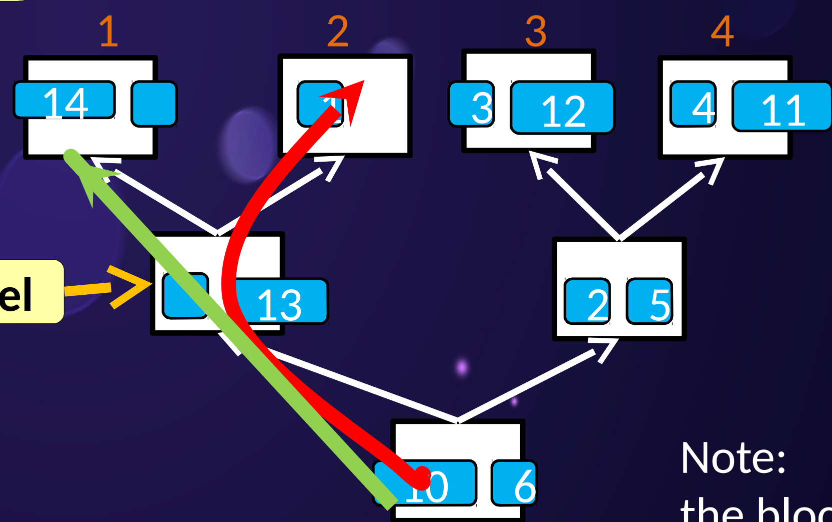
stash

8	9	1	7	13	10	6		

Path ORAM

**Principle A:
Push blocks as deep as possible**

If full, try the upper level



Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

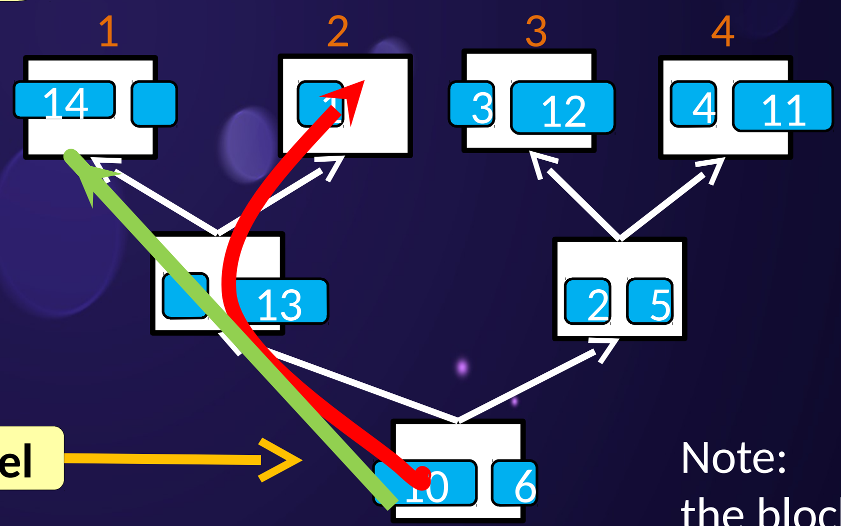
5. Write path back.

stash

8	9	1	7	13	10	6		

Path ORAM

Principle A:
Push blocks as deep as possible



If full, try the upper level

Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

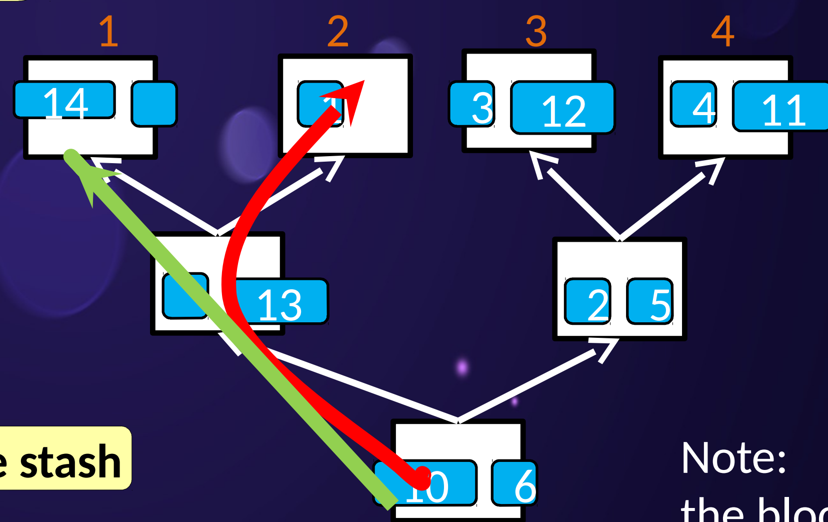
5. Write path back.

stash

8	9	1	7	13	10	6		

Path ORAM

Principle A:
Push blocks as deep as possible



If root is full, store in the stash

Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

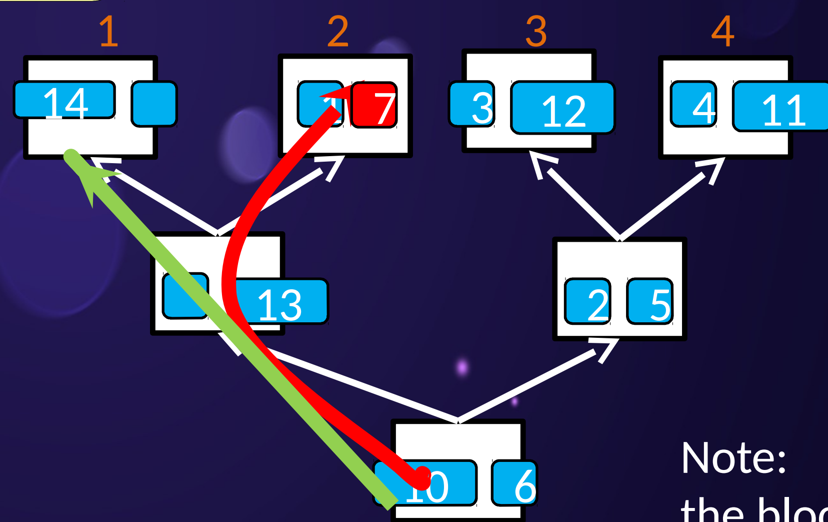
5. Write path back.

stash

8	9	1	7	13	10	6		

Path ORAM

Principle B:
The block should always be on its path to "position"



Note:
the blocks on the tree are encrypted

Server

Client

position map

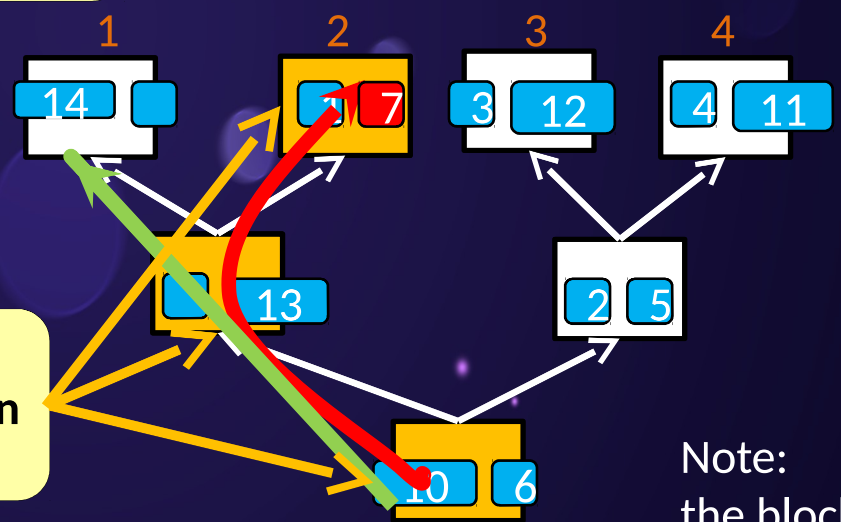
1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

5. Write path back

"positions"	2	1	2	1	2	1	1		
stash	8	9	1	7	13	10	6		

Path ORAM

Principle B:
The block should always be on its path to "position"



Block 8, 1, 13 can end up at any buckets on the entire path

Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

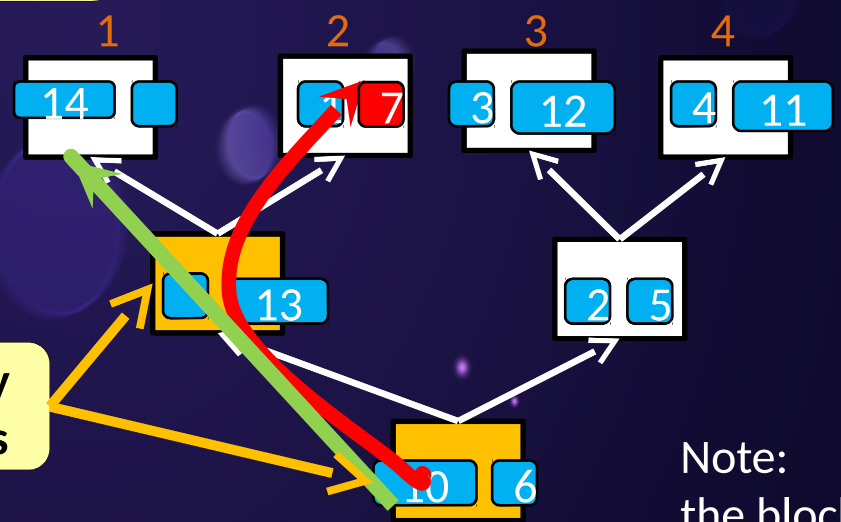
5. Write path back

stash

2	1	2	1	2	1	1		
8	9	1	7	13	10	6		

Path ORAM

Principle B:
The block should always be on its path to “position”



Block 9, 7, 10, 6 can only end up at these buckets

Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

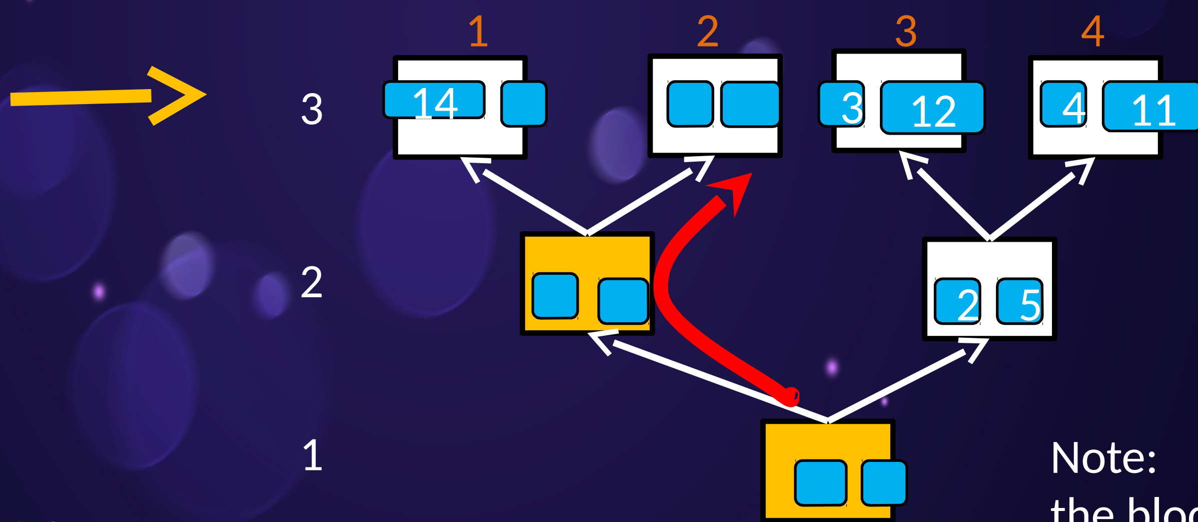
5. Write path back

stash

2	1	2	1	2	1	1		
8	9	1	7	13	10	6		

Path ORAM

Get back to our example:



Note:
the blocks on the tree are encrypted

Server

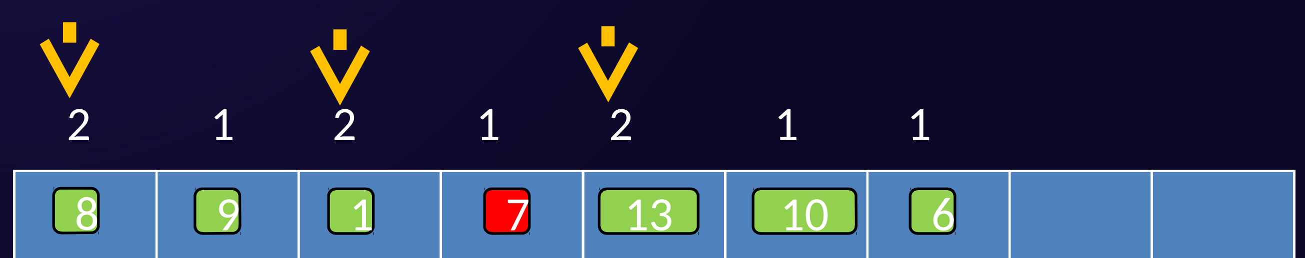
Client

position map

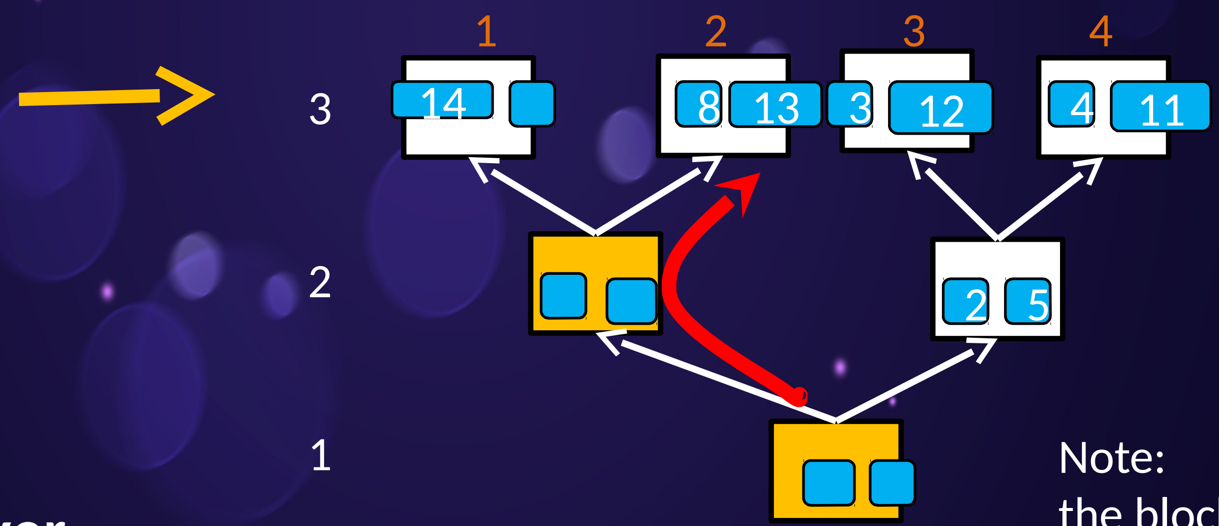
1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

5. Write path back

stash



Path ORAM



Note:
the blocks on the tree are encrypted

Server

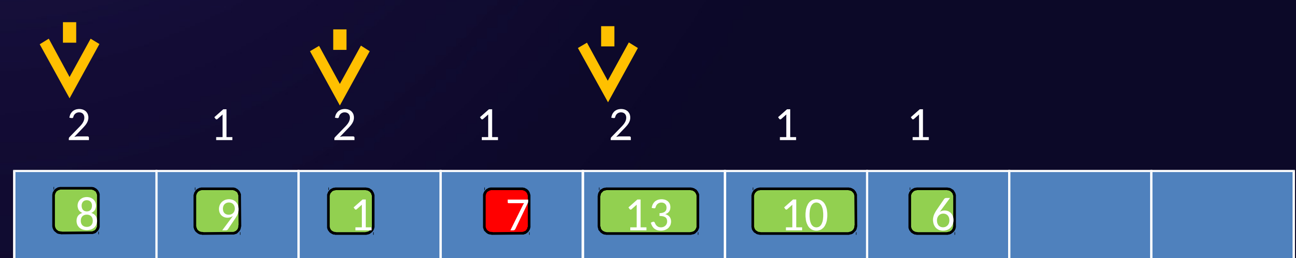
Client

position map

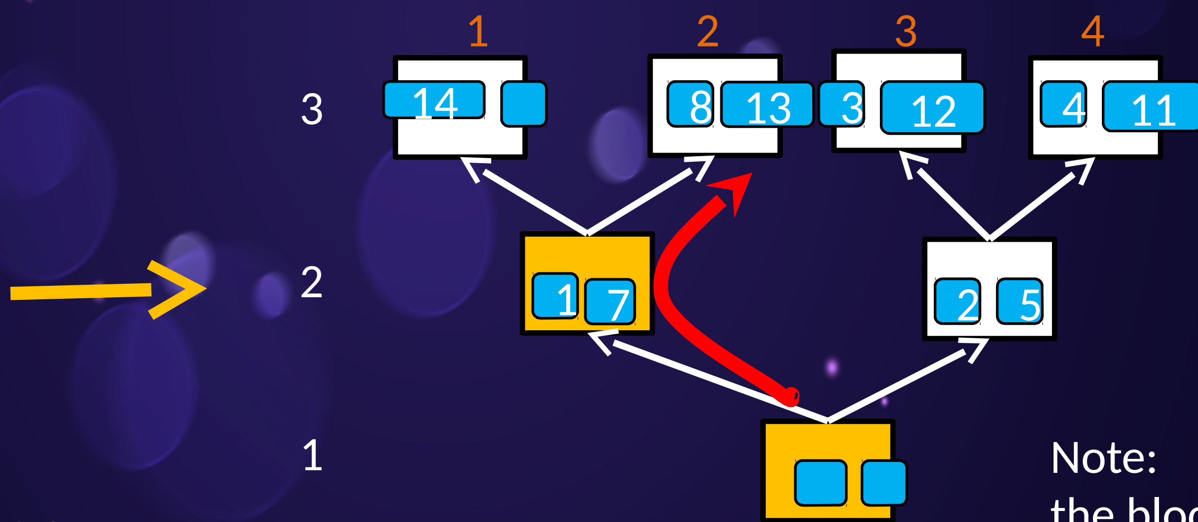
1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

5. Write path back

stash



Path ORAM



Note:
the blocks on the tree are encrypted

Server

Client

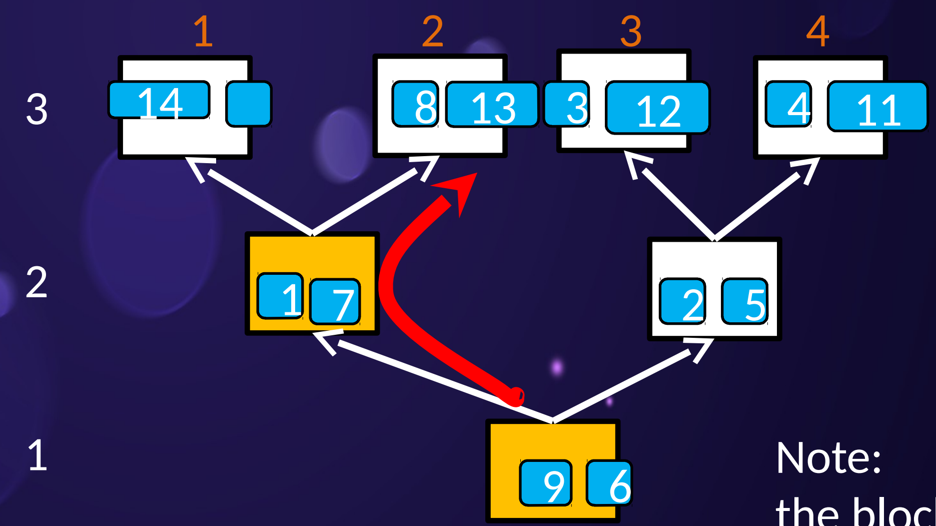
position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

5. Write path back

stash

Path ORAM



Note:
the blocks on the tree are encrypted

Server

Client

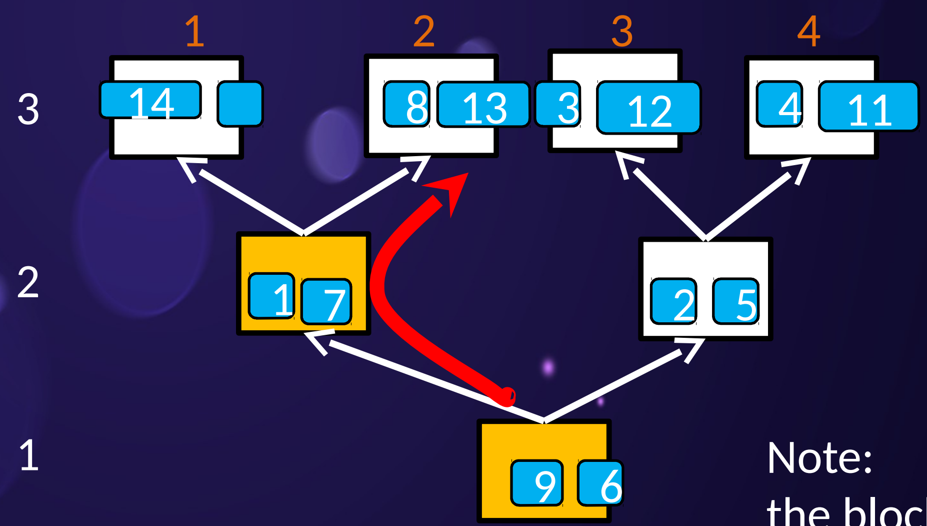
position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

5. Write path back

stash

Path ORAM



Note:
the blocks on the tree are encrypted

Server

Client

position map

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	4	3	4	3	1	1	2	1	1	4	3	2	1

5. Write path back

stash

10								

Client storage

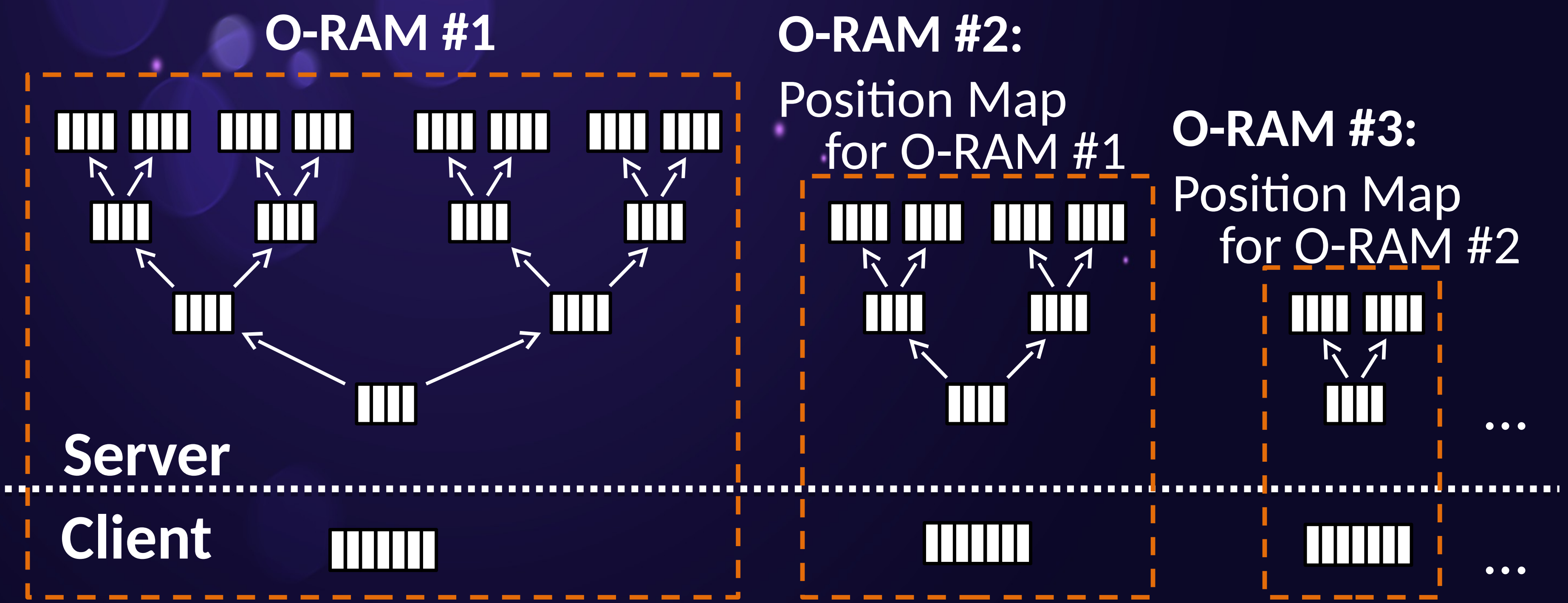
Position map:

- Log N bits per block
- Totally N log N bits

Too much !!!

Recursion

- Store the position map in another ORAM.
- Do this recursively.



Theorems

1. Given two accesses, an adversary that controls the network and the cloud service provider cannot see what blocks were read or written, or if the two accesses referred to the same block or different blocks.
2. The stash is very unlikely to get more than a few blocks in it.