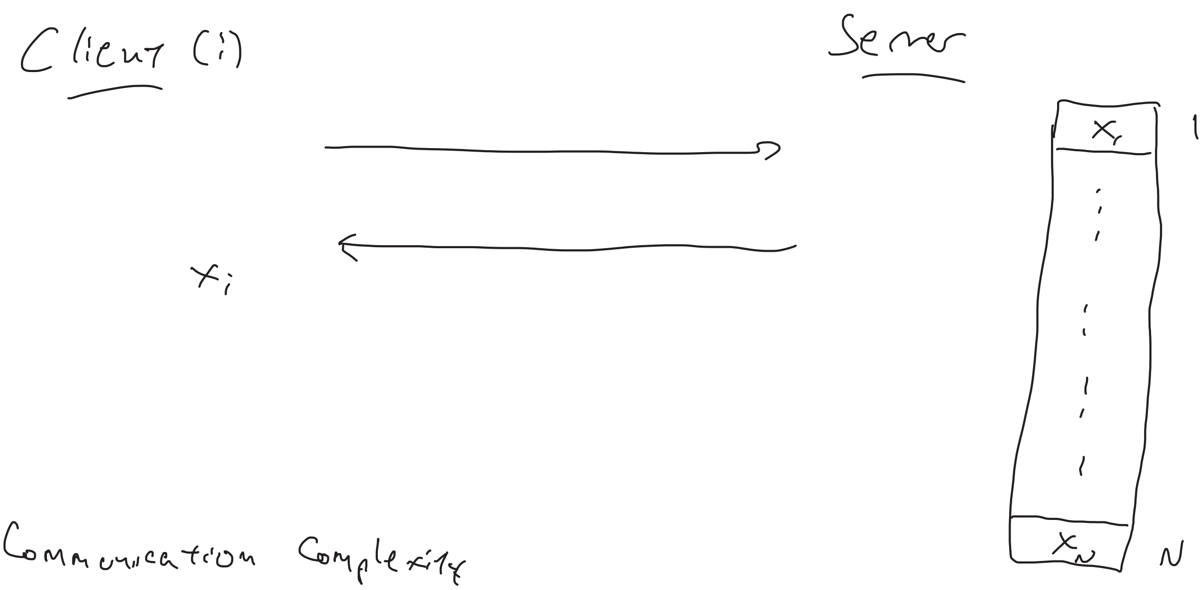
- Scribes?
- lecture recording



1) Communication Complexity
2) # of memory accesses by the server
3) Support for writes

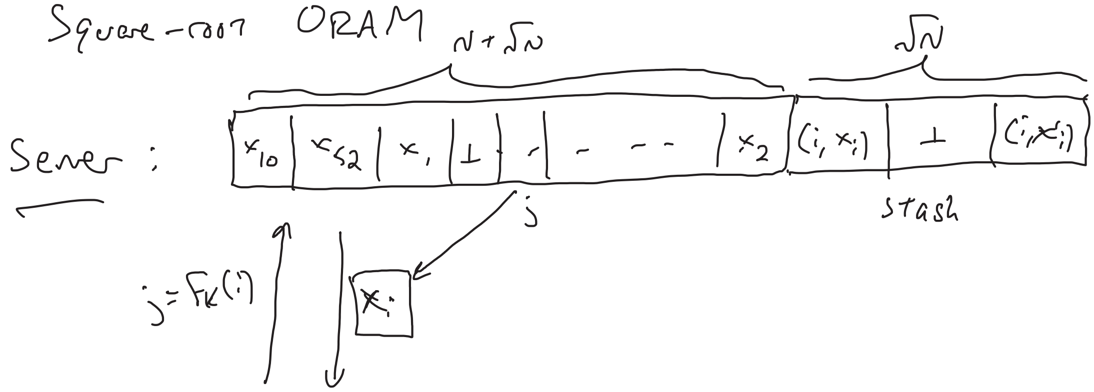
PIR - private information retrieval

- read-only memory
- two-round protocols
- server stores data in cleartext
- single server, computational security
- multiple servers, each storing a copy of the data, information-theoretic security (assuming servers do not collude)

Oblivious RAM (ORAM) Goldreich–Ostrovsky

- interactive protocols
- single-server setting (primarily)
- read/write access
- server updates its storage as part of the protocol

Square-root ORAM



Client:

- choose key K defining pseudorandom permutation over $\{1, \dots, N + \sqrt{N}\}$
- shuffle data so x_i is stored at position $F_K(i)$

To read the value at position i:

- scan the entire stash to see if $\exists$ entry $(i, *)$
  - if so, accept right-most such value
- if no such entry in the stash
  - request from the server the data stored @ position $F_K(i)$
  - write x_i into next available location in stash
- if there was an entry in the stash
  - request from the server the data stored @ position $F_K(N + ctr)$, ctr++
  - write $\perp$ into next available location in the stash

After $\sqrt{N}$ accesses, client needs to refresh data stored at the server

$$\left| (F_K(i), x_i) \,\middle|\, (F_K(i'), x_{i'}) \,\middle|\, \ldots \,\middle|\; \right|$$

Oblivious sorting: in theory, can be done using $O(n \log n)$ swaps
in practice, done using $O(n \log^2 n)$ swaps

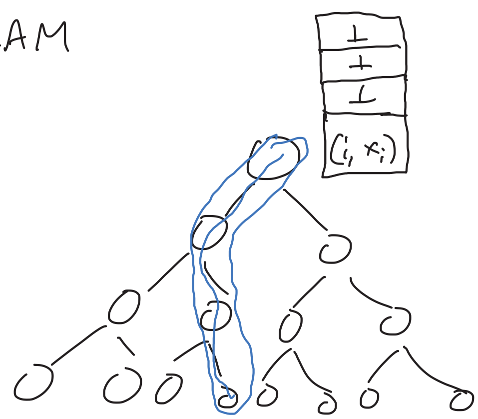
Complexity:

- ignoring refresh, Communication = # memory accesses $= O(\sqrt{n})$
- refresh: Communication $= O(n \log n)$
  amortized comm. $= O\left(\frac{n \log n}{\sqrt{n}}\right) = O(\sqrt{n} \log n)$

$\Longrightarrow$ overall, amortized complexity $O(\sqrt{n} \log n)$

Path ORAM

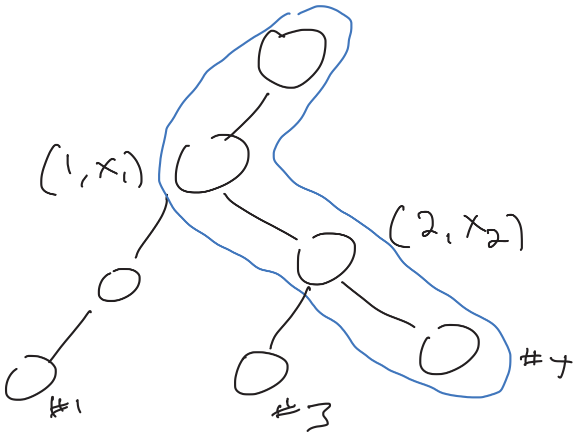
<u>Server</u>



<u>Client</u>

- maintain a mapping M from $\{1, \ldots, N\}$ to leaves of tree
- element x_i will be stored at some node on the path from the root to $M[i]$
- To read the value at index i,
  - Request from server all nodes on the path to $M[i]$

- Choose fresh random value for $M(i)$
- Write back the values to the same path
  - push down all values as far down the path as possible
  - eliminate any duplicates; keep freshest value



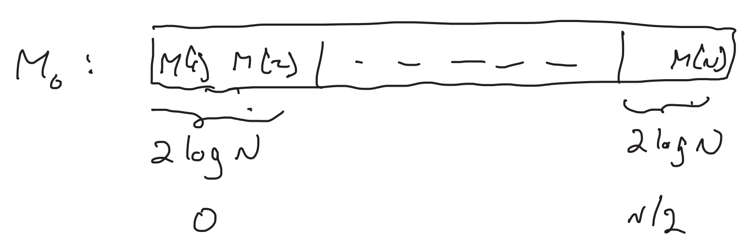
$M(1) = \cancel{4}\ 1$

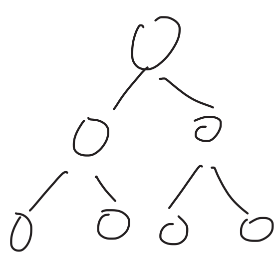
$M(2) = 3$

if $|x_i| = 2 \log N$, and tree has N leaves, then storing M_0 requires $N \log N$ bits

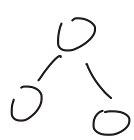
$\Longrightarrow$ Factor of 2 less than trivial

recurse! store M_0 using a smaller version of the scheme





storing $x_1, \ldots, x_N$



storing M_0

. . . . .