Oblivious RAM, Continued

CS 598 DH

Today's objectives

See a more practical construction of ORAM

Prove ORAM lower bound

Setting

Semi-honest Security

GMW
Compiler

Malicious Security

Zero Knowledge

General-Purpose Tools

GMW Protocol

Multi-party

Multi-round

Garbled Circuit

Constant Round

Two Party

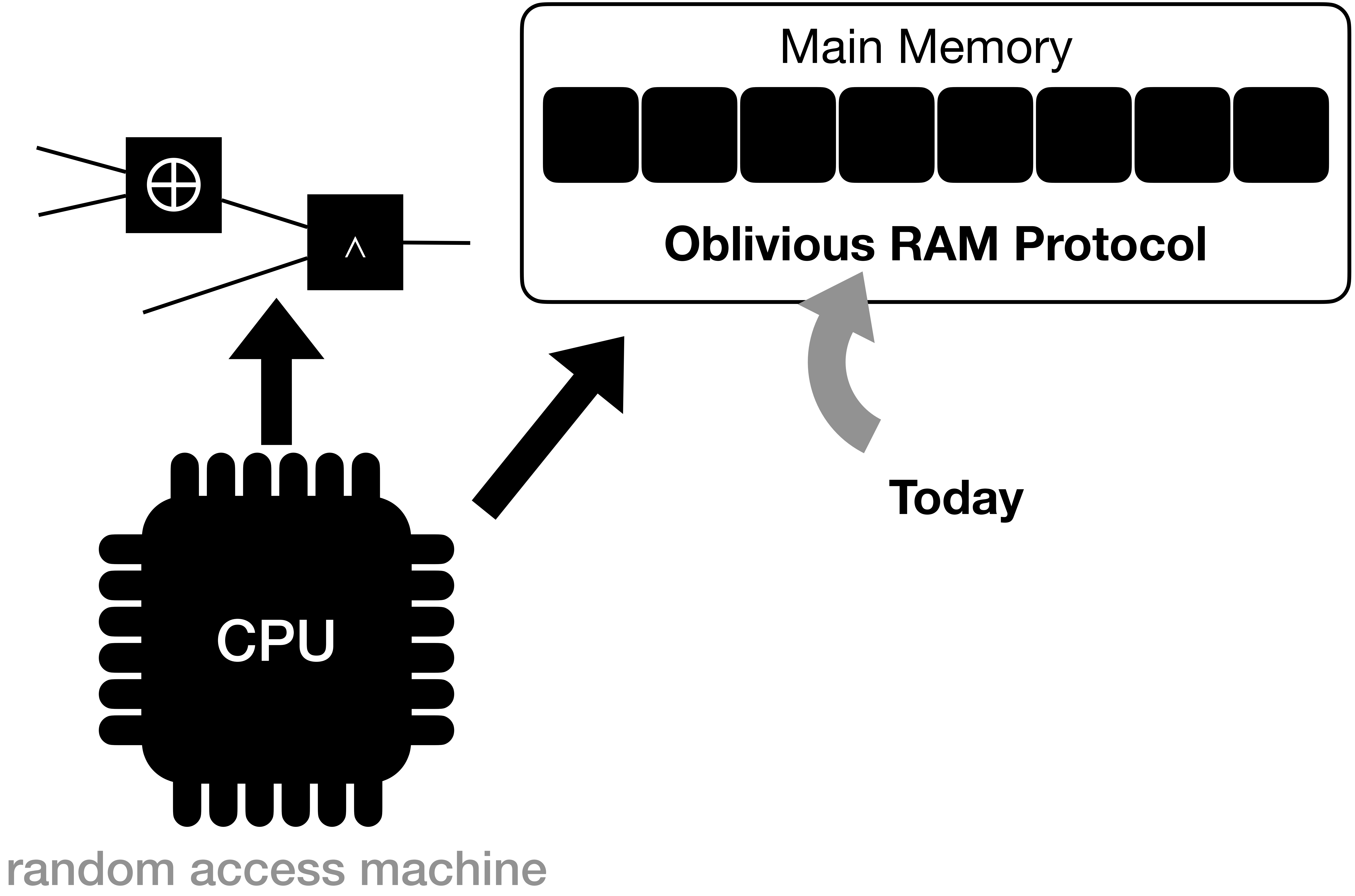
Primitives

Oblivious Transfer

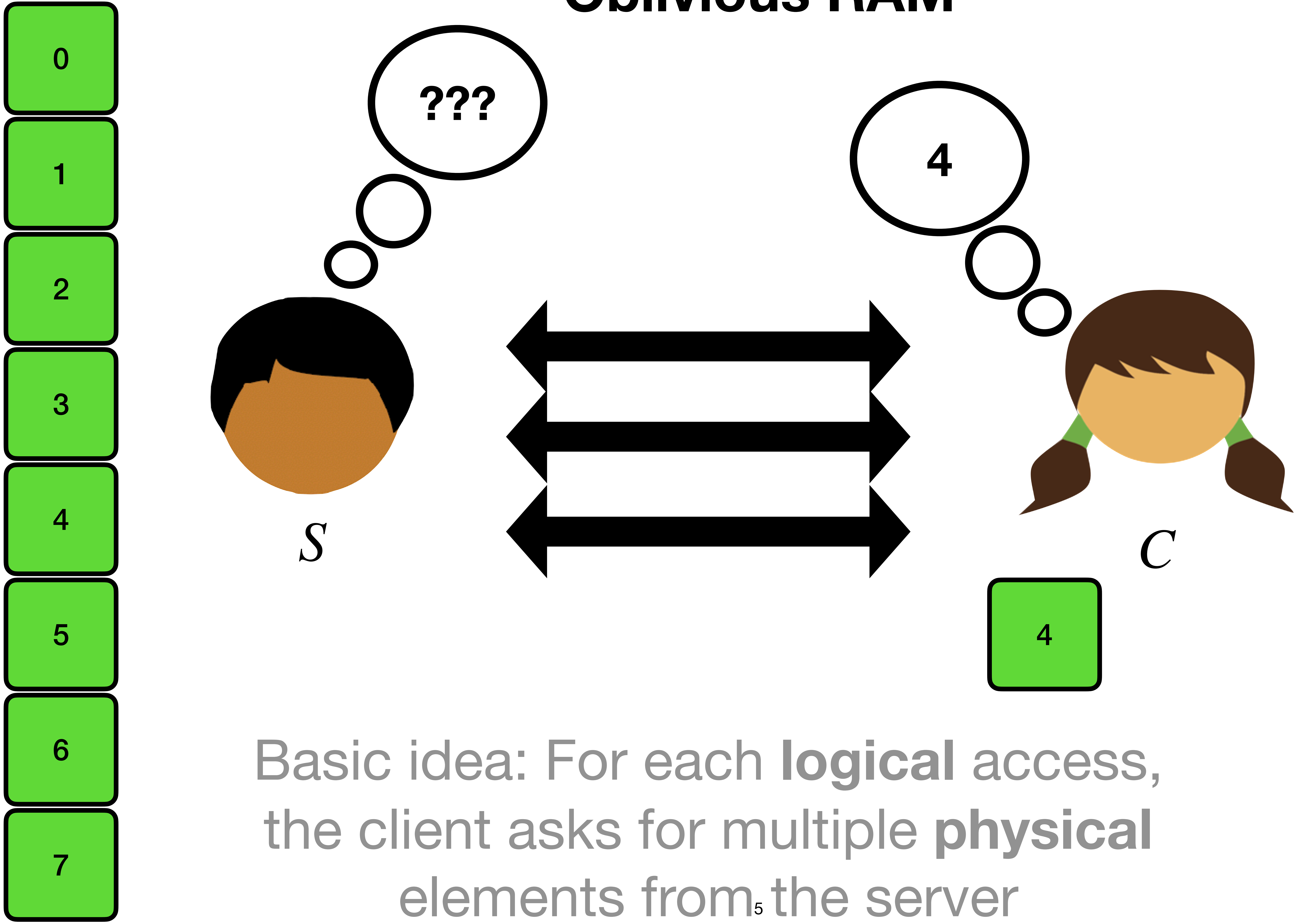
Pseudorandom functions/encryption

Commitments

ORAM

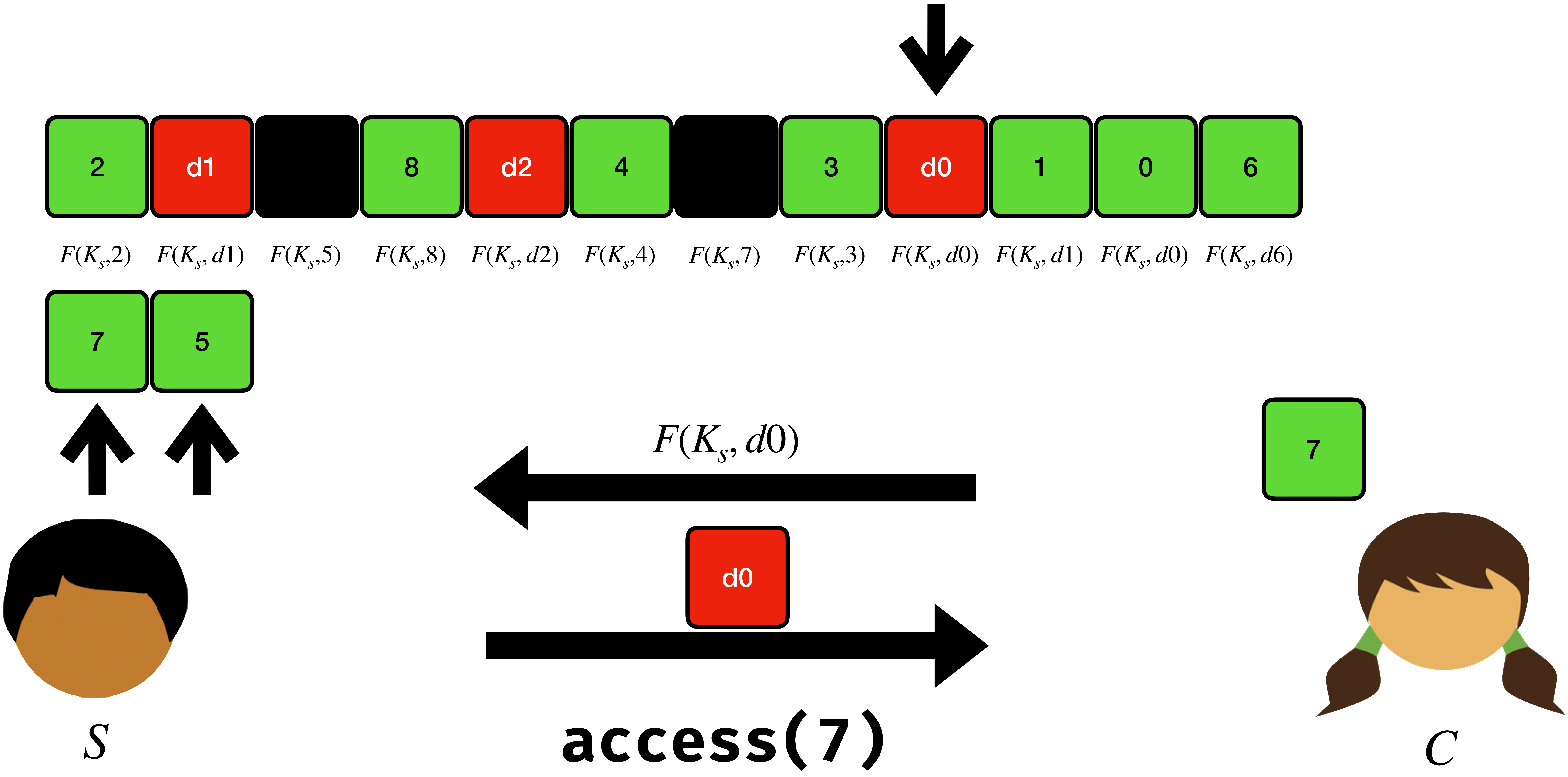

Main Memory
Oblivious RAM Protocol
Today
CPU
random access machine

Oblivious RAM

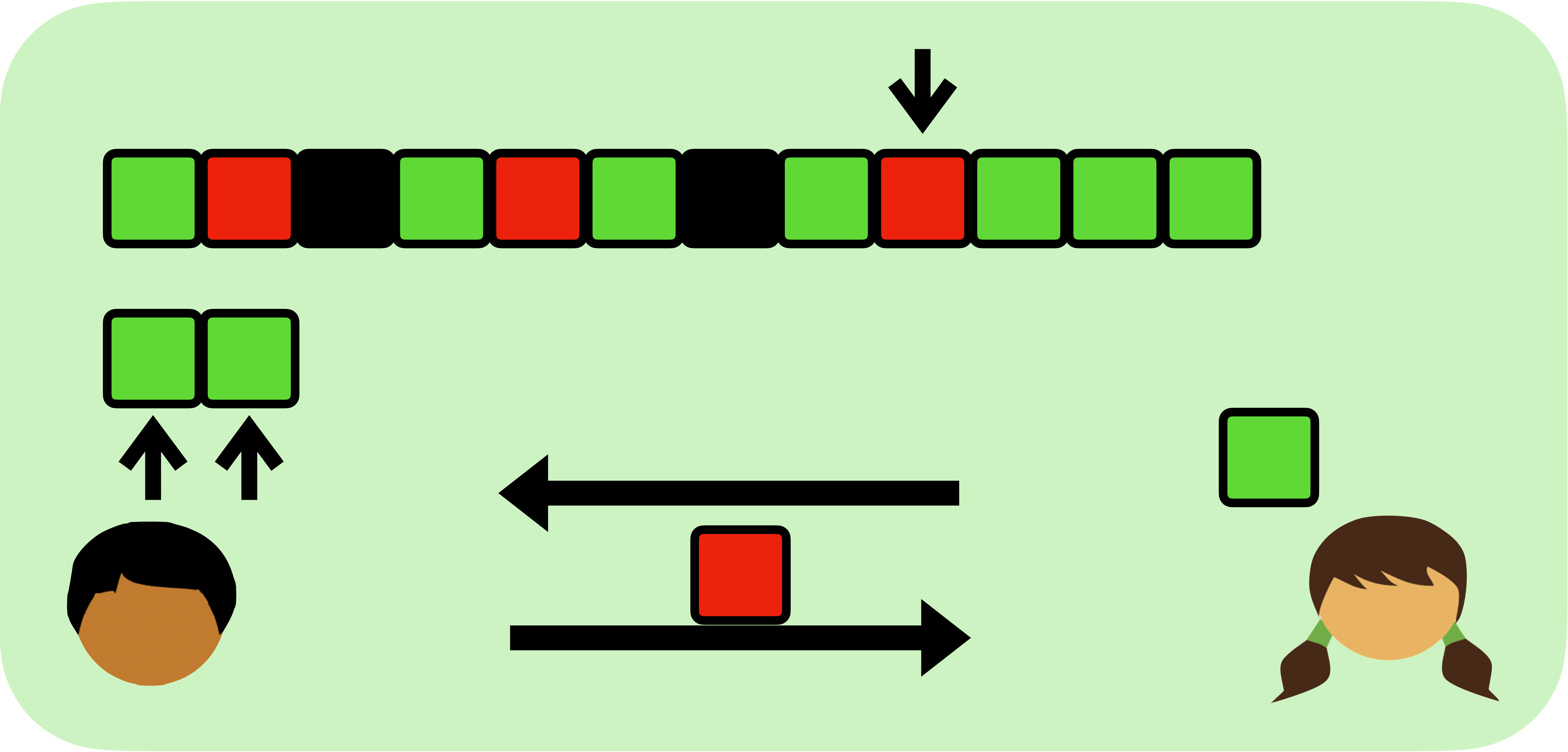


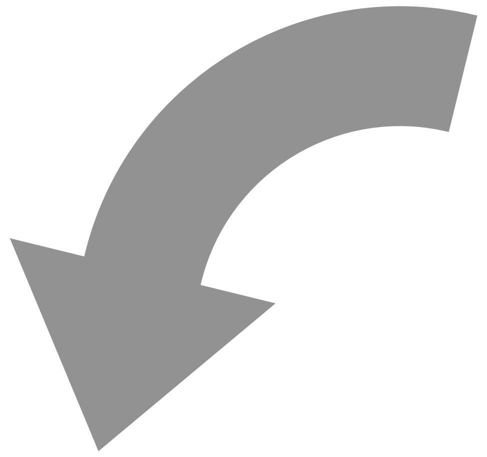
Basic idea: For each **logical** access, the client asks for multiple **physical** elements from the server

Square Root ORAM (Ostrovsky '92)



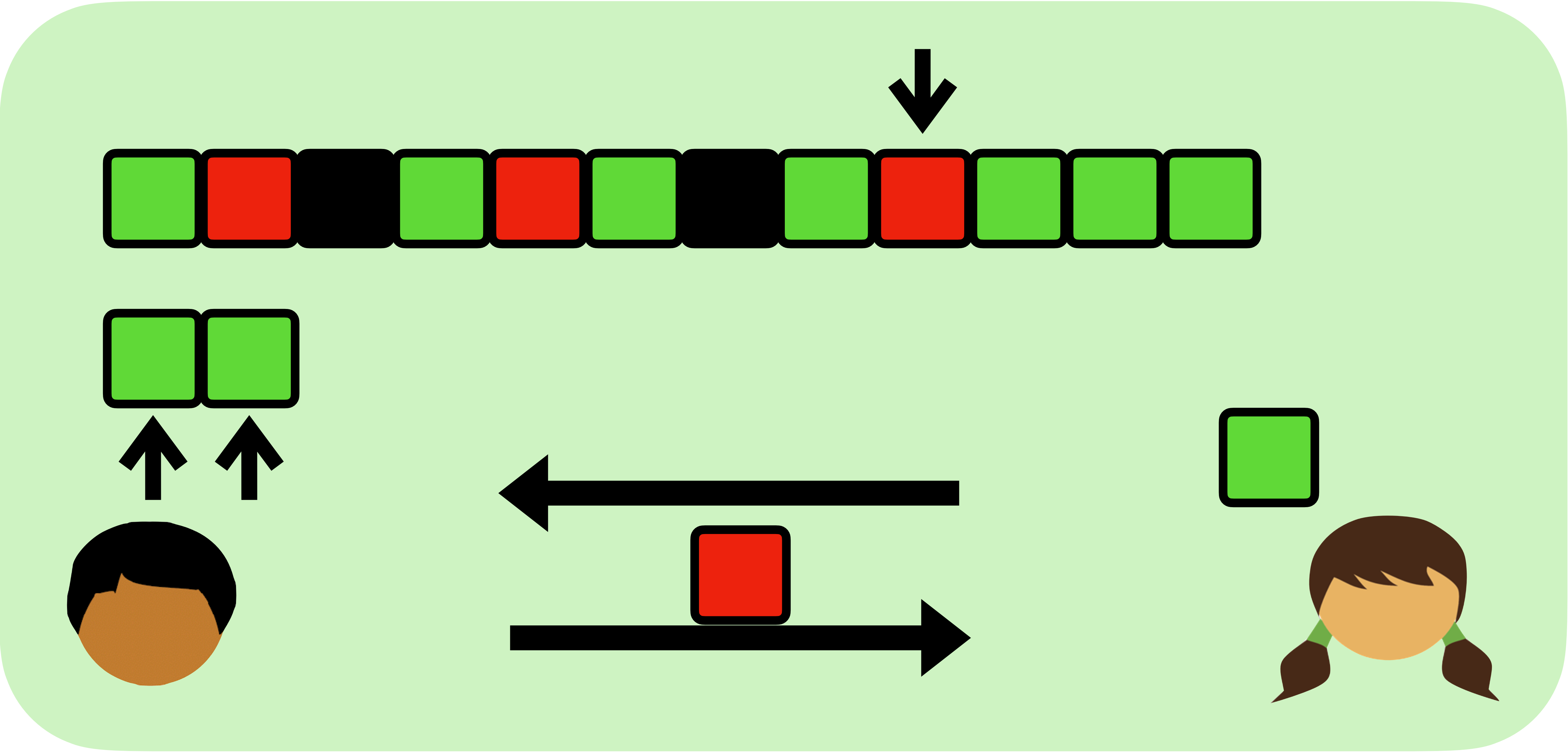
Square Root ORAM (Ostrovsky '92)





For every **logical** access, the server sends to the client amortized $\tilde{O}(\sqrt{n})$ **physical** elements

Square Root ORAM (Ostrovsky '92)



For every **logical** access, the server sends to the client amortized $\tilde{O}(\sqrt{n})$ **physical** elements

Natural question: How low can we go in terms of overhead?

$O(\log^2 n)$ physical accesses

Path ORAM: An Extremely Simple Oblivious RAM Protocol

18

EMIL STEFANOV, UC Berkeley
MARTEN VAN DIJK, University of Connecticut
ELAINE SHI, Cornell University
T.-H. HUBERT CHAN, University of Hong Kong
CHRISTOPHER FLETCHER, University of Illinois at Urbana-Champaign
LING REN, XIANGYAO YU, and SRINIVAS DEVADAS, MIT CSAIL

We present Path ORAM, an extremely simple Oblivious RAM protocol with a small amount of client storage. Partly due to its simplicity, Path ORAM is the most practical ORAM scheme known to date with small client storage. We formally prove that Path ORAM has a $O(\log N)$ bandwidth cost for blocks of size $B = \Omega(\log^2 N)$ bits. For such block sizes, Path ORAM is asymptotically better than the best-known ORAM schemes with small client storage. Due to its practicality, Path ORAM has been adopted in the design of secure processors since its proposal.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Algorithms, Security

Additional Key Words and Phrases: Oblivious RAM, ORAM, Path ORAM, access pattern

ACM Reference format:
Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: An Extremely Simple Oblivious RAM Protocol. *J. ACM* 65, 4, Article 18 (April 2018), 26 pages.
https://doi.org/10.1145/3177872

A conference version of the article has appeared in ACM Conference on Computer and Communications Security (CCS), 2013.
This work is partially supported by the NSF Graduate Research Fellowship grants DGE-0946797 and DGE-1122374, the DoD NDSEG Fellowship, NSF grant CNS-1314857, DARPA CRASH program N66001-10-2-4089, and a grant from the Amazon Web Services in Education program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.
The research was supported in part by a grant from Hong Kong RGC under the contract HKU719312E.
Authors' addresses: E. Stefanov, Department of Electrical Engineering and Computer Sciences, UC Berkeley, CA 94720, USA; email: emil@berkeley.edu; M. V. Dijk, Electrical and Computing Engineering Department, University of Connecticut, Storrs-Mansfield, CT 06269, USA; email: vandijk@engr.uconn.edu; E. Shi, Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, USA; email: elaine@cs.cornell.edu; T.-H. H. Chan, Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong; email: hubert@cs.hku.hk; C. Fletcher, Computer Science Department, University of Illinois–Urbana Champaign, Urbana, IL 61801, USA; email: cwfletch@illinois.edu. L. Ren, X. Yu, and S. Devadas, MIT CSAIL, Cambridge, MA 02139, USA; emails: {renling, yxy, devadas}@csail.mit.edu.

https://doi.org/10.1145/3177872

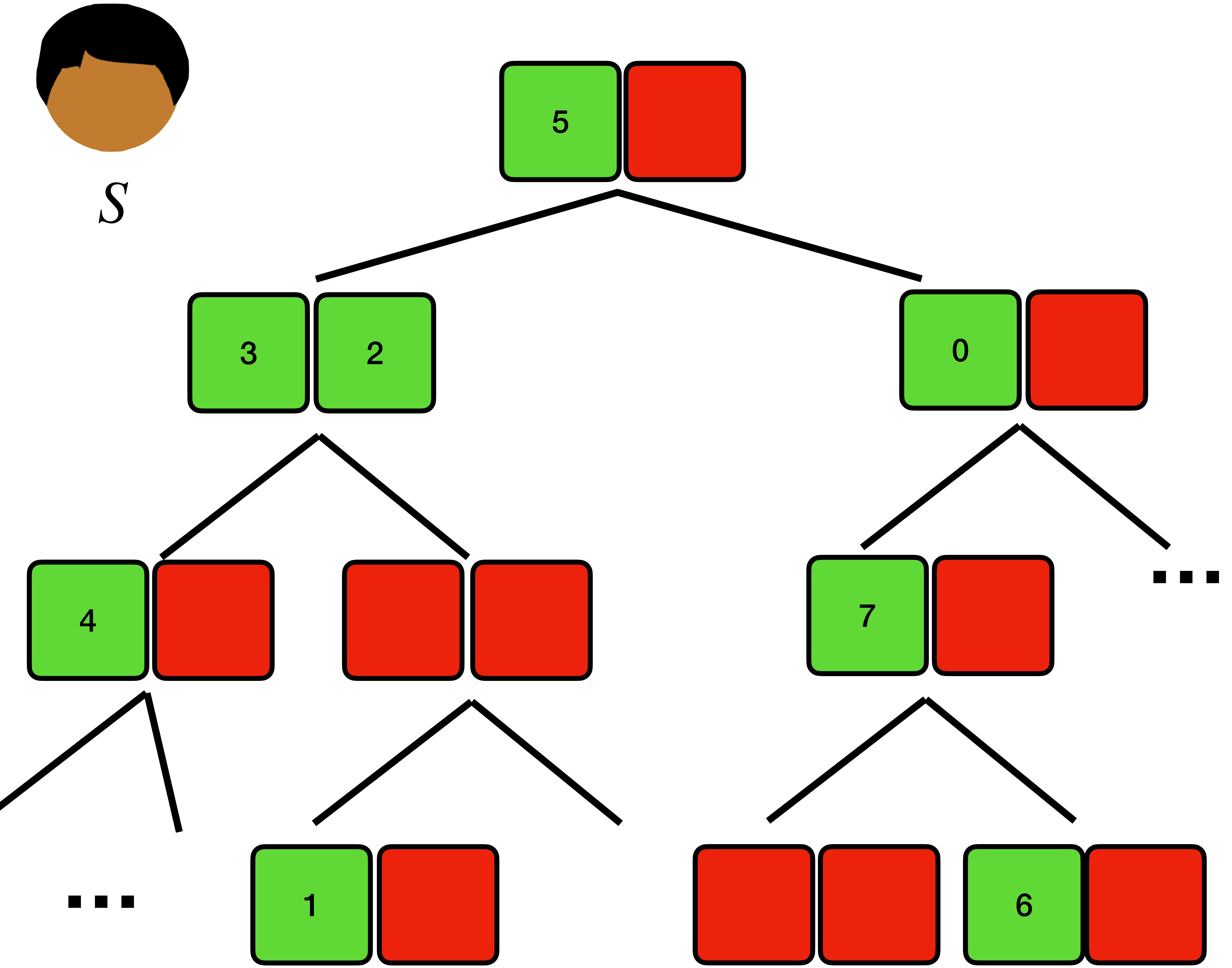
Access(op, a, data*):

```
x ← position[a]
position[a] ← x* ← 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, x, data)}) ∪ {(a, x*, data*)}
end if
for ℓ ∈ {L, L − 1, ..., 0} do
    S' ← {(a', x', data') ∈ S : P(x, ℓ) = P(x', ℓ)}
    S' ← Select min(|S'|, Z) blocks from S'.
    S ← S − S'
    WriteBucket(P(x, ℓ), S')
end for
return data
```

Fig. 1. **Protocol for data access.** Read or write a data block identified by a. If op = read, the input parameter data* = None, and the Access operation reads block a from the ORAM. If op = write, the Access operation writes the specified data* to the block identified by a and returns the block's old data.

S
C
5
3
2
0
4
7
1
6
...
...

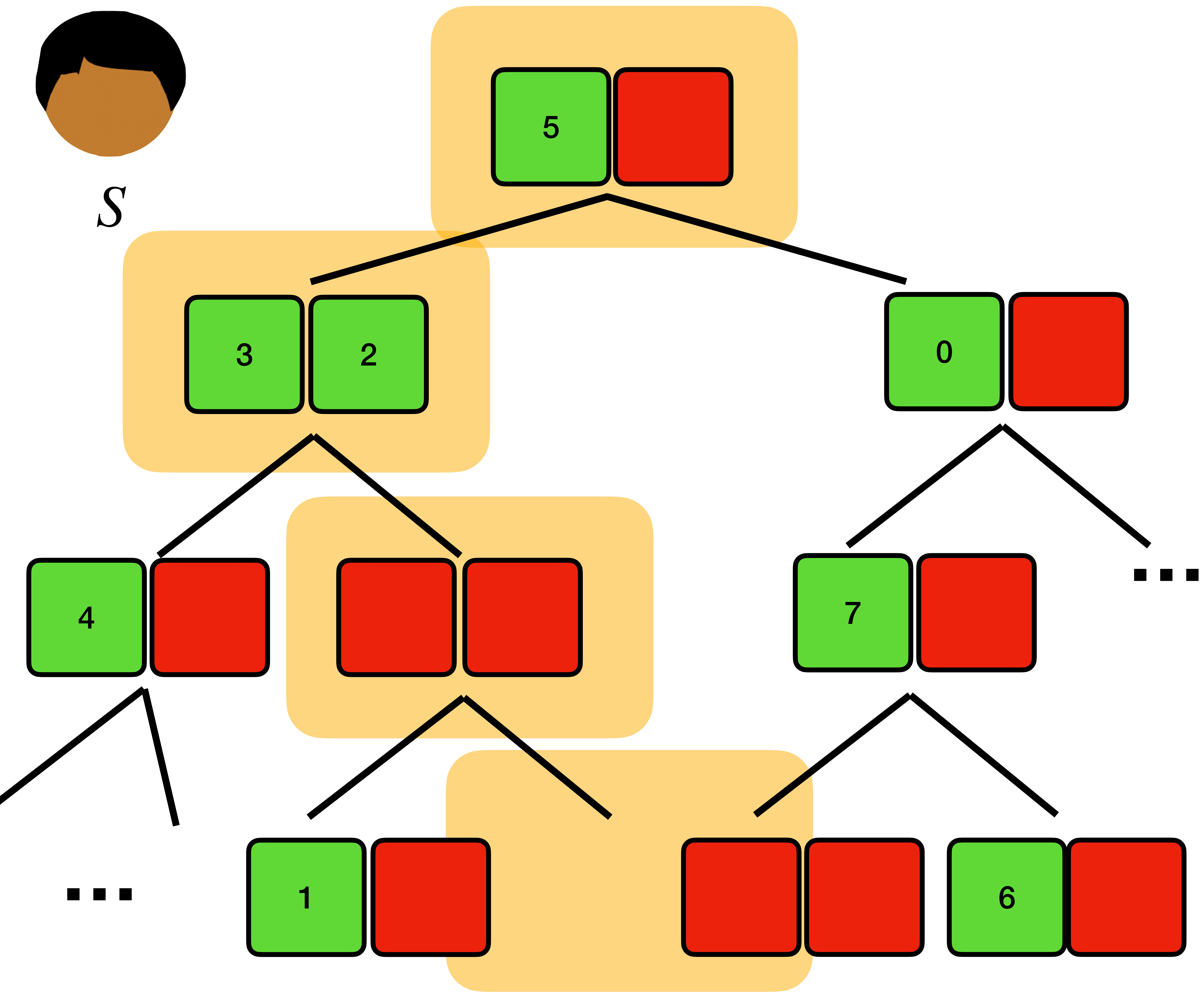
Path Invariant: Each node is assigned a uniformly random leaf

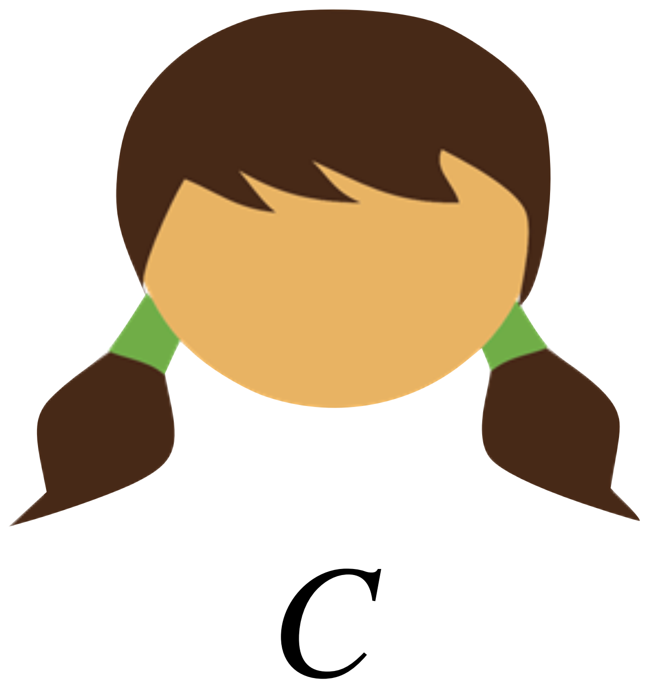


Path Invariant: Each node is assigned a uniformly random leaf

Logical address	Leaf
0	10
1	5
2	7
...	...

Position Map

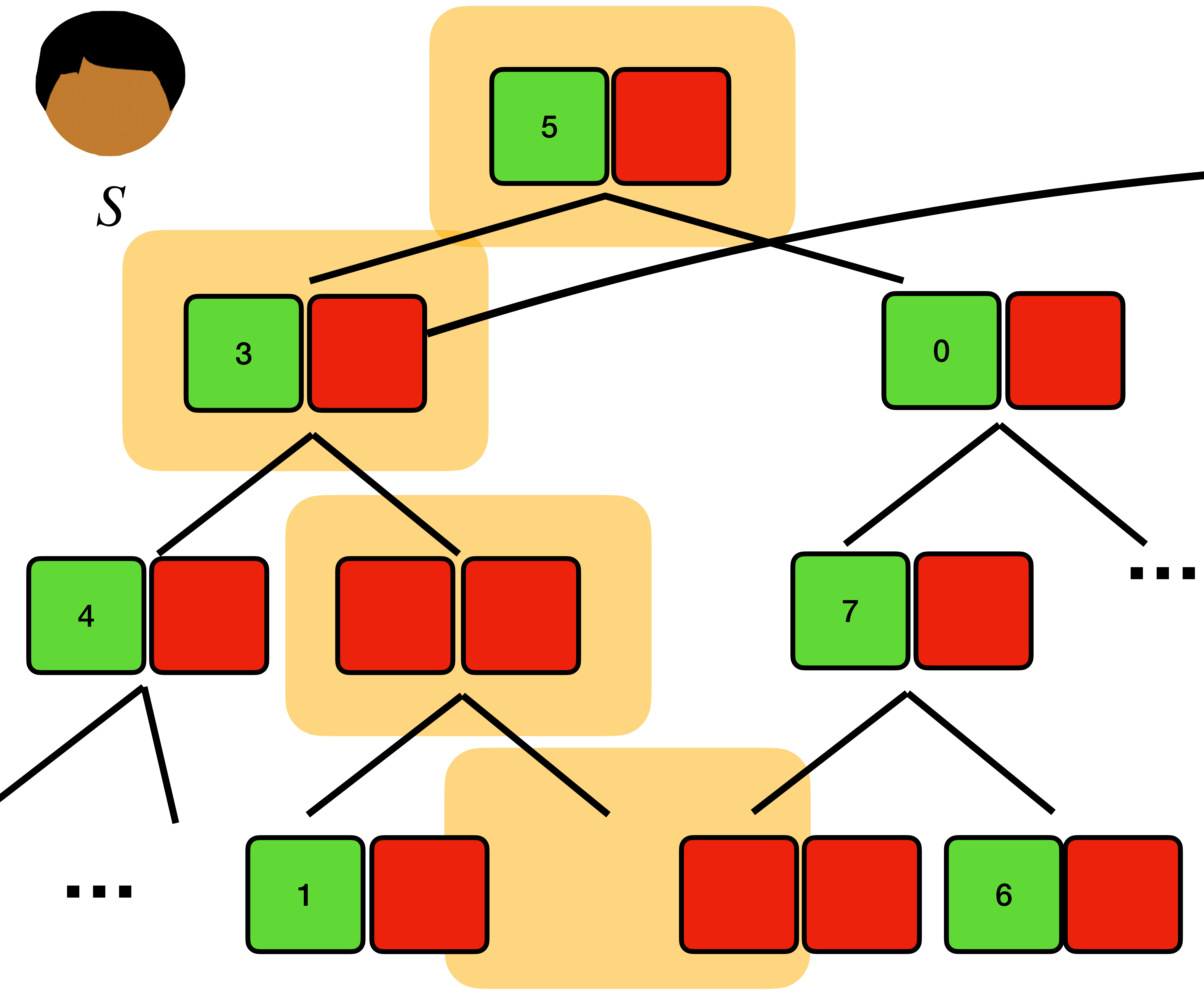


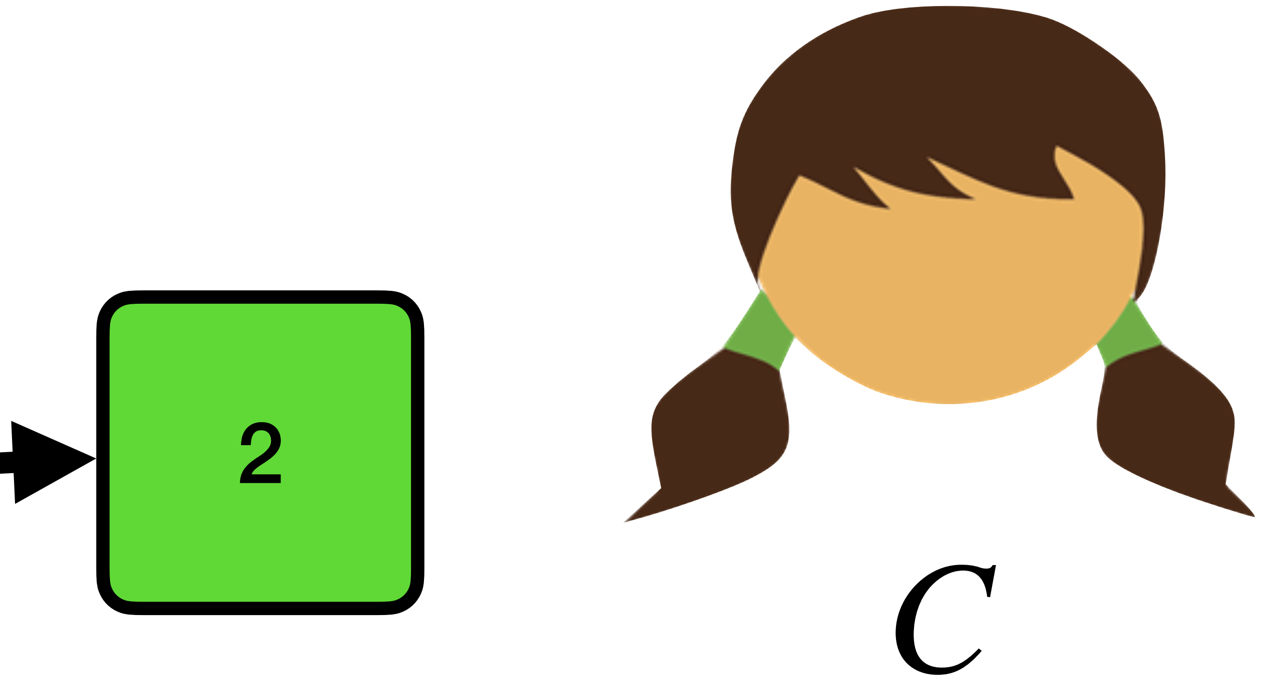


To find an element, client searches the path to the leaf

Logical address	Leaf
0	10
1	5
2	7
...	...

Position Map

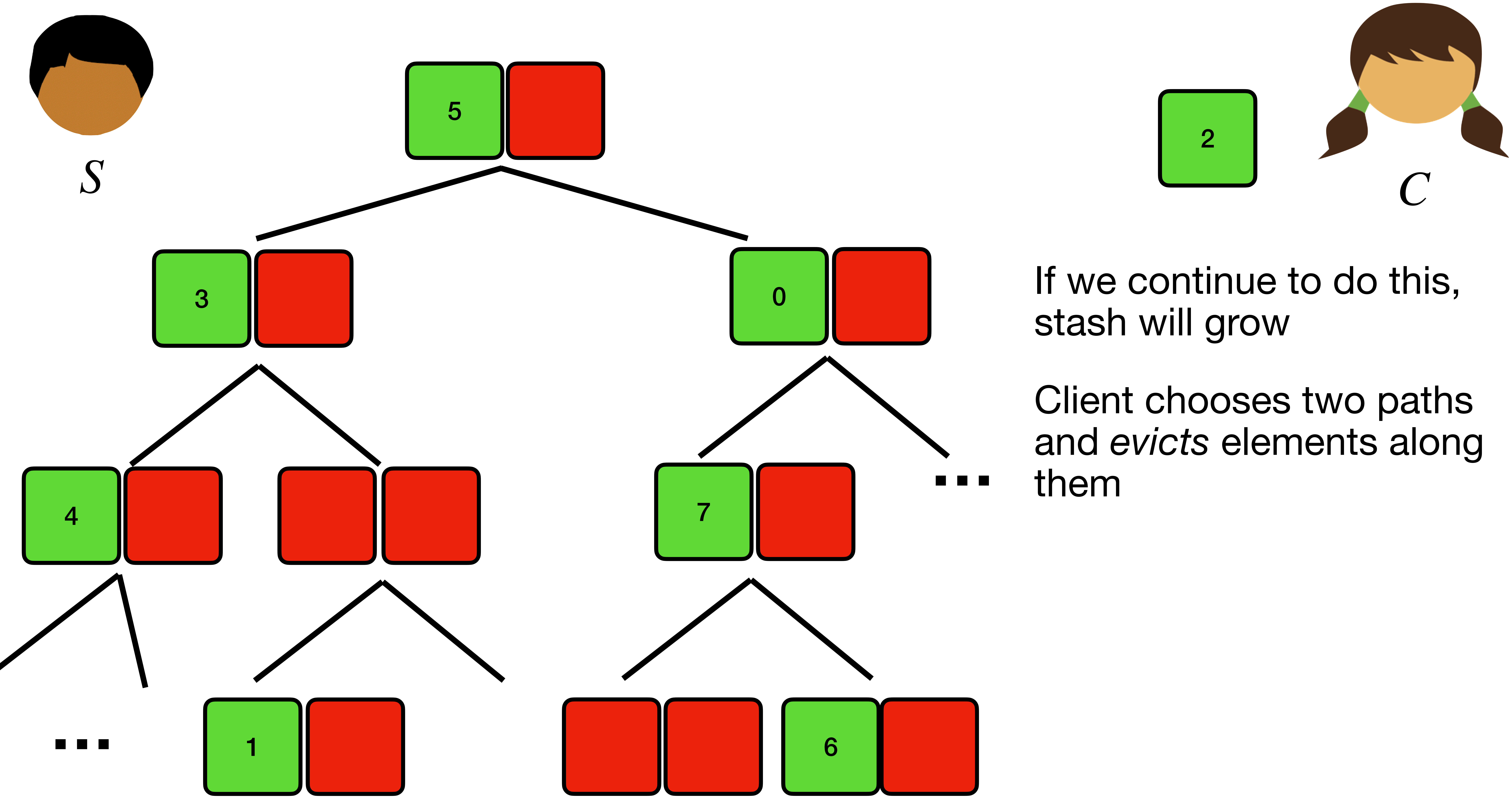




Fetched element is stored in a small **stash** on the client

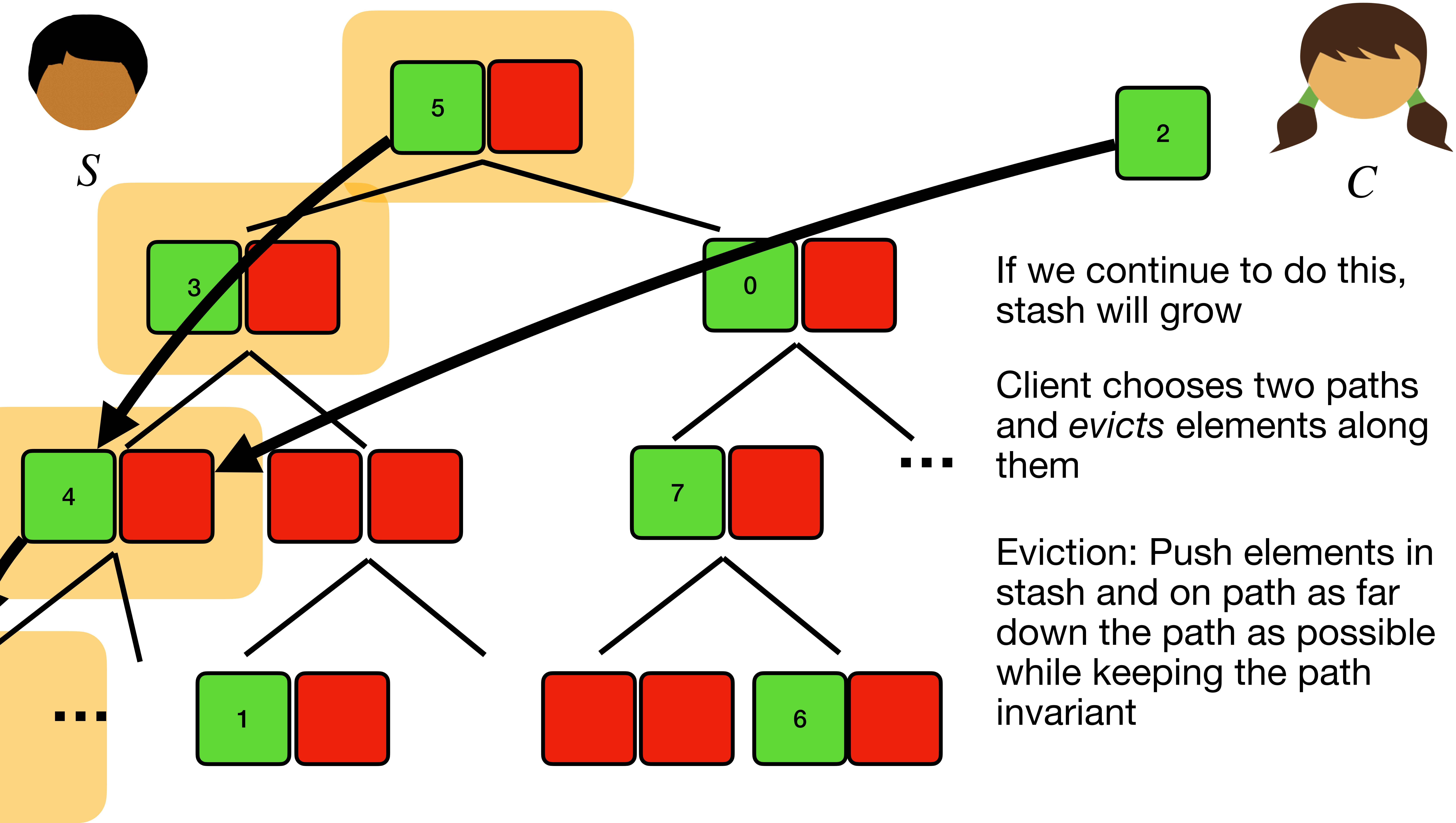
Logical address	**Leaf**
0	10
1	5
2	7
...	...

Position Map



If we continue to do this, stash will grow

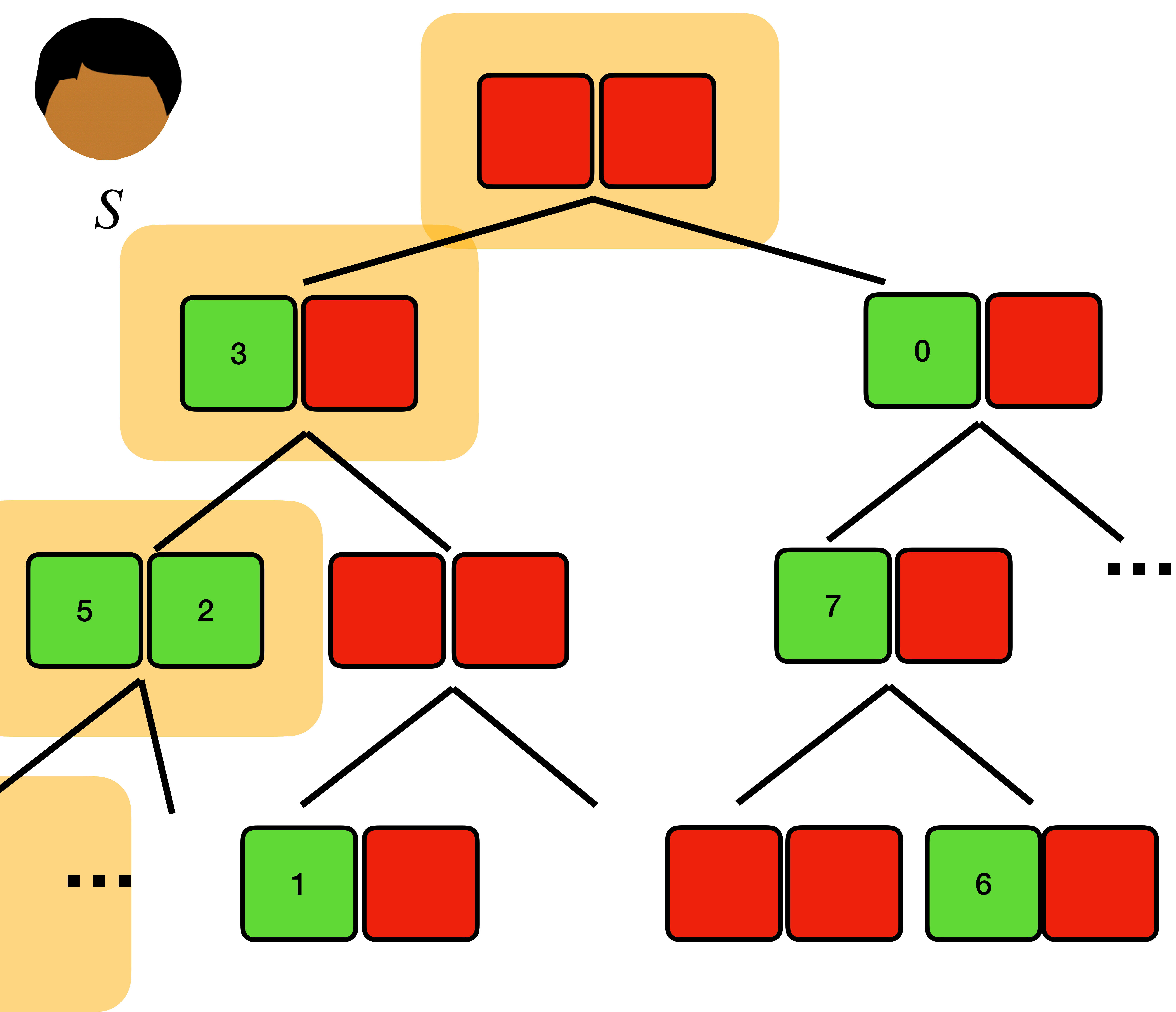
Client chooses two paths and *evicts* elements along them



If we continue to do this, stash will grow

Client chooses two paths and *evicts* elements along them

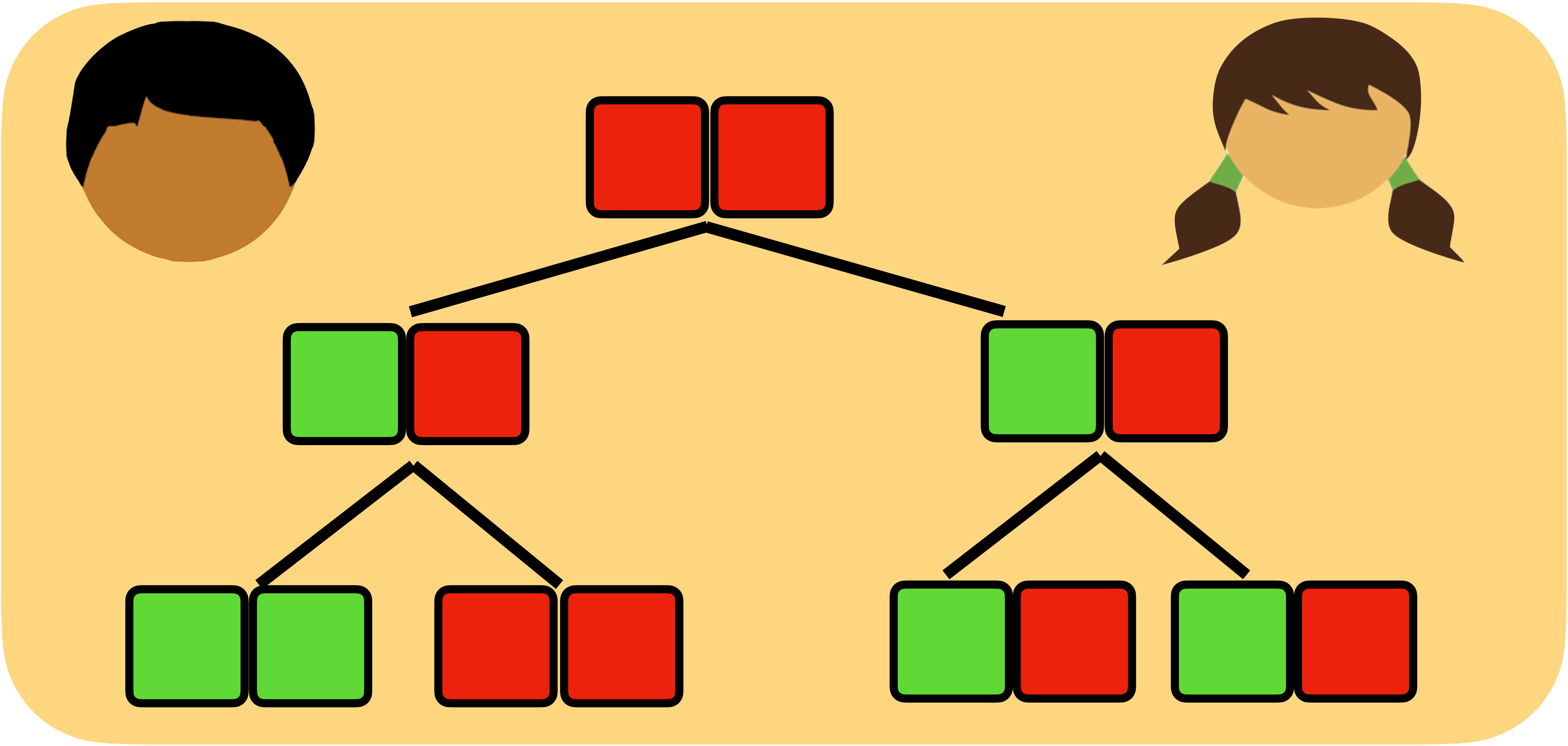
Eviction: Push elements in stash and on path as far down the path as possible while keeping the path invariant



If we continue to do this, stash will grow

Client chooses two paths and *evicts* elements along them

Eviction: Push elements in stash and on path as far down the path as possible while keeping the path invariant



Path ORAM

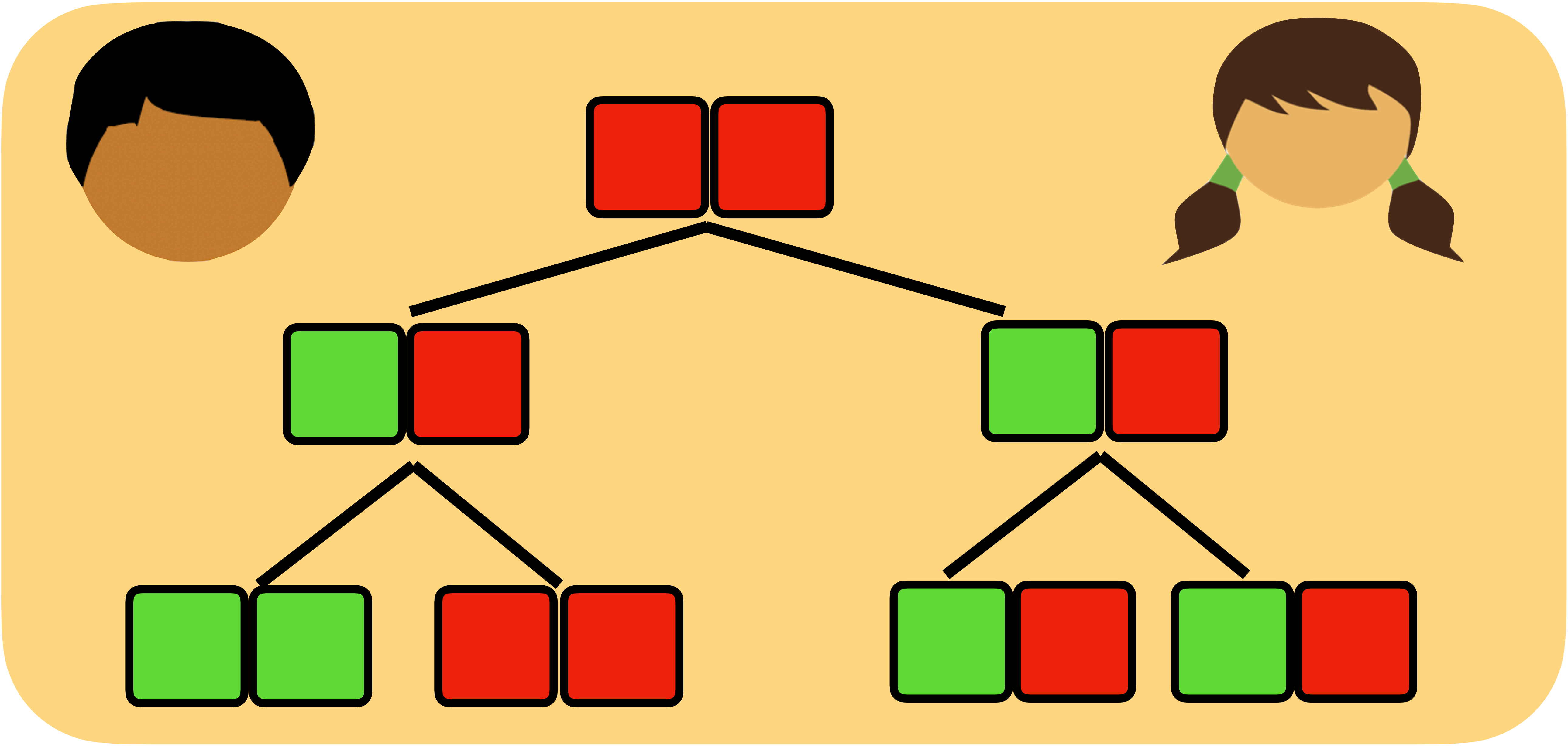
Each data item is assigned a uniformly random leaf

To perform an access, client queries the path to the appropriate leaf

Because leaves are chosen uniformly, we can simulate what the server sees

After access, the client writes back to the stash and assigns a fresh leaf

To avoid the stash growing too large, client reads paths and **evicts** them



Path ORAM

Each data item is assigned a uniformly random leaf

To perform an access, client queries the path to the appropriate leaf

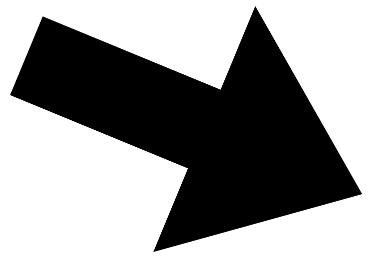
Because leaves are chosen uniformly, we can simulate what the server sees

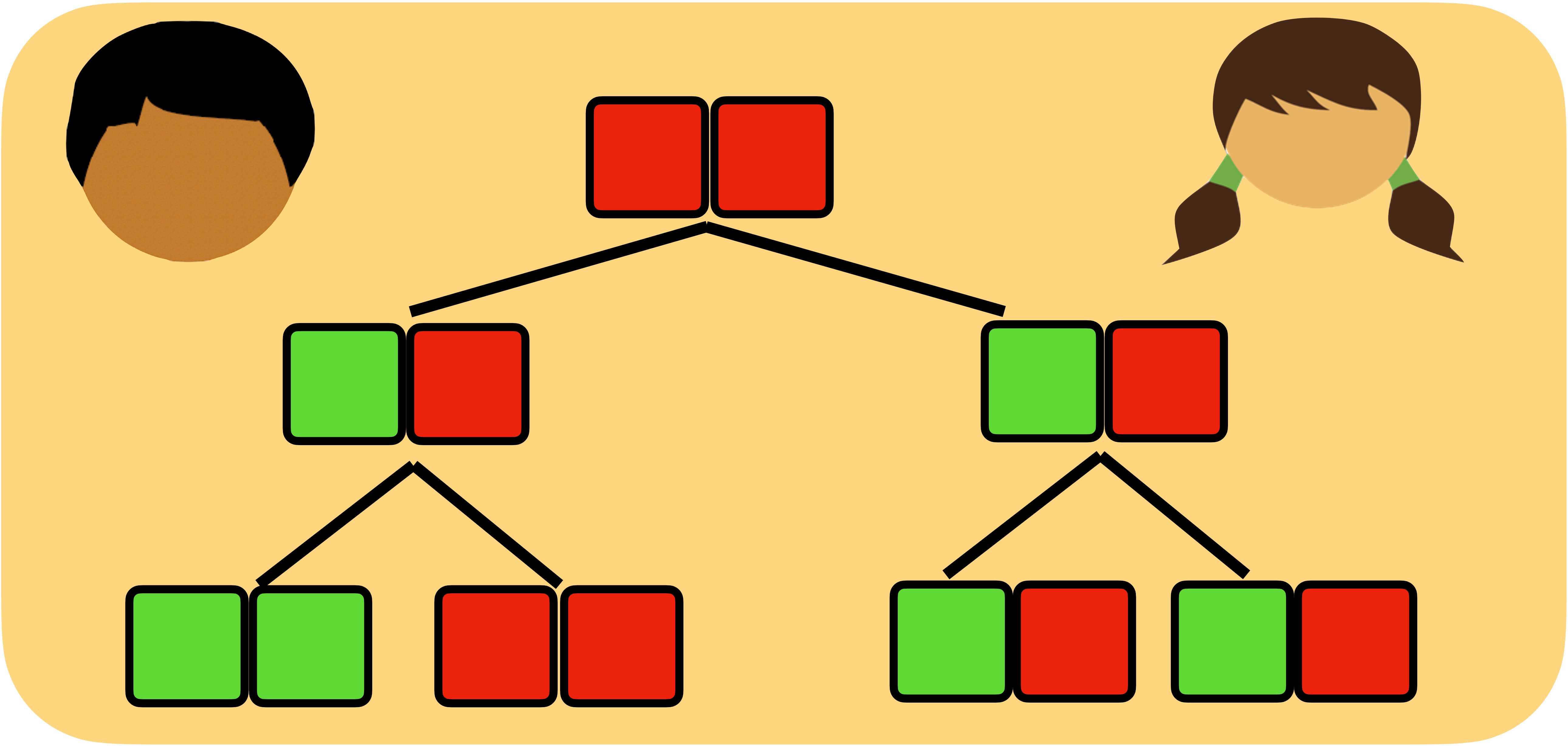
After access, the client writes back to the stash and assigns a fresh leaf

To avoid the stash growing too large, client reads paths and **evicts** them

If an element does not fit on the path, we keep it in stash

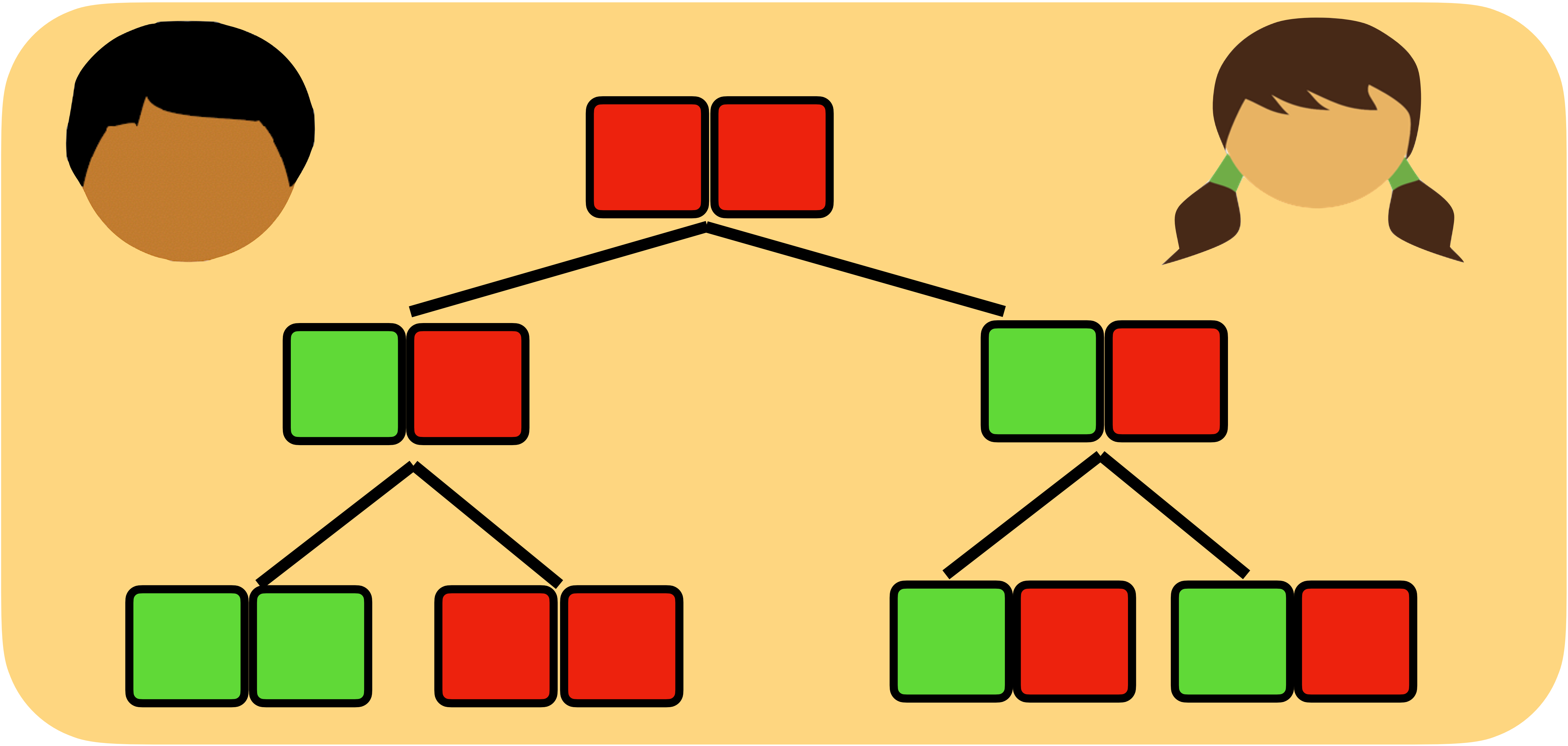
Careful analysis shows that w.h.p. the stash will not grow “too big”





Path ORAM

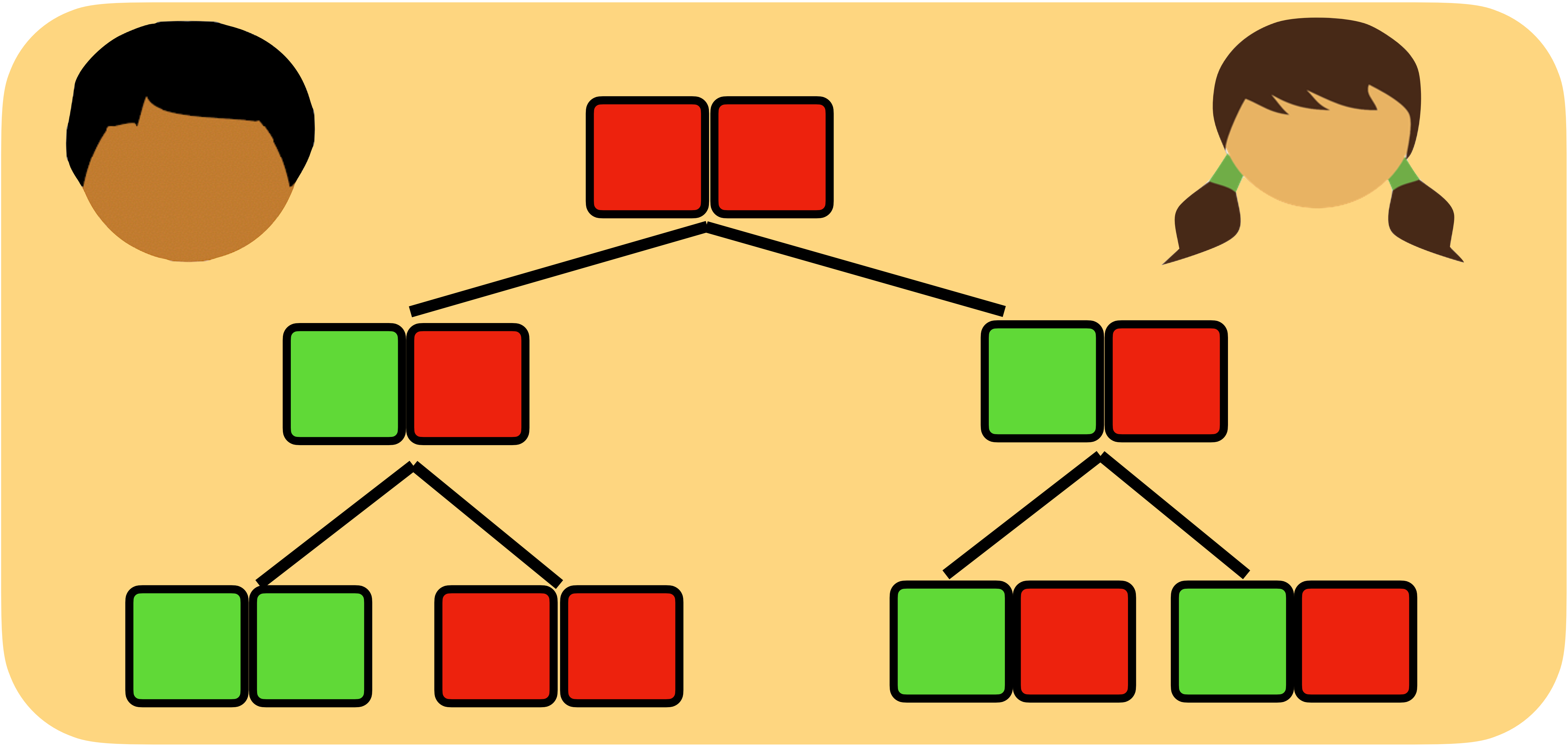
Question: The position map has O(n) size. How does the client store it?



Path ORAM

Question: The position map has O(n) size. How does the client store it?

Answer: With another (recursively instantiated) ORAM. There will be O(log n) total levels of ORAM.



Path ORAM

Question: The position map has O(n) size. How does the client store it?

Answer: With another (recursively instantiated) ORAM. There will be O(log n) total levels of ORAM.

An ORAM requires reading a path of length O(log n), and there are O(log n) ORAMs

$O(\log^2 n)$ total blow-up

Path ORAM: An Extremely Simple Oblivious RAM Protocol

18

EMIL STEFANOV, UC Berkeley
MARTEN VAN DIJK, University of Connecticut
ELAINE SHI, Cornell University
T.-H. HUBERT CHAN, University of Hong Kong
CHRISTOPHER FLETCHER, University of Illinois at Urbana-Champaign
LING REN, XIANGYAO YU, and SRINIVAS DEVADAS, MIT CSAIL

We present Path ORAM, an extremely simple Oblivious RAM protocol with a small amount of client storage. Partly due to its simplicity, Path ORAM is the most practical ORAM scheme known to date with small client storage. We formally prove that Path ORAM has a $O(\log N)$ bandwidth cost for blocks of size $B = \Omega(\log^2 N)$ bits. For such block sizes, Path ORAM is asymptotically better than the best-known ORAM schemes with small client storage. Due to its practicality, Path ORAM has been adopted in the design of secure processors since its proposal.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Algorithms, Security

Additional Key Words and Phrases: Oblivious RAM, ORAM, Path ORAM, access pattern

ACM Reference format:
Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: An Extremely Simple Oblivious RAM Protocol. *J. ACM* 65, 4, Article 18 (April 2018), 26 pages.
https://doi.org/10.1145/3177872

A conference version of the article has appeared in ACM Conference on Computer and Communications Security (CCS), 2013.
This work is partially supported by the NSF Graduate Research Fellowship grants DGE-0946797 and DGE-1122374, the DoD NDSEG Fellowship, NSF grant CNS-1314857, DARPA CRASH program N66001-10-2-4089, and a grant from the Amazon Web Services in Education program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.
The research was supported in part by a grant from Hong Kong RGC under the contract HKU719312E.
Authors' addresses: E. Stefanov, Department of Electrical Engineering and Computer Sciences, UC Berkeley, CA 94720, USA; email: emil@berkeley.edu; M. V. Dijk, Electrical and Computing Engineering Department, University of Connecticut, Storrs-Mansfield, CT 06269, USA; email: vandijk@engr.uconn.edu; E. Shi, Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, USA; email: elaine@cs.cornell.edu; T.-H. H. Chan, Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong; email: hubert@cs.hku.hk; C. Fletcher, Computer Science Department, University of Illinois–Urbana Champaign, Urbana, IL 61801, USA; email: cwfletch@illinois.edu; L. Ren, X. Yu, and S. Devadas, MIT CSAIL, Cambridge, MA 02139, USA; emails: {renling, yxy, devadas}@csail.mit.edu.

https://doi.org/10.1145/3177872

$O(\log^2 n)$ physical accesses

Access(op, a, data*):

$x \leftarrow$ position[a]
position[a] $\leftarrow x^* \leftarrow$ UniformRandom$(0 \ldots 2^L - 1)$
for $\ell \in \{0, 1, \ldots, L\}$ **do**
  $S \leftarrow S \cup$ ReadBucket$(\mathcal{P}(x, \ell))$
end for
data $\leftarrow$ Read block a from S
if op = write **then**
  $S \leftarrow (S - \{(\mathsf{a}, x, \mathsf{data})\}) \cup \{(\mathsf{a}, x^*, \mathsf{data}^*)\}$
end if
for $\ell \in \{L, L-1, \ldots, 0\}$ **do**
  $S' \leftarrow \{(\mathsf{a}', x', \mathsf{data}') \in S : \mathcal{P}(x, \ell) = \mathcal{P}(x', \ell)\}$
  $S' \leftarrow$ Select $\min(|S'|, Z)$ blocks from S'.
  $S \leftarrow S - S'$
  WriteBucket$(\mathcal{P}(x, \ell), S')$
end for
return data

Fig. 1. **Protocol for data access.** Read or write a data block identified by a. If op = read, the input parameter data* = None, and the Access operation reads block a from the ORAM. If op = write, the Access operation writes the specified data* to the block identified by a and returns the block's old data.

ORAM Lower Bound

Natural question: How low can we go in terms of overhead?

Yes, There is an Oblivious RAM Lower Bound!

Kasper Green Larsen* and Jesper Buus Nielsen**

[1] Computer Science. Aarhus University
[2] Computer Science & DIGIT, Aarhus University

Abstract. An Oblivious RAM (ORAM) introduced by Goldreich and Ostrovsky [JACM'96] is a (possibly randomized) RAM, for which the memory access pattern reveals no information about the operations performed. The main performance metric of an ORAM is the bandwidth overhead, i.e., the multiplicative factor extra memory blocks that must be accessed to hide the operation sequence. In their seminal paper introducing the ORAM, Goldreich and Ostrovsky proved an amortized $\Omega(\lg n)$ bandwidth overhead lower bound for ORAMs with memory size n. Their lower bound is very strong in the sense that it applies to the "offline" setting in which the ORAM knows the entire sequence of operations ahead of time.

However, as pointed out by Boyle and Naor [ITCS'16] in the paper "Is there an oblivious RAM lower bound?", there are two caveats with the lower bound of Goldreich and Ostrovsky: (1) it only applies to "balls in bins" algorithms, i.e., algorithms where the ORAM may only shuffle blocks around and not apply any sophisticated encoding of the data, and (2), it only applies to statistically secure constructions. Boyle and Naor showed that removing the "balls in bins" assumption would result in super linear lower bounds for sorting circuits, a long standing open problem in circuit complexity. As a way to circumventing this barrier, they also proposed a notion of an "online" ORAM, which is an ORAM that remains secure even if the operations arrive in an online manner. They argued that most known ORAM constructions work in the online setting as well.

Our contribution is an $\Omega(\lg n)$ lower bound on the bandwidth overhead of any online ORAM, even if we require only computational security and allow arbitrary representations of data, thus greatly strengthening the lower bound of Goldreich and Ostrovsky in the online setting. Our lower bound applies to ORAMs with memory size n and any word size $r \geq 1$. The bound therefore asymptotically matches the known upper bounds

Fact (informal): Any secure ORAM must incur overhead at least $\Omega(\log n)$

ORAM Lower Bound

Natural question: How low can we go in terms of overhead?

Yes, There is an Oblivious RAM Lower Bound!

Kasper Green Larsen* and Jesper Buus Nielsen**

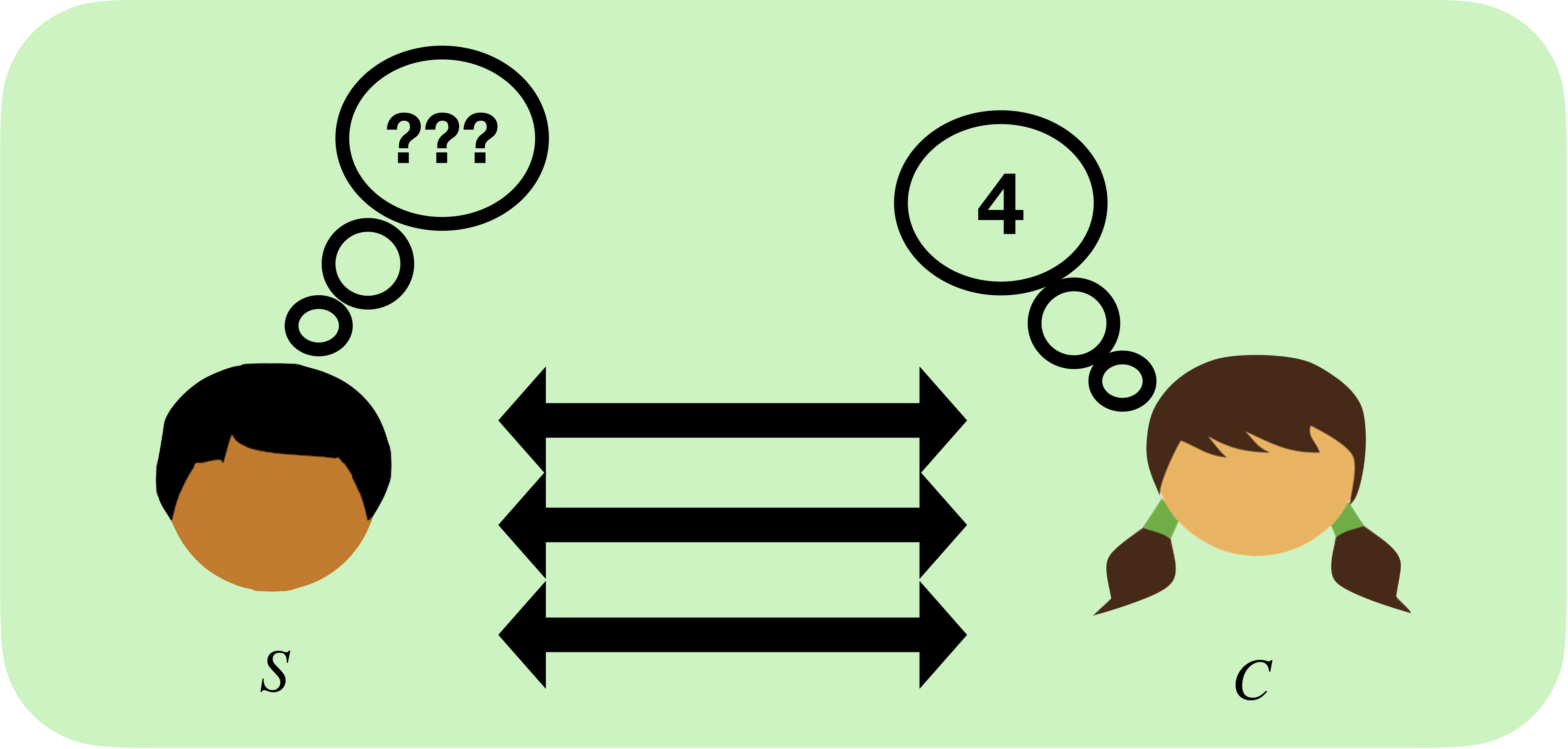
[1] Computer Science. Aarhus University
[2] Computer Science & DIGIT, Aarhus University

Abstract. An Oblivious RAM (ORAM) introduced by Goldreich and Ostrovsky [JACM'96] is a (possibly randomized) RAM, for which the memory access pattern reveals no information about the operations performed. The main performance metric of an ORAM is the bandwidth overhead, i.e., the multiplicative factor extra memory blocks that must be accessed to hide the operation sequence. In their seminal paper introducing the ORAM, Goldreich and Ostrovsky proved an amortized $\Omega(\lg n)$ bandwidth overhead lower bound for ORAMs with memory size n. Their lower bound is very strong in the sense that it applies to the "offline" setting in which the ORAM knows the entire sequence of operations ahead of time.
However, as pointed out by Boyle and Naor [ITCS'16] in the paper "Is there an oblivious RAM lower bound?", there are two caveats with the lower bound of Goldreich and Ostrovsky: (1) it only applies to "balls in bins" algorithms, i.e., algorithms where the ORAM may only shuffle blocks around and not apply any sophisticated encoding of the data, and (2), it only applies to statistically secure constructions. Boyle and Naor showed that removing the "balls in bins" assumption would result in super linear lower bounds for sorting circuits, a long standing open problem in circuit complexity. As a way to circumventing this barrier, they also proposed a notion of an "online" ORAM, which is an ORAM that remains secure even if the operations arrive in an online manner. They argued that most known ORAM constructions work in the online setting as well.
Our contribution is an $\Omega(\lg n)$ lower bound on the bandwidth overhead of any online ORAM, even if we require only computational security and allow arbitrary representations of data, thus greatly strengthening the lower bound of Goldreich and Ostrovsky in the online setting. Our lower bound applies to ORAMs with memory size n and any word size $r \geq 1$. The bound therefore asymptotically matches the known upper bounds

Fact (informal): Any secure ORAM must incur overhead at least $\Omega(\log n)$

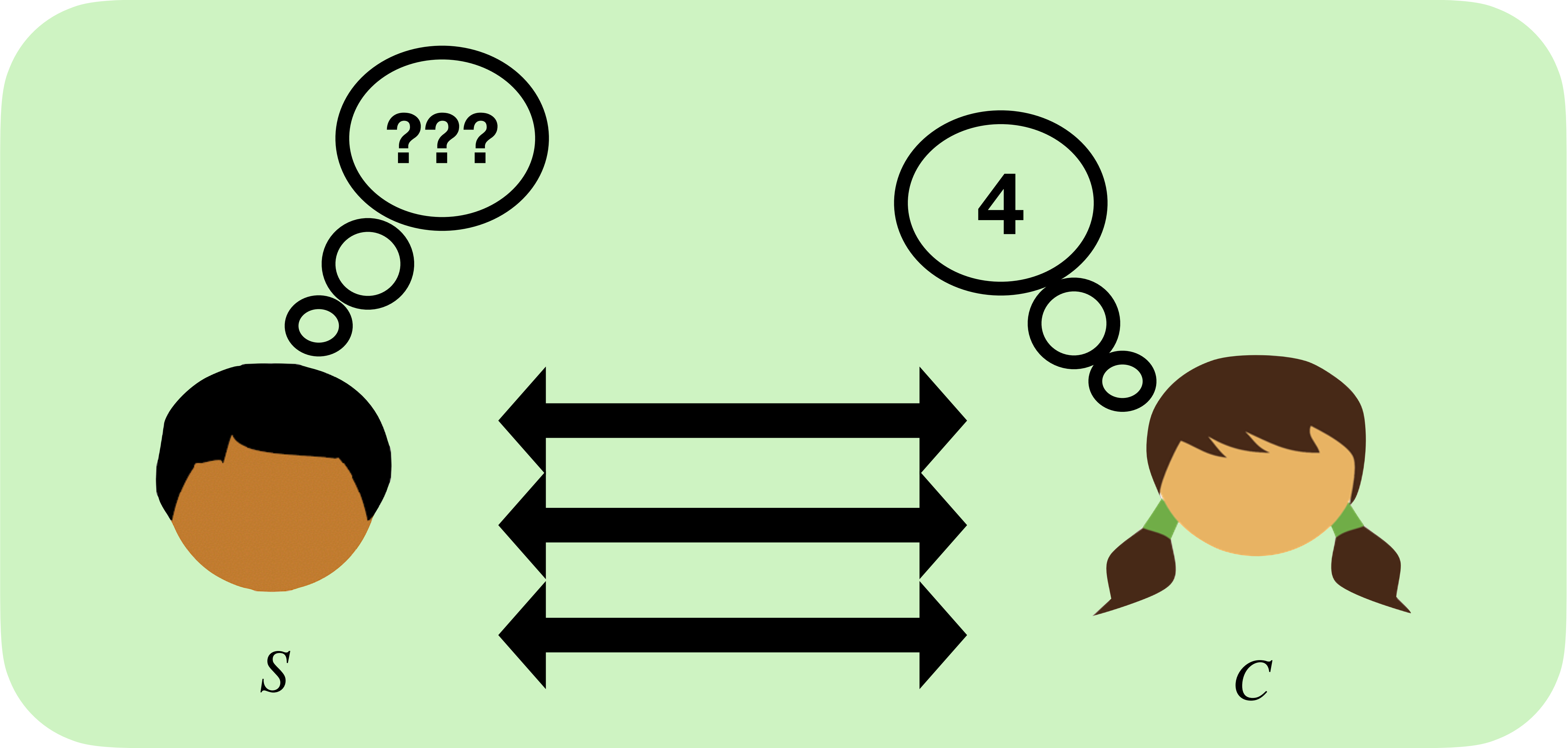
Combines two concepts:

- All access patterns should look the same to the server
- Certain access patterns will **force** the client to save its data on the server, then retrieve it later



We are trying to prove that **any** ORAM protocol must have log overhead

Important to formalize what an ORAM protocol **is**

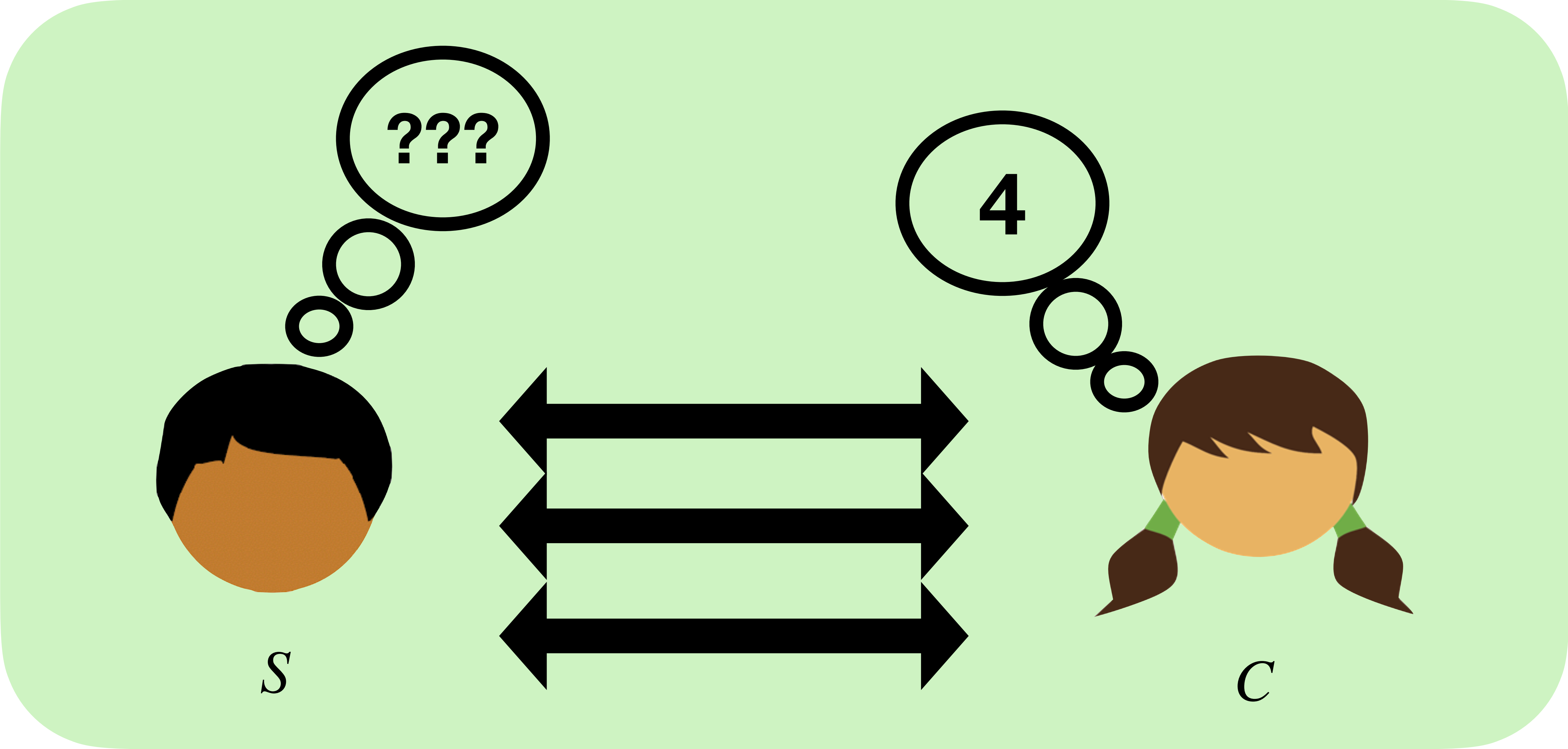


We are trying to prove that **any** ORAM protocol must have log overhead

Important to formalize what an ORAM protocol **is**

Model

Client learns its queries one at a time, and must satisfy any reads as soon as they come in (online)



We are trying to prove that **any** ORAM protocol must have log overhead

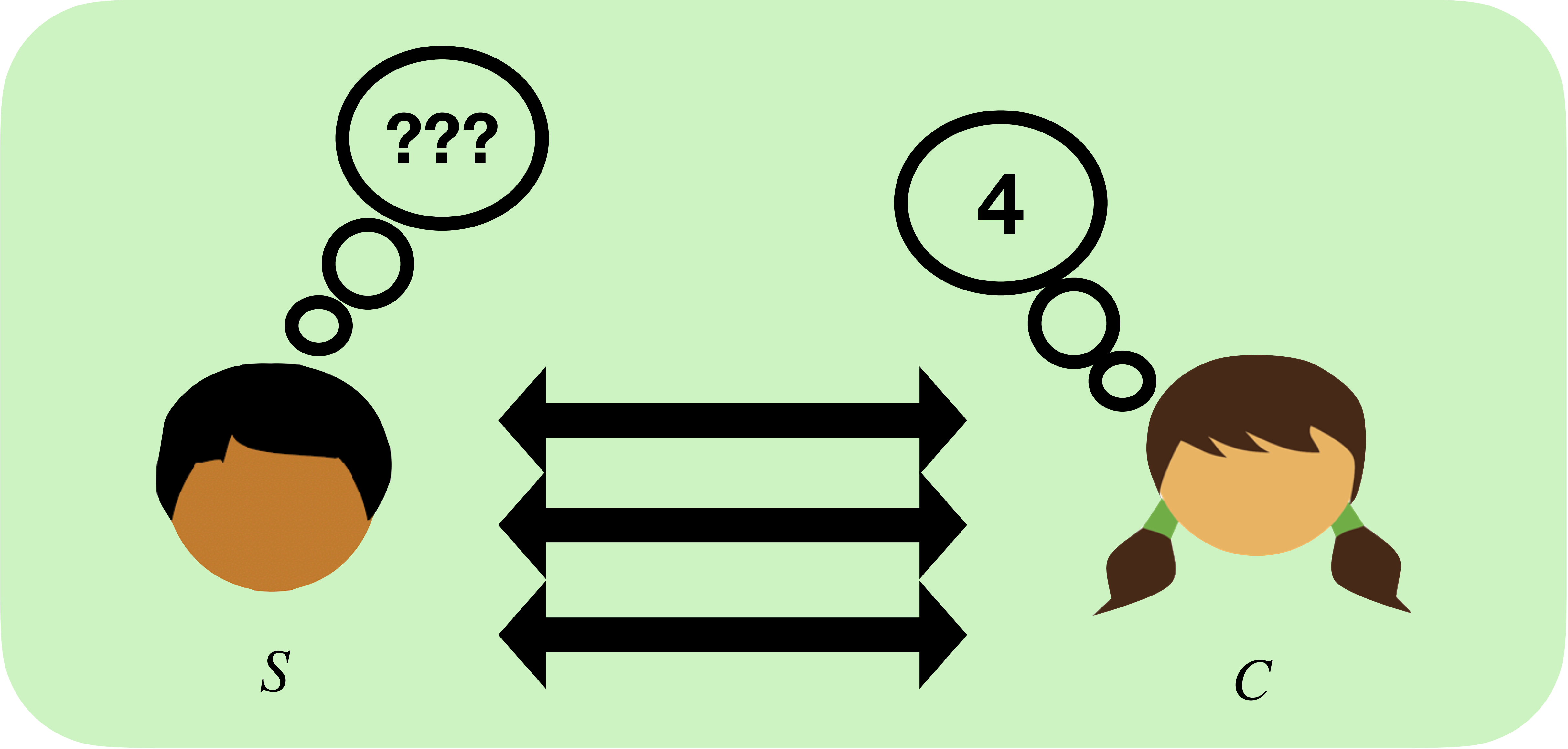
Important to formalize what an ORAM protocol **is**

Model

Client learns its queries one at a time, and must satisfy any reads as soon as they come in (online)

ORAM protocol is a sequence of **probes:**

1. C queries location i
2. S sends content of location i
3. C sends back new value
4. S saves the new value in location i



We are trying to prove that **any** ORAM protocol must have log overhead

Important to formalize what an ORAM protocol **is**

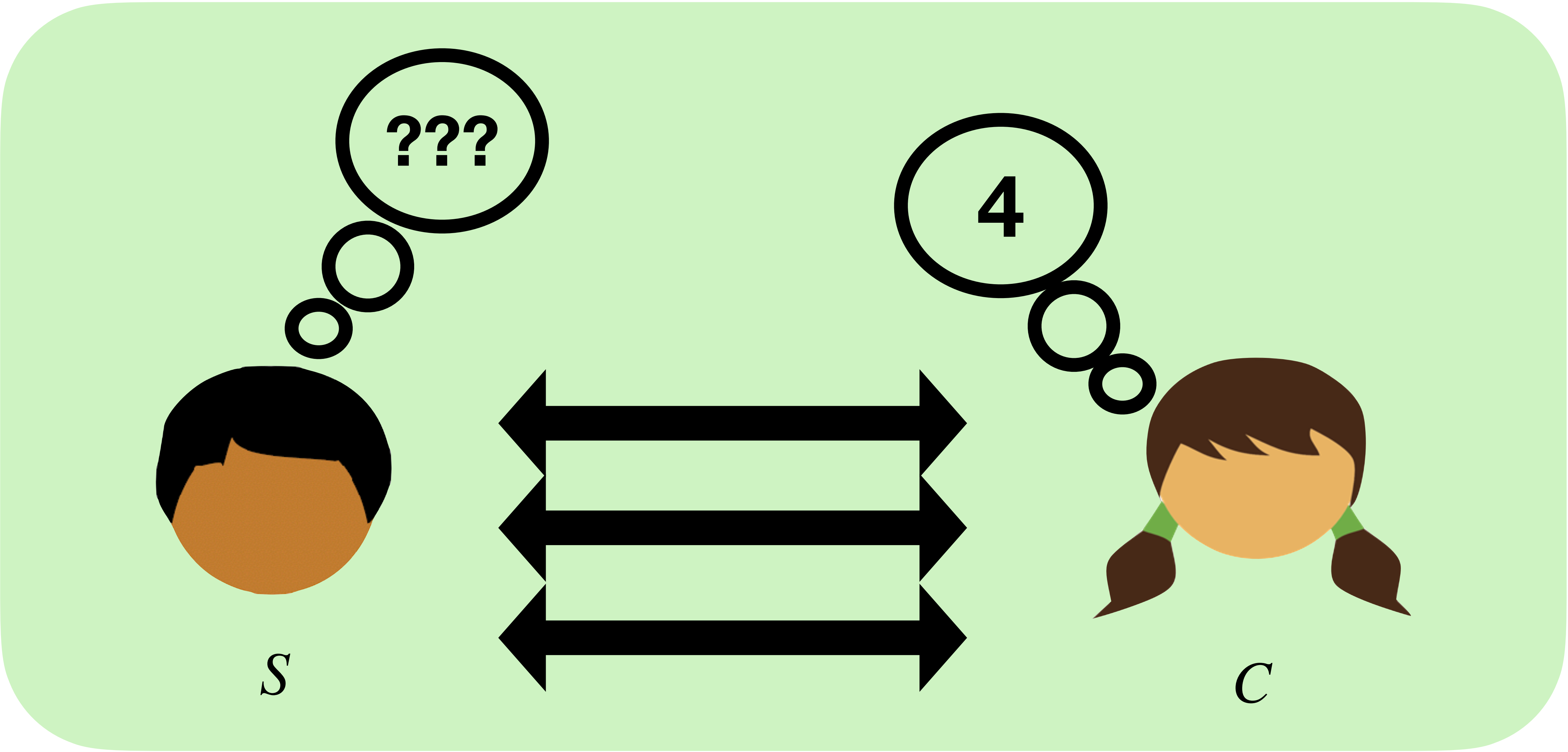
Model

Client learns its queries one at a time, and must satisfy any reads as soon as they come in (online)

ORAM protocol is a sequence of **probes:**

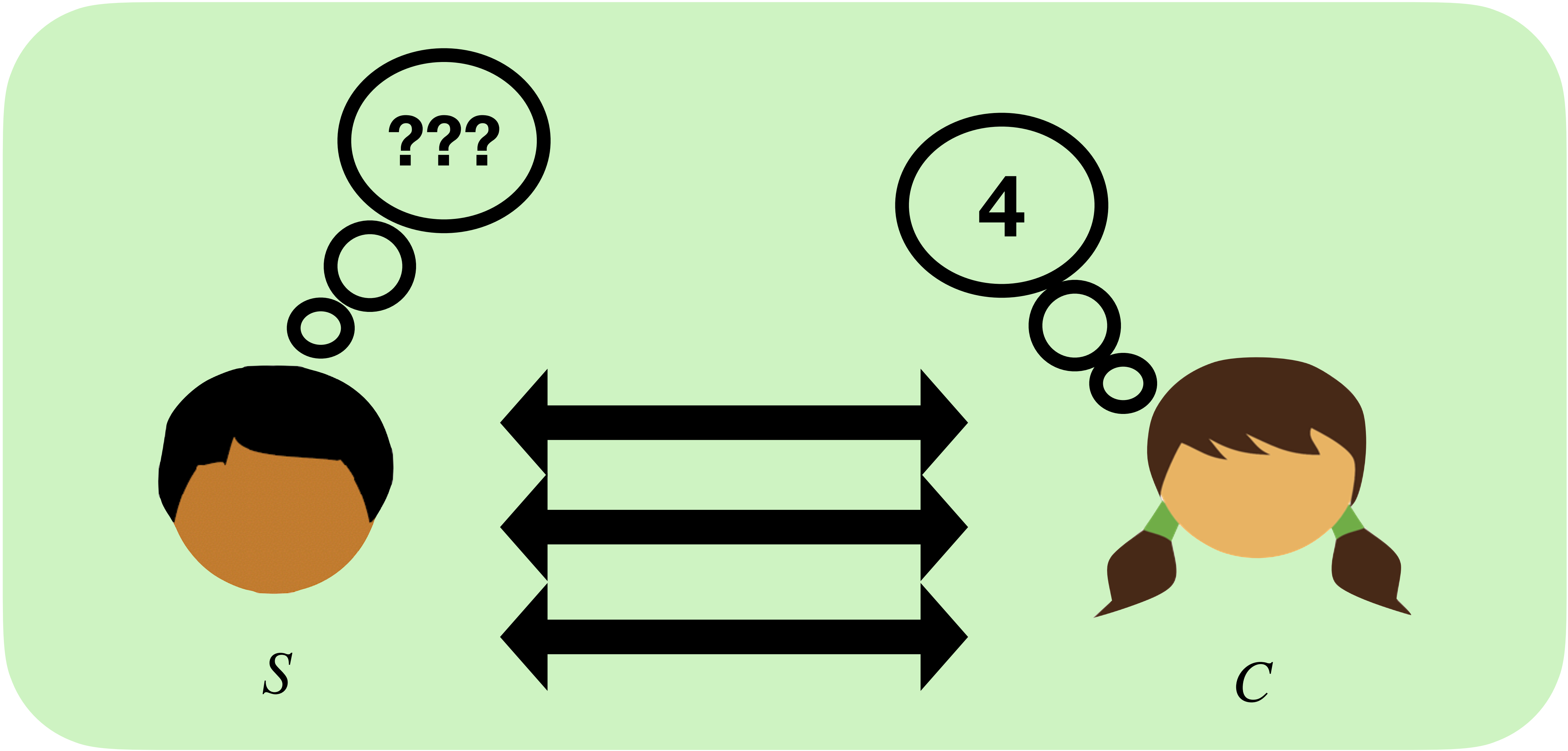
1. C queries location i
2. S sends content of location i
3. C sends back new value
4. S saves the new value in location i

Client can hold only $O(1)$ data items



Logical Access Pattern

Write 0	Write 3	Read 0	Read 0	Write 1	Read 1	Read 3	Write 3



n

Logical Access Pattern

Write 0	Write 3	Read 0	Read 0	Write 1	Read 1	Read 3	Write 3

Logical Access Pattern

Write 0

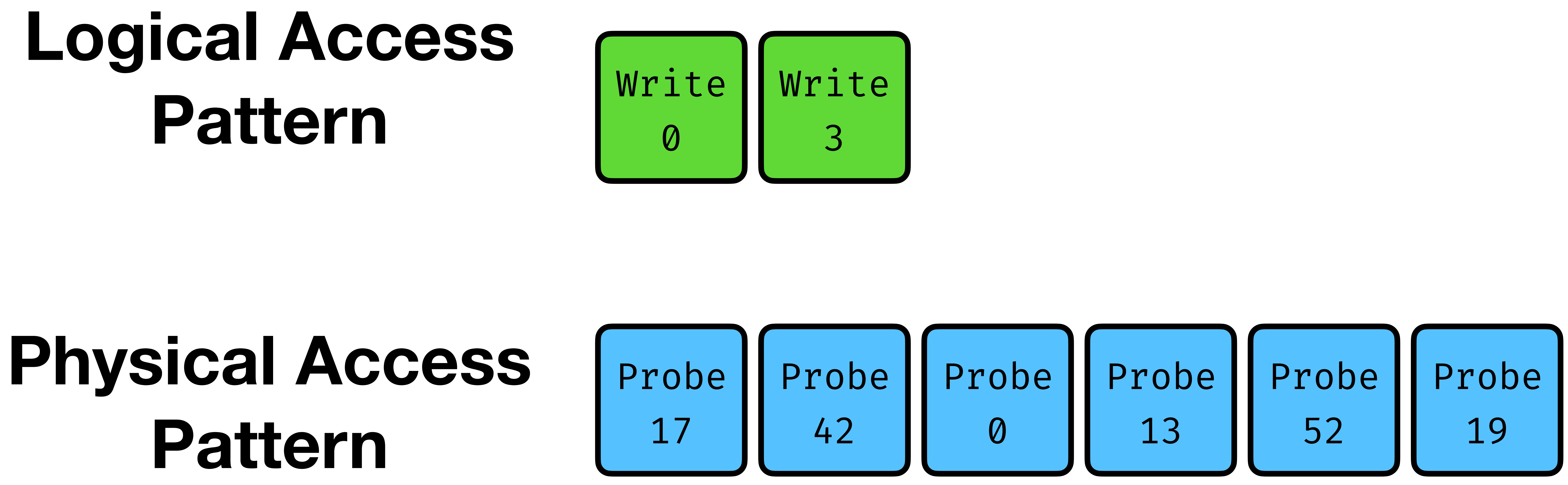
Physical Access Pattern

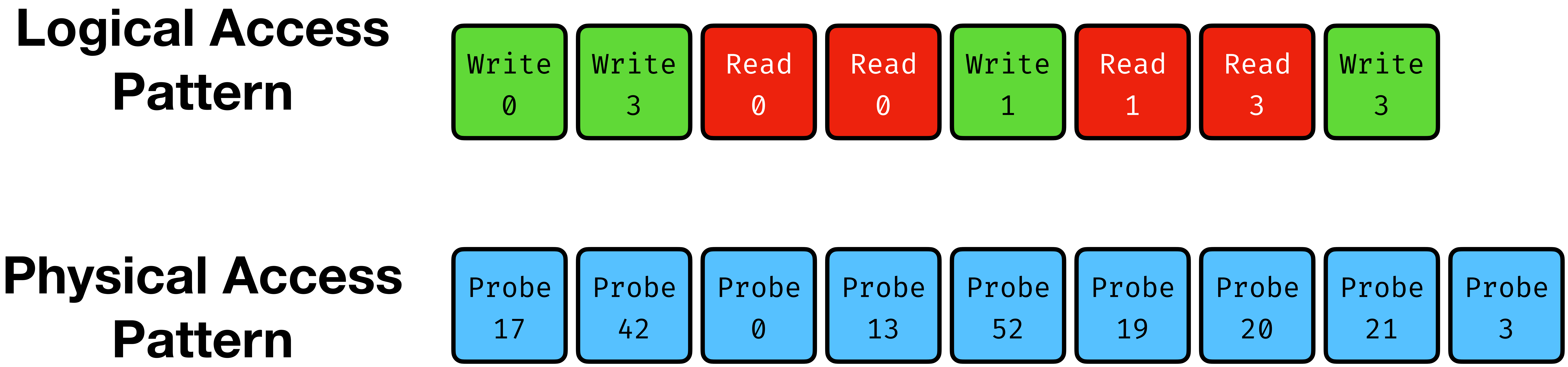
Logical Access Pattern

Write 0

Physical Access Pattern

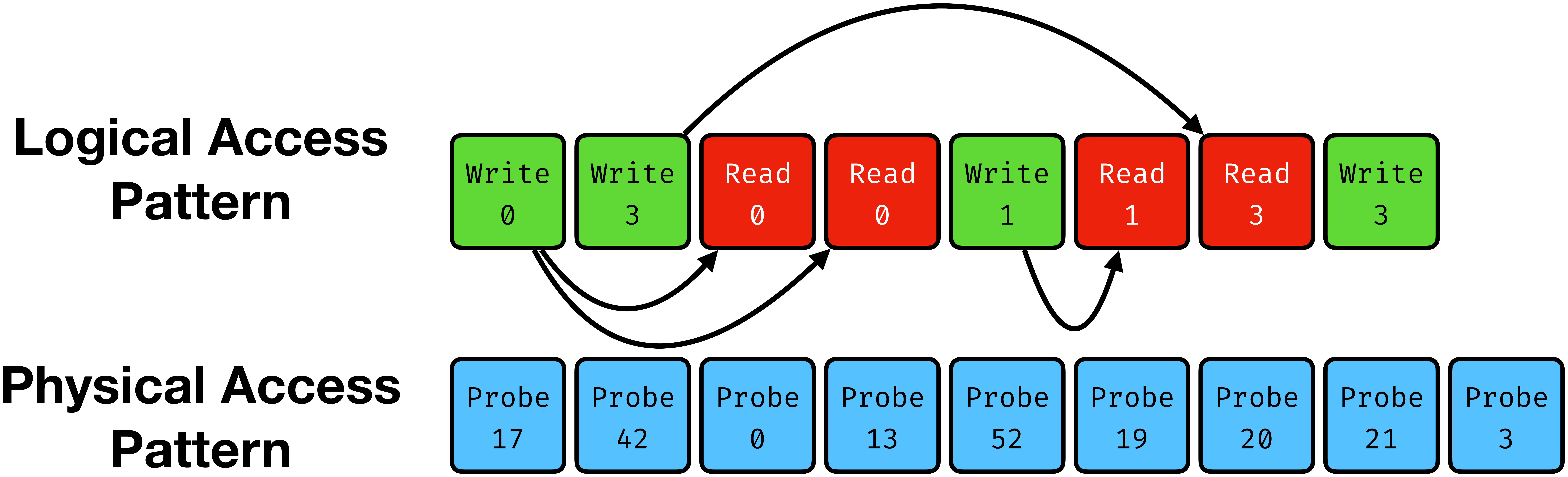
Probe 17 | Probe 42 | Probe 0


Logical Access Pattern
Write 0
Write 3
Physical Access Pattern
Probe 17
Probe 42
Probe 0
Probe 13
Probe 52
Probe 19

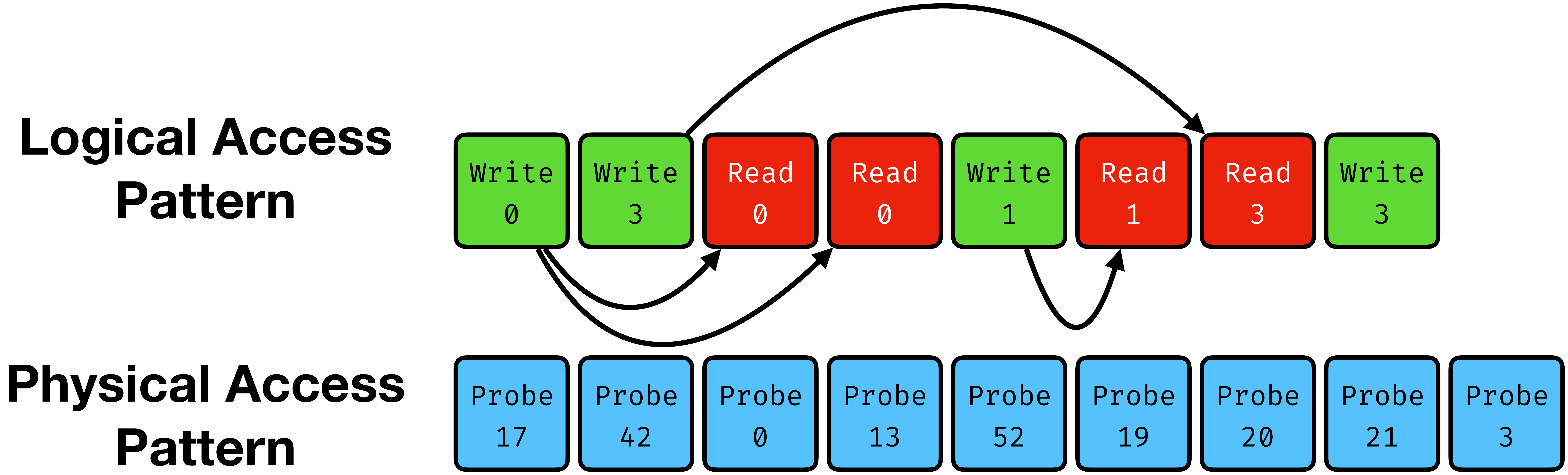

Logical Access Pattern
Write 0
Write 3
Read 0
Read 0
Write 1
Read 1
Read 3
Write 3
Physical Access Pattern
Probe 17
Probe 42
Probe 0
Probe 13
Probe 52
Probe 19
Probe 20
Probe 21
Probe 3

The logical access pattern implicitly has dependencies

The client must somehow get all data to move from the source to the target of the arrow

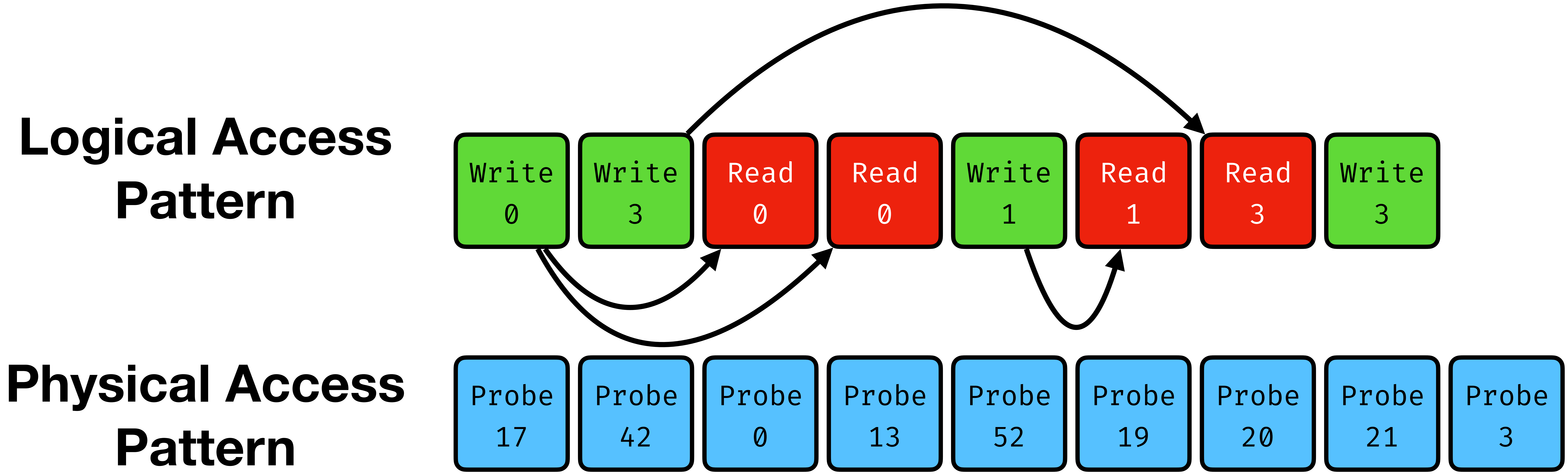


Basic Observation: If the client writes data using a particular sequence of probes, it must probe that same location again to read the data

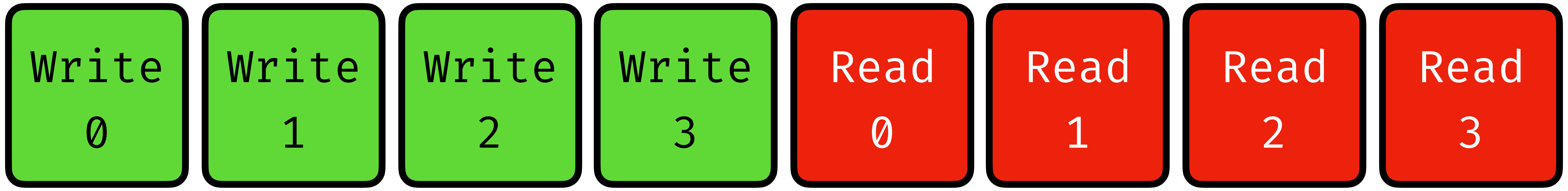


Basic Observation: If the client writes data using a particular sequence of probes, it must probe that same location again to read the data

We can construct access patterns that are particularly “difficult”

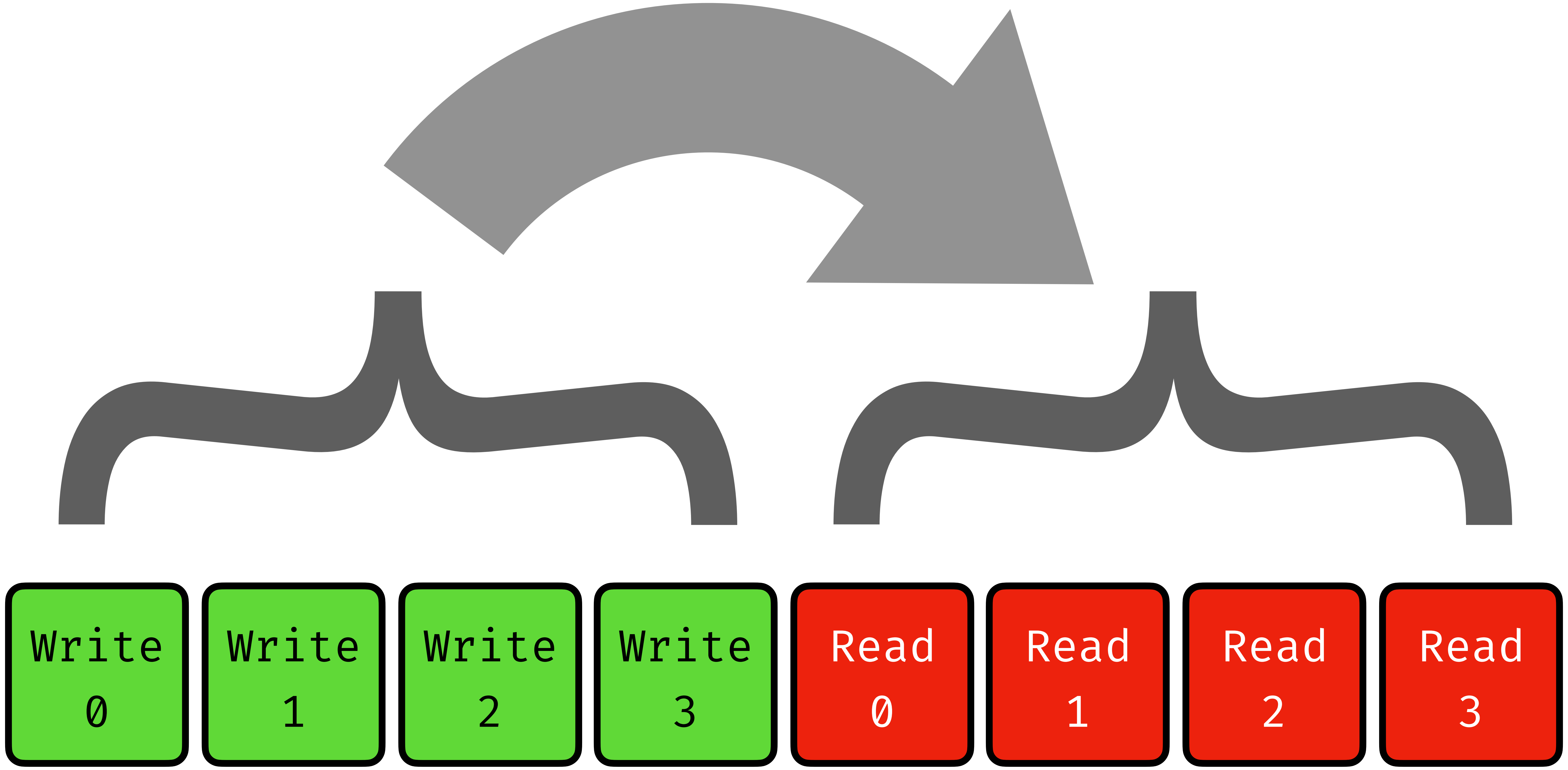


We can construct access patterns that are particularly “difficult”



The client has to move all data from the left half of the access pattern to the right half

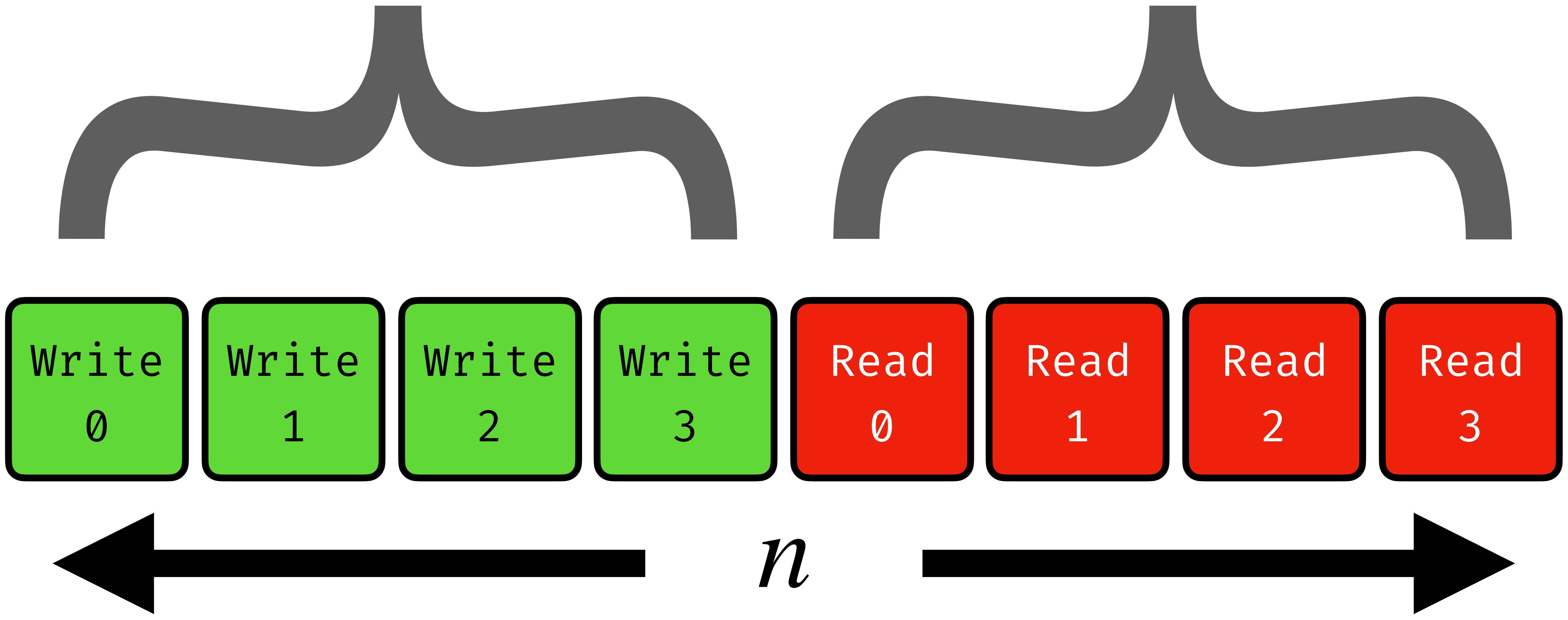
The client cannot remember all of its data locally, so it **must** send it to the server



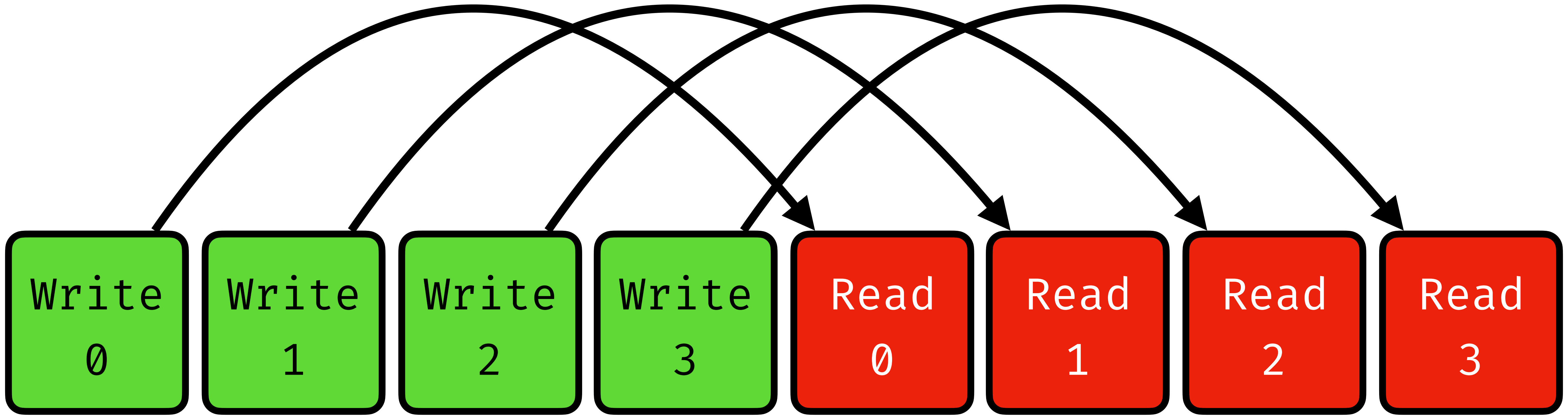
The client has to move all data from the left half of the access pattern to the right half

The client cannot remember all of its data locally, so it **must** send it to the server

Information theoretically, the client must save $O(n)$ items to the server

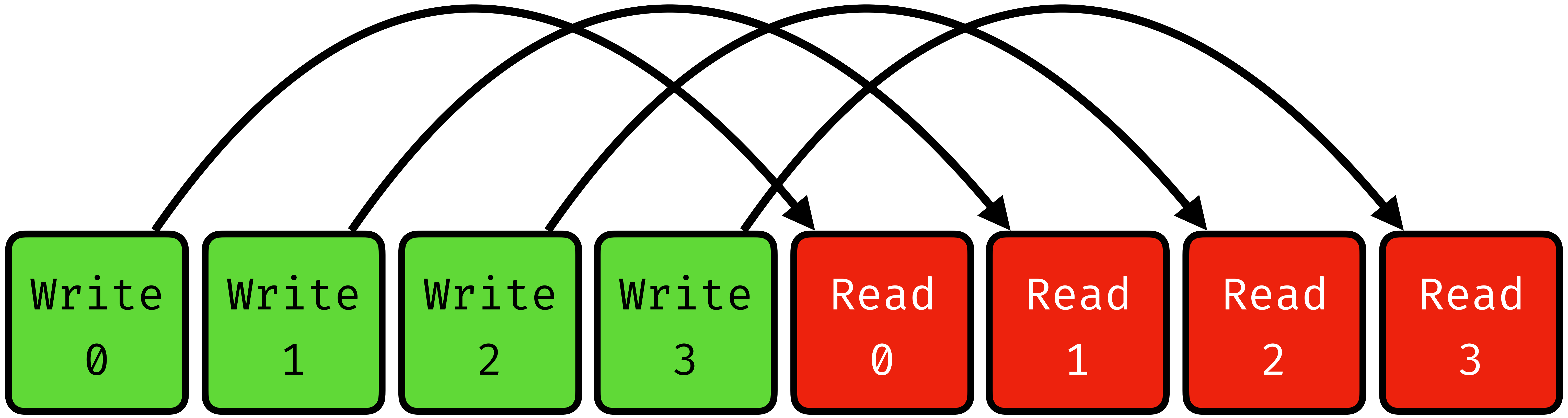


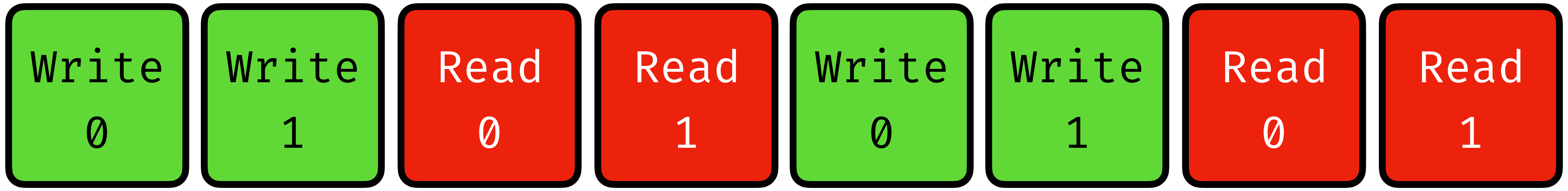
The client **must** perform $\Omega(n)$ repeated probes to accommodate this access pattern



The client **must** perform $\Omega(n)$ repeated probes to accommodate this access pattern

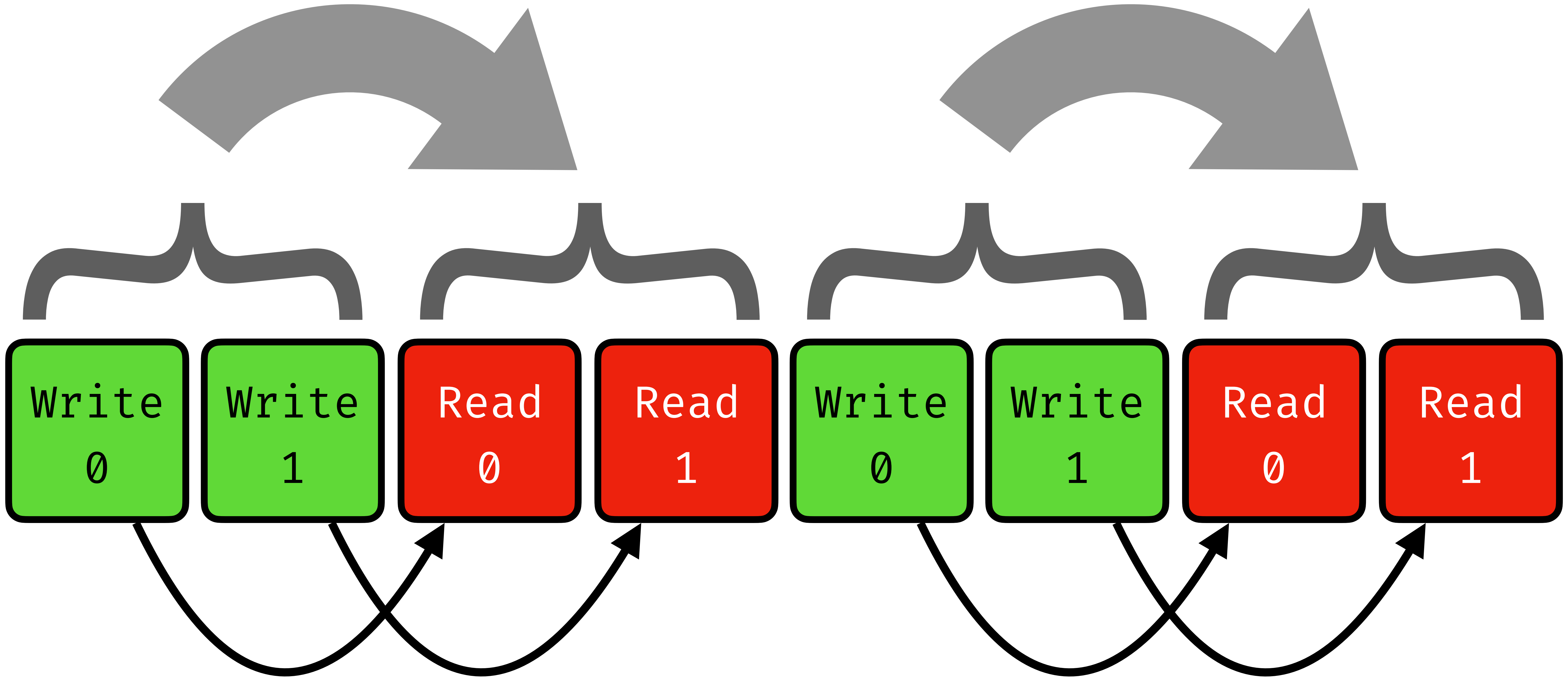
Key Idea: ORAM security implies that even for any other access pattern, there must be at least $\Omega(n)$ probes allowing to move all data from left to right

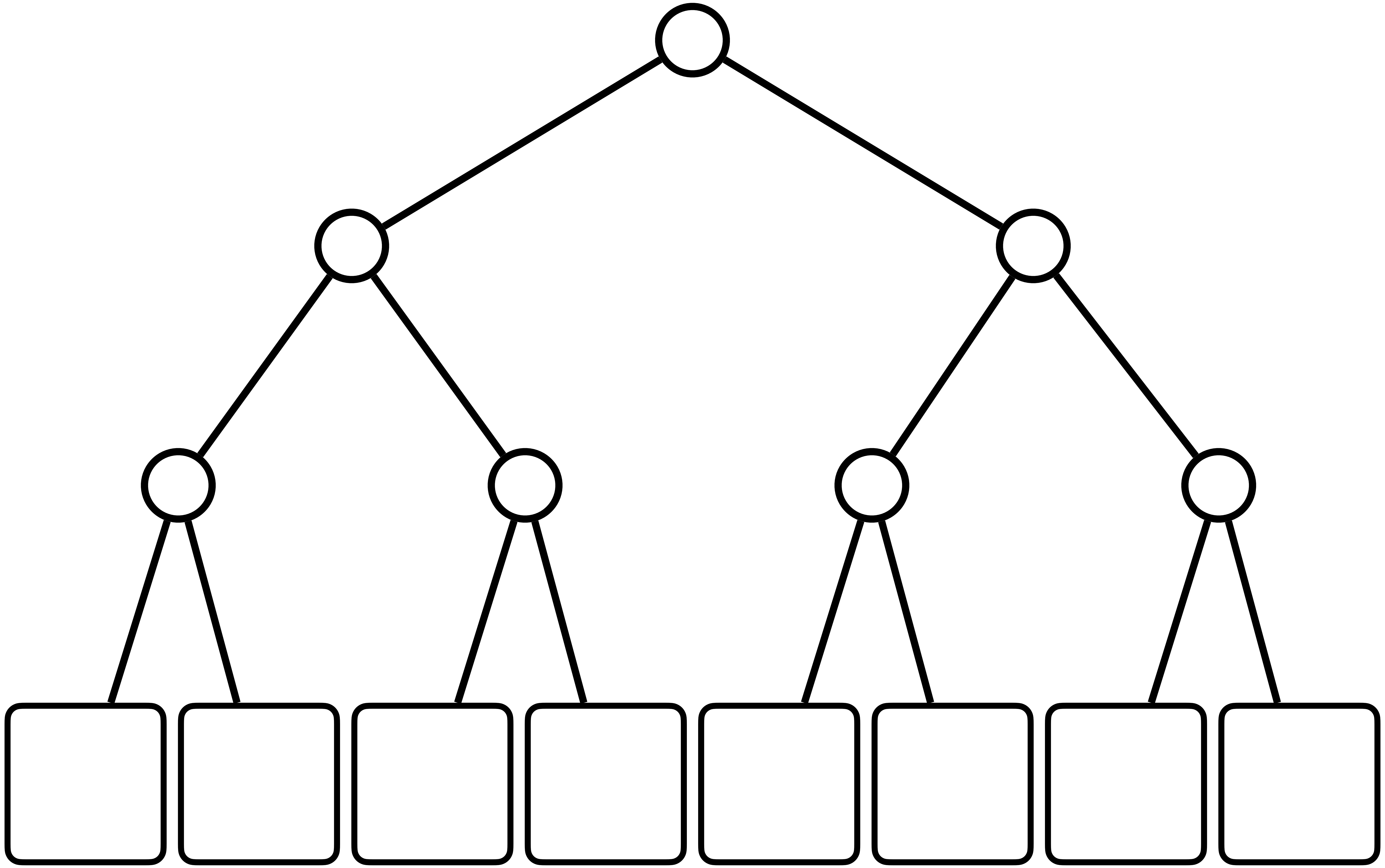


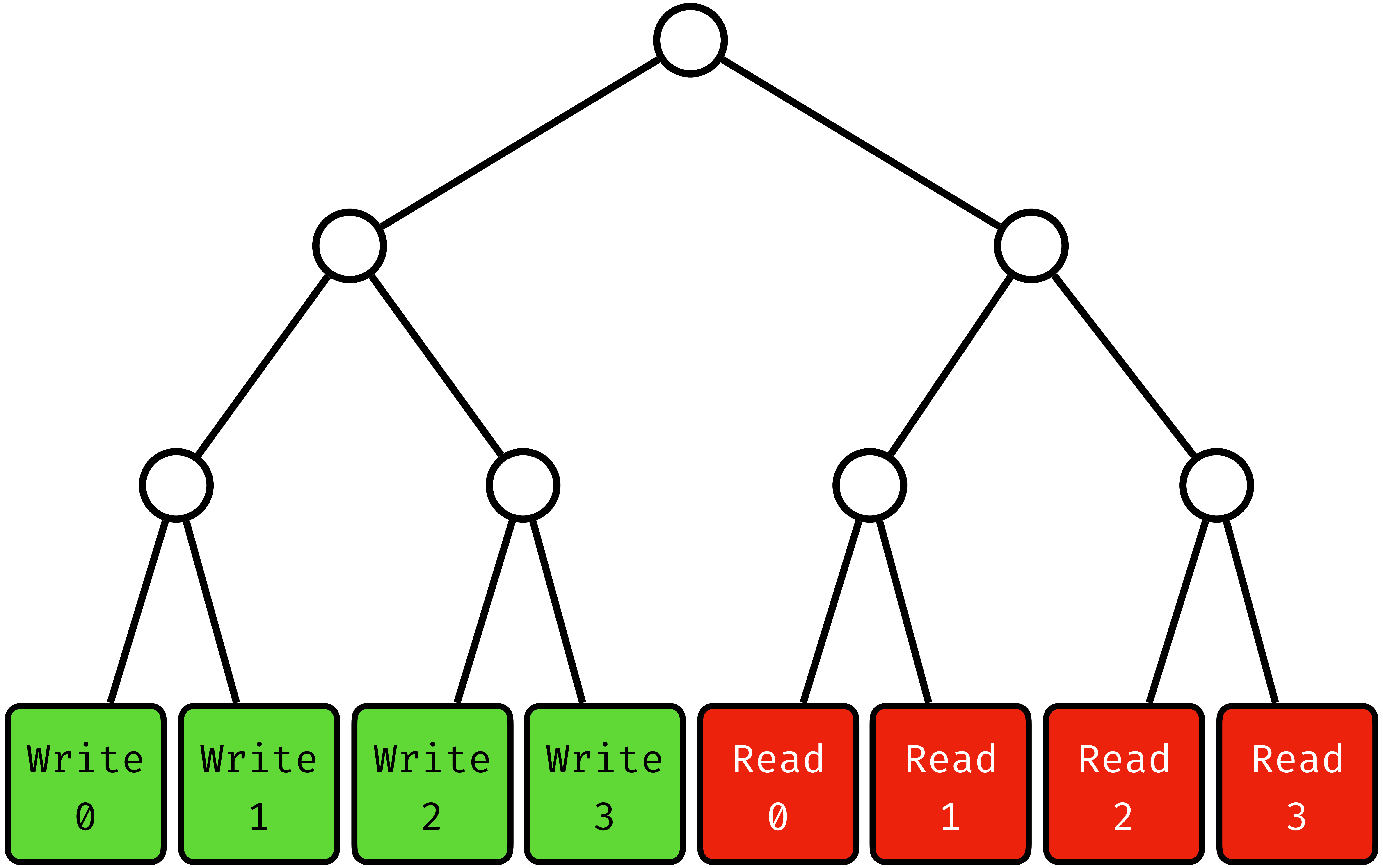

Write 0
Write 1
Read 0
Read 1
Write 0
Write 1
Read 0
Read 1

The client has to move all data from each left half of the access pattern to each right half

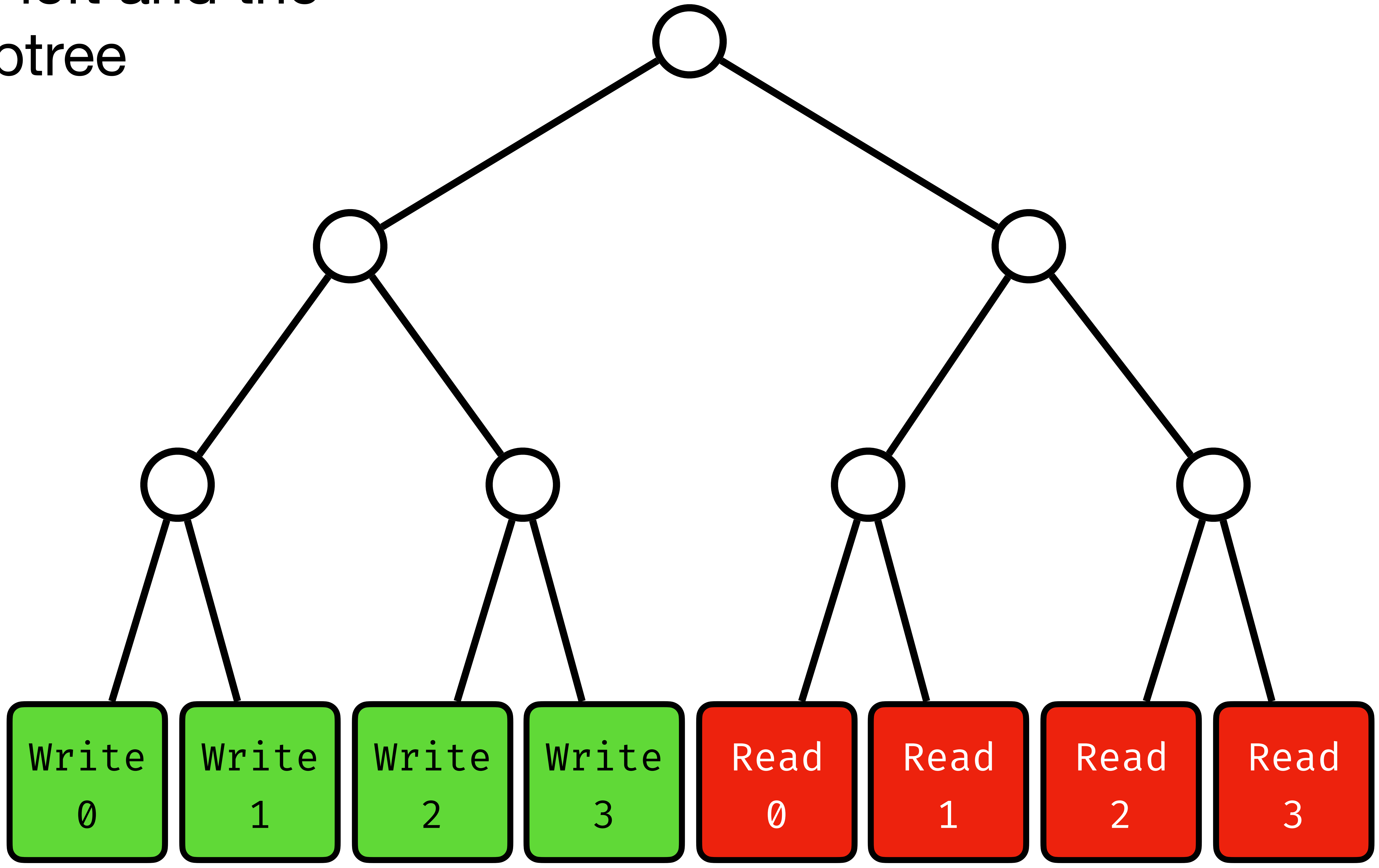
The client cannot remember all of its data locally, so it **must** send it to the server





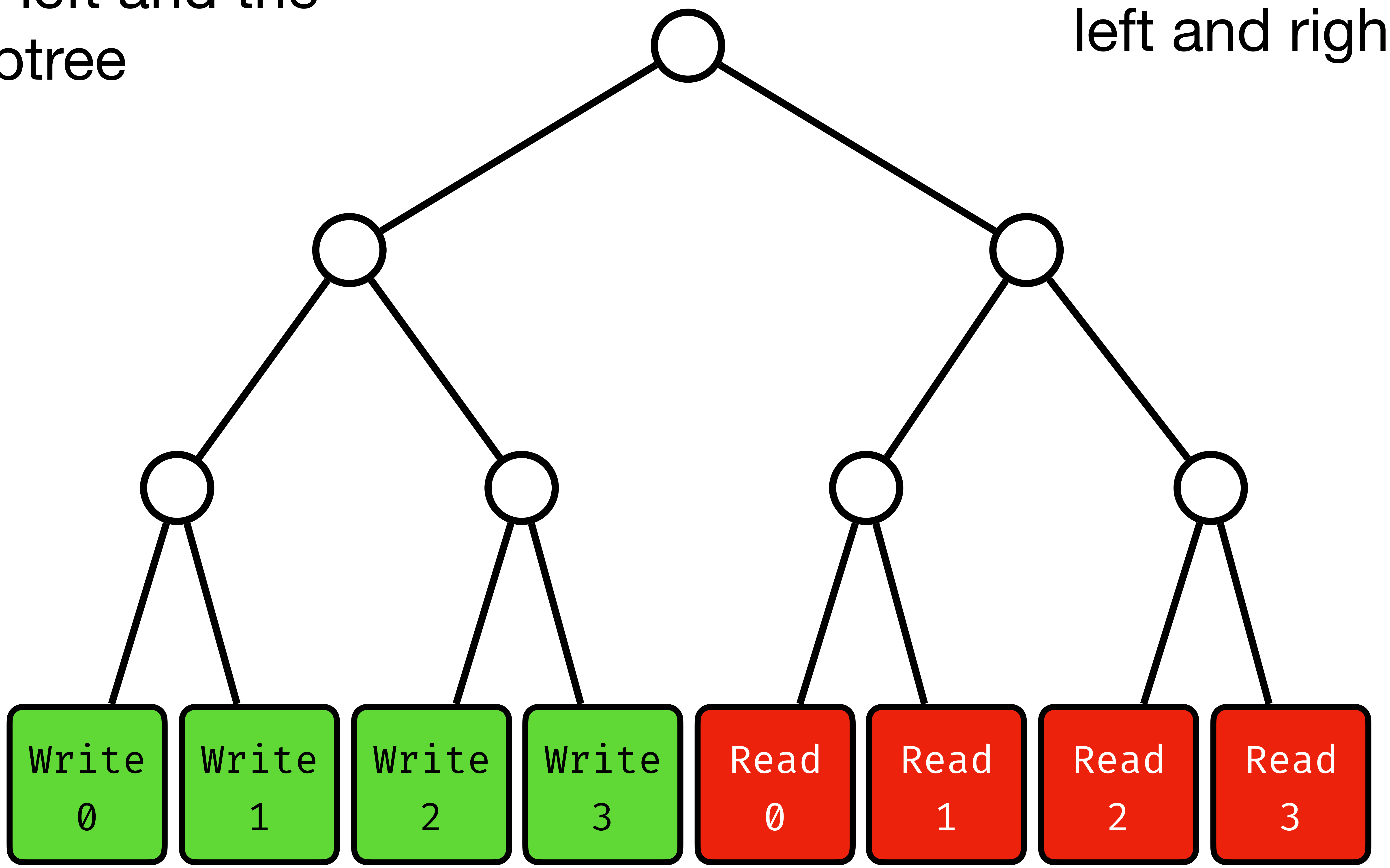

Write 0
Write 1
Write 2
Write 3
Read 0
Read 1
Read 2
Read 3

Client *must* request access to the same memory locations in the left and the right subtree



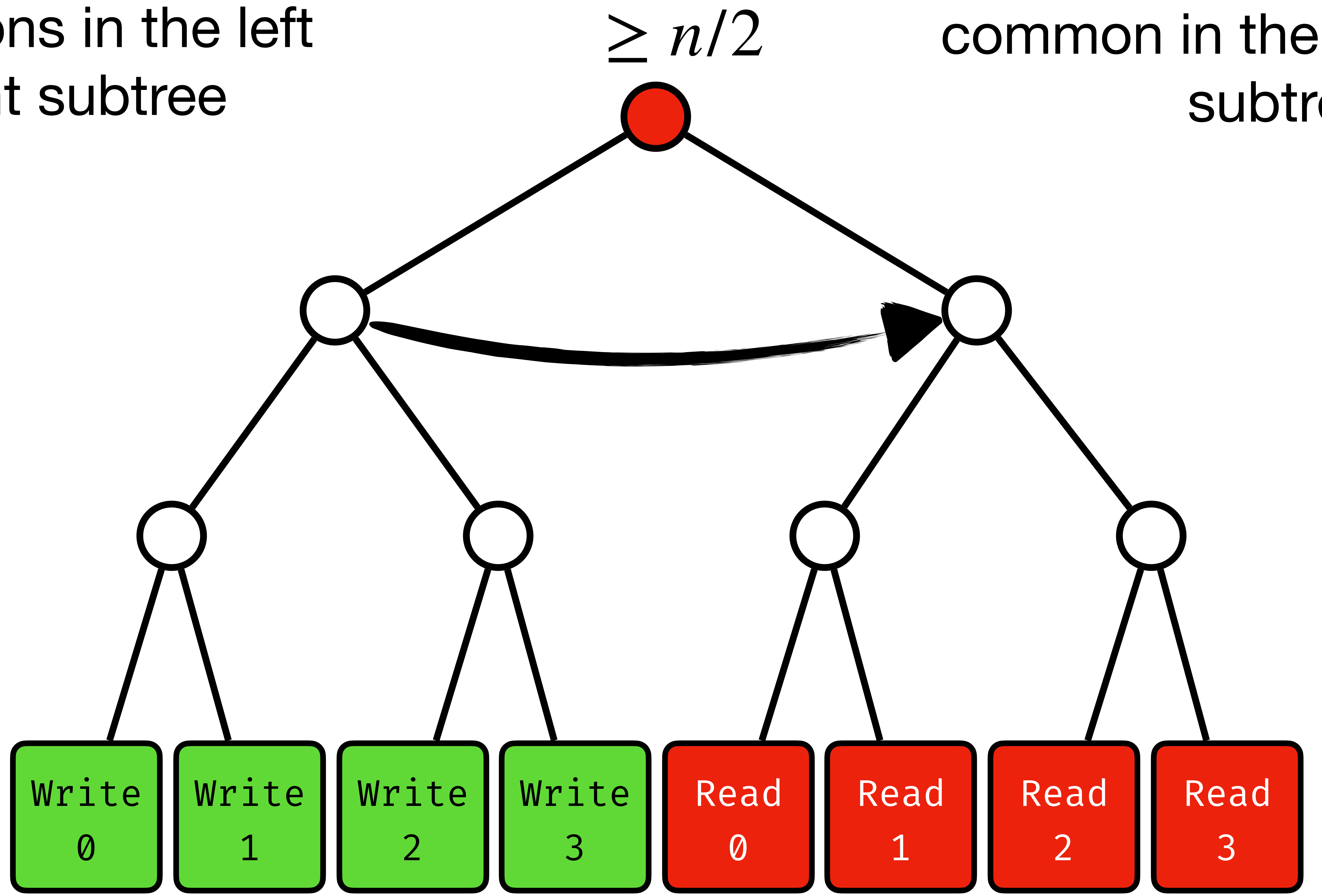
Client *must* request access to the same memory locations in the left and the right subtree

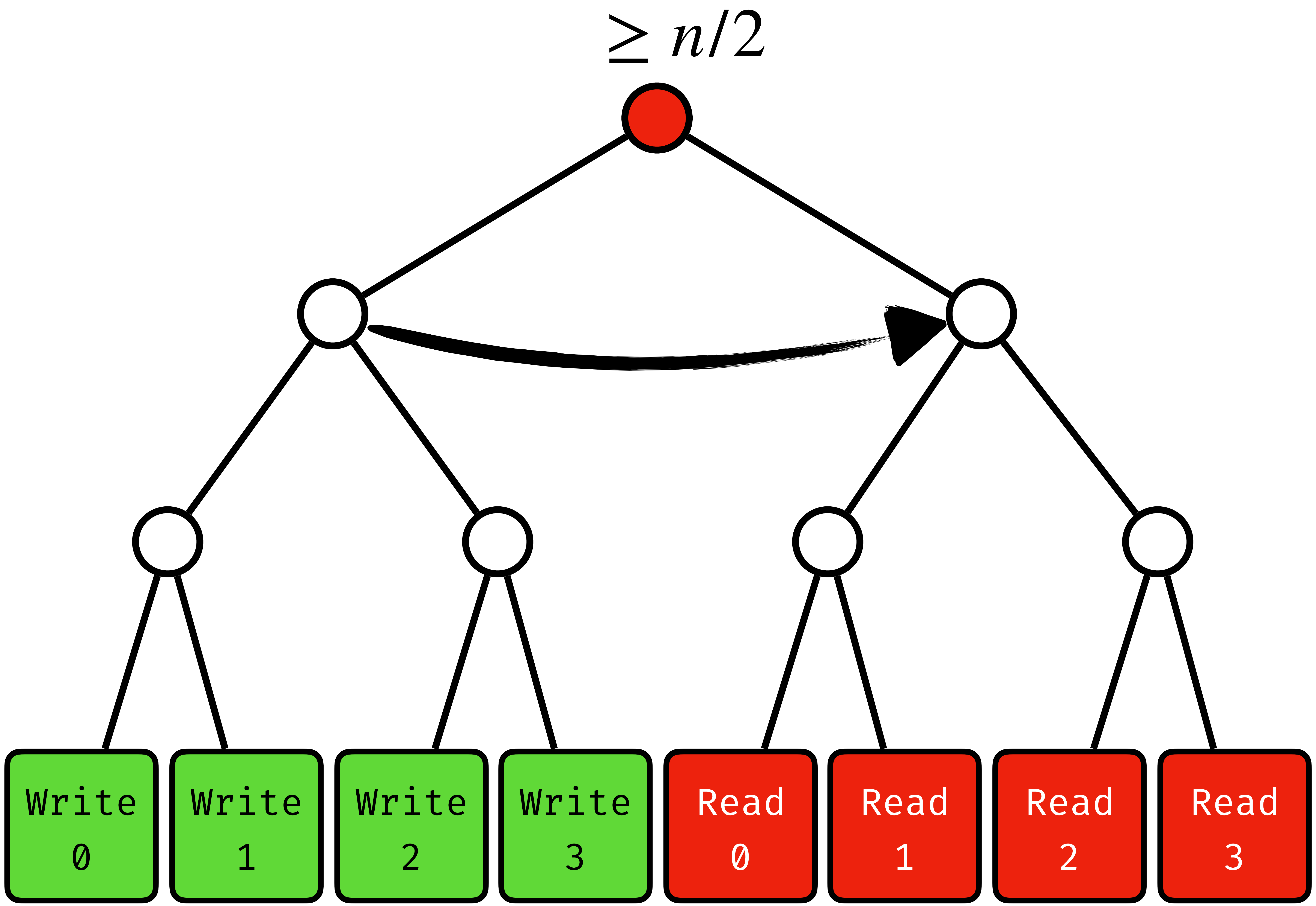
There are $\Omega(n)$ physical locations in common in the left and right subtrees

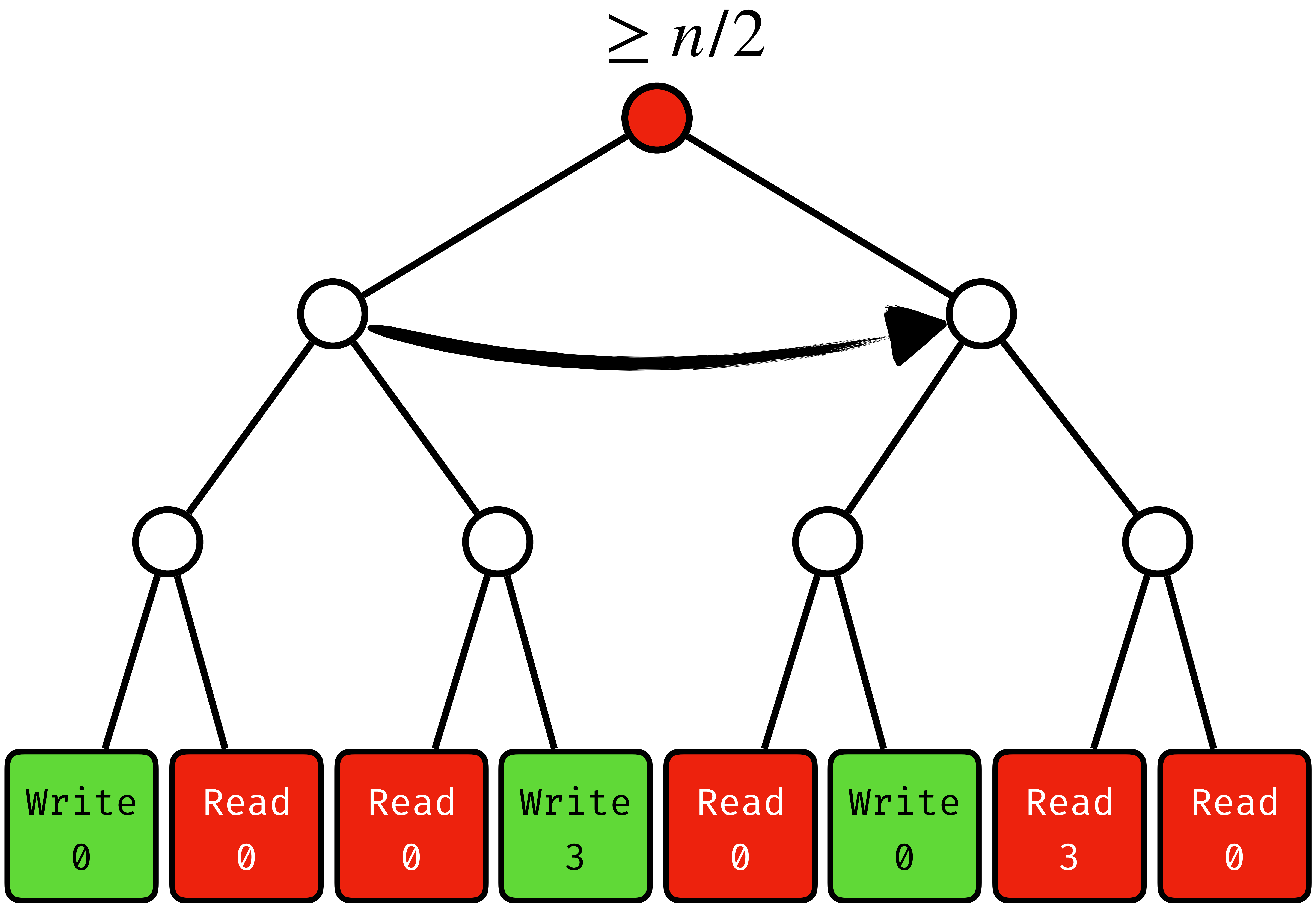


Client *must* probe the same memory locations in the left and the right subtree

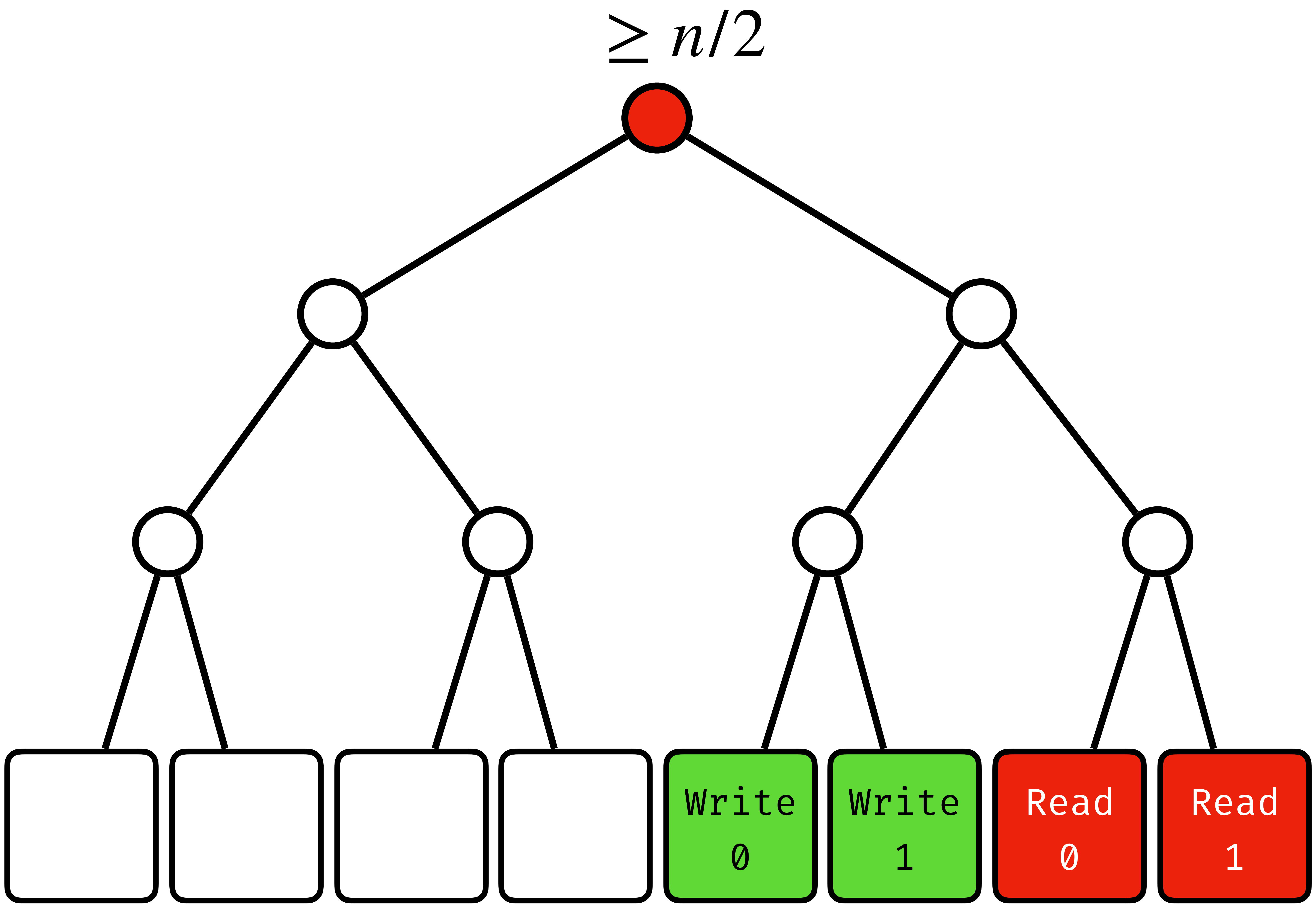
There are $\Omega(n)$ probes in common in the left and right subtrees



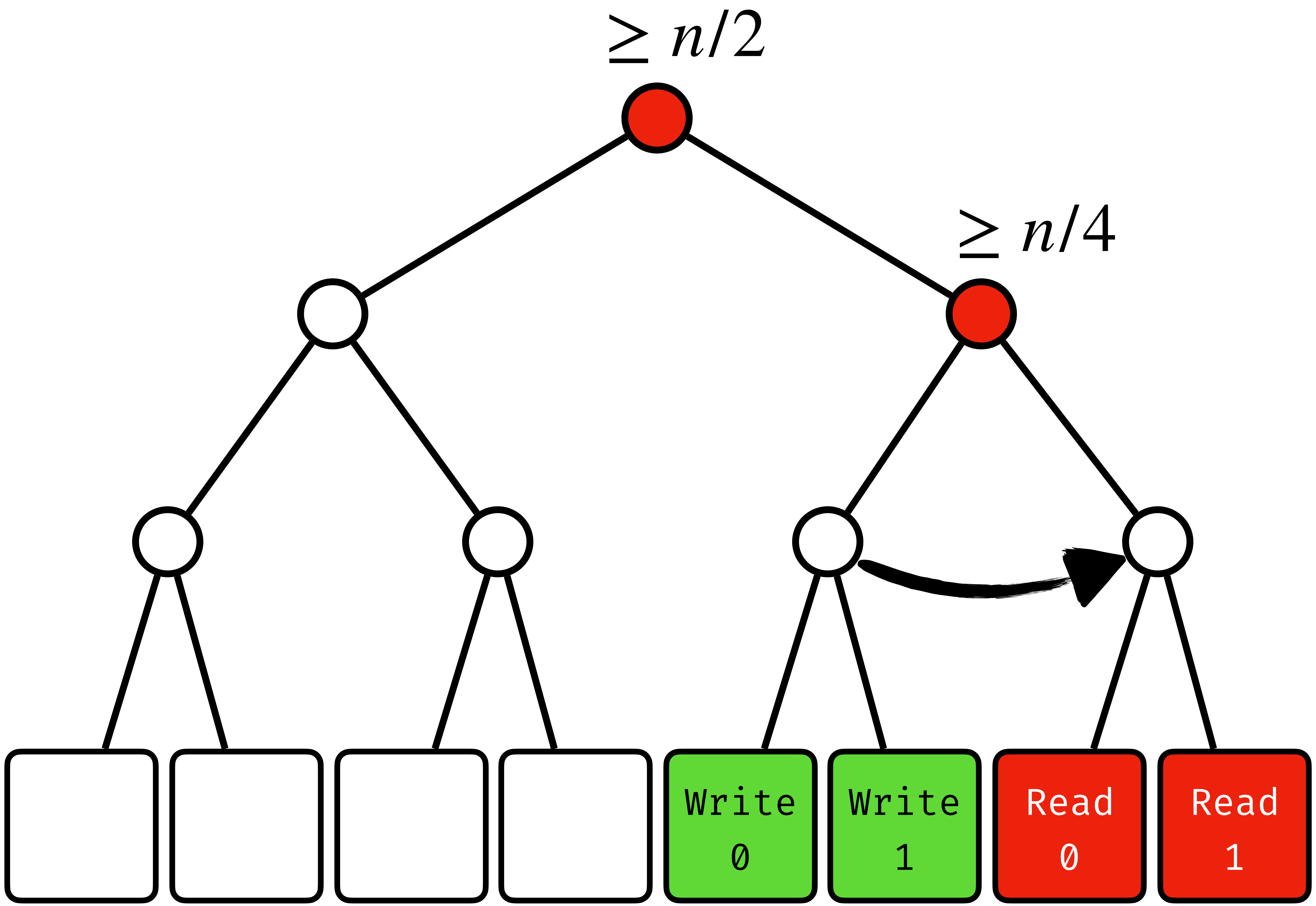

$\geq n/2$
Write 0
Write 1
Write 2
Write 3
Read 0
Read 1
Read 2
Read 3



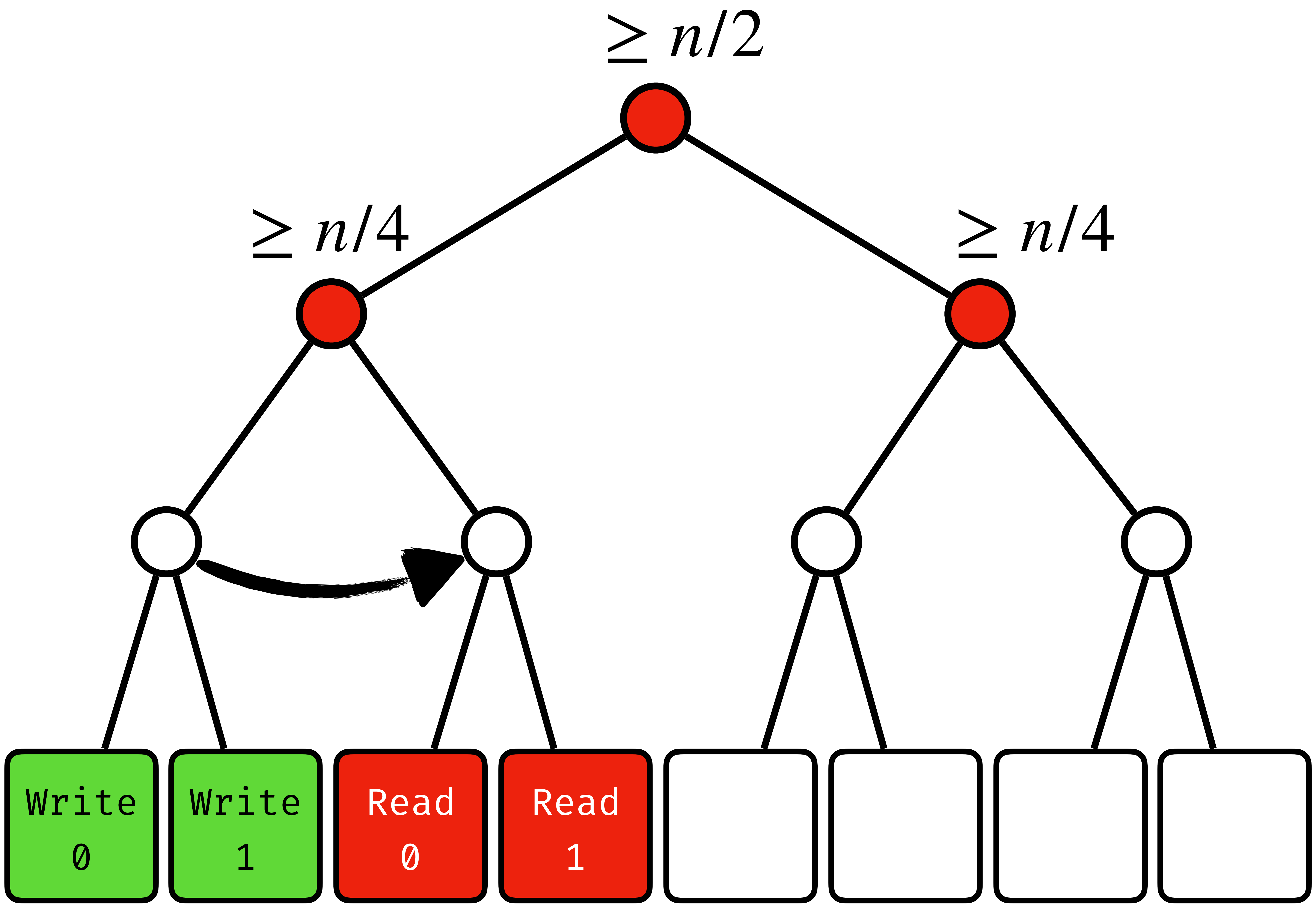
$\geq n/2$
Write 0
Read 0
Read 0
Write 3
Read 0
Write 0
Read 3
Read 0



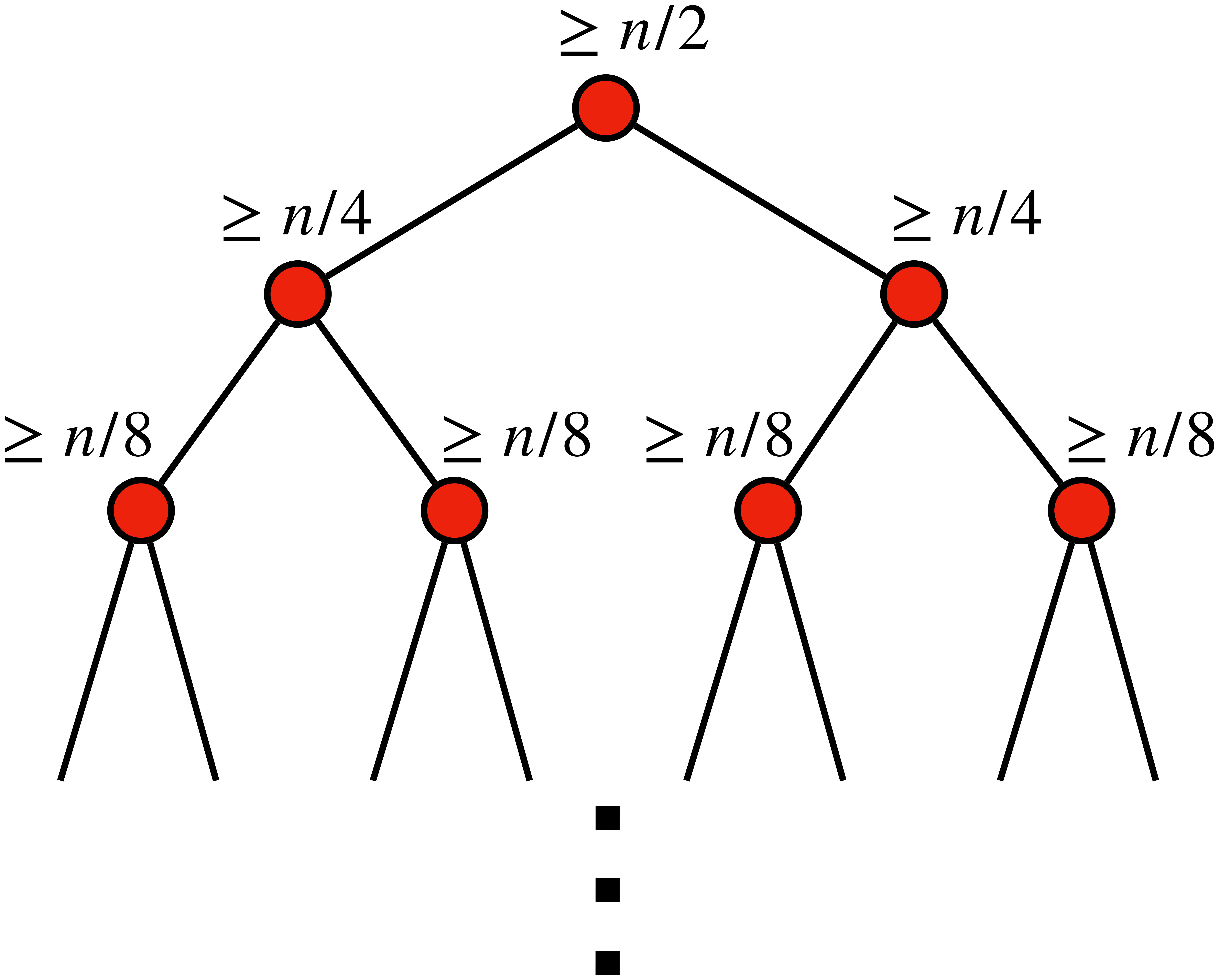
$\geq n/2$
Write 0
Write 1
Read 0
Read 1



$\geq n/2$
$\geq n/4$
Write 0
Write 1
Read 0
Read 1

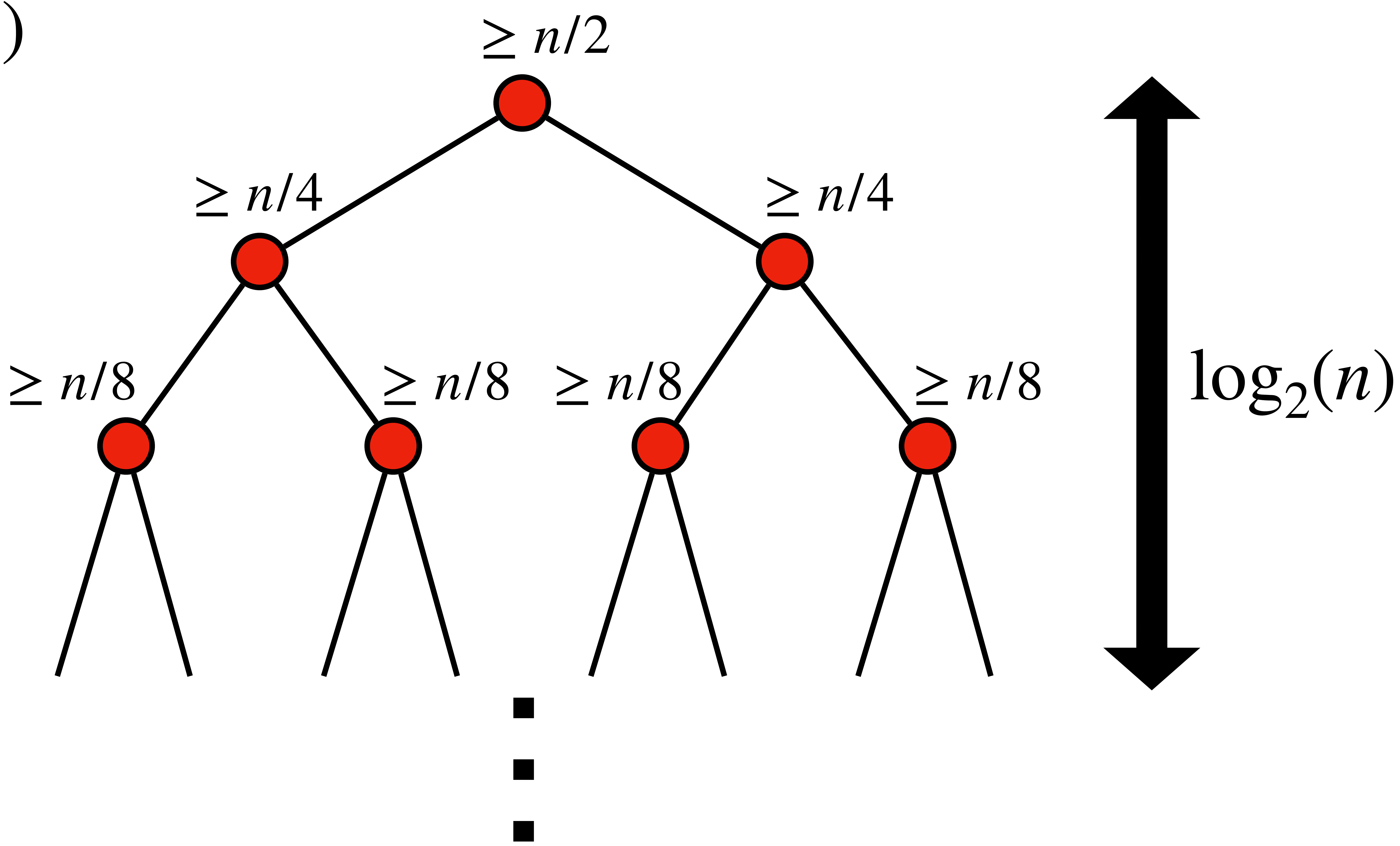


$\geq n/2$
$\geq n/4$
$\geq n/4$
Write 0
Write 1
Read 0
Read 1



≥ n/2
≥ n/4
≥ n/4
≥ n/8
≥ n/8
≥ n/8
≥ n/8

$\Omega(n \log n)$



Yes, There is an Oblivious RAM Lower Bound!

Kasper Green Larsen* and Jesper Buus Nielsen**

[1] Computer Science. Aarhus University
[2] Computer Science & DIGIT, Aarhus University

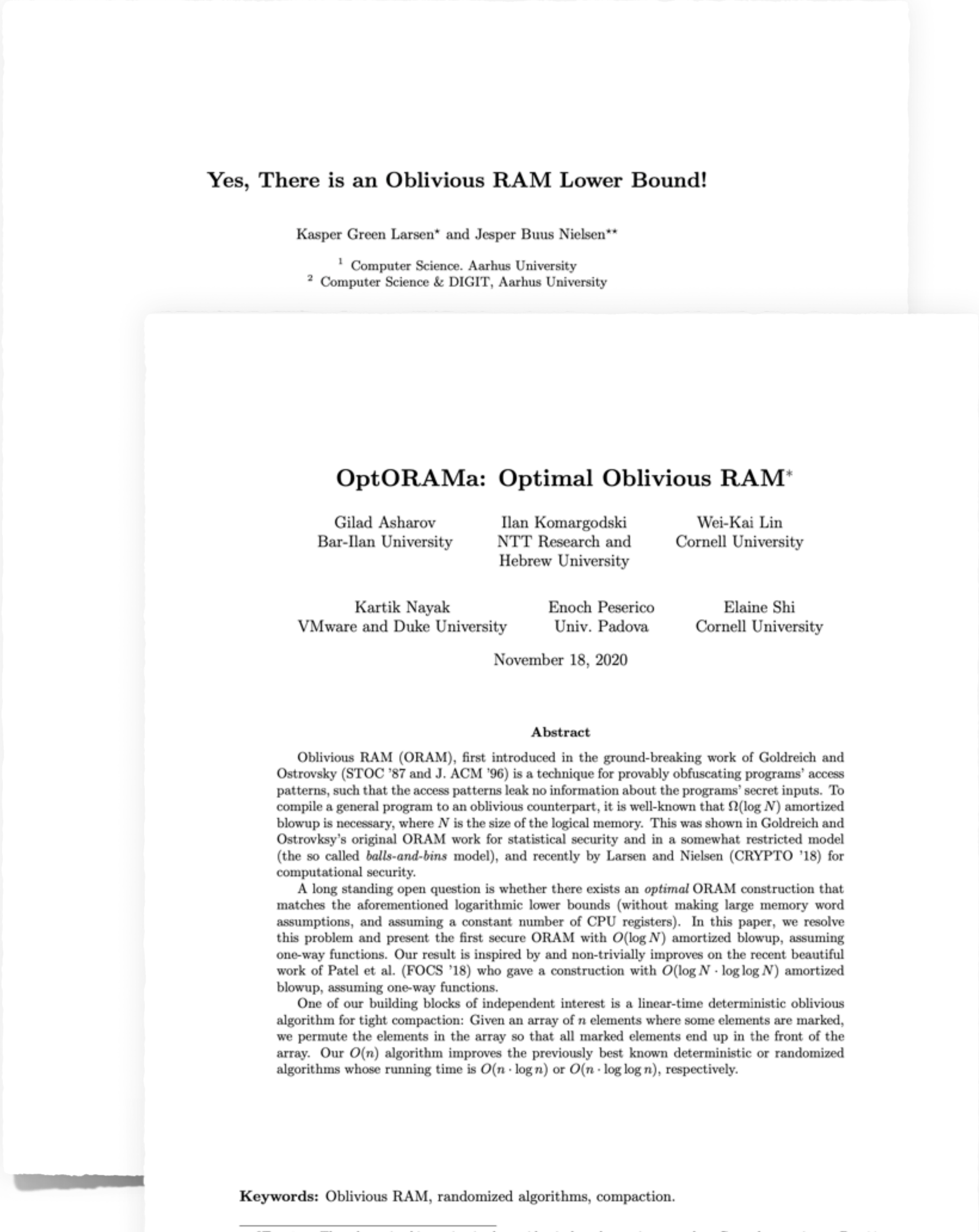
Abstract. An Oblivious RAM (ORAM) introduced by Goldreich and Ostrovsky [JACM'96] is a (possibly randomized) RAM, for which the memory access pattern reveals no information about the operations performed. The main performance metric of an ORAM is the bandwidth overhead, i.e., the multiplicative factor extra memory blocks that must be accessed to hide the operation sequence. In their seminal paper introducing the ORAM, Goldreich and Ostrovsky proved an amortized $\Omega(\lg n)$ bandwidth overhead lower bound for ORAMs with memory size n. Their lower bound is very strong in the sense that it applies to the "offline" setting in which the ORAM knows the entire sequence of operations ahead of time.
However, as pointed out by Boyle and Naor [ITCS'16] in the paper "Is there an oblivious RAM lower bound?", there are two caveats with the lower bound of Goldreich and Ostrovsky: (1) it only applies to "balls in bins" algorithms, i.e., algorithms where the ORAM may only shuffle blocks around and not apply any sophisticated encoding of the data, and (2), it only applies to statistically secure constructions. Boyle and Naor showed that removing the "balls in bins" assumption would result in super linear lower bounds for sorting circuits, a long standing open problem in circuit complexity. As a way to circumvent this barrier, they also proposed a notion of an "online" ORAM, which is an ORAM that remains secure even if the operations arrive in an online manner. They argued that most known ORAM constructions work in the online setting as well.
Our contribution is an $\Omega(\lg n)$ lower bound on the bandwidth overhead of any online ORAM, even if we require only computational security and allow arbitrary representations of data, thus greatly strengthening the lower bound of Goldreich and Ostrovsky in the online setting. Our lower bound applies to ORAMs with memory size n and any word size $r \geq 1$. The bound therefore asymptotically matches the known upper bounds when $r = \Omega(\lg^2 n)$.

1 Introduction

It is often attractive to store data at an untrusted party, and only retrieve the needed parts of it. Encryption can help ensure that the party storing the data

* Supported by a Villum Young Investigator grant 13163 and an AUFF starting grant.
** Supported by the European Union's Horizon 2020 research and innovation programme under grant agreement #731583 (SODA).

Any ORAM **must** have $\Omega(\log n)$ overhead



Yes, There is an Oblivious RAM Lower Bound!

Kasper Green Larsen* and Jesper Buus Nielsen**

[1] Computer Science. Aarhus University
[2] Computer Science & DIGIT, Aarhus University

OptORAMa: Optimal Oblivious RAM*

Gilad Asharov, Bar-Ilan University
Ilan Komargodski, NTT Research and Hebrew University
Wei-Kai Lin, Cornell University
Kartik Nayak, VMware and Duke University
Enoch Peserico, Univ. Padova
Elaine Shi, Cornell University

November 18, 2020

Abstract

Oblivious RAM (ORAM), first introduced in the ground-breaking work of Goldreich and Ostrovsky (STOC '87 and J. ACM '96) is a technique for provably obfuscating programs' access patterns, such that the access patterns leak no information about the programs' secret inputs. To compile a general program to an oblivious counterpart, it is well-known that $\Omega(\log N)$ amortized blowup is necessary, where N is the size of the logical memory. This was shown in Goldreich and Ostrovksy's original ORAM work for statistical security and in a somewhat restricted model (the so called *balls-and-bins* model), and recently by Larsen and Nielsen (CRYPTO '18) for computational security.

A long standing open question is whether there exists an *optimal* ORAM construction that matches the aforementioned logarithmic lower bounds (without making large memory word assumptions, and assuming a constant number of CPU registers). In this paper, we resolve this problem and present the first secure ORAM with $O(\log N)$ amortized blowup, assuming one-way functions. Our result is inspired by and non-trivially improves on the recent beautiful work of Patel et al. (FOCS '18) who gave a construction with $O(\log N \cdot \log\log N)$ amortized blowup, assuming one-way functions.

One of our building blocks of independent interest is a linear-time deterministic oblivious algorithm for tight compaction: Given an array of n elements where some elements are marked, we permute the elements in the array so that all marked elements end up in the front of the array. Our $O(n)$ algorithm improves the previously best known deterministic or randomized algorithms whose running time is $O(n \cdot \log n)$ or $O(n \cdot \log\log n)$, respectively.

Keywords: Oblivious RAM, randomized algorithms, compaction.

Any ORAM **must** have $\Omega(\log n)$ overhead

… and there exists an ORAM with $O(\log n)$ overhead