Yes, There is an Oblivious RAM Lower Bound!

Kasper Green Larsen

Jesper Buus Nielsen

DIGIT
AARHUS UNIVERSITY CENTRE FOR
DIGITALISATION BIG DATA AND DATA ANALYTICS

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF SCIENCE
AARHUS UNIVERSITY

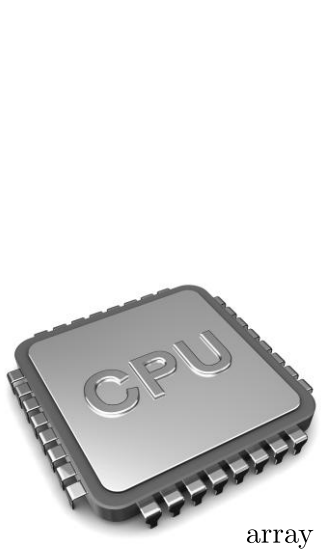
Oblivious RAM

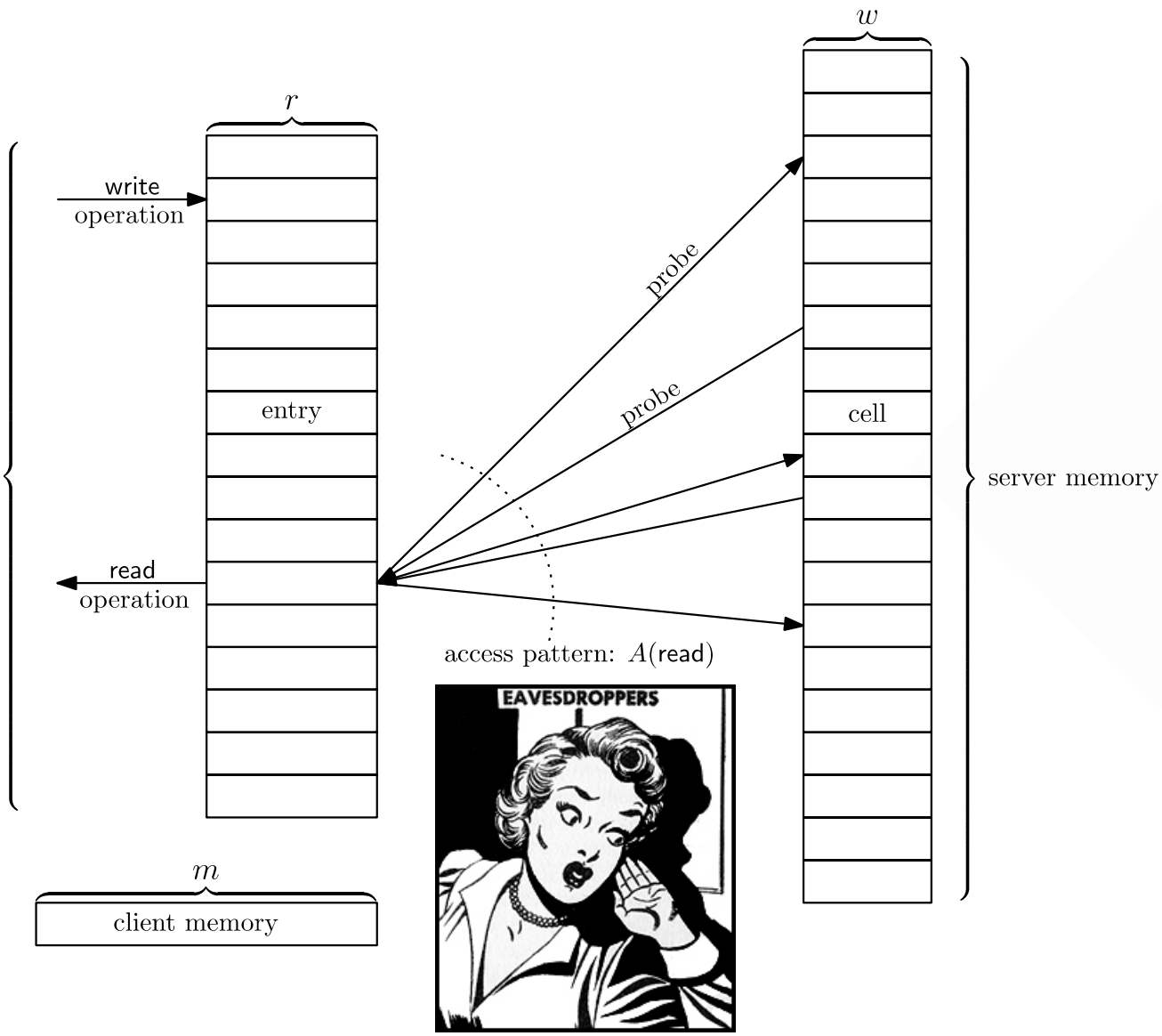
- Introduced by Goldreich and Ostrovsky in 1996
- “Encrypts” the memory access pattern of a random-access algorithm

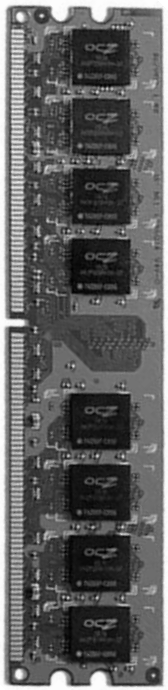
Oblivious RAM, Model (1/2)

- **Server**
  - A large, passive store of data, a random-access memory
- **Client**
  - Runs a program which simulates a large memory (an **array** with random access)
  - Has a small persistent memory
  - Outsources the rest of the data to the server
- **Eavesdropper**
  - Sees **access pattern** to the server
  - Does not see the actual data
- **Security**
  - For any two sequences of access to the array of the same length, the access pattern seen by Eavesdropper are indistinguishable

Oblivious RAM, Model (2/2)



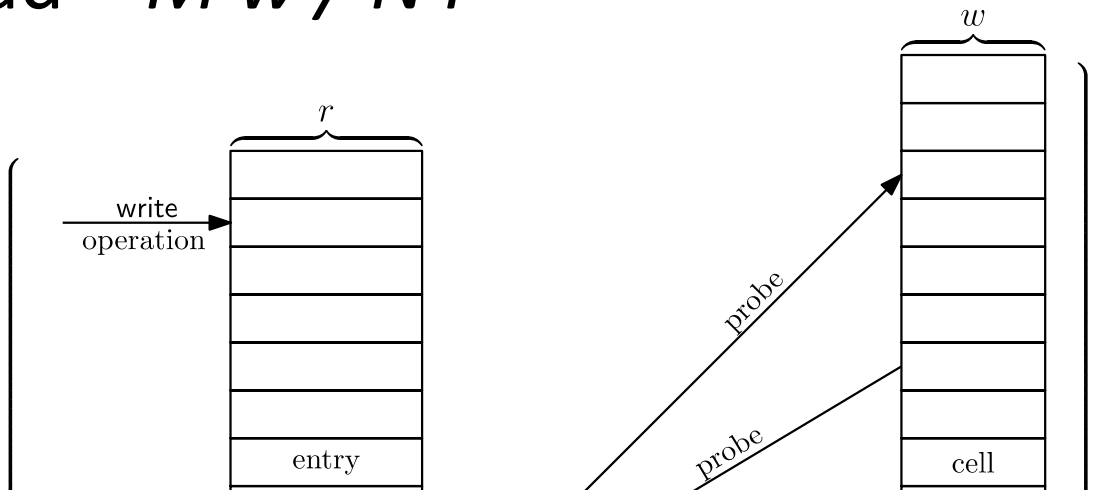




Bandwidth Overhead

- ORAMs have several obvious application: SGX, MPC, Cloud… In all of them the bandwidth overhead is important
- If after *N* accesses the ORAM makes *M* probes, then

Overhead = *M w / N r*



Upper Bounds

- Goldreich, Ostrovsky, 1996: *poly(log(N))*
- A **lot** of research on more efficient ORAMs
- PathORAM, 2013

  [Stefanov, van Dijk, Shi, Fletcher, Ren, Yu, Devadas, CCS'13]

  – Bandwidth overhead = *log(N)*
    - When $w = log(N)$ and $r = w^2$
- PanORAMa, 2018

  [Patel, Persiano, Raykova, Yeo, FOCS'18]

  – Bandwidth overhead = *log(N) log(log(N))*

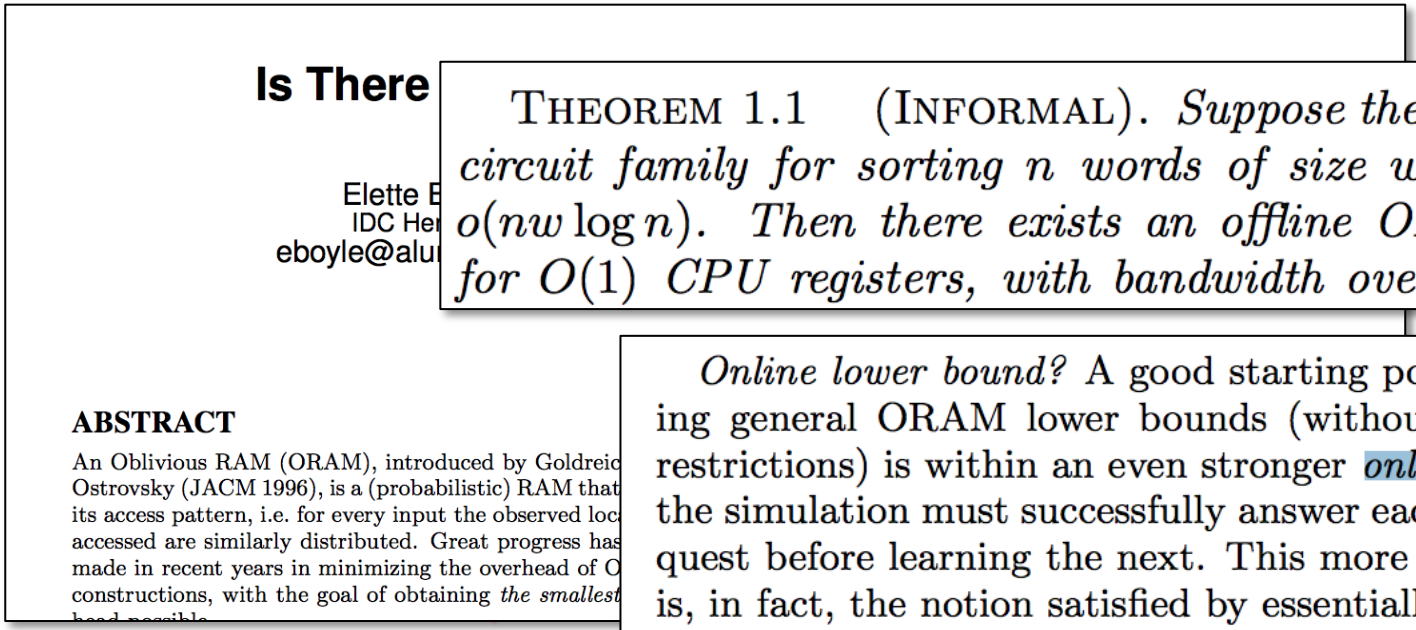
Lower Bounds: *log(N)*

- Goldreich, Ostrovsky, 1996: *log(N)*
- Model for lower bound:
  - Only balls-in-bins algorithms
    - The algorithm cannot look at the data being stored
    - Cannot use for instance error-correcting codes
  - Adversary has unbounded computing time
    - Cannot use computational cryptography
  - Holds even for off-line ORAMs
    - The ORAM is given the entire sequence of array accesses ahead of simulation time

30 years break: *log(N)*

- Goldreich, Ostrovsky, 1996: *log(N)*
- Model for lower bound:
  - Only balls-in-bins algorithms
    - The algorithm cannot look at the data being stored
    - Cannot use for instance error-correcting codes
  - Adversary has unbounded computing time
    - Cannot use computational cryptography
  - Holds even for off-line ORAMs
    - The ORAM is given the entire sequence of array accesses ahead of simulation time

2016: *log(N)*???


Is There

Elette B
IDC He
eboyle@alu

ABSTRACT

An Oblivious RAM (ORAM), introduced by Goldreic
Ostrovsky (JACM 1996), is a (probabilistic) RAM that
its access pattern, i.e. for every input the observed loc
accessed are similarly distributed. Great progress has
made in recent years in minimizing the overhead of O
constructions, with the goal of obtaining *the smallest*

THEOREM 1.1 (INFORMAL). *Suppose there exists a Boolean circuit family for sorting n words of size w-bits with size $o(nw \log n)$. Then there exists an offline ORAM compiler for $O(1)$ CPU registers, with bandwidth overhead $o(\log n)$.*

Online lower bound? A good starting point toward proving general ORAM lower bounds (without balls and bins restrictions) is within an even stronger *online* model, where the simulation must successfully answer each data access request before learning the next. This more stringent variant is, in fact, the notion satisfied by essentially all known positive results in ORAM. We propose a definition of online ORAM in Section 2.2 (Definition 2.10).

- Cannot use c

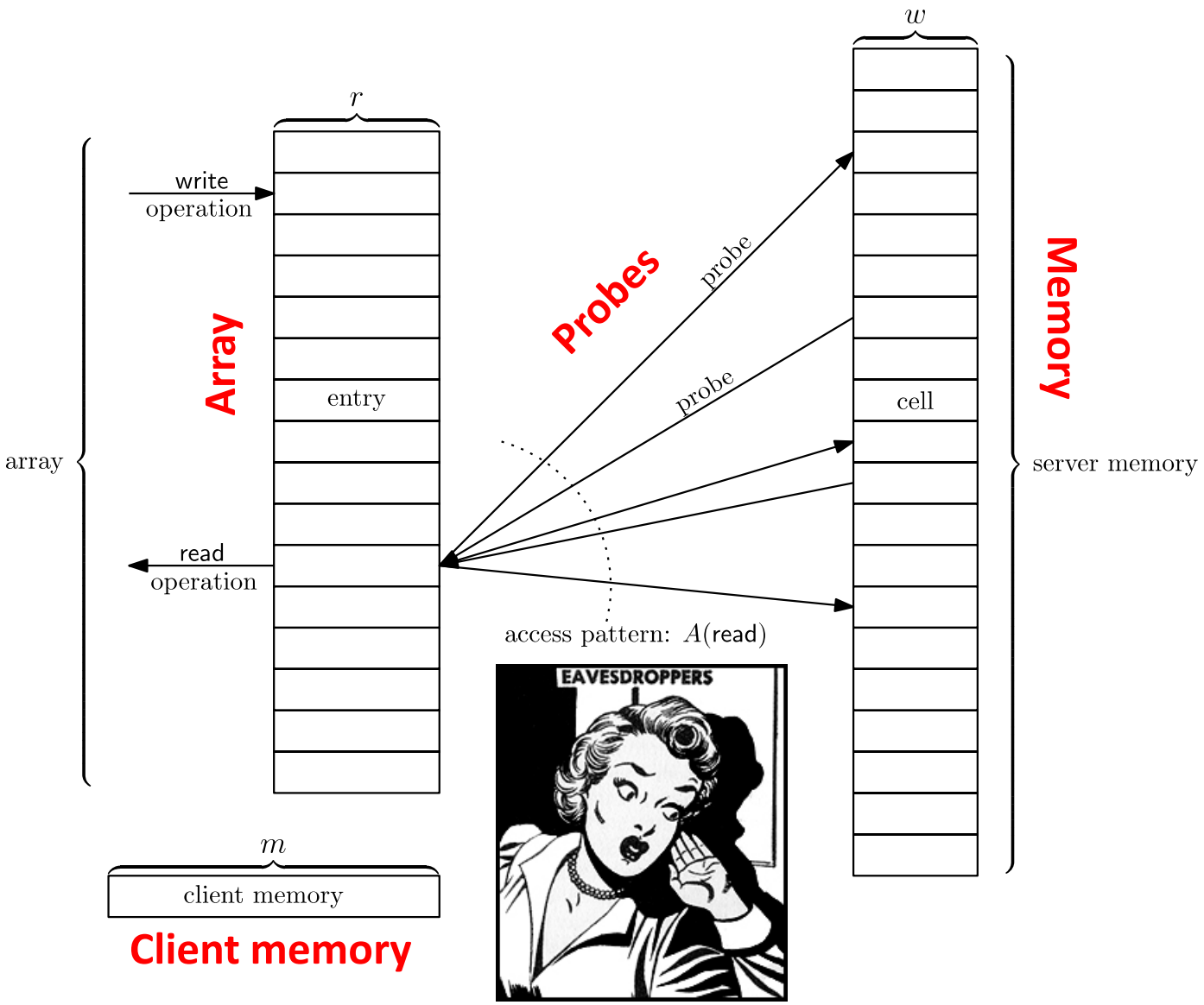
– Holds even for off-line ORAMs

- The ORAM is given the entire sequence of array accesses ahead of simulation time

Today:
Yes, There is an Oblivious RAM Lower Bound!

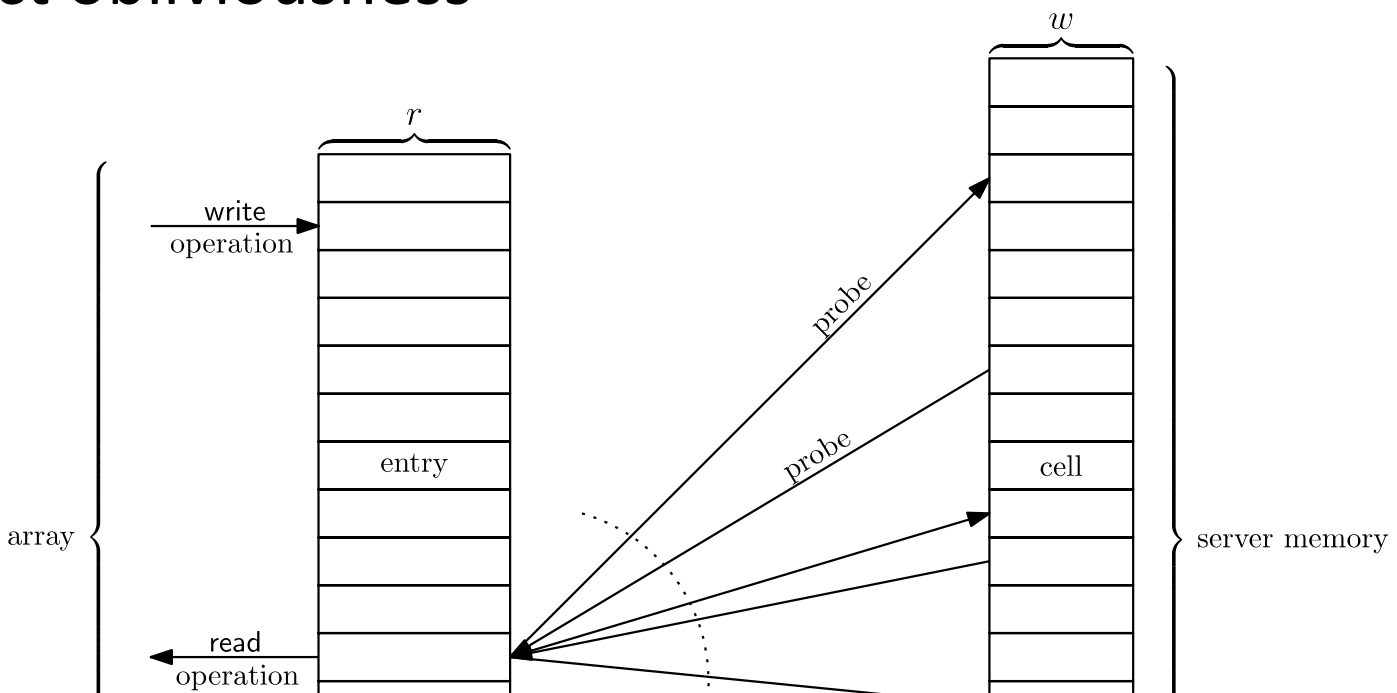
- Our model:
  - The ORAM algorithm can be arbitrary
    - ~~Balls-in-bins algorithms~~
  - The adversary must be efficient
    - ~~Adversary has unbounded computing time~~
  - Holds only for **on-line** ORAMs
    - The ORAM is given the array accesses to process one at a time
    - Anyway what is needed in all applications

Oblivious RAM, Model



Proof

- Simple case:
  - No client memory
  - Perfect correctness
  - Perfect obliviousness
  - r = w



How many times must the read-sequence probe a cell which was last time probed during the write-sequence?

LOGARITHMIC LOWER BOUNDS IN THE CELL-PROBE MODEL*

MIHAI PĂTRAŞCU† AND ERIK D. DEMAINE†

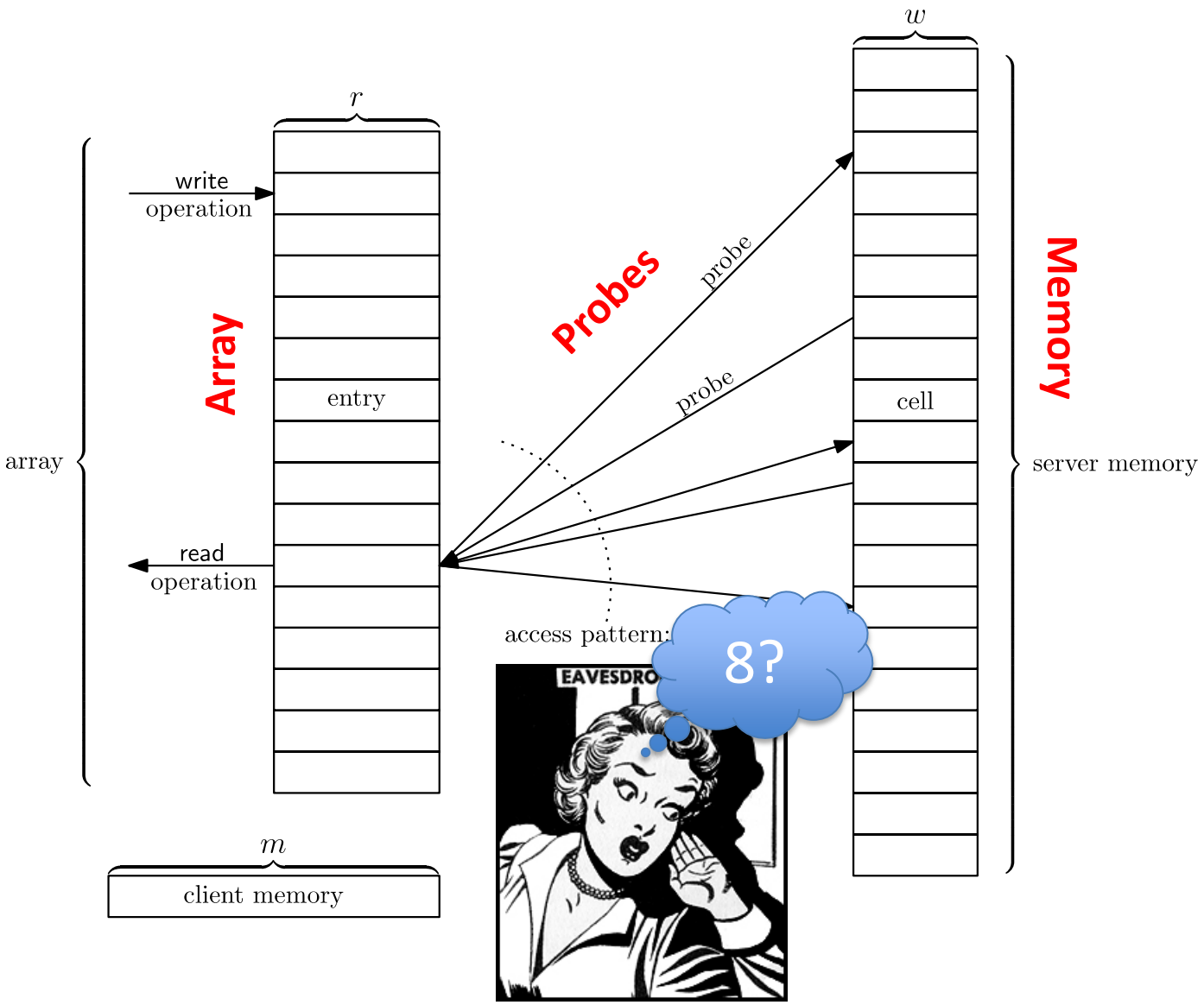
Abstract. We develop a new technique for proving cell-probe lower bounds on dynamic data structures. This technique enables us to prove an amortized randomized $\Omega(\lg n)$ lower bound per operation for several data structural problems on n elements, including partial sums, dynamic connectivity among disjoint paths (or a forest or a graph), and several other dynamic graph problems (by simple reductions). Such a lower bound breaks a long-standing barrier of $\Omega(\lg n/\lg\lg n)$ for any dynamic language membership problem. It also establishes the optimality of several existing data structures, such as Sleator and Tarjan's dynamic trees. We also prove the first $\Omega(\log_B n)$ lower bound in the external-memory model without assumptions on the data structure (such as the comparison model). Our lower bounds also give a query-update trade-off curve matched, e.g., by several data structures for dynamic connectivity in graphs. We also prove matching upper and lower bounds for partial sums when parameterized by the word size and the maximum additive change in an update.

w(1,r_1) w(2,r_2) w(3,r_3) w(4 ... 3)

How many times must the read-sequence probe a cell which was last time probed during the write-sequence?

w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)

Oblivious RAM, Model (2/2)



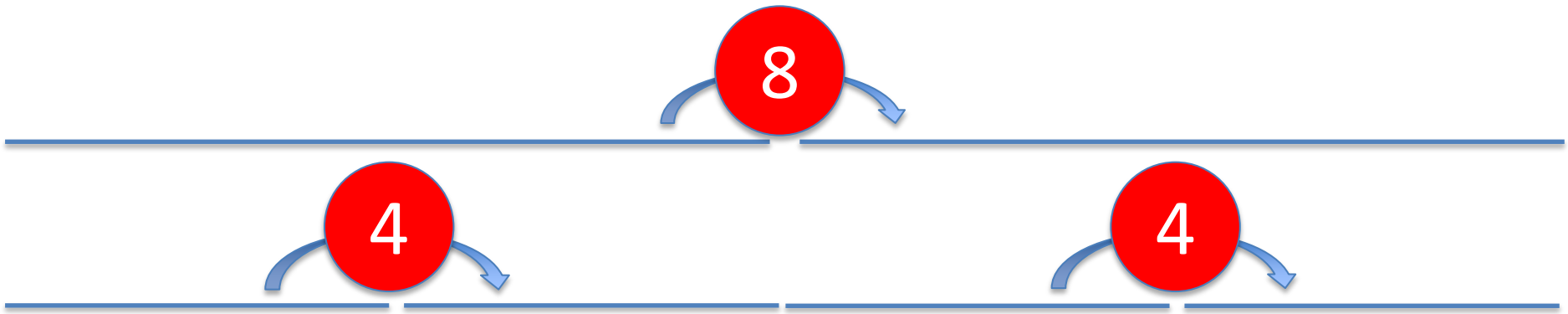
How many times must the read-sequence probe a cell which was last time probed during the write-sequence?

w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)

How many times mus
read-sequence prob
which was last time
during the first write-s

How many times must the second read-sequence probe a cell which was last time probed during the second write-sequence?

$w(1,r_1)$	$w(2,r_2)$	$w(3,r_3)$	$w(4,r_4)$	r(1)	r(2)	r(3)	r(4)	$w(5,r_5)$	$w(6,r_6)$	$w(7,r_7)$	$w(8,r_8)$	r(5)	r(6)	r(7)	r(8)

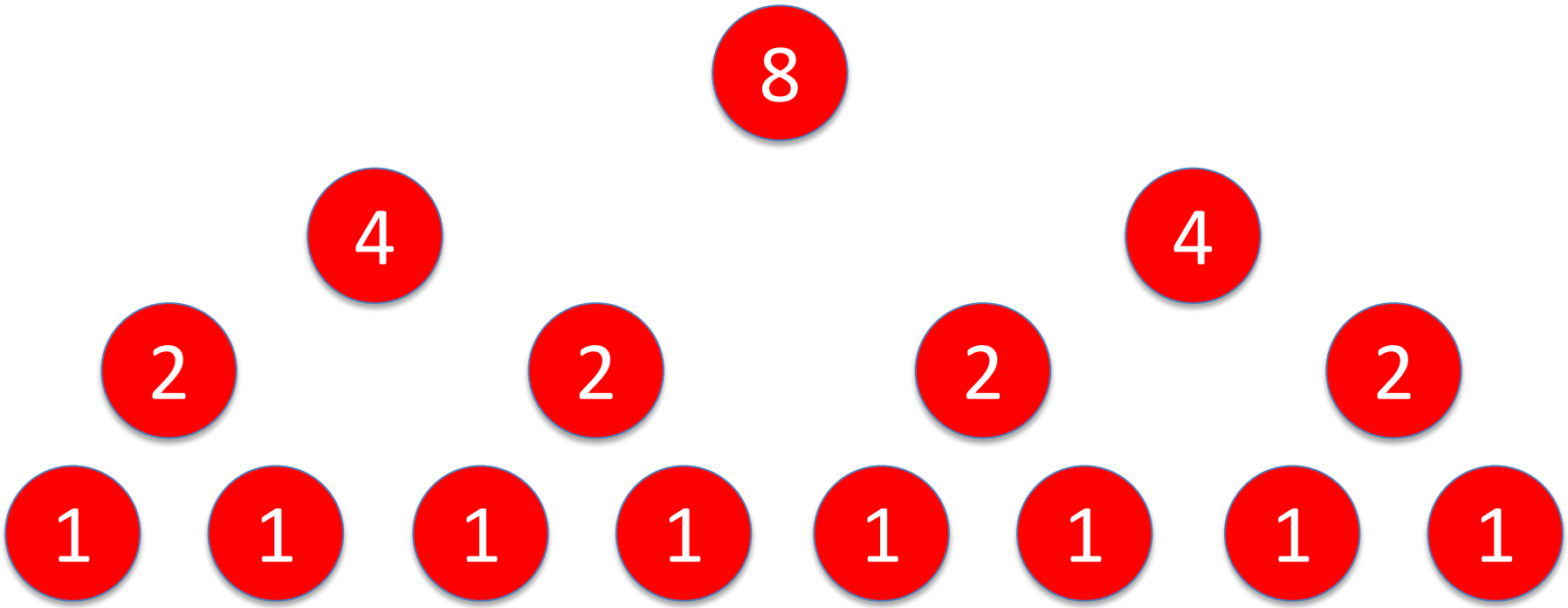


The probes counted in different circles are distinct!

w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)

2 2 2 2

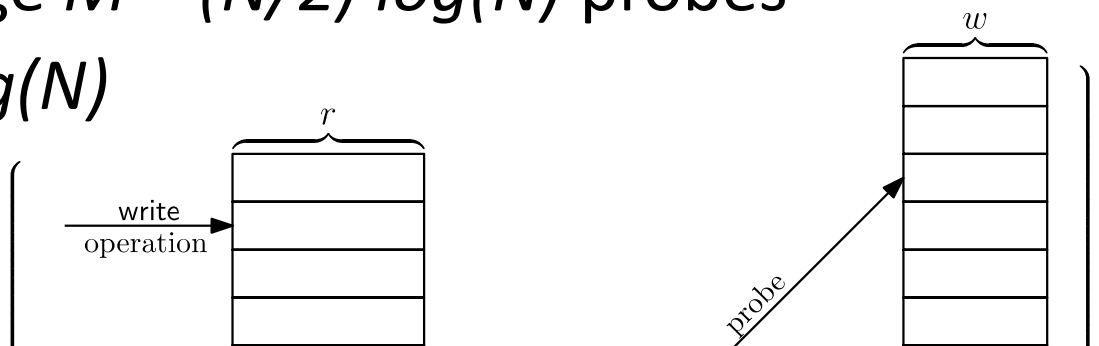
How many times must the first read-sequence probe a cell which was last time probed during the first write-sequence?



w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	w(0,0)	R(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)	r(0)

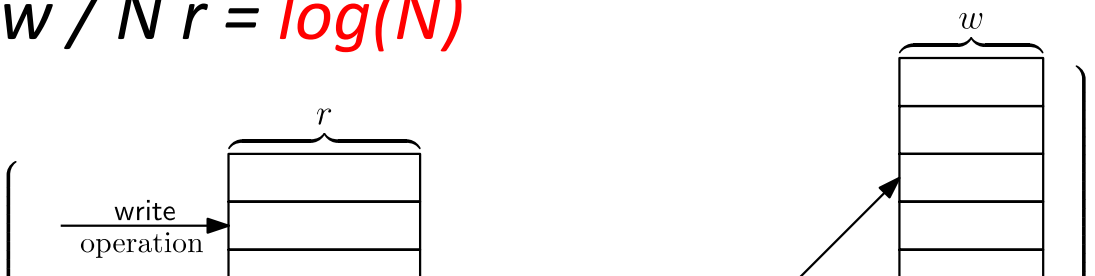
Theorem

- Easy case:
  - No client memory
  - Perfect correctness
  - Perfect obliviousness
  - r = w
- **Theorem**
  - Any ORAM simulating *N* accesses makes on at least on average *M = (N/2) log(N)* probes
  - Overhead = *log(N)*



Theorem

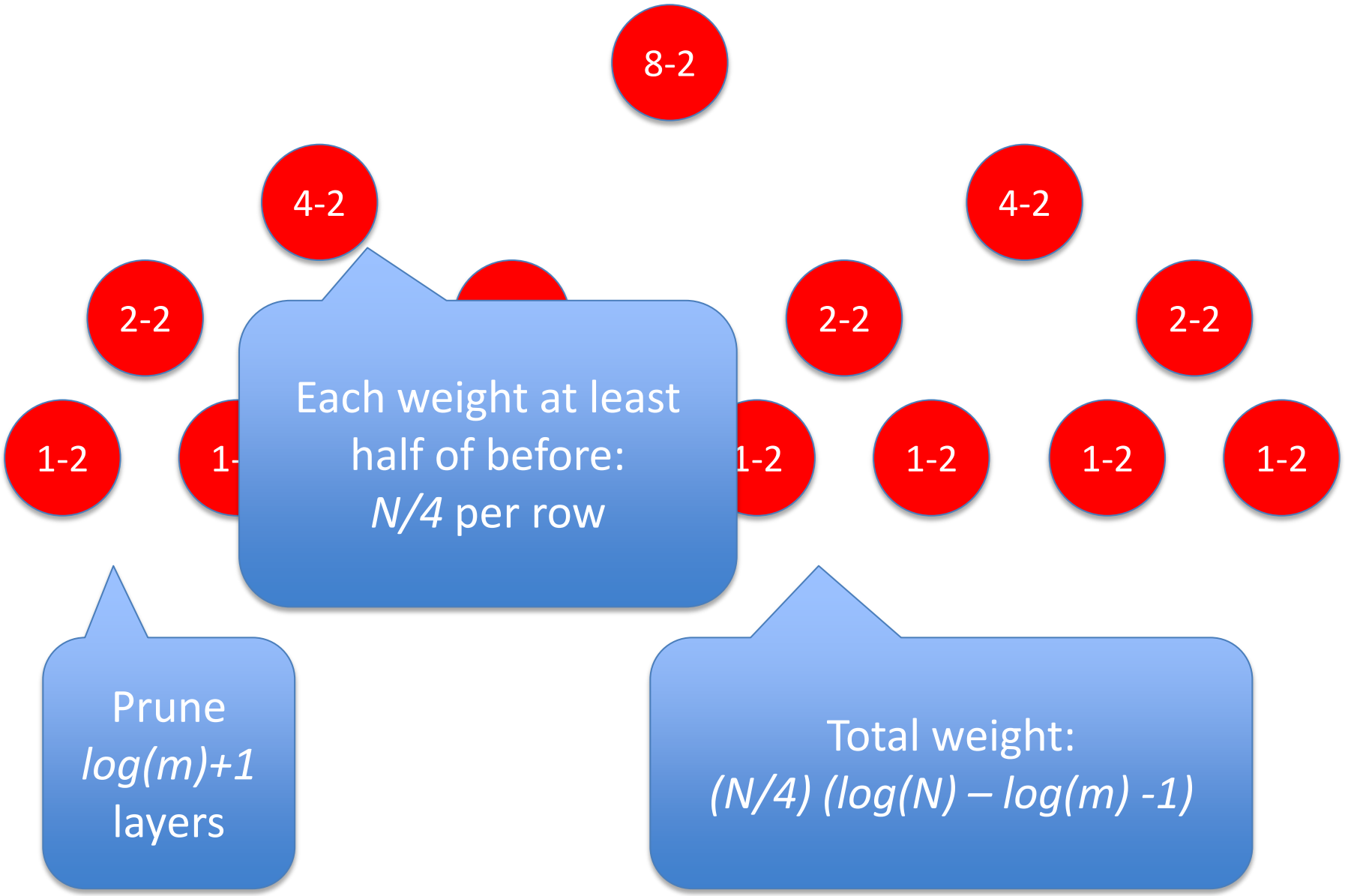
- Easy case:
  - No client memory
  - Perfect correctness
  - Perfect obliviousness
  - ~~r = w~~
- **Theorem**
  - Any ORAM simulating *N* accesses makes at least on average *M = (N/2) log(N)* ***(r/w)*** probes
  - Overhead = *M w / N r = log(N)*



Theorem

- Harder case:
  - Client memory: *m* words
  - Perfect correctness
  - Perfect obliviousness

Client memory: *m = 2*



Theorem

- Harder case:
  - Client memory: *m* words
  - Perfect correctness
  - Perfect obliviousness
- **Theorem**
  - Any ORAM simulating *N* accesses makes on average *(N/4) (log(N) – log(m) – 1)* probes
  - Overhead = *log(N/m)*

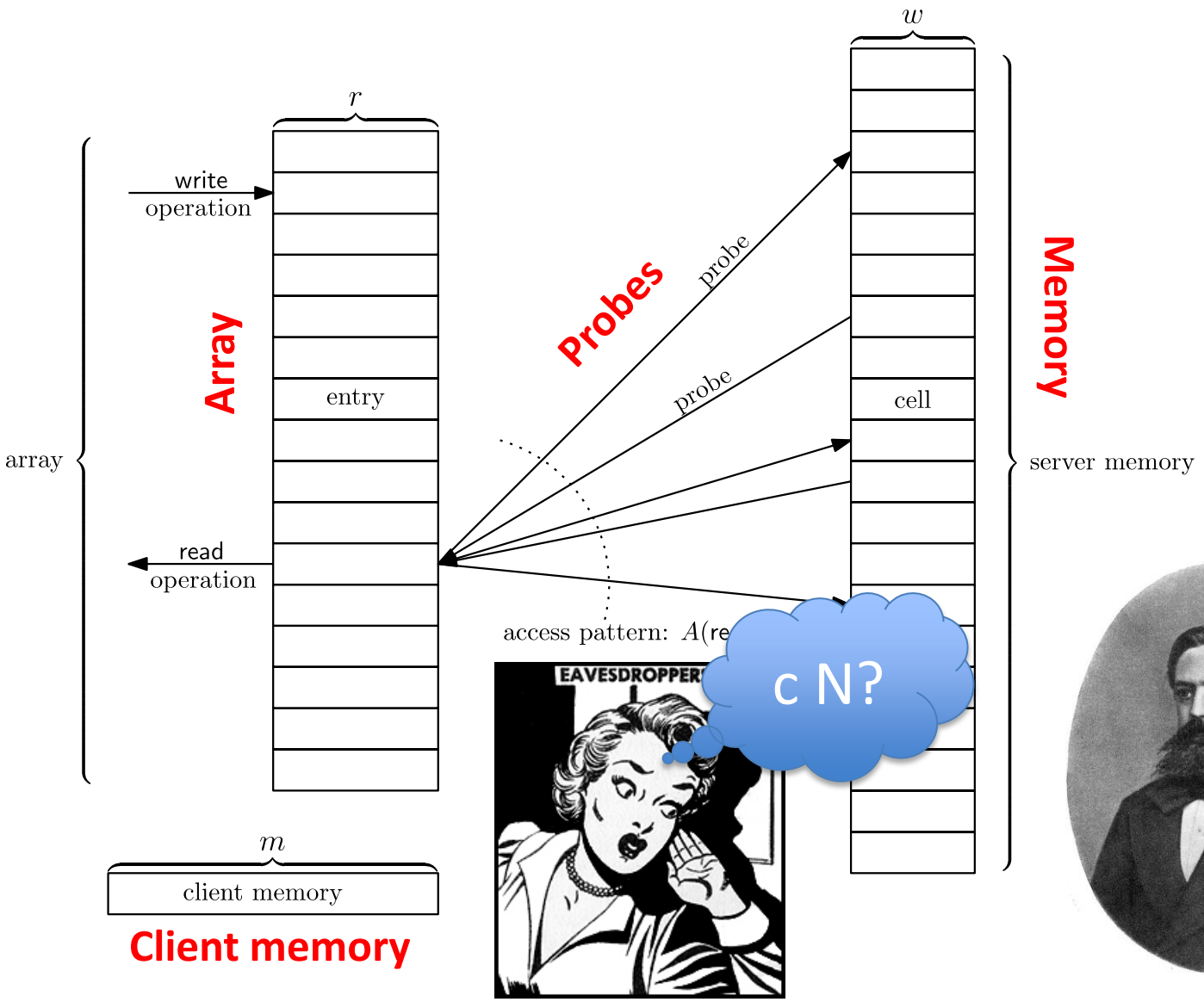
Theorem

- Even harder case:
  - Client memory: *m* words
  - Correctness: *c > 0* on each read
    - Word size *w = log(N)*
  - Obliviousness: *o > 0*

How many times must the read-sequence probe a cell which was last time probed during the write-sequence?

$w(1,r_1)$	$w(2,r_2)$	$w(3,r_3)$	$w(4,r_4)$	$w(5,r_5)$	$w(6,r_6)$	$w(7,r_7)$	$w(8,r_8)$	r(1)	r(2)	r(3)	r(4)	r(5)	r(6)	r(7)	r(8)

Obliviousness + Markov



Theorem

- Even harder case:
  - Client memory: *m* words
  - Correctness: *c > 0* on each read
    - Word size *w = log(N)*
  - Obliviousness: *o > 0*
- **Theorem**
  - Any ORAM simulating *N* accesses has overhead at least *log(N/m)*.

Future Work (1/2)

- There are other cell-probe lower-bound techniques out there
- There are more oblivious data structures out there
- Go prove some lower bounds

Future Work (2/2)

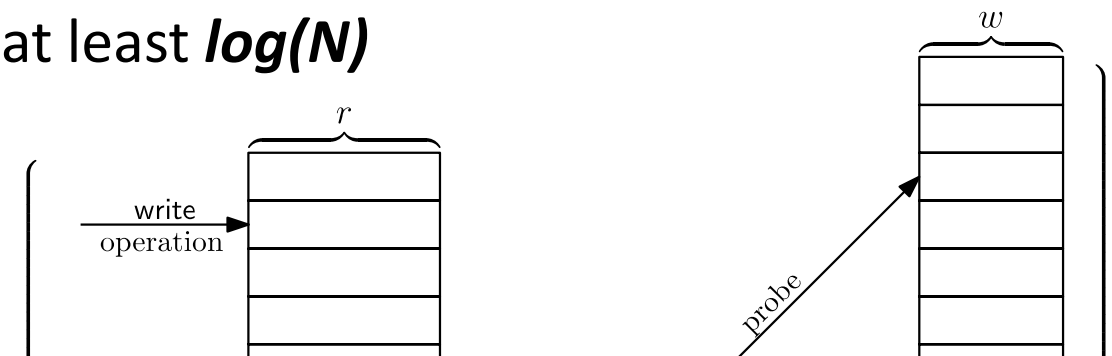
- PathORAM, 2013

  [Stefanov, van Dijk, Shi, Fletcher, Ren, Yu, Devadas, CCS'13]

  - Bandwidth overhead = *log(N)*
    - When *w = log(N)* and *r = w²*
  - Bandwidth overhead = ***log²(N)***
    - When *w = r = log(N)*
- PanORAMa, 2018

  [Patel, Persiano, Raykova, Yeo, FOCS'18]

  - Bandwidth overhead = ***log(N) log(log(N))***
- Today:
  - Overhead must be at least ***log(N)***
- *Close that gap!*



Conclusion

Yes, There is an Oblivious RAM Lower Bound!