

# Lower Bounds for Encrypted Multi-Maps and Searchable Encryption in the Leakage Cell Probe Model

Sarvar Patel\*, Giuseppe Persiano\*\* and [Kevin Yeo\\*](#)

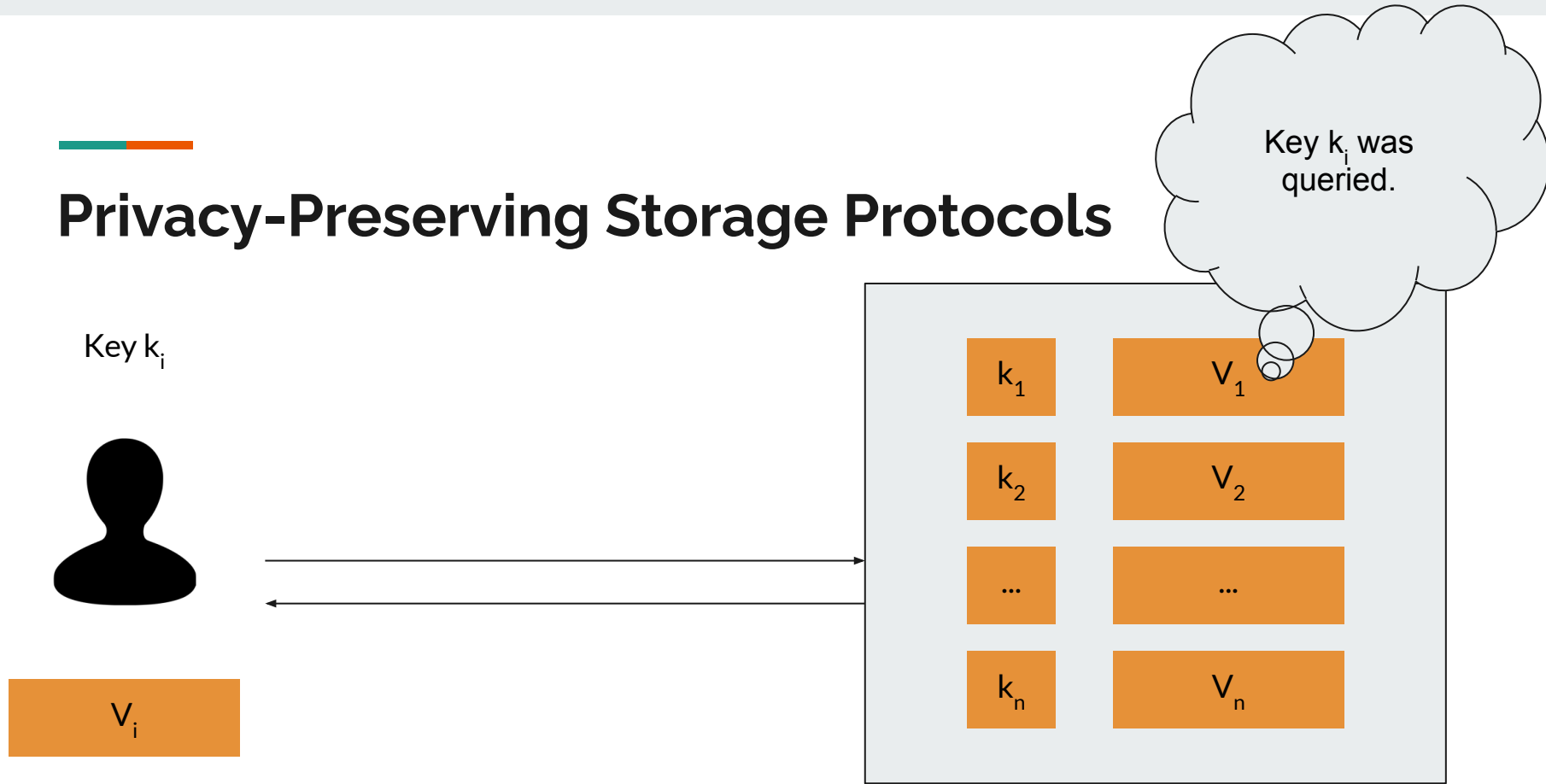
\*Google

\*\*University of Salerno





# Privacy-Preserving Storage Protocols



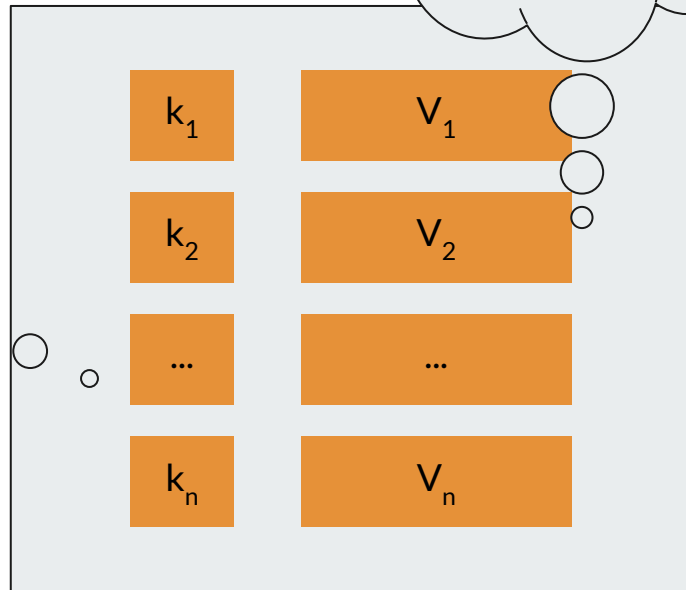


# Privacy-Preserving Storage Protocols

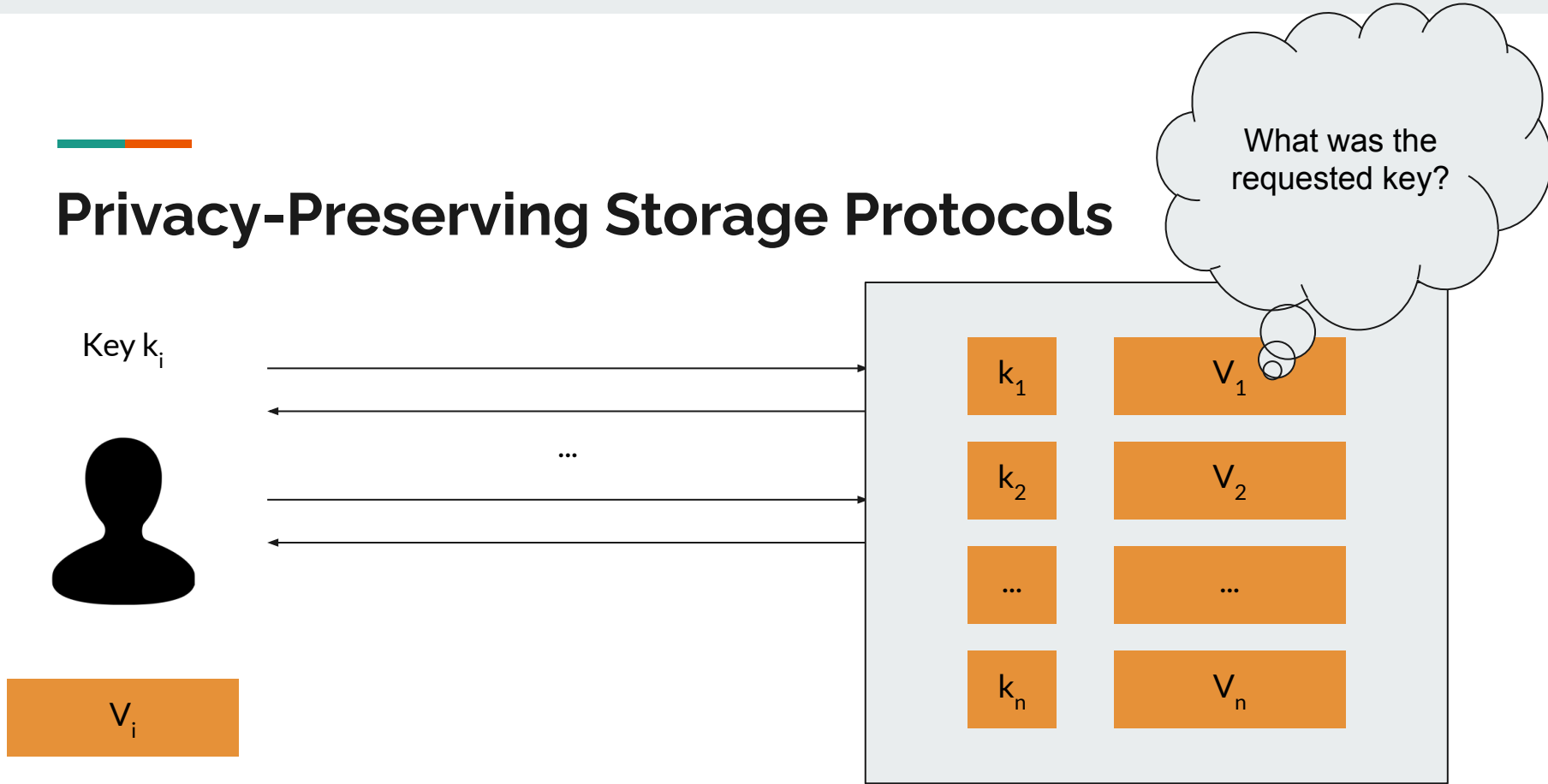


Key  $k_{15}$  was most frequently queried.

Key  $k_2$  was never queried.



# Privacy-Preserving Storage Protocols





# Privacy Spectrum for Maps

Plaintext Maps



# Plaintext Maps

- Classic dictionary problem with many solutions!
  - Perfect Hashing: Static [FKS'84], Dynamic [DKM+'94]
  - Cuckoo Hashing [PR'01]
  - ... and many more



# Plaintext Maps

- Classic dictionary problem with many solutions!
  - Perfect Hashing: Static [FKS'84], Dynamic [DKM+'94]
  - Cuckoo Hashing [PR'01]
  - ... and many more
- **Efficiency:**  $O(1)$  overhead,  $O(n)$  storage
- **Privacy:** None -- Leaks all keys and values.



# Privacy Spectrum for Maps

Plaintext Maps

Structured  
Encryption

Efficiency:  $O(1)$

Leakage: Everything





# Structured Encryption

- **Idea:** Encrypt a data structure while maintaining operations
  - Example: Searchable encryption = Encrypt a search index
- Many works in the past two decades:
  - Static [SWP'00], [BDOP'04], [CGKO'11], ...
  - Dynamic [CJJ+'14], [SPS'14], ...
  - Forward and Backward Privacy [Bost'16], [BMO'17], ...



# Structured Encryption

- **Idea:** Encrypt a data structure while maintaining operations
  - Example: Searchable encryption = Encrypt a search index
- Many works in the past two decades:
  - Static [SWP'00], [BDOP'04], [CGKO'11], ...
  - Dynamic [CJJ+'14], [SPS'14], ...
  - Forward and Backward Privacy [Bost'16], [BMO'17], ...
- **Efficiency:** Typically  $O(1)$  but can be higher depending on leakage
- **Privacy:** Some well-defined leakage function
  - Number of values associated with keys, Key-equality between operations, Number of operations, etc.



# Privacy Spectrum for Maps

Plaintext Maps

Efficiency:  $O(1)$

**Leakage:** Everything

Structured  
Encryption

Efficiency:  $O(1)$

**Leakage:** Non-trivial  
Leakage Function

Oblivious RAM



# Oblivious RAM

- Introduced by Goldreich and Ostrovsky [GO'96]
  - Also, many works in the past decade [PR'10], [SSS'11], [MMOT'12], [SvDS'13], [PPRY'18], ...
  - ... leading to optimal  $O(\log n)$  overhead construction [AKL+'20]



# Oblivious RAM

- Introduced by Goldreich and Ostrovsky [GO'96]
  - Also, many works in the past decade [PR'10], [SSS'11], [MMOT'12], [SvDS'13], [PPRY'18], ...
  - ... leading to optimal  $O(\log n)$  overhead construction [AKL+'20]
- **Efficiency:**  $O(\log n)$ , which is tight due to [GO'96, LN'18]
- **Privacy:** Adversary cannot distinguish two sequences of same length
  - Leakage function is (upper bound on) length of operational sequence



# Privacy Spectrum for Maps

Plaintext Maps

Efficiency:  $O(1)$

**Leakage:** Everything

Structured  
Encryption

Efficiency:  $O(1)$

**Leakage:** Non-trivial  
Leakage Function

Oblivious RAM

Efficiency:  $O(\log n)$

**Leakage:** Length of  
operational sequence



# Privacy Spectrum for Maps

Plaintext Maps

Efficiency:  $O(1)$

Leakage: Everything

Structured  
Encryption

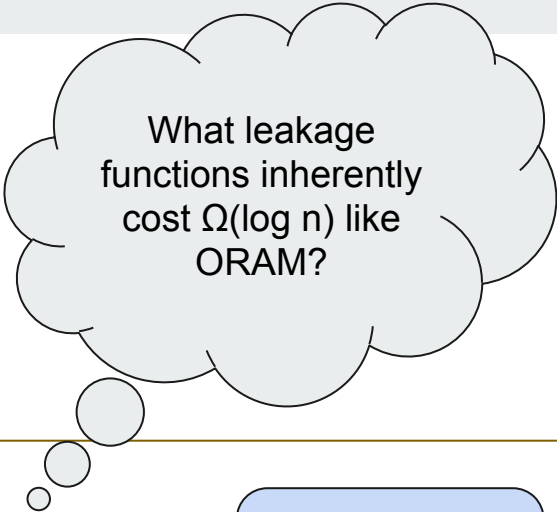
Efficiency:  $O(1)$

**Leakage:** Non-trivial  
Leakage Function

Oblivious RAM

Efficiency:  $O(\log n)$

**Leakage:** Length of  
operational sequence



What leakage  
functions inherently  
cost  $\Omega(\log n)$  like  
ORAM?



# Privacy Spectrum for Maps

Plaintext Maps

Efficiency:  $O(1)$

Leakage: Everything

Structured  
Encryption

Efficiency:  $O(1)$

**Leakage:** Non-trivial  
Leakage Function

Oblivious RAM

Efficiency:  $O(\log n)$

**Leakage:** Length of  
operational sequence





# Hash-and-Encrypt Compiler

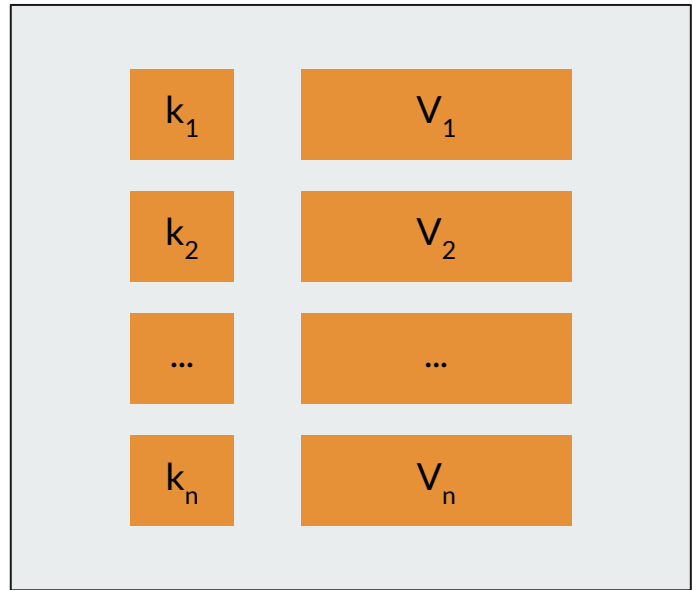
- Consider any plaintext map with operations:
  - Insert( $k, v$ )
  - Get( $k$ )
  - Delete( $k$ )



# Hash-and-Encrypt Compiler



K

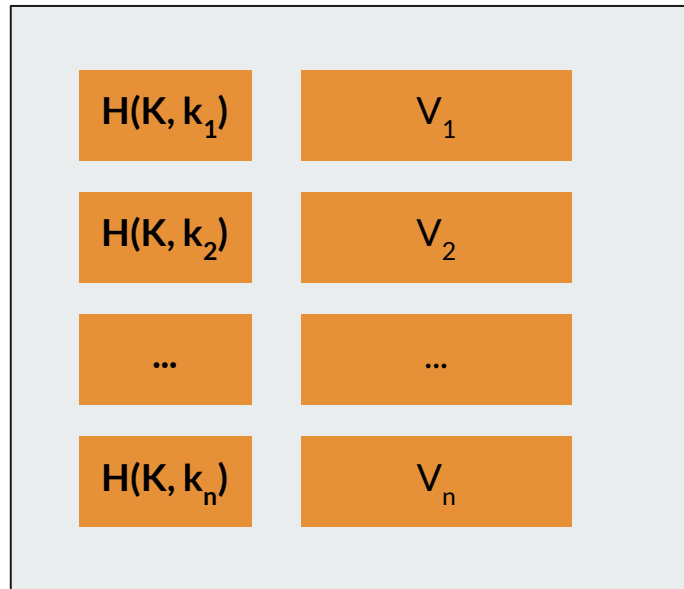




# Hash-and-Encrypt Compiler



K

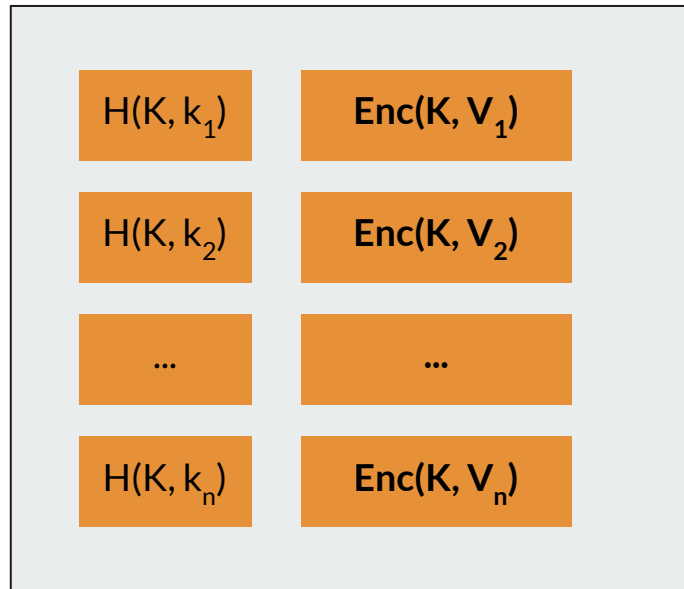




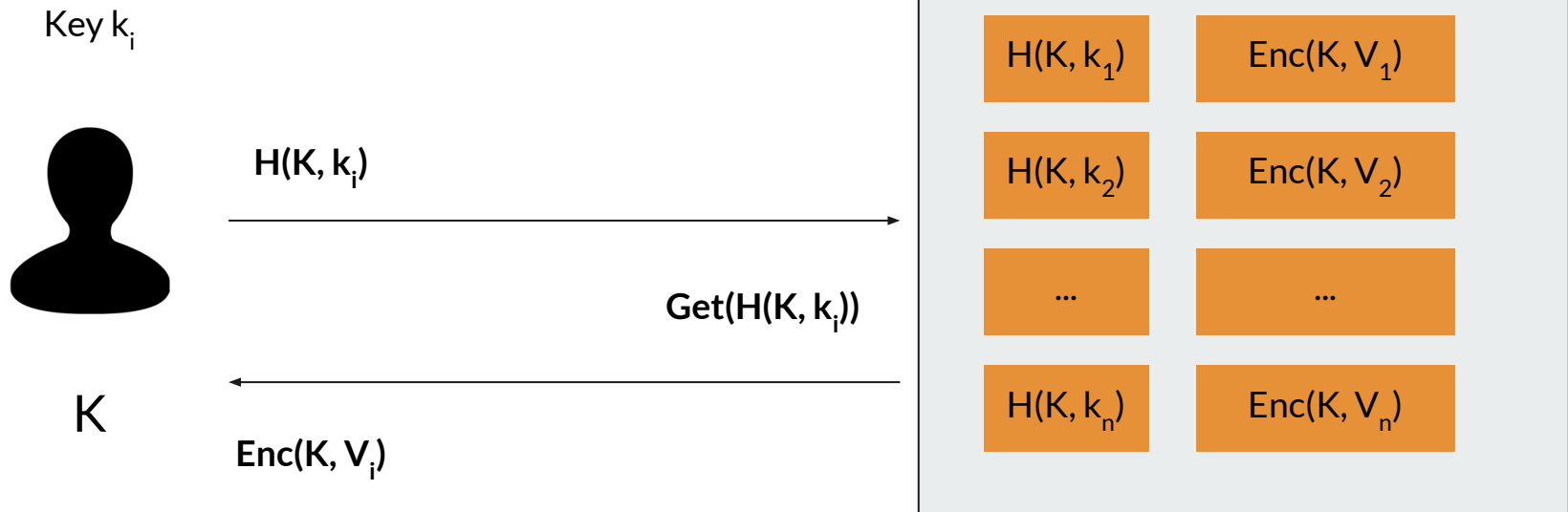
# Hash-and-Encrypt Compiler



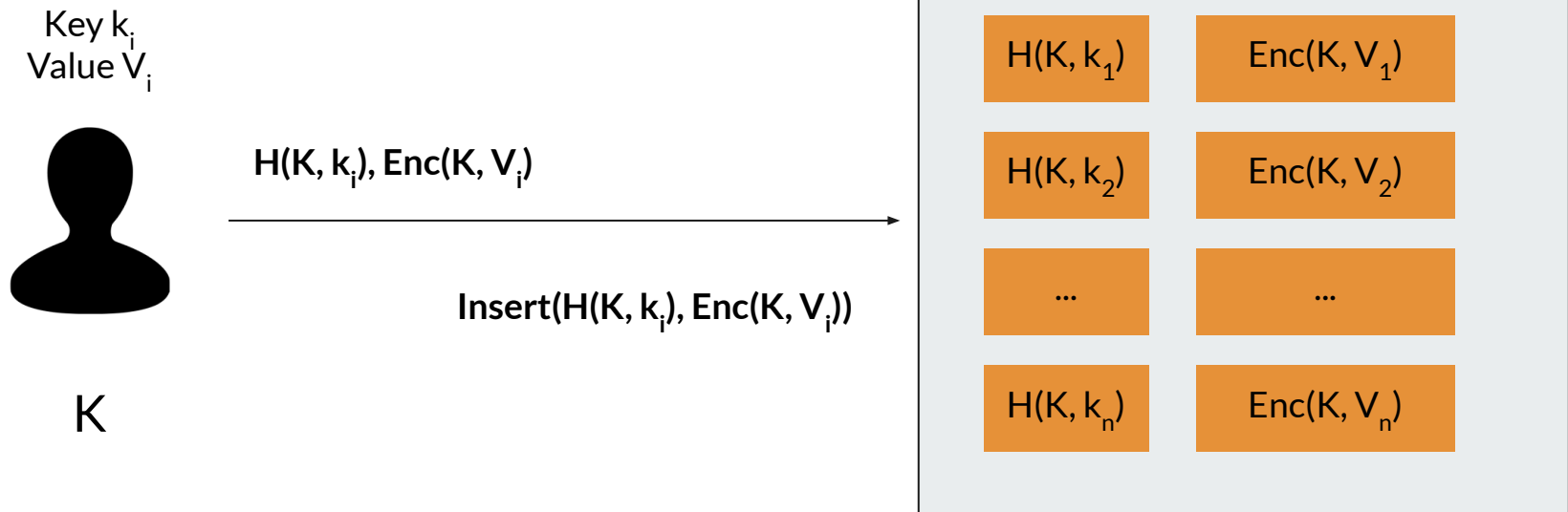
K



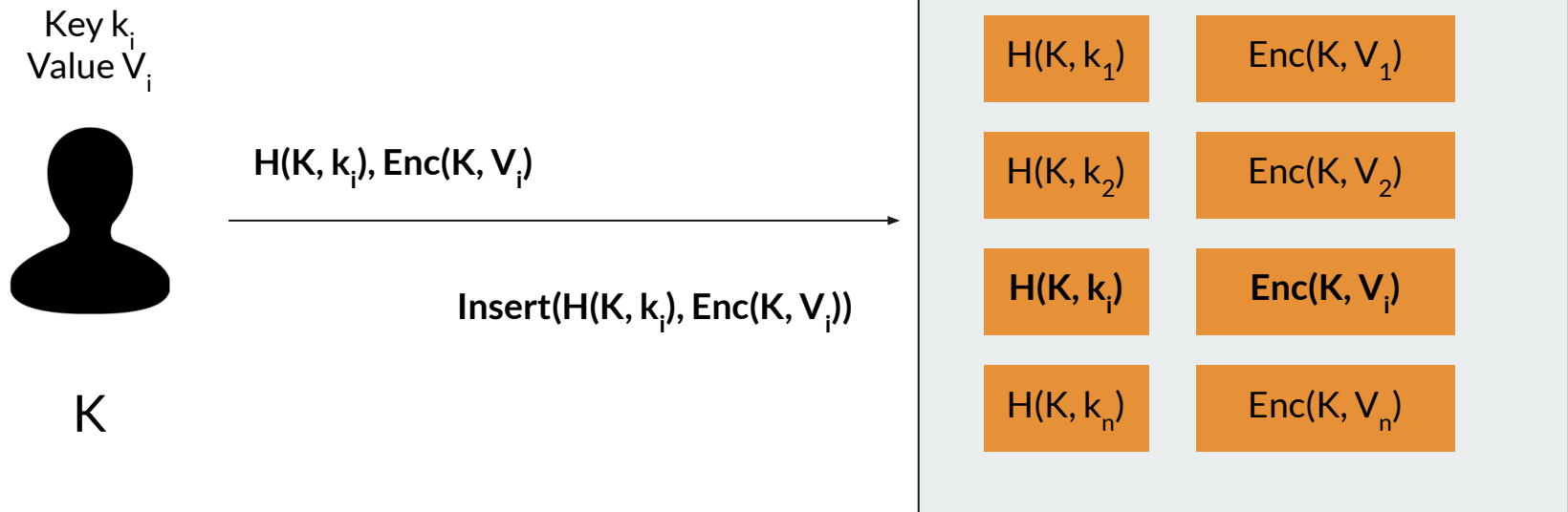
# Hash-and-Encrypt Compiler (Query)



# Hash-and-Encrypt Compiler (Insert)



# Hash-and-Encrypt Compiler (Insert)





# Leakage of Hash-and-Encrypt

Insert

$H(K, \text{"cat"})$

$\text{Enc}(K, \text{"01"})$





# Leakage of Hash-and-Encrypt

Insert

$H(K, \text{"cat"})$

$\text{Enc}(K, \text{"01"})$

Insert

$H(K, \text{"dog"})$

$\text{Enc}(K, \text{"00"})$



# Leakage of Hash-and-Encrypt

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Leakage of Hash-and-Encrypt

- Type of operation performed



# Leakage of Hash-and-Encrypt

**Insert**

$H(K, \text{"cat"})$

$\text{Enc}(K, \text{"01"})$

**Insert**

$H(K, \text{"dog"})$

$\text{Enc}(K, \text{"00"})$

**Query**

$H(K, \text{"dog"})$

$\text{Enc}(K, \text{"00"})$

**Insert**

$H(K, \text{"cat"})$

$\text{Enc}(K, \text{"11"})$

**Query**

$H(K, \text{"cat"})$

$\text{Enc}(K, \text{"01"})$

$\text{Enc}(K, \text{"11"})$

...



# Leakage of Hash-and-Encrypt

- Type of operation performed
- Length of Query response



# Leakage of Hash-and-Encrypt

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Leakage of Hash-and-Encrypt

- Type of operation performed
- Length of Query response
- Key-Equality Pattern



# Leakage of Hash-and-Encrypt

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...





# Leakage of Hash-and-Encrypt

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Leakage of Hash-and-Encrypt

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Leakage of Hash-and-Encrypt

- Type of operation performed
- Length of Query response
- Key-Equality Pattern



# Leakage of Hash-and-Encrypt

- Type of operation performed
- Length of Query response
- Key-Equality Pattern

**Surprisingly, this matches leakage of best STE  $O(1)$  schemes!!!**



# Privacy Spectrum for Maps

Plaintext Maps

Efficiency:  $O(1)$

Leakage: Everything

Structured  
Encryption

Efficiency:  $O(1)$

**Leakage:** Non-trivial  
Leakage Function

Oblivious RAM

Efficiency:  $O(\log n)$

**Leakage:** Length of  
operational sequence



# Can we do better?

- Type of operation performed
- Length of Query response
- Key-Equality Pattern



# Can we do better?

- ~~Type of operation performed~~ (Perform all possible operation types)
- Length of Query response
- Key-Equality Pattern



# Can we do better?

- ~~Type of operation performed~~ (Perform all possible operation types)
- Length of Query response??? (Hard to do without increasing cost significantly)
  - Padding Volume-Hiding STE schemes: [KM'19], [PPYY'19]
- Key-Equality Pattern





# Can we do better?

- ~~Type of operation performed~~ (Perform all possible operation types)
- Length of Query response??? (Hard to do without increasing cost significantly)
  - Padding Volume-Hiding STE schemes: [KM'19], [PPYY'19]
- Key-Equality Pattern



# Decoupled Key-Equality

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Decoupled Key-Equality

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Decoupled Key-Equality

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Decoupled Key-Equality

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$

Insert  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Query  
 $H(K, \text{"dog"})$   
 $\text{Enc}(K, \text{"00"})$

Insert  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"11"})$

Query  
 $H(K, \text{"cat"})$   
 $\text{Enc}(K, \text{"01"})$   
 $\text{Enc}(K, \text{"11"})$

...



# Main Result

**Theorem.** Any encrypted multi-map with leakage at most the decoupled key-equality pattern must have  $\Omega(\log n)$  overhead.



# Main Result

**Theorem.** Any encrypted multi-map with leakage at most the decoupled key-equality pattern must have  $\Omega(\log n)$  overhead.

**Corollary.** This lower bound is tight as there exists  $O(\log n)$  ORAM-based encrypted multi-maps leaking much less than the decoupled key-equality pattern.



# Privacy Spectrum for Maps

Plaintext Maps

Efficiency:  $O(1)$

Leakage: Everything

Structured  
Encryption

Efficiency:  $O(1)$

**Leakage:** Non-trivial  
Leakage Function



Oblivious RAM

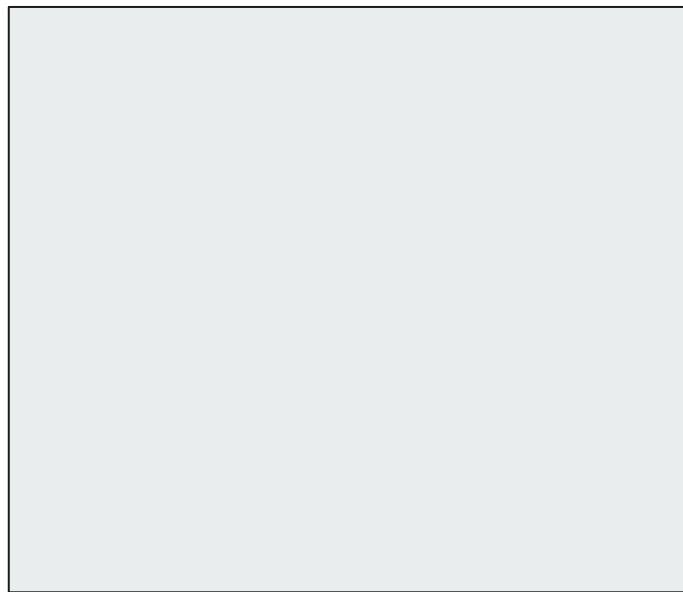
Efficiency:  $O(\log n)$

**Leakage:** Length of  
operational sequence



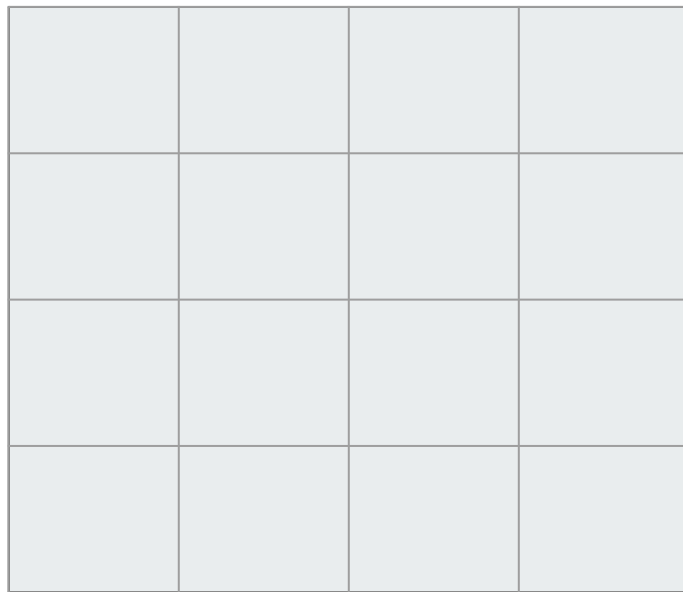


# Cell Probe Model



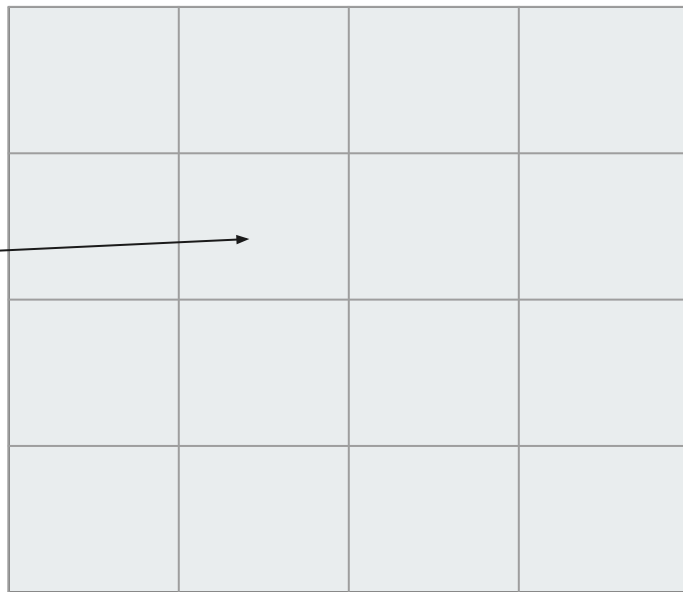


# Cell Probe Model





# Cell Probe Model





# Cell Probe Model

- Only cost is probing (read/write) a cell of  $w$  bits
- Computation is free
- Random oracle is free
- Accessing client storage is free
- Very weak cost model  $\rightarrow$  Very strong lower bounds



# Lower Bound

- Uses Information Transfer technique [PD'06]

## LOGARITHMIC LOWER BOUNDS IN THE CELL-PROBE MODEL\*

MIHAI PĂTRAȘCU<sup>†</sup> AND ERIK D. DEMAINÉ<sup>‡</sup>

**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.

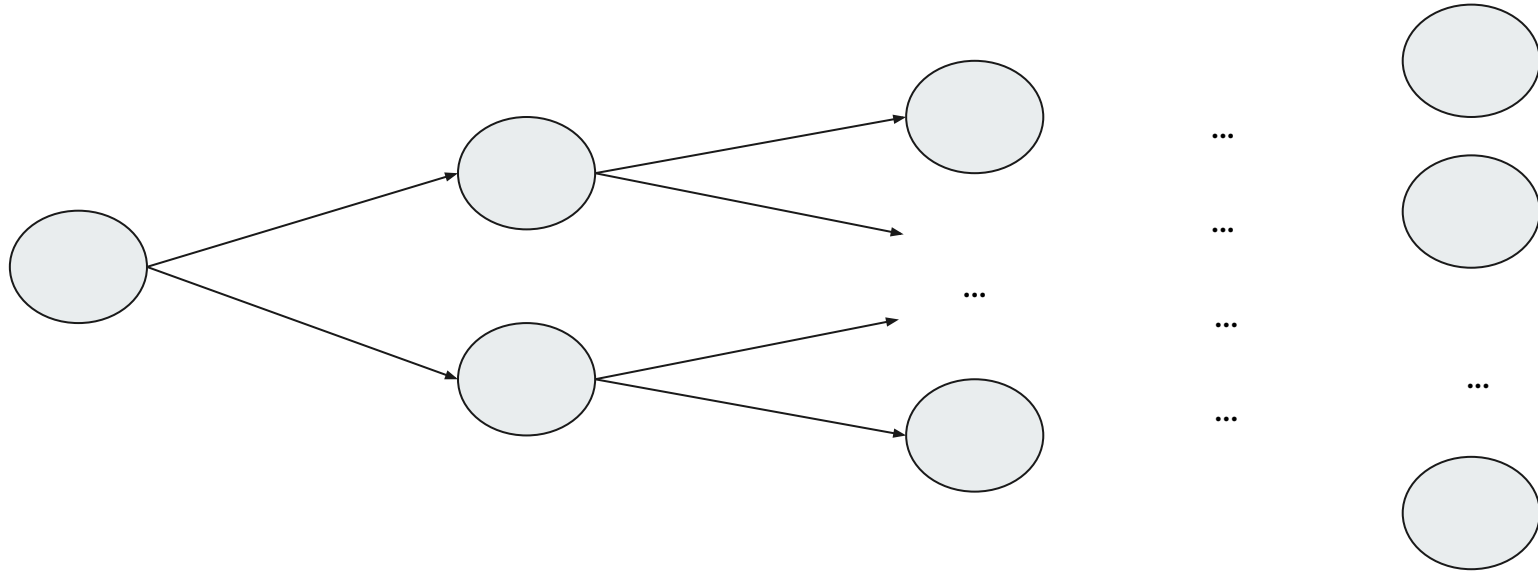
**Key words.** Cell-probe complexity, lower bounds, data structures, dynamic graph problems, partial-sums problem

**AMS subject classification.** 68Q17

**1. Introduction.** The cell-probe model is perhaps the strongest model of computation for data structures, subsuming in particular the common word-RAM model.

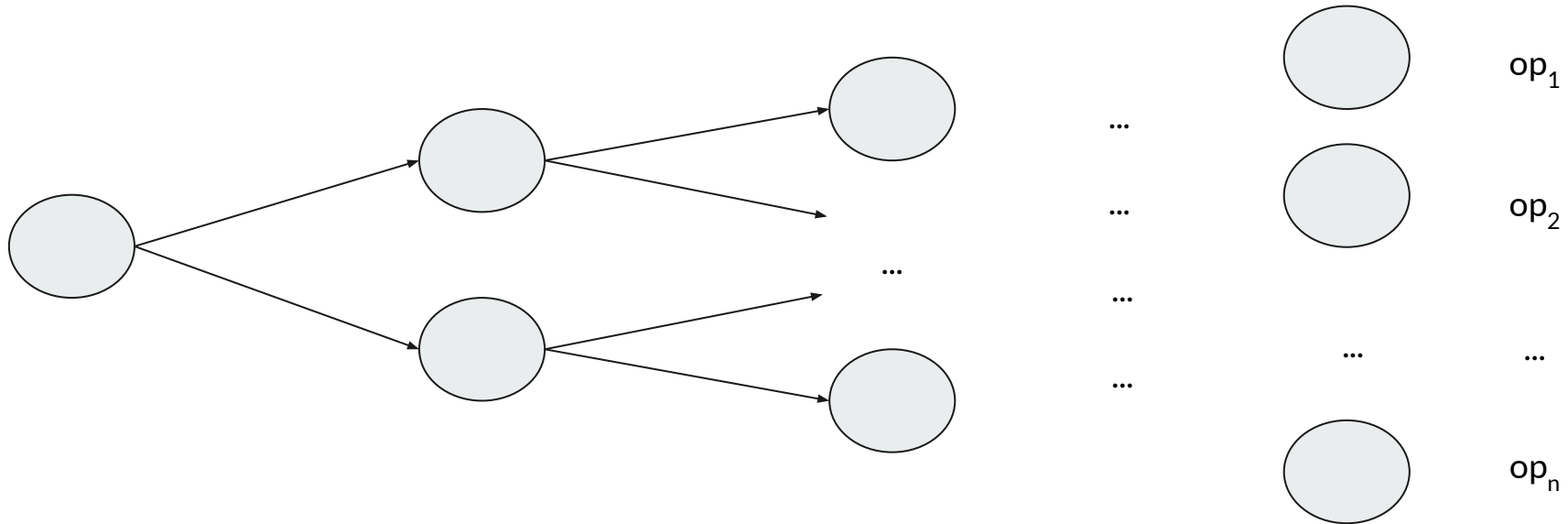


# Lower Bound



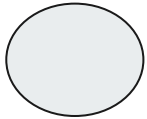


# Lower Bound

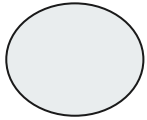




# Lower Bound



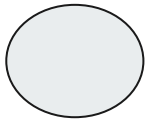
$op_1$



$op_2$

...

...

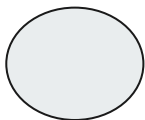


$op_n$

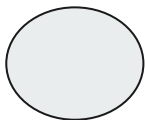




## Lower Bound



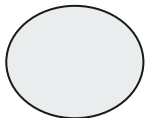
$op_1 \rightarrow \text{cread}(15), \text{cwrite}(72), \text{cwrite}(220), \dots$



$op_2 \rightarrow \text{cwrite}(650), \text{cwrite}(327), \text{cread}(296), \dots$

...

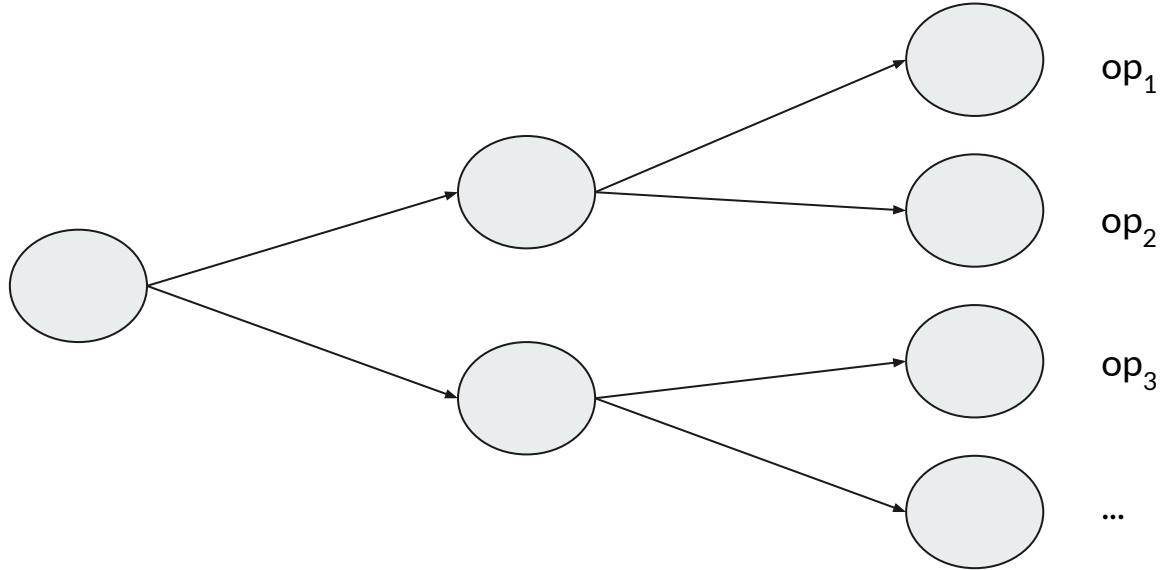
...



$op_n \rightarrow \text{cwrite}(297), \text{cread}(372), \text{cread}(580), \dots$

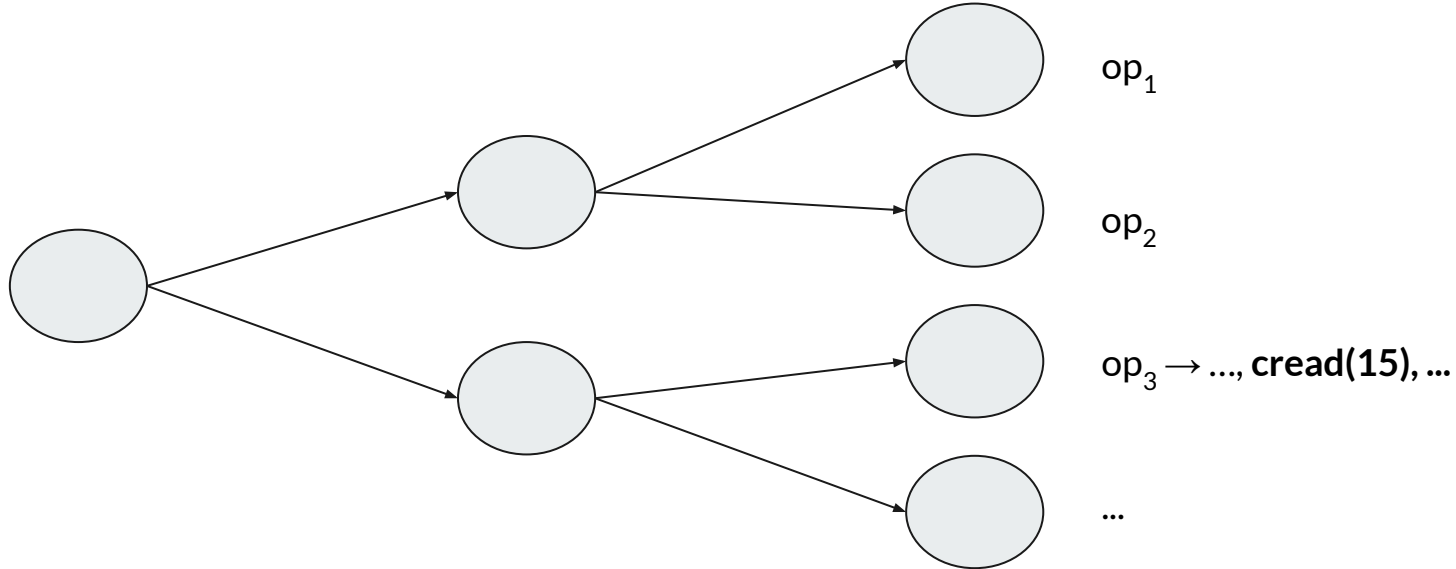


## Lower Bound

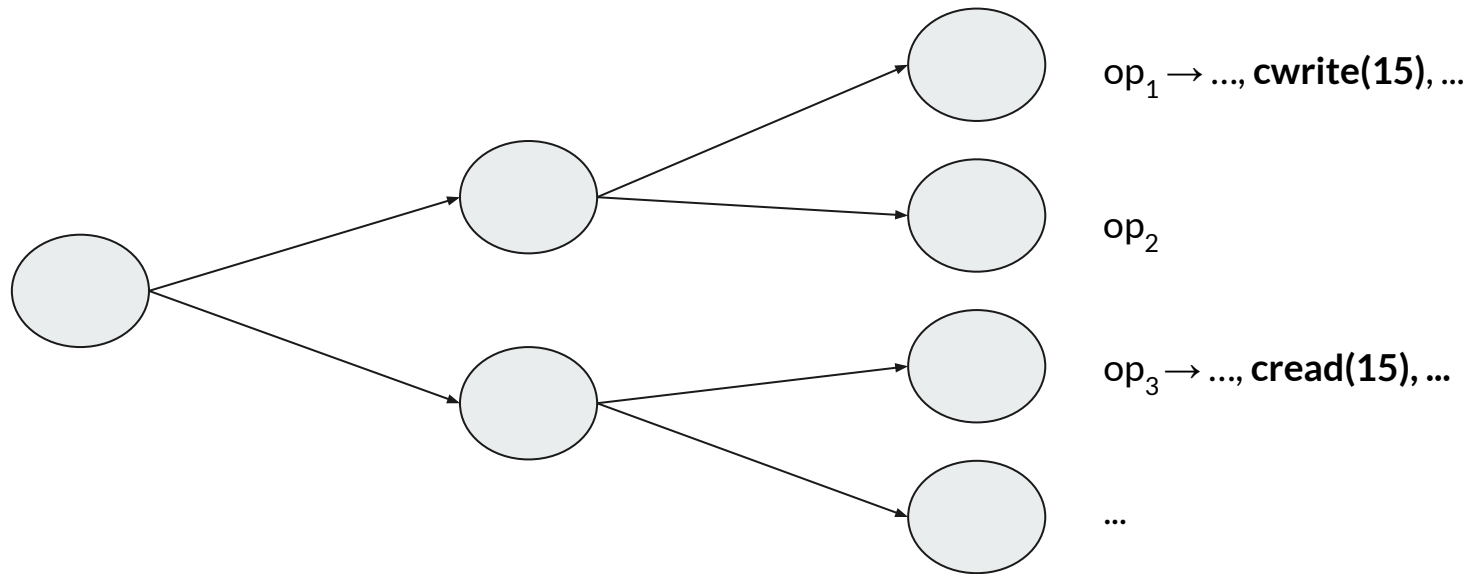




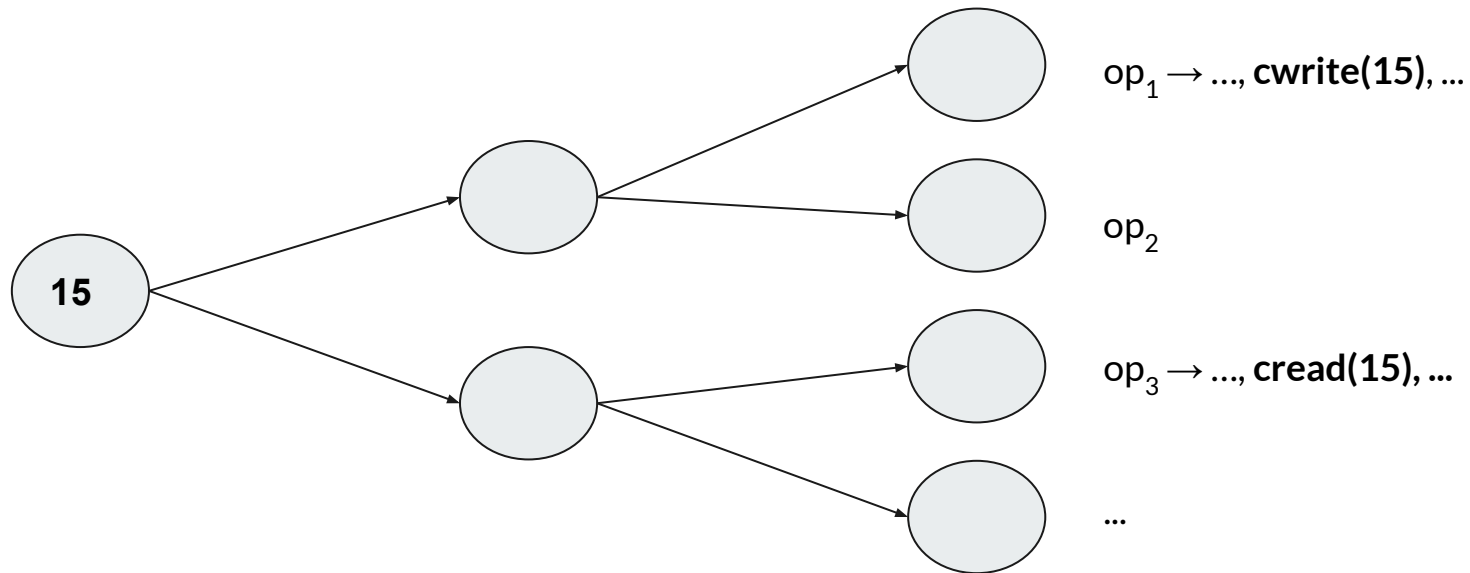
# Lower Bound



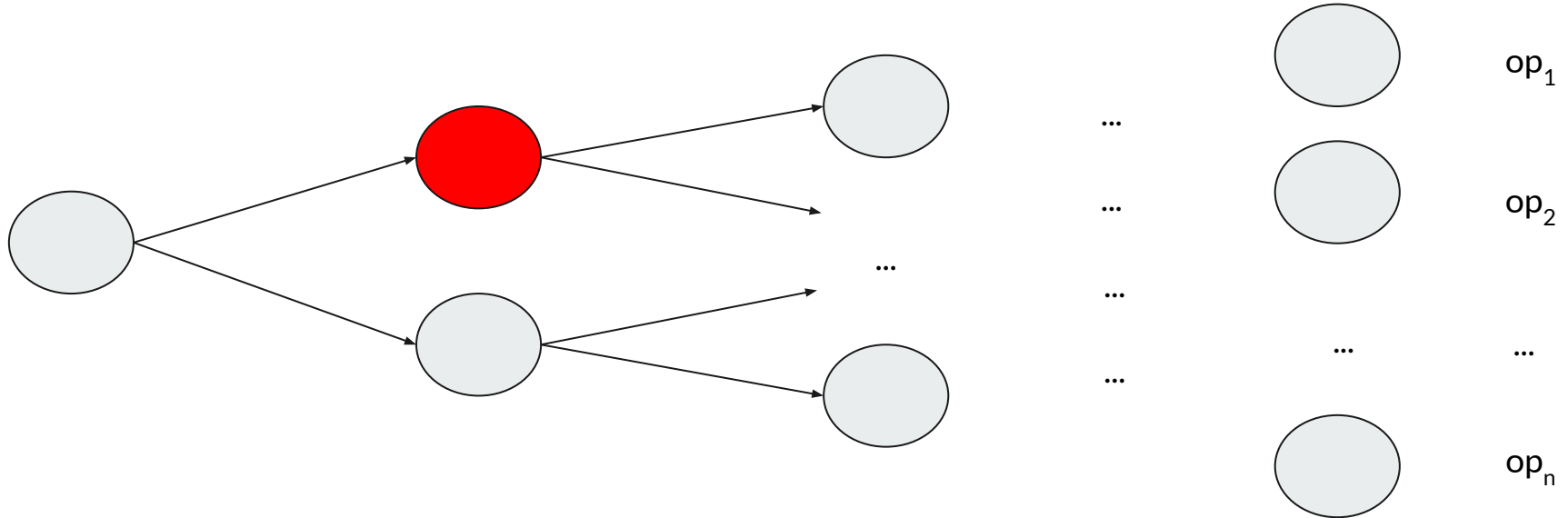
# Lower Bound



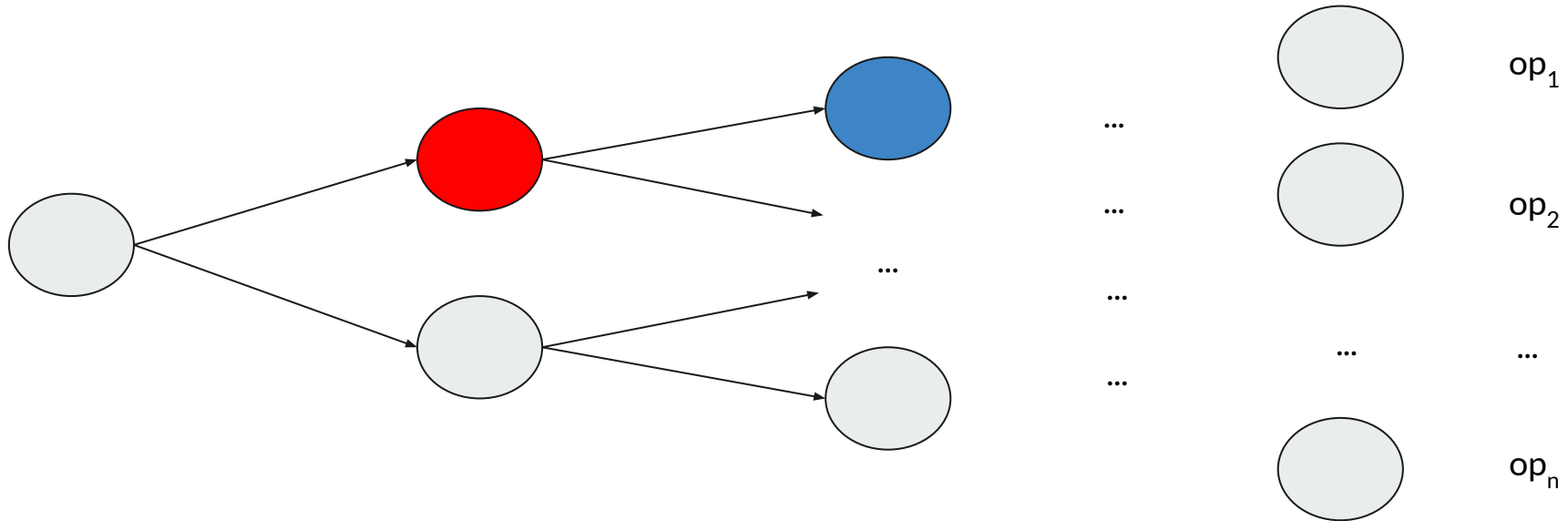
# Lower Bound



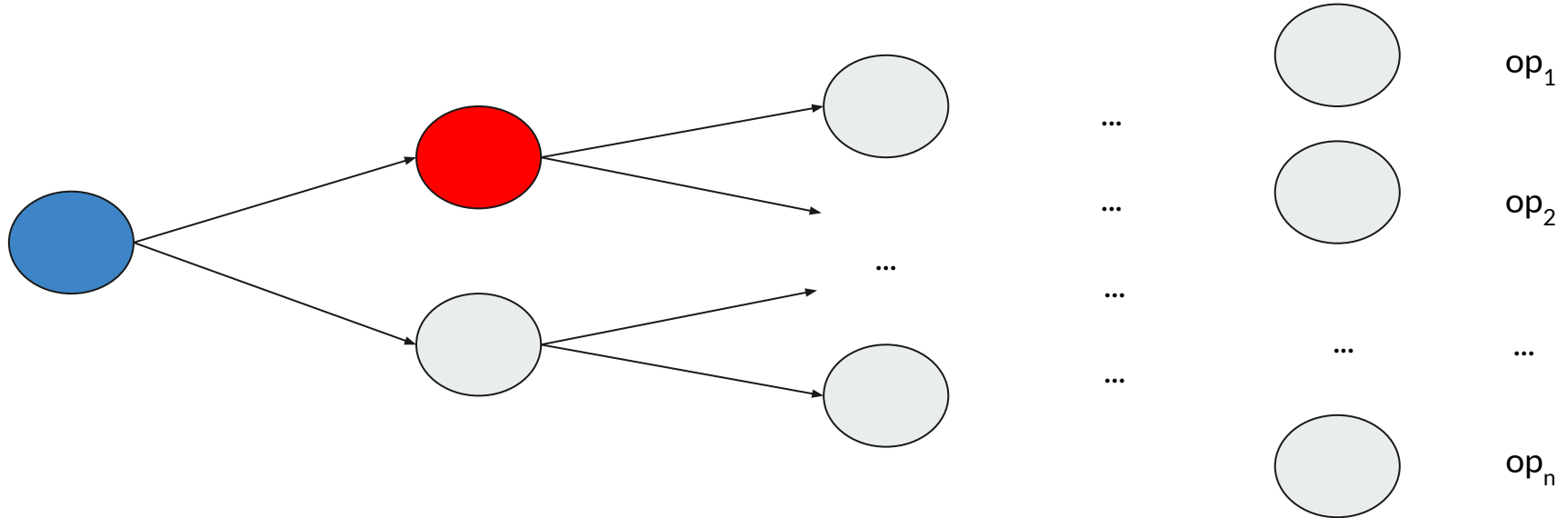
# Lower Bound



# Lower Bound



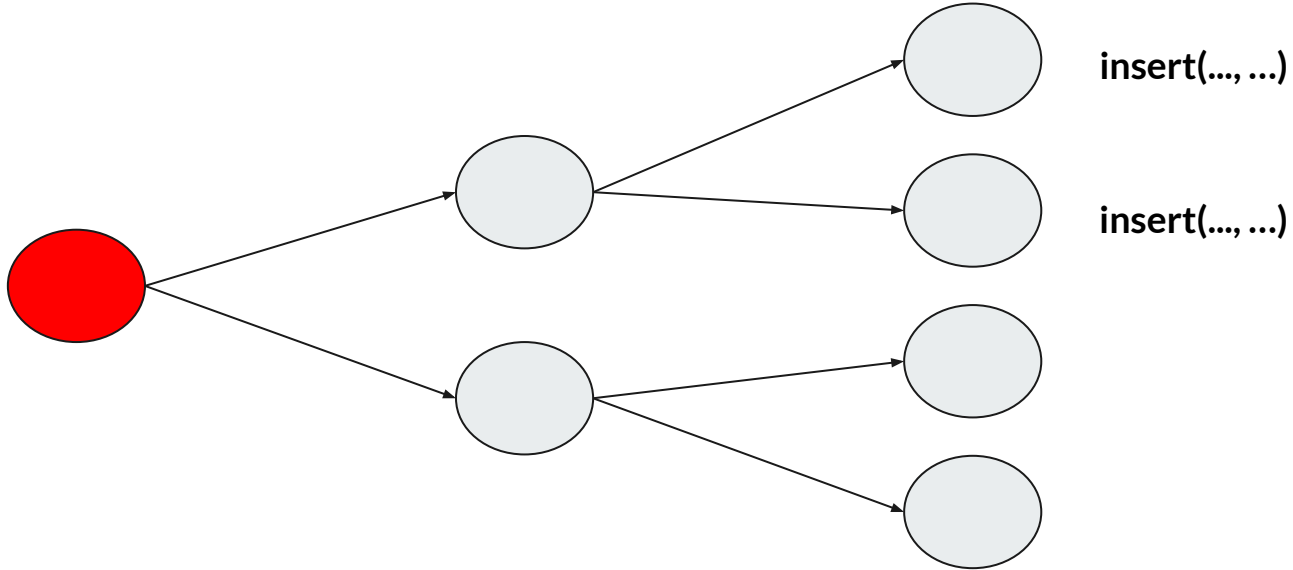
# Lower Bound



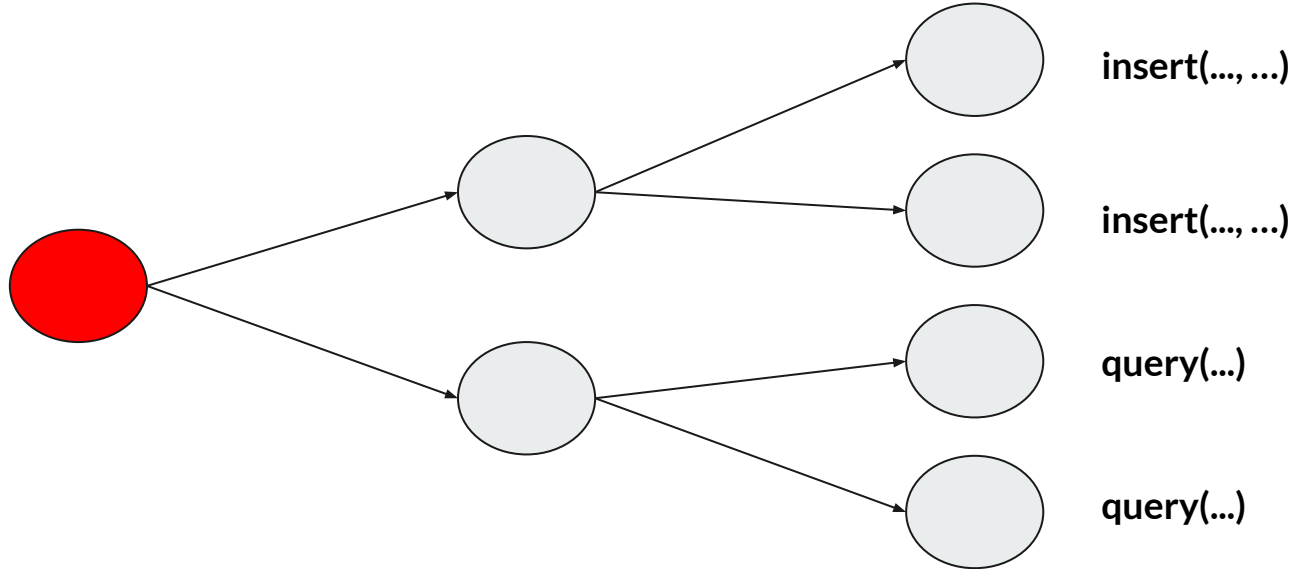




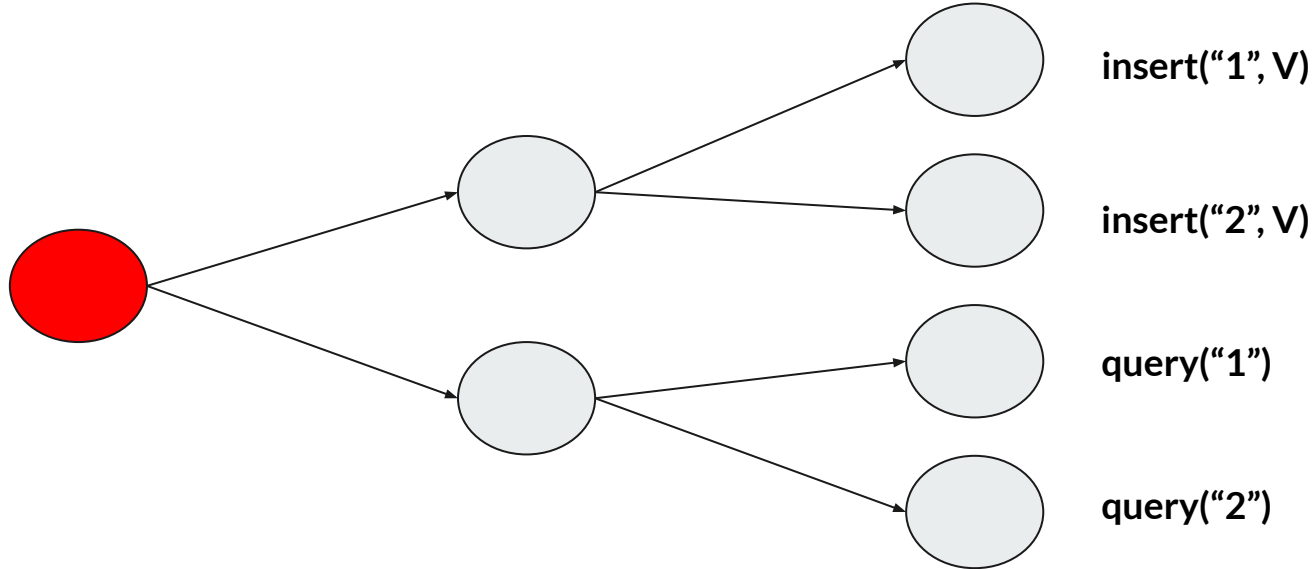
## Lower Bound



# Lower Bound



# Lower Bound





# Lower Bound

- Hard Sequence: insert("1", V), read("1"), insert("2", V), read("2"), insert("3", V), read("3"), ...
  - V contains a large amount of entropy



# Lower Bound

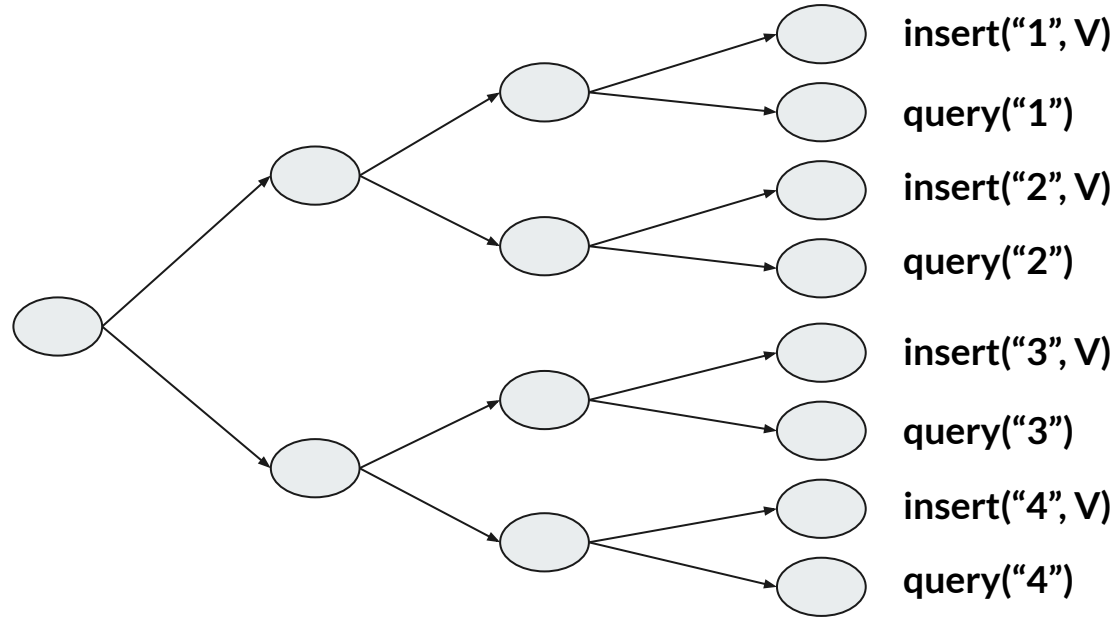
- Hard Sequence: insert("1", V), read("1"), insert("2", V), read("2"), insert("3", V), read("3"), ...
  - V contains a large amount of entropy
- Isn't this operation easy to handle?



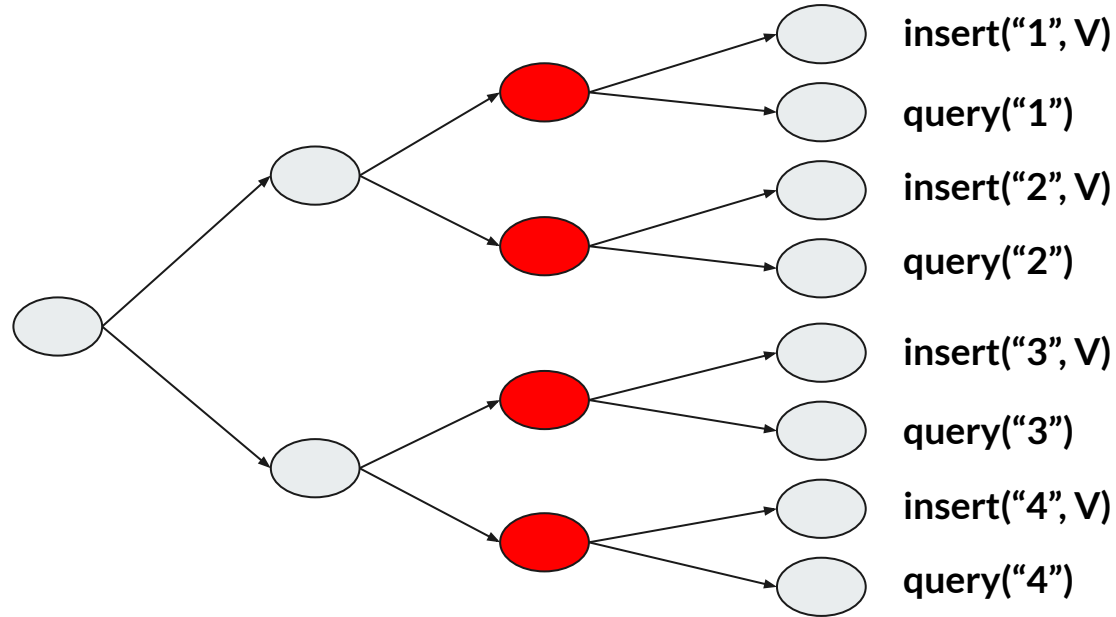
# Lower Bound

- Hard Sequence: insert("1", V), read("1"), insert("2", V), read("2"), insert("3", V), read("3"), ...
  - V contains a large amount of entropy
- Isn't this operation easy to handle?
- Key: Sequence must be indistinguishable from other sequences with identical leakage

# Lower Bound

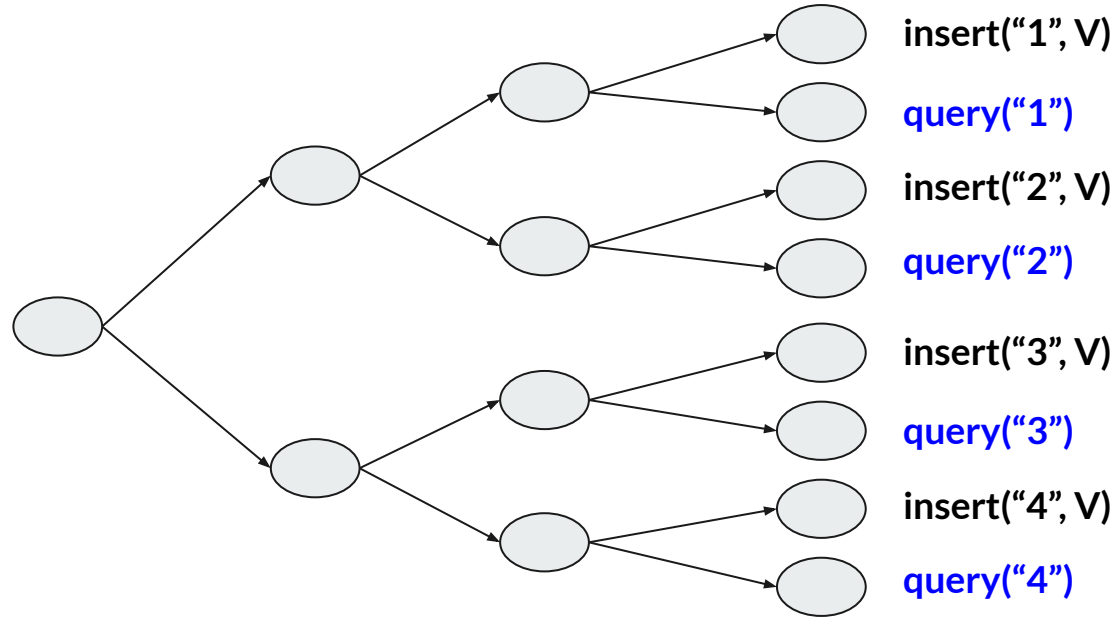


# Lower Bound

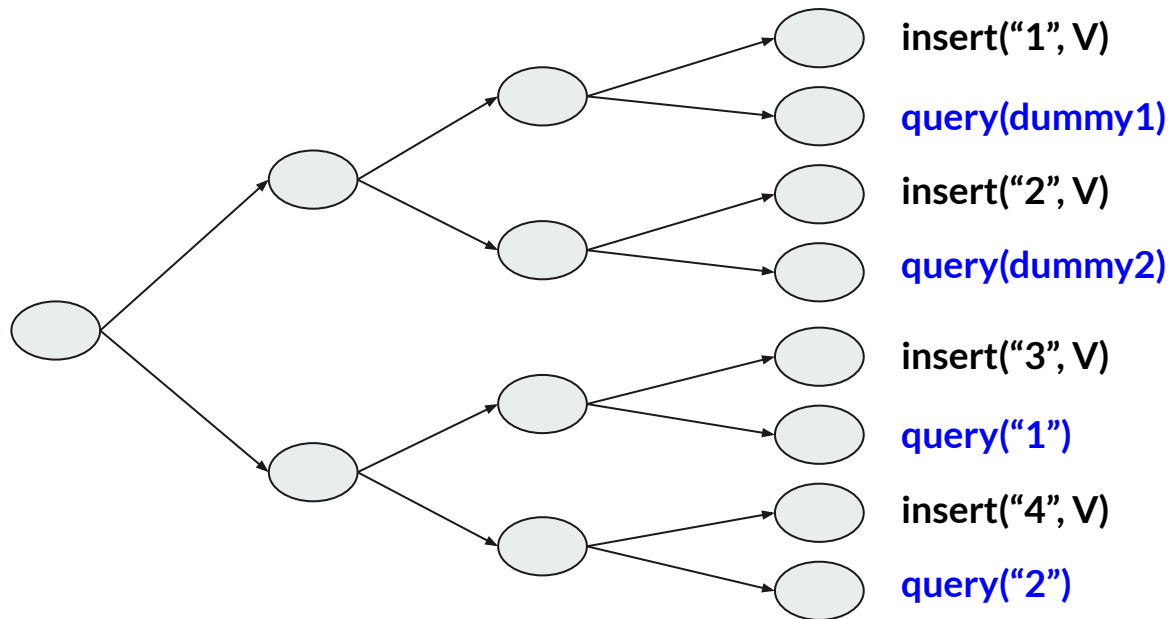




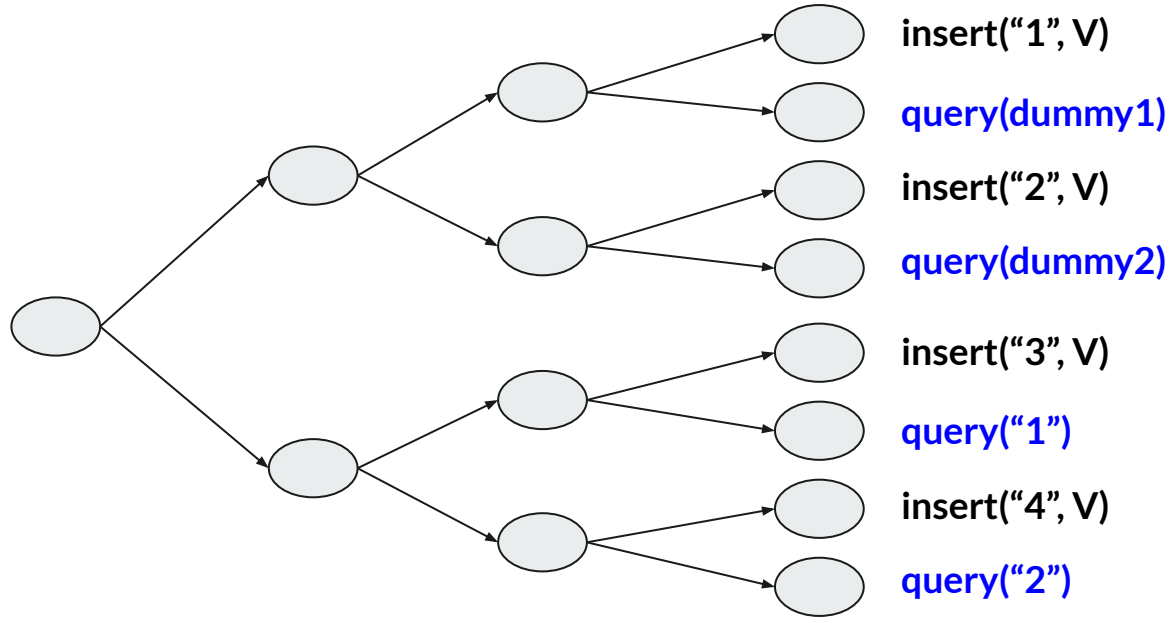
# Lower Bound



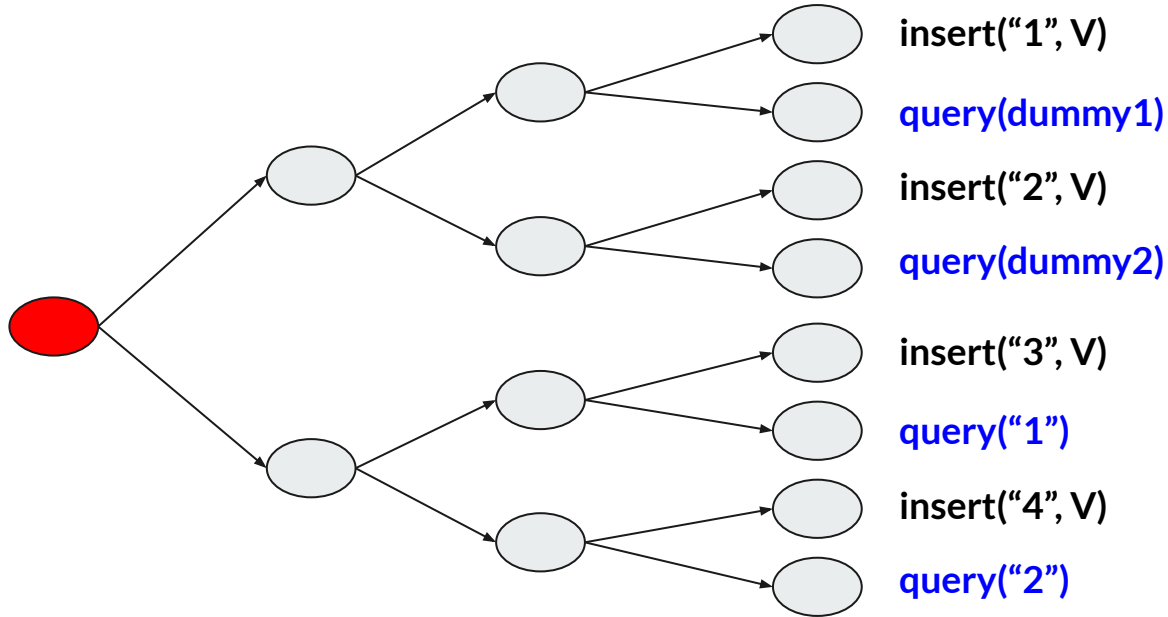
# Lower Bound



# Lower Bound



# Lower Bound



insert(dummy1, V)

insert(dummy2, V)

...



# Lower Bound

- Use these ideas to show that many probes must be assigned to half the internal nodes for this “easy” hard distribution.
- Summing up the probes assigned over all nodes provides the lower bound



# Stronger Lower Bounds

- The lower bounds hold even when one of:
  - Insert operations are performed in plaintext
  - Query operations are performed in plaintext



# Dynamic Searchable Encryption

**Theorem.** Dynamic searchable encryption schemes that are *response-hiding* require overhead  $\Omega(\log n)$  overhead.

**Corollary.** This lower bound is tight as there exist ORAM-based dynamic searchable encryption schemes that are response-hiding with  $O(\log n)$  overhead.



## Other Cryptographic Cell Probe Lower Bounds

- $\Omega(\log n)$  Oblivious RAMs [LN'18]
- $\Omega(\log n)$  Oblivious Data Structures [JLN'19]
- $\Omega(\log n)$  Differentially Private RAMs [PY'19]
- $\Omega(\log^2 n)$  Oblivious Near-Neighbor Search [LMWY'19]
- $\Omega(\log n)$  Multi-Server Oblivious RAMs [LSY'19]





**Thank you!**