Leakage in the Cell Probe Model Lower Bounds for Response Hiding Encrypted Multi-Maps

Giuseppe Persiano

Università di Salerno

June, 2019

Describing joint work with:
Sarvar Patel and Kevin Yeo (Google LLC)

The Model

Cell Probe Model for a Data Structure [Yao]

- Memory is a sequence of *cells* each of w bits
- Accessing (reading/writing) a cell cost 1
- All computation is for free

Classical model used to derive lower bounds for Data Structures

The Oblivious Model

Oblivious Cell Probe Model [Larsen+Nielsen '18]

In a Client-Server setting

- Client outsources storage of the DS to an *honest-but-curios* server
- Client performs DS operations $O = (\mathrm{op}_1, \ldots, \mathrm{op}_l)$ by accessing the Server memory
  - client can read and write any *cell* in Server memory
  - each cell is w-bit wide
- Client has limited private local memory
- Server observes the access pattern and the data downloaded
  - $\mathrm{view}^{\mathrm{DS}}(O) = \left(\mathrm{view}^{\mathrm{DS}}(\mathrm{op}_1), \ldots, \mathrm{view}^{\mathrm{DS}}(\mathrm{op}_l)\right)$
- **Passive** server: performs no computation
- Operations are performed **online**

Security Notion

Definition

DS is Oblivious, if for every PPT machine $\mathcal{A}$ and any two sequences O and O' of the same length

$$\left|\text{Prob}\left[\mathcal{A}(\text{view}^{\text{DS}}(O)) = 1\right] - \text{Prob}\left[\mathcal{A}(\text{view}^{\text{DS}}(O')) = 1\right]\right| \leq \frac{1}{4}.$$

The array maintenance problem (a.k.a. ORAM)

Two operations to maintain an n-slot array A

- $\text{Read}(i)$ returns the current value stored in $A[i]$
- $\text{Write}(i, x)$ sets $A[i] := x$

The array maintenance problem (a.k.a. ORAM)

Two operations to maintain an n-slot array A

- Read(i) returns the current value stored in $A[i]$
- Write(i, x) sets $A[i] := x$

Theorem (Larsen+Nielsen '18)

Expected amortized running time of an ORAM with n b-bit slots is

$$\Omega\left(\frac{b}{w} \cdot \log \frac{nb}{c}\right)$$

where c is the client memory in bits.

The array maintenance problem (a.k.a. ORAM)

Two operations to maintain an n-slot array A

- Read(i) returns the current value stored in $A[i]$
- Write(i, x) sets $A[i] := x$

Theorem (Larsen+Nielsen '18)

Expected amortized running time of an ORAM with n b-bit slots is

$$\Omega\left(\frac{b}{w} \cdot \log \frac{nb}{c}\right)$$

where c is the client memory in bits.

Online Read and Write operations with Passive Server

Proof strategy for ORAM lower bound [Larsen+Nielsen]

The Information Transfer Technique [Pătrașcu+Demaine]

- assign probes to nodes of the Information Tree
  - each probe to at most one node
- show that for most nodes v there exists a hard distribution $\mathbf{HD_v}$ on sequences of operations of the same length that assign lots of probes to v
  - coding argument leveraging on randomness of the entries of the array
- invoke **obliviousness** to show that for each such distribution all nodes must be assigned the same high number of probes

Obliviousness

- very strong requirement
- it hides the type of operation
- it hides the parameters of the operations
  - the content of the array (for Write)
  - the slot of the operation (for Read and Write)
- only number of operations is leaked

Obliviousness

- very strong requirement
- it hides the type of operation
- it hides the parameters of the operations
  - the content of the array (for Write)
  - the slot of the operation (for Read and Write)
- only number of operations is leaked

In several applications more information is leaked for the sake of efficiency

Differential Privacy

Definition

DS is (ϵ, δ)-DP, if for every PPT machine $\mathcal{A}$ and any two sequences O and O' of the same length that differ for exactly one operation

$$\text{Prob}\left[\mathcal{A}(\text{view}^{\text{eMM}}(O)) = 1\right] \leq e^{\epsilon} \cdot \text{Prob}\left[\mathcal{A}(\text{view}^{\text{eMM}}(O')) = 1\right] + \delta$$

The Differentially Private RAM

Theorem (P+Yeo '19)

For every $\epsilon > 0$ and $\delta \leq 1/3$, the expected amortized running time of a Differentially Private RAM with n b-bit slots is

$$\Omega\left(\frac{b}{w} \cdot \log \frac{nb}{c}\right)$$

where c is the client memory in bits.

The Differentially Private RAM

Theorem (P+Yeo '19)

For every $\epsilon > 0$ and $\delta \leq 1/3$, the expected amortized running time of a Differentially Private RAM with n b-bit slots is

$$\Omega\left(\frac{b}{w} \cdot \log \frac{nb}{c}\right)$$

where c is the client memory in bits.

Different proof technique

Leakage Cell Probe Model

A sequence of operations $O = (\mathtt{op}_1, \mathtt{op}_2, \ldots, \mathtt{op}_l)$ is associated with leakage $\mathcal{L}(O)$

$$\mathcal{L}(O) = (\mathcal{L}(\mathtt{op}_1), \ldots, \mathcal{L}(\mathtt{op}_l))$$

Leakage Cell Probe Model

A sequence of operations $O = (\mathtt{op}_1, \mathtt{op}_2, \ldots, \mathtt{op}_l)$ is associated with leakage $\mathcal{L}(O)$

$$\mathcal{L}(O) = (\mathcal{L}(\mathtt{op}_1), \ldots, \mathcal{L}(\mathtt{op}_l))$$

Definition

DS is Non-Adaptively $\mathcal{L}$-INDSecure, if for every PPT machine $\mathcal{A}$ and any two sequences O and O' such that $\mathcal{L}(O) = \mathcal{L}(O')$,

$$\left|\mathrm{Prob}\left[\mathcal{A}(\mathrm{view}^{\mathrm{DS}}(O)) = 1\right] - \mathrm{Prob}\left[\mathcal{A}(\mathrm{view}^{\mathrm{DS}}(O')) = 1\right]\right| \leq \frac{1}{4}.$$

Leakage Cell Probe Model

A sequence of operations $O = (\mathrm{op}_1, \mathrm{op}_2, \ldots, \mathrm{op}_l)$ is associated with leakage $\mathcal{L}(O)$

$$\mathcal{L}(O) = (\mathcal{L}(\mathrm{op}_1), \ldots, \mathcal{L}(\mathrm{op}_l))$$

Definition

DS is Non-Adaptively $\mathcal{L}$-INDSecure, if for every PPT machine $\mathcal{A}$ and any two sequences O and O' such that $\mathcal{L}(O) = \mathcal{L}(O')$,

$$\left|\mathrm{Prob}\left[\mathcal{A}(\mathrm{view}^{\mathrm{DS}}(O)) = 1\right] - \mathrm{Prob}\left[\mathcal{A}(\mathrm{view}^{\mathrm{DS}}(O')) = 1\right]\right| \leq \frac{1}{4}.$$

Oblivious considers leakage $\mathcal{L}(O) = l$

Multi-Maps (MM)

Multi-Maps

A data structure to maintain a collection of pairs $(\mathtt{key}, \vec{\mathtt{v}})$, where $\vec{\mathtt{v}} = (v_1, \ldots, v_l)$ is a tuple

1. $\mathrm{Add}(\mathtt{key}, \mathtt{v})$: *adds* $\mathtt{v}$ *to the tuple associated with* $\mathtt{key}$
2. $\mathrm{Get}(\mathtt{key})$: *returns the tuple associated with* $\mathtt{key}$

Multi-Maps (MM)

Multi-Maps

A data structure to maintain a collection of pairs $(\mathtt{key}, \vec{\mathrm{v}})$, where $\vec{\mathrm{v}} = (v_1, \ldots, v_l)$ is a tuple

1. $\mathrm{Add}(\mathtt{key}, \mathtt{v})$: *adds* $\mathtt{v}$ *to the tuple associated with* $\mathtt{key}$
2. $\mathrm{Get}(\mathtt{key})$: *returns the tuple associated with* $\mathtt{key}$

- A special case of Structured Encryption [Chase-Kamara]

Multi-Maps (MM)

Multi-Maps

A data structure to maintain a collection of pairs $(\mathtt{key}, \vec{\mathrm{v}})$, where $\vec{\mathrm{v}} = (v_1, \ldots, v_l)$ is a tuple

1. $\mathrm{Add}(\mathtt{key}, \mathtt{v})$: *adds* $\mathtt{v}$ *to the tuple associated with* $\mathtt{key}$
2. $\mathrm{Get}(\mathtt{key})$: *returns the tuple associated with* $\mathtt{key}$

- A special case of Structured Encryption [Chase-Kamara]
- A generalization of ORAM:

Multi-Maps (MM)

Multi-Maps

A data structure to maintain a collection of pairs $(\mathtt{key}, \vec{\mathtt{v}})$, where $\vec{\mathtt{v}} = (v_1, \ldots, v_l)$ is a tuple

1. $\mathsf{Add}(\mathtt{key}, \mathtt{v})$: *adds* $\mathtt{v}$ *to the tuple associated with* $\mathtt{key}$
2. $\mathsf{Get}(\mathtt{key})$: *returns the tuple associated with* $\mathtt{key}$

- A special case of Structured Encryption [Chase-Kamara]
- A generalization of ORAM:
  - ORAM is a MM with all tuples of length 1;

How expensive are EMM?

How expensive are EMM?

It depends on the leakage function

How expensive are EMM?

It depends on the leakage function

If no security is sought:

$$O\left(\frac{\log\log n}{\log\log\log n}\right)$$

[Beame and Fich '99]

How expensive are EMM?

It depends on the leakage function

If no security is sought:

$$O\left(\frac{\log\log n}{\log\log\log n}\right)$$

[Beame and Fich '99]

If only number of operations is leaked

$$O(\log n)$$

Use ORAM [Folklore]

How expensive are EMM?

It depends on the leakage function

If no security is sought:

$$O\left(\frac{\log\log n}{\log\log\log n}\right)$$

[Beame and Fich '99]

If only number of operations is leaked

$$O(\log n)$$

Use ORAM [Folklore]

What if we only want to hide the response of the operations?

How expensive are EMM?

It depends on the leakage function

If no security is sought:

$$O\left(\frac{\log \log n}{\log \log \log n}\right)$$

[Beame and Fich '99]

If only number of operations is leaked

$$O(\log n)$$

Use ORAM [Folklore]

What if we only want to hide the response of the operations?

What is the cost of the Response-Hiding EMM?

Response-Hiding Leakage Function – I

Definition (Leakage function $\mathcal{L}^G$ for $O = (\mathrm{op}_1, \ldots, \mathrm{op}_l)$)

$\mathcal{L}^G(O_i)$ is defined as follows:

1. if $\mathrm{op}_i = \mathrm{Get}(\mathtt{key}_i)$ then $\mathcal{L}^G(O_i) = \left(\mathrm{Get}, \mathtt{key}_i, \left|\mathrm{Get}\left(\mathrm{MM}^{O_{i-1}}, \mathtt{key}_i\right)\right|\right)$;
   the key queried and the size of the response are leaked
2. if $\mathrm{op}_i = \mathrm{Add}(\mathtt{key}_i, \mathtt{v}_i)$ then $\mathcal{L}^G(O_i) = \left(\mathrm{Add}, \mathrm{aep}^i\right)$
   the add pattern is leaked

the type of operation is also leaked

add equality pattern $\mathrm{aep}^i := (\mathrm{aep}^i_1, \ldots, \mathrm{aep}^i_{i-1})$ and aep^i_j is defined as follows, for $j = 1, \ldots, i-1$

$$\mathrm{aep}^i_j = \begin{cases} \perp, & \text{if } \mathrm{op}_j \text{ is a Get operation;} \\ 0, & \text{if } \mathrm{op}_j \text{ is an Add operation and } \mathtt{key}_j \neq \mathtt{key}_i; \\ 1, & \text{if } \mathrm{op}_j \text{ is an Add operation and } \mathtt{key}_j = \mathtt{key}_i; \end{cases}$$

Response-Hiding Leakage Function – II

Definition (Leakage function $\mathcal{L}^A$ for $O = (\mathrm{op}_1, \ldots, \mathrm{op}_l)$)

$\mathcal{L}^A(O_i)$ is defined as follows:

1. if $\mathrm{op}_i = \mathrm{Get}(\mathtt{key}_i)$ then $\mathcal{L}^A(O_i) = \left(\mathrm{Get}, \left|\mathrm{Get}\left(\mathrm{MM}^{O_{i-1}}, \mathtt{key}_i\right)\right|, \mathrm{gep}^i\right)$;
   the size of the response and the equality pattern are leaked
2. if $\mathrm{op}_i = \mathrm{Add}(\mathtt{key}_i, \mathtt{v}_i)$ then $\mathcal{L}^A(O_i) = (\mathrm{Add}, \mathtt{key}_i, \mathtt{v}_i)$
   all the parameters of an Add

the type of operation is also leaked

get equality pattern $\mathrm{gep}^i := (\mathrm{gep}^i_1, \ldots, \mathrm{gep}^i_{i-1})$ and gep^i_j is defined as follows, for $j = 1, \ldots, i-1$

$$\mathrm{gep}^i_j = \begin{cases} \perp, & \text{if } \mathrm{op}_j \text{ is a Add operation;} \\ 0, & \text{if } \mathrm{op}_j \text{ is an Get operation and } \mathtt{key}_j \neq \mathtt{key}_i; \\ 1, & \text{if } \mathrm{op}_j \text{ is an Get operation and } \mathtt{key}_j = \mathtt{key}_i; \end{cases}$$

Main result

Theorem (Informal)

$\mathcal{L}^G$-INDSecurity and $\mathcal{L}^A$-INDSecurity EMM have $\Omega(\log n)$ expected amortized overhead.

Main result

Theorem (Informal)

$\mathcal{L}^G$-INDSecurity and $\mathcal{L}^A$-INDSecurity EMM have $\Omega(\log n)$ expected amortized overhead.

A sequence of operations that return R responses requires $\Omega(R \cdot \log n)$ work.

Main result

Theorem (Informal)

$\mathcal{L}^G$-INDSecurity and $\mathcal{L}^A$-INDSecurity EMM have $\Omega(\log n)$ expected amortized overhead.

A sequence of operations that return R responses requires $\Omega(R \cdot \log n)$ work.

This is tight [Folklore]

- Use ORAM and spend $O(\log n)$

Proof technique

We adapt the Information Transfer technique of [P+D] to our setting

- we have a **weaker** security notion
  - can only invoke obliviousness for distribution with same leakage
  - we prove lower bound for very **leaky** implementations
- in our data structure problem entries/values are **not** random
  - need to identify a different source of randomness for the encoding argument

Defining the Hard Distribution **HD** for $\mathcal{L}^G$

we have

1. the following disjoint sets of values
   - V_0 consisting of k values;
   - $V_1, \ldots, V_p$ each consisting of n^ϵ values;

2. the following disjoint sets of keys:
   - sets K_i^{a}, for $i = 1, \ldots, p$, each of size n^ϵ;
   - sets K_i^{g}, for $i = 1, \ldots, p$, each of size n^ϵ;

Defining the Hard Distribution **HD** for $\mathcal{L}^G$

we have

1. the following disjoint sets of values
   - V_0 consisting of k values;
   - $V_1, \ldots, V_p$ each consisting of n^ϵ values;

2. the following disjoint sets of keys:
   - sets K_i^{a}, for $i = 1, \ldots, p$, each of size n^ϵ;
   - sets K_i^{g}, for $i = 1, \ldots, p$, each of size n^ϵ;

$p = n^{1-\epsilon}$

Defining the Hard Distribution **HD**

Phase 0

Execute SubPhase $\mathrm{I_i}$, for $i = 1, \ldots, p$

for each $\texttt{key} \in K_i^{\mathrm{g}}$

output: $\mathrm{Add}(\texttt{key}, V_0)$,

Phase j, for $j = 1, \ldots, p$

Execute SubPhase $\mathrm{A_j}$ and SubPhase $\mathrm{G_j}$

- SubPhase $\mathrm{A_j}$

  for each $\texttt{key} \in K_j^{\mathrm{a}}$,

  randomly select subset $B_{\texttt{key}} \subset V_j$ of k values

  output: $\mathrm{Add}(\texttt{key}, B_{\texttt{key}})$;

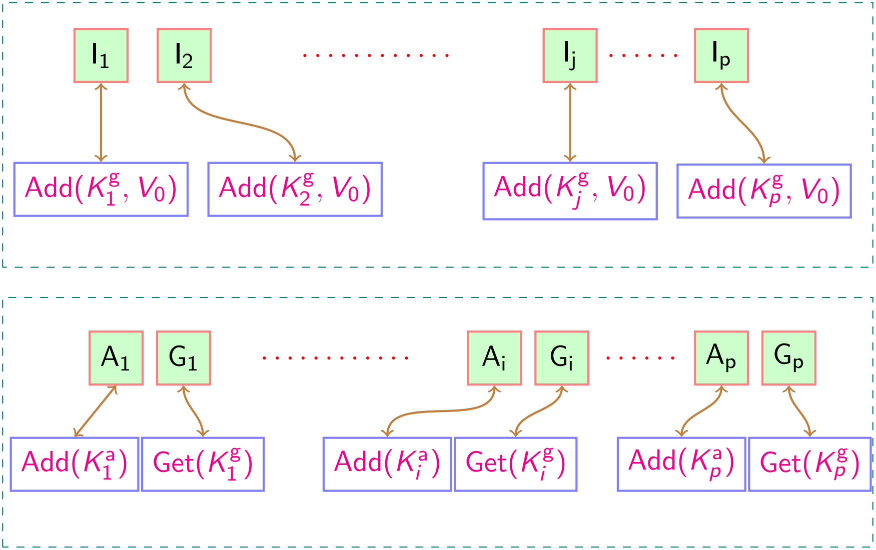
- SubPhase $\mathrm{G_j}$

  for each $\texttt{key} \in K_j^{\mathrm{g}}$

  output: $\mathrm{Get}(\texttt{key})$;

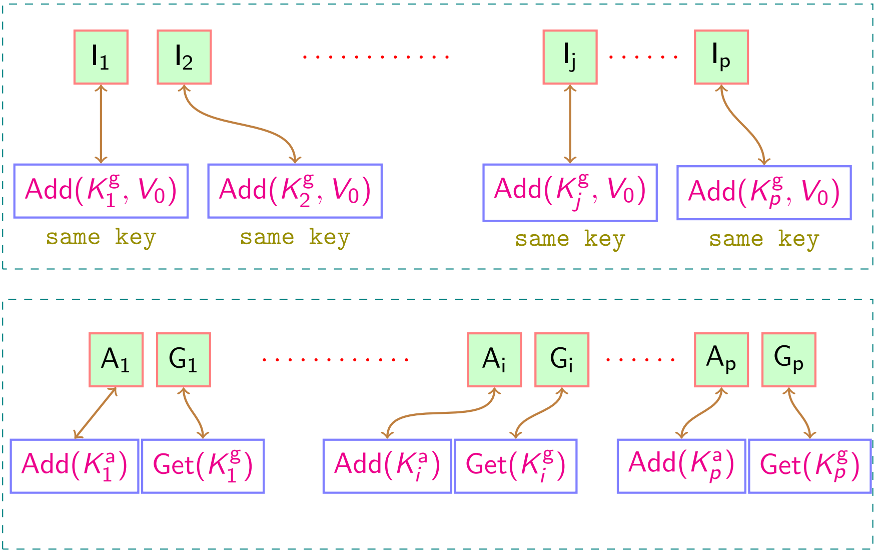
The Hard Distribution **HD**

InitPhase

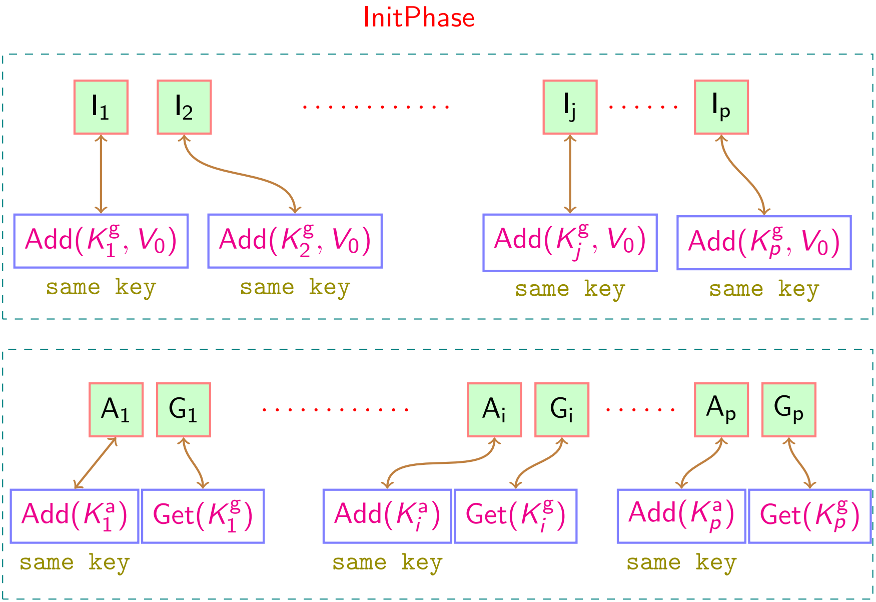


The Hard Distribution **HD**

InitPhase

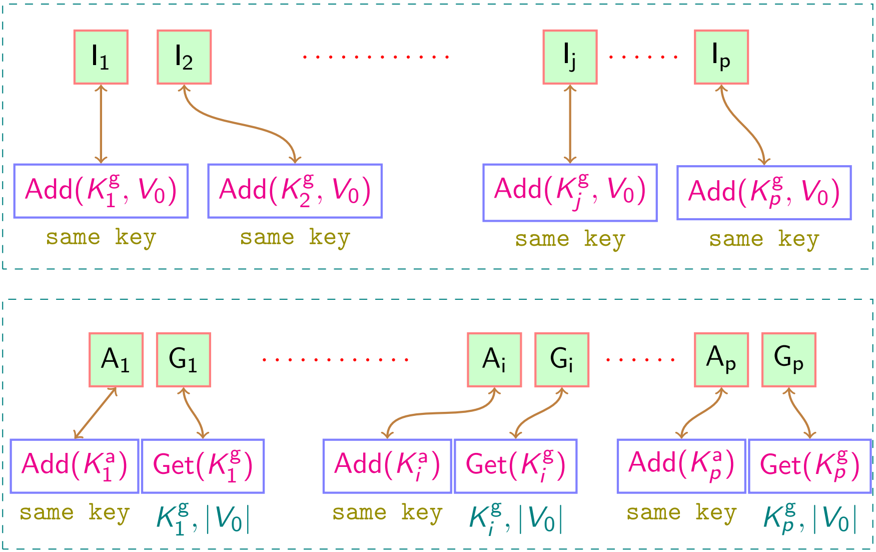


The Hard Distribution **HD**

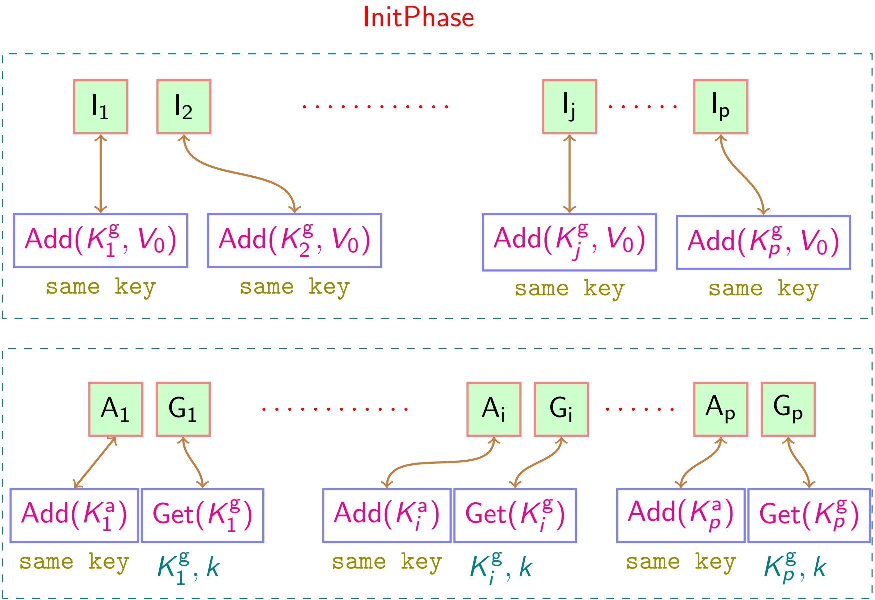


The Hard Distribution **HD**

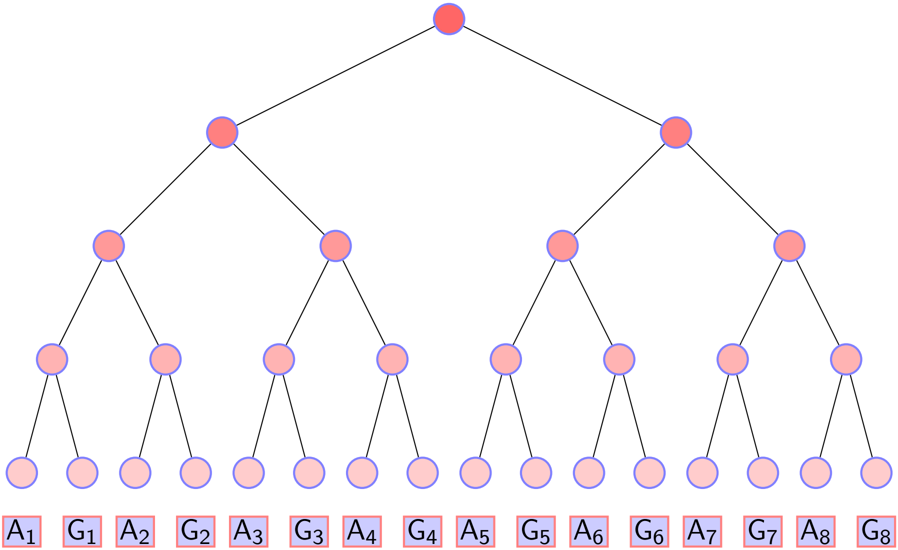
InitPhase



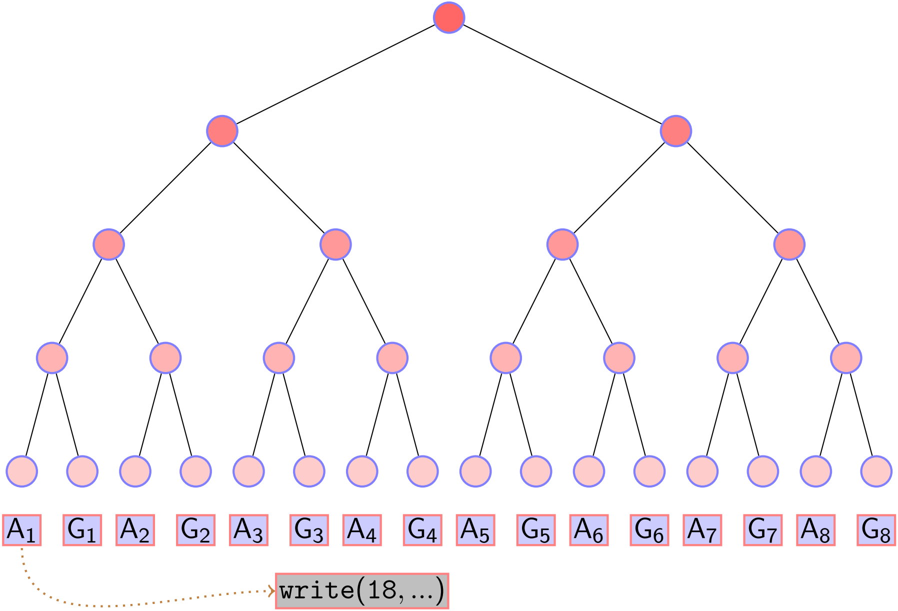
The Hard Distribution **HD**



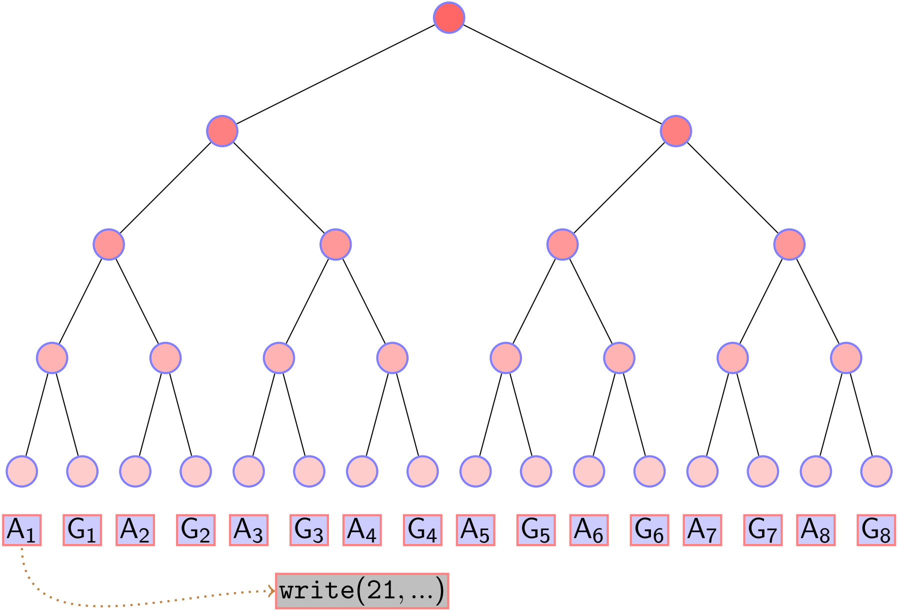
The Information Tree of the Hard Distribution



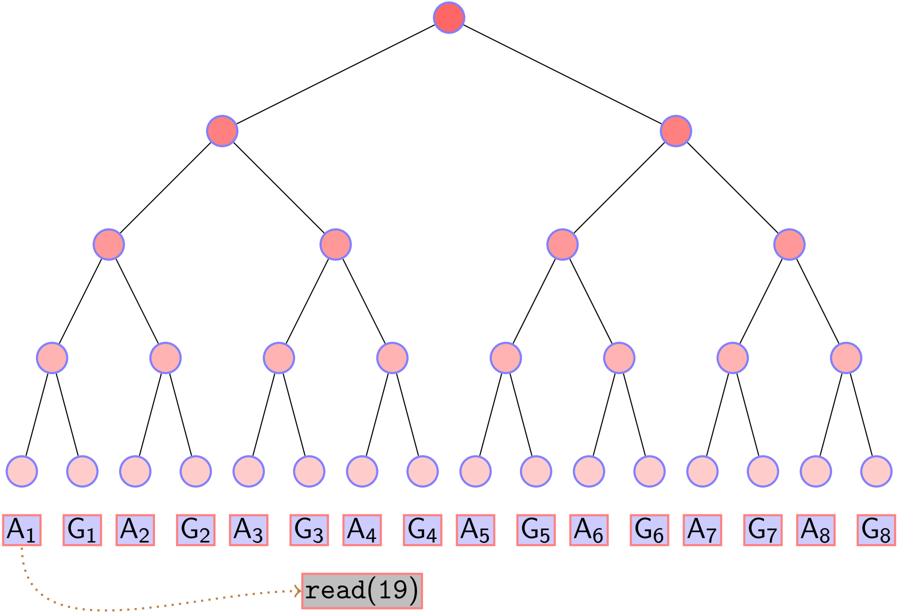
The Information Tree of the Hard Distribution



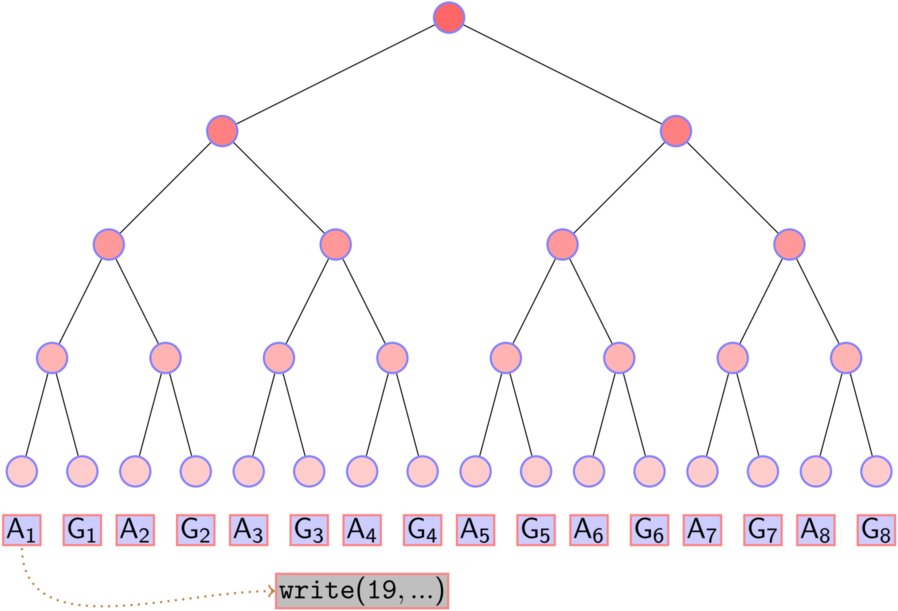
The Information Tree of the Hard Distribution



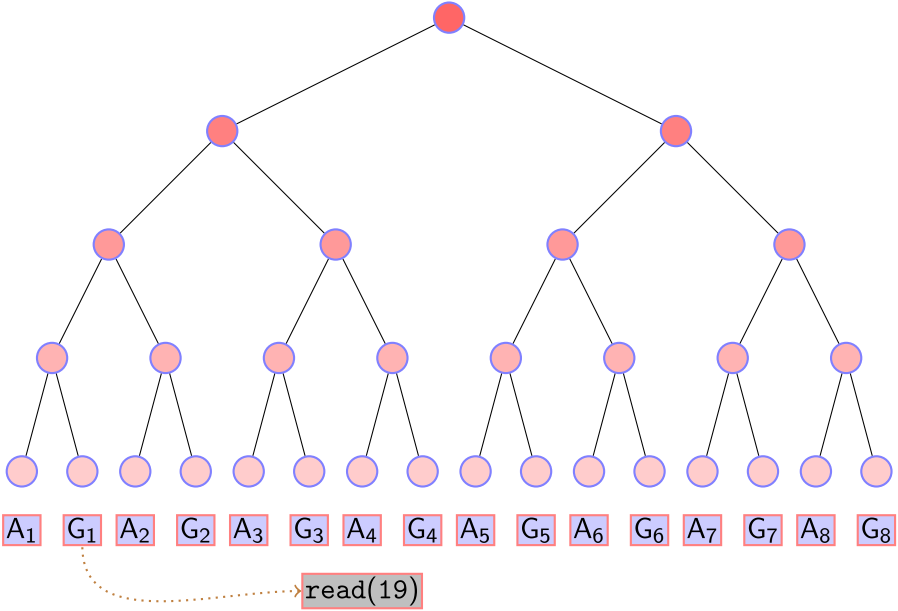
The Information Tree of the Hard Distribution



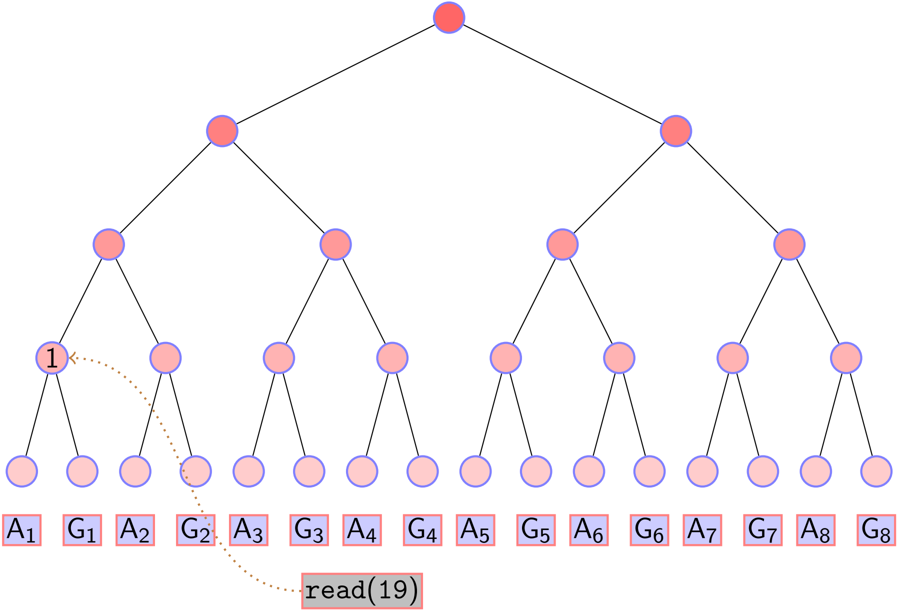
The Information Tree of the Hard Distribution



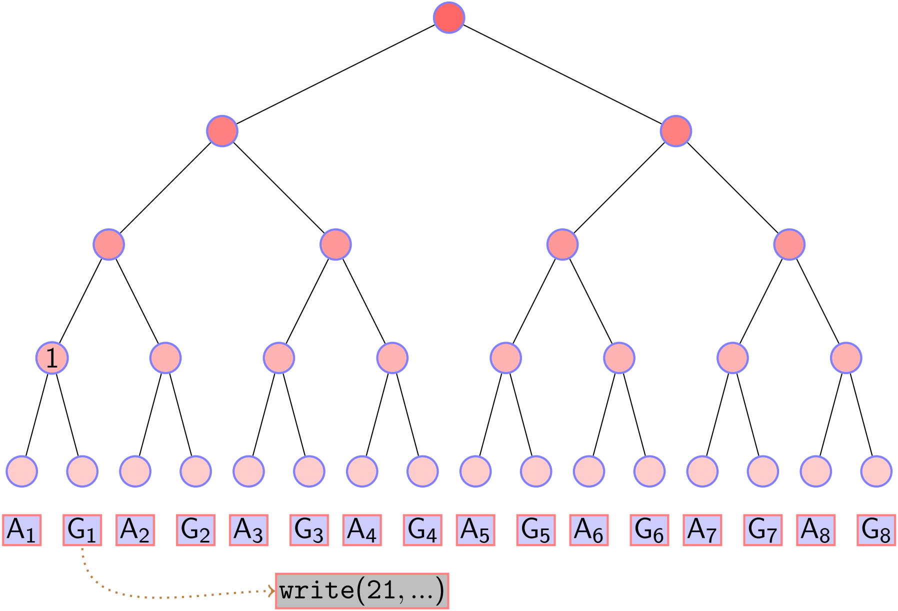
The Information Tree of the Hard Distribution



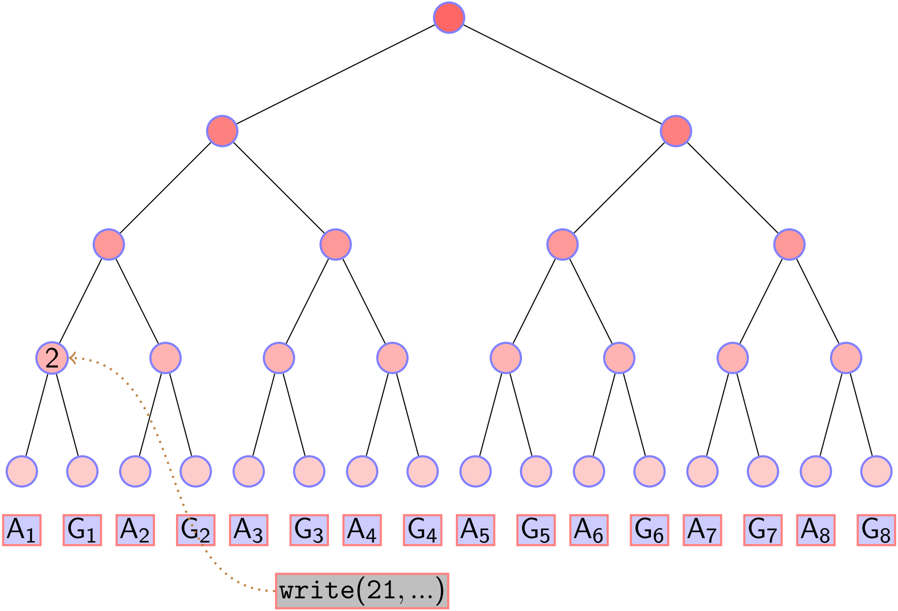
The Information Tree of the Hard Distribution



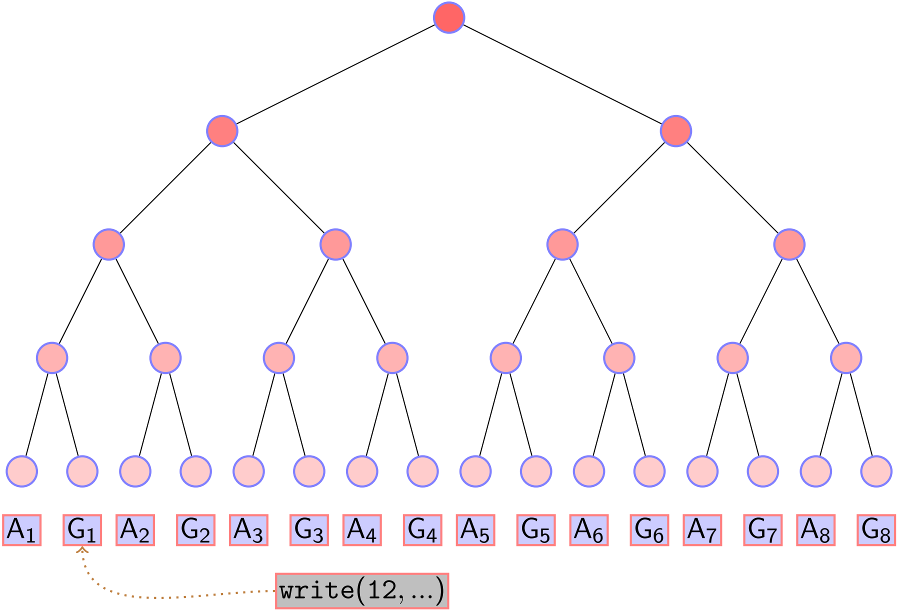
The Information Tree of the Hard Distribution



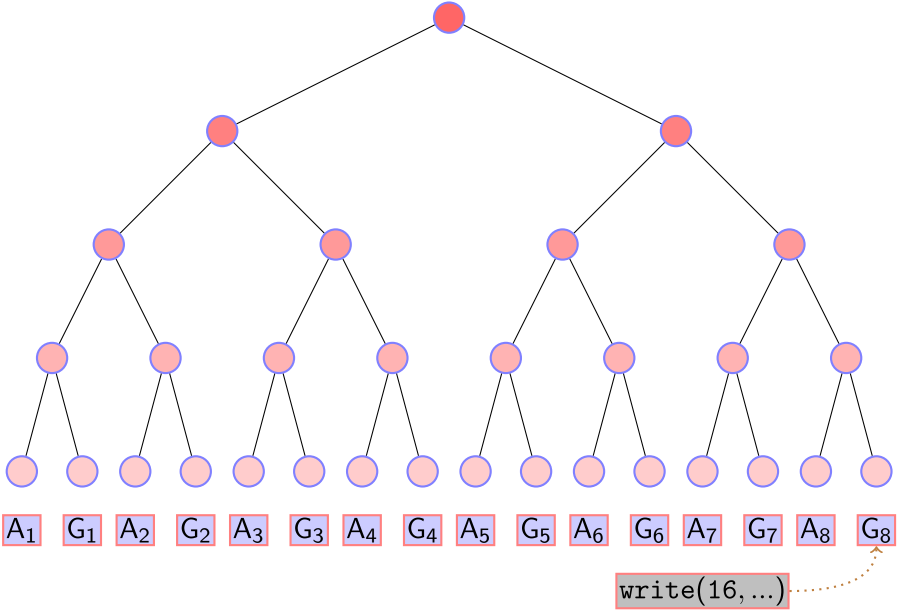
The Information Tree of the Hard Distribution



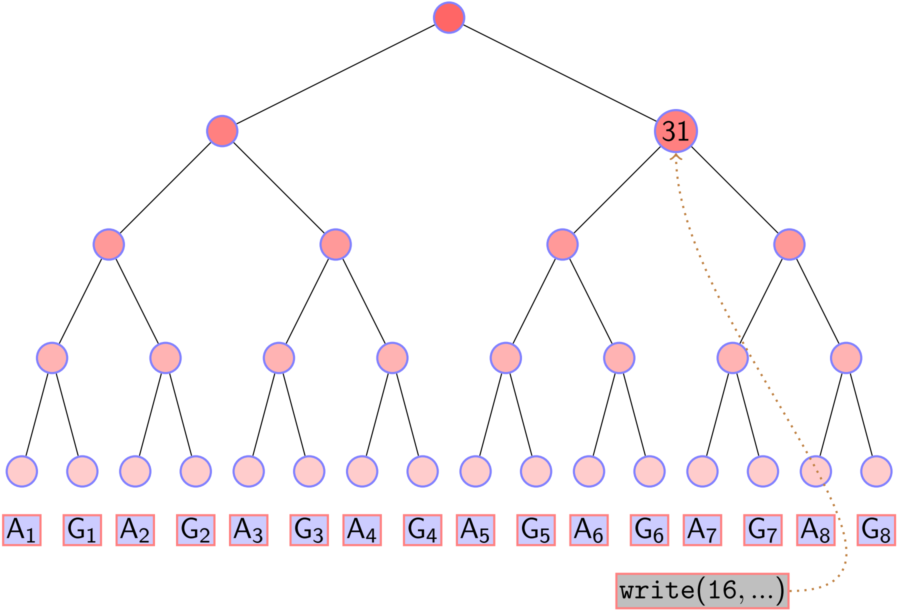
The Information Tree of the Hard Distribution



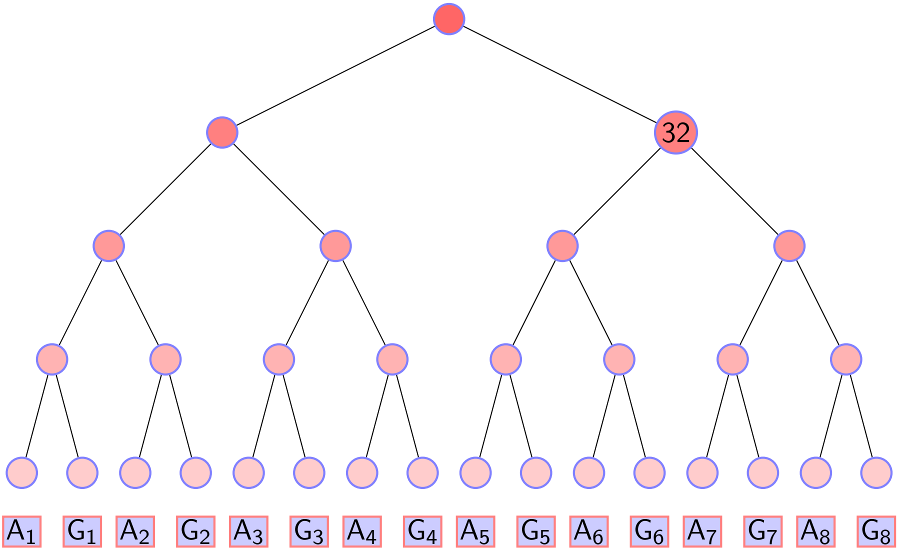
The Information Tree of the Hard Distribution



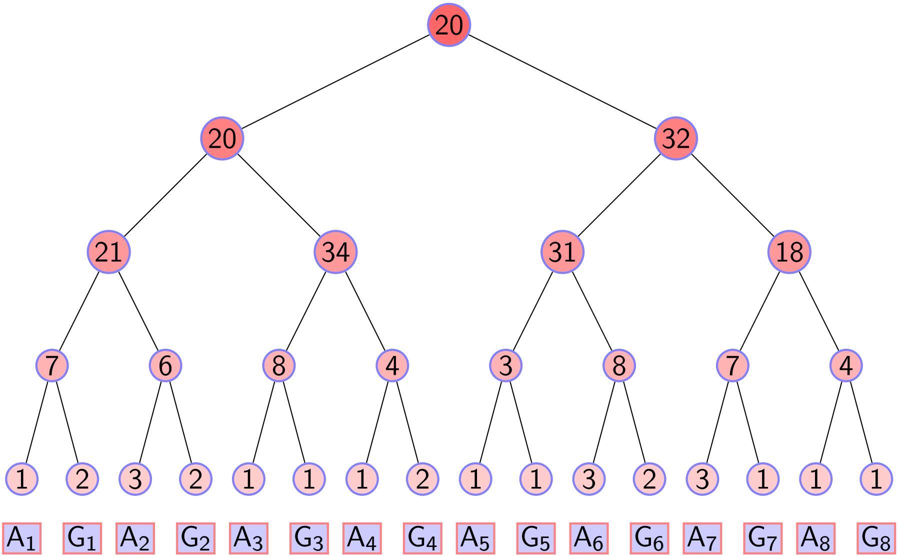
The Information Tree of the Hard Distribution



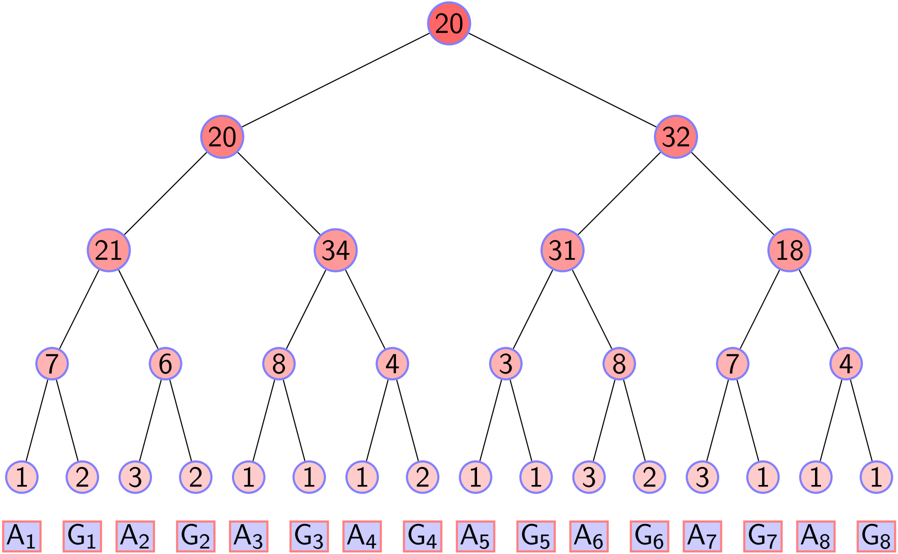
The Information Tree of the Hard Distribution



The Information Tree of the Hard Distribution

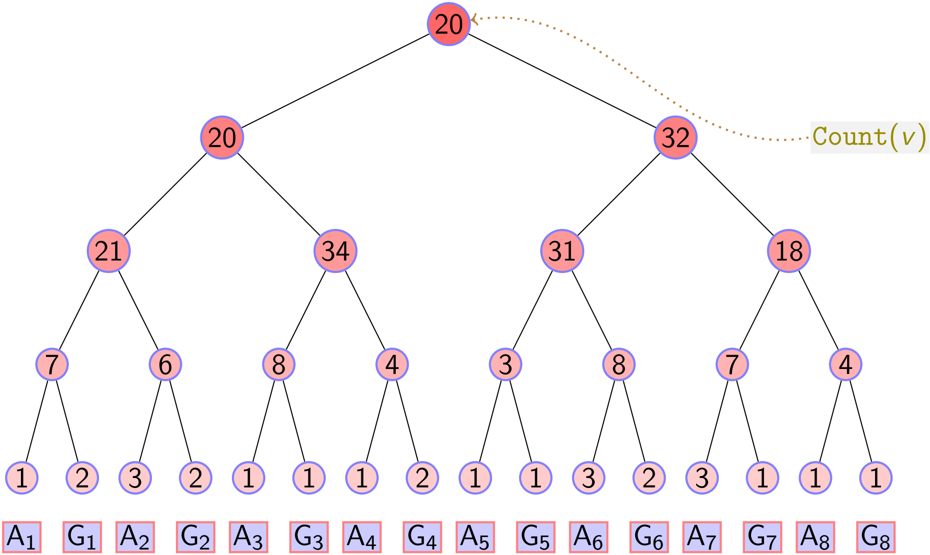


The Information Tree of the Hard Distribution



Each probe is assigned to at most one node

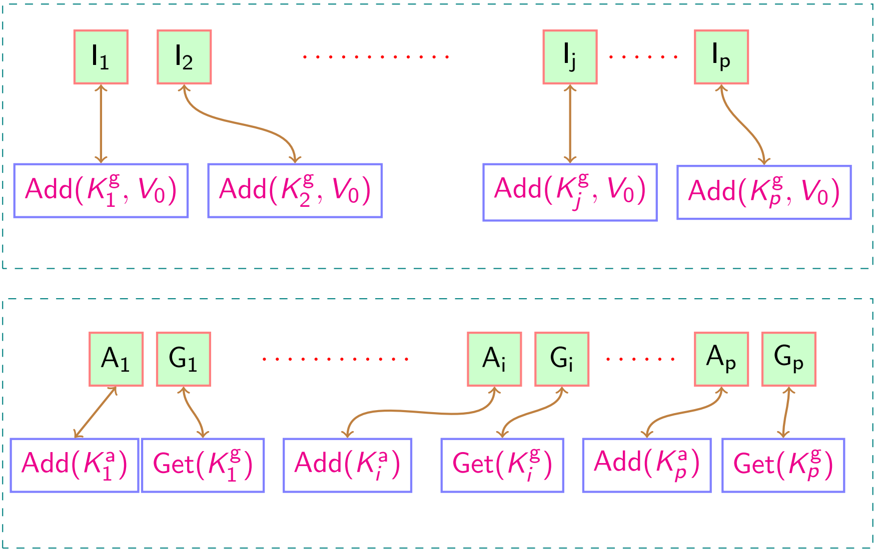
The Information Tree of the Hard Distribution



Each probe is assigned to at most one node

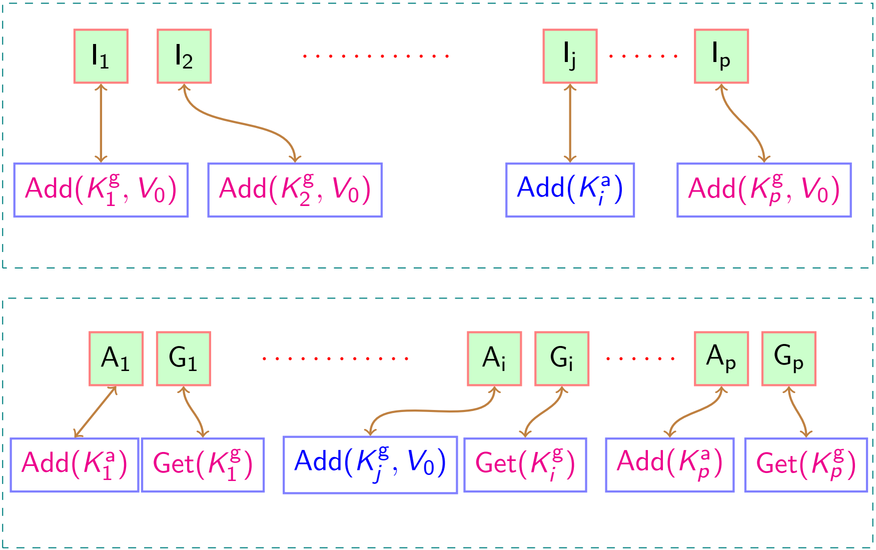
The Neighbor Hard Distributions

InitPhase



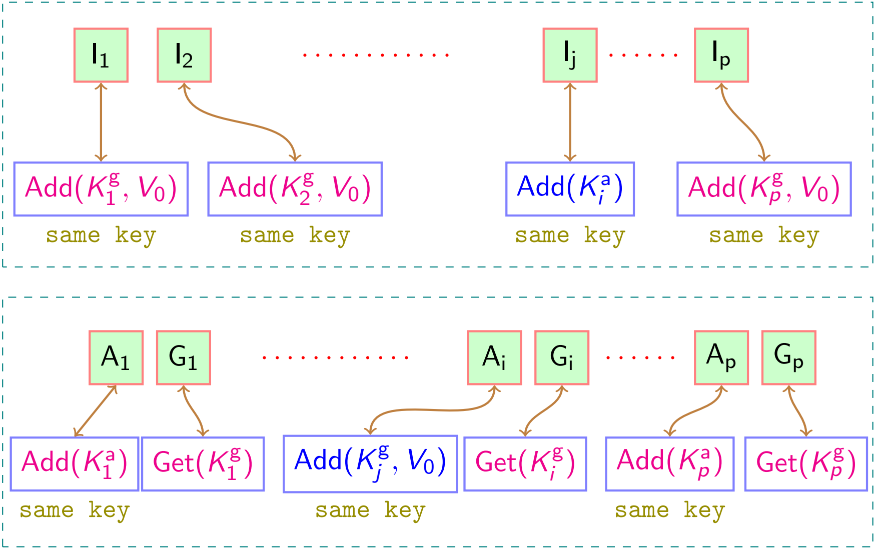
The Neighbor Hard Distributions

InitPhase $\mathbf{i} \leq \mathbf{j}$



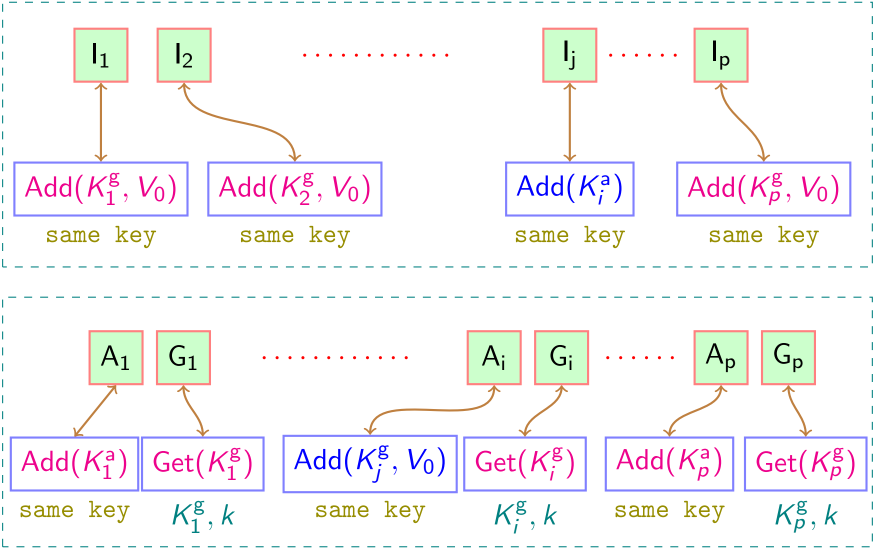
The Neighbor Hard Distributions

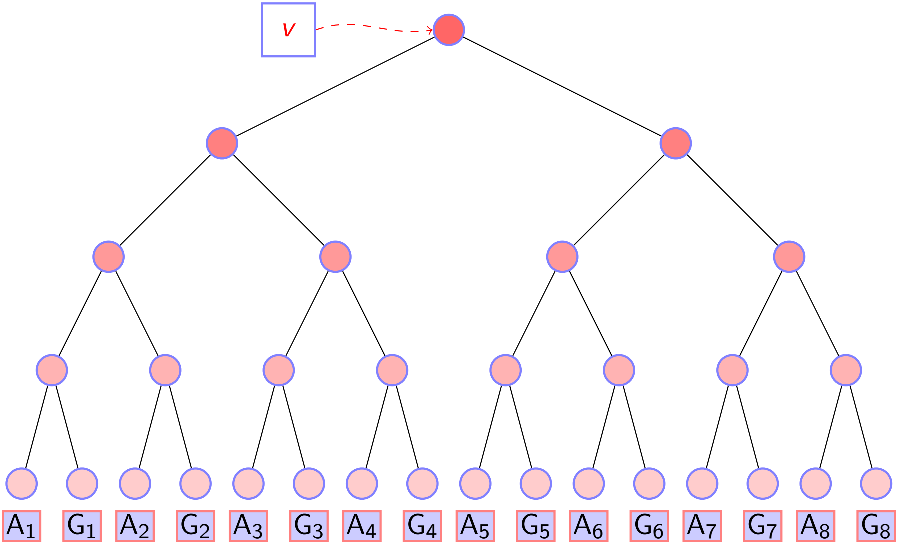
InitPhase $\mathbf{i \leq j}$

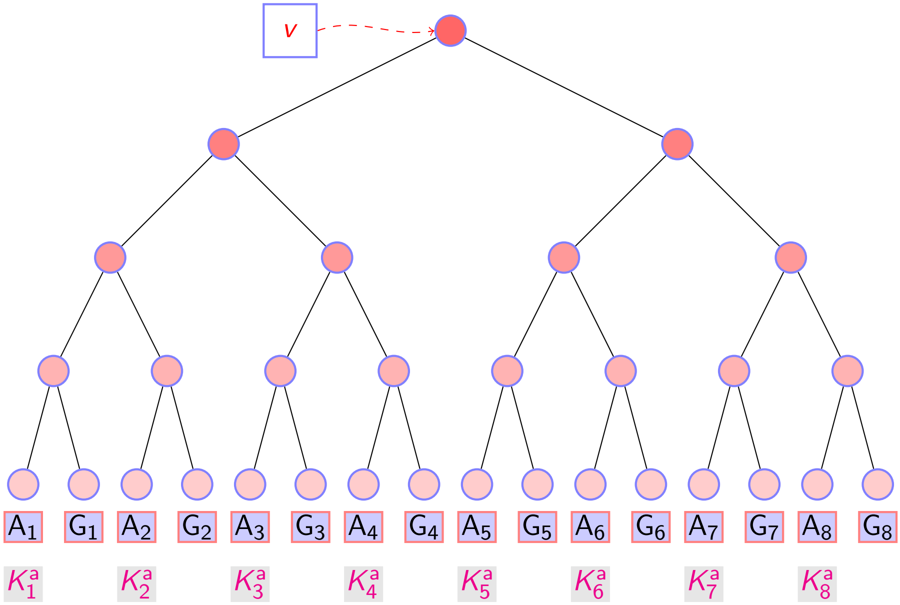


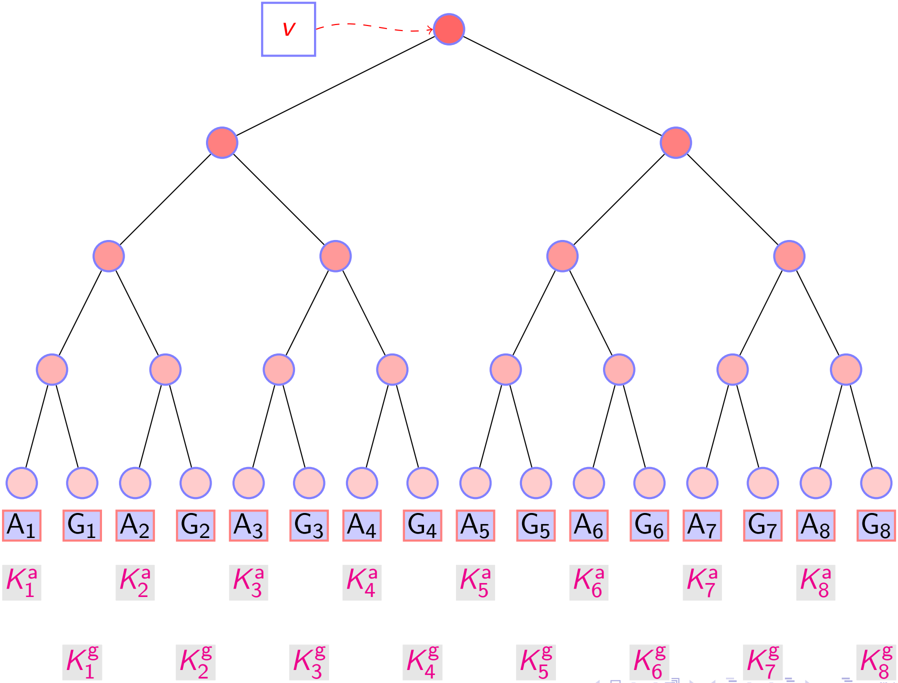
The Neighbor Hard Distributions

InitPhase $\qquad$ $\mathbf{i \leq j}$

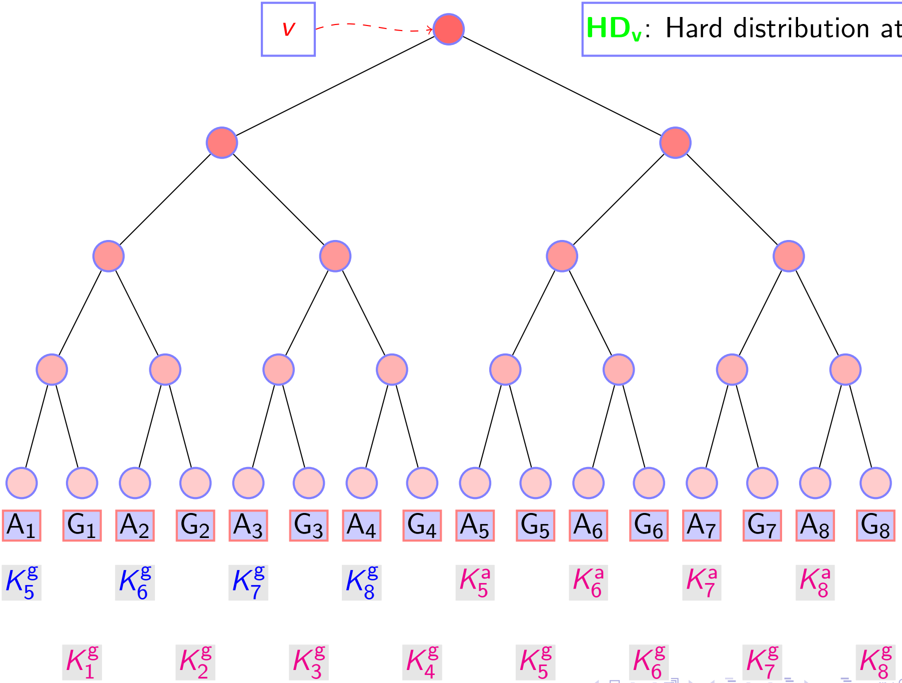


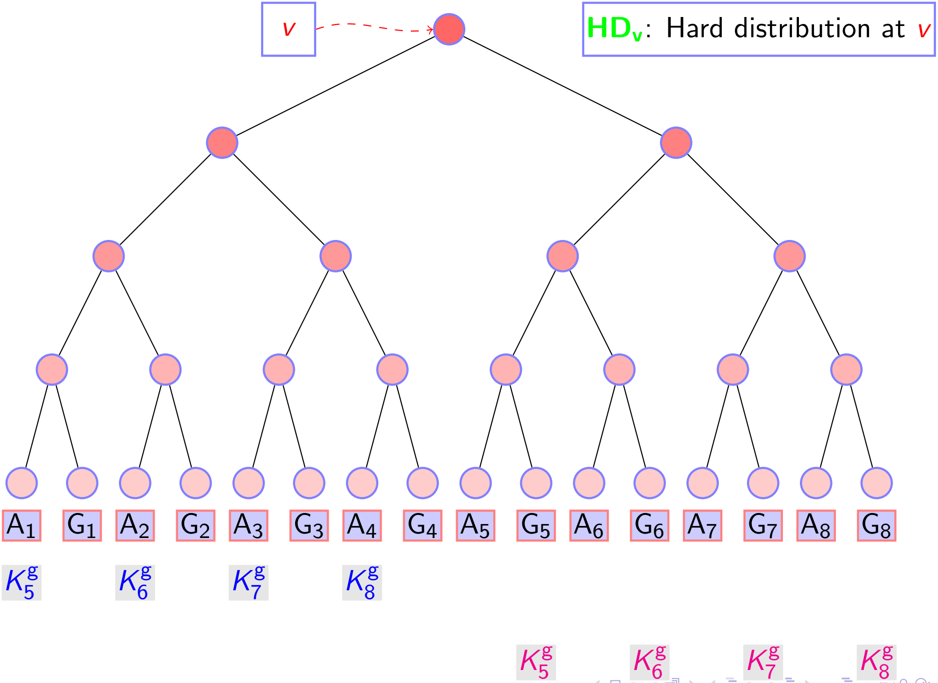

v
A1
G1
A2
G2
A3
G3
A4
G4
A5
G5
A6
G6
A7
G7
A8
G8


v
A1
G1
A2
G2
A3
G3
A4
G4
A5
G5
A6
G6
A7
G7
A8
G8
K1a
K2a
K3a
K4a
K5a
K6a
K7a
K8a


v
A_1 G_1 A_2 G_2 A_3 G_3 A_4 G_4 A_5 G_5 A_6 G_6 A_7 G_7 A_8 G_8
K^a_1 K^a_2 K^a_3 K^a_4 K^a_5 K^a_6 K^a_7 K^a_8
K^g_1 K^g_2 K^g_3 K^g_4 K^g_5 K^g_6 K^g_7 K^g_8

$\mathbf{HD_v}$: Hard distribution at v

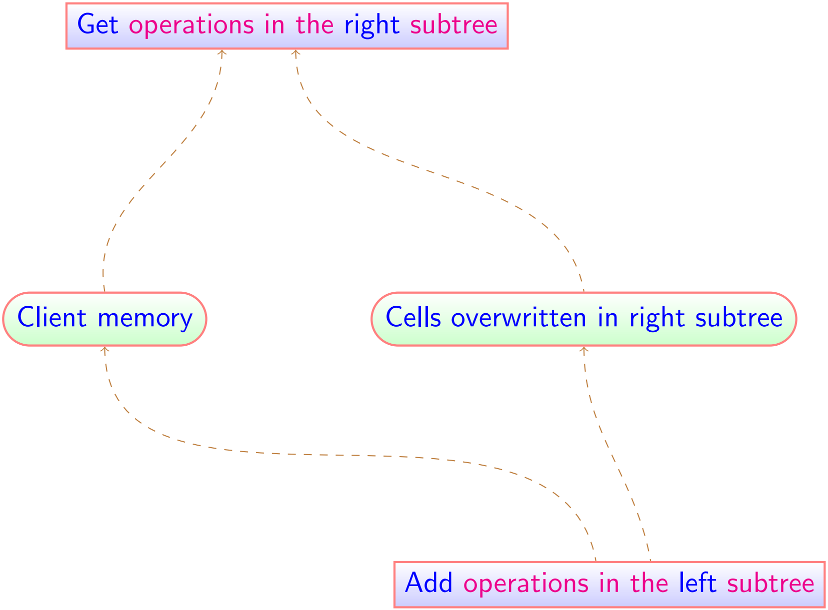


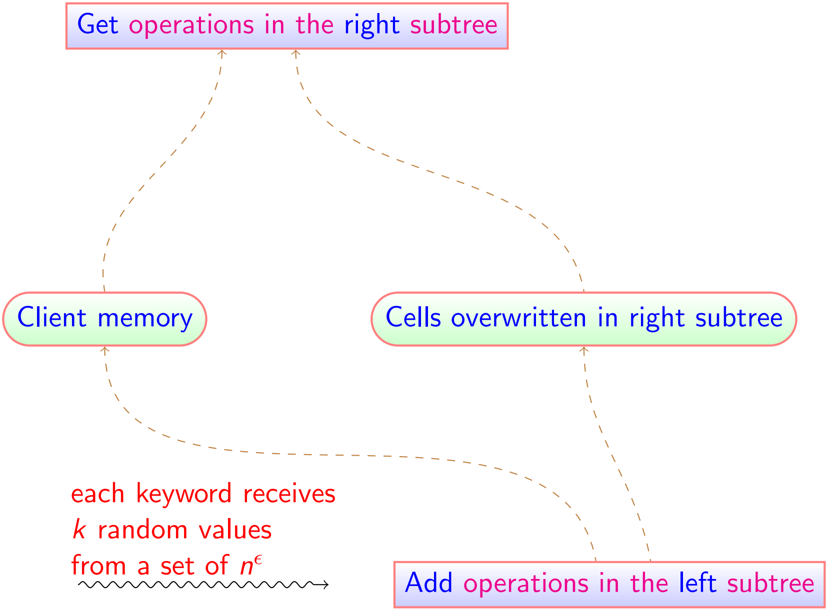


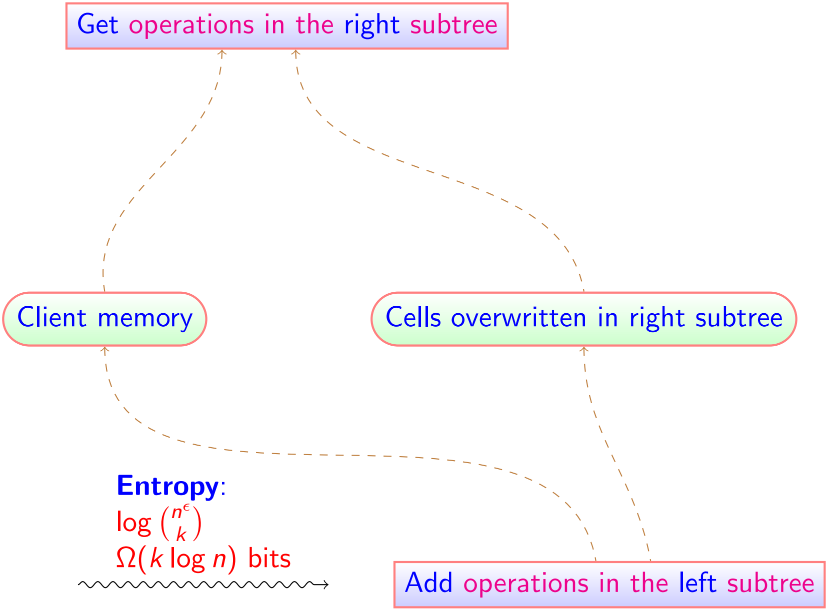
v
HDv: Hard distribution at v
A1 G1 A2 G2 A3 G3 A4 G4 A5 G5 A6 G6 A7 G7 A8 G8
K_5^g K_6^g K_7^g K_8^g
K_5^g K_6^g K_7^g K_8^g

Get operations in the right subtree

Add operations in the left subtree


Get operations in the right subtree
Client memory
Cells overwritten in right subtree
Add operations in the left subtree


Get operations in the right subtree
Client memory
Cells overwritten in right subtree
each keyword receives
k random values
from a set of n^{ϵ}
Add operations in the left subtree


Get operations in the right subtree
Client memory
Cells overwritten in right subtree
Entropy:
$\log \binom{n^{\epsilon}}{k}$
$\Omega(k \log n)$ bits
Add operations in the left subtree

Theorem

For every v *of the* *information tree* *of depth* $8 \leq d \leq \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$\mathbb{E}\left[|\mathrm{Count}(v)|\right] = \Omega\left(\frac{n}{2^d} \cdot k \cdot \frac{\log n}{w}\right)$$

with respect to $\mathbf{HD}_v$.

Theorem

For every v of the information tree of depth $8 \le d \le \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$\mathbb{E}\left[|\mathsf{Count}(v)|\right] = \Omega\left(\frac{n}{2^d} \cdot k \cdot \frac{\log n}{w}\right)$$

with respect to $\mathbf{HD}_v$.

For every v, $\mathcal{L}^G(\mathbf{HD_v}) = \mathcal{L}^G(\mathbf{HD})$, so by $\mathcal{L}^G$-INDsecurity,

Theorem

For every v of the information tree of depth $8 \leq d \leq \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$\mathbb{E}\left[|\mathrm{Count}(v)|\right] = \Omega\left(\frac{n}{2^d} \cdot k \cdot \frac{\log n}{w}\right)$$

with respect to $\mathbf{HD}_v$.

For every v, $\mathcal{L}^G(\mathbf{HD_v}) = \mathcal{L}^G(\mathbf{HD})$, so by $\mathcal{L}^G$-INDsecurity,

Theorem

For every v of the information tree of depth $8 \leq d \leq \frac{1-\epsilon}{2} \log \frac{n}{c}$

$$\mathbb{E}\left[|\mathrm{Count}(v)|\right] = \Omega\left(\frac{n}{2^d} \cdot k \cdot \frac{\log n}{w}\right)$$

with respect to $\mathbf{HD}$.

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\mathsf{Count}(v)$.

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\mathsf{Count}(v)$.
  - $\sum_v \mathsf{Count}(v)$ is a lower bound to the number of probes

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\mathsf{Count}(v)$.
  - $\sum_v \mathsf{Count}(v)$ is a lower bound to the number of probes
- level d has 2^d nodes,

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\mathsf{Count}(v)$.
  - $\sum_v \mathsf{Count}(v)$ is a lower bound to the number of probes
- level d has 2^d nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\mathsf{Count}(v)$.
  - $\sum_v \mathsf{Count}(v)$ is a lower bound to the number of probes
- level d has 2^d nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\mathsf{Count}(v)$.
  - $\sum_v \mathsf{Count}(v)$ is a lower bound to the number of probes
- level d has 2^d nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega\left(n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c}\right)$$

  to execute

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one $\mathsf{Count}(v)$.
  - $\sum_v \mathsf{Count}(v)$ is a lower bound to the number of probes
- level d has 2^d nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega\left(n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c}\right)$$

to execute

- $\Theta(nk)$ Add

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one Count(v).
  - $\sum_v \text{Count}(v)$ is a lower bound to the number of probes
- level d has 2^d nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega\left(n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c}\right)$$

  to execute
  - $\Theta(nk)$ Add
  - $\Theta(n)$ Get each with $\Theta(k)$ results each

Wrapping up

For an eMM that is $\mathcal{L}^G$-IND secure

- each probe contributes 1 to at most one Count(v).
  - $\sum_v$ Count(v) is a lower bound to the number of probes
- level d has 2^d nodes,
  - each level contributes $n \cdot k \cdot \frac{\log n}{w}$
- we have $\Theta(\log \frac{n}{c})$ levels
- number of probes is

$$\Omega\left(n \cdot k \cdot \frac{\log n}{w} \cdot \log \frac{n}{c}\right)$$

  to execute
  - $\Theta(nk)$ Add
  - $\Theta(n)$ Get each with $\Theta(k)$ results each
- amortized efficiency per response

$$\Omega\left(\frac{\log n}{w} \cdot \log \frac{n}{c}\right)$$

Typical parameter regime

$w = \Omega(\log n)$ and $c = n^{\alpha}$, $\alpha < 1$.

Typical parameter regime

$w = \Omega(\log n)$ and $c = n^{\alpha}$, $\alpha < 1$.

amortized efficiency per response of an eMM is

$$\Omega\left(\log n\right)$$

Typical parameter regime

$w = \Omega(\log n)$ and $c = n^{\alpha}$, $\alpha < 1$.

amortized efficiency per response of an eMM is

$$\Omega(\log n)$$

Same for $\mathcal{L}^A$ leakage function

Conclusions

- Response Hiding in a *mildly* Dynamic setting gives $\Omega(\log n)$ overhead
  - static EMM can be implemented with constant slowdown via cuckoo hashing

Conclusions

- Response Hiding in a *mildly* Dynamic setting gives $\Omega(\log n)$ overhead
  - static EMM can be implemented with constant slowdown via cuckoo hashing
  - proof only uses addition of values to keys

Conclusions

- Response Hiding in a *mildly* Dynamic setting gives $\Omega(\log n)$ overhead
  - static EMM can be implemented with constant slowdown via cuckoo hashing
  - proof only uses addition of values to keys
  - no remove operation