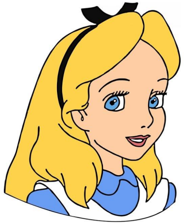
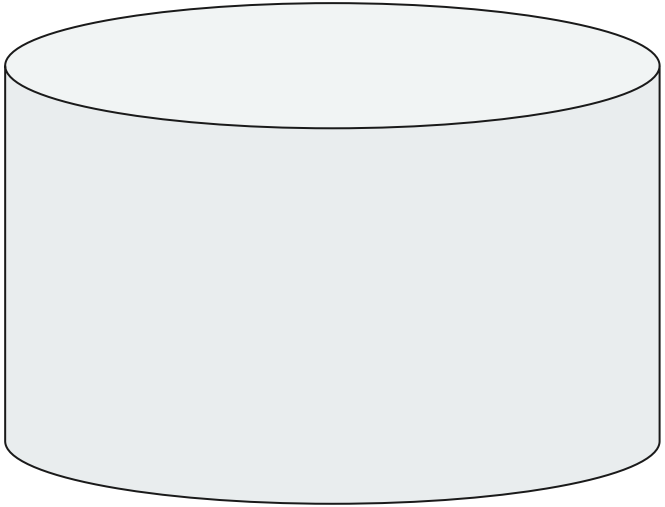
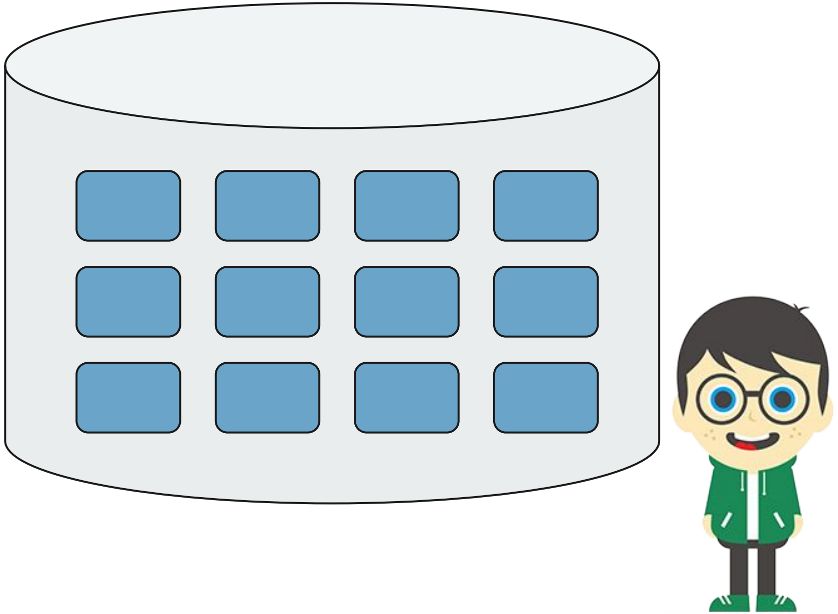
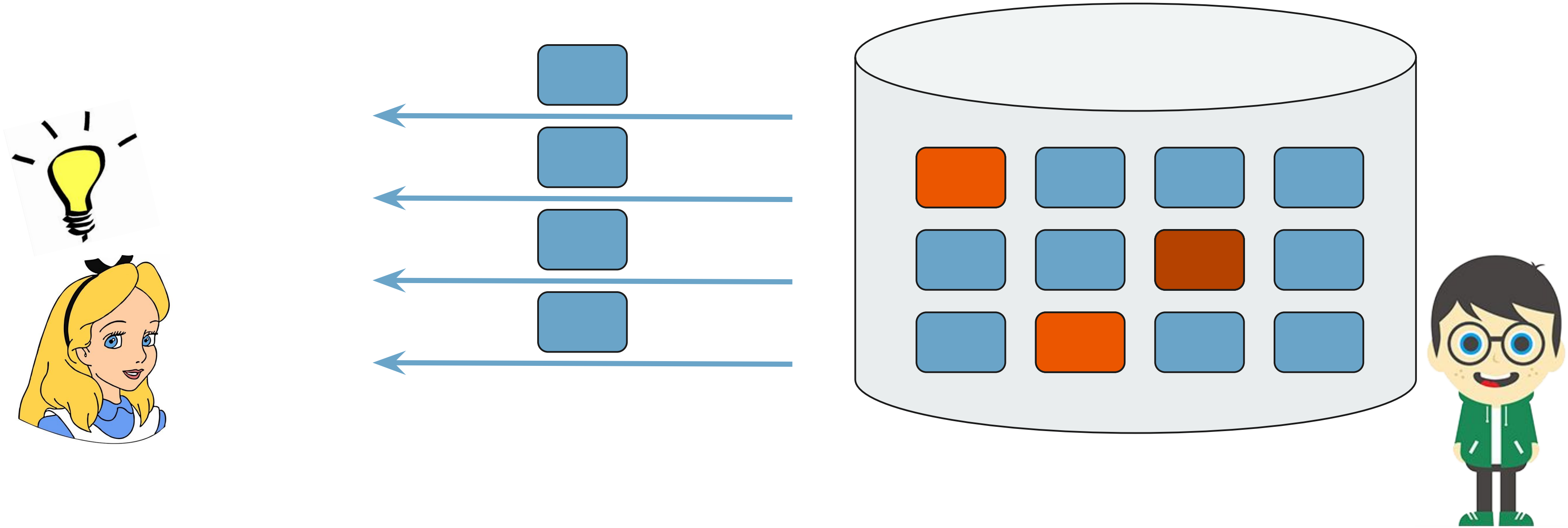
PanORAMa: Oblivious RAM with Logarithmic Overhead

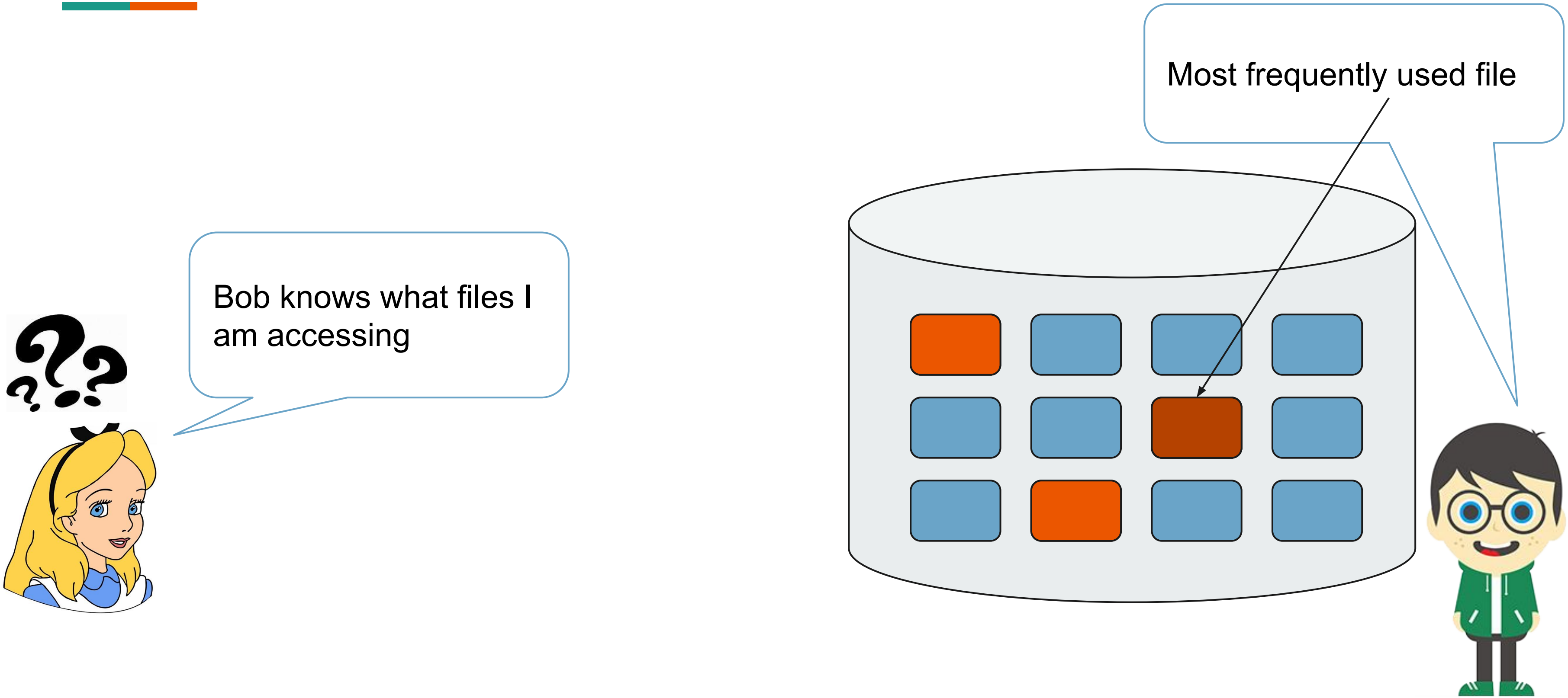
Sarvar Patel, Giuseppe Persiano, Mariana Raykova, Kevin Yeo

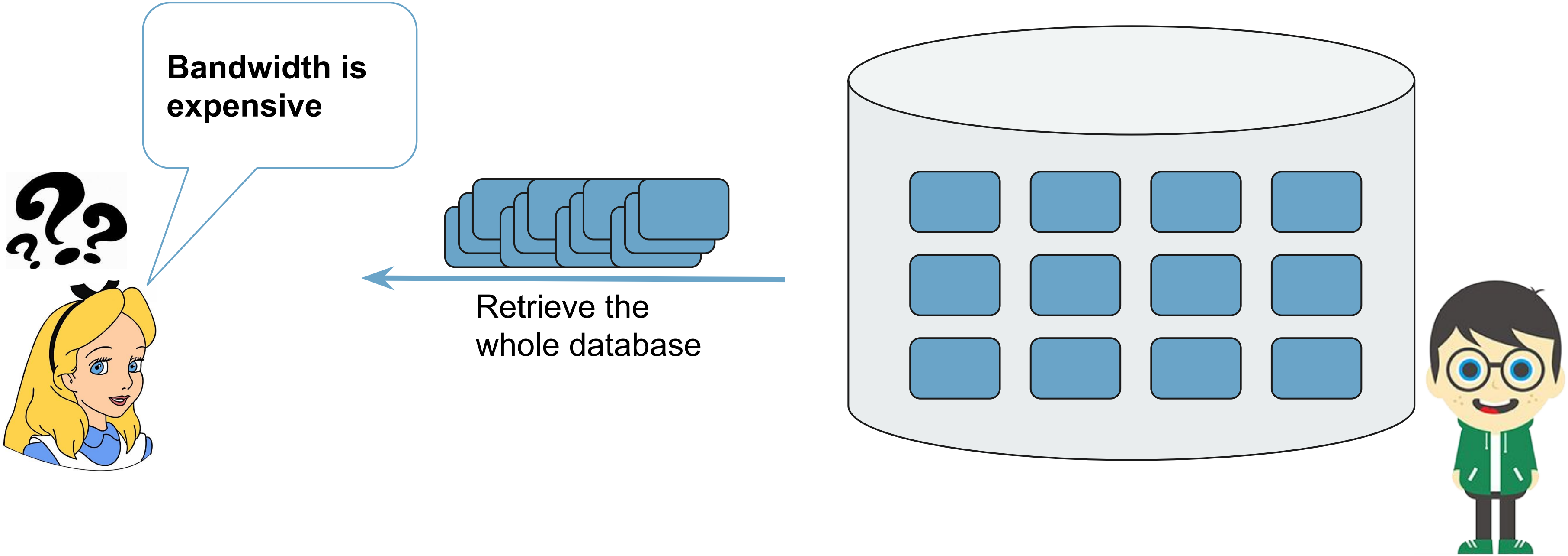




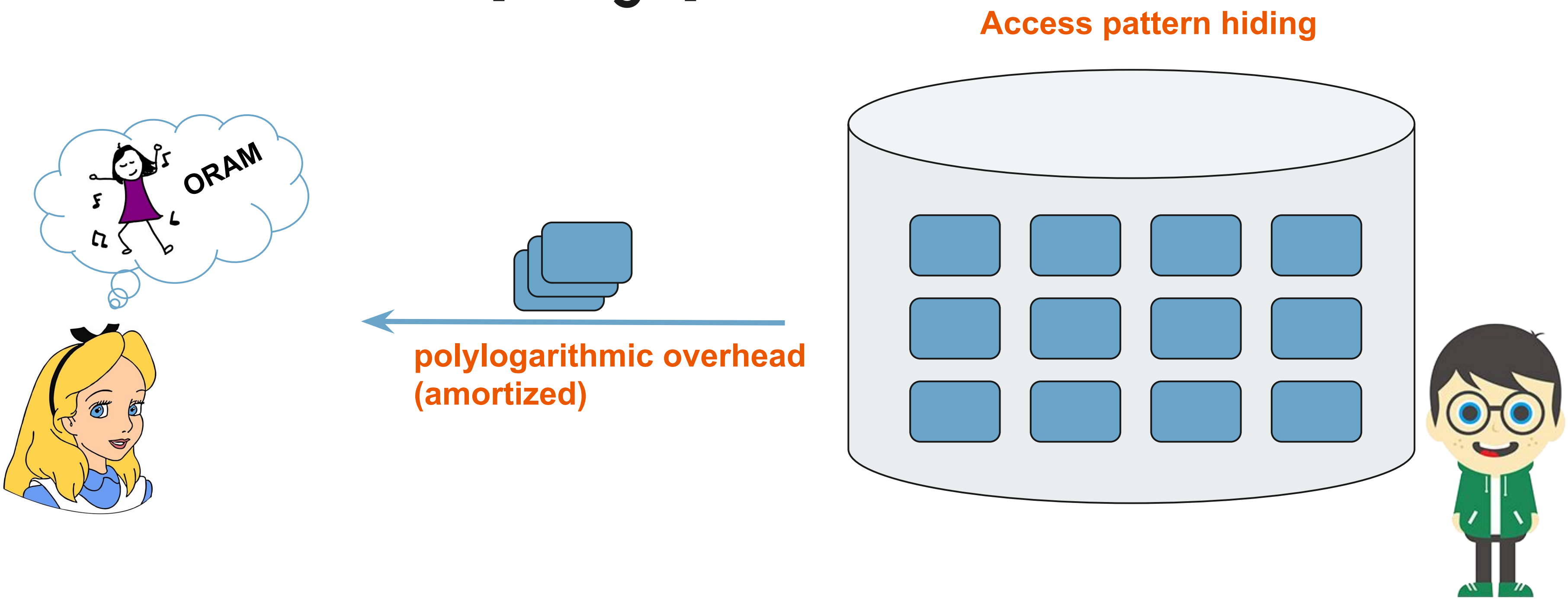





Bob knows what files I am accessing
Most frequently used file


Bandwidth is expensive
Retrieve the whole database

Oblivious RAM [GO'96]



How Efficient Can an ORAM Construction be?

ORAM Lower Bound

ORAM Lower Bound

- Goldreich-Ostrovsky'96
  - Lower bound **$O(\log_C N)$** blocks for database of **N blocks** and client memory of **C blocks**

ORAM Lower Bound

- Goldreich-Ostrovsky'96
  - Lower bound **$O(\log_C N)$** blocks for database of **N blocks** and client memory of **C blocks**
  - Caveats:
    - Server **only moves and retrieves** blocks (does not do any computation)
    - Constructions with **statistical security**
    - Constructions that **work for any block size**
    - The client can have **oracle access to a private random function**

ORAM Lower Bound

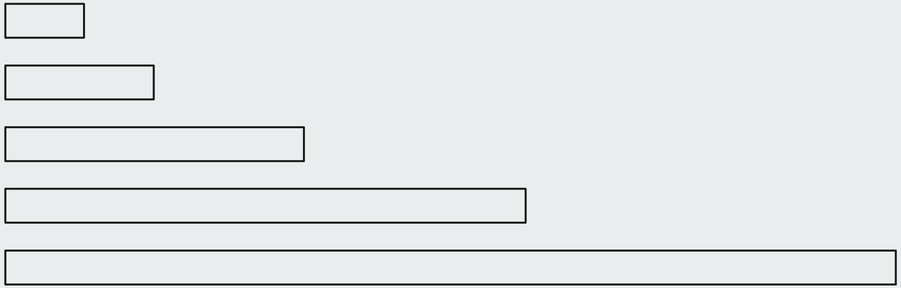
- Goldreich-Ostrovsky'96
  - Lower bound $\Omega(\log_C N)$ blocks for database of **N blocks** and client memory of **C blocks**
  - Caveats:
    - Server **only moves and retrieves** blocks (does not do any computation)
    - Constructions with **statistical security**
    - Constructions that **work for any block size**
    - The client can have **oracle access to a private random function**
- Boyle-Naor'16
  - Formalized the **"balls and bins"** model
  - Evidence for the hardness of extending the lower bound beyond the ball and bins model
    - **Reduction from sorting circuits to ORAM**

ORAM Lower Bound

- Goldreich-Ostrovsky'96
  - Lower bound **$\Omega(\log_C N)$** blocks for database of **N blocks** and client memory of **C blocks**
  - Caveats:
    - Server **only moves and retrieves** blocks (does not do any computation)
    - Constructions with **statistical security**
    - Constructions that **work for any block size**
    - The client can have **oracle access to a private random function**
- Boyle-Naor'16
  - Formalized the **"balls and bins"** model
  - Evidence for the hardness of extending the lower bound beyond the ball and bins model
    - **Reduction from sorting circuits to ORAM**
- Larsen-Nielsen'18
  - Lower bound extended to **computational online ORAM model (only block uploads/downloads)**

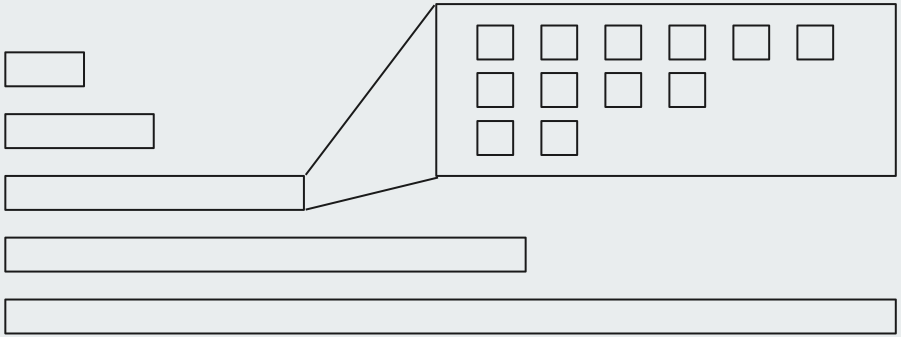
PanORAMa:

PanORAMa:



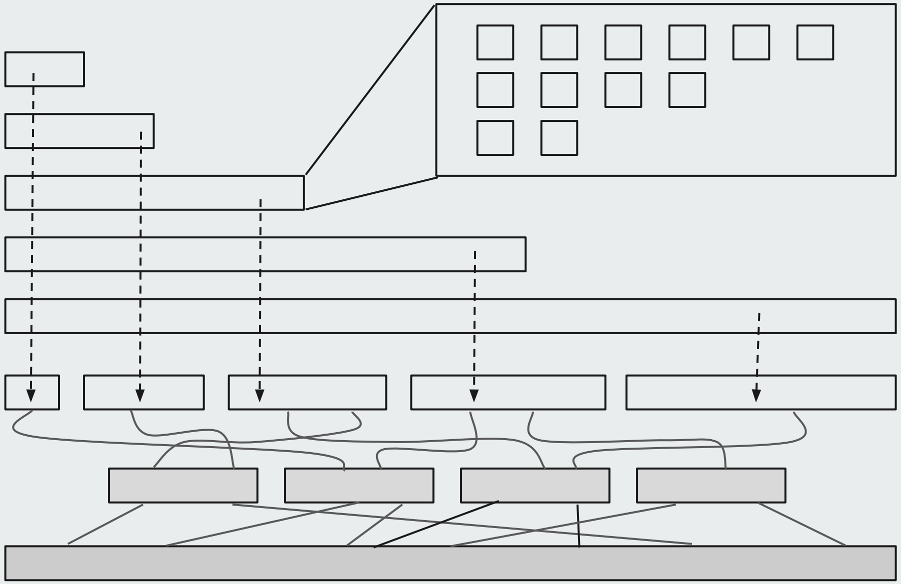
- New Oblivious RAM Construction
    - **Improved asymptotic communication**
        - $O(\log N \cdot \log\log N)$ blocks
    - Block size $\mathbf{\Omega(\log N)}$
    - Can be instantiated in the balls and bins model
    - Follows the hierarchical paradigm

PanORAMa:



- **New Oblivious RAM Construction**
  - **Improved asymptotic communication**
    - **$O(\log N \cdot \log\log N)$ blocks**
  - Block size **$\Omega(\log N)$**
  - Can be instantiated in the balls and bins model
  - Follows the hierarchical paradigm
- **New Oblivious Hash Table Construction**
  - Obliviousness for non-repeating queries
  - **Efficient initialization from random array**
  - Amortized query communication complexity
    - **$O(\log N + \mathrm{poly}(\log\log \lambda))$**

PanORAMa:



- **New Oblivious RAM Construction**
  - **Improved asymptotic communication**
    - **$O(\log N \cdot \log \log N)$ blocks**
  - Block size **$\Omega(\log N)$**
  - Can be instantiated in the balls and bins model
  - Follows the hierarchical paradigm
- **New Oblivious Hash Table Construction**
  - Obliviousness for non-repeating queries
  - **Efficient initialization from random array**
  - Amortized query communication complexity
    - **$O(\log N + \mathrm{poly}(\log \log \lambda))$**
- **New Multi-Array Shuffle Algorithm**
  - Efficient shuffle for **input with entropy**
  - Shuffle multiple sorted arrays
    - **$O(N \log \log \lambda + N \log N \log \lambda)$**
    - Not too many very small arrays

Construction Paradigms

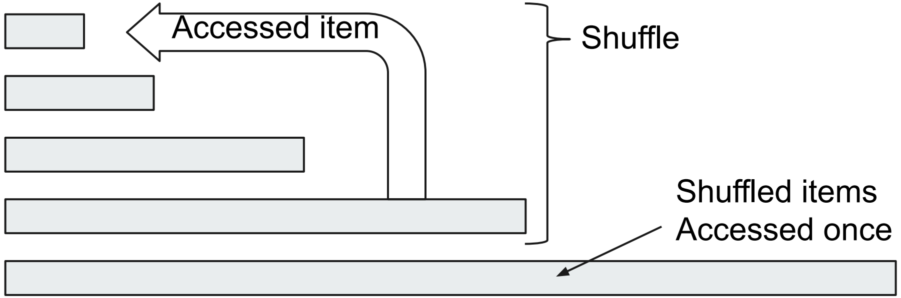
- Hierarchical ORAMs
  - Worst case ≠ Average case
  - Computation assumption beyond encryption in most cases

- Tree ORAMs
  - Worst case = Average case
  - Encryption - the only computational assumption

Construction Paradigms

- Hierarchical ORAMs
    - Worst case ≠ Average case
    - Computation assumption beyond encryption in most cases

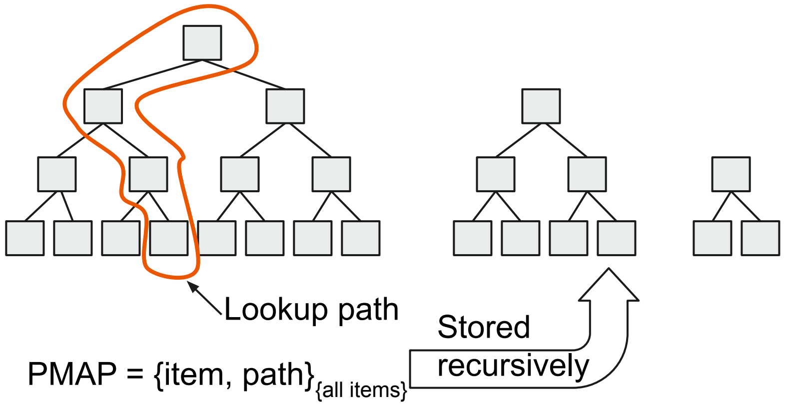
- Tree ORAMs
    - Worst case = Average case
    - Encryption - the only computational assumption



Construction Paradigms

- Hierarchical ORAMs
  - Worst case ≠ Average case
  - Computation assumption beyond encryption in most cases

- Tree ORAMs
  - Worst case = Average case
  - Encryption - the only computational assumption

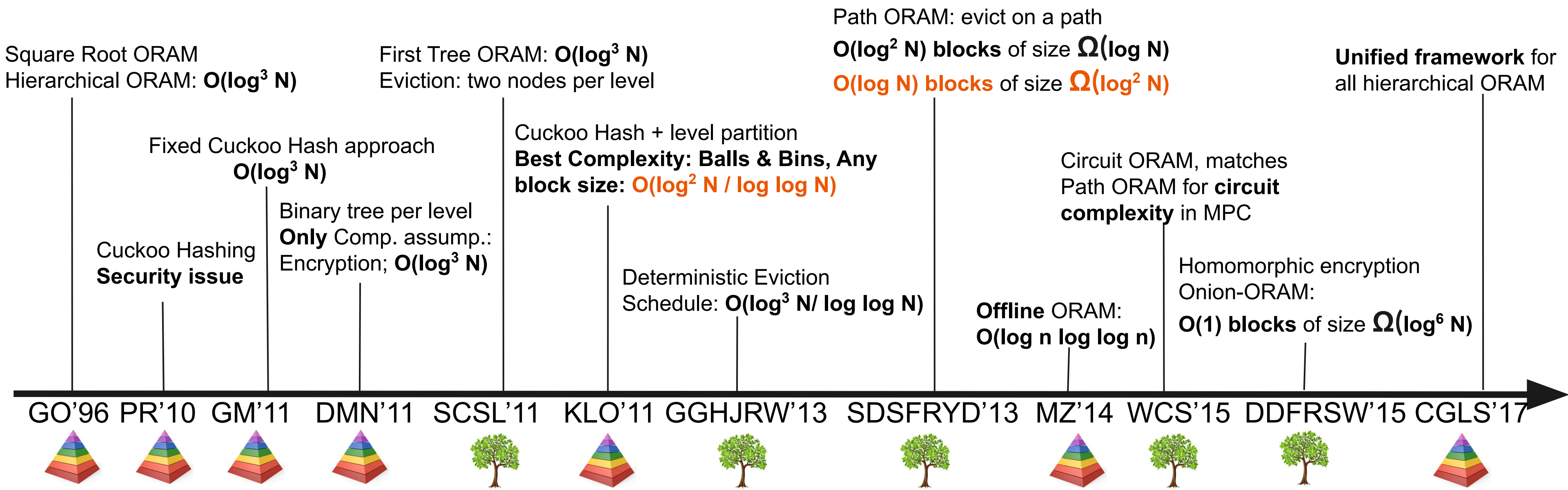


Construction Paradigms

- Hierarchical ORAMs
  - Worst case ≠ Average case
  - Computation assumption beyond encryption in most cases

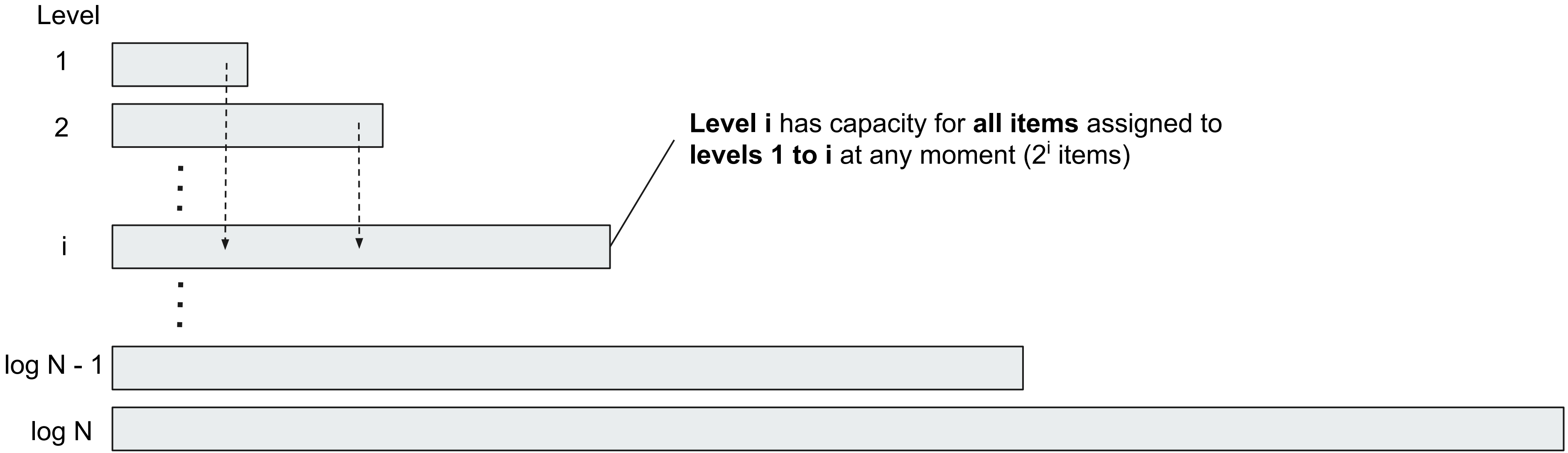
- Tree ORAMs
  - Worst case = Average case
  - Encryption - the only computational assumption

Timeline and Complexity



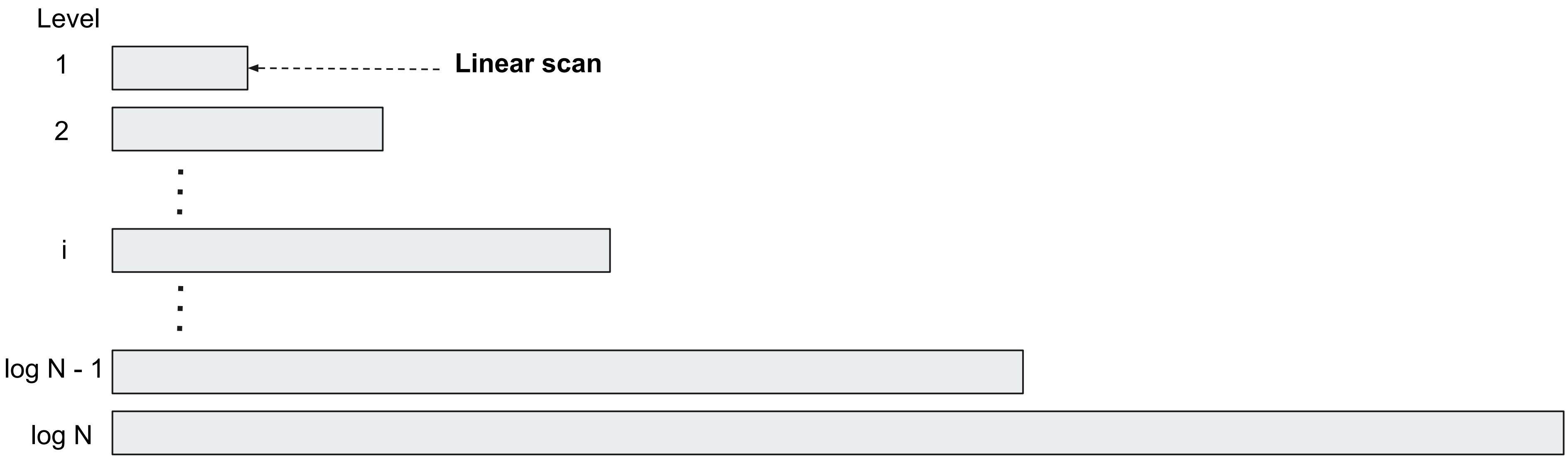
The Hierarchical ORAM Paradigm

Hierarchical Construction

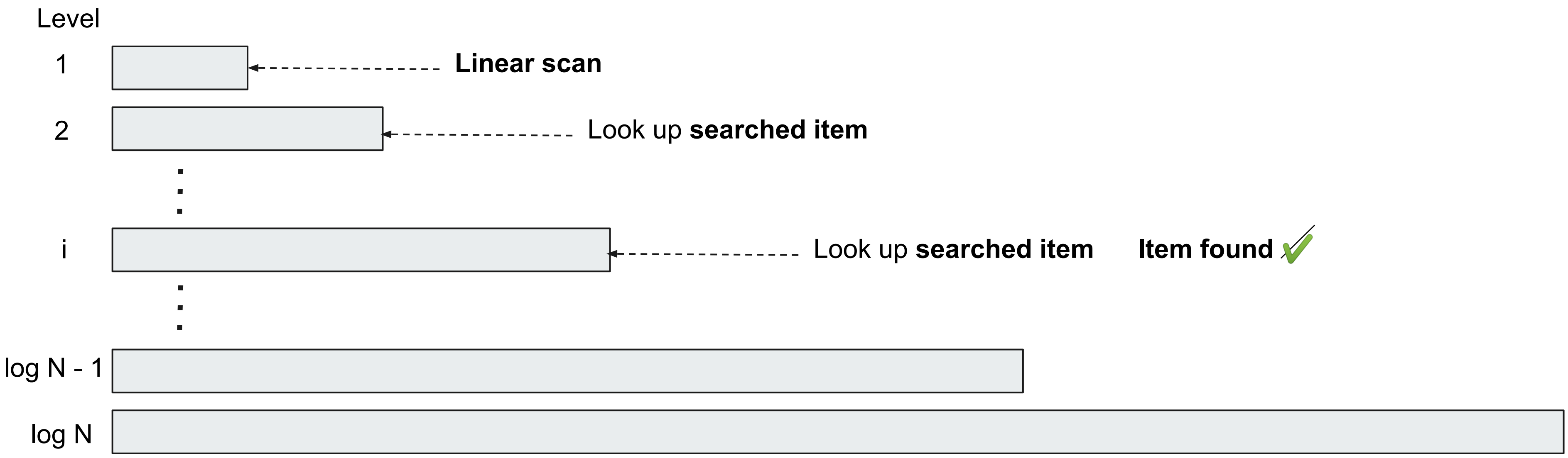


Level i has capacity for **all items** assigned to **levels 1 to i** at any moment (2^i items)

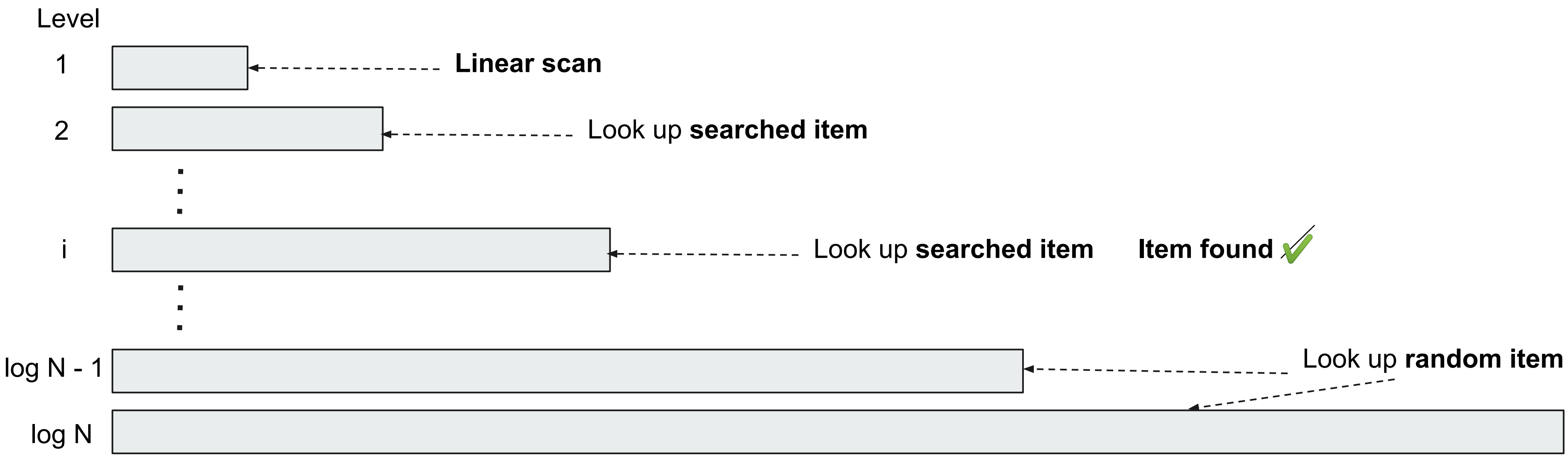
Hierarchical Construction: Search



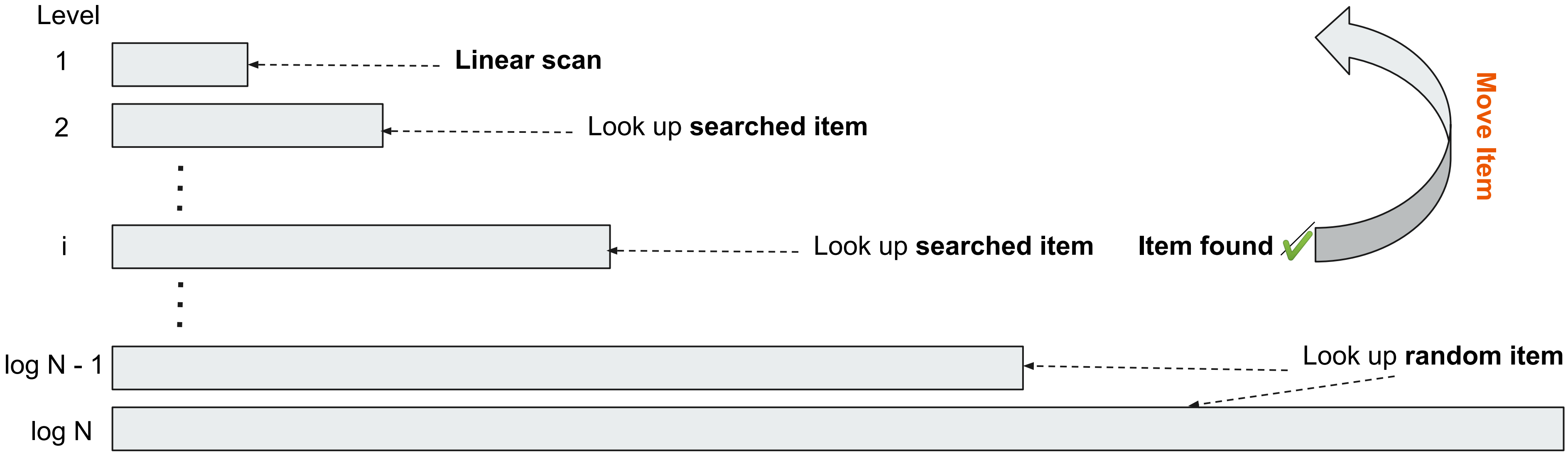
Hierarchical Construction: Search



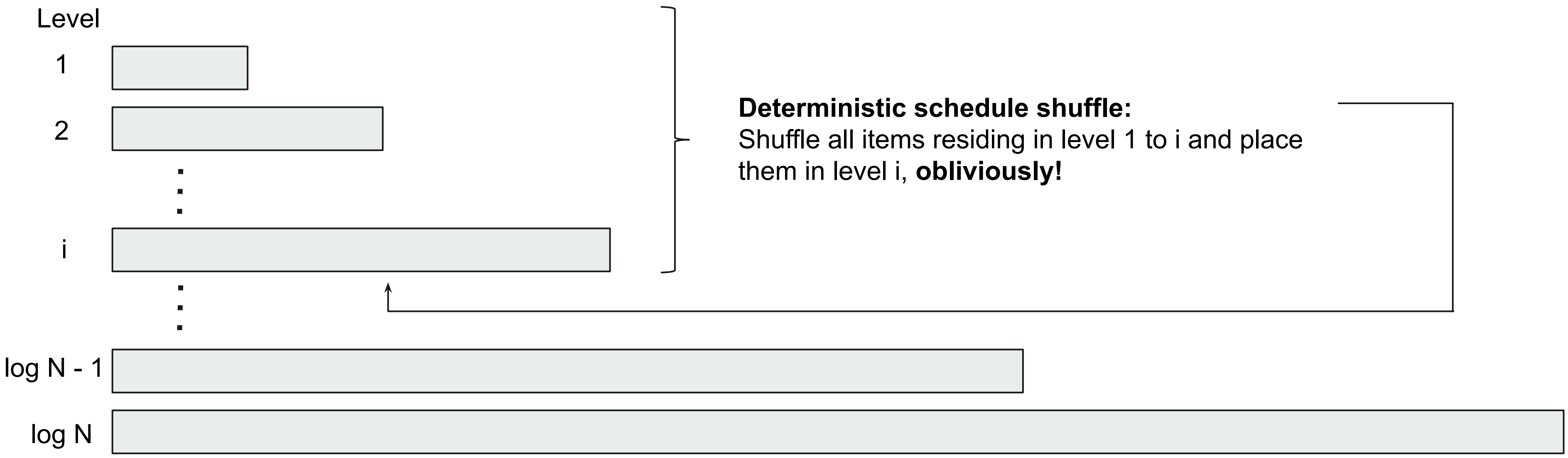
Hierarchical Construction: Search



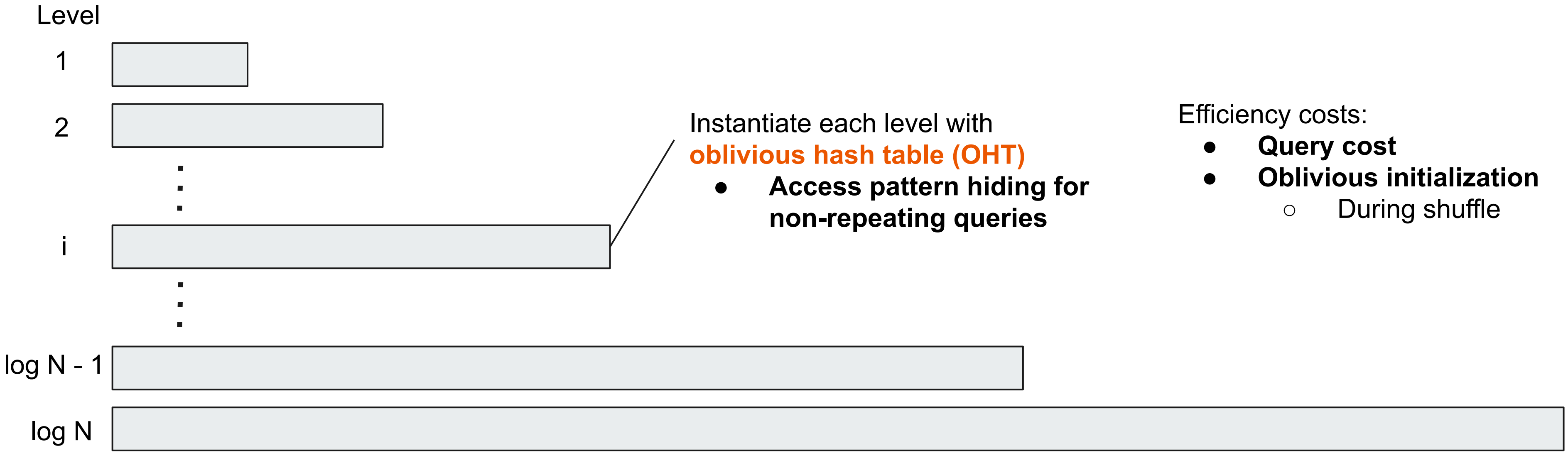
Hierarchical Construction: Search



Hierarchical Construction



Oblivious Hash Table [CGLS'17]



Instantiate each level with **oblivious hash table (OHT)**

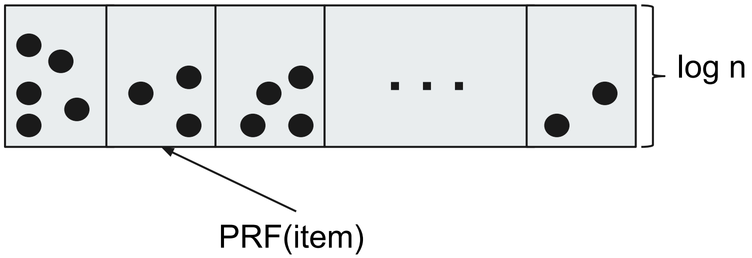
- **Access pattern hiding for non-repeating queries**

Efficiency costs:

- **Query cost**
- **Oblivious initialization**
  - During shuffle

Existing Oblivious Hash Table Constructions

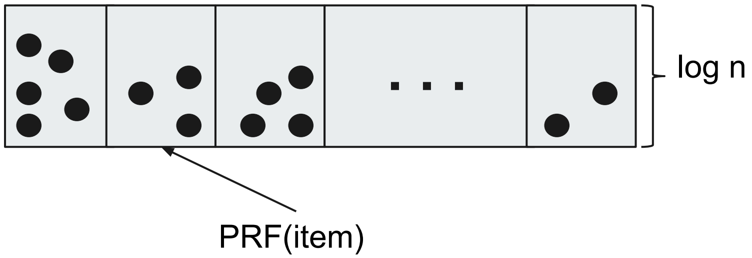
Existing Oblivious Hash Table Constructions



GO'96:

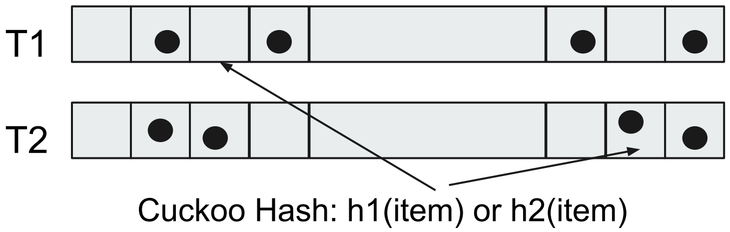
- Use pseudo-random function to match items to level buckets
- Query: retrieve item bucket **O(log n)**
- Oblivious initializations: **several oblivious sorts O(n log n)**

Existing Oblivious Hash Table Constructions



GO’96:

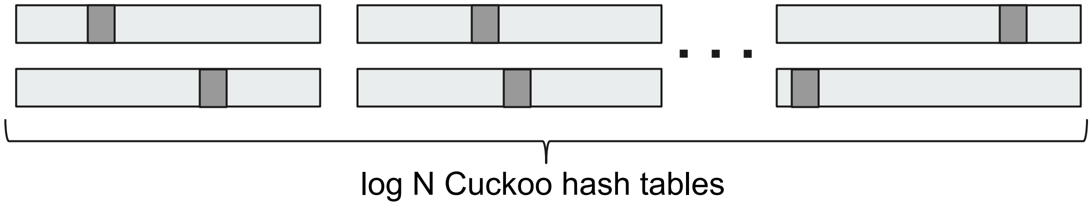
- Use pseudo-random function to match items to level buckets
- Query: retrieve item bucket **O(log n)**
- Oblivious initializations: **several oblivious sorts O(n log n)**



GM’11:

- Cuckoo hash table
- Query: **O(1)**
- Oblivious initialization: oblivious sort **O(n log n)**

Existing Oblivious Hash Table Constructions

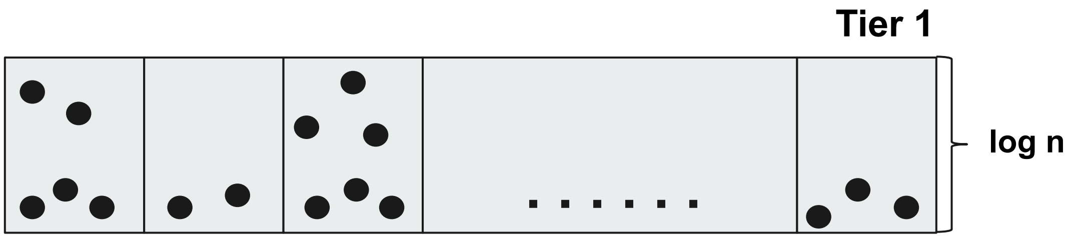


KLO'12:

- Level size n: **log n** Cuckoo hash tables; each shuffle creates a new one
- Query all Cuckoo tables: **O(log n)**
- Oblivious initialization: log n oblivious sorts on **n/log n** items per n queries: **O(log n)**

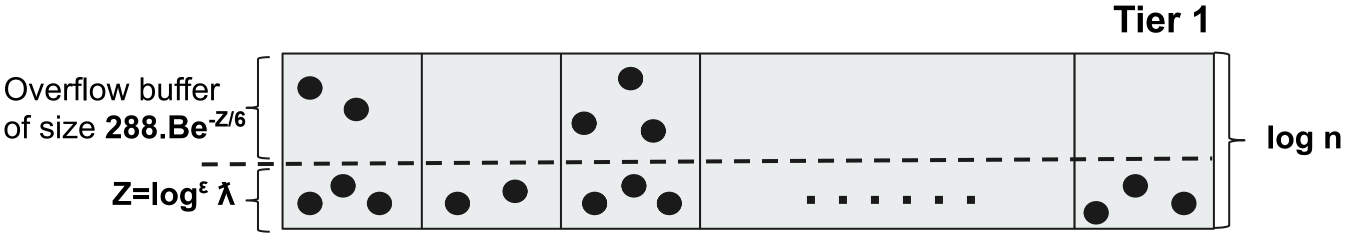
Oblivious Two Tier Hash Table [CGLS'17]

Oblivious Two Tier Hash Table [CGLS'17]



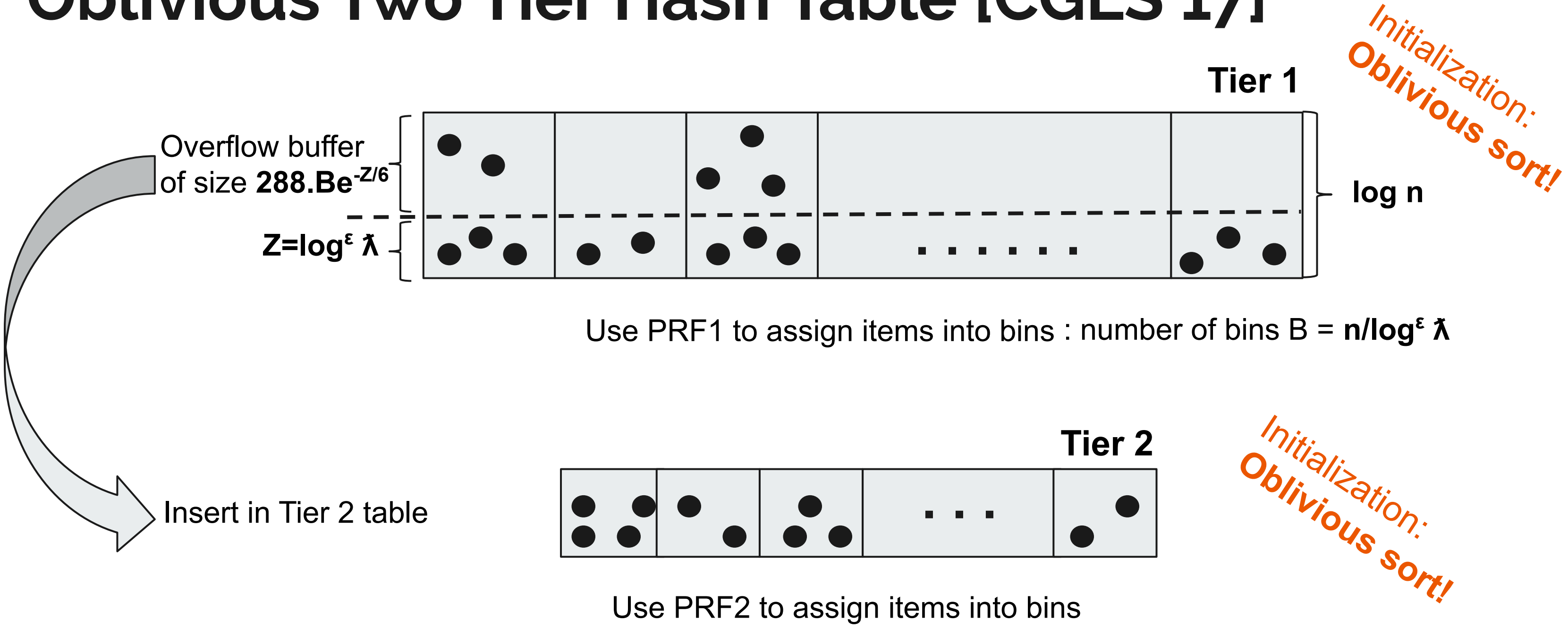
Use PRF1 to assign items into bins : number of bins B = **n/log$^{\varepsilon}$ λ**

Oblivious Two Tier Hash Table [CGLS'17]



Use PRF1 to assign items into bins : number of bins B = **n/log$^{\varepsilon}$ ƛ**

Oblivious Two Tier Hash Table [CGLS'17]



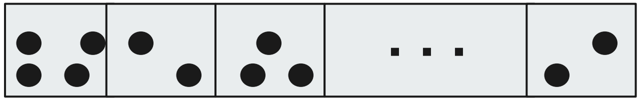
Oblivious Two Tier Hash Table [CGLS'17]

Tier 1

Retrieve bin PRF1(query)

Tier 2

If not found, Retrieve bin PRF2(query)
else, Retrieve random bin.

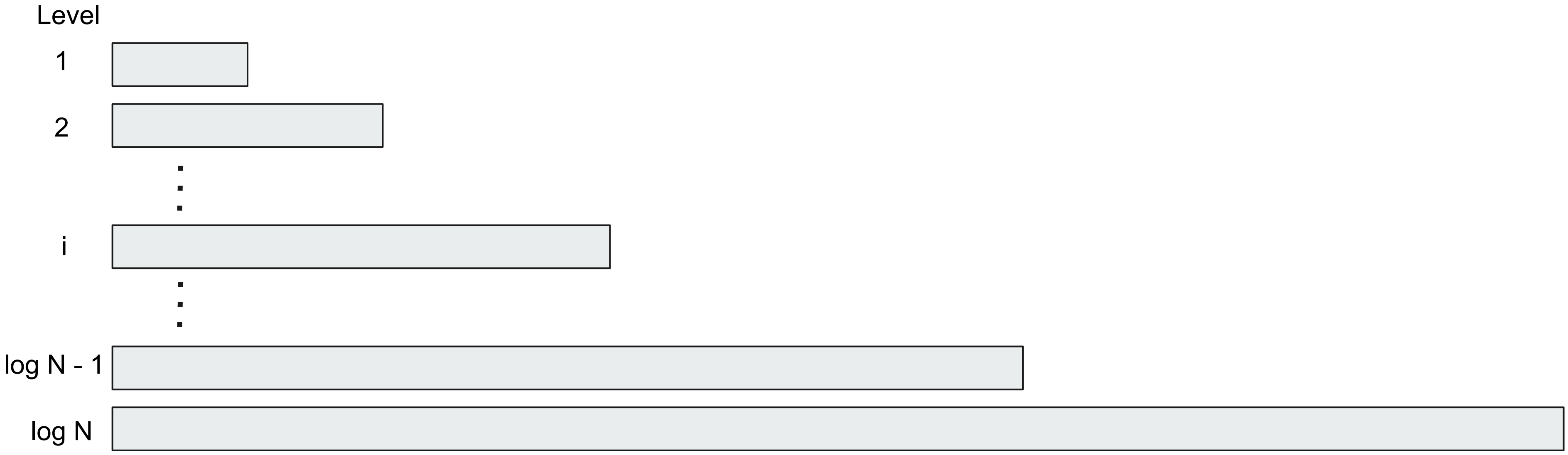


Amortized complexity: $O(\log^2 N / \log \log N)$

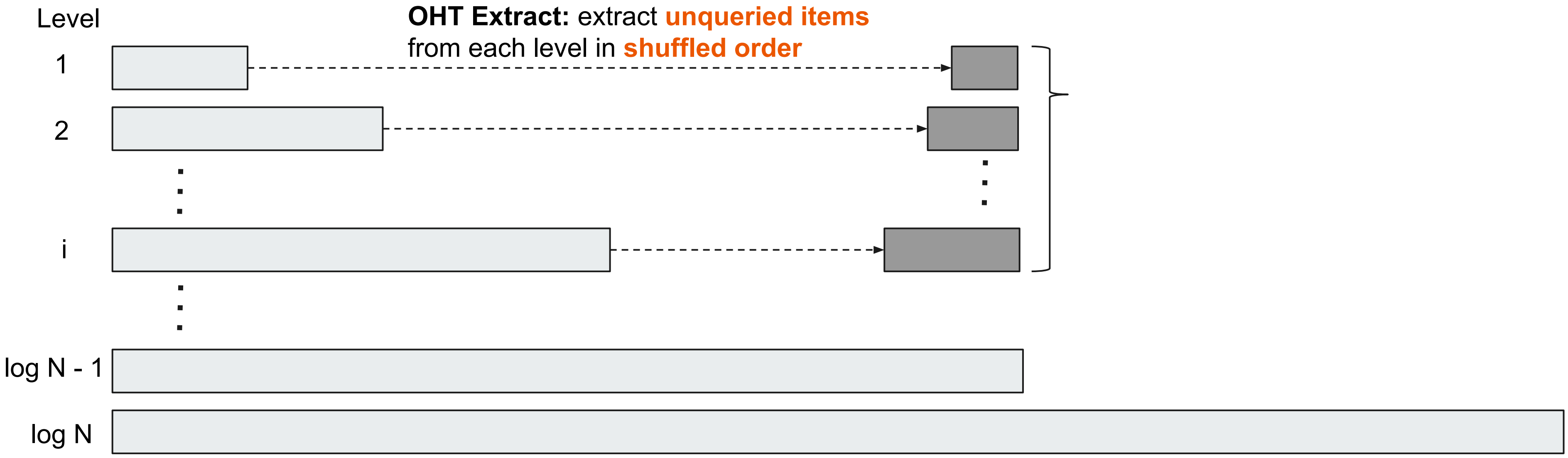
PanORAMa Overview

- ***Leverage entropy reuse to shuffle more efficiently***

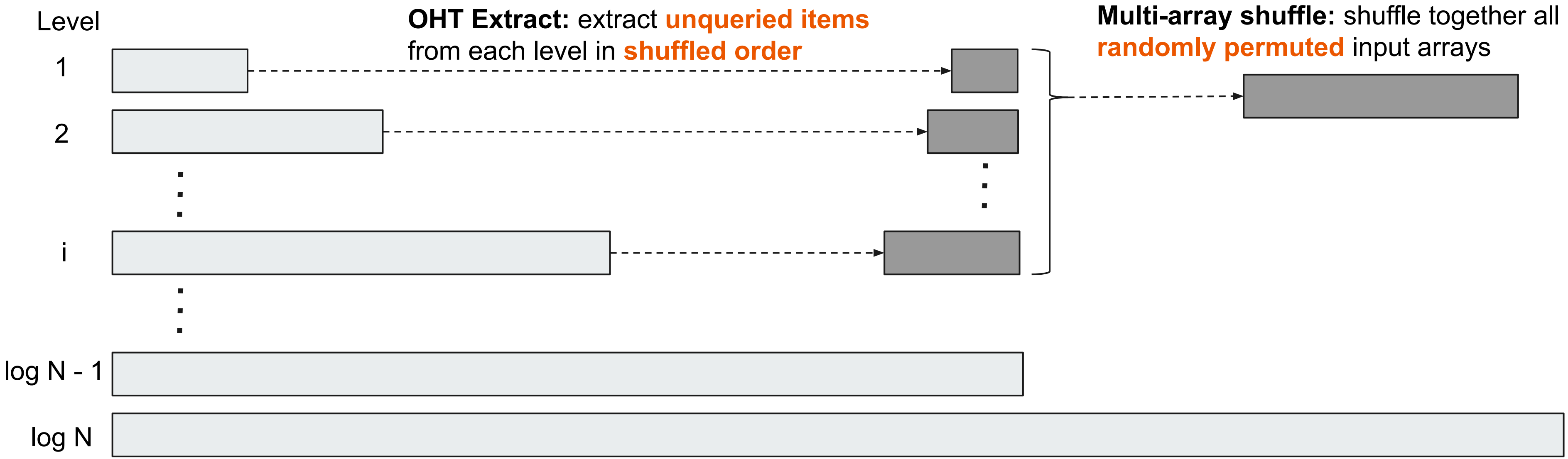
PanORAMa Hierarchical Construction



PanORAMa Hierarchical Construction

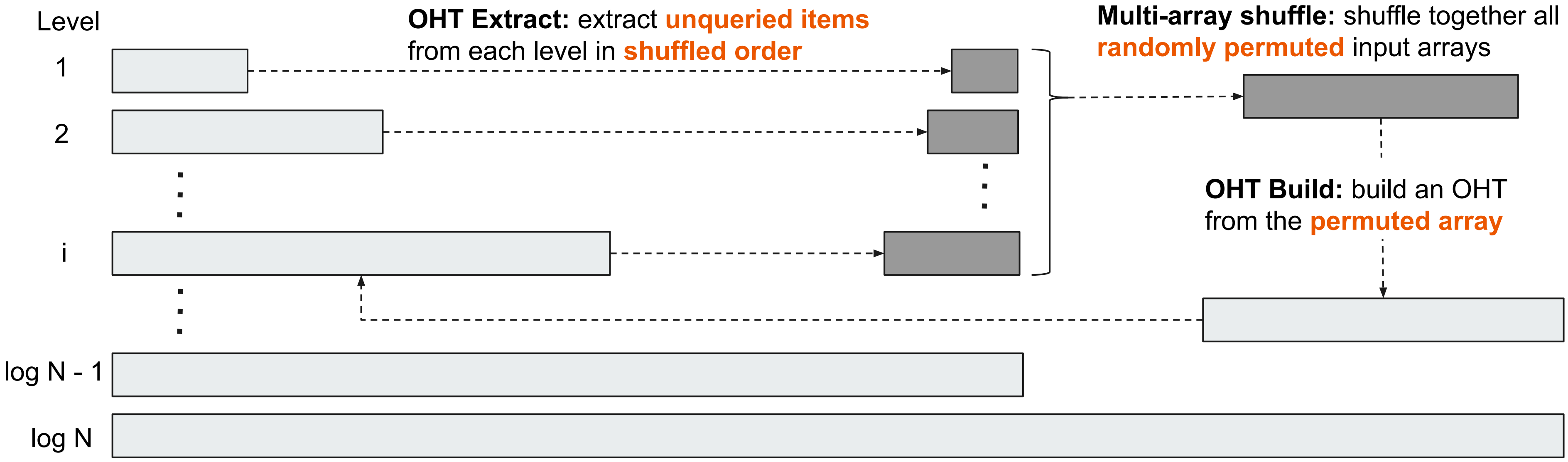


PanORAMa Hierarchical Construction



PanORAMa Hierarchical Construction

No Oblivious Sorting on a whole large level!



Oblivious Hash Table

- ***Oblivious initialization in o(n log n) leveraging input entropy***

Oblivious Hash Table

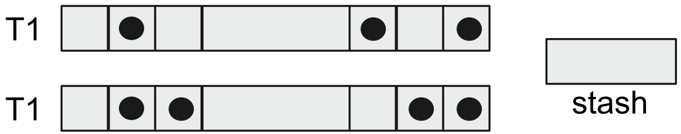
- **Definition.** Oblivious Hash Table (OHT):
    - **OHT.Init** - permutes input
    - **OHT.Build** - builds OHT from permuted input
    - **OHT.Lookup** - execute a query
    - **OHT.Extract** - extracts an array that contains unqueried item in random order
- **Security.**
    - **Access hiding: non-repeating** query sequences
    - Extract output **indistinguishable from random permutation**

Oblivious Hash Table

- **Definition.** Oblivious Hash Table (OHT):
  - **OHT.Init** - permutes input
  - **OHT.Build** - builds OHT from permuted input
  - **OHT.Lookup** - execute a query
  - **OHT.Extract** - extracts an array that contains unqueried item in random order
- **Security.**
  - **Access hiding: non-repeating** query sequences
  - Extract output **indistinguishable from random permutation**
- **Oblivious bin** = mini OHT
  - OHT that is instantiated on small input size **O(polylog N)**
  - We **can use oblivious sorting** without hurting efficiency

Oblivious Bin

- Oblivious Cuckoo Bin

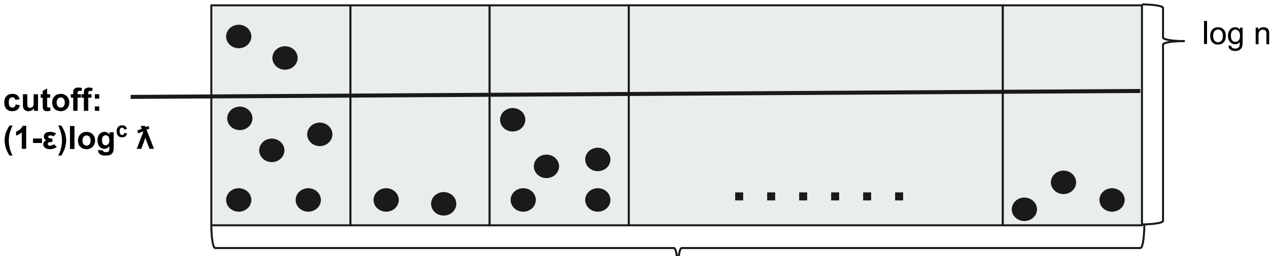


- Cuckoo hash
- Oblivious sort to build and extract
  - Add **n dummies** in Build
  - Extract n items in Extract

- "Dynamic" Bin
  - Items need to be added continuously in non-amortized manner
  - Smallest ORAM level
  - Size: $O(\log^7 n)$
  - Use existing oblivious ram constructions, e.g. Goodrich, Mitzenmacher [GM'11]

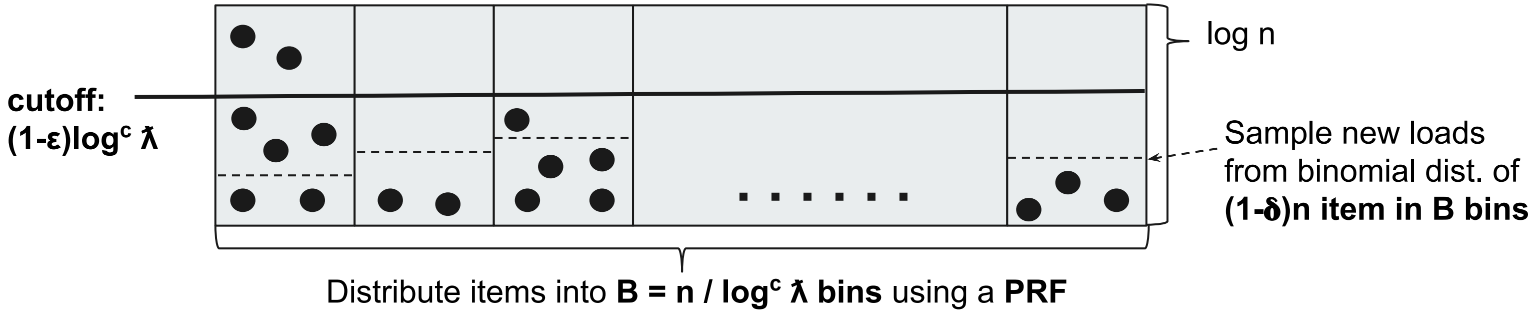
Oblivious Hash Table

Oblivious Hash Table



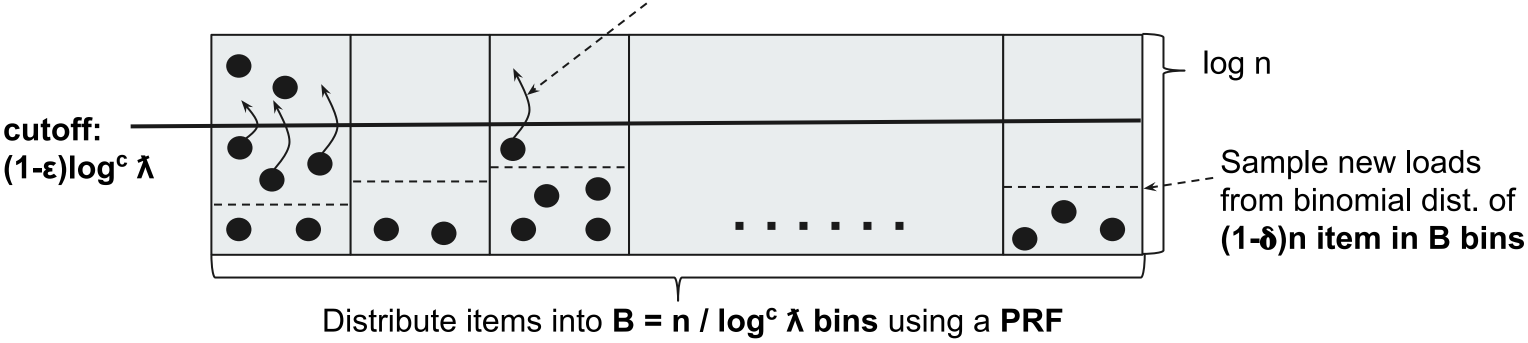
Distribute items into **B = n / log^c ƛ bins** using a **PRF**

Oblivious Hash Table



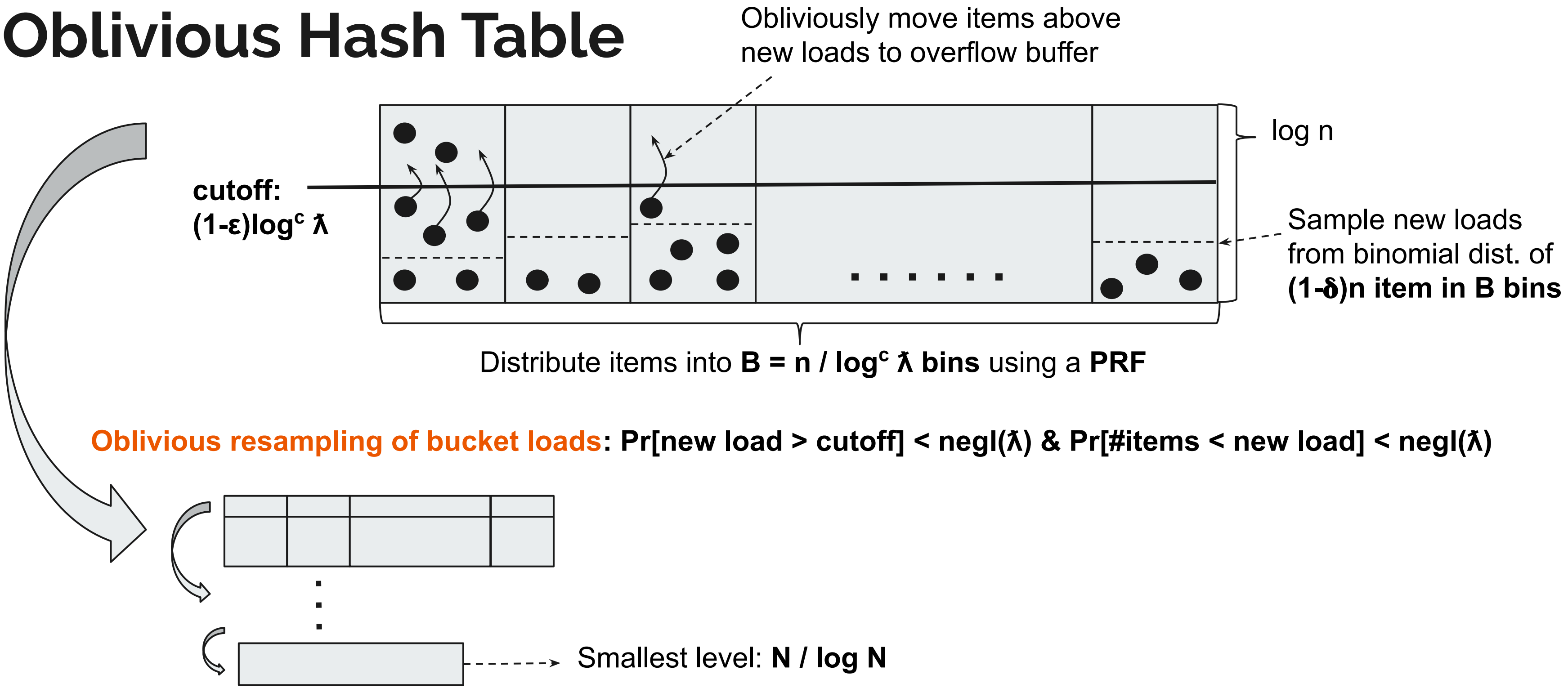
Oblivious resampling of bucket loads: **Pr[new load > cutoff] < negl(ƛ) & Pr[#items < new load] < negl(ƛ)**

Oblivious Hash Table

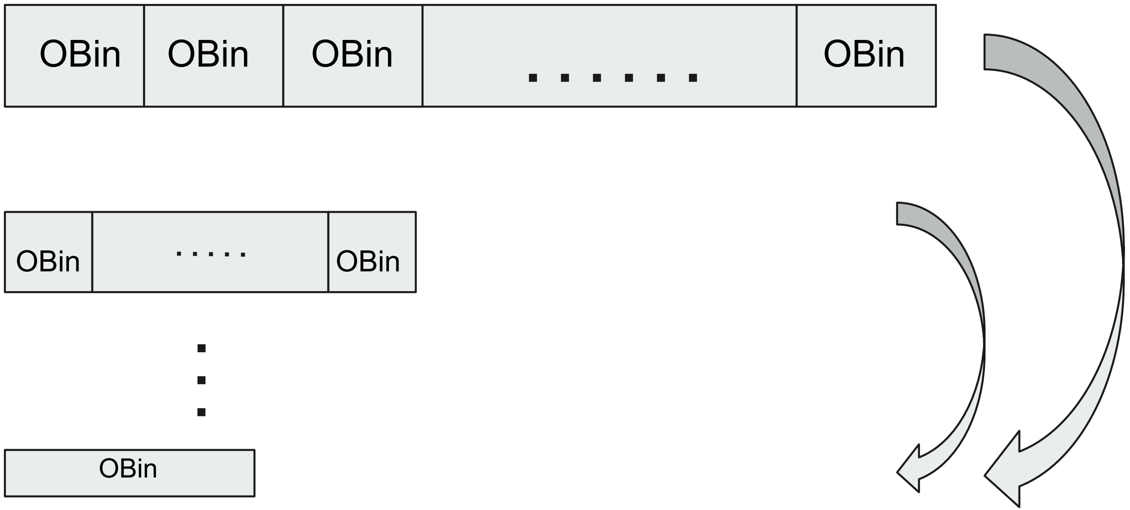


Oblivious resampling of bucket loads: **Pr[new load > cutoff] < negl(λ) & Pr[#items < new load] < negl(λ)**

Oblivious Hash Table



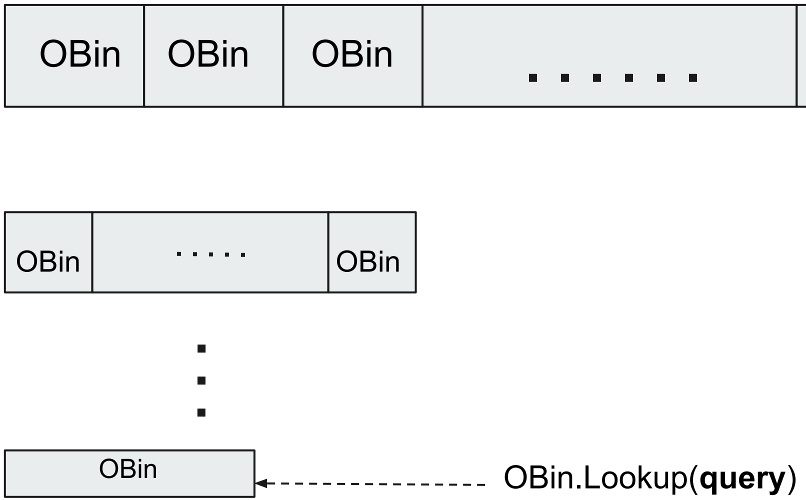
Oblivious Hash Table: Create Oblivious Bins



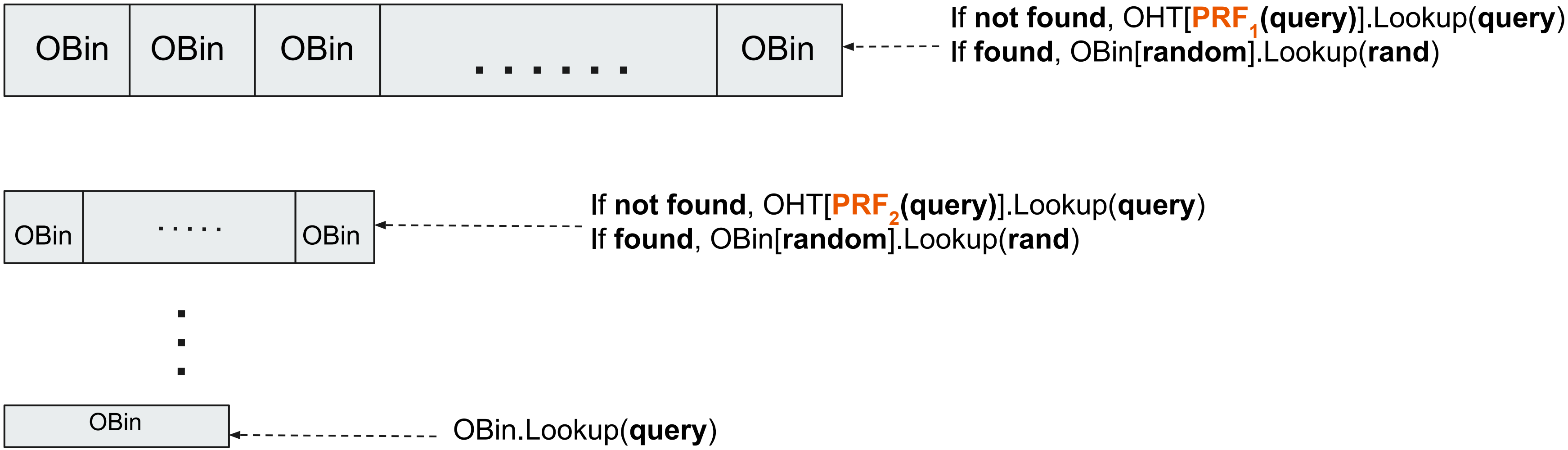
Items from Cuckoo stash are

- (encrypted) indexed with their source bucket
- merged with items used to instantiate the last level

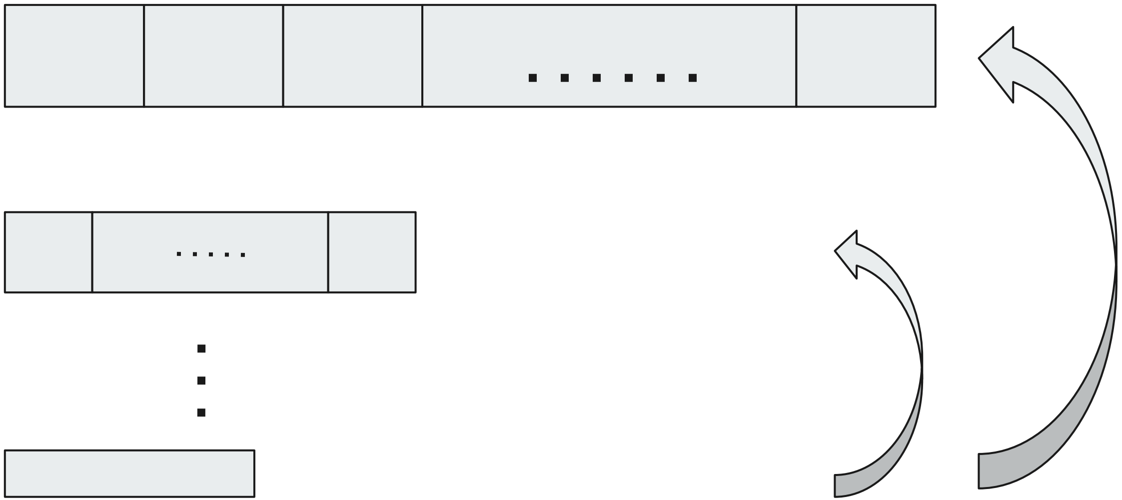
Oblivious Hash Table: Query



Oblivious Hash Table: Query

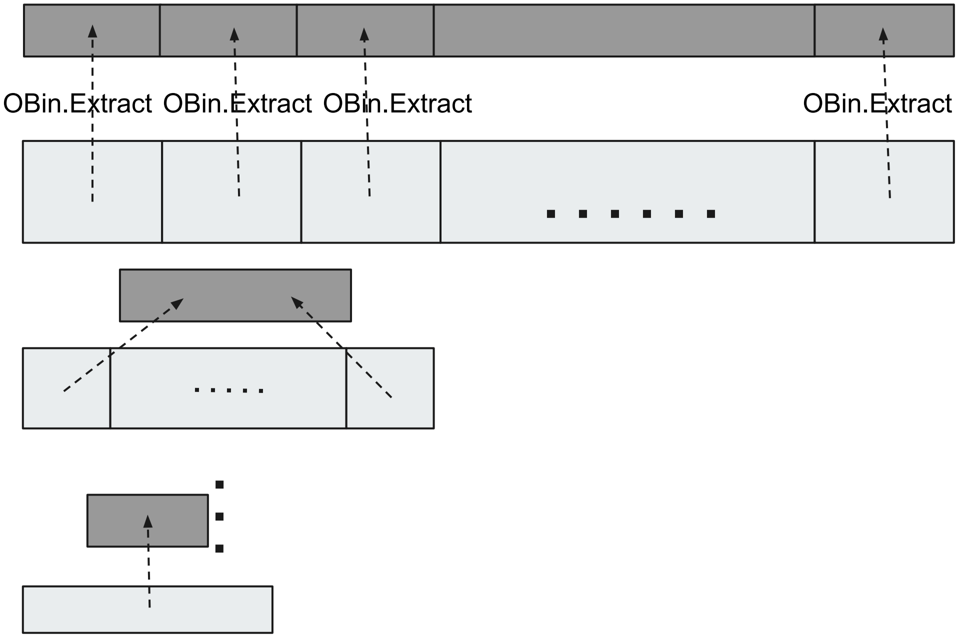


Oblivious Hash Table: Extract

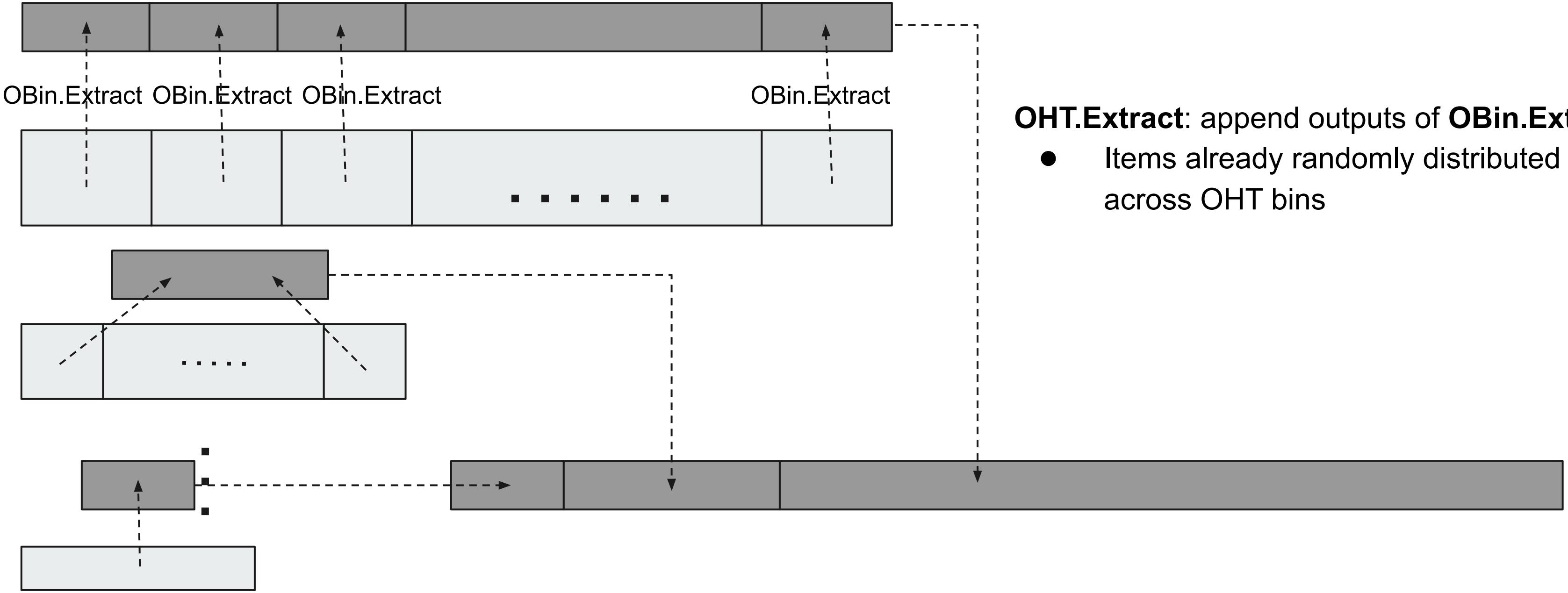


Move items with origin in the Cuckoo stashes back to their corresponding levels

Oblivious Hash Table: Extract



Oblivious Hash Table: Extract



OHT.Extract: append outputs of **OBin.Extract**

- Items already randomly distributed across OHT bins

Oblivious Hash Table

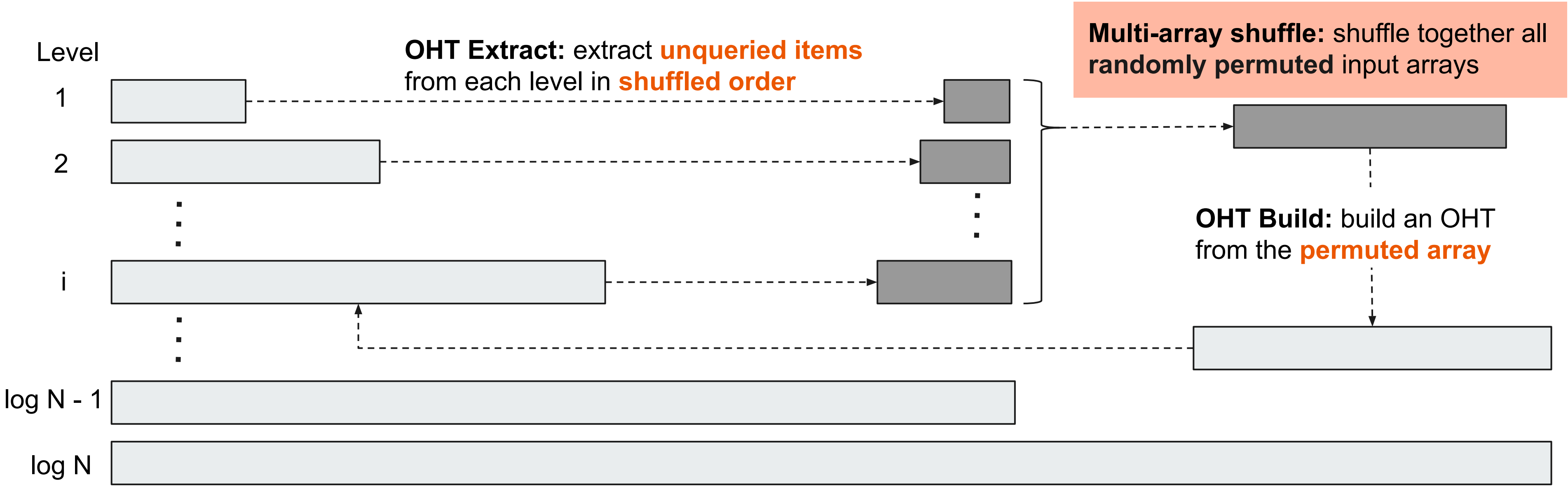
Amortized Communication Complexity over N accesses for OHT on N items:

- OCuckooBin: **$O(\log N + (\log\log \lambda))$**

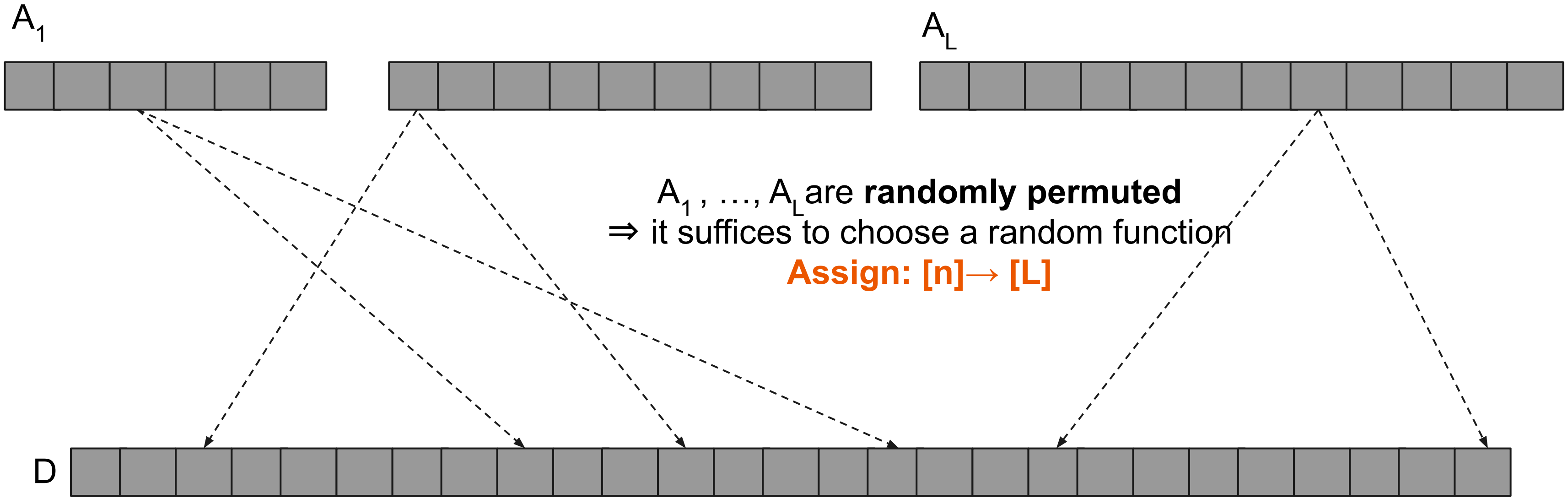
Oblivious Multi-Array Shuffle

- *Random shuffle in o(n log n) leveraging input entropy: independently sorted input arrays*

PanORAMa Hierarchical Construction



Oblivious Multi-Array Shuffle

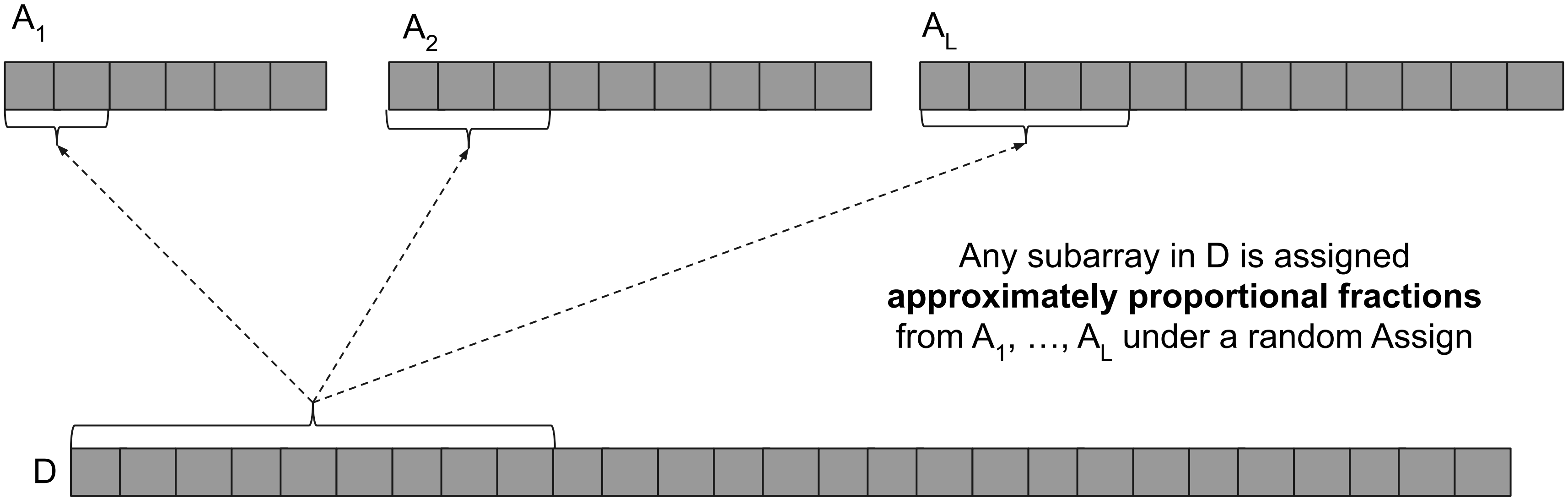


$A_1, \ldots, A_L$ are **randomly permuted**

⇒ it suffices to choose a random function

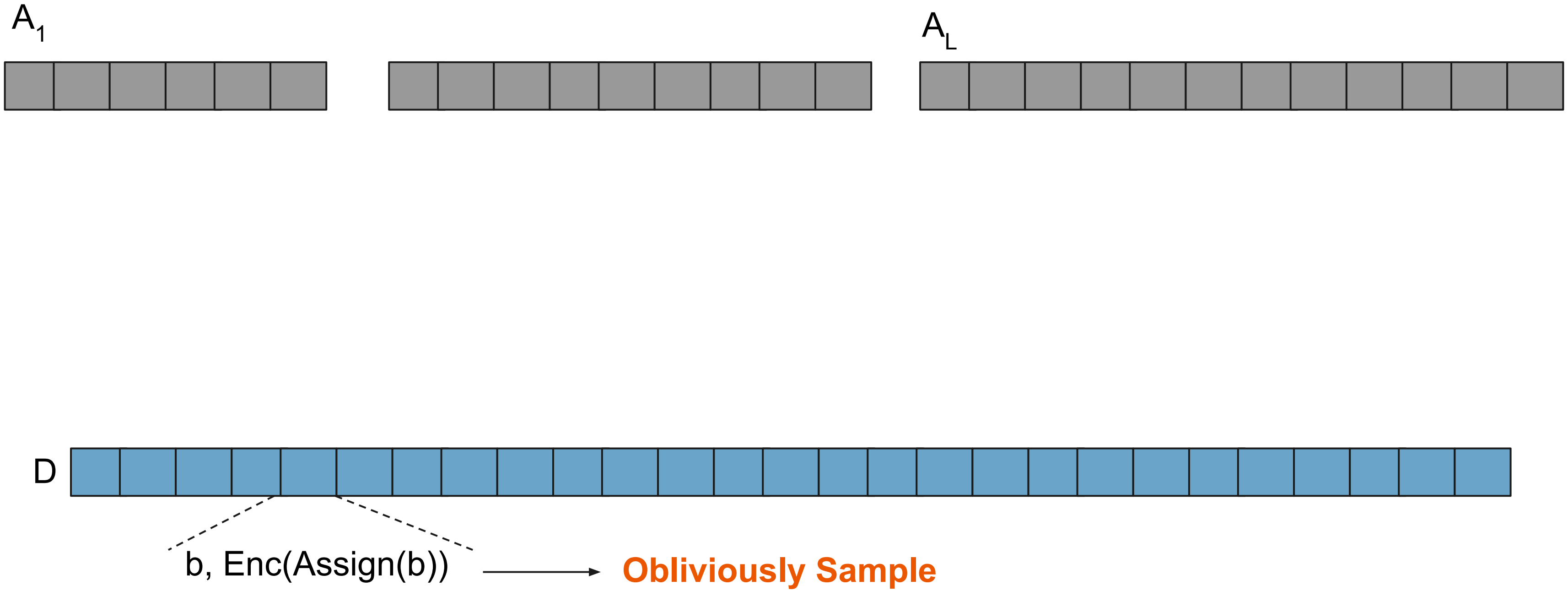
Assign: $[n] \rightarrow [L]$

Oblivious Multi-Array Shuffle



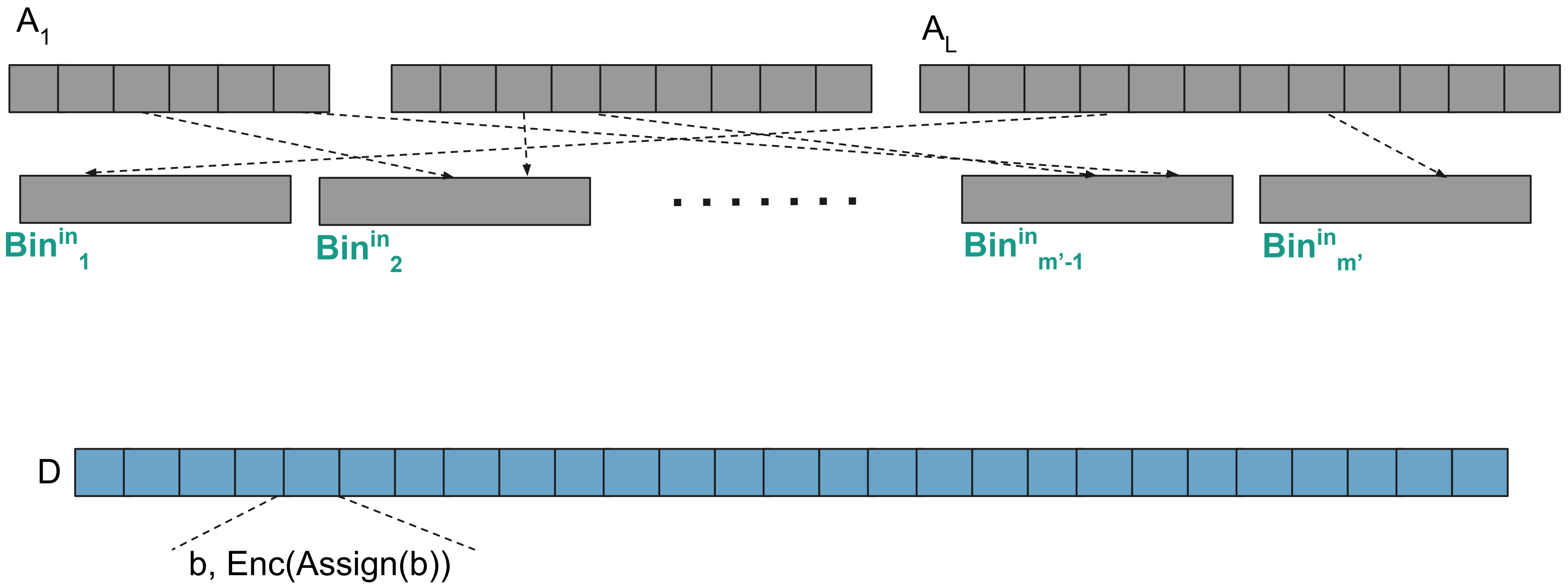
Any subarray in D is assigned **approximately proportional fractions** from A_1, …, A_L under a random Assign

Oblivious Multi-Array Shuffle

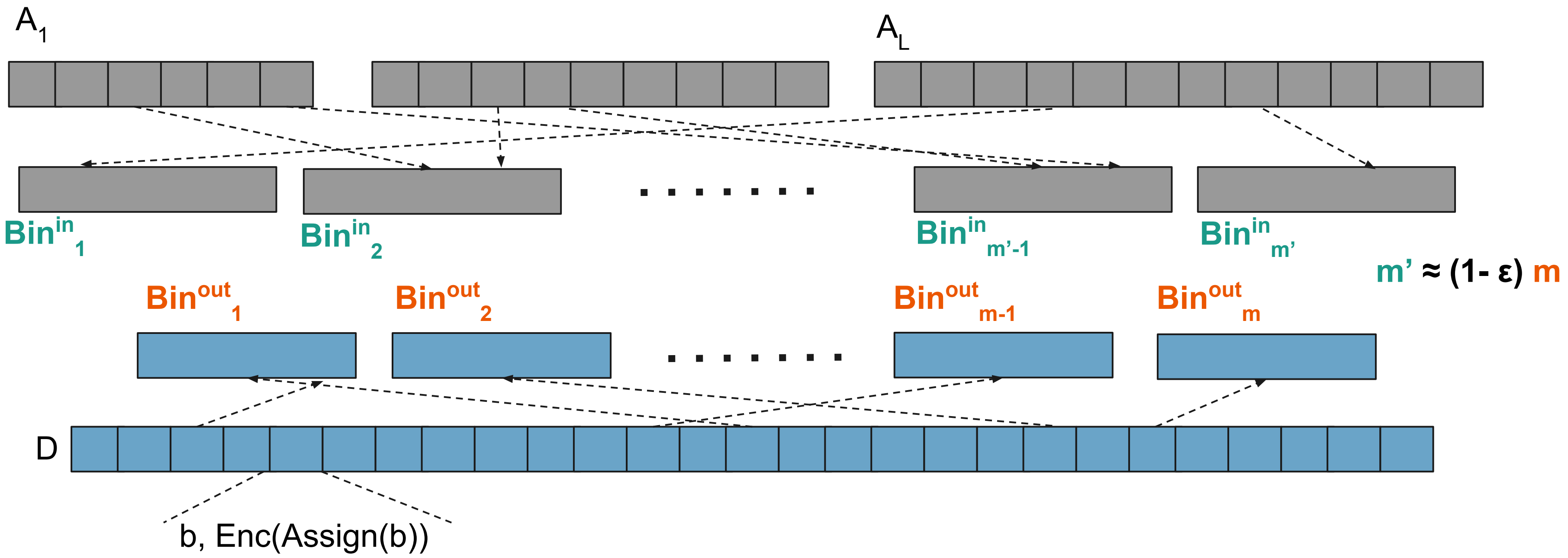


Obliviously Sample

Oblivious Multi-Array Shuffle



Oblivious Multi-Array Shuffle



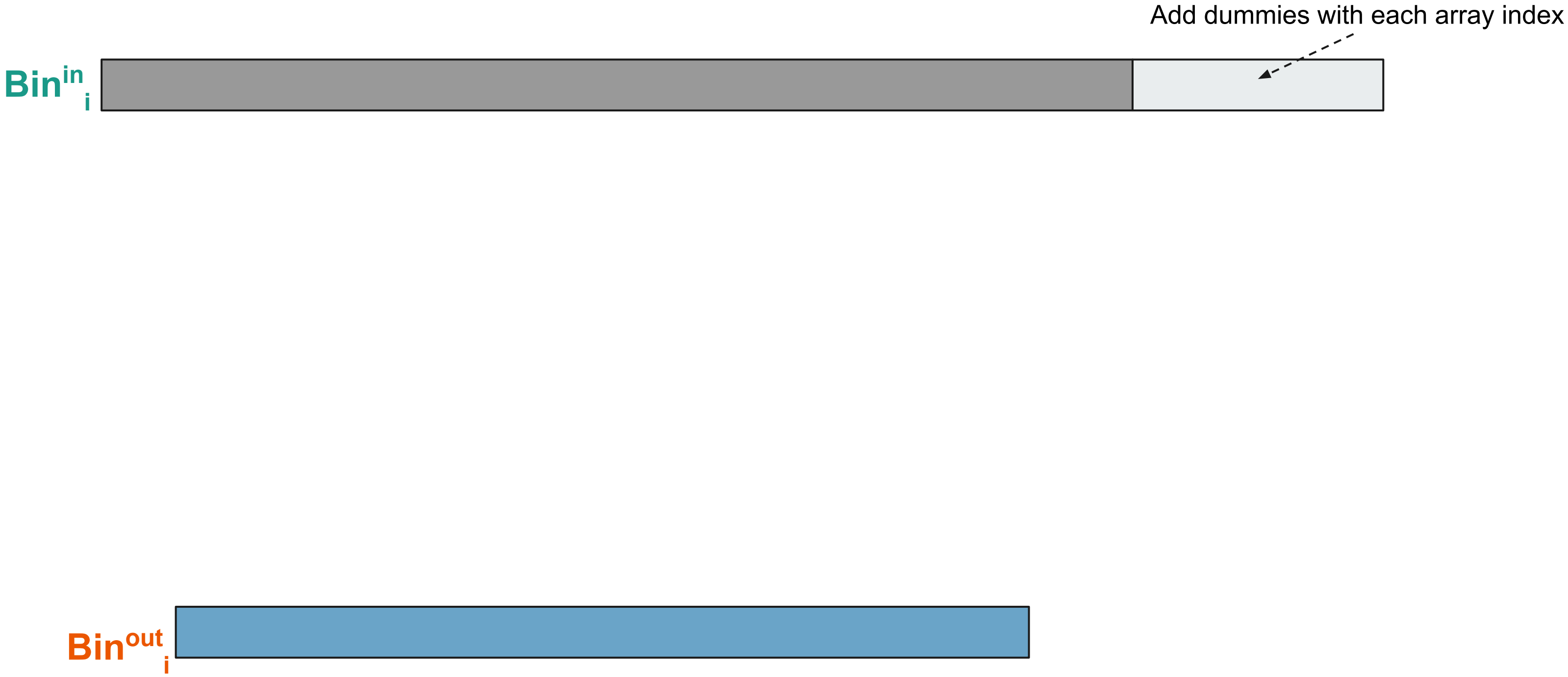
Bin Shuffle

Bin^{in}_i

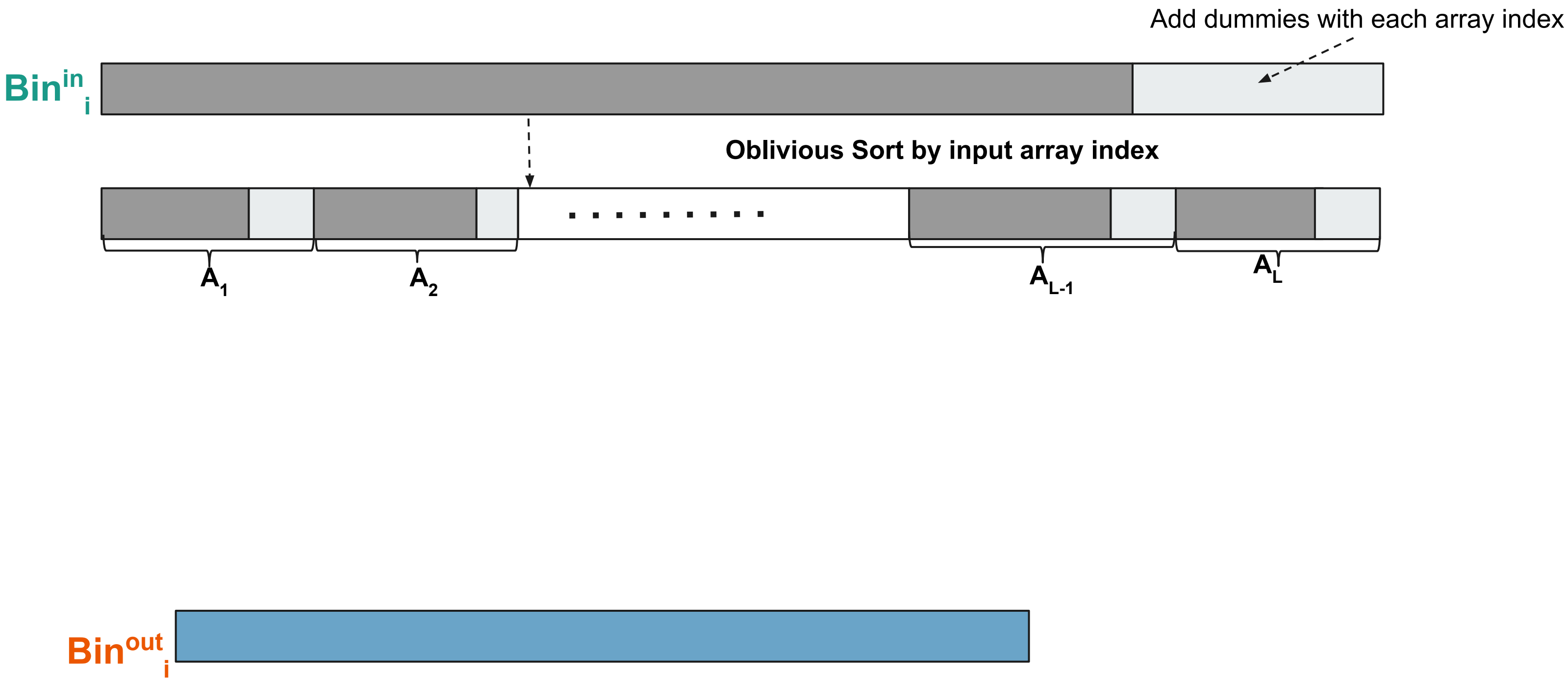
For each j in [L], Bin^{in}_i **contains more elements** from $\mathbf{A_j}$ than the **number of elements from** $\mathbf{A_j}$ **assigned** to Bin^{out}_i

Bin^{out}_i

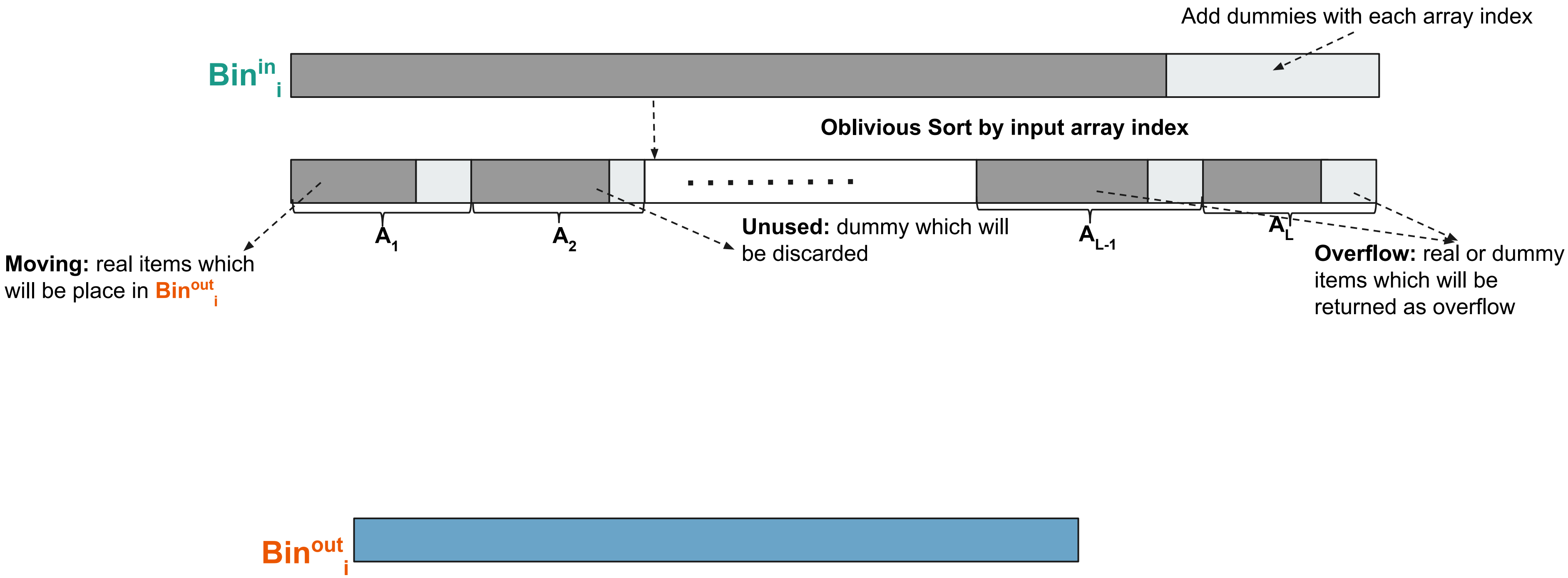
Bin Shuffle



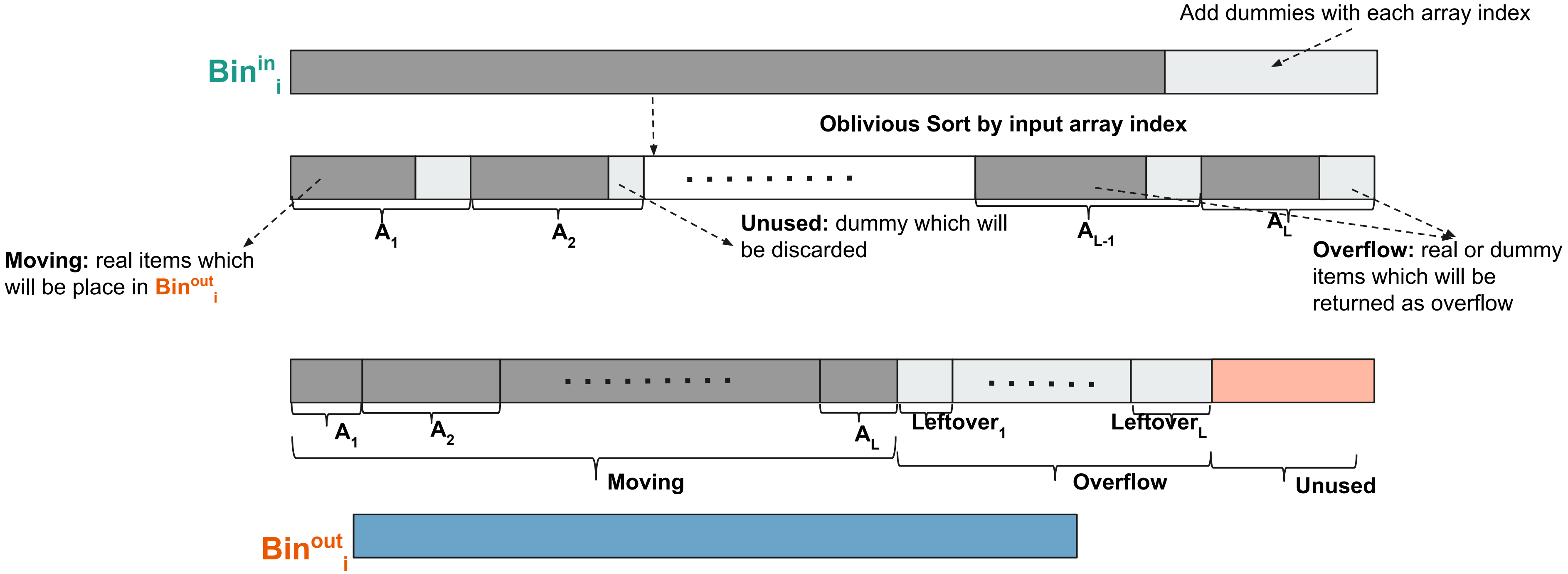
Bin Shuffle


Add dummies with each array index
Bin^{in}_i
Oblivious Sort by input array index
A_1
A_2
A_{L-1}
A_L
Bin^{out}_i

Bin Shuffle

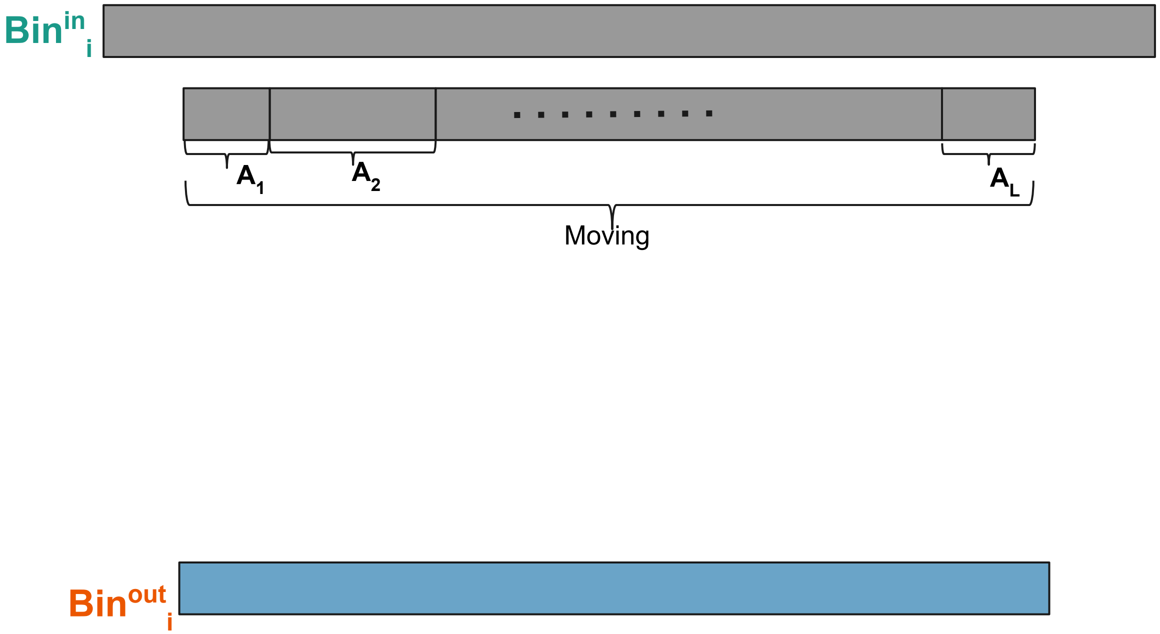

Add dummies with each array index
Bin^{in}_i
Oblivious Sort by input array index
. . . .
A_1
A_2
A_{L-1}
A_L
Moving: real items which will be place in Bin^{out}_i
Unused: dummy which will be discarded
Overflow: real or dummy items which will be returned as overflow
Bin^{out}_i

Bin Shuffle

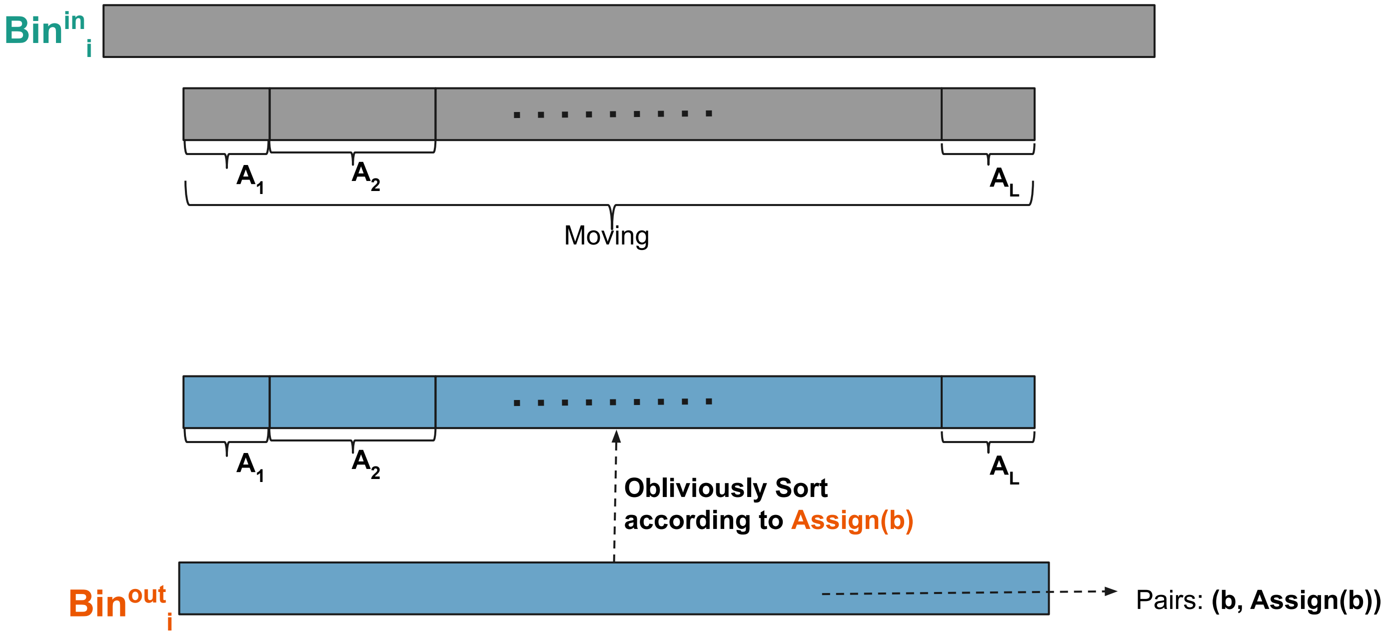


Add dummies with each array index
Bin^in_i
Oblivious Sort by input array index
A1
A2
AL-1
AL
Unused: dummy which will be discarded
Moving: real items which will be place in Bin^out_i
Overflow: real or dummy items which will be returned as overflow
A1
A2
AL
Leftover1
LeftoverL
Moving
Overflow
Unused
Bin^out_i

Bin Shuffle

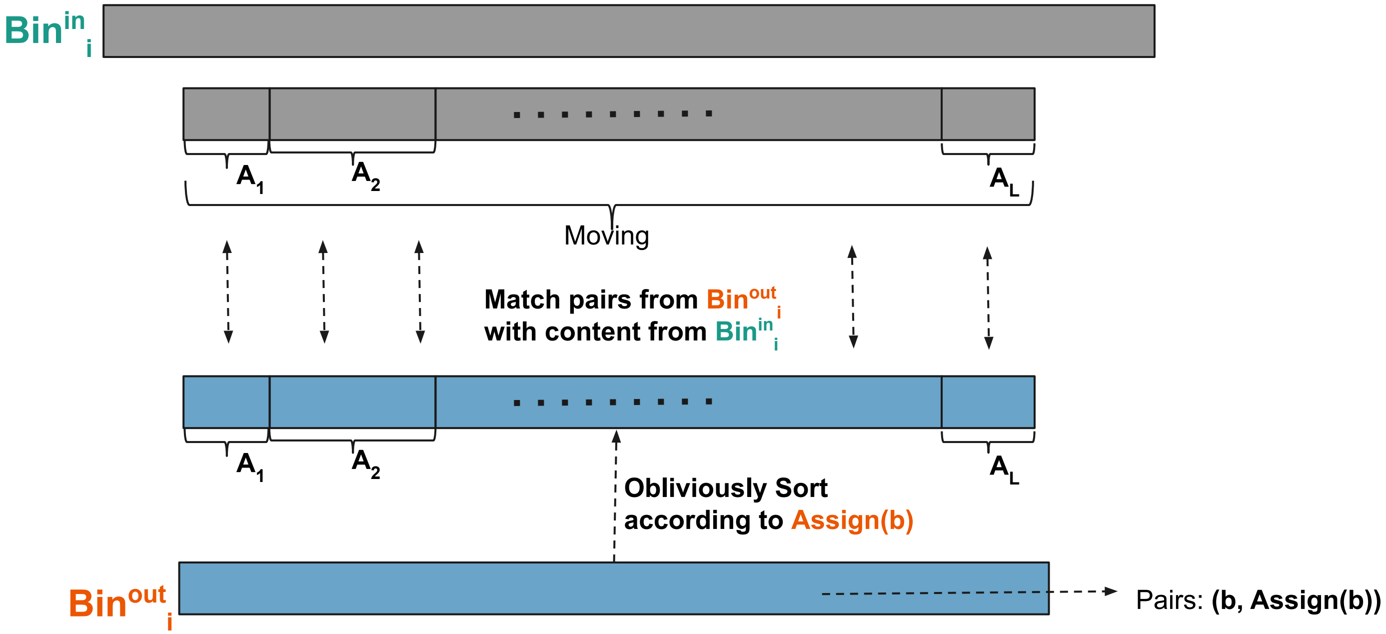


Bin Shuffle



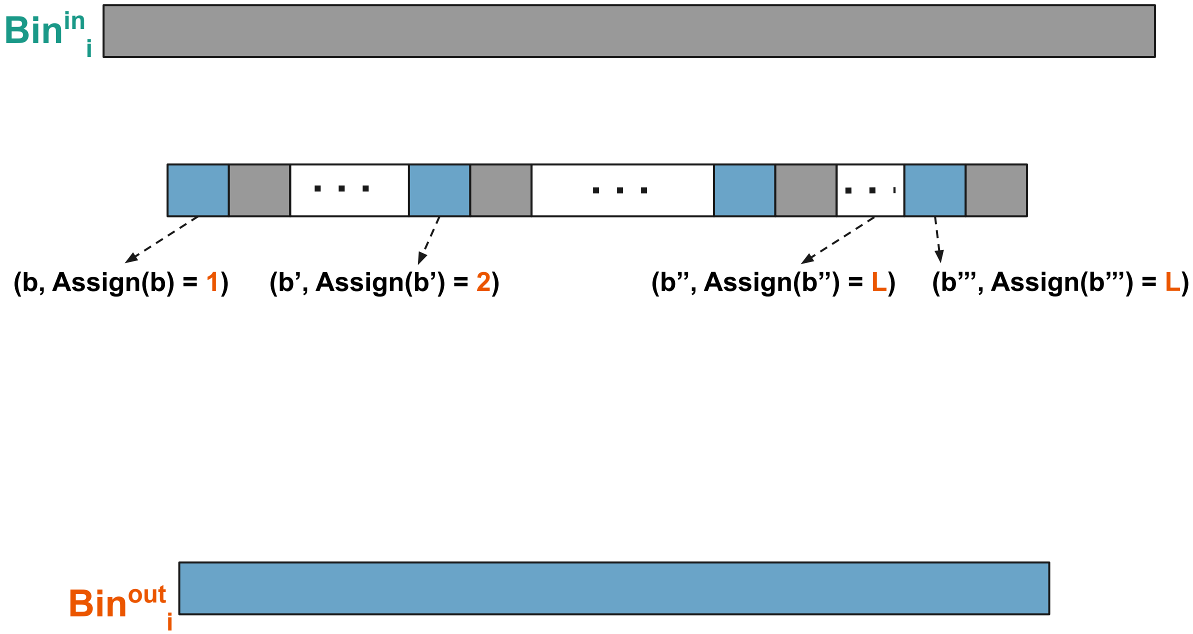
Bin^{in}_i
. . . .
A_1
A_2
A_L
Moving
. . . .
A_1
A_2
A_L
Obliviously Sort
according to Assign(b)
Bin^{out}_i
Pairs: (b, Assign(b))

Bin Shuffle

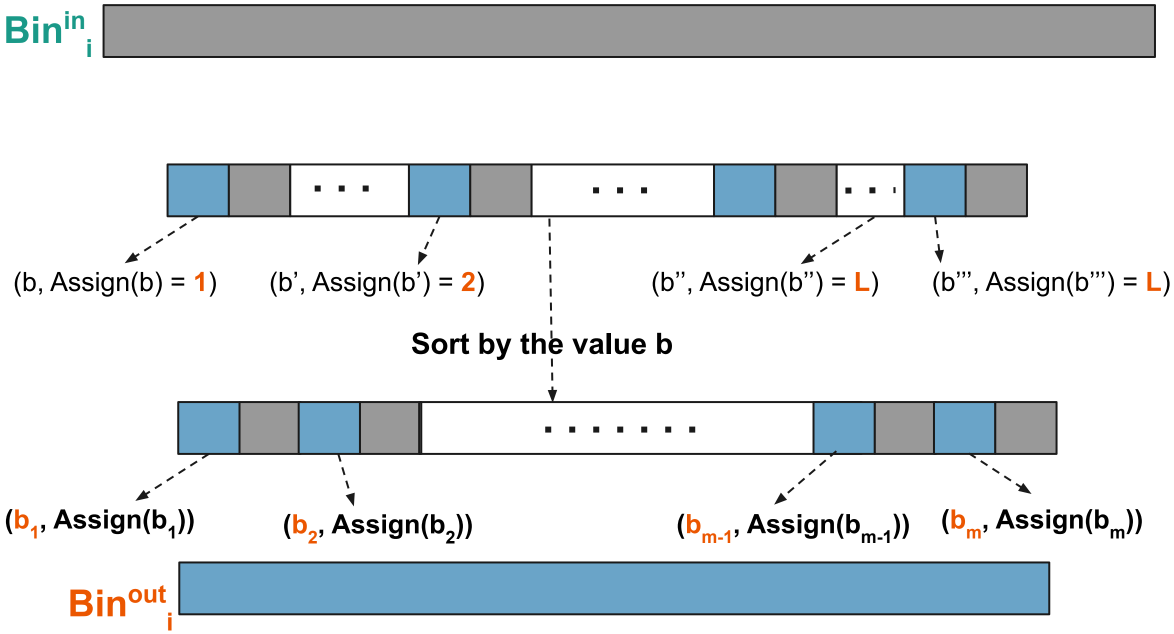


Bin^{in}_i
A_1
A_2
A_L
Moving
Match pairs from Bin^{out}_i with content from Bin^{in}_i
A_1
A_2
A_L
Obliviously Sort according to Assign(b)
Bin^{out}_i
Pairs: **(b, Assign(b))**

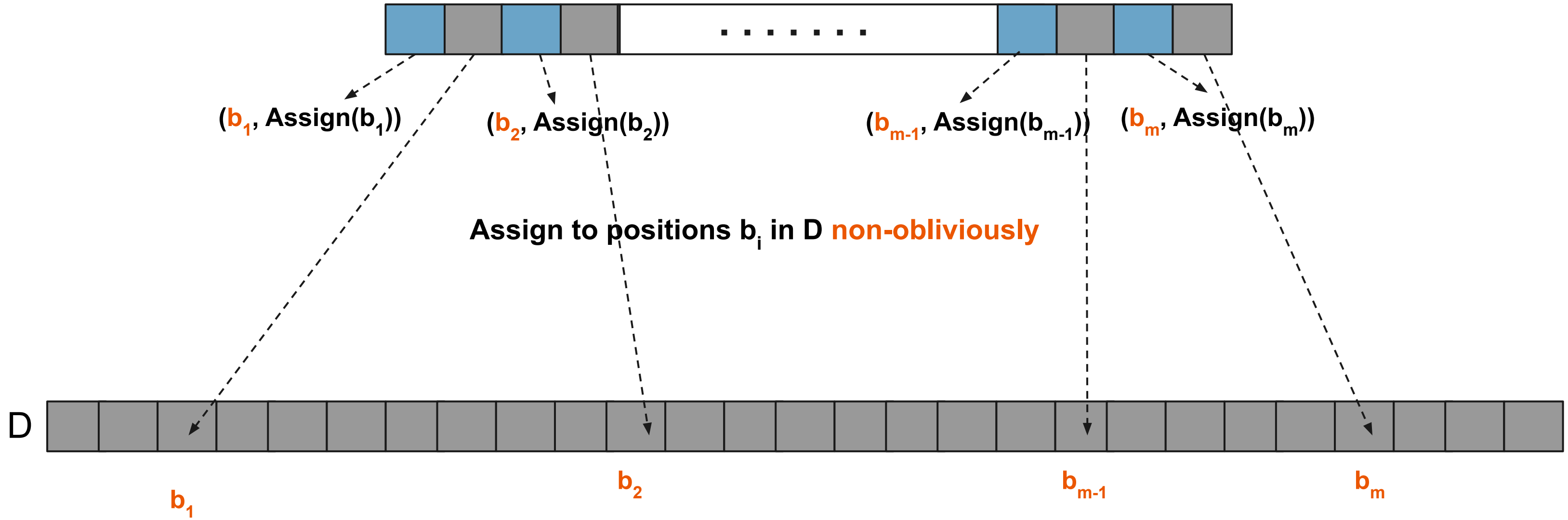
Bin Shuffle



Oblivious Multi-Array Shuffle

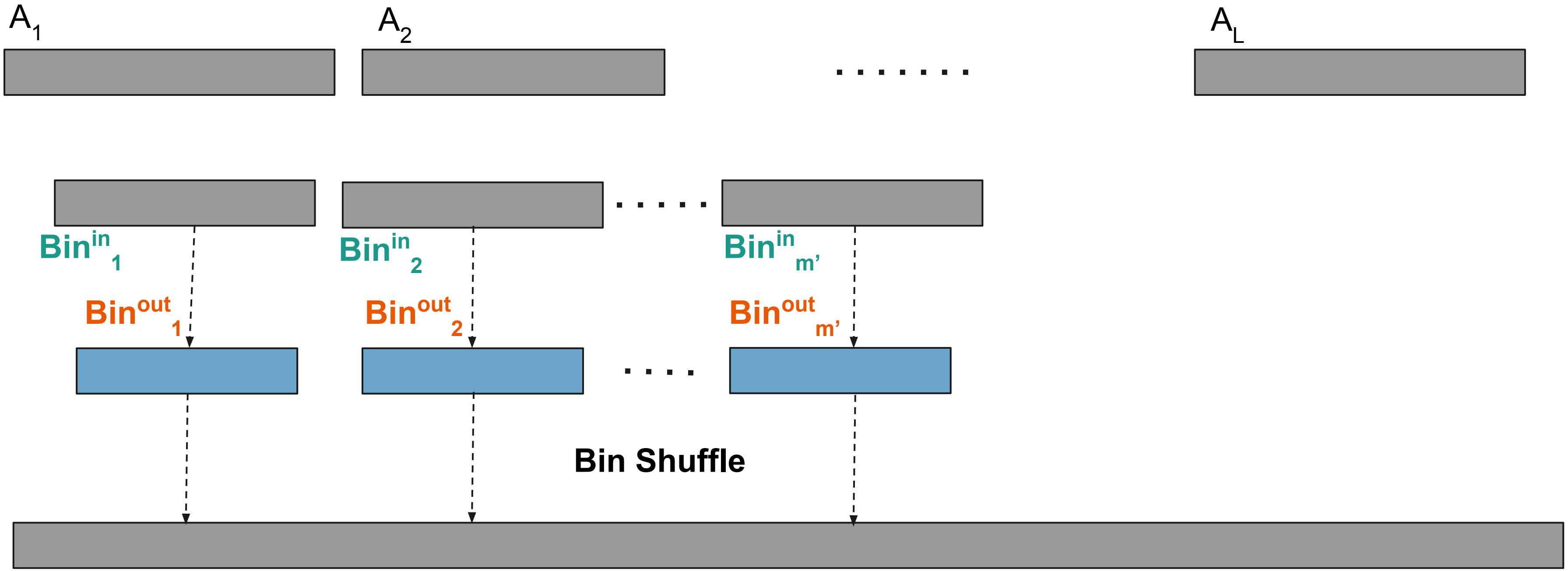


Bin Shuffle

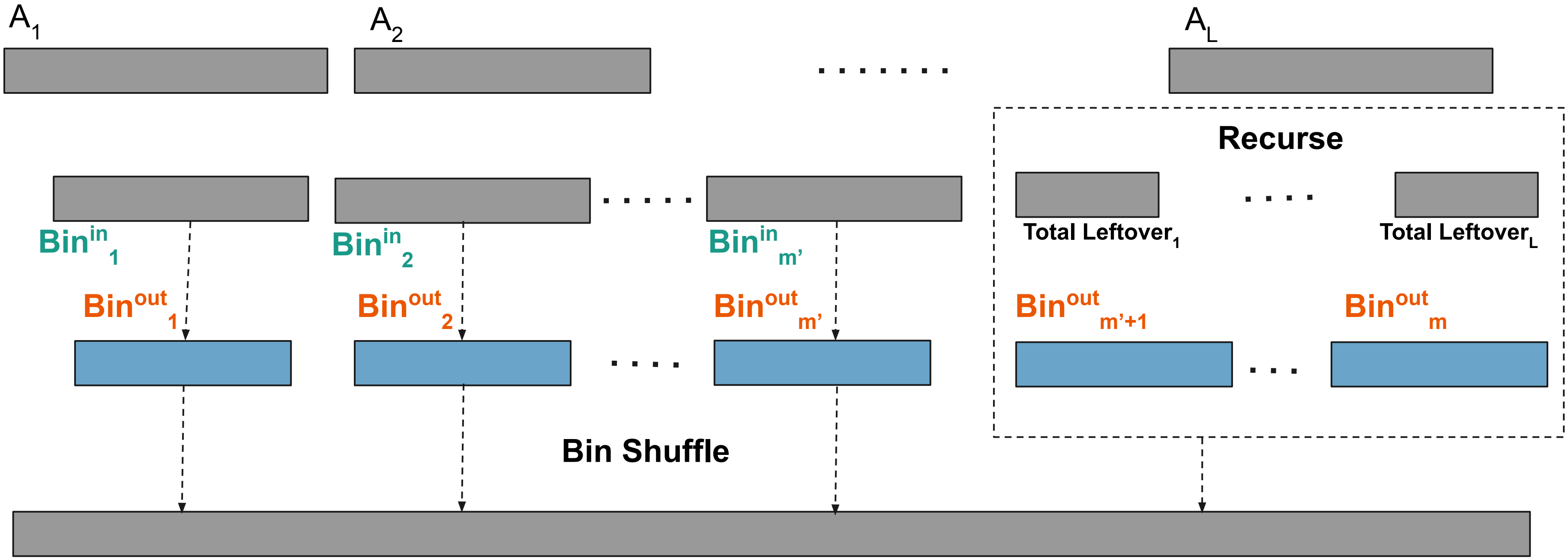


....
(b_1, Assign(b_1))
(b_2, Assign(b_2))
(b_{m-1}, Assign(b_{m-1}))
(b_m, Assign(b_m))
Assign to positions b_i in D non-obliviously
D
b_1
b_2
b_{m-1}
b_m

Oblivious Multi-Array Shuffle



Oblivious Multi-Array Shuffle



Ball and Bins Model

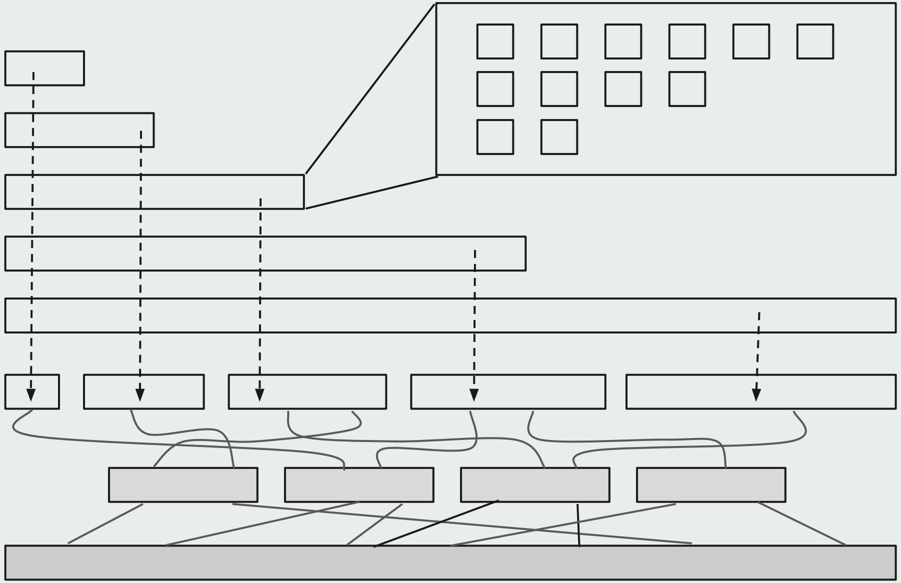
- We can instantiate the PanORAMa construction in the model where GO’96 proved O(log N) lower bound
    - Server does no computation on the data ⇒ satisfy **“balls and bins” requirement**
    - GO’96 allows client to oracle access to private random function ⇒ **replace PRF**
- PanORAMa complexity: **O(log N · log log N)**

Follow-up Work

ORAM with Logarithmic Complexity

- **OptORAMa: Optimal Oblivious RAM [AKLNPS18] (eprint 2018/892)**
  - Communication complexity: **O(log N)**
  - Oblivious compaction: **O(N)**

Overview



- **PanORAMa**: new ORAM construction with improved asymptotic complexity
  - **O(log N · log log N) for block size Ω(log N)**
- New Efficient Building Blocks
  - **Oblivious Hash Table**
  - **Oblivious Multi-Array Shuffle**

Thanks!

Questions?