Searchable Symmetric Encryption: Optimal Locality in Linear Space via Two-Dimensional Balanced Allocations

Gilad Asharov — **Cornell-Tech** (Hebrew University)

Moni Naor — Weizmann

Gil Segev — Hebrew University

Ido Shahaf — Hebrew University

Cloud Storage

- We are outsourcing more and more of our data to clouds
- We trust these clouds less and less
  - Confidentially of the data from the service provider itself
  - Protect the data from service provider security breaches

Solution: Encrypt your Data!

- But…
  - **Keyword search** is now the primary way we access our data
  - By encrypting the data - this simple operation becomes **extremely expensive**

- **How to search on encrypted data??**

Possible Solutions

- **Generic tools:** Expensive, great security
  - Functional encryption
  - Fully Homomorphic Encryption
  - Oblivious RAM*

- **More tailored solutions:** practical**,** security(?)
  - Property-preserving encryption
    (encryption schemes that supports public tests)
    - Deterministic encryption [Bellare-Boldyreva-O'Neill06]
    - Oder-preserving encryption [Agrawal-Kiernan-Srikant-Xu04]
    - Orthogonality preserving encryption [Pandey-Rouselakis04]
  - Searchable Symmetric Encryption [Song-Wagner-Perrig01]

Deterministic and Order Preserving Encryptions

Name	Lastname	Age
Elaine	Samuels	24
Mary	Stein	37
Jim	Stein	81
John	Sommers	3
Mary	Williams	17
John	Garcia	43
John	Gould	37

Name	Lastname	Age
Ge5$#u	Q*6sh#	223
E89(%y	2@#3Br	340
2Tr^#7	2@#3Br	736
qM@9*h	gYv6%t	34
E89(%y	X%3oL7	160
qM@9*h	wnM7#1	308
qM@9*h	8vy8$Z	340

"Inference Attacks against Property-Preserving Encrypted Databases"

[Naveed-Kamara-Wright. CCS2015]

Searchable Symmetric Encryption (SSE)

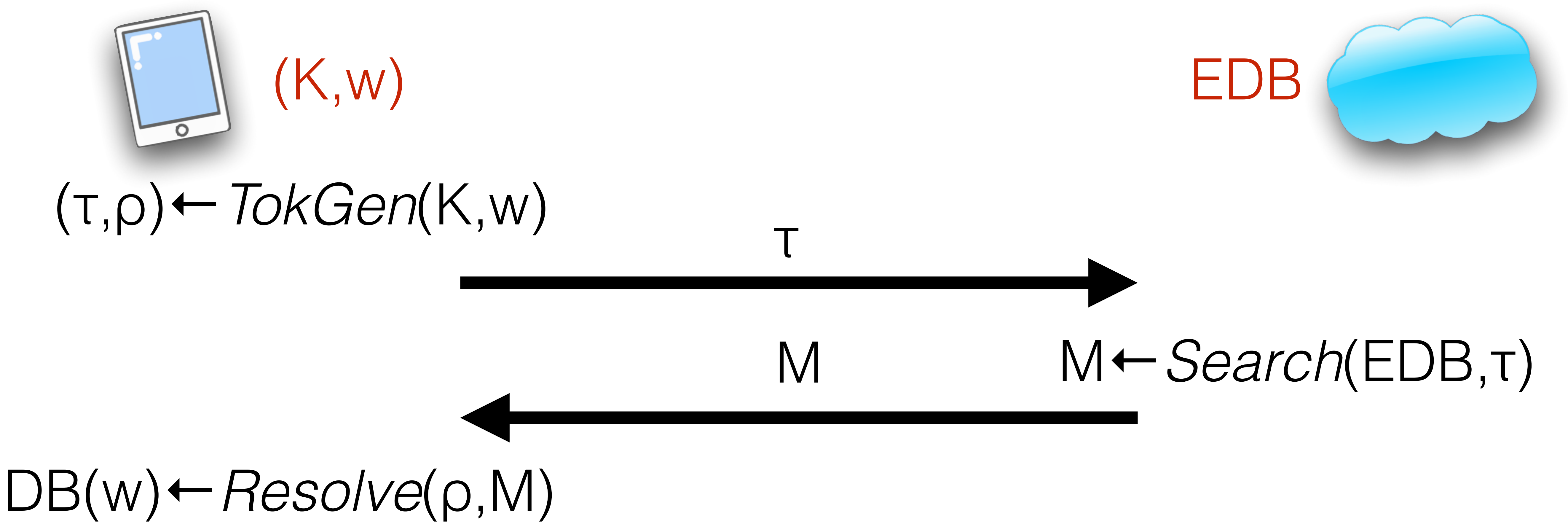
Searchable Symmetric Encryption (SSE)

- **Data:** the database **DB** consists of:
  - **Keywords:** $W=\{w_1,...,w_n\}$ (possible keywords)
  - **Documents:** $D_1,...,D_m$ (list of documents)
  - $DB(w_i)=\{id_1,...,id_{n_i}\}$
    (for every keyword w_i, list of documents / identifiers in which w_i appears)

- **Syntax of SSE:**
  - $K \leftarrow \mathit{KeyGen}(1^k)$ (generation of a private key)
  - $EDB \leftarrow \mathit{EDBSetup}(K,DB)$ (encrypting the database)
  - $(DB(wi),\lambda) \leftarrow \mathit{Search}((K,w_i),EDB)$ (interactive protocol)

The Searching Protocol

- $(DB(w),\lambda) \leftarrow \mathit{Search}((K,w),EDB)$ (interactive protocol)
- Usually - one round protocol



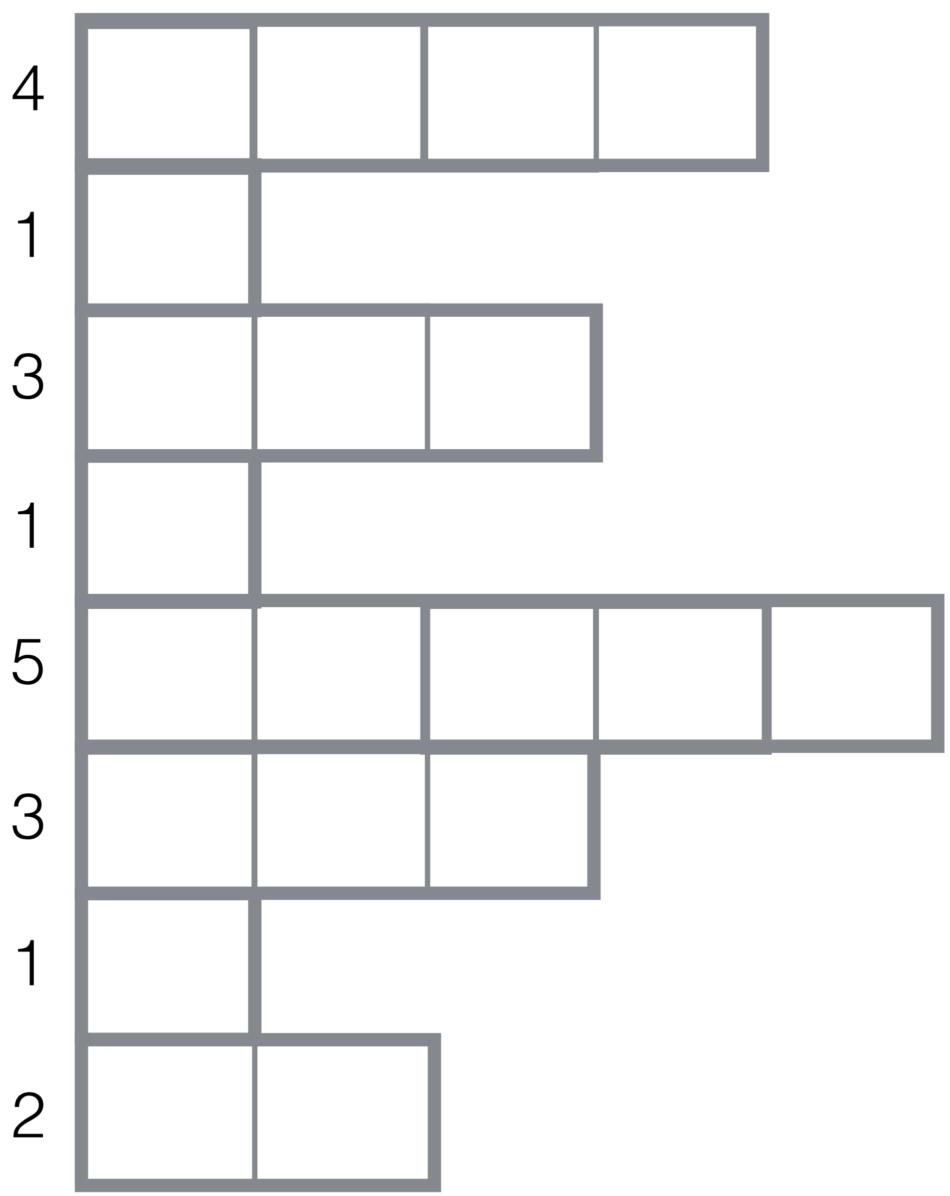
Security Requirement

- Two **equivalent** definitions:
  - Game-based definition
  - Simulation-based definition

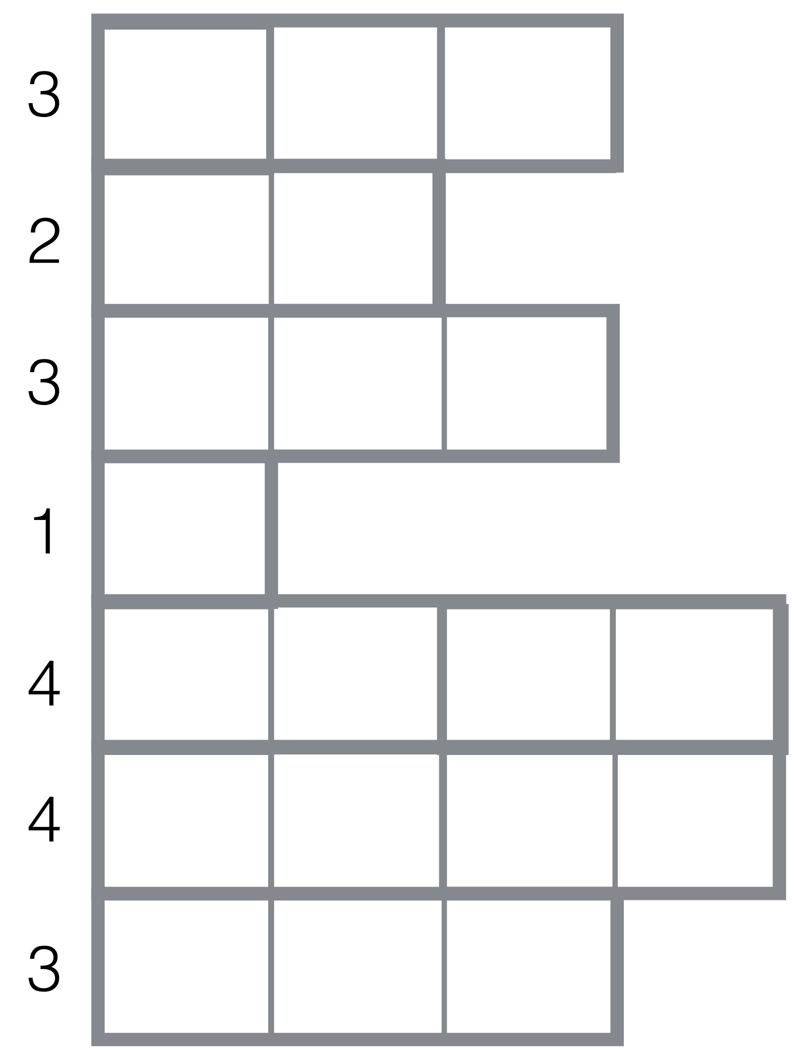
Game-Based Definition

- The adversary controls the "cloud"
- Outputs two databases DB_0,DB_1 with intersection on **w**
  (of the same size, that share some lists $\{DB(w)\}_{w \in \mathbf{w}}$ for some set of keywords **w**)
- The client receives DB_b for some randomly chosen b
- Runs: K←*KeyGen*(1^k), EDB←*EDBSetup*(K,DB) and τ_i=*TokGen(k,w)* for all $w \in \mathbf{w}$
- The adversary receives: (EDB, $\{\tau_w\}_{w \in \mathbf{w}}$), guesses b

Game-Based Definition

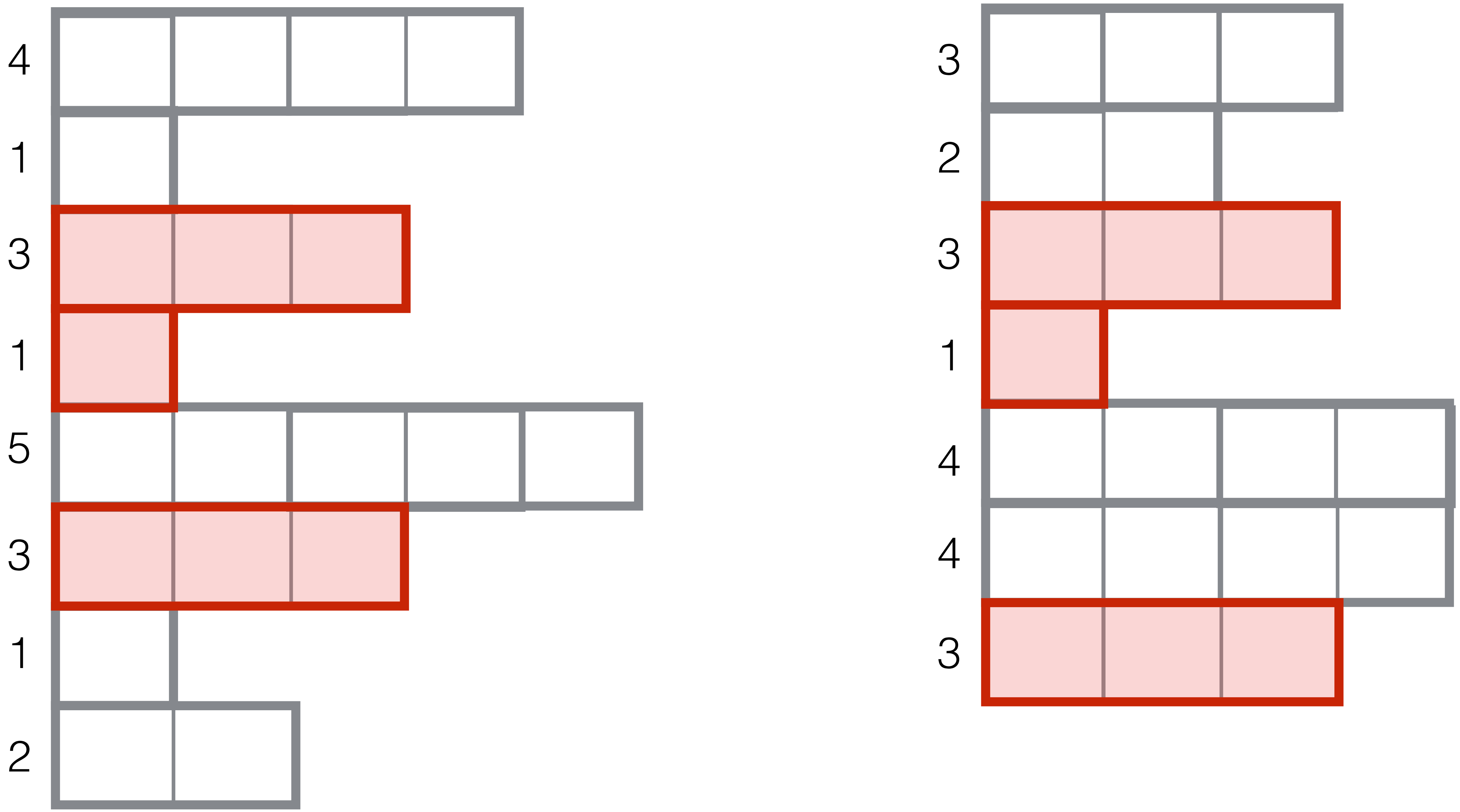


DB_0



DB_1

Game-Based Definition



Need to hide the “structure” of the lists

DB_0 **DB_1**

Simulation Based Security

- The adversary outputs (DB, **w**)
  - **REAL world:**
    - The experiment runs *KeyGen*, *EDBSetup*, and *TokGen* for every $w \in \mathbf{w}$
    - **EDB** (the resulting encrypted DB), $\{\tau_w\}_{w \in \mathbf{w}}$ (the resulting tokens)
  - **IDEAL world:**
    - The simulator receives $\mathcal{L}(\mathrm{DB},\mathbf{w})$ (some leakage on the **queried keywords only**)
    - Outputs **EDB** (the resulting encrypted DB), $\{\tau_w\}_{w \in \mathbf{w}}$ (the resulting tokens)
- The adversary receives **EDB**, $\{\tau_w\}_{w \in \mathbf{w}}$, output REAL/IDEAL

Security

- **Good news:** Semantic security for data; no deterministic or order preserving encryption
- Leakage in the form of access patterns to retrieved data and queries
    - Data is encrypted but server can see intersections b/w query results
      (e.g. identify popular document)
- Additional specific leakage:
    - E.g. we leak |DB(w1)|
    - E.g. the server learns if two documents have the same keyword
- Leads to statistical inference based on side information on data (effect depends on application)

EDBSetup

Keyword	Records
Searchable	5,14
Symmetric	5,14,22,45,67
Encryption	1,2,3,4,5,6,7,8,9,10
Schemes	22,14

inverted index

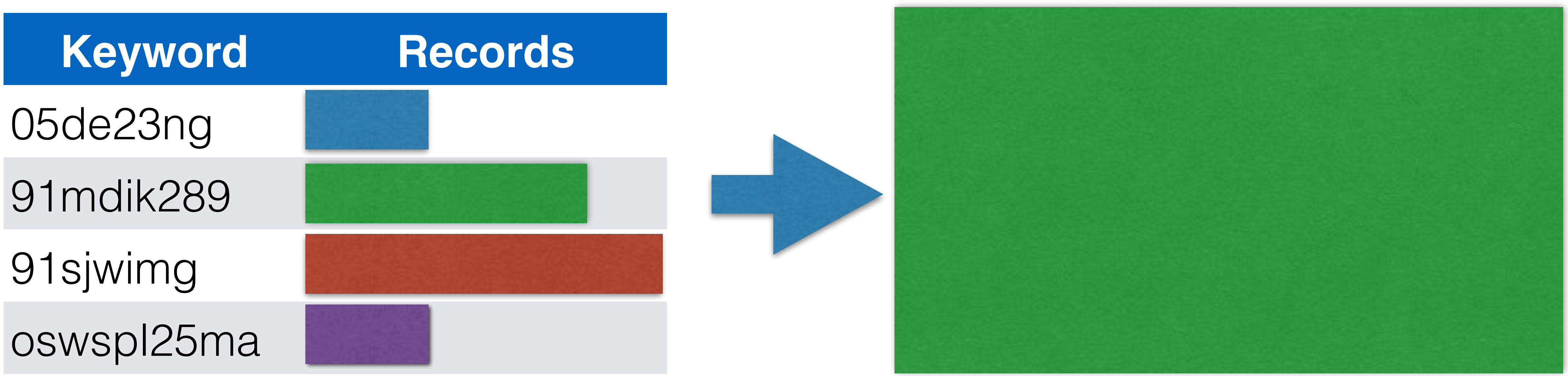
Replace each keyword w with some $PRF_K(w)$

Keyword	Records
05de23ng	5,14
91mdik289	5,14,22,45,67
91sjwimg	1,2,3,4,5,6,7,8,9,10
oswspl25ma	22,14

encrypted index

Keyword	Records
05de23ng	
91mdik289	
91sjwimg	
oswspl25ma	

The Challenge…



No leakage on the structure of the lists!

How to map the lists into memory?

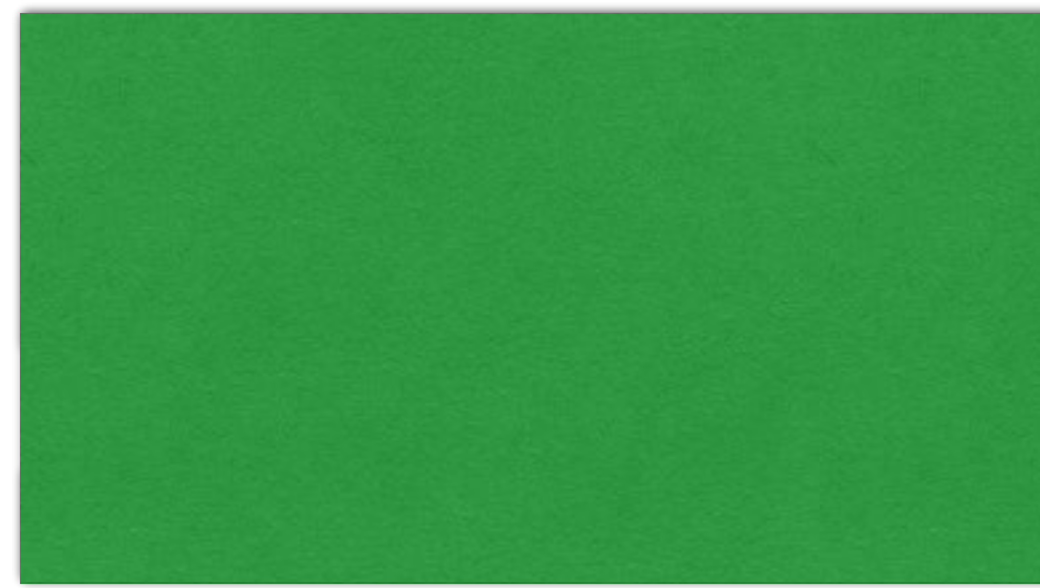
Functionality - Search

(Allow some Leakage…)

(K,w)

Search for keyword:
Encryption

PRF_K(Encryption) →



Security Requirement:
The server should not learn anything
about the structure of lists that were not queried

Mapping Lists into Memory

Maybe shuffle the lists?

Keyword	Records
05de23ng	
91mdik289	
91sjwimg	
oswspl25ma	

Hiding the Structure of the Lists

Maybe shuffle the lists?

Previous Constructions: Maximal Padding [CK10]

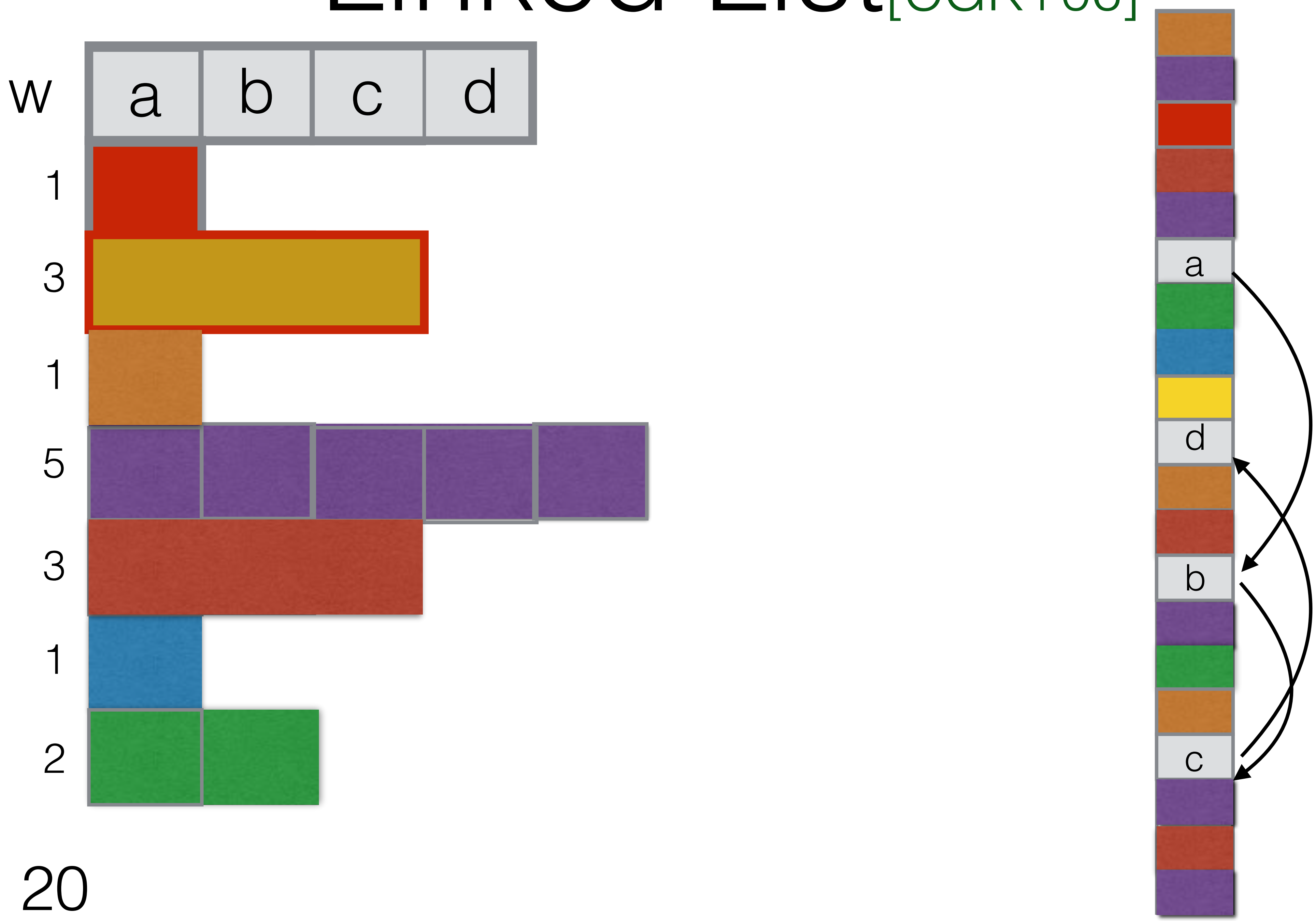
Keyword	Records
05de23ng	
91mdik289	
91sjwimg	
oswspl25ma	

Keyword	Records
05de23ng	
91mdik289	
91sjwimg	
oswspl25ma	

1) Pad each list to maximal size (N?)
2) Store lists in random order
3) Pad with extra lists to hide the number of lists

Size of encrypted DB: $O(N^2)$

Previous Constructions
Linked List[CGK+06]

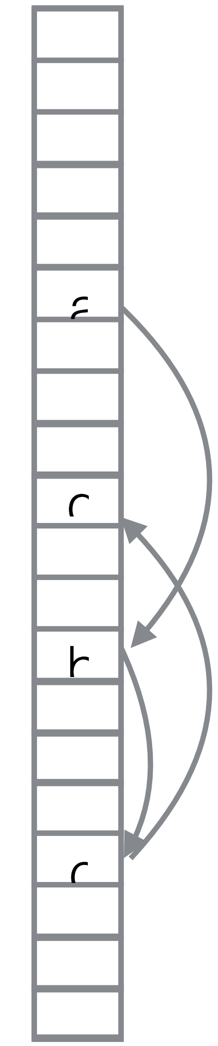


Efficiency Measures [CT14]

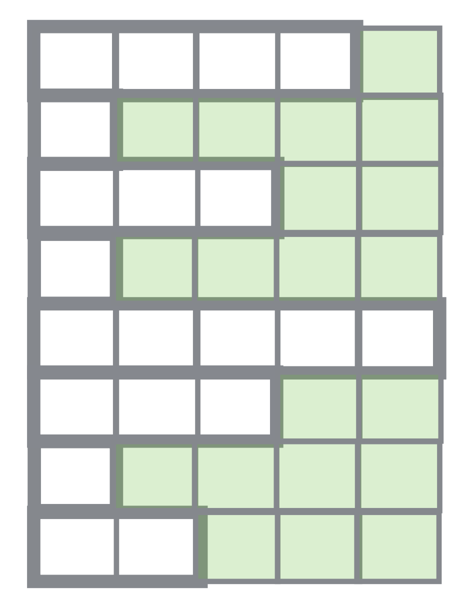
- A variant was implemented in [CJJ+13]
  - Poor performance due to… locality!

- ***Space***: The overall size of the encrypted database (Want: O(N))
- ***Locality***: number of non-continuous memory locations the server accesses with each query (Want: O(1))
- ***Read efficiency***: The ratio between the number of bits the server reads with each query, and the actual size of the answer (Want: O(1))

Efficiency



- Scheme I:
  - **Space**: O(N)
  - **Locality**: O(N)
  - **Read efficiency**: O(1)
- Scheme II:
  - **Space**: O(N^2)
  - **Locality**: O(1)
  - **Read efficiency**: O(1)



SSE and Locality [CT14]

Can we construct an SSE scheme that is optimal in space, locality and read efficiency?

NO!*

- **Lower bound:** any scheme must be sub-optimal in either its ***space*** overhead, ***locality*** or ***read efficiency***
- Impossible to construct scheme with O(N) ***space***, O(1) ***locality*** and O(1) ***read efficiency***

Why **NO***?

Theorem 1.1 *If Π is an $\mathcal{L}$-IND-secure SSE scheme with locality r as well as α-overlapping reads, then Π has $\omega\left(\frac{|\mathsf{BinEnc(DB)}|}{r\cdot(\alpha+1)}\right)$ server storage.*

- Instead of **read efficiency** the theorem captures "**α-overlapping reads**"
- Intuitively, any two reads intersect in at most **α** bits
  - Captures all previous constructions
  - Large α - "waste"
- **Intuition for lower bound:**
  - Reads do not intersect much (**α**-overlapping reads)
  - Any list can be placed only in few positions (locality)
  - We must pad the lists in order to hide their sizes…

SSE and Locality [CT14]

Our Goal:

Constructing a scheme that is nearly optimal?

- Maybe even completely optimal if we do not assume **α-overlapping** reads? (though, it seems counter-intuitive)
  - How do schemes with “large” α look like?

Related Work

- A single keyword search
  - Related work [SWP00,Goh03,CGKO06,ChaKam10]
- Beyond single keyword search
  - Conjunctions, range queries, general boolean expression, wildcards [CashJJKRS13,JareckiJKRS13,CashJJJKRS14,FaberJKNRS15]
  - Schemes that are not based on inverted index [PappasKVKMCGKB14, FischVKKKMB15]
- **Locality** in searchable symmetric encryption [CashTessaro14]
- Dynamic searchable symmetric encryption [….]
- Leakage-abuse attacks [CashGrubbsPerryRistenpart15]

Our Work

Our Results

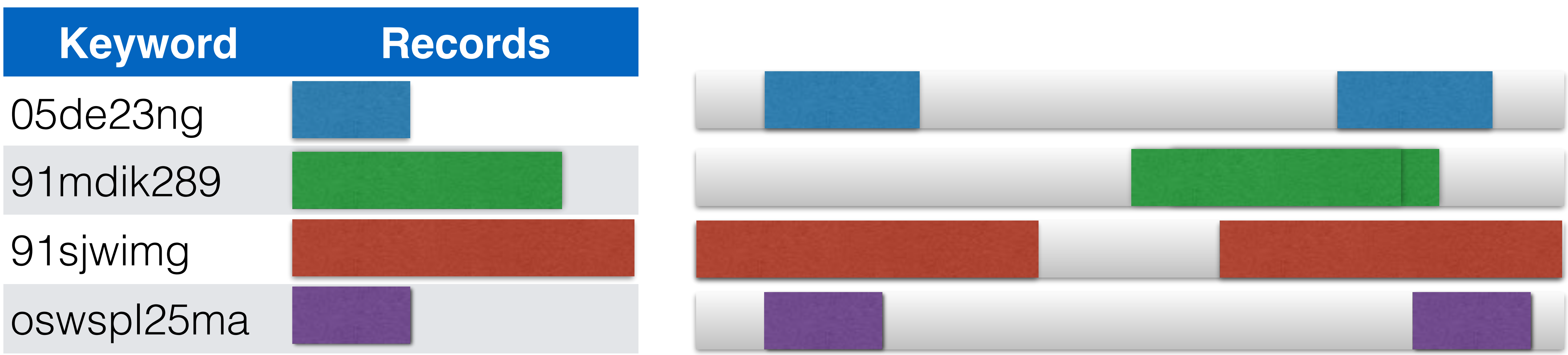
Scheme	Space	Locality	Read Efficiency
[CGK+06,KPR12,CJJ+13]	O(N)	$O(n_w)$	O(1)
[CK10]	$O(N^2)$	O(1)	O(1)
[CT14]	O(NlogN)	O(logN)	O(1)
This work I	O(N)	O(1)	Õ(logN)
This work II*	O(N)	O(1)	Õ(loglogN)
This work III	O(NlogN)	O(1)	O(1)

Õ(f(N))=O(f(n) log f(n))

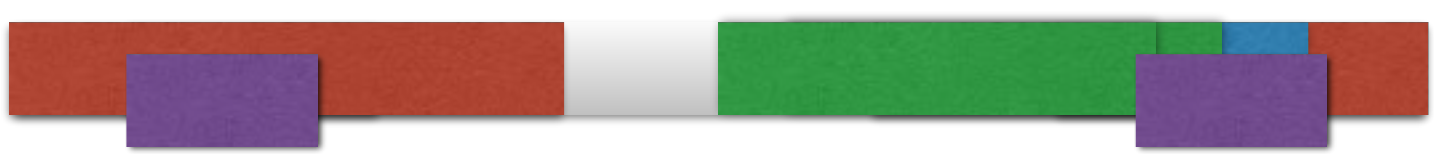
*assumes no keyword appears in more than $N^{1-1/\log\log N}$ documents

Our Schemes

1) Choose for each list "possible ranges" independently



2) Place the elements of each list in its possible ranges



(is it possible?)

Allocation Algorithms

- We show a general transformation:
  - Allocation algorithm ⇒ secure SSE scheme
  - If the allocation algorithm is "efficient" then the SSE is ``efficient'' (successfully places the lists even though each has few possible "small" possible ranges)
- **Security intuition**:
  The ***possible*** locations of each list are completely independent to the ***possible*** locations of the other lists
  - (But many correlations in the **actual** placement)
- With each query, the server reads ***all*** possible ranges of the list
  - We never reveal the decisions made for the actual placement
- **How to construct efficient Allocation algorithms?**

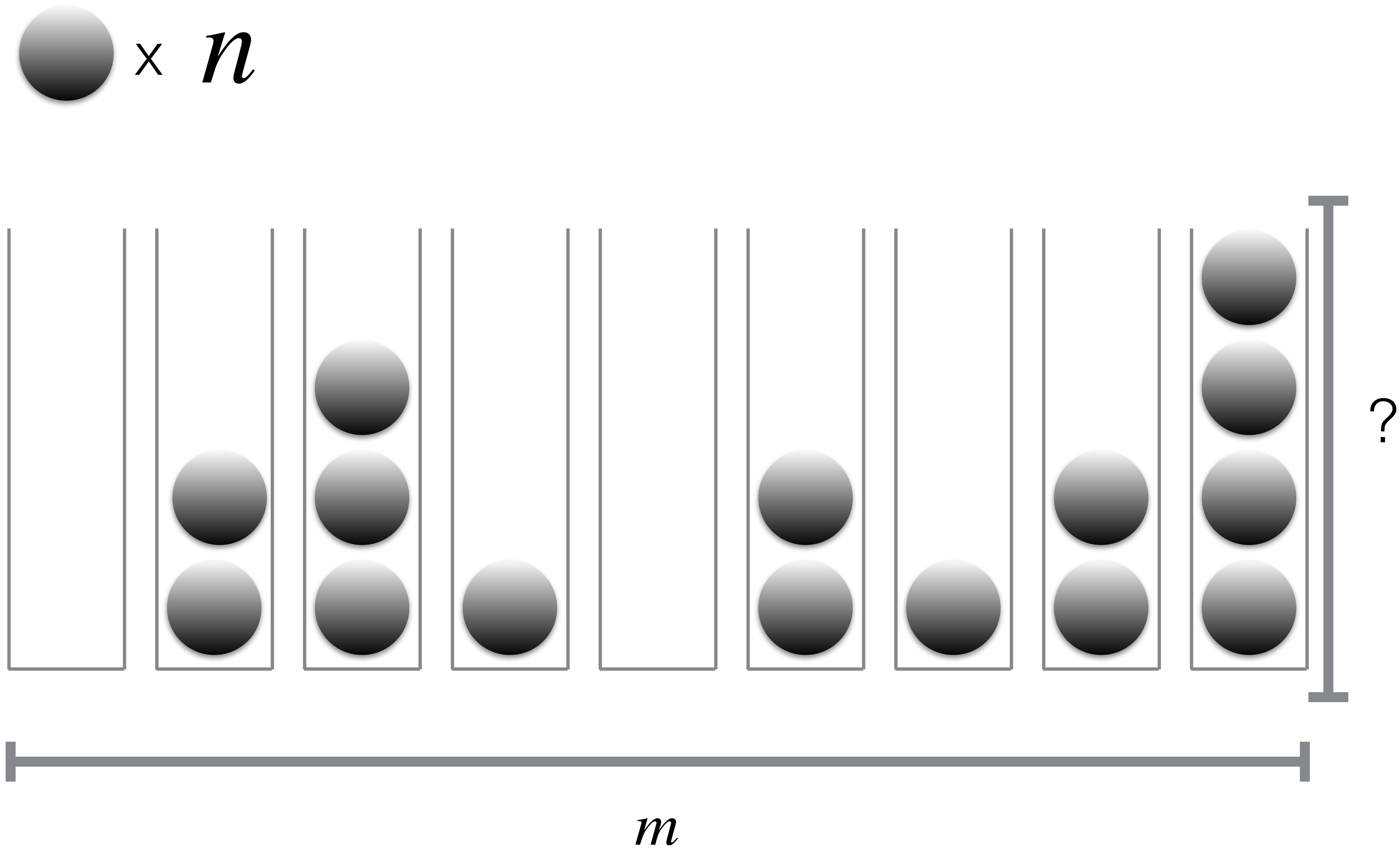
Our Approach

- We put forward a **two-dimensional** generalization of the classic balanced allocation problem (“balls and bins”), considering **lists of various lengths** instead of “balls” (=lists of fixed length)

(1) We construct efficient *2D* balanced allocation schemes

(2) Then, we use cryptographic techniques to transform any such scheme into an SSE scheme

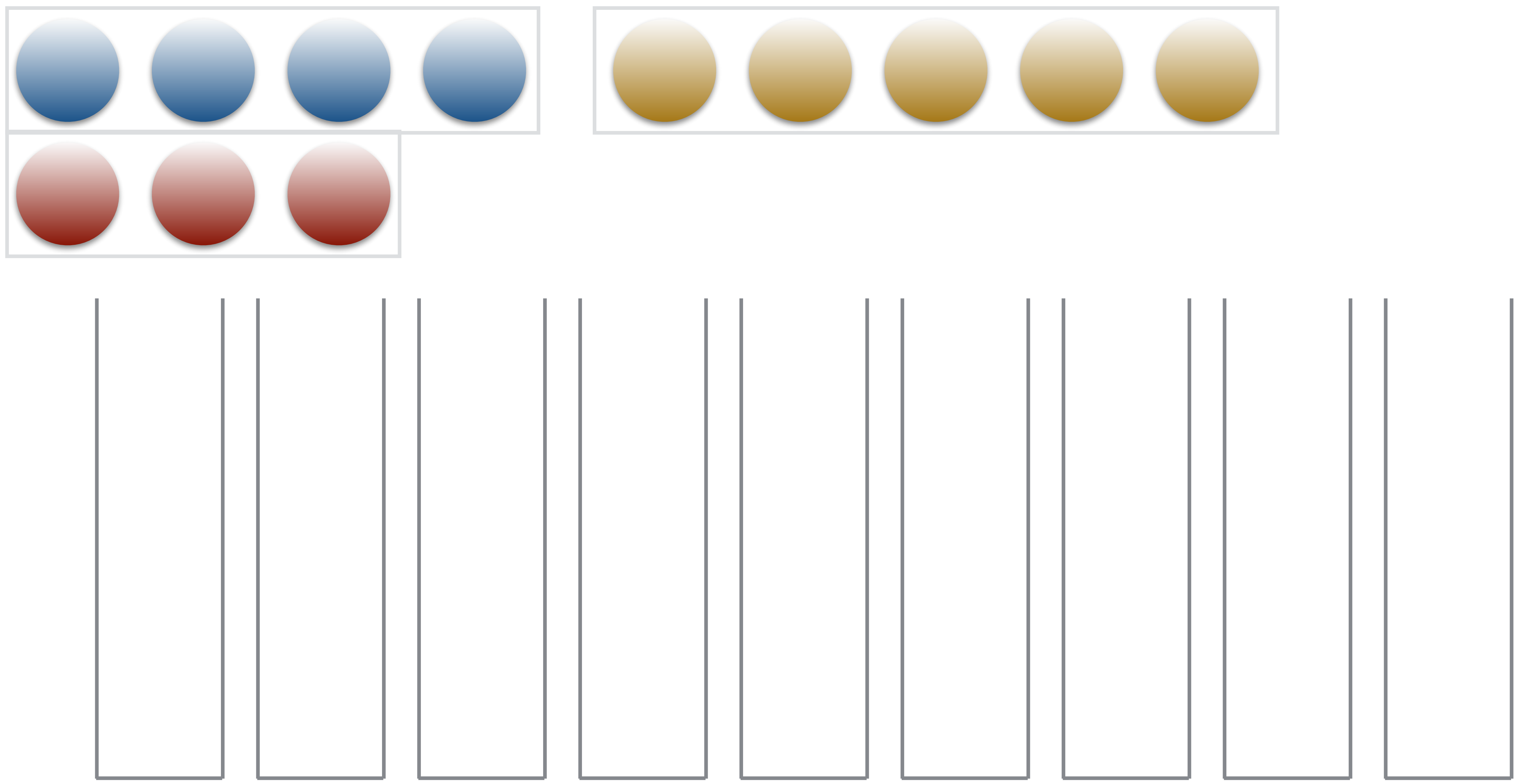
Balls and Bins



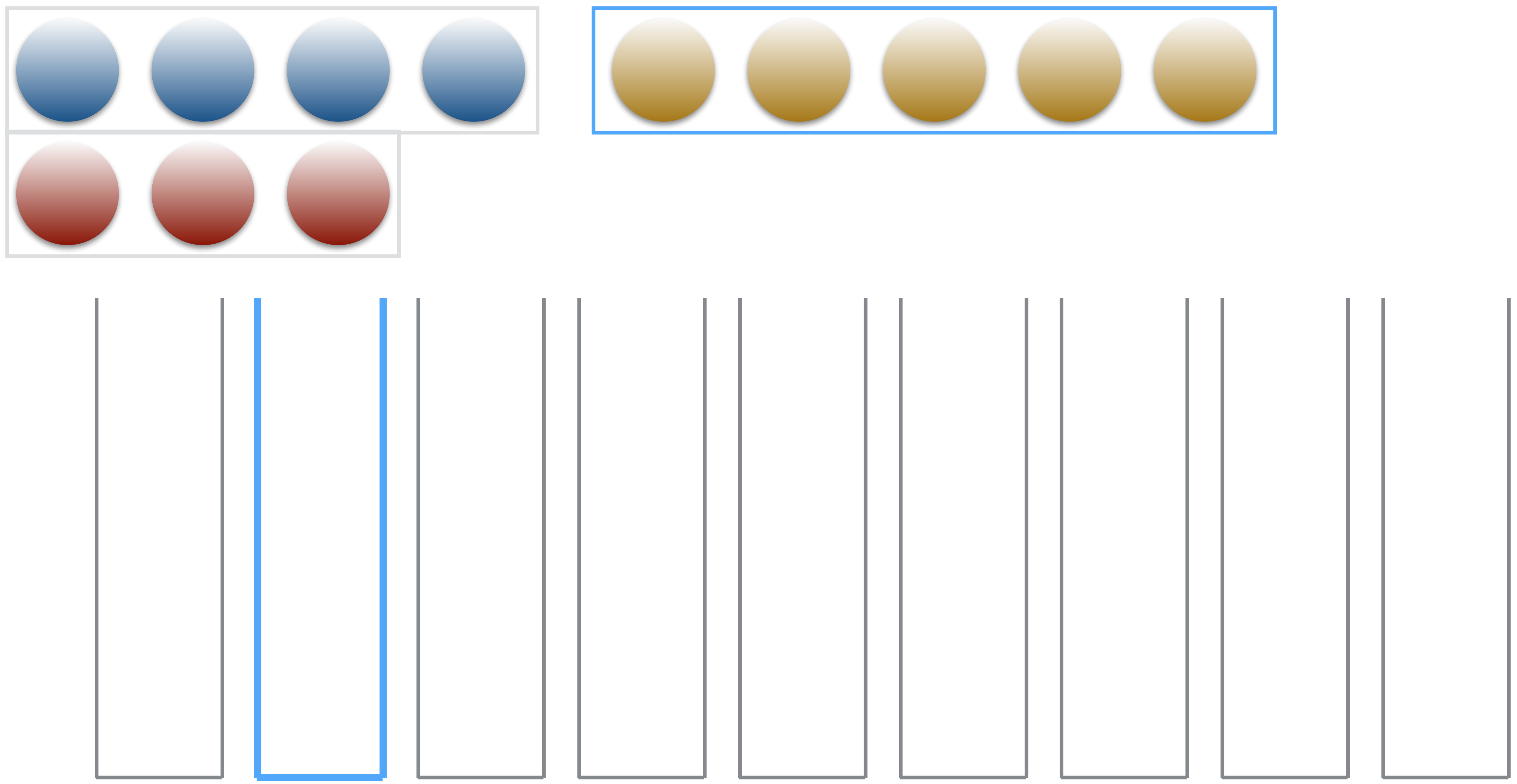
Balls and Bins (Random Allocation)

- n balls, m bins
  - Choose for each ball one bin uniformly at random
  - **m=n:** with high probability - there is no bin with more than $\frac{\log n}{\log \log n} \cdot (1 + o(1))$
  - **m=n/log n:** with overwhelming probability, there is no bin with load greater than Õ(log n)

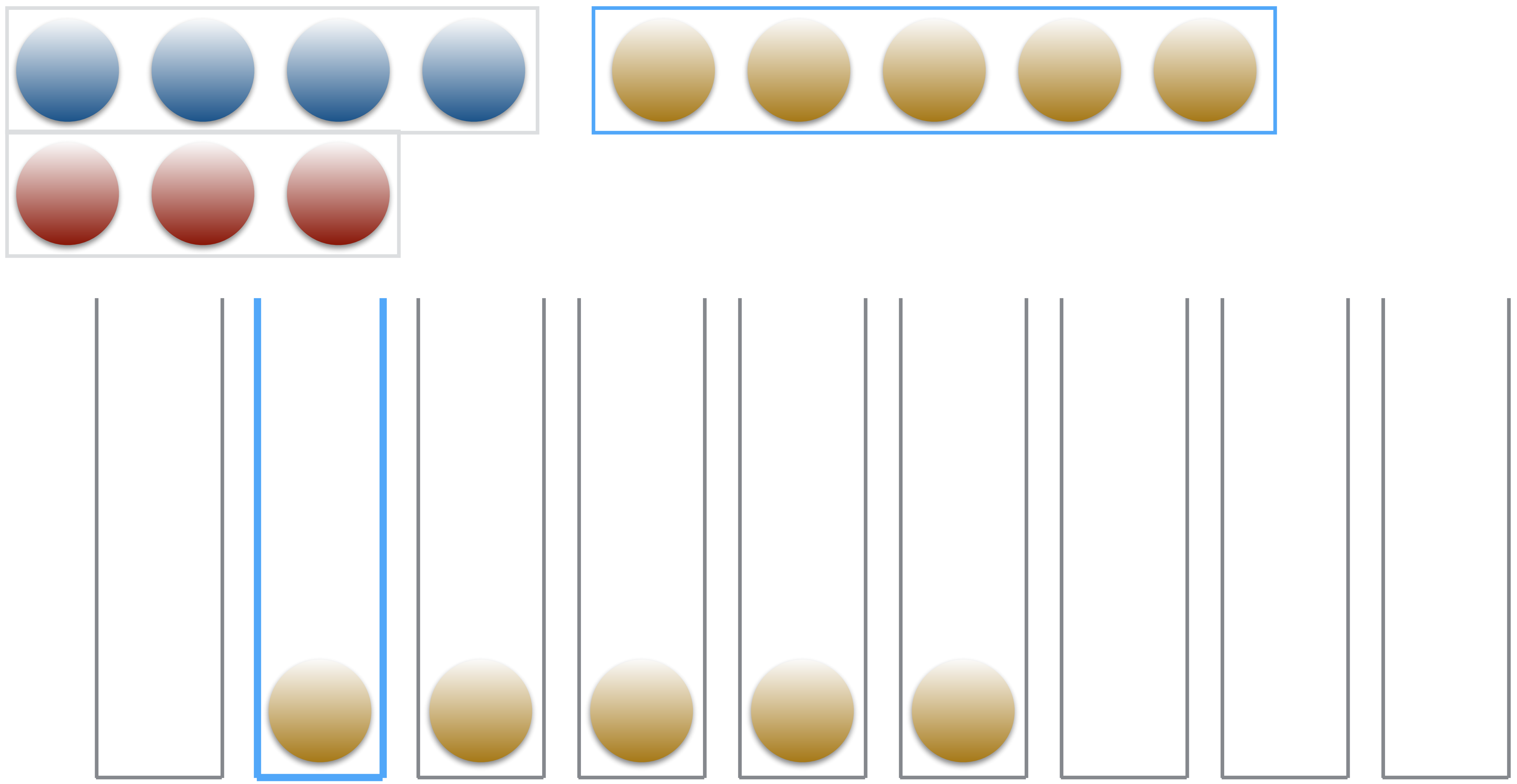
Two-Dimensional Allocation



Two-Dimensional Allocation

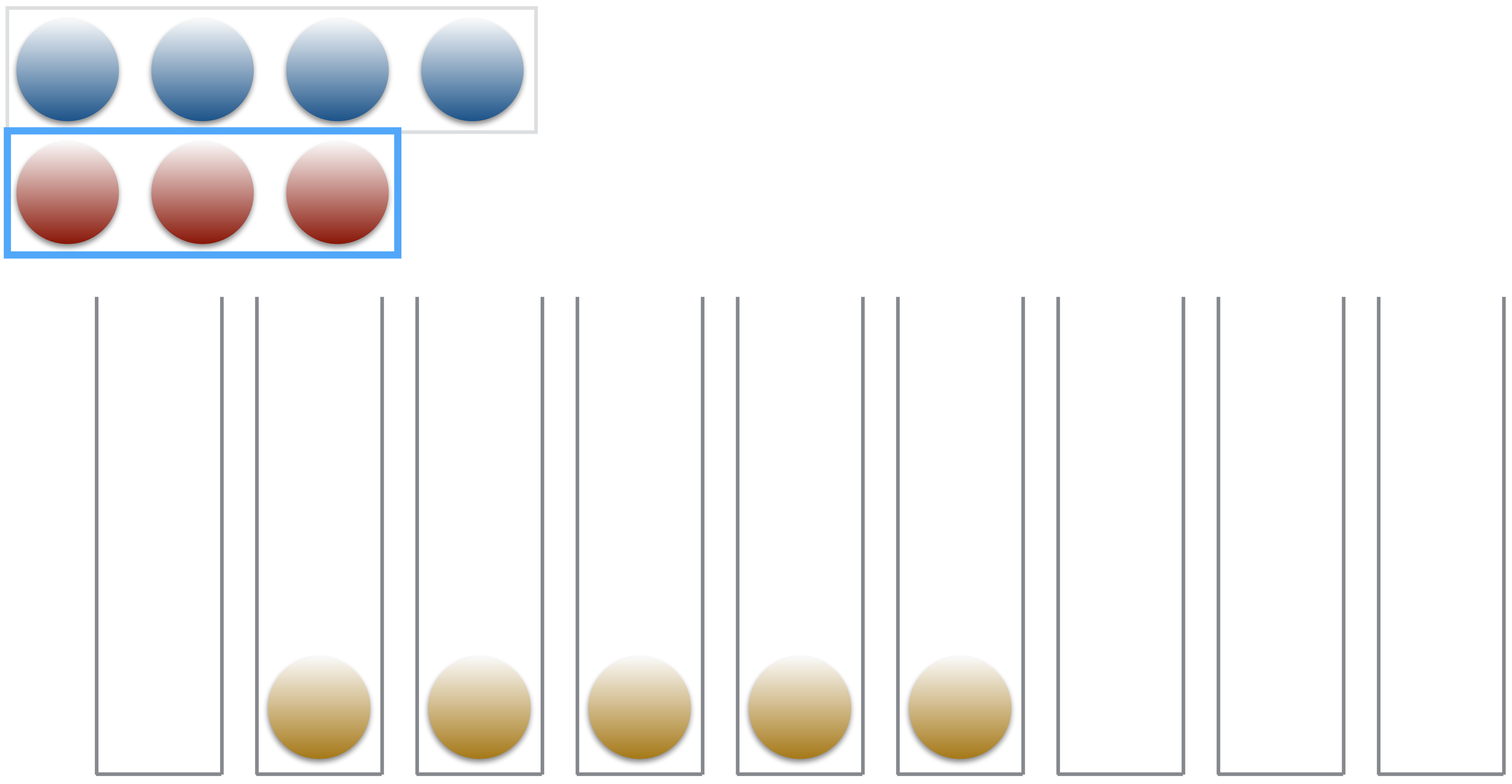


Two-Dimensional Allocation

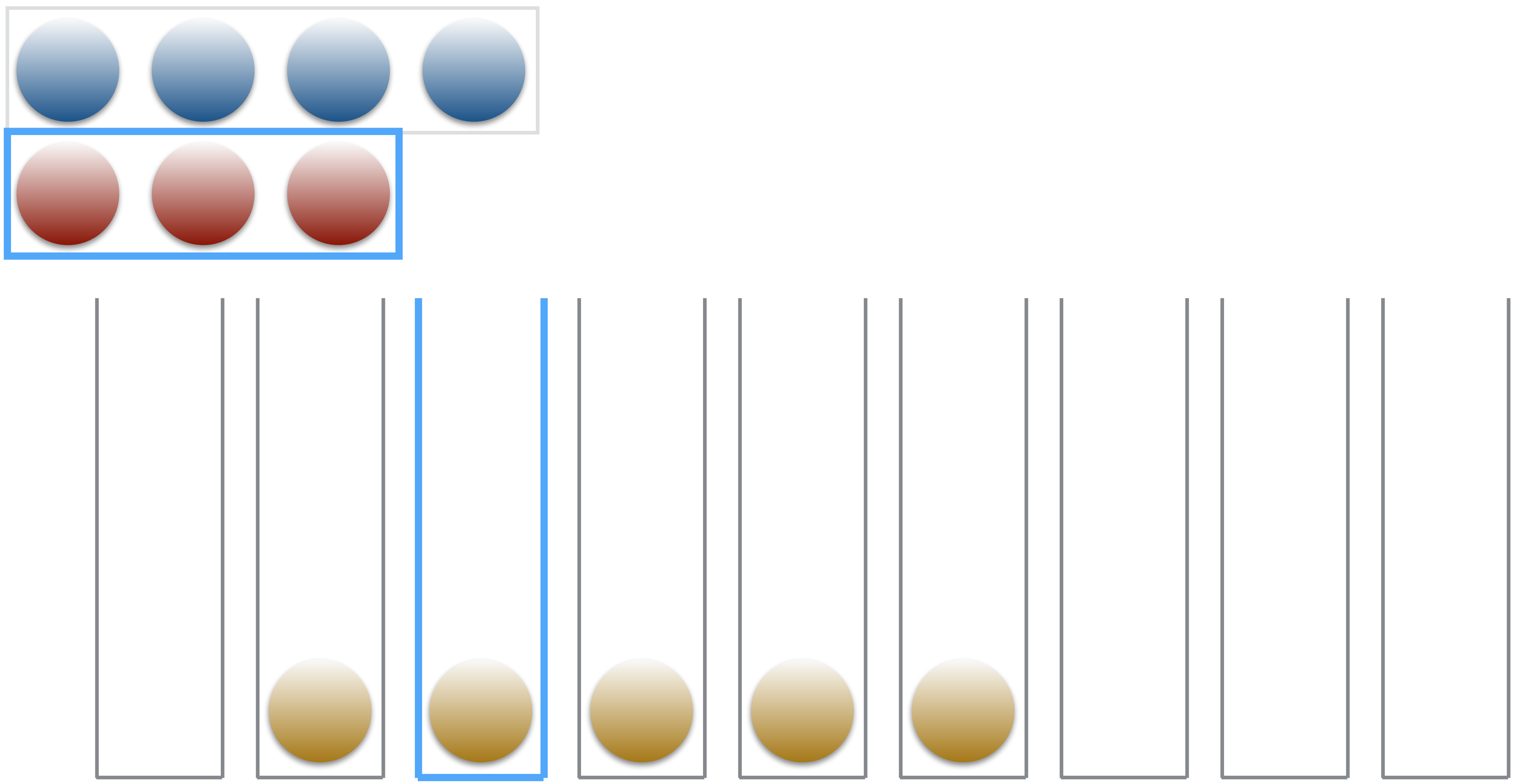


Place the whole list according to
a ***single*** probabilistic choice!

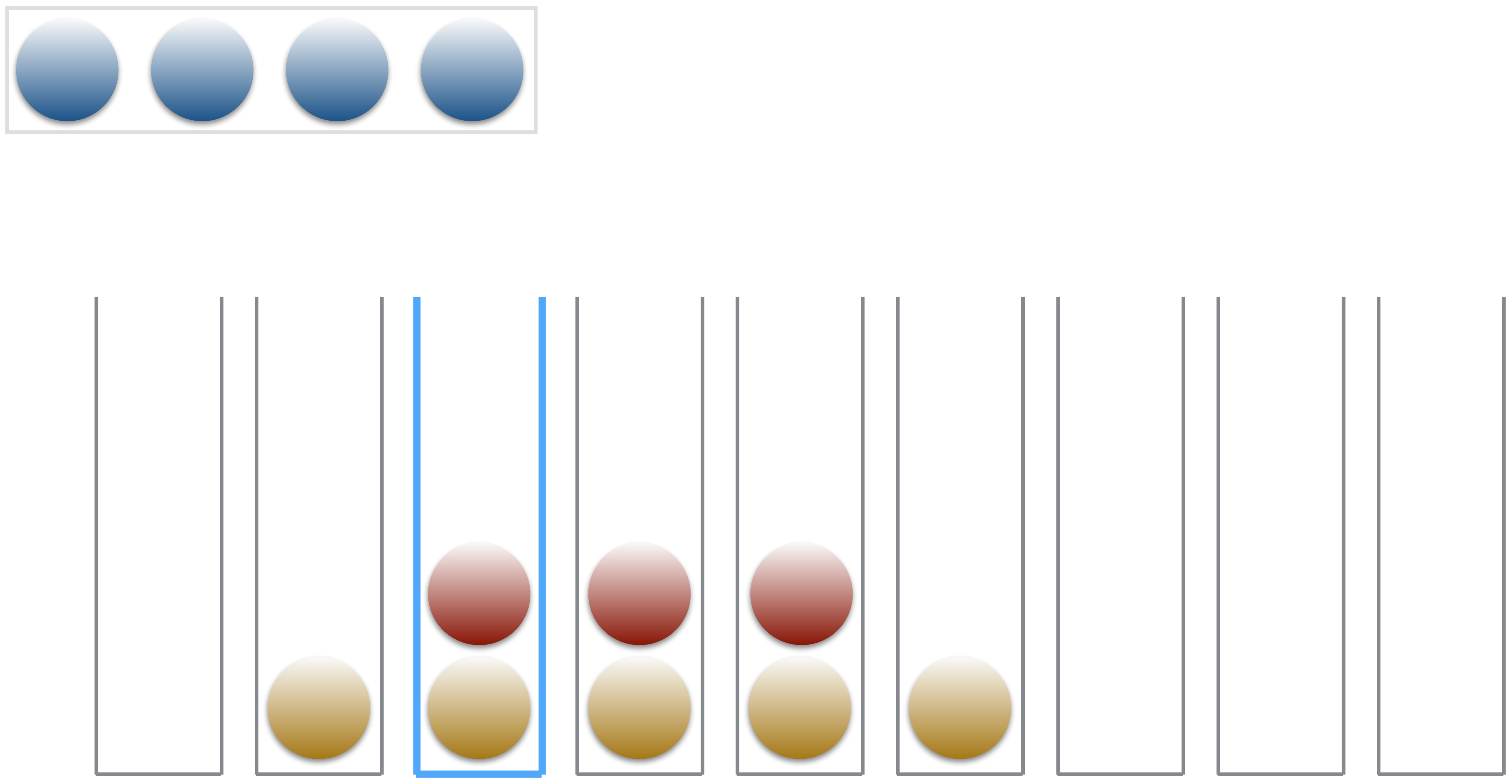
Two-Dimensional Allocation



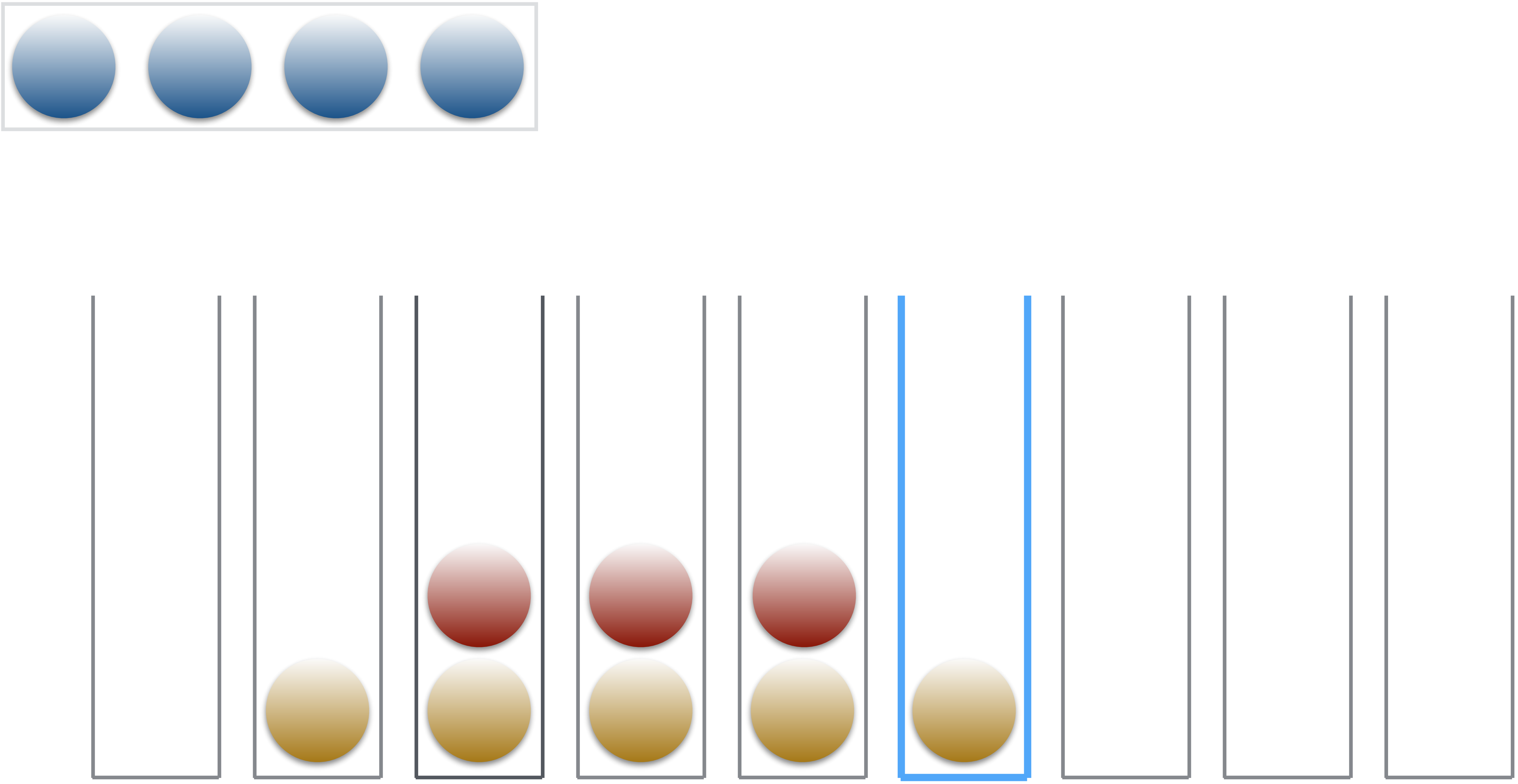
Two-Dimensional Allocation



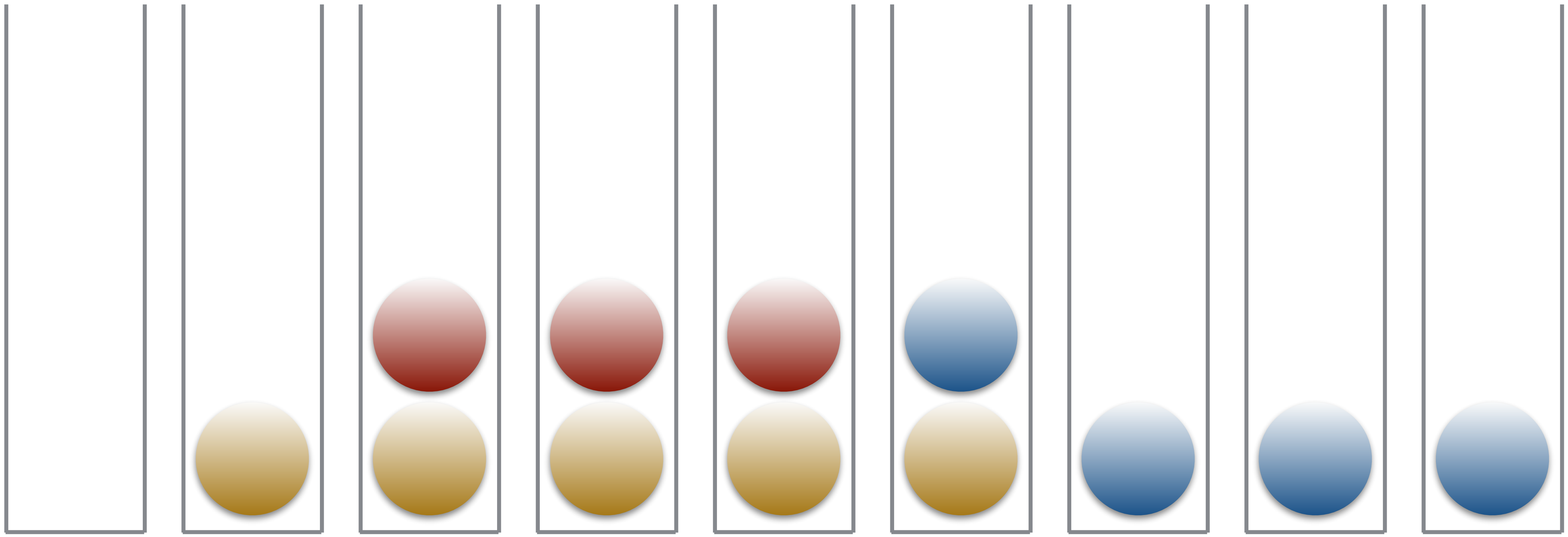
Two-Dimensional Allocation



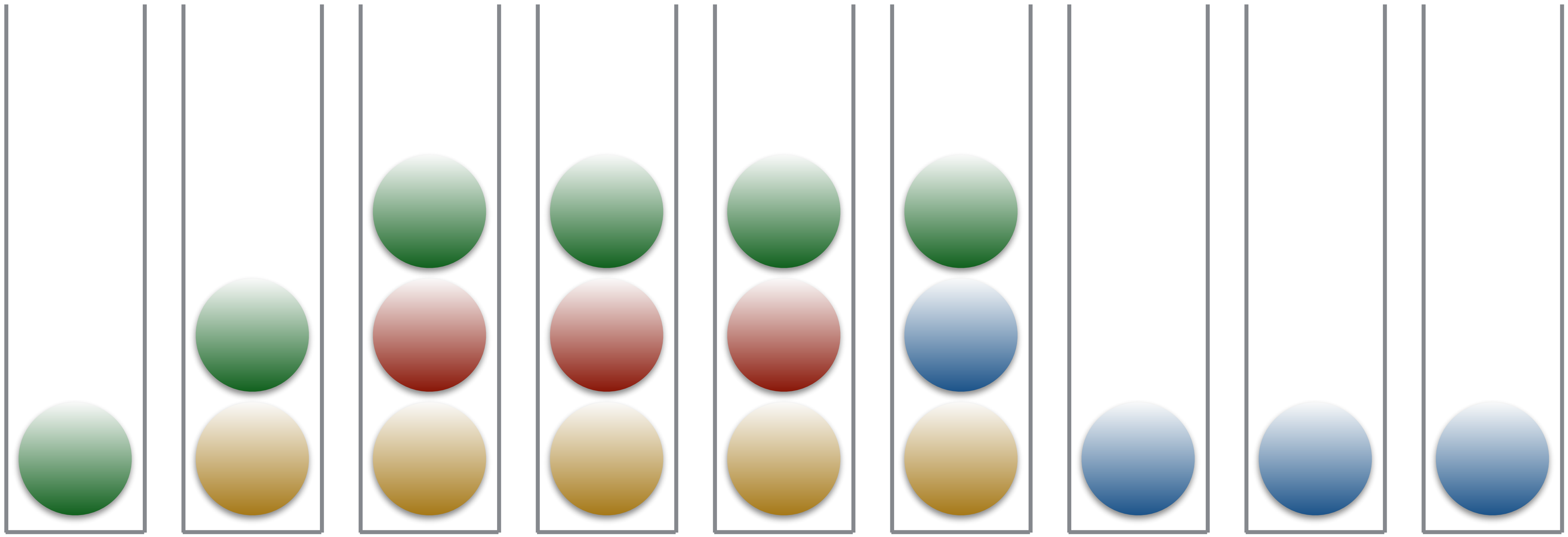
Two-Dimensional Allocation



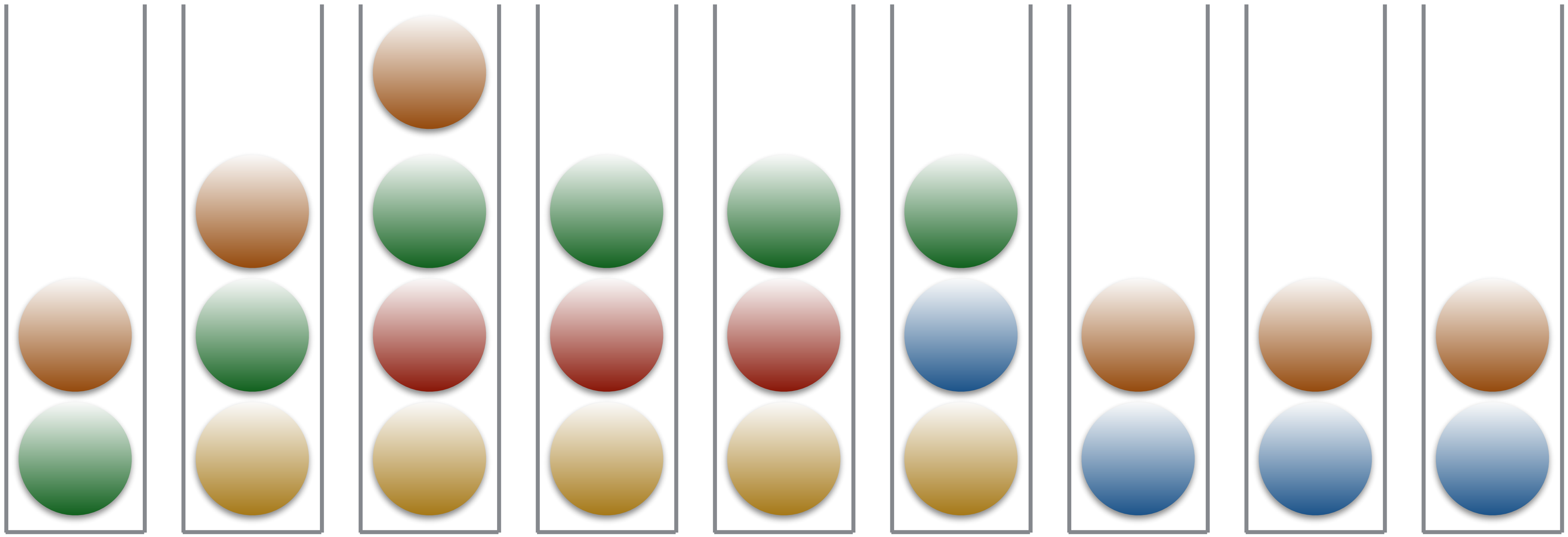
Two-Dimensional Allocation



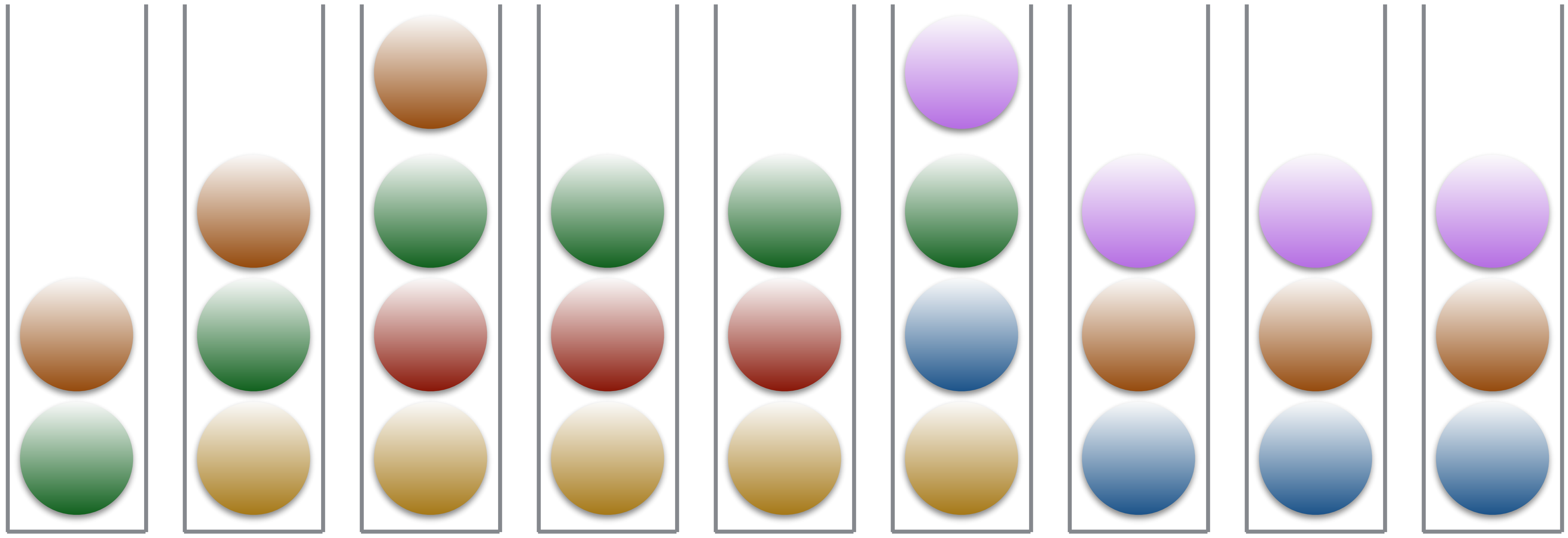
Two-Dimensional Allocation



Two-Dimensional Allocation

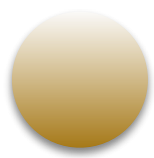


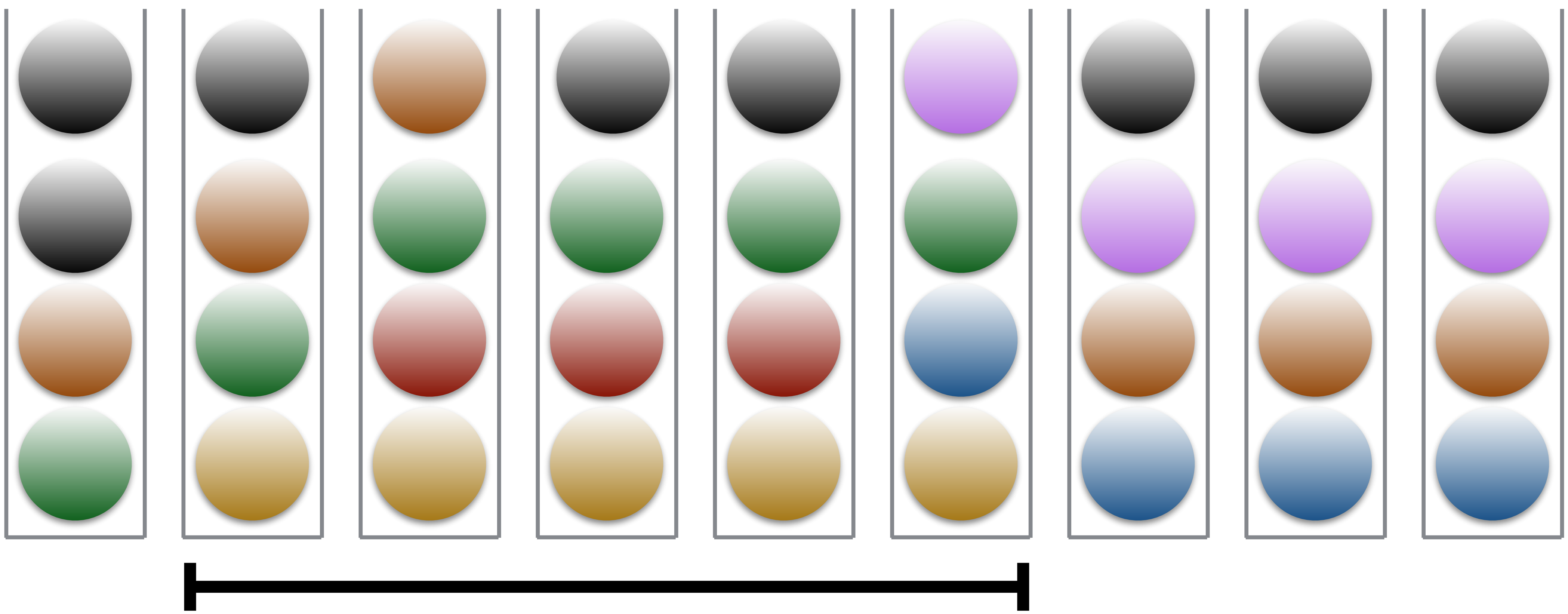
Two-Dimensional Allocation



What is the maximal load?

How Do We Search?

Search()



Our First Scheme: 2D Random Allocation

- **Theorem:** Set **#Bins=N/O(logN loglogN)**. Then, with an overwhelming probability, the maximal load is **3logN loglogN**

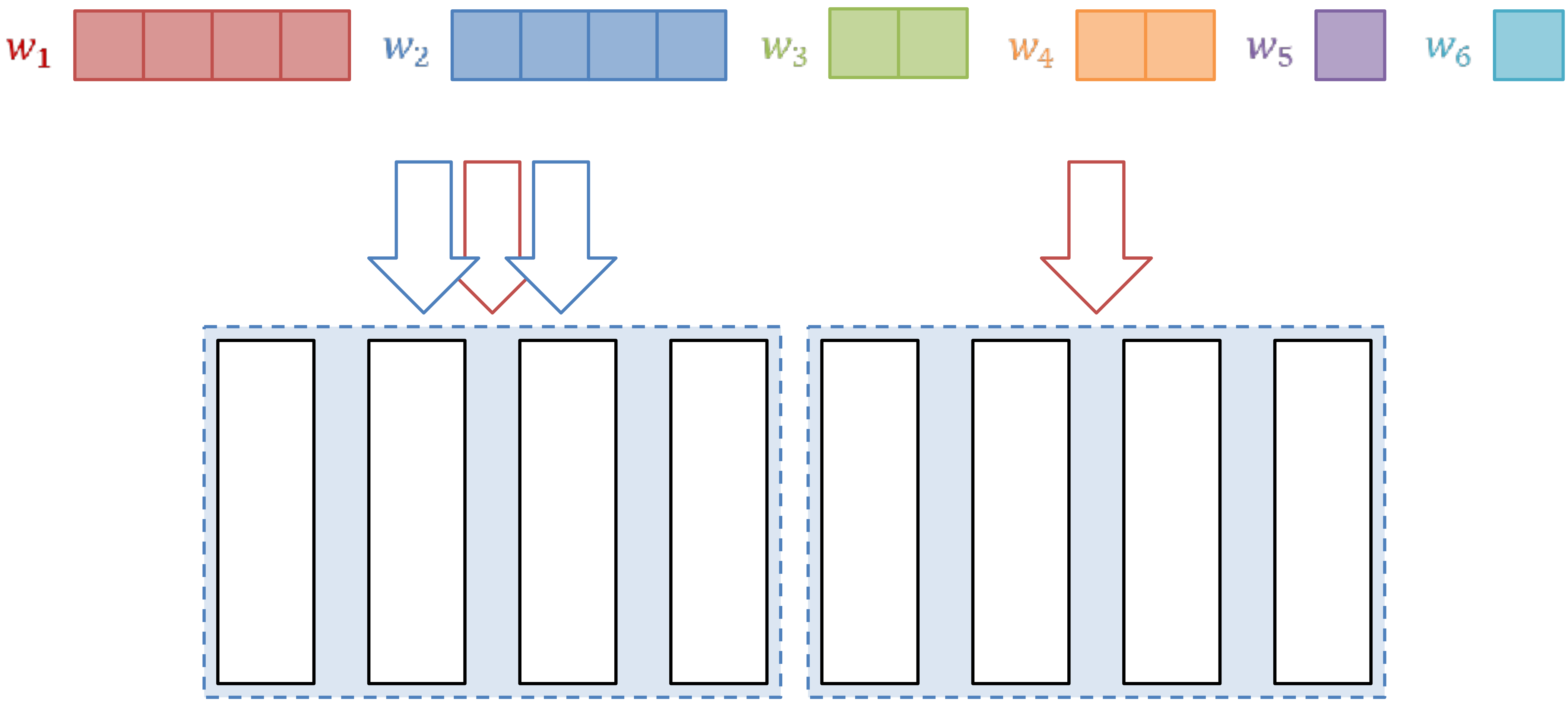
- **Main Challenge** (compared to 1D case):
  Heavy dependencies between the elements of the same list

- **This yields an SSE scheme with:**
  - Space: **#Bins x BinSize = O(N)**
  - Locality: **O(1)**
  - Read efficiency: **Õ(log n)**

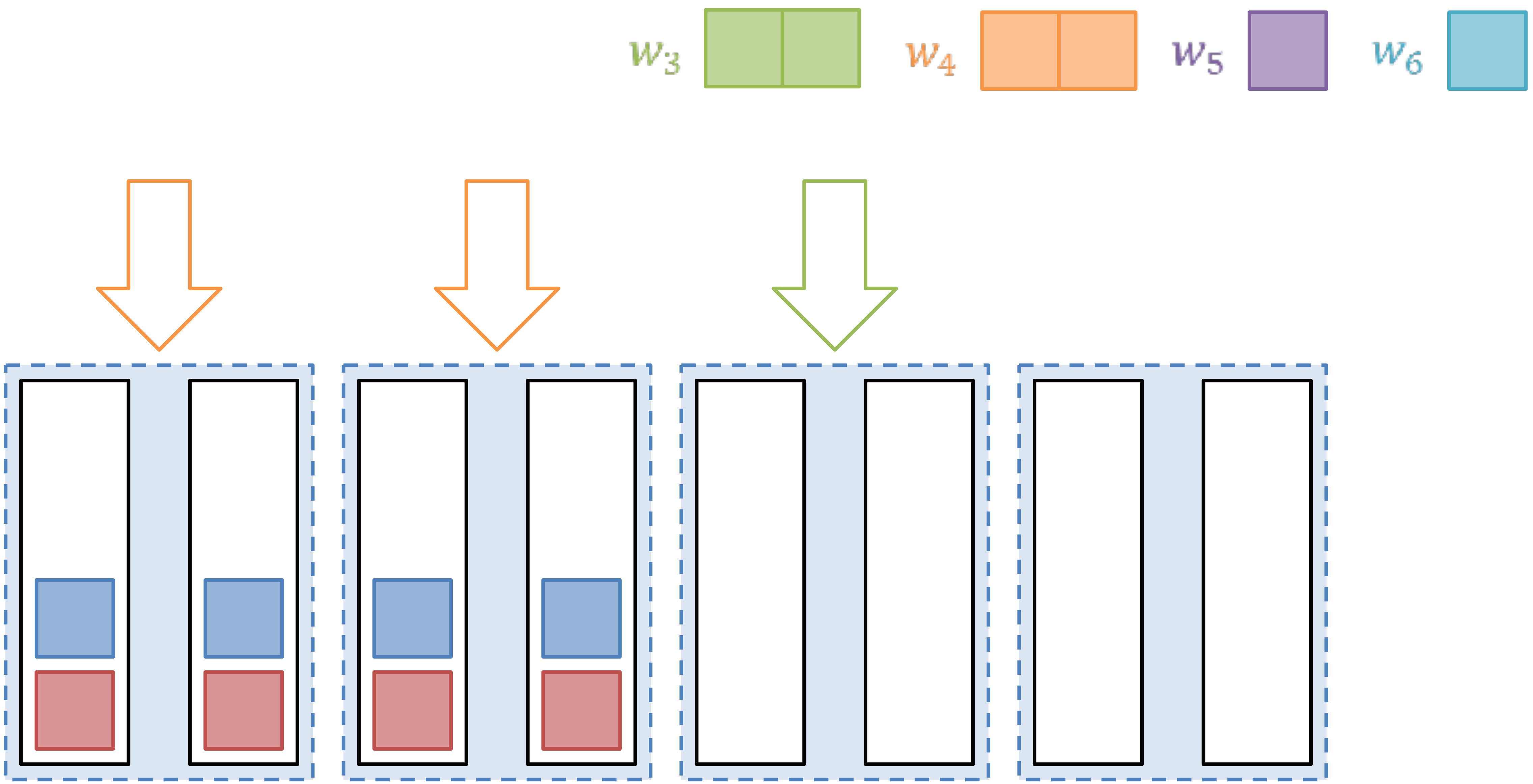
The Power of Two Choices

- In the classic “balls and bins” [ABKU99]:
  - If we choose **one** random bin for each ball, then the maximal load is O(log N/ loglogN)
  - If we choose **two** random bins for each ball, and place the ball in the least loaded one, then the maximal load is O(loglogN)
    - Exponential improvement!
- Can we adapt the two-choice paradigm to the 2D case?

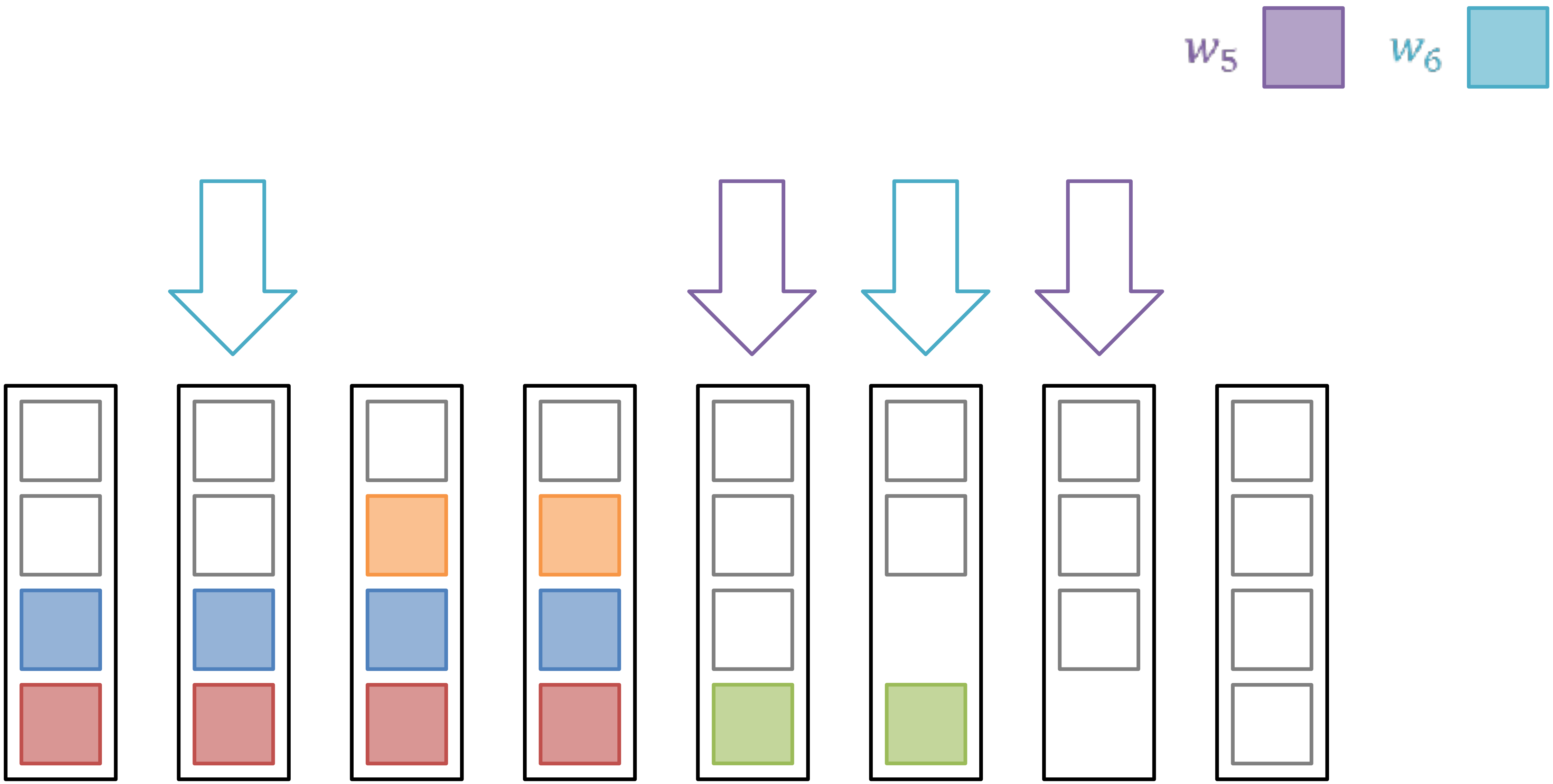
2D Two-Choice Allocation



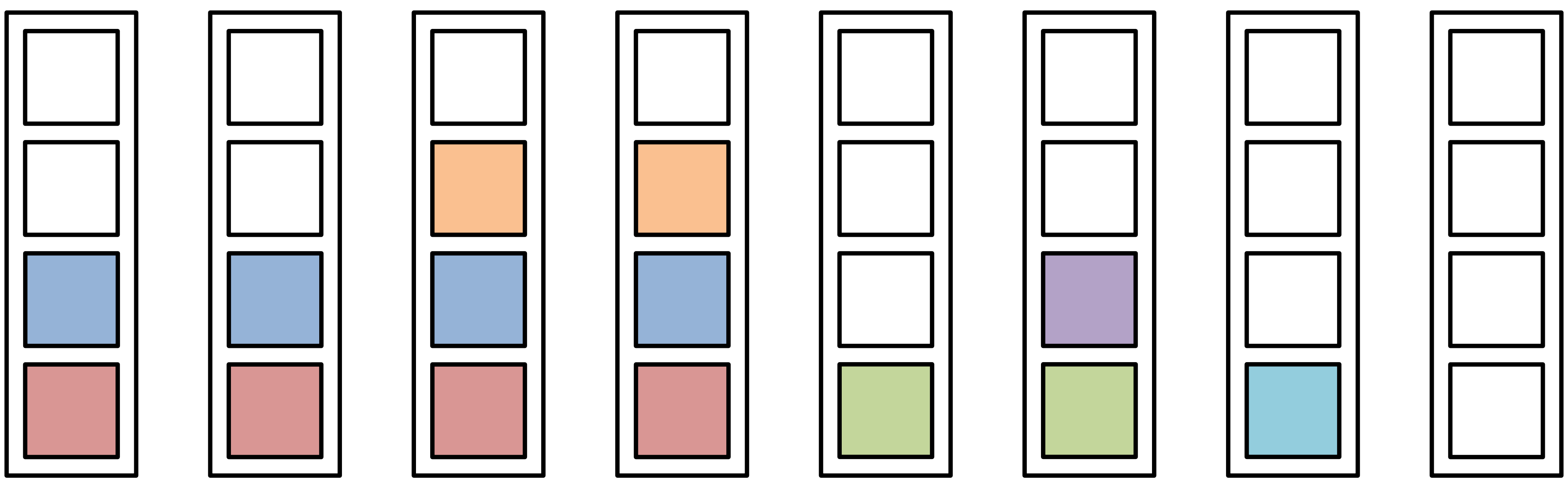
2D Two-Choice Allocation



2D Two-Choice Allocation



2D Two-Choice Allocation



2D Two-Choice Allocation

Theorem: *Assume all lists are of length at most* $N^{1-1/\log\log N}$, and set **#Bins=N/(loglogN (logloglogN)²)**.
Then, with an overwhelming probability, the maximal load is **O(loglogN (logloglogN)²)**

- **Main Challenge:** (compared to 1D case):
  - Manny challenges…

- This yields an SSE scheme with:
  - Space: **#Bins x BinSize = O(N)**
  - Read efficiency: **2BinSize = Õ(loglogN)**
  - Locality: **O(1)**

On the Assumption

- We assume that no keyword appears in more than $n^{1-1/\log\log n}$ documents
  - Keywords with too many occurrences are not indexed by search engines
- **Tightness:**
  - Assume that there are $n^{1/\log\log n}$ lists of size $n^{1-1/\log\log n}$
  - The probability that they all share the same super-bin is noticeable
    - Cannot be placed even using more sophisticated algorithms
  - We generalize this intuition to capture **all** allocation algorithms

Summary

- Novel generalization of classical data structure problem
  - And use it to build a crypto system!
  - The construction seems practical (small constants)

- First constructions of SSE with no bound on the overlapping reads
  - First constructions with **linear** encrypted database size and "good" locality
  - Still, we see limitations of allocation problems (On the size of the maximal list)

- Extending [CT14] lower bound?

Summary

- **Our approach**: SSE via two-dimensional balanced allocations

Scheme	Space	Locality	Read Efficiency
This work I	O(N)	O(1)	Õ(logN)
This work II*	O(N)	O(1)	Õ(loglogN)
This work III	O(NlogN)	O(1)	O(1)

Nice combination between DS and Cryptography

Thank You!